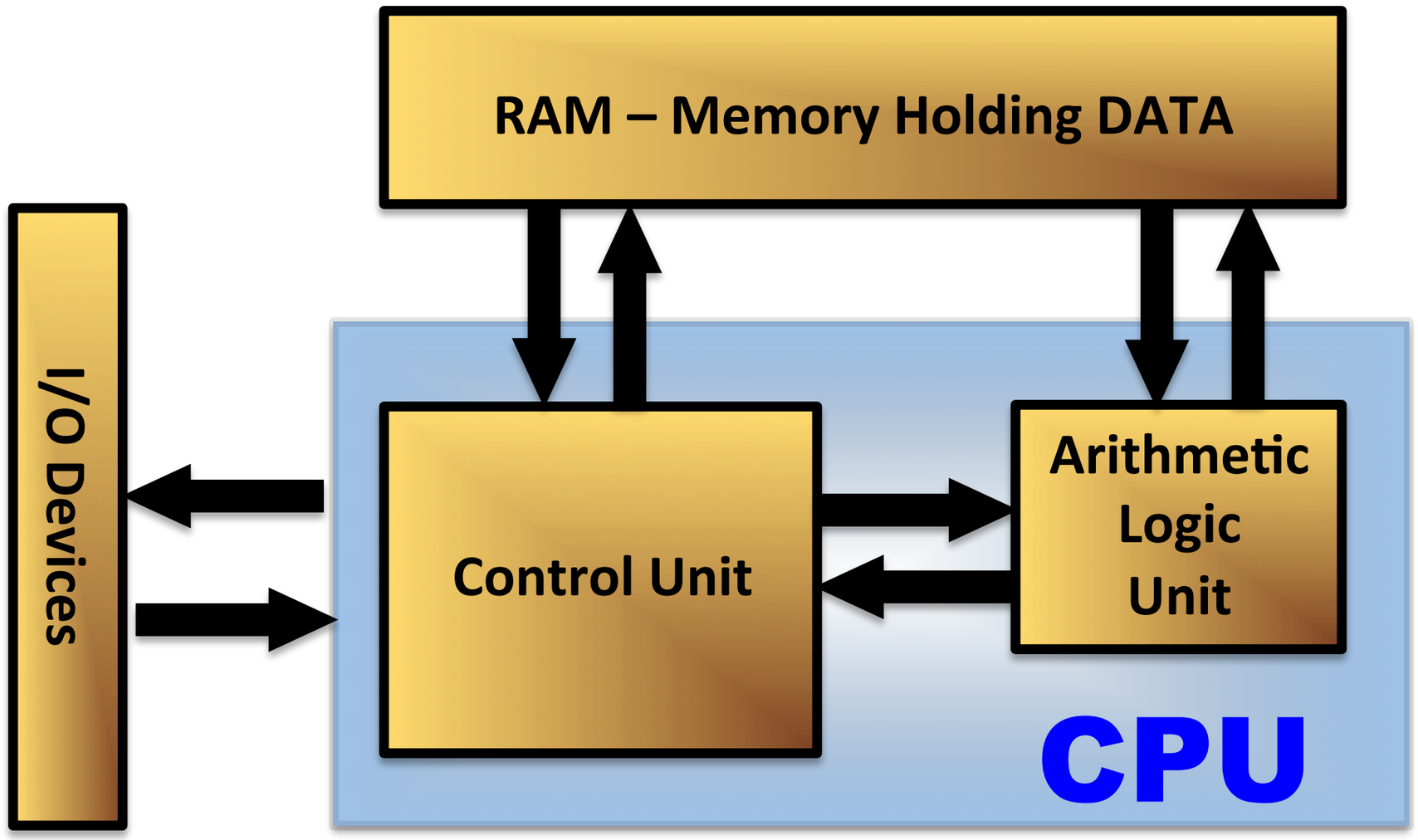
Hardware aspects of optimization

David Grellscheid
based on slides by Ivan Girotto, ICTP

MCnet

John Von Neumann

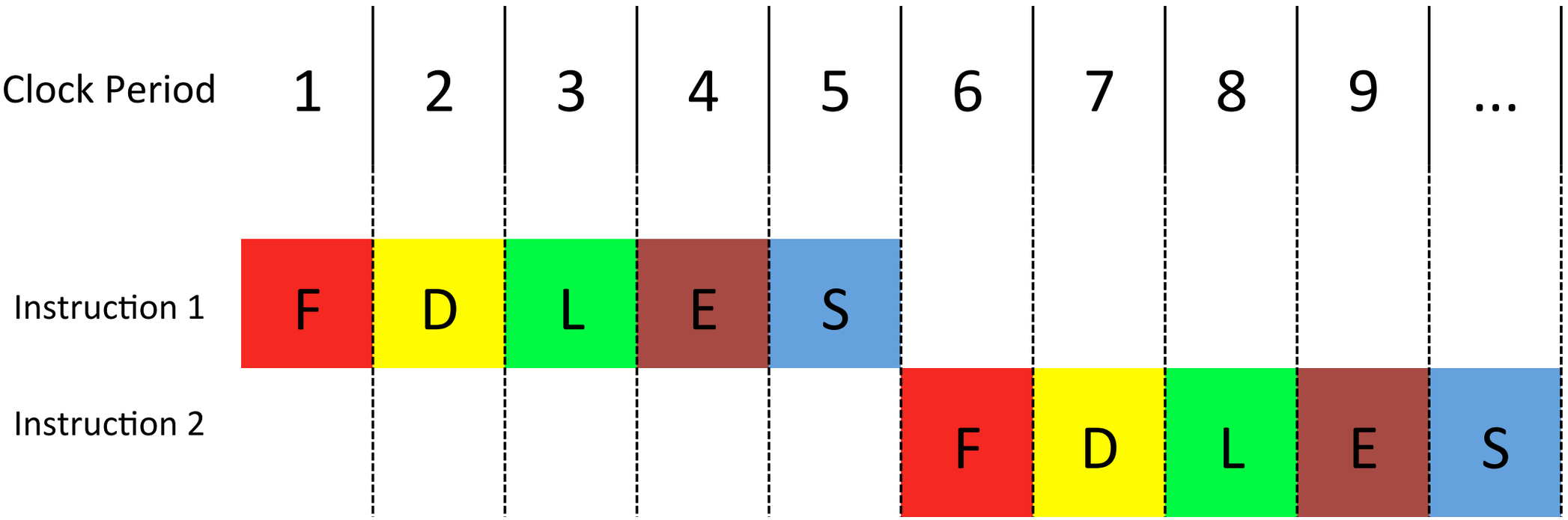
