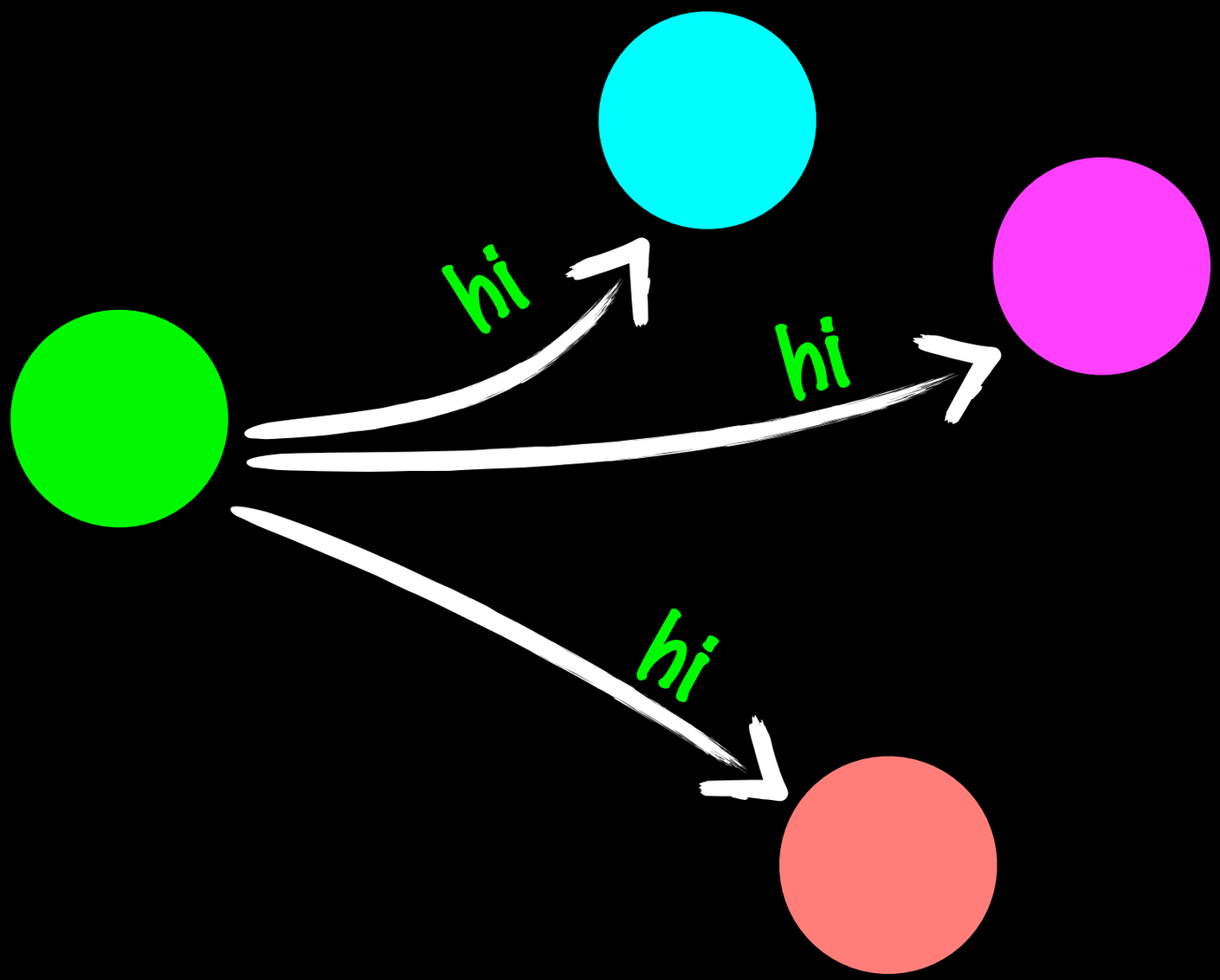
more

Broadcast Algorithms

Gustav Cedersjö


hi
hi
hi

Outline

Part 1 Delivery order

Part 2 Byzantine processes

Module

Reliable Broadcast

Validity: If a correct process *p* broadcasts a message *m*, then *p* eventually delivers *m*.

No duplication: No message is delivered more than once.

No creation: If a process delivers a message *m* with sender *s*, then *m* was previously broadcast by process *s*.

Agreement: If a message *m* is delivered by some correct process, then *m* is eventually delivered by every correct process.

old stuff...

Part 1

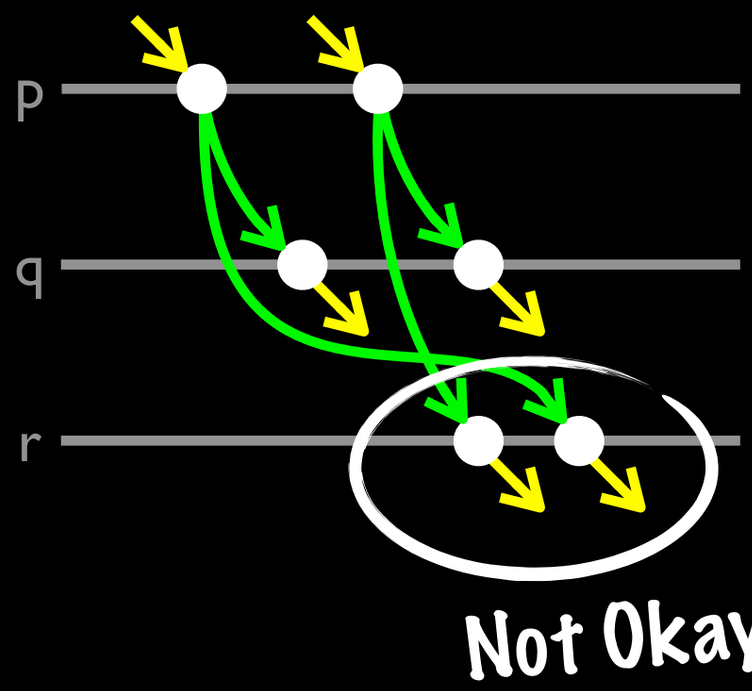
Delivery Order

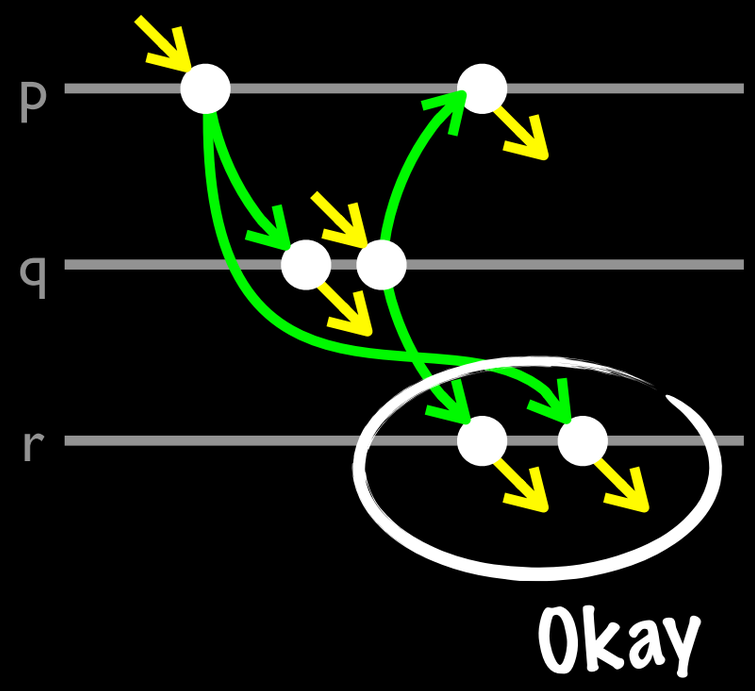
Module

FIFO Reliable Broadcast

Validity, no duplication, no creation, and agreement: Reliable Broadcast

FIFO delivery: If some process broadcasts message m_1 before it broadcasts m_2, then no correct process delivers m_2 unless it has already delivered m_1.





Fail-Silent Algorithm

Broadcast with Sequence Number

- Use a Reliable Broadcast module
- Add sequence numbers to each message
- Keep track of next expected sequence number for each process
- Hold delivery until previous messages are delivered

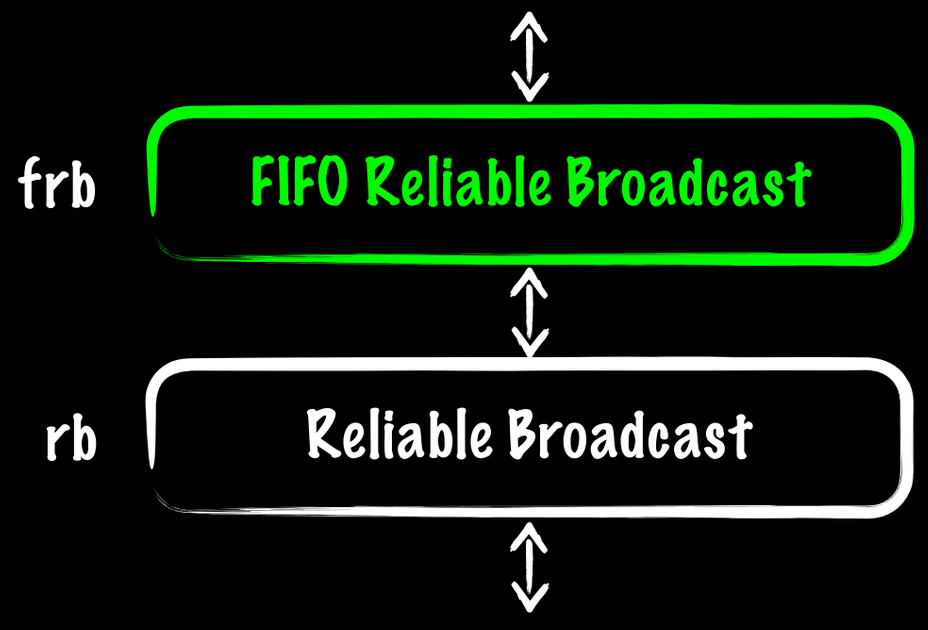
Fail-Silent Algorithm

Broadcast with Sequence Number

upon event ⟨ *frb*, *Init* ⟩ **do**
    lsn := 0;
    pending := ∅;
    next := [1]^N;

upon event ⟨ *frb*, *Broadcast* | *m* ⟩ **do**
    lsn := *lsn* + 1;
    trigger ⟨ *rb*, *Broadcast* | [DATA, *self*, *m*, *lsn*] ⟩;

upon event ⟨ *rb*, *Deliver* | *p*, [DATA, *s*, *m*, *sn*] ⟩ **do**
    pending := *pending* ∪ {(*s*, *m*, *sn*)};
    while exists (*s*, *m*′, *sn*′) ∈ *pending* such that *sn*′ = *next*[*s*] **do**
        next[*s*] := *next*[*s*] + 1;
        pending := *pending* \ {(*s*, *m*′, *sn*′)};
        trigger ⟨ *frb*, *Deliver* | *s*, *m*′ ⟩;



