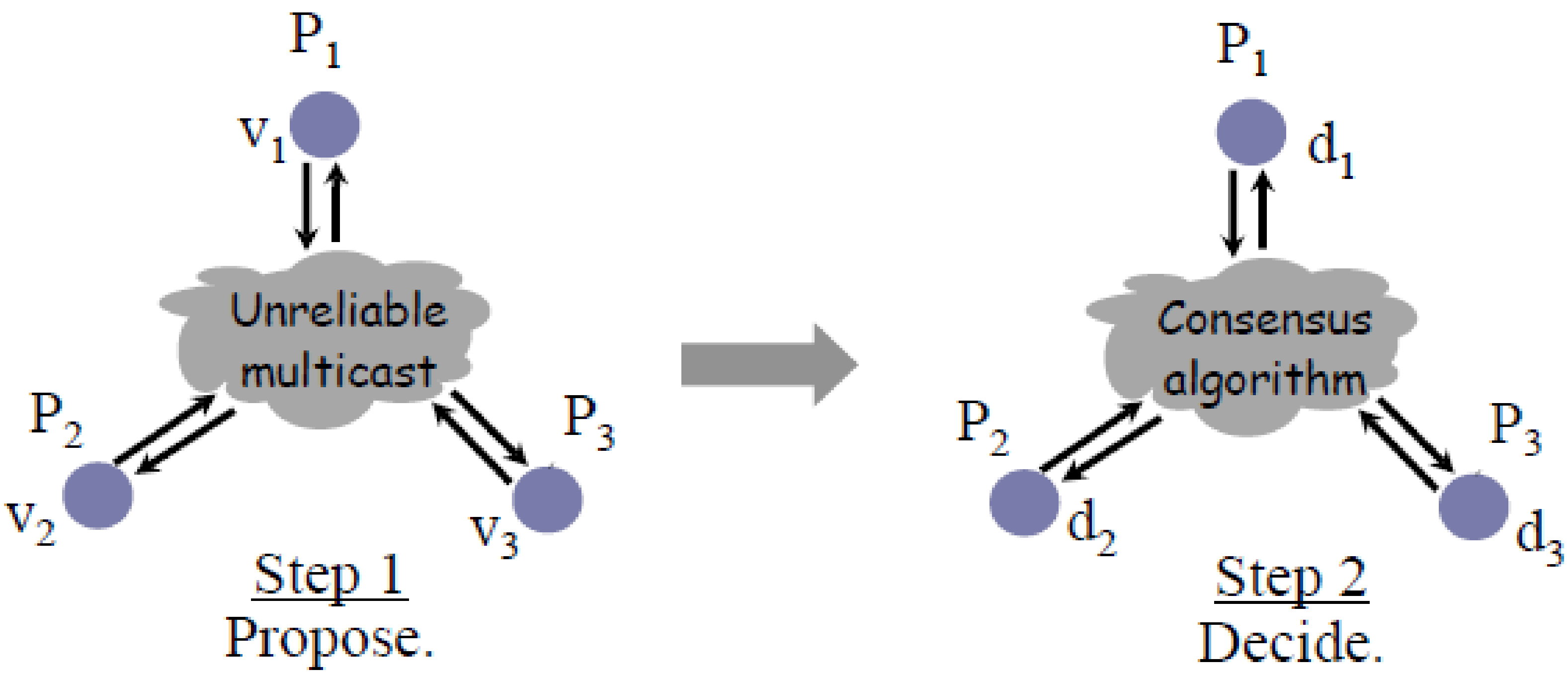
Consensus

In a distributed system, agreement among multiple processes on a single data value, despite failures.

Once they reach a decision on a value, that decision is final.

Why Consensus?

Consensus



Generalizes to N nodes/processes.

The Distributed Consensus Problem

We assume n processes, connected by a synchronous, undirected graph where each process has a unique ID.
Each process u receives an input value i_u from the set S, that is $i_u \in S$.
An algorithm solves the problem of distributed consensus if it adheres to the following specifications:

1. Agreement: No pair of processes agrees on different output values, that is, $\nexists u, v : o_u \neq o_v$
2. Validity: If all processes start with the same value $i \in S$, i.e., $\forall u \in [1, n] : i_u = i$, then value i is the only possible decision value, that is $\forall u \in [1, n] : o_u = i$
3. Termination: All processes eventually decide.

SimpleConsensus Algorithm

Each process $u \in [1, n]$ maintains a list l_u with pairs of IDs and input values. Initially the list contains only one set: the ID of u and the input value $i_u \in S$. In each round, all processes transmit the list l to their local neighborhood. When they receive list l_v from a neighbor v, they merge it with their internal list. After $\delta + 1$ rounds, all processes maintain a list containing a pair (u, i_u) for each other process of the system. Then they apply a predefined consensus rule and terminate by outputing the common value $o \in S$.

