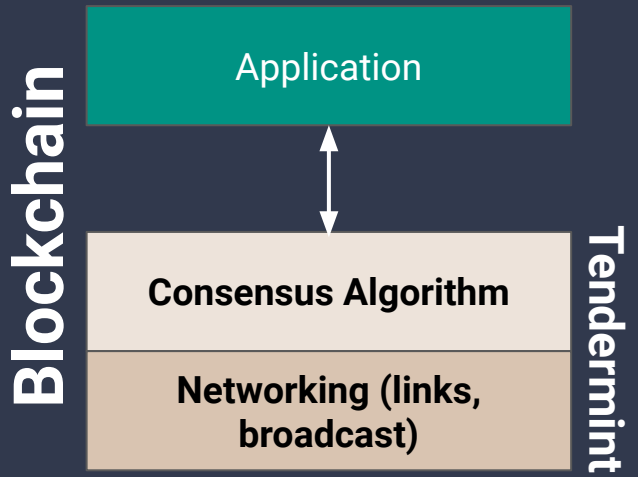
Lawfulness, Crime, & Punishment In Tendermint

Adi Seredinschi
Interchain Foundation

Distributed Algorithms // **EPFL Fall'19**

Consensus vs. Blockchain



Consensus: *"processes propose values and have to agree on one among these values"* (last week)

Models:

- *Benign: crash-stop* processes (P, <>P algorithms)
- **Today: *Byzantine* processes**
  - e.g., buggy, malicious & adversarial, rational
  - Authenticated links (dig. sigs. assumption)

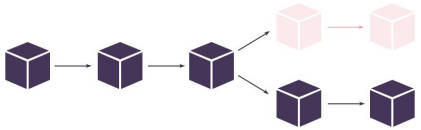
Blockchain

- Can mean different things
- Often, the whole stack is the "blockchain"
- Builds on a consensus core -> total order
  - Multiple instances of consensus

Roadmap

1. **Lawful BFT Consensus Algorithm**
   Basic Tendermint Consensus

2. **Crime & Punishment**
   Forks



3. **Novelties, Open & Projects**

Lawful Algorithm

Basic Tendermint BFT Consensus

Properties

Validity Predicate-based Byzantine Consensus (Crain et al, 2017)

1. **Validity**: A decided value is *valid*, i.e., it satisfies a predicate *valid()*.
2. **Agreement**: No two correct processes decide differently.
3. **Termination**: All correct processes eventually decide on a value.
4. **Integrity:** No correct process decides more than once (w.r.t. a consensus instance).

http://arxiv.org/abs/1807.04938

Tendermint Algorithm Overview

- Similar in spirit to Consensus algorithm III
- We assume a correct “majority”:
  - > ⅔ processes are correct (quorums)
  - < ⅓ processes may be Byzantine
  - N = 3f + 1
- Processes take turns in the role of **proposer**
  - **Round-based model**
  - Each round has a predefined proposer
  - **Goal:** proposer imposes a decision

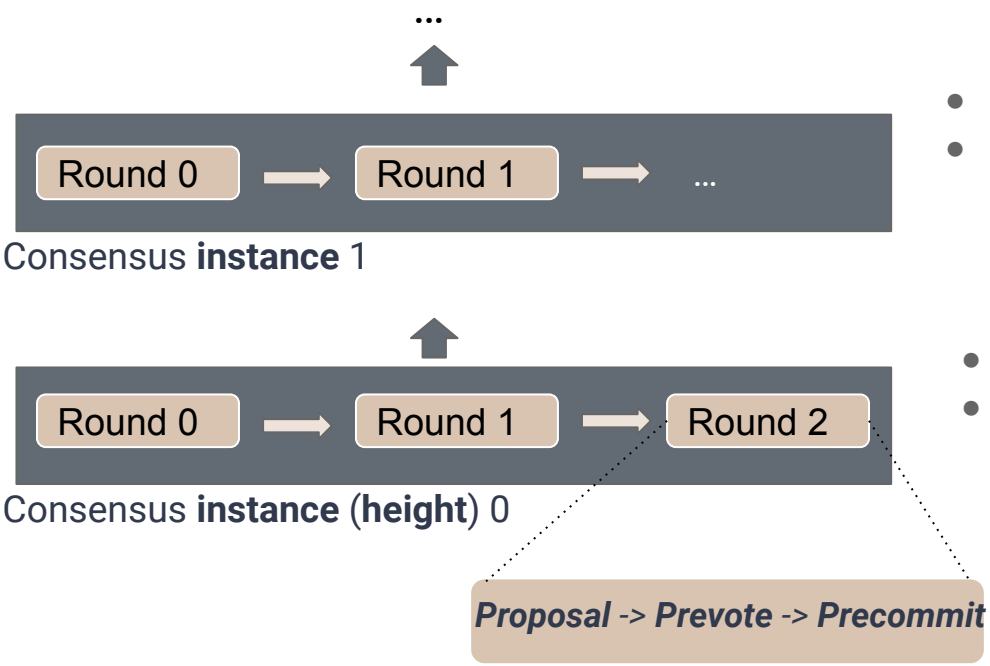
Consensus algorithm III

- A uniform consensus algorithm assuming:
  - a correct majority
  - a <>P failure detector

  - > ½ processes are correct
  - N = 2f + 1
  - Benign (non-Byzantine case)

