In search for lost universality

Journey to the Center of Distributed Computing

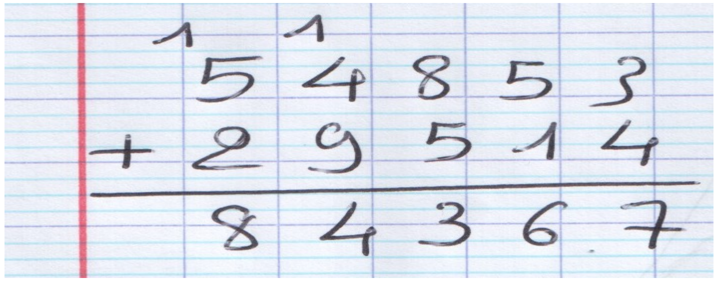
Roadmap

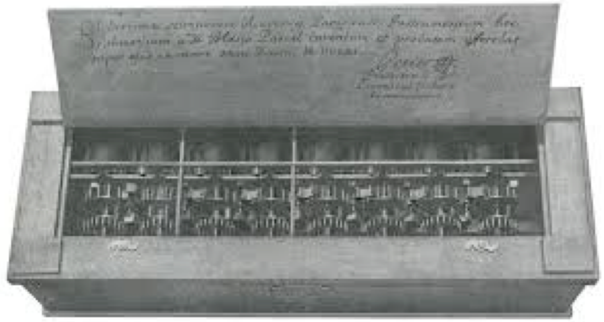
- The lost universality
  - Consensus is necessary but impossible