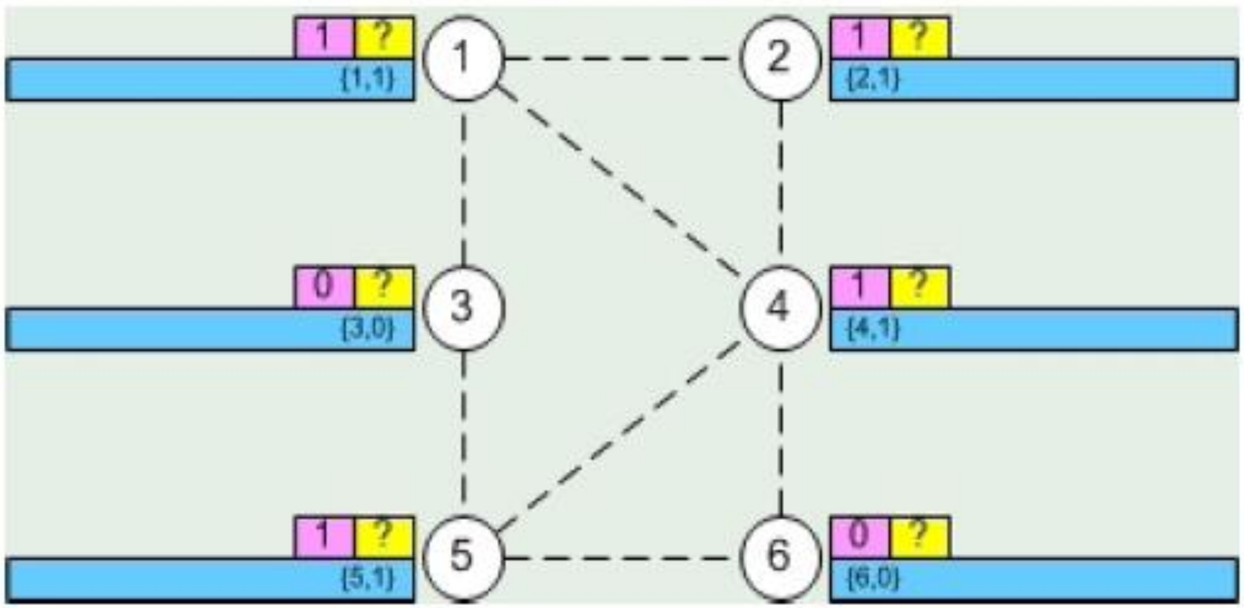
- Each process knows δ.
- The algorithm solves the consensus problem.
- The consensus rule can be: minimum value, average value, majority ...

Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

General Graph

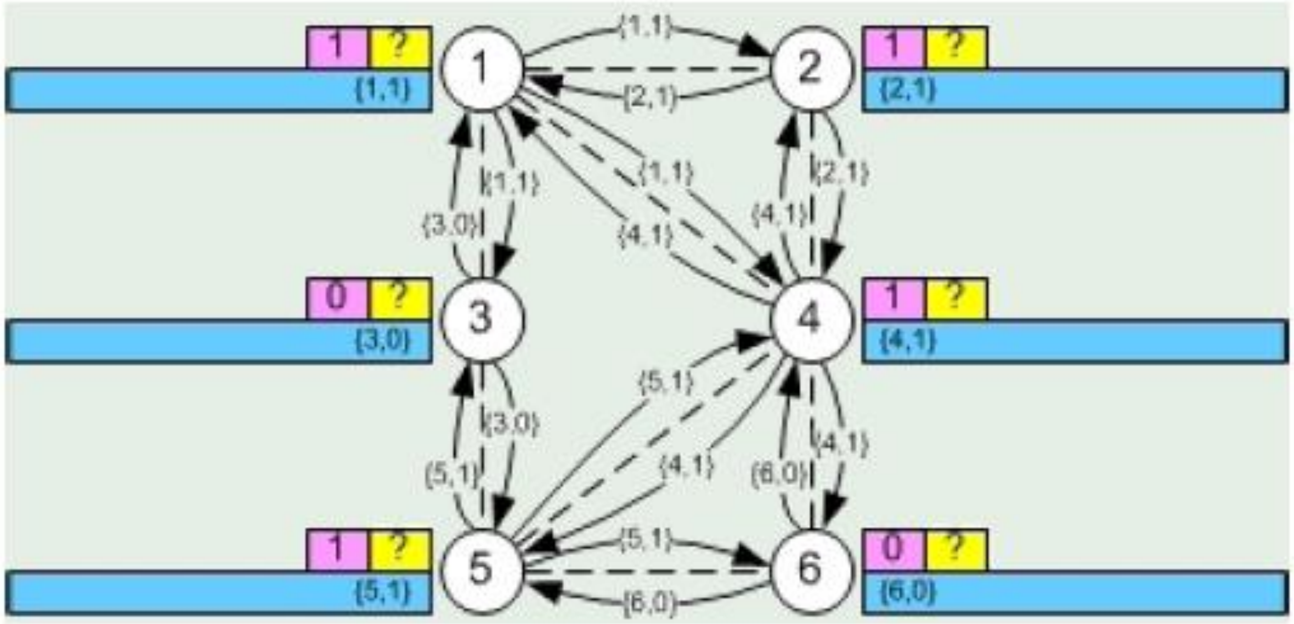


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

1st Round – message transmission

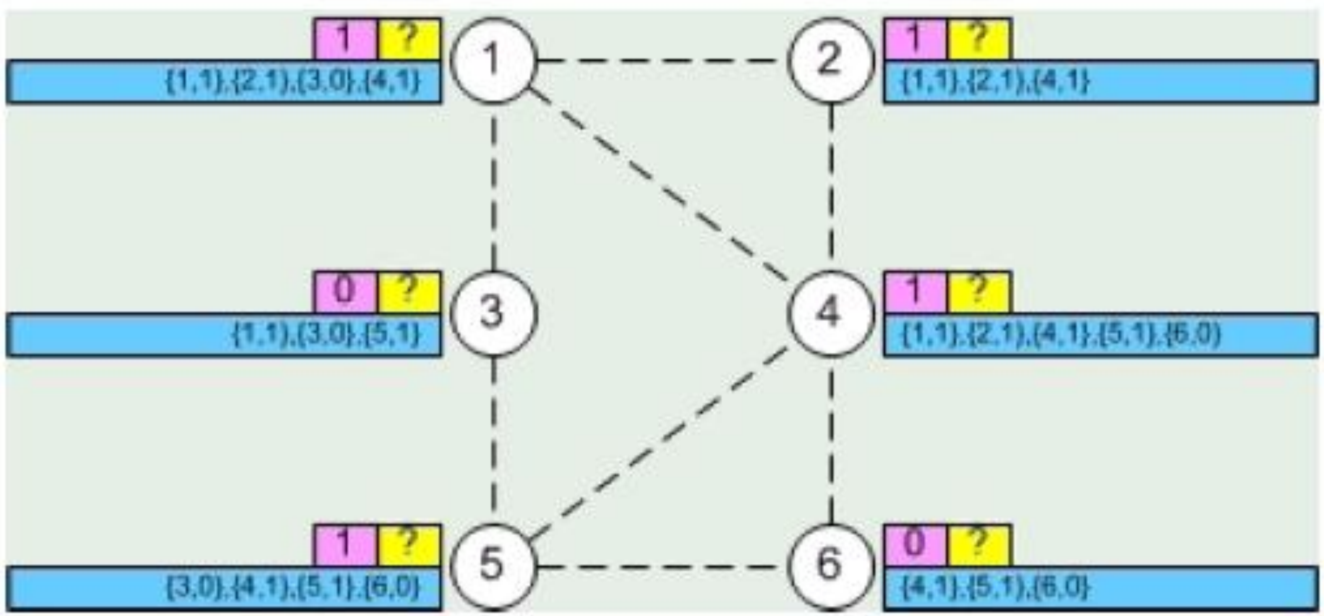


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

1st Round – processing

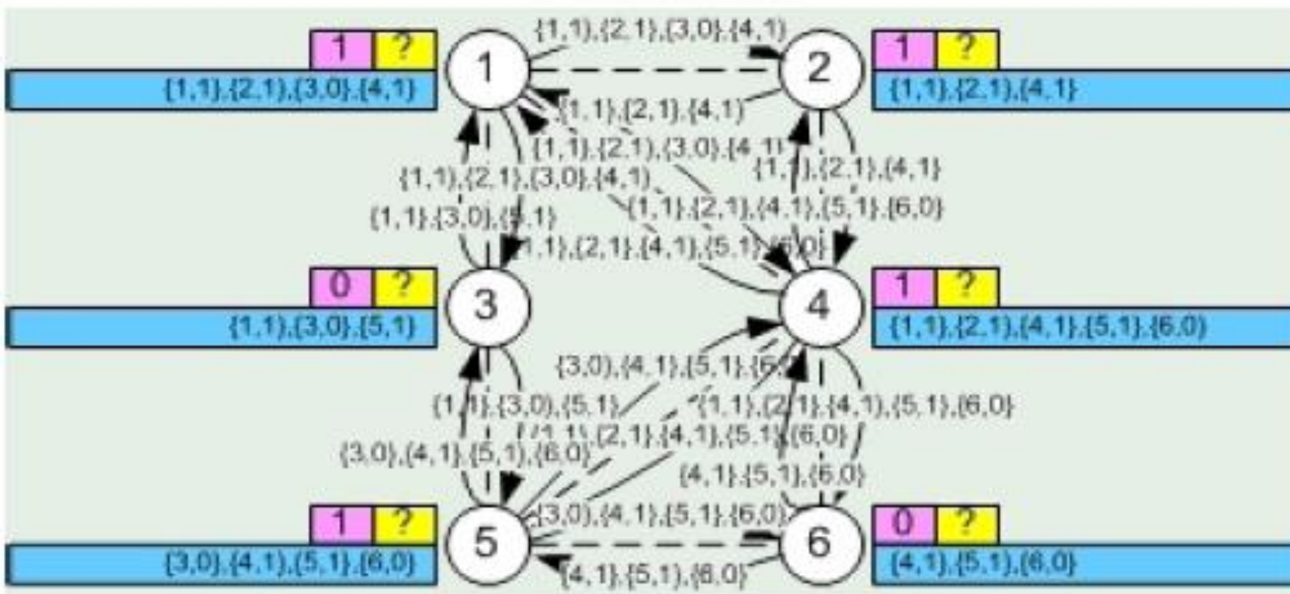


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

2nd Round – message transmission

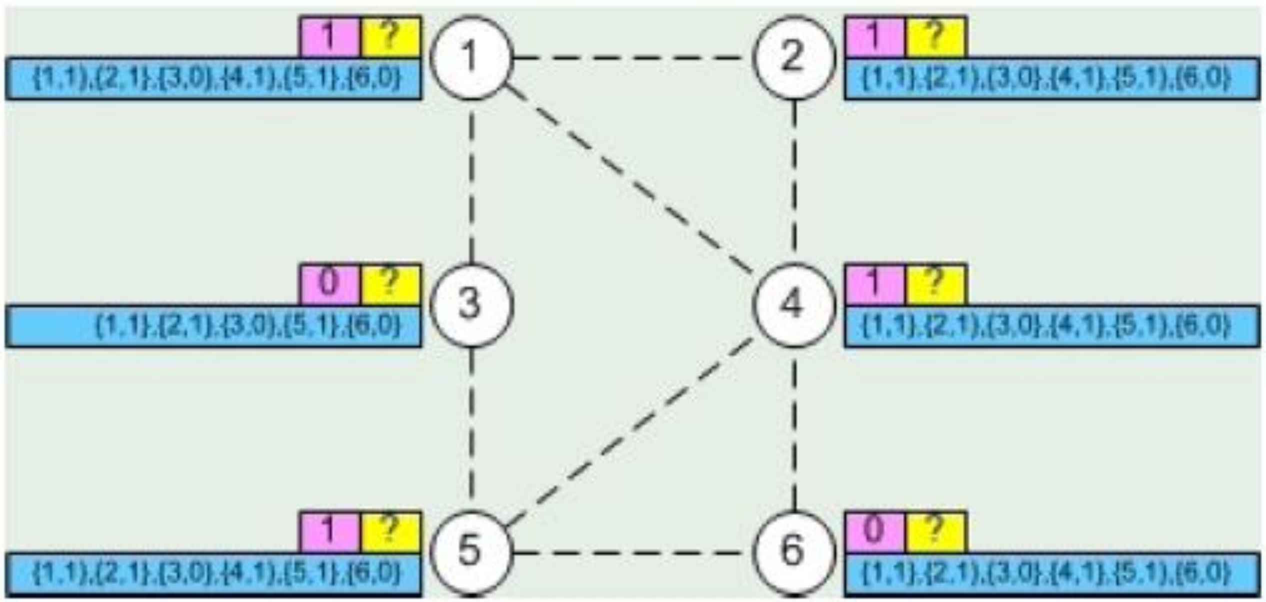


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

2nd Round – processing

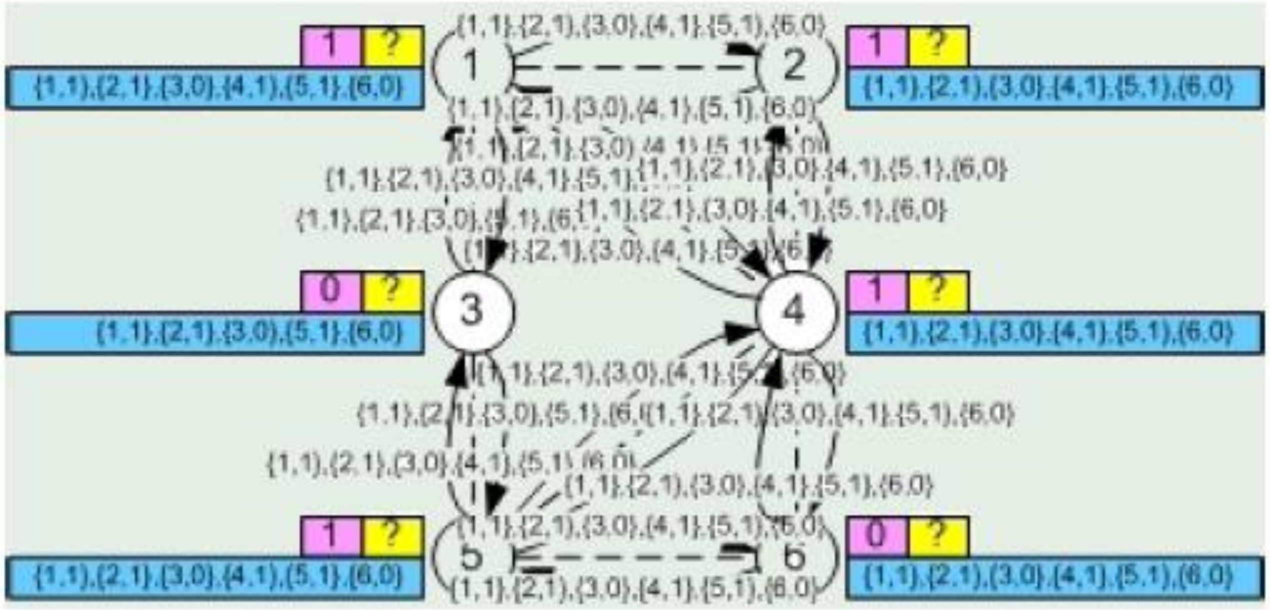


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