https://arxiv.org/pdf/1807.04938.pdf

Rounds & Dances



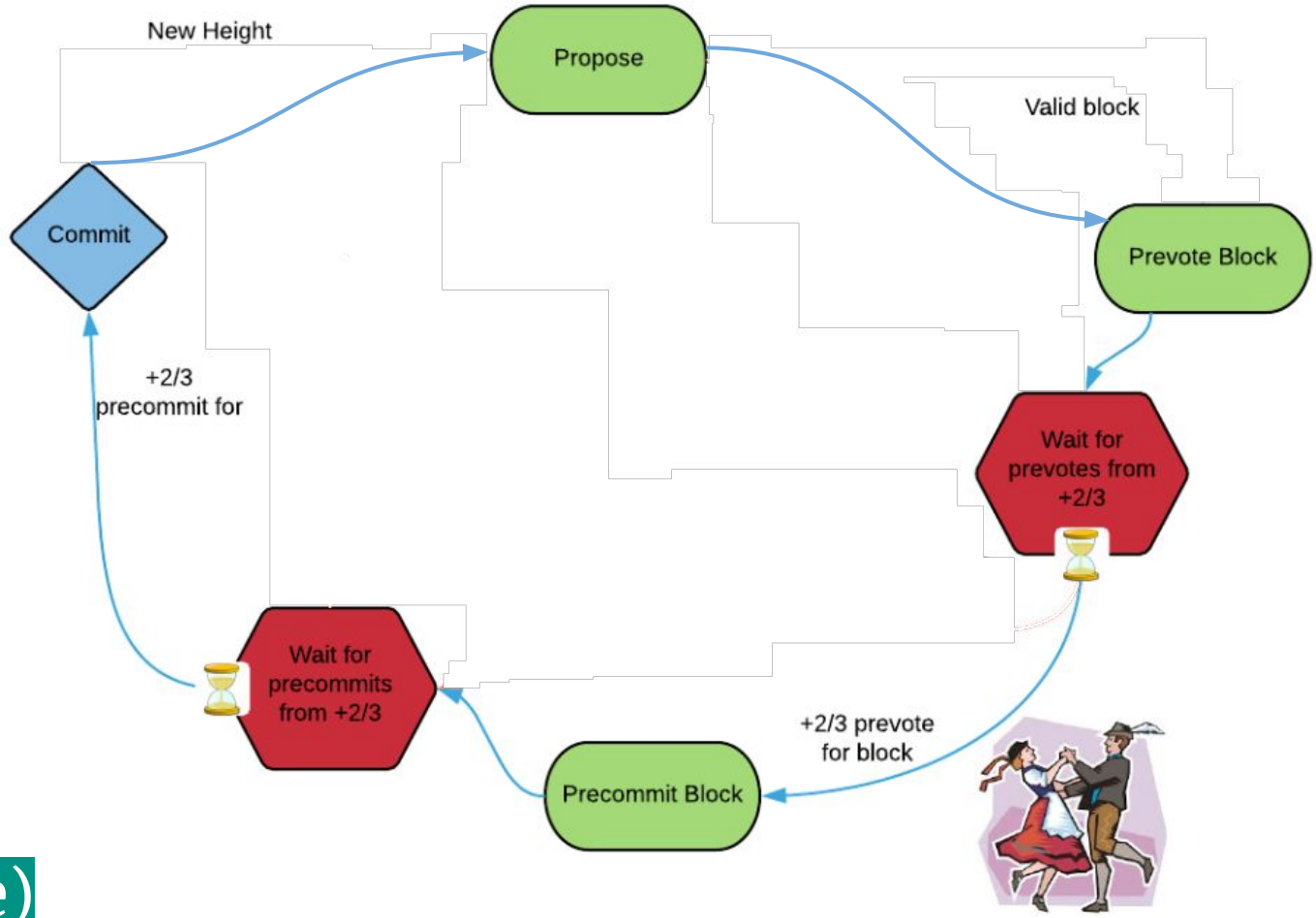
Round

- ***Proposal***, ***Prevote***, ***Precommit*** *-> decision*
- Has a predefined **proposer** process

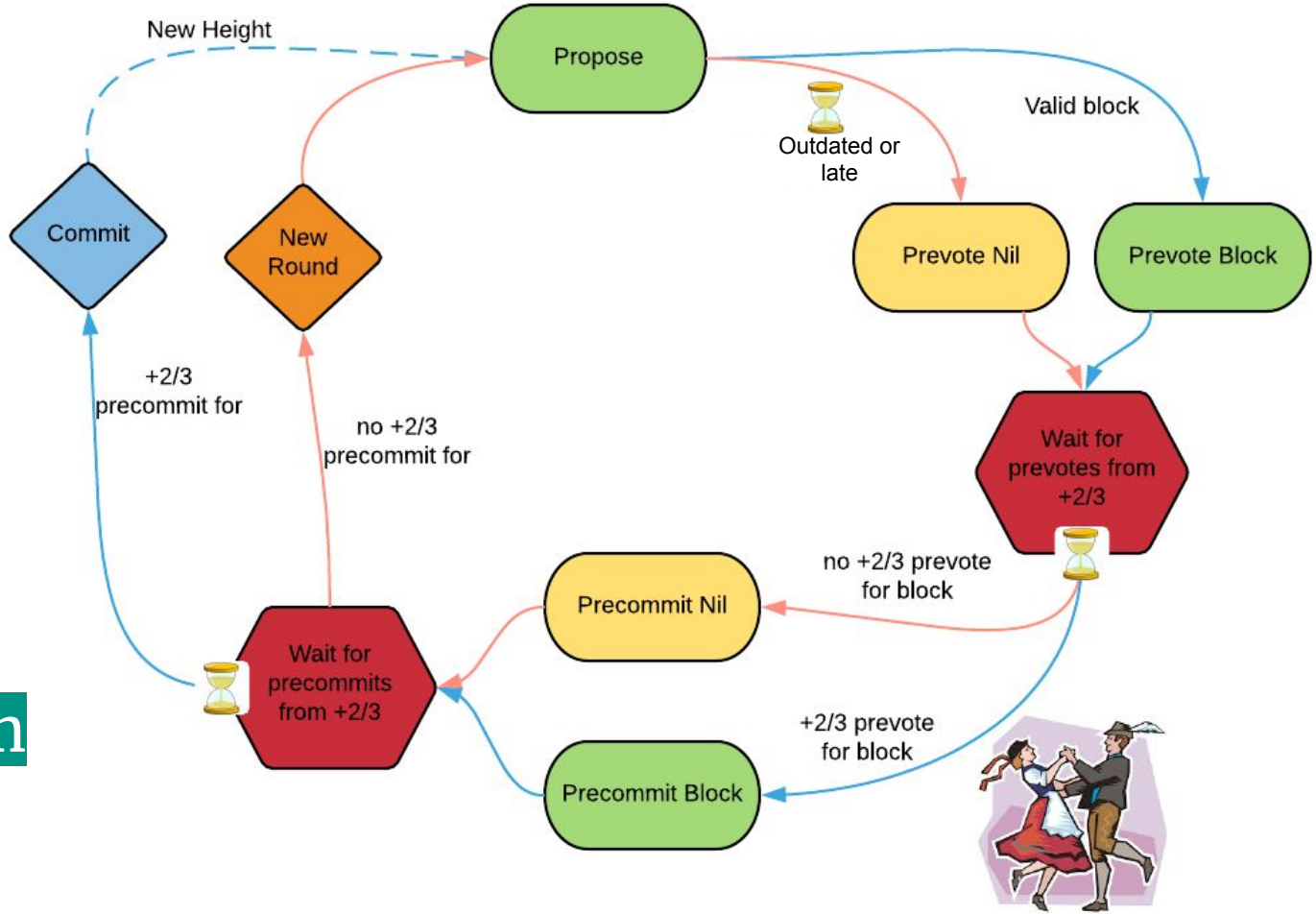
Locking: Doing the Polka

- **Locked values means PRECOMMIT was sent**
- Two variables keep track of the last locked value:
  - *lockedValue*
    - Retains the value itself; initially nil
  - *lockedRound,*
    - Initially -1

