
Consensus

A Key Problem to Master Asynchronous Distributed Systems

Michel RAYNAL
raynal@irisa.fr

IRISA
Campus Universitaire de Beaulieu - 35042 Rennes Cedex - France

WHAT IS THE PROBLEM?

Summary

1. What is the Problem?
2. Why the problem is important
3. Models of Asynchronous Distributed Systems
4. Bad News and Challenges
5. A Few Important Results
6. $\diamond S$ -based Consensus Protocols in the Crash/no Recovery Model
7. Conclusion

What is the **CONSENSUS** Problem? (1)

- A predetermined set of processes: p_1, p_2, \dots, p_n
- Each process p_i proposes a value v_i
- Processes have to agree on a value

What is the **CONSENSUS** Problem? (2)

- **Termination**: Every **correct** process eventually decides some value
- **Validity**: If a process decides v , then v was proposed by some process
- **Agreement**: No two **correct** processes decide differently
- **Uniform Agreement**: No two (**correct** or not) processes decide differently.

What is the **CONSENSUS** Problem? (3)

As we can see, the definition of **CONSENSUS** heavily depends on what is a **correct process**

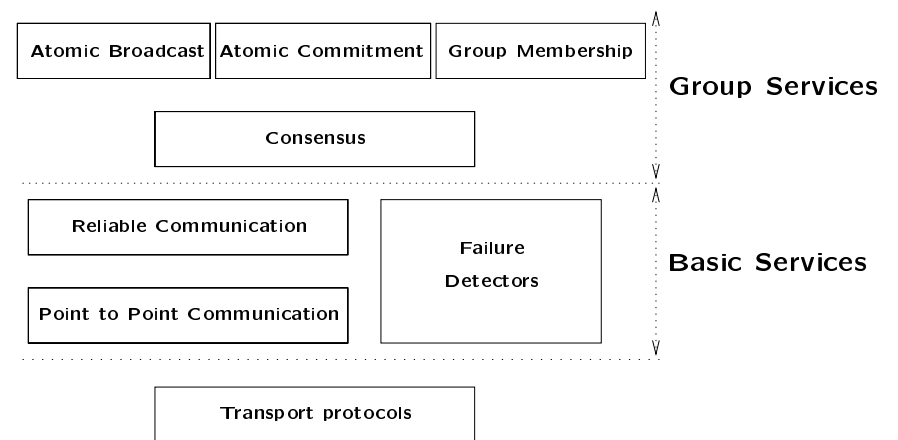
Why **CONSENSUS** is a Fundamental Problem? (1)

Within a world (system) dominated by uncertainty (asynchrony, failures)

CONSENSUS

provides **correct processes** with a **single view** of the system

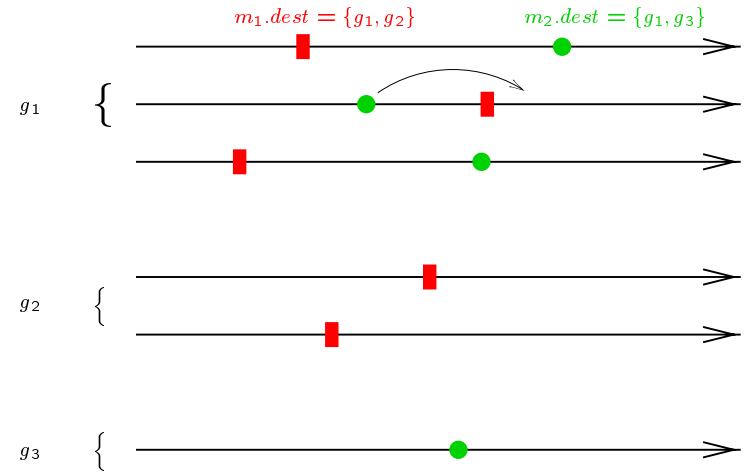
Why **CONSENSUS** is a Fundamental Problem? (2)



