Consensus with Byzantine failures and asynchrony: the Ben-Or's algorithm, revisited

Distributed Algorithms

Recap

Positive Result: *There exists* a deterministic synchronous protocol that solves consensus, while tolerating crash failures

FLP Theorem: **No deterministic** protocol can solve consensus, while tolerating **1 crash** and **asynchrony**

Today

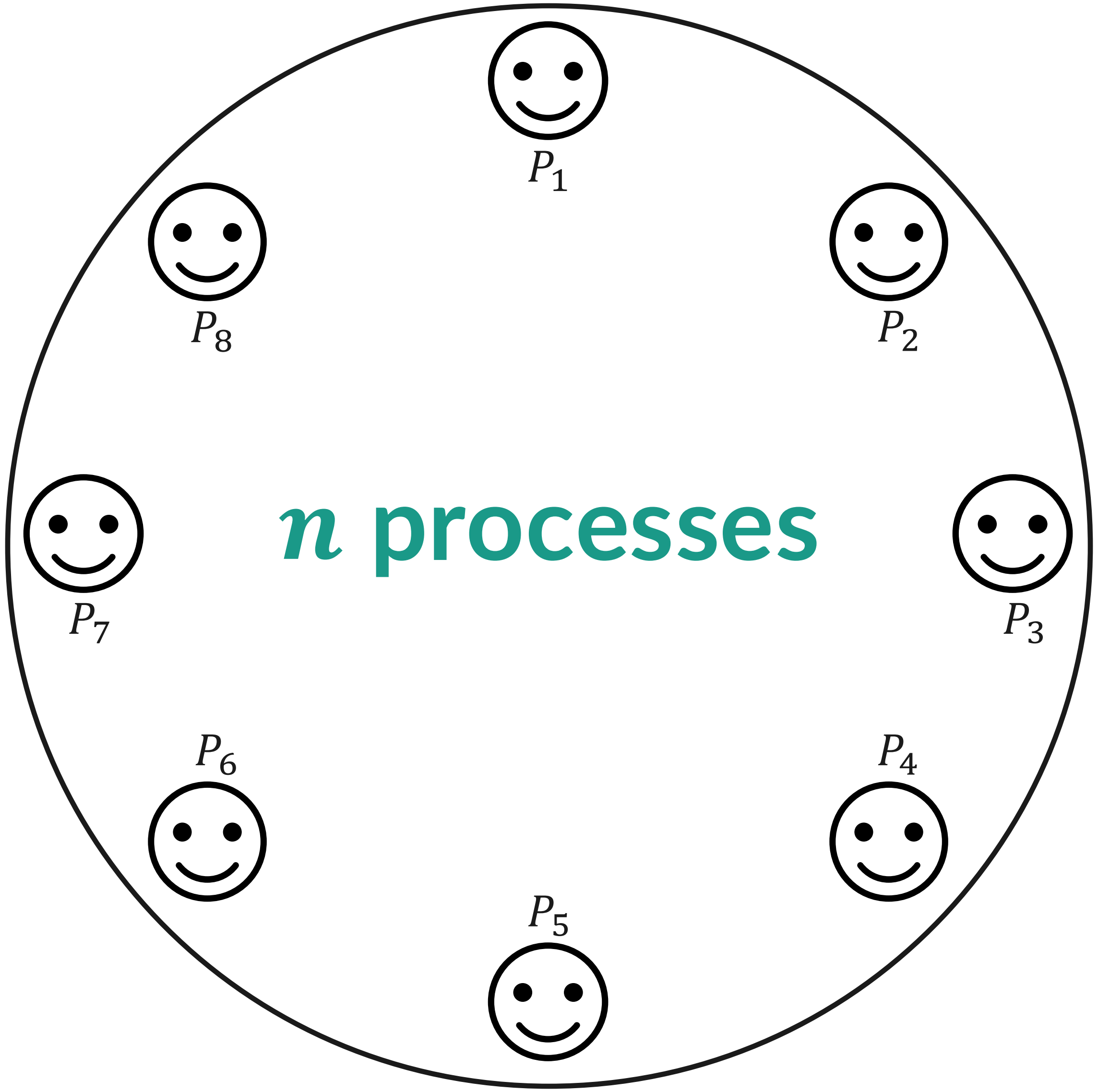
Positive Result 1: *There exists* a **randomized asynchronous** protocol that solves consensus, while tolerating arbitrary (**Byzantine**) failures

Positive Result 2: *There exists* a **deterministic synchronous** protocol that solves consensus, while tolerating arbitrary (**Byzantine**) failures

Consensus

Problem definition

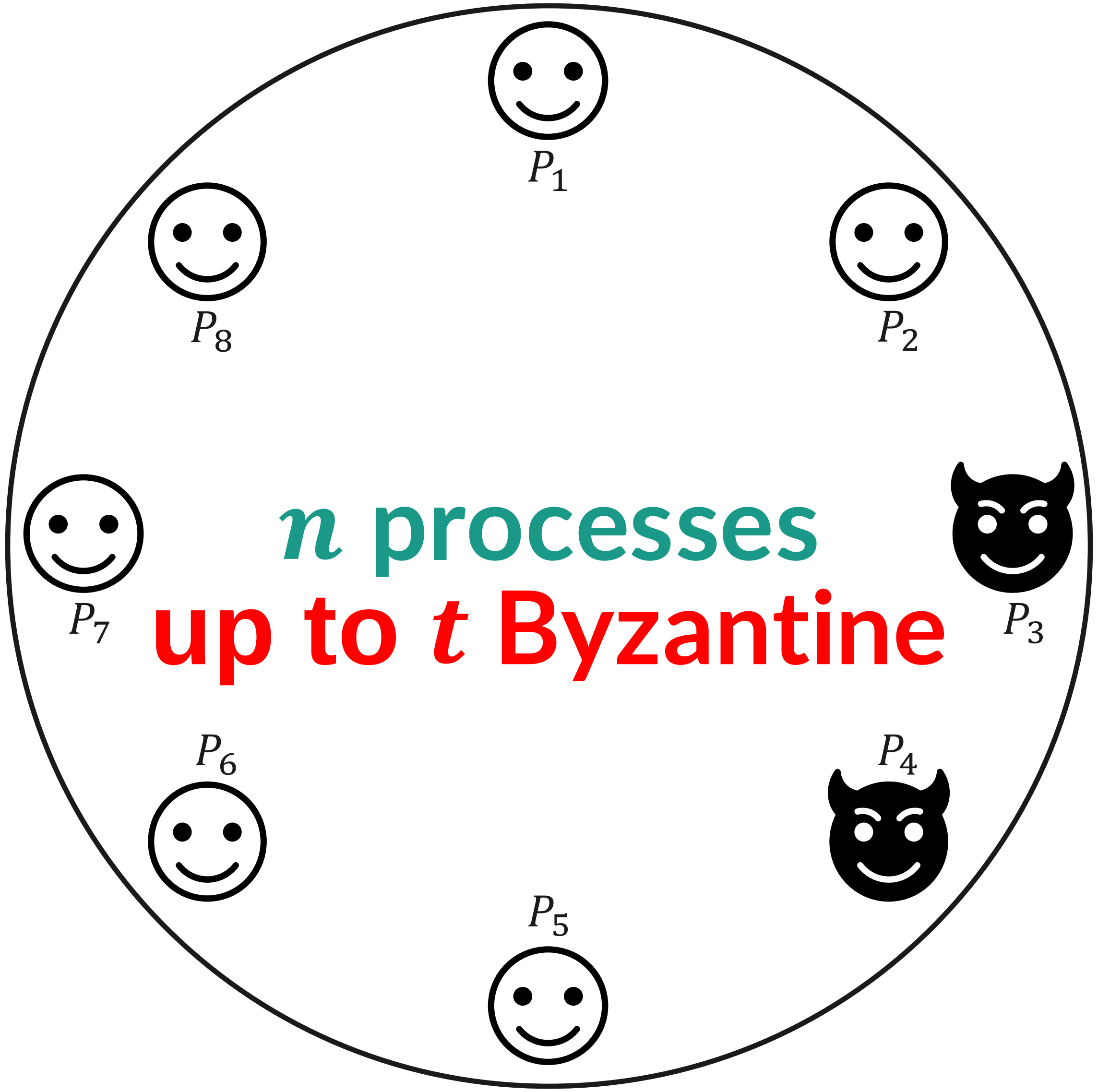
System



Static set of n publicly known identities

Message passing through reliable point-to-point channels

System



Static set of n publicly known identities

Message passing through reliable authenticated point-to-point channels

Interface of Consensus

Propose

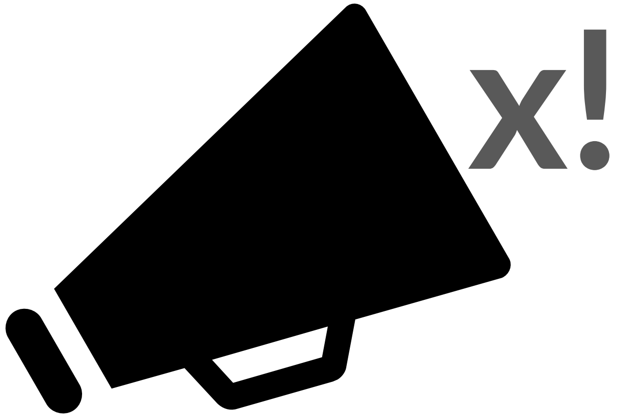
Operation

Decide

Callback

Interface of Consensus

Propose
Operation



Decide
Callback

Interface of Consensus

Propose

Operation

x!

Decide

Callback

y!

Properties of Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Agreement:** No two correct processes decide different values.
- **Validity:** A decided value is proposed by a correct process.

Asynchronous Model

No notion of time

Asynchronous Model (Informal)

- No shared global clock: no shared notion of time
- Arbitrary (but finite) message delays
- Model is purely event-driven: reception → sending

Synchronous Model

Known upper bound Δ on message delays
→ rounds of the form compute, send, receive

Graded Consensus

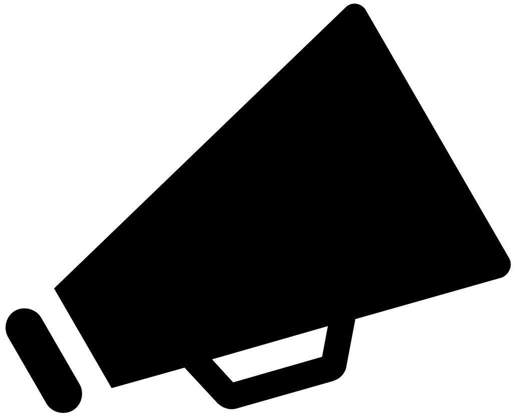
« Stay safe »

Graded Consensus

Specification

Interface of graded consensus

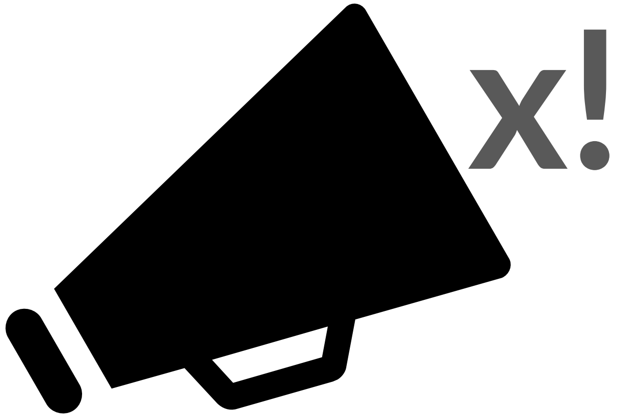
Propose
Operation



Decide
Callback

Interface of graded consensus

Propose
Operation



Decide
Callback

Interface of graded consensus

Propose
Operation

x!

Decide
Callback

(y,g)!

Interface of graded consensus

Propose
Operation

x!

Decide
Callback

(y,g)!

Grade = ‘confidence’

$g_{min}, \dots, g_{max}$

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only [redacted] can be decided.
- **Consistency (~~Agreement~~):** [redacted]

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** [redacted]

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have [redacted].

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have (a) |g-g'|≤ [redacted].

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have (a) |g-g'|≤1 [redacted].

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have (a) |g-g'|≤1, **and** (b) [redacted].

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have (a) |g-g'|≤1, **and** (b) if v≠v', then [redacted].

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have (a) |g-g'|≤1, **and** (b) if v≠v', then g=g'=0.

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have **either** (i) [redacted] **or** (ii) [redacted], and [redacted].

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have **either** (i) g=g'=0 **or** (ii) [redacted], and [redacted].

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have **either** (i) g=g'=0 **or** (ii) |g-g'|≤1, and [redacted].

Properties of Graded Consensus

- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have **either** (i) g=g'=0 **or** (ii) |g-g'|≤1, and v=v'.

