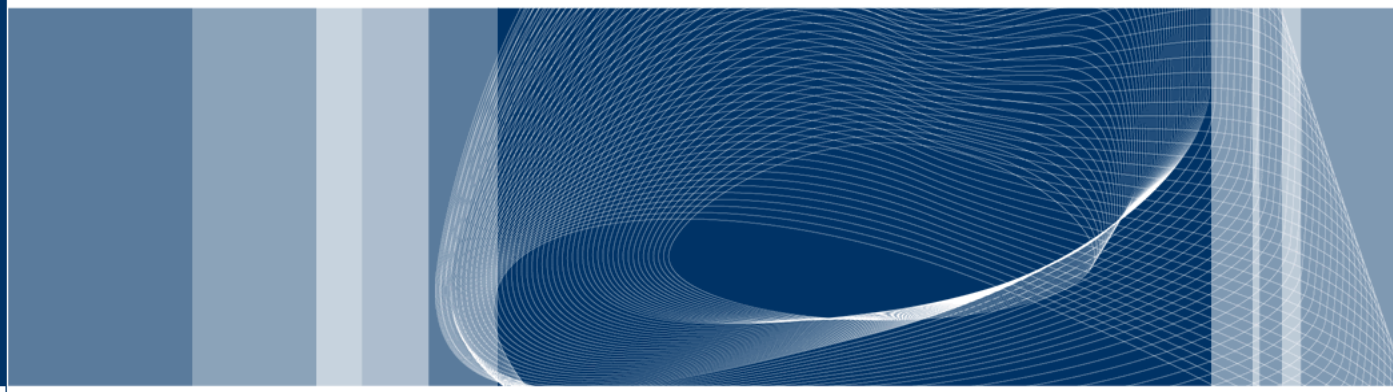
POLITECNICO DI MILANO

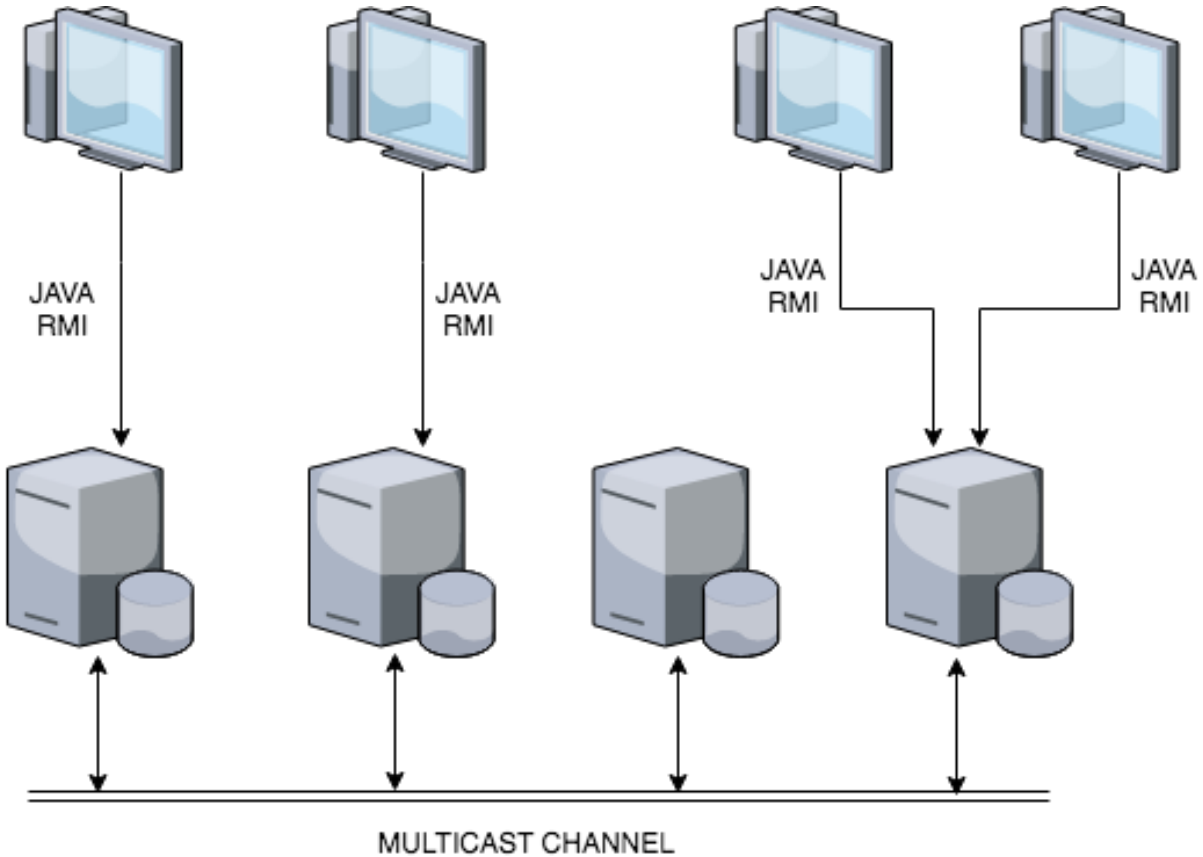


Replicated Data Storage

Luca Santini 883026
Riccardo Remigio 874939

2017-2018

Communication



SEQUENTIAL CONSISTENCY

