Technische Universität Berlin

Distributed Systems
Exercise 2: Clocks & Orderings

FG Komplexe und Verteilte IT System | Distributed Systems SS2018

Agenda

- Organizational Matters
- Ordering of Events and Causality
  - Happened-before Relation
- Causal Communication & Causal Orderings
  - Message Sequencers
  - Lamport Timestamps
  - Vector Clocks
- Exercise 2
  - Proposed Architecture
  - Queues & Multithreading

Organizational Matters

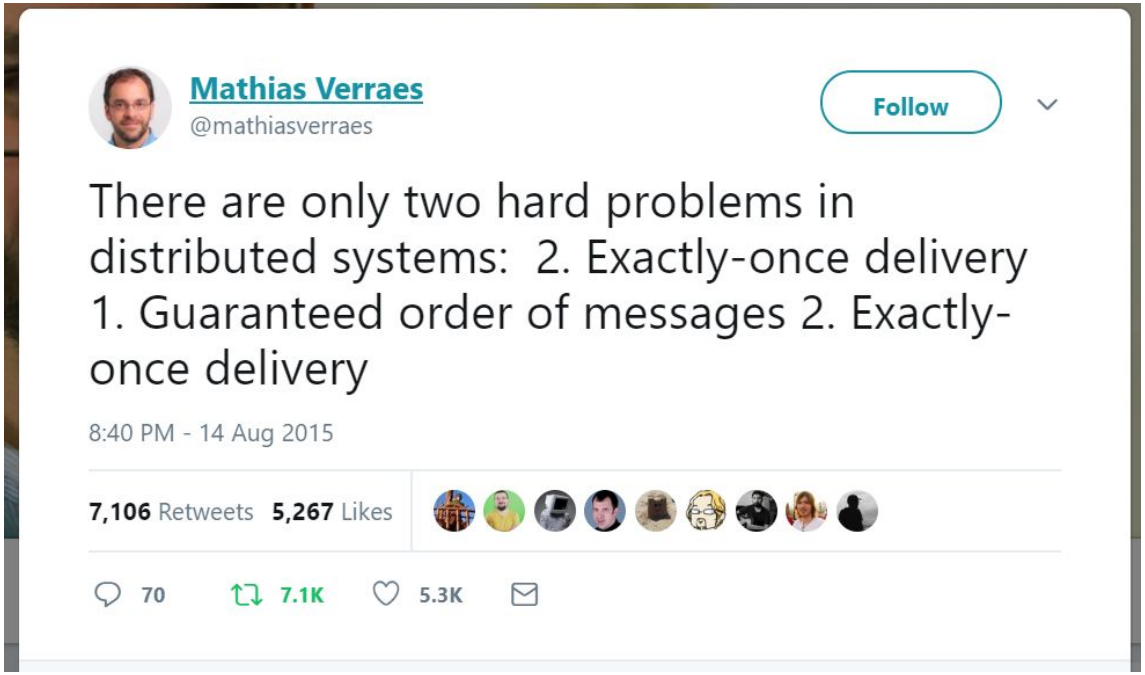
1

- Please register for the exam (Modulprüfung) via QISPOS until **May 18th**
- If you cannot register via QISPOS, please contact Tim or Jana as soon as possible

2

- Please find a group (3-4 people) and register for the first exercise evaluation (Mündliche Rücksprache) in ISIS until **May 23th**

Ordering of Events and Causality



Happened-before Relation

In distributed systems we define a causal relation between events with the happened-before relation.

For two events *a* and *b* we define the happened-before relation (→) as follows:

- If events *a* and *b* occur on the same process, $a \rightarrow b$ if the occurrence of event *a* preceded the occurrence of event *b*.
- If event a is the sending of *a* message and event *b* is the reception of the message sent in event *a*, then $a \rightarrow b$

Events that cannot be ordered by happened-before are said to be concurrent

- E.g. two processes independently send messages

Note: the happened-before relation is no precise way of tracking causality.

- Two completely unrelated events that occurred on the same process are ordered by the happened-before relation, even though there is no real causal dependency.