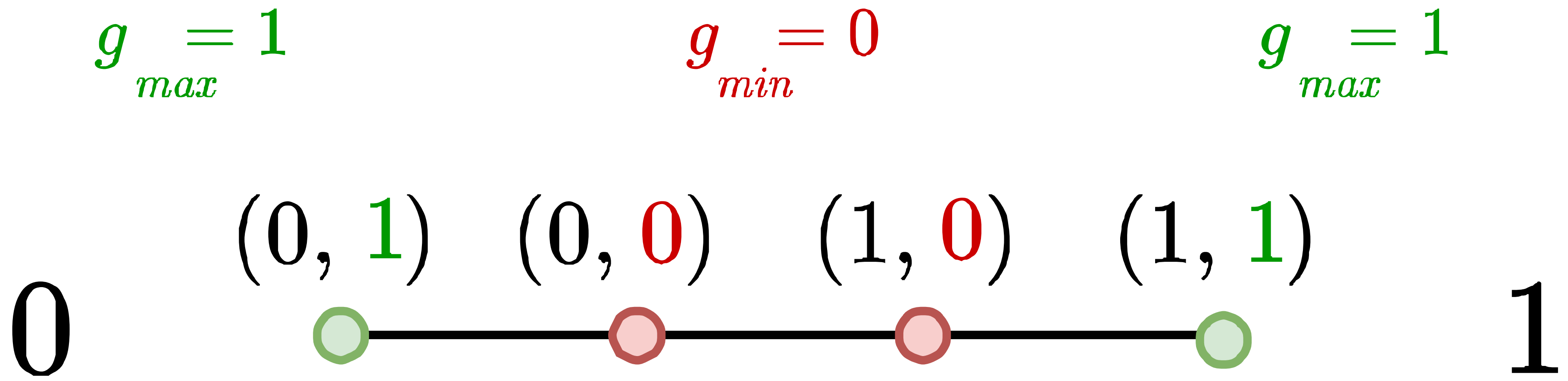
$g_{max} = 1$ $\quad$ $g_{min} = 0$ $\quad$ $g_{max} = 1$

$(0, 1)$ $\quad$ $(0, 0)$ $\quad$ $(1, 0)$ $\quad$ $(1, 1)$

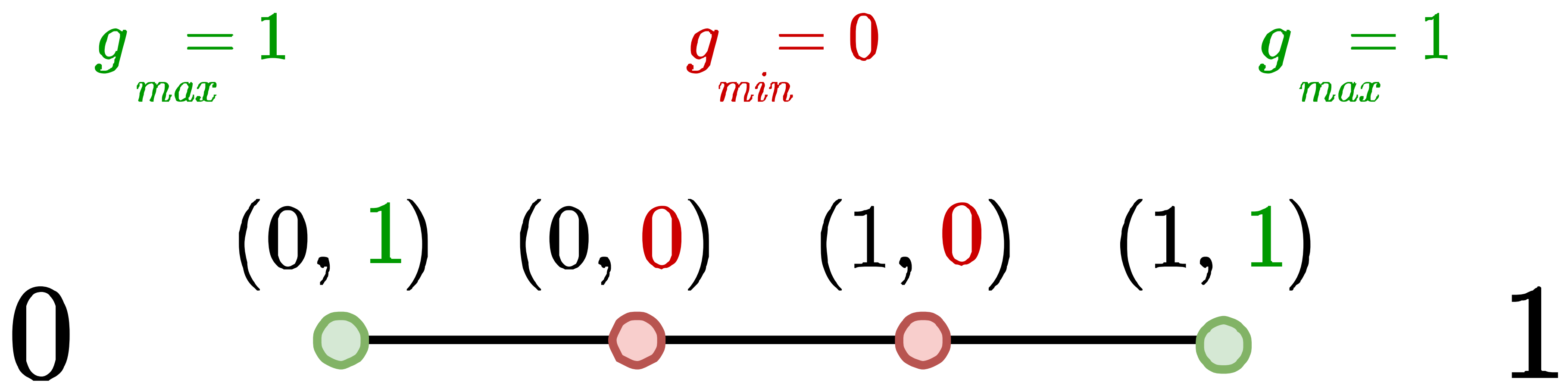
0

1

Consistency + Termination + Unanimity



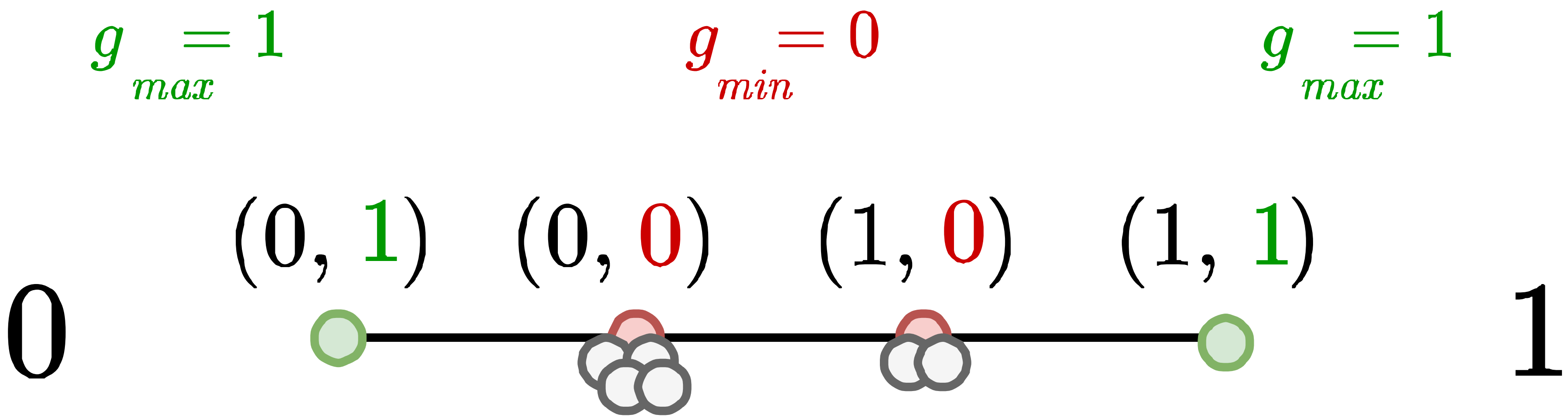
Consistency + Termination + Unanimity



(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) |g-g'|≤1, **and** (b) (v≠v' ⟹ g=g'=0).

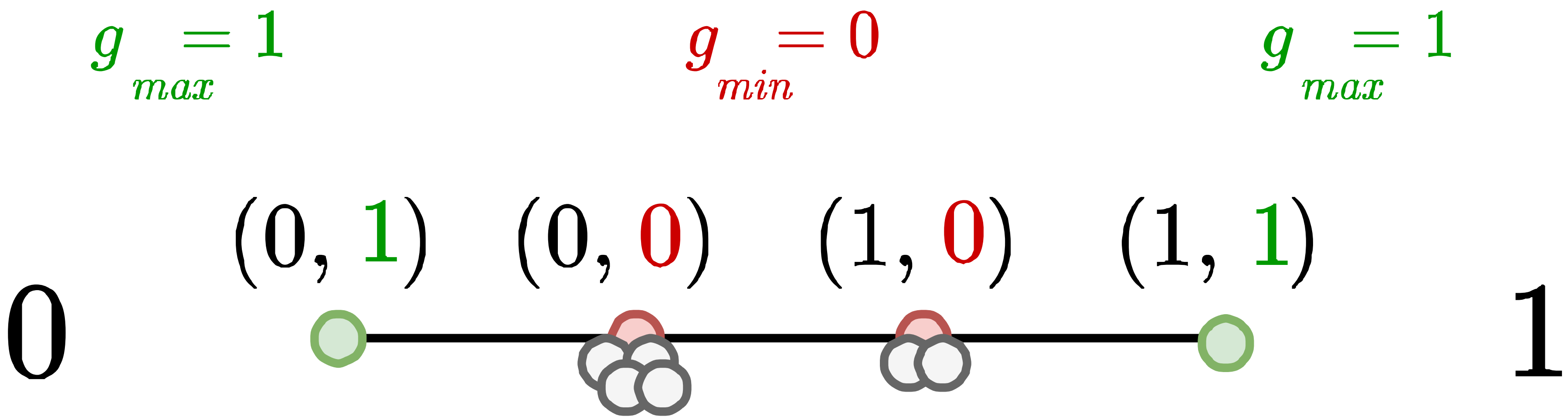
Consistency + Termination + Unanimity



(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) |g-g'|≤1, **and** (b) (v≠v' ⟹ g=g'=0).

Consistency + Termination + Unanimity

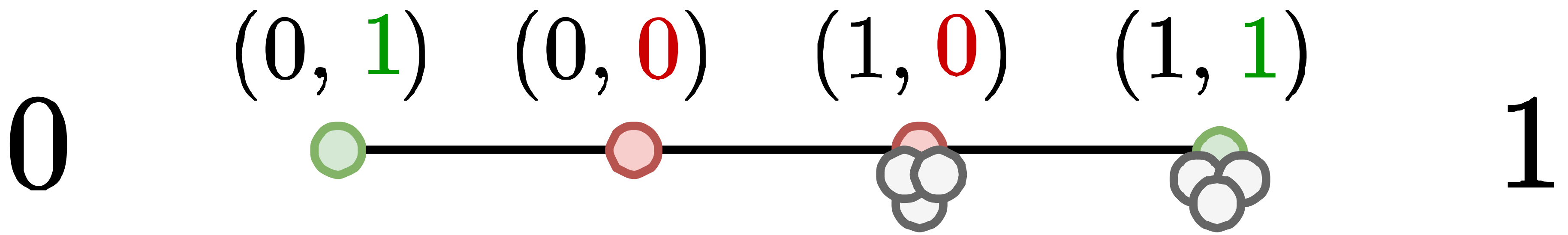


(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) $\boxed{g=g'=0}$, **or** (ii) $|g-g'| \leq 1$, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) $|g-g'| \leq 1$, **and** (b) ($v \neq v' \implies g=g'=0$).

Consistency + **Termination** + **Unanimity**

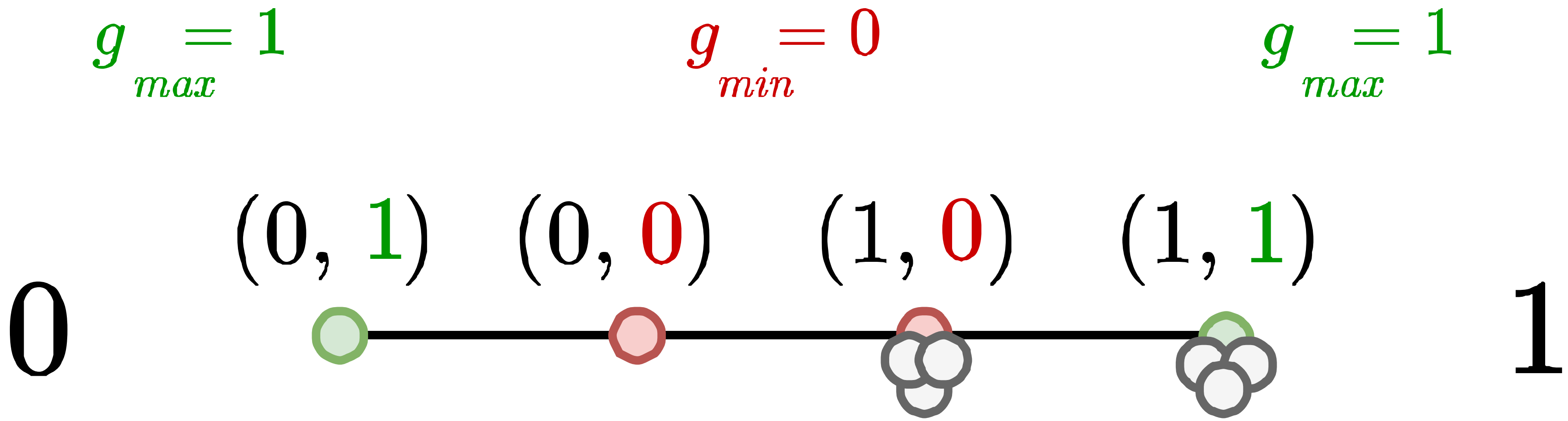
$g_{max} = 1$ $g_{min} = 0$ $g_{max} = 1$



(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) |g-g'|≤1, **and** (b) (v≠v' ⟹ g=g'=0).

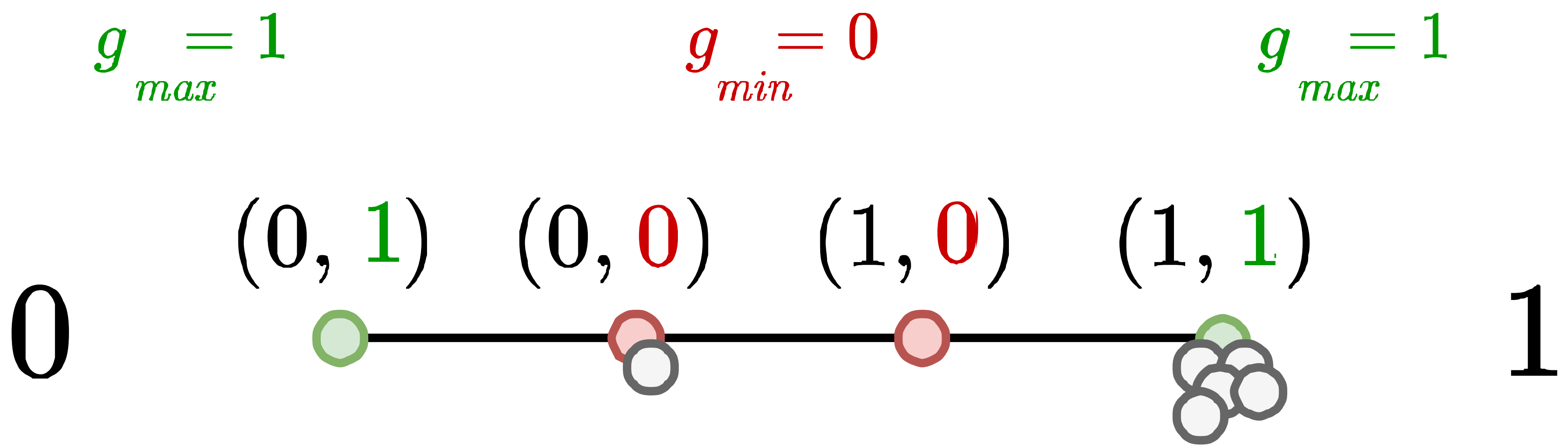
Consistency + Termination + Unanimity



(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) |g-g'|≤1, **and** (b) (v≠v' ⟹ g=g'=0).

