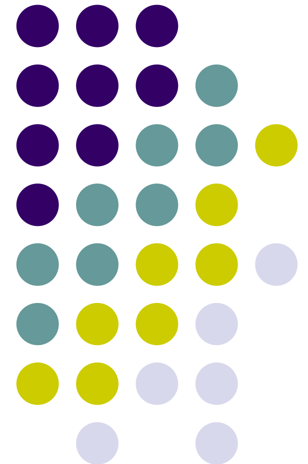
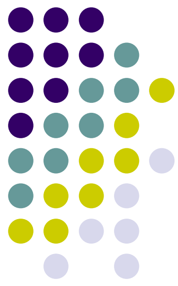
Distributed Algorithms for Building Reliable Systems

Seif Haridi

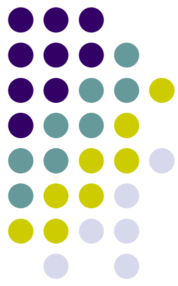
Causal Broadcast





Overview

- **Intuitions:** why causal broadcast?
- **Specifications** of ***causal broadcast***
- **Algorithms:**
  - A ***non-blocking*** algorithm using the ***past*** and
  - A ***blocking*** algorithm using ***vector clocks***



Broadcast

A

m

deliver

B

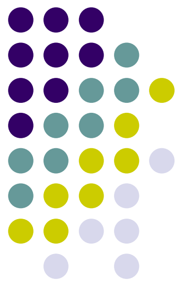
broadcast

m

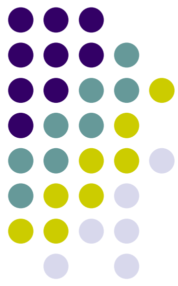
deliver

C

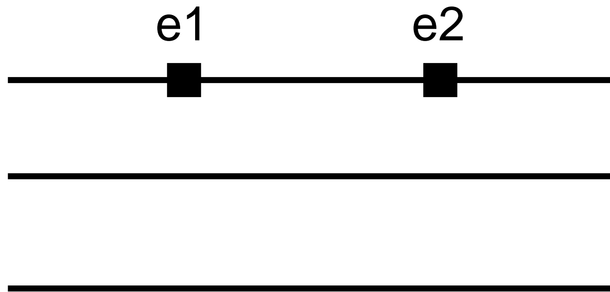
Cause-effect relations in message passing systems

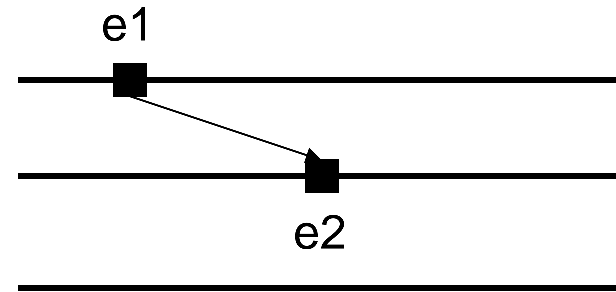


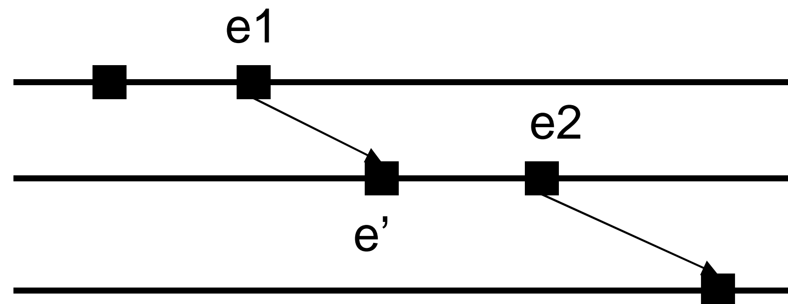
- An event e1 may potentially have caused another event e2 if the following relation, called, *happens-before* and denote by e1 $\rightarrow$ e2 holds
  - e1 and e2 occurs at the same process p, and e1 occurs before e2
  - e1 is the transmission of a message m at process p and e2 is the reception of the same message at process q
  - There exist some event e' such that e1$\rightarrow$e' and e'$\rightarrow$e2



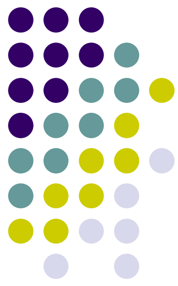
Happens-before relation





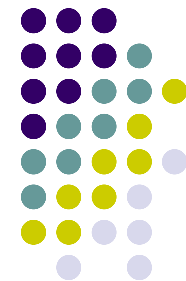


Intuitions (1)



