Basic Computer Security Concepts and Threat Modeling; Begin Operation System Concepts

CMSC 23200/33250, Winter 2023, Lecture 2

David Cash and Blase Ur

University of Chicago

Outline for Lecture 2

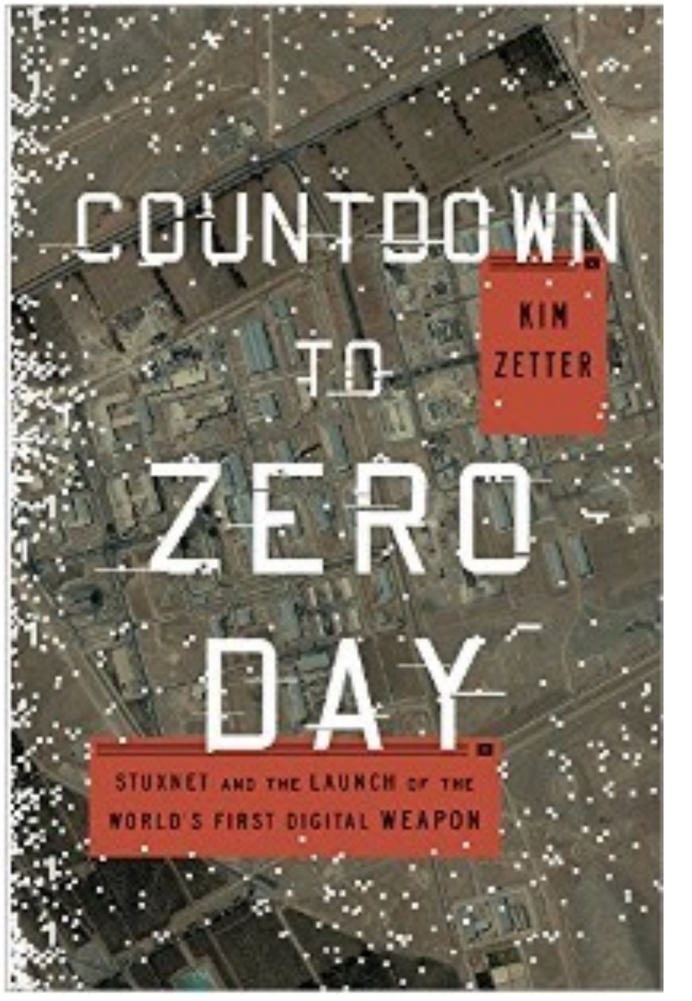
- A history of computer security via major events
- Reflect on Chapter 1
- Run through working example: Police Body Cams
  - Apply concepts from Chapter 1, particularly threat modeling
- Begin OS Security

Event 1: The Morris Worm (1988)

- 99-line C program that exploited vulnerabilities in `sendmail` and `fingerd`, weak passwords, and unsafe default settings
- Spread automatically over networks (the definition of a worm), reinfecting same machines many times (accidentally)
- 1000s of infected machines were knocked offline
  - Real costs to victims
- Morris convicted under the Computer Fraud Act
  - Sentenced to 3 years probation, 400 hours community service, plus fines.
- Led to a sea-change in computer security

Event 2: Stuxnet (2005? Found in 2010)

- Highly advanced attack created by US and Israeli governments to sabotage Iranian nuclear program.
- Attack targeted “air gapped” uranium enrichment systems, specifically to damage centrifuges. Malware would run centrifuges at rates that would cause them to fail often, but not too often.
- Other advanced government threats subsequently discovered.



Event 3: Target (2013)

- Millions of credit card and debit card numbers used at Target were stolen.
- Fazio Mechanical, an HVAC contractor, was compromised via a phishing email that installed the Citadel trojan.
- From the Target vendor portal, the attackers moved laterally to other systems.
- RAM-scraping malware was installed on POS terminals.

Event 4: SolarWinds (2020)

- Widely used network-management software SolarWinds used by many major corporations and governments.
- By October 2019, attackers compromised the software build system used by SolarWinds and installed backdoors
- After deployment, malware would stay dormant for weeks, only operate on potentially high-value targets, and try to mimic legitimate traffic.
- Data exfiltrated from numerous government agencies and corporations.

Event 5: Log4Shell (2021-2022?)

- A major vulnerability in the Log4j logging framework for JavaScript has been causing all sorts of problems for the last couple years.
- Attackers are able to execute arbitrary Java code on other people's servers.

Lessons from history

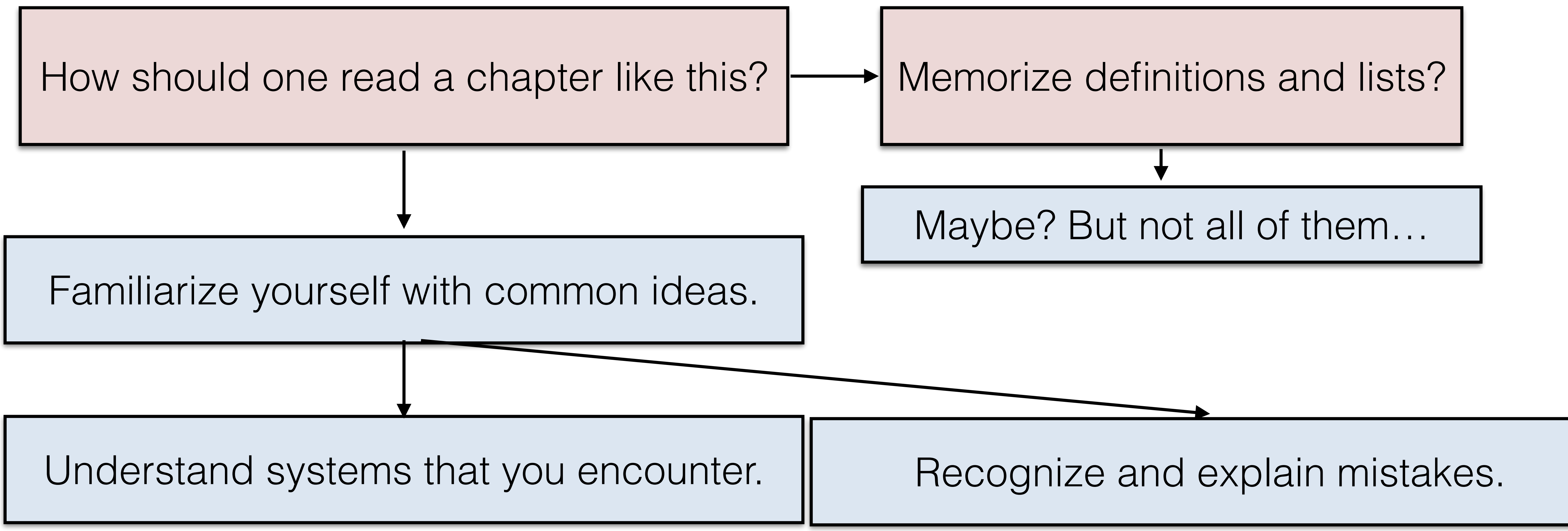
- Security is very, very, hard, even for well-resourced, motivated organizations.
- We need tools and techniques to systematize our thinking rather than scattershot approaches.
- Chapter 1 begins doing this!

[van Oorschot'20], Chapter 1: Summary

1. **Fundamental goals** of computer security
2. Computer security **policies** and **attacks**
3. **Risk**, risk assessment, and modeling expected losses
4. **Adversary modeling** and security analysis
5. **Threat modeling**: diagrams, trees, lists and STRIDE
6. **Model-reality gaps** and real-world outcomes
7. **Design principles** for computer security
8. Why computer security is **hard**

Understanding Chapter 1

1. Fundamental goals of computer security
2. Computer security policies and attacks
3. Risk, risk assessment, and modeling expected losses
4. Adversary modeling and security analysis
5. Threat modeling: diagrams, trees, lists and STRIDE
6. Model-reality gaps and real-world outcomes
7. Design principles for computer security
8. Why computer security is hard



A Running Example: Police Body Cams

- Worn continuously by police while on duty. Records activity to storage.
- Used in court, training, adjudicating complaints, …

These should be "secure" right? → Where to start?

Start with van Oorschot's 6 Fundamental Goals?

1. Confidentiality
2. Integrity
3. Authorization
4. Availability
5. Authentication
6. Accountability

Maybe, but probably start with needs of application.

Example partial list:
- Videos should be useful. Good quality and authenticated.
- Videos should not "disappear" when someone wants them to.
- Videos should be accessible "when appropriate", but otherwise confidential.

Still not clear how to apply goals!

Steps Extracted from Chapter 1

1. Articulate policies surrounding data and other assets.
2. Diagram system in a simple yet useful way.
3. Model adversary categories.
4. Engage in “threat modeling” to enumerate relevant attacks by adversaries against diagrammed system.

Who in the org should actually ***do*** this?

↓

Arguably: Leadership (CIO/CTO/CISO), middle management, developers, operators, outside stake-holders.

Step 1: Assets

1. Video data
2. Actual cameras
3. Camera configuration equipment
4. Administration server
5. Remote storage account (third party)

More?