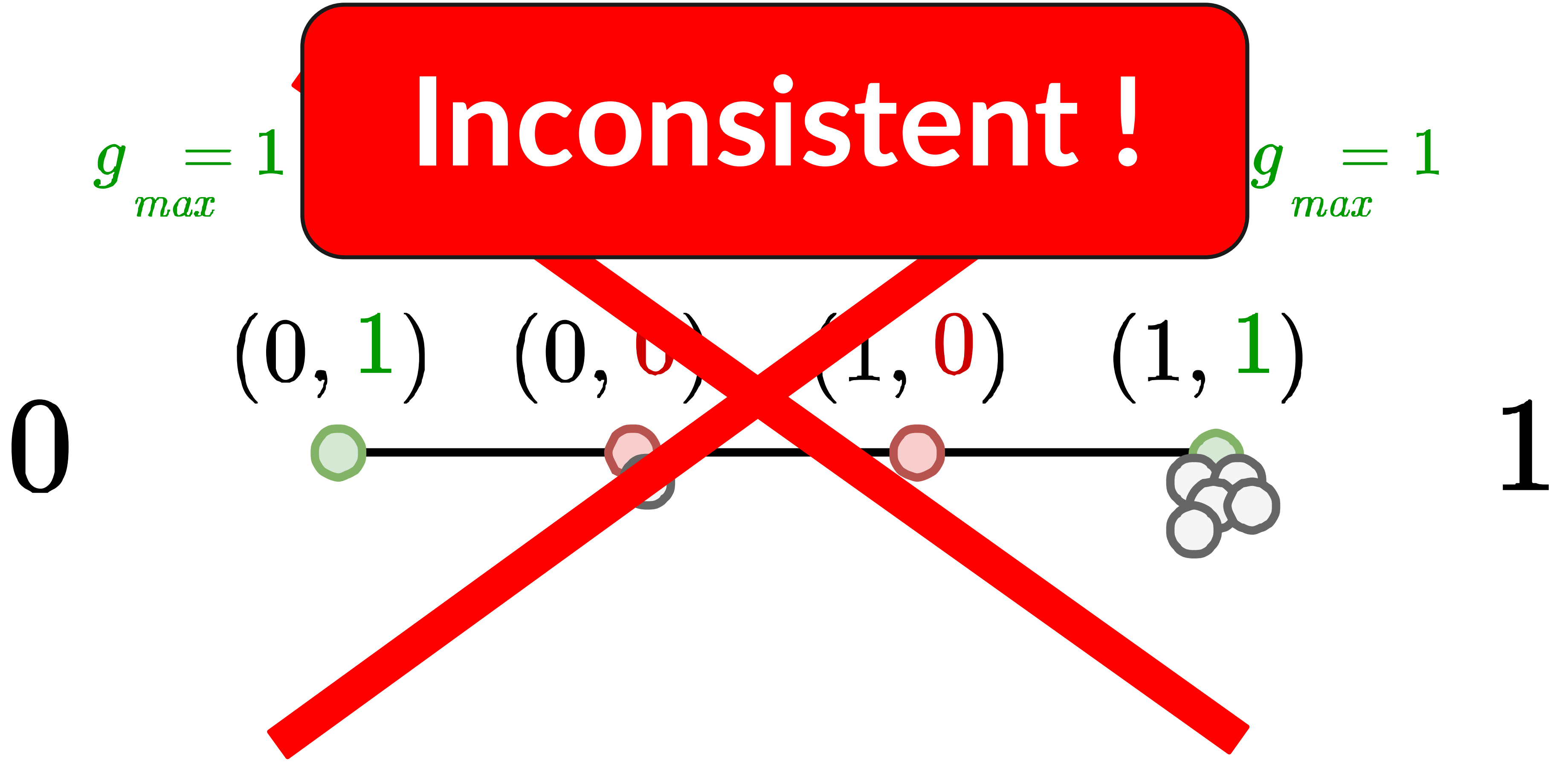
Consistency + Termination + Unanimity



(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) |g-g'|≤1, **and** (b) (v≠v' ⟹ g=g'=0).

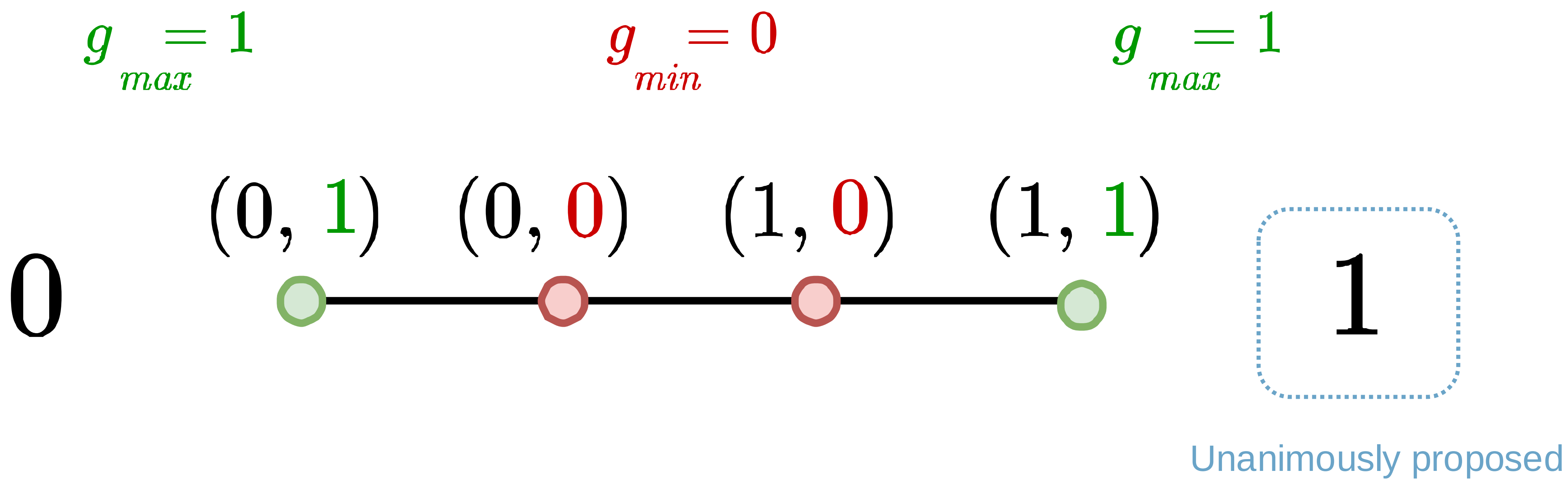
Consistency + Termination + Unanimity



(1) If two correct processes decide (v,g) and (v’,g’), then **either** (i) g=g’=0, **or** (ii) |g-g’|≤1, and v=v’.

(2) If two correct processes decide (v,g) and (v’,g’), then (a) |g-g’|≤1, **and** (b) ($\boxed{v \neq v' \Rightarrow g=g'=0}$).

Consistency + Termination + Unanimity



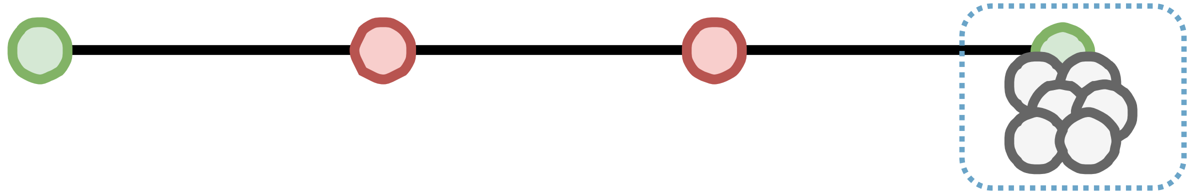
All correct processes eventually decide the unanimous proposal with high confidence value.

Consistency + Termination + Unanimity

$g_{max} = 1$ $g_{min} = 0$ $g_{max} = 1$

$(0, 1)$ $(0, 0)$ $(1, 0)$ $(1, 1)$

0



1

Unanimously proposed

All correct processes eventually decide the unanimous proposal with high confidence value.

Graded Consensus

Asynchronous implementation with t<n/? Byzantine failures

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from [redacted] processes:
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon ⟨ECHO, ·⟩ is received from [redacted] processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from [redacted] processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from [redacted] processes:
    if ∃v'' ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from [redacted] processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from [redacted] processes:
    if ∃v'' ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, $\cdot$⟩ is received from [redacted] processes:
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon ⟨ECHO, $\cdot$⟩ is received from [redacted] processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Termination:

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from [redacted] processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from [redacted] processes:
    if ∃v'' ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Termination: Every correct process eventually triggers a proposal.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
    $\boxed{\textbf{broadcast}\ \langle \text{PROPOSAL}, v_i \rangle}$
upon $\langle$PROPOSAL, $\cdot\rangle$ is received from [redacted] processes:
    broadcast $\langle$ECHO, $v'\rangle$, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$
upon $\langle$ECHO, $\cdot\rangle$ is received from [redacted] processes:
    if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Termination: Every correct process eventually triggers a proposal. Thus, every correct process eventually broadcasts a PROPOSAL message

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩
upon ⟨PROPOSAL, ·⟩ is received from [redacted] processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v') > (n − t)/2
upon ⟨ECHO, ·⟩ is received from [redacted] processes:
    if ∃v'' ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Termination: Every correct process eventually triggers a proposal. Thus, every correct process eventually broadcasts a PROPOSAL message and receives [redacted] PROPOSAL messages.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from $n - t$ processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon ⟨ECHO, ·⟩ is received from [redacted] processes:
    if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Termination: Every correct process eventually triggers a proposal. Thus, every correct process eventually broadcasts a PROPOSAL message and receives $(n - t)$ PROPOSAL messages.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v′⟩ where v′ denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v′) > (n − t)/2

upon ⟨ECHO, ·⟩ is received from [redacted] processes:
    if ∃v″ ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v″:
        trigger decide(v″, 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Termination: Every correct process eventually triggers a proposal. Thus, every correct process eventually broadcasts a PROPOSAL message and receives $(n - t)$ PROPOSAL messages. Consequently, every correct process eventually broadcasts an ECHO message

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from [redacted] processes:
    if ∃v'' ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Termination: Every correct process eventually triggers a proposal. Thus, every correct process eventually broadcasts a PROPOSAL message and receives $(n - t)$ PROPOSAL messages. Consequently, every correct process eventually broadcasts an ECHO message and receives [redacted] ECHO messages

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Termination: Every correct process eventually triggers a proposal. Thus, every correct process eventually broadcasts a PROPOSAL message and receives $(n - t)$ PROPOSAL messages. Consequently, every correct process eventually broadcasts an ECHO message and receives $(n - t)$ ECHO messages

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from $n - t$ processes:
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon ⟨ECHO, ·⟩ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Termination: Every correct process eventually triggers a proposal. Thus, every correct process eventually broadcasts a PROPOSAL message and receives $(n - t)$ PROPOSAL messages. Consequently, every correct process eventually broadcasts an ECHO message and receives $(n - t)$ ECHO messages before triggering a decision.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from $n - t$ processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon ⟨ECHO, ·⟩ is received from $n - t$ processes:
    if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Unanimity Property:

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon $\boxed{\text{propose}(v_i \in \text{Binary\_Value})}$
  broadcast $\langle$PROPOSAL, $v_i\rangle$

upon $\langle$PROPOSAL, $\cdot\rangle$ is received from $n - t$ processes:
  broadcast $\langle$ECHO, $v'\rangle$, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon $\langle$ECHO, $\cdot\rangle$ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
    trigger decide$(v'', 1)$
  else:
    trigger decide$(v^*, 0)$, where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Unanimity Property: Suppose all correct processes propose the same value v.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  $\boxed{\textbf{broadcast}\ \langle \text{PROPOSAL}, v_i \rangle}$

upon ⟨PROPOSAL, ·⟩ is received from $n - t$ processes:
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon ⟨ECHO, ·⟩ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, $\cdot$⟩ is received from $n - t$ processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon ⟨ECHO, $\cdot$⟩ is received from $n - t$ processes:
    if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n - t)$ PROPOSAL messages,

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast ⟨PROPOSAL, v_i⟩

upon $\boxed{\langle\text{PROPOSAL}, \cdot\rangle \text{ is received from } n - t \text{ processes:}}$
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon ⟨ECHO, ·⟩ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n - t)$ PROPOSAL messages, including at least $\boxed{(n - 2t)}$ with value $\boxed{v}$ and at most $\boxed{t}$ with value $\boxed{1 - v}$.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩
upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v′⟩, where v′ denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v′) > (n − t)/2
upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v″ ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v″:
        trigger decide(v″, 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n-t)$ PROPOSAL messages, including at least $\boxed{(n-2t)}$ with value $\boxed{v}$ and at most $\boxed{t}$ with value $\boxed{1-v}$. Given that $n > 3t$,

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v′⟩, where v′ denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v′) > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v″ ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v″:
        trigger decide(v″, 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n - t)$ PROPOSAL messages, including at least $(n - 2t)$ with value v and at most t with value $1 - v$. Given that $n > 3t$, we have $(n - 2t) > t$, ensuring

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩
upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩ where v' denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v') > (n − t)/2
upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n - t)$ PROPOSAL messages, including at least $(n - 2t)$ with value v and at most t with value $1 - v$. Given that $n > 3t$, we have $(n - 2t) > t$, ensuring that every correct process broadcasts an ECHO message with value v.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n - t)$ PROPOSAL messages, including at least $(n - 2t)$ with value v and at most t with value $1 - v$. Given that $n > 3t$, we have $(n - 2t) > t$, ensuring that every correct process broadcasts an ECHO message with value v. Thus, each correct process receives $(n - t)$ ECHO messages,

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast $\langle$PROPOSAL, $v_i\rangle$

upon $\langle$PROPOSAL, $\cdot\rangle$ is received from $n - t$ processes:
  broadcast $\langle$ECHO, $v'\rangle$, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon $\langle$ECHO, $\cdot\rangle$ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least [redacted] ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n-t)$ PROPOSAL messages, including at least $(n-2t)$ with value v and at most t with value $1-v$. Given that $n > 3t$, we have $(n-2t) > t$, ensuring that every correct process broadcasts an ECHO message with value v. Thus, each correct process receives $(n-t)$ ECHO messages, including at least [redacted] with value v,

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n - t)$ PROPOSAL messages, including at least $(n - 2t)$ with value v and at most t with value $1 - v$. Given that $n > 3t$, we have $(n - 2t) > t$, ensuring that every correct process broadcasts an ECHO message with value v. Thus, each correct process receives $(n - t)$ ECHO messages, including at least $(n - 2t)$ with value v,

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Unanimity Property: Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n-t)$ PROPOSAL messages, including at least $(n-2t)$ with value v and at most t with value $1-v$. Given that $n > 3t$, we have $(n-2t) > t$, ensuring that every correct process broadcasts an ECHO message with value v. Thus, each correct process receives $(n-t)$ ECHO messages, including at least $(n-2t)$ with value v, and consequently decides on $(v, 1)$.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from $n - t$ processes:
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon ⟨ECHO, ·⟩ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Consistency:

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
    broadcast $\langle$PROPOSAL, $v_i\rangle$

upon $\langle$PROPOSAL, $\cdot\rangle$ is received from $n - t$ processes:
    broadcast $\langle$ECHO, $v'\rangle$, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon $\langle$ECHO, $\cdot\rangle$ is received from $n - t$ processes:
    if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
        $\boxed{\textbf{trigger}\ \text{decide}(v'', 1)}$
    else:
        trigger decide($v^*, 0$), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate).

