# Reliable Broadcast

**Seif Haridi**

haridi@kth.se

# Designing Algorithms

# Combining Abstractions

- **Fail-stop** (synchronous)
  - Crash-stop process model
  - Perfect links + Perfect failure detector (P)
- **Fail-silent** (asynchronous)
  - Crash-stop process model
  - Perfect links
- **Fail-noisy** (partially synchronous)
  - Crash-stop process model
  - Perfect links + Eventually Perfect failure detector (◊P)
- **Fail-recovery**
  - Crash-recovery process model
  - Stubborn links + …

# Fail-stop model

- **Fail-stop**
  - Crash-stop process model
  - Perfect links + Perfect failure detector (P)

- **Synchronous model**

- Local algorithms can track the set of correct processes
- Without violating liveness properties: use
  - Techniques based request/reply
  - Waiting for acknowledgment for all correct processes

# Fail-silent model

- **Fail-silent**
  - Crash-stop process model
  - Perfect links
- **Asynchronous model**

- No access to failure detectors
- Assumes a majority of processes are always correct
- Often use a majority quorum techniques (next unit)
- Local algorithm cannot wait for more than $\lceil n/2 \rceil + 1$ otherwise it might get stuck

# Fail-noisy model

- **Fail-noisy**
  - Crash-stop process model
  - Perfect links
  - Eventually Perfect failure detector (◊P)
- **Partially synchronous model**

- To guarantee safety properties any algorithm has to assume the failure detector inaccurate

- Eventual accuracy is only used to guarantee liveness

# Fail-recovery model

- **Fail-recovery**
  - Crash-recovery process model
  - Stubborn links  or a persistent links (logs)

- Relies often on a persistent memory to store and retrieve critical information
- After recovery a process may contact other process to retrieve up to date state information

- Some algorithms relax the reliability conditions on channels allowing message loss/duplication/reordering

# Quorums in crash-stop process model

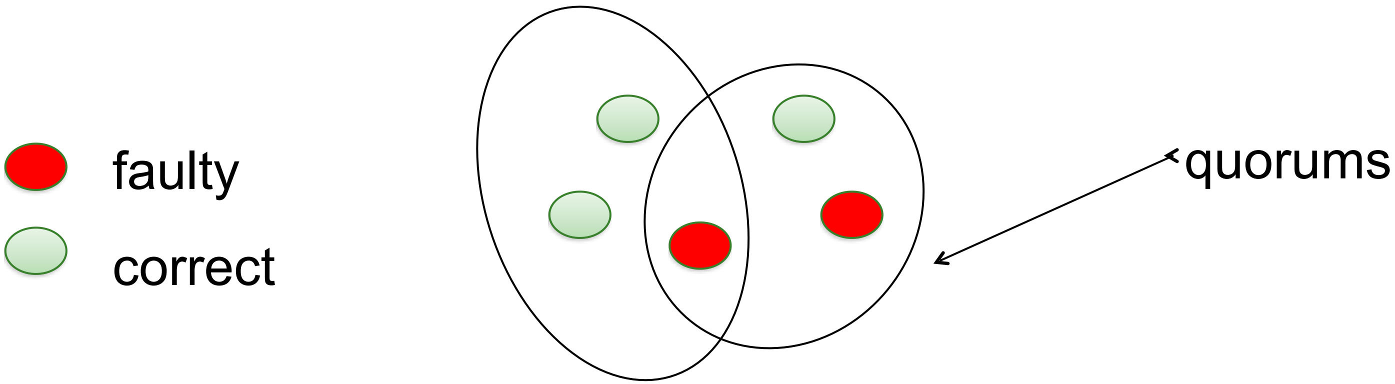
# Quorums

- For N crash-stop process abstractions
- Quorum is any set of majority of processes
- A set with at least $\lfloor N/2 \rfloor+1$ processes

- The algorithms will rely on a majority of processes will not fail
  - $f < N/2$ (f is the max number of faulty processes)
- f is the resilience of the algorithm

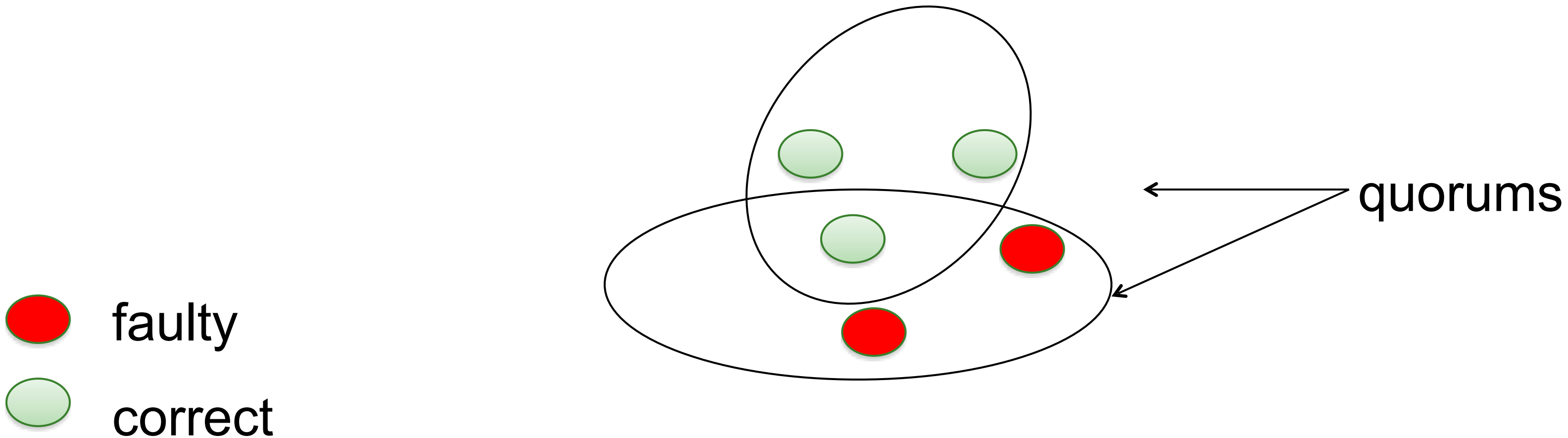
# Quorums crash-stop/recovery model $f < N/2$

