Next edition of OS @ Brown

Fall 2026, Instructor: Malte Schwarzkopf (me)

If you enjoyed the material and want to help make the next version of the course, **apply to TA in Fall 2026**.

Either way, I want to hear from you:

1. *what is great about the course, and*
2. *what you'd change!*

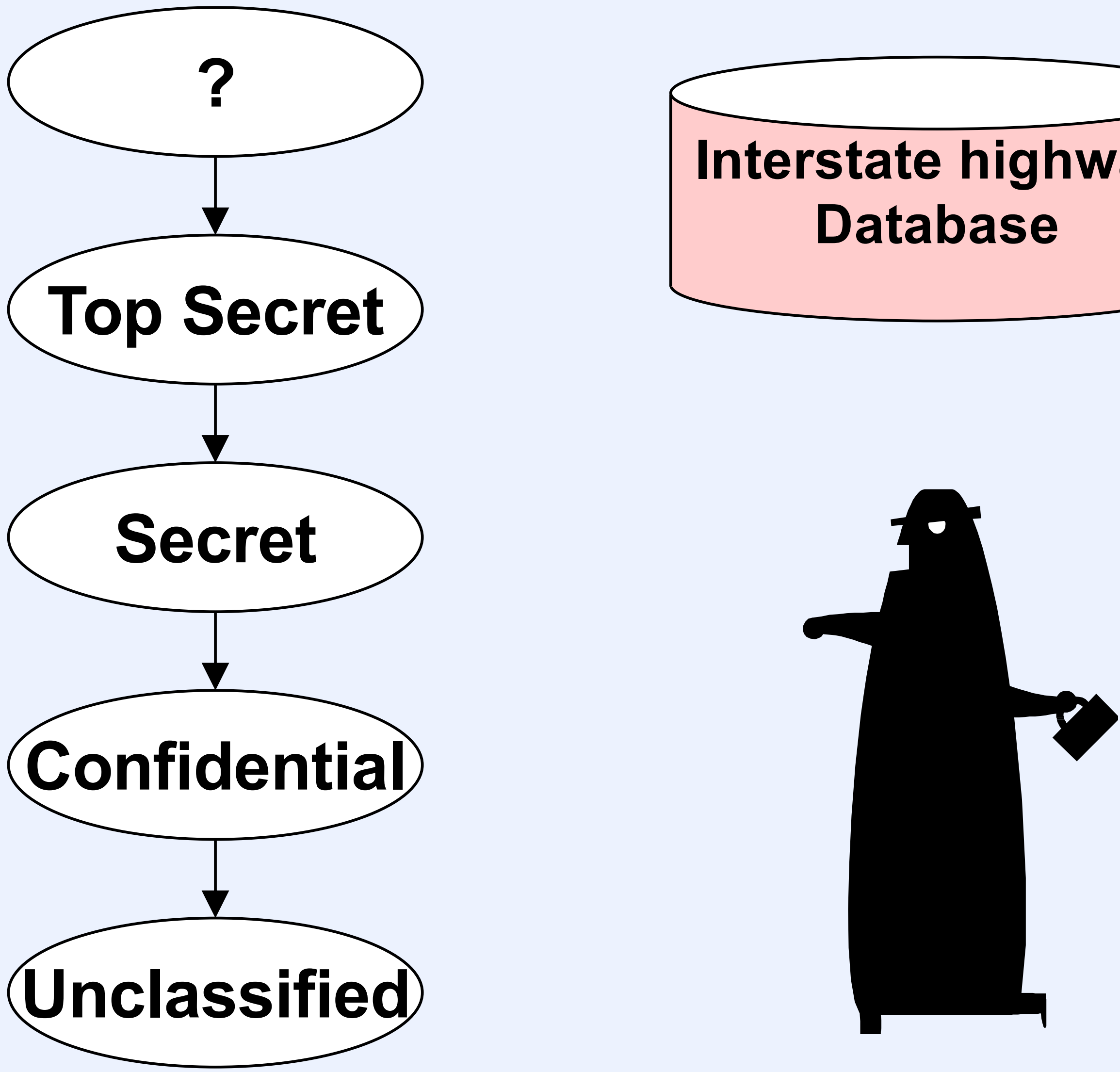
Reach out to meet or share your thoughts:

malte@brown.edu

Security Part 7

Live Anonymous Q&A:
https://tinyurl.com/cs1670feedback

Integrity


?
Top Secret
Secret
Confidential
Unclassified
Interstate highway
Database

Biba Model

- Integrity is what's important
  - no-write-up
  - no-read-down

Quiz 1

You're concerned about downloading malware to your computer and very much want to prevent it from affecting your computer. Which would be the most appropriate policy to use?

a) no write up

b) no read up

c) no write down

d) no read down

Windows and MAC

- Concerns
  - viruses
  - spyware
  - etc.
- Installation is an integrity concern
- Solution
  - adapt Biba model

Windows Integrity Control

- No-write-up
- All subjects and objects assigned a level
  - untrusted
  - low integrity
    - Internet Explorer/Edge
  - medium integrity
    - default
  - high integrity
  - system integrity
- Object owners may lower integrity levels
- May set *no-read-up* on an object

Industrial-Strength Security

- **Target:**
  - embezzlers

Clark-Wilson Model

- Integrity and confidentiality aren't enough
  – there must be control over how data is produced and modified
    - well formed transactions

Cash account

withdrawals here

Accounts-payable account

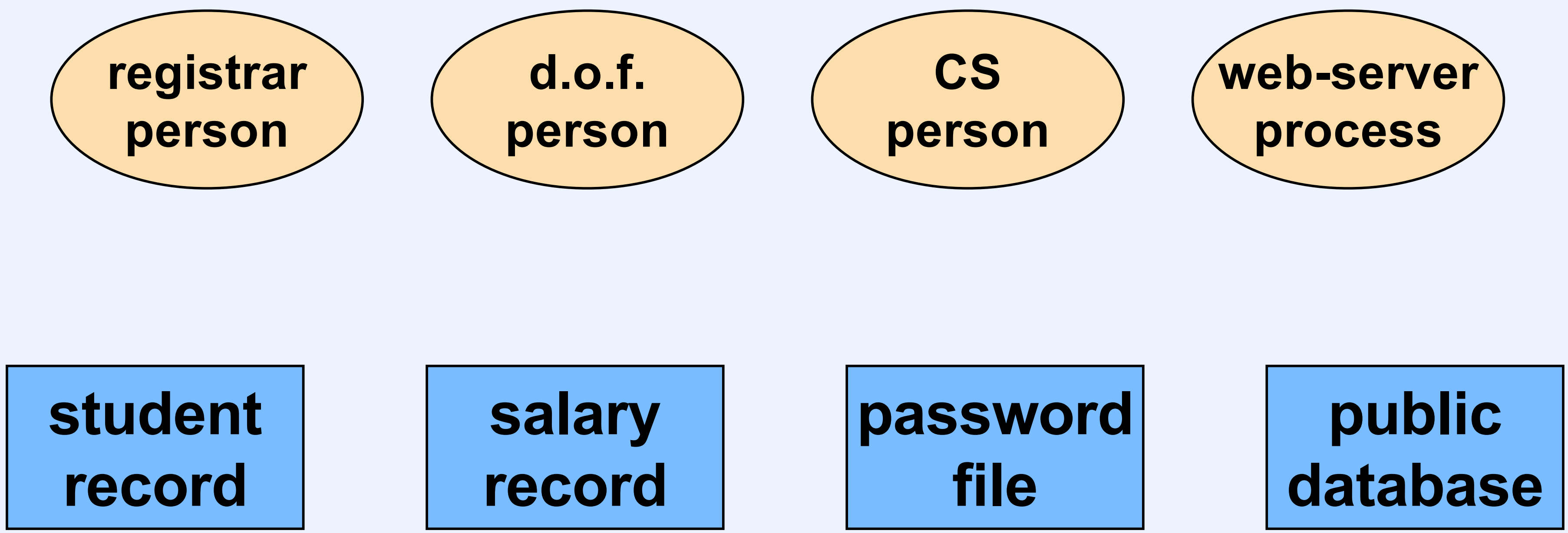
must be matched by entries here

- Separation of duty
  – steps of transaction must involve multiple people

Mandatory Access Control (MAC)

