Consensus

Seif Haridi

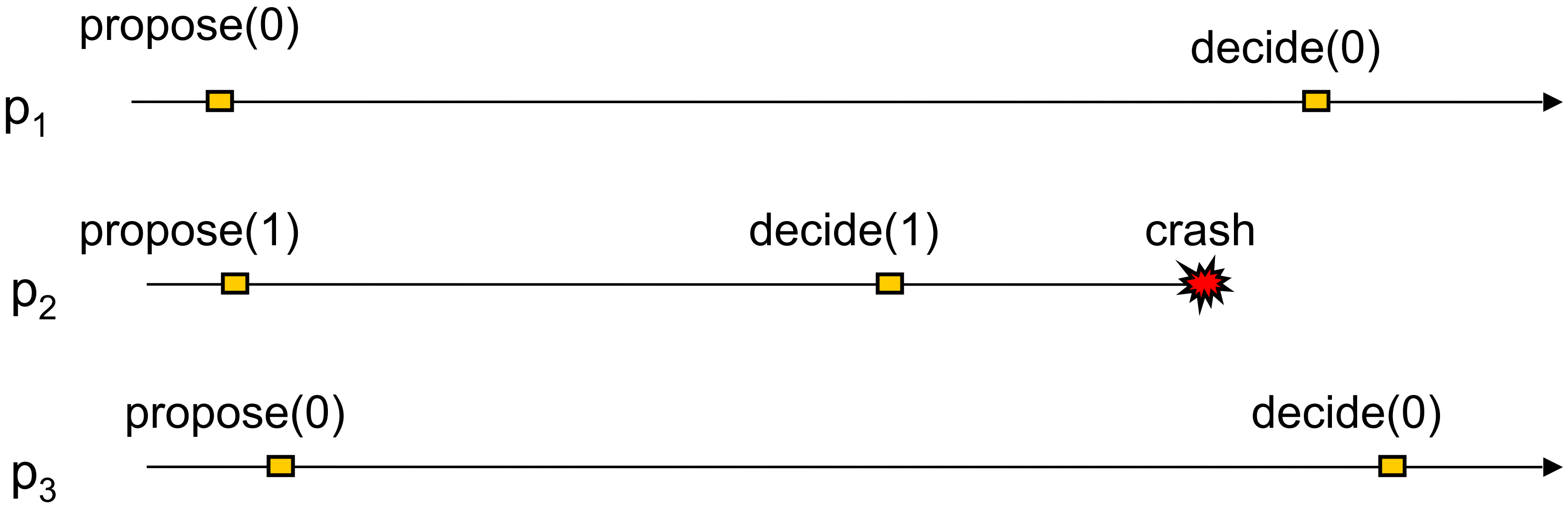
Consensus

- In consensus, the processes propose values
  - they all have to agree on one of these values
- Solving consensus is key to solving many problems in distributed computing
  - Total order broadcast (aka Atomic broadcast)
  - Atomic commit (databases)
  - Terminating reliable broadcast
  - Dynamic group membership
  - Stronger shared store models

Single Value Consensus Properties

- ***C1. Validity***
  - Any value decided is a value proposed
- ***C2. Agreement***
  - No two correct nodes decide differently
- ***C3. Termination***
  - Every correct node eventually decides
- ***C4. Integrity***
  - A node decides at most once