- Two quorums always intersect in at least ONE process



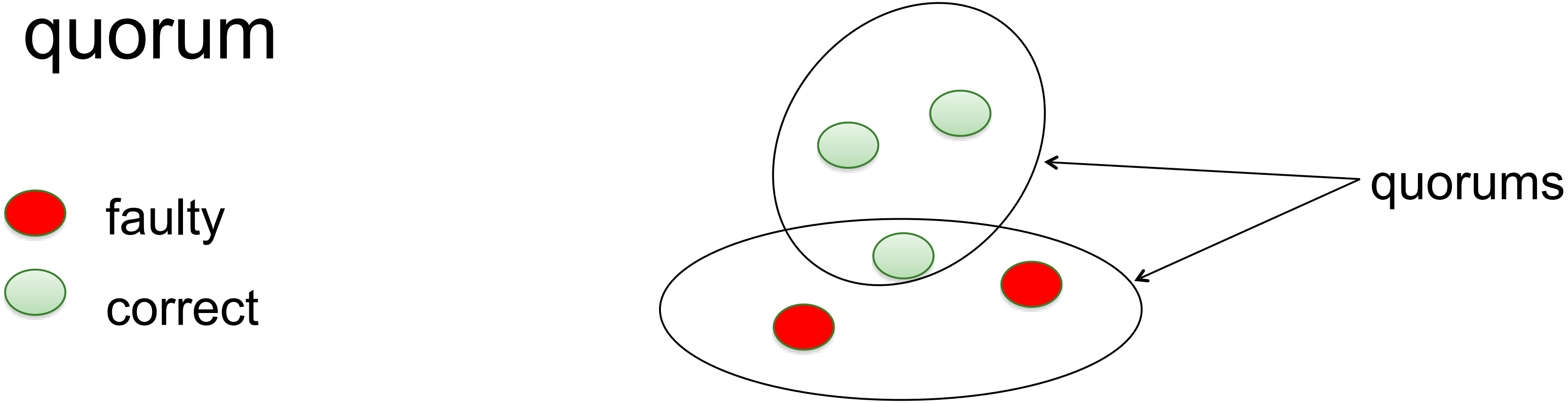
# Quorums crash-stop/recovery model $f < N/2$

- There is at least ONE quorum with only correct processes



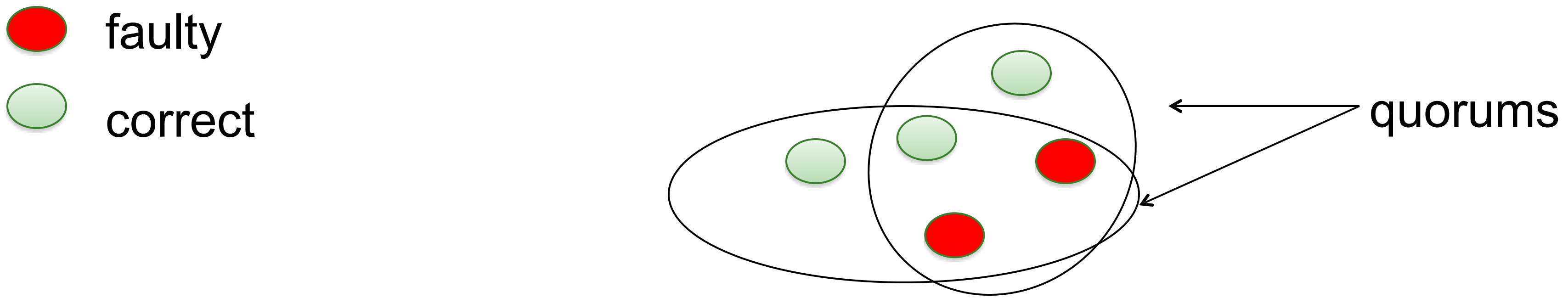
# Quorums crash-stop/recovery model $f < N/2$

- There is at least ONE correct process in each quorum



# Quorums

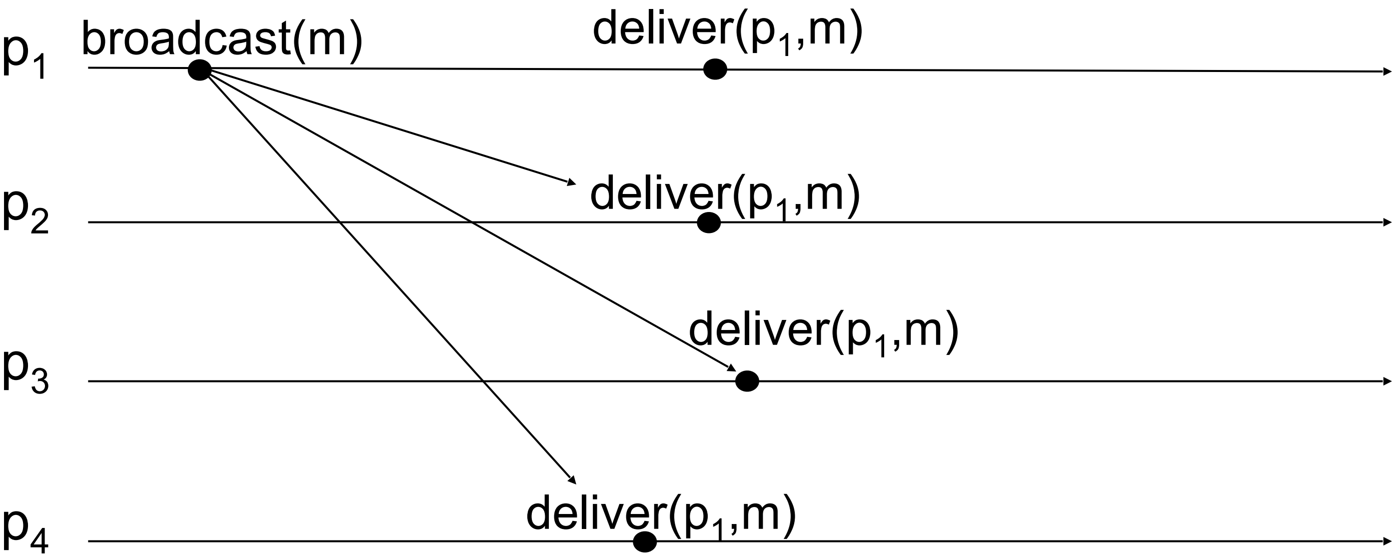
- Quorums used in Fail-Silent and Fail-Noisy algorithms
- A process never waits for messages from more than $\lfloor N/2 \rfloor + 1$ (different) processes