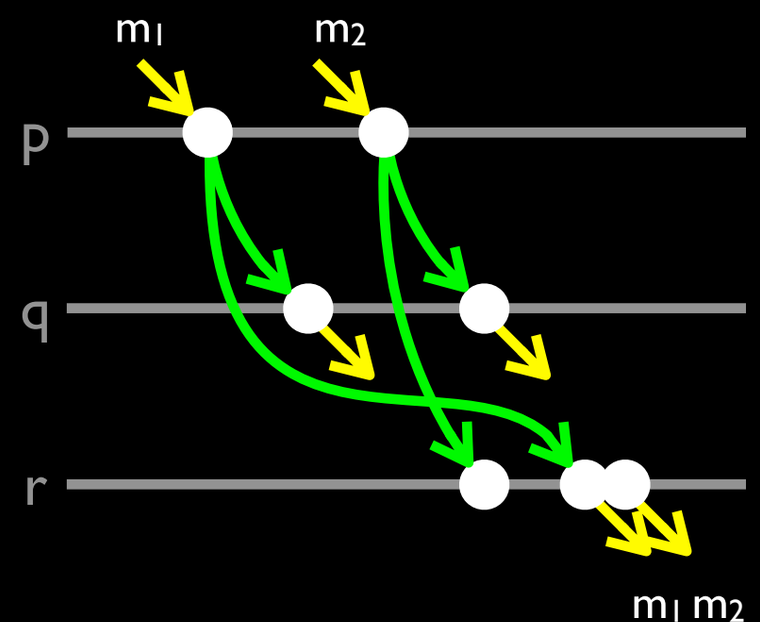
Fail-Silent Algorithm

Broadcast with Sequence Number

upon event ⟨ *frb*, *Init* ⟩ **do**
    lsn := 0;
    pending := ∅;
    next := [1]^N;

upon event ⟨ *frb*, *Broadcast* | *m* ⟩ **do**
    lsn := *lsn* + 1;
    trigger ⟨ *rb*, *Broadcast* | [DATA, *self*, *m*, *lsn*] ⟩;

upon event ⟨ *rb*, *Deliver* | *p*, [DATA, *s*, *m*, *sn*] ⟩ **do**
    pending := *pending* ∪ {(*s*, *m*, *sn*)};
    while exists (*s*, *m*′, *sn*′) ∈ *pending* such that *sn*′ = *next*[*s*] **do**
        next[*s*] := *next*[*s*] + 1;
        pending := *pending* \ {(*s*, *m*′, *sn*′)};
        trigger ⟨ *frb*, *Deliver* | *s*, *m*′ ⟩;

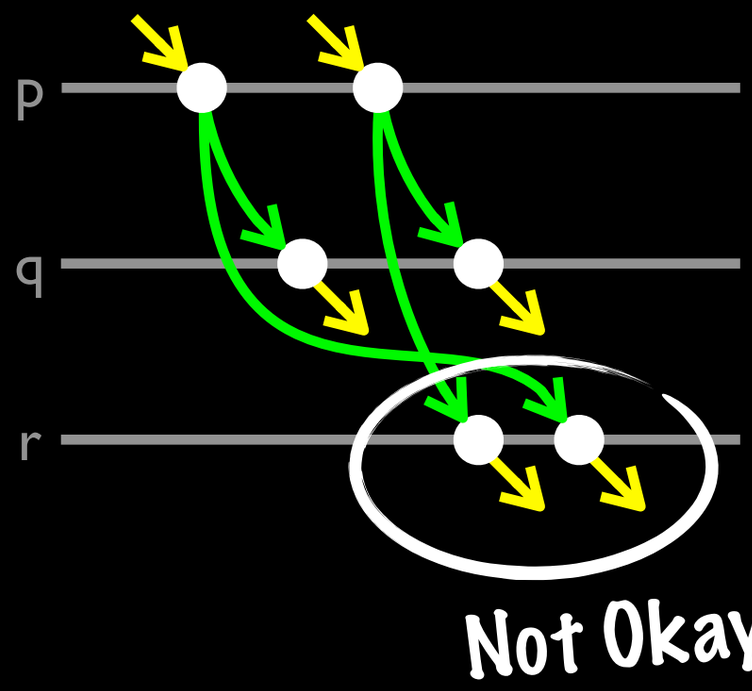


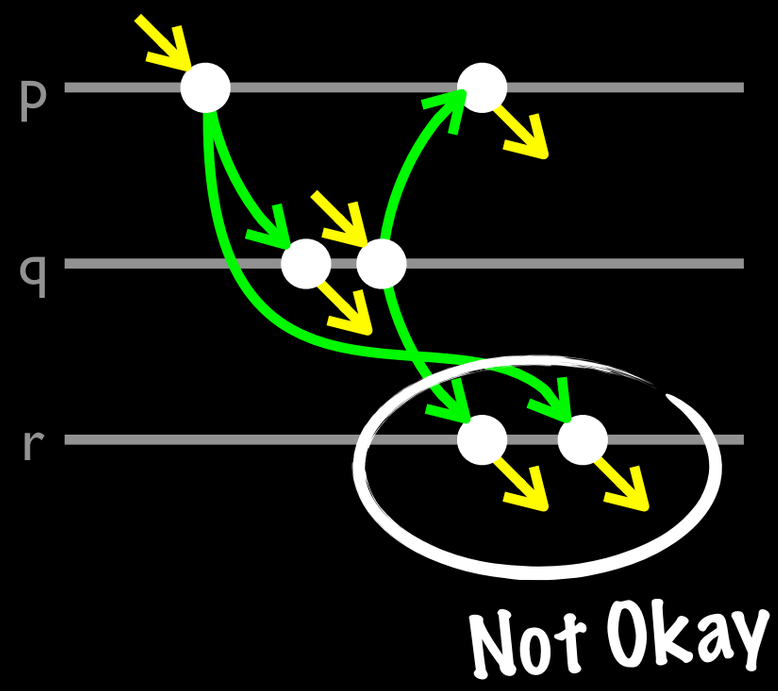
Module

Causal Reliable Broadcast

Validity, no duplication, no creation, and agreement: Reliable Broadcast

Causal order: For any message m_1 that potentially caused a message m_2, no process delivers m_2 unless it has already delivered m_1.

