

Towards a dependable consensus: Broadcast primitives (cont.) and read/write protocols

Highly dependable systems

Lecture 4

Lecturers: Miguel Matos and Paolo Romano

Byzantine Consistent Broadcast: specification

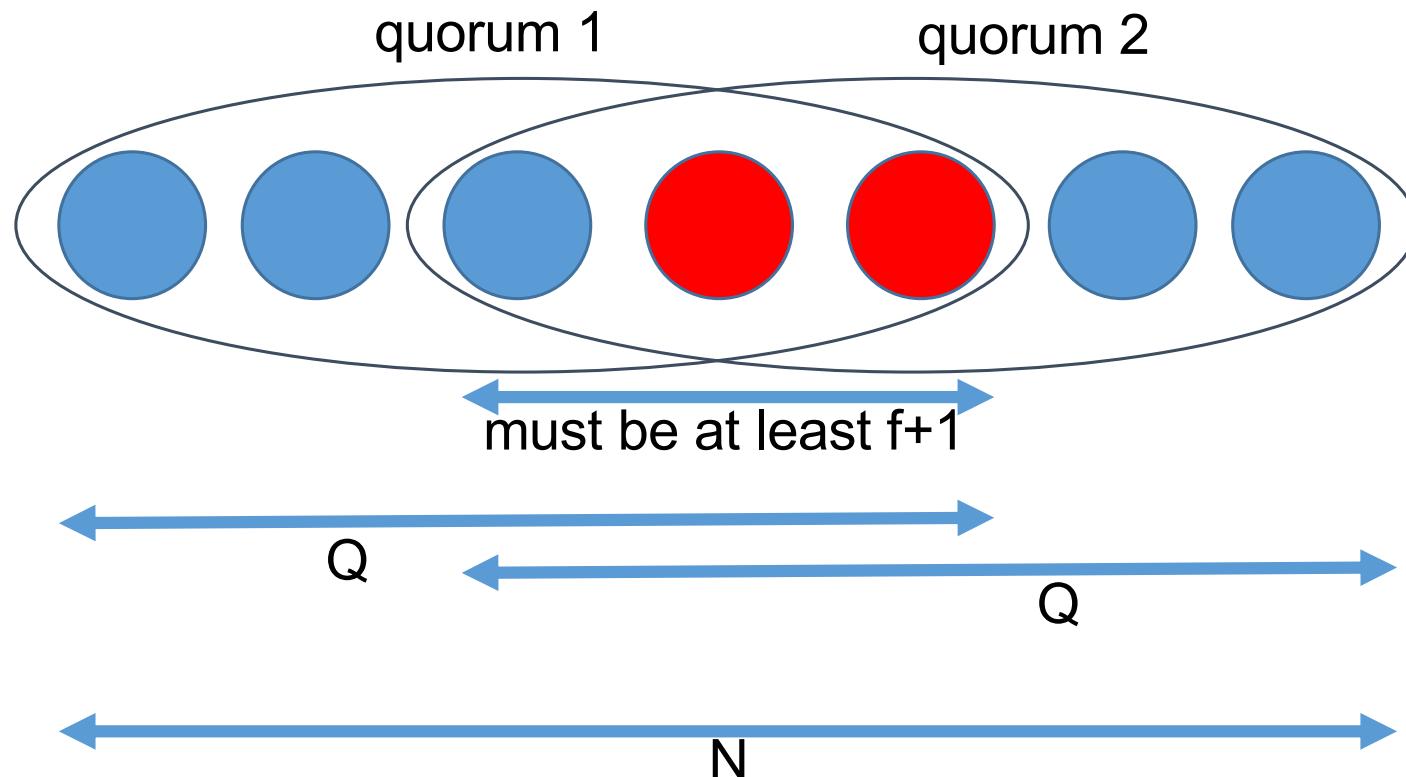
- **BCB1: Validity:** If a correct process p broadcasts a msg m , then every correct process eventually delivers m .
- **BCB2: No duplication:** Every correct process delivers at most one message.
- **BCB3: Integrity:** If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p .

BCB4: Consistency: If some correct process delivers a message m and another correct process delivers a message m' , then $m = m'$.

Note: “Correct” now means non-Byzantine-faulty

Derivation of Byzantine quorum sizes

- Safety: every pair of quorums must intersect in at least one correct process



Solving a system of two equations

- Note that f is a system parameter (constant)

$$\begin{cases} 2Q - N = f+1 \\ N - f = Q \end{cases}$$

- Thus, solving these equations for N and Q yields:

$$N = 3f+1$$

$$Q = 2f+1$$

- Generically, for $N > 3f$, we have $Q = \lceil (N+f)/2 \rceil$

Authenticated Echo Broadcast

- exploits *Byzantine quorums* (uses N,Q derived before)
 - two rounds of message exchanges.
 - 1st round: sender disseminates msg to all processes.
 - 2nd round:
 - every process acts as a witness for m
 - resends m in an ECHO message to all others.
 - deliver only when received more than a quorum (Q) of ECHOs
 - Byzantine processes cannot cause a correct process to *bcb-deliver* a message $m' \neq m$.
- Relies on authenticated links

Authenticated Echo Broadcast

- **Implements:**
ByzantineConsistentBroadcast, with sender s .
- **Uses:**
AuthPerfectPointToPointLinks, **instance** al .
- **upon event** $\langle bcb, \text{Init} \rangle$ **do**
 $\text{sentecho} := \text{FALSE};$
 $\text{delivered} := \text{FALSE};$
 $\text{echos} := [\perp]^N;$

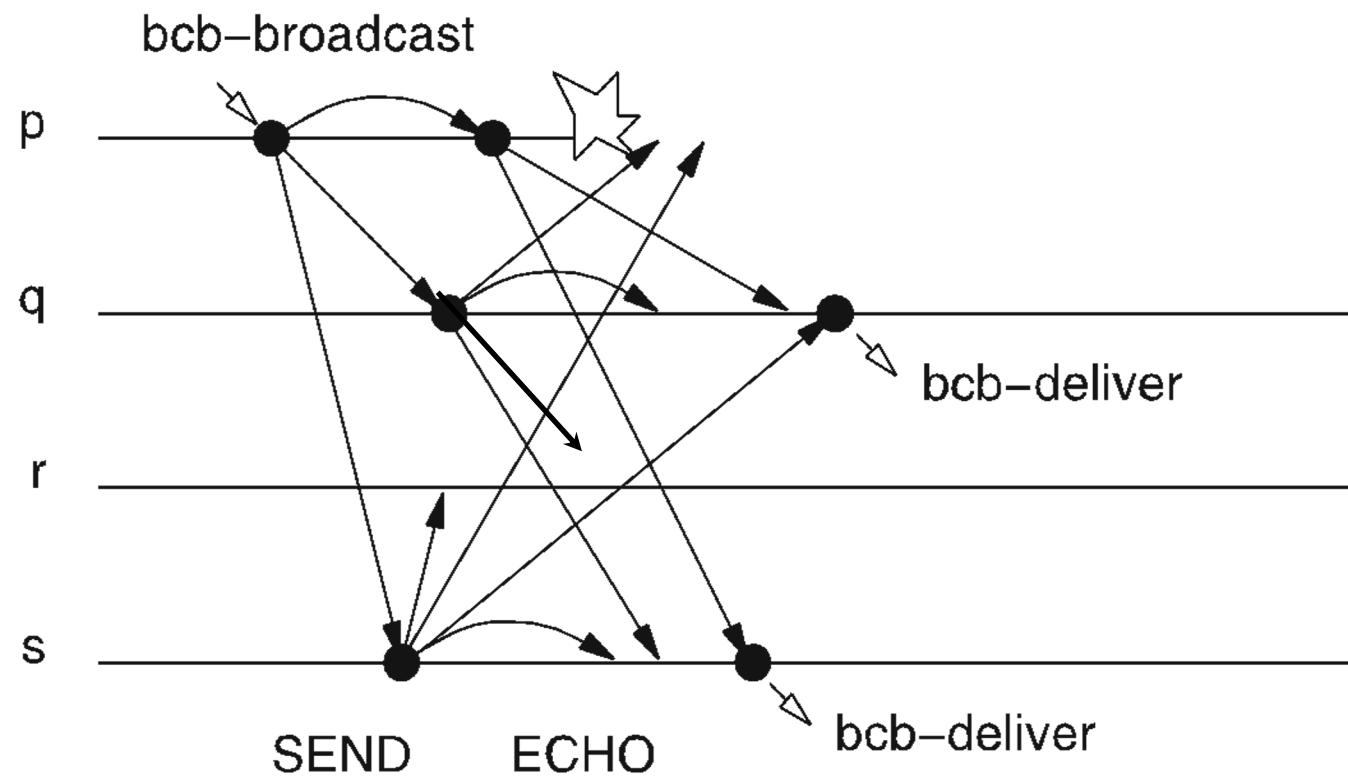
Authenticated Echo Broadcast

- **upon event** $\langle bcb, \text{Broadcast} \mid m \rangle$ **do** // only process s
forall $q \in \Pi$ **do**
 trigger $\langle al, \text{Send} \mid q, [\text{SEND}, m] \rangle$;
- **upon event** $\langle al, \text{Deliver} \mid p, [\text{SEND}, m] \rangle$
 such that $p = s$ **and** $\text{sentecho} = \text{FALSE}$ **do**
 $\text{sentecho} := \text{TRUE}$;
 forall $q \in \Pi$ **do**
 trigger $\langle al, \text{Send} \mid q, [\text{ECHO}, m] \rangle$;

