Ordered Multicast

CSE 486/586: Distributed Systems

Ethan Blanton

Computer Science and Engineering
University at Buffalo

Causal and Total Orderings

We previously saw these definitions:

Causal ordering preserves the causal relationship between messages.

Formally:
If $\mathrm{MSend}(m, G) \rightarrow \mathrm{MSend}(m', G)$, then every correct process that delivers m' must have already delivered m.

Total ordering preserves the order of all messages across all processes.

Formally:
If any correct process delivers m before m', then every correct process that delivers m' must have already delivered m.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

The ISIS System

The ISIS system defined causally and totally ordered multicast [2][3].

It uses vector clocks for causal ordering.

Causal ordering is imposed on multicast message delivery only.

It uses a two-phase protocol for total ordering.

Processes cooperatively arrive at a total ordering for each message.

Safety and Liveness

These protocols maintain two properties [2]:

- Safety: The protocol never delivers messages in an order that violates the ordering constraints.
- Liveness: The protocol never delays a message indefinitely.

The latter requires that every message is delivered.

This can be accomplished via, *e.g.*, R_MCast.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

ISIS VT Protocol

ISIS defines several protocols for causal ordering.

The VT protocol [2] addresses causal ordering with static group membership.

More complicated ISIS protocols handle:

- Dynamic group membership (processes joining and leaving)
- Overlapping groups with causal relationships
- Causal total ordering

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

Vector Timestamps

The VT protocol uses vector timestamps.

Every message is transmitted with its timestamp.

These timestamps look just like our FIFO timestamps!

However, vector entries are causally updated like vector clocks.

Messages must be held back if they do not arrive in causal order.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

VT Protocol Vector Timestamps

The VT protocol maintains a vector VT for:

- Every message m: $\mathrm{VT}(m) = \langle p_1, ..., p_n \rangle$
- Every process p: $\mathrm{VT}(p) = \langle p_1, ..., p_n \rangle$

Each entry in the vector represents process p_i for $0 \leq i < n$ processes.

Every process maintains its own vector.

Every process increments only its own timestamp.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

VT Protocol Methods

$\mathrm{VT_Send}(m, G)$ at p_i:

- Increment $\mathrm{VT}(p_i)[i]$
- $\mathrm{VT}(m) = \mathrm{VT}(p_i)$
- $\mathrm{R_MCast}(\mathrm{VT}(m) \parallel m, G)$

$\mathrm{VT_Recv}(m)$ from p_j at $p_i \neq p_j$:

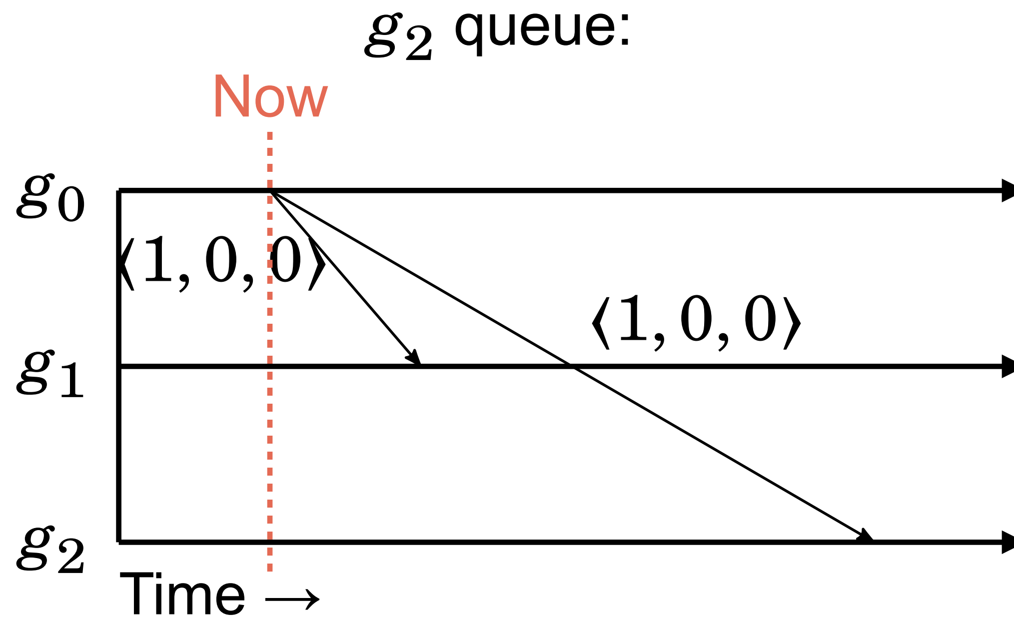
- If $\mathrm{VT}(m)[j] = \mathrm{VT}(p_i)[j] + 1$, and
  - $\forall k \neq j : \mathrm{VT}(m)[k] \leq \mathrm{VT}(p_i)[k]$
    - $\mathrm{VT_Deliver}(m)$
- Else:
  - Hold back m

$\mathrm{VT_Deliver}(m)$ from p_j at $p_i \neq p_j$

- Increment $\mathrm{VT}(p_i)[j]$
- $\mathrm{Deliver}(m)$
- Run hold back queue

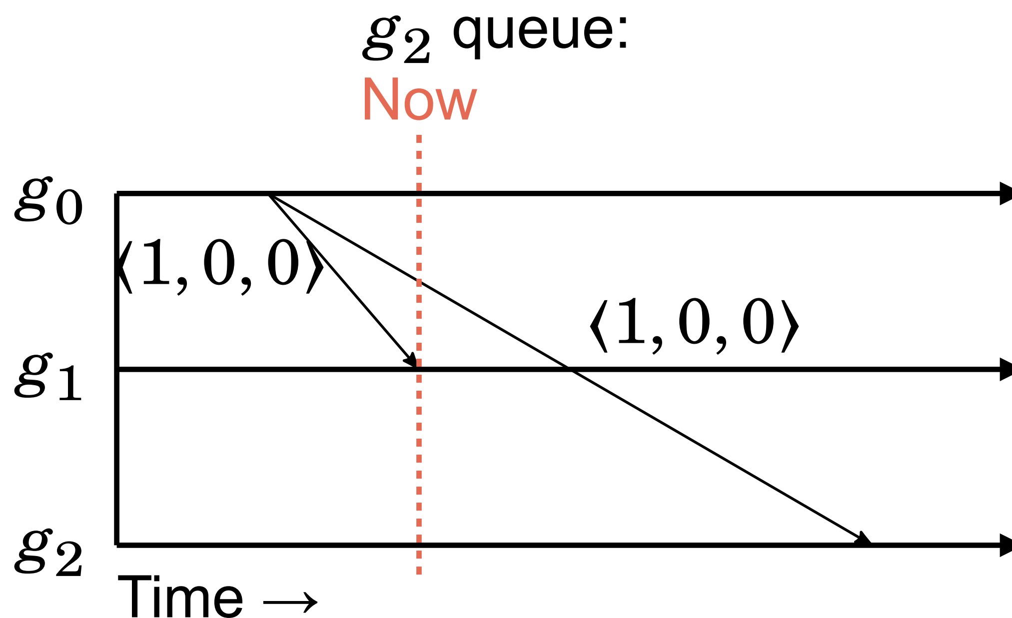
© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

VT Example



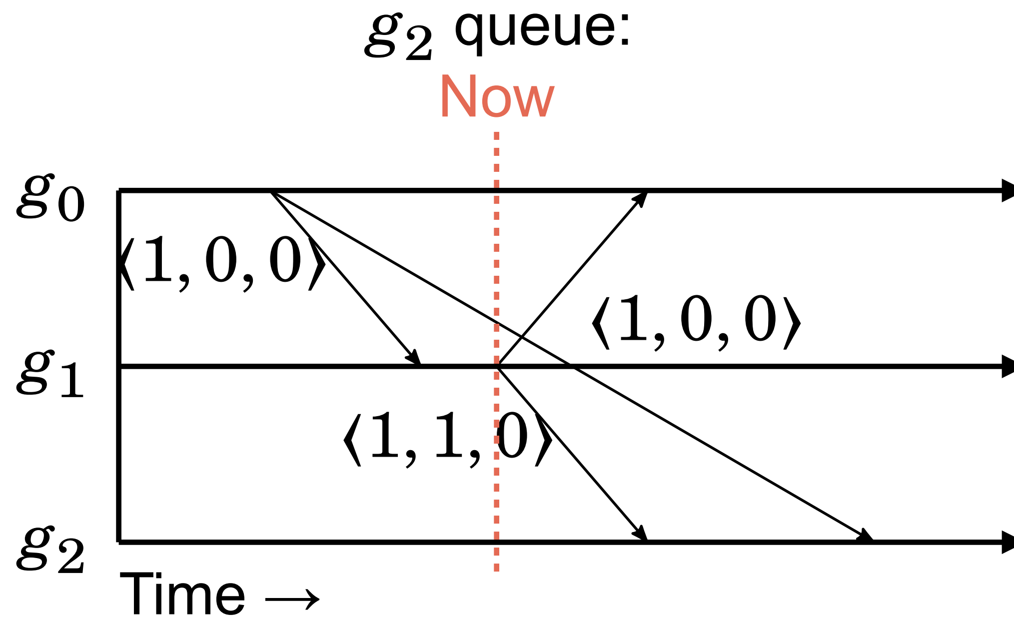
g_0 sends a message with timestamp $\langle 1, 0, 0 \rangle$.

VT Example



g_1 receives and delivers message $\langle 1, 0, 0 \rangle$.

