

9/23/26

COMP 435:

Computer Security Concepts



Lecture 14: Operating Systems Security



The University
of North Carolina
at Chapel Hill

<https://tinyurl.com/comp435-fa26>

Logistics Update



- First lab due tomorrow! 😊
- Quiz Friday!
- Quiz practice and solutions posted

Quiz Topics



- Hardware as the root of trust
 - trusted vs. trustworthy, TCB
- Hardware bugs & errata
 - specifications, implementations, security consequences
- Security properties
 - writing simple properties, counterexamples,
- Information flow
 - explicit vs. implicit flow; what security primitives may be violated with unintended/indirect flows
- Caches & timing side channels
 - Conceptually: the mechanisms required for a timing side channel
 - Understand how timing observations can be used to infer secret information
- Speculative execution & Spectre
 - Steps of the attack and why each step is necessary

Plan for Class Today



- 10:10-10:35 – OS Background
- 10:35-11:00 – OS Security, Race Conditions

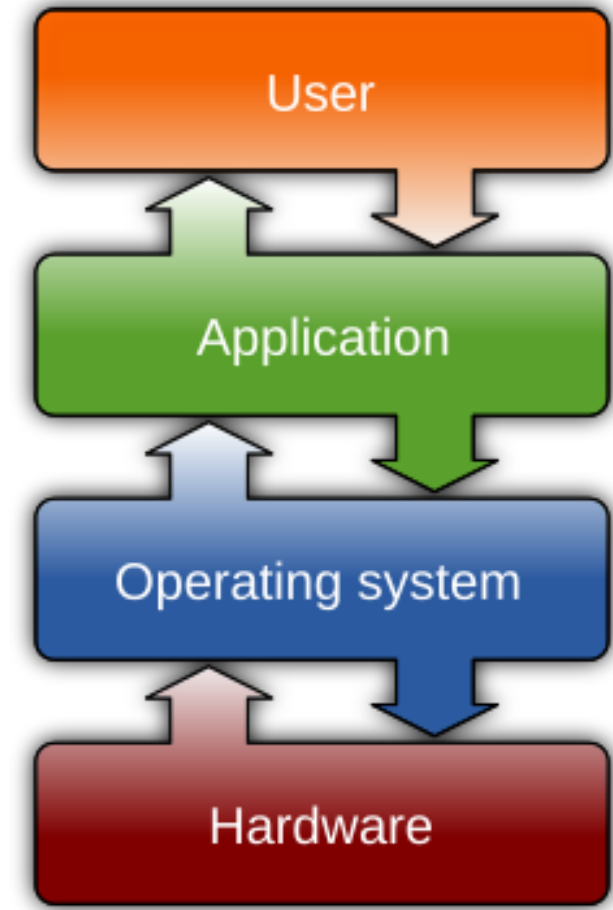
Guiding Question 😊

How does the operating system create isolation and manage access?

What is the Operating System??



Software that **manages** hardware and software **resources**, and **provides common services** for software programs

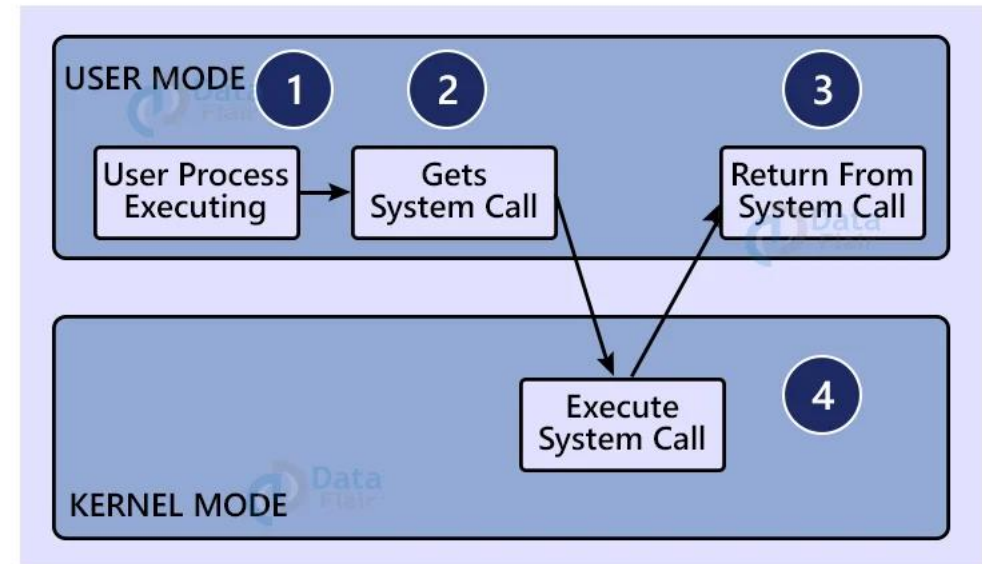


What is a System Call??



