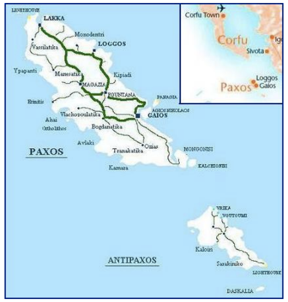
Algorithmic Basics of Blockchains

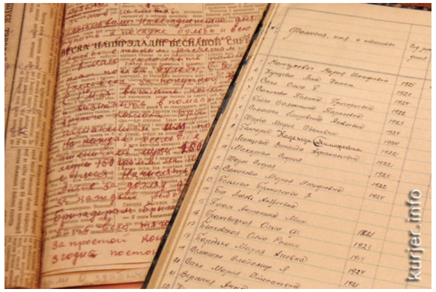


SLR210, P4, 2019

Administrivia

- Language: English. Français sur demande
- Lectures: Fridays (19.04-26.06), 8:30-11:45
- Web page: http://perso.telecom-paristech.fr/~kuznetso/SLR210-2019/
- Project: implementing Paxos (teams by two)
- Office hours (appointments by email)
  - ✓ C213-2, petr.kuznetsov@telecom-paristech.fr
  - ✓ C213-3, matthieu.rambaud@telecom-paristech.fr
- Credit = 0.7*written exam+0.3*project, reports to be submitted by 12.04
  - ✓ Bonus for participation/discussion of exercises
  - ✓ Bonus for bugs found in slides/lecture notes

Blockchain: expectations



- Ledger
    - Record of operations
- Public
    - Can be read/modified by all parties
- Decentralized
    - No trusted party
- Tamper-proof
    - No party can modify a recorded operation

Blockchain: chronology

1982 Byzantine Generals

1990 Paxos/Storage

1992 "ProofOfWork"

