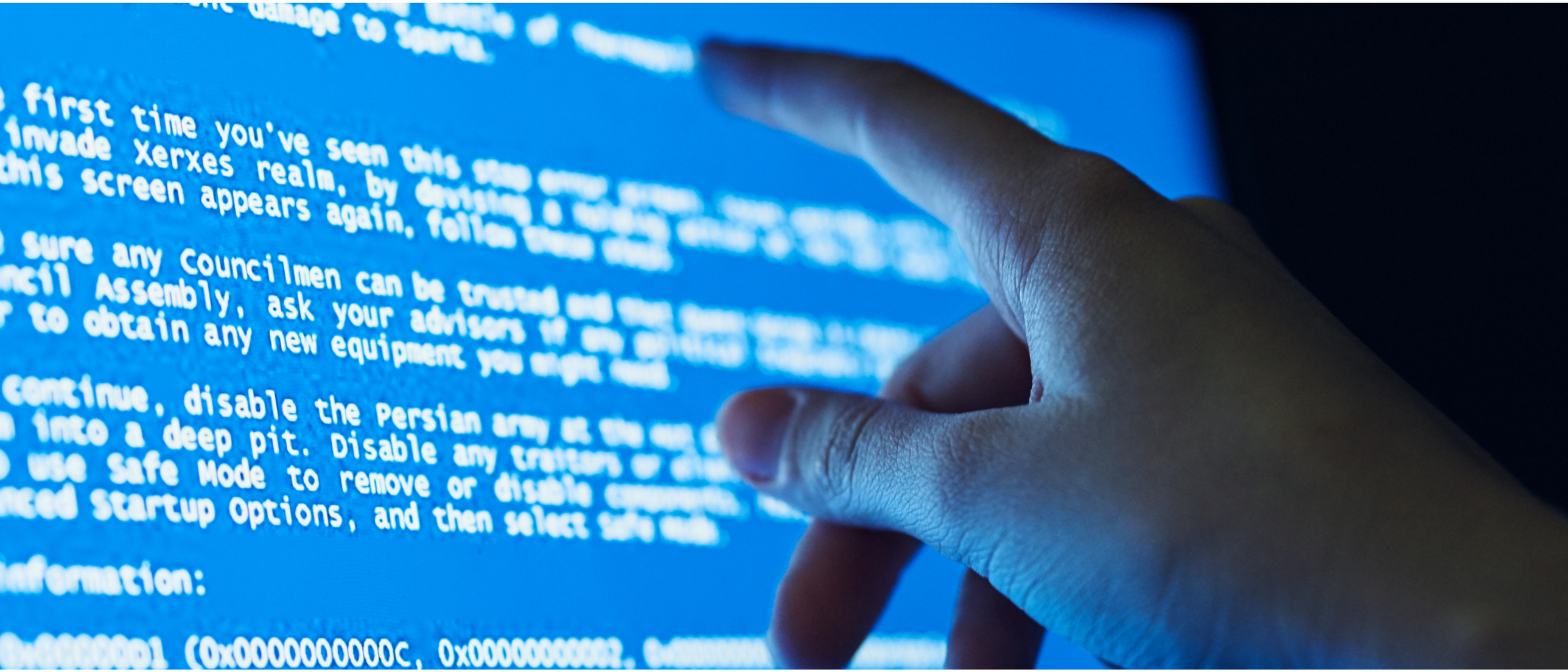
Distributed Algorithms

(83-453)

Prof. Ran Gelles

This pres. is Based on slides from University of L'Aquila, Italy (author unknown)



Fault Tolerance

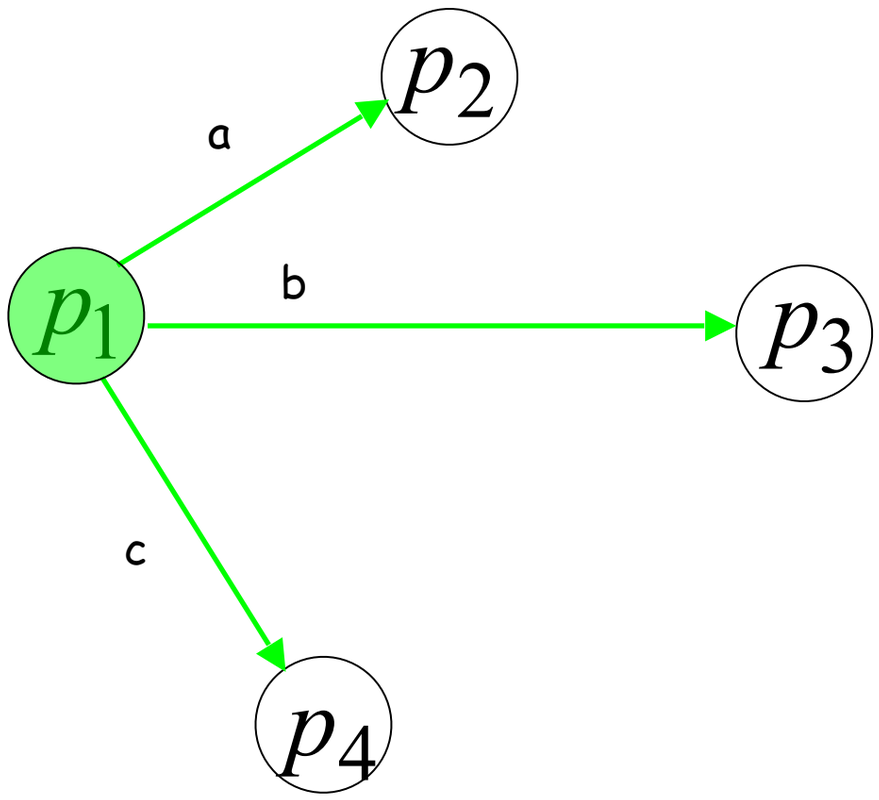
Failures and Noise types

Faults

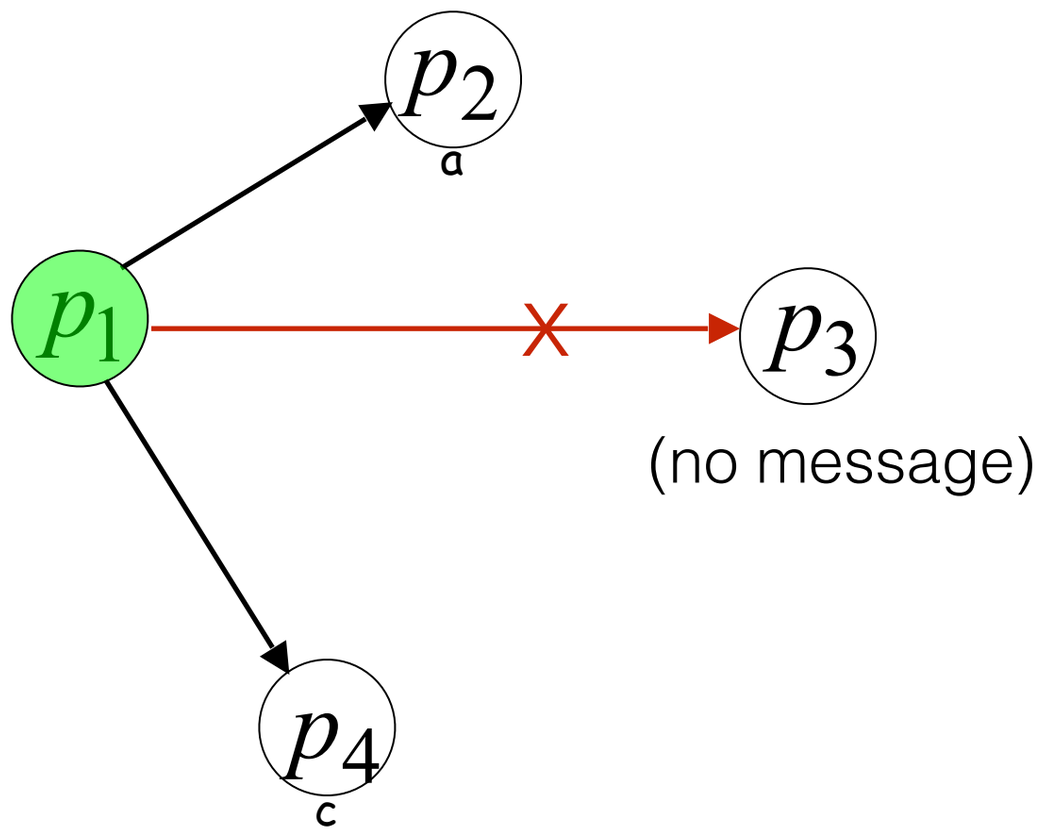
- In “real life” we encounter different forms of failures and faults:
  - **Link fault** - the communication channel gets disconnected (**forever**).
    Link Noise: messages received incorrectly (sometimes)
  - **Crash fault** - A node stops working (**forever**).
    Variant: nodes might get back online (in what state?)
  - **Byzantine fault** - nodes perform arbitrary actions - they don’t follow the algorithm and send arbitrary messages

Link Faults

normal



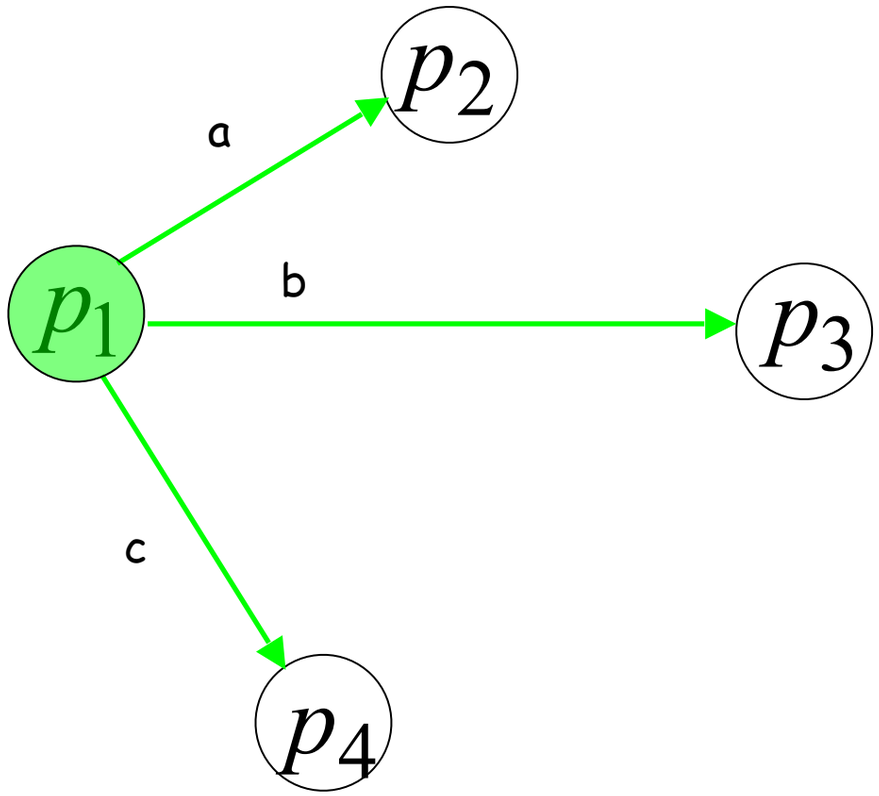
Failure (p1,p3)



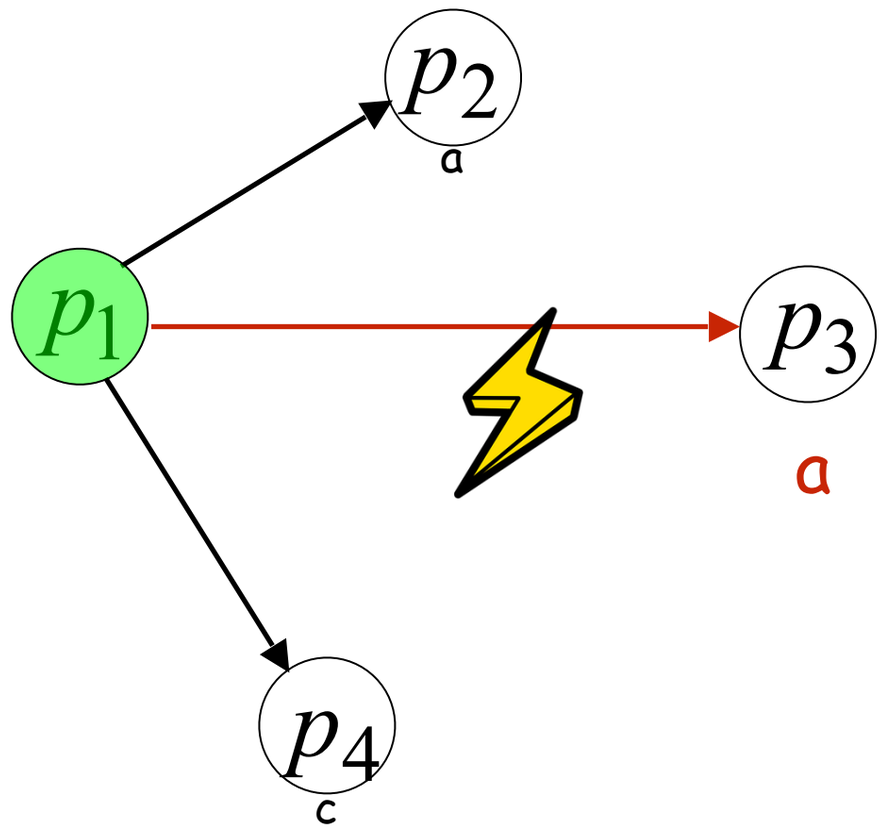
Note: both synchronous / asynchronous

Link Faults

normal



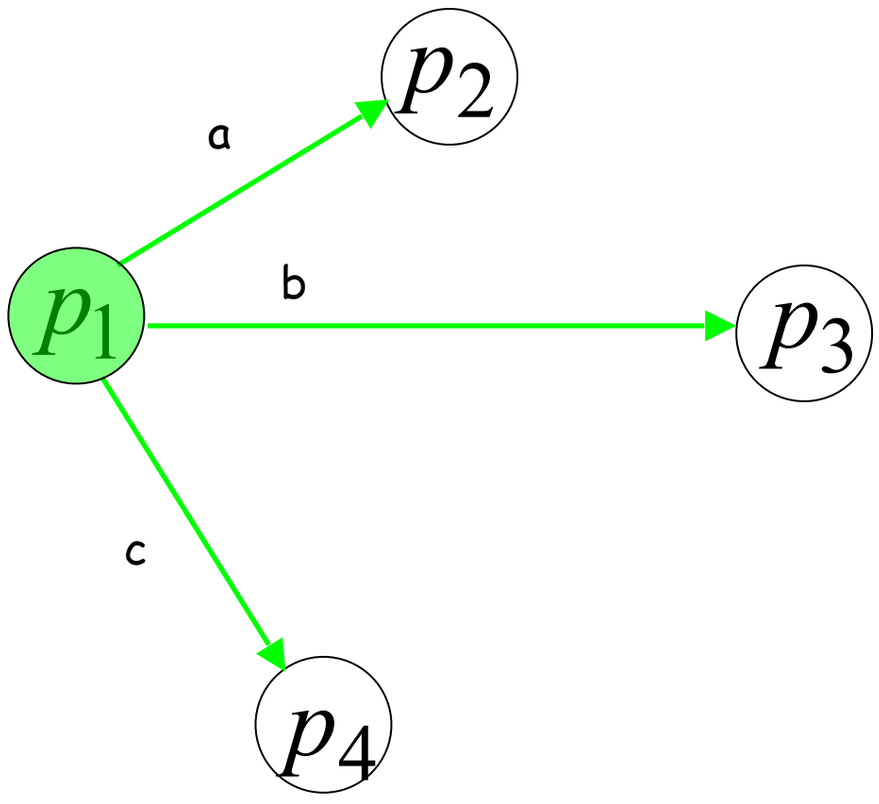
Noise (p1,p3)