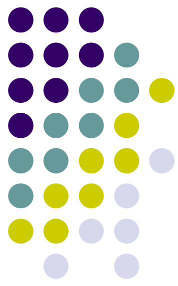
- So far, we did not consider ordering among messages; In particular, we considered messages to be independent
- Two messages from the same process might not be delivered in the order they were broadcast
- A message m1 that causes a message m2 might be delivered by some process after m2

Intuitions (2)



- Consider a replicated distributed bulletin board system that manages two types of news
  - Proposals and comments
  - Every new event that is displayed contains a reference to the event that *caused* it, e.g., a comment includes a reference to the actual proposal
- Even uniform reliable broadcast does not guarantee such a dependency of delivery
- Causal broadcast alleviates the need for the application to deal with such dependencies

Modules of a process



Applications

indication

request

(R-U) Causal broadcast

request

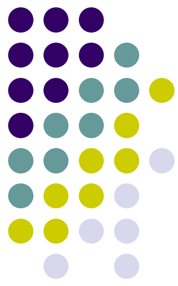
indication

indication

(R-U)Reliable broadcast

Failure detector

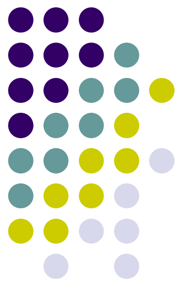
Channels



Overview

- **Intuitions:** why causal broadcast?
- **Specifications** of ***causal broadcast***
- **Algorithms:**
  - A ***non-blocking*** algorithm using the ***past*** and
  - A ***blocking*** algorithm using ***vector clocks***

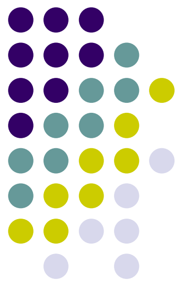
Causal broadcast



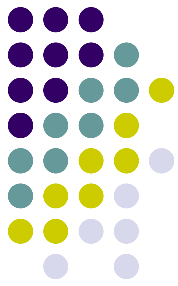
- ***Events***
  - Request: ⟨coBroadcast m⟩
  - Indication: ⟨coDeliver src, m⟩

- ***Property:***
  - ***Causal Order (CO)***

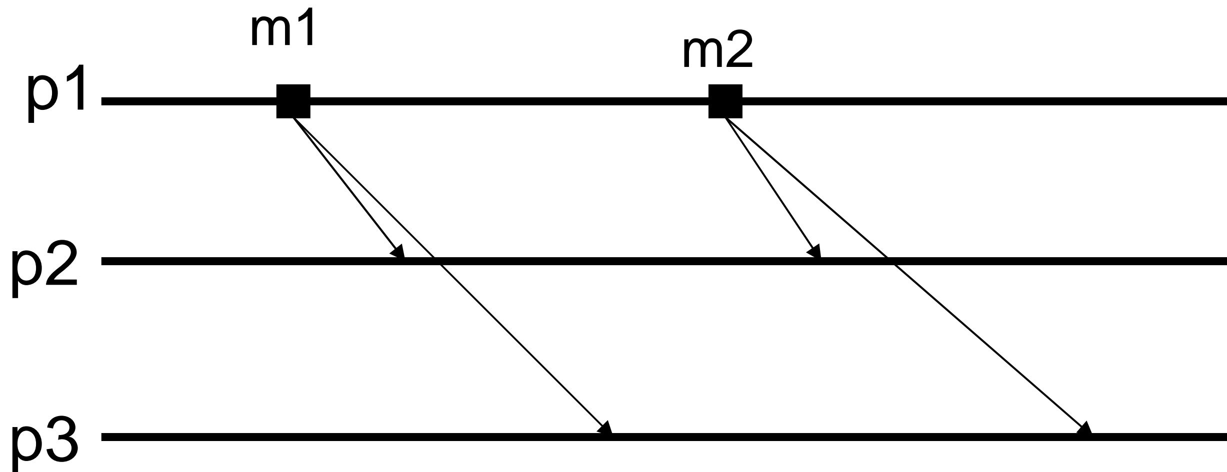
Causality



- ***Let m1 and m2 be any two messages: $m1 \rightarrow m2$ (m1 causally precedes m2) iff***
  - **C1 (FIFO order)**. Some process p_i broadcasts m1 before broadcasting m2
  - **C2 (Local order)**. Some process p_i delivers m1 and then broadcasts m2
  - **C3 (Transitivity)**. There is a message m′ such that $m1 \rightarrow m'$ and $m' \rightarrow m2$