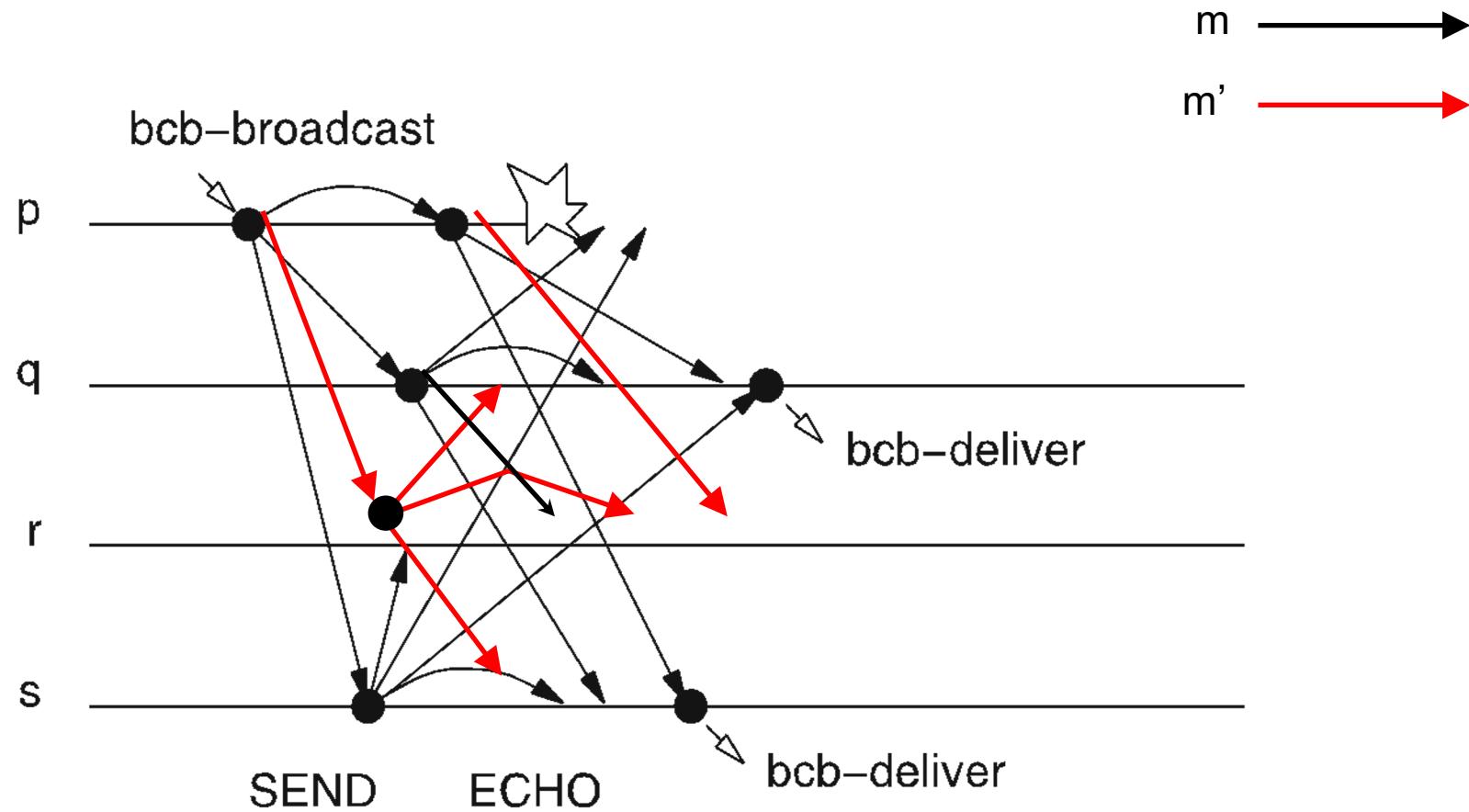
Authenticated Echo Broadcast

- **upon event** $\langle al, \text{Deliver} \mid p, [\text{ECHO}, m] \rangle$ **do**
if $\text{echos}[p] = \perp$ **then**
 $\text{echos}[p] := m;$
- **upon exists** $m \neq \perp$ **such that** :
 $\#\{p \in \Pi \mid \text{echos}[p] = m\} > (N+f)/2$ **and** $\text{delivered} =$
 FALSE do
 $\text{delivered} := \text{TRUE};$
 trigger $\langle bcb, \text{Deliver} \mid s, m \rangle;$

Authenticated Echo Broadcast



Authenticated Echo Broadcast



r can never deliver m' , if q and s deliver m

r should get 3 ECHOES, but can get at most 2 (including its own)

Correctness

BCB1: Validity: If a correct process p broadcasts a msg m , then every correct process eventually delivers m .

Proof (sketch)

- if the sender is correct, then every correct process a/l -sends an ECHO message
- every correct process a/l -delivers at least $N - f$ of them.
- $N - f > (N + f)/2$ under the assumption that $N > 3f$
implies that every correct process also bcb -delivers the message m contained in the ECHO messages

Correctness

- **BCB2: No duplication:** Every correct process delivers at most one message.
- **Proof (sketch):** enforced by the *delivered* variable

Correctness

- **BCB3: *Integrity*:** If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p .
- **Proof (sketch):** by the Authenticated Perfect Link's properties

Correctness

- **BCB4: Consistency:** If some correct process delivers a message m and another correct process delivers a message m' , then $m = m'$.
- For a correct process p to bcb -deliver some m , it needs to receive ECHO messages for m from a Byzantine quorum.

Correctness

- Two Byzantine quorums overlap in at least one correct process.
- Assume, by contradiction, that a different correct process p' *bcb*-delivers some message $m' \neq m$.
 - p' has received a Byzantine quorum of ECHO messages for m'
 - the correct process in the intersection of the two Byzantine quorums sent different ECHO messages to p and to p'

Contradiction found!

Signed echo broadcast

- Exploits digital signatures:
 - more powerful than MACs
 - allow a third process to verify the authenticity of a message sent from a 1st process s to a 2nd process r
 - avoid quadratic number of message exchanges of prev. algorithm

Signed echo broadcast

- High level idea:
 - Witnesses authenticate a request not by sending an ECHO message to all processes
 - but by signing a statement which they return to the sender
 - Sender collects a Byzantine quorum of these signed statements and relays them in a third communication step to all processes.

Signed Echo Broadcast

- **Implements:**
ByzantineConsistentBroadcast, **instance** *bcb*.
- **Uses:**
AuthPerfectPointToPointLinks, **instance** *al*.
- **upon event** < *bcb*, Init > **do**
sentecho := FALSE;
sentfinal := FALSE;
delivered := FALSE;
echos := [\perp]^N ; Σ := [\perp]^N ;

Signed Echo Broadcast

- **upon event** $\langle bcb, \text{Broadcast} \mid m \rangle$ **do** // only process s
forall $q \in \Pi$ **do**
 trigger $\langle al, \text{Send} \mid q, [\text{SEND}, m] \rangle$;
- **upon event** $\langle al, \text{Deliver} \mid p, [\text{SEND}, m] \rangle$
 such that $p = s$ **and** $\text{sentecho} = \text{FALSE}$ **do**
 $\text{sentecho} := \text{TRUE};$
 $\sigma = \text{sign}(self, bcb \parallel self \parallel \text{ECHO} \parallel m);$
 trigger $\langle al, \text{Send} \mid p, [\text{ECHO}, m], \sigma \rangle$;

algorithm instance identifier bcb is included in the input argument to the digital signature (else, a Byzantine process might reuse a signature issued by a correct process in a different context)