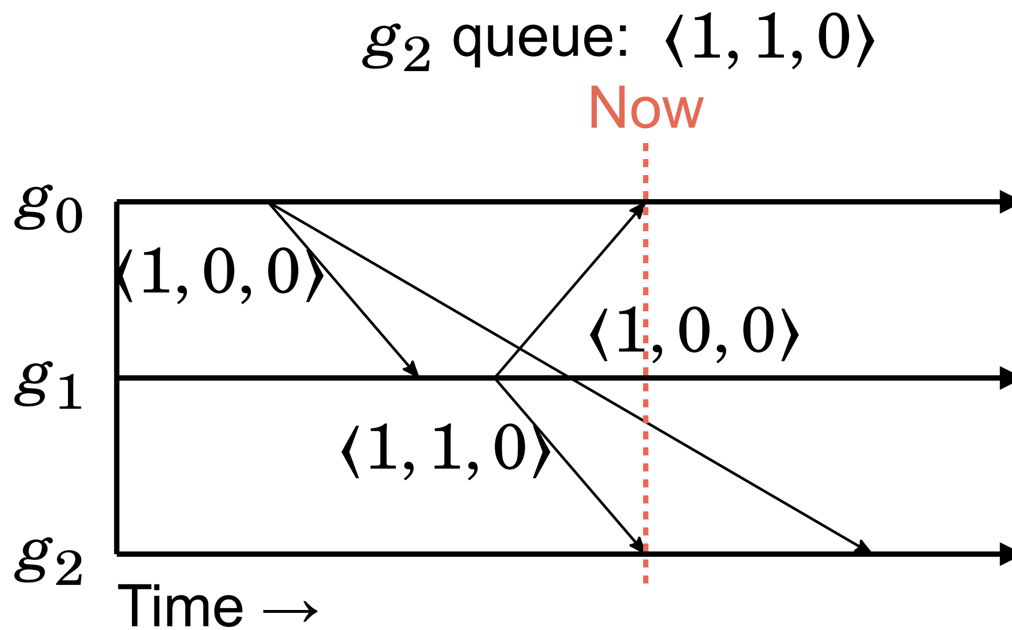
VT Example



g_1 sends a message $\langle 1, 1, 0 \rangle$

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

VT Example

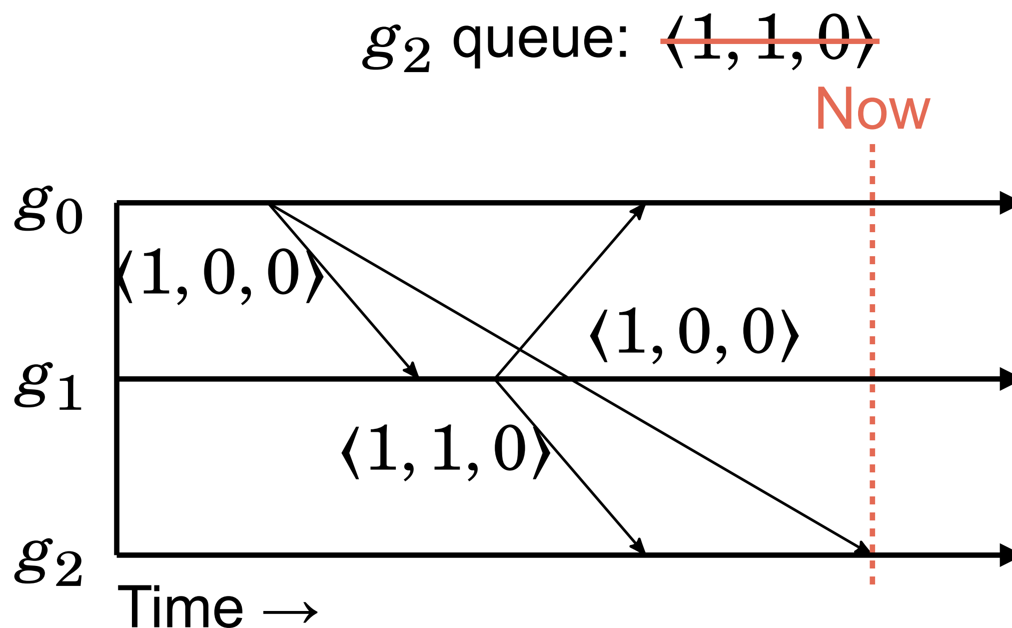


g_0 receives and delivers $\langle 1, 1, 0 \rangle$.

g_2 holds back $\langle 1, 1, 0 \rangle$.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

VT Example



g_2 receives and delivers $\langle 1, 0, 0 \rangle$.

g_2 delivers held-back message $\langle 1, 1, 0 \rangle$.

Total Ordering with a Sequencer

Total ordering can be achieved through a sequencer.

Each time a process p_i wants to send a message:

1. p_i sends m to the sequencer
2. The sequencer sends m with FIFO multicast

All messages are received FIFO from the sequencer.

What are the disadvantages of this?

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

ISIS ABCAST Protocol

The ISIS ABCAST Protocol [3] is totally ordered.

It doesn't require a central sequencer!

It uses a two phase protocol.

Each message is:

- Transmitted without ordering
- Ordered and delivered

Messages are queued but undeliverable until ordered.

The ordering of each message is managed by its sender.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

Intuition

Every host p_i in ABCAST maintains a logical clock T_i.

Every message has two associated timestamps:

- A proposed timestamp T_m^p, set when it is transmitted
- An ordered timestamp T_m^o, set when it is deliverable

The ordered timestamp of a message is the maximum clock on all processes when its proposal was received.

The clock ticks for:

- Sending an unordered message
- Receiving an unordered message

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

ABCAST Phase 1

In the first phase of message transmission:

1. Process p_i increments its local clock T_i.
2. Process p_i adds m to its queue as undeliverable at priority $T_m^p = T_i$.
3. Process p_i multicasts m with timestamp T_m^p.

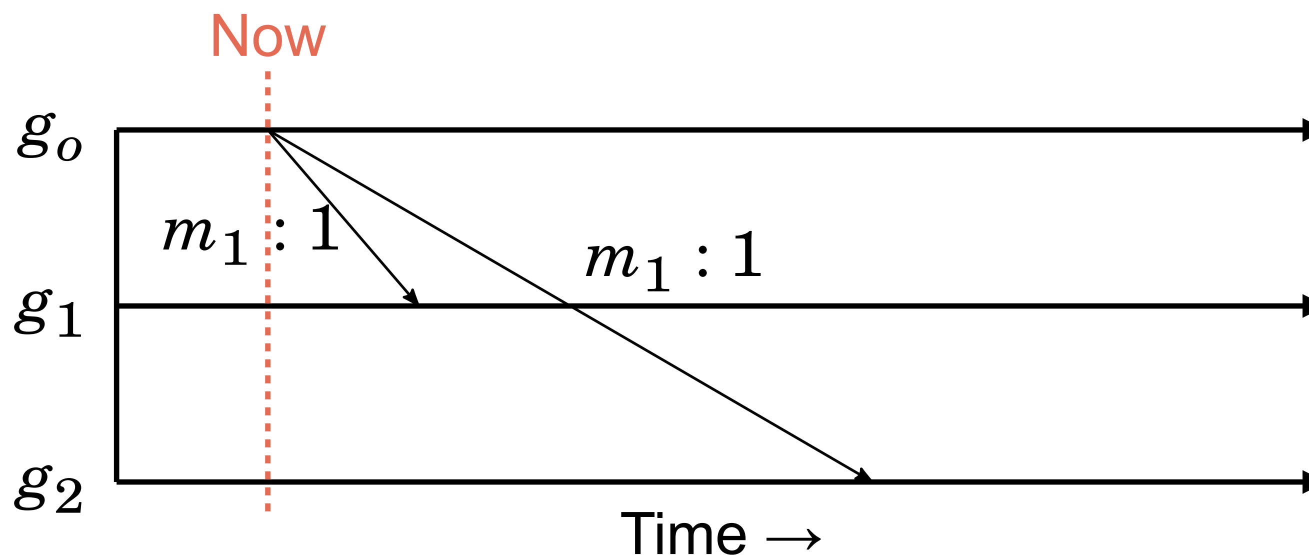
Every process $p_j, j \neq i$ eventually receives m and:

1. p_j increments T_j.
2. p_j sets its local timestamp to $\mathrm{MAX}\left(T_j, T_m^p\right)$.
3. p_j adds m to its queue as undeliverable at priority T_j.
4. p_j sends an acknowledgment for m with timestamp T_j to p_i.

p_i collects all acknowledgments for m.

