Combinatorial Structures for Distributed Computing

Petr Kuznetsov, Telecom ParisTech

Kyoto University, 2018

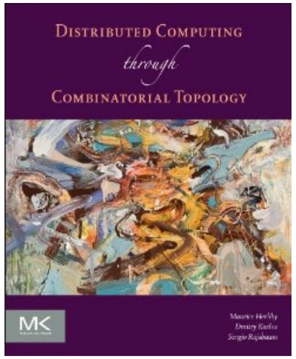
Roadmap

- Distributed computing primer
  - ✓Read-write memory basics
  - ✓IIS model and iterated subdivisions
  - ✓Distributed tasks, consensus, set consensus
- Combinatorial topology for distributed computing
  - ✓Asynchronous Computability Theorem for colorless tasks
  - ✓Adversarial models and general tasks

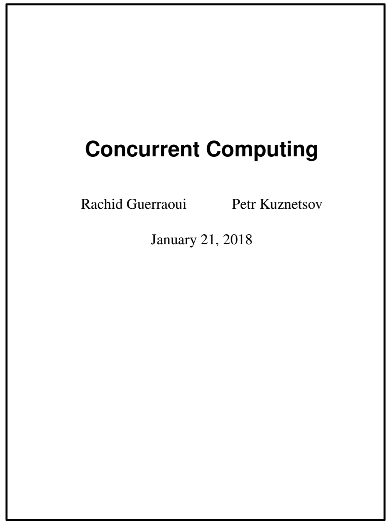
Slides and exercises:
https://perso.telecom-paristech.fr/kuznetso/Kyoto2018

© 2018 P. Kuznetsov

Literature



Distributed Computing
Through Combinatorial Topology
Maurice Herlihy, Dmitry Kozlov, Sergio Rajsbaum
Morgan Kaufman, 2013, available online (TPT library


Concurrent Computing

Rachid Guerraoui Petr Kuznetsov

January 21, 2018

Lecture notes on Concurrent Computing
R. Guerraoui, P. Kuznetsov, 2018 (constantly under construction)

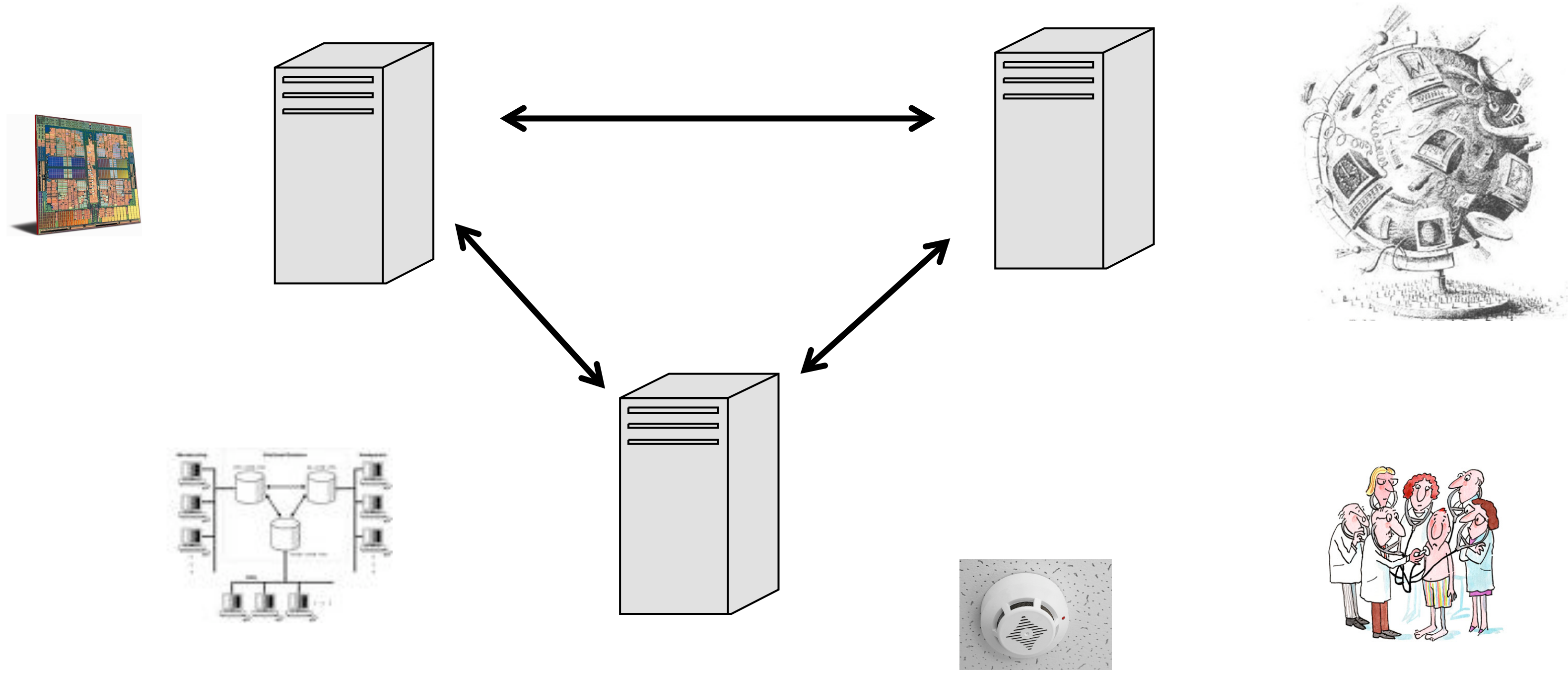
- Distributed Computing: Fundamentals, Simulations and Advanced Topics H. Attiya, J. Welch. (2nd edition). Addison Wesley. 2006
- Distributed Algorithms. N. Lynch. Morgan Kaufmann Publishers. 1996

This course is about distributed computing:

independent sequential processes that communicate

© 2018 P. Kuznetsov

Concurrency is everywhere!



- Multi-core processors
- Sensor networks
- Internet
- …

© 2018 P. Kuznetsov

Communication models

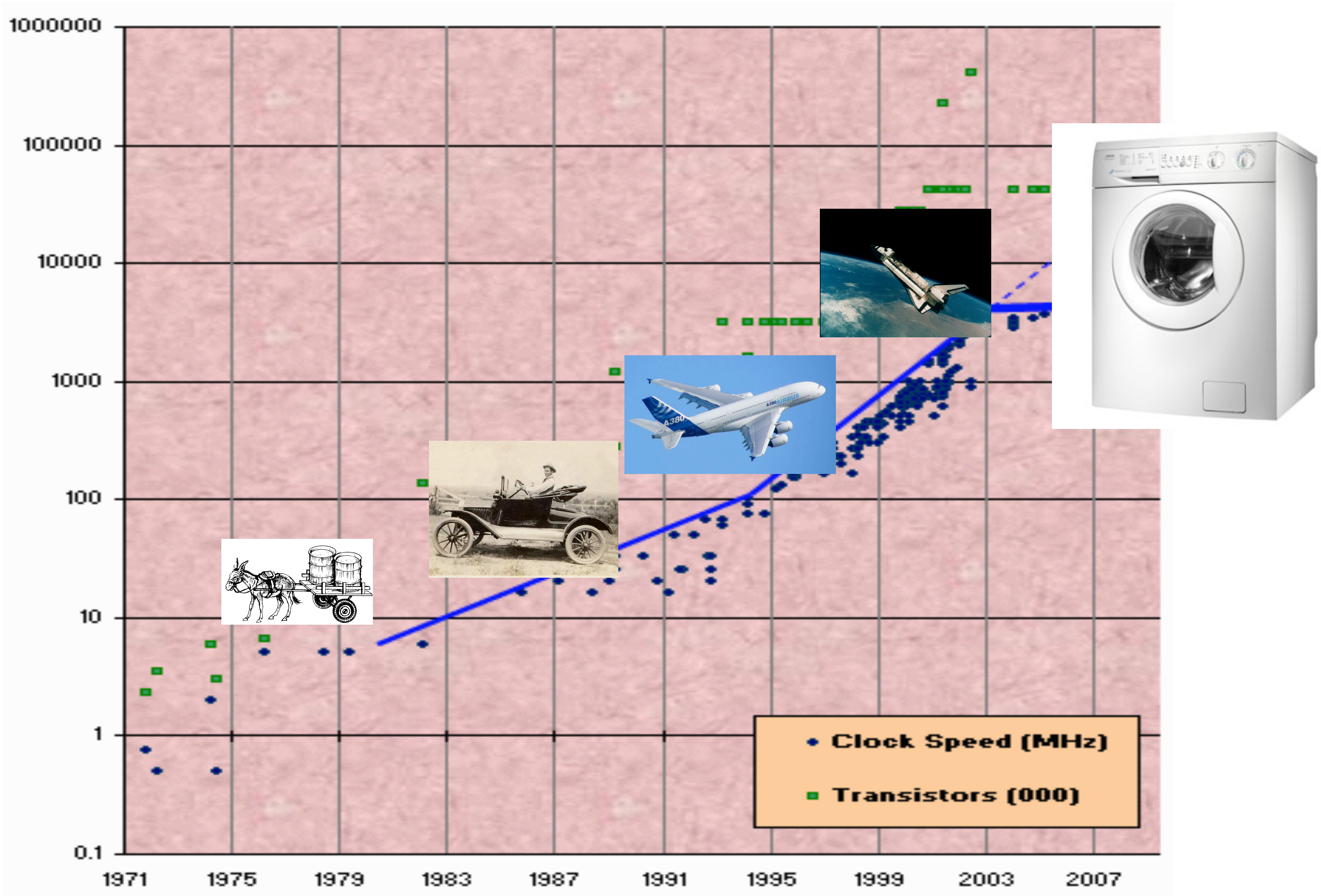
- Shared memory
  - ✓Processes apply operations on shared variables
  - ✓Failures and asynchrony

