31|10|23

Dependable Distributed Systems Master of Science in Engineering in Computer Science

AA 2023/2024

LECTURE 13: TOTAL ORDER BROADCAST

System model

Static set of processes $\Pi = \{p_1 \dots p_n\}$

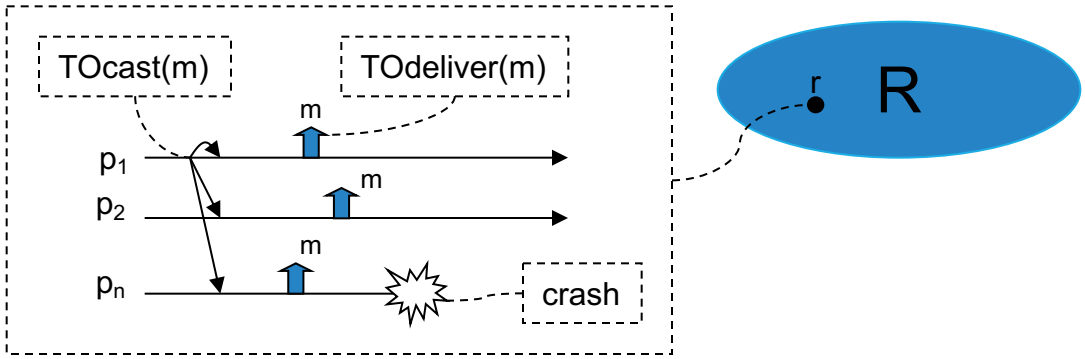
Message passing over perfect channels

- i.e., message exchanged between correct processes are reliably delivered

Asynchronous

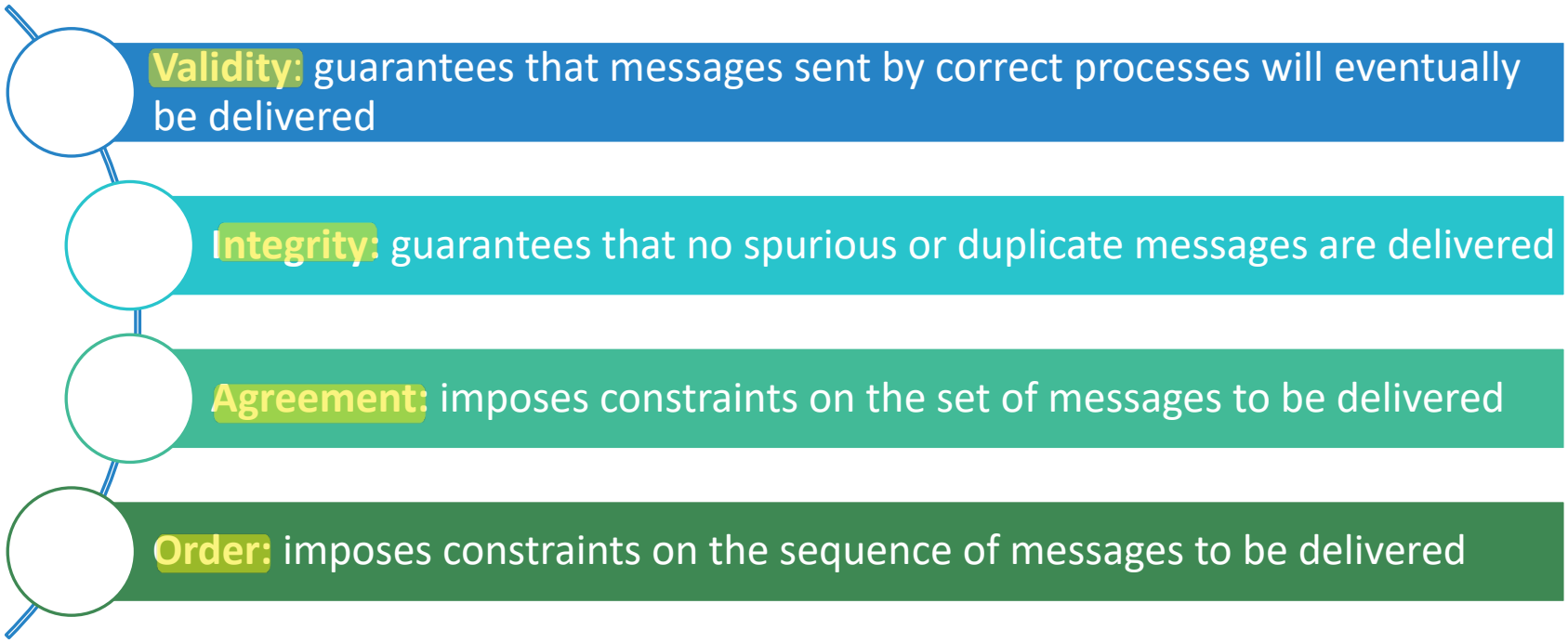
Crash fault model for processes

We characterize the system in terms of its possible runs R



TO specifications

Total order specifications are usually composed by four properties



improved version of RB

↑

TO specifications

Total Order Broadcast = TO(V,I,A,O)

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

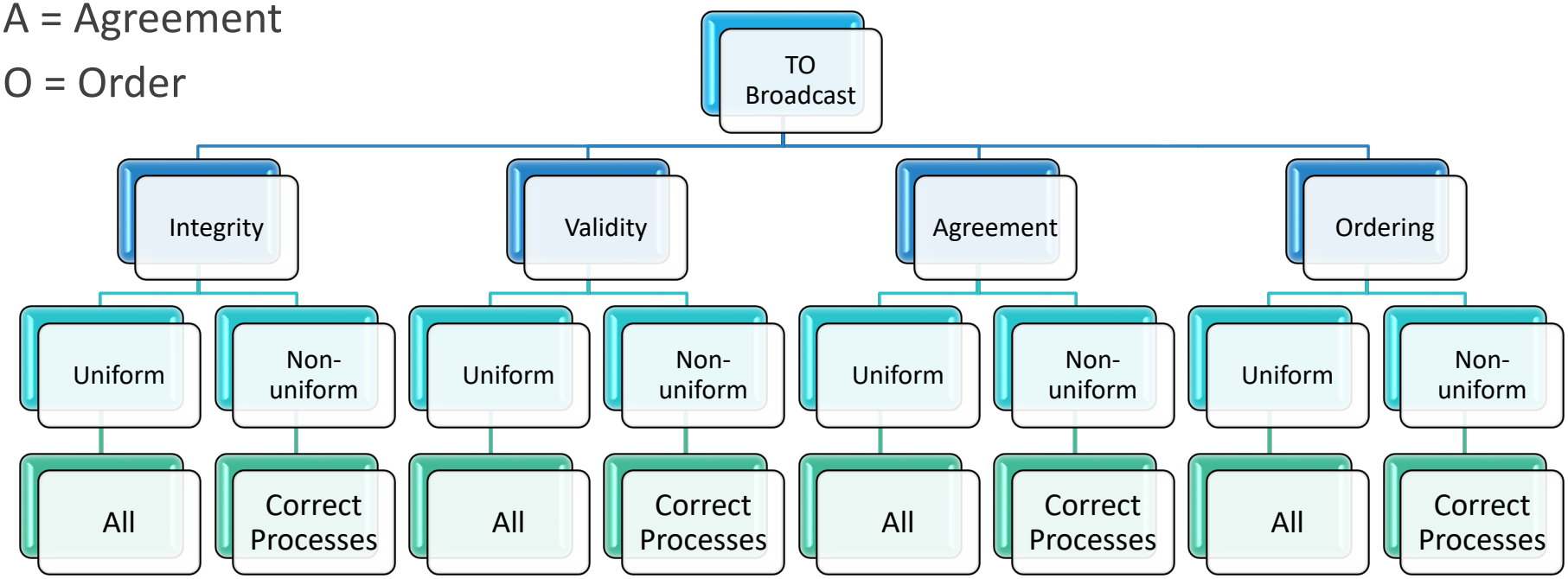
Distinct specifications arise from distinct formulations of each property

- uniform vs non-uniform
- A uniform property imposes restrictions on the behavior of (at least) correct processes

TO specifications

Total Order Broadcast = TO(V,I,A,O)

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



TO Specifications

only crash, not recovery

Crash failure + Perfect channels ⇒ PP2PL

m can arrive or not (if fail)

NO UNIFORM VALIDITY

- **NUV**: if a correct process TOCAST a message m then some *correct* process will eventually deliver m

UNIFORM INTEGRITY

- **UI**: For any message m, every process p delivers m at most once and only if m was previously TOCAST by some (correct or not) process

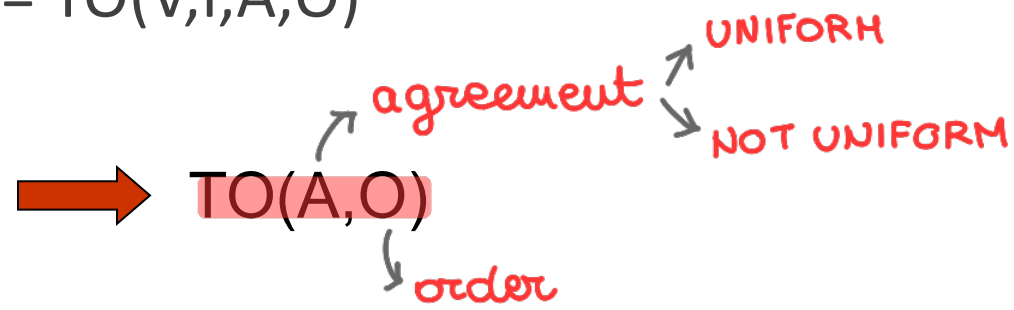
TO specifications

Total Order Broadcast = TO(V,I,A,O)