- The quest for universality
  - Consensus is sufficient

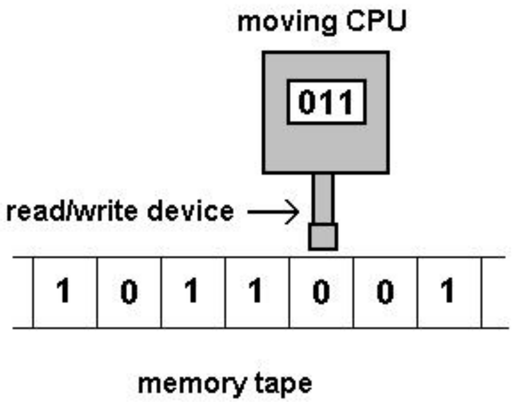
- Circumventing universality

Universality



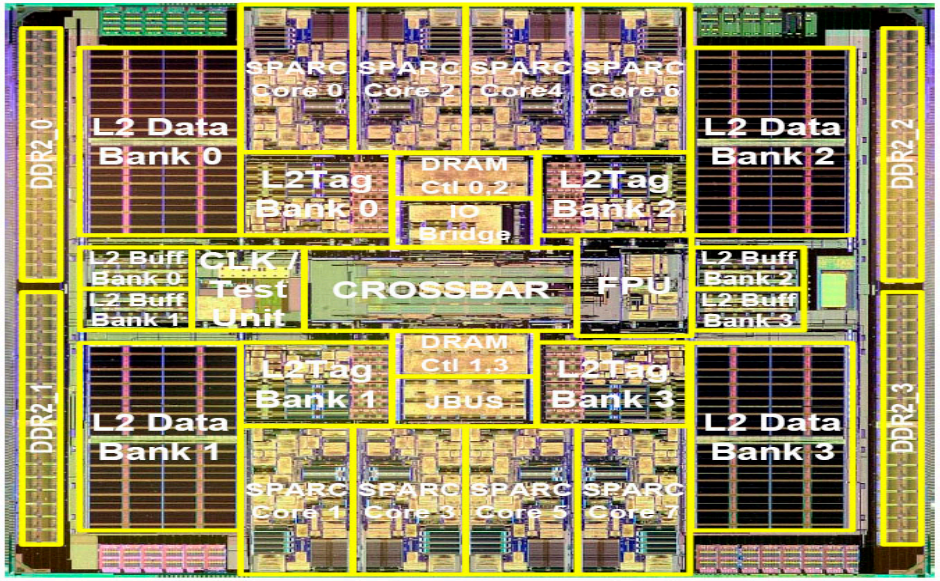


Algorithmi



Turing

The Lost Universality



- The infinitely big
- The infinitely small

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*

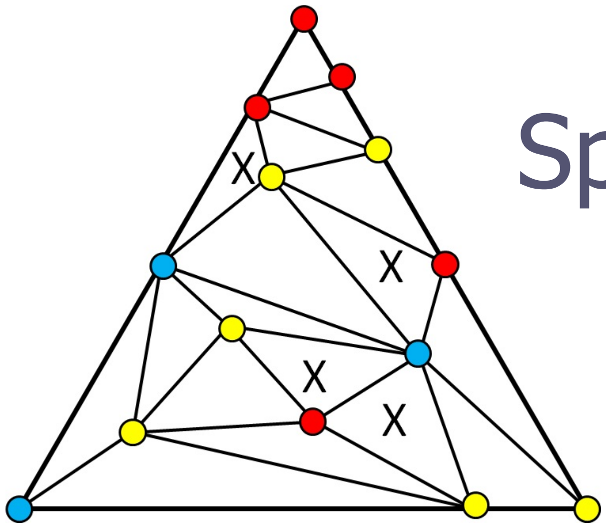
- Registers R0 and R1 and Counter* C - initialized to 1

- 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*



Sperner's Lemma

Roadmap

- The lost universality
  - Consensus is necessary but impossible
- The quest for universality
  - Consensus is sufficient
- Circumventing universality

