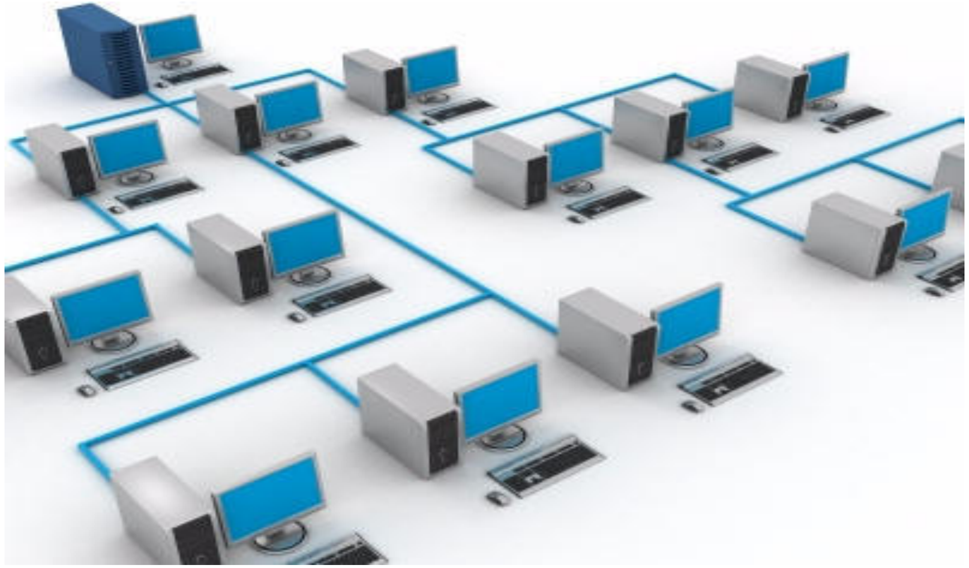
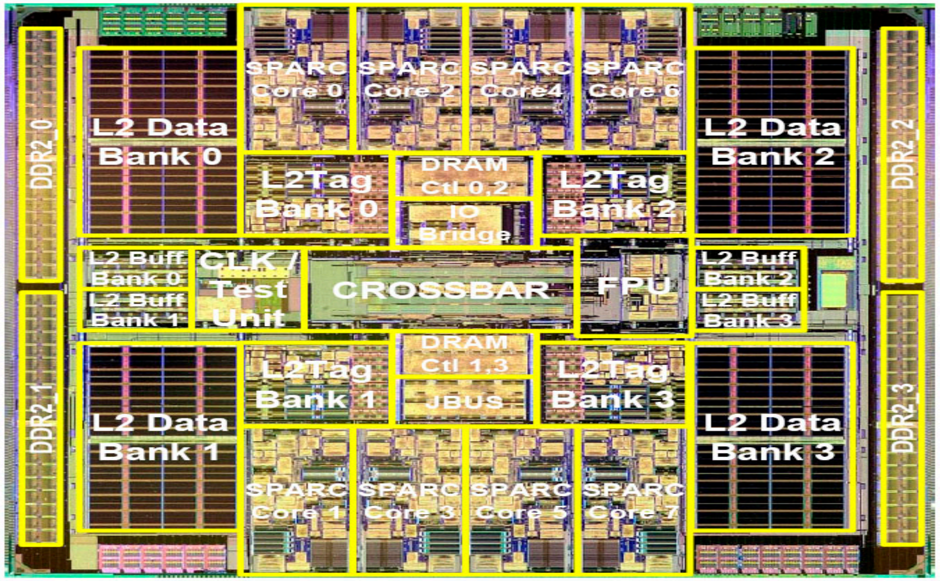
# Journeys to the Center of Distributed Computing