- V = Validity
- I = Integrity

NUV
UI

- 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

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

THE SET OF M DELIVERED BY THE FAULTY IS A SUBSET OF THE M DELIVERED BY THE CORRECT

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. $M_p - M \neq \varnothing$

● Agreement: UA

THE SET OF M DELIVERED BY THE **FAULTY** IS A SUBSET OF THE M DELIVERED BY THE **CORRECT**

● Agreement: NUA

Mp can be s.t. Mp - M ≠ 0

The Agreement property

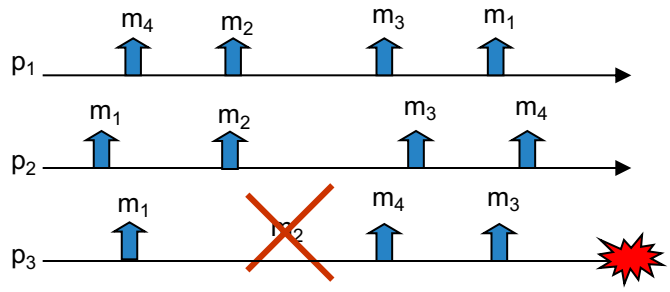
UNIFORM AGREEMENT (UA)

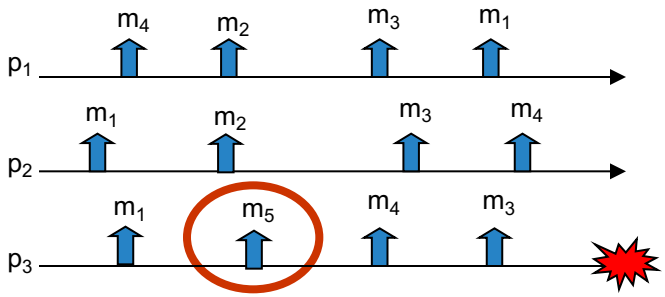
If a process (correct or not) TODelivers a message m, then all correct processes will eventually TODeliver m

NON-UNIFORM AGREEMENT (NUA)

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

UA





NUA

The Ordering Property

STRONG UNIFORM TOTAL ORDER (SUTO)

If some process TODelivers some message m before message m', then a process TODelivers m' only after it has TODelivered m.

- same order
- same prefix of the set of delivered messages
- after an omission, disjoint sets of delivered messages

WEAK UNIFORM TOTAL ORDER (WUTO)

If process p and process q both TODdeliver messages m and m', then p TODelivers m before m' if and only if q TODdelivers m before m'.

- no restrictions on the set of delivered messages

The Order Property

STRONG UNIFORM TOTAL ORDER (SUTO)

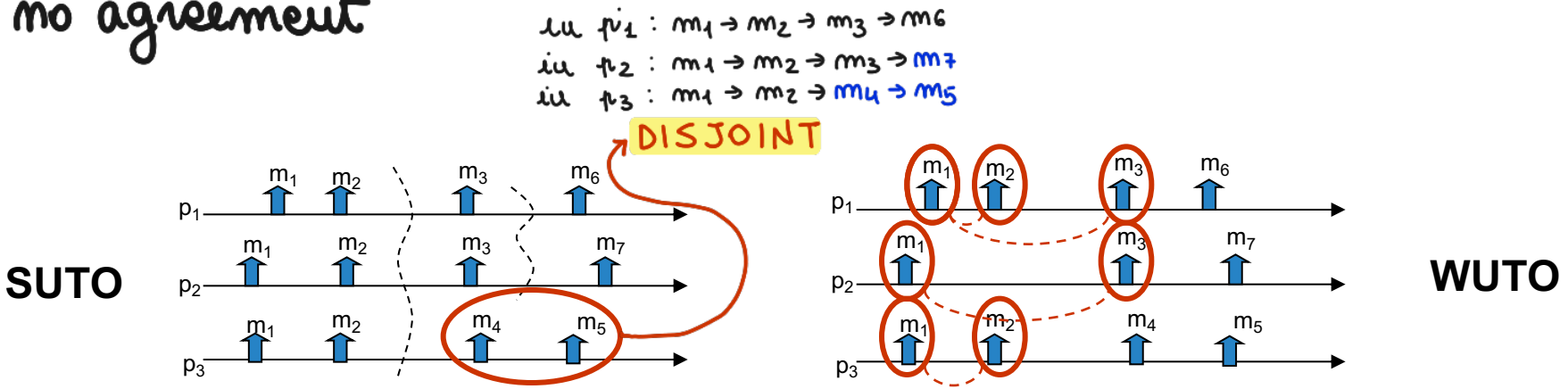
If some process TODelivers some message m before message m', then a process TODelivers m' only after it has TODelivered m.

↓

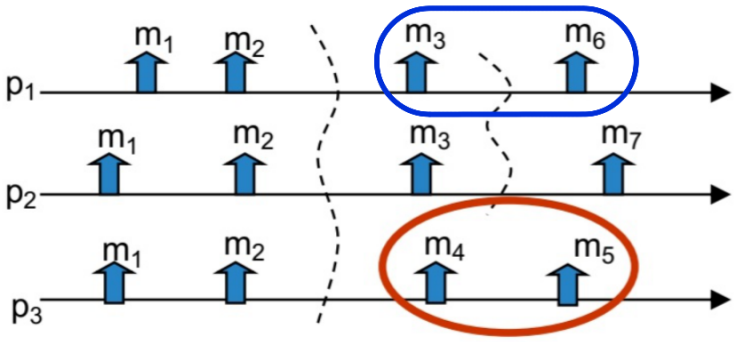
no agreement

WEAK UNIFORM TOTAL ORDER (WUTO)

If process p and process q both TODdeliver messages m and m', then p TODelivers m before m' if and only if q TODdelivers m before m'.



SUTO



N.B.

p_1: we have $m_1 \rightarrow m_2$

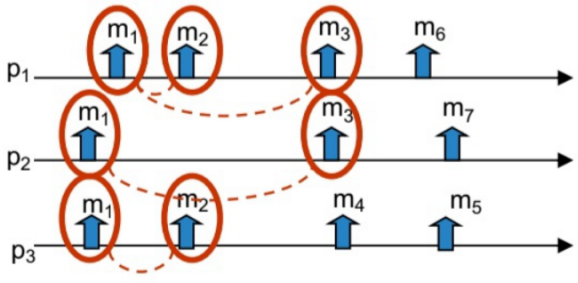
↓

every process have to respect this

p_1: we have m_3; the order imply that m_6 could be delivered only after the delivery of m_3

↓

the other correct processes either deliver m_3 and then m_6, or doesn't deliver m_6 because $m_3 \rightarrow m_6$



WUTO

$m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow m_6$

$m_1 \longrightarrow m_3$

$m_1 \rightarrow m_2$

Take two processes that deliver the same pair of m, I just want that the sequence is the same

$m_1 \rightarrow m_2$

but I can deliver $m_1 \rightarrow m_3$ also if I have not deliver m_2

The Order Property

SUTO and WUTO are uniform but they both have a non-uniform counterpart

STRONG NON-UNIFORM TOTAL ORDER (SNUTO)

If some correct process TODelivers some message m before message m', then a correct process TODelivers m' only after it has TODelivered m.

↓

the FAULTY can do everything he wants

↓

no ordering respected

↓

CORRECT deliver the same SEQUENCE

WEAK NON-UNIFORM TOTAL ORDER (WNUTO)

If correct processes p and q both TODeliver messages m and m', then p TODelivers m before m' if and only if q TODelivers m before m'.

↓

the FAULTY can do everything he wants

↓

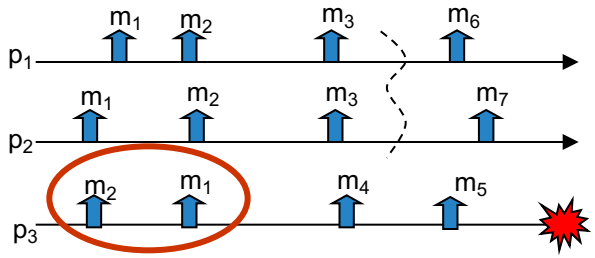
CORRECT p. may have the same prefix

The Order property (2)

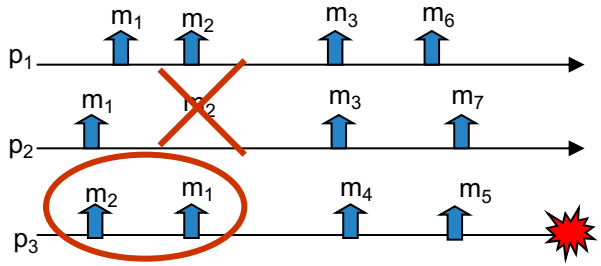
SUTO ⇒ WUTO

SNUTO ⇒ WNUTO

SNUTO



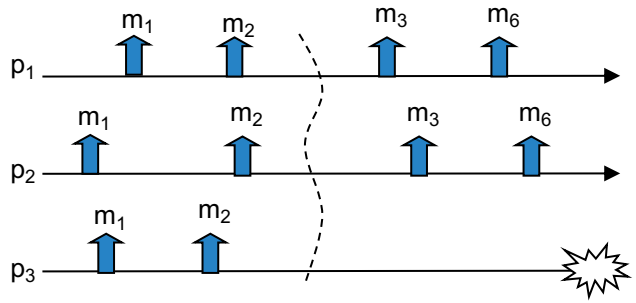
WNUTO



TO specifications

UA: forces the faulty p. to deliver a subset of the correct

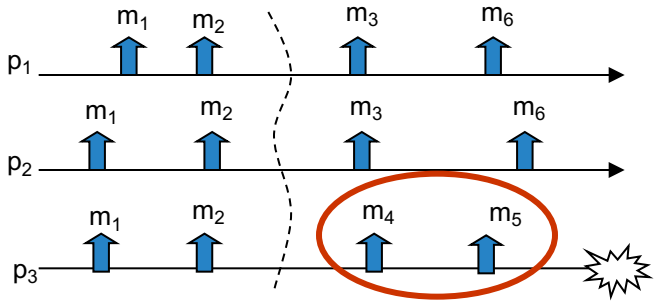
SUTO: impose that the correct p. deliver the same sequence

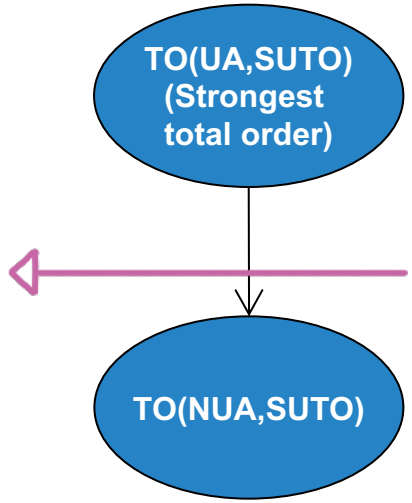


TO(UA,SUTO)

- The strongest TO spec.

