ORDERED COMMUNICATIONS 2 TOTAL ORDER

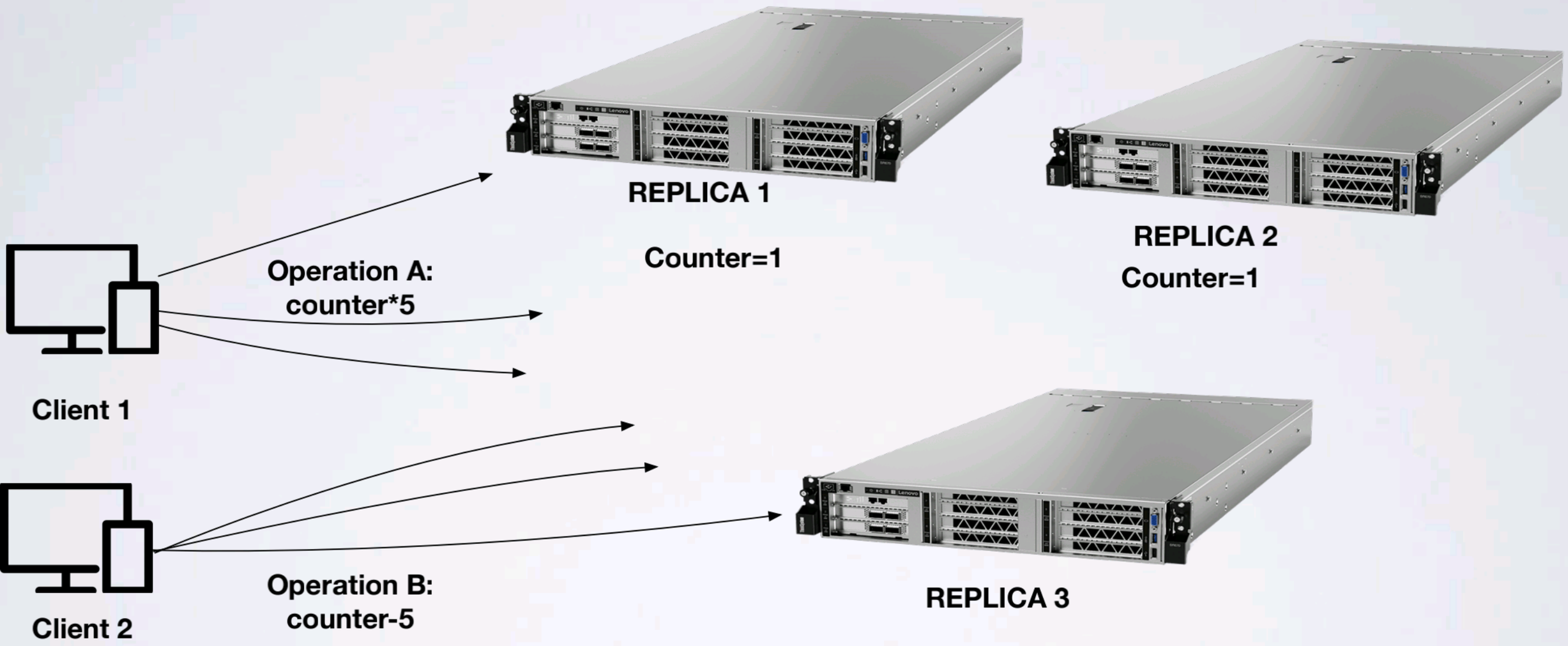
DISTRIBUTED SYSTEMS

Master of Science in Cyber Security

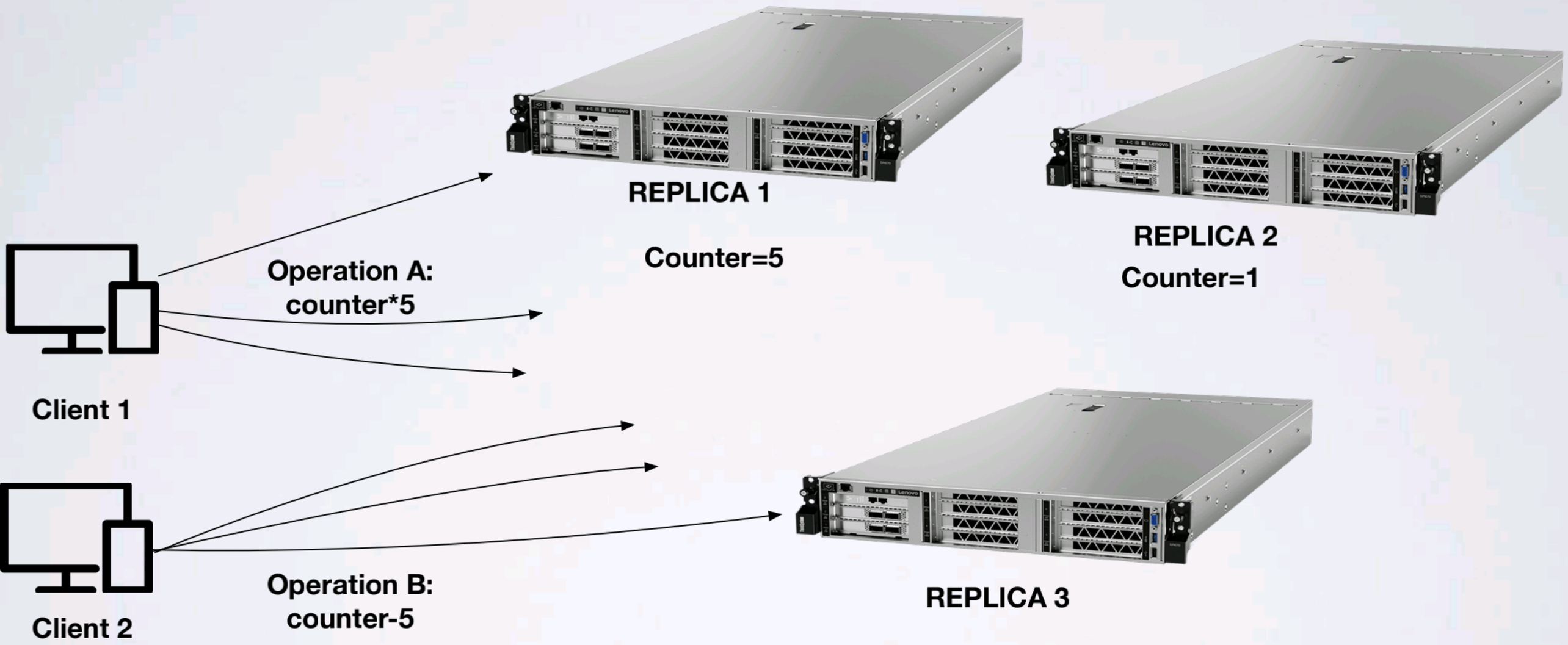
SAPIENZA
UNIVERSITÀ DI ROMA

CIS SAPIENZA
