18|10|23

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

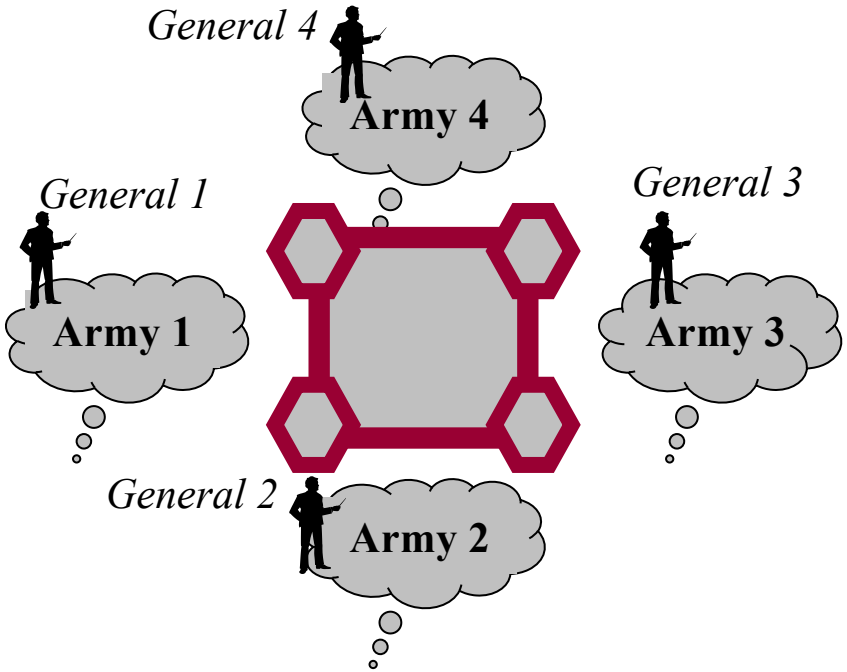
AA 2023/2024

LECTURE 10: CONSENSUS

Consensus Problem

- A group of processes must agree on a value that has been proposed by one of them (e.g., commit/abort of a transaction).
- It is an abstraction of a class of problems where processes start with their opinion and then converge on one of them
- It is a fundamental problem
- We study algorithms working on weak models

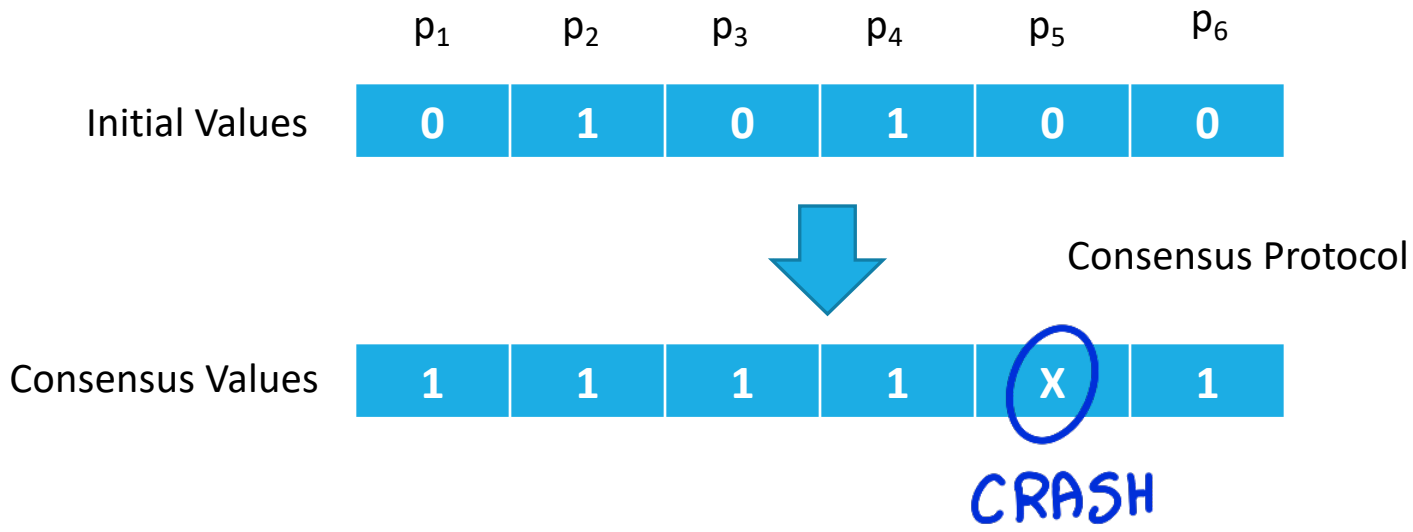
Consensus Example



Generals have to reach consensus between attack and get back

Consensus Definition

- Set of initial values ∈ {0,1}.
- Every process has to decide the same value ∈ {0,1} based on the initial proposals.



Consensus Specification

Module 5.1: Interface and properties of (regular) consensus

Module:

Name: Consensus, **instance** c.

Events:

Request: $\langle\ c, \mathit{Propose} \mid v\ \rangle$: Proposes value v for consensus. 1)

Indication: $\langle\ c, \mathit{Decide} \mid v\ \rangle$: Outputs a decided value v of consensus. 2)

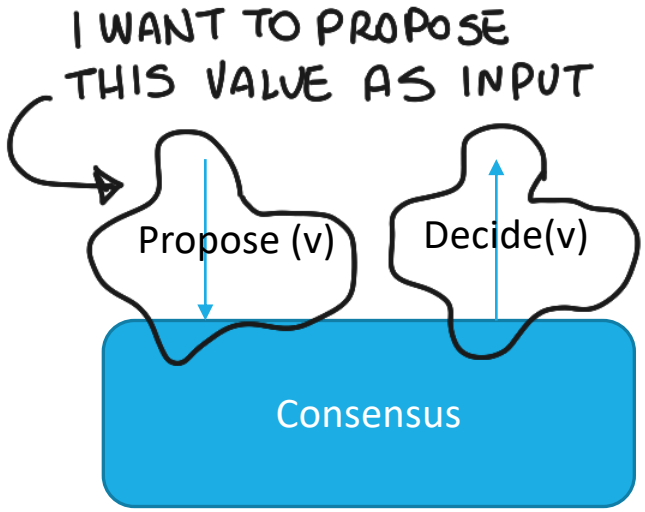
Properties:

C1: *Termination:* Every correct process eventually decides some value.