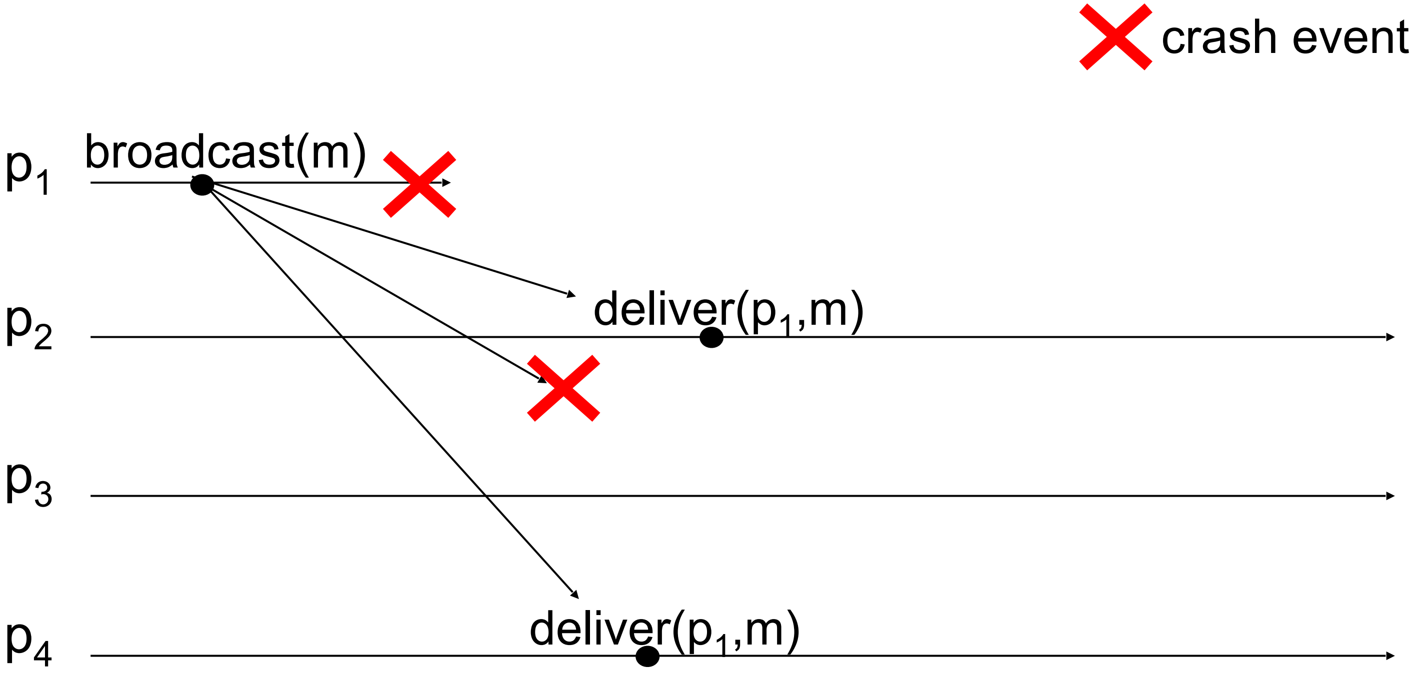
# Broadcast Abstractions

# Broadcast Services

- Send a message to a group of processes



# Unreliable Broadcast



# Reliable Broadcast Abstractions

- **Best-effort broadcast**
  - Guarantees reliability only if sender is correct
- **Reliable broadcast**
  - Guarantees reliability independent of whether sender is correct
- **Uniform reliable broadcast**
  - Also considers behavior of failed nodes
- **FIFO reliable broadcast**
  - Reliable broadcast with FIFO delivery order
- **Causal reliable broadcast**
  - Reliable broadcast with causal delivery order

# Reliable Broadcast Abstractions

- **Probabilistic reliable broadcast**
  - Guarantees reliability with high probability
  - Scales to large number of nodes

- **Total order (atomic) reliable broadcast**
  - Guarantees reliability and same order of delivery

# Specification of Broadcast Abstractions

# Best-effort broadcast (beb)

- ***Instance beb***
- ***Events***
  - Request: ⟨beb Broadcast | m⟩
  - Indication: ⟨beb Deliver | src, m⟩
- ***Properties: BEB1, BEB2, BEB3***

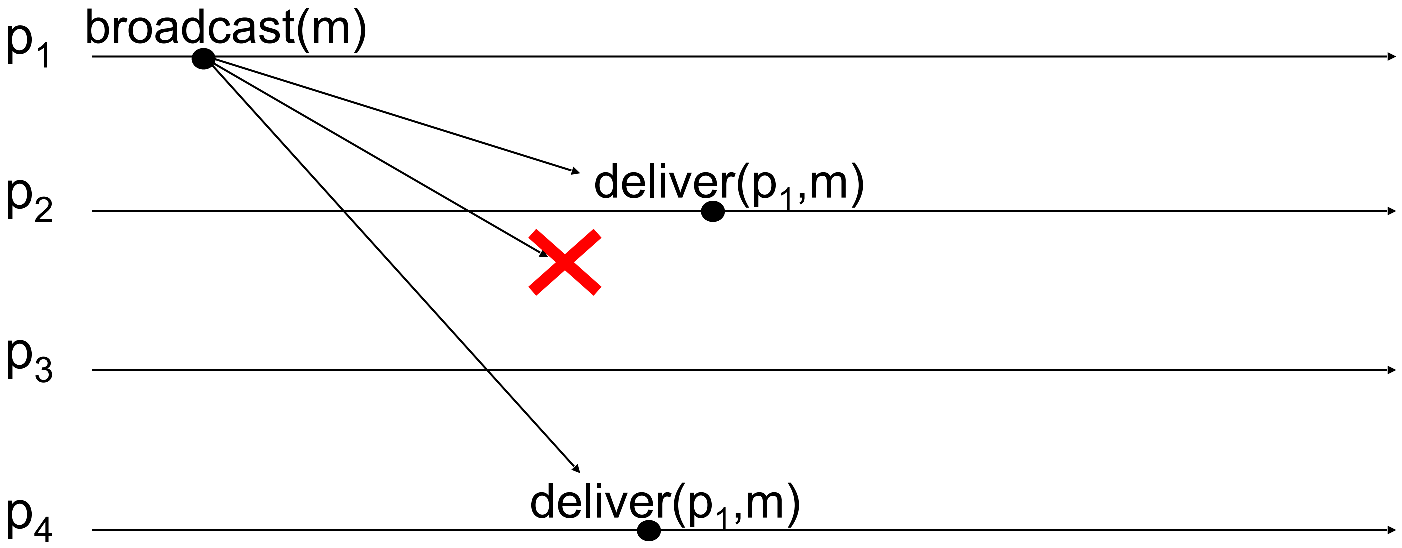
# Best-effort broadcast (beb)

- ***Intuitively***: everything perfect unless sender crash

- ***Properties***
  - ***BEB1. Best-effort-Validity***: If $p_i$ and $p_j$ are correct, then any broadcast by $p_i$ is eventually delivered by $p_j$
  - ***BEB2. No duplication:*** No message delivered more than once
  - ***BEB3. No creation:*** No message delivered unless broadcast