Common Noise types:
different message, inject a message, erase/delete message

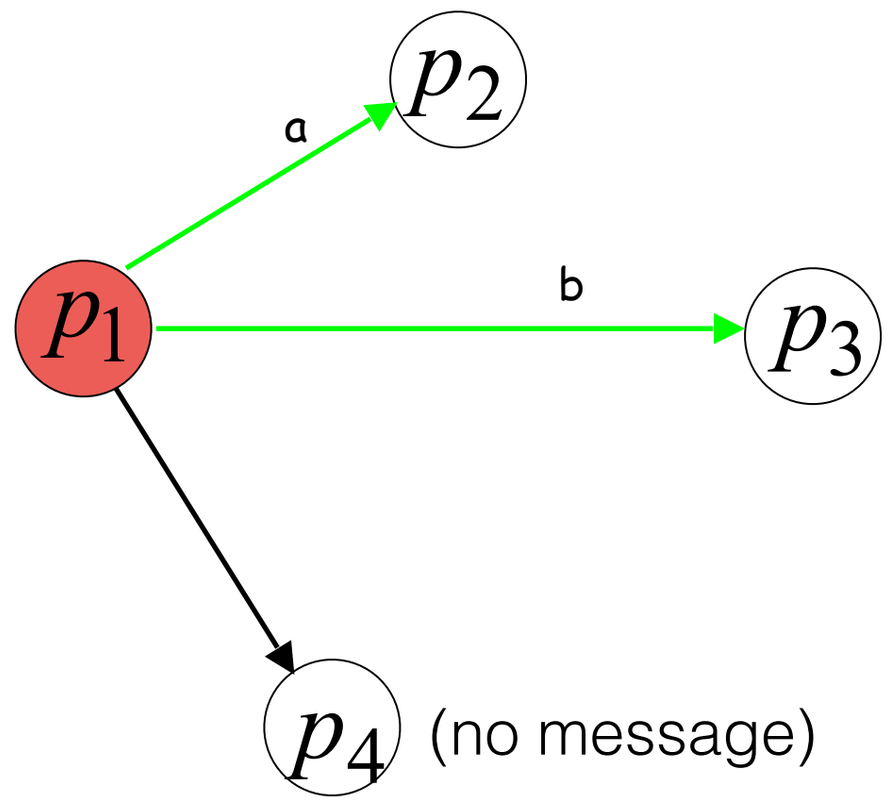
Crash Faults

normal



messages are atomic:
either correct or missing.

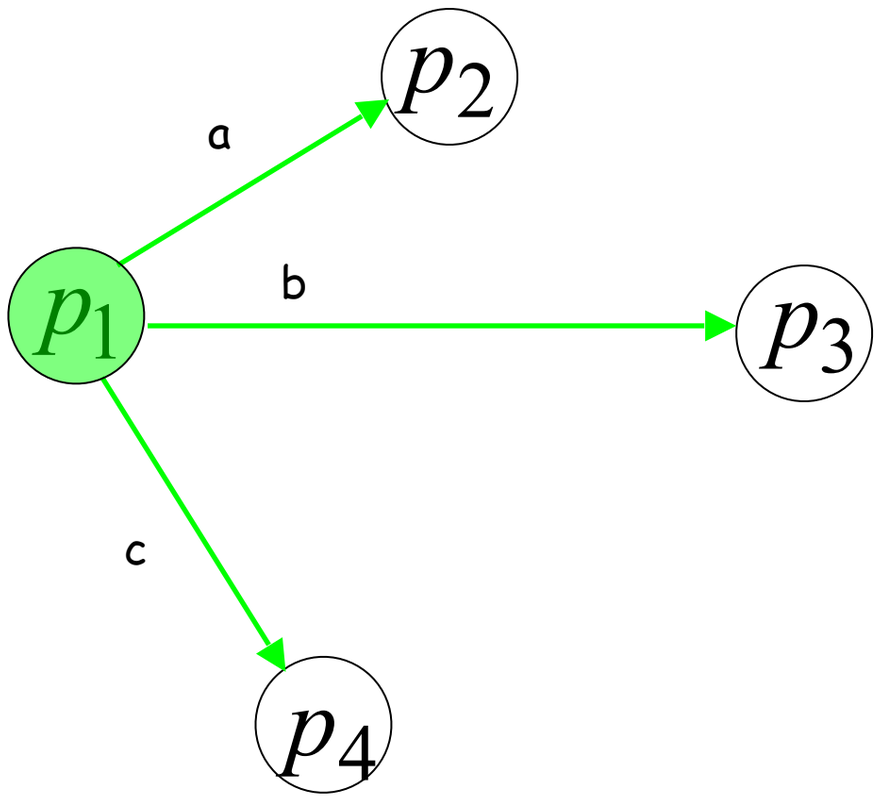
p1 crashes



crash happens after sending
to p_2/p_3, but before sending to p_4
(even in synchronous!)

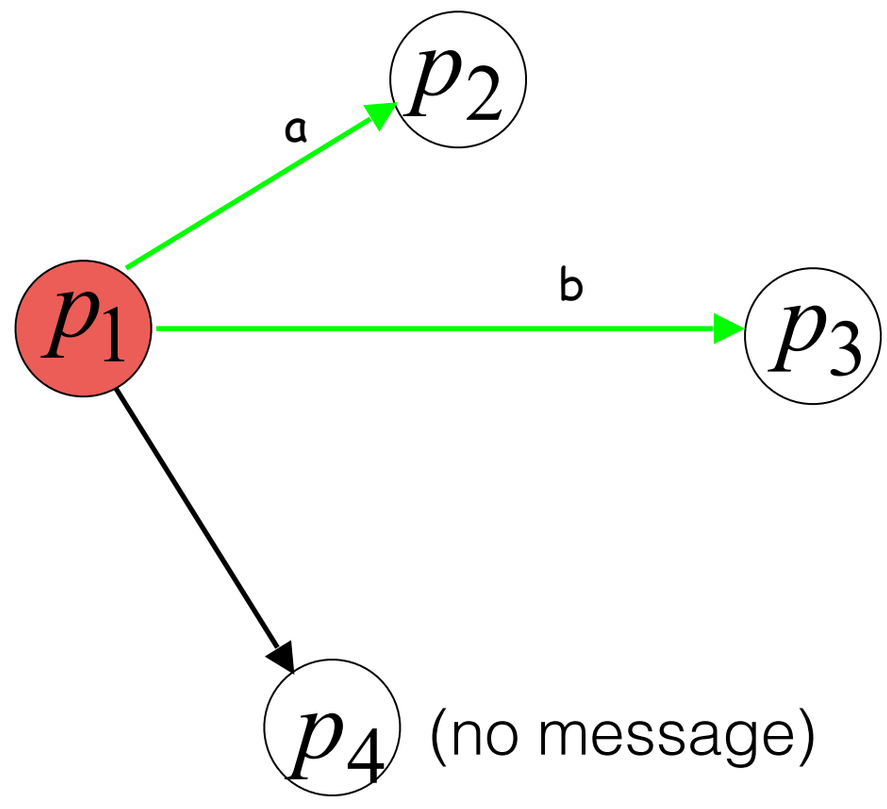
Crash Faults

normal



once p_1 has crashed: it stops sending messages forever

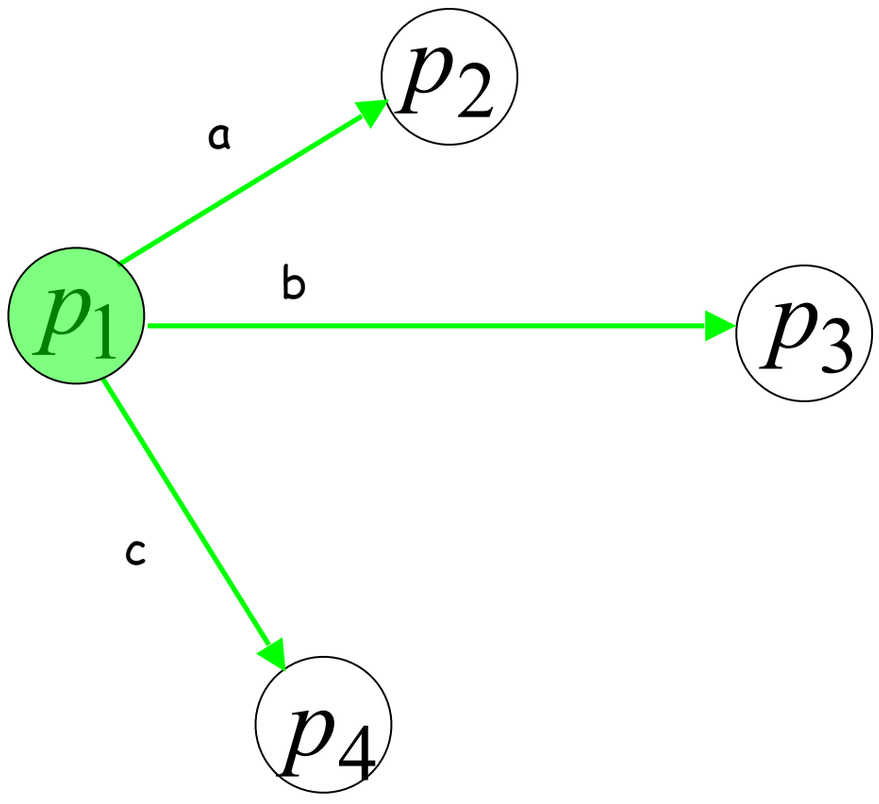
p1 crashes



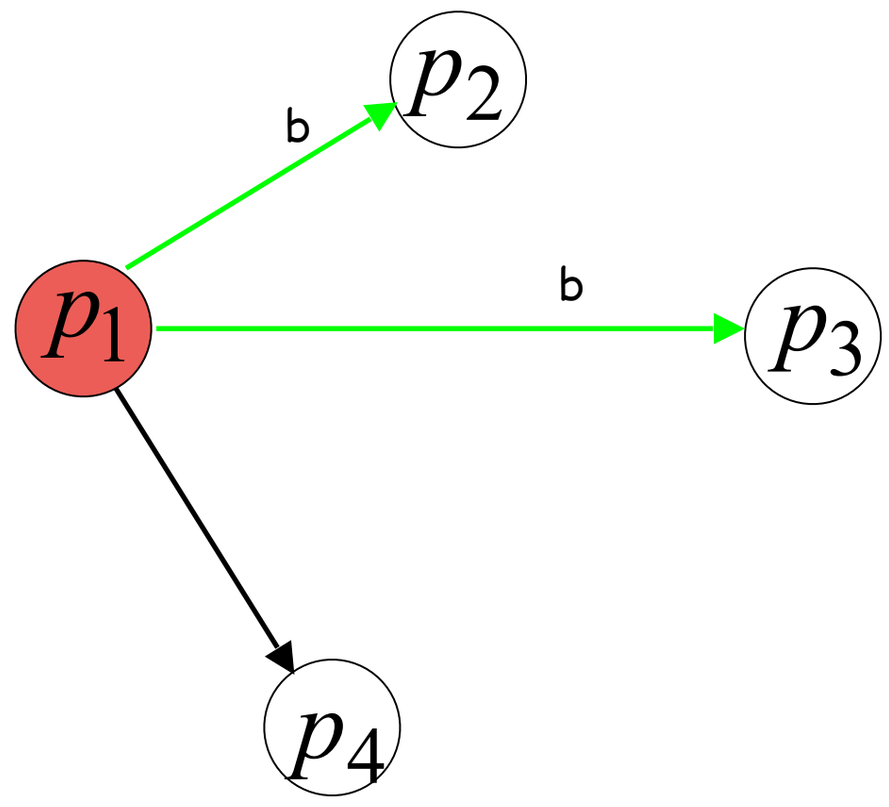
The network might become disconnected!