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

TELECOM ParisTech

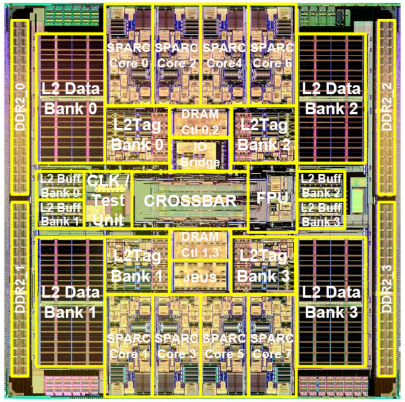
© 2018 P. Kuznetsov

Moore's Law and CPU speed



© 2018 P. Kuznetsov

- Single-processor performance does not improve
- But we can add more cores
- Run concurrent code on multiple processors



Can we expect a proportional speedup? (ratio between sequential time and parallel time for executing a job)

© 2018 P. Kuznetsov

Amdahl's Law

- p – fraction of the work that can be done in parallel (no synchronization)
- n - the number of processors
- Time one processor needs to complete the job = 1

$$S = \frac{1}{1 - p + p / n}$$

Challenges

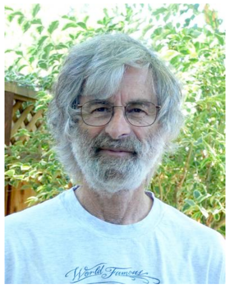
- What is a correct implementation?
  - ✓Safety and liveness
- What is the cost of synchronization?
  - ✓Time and space lower bounds
- Failures/asynchrony
  - ✓Fault-tolerant concurrency?
- How to distinguish possible from impossible?
  - ✓Impossibility results

© 2018 P. Kuznetsov

Distributed ≠ Parallel

- The main challenge is synchronization

- “you know you have a distributed system when the crash of a computer you’ve never heard of stops you from getting any work done” (Lamport)



History

- Dining philosophers, mutual exclusion (Dijkstra)~60's
- Distributed computing, logical clocks (Lamport), distributed transactions (Gray) ~70's
- Consensus (Lynch) ~80's
- Distributed programming models, since ~90's
- Link b/w distributed computing and topology, 90's
- Multicores and large-scale distributed services now

TELECOM ParisTech

Synchronization jungle

Transactional memory

t-resilience

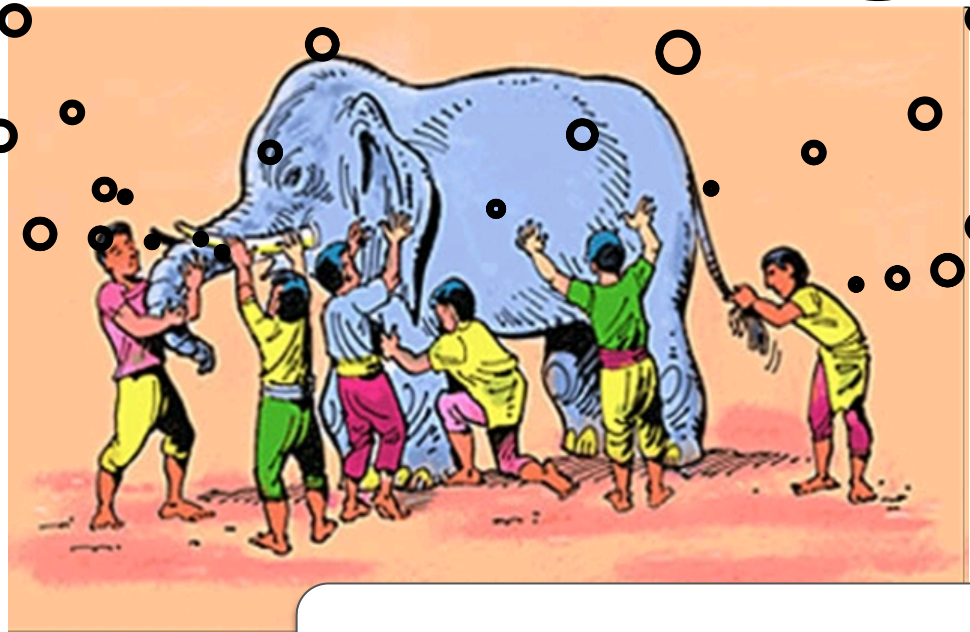
Message-passing

CAS

Shared-memory

Locks

Adversaries



Which features matter?

Matter for what?

Combinatorial Structures for Distributed Computing

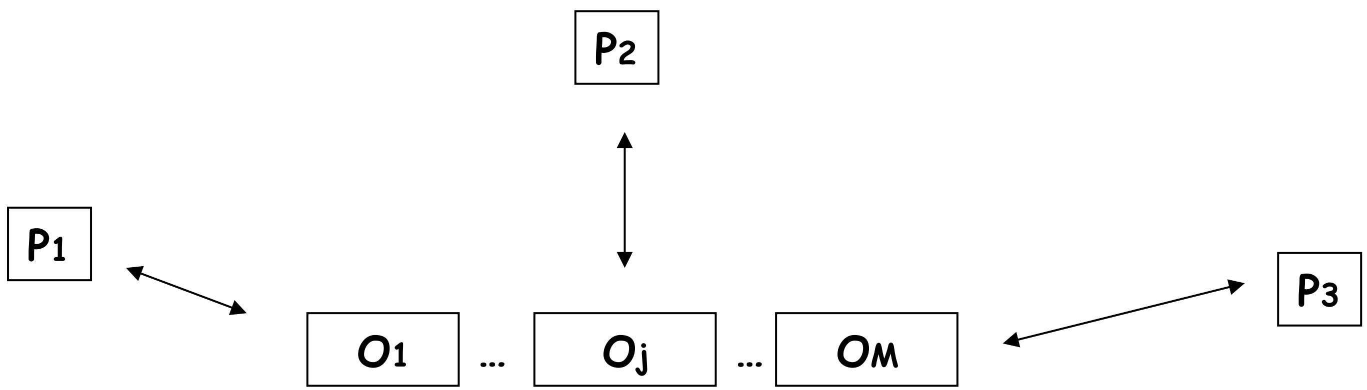
Shared memory basics

Kyoto University, 2018

© 2018 P. Kuznetsov

Shared memory model

- Processes communicate by applying operations on and receiving responses from *shared objects*
- A shared object is a state machine
  - ✓States
  - ✓Operations/Responses
  - ✓Sequential specification
- Examples: read-write registers, TAS,CAS,LLSC,…



Read-write registers

- 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

We assume that registers are atomic: operations take place in indivisible instants

TELECOM ParisTech

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)

Can be implemented from atomic registers!

© 2018 P. Kuznetsov

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)

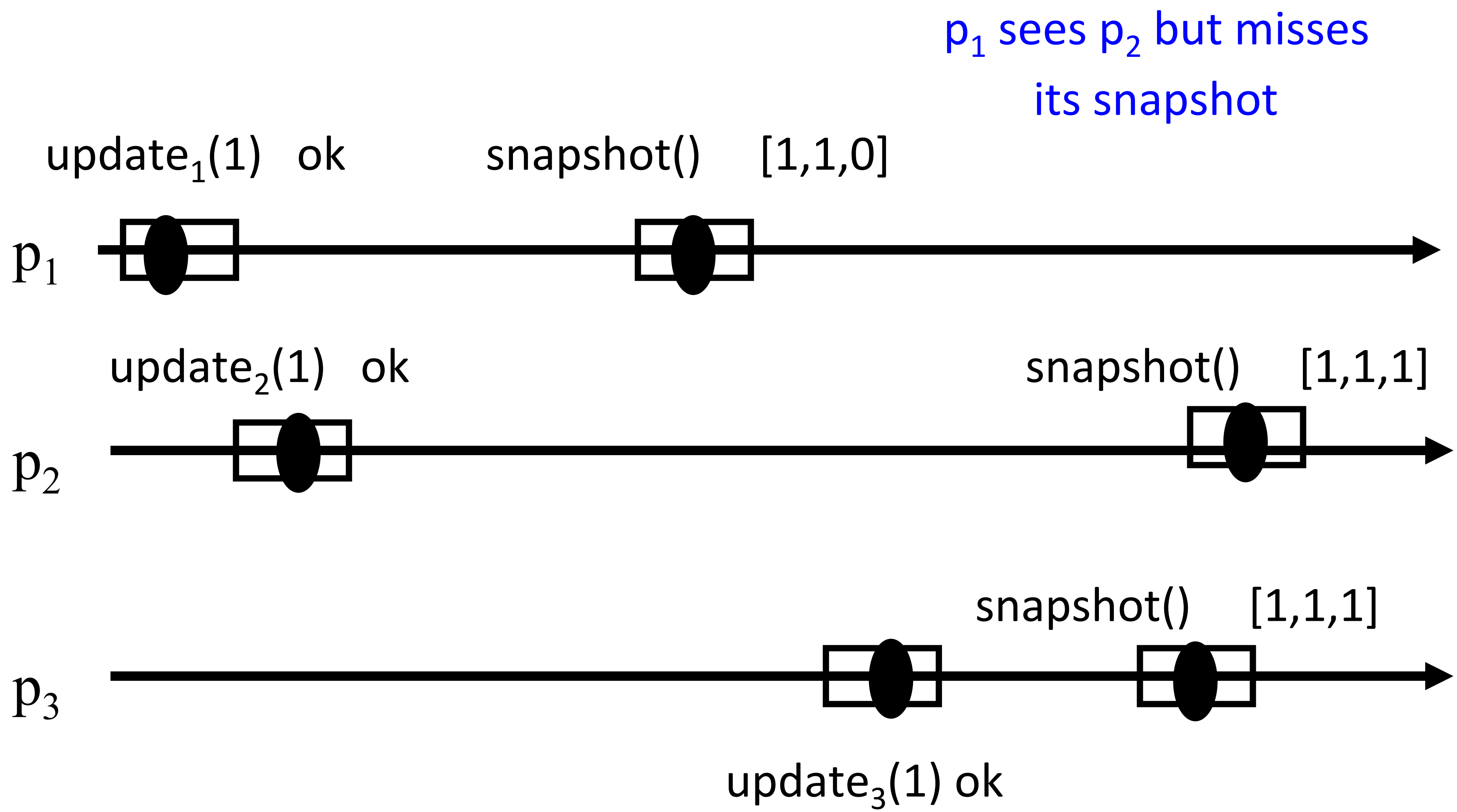
Vectors S_i satisfy:

