Distributed Data Processing Environments

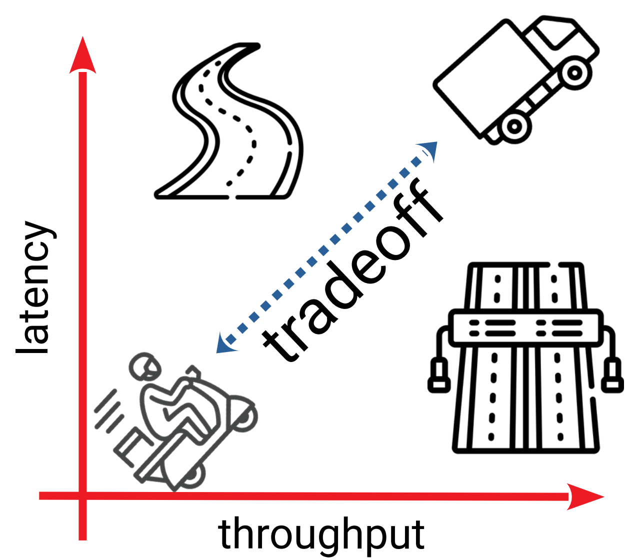
José Orlando Pereira

Departamento de Informática
Universidade do Minho

“Fast”

- Latency: time to complete a task
- Throughput: tasks completed in a unit of time

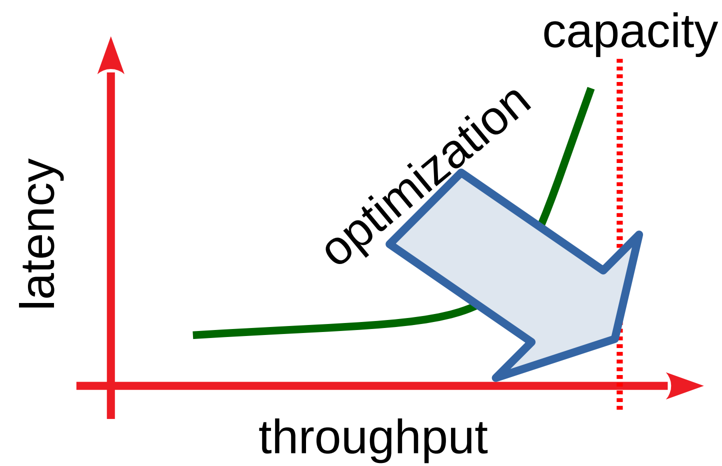
- Hard / expensive to achieve both at the same time



Icons by Flaticon.com.

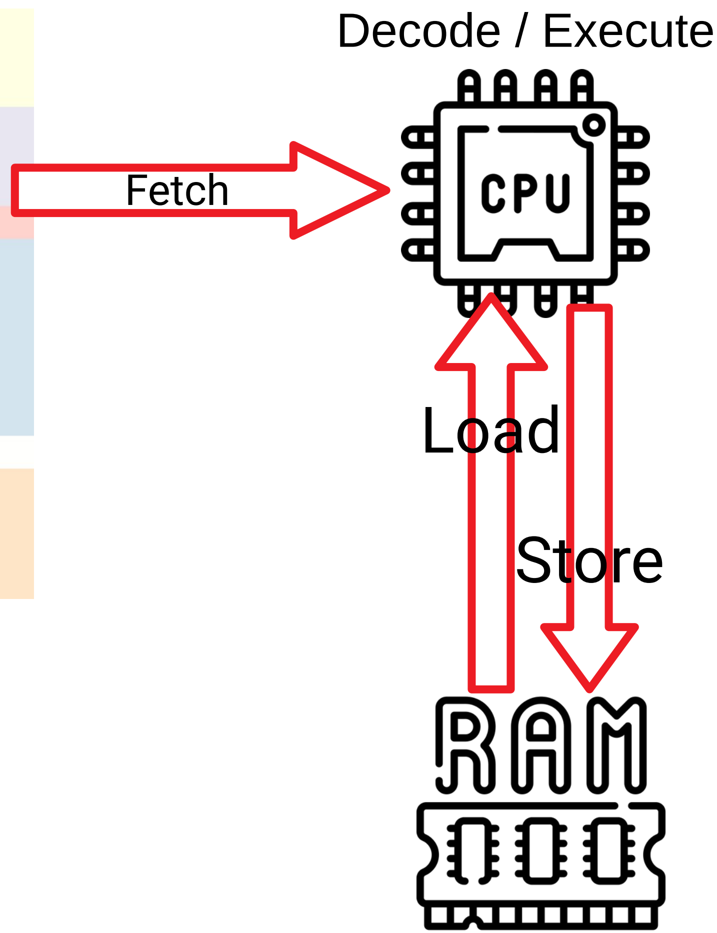
Latency vs. throughput

- Latency vs. bandwidth trade-off changes with load
- When approaching system capacity, latency increases with queuing
- Optimization means pushing the curve right/down



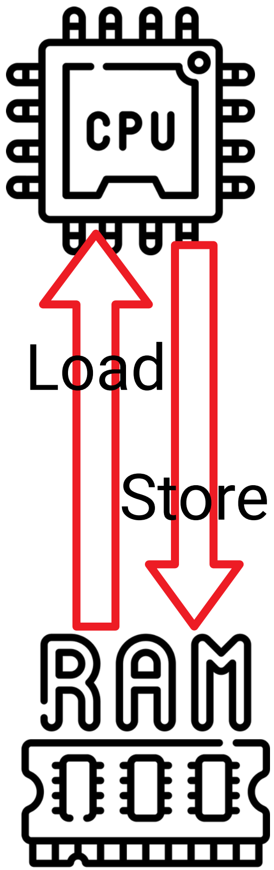
A model of computing

```
        mov     eax, DWORD PTR [rbp-20]
        imul    eax, eax
        mov     DWORD PTR [rbp-8], eax
        mov     eax, DWORD PTR [rbp-20]
        add     eax, eax
IP ->   mov     DWORD PTR [rbp-12], eax
        mov     DWORD PTR [rbp-4], 5
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, DWORD PTR [rbp-12]
        test    eax, eax
        jns     .L2
        mov     eax, DWORD PTR [rbp-12]
        add     DWORD PTR [rbp-4], eax
.L2:
        mov     edx, DWORD PTR [rbp-4]
        mov     eax, DWORD PTR [rbp-8]
        add     edx, eax
        mov     eax, DWORD PTR [rbp-12]
```



Icons by Flaticon.com.