Implementing MAC

- Label subjects and objects
- Security policy makes decisions based on labels and context



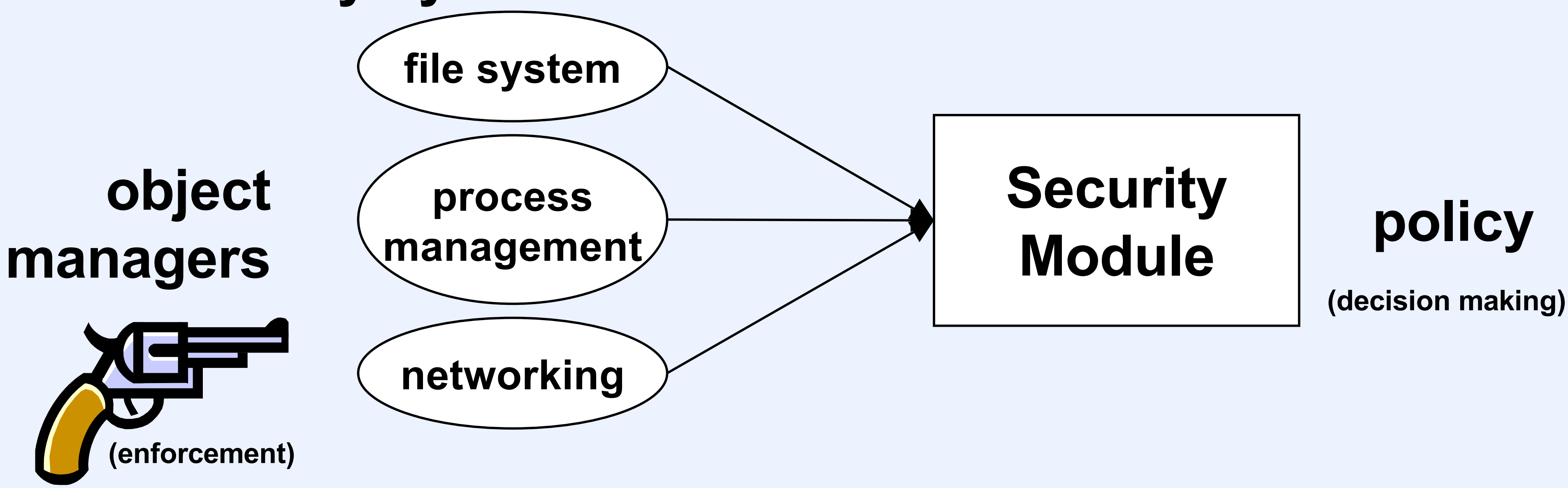
Quiz 2

I have a file that I accidentally set as having rw permission for everyone (0666). You have a process that has opened my file rw. I discover this and immediately change the permissions to 0600 (access only by me). Can your process still read and write the file?

a) It can read and write
b) It can read, but not write
c) It can write, but not read
d) It can do neither

SELinux

- Security-Enhanced Linux
  - MAC-based security
  - labels on all subjects and objects
  - policy-specification language
- Use in Android (since v4.3)
- Deny by default



SELinux Examples (1)

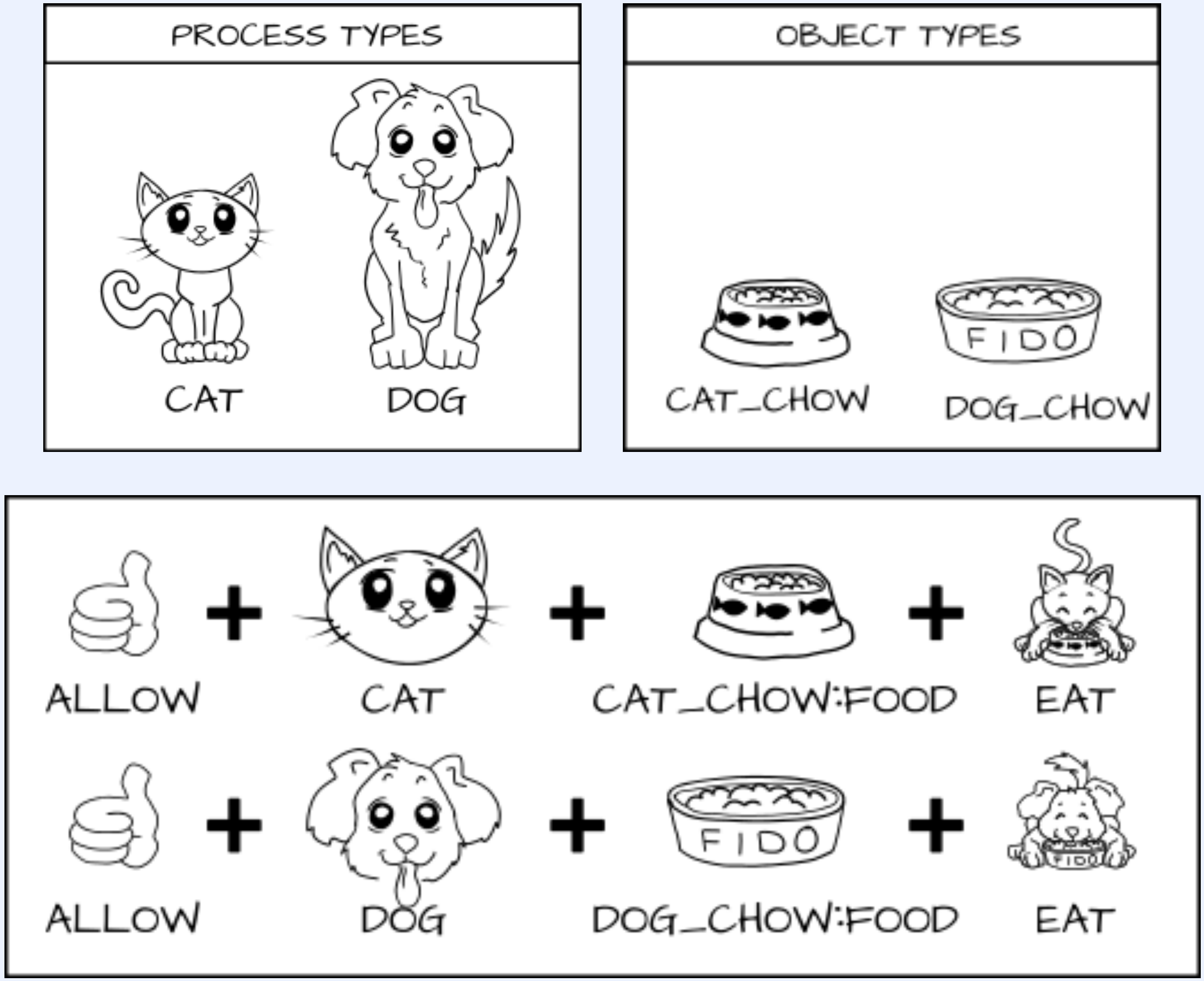
- Publicly readable files assigned type *public_t*
- Subjects of normal users run in domain *user_t*
- /etc/passwd: viewable, but not writable, by all
- /etc/shadow: protected
- SELinux rules

```
allow user_t public_t : file read
```

  - normal users may read public files

```
allow passwd_t passwd_data_t : file {read write}
```