TO(NUA,SUTO)



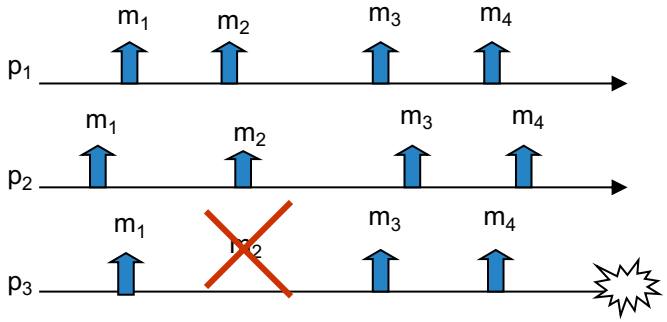


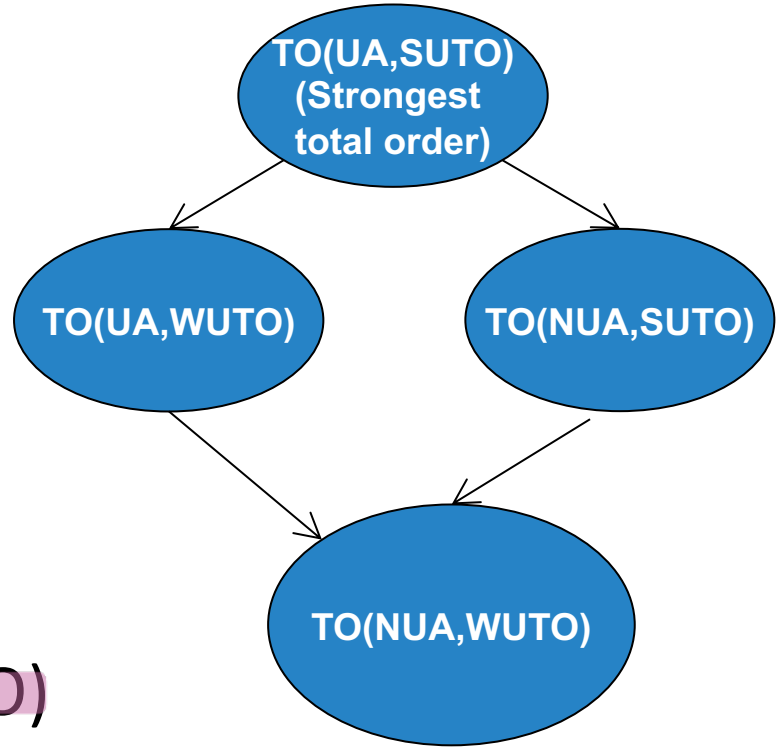
NUA: faulty are delivering something that the correct are not delivering

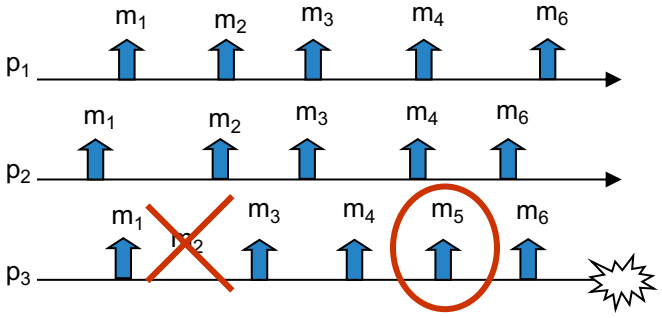
SUTO: impose that the correct p. deliver the same sequence, same set

TO specifications (2)

TO(UA,WUTO)







TO(NUA,WUTO)

TO(UA, WUTO)

after the omission the set cannot diverge

UA: forces the faulty p_i to deliver a subset of the correct

↓

I can tolerate just 1 MISSING EVENT

WUTO: correct deliver the same set in the same sequence

TO(NUA, WUTO)

(1)

NUA: faulty p_i deliver same prefix, than an omission, than I have messages deliver by the others

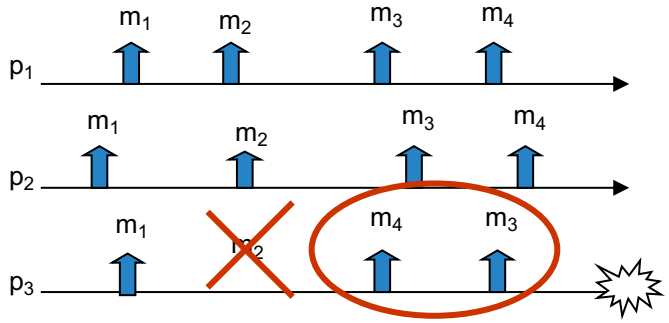
↓

no subset of the correct, no strong ordering

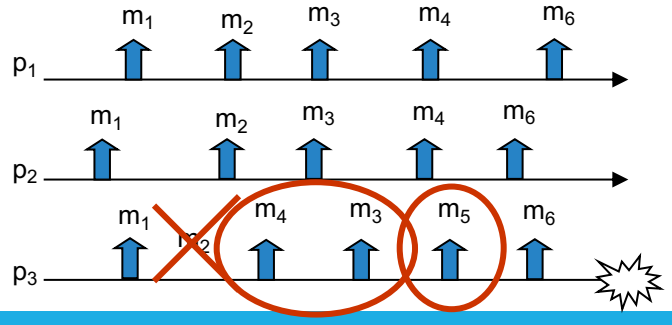
WUTO: correct deliver the same set in the same sequence

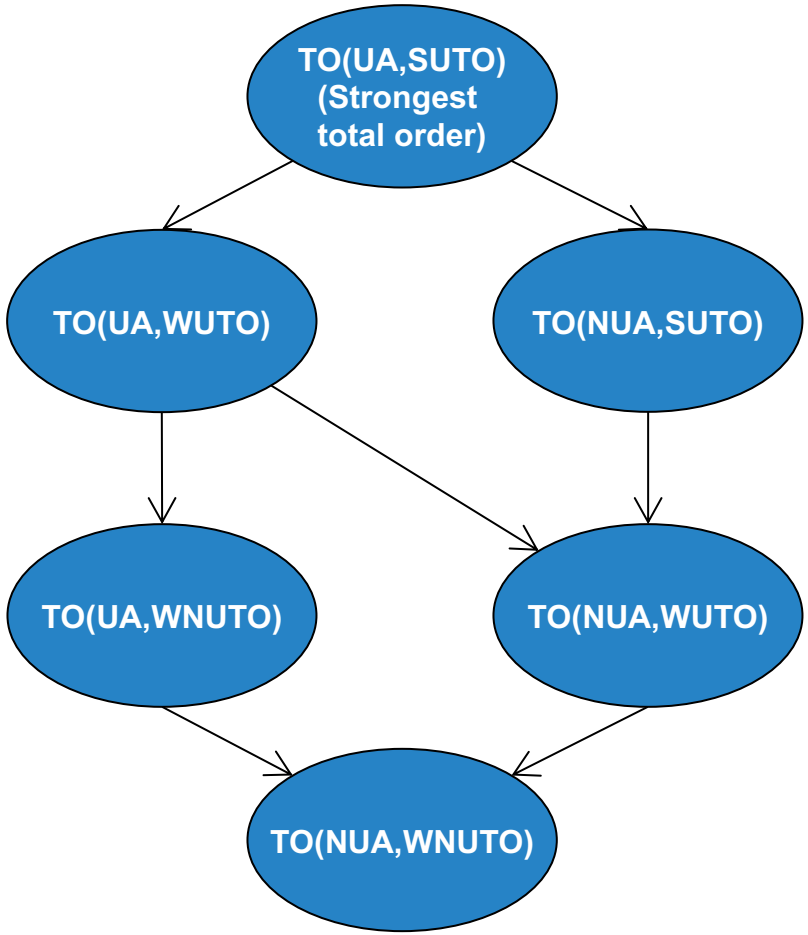
TO specifications (3)

TO(UA,WNUTO)



TO(NUA,WNUTO)





TO(UA, WNUTO)

UA: forces the faulty p. to deliver a subset of the correct
↓
I can tolerate just 1 MISSING EVENT

