

Computer Architecture
Lecture Week 2

Fundamental Ideas of Computer Architecture

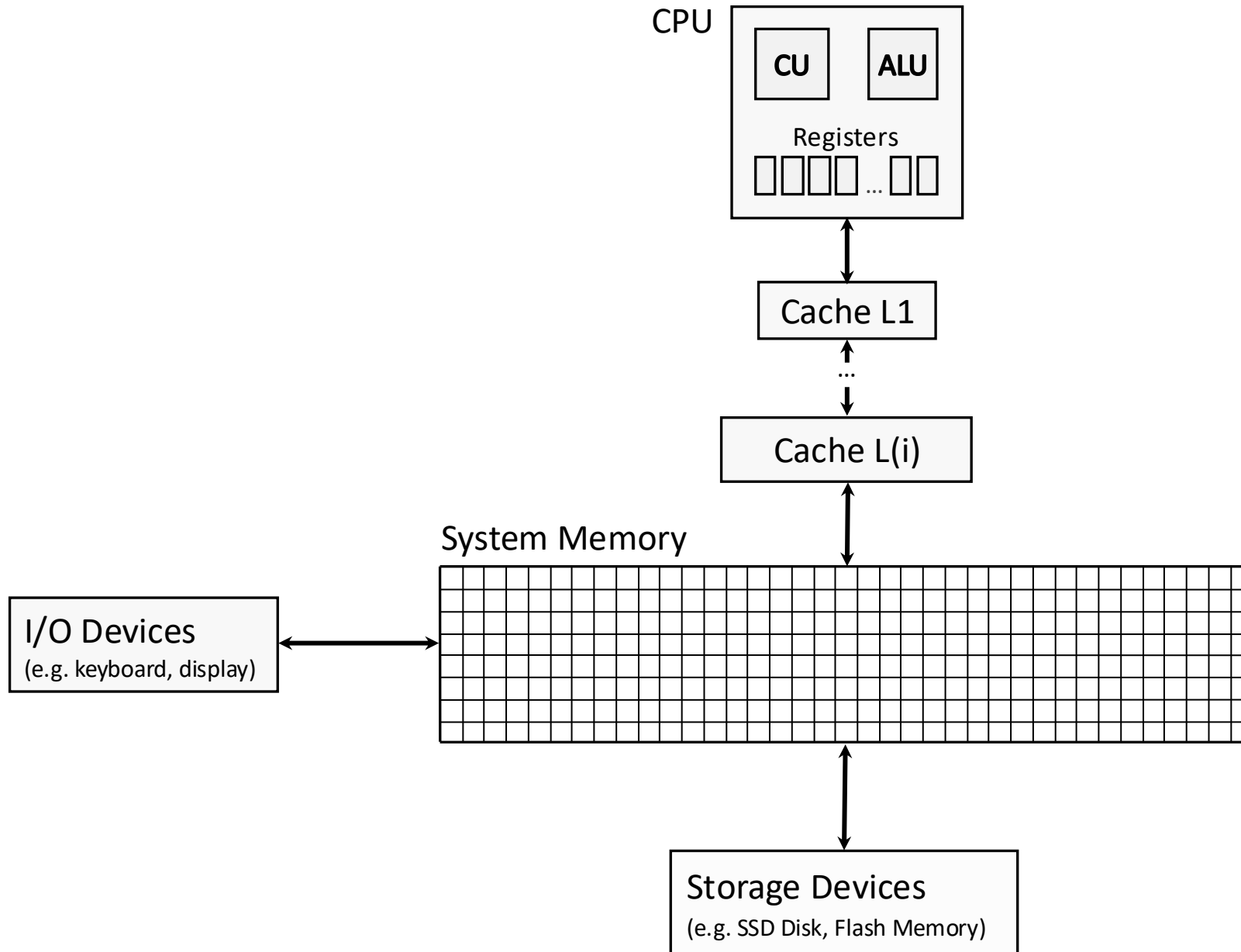
Artem Burmyakov

September 11, 2026



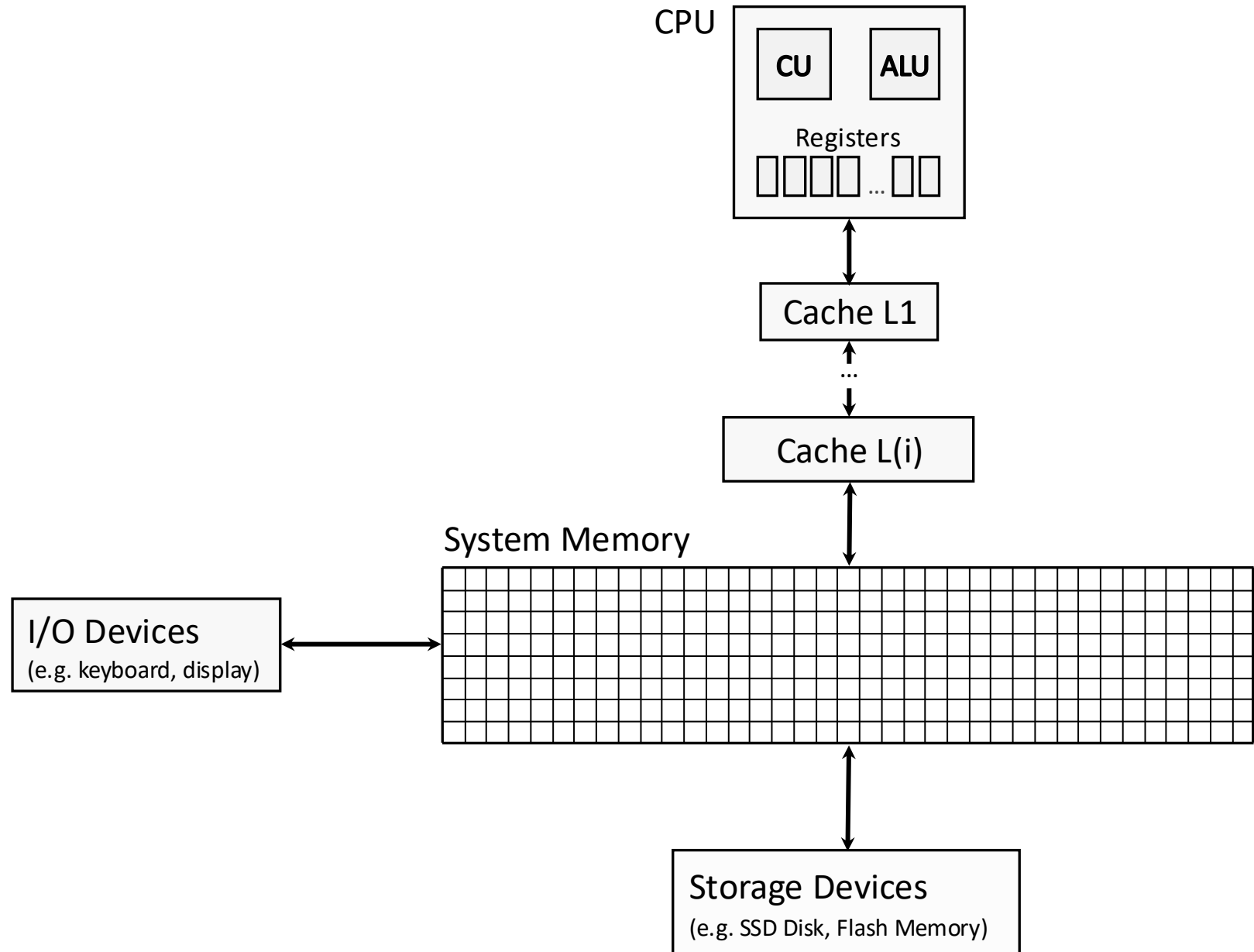
INNOPOLIS
UNIVERSITY

Computer Layout



Q1: Which memory unit provides the fastest access?

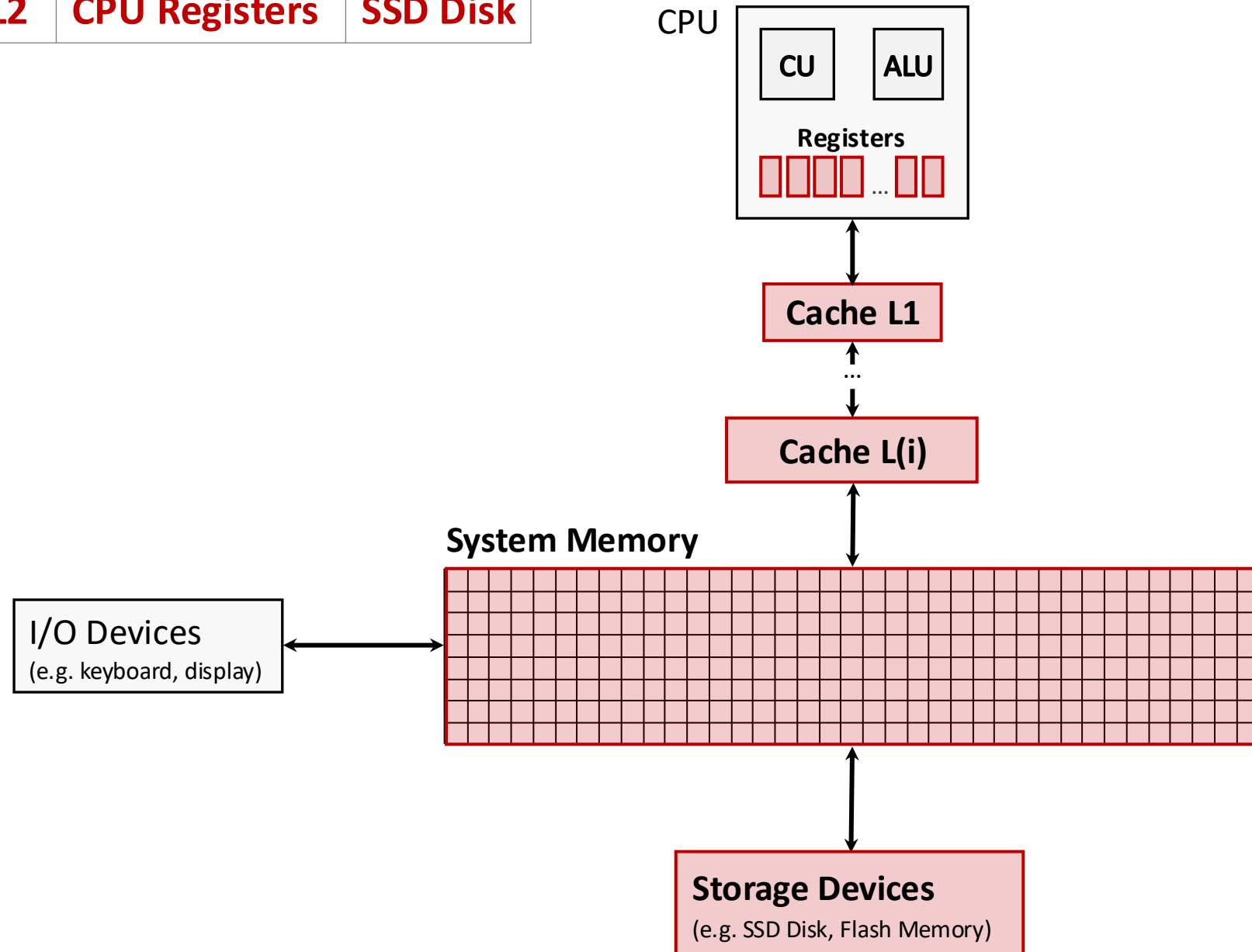
Computer Layout



Q1: Which memory unit provides the fastest access?

System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	---------------	----------

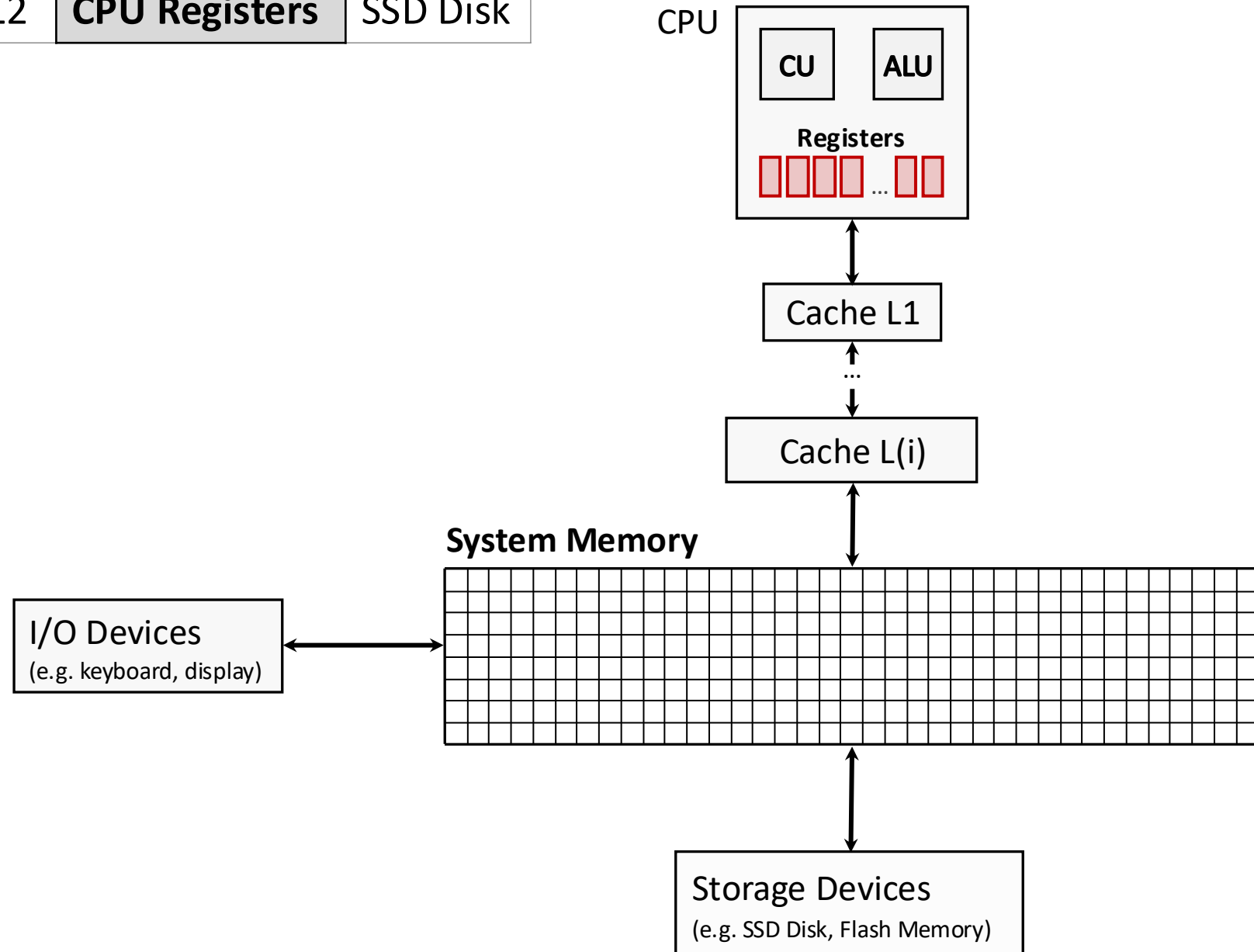
Computer Layout



Q1: Which memory unit provides the fastest access?

System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	----------------------	----------

Computer Layout



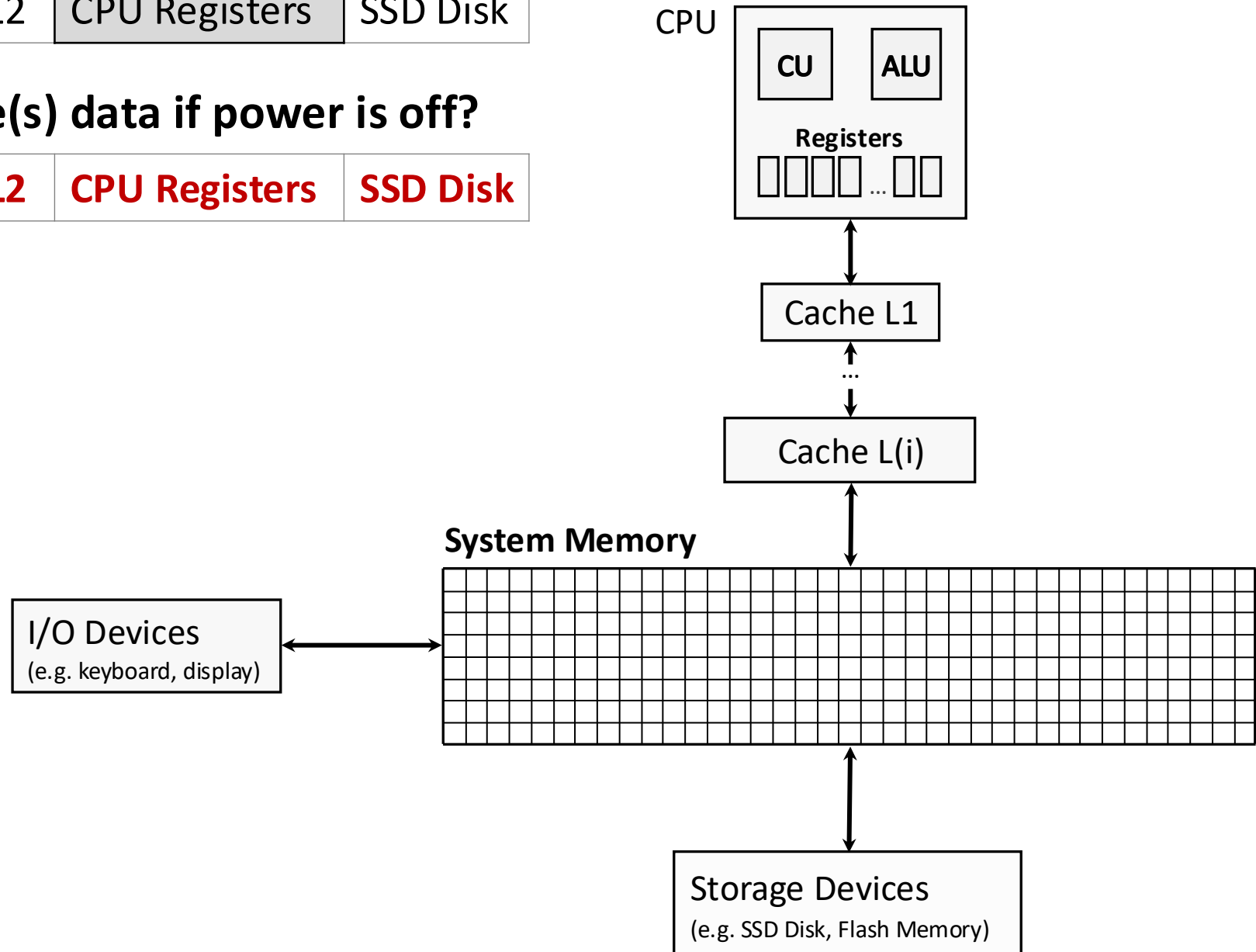
Q1: Which memory unit provides the fastest access?

System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	---------------	----------

Q2: Which memory unit(s) lose(s) data if power is off?

System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	---------------	----------

Recap: Computer Layout



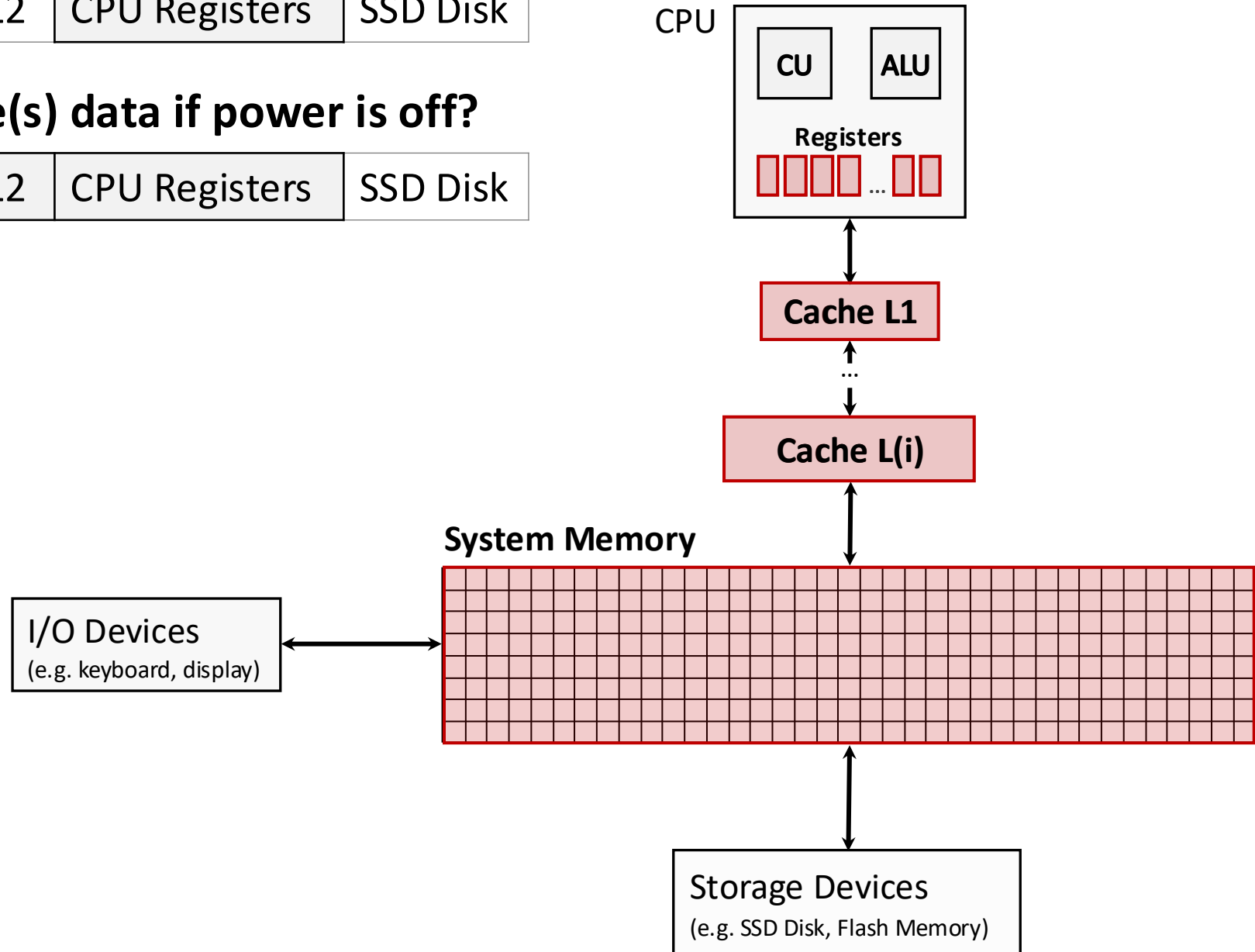
Q1: Which memory unit provides the fastest access?

System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	---------------	----------

Q2: Which memory unit(s) lose(s) data if power is off?

System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	---------------	----------

Recap: Computer Layout



Q1: Which memory unit provides the fastest access?

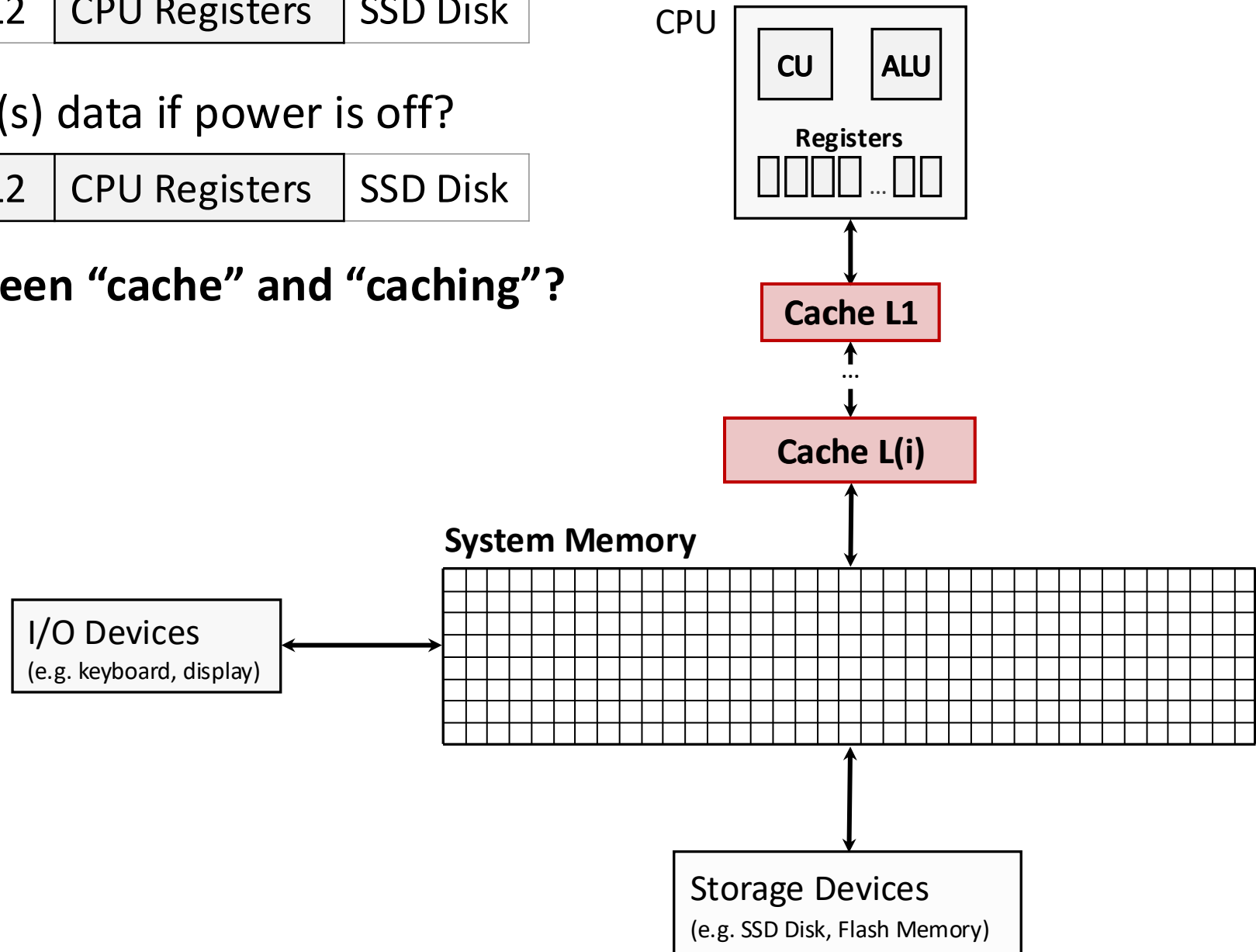
System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	---------------	----------

Q2: Which memory unit(s) lose(s) data if power is off?

System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	---------------	----------

Q3: What is the difference between “cache” and “caching”?

Recap: Computer Layout



Q1: Which memory unit provides the fastest access?

System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	---------------	----------

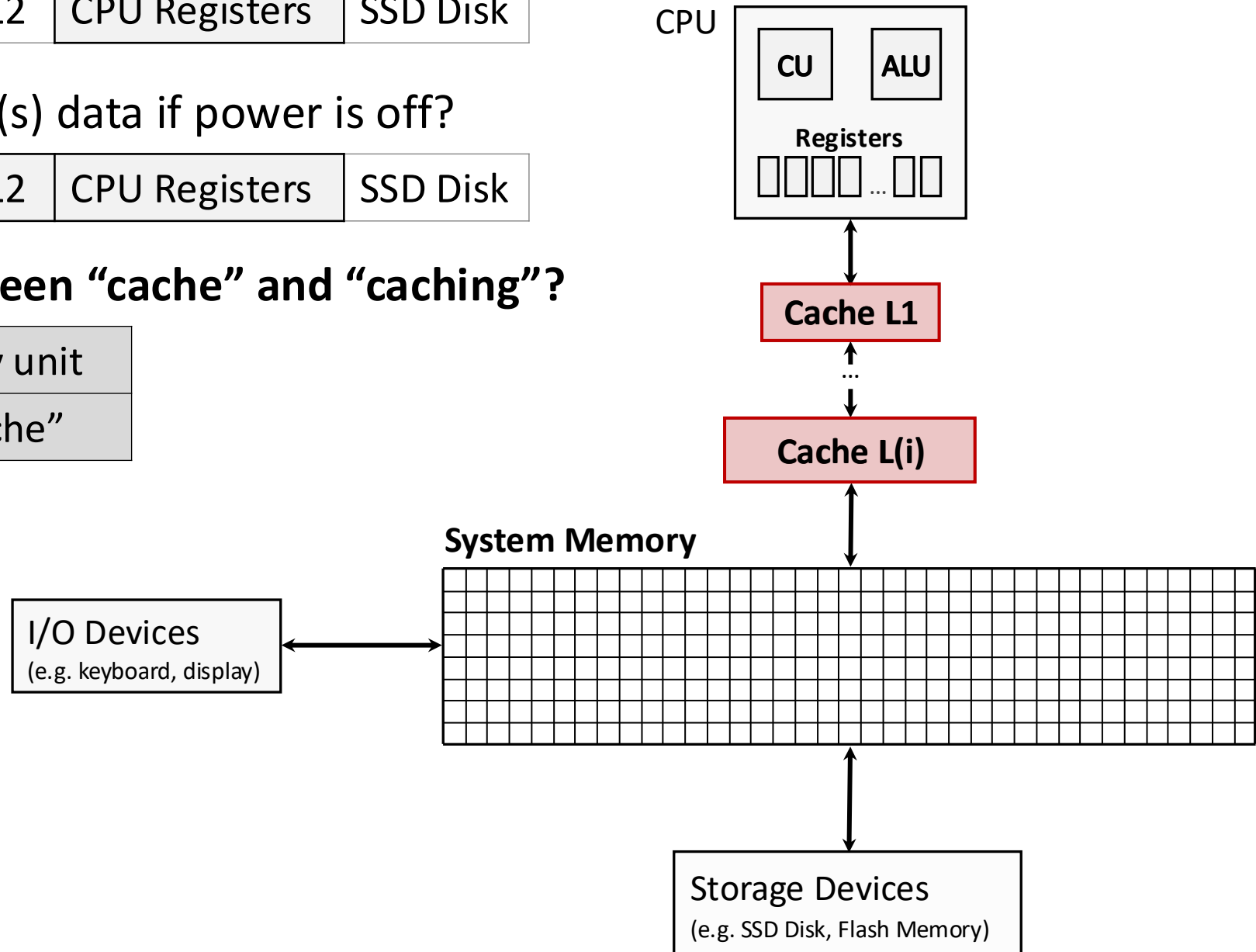
Q2: Which memory unit(s) lose(s) data if power is off?

