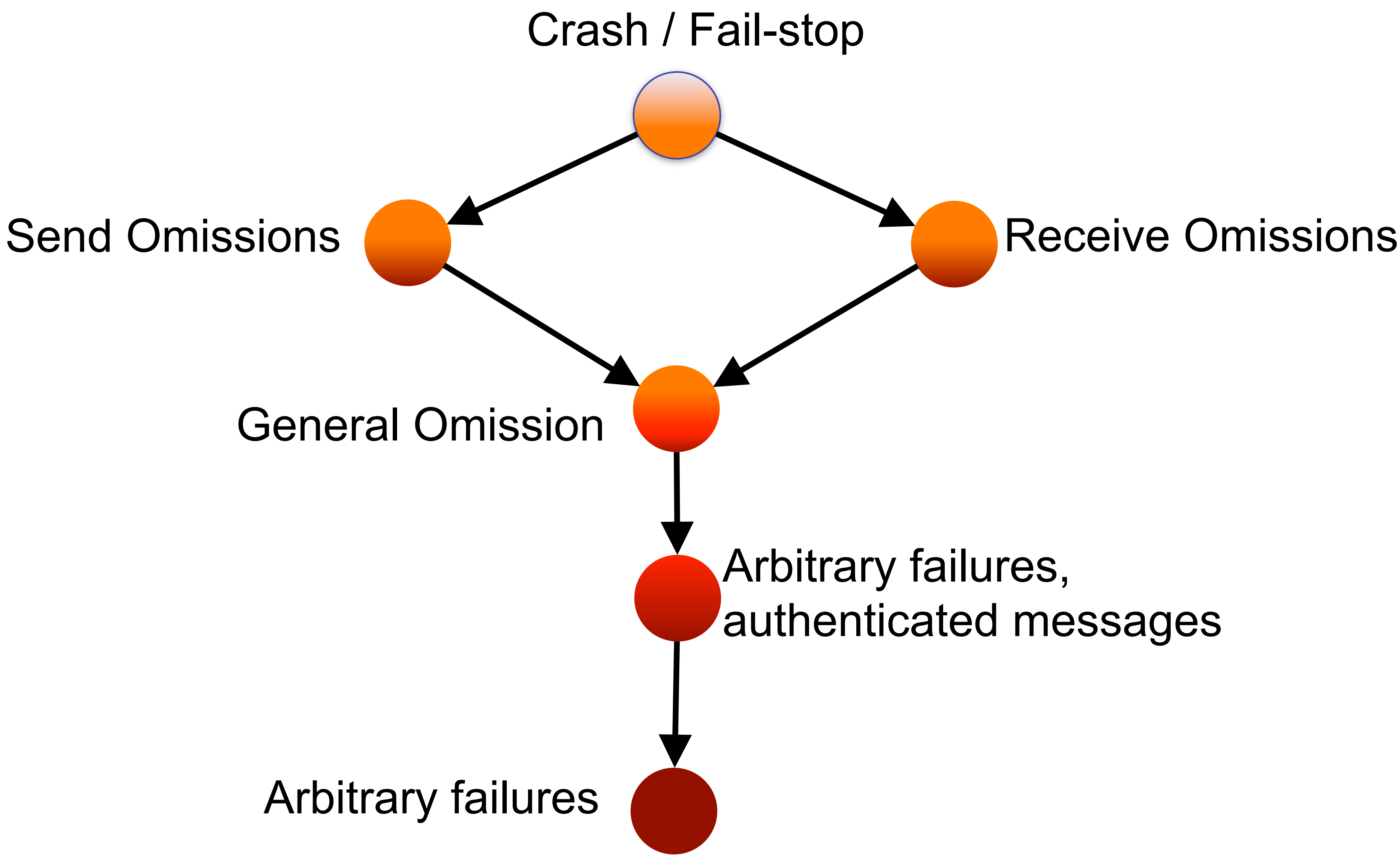
Byzantine Failures

Nikola Knezevic

Different Types of Failures



Arbitrary "Byzantine" Failures

- Some subset of the processes may fail:
  - Send fake messages
  - Not send any messages
  - Try to disrupt the computation

Why Byzantine Fault Tolerance?

