Demystifying Bitcoin

Journey to the center of distributed computing

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

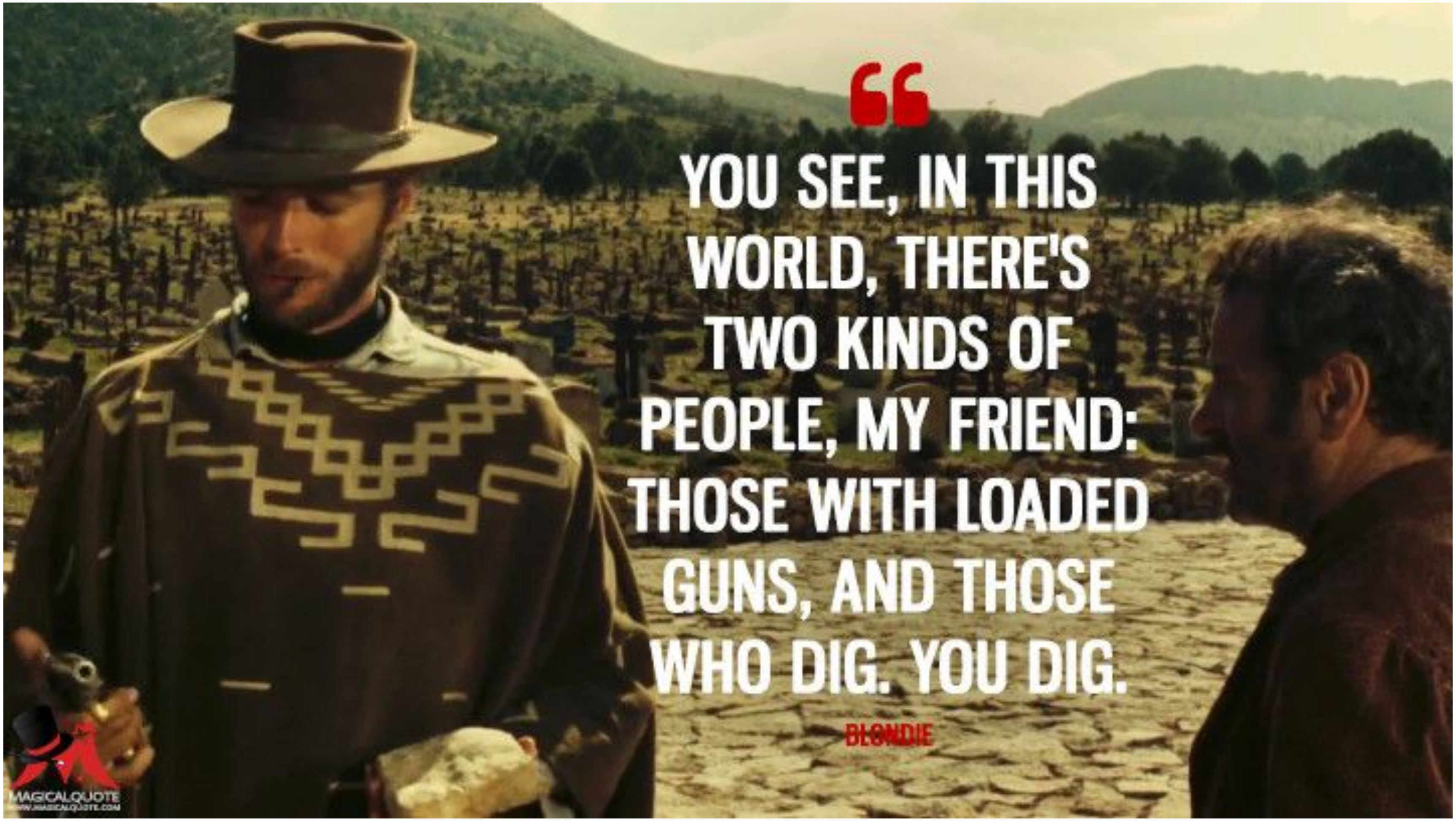
Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As

Roadmap

- **(1) The main question**
  - **(2) The bitcoin problem**
    - **(3) The main answer**

- **X000 implementations**

- « Computing's central challenge is how not to make a mess of it …» E. Dijkstra



YOU SEE, IN THIS WORLD, THERE'S TWO KINDS OF PEOPLE, MY FRIEND: THOSE WITH LOADED GUNS, AND THOSE WHO DIG. YOU DIG.
BLONDIE
MAGICALQUOTE