The Classical Model



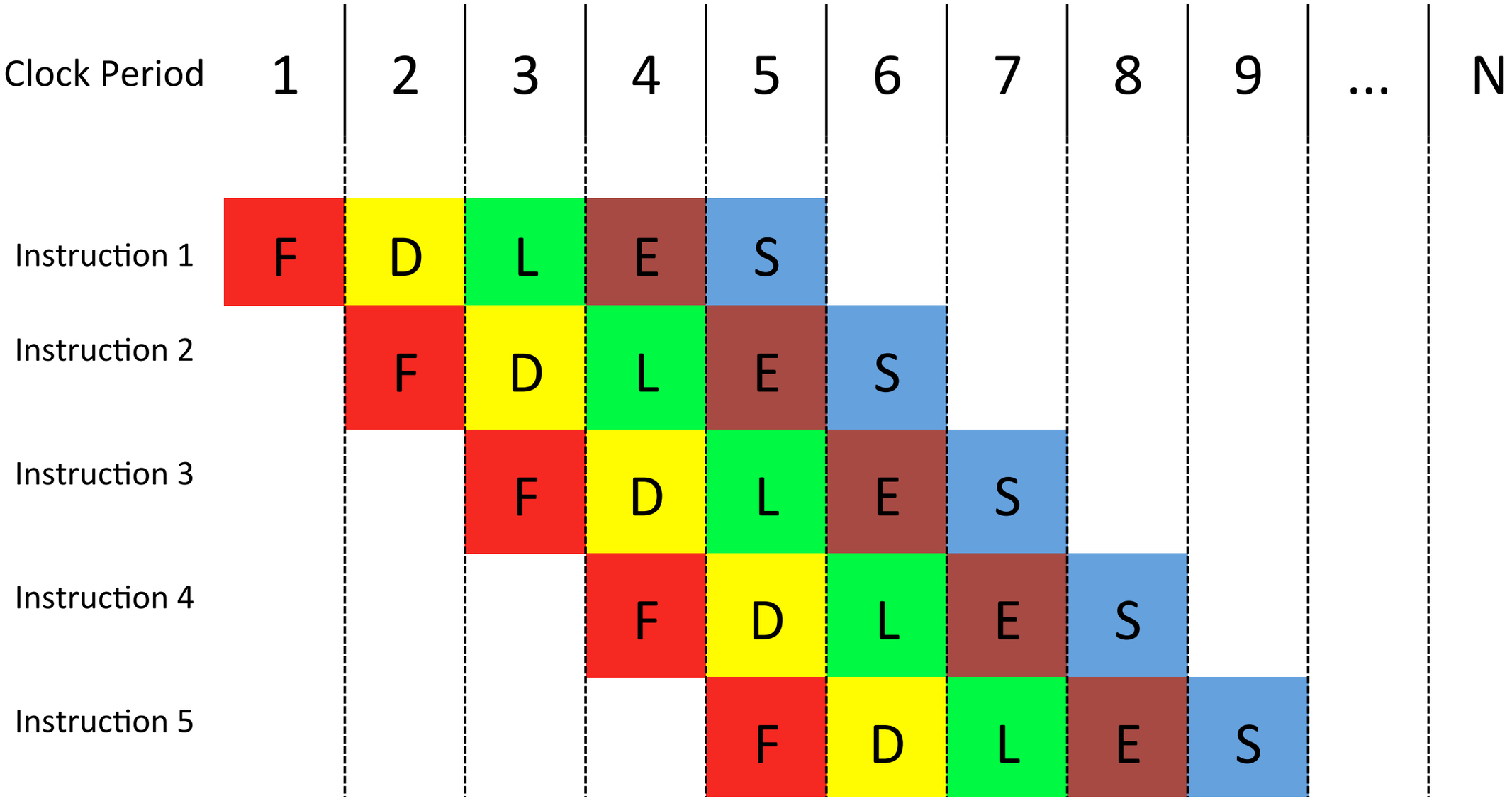
The Instruction Processing Cycle

- Fetch: read the next instruction from memory
  - 001000 00001 00010 0000000100001000
- Decode: operands and operation are decoded
  - add, $r1, $r2, 10
- Load: retrieve the data from memory to registers
- Execute: execute the instruction
  - $r1 = 4500 + 10
- Store: store the results