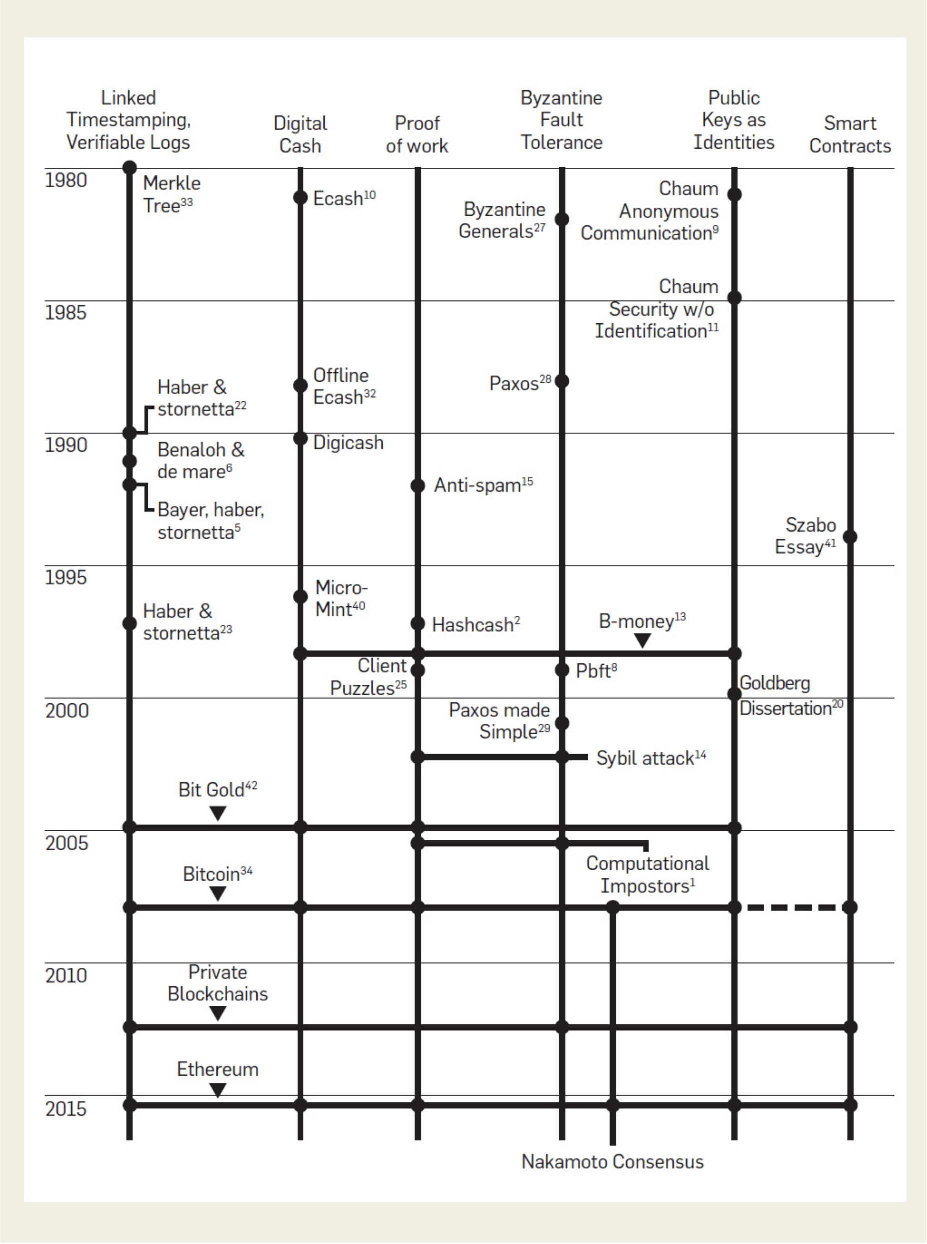
1999 PBFT

1995 Hashcash

2002 Sybil attack

2009 Bitcoin

…



Roadmap

- Storage systems and lattices
- CAP theorem
- State machine replication and Paxos
- Byzantine agreement
- Practical Byzantine fault-tolerance
- Permissioned Blockchains
  - Hyperledger
- Permissionless blockchain
  - Bitcoin/PoW
  - Ethereum/Smart Contracts
  - Casper/PoS

Communication models

- Shared memory
  - ✓Processes apply operations on shared variables
  - ✓Failures and asynchrony

- Message passing
  - ✓Processes send and receive messages
  - ✓Communication graphs
  - ✓Message delays

So far…

Shared-memory computing:

- Wait-freedom and linearizability
- Lock-based and lock-free synchronization
- Consensus and universality

Message-passing

- Consider a network where every two processes are connected via a reliable channel
  - ✓no losses, no creation, no duplication

- Which shared-memory results translate into message-passing?

Read-write register

- Stores *values* (in a *value set* V)
- Exports two operations: read and write
  - ✓Write takes an argument in V and returns ok
  - ✓Read takes no arguments and returns a value in V

Space of registers

- Values: from binary (V={0,1}) to multi-valued
- Number of readers and writers: from 1-writer 1-reader (1W1R) to multi-writer multi-reader (NWNR)
- Safety criteria: from safe to atomic

