EURECOM
Sophia Antipolis

Coordinating distributed systems part II

Marko Vukolić

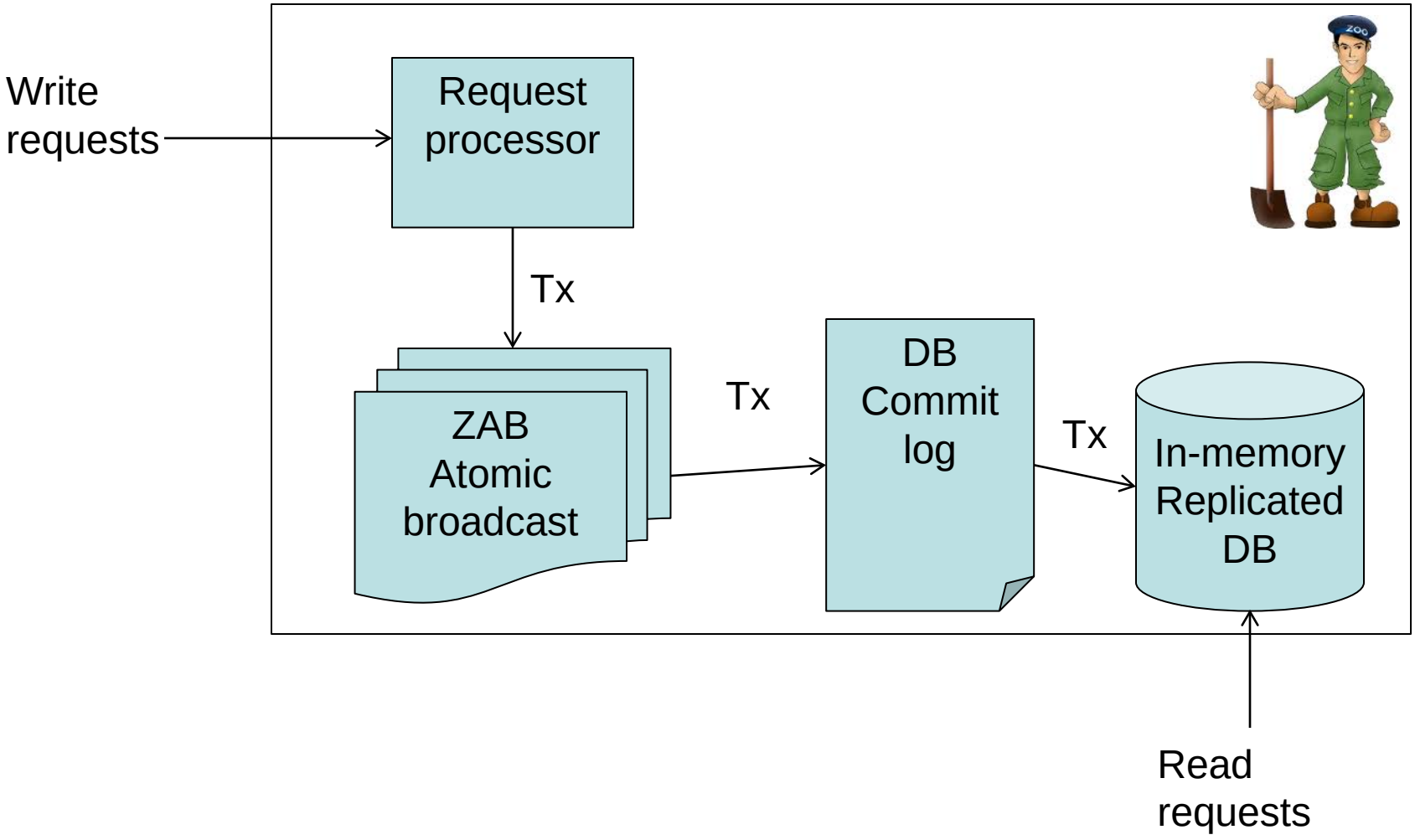
Distributed Systems and Cloud Computing

Last Time

- **Coordinating distributed systems part I**
  - Zookeeper
  - At the heart of Zookeeper is the ZAB atomic broadcast protocol