Sample Execution

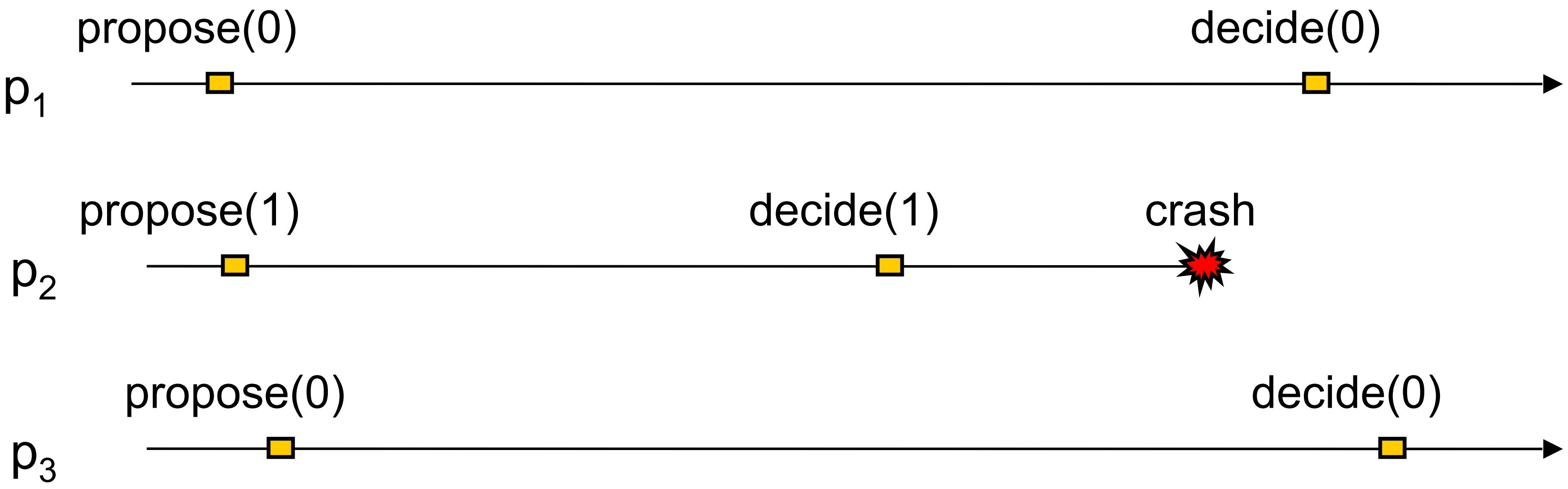


Does it satisfy consensus? yes

Uniform Consensus Properties

- ***C1. Validity***
  - Any value decided is a value proposed
- ***C2'. Uniform Agreement***
  - No two nodes decide differently
- ***C3. Termination***
  - Every correct node eventually decides
- ***C4. Integrity***
  - No node decides twice

Sample Execution



Does it satisfy uniform consensus? no

(Regular) Consensus
Fail-stop model

Consensus Interface

- ***Events***
  - **Request**: ⟨c Propose | v⟩
  - **Indication**: ⟨c Decide | v⟩

- ***Properties:***
  - ***C1, C2, C3, C4***

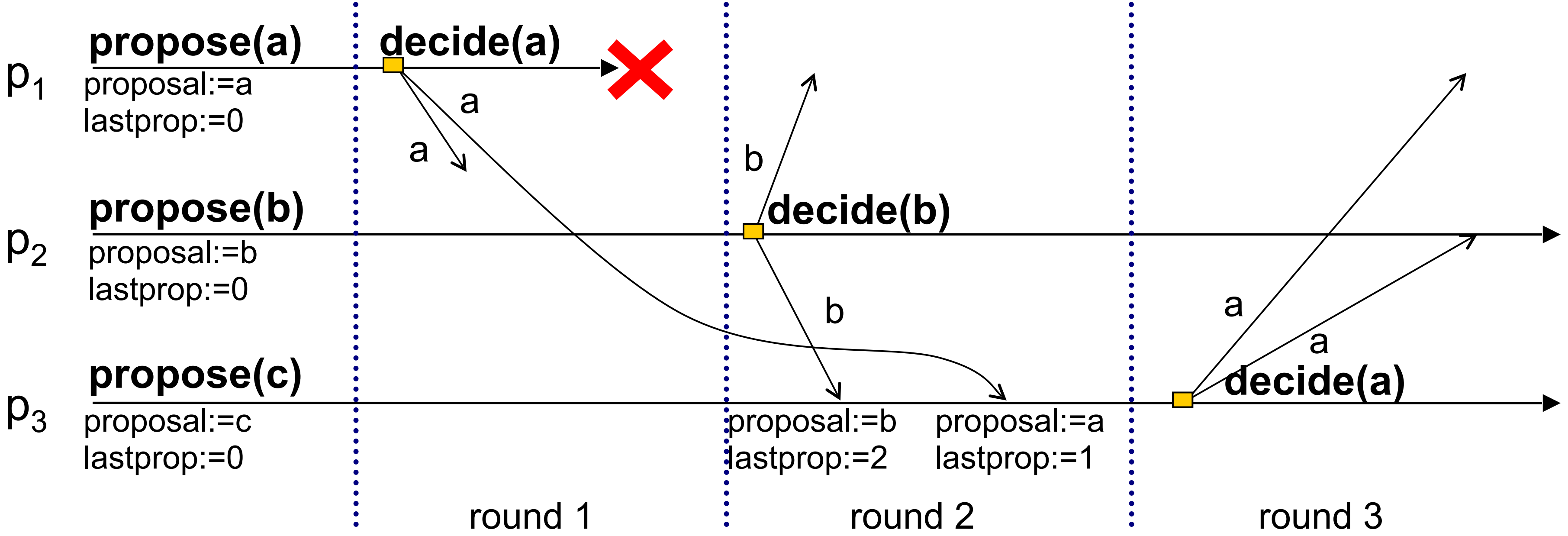
Hierarchical Consensus

- Use perfect fd (**P**) and best-effort bcast (BEB)
- Each process stores its proposal in ***proposal***
  - Possible to adopt another proposal by changing ***proposal***
  - Store identity of last adopted proposer in ***lastprop***
- Loop through rounds 1 to N
  - In round i
    - process i is leader and
      - broadcasts ***proposal*** v, and decides ***proposal*** v
    - other processes
      - adopt i's proposal v and remember ***lastprop*** i or
      - detect crash of i

