Applications of Consensus

Seif Haridi – Royal Institute of Technology
Peter Van Roy – Université catholique de Louvain

haridi(at)kth.se
peter .vanroy(at)uclouvain.be

Consensus

- In consensus, the nodes propose values
  - They all have to agree on one of these values
- Solving consensus is key to solving many problems in distributed computing
  - Total order broadcast (aka Atomic broadcast)
  - Atomic commit (databases)
  - Terminating reliable broadcast
  - Group membership

Consensus Properties

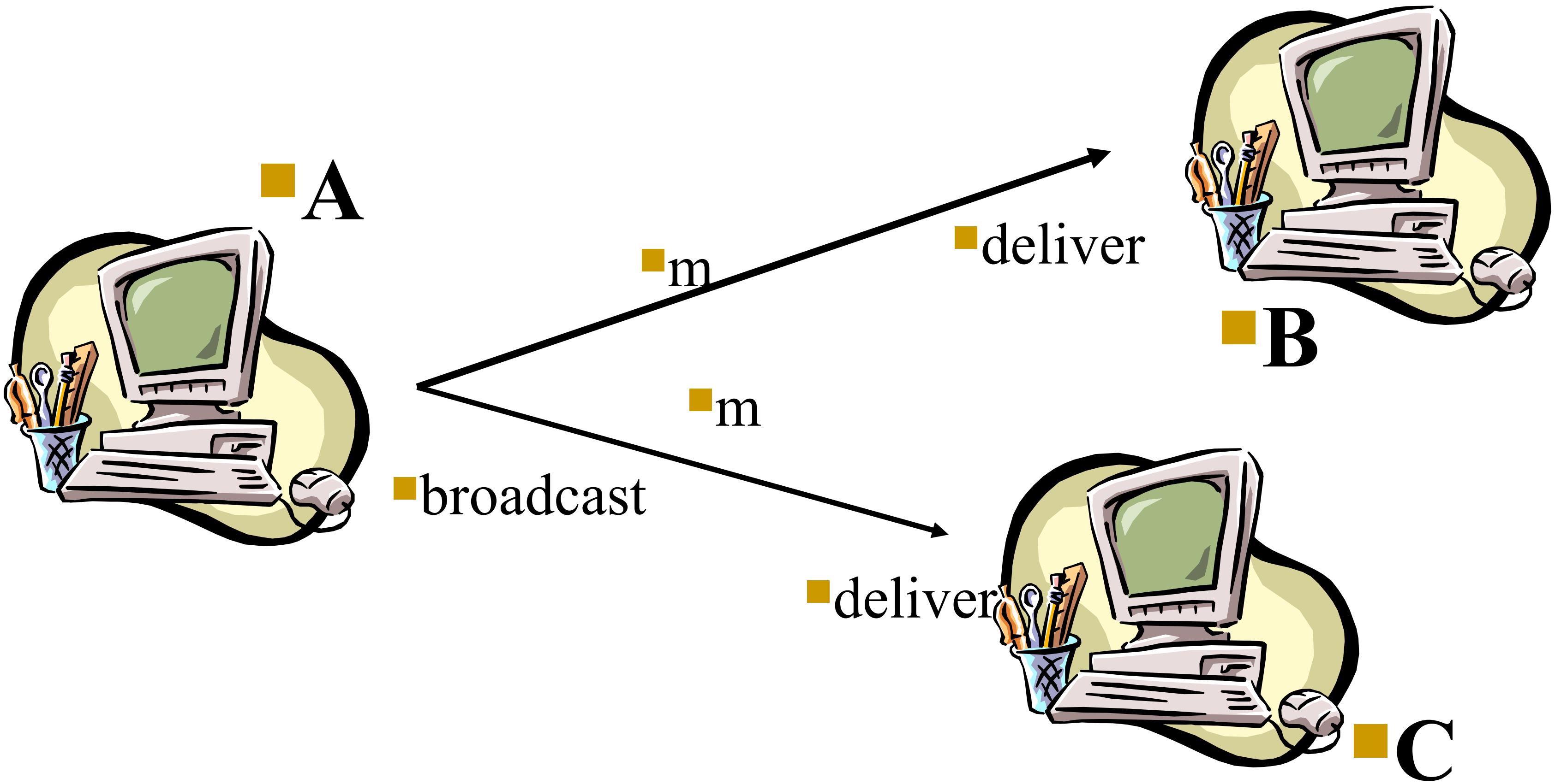
- ***C1. Validity***
  - Any value decided is a value proposed
- ***C2. Agreement***
  - No two correct nodes decide differently
- ***C3. Termination***
  - Every node eventually decides
- ***C4. Integrity***
  - A node decides at most once

Total Order Broadcast

Overview

- **Intuitions:** what does total order broadcast bring?
- **Specification** of ***total order broadcast***
- **Consensus-**based total order algorithm