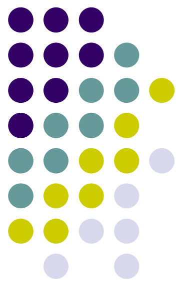


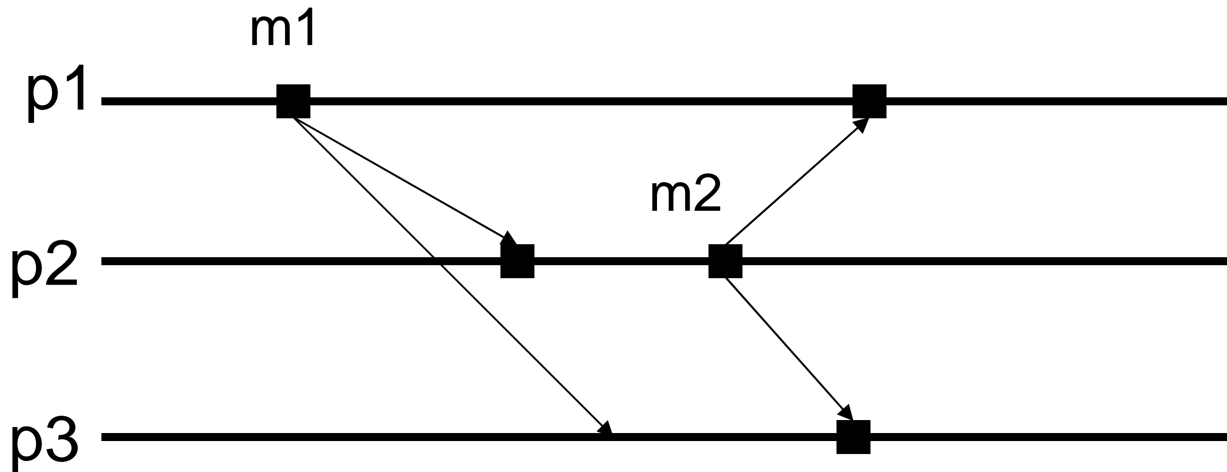
Causality



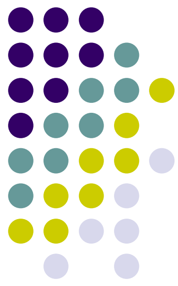
C1 (FIFO order).