Step 1: Policies

1. Video data should be retained for X years.

   - Internal process to redact accidental recordings (e.g. officer in bathroom)

2. Video should only be accessible with court approval.

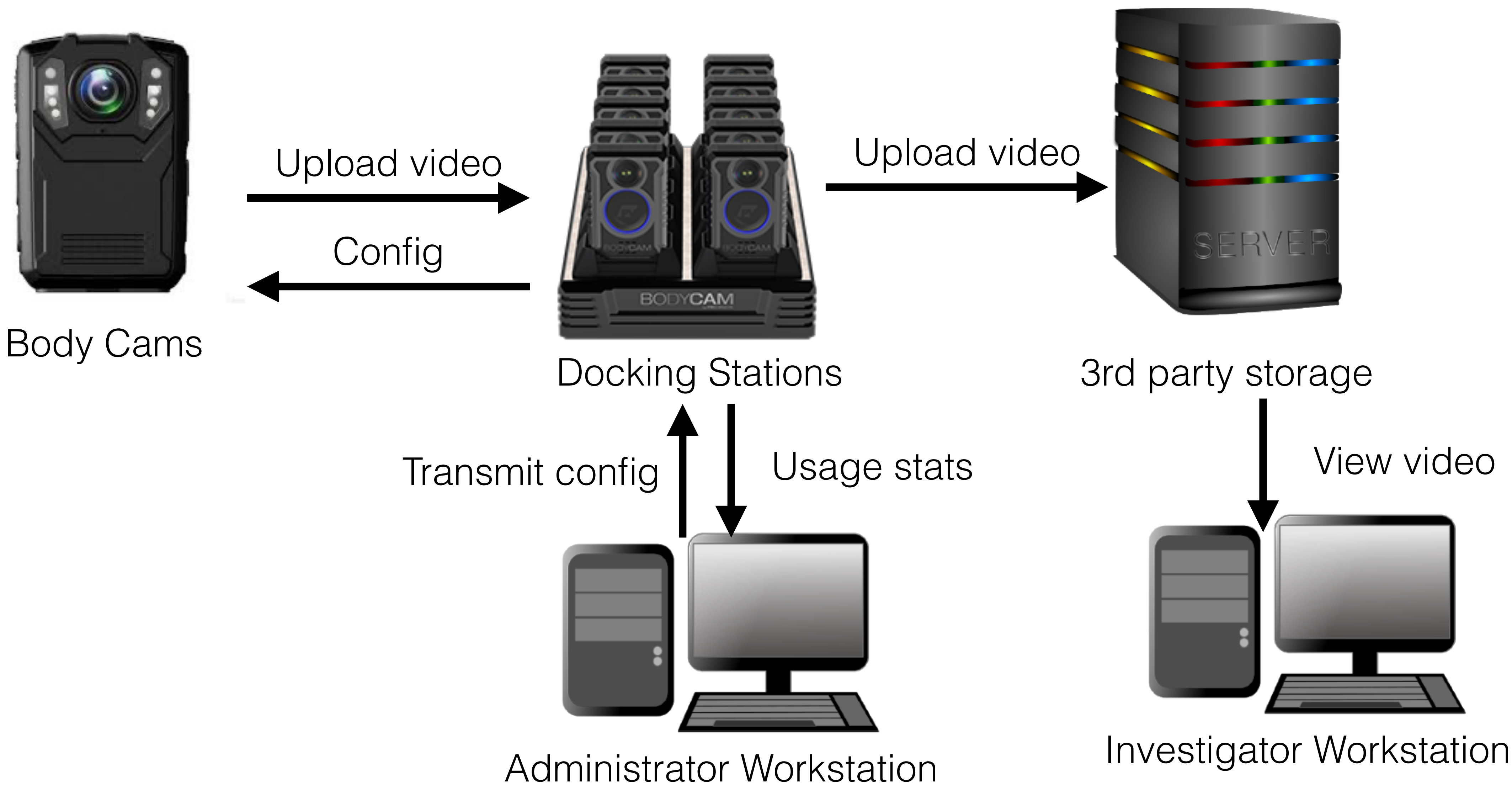
3. Only authentic videos from official cams should be stored.

4. Police should not be able to turn camera off without being logged.

More?

Step 2: Diagram the System

- Principle components
- Interactions
- Sometimes: “Trust boundaries” (e.g. cloud vs. on-premises)



Step 3: Begin Adversary Modeling

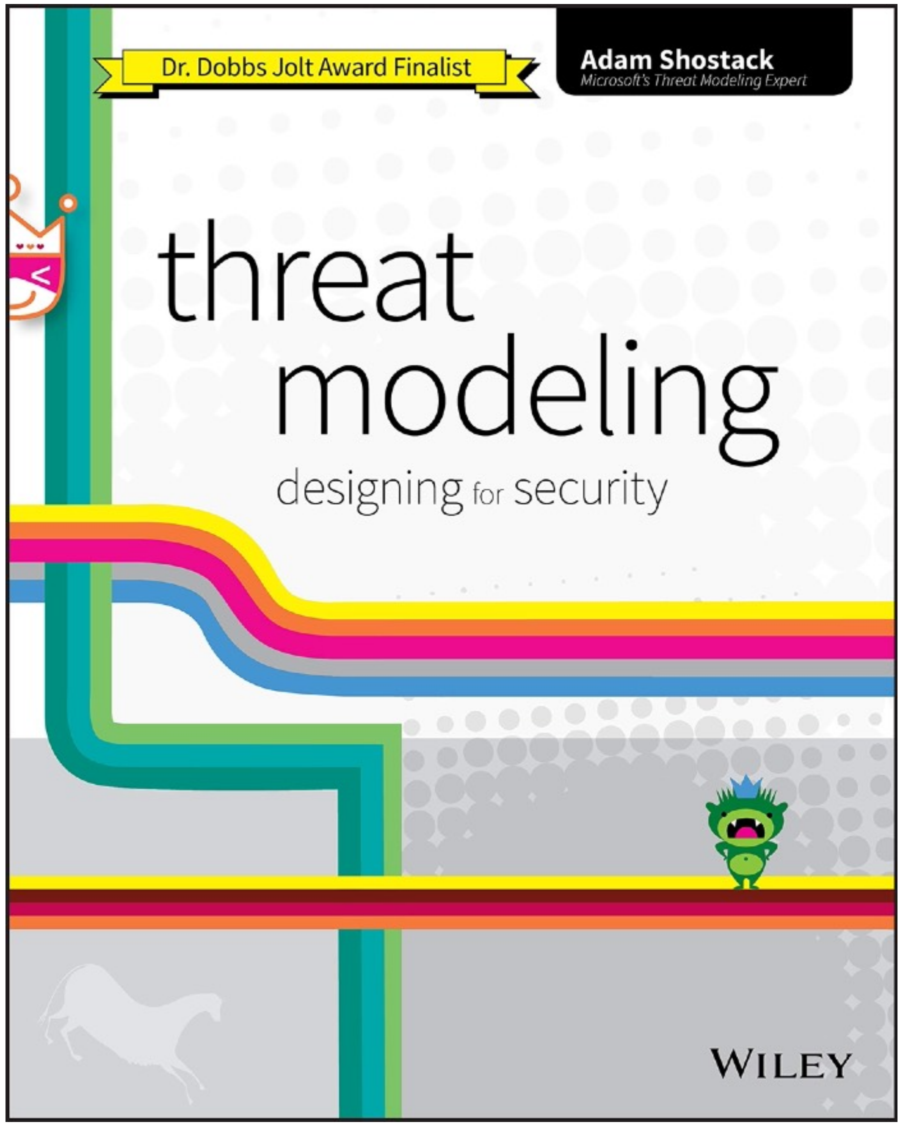
1. Corrupt police officer hiding activity
2. Corrupt police department hiding activity
3. Corrupt administrator spying
4. Criminal trying to delete video
5. Domestic hacker (outsider) seeking videos
6. Insider at body cam vendor planting backdoor
7. Insider at storage provider snooping videos
8. Foreign government-level hackers fomenting distrust of government

More?

Step 4: Threat Modeling

Threat Modeling = Brainstorming Crutch for "What could go wrong?"

Examples:
- STRIDE (Microsoft)
- Attack Trees
- Center of Gravity (CoG)
- PASTA
- DREAD
- …



STRIDE Threat Modeling

Brainstorm attacks that fit each of six categories:

Spoofing

Tampering

Repudiation

Information disclosure

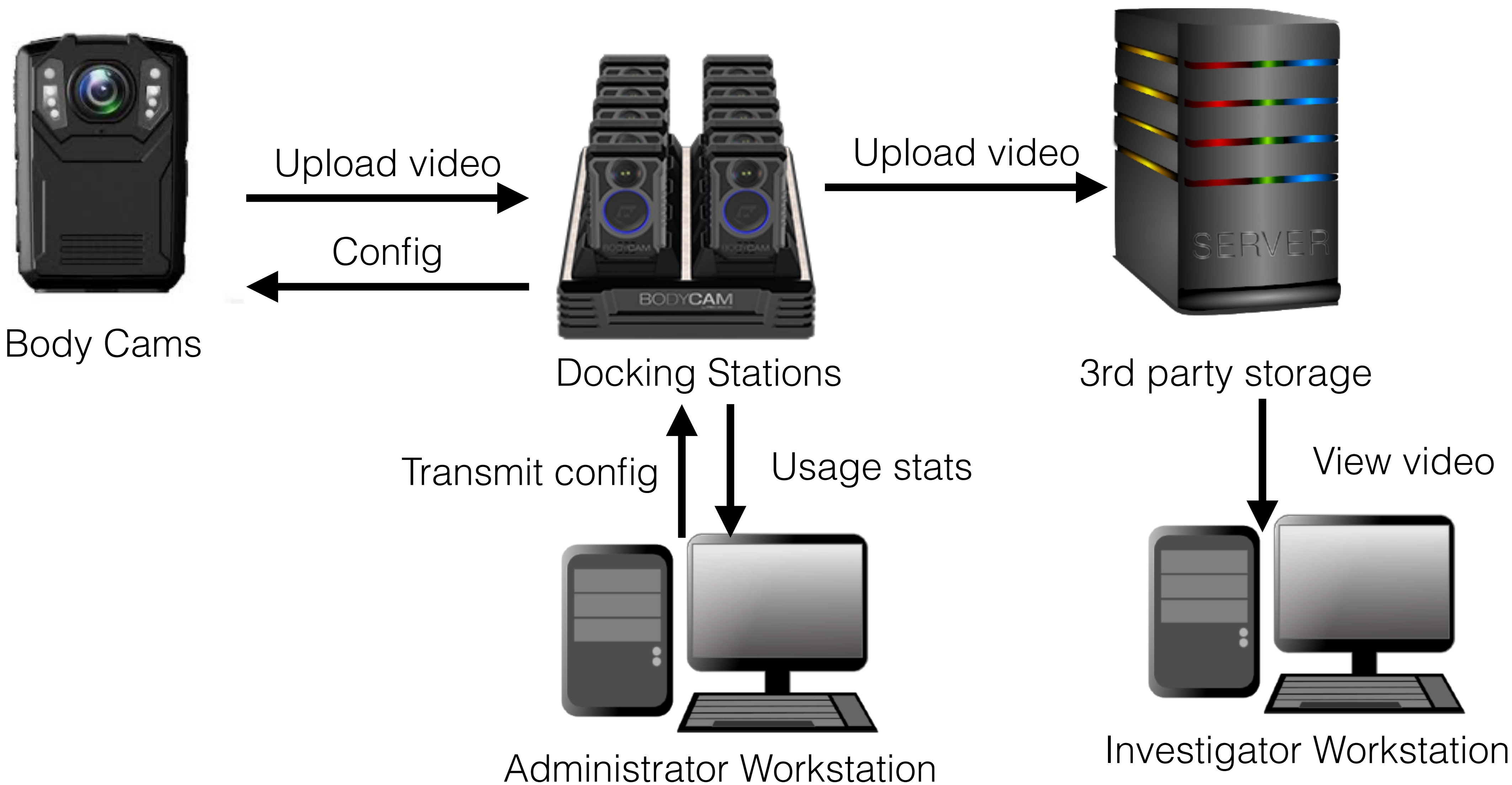
Denial of service

Elevation of privilege

- Can search for each type against each component in diagram
- Can search for each type as mounted by each adversary