Lawful Algorithm Overview (good case)



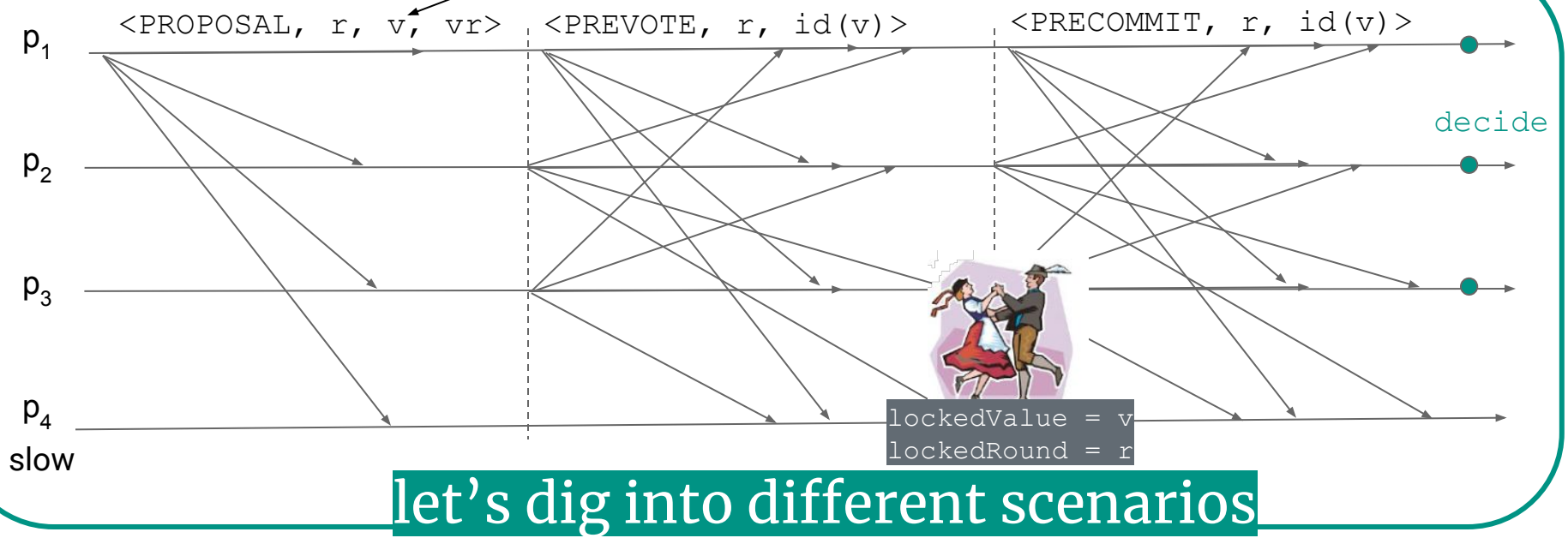
Lawful Algorithm Overview



Tendermint consensus algorithm

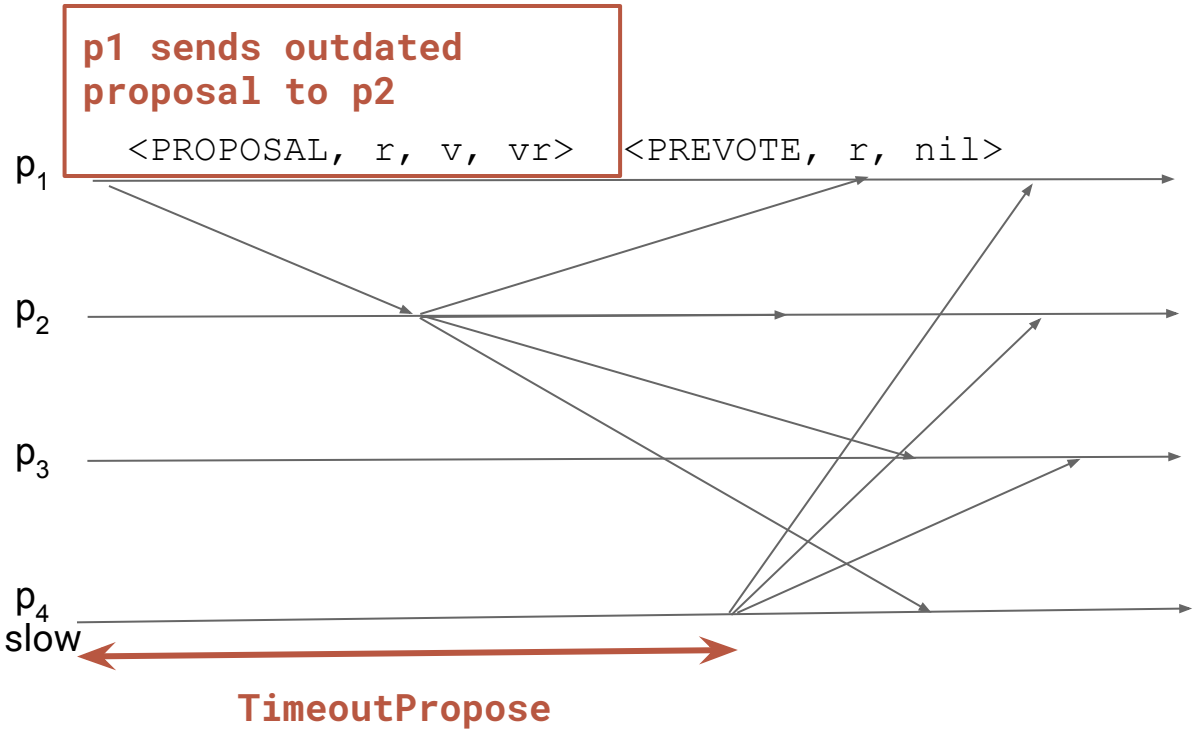
```
lockedValue = nil
lockedRound = -1
validValue = nil
validRound = -1
step = Propose
```

