Consensus and Related Problems

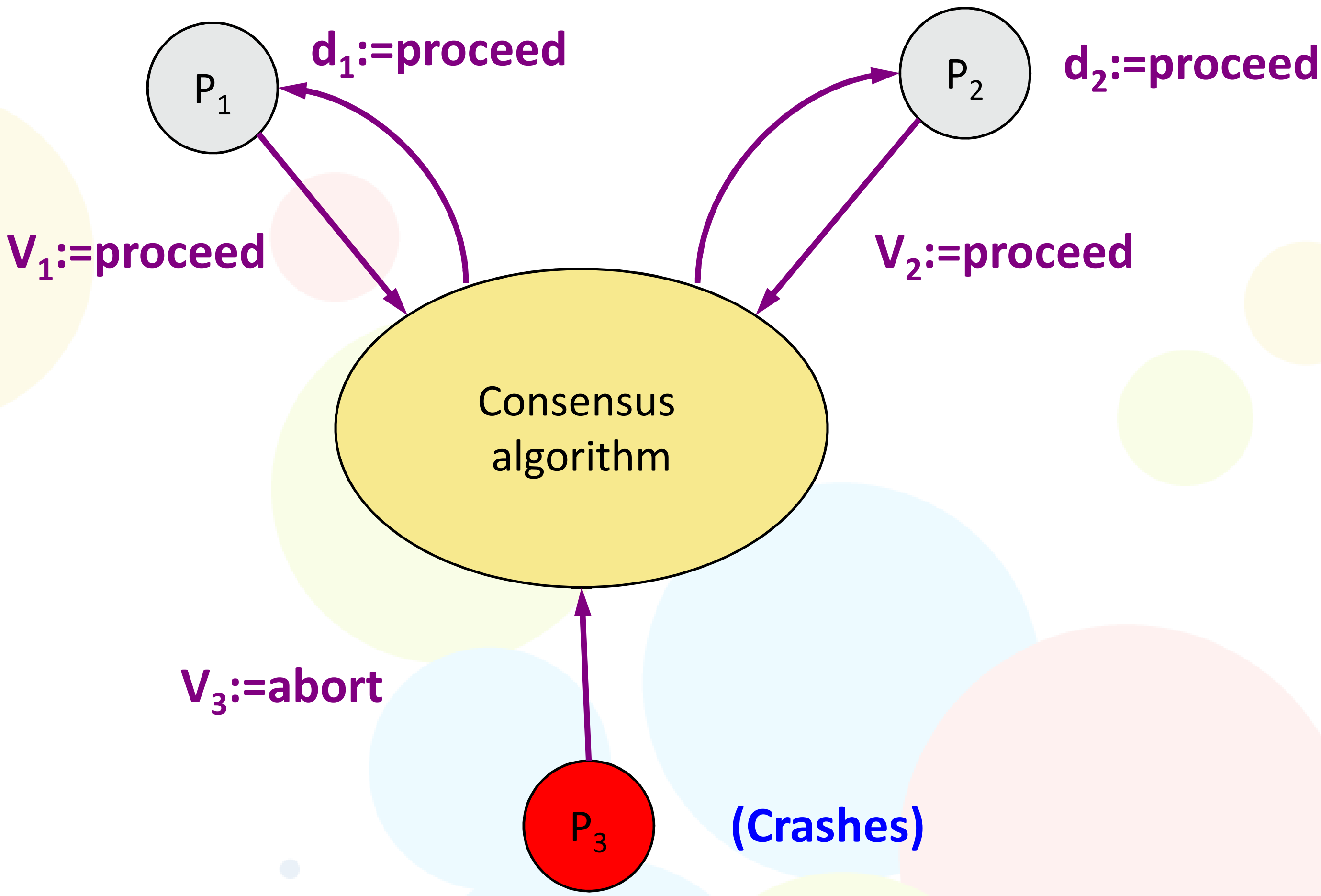
George Coulouris, Jean Dollimore and Tim Kindberg, "Distributed Systems Concepts and Design", Fifth Edition, Pearson Education, 2012

Consensus (1)

- **Objective:** processes must agree on a value after one or more of the processes has proposed what that value should be
- **Hypotheses:** reliable communication, but processes may fail
- **Consensus problem:**
  - Every process P_i begins in the *undecided* state
  - Proposes a value $V_i \subset D$ (i=1, …, N)
  - Processes communicate with one another, exchanging values
  - Each process then sets the value of a decision variable d_i

Enters the state *decided*, in which it may no longer change d_i (i=1, …, N)

Consensus (2)



P_1
d_1:=proceed
P_2
d_2:=proceed
V_1:=proceed
V_2:=proceed
Consensus algorithm
V_3:=abort
P_3
(Crashes)

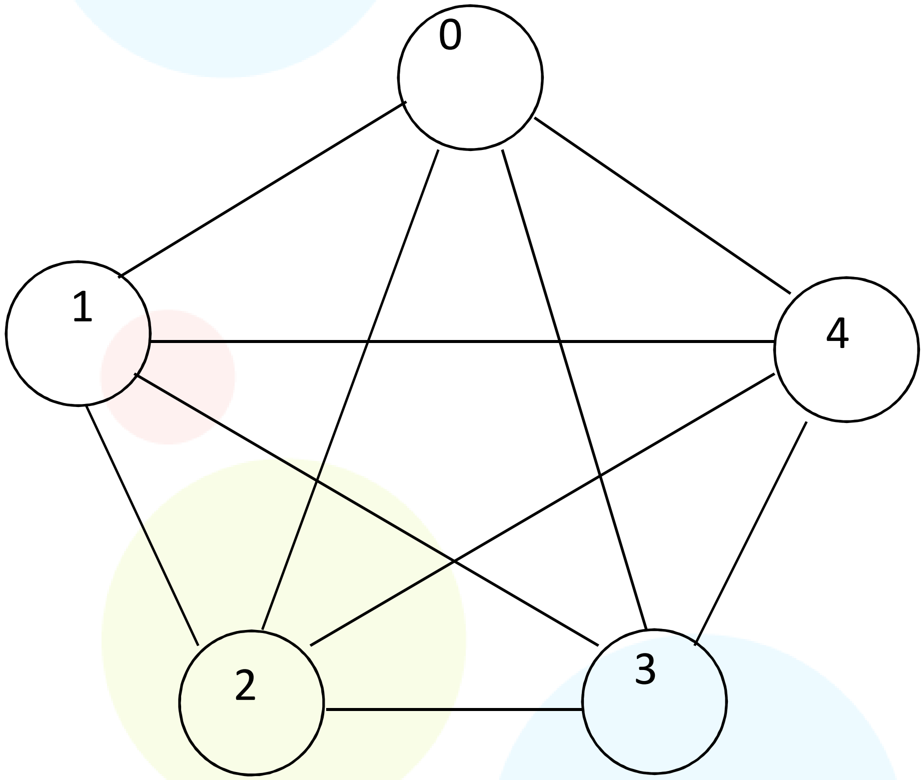
A simple algorithm for fault-free consensus

Each processor:

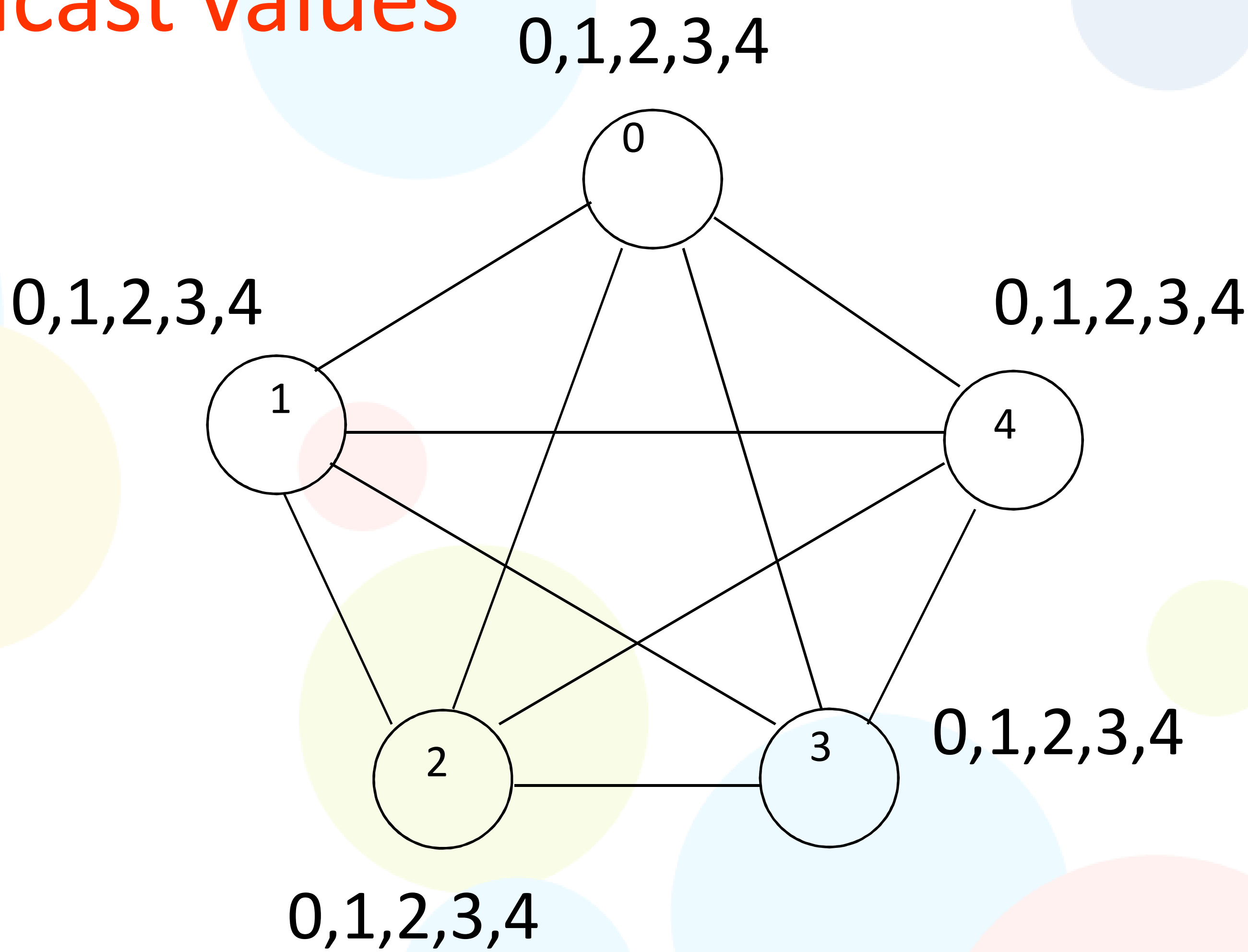
1. Broadcast its input to all processors
2. Decide on the minimum

(only one round is needed, since the graph is complete)