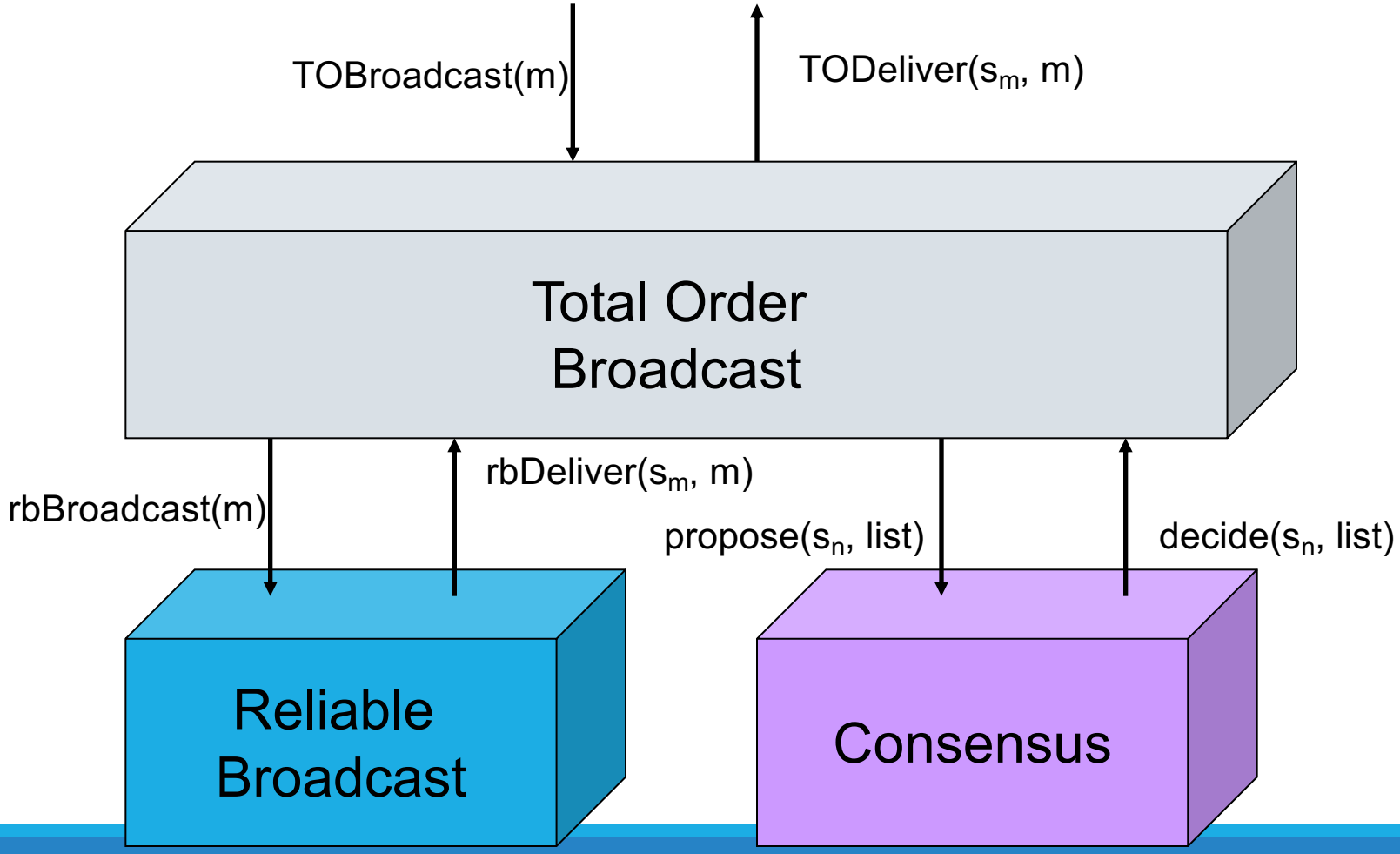
WNUTO: ordering is not strong, faulty can deliver in whenever order it wants

TO(NUA, WNUTO)

NUA: faulty p. can deliver something that the correct one not delivered, the order cannot be respected

WNUTO: ordering is not strong, faulty can deliver in whenever order it wants

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 $\langle$ *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 $\rangle$;

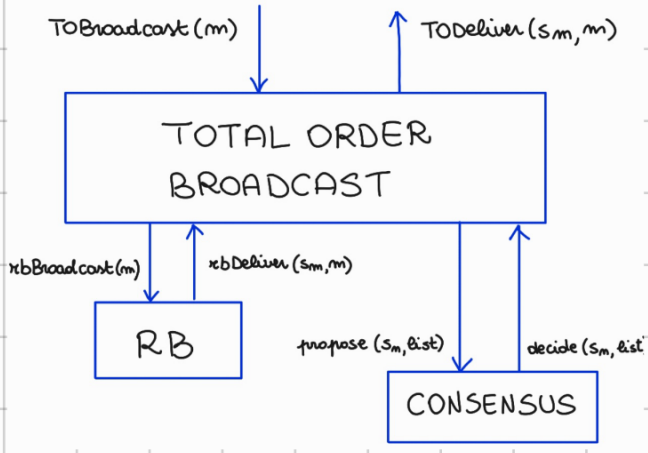
upon event $\langle$ *rb*, *Deliver* | p, m $\rangle$ **do**
if $m \notin$ *delivered* **then**
unordered := *unordered* $\cup \{(p, m)\}$;

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 the resulting sorted list
trigger $\langle$ *tob*, *Deliver* | s, m $\rangle$;
delivered := *delivered* $\cup$ *decided*;
unordered := *unordered* $\setminus$ *decided*;
round := *round* + 1;
wait := FALSE;

13: TOTAL

- **Validity**: guarantees that messages sent by correct processes will eventually be delivered
- **Integrity**: guarantees that no spurious or duplicate messages are delivered
- **Agreement**: imposes constraints on the set of messages to be delivered
- **Order**: imposes constraints on the sequence of messages to be delivered



Algorithm 6.1: Consensus-Based Total-Order Broadcast

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

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

Keep track of the m in the system that are not yet delivered; need to be ordered

upon event ⟨ *tob*, Init ⟩ **do**
unordered := ∅;
delivered := ∅; → keep track of what I already delivered
round := 1; → for counting how many consensus I am running
wait := FALSE; → serialize the execution of the consensus and to avoid more than one instances of consensus

for repetition

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

upon event ⟨ *rb*, Deliver | *p*, *m* ⟩ **do** (*p*: source/sender)
if *m* ∉ *delivered* **then**
unordered := *unordered* ∪ {(*p*, *m*)};

upon *unordered* ≠ ∅ ∧ *wait* = FALSE **do** (you are not going anything)
wait := TRUE; → new consensus instance
Initialize a new instance *c.round* of consensus;
trigger ⟨ *c.round*, Propose | *unordered* ⟩; (buffer is a set of pair (sender, msg))

for a specific consensus round

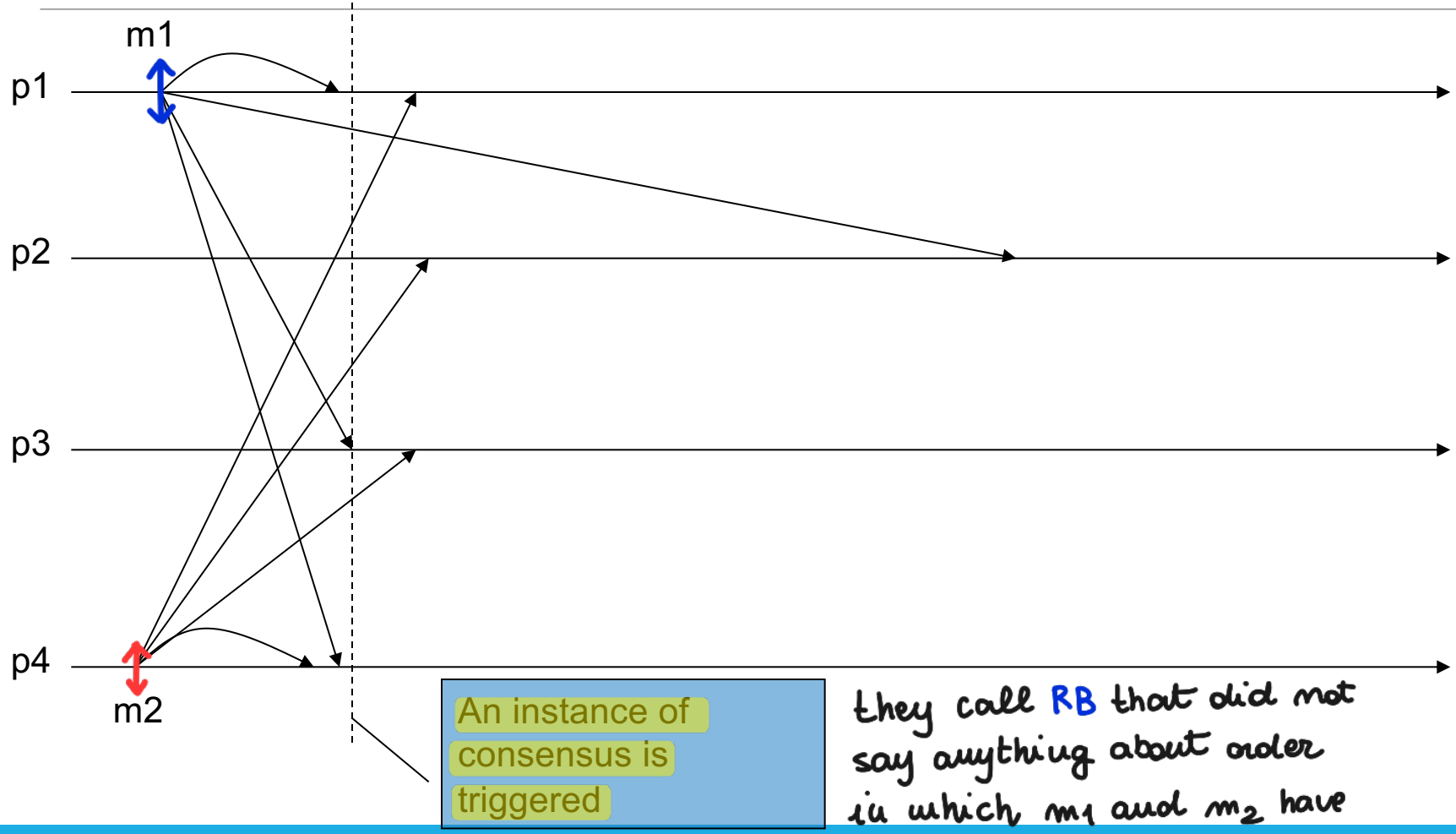
upon event ⟨ *c.r*, Decide | *decided* ⟩ **such that** *r* = *round* **do**
forall (*s*, *m*) ∈ sort(*decided*) **do** (*s*: source; is a list and need to be sorted)
trigger ⟨ *tob*, Deliver | *s*, *m* ⟩;
delivered := *delivered* ∪ *decided*;
unordered := *unordered* \ *decided*;
round := *round* + 1;
wait := FALSE;