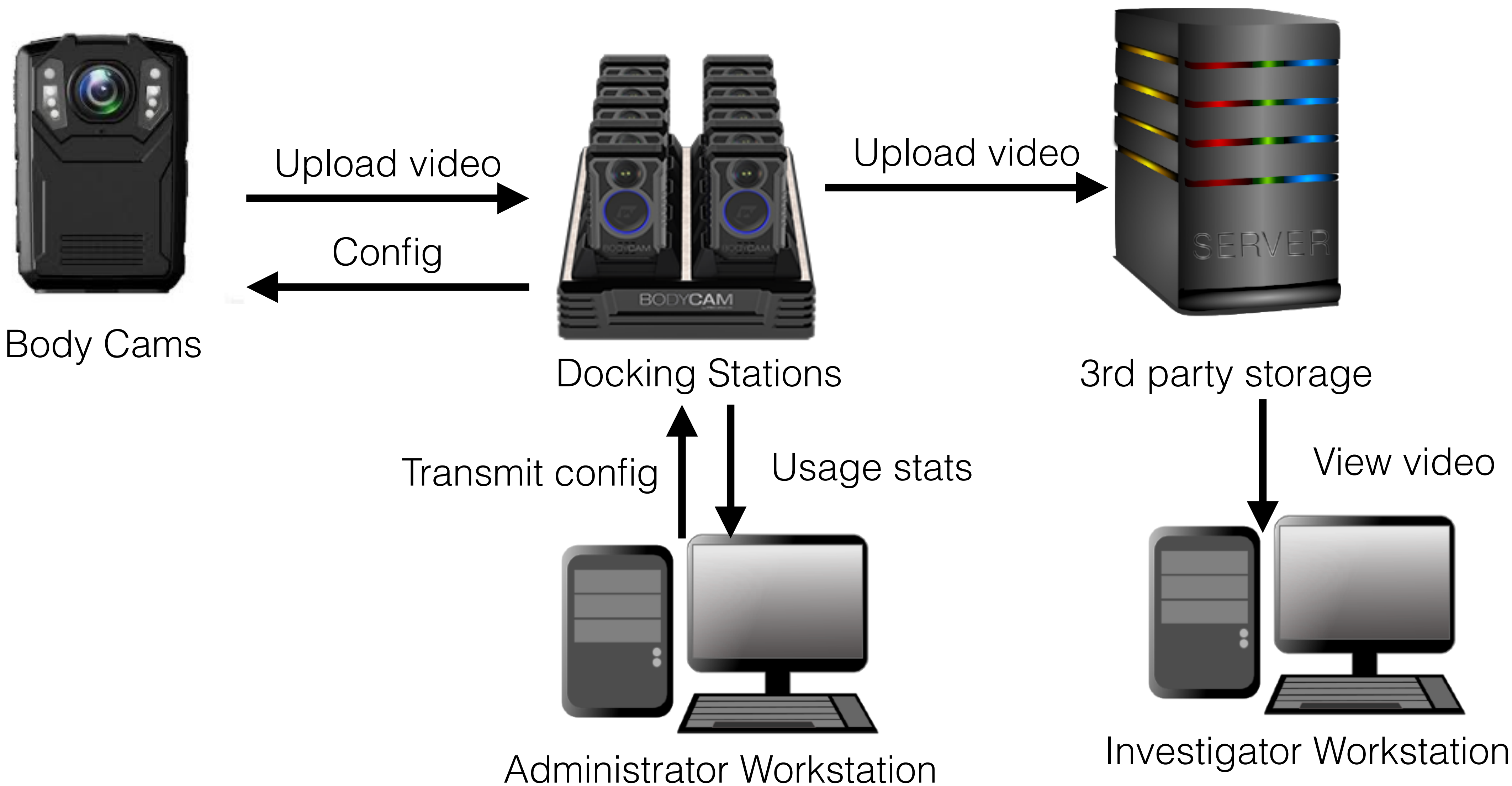
STRIDE-by-Component Exercise

Spoofing in…



STRIDE-by-Component Exercise

Tampering in…



Chapter 1 Conclusions

- Brainstorming hopefully leads to reasonably complete list of threats.
- Feed them into mitigation strategies (e.g. “use strong passwords”).
- Threat modeling is incomplete, and still relies on experience.

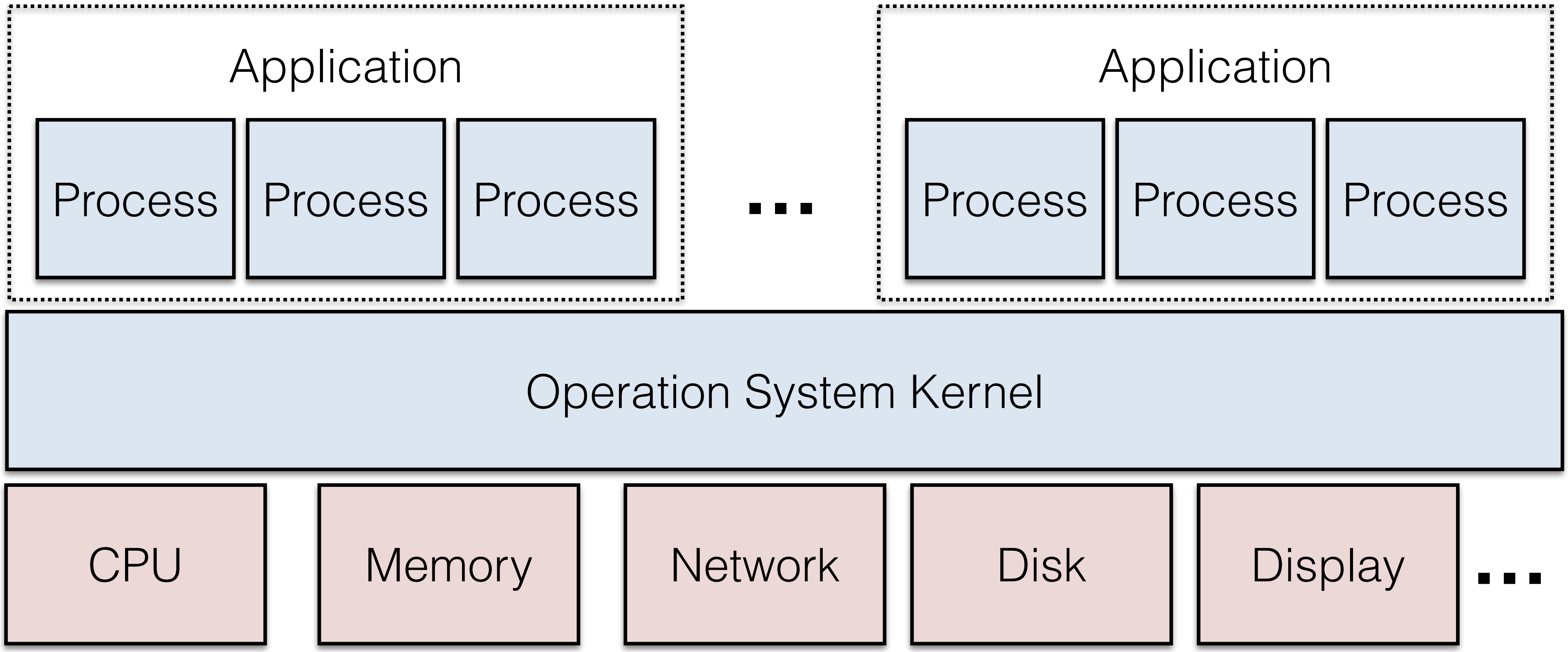
Where does the class go from here?

- We look at security issues in a variety of important settings
- Aim is for security to be a vehicle itself to learn about hardware, OSes, networking, databases, phones, …
- Issues in Chapter 1 will suffuse through topics, giving us a language to explain designs and mistakes
- But we won’t engage in systematic threat modeling.

