Today

- The *consensus* problem in synchronous systems
  - Crash.
  - Byzantine.
  - Some lower bounds.
  - One value of cryptography.

Consensus

Reliable broadcast: There is a transmitter p_1 and a set of receivers p_2, p_3, ... p_n. p_1 has an *initial value* v and each p_i computes a *decision value* d_i.

RB1: If p_1 is nonfaulty, then for all nonfaulty p_i: $d_i = v$.

RB2: If p_1 is faulty, then for all nonfaulty p_i and p_j: $d_i = d_j$.

Uniformity: For any two p_i and p_j that decide: $d_i = d_j$.

Other versions of Consensus

Each process has an *input bit* x_i and an *output bit* y_i.

The protocol terminates with each nonfaulty processor agreeing on y_i: $\exists y\ \forall\ i$: p_i not faulty: $y_i = y$ (RB2)

strong: If $\forall\ i$: $x_i = x$ then $y = x$

weak: If $\forall\ i$: $x_i = x$ then $y = x$ provided there are no failures in the run.

very weak: For $ys \in \{0, 1\}$ $\exists$ $\{x_1, x_2, ..., x_n\}$ and a run in which $y = ys$.

System model

We assume that:

- Clocks are approximately synchronized.
- Messages have an upper bound on time from sending to delivery.

These assumptions allow us to build the protocols in a *round-based* approach.

- In a round each process can receive messages sent to it in the previous round and subsequently send messages.

A protocol for crash failures

p_1 round 0:

  p_1 sends v to all

  $d_1 = v$

  halt

half round

p_i round a: $1 \le a \le t$:

  receive messages sent in round $a - 1$

  if received a message, let x be the value in some message

    $d_i = x$

    send x to all

    halt

round

p_i round $t + 1$:

  receive message sent to it in round t

  if received a message, let x be the value in some message

    $d_i = x$

  else $d_i = 0$

  halt

half round

Informal proof of crash failure protocol

RB1 holds:

- If p_1 is not faulty, then in round 0 it sends v to all processes.
- All processes that have not crashed deliver v in round 1, set d_i to v, and forward v to all.

... hence, all processes that have not crashed by the end of round 1 set d_i to v.

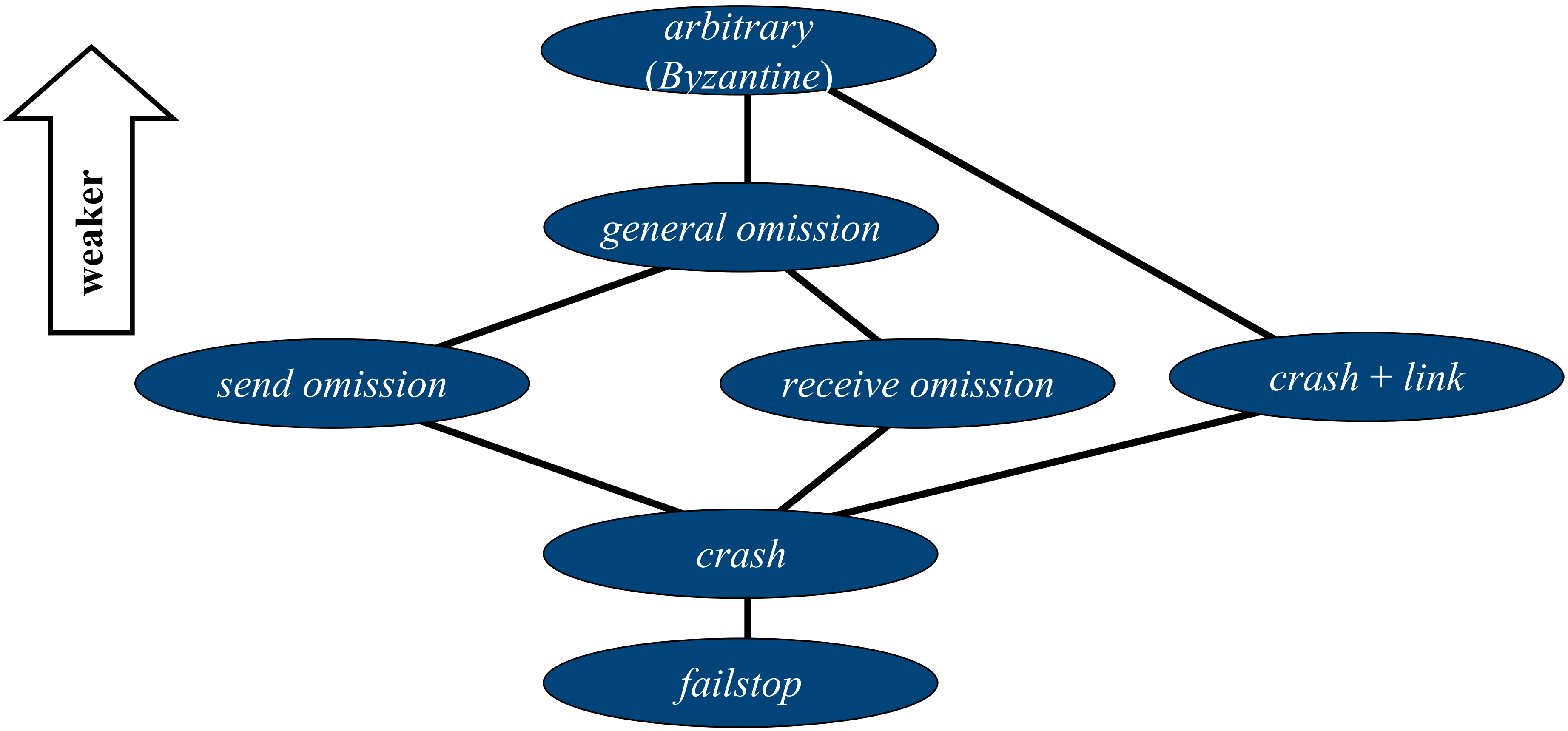
Informal proof of crash failure protocol (continued)