P1's round



let's dig into different scenarios

Propose Step -> Prevote Step

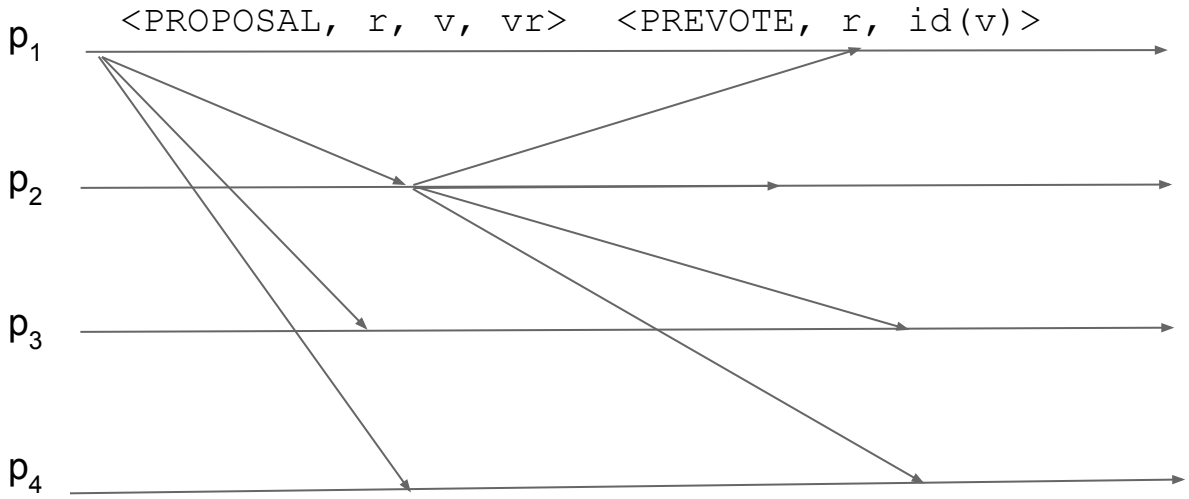


Reject Proposal at p2 if:

lockedRound >= vr and
lockedValue != v
OR

TimeoutPropose expired and no Proposal is received

Propose Step -> Prevote Step



p2 accepts Proposal if:

v is valid AND

lockedRound = -1

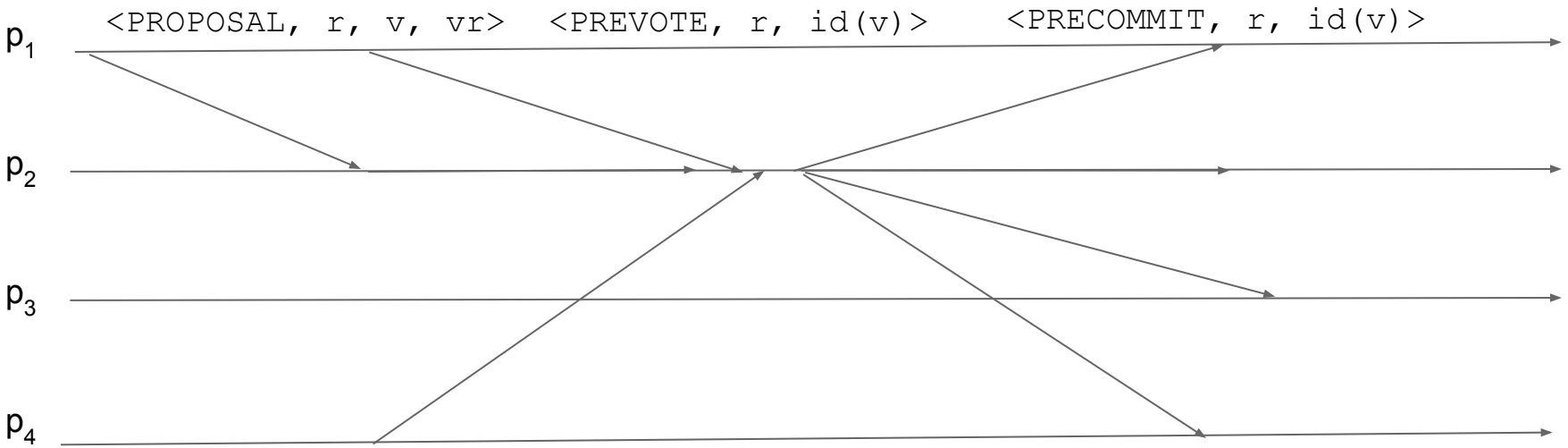
OR lockedValue = v

OR

Polka:
vr > lockedRound and
p2 delivered
2f+1 <PREVOTE, vr, id(v)>

Prepare Step -> Precommit Step

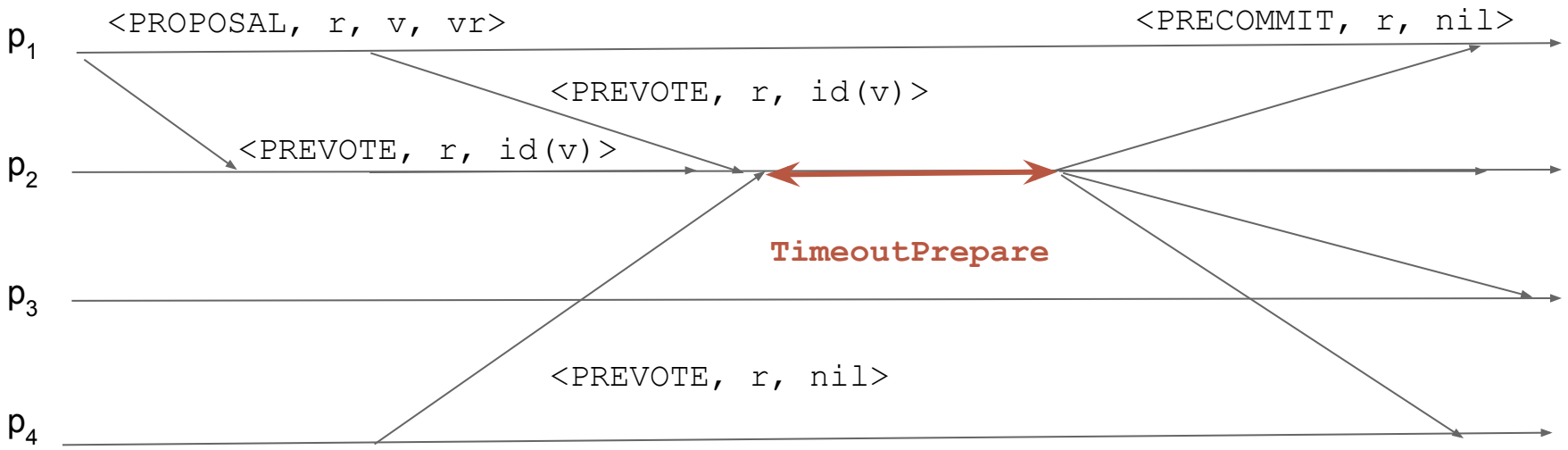
```
UPON <PROPOSAL, r, v, vr> AND
2f+1 <PREVOTE, r, id(v)>
     lockedValue = v; lockedRound = r
     validValue = v; validRound = r
     BROADCAST <PRECOMMIT, r, id(v)>
```