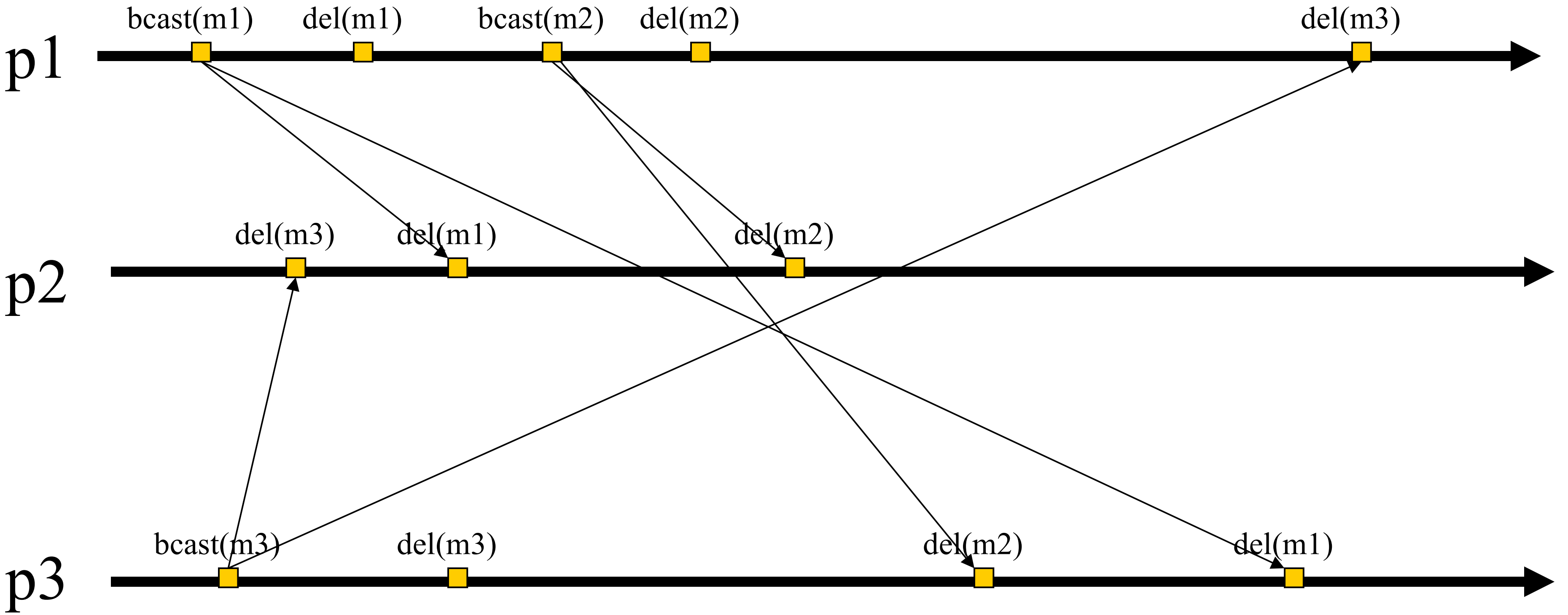
Broadcast



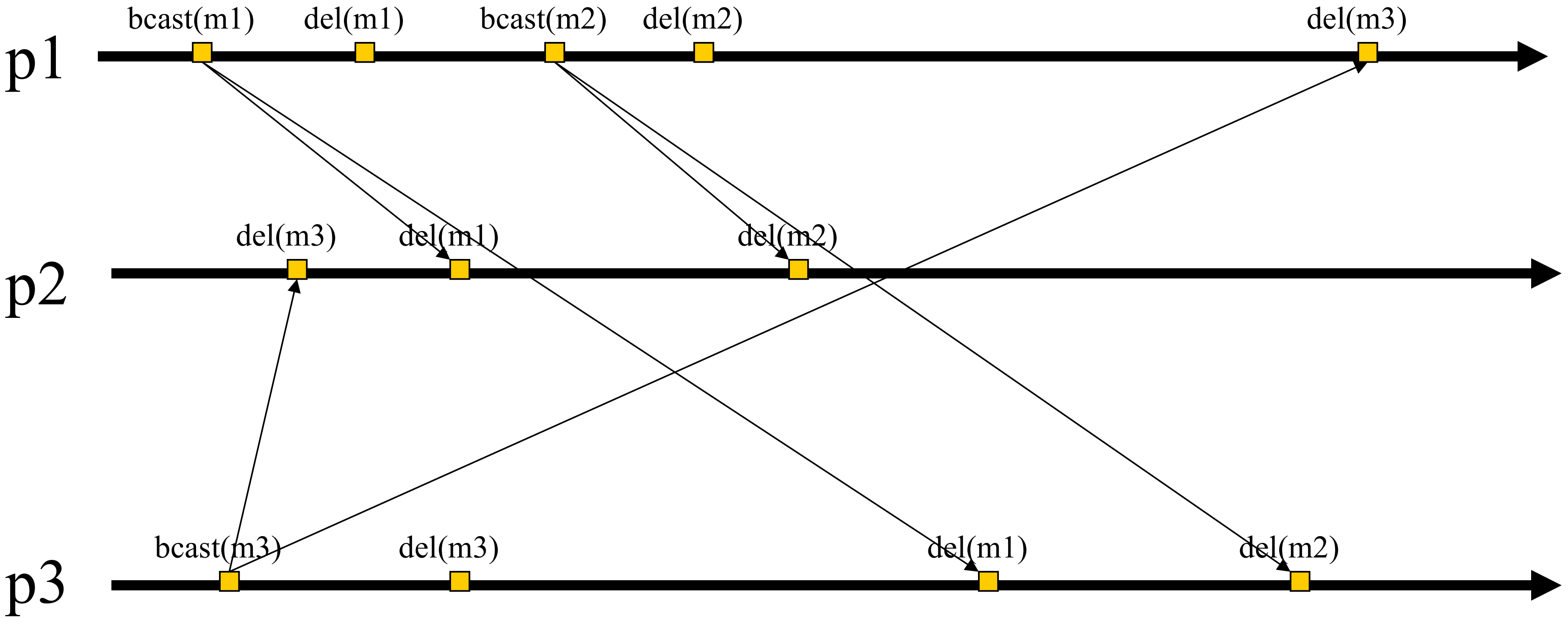
Intuitions (1)

- In ***reliable*** broadcast, the processes are free to deliver messages in any order they wish
- In ***causal*** broadcast, the processes need to deliver messages according to some order (causal order), which is related to the send order
- The order imposed by causal broadcast is however partial: some messages might be delivered in different order by different processes

Reliable Broadcast



Causal Broadcast



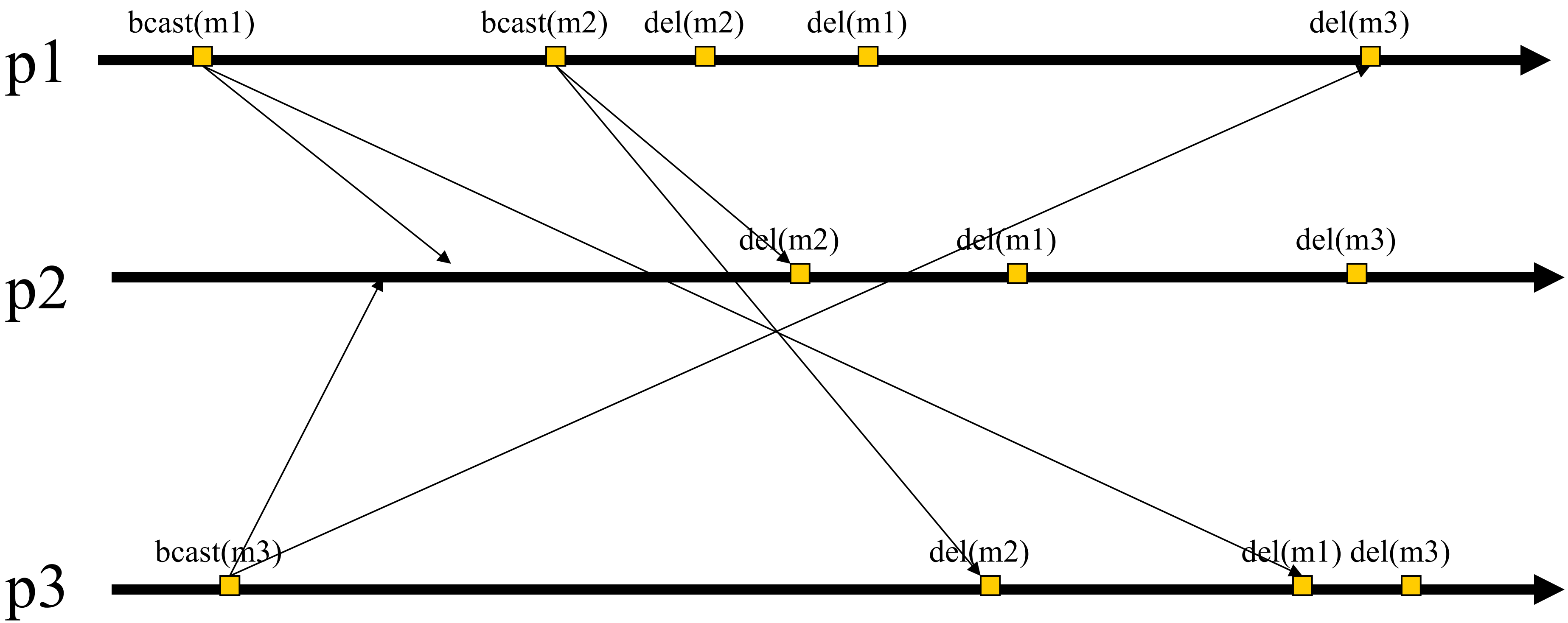
Intuitions (2)