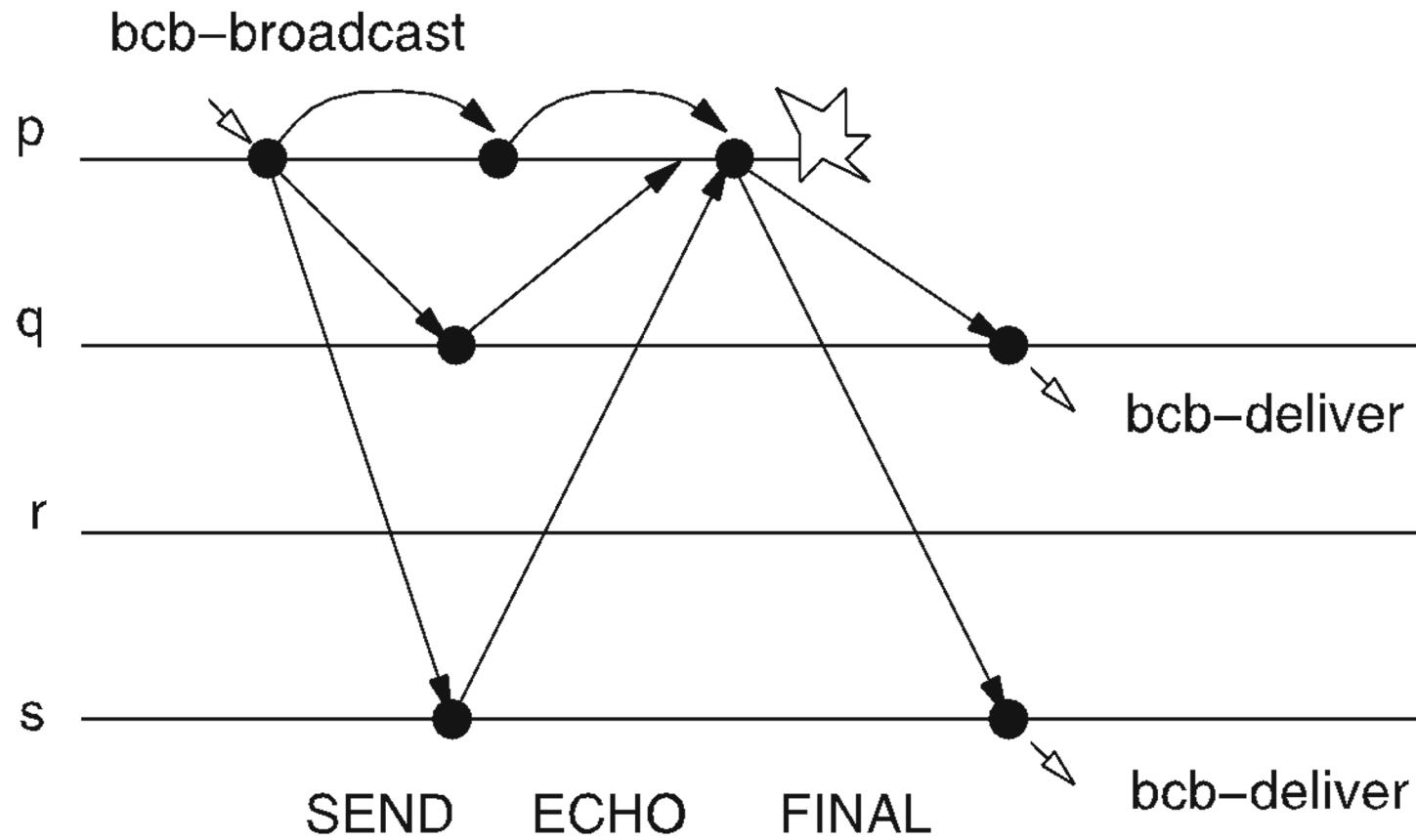
Signed Echo Broadcast

- *// only process s*
upon event $\langle al, \text{Deliver} \mid p, [\text{ECHO}, m, \sigma] \rangle$ **do**
 if $\text{echos}[p] = \perp \wedge \text{verifysig}(p, bcb \parallel p \parallel \text{ECHO} \parallel m, \sigma)$
 then $\text{echos}[p] := m; \Sigma[p] := \sigma;$
- **upon exists** $m \neq \perp$ **such that :**
 $\#\{p \in \Pi \mid \text{echos}[p] = m\} > (N+f)/2$ **and** $\text{sentfinal} =$
 FALSE do
 $\text{sentfinal} := \text{TRUE};$
 forall $q \in \Pi$ **do**
 trigger $\langle al, \text{Send} \mid q, [\text{FINAL}, m, \Sigma] \rangle;$

Signed Echo Broadcast

- **upon event** $\langle al, \text{Deliver} \mid p, [\text{FINAL}, m, \Sigma] \rangle$ **do**
if ($delivered = \text{FALSE}$) **and**
 ($\#\{p \in \Pi \mid \Sigma[p] \neq \perp \wedge$
 $\text{verifysig}(p, bcb \parallel p \parallel \text{ECHO} \parallel m, \Sigma[p])\} > (N+f)/2$)
 $delivered := \text{TRUE};$
trigger $\langle bcb, \text{Deliver} \mid s, m \rangle;$

Signed Echo Broadcast



Correctness

- **BCB1-BCB3:** as in prev. algorithm
- **BCB4: Consistency:** If some correct process delivers a message m and another correct process delivers a message m' , then $m = m'$.
 - The digitally signed echo messages convey the same information as the ECHO message that a process sends directly to all others in the prev. algorithm

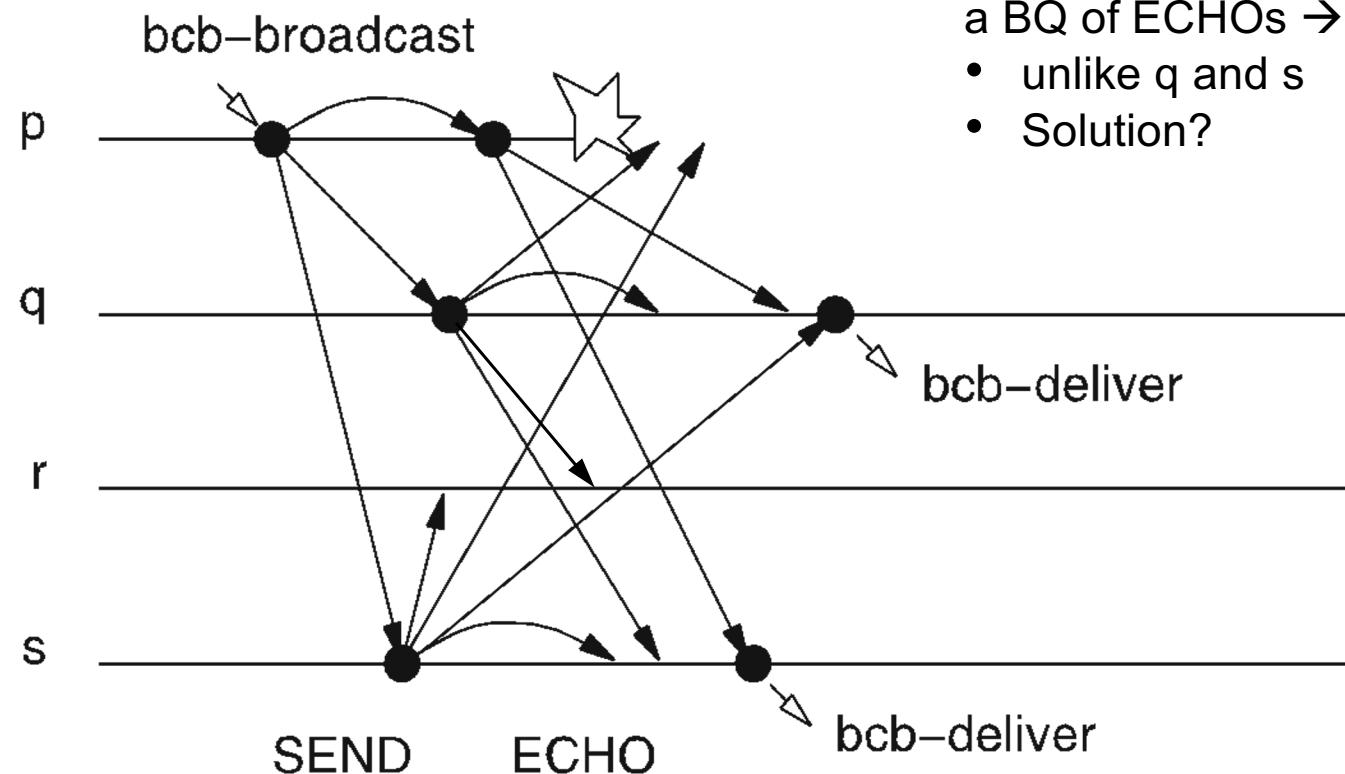
Byzantine Reliable Broadcast: specification

- **BRB1–BRB4** : same as in BCB1-BCB4
- **BRB5:** *Totality*: If some message is delivered by any correct process, every correct process eventually delivers a message.
- **Note**
- Both BCB and BRB are defined for a designated sender, a given message
 - not for a stream of messages, unlike in crash model
- *Totality + Consistency* are equivalent to *Agreement* of Reliable Broadcast in the crash model

Byzantine Reliable Broadcast

- Specification of Byzantine Reliable Broadcast
 - Ensures that if a correct process delivers m , then every correct process delivers m
 - **independently of whether the sender is faulty**
 - Stronger spec than Byzantine Consistent Broadcast
- Algorithm: Authenticated Double-Echo Broadcast
- Extends Authenticated Echo Broadcast. How?

