



CSE 451: Operating Systems, 26au

L1: OS Overview

Instructor: Naomi Alterman

TAs:

Jonathan Trinh (Head)

Mahir Emran

Ted Guan

Soham Raut

Jai Rungrachakanont

Sagar Saby

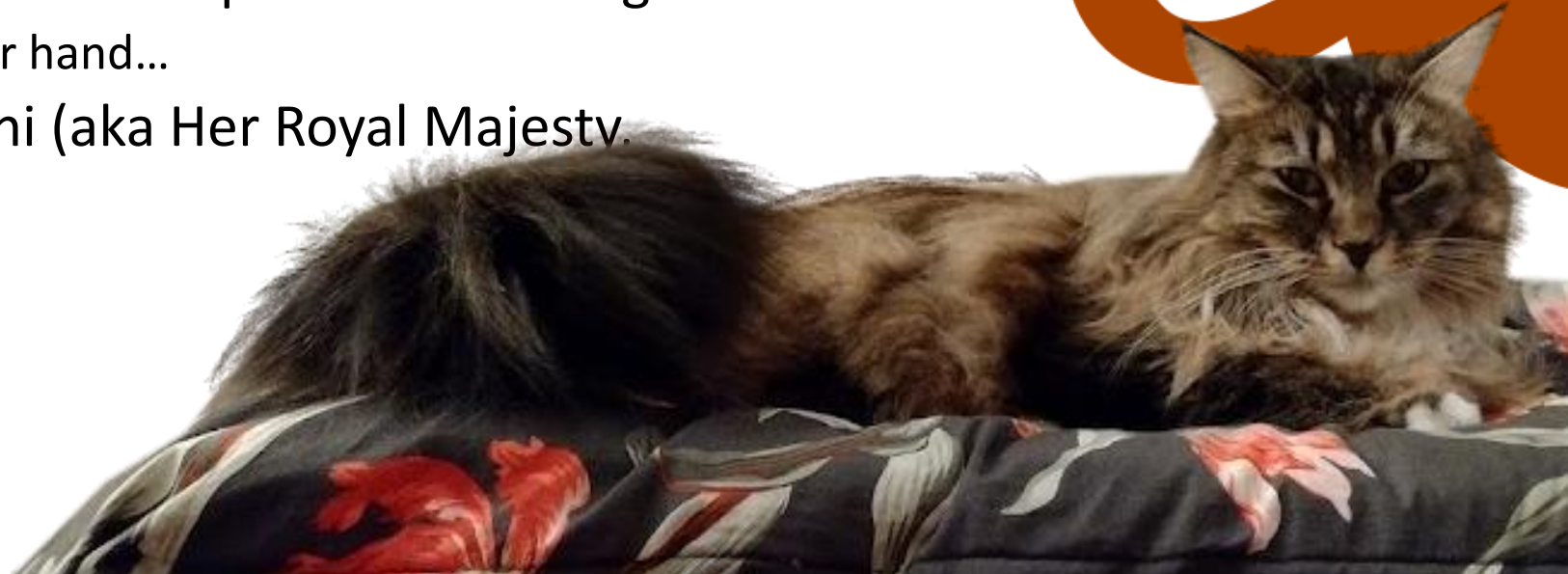
Andrew Wood

Adapted from slides by Tom Anderson

New class, who dis?

Oh hi

- Professor Naomi
 - Electrical engineer by training
 - Bopped around Silicon Valley hacking on everything from internet backbone routing chips to OS kernels to LIDAR firmware to mobile graphics libraries
 - Discovered in industry that computers are boring
 - But *people* on the other hand...
 - Proud cat mom to Danni (aka Her Royal Majesty Queen Baby)



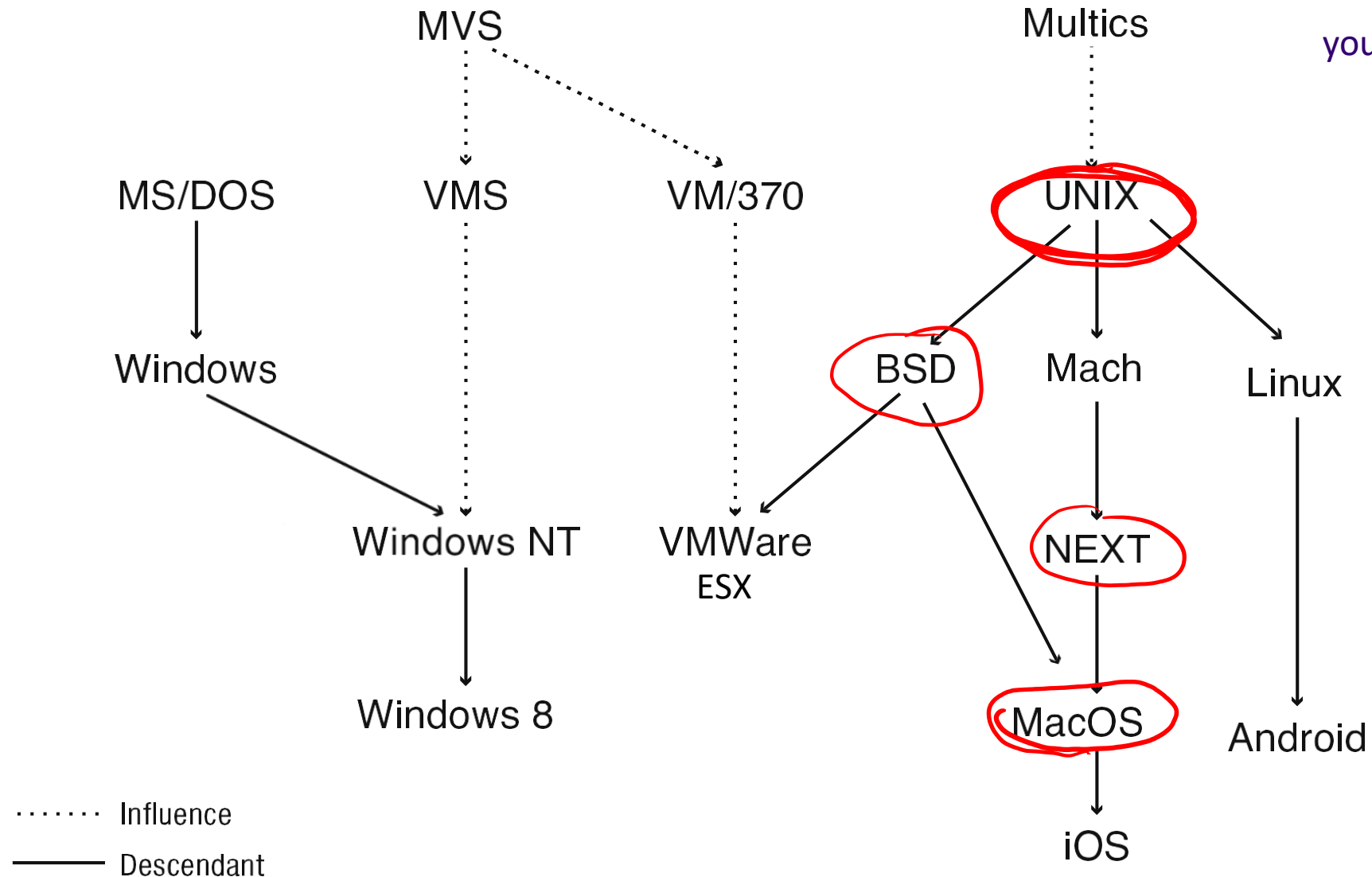
OS kernels, you say?