CYBER INTELLIGENCE AND INFORMATION SECURITY

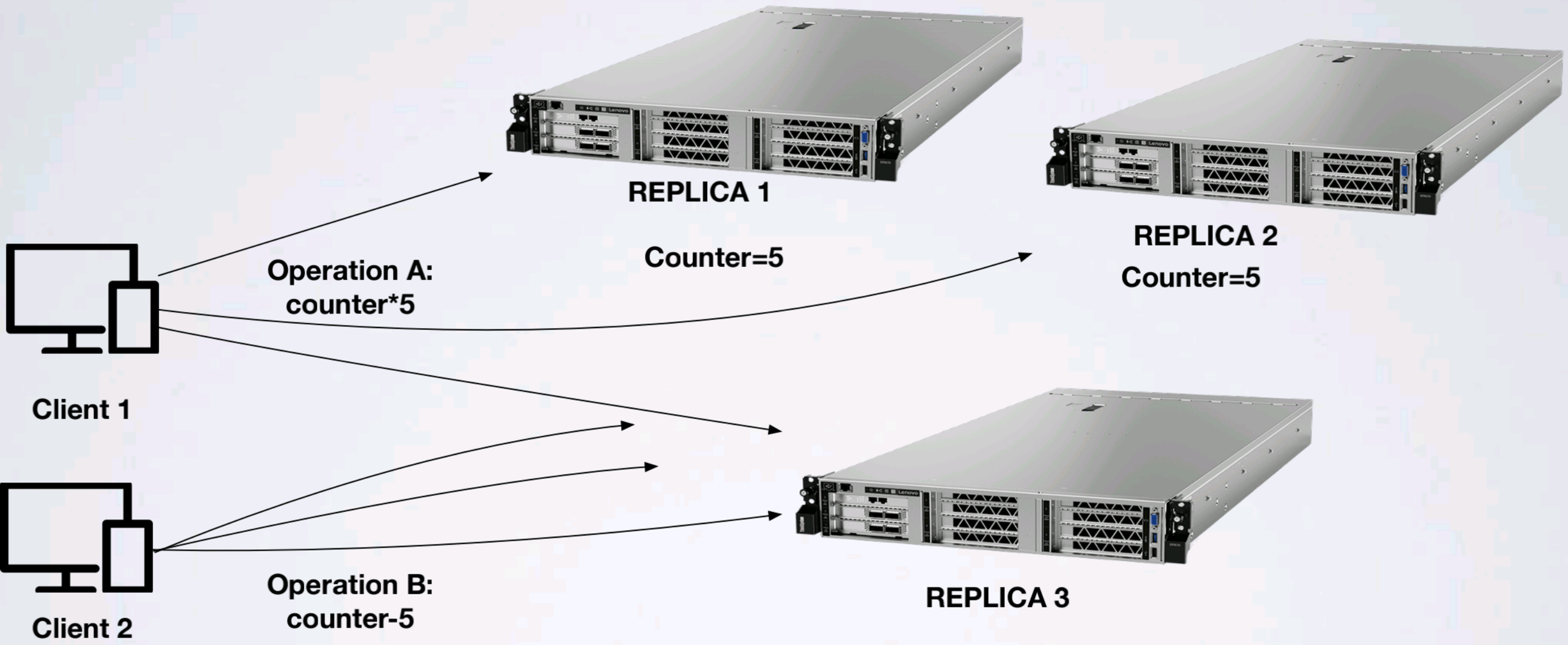
USES OF **TO-BROADCAST/(ATOMIC BROADCAST)**



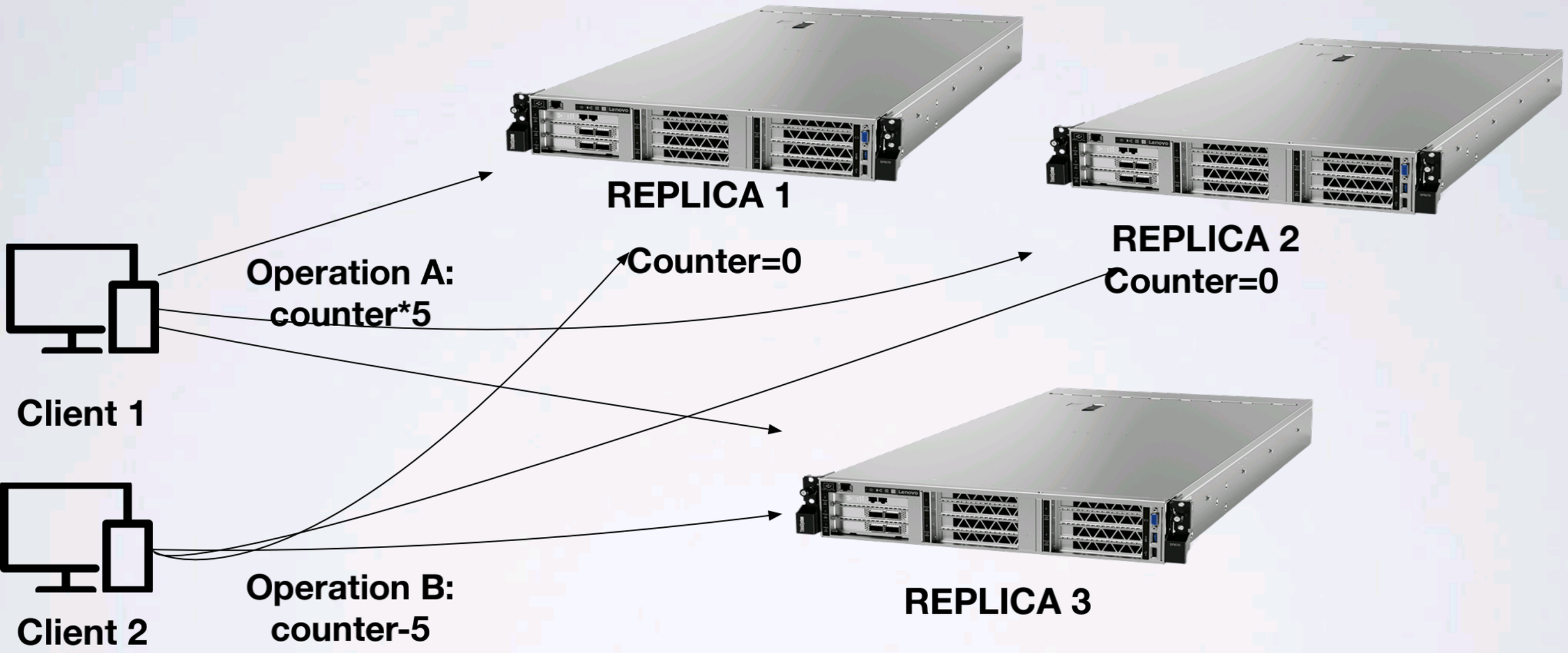
USES OF **TO-BROADCAST/(ATOMIC BROADCAST)**