- **Today**
  - Atomic broadcast protocols
  - Paxos and ZAB
  - Very briefly

Zookeeper components (high-level)



Atomic broadcast

- **A.k.a. total order broadcast**
- **Critical synchronization primitive in many distributed systems**
- **Fundamental building block to building replicated state machines**

Atomic Broadcast (safety)

- **Total Order property**
  - Let m and m' be any two messages.
  - Let pi be any correct process that delivers m without having delivered m'
  - Then no correct process delivers m' before m
- **Integrity (a.k.a. No creation)**
  - No message is delivered unless it was broadcast
- **No duplication**
  - No message is delivered more than once
  - ZAB deviates from this

State machine replication

- **Think of, e.g., a database**
  - Use atomic broadcast to totally order database operations/transactions
- **All database replicas apply updates/queries in the same order**
  - Since database is deterministic, the state of the database is fully replicated
- **Extends to any (deterministic) state machine**

Consistency of total order

- **Very strong consistency**
- **“Single-replica” semantics**

Atomic broadcast implementations

- **Numerous**
- **Paxos [Lamport98, Lamport01] is probably the most celebrated**
- **We will cover the basics of Paxos and compare then to ZAB, the atomic broadcast used in Zookeeper**

Paxos

- **Assume a module that elects a leader within a set of replicas**
  - Election of leader is only eventually reliable
  - For some time multiple processes may believe that they are the leader
- **2f+1 replicas, crash-recovery model**
  - At any given point in time a majority of replicas is assumed to be correct
- **Q: Is Paxos in CP or AP?**

Simplified Paxos

upon tobroadcast(val) by leader

inc(seqno**)**

send [IMPOSE, seqno, val]› to all

upon receive [IMPOSE, seq, v]