C2: *Validity:* If a process decides v, then v was proposed by some process.

C3: *Integrity:* No process decides twice. → IMPLIES

C4: *Agreement:* No two correct processes decide differently.



FLP Impossibility Result

no algorithm can guarantee to reach consensus in an asynchronous system, even with one process crash failure.

Fisher, Lynch e Patterson (FLP result)
Ref: Journal of the ACM, Vol. 32, No. 2, April 1985.

A Consensus Implementation in Synchronous Systems: Flooding Consensus

Basic Idea:

- Processes exchange their values
- when all the proposals from correct processes are available a one value can be chosen

Problem:

- due to failures, some values can be lost

Solution:

- A value can be selected only when no failures happen during the communication

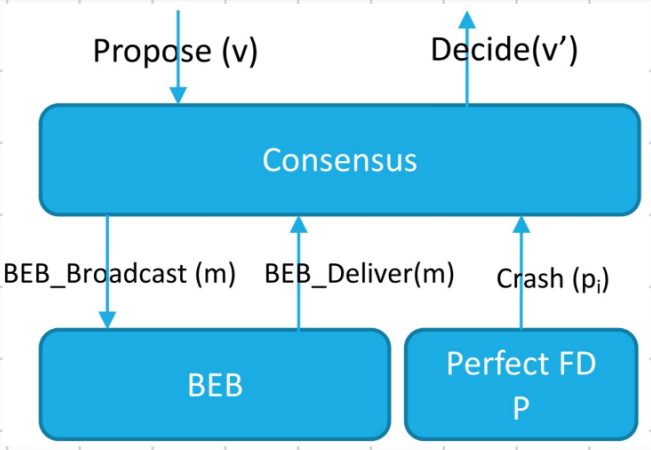
10: CONSENSUS

C1: *Termination:* Every correct process eventually decides some value.

C2: *Validity:* If a process decides v, then v was proposed by some process.

C3: *Integrity:* No process decides twice.

C4: *Agreement:* No two correct processes decide differently.



Algorithm 5.1: Flooding Consensus

Implements:
Consensus, **instance** c.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ c, *Init* $\rangle$ **do**
correct := Π; all processes
round := 1; count nº of interaction before decision
decision := $\bot$; store value that we are going to decide
receivedfrom := $[\emptyset]^N$; list composed by n entries, one for each process in the system
value proposed *proposals* := $[\emptyset]^N$;
receivedfrom[0] := Π;
round 0
msgs received from a p in a specific round

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
correct := *correct* $\setminus$ $\{p\}$;

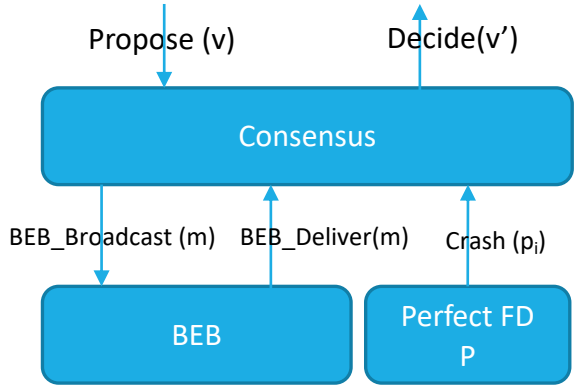
upon event $\langle$ c, Propose | v $\rangle$ **do**
proposals[1] := *proposals*[1] $\cup$ $\{v\}$;
trigger $\langle$ *beb*, Broadcast | [PROPOSAL, 1, *proposals*[1]] $\rangle$;
ROUND = 1
set of proposal for round 1
ROUND

upon event $\langle$ *beb*, Deliver | p, [PROPOSAL, r, ps] $\rangle$ **do**
proposal set
round
receivedfrom[r] := *receivedfrom*[r] $\cup$ $\{p\}$;
sender of proposal
proposals[r] := *proposals*[r] $\cup$ ps;

upon *correct* $\subseteq$ *receivedfrom*[*round*] $\wedge$ *decision* = $\bot$ **do** = TRUE
every correct p is in receivedfrom
not decided yet
if *receivedfrom*[*round*] = *receivedfrom*[*round* − 1] **then**
check if the p have crashed in the current round, checking the previous round
decision := min(*proposals*[*round*]);
min value in set
trigger $\langle$ *beb*, Broadcast | [DECIDED, *decision*] $\rangle$;
trigger $\langle$ c, Decide | *decision* $\rangle$;
else = FALSE, somone failed
round := *round* + 1;
trigger $\langle$ *beb*, Broadcast | [PROPOSAL, *round*, *proposals*[*round* − 1]] $\rangle$;

upon event $\langle$ *beb*, Deliver | p, [DECIDED, v] $\rangle$ **such that** $p \in$ *correct* $\wedge$ *decision* = $\bot$ **do**
decision := v;
trigger $\langle$ *beb*, Broadcast | [DECIDED, *decision*] $\rangle$;
trigger $\langle$ c, Decide | *decision* $\rangle$;
IF P1 DECIDE BEFORE P2, P1 TAKE THE DECISION

A Consensus Implementation in Synchronous Systems: Flooding Consensus



Algorithm 5.1: Flooding Consensus

Implements:
Consensus, **instance** c.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ c, *Init* $\rangle$ **do**
correct := Π; all processes
round := 1; count n° of interaction before decision
decision := $\perp$; store value that we are going to decide
receivedfrom := $[\emptyset]^N$; list composed by n entries, one for each process in the system
proposals := $[\emptyset]^N$; value proposed
receivedfrom[0] := Π; round 0 — msgs received from a p in a specific round