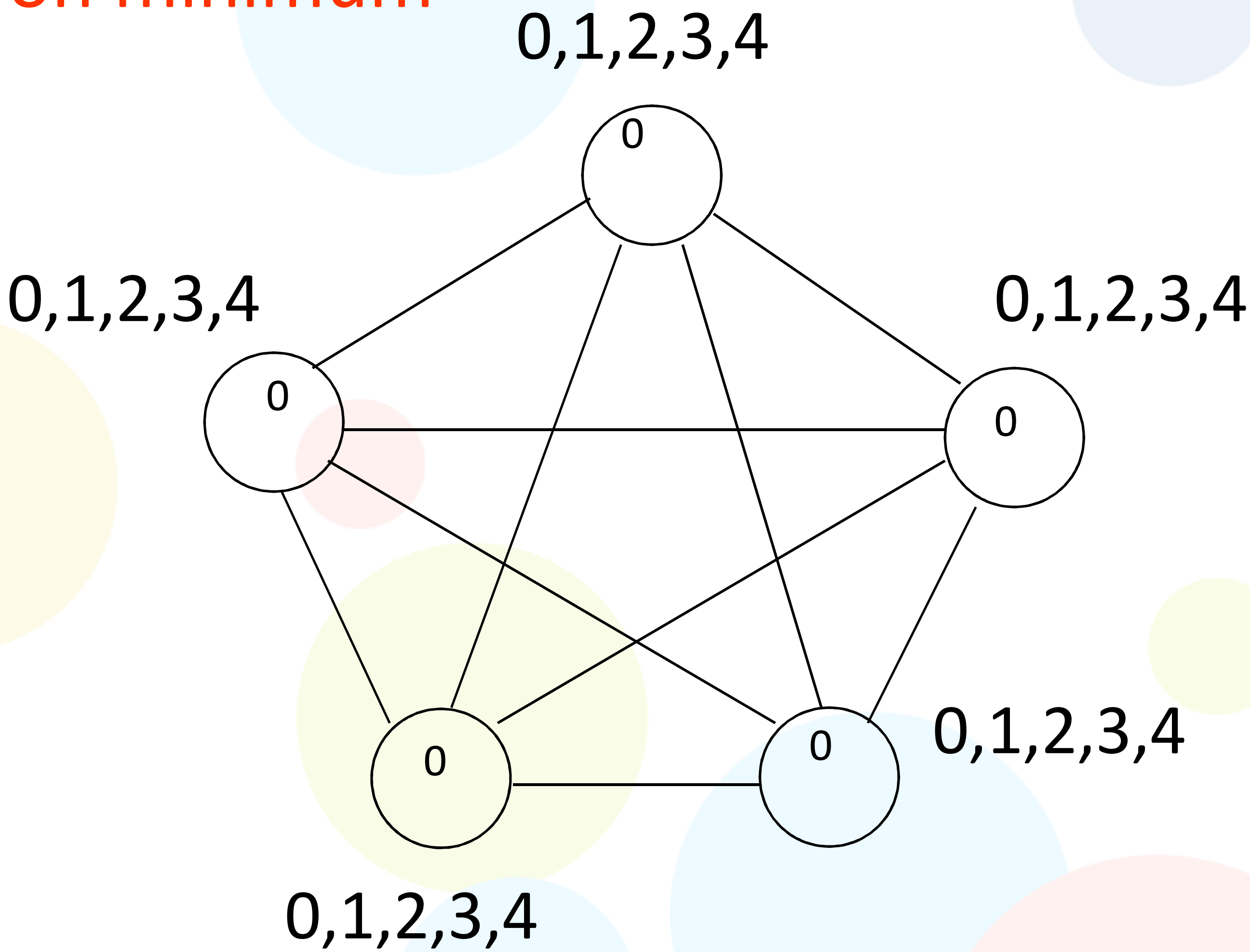
Start



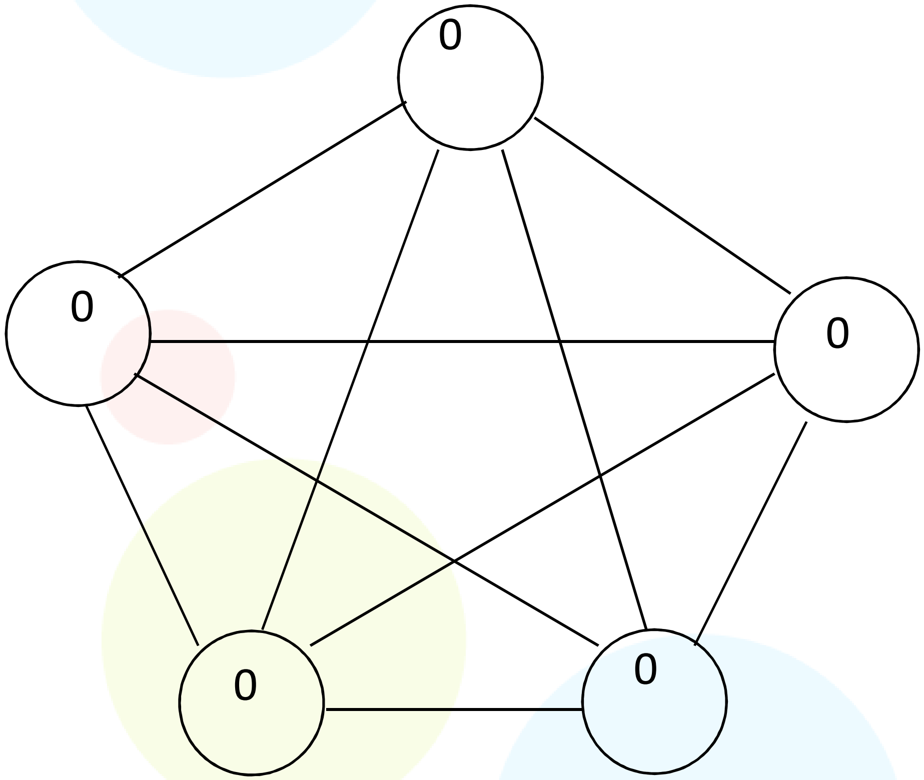
Broadcast values



Decide on minimum

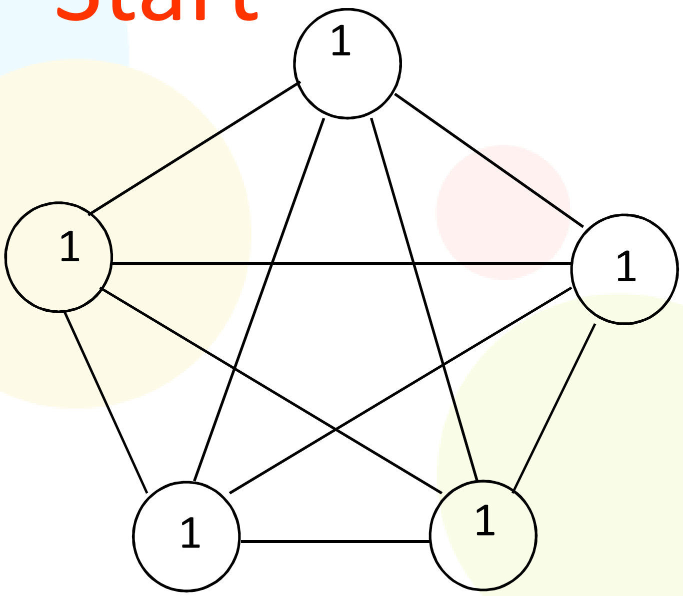


Finish

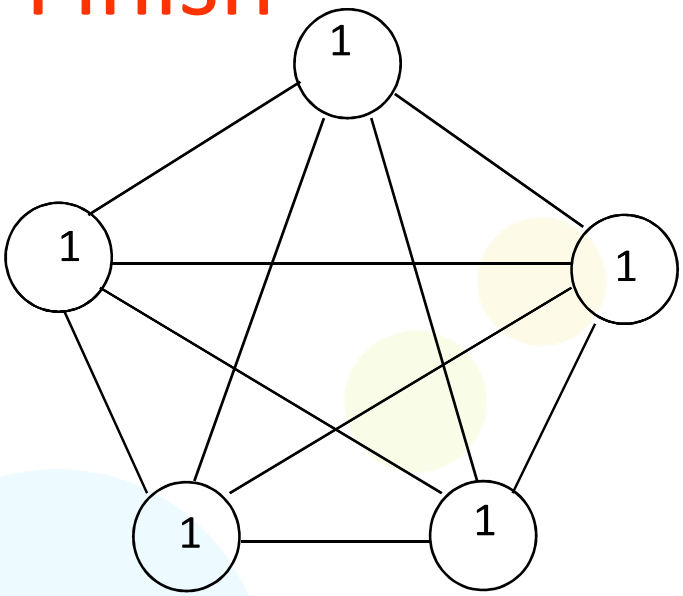


This algorithm satisfies the validity condition

Start



Finish



If everybody starts with the same initial value, everybody decides on that value (minimum)

Consensus with Crash Failures

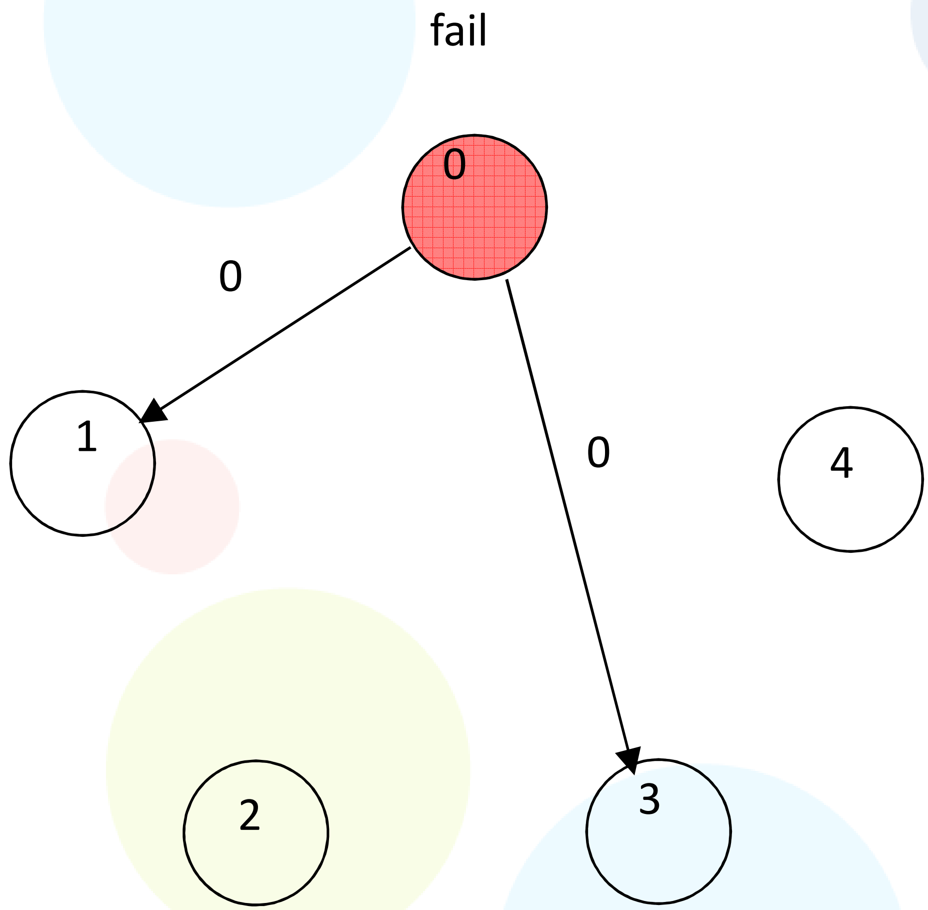
The simple algorithm doesn't work

Each processor:

1. Broadcast value to all processors

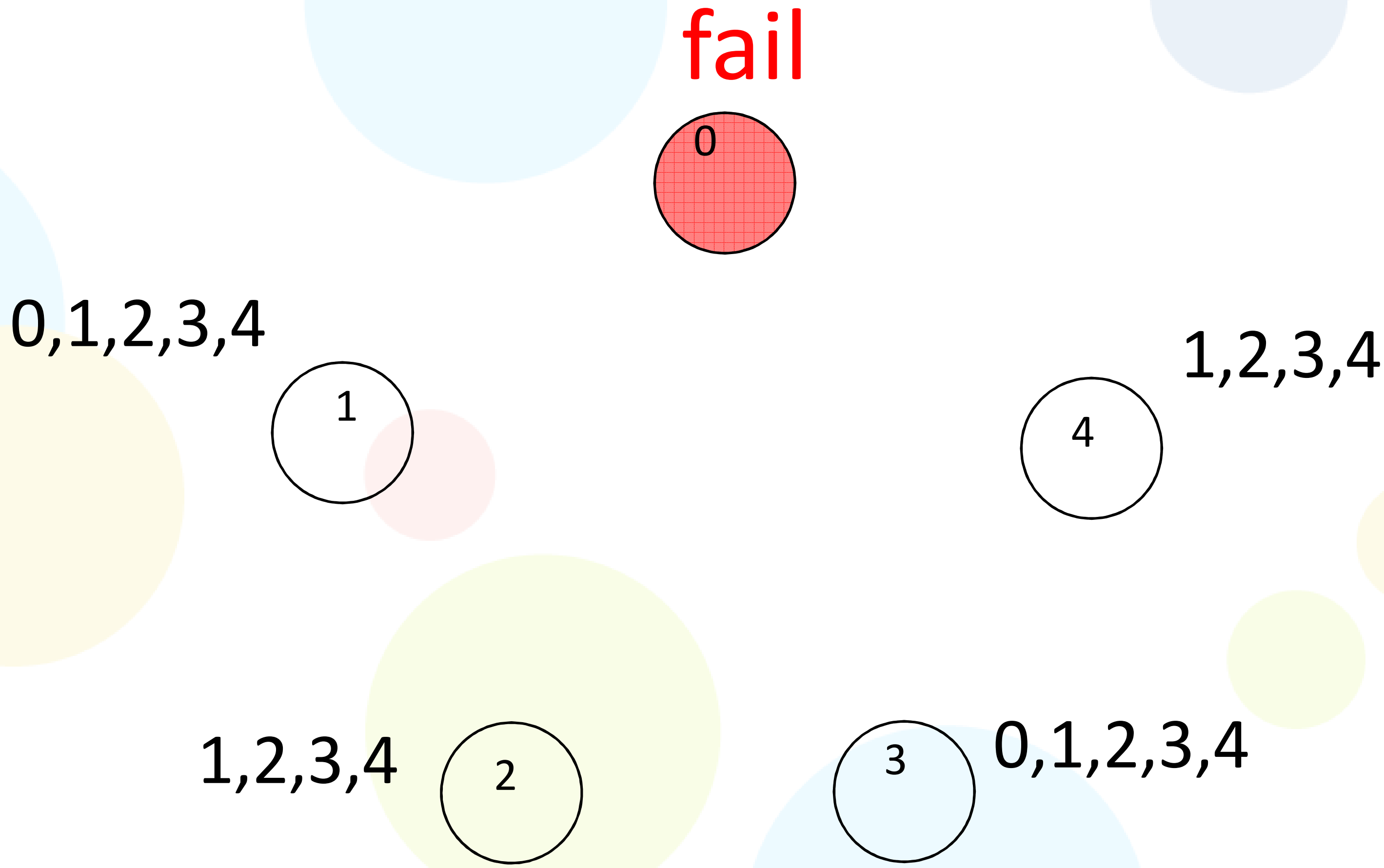
2. Decide on the minimum

Start

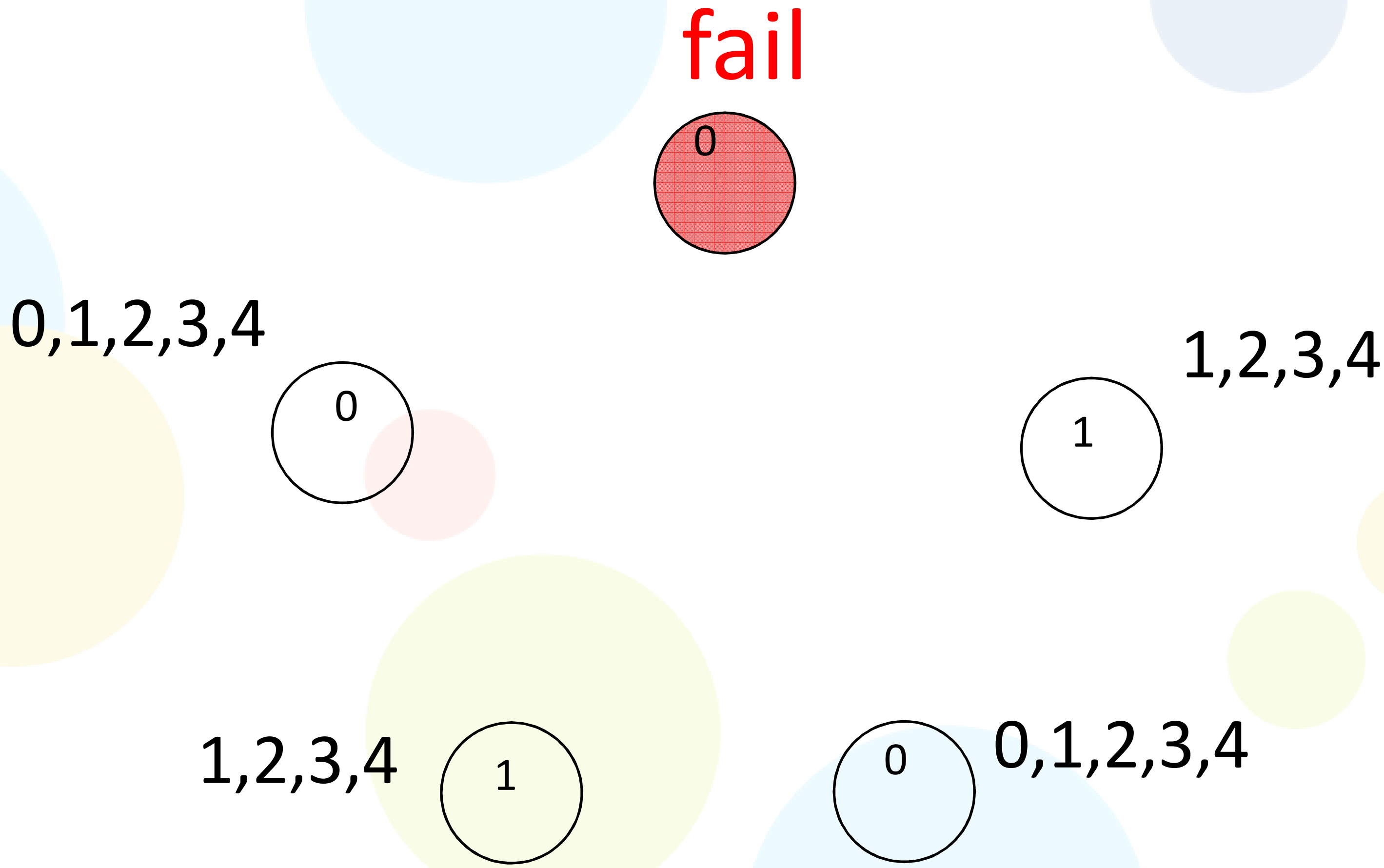


The failed processor doesn't broadcast its value to all processors

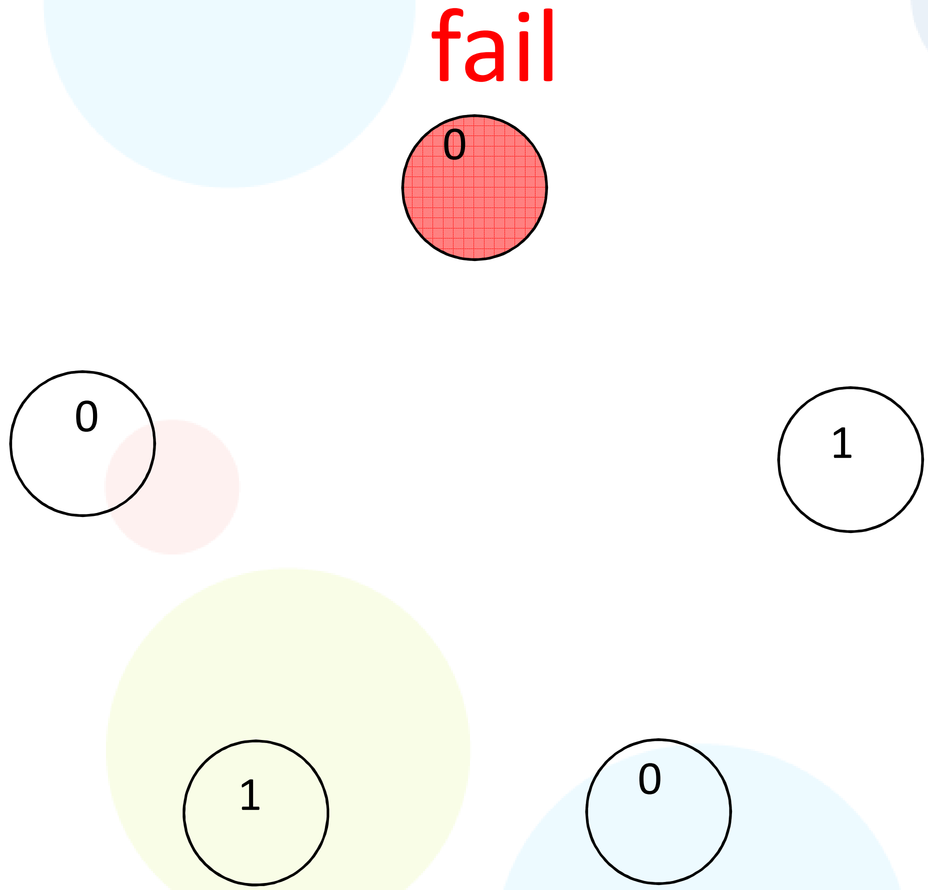
Broadcasted values



Decide on minimum



Finish



No Consensus!!!

If an algorithm solves consensus for
f failed (crashing) processors we say it is:

an f-resilient consensus algorithm

An f-resilient algorithm

Round 1:

Broadcast my value

Round 2 to round f+1:

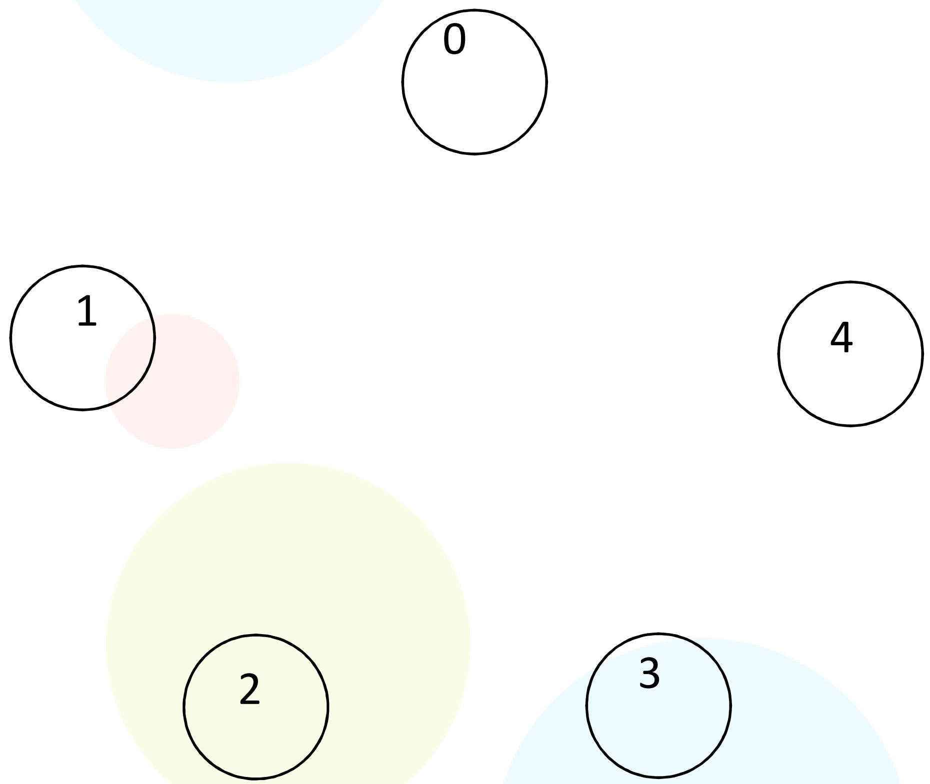
Broadcast any new received values

End of round f+1:

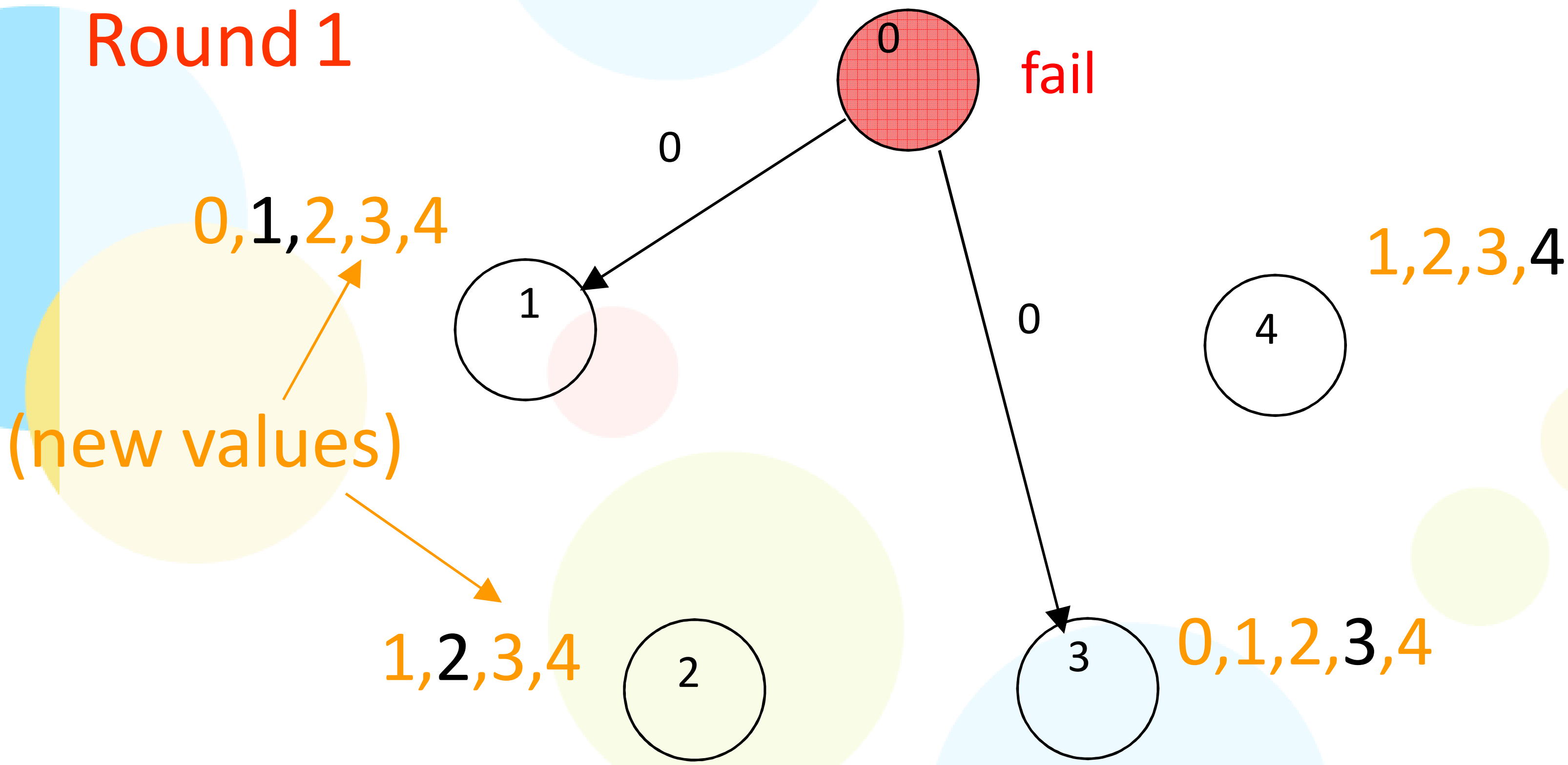
Decide on the minimum value received

Example: f=1 failures, f+1 = 2 rounds needed

Start



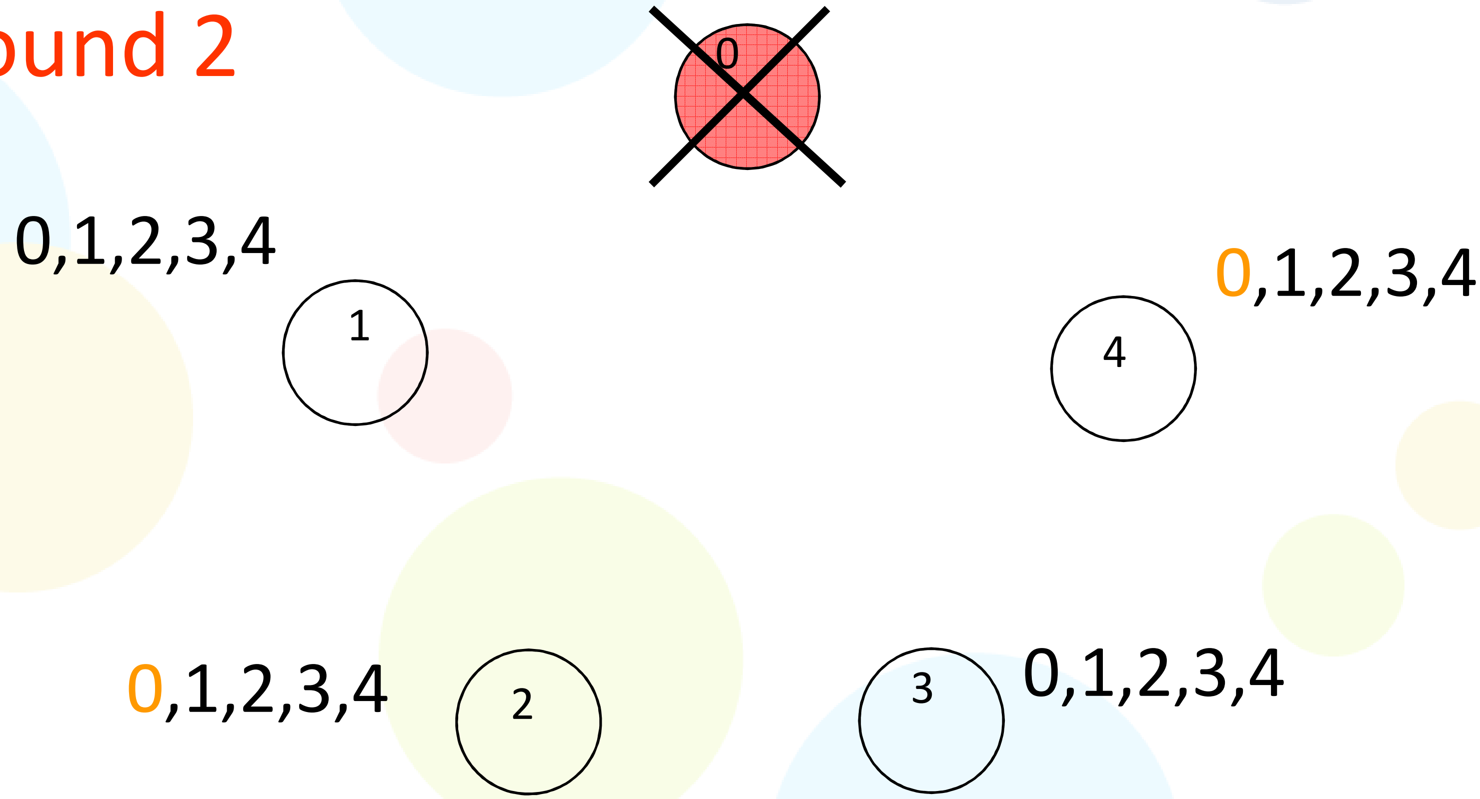
Example: f=1 failures, f+1 = 2 rounds needed



