TÉCNICO LISBOA

Towards a dependable consensus: Broadcast primitives

Highly dependable systems

Lecture 3

Lecturers: Miguel Matos and Rodrigo Miragaia Rodrigues

Last lecture: safety and liveness

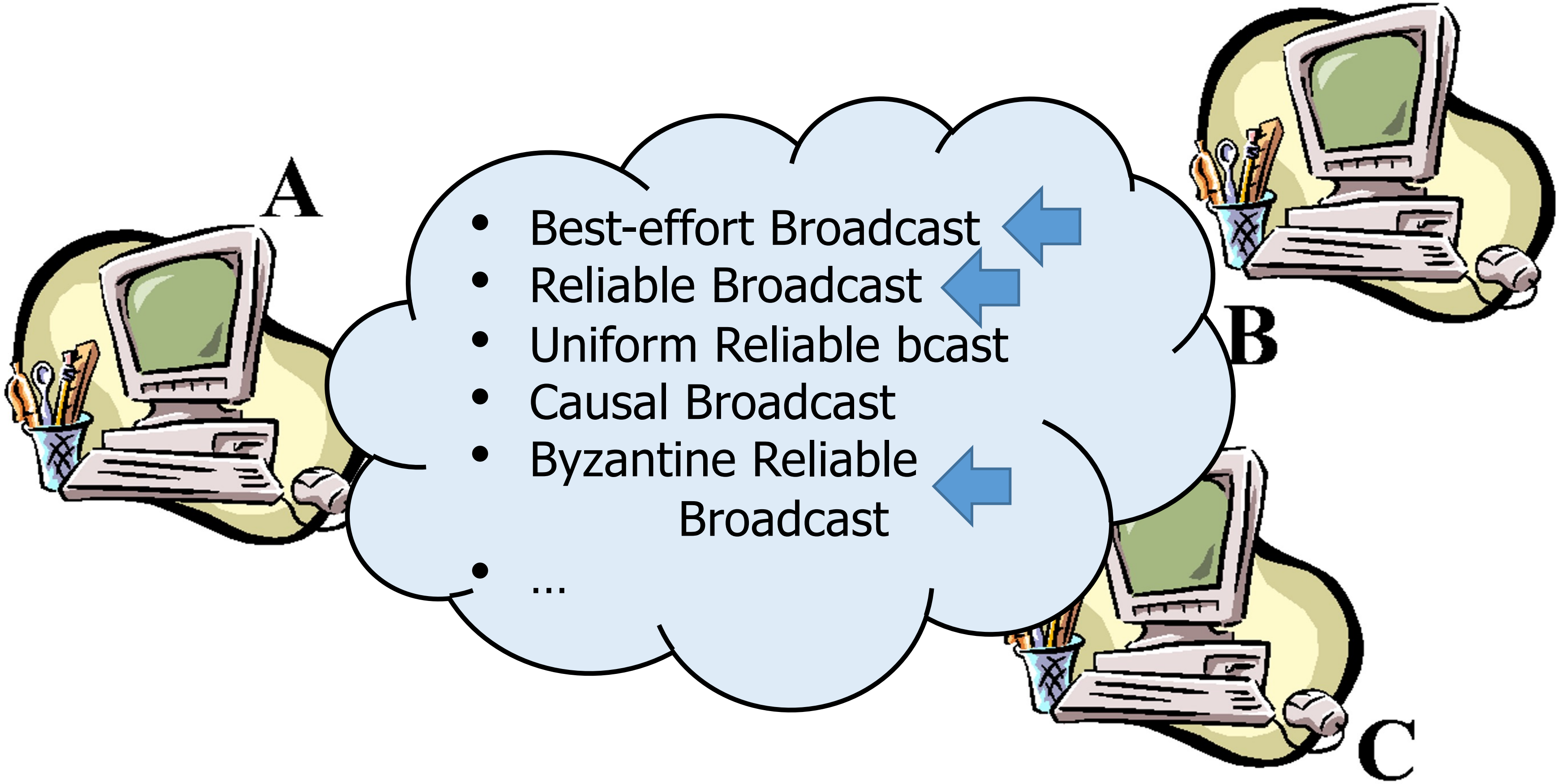
- Given a trace (sequence of outputs) of a distributed system
  - A safety property obeys:
    - If a finite trace does not obey the property, no extension of that trace obeys that property
  - A liveness property obeys:
    - A finite trace that does not obey the property can be extended so that the liveness property is upheld
- Any specification can be expressed in terms of liveness and safety properties

Last lecture - Byzantine Leader Election

- Properties
  - **Eventual succession**: if more than f correct processes that trust some process p complain about p, then every correct process eventually trusts a different process than p
  - **Putsch resistance:** A correct process does not trust a new leader unless at least one correct process has complained against the previous leader
  - **Eventual agreement**: there is a time after which no two correct processes trust different processes

- Eventually every <u>correct</u> process trusts some process that appears to perform its task in the higher-level algorithm.