- Does this happen in the real world?
  - Malfunctioning hardware
  - Buggy software
  - Compromised system due to hackers
- Assumptions are vulnerabilities
- Is the cost worth it?
  - Hardware is always getting cheaper
  - Protocols are getting more and more efficient.

Byzantine Fault Tolerance

- Seen in the fail-stop model:
  - Question: "How do you build a reliable service?"
  - Answer: State machine replication solves the problem of crash failures.
- This week:
  - What if we want to build a service that can tolerate buggy software, hackers, etc... How do we build a service that tolerates arbitrary failures?

Consensus

- Key building block for state machine replication.
  - Agreement: not two processes decide on different values.
  - Validity: every decision is the initial value of some process.
  - Termination: every correct process eventually decides.

Validity

- (Strong) Validity: If all correct processes start with the same initial value v, then value v is the only decision value of correct processes.

- (Weak) Validity: If there are no failures, then every decision is the initial value of some process.

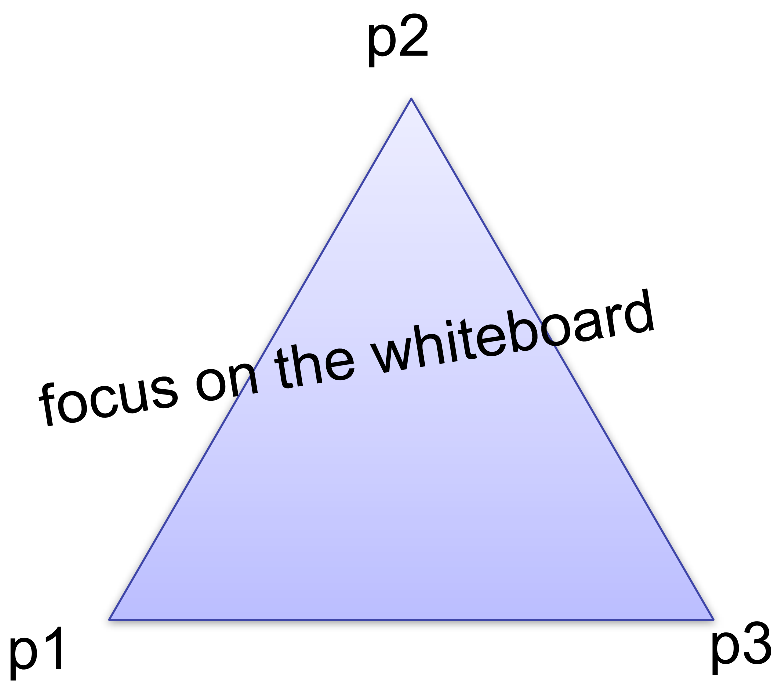
Achieving Fault Tolerance

- How many arbitrary failures can we tolerate?
  - Assume a system of n processes.
  - For crash failures:
    - If failure-detector P, then n-1 failures.
    - If failure-detector <>P, then $< n/2$ failures.
  - For arbitrary failures:
    - If failure-detector P, then $< n/3$ failures.

Number of Possible Failures

Lemma 1: In a system with n=3 and 1 failure, there is no algorithm that solves consensus.

Assume there exists some such protocol A.



Another look on why n>3t

- read-write register
- t faulty, liveness => n-t responses
- W set: n-t responses
- R set: n-t responses
- W and R have n-2t in common
  - t may not be faulty
  - W and R have n-3t in common
- => n>3t

Number of Possible Failures

Lemma 1: In a system with n processes & t = n/3 failures, there is no algorithm that solves consensus.

Proof: By reduction. Given an algorithm A that solves consensus for system (3t,t) where n=3t, we show how to solve consensus in a system (3,1) where n=3 and t=1. Since we know this is impossible, we get a contradiction.

Number of Possible Failures

Lemma 1: In a system with n processes and t = n/3 failures, there is no algorithm that solves consensus.

Proof: (Continued)

- Given algorithm A for (3t, t)-system.
- In (3,1) system, each process pi simulates t process in system (3t, t). It assigns its own value to each process that it is simulating, and outputs a decision if any of them decide.