Prepare Step -> Precommit Step

```
UPON <PROPOSAL, r, v, vr> AND
any 2f+1 <PREVOTE, r, *>
    Trigger TimeoutPrepare
```

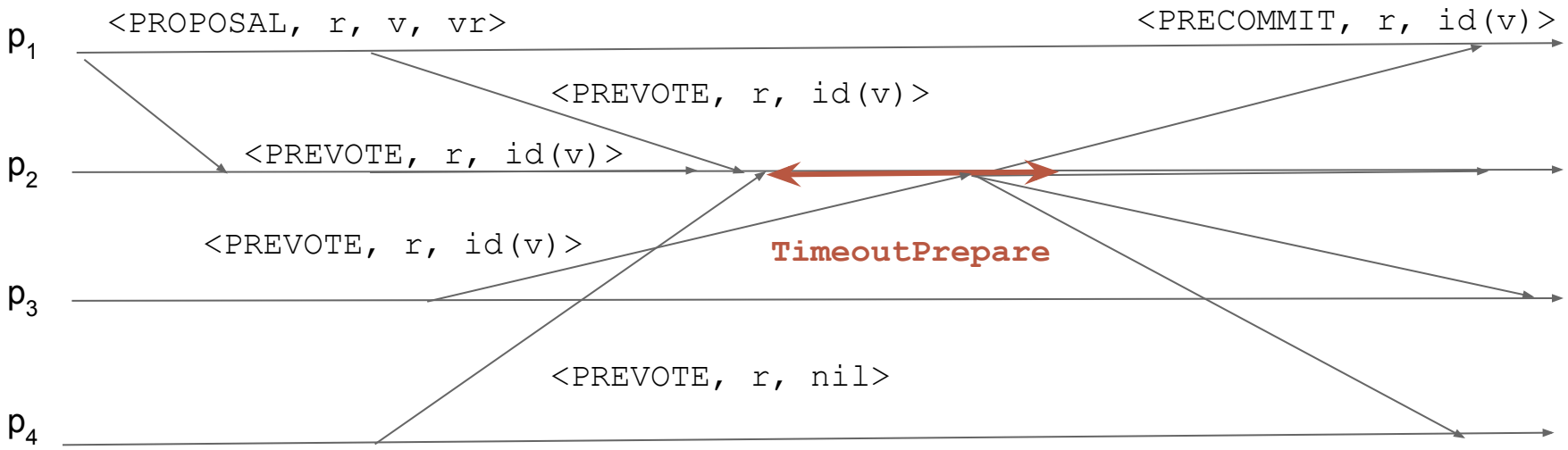
(any values)



Prepare Step -> Precommit Step

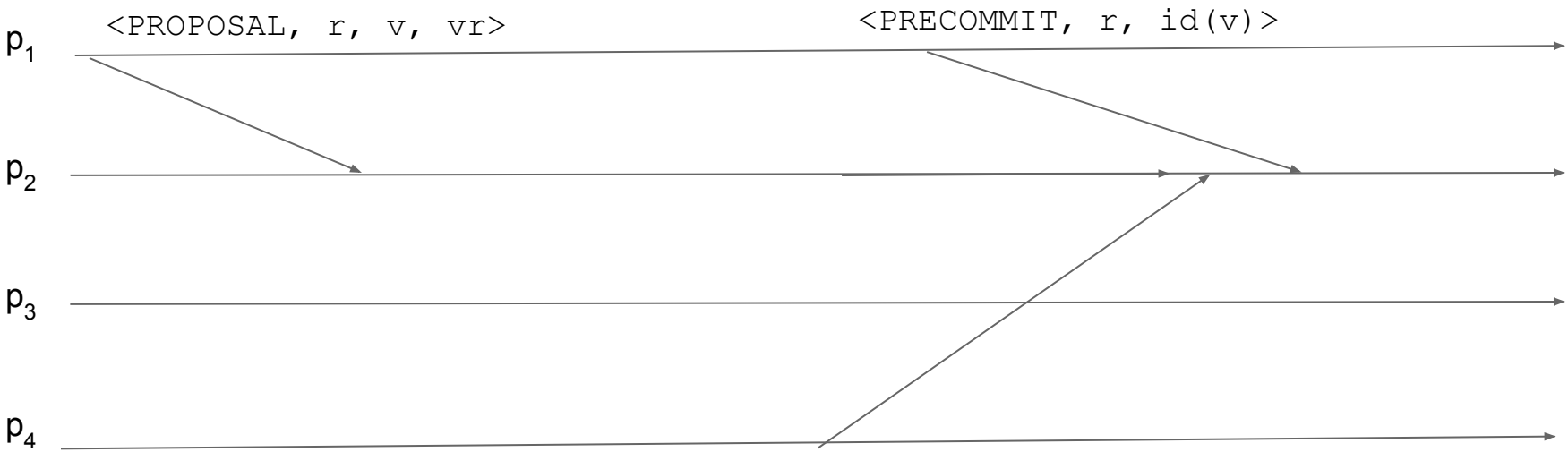
```
UPON <PROPOSAL, r, v, vr> AND
any 2f+1 <PREVOTE, r, *>
    Trigger TimeoutPrepare
```

(any values)



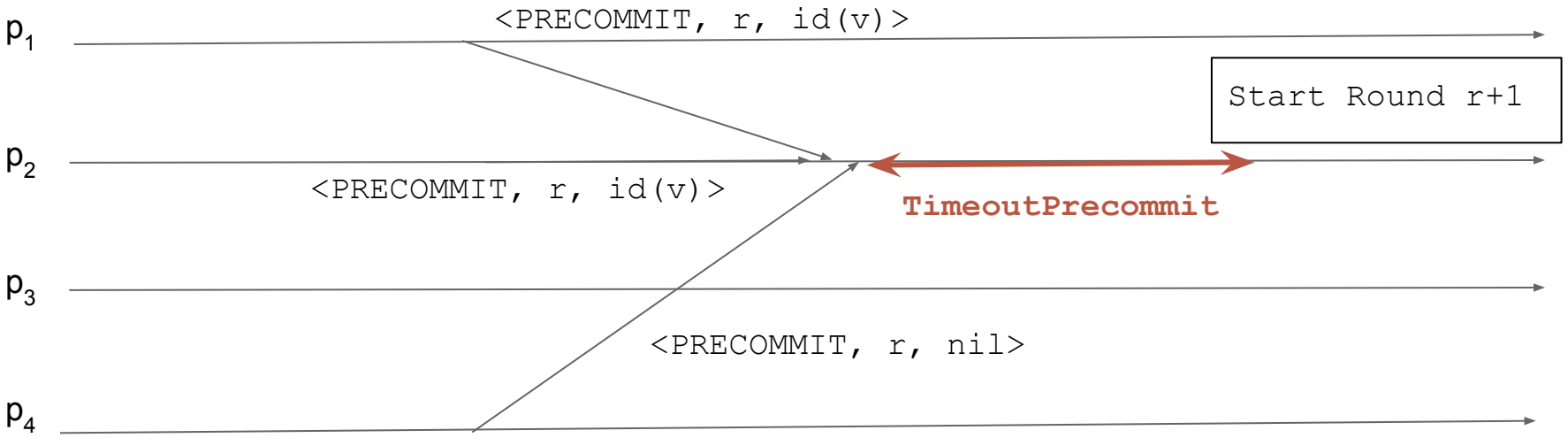
Decide v

```
UPON <PROPOSAL, r, v, vr> AND
2f+1 <PRECOMMIT, r, id(v)>
     DECIDE v
```

