Broadcast Communication

Ashvin Goel

Electrical and Computer Engineering
University of Toronto

Distributed Systems
ECE419

With thanks to Tim Harris and Martin Kleppmann,
Lecture notes on Concurrent and Distributed Systems

Overview

- What is broadcast communication?
- Broadcast models
- Broadcast algorithms

What is broadcast communication?

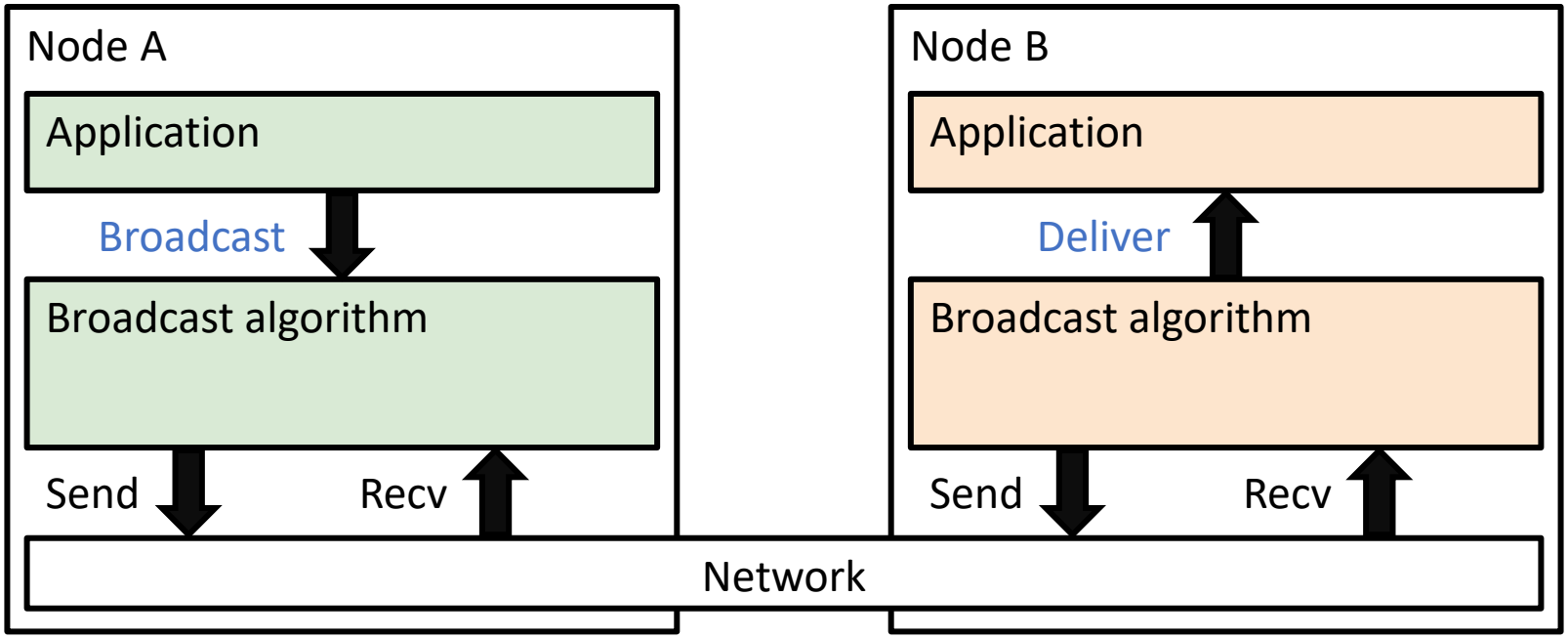
- Broadcast (multicast) is group communication:
  - Terminology: one node broadcasts message,
    all nodes in group deliver message to application
  - Set of group members may be fixed (static) or
    dynamic (nodes may join or leave the group)
  - If one node is faulty, remaining group members carry on
  - Assume point-to-point messages between nodes

Broadcast system model

- Links: best-effort (messages may be lost or duplicated) or reliable (correct nodes deliver message to all nodes)
- Nodes: may have crash-stop failures
- Timing: asynchronous or partially synchronous,
  i.e., no upper bound on message latency

Broadcast architecture

- A node sends a message to nodes in the group (including itself) by calling a broadcast algorithm
- Algorithm uses point-to-point send/recv messages, delivers message to application (possibly with delay)