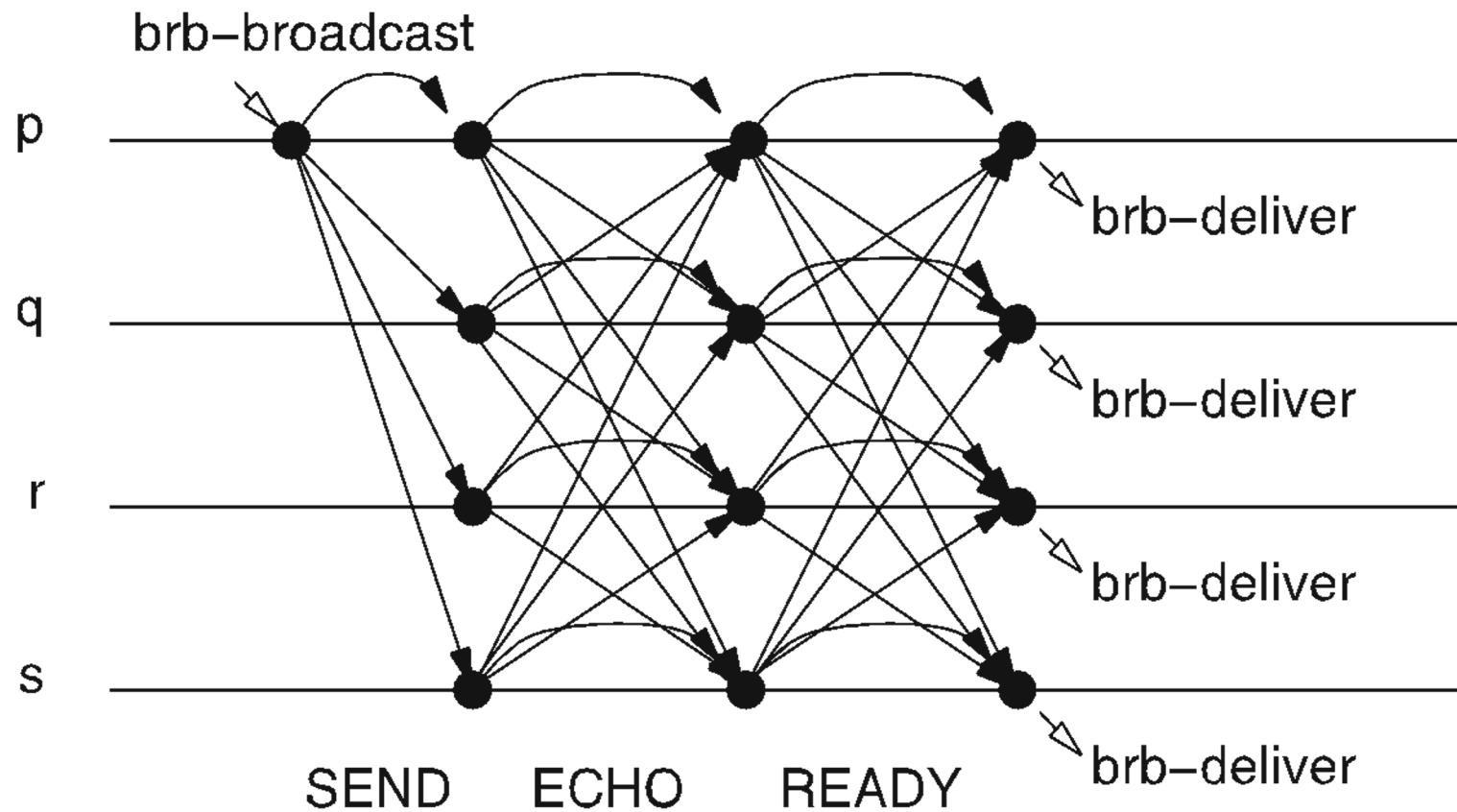
Authenticated Echo Broadcast



PROBLEM:

- process r never gets to receive a BQ of ECHOes → does not deliver m
- unlike q and s
- Solution?

Authenticated Double-Echo Broadcast



Authenticated Double Echo Broadcast

- **Implements:**

- ByzantineReliableBroadcast, with sender s .

- **Uses:**

- AuthPerfectPointToPointLinks, **instance** al .

- **upon event** $\langle brb, \text{Init} \rangle$ **do**

- $sentecho := \text{FALSE};$

- $sentready := \text{FALSE};$

- $delivered := \text{FALSE};$

- $echos := [\perp]^N;$

- $readys := [\perp]^N;$

Authenticated Double Echo Broadcast

- **upon event** $\langle brb, \text{Broadcast} \mid m \rangle$ **do** // only process s
forall $q \in \Pi$ **do**
 trigger $\langle al, \text{Send} \mid q, [\text{SEND}, m] \rangle$;
- **upon event** $\langle al, \text{Deliver} \mid p, [\text{SEND}, m] \rangle$
 such that $p = s$ and $\text{sentecho} = \text{FALSE}$ **do**
 $\text{sentecho} := \text{TRUE}$;
forall $q \in \Pi$ **do**
 trigger $\langle al, \text{Send} \mid q, [\text{ECHO}, m] \rangle$;

Authenticated Double Echo Broadcast

- **upon event** $\langle al, \text{Deliver} | p, [\text{ECHO}, m] \rangle$ **do**
 if $\text{echos}[p] = \perp$ **then** $\text{echos}[p] := m;$
- **upon exists** $m \neq \perp$ **such that :**
 $|\{p \in \Pi \mid \text{echos}[p] = m\}| > (N+f)/2$ **and** $\text{sentready}=\text{FALSE}$ **do**
 $\text{sentready}:= \text{TRUE}$
 forall $q \in \Pi$ **do**
 trigger $\langle al, \text{Send} | q, [\text{READY}, m] \rangle;$
- **upon event** $\langle al, \text{Deliver} | p, [\text{READY}, m] \rangle$ **do**
 if $\text{readys}[p] = \perp$ **then** $\text{readys}[p] := m;$

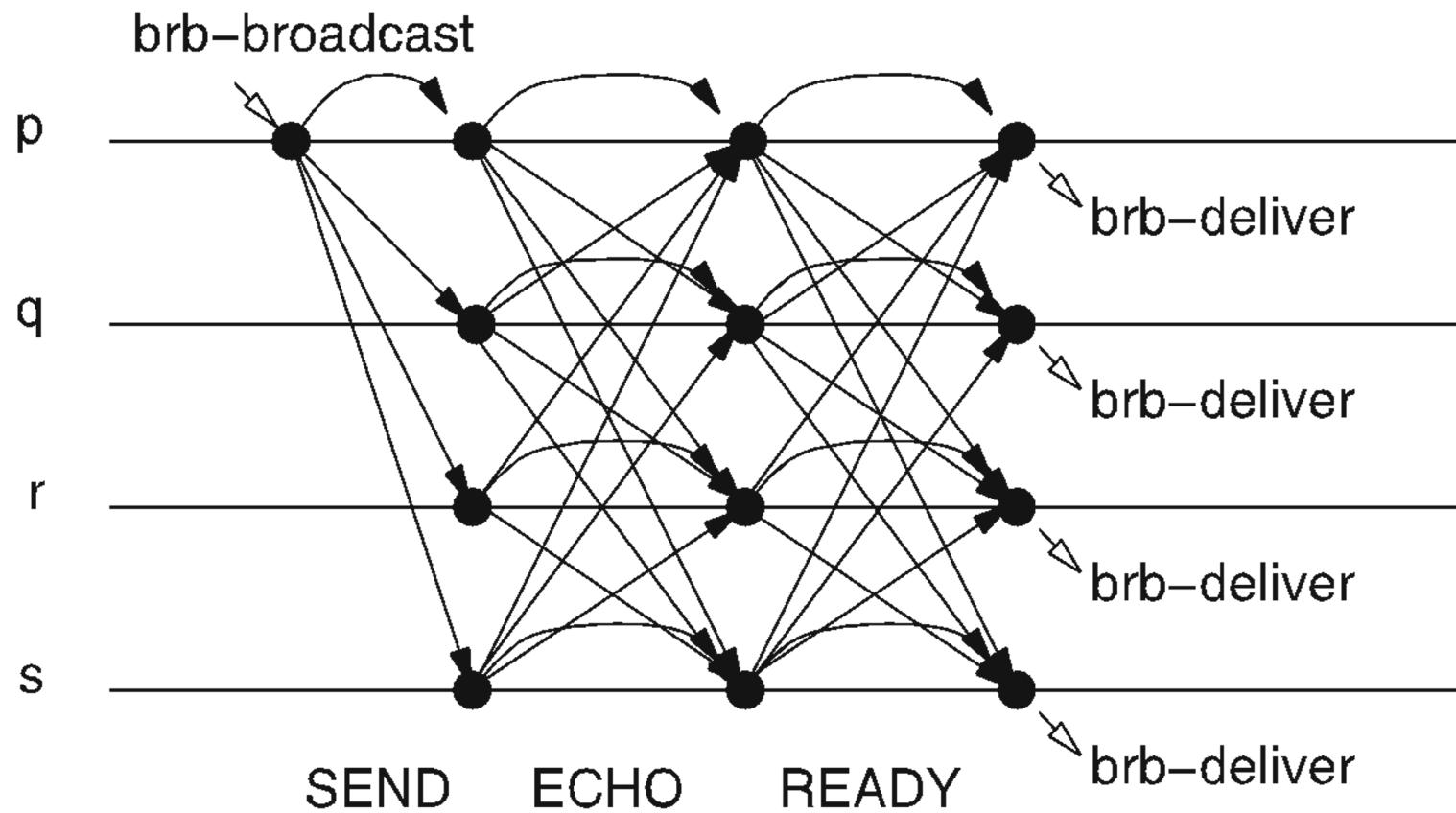
Retransmission step:
inform other correct
processes about
existence of byzantine
quorum for m