- Self-inclusion: for all i: v_i is in S_i
- Containment: for all i and j: S_i is subset of S_j or S_j is subset of S_i

TELECOM ParisTech

© 2015 P. Kuznetsov

“Unbalanced” snapshots



© 2018 P. Kuznetsov

Enumerating possible runs: two processes

Each process p_i (i=1,2):

$update_i(v_i)$

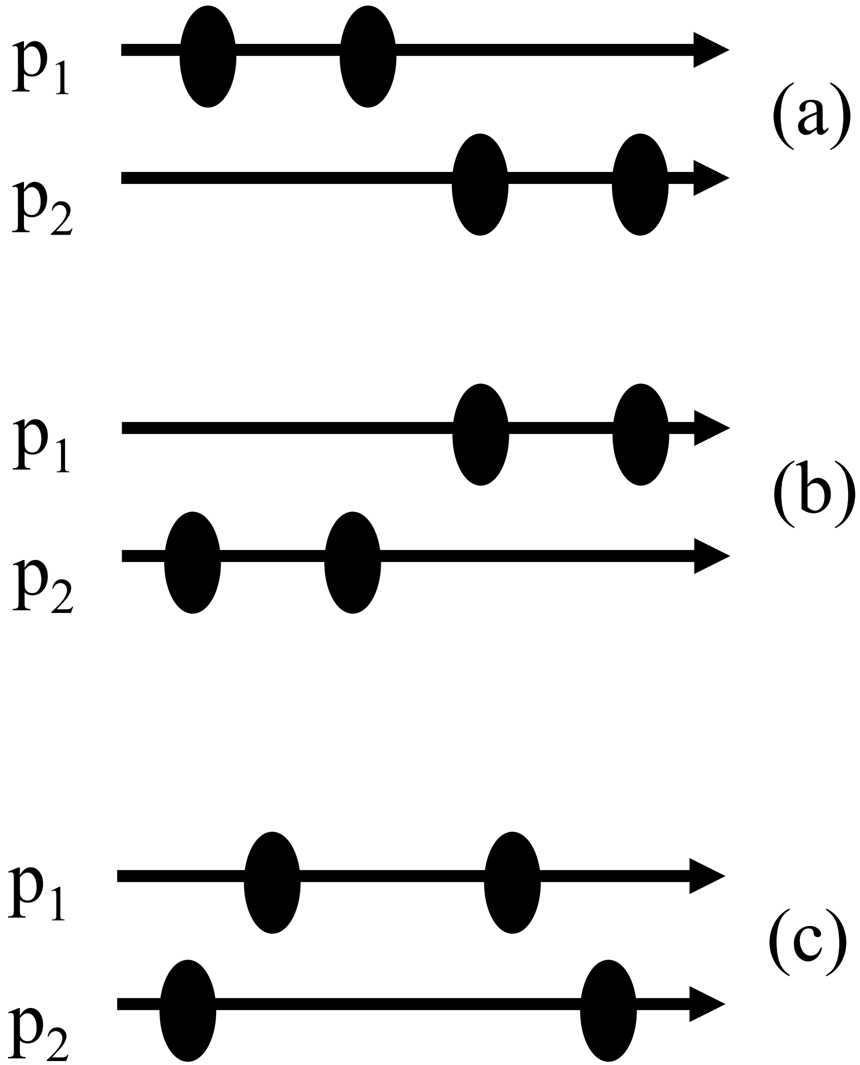
S_i := snapshot()

Three cases to consider:

(a) p_1 reads before p_2 writes

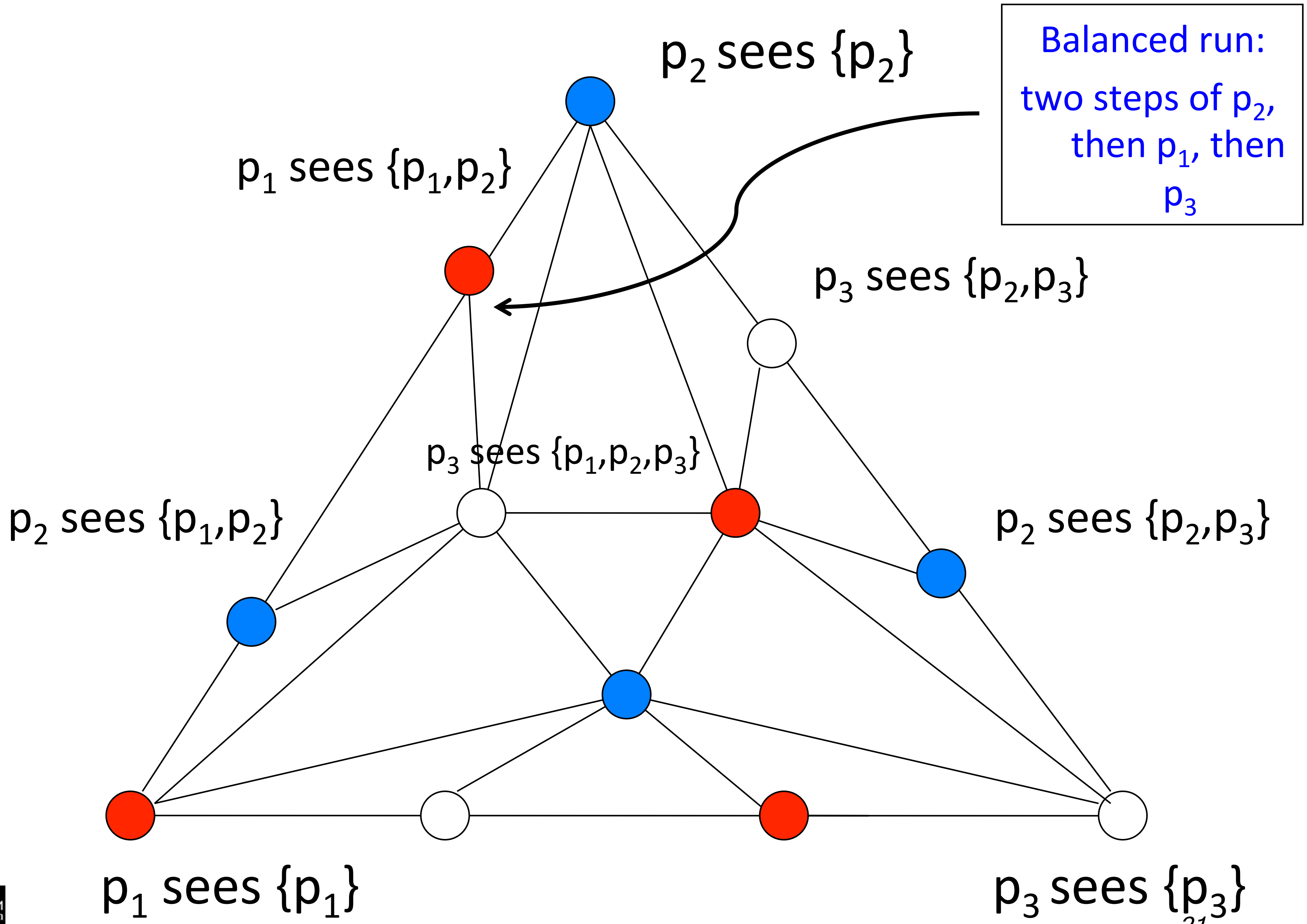
(b) p_2 reads before p_1 writes

(c) p_1 and p_2 go "lock-step": first both write, then both read

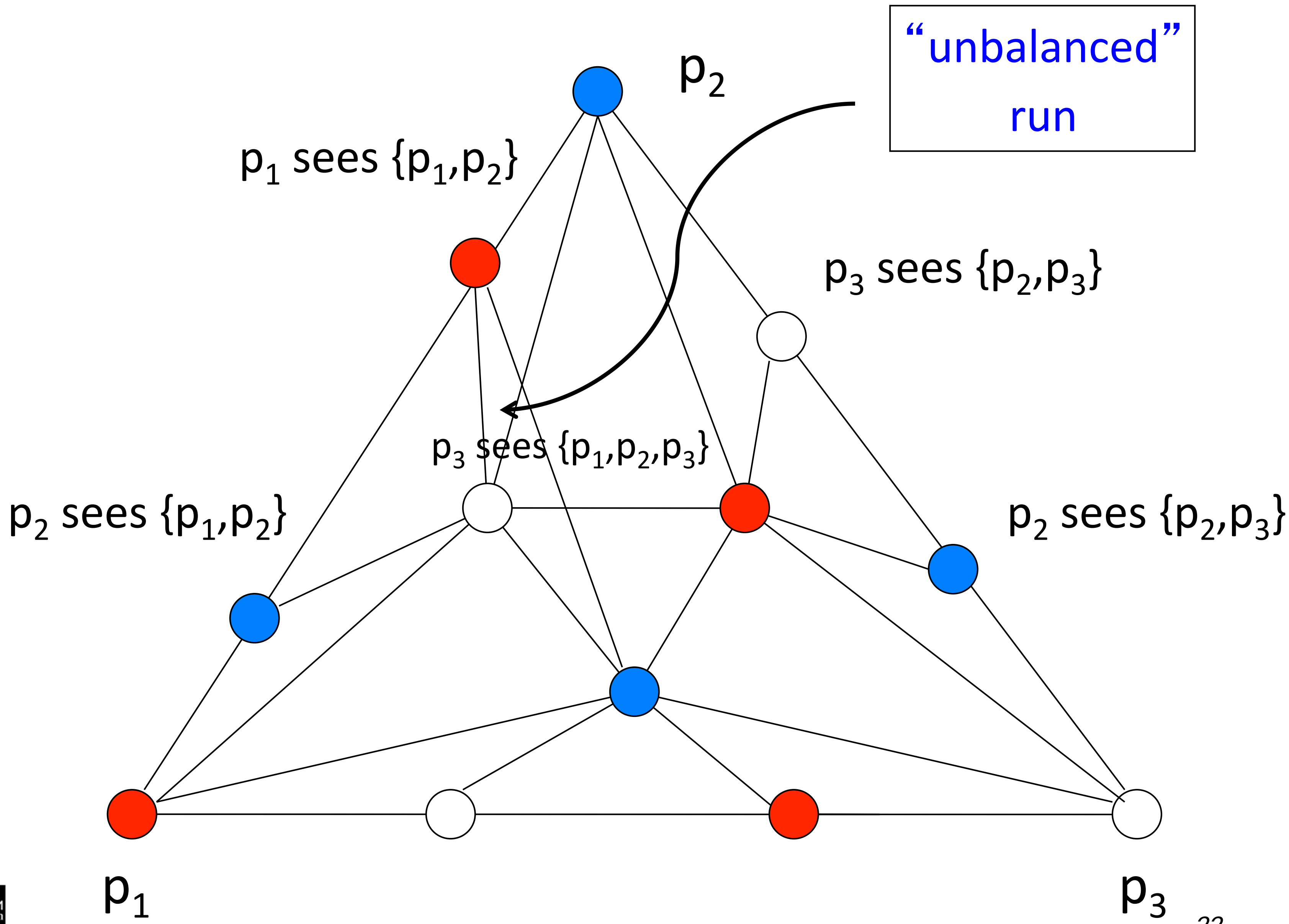


© 2018 P. Kuznetsov

Topological representation: one-shot AS



Topological representation: one-shot AS



One-shot *immediate* snapshot (IS)

One operation:
WriteRead(v)

Each process p_i:
S_i := $WriteRead_i(v_i)$

Vectors $S_1,\ldots,S_N$ satisfy:

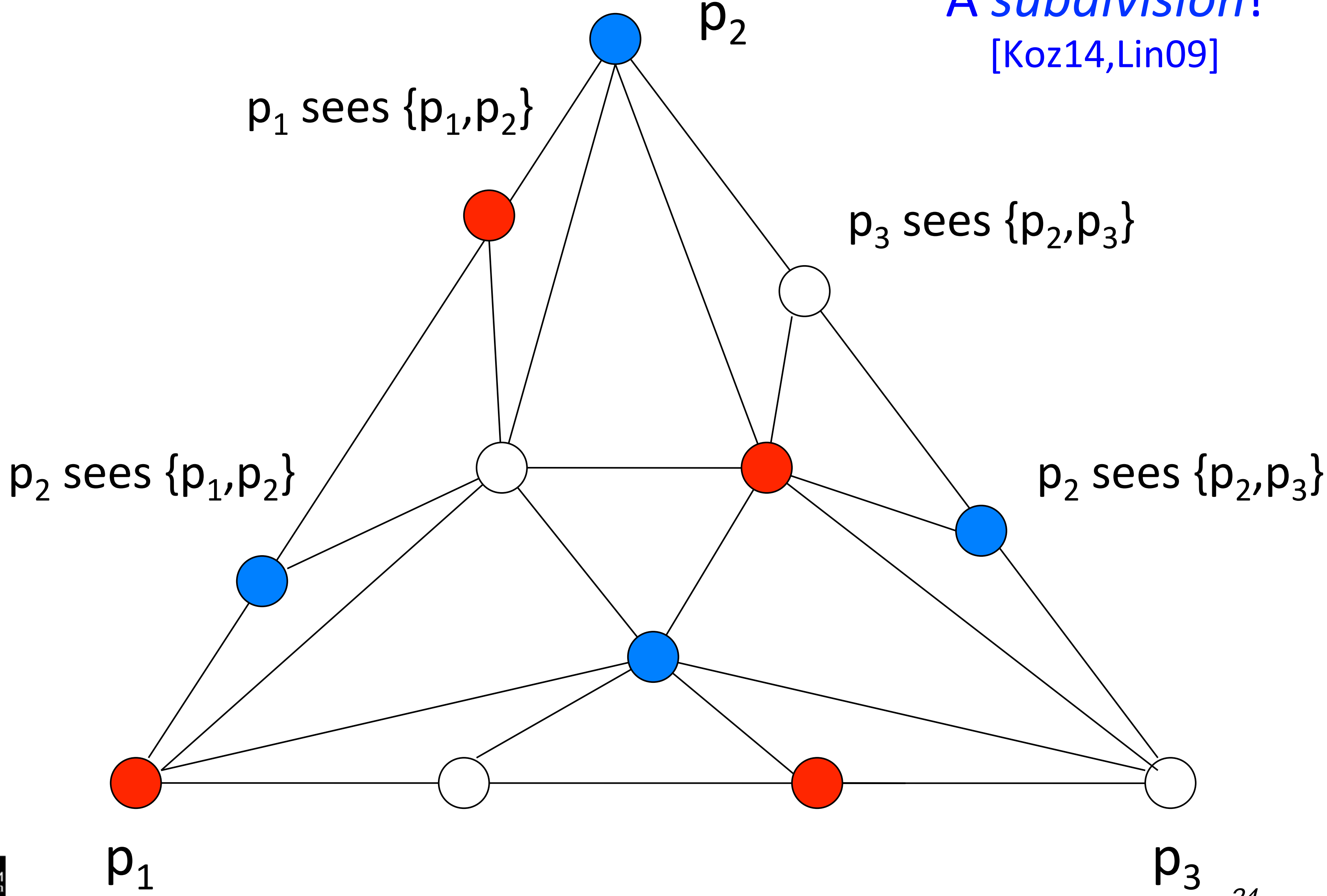
- Self-inclusion: for all i: v_i is in S_i
- Containment: for all i and j: S_i is subset of S_j or S_j is subset of S_i
- Immediacy: for all i and j: if v_i is in S_j, then is S_i is a subset of S_j

© 2015 P. Kuznetsov

Topological representation: one-shot IS

A *subdivision*!

[Koz14,Lin09]



IS is equivalent to AS (one-shot)

- IS is a restriction of one-shot AS => IS is stronger than one-shot AS
  - ✓Every run of IS is a run of one-shot AS
- Show that a few (one-shot) AS objects can be used to implements IS
  - ✓One-shot ReadWrite() can be implemented using a series of update and snapshot operations

© 2015 P. Kuznetsov

IS from AS

shared variables:

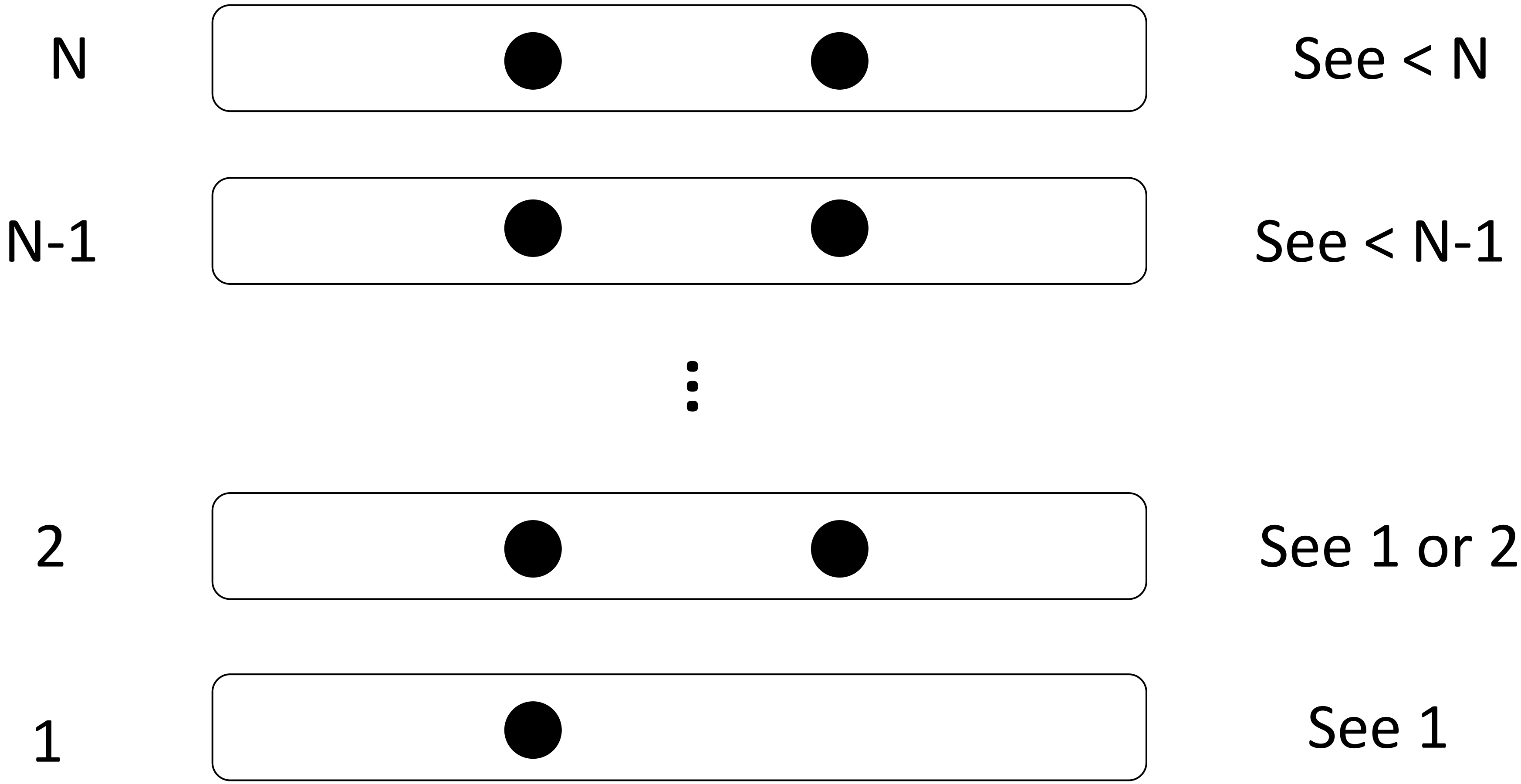
$A_1,\ldots,A_N$ – atomic snapshot objects, initially [T,…,T]

Upon WriteRead$_i$(v_i)

```
r := N+1
while true do
        r := r-1                // drop to the lower level
        A_r.update_i(v_i)
        S :=  A_r.snapshot()
        if |S|=r then           // |S| is the number of non-T values in S
            return S
```

© 2015 P. Kuznetsov

Drop levels: two processes, N>3



© 2015 P. Kuznetsov

Correctness

The outcome of the algorithm satisfies Self-Inclusion, Snapshot, and Immediacy

- By induction on N: for all N>1, if the algorithm is correct for N-1, then it is correct for N
- Base case N=1: trivial

© 2015 P. Kuznetsov

Correctness, contd.

- Suppose the algorithm is correct for N-1 processes
- N processes come to level N
  - ✓ At most N-1 go to level N-1 or lower
  - ✓ (At least one process returns in level N)
  - ✓ Why?
- Self-inclusion, Containment and Immediacy hold for all processes that return in levels N-1 or lower
- The processes returning at level N return all N values
  - ✓ The properties hold for all N processes! Why?

© 2015 P. Kuznetsov

Iterated Immediate Snapshot (IIS)

Shared variables:

$IS_1, IS_2, IS_3, \ldots$ // a series of one-shot IS

Each process p_i with input v_i:

r := 0

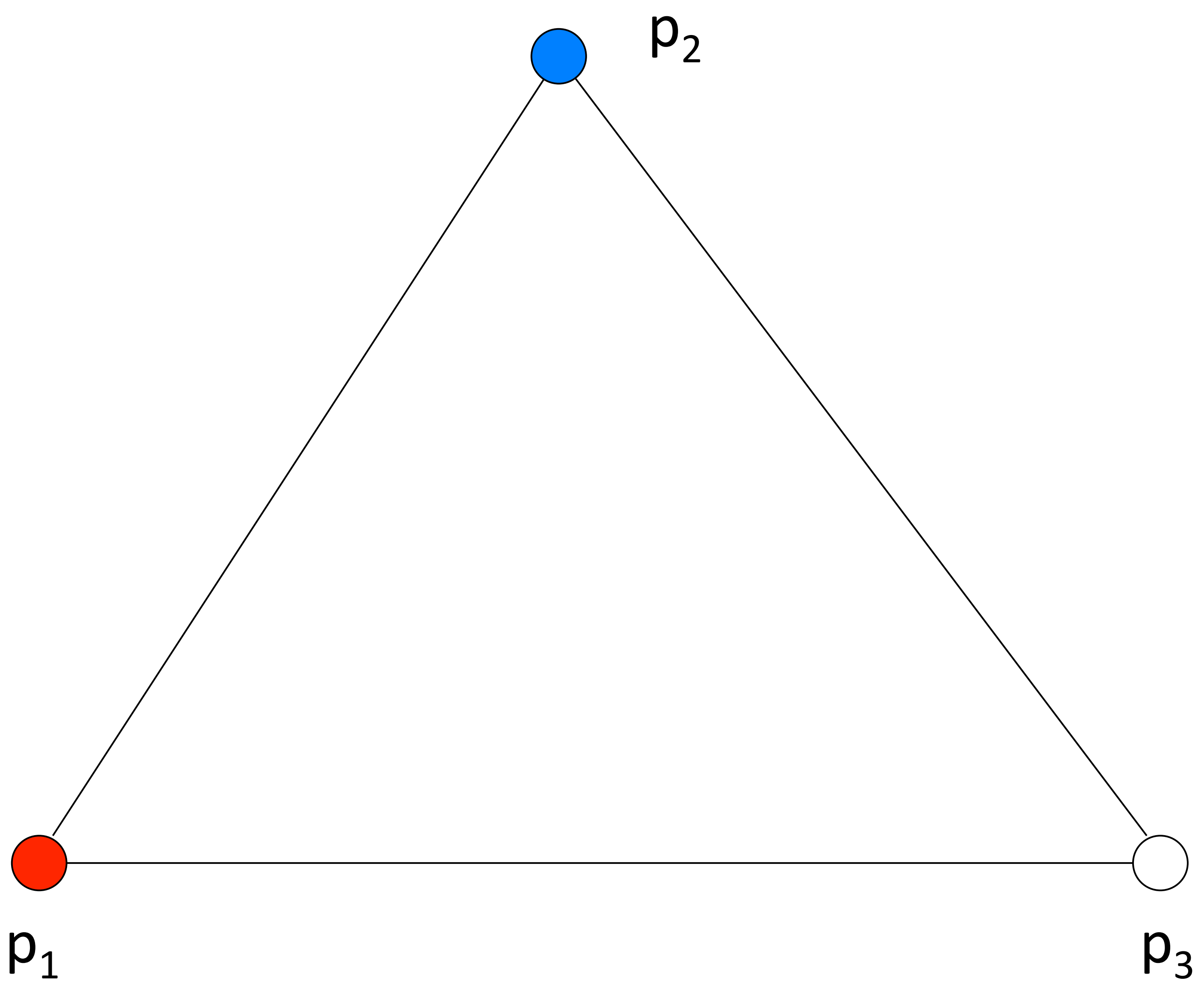
while true do

  r := r+1

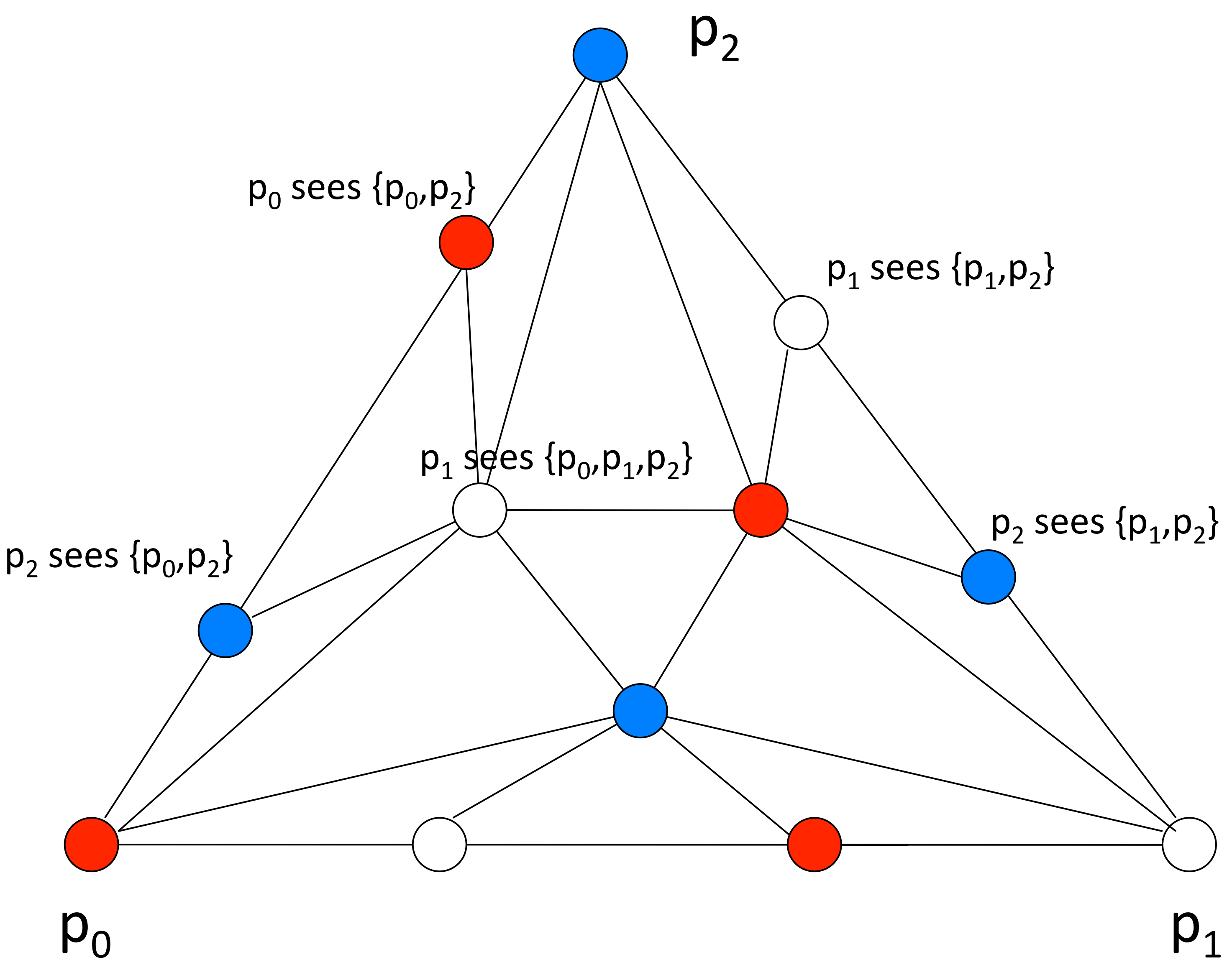
  $v_i := IS_r.WriteRead_i(v_i)$

© 2015 P. Kuznetsov

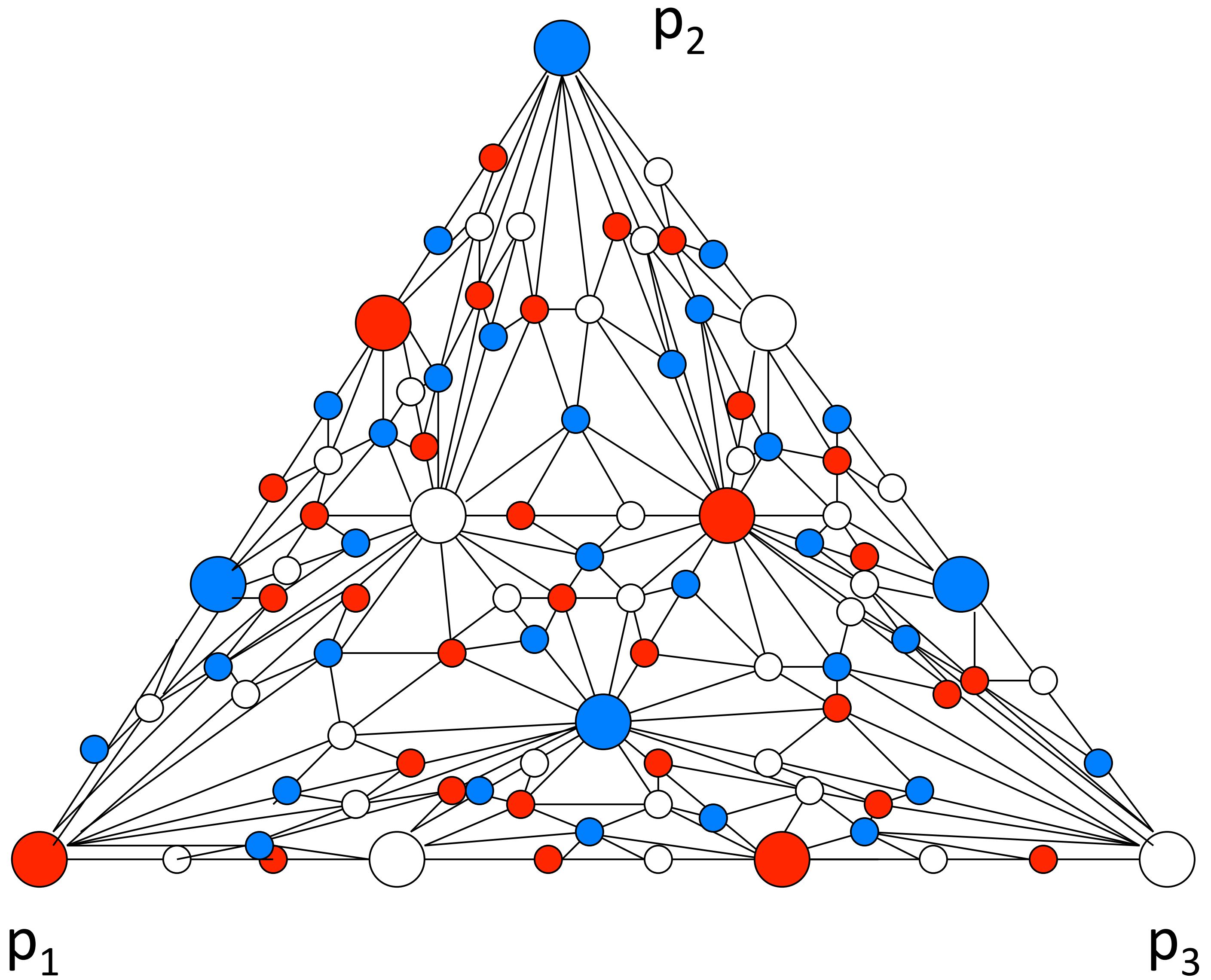
Iterated standard chromatic subdivision (ISDS)



$X(s^2)$: one-shot IS for 3 processes



ISDS: two rounds of IIS



IIS is equivalent to (multi-shot) AS

- AS can be used to implement IIS (wait-free)
  - ✓Multiple instances of the construction above (one per iteration)
- IIS can be used to implement multi-shot AS in the lock-free manner [BG93,GR10]:
  - ✓At least one correct process performs infinitely many read or write operations
  - ✓Good enough for protocols solving distributed tasks!

© 2015 P. Kuznetsov

IIS=AS for wait-free task solutions

- Suppose we simulate a wait-free protocol for solving a task:
  - ✓Every process starts with an input
  - ✓Every process taking sufficiently many steps (of the full-information protocol) eventually decides (and thus stops writing new values, but keeps writing the last one)
  - ✓Outputs match inputs (we'll see later how it is defined)
- If a task can be solved in AS, then it can be solved in IIS
  - ✓We consider IIS from this point on

© 2015 P. Kuznetsov

Combinatorial Structures for Distributed Computing

Distributed tasks and consensus

Kyoto University, 2018

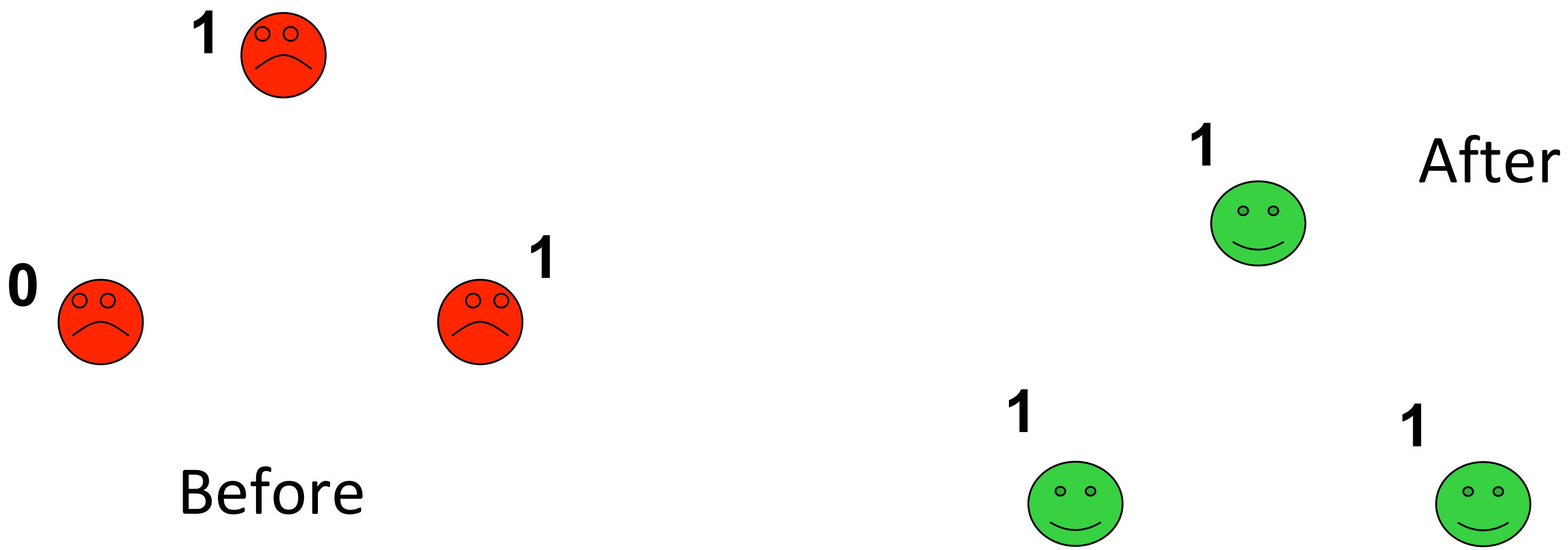
System model

- N *asynchronous* (no bounds on relative speeds) processes $p_0,\ldots,p_{N-1}$ ($N\geq 2$) communicate via atomic read-write registers