Byzantine Faults

normal



p1 byzantine



- We think of p_1 as being evil (adversarial), doing the thing that will hurt us the most = worst case faults
- In the next round, p_1 might behave “nicely”, to fool us

Fault-Tolerant Computation

- We will focus on synchronous protocols, Topology is a clique K_n, so all connected to all
- crash failures

(later we will get to byzantine, and link noise)

- Let f be the maximal number of nodes that can fail
- An algorithm that works correctly even if **up to** f nodes might fail, is called f-resilient.
- crashed nodes need not give output, but all the other nodes must give the correct output

Consensus

Fault-Tolerant Computation

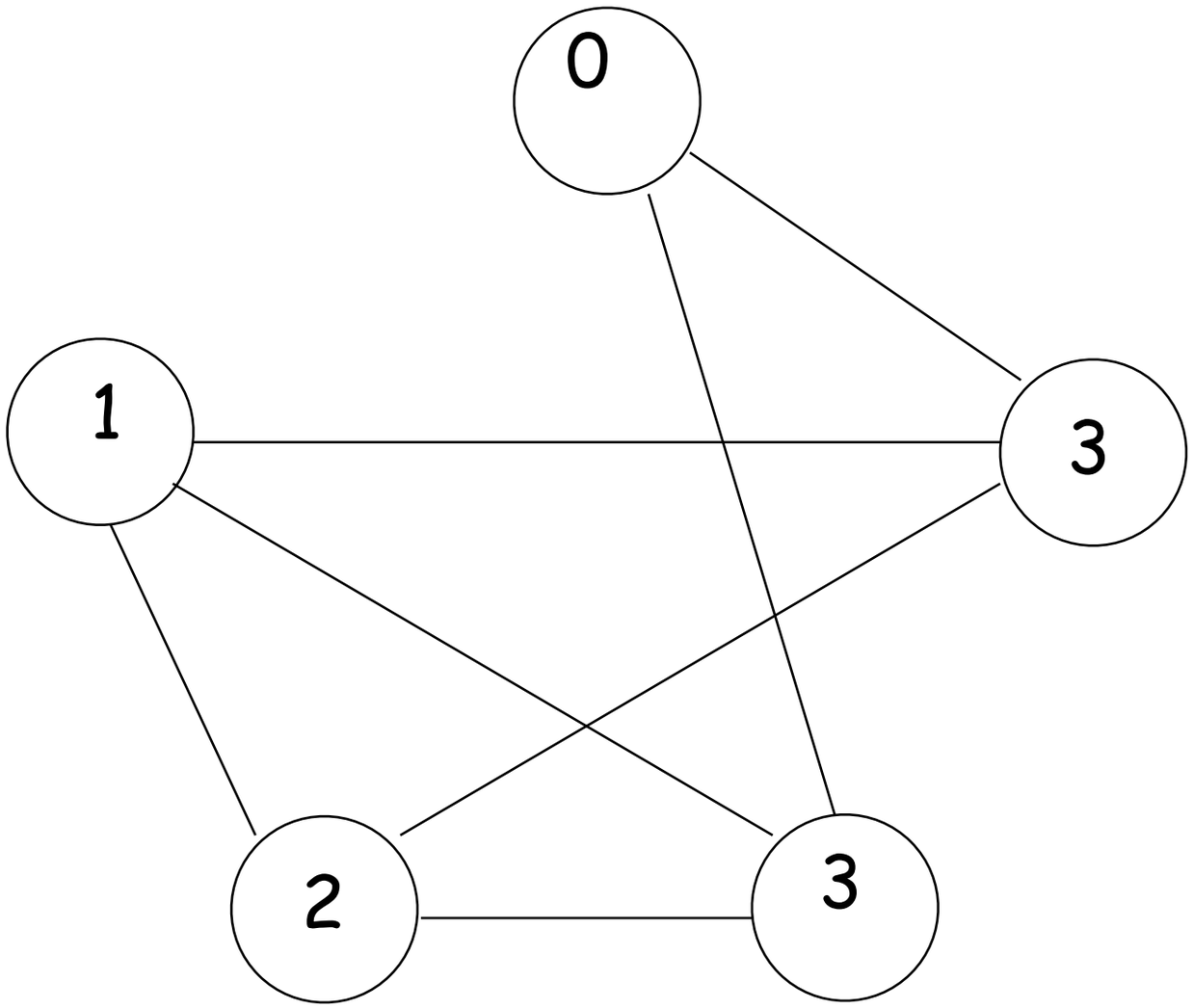
Consensus

- Consensus = agreement
- **Input**: node i gets a private input x_i .
- **Output**: all non-faulty nodes output the same value v .

- **Required Properties:**
  - **termination:** every non-faulty node eventually terminates (gives an output o_i)
  - **agreement:** all non-faulty nodes agree on the same output v, $\forall i \in \text{NonFaulty}, o_i = v$
  - **validity:** the agreed output is a valid input of some party, $v \in \{x_i\}$.

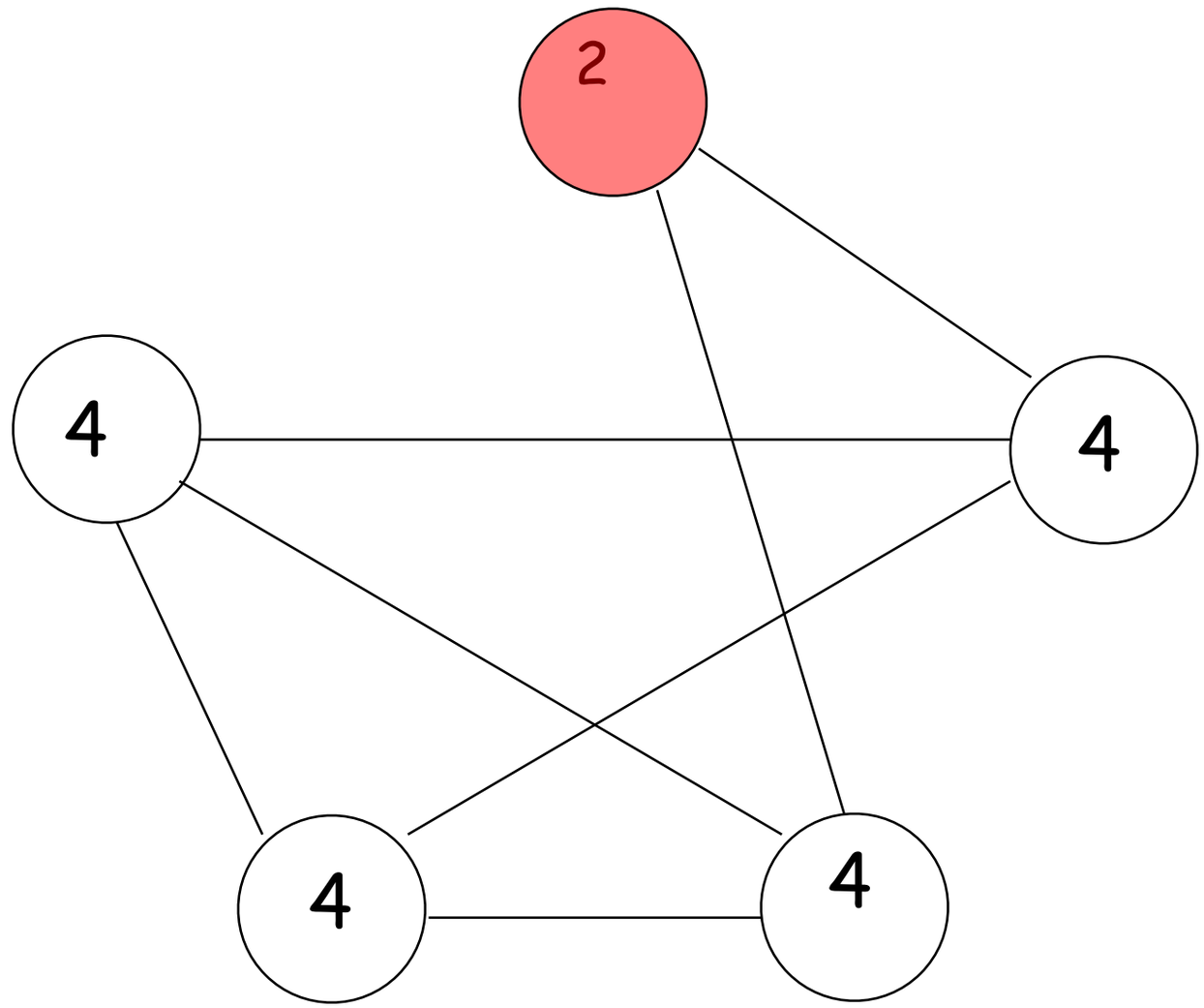
Agreement

Start



Everybody has an initial value

Finish



All non-faulty must output the same value

Validity

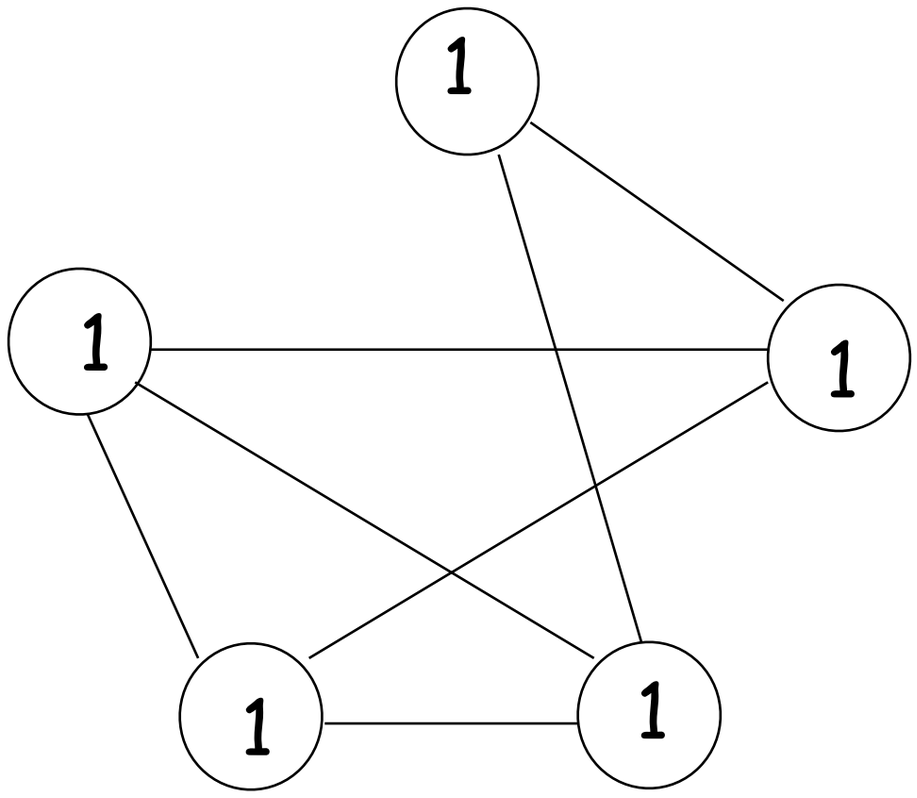
The output is a valid input

Thus:

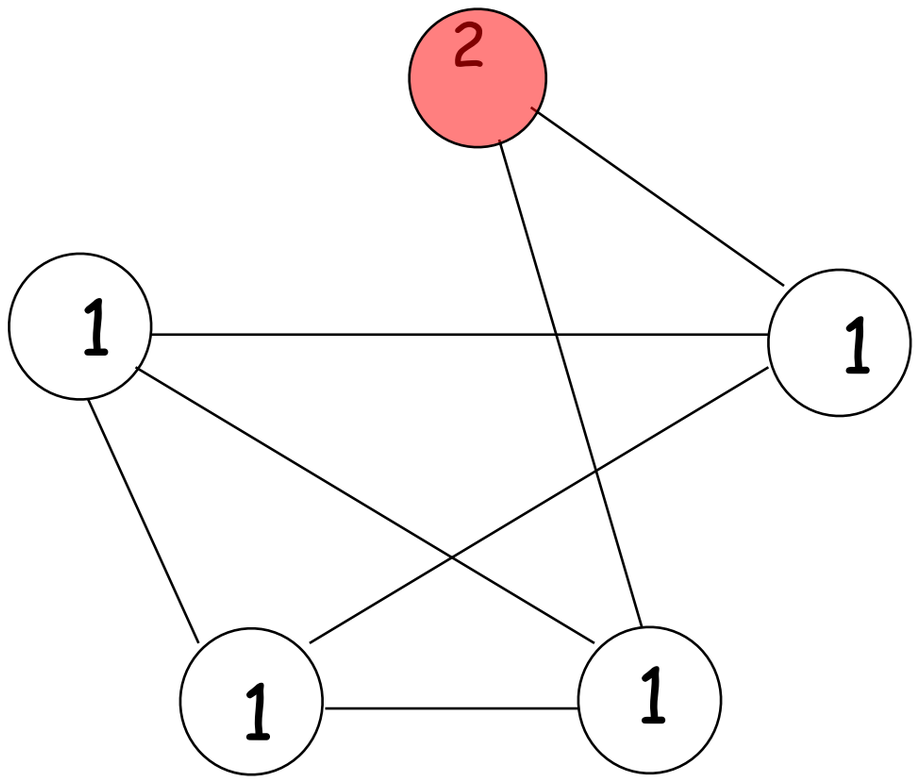
If everybody starts with the same value,

then non-faulty nodes must output that value

Start



Finish

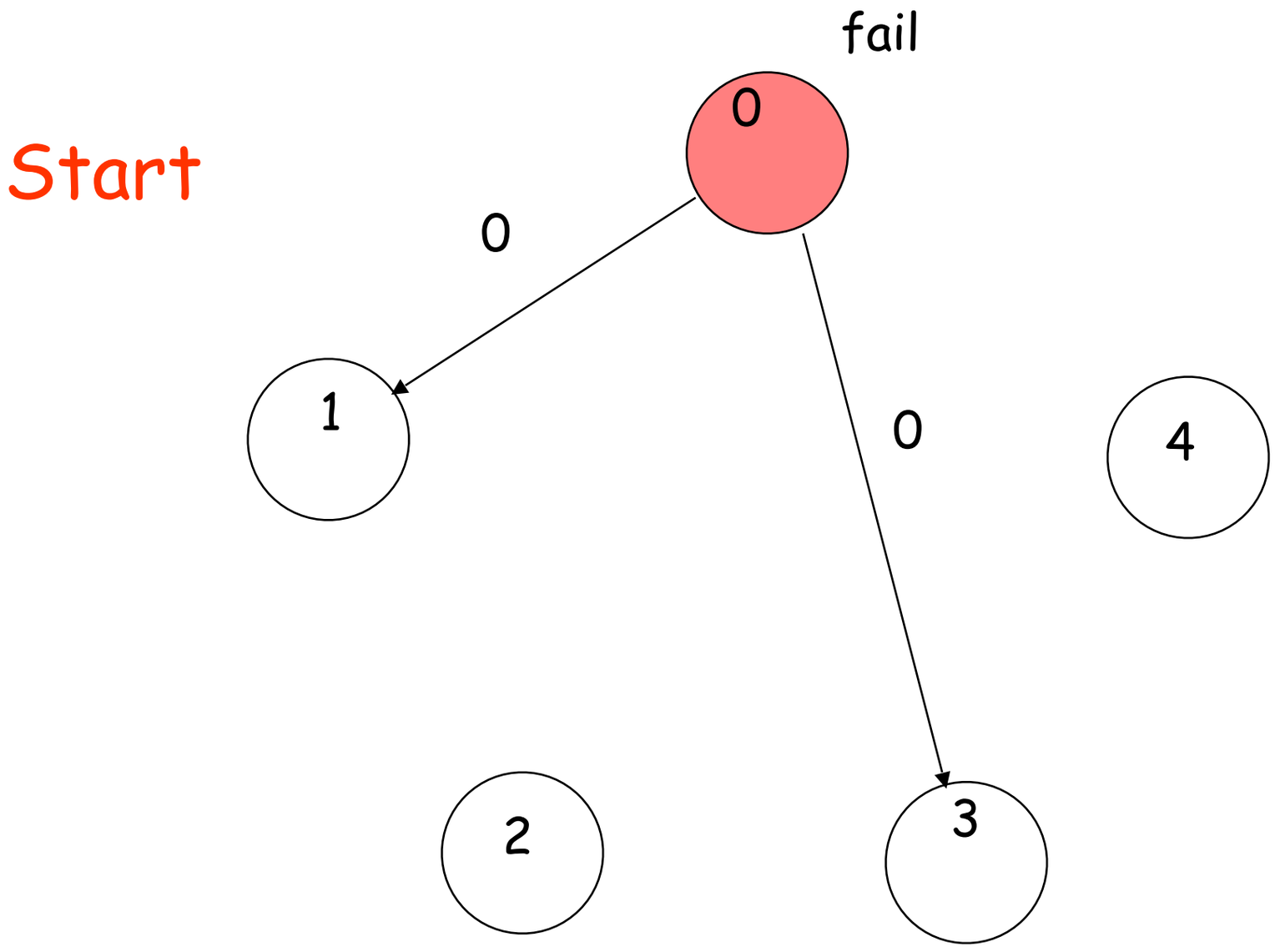


Fault-Free Consensus

- If there are no faults, f=0, then consensus is easy!
- Suggestions?!
  - **solution 1**: everybody broadcasts their value. Take minimal value
  - **solution 2**: elect a leader - leader broadcasts its own input to everybody
- What happens if there are faults?