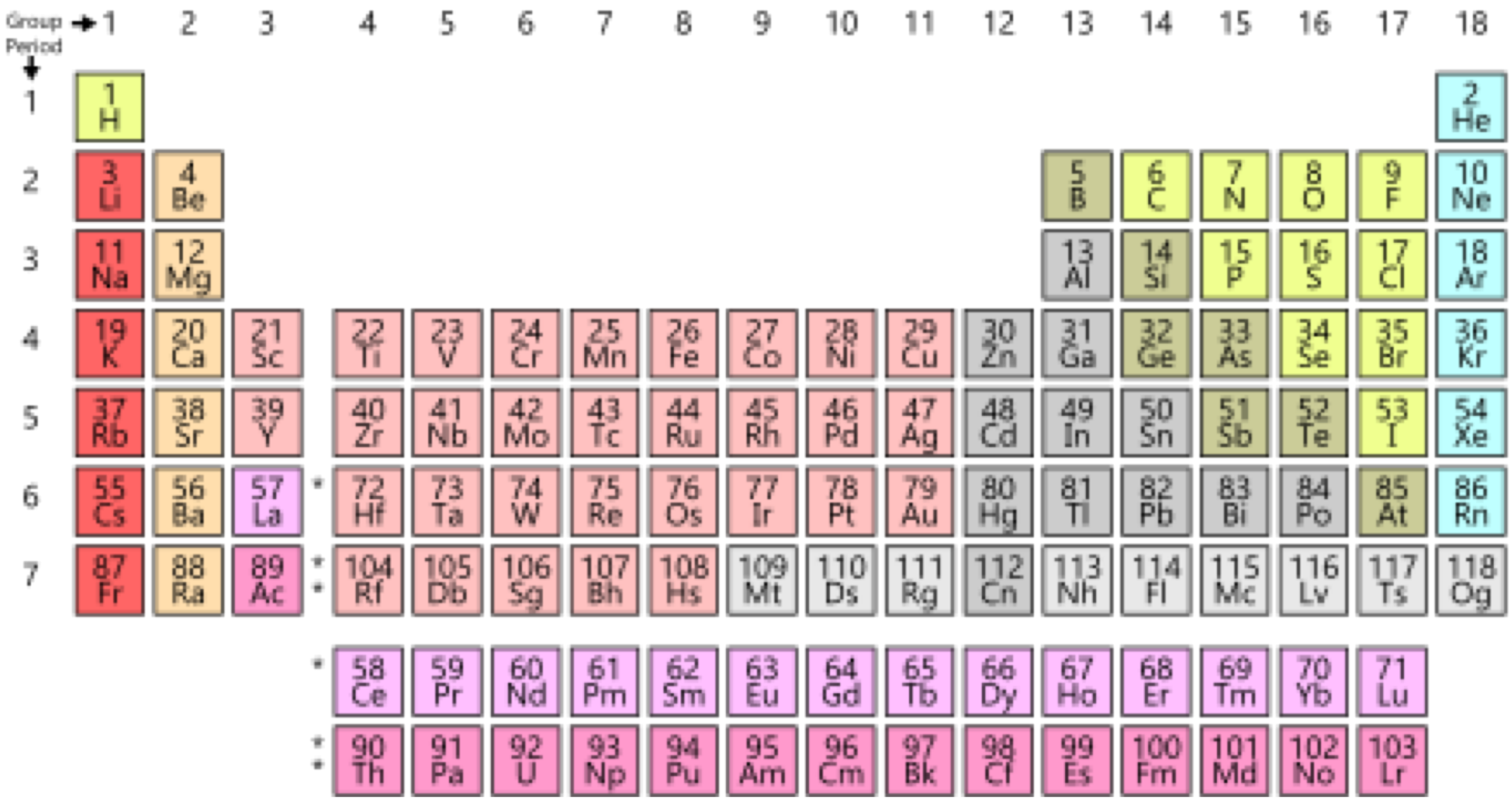
Roadmap

- The lost universality
  - Consensus is necessary but impossible

- The quest for universality
  - Consensus is sufficient

- Circumventing universality

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



Roadmap

- The lost universality
  - Consensus is necessary but impossible

- The quest for universality
  - Consensus is sufficient

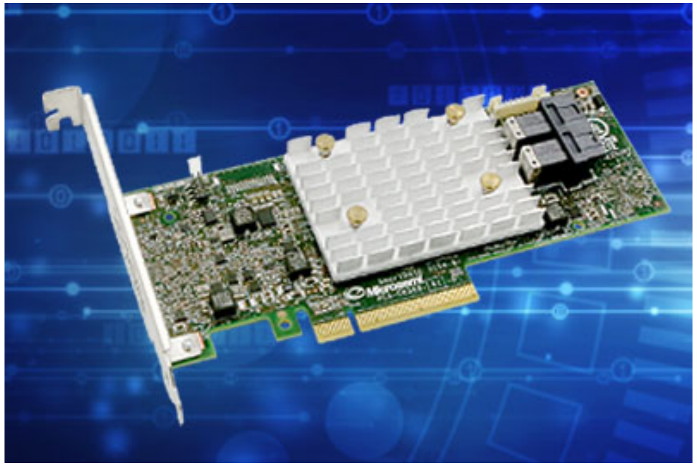
- Circumventing universality

Consensus Universality [L78]