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
correct := *correct* $\setminus$ $\{p\}$;

upon event $\langle$ c, *Propose* | v $\rangle$ **do**
proposals[1] := *proposals*[1] $\cup$ $\{v\}$;
trigger $\langle$ *beb*, Broadcast | [PROPOSAL, 1, *proposals*[1]] $\rangle$;
ROUND = 1 — ROUND — set of proposal for round 1

upon event $\langle$ *beb*, *Deliver* | p, [PROPOSAL, r, ps] $\rangle$ **do**
round — proposal set
receivedfrom[r] := *receivedfrom*[r] $\cup$ $\{p\}$; sender of proposal
proposals[r] := *proposals*[r] $\cup$ ps;

upon *correct* $\subseteq$ *receivedfrom*[*round*] $\wedge$ *decision* = $\perp$ **do** = TRUE
every correct p is in receivedfrom — not decided yet
if *receivedfrom*[*round*] = *receivedfrom*[*round* − 1] **then** → check if the p have crashed in the current round, checking the previous round
decision := min(*proposals*[*round*]); min value in set
trigger $\langle$ *beb*, Broadcast | [DECIDED, *decision*] $\rangle$;
trigger $\langle$ c, *Decide* | *decision* $\rangle$;
else = FALSE, somove failed
round := *round* + 1;
trigger $\langle$ *beb*, Broadcast | [PROPOSAL, *round*, *proposals*[*round* − 1]] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [DECIDED, v] $\rangle$ **such that** $p \in$ *correct* $\wedge$ *decision* = $\perp$ **do**
decision := v;
trigger $\langle$ *beb*, Broadcast | [DECIDED, *decision*] $\rangle$;
trigger $\langle$ c, *Decide* | *decision* $\rangle$;
IF P1 DECIDE BEFORE P2, P1 TAKE THE DECISION