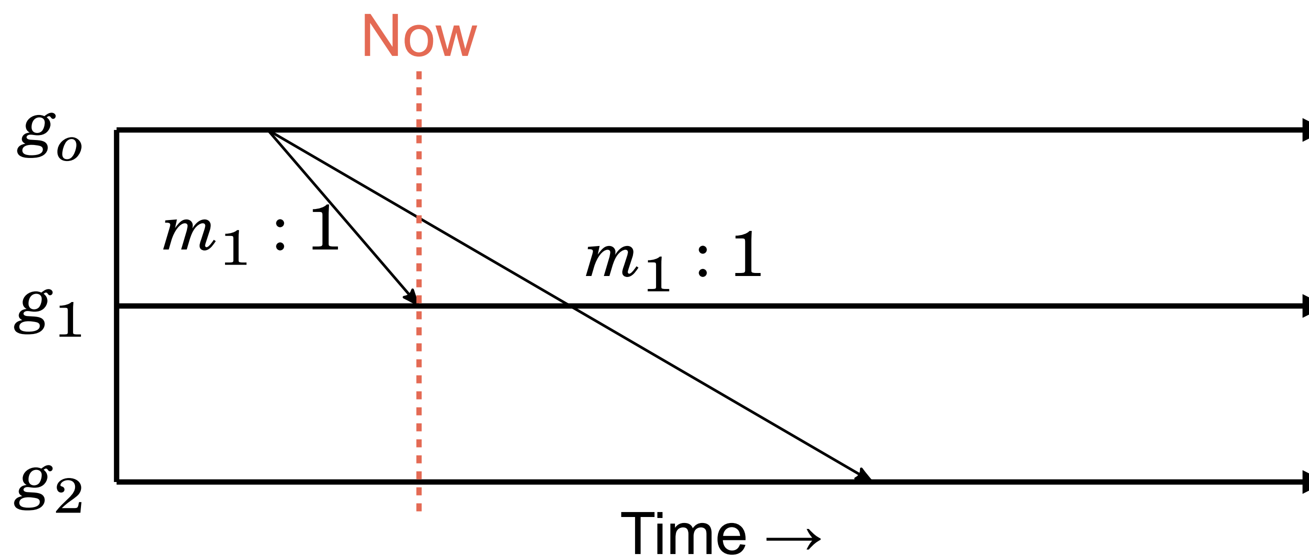
© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

Phase 1 Example



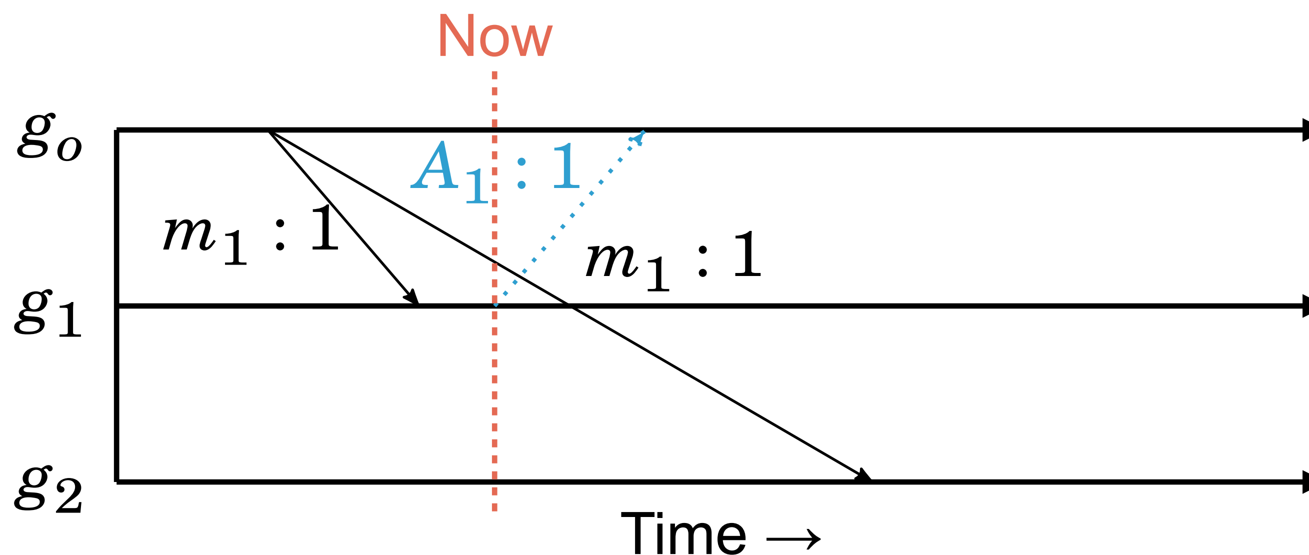
g_1 sends message m_1 with timestamp 1.

Phase 1 Example



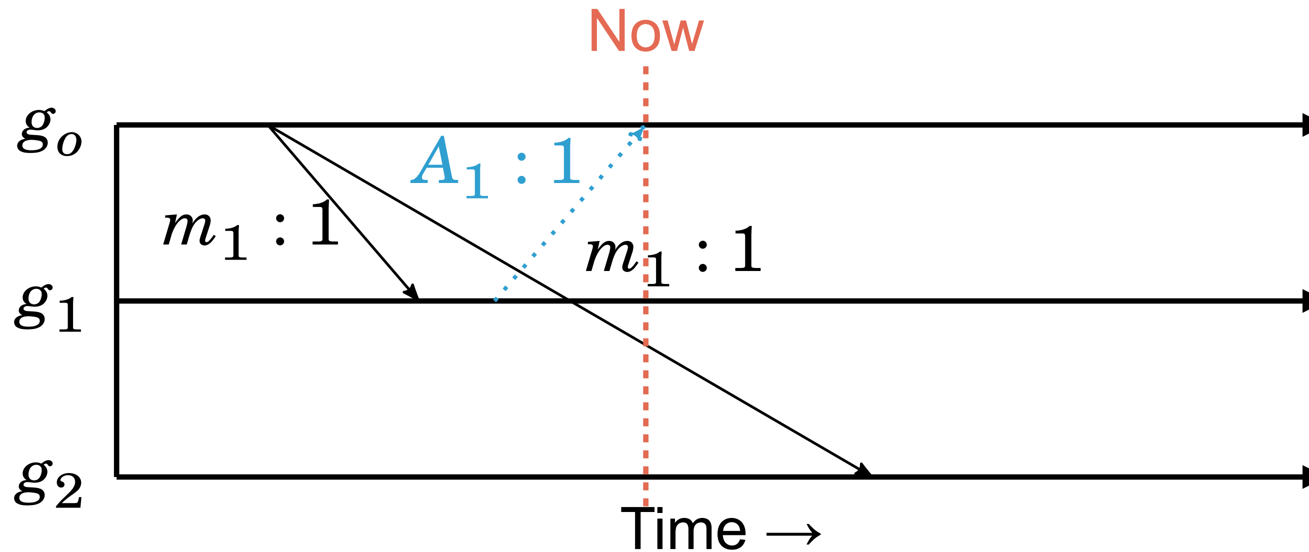
g_1 receives m_1.

Phase 1 Example



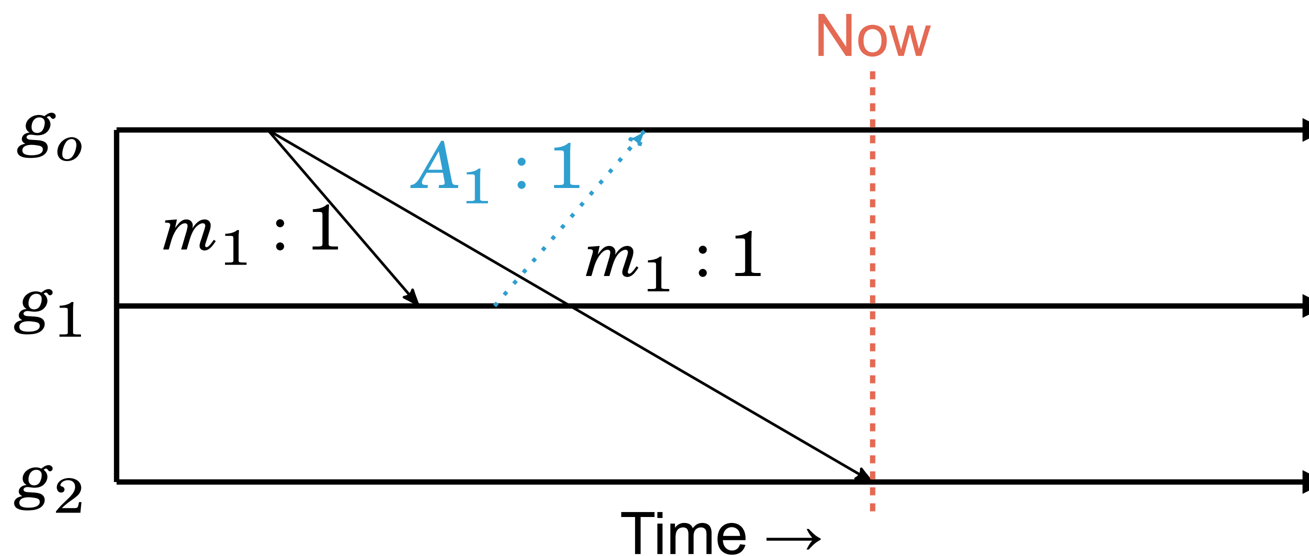
g_1 returns and acknowledgment for m_1 with timestamp 1.

Phase 1 Example



g_0 receives g_1's acknowledgment.