Broadcast Abstractions



Intuition

- Broadcast is useful per se, for instance, in applications where some processes subscribe to events published by other processes (e.g., stocks)
- But also, a crucial building blocks in other protocols, namely consensus
- The receivers might require some **reliability** guarantees from the broadcast service (we say sometimes quality of service QoS) that the underlying network does not provide

Overview

We shall consider three forms of reliability for a broadcast primitive

- **(1) Best-effort broadcast**
- **(2) Reliable broadcast in the crash fault model**
- **(3) Reliable broadcast in the arbitrary fault model**

- We provide **specifications** and then **algorithms**

Best-effort Broadcast (beb)

- **Events**

  Request: <bebBroadcast, m>

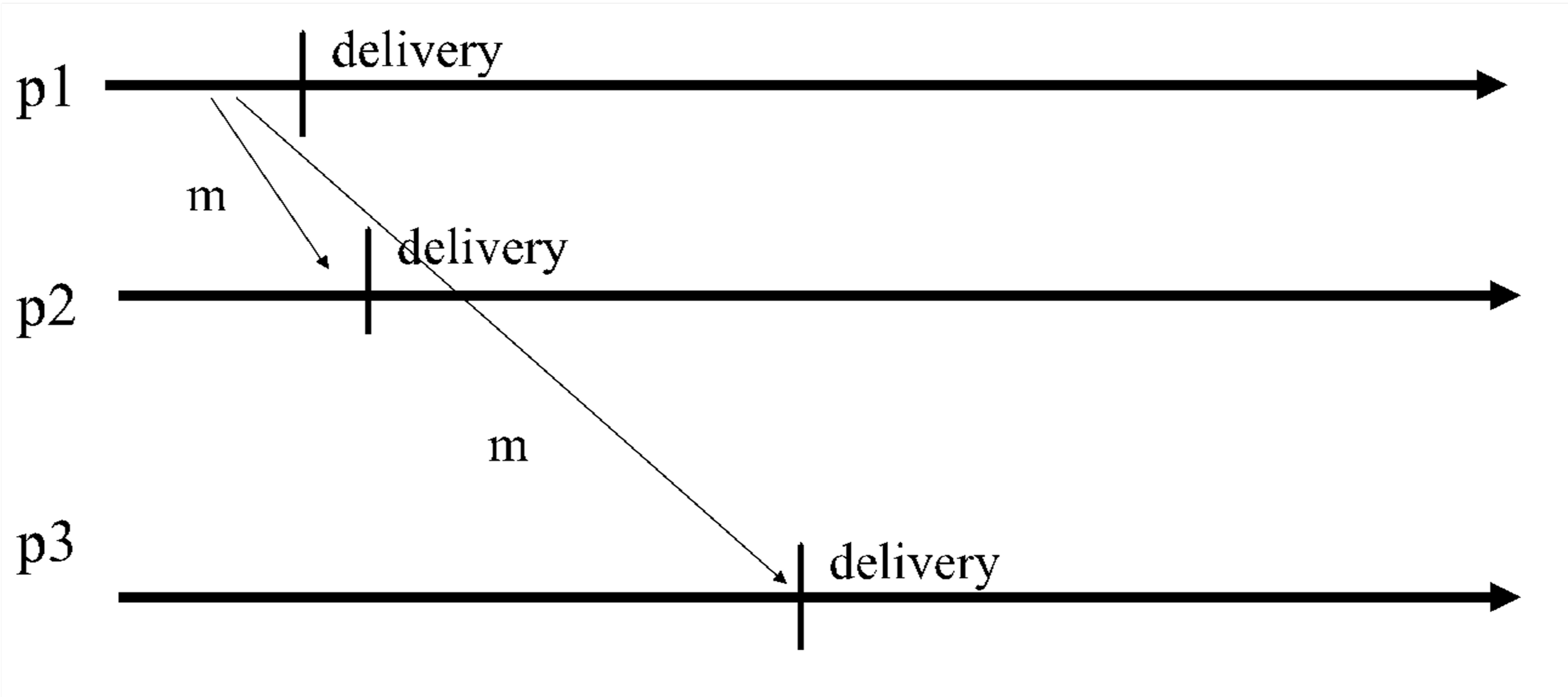
  Indication: <bebDeliver, src, m>

- **Properties: BEB1, BEB2, BEB3**

Best-effort broadcast (beb)

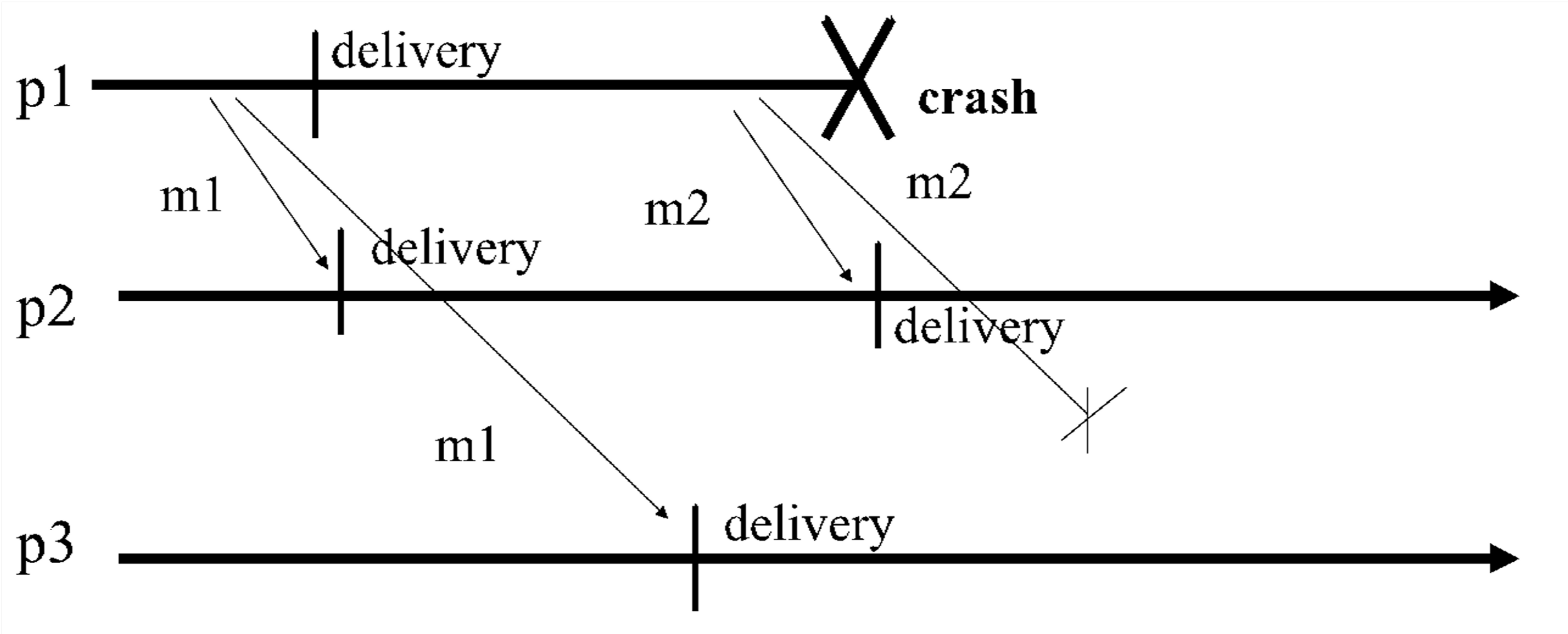
- **Properties**
  - **BEB1. Validity**: If pi and pj are correct, then every message broadcast by pi is eventually delivered by pj
  - **BEB2. No duplication:** No message is delivered more than once
  - **BEB3. No creation:** No message is delivered unless it was broadcast