P vs NP

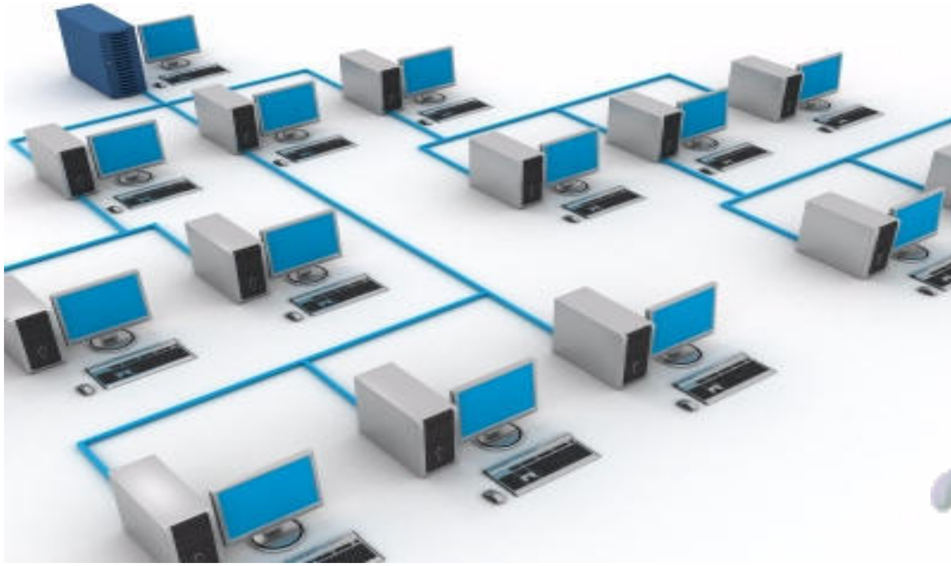
$7 * 13 = ?$ $? * ? = 91$

Asynchronous vs Synchronous

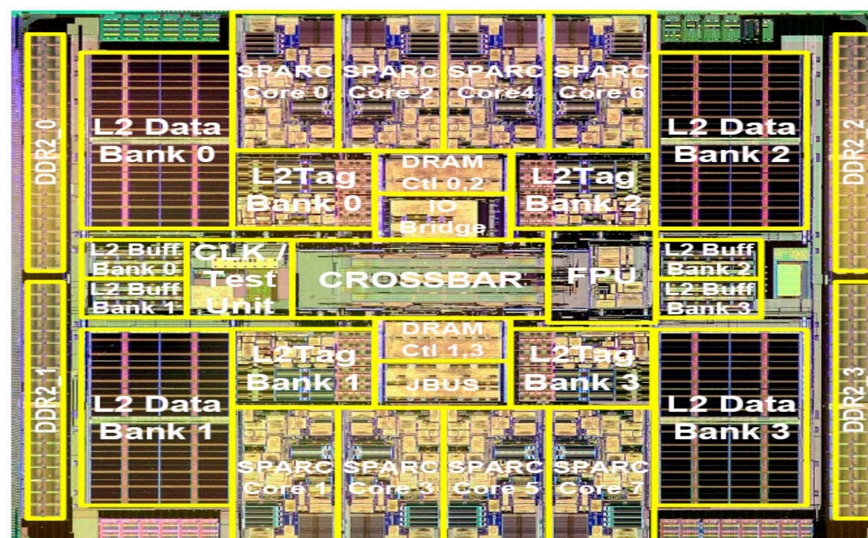
Is payment an asynchronous problem?

- « To understand a distributed computing problem: bring it to shared memory » T. Lannister