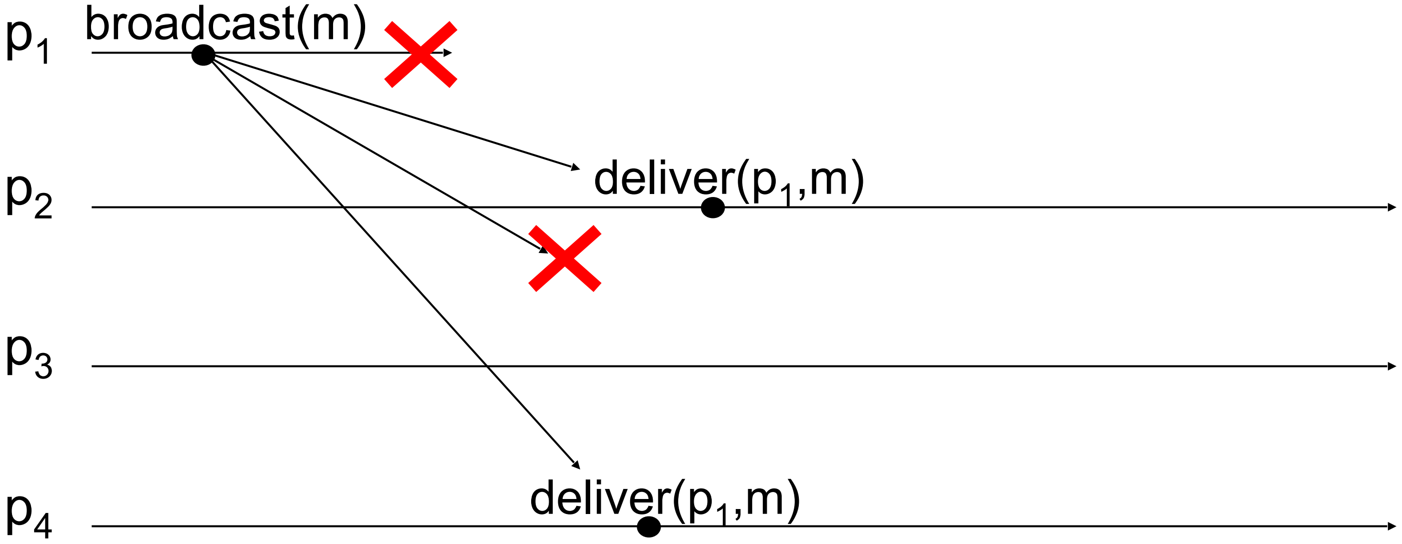
# BEB Example

- Is this allowed? No



# BEB Example (2)

- Is this allowed? Yes



# Reliable Broadcast

- BEB gives no guarantees if sender crashes
  - Strengthen to give guarantees if sender crashes

- Reliable Broadcast Intuition
  - Same as BEB, plus
  - If sender crashes:

ensure *all or none* of the correct nodes get msg

# Reliable Broadcast (rb)

- ***Instance rb***
- ***Events***
  - Request: ⟨rb Broadcast | m⟩
  - Indication: ⟨rb Deliver | src, m⟩
- ***Properties: RB1, RB2, RB3, RB4***

# Reliable Broadcast Properties

- ***Properties***
  - ***RB1 = BEB1. Validity***
  - ***RB2 = BEB2. No duplication***
  - ***RB3 = BEB3. No creation***

  - ***RB4. Agreement.***
    - If a **correct process delivers** m, then every correct process delivers m

# Refining correctness

- Can weaken RB1 without any effect

**Old Validity** ←equivalent with→ **New Validity**

***RB1 = BEB1 Validity***

- If $p_i$ and $p_j$ are **correct**, then any broadcast by $p_i$ is eventually delivered by $p_j$

