

Specifications of Totally Ordered Communication

Corso di Sistemi distribuiti

Laurea Specialistica in Ingegneria Informatica

Total Order Communication

Common understanding: “processes deliver messages in the same order”

- Always true for correct processes, but faulty?
 - A faulty process could skip or deliver messages in a order different from a correct process
- Some of these behaviors are known in the literature
..... not the whole story
 - (Hadzilacos-Toueg 93, Schiper et al 95)

Motivation of the work

- Active Replication in Air Traffic control
- Application designers use TO communication as a basic block to build applications resilient to crash failures
- If TO behaviors are not well understood they can kill (or violate the safety of) the application upon the occurrence of failures.
- ...nice paradox
- So we need a clear and common specification!

Specification

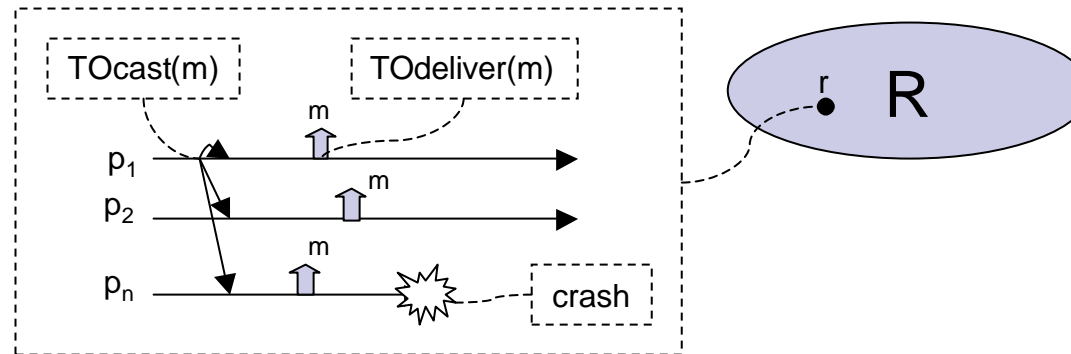
- A specification allows to understand how behaviors of faulty processes are constrained.
 - Application designers can thus understand if such behaviors can be tolerated by its application
 - TO designer can position their implementation in a right context looking at which behaviors can be generated by an implementation
- Our contribution has been to provide a unifying formal framework:
 - understanding all possible bad behavior of faulty processes through a systematic approach
 - derive all possible TO specifications
 - showing relations among such *specifications*,
 - allowing to classify and compare TO *implementations*

Content

- Hierarchy of TO specifications in static system
- Simple Methodology to Classify TO implementations
- Impact on applications
- Extension to dynamic systems
- TO in Group Toolkit

System model

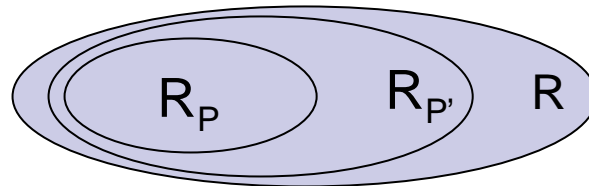
- Static set of processes ? = $\{p_1 \dots p_n\}$
- Message passing over reliable channels (message exchanging between correct processes is reliable)
- Asynchronous
- Crash fault model for processes
- We characterize the system in terms of its possible runs
R



A few notation

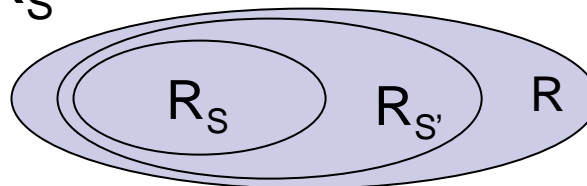
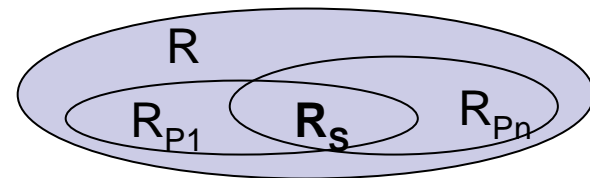
- Property P : predicate on the system, identifying a set of runs $R_P \subseteq R$

□ $P \Rightarrow P'$ iff $R_P \subseteq R_{P'}$



- Specification $S(P_1, \dots, P_m)$: logical and of m properties, identifying a set of runs $R_S = R_{P_1} \cap \dots \cap R_{P_m} \subseteq R$

□ $S \supseteq S'$ iff $R_S \subseteq R_{S'}$



TO specifications

Total order specifications are usually composed by four properties, namely Validity, Integrity, Agreement, and Order.

- 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 = $S(V, I, A, O)$

- V = ~~NUV~~ Validity

- I = Integrity

- A = Agreement

- O = Order



TO(A, O)

- 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 (correct or not)

TO Specifications

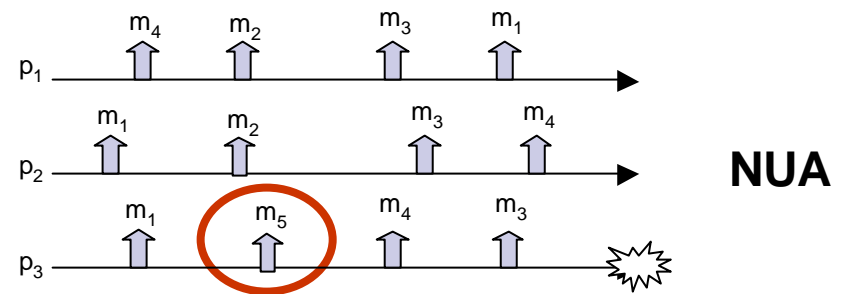
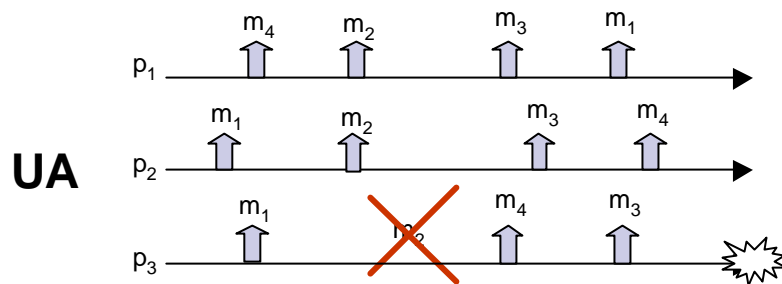
- Crash failure + Reliable channels \Rightarrow
 - **NUV.** if a *correct* process TOCAST a message m then some *correct* process will eventually deliver m
 - **UI.** For any message m , every process p delivers m at most once and only if m was previously tocast by some (correct or not) process.