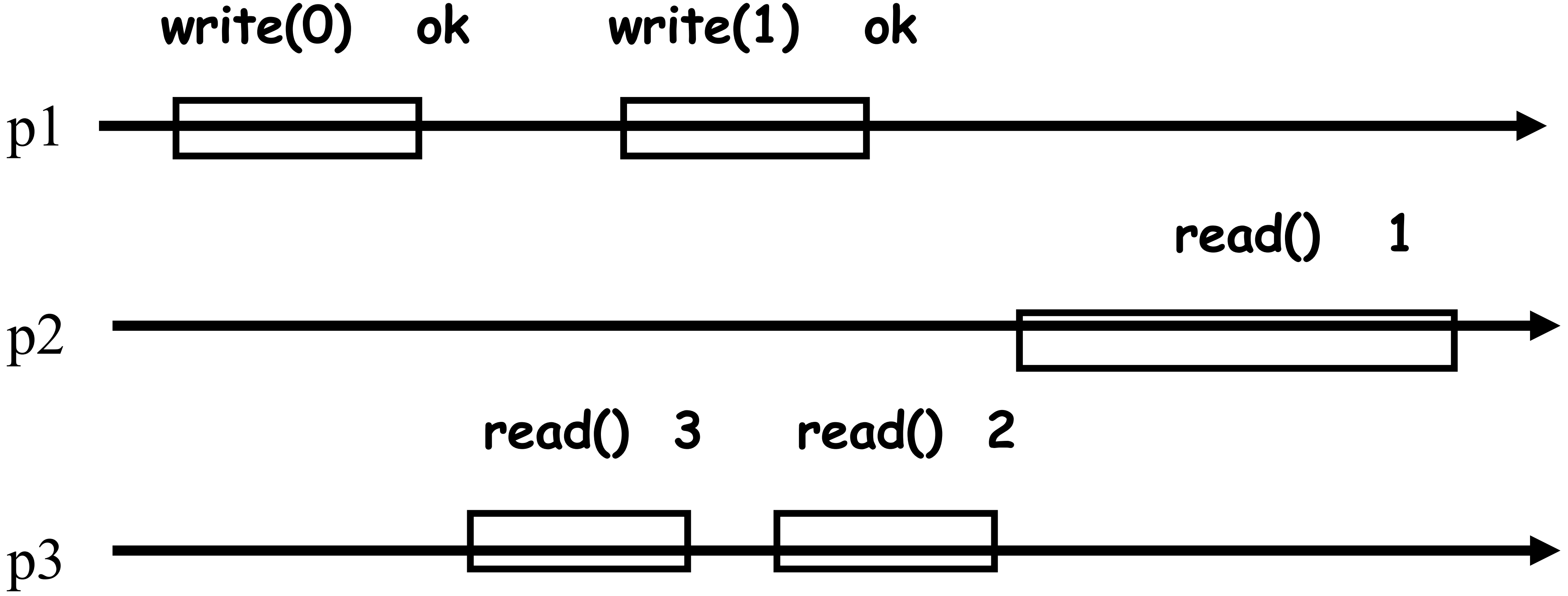
Safety criteria

- Safe registers: every read that does not overlap with a write returns the last written value
- Regular registers: every read returns the last written value, or the concurrently written value

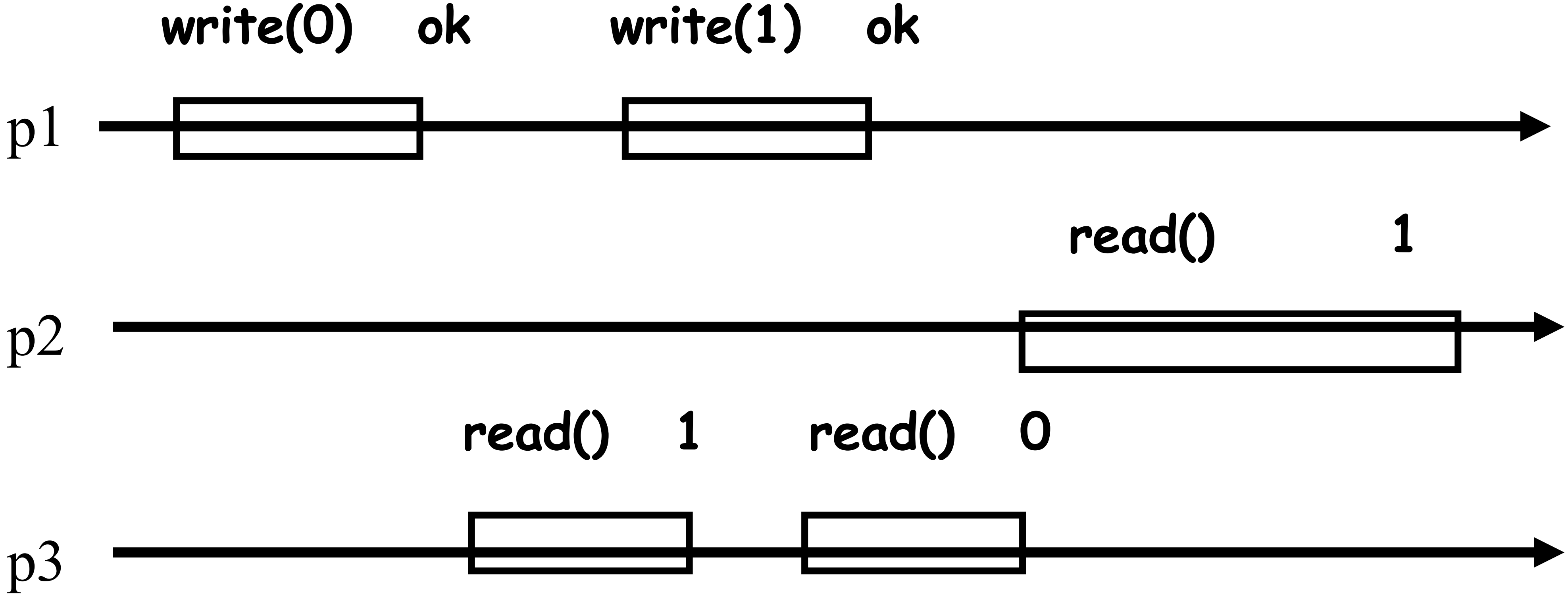
  (assuming one writer)

