Lamport Logical Clocks, Totally Ordered Multicast, Vector Clocks

12 January 2025
Lecture 10

Slide Credits: Maarten van Steen

Topics for Today

- Logical Clocks
  - Lamport
  - Totally Ordered Multicast
- (Mattern) Vector Clocks

Source: TvS 6.2-6.4

The Happened-Before Relationship

Problem: We first need to introduce a notion of order in before we can order anything.

The **happened-before** relation on the set of events in a distributed system:

- If a and b are two events in the same process, and *a* comes before b, then $\mathrm{a} \rightarrow b$.
- If *a* is the sending of a message, and b is the receipt of that message, then $a \rightarrow b$
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Note: this introduces a **partial ordering of events** in a system with concurrently operating processes.

Logical Clocks (1/2)

Problem: How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?

Solution: attach a timestamp $C(e)$ to each event e, satisfying the following properties:

P1: If a and b are two events in the same process, and $a \to b$, then we demand that $C(a) < C(b)$.

P2: If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Problem: How to attach a timestamp to an event when there's no global clock → maintain a **consistent** set of logical clocks, one per process

Logical Clocks (2/2)

Solution

Each process P_i maintains a **local** counter C_i and adjusts this counter according to the following rules:

1: For any two successive events that take place within P_i, C_i is incremented by 1.

2: Each time a message m is sent by process P_i, the message receives a timestamp $ts(m) = C_i$.

3: Whenever a message m is received by a process P_j, P_j adjusts its local counter C_j to $\max\{C_j, ts(m)\}$; then executes step 1 before passing m to the application.

Property **P1** is satisfied by (1);

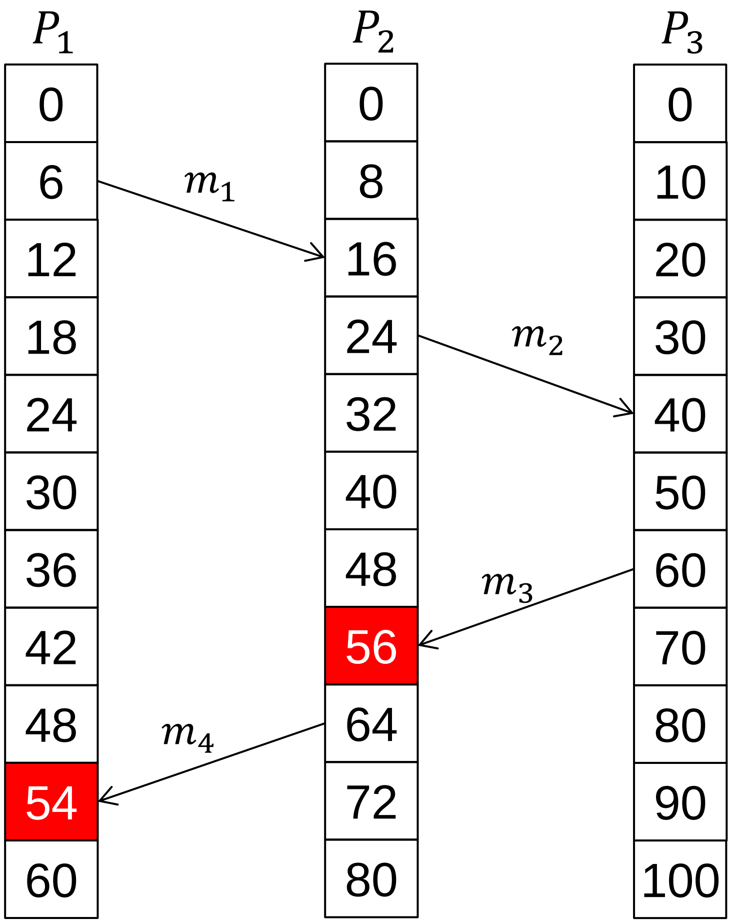
Property **P2** by (2) and (3).

Note

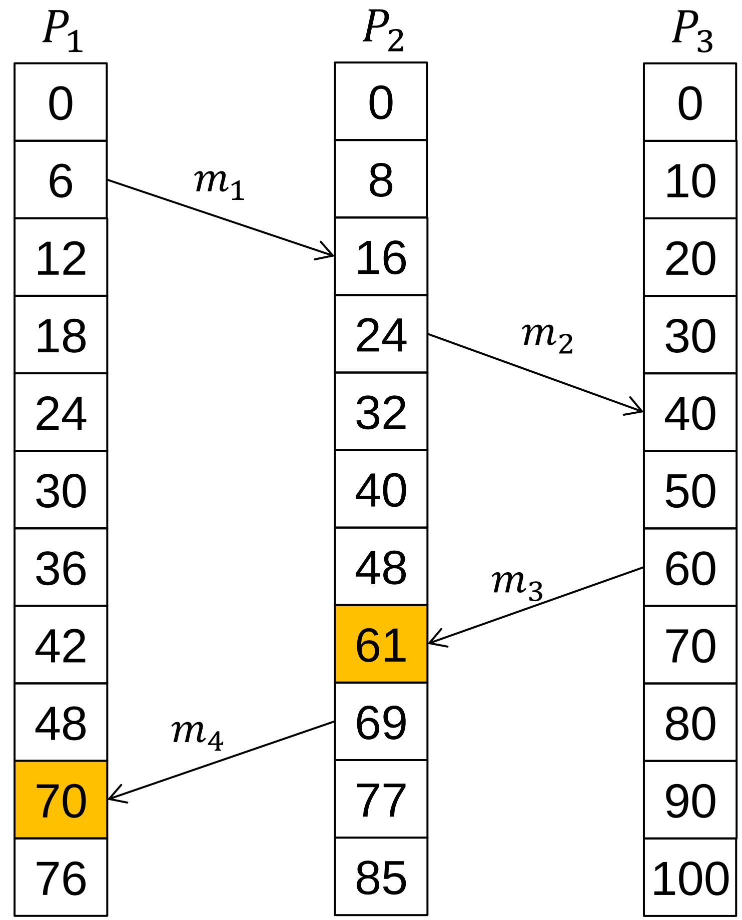
It is still possible for two events to happen at the same time. Avoid this by breaking ties through process IDs.

Logical Clocks - Example

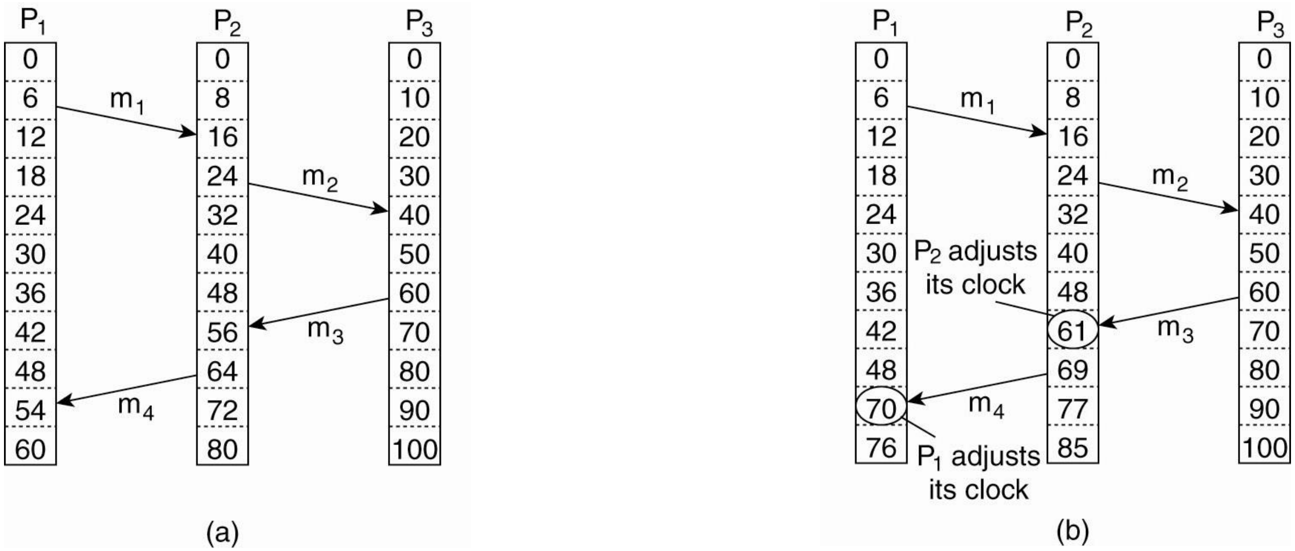
No Clock Adjustment



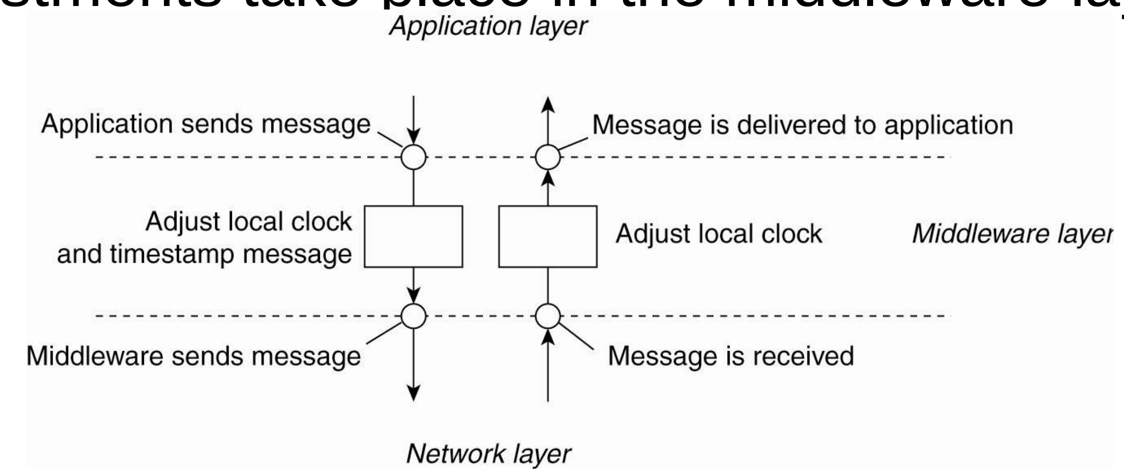
Clock Adjustment



Logical Clocks - Example



Note: Adjustments take place in the middleware layer:



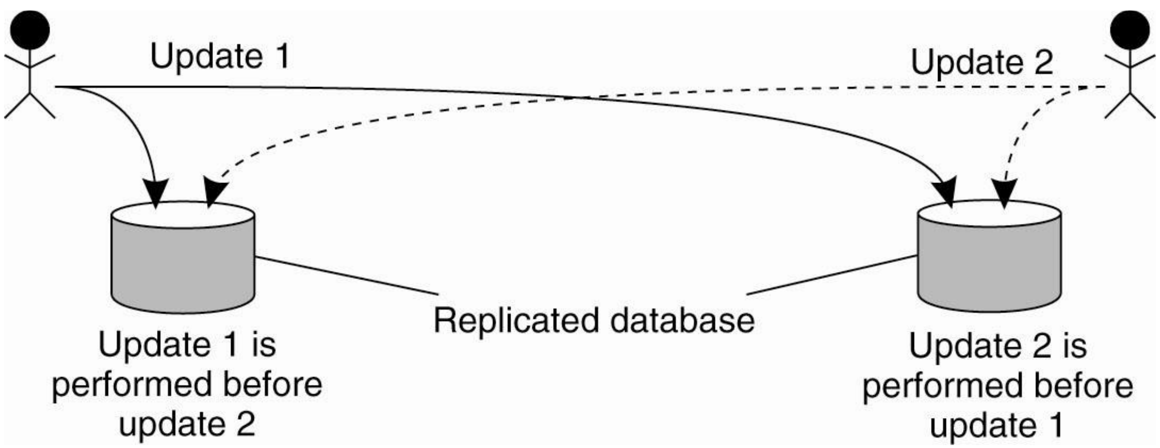
So Far

- Logical Clocks
  - Lamport
  - Totally Ordered Multicast
- (Mattern) Vector Clocks

Example: Totally Ordered Multicast (1/2)

Problem: We sometimes need to guarantee that concurrent updates on a replicated database are seen in the same order everywhere:

- P_1 adds \$100 to an account (initial value: \$1000)
- P_2 increments account by 1%
- There are two replicas



Result: in absence of proper synchronization:
replica #1 ← \$1111, while replica #2 ← \$1110.