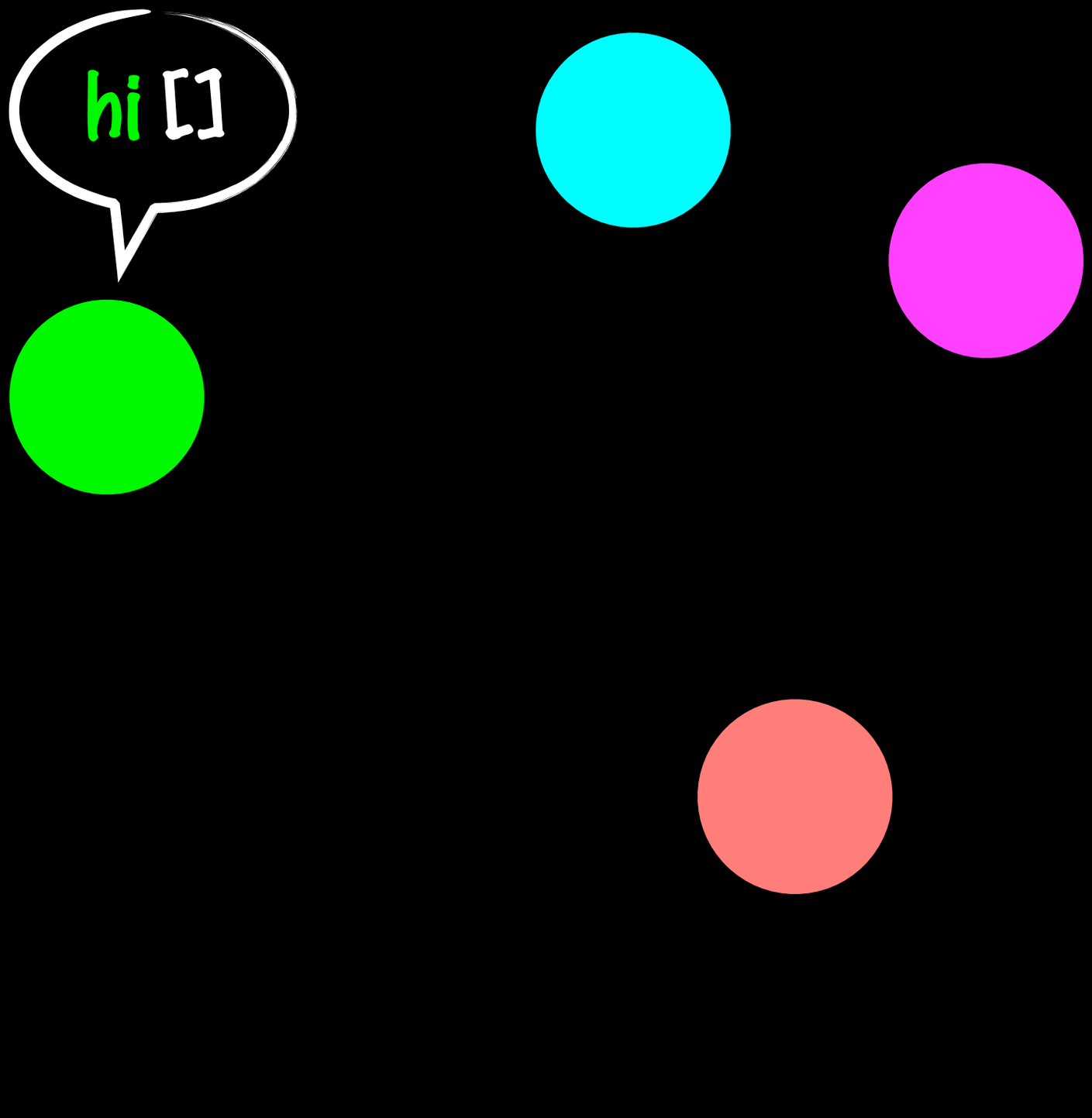


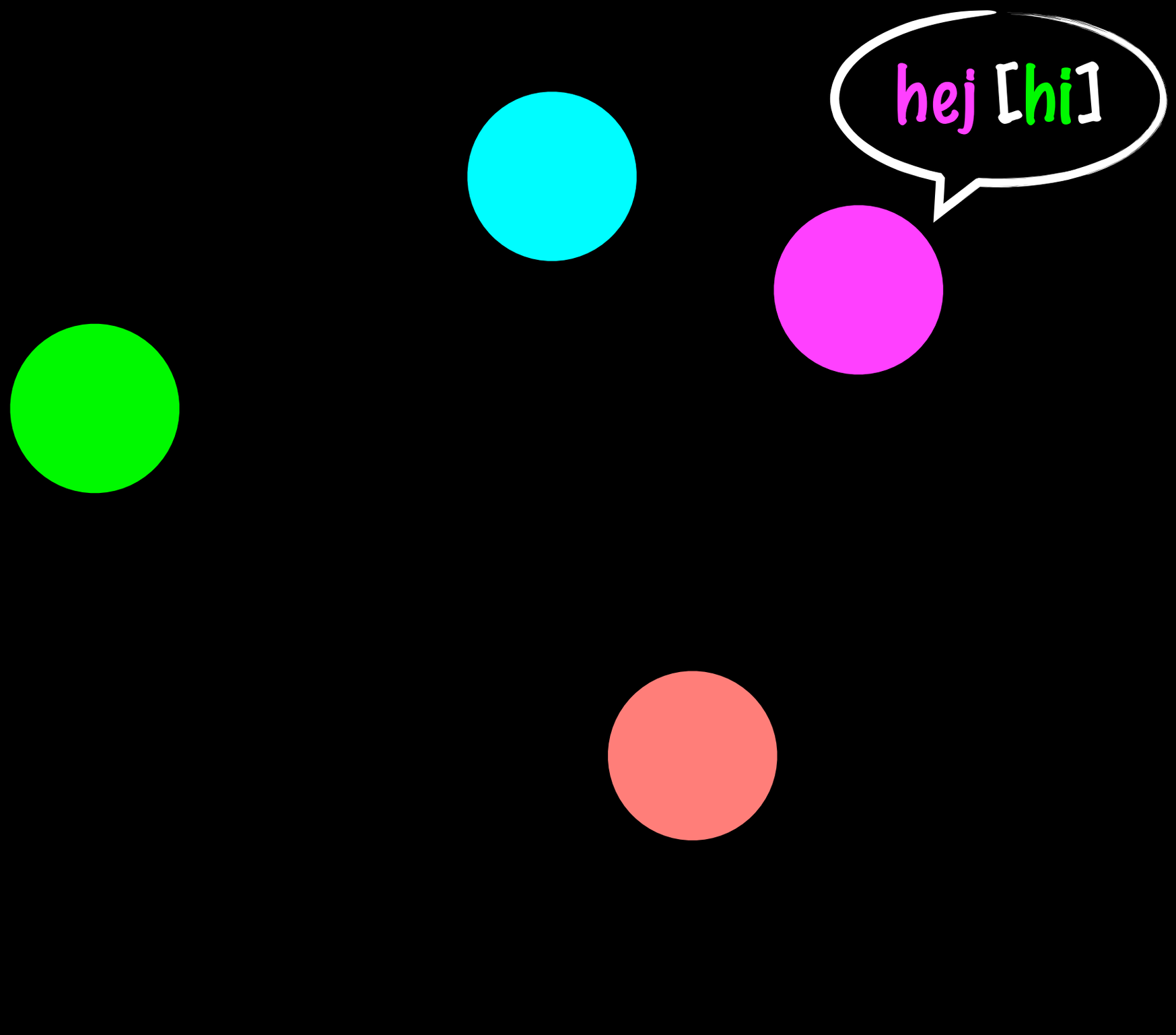


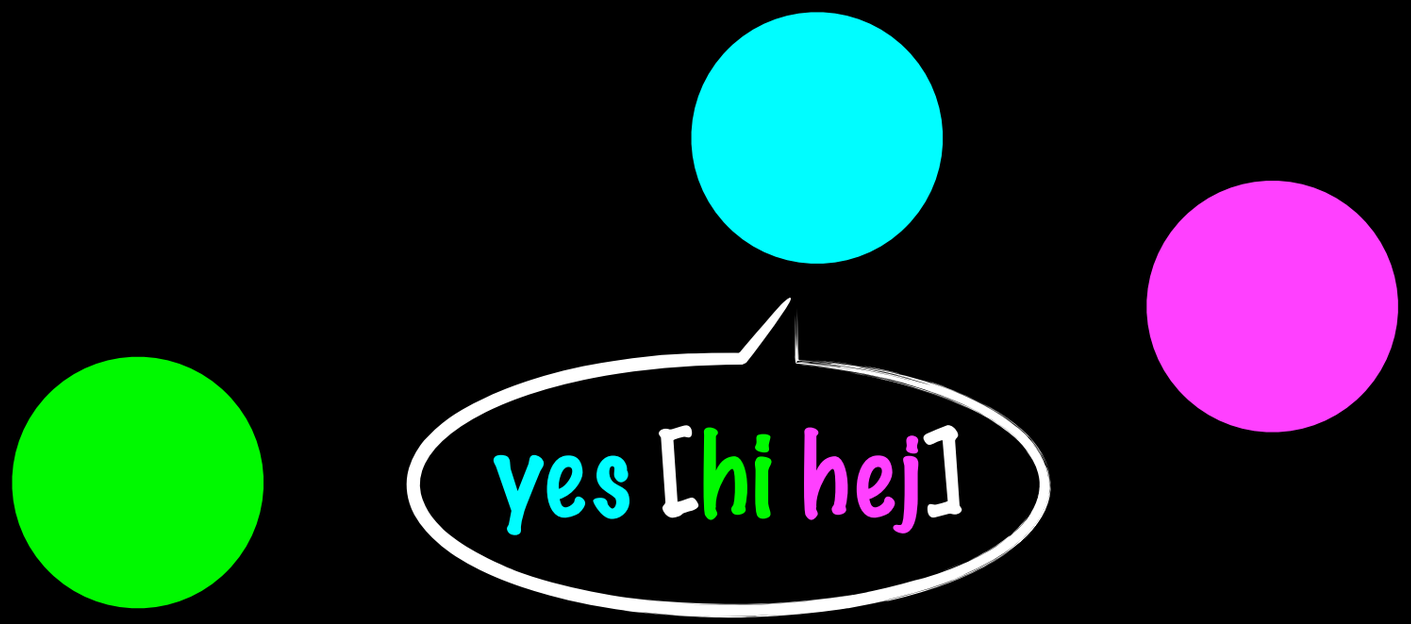
Fail-Silent Algorithm

No-Waiting Causal Broadcast

- Use a Reliable Broadcast module
- Send the history of all received messages together with each message
- When receiving a message, deliver all undelivered messages in its history before delivering the new message itself


hi []


hej [hi]


yes [hi hej]

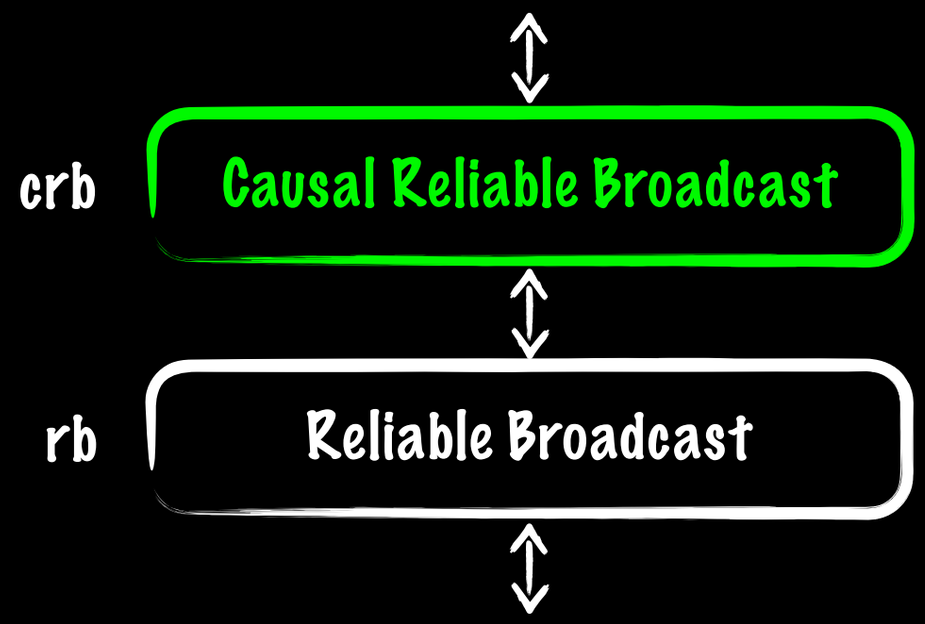
Fail-Silent Algorithm

No-Waiting Causal Broadcast

upon event ⟨ *crb*, Init ⟩ **do**
    delivered := ∅; *past* := [];

upon event ⟨ *crb*, *Broadcast* | *m* ⟩ **do**
    trigger ⟨ *rb*, *Broadcast* | [DATA, *past*, *m*] ⟩;
    append(*past*, (*self*, *m*));

upon event ⟨ *rb*, Deliver | p, [DATA, *mpast*, *m*] ⟩ **do**
    if *m* ∉ *delivered* **then**
        forall (*s*, *n*) ∈ *mpast* **do** // by the order in the list
            if *n* ∉ *delivered* **then**
                trigger ⟨ *crb*, *Deliver* | *s*, *n* ⟩;
                delivered := *delivered* ∪ {*n*};
                if (*s*, *n*) ∉ *past* **then** *append*(*past*, (*s*, *n*));
        trigger ⟨ *crb*, *Deliver* | *p*, *m* ⟩;
        delivered := *delivered* ∪ {*m*};
        if (*p*, *m*) ∉ *past* **then** *append*(*past*, (*p*, *m*));



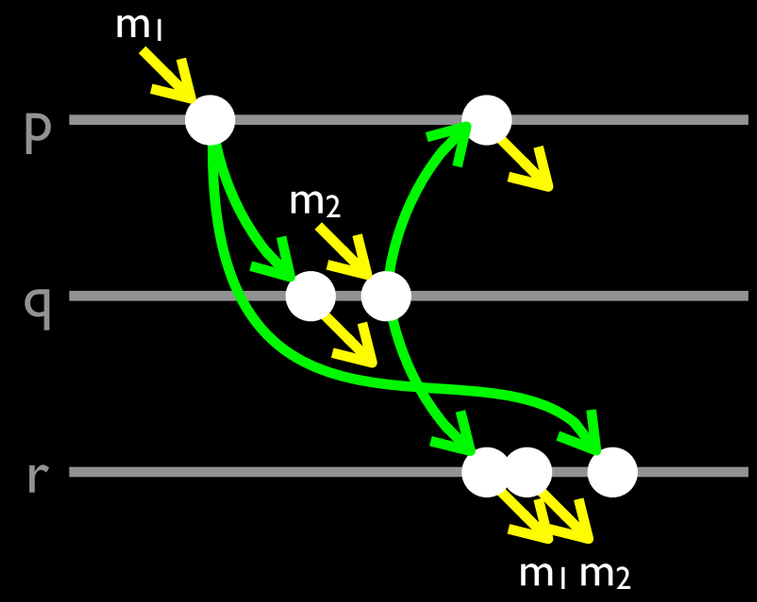
Fail-Silent Algorithm

No-Waiting Causal Broadcast

upon event ⟨ *crb*, Init ⟩ **do**
    delivered := ∅; *past* := [];

upon event ⟨ *crb*, *Broadcast* | *m* ⟩ **do**
    trigger ⟨ *rb*, *Broadcast* | [DATA, *past*, *m*] ⟩;
    append(*past*, (*self*, *m*));