- Atomic registers: the operations can be totally ordered, preserving legality and precedence (linearizability)
  - ✓≈ if read1 returns v, read2 returns v’, and read1 precedes read2, then write(v’) cannot precede write(v)

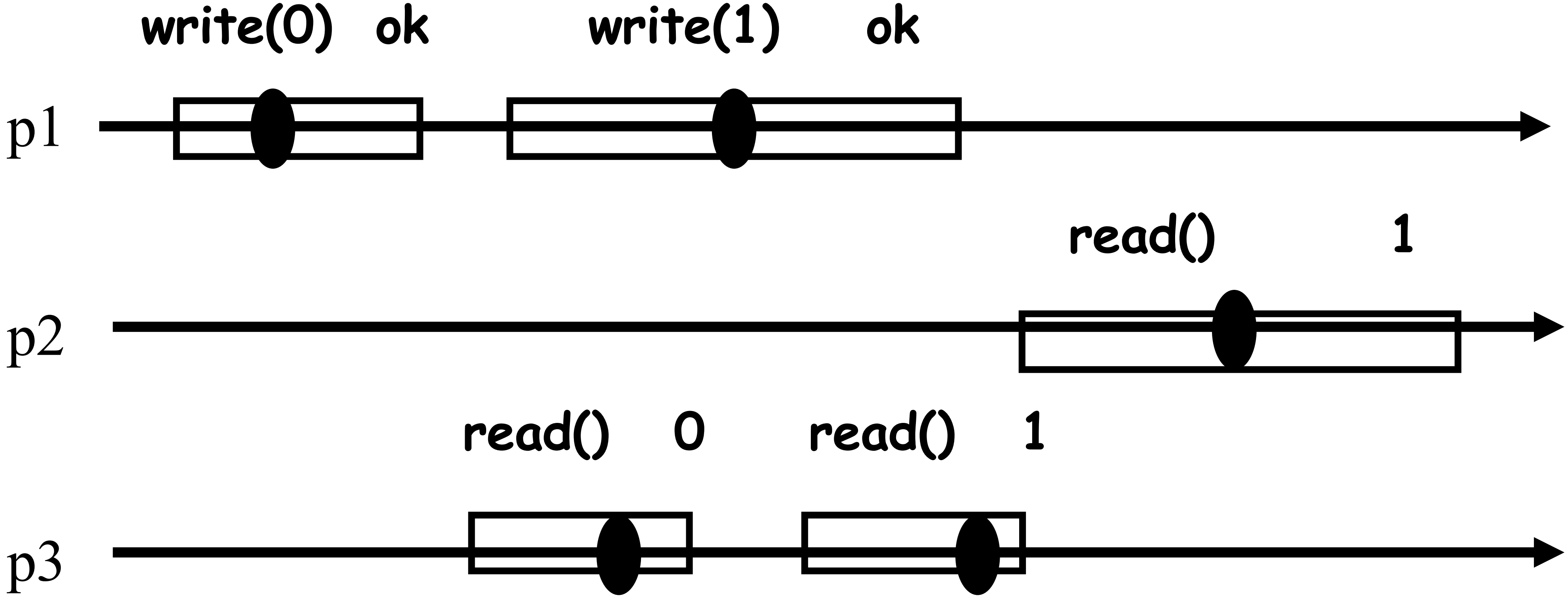
Safe register



Regular register



Atomic register



Space of registers

- Values: from binary ($V=\{0,1\}$) to multi-valued
- Number of readers and writers: from 1-writer 1-reader (1W1R) to multi-writer multi-reader (NWNR)
- Safety criteria: from safe to atomic

1W1R binary safe registers can be used to implement

an NWNR multi-valued atomic registers!