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v′⟩, where v′ denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v′) > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v″ ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v″:
        trigger decide(v″, 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
    broadcast $\langle$PROPOSAL, $v_i\rangle$

upon $\langle$PROPOSAL, $\cdot\rangle$ is received from $n - t$ processes:
    broadcast $\langle$ECHO, $v'\rangle$, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon $\langle$ECHO, $\cdot\rangle$ is received from $n - t$ processes:
    if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j,

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v′⟩, where v′ denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v′) > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v″ ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v″:
        trigger decide(v″, 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
    broadcast $\langle$PROPOSAL, $v_i\rangle$

upon $\langle$PROPOSAL, $\cdot\rangle$ is received from $n - t$ processes:
    broadcast $\langle$ECHO, $v'\rangle$, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon $\langle$ECHO, $\cdot\rangle$ is received from $n - t$ processes:
    if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, $\cdot$⟩ is received from $n - t$ processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon ⟨ECHO, $\cdot$⟩ is received from $n - t$ processes:
    if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

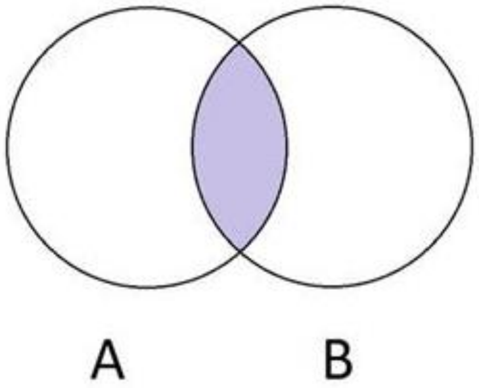
Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| =$

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, $\cdot$⟩ is received from $n - t$ processes:
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.

upon ⟨ECHO, $\cdot$⟩ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages.



LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j|$

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v′⟩, where v′ denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v′) > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v″ ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v″:
        trigger decide(v″, 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n - 2t) + (n - t) - n$

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
 broadcast $\langle$PROPOSAL, $v_i\rangle$

upon $\langle$PROPOSAL, $\cdot\rangle$ is received from $n - t$ processes:
 broadcast $\langle$ECHO, $v'\rangle$, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon $\langle$ECHO, $\cdot\rangle$ is received from $n - t$ processes:
 if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
  trigger decide(v'', 1)
 else:
  trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n - 2t) + (n - t) - n = n - 3t$,

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v′⟩, where v′ denotes the value with the highest frequency among the PROPOSAL messages.    ▷ #(v′) > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v′′ ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v′′:
        trigger decide(v′′, 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.    ▷ #(v*) > (n − t)/2
```

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n - 2t) + (n - t) - n = n - 3t$, so $|Q_i \cap Q_j \cap \text{Corrects}| \geq$

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from $n - t$ processes:
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon ⟨ECHO, ·⟩ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n - 2t) + (n - t) - n = n - 3t$, so $|Q_i \cap Q_j \cap \text{Corrects}| \geq n - 4t$.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, $\cdot$⟩ is received from $n - t$ processes:
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon ⟨ECHO, $\cdot$⟩ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n-2t) + (n-t) - n = n - 3t$, so $|Q_i \cap Q_j \cap \text{Corrects}| \geq n - 4t$. Therefore, process p_j receives at least [redacted] ECHO messages with value w, which ensures that [redacted] if [redacted]

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/$X Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v′⟩, where v′ denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v′) > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v″ ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v″:
        trigger decide(v″, 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and $n/$X-resiliency.*

Proof.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n - 2t) + (n - t) - n = n - 3t$, so $|Q_i \cap Q_j \cap \text{Corrects}| \geq n - 4t$. Therefore, process p_j receives at least $n - 4t$ ECHO messages with value w, which ensures that [redacted] if [redacted]

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
    broadcast $\langle$PROPOSAL, $v_i\rangle$

upon $\langle$PROPOSAL, $\cdot\rangle$ is received from $n - t$ processes:
    broadcast $\langle$ECHO, $v'\rangle$, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n - t)/2$

upon $\langle$ECHO, $\cdot\rangle$ is received from $n - t$ processes:
    if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n - t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n - 2t) + (n - t) - n = n - 3t$, so $|Q_i \cap Q_j \cap \text{Corrects}| \geq n - 4t$. Therefore, process p_j receives at least $n - 4t$ ECHO messages with value w, which ensures that $w' = w$ if [redacted]

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

upon propose($v_i \in$ Binary_Value):
  broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from $n - t$ processes:
  broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v') > (n-t)/2$

upon ⟨ECHO, ·⟩ is received from $n - t$ processes:
  if $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ ECHO messages contain value v'':
    trigger decide(v'', 1)
  else:
    trigger decide(v^*, 0), where v^* denotes the value with the highest frequency among the ECHO messages. ▷ $\#(v^*) > (n-t)/2$

LEMMA 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

PROOF.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n-2t) + (n-t) - n = n - 3t$, so $|Q_i \cap Q_j \cap \text{Corrects}| \geq n - 4t$. Therefore, process p_j receives at least $n - 4t$ ECHO messages with value w, which ensures that $w' = w$ if $n - 4t >$ [redacted]

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/X$ Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and n/X-resiliency.*

Proof.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n - 2t) + (n - t) - n = n - 3t$, so $|Q_i \cap Q_j \cap \text{Corrects}| \geq n - 4t$. Therefore, process p_j receives at least $n - 4t$ ECHO messages with value w, which ensures that $w' = w$ if $n - 4t > (n - t)/2$,

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/7$: Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and $n/7$-resiliency.*

Proof.

Consistency: Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received ECHO messages with value w from a set Q_i of $|Q_i| = n - 2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.

Process p_j's decision was based on receiving ECHO messages from a set Q_j with $|Q_j| = n - t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n - 2t) + (n - t) - n = n - 3t$, so $|Q_i \cap Q_j \cap \text{Corrects}| \geq n - 4t$. Therefore, process p_j receives at least $n - 4t$ ECHO messages with value w, which ensures that $w' = w$ if $n - 4t > (n - t)/2$, i.e., if $n > 7t$.

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/7$: Pseudocode (for process p_i)

```
upon propose(v_i ∈ Binary_Value):
    broadcast ⟨PROPOSAL, v_i⟩

upon ⟨PROPOSAL, ·⟩ is received from n − t processes:
    broadcast ⟨ECHO, v'⟩, where v' denotes the value with the highest frequency among the PROPOSAL messages.   ▷ #(v') > (n − t)/2