A system call (syscall) is the mechanism by which a program in user space asks the operating system to do something on its behalf.

WORKING OF A SYSTEM CALL



Source: <https://data-flair.training/blogs/system-call-in-os/>

Source:
https://en.wikipedia.org/wiki/Operating_system

Important Note: User Mode vs. Kernel Mode



- CPU can operate in various modes when executing instructions
- Most common:
 - User mode
 - Supervisor mode (or kernel mode)
- When executing code that makes up the OS, **AKA system-level code**, CPU in supervisor mode
 - We have access to more instruction types, more memory ranges
- Processor should never execute user-provided code with supervisor-level privileges
- **System calls** are the mechanism for mode transition!



OS Responsibilities

- manage resources
 - `fork()`, `mmap()`
- provide interface to HW
 - `read()`, `write()`
- provide user authentication
 - `getuid()`, `setuid()`
- mediate interprocess communication
 - `pipe()`
- protect itself
 - `kill()`, `mprotect()`

Manage Resources



- allocation
- sharing
- protection — authorization



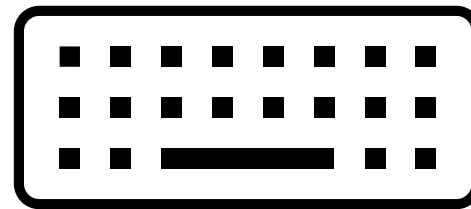
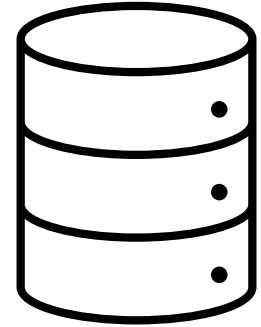
memory,
disk space
access to device

✓ Access condition ✓

Provide Interface to Hardware



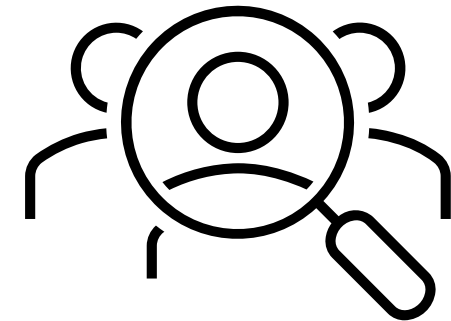
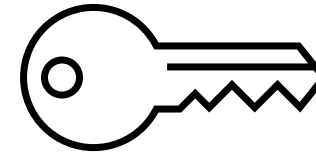
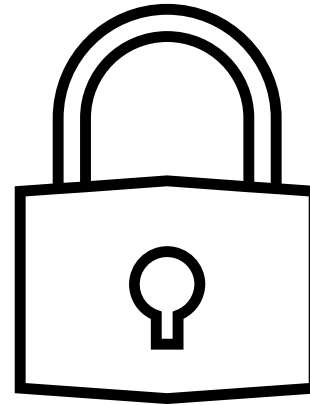
- memory
- file system
- device I/O



