Demystifying Bitcoin

Prof R. Guerraoui EPFL

Demystifying

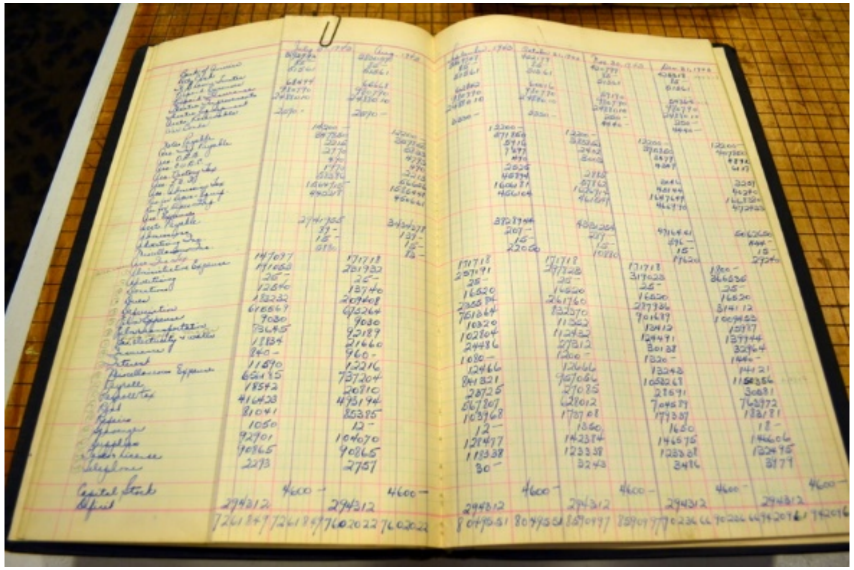
- Bitcoin
- Blockchain
- Ethereum
- Proof of work
- Smart contracts
- Leader
- Consensus
- Broadcast
- Snapshot

Perspectives

- (1) The journalist
- (2) The user
- (3) The participant
- (4) The engineer
- (5) The scientist

(1) The Journalist

- 2008: Financial crisis – Nakamoto (1/21m)
  - From 1c to 10000$ through 20000$ (16.000$)
- From trading hardware to general trading
- 2014: Ethereum (CH) - Now 555 $

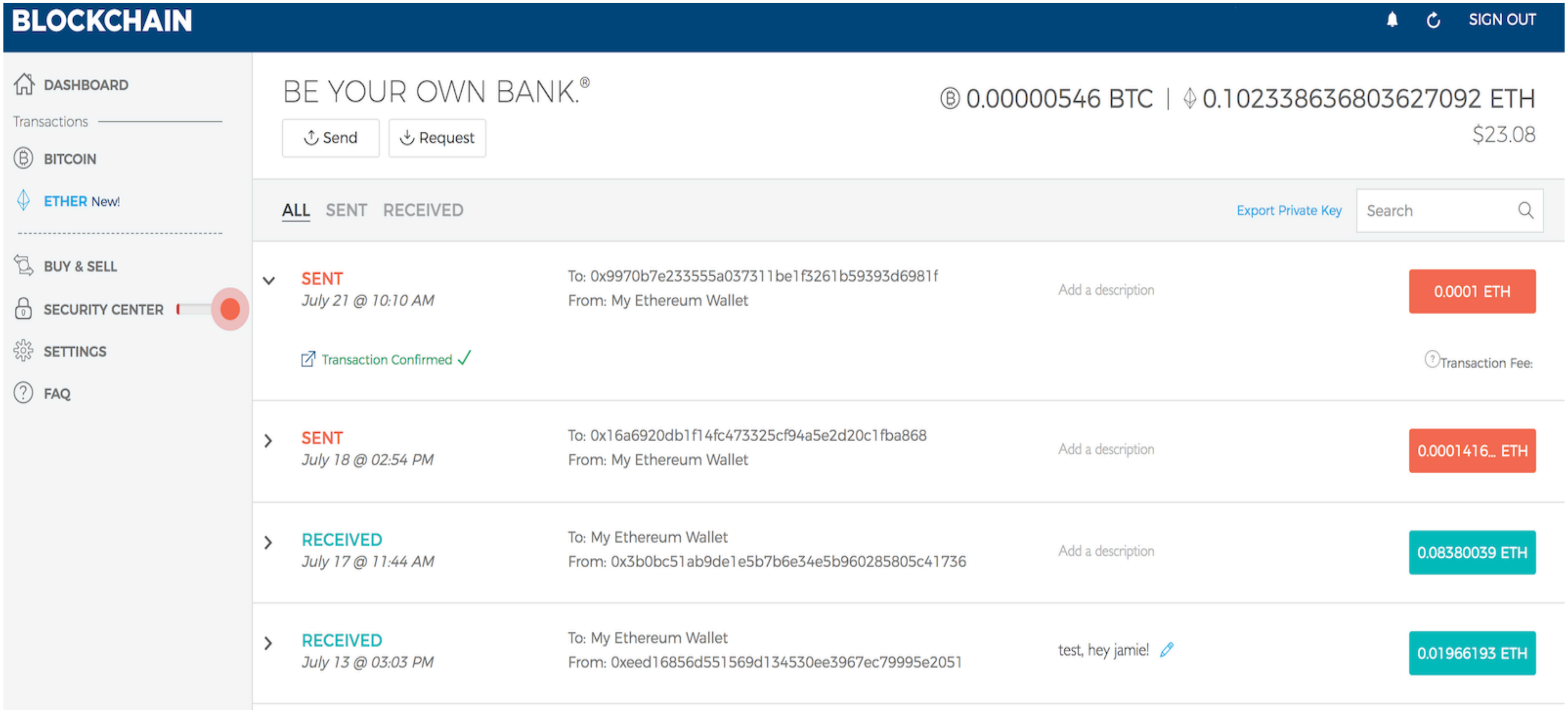


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Perspectives

- (1) The journalist
  - (2) The user
    - (3) The participant
      - (4) The engineer
        - (5) The scientist

(2) The User


BLOCKCHAIN
SIGN OUT
DASHBOARD
Transactions
BITCOIN
ETHER New!
BUY & SELL
SECURITY CENTER
SETTINGS
FAQ
BE YOUR OWN BANK.®
Send
Request
0.00000546 BTC | 0.102338636803627092 ETH
$23.08
ALL
SENT
RECEIVED
Export Private Key
Search
SENT
July 21 @ 10:10 AM
To: 0x9970b7e233555a037311be1f3261b59393d6981f
From: My Ethereum Wallet
Add a description
0.0001 ETH
Transaction Confirmed
Transaction Fee:
SENT
July 18 @ 02:54 PM
To: 0x16a6920db1f14fc473325cf94a5e2d20c1fba868
From: My Ethereum Wallet
Add a description
0.0001416... ETH
RECEIVED
July 17 @ 11:44 AM
To: My Ethereum Wallet
From: 0x3b0bc51ab9de1e5b7b6e34e5b960285805c41736
Add a description
0.08380039 ETH
RECEIVED
July 13 @ 03:03 PM
To: My Ethereum Wallet
From: 0xeed16856d551569d134530ee3967ec79995e2051
test, hey jamie!
0.01966193 ETH

(2) The User

- The wallet: 1 private key + several public keys
- Transaction validation
  - Signing + gossiping + mining + chaining
- Transaction commitment
  - After time t: thousands of users have seen it

(3) The Participant

Honey, I'm home!

I found a block today!

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Miner Jack

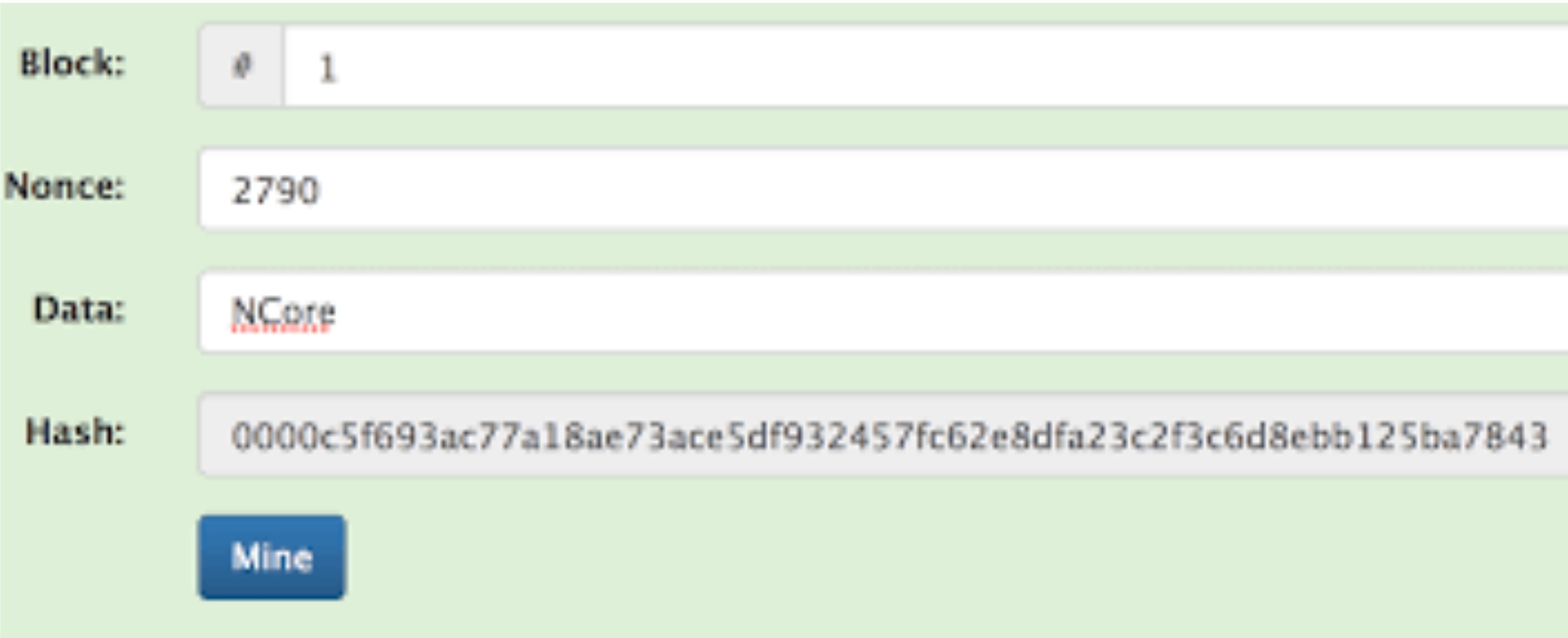
P vs NP (Nash/GV 50 – Ford 70)