upon ⟨ECHO, ·⟩ is received from n − t processes:
    if ∃v'' ∈ Binary_Value, s.t. at least n − 2t ECHO messages contain value v'':
        trigger decide(v'', 1)
    else:
        trigger decide(v*, 0), where v* denotes the value with the highest frequency among the ECHO messages.   ▷ #(v*) > (n − t)/2
```

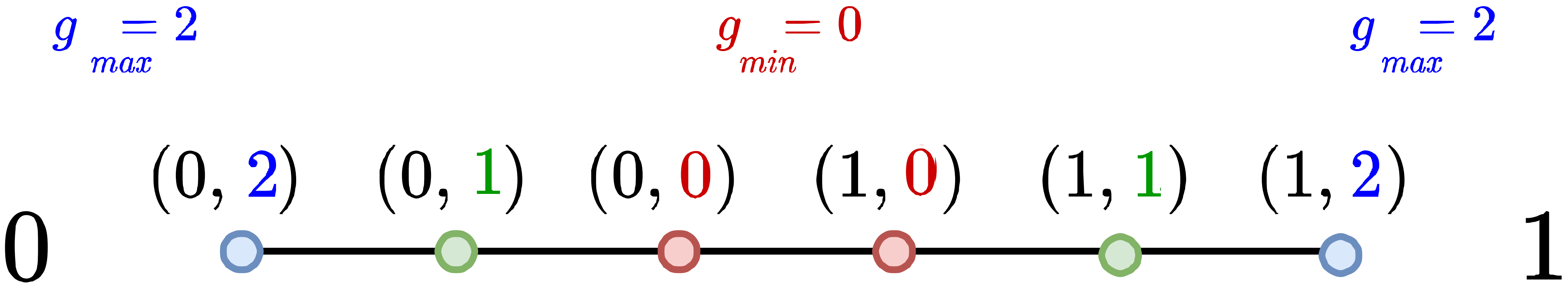
Lemma 1.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and $n/7$-resiliency.*

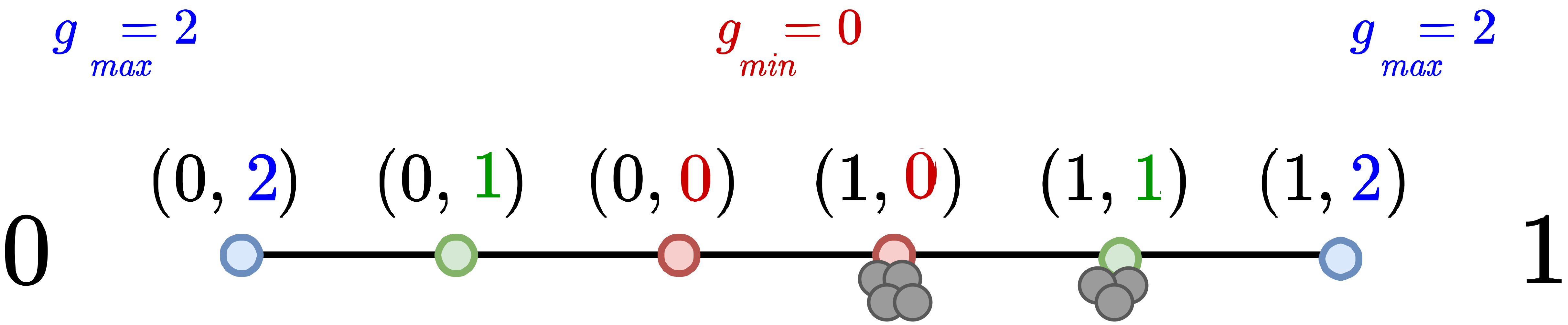
$$\boxed{Alg_1} \sqsubseteq \boxed{GC^{R=2}}$$

One more refinement

Properties of Graded Consensus

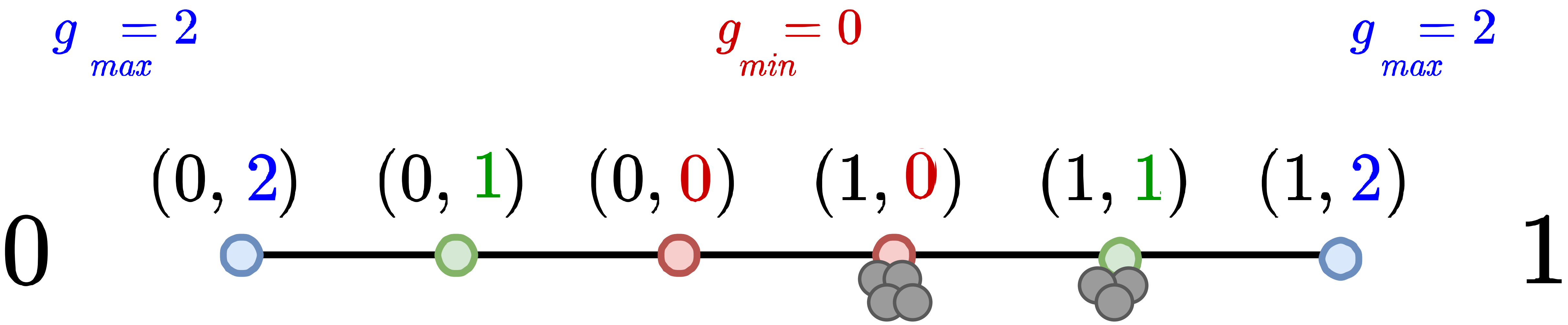
- **Termination:** All correct processes decide.
- **Integrity:** No process decides more than once.
- **Consistency (~~Agreement~~):** Assume two correct processes decide (v, g) and (v', g'). We have (a) |g-g'|≤1, and (b) if v≠v', then g=g'=0.
- **Unanimity:** If only v is proposed, then only (v, g_{max}) can be decided.


$g_{max} = 2$
$g_{min} = 0$
$g_{max} = 2$
$(0, 2)$
$(0, 1)$
$(0, 0)$
$(1, 0)$
$(1, 1)$
$(1, 2)$
0
1



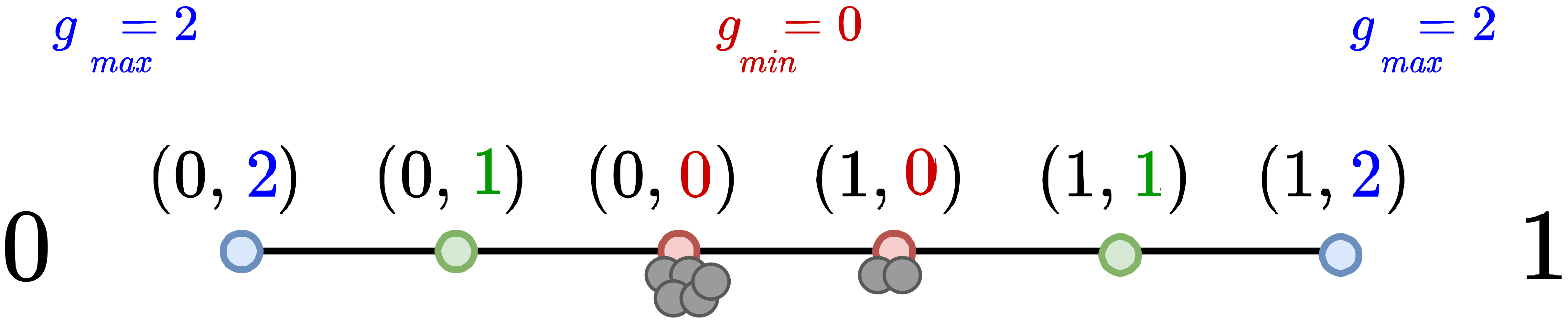
(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) |g-g'|≤1, **and** (b) (v≠v' ⟹ g=g'=0).



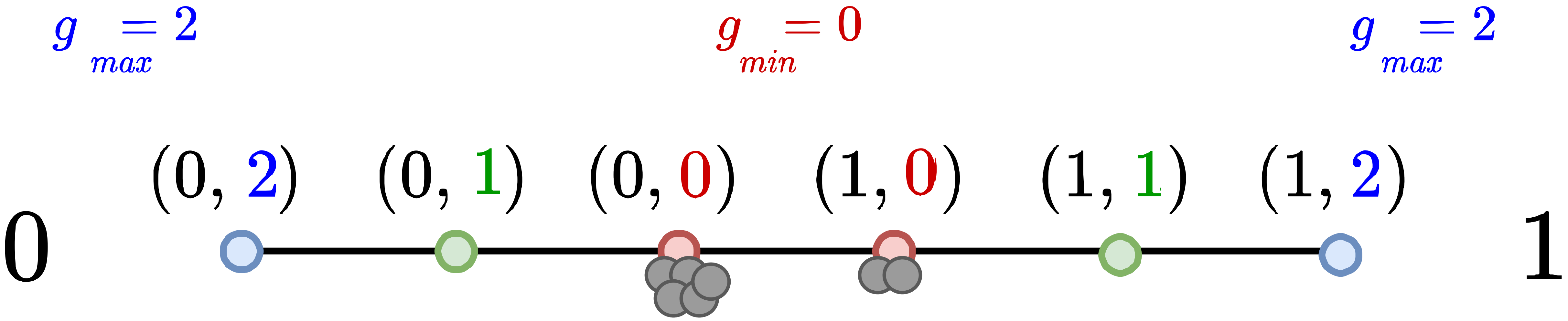
(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) |g-g'|≤1, **and** (b) (v≠v' ⟹ g=g'=0).



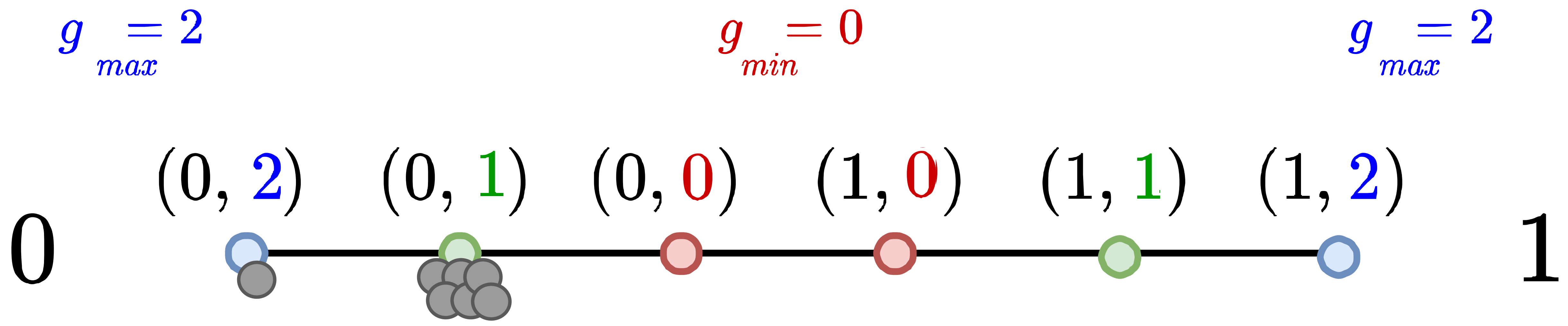
(1) If two correct processes decide (v,g) and (v’,g’), then **either** (i) g=g’=0, **or** (ii) |g-g’|≤1, and v=v’.

(2) If two correct processes decide (v,g) and (v’,g’), then (a) |g-g’|≤1, **and** (b) (v≠v’ ⟹ g=g’=0).



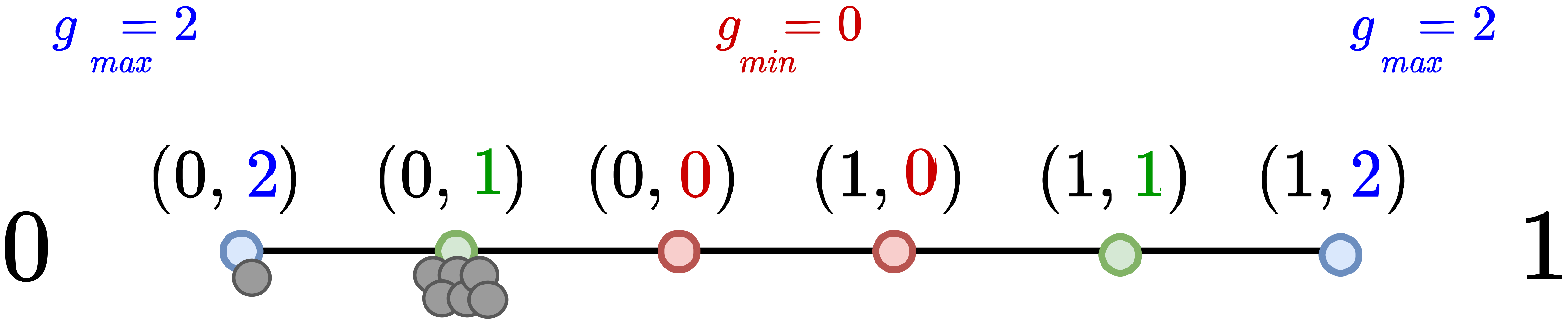
(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) $\boxed{g=g'=0}$, **or** (ii) $|g-g'| \leq 1$, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) $|g-g'| \leq 1$, **and** (b) ($v \neq v' \Rightarrow g=g'=0$).



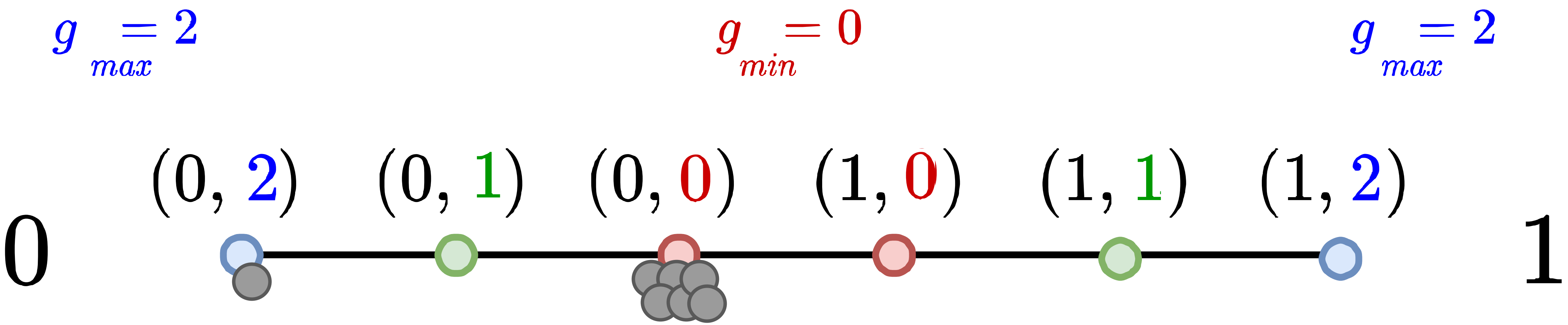
(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) |g-g'|≤1, **and** (b) (v≠v' ⟹ g=g'=0).



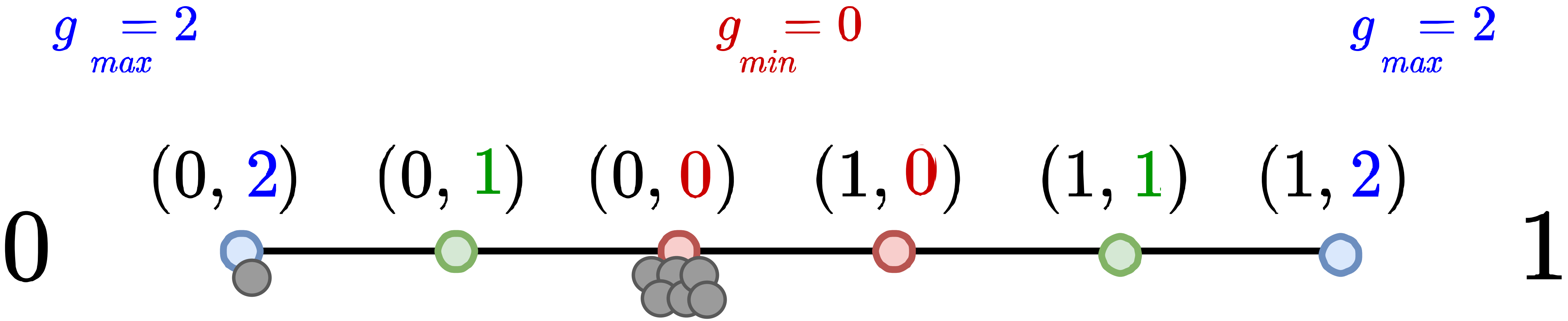
(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) $\boxed{|g-g'| \leq 1, \text{ and } v=v'.}$

(2) If two correct processes decide (v,g) and (v',g'), then (a) $|g-g'| \leq 1$, **and** (b) ($v \neq v' \implies g=g'=0$).



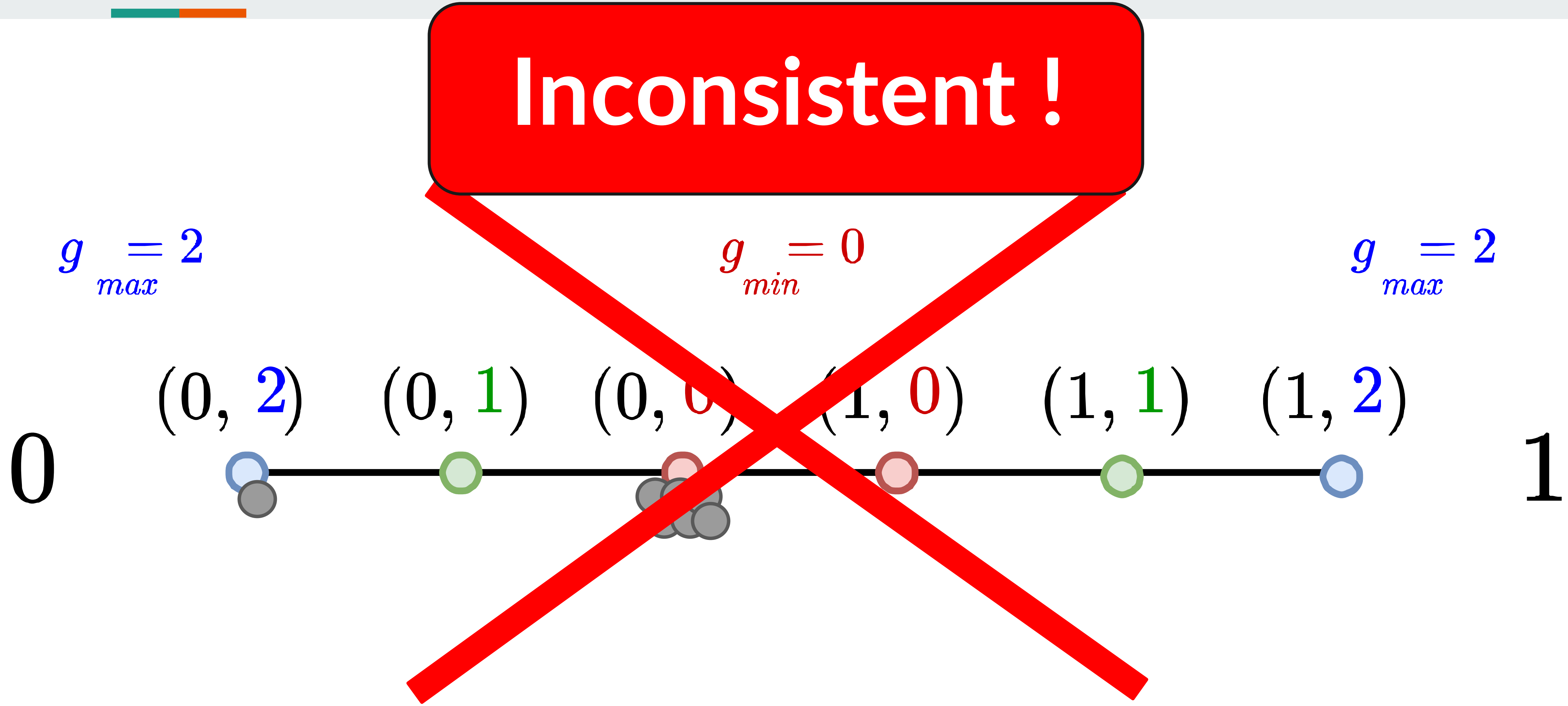
(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) |g-g'|≤1, **and** (b) (v≠v' ⟹ g=g'=0).



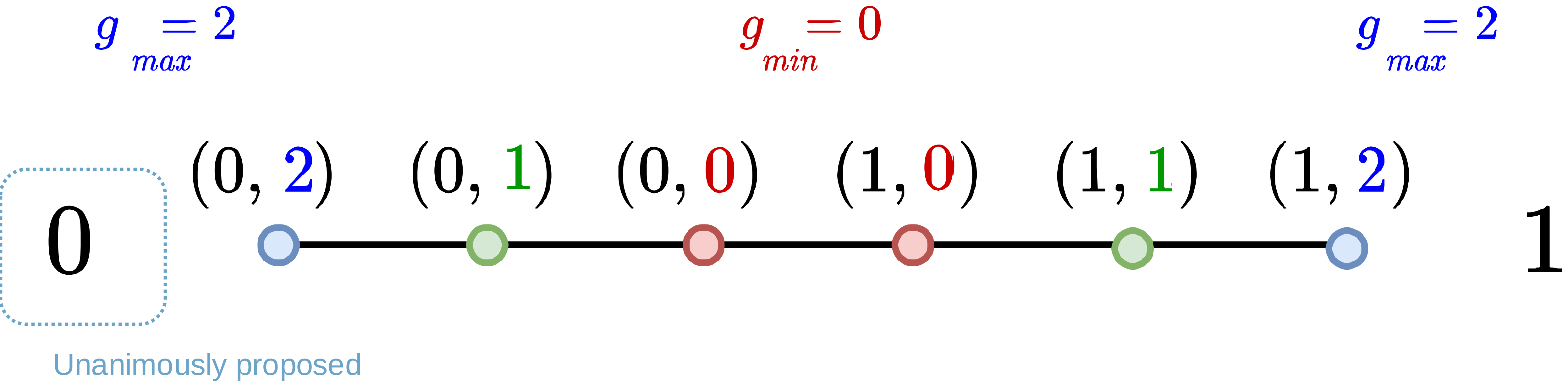
(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) $\boxed{|g-g'|\leq 1}$, **and** (b) (v≠v' ⟹ g=g'=0).

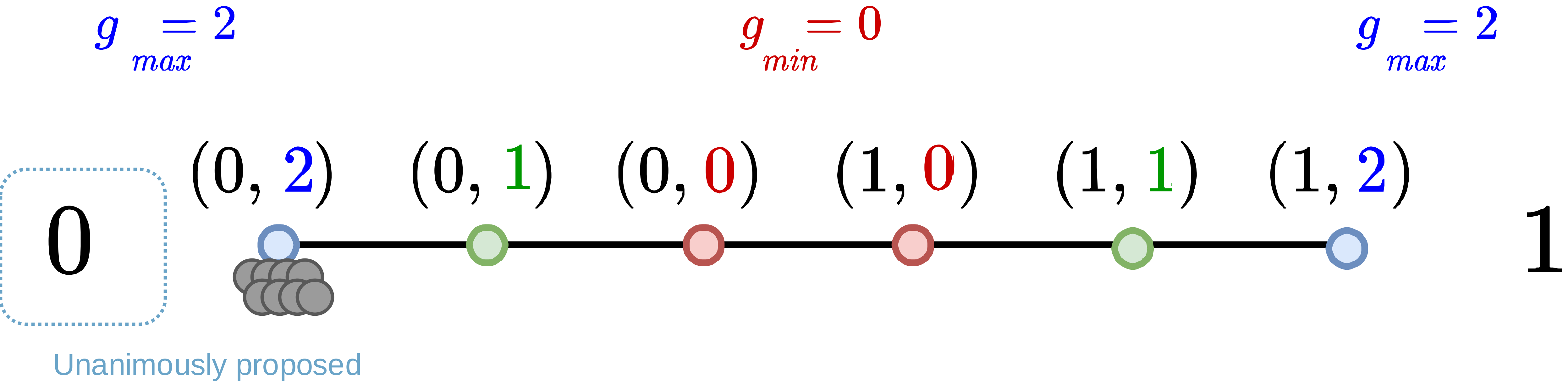


(1) If two correct processes decide (v,g) and (v',g'), then **either** (i) g=g'=0, **or** (ii) |g-g'|≤1, and v=v'.

(2) If two correct processes decide (v,g) and (v',g'), then (a) $\boxed{|g-g'|\leq 1}$, **and** (b) (v≠v' ⟹ g=g'=0).



All correct processes eventually decide the unanimous proposal with high confidence value.



All correct processes eventually decide the unanimous proposal with high confidence value.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
    Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
    Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
    invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
    upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
        [redacted] ▷ Update the grade (confidence)
        invoke $\mathcal{GC}_2$.propose([redacted]) ▷ Propose input value to the 2nd graded consensus instance
    upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
        [redacted] ▷ Update the grade (confidence)
        trigger decide([redacted]) ▷ Decide the final value and grade

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    [redacted] ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    [redacted] ▷ Update the grade (confidence)
    trigger decide(v_i^2, [redacted]) ▷ Decide the final value and grade

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance

  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    [redacted] ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance

  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    [redacted] ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▹ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
  Integer $g_i \leftarrow 0$ ▹ Grade

upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▹ Propose input value to the 1st graded consensus instance

  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▹ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▹ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▹ Propose input value to the 2nd graded consensus instance

  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▹ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▹ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▹ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
Binary Graded Consensus, **instances** $\mathcal{GC}_1$, $\mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
Integer $g_i \leftarrow 0$ ▷ Grade
upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

Lemma 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

Proof.

- Termination follows directly from the termination of $\mathcal{GC}_1$ and $\mathcal{GC}_2$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade
upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

PROOF.

- Termination follows directly from the termination of $\mathcal{GC}_1$ and $\mathcal{GC}_2$.
- Unanimity property follows directly from the unanimity property of $\mathcal{GC}_1$ and $\mathcal{GC}_2$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

