Confiabilidade de Sistemas Distribuídos
Dependable Distributed Systems

DI-FCT-UNL, Henrique Domingos, Nuno Preguiça

Lect. 4
Randomized Consensus

2015/2016, 2nd SEM

MIEI
Mestrado Integrado em Engenharia Informática

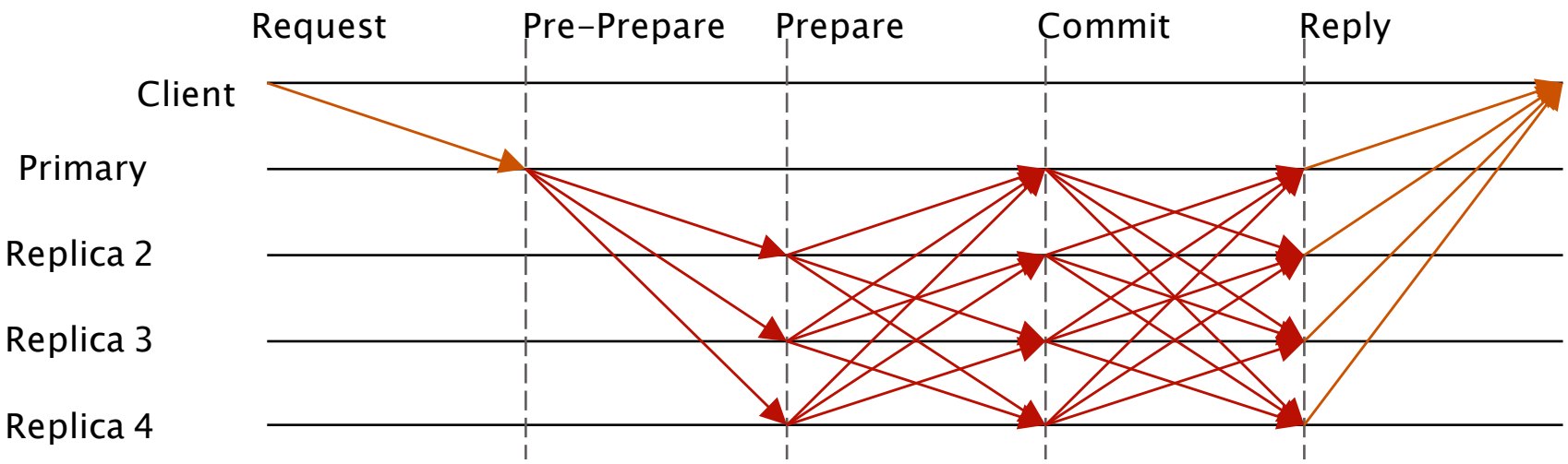
Last lecture: Byzantine fault model

- Processes that fail can exhibit arbitrary behavior
  - Return wrong replies
  - Take too long to execute a computation step
  - Do not follow the communication protocol
  - Collude with other processes

Byzantine fault-tolerant read/write register

- ABD with Byzantine quorums
  - Client sign requests
  - Larger quorum

Byzantine fault-tolerant SMR: PBFT



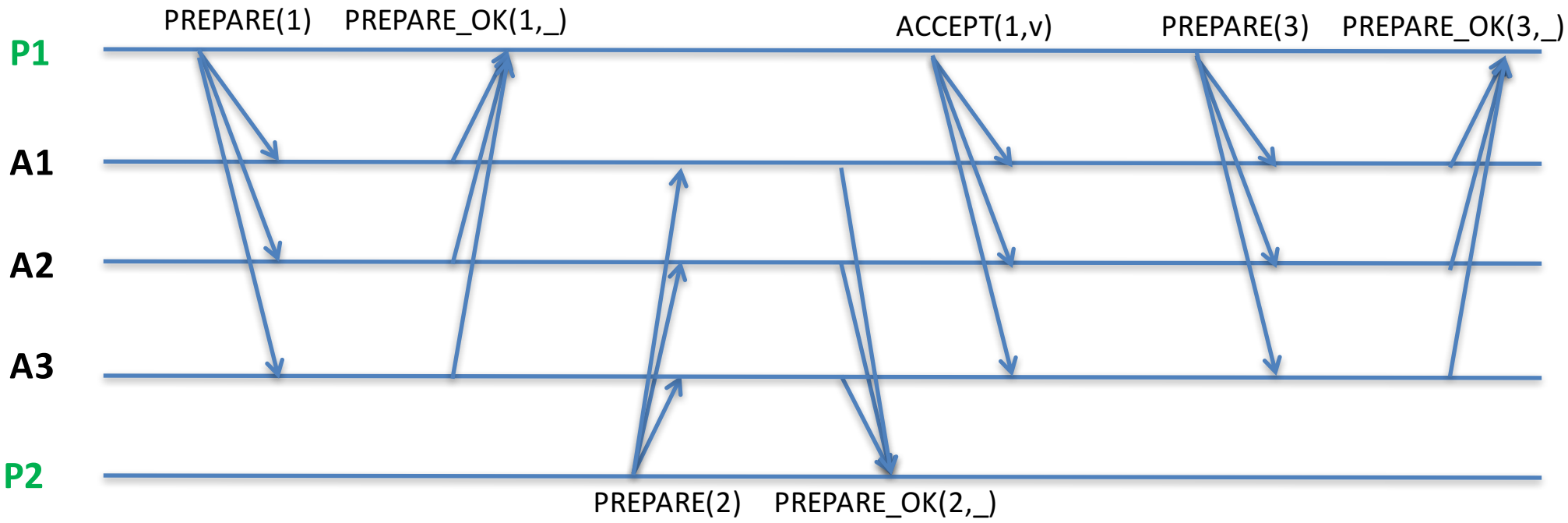
Today

- Randomized consensus

Consensus

- Inputs: each process has its initial proposal in variable v_i
- Outputs: each process has an output variable $decision_i$, initially *null*
- C1 [Validity] If all processes have $v_i = v$, then v is the only allowed output
- C2 [Agreement] Two correct processes cannot decide different values
- C3 [Termination] All correct processes eventually decide
- C4[integrity] If a correct process decides v, then v was the initial proposal of some process