$$? * ? = 91$$

$$7 * 13 = ?$$

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

(3) The Participant



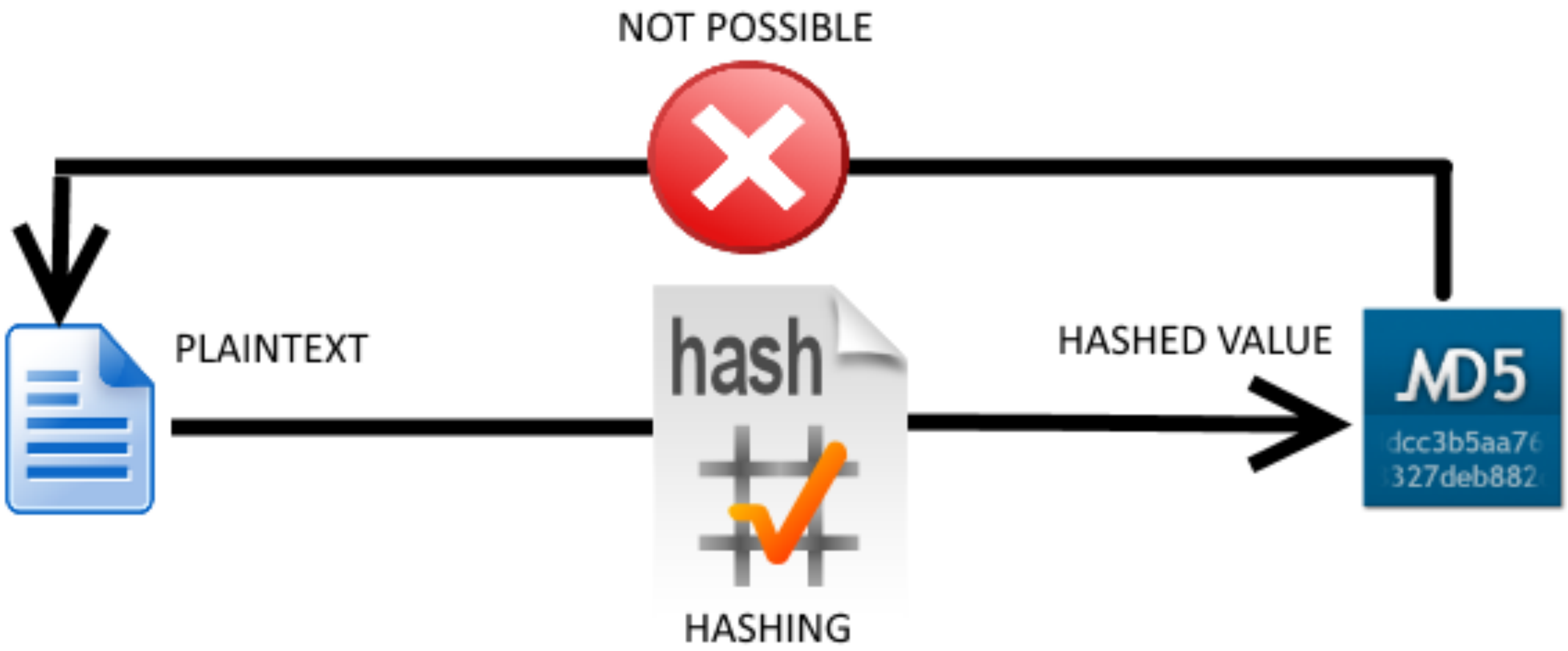
(3) The Participant

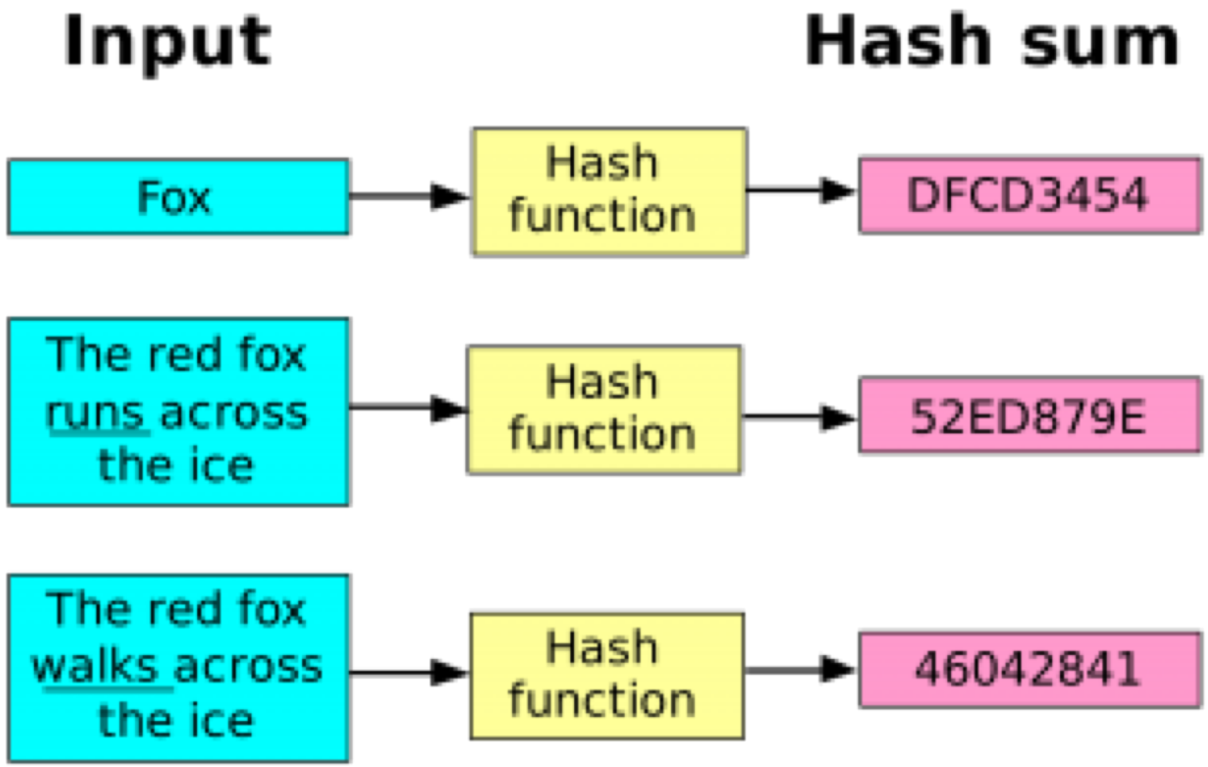
- To validate a transaction, a miner has to solve a puzzle including it
  - Fairness and cooperation
- Incentive: 6.25 bitcoins / puzzle
  - 50 bitcoins 3 years ago
- Total: 21 millions bitcoins
  - Now: 18 millions

(4) The Engineer

- Joinning (a P2P network)
  - Signing (a transaction)
    - Gossiping (the transaction)
      - Gathering (a block)
        - Mining (proof of work - nonce)
          - Chaining (hash)
            - Gossiping (the block)
              - Committing/Aborting