Outline for Lecture 2

- Reflect on Chapter 1
- Run through working example: Police Body Cams
  - Apply concepts from Chapter 1, particularly threat modeling
- **Begin OS Security**

Review of OS Structure

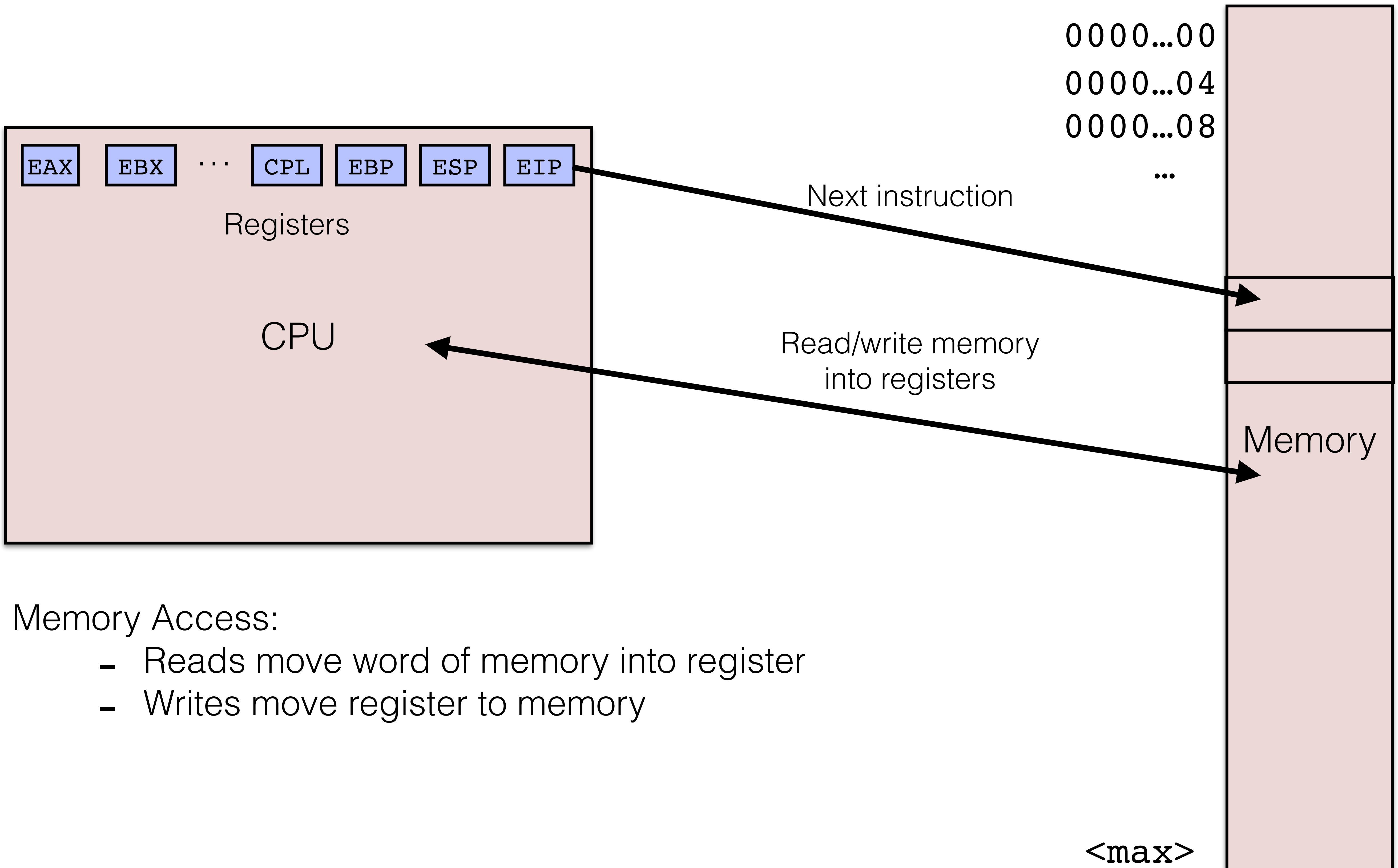


Security/safety: Must protect processes from each other, protect hardware, …

Questions, though:

- What distinguishes the kernel from not-kernel?
- What *is* a process?

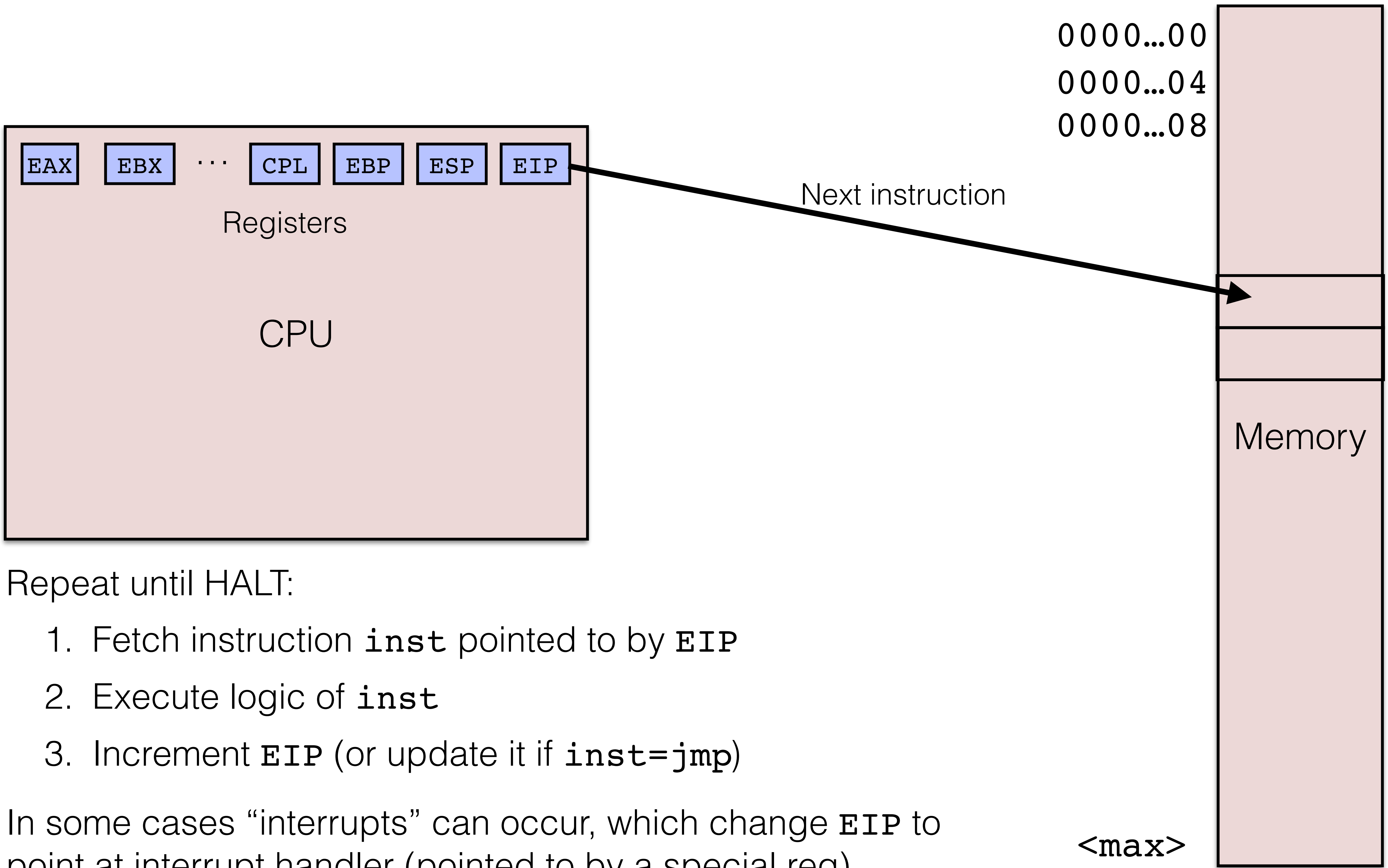
How a CPU (x86) Works (extremely high level)



Memory Access:

- Reads move word of memory into register
- Writes move register to memory

How a CPU (x86) Works (extremely high level)

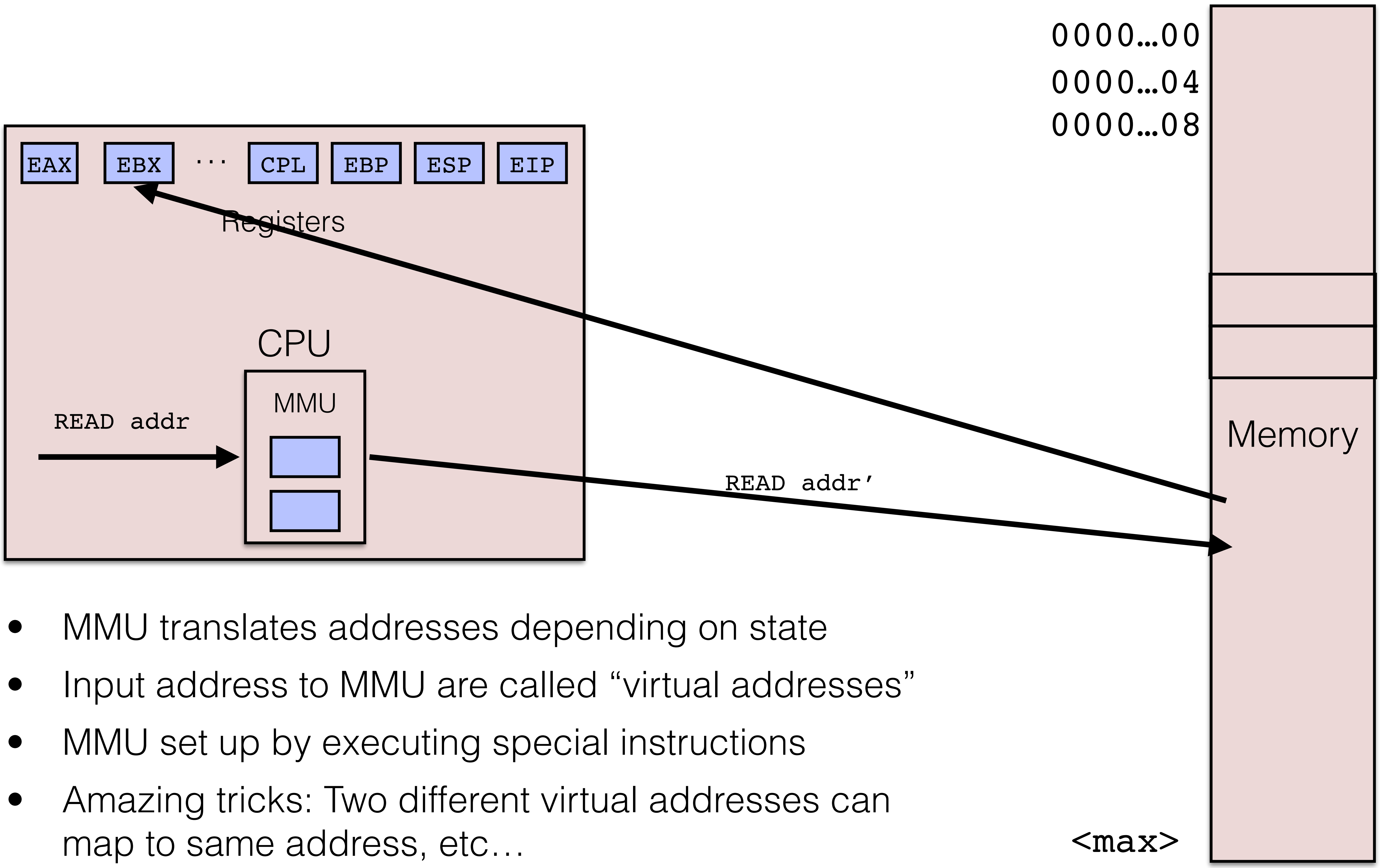


Repeat until HALT:

1. Fetch instruction `inst` pointed to by `EIP`
2. Execute logic of `inst`
3. Increment `EIP` (or update it if `inst=jmp`)

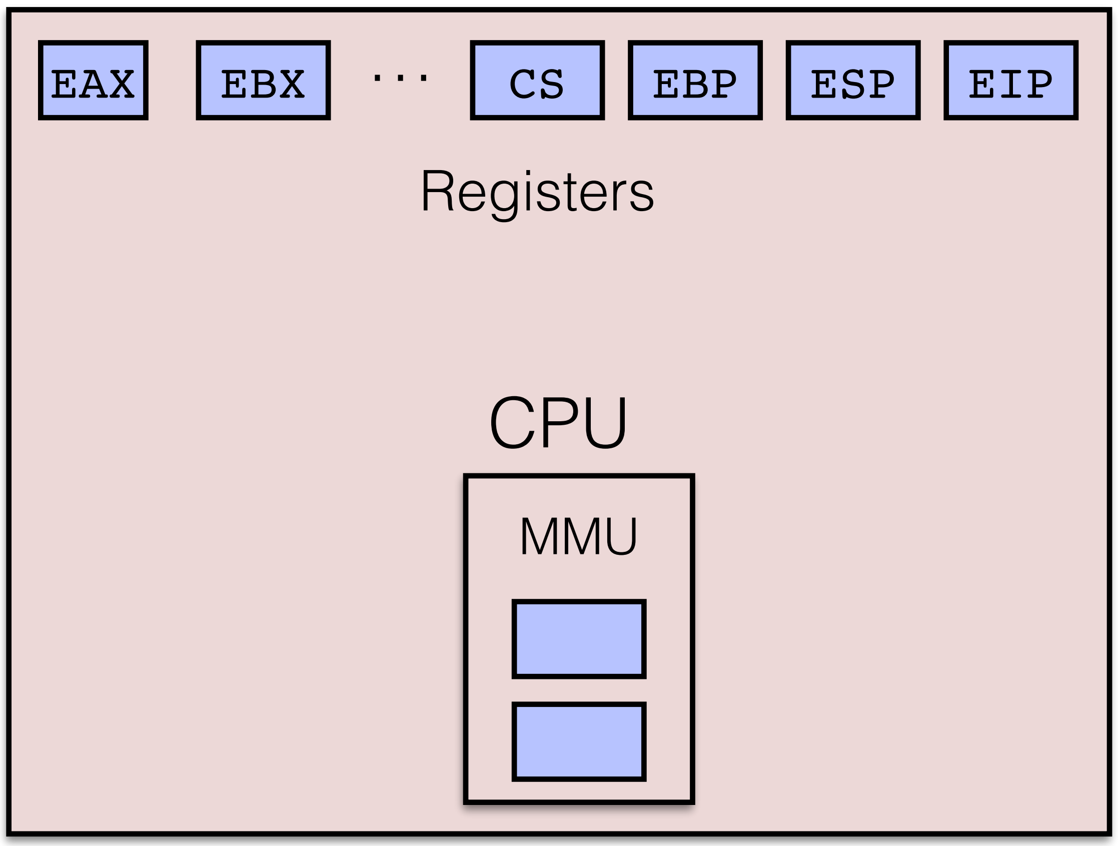
In some cases "interrupts" can occur, which change `EIP` to point at interrupt handler (pointed to by a special reg).

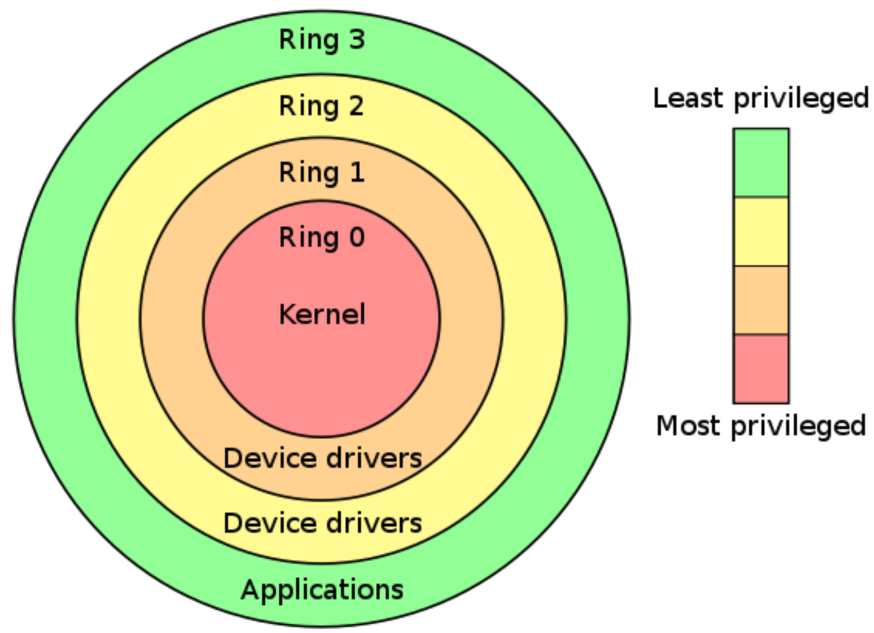
Memory Management Unit (MMU)

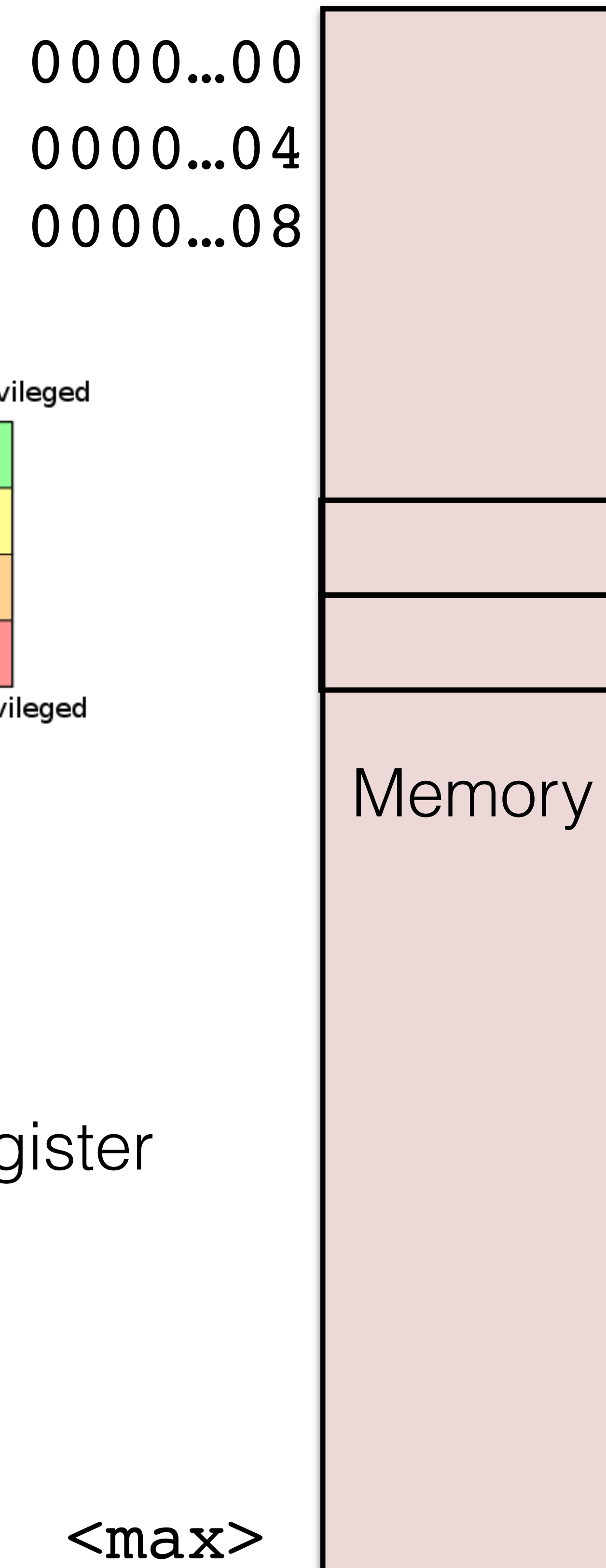


- MMU translates addresses depending on state
- Input address to MMU are called “virtual addresses”
- MMU set up by executing special instructions
- Amazing tricks: Two different virtual addresses can map to same address, etc…

