K22 - Operating Systems: Design Principles and Internals

Fall 2025 @dit

Vaggelis Atlidakis

References: Similar OS courses @Columbia, @Stanford, @UC San Diego, @Brown, @di (previous years); and textbooks: Operating Systems: Three Easy Pieces, Operating Systems: Principles and Practice, Operating System Concepts, Linux Kernel Development, Understanding the Linux Kernel

K22 - Operating Systems: Design Principles and Internals

Fall 2025 @dit

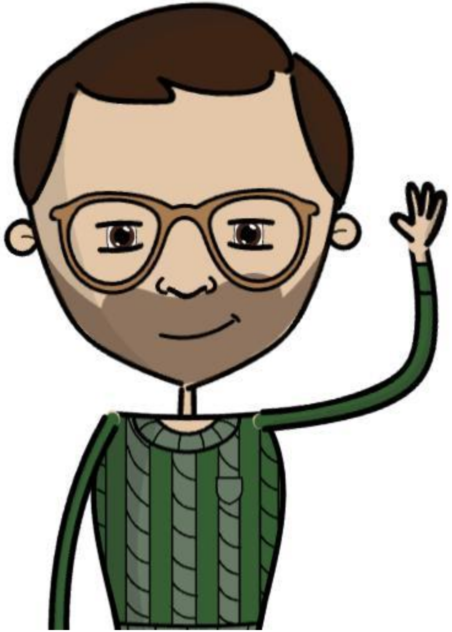
Vaggelis Atlidakis

std05010

References: Similar OS courses @Columbia, @Stanford, @UC San Diego, @Brown, @di (previous years); and textbooks: Operating Systems: Three Easy Pieces, Operating Systems: Principles and Practice, Operating System Concepts, Linux Kernel Development, Understanding the Linux Kernel

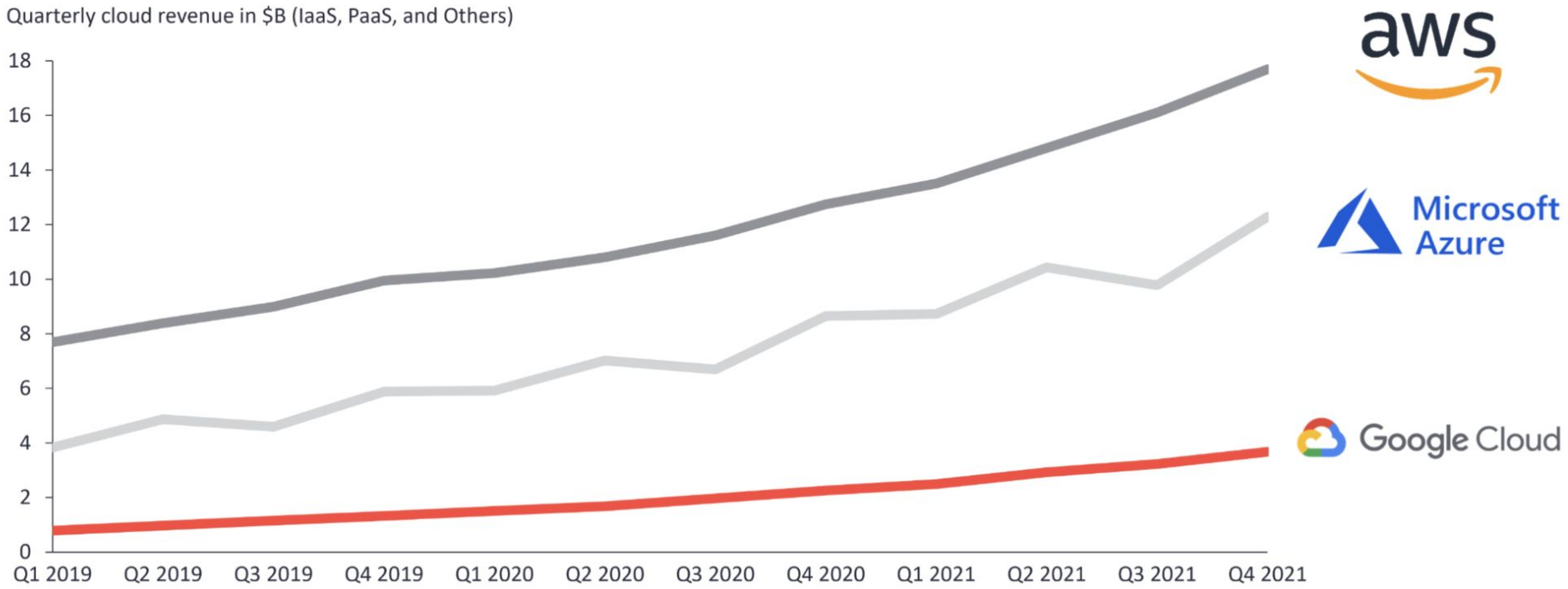
What you probably ordered?

What you ended up getting?



Why study OSes?

Revenue of leading cloud vendors (2019-21)



[1] https://iot-analytics.com/cloud-market/

Why study OSes?

- **These days:** Code developed with superficial understanding
  - No free lunch: Bugs won't fix themselves in production

- **In the near future:** Requirement for software engineers with legit systems understanding will soon be at an all-time high
  - A line of code is not just a line of code
  - Myriads of things happen under the hood

Why study OSes?

- These days: Code developed with superficial understanding
  - No free lunch: Bugs won't fix themselves in production

- In the near future: Requirement for software engineers with legit systems understanding will soon be at an all-time high
  - A line of code is not just a line of code
  - Myriads of things happen under the hood

- Learning to navigate a complex codebase, such as an OS kernel, will make you a **fearless** software engineer!

What you need to do to pass the class?

- Programming assignments (groups of three): **50%**
  - You'll add features to the Linux kernel and run a VM with it
  - You'll need to understand lots of code; won't need to write much
  - **Prerequisites**
    - Do you have a laptop that can run a Linux VM?
    - Can you make teams of three ASAP?
    - Can you use git?
- Exams: **50%**
  - Midterm: 20%
  - Final: 30%

What you <u>really</u> need to do?

- Come to class, ask questions, be curious
- Choose a team you can collaborate well with
- Own your code
  - You know what you must not do…
  - This is not a class you pass by outsourcing your work
  - There is no magic: You do the work ⇒ You pass the class
  - GenAI usage: Use LLMs to help you understand

- It's a difficult course. Hang in there, we will help…

Overview

- We'll start from hardware and follow a question-oriented approach

Overview

- We'll start from hardware and follow a question-oriented approach
  - Intro [Q: What is an OS?]
  - Events [Q: When does the OS run?]
  - Runtime [Q: How does a program look like in memory?]

* Basic (H/W & S/W)

Overview

- We'll start from hardware and follow a question-oriented approach
  - Intro [Q: What is an OS?]
  - Events [Q: When does the OS run?]
  - Runtime [Q: How does a program look like in memory?]
  - Processes [Q: What is a process?]
  - Threads [Q: What is a thread?]
  - Files [Q: What is a file descriptor?]

* Basic (H/W & S/W)
* Abstractions

Overview

- We'll start from hardware and follow a question-oriented approach
  - Intro [Q: What is an OS?]
  - Events [Q: When does the OS run?]
  - Runtime [Q: How does a program look like in memory?]
  - Processes [Q: What is a process?]
  - IPC [Q: How do processes communicate?]
  - Threads [Q: What is a thread?]
  - Synchronization [Q: What goes wrong w/o synchronization?]

  - Files [Q: What is a file descriptor?]