To provide sequential consistency we have used Lamport Algorithm, that provide Totally ordered multicast, on top of a reliable FIFO channel.

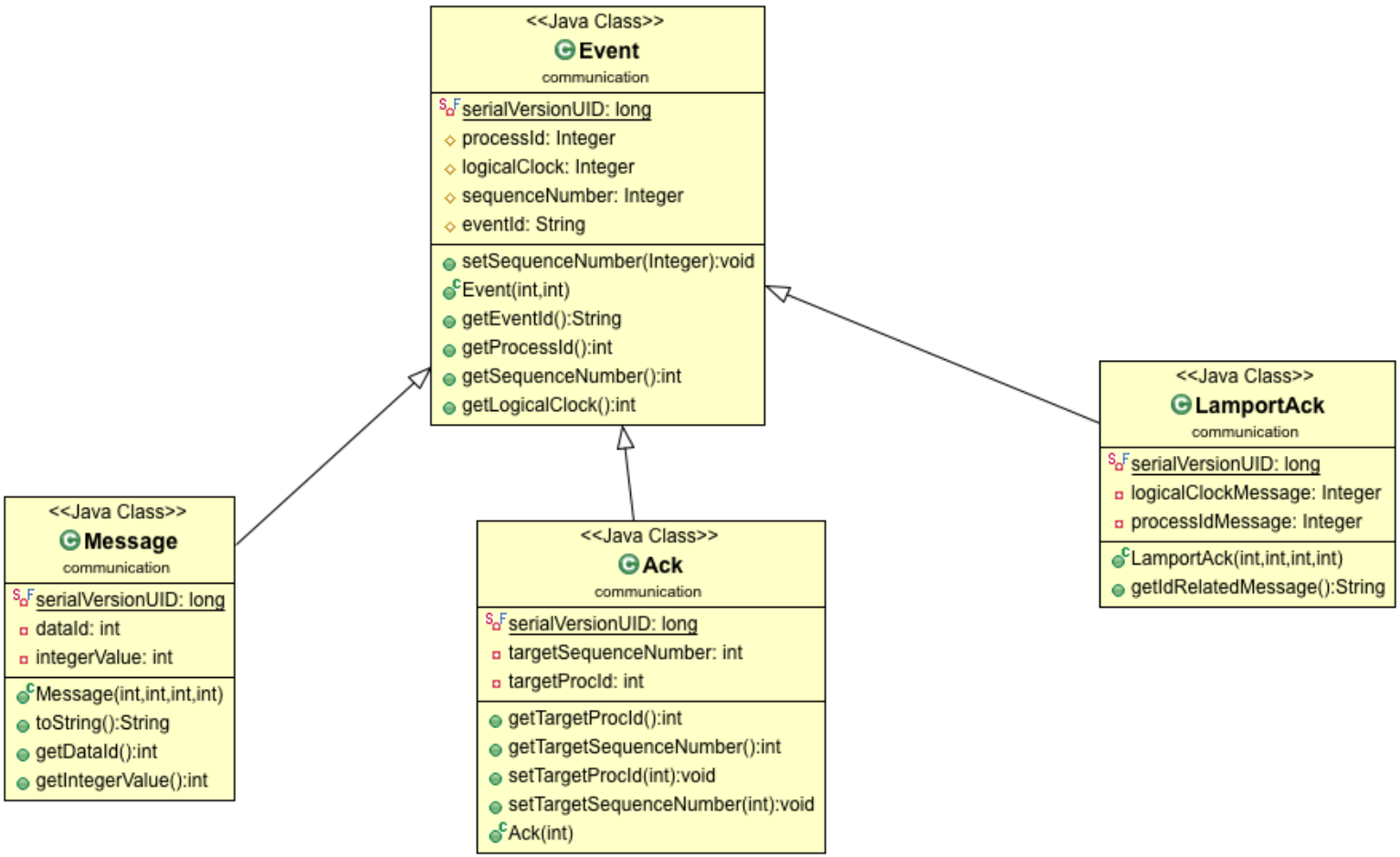
Totally ordered multicast implementation