A model of computing

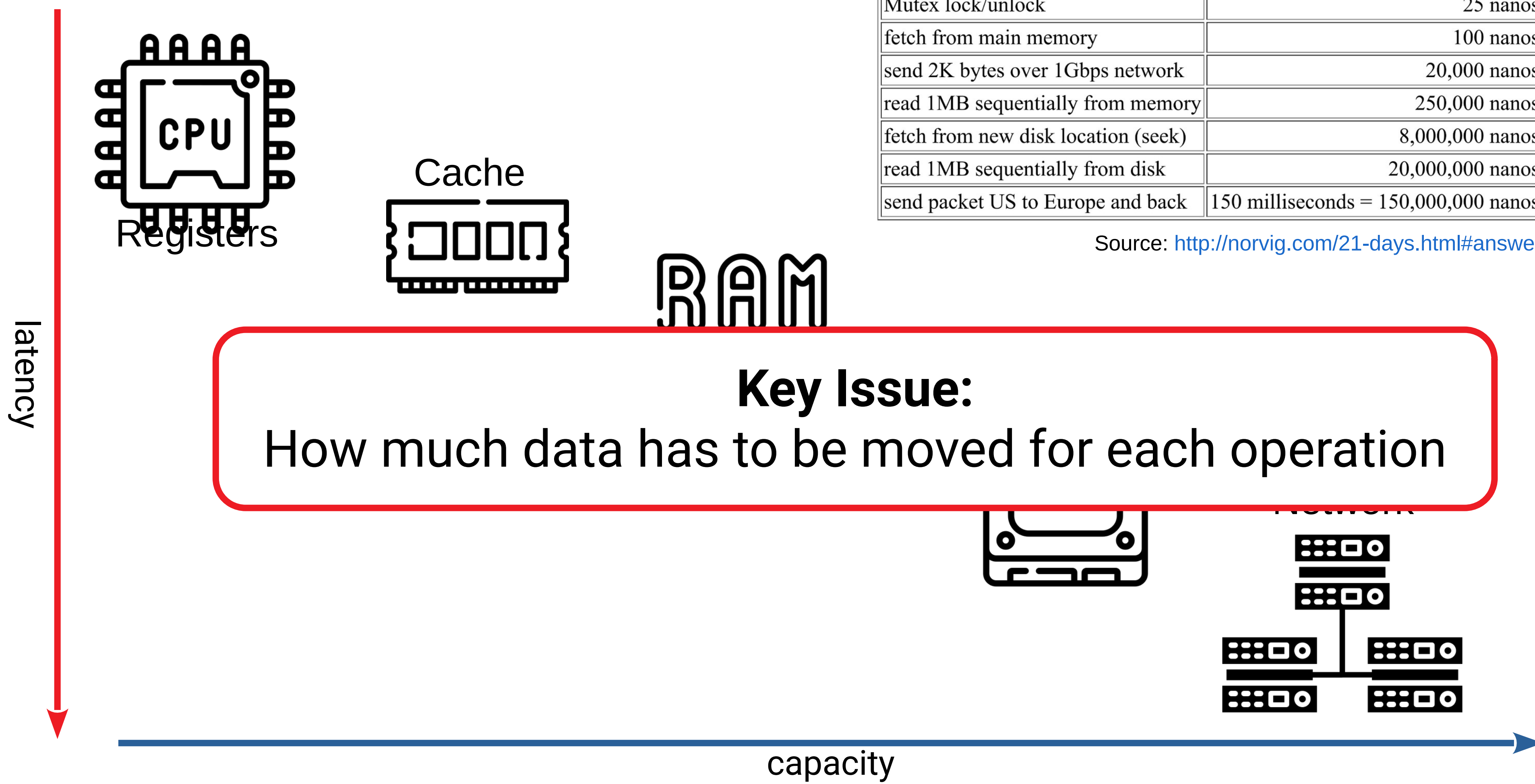


- Challenges for data-intensive programs:
  - RAM memory is not big enough
  - RAM memory is not fast enough

Memory hierarchy

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

Source: http://norvig.com/21-days.html#answers



Key Issue:
How much data has to be moved for each operation

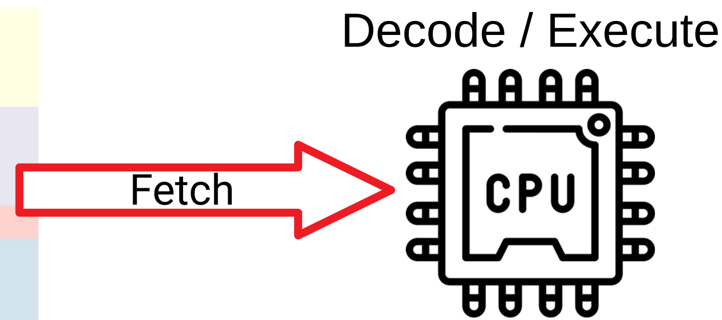
Memory hierarchy

- Minimize data movement to optimize performance
- General strategies:
  - Improve <u>locality</u> → Do more with data that is already loaded up in the memory hierarchy
  - Be <u>thrifty</u> → Avoid loading data that is not strictly necessary