3rd Round – message transmission

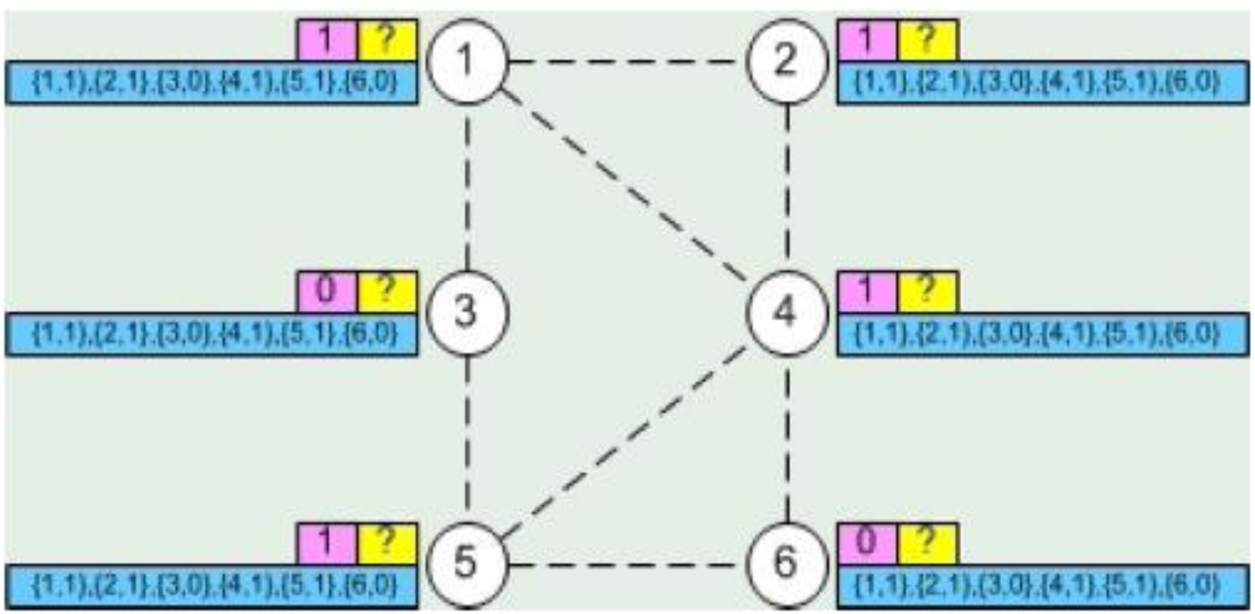


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

3rd Round – processing

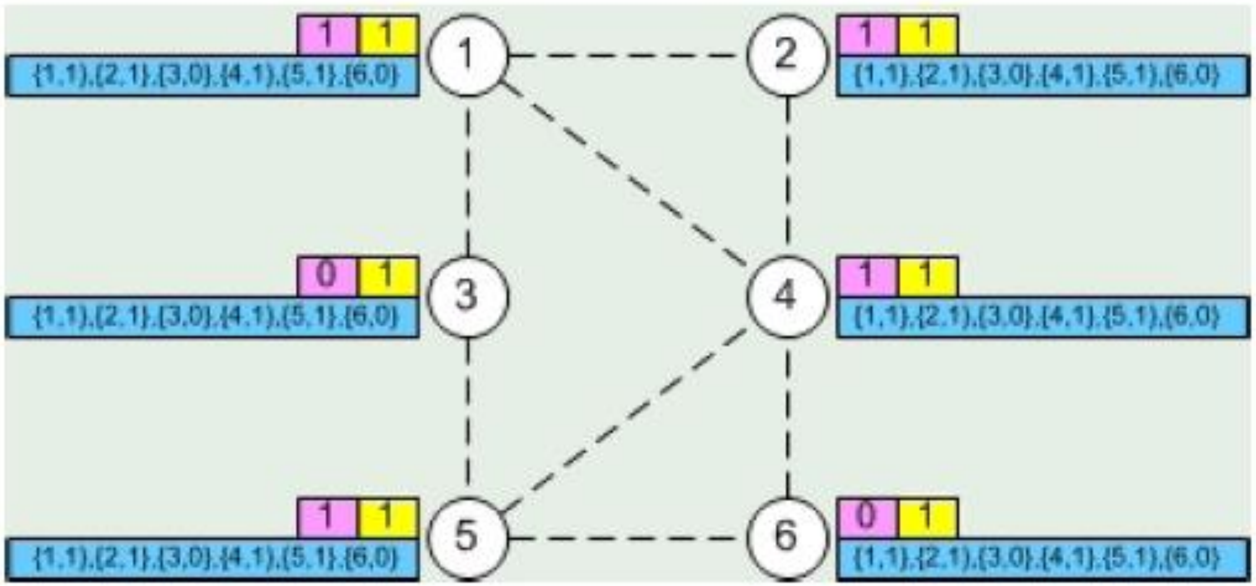


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

3rd Round– decision



Properties of SimpleConsensus Algorithm

Let a synchronous network G with n processes and m channels

- At the end of round δ each process $u \in [1, n]$ will maintain a list $I_u = \{(1, i_1), (2, i_2), \ldots, (n, i_n)\}$
- The lists maintained by all processes are identical, i.e., $\forall u \in [1, n] : I_u = I$
- The time complexity is $\mathcal{O}\left(diam(G)\right)$
- The message complexity is $\mathcal{O}\left(diam(G) \cdot m\right)$