Example of Flooding Consensus

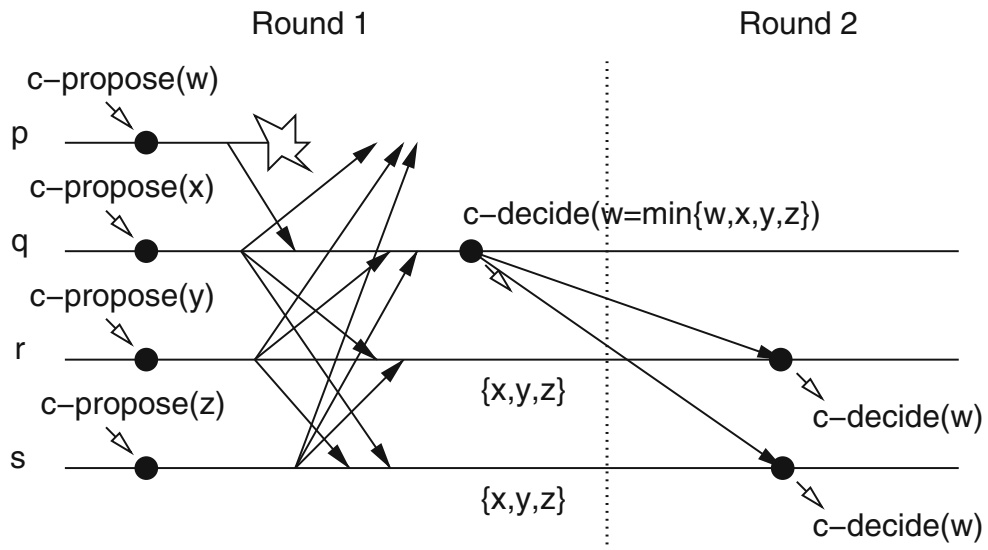
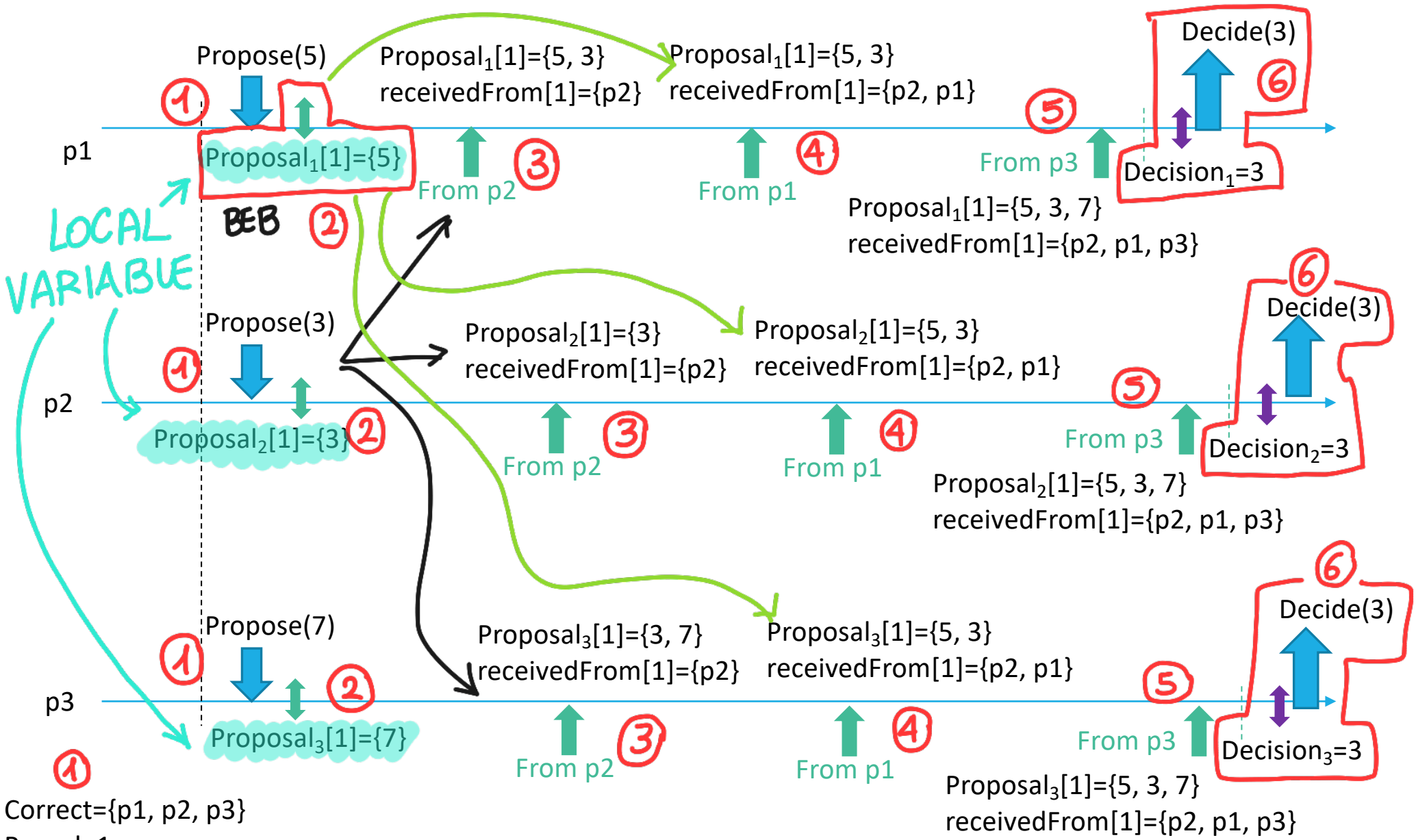
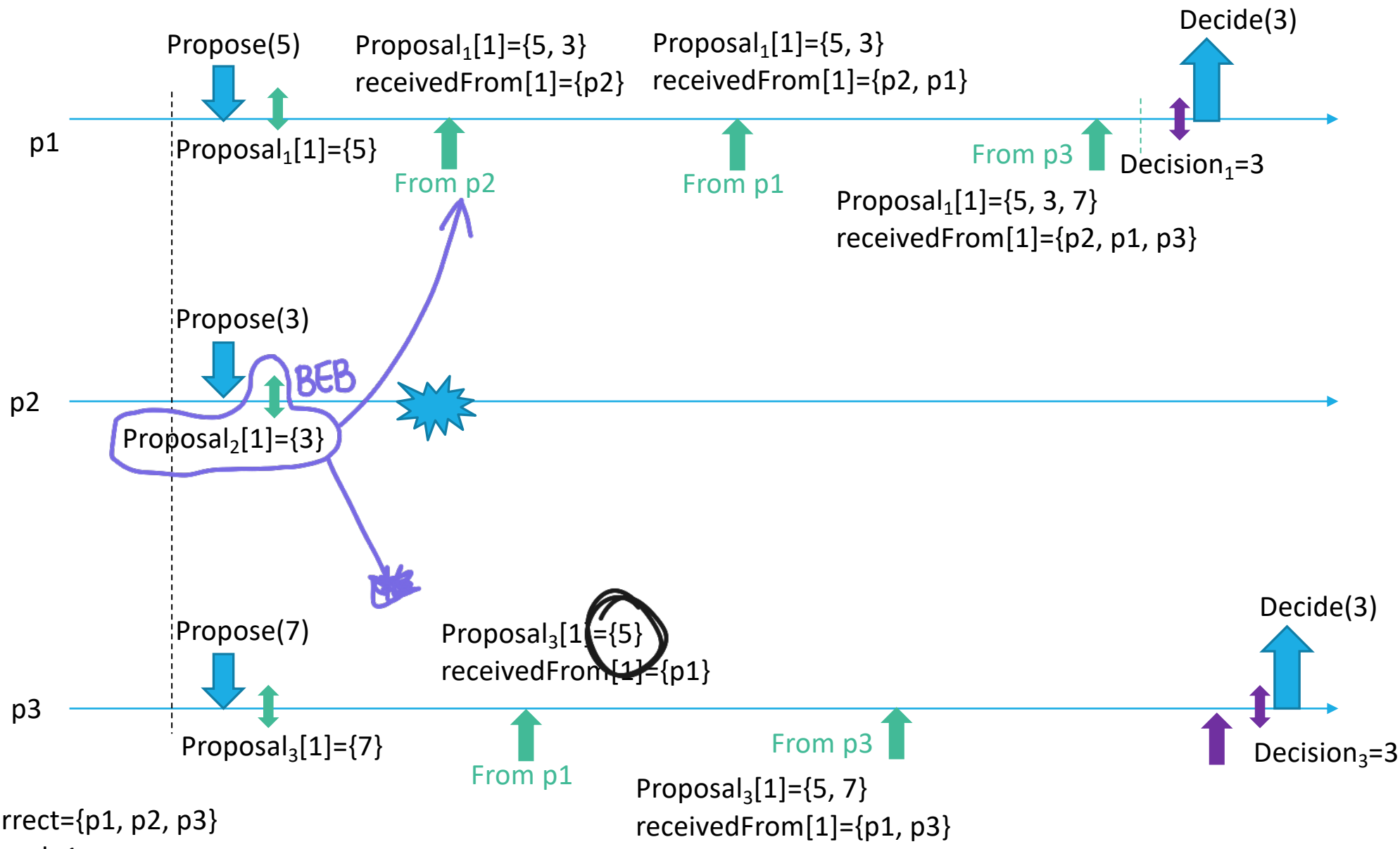
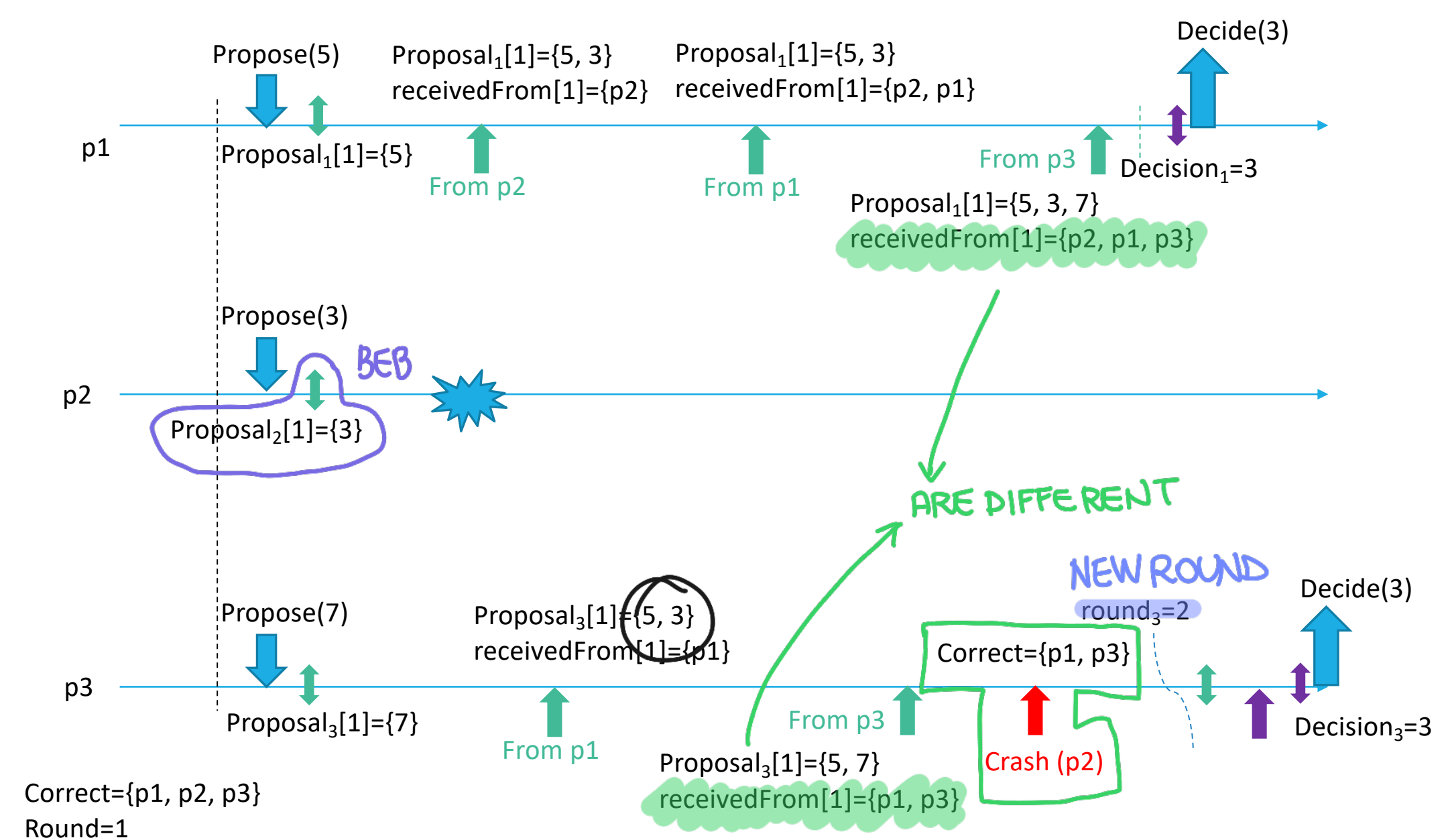