myestimates[seq] = v

send ‹[ACK, seq, v]› to ALL

upon receive[ACK, seq, v] from majority and myestimates[seq] = v

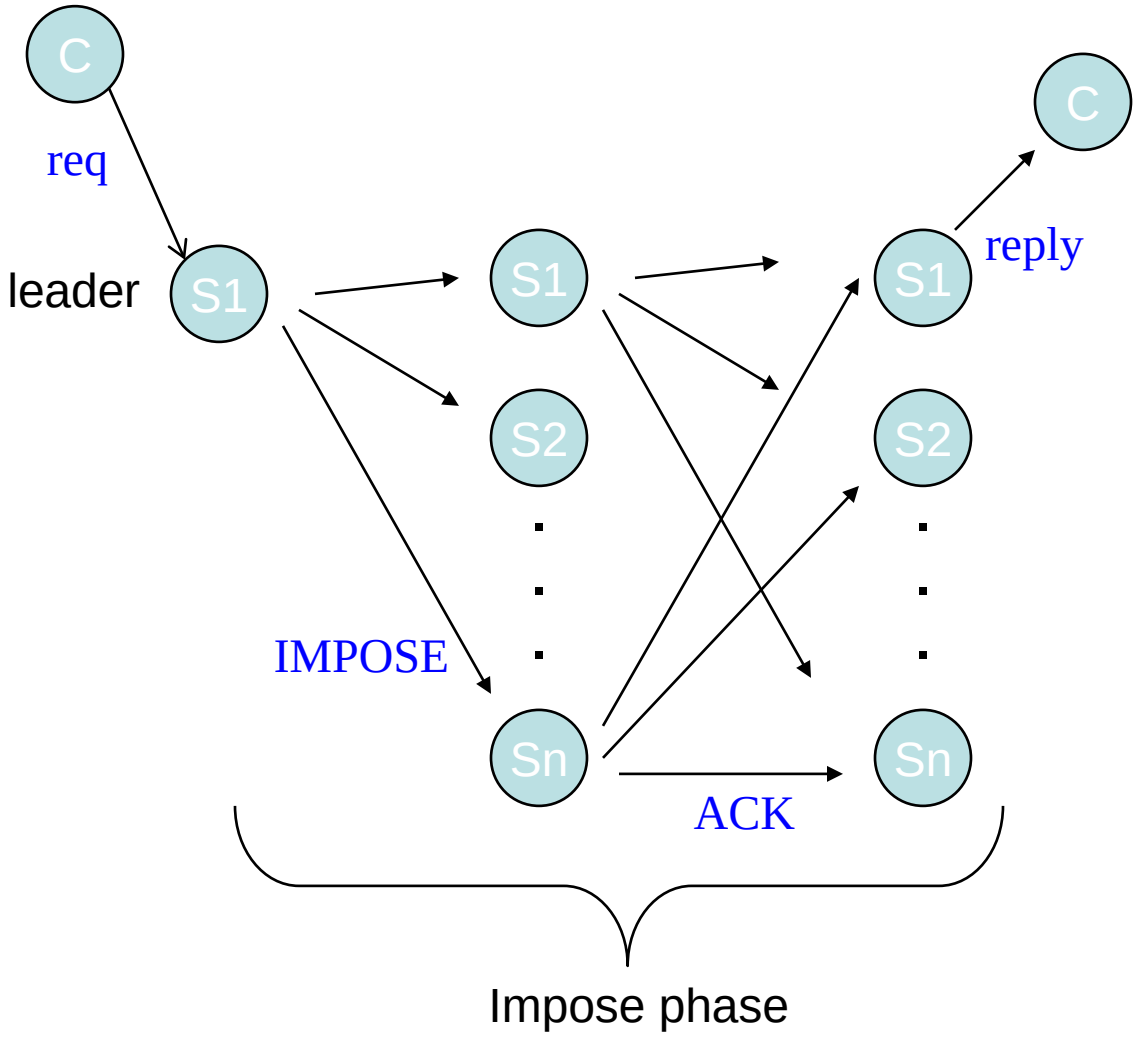
ordered [seq] = v

upon exists sno: ordered[sno]≠nil and delivered[sno]=nil

and forall sno'< sno: delivered[sno']!=nil

delivered[sno] = ordered[sno]

Simplified Paxos Failure-Free Message Flow



Simplified Paxos

- **Works very fine if:**
  - Leader is stable (no multiple processes that believe they are the leader)
  - Leader is correct

- **This will actually be the case most of the time**
  - Yet there will certainly be time when it is not

What if the leader is not stable?

- **Two leaders might compete to propose different commands for the same sequence number**

- **The leader might fail without having completed broadcast**
  - This is dangerous in case of a partition, cannot distinguish from the case where the leader completed its part of broadcast, some replicas already delivered the command whereas others were partitioned

Accounting for multiple leaders

- **Leader failover**
  - New leader must learn what the previous leader imposed
- **Multiple leaders**
  - Need to distinguish among values imposed by different leader
- **To this end we use epoch (a.k.a. ballot) numbers**
  - Assume these are also output by the leader election module
  - Monotonically increasing

Multi-leader Paxos: Impose phase

upon tobroadcast(val) by leader

inc(seqno**)**

send [IMPOSE, seqno, ***epoch,*** val]› to all

upon receive [IMPOSE, seq, ***epoch,*** v]

if lastKnownEpoch <= epoch

myestimates[seq] = ***<v,epoch>***

send ‹[ACK, seq, ***epoch,*** v]› to ALL

upon receive[ACK, seq, ***epoch,*** v] from majority and myestimates[seq] = v

ordered [seq] = v

…

Read phase

- **Need read phase as well**
  - For leader failover
  - New leader must learn what previous leader(s) left over and pick up from there
- **Additional latency**
  - Upside: need to do read phase only once per leader change