- Update messages are timestamped with sender's logical clock
- Update messages are multicast
- When a message is received:
  - It is put into local queue
  - Ordered according to timestamp and process id
  - When one of these conditions is satisfied an acknowledgement is multicast:
    - The server has not made an update (with the same logical clock of the message received)
    - The server has made an update and his process id is greater than the message just received
- When a message is on top of the local queue and the server has received all the acknowledgement it writes the update in his database

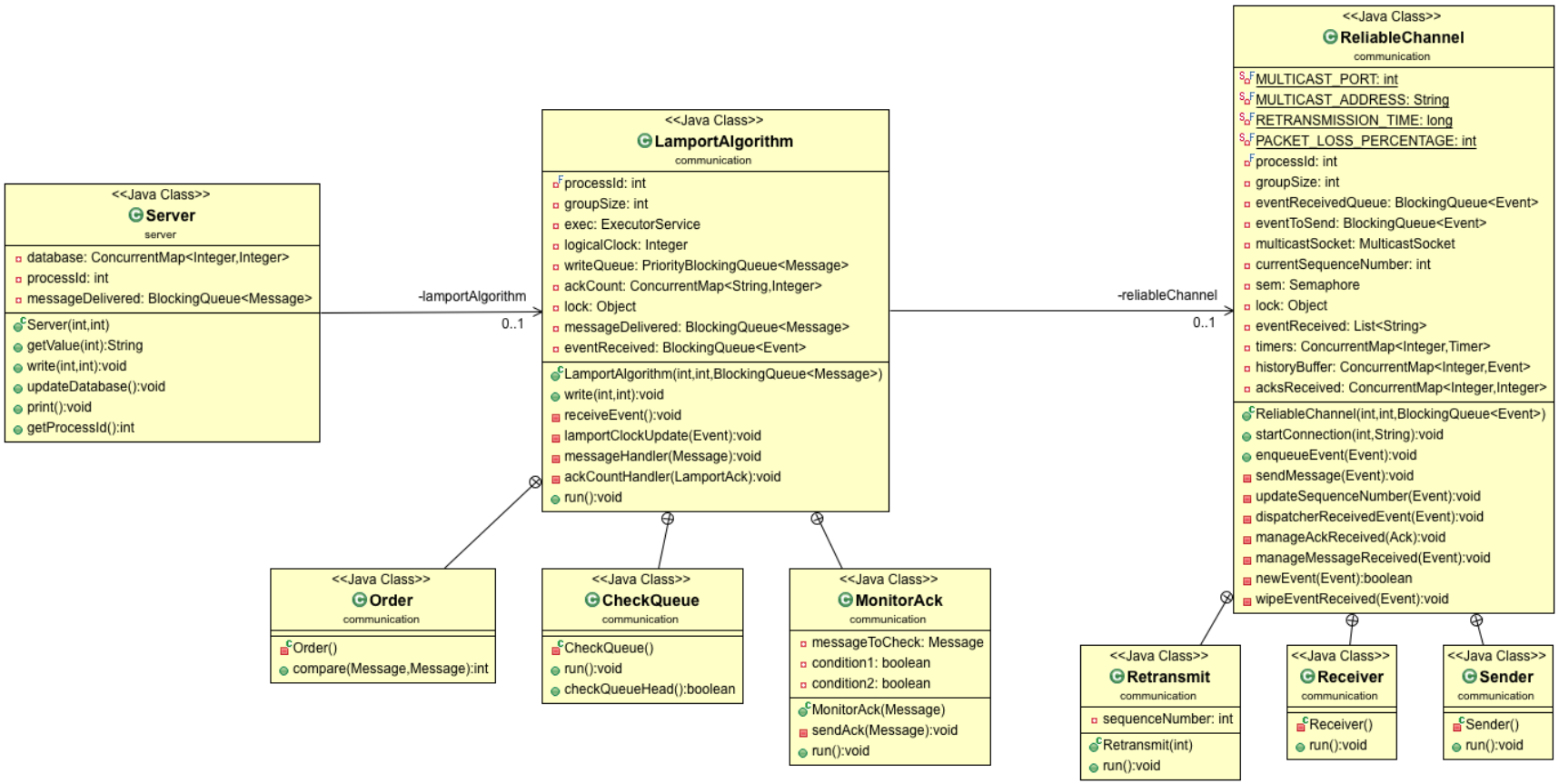
Reliable FIFO Channel implementation

- Messages to send are organized in a local queue
- A sequence number is assigned to the message that has to be sent
- When a process receives a message it sends immediately an acknowledgment
- When sending a message, a retransmission(with a different sequence number) is scheduled after a fixed interval unless all acknowledgments have been received before the time expires
- The next message in the queue is sent only after have been received all the acknowledgements related to the previous message