Figure 5.1: Sample execution of flooding consensus


Propose(5)
Proposal$_1$[1]={5, 3}
receivedFrom[1]={p2}
Proposal$_1$[1]={5, 3}
receivedFrom[1]={p2, p1}
Decide(3)
p1
Proposal$_1$[1]={5}
BEB
From p2
From p1
From p3
Decision$_1$=3
Proposal$_1$[1]={5, 3, 7}
receivedFrom[1]={p2, p1, p3}
LOCAL VARIABLE
Propose(3)
Proposal$_2$[1]={3}
receivedFrom[1]={p2}
Proposal$_2$[1]={5, 3}
receivedFrom[1]={p2, p1}
Decide(3)
p2
Proposal$_2$[1]={3}
From p2
From p1
From p3
Decision$_2$=3
Proposal$_2$[1]={5, 3, 7}
receivedFrom[1]={p2, p1, p3}
Propose(7)
Proposal$_3$[1]={3, 7}
receivedFrom[1]={p2}
Proposal$_3$[1]={5, 3}
receivedFrom[1]={p2, p1}
Decide(3)
p3
Proposal$_3$[1]={7}
From p2
From p1
From p3
Decision$_3$=3
Proposal$_3$[1]={5, 3, 7}
receivedFrom[1]={p2, p1, p3}
Correct={p1, p2, p3}
Round=1
Decision=⊥
receivedFrom[0]={p1, p2, p3}
INIT. STEP
Proposals[0]={}



Correct={p1, p2, p3}
Round=1
Decision=⊥
receivedFrom[0]={p1, p2, p3}
Proposals[0]={}



Correct={p1, p2, p3}
Round=1
Decision=⊥
receivedFrom[0]={p1, p2, p3}
Proposals[0]={}