Hierarchical Consensus Idea

- Basic idea of hierarchical consensus
  - There must be a first correct leader p,
    - p decides its value v and beb-casts v
    - BEB ensures all correct process get v
      - Every correct process adopts v
      - Future rounds will only propose v

Problem with orphan messages…



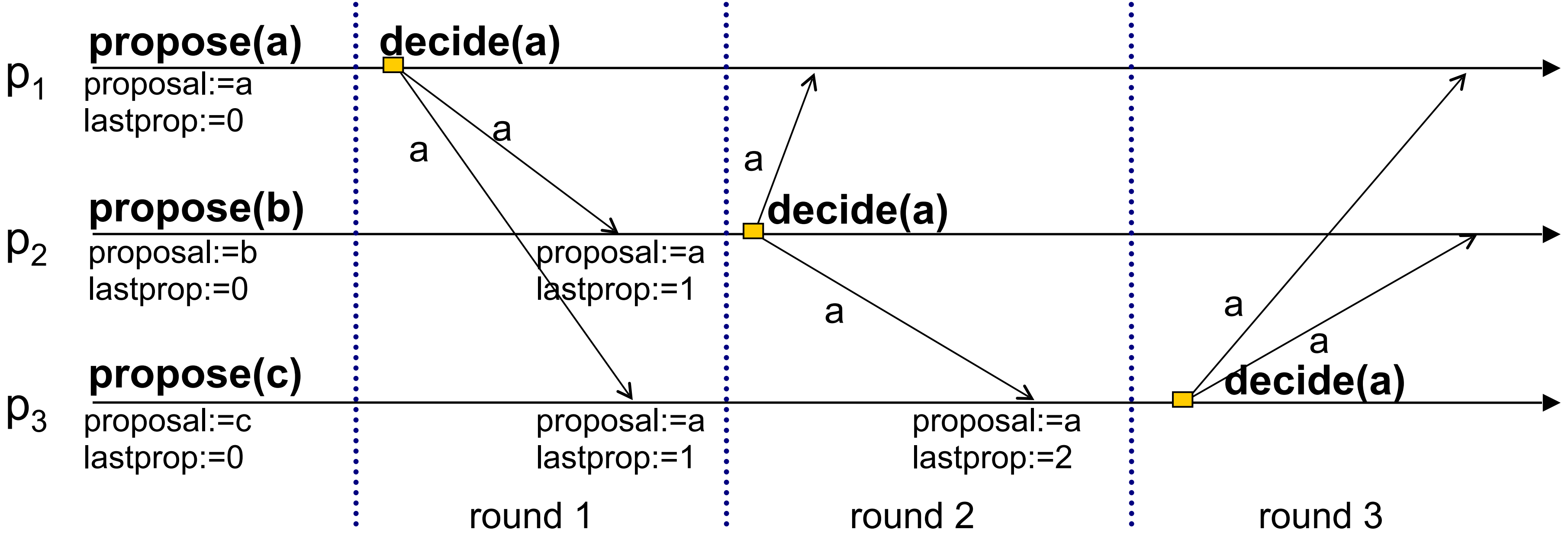
Only adopt from node i if i > lastProp?

Invariant to avoid orphans

- Leader in round r might crash,
  - but much later affect some node in round > r
- Rank: p1 < p2 < p3 < ...
- Invariant
  - adopt if proposer p is ranked higher than ***lastprop***
  - otherwise p has crashed and should be ignored

12

Execution without failure…



Implementation and correctness

Hierarchical Consensus Impl. (1)

- **Implements:** Consensus (c)
- **Uses:**
  - BestEffortBroadcast (beb)
  - PerfectFailureDetector (P)
- **upon event** ⟨Init⟩ **do**
  - detected := ∅; round := 1;
  - proposal := ⊥; lastprop := 0
  - **for** i = 1 **to** N **do**
    - broadcast[i] := delivered[i] := false
- **upon event** ⟨crash | p_i⟩ **do**
  - detected := detected ∪ { rank(p_i) }
- **upon event** ⟨cPropose | v⟩ **do**
  - **if** proposal = ⊥ **then**
  - proposal := v

last adopted proposal and last adopted proposer id

Set process's initial proposal, unless it has already adopted another node's

Hierarchical Consensus Impl. (2)