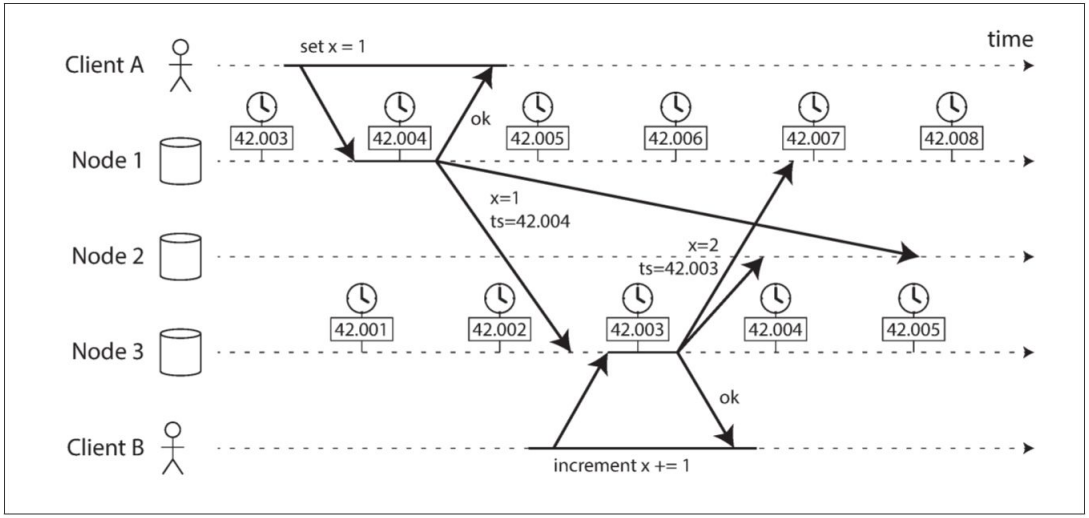
How (Not) to Order Events

How can multiple processes agree on an order of events?

Idea 1: Ordering by timestamps of the physical clocks of the processes!

Problem: clock drift

- The physical clocks can count the passed time with a different speed
  - Google assumes a clock drift of 6ms for a clock that is resynchronized every 30sec.



From: Designing Data-Intensive Applications (Kleppmann)

How (Not) to Order Events

How can multiple processes agree on an order of events?

Idea 2: Use independent sequence numbers for the processes! For example, the first process uses even numbers and the second process uses odd numbers to mark events.

Problem: No relative speed between the processes.

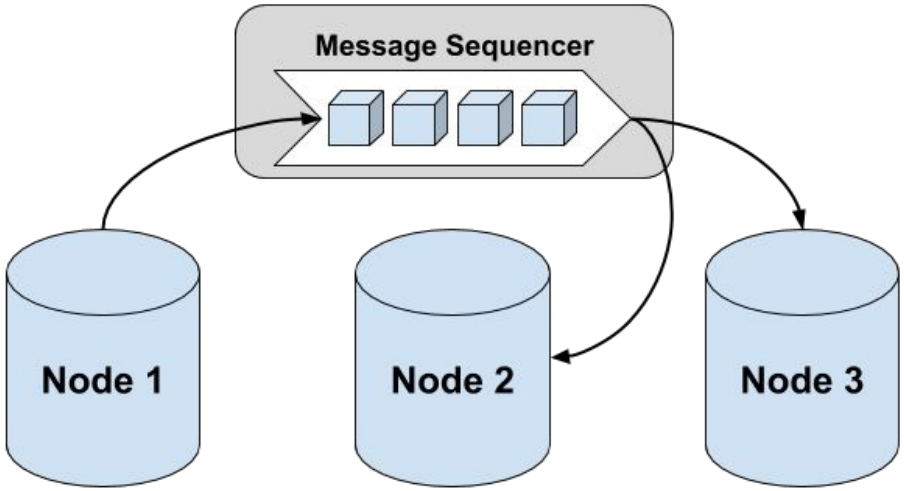
- Processes can observe a different number of events per second.
- One process can lag behind.
- Thus, having for two events from different processes it is impossible to accurately determine which one causally happened first.

Solution 1: Message Sequencing

How can multiple processes agree on an order of events?