Best-effort Broadcast



Meets the specification

Best-effort Broadcast



Meets the specification

Overview

We shall consider three forms of reliability for a broadcast primitive

- **(1) Best-effort broadcast**
- **(2) Reliable broadcast in the crash fault model**
- **(3) Reliable broadcast in the arbitrary fault model**

- We provide **specifications** and then **algorithms**

Algorithm (beb)

- **Implements:** BestEffortBroadcast (beb).
- **Uses:** PerfectLinks (pp2p).

- **upon event** < bebBroadcast, m> **do**
  - **for all** pi∈S **do**
    - **trigger** < pp2pSend, pi, m>;

- **upon event** < pp2pDeliver, pi, m> **do**
  - **trigger** < bebDeliver, pi, m>;

Algorithm (beb)

- **Proof (sketch)**
  - **BEB1. Validity**: By the validity property of perfect links and the very facts that (1) the sender sends the message to all and (2) every correct process that pp2pDelivers a message bebDelivers it
  - **BEB2. No duplication:** By the no duplication property of perfect links
  - **BEB3. No creation:** By the no creation property of perfect links

Overview

We shall consider three forms of reliability for a broadcast primitive

- **(1) Best-effort broadcast**
- **(2) Reliable broadcast in the crash fault model**
- **(3) Reliable broadcast in the arbitrary fault model**

- We provide **specifications** and then **algorithms**

Reliable Broadcast (rb)

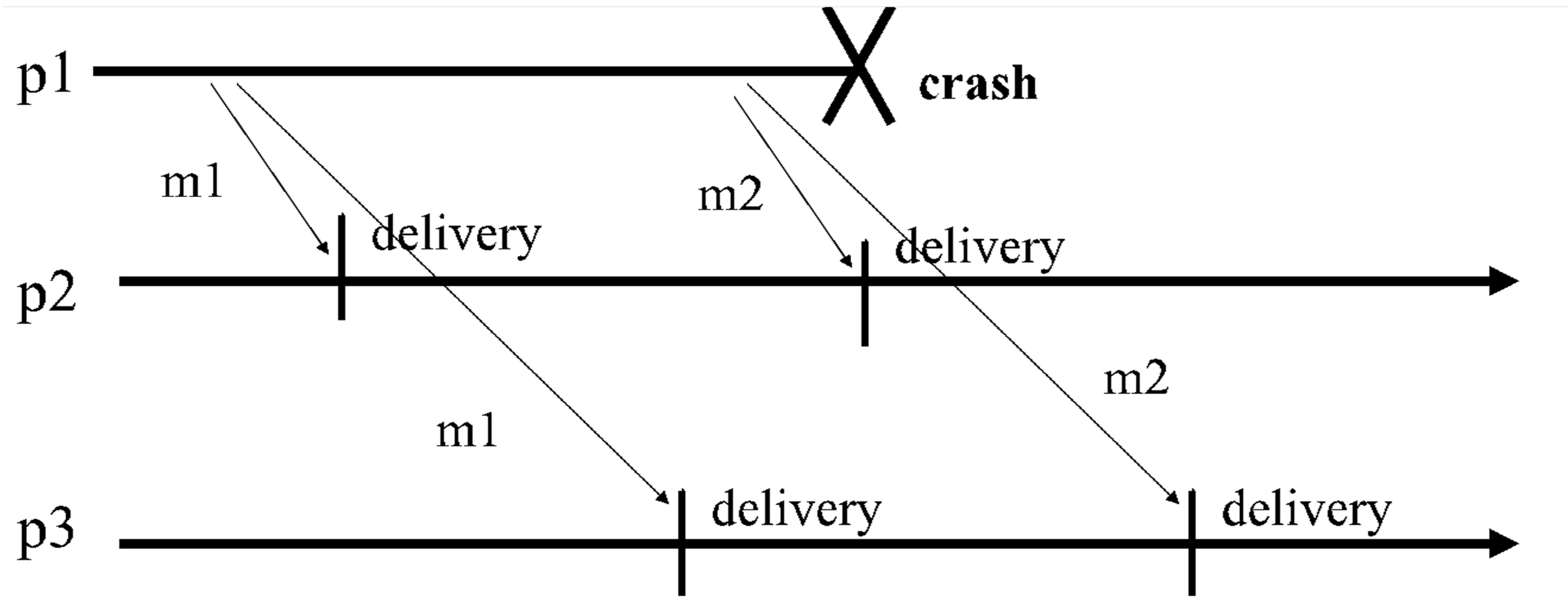
- **Events**
  - Request: <rbBroadcast, m>
  - Indication: <rbDeliver, src, m>