Read phase

upon elected leader

send [READ, epoch]

upon receive [READ,epoch] from p

if lastknownEpoch <epoch

lastknownEpoch=epoch

send [GATHER, epoch, myestimates] to p

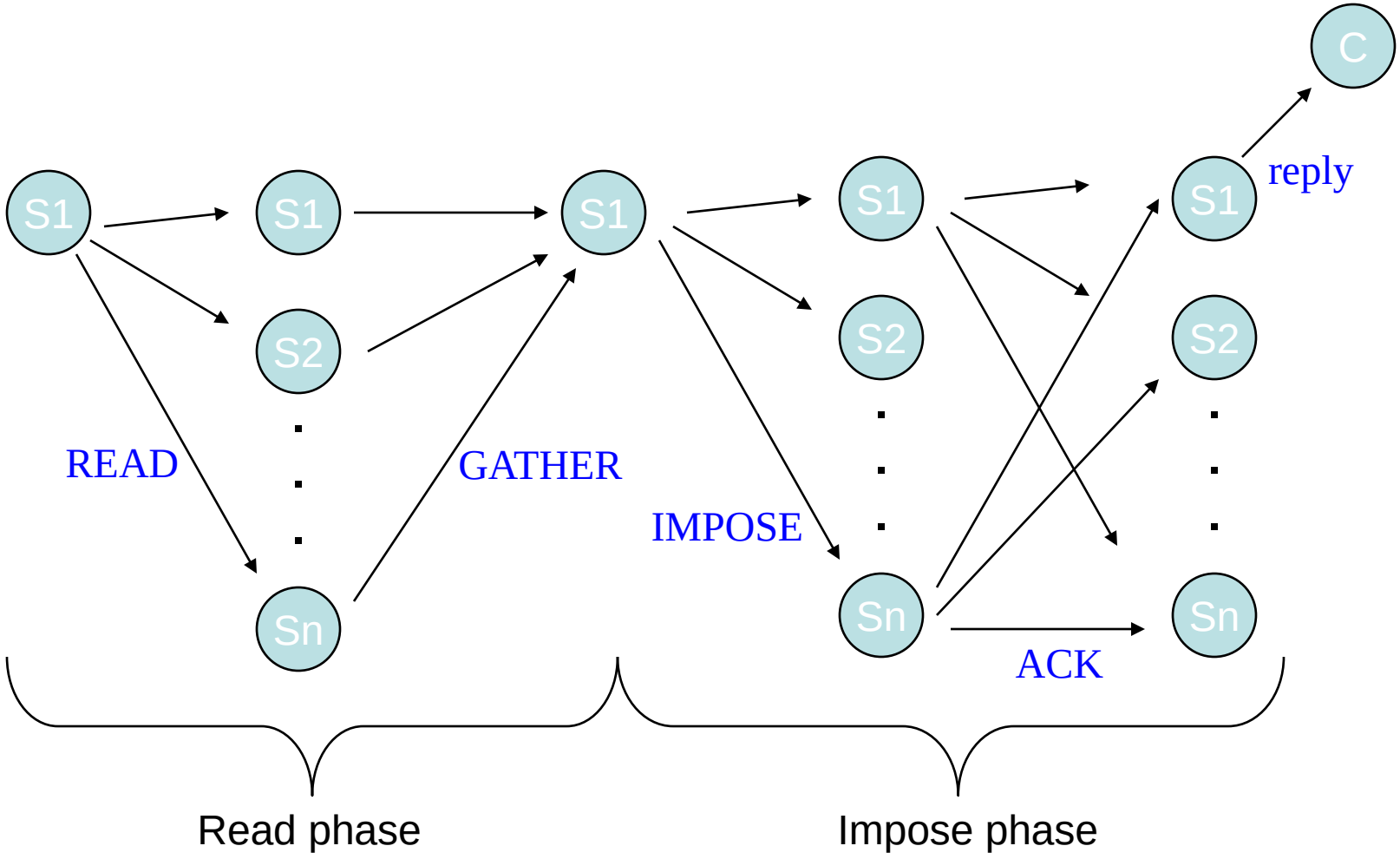
Upon receive GATHER messages from majority (at p)

foreach seqno select the val in myestimates[seqno] with highest epoch number

For other (missing) seqno select noop

proceed to impose phase for all seqno

Paxos Leader failover Message Flow



Paxos

- **This completes high level pseudocode of Paxos**
- **Implements atomic broadcast**
- **Noop fills holes**