Sequential Processing

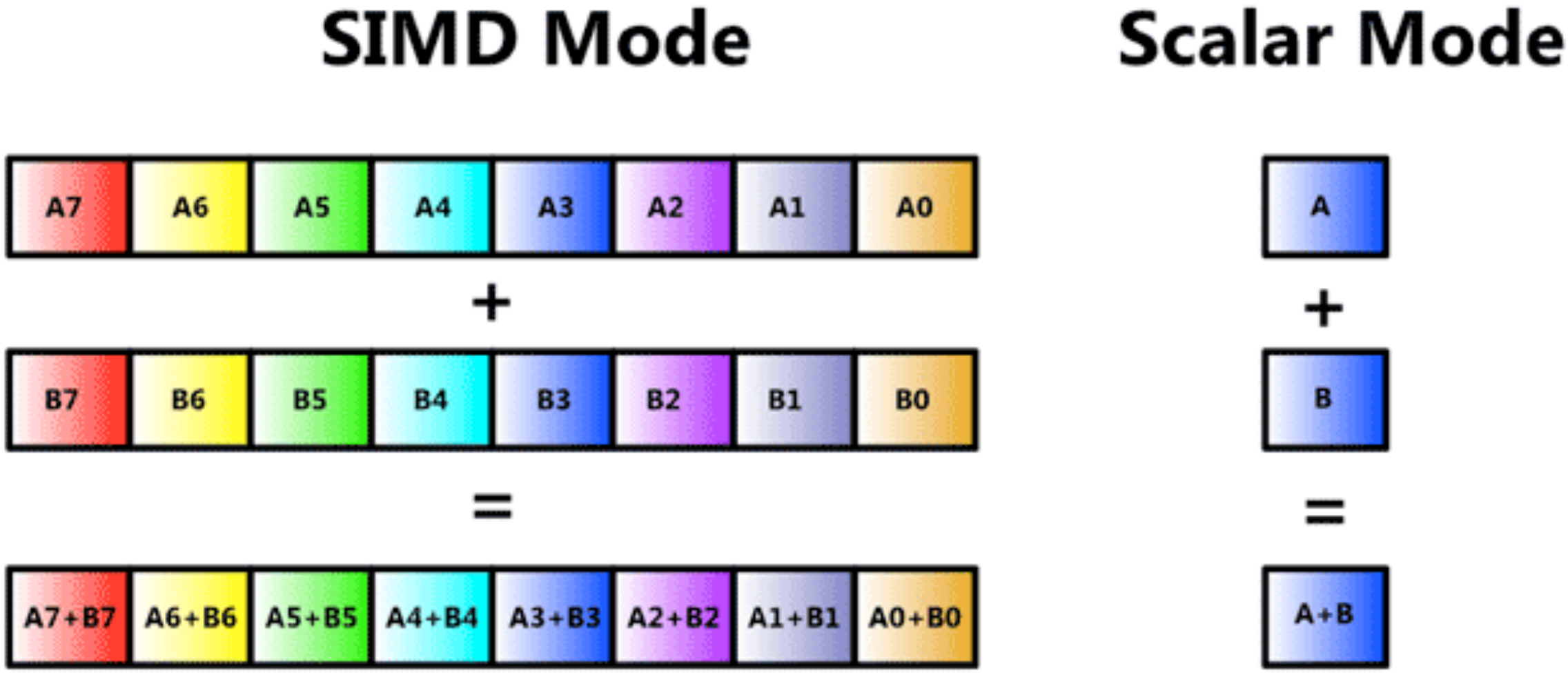


Pipelining


Clock Period
1
2
3
4
5
6
7
8
9
...
N
Instruction 1
F D L E S
Instruction 2
F D L E S
Instruction 3
F D L E S
Instruction 4
F D L E S
Instruction 5
F D L E S