A model of computing

IP ->

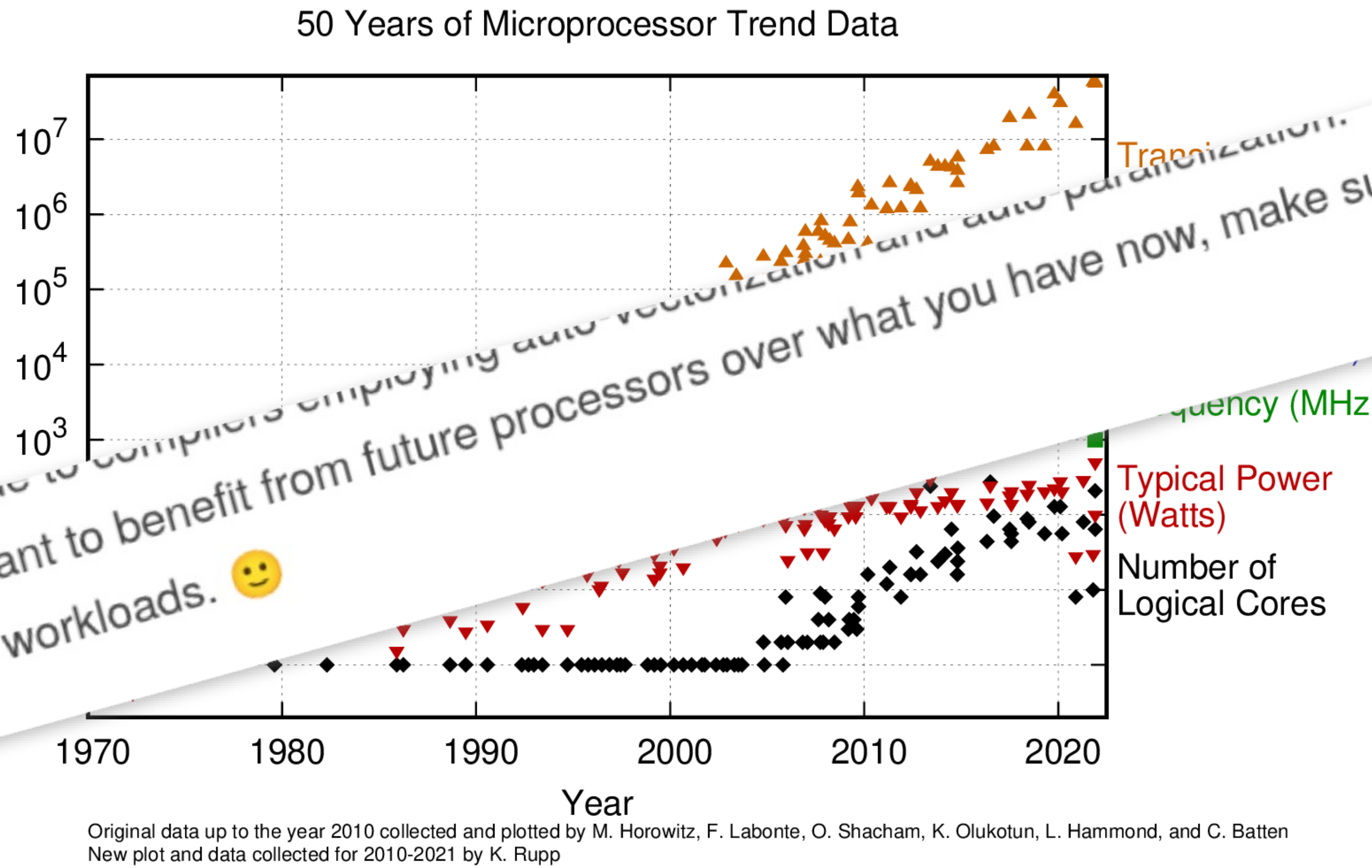
```
        mov     eax, DWORD PTR [rbp-20]
        imul    eax, eax
        mov     DWORD PTR [rbp-8], eax
        mov     eax, DWORD PTR [rbp-20]
        add     eax, eax
        mov     DWORD PTR [rbp-12], eax
        mov     DWORD PTR [rbp-4], 5
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, DWORD PTR [rbp-12]
        test    eax, eax
        jns     .L2
        mov     eax, DWORD PTR [rbp-12]
        add     DWORD PTR [rbp-4], eax
.L2:
        mov     edx, DWORD PTR [rbp-4]
        mov     eax, DWORD PTR [rbp-8]
        add     edx, eax
        mov     eax, DWORD PTR [rbp-12]
```



- Challenge for data-intensive programs:
  - Computation is not fast enough

Icons by Flaticon.com.

Moore's Law



are primarily due to compilers employing auto-vectorization and auto-parallelization.

Thus, if you want to benefit from future processors over what you have now, make sure to have parallel workloads. 🙂

Source https://github.com/karlrupp/microprocessor-trend-data

Pipelining

instruction latency = 5 cycles

Fetch	Decode	Load	Execute	Store
mov DWORD PTR [rbp-8], eax				
mov eax, DWORD PTR [rbp-20]	DWORD PTR [rbp-8], eax			
…	mov eax, DWORD PTR [rbp-20]	DWORD PTR [rbp-8], eax		
…	…	mov eax, DWORD PTR [rbp-20]	DWORD PTR [rbp-8], eax	
…	…	…	mov eax, DWORD PTR [rbp-20]	DWORD PTR [rbp-8], eax
…	…	…	…	mov eax, DWORD PTR [rbp-20]

throughput = 1 instruction / cycle

Pipelining

Fetch	Decode	Load	Execute	Store
`mov eax, DWORD PTR [rbp-20]`				
`add eax, eax`	`mov eax, DWORD PTR [rbp-20]`			
…	`add eax, eax`	`eax, DWORD PTR [rbp-20]`		
…	…	???	`mov eax, DWORD PTR [rbp-20]`	
…	…	…	stall	`mov eax, DWORD PTR [rbp-20]`
…	…	…	…	stall

- Data dependency:
  - Trying to load a value that has not yet been computed

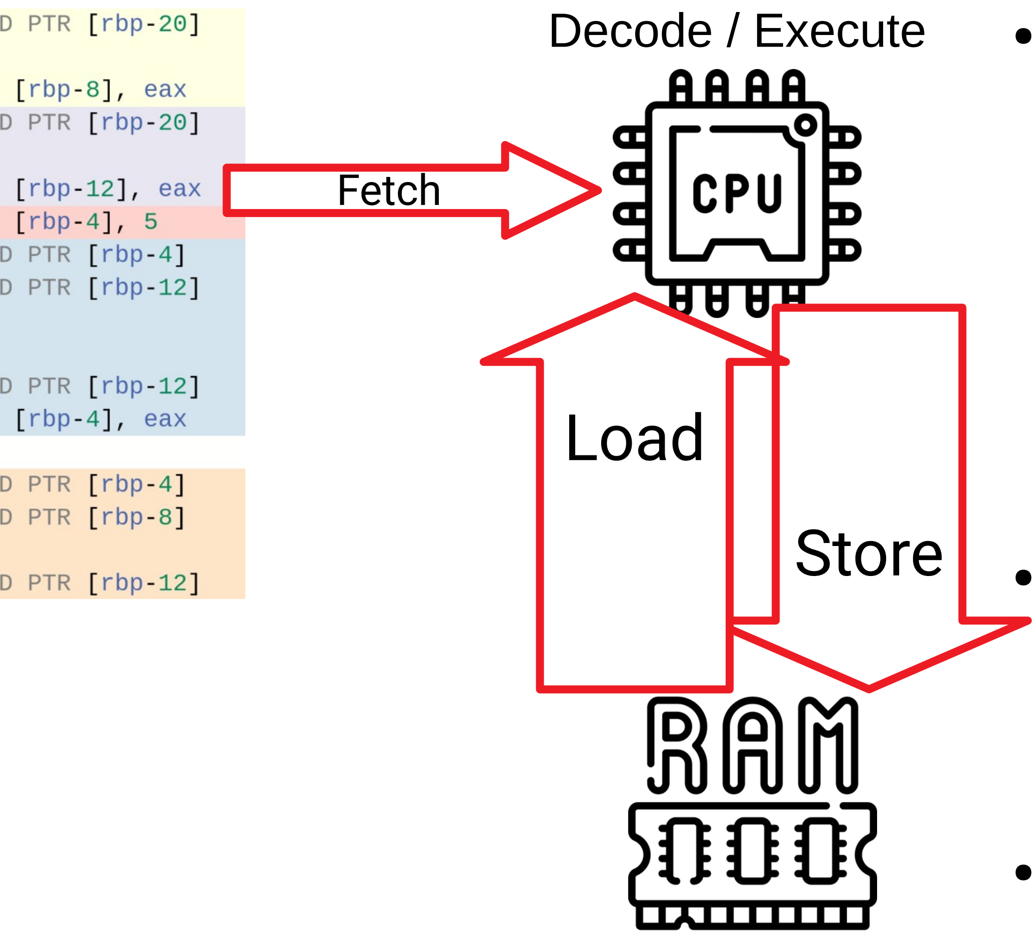
Pipelining

Fetch	Decode	Load	Execute	Store
`jns .L2`				
???	`jns .L2`			
stall	stall	`jns .L2`		
stall	stall	stall	`jns .L2`	
stall	stall	stall	stall	`jns .L2`
		stall	stall	stall

```
.L2:
        mov     edx, DWORD PTR [rbp-4]
```