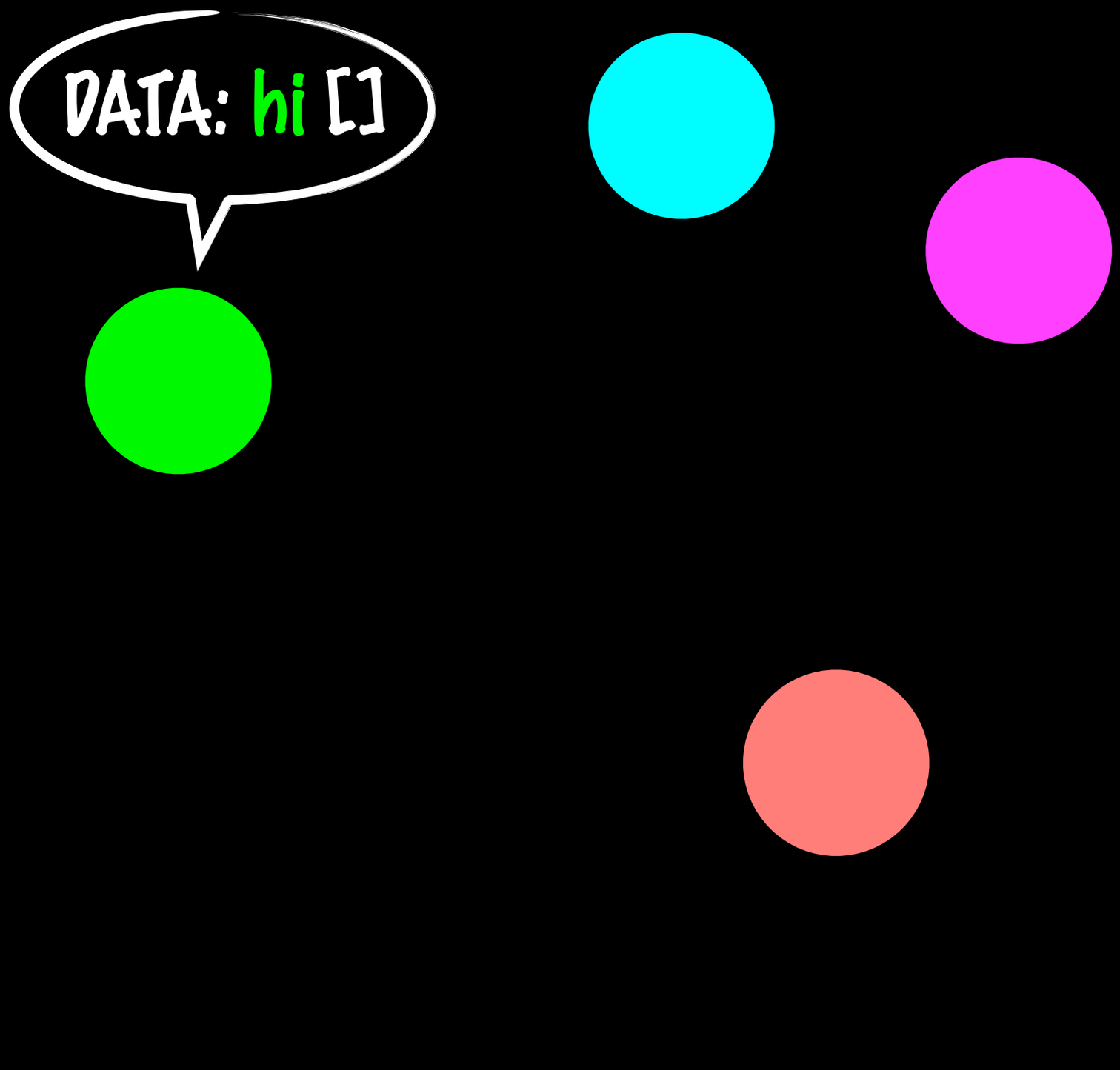
upon event ⟨ *rb*, Deliver | p, [DATA, *mpast*, *m*] ⟩ **do**
    if *m* ∉ *delivered* **then**
        forall (*s*, *n*) ∈ *mpast* **do** // by the order in the list
            if *n* ∉ *delivered* **then**
                trigger ⟨ *crb*, *Deliver* | *s*, *n* ⟩;
                delivered := *delivered* ∪ {*n*};
                if (*s*, *n*) ∉ *past* **then** *append*(*past*, (*s*, *n*));
        trigger ⟨ *crb*, *Deliver* | *p*, *m* ⟩;
        delivered := *delivered* ∪ {*m*};
        if (*p*, *m*) ∉ *past* **then** *append*(*past*, (*p*, *m*));

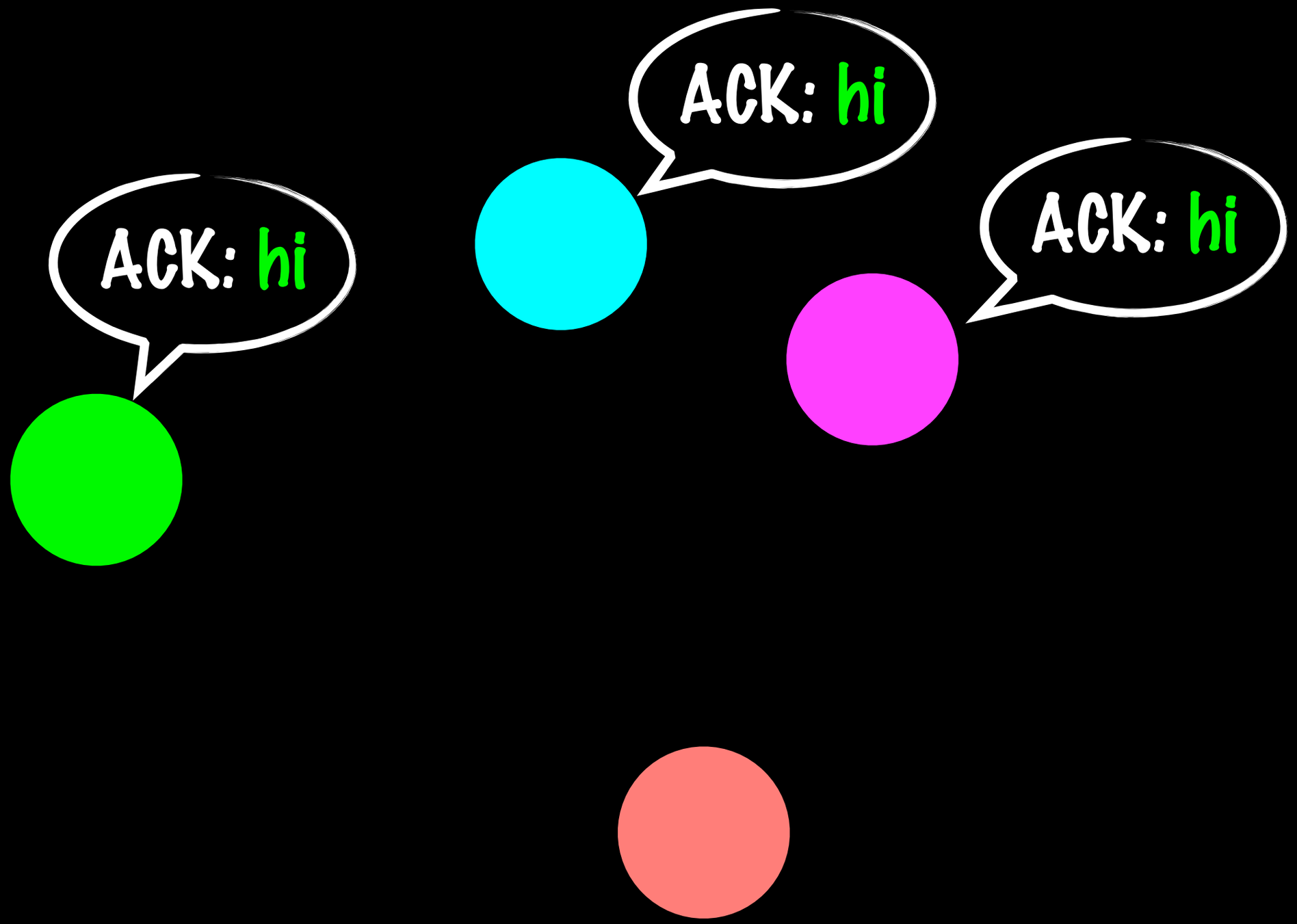


Fail-Stop Algorithm

Garbage-Collection of Causal Past

- Use a Reliable Broadcast module
- Use a Perfect Failure Detector module
- Send message history with each message
- Broadcast an *ack* for each delivered message.
- Remove a message from history when all correct processes have *ack*'ed a message


DATA: hi []


ACK: hi
ACK: hi
ACK: hi

DATA: hej [hi]

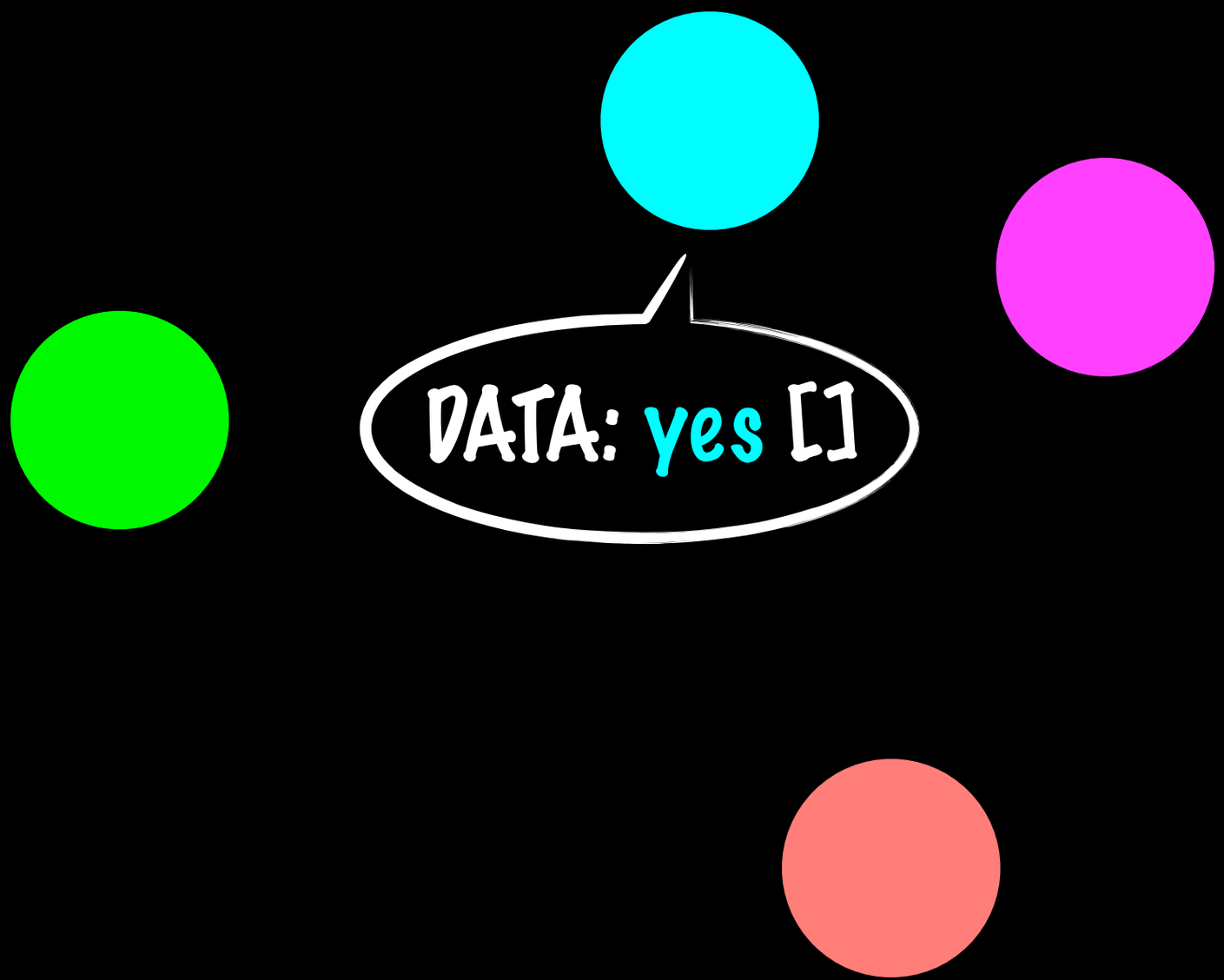
ACK: hej

ACK: hej

ACK: hej

ACK: hej

ACK: hi


DATA: yes []