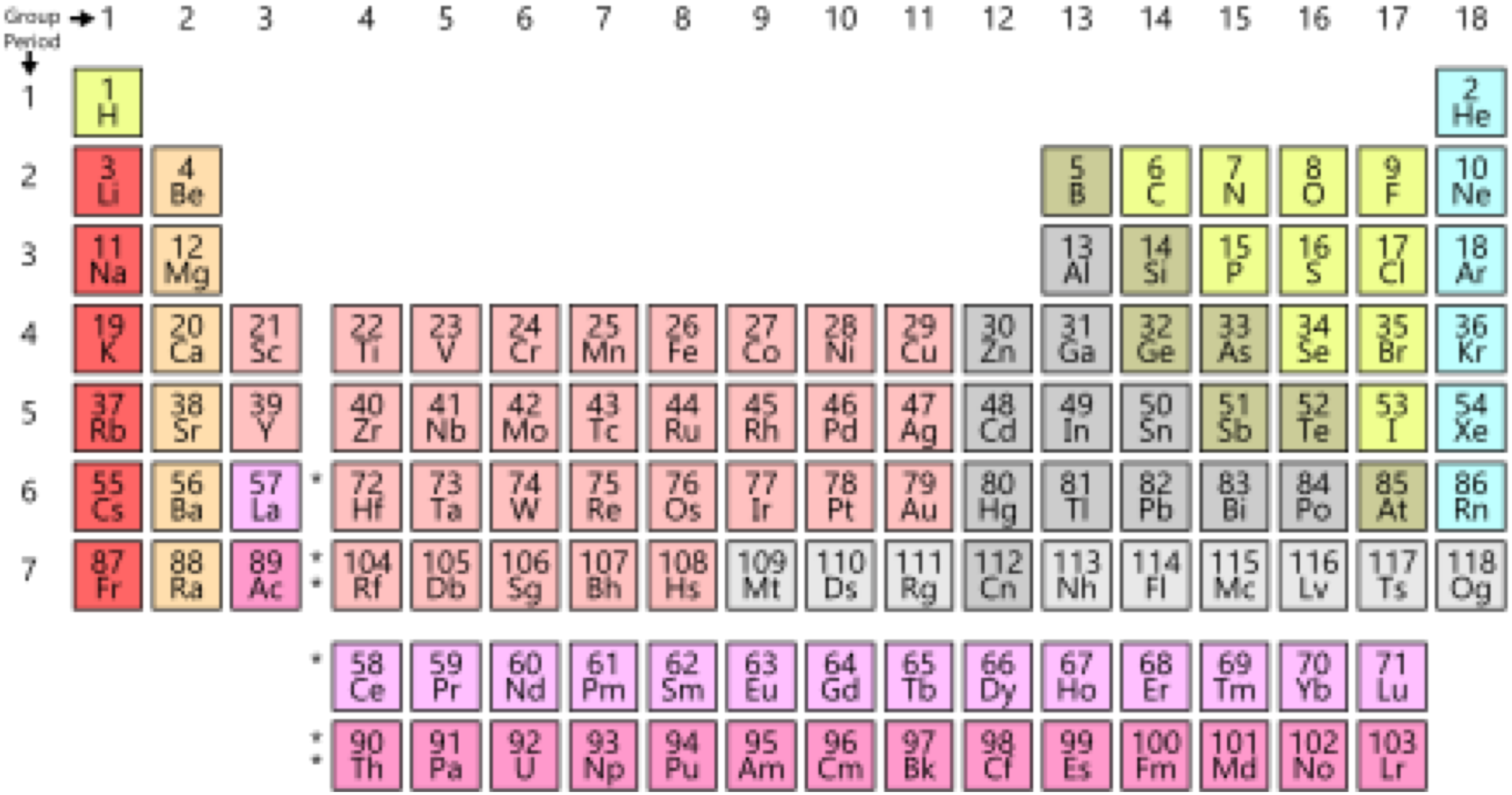
- ***Theorem:*** every object can be implemented with consensus

Eventual Synchrony

- The weakest failure detector question
- Indulgent algorithms: Paxos, PBFT
- The next 700 BFT protocols



Hardware



Group
Period
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
1
1 H
2 He
2
3 Li
4 Be
5 B
6 C
7 N
8 O
9 F
10 Ne
3
11 Na
12 Mg
13 Al
14 Si
15 P
16 S
17 Cl
18 Ar
4
19 K
20 Ca
21 Sc
22 Ti
23 V
24 Cr
25 Mn
26 Fe
27 Co
28 Ni
29 Cu
30 Zn
31 Ga
32 Ge
33 As
34 Se
35 Br
36 Kr
5
37 Rb
38 Sr
39 Y
40 Zr
41 Nb
42 Mo
43 Tc
44 Ru
45 Rh
46 Pd
47 Ag
48 Cd
49 In
50 Sn
51 Sb
52 Te
53 I
54 Xe
6
55 Cs
56 Ba
57 La
*
72 Hf
73 Ta
74 W
75 Re
76 Os
77 Ir
78 Pt
79 Au
80 Hg
81 Tl
82 Pb
83 Bi
84 Po
85 At
86 Rn
7
87 Fr
88 Ra
89 Ac
**
104 Rf
105 Db
106 Sg
107 Bh
108 Hs
109 Mt
110 Ds
111 Rg
112 Cn
113 Nh
114 Fl
115 Mc
116 Lv
117 Ts
118 Og
*
58 Ce
59 Pr
60 Nd
61 Pm
62 Sm
63 Eu
64 Gd
65 Tb
66 Dy
67 Ho
68 Er
69 Tm
70 Yb
71 Lu
**
90 Th
91 Pa
92 U
93 Np
94 Pu
95 Am
96 Cm
97 Bk
98 Cf
99 Es
100 Fm
101 Md
102 No
103 Lr

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 – OSDI 2020

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)