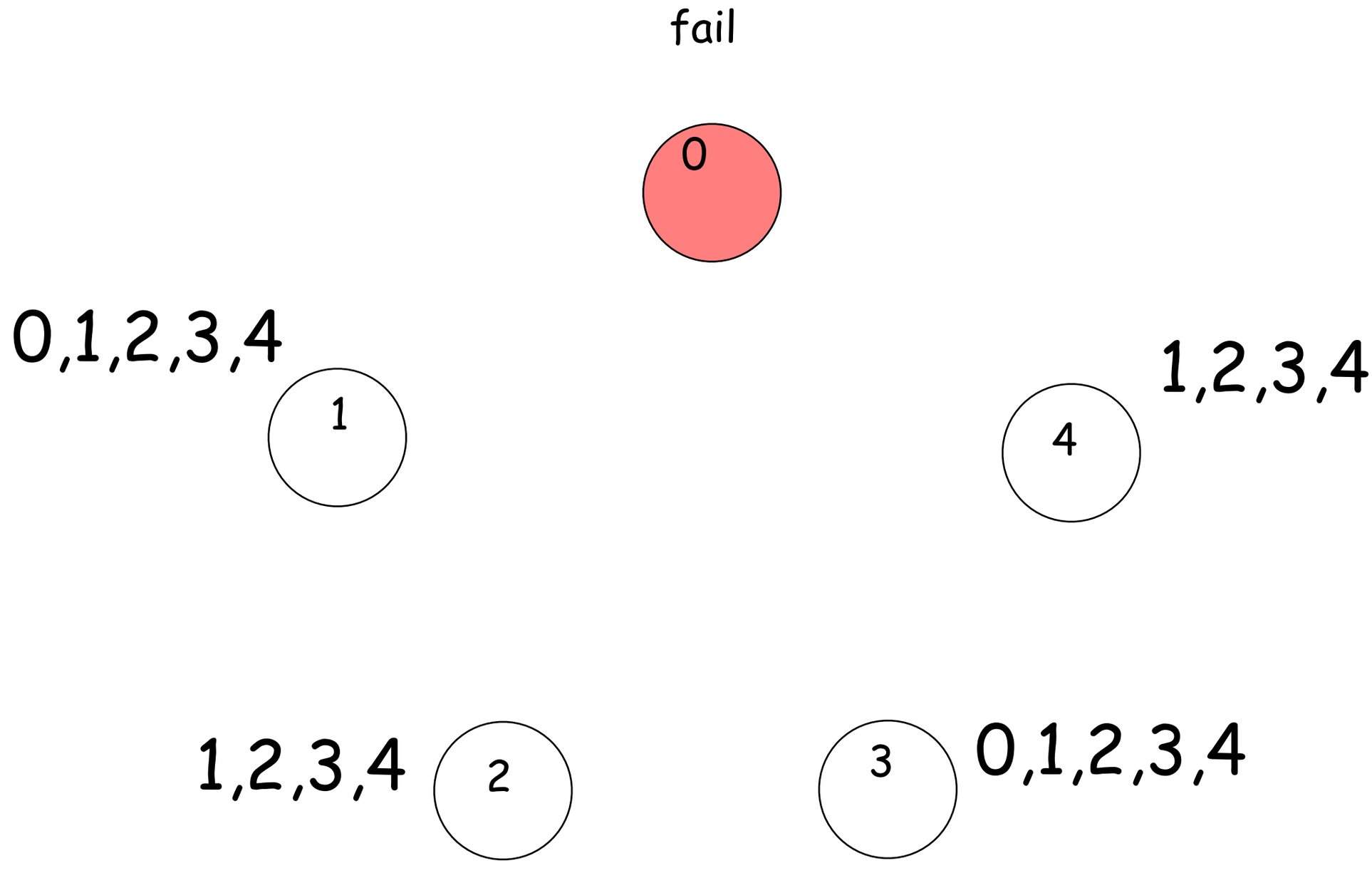
Consensus with Crash Failures

The simple algorithm doesn't work

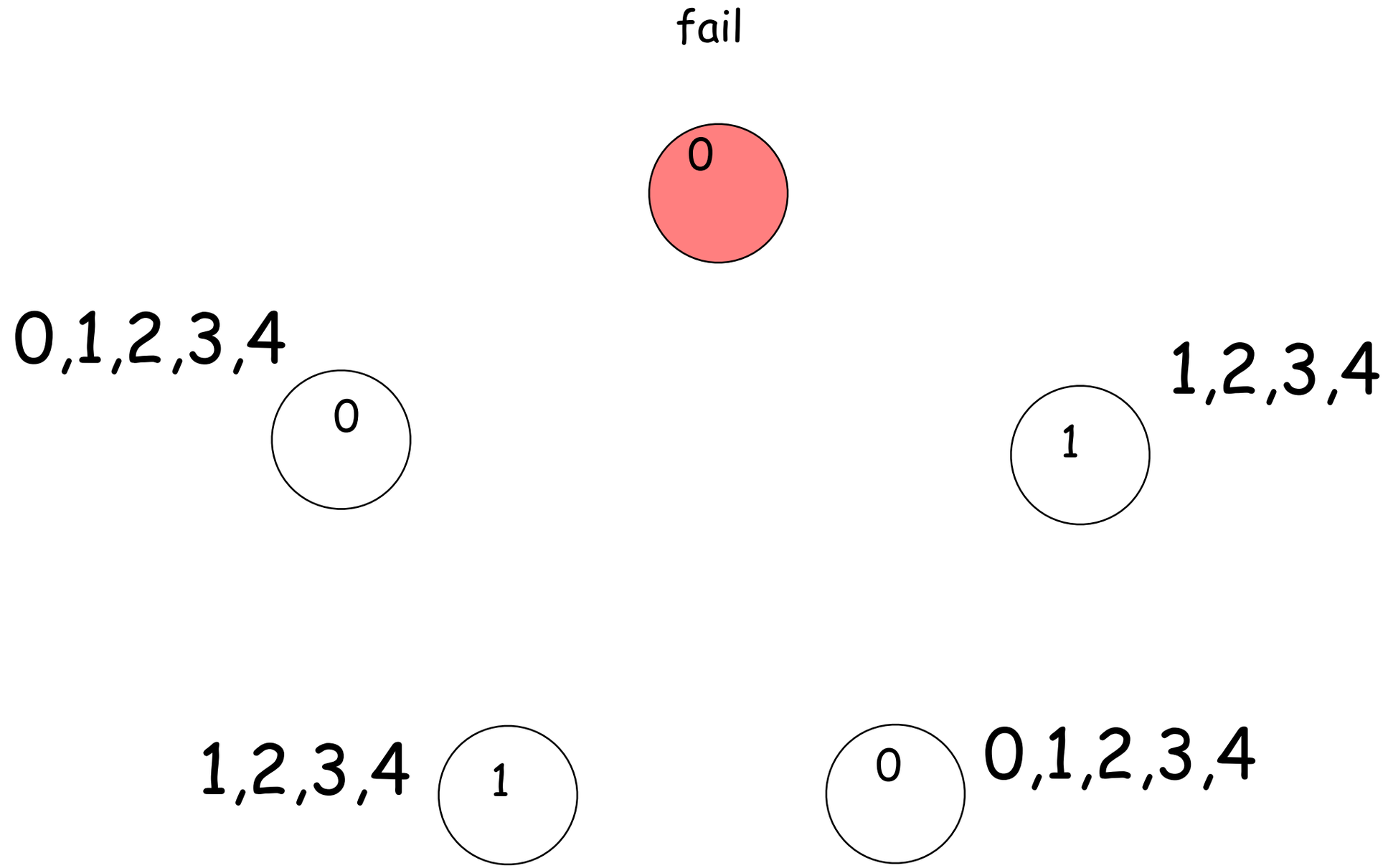


The failed processor doesn't broadcast its value to all processors

Broadcasted values

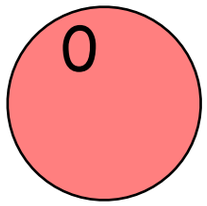


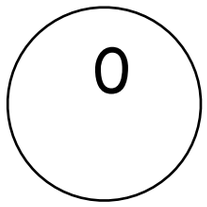
Decide on minimum

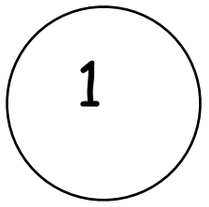


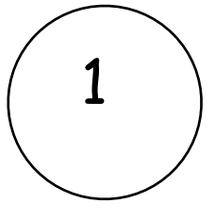
Finish

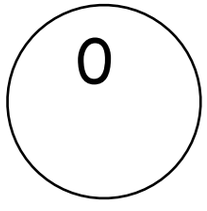
fail











No agreement!!!

f-Resilient Consensus

- **Model**: synchronous, clique, anonymous(?), deterministic, LOCAL.

Input: a value **x**
Output: all non-faulty nodes output the same value **v**

1 init **V** = { **x** }
2 **for** round i = 1 to $f+1$ **do**
3 send **V** to all parties
4 receive S_t from party t // (if no message, t is faulty)
5 **V** ← **V** ∪ $\left(\bigcup_t S_t\right)$ for all received S_t in this round
6 **output** v = min(**V**)

f-Resilient Consensus

- **Analysis**
  - **Termination**: trivial. Every node runs for at most $f+1$ rounds (unless crashed beforehand).
  - **Agreement**: there must exist a round $j \in [1, f+1]$ during which **no** node crashes.
    Then, at the end of round j, all non-faulty nodes set their own V to be the same $V^* = \bigcup_{i \in NF} V_i(j-1)$
    that is, the union of all the V's of the end of round j -1 for all the non-faulty (NF) parties of round j.

f-Resilient Consensus

- **Agreement** (cont'):
  - If all parties hold the same V*, any future message contains exactly V*, therefore, at round f+1, all non-faulty parties hold V*.
  - The output of all parties is min(V*).
- **Validity:**
  - **claim**: at any given round, any V holds only valid inputs. By induction: **Base case** V=$\{x\}$, which is a valid input.

    Step: we union V with S_t which are just V's of previous round, hence they hold only valid inputs by the induction hypothesis.

f-Resilient Consensus

- **Note:** proof will break if we:
  - allow byzantine errors
  - allow crashed parties to resurrect.
- **Exercise**: Go over the proof and identify the point(s) that break for each of the above!

Complexity?

- $n \geq f$ parties
- f +1 rounds
- Communication: $O(f\ n^2) = O(n^3)$ messages of size $O(n \log\ n)$ bits.
- Communication can be reduced: instead of sending the entire V, send only the new elements. Gives $O(k\ n^2)$ messages of $O(\log\ k)$ bits assuming there are k different inputs; still $O(n^3)$ messages.

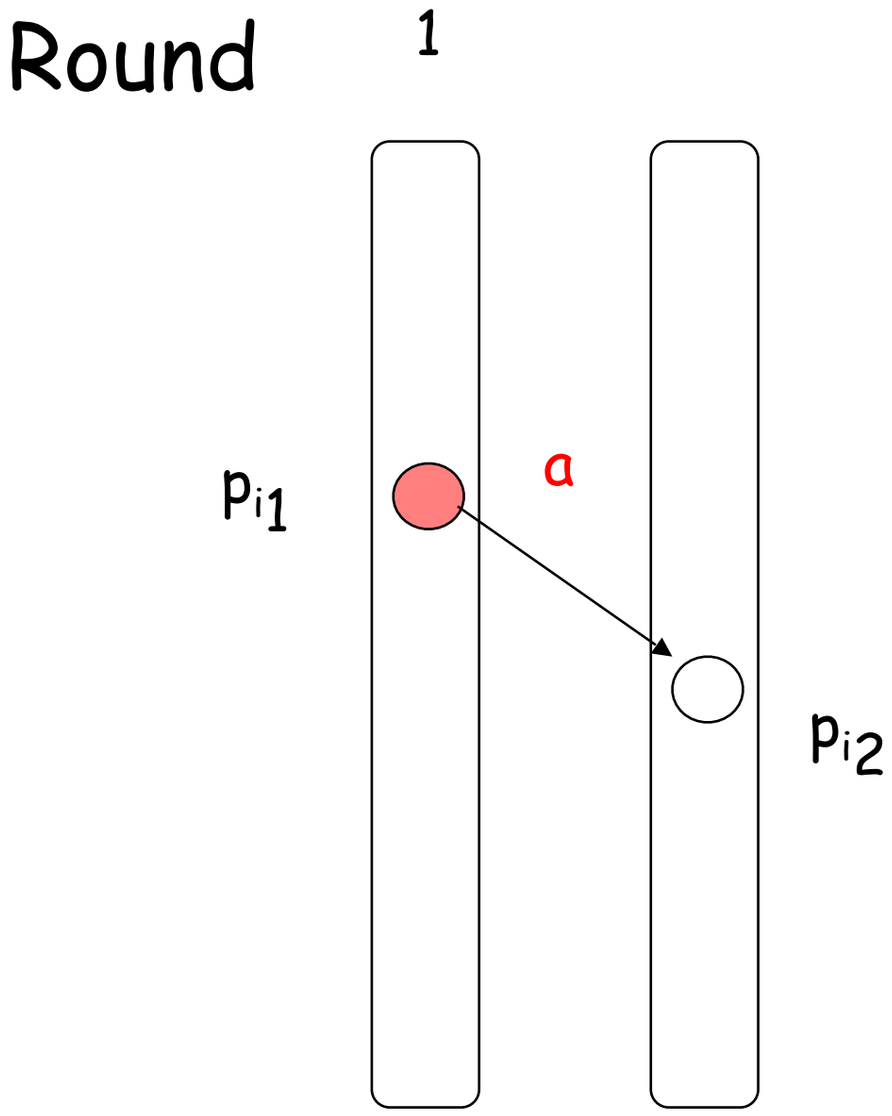
A Lower Bound

- **Theorem**: any f-resilient consensus algorithm takes at least f+1 rounds (in some execution).
- The proof is somewhat technical, so we will see a "proof" for a simpler case that gives enough motivation:
- **weaker theorem**: Our minimum based algorithm must take f+1 rounds to be f-resilient.