# The Rise and Fall of Distributed Computing

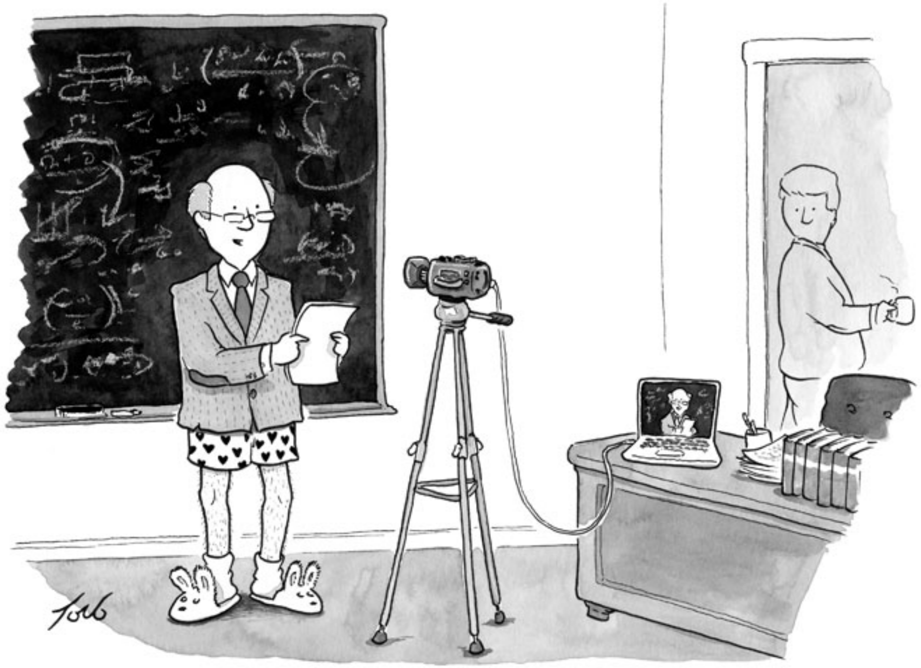
- **The infinitely big**
- **The infinitely small**






# The Rise and Fall of Distributed Computing

# Academics vs. Engineers



"I'm honored to share my research at your virtual academic conference."

# Relevance vs. Innovation

# Distributed Computing Research

# Road to Salvation

**"Puisque ces mystères nous dépassent, feignons d'en être les organisateurs" J. Cocteau**

# Journeys to the Center of DC

Applications (SIFT 1978)

Middleware (FLP 1985)

Hadware (Mutex 1965)


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


# Distributed Payment

## X000 implementations

### Bitcoin: A Peer-to-Peer Electronic Cash System

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

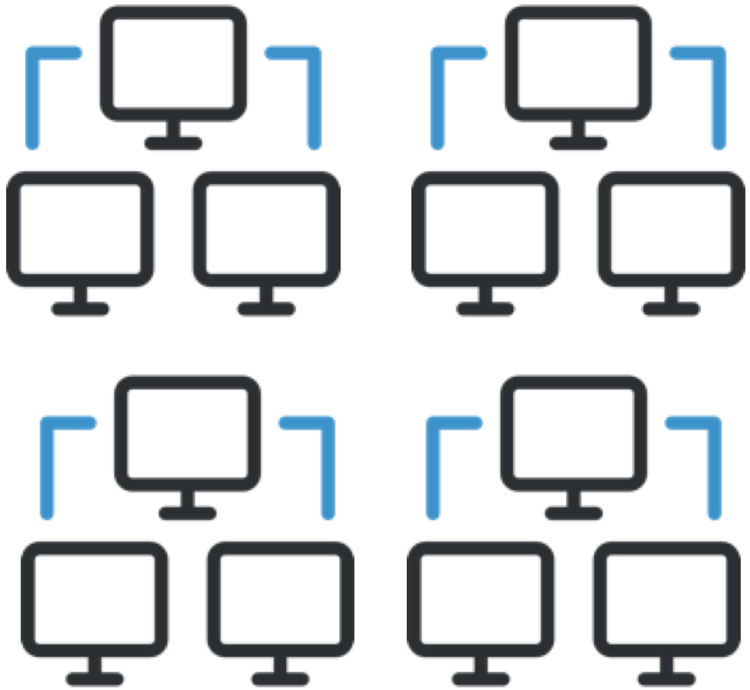
**Abstract.** A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required 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