RB2 holds as long as $n > t$:

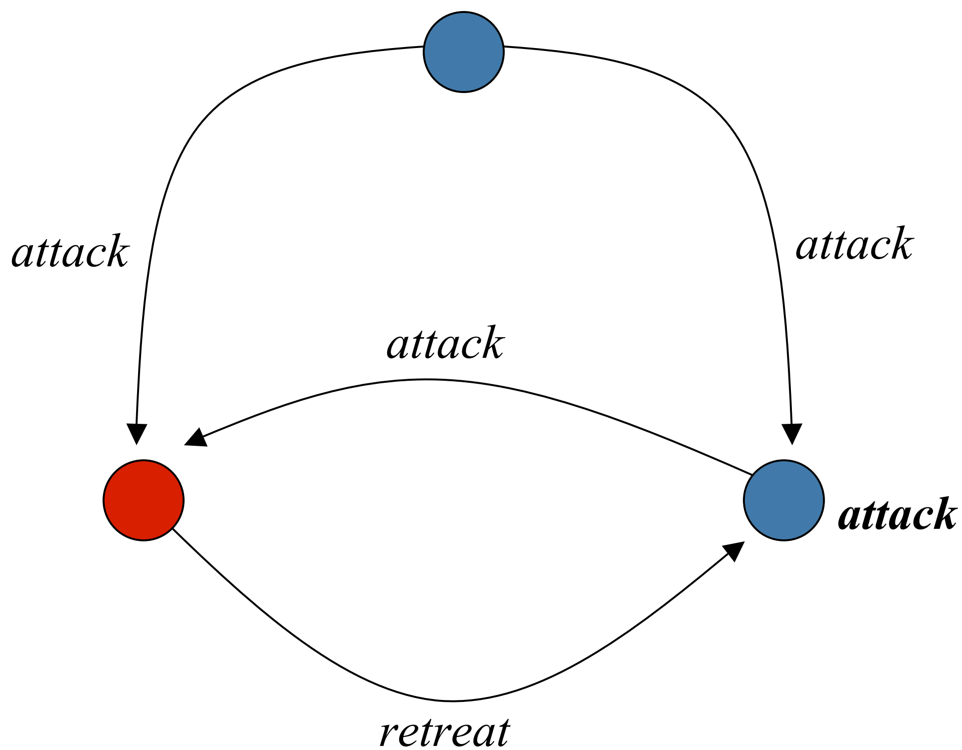
- A value received in round a has been sent by a chain of a processes.
- If any nonfaulty process received a value x in round $a \le t$, then all processes that have not yet crashed by the end of round $a + 1$ will receive the value x.
- If a process receives a value x sent in round $t + 1$, then at least one process in the chain is nonfaulty.