Authenticated Double Echo Broadcast

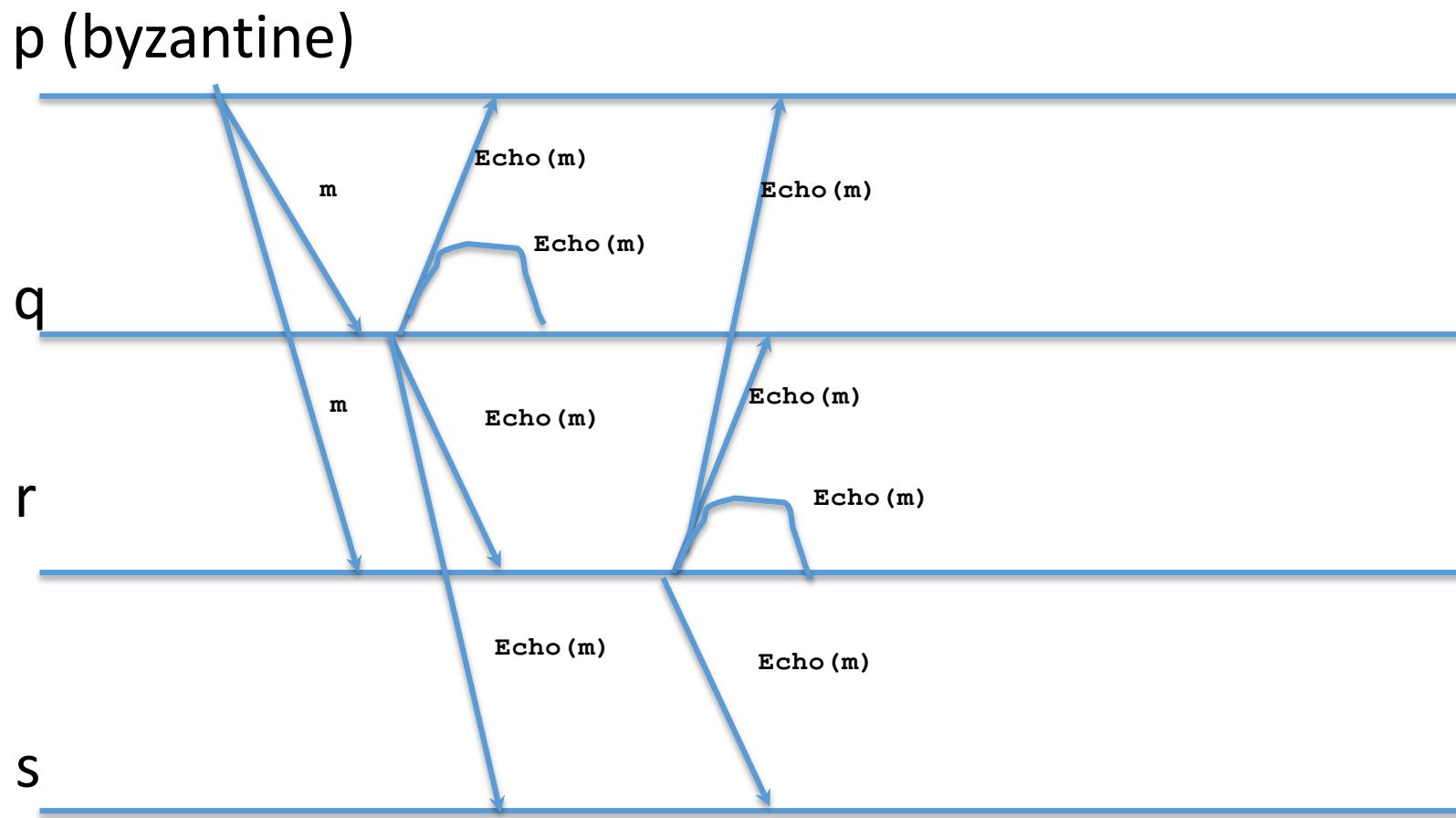
Amplification step:
Why is it needed?

- **upon exists** $m \neq \perp$ such that
 $|\{p \in \Pi | \text{readys}[p] = m\}| > f$ **and** $\text{sentready} = \text{FALSE}$ **do**
 $\text{sentready} := \text{TRUE};$
forall $q \in \Pi$ **do**
trigger $\langle al, \text{Send} | q, [\text{READY}, m] \rangle;$
- **upon exists** $m \neq \perp$ **such that**
 $|\{p \in \Pi | \text{readys}[p] = m\}| > 2f$ **and** $\text{delivered} = \text{FALSE}$ **do**
 $\text{delivered} := \text{TRUE};$
trigger $\langle brb, \text{Deliver} | s, m \rangle;$

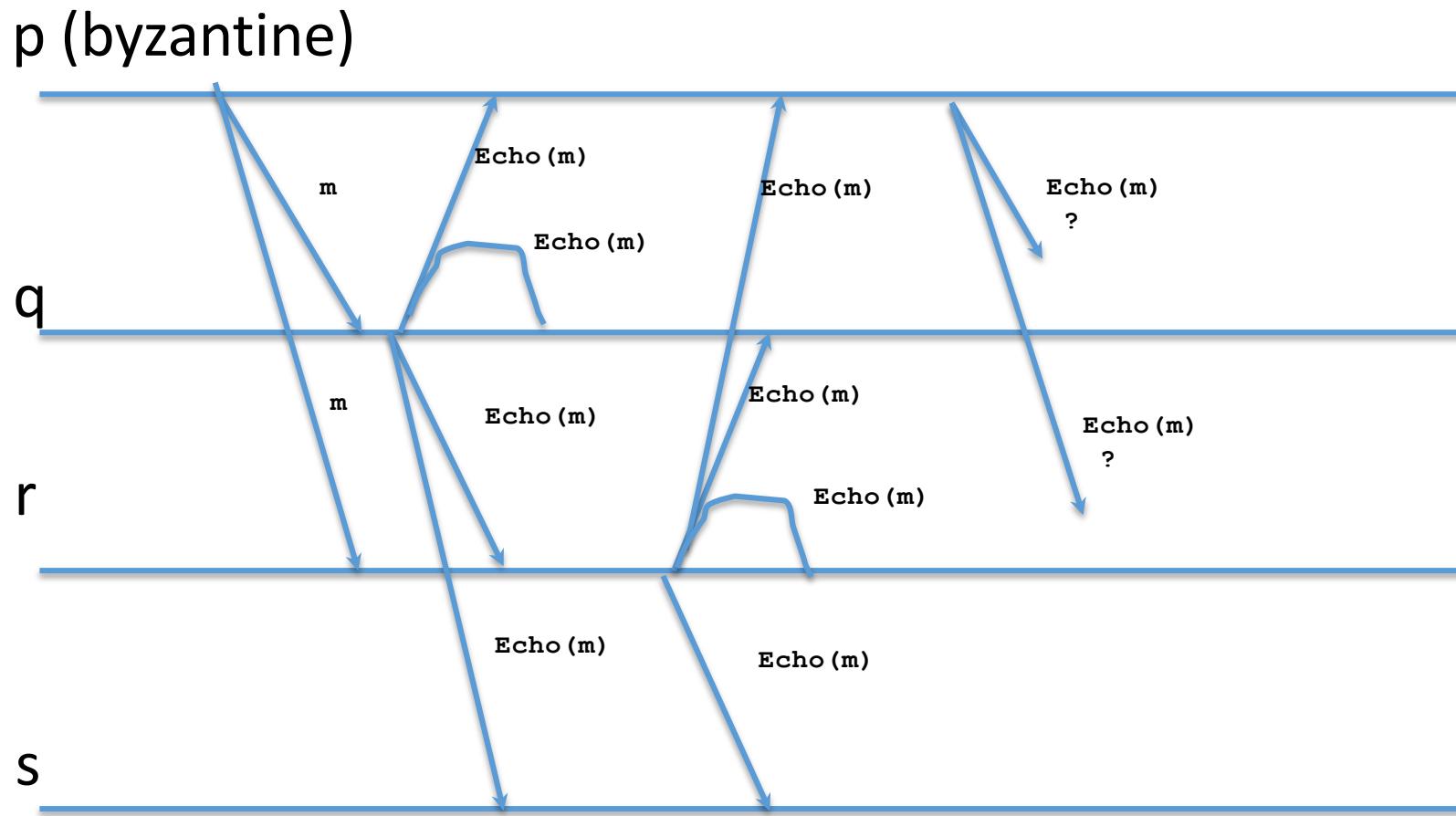
Authenticated Double-Echo Broadcast



Is it possible that only q or only r send a ready msg?