- In ***total order*** broadcast, the processes must deliver all messages according to the same order (i.e., the order is now total)

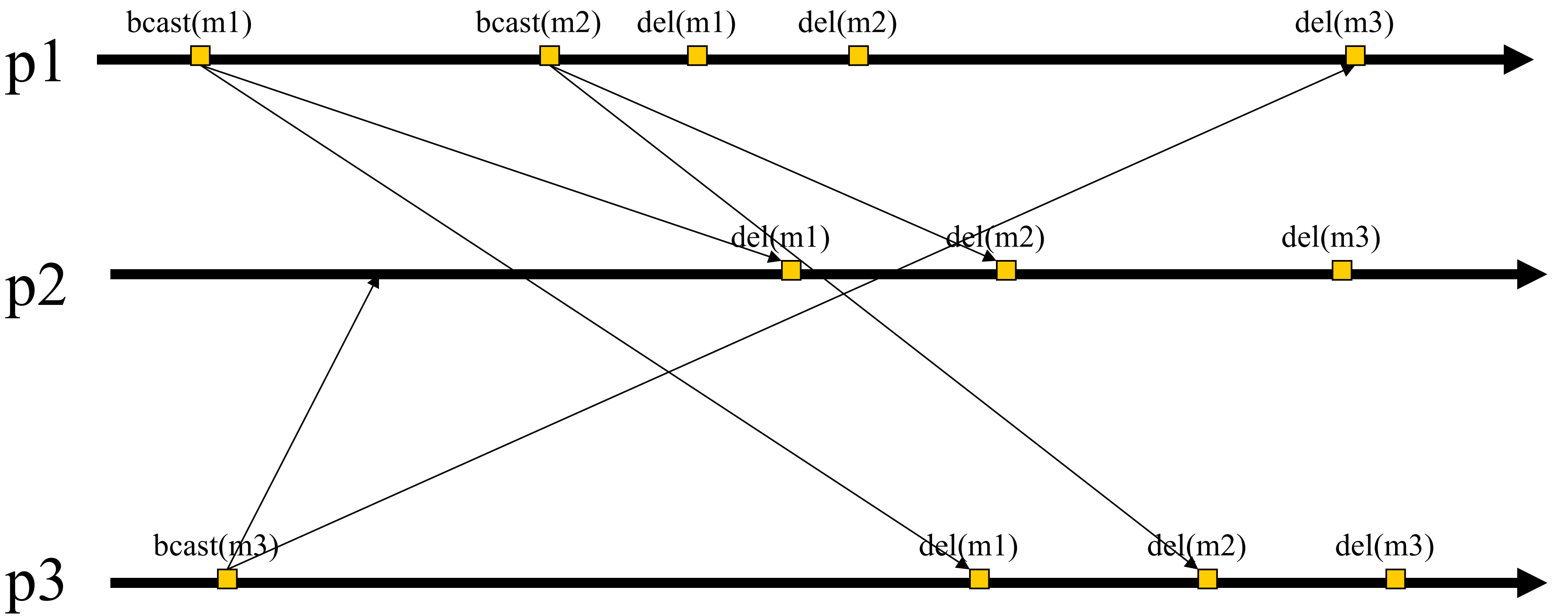
- Note that this order does **not** need to respect causality (or even FIFO ordering)

- Total order broadcast can be extended to respect causal (or FIFO) ordering

Total order broadcast (1)



Total order broadcast (2)



Intuitions (3)

- A replicated service where the replicas need to treat the requests in the ***same order*** to preserve consistency
  - For example, a distributed database

- A notification service where the subscribers need to get notifications in the same order
  - Because the order might be crucial, e.g., software updates

Overview

- **Intuitions:** what does total order broadcast bring?
- **Specification** of ***total order broadcast***
- **Consensus-**based algorithm

Total order broadcast (tob)

- ***Events***
  - Request: ⟨toBroadcast, m⟩
  - Indication: ⟨toDeliver, src, m⟩
- ***Properties:***
  - ***RB1, RB2, RB3, RB4***
  - ***Total order property***

Specification (I)

- ***Validity***: If pi and pj are correct, then every message broadcast by pi is eventually delivered by pj
- ***No duplication:*** No message is delivered more than once
- ***No creation:*** No message is delivered unless it was broadcast
- ***Uniform Agreement:*** For any message m: if any process delivers m, then every correct process delivers m

TO: Total order

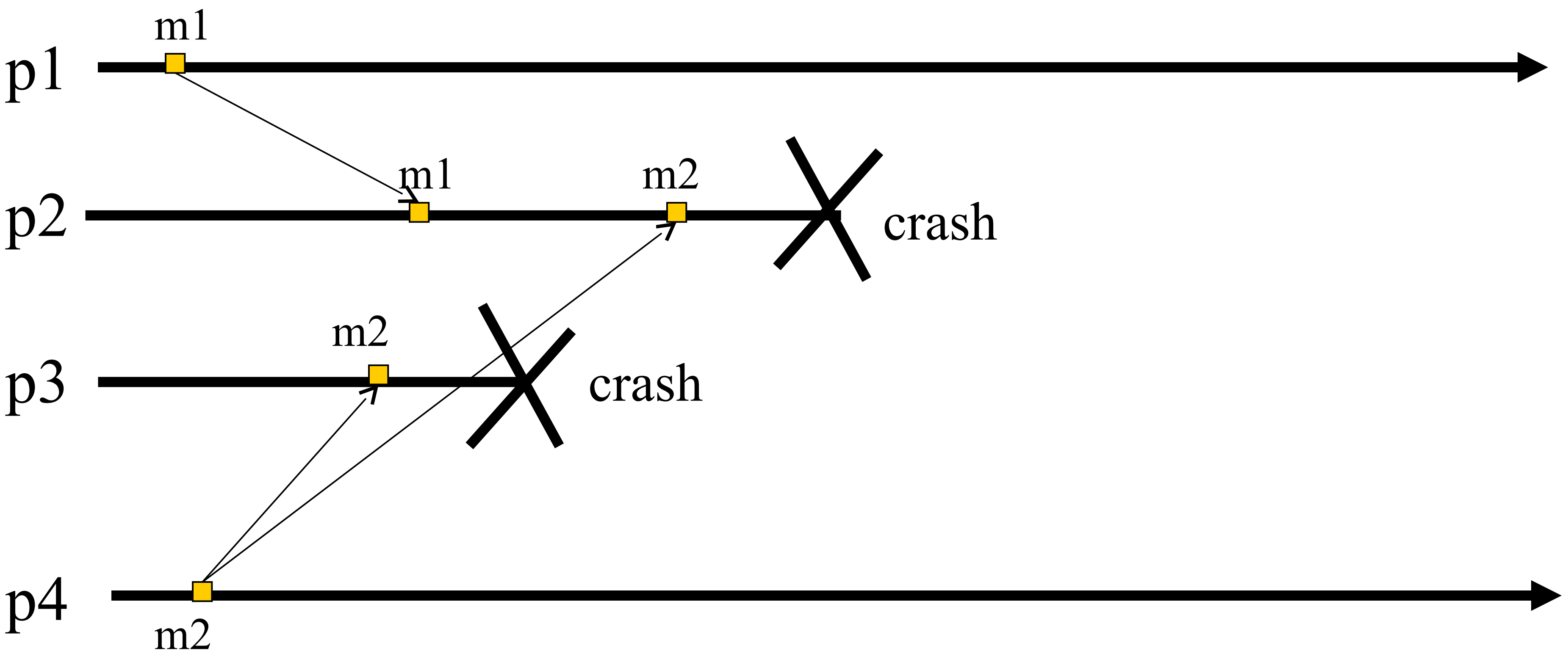
- Let m1 and m2 be any two messages and let pi and pj be any two correct processes that deliver m1 and m2
- If pi delivers m1 before m2, then pj delivers m1 before m2