* Basic (H/W & S/W)
* **Abstractions**
* **Primitives**

Overview

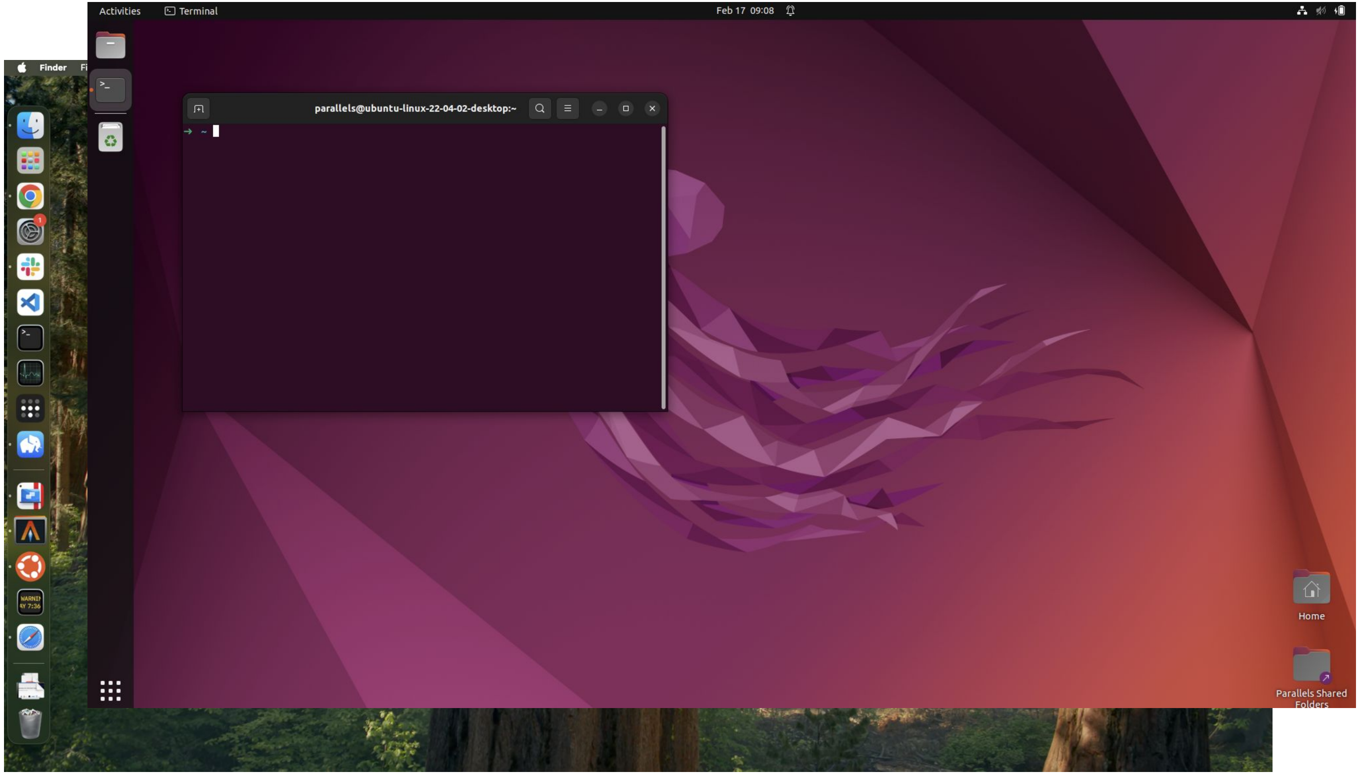
- We'll start from hardware and follow a question-oriented approach
    - Intro [Q: What is an OS?]
    - Events [Q: When does the OS run?]
    - Runtime [Q: How does a program look like in memory?]
    - Processes [Q: What is a process?]
    - IPC [Q: How do processes communicate?]
    - Threads [Q: What is a thread?]
    - Synchronization [Q: What goes wrong w/o synchronization?]
    - Time Management [Q: What is scheduling?]
    - Memory Management [Q: What is virtual memory?]
    - Files [Q: What is a file descriptor?]
    - Storage Management [Q: How do we allocate disk space to files?]

* Basic (H/W & S/W)
* Abstractions
* Primitives
* Mechanisms

What is an OS?

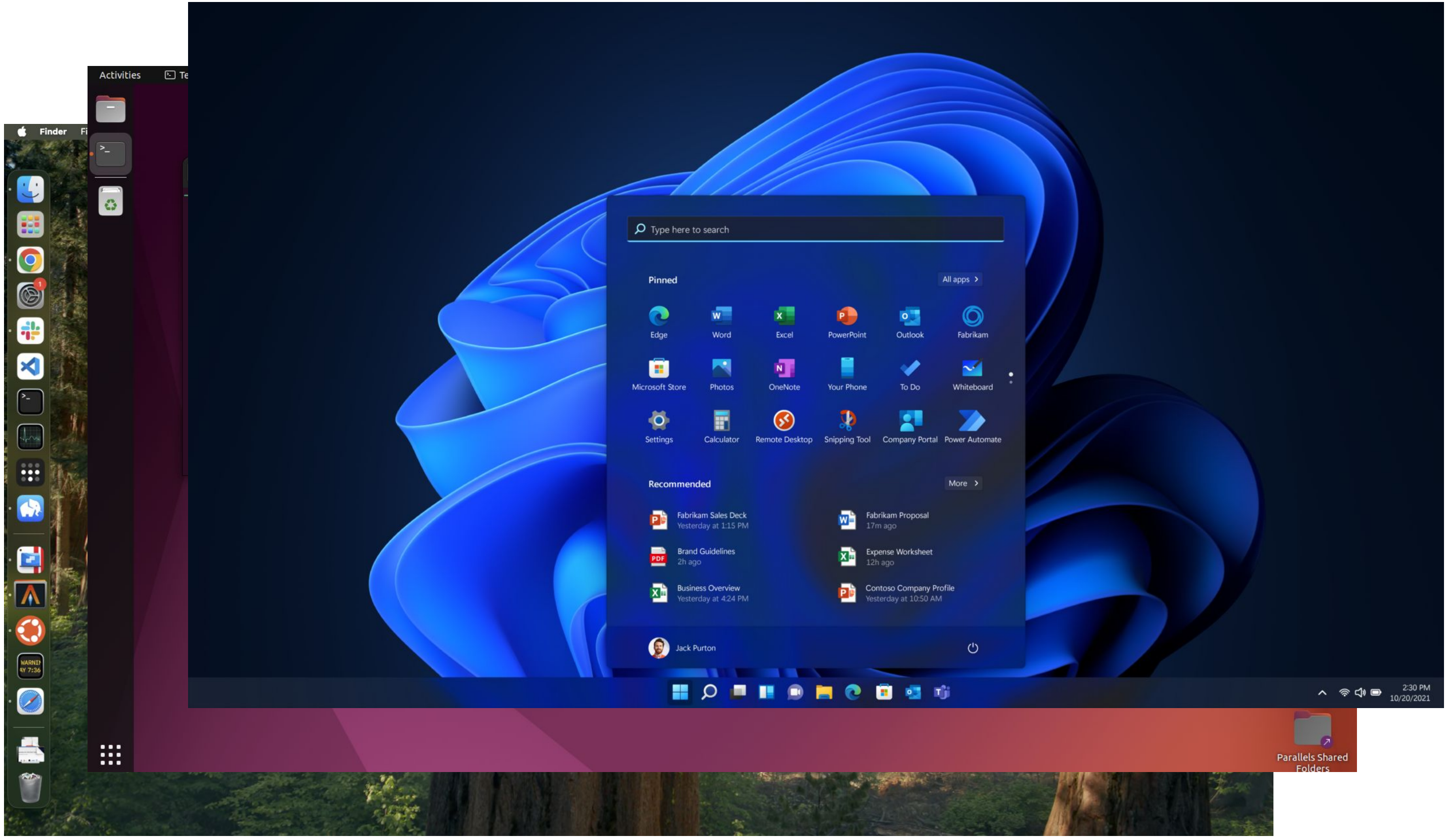

Finder File Edit View Go Window Help
Mon Feb 17 9:07 AM
Alacritty
Recents
stuff

What is an OS?



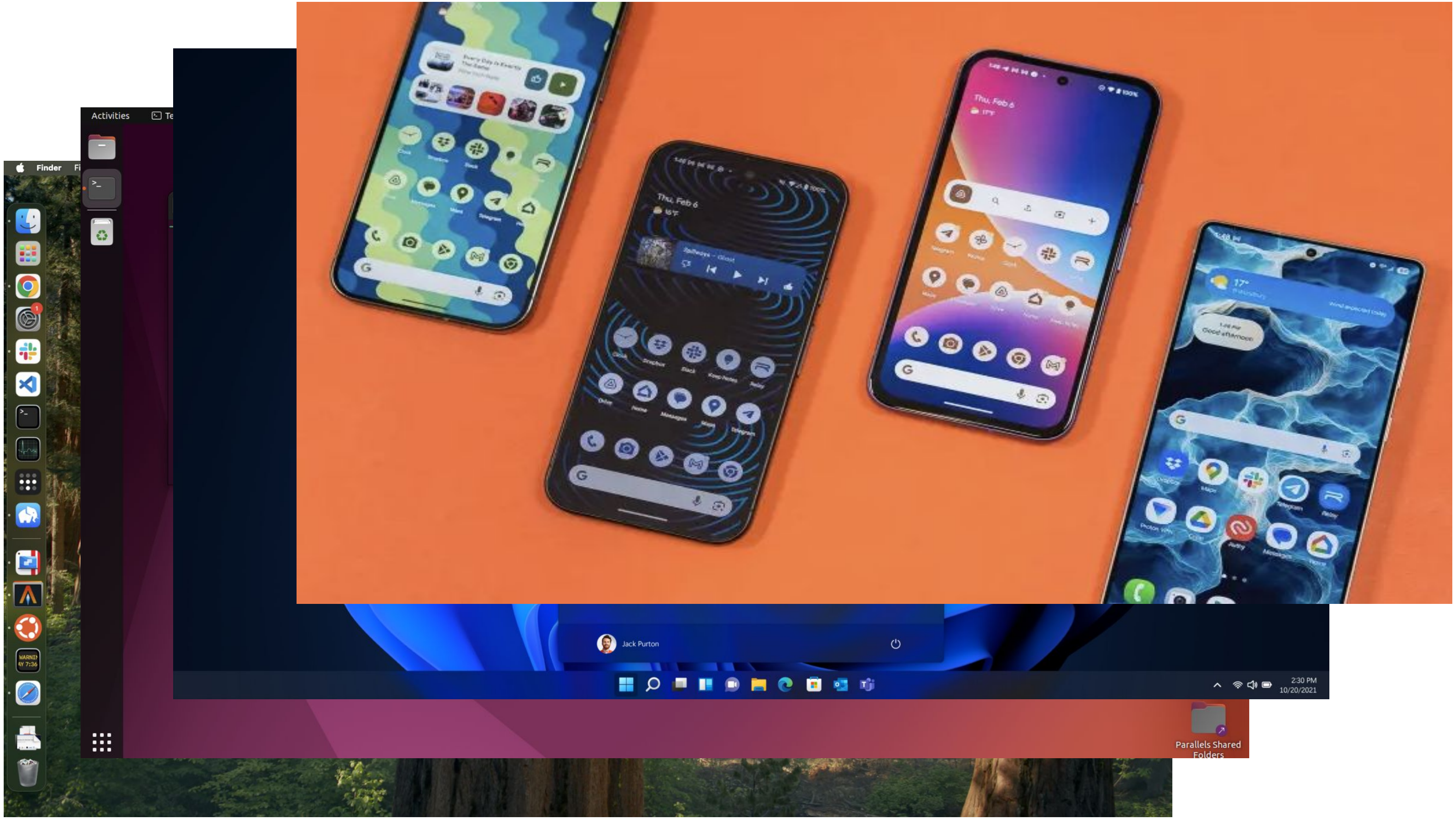
Activities
Terminal
Feb 17 09:08
parallels@ubuntu-linux-22-04-02-desktop:~
Finder
Home
Parallels Shared Folders

What is an OS?