- The message complexity in bits is $\mathcal{O}\left(diam(G) \cdot n \cdot m\right)$

Considerations

How will the execution evolve if failures occur during the transmission of messages ?

Given the presence of failures,

- can we guarantee the correctness of SimpleConsensus ?
- can we identify failure ?
- can we prevent/deal with failure ?

Fundamental limitation

Theorem

Let G be the graph constituting of nodes 1 and 2 connected by a single edge. Then, there is no algorithm that solves the coordinated attack problem on G given an unbounded number of link failures.

- Impossible to solve basic consensus problems when dealing with totally unreliable network.
- To overcome, it is necessary to strengthen the model
  - Assume an upper bound on the number of link failures.
  - Assume that link failures occur with a probability p.
- or relax the problem requirements
  - Allow the possibility of violating the agreement condition.
  - Allow the possibility of violating the validity condition.
- Allow processes to use randomization.

Stopping Failures

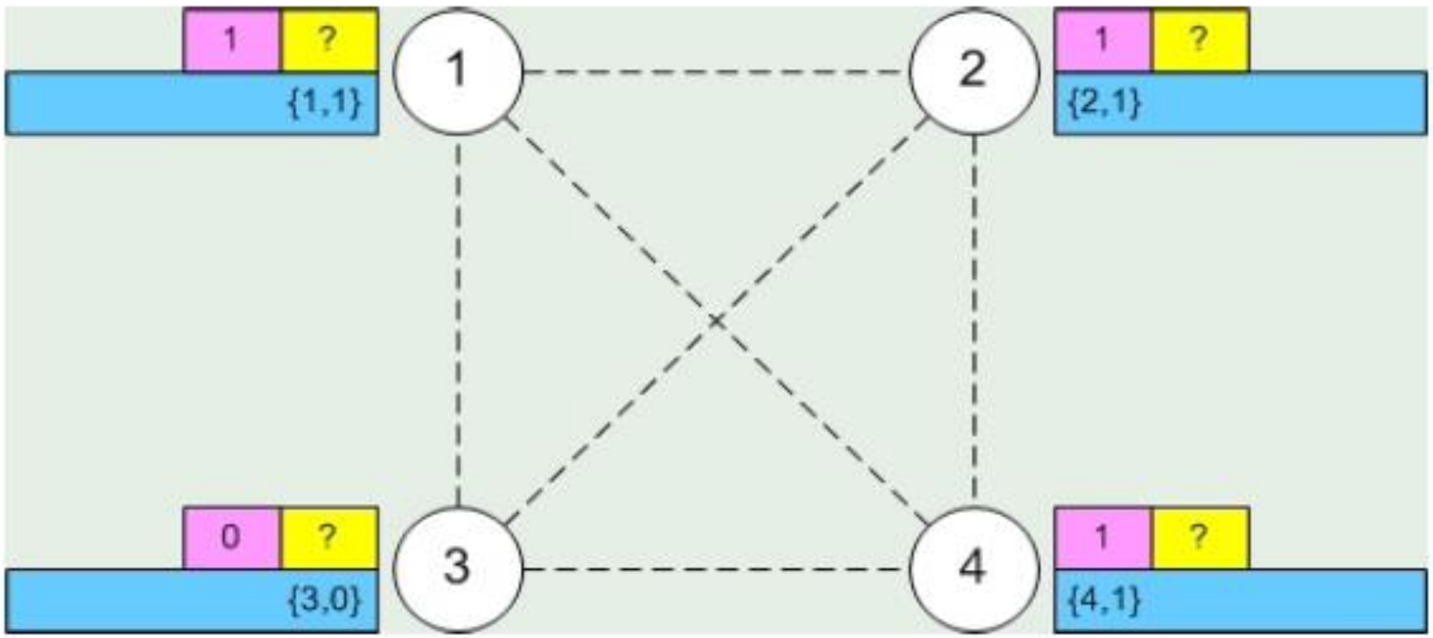
Processes may simply stop arbitrarily without warning, at any point during a round of execution of a distributed algorithm. The process will halt immediately and terminate without further interaction with the other processes of the system.