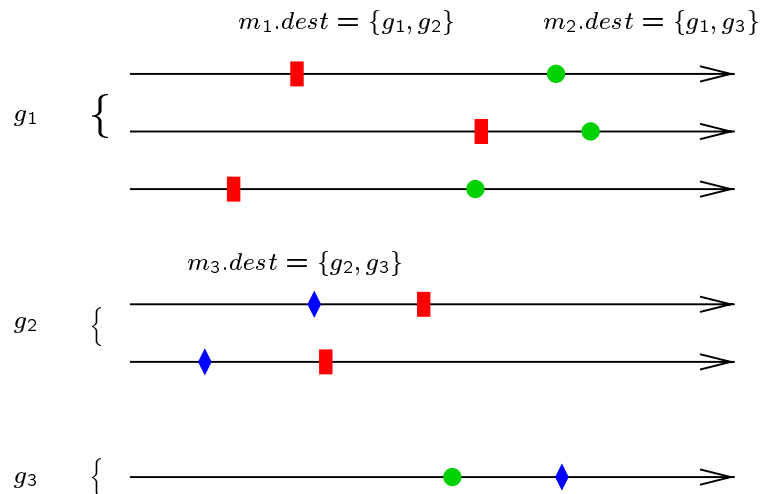
Why **CONSENSUS** is Practically Important? (1)

- Atomic Braodcast
- Atomic Multicast
- Non-Blocking Atomic Commit
- etc.

Why **CONSENSUS** is Practically Important? (2)



Why **CONSENSUS** is Practically Important? (3)



Why **CONSENSUS** is Practically Important? (4)

TO-Multicast =
Reliable Multicast + Global Total Order

Why **CONSENSUS** is Practically Important? (5)

NBAC in Asynchronous Distributed Systems

- **Termination** : If the set of correct processes launch the commitment protocol (each of them voting YES or NO), then each of them will eventually decide a value.
- **Integrity** : A process decides at most once.
- **Validity** : The decision value is COMMIT or ABORT.
- **Uniform Agreement** : All the processes that decide, decide the same value.
- **Justification** : If the decision value is COMMIT, then all processes have voted YES.
- **Obligation** : If all processes vote YES and they do not suspect each other then the decision value is COMMIT.

Reducing **Atomic Commit** to **CONSENSUS**(6)

```
procedure nbac (votei):  
begin  
  Multisend votei to all processes;  
  let Cond_yes = n msgs YES have been received by pi;  
  let Cond_nof = a msg NO has been received  
                 or a failure has been suspected;  
  wait ( Cond_yes or Cond_nof);  
  case Cond_yes → local_viewi:=COMMIT;  
       Cond_nof → local_viewi:=ABORT;  
  endcase;  
  decidedi:=Consensus(local_viewi)  
end
```

MODELS of ASYNCHRONOUS DISTRIBUTED SYSTEMS

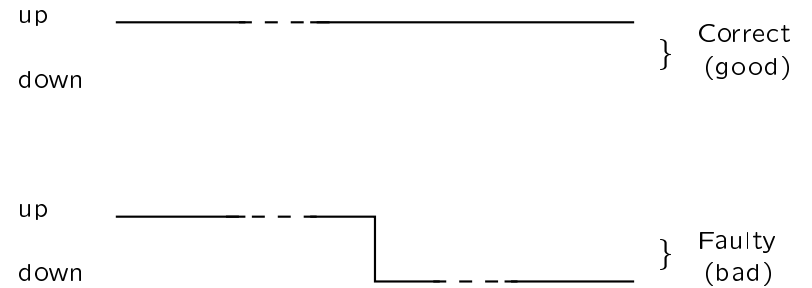
ASYNCHRONOUS Distributed Systems

- A finite set of processes: $\Pi = \{p_1, p_2, \dots, p_n\}$
- No bound on message transfer delays
- No upper bound on the time required by a process to execute a step
- A **failure model** for processes
- A **failure model** for communications

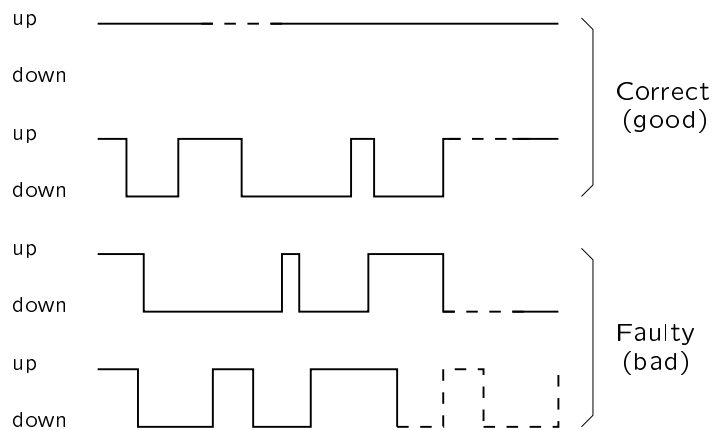
FAILURE MODEL for Processes

- No failures
- Crash failures (Crash is definitive)
- Crash-Recovery
- Byzantine failures