Solution 1: One could use a *central* instance that sequences events.

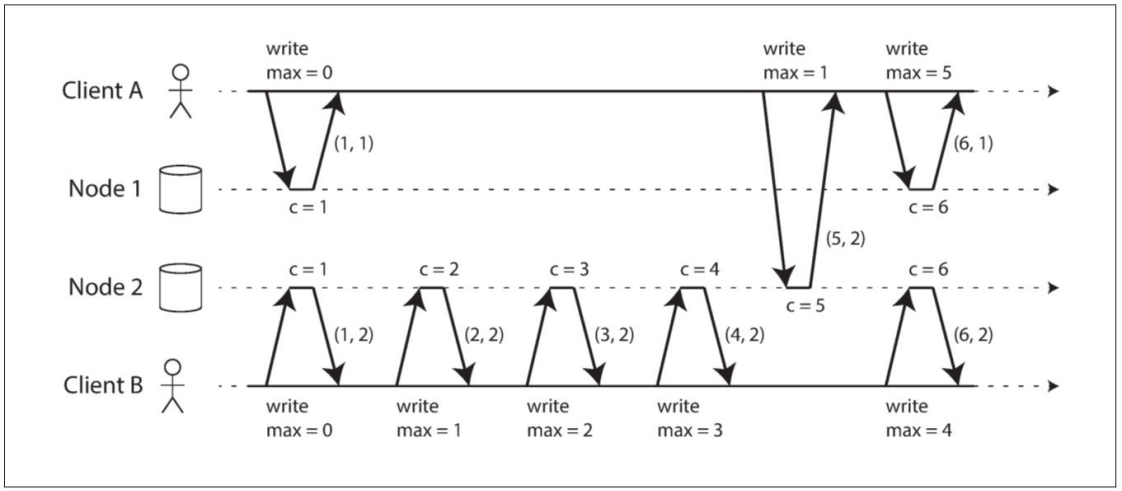
- All communication is proxied by the message sequencer.
- The message sequencer broadcasts the events in the order which they were received at the sequencer.



Solution 2: Lamport Timestamps

How can multiple processes agree on an order of events?

Solution 2: Annotate the events with logical clocks and the current maximum (Lamport Timestamps)



From: Designing Data-Intensive Applications (Kleppmann)

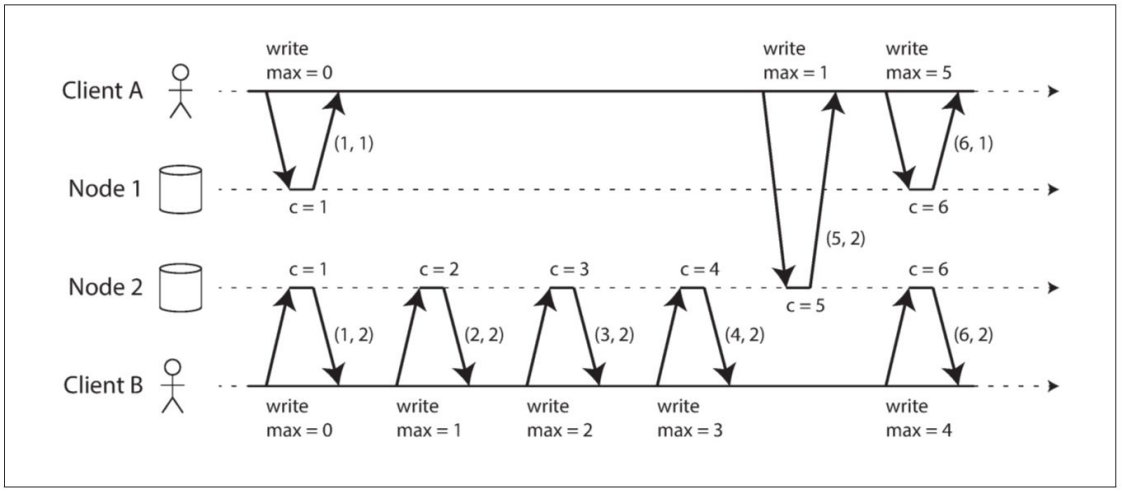
Lamport Timestamps:

- Each node stores a counter of the number of processed events.
- Each event is tagged with the node ID and the current counter value.
- Once a node receives an event with a higher counter, the local counter is updated to the received value.