- Control flow dependency:
  - Cannot predict the next instruction

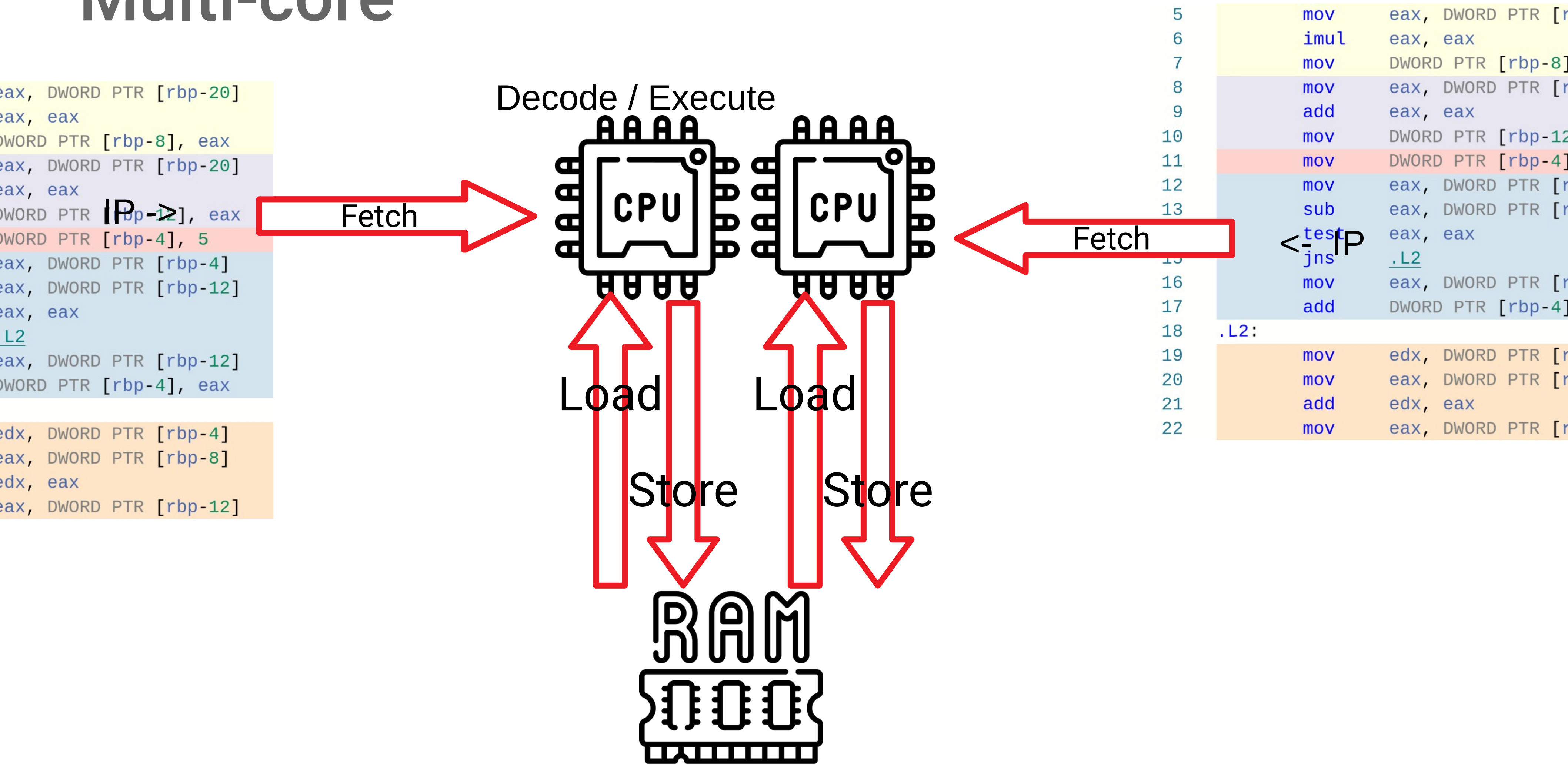
Vectorization



- Use wide registers that can fit vectors instead of scalars:
  - Example: Intel AVX512 → 512 bits
    - 64 byte vector
    - 32 shorts
    - 16 ints
    - …
- Load, execute, and store full vectors, or slices of vectors, in a single instruction
- Key technique in GPUs