The Agreement property

- **(Uniform Agreement, UA)** If a process delivers a message m , then all correct processes will eventually deliver m ;
- **(Non-uniform Agreement, NUA)** If a correct process delivers a message m , then all correct processes will eventually deliver m

The Agreement property

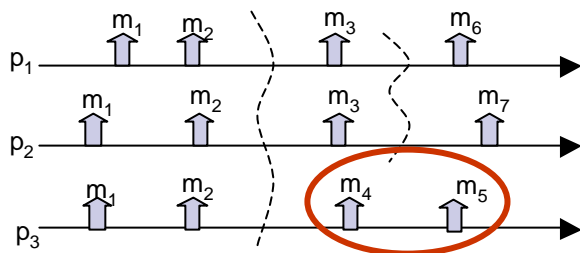
- Constrains the set of delivered messages
 - Correct processes always deliver the same set of messages M
 - Each faulty process p delivers a set M_p
 - UA: $M_p \subseteq M$
 - NUA: M_p can be s.t. $M_p - M \neq \emptyset$



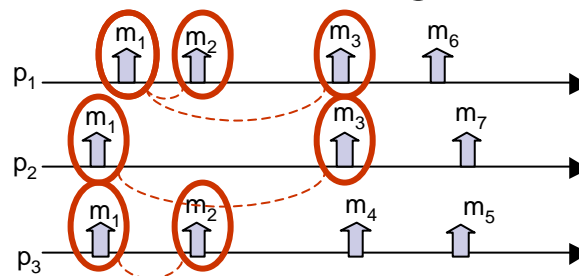
The Order property

- Constrains the order of message deliveries and possibly the set of delivered messages
 - SUTO: if p delivers $m < m'$, q delivers m' only after m
 - same order
 - same prefix of the set of delivered messages
 - after an omission, disjoint sets of delivered messages
 - WUTO: if p, q deliver m, m' , they get the same order
 - no restrictions on the set of delivered messages

SUTO



WUTO

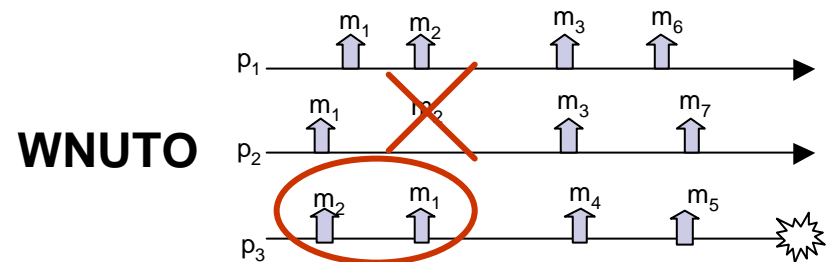
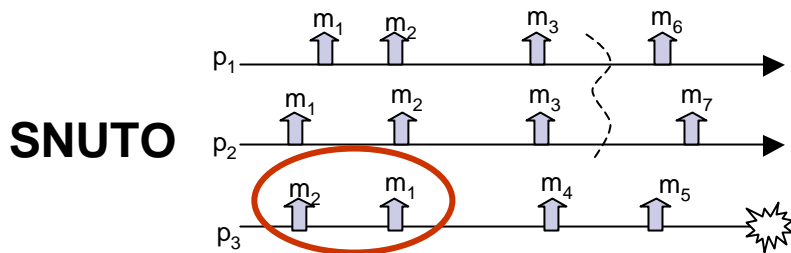


The Order property (2)

- SUTO and WUTO are uniform
- They both have a non-uniform counterparts: SNUTO and WNUTO
- (Strong Non-uniform Total Order, SNUTO). If some correct process p delivers some message m before message m' , then a correct process q delivers m' only after it has delivered m .
- (Weak Non-uniform Total Order, WNUTO) If correct processes p and q both deliver messages m and m' , then p delivers m before m' if and only if q delivers m before m'

The Order property (2)

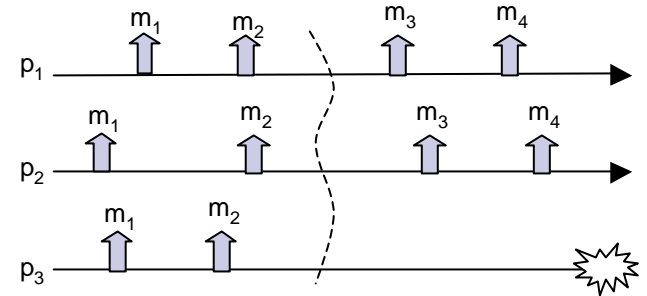
- SUTO \Rightarrow WUTO
- SNUTO \Rightarrow WNUTO



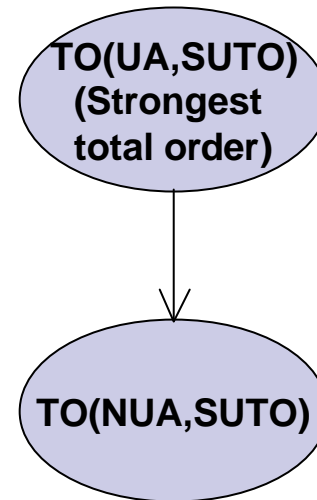
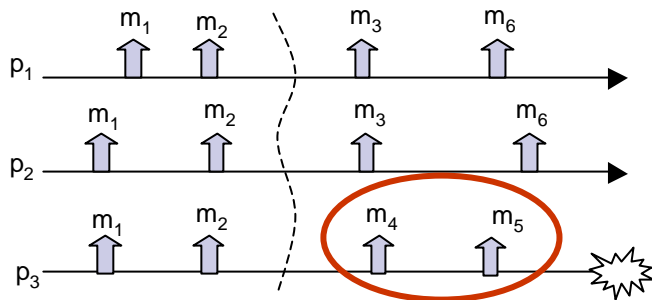
TO specifications

- **TO(UA,SUTO)**

 - The strongest TO spec.