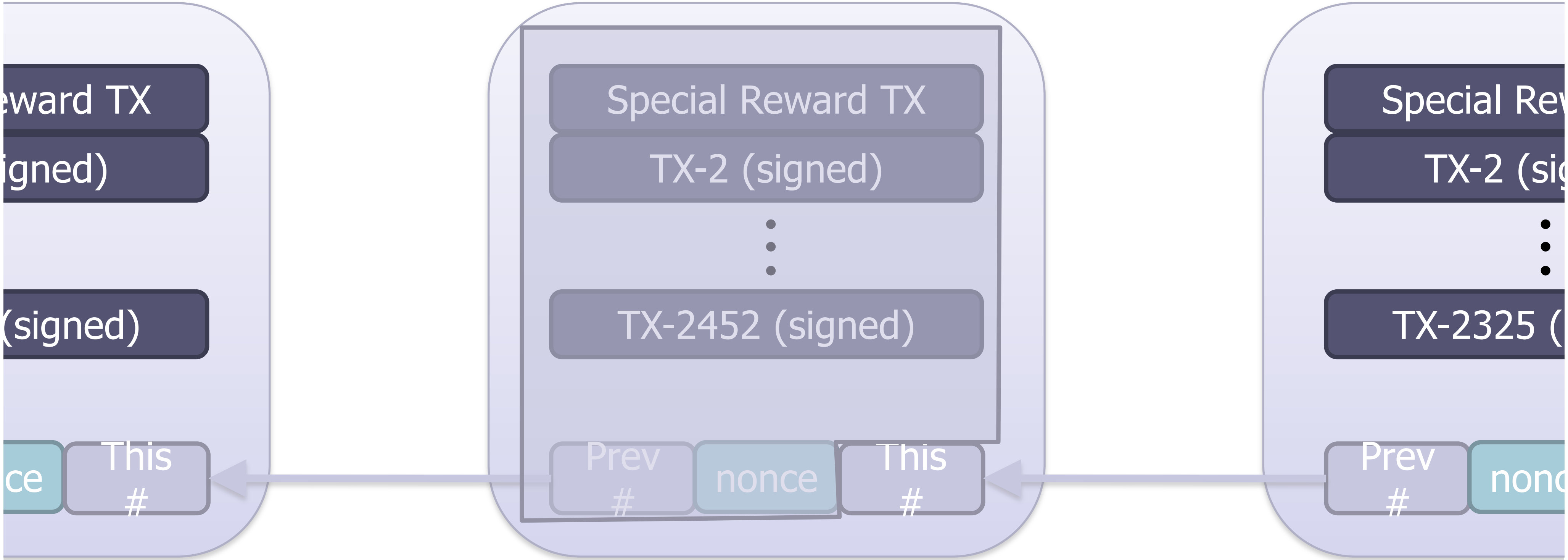
Hashing


NOT POSSIBLE
PLAINTEXT
hash
HASHED VALUE
MD5
dcc3b5aa76
327deb882
HASHING

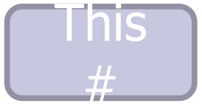

Input
Hash sum
Fox
Hash function
DFCD3454
The red fox runs across the ice
Hash function
52ED879E
The red fox walks across the ice
Hash function
46042841

The Big Picture

Bitcoin block



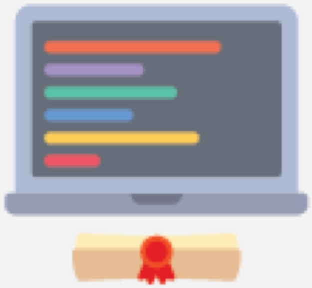
Mining: find 

such that 

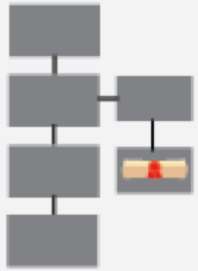
< d

How? By trying different nonces (brute force)

Smart Contracts



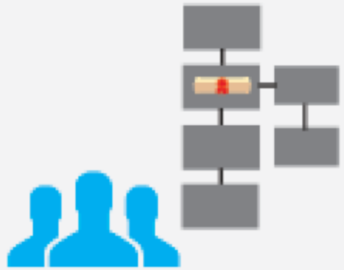
Option contract written as code into a blockchain.



Contract is part of the public blockchain.

Parties involved in the contract are anonymous.

Contract executes itself when the conditions are met.



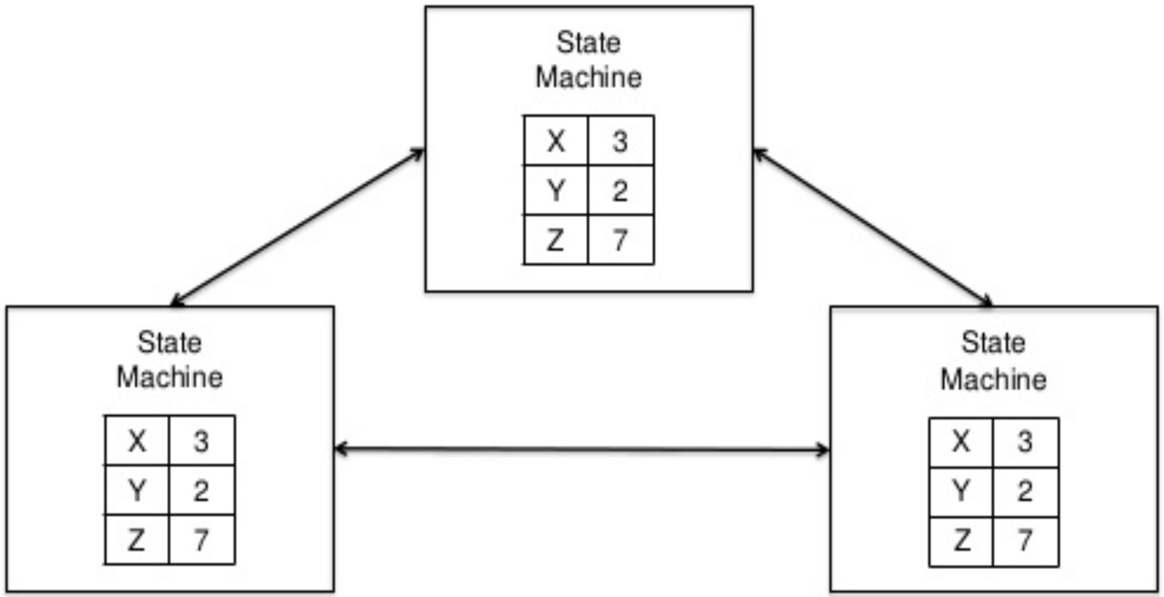
Regulators use blockchain to keep an eye on contracts.

Perspectives

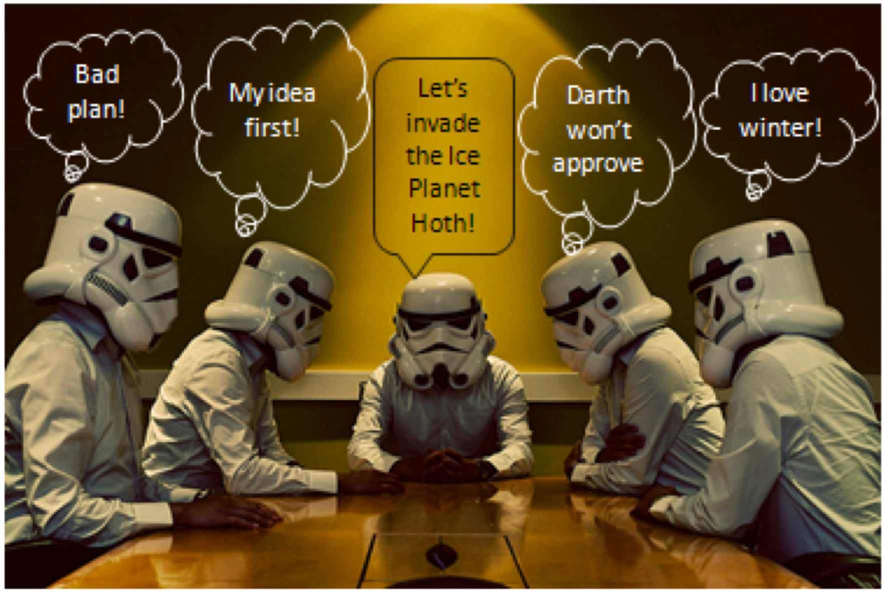
- (1) The journalist
  - (2) The user
    - (3) The participant
      - (4) The engineer
        - (5) The scientist

(5) The Scientist
State Machine Replication (78)

Basic consensus



Consensus Universality (78)



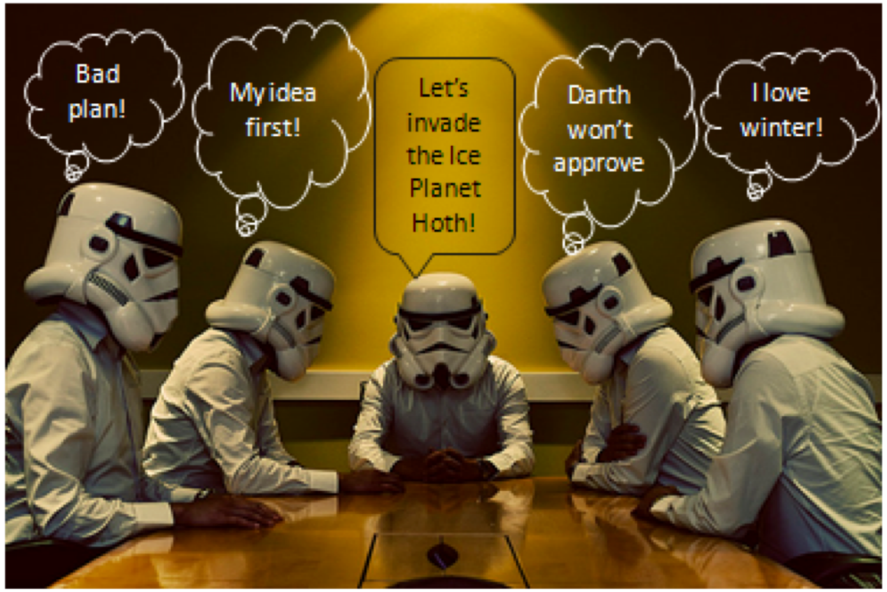
Safety: No two nodes must choose different values.