Provide User Authentication



- user accounts
- roles
- passwords

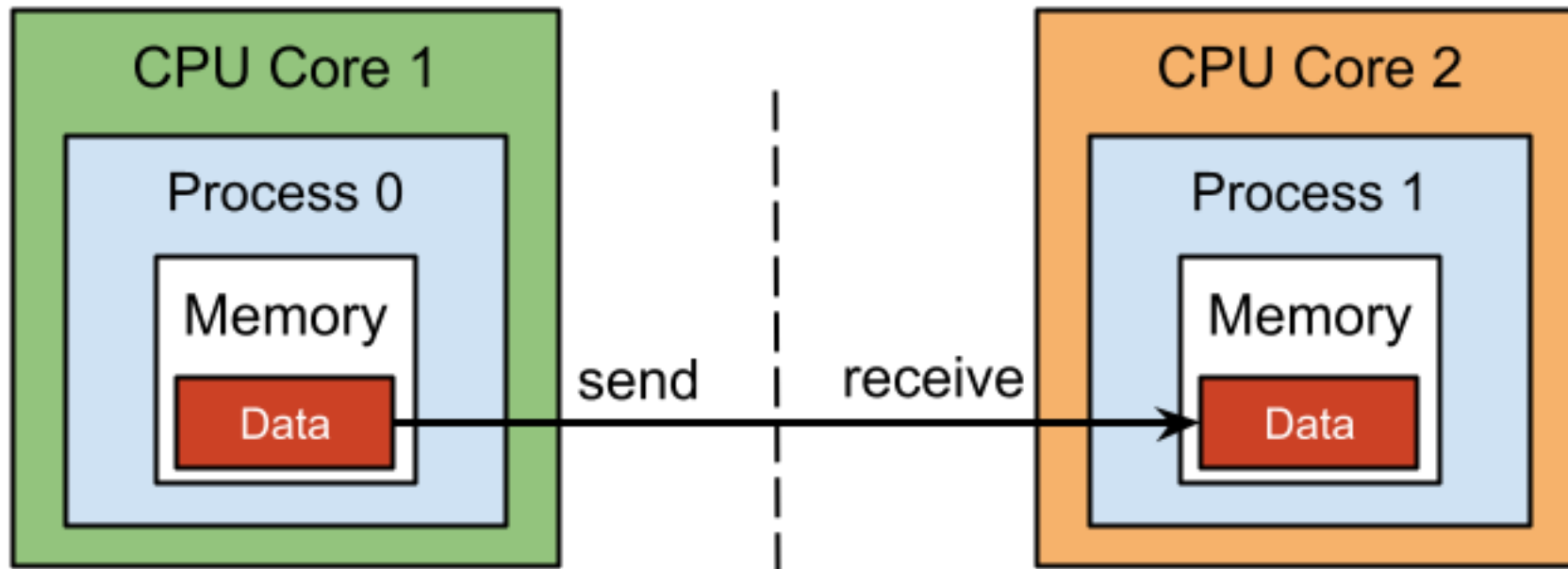


password
file!



Mediate Interprocess Communication

- *Interprocess communication facilitated by OS ✓*
- shared memory
 - message passing





Protect the Operating System (Itself!)

- data
- code
- permissions

The data structures and code the OS uses to enforce proper access control, process separation and fair time sharing all reside in memory →

need to be protected from other code running on the machine!

Features of a Trusted System



- ***TCB***
- trusted path
- secure start-up
- object sanitization
- auditing



should be small and well defined



Features of a Trusted System

- TCB
- ***trusted path***
- secure start-up
- object sanitization
- auditing



authenticates the
OS to the user



Features of a Trusted System

- TCB
- trusted path
- ***secure start-up***
- object sanitization
- auditing



starts in a known
good state



Features of a Trusted System

- TCB
- trusted path
- secure start-up
- ***object sanitization***
- auditing



no object reuse
(free your memory!)



Features of a Trusted System

- TCB
- trusted path
- secure start-up
- object sanitization
- ***auditing***



audit log tracks any
changes

UNIT 1: TRUSTING THE MACHINE

Memory Separation

Styles of Separation



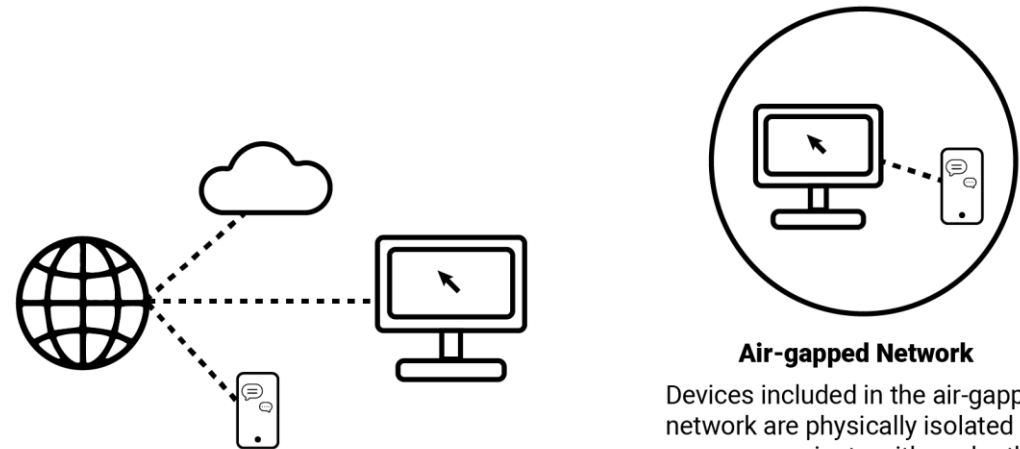
- Physical
- Temporal
- Logical
- Cryptographic