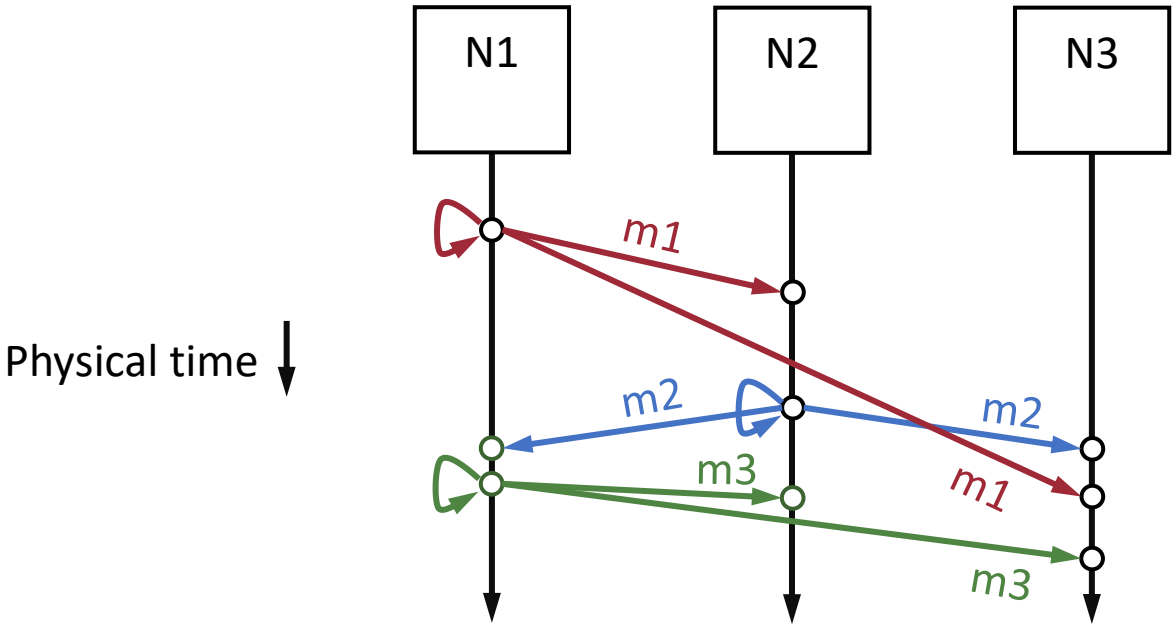
Broadcast Models

Broadcast models

- All the models below provide reliable broadcast, differ in delivery order of messages to nodes
- FIFO broadcast:
  - If m1 and m2 are broadcast by the same node, and broadcast(m1) → broadcast(m2), then m1 must be delivered before m2
- Causal broadcast:
  - If broadcast(m1) → broadcast(m2), then m1 must be delivered before m2
- Total order broadcast:
  - If m1 is delivered before m2 on one node, then m1 must be delivered before m2 on all nodes

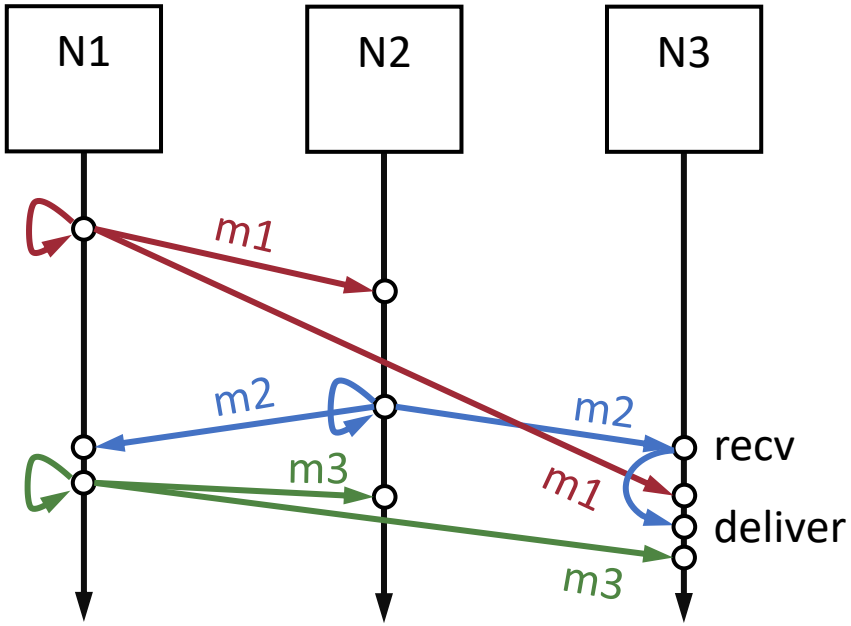
FIFO broadcast example

- N1 sends messages m1 then m3
- N2 sends message m2
- Is this a FIFO broadcast?