The chosen value must have been proposed by a node.

Liveness: Each node must eventually choose a value.

Every service can be implemented in a highly available manner using Consensus

Consensus Impossibility (84)



Consensus is impossible in an asynchronous system

- **X000 implementations**

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

Payment System

Can we implement a payment system asynchronously?

P vs NP

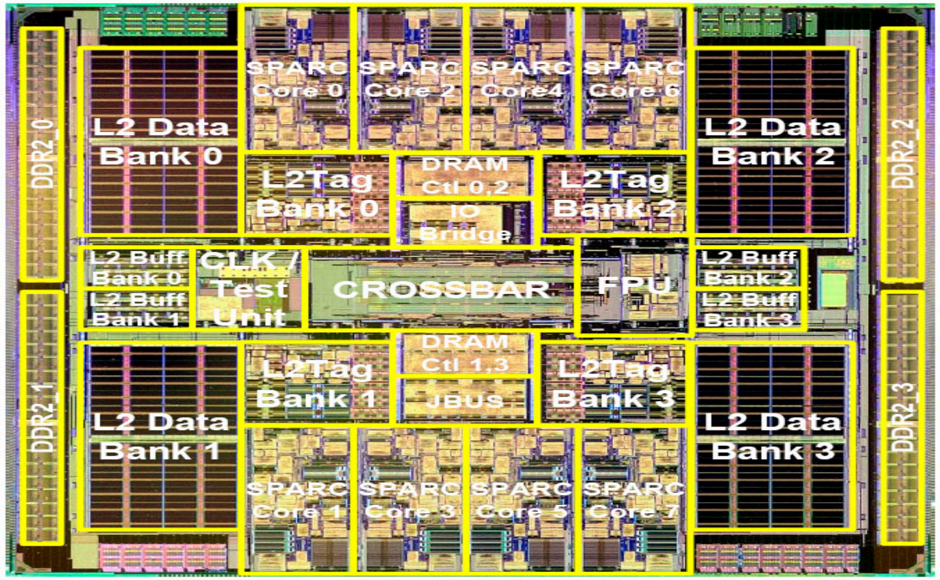
$7 * 13 = ?$ $? * ? = 91$

Asynchronous vs Synchronous

Is payment an asynchronous problem?

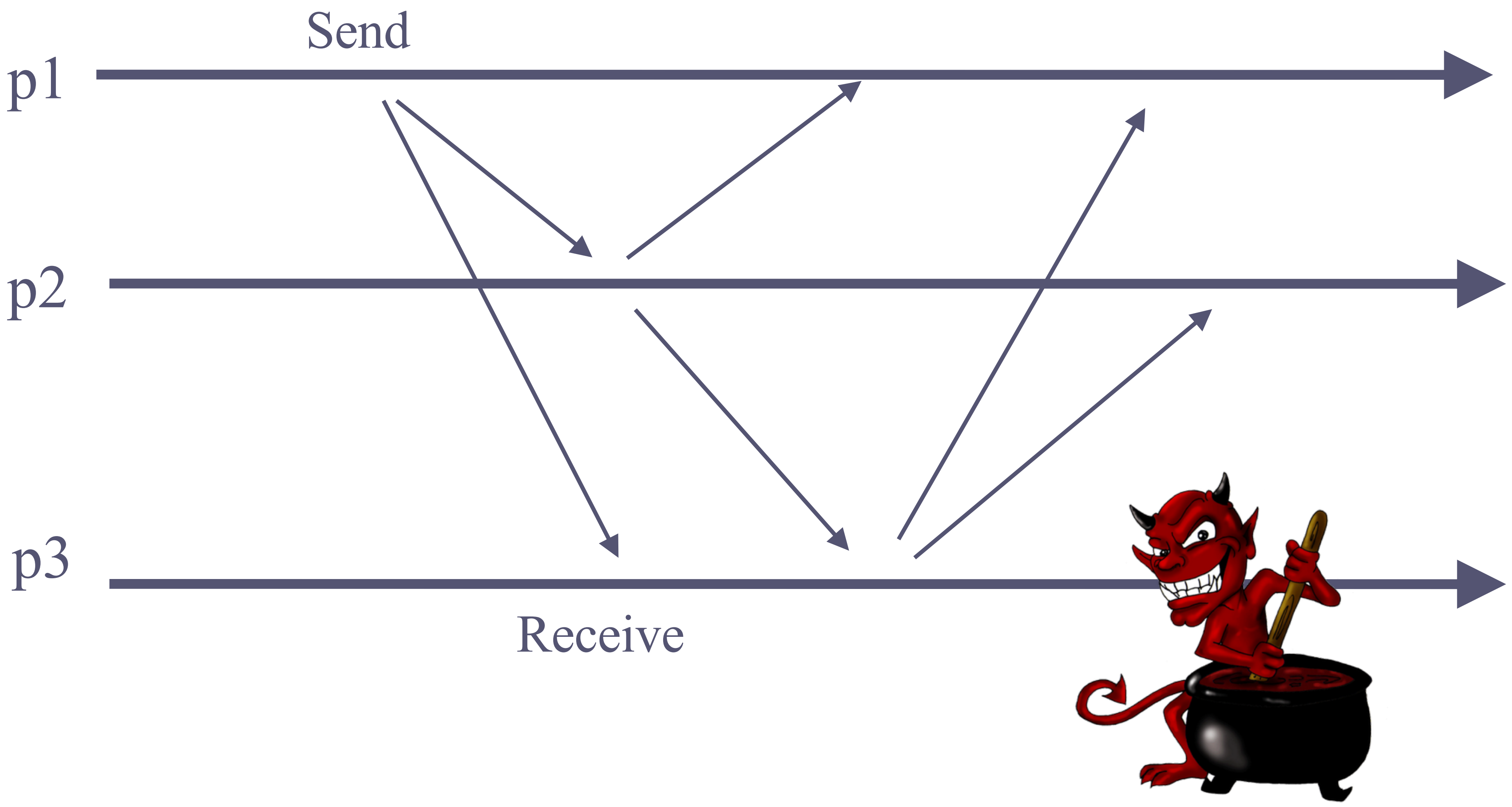
- « To understand a distributed computing problem: bring it to shared memory » T. Lannister