UTO: Uniform total order

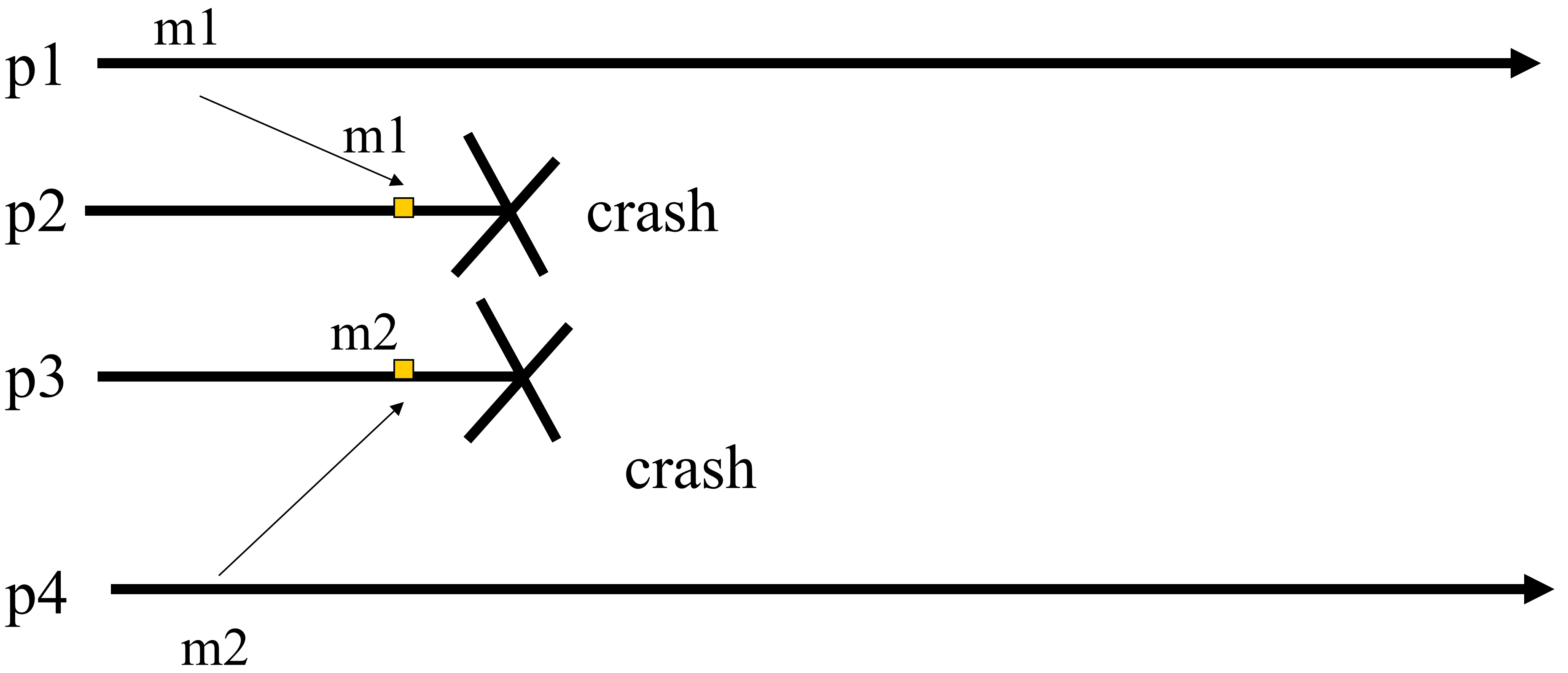
- Let m1 and m2 be any two messages. Let pi and pj be any two processes that deliver m2. If pi delivers m1 before m2, then pj delivers m1 before m2.

- Note: it's not "any two processes that deliver m1 and m2"!

Not uniform (why not?)