A Lower Bound

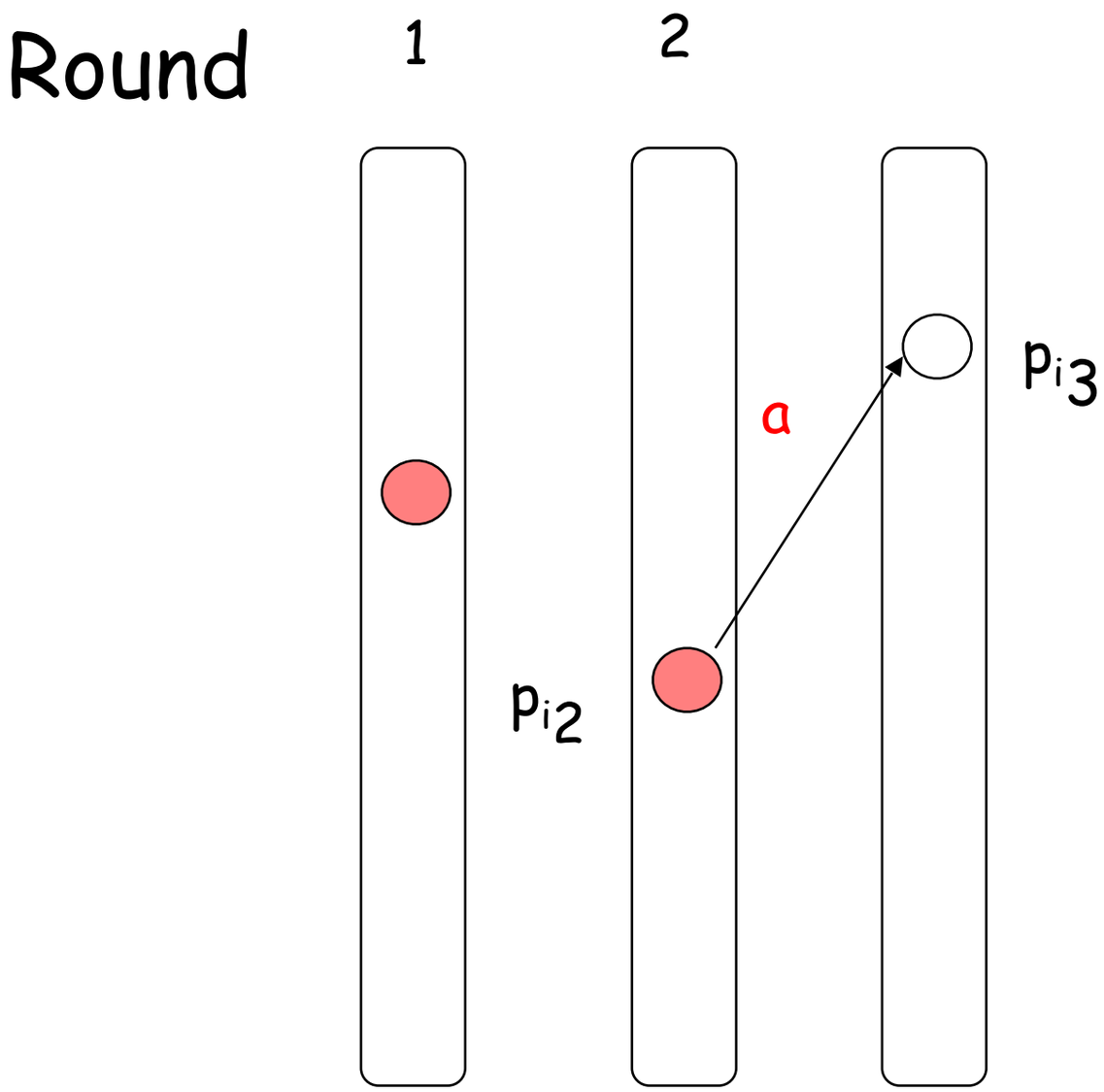
- To prove the weaker theorem, we will show an example for an execution that fails.
- The **Worst Case** pattern of failures:
  - Every round, there is a node that crashes.
  - Every round, that node sends **one** message and then crashes.

Worst case scenario



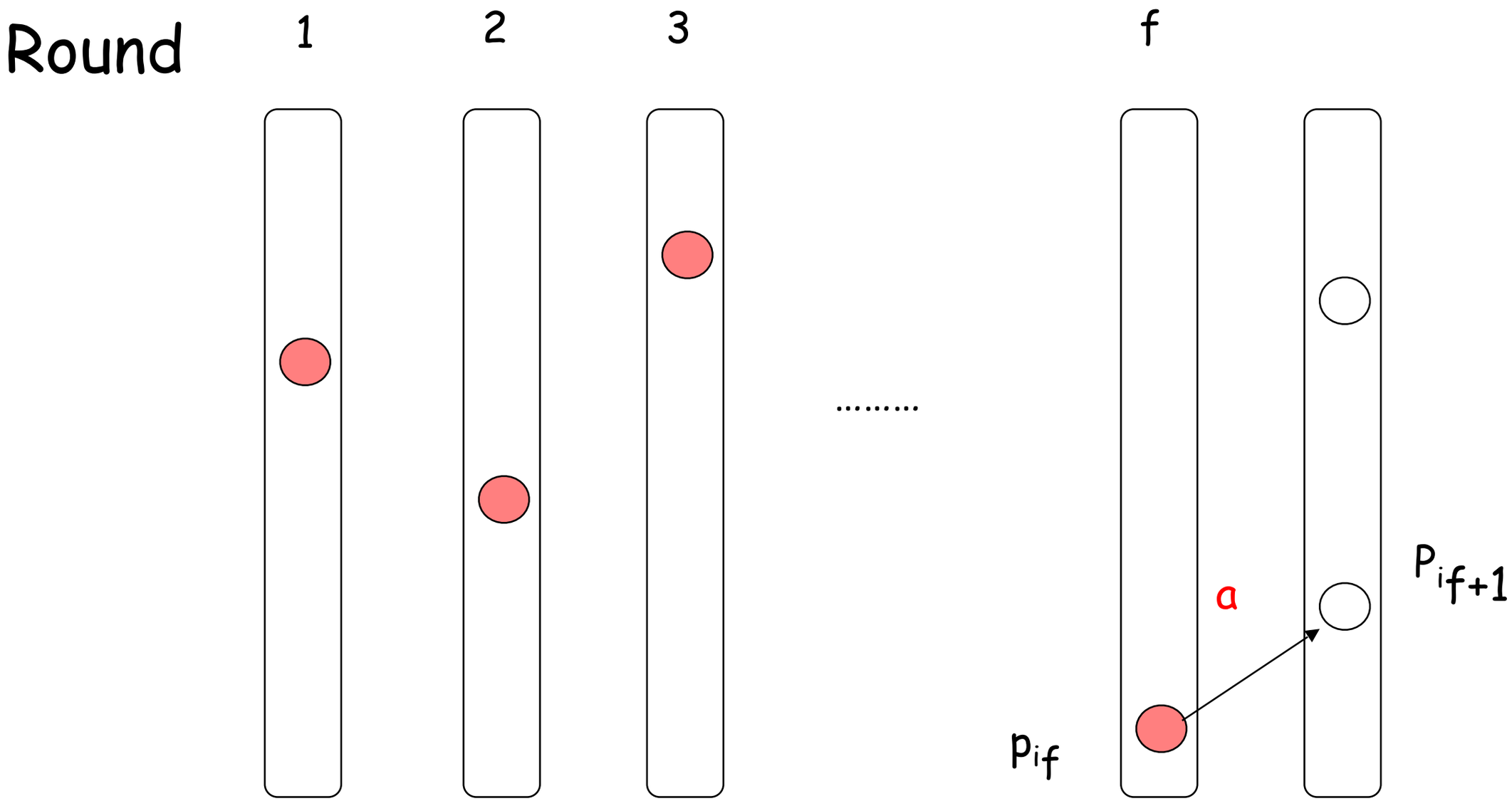
before processor p_{i1} fails, it sends its value a to only one processor p_{i2}

Worst case scenario



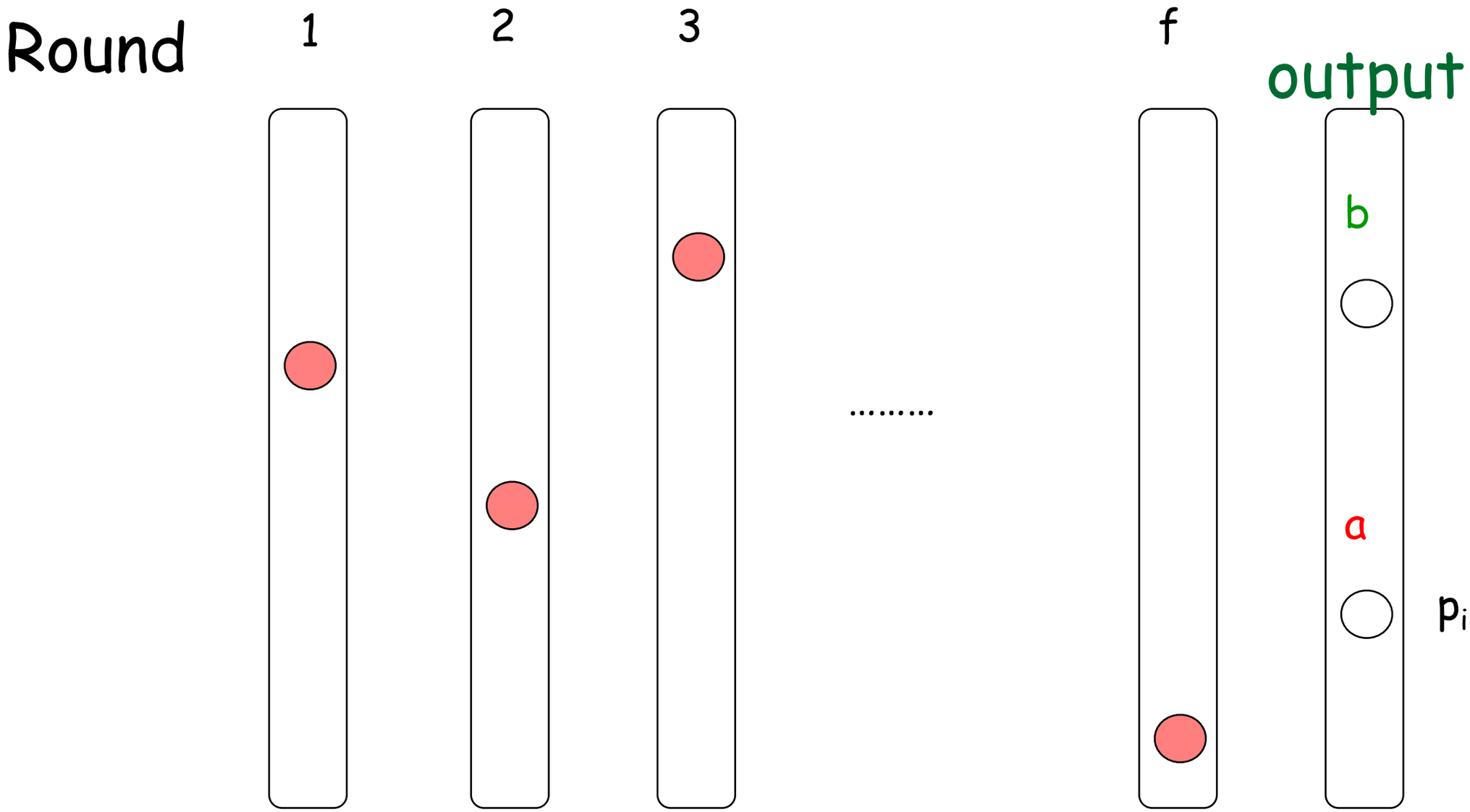
before processor p_{i_2} fails, it sends its value a to only one processor p_{i_3}

Worst case scenario



Before processor p_{if} fails, it sends its value a to only one processor p_{if+1}. Thus, at the end of round f only one processor knows about a

Worst case scenario

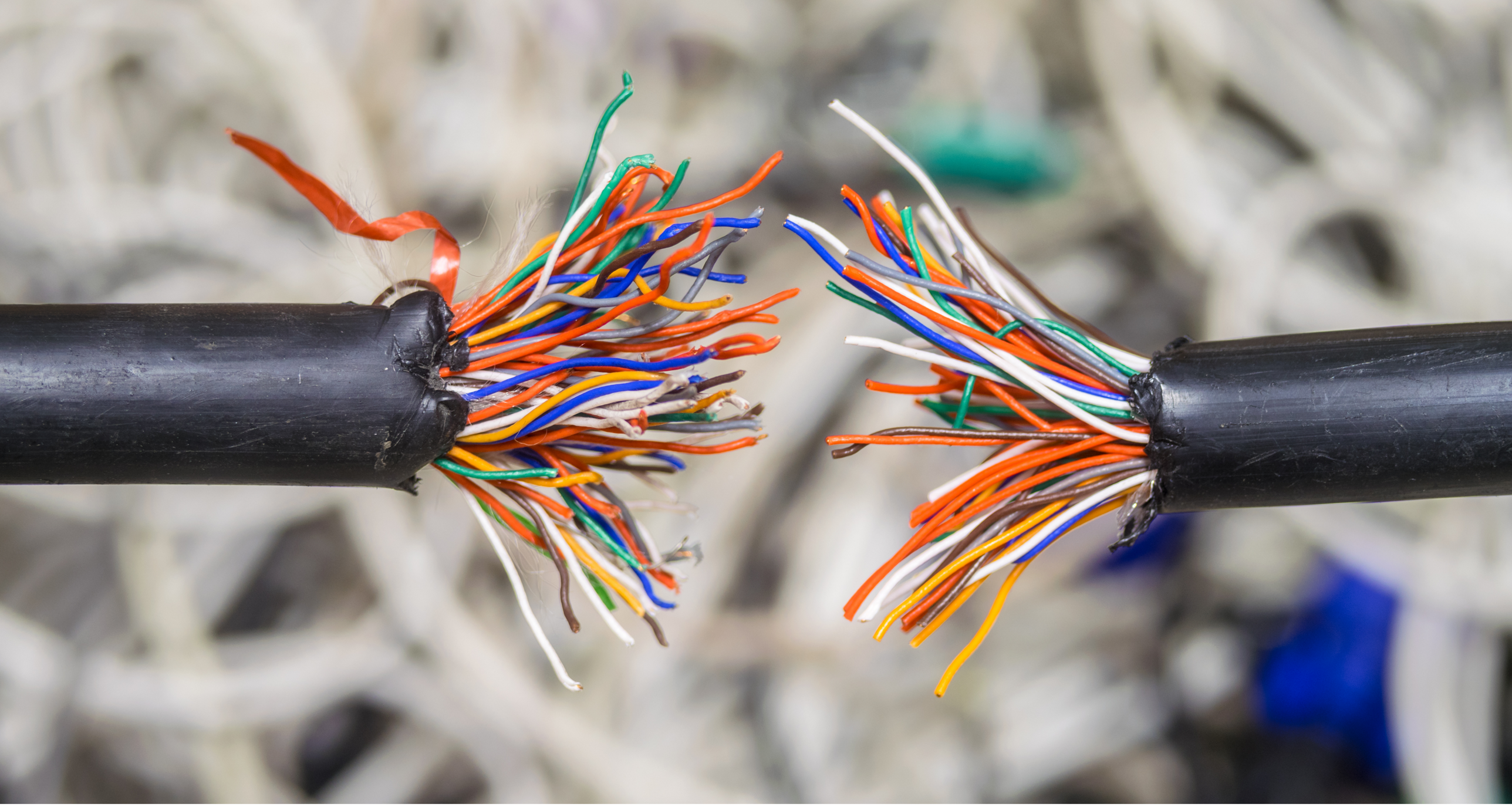


No agreement: Processor p_{if+1} may see a, and all other processors don't see a, but see some b > a

$\Rightarrow$ failure of agreement.