Implementing message-passing

Theorem 1 A reliable message-passing channel between two processes can be implemented using two one-writer one-reader (1W1R) read-write registers

Corollary 1 Consensus is impossible to solve in an asynchronous message-passing system if at least one process may crash

ABD algorithm: implementing shared memory

Theorem 2[ABD] A 1W1R atomic register can be implemented in a (reliable) message-passing model where a majority of processes are correct

- Every process is a replica of the implemented register

Implementing a 1W1R register

```
Upon write(v)
  t++
  send [v,t] to all
  wait until received [ack,t] from a majority
  return ok

Upon read()
  r++
  send [?,r] to all
  wait until received {(t',v',r)} from a
  majority
  return v' with the highest t'
```

Implementing a 1W1R register, contd.

```
Upon receive [v,t]
  if $t>t_i$ then
        $v_i$ := v
        $t_i$ := t
        send [ack,t] to the writer

Upon receive [?,r]
  send [$v_i$,$t_i$,r] to the reader
```

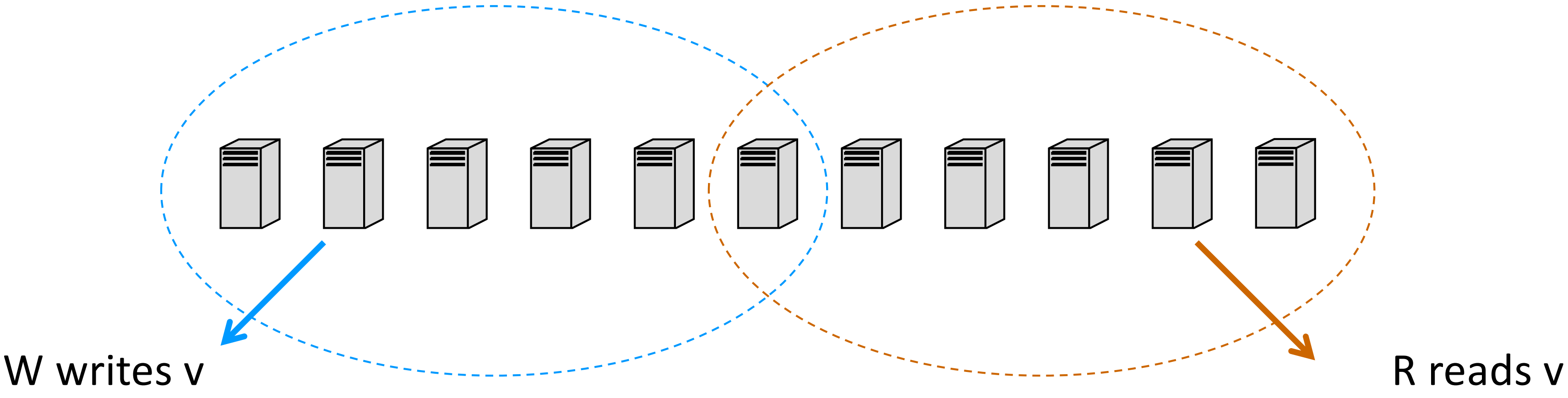
Quiz 1

- Show that the ABD algorithm executed by one writer and multiple readers implements a regular but not atomic register
- Turn the algorithm into an atomic 1WNR one
- An atomic NWNR?

A correct majority is necessary

Otherwise, the reader may miss the latest written value

The quorum (set of involved processes) of any write operation must intersect with the quorum of any read operation:



Quorum systems

Let P be the set of processes

A quorum system on P is a tuple $(WP, RP), W_P, RP \in 2^P$

Safety:

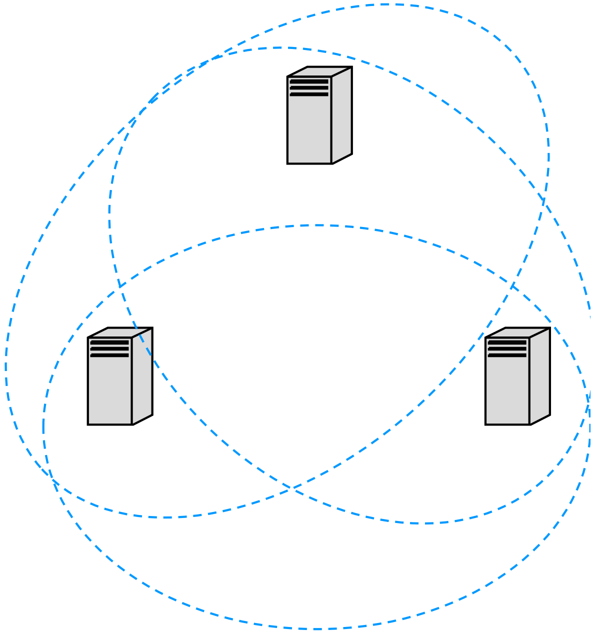
- $\forall W \in W_P, \forall R \in R_P: W \cap R \neq \emptyset$