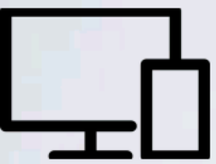
USES OF **TO-BROADCAST/(ATOMIC BROADCAST)**



USES OF **TO-BROADCAST/(ATOMIC BROADCAST)**



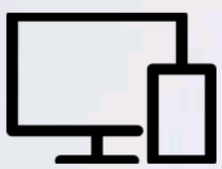
USES OF **TO-BROADCAST/(ATOMIC BROADCAST)**



Client 1

Operation A:
counter*5

TO BROADCAST



Client 2

Operation B:
counter-5

TO BROADCAST

REPLICA 1

Operation A
Operation B

Counter=0

TO BROADCAST

REPLICA 2

Counter=0

Operation A
Operation B

TO BROADCAST

REPLICA 3

Counter=0

Operation A
Operation B

TO BROADCAST

TO SPECIFICATIONS

Total order specifications are usually composed by four properties:

- a **Validity** property guarantees that messages sent by correct processes will eventually be delivered at least by correct processes;
- an **Integrity** property guarantees that no spurious or duplicate messages are delivered;
- an **Agreement** property ensures that (at least correct) processes deliver the same set of messages;
- an **Order** property constrains (at least correct) processes delivering the same messages to deliver them in the same order.

TO SPECIFICATIONS

Total Order Broadcast

- V = Validity
- I = Integrity
- A = Agreement
- O = Order

Distinct specifications arise from distinct formulations of each property

- uniform vs non-uniform
- A uniform property imposes restrictions on the behavior of (at least) correct processes on the basis of events occurred in some process

THE AGREEMENT PROPERTY

Uniform Agreement (UA)

If a process (correct or not) TODelivers a message m, then all correct processes will eventually TODeliver m

Non Uniform Agreement (NUA)

If a correct process TODelivers a message m, then all correct processes will eventually TODeliver m

Constrains the set of delivered messages

Correct processes always deliver the same set of messages M

Each faulty process p delivers a set M_p

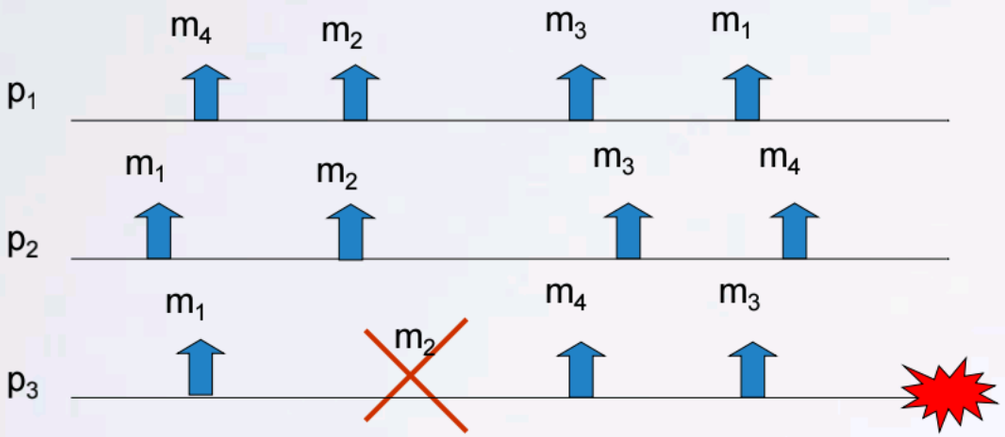
UA: $M_p \subseteq M$

NUA: M_p ***can be*** s.t. $M_p \cap M \neq \emptyset$

THE AGREEMENT PROPERTY

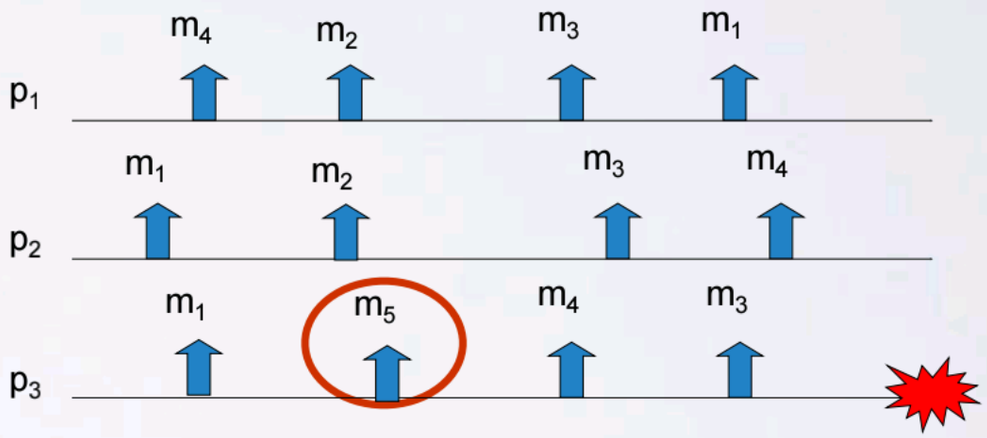
Uniform Agreement (UA)

If a process (correct or not) TODelivers a message m, then all correct processes will eventually TODeliver m



Non Uniform Agreement (NUA)

If a correct process TODelivers a message m, then all correct processes will eventually TODeliver m



Module 6.1: Interface and properties of regular total-order broadcast

Module:

Name: TotalOrderBroadcast, **instance** *tob*.

Events:

Request: $\langle$ *tob*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *tob*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

- **TOB1:** *Validity:* If a correct process p broadcasts a message m, then p eventually delivers m.