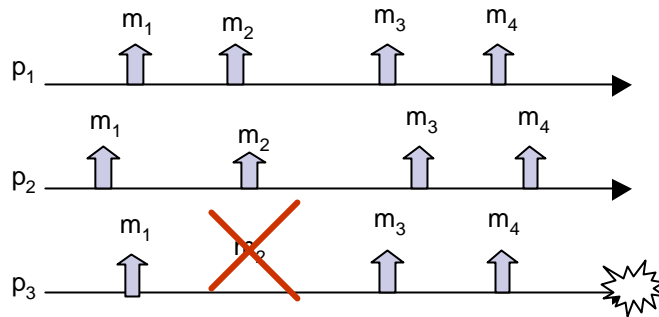


- **TO(NUA,SUTO)**

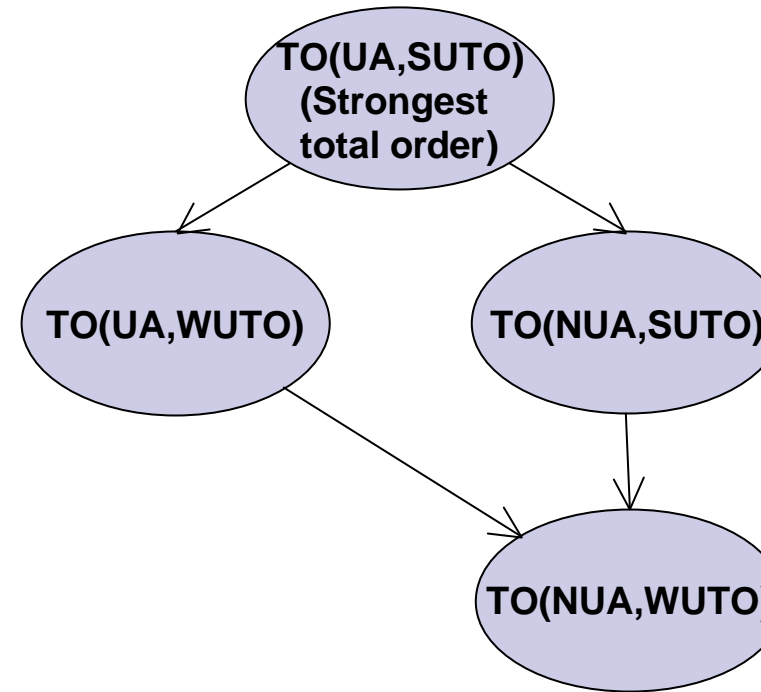
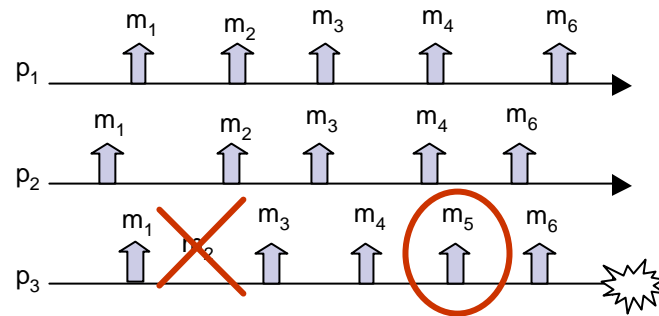


TO specifications (2)

■ TO(UA,WUTO)

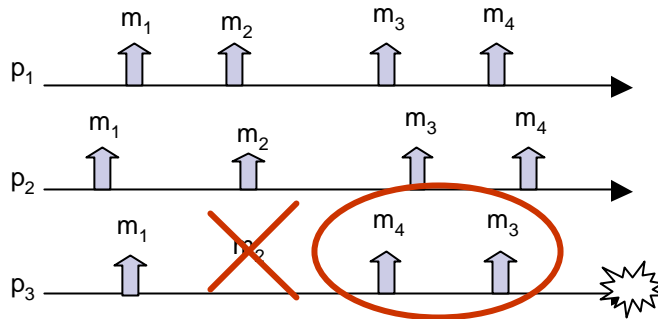


■ TO(NUA,WUTO)

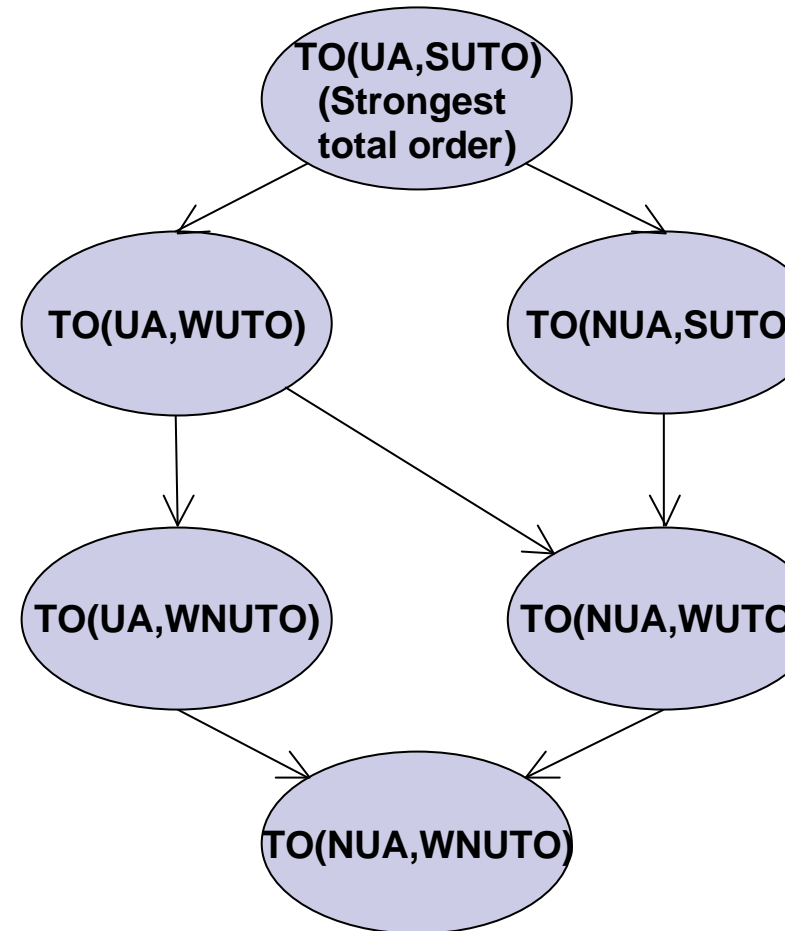
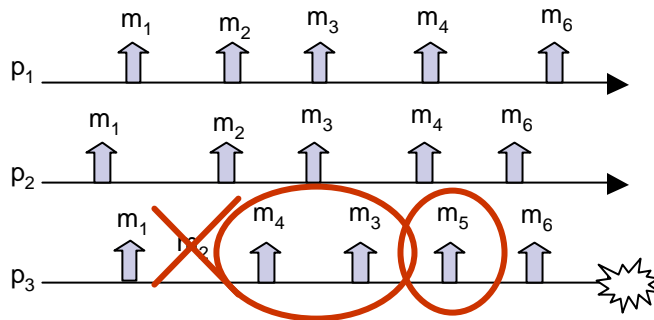