# P vs NP

# Asynchronous vs Synchronous

## Is payment an asynchronous problem?

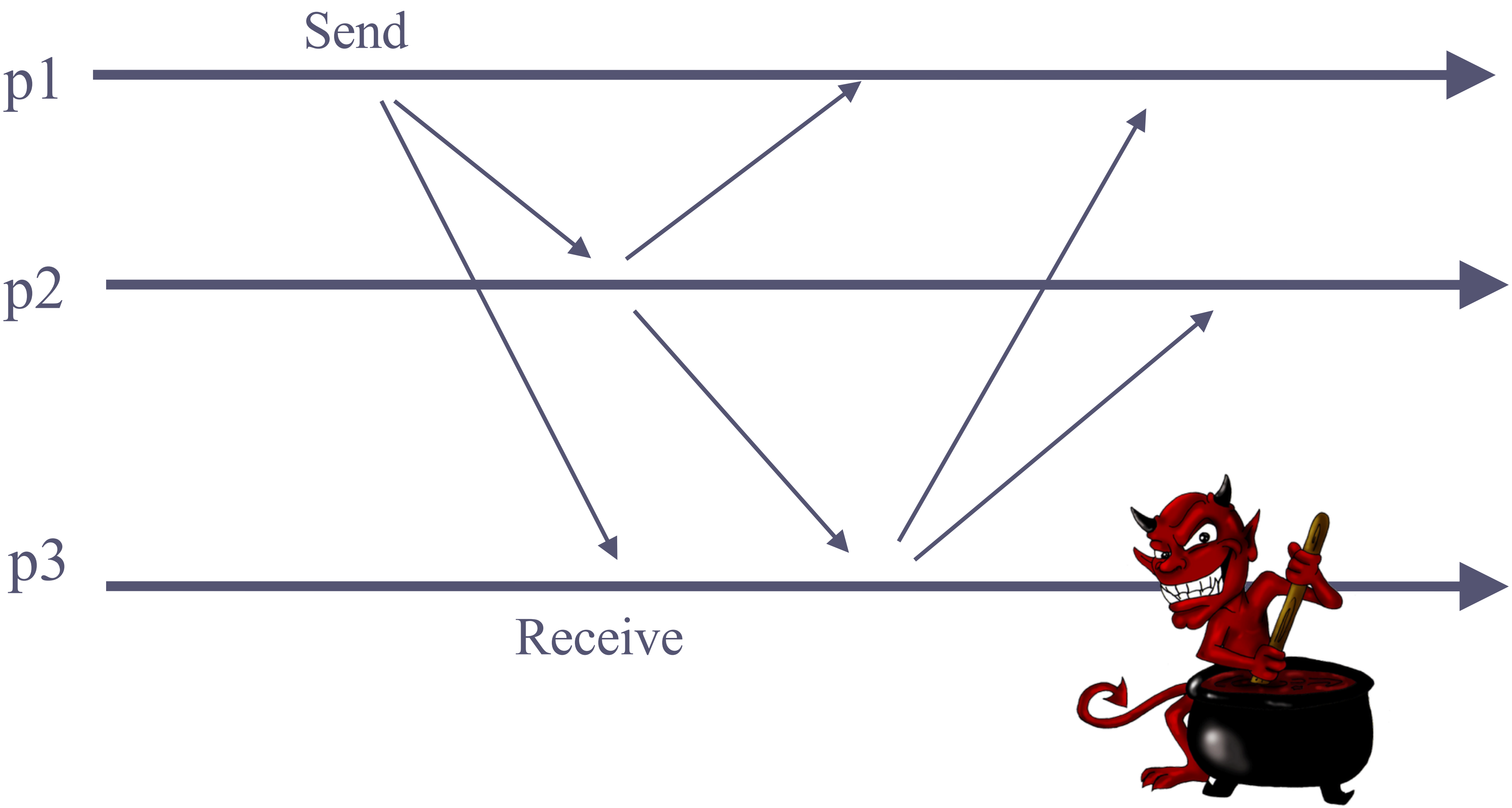
- «  To understand a distributed computing problem: bring it to shared memory » T. Lannister