… is this uniform?

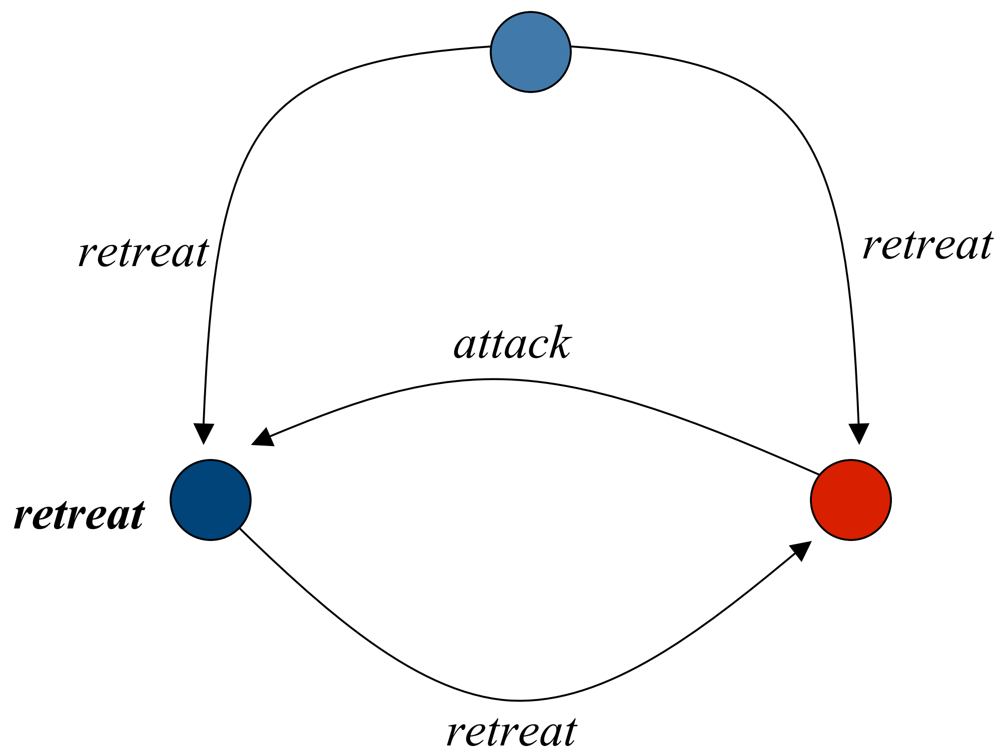
What about other failures?