System Memory	Cache L1	Cache L2	CPU Registers	SSD Disk
---------------	----------	----------	---------------	----------

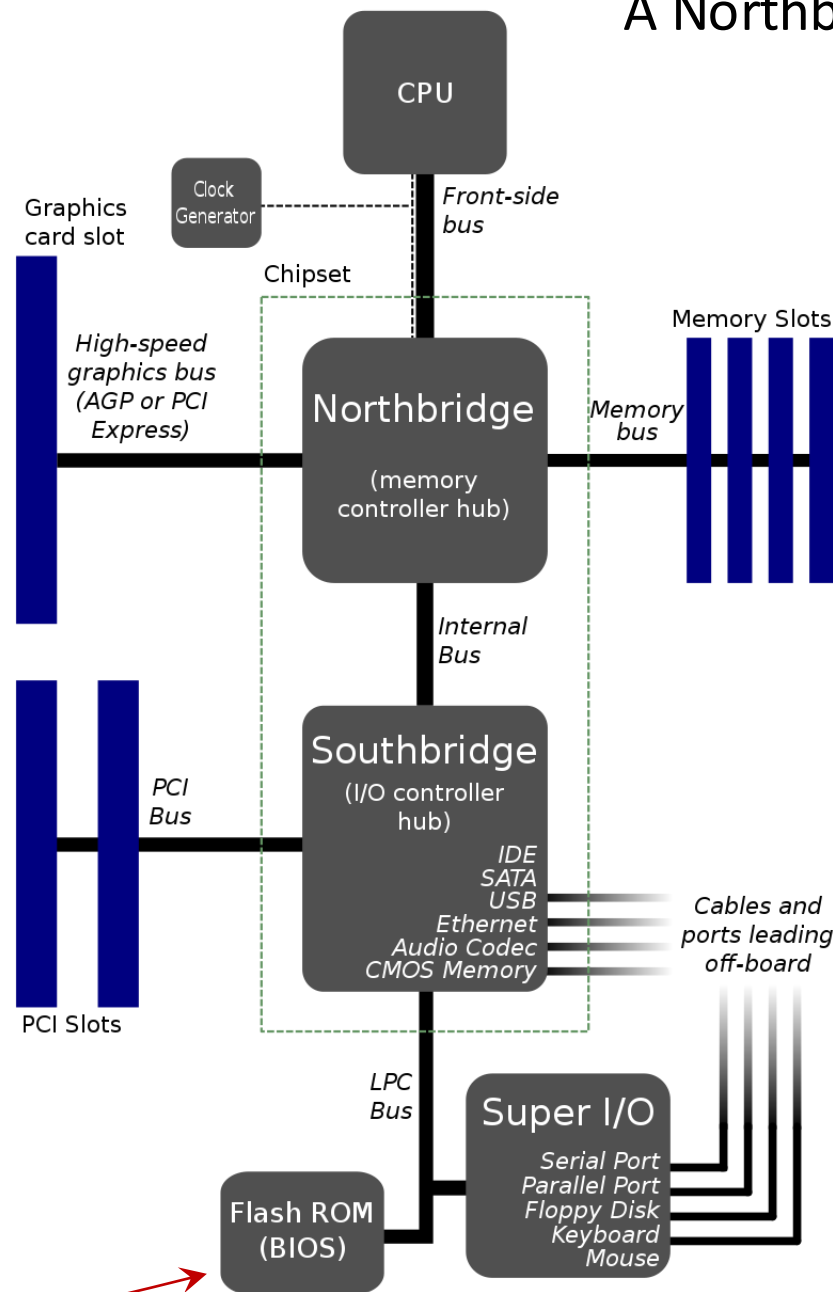
Q3: What is the difference between “cache” and “caching”?

Cache	Physical or logical memory unit
Caching	Data pre-fetching into “cache”

Recap: Computer Layout



A Northbridge / Southbridge Hardware Layout

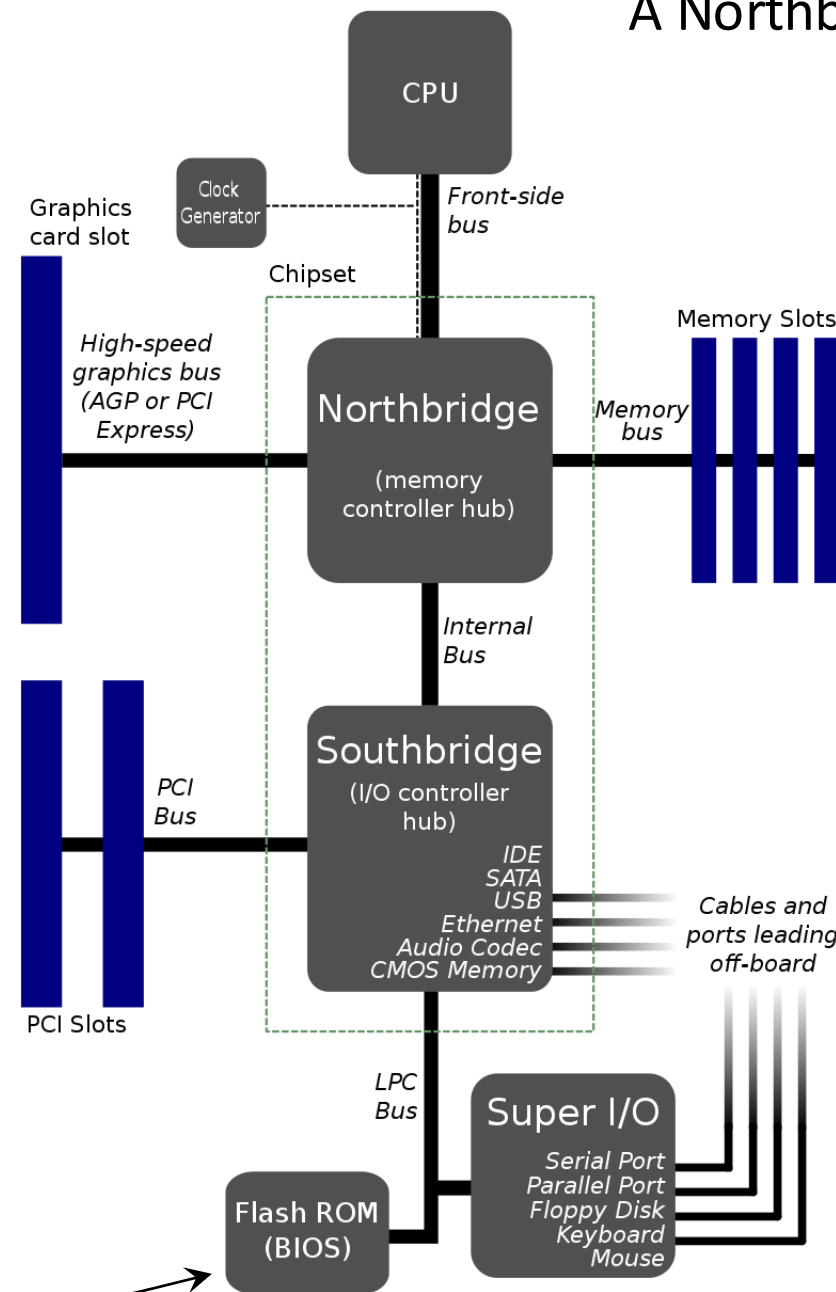


Read-Only Memory (ROM),
e.g. for BIOS

BIOS: Basic Input/output System

A Northbridge / Southbridge Hardware Layout

Does “BIOS” uses CPU?

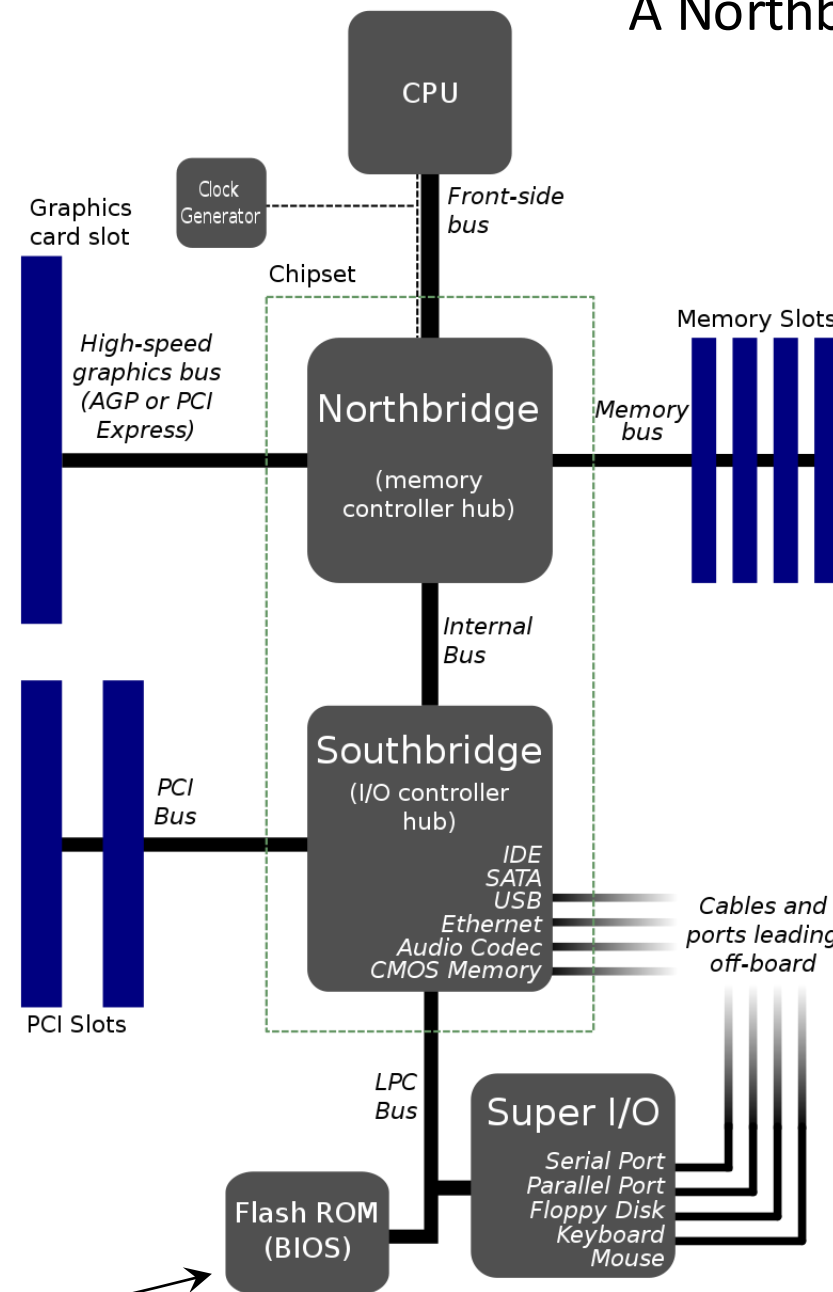


Read-Only Memory (ROM),
e.g. for BIOS

BIOS: Basic Input/output System

Does “BIOS” uses CPU?

Yes



Read-Only Memory (ROM),
e.g. for BIOS

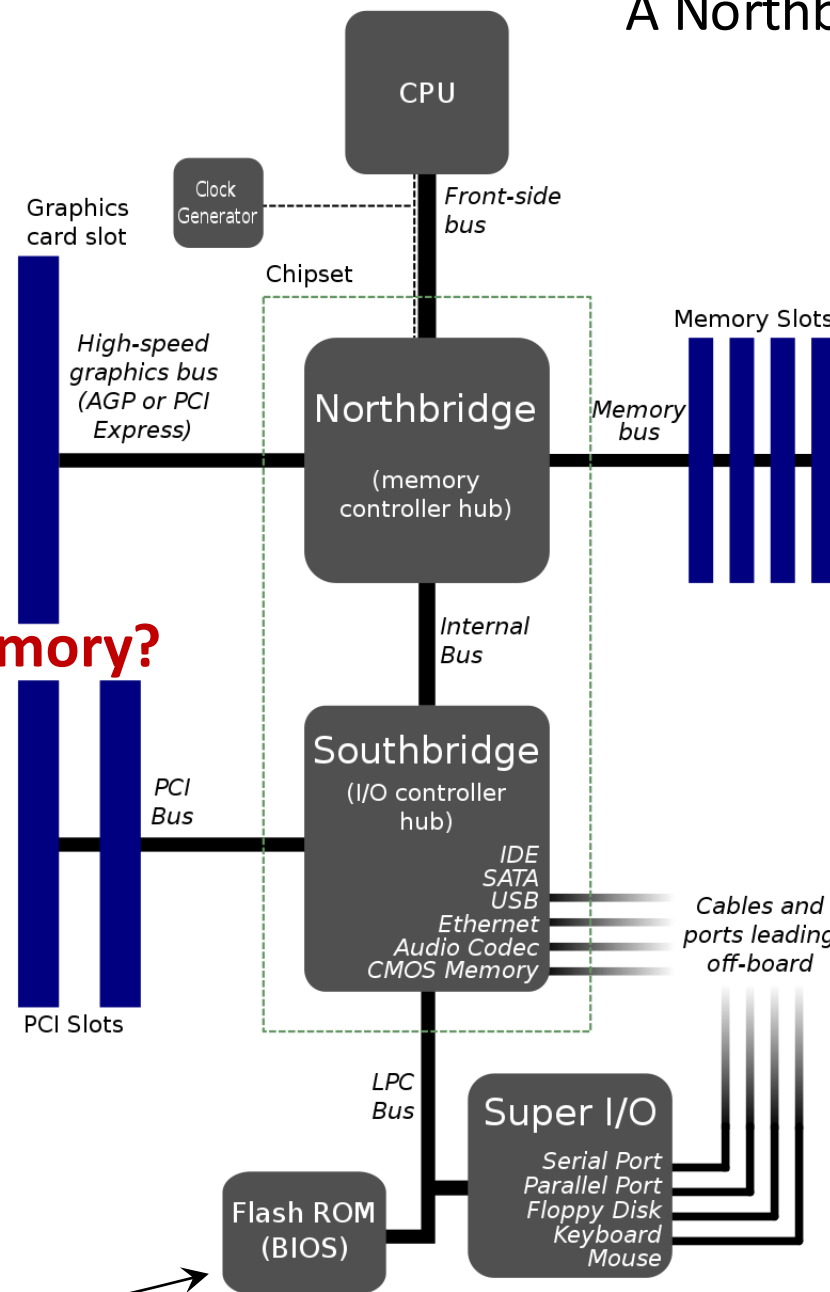
BIOS: Basic Input/output System

A Northbridge / Southbridge Hardware Layout

Does “BIOS” uses CPU?

Yes

Does “BIOS” uses system memory?



Read-Only Memory (ROM),
e.g. for BIOS

BIOS: Basic Input/output System

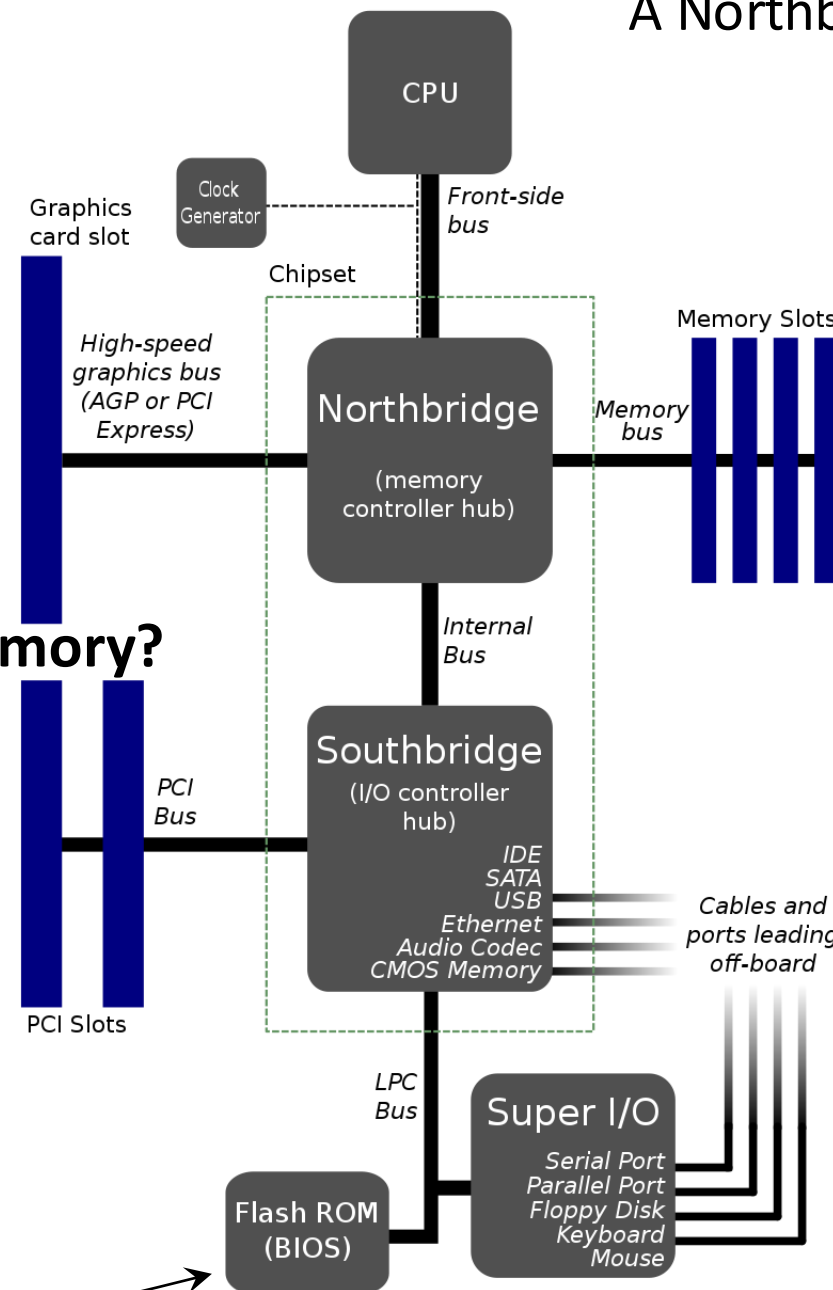
A Northbridge / Southbridge Hardware Layout

Does “BIOS” uses CPU?

Yes

Does “BIOS” uses system memory?

Yes



Read-Only Memory (ROM),
e.g. for BIOS

BIOS: Basic Input/output System

A Northbridge / Southbridge Hardware Layout

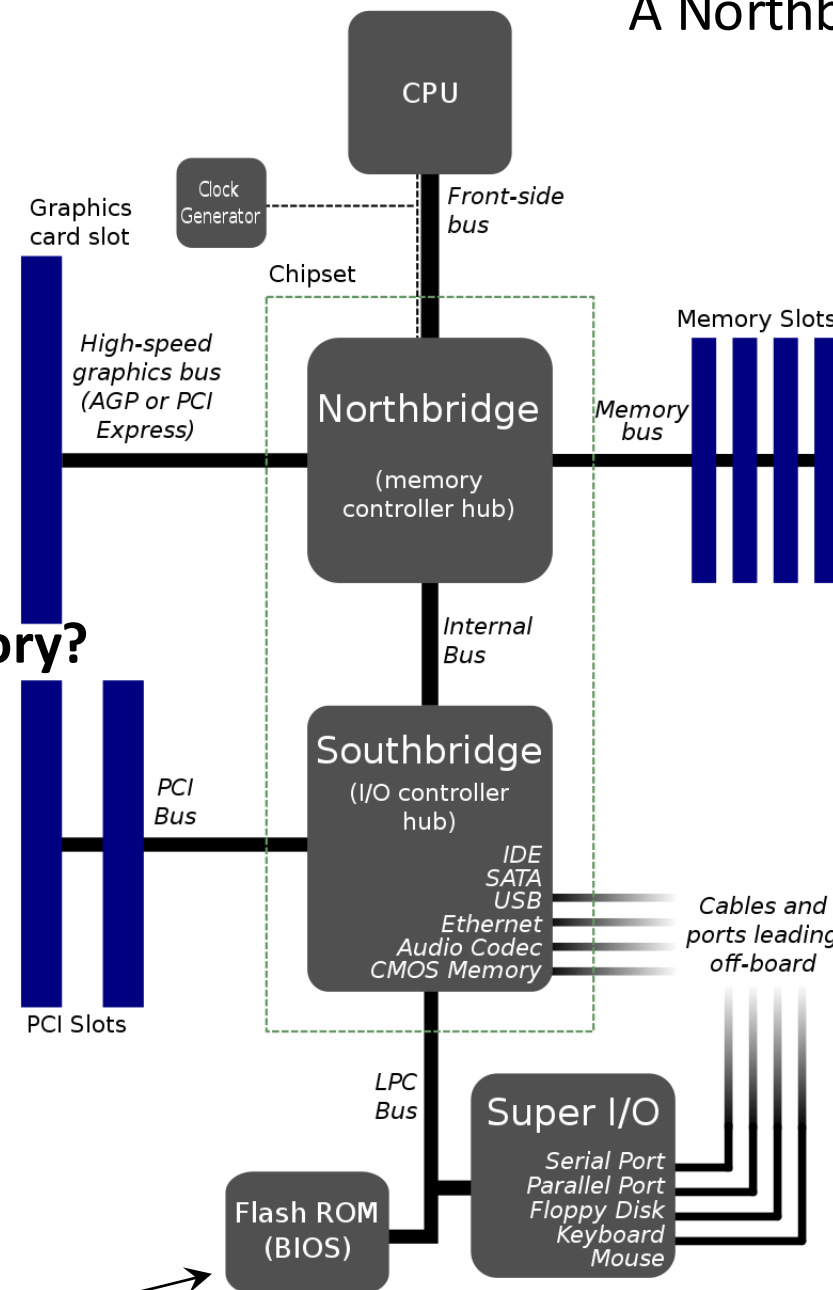
Does “BIOS” uses CPU?

Yes

Does “BIOS” uses system memory?

Yes

Does “BIOS” uses caches?



Read-Only Memory (ROM),
e.g. for BIOS

BIOS: Basic Input/output System

A Northbridge / Southbridge Hardware Layout

Does “BIOS” uses CPU?

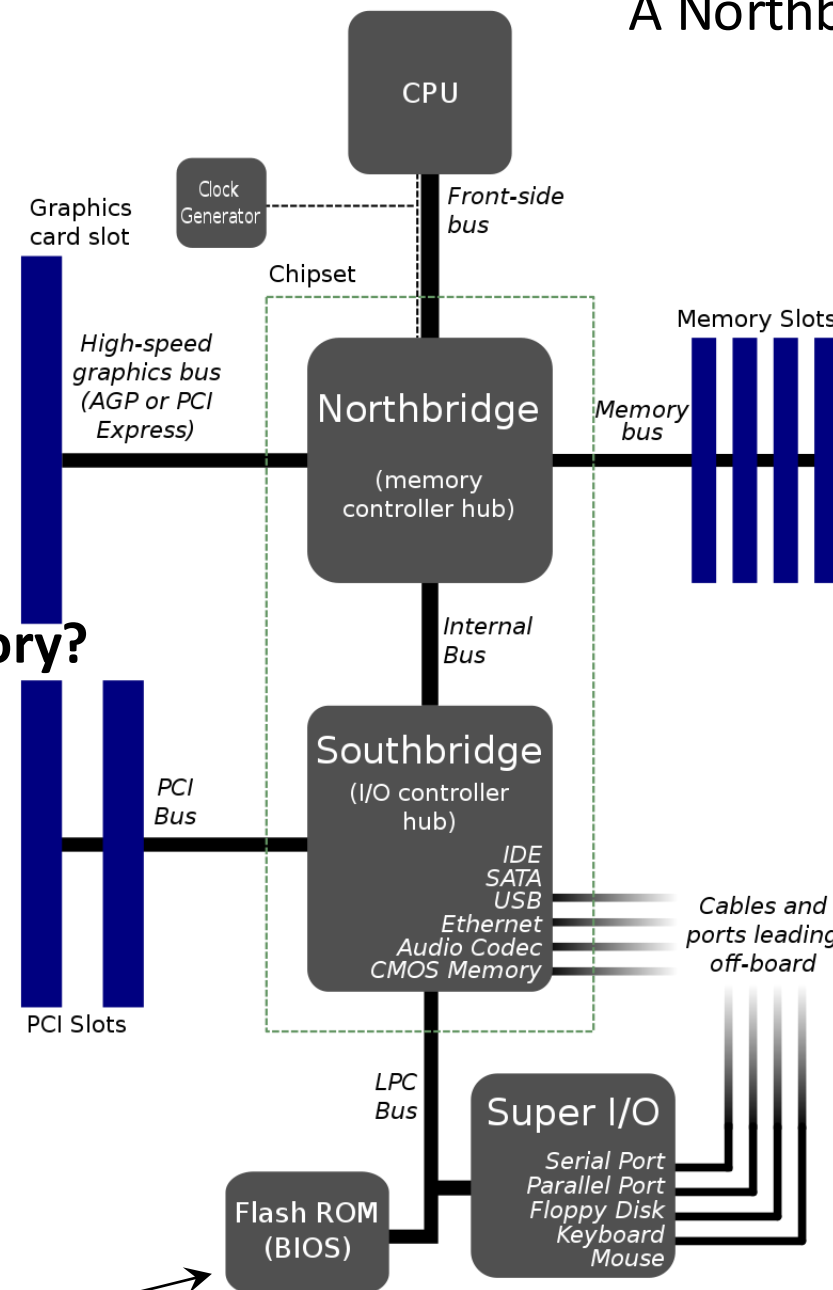
Yes

Does “BIOS” uses system memory?

Yes

Does “BIOS” uses caches?

Yes



Read-Only Memory (ROM),
e.g. for BIOS

BIOS: Basic Input/output System

A Northbridge / Southbridge Hardware Layout

Does “BIOS” uses CPU?

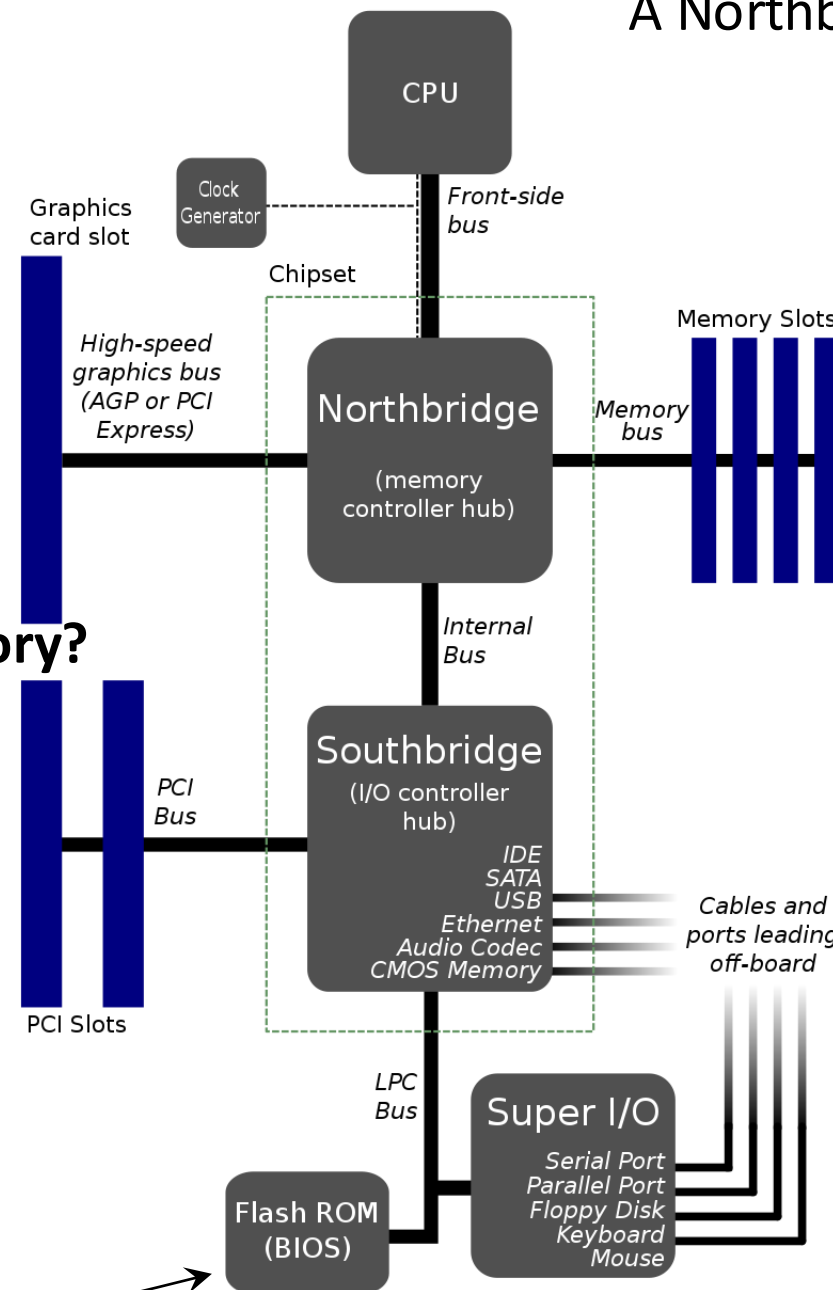
Yes

Does “BIOS” uses system memory?

Yes

Does “BIOS” uses caches?