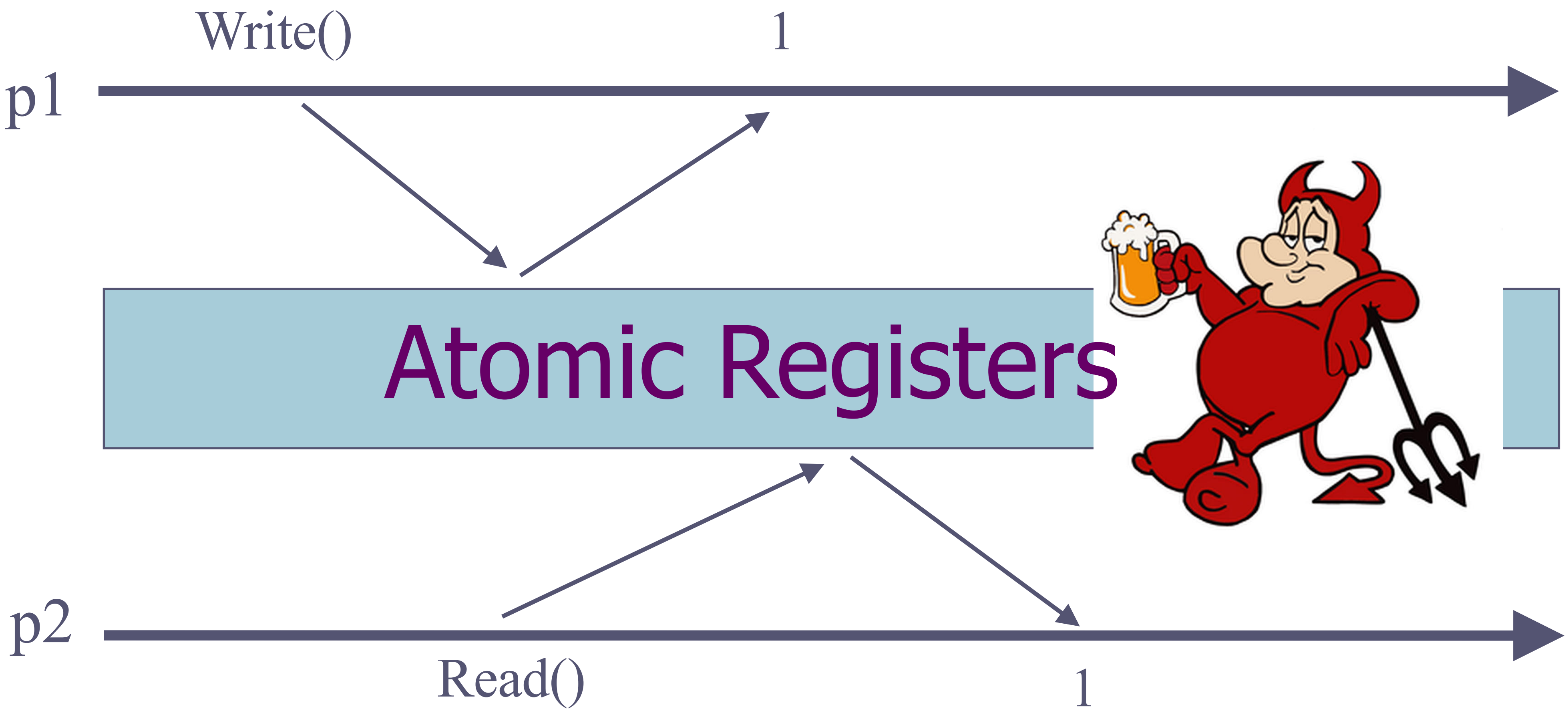
## Message Passing

## Shared Memory

# Message Passing



# Shared Memory



# Message Passing ⇔ Shared Memory Modulo Quorums

⇔

# 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?

# 2-Consensus with Counter*

- Registers R0 and R1 and Counter* C - initialized to 1

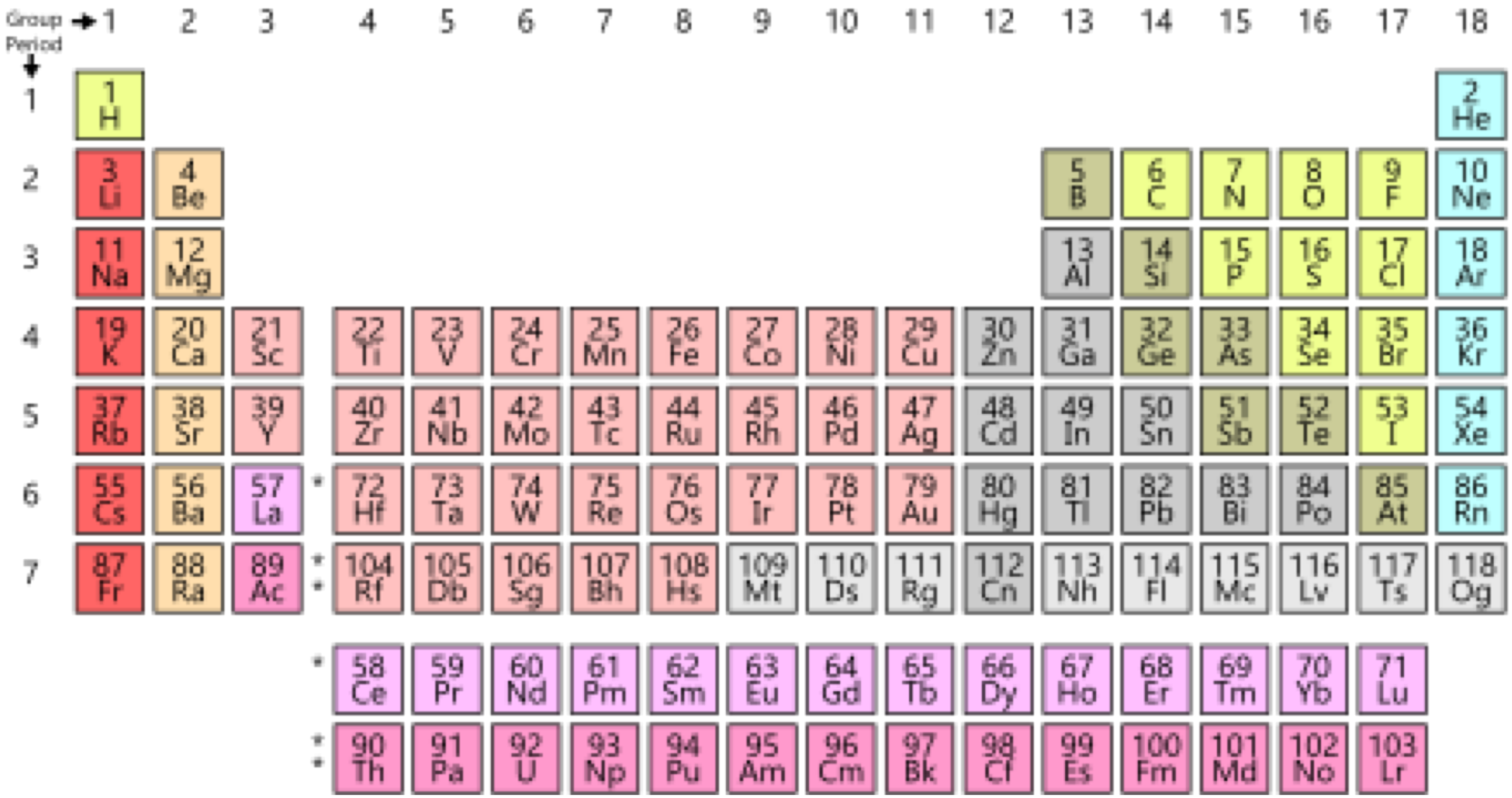
- Process pI:
- propose(vI)
- RI.write(vI)
- res := C.dec()
- if(res = ok) then
  - ✓ return(vI)
  - ✓ else return(R{1-I}.read())

# 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



# 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,*,*)

- 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)

# 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

# Faster and Simpler Payment Systems (AT2)

- AT2_S (PODC 2019)
- AT2_D (DNS 2020)
- AT2_R (DISC 2019)