Example: Totally Ordered Multicast (2/2)

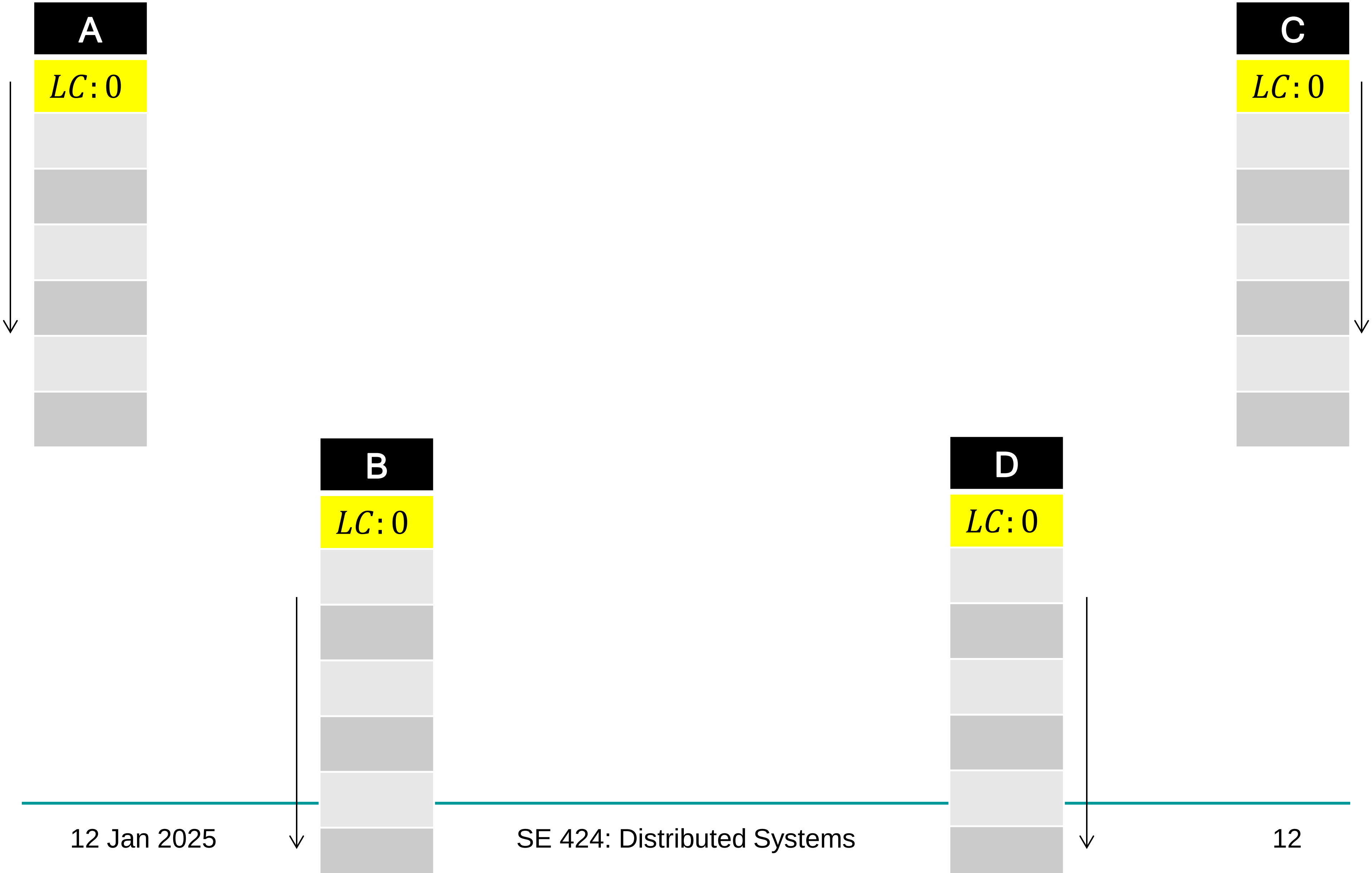
Solution:

- Process P_i sends timestamped message msg_i to all others. The message itself is put in a local queue $queue_i$.
- Any incoming message at P_j is queued in $queue_j$, according to its timestamp, and acknowledged to every other process.

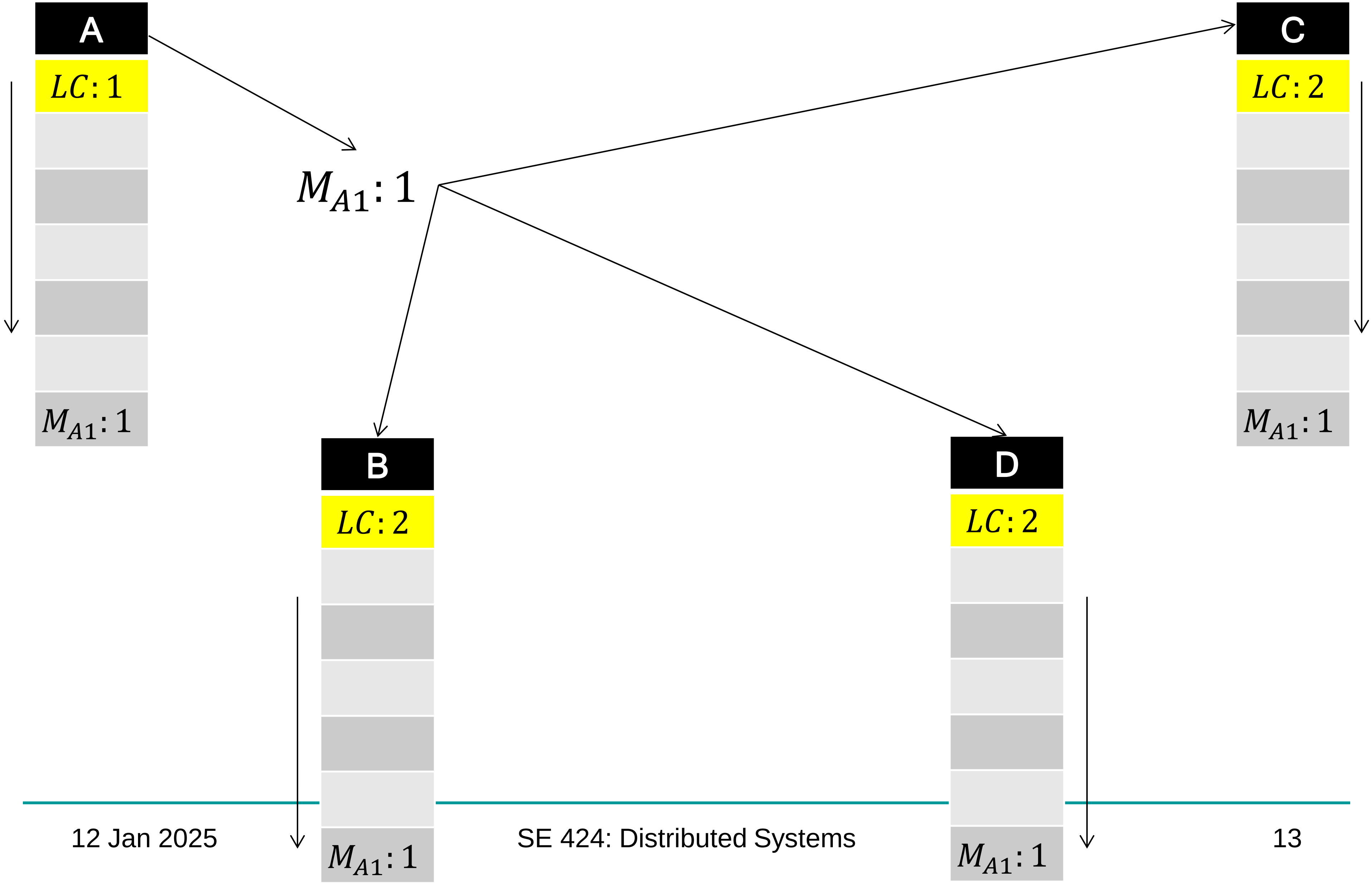
> P_j passes a message msg_i to its application if:
>
> (1) msg_i is at the head of $queue_j$
>
> (2) for each process P_k, there is a message msg_k in $queue_j$ with a larger timestamp.

Note: We are assuming that communication is reliable and FIFO ordered (i.e. messages from a single sender arrive in the order sent)