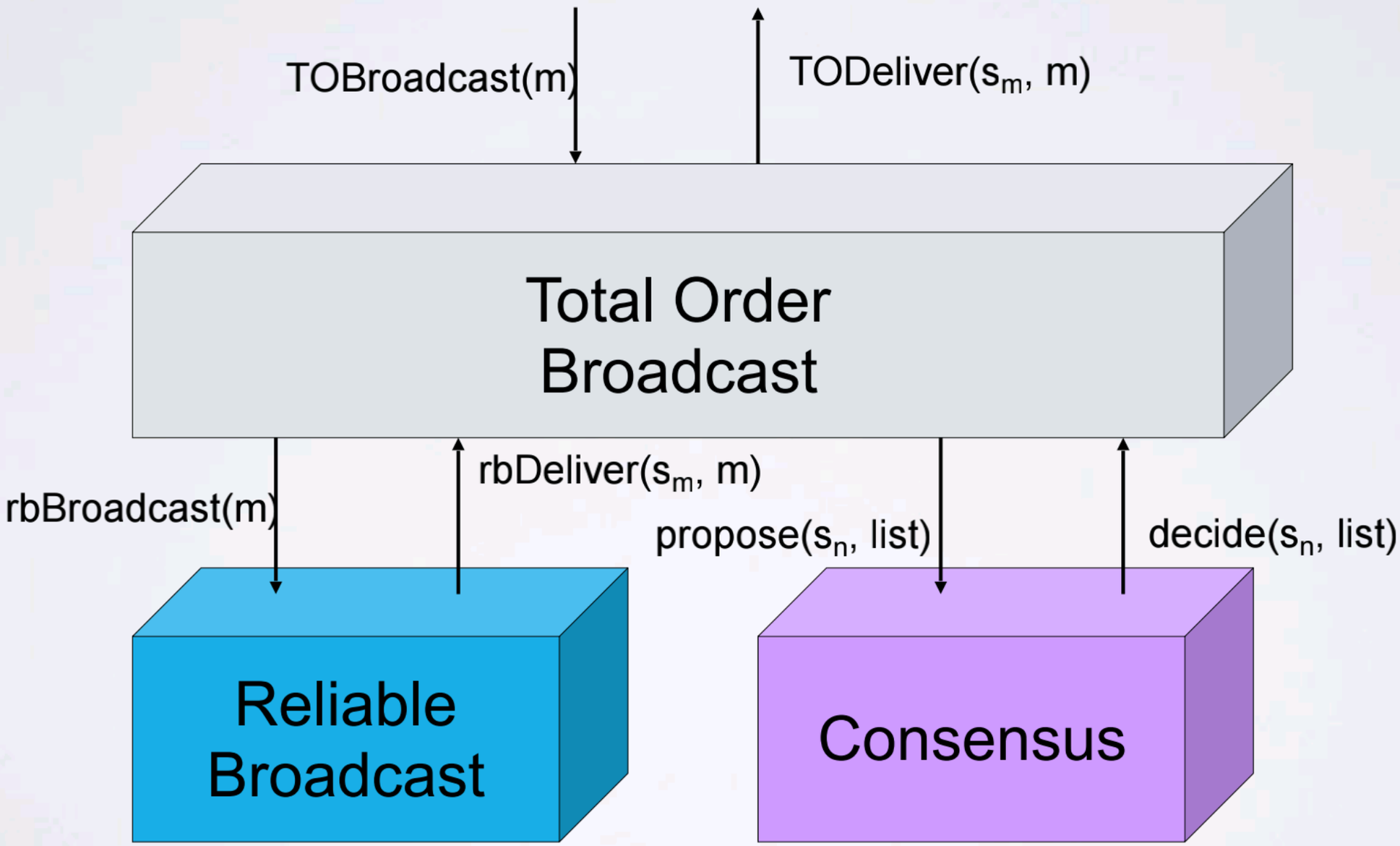
TOB2: *No duplication:* No message is delivered more than once.

TOB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

- **TOB4:** *Agreement:* If a message m is delivered by some correct process, then m is eventually delivered by every correct process.

TOB5: *Total order:* Let m_1 and m_2 be any two messages and suppose p and q are any two correct processes that deliver m_1 and m_2. If p delivers m_1 before m_2, then q delivers m_1 before m_2.

TOTAL ORDER IMPLEMENTATION



TOTAL ORDER ALGORITHM

Algorithm 6.1: Consensus-Based Total-Order Broadcast

Implements:
TotalOrderBroadcast, **instance** *tob*.

Uses:
ReliableBroadcast, **instance** *rb*;
Consensus (multiple instances).

upon event ⟨ *tob*, *Init* ⟩ **do**
unordered := ∅;
delivered := ∅;
round := 1;
wait := FALSE;

upon event ⟨ *tob*, *Broadcast* | m ⟩ **do**
trigger ⟨ *rb*, *Broadcast* | m ⟩;

upon event ⟨ *rb*, *Deliver* | p, m ⟩ **do**
if $m \notin$ *delivered* **then** → no duplication
unordered := *unordered* ∪ {(p, m)}; ⇐ total order

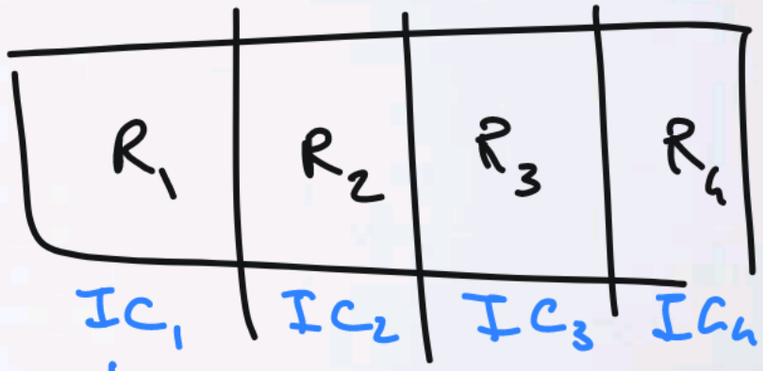
upon *unordered* ≠ ∅ ∧ *wait* = FALSE **do**
wait := TRUE;
Initialize a new instance *c.round* of consensus;
trigger ⟨ *c.round*, *Propose* | *unordered* ⟩;

messages to send

upon event ⟨ *c.r*, *Decide* | *decided* ⟩ **such that** r = *round* **do**
forall $(s, m) \in$ *sort*(*decided*) **do** // by the order in t
trigger ⟨ *tob*, *Deliver* | s, m ⟩; // in the resulting sorted list
delivered := *delivered* ∪ *decided*;
unordered := *unordered* \ *decided*;
round := *round* + 1;
wait := FALSE;

