

# Distributed Systems

## Master of Science in Engineering in Computer Science

AA 2018/2019

---

LECTURE 7: CONSENSUS

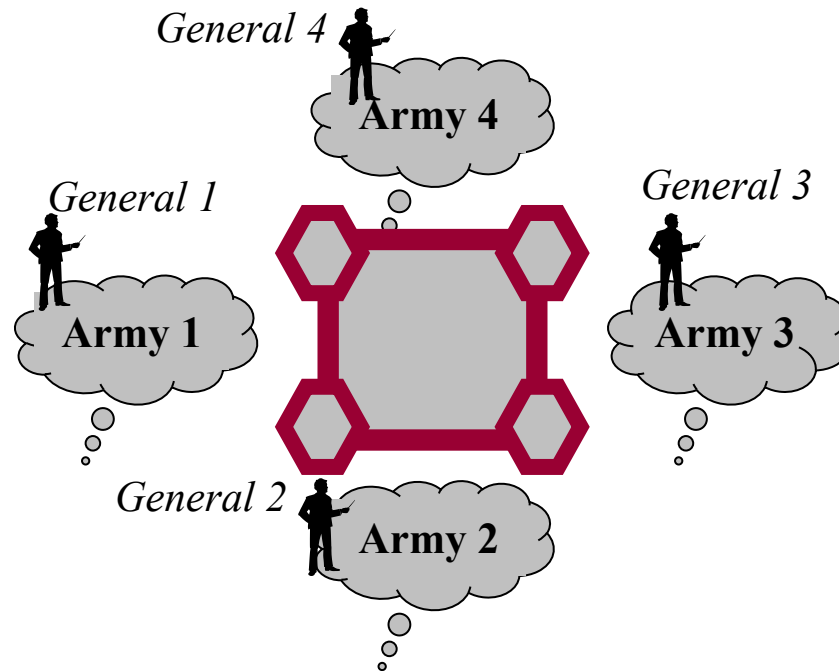
# Consensus Problem

---

- 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

---

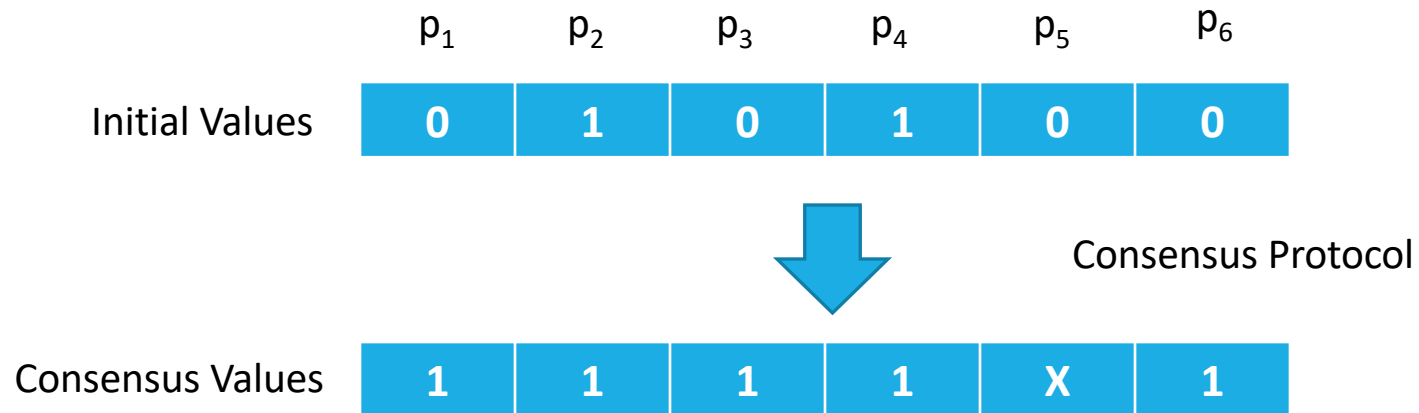


Generals have to reach consensus between attack and get back

# Consensus Definition

---

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



# Consensus Specification

---

---

## Module 5.1: Interface and properties of (regular) consensus

---

### Module:

**Name:** Consensus, **instance**  $c$ .

### Events:

**Request:**  $\langle c, \text{Propose} \mid v \rangle$ : Proposes value  $v$  for consensus.

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

### Properties:

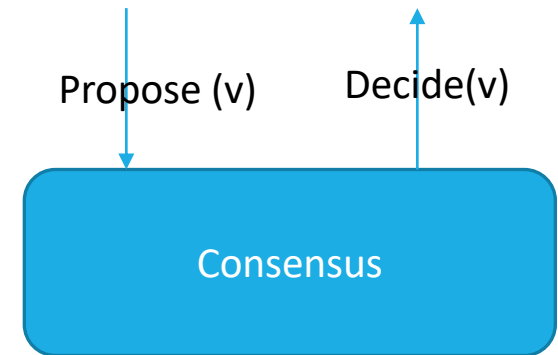
**C1: Termination:** Every correct process eventually decides some value.