  - /etc/shadow is of type *passwd_data_t*
  - subjects in *passwd_t* domain may read/write /etc/shadow



Cartoon credit: https://opensource.com/business/13/11/selinux-policy-guide

Result

Cartoon credit: https://opensource.com/business/13/11/selinux-policy-guide

SELinux Examples (2)

- How does a program get into the *passwd_t* domain?
  - assume passwd program is of type *passwd_exec_t*

```
allow passwd_t passwd_exec_t : file entrypoint
allow user_t passwd_exec_t : file execute
allow user_t passwd_t : process transition
type_transition user_t passwd_exec_t : process
  passwd_t
```

Quiz 3

We’ve seen how the setuid feature in Unix is used to allow normal users to change their passwords in /etc/shadow.

a) This approach actually isn’t secure, which is among the reasons why SELinux exists

b) The approach is secure and thus SELinux doesn’t really add any additional protection to /etc/shadow

c) The approach is secure but there are other potential /etc/shadow-related vulnerabilities that SELinux helps deal with

Off-the-Shelf SELinux

- **Strict policy**
  - normal users in *user_r* role
  - users allowed to be administrators are in *staff_r* role
    - but may run admin commands only when in *sysadm_r* role
  - policy requires > 20,000 rules
  - tough to live with
- **Targeted policy**
  - targets only “network-facing” applications
  - everything else in *unconfined_t* domain
  - ~11,000 rules

Capability-Based Systems

Confused-Deputy Problem

- The system has a pay-per-use compiler
  - keeps billing records in file /u/sys/comp/usage
  - puts output in file you provide
    - /u/you/comp.out
- The concept of a pay-per-use compiler annoys you
  - you send it a program to compile
  - you tell it to put your output in /u/sys/comp/usage
  - it does
    - it's confused
    - you win

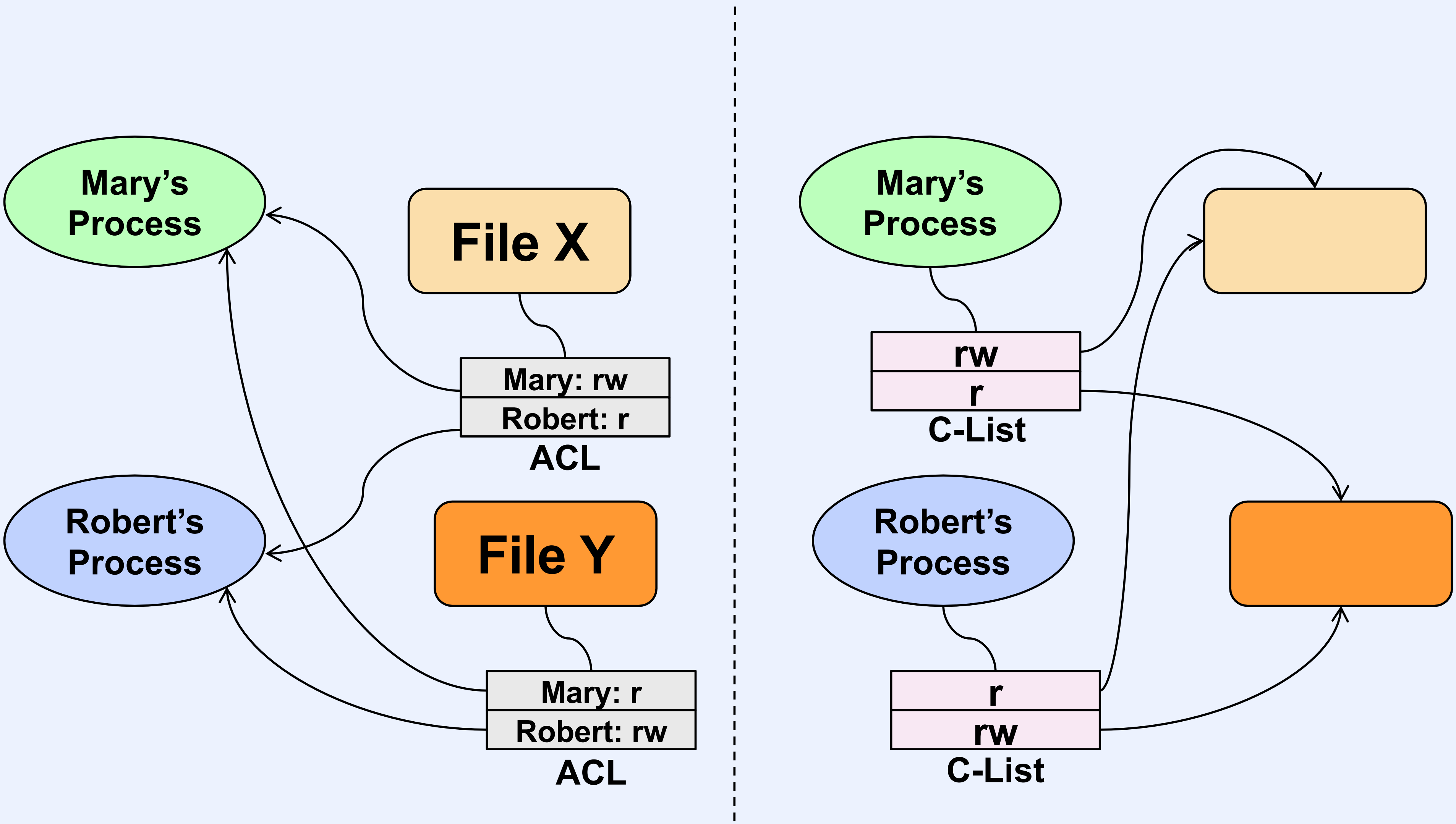
Unix and Windows to the Rescue

- Unix
  - compiler is "su-to-compiler-owner"
- Windows
  - client sends impersonation token to compiler
- Result
  - malicious deputy problem
- Could be solved by passing file descriptors
  - not done
  - should be …

Authority

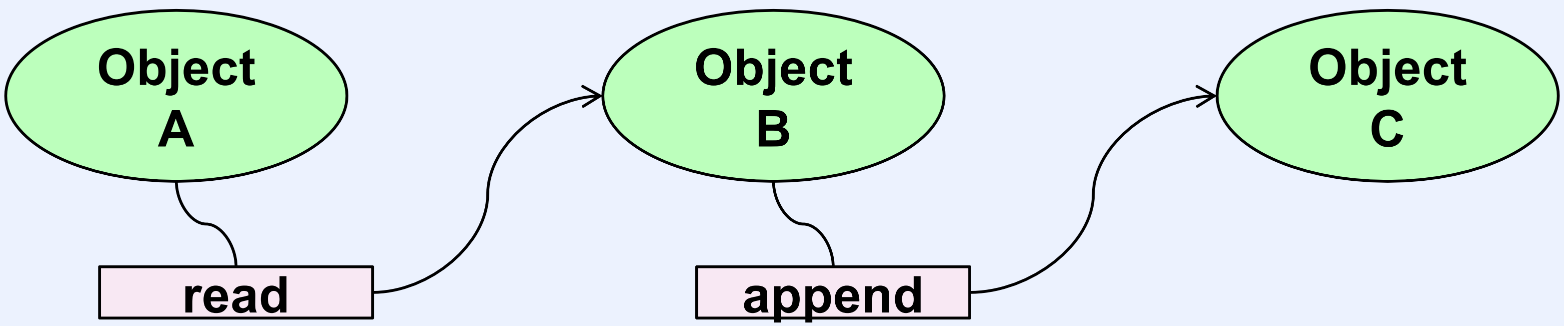
- **Pure ACL-based systems**
  - authority depends on subject’s user and group identities
- **Pure capability-based systems**
  - authority depends upon capabilities possessed by subject