Solving Consensus

- Assume synchronous rounds (i.e., failure detector P):

  In every round, each correct process:

  1. sends a message
  2. receives all messages sent in that round
  3. updates its state.

Solving Consensus

- Two variants:
  1. No signatures, no cryptography.
  2. Signatures: a process can sign a message such that every other process that receives the signed message can determine precisely who it came from.
     - Note: If a process forwards a signed message to someone else, they can check the signature too!

Solving Consensus

- Today:
  1. Algorithm Signed_Consensus: solves consensus when processes can sign messages.
  2. Algorithm Echo_Broadcast: provides some special reliable broadcast guarantees. NOT (today)
  3. Use Echo_Broadcast to solve consensus with no signatures. NOT (today)

Authenticated Agreement

- Each process pi begins with initial value vi.
- Each process maintains a set of candidate values VALUES.
- For any string x, process pi can generate a signature sign(x,i) that authenticates the fact that x came from pi.
  - No other process except pi can generate sign(x,i).
  - Every other process can verify that sign(x,i) is correct.

Authenticated Agreement

- In round 0, process pi:
  1. Adds its initial value vi to VALUES.
  2. Generates signature s=sign(vi).
  3. Sends [vi, s] to all in round 1.

Authenticated Agreement

- In each round $0 < r \leq t+1$, process pi:
  1. Receives a set of message M.
  2. Each m in M is [v : s1 : s2 : s3 : ...]
     - Each sj is the signature of some process pj on the string [v : s1 : s2 : ... : s[j-1]].
     - If any signatures are invalid, then discard m.
  3. For each m in M:
     - If pi already signed m, then do nothing.
     - Otherwise, pi adds its signature s = sign(m)
     - Then in the next round, pi sends [m : s] to every process that has not already signed m.
     - Add v to set VALUES.

Authenticated Agreement

- At the end of round t+1:
  - Choose the minimum v in VALUES.
  - Decide(v).

Authenticated Agreement

- Proof: Termination
  - Every node decides after t+1 rounds.

Authenticated Agreement

- Proof: Weak Validity
  - Assume every process is correct. Then all messages sent contain values that were proposed by some process. So every value in VALUES was proposed by some value.

Authenticated Agreement

- Proof: Agreement
  - Let v be the minimum value decided by some correct process.
  - Let r be the first round in which any correct process adds v to VALUES.
  - Let pj be a correct process that adds v to VALUES in round r.

Authenticated Agreement

- Proof: Agreement
  - Case 1: r=0
    - v = vj is the initial value of process pj.
    - Since pj is correct, it sends [vj : sign(vj)] to every process in round 1.
    - Every process receives this message and adds v to VALUES.

Authenticated Agreement

- Proof: Agreement
  - Case 2: $1 < r < t+1$
    - Process pj receives a message m containing value v.
    - Every process that has already sign m has already received value v.
    - Process pj signs m and in the next round sends it to every process that has not yet received v.
    - Since pj is correct, every process receives the signed m in the next round and extracts v, adding it to VALUES.

Authenticated Agreement

- Proof: Agreement
  - Case 3: r = t+1
    - Process pj receives a message m containing value v.
    - Message m contains t+1 signatures.
    - Thus, one of the processes that signed m must have been correct!
    - But, by assumption, round r is the first round in which any process puts value v in VALUES.
    - Contradiction!! Case 3 can't happen.

Authenticated Agreement

- Proof: Agreement
  - Thus, we conclude that every process adds v to VALUES by the end of round t+1.
  - By assumption, v is the minimum value decided by any process.
  - Since each process decides the smallest value it has in VALUES, every process decides v.

Byzantine Agreement

- Summary:
  - In a synchronous model, in 2(t+1) rounds we can solve Byzantine agreement as long as $t<n/3$.
  - Note: consensus can be solved in t+1 rounds. But standard solutions use exponential message complexity! (See EIG trees.)

Byzantine Fault Tolerance

- Idea: use Byzantine Agreement to build a replicated state machine!
- Robust: tolerates arbitrary failures.
- But: how to do it efficiently? What about with <>P? Can we do Byzantine Paxos??