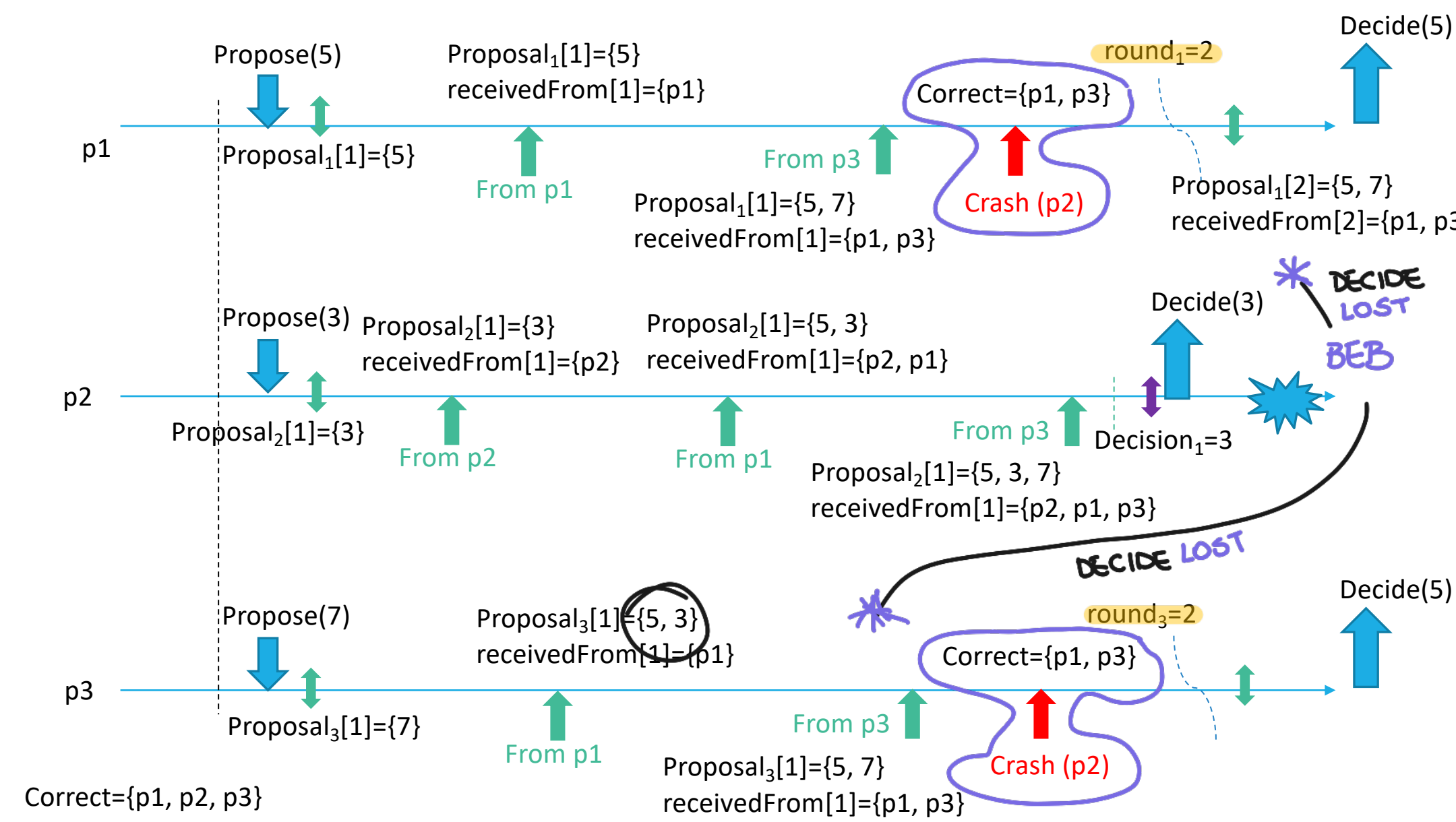
Correctness and Performance

Correctness

- Validity and integrity follow from the properties on the communication channels
- Termination. At most after N rounds processes decide
- Agreement. The same deterministic function is applied to the same values by correct processes

Performance

- Best Case (No failures). One communication round ($2N^2$ messages)
- Worst case (n-1 failures). N^2 messages exchanged for each communication step and at most N rounds => N^3 messages



Correct={p1, p2, p3}
Round=1
Decision=⊥
receivedFrom[0]={p1, p2, p3}
Proposals[0]={}

If you decide and you fail → you decision influence the result

Uniform Consensus Specification

Module 5.2: Interface and properties of uniform consensus

Module:

Name: UniformConsensus, **instance** *uc*.

Events:

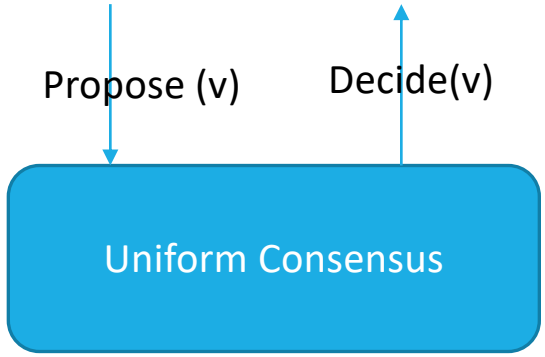
Request: ⟨ *uc*, *Propose* | *v* ⟩: Proposes value *v* for consensus.

Indication: ⟨ *uc*, *Decide* | *v* ⟩: Outputs a decided value *v* of consensus.

Properties:

UC1–UC3: Same as properties C1–C3 in (regular) consensus (Module 5.1).

UC4: *Uniform agreement:* No two processes decide differently.



Uniform Consensus and Flooding Consensus algorithm

Does the flooding consensus algorithm (described in the previous slides) satisfy the Uniform Consensus specification?

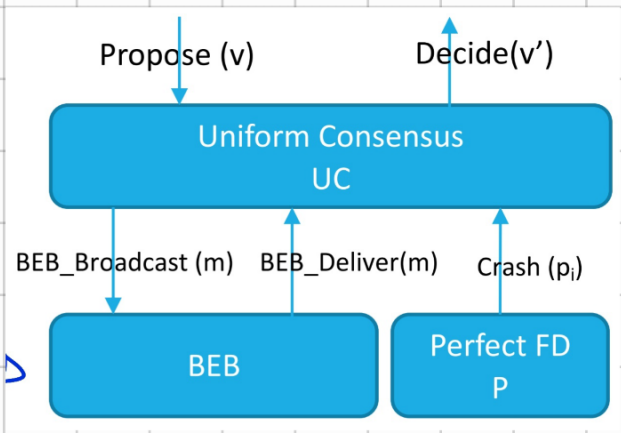
10: UNIFORM CONSENSUS

C1: *Termination:* Every correct process eventually decides some value.

C2: *Validity:* If a process decides v, then v was proposed by some process.

C3: *Integrity:* No process decides twice.

UC4: *Uniform agreement:* No two processes decide differently.