1:03:29

Example

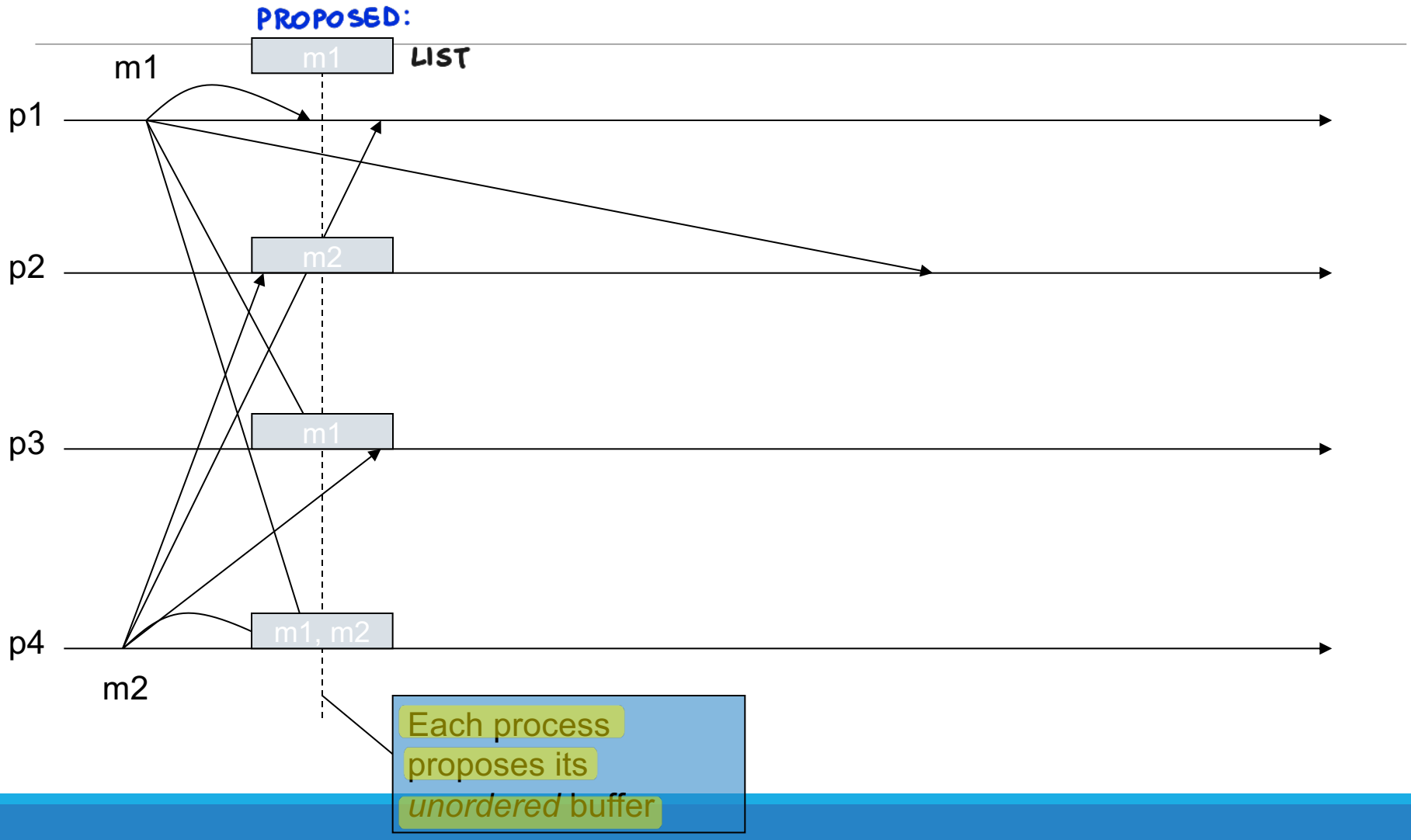
4 processes

p_1, p_2 } broadcast concurrently m_1, m_2

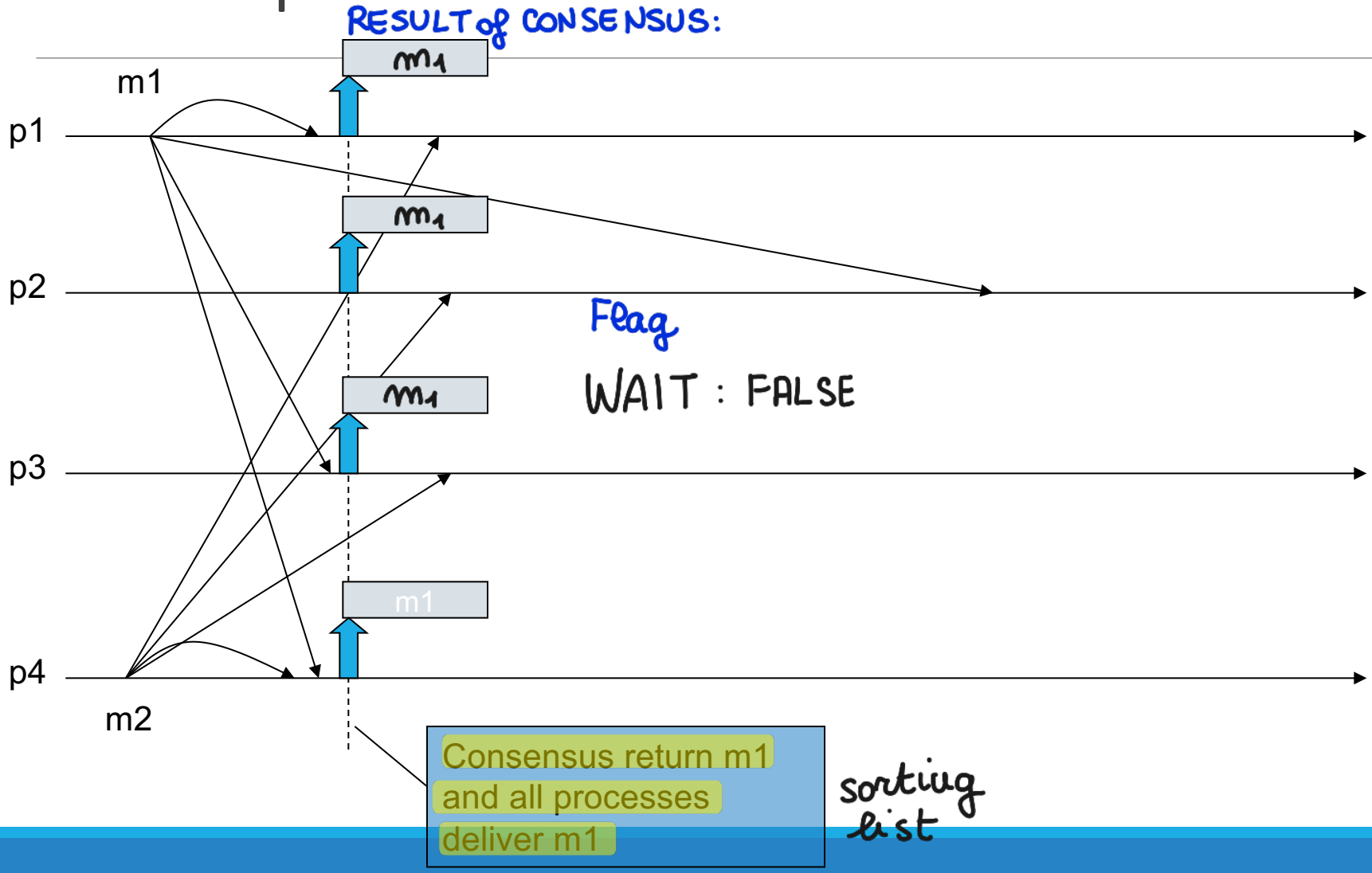


they call RB that did not say anything about order in which m_1 and m_2 have to be delivered