Uniform



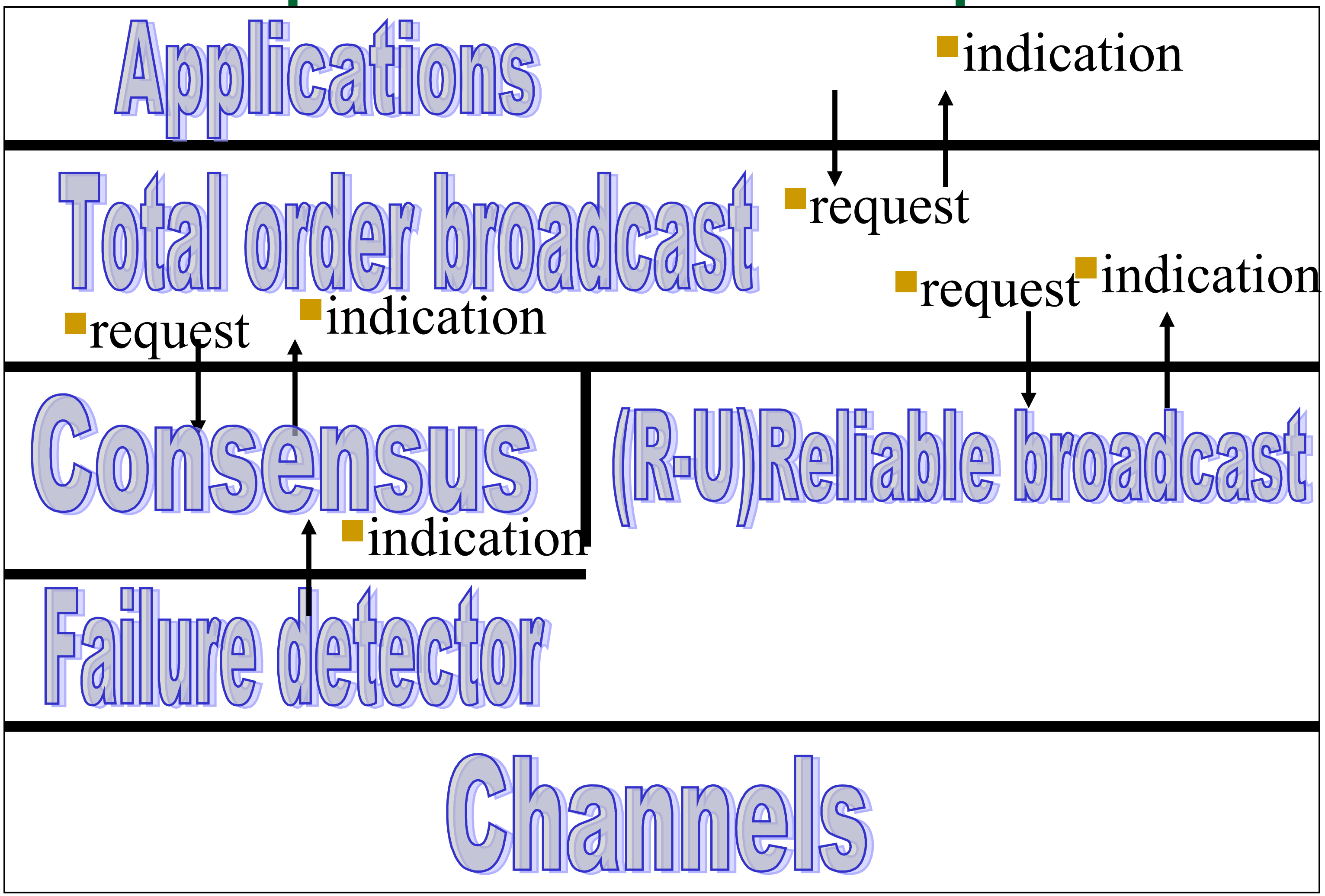
Overview

- **Intuitions:** what does total order broadcast can?

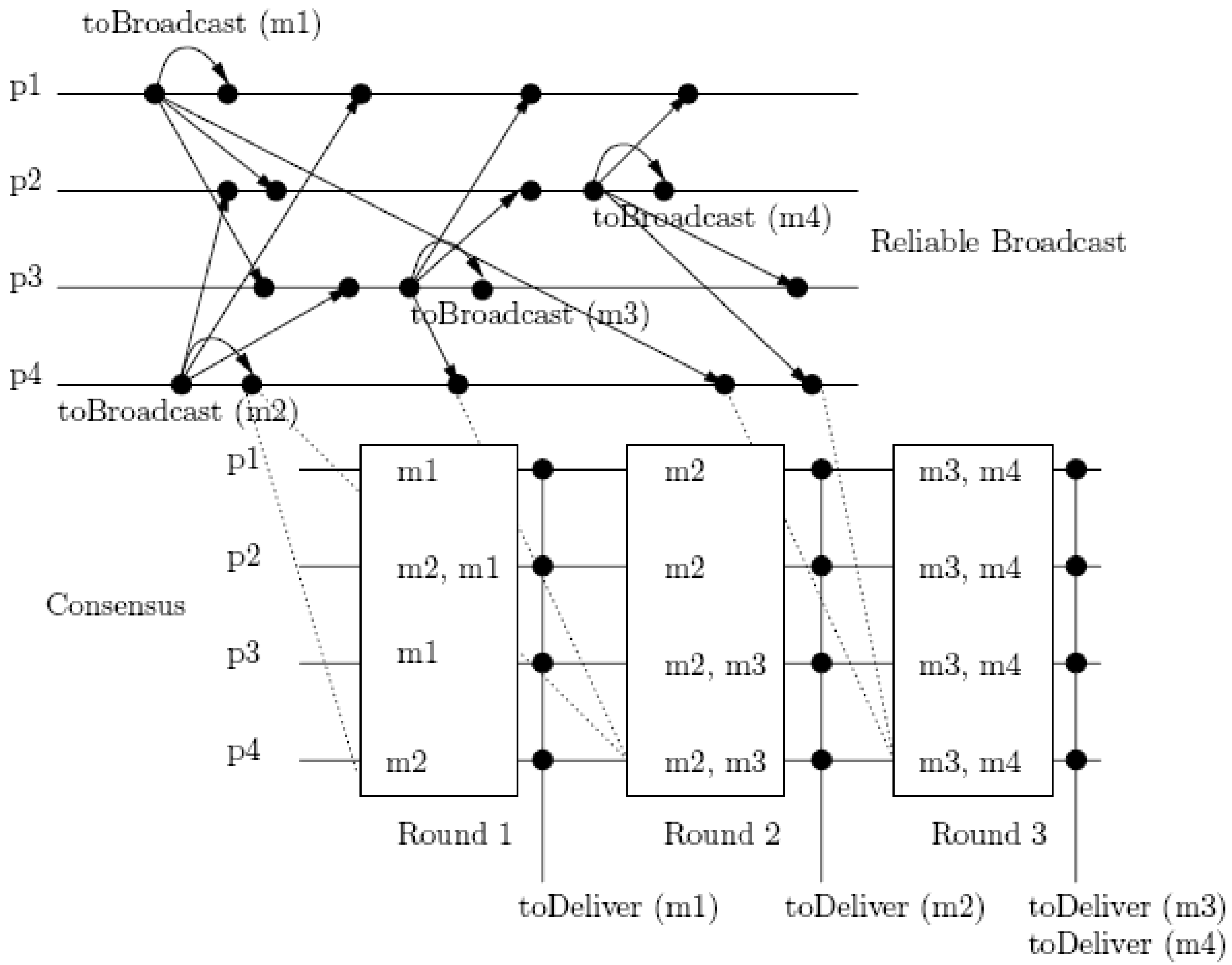
- **Specification** of ***total order broadcast***

- **Consensus-**based algorithm

Components of a process



ToBroadcast



Algorithm

- **Implements:** TotalOrder (to)
- **Uses:**
  - ReliableBroadcast (rb)
  - Consensus (cons)
- **upon event** < Init > **do**
  - unordered = delivered = ∅
  - wait := false
  - sn := 1

No consensus running

Consensus instance number

Algorithm (cont’d)

- **upon event** <toBroadcast,m> **do**
  - **trigger** <rbBroadcast,m>

- **upon event** <rbDeliver,sm,m> and (m $\notin$ delivered) **do**
  - unordered := unordered U {(sm,m)}

- **upon** (unordered $\neq \emptyset$) and not(wait) **do**
  - wait := true
  - **trigger** <Propose sn, unordered>

consensus instance sn

Algorithm (cont'd)

- **upon event** <Decide sn, decided> **do**
  - unordered := unordered \ decided
  - ordered := deterministicSort(decided)
  - **for all** (sm,m) in ordered:
    - **trigger** <toDeliver,sm,m>
    - delivered := delivered U {m}
  - sn : = sn + 1;
  - wait := false;

Equivalences

1. One can build consensus with total order broadcast (how?)
2. One can build total order broadcast with consensus and reliable broadcast

- ***Therefore, consensus and total order broadcast are equivalent problems in a system with reliable channels***

Terminating Reliable Broadcast (TRB)

Need for stronger RB

- In a chat application
  - clients don't know when or if a message will be delivered

- But in some applications that use RB
  - Some server uses RB and clients await delivery
  - How long should clients await delivery?
  - TRB provides the solution
    - Clients wait until they receive a message!