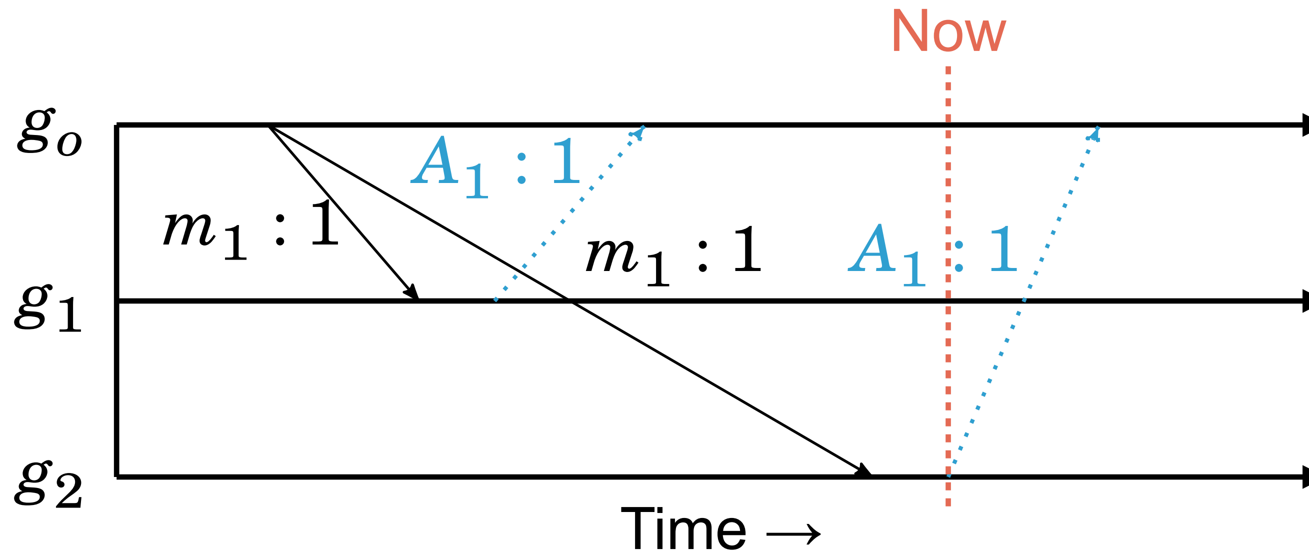
Phase 1 Example



g_2 receives m_1.

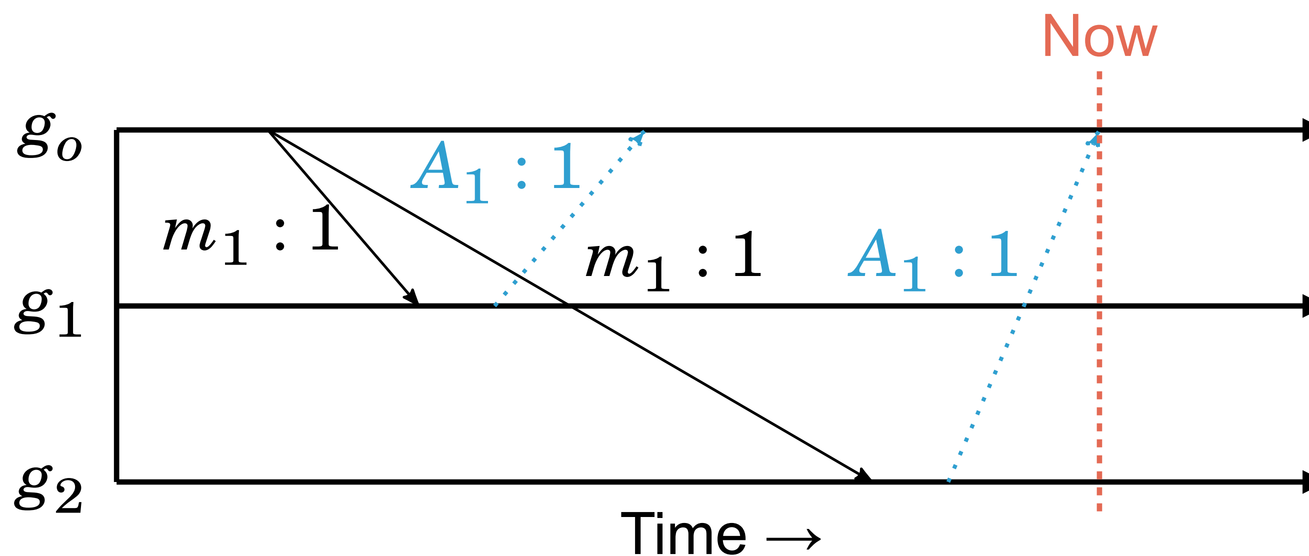
© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

Phase 1 Example



g_2 returns an acknowledgment for m_1 with timestamp 1.

Phase 1 Example



g_0 receives g_2's acknowledgment.

ABCAST Phase 2

In the second phase of message m transmission from p_i:

p_i performs the following steps:

1. compute the maximum timestamp T_m^o from all acknowledgments of m
2. multicast the ordered message m with timestamp T_m^p

Each process $p_j, j \neq i$ eventually receives the ordered m and:

1. marks m as deliverable
2. delivers all deliverable messages at the front of its queue

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

Tie Breaking

There's one wrinkle:
What if two processes propose the same max timestamp for different messages?

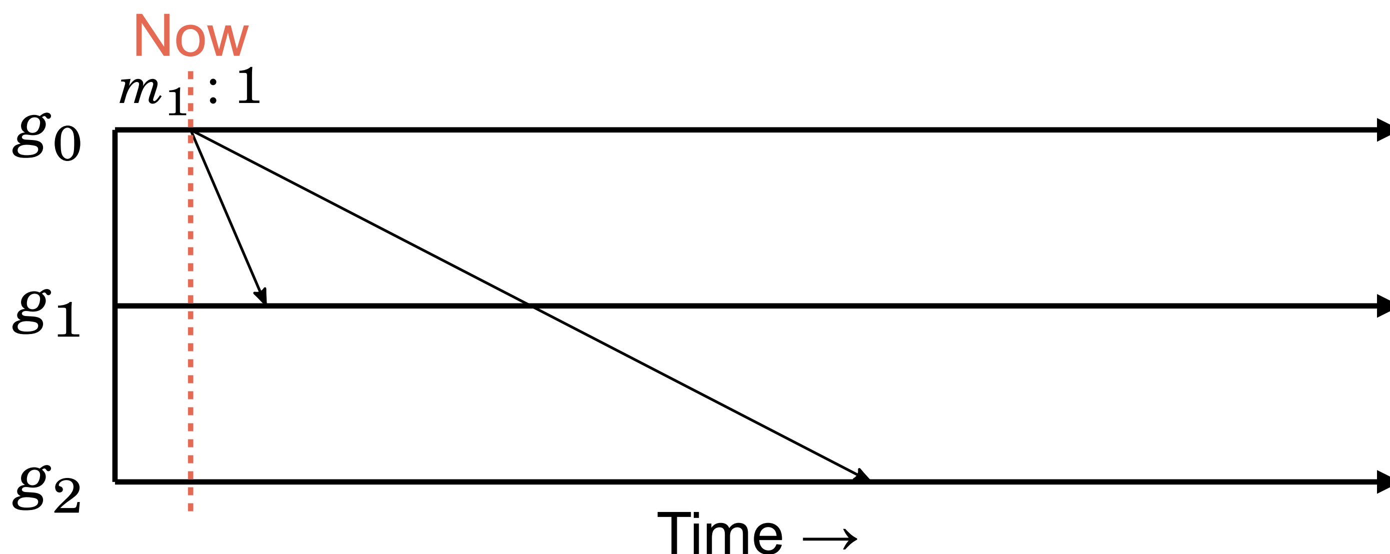
Those messages are tie broken by appending the process ID.

If timestamp $T_i = k$, it is treated as $k.i$; for example:
Timestamp 3 at g_2 is 3.2.

We will elide this suffix when it is irrelevant.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

ABCAST Example

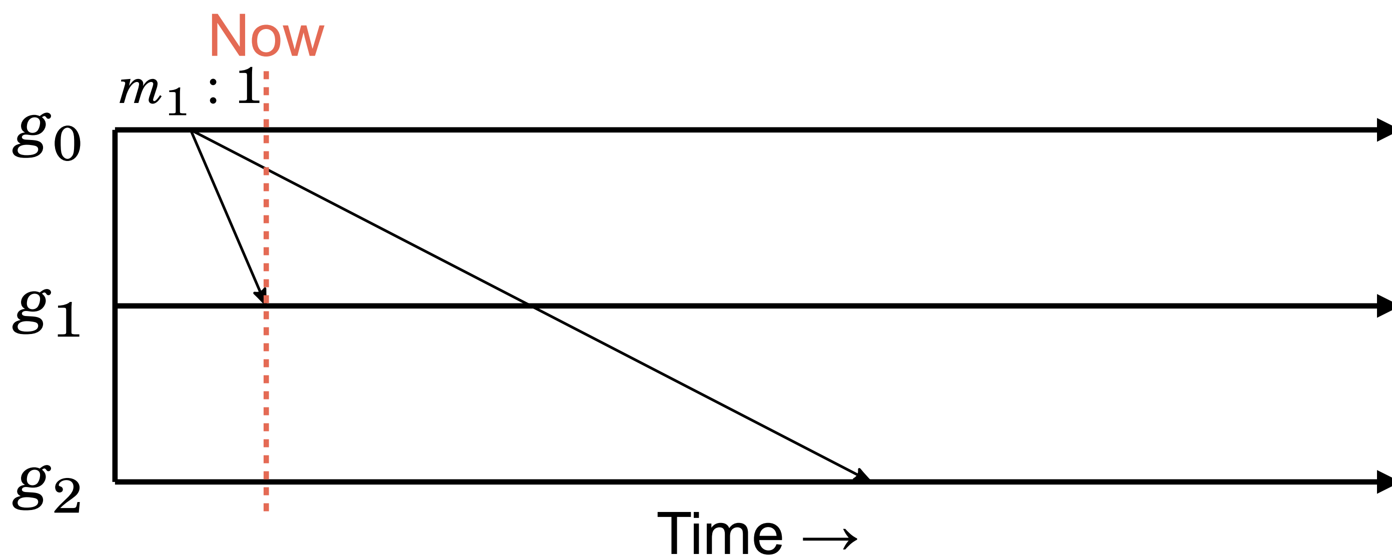


Queues:

$m_1 : 1$

g_0 multicasts m_1 with proposed priority 1.