A Lower Bound

- Of course, this proof assumes that a node decides by the minimum of all values it sees.
  But an arbitrary algorithm might work very differently.
- To prove the full **theorem** one needs to show that there always exists a node that outputs a in one case but outputs b in a different case.
- Then, using the above “failure pattern” we can cause that party to output either a or b **without affecting** the other parties! This contradicts the agreement.
- Full details:
  Theorem 5.3 in the book of Attiya and Welch



Consensus

Link Failures and Byzantine Participants

Link Failures

- **Link Failures** -
  - The communication channel stops delivering messages at some point
    - Relevant especially in **asynchronous** settings: we cannot know if the link has failed or the messages is yet-to-be-delivered
- Many tasks are impossible even if one channel might fail.
  - E.g., when the network gets disconnected

The General's Problem

The General's Problem

General **A**

General **B**

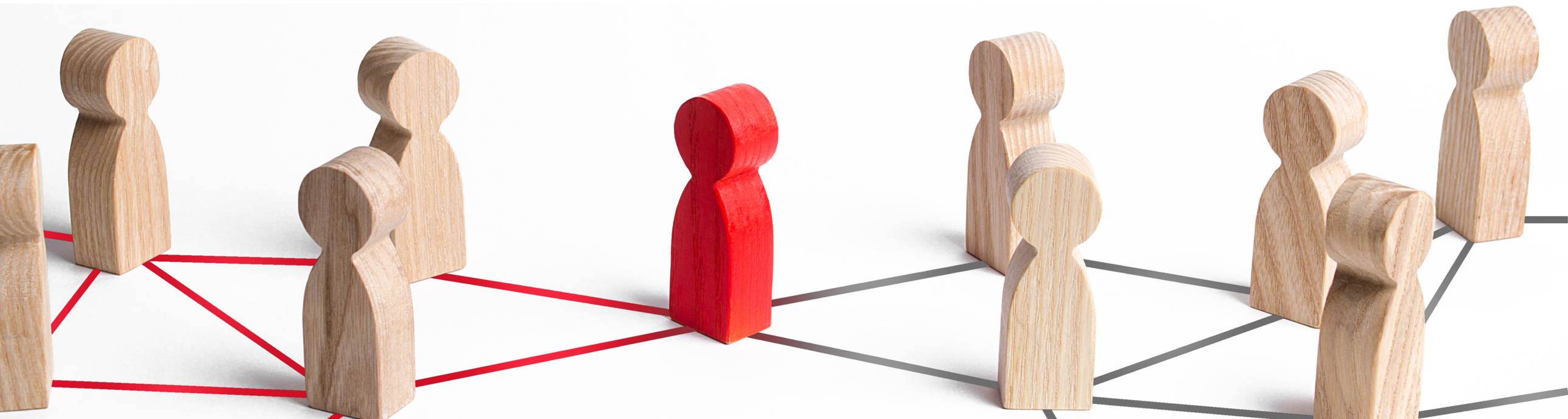
The General's Problem

- This is basically consensus with link failures
- **Impossibility!**
- Intuition:
  - If General **A** sends a message, how would he know if General **B** received it?
  - **B** needs to send an ACK message.
  - But how would **B** know that **A** received his ACK ?
  - **A** needs to Ack the ACK!
  - and so on….
- Formally (2party): Assume the link gets disconnected before any message is sent. If **A**/**B** have different inputs then they cannot reach consensus without sending messages.

Byzantine Agreement

- Byzantine generals wish to reach consensus
- A faulty general might send arbitrary messages. might not send messages at all. might send some messages.

=> stronger noise than Link/Crash failures.



Byzantine Agreement

- **Input**: node i gets a private input x_i .
- **Required Properties:**
  - **termination:** every non-faulty node eventually terminates (gives an output o_i)
  - **agreement:** all non-faulty nodes agree on the same output v, $\quad \forall i \in \text{NonFaulty}, o_i = v$
  - **validity:** the agreed output is a valid input of some **non-faulty** party, $v \in \{x_i \mid i \text{ is non-faulty}\}$.

Byzantine Agreement

- **Attention:**
  A byzantine node can **choose** a new input and play honestly. => No hope to achieve validity.

- all-same **validity:** If all non-faulty nodes get the same value v, the agreed output will be v.

- **Claim**:
  If the inputs are binary, then “all-same validity” implies the “non-faulty-node input” validity.

Byzantine Agreement: Impossibility

- **Theorem**:
  there is no f-resilient byzantine agreement protocol
  when $f \geq n/3$

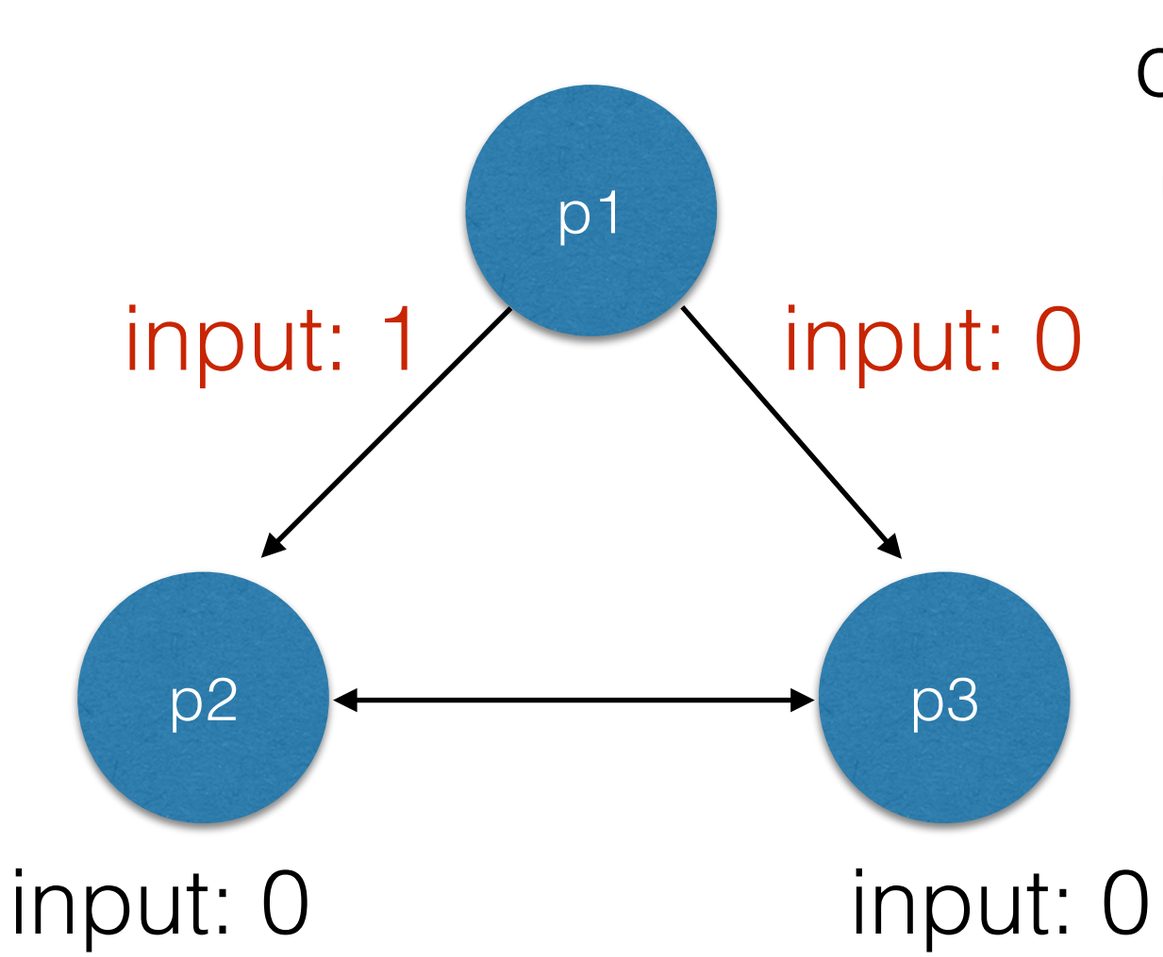
Byzantine Agreement: Impossibility

- **Special case (n=3):** There is no 1-resilient byzantine agreement protocol with n=3 parties.
- **Proof:** suppose, towards contradiction, you are given a 1-resilient algorithm Π for 3-party byzantine agreement.
- we will show 2 different instances of running Π where in each instance the faulty party is different.

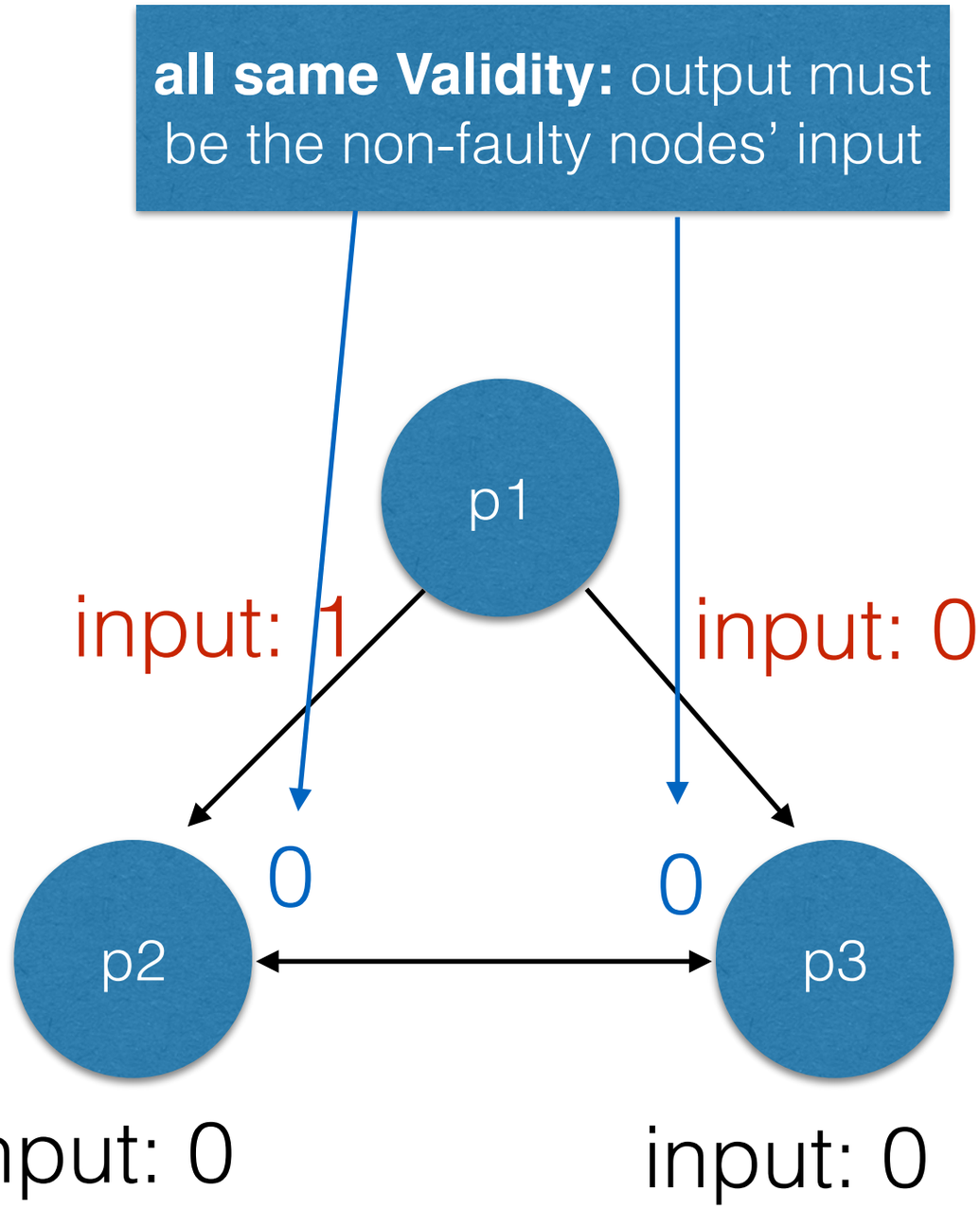
  The 2 instances will tell us how non-faulty party must behave. We then could show a third instance on which Π fails to reach agreement.