Terminating Reliable Broadcast

- Intuition
  - TRB is reliable broadcast in which
    - Sender broadcasts M
    - Receivers await delivery M
    - All nodes either deliver M or "abort"
- "Abort" indicated by special <SF> message
  - "Sender Faulty": delivered by clients that don't deliver a message

TRB Interface (1)

- **Module:**
  - Name: TerminatingReliableBroadcast (trb)

- ***Events***
  - Request: ⟨trbBroadcast | src, m⟩
    - Called by all nodes. If src≠self then m=nil

  - Indication: ⟨trbDeliver | src, m⟩
    - m may be <SF> (sender faulty) if src crashes

- ***Property:***
  - ***TRB1-TRB4***

TRB Interface (2)

- **Termination:**
  - Every correct node eventually delivers one message
- **Validity**:
  - If correct src sends m, then src will deliver m
- **Uniform agreement**:
  - If any node delivers m, then every correct node eventually delivers m
- **Integrity** (no creation):
  - If a node delivers m, then either m=<SF> or m was broadcast by src

Consensus-based Implementation

- Src RB broadcast m
  - Deliver <SF> if src is suspected by P

- Caveat
  - Src crash,
    - Some get m before detected crash
    - Some detect crash before getting m (with P!)

- Intuitive idea
  - Src BEB broadcast m
  - Nodes propose (consensus) whichever comes first:
    - Crash suspicion of src (<SF>) (from failure detector P)
    - BEB delivery from src (M)
  - Deliver consensus decision

TRB Correctness

- Intuitive correctness
  - If src correct, everyone gets m, and consensus decides m
- **Termination:**
  - Completeness of P and validity of BEB ensure a propose
  - Termination of consensus ensures a delivery
- **Validity**:
  - Assume a correct src sends m
  - All nodes get m (BEB validity) before suspecting src (P accuracy)
  - All propose m
  - All decide m (Consensus termination and validity)
- **Uniform agreement**:
  - By agreement of consensus
- **Integrity** (no creation):
  - Validity of consensus and no creation of BEB ensure <SF> or m is delivered

Hardness of TRB (1)

- Can we implement TRB in asynchronous systems? [d]
  - No, because Consensus is reducible to TRB
  - I.e., Consensus$\preccurlyeq$TRB
    - Why does this imply impossibility of implementing TRB?

- Given TRB, implement Consensus
  - Each node TRB its proposal
  - Save delivered values in a vector
  - Decide using a deterministic function
    - E.g. median, majority, or first non <SF> msg

Hardness of TRB (2)

- Can we implement TRB in partially synchronous systems? [d]
  - No, because P is reducible to TRB (P$\preccurlyeq$TRB)
    - That is, given TRB we can implement P
    - Furthermore: since TRB$\preccurlyeq$P we have TRB$\simeq$P
- Given TRB, implement P
  - Each node TRB heartbeats all the time
  - If ever receive <SF> for a node, suspect it

Hardness of TRB (3)

- **Accuracy**
  - TRB guarantees:
    - If src is correct, then all correct nodes will deliver m (validity and agreement)
  - Contrapositive
    - If any correct node doesn't deliver m, src has crashed
    - <SF> delivery implies src is dead (like <crash> for P!)

- **Completeness**
  - If source crashes, eventually <SF> will be delivered (integrity)

TRB requires synchrony!

Group Membership

- **A**

- Who is there?