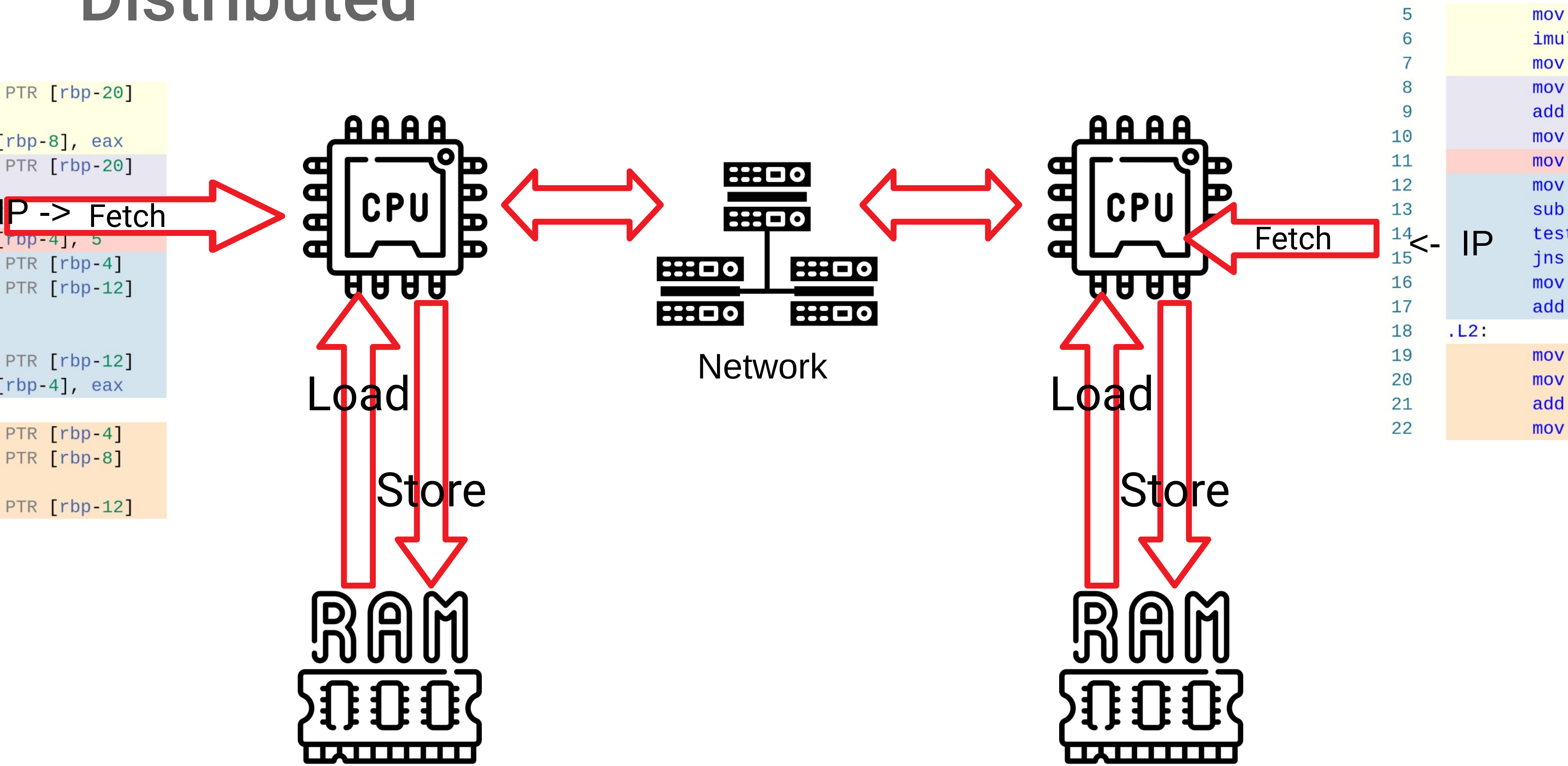
Icons by Flaticon.com.

Multi-core



Icons by Flaticon.com.

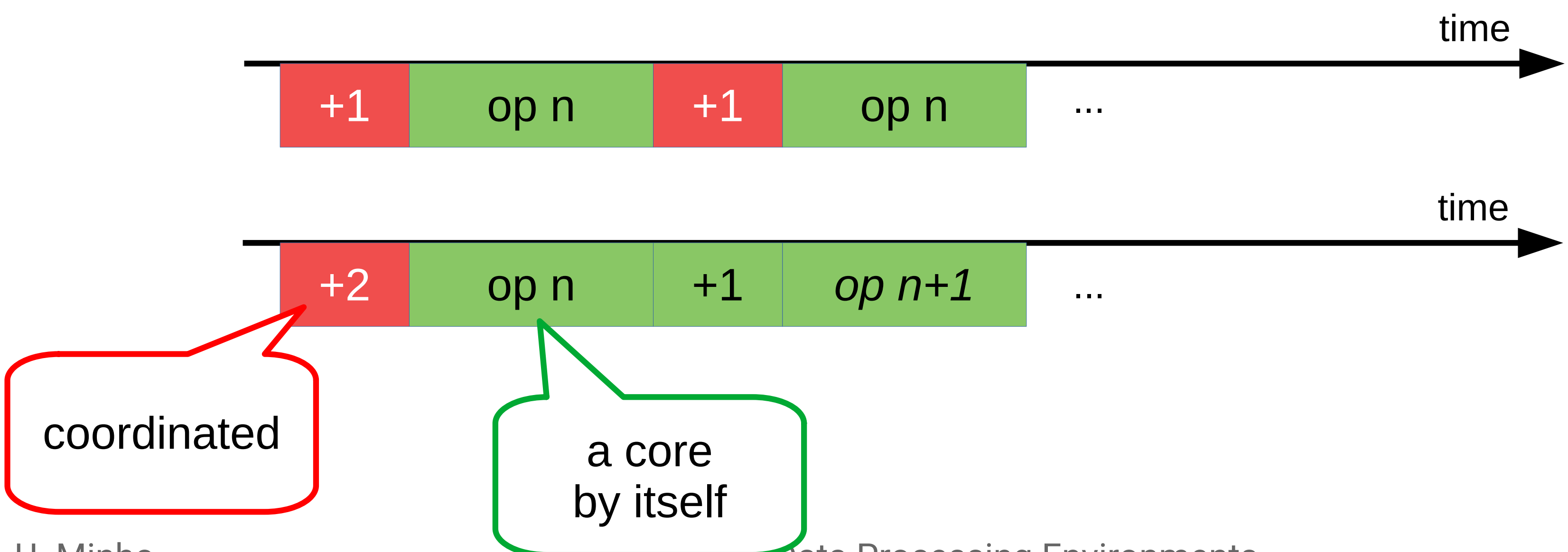
Distributed



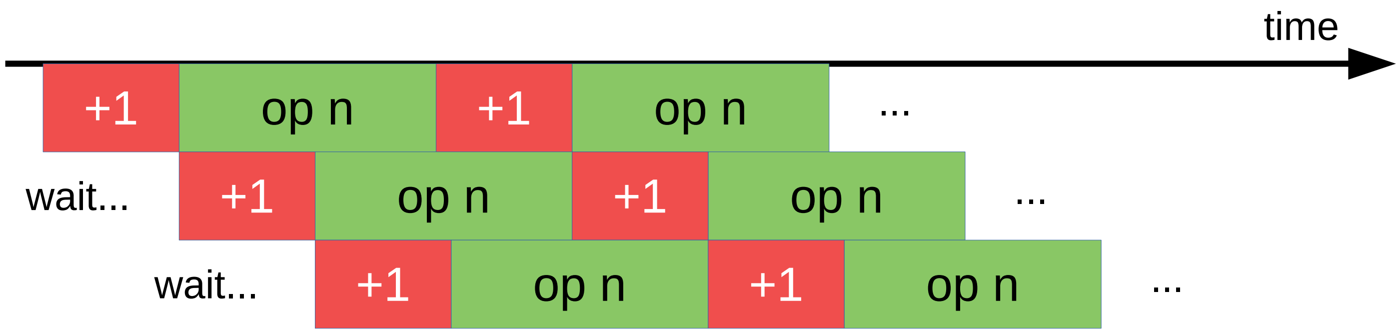
Icons by Flaticon.com.