Activities
Finder
Type here to search
Pinned
All apps
Edge
Word
Excel
PowerPoint
Outlook
Fabrikam
Microsoft Store
Photos
OneNote
Your Phone
To Do
Whiteboard
Settings
Calculator
Remote Desktop
Snipping Tool
Company Portal
Power Automate
Recommended
More
Fabrikam Sales Deck
Yesterday at 1:15 PM
Fabrikam Proposal
17m ago
Brand Guidelines
2h ago
Expense Worksheet
12h ago
Business Overview
Yesterday at 4:24 PM
Contoso Company Profile
Yesterday at 10:50 AM
Jack Purton
2:30 PM
10/20/2021
Parallels Shared Folders

What is an OS?


Finder
Activities
Thu, Feb 6
Jack Purton
2:30 PM
10/20/2021
Parallels Shared Folders

What is an OS?

What is an OS?

- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

What is an OS?

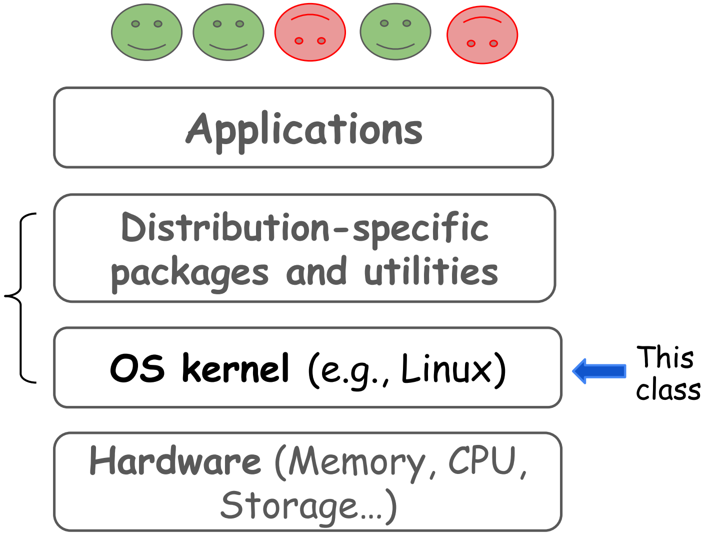
- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

Why need an OS?

What is an OS?

- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

Why need an OS?

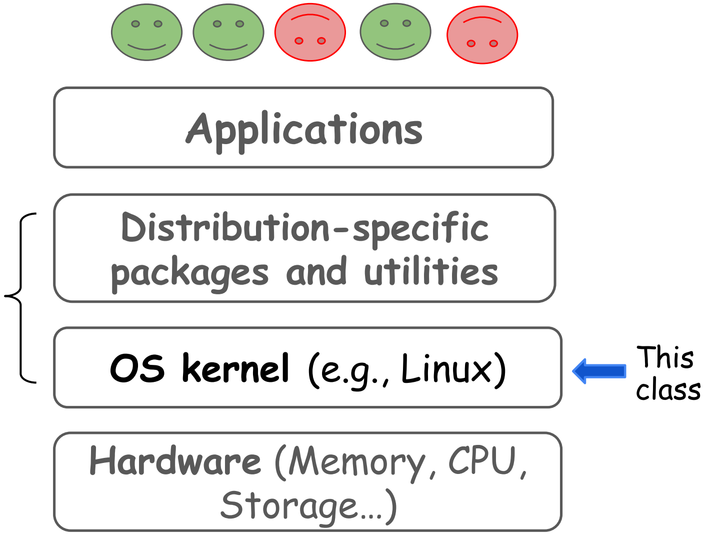


What is an OS?

- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

Why need an OS?

- Multiple users not necessarily sane, build and run different applications, at the same time on shared hardware

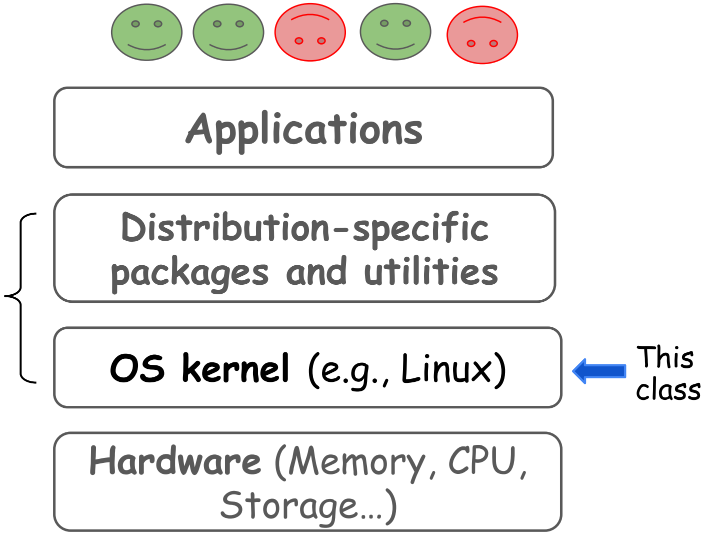


What is an OS?

- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

Why need an OS?

- Multiple users not necessarily sane, build and run different applications, at the same time on shared hardware (design assumptions)



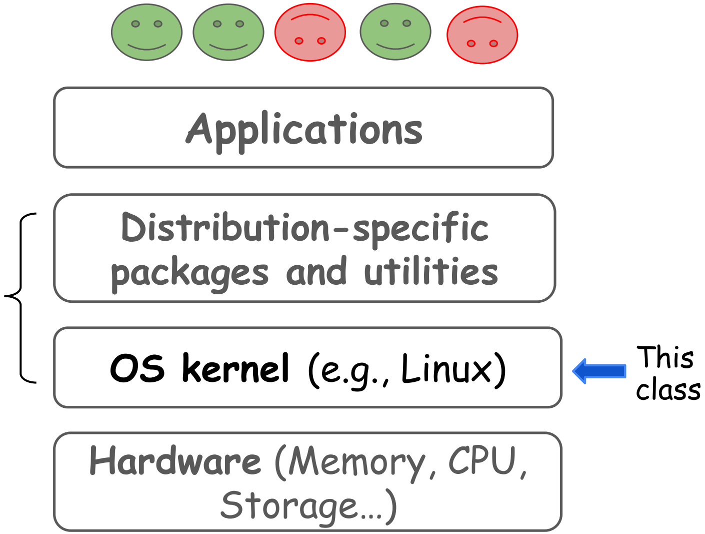
What is an OS?

- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

Why need an OS?

- Multiple users not necessarily sane, build and run different applications, at the same time on shared hardware (design assumptions)

Desirable properties?



What is an OS?

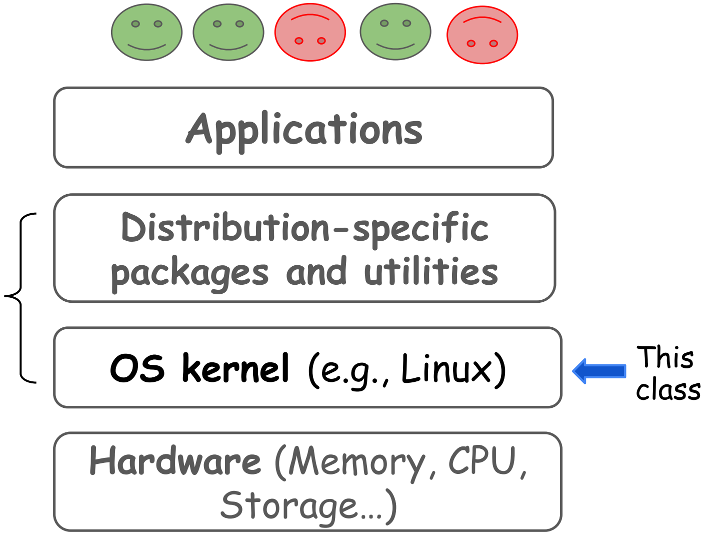
- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

Why need an OS?

- Multiple users not necessarily sane, build and run different applications, at the same time on shared hardware (design assumptions)

Desirable properties?

- ***Security***: Does the system behave as expected in adversarial contexts?



What is an OS?