The infinitely big

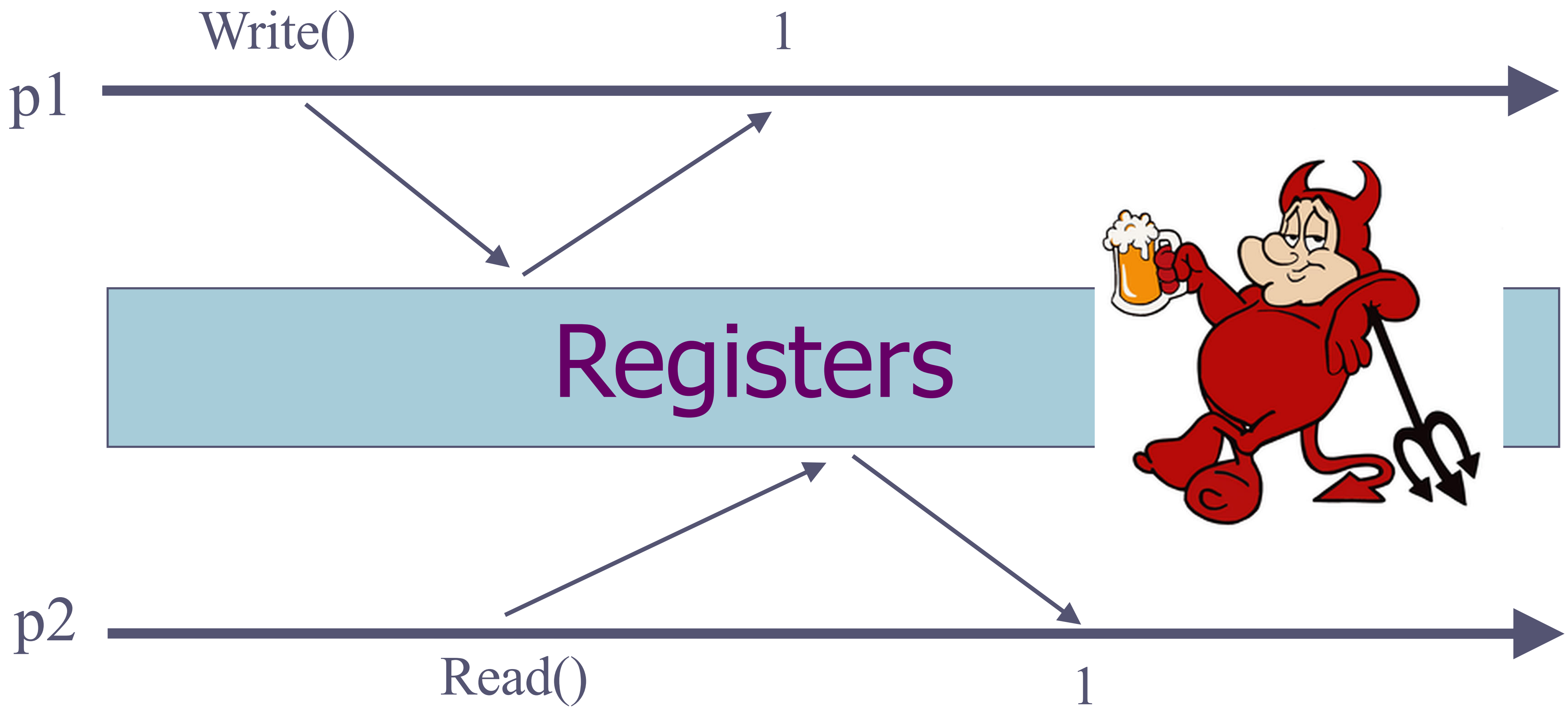


The infinitely small

Message Passing

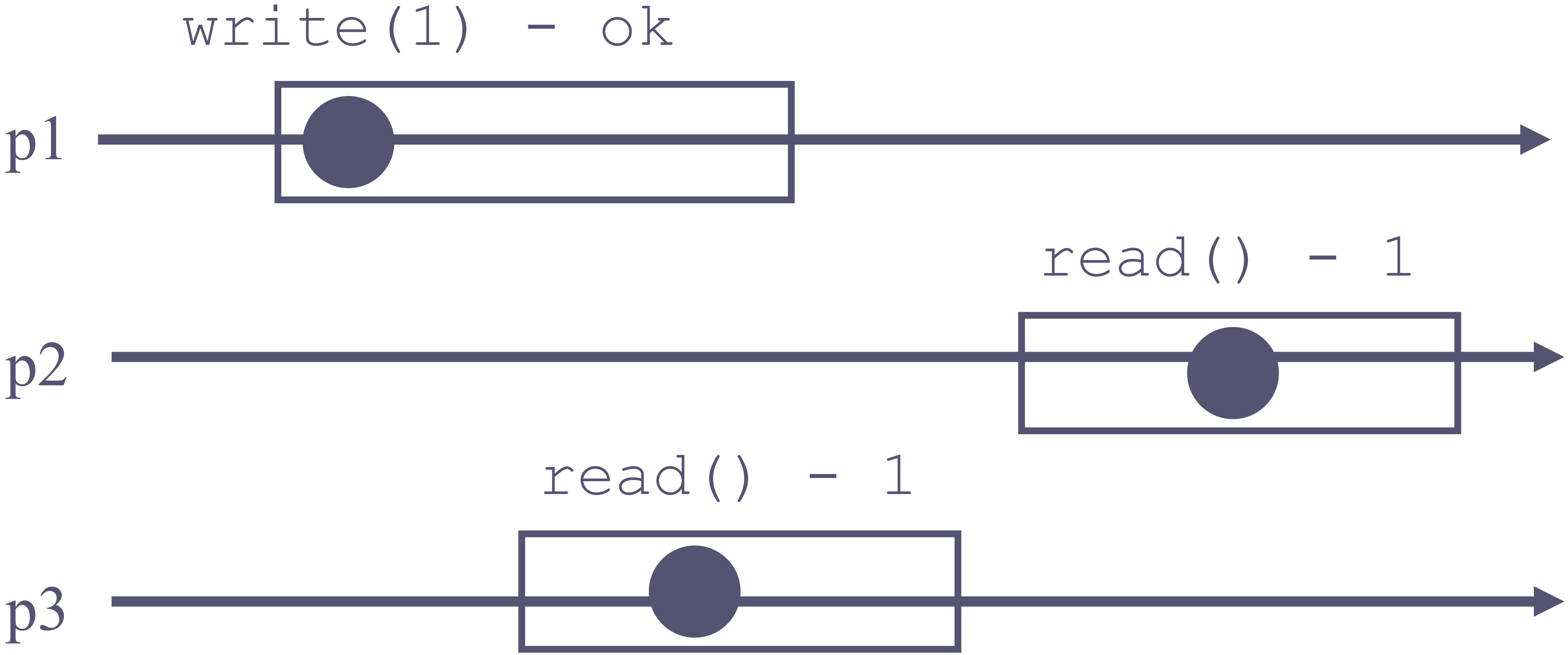


Shared Memory

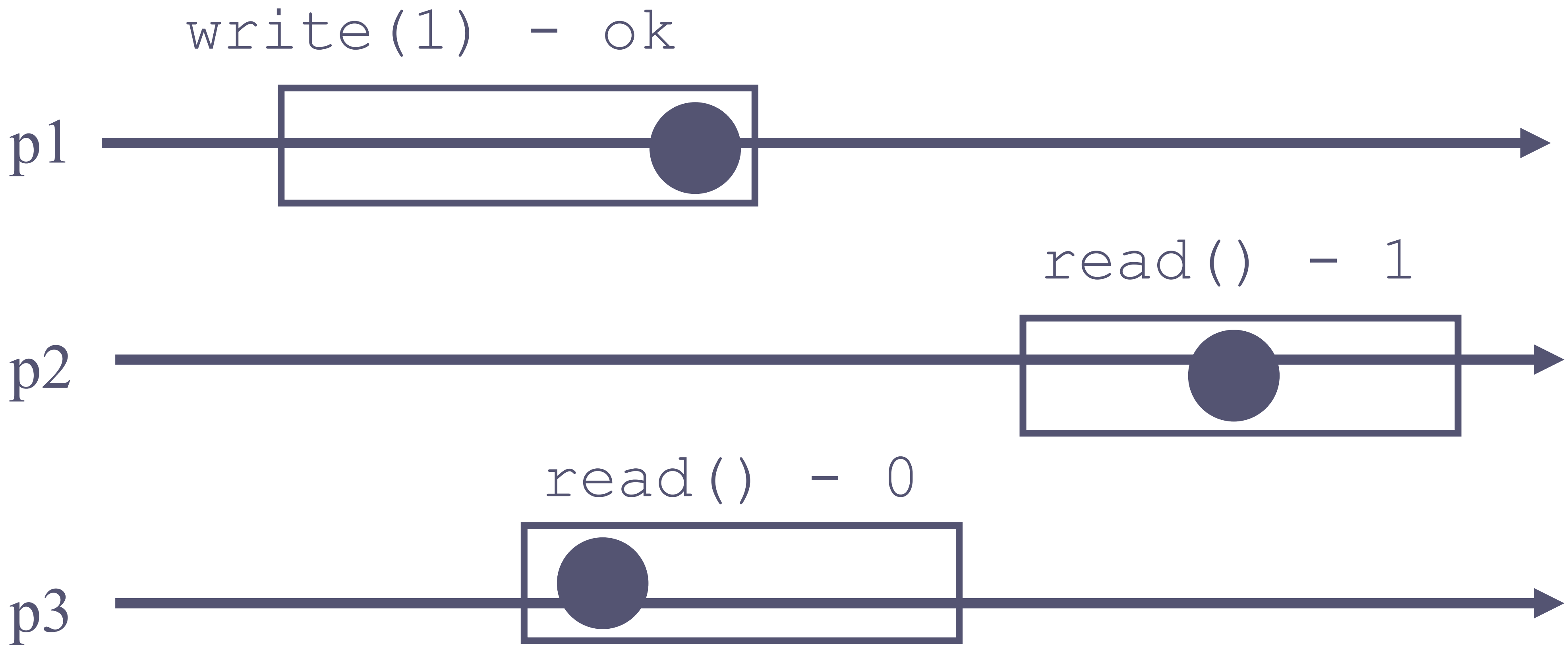


⇔ Message Passing

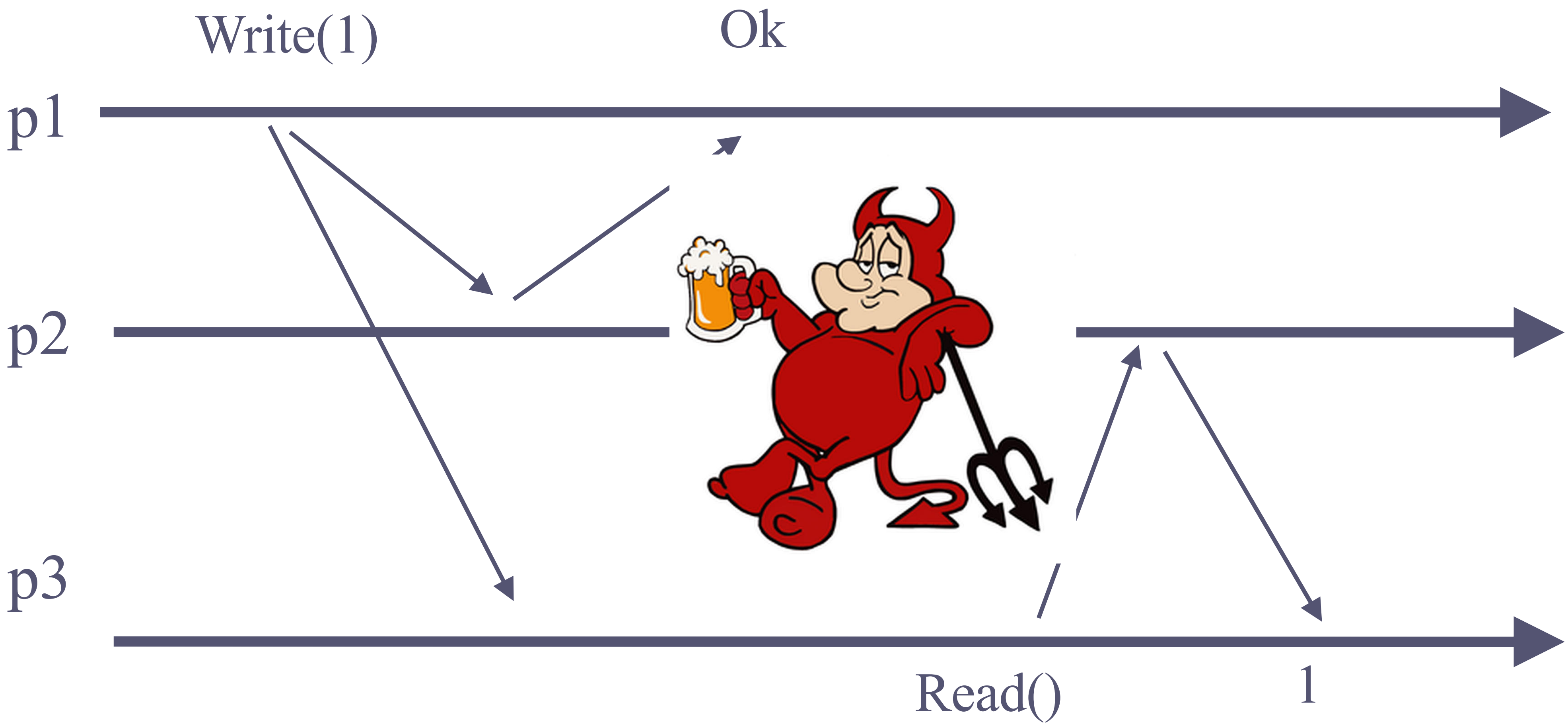
Atomic Shared Memory



Atomic Shared Memory

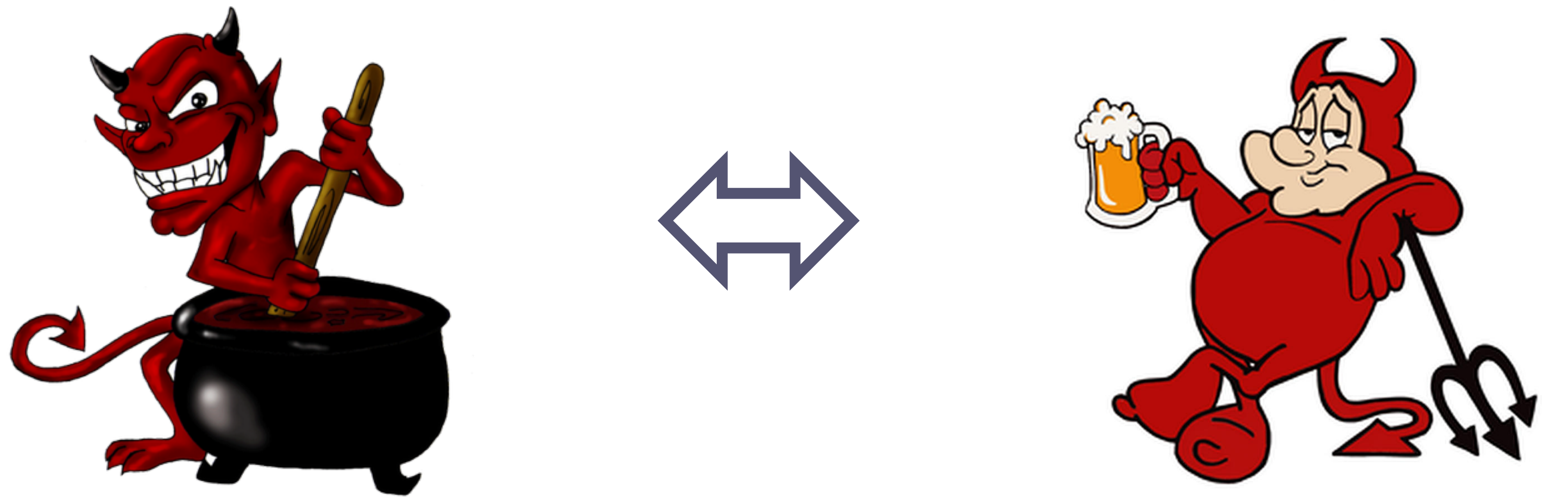


Message Passing ⇔ Shared Memory



Quorums (asynchrony)

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