- Processes can fail by crashing
  - ✓A crashed process takes only finitely many steps (reads and writes)
  - ✓Up to t processes can crash: t-resilient system
  - ✓t=N-1: wait-free

Consensus

Processes *propose* values and must *agree* on a common decision value so that the decided value is a proposed value of some process



Consensus: definition

A process *proposes* an *input* value in V (|V|≥2) and tries to *decide* on an *output* value in V

- *Agreement:* No two processes decide on different values
- *Validity:* Every decided value is a proposed value
- *Termination:* No process takes infinitely many steps without deciding

  (Every correct process decides)

Optimistic (0-resilient) consensus

Consider the case t=0, no process fails

Shared: 1WNR register D, initially T (default value not in V)

Upon propose(v) by process p_i:

if i = 0 then D.write(v) // if p_0 decide on v

wait until D.read() ≠ T // wait until p_0 decides

return D

(every process decides on p_0's input)

TELECOM ParisTech

© 2018 P. Kuznetsov

Impossibility of wait-free consensus

Theorem 1 No wait-free algorithm solves consensus using read-write memory

We give the proof for N=2, assuming that p_0 proposes 0 and p_1 proposes 1

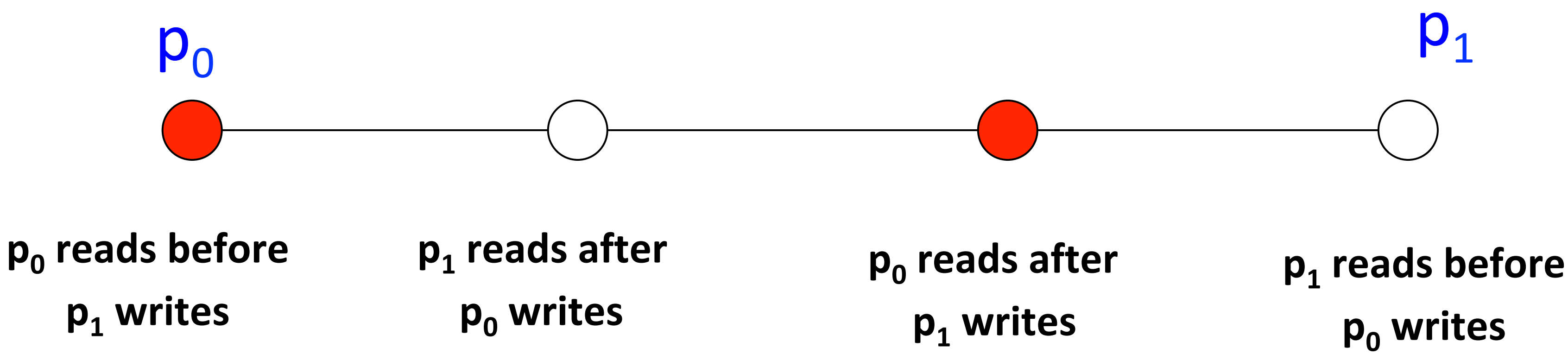
Implies the claim for all N≥2

Consider the IIS model

Proof of Theorem 1

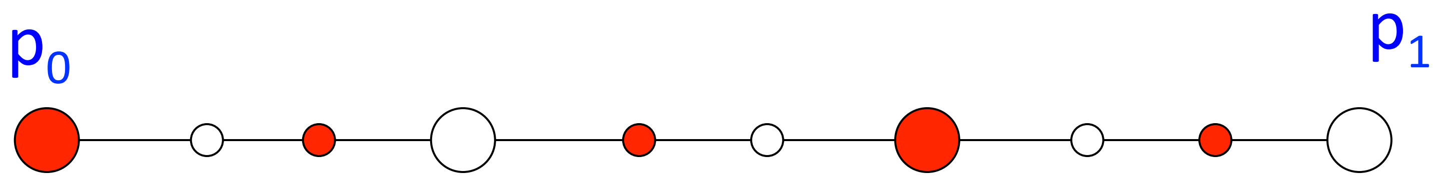
Initially each p_i only knows its input

One round of IIS:



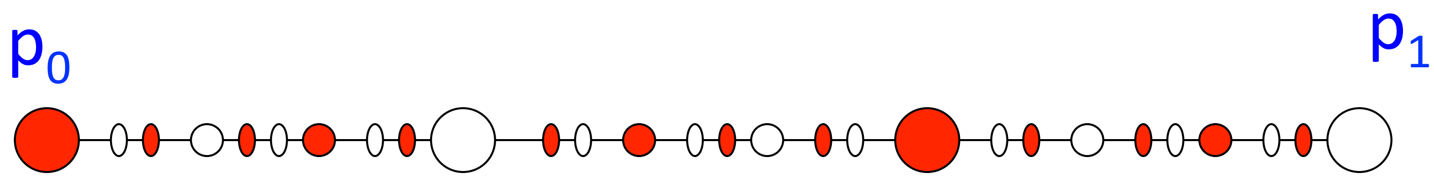
Proof sketch for Theorem 1

Two rounds:



Proof of Theorem 1

And so on…



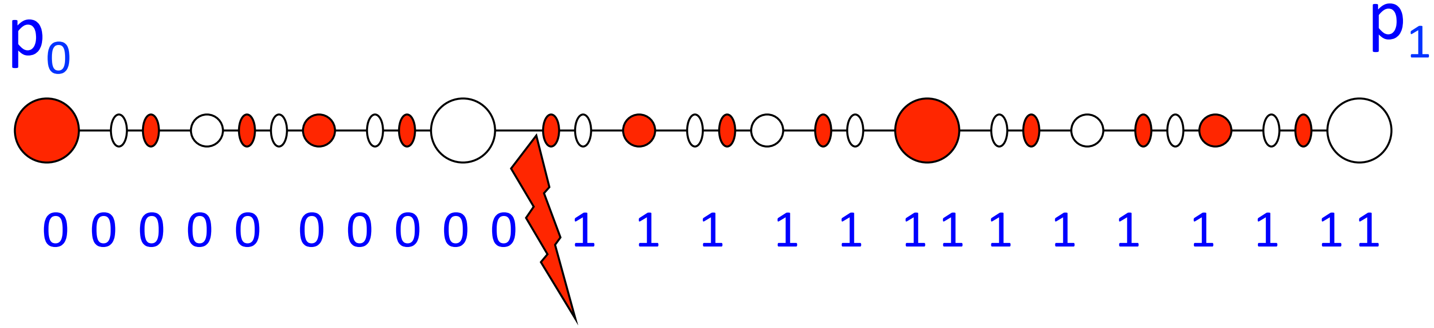
Solo runs remain connected - no way to decide!

Proof of Theorem 1

Suppose p_i (i=0,1) proposes i

- p_i must decide i in a solo run!

Suppose by round r every process decides



There exists a run with conflicting decisions!

1-resilient consensus?

What if we have 1000000 processes and one of them can crash?

NO

A more sophisticated proof is needed [FLP85,LA87]

TELECOM ParisTech

© 2018 P. Kuznetsov

But why consensus is interesting?

Because it is universal!

- If we can solve consensus among N processes, then we can *implement* any object shared by N processes
  - ✓T&S and queues are universal for 2 processes
- A key to implement a generic fault-tolerant service (replicated state machine)

© 2018 P. Kuznetsov

Universal construction

Theorem 1 [Herlihy, 1991] If N processes can solve consensus, then N processes can (wait-free) implement any object $O=(Q,O,R,\sigma)$

© 2018 P. Kuznetsov

Consensus number

An object O has consensus number k (we write cons(O)=k) if

- k-process consensus can be solved using registers and any number of copies of O but (k+1)-consensus cannot

If no such number k exists for O, then cons(O)=∞

(k=cons(O) is the maximal number of processes that can be synchronized using copies of O and registers)