- Stopping failures model unpredictable processor crashes.
- We assume an upper bound σ on the number of stopping failures
  - such an upper bound holds for the complete execution of the distributed system.
  - is equivalent to other measures, e.g., rate of stopping failure per round.

Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

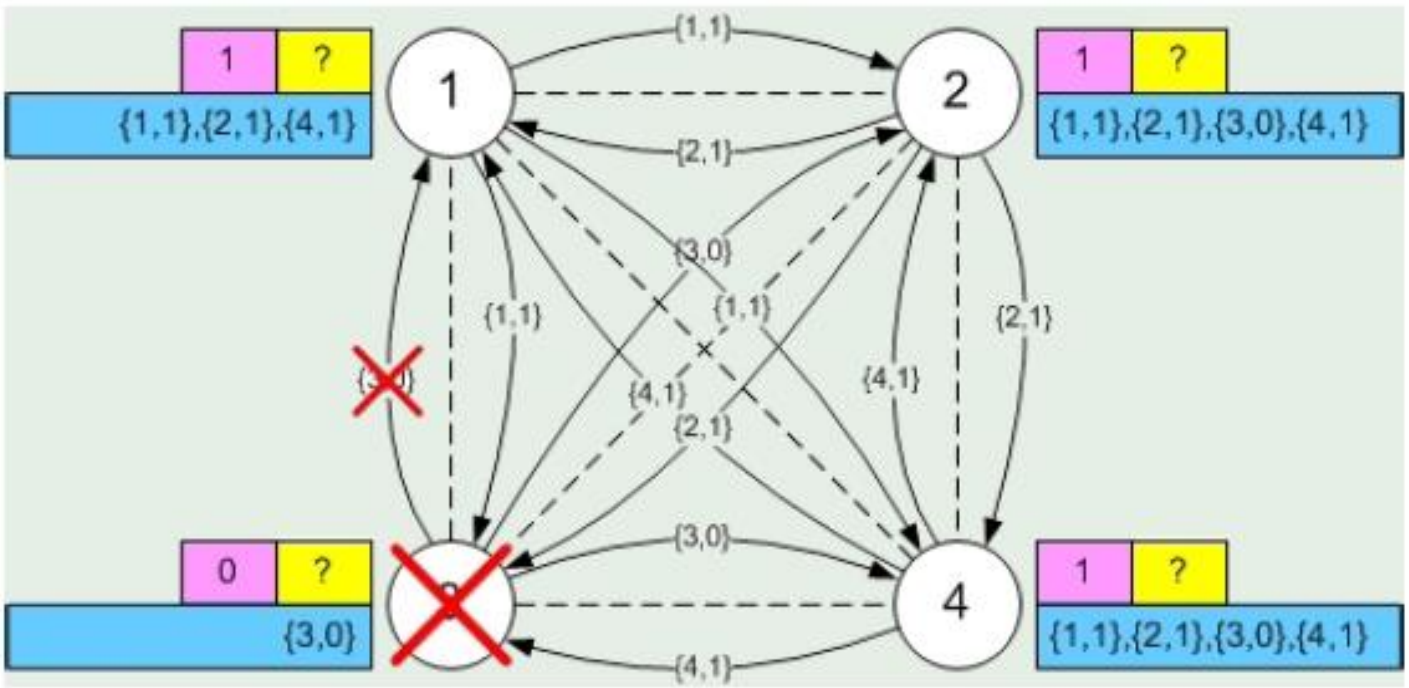
Complete Graph



Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

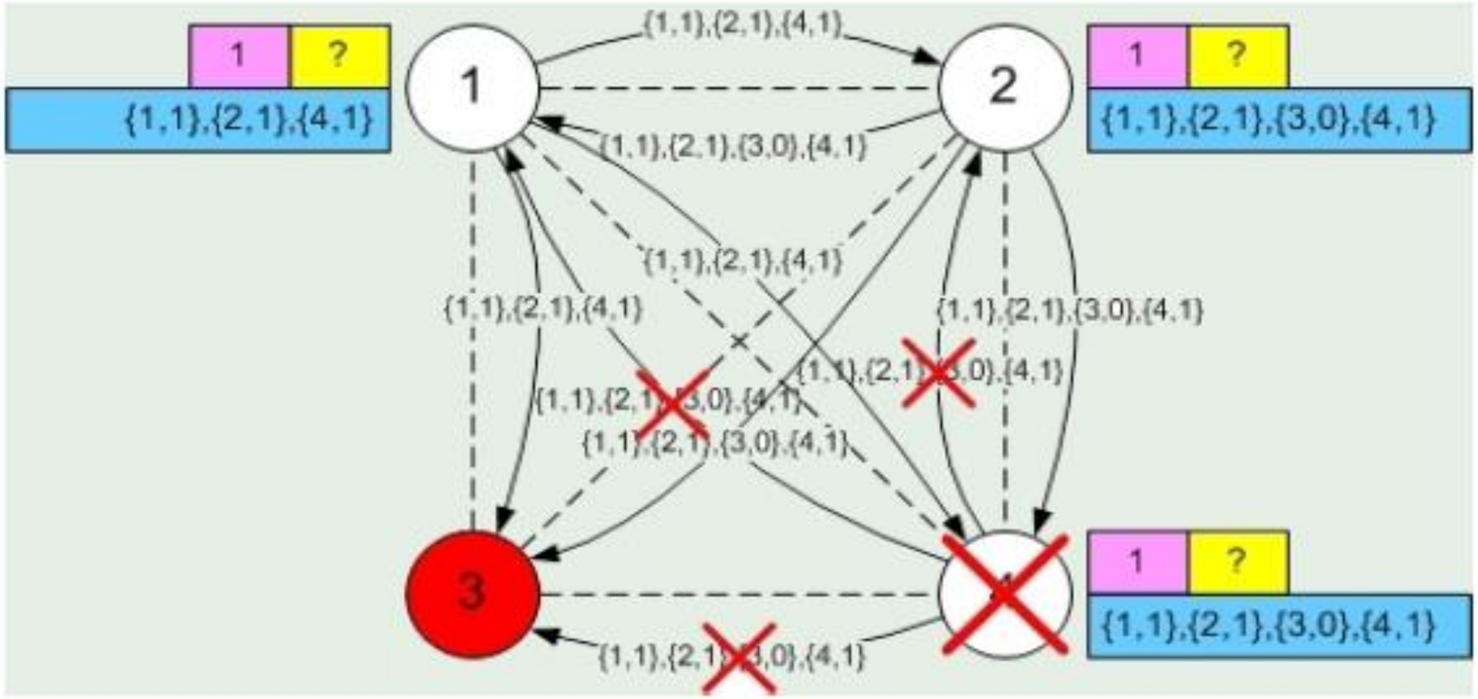
1st Round – process 3 fails



Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

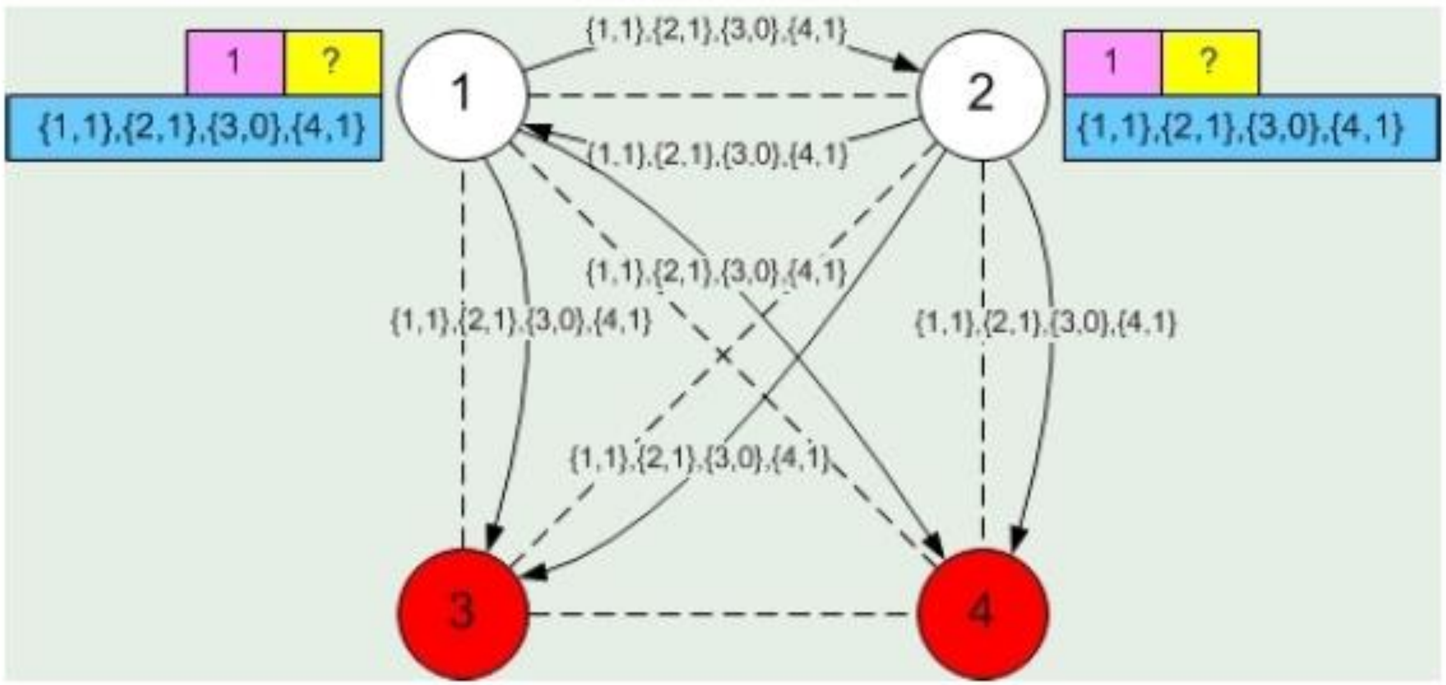
2nd Round – process 4 fails



Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

3rd Round – no failures



Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

3rd Round – agreement