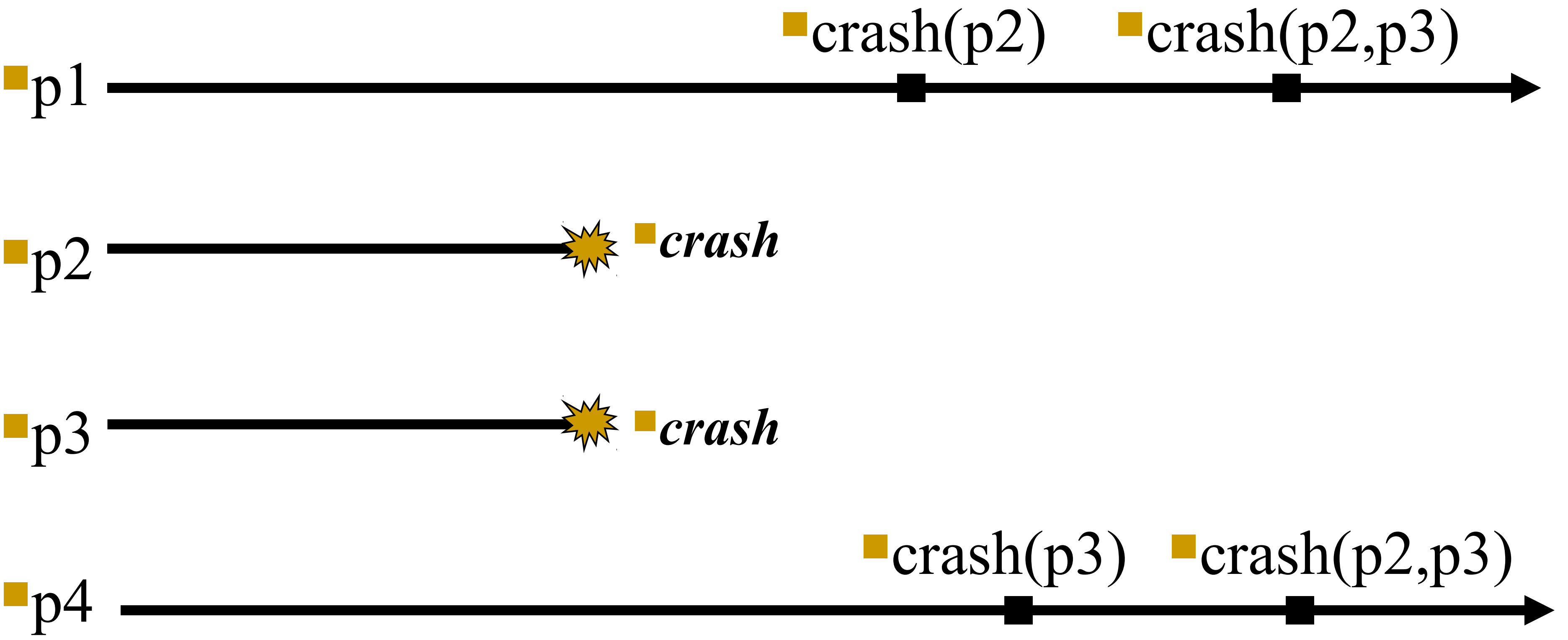
- **B**

- **C**

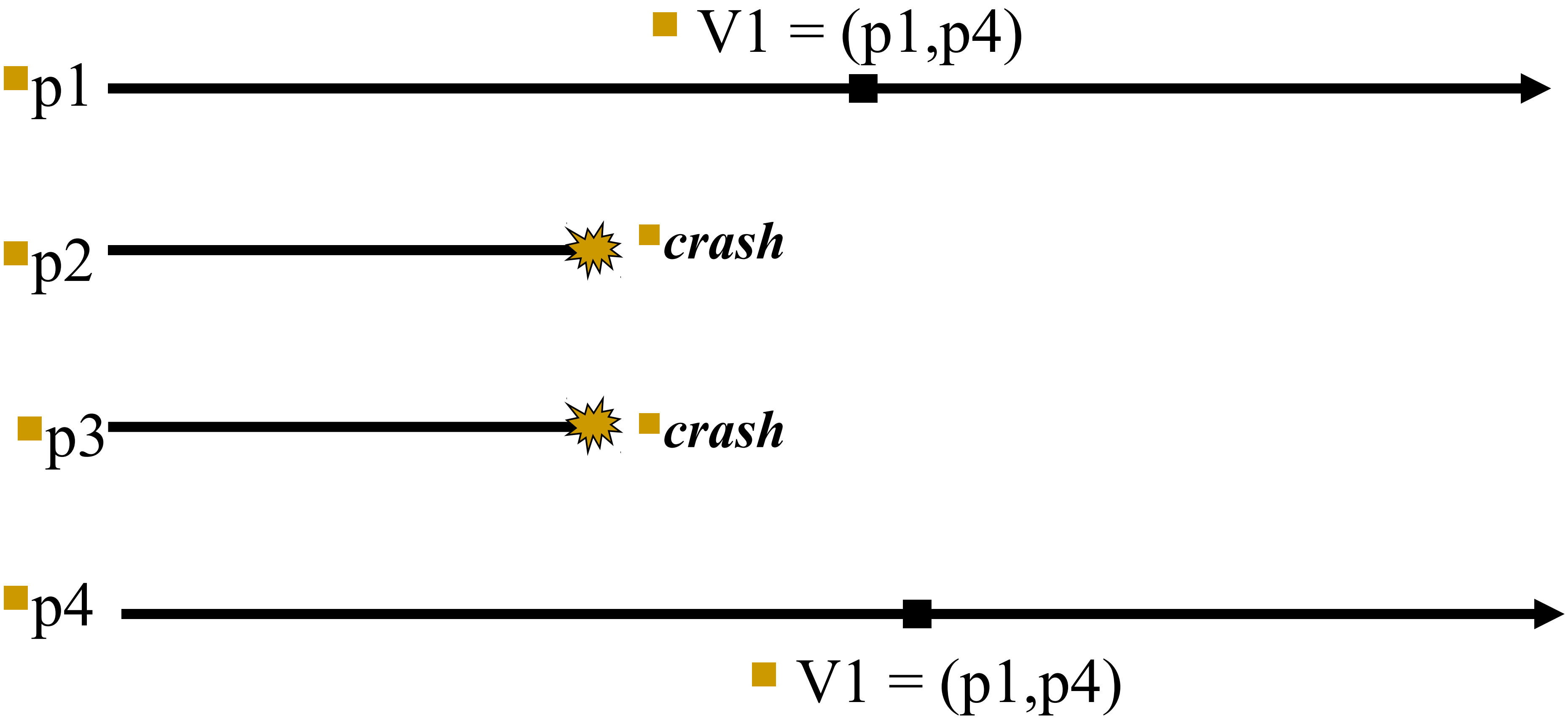
Group Membership

- In some distributed applications, processes need to know which processes are ***participating*** in the computation and which are not
- Failure detectors provide such information; however, that information is ***not coordinated*** (see next slide) even if the failure detector is perfect

Perfect Failure Detector



Group Membership



Group Membership

- To illustrate the concept, we focus here on a group membership abstraction to coordinate the information about ***crashes***
- In general, a group membership abstraction can also be used to coordinate the processes ***joining*** and ***leaving*** explicitly the set of processes (i.e., without crashes)

Group Membership

- ***Like*** a failure detector, the processes are informed about failures; we say that the processes ***install views***
- ***Like*** a perfect failure detector, the processes have accurate knowledge about failures
- ***Unlike*** a perfect failure detector, the information about failures are ***coordinated***: the processes install the same sequence of views

Group Membership

- ***Memb1. Local Monotonicity:*** If a process installs view (j,M) after installing (k,N), then j > k and M $\subset$ N
  - (This property does not hold in the general case where processes may join)
- ***Memb2. Agreement:*** No two processes install views (j,M) and (j,M') such that M $\neq$ M'
- ***Memb3. Completeness:*** If a process p crashes, then there is an integer j such that every correct process eventually installs view (j,M) such that p is not in M
- ***Memb4. Accuracy:*** If some process installs a view (i,M) and p is not in M, then p has crashed
  - (In the general case this might not be true)

Group Membership

- ***Events***
  - Indication: ⟨membView V⟩

- ***Properties:***
  - ***Memb1, Memb2, Memb3, Memb4***

Algorithm (gmp)

- **Implements:** groupMembership (gmp)
- **Uses:**
  - PerfectFailureDetector (P)
  - UniformConsensus(Ucons)
- **upon event** ⟨Init⟩ **do**
  - view := (id:0, memb:Π)
  - correct := Π
  - wait := false

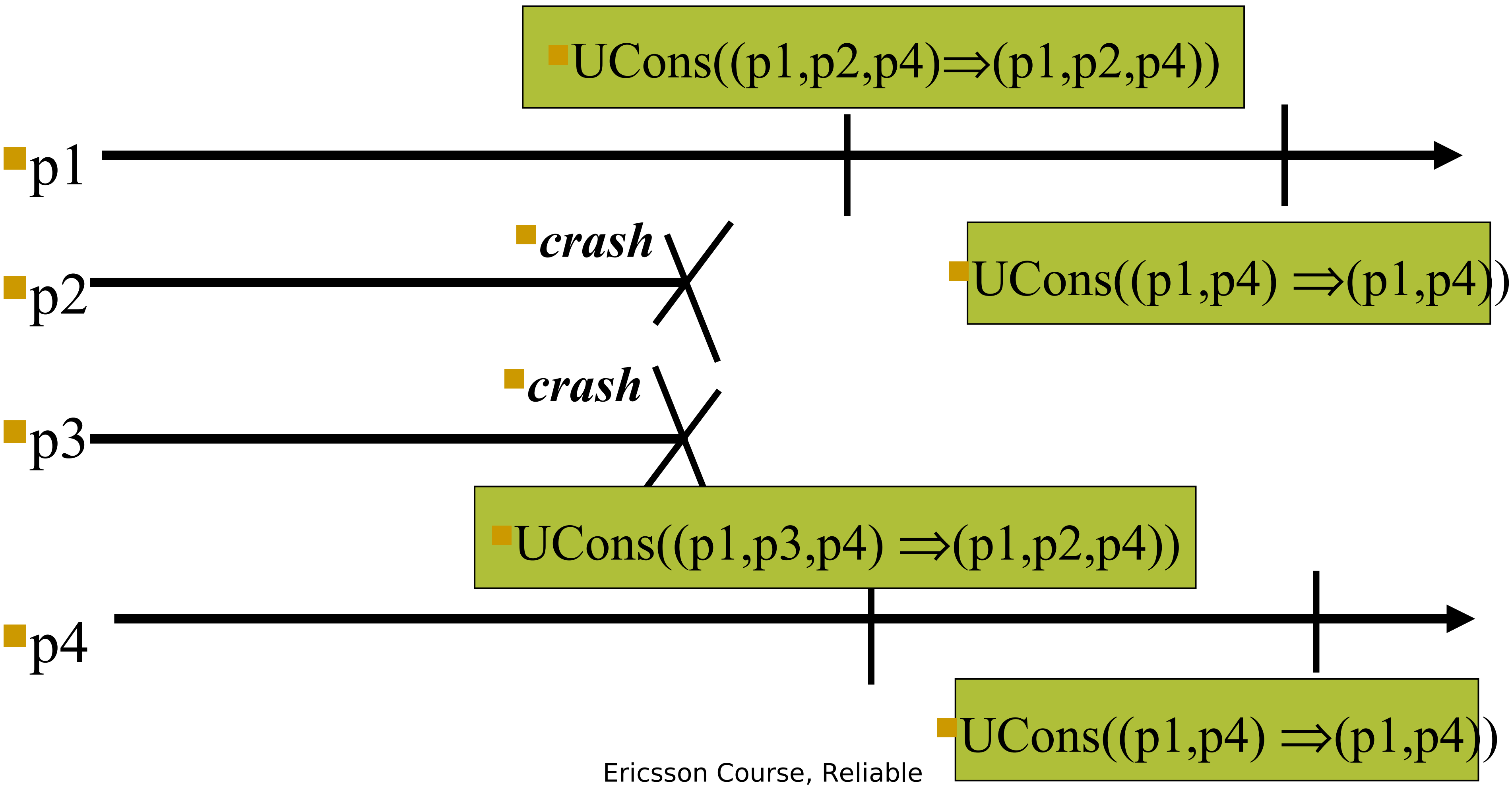
Algorithm (gmp)