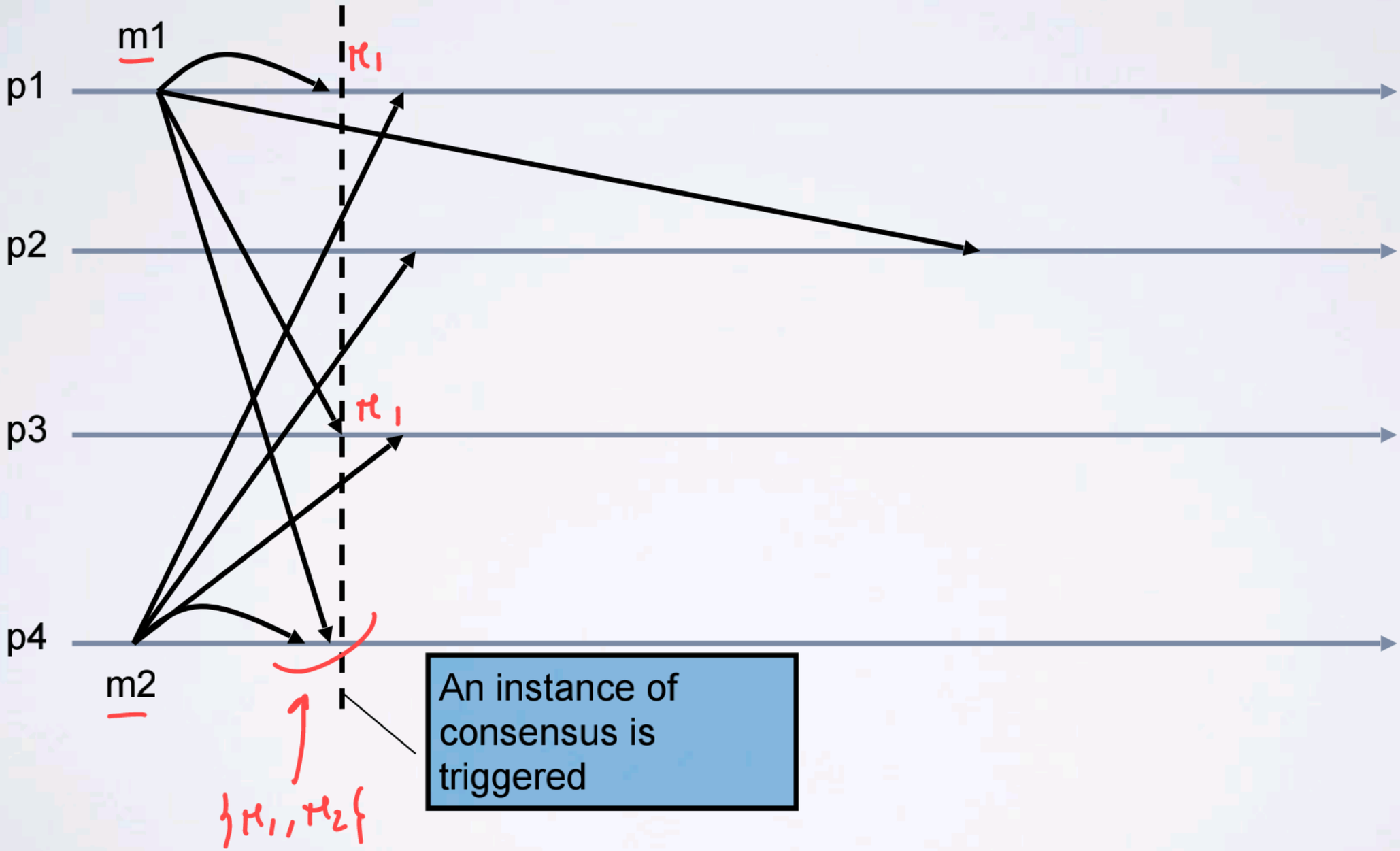
remove M_2 in the example →

P_1 × P_2
$\{M_1, M_2\}$ $\{M_2, M_3\}$
$\{M_2, M_3\}$ =

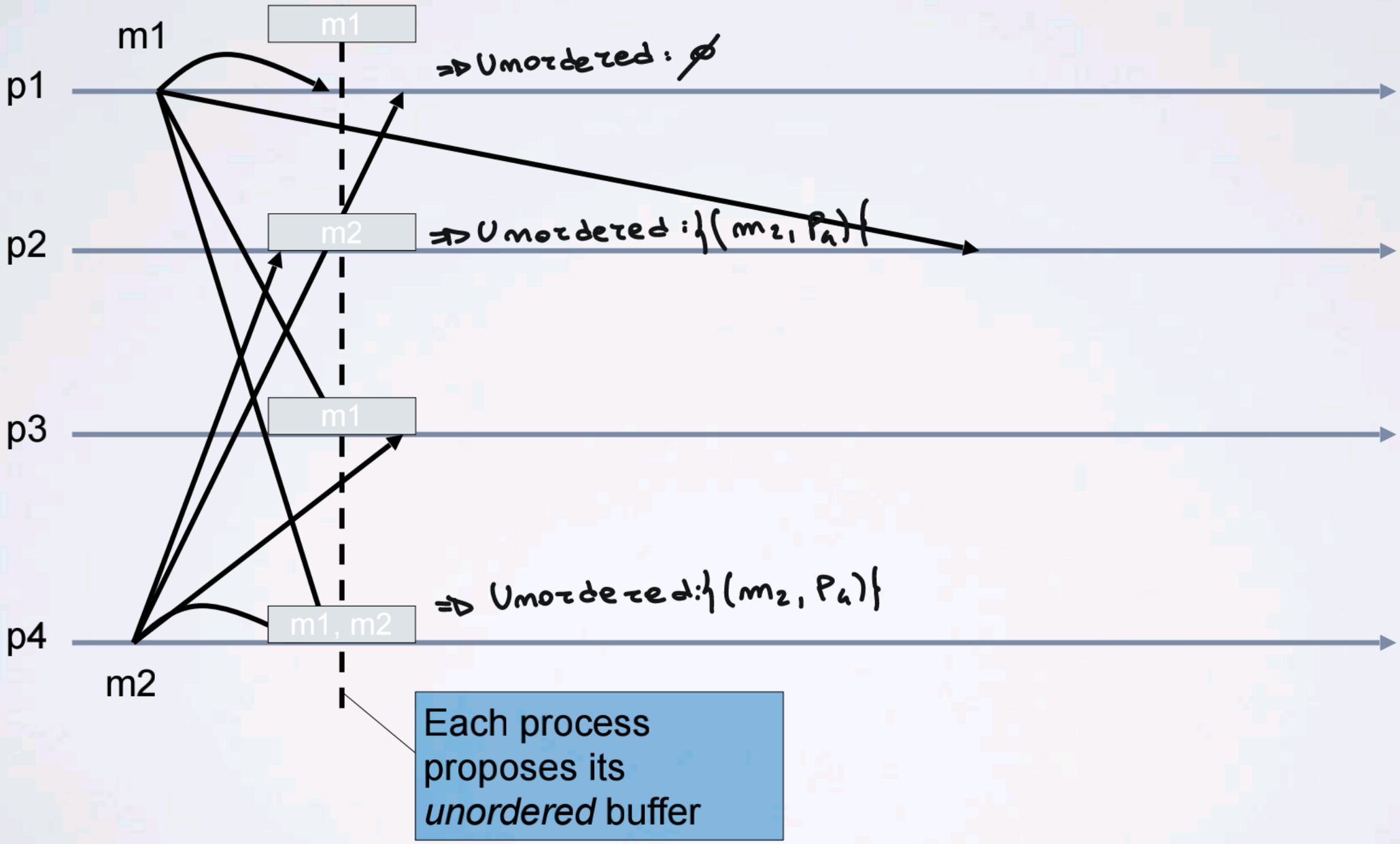


istance of consensous

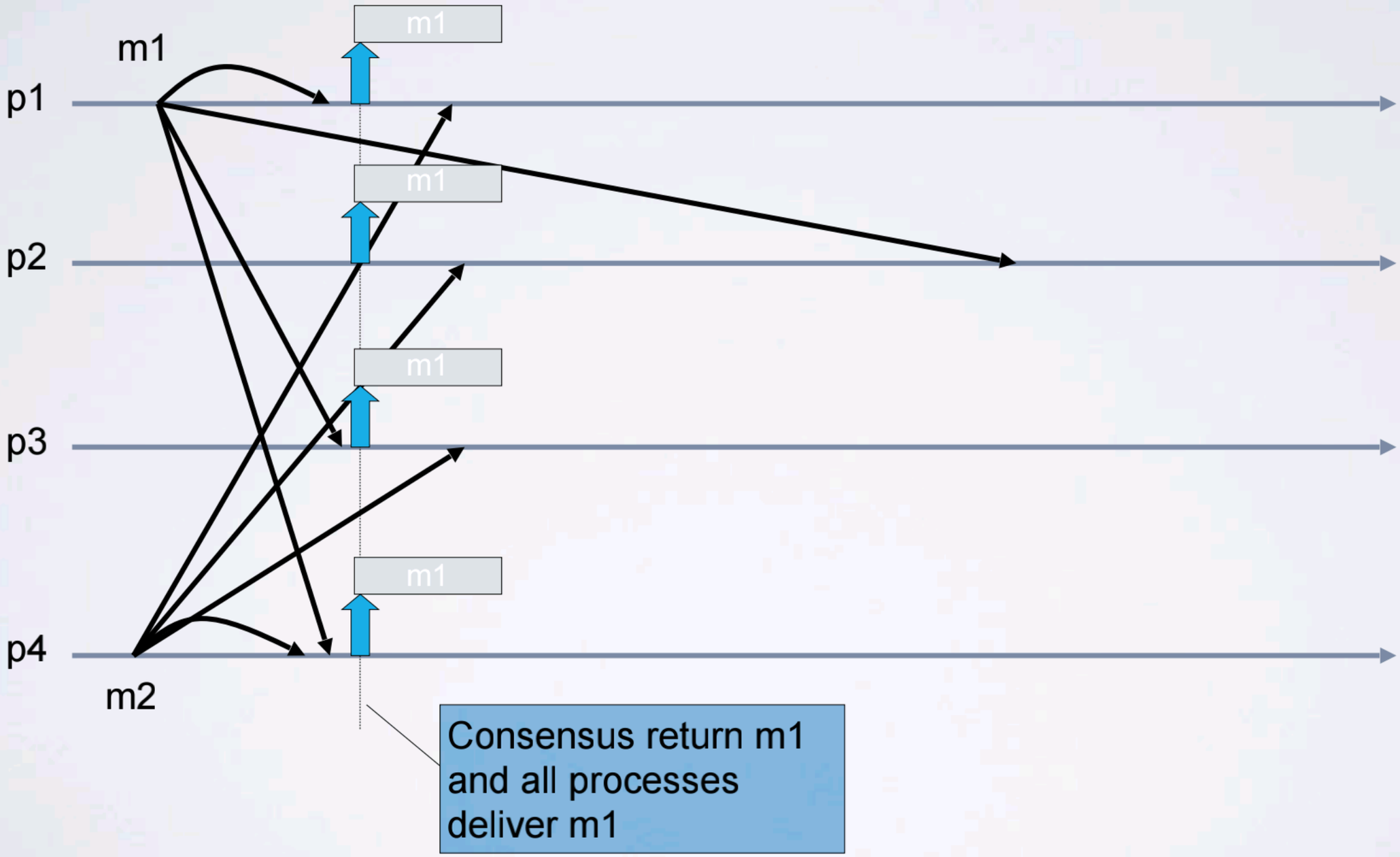