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

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

**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

# 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 $\langle c, \text{Init} \rangle$ do

```

correct :=  $\Pi$ ;
round := 1;
decision :=  $\perp$ ;
receivedfrom :=  $[\emptyset]^N$ ;
proposals :=  $[\emptyset]^N$ ;
receivedfrom[0] :=  $\Pi$ ;

```

### upon event $\langle \mathcal{P}, \text{Crash} \mid p \rangle$ do

```

correct := correct  $\setminus \{p\}$ ;

```

### upon event $\langle c, \text{Propose} \mid v \rangle$ do

```

proposals[1] := proposals[1]  $\cup \{v\}$ ;
trigger  $\langle beb, \text{Broadcast} \mid [\text{PROPOSAL}, 1, \text{proposals}[1]] \rangle$ ;

```

### upon event $\langle beb, \text{Deliver} \mid p, [\text{PROPOSAL}, r, ps] \rangle$ do

```

receivedfrom[r] := receivedfrom[r]  $\cup \{p\}$ ;
proposals[r] := proposals[r]  $\cup ps$ ;

```

### upon correct $\subseteq$ receivedfrom[round] $\wedge$ decision = $\perp$ do

```

if receivedfrom[round] = receivedfrom[round - 1] then
    decision := min(proposals[round]);
    trigger  $\langle beb, \text{Broadcast} \mid [\text{DECIDED}, \text{decision}] \rangle$ ;
    trigger  $\langle c, \text{Decide} \mid \text{decision} \rangle$ ;
else

```

```

    round := round + 1;

```

```

    trigger  $\langle beb, \text{Broadcast} \mid [\text{PROPOSAL}, \text{round}, \text{proposals}[\text{round} - 1]] \rangle$ ;

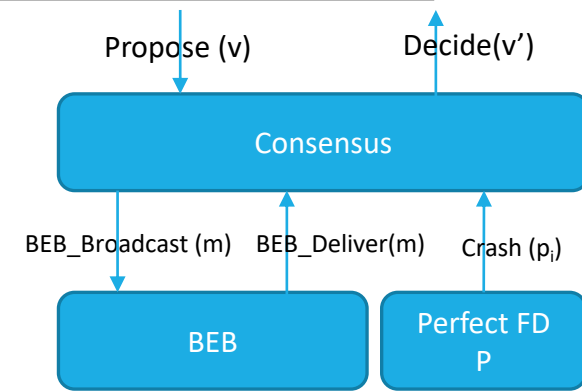
```

### upon event $\langle beb, \text{Deliver} \mid p, [\text{DECIDED}, v] \rangle$ such that $p \in \text{correct} \wedge \text{decision} = \perp$ do

```

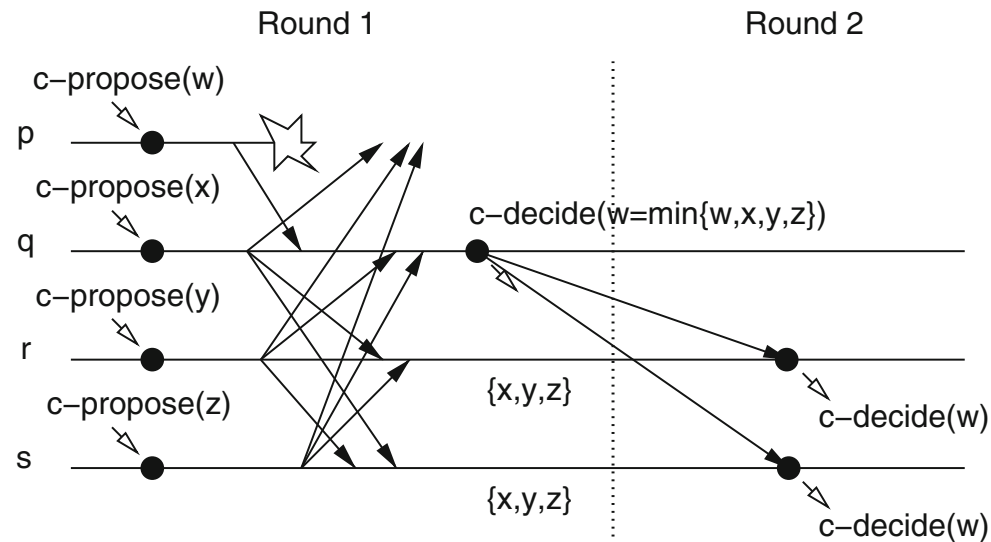
decision := v;
trigger  $\langle beb, \text{Broadcast} \mid [\text{DECIDED}, \text{decision}] \rangle$ ;
trigger  $\langle c, \text{Decide} \mid \text{decision} \rangle$ ;