Isolation in x86: It all comes down to CPL

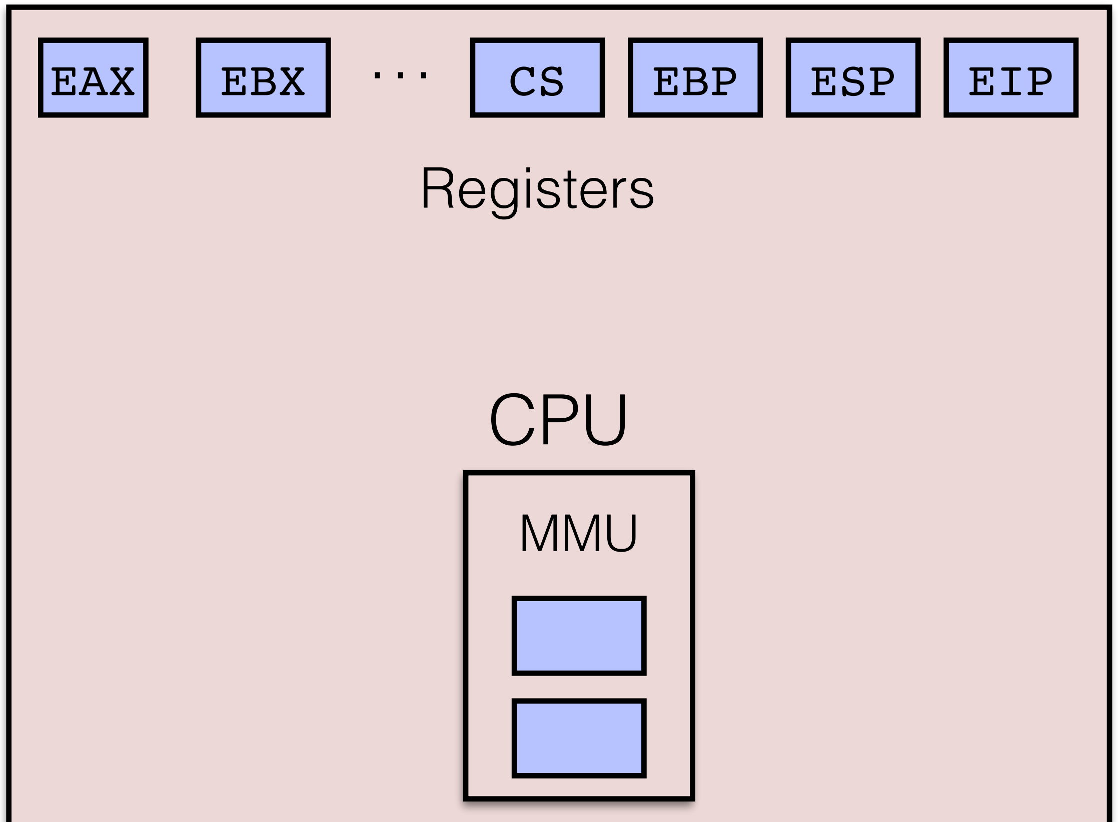






- `CPL` is “current privilege level”, two designated bits in `CS` register
- If `CPL = 0`: Then processor will execute any instruction
- If `CPL = 3`: Then processor will only execute subset of instructions

Isolation in x86: It all comes down to CPL



Big Idea: Kernel runs with **CPL**=0, and *all* other programs run with **CPL**=3.

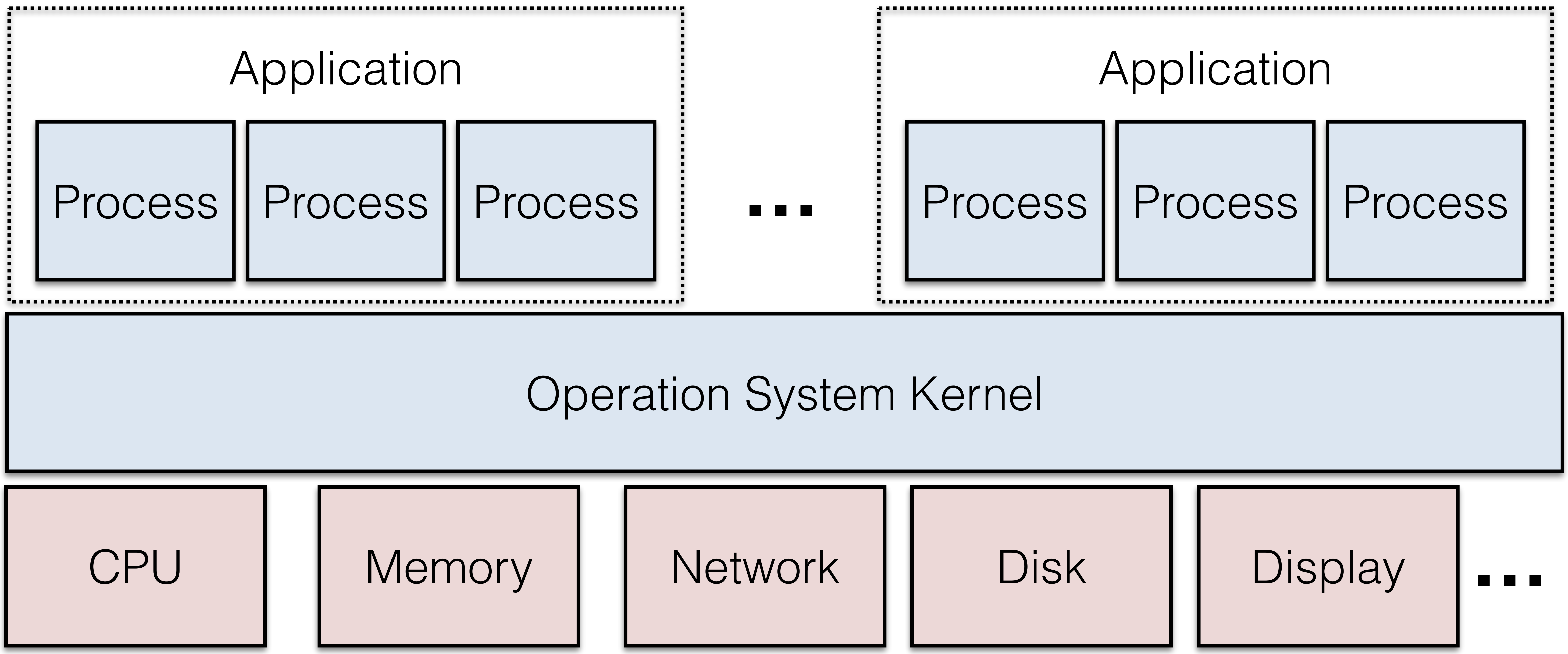
If **CPL**=0, then CPU **will** allow…

- Direct access to (almost) any addr
- Changes to (almost) any register
- Changes internal state of MMU
- Including setting **CPL**=3!

If CPL=3, then CPU **will not** allow…

- Direct access to memory (only via MMU)
- Changes to several registers
- Changes to internal state of MMU
- Setting **CPL**=0 (!)