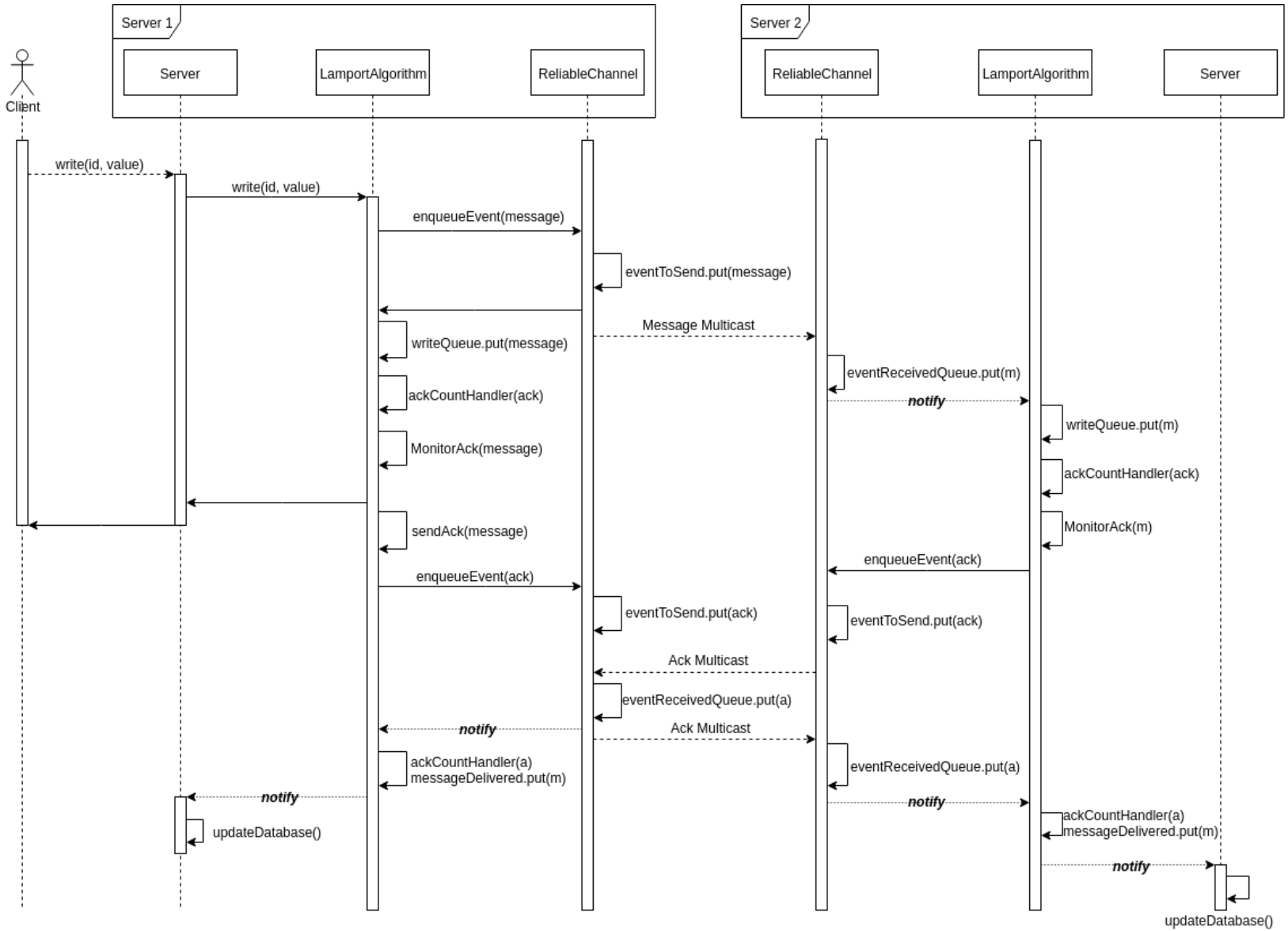
Communication



Server


<<Java Class>>
Server
server
database: ConcurrentMap<Integer,Integer>
processId: int
messageDelivered: BlockingQueue<Message>
Server(int,int)
getValue(int):String
write(int,int):void
updateDatabase():void
print():void
getProcessId():int
-lamportAlgorithm
0..1
<<Java Class>>
LamportAlgorithm
communication
processId: int
groupSize: int
exec: ExecutorService
logicalClock: Integer
writeQueue: PriorityBlockingQueue<Message>
ackCount: ConcurrentMap<String,Integer>
lock: Object
messageDelivered: BlockingQueue<Message>
eventReceived: BlockingQueue<Event>
LamportAlgorithm(int,int,BlockingQueue<Message>)
write(int,int):void
receiveEvent():void
lamportClockUpdate(Event):void
messageHandler(Message):void
ackCountHandler(LamportAck):void
run():void
-reliableChannel
0..1
<<Java Class>>
ReliableChannel
communication
MULTICAST_PORT: int
MULTICAST_ADDRESS: String
RETRANSMISSION_TIME: long
PACKET_LOSS_PERCENTAGE: int
processId: int
groupSize: int
eventReceivedQueue: BlockingQueue<Event>
eventToSend: BlockingQueue<Event>
multicastSocket: MulticastSocket