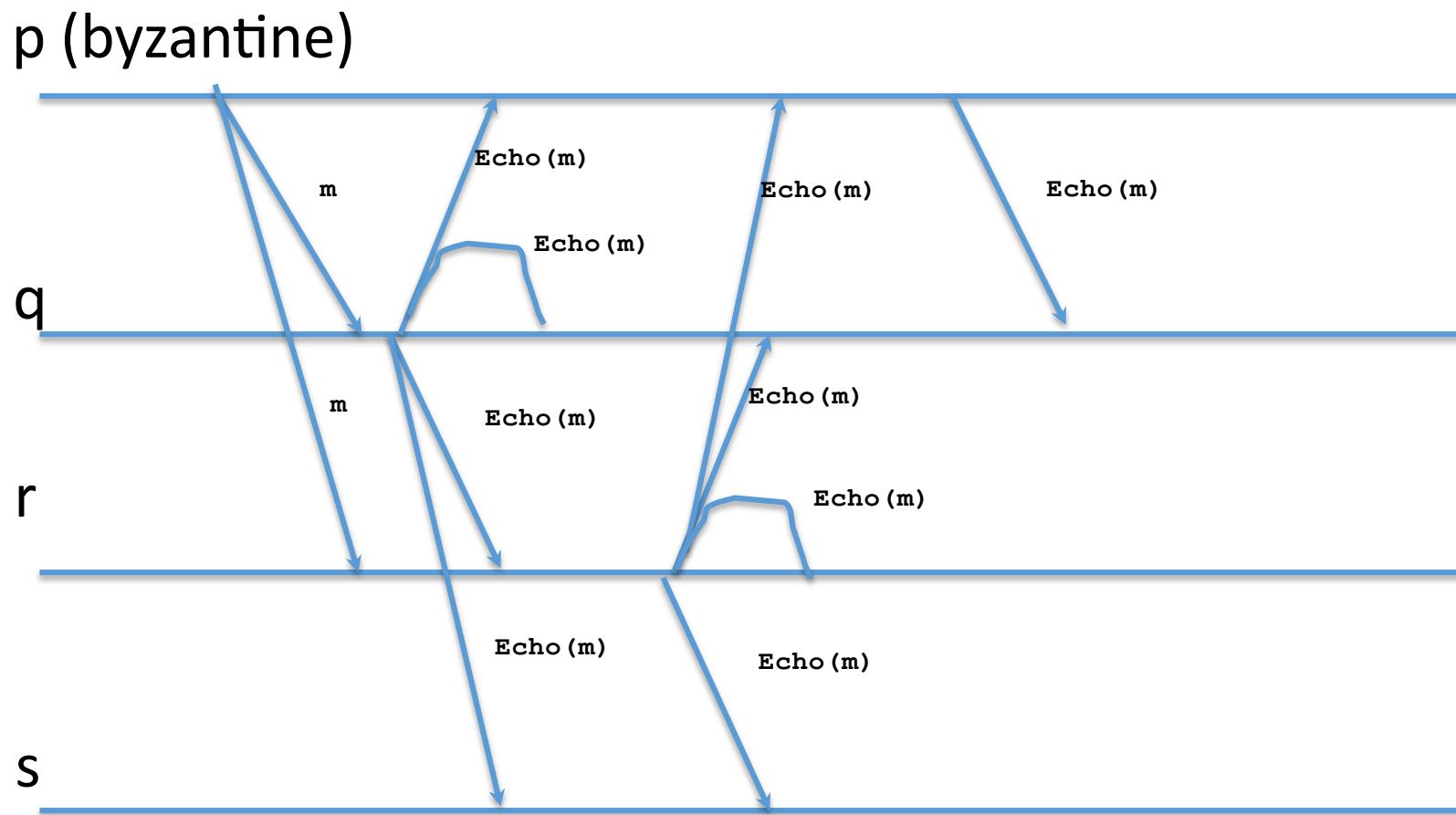


Is it possible that only q or r send a ready msg?



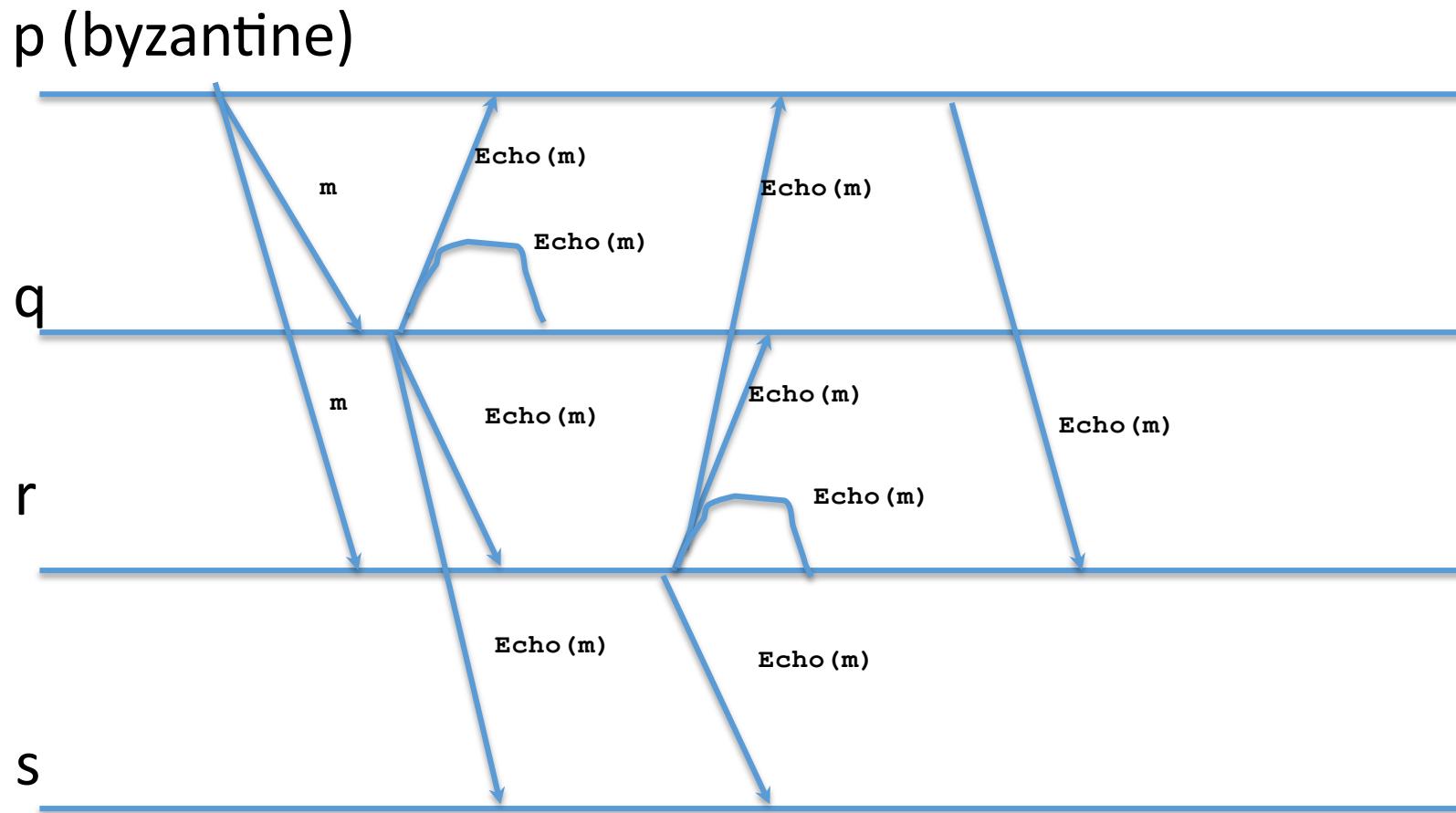
Yes, depending on whether p sends an echo to q and/or r

Is it possible that only q or r send a ready msg?



Yes, depending on whether p sends an echo to q and/or r

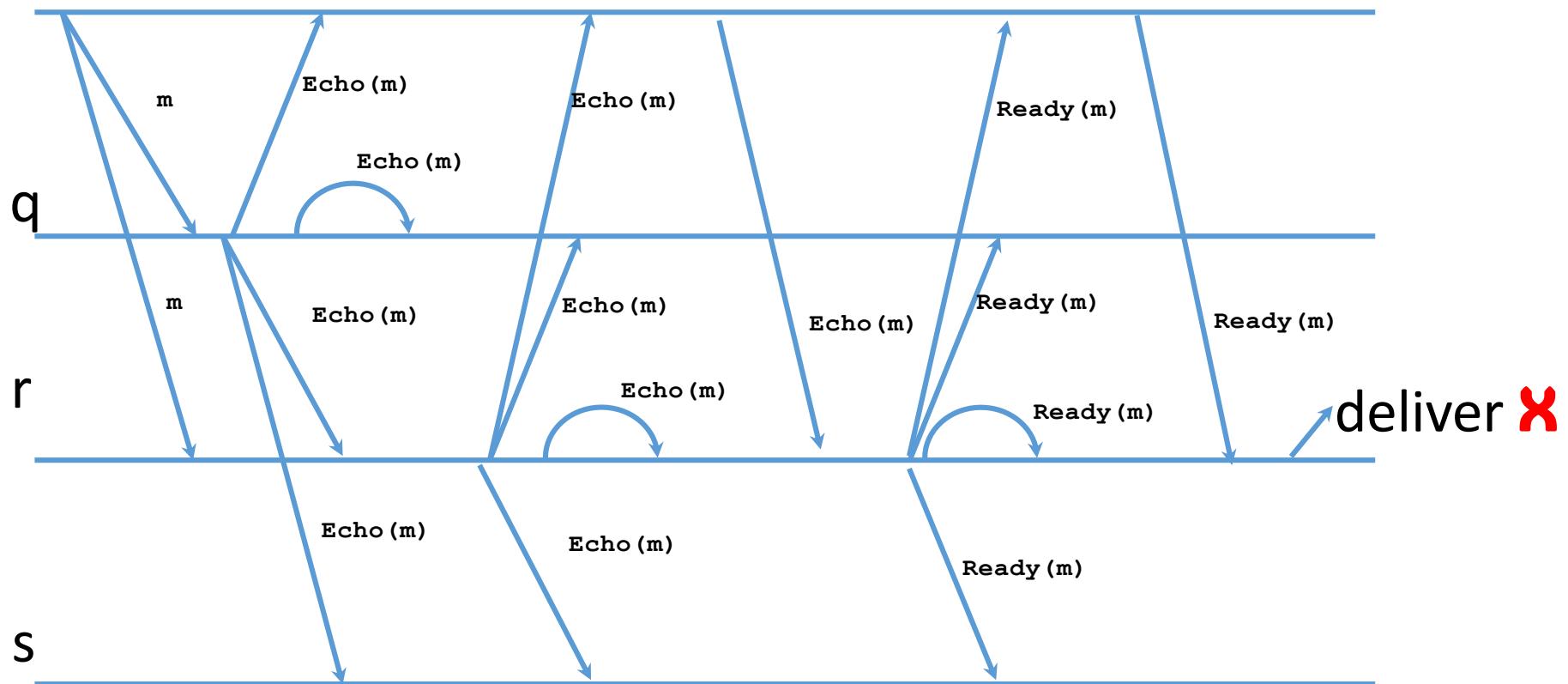
Is it possible that only q or r send a ready msg?



Yes, depending on whether p sends an echo to q and/or r

Is it ok to deliver after $f+1$ Ready msgs?