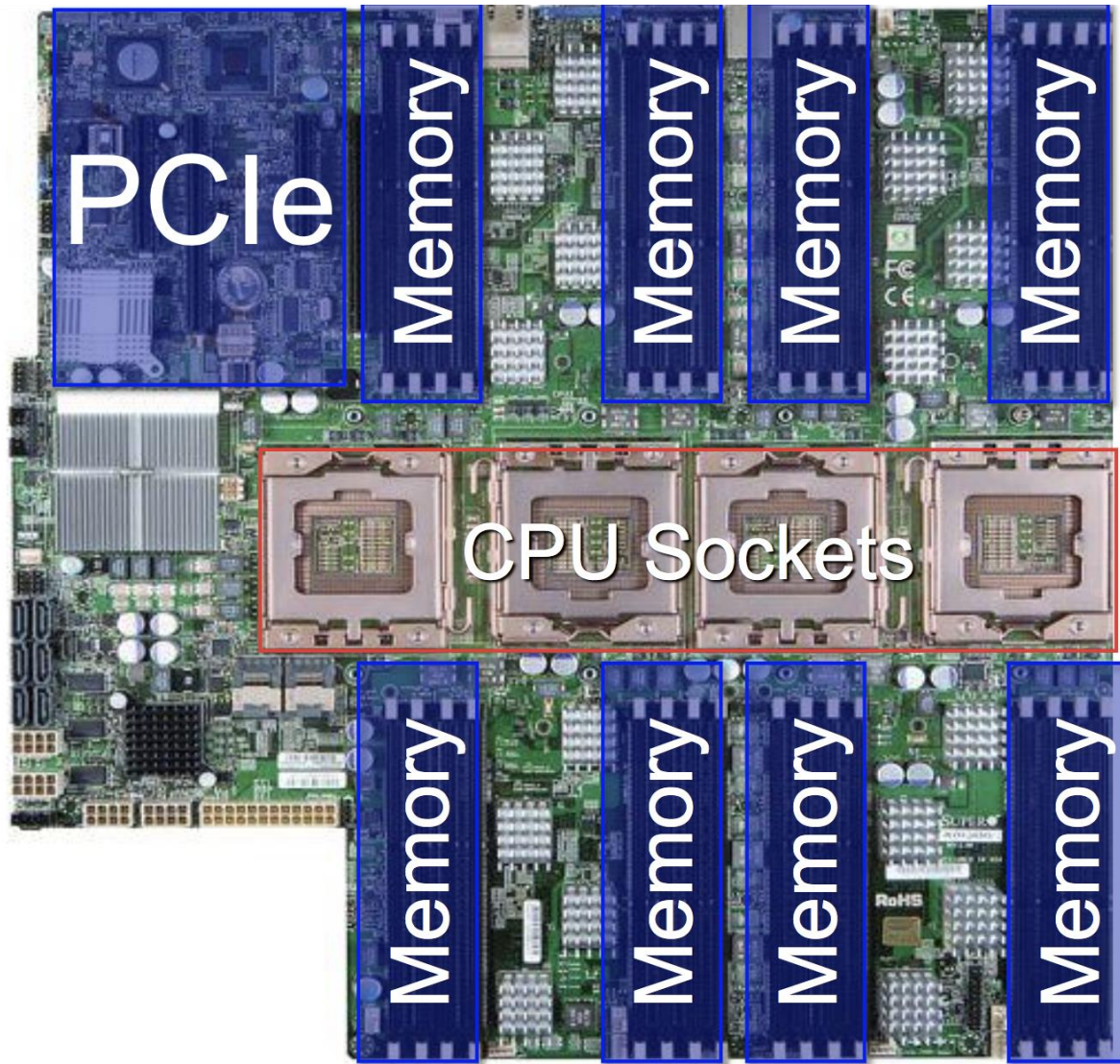
Yes



Read-Only Memory (ROM),
e.g. for BIOS

BIOS: Basic Input/output System

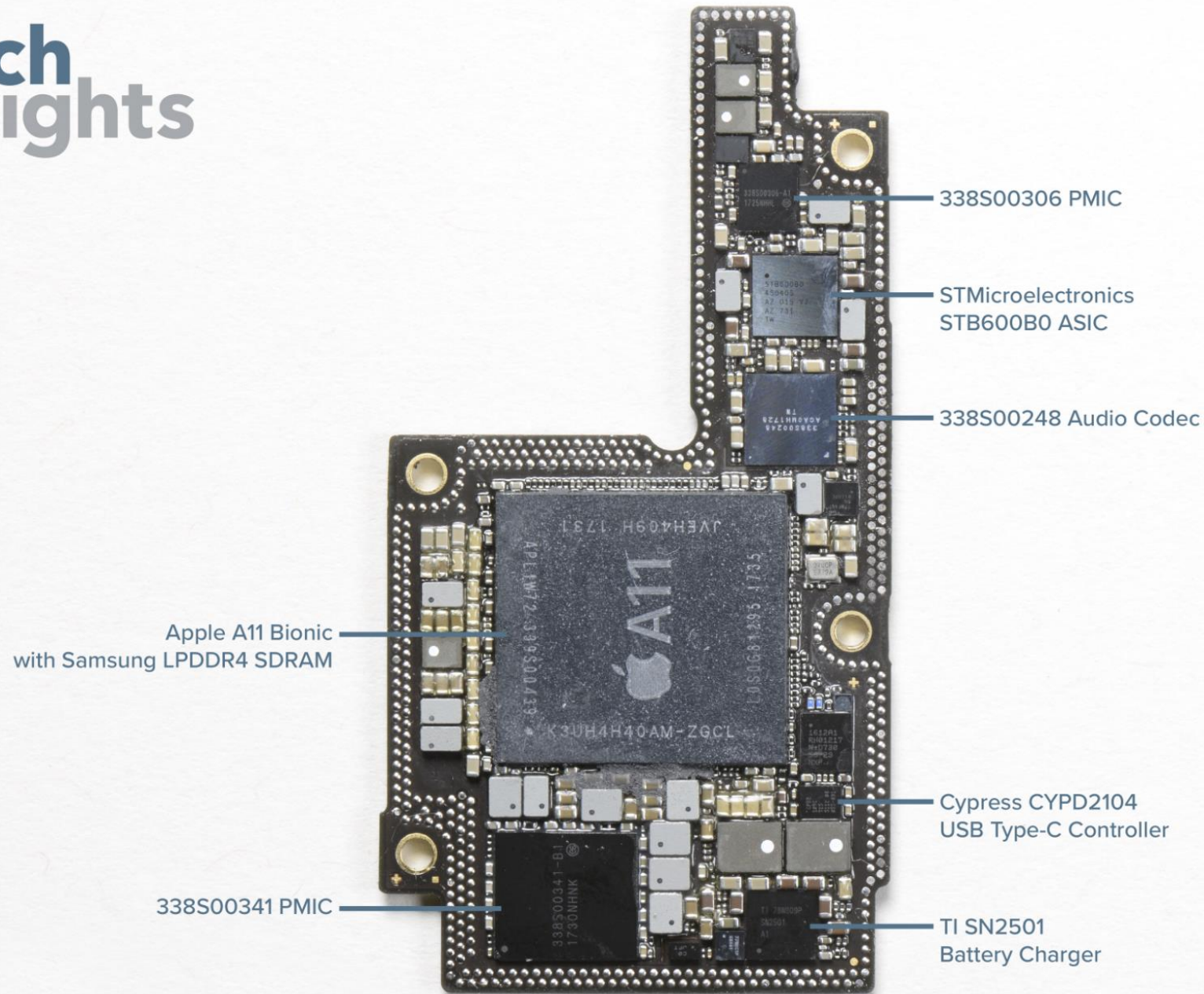
Example: Computer Server Architecture



Server has multiple CPUs and system memory slots

Another Example: iPhone X Architecture

Tech
Insights



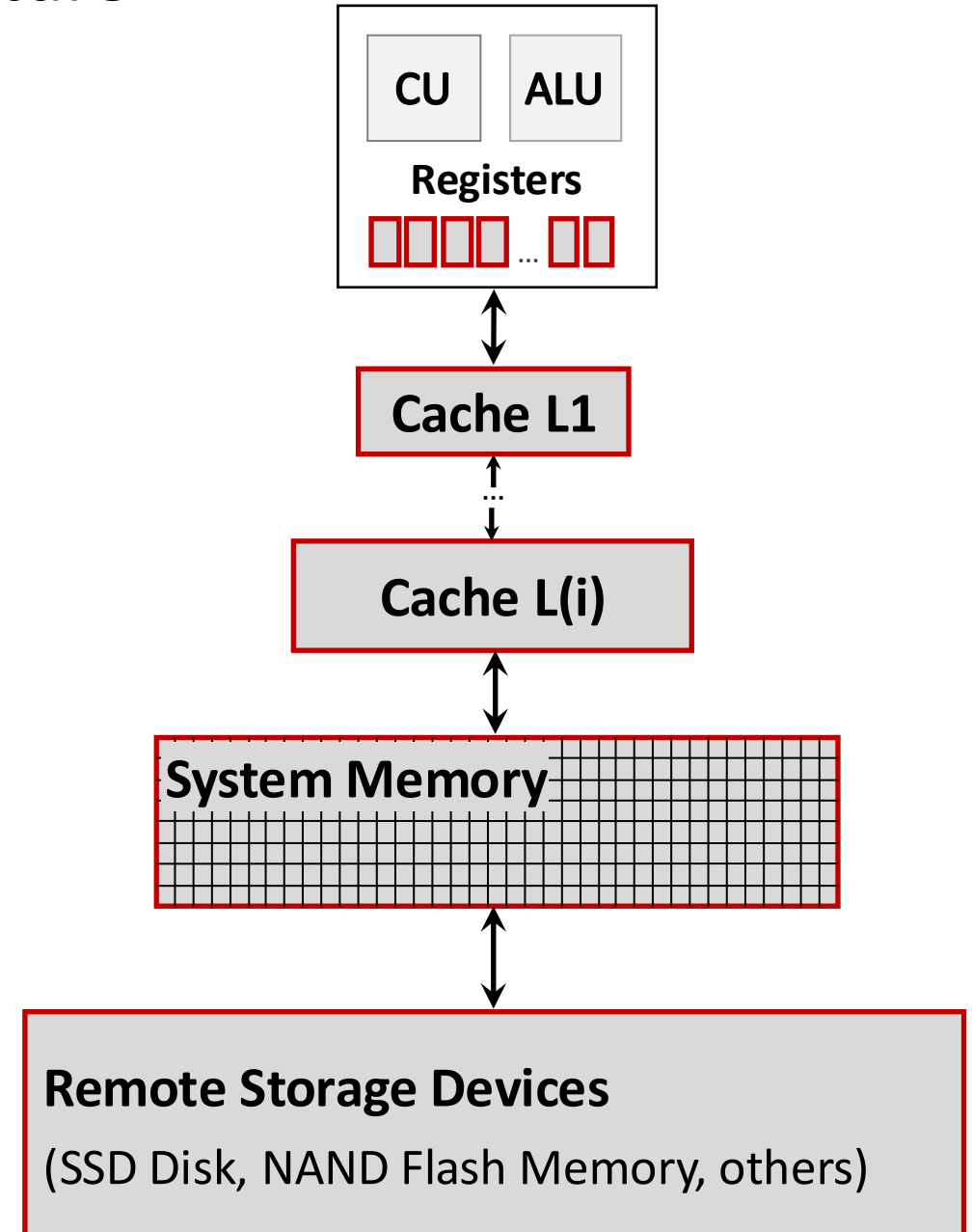
Fundamental Ideas of Computer Architecture

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories

Fundamental Ideas of Computer Architecture

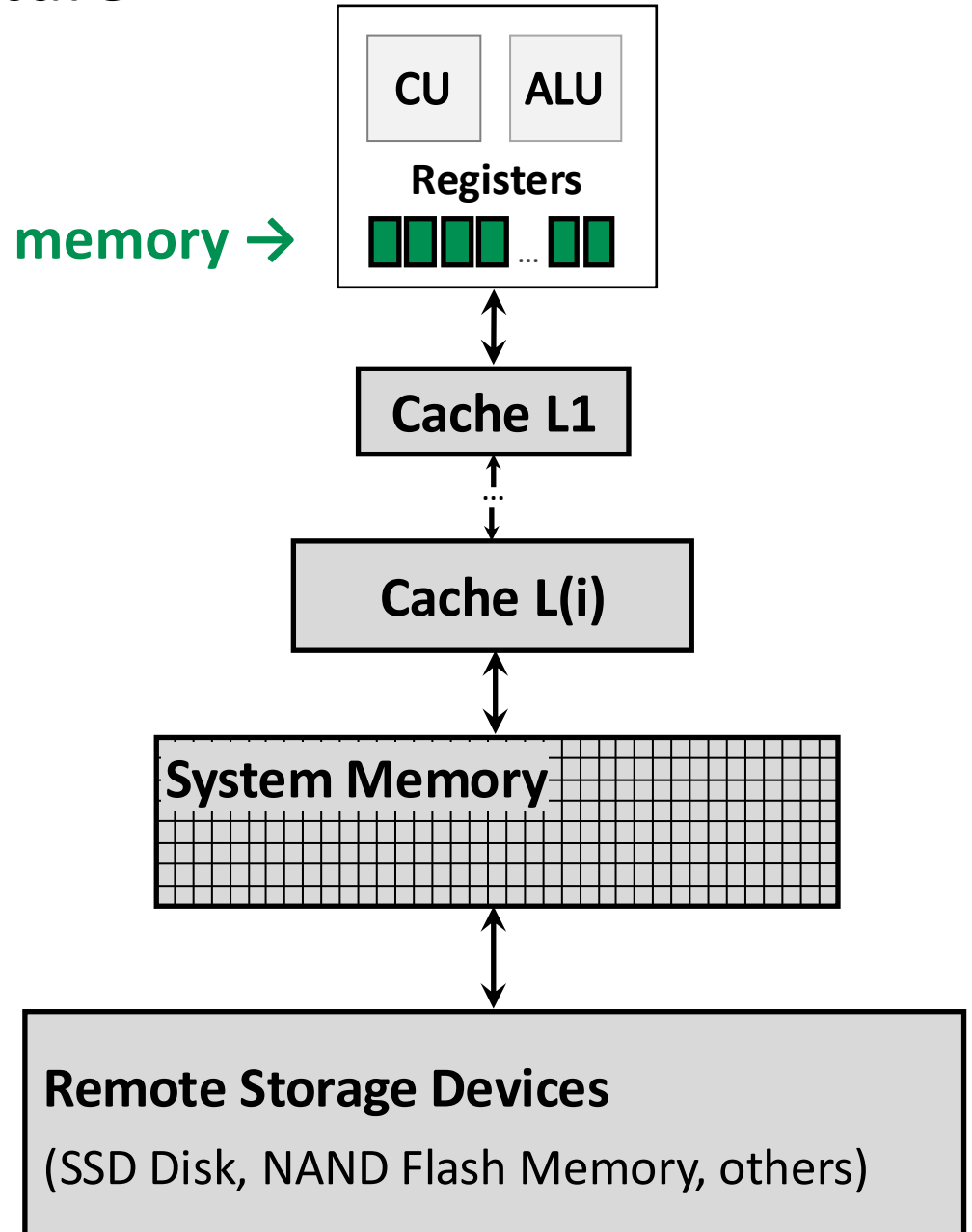
1. Hierarchy of memories



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories

The fastest memory →

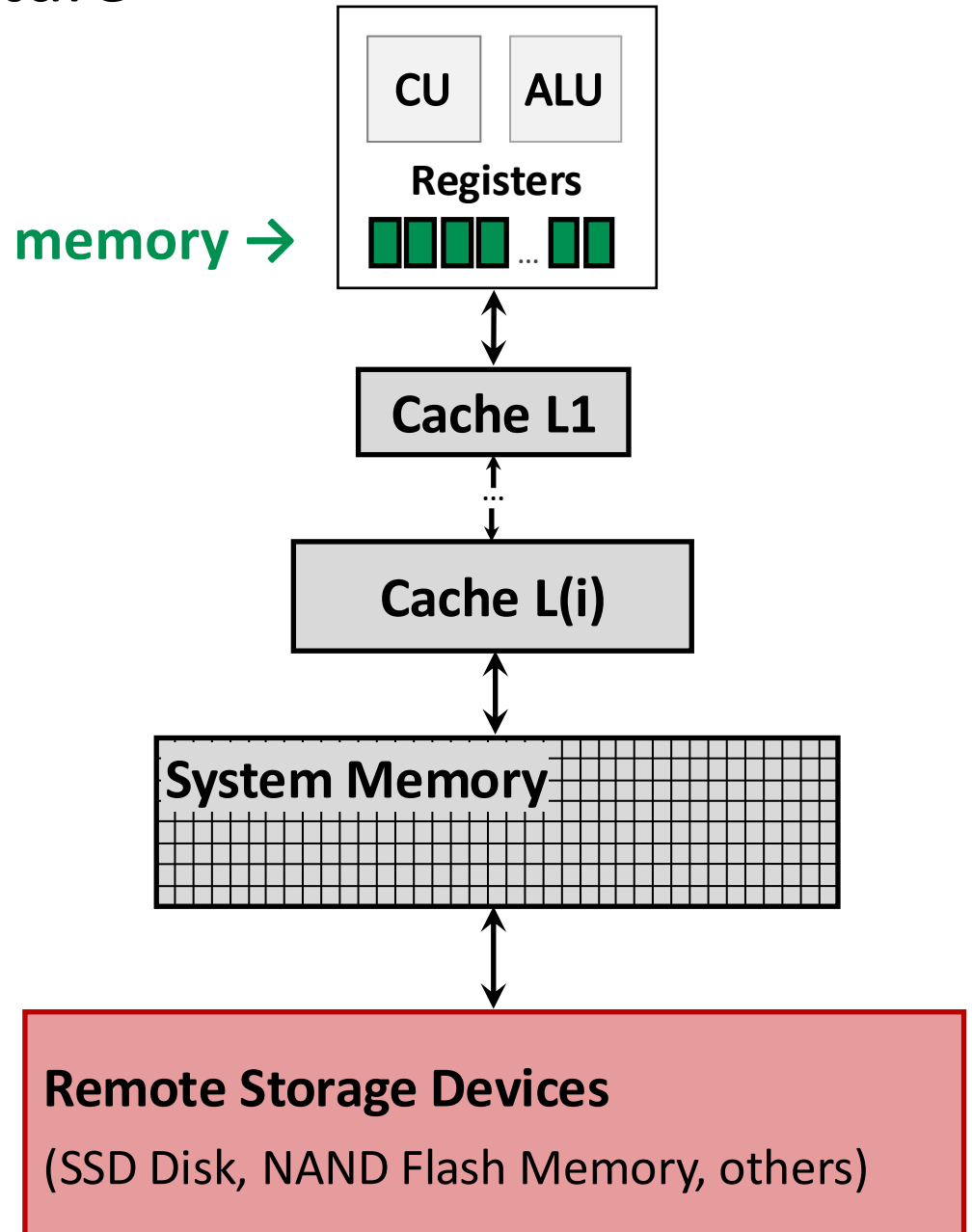


Fundamental Ideas of Computer Architecture

1. Hierarchy of memories

The slowest memory →

The fastest memory →

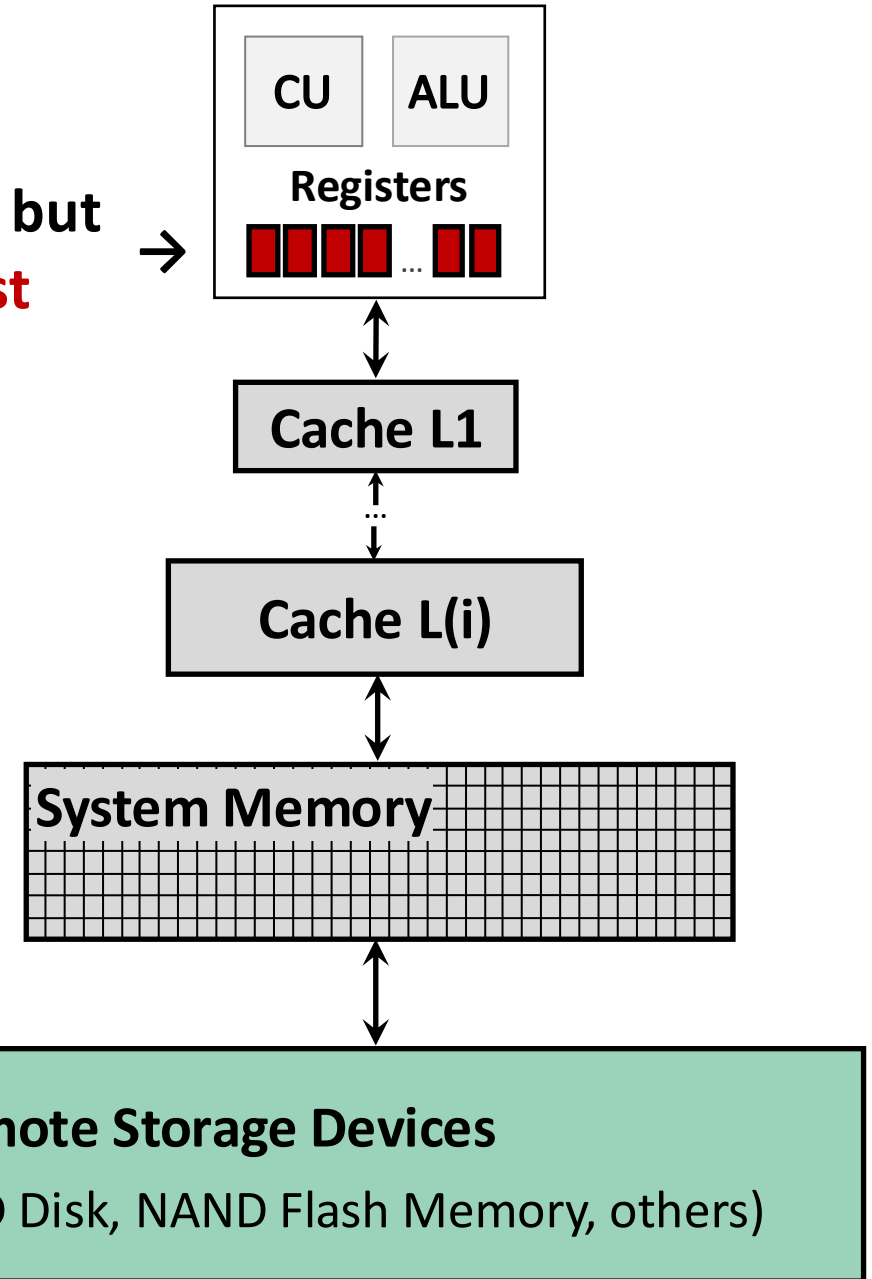


Fundamental Ideas of Computer Architecture

1. Hierarchy of memories

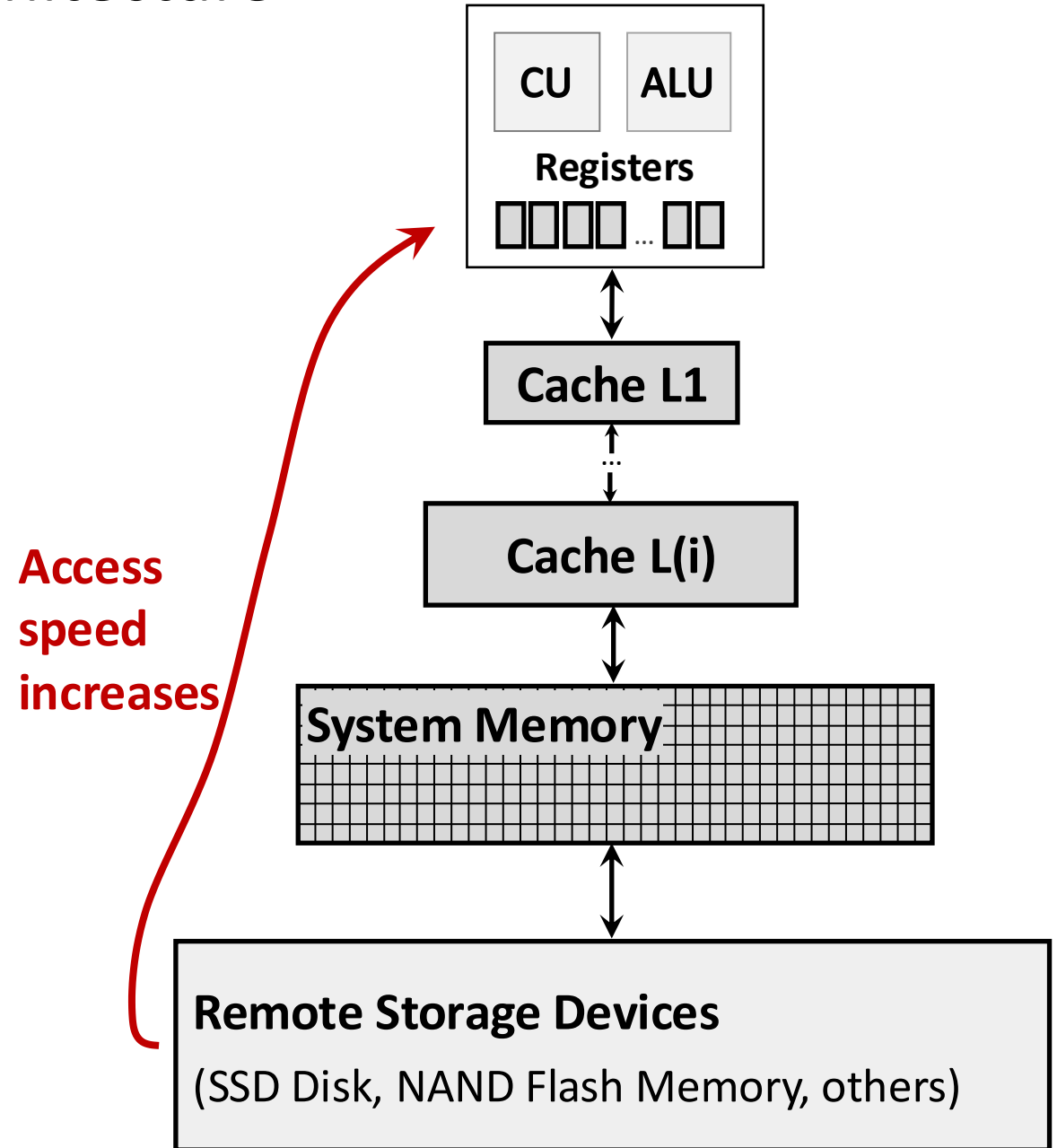
The slowest
but **the largest** →

The fastest but
the smallest →



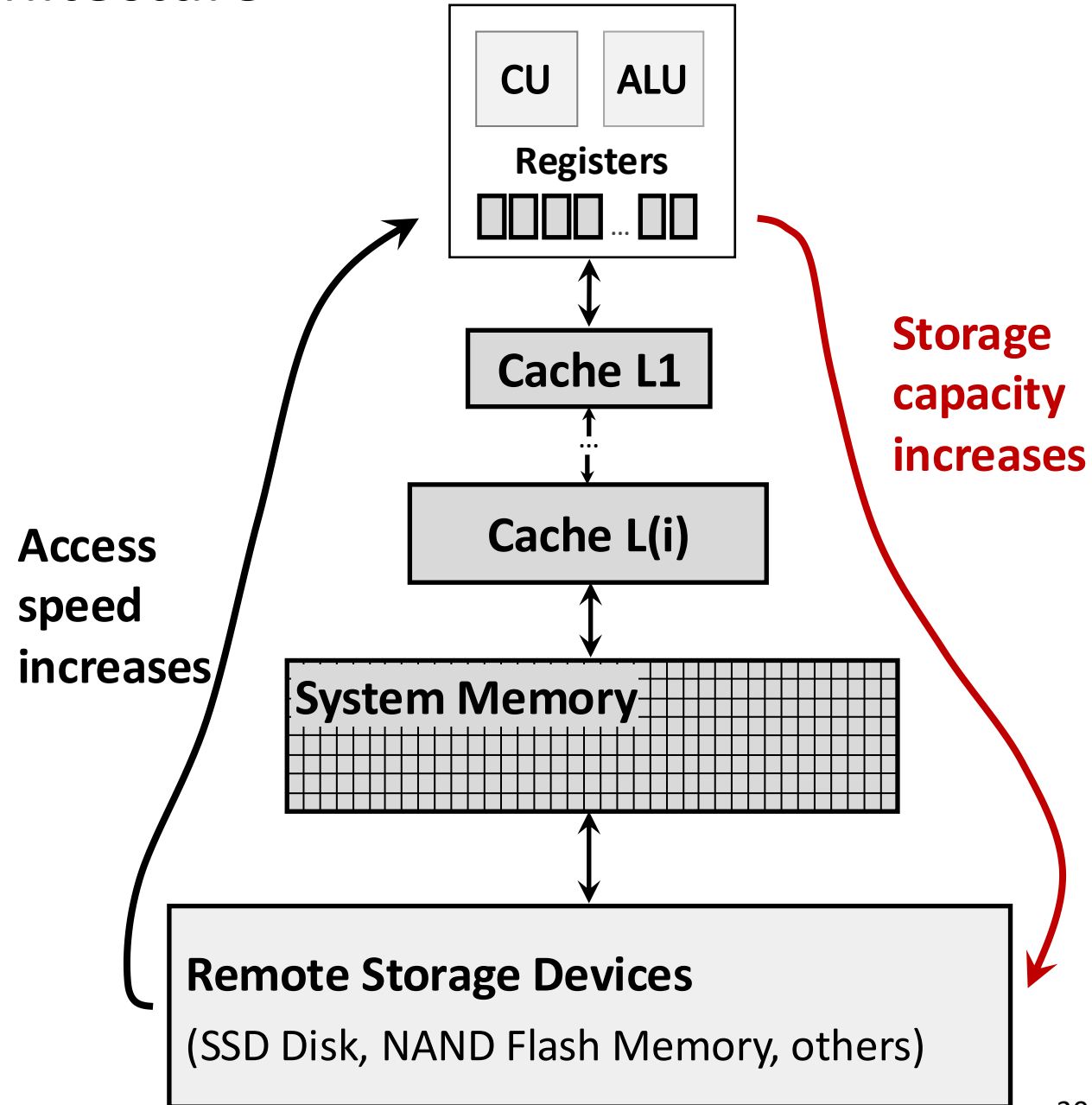
Fundamental Ideas of Computer Architecture

1. Hierarchy of memories



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories

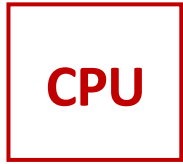


Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
- 2. Design simplification via abstraction use**

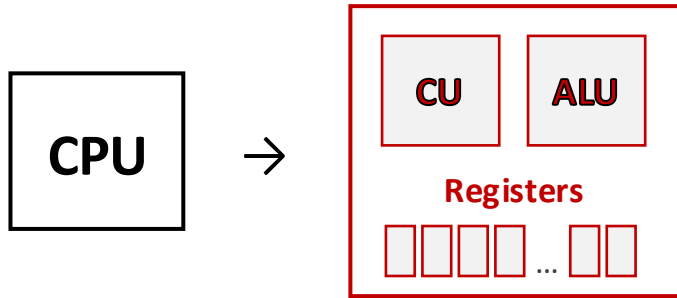
Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. **Design simplification via abstraction use** (a CPU example)



Fundamental Ideas of Computer Architecture

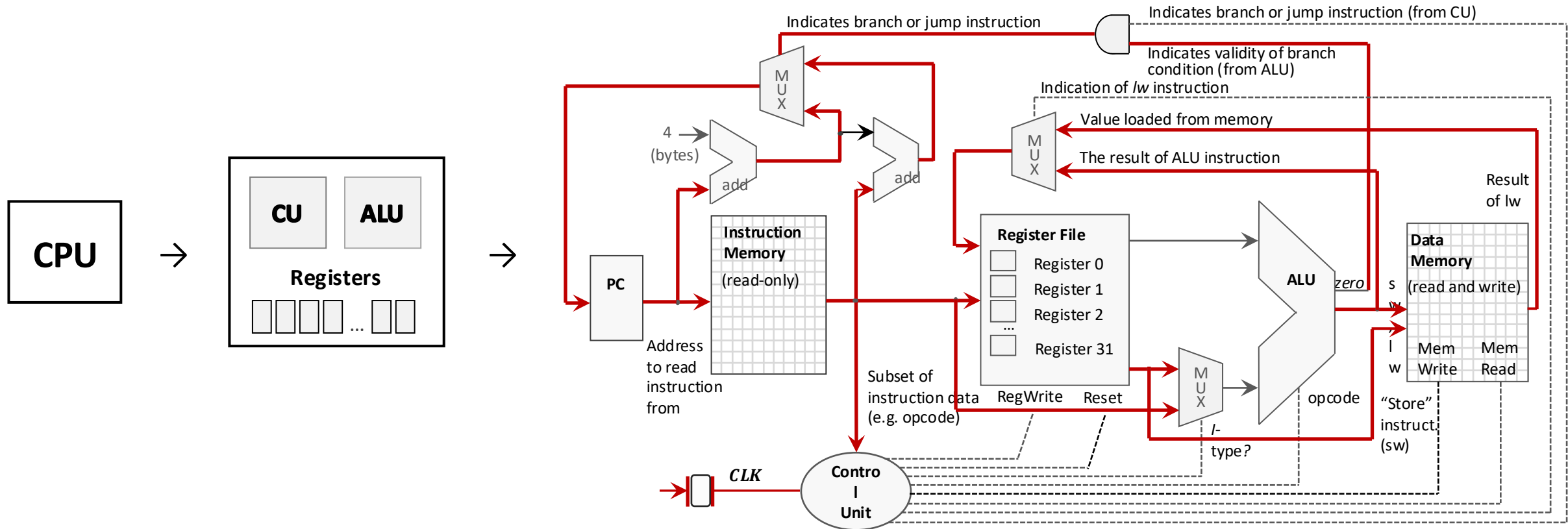
1. Hierarchy of memories
- 2. Design simplification via abstraction use (a CPU example)**



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories

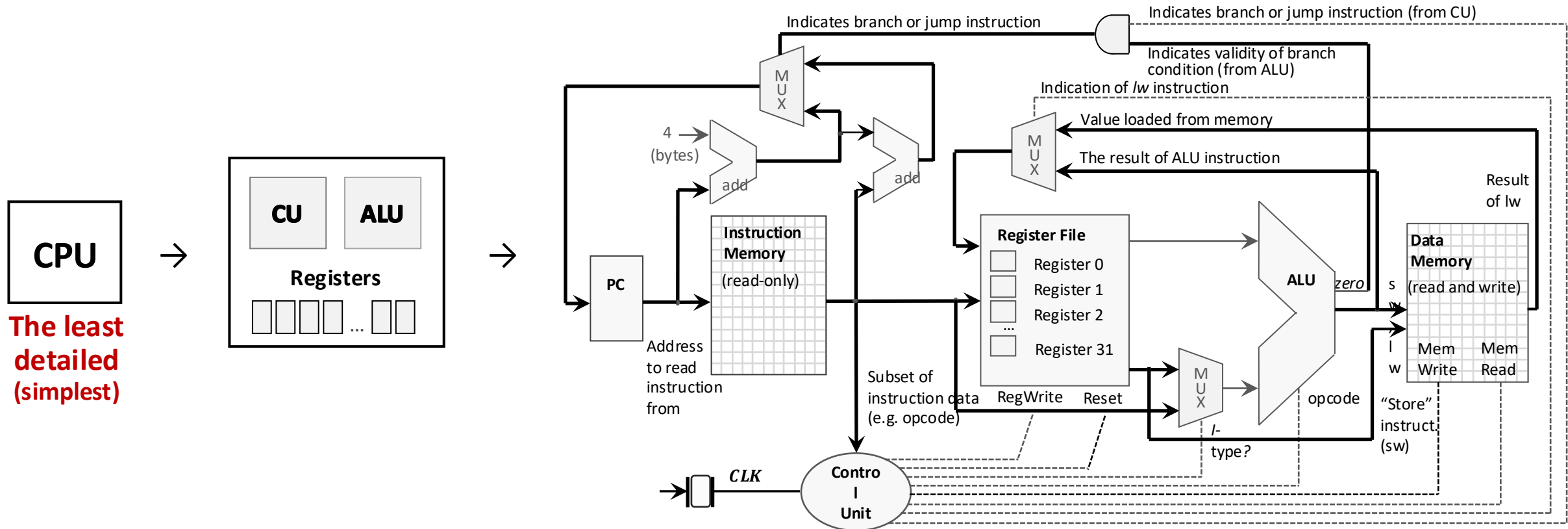
2. Design simplification via abstraction use (a CPU example)