Byzantine Agreement: Impossibility

- **Instance 1 (blue):** p1 is faulty

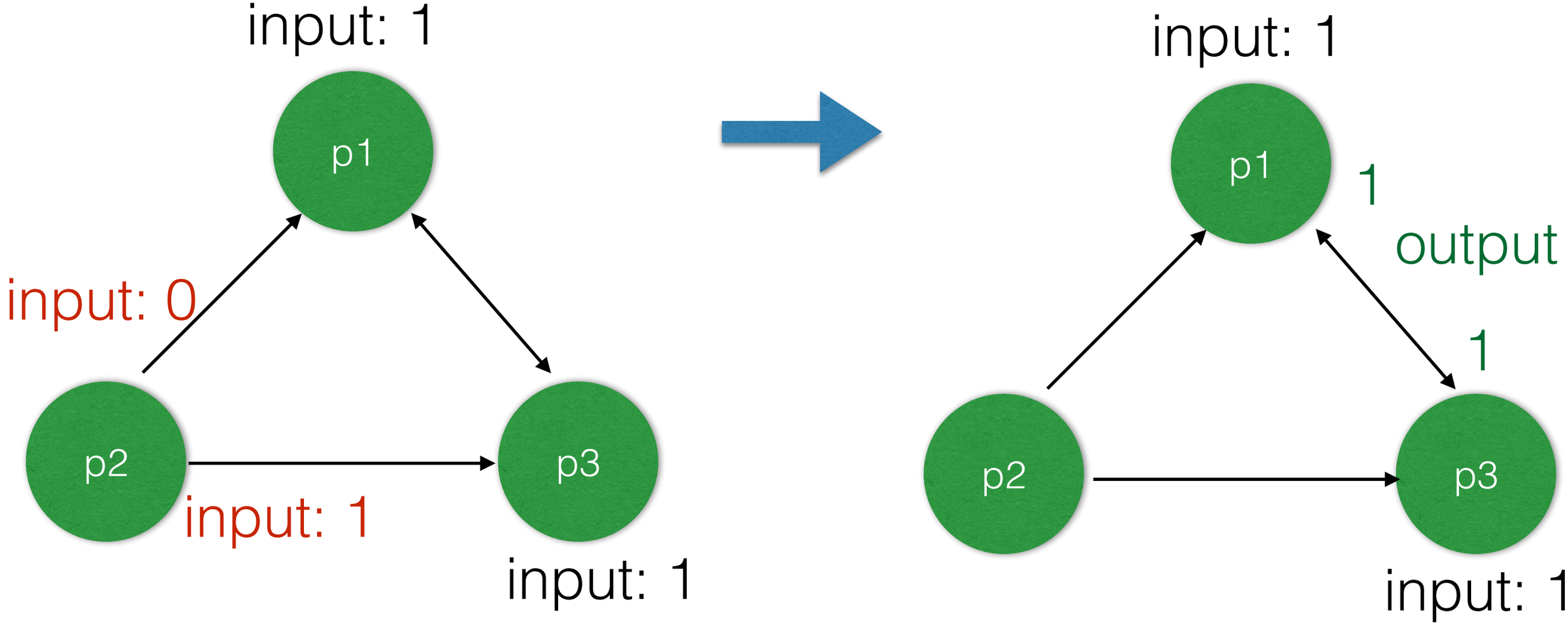


output



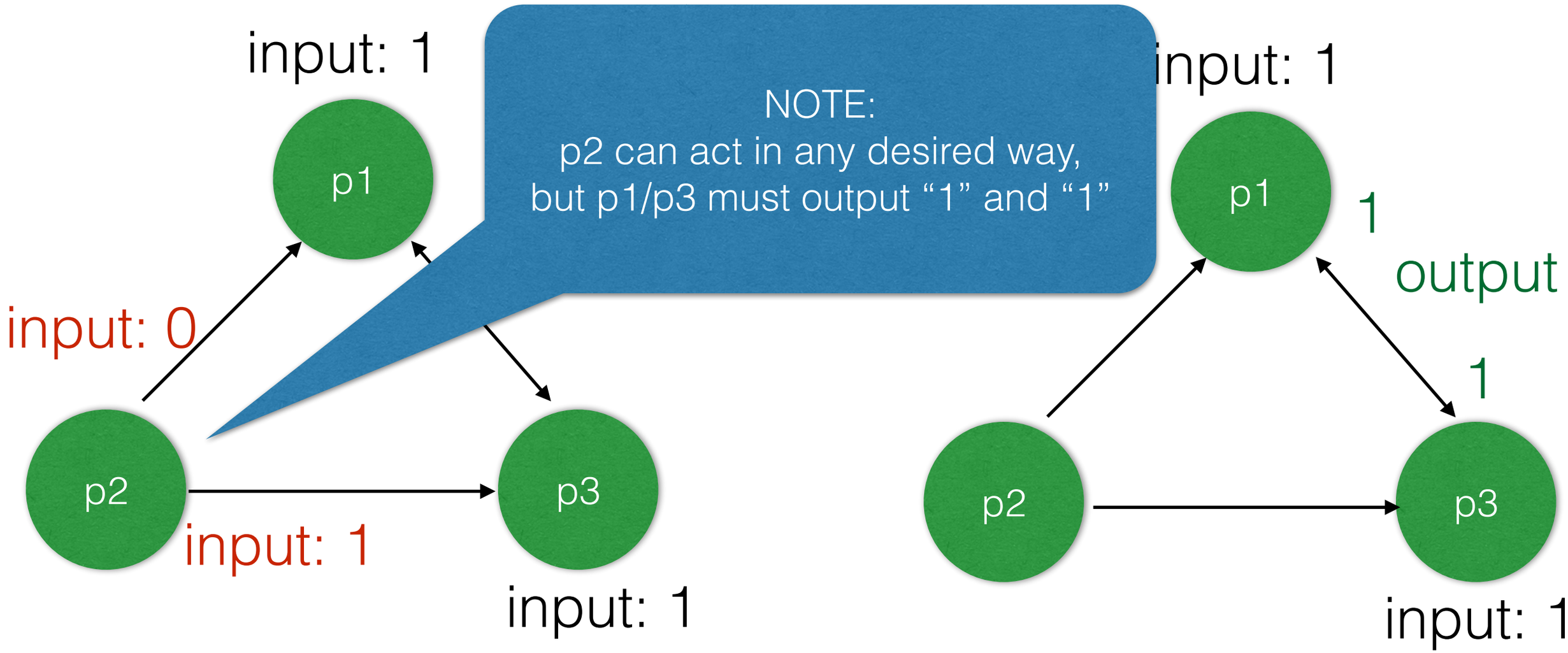
Byzantine Agreement: Impossibility

- **Instance 2 (green):** p2 is faulty

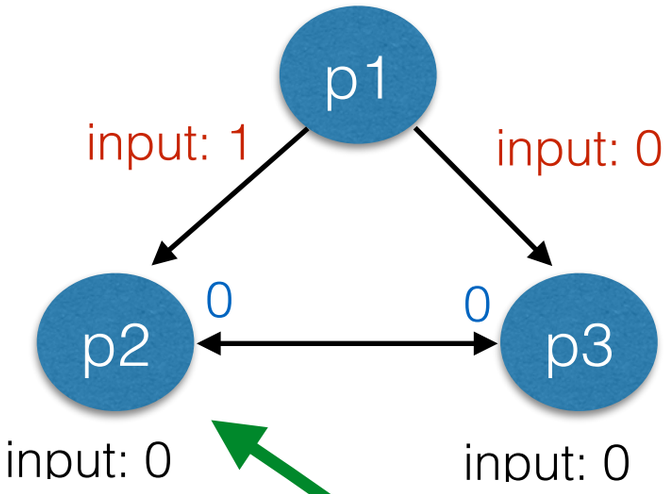


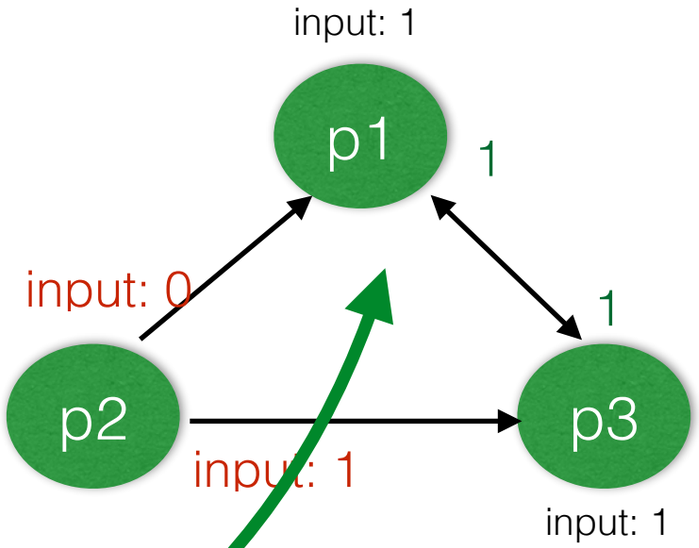
Byzantine Agreement: Impossibility

- **Instance 2 (green):** p2 is faulty



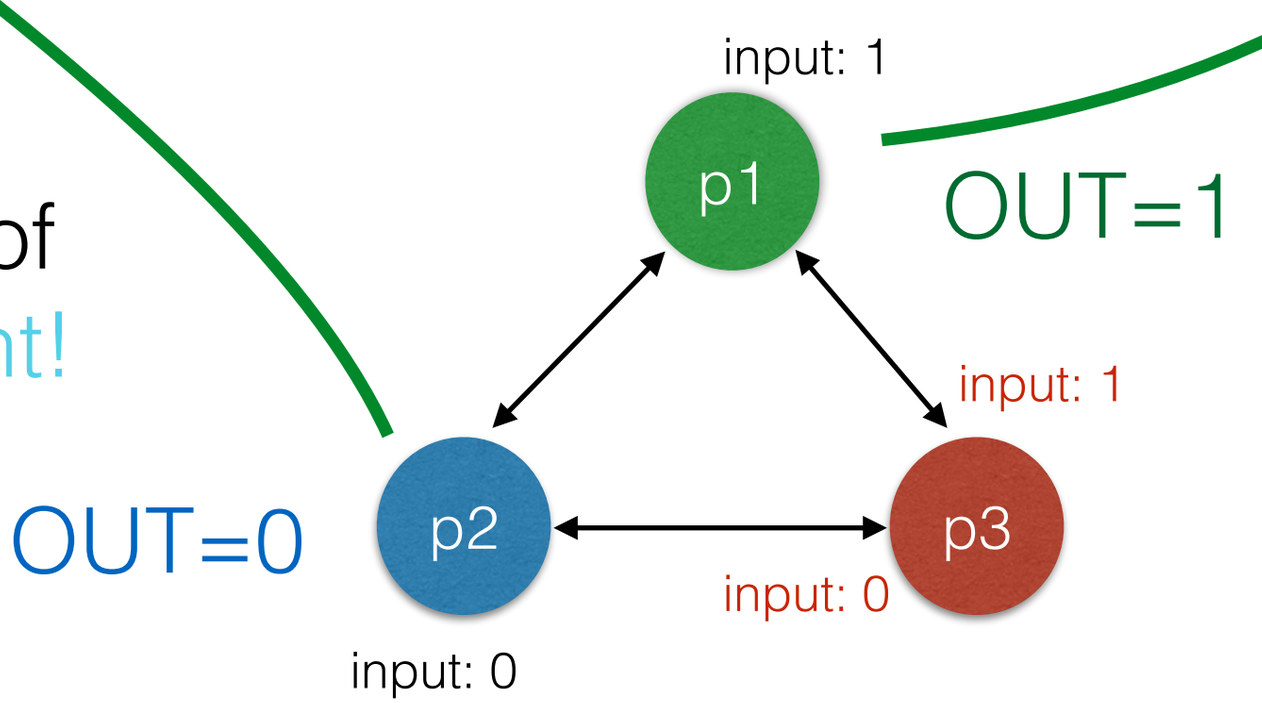
Byzantine Agreement: Impossibility





- **Instance 3 (red):** p3 is faulty

Violation of Agreement!



Byzantine Agreement: Impossibility

- **Theorem**:
  there is no f-resilient byzantine agreement protocol
  when $f \geq n/3$

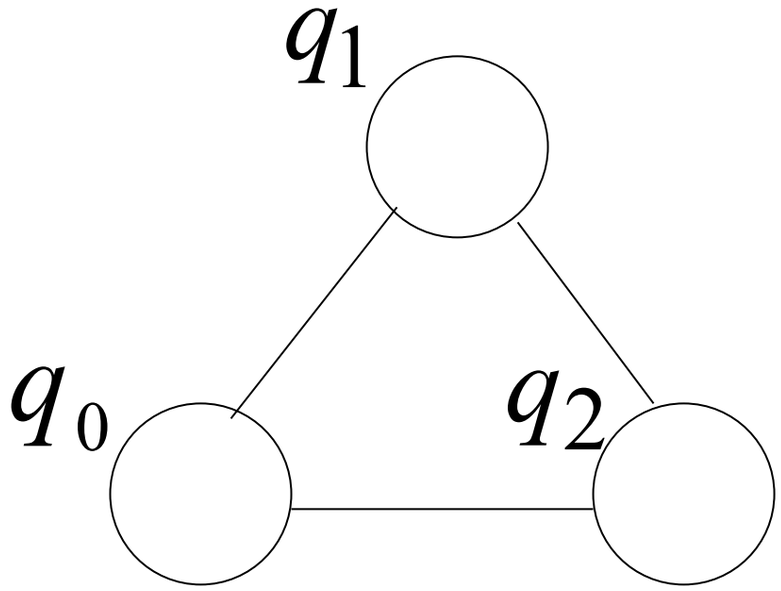
- **proof** (general case):
  suppose, towards contradiction, we have a
  $n/3$-resilient algorithm Π.

  We construct a 1-resilient protocol for 3 parties!

Byzantine Agreement: Impossibility

We construct a 1-resilient protocol for 3 parties from Π

- Assume we have 3 parties, q_0, q_1, q_2.

