Consensus

Seif Haridi – Royal Institute of Technology
Peter Van Roy – Université catholique de Louvain

haridi(at)kth.se
peter.vanroy(at)uclouvain.be

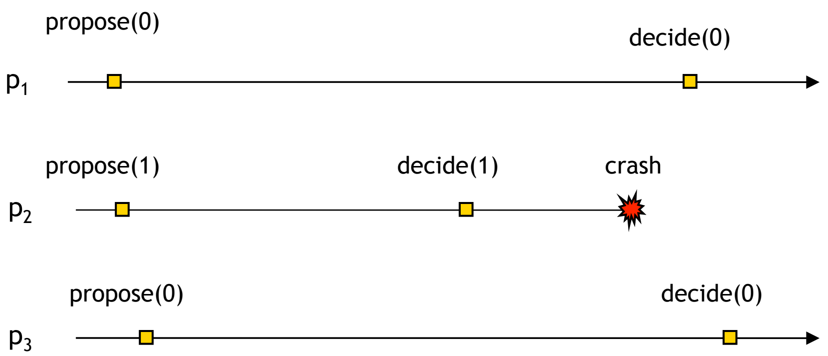
Consensus

- In consensus, the nodes 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
- The peer-to-peer transactional store that we will introduce later is based on consensus
  - Consensus can do “heavy lifting”

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

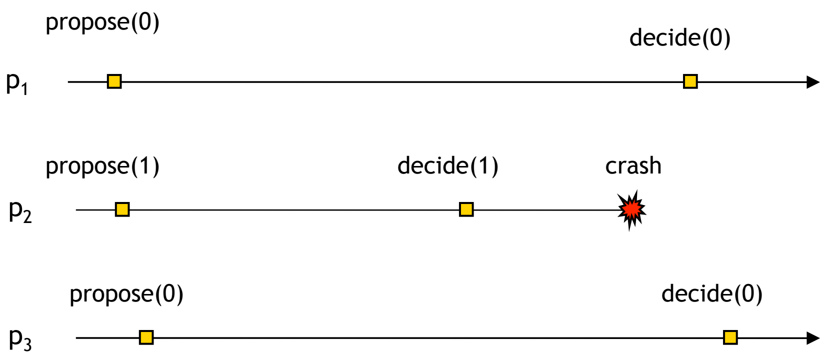


- Does it satisfy consensus? yes

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

Consensus Interface

- ***Events***
  - **Request:** ⟨cPropose | v⟩
  - **Indication:** ⟨cDecide | v⟩
- ***Properties:***
  - ***C1, C2, C3, C4***

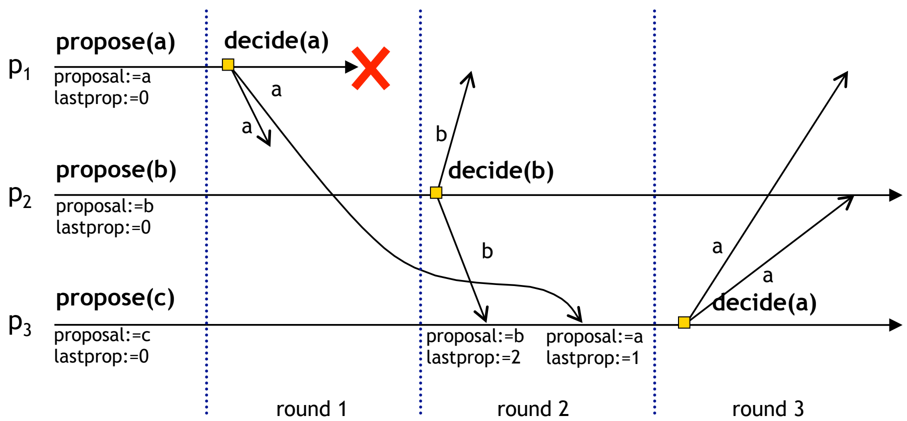
Hierarchical Consensus

- Use perfect fd (P) and best-effort bcast (BEB)
- Each node 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
    - node i is leader and
      - broadcasts ***proposal*** v, and decides ***proposal*** v
    - other nodes
      - 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 bcasts v
    - BEB ensures all correct nodes get v
      - Every correct node adopts v
      - Future rounds will only propose v