```
Uses:
    Binary Graded Consensus, instances GC_1, GC_2   ▷ 2 instances of the Binary Graded Consensus protocol with Refinement R' = 2
Local Variables:
    Integer g_i ← 0                                  ▷ Grade
upon propose(v_i ∈ Value):
    invoke GC_1.propose(v_i)                         ▷ Propose input value to the 1st graded consensus instance
    upon GC_1.decide(v_i^1, g_i^1):                  ▷ Received decision from the 1st graded consensus instance
        g_i ← g_i + g_i^1                            ▷ Update the grade (confidence)
        invoke GC_2.propose(v_i^1)                   ▷ Propose input value to the 2nd graded consensus instance
    upon GC_2.decide(v_i^2, g_i^2):                  ▷ Received decision from the 2nd graded consensus instance
        g_i ← g_i + g_i^2                            ▷ Update the grade (confidence)
        trigger decide(v_i^2, g_i)                   ▷ Decide the final value and grade
```

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance

  upon $\boxed{\mathcal{GC}_1}$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance

  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

1  $(v_i^1 = w, 1)$!

Lemma 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

$\boxed{\text{Case 1: } j = 1.}$

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
 Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
 Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
 invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance

$(v_i^1 = w, 1)$!

$(v_j^1 = w, *)$!

 upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
  $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
  invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance

 upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
  $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
  trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

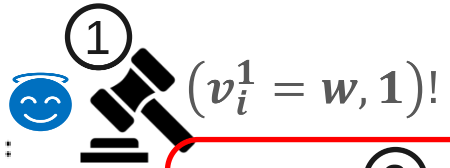
Case 1: $j = 1$. By consistency, each correct process outputs $(w, \cdot)$ from $\mathcal{GC}_1$ for some value w, and thus proposes w to $\mathcal{GC}_2$.

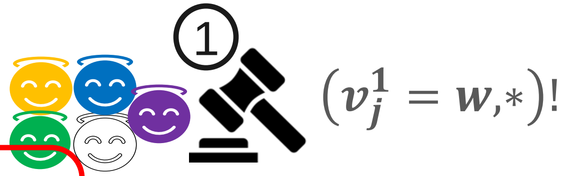
Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1$, $\mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance





  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance

  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 1: $j = 1$. By consistency, each correct process outputs $(w, \cdot)$ from $\mathcal{GC}_1$ for some value w, and thus proposes w to $\mathcal{GC}_2$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade
upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) 1 $(v_i^1 = w, 1)$! 1 $(v_j^1 = w, *)$! ▷ Propose input value to the 1st graded consensus instance
  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) 2 w! ▷ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): 2 $(v_*^2 = w, 1)$! ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

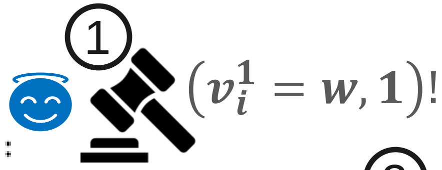
Lemma 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

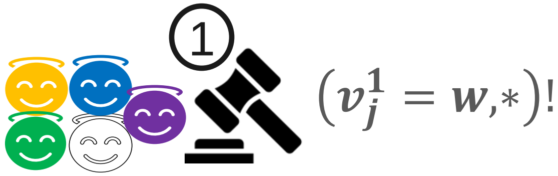
- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 1: $j = 1$. By consistency, each correct process outputs $(w, \cdot)$ from $\mathcal{GC}_1$ for some value w, and thus proposes w to $\mathcal{GC}_2$. Therefore, due to the unanimity property of graded consensus $\mathcal{GC}_2$, every correct process returns $(w, 1)$ from $\mathcal{GC}_2$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▹ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
Integer $g_i \leftarrow 0$ ▹ Grade
upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▹ Propose input value to the 1st graded consensus instance
  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▹ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▹ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▹ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▹ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▹ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▹ Decide the final value and grade





Lemma 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 1: $j = 1$. By consistency, each correct process outputs $(w, \cdot)$ from $\mathcal{GC}_1$ for some value w, and thus proposes w to $\mathcal{GC}_2$. Therefore, due to the unanimity property of graded consensus $\mathcal{GC}_2$, every correct process returns $(w, 1)$ from $\mathcal{GC}_2$. Hence, consistency follows directly from the consistency of $\mathcal{GC}_1$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
Integer $g_i \leftarrow 0$ ▷ Grade
upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
  upon $\boxed{\mathcal{GC}_2}$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

Lemma 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$. By construction, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
Integer $g_i \leftarrow 0$ ▷ Grade

upon propose$(v_i \in \text{Value})$:
invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance

upon $\boxed{\mathcal{GC}_1}$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
$g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance

upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
$g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

Lemma 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$. By construction, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance

$(*, \mathbf{0})$!

upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
$g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance

upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
$g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

$(*,*)$!

Lemma 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$. By construction, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$. Therefore, due to the consistency property of graded consensus $\mathcal{GC}_2$,

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance

  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$. By construction, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$. Therefore, due to the consistency property of graded consensus $\mathcal{GC}_2$, if two correct processes p_i and p_j decide on (v_i, g_i) and (v_j, g_j), respectively, then

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
    Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$

Local Variables:
    Integer $g_i \leftarrow 0$ ▷ Grade

upon propose($v_i \in$ Value):
    invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance

    upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
        $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
        invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance

    upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
        $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
        trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$. By construction, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$. Therefore, due to the consistency property of graded consensus $\mathcal{GC}_2$, if two correct processes p_i and p_j decide on (v_i, g_i) and (v_j, g_j), respectively, then $|g_i - g_j| \leq 1$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
Integer $g_i \leftarrow 0$ ▷ Grade
upon propose($v_i \in$ Value):
invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
$g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
$g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$. By construction, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$. Therefore, due to the consistency property of graded consensus $\mathcal{GC}_2$, if two correct processes p_i and p_j decide on (v_i, g_i) and (v_j, g_j), respectively, then $|g_i - g_j| \leq 1$. Moreover,

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade
upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance

  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance

  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$. By construction, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$. Therefore, due to the consistency property of graded consensus $\mathcal{GC}_2$, if two correct processes p_i and p_j decide on (v_i, g_i) and (v_j, g_j), respectively, then $|g_i - g_j| \leq 1$. Moreover, if $g_i \neq 0$,

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade
upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

Lemma 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$. By construction, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$. Therefore, due to the consistency property of graded consensus $\mathcal{GC}_2$, if two correct processes p_i and p_j decide on (v_i, g_i) and (v_j, g_j), respectively, then $|g_i - g_j| \leq 1$. Moreover, if $g_i \neq 0$, $v_i = v_j$.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
  Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
  Integer $g_i \leftarrow 0$ ▷ Grade
upon propose($v_i \in$ Value):
  invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance

  upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
    $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
    invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
  upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
    $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
    trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

LEMMA 1.2. *Algorithm 2 implements graded consensus with the refinement parameter $R = 3$.*

- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.

Case 2: $j = 2$. By construction, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$. Therefore, due to the consistency property of graded consensus $\mathcal{GC}_2$, if two correct processes p_i and p_j decide on (v_i, g_i) and (v_j, g_j), respectively, then $|g_i - g_j| \leq 1$. Moreover, if $g_i \neq 0$, $v_i = v_j$. This implies consistency.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
    Binary Graded Consensus, **instances** $\mathcal{GC}_1$, $\mathcal{GC}_2$ ▷ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
    Integer $g_i \leftarrow 0$ ▷ Grade
upon propose($v_i \in$ Value):
    invoke $\mathcal{GC}_1$.propose(v_i) ▷ Propose input value to the 1st graded consensus instance
    upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▷ Received decision from the 1st graded consensus instance
        $g_i \leftarrow g_i + g_i^1$ ▷ Update the grade (confidence)
        invoke $\mathcal{GC}_2$.propose(v_i^1) ▷ Propose input value to the 2nd graded consensus instance
    upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▷ Received decision from the 2nd graded consensus instance
        $g_i \leftarrow g_i + g_i^2$ ▷ Update the grade (confidence)
        trigger decide(v_i^2, g_i) ▷ Decide the final value and grade

$$\boxed{Alg_2} = \boxed{GC^{R=2}} \rhd \boxed{GC^{R=2}} \sqsubseteq \boxed{GC^{R=3}}$$

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

Uses:
    Binary Graded Consensus, **instances** $\mathcal{GC}_1, \mathcal{GC}_2$ ▹ 2 instances of the Binary Graded Consensus protocol with Refinement $R' = 2$
Local Variables:
    Integer $g_i \leftarrow 0$ ▹ Grade
upon propose($v_i \in$ Value):
    invoke $\mathcal{GC}_1$.propose(v_i) ▹ Propose input value to the 1st graded consensus instance
    upon $\mathcal{GC}_1$.decide(v_i^1, g_i^1): ▹ Received decision from the 1st graded consensus instance
        $g_i \leftarrow g_i + g_i^1$ ▹ Update the grade (confidence)
        invoke $\mathcal{GC}_2$.propose(v_i^1) ▹ Propose input value to the 2nd graded consensus instance
    upon $\mathcal{GC}_2$.decide(v_i^2, g_i^2): ▹ Received decision from the 2nd graded consensus instance
        $g_i \leftarrow g_i + g_i^2$ ▹ Update the grade (confidence)
        trigger decide(v_i^2, g_i) ▹ Decide the final value and grade

$$\boxed{Alg_2} = \boxed{GC^{R=2}} \rhd \boxed{GC^{R=2}} \sqsubseteq \boxed{GC^{R=3}}$$

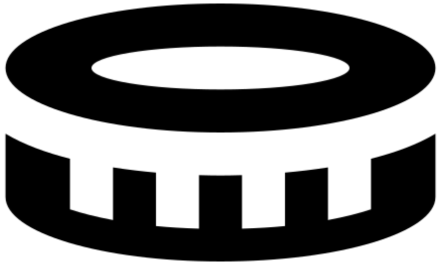
$$\boxed{Alg_2} \sqsubseteq \boxed{GC^{R=3}}$$

Common Coin

Interface of the common coin

Flip
Operation

Yield
Callback



Interface of the common coin

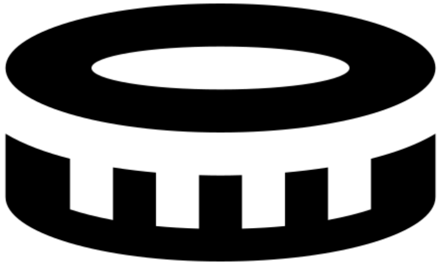
Flip

Operation

$\emptyset$

Yield

Callback



Interface of the common coin

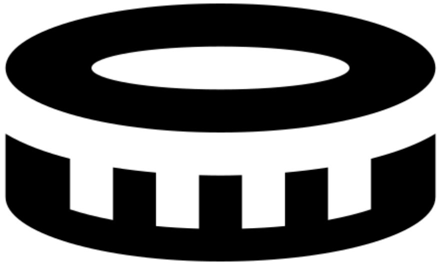
Flip

Operation

$\emptyset$

Yield

Callback



$v!$

Properties of the Common Coin

- **Termination:** All correct processes eventually yield a binary value.
- **Agreement:** All correct processes agree on 0 (or 1) with probability >0.
- **Unpredictability:** As soon no correct process has triggered ‘flip’, the adversary cannot predict the output with probability greater than 1/2.

A naive implementation of the Common Coin

Algorithm 3 Common Coin

1: **upon** flip():
2: $\quad b \xleftarrow{\$} \{0, 1\}$ ▷ Choose either 0 or 1 with probability 1/2
3: $\quad$ **trigger** yield(b)

A naive implementation of the Common Coin

Algorithm 3 Common Coin

1: **upon** flip():
2: $b \xleftarrow{\$} \{0, 1\}$ ▷ Choose either 0 or 1 with probability 1/2
3: **trigger** yield(b)

$$\boxed{Alg_3} \sqsubseteq \boxed{CC}$$

Consensus = Stay safe + Try (and try again)

$$\boxed{\boxed{GC^{R=3}} \longrightarrow \boxed{CC} \longrightarrow \cdots \longrightarrow \boxed{GC^{R=3}} \longrightarrow \boxed{CC} \longrightarrow \cdots}$$

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

Uses:
 Binary "Extended" Graded Consensus, **instances** $\mathcal{EGC}_1, \mathcal{EGC}_2, \ldots$ ▹ ∞ instances of the Binary Graded Consensus with Refinement $R = 3$

Constants:
 Integer $g_{min} \leftarrow 0$
 Integer $g_{max} \leftarrow 2$

Local Variables:
 Binary_Value $est_i \leftarrow 0$ ▹ Estimate Value
 Integer $g_i \leftarrow g_{min}$ ▹ Grade (Confidence) in $\{0, 1, 2\}$
 Integer $attempt \leftarrow 0$
 Integer $halt \leftarrow \infty$
 Boolean $decided \leftarrow false$

upon propose($v_i \in$ Value):
 $est_i \leftarrow v_i$:
 while $halt \geq attempt$:
  //safety guard
  $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▹ Execute instance of extended graded consensus
  if [redacted]
   trigger decide(est_i) ▹ Decide
   $decided \leftarrow true$
   $halt \leftarrow attempt + 1$ ▹ Halt after the next attempt after having helped the remaining processes to decide
  //try to converge
  $b_i \leftarrow CC_{attempt}$.flip() ▹ Execute instance of common coin
  if [redacted]
   $est_i \leftarrow b_i$
  $attempt \leftarrow attempt + 1$

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

Uses:
  Binary "Extended" Graded Consensus, **instances** $\mathcal{EGC}_1, \mathcal{EGC}_2, \ldots$ ▹ ∞ instances of the Binary Graded Consensus with Refinement $R = 3$
Constants:
  Integer $g_{min} \leftarrow 0$
  Integer $g_{max} \leftarrow 2$
Local Variables:
  Binary_Value $est_i \leftarrow 0$ ▹ Estimate Value
  Integer $g_i \leftarrow g_{min}$ ▹ Grade (Confidence) in $\{0, 1, 2\}$
  Integer $attempt \leftarrow 0$
  Integer $halt \leftarrow \infty$
  Boolean $decided \leftarrow false$
upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▹ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▹ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▹ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▹ Execute instance of common coin
    if [redacted]
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

Uses:
  Binary "Extended" Graded Consensus, **instances** $\mathcal{EGC}_1, \mathcal{EGC}_2, \ldots$ ▷ ∞ instances of the Binary Graded Consensus with Refinement $R = 3$

Constants:
  Integer $g_{min} \leftarrow 0$
  Integer $g_{max} \leftarrow 2$

Local Variables:
  Binary_Value $est_i \leftarrow 0$ ▷ Estimate Value
  Integer $g_i \leftarrow g_{min}$ ▷ Grade (Confidence) in $\{0, 1, 2\}$
  Integer $attempt \leftarrow 0$
  Integer $halt \leftarrow \infty$
  Boolean $decided \leftarrow false$

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow \mathcal{CC}_{attempt}$.flip() ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
    $est_i \leftarrow v_i$:
    while $halt \geq attempt$:
        //safety guard
        $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▹ Execute instance of extended graded consensus
        if $g_i == g_{max} \wedge decided = false$:
            trigger decide(est_i) ▹ Decide
            $decided \leftarrow true$
            $halt \leftarrow attempt + 1$ ▹ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        $b_i \leftarrow CC_{attempt}$.flip() ▹ Execute instance of common coin
        if $g_i == g_{min}$:
            $est_i \leftarrow b_i$
        $attempt \leftarrow attempt + 1$

LEMMA 1.3. *If all correct processes begin attempt k with the same estimate value v, they will all decide on v by attempt k and halt by attempt $k + 1$.*

PROOF.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \boxed{\mathcal{EGC}_{attempt}.\text{propose}(est_i)}$ ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}.\text{flip}()$ ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

Lemma 1.3. *If all correct processes begin attempt k with the same estimate value v, they will all decide on v by attempt k and halt by attempt $k + 1$.*

Proof. By the unanimity property of $\mathcal{EGC}_k$,

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
    $est_i \leftarrow v_i$:
    while $halt \geq attempt$:
        //safety guard
        $\boxed{(est_i, g_i)} \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
        if $\boxed{g_i == g_{max}} \wedge decided = false$:
            trigger decide(est_i) ▷ Decide
            $decided \leftarrow true$
            $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
        if $g_i == g_{min}$:
            $est_i \leftarrow b_i$
        $attempt \leftarrow attempt + 1$

LEMMA 1.3. *If all correct processes begin attempt k with the same estimate value v, they will all decide on v by attempt k and halt by attempt $k + 1$.*

PROOF. By the unanimity property of $\mathcal{EGC}_k$, all correct processes return (v, g_{max}) from $\mathcal{EGC}_k$.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    if $\boxed{g_i == g_{max} \wedge decided = false}$:
      $\boxed{\textbf{trigger}\ \text{decide}(est_i)}$ ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

LEMMA 1.3. *If all correct processes begin attempt k with the same estimate value v, they will all decide on v by attempt k and halt by attempt $k + 1$.*

PROOF. By the unanimity property of $\mathcal{EGC}_k$, all correct processes return (v, g_{max}) from $\mathcal{EGC}_k$. The rest follows directly from the protocol. □

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      $\boxed{\textbf{trigger}\ \text{decide}(est_i)}$ ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

Lemma 1.4. *If a correct process decides on v in attempt k, then all correct processes will decide on v by attempt $k + 1$.*

Proof. Let p_i be the first correct process to decide, and assume it decides on v at attempt k.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $\boxed{(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}.\text{propose}(est_i)}$ ▷ Execute instance of extended graded consensus
    if $\boxed{g_i == g_{max}} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}.\text{flip}()$ ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

Lemma 1.4. *If a correct process decides on v in attempt k, then all correct processes will decide on v by attempt $k + 1$.*

Proof. Let p_i be the first correct process to decide, and assume it decides on v at attempt k. This implies that p_i returned (v, g_{max}) from $\mathcal{EGC}_k$.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $\boxed{(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}.\text{propose}(est_i)}$ ▷ Execute instance of extended graded consensus
    if $\boxed{g_i == g_{max}} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}.\text{flip}()$ ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

Lemma 1.4. *If a correct process decides on v in attempt k, then all correct processes will decide on v by attempt $k + 1$.*

Proof. Let p_i be the first correct process to decide, and assume it decides on v at attempt k. This implies that p_i returned (v, g_{max}) from $\mathcal{EGC}_k$. By the consistency property of $\mathcal{EGC}_k$, every correct process p_j returns $(v, g_j \in \{1, 2\})$ from $\mathcal{EGC}_k$

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
    $est_i \leftarrow v_i$:
    while $halt \geq attempt$:
        //safety guard
        $\boxed{(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}.\text{propose}(est_i)}$ ▹ Execute instance of extended graded consensus
        if $\boxed{g_i == g_{max}} \wedge decided = false$:
            trigger decide(est_i) ▹ Decide
            $decided \leftarrow true$
            $halt \leftarrow attempt + 1$ ▹ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        $b_i \leftarrow CC_{attempt}.\text{flip}()$ ▹ Execute instance of common coin
        if $g_i == g_{min}$:
            $est_i \leftarrow b_i$
        $attempt \leftarrow attempt + 1$

LEMMA 1.4. *If a correct process decides on v in attempt k, then all correct processes will decide on v by attempt $k+1$.*

PROOF. Let p_i be the first correct process to decide, and assume it decides on v at attempt k. This implies that p_i returned (v, g_{max}) from $\mathcal{EGC}_k$. By the consistency property of $\mathcal{EGC}_k$, every correct process p_j returns $(v, g_j \in \{1,2\})$ from $\mathcal{EGC}_k$, updating its estimate est_j to v.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