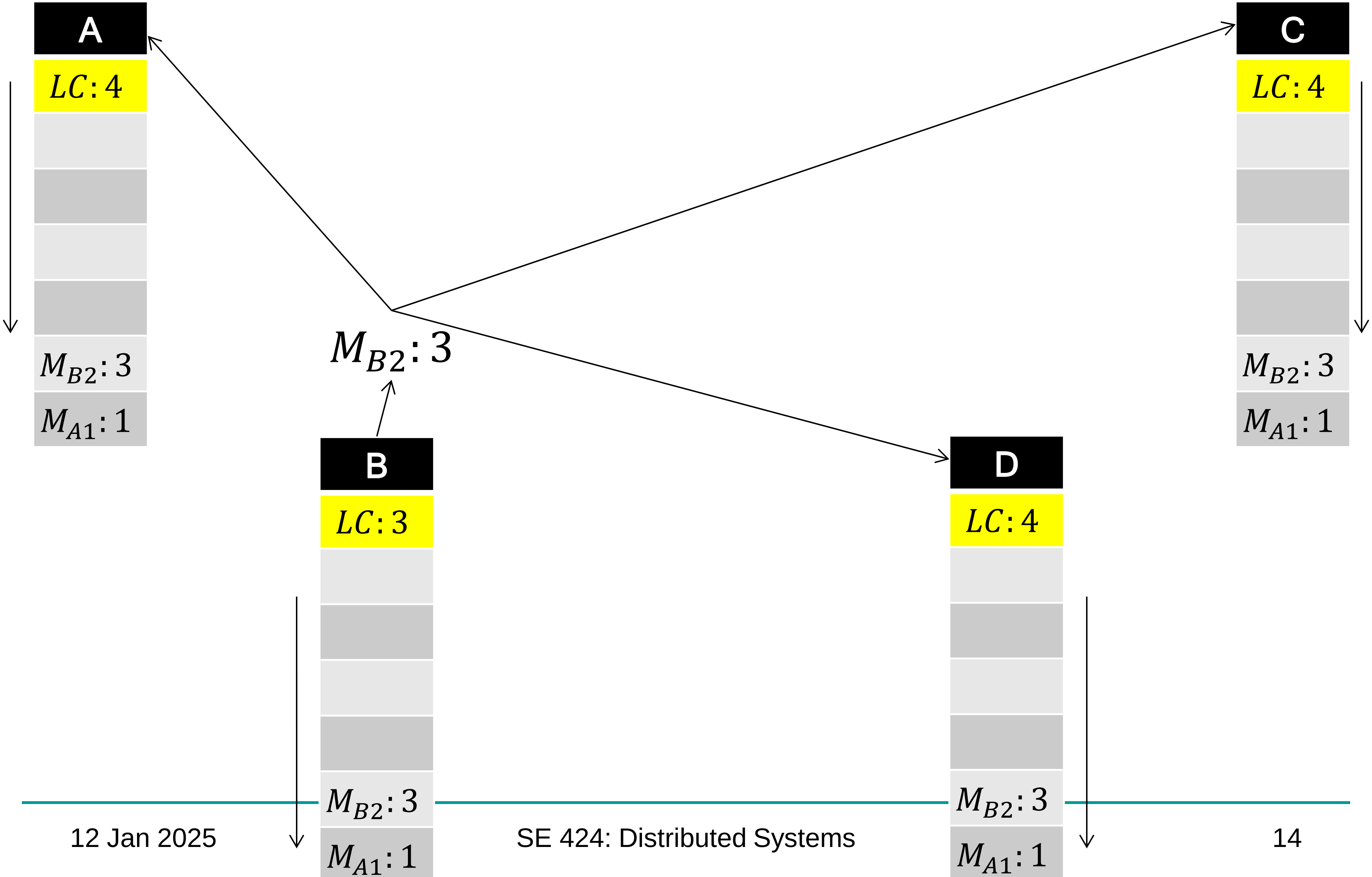
TOM Illustrated 1



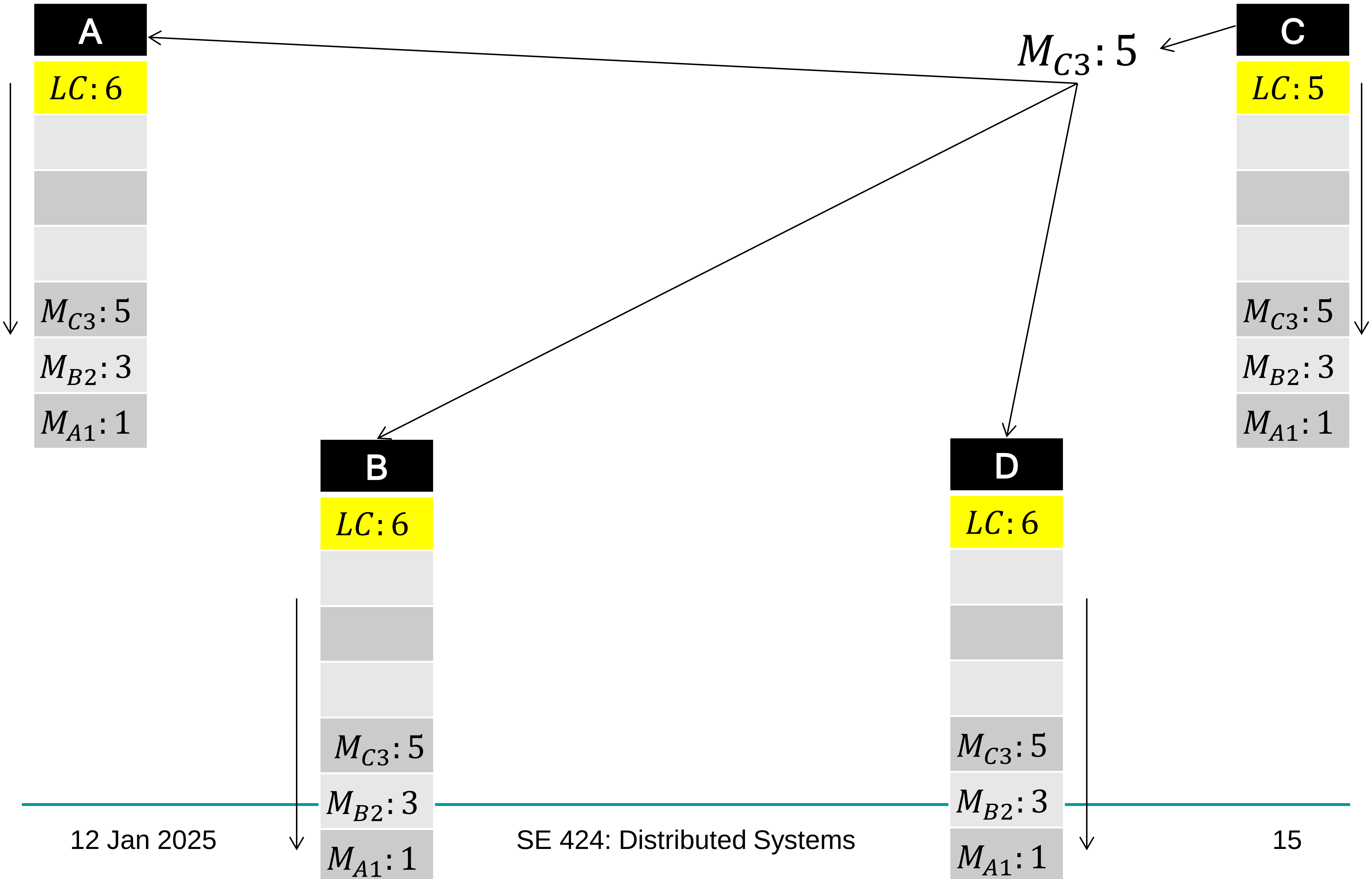
TOM Illustrated 2



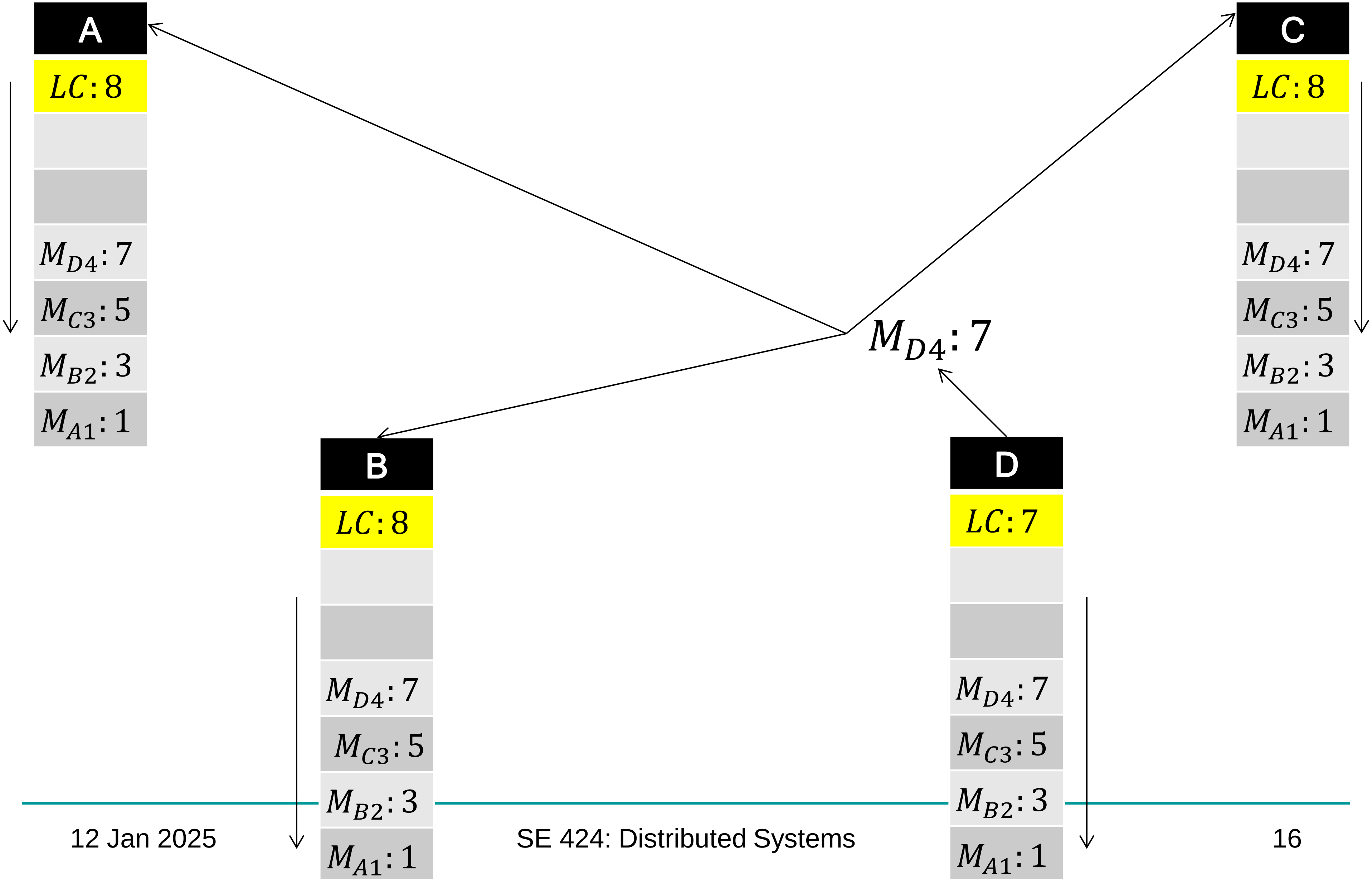
TOM Illustrated 3



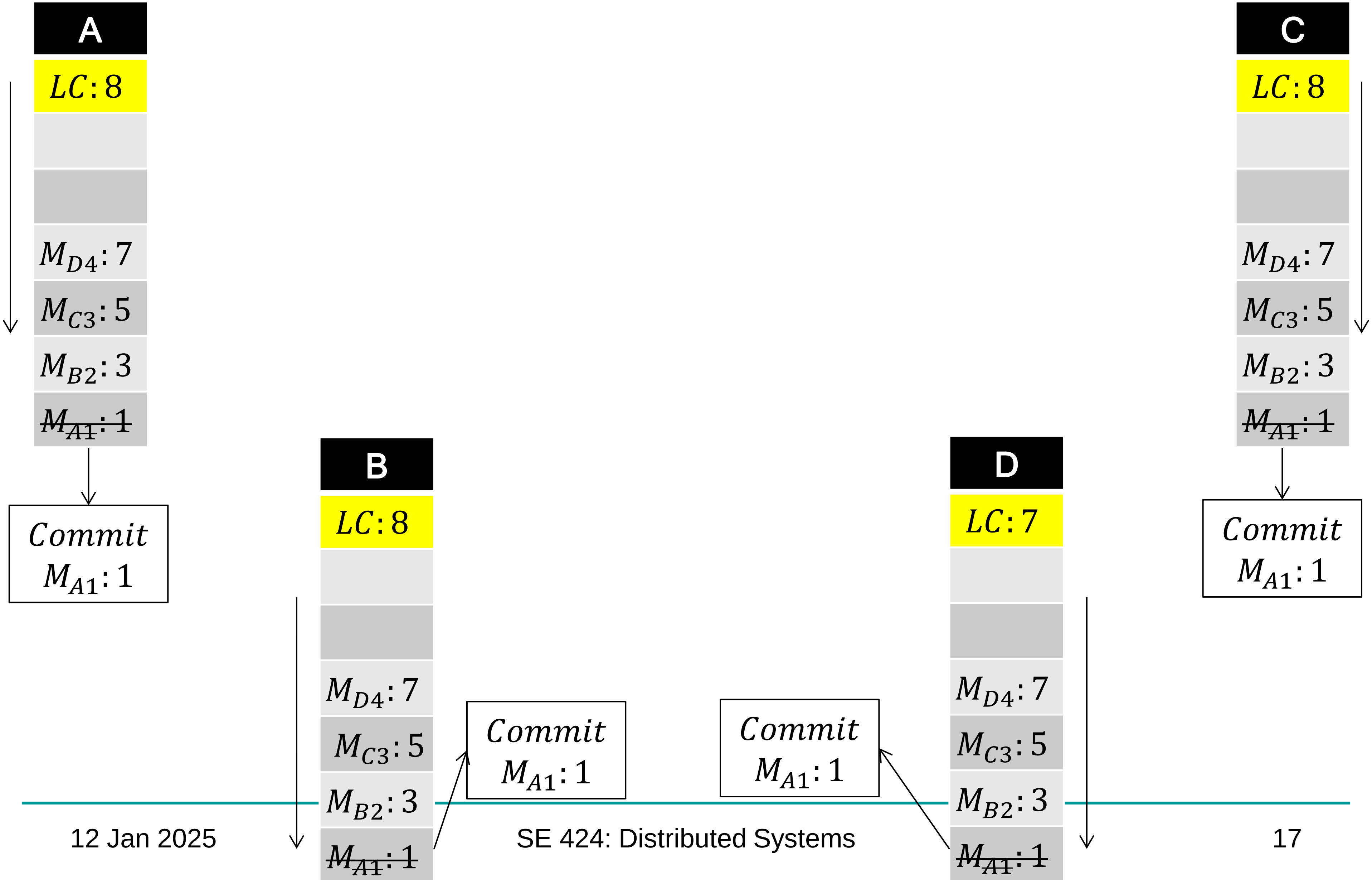
TOM Illustrated 4



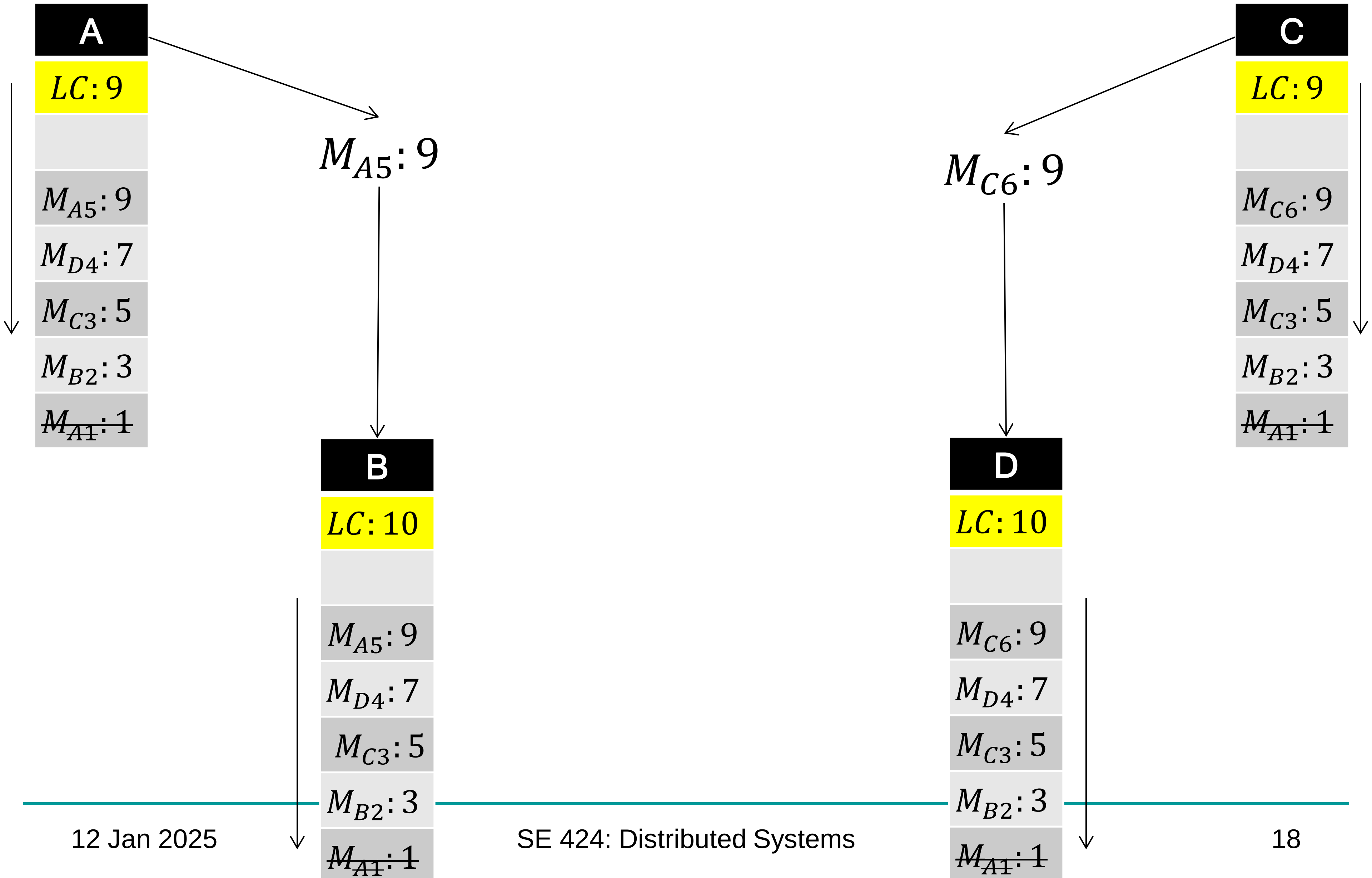
TOM Illustrated 5