***RB2 = BEB2. No duplication***

***RB3 = BEB3. No creation***

***RB4. Agreement.***

- If a **correct node delivers** m, then every correct node delivers m

***RB1 Validity.***

- If **correct** $p_i$ broadcasts m, $p_i$ itself eventually delivers m

***RB2 = BEB2. No duplication***

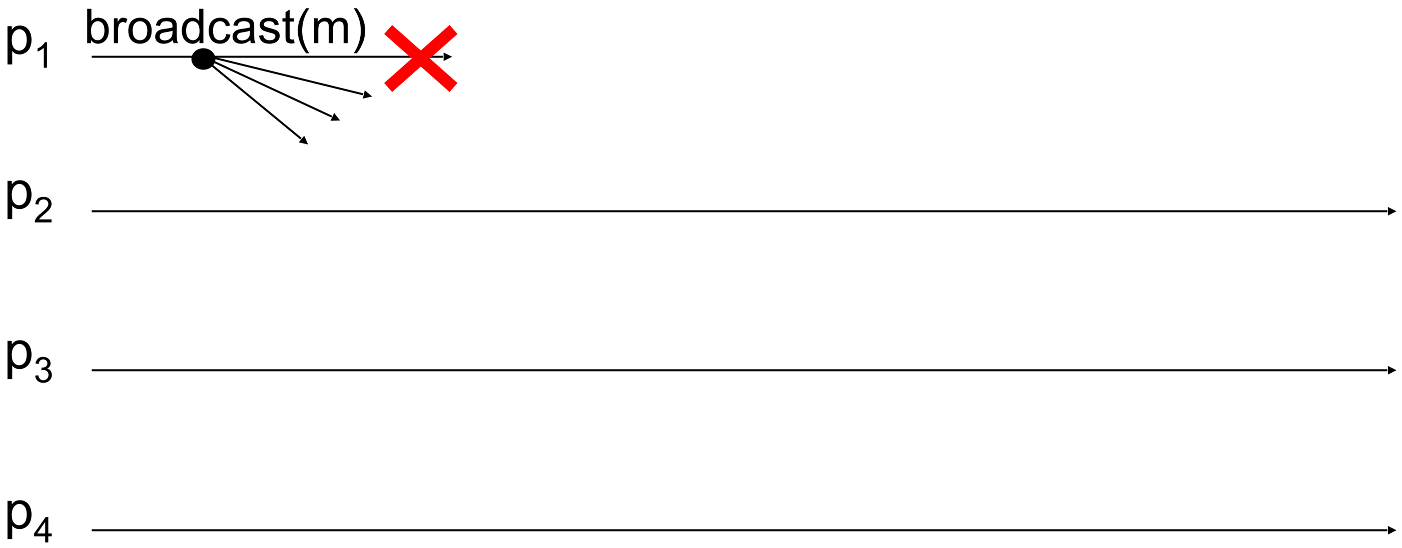
***RB3 = BEB3. No creation***

***RB4. Agreement.***

- If a **correct node delivers** m, then every correct process delivers m

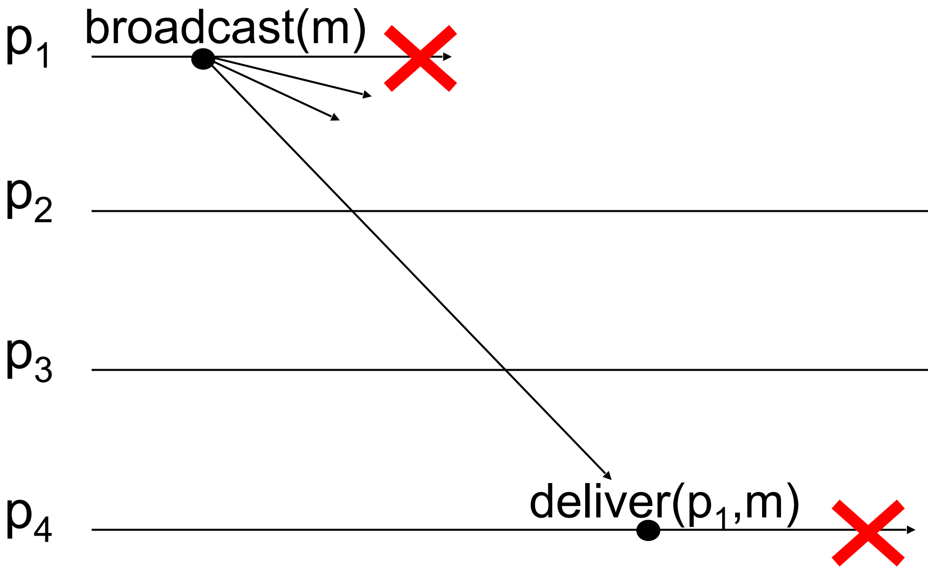
# RB Example

- Is this allowed? Yes



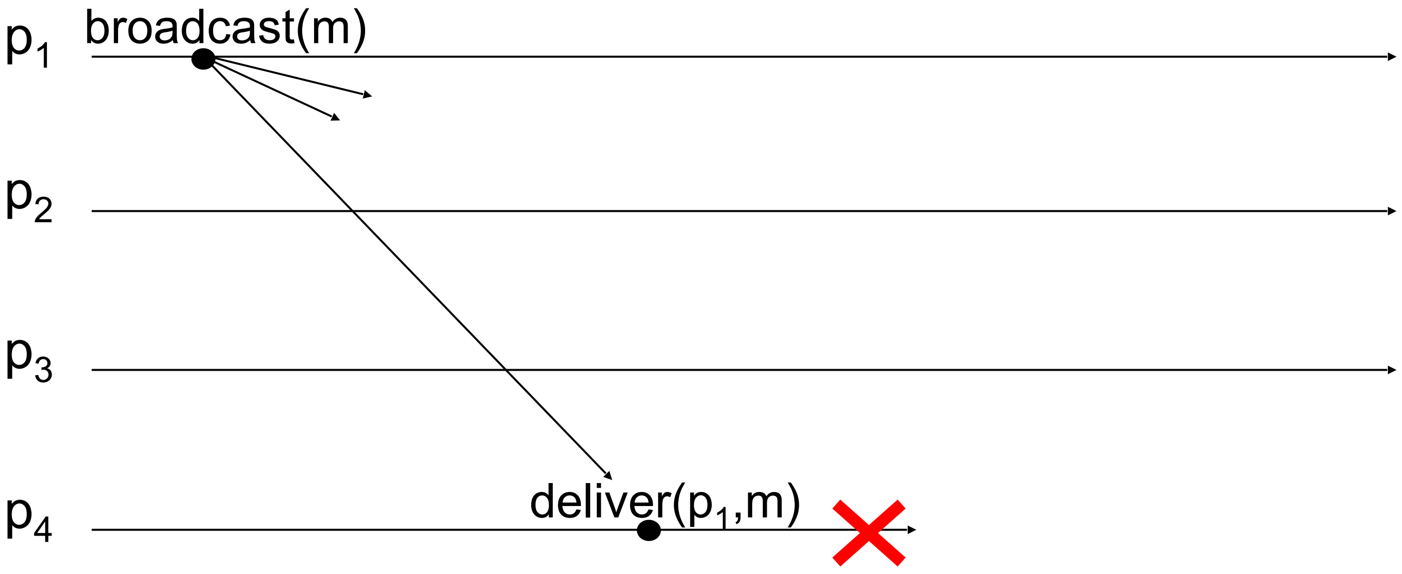
# RB Example

- Is this allowed? Yes



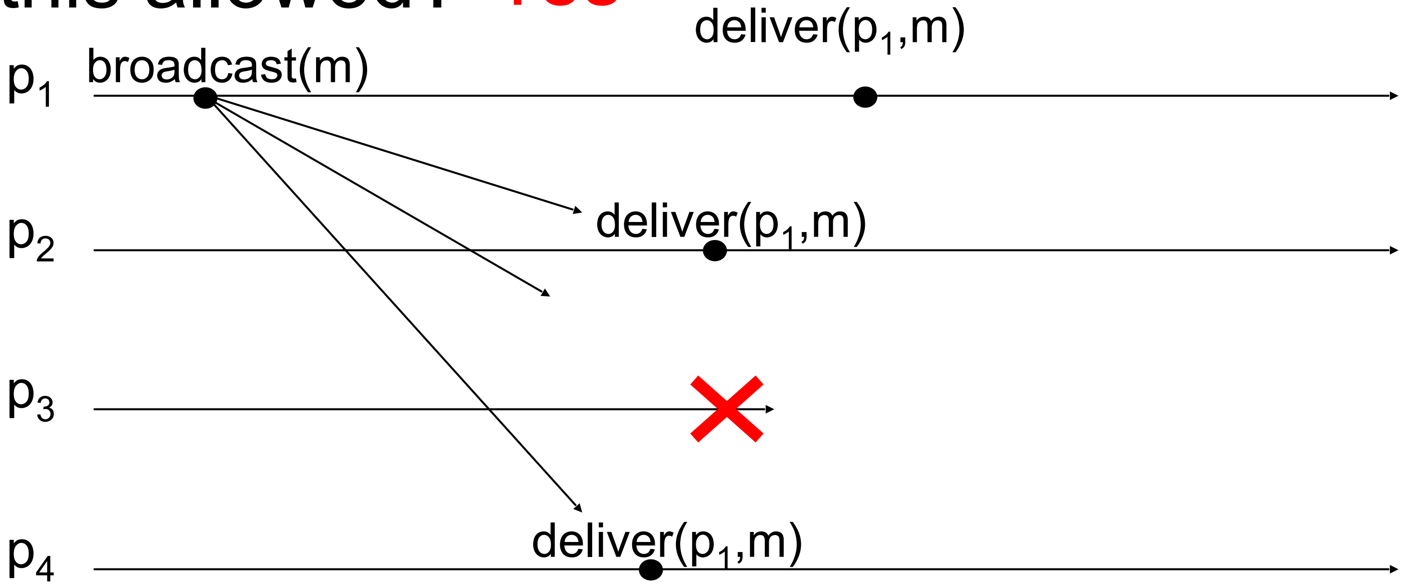
# RB Example

- Is this allowed? No



# RB Example

- Is this allowed? Yes



# Uniform Reliable Broadcast

- Assume sender broadcasts message
  - Sender fails
  - No correct process delivers message
  - Some failed processes deliver message

Is it ok

- Assume the broadcast enforces
  - Printing a message on paper
  - Withdrawing money from account
- Uniform reliable broadcast intuition
  - If a failed node delivers, everyone must deliver…

At least correct nodes, we cannot revive the dead…

# Uniform broadcast (urb)

- ***Events***
  - Request: ⟨urb Broadcast | m⟩
  - Indication: ⟨urb Deliver | src, m⟩

- ***Properties:***
  - ***URB1***
  - ***URB2***
  - ***URB3***
  - ***URB4***

# Uniform Broadcast Properties

- ***Properties***

  - ***URB1 = RB1.***
  - ***URB2 = RB2.***
  - ***URB3 = RB3.***
  - ***URB4. Uniform Agreement:*** For any message m, if a **process delivers** m, then every correct process delivers m

Wanted: Dead & Alive!