- **upon** round = rank(self) **and** broadcast[round] = false **and** proposal ≠ ⊥ **do**
  - broadcast[round] := true
  - **trigger** ⟨cDecide | proposal⟩
  - **trigger** ⟨bebBroadcast | (DECIDED, round, proposal)⟩

if I am leader

trigger once per round

trigger if I have proposal

permanently decide

- **upon event** ⟨bebDeliver | pi, (DECIDED, r, v)⟩ **do**
  - **if** r > lastprop **then**
  - proposal := v; lastprop := r
  - delivered[r] := true

Invariant: only adopt "newer" than what you have

- **Upon** delivered[round] **or** round ∈ detected **do**
  - round := round + 1

next round if deliver or crash

Correctness

- Validity
  - Always decide own proposal or adopted value
- Integrity
  - Rounds increase monotonically
  - A node only decide once in the round it is leader
- Termination
  - Every correct node makes it to the round it is leader:
    - If some leader fails, completeness of P ensures progress
    - If leader correct, validity of BEB ensures delivery

Correctness (2)

- Agreement
  - No two correct nodes decide differently

  - Take correct leader with minimum id **i**
    - By termination it will decide v
    - It will BEB v
      - Every correct node gets v and adopts it
      - No older proposals can override the adoption
      - All future proposals and decisions will be v

- How many failures can it tolerate? [d]
  - N-1

How about uniform consensus?

Formalism and notation important…

p_i

```
xi := proposal
for r:=1 to N do
    if r=i then
        forall j in 1..N do send <val, xi, r> to pj;
        decide xi
    if collect<val, x´, r> from r then
        xi := x´;
end
```

- Control-oriented vs. event-based notation
  - **collect<> from** r: is false iff FD detects p_r as failed
- NB: the control-oriented code ensures proposals are adopted in monotonically increasing order!

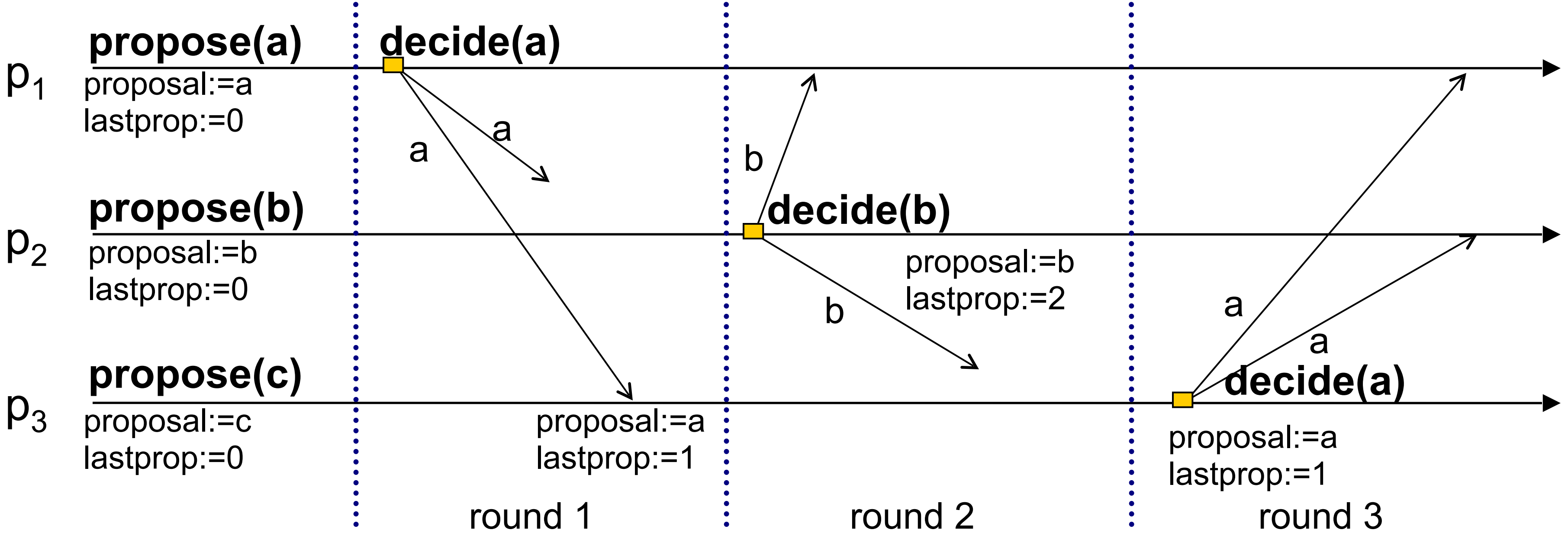
Uniform Consensus with P

- Move decision to the end

```
x_i := input
for r:=1 to N do
    if r=i then
        forall j in 1..N do send <val, x_i, r> to Pj;
        decide x_i
    if collect<val, x´, r> from r then
        x_i := x´;
end
decide x_i
```