ABCAST Example



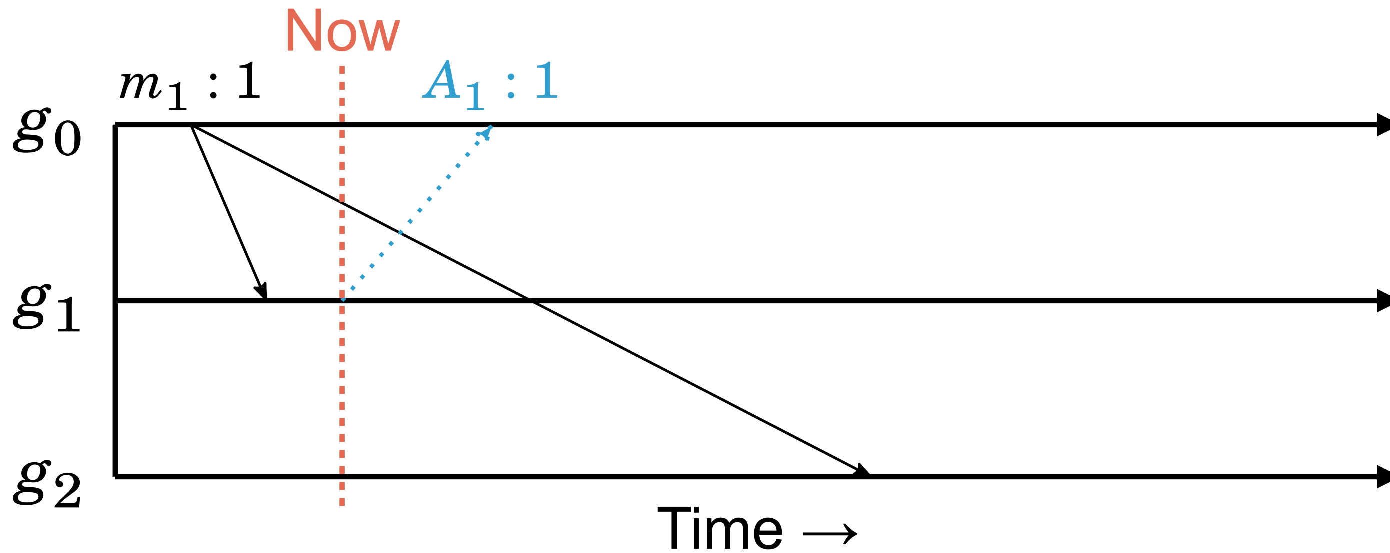
Queues:

$m_1 : 1$

$m_1 : 1$

g_1 receives m_1 and enqueues it at priority 1.

ABCAST Example



Queues:

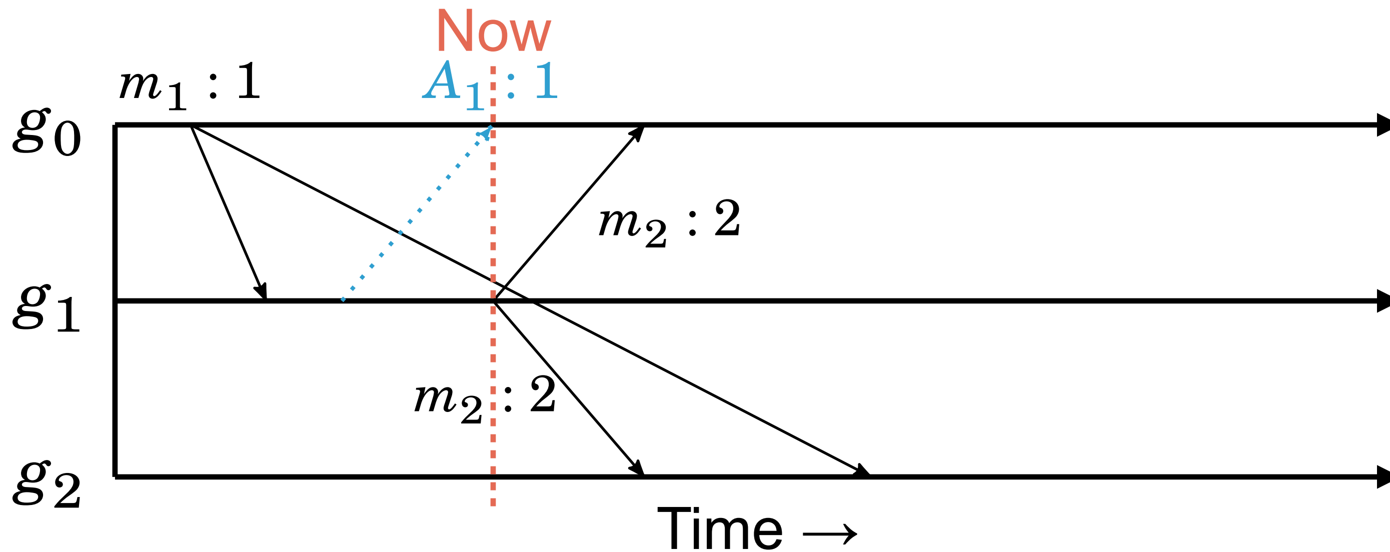
$m_1 : 1$

$m_1 : 1$

g_1 sends an acknowledgment for m_1 at priority 1.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

ABCAST Example



Queues:

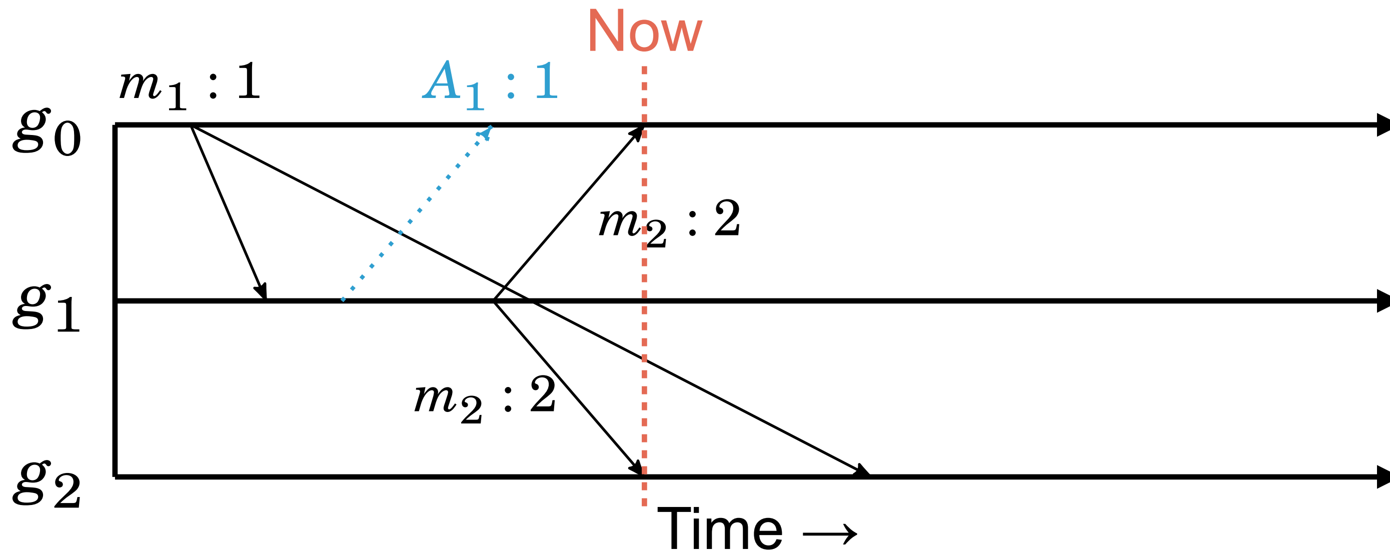
$m_1 : 1$

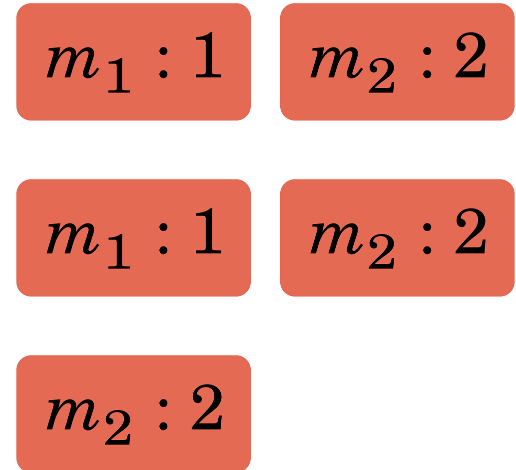
$m_1 : 1$ $m_2 : 2$

g_0 receives the acknowledgment from g_1 and records it.

g_1 multicasts m_2 with proposed priority 2.