Broadcast all values to everybody

Example: f=1 failures, f+1 = 2 rounds needed

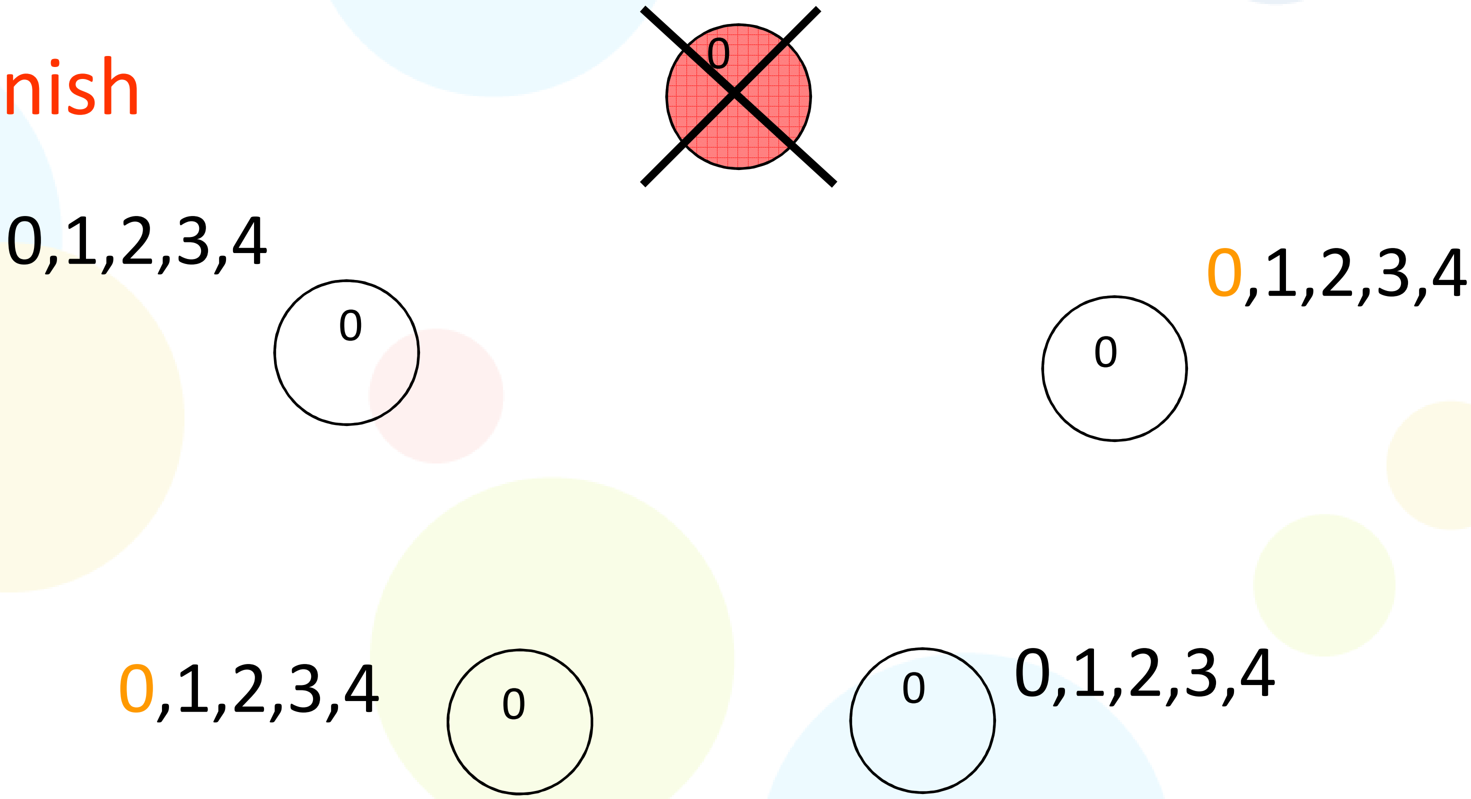
Round 2



Broadcast all new values to everybody

Example: f=1 failures, f+1 = 2 rounds needed

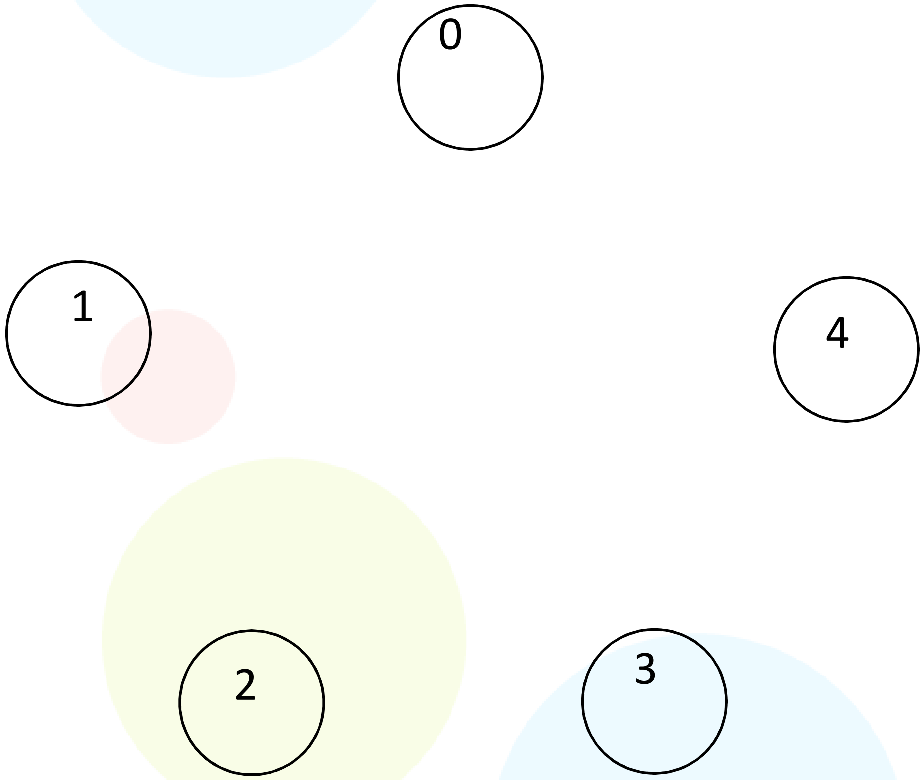
Finish



Decide on minimum value

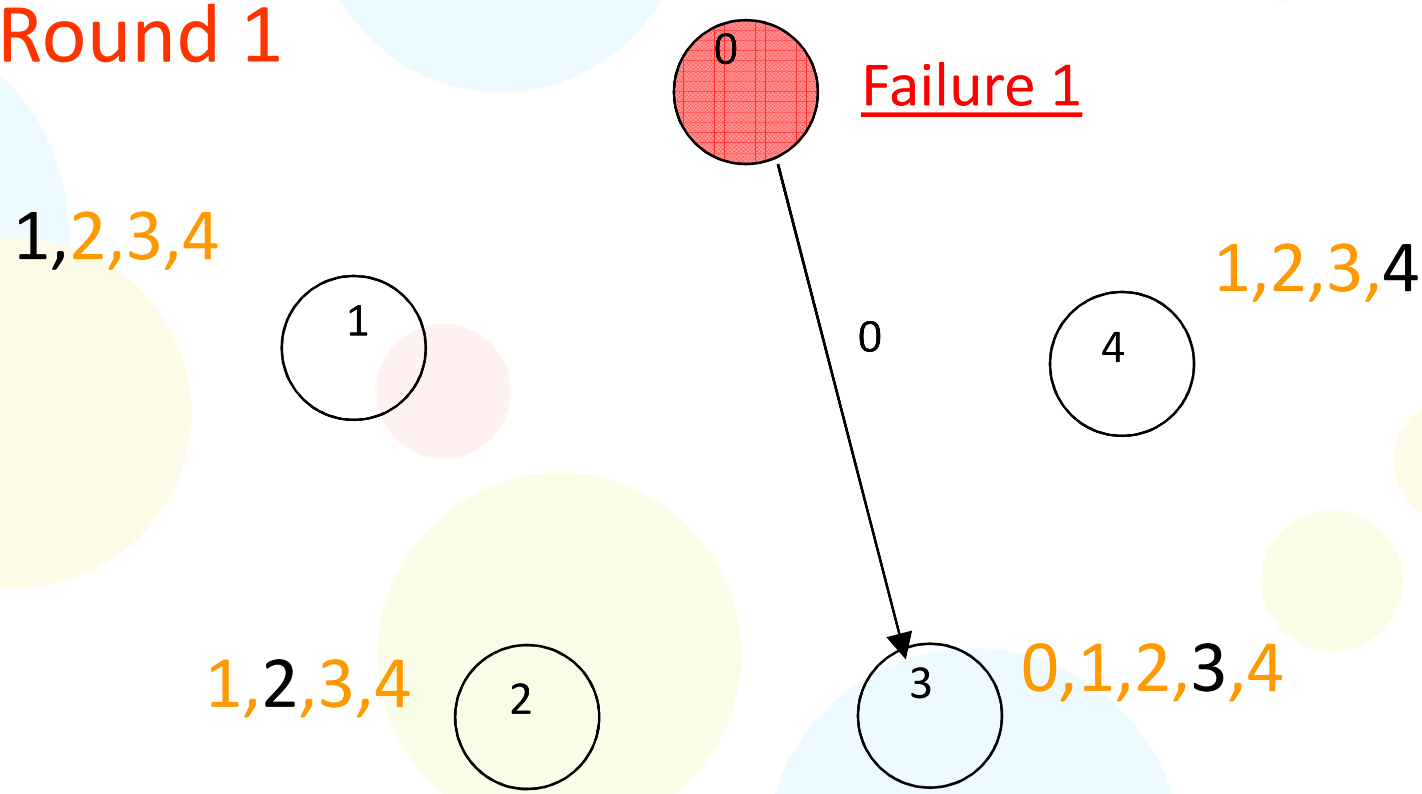
Example: f=2 failures, f+1 = 3 rounds needed

Start



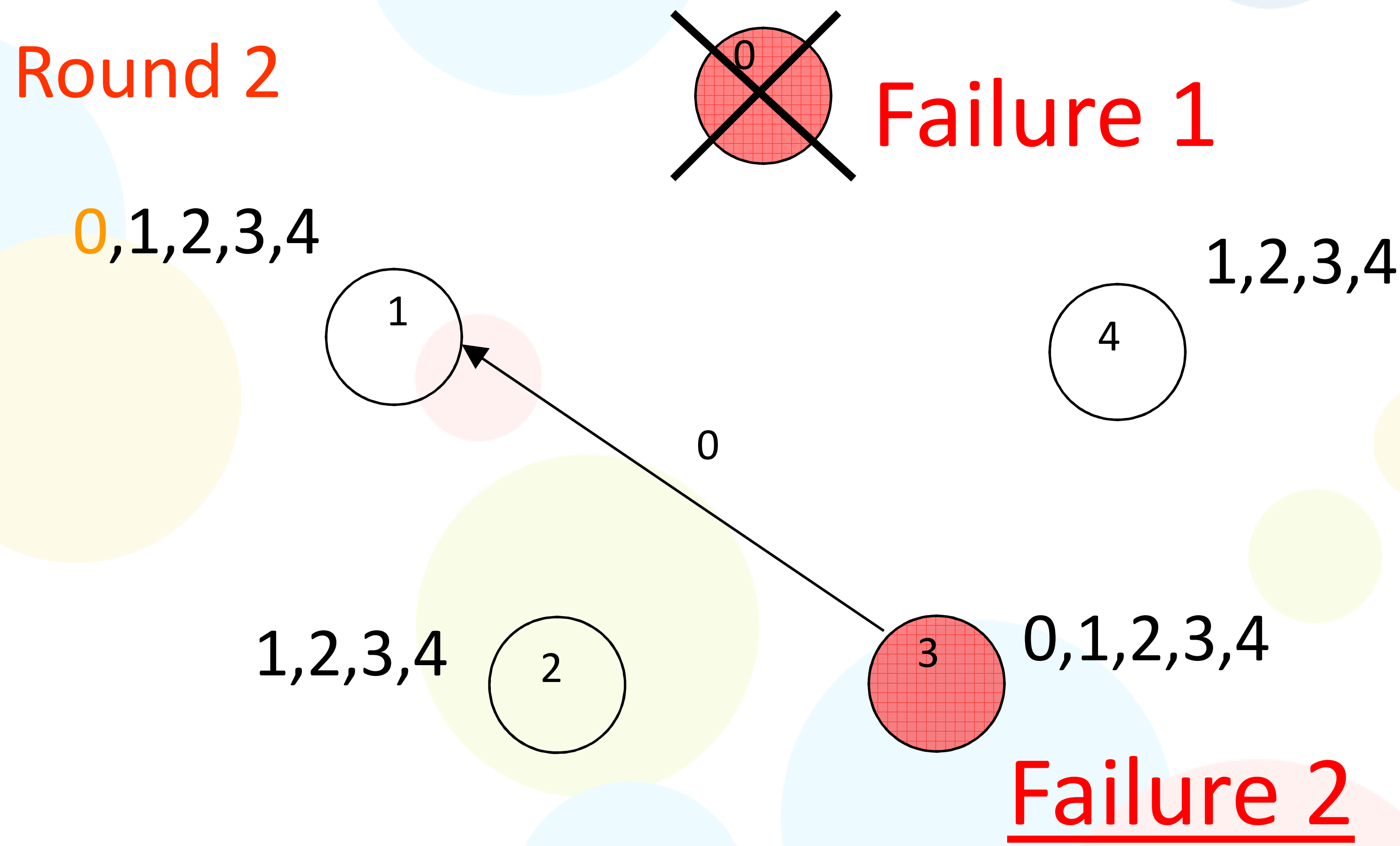
Example: f=2 failures, f+1 = 3 rounds needed