# Broadcast Abstractions

# Implementation of Broadcast Abstractions

# Implementing BEB

- Use Perfect channel abstraction
  - Upon ⟨beb Broadcast | m⟩ send message m to all processes (for-loop)
- Correctness
  - If sender doesn't crash, every other correct process receives message by perfect channels (Validity)
  - No creation & No duplication already guaranteed by perfect channels

# Fail-Stop
# Lazy Reliable Broadcast

# Fail-Stop: Lazy Reliable Broadcast

- Requires perfect failure detector (**P**)

- To broadcast m:
  - best-effort broadcast m
  - When get **beb** Deliver
    - Save message, and
    - **rb** Deliver message

- If sender s crash, detect & relay msgs from s to all
  - **case 1**: get m from s, detect crash s, redistribute m
  - **case 2**: detect crash s, get m from s, redistribute m

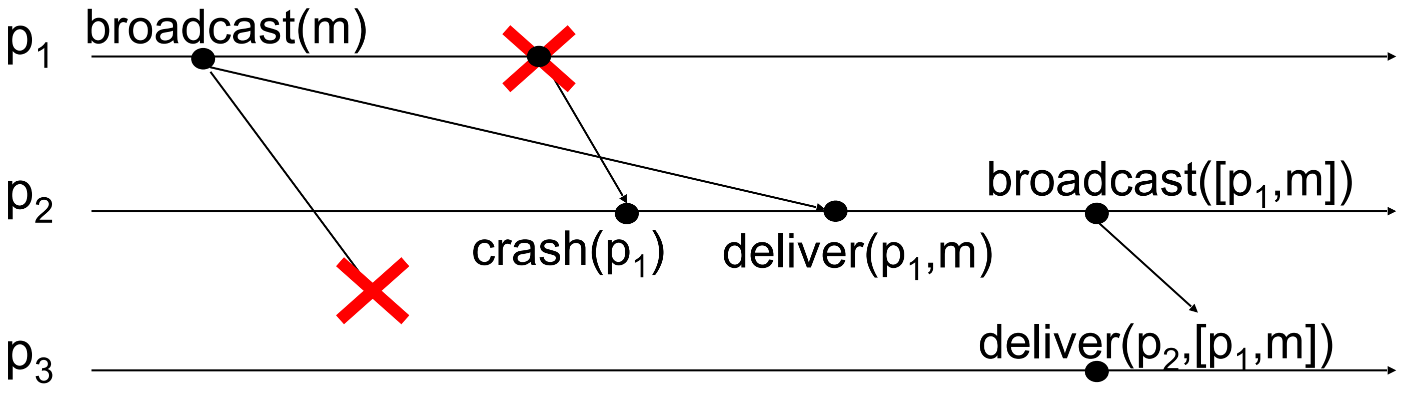
- Filter duplicate messages before delivery

# Fail-Stop: Lazy Reliable Broadcast

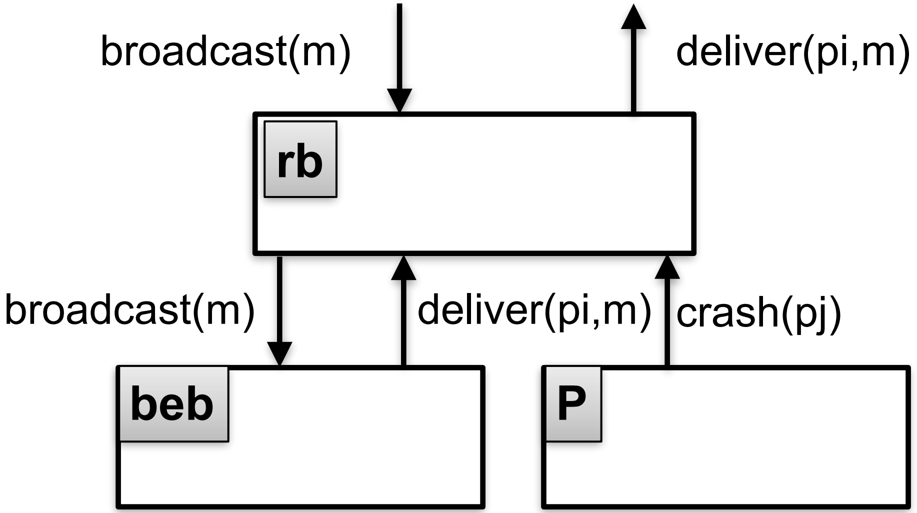
- If sender s crash, detect & relay msgs from s to all
  - **case 1**: get m from s, detect crash s, redistribute m
  - **case 2**: detect crash s, get m from s, redistribute m
    - Why case 2? [d]

# Lazy Reliable Broadcast

## Case 2



# Fail-stop Lazy Reliable Broadcast



# Lazy Reliable Broadcast

- **Implements:** ReliableBroadcast (rb)
- **Uses:**
  - BestEffortBroadcast (beb)
  - PerfectFailureDetector (P)
- **upon event** ⟨Init⟩ **do**
  - delivered := ∅
  - correct := Π
  - **forall** $p_i \in \Pi$ **do** from[$p_i$] := ∅
- **upon event** ⟨rb Broadcast | m⟩ **do**
  - **trigger** ⟨beb Broadcast | (DATA, self, m)⟩

**for filtering duplicates**

**storage for saved messages**

# Lazy Reliable Broadcast (2)

- **upon event** ⟨crash | $p_i$⟩ **do**
  - correct := correct \ {$p_i$}
  - **forall** ($s_m$,m) ∈ from[$p_i$] **do**

    **trigger** ⟨beb Broadcast | (DATA, $s_m$ ,m)⟩

Case 1: redistribute anything we have from failed node

- **upon event** ⟨beb Deliver | $p_i$, (DATA, $s_m$ , m)⟩ **do**
  - **if** m ∉ delivered **then**
  - delivered := delivered ∪ {m}
  - from[$p_i$] := from[$p_i$] ∪ { ($s_m$, m)}
  - **trigger** ⟨rb Deliver | $s_m$ , m⟩
  - **if** pi ∉ correct **then**