```
upon propose(v_i ∈ Value):
    est_i ← v_i:
    while halt ≥ attempt:
        //safety guard
        (est_i, g_i) ← EGC_attempt.propose(est_i)          ▷ Execute instance of extended graded consensus
        if g_i == g_max ∧ decided = false:
            trigger decide(est_i)                           ▷ Decide
            decided ← true
            halt ← attempt + 1                              ▷ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        b_i ← CC_attempt.flip()                             ▷ Execute instance of common coin
        if g_i == g_min:
            est_i ← b_i
        attempt ← attempt + 1
```

Lemma 1.4. *If a correct process decides on v in attempt k, then all correct processes will decide on v by attempt $k+1$.*

Proof. Let p_i be the first correct process to decide, and assume it decides on v at attempt k. This implies that p_i returned (v, g_{max}) from $\mathcal{EGC}_k$. By the consistency property of $\mathcal{EGC}_k$, every correct process p_j returns $(v, g_j \in \{1, 2\})$ from $\mathcal{EGC}_k$, updating its estimate est_j to v. Thus, $g_j > g_{min}$,

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
    $\boxed{\textbf{if } g_i == g_{min}}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

LEMMA 1.4. *If a correct process decides on v in attempt k, then all correct processes will decide on v by attempt $k+1$.*

PROOF. Let p_i be the first correct process to decide, and assume it decides on v at attempt k. This implies that p_i returned (v, g_{max}) from $\mathcal{EGC}_k$. By the consistency property of $\mathcal{EGC}_k$, every correct process p_j returns $(v, g_j \in \{1, 2\})$ from $\mathcal{EGC}_k$, updating its estimate est_j to v. Thus, $g_j > g_{min}$, so all correct processes ignore the output of CC_k and retain $est_j = v$.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
    $est_i \leftarrow v_i$:
    while $halt \geq attempt$:
        //safety guard
        $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
        if $g_i == g_{max} \land decided = false$:
            trigger decide(est_i) ▷ Decide
            $decided \leftarrow true$
            $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
        if $g_i == g_{min}$:
            $est_i \leftarrow b_i$
        $attempt \leftarrow attempt + 1$

Lemma 1.4. *If a correct process decides on v in attempt k, then all correct processes will decide on v by attempt $k+1$.*

Proof. Let p_i be the first correct process to decide, and assume it decides on v at attempt k. This implies that p_i returned (v, g_{max}) from $\mathcal{EGC}_k$. By the consistency property of $\mathcal{EGC}_k$, every correct process p_j returns $(v, g_j \in \{1, 2\})$ from $\mathcal{EGC}_k$, updating its estimate est_j to v. Thus, $g_j > g_{min}$, so all correct processes ignore the output of CC_k and retain $est_j = v$. Consequently, all correct processes begin attempt $k+1$ with estimate value v. □

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \land decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

Lemma 1.4. *If a correct process decides on v in attempt k, then all correct processes will decide on v by attempt $k + 1$.*

Proof. Let p_i be the first correct process to decide, and assume it decides on v at attempt k. This implies that p_i returned (v, g_{max}) from $\mathcal{EGC}_k$. By the consistency property of $\mathcal{EGC}_k$, every correct process p_j returns $(v, g_j \in \{1, 2\})$ from $\mathcal{EGC}_k$, updating its estimate est_j to v. Thus, $g_j > g_{min}$, so all correct processes ignore the output of CC_k and retain $est_j = v$. Consequently, all correct processes begin attempt $k + 1$ with estimate value v. Lemma 1.3 then completes the proof. □

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

```
upon propose(v_i ∈ Value):
    est_i ← v_i:
    while halt ≥ attempt:
        //safety guard
        (est_i, g_i) ← EGC_attempt.propose(est_i)        ▷ Execute instance of extended graded consensus
        if g_i == g_max ∧ decided = false:
            trigger decide(est_i)                       ▷ Decide
            decided ← true
            halt ← attempt + 1                          ▷ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        b_i ← CC_attempt.flip()                         ▷ Execute instance of common coin
        if g_i == g_min:
            est_i ← b_i
        attempt ← attempt + 1
```

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

PROOF.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
    $est_i \leftarrow v_i$:
    while $halt \geq attempt$:
        //safety guard
        $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$)   ▷ Execute instance of extended graded consensus
        if $g_i == g_{max} \wedge decided = false$:
            trigger decide(est_i)   ▷ Decide
            $decided \leftarrow true$
            $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        $b_i \leftarrow CC_{attempt}$.flip()   ▷ Execute instance of common coin
        if $g_i == g_{min}$:
            $est_i \leftarrow b_i$
        $attempt \leftarrow attempt + 1$

Theorem 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

Proof. **Validity** follows from

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

PROOF. **Validity** follows from Lemma 1.3,

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
    $est_i \leftarrow v_i$:
    while $halt \geq attempt$:
        //safety guard
        $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
        if $g_i == g_{max} \wedge decided = false$:
            trigger decide(est_i) ▷ Decide
            $decided \leftarrow true$
            $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
        if $g_i == g_{min}$:
            $est_i \leftarrow b_i$
        $attempt \leftarrow attempt + 1$

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

PROOF. **Validity** follows from Lemma 1.3, and **Agreement** follows from

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

```
upon propose(v_i ∈ Value):
    est_i ← v_i:
    while halt ≥ attempt:
        //safety guard
        (est_i, g_i) ← EGC_attempt.propose(est_i)      ▹ Execute instance of extended graded consensus
        if g_i == g_max ∧ decided = false:
            trigger decide(est_i)                      ▹ Decide
            decided ← true
            halt ← attempt + 1                         ▹ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        b_i ← CC_attempt.flip()                        ▹ Execute instance of common coin
        if g_i == g_min:
            est_i ← b_i
        attempt ← attempt + 1
```

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability 1 and has the same resiliency as the underlying graded consensus object.*

PROOF. **Validity** follows from Lemma 1.3, and **Agreement** follows from Lemma 1.4.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