ACK: yes

ACK: yes

ACK: yes

ACK: yes

Fail-Stop Algorithm

Garbage-Collection of Causal Past

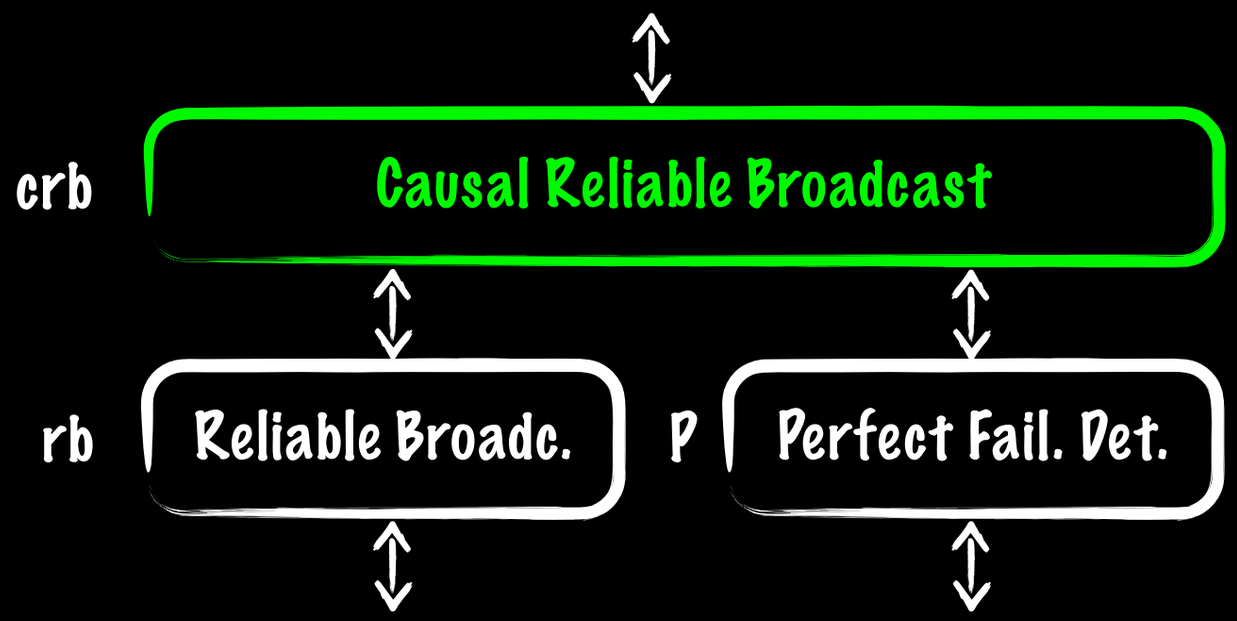
upon event ⟨ *crb*, *Init* ⟩ **do**
  delivered := ∅;
  past := [];
  correct := Π;
  forall *m* **do** *ack*[*m*] := ∅;

upon event ⟨ *P*, *Crash* | *p* ⟩ **do**
  correct := *correct* \ {*p*};

upon exists *m* ∈ *delivered* such that *self* ∉ *ack*[*m*] **do**
  ack[*m*] := *ack*[*m*] ∪ {*self*};
  trigger ⟨ *rb*, *Broadcast* | [ACK, *m*] ⟩;

upon event ⟨ *rb*, *Deliver* | *p*, [ACK, *m*] ⟩ **do**
  ack[*m*] := *ack*[*m*] ∪ {*p*};

upon *correct* ⊆ *ack*[*m*] **do**
  forall (*s*′, *m*′) ∈ *past* such that *m*′ = *m* **do**
    remove(*past*, (*s*′, *m*));



upon event ⟨ *crb*, *Broadcast* | *m* ⟩ **do**
  // same as before

upon event ⟨ *rb*, Deliver | *p*, [DATA, *mp*, *m*] ⟩ **do**
  // same as before

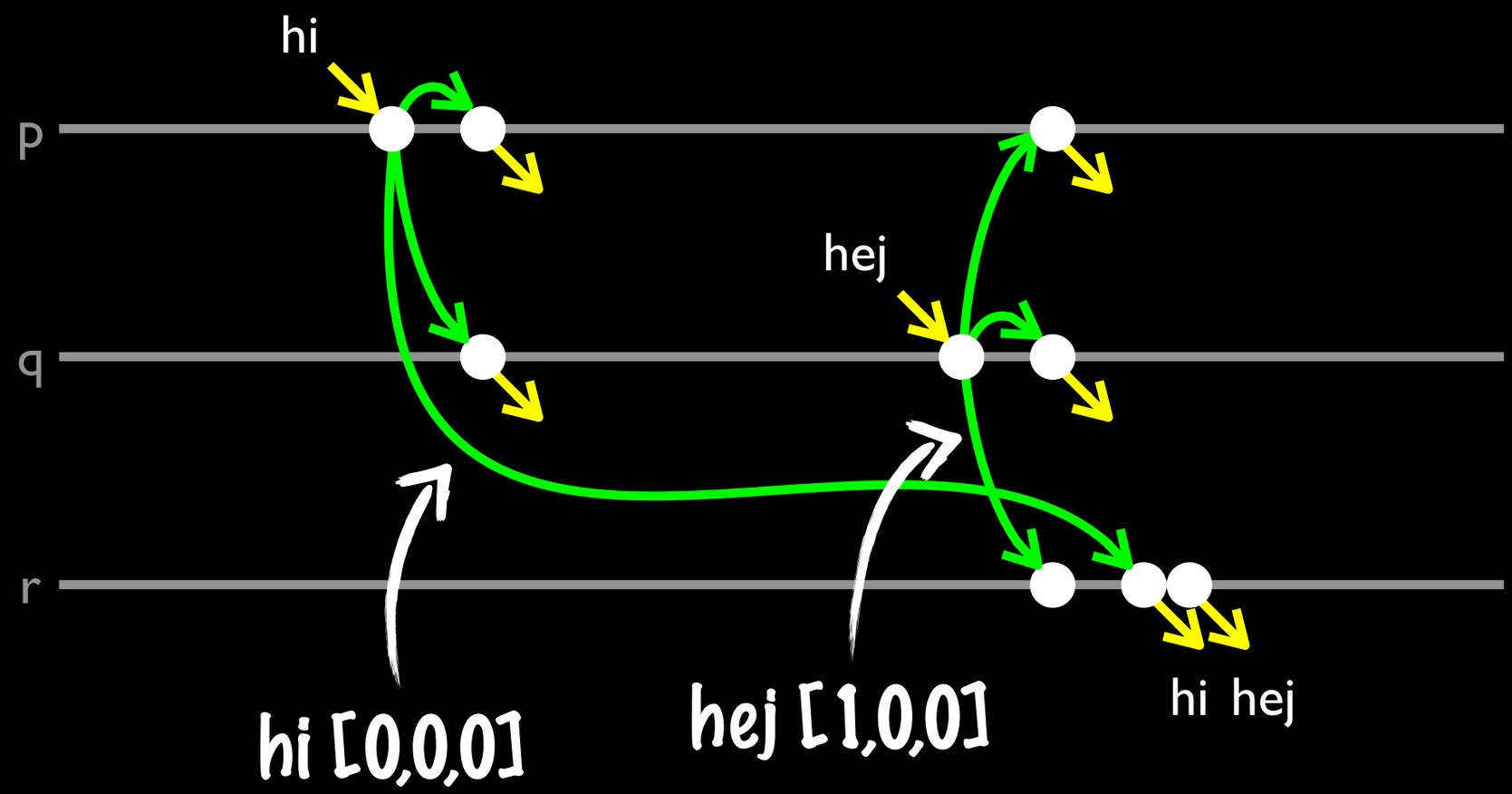
Fail-Silent Algorithm

Waiting Causal Broadcast

- Use a Reliable Broadcast module
- Record the number of delivered messages from each process in a vector *V*
- Send the vector *V* with each message
- Delay the delivery of a message *m* with vector *W* until $W \leq V \Leftrightarrow \forall i\ W[i] \leq V[i]$

Fail-Silent Algorithm

Waiting Causal Broadcast



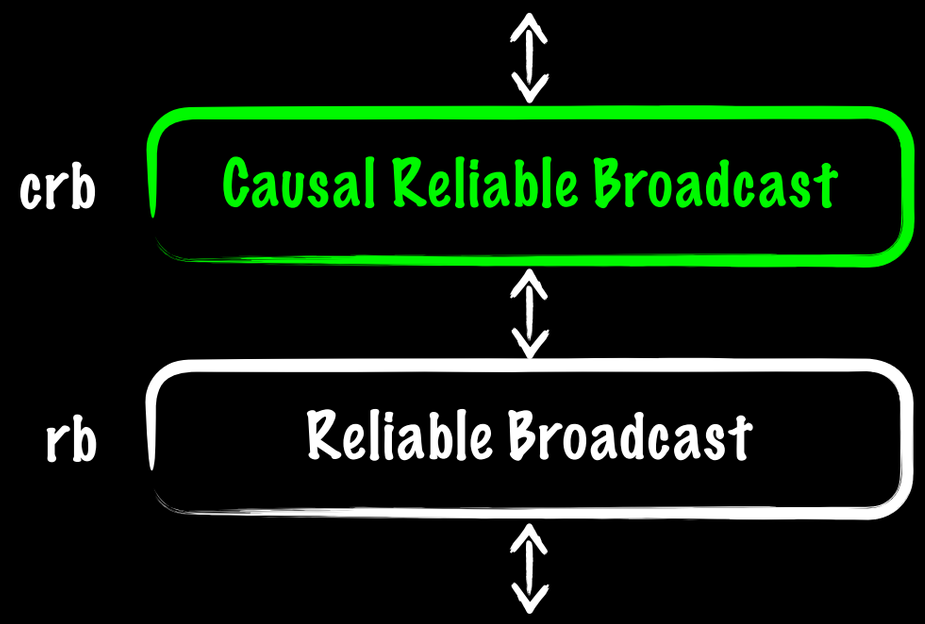
Fail-Silent Algorithm

Waiting Causal Broadcast

upon event ⟨ *crb*, *Init* ⟩ **do**
  $V := [0]^N$;
  $lsn := 0$;
  $pending := \emptyset$;

upon event ⟨ *crb*, *Broadcast* | *m* ⟩ **do**
  $W := V$;
  W [*rank*(*self*)] := *lsn*;
  $lsn := lsn + 1$;
  trigger ⟨ *rb*, *Broadcast* | [DATA, *W*, *m*] ⟩;

upon event ⟨ *rb*, *Deliver* | *p*, [DATA, *W*, *m*] ⟩ **do**
  $pending := pending \cup \{(p, W, m)\}$;
  while exists $(p', W', m') \in pending$ such that $W' \leq V$ **do**
    $pending := pending \setminus \{(p', W', m')\}$;
    $V[rank(p')] := V[rank(p')] + 1$;
    trigger ⟨ *crb*, *Deliver* | p', m' ⟩;



Part 2

Byzantine Processes

Module

Byzantine Consistent Broadcast

Validity: If a correct process p broadcasts a message m, then every correct process eventually delivers m.