- **Properties: RB1, RB2, RB3, RB4**

Reliable broadcast

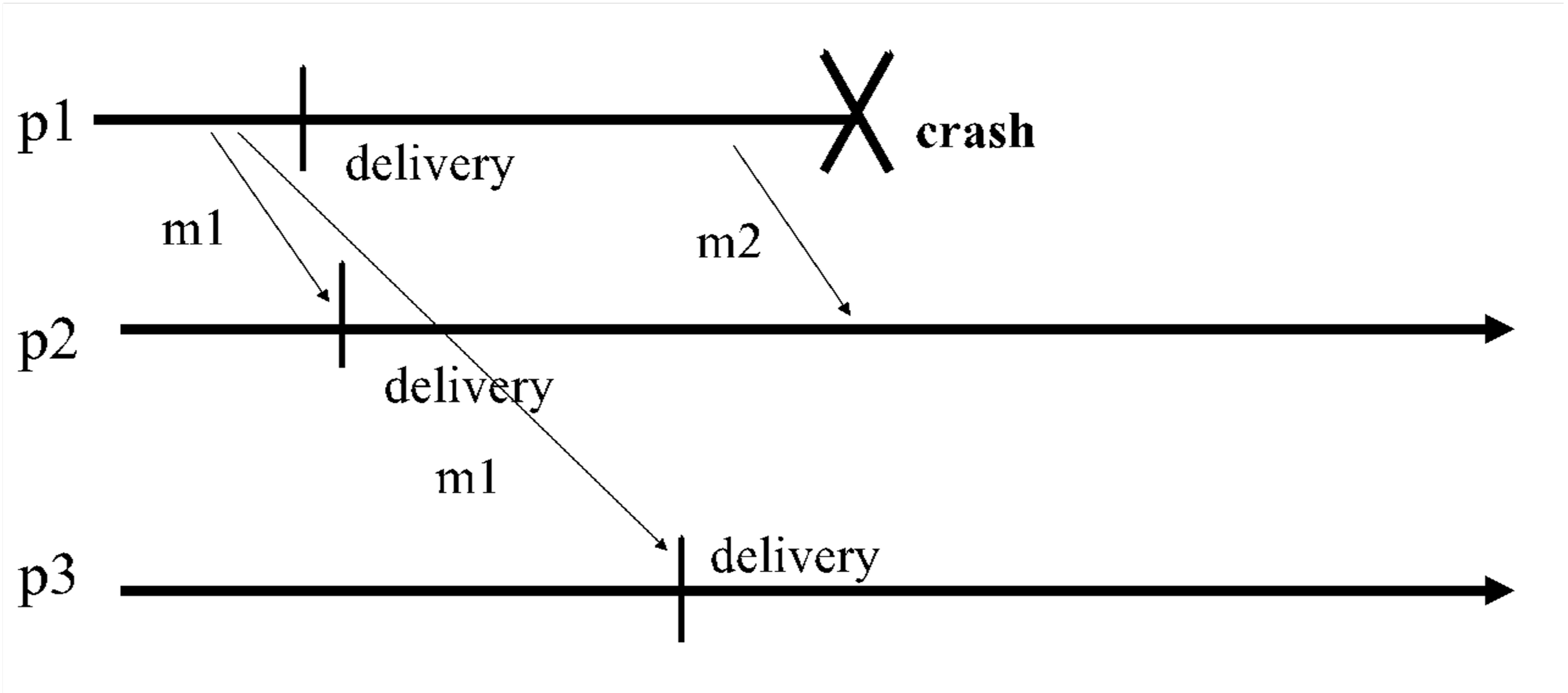
- **Properties**
  - **RB1 = BEB1.**
  - **RB2 = BEB2.**
  - **RB3 = BEB3.**
  - **RB4. Agreement:** For any message m, if a correct process delivers m, then every correct process delivers m