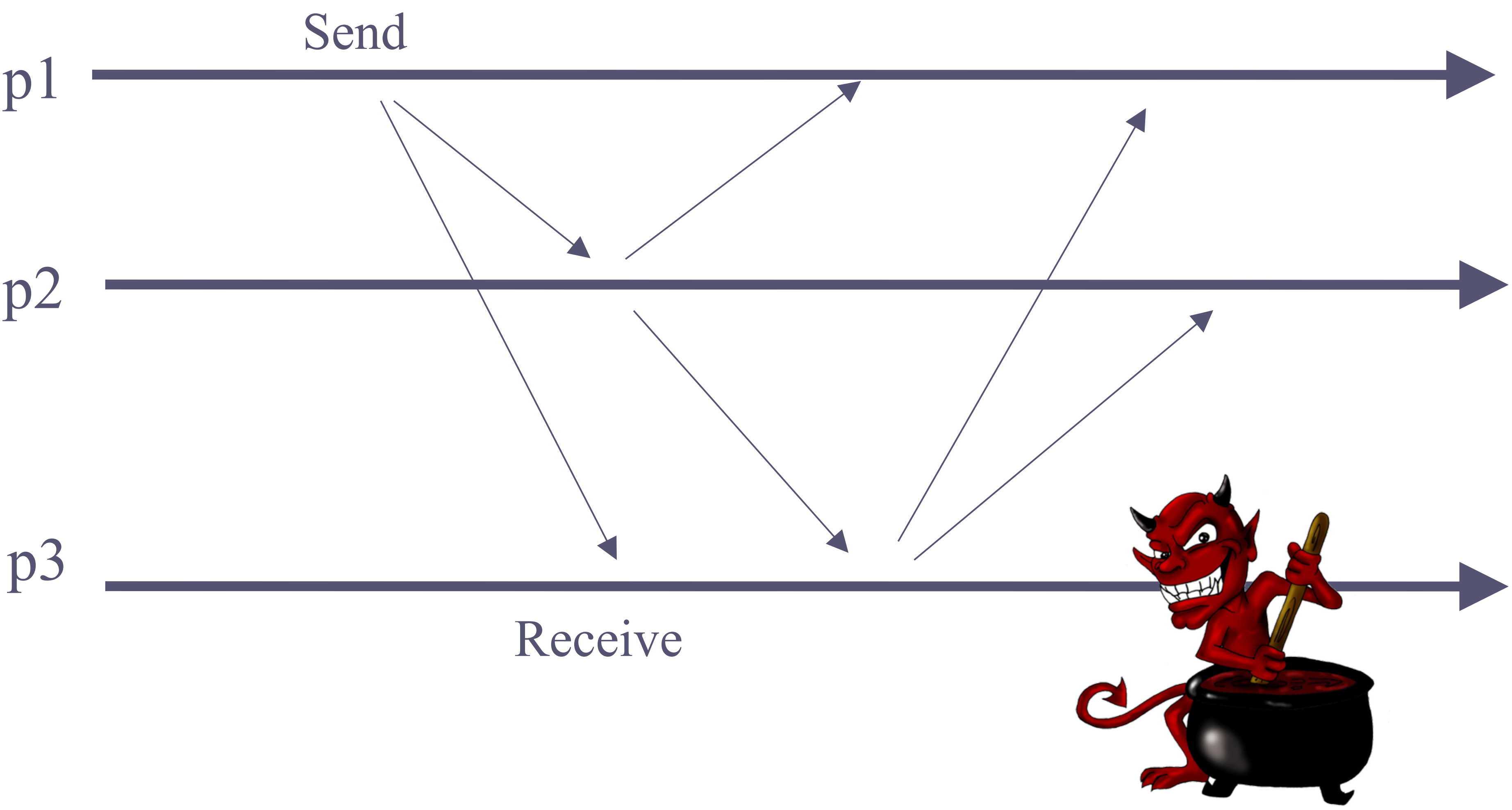
The infinitely big



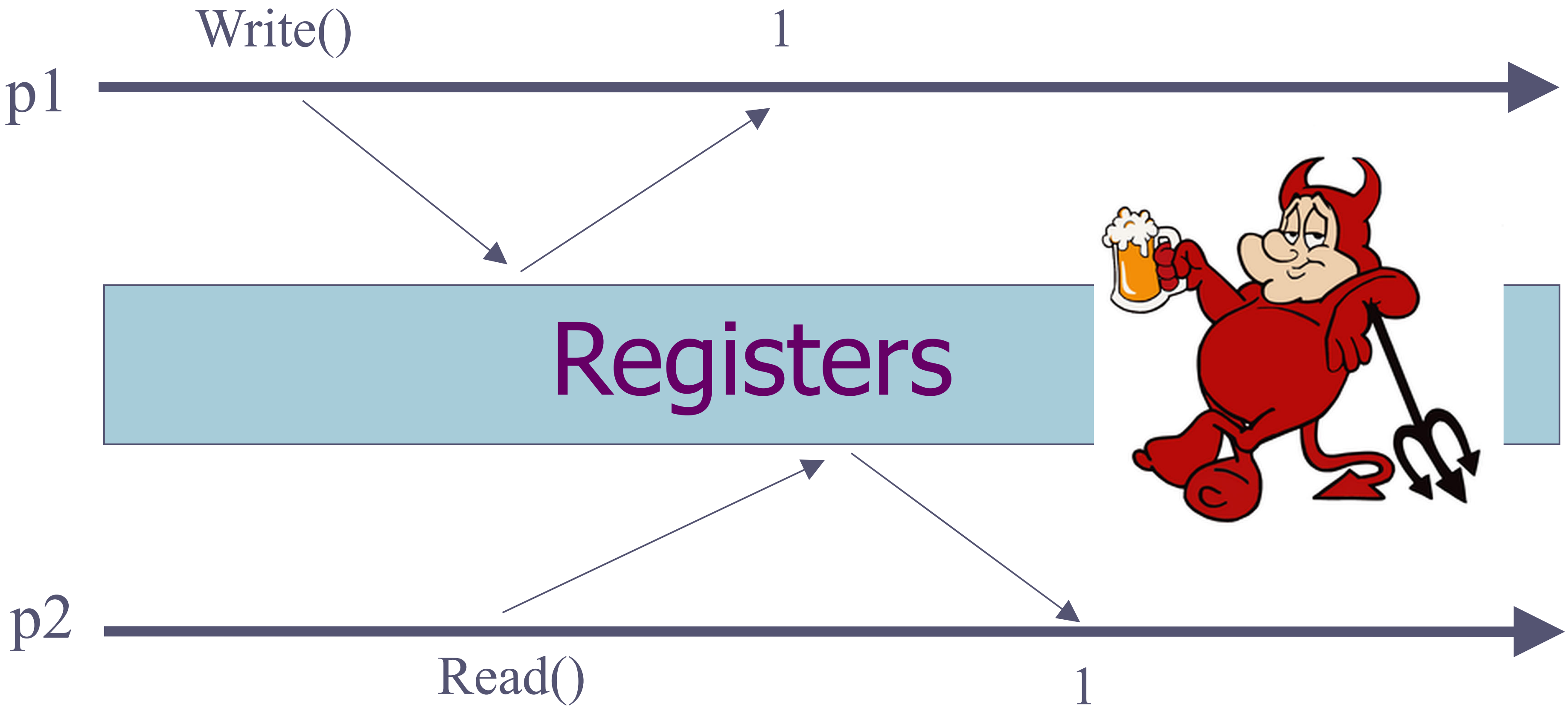
The infinitely small



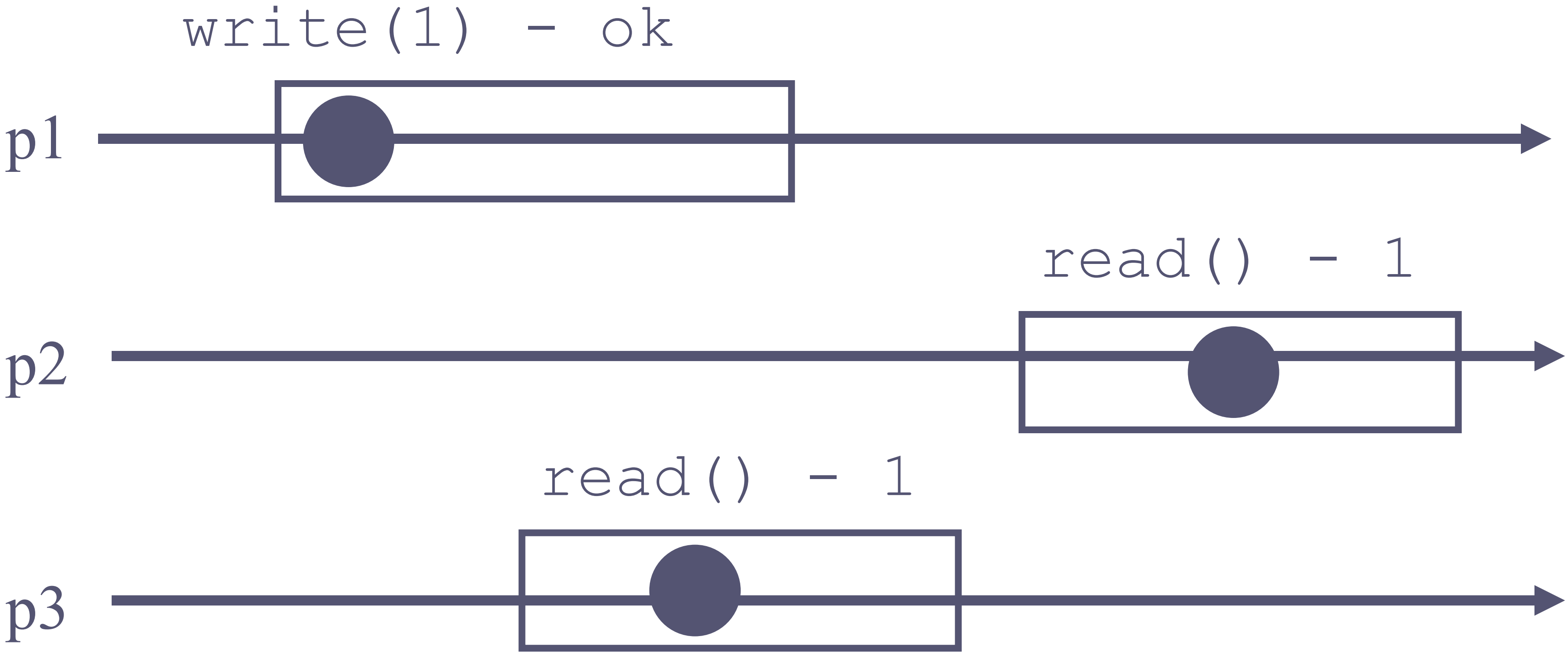
Message Passing



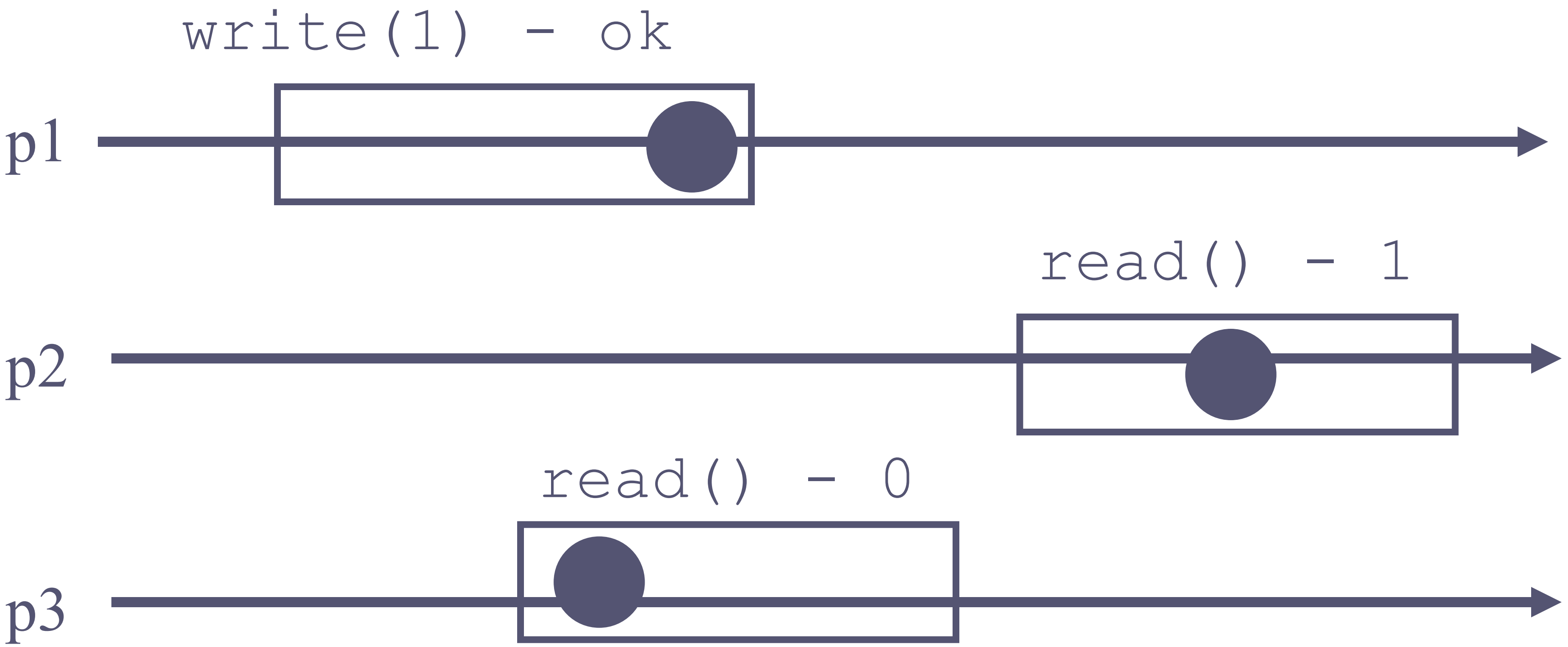
Shared Memory



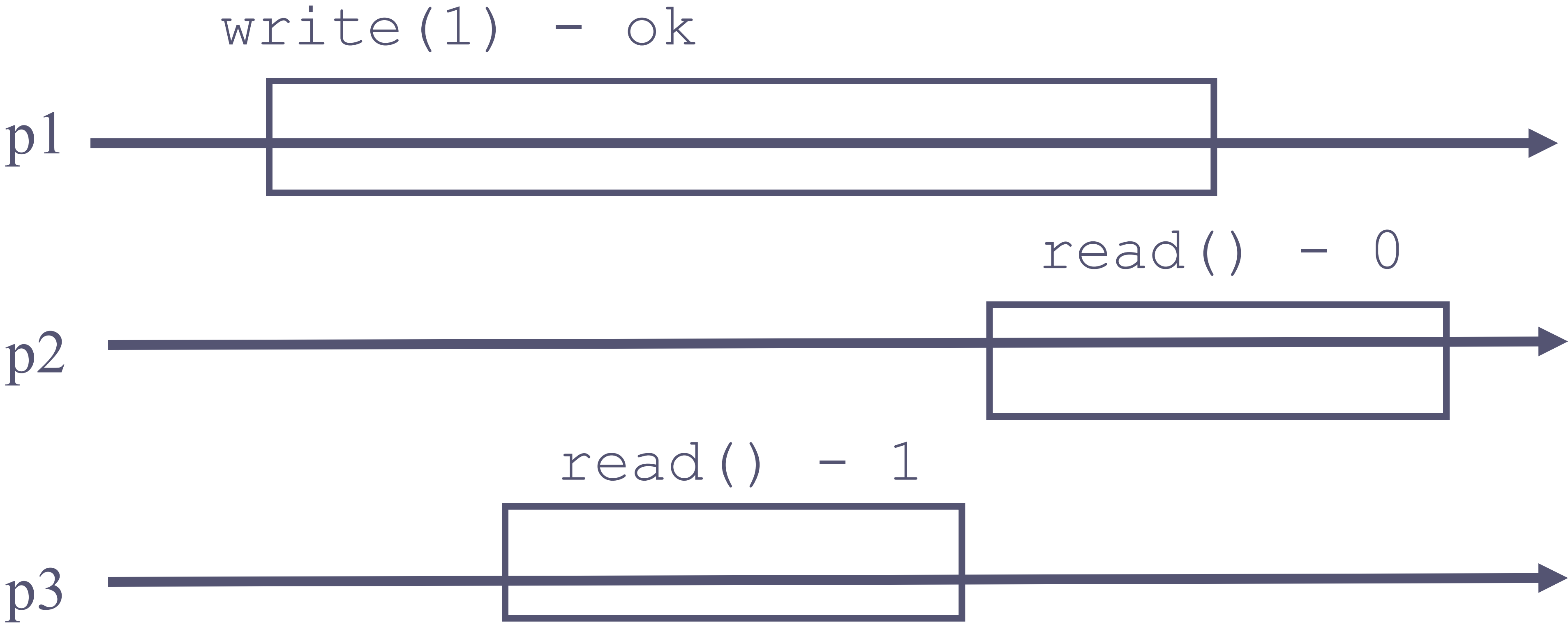
Atomic Shared Memory



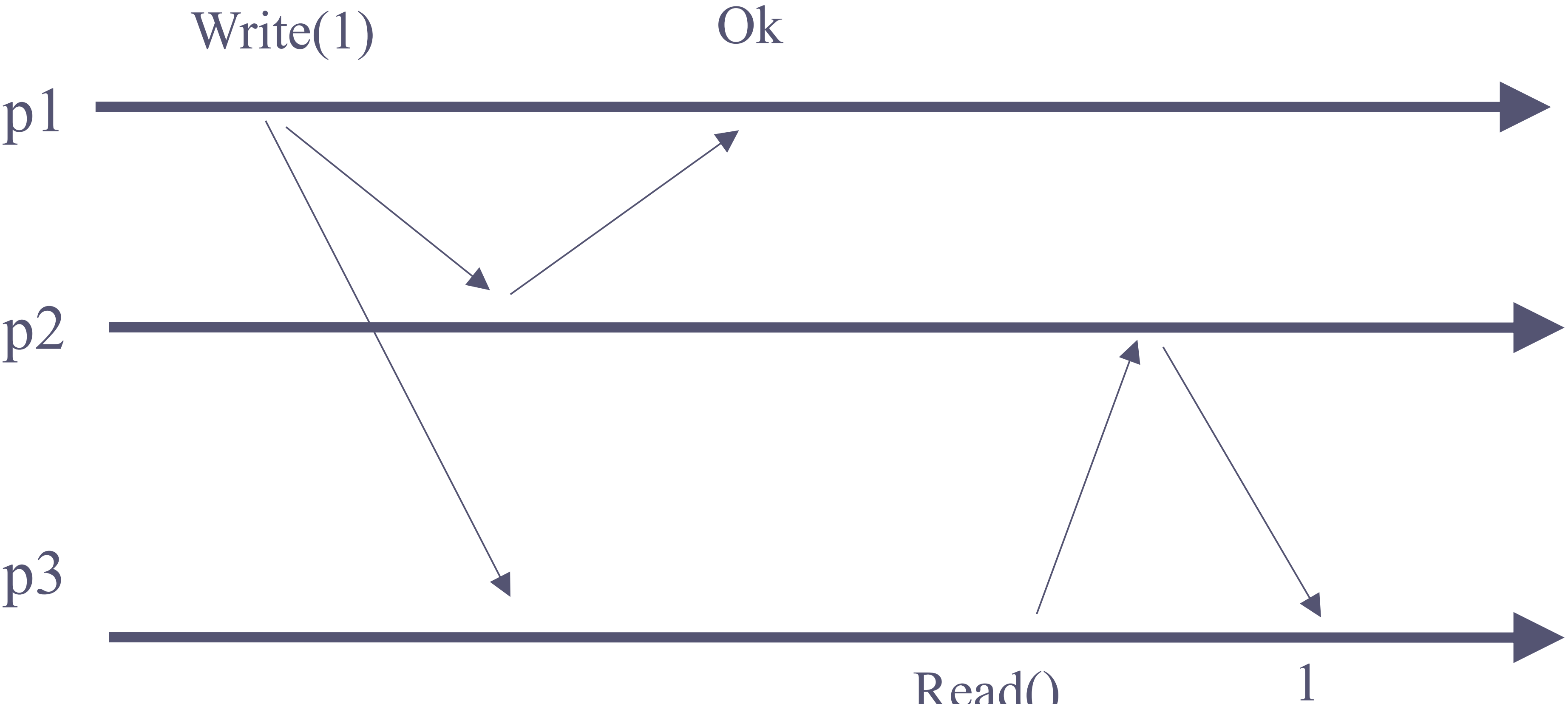
Atomic Shared Memory



Non-Atomic Shared Memory

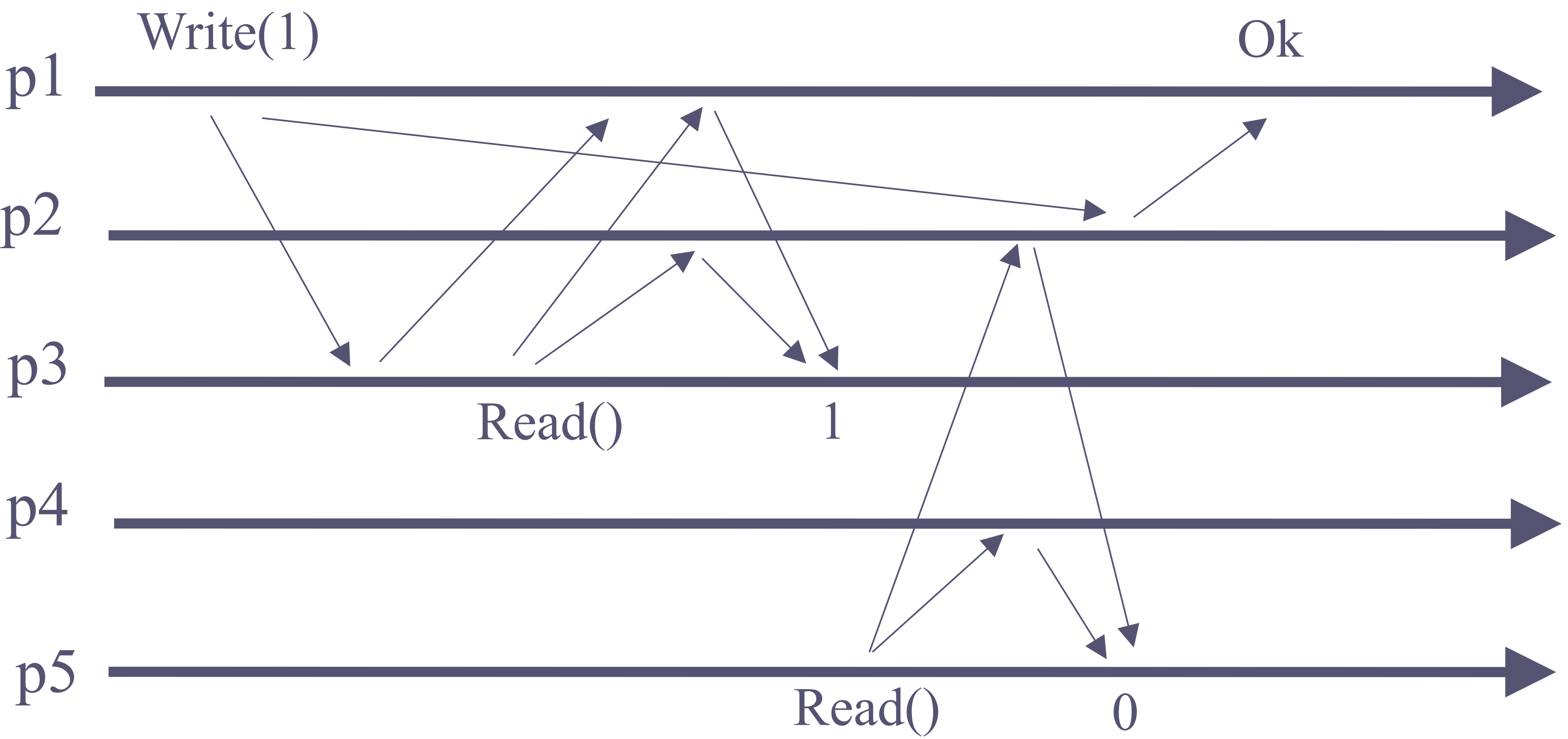


Message Passing ⇔ Shared Memory