For example, t-resilient n-process, t<n/2

$W_P = R_P = \{S \in 2^P : |S| = n - t\}$

Liveness:

- Some $W \in W_P$, $R \in R_P$ contains only correct processes



Implementing a 1W1R register

```
Upon write(v)
  t++
  send [v,t] to all
  wait until received [ack,t] from a write quorum
  return ok

Upon read()
  r++
  send [?,r] to all
  wait until received {(t',v',r)} from a read quorum
  return v' with the highest t'
```

Quiz 2

- For a fault-free system, design a read-optimized quorum system:
  - ✓A read operation involves a single replica
- For a t-resilient system, design a quorum system ensuring a stronger property
  - ✓$\forall W \in W_P, \forall R \in R_P$: $W \cap R$ contains at least one correct process

Beyond reads and writes: lattices

Imagine a lattice partial order $(L, \sqsubseteq, \sqcup)$

- L is a set of value
- $\sqsubseteq$ partial order on L
- $\sqcup$ join (least upper-bound) operator on L:

$$\forall U \subseteq L, \bigsqcup V = \min\{u : \forall v \in V, v \sqsubseteq u\}$$

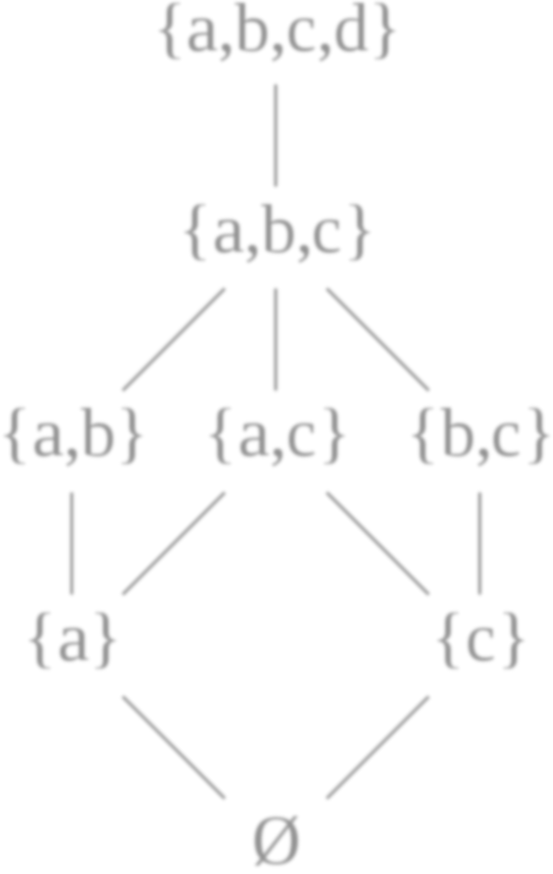
We also assume the origin element u_0:

$$\forall u \in L : u_0 \sqsubseteq u$$

Beyond reads and writes: lattices

$$(L, \sqsubseteq, \sqcup)$$

- $L = \{abcd, abc, ab, ac, bc, a, c, \emptyset\}$
- $\sqsubseteq$ - inclusion $\subseteq$
- $\sqcup$ - union $\cup$
- $\emptyset$ - origin



Beyond reads and writes: Lattice agreement

Every process i proposes $u_i \in L$ and decides on $v_i \in L$:

- Comparability: $\forall i, j\colon vi \sqsubseteq v_j \vee v_j \sqsubseteq v_i$
- Validity: $\forall i\colon\ v_i \sqsubseteq \bigsqcup_j u_j$
- Monotonicity: $\forall i\colon\ u_i \sqsubseteq v_i$
- Liveness: every correct process eventually decides

Atomic snapshot: sequential specification

- Each process p_i is provided with operations:
  - ✓ $update_i(v)$, returns ok
  - ✓ $snapshot_i()$, returns $[v_1,\ldots,v_N]$

- In a sequential execution:

  For each $[v_1,\ldots,v_N]$ returned by $snapshot_i()$,
  v_j $(j=1,\ldots,N)$ is the argument of the last $update_j(.)$
  (or the initial value if no such update)