Coordination overhead

- Splitting a task incurs in coordination overhead
- Consider two versions of a chunked vector operation:
  - Get chunk of size 1, execute
  - Get chunk of size 2, execute one and the other

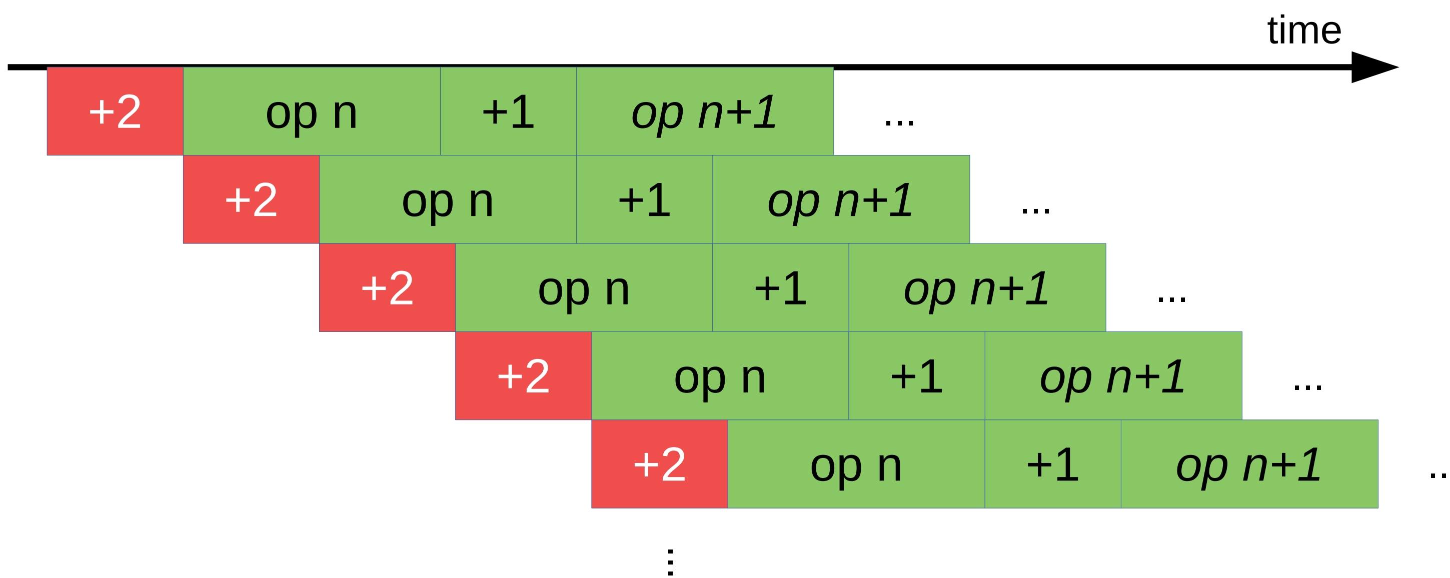


Coordination overhead



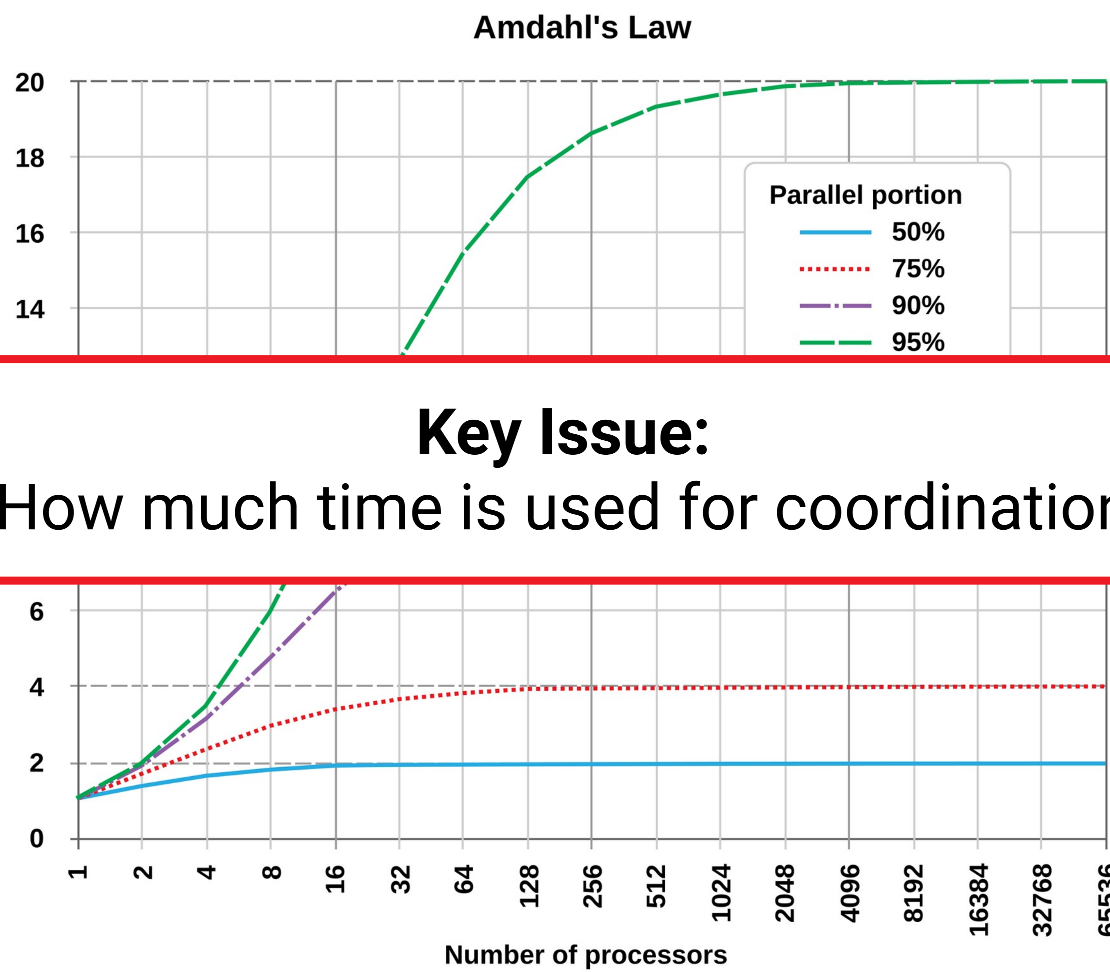
- Eventually, at least one core is blocked waiting for coordination

Coordination overhead



- Reducing the contention on coordination improves performance, even if doing the same work!