TELECOM ParisTech

© 2018 P. Kuznetsov

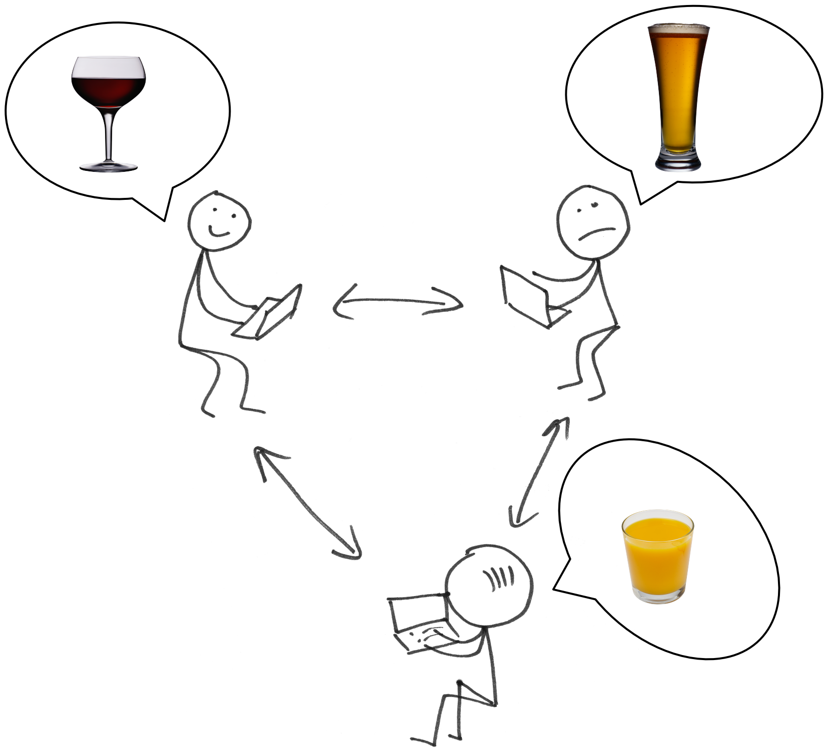
Consensus power

- cons(register)=1
- cons(T&S)=cons(queue)=2
- …
- cons(N-consensus)=N
  - ✓N-consensus is N-universal!
- …
- cons(CAS)=∞

TELECOM ParisTech

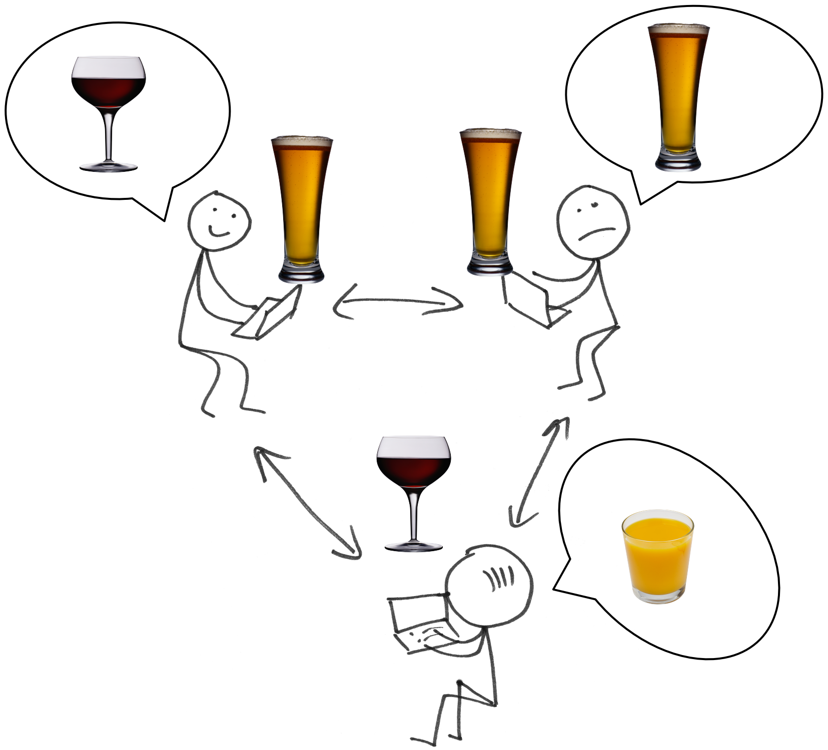
© 2018 P. Kuznetsov

Set consensus



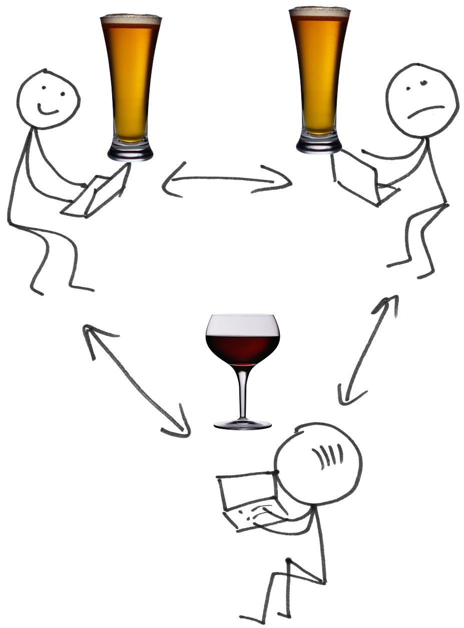
Processes start with private inputs

Set consensus



Outputs should form a bounded subset of inputs

Set consensus



2-set consensus

~ two replicated state machines: one making progress

k-set consensus ~ k replicated state machines [GG10]

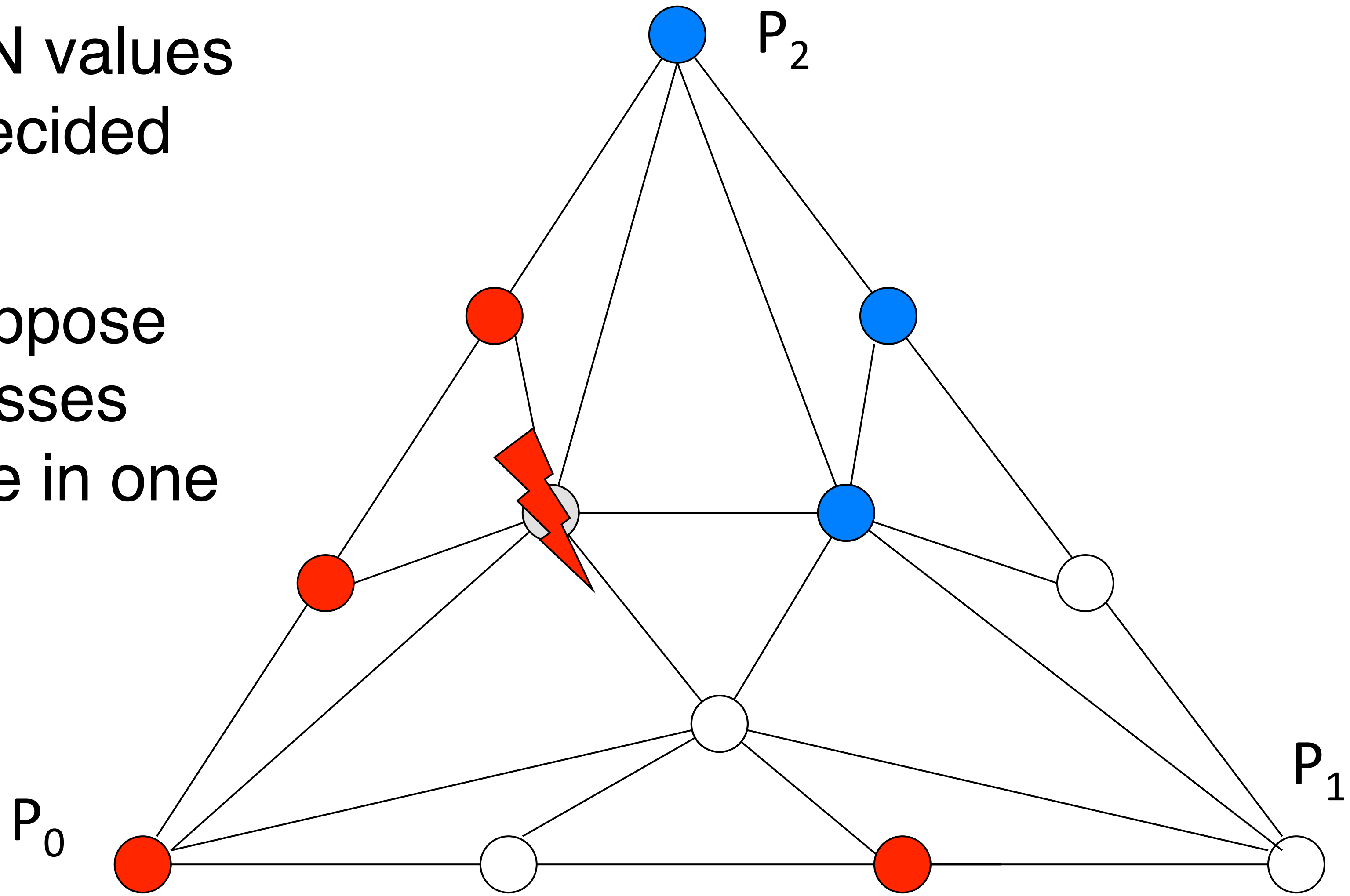
Impossibility of wait-free set consensus

Theorem 1 No wait-free algorithm solves (N-1)-set consensus in IIS (and, thus, in read-write memory)

Reduces to Sperner lemma: impossibility of Sperner coloring on a manifold

In at least one IIS run, N values are decided

E.g.: suppose processes decide in one round



Takeaways

- The read-write model can be represented as a standard chromatic subdivision
  - ✓RW $\equiv$ IIS (for tasks)
  - ✓IIS $\equiv$ standard chromatic subdivision [BG97,Lin09,Koz14]
- Wait-free set consensus is impossible
  - ✓Equivalent to Sperner coloring of a subdivided simplex
- Next: topological characterization of task computability
  - ✓Wait-free and beyond