TO specifications (3)

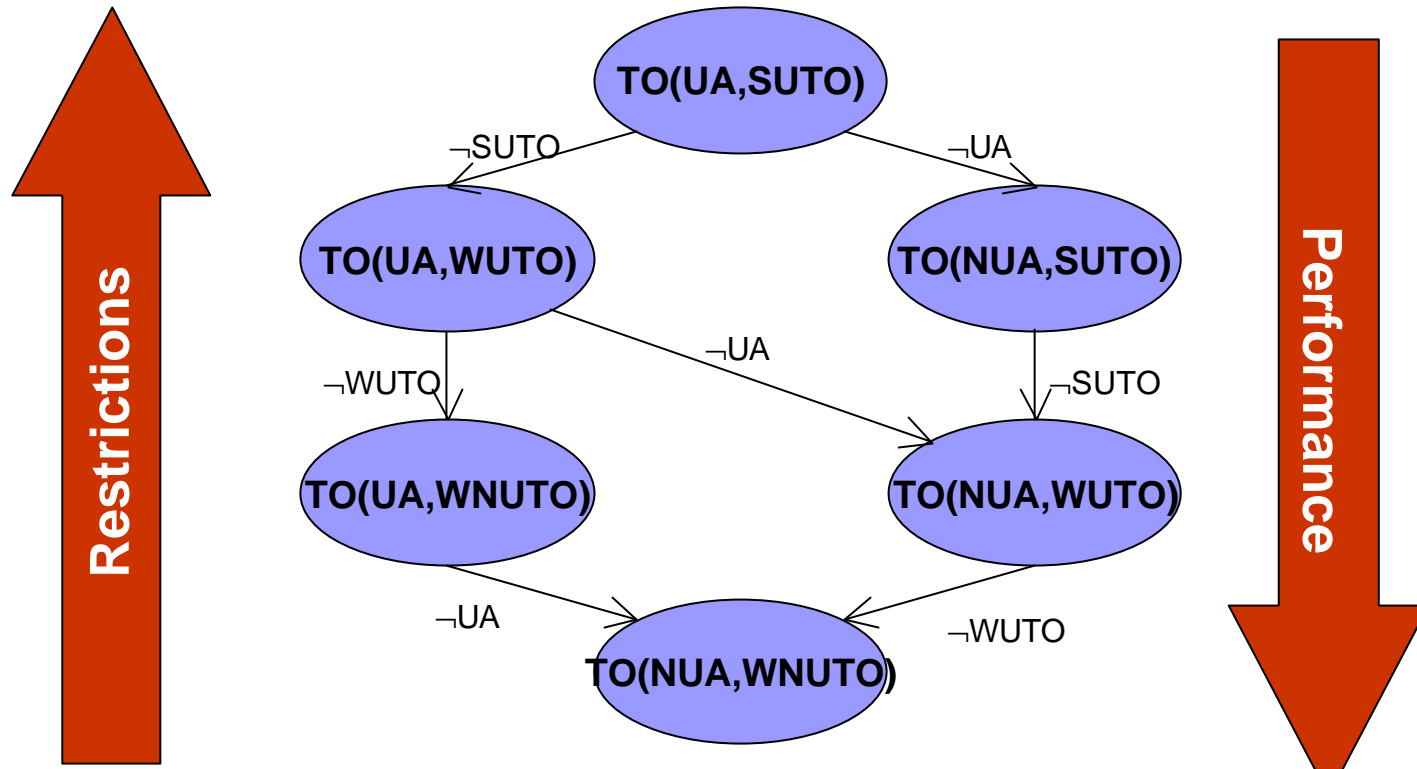
■ TO(UA,WNUTO)



■ TO(NUA,WNUTO)



Simple Methodology to classify implementations



$\neg UA$	\triangleq	$\exists p, q \exists m \text{ del}(p, m) \wedge \neg \text{del}(q, m) \wedge \text{correct}(q)$
$\neg SUTO$	\triangleq	$\exists p, q \exists m, m' \text{ del}(p, m) < \text{del}(p, m') \wedge \text{del}(q, m') \wedge (\neg \text{del}(q, m) \vee \text{del}(q, m') < \text{del}(q, m))$
$\neg WUTO$	\triangleq	$\exists p, q \exists m, m' \text{ del}(p, m) < \text{del}(p, m') \wedge \text{del}(q, m') < \text{del}(q, m)$

Behavior of Faulty Processes

- EP1: a faulty process p delivers a prefix of the ordered set of messages delivered by correct processes;
- EP2: a faulty process p delivers some messages which are not delivered by correct processes;
- EP3: a faulty process p skips the delivery of some messages delivered by correct processes;
- EP4: a faulty process p delivers some messages in an order different from correct processes.