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories

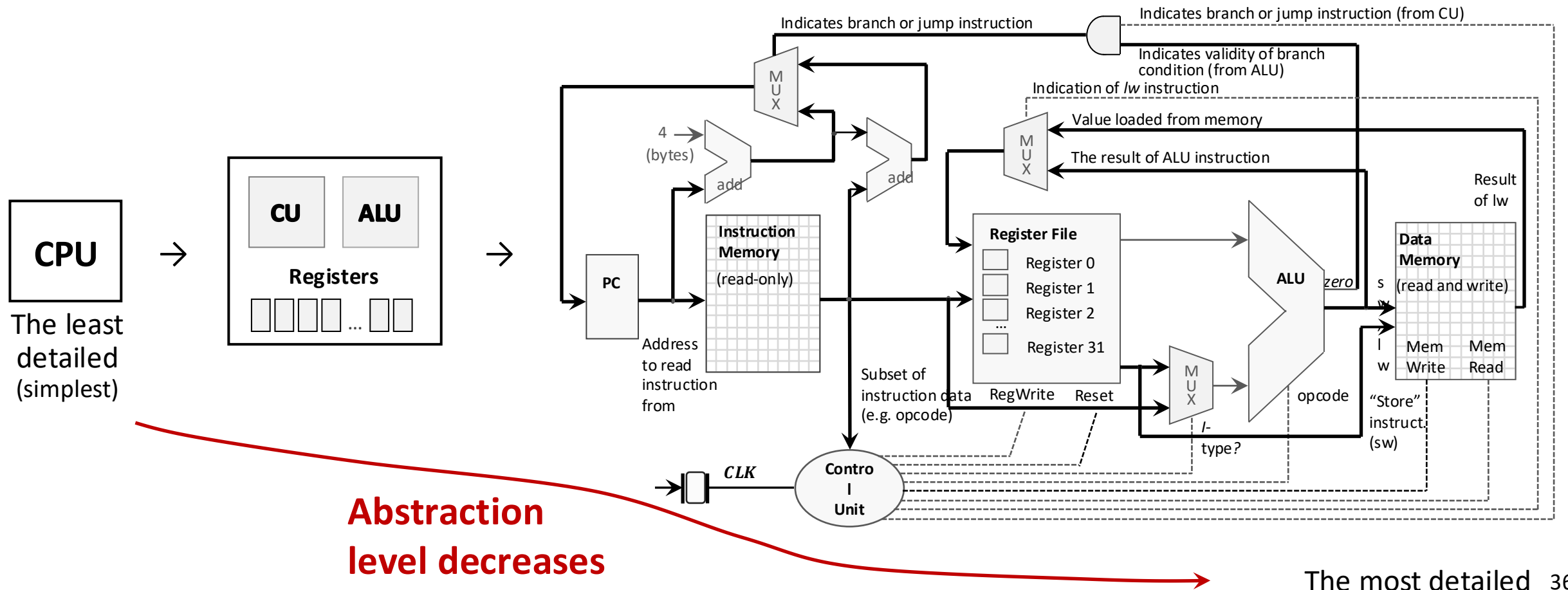
2. Design simplification via abstraction use (a CPU example)



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories

2. Design simplification via abstraction use (a CPU example)



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
- 3. Moore's law**

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. **Moore's law**

**The CPU execution speed
doubles every 18-24 months**

(due to an increasing number of
transistors on a CPU chip)

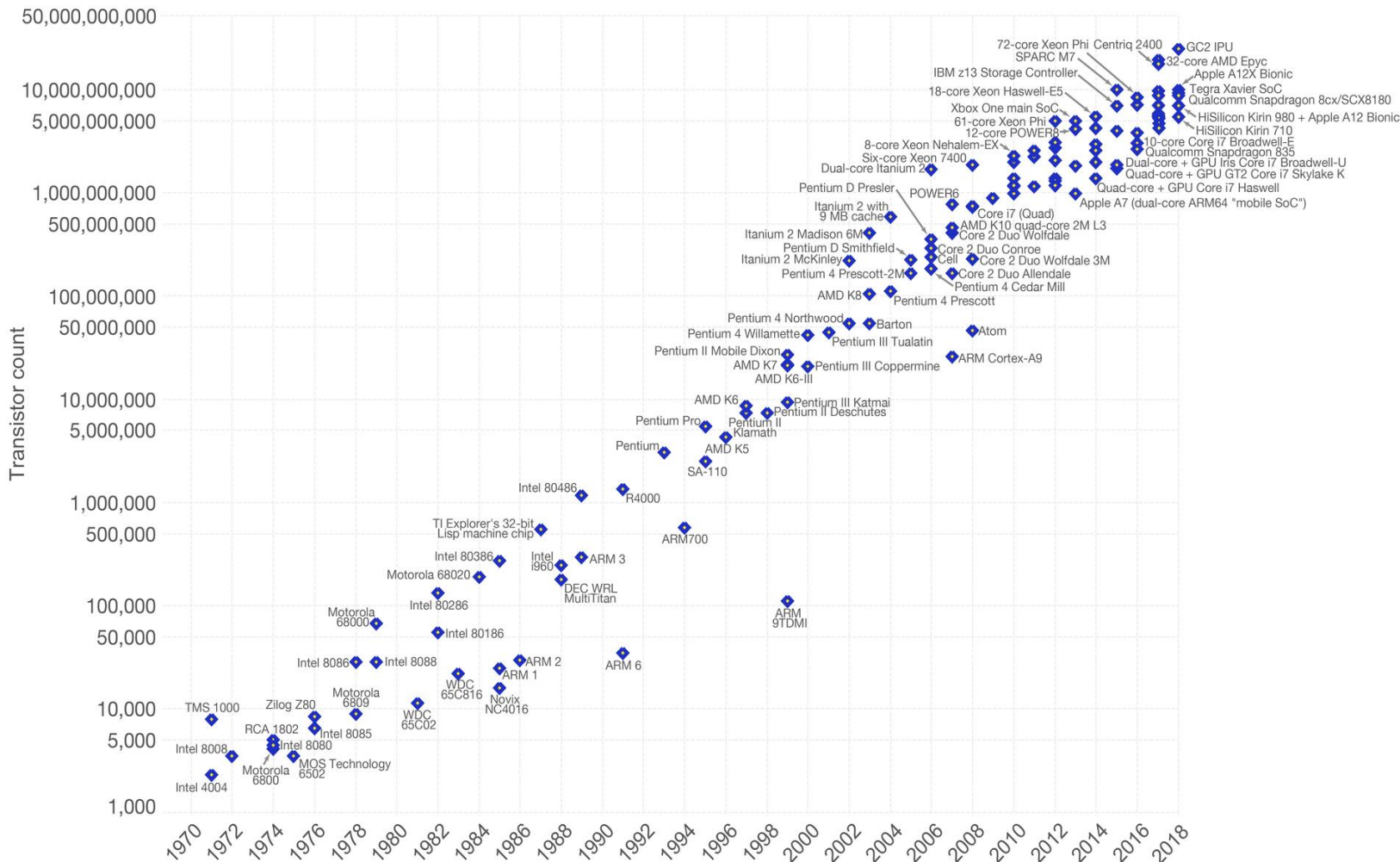
Fundamental Ideas of Computer Architecture

- 1. Hierarchy of memories
- 2. Design simplification via abstraction
- 3. Moore's law

The CPU execution speed doubles every 18-24 months (due to an increasing number of transistors on a CPU chip)

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

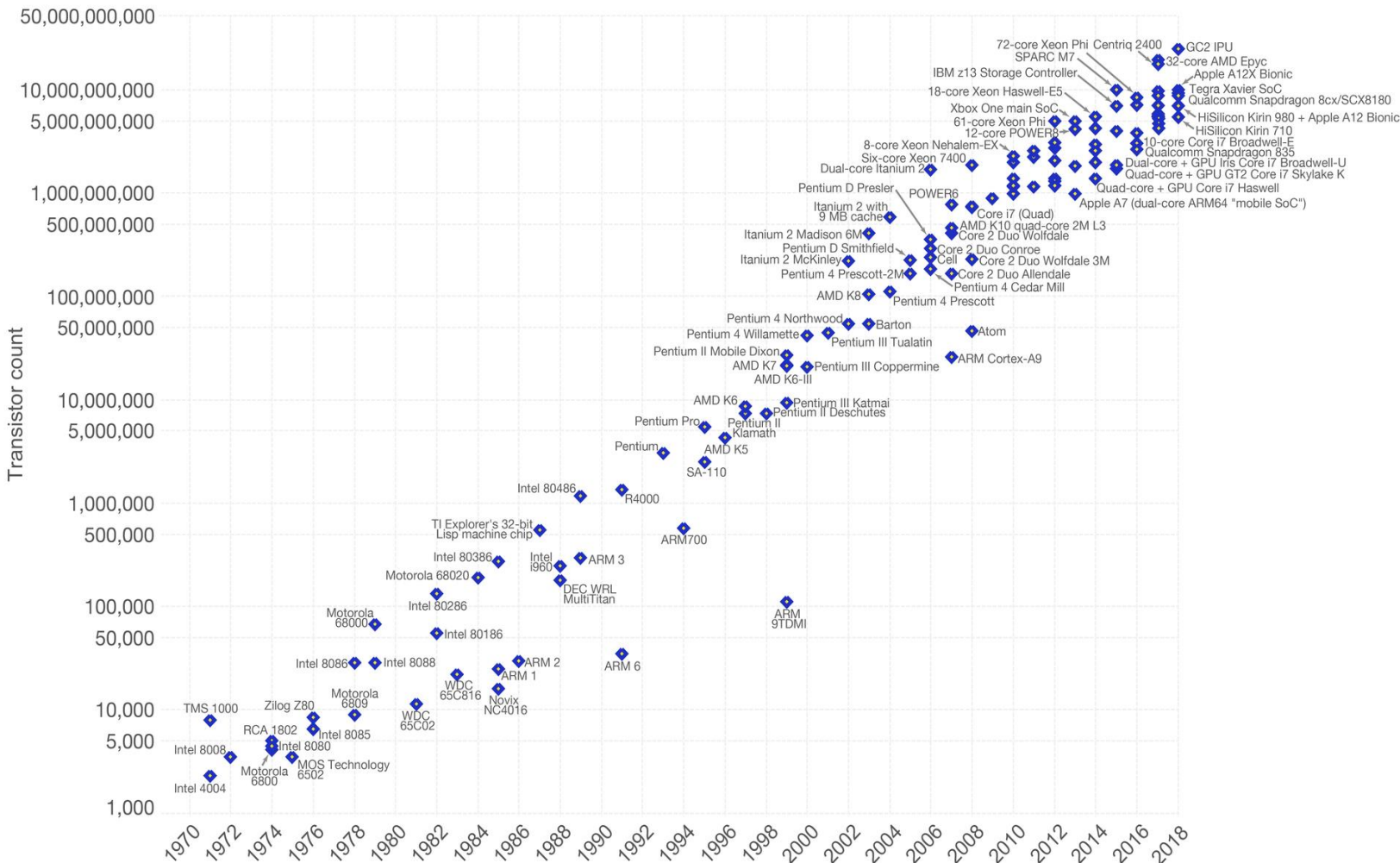


Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at [OurWorldinData.org](https://www.ourworldindata.org). There you find more visualizations and research on this topic.

1. Hierarchy of memories
2. Design simplification via abstraction
3. **Moore's law**

1. Hierarchy of memories
2. Design simplification via abstraction
3. **Moore's law**

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Licensed under [CC-BY-SA](#) by the author Max Roser.

Fundamental Ideas of Computer Architecture

- 1. Hierarchy of memories
- 2. Design simplification via abstraction
- 3. Moore's law

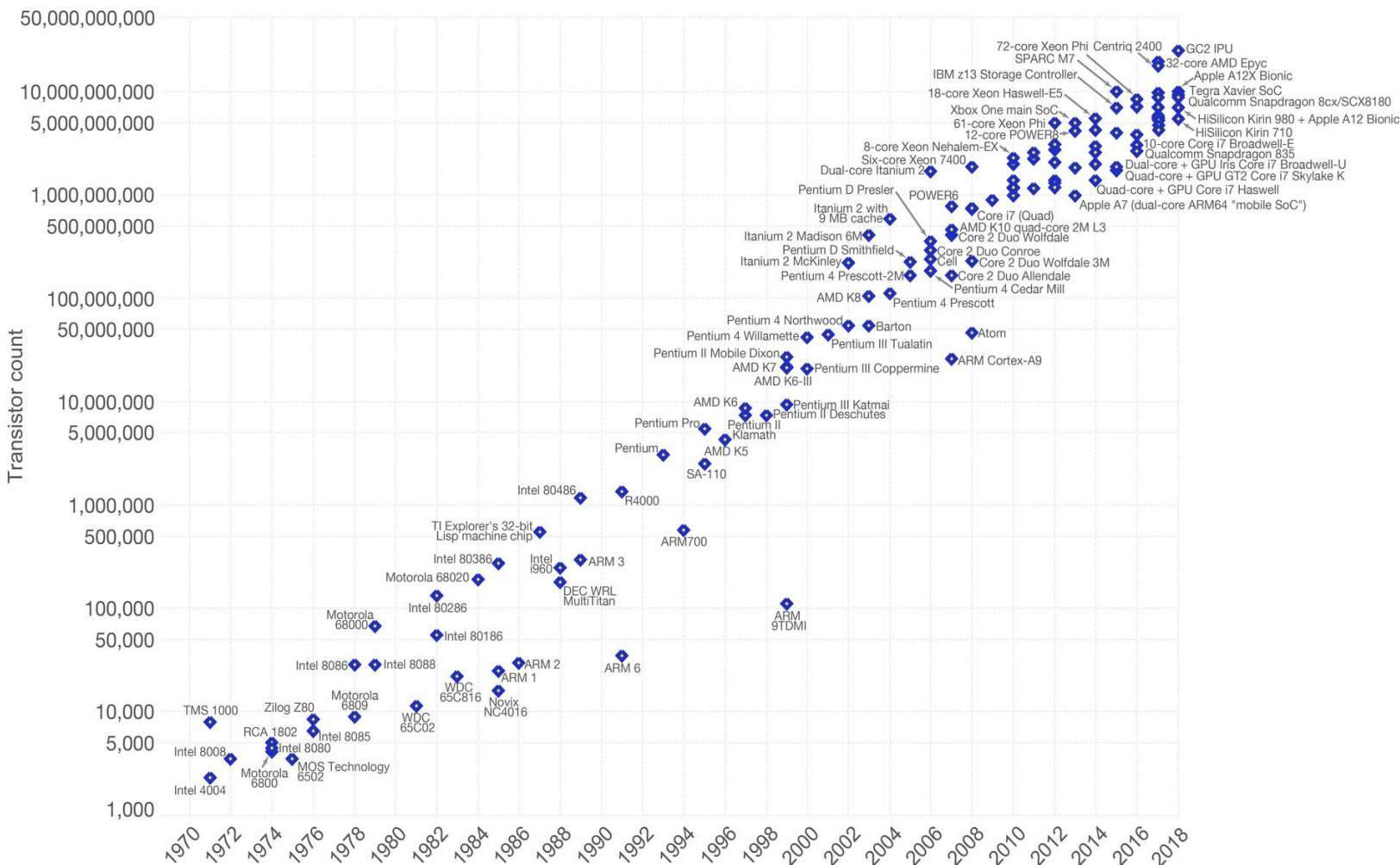
The CPU execution speed doubles every 18-24 months

This observation is useful for long term projects:

By a project completion time, CPU becomes faster

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at [OurWorldinData.org](https://www.ourworldindata.org). There you find more visualizations and research on this topic.

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. **Moore's law**

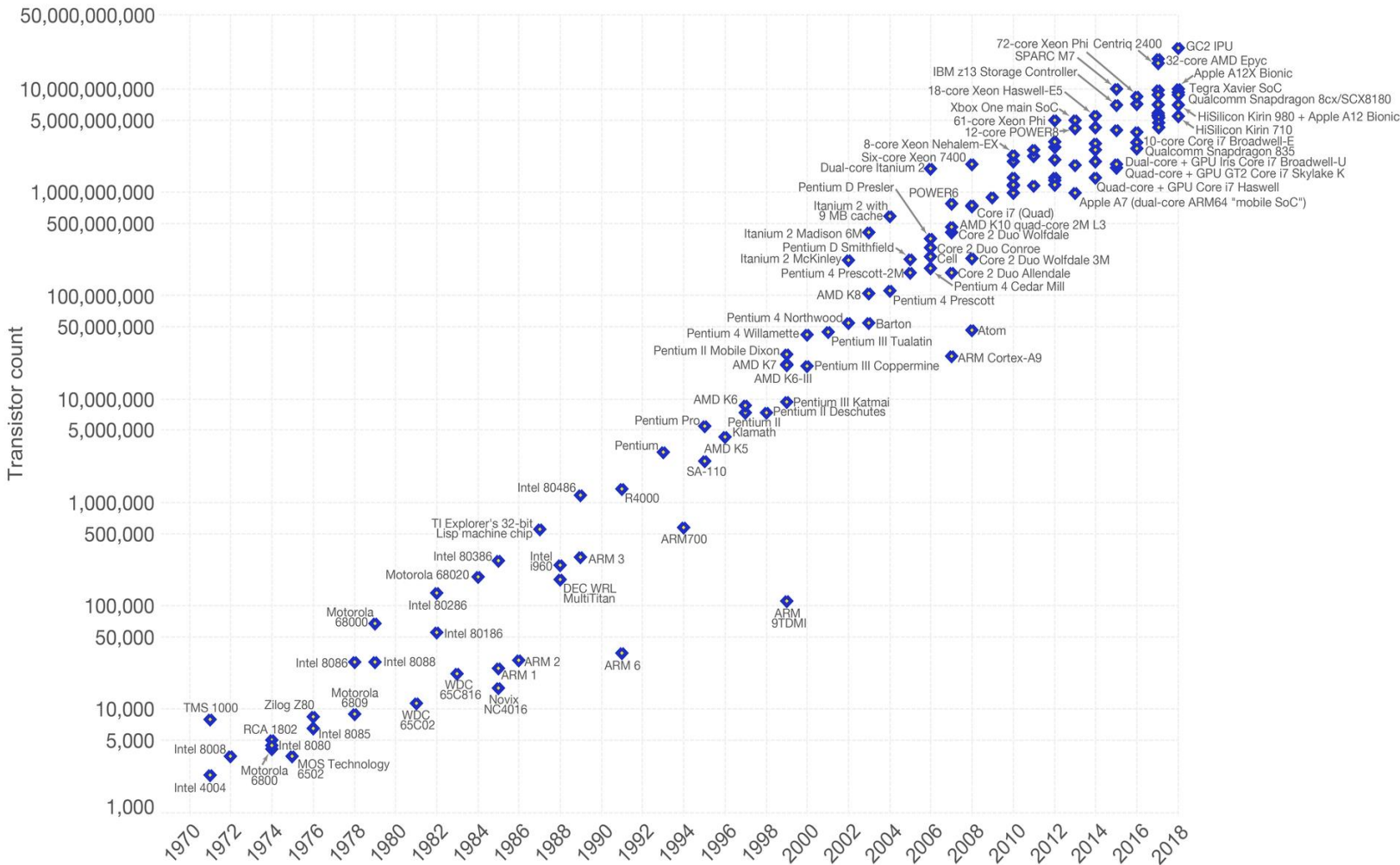
The CPU execution speed
doubles every 18-24 months

This observation is useful
for long term projects

Note:
**There is an ongoing discussion,
if Moore's law still holds**

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

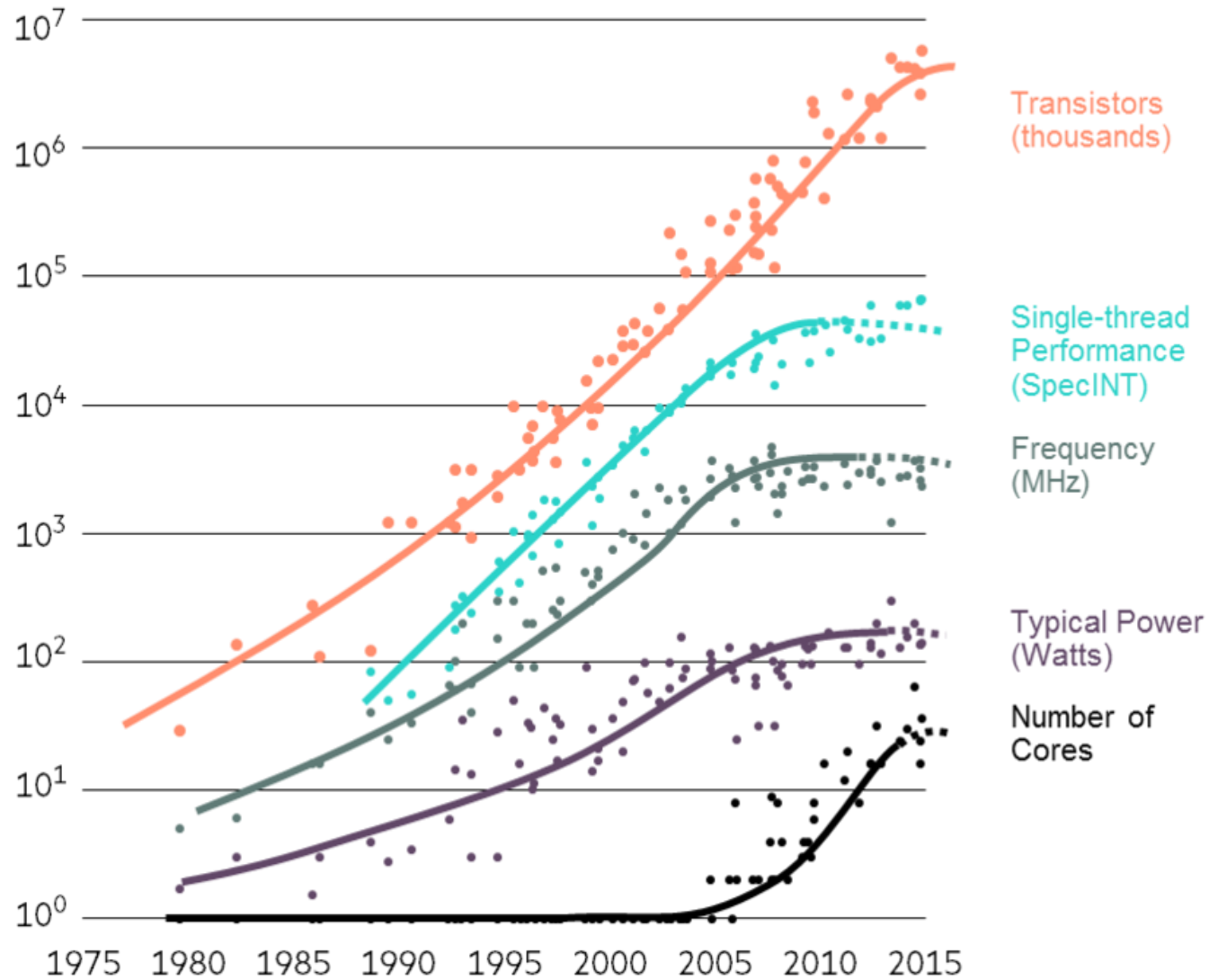


Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at [OurWorldinData.org](https://ourworldindata.org). There you find more visualizations and research on this topic.

Licensed under [CC-BY-SA](#) by the author Max Roser.