```
upon propose(v_i ∈ Value):
    est_i ← v_i:
    while halt ≥ attempt:
        //safety guard
        (est_i, g_i) ← EGC_attempt.propose(est_i)          ▹ Execute instance of extended graded consensus
        if g_i == g_max ∧ decided = false:
            trigger decide(est_i)                          ▹ Decide
            decided ← true
            halt ← attempt + 1                             ▹ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        b_i ← CC_attempt.flip()                            ▹ Execute instance of common coin
        if g_i == g_min:
            est_i ← b_i
        attempt ← attempt + 1
```

Theorem 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

Proof. **Validity** follows from Lemma 1.3, and **Agreement** follows from Lemma 1.4. We now prove **Termination**.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▹ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▹ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▹ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▹ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

PROOF. **Validity** follows from Lemma 1.3, and **Agreement** follows from Lemma 1.4. We now prove **Termination**. Let p_i be the first correct process calling CC_k for some attempt k.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

```
upon propose(v_i ∈ Value):
    est_i ← v_i:
    while halt ≥ attempt:
        //safety guard
        (est_i, g_i) ← EGC_attempt.propose(est_i)      ▷ Execute instance of extended graded consensus
        if g_i == g_max ∧ decided = false:
            trigger decide(est_i)                      ▷ Decide
            decided ← true
            halt ← attempt + 1                         ▷ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        b_i ← CC_attempt.flip()                        ▷ Execute instance of common coin
        if g_i == g_min:
            est_i ← b_i
        attempt ← attempt + 1
```

(b, g_i)!

Theorem 1.5. *Algorithm 4 implements binary Byzantine consensus with probability 1 and has the same resiliency as the underlying graded consensus object.*

Proof. **Validity** follows from Lemma 1.3, and **Agreement** follows from Lemma 1.4. We now prove **Termination**. Let p_i be the first correct process calling CC_k for some attempt k. Let (b, g_i) the pair returned by p_i from $\mathcal{EGC}_k$.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$;
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$)   ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i)   ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $\boxed{b_i} \leftarrow CC_{attempt}$.flip()   ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

$(\boxed{b}, g_i)$!

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

PROOF. **Validity** follows from Lemma 1.3, and **Agreement** follows from Lemma 1.4. We now prove **Termination**. Let p_i be the first correct process calling CC_k for some attempt k. Let (b, g_i) the pair returned by p_i from $\mathcal{EGC}_k$. With non-zero probability ρ, all correct processes return b from CC_k.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
 $est_i \leftarrow v_i$;
 while $halt \geq attempt$:
  //safety guard
  $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($\boxed{est_i}$) ▷ Execute instance of extended graded consensus
  $(\boxed{b}, \boxed{g_i})$!
  if $g_i == g_{max} \wedge decided = false$:
   trigger decide(est_i) ▷ Decide
   $decided \leftarrow true$
   $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
  //try to converge
  $\boxed{b_i} \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
  if $g_i == g_{min}$:
   $est_i \leftarrow b_i$
  $attempt \leftarrow attempt + 1$

Theorem 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

Proof. **Validity** follows from Lemma 1.3, and **Agreement** follows from Lemma 1.4. We now prove **Termination**. Let p_i be the first correct process calling CC_k for some attempt k. Let (b, g_i) the pair returned by p_i from $\mathcal{EGC}_k$. With non-zero probability ρ, all correct processes return b from CC_k. We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 1.3.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

```
upon propose(v_i ∈ Value):
    est_i ← v_i:
    while halt ≥ attempt:
        //safety guard
        (est_i, g_i) ← EGC_attempt.propose(est_i)        ▷ Execute instance of extended graded consensus
        if g_i == g_max ∧ decided = false:
            trigger decide(est_i)                          ▷ Decide
            decided ← true
            halt ← attempt + 1                             ▷ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        b_i ← CC_attempt.flip()                            ▷ Execute instance of common coin
        if g_i == g_min:
            est_i ← b_i
        attempt ← attempt + 1
```

(b, g_i)!

Theorem 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 1.3.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$;
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    $(b, g_i)!$
    $(b, \boxed{g_j > g_{min}})!$
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow \mathcal{CC}_{attempt}$.flip() ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability 1 and has the same resiliency as the underlying graded consensus object.*

We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 1.3.

- **Case** $\boxed{\exists p_j \in \text{Correct}, g_j > g_{min}}$:

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \boxed{\mathcal{EGC}_{attempt}}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow \mathcal{CC}_{attempt}$.flip() ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

(b, g_i)!

$(b, \boxed{g_j > g_{min}})$!

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 1.3.

- **Case** $\boxed{\exists p_j \in \text{Correct}, g_j > g_{min}}$: By $\mathcal{EGC}_k$'s consistency,

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$;
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

$(b, g_i)!$

$(b, g_j > g_{min})!$

$(b, *)!$

Theorem 1.5. *Algorithm 4 implements binary Byzantine consensus with probability 1 and has the same resiliency as the underlying graded consensus object.*

We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 1.3.

- **Case** $\exists p_j \in \text{Correct}, g_j > g_{min}$: By $\mathcal{EGC}_k$'s consistency, (a) all processes return $(b, \cdot)$ from $\mathcal{EGC}_k$,

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(\boxed{est_i, g_i}) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▹ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▹ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▹ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $\boxed{b_i} \leftarrow CC_{attempt}$.flip() ▹ Execute instance of common coin
    if $g_i == g_{min}$:
      $\boxed{est_i \leftarrow b_i}$
    $attempt \leftarrow attempt + 1$

$(\boxed{b,} g_i)!$

$(b, g_j > g_{min})!$

$(b, *)!$

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability 1 and has the same resiliency as the underlying graded consensus object.*

We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 1.3.

- **Case** $\boxed{\exists p_j \in \text{Correct}, g_j > g_{min}}$: By $\mathcal{EGC}_k$'s consistency, (a) all processes return $(b, \cdot)$ from $\mathcal{EGC}_k$, so they will all have the same estimate value at attempt $k+1$, either by adopting CC_k's output or by retaining the value from $\mathcal{EGC}_k$.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$;
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▹ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▹ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▹ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▹ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

(b, g_i)!

$(b, g_j > g_{min})$!

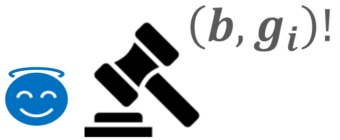
$(b, *)$!

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability 1 and has the same resiliency as the underlying graded consensus object.*

We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 1.3.

- **Case** $\exists p_j \in \text{Correct}, g_j > g_{min}$: By $\mathcal{EGC}_k$'s consistency, (a) all processes return $(b, \cdot)$ from $\mathcal{EGC}_k$, so they will all have the same estimate value at attempt $k+1$, either by adopting CC_k's output or by retaining the value from $\mathcal{EGC}_k$.

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$;
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus


    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide


    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin

    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

THEOREM 1.5. *Algorithm 4 implements binary Byzantine consensus with probability 1 and has the same resiliency as the underlying graded consensus object.*

We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 1.3.

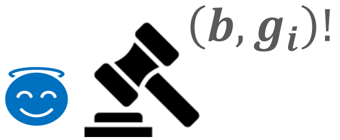
- **Case** $\exists p_j \in$ Correct, $g_j > g_{min}$: By $\mathcal{EGC}_k$'s consistency, (a) all processes return $(b, \cdot)$ from $\mathcal{EGC}_k$, so they will all have the same estimate value at attempt $k+1$, either by adopting CC_k's output or by retaining the value from $\mathcal{EGC}_k$.

- **Case** $\boxed{\forall p_j \in \text{Correct}, g_j = g_{min}}$:

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

---

upon propose($v_i \in$ Value):
    $est_i \leftarrow v_i$;
    while $halt \geq attempt$:
        //safety guard
        $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($\boxed{est_i}$) ▹ Execute instance of extended graded consensus



        if $g_i == g_{max} \wedge decided = false$:
            trigger decide(est_i) ▹ Decide
            $decided \leftarrow true$
            $halt \leftarrow attempt + 1$ ▹ Halt after the next attempt after having helped the remaining processes to decide

        //try to converge
        $\boxed{b_i} \leftarrow CC_{attempt}$.flip() ▹ Execute instance of common coin

        if $g_i == g_{min}$:
            $est_i \leftarrow b_i$
        $attempt \leftarrow attempt + 1$

Theorem 1.5. *Algorithm 4 implements binary Byzantine consensus with probability* 1 *and has the same resiliency as the underlying graded consensus object.*

We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 1.3.

- **Case** $\exists p_j \in \text{Correct}, g_j > g_{min}$: By $\mathcal{EGC}_k$'s consistency, (a) all processes return $(b, \cdot)$ from $\mathcal{EGC}_k$, so they will all have the same estimate value at attempt $k+1$, either by adopting CC_k's output or by retaining the value from $\mathcal{EGC}_k$.

- **Case** $\boxed{\forall p_j \in \text{Correct}, g_j = g_{min}}$: All processes adopt the value provided by the common coin, which is identical across processes. □

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

```
upon propose(v_i ∈ Value):
    est_i ← v_i:
    while halt ≥ attempt:
        //safety guard
        (est_i, g_i) ← EGC_attempt.propose(est_i)      ▹ Execute instance of extended graded consensus
        if g_i == g_max ∧ decided = false:
            trigger decide(est_i)                       ▹ Decide
            decided ← true
            halt ← attempt + 1                          ▹ Halt after the next attempt after having helped the remaining processes to decide
        //try to converge
        b_i ← CC_attempt.flip()                         ▹ Execute instance of common coin
        if g_i == g_min:
            est_i ← b_i
        attempt ← attempt + 1
```

$$\boxed{Alg_4} = \boxed{\boxed{GC^{R=3}} \longrightarrow \boxed{CC} \longrightarrow \dots \longrightarrow \boxed{GC^{R=3}} \longrightarrow \boxed{CC} \longrightarrow \dots} \sqsubseteq \boxed{BA}$$

Algorithm 4 Byzantine Agreement Protocol with Extended Graded Consensus

upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt \geq attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▹ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided = false$:
      trigger decide(est_i) ▹ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▹ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▹ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

$$\boxed{Alg_4} = \boxed{\boxed{GC^{R=3}} \longrightarrow \boxed{CC} \longrightarrow \dots \longrightarrow \boxed{GC^{R=3}} \longrightarrow \boxed{CC} \longrightarrow \dots} \sqsubseteq \boxed{BA}$$

$$\boxed{GC^{R=2}} \rhd \boxed{GC^{R=2}} \sqsubseteq \boxed{GC^{R=3}}$$

Positive Result 1: *There exists* a **randomized asynchronous** protocol that solves consensus, while tolerating arbitrary (**Byzantine**) failures

A general perspective

$$\boxed{Alg_4} = \boxed{\boxed{GC^{R=3}} \longrightarrow \boxed{CC} \longrightarrow \cdots \longrightarrow \boxed{GC^{R=3}} \longrightarrow \boxed{CC} \longrightarrow \cdots} \sqsubseteq \boxed{BA}$$

$$\boxed{Alg'} = \boxed{\boxed{GC^{R=3}} \longrightarrow \boxed{Try} \longrightarrow \cdots \longrightarrow \boxed{GC^{R=3}} \longrightarrow \boxed{Try} \longrightarrow \cdots} \sqsubseteq \boxed{BA}$$

Safety Guards

$$Alg' = \boxed{GC^{R=3} \longrightarrow Try \longrightarrow \dots \longrightarrow GC^{R=3} \longrightarrow Try \longrightarrow \dots} \sqsubseteq BA$$

Safety Guards

$$Alg' = \boxed{GC^{R=3} \longrightarrow Try \longrightarrow \dots \longrightarrow GC^{R=3} \longrightarrow Try \longrightarrow \dots} \sqsubseteq BA$$

Convergence under good circumstances

Safety Guards

$$\boxed{Alg'} = \boxed{\boxed{GC^{R=3}} \longrightarrow \boxed{Try} \longrightarrow \dots \longrightarrow \boxed{GC^{R=3}} \longrightarrow \boxed{Try} \longrightarrow \dots} \sqsubseteq \boxed{BA}$$

Convergence under good circumstances

Luckyness via a common coin

Safety Guards

$$Alg' = \left[\boxed{GC^{R=3}} \longrightarrow \boxed{Try} \longrightarrow \dots \longrightarrow \boxed{GC^{R=3}} \longrightarrow \boxed{Try} \longrightarrow \dots \right] \sqsubseteq \boxed{BA}$$

Convergence under good circumstances

Luckyness via a common coin

Eventual Synchrony + Synchronization

Safety Guards

$$\boxed{Alg'} = \boxed{\boxed{GC^{R=3}} \longrightarrow \boxed{Try} \longrightarrow \dots \longrightarrow \boxed{GC^{R=3}} \longrightarrow \boxed{Try} \longrightarrow \dots} \sqsubseteq \boxed{BA}$$

Convergence under good circumstances

- Luckyness via a common coin
- Eventual Synchrony + Synchronization
- Unreliable Failure Detectors

Safety Guards

$$Alg' = GC^{R=3} \longrightarrow Try \longrightarrow \ldots \longrightarrow GC^{R=3} \longrightarrow Try \longrightarrow \ldots \sqsubseteq BA$$

Convergence under good circumstances

- Luckyness via a common coin
- Eventual Synchrony + Synchronization
- Unreliable Failure Detectors
- Fair scheduling / Noisy Environement

Safety Guards

$$Alg' = \left(GC^{R=3} \longrightarrow Try \longrightarrow \dots \longrightarrow GC^{R=3} \longrightarrow Try \longrightarrow \dots \right) \sqsubseteq BA$$

Convergence under good circumstances

- Luckyness via a common coin
- Eventual Synchrony + Synchronization
- Unreliable Failure Detectors
- Fair scheduling / Noisy Environement
- Synchrony + round-robin rotating leader

$$Alg' = \boxed{GC^{R=3} \longrightarrow L_i \longrightarrow \cdots \longrightarrow GC^{R=3} \longrightarrow L_j \longrightarrow \cdots} \sqsubseteq BA$$

Correct Leader

Synchrony

Positive Result 2: *There exists* a **deterministic synchronous** protocol that solves consensus, while tolerating arbitrary (**Byzantine**) failures