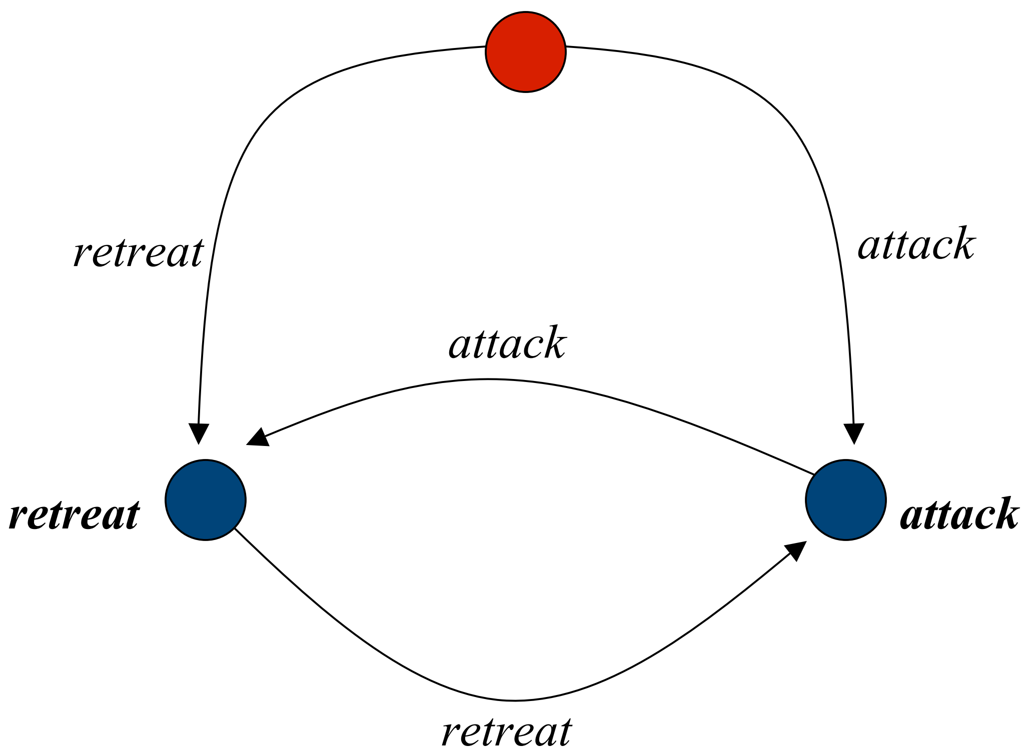
Arbitrary, $n = 3$, $t = 0$



Arbitrary, $n = 3$, $t = 0$

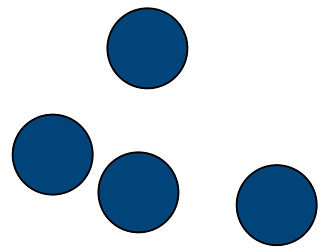


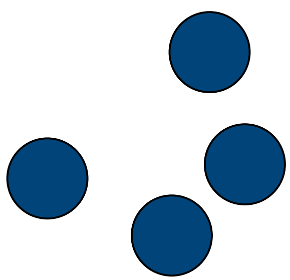
Arbitrary, $n = 3$, $t = 0$

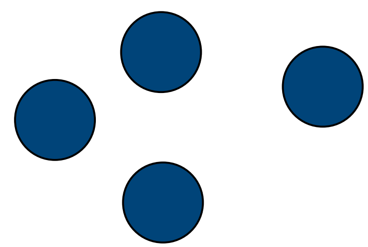


Arbitrary, $n > 3t$

Suppose a solution exists for $n \leq 3t$ (e.g., $t = 4$, $n = 12$)