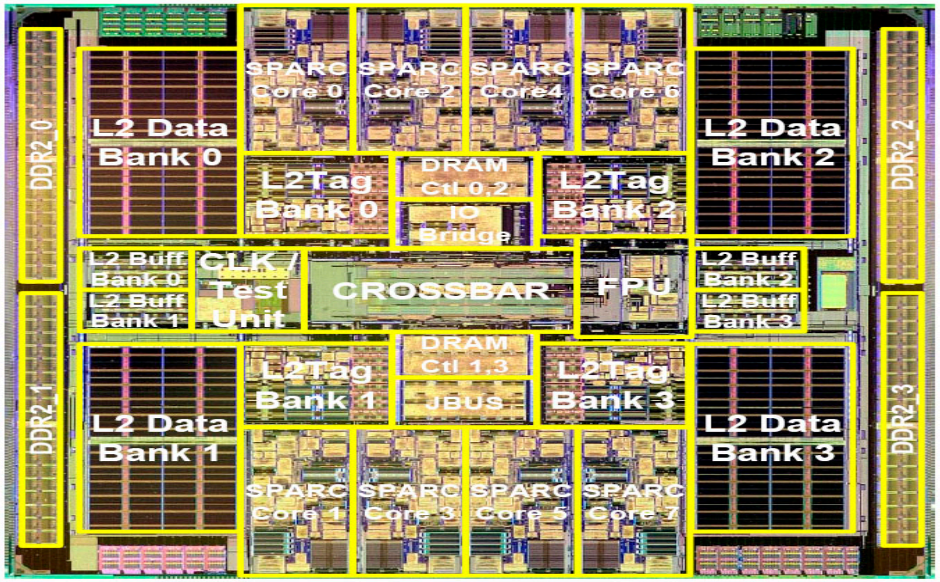


- « Optimization is the source of all evil » D. Knuth

P vs NP

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

Asynchronous vs Synchronous



Payment System

- Atomicity
- Wait-freedom

Can we implement a payment system asynchronously?

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?

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())

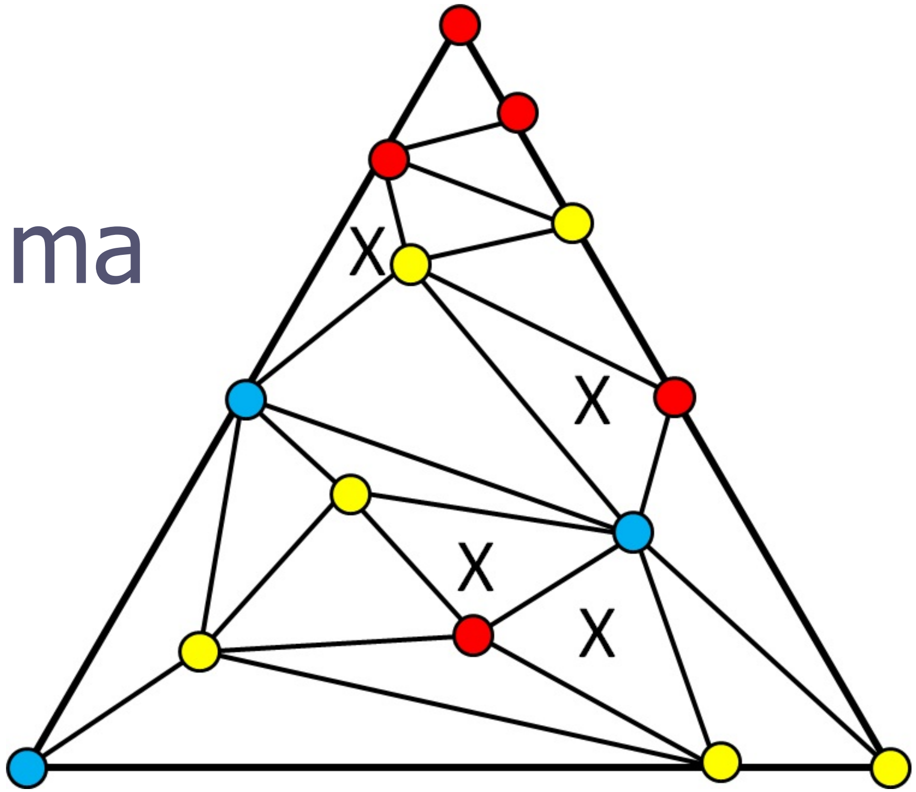
Impossibility [FLP85,LA87]