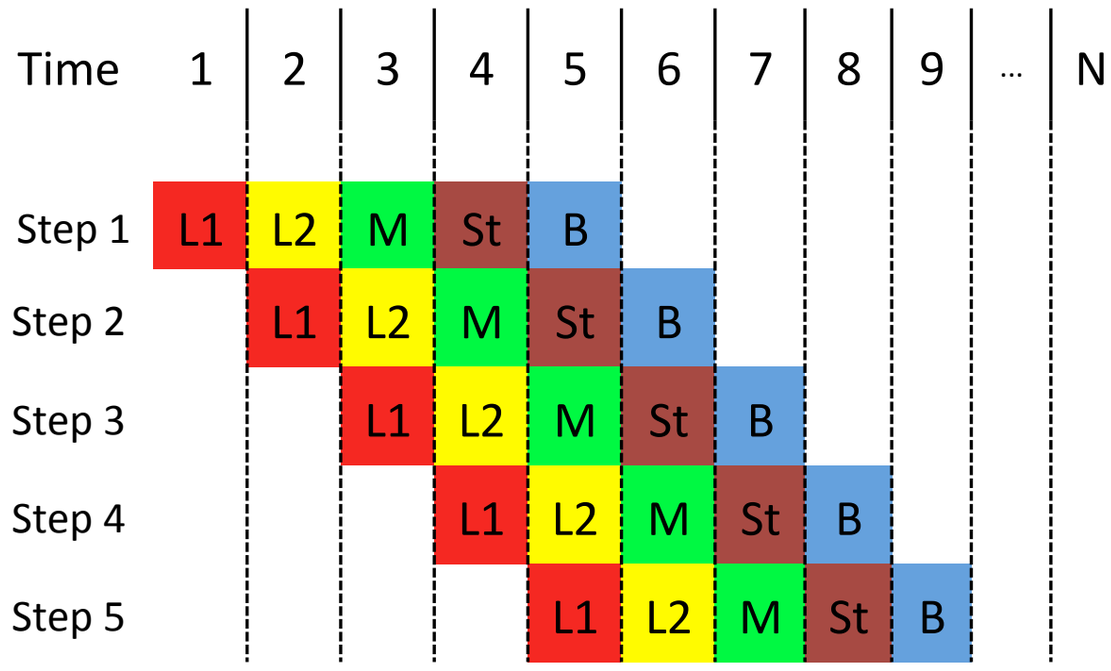
Superscalaring

Vectorization



demo

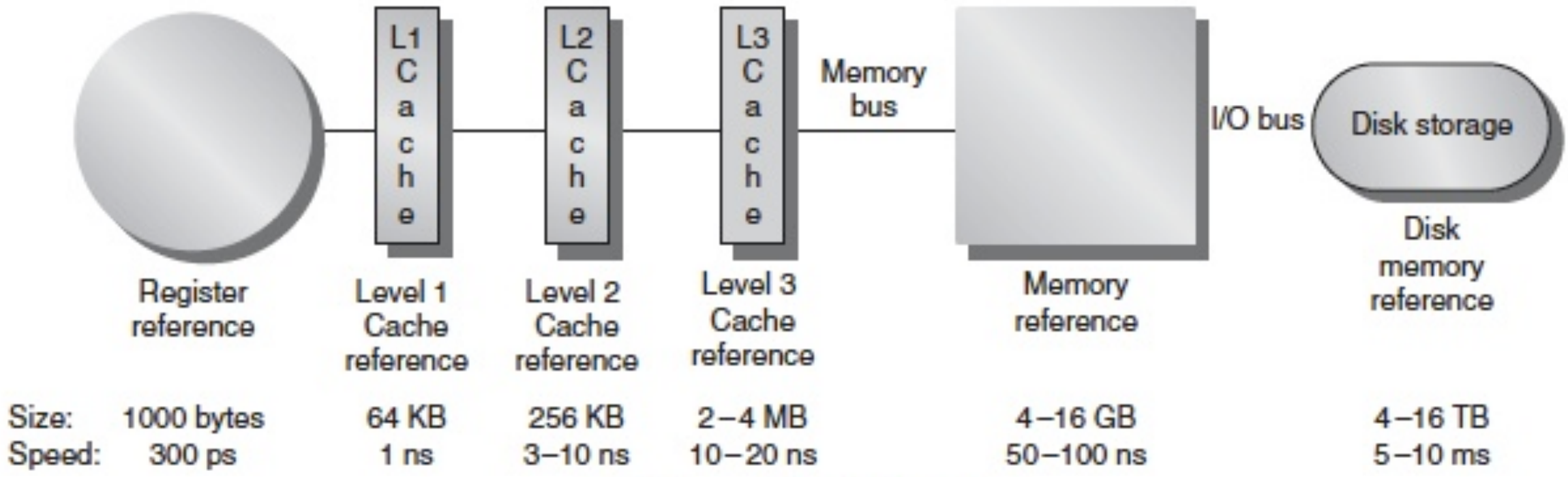
Loops and Pipeline




```
for( i = 0; i < N; i += 1 )
{
    A[i] = s * A[i]
}
```

```
Loop:   load r1, A(i)
        load r2, s
        mult r3, r2, r1
        store A(i), r3
        branch => loop
```