TO specification	Admitted differences in the histories of processes p (faulty) and q (correct)
$TO(UA, SUTO)$	EP1
$TO(UA, WUTO)$	EP1 or EP3
$TO(UA, WNUTO)$	EP1 or EP3 or EP4
$TO(NUA, SUTO)$	EP1 or EP2
$TO(NUA, WUTO)$	EP1 or EP2 or EP3
$TO(NUA, WNUTO)$	EP1 or EP2 or EP3 or EP4

Impact on Applications

- Guaranteeing one-copy serializability on a replicated database with deterministic transactions.
- It suffices
 1. All replicas serialize transactions in the same order
 2. Replicas executes the same set of transactions
- Assuming replicas serialize transactions according to the order of transactions delivery, using a primitive compliant to TO(UA, SUTO) solves the problem

Impact on Applications

- Weaker TO primitives needs an application logic to handle possible inconsistencies due to faulty processes
- Introducing a voting phase managed by a “fault-tolerant” software component C into the application logic
- Assume a majority of correct processes

Impact on Applications

- TO(NUA,SUTO). Due to EP2, a faulty replica could commit a transaction never delivered to correct replicas violating OCS
- Solution:
 - C sets a timer and sends the transactions to replicas using the TO(NUA,SUTO) primitive.
 - Each replica sends a prepare-to-commit message
 - C waits either for a majority of prepare-to-commit messages or for timer expiration.
 - In the first case, send a commit message through a reliable broadcast primitive.
 - In the second case sends an abort message

Impact on Applications

- $TO(UA, WUTO)$ or $TO(NUA, WUTO)$. Due to **EP2 and EP3**, a faulty replica could skip the delivery of a transaction violating OCS
- Solution:
 - C sets a timer and sends the transactions to replicas using the TO primitive.
 - Each replica sends a prepare-to-commit message
 - C waits either for a prepare-to-commit message from each non-faulty replicas or for timer expiration.
 - As in the previous case
- C requires on-the-fly monitoring replicas

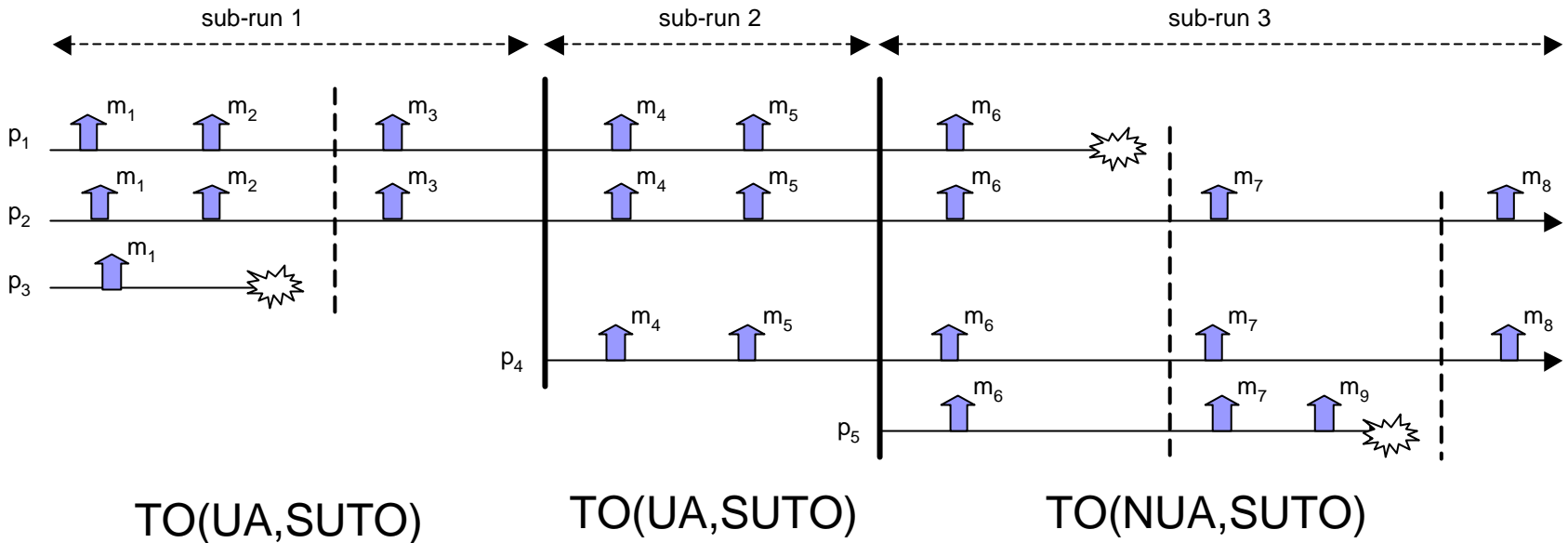
Impact on Applications

- TO(UA,WNUTO) or TO(NUA,WNUTO). Due to EP2, EP3 **and** EP4, a faulty replica could deliver transactions in different orders wrt a correct replica violating OCS
- C requires to handle out of order arrivals of prepare-to-commit messages.

Extension to dynamic systems

- Processes can join and leave at any time
- Consider a dynamic system as a sequence of static system runs
- The TO specification implemented by an implementation I is the weakest TO specification guaranteed in a static run belonging to one of all runs generated by I