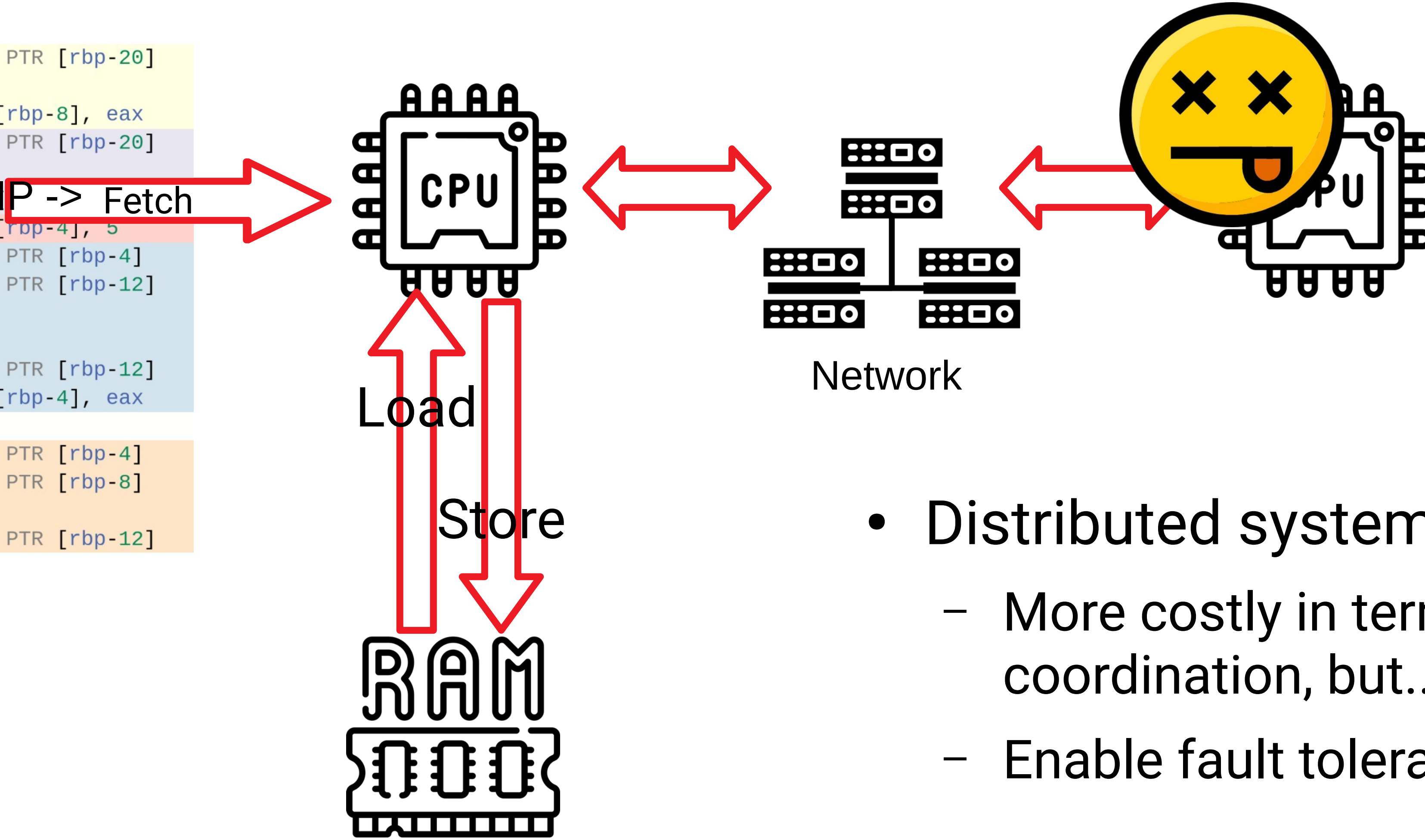
Amdahl’s Law



Key Issue:
How much time is used for coordination

Image from Wikipedia.

Fault tolerance

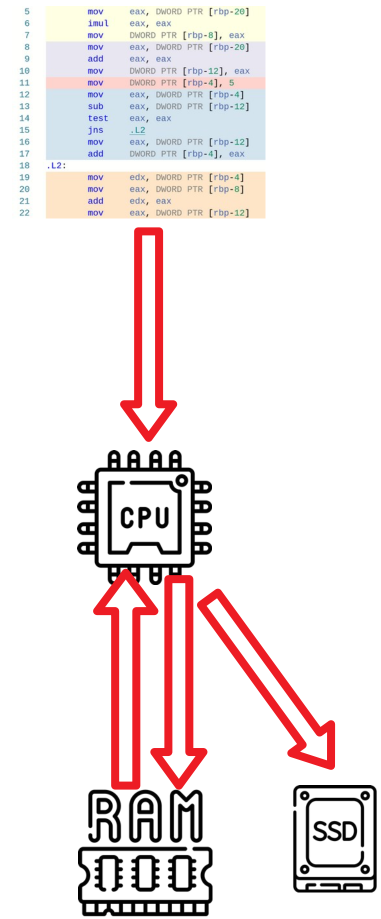


- Distributed systems:
  - More costly in terms of coordination, but...
  - Enable fault tolerance

Icons by Flaticon.com.