```





# Example of Flooding Consensus



**Figure 5.1:** Sample execution of flooding consensus

# 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  $\Rightarrow N^3$  messages

# Uniform Consensus Specification

---

---

**Module 5.2:** Interface and properties of uniform consensus

---

**Module:**

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

**Events:**

**Request:**  $\langle uc, Propose \mid v \rangle$ : Proposes value  $v$  for consensus.

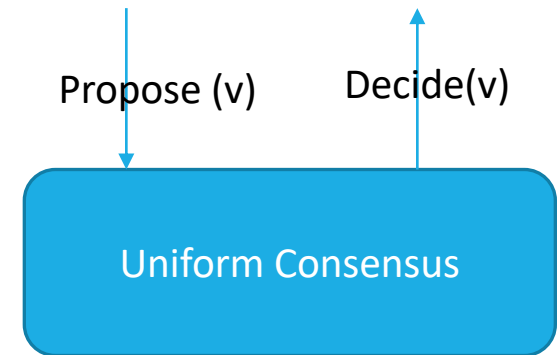
**Indication:**  $\langle uc, Decide \mid v \rangle$ : Outputs a decided value  $v$  of consensus.

**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?

# Uniform Consensus Implementation in Synchronous Systems

---

## Algorithm 5.3: Flooding Uniform Consensus

---

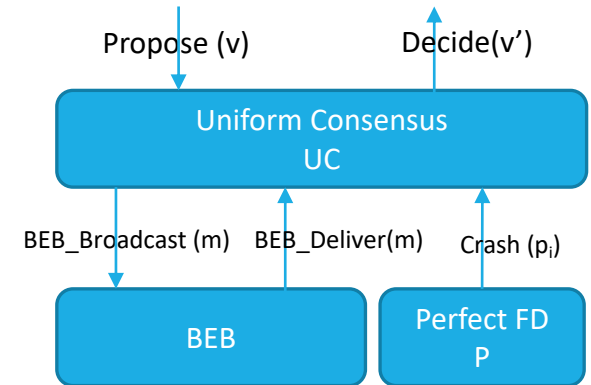
### Implements:

UniformConsensus, **instance** *uc*.

### Uses:

BestEffortBroadcast, **instance** *beb*;

PerfectFailureDetector, **instance**  $\mathcal{P}$ .



### upon event $\langle uc, Init \rangle$ do

```

correct :=  $\perp$ ;
round := 1;
decision :=  $\perp$ ;
proposalset :=  $\emptyset$ ;
receivedfrom :=  $\emptyset$ ;

```

### upon event $\langle \mathcal{P}, Crash \mid p \rangle$ do

```

correct := correct  $\setminus$  {p};

```

### upon event $\langle uc, Propose \mid v \rangle$ do

```

proposalset := proposalset  $\cup$  {v};
trigger  $\langle beb, Broadcast \mid [PROPOSAL, 1, proposalset] \rangle$ ;

```

No more related to the round

### upon event $\langle beb, Deliver \mid p, [PROPOSAL, r, ps] \rangle$ 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  $\langle uc, Decide \mid decision \rangle$ ;

```

#### else

```

round := round + 1;
receivedfrom :=  $\emptyset$ ;
trigger  $\langle beb, Broadcast \mid [PROPOSAL, round, proposalset] \rangle$ ;

```

Decision only at the end

Cleaned at the beginning of each round

# 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  $O(N^3)$  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