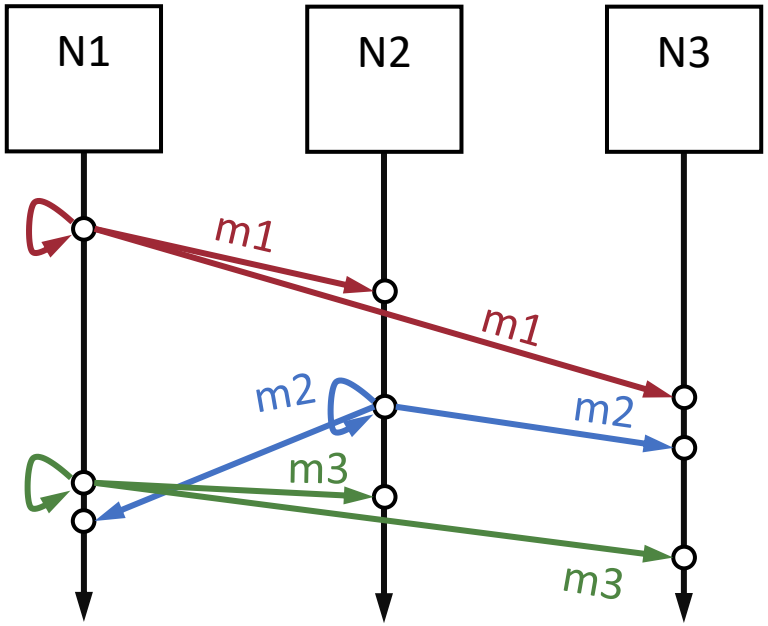


Is this a causal broadcast?

- How can we make it a causal broadcast?
- If m2 received before m1, delay delivery of m2 after m1
- Can we delay delivery of m2 after m3 on N3?

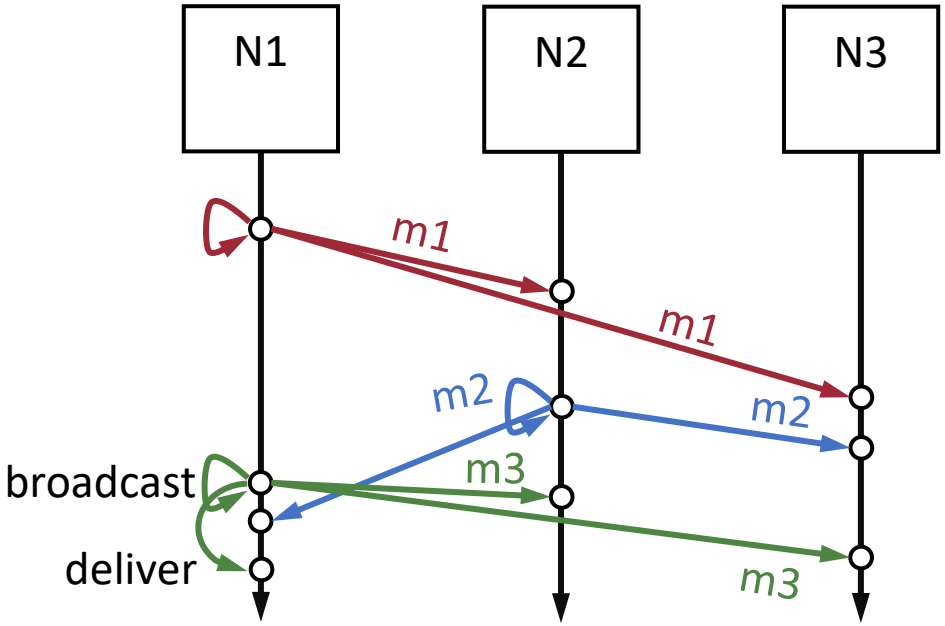


Is this a causal broadcast?



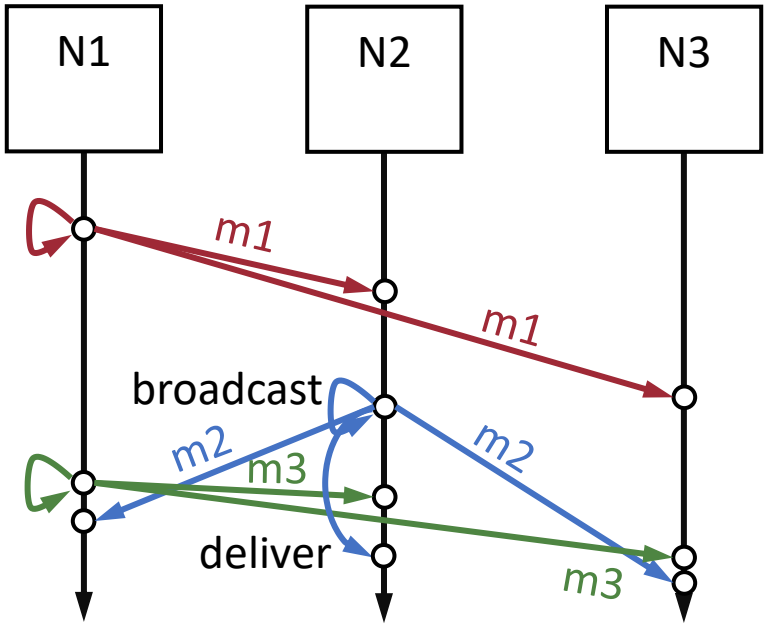
Is this a total-order broadcast?