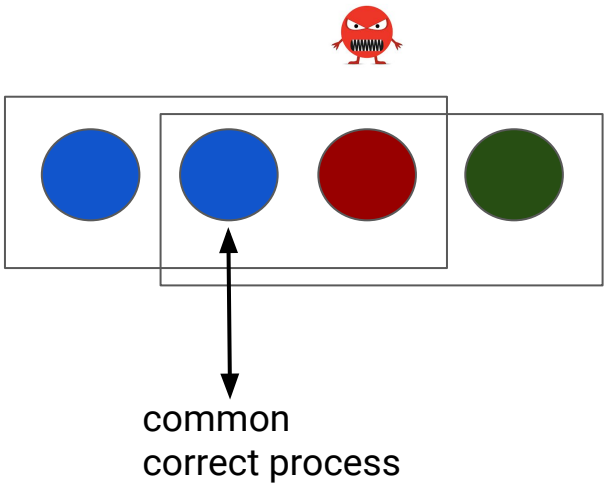


Precommit Wait Step -> Start Next Round

```
UPON any 2f+1 <PRECOMMIT, r, *>
    Trigger TimeoutPrecommit
```



Agreement (**Safety**) & Intersection



- Any two quorums of size 2f+1 have at least a correct process in common
- At most one value can be locked in a round
- If a correct process decided v in round r, only v can be locked in rounds r' >= r
- Only v can be decided in rounds r' >= r

impossible

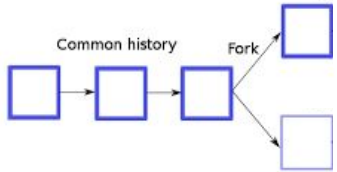
Question: What can happen if there are more than ⅓ faulty processes?

Crime & Punishment

Forks & fork-accountability

Fork-accountability

Identify & Punish Malicious Processes



- Upon a fork, every correct process proves its innocence by justifying its transitions (revealing its message log)
- Key trick is proving that in event of fork, we have enough information seen (and stored) by correct processes

Algorithm:

1. Processes send their vote sets to a verification entity (assumes synchrony, i.e., ***P***)
2. Cluster the vote sets by the senders (identify senders based on digital signatures).
3. Determine – for each process – if protocol transitions are valid or not, justifying the actions of correct processes and finding malicious processes.

https://github.com/tendermint/tendermint/blob/master/docs/spec/consensus/fork-accountability.md

Malicious Processes

Examples of malicious actions:

1. Sending more than one PREVOTE message in a round.
2. Sending more than one PRECOMMIT message in a round.
3. Sending PRECOMMIT message without receiving +2/3 matching PREVOTE messages
4. Sending PREVOTE message for the value V in round R but having lock on some other value V’

Punishment

– slash bonded stake (remove from process set)

Novelties

1. **Gossip layer** (instead of all-to-all links)
2. **Lite client** (e.g., a mobile phone)
3. **Robustness** (Jepsen tests)
4. **ABCI** – interface b/t consensus and application layer

Open Challenges

1. **Rust** implementation of consensus
2. **Formal verification** of consensus
3. **Interblockchain Communication Protocol – IBC**

Student Projects

- **AT2** ~ implementation of consensus-less payments
- **IBC** ~ a “TCP/IP” for interconnecting ledgers
- **Rust** ~ Implementation of Tendermint modules (consensus, mempool, fast sync) using Prusti and Rust.
- **Stainless** ~ Implementation of Tendermint modules (consensus, mempool) using Stainless and Scala.
- **Facebook Libra** ~ comparative analysis of consensus algorithms.
- Mempool (performance analysis); adversarial engineering.

Complete list:
https://dcl.epfl.ch/site/education#collaborative_projects

drop by anytime
INR 327
(Innovation Park soon)