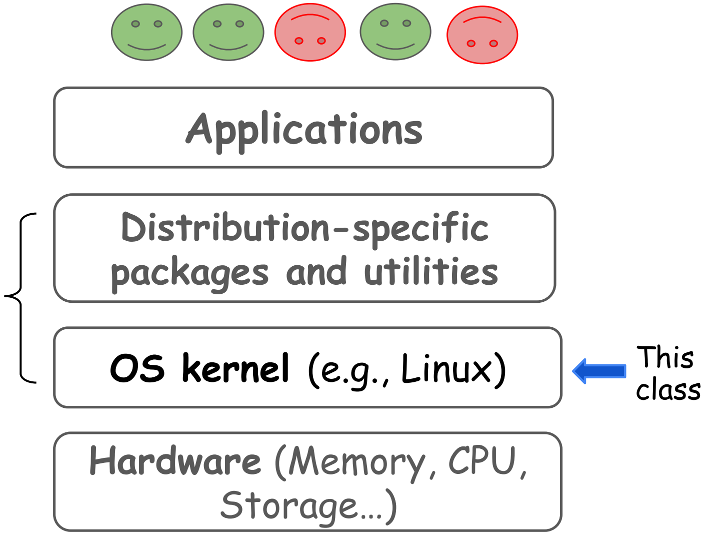
- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

Why need an OS?

- Multiple users not necessarily sane, build and run different applications, at the same time on shared hardware (design assumptions)

Desirable properties?

- ***Security***: Does the system behave as expected in adversarial contexts?
- ***Reliability***: Does the system behave as expected given benign failures?



What is an OS?

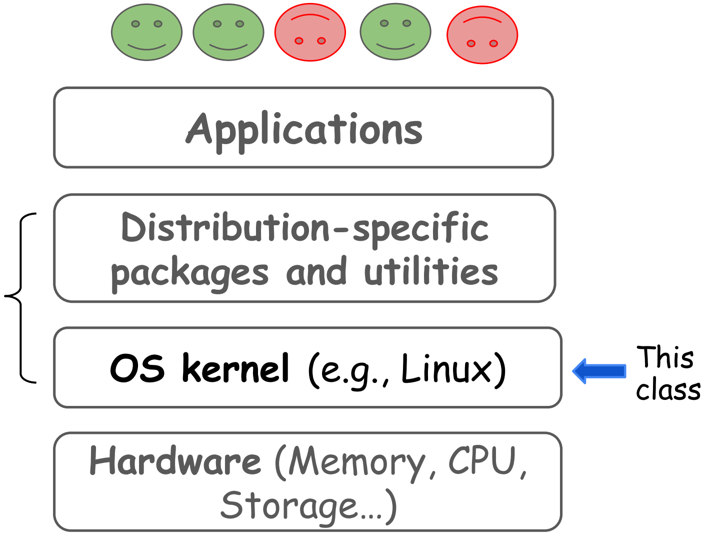
- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

Why need an OS?

- Multiple users not necessarily sane, build and run different applications, at the same time on shared hardware (design assumptions)

Desirable properties?

- ***Security***: Does the system behave as expected in adversarial contexts?
- ***Reliability***: Does the system behave as expected given benign failures?
- ***Portability***: Is it easy for developers to build and maintain apps?



What is an OS?

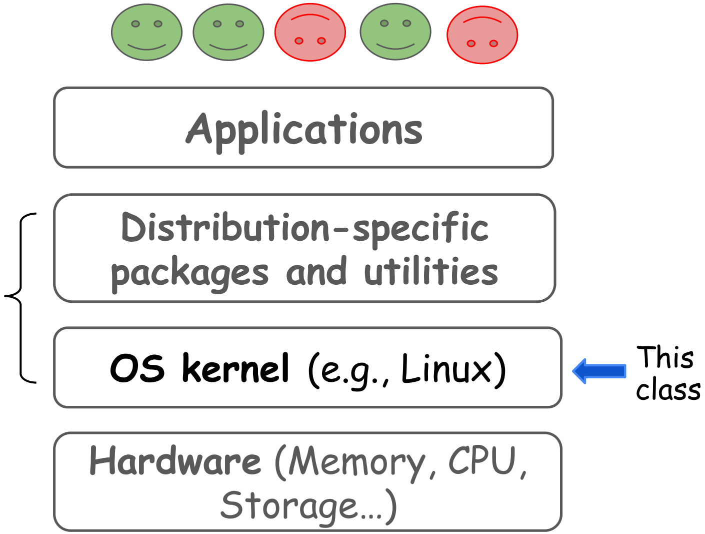
- A layer of abstraction that translates the hardware of a machine into (standardized) software concepts that applications use

Why need an OS?

- Multiple users not necessarily sane, build and run different applications, at the same time on shared hardware (design assumptions)

Desirable properties?

- ***Security***: Does the system behave as expected in adversarial contexts?
- ***Reliability***: Does the system behave as expected given benign failures?
- ***Portability***: Is it easy for developers to build and maintain apps?
- ***Fairness***: Is the system "pleasant" to use?



What is the hardware we build OSes for?

What is the hardware we build OSes for?

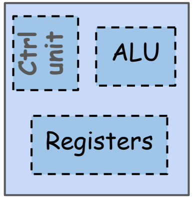
- **Physical Memory (PM)**
  - Stores data addressable on a byte granularity

Physical memory

Addr	Data
0x000	0x1
...	...
0x3FF	0xa
0x400	0x5

What is the hardware we build OSes for?

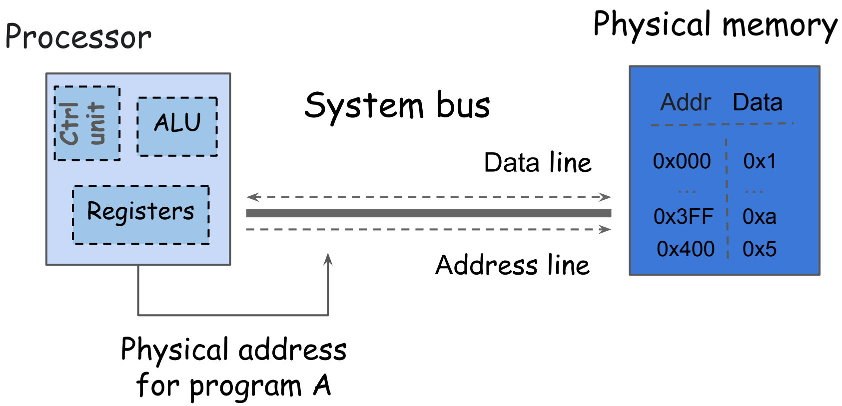
Processor



Physical memory

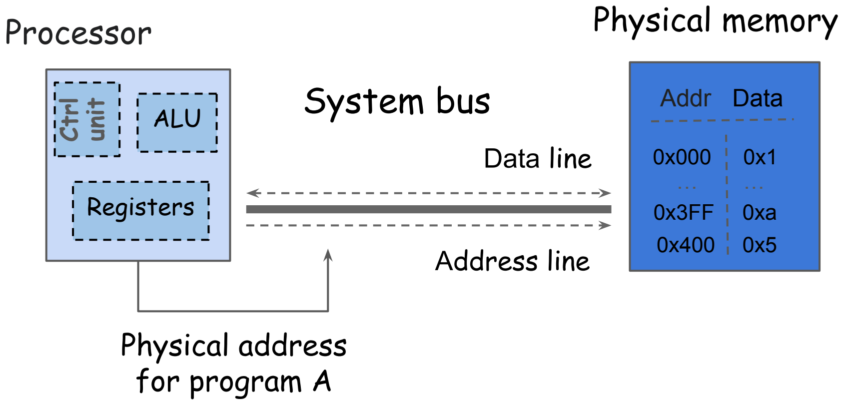
- **Physical Memory (PM)**
  - Stores data addressable on a byte granularity
- **Processor**
  - Reads data from memory to its registers
  - Performs computations
  - Writes the data from its registers to memory

What is the hardware we build OSes for?



- **Physical Memory (PM)**
  - Stores data addressable on a byte granularity
- **Processor**
  - Reads data from memory to its registers
  - Performs computations
  - Writes the data from its registers to memory
- **System bus**
  - Memory and processor communication channel

What is the hardware we build OSes for?

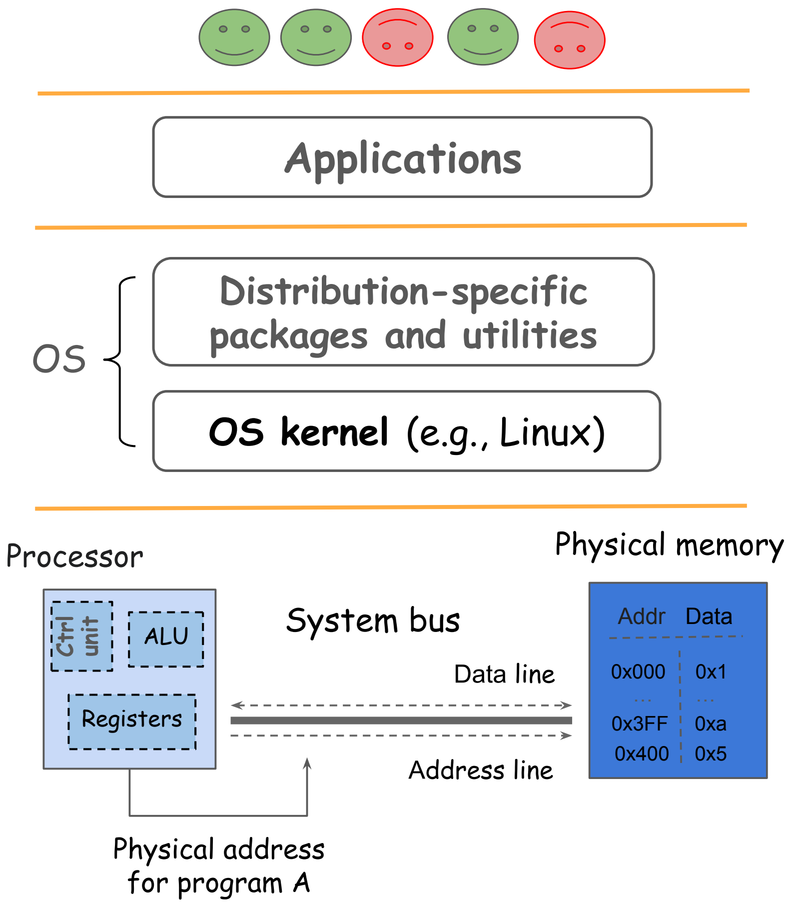


- **Physical Memory (PM)**
  - Stores data addressable on a byte granularity
- Processor
  - Reads data from memory to its registers
  - Performs computations
  - Writes the data from its registers to memory
- **System bus**
  - Memory and processor communication channel
- Is this model satisfactory?
- Reminds you of something?