One-shot atomic snapshot (AS)

Each process p_i:

update$_i(v_i)$

S_i := snapshot()

$S_i = S_i[1],\ldots,S_i[N]$

(one position per process)

Vectors S_i satisfy:

- Self-inclusion: $\forall i$: $v_i \in S_i$
- Containment: $\forall$ i, j: $S_i \subseteq S_j \vee S_j \subseteq S_i$

Quiz 3

In a read-write shared memory model:

- Show that Lattice Agreement (LA) is equivalent to one-shot atomic snapshot (1AS)
  - ✓ Find the matching lattice and propose two-way wait-free transformations
    - 1AS $\Leftrightarrow$ LA

Generalized lattice agreement

Every process p receives values $u_p{}^i \in L$ and learns values on $v_p{}^i \in L$ $(i=1,2,\ldots)$:

- Comparability: $\forall p, q, i, j\colon v_p{}^i \sqsubseteq v_q{}^j \vee v_q{}^j \sqsubseteq v_p{}^i$
- Validity: $\forall p, i\colon v_p{}^i \sqsubseteq \bigsqcup_{q,j} u_q{}^j$
- Monotonicity: $\forall p, i < j\colon v_p{}^i \sqsubseteq v_p{}^j$
- Liveness: every value received by a correct process p is eventually learned by every correct process q: $\exists j, u_p{}^i \sqsubseteq v_q{}^j$

Using GLA

Natural for objects with reads and commuting updates

- Reads return the state without modifying
- Updates commute: s.u1.u2=s.u2.u1
- E.g., add-only set (add and contains), counter (inc and read)

Quiz 4

In a read-write shared memory model:

- Show that Generalized Lattice Agreement (GLA) is equivalent to (long-lived) atomic snapshot (AS)
  - ✓Find the matching lattice and propose two-way wait-free transformations
    - AS ⇔ GLA

Universal construction with GLA

```
Upon Update(cmd)
  ReceiveValue({cmd})
  wait until cmd ∈ LearntValue()

Upon Read()
   Update(noop)
  // does not modify the state
  return Apply(LearntValue())
```

Linearizable update-commutable object

Implementing GLA

```
Local variables:
  bufferedValues = {}
  proposedValue = origin
  learnValue = origin
  acceptedValue = origin

Upon ReceiveValue(v) // process p
  t++ // sequence number of the proposal
  bufferedValues = bufferedValues ⊔ {v}
  send proposal(v,t,p) to all

Upon Learn()
  return learntValue
```

Implementing GLA (contd.)

```
Upon received [nack,val,t,p]
// t – seq num of the current proposal
  proposedValue = proposedValue ⊔ val

Upon received >N/2 [ack/nack,*,t’,p’]
  if no [nack,*,t’,p’] received then
        if learntVaue ⊏ v then LearntValue = v
        // learn a new value
  else if p’ = p and t’ = t then
        // responses to the current proposal
        t++
        send proposal(proposedValue,t,p) to all
        // send a new proposal
```

Implementing GLA (contd.)

```
Upon received proposal(v',t',p')
  if acceptedValue $\sqsubseteq$ v' then
        acceptedValue = v'
        send [ack,v',t',p'] to all
        // accept the proposal
  else
        acceptedValue = acceptedValue $\sqcup$ v'
        send [nack,acceptedValue,t',p'] to p'
        // reject the proposal
```

GLA implementation: correctness

Safety

- Validity & Monotonicity -> immediate
- Comparability:
  - ✓any learnt value is accepted by a majority of processes
  - ✓only comparable values are accepted

Liveness

- ✓Check

Literature

- C. Cachin, R. Guerraoui, L. Rodrigues. Introduction to Reliable and Secure Distributed Programming. Springer, 2011
- N. Lynch. Distributed Algorithms. Morgan Kaufmann Publishers. 1996
- H. Attiya, A. Bar-Noy, D. Dolev: Sharing Memory Robustly in Message-Passing Systems. J. ACM 42(1): 124-142 (1995)
- H. Attiya, M. Herlihy, O. Rachman: Atomic Snapshots Using Lattice Agreement. Distributed Computing 8(3): 121-132 (1995)
- J. M. Falerio, S. K. Rajamani, K. Rajan, G. Ramalingam, K. Vaswani: Generalized lattice agreement. PODC 2012: 125-134