The Crash/No Recovery Model



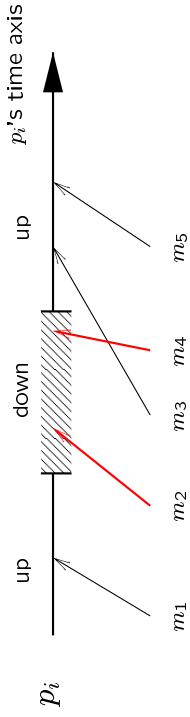
The Crash/Recovery Model



FAILURE MODEL for Communications

- Reliable channels
- Fair lossy links
- Eventually reliable channels
- etc
- \mathcal{LM} property: the last message sent by a correct process is eventually received by its (correct) destination (Dolev-Friedman-Keidar-Malki, Guerraoui-Schiper < *stubborn channel* >, Hurfin-Raynal, ...)

Net Effect of Failure Models



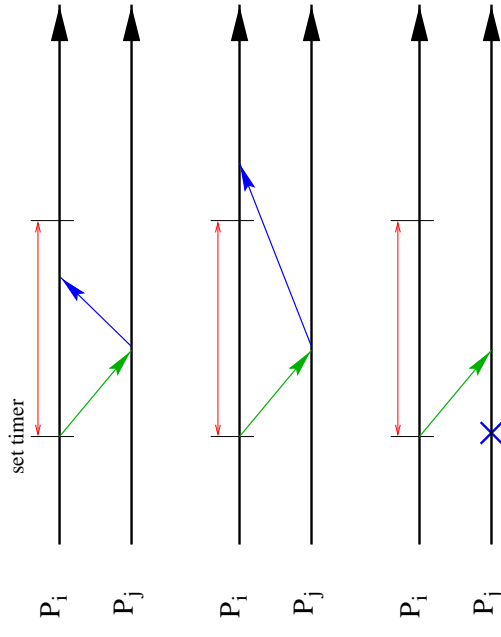
BAD NEWS and CHALLENGES

The Main Theoretical Result (1)

Fischer-Lynch-Paterson's Impossibility result (1985)

There is **no deterministic protocol** that solves the consensus problem in an asynchronous distributed system that is subject to even a **single process crash failure**

The Main Theoretical Result: Intuition (2)



There are a lot of Impossibility Results!

- Classical example:
 - ★ Problem: build a reliable channel on top of an unreliable channel
 - ★ Requires an additional assumption: the underlying channel is fair (this is assumed by the protocol, not implemented by it!)
- Why Impossibility results are good:
 - ★ Practitioners: can settle solutions that work with high probability (when scenarios that make the pb impossible to solve are unlikely to occur)
 - ★ Theoreticians: can delineate the exact borderline (minimal assumptions) beyond which there is no solution

Challenge and Motivation

- Probabilistic protocols (Ben-Or 1983, Rabin 1983)
- Consider additional Synchrony properties
 - ★ Minimal synchronism (Dolev-Dwork-Stockmeyer 1986)
 - ★ Partial synchrony (Dwork-Lynch-Stockmeyer 1987)
- Consider additional assumptions related to failure detection
 - Unreliable failure detectors (Chandra-Toueg 1991)

Unreliable Failure Detectors

- A Failure Detector is an oracle that provides processes with information on the behavior of the other processes
- This information is intrinsically Unreliable
- But to be useful, a Failure Detector has to satisfy properties:
 - ★ Completeness: Eventually, bad processes are detected as being bad
 - ★ Accuracy: Restricts the mistakes a FD can make

Failure Detectors in the C/no R Model (1)

- Each process p_i can access (read) a local variable $suspected_i$ that is assumed to contain the list of processes that have crashed (i.e., suspected to have crashed)
- According to the Completeness property and to the Accuracy property, several FD classes can be defined

\mathcal{P} and $\diamond\mathcal{P}$ Failure Detectors in the C/no R Model (2)

- \mathcal{P} (Perfect)
 - ★ **Strong Completeness**: Eventually every process that crashes is permanently suspected by every correct process
 - ★ **Strong Accuracy**: No process is suspected before it crashes
- $\diamond\mathcal{P}$ (Eventually perfect)