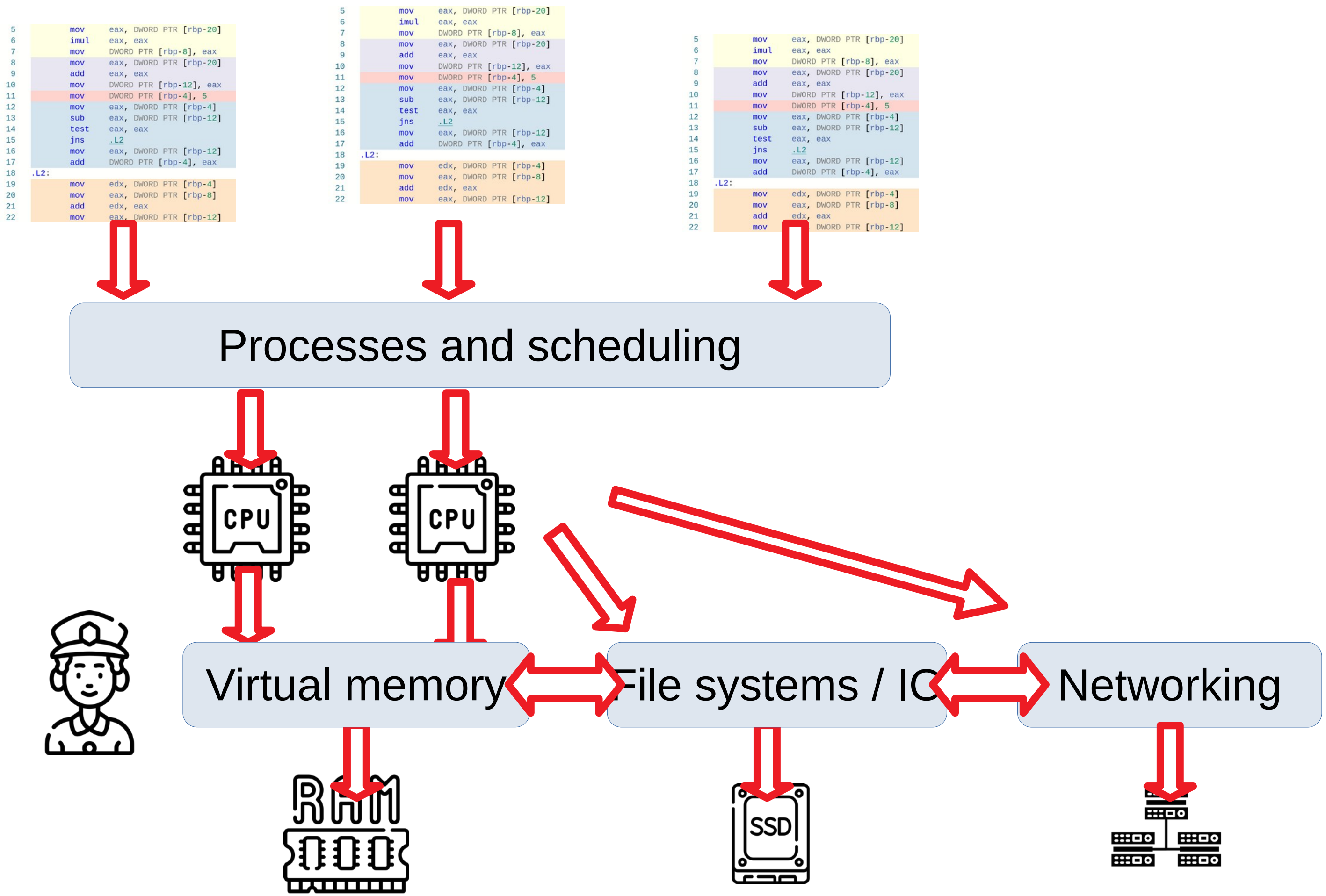
Hardware abstraction and protection?

- How to run programs on computers with different configurations?
  - Memory capacity
  - # of CPU cores
- How to preven the running program from acessing all resources?
  - Stored data



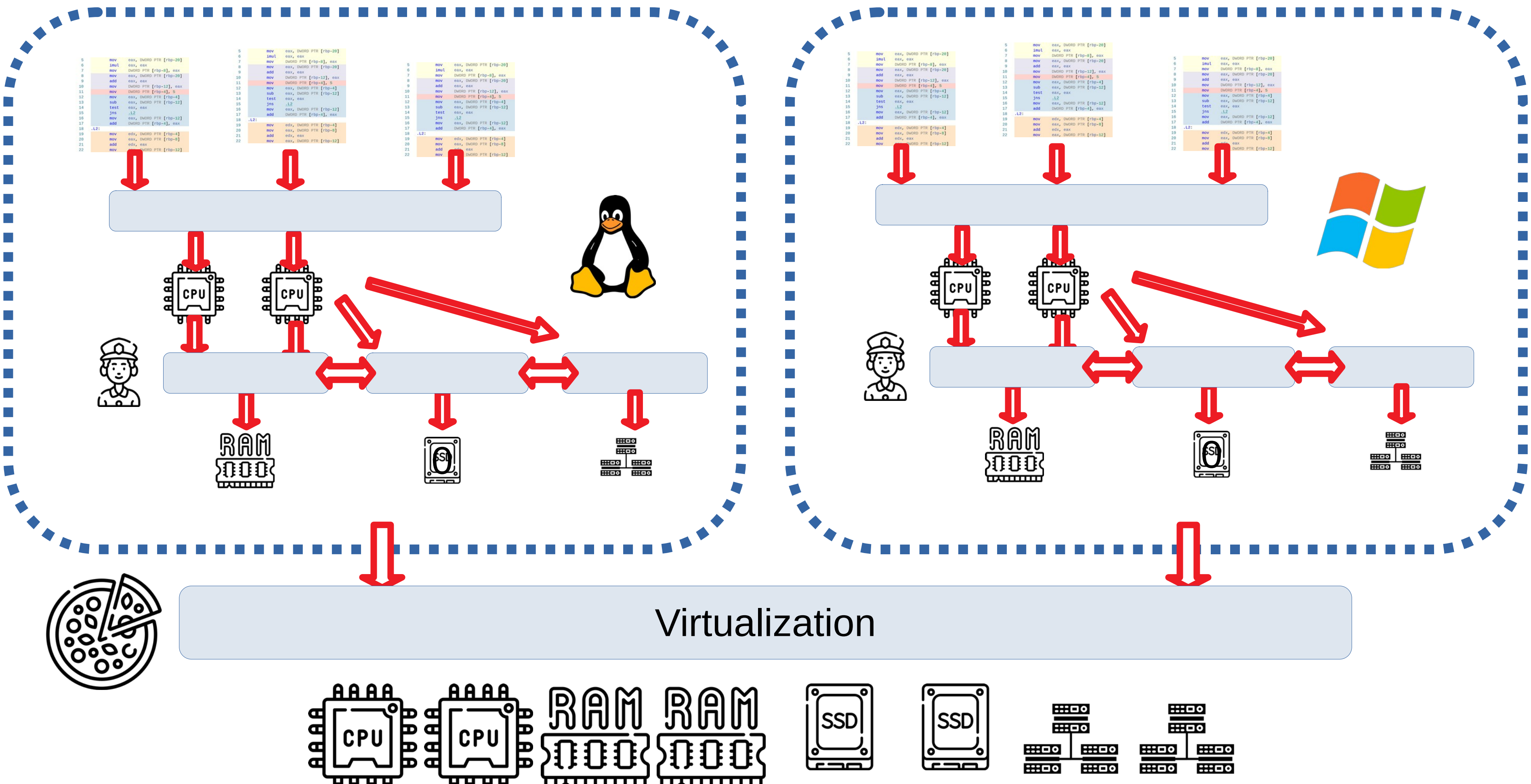
Icons by Flaticon.com.

Operating system



Icons by Flaticon.com.

Hypervisor



Icons by Flaticon.com.

Cloud computing

- Hypervisors allow resources to be pooled and sliced
  - Elasticity
  - Computing as an utility
- Available in Infrastructure as a Service (IaaS) from cloud providers
  - Cost effective for data storage and processing

Key Issue:
Exploiting cloud computing

Summary

- Key issues for distributed data processing:
  - Data movement
  - Parallelism
  - Coordination
  - Financial cost
  - Fault tolerance

- We will often justify design and implementation decisions with these issues!