- **upon event** ⟨crash pi⟩ **do**
  - correct := correct \ { pi }
- **upon event** (correct $\subset$ view.memb) **and** (wait = false) **do**
  - wait := true
  - **trigger** ⟨ucPropose (view.id + 1, correct) ⟩

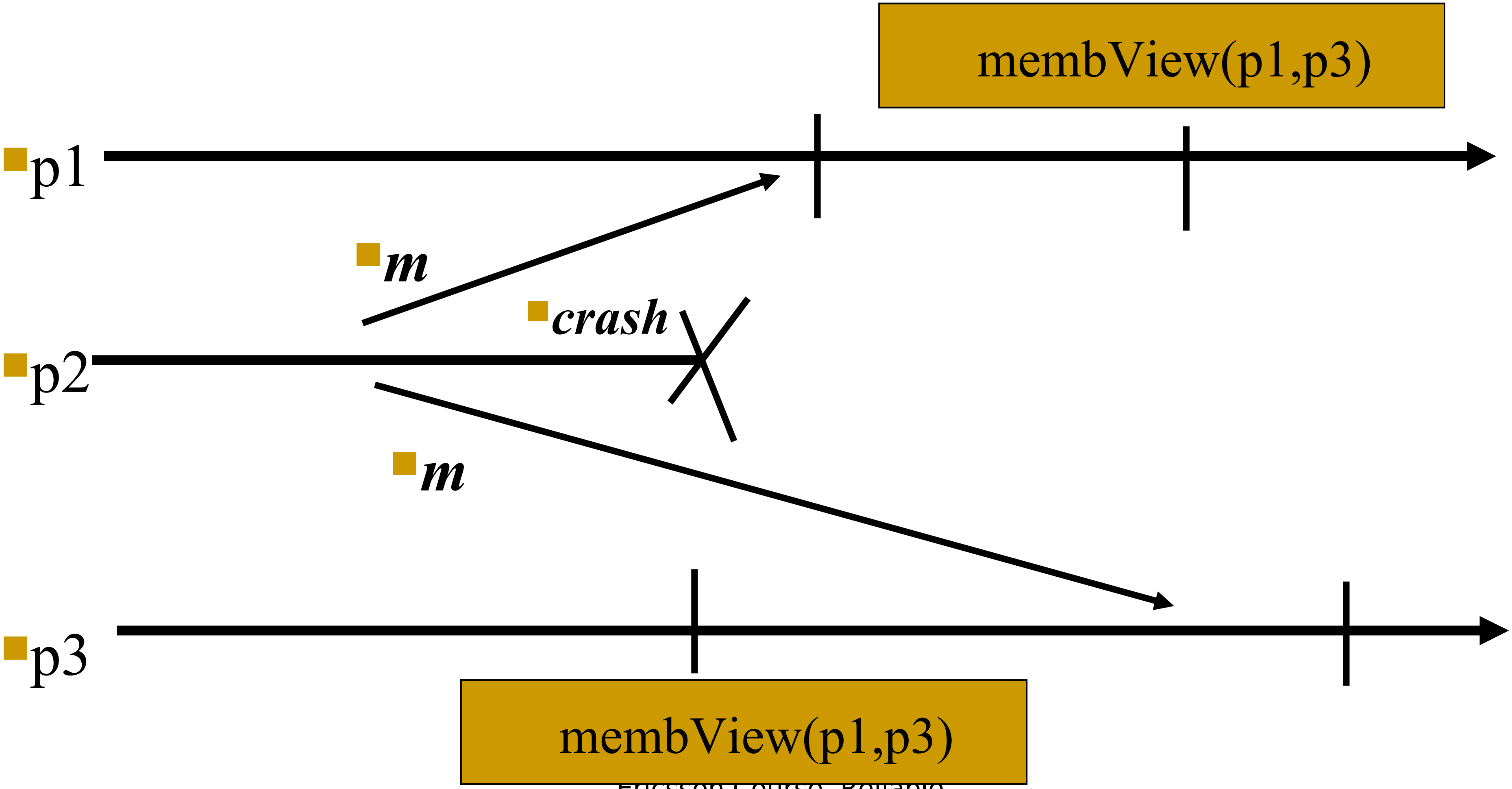
Algorithm (gmp)

- **upon event** ⟨ucDecided (i, m)⟩ **do**
  - view := (id:i, memb:m)
  - wait := false
  - **trigger** ⟨membView view⟩

Algorithm (gmb)



Group Membership Broadcast



Non-blocking Atomic Commit

- **Module:**
  - **Name:** Non-BlockingAtomicCommit (nbac).
- **Events:**
  - **Request:** ⟨nbacPropose | v⟩ : Used to propose a value for the commit (0 or 1).
  - **Indication:** ⟨nbacDecide | v⟩ : Used to indicate the decided value for nbac.

Non-blocking Atomic Commit

- **NBAC1:** Uniform Agreement:
  - No two processes decide different values.
- **NBAC2:** Integrity:
  - No process decides two values.
- **NBAC3:** Abort-Validity:
  - 0 can only be decided if some process proposes 0 or crashes.
- **NBAC4:** Commit-Validity:
  - 1 can only be decided if no process proposes 0.
- **NBAC5:** Termination:
  - Every correct process eventually decides.