OS Family Tree * **

* grossly over-simplified

** we aren't going to test you on this

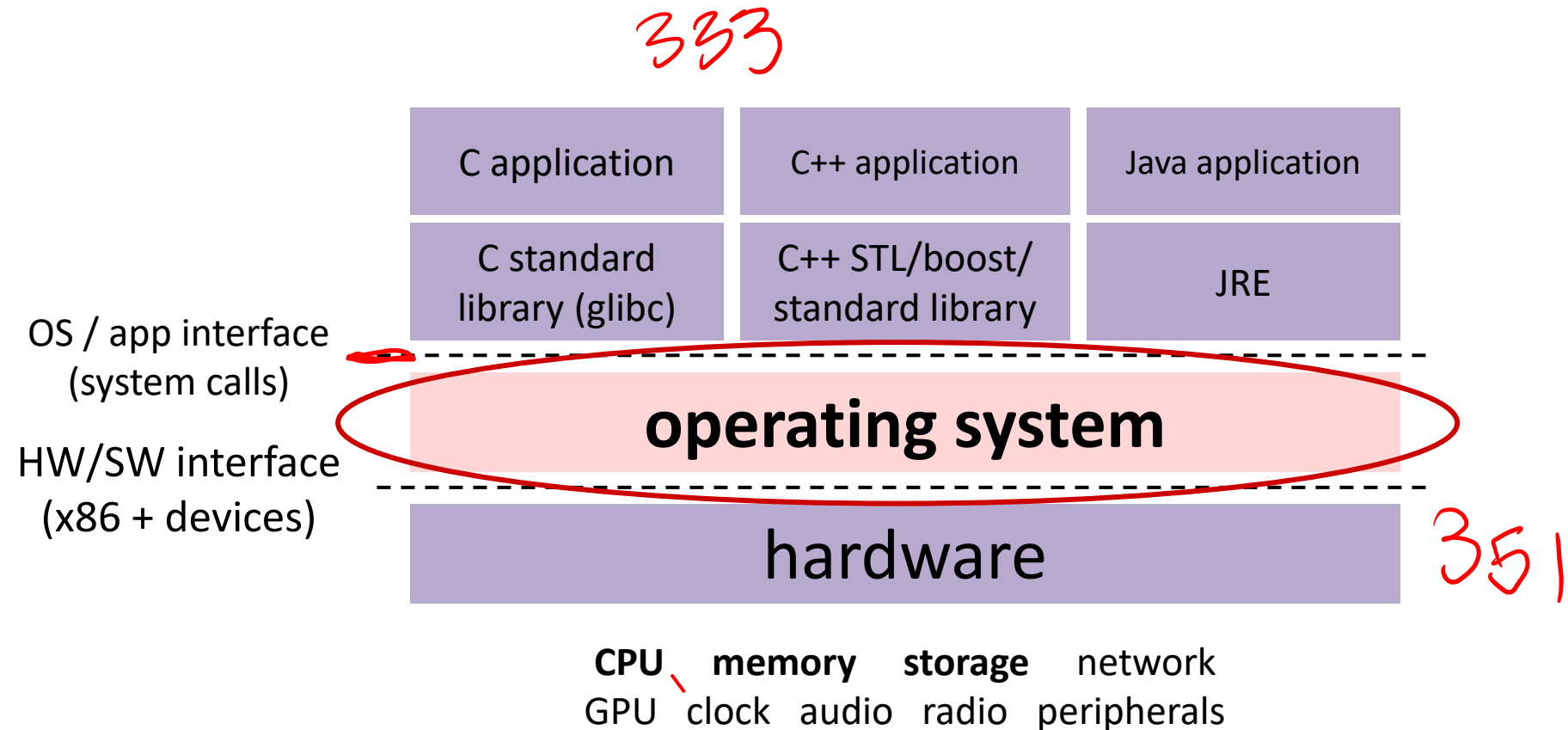


Computer Performance Over Time

	1983	2026	Factor
CPU Speed (MIPS/s)	1	5K-10K	5K-10K
CPUs per server	1	128-256	128-256
Processor speed \$/MIPS	\$100K	\$0.005	20M
DRAM capacity (MB/\$)	0.002	20K-40K	10-20M
Disk capacity (GB/\$)	0.003	200K-500K	70-150M
Network	10Mbps (shared)	800Gbps (switched)	8M
Ratio users per CPU	100:1	1:20+	1K+

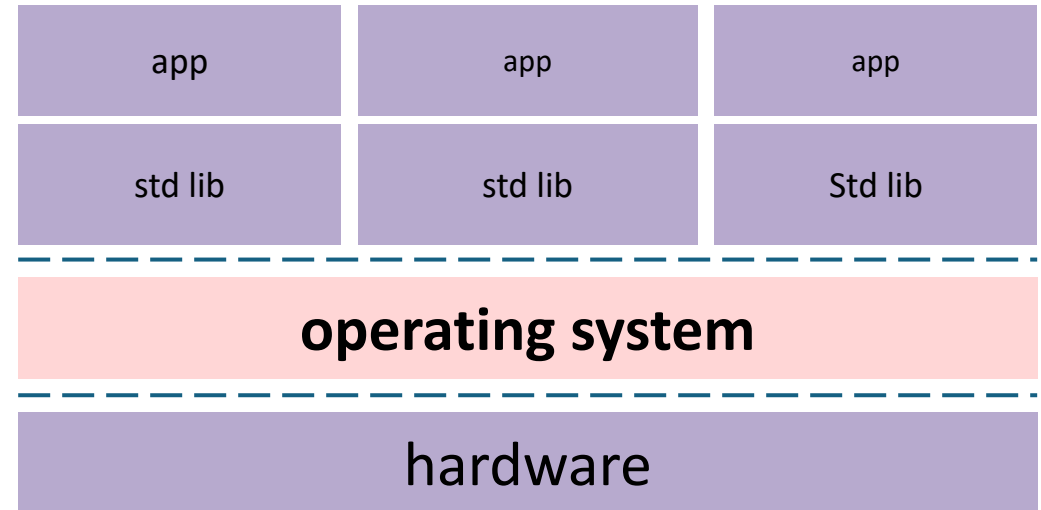
And let's hear it for your TAs!

Course Motivation



Operating system?

- Yes m'am!
- Just a big ol' C program
 - ✧ ...that talks to hardware
 - ✧ ...and keeps other software in line
- For any area of an OS (CPU, memory, storage, graphics, network, ...):
 - What is the **hardware interface**? ✧
 - What is the **application interface**? ✧



Operating System Roles



Referee:

- Isolation of different applications from each other
- Resource allocation when there is contention
- Communication between users, applications (pipes, shared memory, files)



Illusionist:

- Each application appears to have the entire machine to itself, without limitation
- Thread count not limited by # of CPU cores
- Reliable storage, reliable network transport



Glue:

- User interface libraries
- Centralized system configuration
- Clipboard

OS principles show up everywhere

- Web browsers: web pages embed programs
 - how do we isolate these programs from one another?
 - How do we protect the browser from attacks launched by these programs?
- Data centers: millions of CPUs, disks
 - Shared resources at the large scale; how do we arbitrate contention?
 - How do we isolate different users from one another, even if some are malicious?
- Cloud services (eg, Amazon's elastic block store)
 - How do we ensure no user monopolizes service or steals other user data?
- Databases: shared data for many different users
 - How do we keep data from different users private?
- AI infrastructure
 - Training and serving systems: how do we efficiently orchestrate training and inference systems?

Why is OS programming hard?

- Adversarial users
 - Every application is a potential source of attack
 - Can use entire capability of machine to launch attack
- OS code needs to be perfect
 - ★ Bug in one application affects that application
 - Bug in kernel affects all applications
 - Security bug in kernel compromises all user data
 - Often performance critical
- The curse of abstractions
 - OS provides convenient abstractions to applications
 - ★ ...so the OS can't use those abstractions itself



Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

85% complete

For more information about this issue and possible fixes, visit our website.