Algorithm 5.3: Flooding Uniform Consensus

Implements:
UniformConsensus, **instance** *uc*.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *uc*, *Init* $\rangle$ **do**
correct := Π;
round := 1;
decision := $\perp$;
proposalset := $\emptyset$;
receivedfrom := $\emptyset$;

upon event $\langle$ $\mathcal{P}$, *Crash* | *p* $\rangle$ **do**
correct := *correct* $\setminus$ {*p*};

upon event $\langle$ *uc*, *Propose* | *v* $\rangle$ **do**
proposalset := *proposalset* $\cup$ {*v*};
trigger $\langle$ *beb*, *Broadcast* | [PROPOSAL, 1, *proposalset*] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | *p*, [PROPOSAL, *r*, *ps*] $\rangle$ **such that** *r* = *round* **do**
receivedfrom := *receivedfrom* $\cup$ {*p*};
proposalset := *proposalset* $\cup$ *ps*;

upon *correct* $\subseteq$ *receivedfrom* $\wedge$ *decision* = $\perp$ **do**
if *round* = *N* **then**
decision := min(*proposalset*);
trigger $\langle$ *uc*, *Decide* | *decision* $\rangle$;
else
round := *round* + 1;
receivedfrom := $\emptyset$;
trigger $\langle$ *beb*, *Broadcast* | [PROPOSAL, *round*, *proposalset*] $\rangle$;

I TAKE THE DECISION ONLY WHEN I HAVE ALL THE DECISIONS, AFTER m ROUND

do not depend on the round

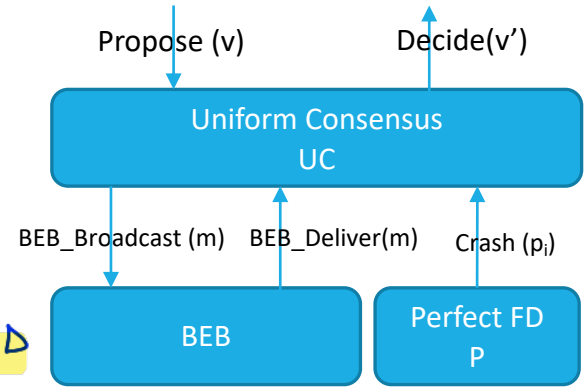
No more related to the round

= TRUE

Decision only at the end

Cleaned at the beginning of each round

Uniform Consensus Implementation in Synchronous Systems



Algorithm 5.3: Flooding Uniform Consensus

Implements:
UniformConsensus, **instance** *uc*.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *uc*, *Init* $\rangle$ **do**
correct := Π;
round := 1;
decision := $\bot$;
proposalset := $\emptyset$;
receivedfrom := $\emptyset$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
correct := *correct* $\setminus$ $\{p\}$;

upon event $\langle$ *uc*, *Propose* | v $\rangle$ **do**
proposalset := *proposalset* $\cup$ $\{v\}$;
trigger $\langle$ *beb*, *Broadcast* | [PROPOSAL, 1, *proposalset*] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [PROPOSAL, r, ps] $\rangle$ **such that** r = *round* **do**
receivedfrom := *receivedfrom* $\cup$ $\{p\}$;
proposalset := *proposalset* $\cup$ ps;

upon *correct* $\subseteq$ *receivedfrom* $\wedge$ *decision* = $\bot$ **do**
if *round* = N **then**
decision := min(*proposalset*);
trigger $\langle$ *uc*, *Decide* | *decision* $\rangle$;
else
round := *round* + 1;
receivedfrom := $\emptyset$;
trigger $\langle$ *beb*, *Broadcast* | [PROPOSAL, *round*, *proposalset*] $\rangle$;