ABCAST Example

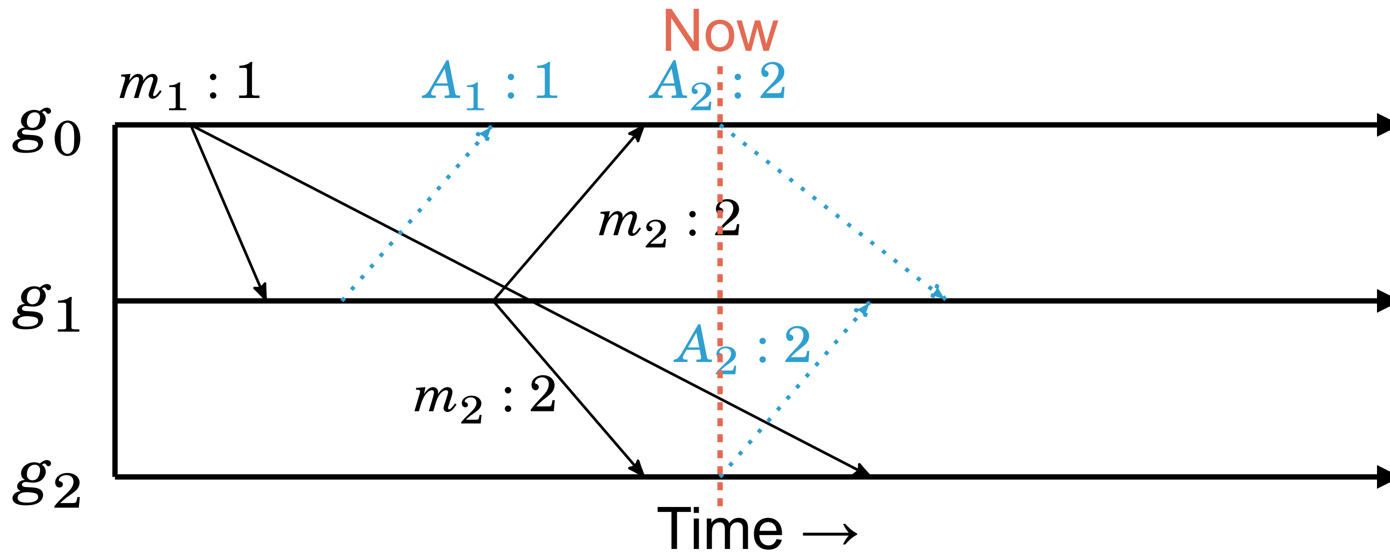


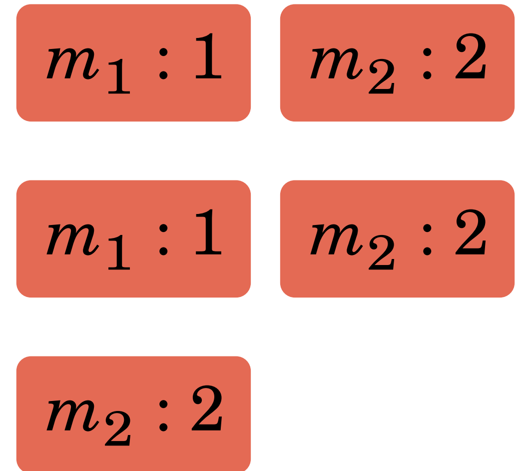


g_0 and g_2 receive and enqueue m_2 at priority 2.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

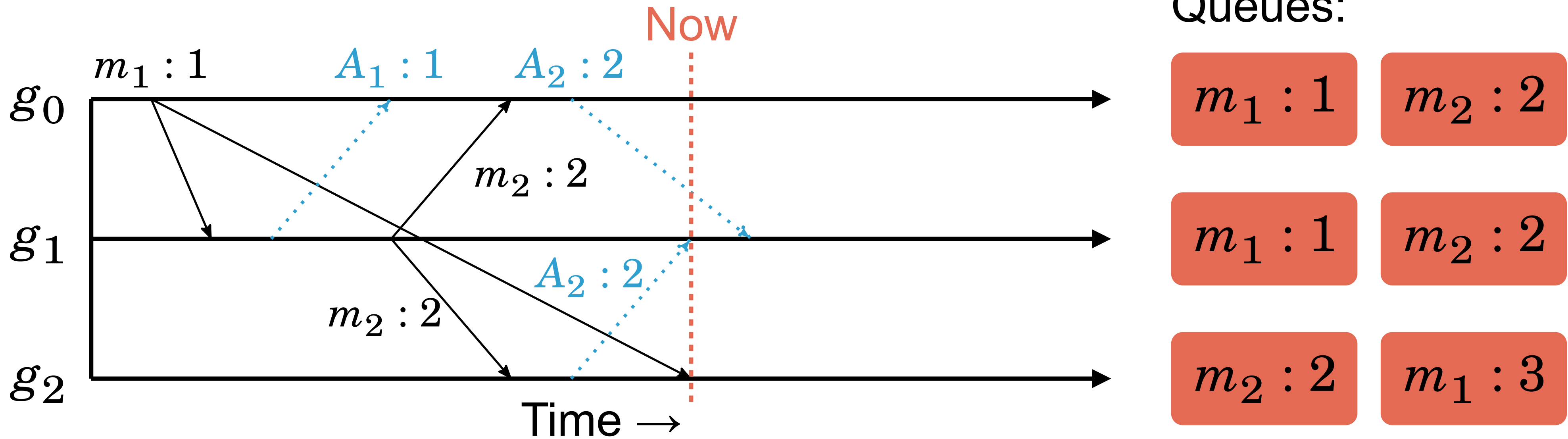
ABCAST Example





g_0 and g_2 send their respective acknowledgments for m_2 at priority 2.

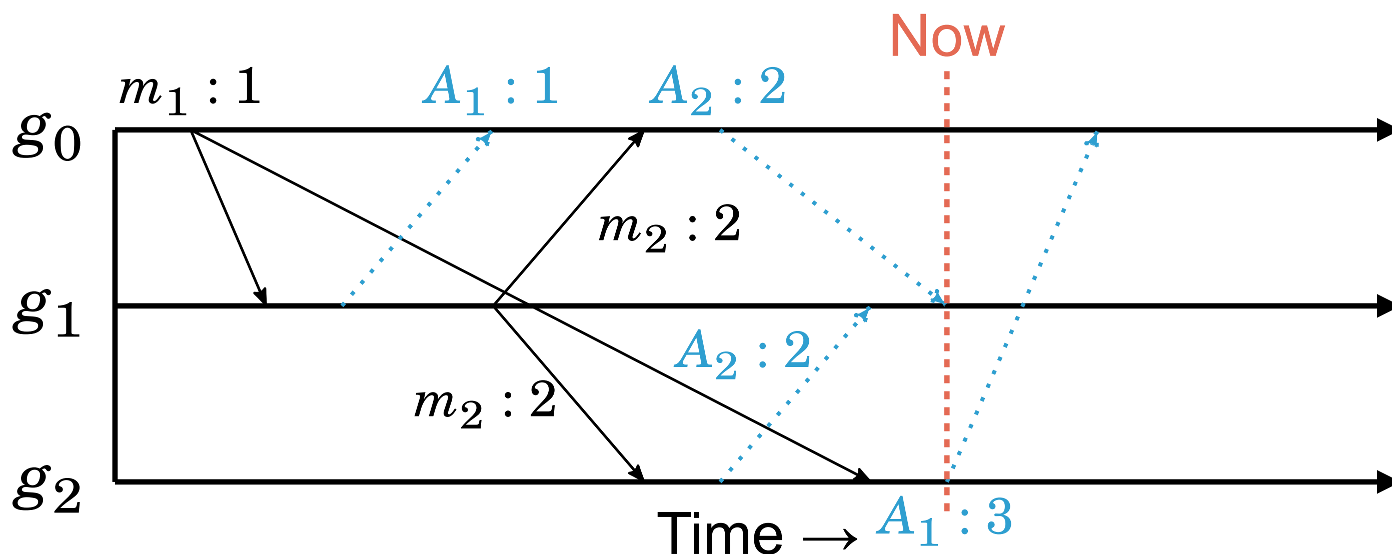
ABCAST Example

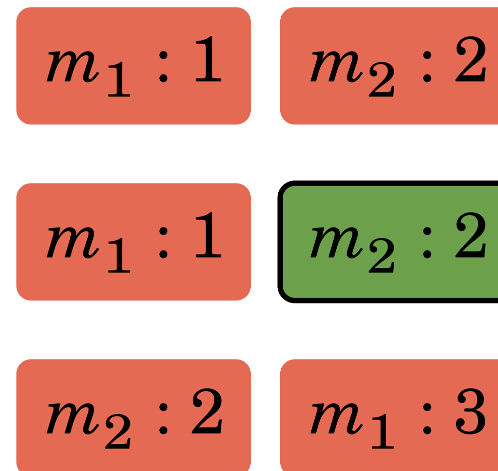


g_1 receives the acknowledgment from g_2 and records it.

g_2 receives m_1 and enqueues it for priority 3, as m_2 is at priority 2.