Reliable Broadcast



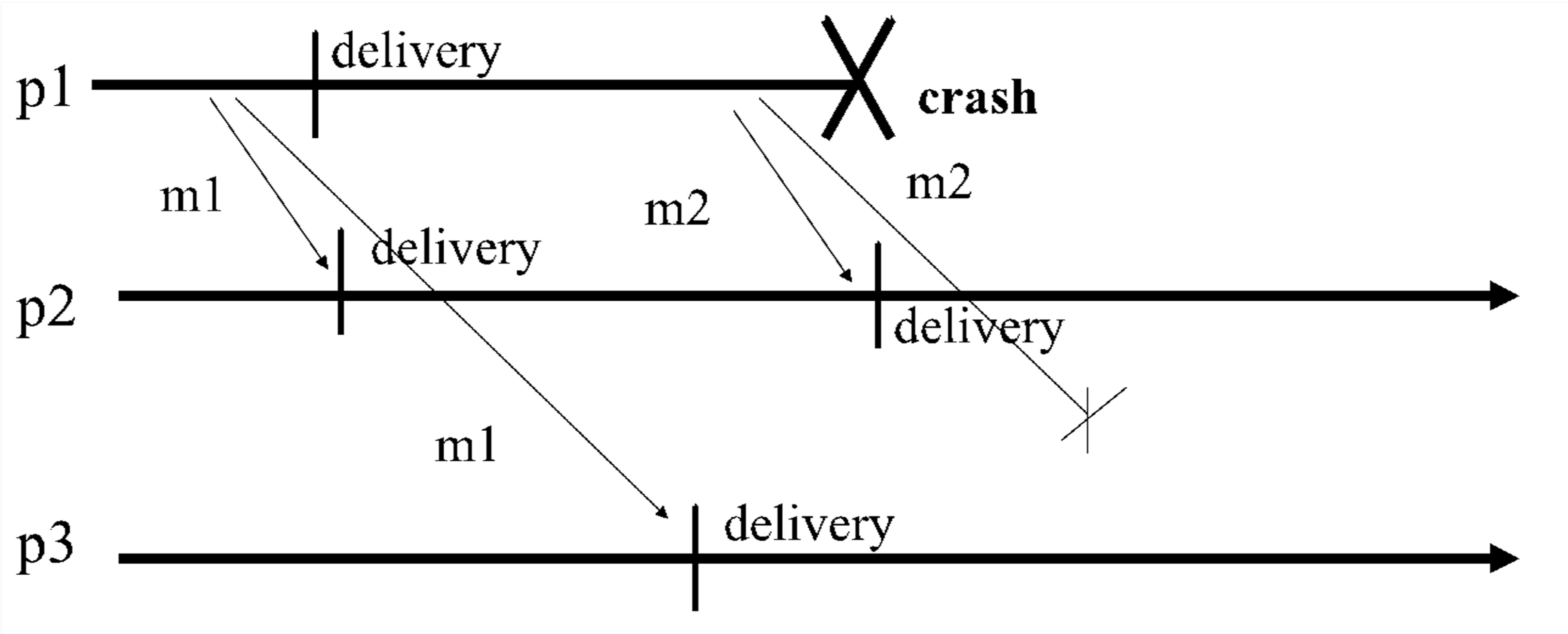
Meets the specification

Reliable Broadcast



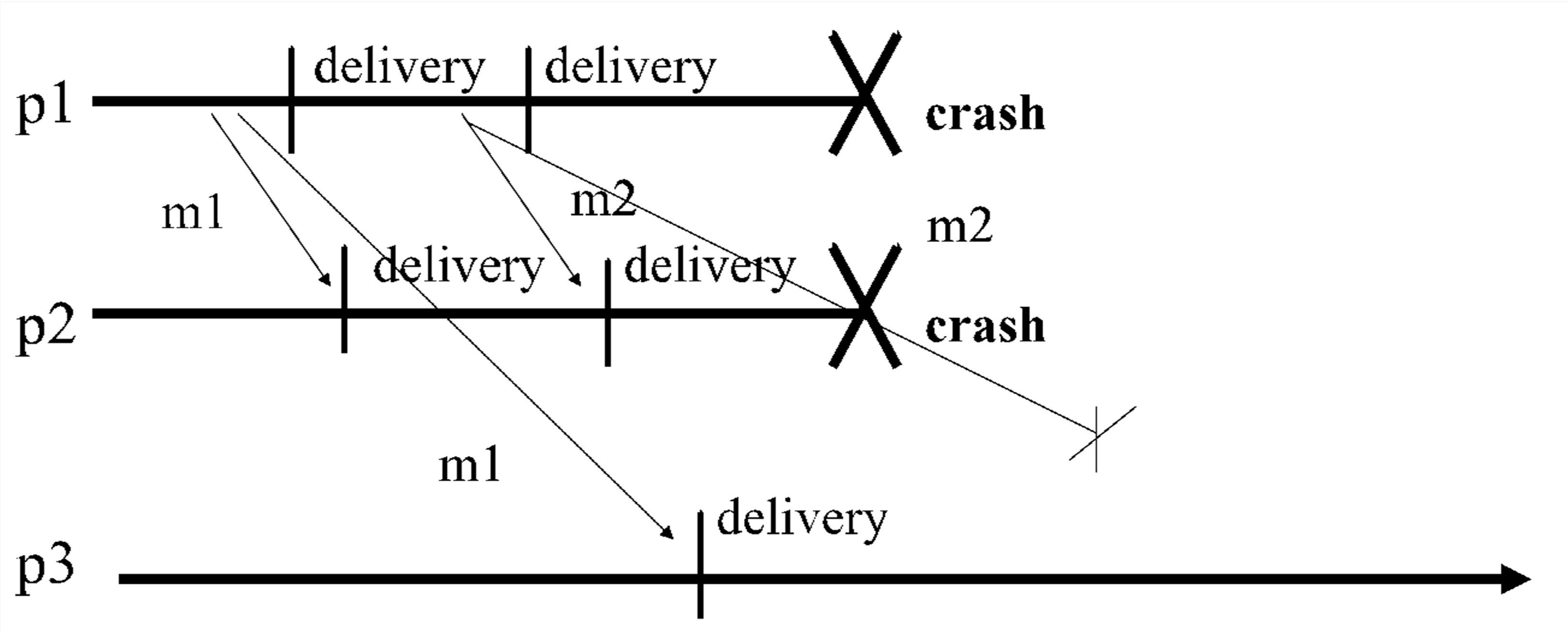
Meets the specification

Reliable Broadcast



Does not meet the specification

Reliable Broadcast



Meets the specification

Overview

We shall consider three forms of reliability for a broadcast primitive

- **(1) Best-effort broadcast**
- **(2) Reliable broadcast in the crash fault model**
- **(3) Reliable broadcast in the arbitrary fault model**

- We provide **specifications** and then **algorithms**

Eager Reliable Broadcast

Implements:

ReliableBroadcast, **instance** *rb*.

Uses:

BestEffortBroadcast, **instance** *beb*.