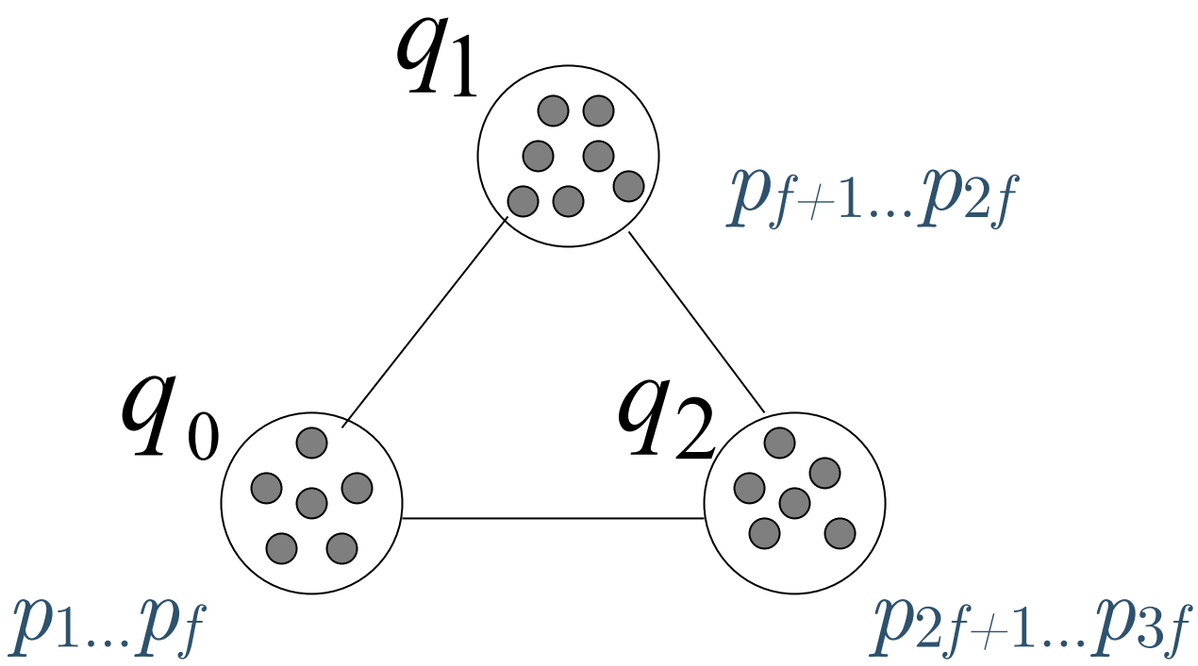


- We will use Π to do so. But Π needs n parties!

Byzantine Agreement: Impossibility

We construct a 1-resilient protocol for 3 parties from Π

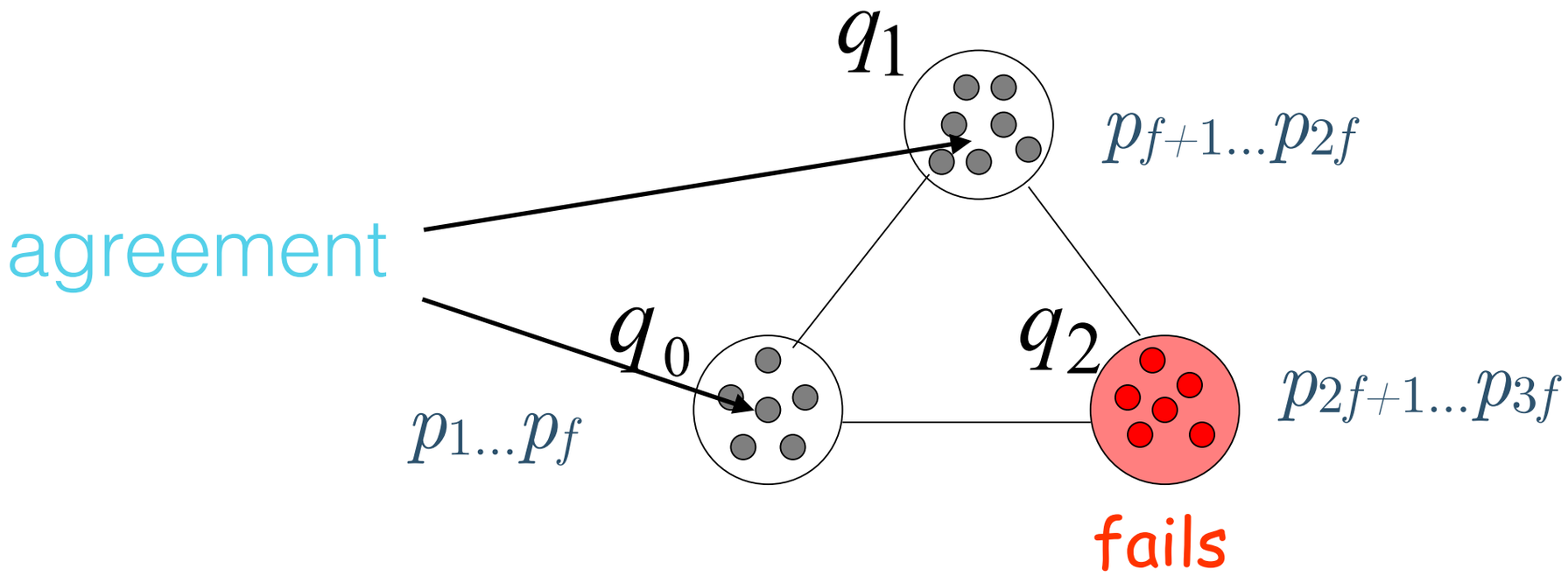
- Each party will **simulate** $f = n/3$ parties **in its head**, and run Π among them



Byzantine Agreement: Impossibility

We construct a 1-resilient protocol for 3 parties from Π

- If q_i is faulty, then in Π at most f parties fail.



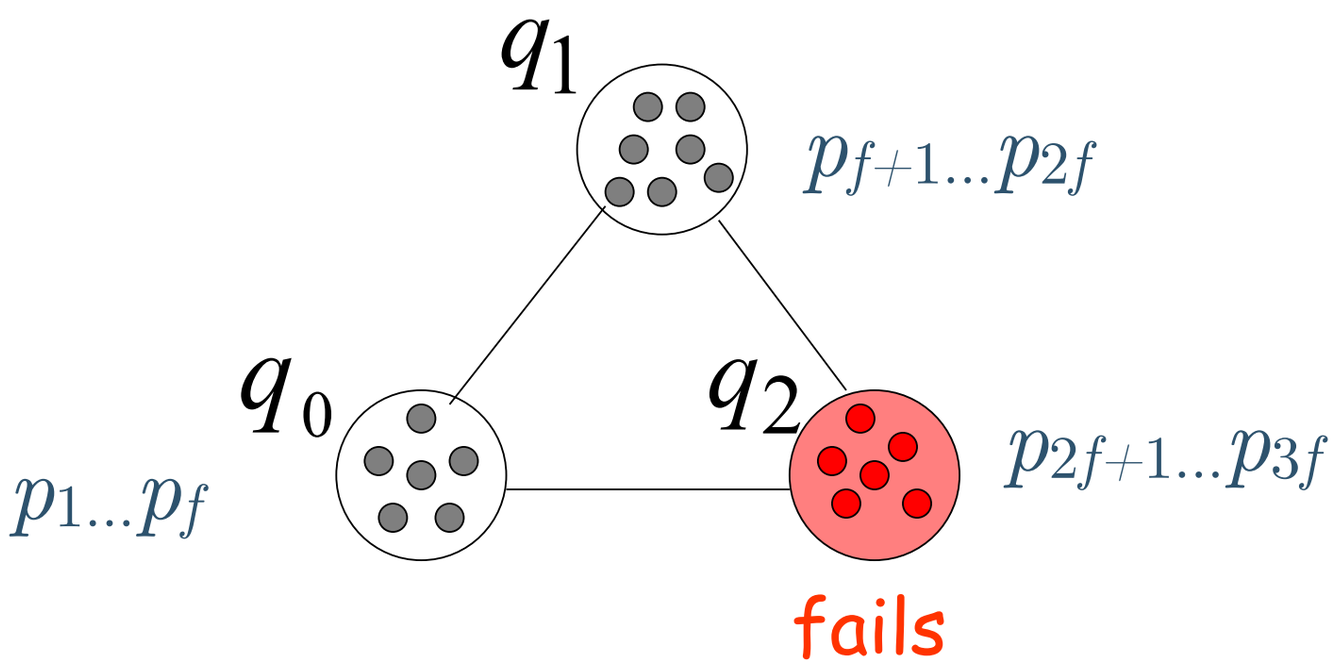
- but Π is f-resilient! ➔ q_0 and q_1 will agree!

Byzantine Agreement: Impossibility

We construct a 1-resilient protocol for 3 parties from Π

- but this is a contradiction, since 1-resilient consensus for 3 parties does not exists. Thus, Π does not exists.

QED



Agreement Algorithms

- We saw that $f \geq n/3$ is impossible.
- What can be done?

$n = 2f + 1$
in synch. settings with
signatures

optimal resilience: $n = 3f + 1$

Scheme	type	rounds	Comm.
Pease, Shostak and Lamport (1980)	synch. det.	f+1	$\exp(n)$
Garay and Moses (1993)	synch. det.	f+1	$\text{poly}(n)$
King and Saia (2011)	synch. rand.	$\text{polylog}(n)$	$O(\sqrt{n})$
Micali (2018)	synch. rand. crypt.	$O(1)$	$O(\text{n})$

(NOT AN EXHAUSTIVE LIST!)

Randomized BA

- [Micali, 2018]
- randomized (cryptographic) **binary** Byzantine Agreement protocol for $f < n/3$
- Synchronized. O(1) rounds **in expectation**.
- Implementation requires cryptographic tools: unique signatures, hash, shared randomness.

Randomized BA

- **Notations**

  $\#_v$(b) = number of messages party v received with the bit ‘b’

- Simplified Version (ignoring crypto-implementation of the bit c):

Input: bit $b_v \in \{0,1\}$

1 **Repeat (for party v):**
2 broadcasts b_v to all and receive messages from all
3 all parties obtain a (fresh) random independent bit c // same bit for all
4 **if** $\#_v(0) > 2n/3$, set b_v=0
5 **if** $\#_v(1) > 2n/3$, set b_v=1
6 **otherwise**: set b_v= c

Randomized BA

- **Claim 1:** if all honest parties agree on some bit **b** at the beginning of a phase, they all agree on it at the end of the phase.
- **Proof**:
  - There are more than $2n/3$ honest parties ($f<n/3$)
  - If all honest parties start with b_v=**b**, then each honest party receives at least $2n/3$ copies of **b**.
  - In line **4** (or **5**) all honest parties set b_v=**b**.

Randomized BA

- **Claim 2:**
  if honest parties **do not** agree at the beginning of a phase, they end in agreement (on some **b**) with probability 1/2 over the choice of c.

- **Proof**: Let's split into cases:
  - all honest perform line **4** or all perform **5**, or all perform **6**
  - some perform **4** and some **6**
  - some perform **5** and some **6**

```
1  Repeat (for party v):
2      broadcasts b_v to all and receive messages from all
3      all parties obtain a (fresh) random independent bit c
4      if #_v(0) > 2n/3, set b_v=0
5      if #_v(1) > 2n/3, set b_v=1
6      otherwise: set b_v= c
```

Randomized BA

```
1  Repeat  (for party v):
2        broadcasts b_v to all and receive messages from all
3        all parties obtain a (fresh) random independent bit c
4        if #_v(0) > 2n/3, set b_v=0
5        if #_v(1) > 2n/3, set b_v=1
6        otherwise: set   b_v= c
```

- **Proof** (cont'):
  - If all perform **4** (or **5**, or **6**): clearly agree on the same **b**
- **Note**: it is impossible that some honest perform **4** while other honest perform **5**!
- **Case**: some honest perform **4**, and others **6.**
  - with probability 1/2, c=0; if so, all honest agree on 0. Otherwise - no agreement.

Randomized BA

- **Claim 1:** if all honest parties agree on some bit **b** at the <u>beginning</u> of a phase, they will all agree on **b** at the <u>end</u> of that phase.

- **Claim 2:**
  if honest parties **do not** agree at the beginning of a phase, they end in agreement (on some **b**) with probability 1/2 over the choice of c.

Randomized BA

- **validity**: follows from by **Claim1**
- **agreement**: (with high probability)
  follows from **Claim2** and **Claim1**

  ➡ After t rounds, $\Pr[\text{not all agree}] < 2^{-t}$

Expected running time: $\sum_{t=1}^{\infty} t2^{-t} = 2 = O(1)$

- **termination:**
  - Parties do not know when the network reaches agreement!
  - A possible solution: run exactly 100 rounds. Pr[fail] < 2^{-100}
  - Another option: if a party is “sure”, it terminates, sending its finally decision to all other nodes.

Randomized BA with Termination

Input: bit $b_v \in \{0,1\}$

Repeat until termination:

1 (Part I: use c=0.) broadcasts b_v to all

2 **if** $\#_v(0) > 2n/3$, set b_v=0 . Send 0* to all. Terminate with output 0.

3 **if** $\#_v(1) > 2n/3$, set b_v=1

4 **otherwise**: set b_v= **0**

5 (Part II: use c=1.) broadcasts b_v to all

6 **if** $\#_v(1) > 2n/3$, set b_v=1 . Send 1* to all. Terminate with output 1.

7 **if** $\#_v(0) > 2n/3$, set b_v=0

8 **otherwise**: set b_v= **1**

9 (Part III: use random c.) broadcasts b_v to all. **Assume** c is a fresh indep. random bit known to all

10 **if** $\#_v(0) > 2n/3$, set b_v=0

11 **if** $\#_v(1) > 2n/3$, set b_v=1

12 **otherwise**: set b_v= **c**

0* means:
assume my messages are 0
from now and on

Example

Randomized BA with Termination

- **Claim 1':**
  if all honest parties agree on some bit **b** at the <u>beginning</u> of a part, they all agree on **b** it at the <u>end</u> of the part.

- **Claim 2':**
  if honest parties **do not** agree at the beginning of <u>part III</u>, they end in agreement (on some **b**) with probability 1/2 over the choice of c.

Randomized BA with Termination

1 (Part I: use c=0.) broadcasts $\mathbf{b_v}$ to all
2 **if** $\#_v(0) > 2n/3$, set $\mathbf{b_v}$=0 . Send 0* to all. Terminate with 0.
3 **if** $\#_v(1) > 2n/3$, set $\mathbf{b_v}$=1
4 **otherwise**: set $\mathbf{b_v}$= **0**

- **Claim 3':**
  if some honest party terminates with output b, agreement holds at the end of that part.
- **proof:**
  - wlog, assume **part I**. If v terminates (line **2**), then at least $n/3$ honest parties hold 0. Then, step **3** can never happen to any of them.
    In step **4**, all honest u's set $\mathbf{b}_u$**=0.** agreement.

Randomized BA with Termination

- **Claim 4':**
  All honest parties halt with probability 1
- **proof:**
  - From **claim1** and **claim2**, Pr[Agreement]→1. Let's assume agreement on 0 has been reached (with **claim1**, it remains forever)
  - Consider the next time we are at **part I**
    all honest parties send 0 (including those who terminated, who sent 0*)
  - Thus any honest party that is still "alive" will receive $> 2n/3$ copies of 0 and will terminate in line **2**

Randomized BA with Termination

- **validity**: follows from by **Claim1**
- **agreement**: (with high probability) follows from **Claim2** and **Claim1.**
- **termination**:
  - All honest parties terminate one phase (< 3 rounds) after reaching agreement
  - ***Expected*** running time:

$$1 + \sum_{t=1}^{\infty} t2^{-t} = 3$$ phases, i.e., 9 rounds in expectation.