The previous two properties are not necessarily satisfied from the beginning, but there is a time after which they are satisfied

$\diamond\mathcal{S}$ Failure Detector in the C/no R Model (3)

Eventually Strong Failure Detector $\diamond\mathcal{S}$

- **Strong Completeness**: Eventually every process that crashes is permanently suspected by every correct process
- **Eventual Strong Accuracy**: There is a time after which **SOME** correct process is not suspected by any correct process

Implementing Failure Detectors

- General case: **Approximate (Best effort)** implementations (based on timeouts)

Quality of Service of the underlying failure detector
 $0 \leq QoS \leq 1$

$QoS = 0$: means that FD never satisfies its properties
 $QoS = \text{some threshold}$: means that FD satisfies its properties
 $QoS = 1$: the FD behaves as a perfect FD

- Particular cases: **Correct** when we can benefit from some knowledge on the upper layer application or on the underlying system

$\diamond\mathcal{S}$: Timeout Based Implementation

- **Strong Completeness**: Eventually, every crashed process is permanently suspected by every correct process.
Can be ensured.
- **Eventually Weak Accuracy**: There is a time after which some correct process is never suspected by any correct process.
Cannot be ensured.

Implementing a $\diamond\mathcal{P}$ Failure Detector (1)

- Upper layer assumption: **Communication is FAIR**

A correct process can receive **at most m messages** from any one of its correct neighbors without receiving **at least one message** from every other correct neighbor

- m can be:

- ★ Fixed, known and the same for all processes, or
- ★ Variable (but eventually stable), unknown and non-uniform

Implementing a $\diamond\mathcal{P}$ Failure Detector (2)

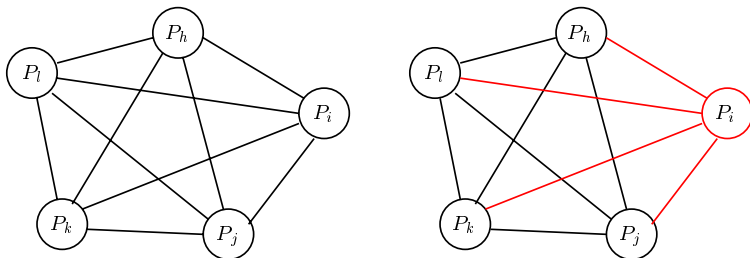
$count_i[j, k]$: # messages received from p_j since the last message received from p_k

```

upon reception of msg from  $p_j$ 
  if ( $p_j \in suspected_i$ ) then  $suspected_i \leftarrow suspected_i - \{p_j\}$  endif;
  forall  $k \neq j$  do
    if ( $p_k \notin suspected_i$ ) then
       $count_i[j, k] \leftarrow count_i[j, k] + 1$ ;
      if ( $count_i[j, k] > m$ ) then
         $suspected_i \leftarrow suspected_i \cup \{p_k\}$ 
      endif;
    endif
     $count_i[k, j] \leftarrow 0$ 
  endforall
  
```

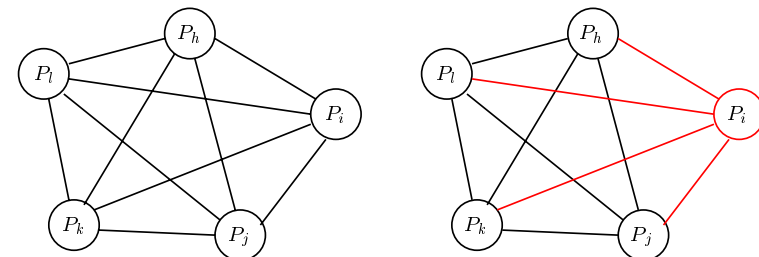
Implementing a Failure Detector (1)

Difficulty to implement $\diamond\mathcal{P}$ in the general case: $O(n^2)$



Implementing a Failure Detector (2)

Why the properties of $\diamond\mathcal{S}$ are easier to satisfy in the general case: $O(n)$



A FEW IMPORTANT RESULTS

CONSENSUS in the **Crash/no Recovery** Model

- $\diamond S$ is the **weakest** failure detector to solve Consensus (Chandra-Hadzilacos-Toueg, 1992)
- $\diamond S$ requires a **majority** of processes to be **correct** (Chandra-Toueg, 1991)
- Any protocol that solves consensus with $\diamond S$ solves also uniform consensus (Guerraoui, 1995)
- Any protocol based on $\diamond S$ can lose its **Liveness** property when the properties associated with $\diamond S$ are not satisfied, but it never lose its **Safety** property

CONSENSUS in the **Crash/Recovery** Model (1)

- **Stable storage:**
 - Is it always necessary?
 - Log as few data as possible
 - Reduce the number of logging operations
- **Failure detection:**
 - Which properties has to satisfy the underlying failure detector?