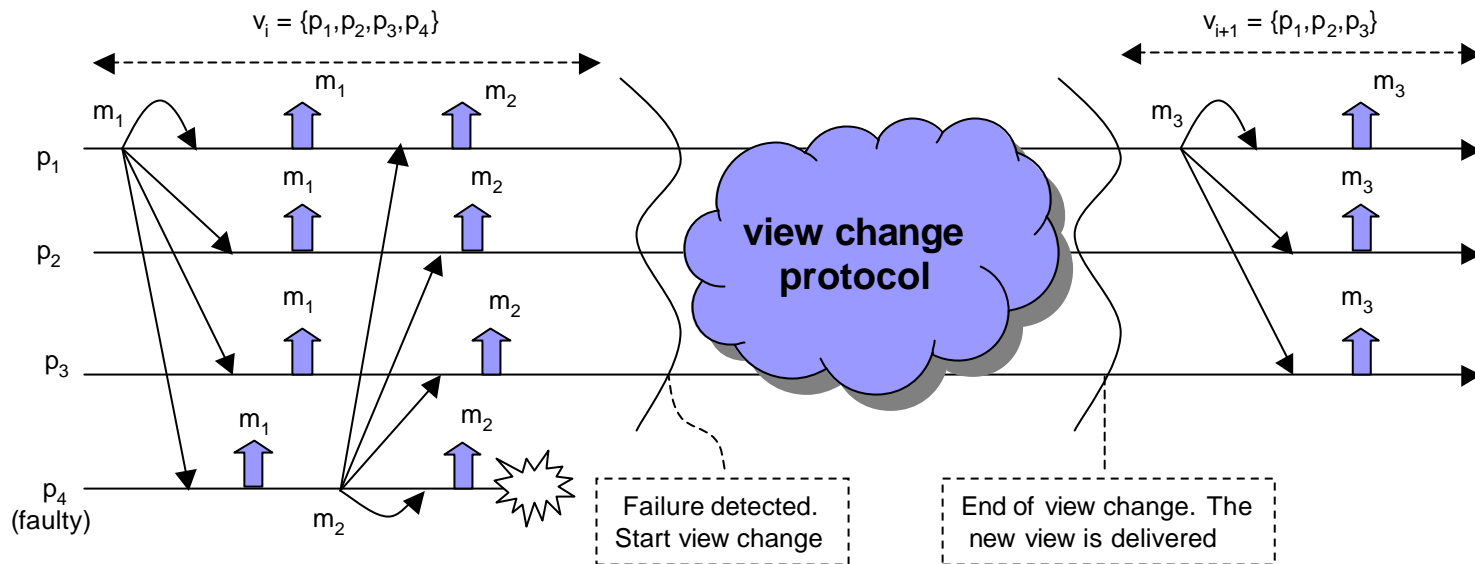
Extension to dynamic systems



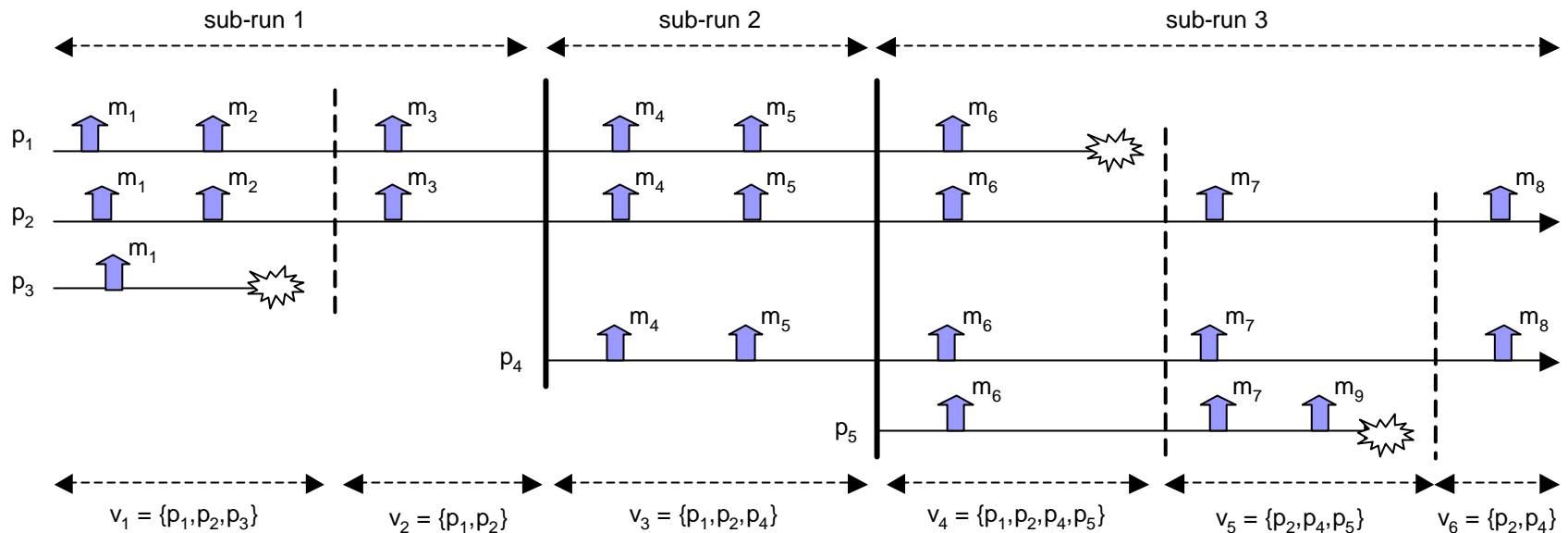
A protocol Π producing this message pattern satisfies at most $TO(NUA, SUTO)$ i.e., it does not satisfy $TO(UA, SUTO)$

Group Toolkits

- Notion of Join, leave, view, view synchrony etc.