ACLs vs. C-Lists

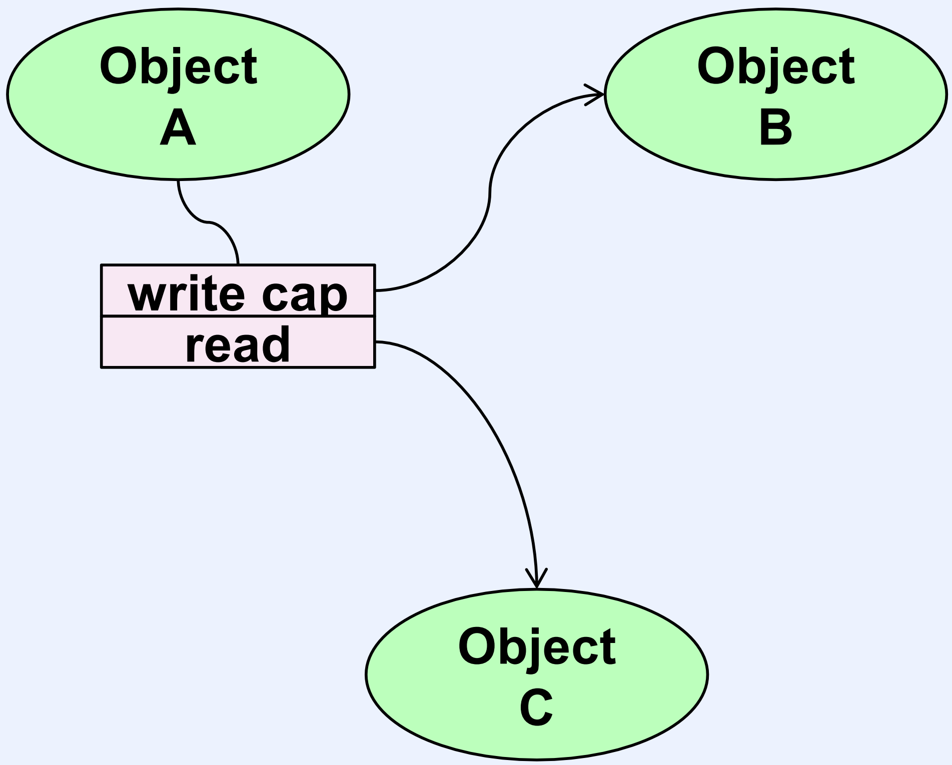


More General View

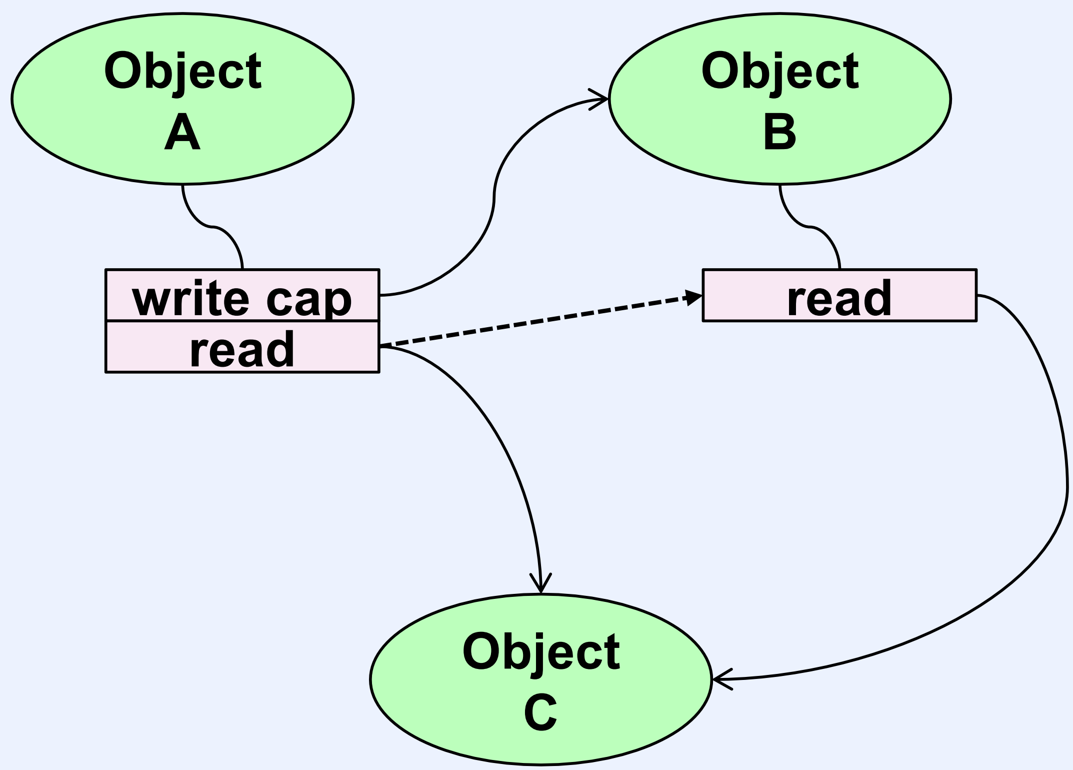
- Subjects and resources are *objects* (in the OO sense)



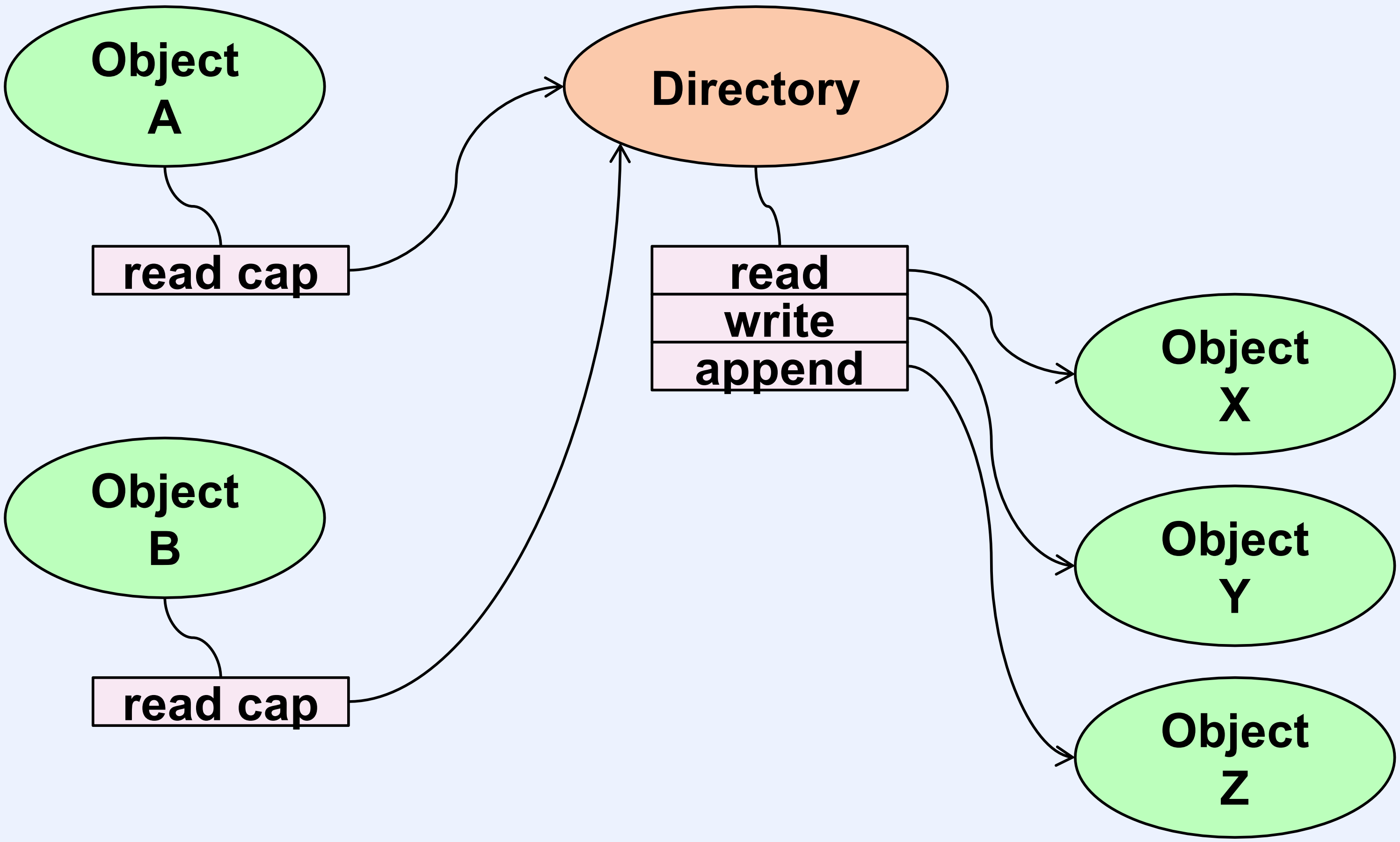
Copying Capabilities (1)