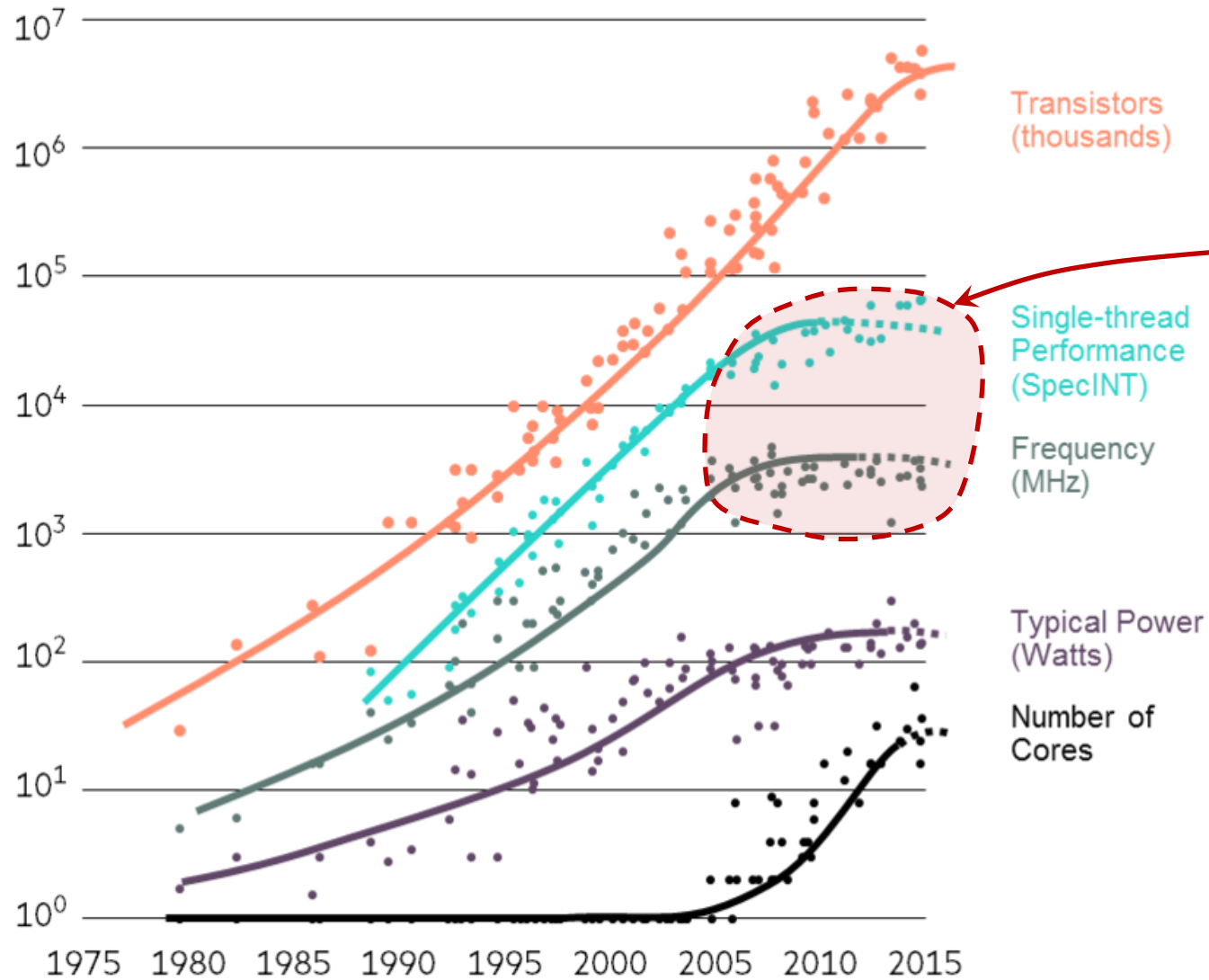
Moore's Law Stagnation

Microprocessors



Moore's Law Stagnation

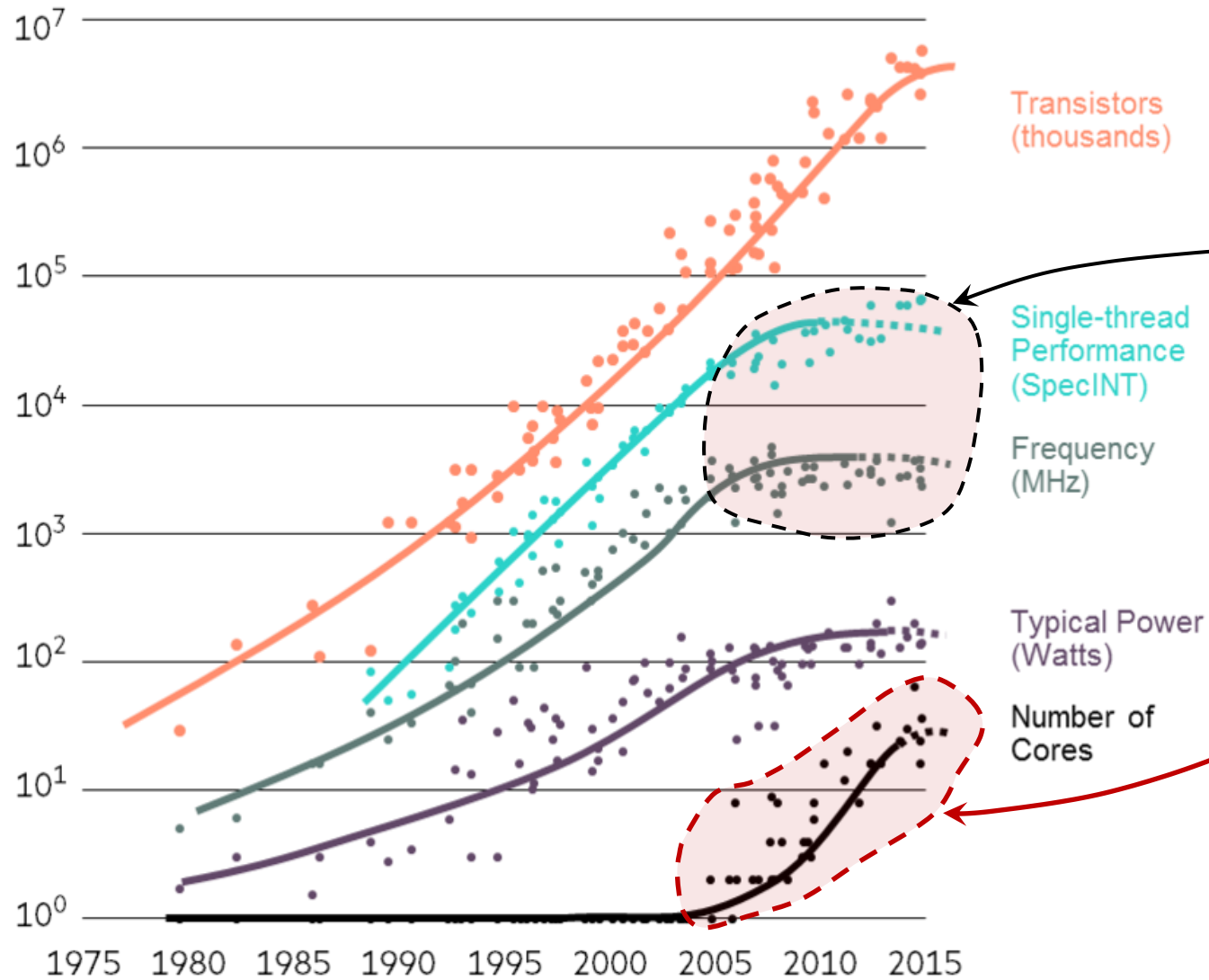
Microprocessors



CPU speed and single-thread performance seem not to increase since ~2008

Moore's Law Stagnation

Microprocessors

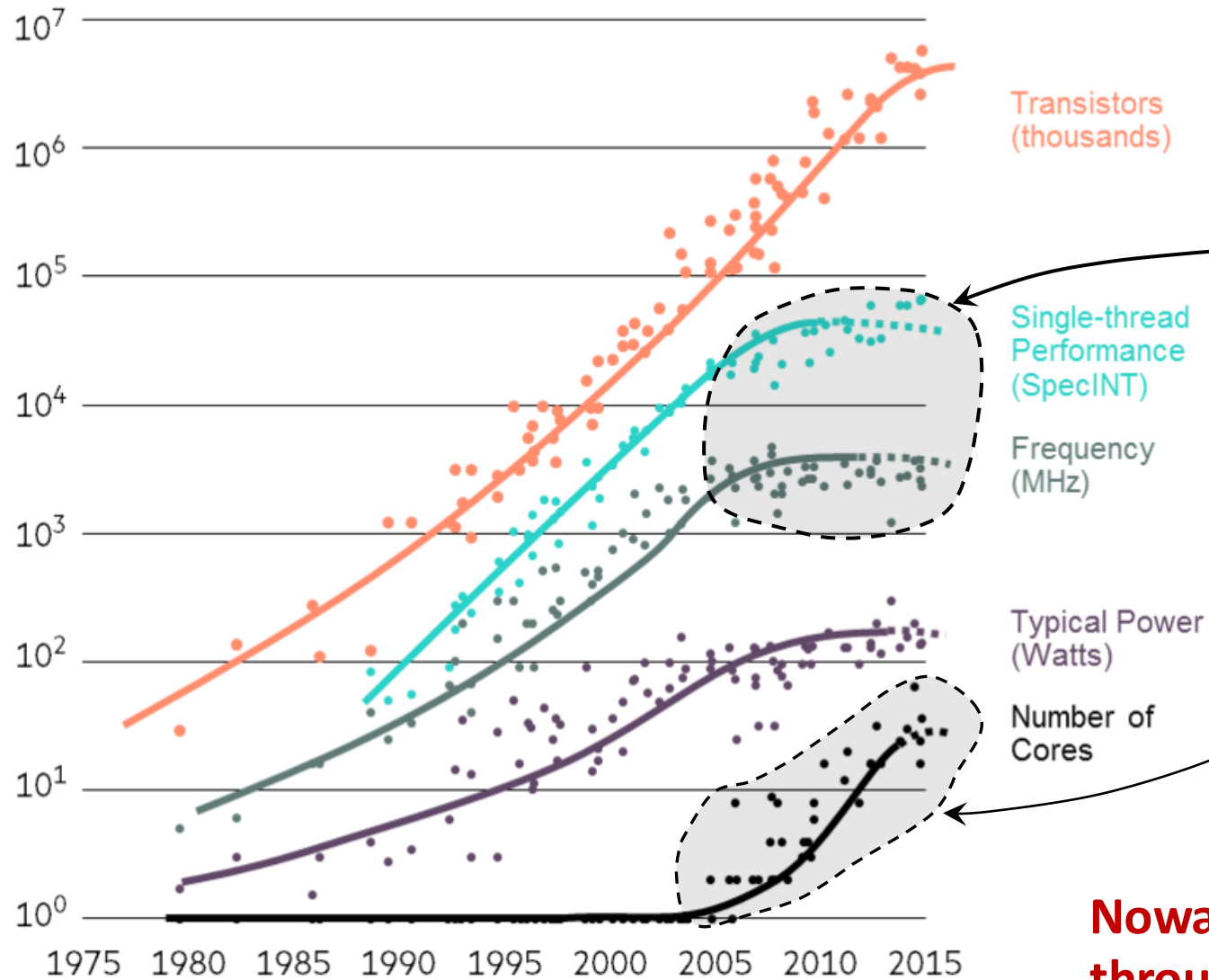


CPU speed and single-thread performance seem not to increase since ~2008

The number of cores increases instead

Moore's Law Stagnation

Microprocessors



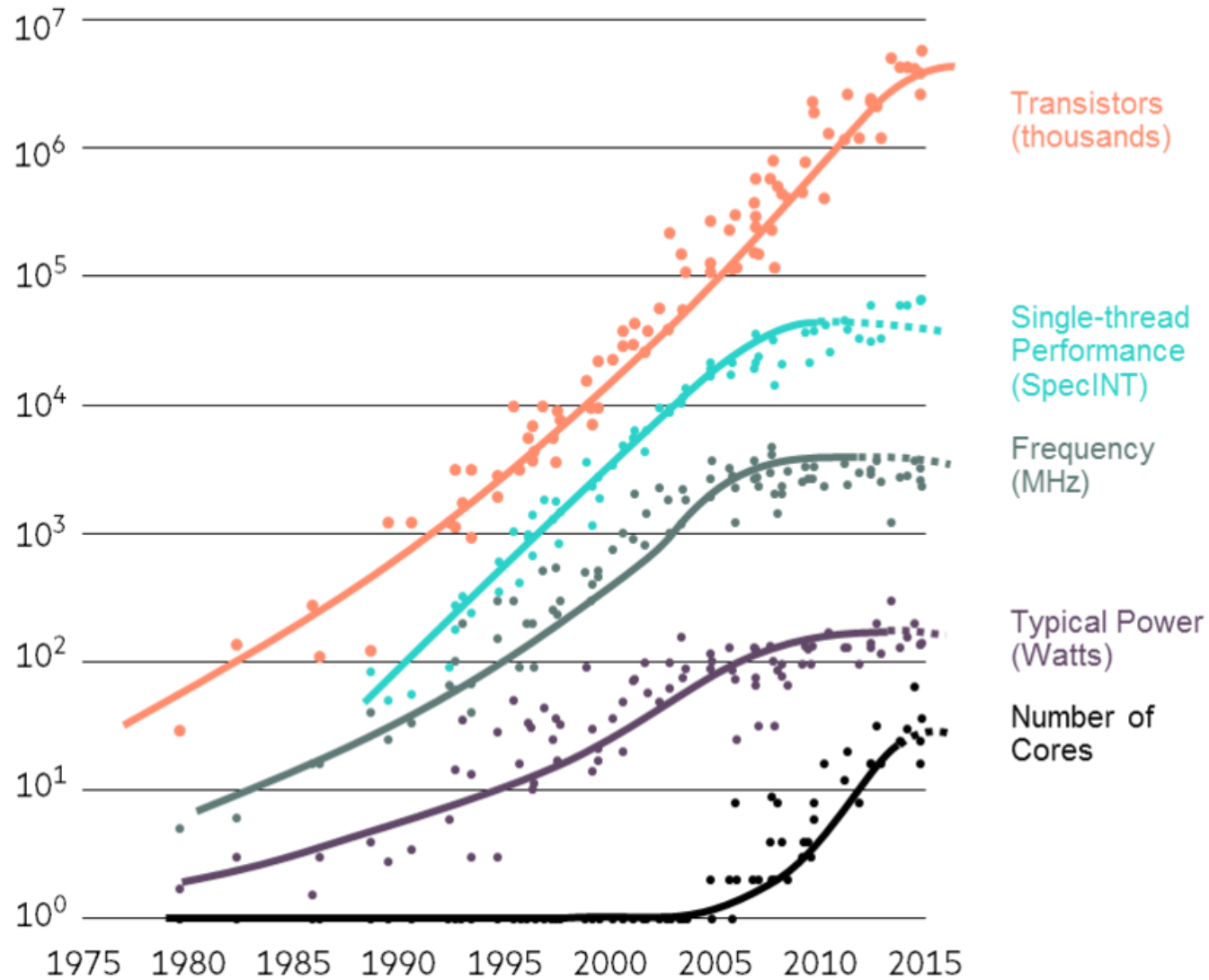
CPU speed and single-thread performance seem not to increase since ~2008

The number of cores increases instead

Nowadays, the increase of performance is achieved through the number of CPUs (or CPU cores), rather than a further increase of CPU speed

Moore's Law Stagnation

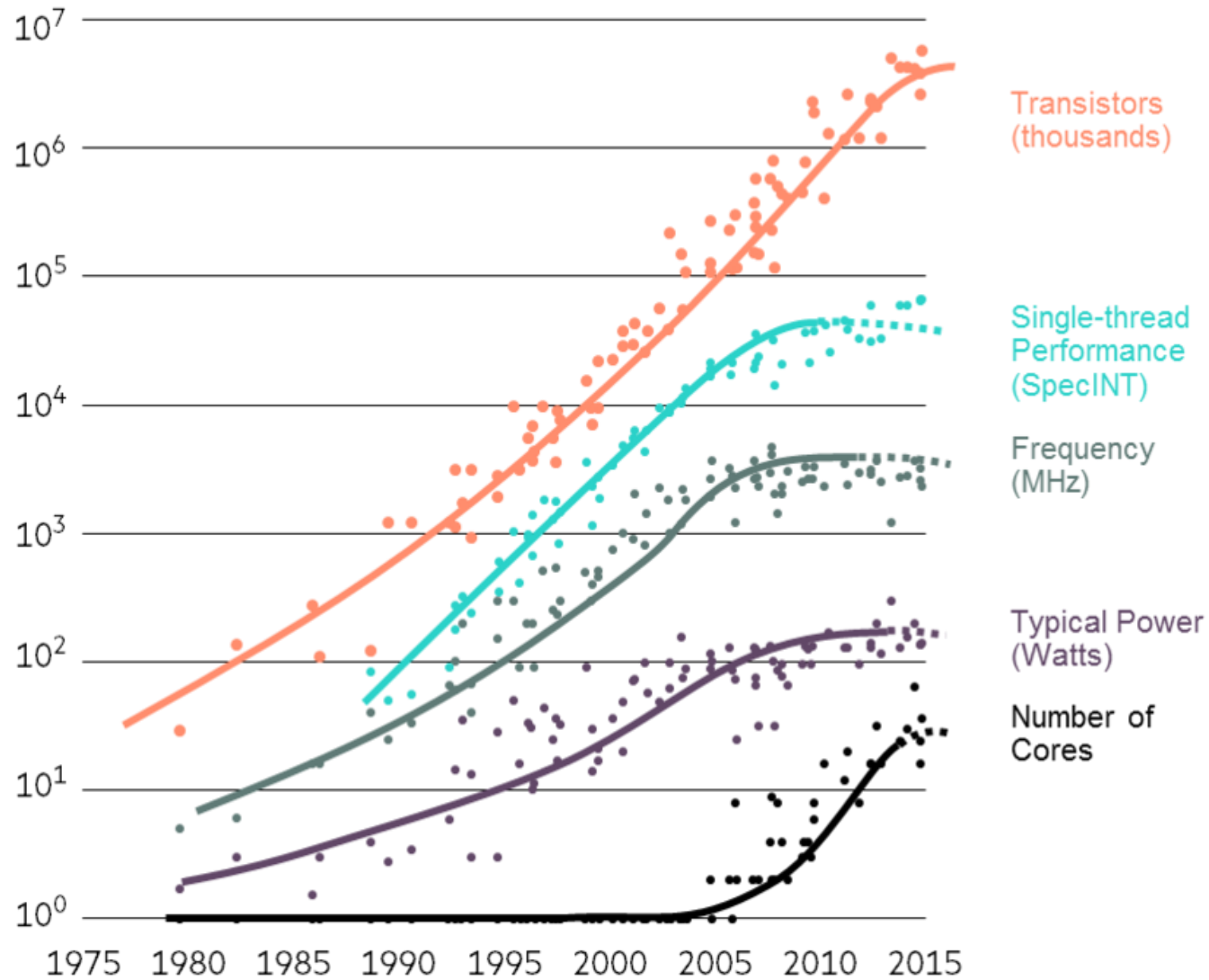
Microprocessors



Heat dissipation problem –
the key reason for the CPU speed stagnation:

Moore's Law Stagnation

Microprocessors

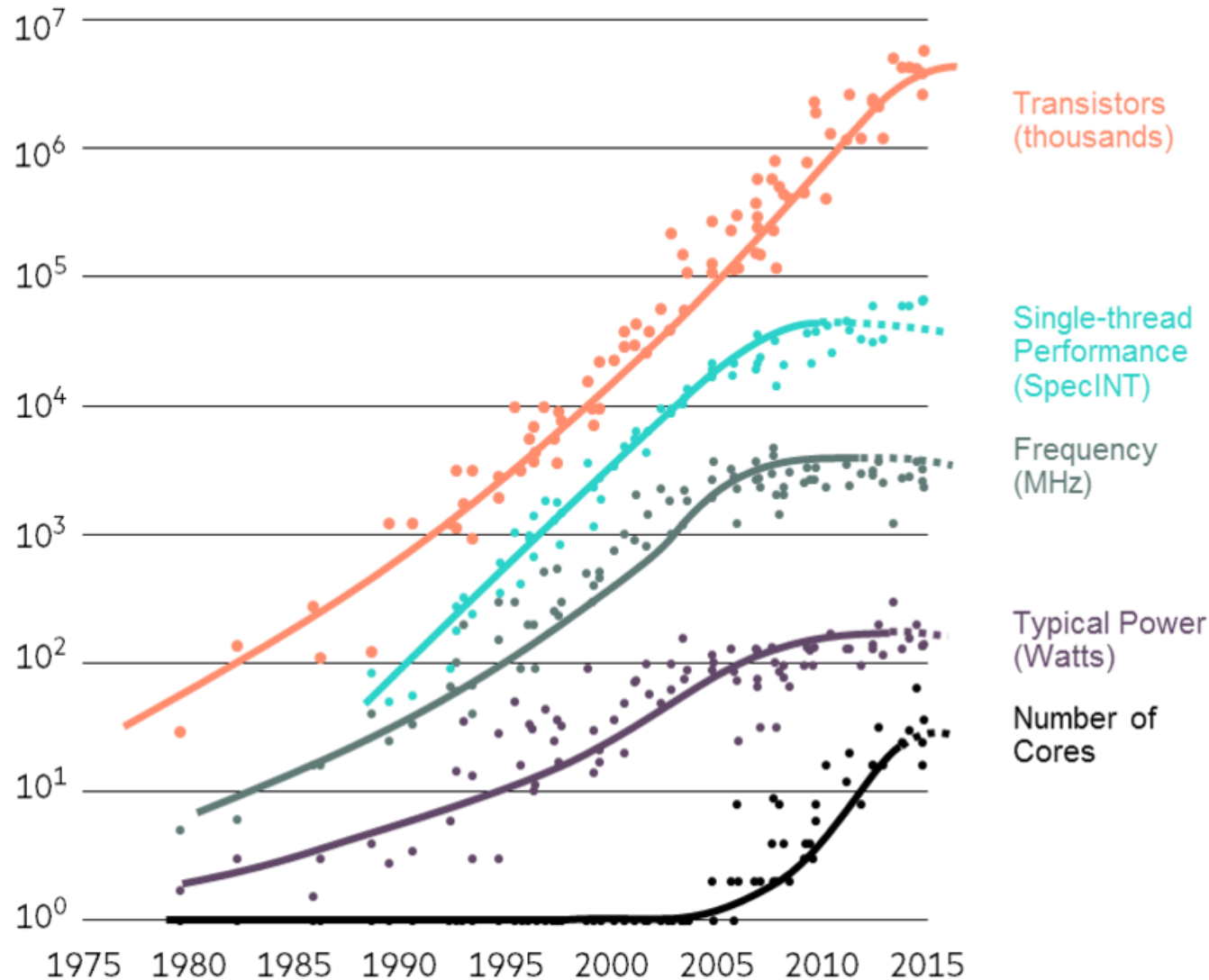


Heat dissipation problem –
the key reason for the CPU speed stagnation:

a higher speed →
a higher consumed power →
a higher power loss →
CPU overheating

Moore's Law Stagnation

Microprocessors



Heat dissipation problem –

the key reason for the CPU speed stagnation:

a higher speed →

a higher consumed power →

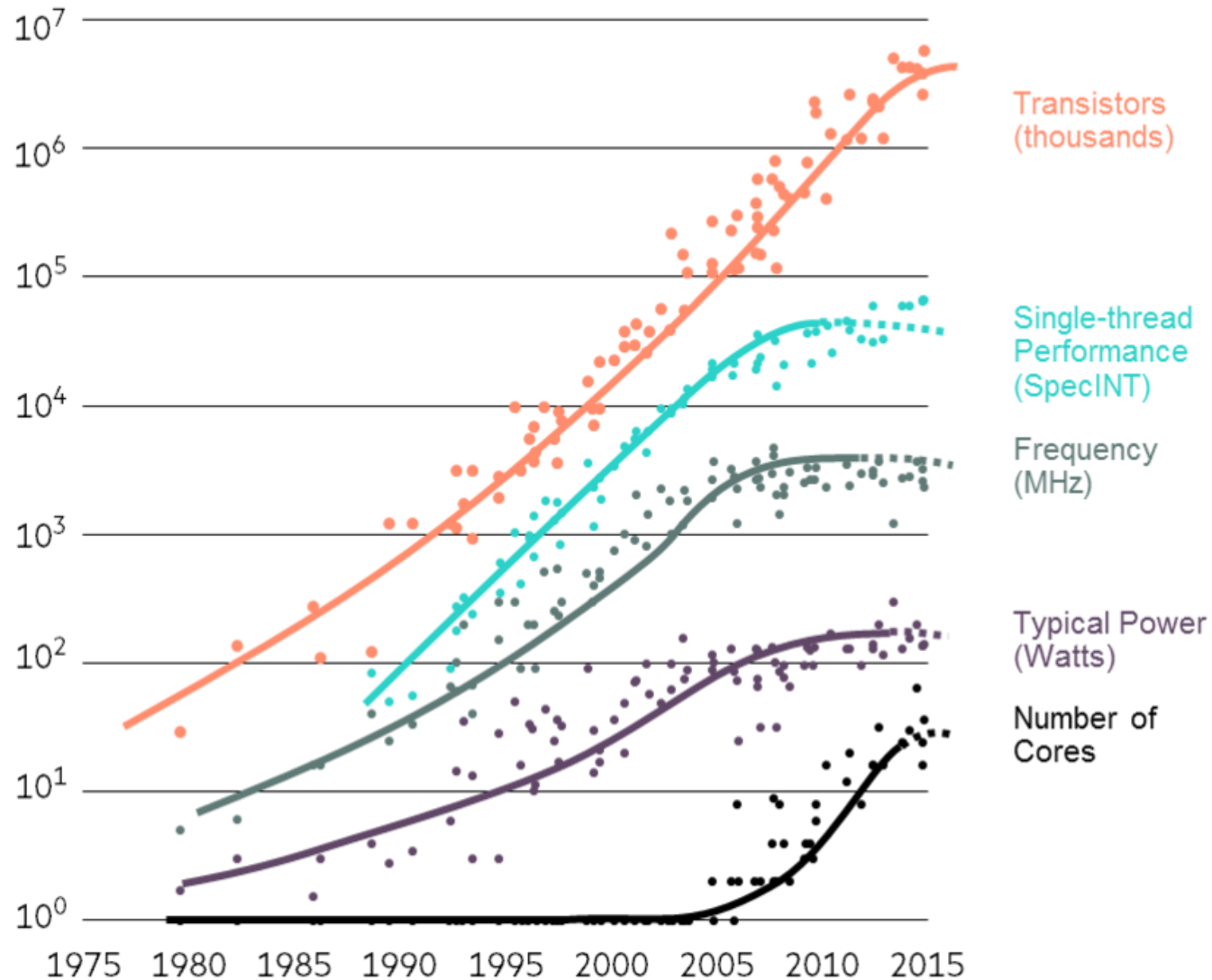
a higher power loss →

CPU overheating

**The limitation of the speed of light –
another problem**

Moore's Law Stagnation

Microprocessors



Heat dissipation problem –

the key reason for the CPU speed stagnation:

a higher speed →

a higher consumed power →

a higher power loss →

CPU overheating

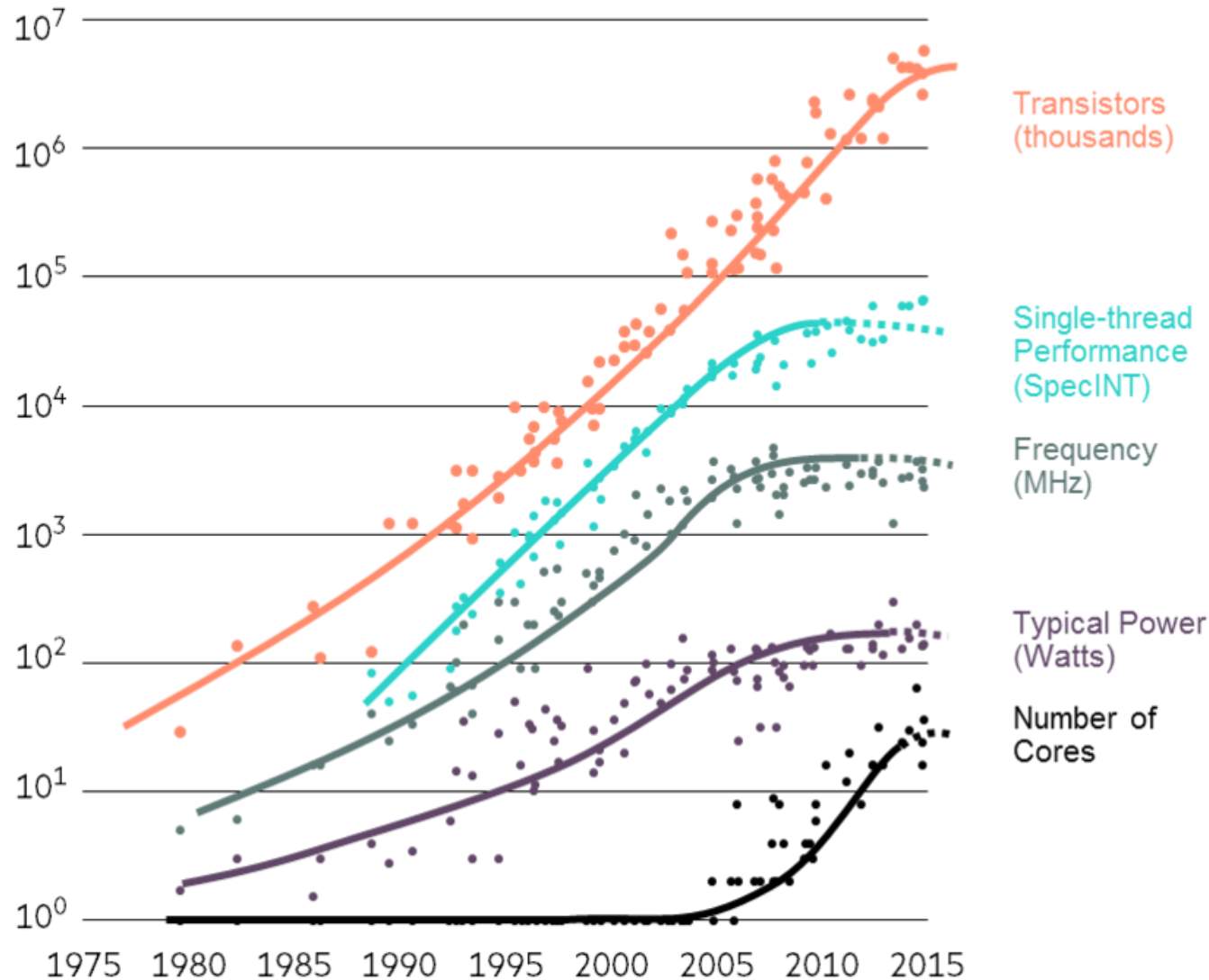
The limitation of the speed of light –

another problem:

signals within a CPU chip travel nearly at a speed of light, between input and output pins

Moore's Law Stagnation

Microprocessors



Heat dissipation problem –

the key reason for the CPU speed stagnation:

a higher speed →

a higher consumed power →

a higher power loss →

CPU overheating

The limitation of the speed of light –

another problem:

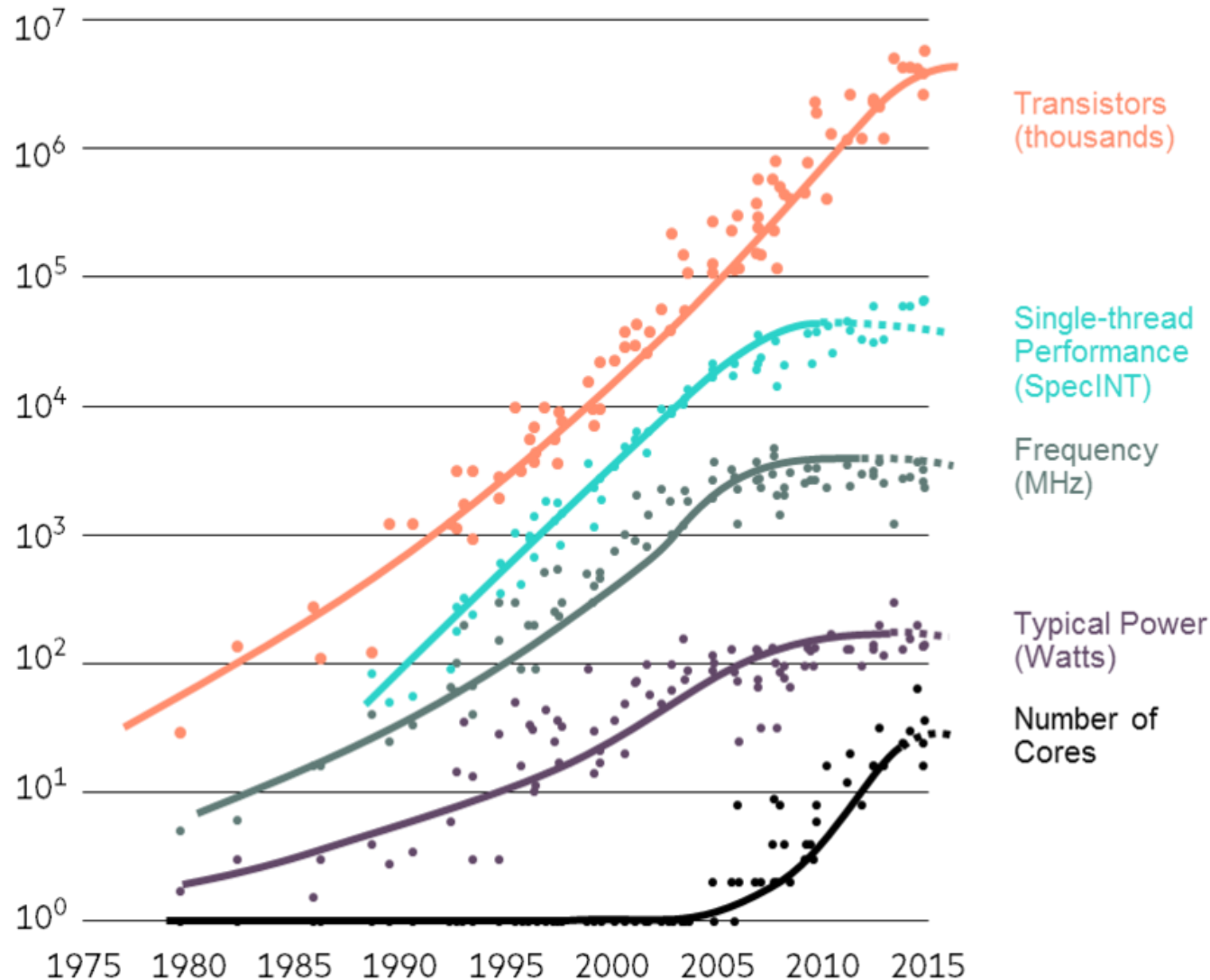
signals within a CPU chip travel nearly at

a speed of light, between input and output pins

Conclusion?

Moore's Law Stagnation

Microprocessors



Heat dissipation problem –

the key reason for the CPU speed stagnation:

a higher speed →

a higher consumed power →

a higher power loss →

CPU overheating

The limitation of the speed of light –

another problem:

signals within a CPU chip travel nearly at

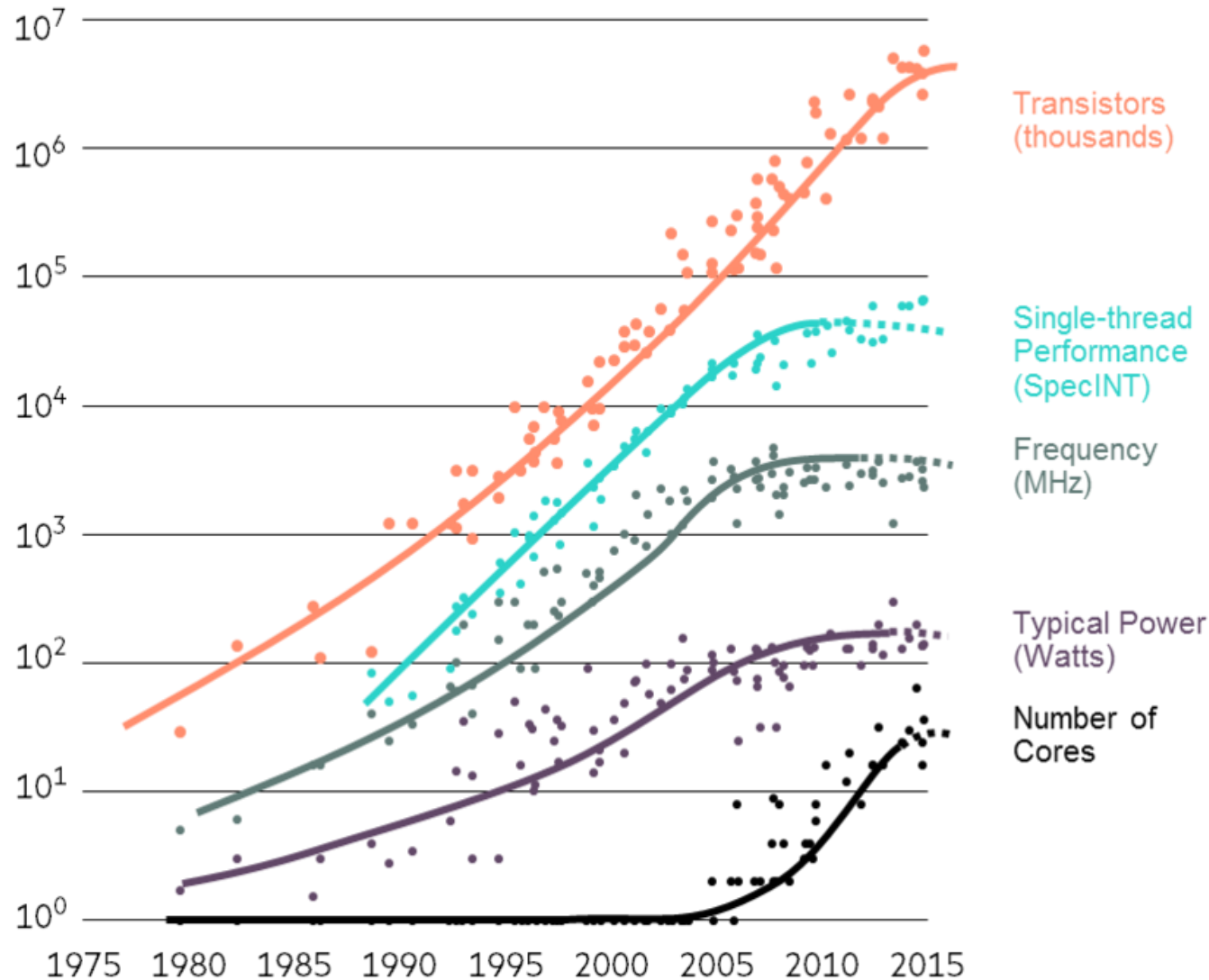
a speed of light, between input and output pins

Conclusion:

- 1) **There is a strong need for multiprocessor/multicore systems**

Moore's Law Stagnation

Microprocessors



Heat dissipation problem –

the key reason for the CPU speed stagnation:

a higher speed →

a higher consumed power →

a higher power loss →

CPU overheating

The limitation of the speed of light –

another problem:

signals within a CPU chip travel nearly at a speed of light, between input and output pins

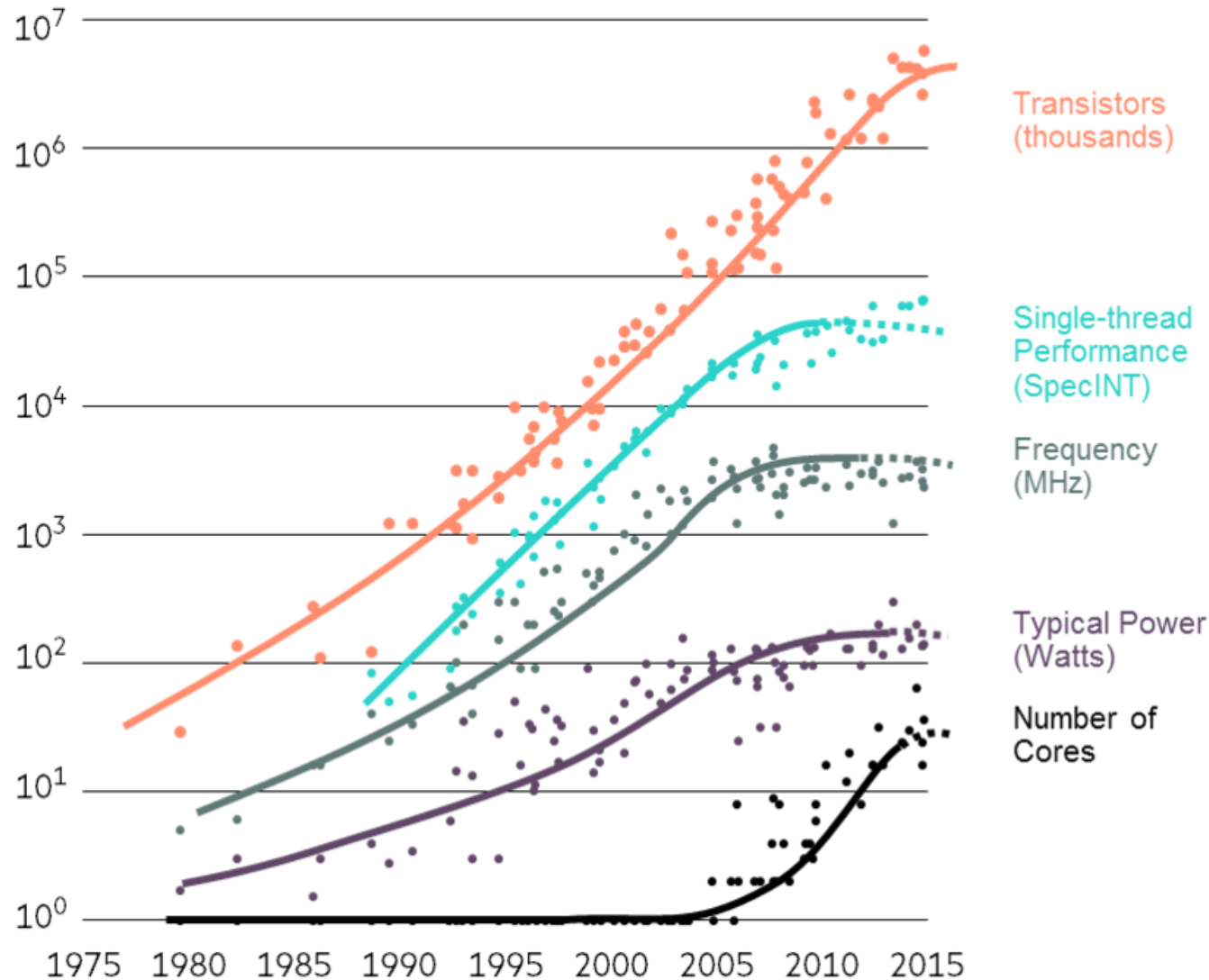
Conclusion:

- 1) **There is a strong need for multiprocessor/multicore systems**



Moore's Law Stagnation

Microprocessors



Heat dissipation problem –

the key reason for the CPU speed stagnation:

a higher speed →

a higher consumed power →

a higher power loss →

CPU overheating

The limitation of the speed of light –

another problem:

signals within a CPU chip travel nearly at

a speed of light, between input and output pins

Conclusion:

1) There is a strong need for multiprocessor/multicore systems;

2) **Solutions to various scheduling and concurrency related problems needed**

Fundamental Ideas of Computer Architecture

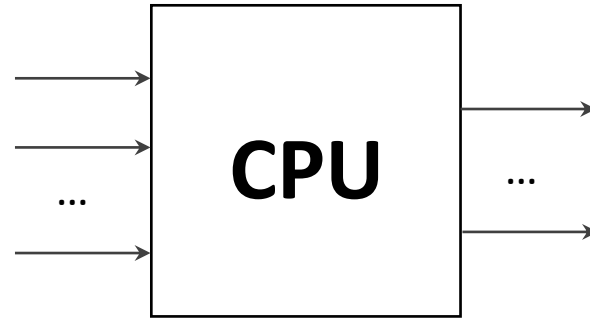
1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
- 4. Performance via parallelism**

Performance via Parallelism Idea

Computer program
(a set of instructions to be executed);
loaded into system memory

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
...