No duplication: Every correct process delivers at most one message.

Integrity: If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

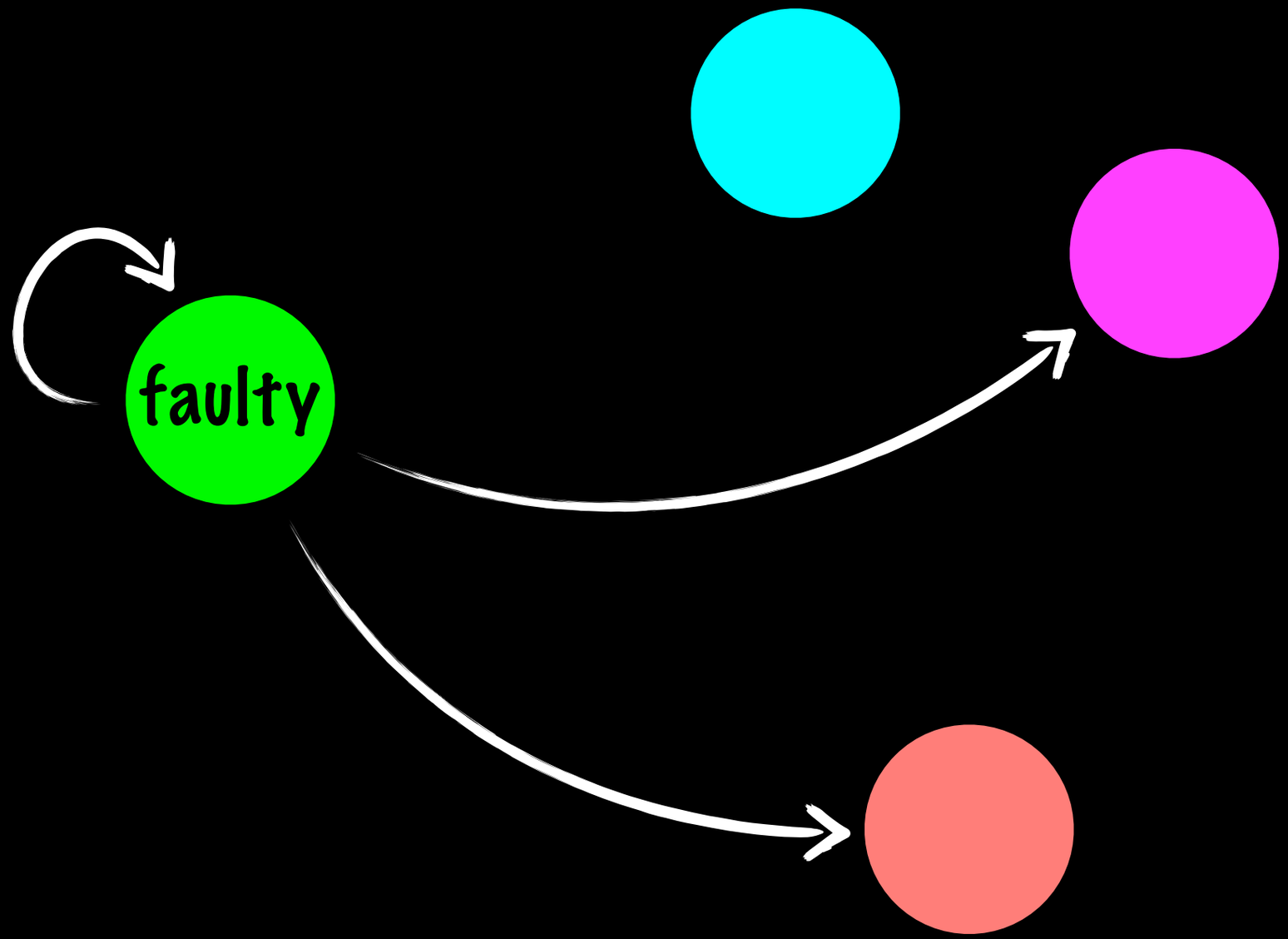
Consistency: If some correct process delivers a message m and another correct process delivers a message m', then $m = m'$.

Fail-Arbitrary Algorithm

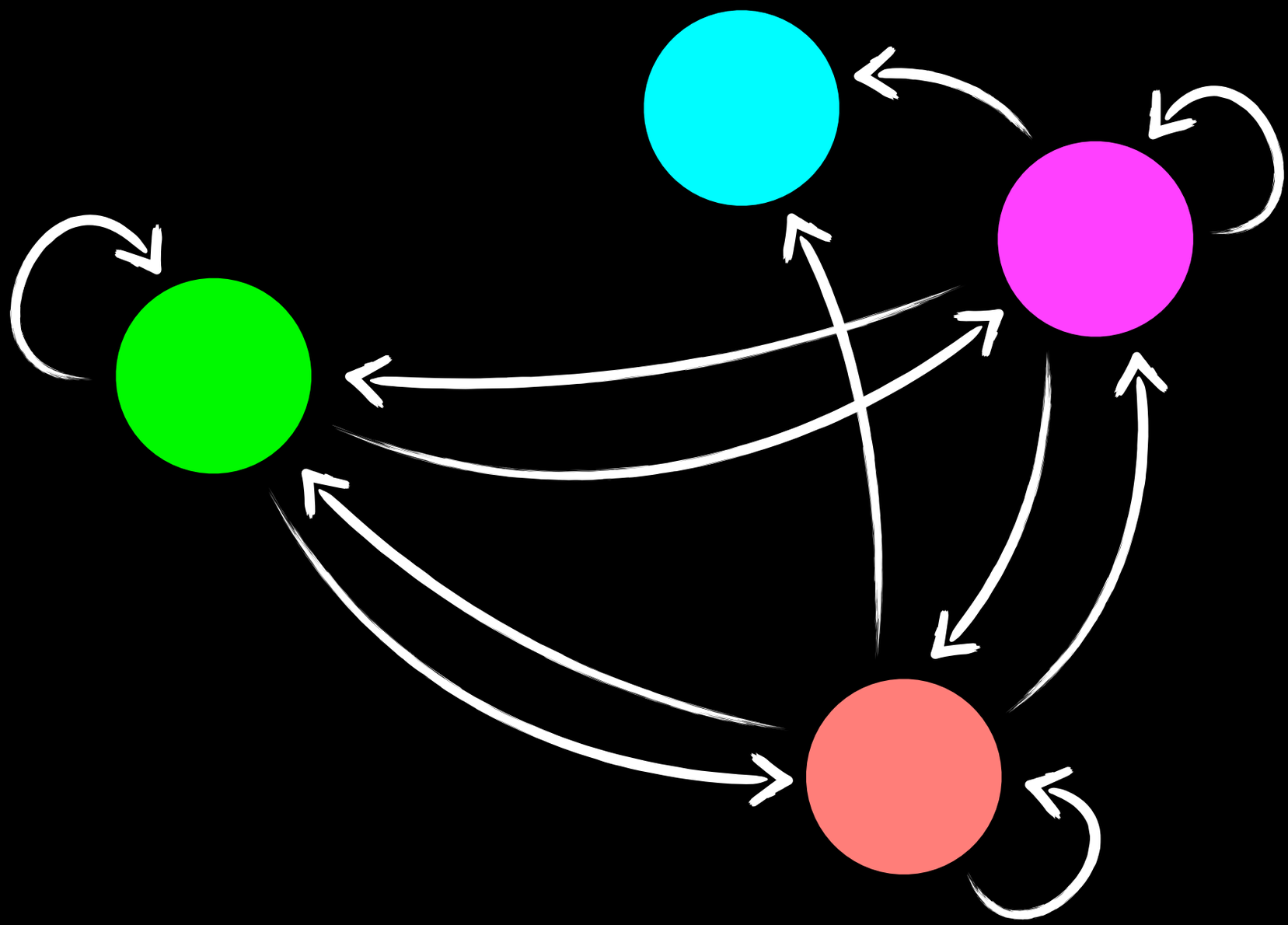
Authenticated Echo Broadcast

- Use Authenticated Point to Point Links
- When you receive the first message *m* from sender *s*, send *m* as ECHO to all other processes
- If you get more than (*N* + *f*) / 2 ECHO with the same message *m*, then deliver *m*

SEND



ECHO



DELIVERY

Fail-Arbitrary Algorithm

Authenticated Echo Broadcast

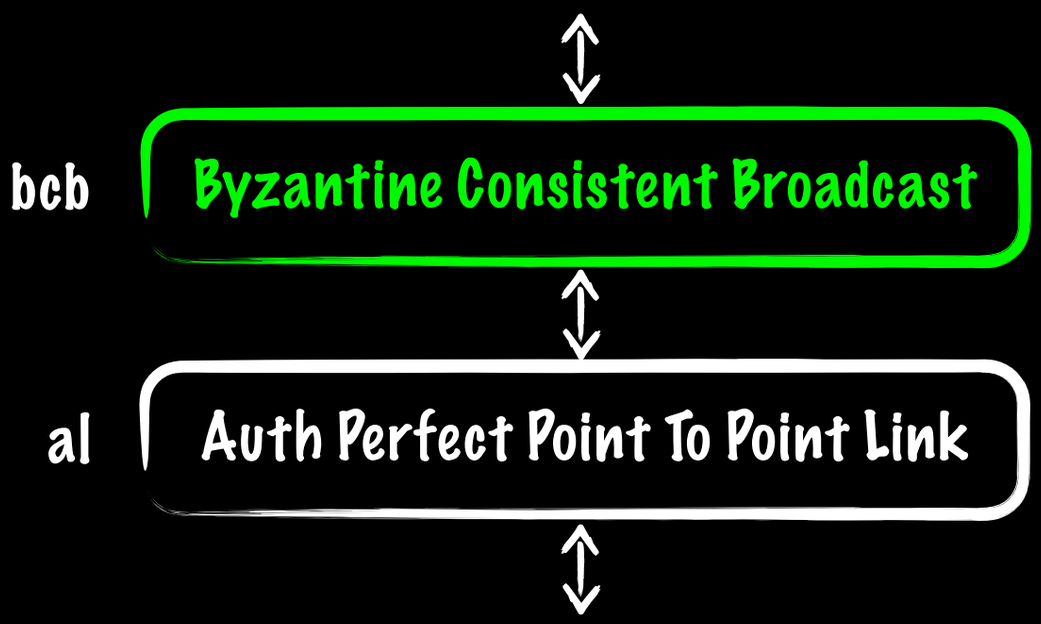
upon event ⟨ *bcb*, *Init* ⟩ **do**
    sentecho := FALSE;
    delivered := FALSE;
    echos := $[\bot]^N$;

upon event ⟨ *bcb*, *Broadcast* | *m* ⟩ **do**
    forall $q \in \Pi$ **do**
        trigger ⟨ *al*, *Send* | q, [SEND, *m*] ⟩;

upon event ⟨ *al*, *Deliver* | *p*, [SEND, *m*] ⟩ such that *p* = *s*
        and *sentecho* = FALSE **do**
    sentecho := TRUE;
    forall $q \in \Pi$ **do**
        trigger ⟨ *al*, *Send* | *q*, [ECHO, *m*] ⟩;

upon event ⟨ *al*, *Deliver* | *p*, [ECHO, *m*] ⟩ **do**
    if *echos*[*p*] = ⊥ **then**
        echos[*p*] := *m*;

upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid echos[p] = m\}) > (N+f)/2$
        and *delivered* = FALSE **do**
    delivered := TRUE;
    trigger ⟨ *bcb*, *Deliver* | *s*, *m* ⟩;