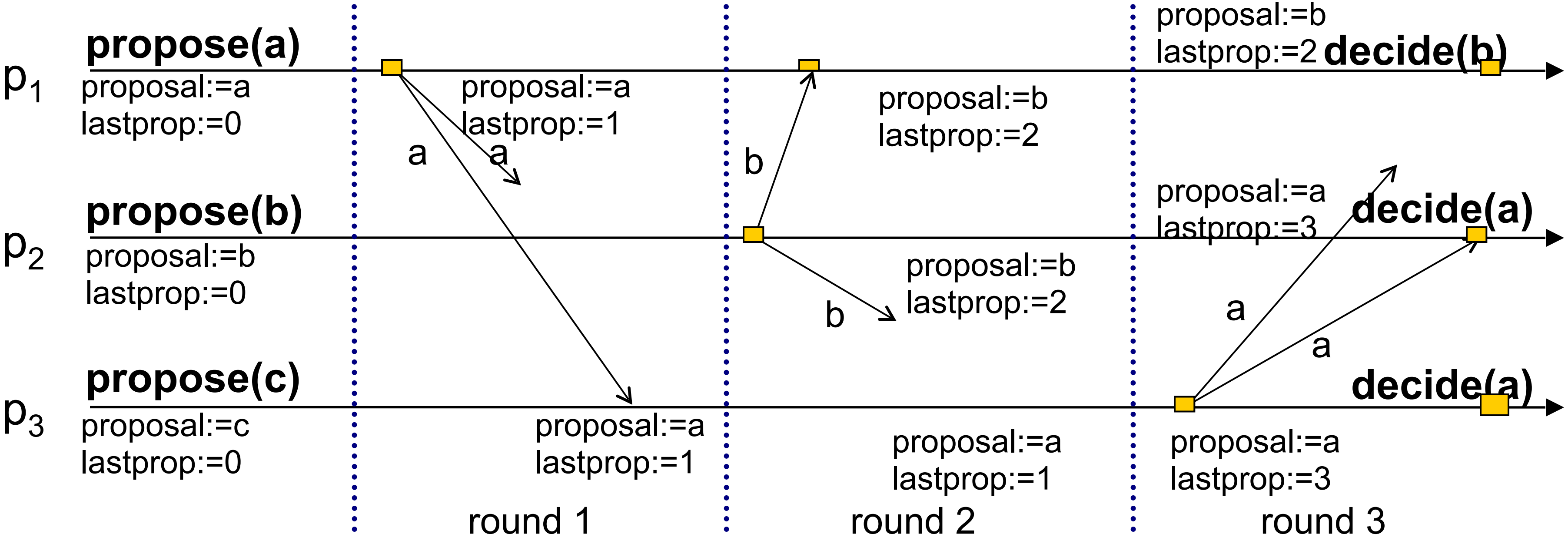
Execution with inaccurate FD

p2 suspects p1, p3 suspects p2 (regular consensus)



Execution with inaccurate FD

p2 susp p1, p3 susp p2, p1 susp p3 (uniform consensus)



Possible with weaker FD than P?

Same algorithm, just use S!

- Recall, Strong Detector (S)
  - Strong Completeness
    - Eventually every failure is detected
  - Weak Accuracy
    - There exists a correct process which is never suspected by any other node
  - Roughly, like P, but accuracy with respect to one process

Correctness

- Validity
  - Always decide own proposal or adopted value
- Integrity
  - Rounds increase monotonically
  - A node only decides once in the end
- Termination
  - Every correct node makes it to the last round
    - If some leader fails, completeness of S ensures progress
    - If leader correct, validity of BEB ensures delivery

Correctness (2)

- Uniform Agreement
  - No two processes decide differently
  - Take an “accurate” correct leader with id **i**
    - By weak accuracy (S) & termination such a process exists
    - It will BEB v
      - Every correct process gets v and sets x_i=v
      - x_i is v in subsequent rounds, final decision is v by all
  - NB: the control-oriented code ensures proposals are adopted in monotonically increasing order!

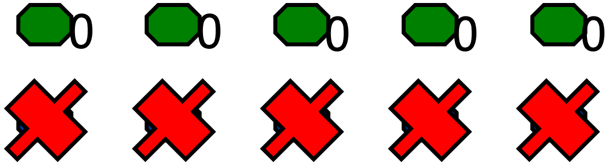
Possible with weaker FD than P?