Single processor (or uniprocessor) system:



One instruction is
executed at a time
(no pipelining is assumed)

Assumptions for uniprocessor case:

- **Instructions are executed sequentially;**
- **CPU executes one instruction at a time (not always the case)**

Performance via Parallelism Idea

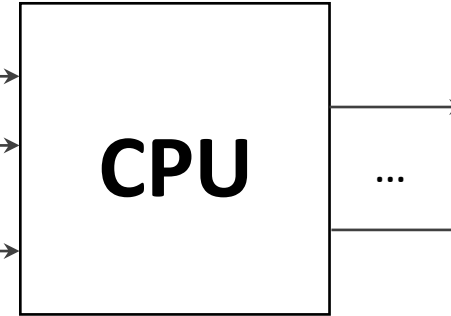
Computer program

(a set of instructions to be executed);
loaded into system memory

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
...

**Step 1: Instruction
loaded into CPU**

Single processor (or uniprocessor) system:



One instruction is
executed at a time
(no pipelining is assumed)

Assumptions for uniprocessor case:

- Instructions are executed sequentially;
- CPU executes one instruction at a time (not always the case)

Performance via Parallelism Idea

Computer program

(a set of instructions to be executed);
loaded into system memory

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
...

Step 1: Instruction
loaded into CPU

Single processor (or uniprocessor) system:

Step 2: Instruction
is executed by CPU

CPU

One instruction is
executed at a time
(no pipelining is assumed)

Assumptions for uniprocessor case:

- Instructions are executed sequentially;
- CPU executes one instruction at a time (not always the case)

Performance via Parallelism Idea

Computer program

(a set of instructions to be executed);
loaded into system memory

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
...

**Step 1: Instruction
loaded into CPU**

Single processor (or uniprocessor) system:

**Step 2: Instruction
is executed by CPU**

CPU

One instruction is
executed at a time
(no pipelining is assumed)

**Step 3: Result is
stored into memory**

Assumptions for uniprocessor case:

- Instructions are executed sequentially;
- CPU executes one instruction at a time (not always the case)

Performance via Parallelism Idea

Computer program

(a set of instructions to be executed);
loaded into system memory

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
...

Step 1: Instruction
loaded into CPU

Single processor (or uniprocessor) system:

Step 2: Instruction
is executed by CPU

CPU

One instruction is
executed at a time
(no pipelining is assumed)

Step 3: Result is
stored into memory

Each of these steps takes time!

Assumptions for uniprocessor case:

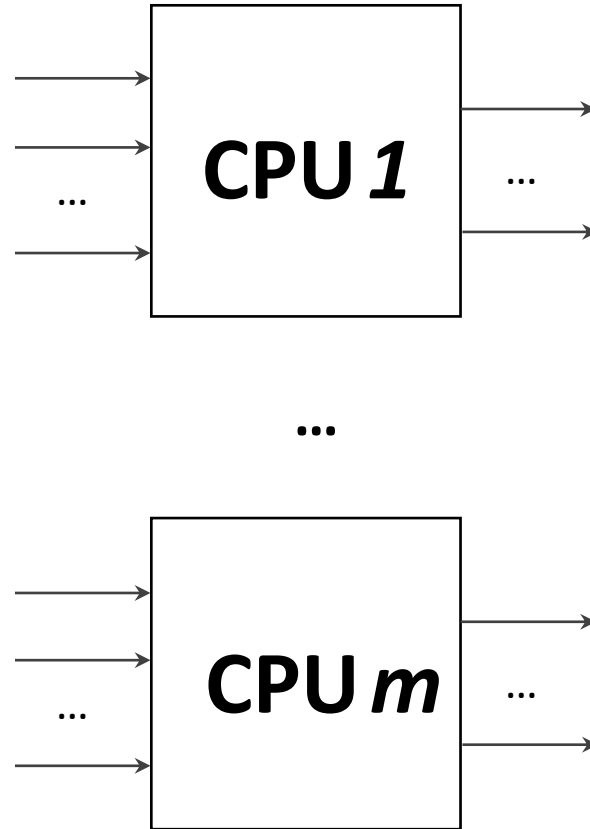
- Instructions are executed sequentially;
- CPU executes one instruction at a time (not always the case)

Performance via Parallelism Idea

Computer program
(a set of instructions to be executed);
loaded into system memory

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
...

Multiprocessor (or multicore) system:



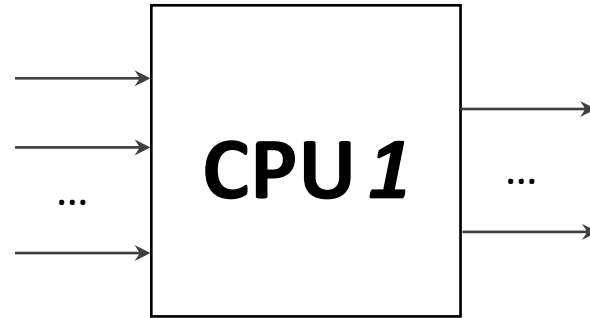
***M* instructions can be
executed simultaneously**

Performance via Parallelism Idea

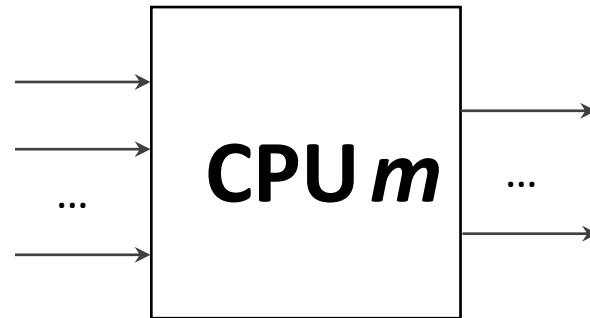
Computer program
(a set of instructions to be executed);
loaded into system memory

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
...

Multiprocessor (or multicore) system:



...



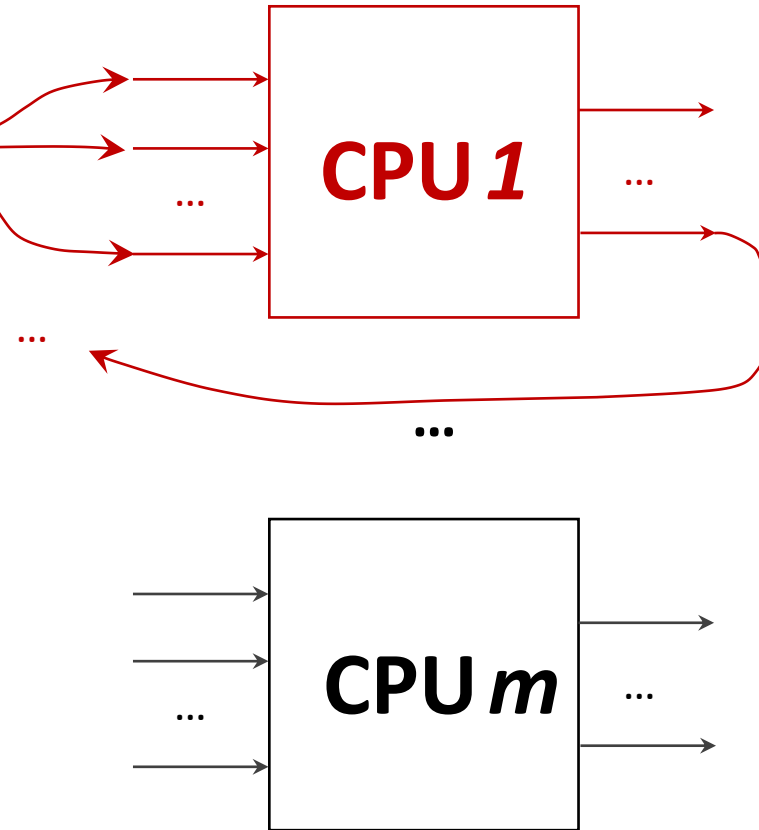
**M instructions can be
executed simultaneously**

Performance via Parallelism Idea

Computer program
(a set of instructions to be executed);
loaded into system memory

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
...

Multiprocessor (or multicore) system:



***M* instructions can be
executed simultaneously**

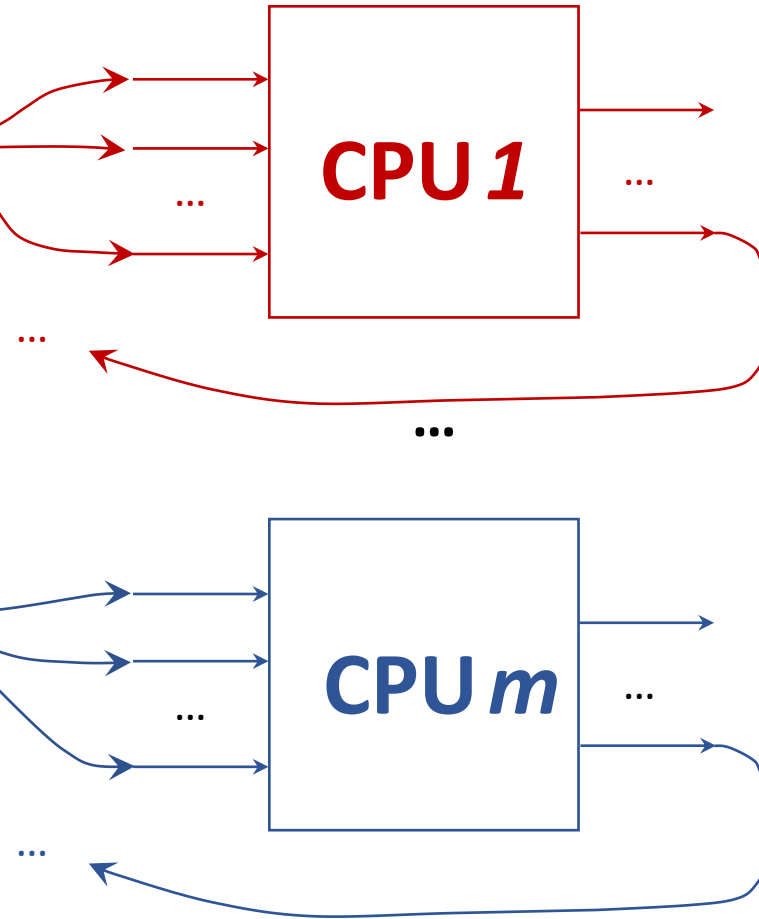
Performance via Parallelism Idea

Computer program
(a set of instructions to be executed);
loaded into system memory

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
...

Simultaneous
execution
(restrictions apply)

Multiprocessor (or multicore) system:

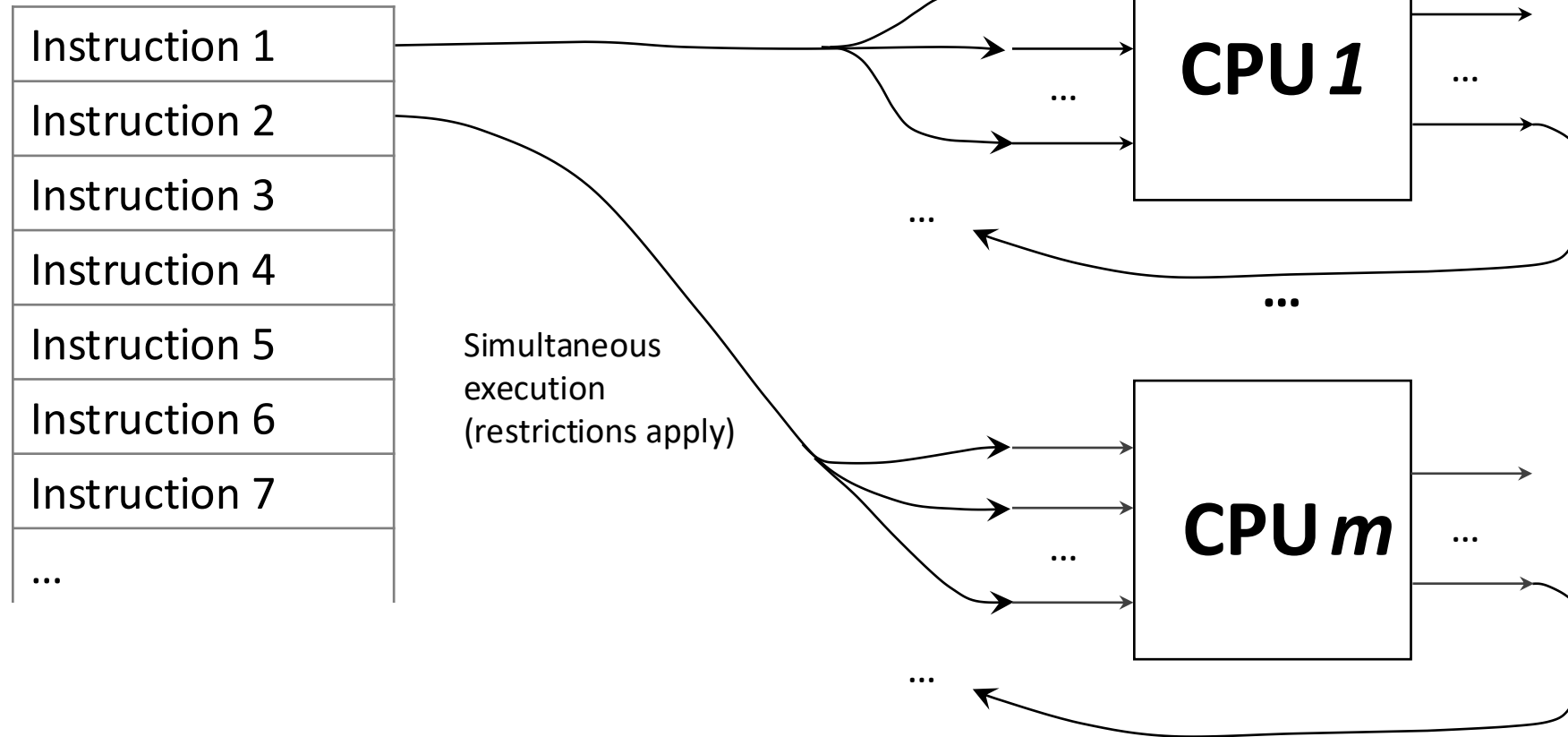


**M instructions can be
executed simultaneously**

**Multiprocessing aims at
speeding-up program
execution, or simultaneous
execution of multiple
programs**

Performance via Parallelism Idea

Computer program
(a set of instructions to be executed);
loaded into system memory



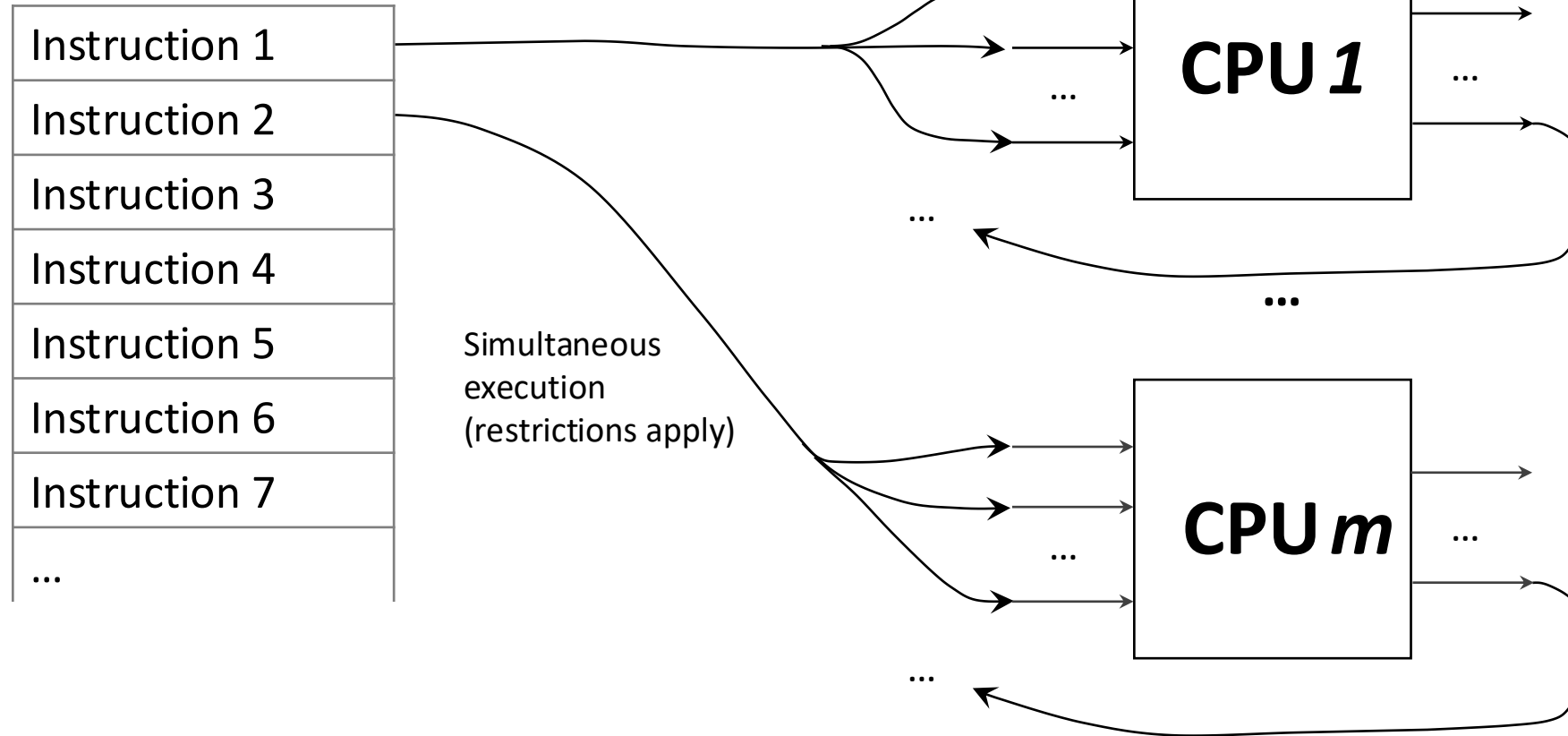
M instructions can be
executed simultaneously

Multiprocessing aims at
speeding-up program
execution, or simultaneous
execution of multiple
programs

The major problem:
Some instructions depend on
the execution result of
previous instructions, and thus,
have to wait for their
completion

Performance via Parallelism Idea

Computer program
(a set of instructions to be executed);
loaded into system memory



Multiprocessor (or multicore) system:

M instructions can be
executed simultaneously

Multiprocessing aims at
speeding-up program
execution, or simultaneous
execution of multiple
programs

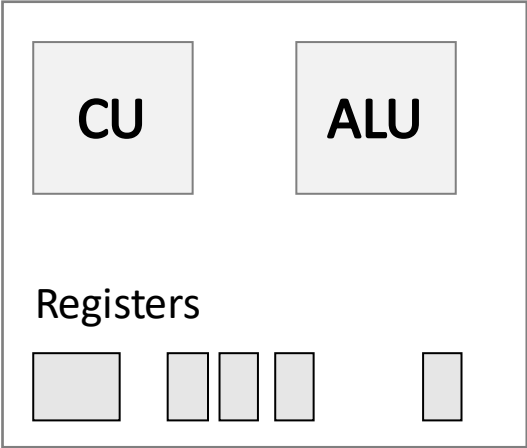
Many problems remain open;
These problems got a lot of
attention after Moore's law
"stagnation"

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
- 5. Performance via pipelining**

Pipelined Execution of CPU Instructions

Our custom CPU

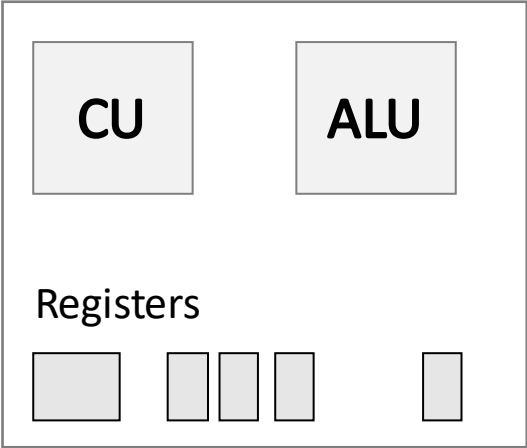


Instruction Set,
assumed to be supported by our CPU:

Operation	Operation Code
Logical “OR”	0
Logical “AND”	1

Pipelined Execution of CPU Instructions

Our custom CPU



Instruction Set,
assumed to be supported by our CPU:

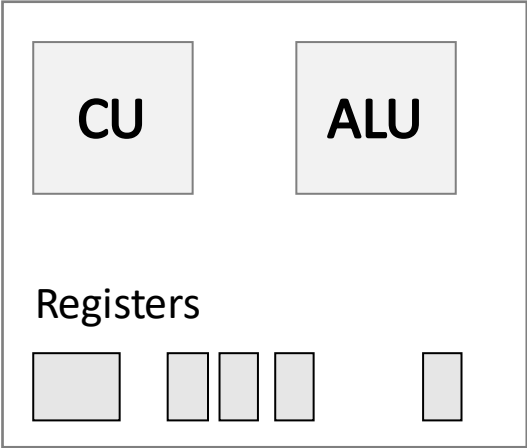
Operation	Operation Code
Logical “OR”	0
Logical “AND”	1

Instruction Format:

Operation Code	Argument 1	Argument 2
----------------	------------	------------

Pipelined Execution of CPU Instructions

Our custom CPU



Instruction Set,
assumed to be supported by our CPU:

Operation	Operation Code
Logical “OR”	0
Logical “AND”	1

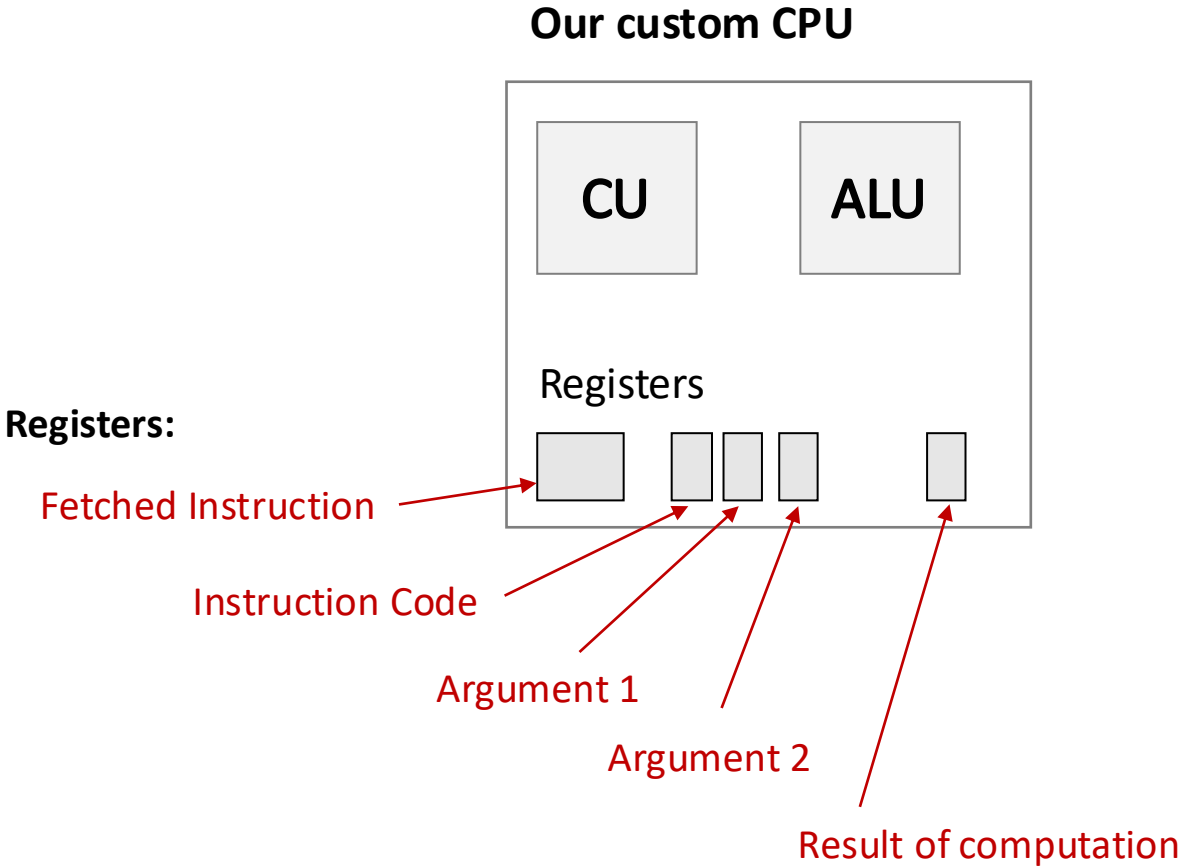
Instruction Format:

Operation Code	Argument 1	Argument 2
----------------	------------	------------

Sample Instruction:

0	1	0
---	---	---

Pipelined Execution of CPU Instructions



Instruction Set,
assumed to be supported by our CPU:

Operation	Operation Code
Logical “OR”	0
Logical “AND”	1

Instruction Format:

Operation Code	Argument 1	Argument 2
----------------	------------	------------

Sample Instruction:

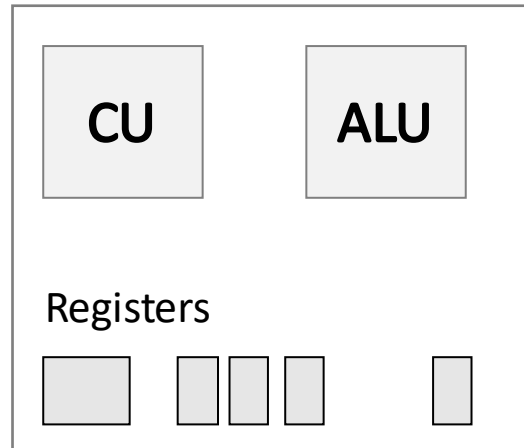
0	1	0
---	---	---

Pipelined Execution of CPU Instructions

Instructions in System Memory

Instruction 1: 010
Instruction 2
Instruction 3
Instruction 4
Instruction 5
...

Our custom CPU



Pipelined Execution of CPU Instructions

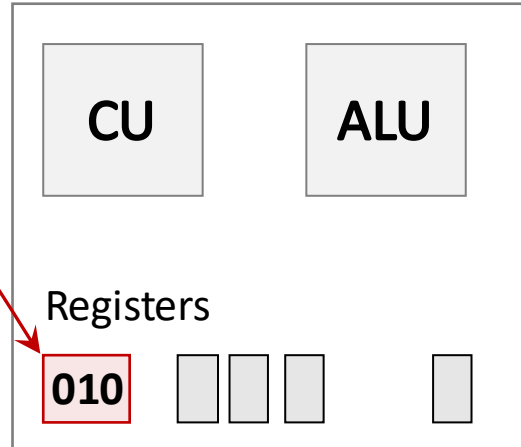
Each instruction is executed in several steps:

Instructions in System Memory

Instruction 1: 010
Instruction 2
Instruction 3
Instruction 4
Instruction 5
...