Designing an OS kernel

Desirable properties

1) *Security*, 2) *Reliability*, 3) *Portability*, 4) *Fairness*

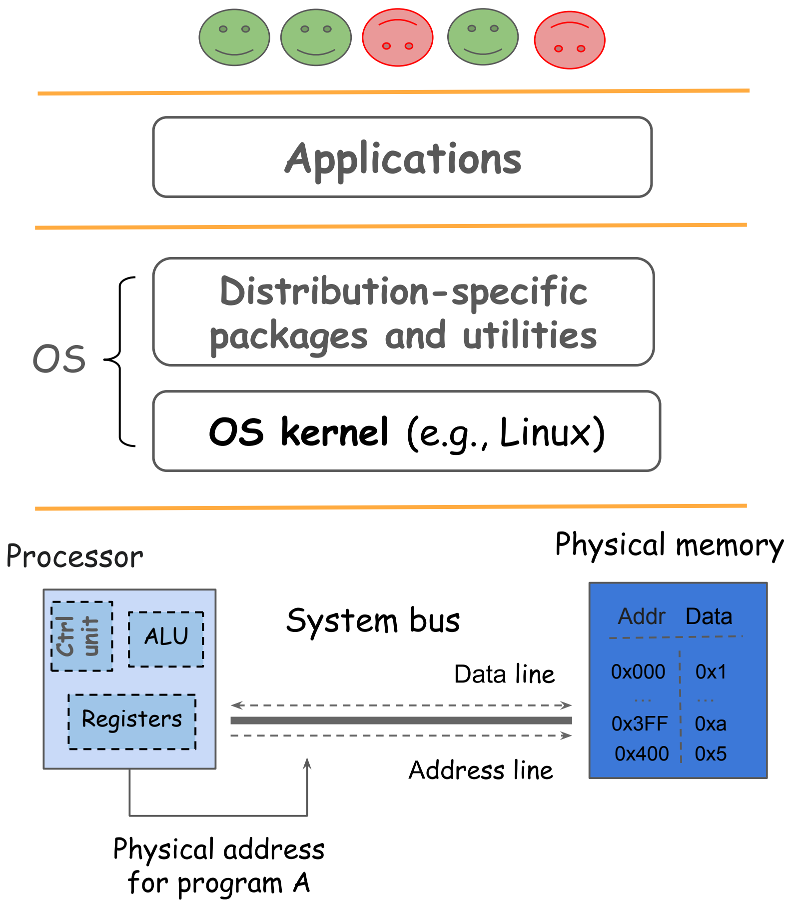


Designing an OS kernel

Desirable properties

1) Security, 2) Reliability, 3) Portability, 4) Fairness

From desirable properties to design principles



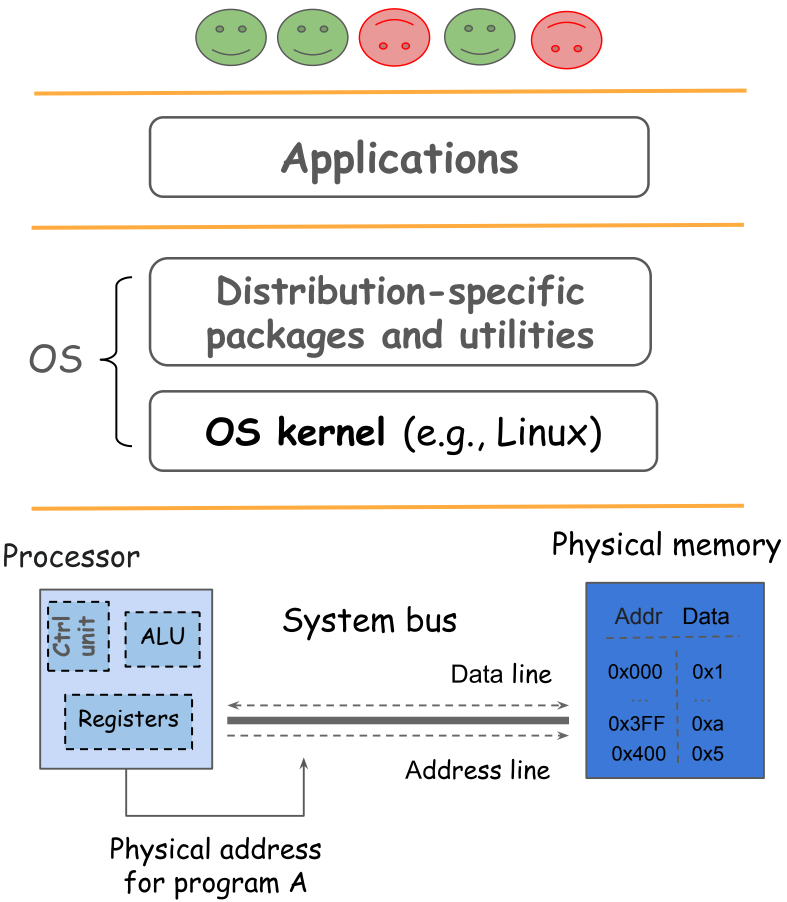
Designing an OS kernel

Desirable properties

1) Security, 2) Reliability, 3) Portability, 4) Fairness

From desirable properties to design principles

- <u>*Fault Isolation*</u>: Errors in one running program do not affect any other program



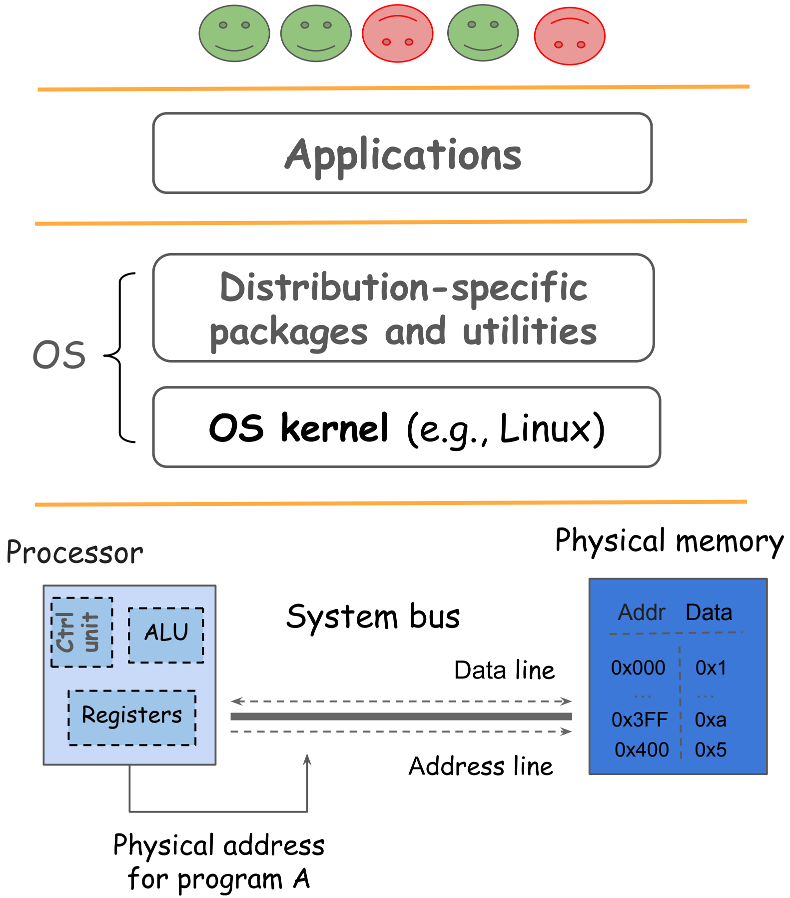
Designing an OS kernel

Desirable properties

1) Security, 2) Reliability, 3) Portability, 4) Fairness

From desirable properties to design principles

- ***Fault Isolation***: Errors in one running program do not affect any other program
- ***Principle of Least Privilege (PoPL)***: Any program has the minimum privileges necessary to perform its function