Tolerance of Eventuality

Tolerance of Eventuality (1/3)

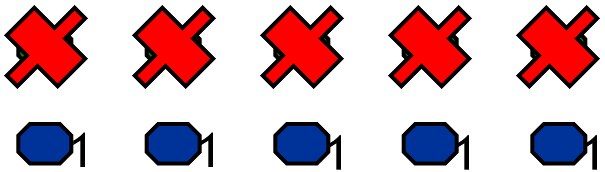
- Eventually perfect detector, cannot solve consensus with resilience $t \geq n/2$
- Proof by contradiction (specific case):
  - Assume it is possible, and assume N=10 and t=5
  - The ◊P detector initially tolerates any behavior

Green nodes correct
Blue nodes crashed
Detectors behave perfectly
Consensus is 0 at time t_0



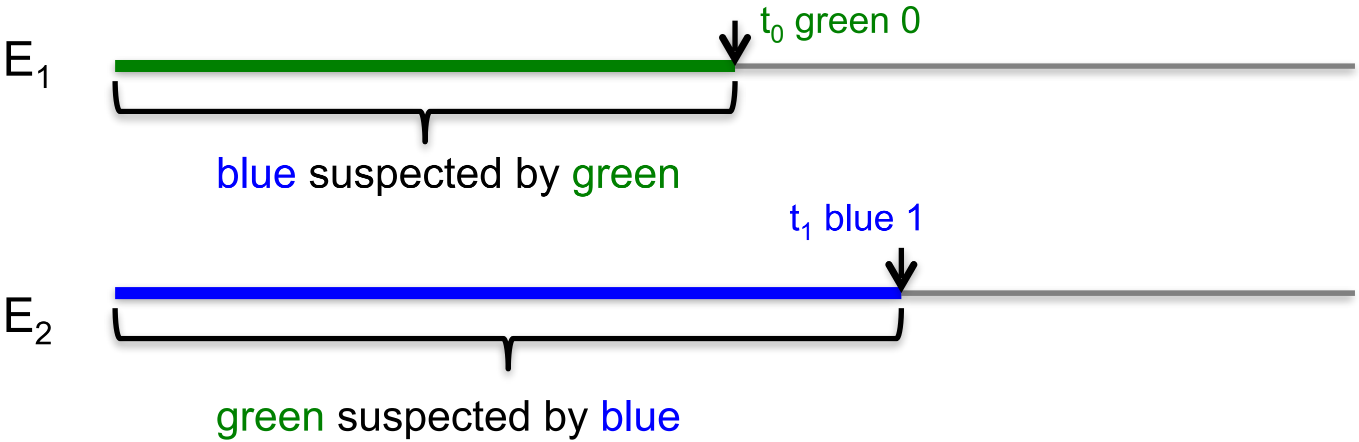
Tolerance of Eventuality (2/3)

- Eventually perfect detector, cannot solve consensus with resilience $t \geq n/2$
- Proof by contradiction:
  - Assume it is possible, and assume N=10 and t=5
  - The ◊P detector initially tolerates any behavior

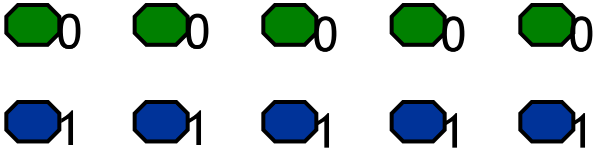


Blue nodes correct
Green nodes crashed
Detectors behave perfectly
Consensus is 1 at time t_1

Tolerance of Eventuality (3/3)

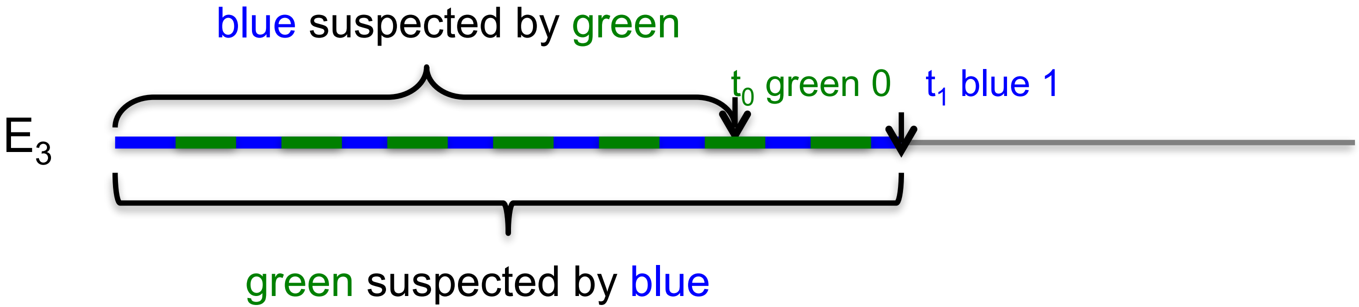


For t_0 time, green nodes suspect blue are dead
Green nodes decide 0
Thereafter detectors behave perfectly