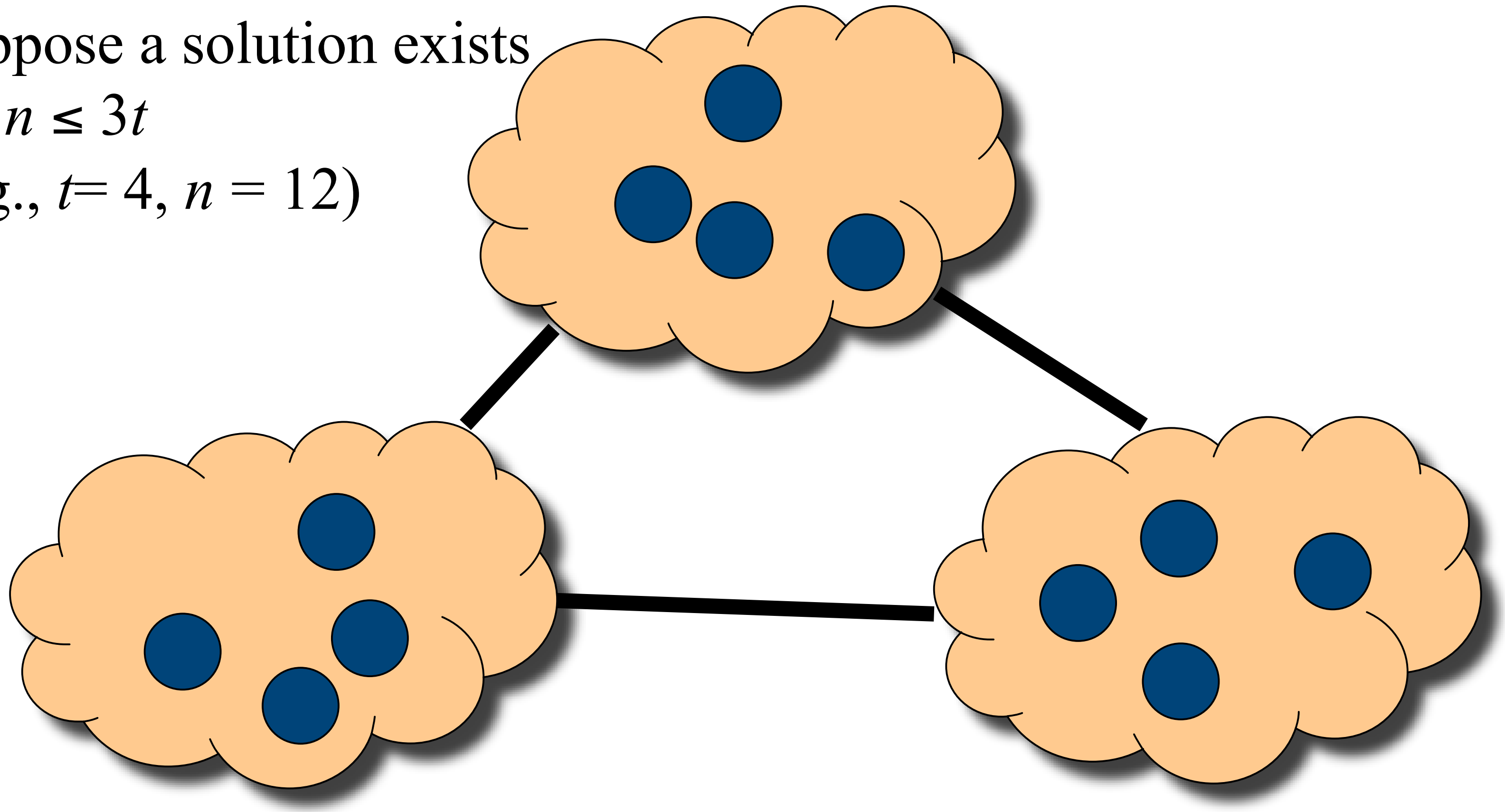






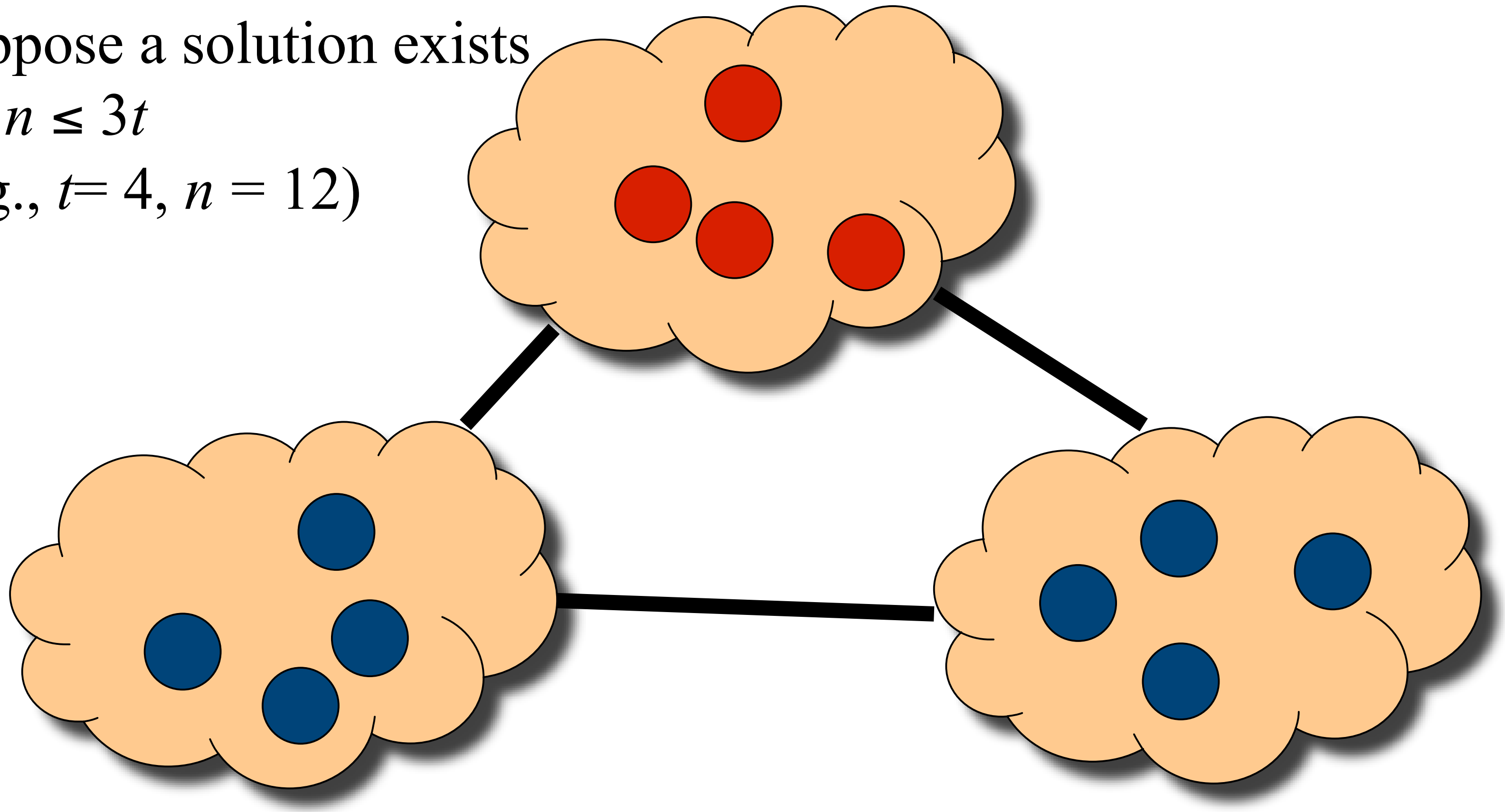
$n > 3t$

Suppose a solution exists for $n \le 3t$
(e.g., $t = 4$, $n = 12$)



$n > 3t$

Suppose a solution exists for $n \leq 3t$
(e.g., $t = 4$, $n = 12$)



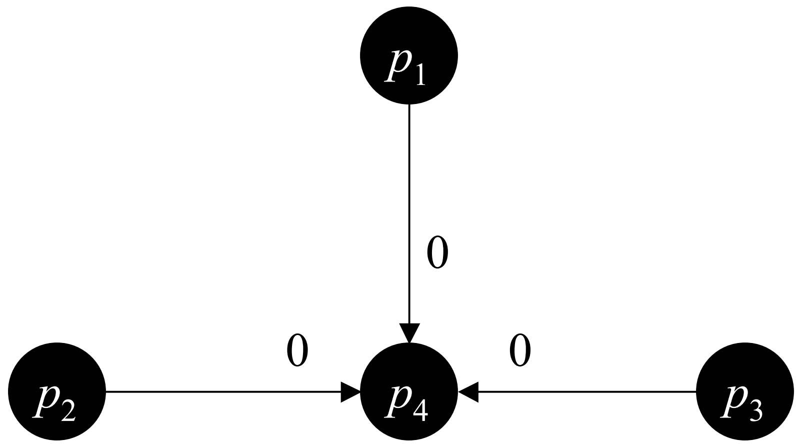
Arbitrary, $n = 4$, $t = 1$