- How can we make it a total-order broadcast?
- We can delay delivery of m3 at N1

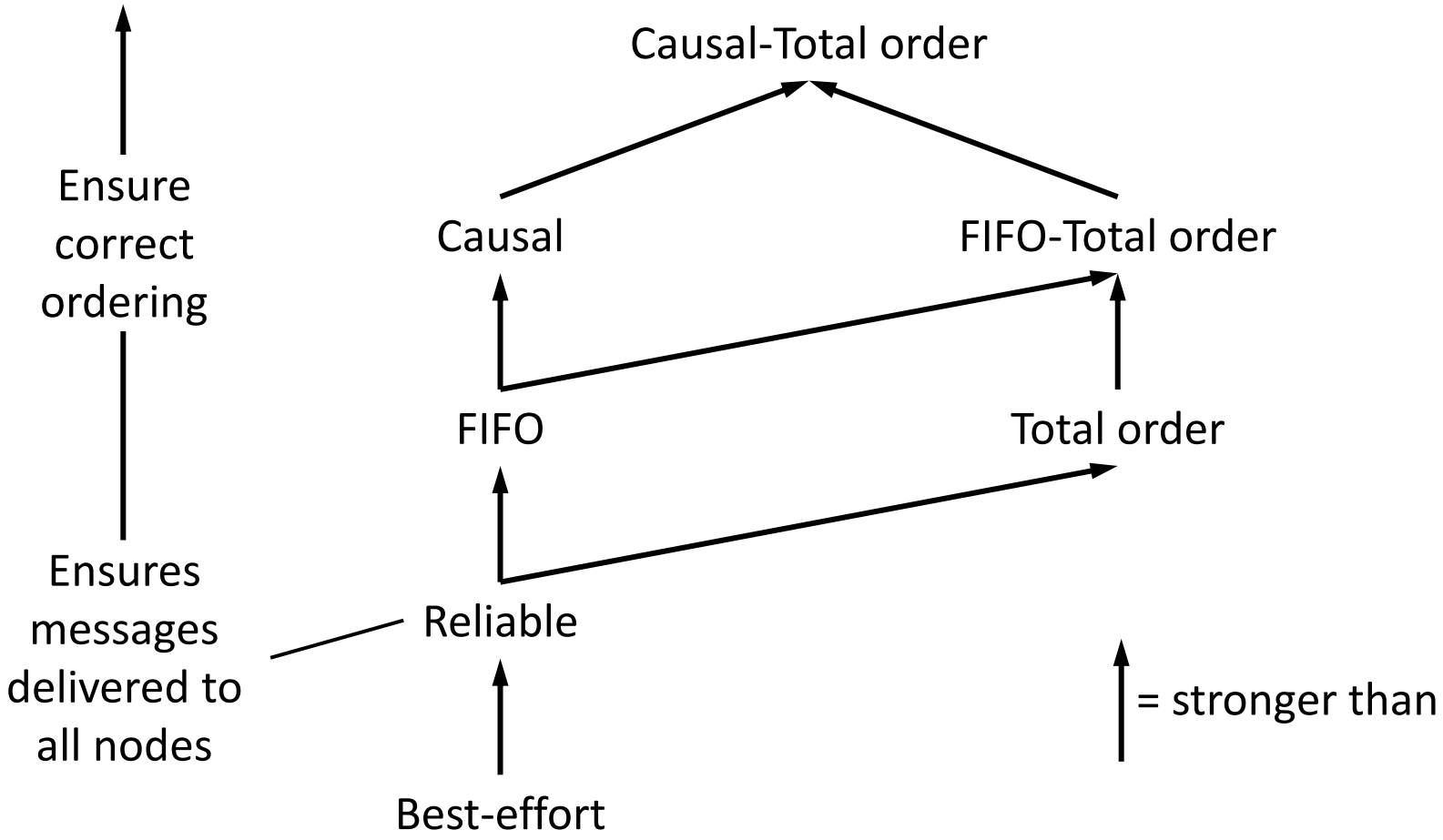


Is this a total-order broadcast?

- How can we make it a total-order broadcast?
- We can delay delivery of m2 at N2