If you call a support person, give them this info:

Stop code: 0x00000000



Check-In Information

की जानकारी

DELTA

GMR

चेक इन एव उड़ान



10:55	11:30	AI 621	रायपुर
10:58	11:30	AI 621R	रायपुर
11:00	11:00	UK 215	वीरवार
11:40	11:40	UK 643	वीरवार
12:00		6E 5303	बाराबात
12:20	12:35	6E 5009	हैदराबाद
12:30	13:35	6E 5154	चेन्नई
12:45	14:05	6E 5398	बंगलूर
12:45	15:30	GI 833	बंगलूर - कोलकोता
12:55	14:15	6E 5257	बंगलूर
13:05	13:05	AI 857	पुणे
13:15	13:15	AI 388	दो-बी-निम्न
13:15	13:45	UK 753	रांची
13:25	13:10	UK 809	बंगलूर

GMR की जानकारी या गेटों के लिए 'www.airsewa.gov.in' पर जाएं।

14:30	14:35	AI 357	कोलकोता
14:35	14:35	AI 352	बिहार
14:35	14:35	UK 501	पुणे
14:35	14:35	6E 5373	बाराबात
14:35	14:35	AI 538	दो-बी-निम्न
14:35	14:35	GI 841	बंगलूर
14:35	14:35	AI 617	हैदराबाद
14:35	14:35	UK 554	हैदराबाद
14:35	14:35	GI 758	रांची
14:35	14:35	6E 8150	बंगलूर
14:35	14:35	AI 433	बंगलूर
14:35	14:35	AI 457	बंगलूर
14:35	14:35	AI 857	पुणे
14:35	14:35	UK 753	रांची
14:35	14:35	UK 809	बंगलूर

GMR 124-4797300 / feedback.igairport@airsewa.gov.in

15:10	15:10	6E 5095	रांची
15:10	15:10	AI 547	बंगलूर
15:10	15:10	UK 017	बंगलूर
15:10	15:10	UK 721	बंगलूर
15:10	15:10	AI 801	बंगलूर
15:10	15:10	UK 548	बंगलूर
15:10	15:10	6E 5325	हैदराबाद
15:10	15:10	UK 925	पुणे
15:10	15:10	6E 5332	बंगलूर
15:10	15:10	AI 807	बंगलूर
15:10	15:10	AI 815	बंगलूर
15:10	15:10	UK 753	रांची
15:10	15:10	UK 1733	बंगलूर
15:10	15:10	AI 809	बंगलूर

GMR 124-4797300 / + 0124-6838410 / feedback.igairport@airsewa.gov.in

15:55	16:30	AI 549	बंगलूर
16:00	16:00	AI 853	बंगलूर
16:00	16:00	WY 246	बंगलूर
16:05	16:05	6E 5841	बंगलूर
16:05	16:05	UK 017	बंगलूर
16:10	16:10	IG 776	रांची
16:15	16:15	AI 537	बंगलूर
16:15	16:15	UK 517	बंगलूर
16:20	16:20	AI 835	बंगलूर
16:20	16:20	HA 218	बंगलूर
16:20	17:40	GI 8171	बंगलूर
16:20	16:20	UK 805	बंगलूर
16:30	17:20	6E 5066	बंगलूर
16:30	16:20	6E 5843	बंगलूर

GMR 124-4797300 / + 0124-6838410 / feedback.igairport@airsewa.gov.in

FROM DELHI AIRPORT

APPAREL | LEATHER | JEWELLERY | PICKED FOOD | MISC. JEWELLERY

SAME AS

GRAB IT ALL FROM DELHI AIRPORT



Soft Error

Return

Date	(2020:03:06 - 14:41:21)
Process	aulapp.exe
ThreadID	0x33519af2
Exception Address	0x00000030
Exception Code	0xc0000005
Instruction Pointer	0x0383141e



1 / 3

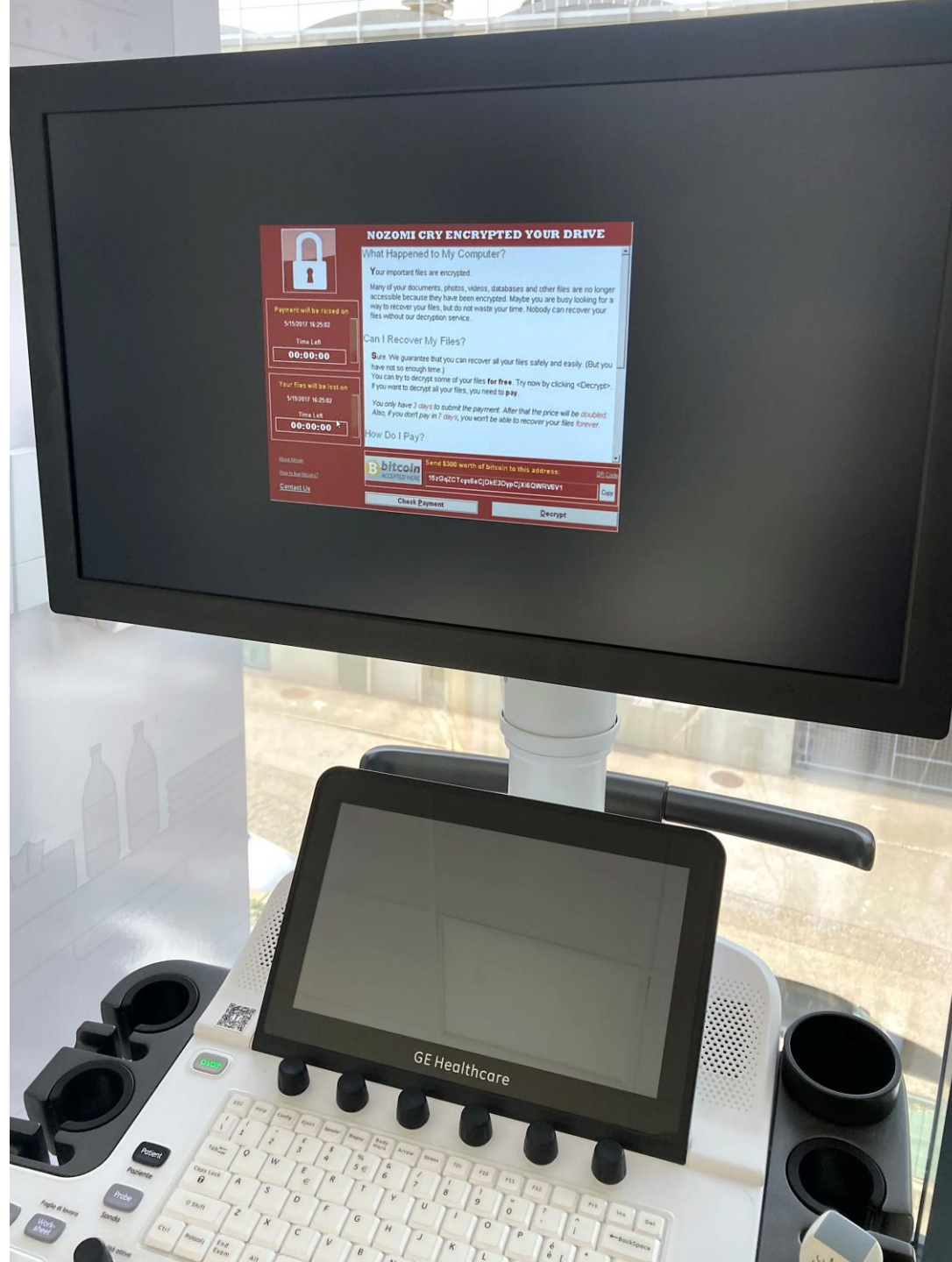


Clear

Save

PASSENGER
AIRBAG





NOZOMI CRY ENCRYPTED YOUR DRIVE

What Happened to My Computer?

Your important files are encrypted.

Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?

Sure. We guarantee that you can recover all your files safely and easily. (But you have not so enough time.)

You can try to decrypt some of your files **for free**. Try now by clicking <Decrypt>. If you want to decrypt all your files, you need to **pay**.

You only have 3 days to submit the payment. After that the price will be doubled. Also, if you don't pay in 7 days, you won't be able to recover your files forever.