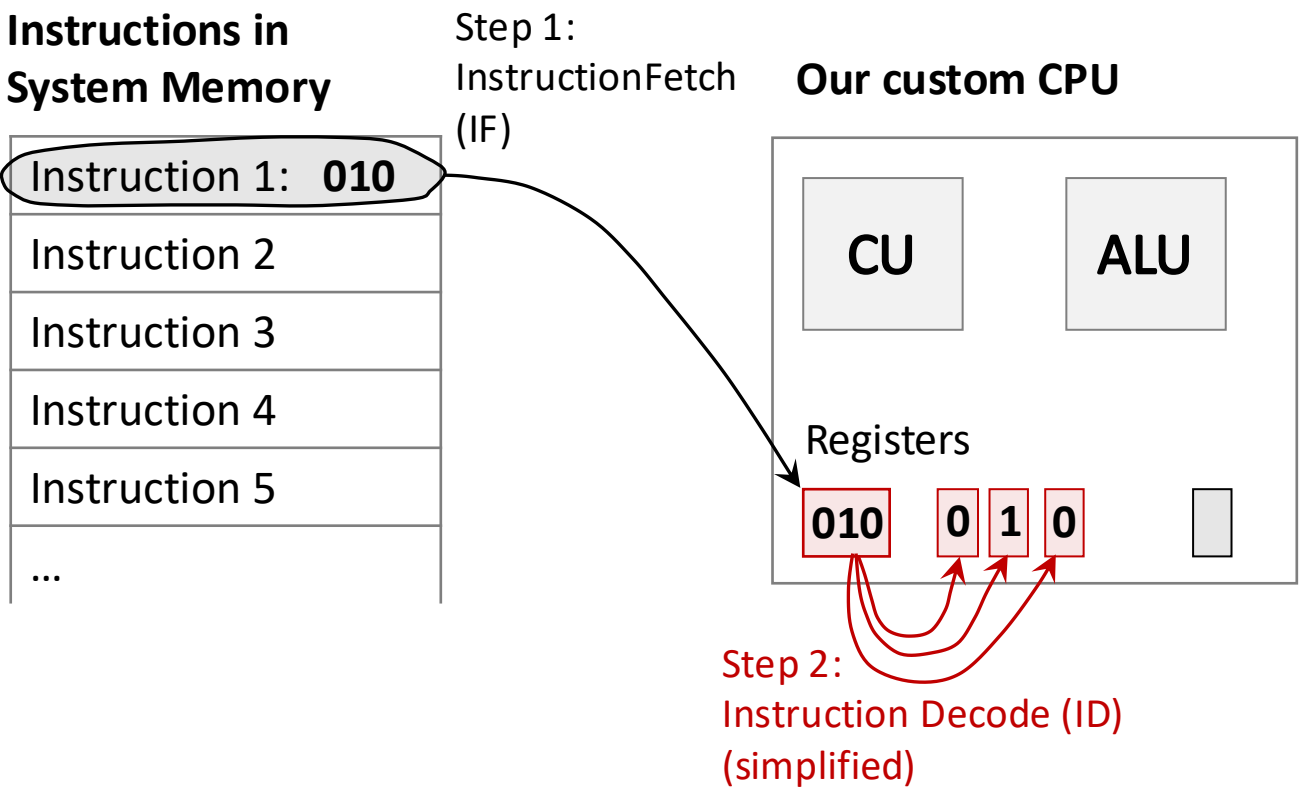
Step 1:
Instruction Fetch
(IF)

Our custom CPU



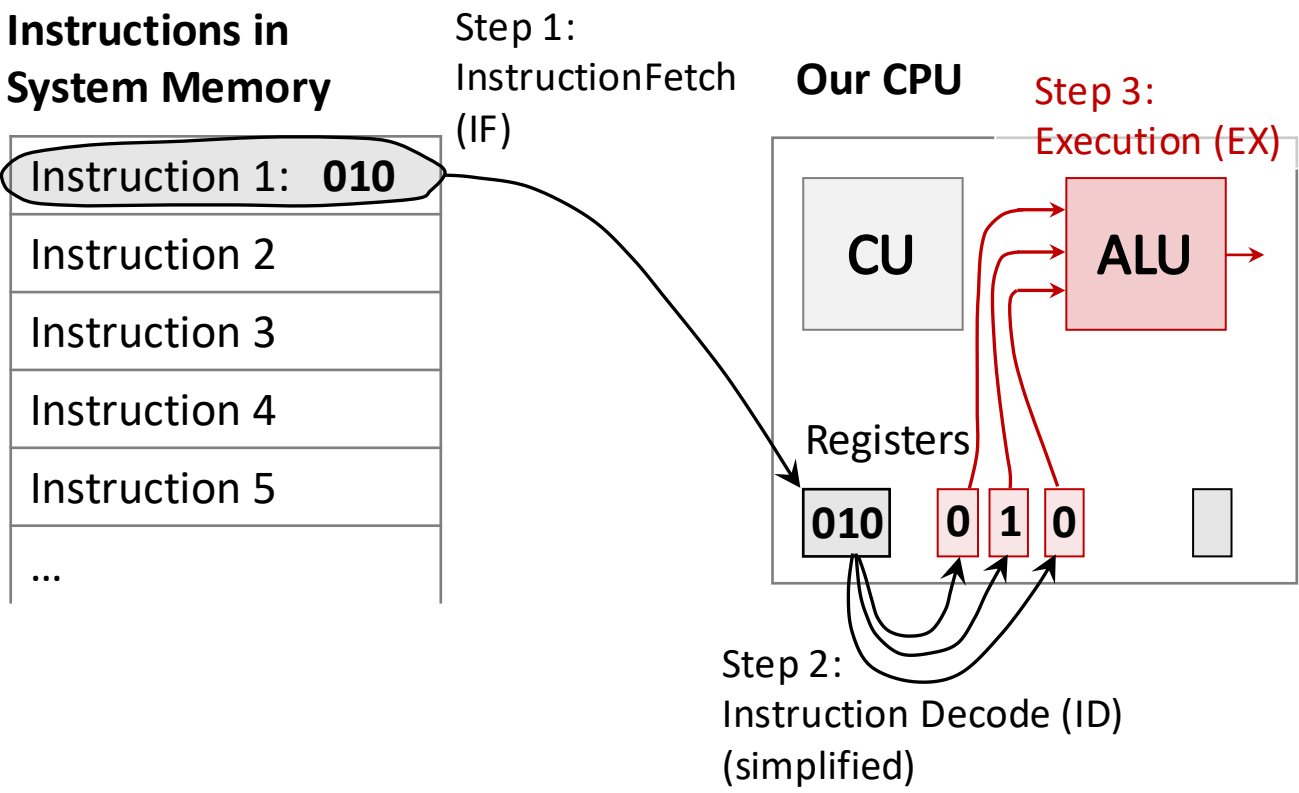
Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:



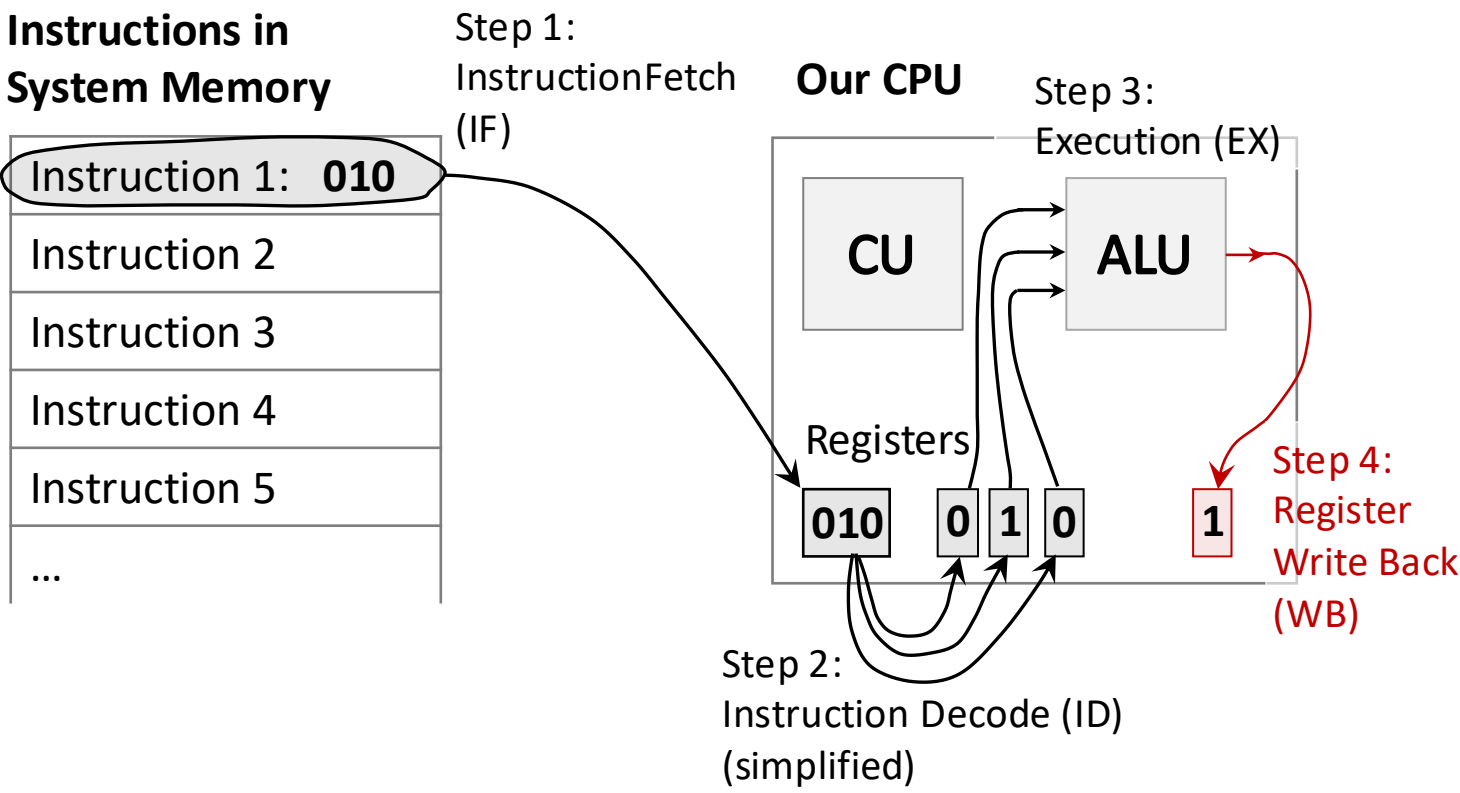
Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:



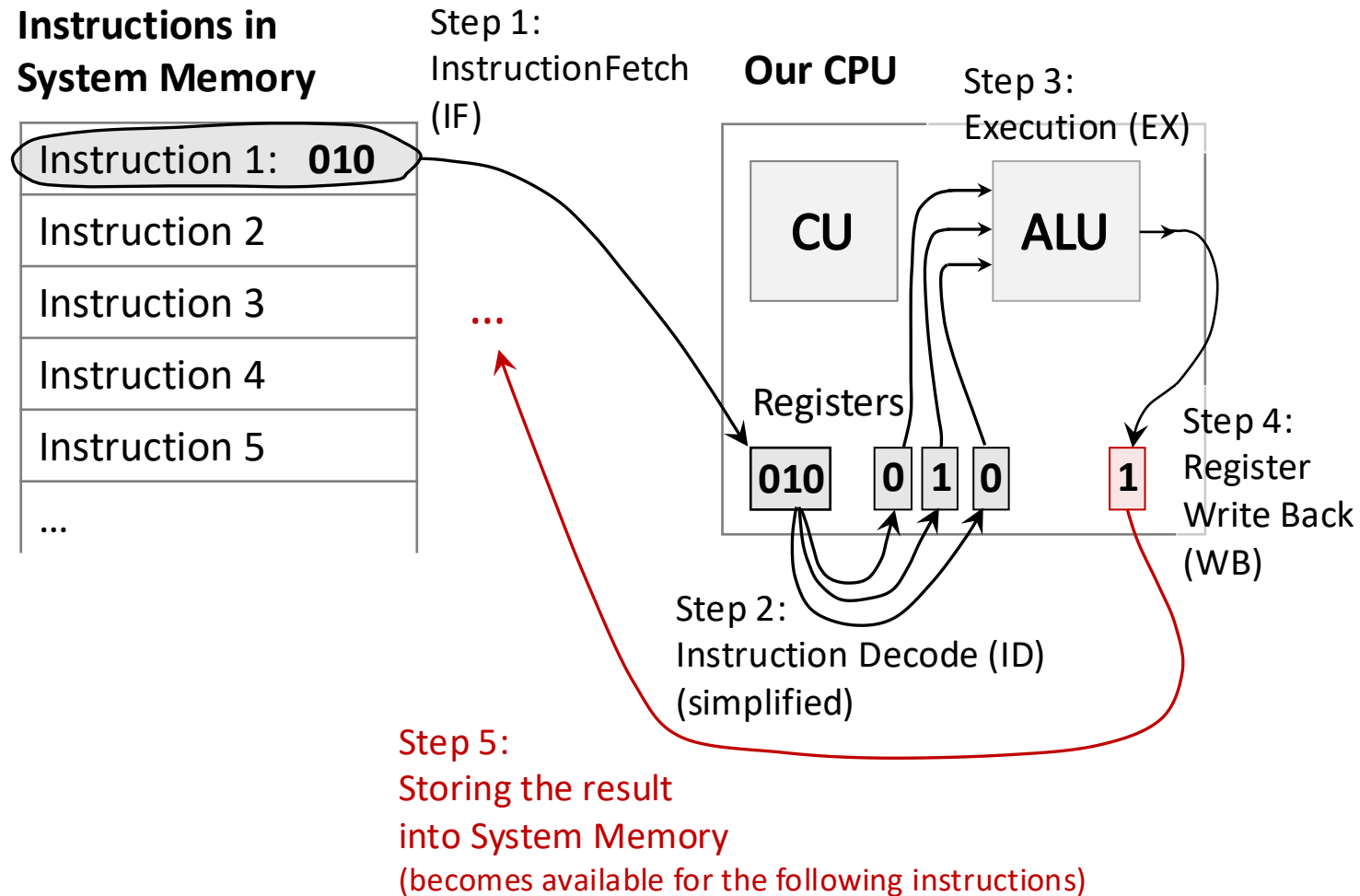
Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:



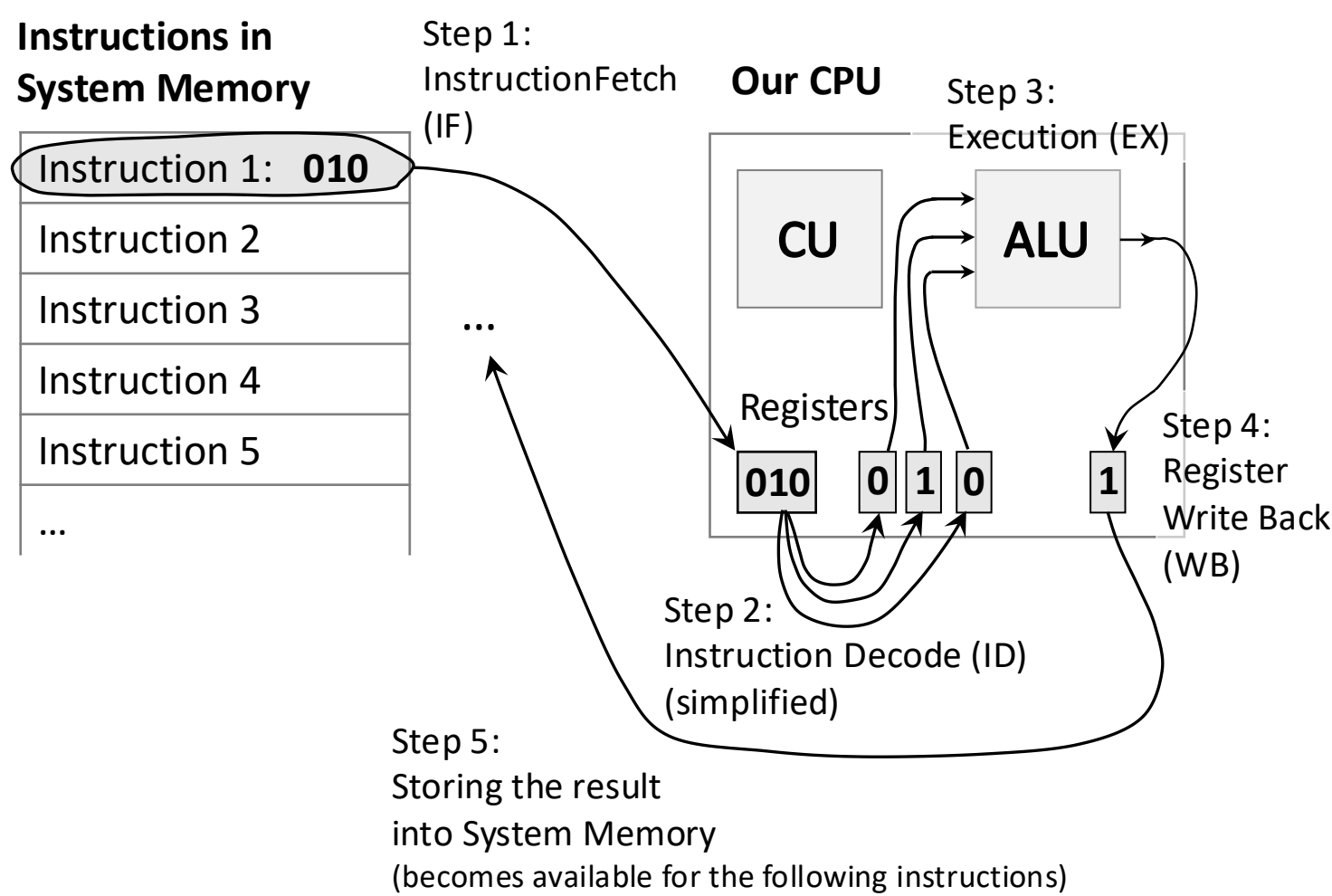
Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:



Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:

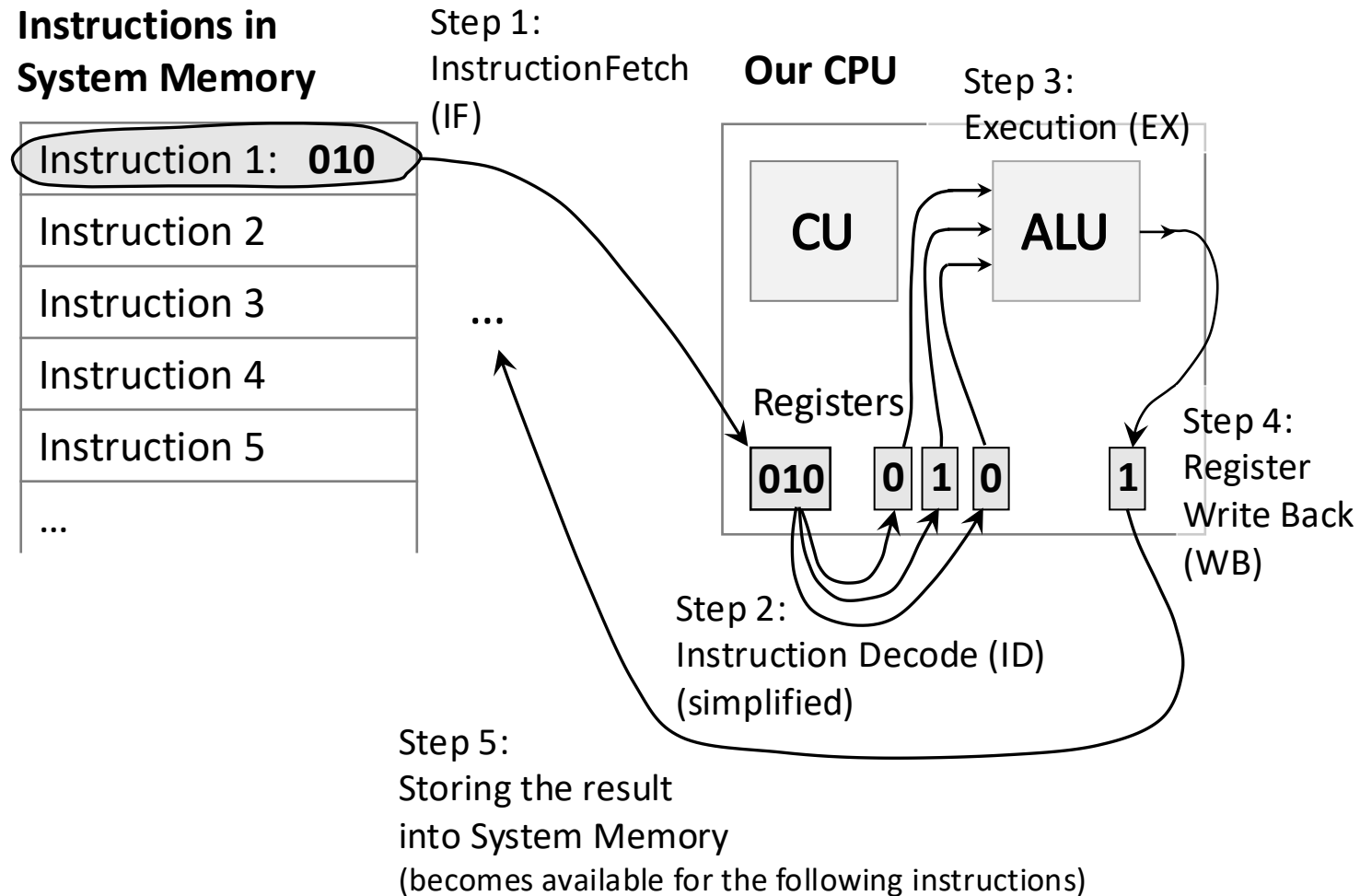


Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:

Idea of pipelining:

The same CPU executes several instructions simultaneously, but with each instruction being at a different execution stage

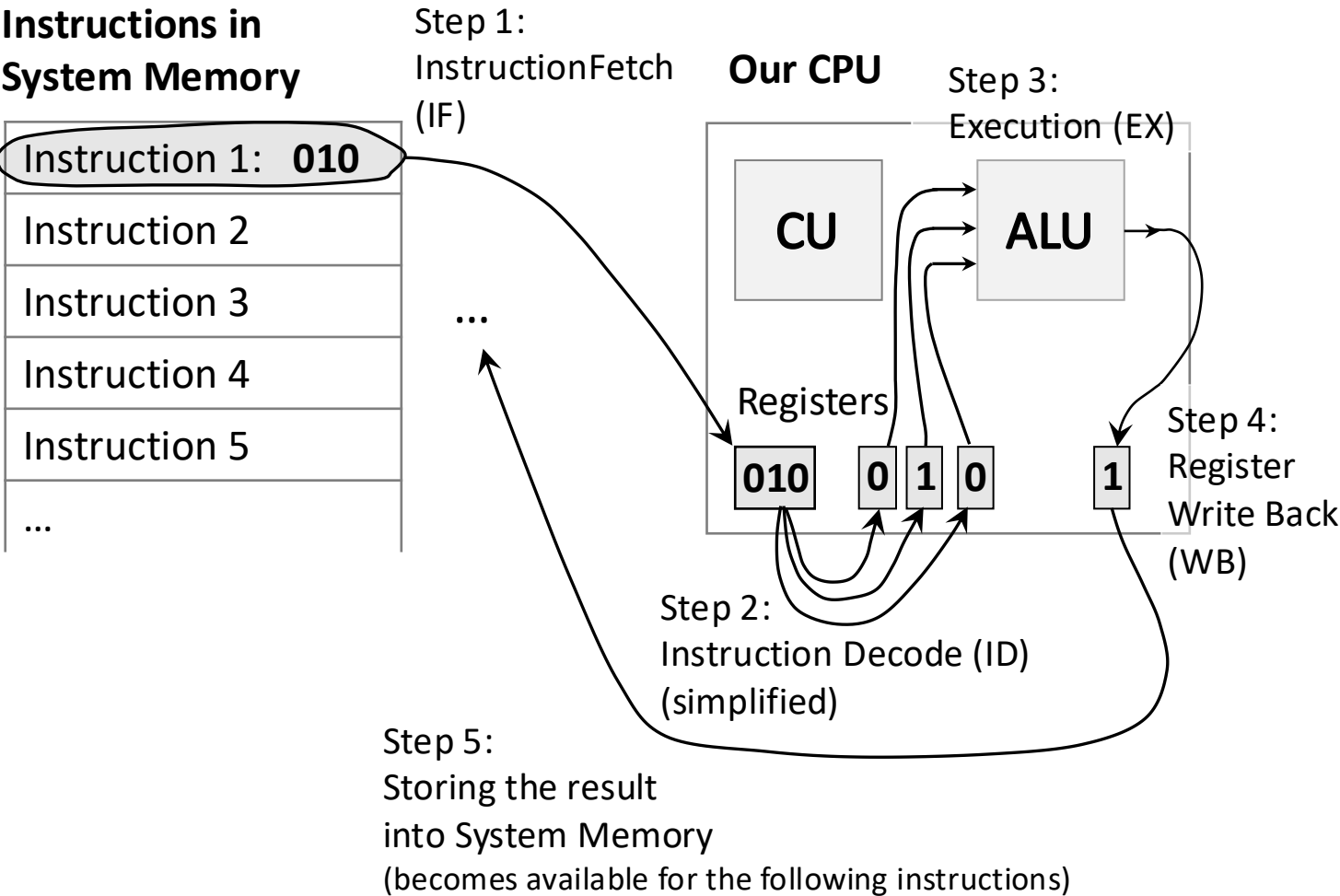


Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:

Idea of pipelining:

The same CPU executes several instructions simultaneously, but with each instruction being at a different execution stage



Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1							
2							
3							
4							
5							

Pipelined Execution of CPU Instructions

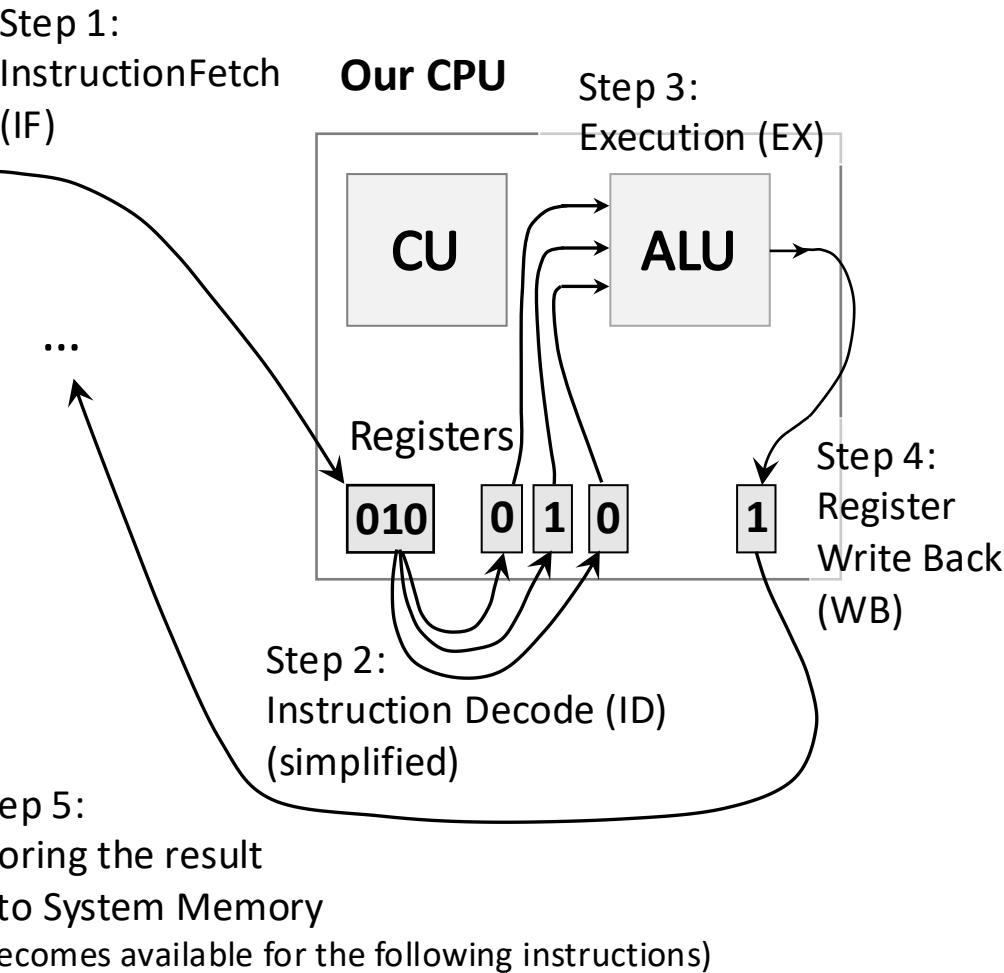
Each instruction is executed in several steps:

Idea of pipelining:

The same CPU executes several instructions simultaneously, but with each instruction being at a different execution stage

Instructions in System Memory

Instruction 1: 010
Instruction 2
Instruction 3
Instruction 4
Instruction 5
...



Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF						
2							
3							
4							
5							

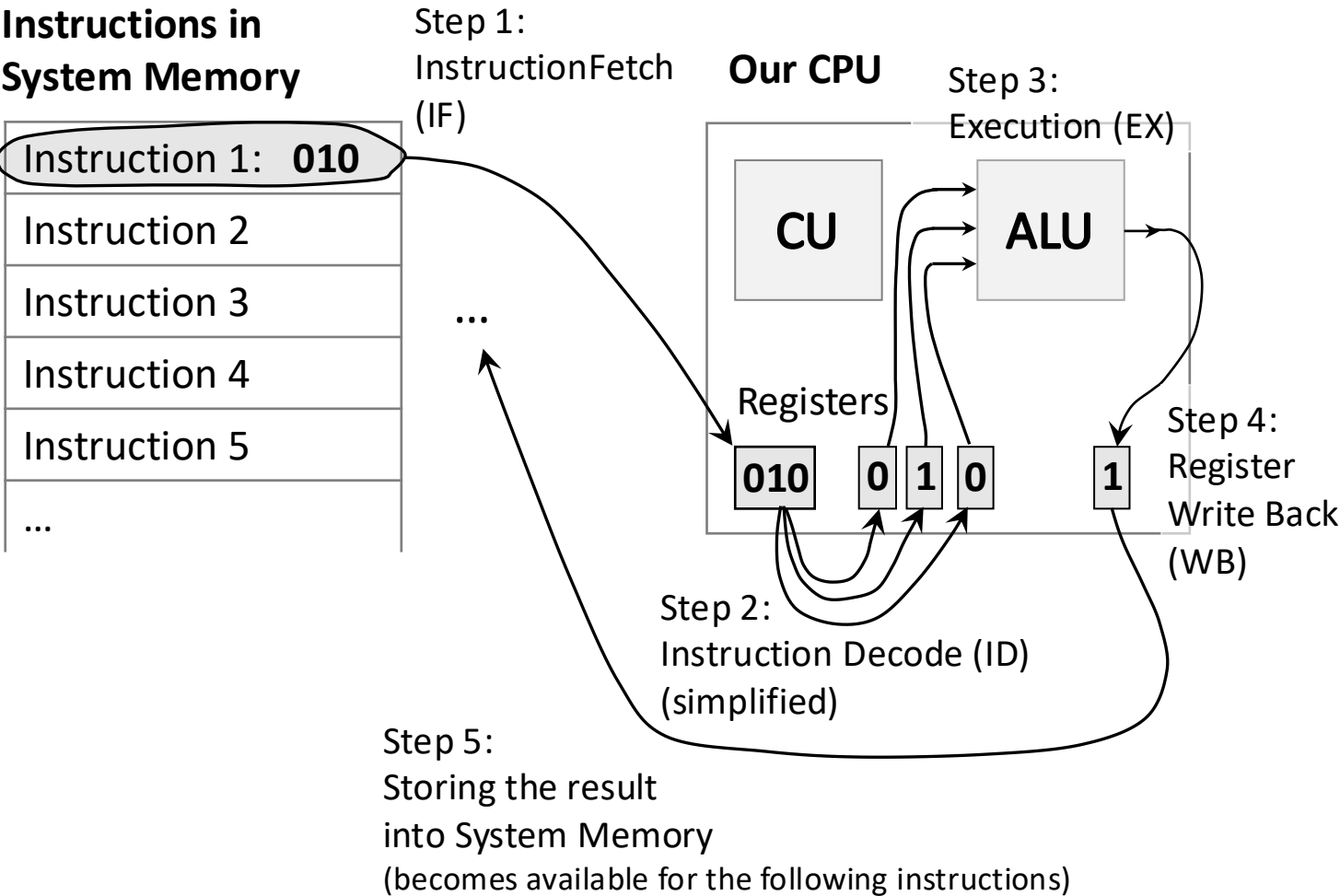
IF: Instruction Fetch

Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:

Idea of pipelining:

The same CPU executes several instructions simultaneously, but with each instruction being at a different execution stage



Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID					
2		IF					
3							
4							
5							

IF: Instruction Fetch
ID: Instruction Decode

Pipelined Execution of CPU Instructions

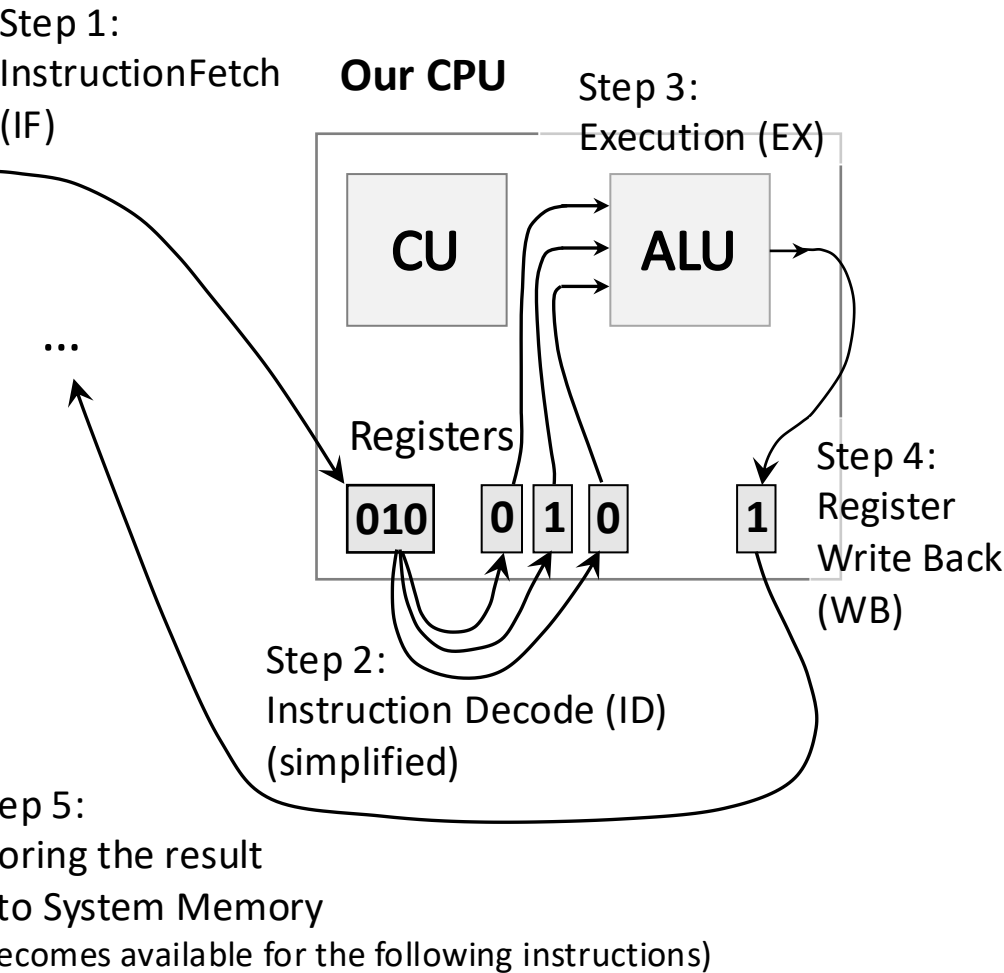
Each instruction is executed in several steps:

Idea of pipelining:

The same CPU executes several instructions simultaneously, but with each instruction being at a different execution stage

Instructions in System Memory

Instruction 1: 010
Instruction 2
Instruction 3
Instruction 4
Instruction 5
...



Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID					
2		IF					
3							
4							
5							

IF: Instruction Fetch
ID: Instruction Decode

Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:

Idea of pipelining:

The same CPU executes several instructions simultaneously, but with each instruction being at a different execution stage

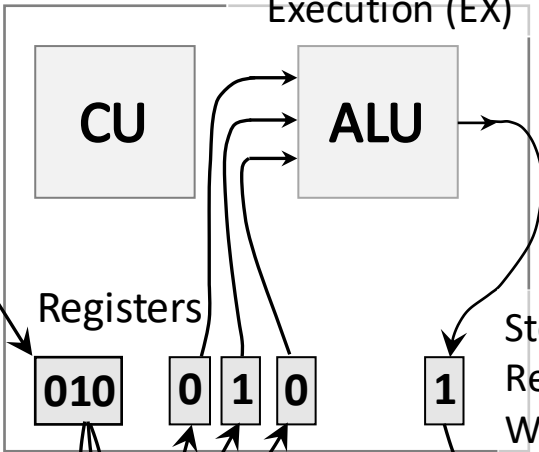
Instructions in System Memory

Instruction 1: 010
Instruction 2
Instruction 3
Instruction 4
Instruction 5
...

Step 1:
Instruction Fetch (IF)

Our CPU

Step 3:
Execution (EX)



Step 4:
Register Write Back (WB)

Step 2:
Instruction Decode (ID)
(simplified)

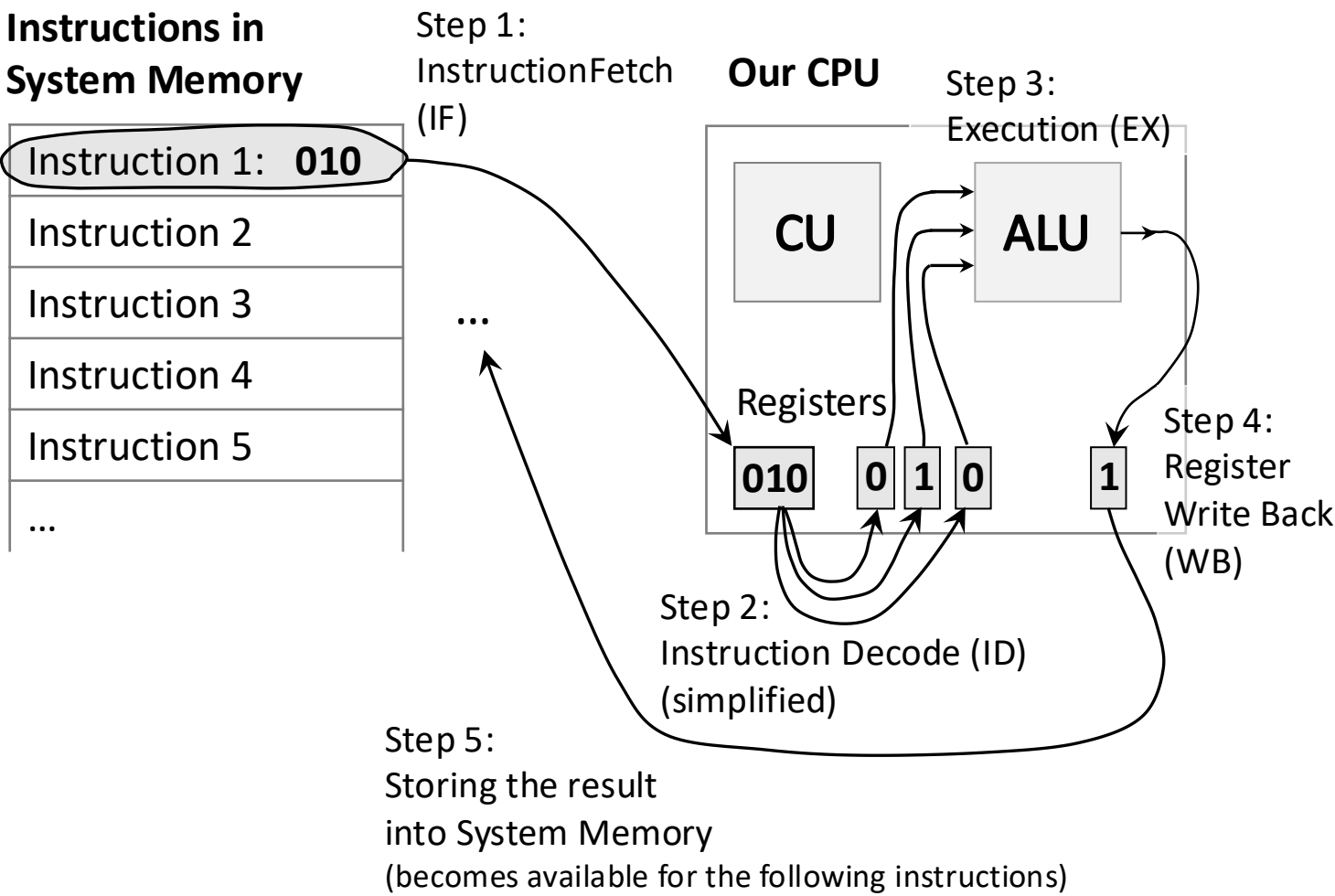
Step 5:
Storing the result
into System Memory
(becomes available for the following instructions)

Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID	EX				
2		IF	ID				
3			IF				
4							
5							

IF: Instruction Fetch
ID: Instruction Decode
EX: Execution

Pipelined Execution of CPU Instructions

Each instruction is executed in several steps:



Idea of pipelining:

The same CPU executes several instructions simultaneously, but with each instruction being at a different execution stage

Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX

IF: Instruction Fetch
ID: Instruction Decode
EX: Execution
MEM: Memory access (optional stage)
WB: register Write Back

What is the difference of “pipelining” from the “performance via parallelism” idea?

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
- 6. Performance via speculation
(prediction of program flow)**

Fundamental Ideas of Computer Architecture

Illustration with a code sample

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. **Performance via speculation
(prediction of program flow)**

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. **Performance via speculation
(prediction of program flow)**

Illustration with a code sample

Assumptions:

- **4 processors are available;**
- **No dependencies between functions**

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. **Performance via speculation
(prediction of program flow)**

Illustration with a code sample

Assumptions:

- 4 processors are available;
- No dependencies between functions

```
switch (some_int_func(...)) {  
    case 1:  func1(...);  
    case 2:  func2(...);  
    case 3:  func3(...);  
    case 4:  func4(...);  
    default: func_default(...);  
}
```

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. **Performance via speculation
(prediction of program flow)**

Illustration with a code sample

Assumptions:

- 4 processors are available;
- No dependencies between functions

 **Processor 1**

```
switch (some_int_func(...)) {  
    case 1:  func1(...);  
    case 2:  func2(...);  
    case 3:  func3(...);  
    case 4:  func4(...);  
    default: func_default(...);  
}
```

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. **Performance via speculation
(prediction of program flow)**

Illustration with a code sample

Assumptions:

- 4 processors are available;
- No dependencies between functions

```
switch (some_int_func(...)) {  
    case 1: func1(...);  
    case 2: func2(...);  
    case 3: func3(...);  
    case 4: func4(...);  
    default: func_default(...);  
}
```

Processor 1

Processor 2

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. **Performance via speculation
(prediction of program flow)**

Illustration with a code sample

Assumptions:

- 4 processors are available;
- No dependencies between functions

```
switch (some_int_func(...)) {  
    case 1: func1(...);  
    case 2: func2(...);  
    case 3: func3(...);  
    case 4: func4(...);  
    default: func_default(...);  
}
```

Processor 1

Processor 3

Processor 2

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. **Performance via speculation
(prediction of program flow)**

Illustration with a code sample

Assumptions:

- 4 processors are available;
- No dependencies between functions

```
switch (some_int_func(...)) {  
    case 1: func1(...);  
    case 2: func2(...);  
    case 3: func3(...);  
    case 4: func4(...);  
    default: func_default(...);  
}
```

Processor 1

Processor 3

Processor 4

Processor 2

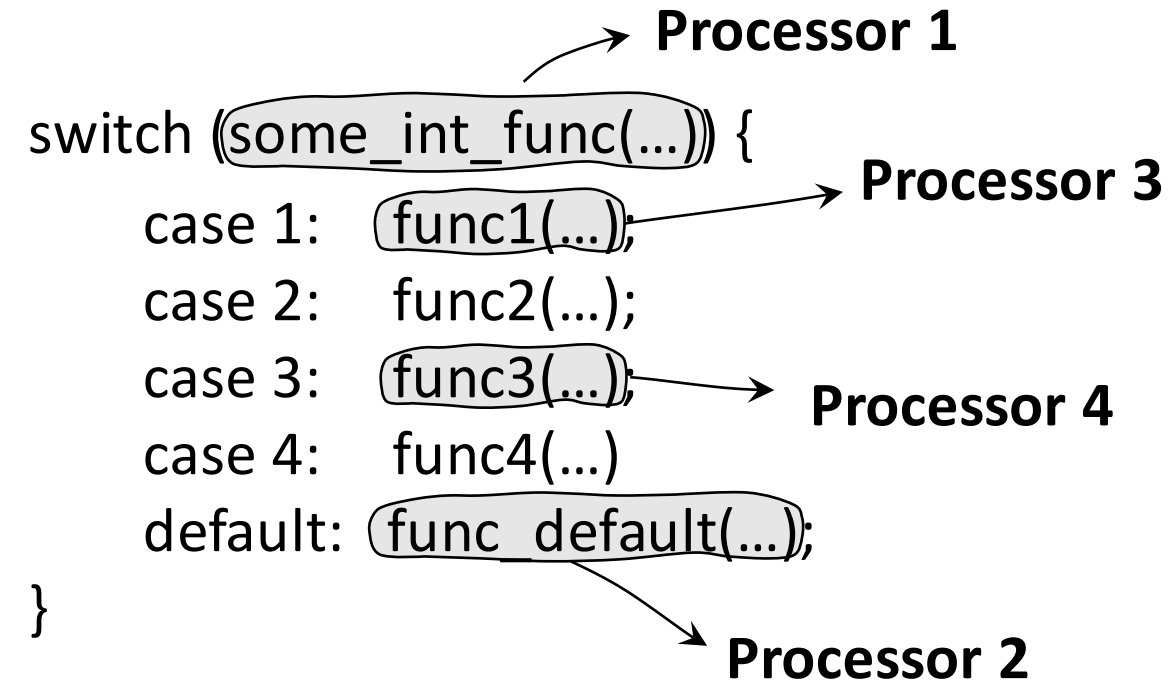
Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. **Performance via speculation
(prediction of program flow)**

Illustration with a code sample

Assumptions:

- 4 processors are available;
- No dependencies between functions



For this example, we aim at predicting (“speculating”) 3 cases with the highest probability to occur

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. **Dependability via redundancy**

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
- 7. Dependability via redundancy**

Some redundant (or “spare”) components are introduced (e.g. CPU or memory unit), to increase the reliability of a computer platform (e.g. in a spacecraft);

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. **Make the common case fast**

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
- 8. Make the common case fast**

The computer platform should be optimized for the most common use case expected

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
- 9. Finite State Machines (FSM)**

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

Finite State Machine – a (finite) set of states and transitions between them; a convenient way to model systems behaviour

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

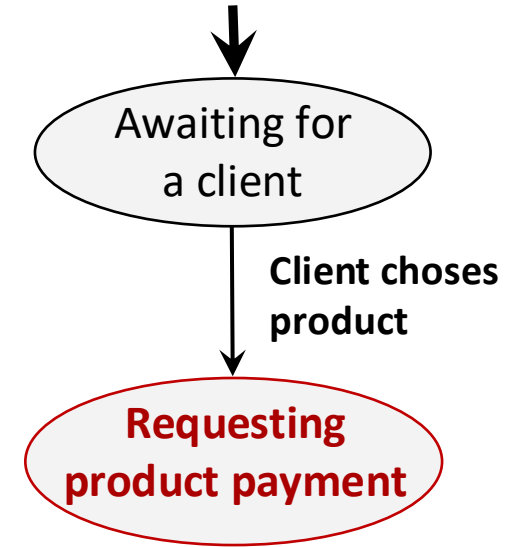
Sample FSM for a vending machine:



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

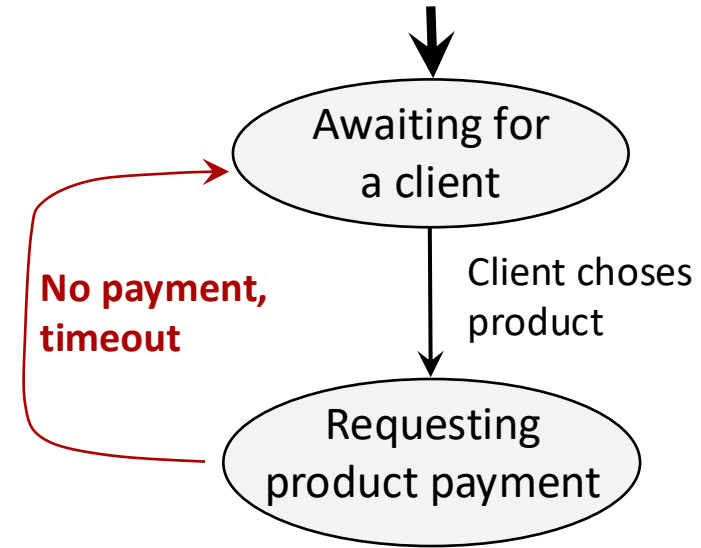
Sample FSM for a vending machine:



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

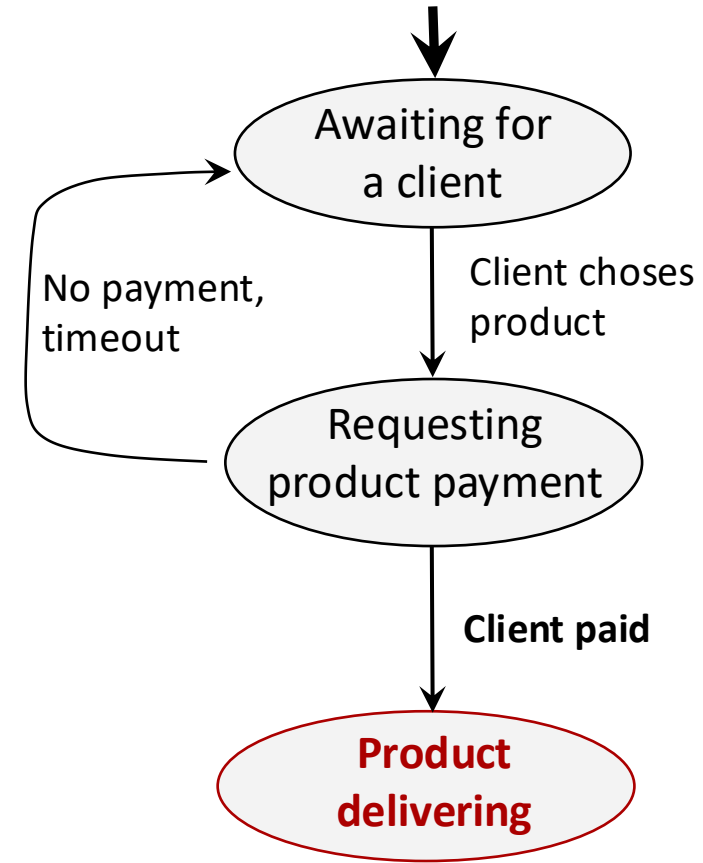
Sample FSM for a vending machine:



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

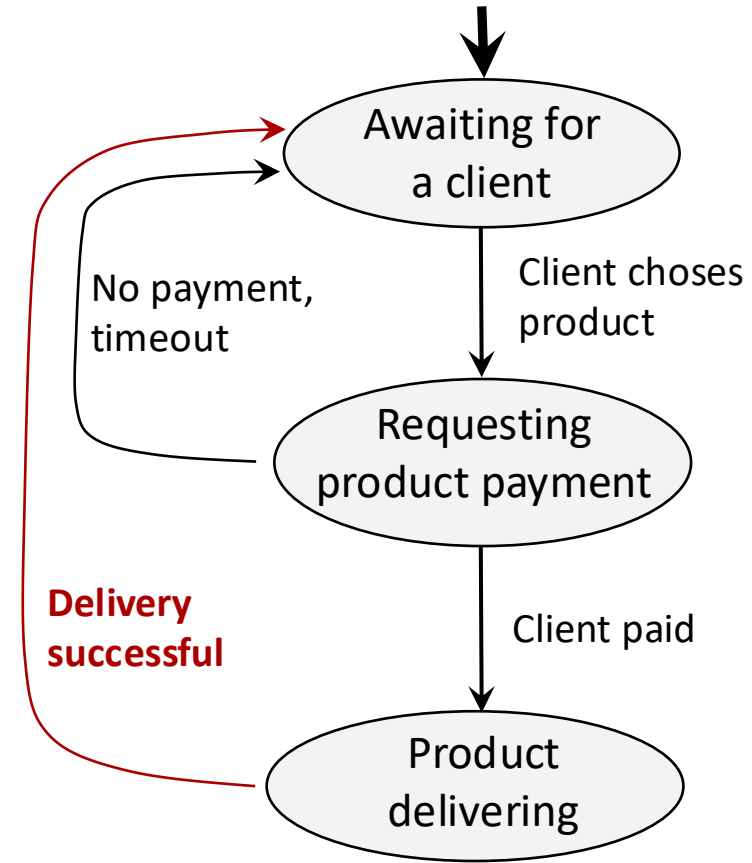
Sample FSM for a vending machine:



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

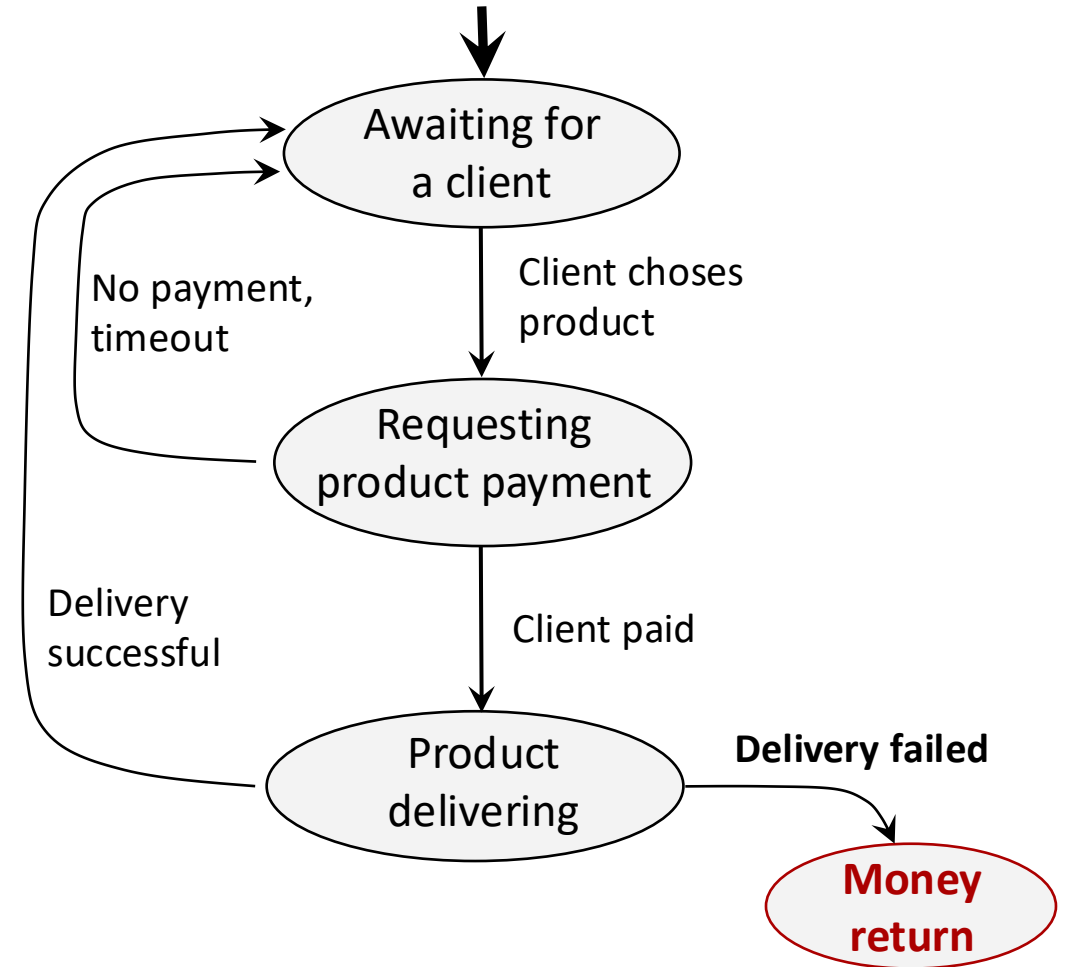
Sample FSM for a vending machine:



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

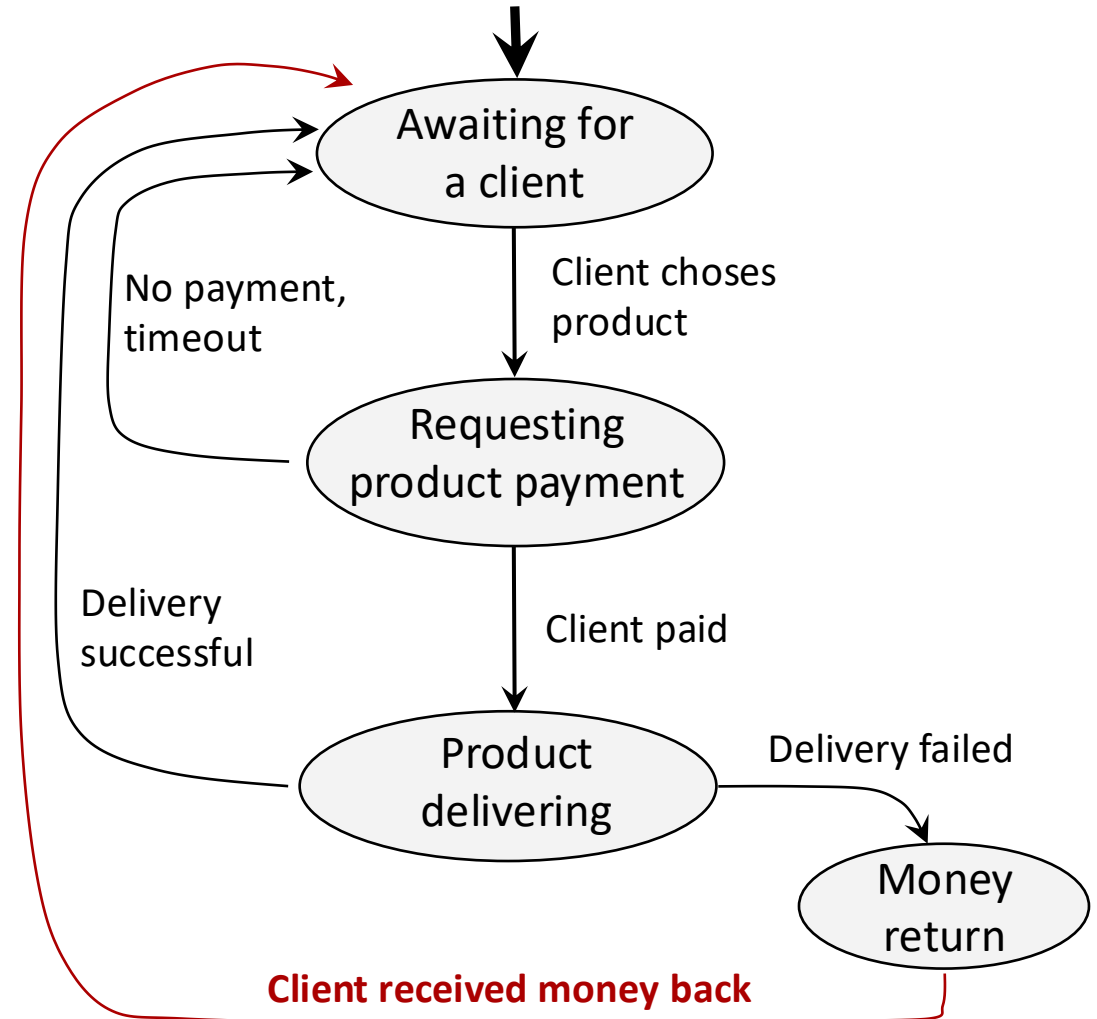
Sample FSM for a vending machine:



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

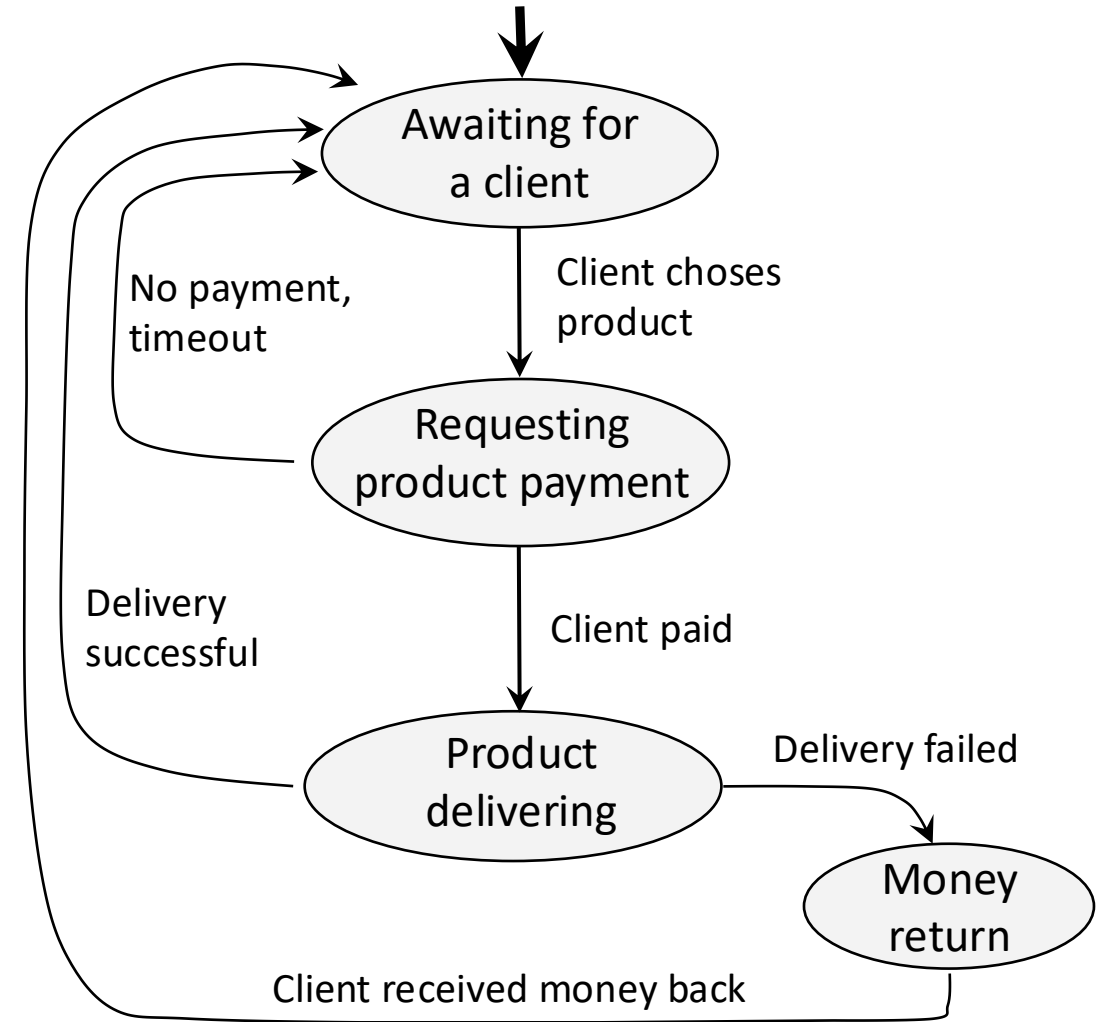
Sample FSM for a vending machine:



Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

Sample FSM for a vending machine:

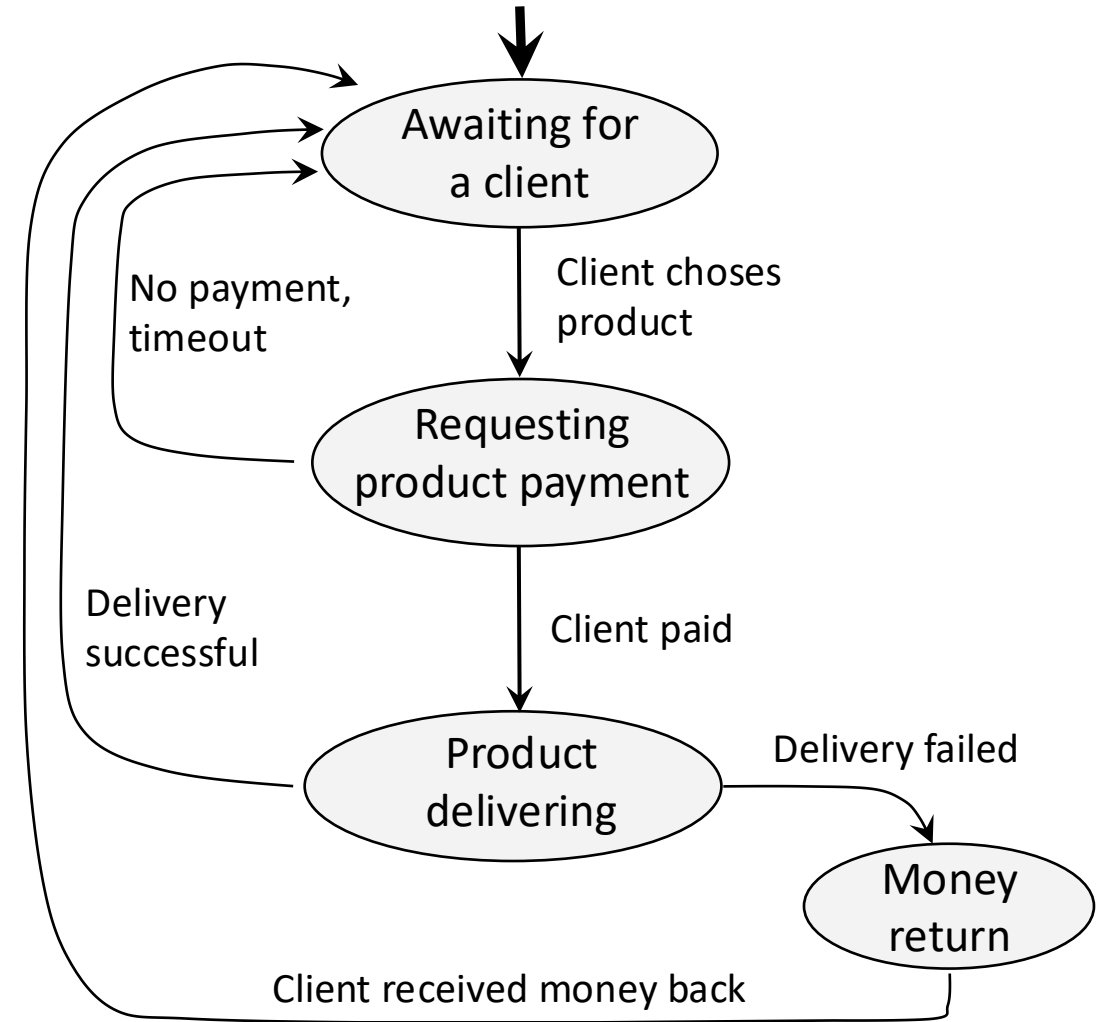


This FSM describes a process from a vending machine perspective, but we could also describe it from a client perspective

Fundamental Ideas of Computer Architecture

1. Hierarchy of memories
2. Design simplification via abstraction
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation
7. Dependability via redundancy
8. Make the common case fast
9. **Finite State Machines (FSM)**

Sample FSM for a vending machine:



FSM is used to model the behavior of computer systems, e.g. processor cache

Fundamental CA Ideas: Summary

1. Hierarchy of memories
2. Design simplification via abstraction use
3. Moore's law for CPU performance
4. Performance via parallelism
5. Performance via pipelining
6. Performance via speculation (prediction)
7. Dependability (reliability) via redundancy (spare parts)
8. Make the common case fast
9. Finite State Machines