Problem with orphan messages…

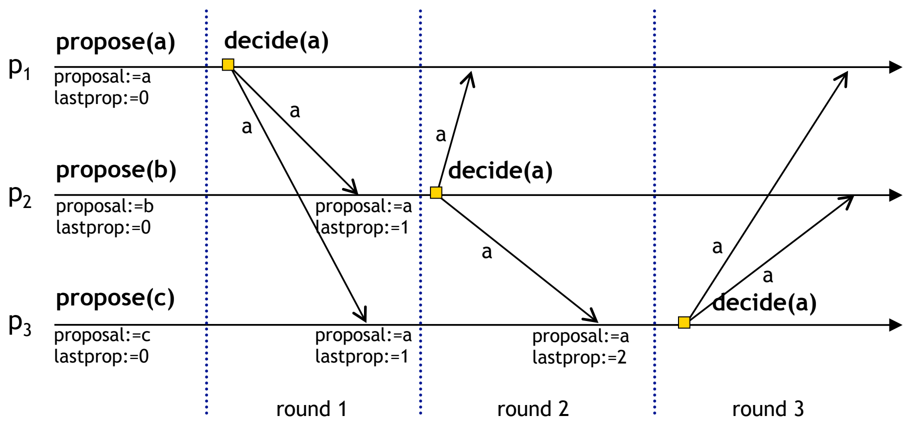


- Only adopt from node i if i>lastProp?

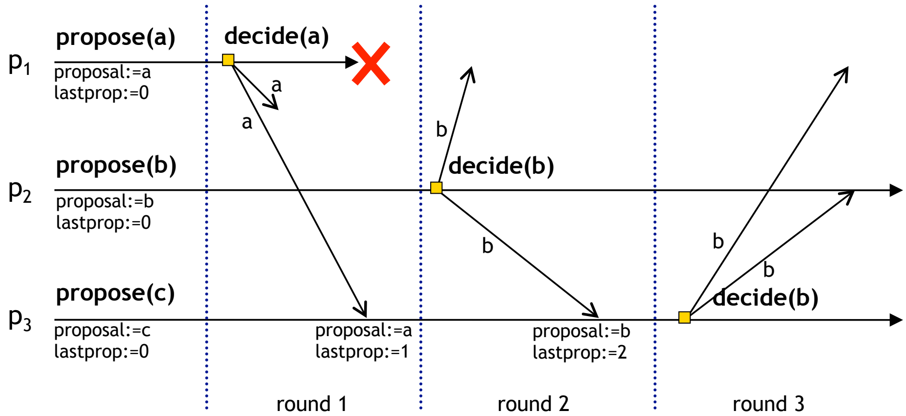
Invariant to avoid orphans

- 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 lower than ***lastprop***
  - otherwise p has crashed and should be ignored

Execution without failure…



Execution with failure…



- Uniform consensus? no

Hierarchical Consensus Impl. (1)

- **Implements:** Consensus (c)
- **Uses:**
  - BestEffortBroadcast (beb)
  - PerfectFailureDetector (P)

- **upon event** ⟨Init⟩ **do**
  - detected := ∅; round := 1;
  - proposal := ⊥; lastprop := 0 ← last adopted proposal and last adopted proposer id
  - **for** i = 1 **to** N **do**
    - broadcast[i] := delivered[i] := false

- **upon event** ⟨crash | p_i⟩ **do**
  - detected := detected ∪ { rank(p_i) }

Hierarchical Consensus Impl. (2)

- **upon event** ⟨cPropose | v⟩ **do**
  - **if** proposal = ⊥ **then**
  - proposal := v

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

- **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 decides in the round in which it is leader
- Termination
  - Every correct node makes it to the round it is leader in
    - 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

Formalism and notation important…

P_i

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

- Control-oriented vs event-based notation
  - "**collect<> from** r" is false iff FD detects P_r as failed
- What happens to orphan messages?
  - "**collect<> from** r" drops them

How about uniform consensus?

Uniform Consensus with P

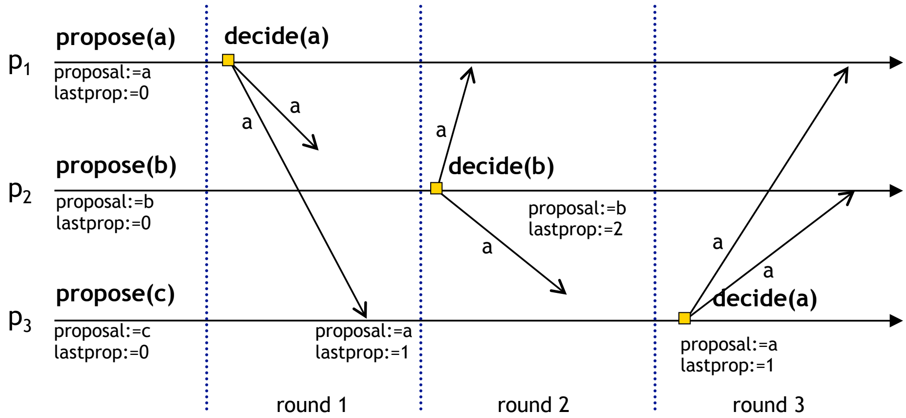
- Move decision to the end

```
xi := input
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
decide xi
```