How Do I Pay?

Send \$300 worth of bitcoin to this address:

15d2qCtYp8aCjDh3DypCjK6QWRVvY1

[Check Payment](#) [Decrypt](#)





Por favor ligue para 220 934 085
para ativar este sistema.

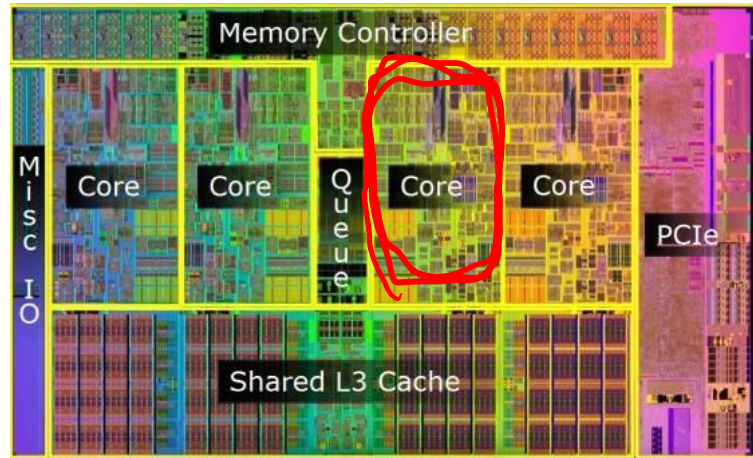
Por favor indique a seguinte referência:

xarevisio



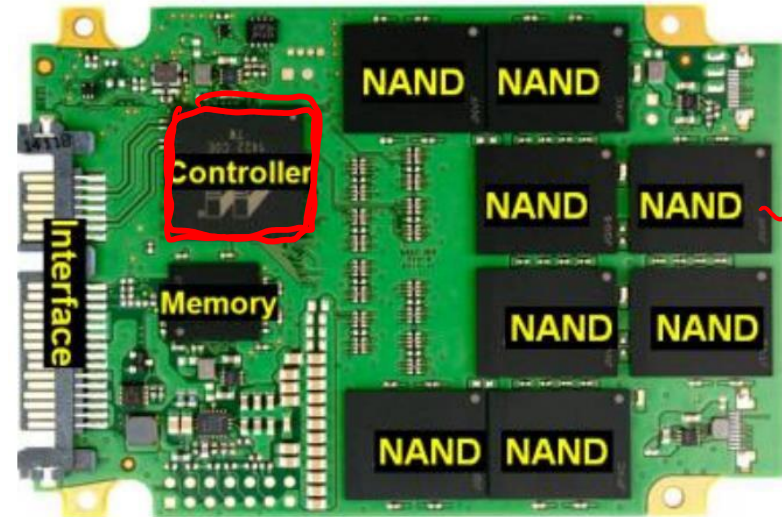
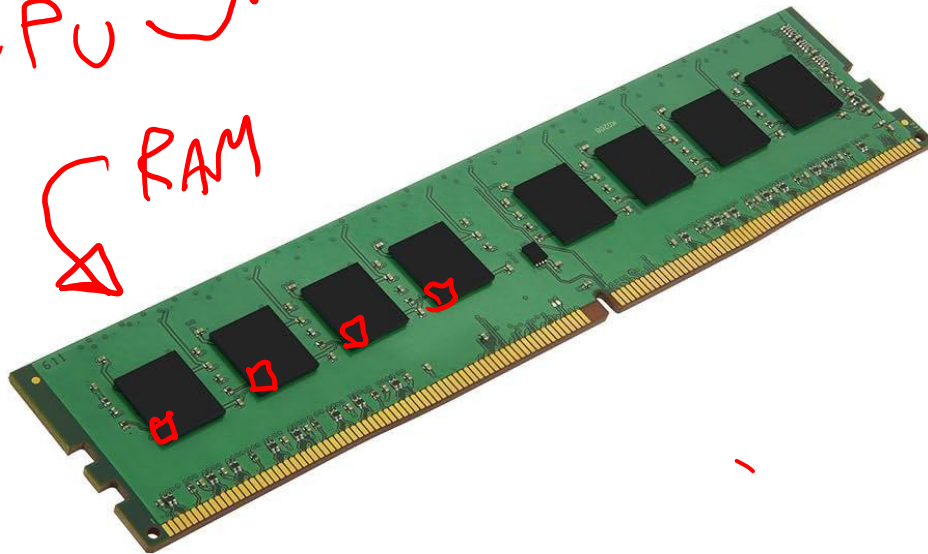
What is computer hardware?