Not receiving a value is equivalent to receiving 0.

- Round 0: p_1 sends v to all
- Round 1: p_i receives value sent in round 0 and forwards to all.
- Round 2: p_i receives value sent in round 0.

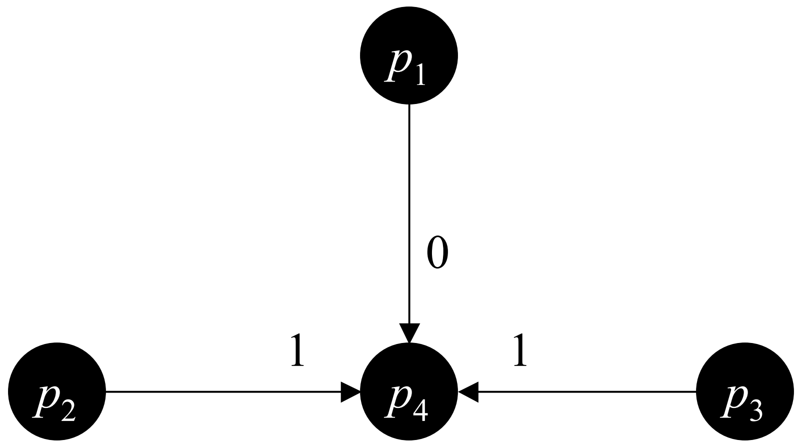
Each process has received three values. Set d_i to the majority value.