currentSequenceNumber: int
sem: Semaphore
lock: Object
eventReceived: List<String>
timers: ConcurrentMap<Integer,Timer>
historyBuffer: ConcurrentMap<Integer,Event>
acksReceived: ConcurrentMap<Integer,Integer>
ReliableChannel(int,int,BlockingQueue<Event>)
startConnection(int,String):void
enqueueEvent(Event):void
sendMessage(Event):void
updateSequenceNumber(Event):void
dispatcherReceivedEvent(Event):void
manageAckReceived(Ack):void
manageMessageReceived(Event):void
newEvent(Event):boolean
wipeEventReceived(Event):void
<<Java Class>>
Order
communication
Order()
compare(Message,Message):int
<<Java Class>>
CheckQueue
communication
CheckQueue()
run():void
checkQueueHead():boolean
<<Java Class>>
MonitorAck
communication
messageToCheck: Message
condition1: boolean
condition2: boolean
MonitorAck(Message)
sendAck(Message):void
run():void
<<Java Class>>
Retransmit
communication
sequenceNumber: int
Retransmit(int)
run():void
<<Java Class>>
Receiver
communication
Receiver()
run():void
<<Java Class>>
Sender
communication
Sender()
run():void

Sequence Diagram: Writing to the database



Producer/Consumer