adi@interchain.io

Acknowledgements & Collaborators



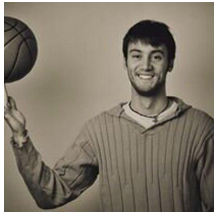












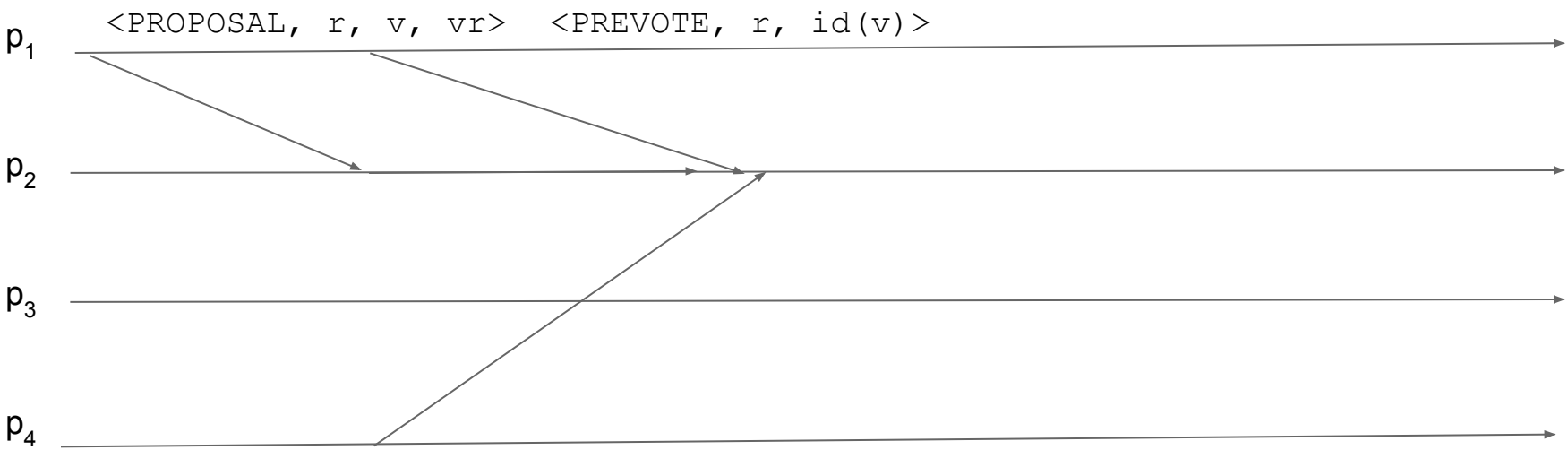
EPFL



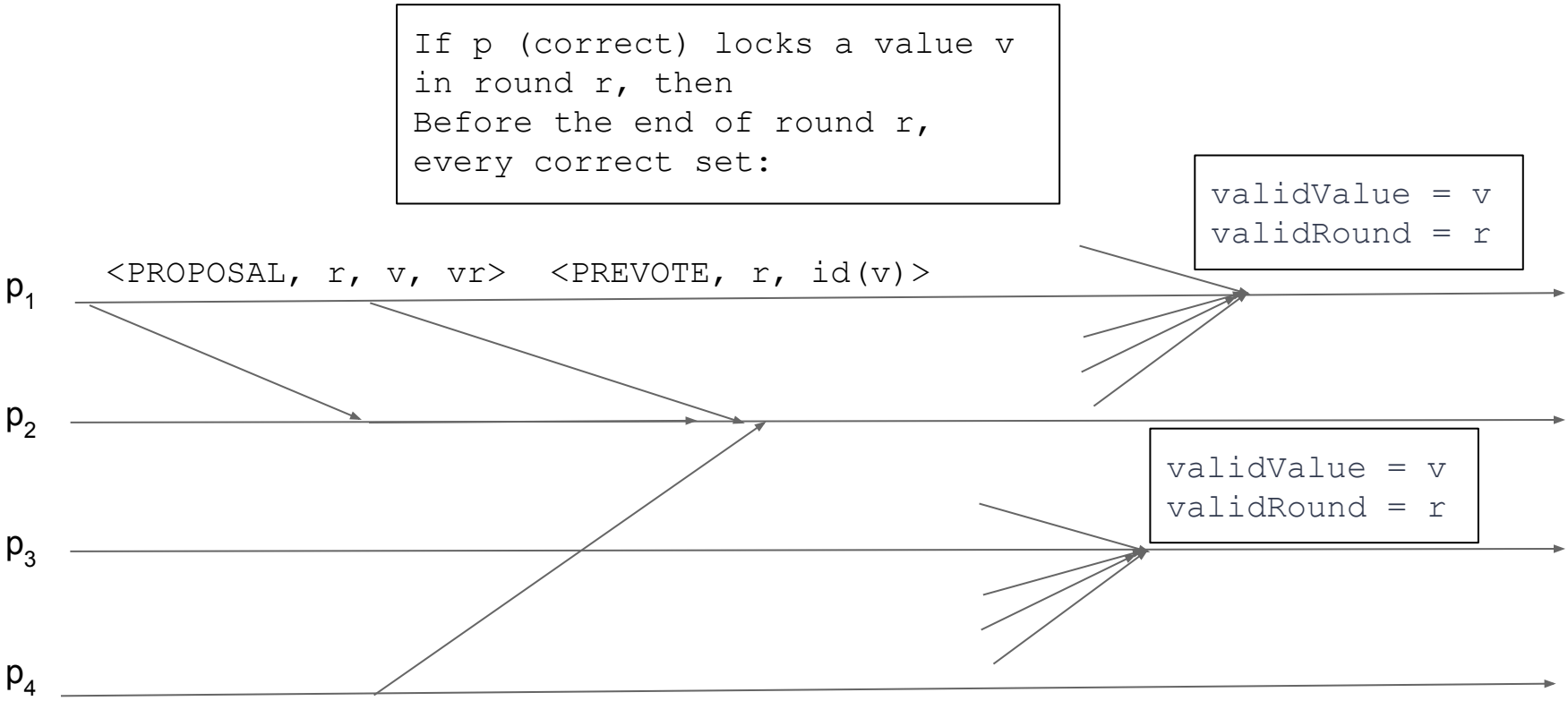
STANFORD UNIVERSITY
THE UNIVERSITY OF CALIFORNIA BERKELEY 1868
Inria

Update `validValue` while Step = any

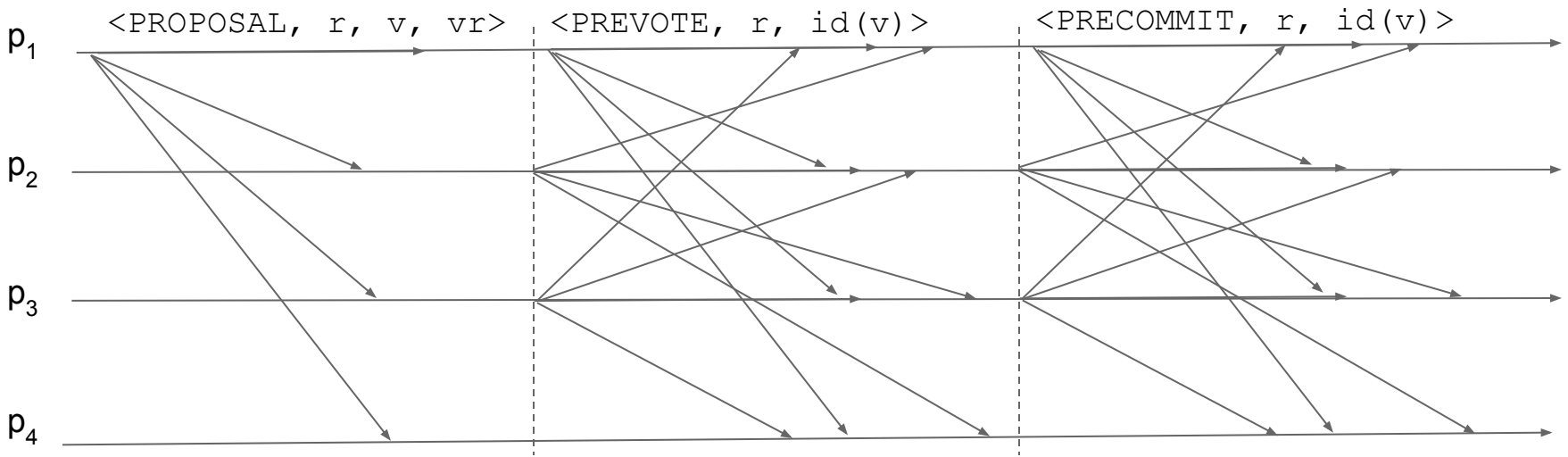
```
UPON <PROPOSAL, r, v, vr> AND
2f+1 <PREVOTE, r, id(v)>
    validValue = v; validRound = r
```