Back to our diagram…

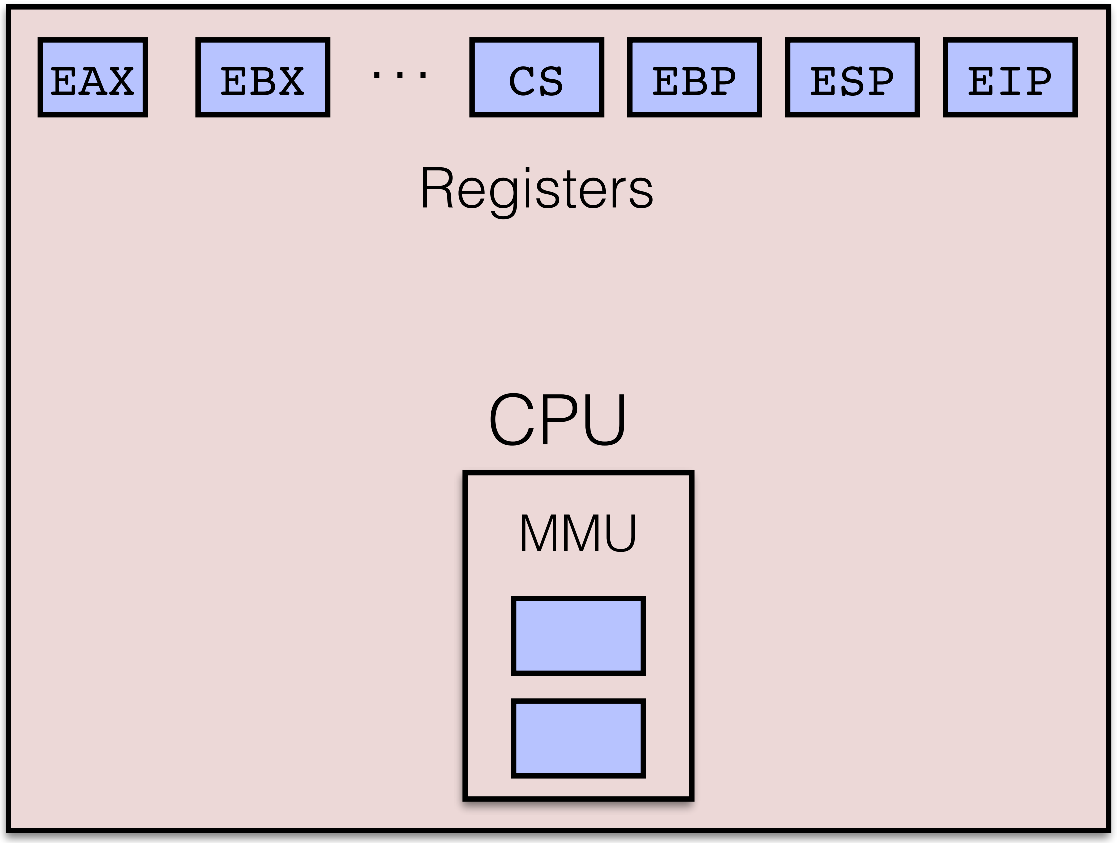


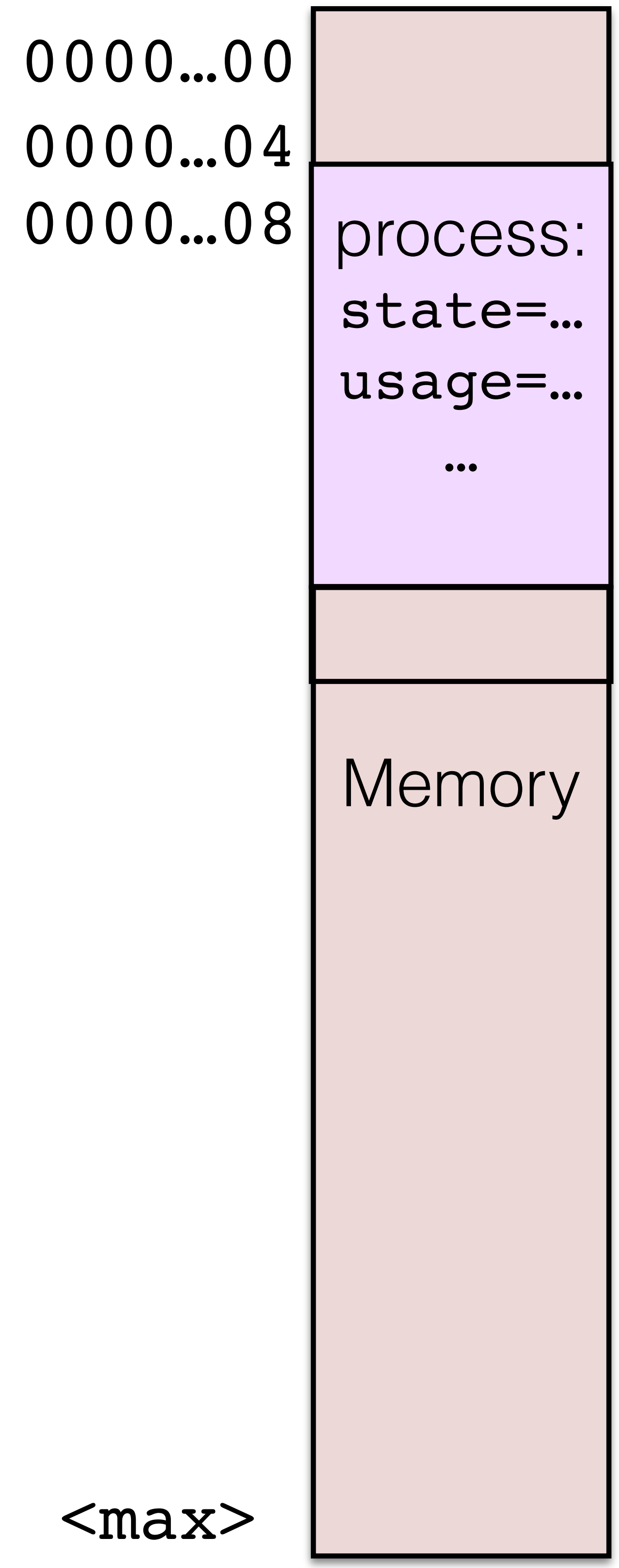
The CPL!

Questions, though:

- What distinguishes the kernel from not-kernel?
- What *is* a process?

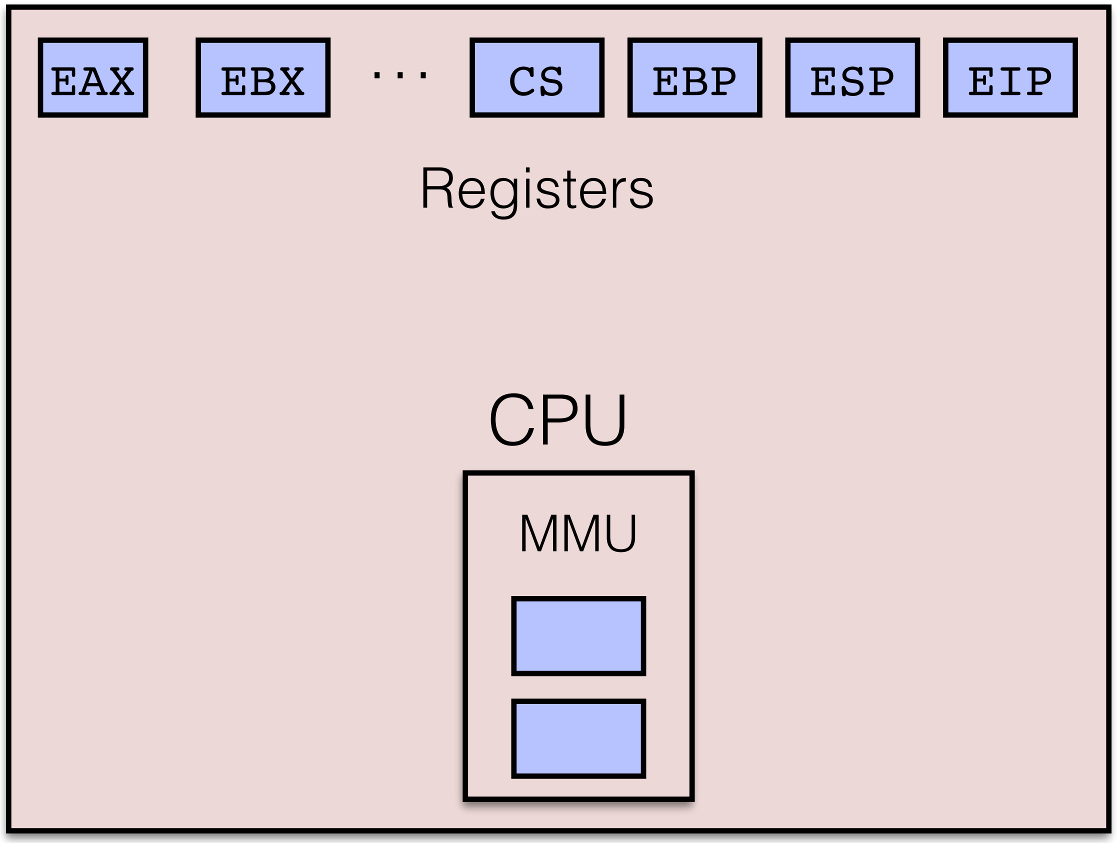
What *is* a process?

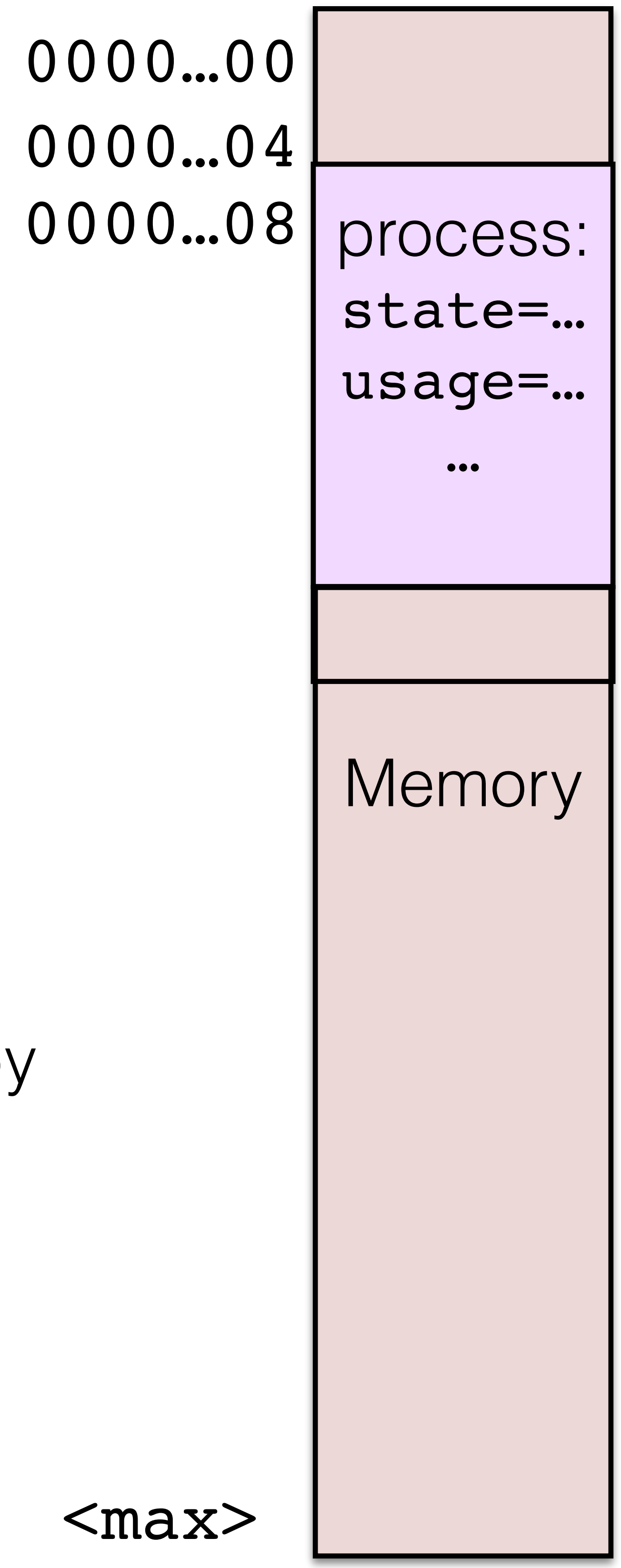




- One Answer: A data structure in “kernel memory”, including
  - MMU configuration
  - Register values
- Kernel can load these values up, set CPL=3, and turn over control “to the process” (i.e. set EIP)
- If kernel regains control, it can save these values to swap process out