    **trigger** ⟨beb Broadcast |(DATA, $s_m$, m) ⟩

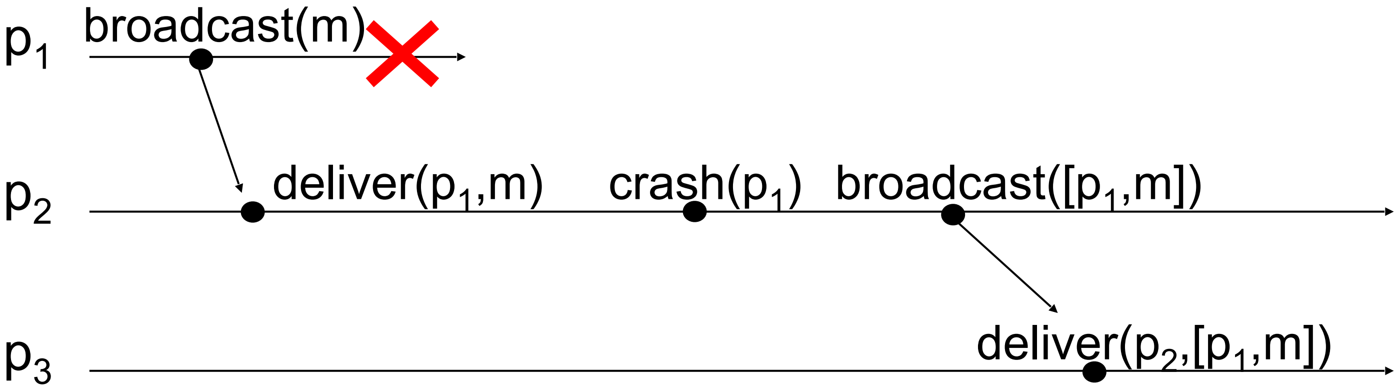
Avoid duplicates

Store for future

Case 2: redistribute

# RB Example

- Which case? Case 1



# Correctness of Lazy RB

- ***RB1-RB3*** satisfied by BEB
- Need to prove ***RB4***
  - If a **correct node delivers** m, then every correct node delivers m
- Assume Correct $p_k$ delivers message bcast by $p_i$
  - If $p_i$ is correct, BEB ensures correct delivery
  - If $p_i$ crashes,
    - $p_k$ detects this (completeness)
    - $p_k$ uses BEB to ensure (BEB1) every correct node gets it

# Measuring Performance

# Message Complexity

- The number of messages required to terminate an operation of an abstraction

- Lazy reliable broadcast
  - The number of messages initiated by broadcast(m)
  - Until a deliver(src, m) event is issued at each process

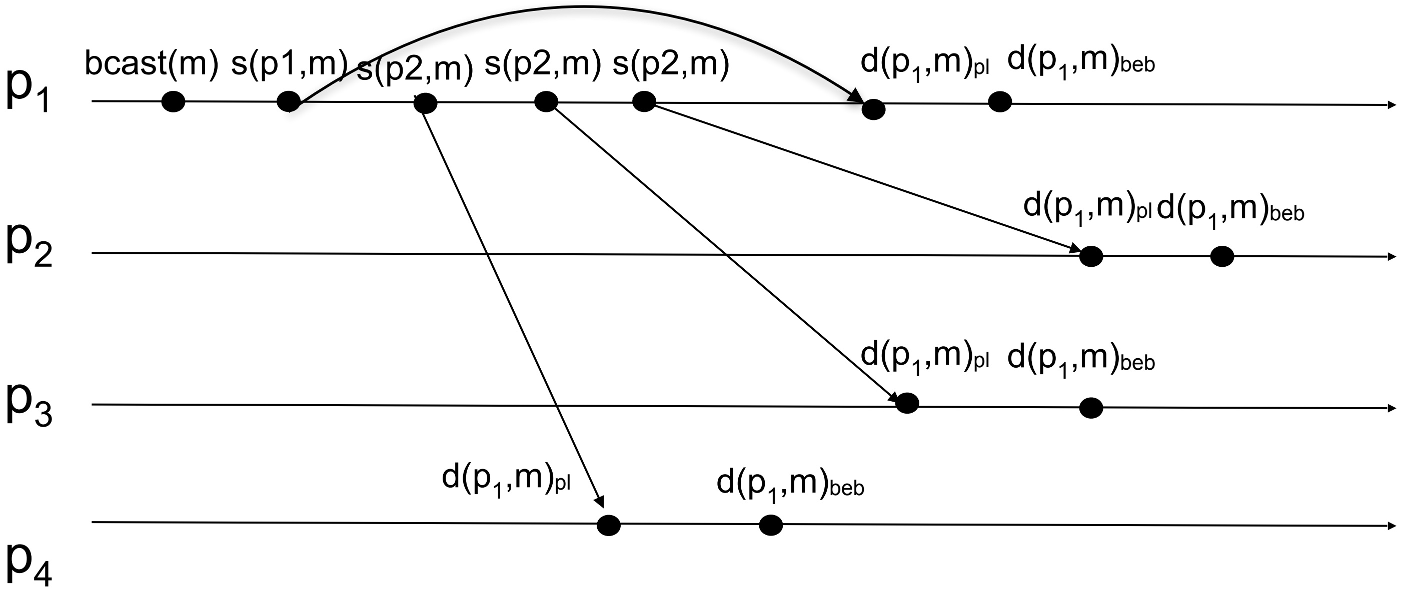
- Bit complexity
  - Number of bits sent, if messages can vary in size

# Time Complexity

- **One time unit** in an Execution E is the **longest** message delay in E
- **Time Complexity is** Maximum time taken by any execution of the algorithm under the assumptions
  - A process can execute any finite number of actions (events) in **zero** time
  - The time between $send(m)_{i,j}$ and $deliver(m)_{i,j}$ is **at most one** time unit
- In most algorithms we study we assume all communication steps takes one time unit

# Best effort broadcast

- Takes **one time unit** from broadcast$(m)_p$ to last deliver(p,m)
- We also call it one **communication step**



# Complexity of lazy reliable broadcast

- **Assume** N processes
- **Message complexity**
  - Best case: O(N) messages
  - Worst case: $O(N^2)$ messages
- **Time complexity**
  - Best case: 1 time unit
  - Worst case: 2 time units

# Fail-Silent
# Eager Reliable Broadcast

# Eager Reliable Broadcast

- What happens if we replace P with ◊P? [d]
  - Only affects performance
  - Only affects correctness
  - No effect
  - Affects performance  and correctness

# Eager Reliable Broadcast

- Can we modify Lazy RB to not use P? [d]
  - Just assume all processes failed
  - BEB Broadcast as soon as you get a msg

# Eager Reliable Broadcast

- **Uses:** BestEffortBroadcast (beb)

- **upon event** ⟨Init⟩ **do**
  - delivered := ∅

- **upon event** ⟨rb Broadcast | m⟩ **do**
  - delivered := delivered ∪ {m}
  - **trigger** ⟨rb Deliver | self , m⟩
  - **trigger** ⟨beb Broadcast | (DATA, self, m)⟩

- **upon event** ⟨beb Deliver |$p_i$, (DATA, $s_m$ , m)⟩ **do**
  - **if** m ∉ delivered **then**
  - delivered := delivered ∪ {m}
  - **trigger** ⟨rb Deliver | $s_m$ , m⟩
  - **trigger** ⟨beb Broadcast | (DATA, $s_m$, m)⟩

**Immediately deliver**

**Immediately BEB broadcast**

**Immediately deliver**

**Immediately BEB broadcast**

# Correctness of Eager RB

- ***RB1-RB3*** satisfied by BEB
- Need to prove ***RB4***
  - If a **correct process delivers** m, then every correct node delivers m