Quorums

Message Passing ⇔ Shared Memory



Message Passing ⇔ Shared Memory Modulo Quorums

⇔

- « Optimization is the source of all evil » D. Knuth

Is payment an asynchronous problem?

- **Payment Object**

  - Atomicity
  - Wait-freedom

Counter: Specification

- A *counter* has two operations *inc()* and *read()*; it maintains an integer *x init to 0*
- *read():*
  - return(x)
- *inc():*
  - x := x + 1;
  - return(ok)

Counter: Algorithm

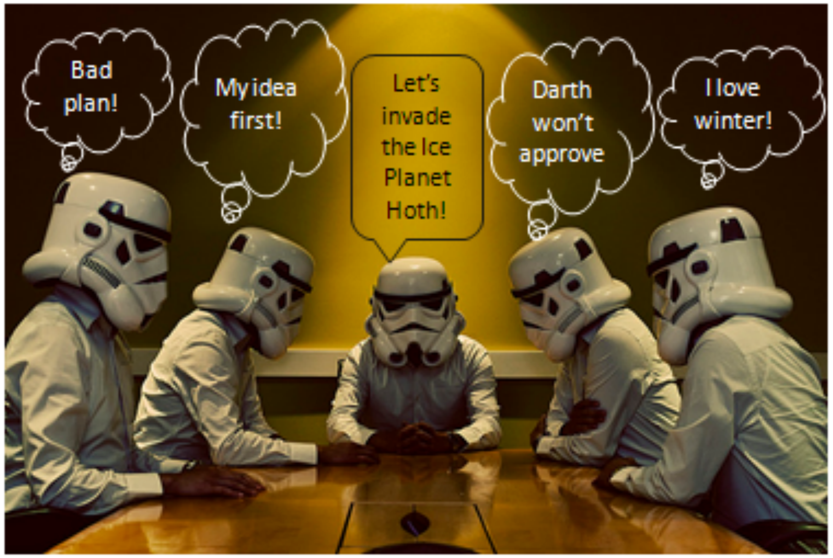
- The processes share an array of registers Reg[1,..,N]
- *inc():*
  - Reg[i].write(Reg[i].read() +1);
  - return(ok)
- *read():*
  - sum := 0;
  - for j = 1 to N do
    - sum := sum + Reg[j].read();
  - return(sum)

Counter*: Specification

- *Counter** has, in addition, operation *dec()*
- *dec():*
  - if x > 0 then x := x - 1; return(ok)
  - else return(no)

Can we implement Counter* asynchronously?

Consensus