EXAMPLE



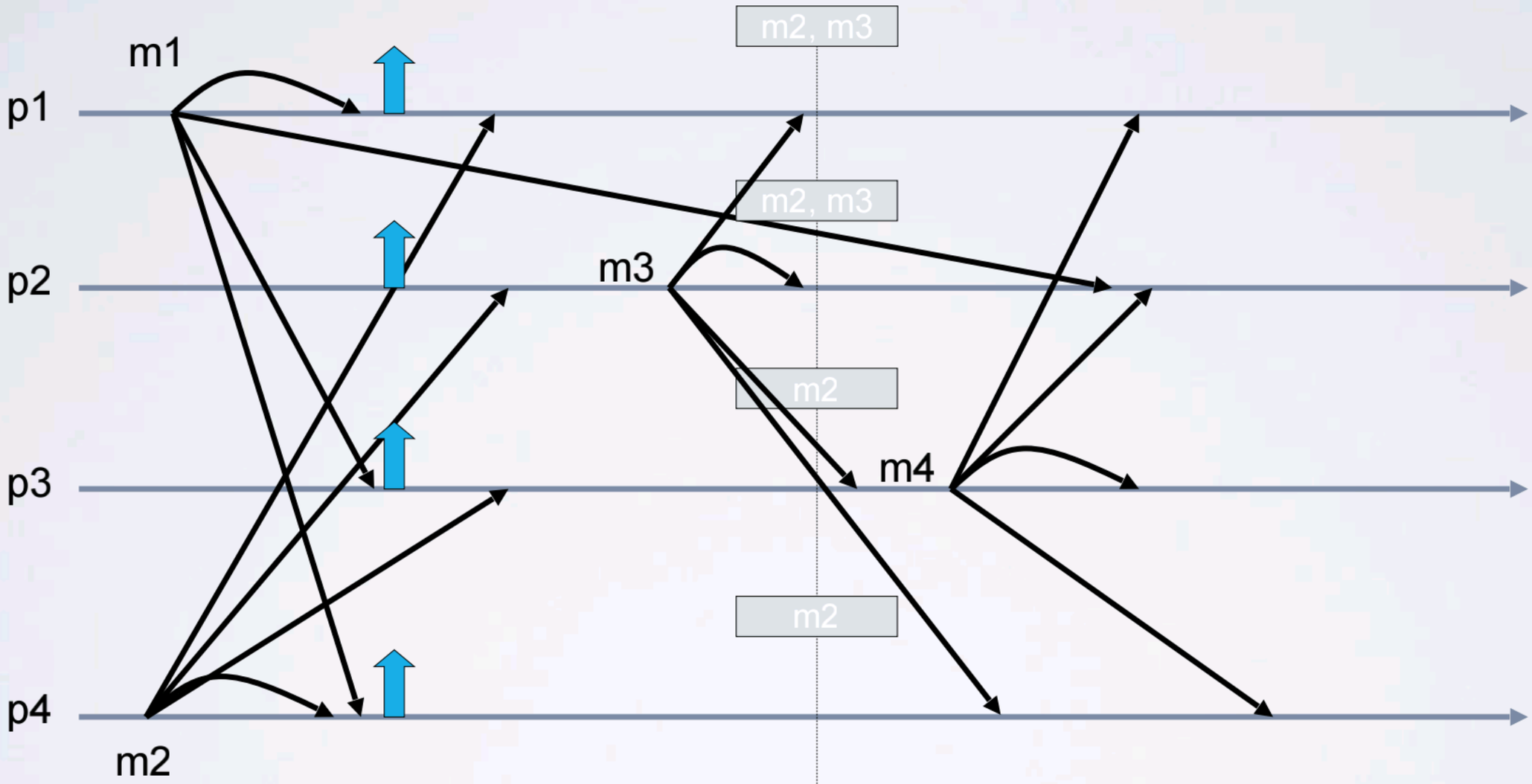
EXAMPLE


m1
m1
p1
⇒ Unordered : ∅
m2
p2
⇒ Unordered : {(m2, P4)}
m1
p3
m1, m2
p4
⇒ Unordered : {(m2, P4)}
m2
Each process proposes its *unordered* buffer