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

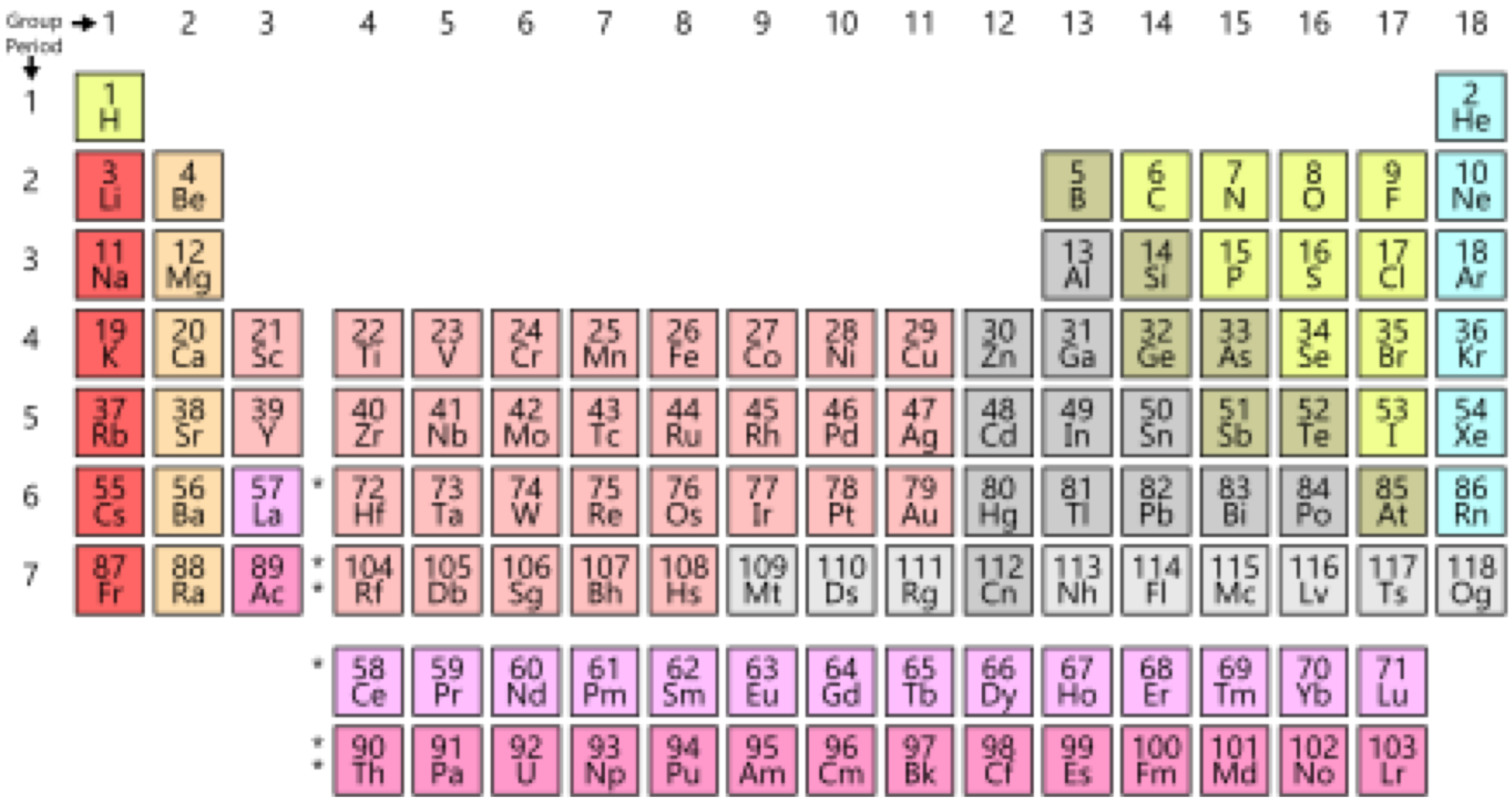
- ***Corollary**:* no asynchronous algorithm implements Counter* among two processes using *registers*

- ***Theorem:*** no *asynchronous* algorithm implements *set-agreement* using *registers*

Sperner's Lemma



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



Payment Object (PO): Specification

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

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

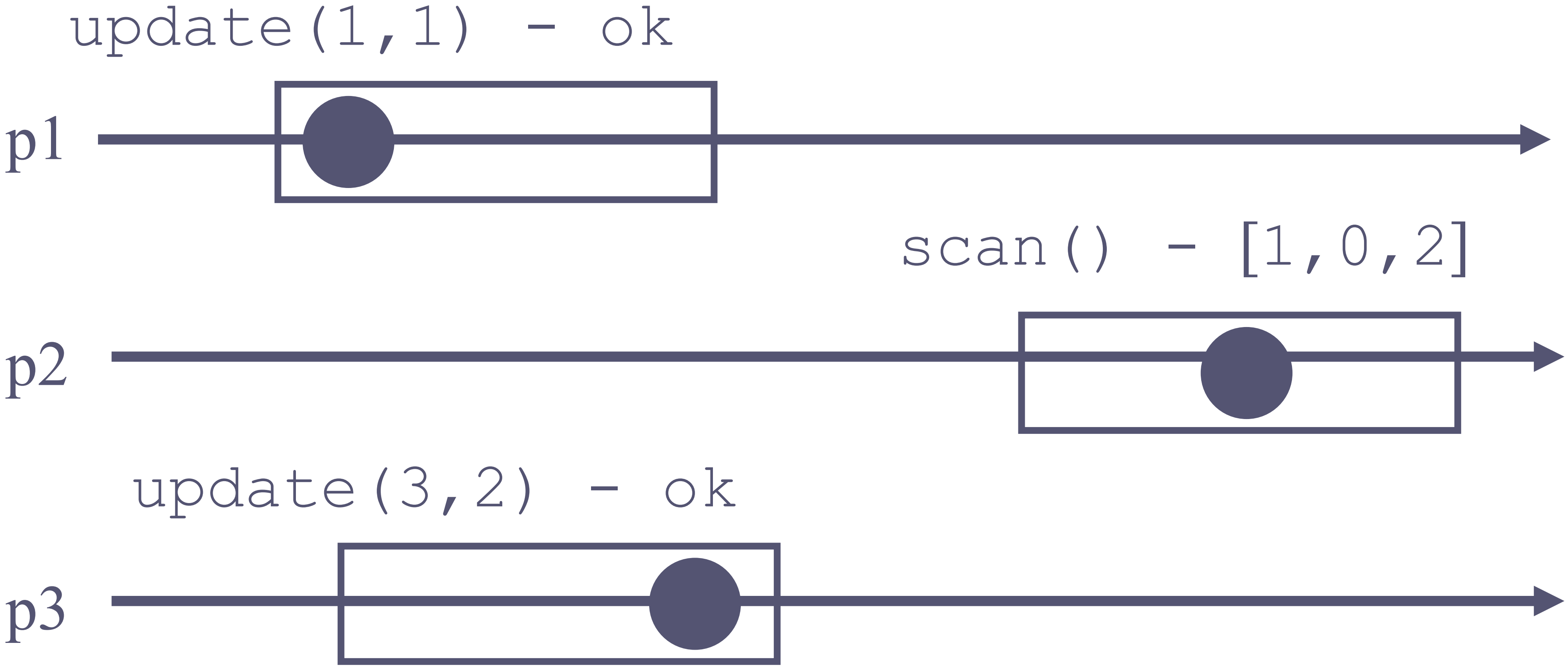
Snapshot: Specification

- 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)

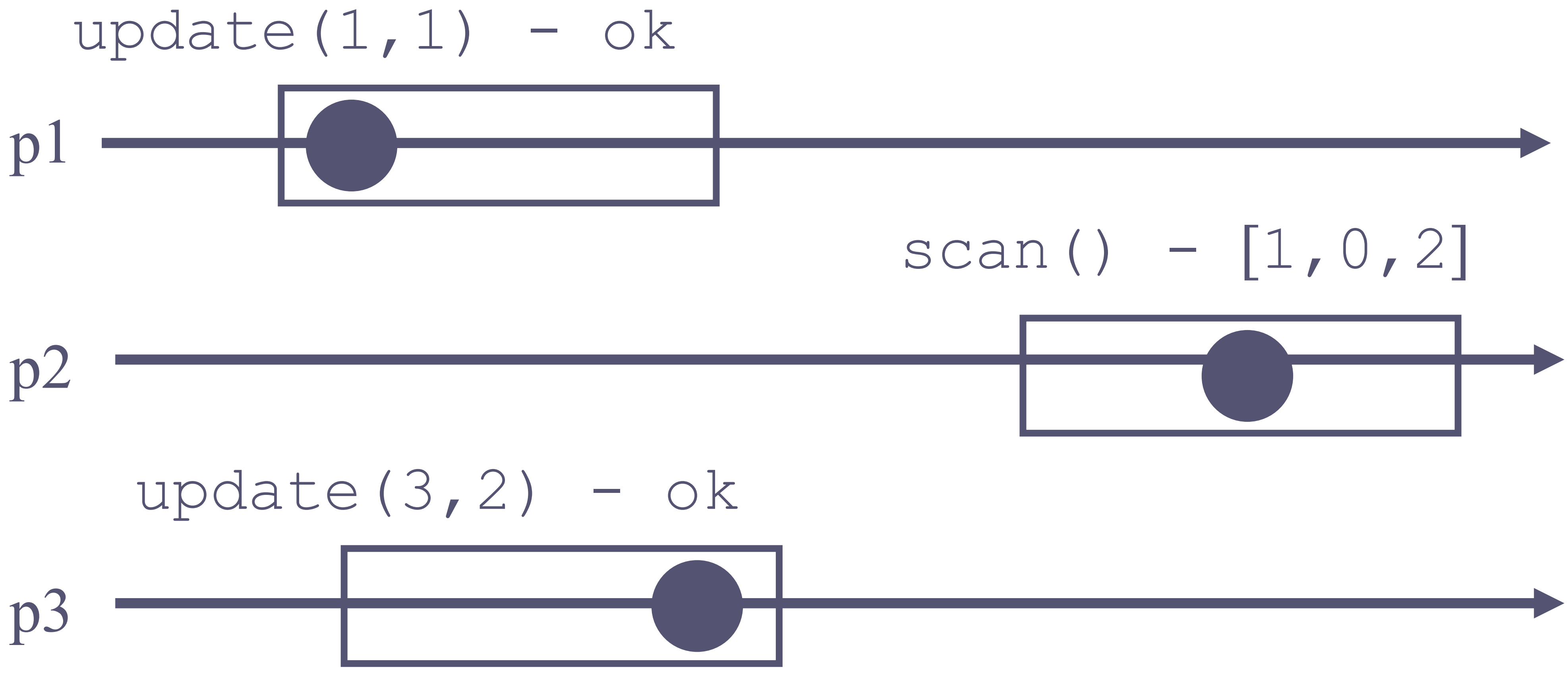
Algorithm?