Distributed Payment

X000 implementations

Bitcoin: A Peer-to-Peer Electronic Cash System

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

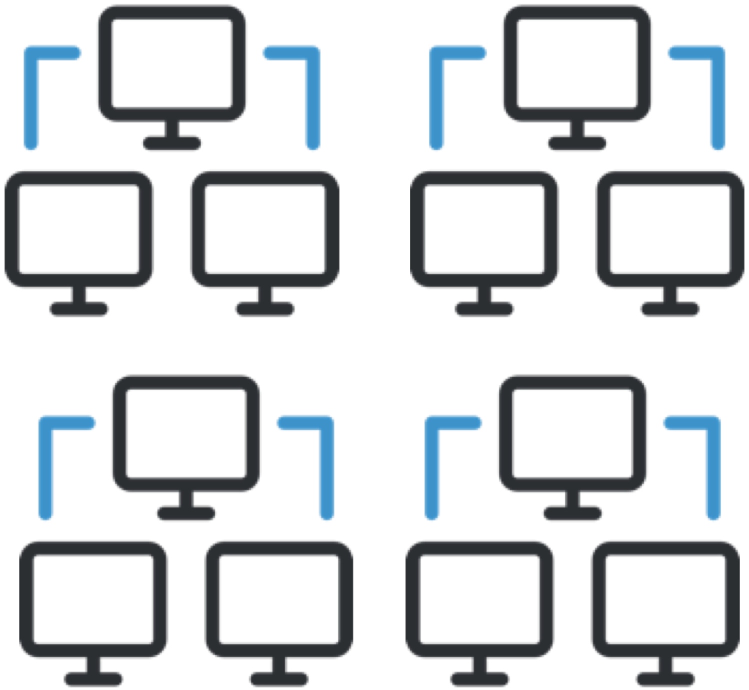
Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required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

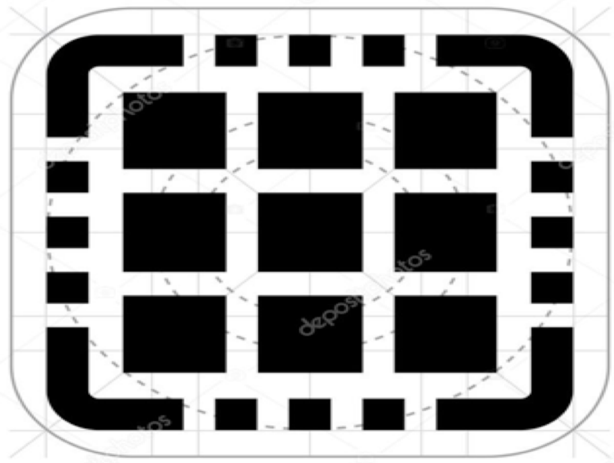
P vs NP

Asynchronous vs Synchronous

Is payment an asynchronous problem?