EXAMPLE

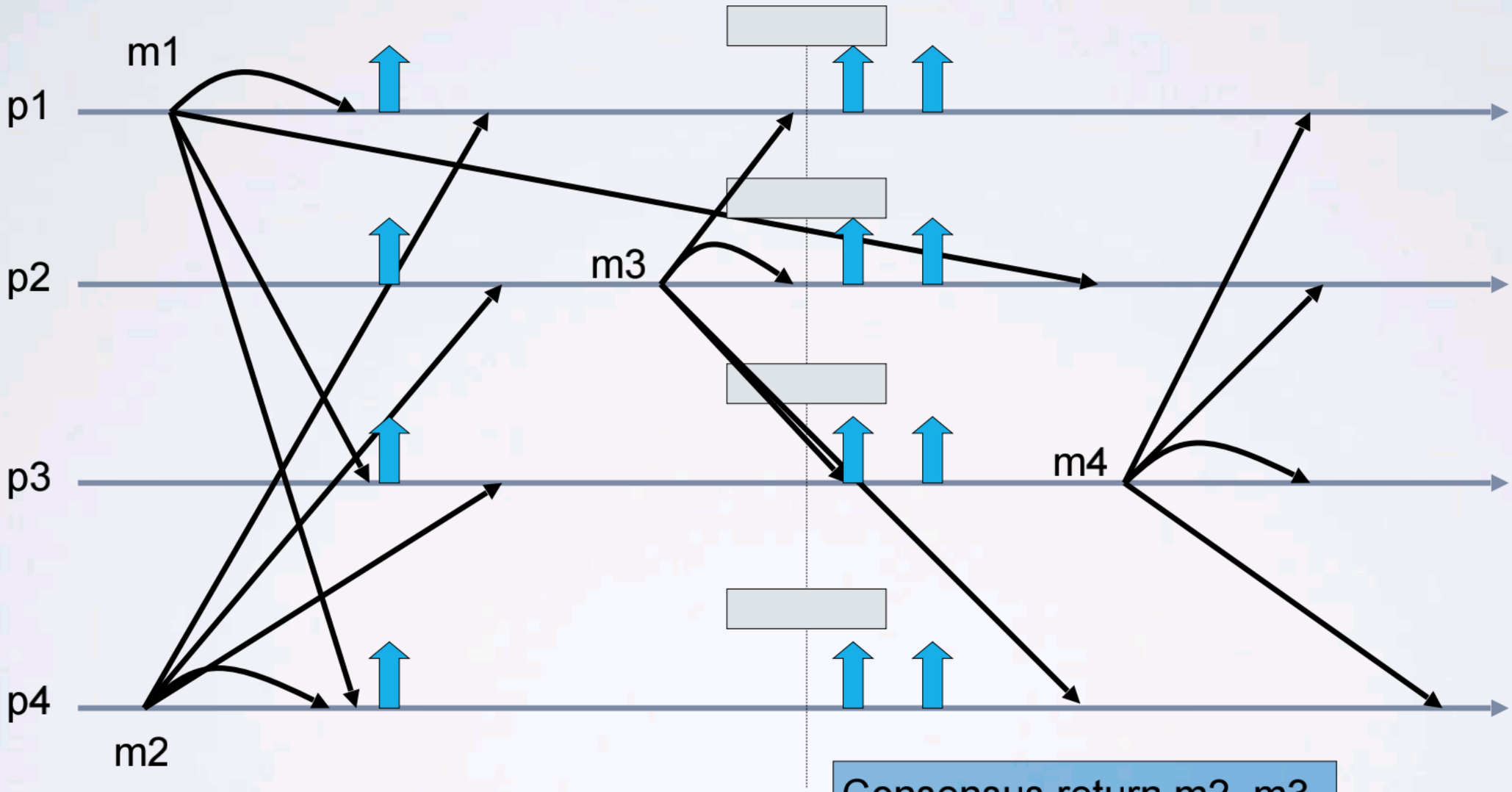

m1
m1
p1
m1
p2
m1
p3
m1
p4
m2
Consensus return m1
and all processes
deliver m1

EXAMPLE



An instance of consensus is triggered	Each process proposes its *unordered* buffer