SSD ↗

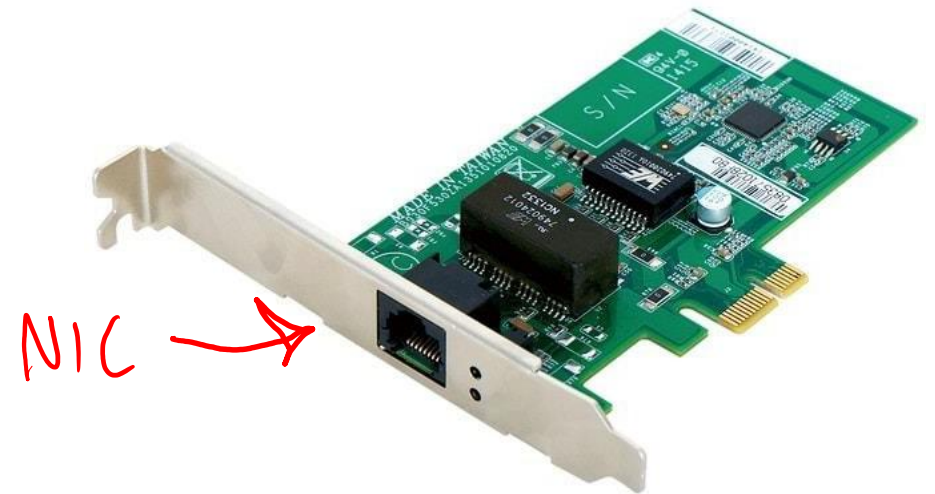


CPU ↗

RAM ↘



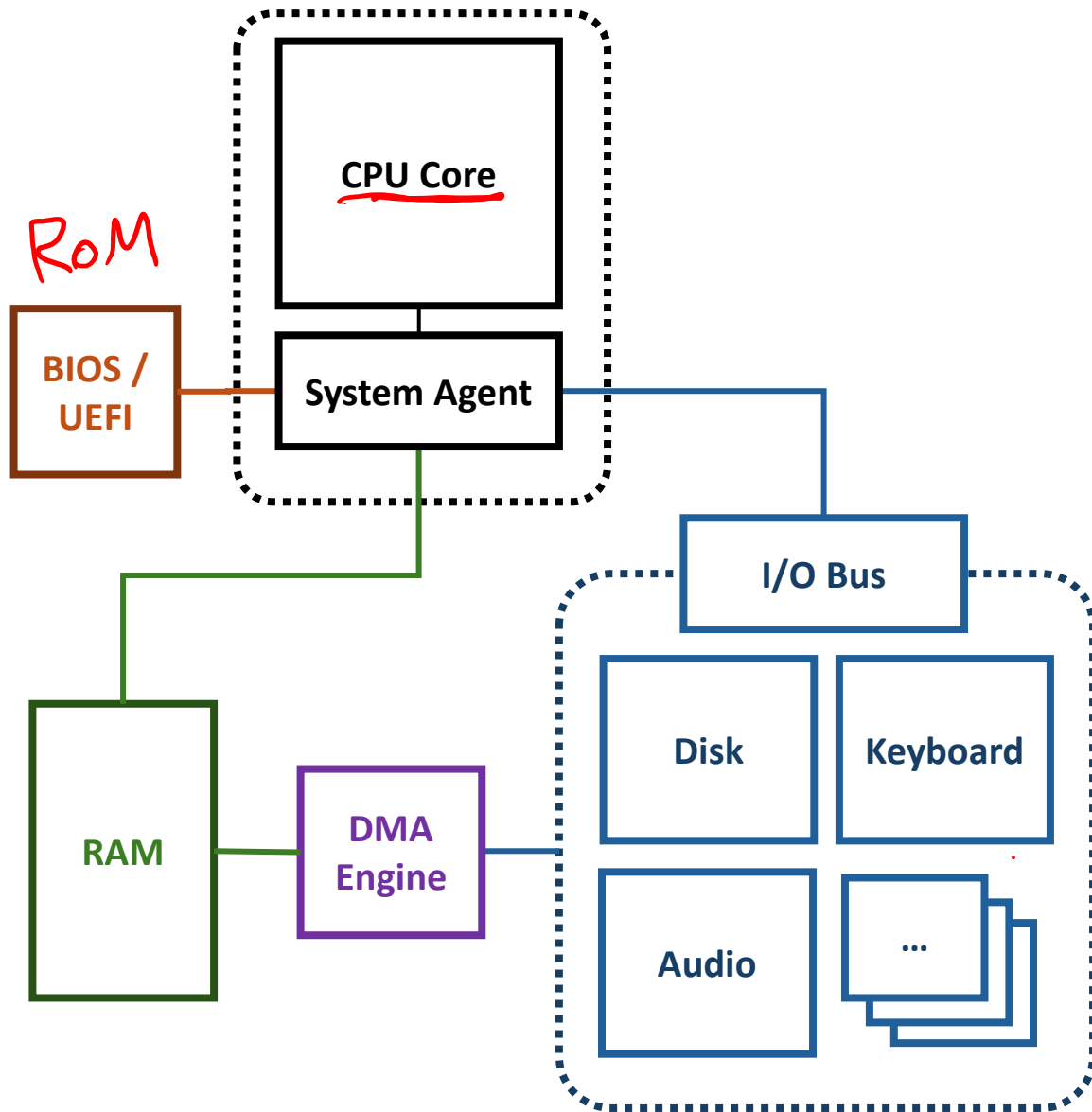
Flash ~



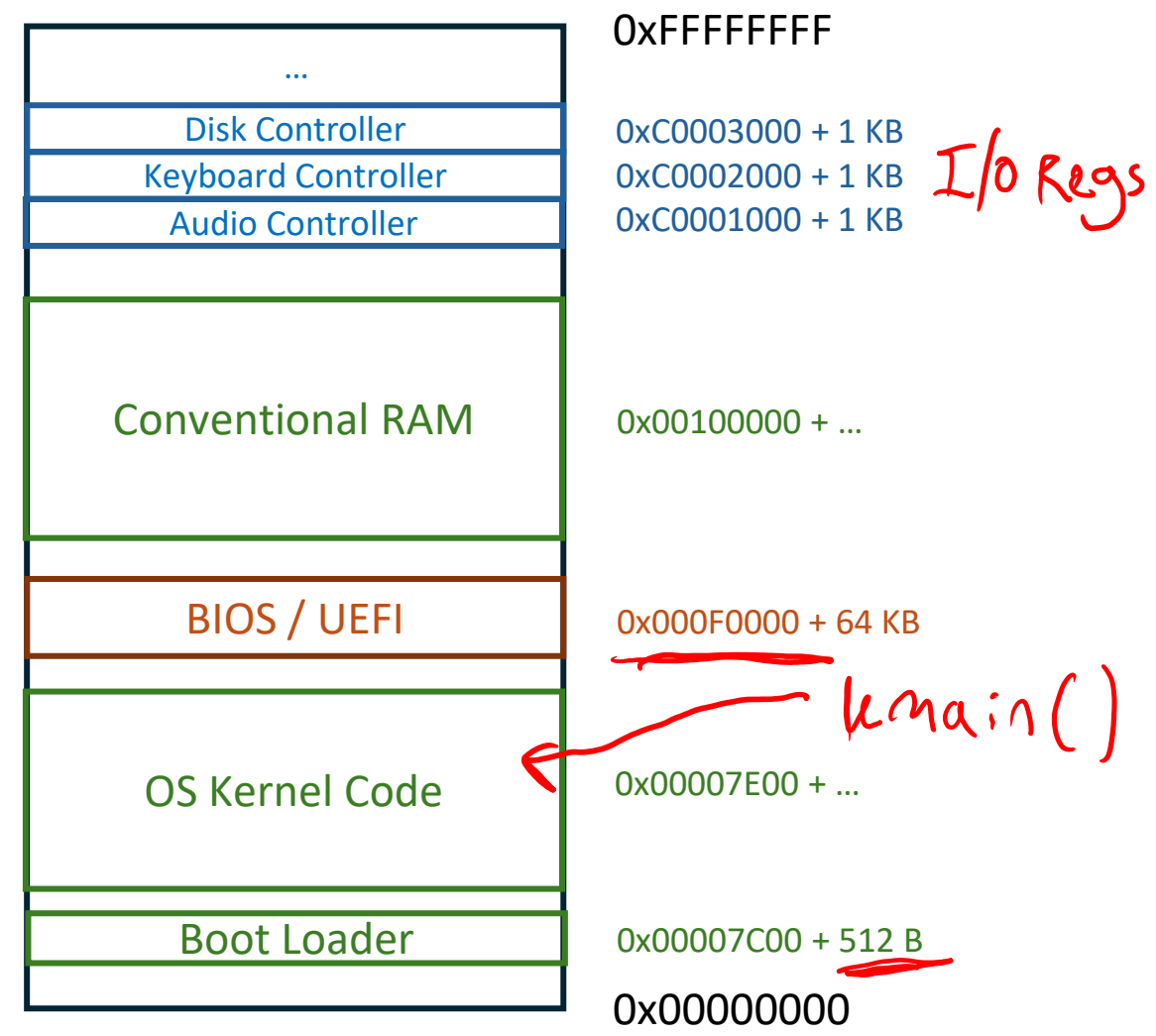
NIC ↗

What is computer hardware?