Group Toolkits



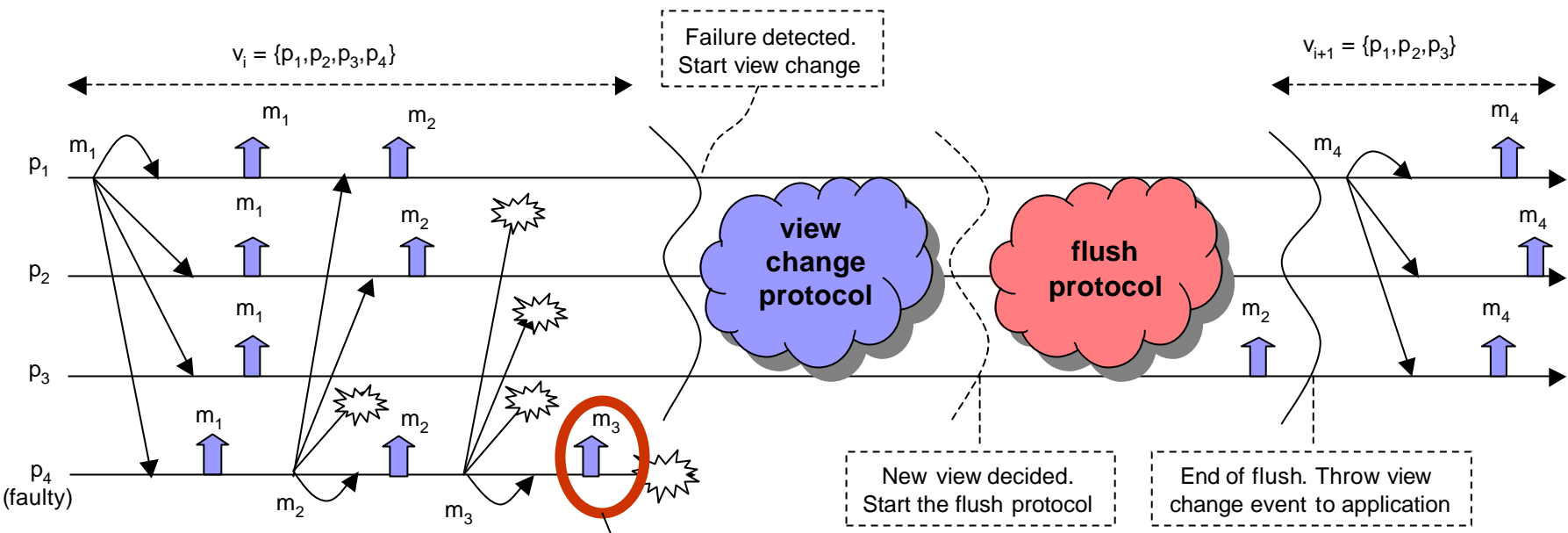
View Synchrony property: “all correct processes that pass to the next view deliver the same set of messages in the current view”

Protocol mechanisms to implement view synchrony:

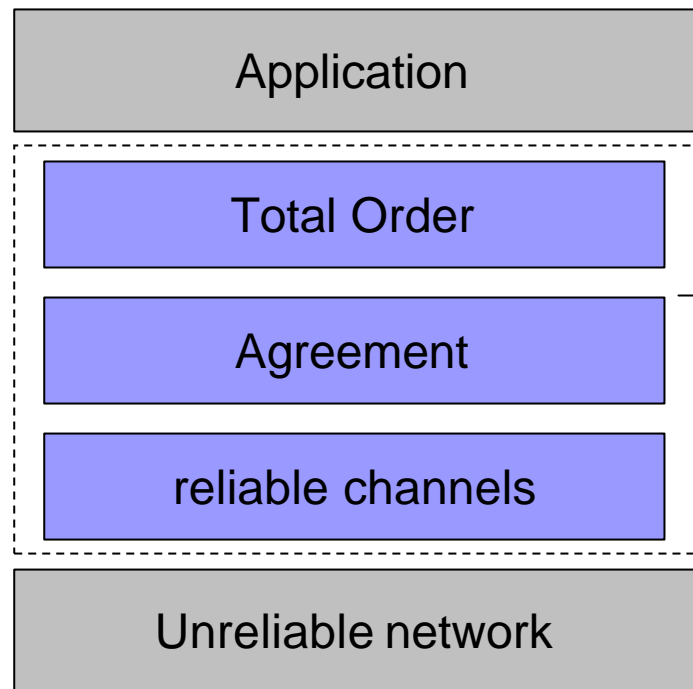
- Membership protocol

- Flush

- Stability tracking



Example of a run with a flush protocol
 This run is NUA due to m_3



UA or NUA

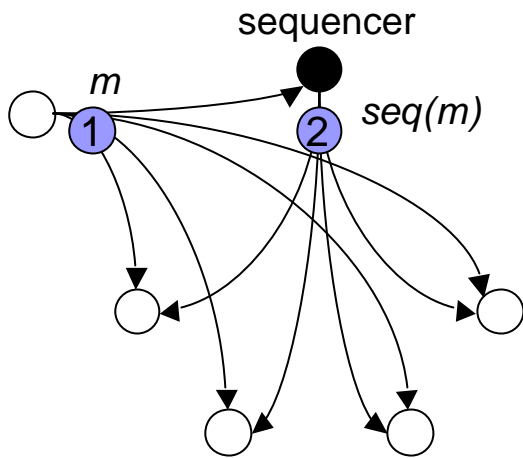
NUA is guaranteed by
"View synchrony and
reliable channels"

UA is guaranteed by
"View synchrony,
Stability tracking and
reliable channels"

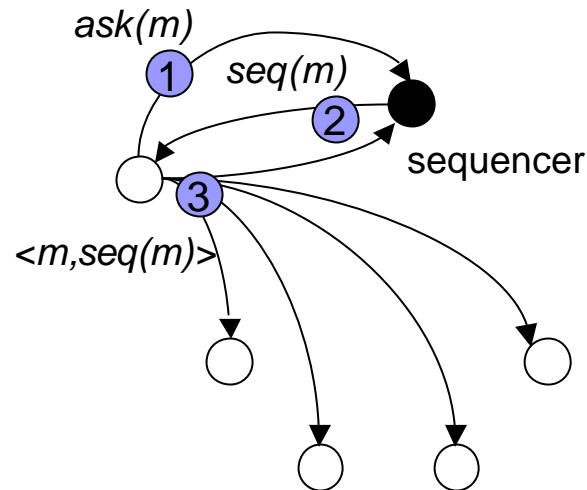
Algorithms to implement order in group toolkits

- Token Based (Spread, JavaGroups)
- Sequencer Based (JavaGroups, Ensemble)

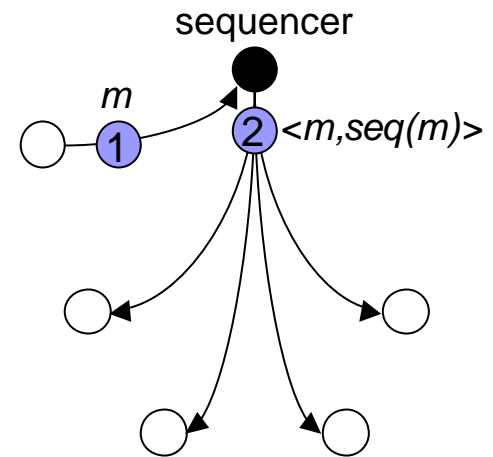
Sequencer Based



Broadcast-broadcast (BB)