# Journey to the Center of DC

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

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

# Distributed Tracing

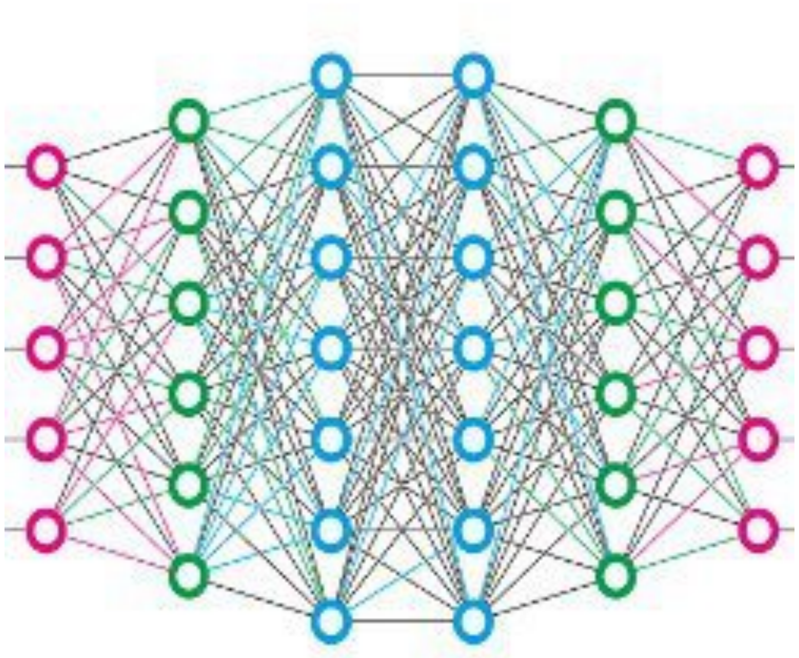
## Gossip Algorithms (DISC 2020)
## Populations Protocols

# Journeys to the Center of DC

**Applications**

**Middleware**

# Distributed ML



**PODC 2020 / ArXiv 2020 / SRDS 2020**

# Folklore & Misunderstandings

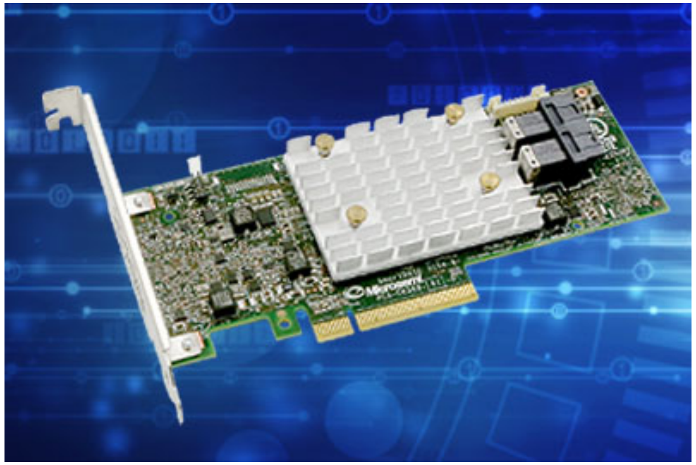
- Distributed systems are synchronous (DC 2018)
- Causal transactions are fast & robust (IPDPS 2020)
- SMR ⇔ Consensus (DISC 2018)

# Journeys to the Center of DC

**Applications**

**Middleware**

**Hardware**



# RDMA

- Remote shared / protected memory
- Consensus with 2f+1 and f+1 (vs 3f+1 and 2f+1) and 2 steps (vs 4 steps) – PODC 2018/2019
- µ: SMR in 1µs / 1ms

# NVRAM

- Persistent objects with durable linearizability / detectable recovery
- Tight bound: 1 pfence per operation (SPAA 2019)
- MCAS with 2 pfences and k+1 CASes per k-Cas (DISC 2020)

# The Rise and Fall of Distributed Computing


## Journeys to the Center of DC

- Applications (SIFT 1978)
- Middleware (FLP 1985)
- Hardware (Mutex 1965)

S. Baehni
R. Boichat
A.Doudou
P. Dutta
M. Kapalka
R. Levy
M. Monod
B. Pochon
J. Spring
P. Eugster
S. Handurukande
P. Kouznetsov
M. Vukolic
G. Losa
A. Dragojevic
V. Buschkov
D. Khozaya
K. Antoniadis
J. Komatovic

G. Damaskinos
M. El Mhamdi
M. Matteo
S. Rouault
A. Guirgis
I. Zablotchi
A. Xygkis
M. Pavlovic
A. Seredinshi
M. Taziki
R. Patra
T. David
M. Yabandeh
D. Alistarh
M. Letia
V. Trigonakis
J. Wang
R. Banabic
N. Knezevic