Solution 2: Lamport Timestamps

How can multiple processes agree on an order of events?

Solution 2: Annotate the events with logical clocks and the current maximum (Lamport Timestamps)



From: Designing Data-Intensive Applications (Kleppmann)

Ordering of Events for two Lamport Timestamps:

- The event with the greater counter value has the greater timestamp.
- If the counter values are identical, the one with the greater node ID has the greater timestamp

Solution 3: Vector Clocks

How can multiple processes agree on an order of events?

Solution 3: Annotate the events with counters for every process (vector clocks)!

Algorithm sketch:

- We assume n processes $p_1 \ldots p_n$
- VC1: Initially, all clocks are set to 0 at every process ($V_i[j] = 0$ for all $i,j = 1 \ldots n$)
- VC2: For every event the at p_i the clock is increased by 1 ($V_i[i] = V_i[i] + 1$)
- VC3: The current vector clock is attached to every message
- VC4: When p_i receives a timestamp t a merge operation is applied:
  - $V_i[j] = \max(V_i[j], t[j])$ for $j = 1 \ldots n$

Solution 3: Vector Clocks

How can multiple processes agree on an order of events?

Solution 3: Annotate the events with counters for every process (vector clocks)!

Two vector clocks V and V' can be compared as follows:

- $V = V'$, if $V[\, i\,] = V'[\, i\,]$ for every i
- $V \leq V'$, if $V[\, i\,] \leq V'[\, i\,]$ for every i
- $V < V'$, if $(V \leq V') \wedge (V \neq V')$

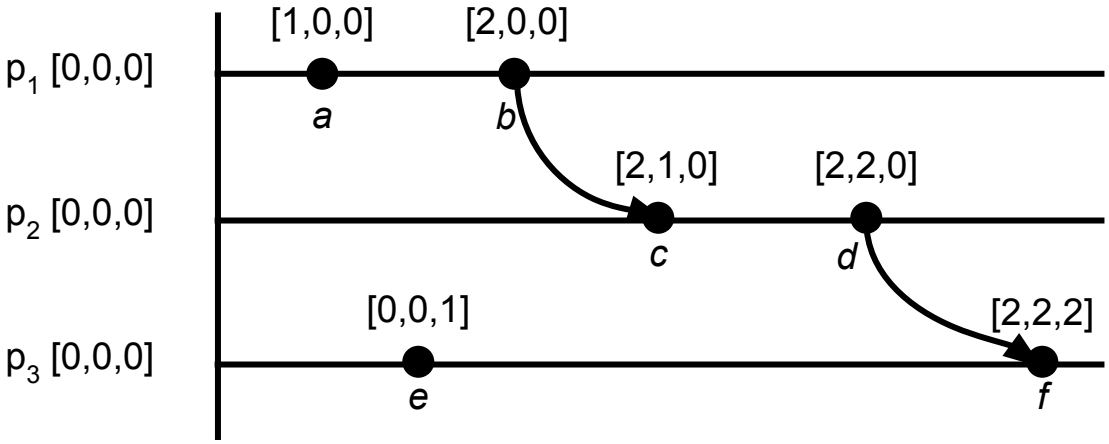
Note that the vector clock algorithm generates a *partial* order, not every two events can be ordered!

However, if $V(\, a\,) < V(\, b\,)$, then $a \rightarrow b$ (vector clocks ensure causal ordering)

Solution 3: Vector Clocks

How can multiple processes agree on an order of events?