Causality

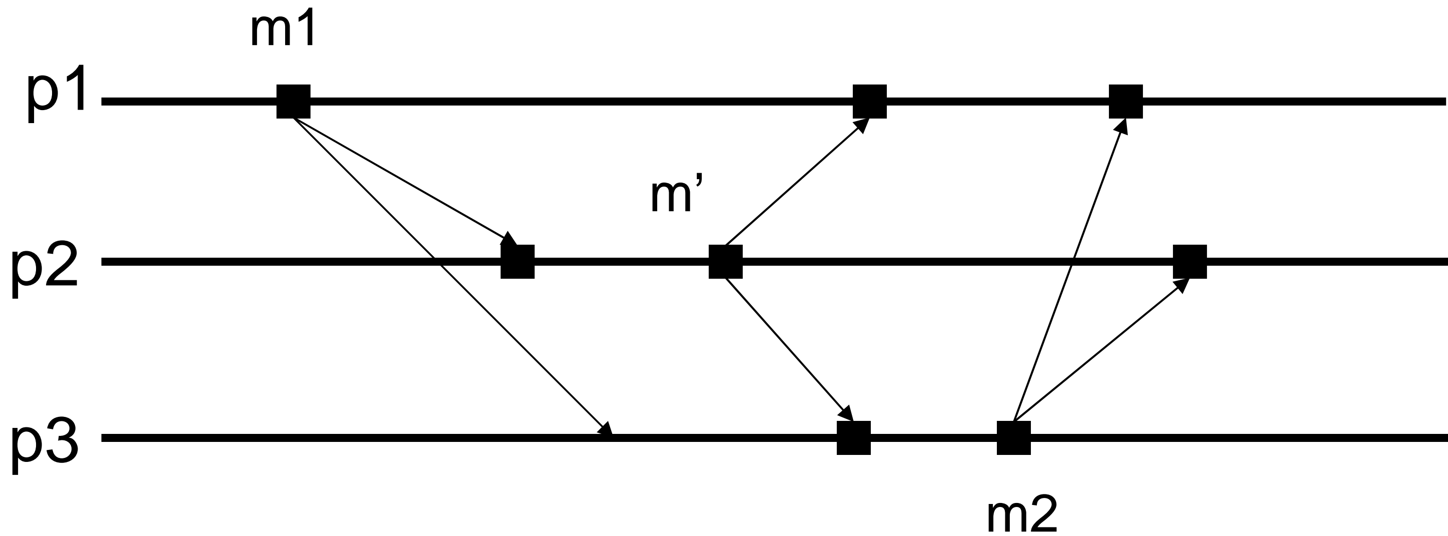




C2 (Local order).

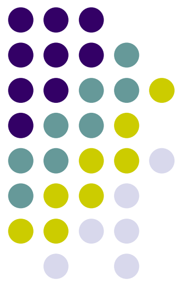


Causality



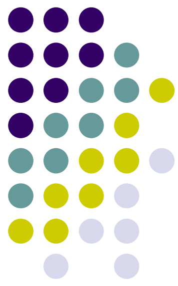
- **C3 (Transitivity).**

Causal broadcast



- *Events*
  - Request: ⟨coBroadcast m⟩
  - Indication: ⟨coDeliver src, m⟩
- ***Property:***
  - ***CO***: If any process p_i delivers a message m2, then p_i must have delivered every message m1 such that $m1 \rightarrow m2$

Causality ?
d(elivery) b(roadcast)



b(m1) d(m1) b(m2) d(m2)

p1

d(m2)

p2

d(m1)

m2

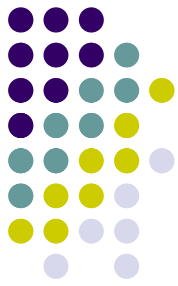
m1

p3

d(m2)

d(m1)

Causality ?
d(elivery) b(roadcast)



b(m1) d(m1) d(m2)

p1

p2

d(m1) b(m2)

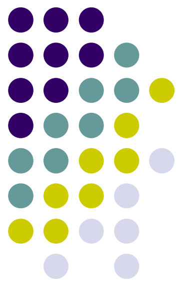
m2

m1

p3

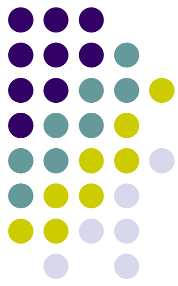
d(m2) d(m1)

Reliable causal broadcast (rcb)



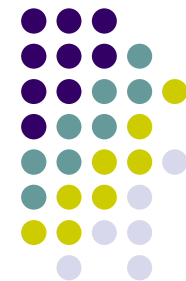
- ***Events***
  - Request: ⟨rcoBroadcast m⟩
  - Indication: ⟨rcoDeliver src, m⟩
- ***Properties:***
  - ***RB1, RB2, RB3, RB4 +***
  - ***CO***

Uniform causal broadcast (ucb)



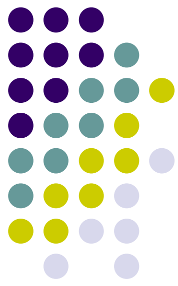
- ***Events***
  - Request: <ucoBroadcast, m>
  - Indication: <ucoDeliver, src, m>
- ***Properties:***
  - ***URB1, URB2, URB3, URB4 +***
  - ***CO***

Reliable broadcast (rb)

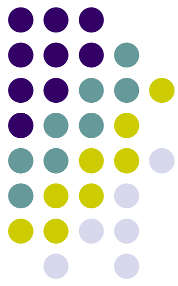


- ***Properties***
  - ***RB1. Validity.*** If a correct process pi broadcasts a message m, then pi eventually delivers m.
  - ***RB2 = BEB2.***
  - ***RB3 = BEB3.***
  - ***RB4. Agreement:*** For any message m, if a **correct process delivers** m, then every correct process delivers m

Uniform broadcast (urb)