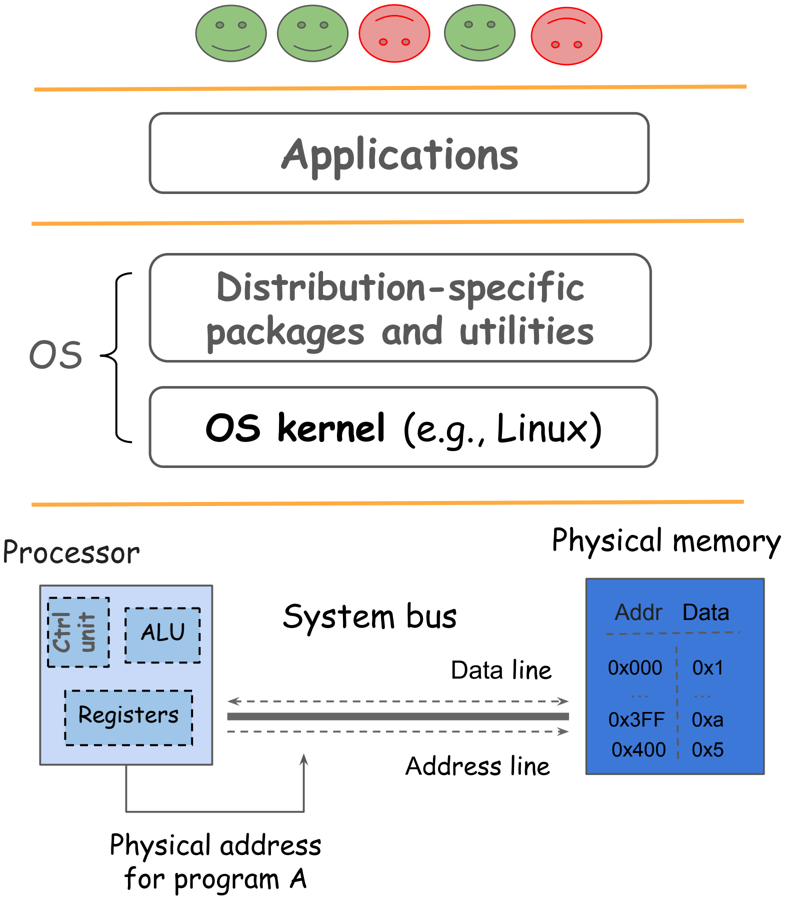
Designing an OS kernel

Desirable properties

1) Security**, **2) Reliability**, **3) Portability**, **4) Fairness

From desirable properties to design principles

- ***Fault Isolation***: Errors in one running program do not affect any other program
- ***Principle of Least Privilege (PoPL)***: Any program has the minimum privileges necessary to perform its function
- ***Preemption***: The OS is always able to take control of the processor, regardless of what programs are running

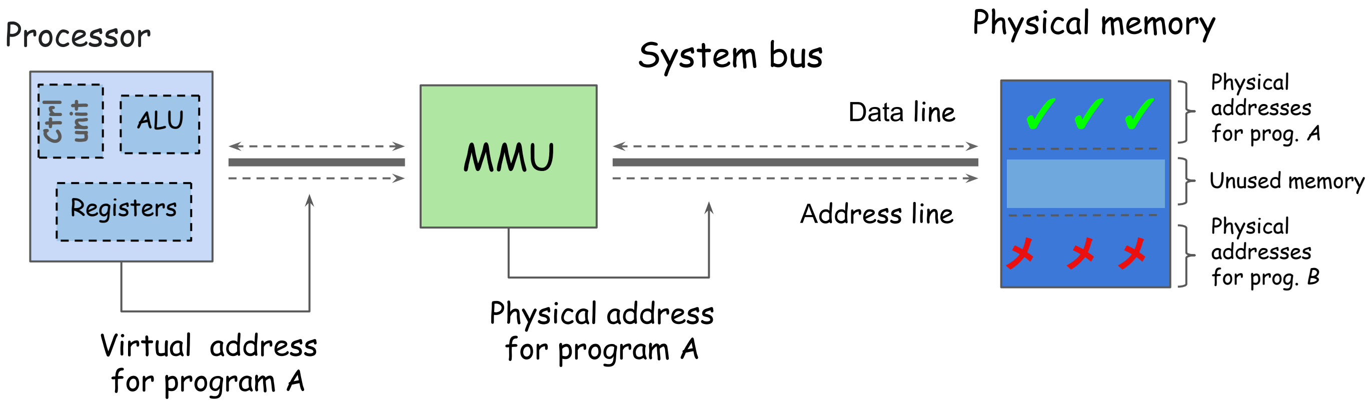


Implementing fault isolation

- In simple words: load/ store/ jmp instructions of a program cannot read, write, or jump to another program's memory

Implementing fault isolation

- In simple words: load/ store/ jmp instructions of a program cannot read, write, or jump to another program's memory

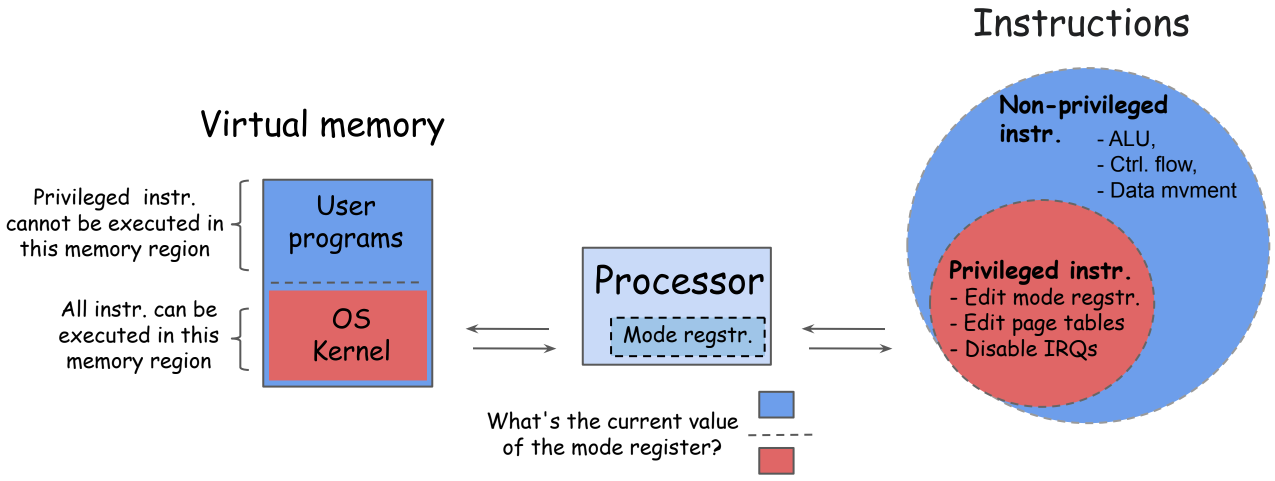


Implementing dual-mode execution

- In simple words: A trusted portion of the code (the OS) must have full control of the hardware

Implementing dual-mode execution

- In simple words: A trusted portion of the code (the OS) must have full control of the hardware

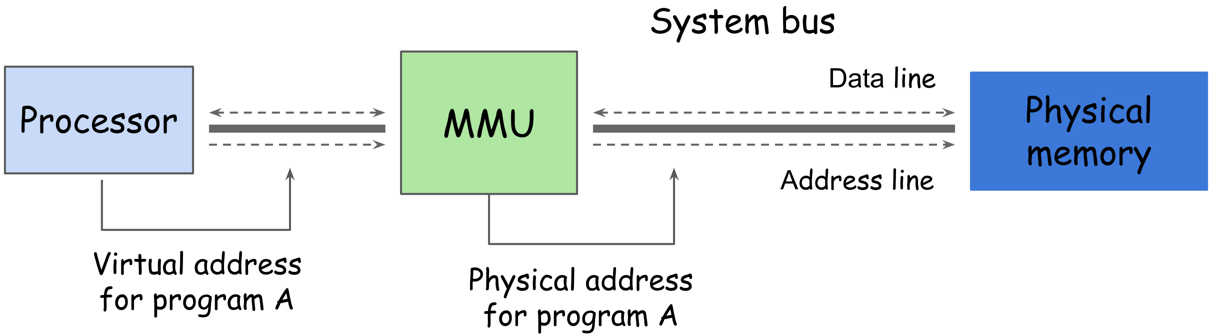


Implementing preemption

- In simple words: The OS should be able to periodically gain control of the processor regardless of what programs are executing

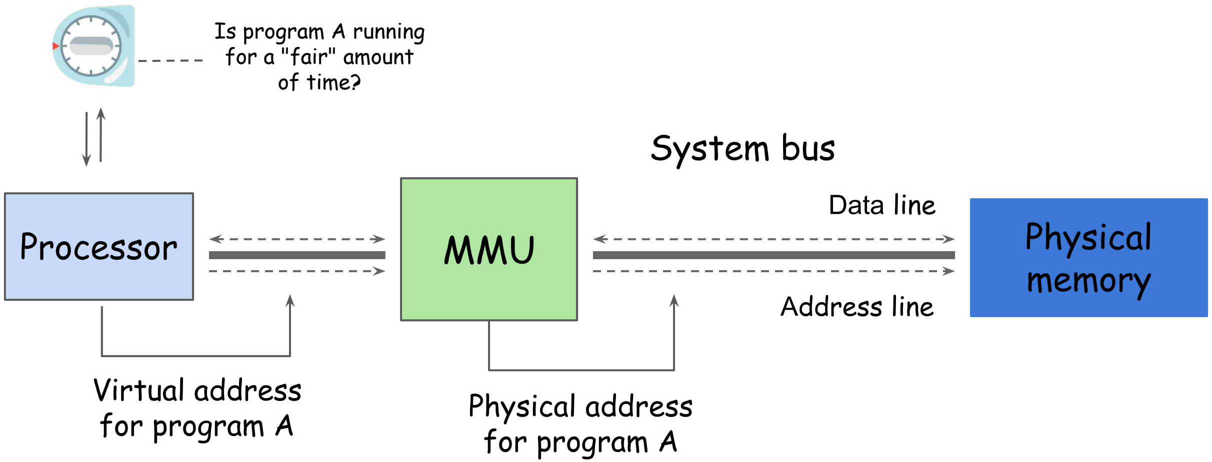
Implementing preemption

- In simple words: The OS should be able to periodically gain control of the processor regardless of what programs are executing