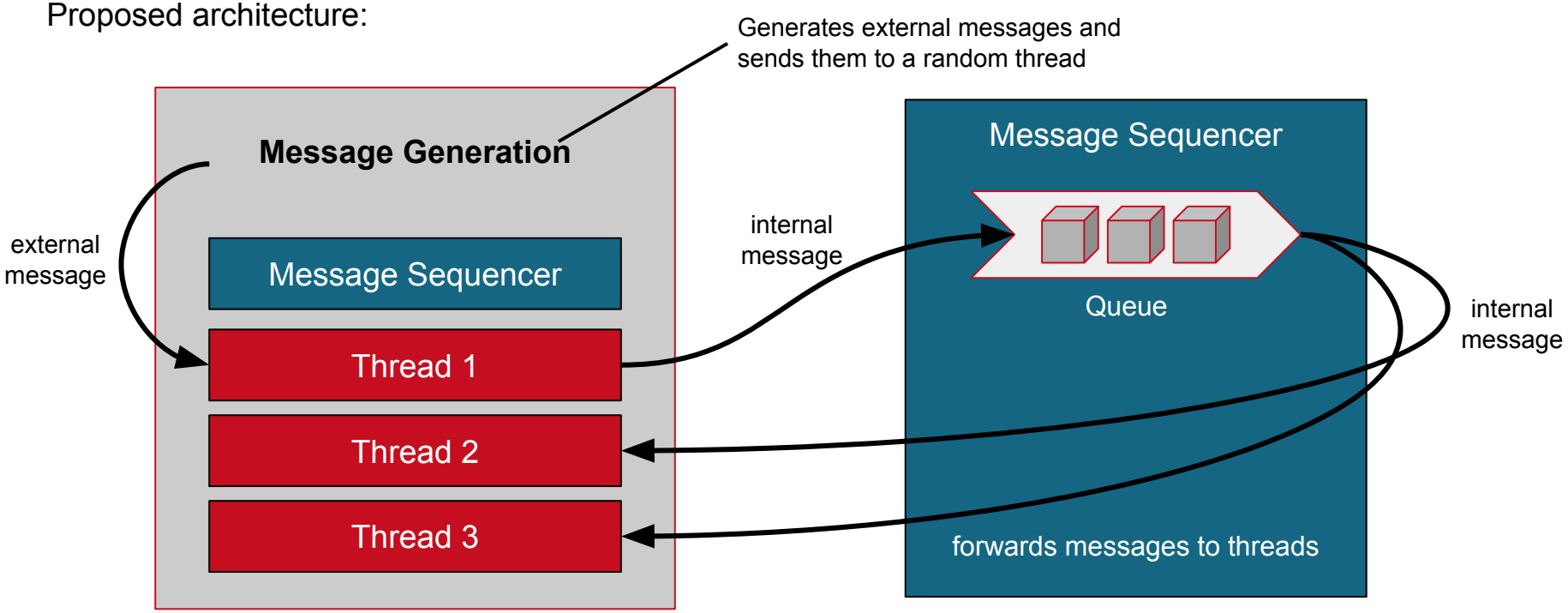
Solution 3: Annotate the events with counters for every process (vector clocks)!



Hints for the Exercise 2

Part 1: Implement a message sequencer for the communication between multiple threads.