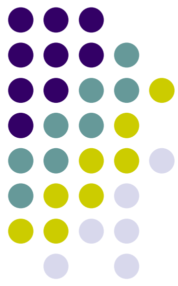
- ***Properties***
  - ***URB1 = BEB1.***
  - ***URB2 = BEB2.***
  - ***URB3 = BEB3.***
  - ***URB4. Uniform Agreement:*** For any message m, if a **process delivers** m, then every correct process delivers m



Overview

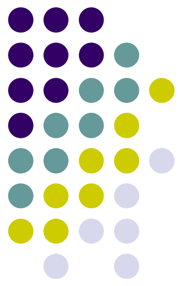
- **Intuitions:** why causal broadcast?
- **Specifications** of ***causal broadcast***
- **Algorithms:**
  - A ***non-blocking*** algorithm using the ***past*** and
  - A ***blocking*** algorithm using ***vector clocks***

Algorithms



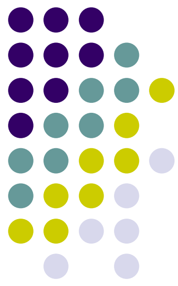
- We present **reliable causal broadcast** algorithms using **reliable broadcast** underlying abstractions
- We obtain **uniform causal broadcast** algorithms by using instead an underlying **uniform reliable broadcast**

Algorithm1 Fail-Silent No-Waiting Causal Broadcast



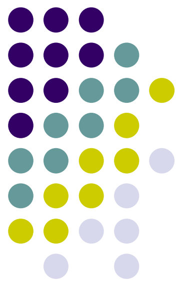
- Whenever a process rbDelivers m, it coDelivers m (if not already coDelivered)
- Each message m carried an ordered list of messages $past_m$ that causally precede m
- Before m is coDelivered, $past_m$ is inspected
- Messages in $past_m$ that are not coDelivered are coDelivered before m

Algorithm 1
Fail-silent No Waiting Causal Broadcast



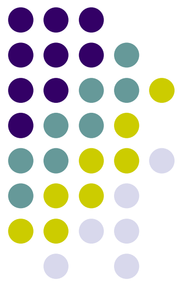
- **Implements:** ReliableCausalOrderBroadcast (rco)
- **Uses:** ReliableBroadcast (rb).
- **upon event** ⟨Init⟩ **do**
  - delivered := ∅; past := nil
- **upon event** ⟨rcoBroadcast m⟩ **do**
  - **trigger** ⟨rbBroadcast (DATA, past, m)⟩
  - **past := past ++ (pi, m)**

Algorithm 1 (cont'd)



- **upon event** ⟨rbDeliver pi, (DATA, $past_m$, m)⟩ **do**
  - **if** m $\notin$ delivered **then**
    - **forall** (s_n, n) $\in$ $past_m$ **do** // in ascending order
      - **if** n $\notin$ delivered **then**
      - **trigger** ⟨rcoDeliver s_n, n⟩
      - delivered := delivered $\cup$ {n}
      - past := past ++ (s_n, n) // sequence (list) append
    - **trigger** ⟨rcoDeliver pi, m⟩
    - delivered := delivered $\cup$ {m}
    - past := past ++ (pi, m) // sequence (list) append

Algorithm 1



coB(m1) coD(m1) coD(m2)

p1

m2(m1)

m1

coB(m2) coD(m2)

p2

coD(m1)

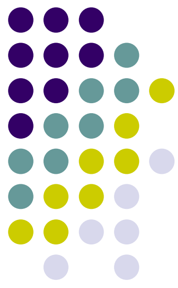
m1

m2(m1)

p3

rbD(m2) coD(m1) coD(m2)

Algorithm 1



coB(m1) coB(m2) coD(m1) coD(m2)

p1

m2(m1)

m1

coD(m2)

p2

coD(m1)

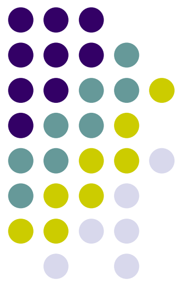
m1

m2(m1)

p3

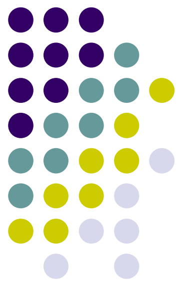
rbD(m2) coD(m1) coD(m2)

Algorithm 1 comments



- Consider the sequence of events at p1
  - $\langle$rcoB m1$\rangle_{p1}$, $\langle$rcoB m2$\rangle_{p1}$,
    $\langle$rbD p1, (DATA, $past_{m2}$, m2)$\rangle_{p1}$,
    $\langle$rbD p1, (DATA, $past_{m1}$, m1)$\rangle_{p1}$
  - What is the order of co-delivery of m1 and m2 at p1
- We know that reliable broadcast (rb) has the no-duplicate property why in this case we need the test

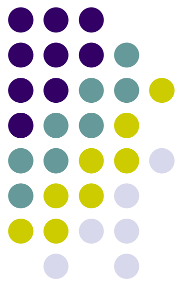
  if m $\notin$ delivered **then** … ?