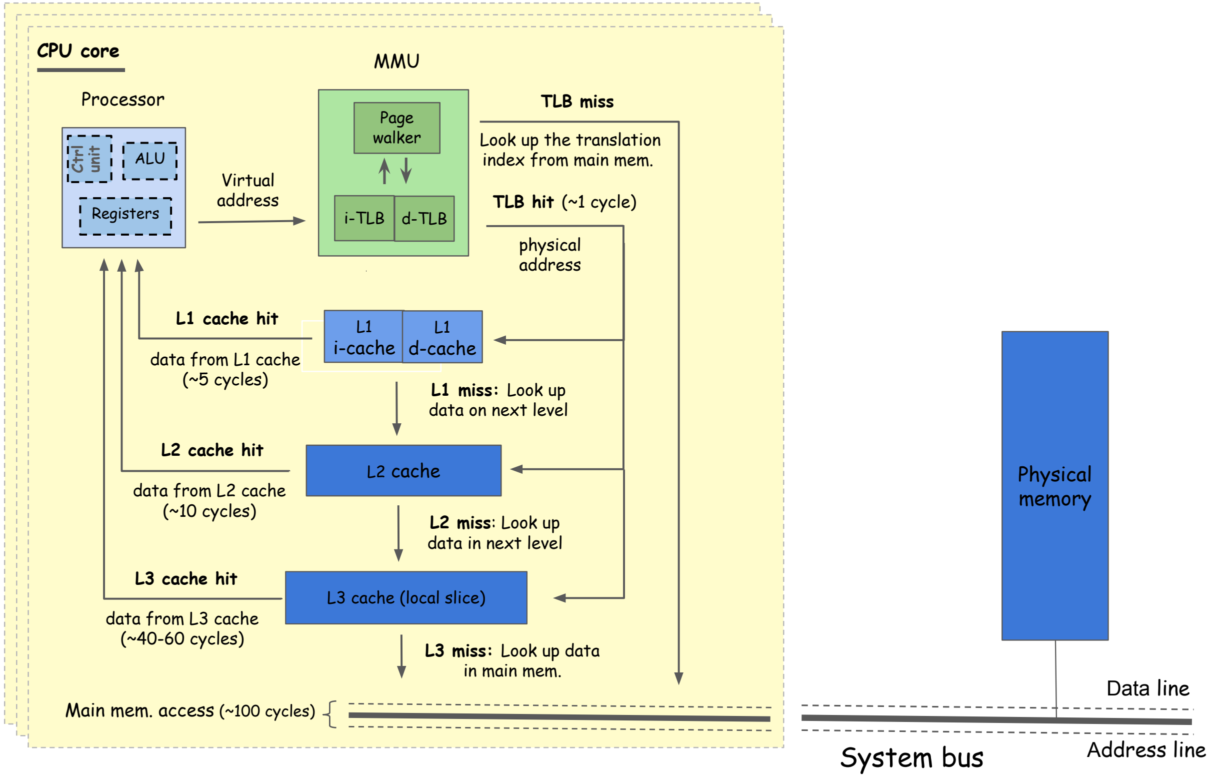


Implementing preemption

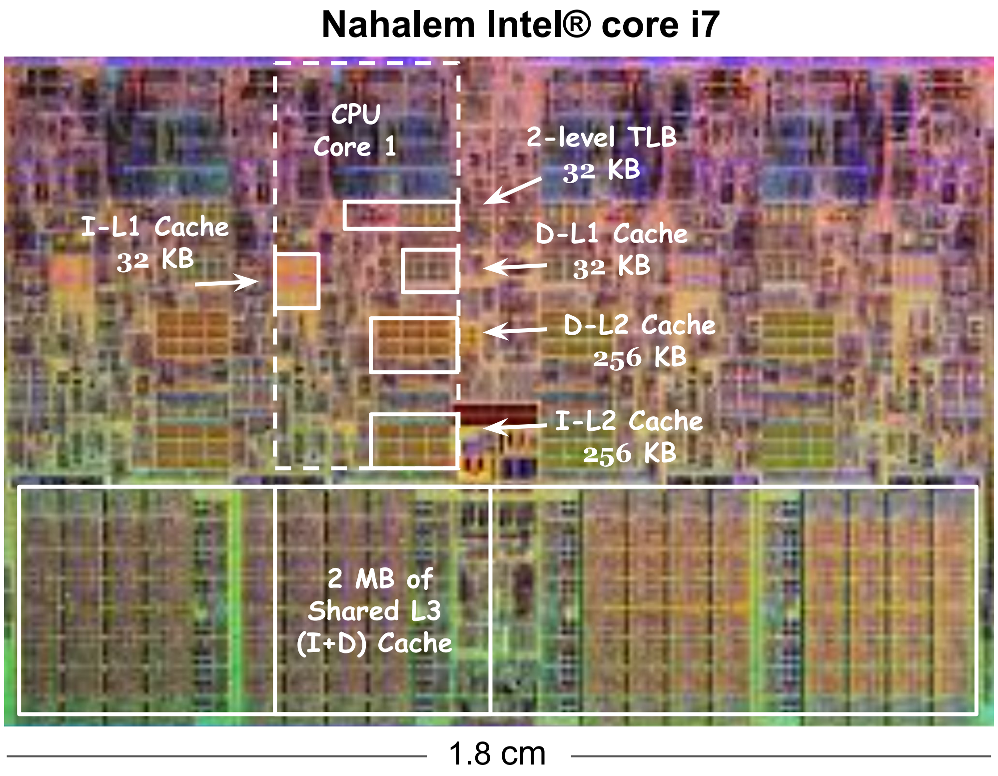
- In simple words: The OS should be able to periodically gain control of the processor regardless of what programs are executing