After GST


If p (correct) locks a value v
in round r, then
Before the end of round r,
every correct set:
validValue = v
validRound = r
<PROPOSAL, r, v, vr>
<PREVOTE, r, id(v)>
p1
p2
p3
p4
validValue = v
validRound = r

Termination Scenario



P_1 is correct AND for every correct process:
  lockedValue = v OR lockedRound < vr AND
  Communication between correct processes is timely

Then all correct processes decide v in round r

How to ensure Termination Scenario

- If GST starts at time t AND
- Highest round started by correct process at time t is r_0 AND
- At the end of round r_0, p_1 is a correct process with the highest lockedRound

Then, in a round $r_1 > r_0$ in which p_1 is a proposer:

P_1 sends `<PROPOSAL, r1, validValue, validRound>` AND

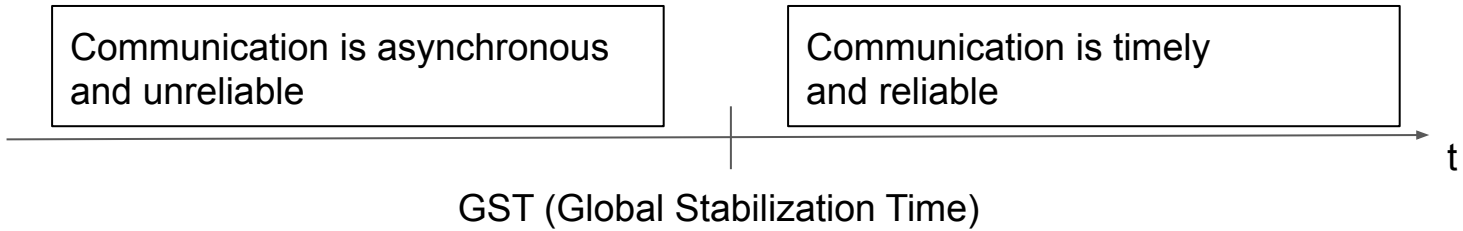
validRound of p_1 is > lockedRound or validValue = lockedValue for every correct process

AS for every round r ($r_0 < r < r_1$),

- if a correct process locked some value v' in the round r,
- before the end of the round r, p_1 sets validValue to v' and validRound to r

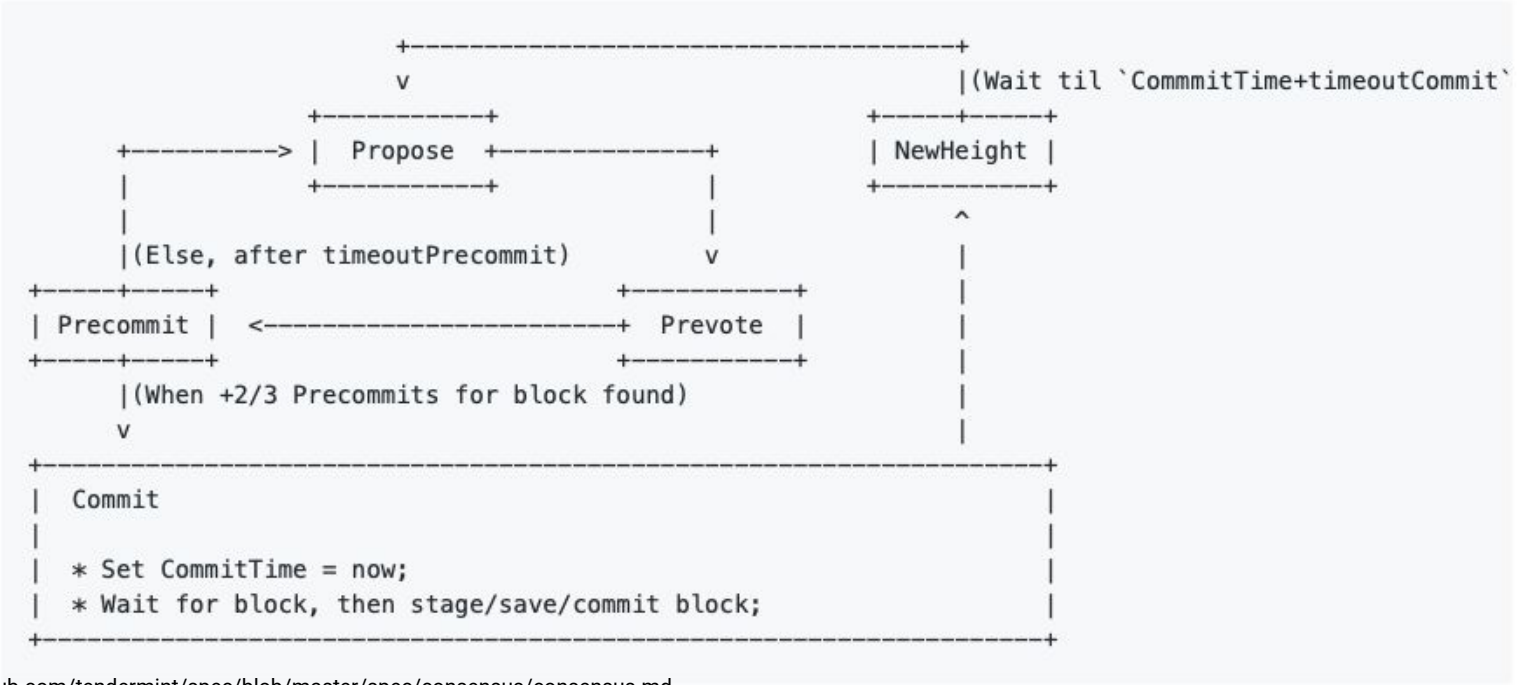
System model

- Partially synchronous system model (DLS88)
  - Communication between correct processes is reliable and timely (bounded with Delta) after GST
- At most f processes can be faulty (Byzantine faults)
- Gossip communication:
  - If a correct process receives a message m at time t, all correct processes will receive m before max(t, GST) + Delta



States in Tendermint consensus

State Machine Diagram



https://github.com/tendermint/spec/blob/master/spec/consensus/consensus.md