Implementing Paxos

- **[Chandra07]**
  - Google Paxos implementation for Chubby lock service
- **Much more difficult to implement Paxos than 2 page pseudocode**
  - “our complete implementation contains several thousand lines of C++ code”

Some of the engineering concerns

- **Crash recovery**
- **Database snapshots**
- **Operator errors**
  - give wrong address of only one node in the cluster → Paxos will mask it but will effectively tolerate f-1 failure
- **Adapting to the higher level spec**
  - In Google case of the Chubby spec
- **Handling disk corruption**
  - Replica is correct but disk is corrupted
- **And a few more…**

Example: Corrupted disks

- **A replica with a corrupted disk rebuilds its state as follows**
  - It participates in Paxos as a non-voting member;
  - meaning that it uses the catch-up mechanism to catch up but does not respond with GATHER/ACK messages
  - It remains in this state until it observes one complete instance of Paxos that was started after the replica started rebuilding its state
  - Waiting for the extra instance of Paxos, ensures that this replica could not have reneged on an earlier promise.

ZAB

- **ZAB is atomic broadcast used in Zookeeper**
  - It is a variant of Paxos
- **Differences**
  - ZAB implements leader order as well
  - Based on the observation that commands proposed by the same leader might have causal dependencies
  - Paxos does not account for this

Leader order

- **Local leader order**
  - If a leader broadcasts a message m before it broadcasts m' then a process that delivers m' must also deliver m before m'
- **Global leader order**
  - Let mi and mj be two messages broadcast as follows:
  - A leader i broadcast mi in epoch ei
  - A leader j in epoch ej>ei broadcasts mj
  - Then, if a process p delivers both mj and mi, p must deliver mi before mj
- **Paxos does not implement leader order**

Leader order and Paxos

- **Assume 26 commands are properly ordered**
- **Assume 3 replicas**
- **A leader l1 starts epoch 126**
  - Learns nothing about commands after 26
  - Imposes A as 27th command and B as 28th command
  - These IMPOSE messages reaches only one replica (l1)
- **Then leader l2 starts epoch 127**
  - Learns nothing about commands after 26
  - Imposes C as 27th command
  - THESE Impose messages reach only l2 and l3

Leader order and Paxos

- **Then leader l3 starts epoch 128**
  - Only l1 and l3 are alive
  - l3 will impose C as 27th command and B as 28th command
  - But l1 did impose A as 27th command before it imposed B as 28th command
  - Leader order violation

- **Sketch these executions**

Further reading (optional)

Flavio Paiva Junqueira, Benjamin C. Reed, Marco Serafini: Zab: High-performance broadcast for primary-backup systems. DSN 2011: 245-256

Tushar Deepak Chandra, Robert Griesemer, Joshua Redstone: Paxos made live: an engineering perspective. PODC 2007: 398-407

Leslie Lamport: Paxos made simple. SIGACT news. (2001)

Leslie Lamport: The Part-Time Parliament. ACM Trans. Comput. Syst. 16(2): 133-169 (1998)

Exerise: Read/Write locks

WriteLock(filename)

```
1:      myLock=create(filename + “/write-”, “”, EPHEMERAL & SEQUENTIAL)
2:      C = getChildren(filename, false)
3:      if myLock is the lowest znode in C then return
4:      else
5:              precLock = znode in C ordered just before myLock
6:              if exists(precLock, true)
7:                      wait for precLock watch
8:              goto 2:
```

Exercise: Read/Write Locks

ReadLock(filename)

```
1:      myLock=create(filename + “/read-”, “”, EPHEMERAL & SEQUENTIAL)
2:      C = getChildren(filename, false)
3:      if no “/write-” znode in C then return
4:      else
5:              precLock = “/write-” znode in C ordered just before myLock
6:              if exists(precLock, true)
7:                      wait for precLock watch
8:              goto 2:
```

Release(filename)

```
        delete(myLock)
```