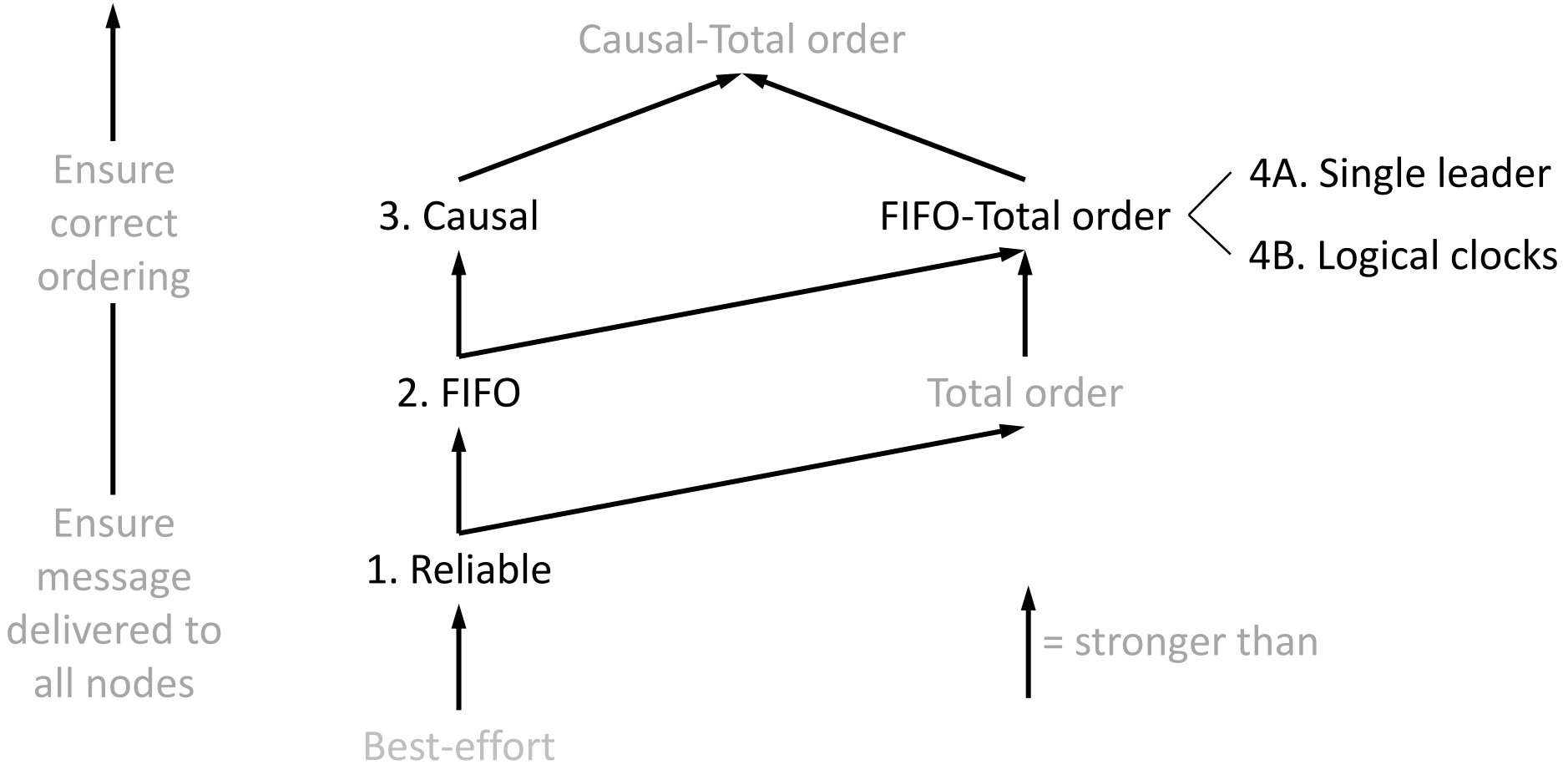


Relationship between broadcast models



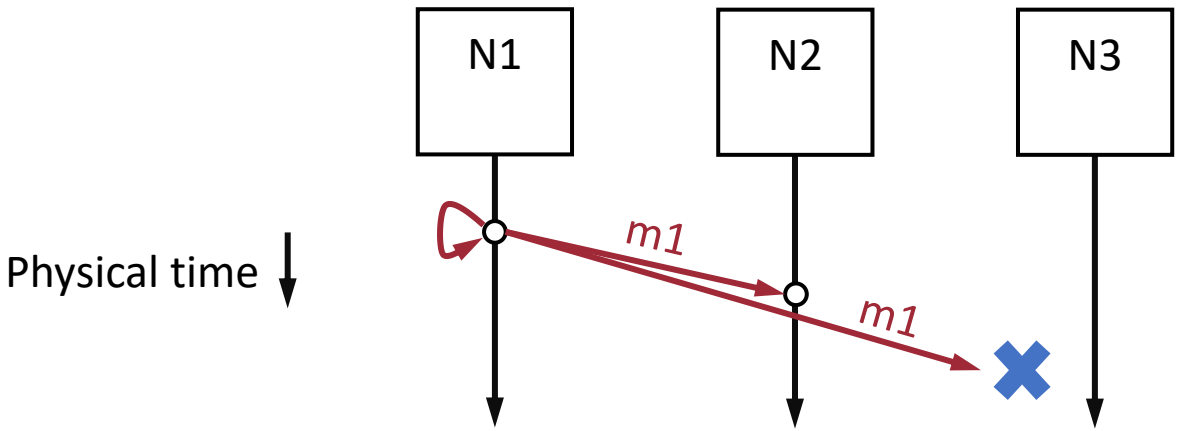
Broadcast Algorithms

Broadcast algorithms



1. Reliable broadcast - first try

- Broadcasting node sends message directly to each node
- Make best-effort broadcast reliable by retransmitting dropped messages and deduplicating messages
- Any problem?
  - Broadcasting node may crash before all messages are delivered