- What happens if a node decides and then crashes? [d]

Execution with inaccurate FD

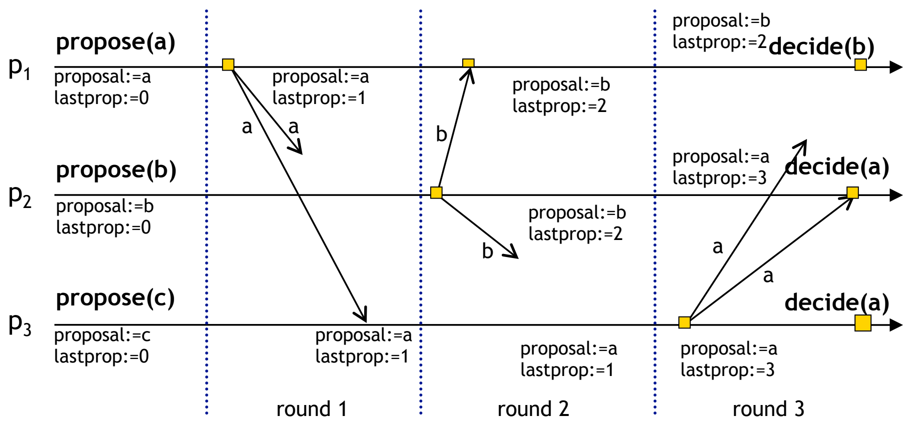
p2 suspects p1, p3 suspects p2 (nonuniform consensus)



- The algorithm doesn't work with inaccurate FD!

Execution with inaccurate FD

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



- The algorithm doesn't work with inaccurate FD!

Possible with weaker FD than P?

Uniform algorithm works with S!

- Recall, Strong Detector (S)
  - Strong Completeness
    - Eventually every failure is detected
  - Weak Accuracy
    - There exists a correct node which is never suspected by any other node
  - Roughly, like P, but accuracy w.r.t. one node
    - Called “Strong” but weaker than P ☺ !

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)

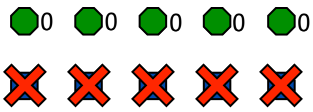
- Uniform Agreement
  - No two nodes decide differently
  - Take an "accurate" correct leader with id **i**
    - By weak accuracy (S) & termination such a node exists
    - It will BEB v
      - Every correct node gets v and sets x_i=v
      - x_i is v in subsequent rounds, final decision is v by all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

What about *eventual* failure detectors?

Tolerance of Eventuality (1/3)

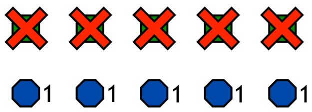
- Eventually perfect detector cannot solve consensus with resilience $t \geq n/2$
  - Resilience = how many failures it tolerates
- Proof by contradiction (specific case):
  - Assume it is possible, and assume N=10 and t=5
  - The ◊P detector initially tolerates any behavior

- 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 $t \geq n/2$
  - Resilience = how many failures it tolerates
- Proof by contradiction:
  - Assume it is possible, and assume N=10 and t=5
  - The ◊P detector initially tolerates any behavior

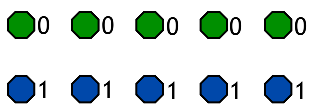


- Blue nodes correct
- Green nodes crashed
- Detectors behave perfectly
- Consensus is 1 at time t_1

Tolerance of Eventuality (3/3)

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

- 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

Proof technique

- Referred to as partitioning argument
- How to formalize it? [d]
  - Time doesn't exist
  - Reason on prefix of executions
    - Schedule only contains events of green nodes…
    - Schedule only contains events of blue nodes…
    - Combine the two schedules…
      - To make a new, valid schedule!

Consensus possible with eventual FD?