Arbitrary, $n = 4$, $t = 1$



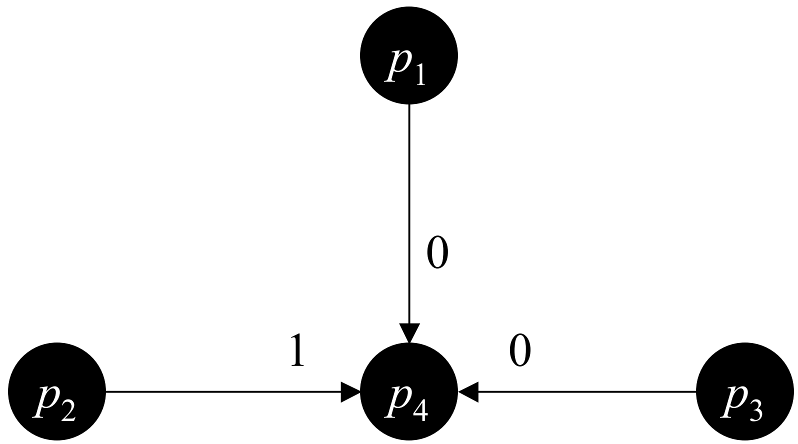
p_1 can't be faulty, so decide 0

Arbitrary, $n = 4$, $t = 1$



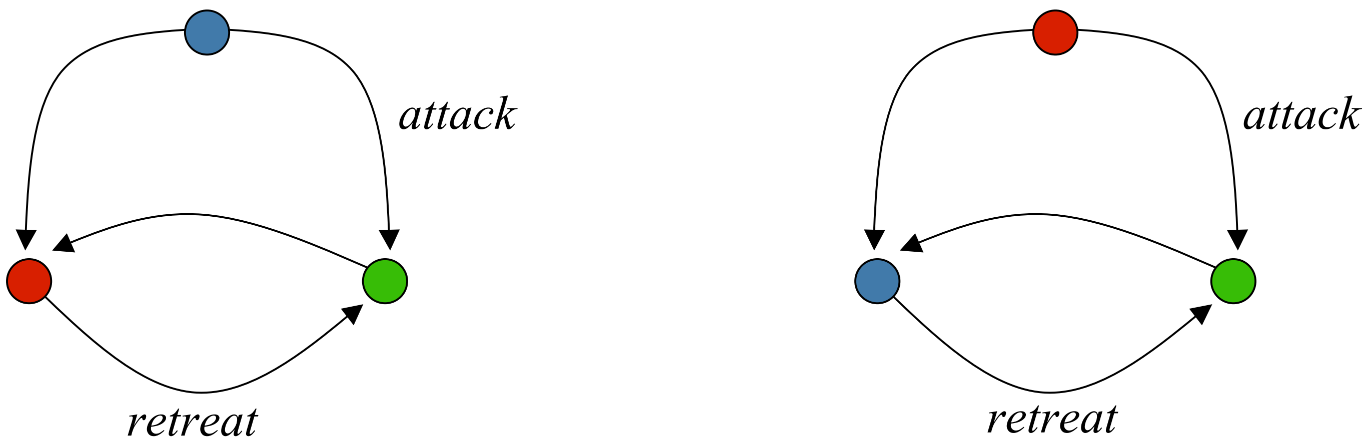
p_1 is faulty, so p_2, p_3, p_4 are not faulty.
All decide 1.

Arbitrary, $n = 4$, $t = 1$



If p_1 is faulty, then p_2, p_3, p_4 are not faulty. Each gets {0, 0, 1} and decides 0.
If p_1 is not faulty, then p_2 is faulty. Each gets {0, 0, x} and decides 0.

Indistinguishability



can distinguish between these two cases using *digital signatures*.

Digital signatures

With public key cryptosystems, there are two keys: a publicly known key K_E and a privately known key K_D.

- $[M]K_E$ is a message that can be encrypted by anyone and decrypted only by one.
- $[M]K_D$ is a message that can be encrypted only by one and decrypted by anyone.