CSE 369



Physical Address Space

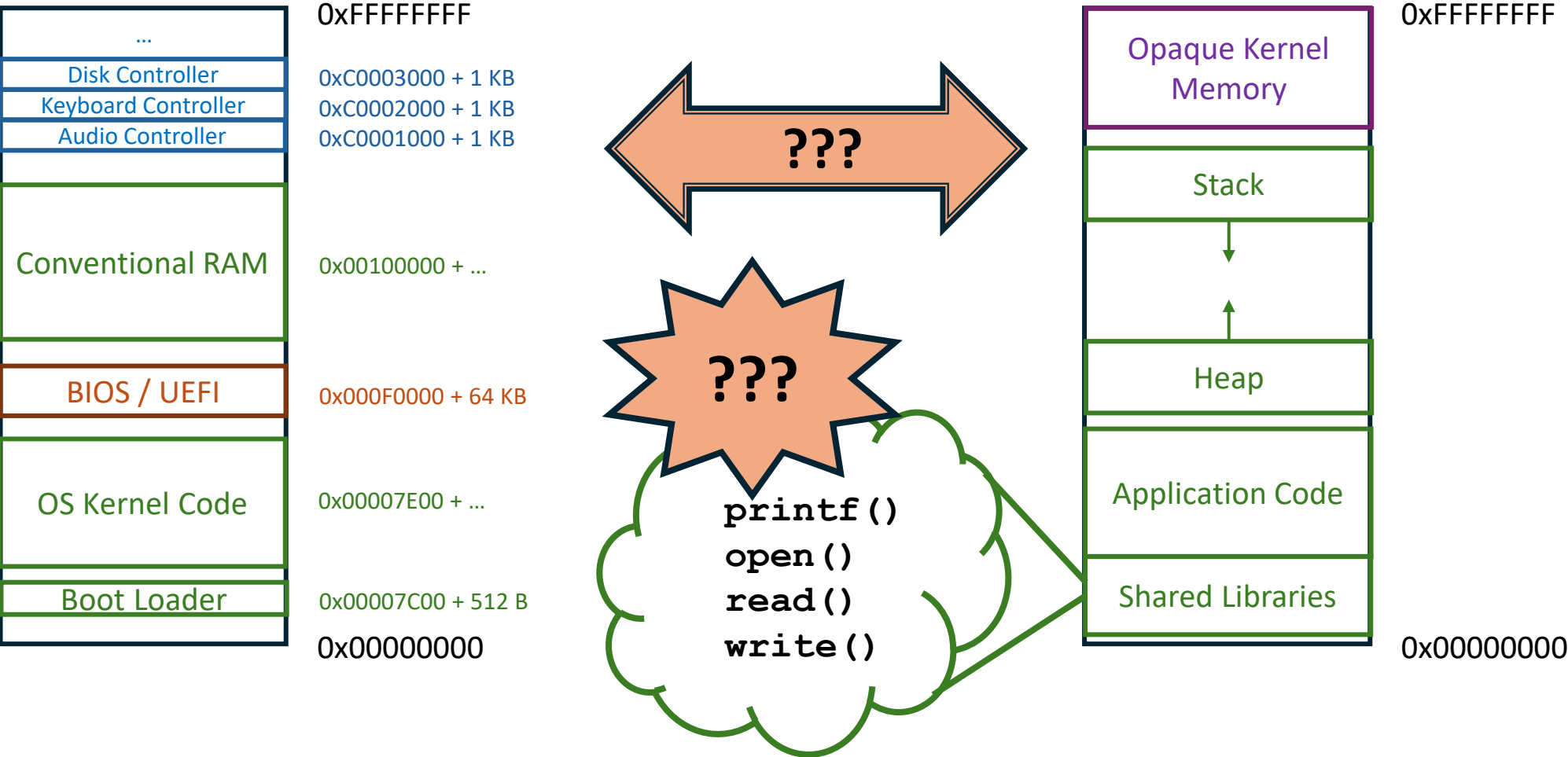


Oh by the way



- Which of these details are specific to x86, and which are universal?
- Which of these details are there because of backwards compatibility?

Physical Address Space



Class Administrivia

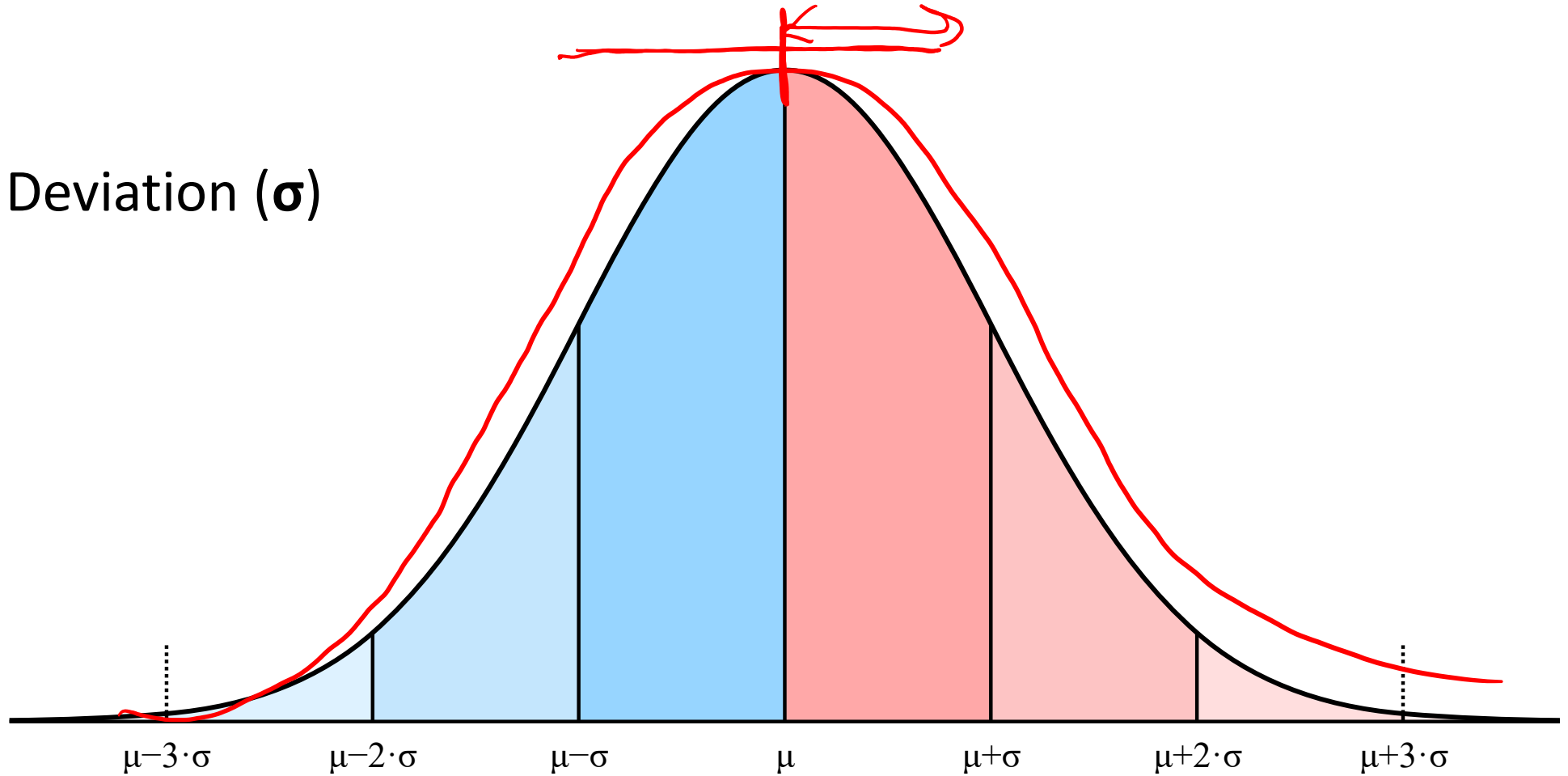
Grading

- Grade breakdown
 - 🔑 40% Labs
 - 🔑 15% Problem Sets
 - 🔑 5% Design Docs
 - 🔑 15% Midterm
 - 🔑 25% Final
- Need to take both exams to pass

How to interpret exam scores

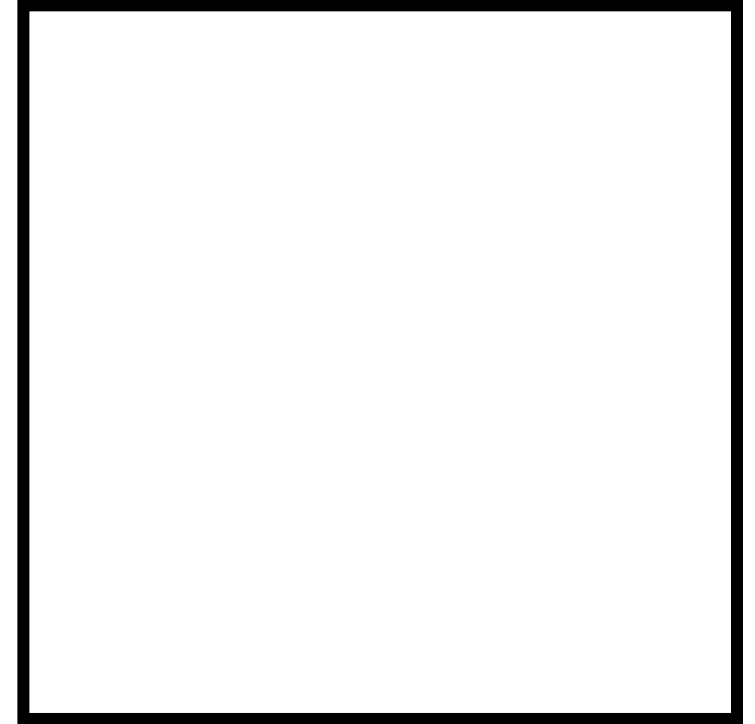
Mean (μ)

Standard Deviation (σ)



Course Project: xk

- Build an operating system :O !
 - That can boot on real hardware!!
 - Wowowow!!!
 - We give you some basic starter code
- Four assignments that build on each other:
 - Syscalls
 - Multi-processing
 - Virtual memory
 - Filesystems



Project logistics

- **Must work in groups of 2**

- Fill out one form per group:

➔ <https://forms.gle/BmhR28c5WsUFH39B7>

- Dedicated time in first section (tomorrow) for finding a partner
 - Form due by end-of-day Friday

➔ • Each group will get a git repo

- Contains xk starter code and “getting started” docs

➔ • Assignments released as markdown files in repo

➔ • Work turned in as a zip uploaded to Gradescope



Class AI Policy



“AI as a Reference Tool”

You may use AI for general course questions and ungraded work, but not for graded assignments, quizzes, exams, or any deliverable that counts toward your grade.