CONSENSUS in the **Crash/Recovery** Model (2)

Aguilera-Chen-Toueg 1998

n_a = min # of processes that are guaranteed to remain up (from the beginning)

n_b = max # of processes that are bad (eventually down or forever oscillating)

- $n_a > n_b$: Consensus **can be solved** without stable storage (with an appropriate FD)
- $n_a \leq n_b$: Consensus **can not be solved** without stable storage (even with $\diamond P$)

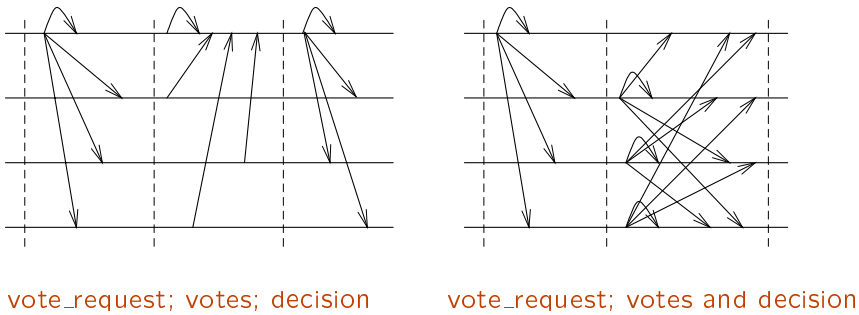
Going to the past: Skeen's Protocols (1)

Skeen (1981) has investigated two messages exchange patterns to design protocols solving the Non-Blocking Atomic Commit Problem in the context of distributed systems equipped with perfect reliable failure detectors

◇ S-based CONSENSUS PROTOCOLS in the CRASH/no RECOVERY MODEL

Going to the past: Skeen's Protocols (2)

Centralized scheme vs Decentralized scheme



- Challenge: Ensure Agreement and Termination:
 - ★ Adversary: Despite asynchrony
 - ★ Adversary: Despite the inherent unreliability (mistakes) of the underlying failure detector
 - ★ Allied: Assuming the properties of the underlying failure detector are satisfied

Underlying Principles (2)

- Challenge
- How to take up this challenge:
 - ★ Each process p_i manages a local variable est_i , representing its current estimate of the consensus value. This value will **converge** to the decision value.
 - ★ Protocols proceed in **consecutive asynchronous rounds**. During round r , a predetermined process p_c (e.g., $c = (r \bmod n) + 1$) acts a particular role. This is the
Rotating Coordinator Paradigm
(This paradigm fits with the **Eventual Weak Accuracy** property of $\diamond S$: there will be a round during which the current coordinator will not be suspected)

Underlying Principles (3)

- Challenge
- How to take up the Challenge:
 - ★ During a round: the current coordinator selects a value and the protocol tries to impose this value as the decision value
 - ★ Crashes (or crash suspicions) can be dealt with by moving to the next round (Liveness) (easy part)
 - ★ But it is possible not all processes decide during the same round
If two processes decide during distinct rounds they have to decide the same value (Safety)! (difficult part)

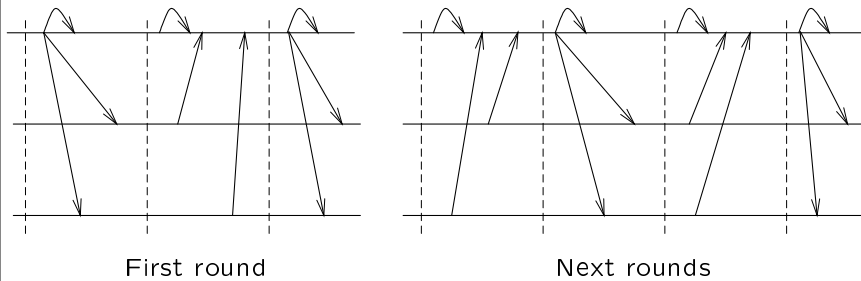
Underlying Principles (4)

Adversaries (asynchrony, unreliable failure detectors) make Consensus protocols far from being trivial!