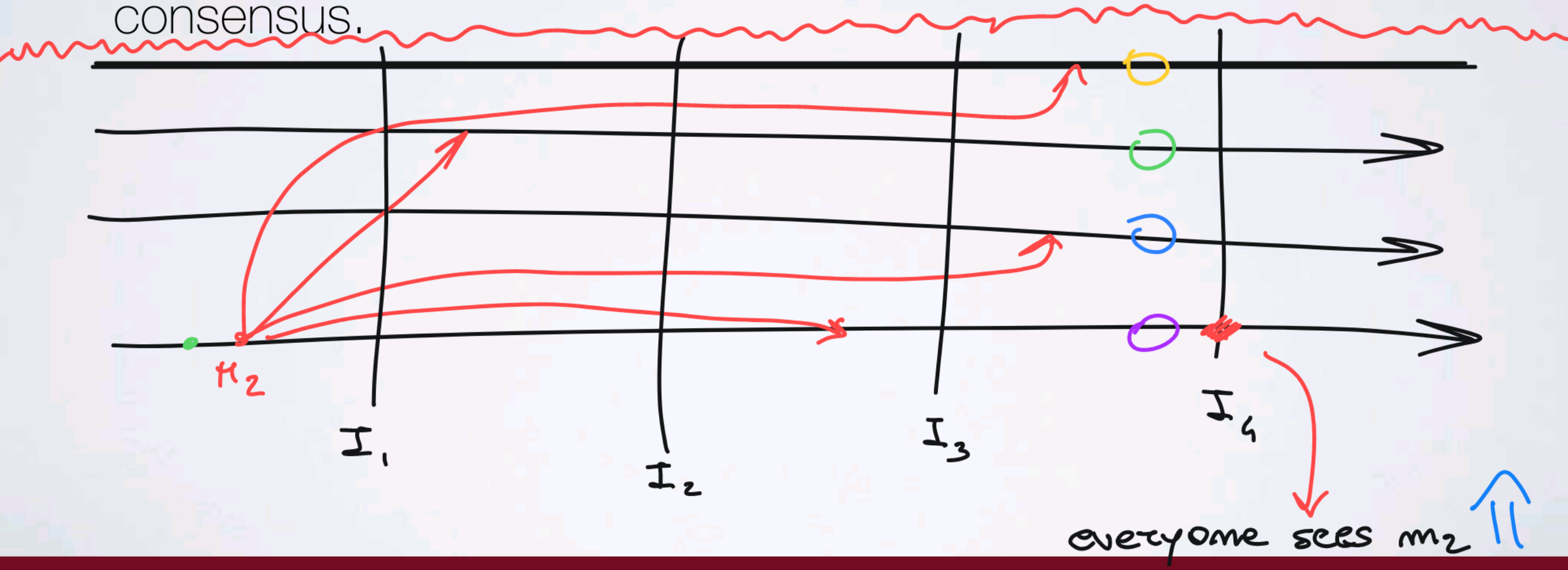
EXAMPLE


m1
p1
p2
m3
p3
m4
p4
m2
Consensus return m2, m3 and all processes deliver m2 and then m3

FORMAL PROOF

No creation: There are two points at which a message may be created:

- <rb, deliver, p, m>. -> impossible by no creation of rb
- <c,r, decided, messages> -> impossible by validity of consensus.



FORMAL PROOF

No duplication: Suppose by contradiction you deliver message m twice.

Then you deliver it because it is in the decided of two different instances of consensus (r, and r' with r' > r).

Why?

But at round r you put m in delivered (and also the others), and you remove it from unordered. Now if at r' you see m in decided then someone, let it p, triggered <r', Propose, unordered> with unordered containing m. **WHY?**

But p at the end of round r, p puts m in delivered and removes it from unordered. So it means it had to add it between the end of round r and the beginning of round r'. **But unordered is guarded by delivered!**

FORMAL PROOF

Agreement: Suppose a correct process p0 delivers m at round r.

Then m is in decided of the consensus of round r.

If m is not in the decided of the consensus of round r of a correct process p1, which **property of consensus are you violating?**

So m has to be in the decided of any correct process at round r. -> agreement.

FORMAL PROOF

Total order: We have to divide the proof in two cases.

1) m and m' are delivered in the same round -> they must be delivered in the same round on each pair of correct process (WHY?)-> the order is fixed by the deterministic function order. Thus m and m' are delivered in the same order on each correct process.

1) m and m' are delivered in two different rounds r and r' on correct process p -> The only thing that we have to show is that each other correct delivers them at the exact same round r and r' -> but this is direct from the agreement of the consensus.

QUESTIONS

- What happens if I have Beb instead of RB? Is the algorithm still correct or not?
- Why I need a broadcast?

Module 6.2: Interface and properties of uniform total-order broadcast

Module:

Name: UniformTotalOrderBroadcast, **instance** *utob*.

Events:

Request: $\langle$ *utob*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *utob*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

UTOB1–UTOB3: Same as properties TOB1–TOB3 in regular total-order broadcast (Module 6.1).

UTOB4: *Uniform agreement:* If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.

UTOB5: *Uniform total order:* Let m_1 and m_2 be any two messages and suppose p and q are any two processes that deliver m_1 and m_2 (whether correct or faulty). If p delivers m_1 before m_2, then q delivers m_1 before m_2.

UNIFORM

Algorithm 6.1: Consensus-Based Total-Order Broadcast

Implements:
TotalOrderBroadcast, **instance** *tob*.

Uses:
ReliableBroadcast, **instance** *rb*;
Consensus (multiple instances).

upon event $\langle$ *tob*, *Init* $\rangle$ **do**
unordered := $\emptyset$;
delivered := $\emptyset$;
round := 1;
wait := FALSE;

upon event $\langle$ *tob*, *Broadcast* | m $\rangle$ **do**
trigger $\langle$ *rb*, *Broadcast* | m $\rangle$;

upon event $\langle$ *rb*, *Deliver* | p, m $\rangle$ **do**
if $m \notin$ *delivered* **then**
unordered := *unordered* $\cup \{(p, m)\}$;

upon *unordered* $\neq \emptyset \wedge$ *wait* = FALSE **do**
wait := TRUE;
Initialize a new instance *c.round* of consensus;
trigger $\langle$ *c.round*, *Propose* | *unordered* $\rangle$;

upon event $\langle$ *c.r*, *Decide* | *decided* $\rangle$ **such that** r = *round* **do**
forall $(s, m) \in$ *sort*(*decided*) **do** // by the order in t
trigger $\langle$ *tob*, *Deliver* | s, m $\rangle$; // in the resulting sorted list
delivered := *delivered* $\cup$ *decided*;
unordered := *unordered* $\setminus$ *decided*;
round := *round* + 1;
wait := FALSE;

EXERCISE

Consider the previous algorithm. Which TO specification does it satisfy?

It depends from the assumptions about Reliable Broadcast and Consensus…

Consensus / Reliable Broadcast	Uniform	Non Uniform
Uniform	?	?
Non Uniform	?	**?**

EXERCISE

Consider the previous algorithm. Which TO specification does it satisfy?

It depends from the assumptions about Reliable Broadcast and Consensus…

Consensus / Reliable Broadcast	Uniform	Non Uniform
Uniform	?	**?**
Non Uniform	?	Non-uniform TO

EXERCISE

Consider the previous algorithm. Which TO specification does it satisfy?

Consensus / Reliable Broadcast	Uniform	Non Uniform
Uniform	?	**Non-uniform TO**
Non Uniform	?	Non-uniform TO

EXERCISE

Consider the previous algorithm. Which TO specification does it satisfy?

You only deliver the

Decided things, however

The decided are the same

For each one (correct or not)!

Consensus / Reliable Broadcast	Uniform	Non Uniform
Uniform	Uniform TO	Non-uniform TO
Non Uniform	**?**	Non-uniform TO

EXERCISE

Consider the previous algorithm. Which TO specification does it satisfy?

You only deliver the Decided things, however The decided are the same For each one (correct or not)!

Consensus / Reliable Broadcast	Uniform	Non Uniform
Uniform	Uniform TO	Non-uniform TO
Non Uniform	**Uniform TO**	Non-uniform TO

EXERCISE

You have a total order broadcast algorithm A, is it also a causal broadcast? Or a FIFO?

EXERCISE

You have a total order broadcast algorithm A, is it also a causal broadcast? Or a FIFO?

NO! Take one algorithm that uses as order function, a function that on purpose uses a reverse of causal and fifo order. It is still total!

EXERCISE

**Take the total order algorithm that we have seen in the lecture and modify it to implement a total order that is also causal.