Agreement on a single value among multiple

Safety: No two processes must choose different values.

The chosen value must have been proposed by a process.

Liveness: Each process must eventually choose a value.

2-Consensus with Counter*

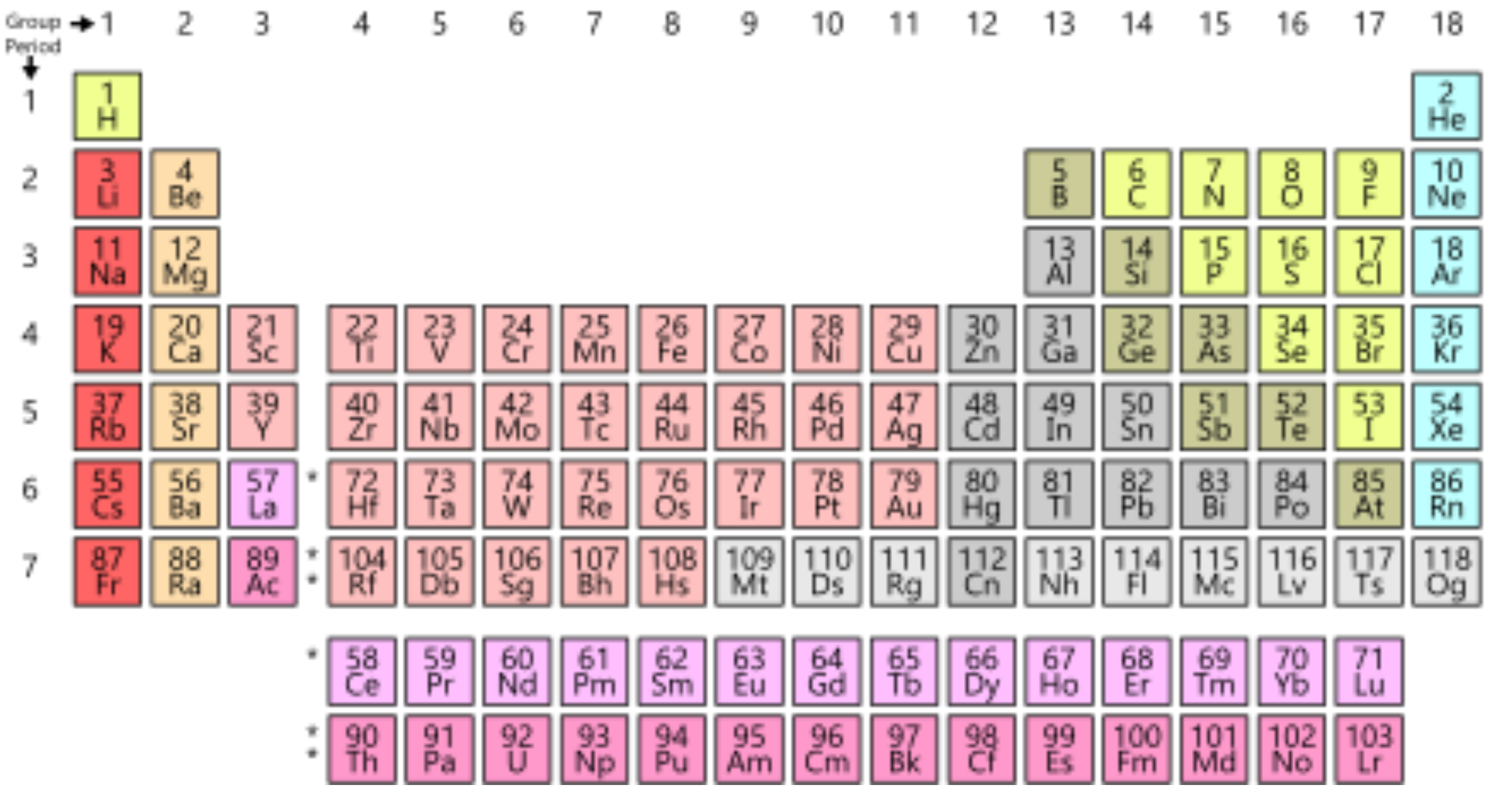
- Registers R0 and R1 and Counter* C - initialized to 1
- Process pI:
- propose(vI)
- RI.write(vI)
- res := C.dec()
- if(res = ok) then
  - ✓ return(vI)
  - ✓ else return(R{1-I}.read())