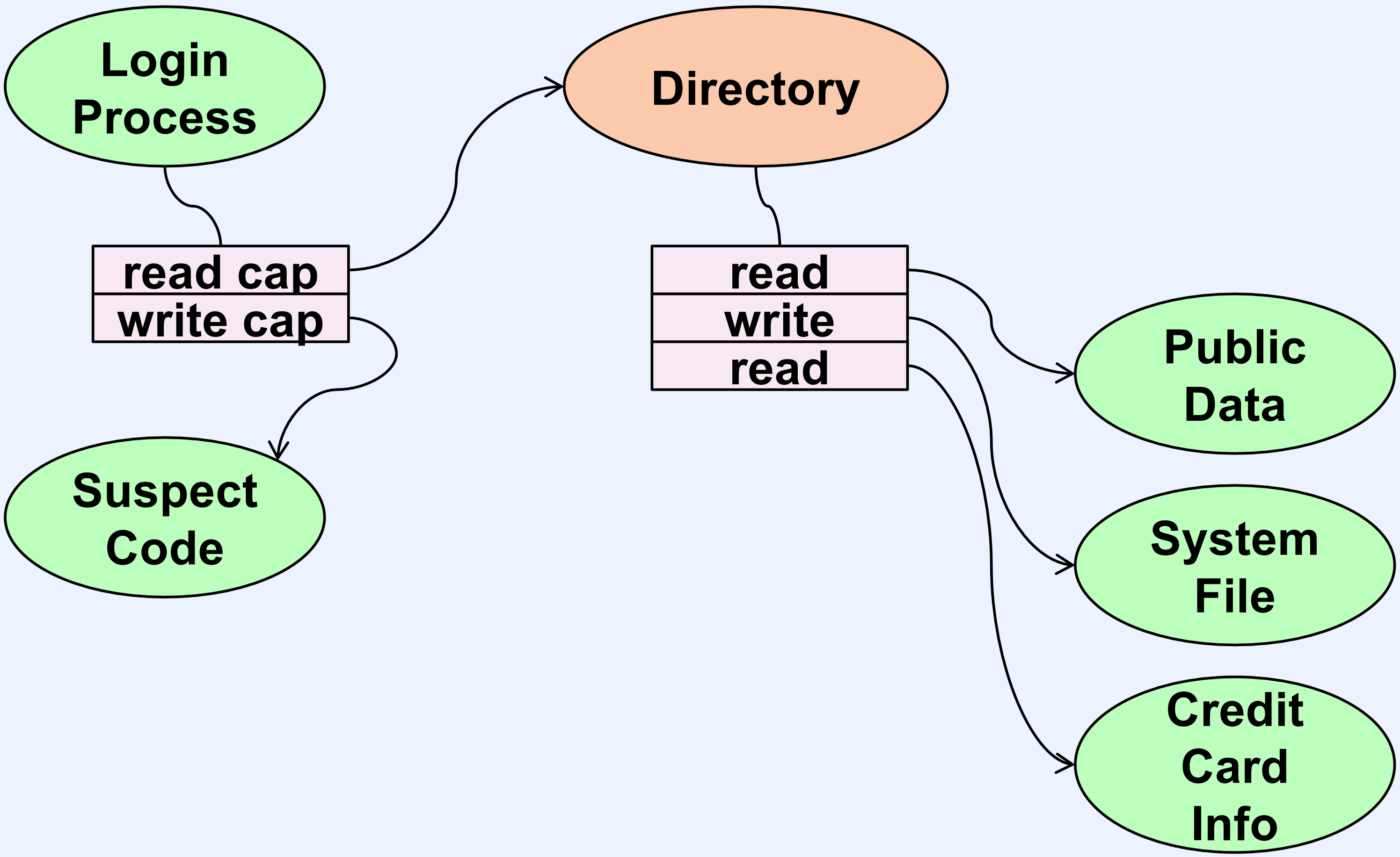
Copying Capabilities (2)



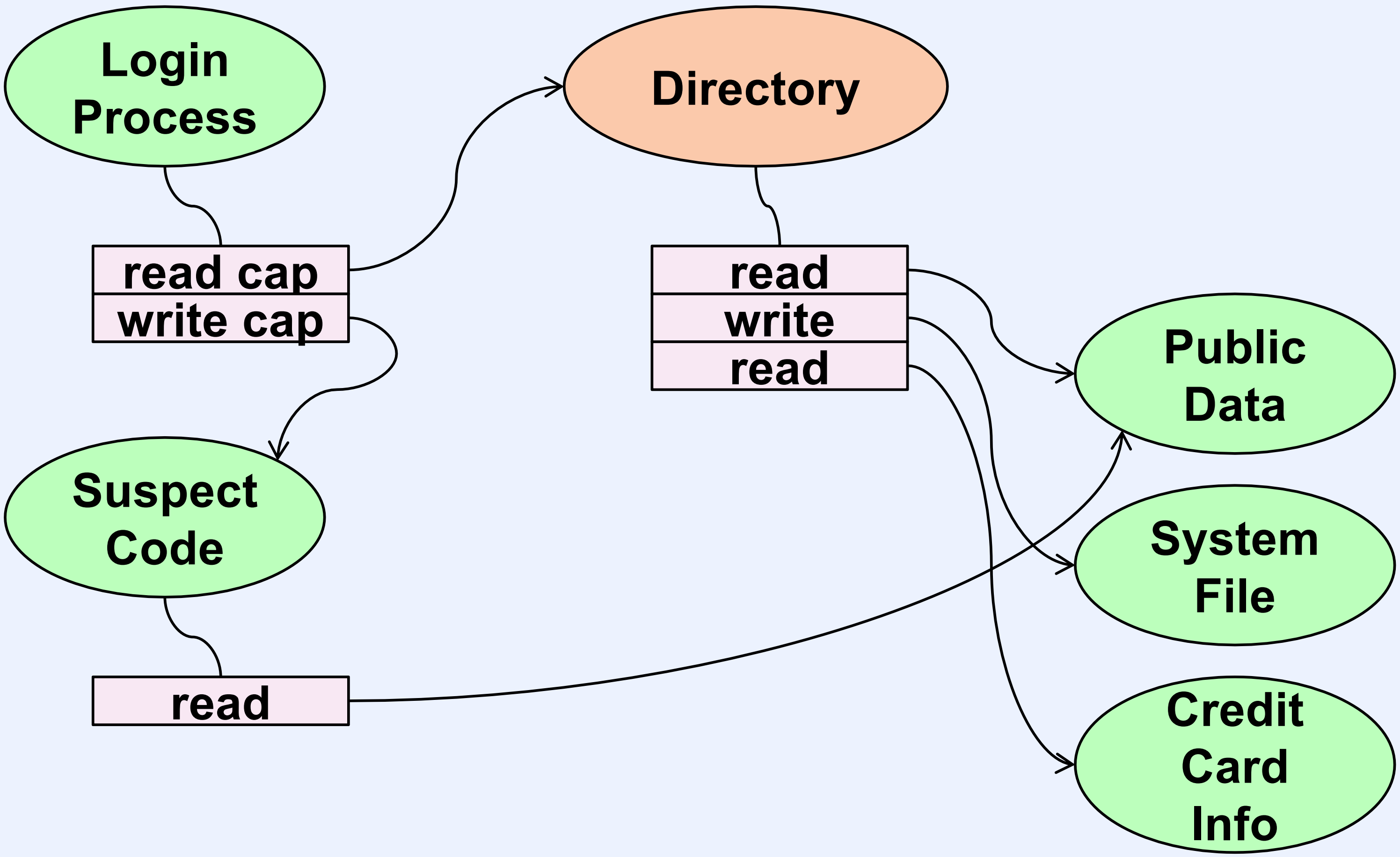
“Directories”