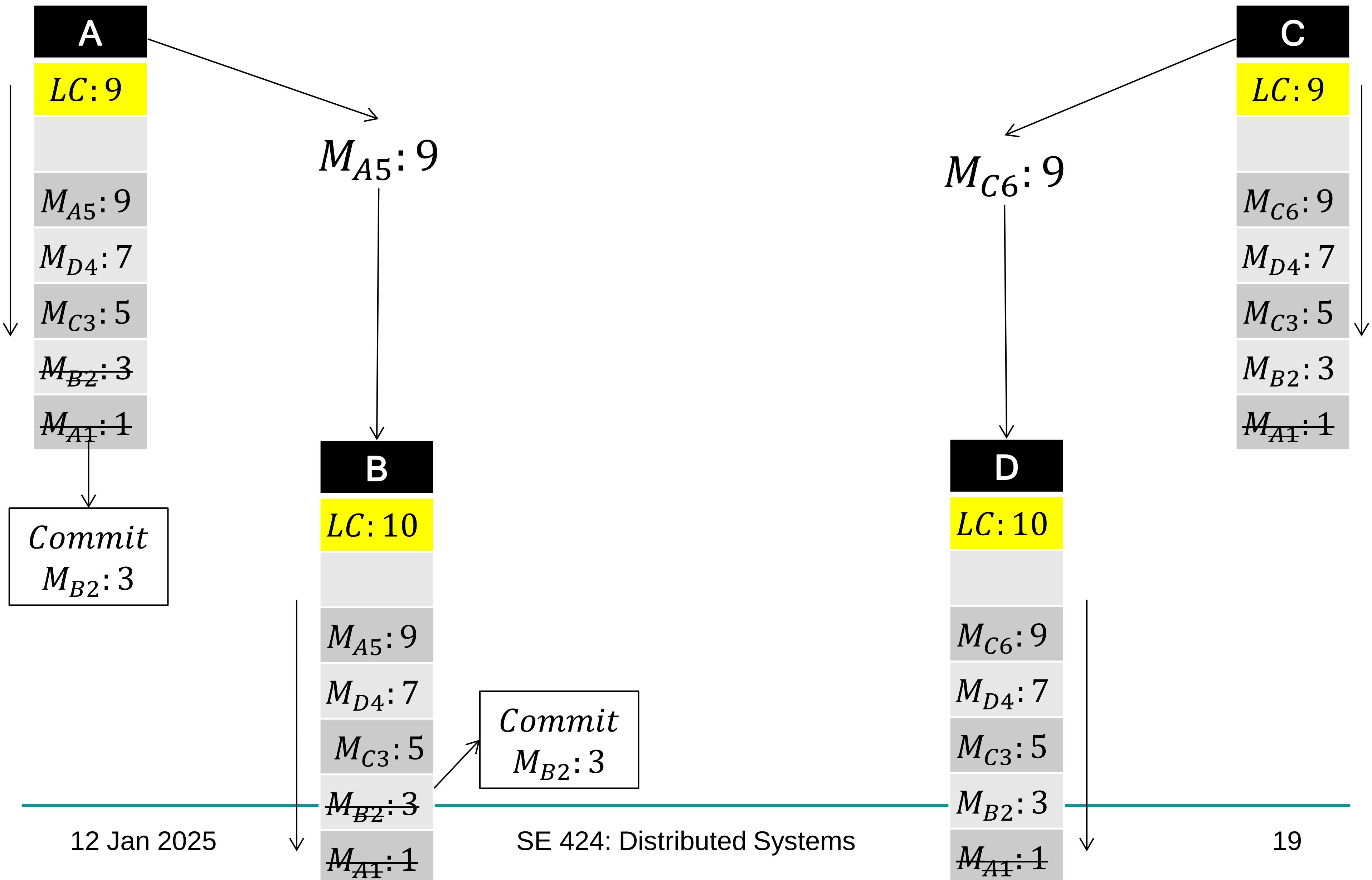
TOM Illustrated 6



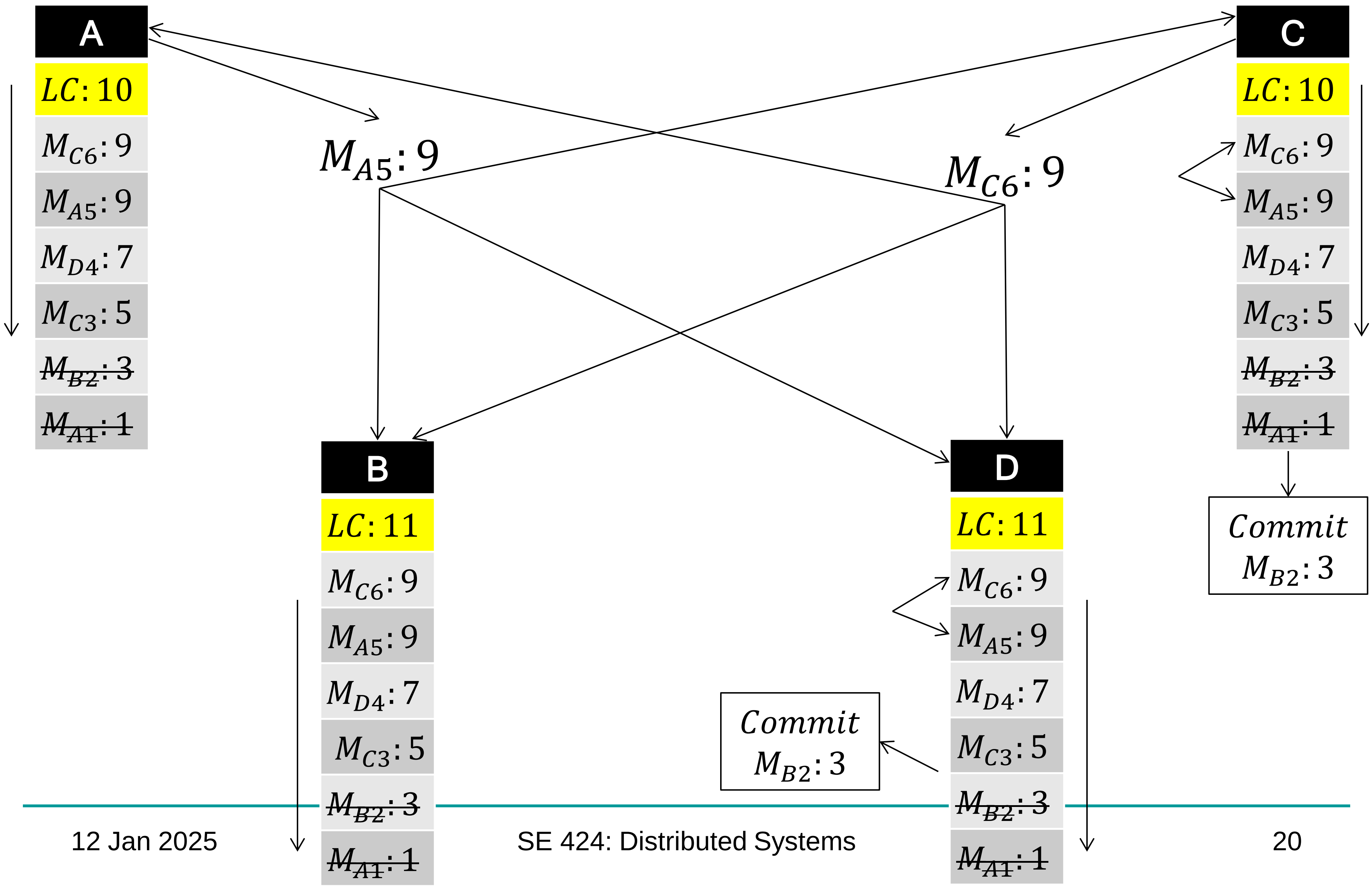
TOM Illustrated 7



TOM Illustrated 8



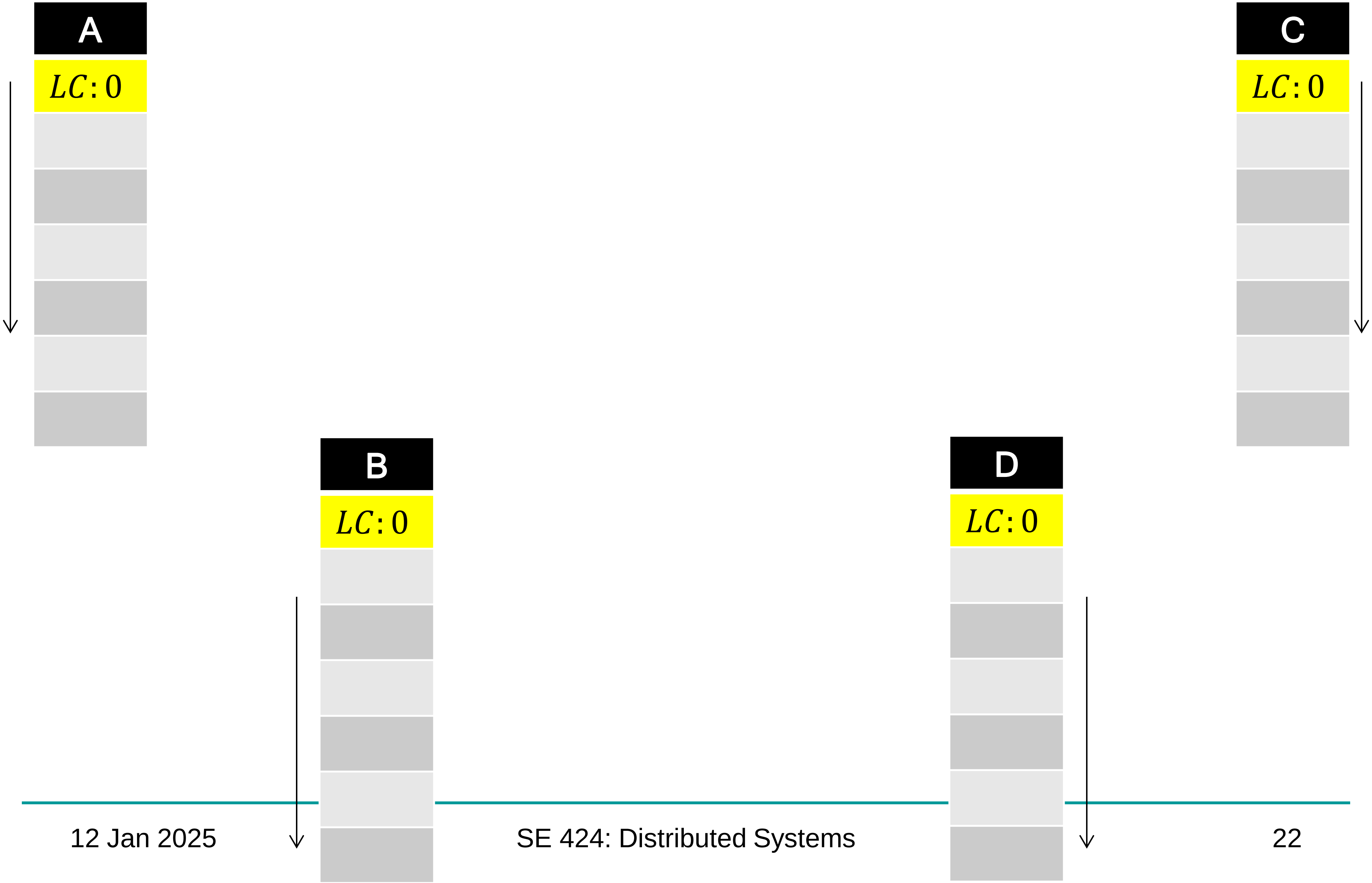
TOM Illustrated 9



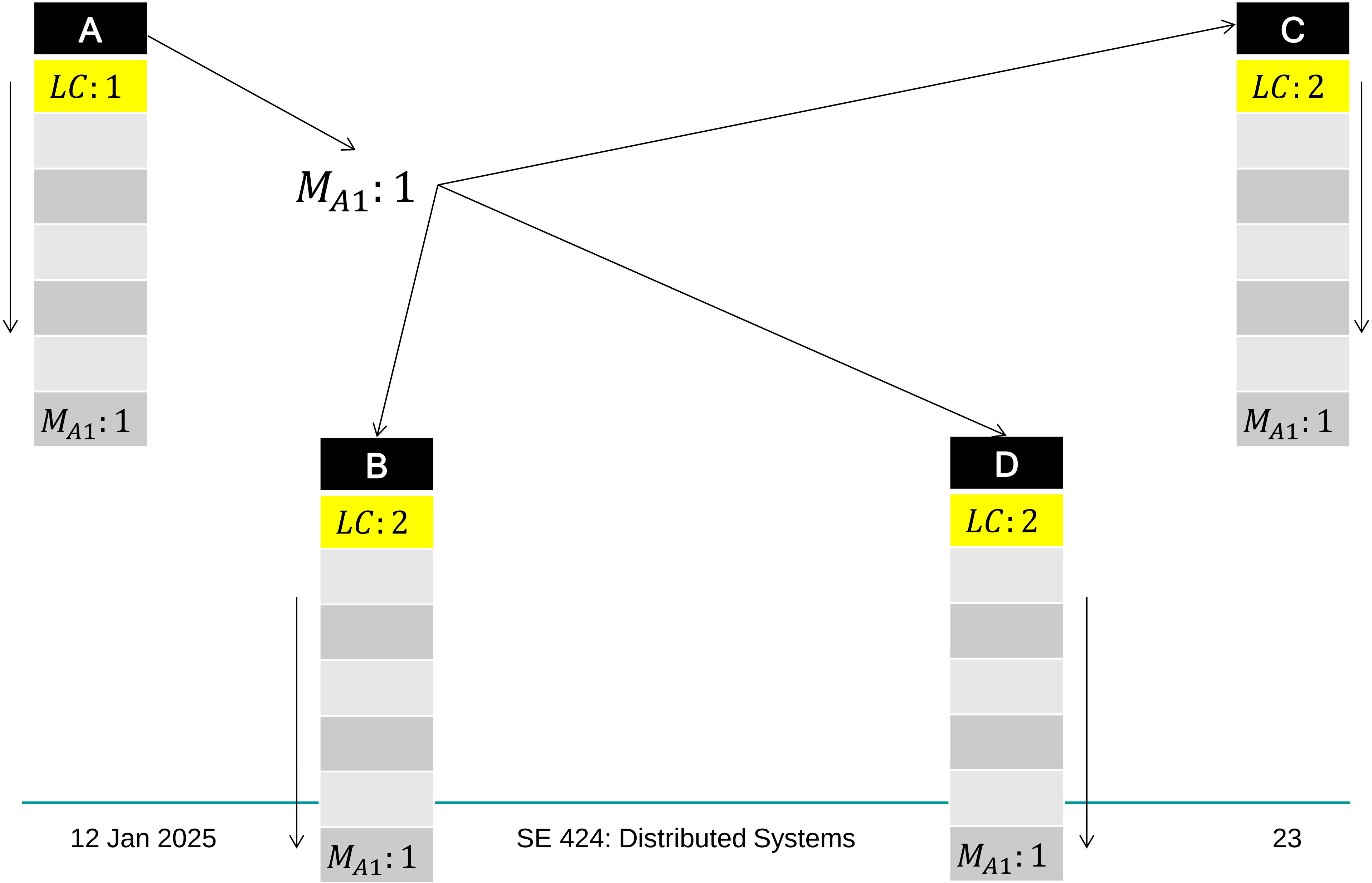
TOM with ACKs

- For systems with slower message sending, we can use ACKs to make TOM work better