- Consider first whether you have the time to ask your TAs, the professor, or peers for help and then use AI for subsequent clarification.
- Use AI as a tutor, not a substitute for your own work.
- Ask for explanations, hints, or examples rather than full solutions.
- Verify AI output against course materials and official documentation.
- If AI starts doing the work for you, stop and switch to a more limited prompt.

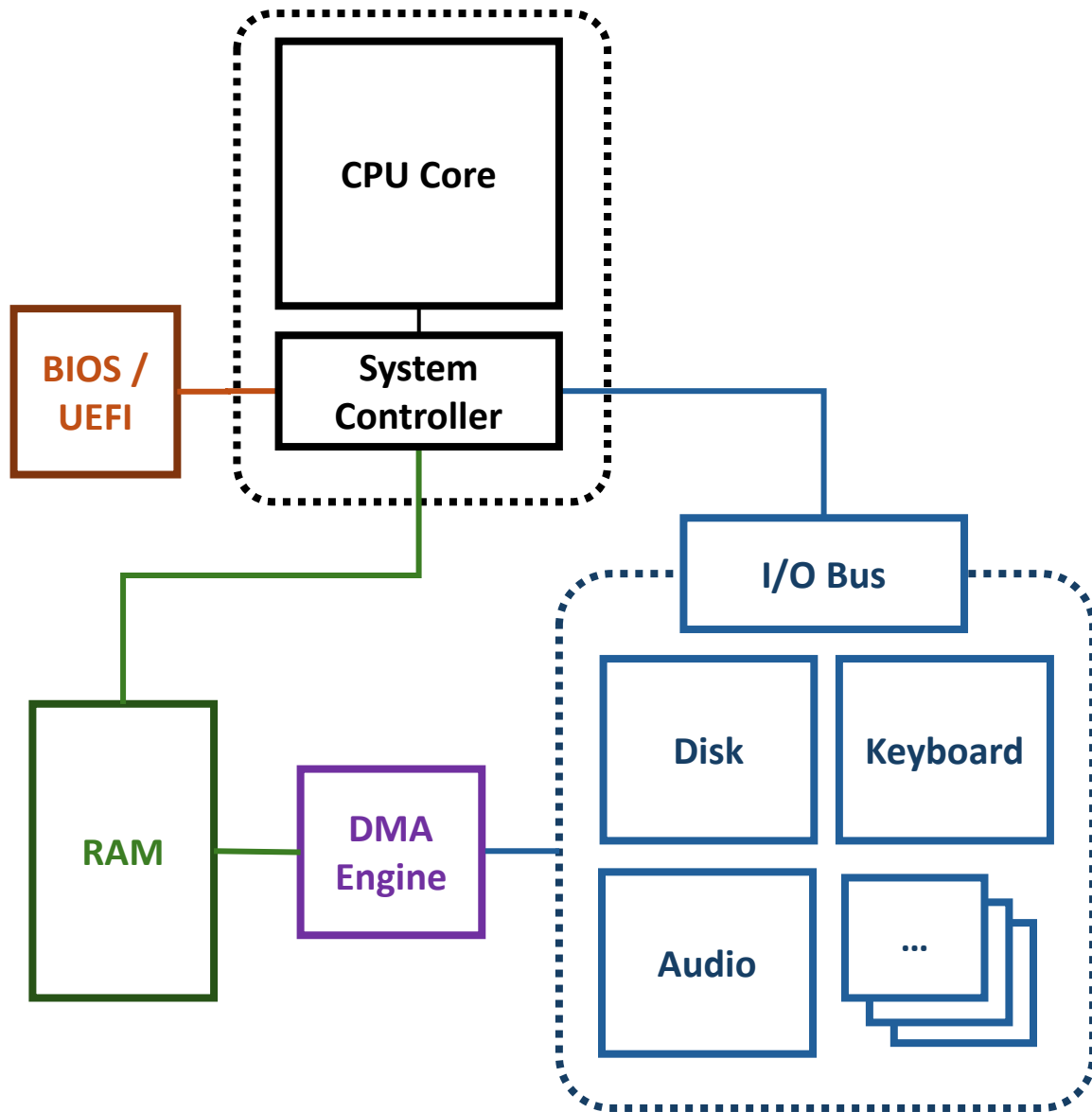
Academic Honesty

- Don't share code or problem solutions outside of your group
- Don't use AI to write code or design docs
- Don't use AI to help brainstorm the content of your design docs
 - The process *behind* your design doc is the actually important part; the doc itself just the "proof of work"
- Both group members are responsible for upholding academic honesty

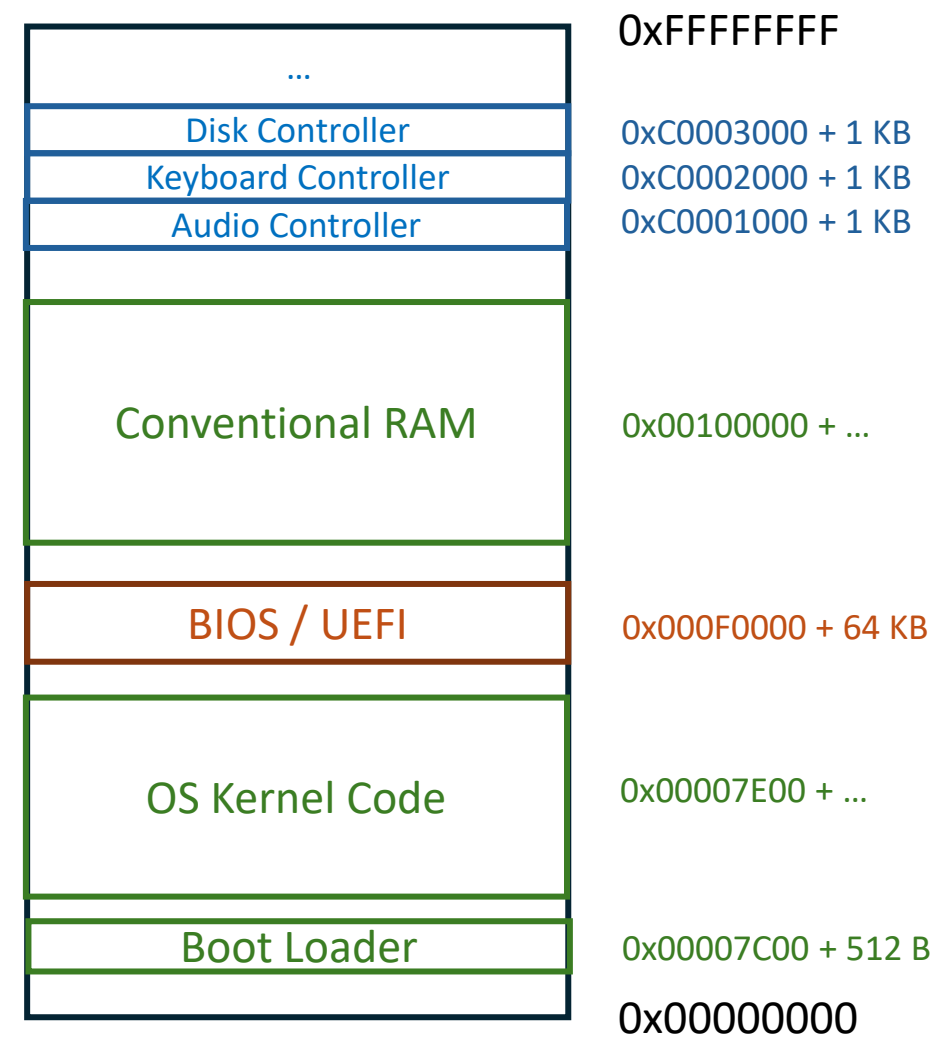
Class concluded here on 9/30
and will resume on 10/2