Module

Byzantine Reliable Broadcast

Validity, No duplication, Integrity and Consistency: Byzantine Consistent Broadcast

Totality: If some message is delivered by any correct process, every correct processes eventually delivers a message

Fail-Arbitrary Algorithm

Authenticated Double-Echo Broadcast

- Use Authenticated Point to Point Links
- When you receive the first message *m* from sender *s*, send *m* as ECHO to all other processes
- If you get more than (*N* + *f*) / 2 ECHO or more than *f* READY with the same message *m*, then send *m* as READY to all other processes
- If you get more than 2*f* READY with the same message *m*, then deliver *m*

SEND

faulty

0 Echo
0 Ready

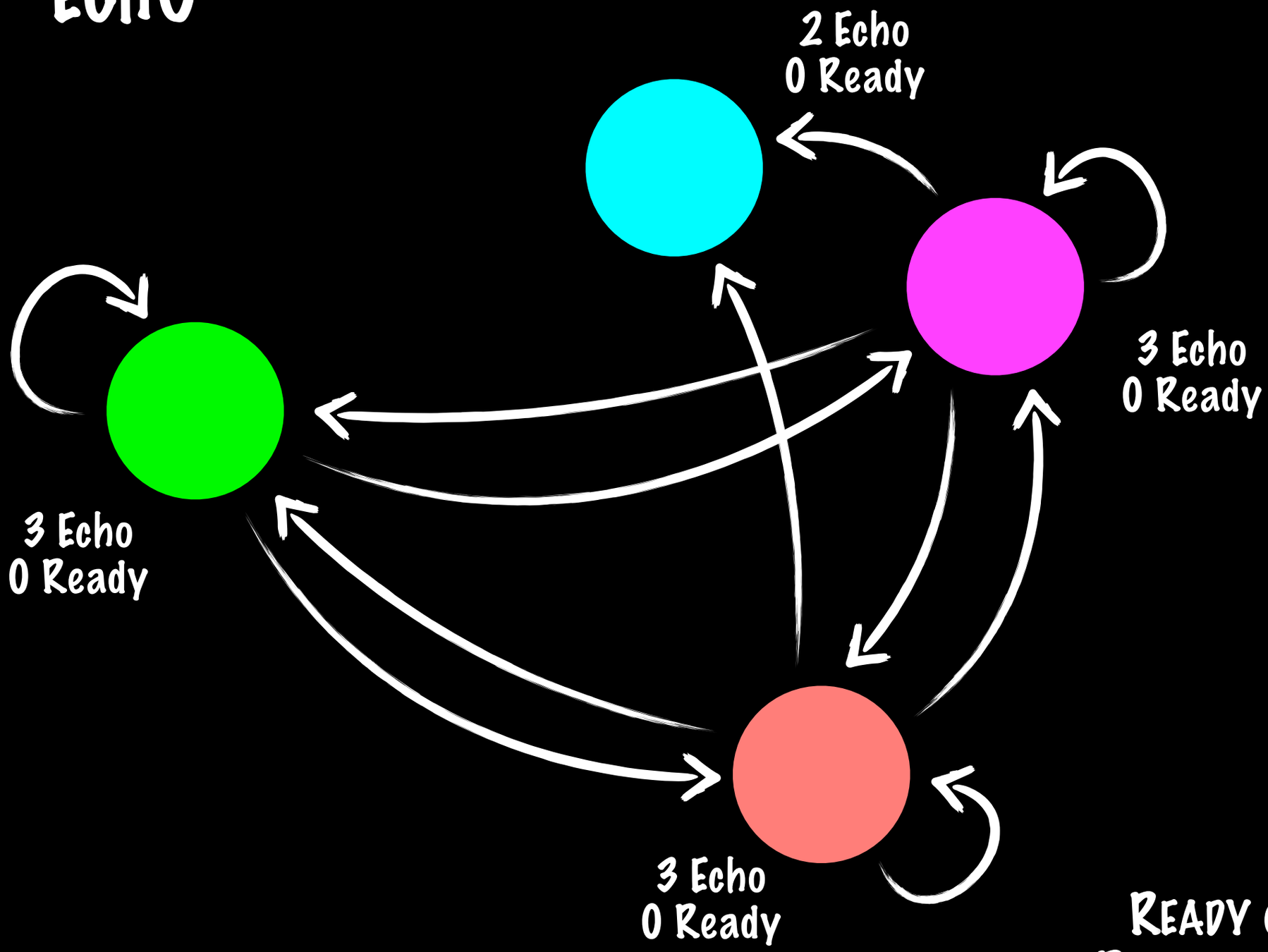
0 Echo
0 Ready

0 Echo
0 Ready

0 Echo
0 Ready

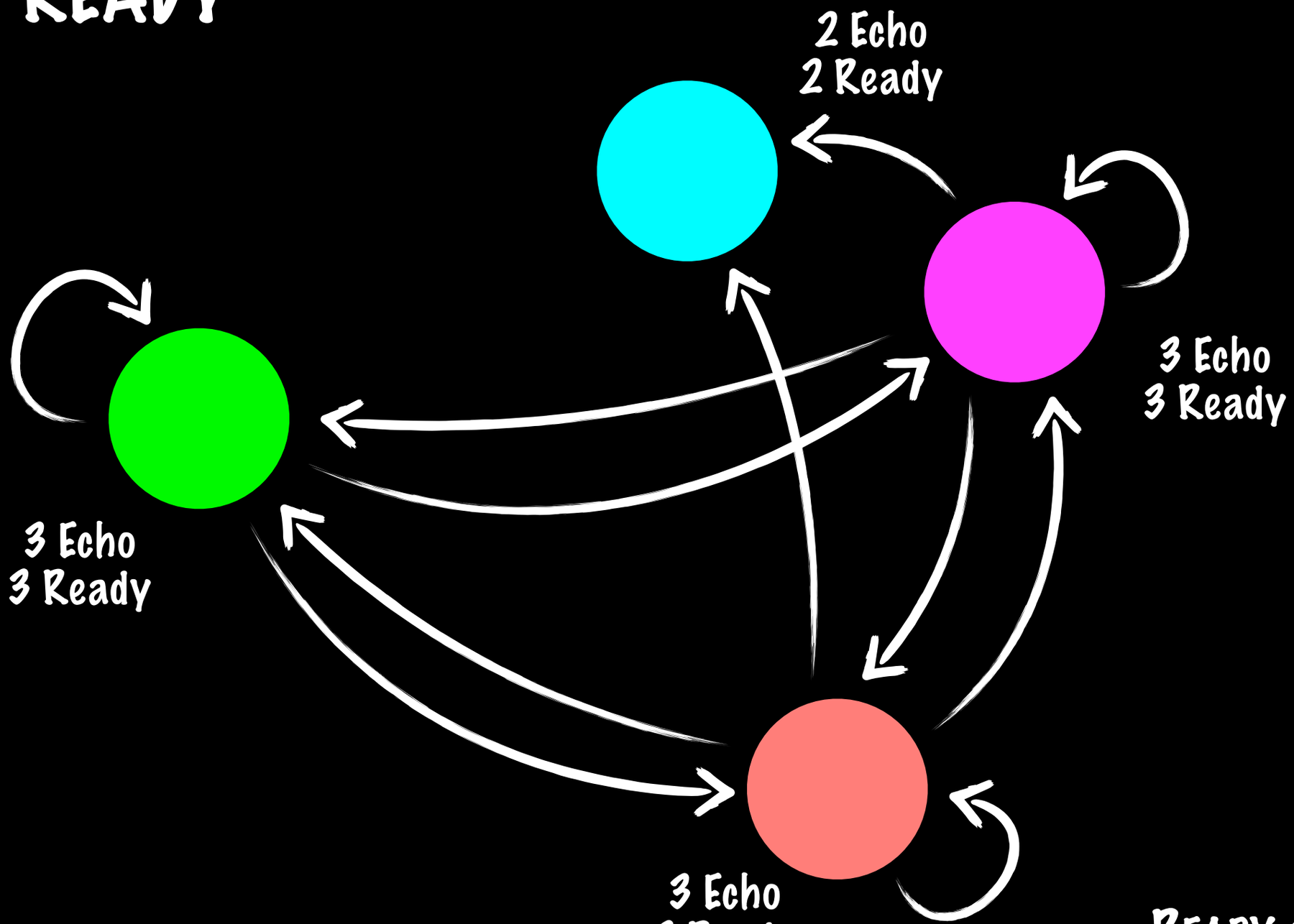
N = 4
f = 1
READY on 3 ECHO
READY on 2 READY