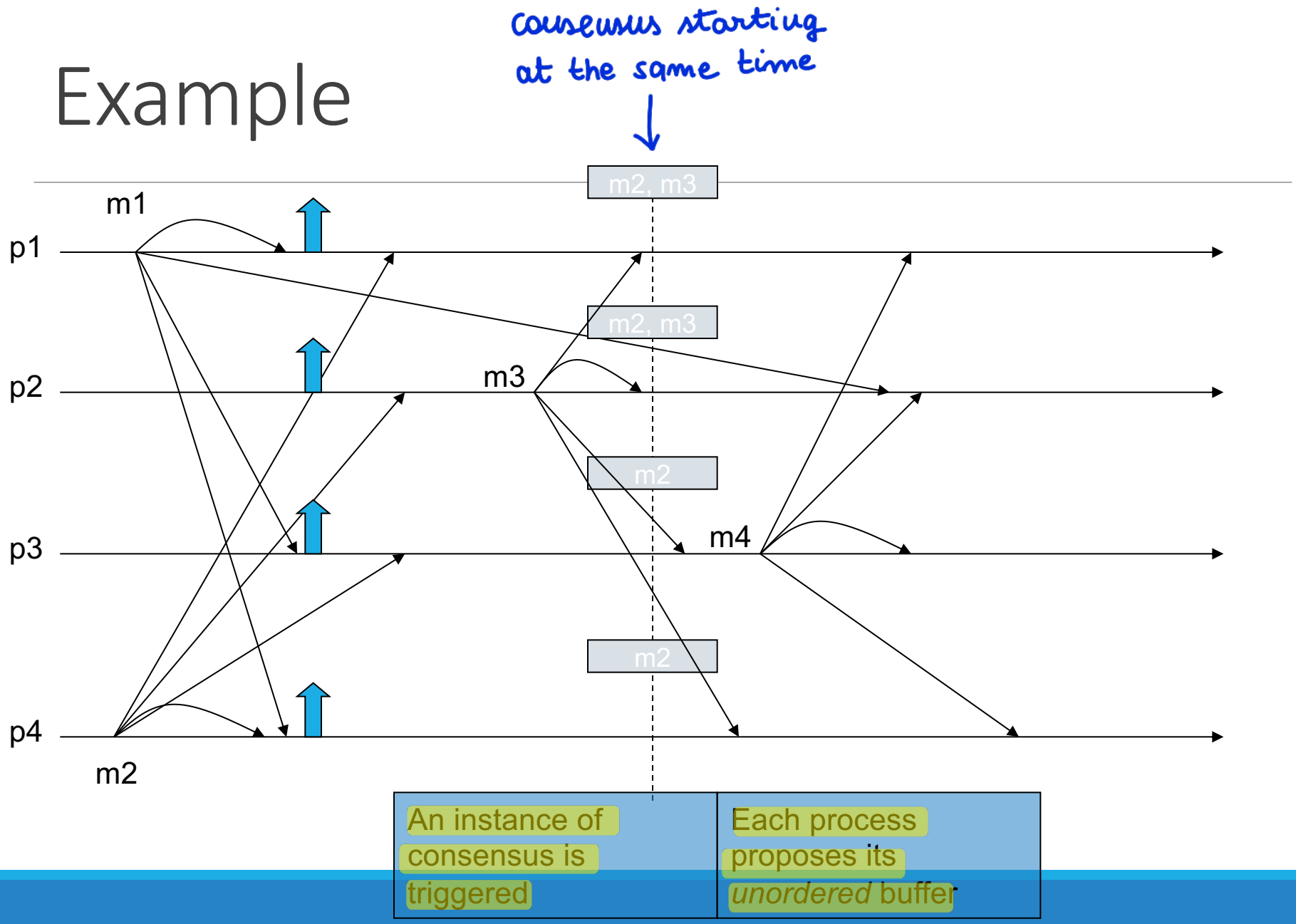
Example


PROPOSED:
LIST
m1
p1
p2
p3
p4
m2
m1
m1, m2
m2
Each process proposes its *unordered* buffer

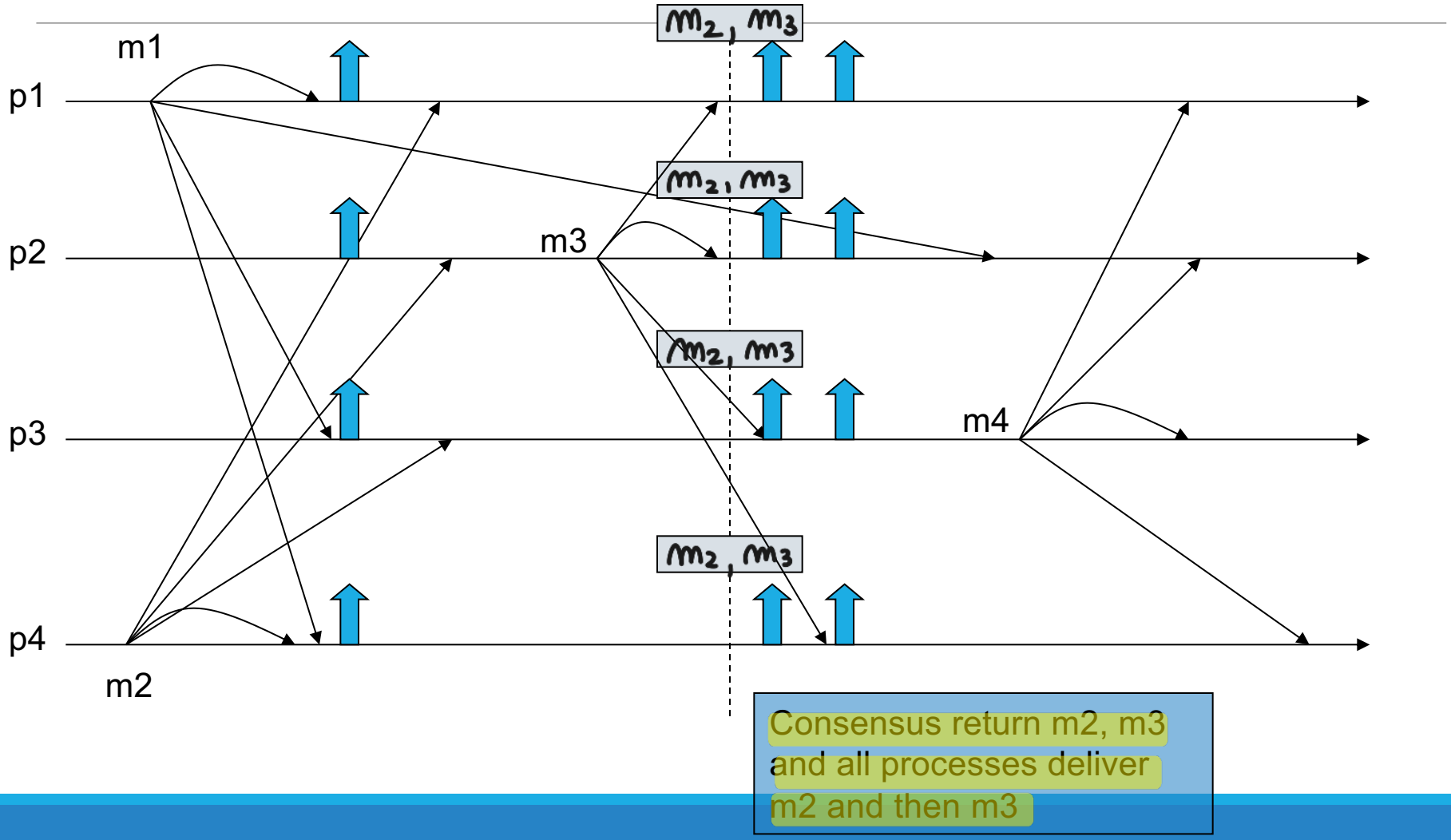
Example


RESULT of CONSENSUS:
m1
m1
p1
m1
p2
Flag
WAIT : FALSE
m1
p3
m1
p4
m2
Consensus return m1
and all processes
deliver m1
sorting list

Example


consensus starting at the same time
m2, m3
m1
p1
m2, m3
p2
m3
m2
p3
m4
m2
p4
m2
An instance of consensus is triggered
Each process proposes its *unordered* buffer