Device I/O



Physical Address Space



Device I/O

- OS kernel needs to communicate with physical devices
 - Network, disk, video, USB, keyboard, mouse, ...
 - Devices operate asynchronously from the CPU
 - Have their own lil processors (“**microcontrollers**”) running their own lil code (“**firmware**”)
- How does the OS communicate with this firmware?
 - Some addresses in the CPU’s physical address space are effectively “wormholed” to these external chips
 - Writing bytes -> sending commands to the firmware
 - Reading bytes -> getting results back from those commands
 - In C, this looks like **setting a pointer** to a SUPER SPECIFIC but ARBITRARY LOOKING value, and then **dereferencing it**

Waiting for I/O to finish

- Synchronous I/O: **Polling**

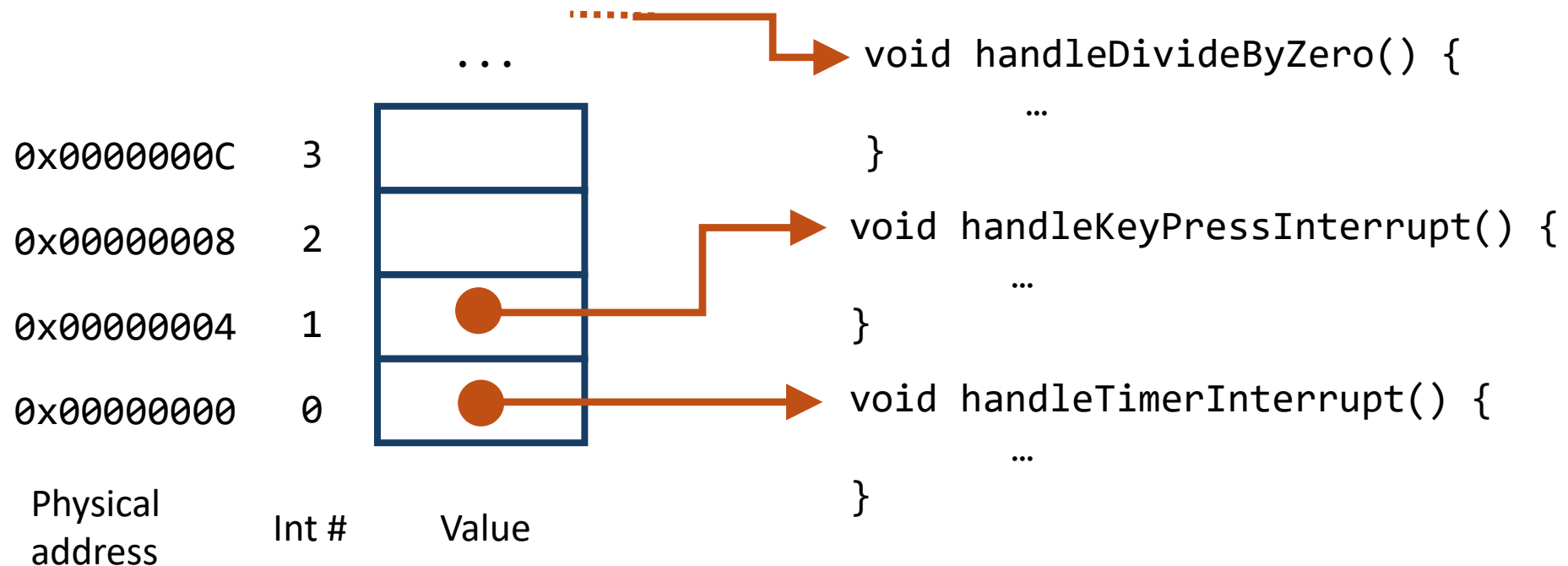
- OS pokes I/O register
- OS sits in a while() loop reading a “status” I/O register (“**polls** it”).
- Loop exits when the “status” register magically becomes **DEVICE_DONE** instead of **DEVICE_WORKING**.
- OS copies relevant data from the rest of the I/O registers into memory

- Asynchronous I/O: **Interrupts**

- OS pokes I/O register
- OS goes on and does something entirely different
- When the device is done, it smacks the CPU (“**triggers an interrupt**”)
- The interrupt hardware forcibly moves the CPU’s instruction pointer to a callback function (the “**interrupt handler**”) and pushes a new stack frame for it.
- When the handler is done, OS carefully returns to the stack frame that got interrupted

Interrupt Nitty-Gritty

- **Interrupt Vector:** an array of function pointers in RAM
 - Each pointer is to a different interrupt handler (eg, callback function)
 - Purpose of each callback hard-coded by hardware



Interrupt Nitty-Nitty-Gritty-Gritty

- See:
 - `xk/kern/vectors.s`
 - `xk/kern/trapasm.s`
 - `xk/kern/trap.c:trap`
 - `xk/inc/trap.h`



Interrupt Masking

- Interrupt handler runs with interrupts off
 - Re-enabled when interrupt completes
- OS kernel can also turn interrupts off
 - Eg., when determining the next process/thread to run
 - On x86:
 - CLI: disable interrupts
 - STI: enable interrupts
 - Only applies to the current CPU (on a multicore)
- We'll need this to implement context switching

Oh by the way



- Which of these details are specific to x86, and which are universal?
- Which of these details are there because they allow **hardware simplicity**?

High-bandwidth I/O

“Doing I/O in bulk”

- Direct memory access (**DMA**) engines
 - Dedicated circuits that sit outside of the CPU and copy bytes from external devices into RAM
 - CPU communicates with DMA engines through regular I/O registers
 - Glorified hardware `memcpy()`
- A simple toy DMA engine:

```
extern volatile uint32_t **ram_ptr;  
extern volatile uint32_t *disk_sector;  
extern volatile uint32_t *n_bytes;  
extern volatile uint32_t *direction; // DMA_READ or DMA_WRITE
```

Realistic DMA

- Store a queue of DMA transfers for the engine to work on
 - Each entry in the queue represents is a structure called a **buffer descriptor**
 - Descriptor specifies things like RAM address, number of bytes, directionality (where applicable), etc
 - CPU pushes descriptors, DMA engine pops them
- Queue implementation: “**ring buffer**”
 - Fixed-sized array of buffer descriptor structs stored in RAM
 - CPU keeps a “write pointer” to the head of the queue. Advances the pointer on push.
 - DMA engine keeps a “read pointer” to the back of the queue. Advances the pointer on pop.
 - Pointers wrap to beginning of array when they reach the end.
 - Position of ring buffer itself communicated using an I/O register
- CPU interrupt only if buffer empties/fills

Buffer Descriptors Illustrated

```
uint8_t valid;  
uint8_t direction;  
uint8_t n_bytes;  
uint32_t disk_sector;  
uint32_t *ram_addr;  
uint8_t unused;
```



RAM

```
struct disk_dma_descriptor dma_queue[8];  
extern volatile uint8_t **IO_REG_DISK_DMA_QUEUE;  
*IO_REG_DISK_DMA_QUEUE = &queue;
```

CPU

DMA
Engine