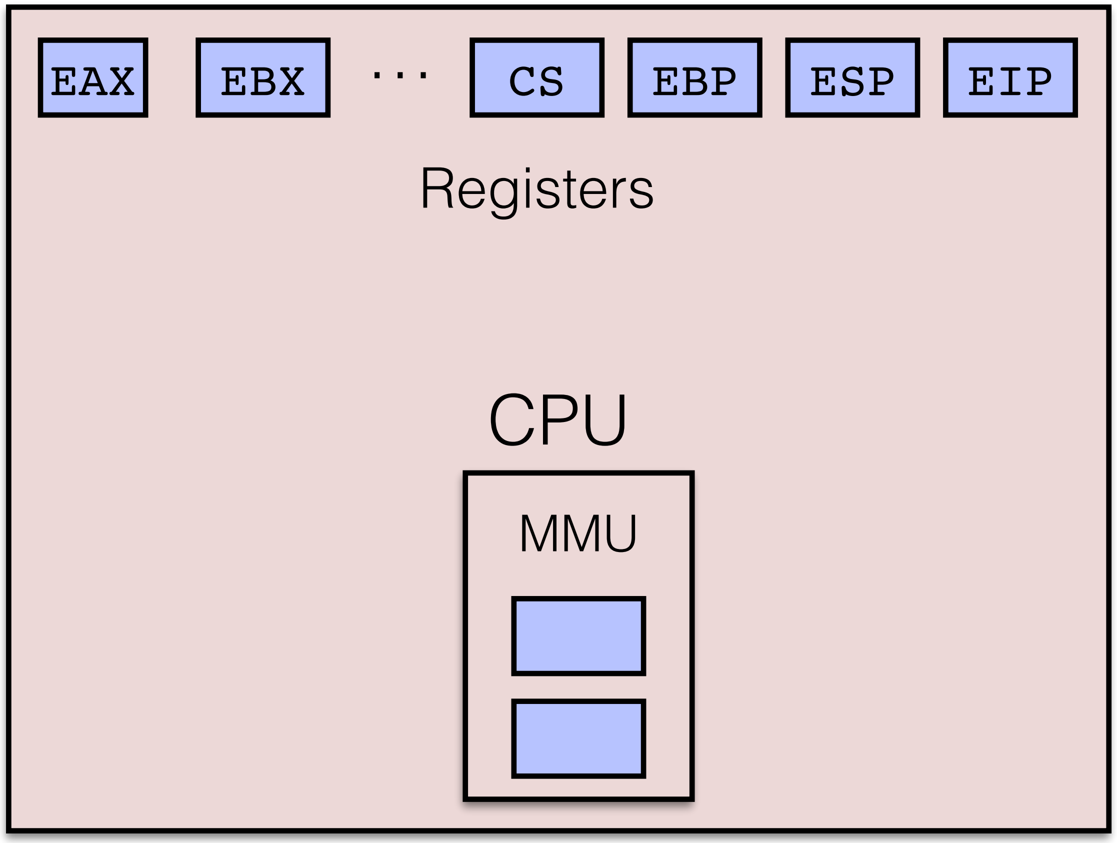
Handling Memory for a Process

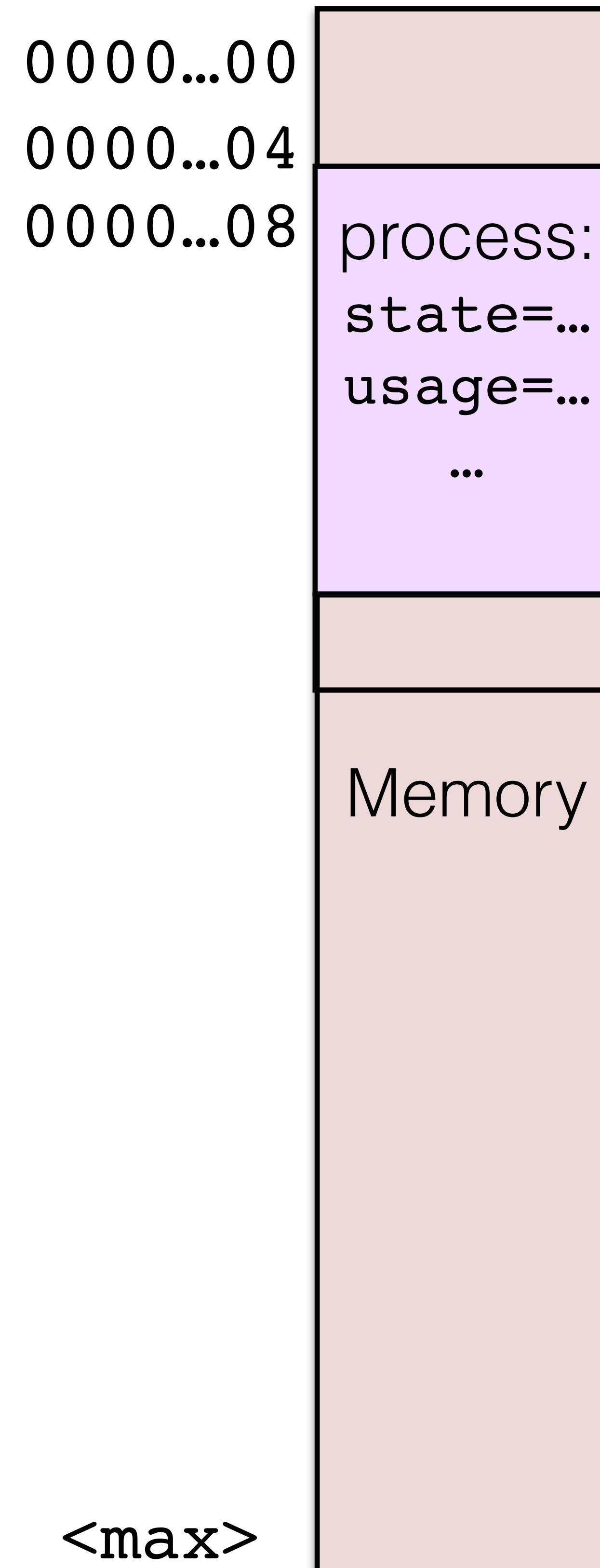




- Kernel creates a “virtual address space” for each process.
- Same virtual addresses (e.g. starting near 0) can be used by every process! They get translated to different physical addresses.
- Kernel can also mark some virtual address ranges (called segments) as “read only” or “do not execute” (**`EIP`** not allowed to point there).
- Violations are **`SEGFAULT`**s: MMU will take over in this case

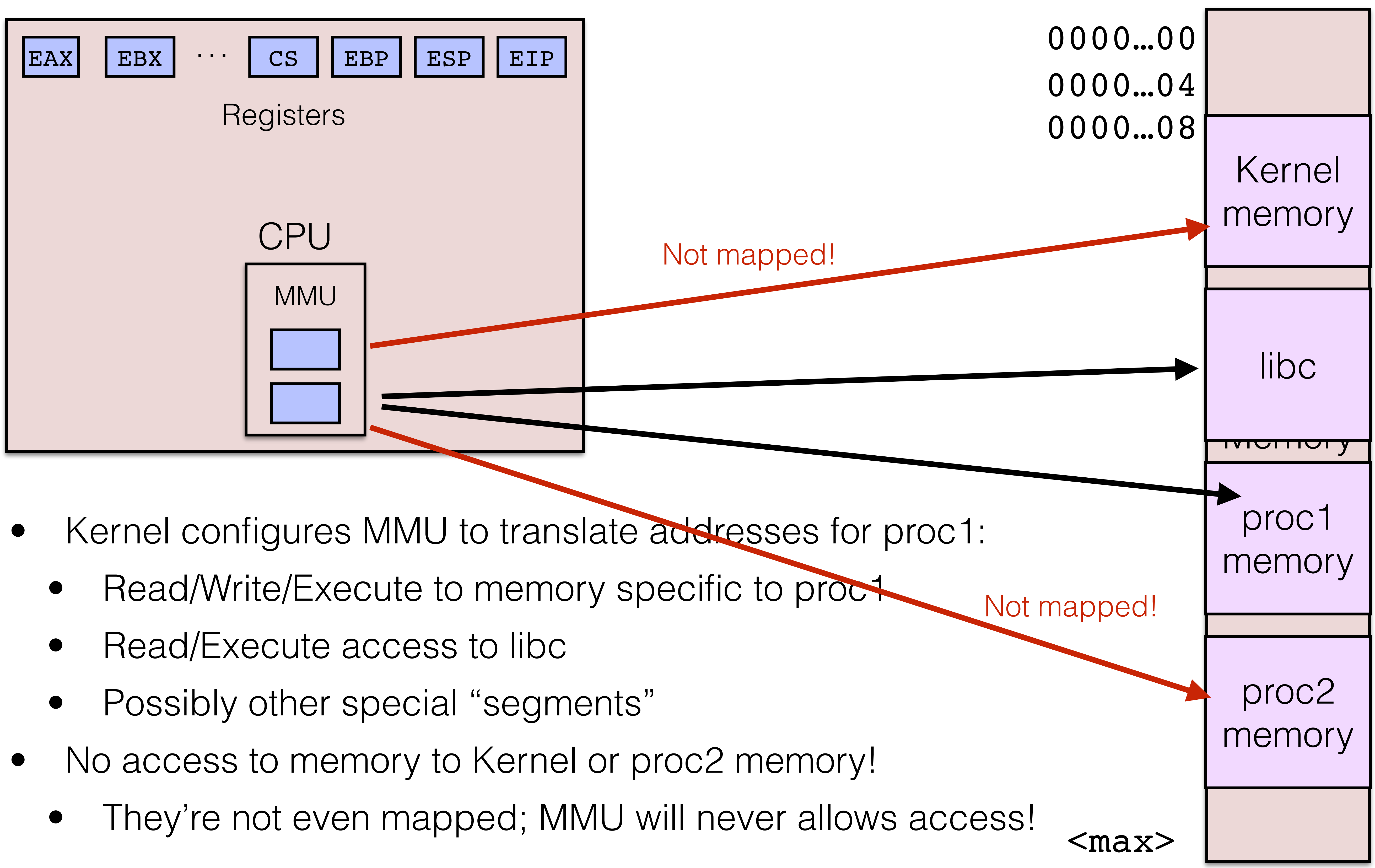
Handling Memory for a Process (cont.)





- Kernel can also map same memory into several processes' virtual address space
- Ex: Code for **`malloc`** is not copied for every process.

Handling Memory for a Process (cont.)



- Kernel configures MMU to translate addresses for proc1:
  - Read/Write/Execute to memory specific to proc1
  - Read/Execute access to libc
  - Possibly other special “segments”
- No access to memory to Kernel or proc2 memory!
  - They’re not even mapped; MMU will never allows access!

Next Time

- The UNIX security model (users/root, file/process permissions, …)
- Running an executable: Function calls and “the stack”
- Begin control hijacking (i.e. how all this nice work falls apart!)

The End