Example


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

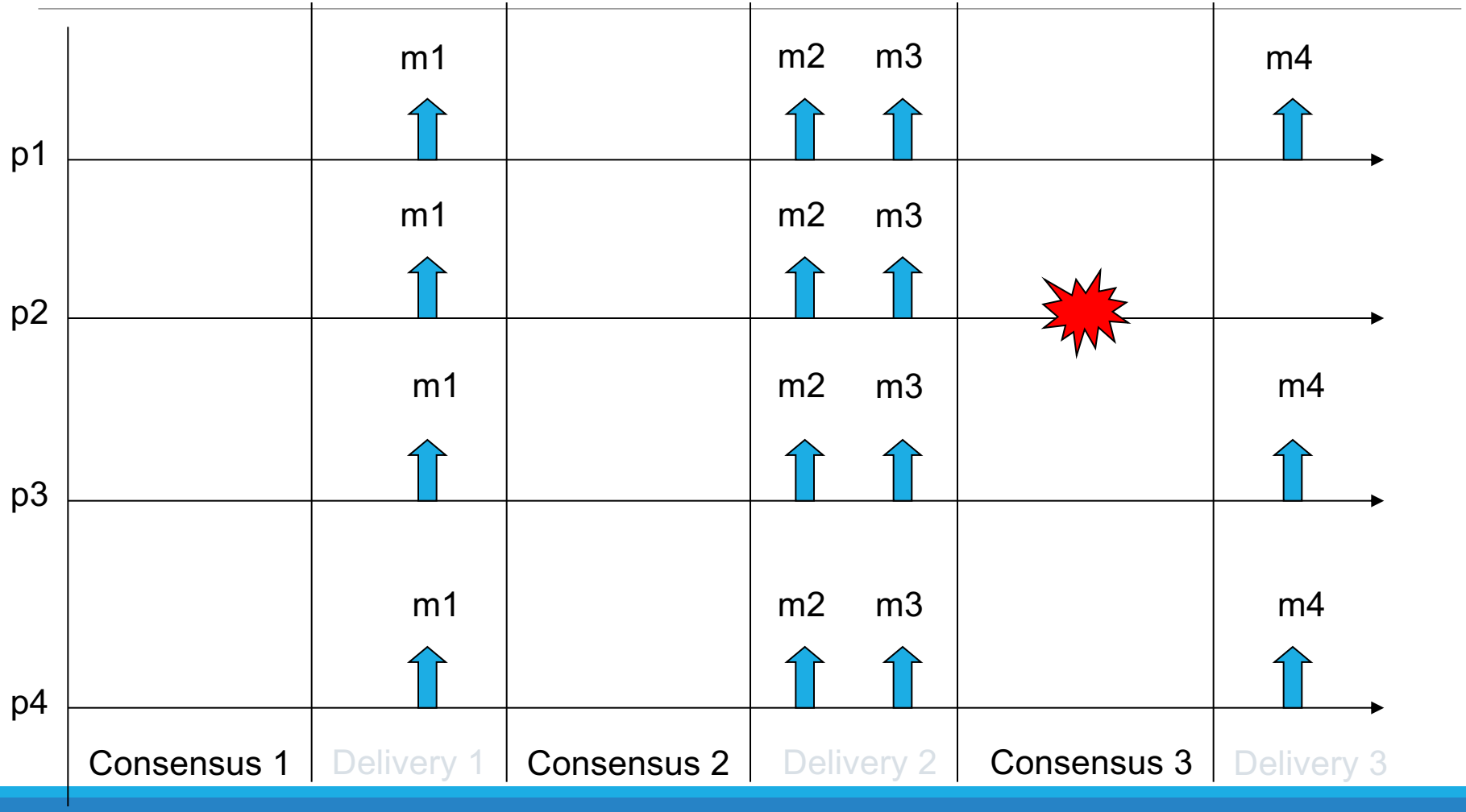
Exercise

Which TO specification is satisfied by this algorithm?

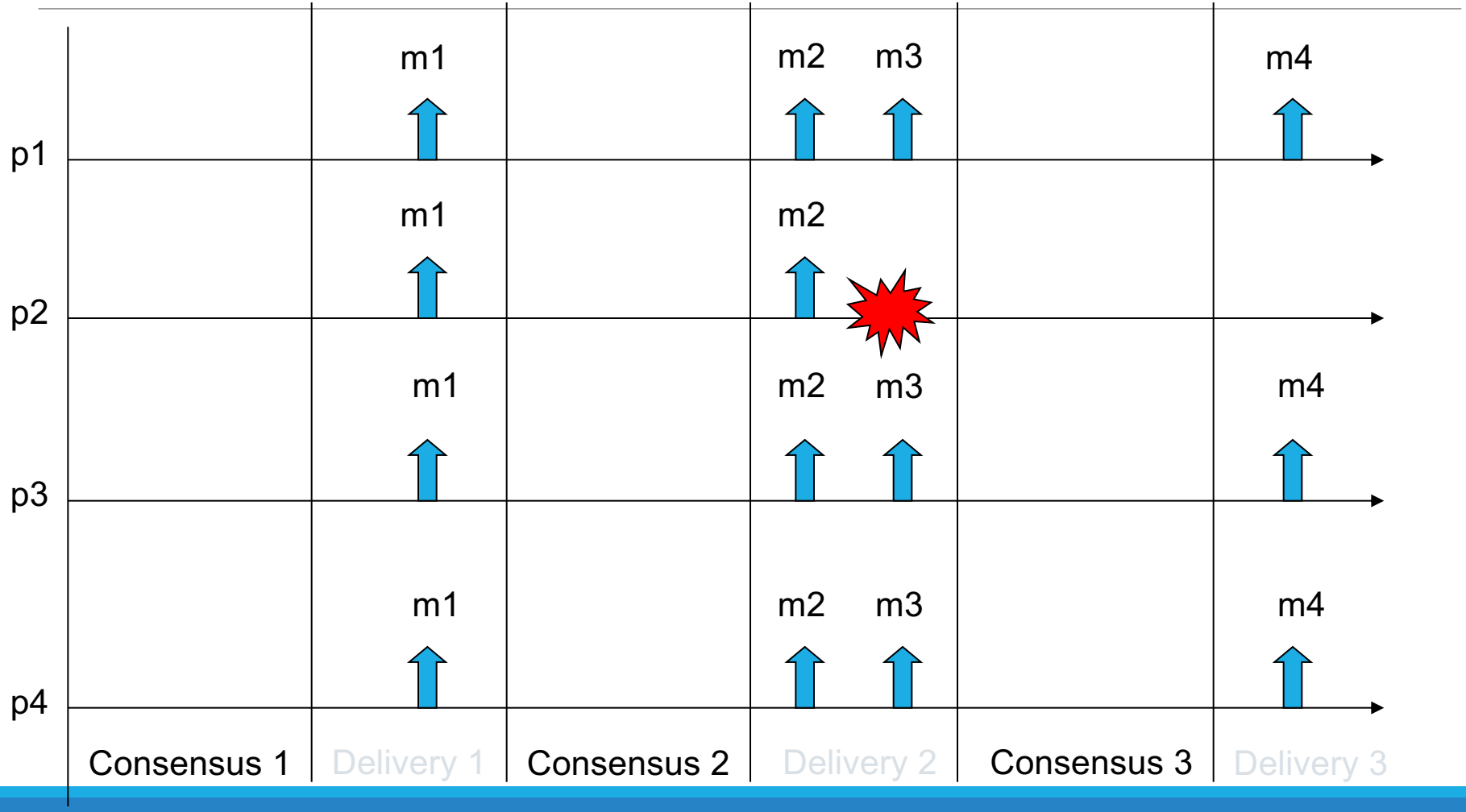
It depends from the assumptions about Reliable Broadcast and Consensus

Consensus / Reliable Broadcast	Uniform	Non Uniform
Uniform		
Non Uniform		

Example 1 (UC and URB)



Example 2 (UC and URB)



Uniform Consensus (UC) and Uniform Reliable Broadcast (URB)

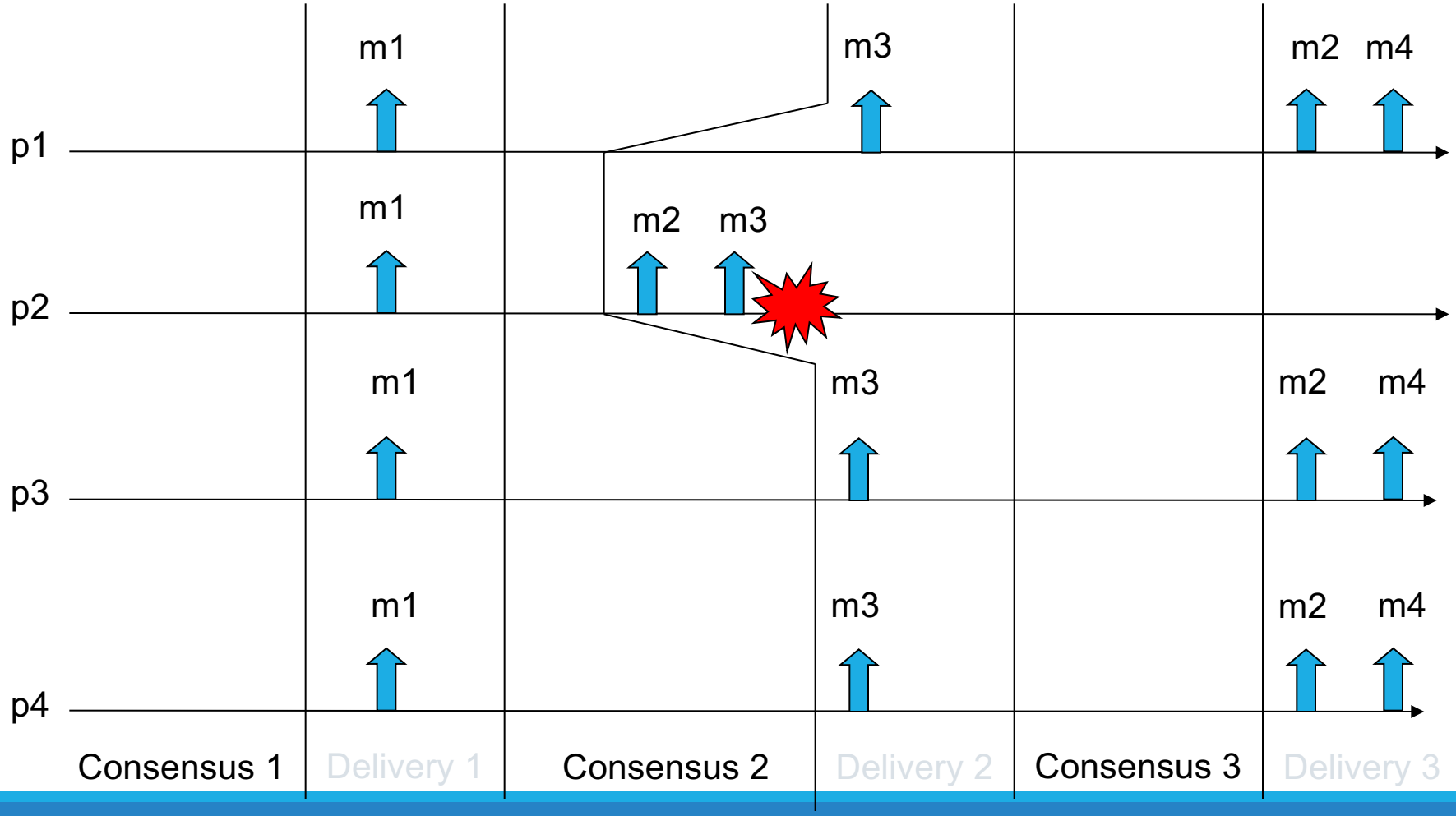
Assuming both Consensus and Reliable Broadcast uniform we have

TO (UA, SUTO)

Proof.