Liveness not guaranteed: Paxos



Source of the impossibility

- Consider n process, 1 possible fault
- Each process votes on his proposal
- Decide if a majority of processes vote in the same value
- Need to proceed when receiving n-1 votes, but we can have a draw.
  - Solution: execute another round
  - The same outcome may occur.

Randomized Consensus

- Getting around FLP negative result for asynchronous consensus:
  - weaken the termination condition: non-faulty processors must decide with some nonzero probability
  - keep the same agreement and validity conditions

Consensus

- Inputs: each process has its initial proposal in variable v_i
- Outputs: each process has an output variable $decision_i$, initially *null*
- C1 [Validity] If all processes have $v_i = v$, then v is the only allowed output
- C2 [Agreement] Two correct processes cannot decide different values
- C3 [Termination] With probability 1, all correct processes eventually decide
- C4[integrity] If a correct process decides v, then v was the initial proposal of some process

Randomized algorithm

- M. Ben Or. “Another advantage of free choice: completely asynchronous agreement protocols” (PODC 1983, pp. 27-30)
  - exponential number of operations per process
  - With n processes, it tolerates f < n/2 faults

Algorithm idea

- An infinite repetition of asynchronous rounds
  - in round r, p only handles messages with timestamp r
  - each round has two phases
  - in the first phase, each p broadcasts its proposal
    - The proposal is a function of the values collected in the second phase of last round (in the first round, it is the input)
  - in the second phase, each p broadcasts a value which is a function of the values collected in the first phase
  - decide stutters

Ben Or’s algorithm

State:

Input → boolean

output → boolean

preference → boolean

round: integer

Ben Or's algorithm

```
preference ← input
round ← 1
while true do
        send (1, round, preference) to all processes
        wait to receive n − f (1, round, ∗) messages
        if received n-f (1, round, v) messages then
                send (2, round, v, ratify) to all processes
        else
                send (2, round, ⊥) to all processes
        end
        wait to receive n − f (2, round, ∗) messages
        if received a (2, round, v, ratify) message then
                preference ← v
                if received n - f (2, round, v, ratify) messages then
                        output ← v
                end
        else
                preference ← CoinFlip()
        end
        round ← round + 1
end
```

Validity

```
while true do
        send (1, round, preference) to all processes
        wait to receive n − f (1, round, ∗) messages
        if received n-f (1, round, v) messages then
                send (2, round, v, ratify) to all processes
        else
                send (2, round, ⊥) to all processes
        end
        wait to receive n − f (2, round, ∗) messages
        if received a (2, round, v, ratify) message then
                preference ← v
                if received n - f (2, round, v, ratify)
messages then
                        output ← v
                end
        else
                preference ← CoinFlip()
        end
        round ← round + 1
end
```

C1 [Validity] If all processes have $v_i = v$, then v is the only allowed output

- If all processes start with the same value, that value is decided

Agreement

```
while true do
        send (1, round, preference) to all processes
        wait to receive n − f (1, round, ∗) messages
        if received n-f (1, round, v) messages then
                send (2, round, v, ratify) to all processes
        else
                send (2, round, ⊥) to all processes
        end
        wait to receive n − f (2, round, ∗) messages
        if received a (2, round, v, ratify) message then
                preference ← v
                if received n -f (2, round, v, ratify)
messages then
                        output ← v
                end
        else
                preference ← CoinFlip()
        end
        round ← round + 1
end
```

C2 [Agreement] Two correct processes cannot decide different values

- Lemma: In a given round, a single value can be ratified, at most
- Corollary: In a given round, a single value can be decided at most

Validity

```
while true do
        send (1, round, preference) to all processes
        wait to receive n − f (1, round, ∗) messages
        if received n-f (1, round, v) messages then
                send (2, round, v, ratify) to all processes
        else
                send (2, round, ⊥) to all processes
        end
        wait to receive n − f (2, round, ∗) messages
        if received a (2, round, v, ratify) message then
                preference ← v
                if received n - f (2, round, v, ratify)
messages then
                        output ← v
                end
        else
                preference ← CoinFlip()
        end
        round ← round + 1
end
```

C2 [Agreement] Two correct processes cannot decide different values

- Lemma: After a process decides on a value, it is impossible to decide a different value
- Proof:
  - For deciding, n-f processes had to send ratify
  - All processes will vote the decided value, as they have received at least n - 2f ratifies

Termination

```
while true do
        send (1, round, preference) to all processes
        wait to receive n − f (1, round, ∗) messages
        if received n-f (1, round, v) messages then
                send (2, round, v, ratify) to all processes
        else
                send (2, round, ⊥) to all processes
        end
        wait to receive n − f (2, round, ∗) messages
        if received a (2, round, v, ratify) message then
                preference ← v
                if received n - f (2, round, v, ratify)
messages then
                        output ← v
                end
        else
                preference ← CoinFlip()
        end
        round ← round + 1
end
```

C3 [Termination] All correct processes eventually decide with probability that tends to 1

- For the next round
  - some processes will vote on a value as a result of a ratify
  - others will vote on a random value
- There is some probability that all values will be equal