Impossibility [FLP85,LA87]

- ***Theorem**: no asynchronous* algorithm implements *consensus* among two processes using *registers*

- ***Corollary:*** no asynchronous algorithm implements Counter* among two processes using *registers*

- The **consensus number** of an object is the maximum number of processes than can solve consensus with it



Roadmap

- (1) The main question
    - (2) The bitcoin problem/object
        - (3) The main answer

Payment Object (PO): Specification

- Pay(a,b,x): transfer amount x from a to b if a > x (return ok; else return no)
- **Important**. Only the owner of a invokes Pay(a,*,*)

- ***Questions***:
- - can PO be implemented asynchronously?
- - what is the consensus number of PO?

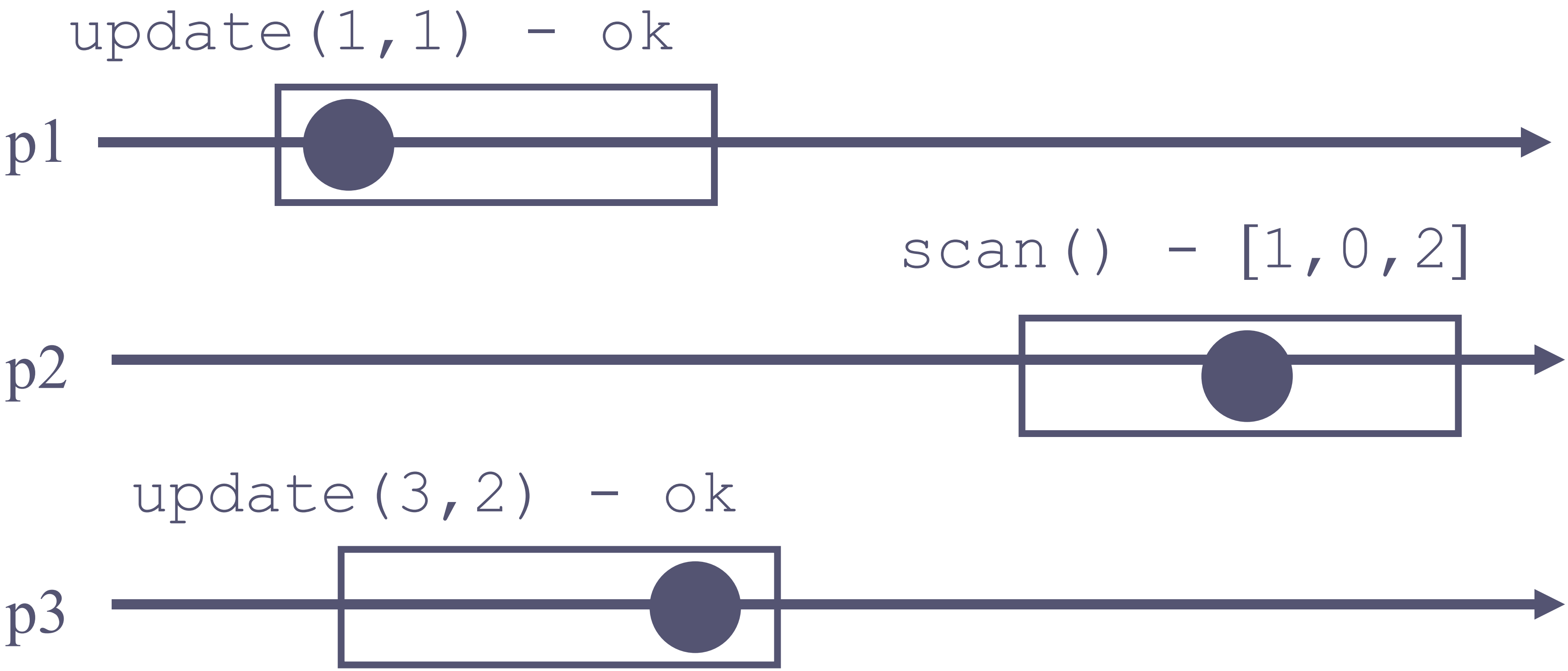
Snapshot: Specification

- A *snapshot* has operations *update()* and *scan()*; it maintains an array *x* of size *N*
- *scan():*
  - return(x)
- *update(i,v):*
  - x[i] := v;
  - return(ok)