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

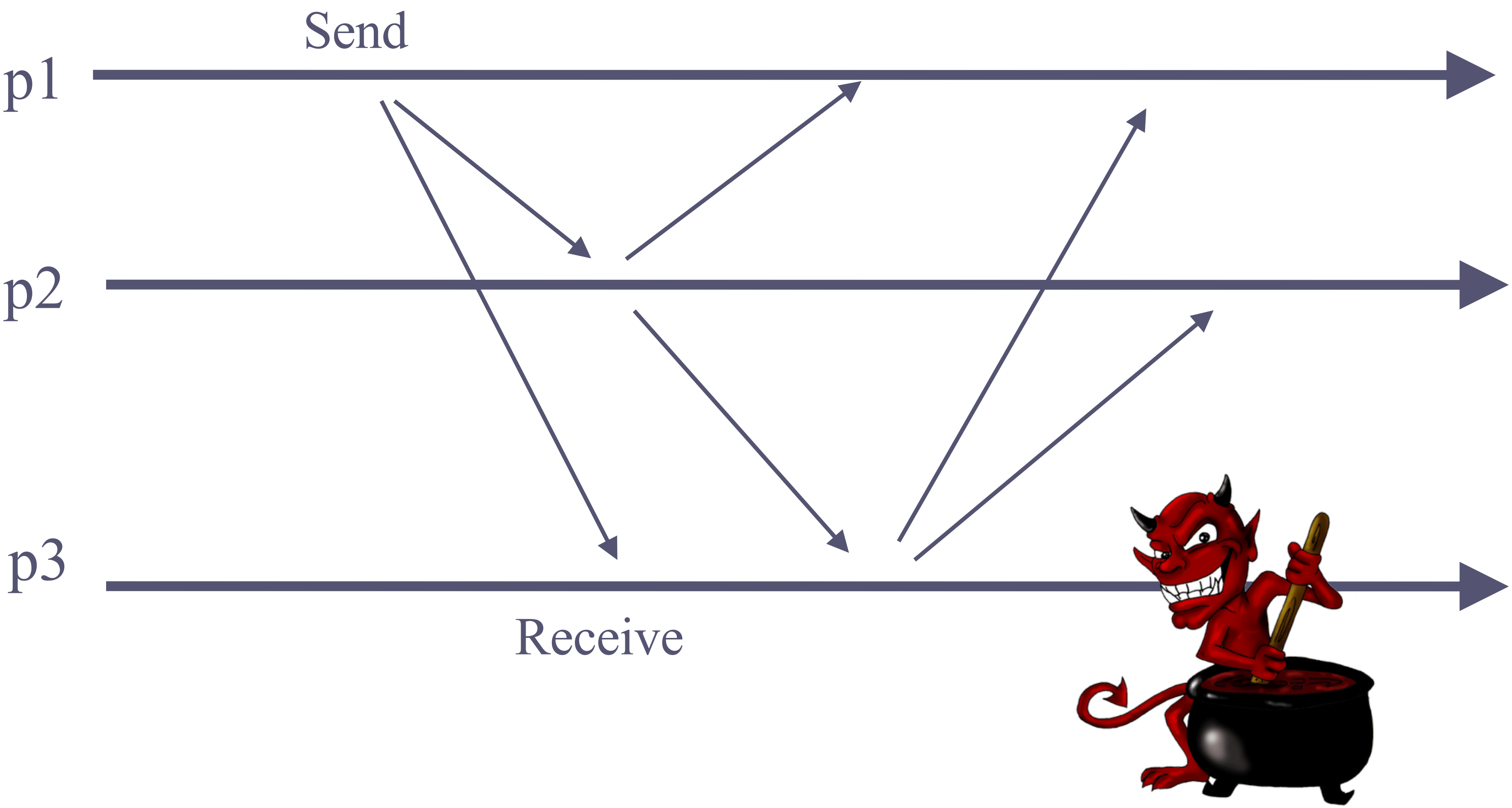




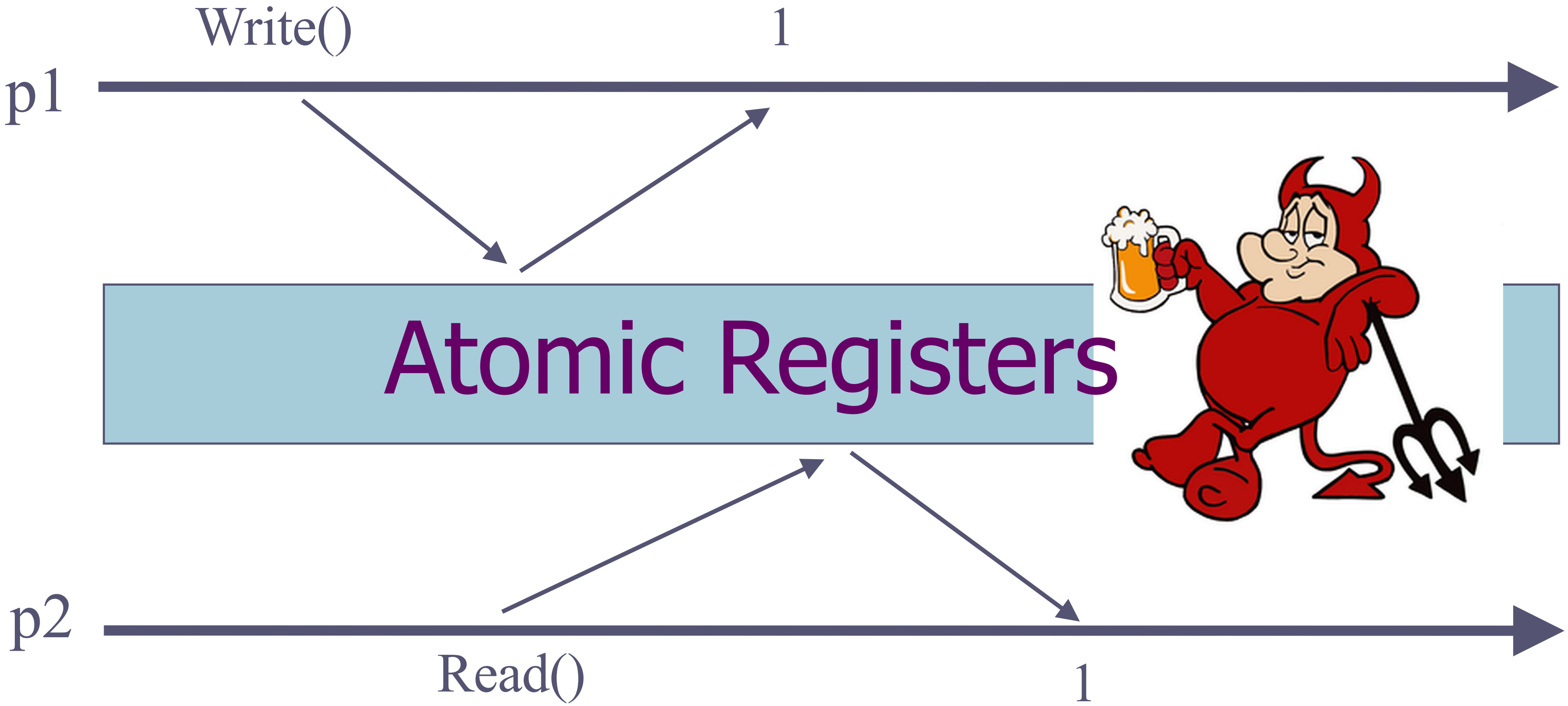
Message Passing

Shared Memory

Message Passing



Shared Memory



Message Passing ⇔ Shared Memory Modulo Quorums

⇔

Is payment an asynchronous problem?

- **Payment Object**

  - Atomicity
  - Wait-freedom

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)