$\diamond S$ -based Protocols

- Centralized protocol: Chandra-Toueg 1991
- Decentralized protocols:
 - ★ Schiper (1996):
Well suited to resist erroneous suspicions (when a minority of failure detector modules makes mistakes)
(There is a *majority of suspicions* condition for a process to progress to the next round)
Intuition: this protocol basically *does not trust the FD*
 - ★ Hurfin-Raynal (1997):
Efficient when the failure detector makes no erroneous suspicions (whether there are failures or not)
Intuition: this protocol basically *does trust the underlying FD*

Chandra-Toueg's Centralized Protocol



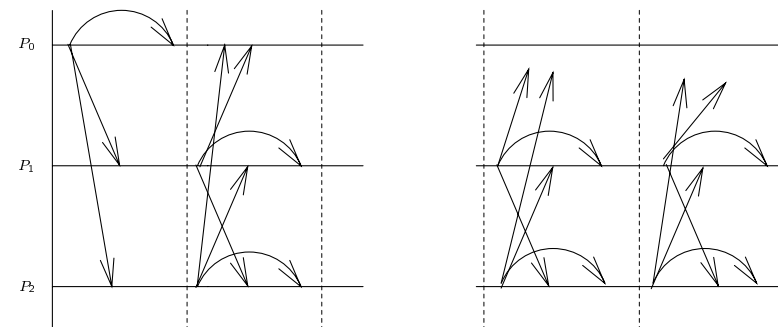
Decentralized Consensus Protocols

- Schiper (1996):
Well suited to resist erroneous suspicions (when a minority of failure detector modules makes mistakes)
(There is a *majority of suspicions* condition for a process to progress to the next round)
Intuition: this protocol basically *does not trust the FD*
- Hurfin-Raynal (1997):
Efficient when the failure detector makes no erroneous suspicions (whether there are failures or not)
Intuition: this protocol basically *does trust the underlying FD*

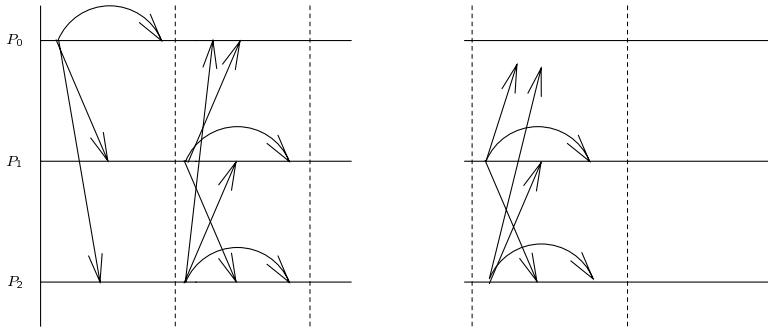
Schiper and Hurfin-Raynal Decentralized Protocols

- They behave as Skeen's decentralized scheme when there are neither failures, nor erroneous suspicions (i.e., in a *perfect world*)
- They behave
 - ★ Differently from Skeen's scheme
 - ★ Differently from each other
 when there are failures or erroneous suspicions

Schiper Protocol with a Perfect FD



Hurfin-Raynal Protocol with a Perfect FD



Comparing the Protocols (1)

- Assumptions:
 - ★ Wrt the underlying failure detector: FD behaves perfectly (freedom wrt FD)
 - ★ Wrt asynchrony : all messages take one time unit (freedom wrt asynchrony)
- Processes p_1, \dots, p_7 (p_1 being the first coordinator, ...)
- Failure scenarios:
 - FP_0 : all processes are correct
 - FP_1 : all processes but p_1 are correct
 - FP_x : a minority of processes not including p_1 is faulty

Comparing the Protocols (2)

	FP_0	FP_x	FP_1
CT	3	3	4
SC	2	2	4
HR	2	2	3

Hurfin-Raynal PROTOCOL

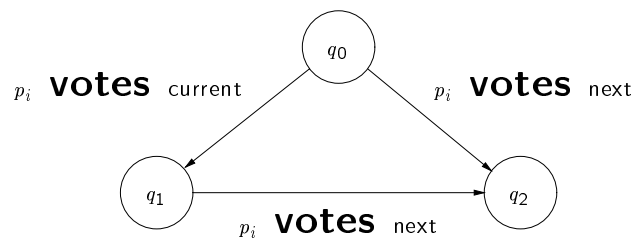
HR Protocol: Principles (1)