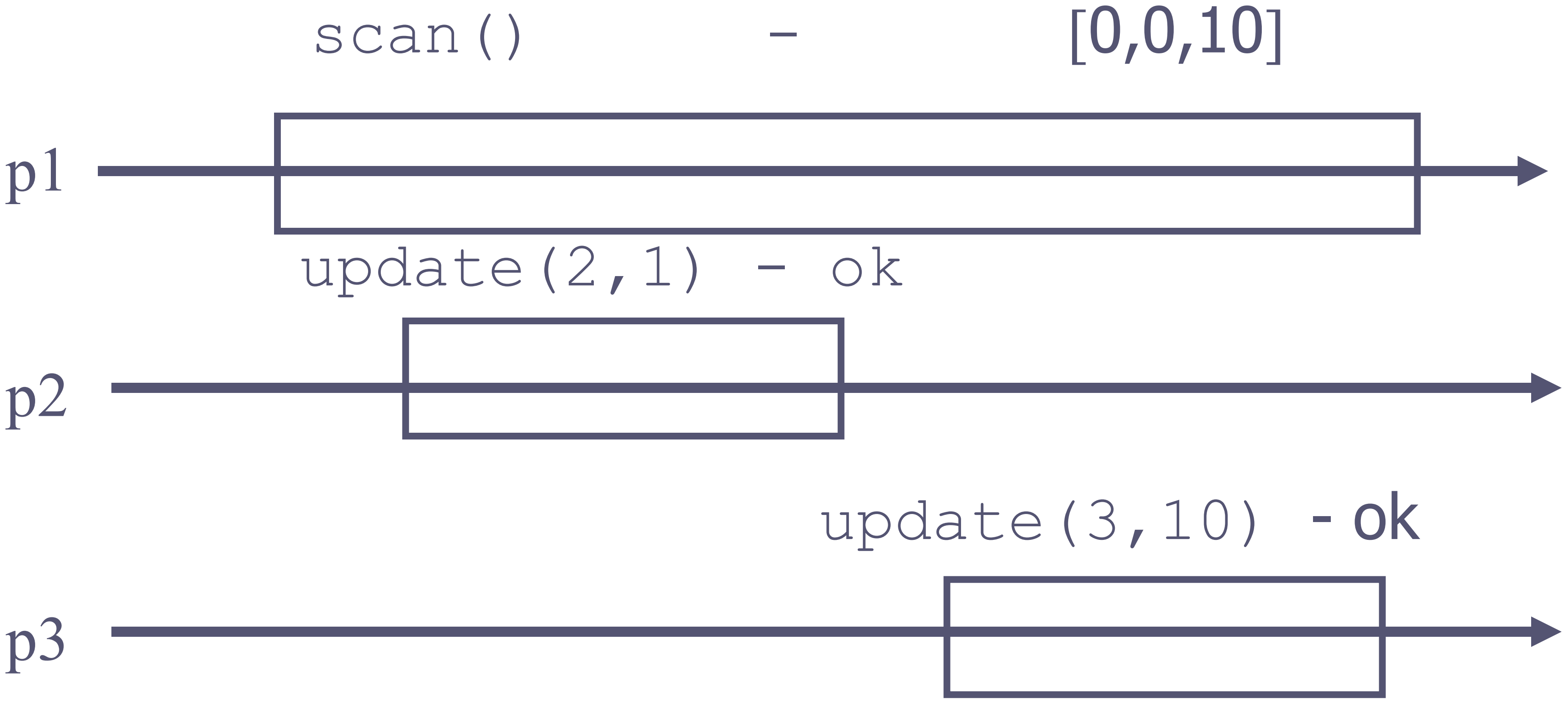
Algorithm?

- The processes share one array of N registers Reg[1,..,N]
- *scan():*
  - for j = 1 to N do
    - x[j] := Reg[j].read();
  - return(x)
- *update(i,v):*
  - Reg[i].write(v); return(ok)

Atomicity?



Atomicity?



Key idea for atomicity

- To *scan*, a process keeps reading the entire snapshot (i.e., *collecting*), until two arrays are the same

Key idea for wait-freedom

- To update, scan then write the value and the scan

- To *scan*, a process keeps collecting and returns a collect if it did not change, or some collect returned by a concurrent *scan*

The Payment Object: Algorithm

- Every process stores the sequence of its outgoing payments in its snapshot location

- To ***pay***, the process scans, computes its current balance: if bigger than the transfer, updates and returns ok, otherwise returns no
- To ***read***, scan and return the current balance

PO can be implemented asynchronously

Consensus number of PO is 1

Consensus number of PO(k) is k

Message Passing ⇔ Shared Memory Modulo Samples

⇔

Roadmap

- (1) The main question
  - (2) The bitcoin problem/object
    - (3) The main answer

Payment System (AT2)

- **AT2_S**
  - **AT2_D**
    - **AT2_R**

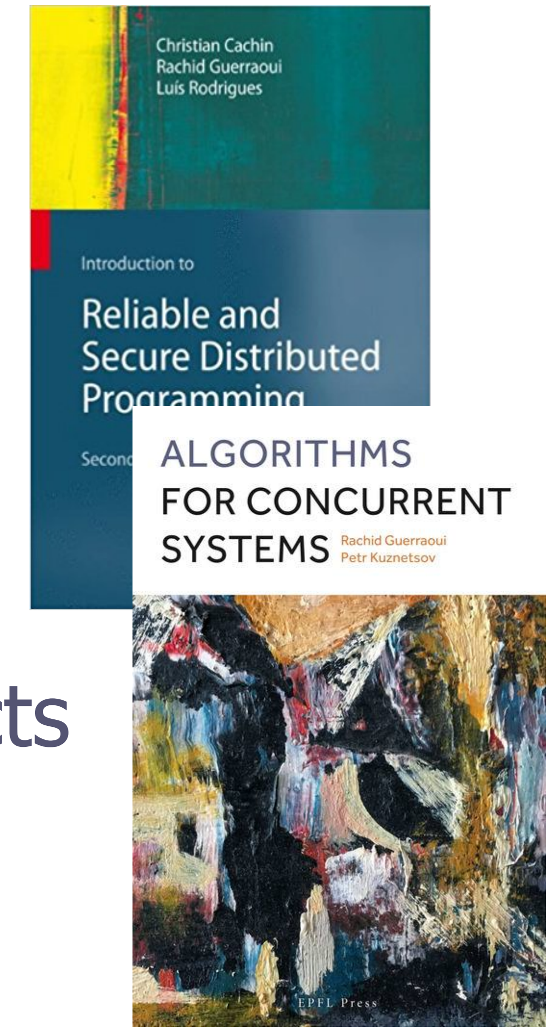
- « The price of software reliability is the pursuit of the utmost simplicity » T. Hoare
- Number of lines of code: one order of magnitude less

References

- (1) The Consensus Number of a Cryptocurrency – ACM PODC 2019– Guerraoui et al
- (2) Scalable Byzantine Reliable Broadcast - DISC 2019 - Guerraoui et al
- (3) Online Payments by Mereley Broadcasting Messages IEEE DSN 2020 - Collin et al

Journey to the center of distributed computing

- Bitcoin
- Blockchain
- Proof of work
- Smart contracts
- Ethereum



- Atomicity
- Wait-freedom
- Snapshot
- Consensus
- Quorums
- Secure Broadcast