Round 1



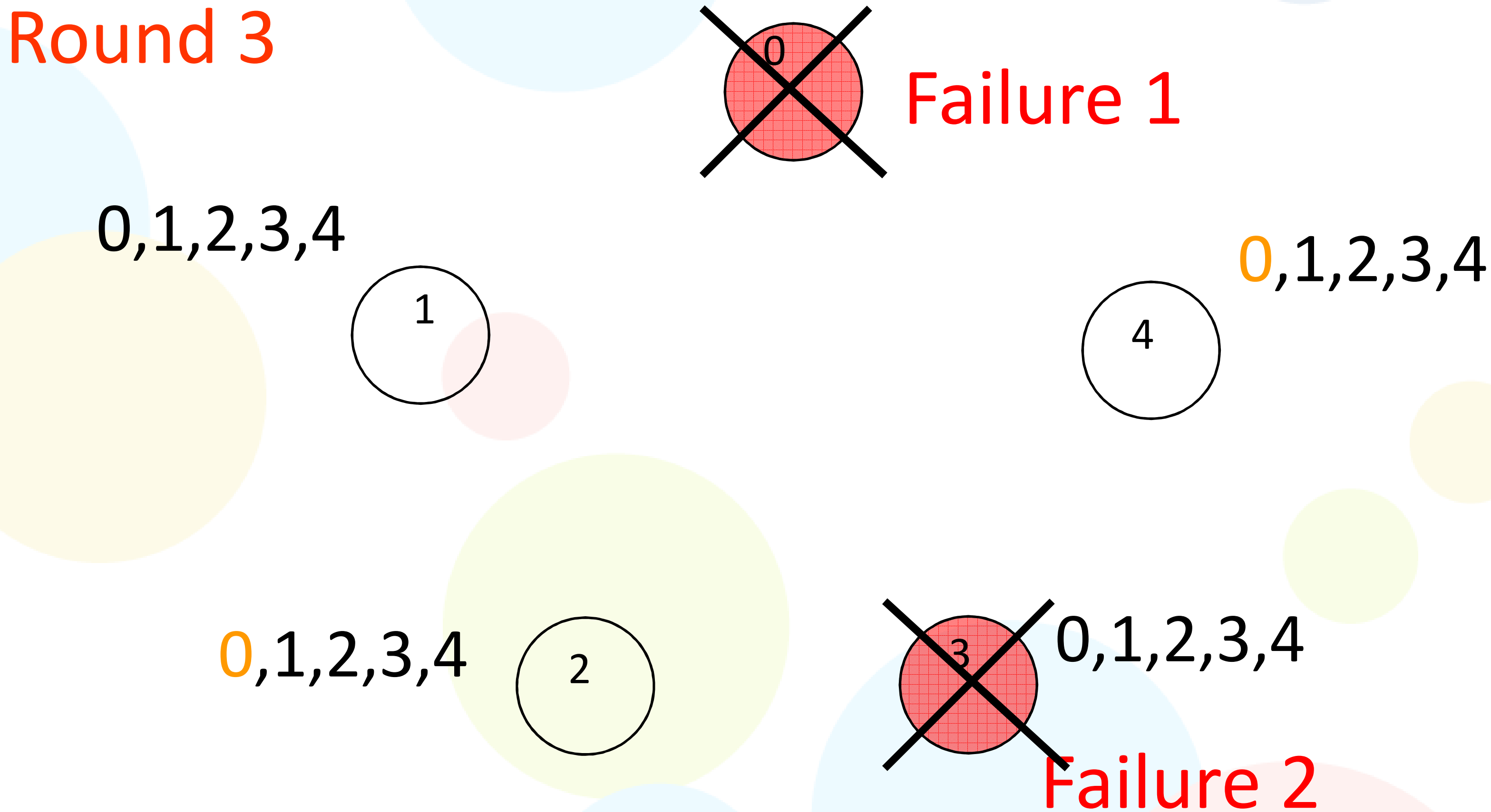
Broadcast all values to everybody

Example: f=2 failures, f+1 = 3 rounds needed



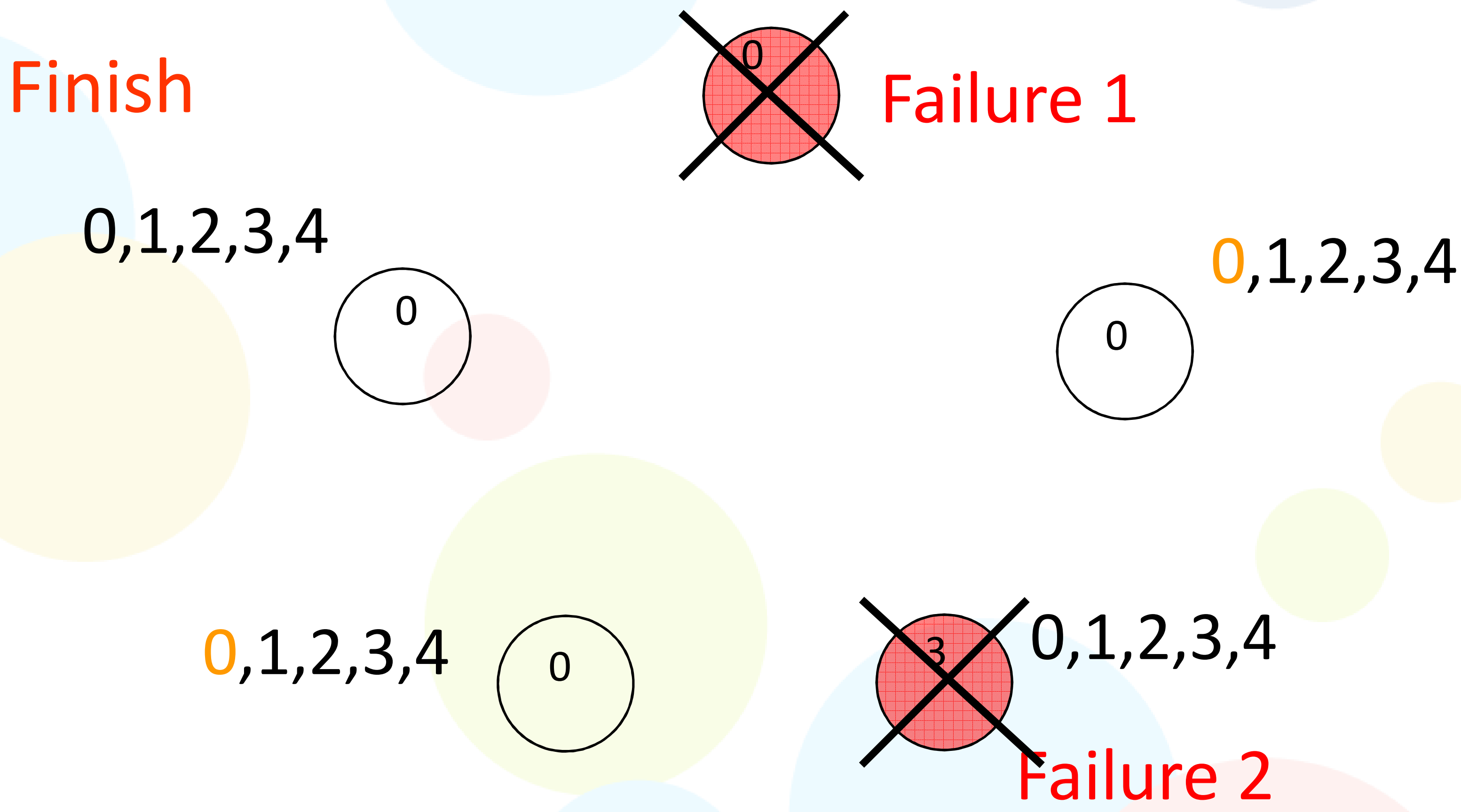
Broadcast new values to everybody

Example: f=2 failures, f+1 = 3 rounds needed



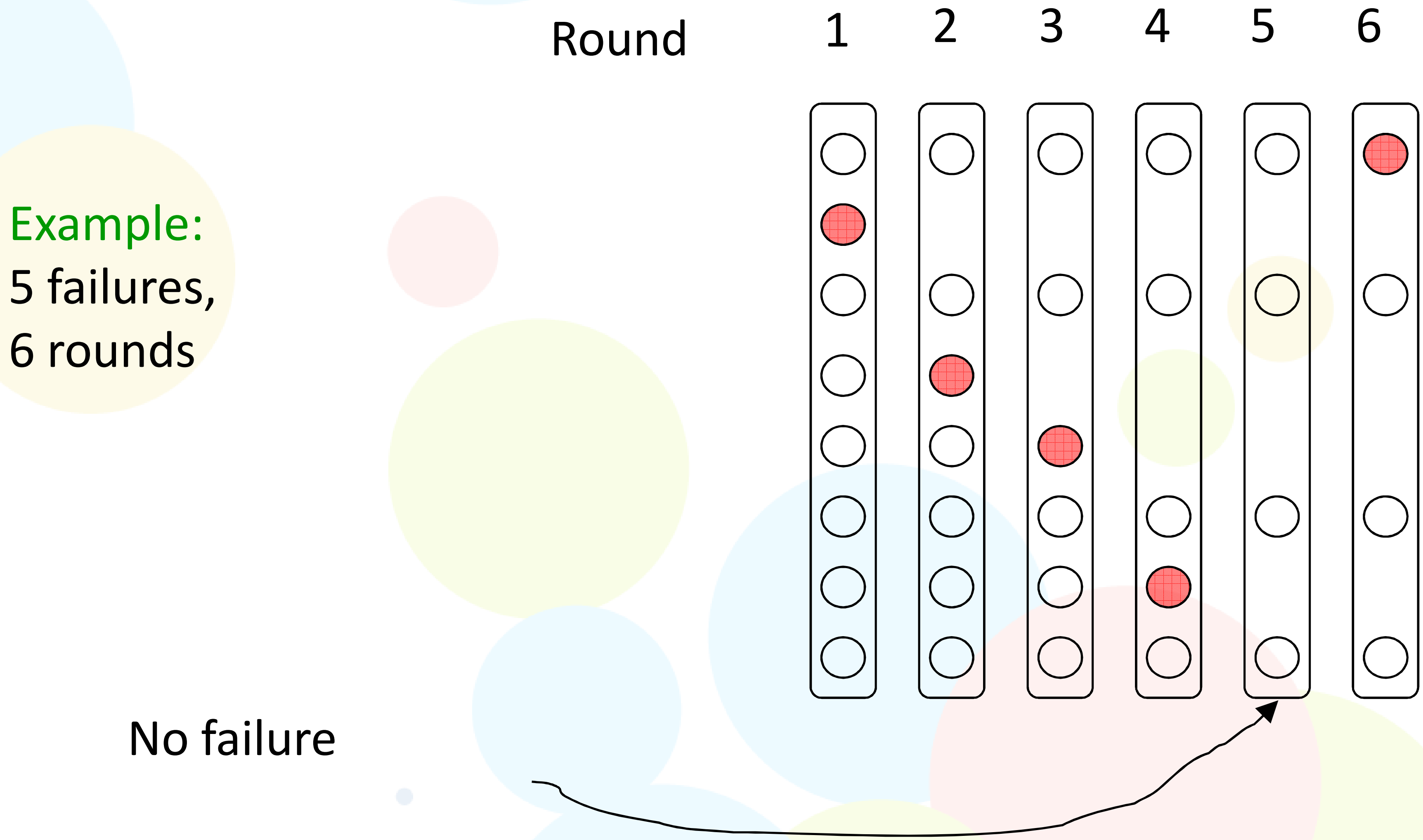
Broadcast new values to everybody

Example: f=2 failures, f+1 = 3 rounds needed



Decide on the minimum value

If there are f failures and f+1 rounds then there is at least a round with no failed processors:



In the algorithm, at the end of the round with no failure:

- Every (non faulty) process knows about all the values of all other participating processes

- This knowledge doesn’t change until the end of the algorithm

Therefore, at the end of the round with no failure:

everybody would decide the same value

However, we don’t know the exact position of this round, so we have to let the algorithm execute for f+1 rounds

Validity of algorithm

when all processes start with the same input value then the consensus is that value