Least Privilege (1)



Least Privilege (2)

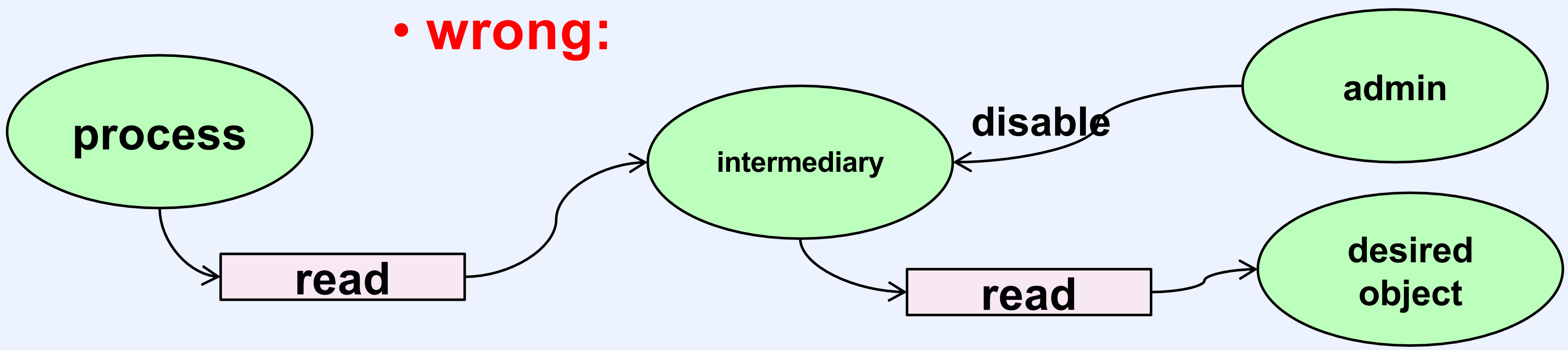


Issues

- Files aren't referenced by names. How do your processes get capabilities in the first place?
  - your "account" is your login process
    - created with all capabilities it needs
    - persistent: survives log-offs and crashes

Issues

- Can MAC be implemented on a pure capability system?
  – proven impossible twice
    - capabilities can be transferred to anyone
      - wrong: doesn't account for write-capability and read-capability capabilities
    - capabilities can't be retracted once granted
      - wrong:



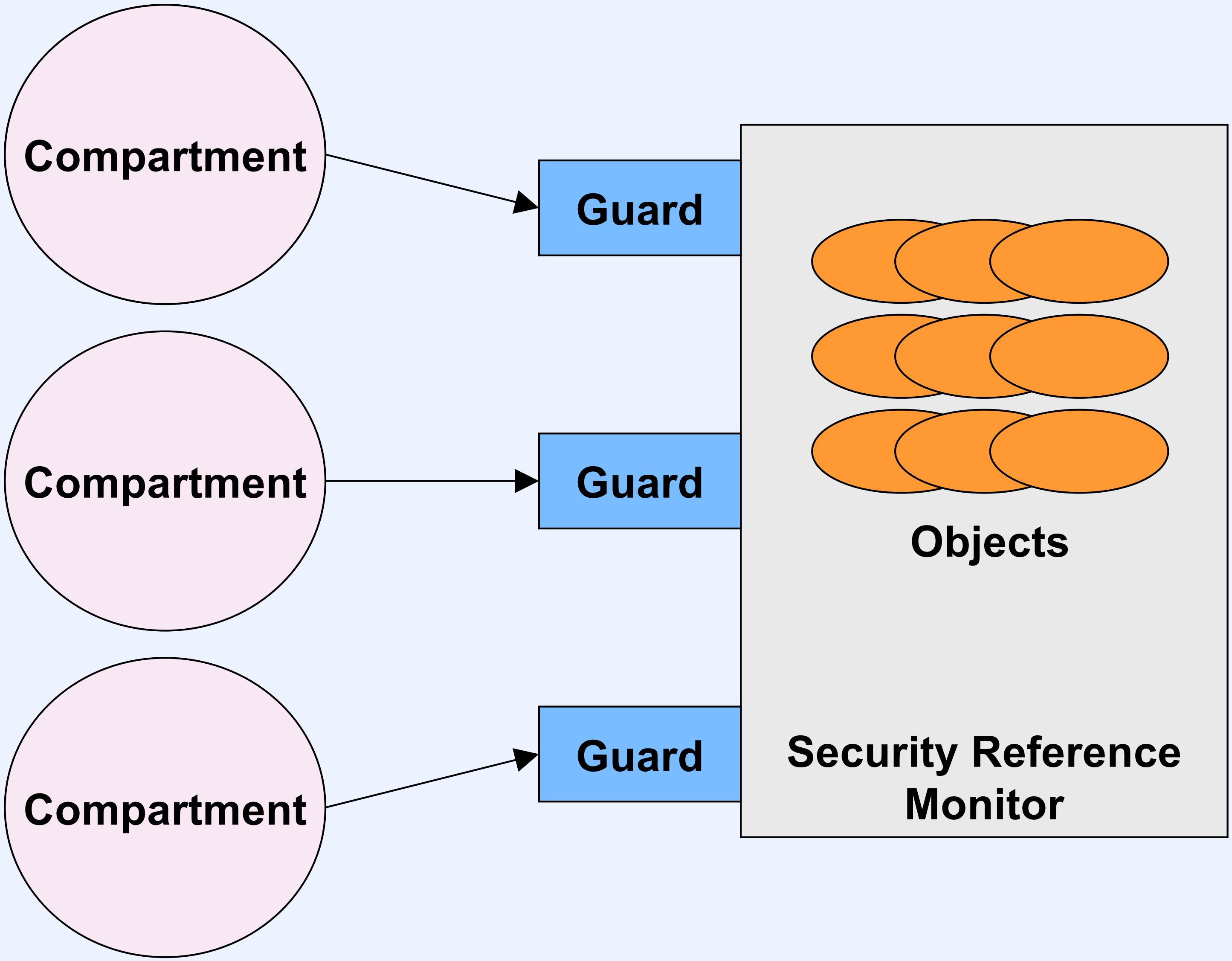
Do Pure Capability Systems Exist?

- Yes!
  - long history
    - Cambridge CAP System
    - Plessey 250
    - IBM System/38 and AS/400
    - Intel iAPX 432
    - KeyKOS
    - EROS
    - CHERI

A Real Capability System

- KeyKOS
  - commercial system
  - capability-based microkernel
  - used to implement Unix
    - (sort of defeating the purpose of a capability system …)
  - used to implement KeySafe
    - designed to satisfy "high B-level" orange-book requirements
    - probably would have worked
    - company folded before project finished

KeySafe


Compartment
Compartment
Compartment
Guard
Guard
Guard
Objects
Security Reference Monitor

Speculative Execution Attacks

An Important Assumption

- Pages cannot be read if read permission is off
  – on Intel x86-64
    - rings 0-2 can read all mapped pages
    - ring 3 can read only those pages for which read permission is explicitly given (in the page-table entries)

What If the Assumption is Wrong?