- Yes, we'll solve it for ◊S
  - Weaker than ◊P. Will the algorithm also work for ◊P? [d]
  - We'll show binary consensus (proposed values are 0 or 1)
- 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 rotating coordinator we saw before 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 coordinator's value
- Problem
  - How do we know when to decide? This is the key!

Idea for termination

- Bound the number of failures
  - Less than a third can fail ($f<n/3$)
  - This will let us decide using a majority vote
- 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 get broadcast, change vote to **V**
  - **4)** Change coordinator **C** and goto **1)**

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 }
```


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
```


Termination Detection

- Majority Claim
  - If at least N-f nodes vote V in a round r
    - Every leader will see a majority for V in all future rounds > r
- Proof
  - At most f nodes don't vote V
  - We have $2n/3=n-n/3$
  - Then $n/3<(n-f)/2$ (because $f<n/3$)
  - Then $f<(n-f)/2$ (because $f<n/3$)
    - Less than half of any n-f nodes do not vote V
    - More than half of any n-f nodes vote 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:=(process index)
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!=0 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 coordinator q (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 (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 summary

- We solved consensus for
  - Synchrony (using S, also works for P)
  - Partial synchrony (using ◊S, with <N/3 fail)
- What about uniform consensus?
  - Synchrony (same algorithm as before)
  - Partial synchrony (using ◊P, with <N/2 fail)
    - Famous algorithm called Paxos … see later in course!
    - Note that this algorithm also works for consensus!
- What about consensus in asynchronous model?
  - It's impossible if at least one process can fail
  - Famous FLP impossibility result

Terminating Reliable Broadcast (TRB)

Need for stronger RB

- In a chat application
  - Clients don't know when or if a message will be delivered

- But in some applications that use RB
  - Some server uses RB and clients await delivery
  - How long should clients await delivery?
  - TRB provides the solution

Terminating Reliable Broadcast

- Intuition
  - TRB is reliable broadcast in which
    - Sender broadcasts M
    - Receivers await delivery M
    - All nodes either deliver M or "abort"

- "Abort" indicated by special <SF> message
  - Sender Faulty

TRB Interface (1)

- **Module:**
  - Name: TerminatingReliableBroadcast (trb)
- ***Events***
  - Request: ⟨trbBroadcast | src, m⟩
    - Called by all nodes. If src≠self then m=nil
  - Indication: ⟨trbDeliver | src, m⟩
    - m may be <SF> (sender faulty) if src crashes
- ***Property:***
  - ***TRB1-TRB4***

TRB Interface (2)

- **Termination:**
  - Every correct node eventually delivers one message
- **Validity**:
  - If correct src sends m, then src will deliver m
- **Uniform agreement**:
  - If any node delivers m, then every correct node eventually delivers m
- **Integrity** (no creation):
  - If a node delivers m, then either m=<SF> or m was broadcast by src

Consensus Based TRB

- Src RB broadcast m
  - Deliver <SF> if src is suspected by P
- Caveat
  - Src crash,
    - Some get m before detected crash
    - Some detect crash before getting m (no agreement)
- Intuitive idea
  - Src BEB broadcast m
  - Nodes propose (consensus) whichever comes first:
    - Crash suspicion of src (<SF>)
    - BEB delivery from src (M)
  - Deliver consensus decision

TRB Interface (2)

- Intuitive correctness
  - If src correct, everyone gets m, and consensus decides m
- **Termination:**
  - Completeness of P and validity of BEB ensure a propose
  - Termination of consensus ensures a delivery
- **Validity**:
  - Assume a correct src sends m
  - All nodes get m (BEB validity) before suspecting src (P accuracy)
  - All propose m
  - All decide m (Consensus termination and validity)
- **Uniform agreement**:
  - By agreement of consensus
- **Integrity** (no creation):
  - Validity of consensus and no creation of BEB ensure <SF> or m is delivered

Hardness of TRB (1)

- Can we implement TRB in asynchronous networks? [d]
  - No, since consensus is reducible to TRB
  - I.e., consensus≼TRB
- Given TRB, implement consensus
  - Each node TRB its proposal
  - Save delivered values in a vector
  - Decide using a deterministic function
    - E.g. median, majority, or first non <SF> msg

Hardness of TRB (2)

- Can we implement TRB in eventually synchronous systems (with ◊P)? [d]
  - No, since P is reducible to TRB
  - I.e., P≼TRB, since TRB≼P we have TRB≃P
- Given TRB, implement P
  - Each node TRB heartbeats all the time
  - If ever receive <SF> for a node, suspect it

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!