I TAKE THE DECISION ONLY WHEN I HAVE ALL THE DECISIONS, AFTER N ROUND

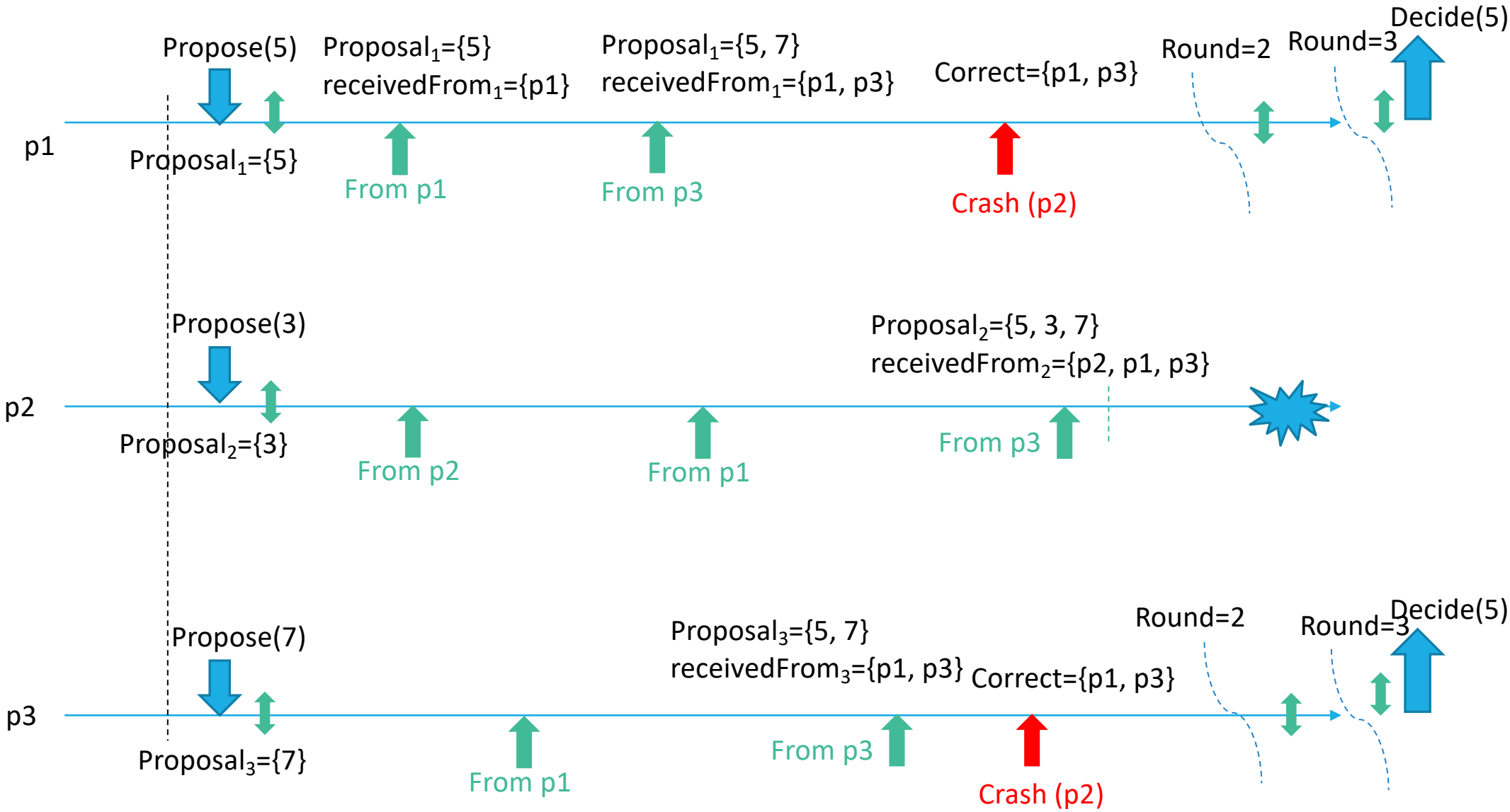
do not depend on the round

No more related to the round

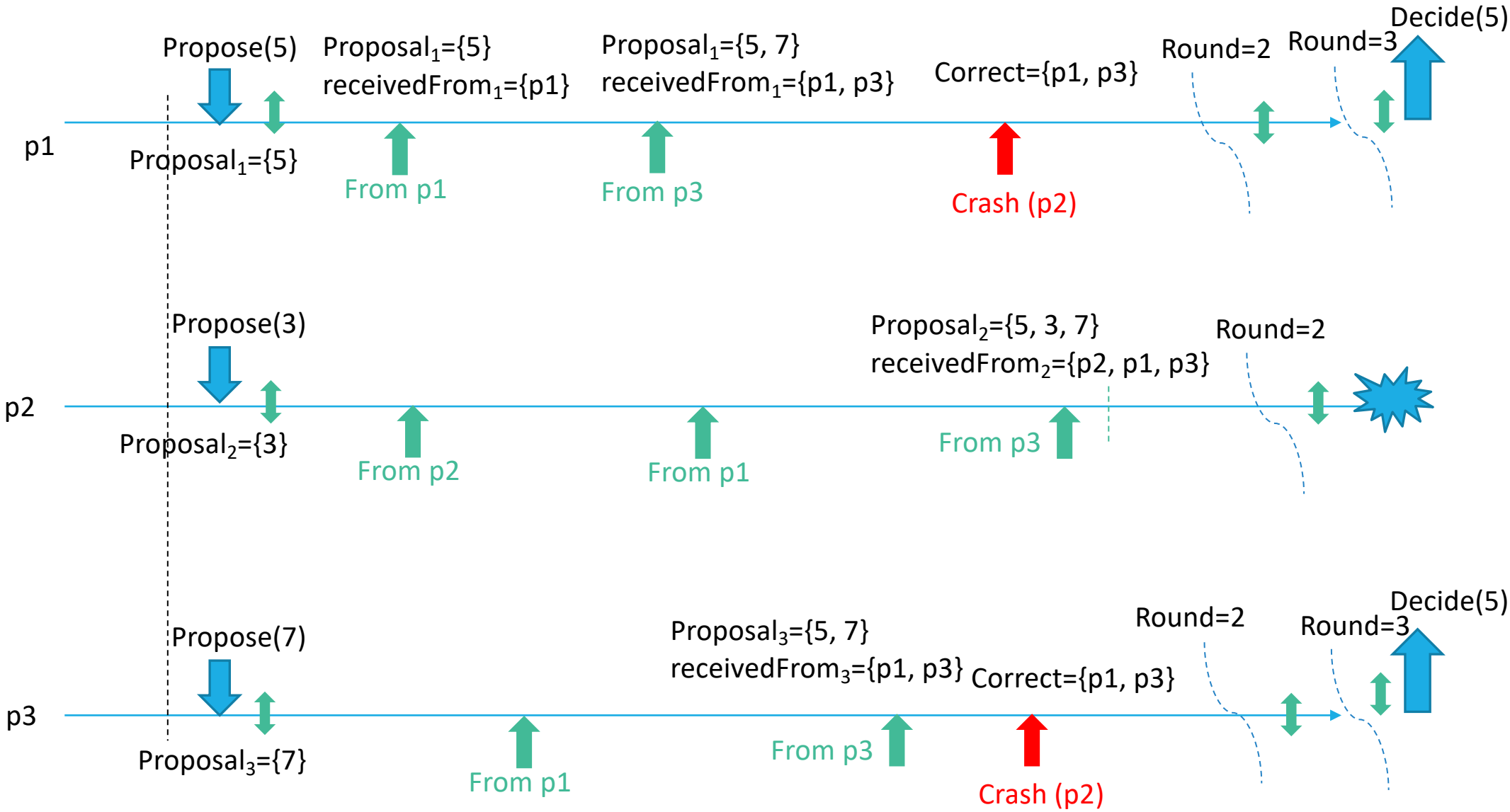
= TRUE

Decision only at the end

Cleaned at the beginning of each round



Correct={p1, p2, p3}
Round=1
Decision=⊥
receivedFrom={}
Proposals={}



Correct={p1, p2, p3}
Round=1
Decision=⊥
receivedFrom={}
Proposals={}

Correctness and Performance

Correctness

- The *validity* and *integrity* properties follow from the algorithm and from the properties of best-effort broadcast
- The *termination* property is ensured as all correct processes reach round N and decide in that round
  - the *strong completeness* property of the failure detector implies that no correct process waits indefinitely for a message from a process that has crashed, as the crashed process is eventually removed from *correct*.
- The *uniform agreement* holds because all processes that reach round N have the same set of values in their variable *proposalset*.

Performance

- N communication steps and O(N3) messages for all correct processes to decide

References

C. Cachin, R. Guerraoui and L. Rodrigues. Introduction to Reliable and Secure Distributed Programming, Springer, 2011

- Chapter 5, Sections 5.1.1, 5.1.2, 5.2.1, 5.2.2