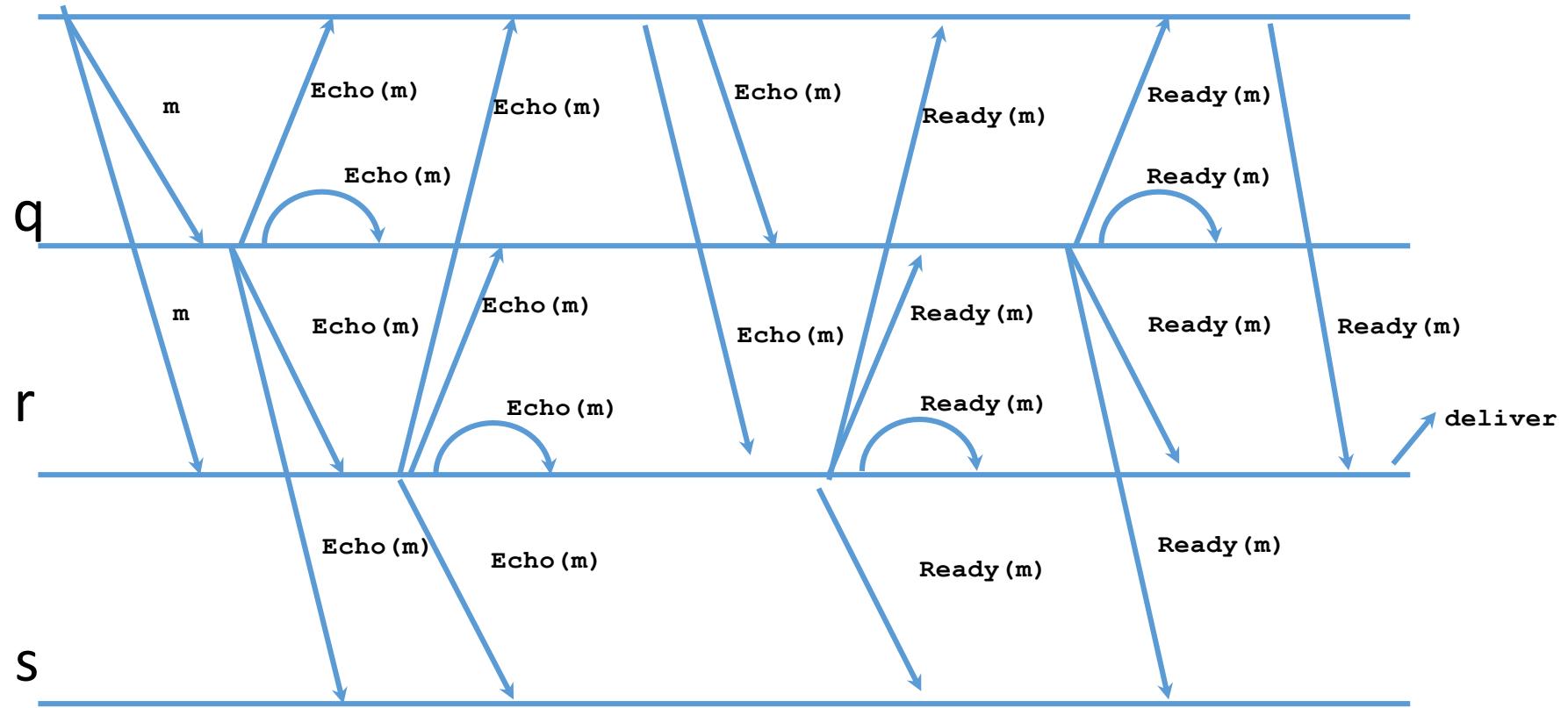
p (byzantine)



Need to wait for $2f+1$ Ready msgs to secure totality

Why is the amplification phase needed?

p (byzantine)



- r receives 3 Ready(m) msgs and delivers m ...
- but q does not, as p does not send Ready(m) to q

Correctness (1/3)

- Validity, No duplication, Integrity:
 - same as in Authenticated Echo Broadcast

Correctness (2/3)

- Consistency: If some correct process delivers a message m and another correct process delivers a message m' , then $m = m'$.
 - if some correct process broadcasts ready messages for m and another correct process broadcasts ready for m' , then $m=m'$
 - Proof by contradiction, same as in previous proof
 - Directly implies consistency because cannot have more than f ready messages for different messages

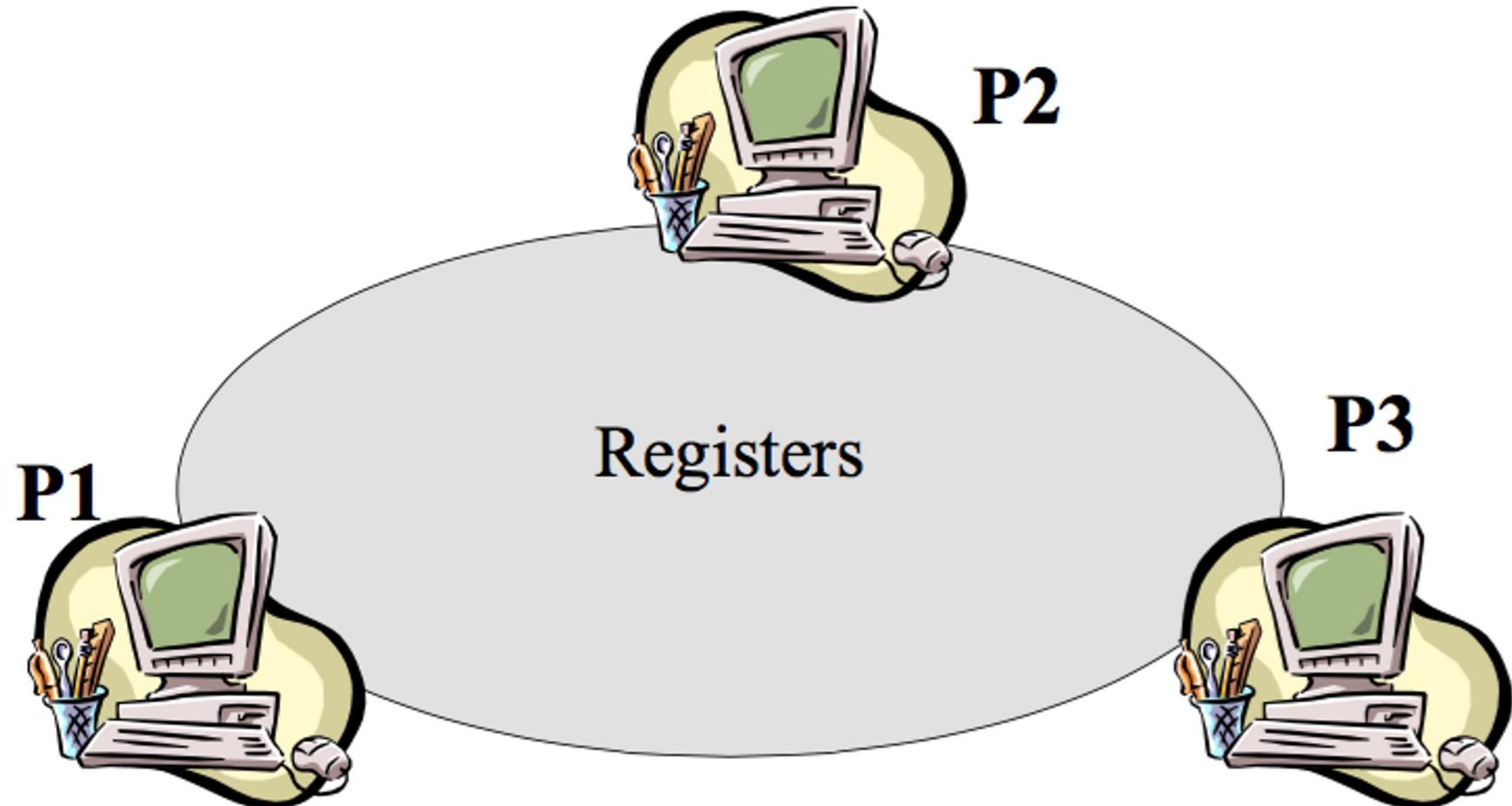
Correctness (3/3)

- Totality:
 - if a correct process delivers, then it received at least $2f+1$ ready messages, at least $f+1$ from correct process.
 - These $f+1$ will eventually be delivered to all correct processes, triggering amplification step, and consequently sufficient ready messages for totality

Byzantine broadcast channels

- Byzantine broadcast works only for a single message
- It is simple to implement a broadcast channel
 - Either consistent or reliable broadcast channels
- Idea: use multiple instances of Byzantine broadcast