1. Reliable broadcast

- For reliable broadcast, we need help from other nodes
- Eager reliable broadcast
  - When a node receives a message for the first time, it forwards message to all other nodes
    - Pros: ensures that even if some nodes crash, the rest of the correct nodes will receive the message
    - Cons: inefficient, per broadcast, $O(n^2)$ forwarded messages
- Gossip protocol
  - Broadcast node sends to a fixed number of random nodes, when a node receives a message for the first time, it forwards message to a fixed number of random nodes
    - Pros: efficient, resilient to loss, crashes
    - Cons: only guarantees reliable delivery with high probability

2. Why FIFO Broadcast?

- With FIFO broadcast, messages from a node are delivered in order (reliably) to other nodes
  - Node 1
    - M1: I am going to Pub on College Street
    - M2: Meet me there
  - Node 2
    - If the node received M2 before M1, it is confused

2. FIFO broadcast - setup

- Assuming N nodes, each node maintains three variables:
  - seq: sequence number (count) of messages broadcast by this node
  - D: vector of size N, containing count of messages from each node delivered at this node
  - buffer: buffer for received messages
- sequence number and D vector at a node help track next message from a sender that can be delivered to the node
- buffer will buffer the messages received out-of-order

2. FIFO broadcast - algorithm

```
seq := 0;                  # count of messages broadcast by node
D := [0, 0, . . . , 0]; # count of messages delivered
buffer := {}               # buffer for received messages

when Node i broadcasts a message m:
    send msg = (i, seq, m) via reliable broadcast
    seq = seq + 1


when Node i receives a message msg:
    append(buffer, msg)
    foreach (msg = (j, seq, m) in buffer) and (seq == D[j]):
        deliver m to application
        remove(buffer, msg)
        D[j] = D[j] + 1
```

check whether msg from j is the next one in FIFO order

increase count of messages delivered from j

3. Why Causal Broadcast?

- FIFO broadcast assumes messages broadcast from different nodes are independent
- With causal broadcast,
  if broadcast of M1 happens before broadcast of M2,
  then M1 is delivered (reliably) before M2
  - Node 1
    - M1: I am going to Pub on College Street
  - Node 2
    - M1 delivered
    - M2: Let's meet at 8 pm (M2 causally depends on M1)
  - Node 3
    - If M2 delivered before M1, it is confused

3. Causal broadcast - setup

- Each node maintains the same three variables as FIFO:
  - seq: sequence number (count) of messages broadcast by this node
  - D: vector of size N, containing count of messages from each node delivered at this node
  - buffer: buffer for received messages
- Besides sending sequence number, also send D vector
  - Why send this vector?
  - This vector has some similarities to a vector clock (since both ensure causality) but it is updated differently

3. Causal broadcast - algorithm

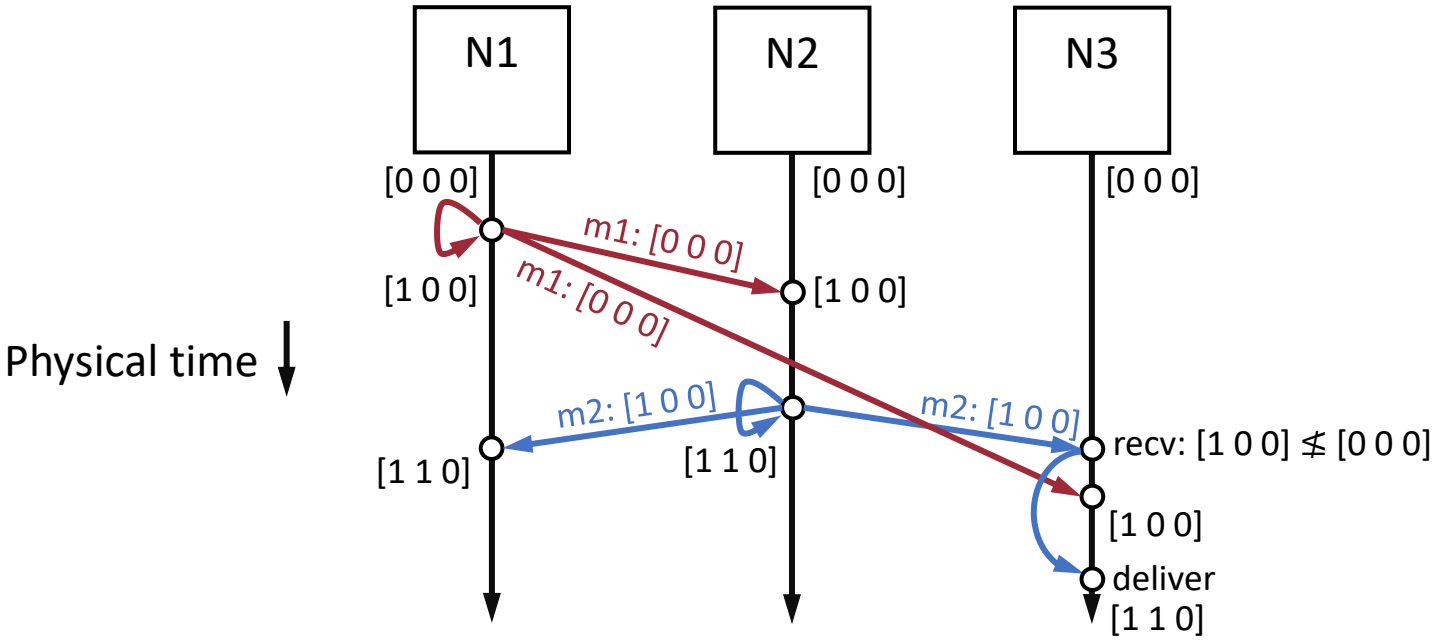
```
seq := 0;                  # count of messages broadcast by node
D := [0, 0, . . . , 0]; # count of messages delivered
buffer := {}               # buffer for received messages

when Node i broadcasts a message m:
    deps = D; deps[i] = seq;
    send msg = (i, deps, m) via reliable broadcast
    seq = seq + 1


when Node i receives a message msg:
    append(buffer, msg)
    foreach (msg = (j, deps, m) in buffer) and (deps <= D):
        deliver m to application
        remove(buffer, msg)
        D[j] = D[j] + 1
```