- **Code in ring 3 can read everything that's currently mapped**
- **In Linux, all physical memory is mapped into kernel's address space**
  - **kernel address space mapped into every process**
  - **every process can read all physical memory**

Making the Assumption True

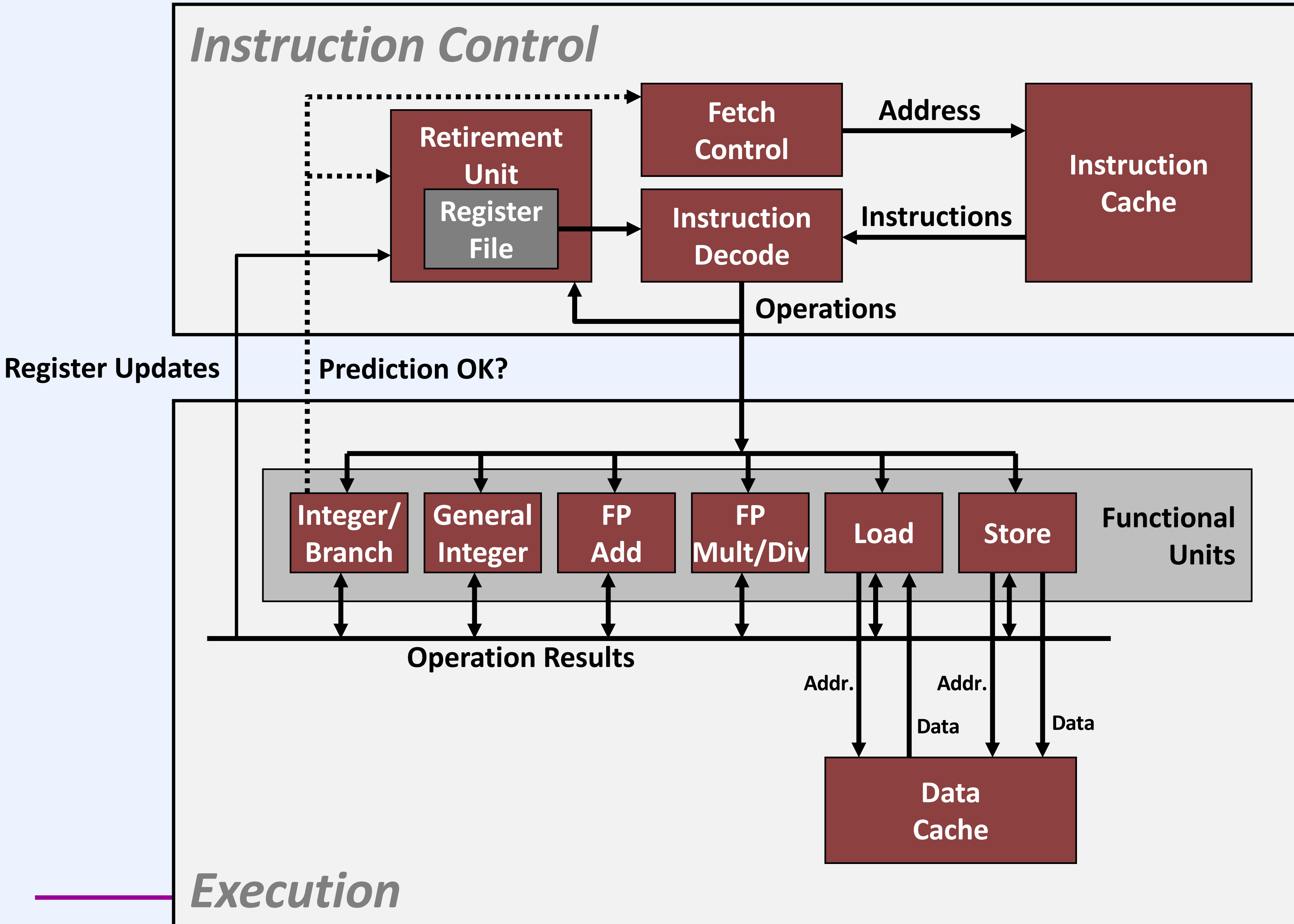
- Processor checks page protection on each access of memory
  - a fault occurs and access is not allowed if page is marked not readable

Speculative Execution

```
1:   movl   $0x1,%ecx
2:   xorq   %rdx,%rdx
3:   cmpq   %rsi,%rdx
4:   jne    someplace_else
5:   movl   %esi,%edi
6:   imull  (%rax,%rdx,4),%ecx
```

perhaps execute these instructions

Modern CPU Design



Speculative Execution

- **If speculatively executed instruction turns out to be used (branch not taken), its results are *retired***
  - **e.g., if a load into a register, the register is updated**
  - **if speculatively executed instruction results in exception, exception is taken**
- **If not used (branch taken), register is not updated**
  - **exceptions are ignored**

Micro Operations

- Machine instructions are actually composed of micro operations, which can be executed concurrently by the various functional units
- To fetch from memory into register, (at least) two micro operations are used:
  - load value from memory
    - value goes into cache
  - check if access is allowed
    - if not, register not updated
    - exception occurs
      - unless executed speculatively and instruction not used

Speculative Execution

```
1:   movq    kernel_address,%rcx
2:   movq    random_value,%rdx
3:   cmpq    %rcx,%rdx
4:   jne     someplace_else
5:   movb    (%rcx),%al
```

jump is taken

No exception, %rax not modified, but cache is updated

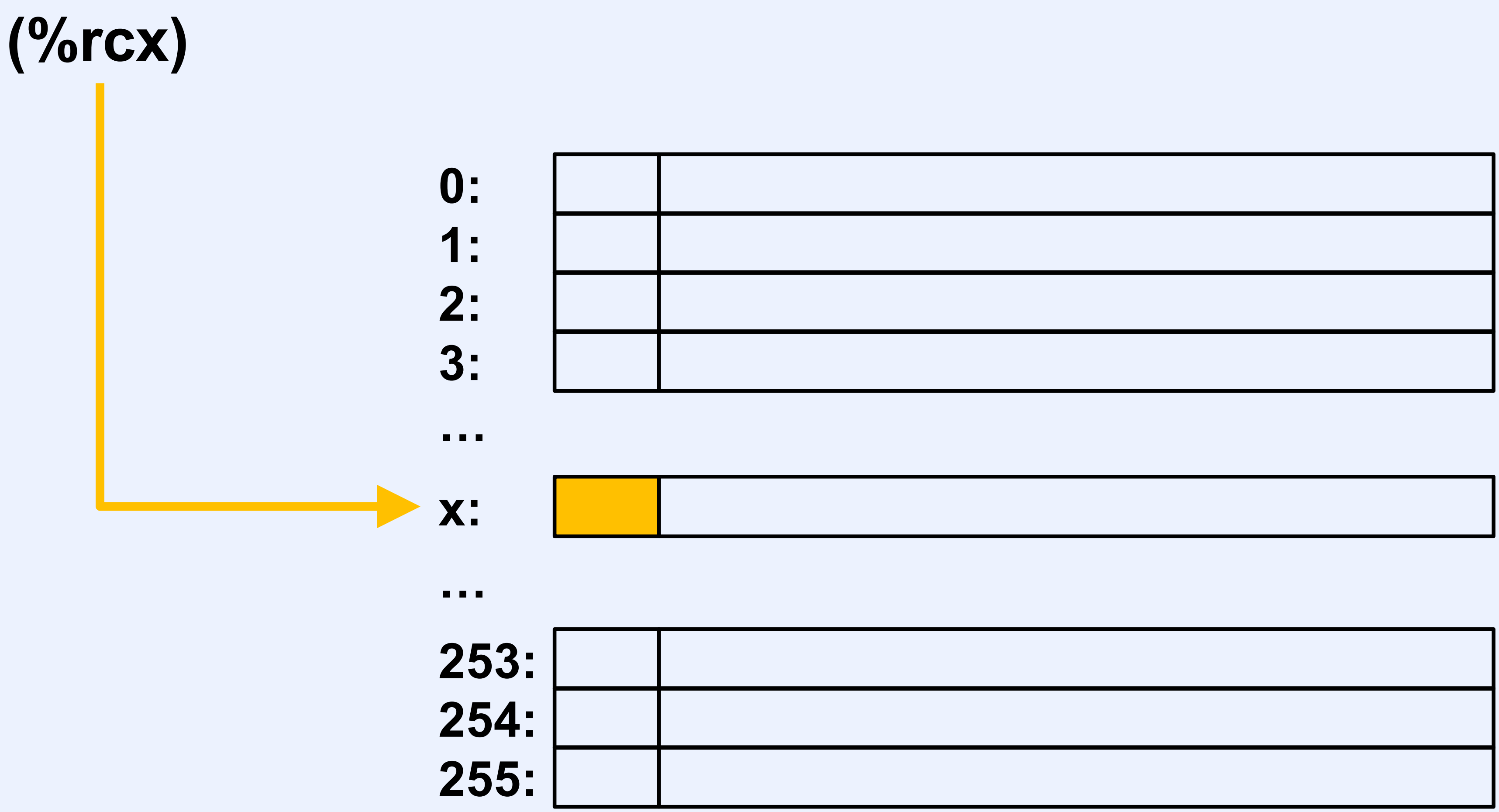
Not a security problem, because there's no way to determine what went into the cache.