Ask-broadcast (BB)

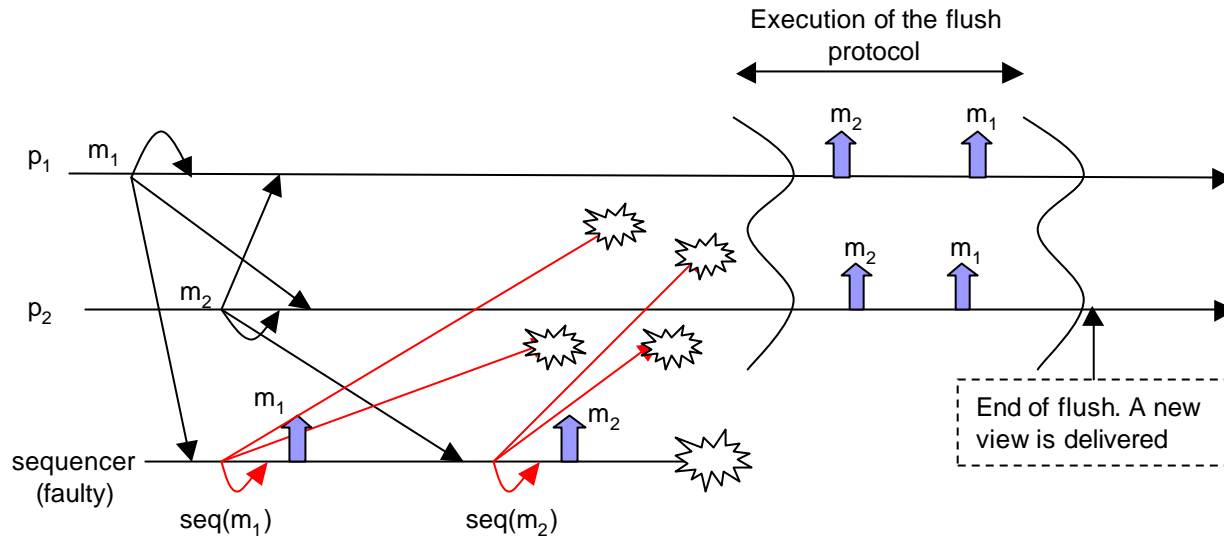


Send-broadcast (BB)

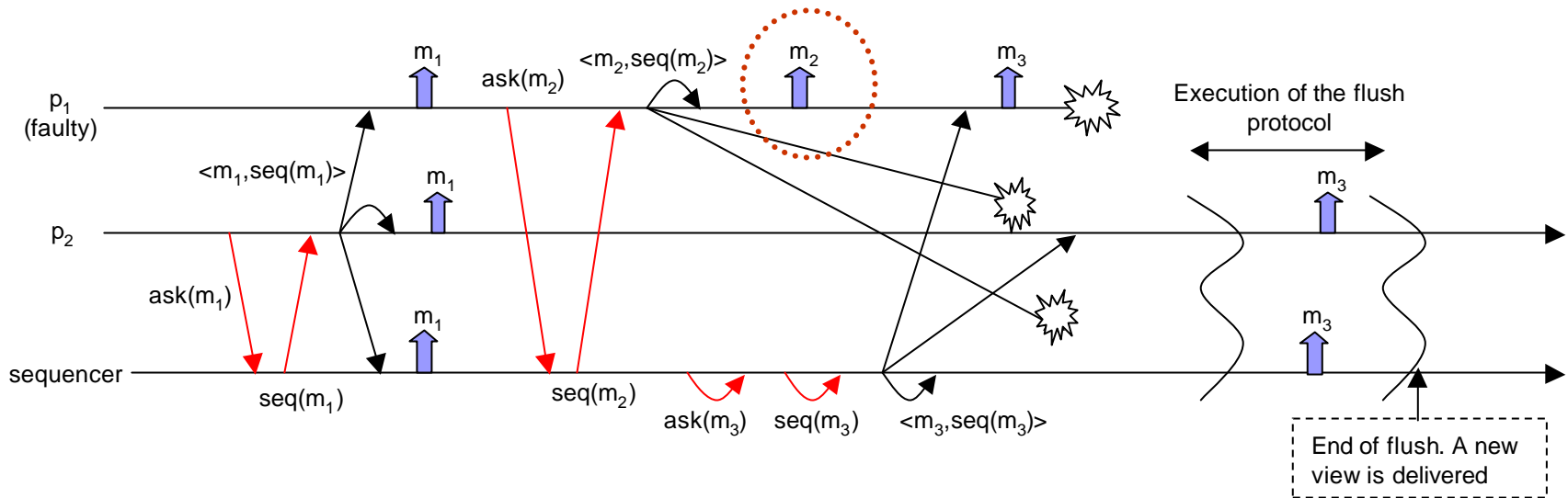
Sequencer Based

- Underlying system enforces UA
 - All sequencer based algorithms guarantee $TO(UA, SUTO)$
- Underlying system enforces NUA and not(UA)
 - BB and SB guarantee $TO(NUA, WNUTO)$
 - AB guarantees $TO(NUA, WUTO)$

BB satisfies WNUTO



AB satisfies WUTO



Sequencer Based

Ordering protocol	Agreement	TO specification	System
Broadcast-broadcast sequencer	NUA	$TO(NUA, WNUTO)$	Ensemble(Seqbb)
	UA	$TO(UA, SUTO)$	-
Send-broadcast sequencer	NUA	$TO(NUA, WNUTO)$	Ensemble(Sequencer)
	UA	$TO(UA, SUTO)$	-
Ask-broadcast sequencer	NUA	$TO(NUA, WUTO)$	JavaGroups(TOTAL)
	UA	$TO(UA, SUTO)$	-
Token with stability tracking	NUA	$TO(UA, SUTO)$	Spread(Safe) JavaGroups(TOTAL_TOKEN)
Token without stability tracking	NUA	$TO(NUA, WUTO)$	Spread(Agreed)

Table 5: TO specification enforced by each ordering protocol