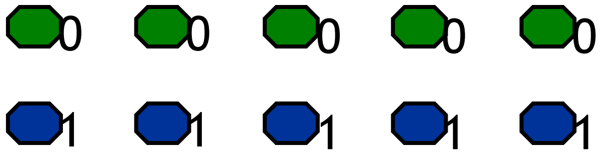


For t_1 time, blue nodes suspect green are dead
Blue nodes decide 1
Thereafter detectors behave perfectly

Tolerance of Eventuality (3/3)



- E3 is an execution that combines E1 and E2
- The view of each green process is the same as E1
- The view of each blue process is the same as E1
- But they decide different values



Proof technique

- Referred to as partitioning argument
- How to formalize it? [d]
  - Time doesn't exist
  - Reason on prefix of executions
    - Traces only contains events of green nodes… (E1)
    - Traces only contains events of blue nodes… (E2)
    - Combine the two traces (E3)
    - View of each process is the same as before

Consensus possible with weaker FD?

- Yes, we'll solve it for ◊S
  - Weaker than ◊P
  - We'll show binary consensus
- Recall, Eventually Strong Detector (◊S)
  - Strong Completeness
    - Eventually every failure is detected
  - Eventual Weak Accuracy
    - Eventually there exists a correct node which is never suspected by any other node
  - Roughly, like ◊P, but accuracy w.r.t. one node

Rotating Coordinator for ◊S

- For the eventually strong detector
  - The trivial rotating coordinator will not work
  - Why?
    - “Eventually” might be after the first N rounds
- Basic idea (rotating coordinator for ◊S)
  - Rotate forever
  - Eventually all nodes correct w.r.t. 1 coordinator
    - Everyone adopts coordinators value
- Problem
  - How do we know when to decide?

Idea for termination

- Bound the number of failures
  - Less than a third can fail ($f<n/3$)
- Similar to rotating coordinator for S:
  - **1)** Everyone send vote to coordinator **C**
  - **2)** **C** picks majority vote **V**, and broadcasts **V**
  - **3)** Every node that gets broadcast, change own vote to **V**
  - **4)** Change coordinator **C** and goto **1)**

Consensus: Rotating Coordinator for ◊S

```
x_i := input    r=0
while true do
begin
   r:=r+1    c:=(r mod N)+1                        { rotate to coordinator c }
   send <value, x_i, r> to p_c                     { all send value to coord }
```

Consensus: Rotating Coordinator for S

```
xi := input    r=0
while true do
begin
   r:=r+1     c:=(r mod N)+1                        { rotate to coordinator c }
   send <value, xi, r> to pc                        { all send value to coord }

   if i==c then                                     { coord only }
   begin
      msgs[0]:=0; msgs[1]:=0;                       { reset 0 and 1 counter }
      for x:=1 to N-f do
      begin
         receive <value, V, R> from q               { receive N-f msgs }
         msgs[V]++;                                 { increase relevant counter }
      end
      if msgs[0]>msgs[1] then v:=0 else v:=1 end { choose majority value }
      forall j do send <outcome, v, r> to pj        { send v to all }
end
```

Consensus: Rotating Coordinator for S

```
xi := input   r=0
while true do
begin
   r:=r+1    c:=(r mod N)+1                      { rotate to coordinator c }
   send <value, xi, r> to pc                     { all send value to coord }

   if i==c then                                  { coord only }
   begin
      msgs[0]:=0; msgs[1]:=0;                    { reset 0 and 1 counter }
      for x:=1 to N-f do
      begin
         receive <value, V, R> from q            { receive N-f msgs }
          msgs[V]++;                             { increase relevant counter }
      end
      if msgs[0]>msgs[1] then v:=0 else v:=1 end { choose majority value }
      forall j do send <outcome, v, r> to pj     { send v to all }
   end

   if collect<outcome, v, r> from pc then        { collect value from coord }
   begin
      xi := v                                    { adopt v }
   end
end
```

Majority Claim

- Majority Claim
  - If at least N-f nodes have (vote) **v** at start of round r:
    - At least N-f nodes have **v** at the end of round r,
    - Every leader will see a majority for **v** in all future rounds > r
- Proof
  - Each node that suspects a leader keeps previous value
  - A node change a value by receiving a message from leader
  - The leader takes a majority of N-f values received
  - At most f values received are different from **v**
    - N-2f values received are **v**
    - N-2f is a majority, i.e. > (N-f)/2 if N > 3f
  - Leader broadcasts v, and at least N-f nodes have v

Enforcing Decision

- Coordinator checks if all N-f voted same
  - Broadcast that information
- If coordinator says all N-f voted same
  - Decide for that value!