Correct?!?!

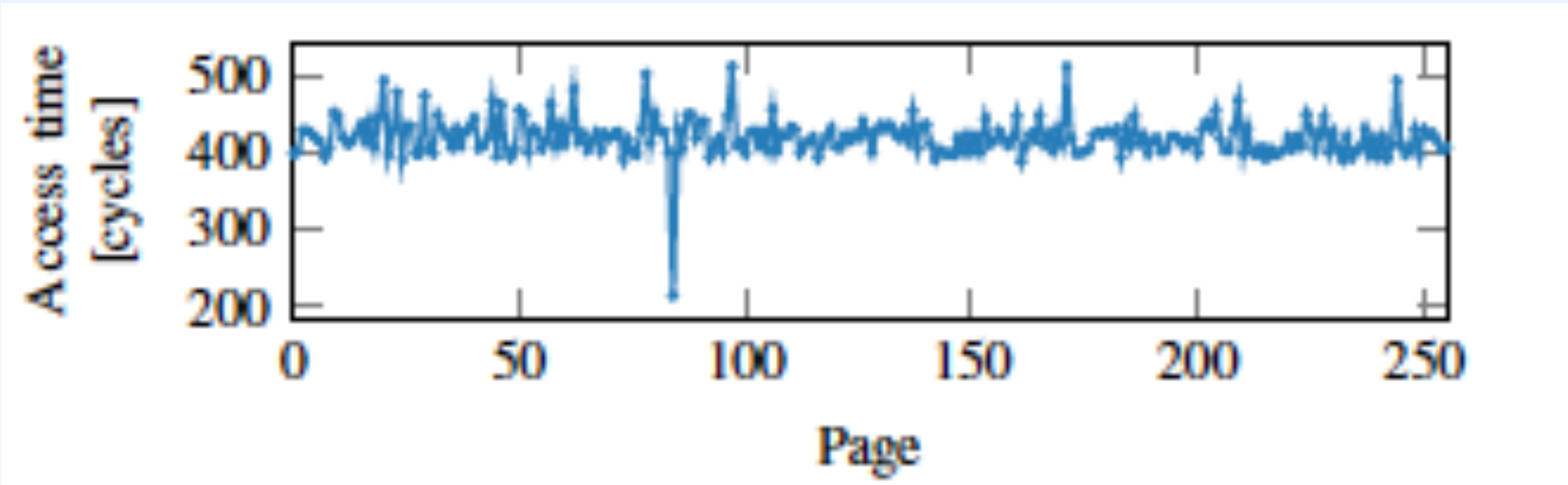
Speculative Execution

```
1:   movq   kernel_address,%rcx
2:   movq   random_value,%rdx
3:   cmpq   %rcx,%rdx
4:   jne    someplace_else
5:   movb   (%rcx),%al
6:   shl    %rax,$0xc    # multiply by 4096
7:   movq   %rax,0(%rbx,%rax)
            # %rbx contains the address of a
            # 1MB (256 x 4096 byte) array in
            # user-accessible memory
```

The Array



Accessing the Array



Speculative Execution

```
movq    kernel_address,%rcx
movq    random_value,%rdx
cmpq    %rcx,%rdx
jne     someplace_else
movb    (%rcx),%al
shl     %rax,$0xc    # multiply by 4096
movq    %rax,0(%rbx,%rax)
        # %rbx contains the address of a
        # 1MB (256 x 4096 byte) array in
        # user-accessible memory
```

Can we be sure processor won’t be able to guess whether jump is taken?

Alternative Approach

```
1:   movq   kernel_address,%rcx
2:   movq   random_value,%rdx
3:   cmpq   %rcx,%rdx
4:   movb   $1,0          # cause a segfault
5:   movb   (%rcx),%al
6:   shl    %rax,$0xc     # multiply by 4096
7:   movq   %rax,0(%rbx,%rax)
            # %rbx contains the address of a
            # 1MB (256 x 4096 byte) array in
            # user-accessible memory
```

Kernel Address Mapping

- On Linux and OSX
  - kernel is mapped into every user process
  - all physical memory is mapped into kernel
  - thus all physical memory can be accessed via speculative execution

Meltdown

- Can read all of kernel memory on a Linux machine at the rate of 503 KB/sec
  - it's not dependent on software bugs: works on a bug-free kernel
  - achieved by using Intel TSX so that the speculatively executed instructions are not retired
  - slows down to 122 KB/sec if segfaults are used

Countermeasures

- KASLR
  - kernel address-space layout randomization
  - kernel starts at a randomly chosen address
    - 40 bits of randomization
    - try all possibilities …
- KAISER
  - "kernel address-space isolation to have side channels efficiently removed"
  - kernel address space is not mapped when in user mode
  - prevents meltdown!

Quiz 4

As stated, KAISER requires that there are no mappings of kernel virtual addresses when the processor is running in ring 3 (user mode).

a) This is completely innocuous and has no adverse effects

b) This may “break” some otherwise correct user code

c) This may have noticeable performance effects

d) Both b and c are true

Other Similar Attacks (1)

- RIDL: Rogue In-Flight Data Load
  - certain processor buffers hold data coming from memory
    - loaded by victim thread
  - before doing address translations and protection checks, hardware guesses that it might be what's needed by attacker thread
    - attacker puts value in array (and thus in cache)
  - hardware determines addresses are different and does not retire results
    - attacker determines value by access times in array
  - not prevented by Kaiser

Other Similar Attacks (2)

- **Fallout**
  - stores deposited in store buffer
    - allowing thread to continue executing without waiting for memory to be updated
  - subsequent loads check store buffer to see if it contains data to be loaded
  - address/protections checks are optimistic
  - attacker can determine what data is being stored by victim

Summary

- Attacks rely on extreme optimization done in hardware
  – depend on detailed knowledge of hardware architecture
  – much of this was learned via reverse engineering
- Attacks allow one to extract information across virtual machine and container boundaries