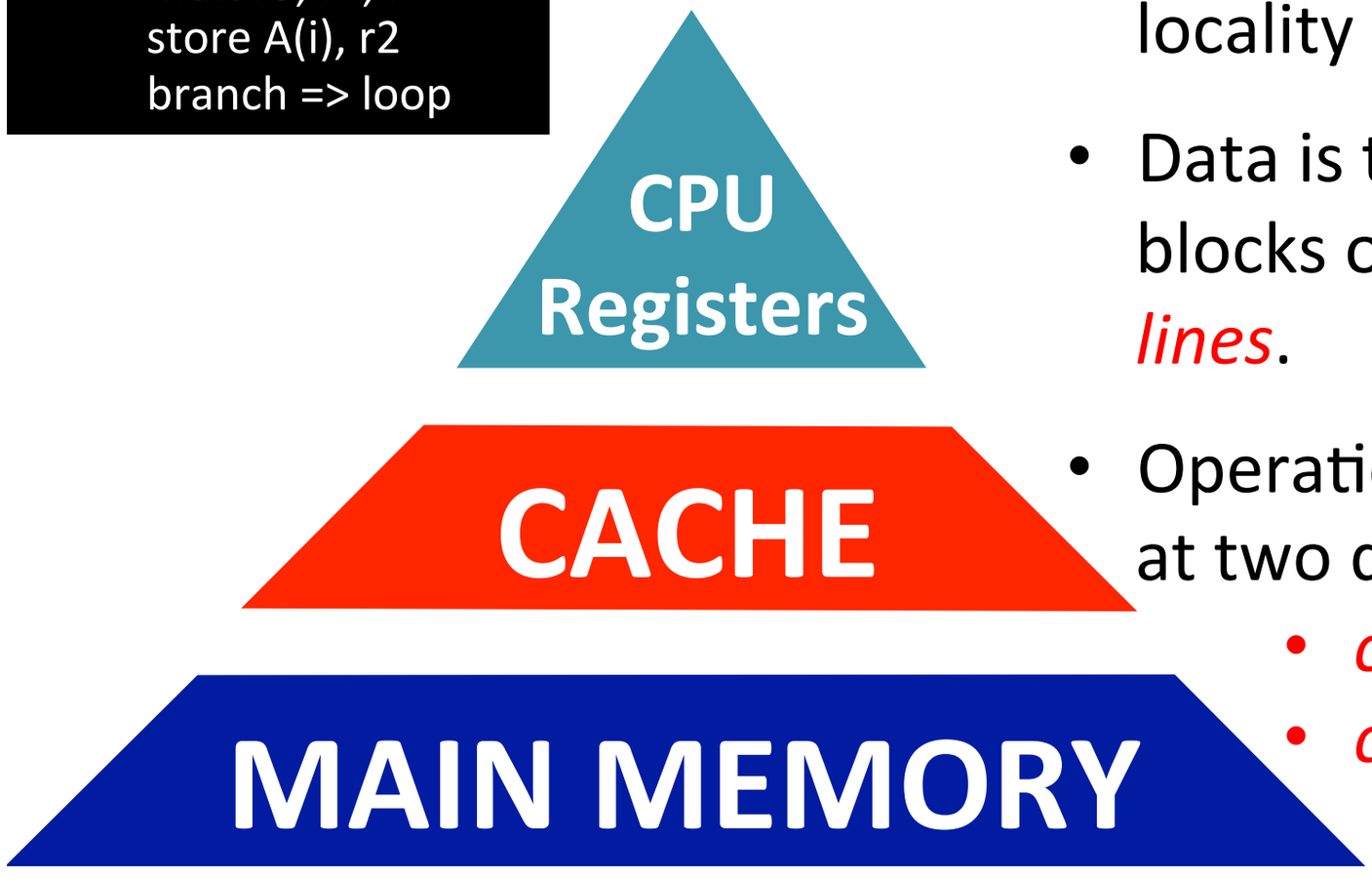
The CPU Memory Hierarchy



```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 60
Model name:            Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
Stepping:              3
CPU MHz:               1074.492
BogoMIPS:              5986.16
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              8192K
NUMA node0 CPU(s):     0-7
```

Cache Memory

```
Loop: load r1, A(i)
      load r2, s
      mult r3, r2, r1
      store A(i), r2
      branch => loop
```



- Designed for temporal/spatial locality
- Data is transferred to cache in blocks of fixed size, called *cache lines*.
- Operation of LOAD/STORE can lead at two different scenario:
  - *cache hit*
  - *cache miss*

L1 cache reference ..... 0.5 ns

Branch mispredict ..... 5 ns

L2 cache reference ..... 7 ns

Mutex lock/unlock ..... 25 ns

Main memory reference ..... 100 ns

SSD random read ..... 150,000 ns = 150 µs

Read 1 MB sequentially from memory ..... 250,000 ns = 250 µs

Read 1 MB sequentially from SSD ..... 1,000,000 ns = 1 ms

Disk seek ..... 10,000,000 ns = 10 ms

Read 1 MB sequentially from disk .... 20,000,000 ns = 20 ms

Send packet CA->Netherlands->CA .... 150,000,000 ns = 150 ms

L1 cache reference	0.5 s
Branch mispredict	5 s
L2 cache reference	7 s
Mutex lock/unlock	25 s
Main memory reference	100 s

L1 cache reference	0.5 s
Branch mispredict	5 s
L2 cache reference	7 s
Mutex lock/unlock	25 s
Main memory reference	100 s
SSD random read	1.7 days
Read 1 MB sequentially from memory	2.9 days
Read 1 MB sequentially from SSD	11.6 days

L1 cache reference	0.5 s
Branch mispredict	5 s
L2 cache reference	7 s
Mutex lock/unlock	25 s
Main memory reference	100 s
SSD random read	1.7 days
Read 1 MB sequentially from memory	2.9 days
Read 1 MB sequentially from SSD	11.6 days
Disk seek	16.5 weeks
Read 1 MB sequentially from disk	7.8 months

L1 cache reference	0.5 s
Branch mispredict	5 s
L2 cache reference	7 s
Mutex lock/unlock	25 s
Main memory reference	100 s
SSD random read	1.7 days
Read 1 MB sequentially from memory	2.9 days
Read 1 MB sequentially from SSD	11.6 days
Disk seek	16.5 weeks
Read 1 MB sequentially from disk	7.8 months
Send packet CA->Netherlands->CA	4.8 years