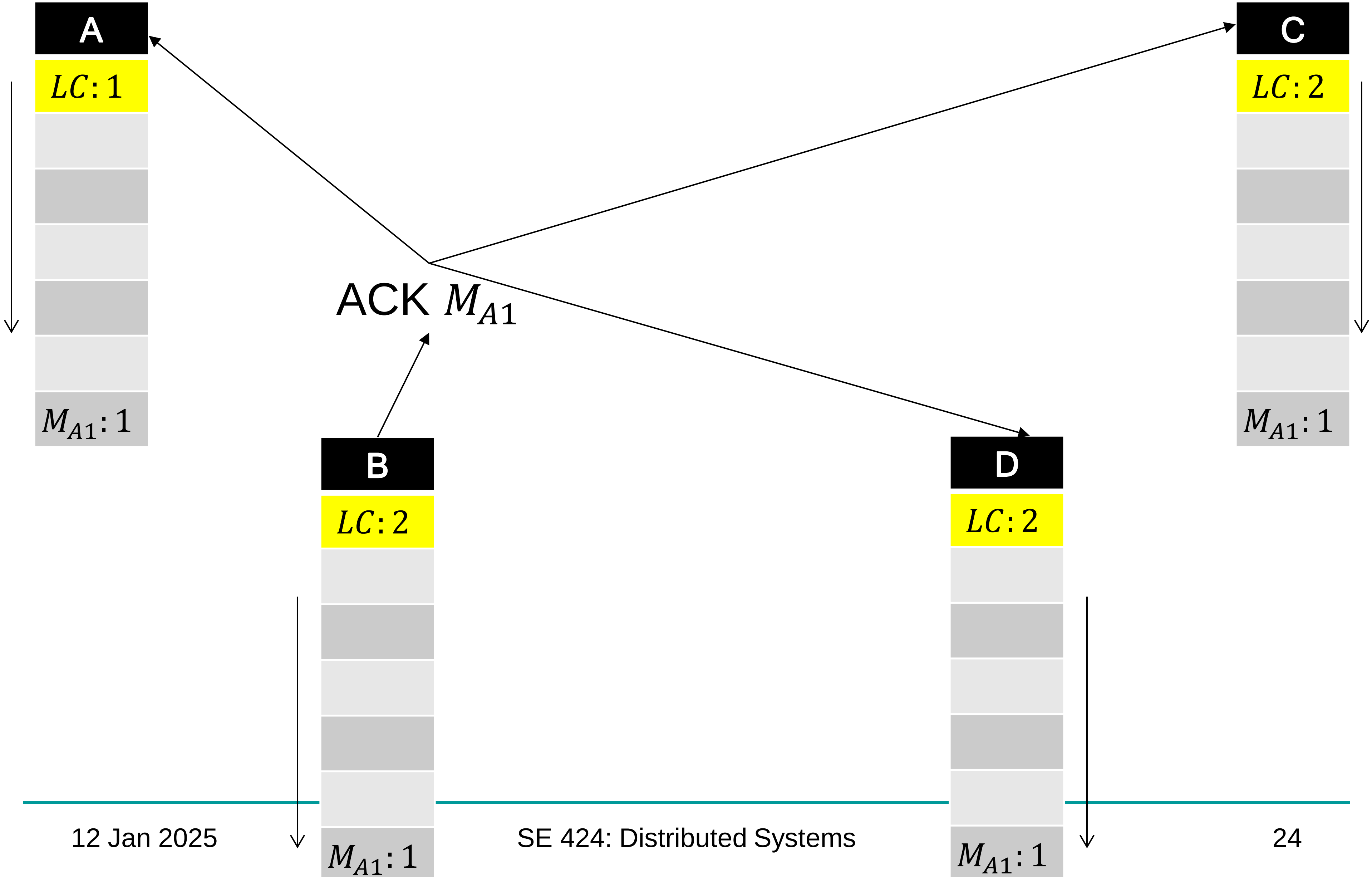
TOM ACKs Illustrated 1



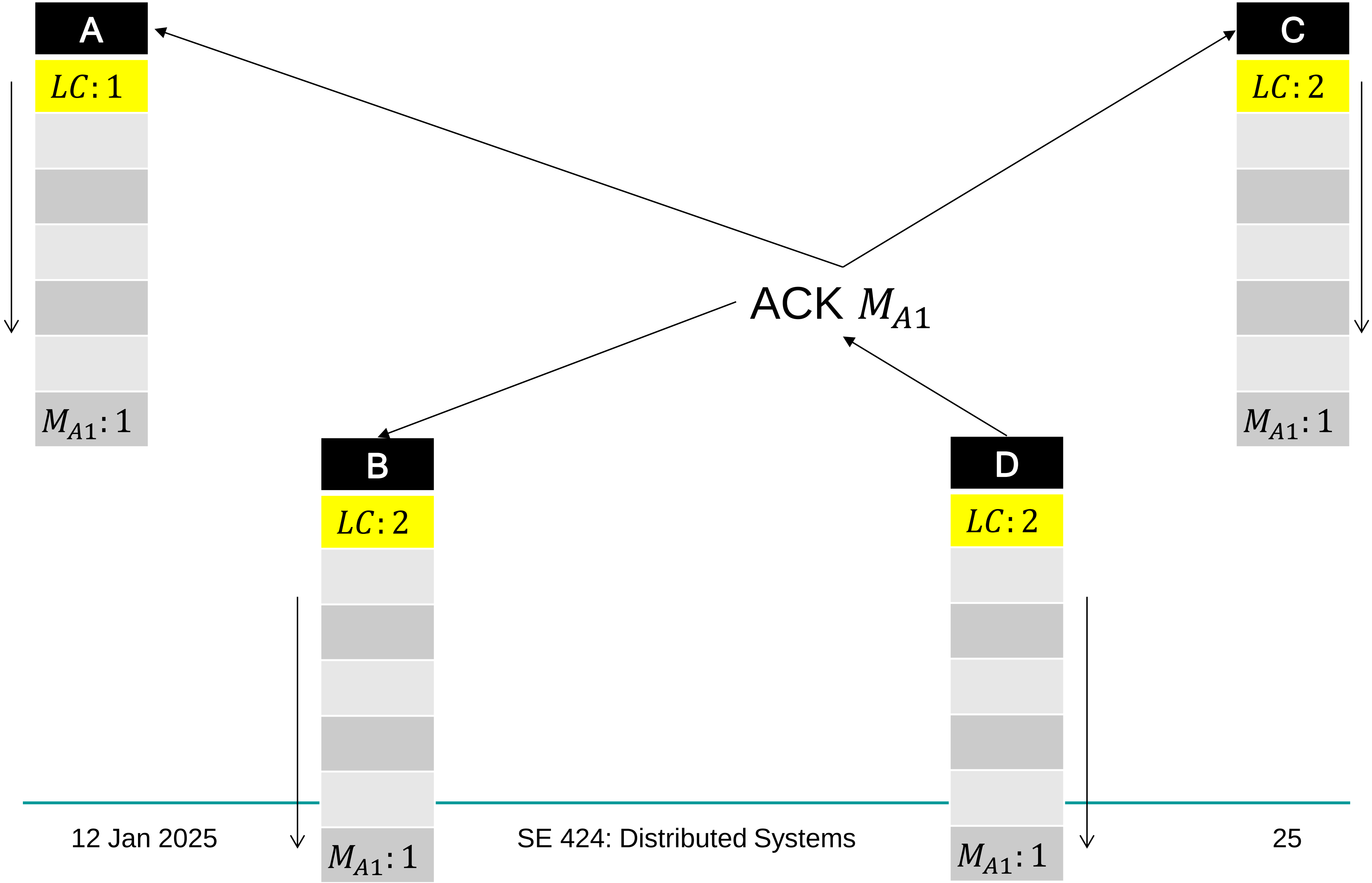
TOM ACKs Illustrated 2



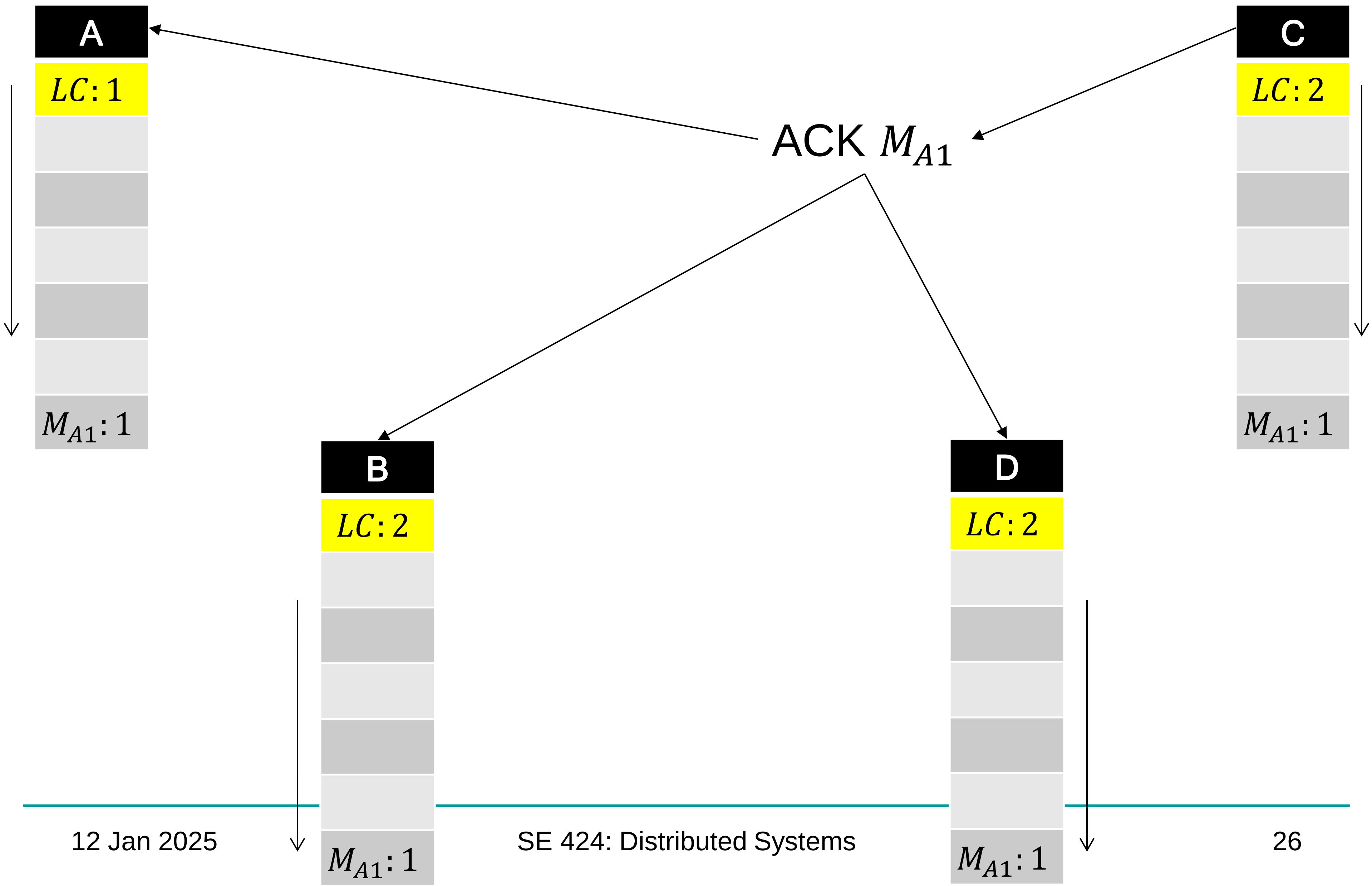
TOM ACKs Illustrated 3



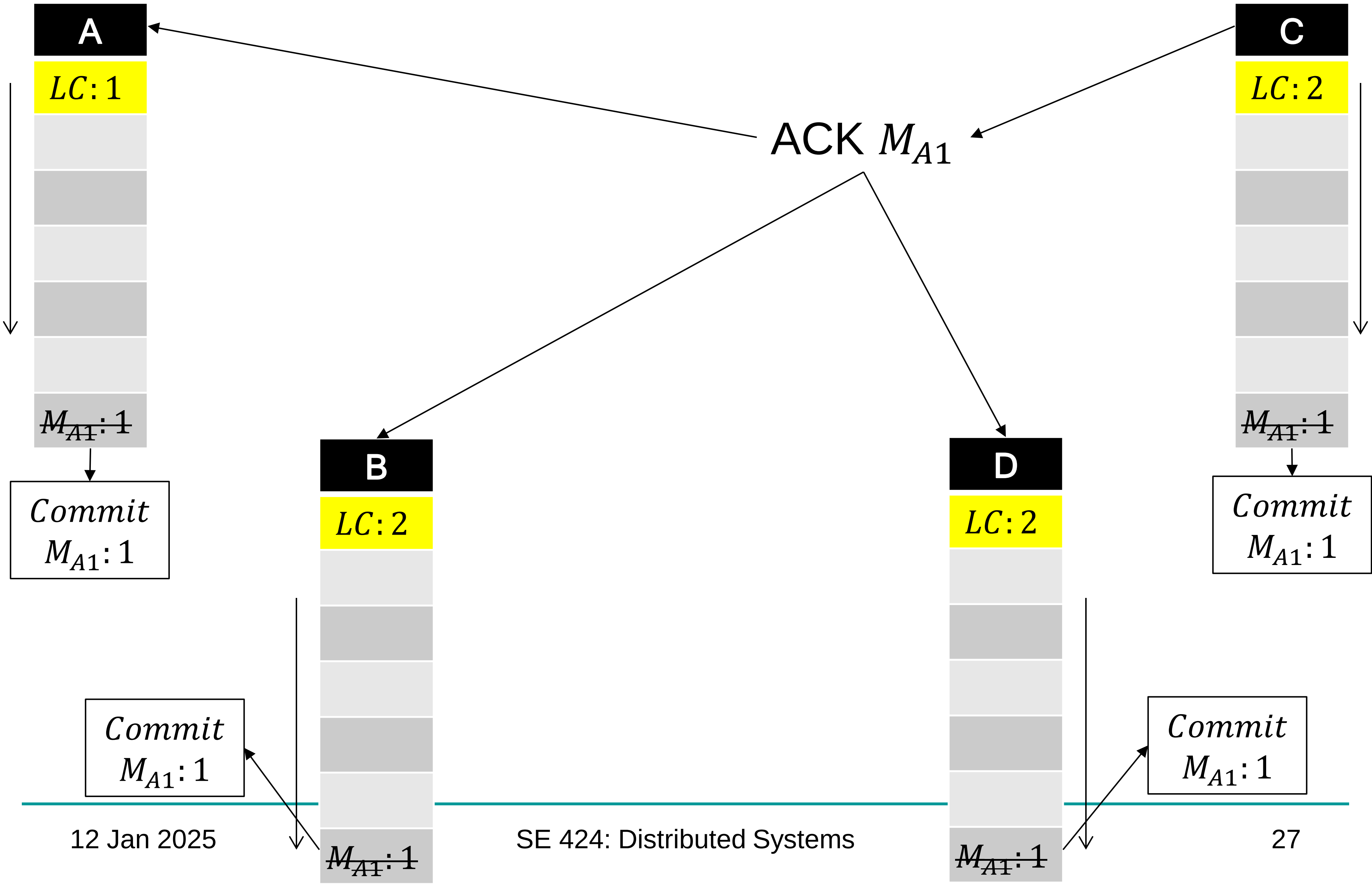
TOM ACKs Illustrated 4



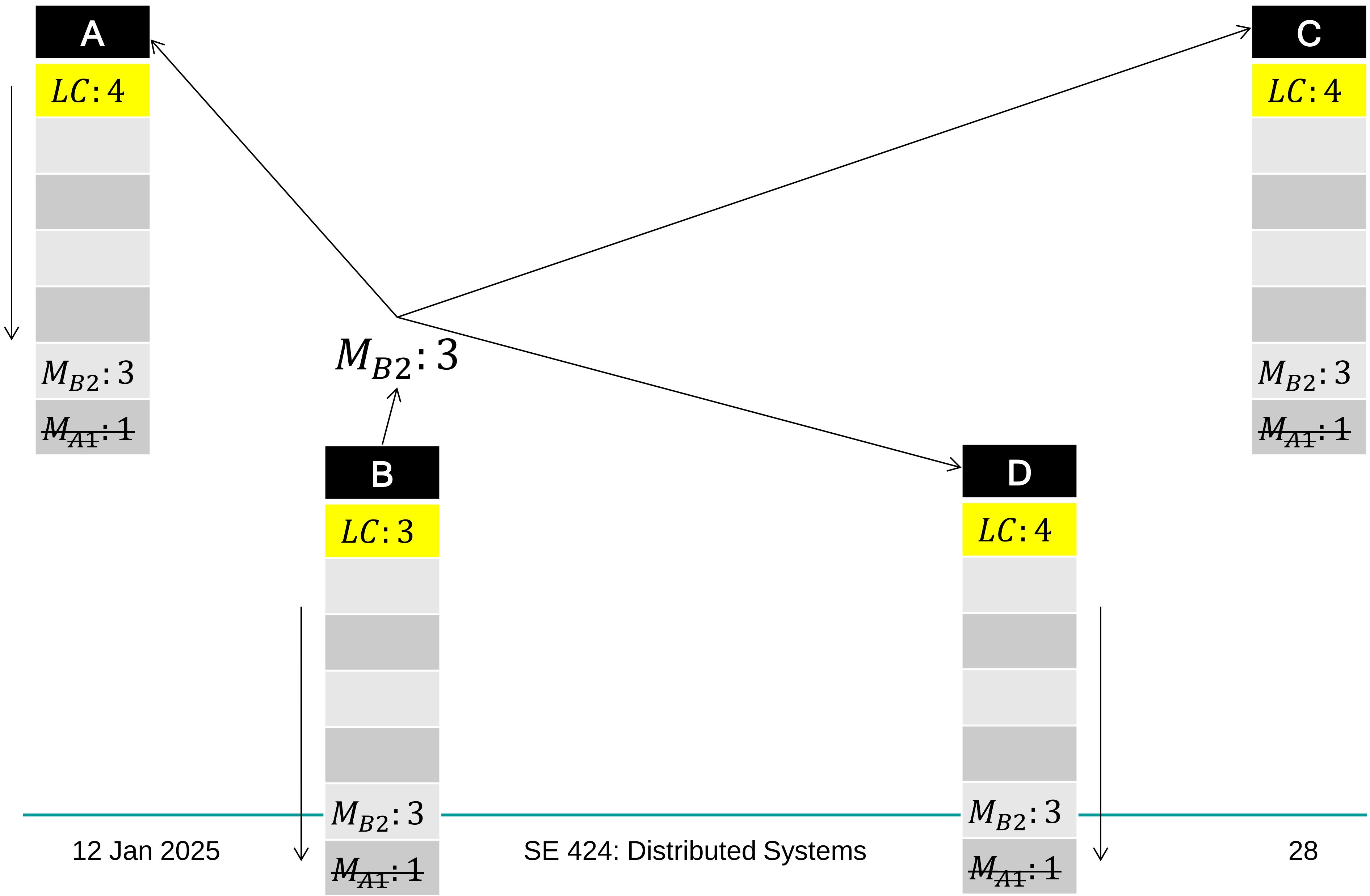
TOM ACKs Illustrated 5



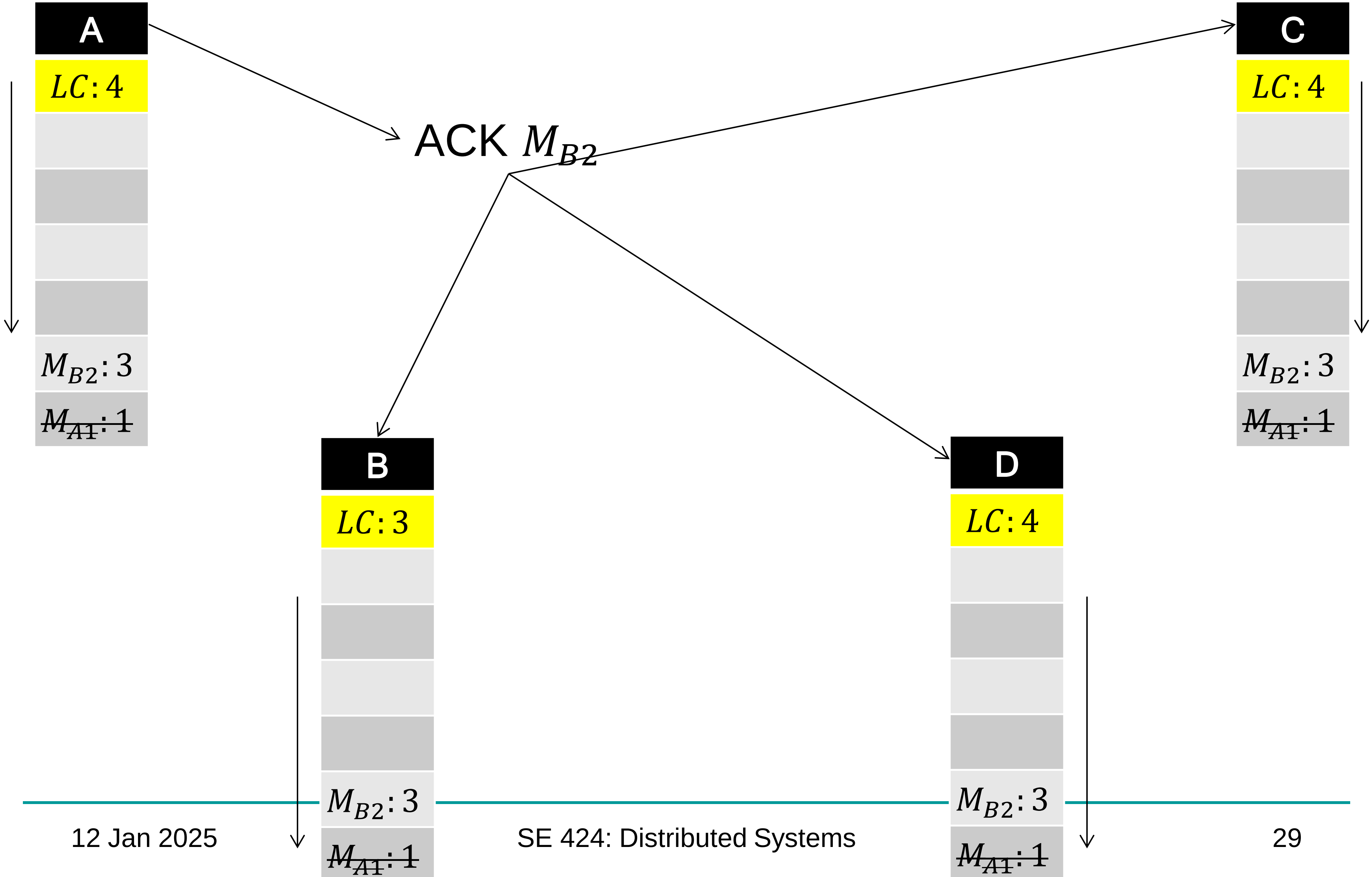
TOM ACKs Illustrated 6



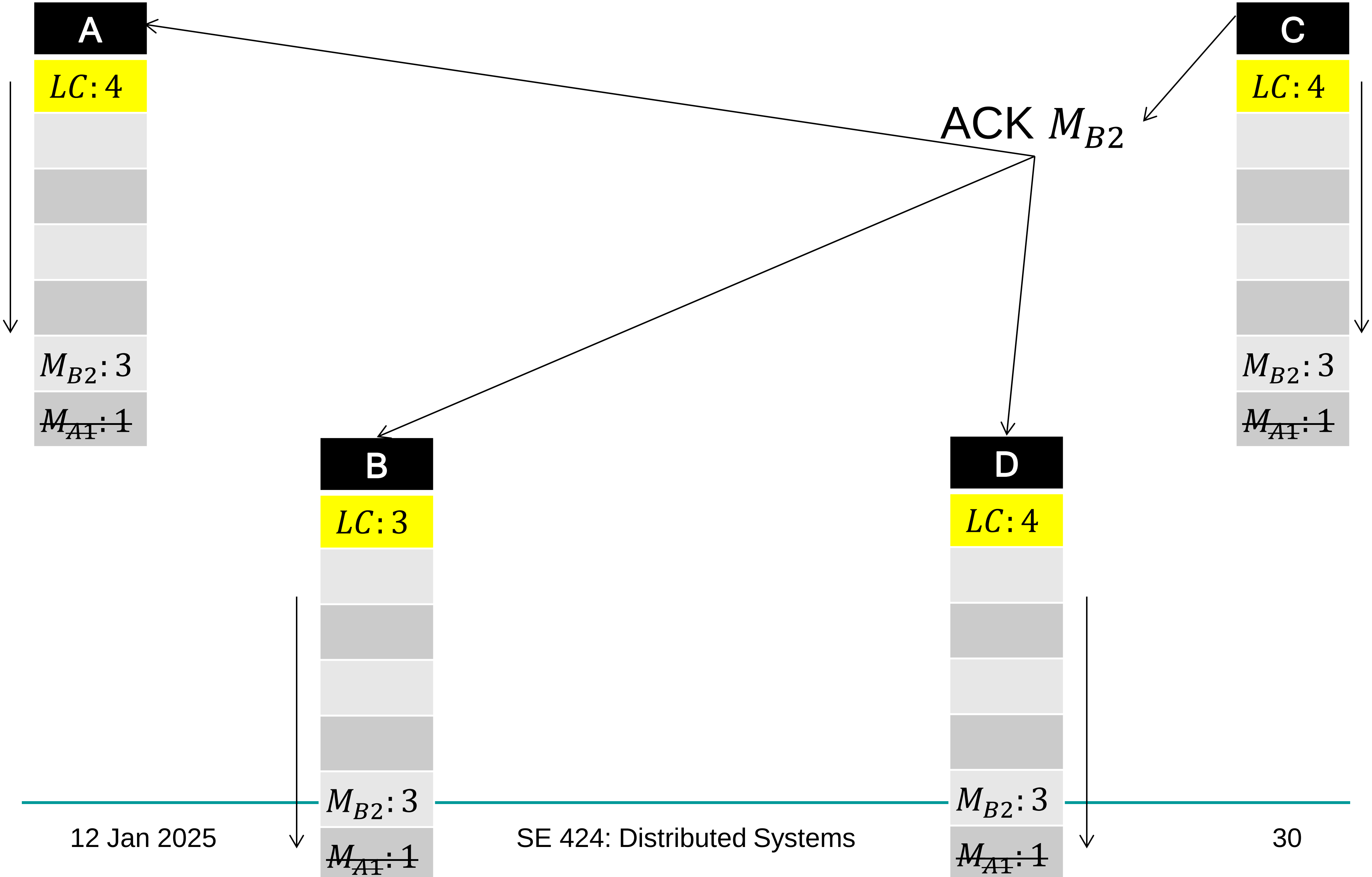
TOM ACKs Illustrated 7



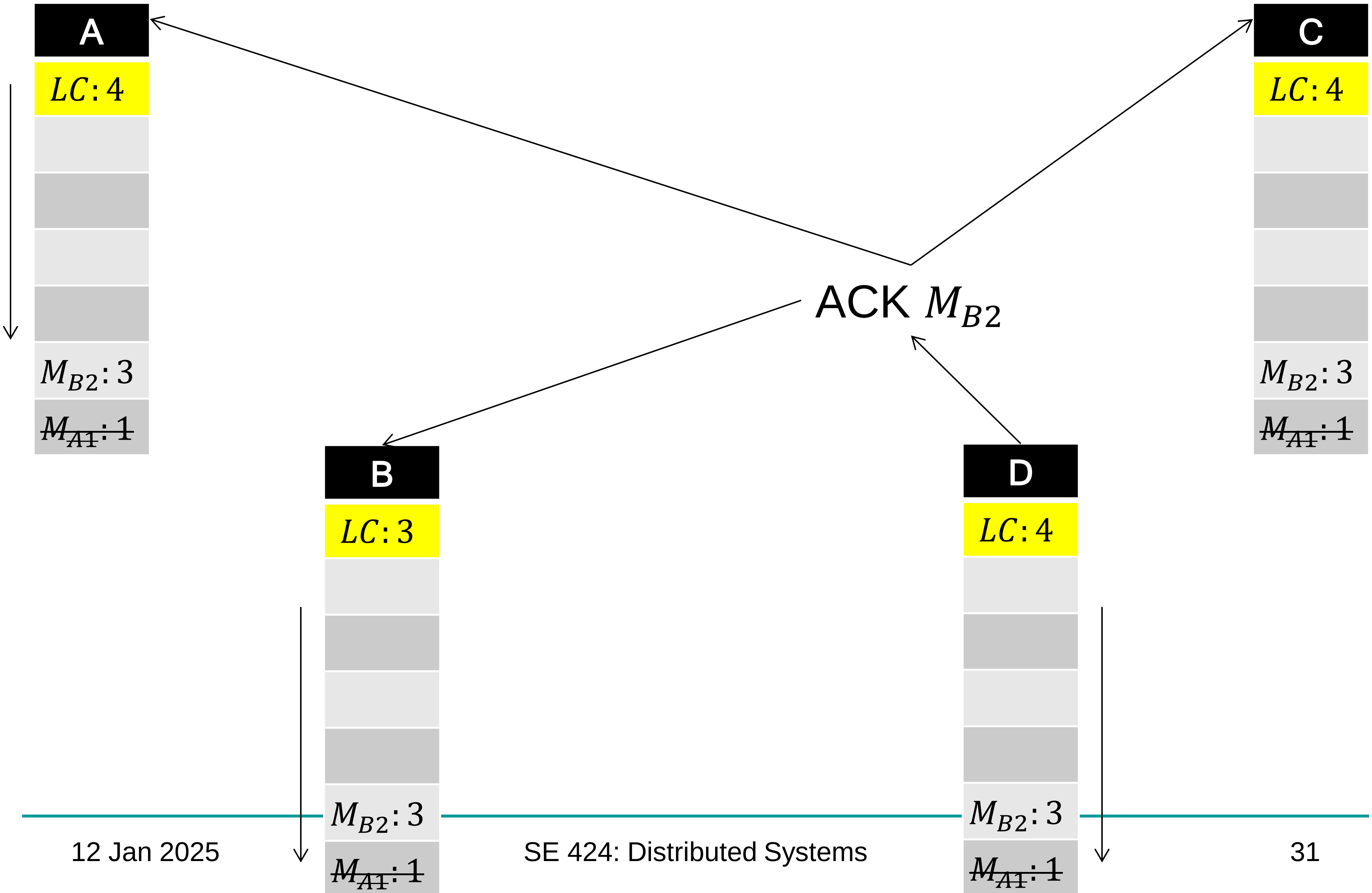
TOM ACKs Illustrated 8



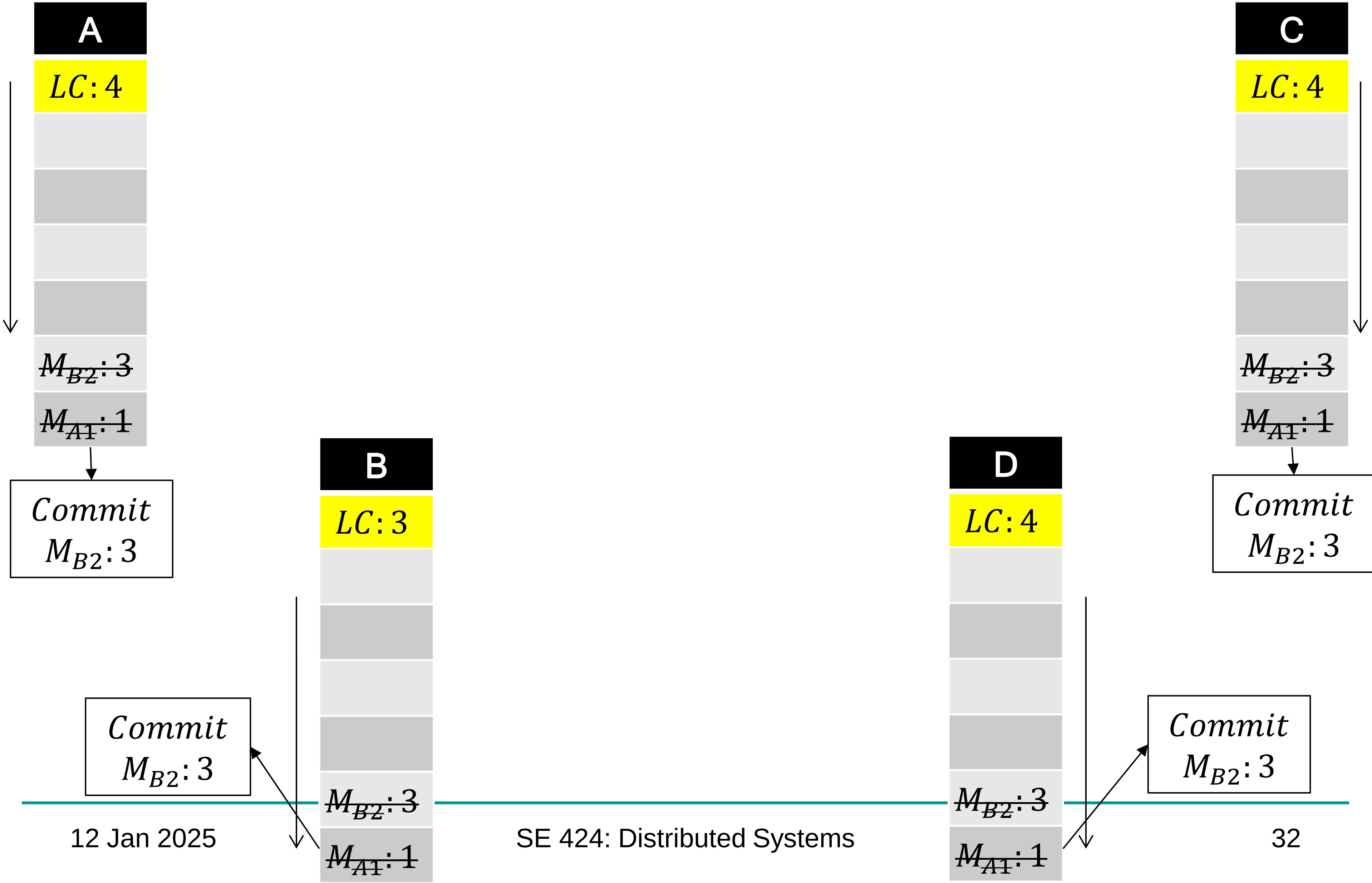
TOM ACKs Illustrated 9



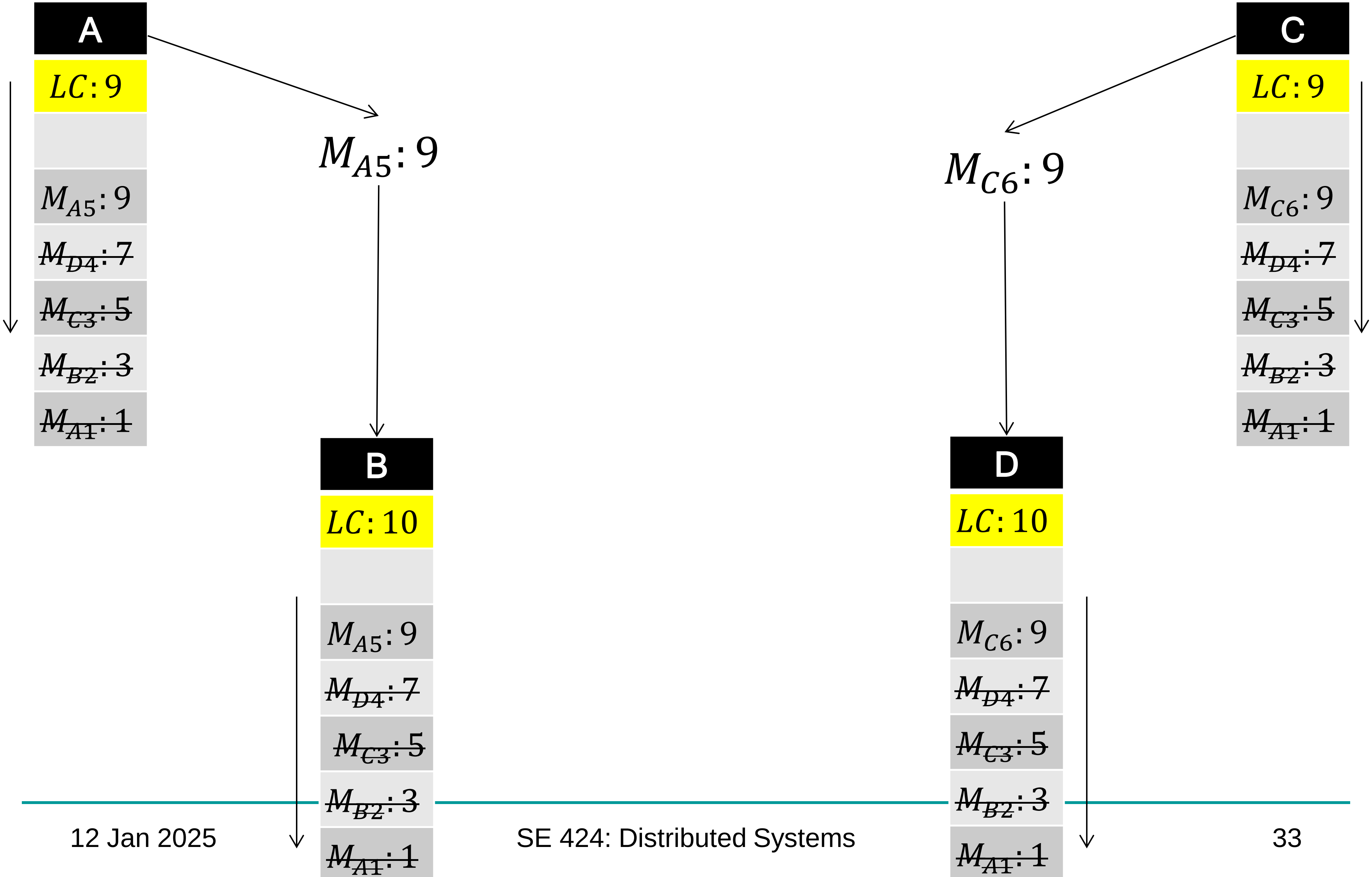
TOM ACKs Illustrated 10



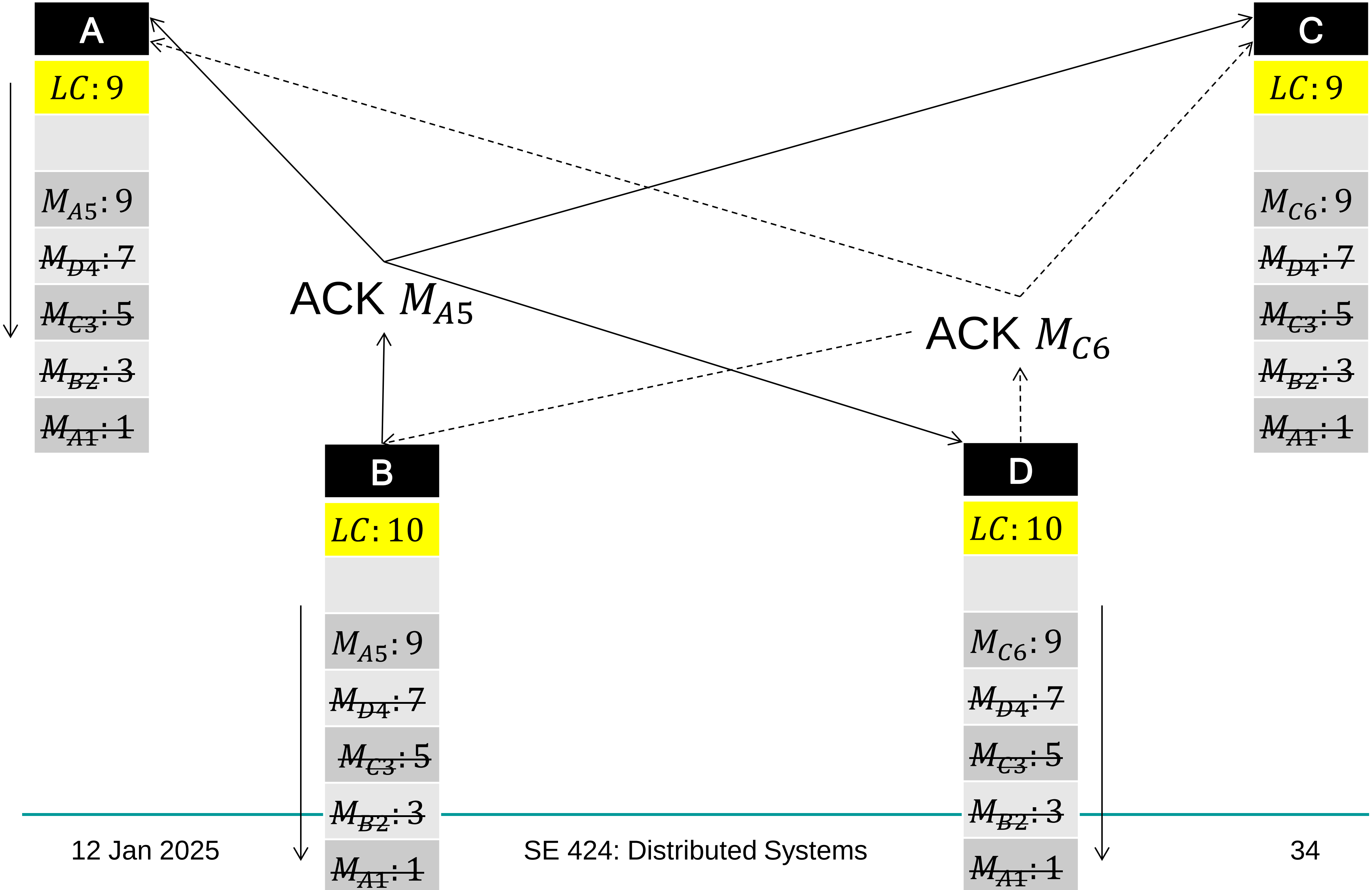
TOM ACKs Illustrated 11



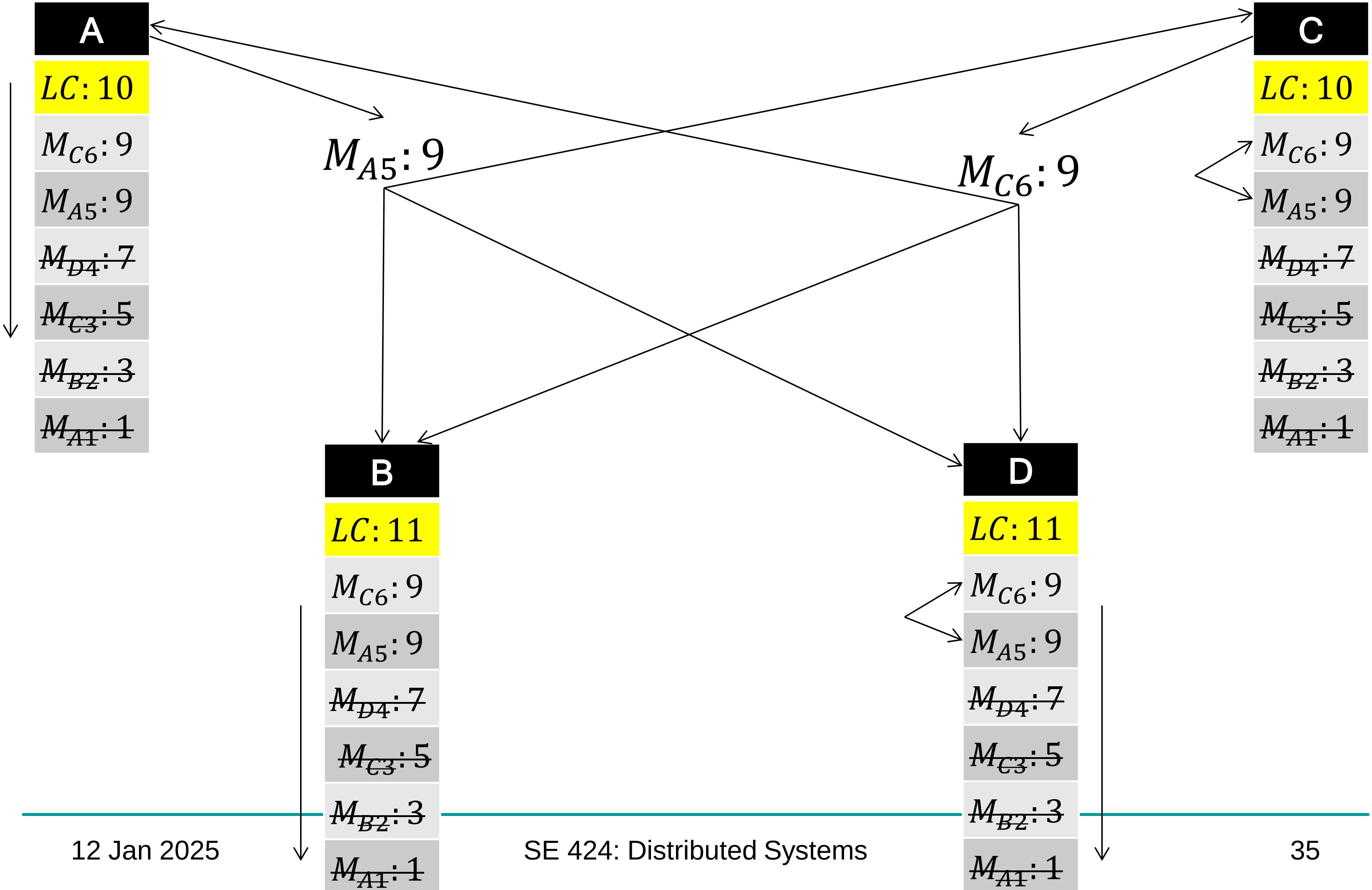