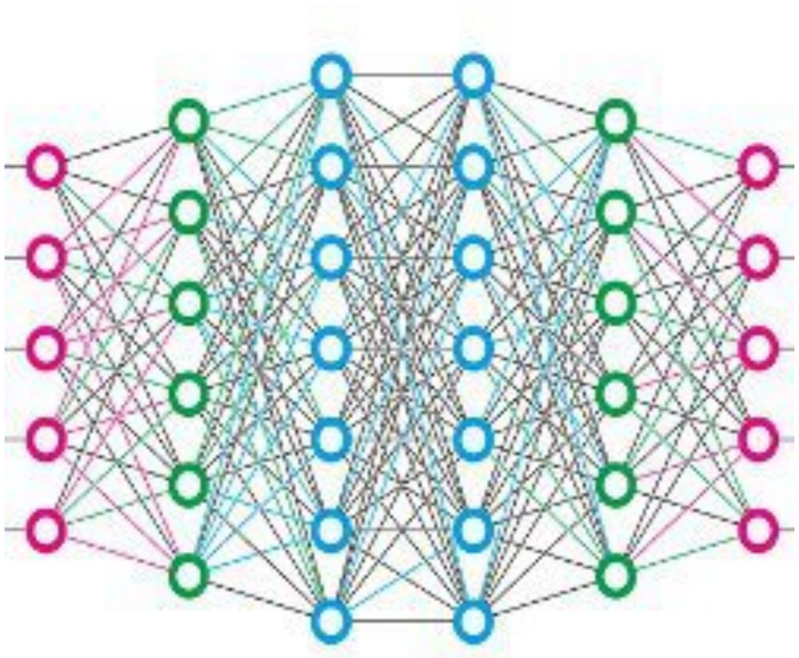
- AT2_S (PODC 2019)
- AT2_D (DNS 2020)
- AT2_R (DISC 2019)

Journey to the Center of DC

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

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

Distributed ML



PODC 2020 / ArXiv 2020 / SRDS 2020

Programming languages to the rescue

How to write a better universal Internet machine ?

How to write better universal programs?