Hardware components leveraged by OS mechanisms

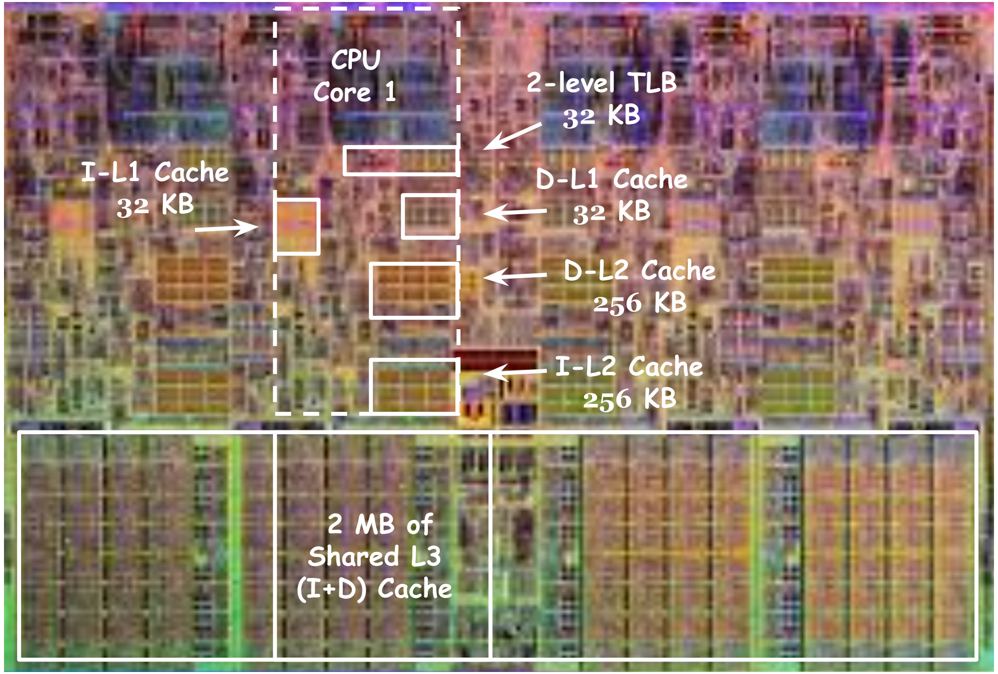


Reality check on hardware's scale



Reality check on hardware's scale

Nahalem Intel® core i7

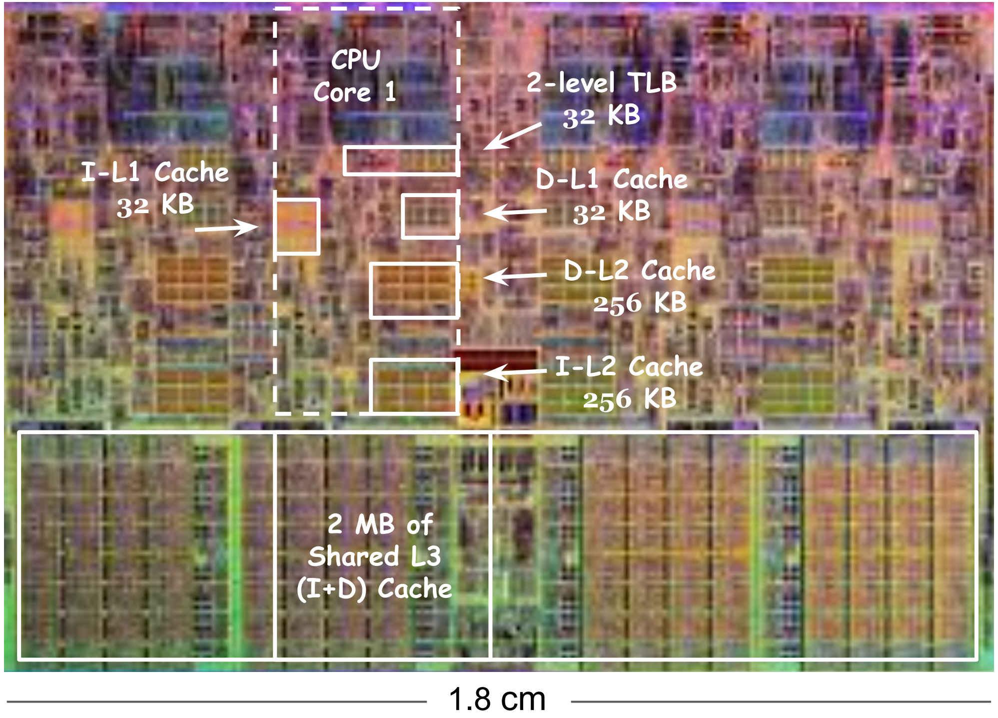


1.8 cm

Access times (1 clock cycle ~ 0.5ns)

Reality check on hardware's scale

Nahalem Intel® core i7

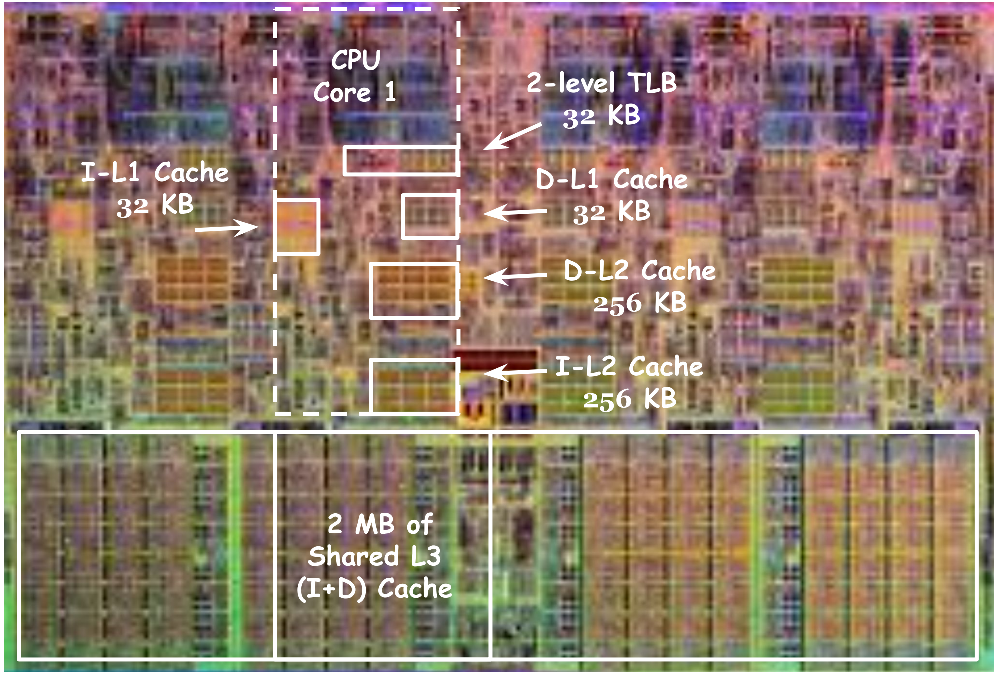


<u>Access times</u> (1 clock cycle ~ 0.5ns)

- L1 Cache (hit): ~ 5 clock cycles

Reality check on hardware's scale

Nahalem Intel® core i7



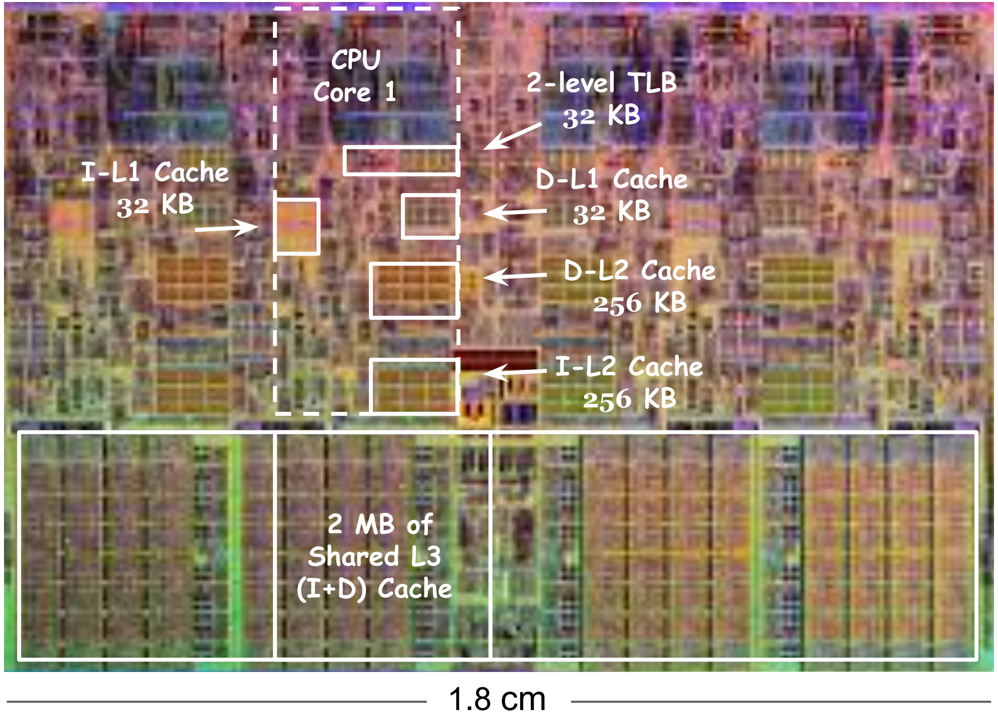
1.8 cm

<u>**Access times**</u> (1 clock cycle ~ 0.5ns)

- L1 Cache (hit): ~ 5 clock cycles
- L2 Cache (hit): ~10 clock cycles

Reality check on hardware's scale

Nahalem Intel® core i7

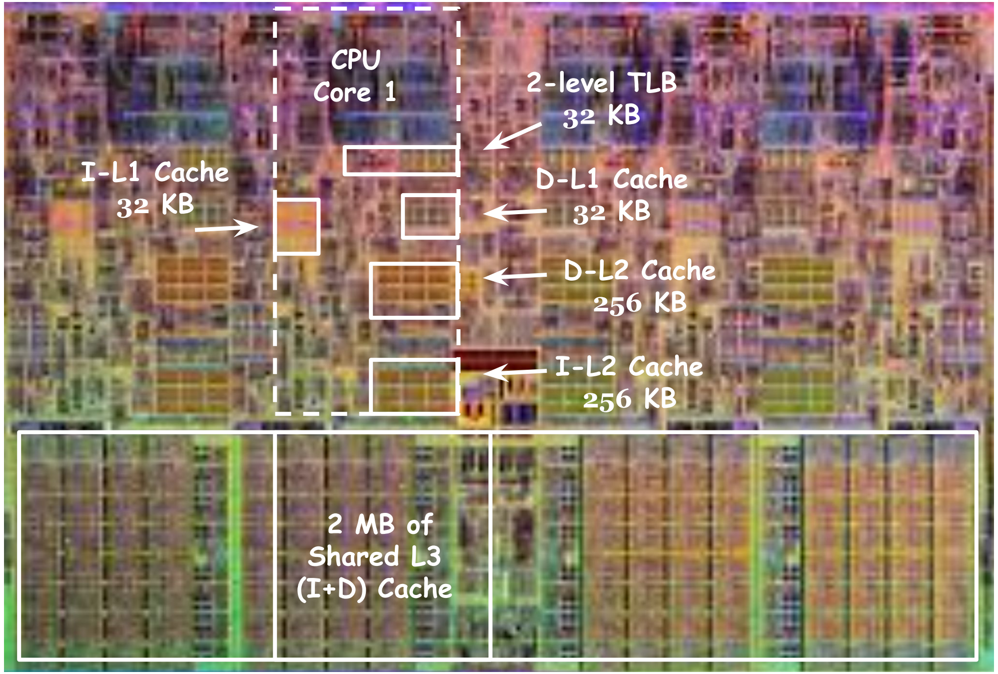


<u>**Access times**</u> (1 clock cycle ~ 0.5ns)

- L1 Cache (hit): ~ 5 clock cycles
- L2 Cache (hit): ~10 clock cycles
- L3 Cache (hit): ~40-60 clock cycles

Reality check on hardware's scale

Nahalem Intel® core i7



1.8 cm

Access times (1 clock cycle ~ 0.5ns)

- L1 Cache (hit): ~ 5 clock cycles
- L2 Cache (hit): ~10 clock cycles
- L3 Cache (hit): ~40-60 clock cycles
- L3 Cache miss (MM access): ~100 clock cycles