Consensus: Rotating Coordinator for S

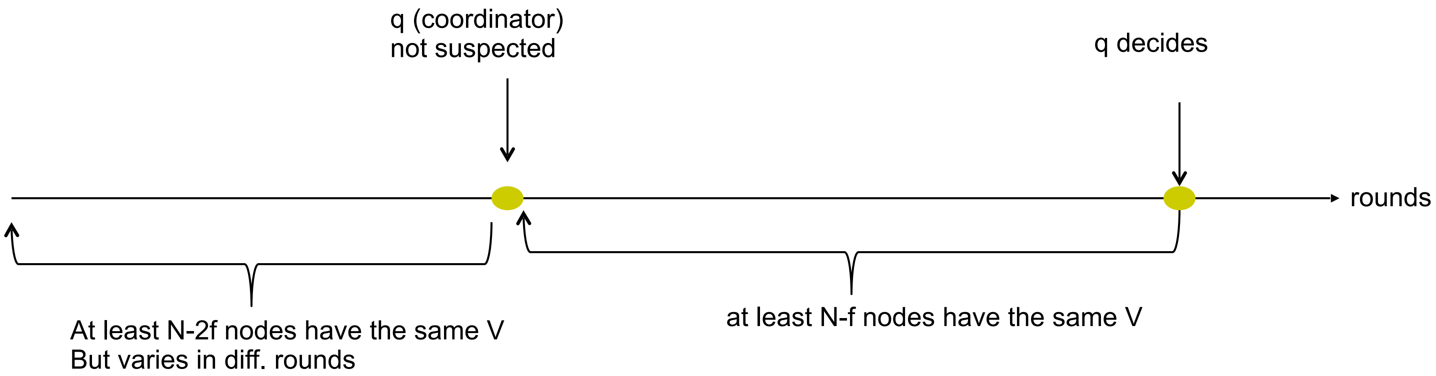
```
xi := input    r=0        i:=1
while true do
begin
   r:=r+1    c:=(r mod N)+1                        { rotate to coordinator c }
   send <value, xi, r> to pc                       { all send value to coord }
________________________________________________________________________________

   if i==c then                                    { coord only }
   begin
      msgs[0]:=0; msgs[1]:=0;                      { reset 0 and 1 counter }
      for x:=1 to N-f do
      begin
         receive <value, V, R> from q              { receive N-f msgs }
          msgs[V]++;                               { increase relevant counter }
      end
      if msgs[0]>msgs[1] then v:=0 else v:=1 end { choose majority value }
      if msgs[0]==0 or msgs[1]==0 then d:=1 else d:=0 end { all N-f same? }
      forall j do send <outcome, d, v, r> to pj    { send v to all }
   end
________________________________________________________________________________

   if collect<outcome, d, v, r> from pc then       { collect value from coord }
   begin
      xi := v                                      { change input to v }
      if d and i then begin decide(v); i:=0; end   { decide if d is true }
   end
end
```

Correctness

- Termination:
  - Eventually some q will not be falsely detected
    - Eventually q is coordinator
    - Everyone sends vote to server (majority)
    - Everyone collects q's vote (completeness)
    - Everyone adopts V
    - From now all alive nodes will vote V
    - Next time q is coordinator, d=1
    - Everyone decides

- So all alive nodes will vote the same
  - Why did we have the complex majority claim? [d]
  - To rule out situation where N-f vote 0, and f vote 1, but later everyone adopts 1

Correctness



Correctness (2)

- Agreement:
  - Decide V happens after majority of N-f vote V
  - Majority claim ensures all leaders will see majority for V
  - Only V can be proposed from then on
  - Only V can be decided
- Integrity & Validity by design…

Consensus in fail-silent?

- We solved consensus for
  - Synchrony using P
  - Partial synchrony using ◊S
- How about consensus in asynchronous setting?
  - No, it's impossible
  - Famous FLP impossibility

The End of This Lecture…

Hardness of TRB (3)

- **Accuracy**
  - TRB guarantees:
    - if src is correct, then all correct nodes will deliver m (validity and agreement)
  - Contrapositive
    - If any correct node doesn’t deliver m, src has crashed
    - <SF> delivery implies src is dead
- **Completeness**
  - If source crashes, eventually <SF> will be delivered (integrity)

TRB requires synchrony!