ECHO



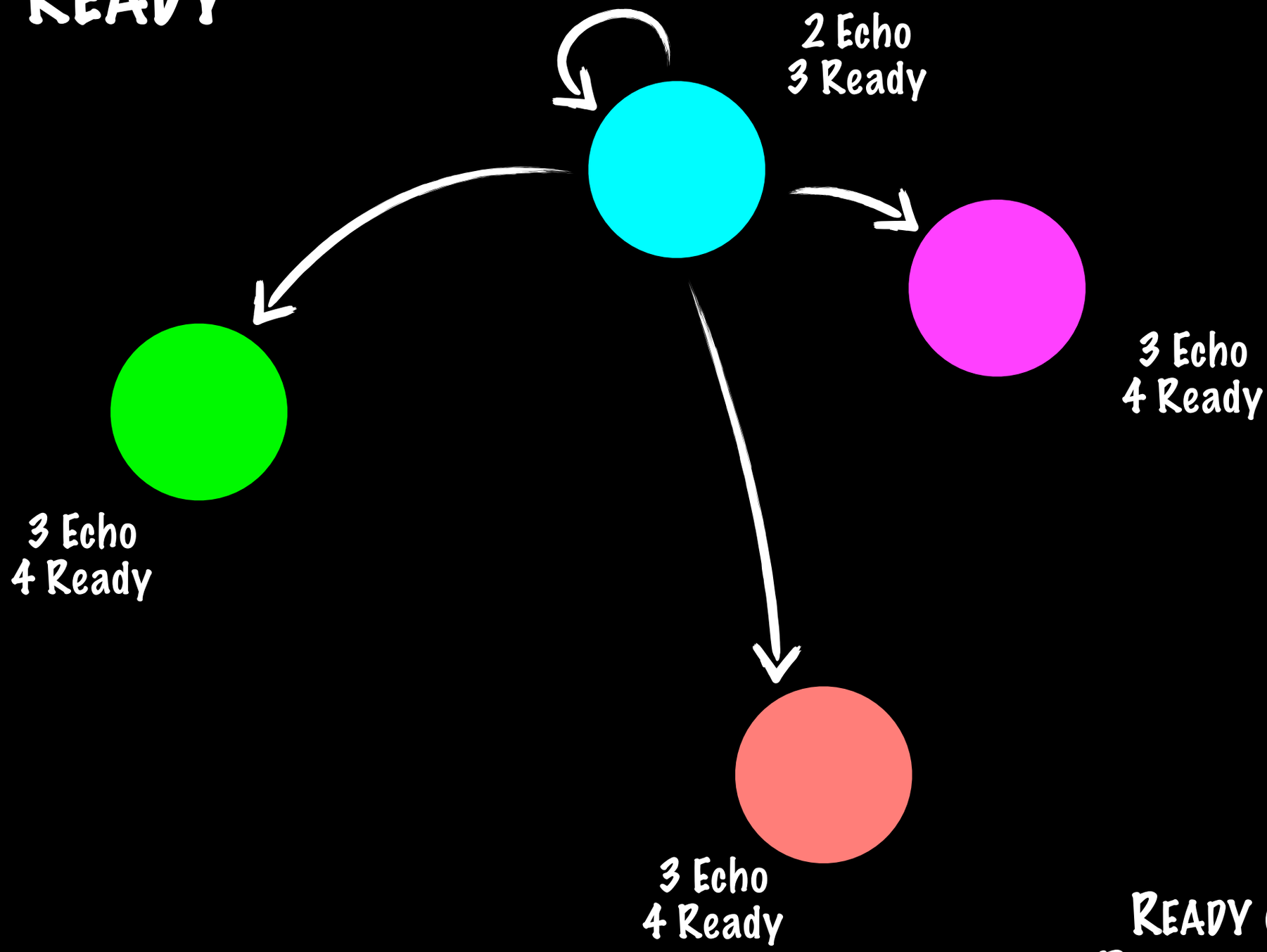
N = 4
f = 1
Ready on 3 Echo
Ready on 2 Ready

READY



N = 4
f = 1
READY on 3 ECHO
READY on 2 READY

READY



N = 4
f = 1
READY on 3 ECHO
READY on 2 READY

DELIVERY

2 Echo
3 Ready

3 Echo
4 Ready

3 Echo
4 Ready

3 Echo
4 Ready

N = 4
f = 1
Ready on 3 Echo
Ready on 2 Ready

Module

Byzantine Consistent Broadcast Channel

Validity: If a correct process p broadcasts a message m, then every correct process eventually delivers m.

No duplication: For every process p and label l, every correct process delivers at most one message with label l and sender p.

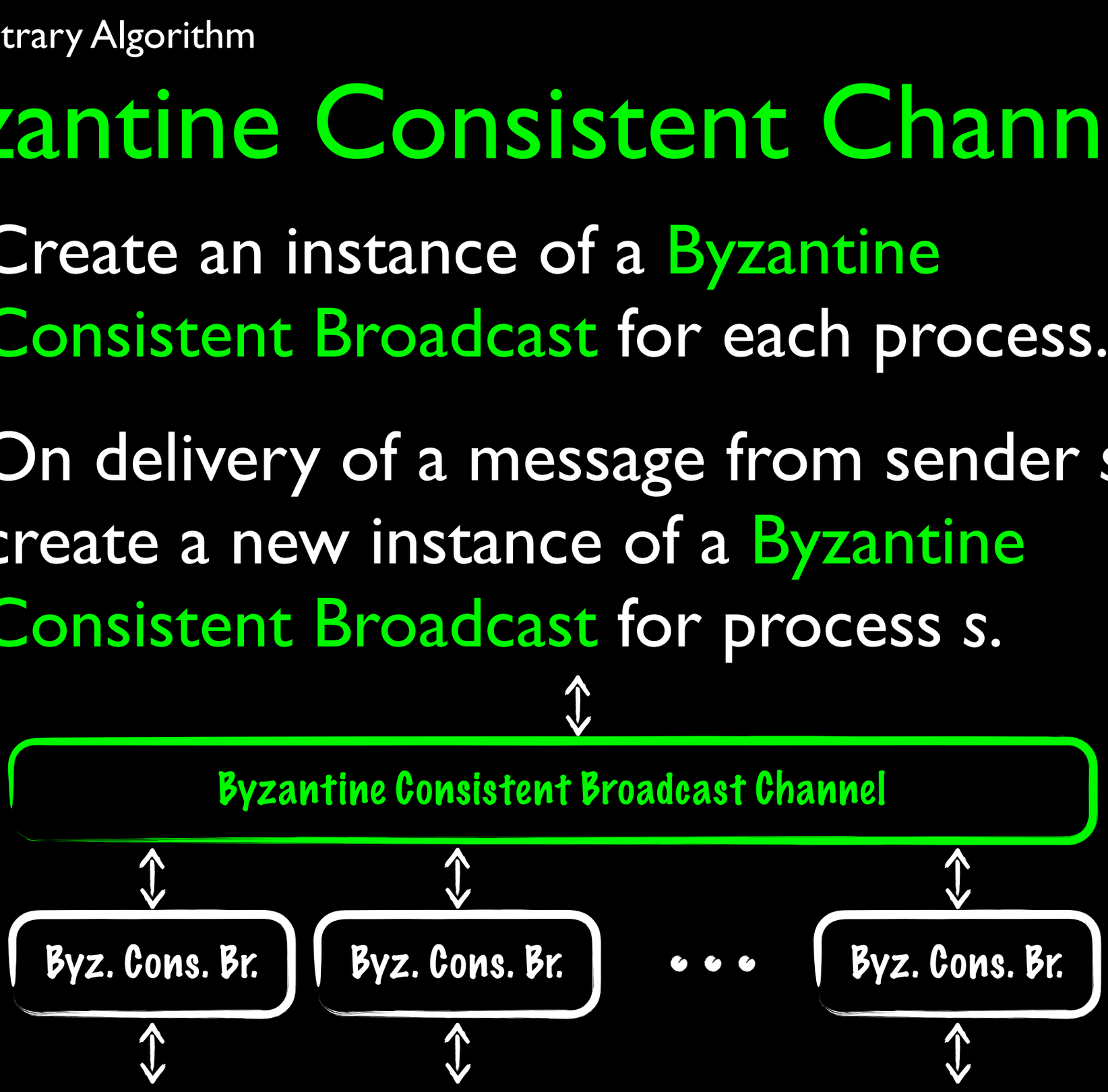
Integrity: If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

Consistency: If some correct process delivers a message m with label l and sender s, and another correct process delivers a message m' with label l and sender s, then $m = m'$.

Fail-Arbitrary Algorithm

Byzantine Consistent Channel

- Create an instance of a Byzantine Consistent Broadcast for each process.
- On delivery of a message from sender *s*, create a new instance of a Byzantine Consistent Broadcast for process *s*.



Module

Byzantine Reliable Broadcast Channel

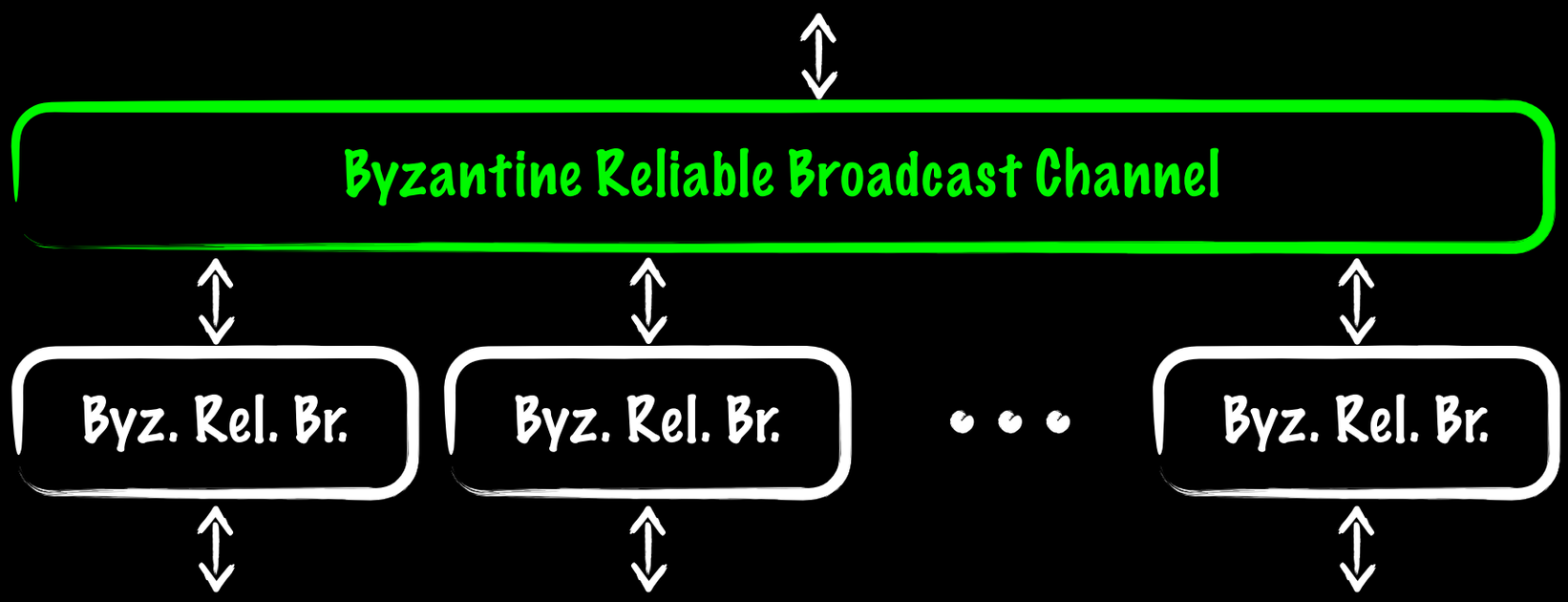
Validity, No duplication, Integrity and Consistency: Byzantine Consistent Broadcast Channel.

Agreement: If some correct process delivers a message *m* with label *l* and sender *s*, then every correct process eventually delivers message *m* with label *l* and sender *s*.

Fail-Arbitrary Algorithm

Byzantine Reliable Channel

- Create an instance of a Byzantine Reliable Broadcast for each process.
- On delivery of a message from sender *s*, create a new instance of a Byzantine Reliable Broadcast for process *s*.



Lunch