Styles of Separation



- ***Physical***
- Temporal
- Logical
- Cryptographic

not efficient, not
complex, secure



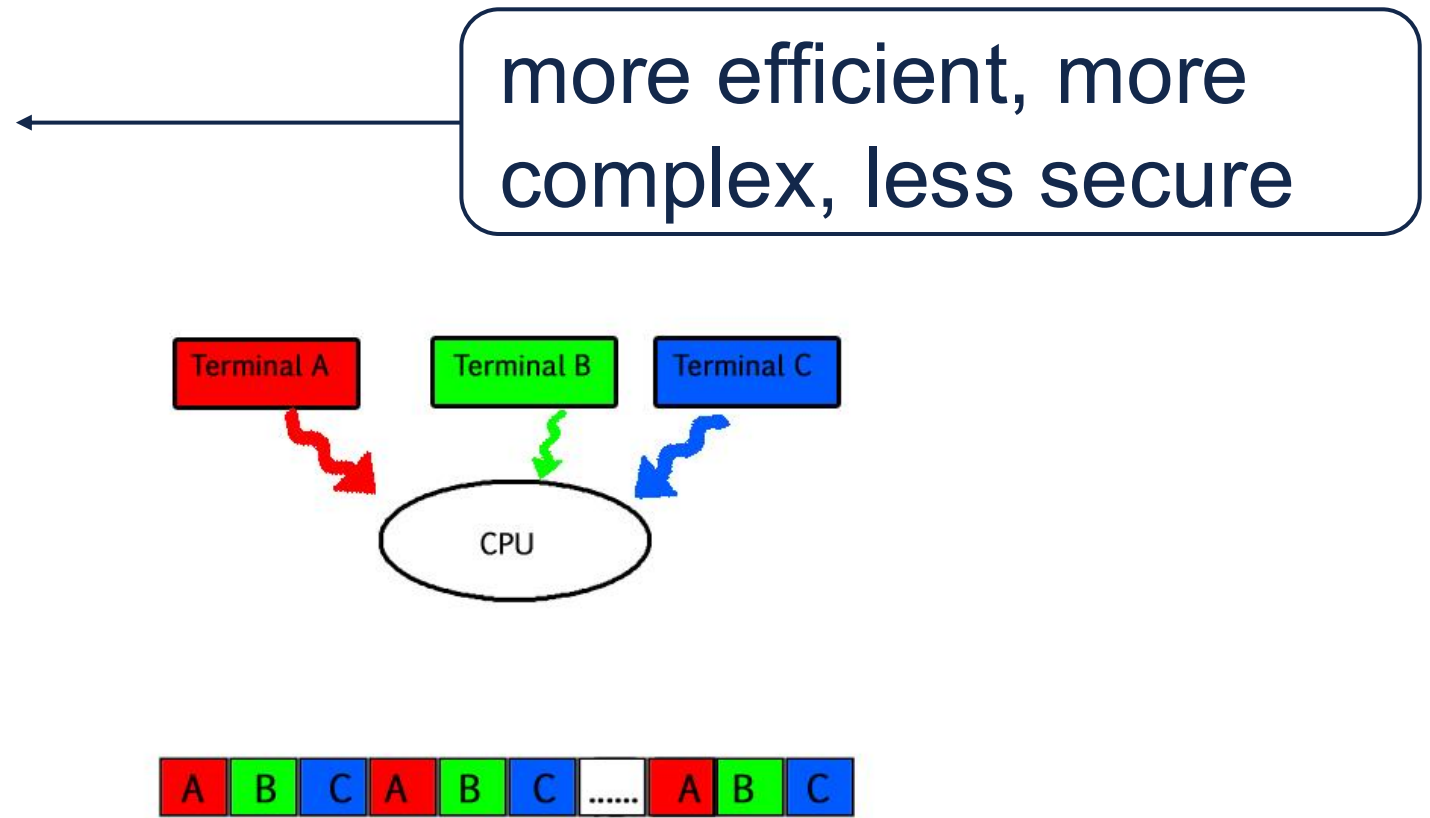
Air-gapped Network

Devices included in the air-gapped network are physically isolated and can communicate with each other, but cannot communicate with any other network outside of the air-gap.