This holds, since the value decided from each process is some input value

A Lower Bound

Theorem: Any f-resilient consensus algorithm requires at least f+1 rounds

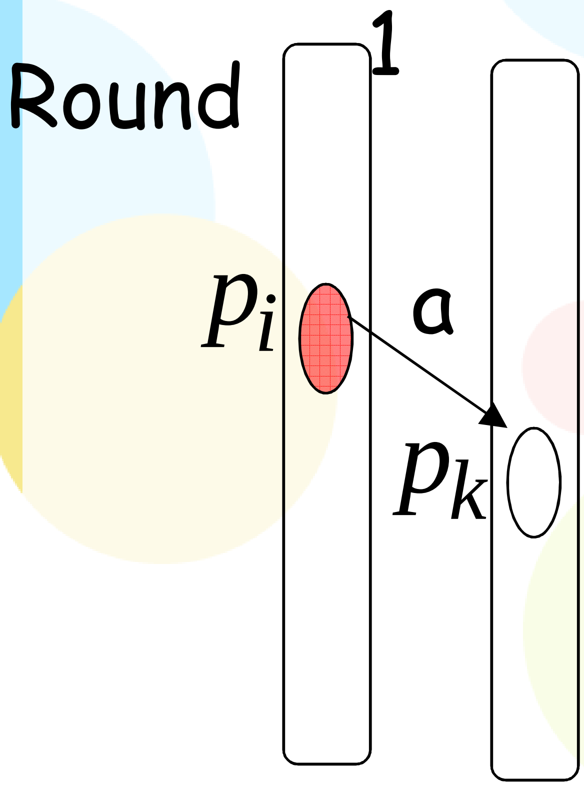
Proof sketch:

Assume for contradiction that f or less rounds are enough

Worst case scenario:

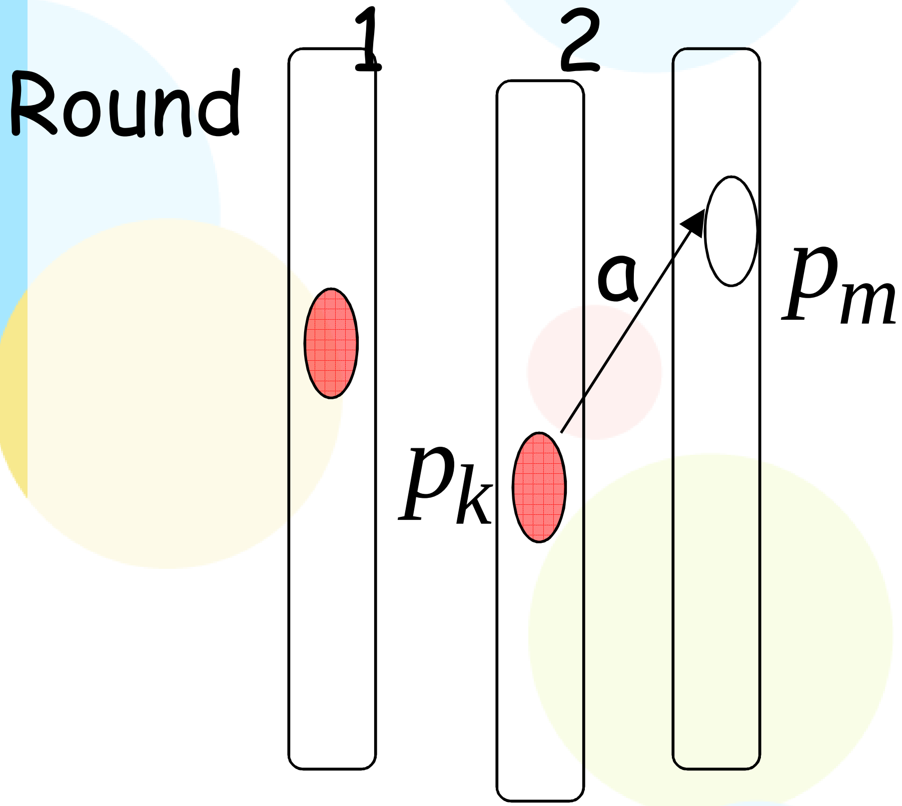
There is a process that fails in each round

Worst case scenario



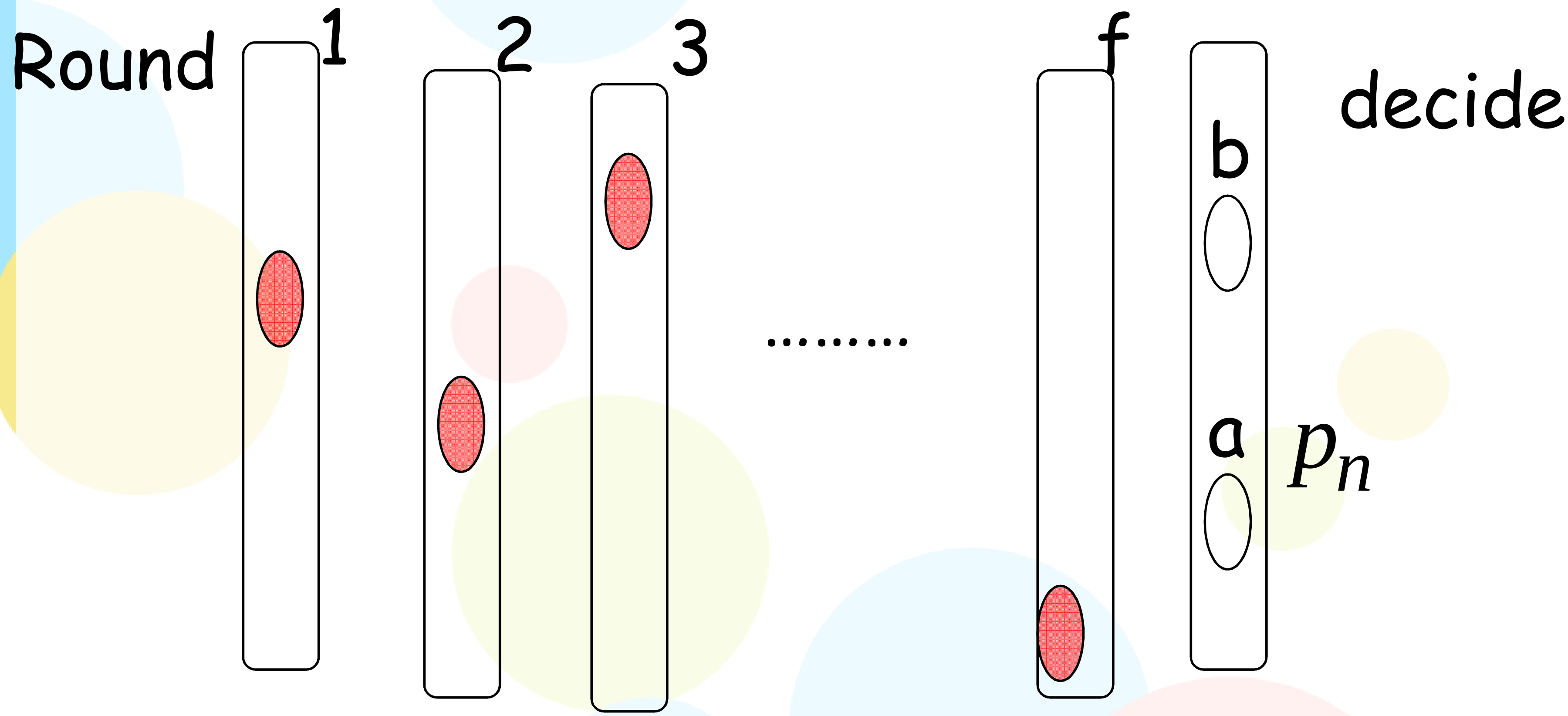
before process p_i fails, it sends its value **a** to only one process p_k

Worst case scenario



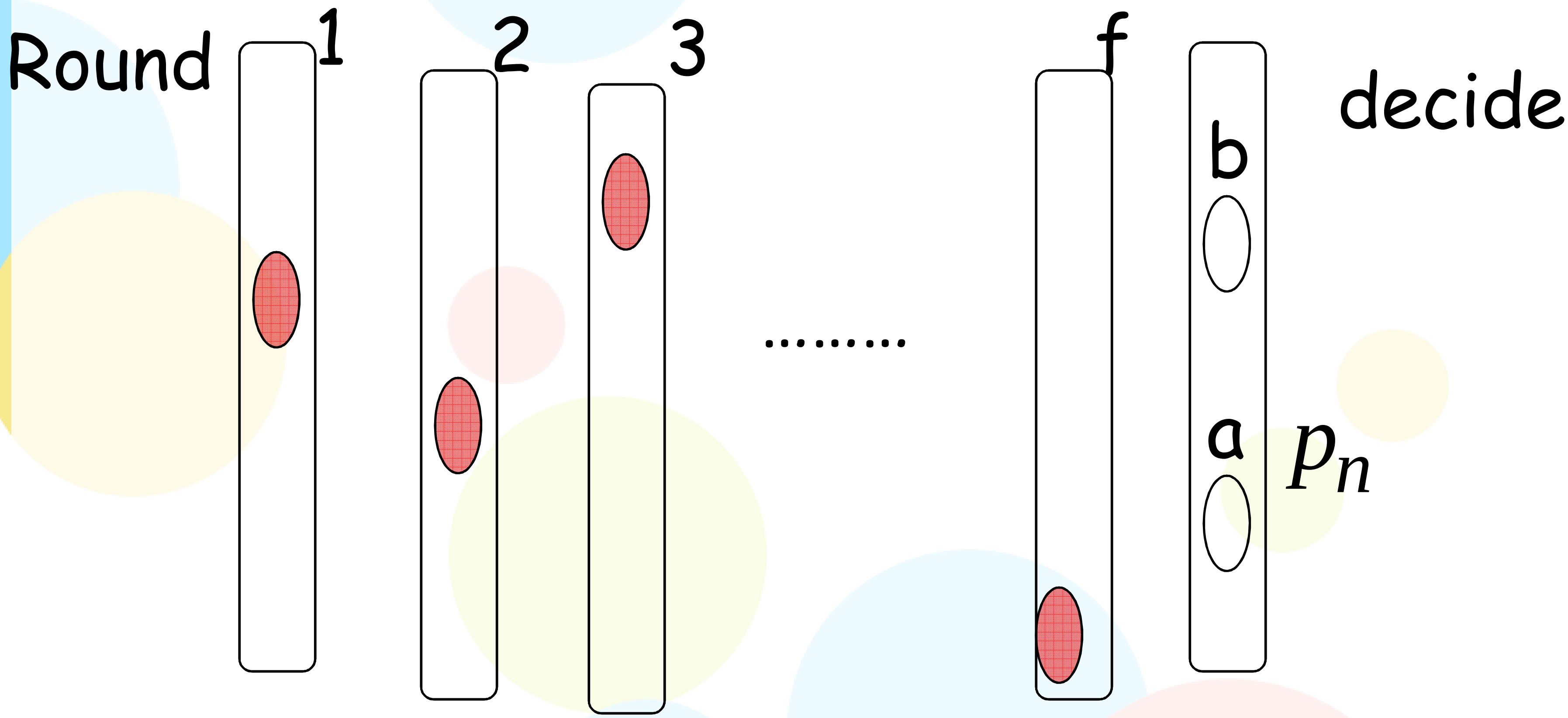
before process p_k fails, it sends value **a** to only one process p_m

Worst case scenario



Process p_n may decide **a**, and all other processes may decide another value **(b)**