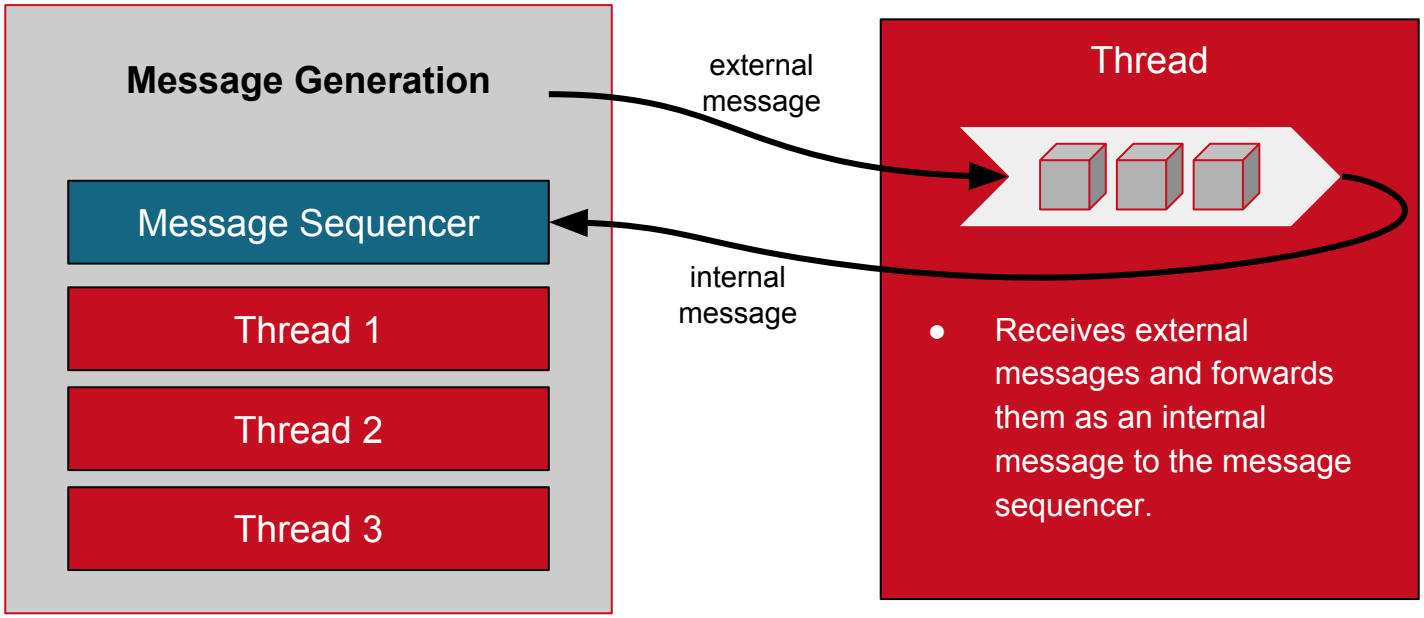
Proposed architecture:



Hints for the Exercise 2

Part 1: Implement a message sequencer for the communication between multiple threads.

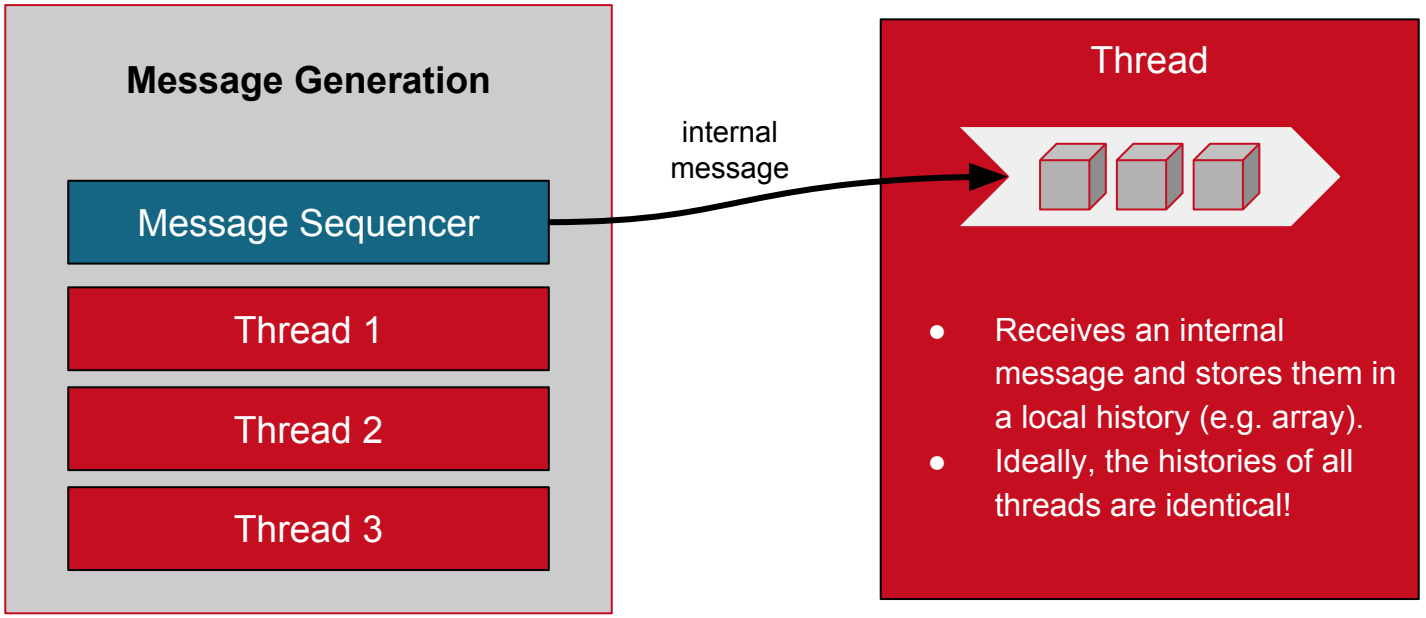
Proposed architecture:



Hints for the Exercise 2

Part 1: Implement a message sequencer for the communication between multiple threads.

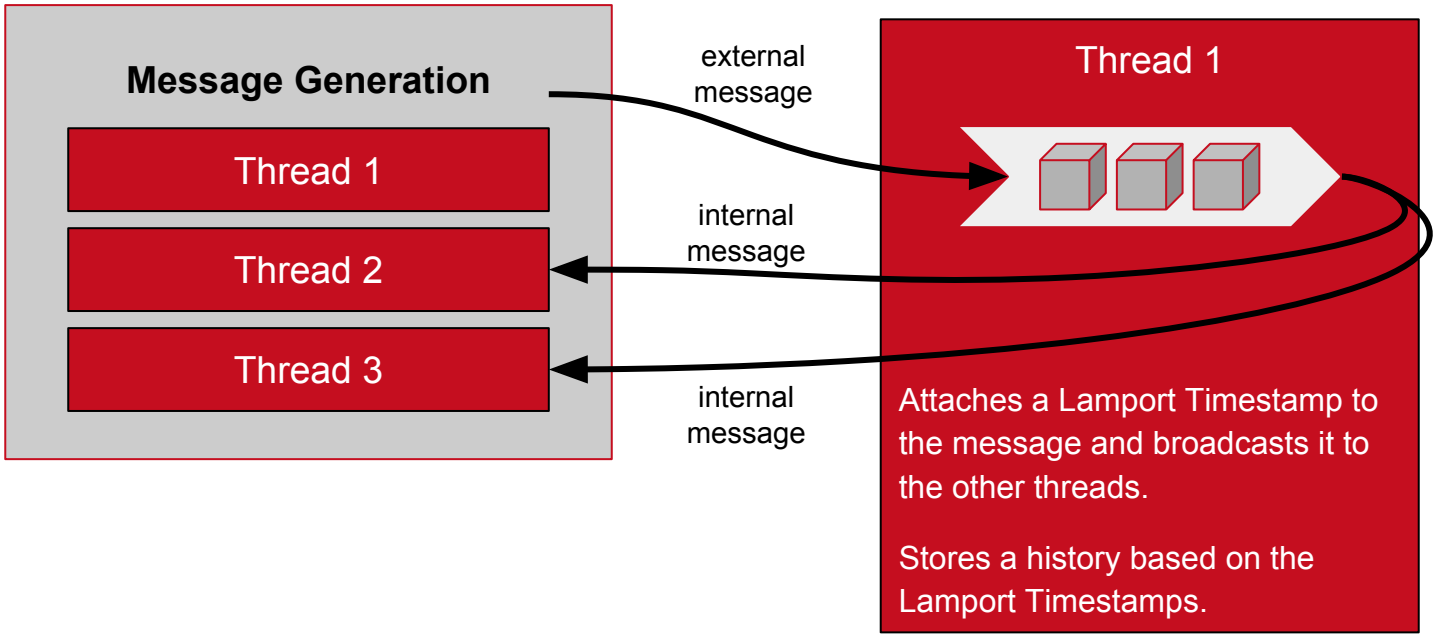
Proposed architecture:



Hints for the Exercise 2

Part 2: Implement Lamport Timestamps for the communication between multiple threads.

Proposed architecture:



Technische Universität Berlin

Final Remarks

- The exercise will be evaluated in the week from the 21th to 25th of May.
- Use the Slack channel for questions and more hints.
- Find a group and choose a time slot for the evaluation on ISIS.
- Register for the exam via QISPOS!

CIT
Complex and Distributed IT Systems