- The use of the **rotating coordinator** paradigm. As the other protocols, the fast consensus protocol proceeds in consecutive asynchronous rounds, each round being coordinated by a predetermined process. Combined with the properties of $\diamond S$, this ensures that there will eventually be a round in which the coordinator will not be suspected.
- The use of the **voting** paradigm. Used at each round, it consists of:

HR Protocol: Principles (2)

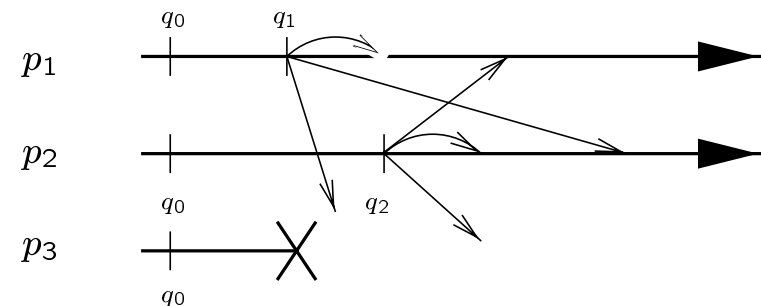
- **Two votes.** A **current** (resp. **next**) vote means that the issuing process is in favor of deciding at the current round (resp. proceeding to the next round).
- **A majority rule.** A process decides at the current round (resp. progresses to the next round) as soon as it has received a majority of **current** (resp. **next**) votes.
- **A simple automaton.** This automaton defines the state of a process with respect to the votes it issues.
- **The possibility for a process to change its mind.** A process that voted **current** can later issue a **next** vote. This prevents processes from blocking.

HR Protocol: The Finite State Automaton



$state_i = q_0$: p_i has not yet voted
 $state_i = q_1$: p_i has voted **current**
 $state_i = q_2$: p_i has voted **next**

HR Protocol: Deadlock Prevention



HR Protocol: Local variables

- $est_i, state_i, suspected_i$.
- r_i : current round number
- $nb_current_i$ (resp. nb_next_i) counts the number of **current** (resp. **next**) votes received by p_i during the current round.
- rec_from_i : set composed of the process identities from which p_i has received a (**current** or **next**) vote during the current round.

HR Protocol: Behavior (1)

function **consensus**(v_i)

```
(01)  $r_i \leftarrow 0$ ;  $est_i \leftarrow v_i$ ;  
    cobegin  
(02)   || upon reception of decide( $p_k, r_k, est_k$ )  
(03)     send decide( $p_i, r_k, est_k$ ) to  $\Pi - \{p_i, p_k\}$ ;  
(04)     return( $est_k$ )  
  
(05)   || loop % Execution of a round % endloop  
    coend
```

HR Protocol: Execution of a round

```
(01)  $r_i \leftarrow r_i + 1$ ;  $c \leftarrow (r_i \bmod n) + 1$ ;  
(02)  $state_i \leftarrow q_0$ ;  $rec\_from_i \leftarrow \emptyset$ ;  $nb\_next_i \leftarrow 0$ ;  
(03) if ( $i = c$ )  
(04)   then send current( $p_i, r_i, est_i$ ) to itself;  
(05)      $nb\_current_i \leftarrow -1$   
(06)   else  $nb\_current_i \leftarrow 0$  endif;  
(07) while ( $nb\_next_i \leq n/2$ ) do  
(08)   % select one among 4 actions % endwhile;  
(09) if ( $state_i = q_0$ )  
(10)   then  $state_i \leftarrow q_2$ ;  
(11)     send next( $p_i, r_i, est_i, susp$ ) to  $\Pi - \{p_i\}$  endif;  
(12) if ( $state_i = q_1$ )  
(13)   then  $state_i \leftarrow q_2$ ;  
(14)     send next( $p_i, r_i, est_i, dlk\_prev$ ) to  $\Pi - \{p_i\}$  endif
```