New abstraction: read/write registers



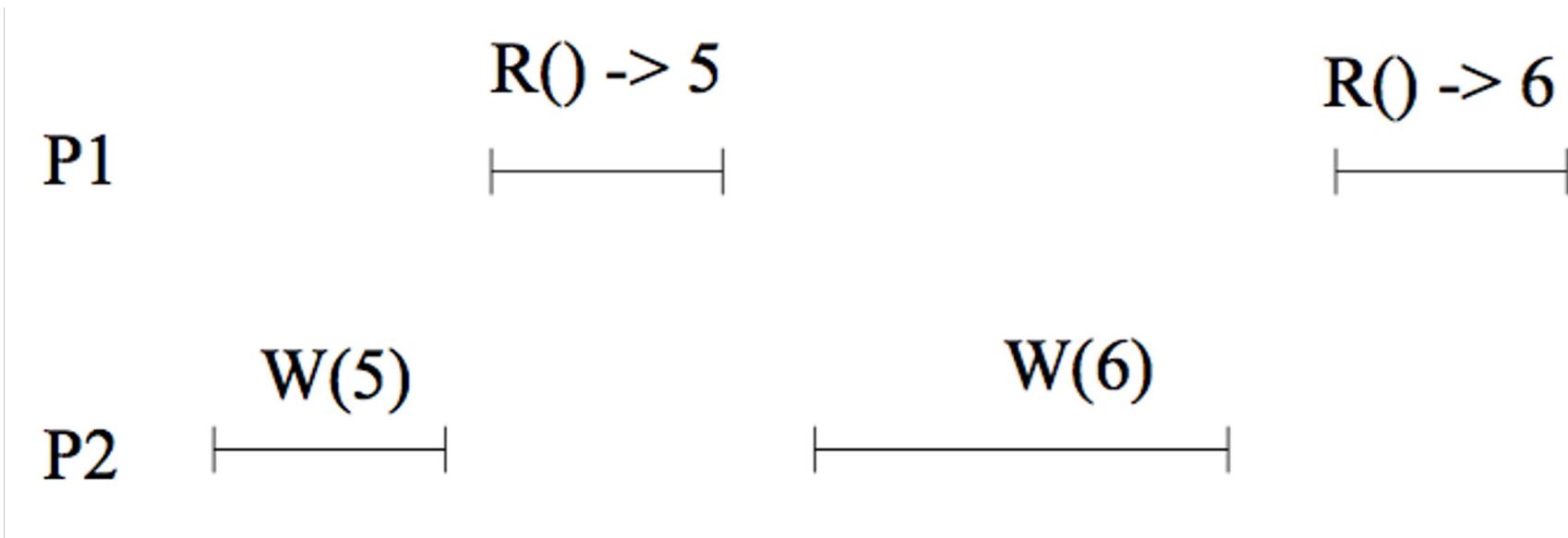
P1,P2,P3 are both clients and replicas (without loss of generality)

Assumptions

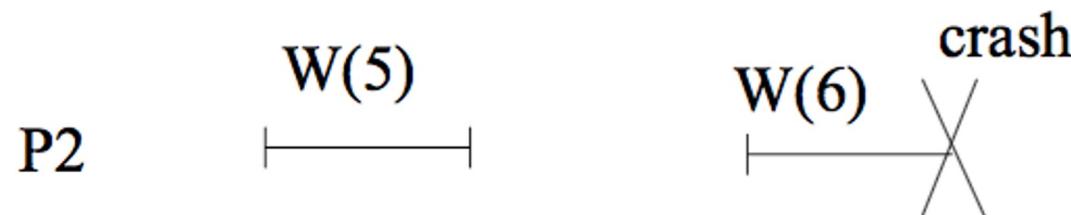
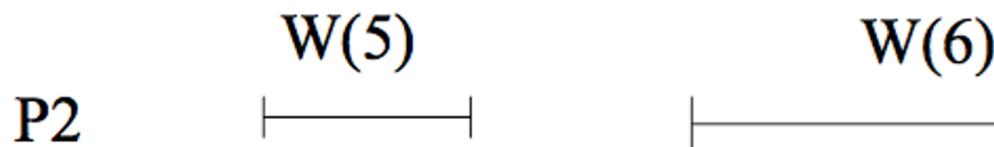
- Single register, shared by all processes
- Interface:
 - `Read()` → returns `value`
 - `Write(value)` → returns "ack"
- Values written are integers
- Initial value of register is zero
- Every written value is unique
 - this can be enforced by associating a process id and a timestamp with the value

Sequential specification

- **Read()** returns the most recent **value** that was written



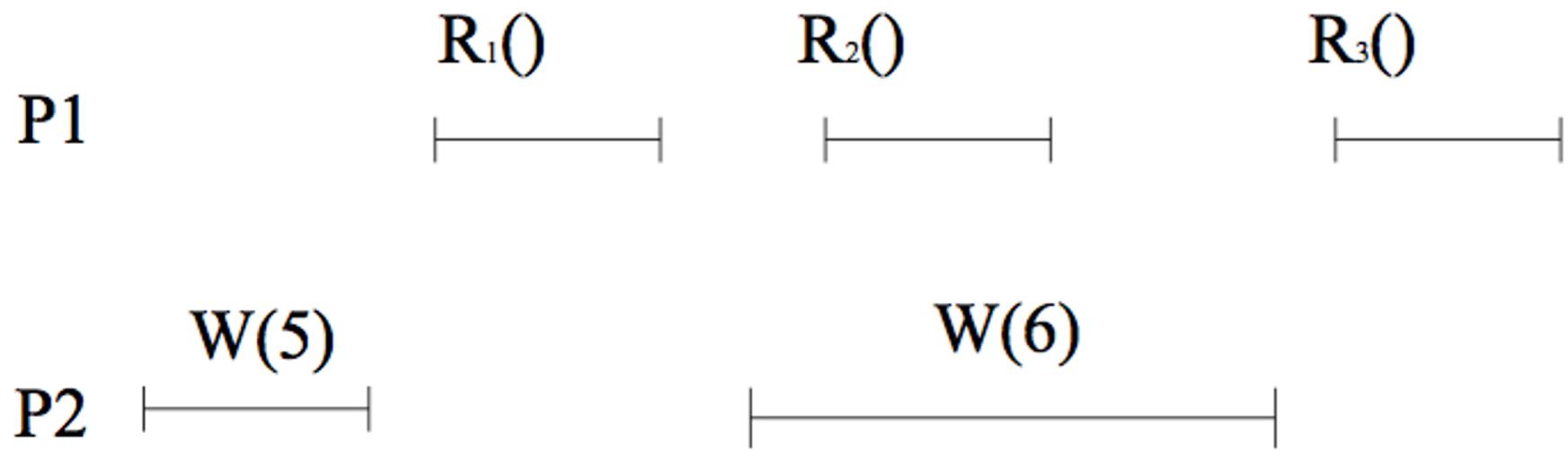
How to specify concurrent / faulty execution?



Specification #1: Regular register

- Initially, we will assume a single writer
- [Liveness] If a correct process invokes an operation, then the operation eventually completes.
- [Safety] Read must return:
 - the last value written if there is no concurrent write operation, otherwise
 - either the last value or any value concurrently written
- When a process crashes in the middle of reading/writing, the operation interval does not have an upper limit

Exercise: which values are admissible for R₁, R₂, R₃?



Exercise: which values are admissible?

P1

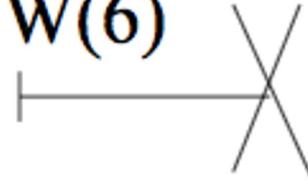
$R(0) \rightarrow ?$



P2

$W(5)$

$W(6)$ crash



(On a side note, there exists a weaker specification: safe register)

- Similar to regular register, but:
- [Safety] Read must return:
 - the last value written if there is no concurrent write operation, otherwise
 - **any** value if there are concurrent writes
- In this course we will skip the design of safe registers

Implementation of a (1,N)-Regular register

- Uses BestEffortBroadcast + Perfect-p2p-links
- Each process maintains:

```
<ts,val>    /* Current value and associated timestamp, ts */
readlist      /* List of returned values, for reading */
rid          /* id of current read operation */
```
- Writer process maintains:

```
wts        /* Next timestamp to be written */
acks       /* How many writes have been acknowledged */
```
- Algorithm must tolerate up to f crash faults. How?

Implements:

(1, N)-RegularRegister, **instance** $onrr$.

Uses:

BestEffortBroadcast, **instance** beb ;
PerfectPointToPointLinks, **instance** pl .

upon event $\langle onrr, Init \rangle$ **do**

$(ts, val) := (0, \perp)$;
 $wts := 0$;
 $acks := 0$;
 $rid := 0$;
 $readlist := [\perp]^N$;

upon event $\langle onrr, Write \mid v \rangle$ **do**

$wts := wts + 1$;
 $acks := 0$;
trigger $\langle beb, Broadcast \mid [WRITE, wts, v] \rangle$;

upon event $\langle beb, Deliver \mid p, [WRITE, ts', v'] \rangle$ **do**

if $ts' > ts$ **then**
 $(ts, val) := (ts', v')$;
trigger $\langle pl, Send \mid p, [ACK, ts'] \rangle$;

upon event $\langle pl, Deliver \mid q, [ACK, ts'] \rangle$ **such that** $ts' = wts$ **do**

$acks := acks + 1$;
if $acks > N/2$ **then**
 $acks := 0$;
trigger $\langle onrr, WriteReturn \rangle$;

```

upon event < onrr, Read > do
    rid := rid + 1;
    readlist := [⊥]N;
    trigger < beb, Broadcast | [READ, rid] >;
    
upon event < beb, Deliver | p, [READ, r] > do
    trigger < pl, Send | p, [VALUE, r, ts, val] >;
    
upon event < pl, Deliver | q, [VALUE, r, ts', v'] > such that r = rid do
    readlist[q] := (ts', v');
    if #(readlist) > N/2 then
        v := highestval(readlist);
        readlist := [⊥]N;
    trigger < onrr, ReadReturn | v >;

```

Is this execution valid?



Yes, reads might return the last value written of the value concurrently being written

Homework: come up with an execution that leads to this trace of outputs

Acknowledgements

- Rachid Guerraoui, EPFL