Uniformity

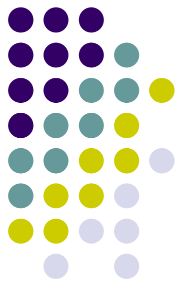
- Algorithm 1 ensures causal reliable broadcast
- If we replace reliable broadcast with uniform reliable broadcast, Algorithm 1 would ensure uniform causal broadcast

Algorithm 1’ Garbage collection of past



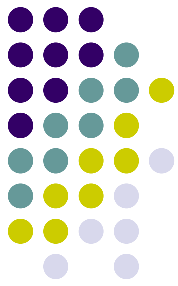
- Assume a fail-stop model, and use a perfect failure detector
- When a process rbDelivers *m*, it rbBroadcasts an ACK_m to all processes
- When ACK_m is rbDelivered from all correct processes m is purged from *past*

Algorithm 1′ (gc)



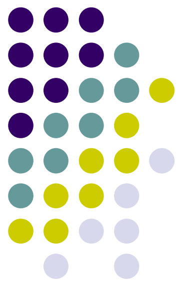
- **Implements:** GarbageCollection (+ Algo 1)
- **Extends:** Algorithm 1
- **Uses:**
  - ReliableBroadcast (rb)
  - PerfectFailureDetector(P)
- **upon event** ⟨Init⟩ **do**
  - delivered := ∅; past := nil
  - correct := Π
  - ack[m] := ∅ (for all m)

Algorithm 1' (gc – cont'd)



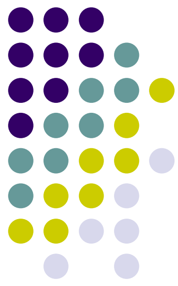
- **upon event** ⟨crash pi⟩ **do**
  - correct := correct \ {pi}
  - **if exists** m **such that** correct ⊆ ack[m] **then**
    - past := past \ {(sm, m)}
- **upon exists** m ∈ delivered **such that** self ∉ ack[m] **do**
  - ack[m] := ack[m] ∪ {self}
  - **trigger** ⟨rbBroadcast (ACK, m)⟩

Algorithm 1' (gc – cont'd)



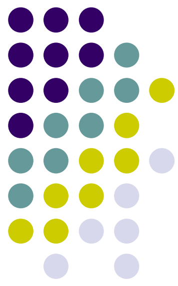
- **upon event** ⟨rbDeliver pi, (ACK, m)⟩ **do**
  - ack[m] := ack[m] $\cup$ {pi}
  - **if** correct $\subseteq$ ack[m] **then**
    - past := past \ {(s_m, m)}

Algorithm 1′ (gc – cont'd) Alternative formulation



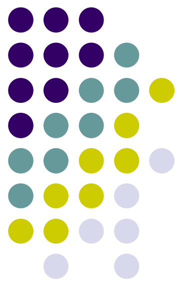
- **upon event** ⟨crash pi⟩ **do**
  - correct := correct \ {pi}
  - **if** correct ⊆ ack[m] **then**
    - past := past \ {(sm, m)}
- **upon** ⟨rcoDeliver pi, m⟩ **do**
  - **trigger** ⟨rbBroadcast (ACK, m)⟩

Algorithm 1' (gc – cont'd) Alternative formulation



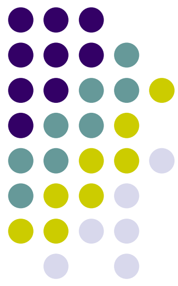
- **upon event** ⟨rbDeliver pi, (ACK, m)⟩ **do**
  - **if** pi $\in$ correct **then**
    - ack[m] := ack[m] $\cup$ {pi}
    - **if** correct $\subseteq$ ack[m] **then**
      - past := past \ {(s_m, m)}
      - delete ack[m]