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

$delivered := \emptyset$;

upon event $\langle$ *rb*, *Broadcast* | m $\rangle$ **do**

trigger $\langle$ *beb*, *Broadcast* | $[\text{DATA}, self, m]$ $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, $[\text{DATA}, s, m]$ $\rangle$ **do**

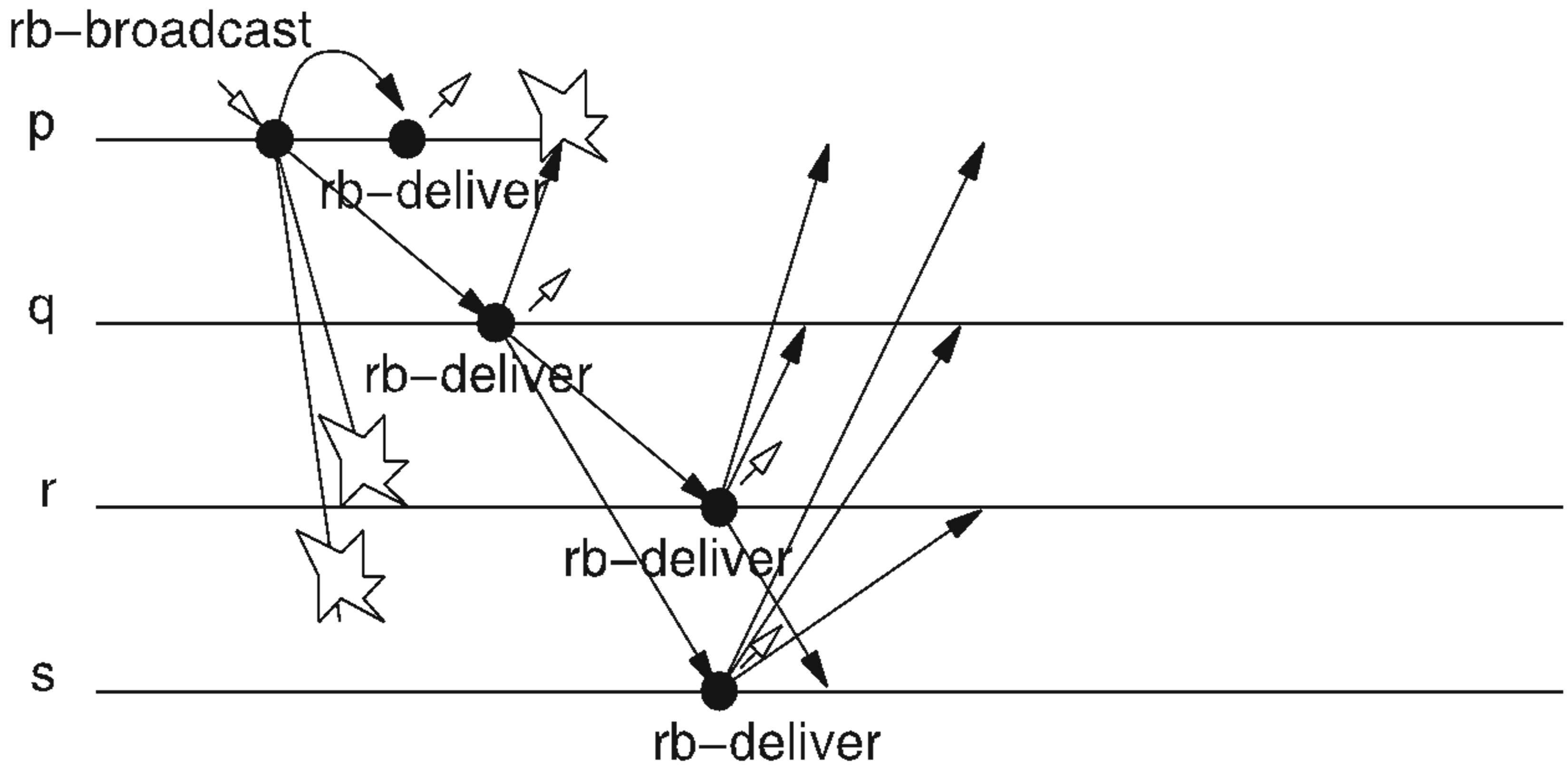
if $m \notin delivered$ **then**

$delivered := delivered \cup \{m\}$;

trigger $\langle$ *rb*, *Deliver* | s, m $\rangle$;

trigger $\langle$ *beb*, *Broadcast* | $[\text{DATA}, s, m]$ $\rangle$;

Eager Reliable Broadcast



Proof (sketch)

- **RB1. RB2. RB3:** as for the 1st algorithm
- **RB4. Agreement:**
  - every correct process immediately relays every message it *rb*-delivers
  - the *validity* property of the underlying best-effort broadcast primitive ensures that all other correct process will eventually deliver the message.

Note: Does not use a failure detector ⇔
Does not make any synchrony assumptions

Overview

We shall consider three forms of reliability for a broadcast primitive

- **(1) Best-effort broadcast**
- **(2) Reliable broadcast in the crash fault model**
- **(3) Reliable broadcast in the arbitrary fault model**

- We provide **specifications** and then **algorithms**

Reliable broadcast in the byzantine model

- A byzantine process may, e.g.:
  - broadcast messages different from the ones sent by applications,
  - forge messages so that they look as originated by other processes
    - violation of the No creation property
  - send different messages to different processes
    - violation of the *agreement* property

Reliable broadcast in the byzantine model

- Authenticated Perfect Links are a useful tool:
  - disallows forging messages as originated from other processes
  - still does not solve the problems above
- Digital signatures can also be helpful:
  - allow any process to verify the authenticity of a message
  - but faulty processes may sign “illegal” msgs
    - e.g., sending two different messages, after having signed them

Byzantine Consistent and Reliable Broadcasts

- We consider two variants of reliable broadcasts in the byzantine model:

  - Byzantine Consistent Broadcast

  - Byzantine Reliable Broadcast

Byzantine Consistent and Reliable Broadcasts

- Byzantine Consistent Broadcast
  - If the sender s is correct then every correct process should later deliver m.
  - If s is faulty, then every correct process delivers the same message, <u>if it delivers one at all.</u>
    - i.e., correct processes are <u>not</u> guaranteed to deliver the same set of messages if sender is faulty
- Byzantine Reliable Broadcast
  - Ensures that if a correct process delivers m, then every correct process delivers m
    - <u>**independently of whether the sender is faulty**</u>

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

Overview

We shall consider three forms of reliability for a broadcast primitive

- **(1) Best-effort broadcast**
- **(2) Reliable broadcast in the crash fault model**
- **(3) Reliable broadcast in the arbitrary fault model**

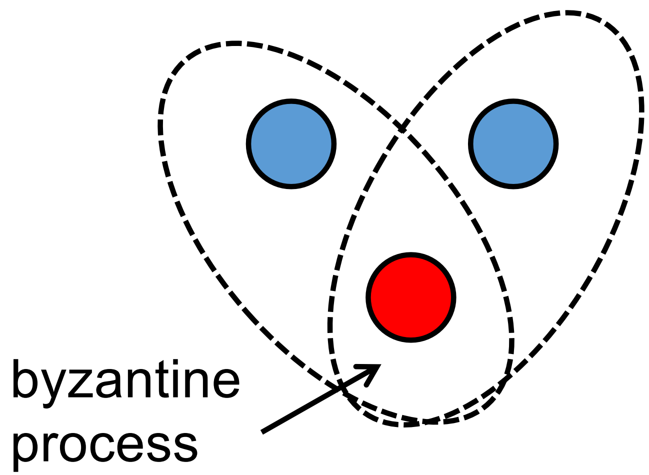
- We provide **specifications** and then **algorithms**

Byzantine Consistent Broadcast: Algorithms

- Byzantine Consistent Broadcast
  - If the sender s is correct then every correct process should later deliver m.
  - If s is faulty, then every correct process delivers the same message, <u>if it delivers one at all.</u>
    - i.e., correct processes are <u>not</u> guaranteed to deliver the same set of messages
- Two algorithms:
  - Authenticated Echo Broadcast
    - exchanges a quadratic number of messages
  - Signed Echo Broadcast
    - linear no. of messages but uses digital signatures (costly)
  - Leverage **Byzantine quorums**

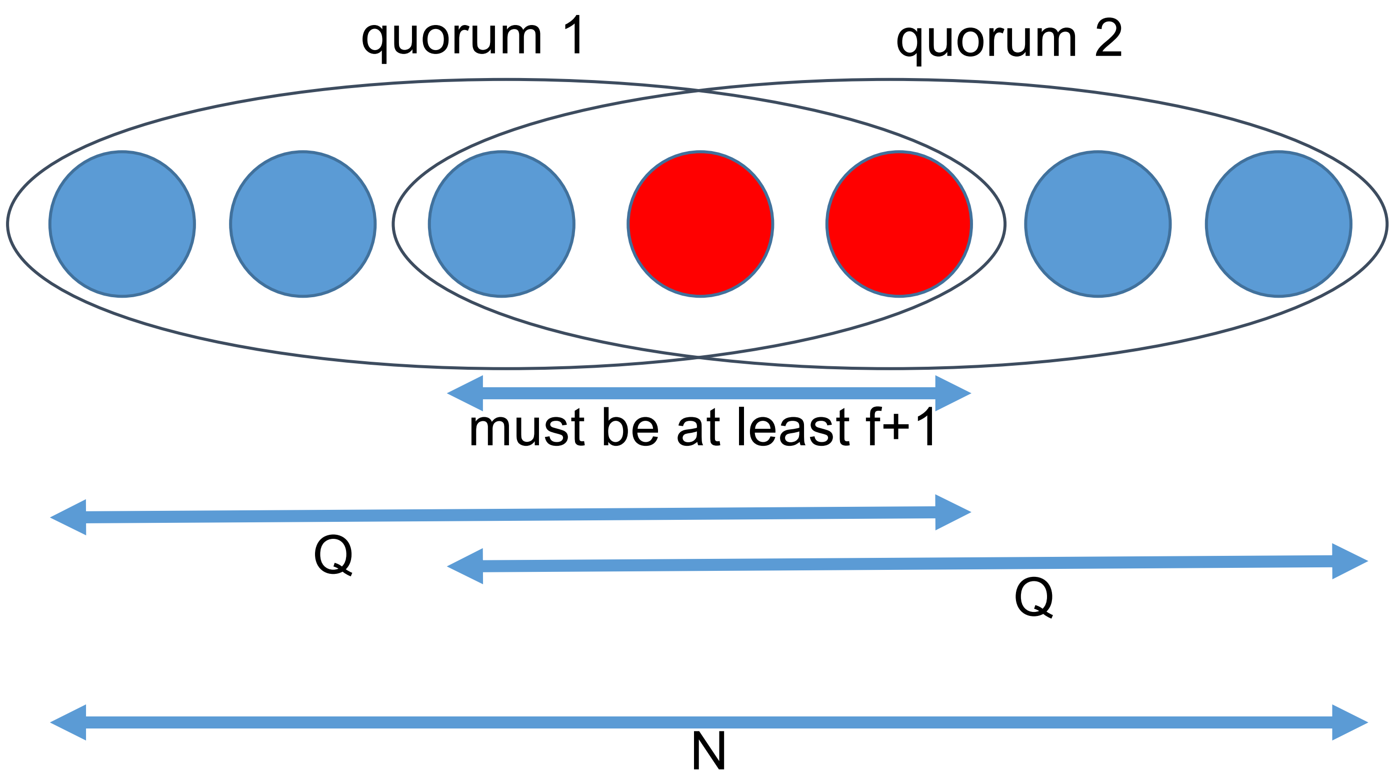
Quorums and Byzantine Quorums

- Quorums are sets of subsets with a specific intersection property:
  - they always overlap in at least one correct process
  - example:
    - majority of processes, assuming f<N/2 **crash failure model**
    - intersection of 2 majority quorums is necessarily a correct process with crash faults
    - no longer true with byzantine faults!