TOM ACKs Illustrated 12



TOM ACKs Illustrated 12



TOM ACKs Illustrated 13

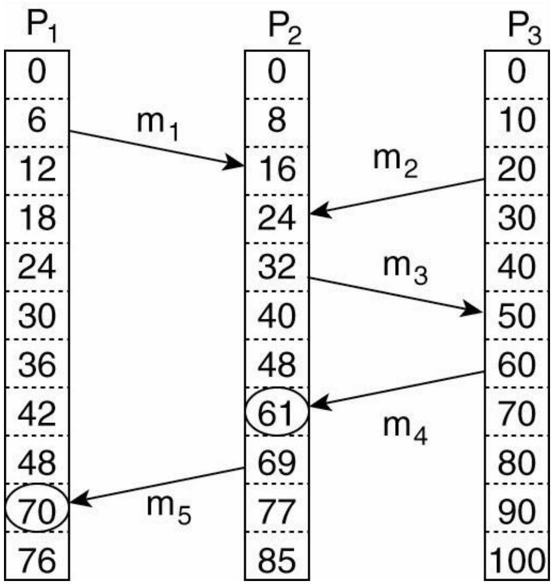


So Far

- Logical Clocks
  - Lamport
  - Totally Ordered Multicast
- (Mattern) Vector Clocks

Vector Clocks

Observation: Lamport's clocks do not guarantee that if $C(a) < C(b)$ that a **causally preceded** b:



Observation:

Event a: m_1 is received at $T = 16$.

Event b: m_2 is sent at $T = 20$.

We **cannot** conclude that a causally precedes b.

Vector Clocks

Solution:

- Each process P_i has an array $VC_i[1..n]$, where $VC_i[j]$ denotes the number of events that process P_i knows have taken place at process P_j
- When P_i sends a message m, it adds 1 to $VC_i[i]$, and sends VC_i along with m as **vector timestamp** $vt(m)$. Result: upon arrival, recipient knows P_i's timestamp.