Worst case scenario



Therefore f rounds are not enough

At least f+1 rounds are needed

Consensus in synchronous systems

Up to ***f*** faulty processes

Duration of round:
max. delay of B-multicast

Algorithm for process $p_i \in g$; algorithm proceeds in $f + 1$ rounds

On initialization
$Values_i^1 := \{v_i\};\ Values_i^0 = \{\};$

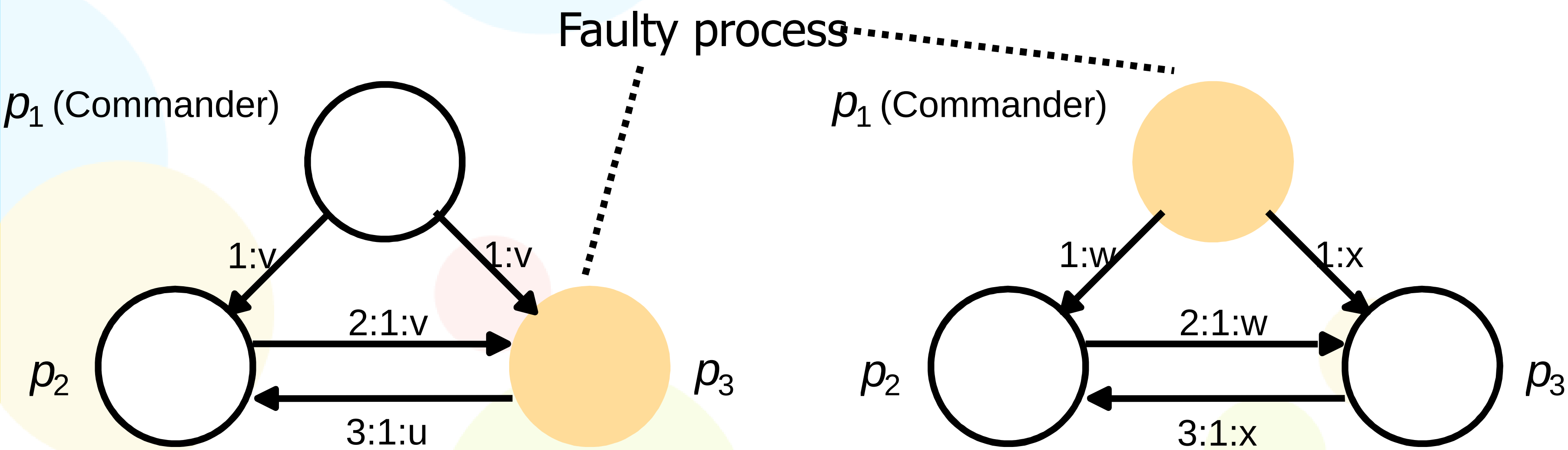
In round r $(1 \le r \le f + 1)$
B-multicast$(g,\ Values_i^r - Values_i^{r-1})$; // Send only values that have not been sent
$Values_i^{r+1} := Values_i^r;$
while (in round r)
{
On B-deliver(V_j) *from some* p_j
$Values_i^{r+1} := Values_i^{r+1} \cup V_j;$
}

After $(f + 1)$ *rounds*
Assign $d_i = minimum(Values_i^{f+1});$

Only crashes, no Byzantine faults

<u>Dolev & Strong, 1983:</u>
Any algorithm to reach consensus despite up to f failures requires (f +1) rounds.

Byzantine agreement: synchronous



3 says 1 says ‘u’

Nothing can be done to improve a correct process’ knowledge beyond the first stage:
- It <u>cannot</u> tell which process is faulty.

<u>Lamport et al, 1982:</u>
No solution for N = 3, f = 1

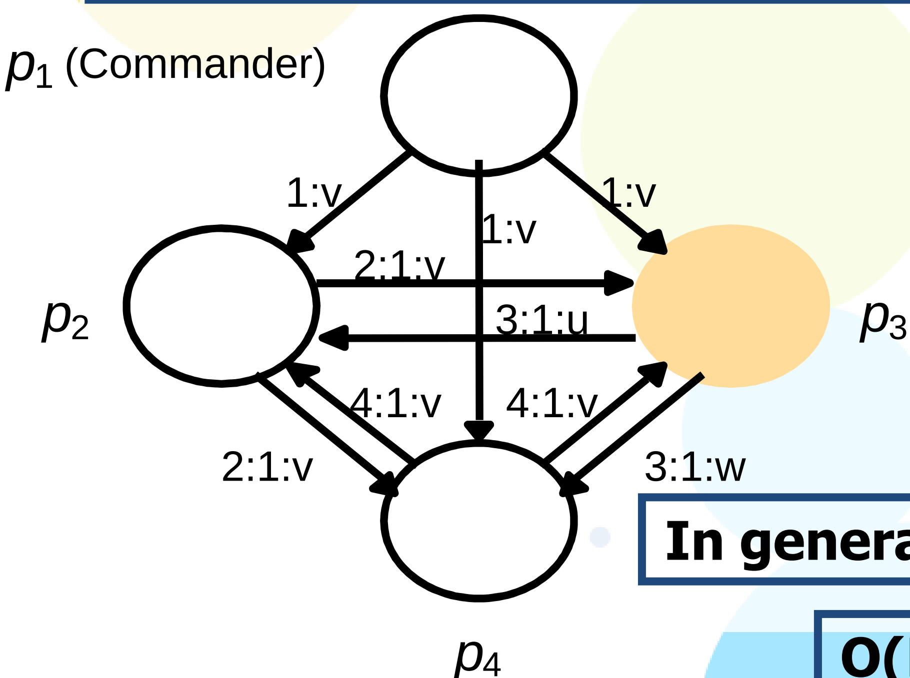
<u>Pease et al, 1982:</u>
No solution for N<= 3*f

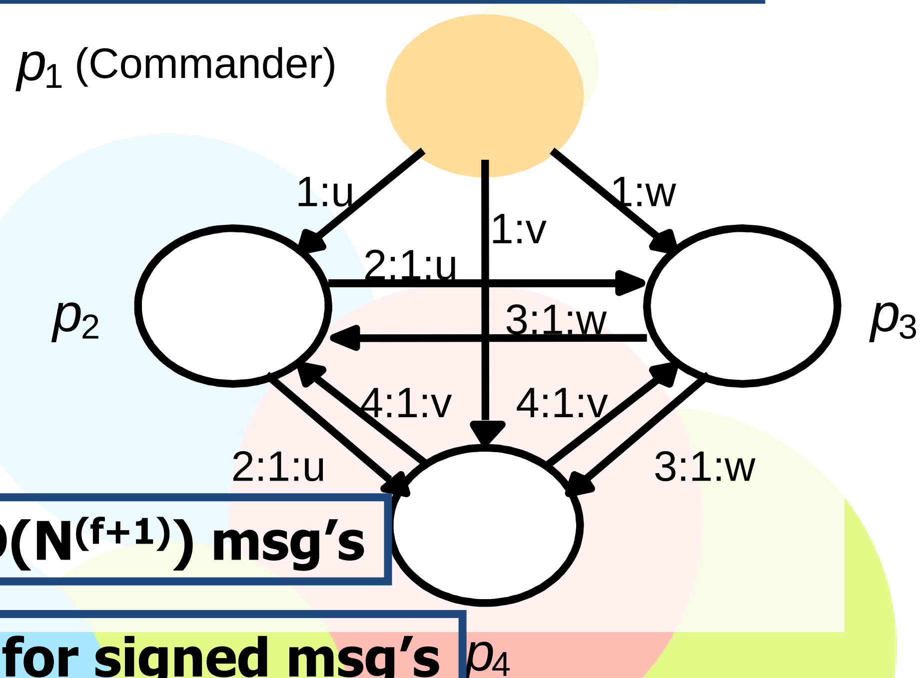
(assuming private comm. channels)

Byzantine agreement for N > 3*f

Example with N=4, f=1:

- 1^{st} round: Commander sends a value to each lieutenant
- 2^{nd} round: Each of the lieutenants sends the value it has received to each of its peers.
- A lieutenant receives a total of (N – 2) + 1 values, of which (N – 2) are correct.
- By majority(), the correct lieutenants compute the same value.

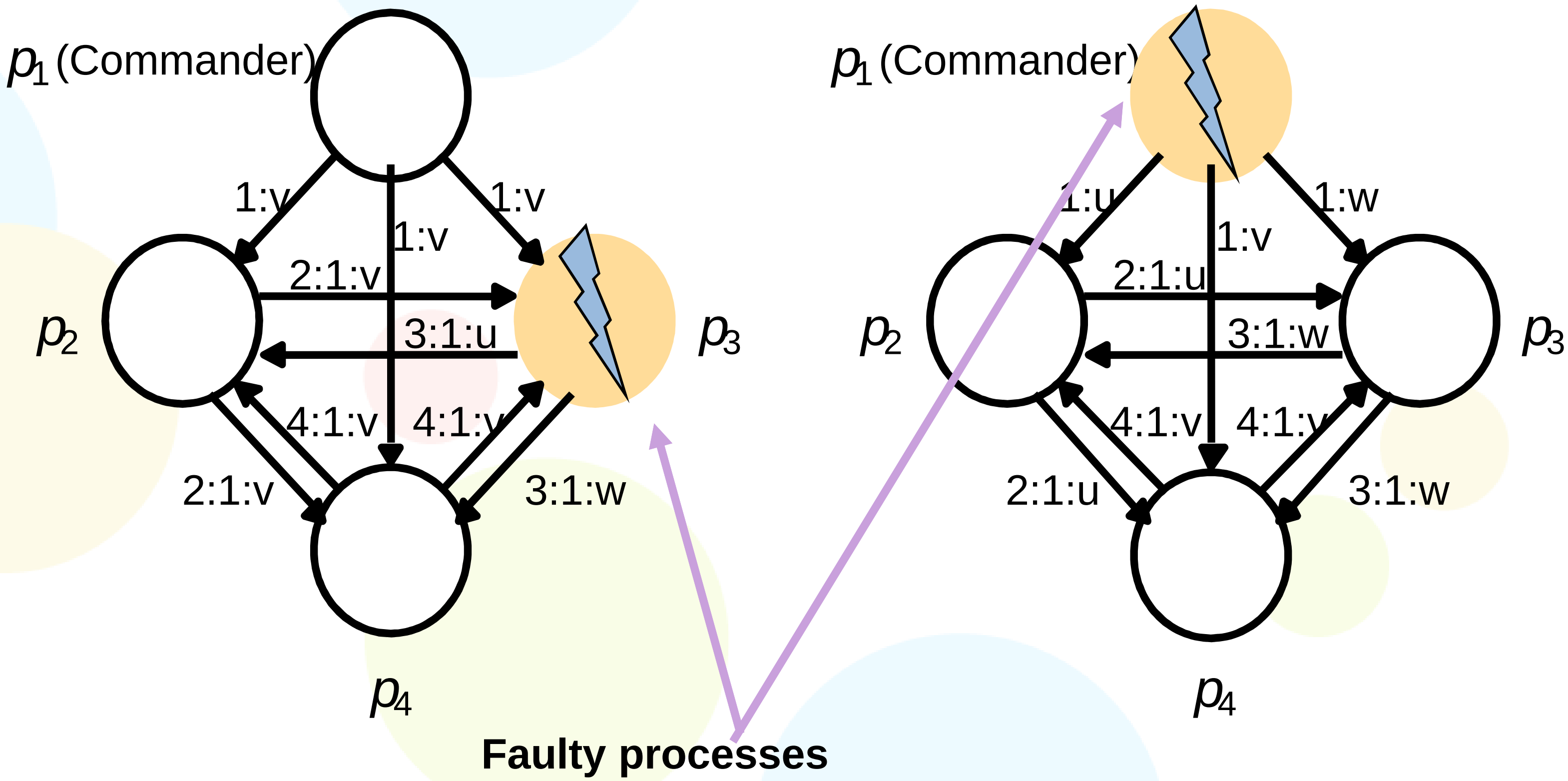




In general, $O(N^{(f+1)})$ msg's

$O(N^2)$ for signed msg's

Four Byzantine Generals: *N = 4, f = 1* in a Synchronous DS



p2 decides on *majority(v,u,v)* = *v*
p4 decides on *majority(v,v,w)* = *v*

p2, p3, p4 decide on *majority(u,v, w)* = $\perp$

Thank You