- Assume correct $p_k$ delivers message bcast by $p_i$
  - $p_k$ uses BEB to ensure (BEB1) every correct process gets it

# Uniform Reliable Broadcast

# Uniformity

- Is the proposed algorithm also uniform? [d]
- Uniformity necessitates
  - If a **failed process** delivers a message m then every correct node delivers m

# Uniformity

- No.
  - Sender p immediately RB delivers and crashes
  - Only p delivered message
- **upon event** ⟨rb Broadcast | m⟩ **do**
  - delivered := delivered ∪ {m}
  - **trigger** ⟨rb Deliver | self , m⟩
  - **trigger** ⟨beb Broadcast | (DATA, self, m)⟩

# Uniform Eager RB

- Necessary condition for uniform RB delivery
  - All correct processes will get the msg
  - How do we know the correct processes got msg? [d]

- Messages are pending until all correct processes get it
  - Collect acks from processes that got msg

Use vector ***ack*[*m*]** at $p_i$: the set of processes that acked m

- Deliver once all correct processes acked
  - **Use perfect FD**
  - **function canDeliver**(m):
    - **return** correct $\subseteq$ ack[m]

# Uniform Eager RB implementation

- **upon event** ⟨urb Broadcast | m⟩ **do**
  - pending := pending $\cup$\{(self, m)\}
  - **trigger** ⟨beb Broadcast | (DATA, self, m)⟩

remember sent messages

- **upon event** ⟨beb Deliver | pi, (DATA, $s_m$, m)⟩ **do**
  - ack[m] := ack[m] $\cup$\{pi\}
  - **if** ($s_m$ , m) $\notin$ pending **then**
    - pending := pending $\cup$($s_m$, m)
    - **trigger** ⟨beb Broadcast | (DATA, $s_m$ , m)⟩

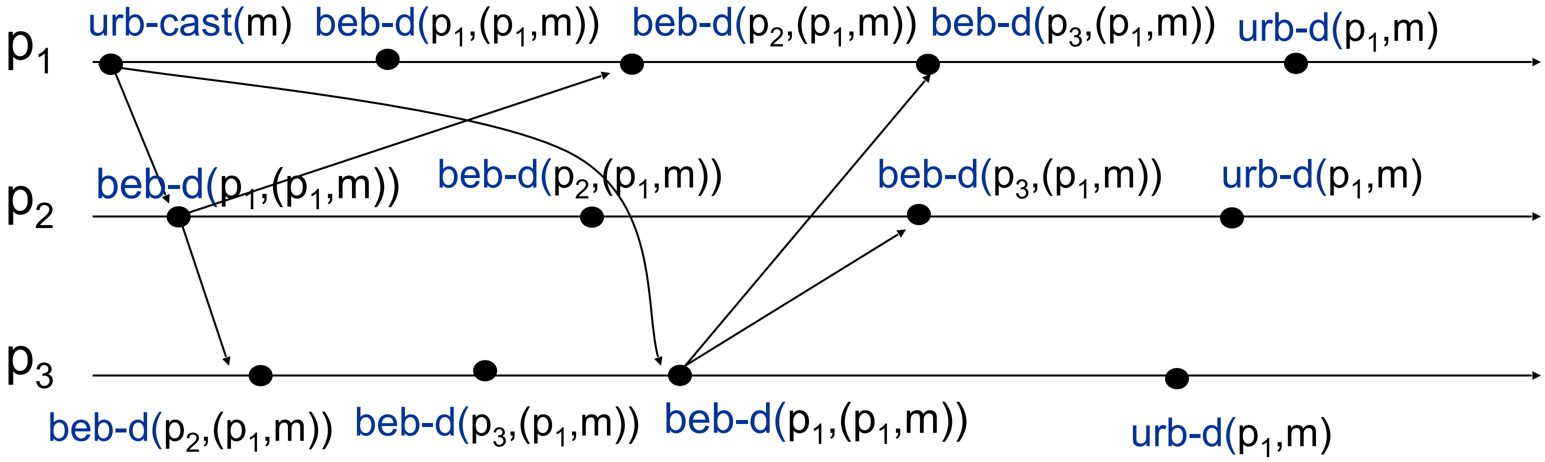
$p_i$ obviously got m

avoid resending

- **Upon exists** ($s_m$,m)$\in$pending **s.t.**
  - **canDeliver(m) and** m $\notin$ delivered **do**
    - delivered := delivered $\cup$ \{m\}
    - **trigger** ⟨urb Deliver | $s_m$, m⟩

deliver when all correct nodes have acked

# URB Eager Algorithm Example



# Correctness of Uniform RB

- No creation from BEB
- No duplication by using ***delivered*** set
- **Lemma**
  - If a **correct** process $p_i$ bebDelivers m, then $p_i$ eventually urbDelivers m
- **Proof**
  - Correct process $p_i$ bebBroadcasts m as soon as it gets m
    - By BEB1 every correct process gets m and bebBroadcasts m
    - $p_i$ gets bebDeliver(m) from every correct process by BEB1
    - By completeness of **P**, it will not wait for dead nodes forever
      - **canDeliver(m)** becomes true and $p_i$ delivers m

# Correctness of Uniform RB

- Validity
  - If sender s is correct, it'll by validity (BEB1) bebDeliver m
  - By the lemma, it will eventually urbDeliver(m)

# Correctness of Uniform RB

- Uniform agreement
  - Assume some process (possibly failed) URB delivers m
    - Then canDeliver(m) was true,

    by **accuracy** of P **every** correct process has BEB delivered m
  - By lemma each of the nodes that BEB delivered m will URB deliver m

# Uniform Broadcast
# Fail-Silent

# How useful is the uniform algorithm?

- Strong failure detectors necessary for URB?
  - No, we'll provide RB for fail-silent model

- Assume a majority of correct nodes
  - Majority $=\lfloor n/2 \rfloor+1$, i.e. 6 of 11, 7 of 12…
- Every node eagerly BEB broadcast m
  - URB deliver m when received m from a majority

# Majority-ACK Uniform RB

- Same algorithm as uniform eager RB
  - Replace one function
  - **function** canDeliver(m)
    - **return** |ack[m]|>n/2

**majority has acknowledged m**

- Agreement (main idea)
  - If a process URB delivers, it got ack from majority
  - In that majority, one node, p, must be correct
  - p will ensure all correct processes BEB deliver m
    - The correct processes (majority) will ack and URB deliver

# Majority-ACK Uniform RB

- Validity
  - If correct sender sends m
    - All correct nodes BEB deliver m
    - All correct nodes BEB broadcast
    - Sender receives a majority of acks
    - Sender URB delivers m

# Resilience

- The maximum number of faulty processes an algorithm can handle
- The Fail-Silence algorithm
  - Has resilience less than N/2
- The Fail-Stop algorithm
  - Has resilience = N – 1