- 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)

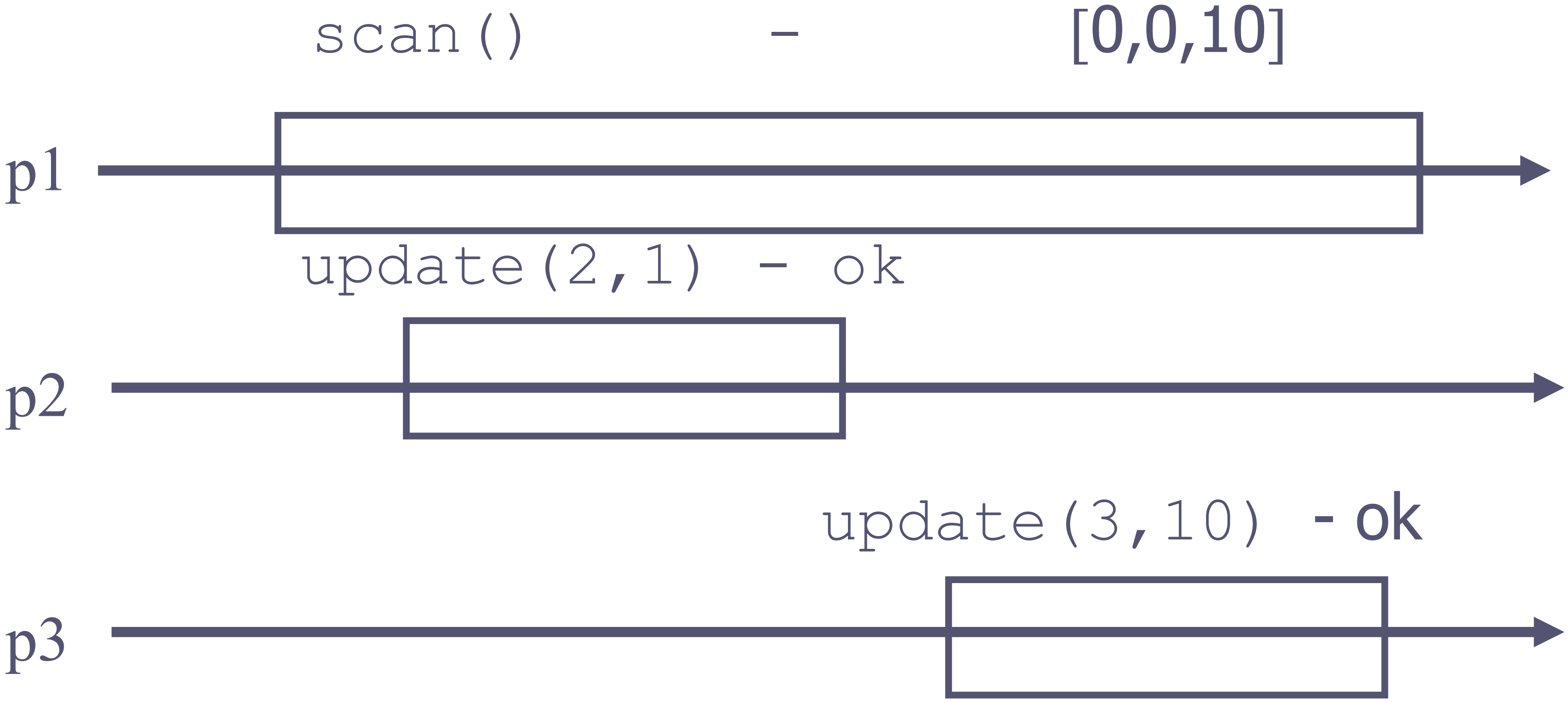
Atomicity?



Atomicity?



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

Payment System (AT2)

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

- Number of lines of code: one order of magnitude less
- Latency: seconds (at most)

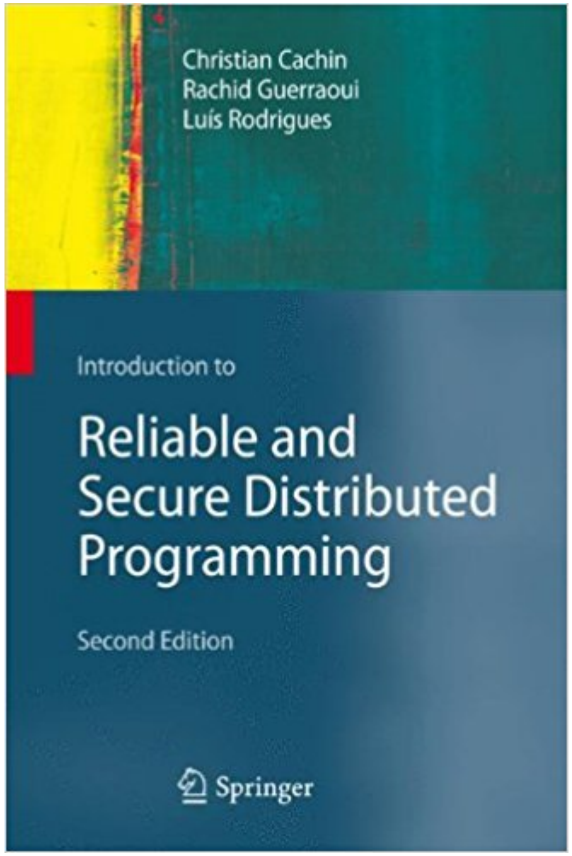
References

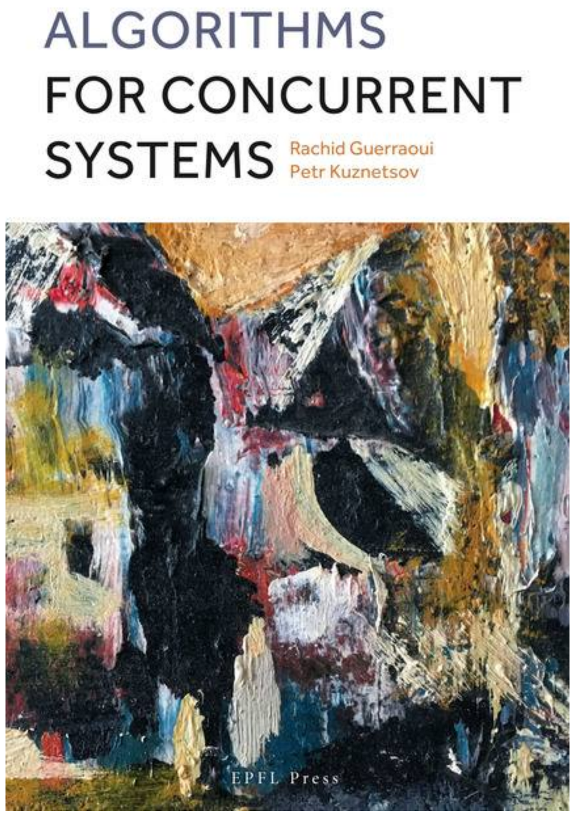
Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Dragos-Adrian Seredinschi: **The Consensus Number of a Cryptocurrency.** PODC 2019: 307-316

Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Dragos-Adrian Seredinschi: **Scalable Byzantine Reliable Broadcast.** DISC 2019: 1-16 **(Best Paper Award)**

Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, Athanasios Xygkis: **Online Payments by Merely Broadcasting Messages.** DSN 2020: 26-38 **(Runner for the Best Paper Award)**

References


Christian Cachin
Rachid Guerraoui
Luís Rodrigues
Introduction to
Reliable and Secure Distributed Programming
Second Edition
Springer


ALGORITHMS
FOR CONCURRENT
SYSTEMS
Rachid Guerraoui
Petr Kuznetsov
EPFL Press