Derivation of Byzantine quorum sizes

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



Derivation of Byzantine quorum sizes

- $Q + Q – N \geq f+1$
- To simplify, we take the optimal size (no need to intersect more than strictly necessary)
- $2Q – N = f+1$

Derivation of Byzantine quorum sizes

- Liveness: a quorum always needs to be available
- Challenge: f processes may never reply (crashed or deliberately silent)
- Thus quorums can't be larger than $N - f$
- $N - f \geq Q$
- Again, we turn this into an equality (don't use more processes than strictly necessary)
- $N - f = Q$

Solving a system of two equations

- Note that f is a system parameter (constant)

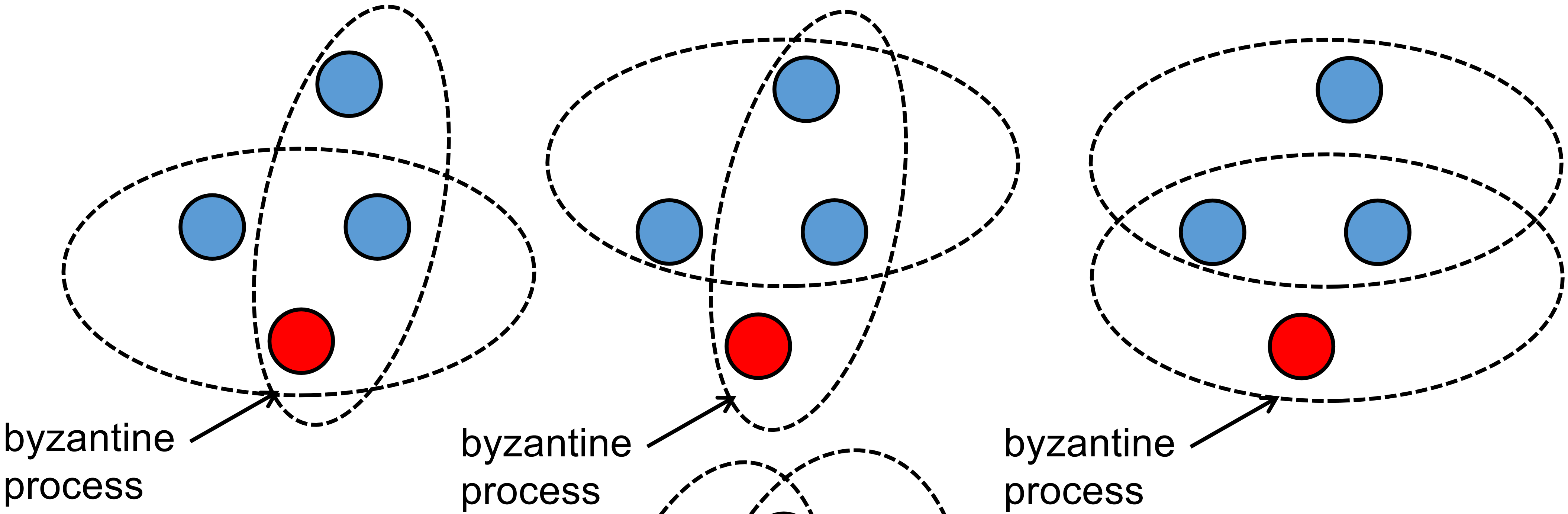
$$\begin{cases} 2Q - N = f+1 \\ N - f = Q \end{cases}$$

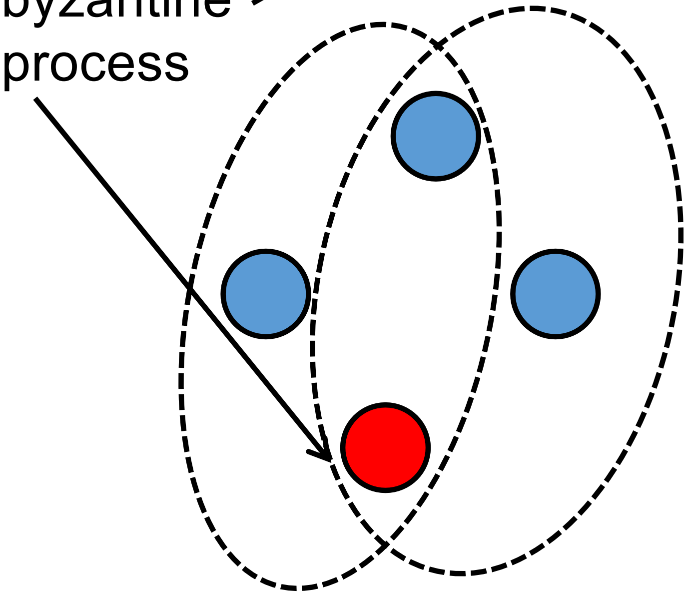
- Thus, solving these equations for N and Q yields:

$$N = 3f+1$$

$$Q = 2f+1$$

Byzantine Quorums: examples





The intersection of any two Byzantine Quorums contains at least a correct process

Acknowledgements

- Rachid Guerraoui, EPFL