ABCAST Example

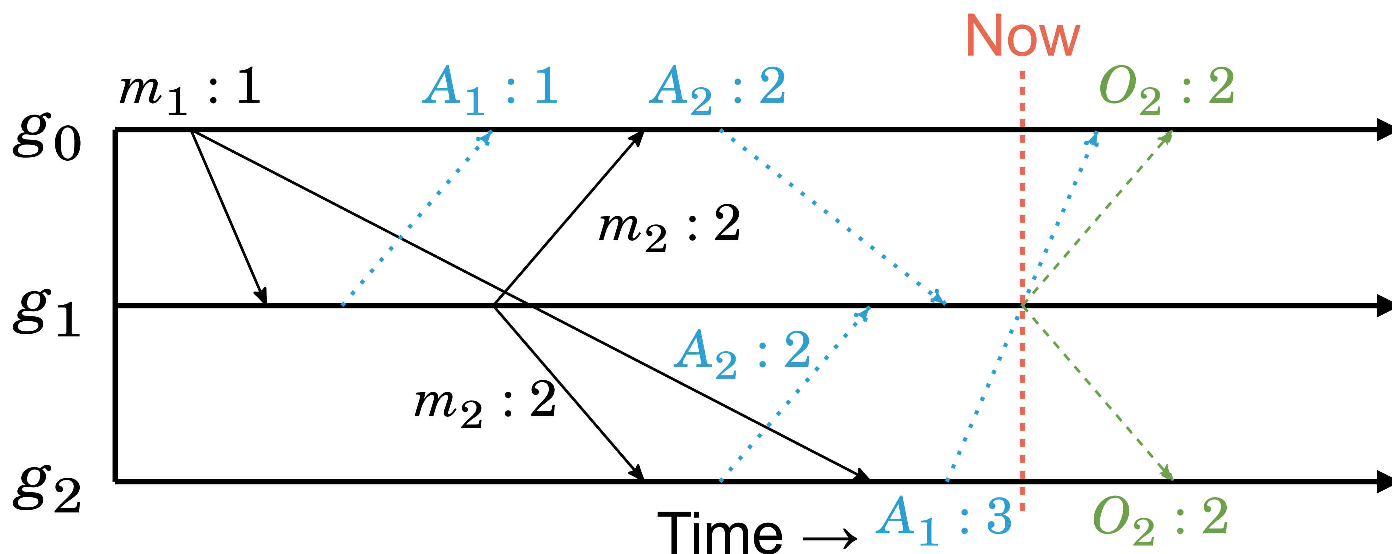


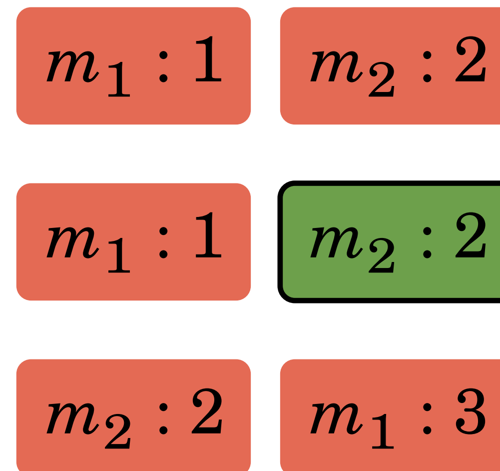


g_1 receives g_0's acknowledgment for m_2 and orders it at priority 2.

g_2 sends an acknowledgment for m_1 at priority 3.

ABCAST Example

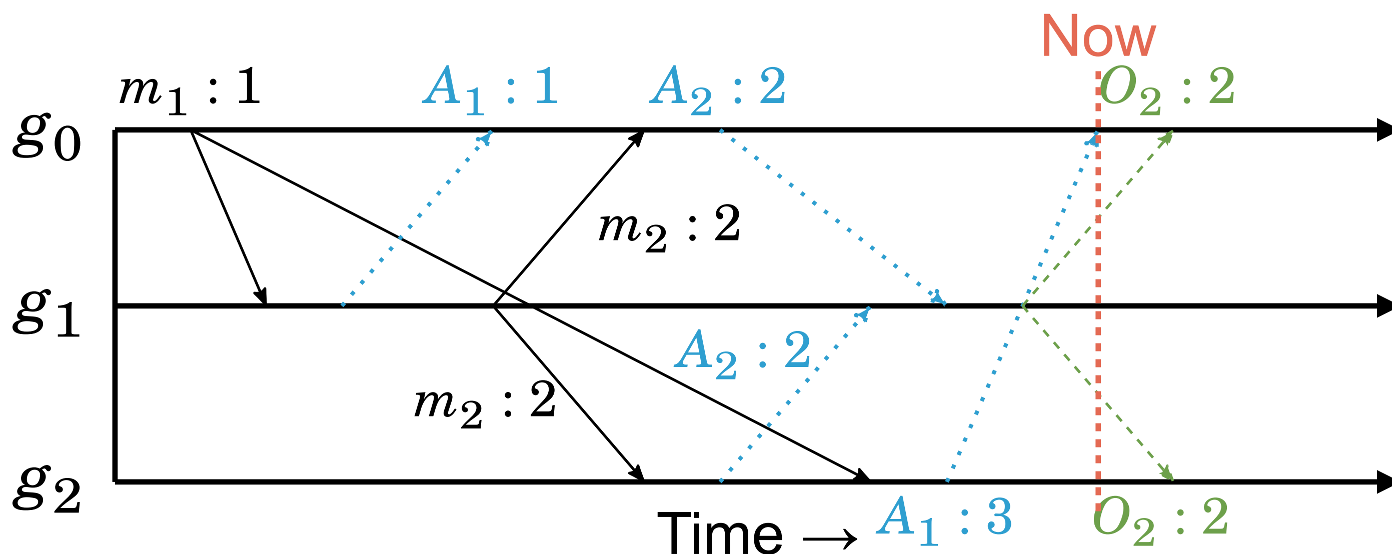


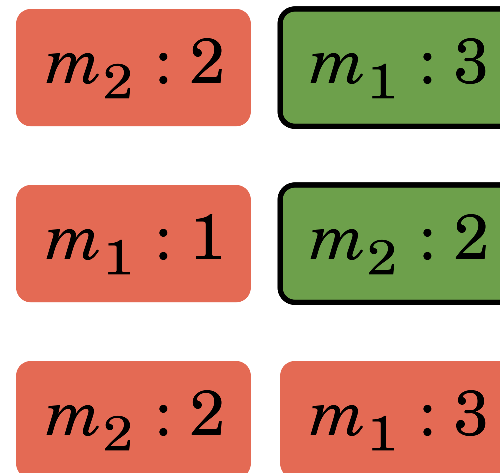


g_1 sends out an ordering for m_2 at priority 2.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

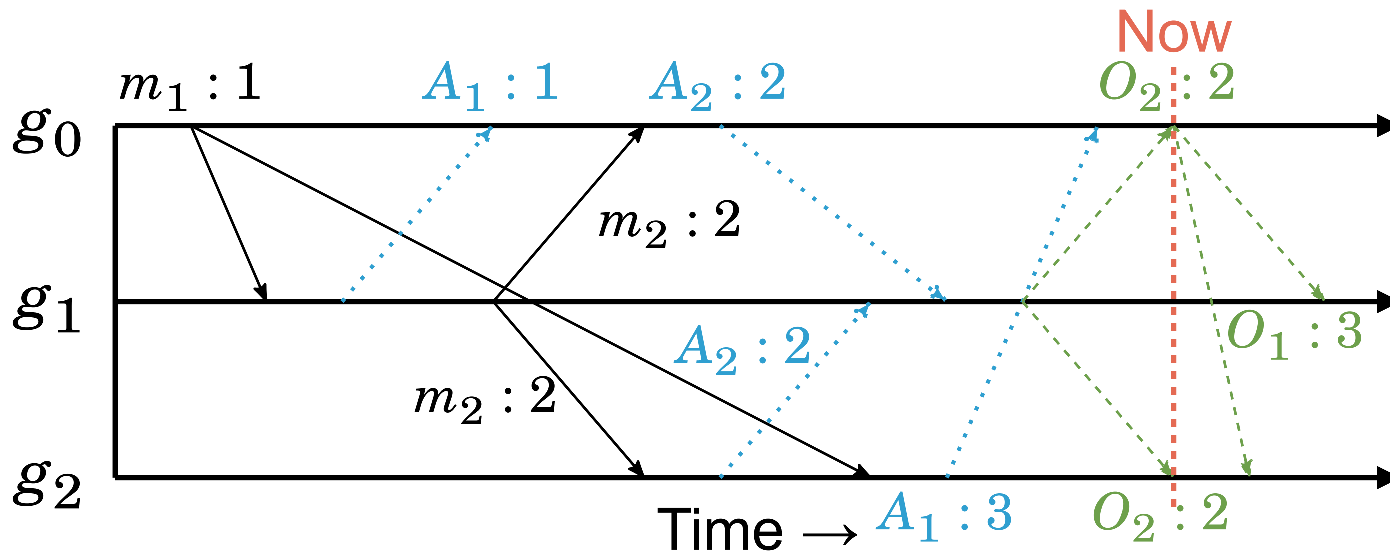
ABCAST Example

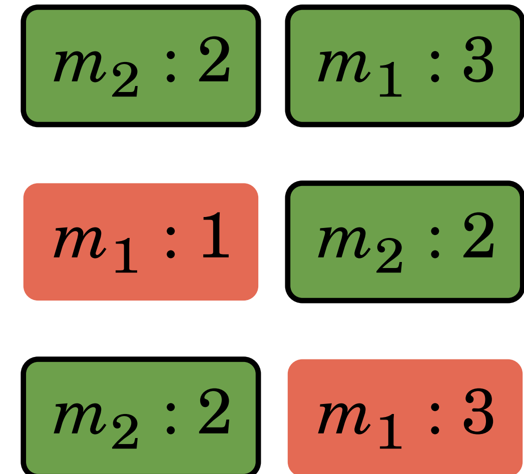




g_0 receives g_2's acknowledgment for m_1, and orders m_1 at priority 3.