HR Protocol: Behavior (3)

```
(01) upon reception of current( $p_k, r_i, est_k$ )  
(02)   if ( $nb\_current_i = 0$ ) then  $est_i \leftarrow est_k$  endif;  
(03)    $nb\_current_i \leftarrow nb\_current_i + 1$ ;  
(04)    $rec\_from_i \leftarrow rec\_from_i \cup \{p_k\}$ ;  
(05)   if ( $state_i = q_0$ )  
(06)     then  $state_i \leftarrow q_1$ ;  
(07)       send current( $p_i, r_i, est_i$ ) to  $\Pi - \{p_i\}$ ;  
(08)        $nb\_current_i \leftarrow nb\_current_i + 1$ ;  
(09)        $rec\_from_i \leftarrow rec\_from_i \cup \{p_i\}$  endif;  
(10)   if ( $nb\_current_i > n/2$ ) % a value is decided %  
(11)     then send decide( $p_i, r_i, est_i$ ) to  $\Pi - \{p_i\}$ ;  
(12)     return( $est_i$ ) endif
```

HR Protocol: Behavior (4)

```
(01) upon ( $p_c \in suspected_i$ )
(02)   if ( $state_i = q_0$ )
(03)     then  $state_i \leftarrow q_2$ ;
(04)       send  $next(p_i, r_i, est_i, susp)$  to  $\Pi - \{p_i\}$ ;
(05)        $nb\_next_i \leftarrow nb\_next_i + 1$ ;
(06)        $rec\_from_i \leftarrow rec\_from_i \cup \{p_i\}$  endif
```

HR Protocol: Behavior (5)

```
(01) upon reception of  $next(p_k, r_i, est_k, flag_k)$ 
(02)    $nb\_next_i \leftarrow nb\_next_i + 1$ ;
(03)    $rec\_from_i \leftarrow rec\_from_i \cup \{p_k\}$ ;
(04)   if ( $(nb\_current_i = 0) \wedge (flag_k = dlk\_prev)$ )
(05)     then  $est_i \leftarrow est_k$  endif
```

HR Protocol: Behavior (6)

```
(01) upon ( $(state_i = q_1) \wedge$   
  ( $|rec\_from_i| > n/2$ )  $\wedge$   
  ( $\forall p_k: p_k \in rec\_from_i \cup suspected_i$ ))
(02)    $state_i \leftarrow q_2$ ;
(03)   send  $next(p_i, r_i, est_i, dlk\_prev)$  to  $\Pi - \{p_i\}$ ;
(04)    $nb\_next_i \leftarrow nb\_next_i + 1$ ;
```

HR Protocol - Proof: Hypotheses

- H1: There is a majority of processes that are correct (i.e., $f < n/2$)
- H2: The underlying failure detector belongs to the class $\diamond S$, i.e., it satisfies:
 - ★ H2.1: Strong Completeness (Eventually, every crashed process is permanently suspected by every correct process). And,
 - ★ H2.2: Eventual Weak Accuracy (There is a time after which some correct process is never suspected by any correct process)
- H3: Communication channels are reliable

HR Protocol - Proof: Validity

- **Theorem 1:** If a process p_i decides v , then v was proposed by some process

HR Protocol - Proof: Termination

- **Lemma 1:** If no process decides during a round $r' \leq r$, then all correct process start round $r + 1$
- **Lemma 2:** If during a round r , no process moves from q_0 to state q_2 , then no process moves from q_1 to q_2
- **Theorem 2:** Every correct process eventually decides some value

HR Protocol - Proof: Uniform Agreement

- **Lemma 3:** During round r , we have $(nb_current_i \neq 0) \Rightarrow (est_i = est_c)$, where p_c is the current coordinator.
- **Lemma 4:** Any process p_i that sends a decide message labeled with the round number r , decides value est_c (where p_c is the coordinator of round r).
- **Lemma 5:** If process p_i decides v and sends a decide message labeled with the round number r , then all processes p_j that start round $r + 1$ do so with $est_j = v$.
- **Theorem 3:** No two processes decide different values.

HR Protocol - Properties

- The case of FIFO channels
- Proceeding to the next round

CONCLUSION

To Conclude

- Consensus is a fundamental problem of computer science (distributed systems)
- Consensus provides a building block on top of which solutions to practical problems (Atomic Commit, VSC, TO-Broadcast, TO-Multicast,...) can be built
- Consensus allows to understand why a solution works or why it does not
- Understand Consensus means understand Liveness property
- Understand Consensus Protocols means understand how to preserve a Safety property despite asynchrony and failures