- When a process P_j delivers a message m that it received from P_i with vector timestamp $ts(m)$, it
  1) updates each $VC_j[k]$ to $\max\{VC_j[k], ts(m)[k]\}$ for each k
  2) increments $VC_j[j]$ by 1.

Question: What does $VC_i[j] = k$ mean in terms of messages sent and received?

Vector and Lamport Clocks

Lamport Clocks

Rule 1: Each process has its own version of the global clock

Rule 2: Each process increments its global clock version when it performs an internal event or sends a message (which includes a timestamp)

Rule 3: When a process receives a message from another process it updates its global clock version **if the received timestamp is larger.**

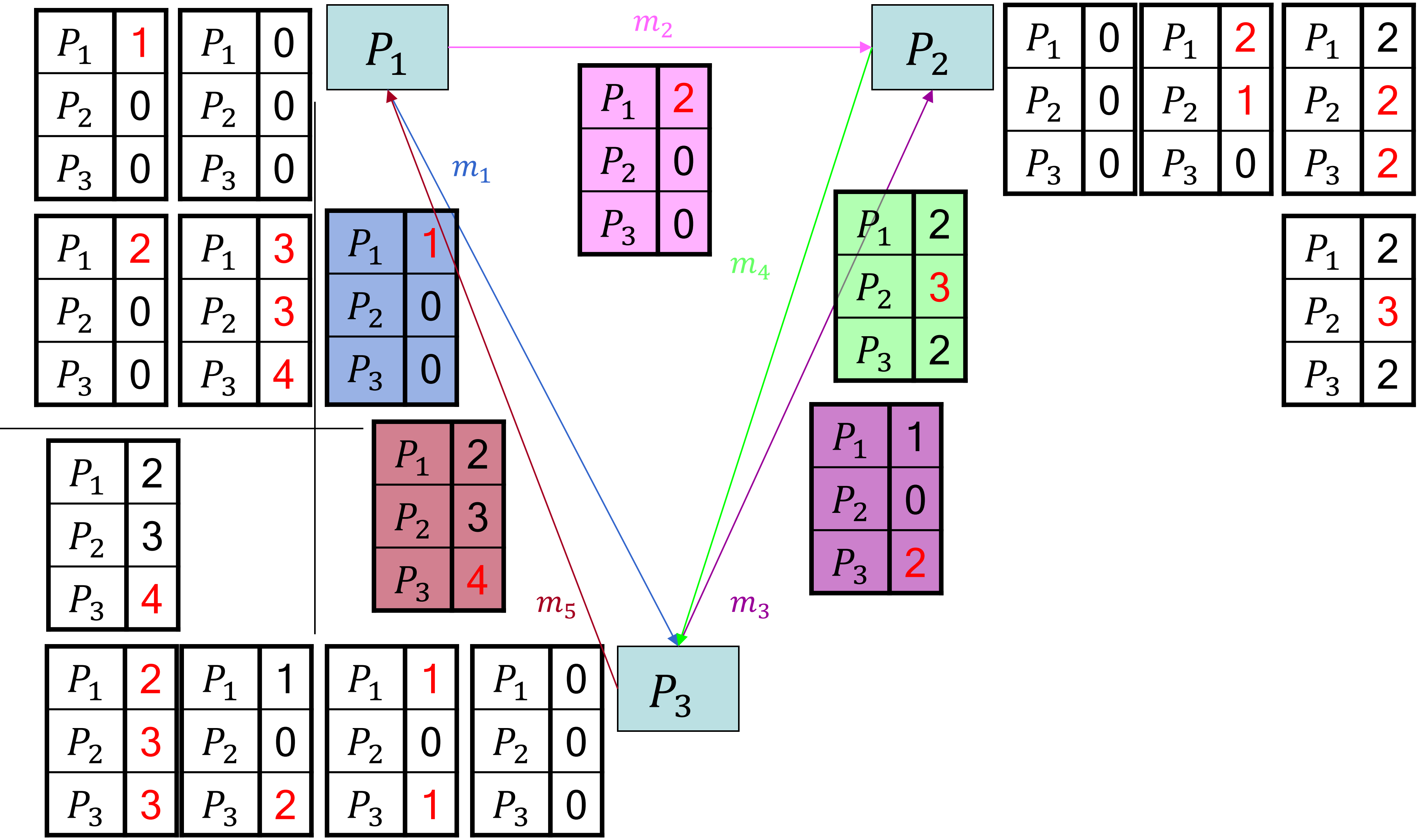
Vector clocks

Rule 1: Each process has its own clock and a version of every other processes' clock.

Rule 2: Each process increments its own clock when it sends or receives a message.

Rule 3: When a process receives a message from another process it updates its version of the other clocks' timestamps **if the received timestamp is larger**

Vector Clock Example



Vector Clock Example

m_1

P_1	1
P_2	0
P_3	0

m_4

P_1	2
P_2	3
P_3	2

m_2

P_1	2
P_2	0
P_3	0

m_5

P_1	2
P_2	3
P_3	4

m_3

P_1	1
P_2	0
P_3	2

1. $m_1 < m_2$
2. $m_1 < m_3$
3. $m_1 < m_4$
4. $m_1 < m_5$
5. $m_2 <> m_3$
6. $m_2 < m_4$
7. $m_2 < m_5$
8. $m_3 < m_4$
9. $m_3 < m_5$
10. $m_4 < m_5$

Conclusion

- Logical Clocks
  - Lamport
  - Totally Ordered Multicast
- (Mattern) Vector Clocks