ABCAST Example



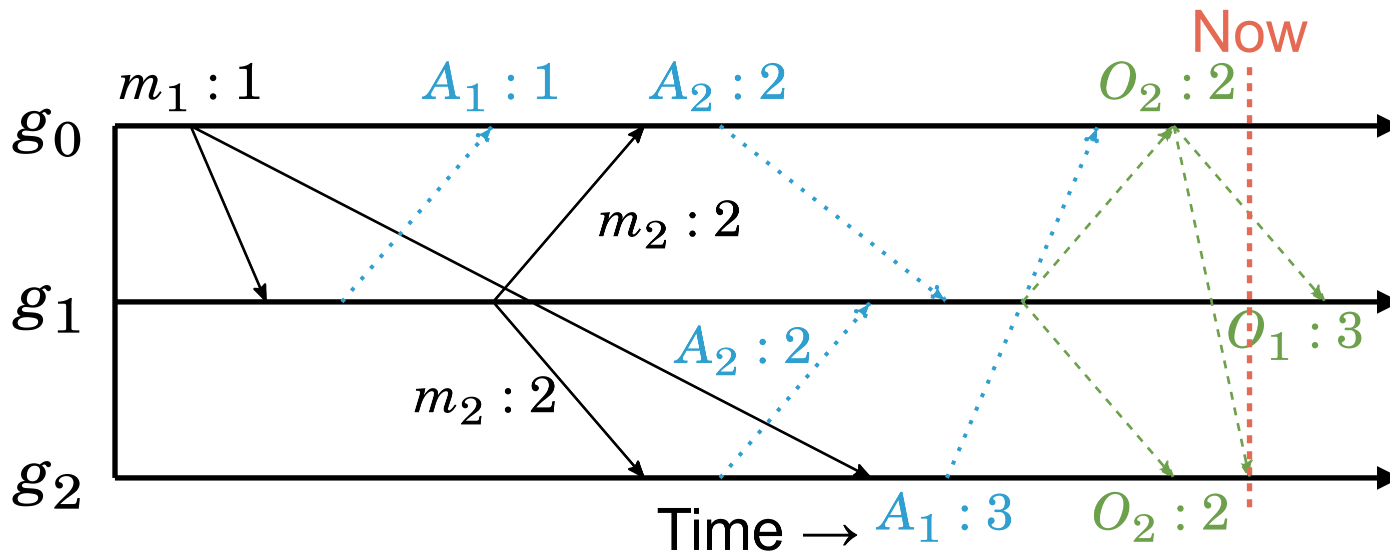


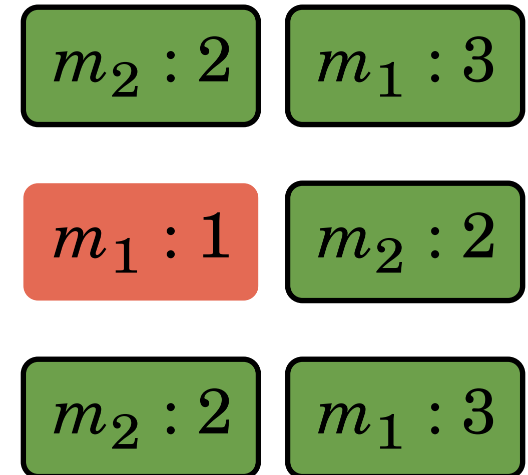
g_0 receives the ordering for m_2 and delivers m_2 followed by m_1.

g_0 sends out an ordering for m_1 at priority 3.

g_2 receives the ordering for m_2 and delivers it.

ABCAST Example

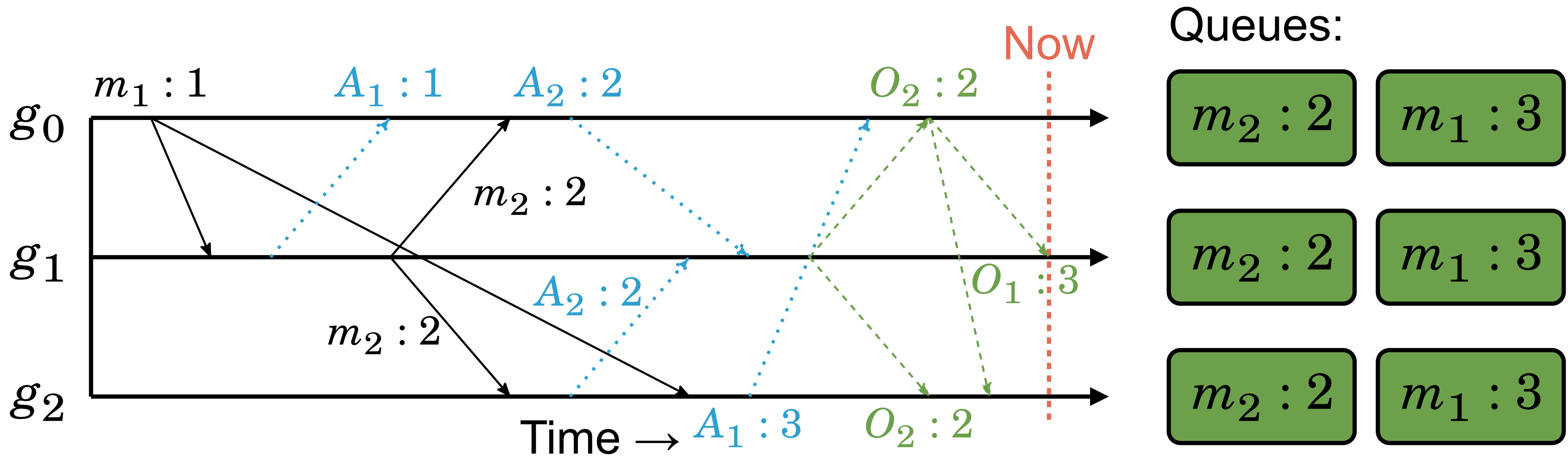




g_2 receives the ordering for m_1 and delivers it.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

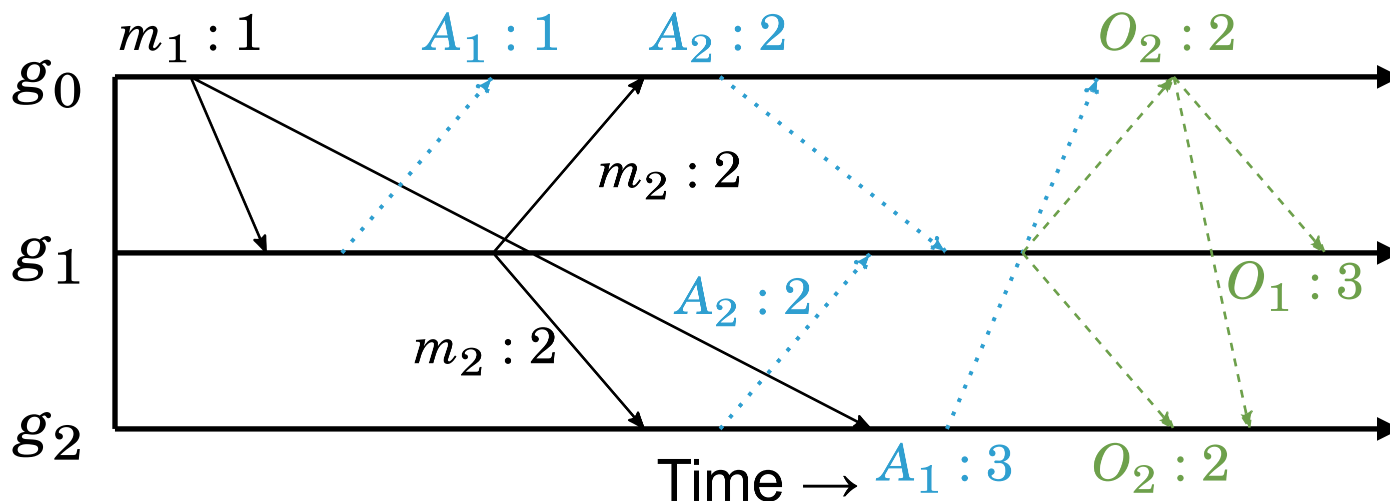
ABCAST Example

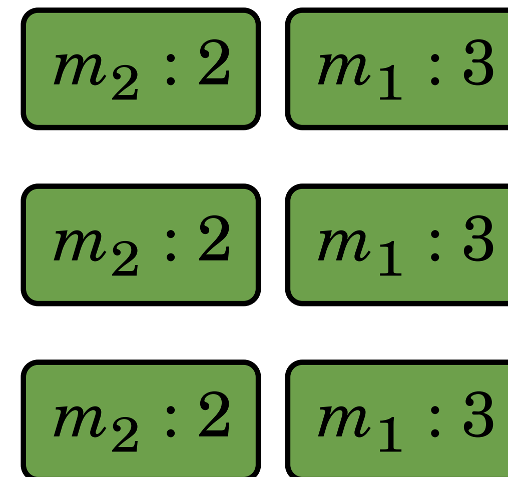


g_1 receives the ordering for m_1, re-enqueues it, then delivers m_2 followed by m_1.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

ABCAST Example





All processes delivered m_2 followed by m_1.

Sketch for Correctness

Why does this work?

Every process knows that m will be delivered no earlier than its acknowledged ordering.

The sequencer for m takes the maximum observed timestamp.

When a deliverable message is in the queue the local process will never propose an earlier sequence!

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

Summary

- Safety means constraints will never be violated
- Liveness means every message is eventually delivered
- ISIS provides causally and totally ordered multicast
- The VT protocol uses vector clocks to causally order
- ISIS ABCAST uses distributed sequencing to totally order

Bibliography

Required Readings

[1] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Chapter 6: 6.6. Cambridge University Press, 2008.

Optional Readings

[2] Kenneth Birman, Andre Schiper, and Pat Stephenson. *Fast Causal Multicast*. technical report Contractor Report 19900012217. National Aeronautics and Space Administration, April 1990.

[3] Kenneth P. Birman and Thomas A. Joseph. “Reliable Communication in the Presence of Failures”. In: *ACM Transactions on Computer Systems* 5.1 (February 1987), pages 47–76.

© 2026 Ethan Blanton / CSE 486/586: Distributed Systems

Copyright

Copyright 2021, 2023–2026 Ethan Blanton, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see https://cse.buffalo.edu/~eblanton/.