- Due to URB all the processes (even the faults) deliver the same set of messages
- The unordered buffer contains the same set of messages for each process
  - All the processes will deliver the same set of messages (UA)
- Due to UC, all processes (even the faults) decide for the same list of messages
- Messages are sorted by a deterministic rule
  - All processes will deliver the messages in the same order

Example (NUC and URB)



Non Uniform Consensus (NUC) and Uniform Reliable Broadcast (URB)

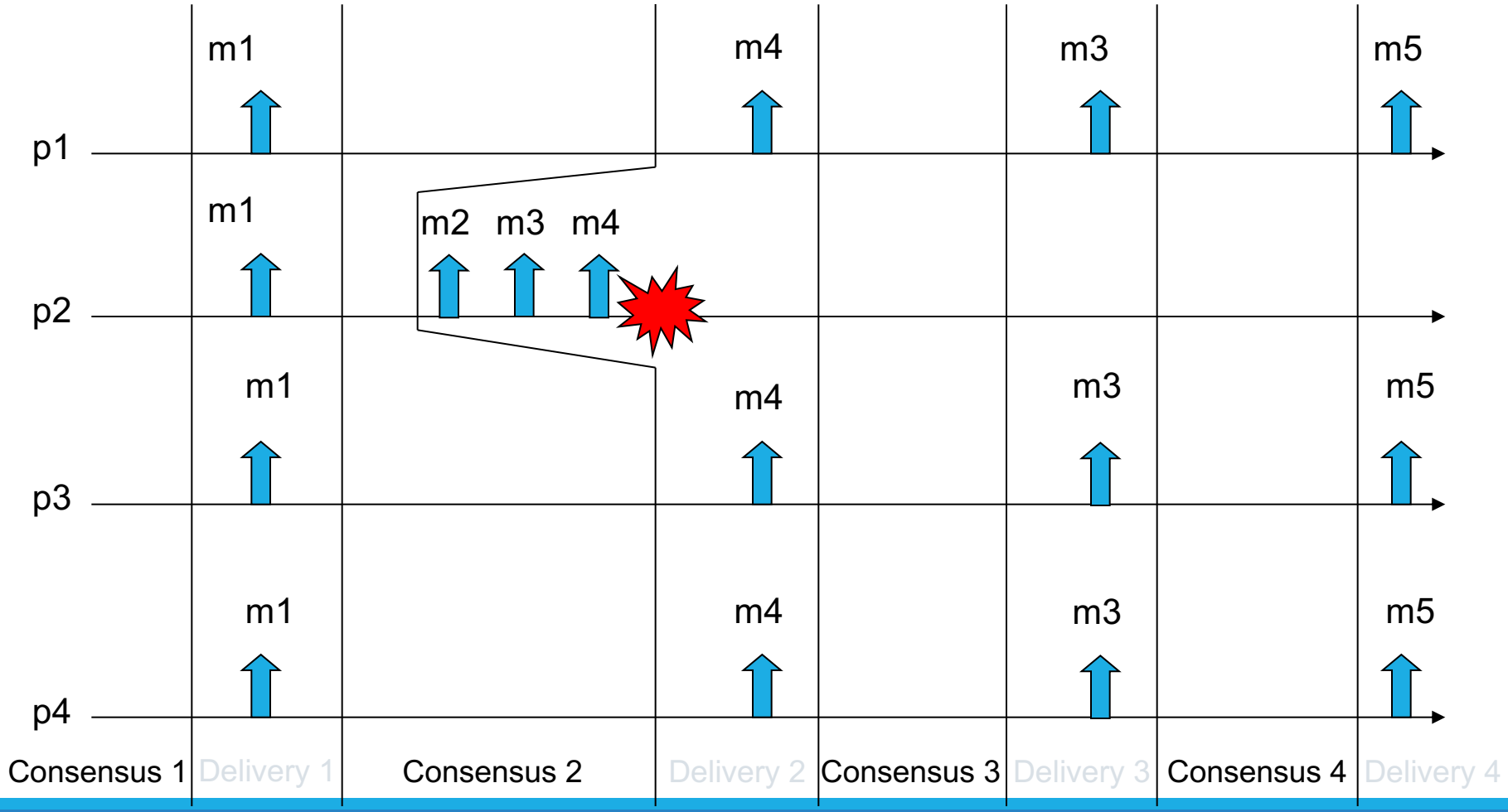
Assuming both Consensus and Reliable Broadcast uniform we have

TO (UA, WNUTO)

Proof.

- Due to URB all the processes (even the faults) deliver the same set of messages
- The unordered buffer contains the same set of messages for each process
  - All the processes will deliver the same set of messages (UA)
- Due to NUC, all correct processes decide for the same list of messages
- Faulty processes can decide differently
  - All correct processes will deliver the messages in the same order
  - Faulty processes will deliver, just before a crash, a different sequence of messages

Example (NUC and NURB)



Non Uniform Consensus (NUC) and Non Uniform Reliable Broadcast (NURB)

Assuming both Consensus and Reliable Broadcast uniform we have

TO (NUA, WNUTO)

Proof.

- Due to NURB correct processes deliver the same set of messages
- Faulty processes can deliver other messages
  - Only correct processes will deliver the same set of messages (NUA)
- Due to NUC, all correct processes decide for the same list of messages
- Faulty processes can decide differently
  - All correct processes will deliver the messages in the same order
  - Faulty processes will deliver, just before a crash, a different sequence of messages

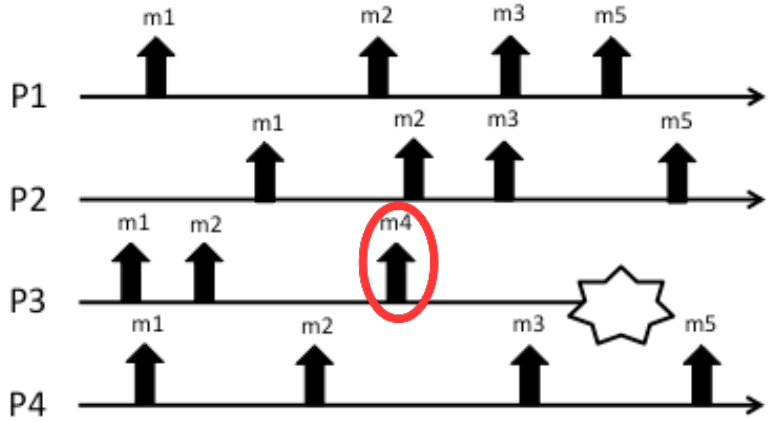
Consensus / Reliable Broadcast	Uniform	Non Uniform
Uniform	UA SUTO	UA WNUTO
Non Uniform		NUA WNUTO

Exercice

Which specification is satisfied assuming UC and NURB?

Exercice

Consider the run depicted in the figure:



1. Which type of total ordering is satisfied by the run? Specify both the agreement and the ordering properties.
2. Modify the run in order to satisfy TO(UA, WUTO) but not TO (UA SUTO)
3. Modify the run in order to satisfy TO(NUA, WNUTO) but not TO(NUA, WUTO)

1) Correct deliver the same set : m_1 m_2 m_3 m_5

Faulty : deliver m_1 m_2 m_4 — not delivered by correct

Agreement : NON-UNIFORM

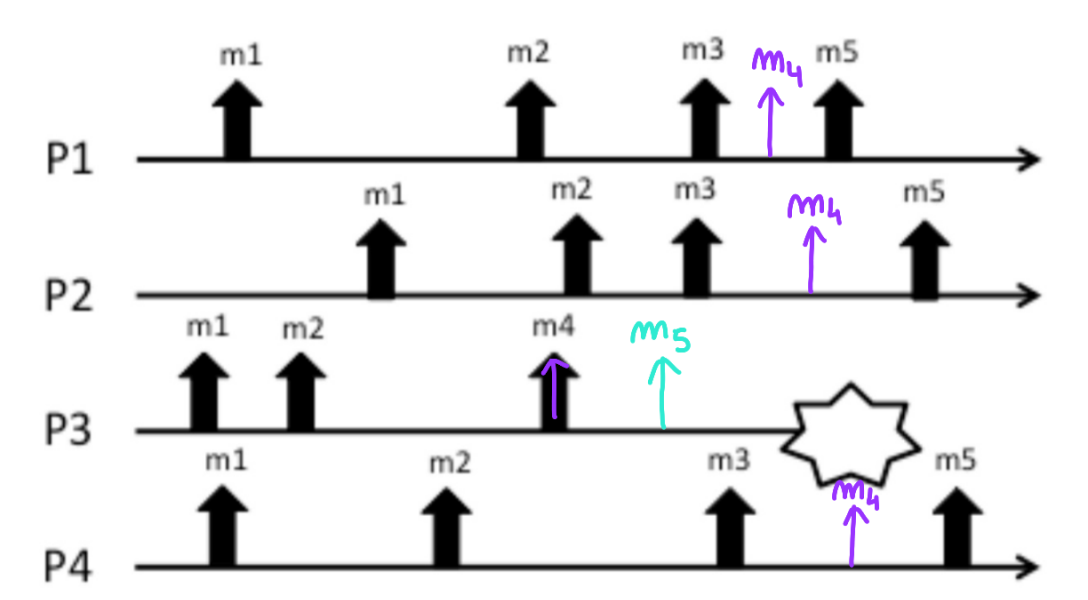
Ordering : SUTO

↓

m_1 m_2 omission (m_3) m_4

↳ diverging

2) TO(UA, WUTO) but not TO(UA, SUTO)



References

C. Cachin, R. Guerraoui and L. Rodrigues. Introduction to Reliable and Secure Distributed Programming, Springer, 2011

- Chapter 6 – Section 6.1

Stefano Cimmino, Carlo Marchetti, Roberto Baldoni ”*A Guided Tour on Total Order Specifications*” WORDS Fall 2003: 187-194