Fail-Silent Algorithm Waiting Causal Broadcast



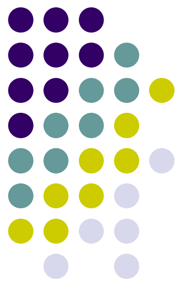
- Represent the past by a vector of sequence numbers (array of integers) called vector clock (VC)
- rank(pi) = i, convert the process identifier to an integer that is used to index the vector
- At process pi
  - VC[i]: the number of messages pi coBroadcasted
  - VC[j], $j \neq i$: the number of messages pi coDelivered from pj

Fail-Silent Algorithm Waiting Causal Broadcast



- This vector is attached to each message m that p coBroadcasts
- A process q that rbDelivers m compares this vector with its own vector to determine which messages are missing, and from which processes
- Process q does does not coDeliver m until the missing messages are coDelivered (Waiting)

Algorithm 2
rcod(elivery) rcob(roadcast)



p1 (0,0,0) b(m1) (1,0,0) d(m1) b(m2) (2,0,0) d(m2)

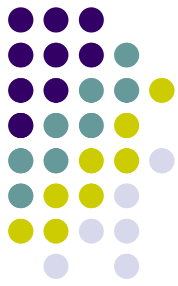
m1(0,0,0) m2(1,0,0)

p2 (0,0,0) d(m1) (1,0,0) d(m2)

m2(1,0,0) m1(0,0,0)

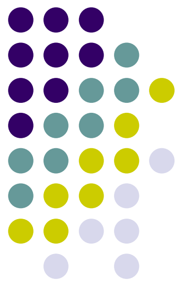
p3 (0,0,0) d(m1) (1,0,0) d(m2) (2,0,0)

Algorithm 2



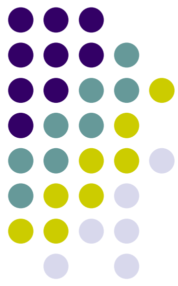
- **Implements:** ReliableCausalOrderBroadcast (rco)
- **Uses:** ReliableBroadcast (rb)
- **upon event** ⟨Init⟩ **do**
  - **forall** pi $\in \Pi$ **do** VC[rank(pi)] := 0
- **upon event** ⟨rcoBroadcast m⟩ **do**
  - **trigger** ⟨rbBroadcast (DATA, VC, m)⟩
  - VC[self] := VC[self] + 1
  - **trigger** ⟨rcoDeliver self, m⟩

Algorithm 2



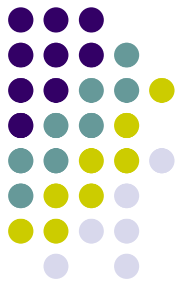
- **upon event** ⟨rbDeliver pj, (DATA, VC_m , m)⟩ **do**
  - **if** pj ≠ self **then**
    - pending := pending ∪ (pj, (DATA, VC_m, m))
    - deliver-pending

Algorithm 2

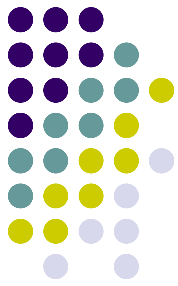


- **procedure** deliver-pending
  - **while exists** $x=(s_m, (DATA, VC_m, m)) \in$ pending **such that** $VC \geq VC_m$ **do**
    - pending := pending \ $(s_m, (DATA, VC_m, m)$
    - $VC[\ rank(s_m)\ ] := VC[\ rank(s_m)\ ] + 1$
    - **trigger** $\langle$rcoDeliver s_m, m$\rangle$

Algorithm 2
rco**d**(elivery) rco**b**(roadcast)

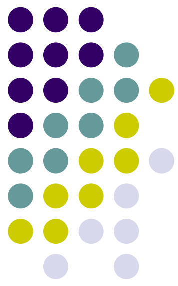


(0,0,0)
(1,0,0)
(2,0,0)
b(m1) d(m1) b(m2) d(m2)
p1
m1(0,0,0)
m2(1,0,0)
(0,0,0)
d(m2)
p2
d(m1)
(1,0,0)
m2(1,0,0)
m1(0,0,0)
p3 (0,0,0)
d(m1) d(m2)
(1,0,0) (2,0,0)



# Algorithm 2
rcod(elivery) rcob(roadcast)

Algorithm 2
rcod(elivery) rcob(roadcast)



p1

(1,0) (1,1)

b(m1) d(m1) d(m2)

m2(0,0)

m1(0,0)

p2

b(m2) d(m2) d(m1)

(0,1) (1,1)

Notice different delivery order