check whether dependencies in msg from j have been delivered by this node

increase count of messages delivered from j

3. Causal broadcast - example

- Vectors shown at nodes are the D (delivered) vector
- Vectors shown with messages are deps vector



4. Why Total Order Broadcast?

- Causal broadcast delivers message in causal (partial) order, but messages may be delivered by nodes in different order
  - E.g., independent M1 and M2 broadcast by Nodes A and B can be delivered to A and B in different order
- With total order broadcast, messages are delivered in same order on all nodes
  - By default, may not be FIFO or causal order

game score = 10	
Replica 1	Replica 2
M1: Add 1 to score M2: Double score	M2: Double score M1: Add 1 to score
game score is inconsistent	

4A. Total order broadcast: single leader

- One node is designated as leader (sequencer)
- To broadcast message, node sends it to the leader
- Leader broadcasts it via FIFO broadcast
  - Ensures FIFO-total order broadcast
- Assumption: leader does not crash
  - Leader crashes ⇒ no more messages delivered
  - Changing the leader safely is difficult
    - Who detects a leader failure?
    - What if a leader can communicate with some (but not all) clients?
    - What if a leader fails temporarily, another node is chosen as the new leader, and then the old leader starts responding again?

4B. Total order broadcast: logical clocks

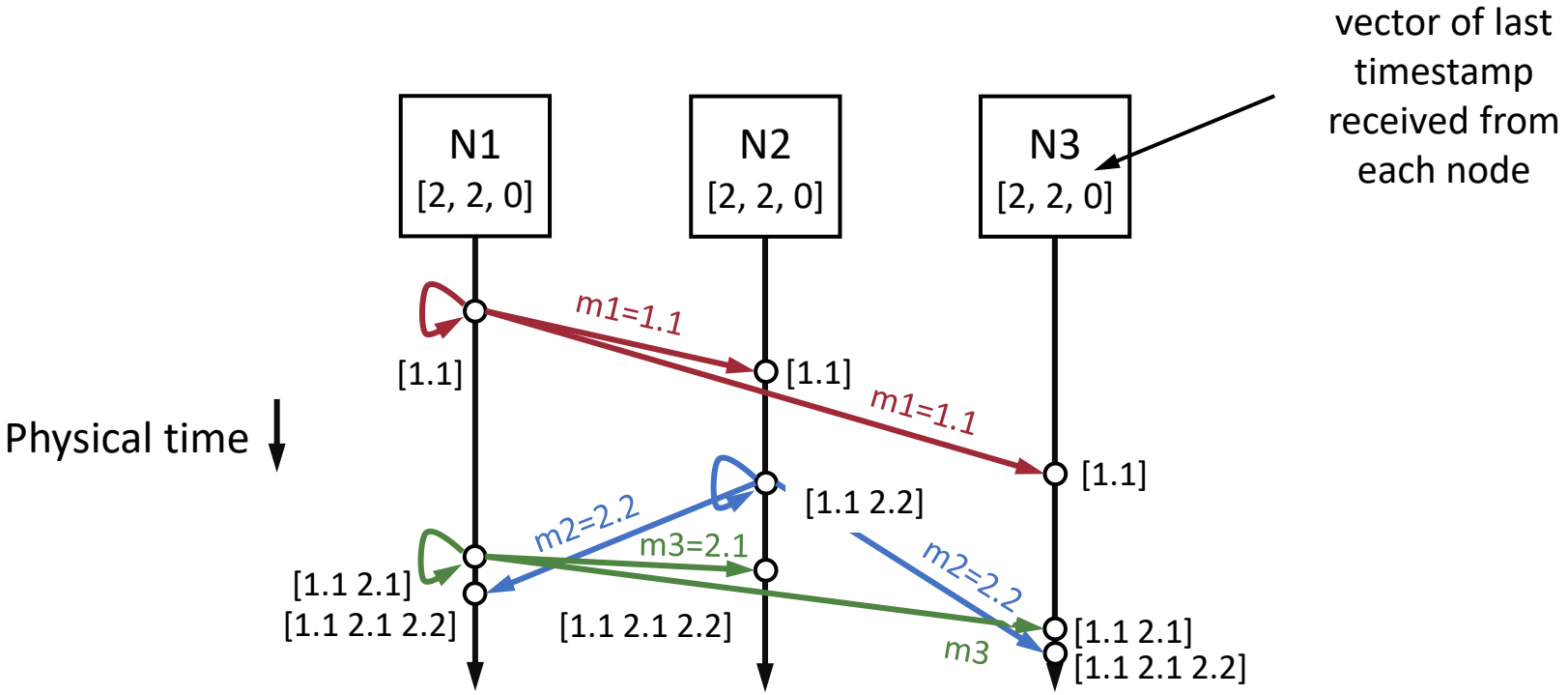
- When node broadcasts message:
  - Attach logical clock timestamp to the message
  - Send message via reliable broadcast
- When node receives message:
  - Buffer message in total order of timestamps
  - Suppose the earliest message in the buffer has timestamp T
    - When can we safely deliver it?
    - When we have seen all messages with timestamp < T