A protocol for arbitrary failures

$vals_i = \{\ \}$

p_1 round 0:

  p_1 signs v and sends to all

p_i round a: $1 \le a \le t$:

  receive messages sent in round $a - 1$

  while (there is a received *valid* message m that contains $x \notin val_i$)

    $val_i = val_i \cup \{ x \}$

    sign and forward m to all

m is *valid* if, when received in round a, m has a distinct signatures, the first of which is by p_1.

p_i round $t + 1$:

  receive message sent to it in round t

  while (there is a received *valid* message m that contains $x \notin val_i$) $val_i = val_i \cup \{ x \}$

  if $(|val_i| \neq 1)$ $d_i = 0$

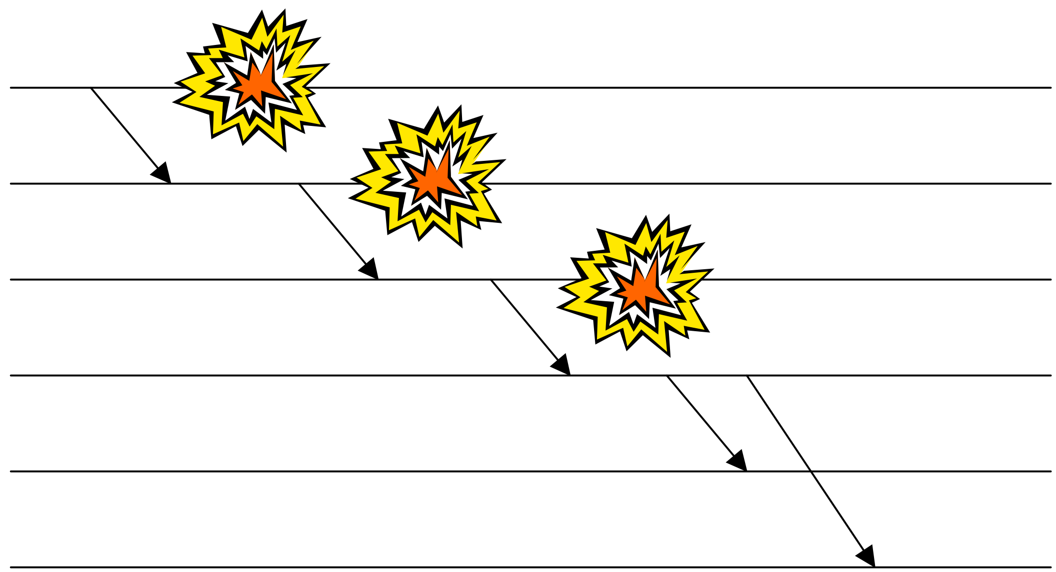
  else d_i = value in val_i

Informal Proof

Much like the previous protocol.

Bounds on number of rounds

For any failure model, there is a run of $t + 1$ rounds.



Early stopping consensus, crash failures

An *early stopping* consensus protocol has *some* processes decide in $f + 2$ rounds where $f \leq t$ is the actual number of failures that occur in the run.

The idea is to have each process send a message in each run as a kind of *heartbeat*. When failures stop for long enough, then consensus is reached.

Early-stopping, crash failures

Each process p_i maintains UP_i which is initially all of the processes.

If in round a, p_i expects to receive a message (sent in round $a - 1$) by p_j but does not receive one, then p_i removes p_j from UP_i.

Each process sends a message in each (but the first) round. The message will be:

- A value, 0 or 1 if it has received that value previously.
- $\perp$ default value.
- $\varnothing$ heartbeat, sent when didn't receive a value in round 1, or in later rounds the process can't decide.

Early-stopping, crash failures

round 0: p_1 sends v to all

round $a \leq t$, each process p_i that has not yet halted:

  receive messages sent in round $a - 1$

  if received $x \in \{0, 1, \perp\}$ { $d_i = x;$ send x to all; **halt** }

  else if ($a > 1$)

    if (have received $\varnothing$ from all $p_j \in UP_i$) { $d_i = \perp$; send $\perp$ to all; **halt** }

    else $UP_i = UP_i \setminus \{p_j : p_j \in UP_i$ and p_i did not receive a message from $p_j\}$

  send $\varnothing$ to all

round $t + 1$, each process p_i that has not yet halted:

  receive messages sent in round t

  if received $x \in \{0, 1, \perp\}$ $d_i = x$

  else $d_i = \perp$