4B. Total order broadcast: logical clocks

- When node broadcasts message:
  - Attach logical clock timestamp to the message
  - Send message via reliable broadcast
- When node receives message:
  - Buffer message in total order of timestamps
  - Ensure that messages are received in FIFO order
  - Deliver earliest message in the buffer with timestamp T when messages with timestamp ≥ T are received from every node
  - Ensures FIFO-total order broadcast
- Assumption: nodes do not crash, why?

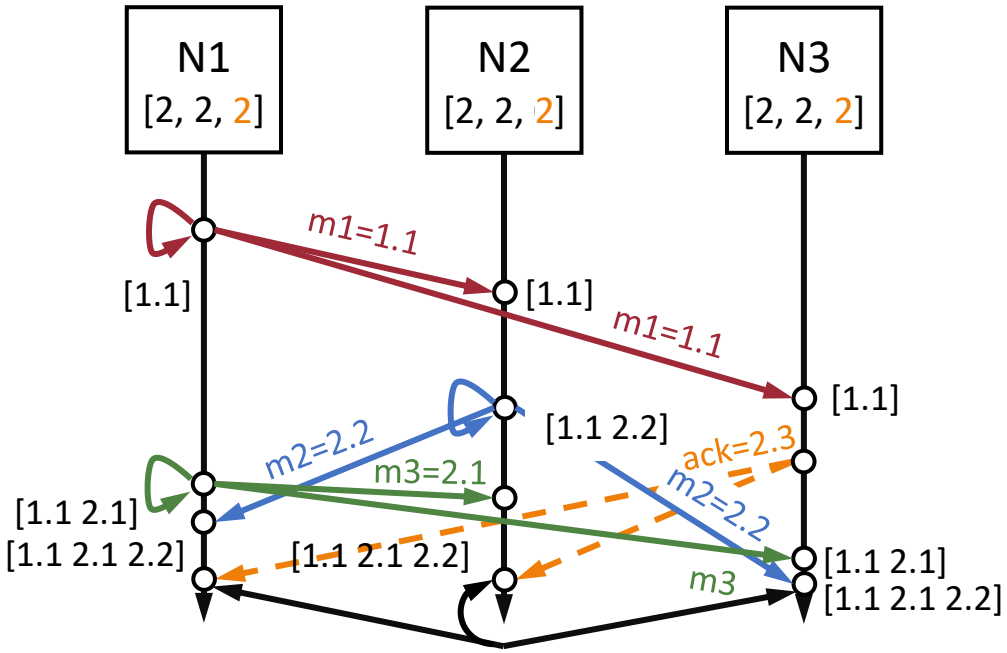
4B. Total order broadcast: example

- An arrow shows msg sent and received (not delivered)
- A […] buffer shows logical timestamps in sorted order
- Can m1=1.1 be delivered by any node now?



4B. Total order broadcast: example

- If a node doesn't send messages, others can't proceed
- Receiver should send ack messages when its last timestamp < received message timestamp



N1, N2 and N3 can deliver m1=1.1

Conclusions

- Broadcast (multicast) communication is used by a node to send messages to a group of nodes
  - It is useful for many distributed algorithms, e.g., replication
- Broadcast models include best-effort, reliable, FIFO, causal and total order
  - Other than best-effort, others offer reliable delivery
  - FIFO, causal and total order prescribe delivery order
  - With total order broadcast, when a node broadcasts a message, it needs to wait to deliver the message to itself
- In these slides, algorithms for all models, except total order broadcast, handle node failures
  - Later, we will look at fault-tolerant total order broadcast