Styles of Separation



- Physical
- **Temporal**
ex: time sharing,
object sanitization
- Logical
- Cryptographic



Time sharing!



Styles of Separation

- Physical
- ***Temporal***
- Logical
- Cryptographic

Object reuse requires sanitization!

c

 Copy code

```
#include <string.h>
#include <stdlib.h>

int main() {
    char *password = malloc(100);
    if (!password) return 1;

    // password is used here...
    strcpy(password, "SuperSecret123!");

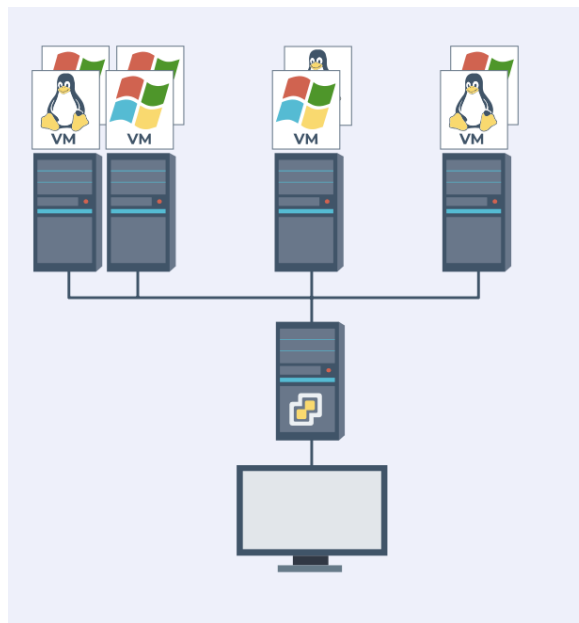
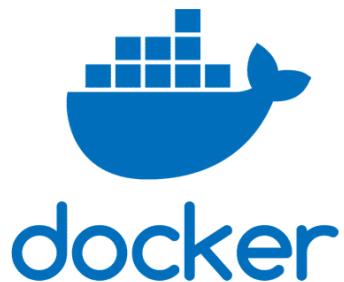
    //  Object sanitization: overwrite before freeing
    memset(password, 0, 100);

    free(password);
    return 0;
}
```

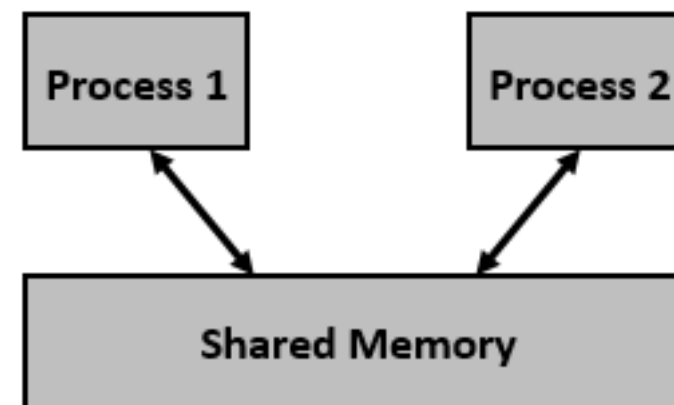


Styles of Separation

- Physical
- Temporal
- **Logical**
aka algorithmic
- Cryptographic



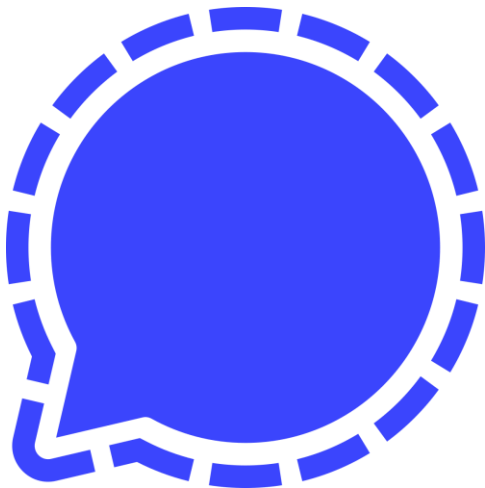
more efficient, more complex, less secure





Styles of Separation

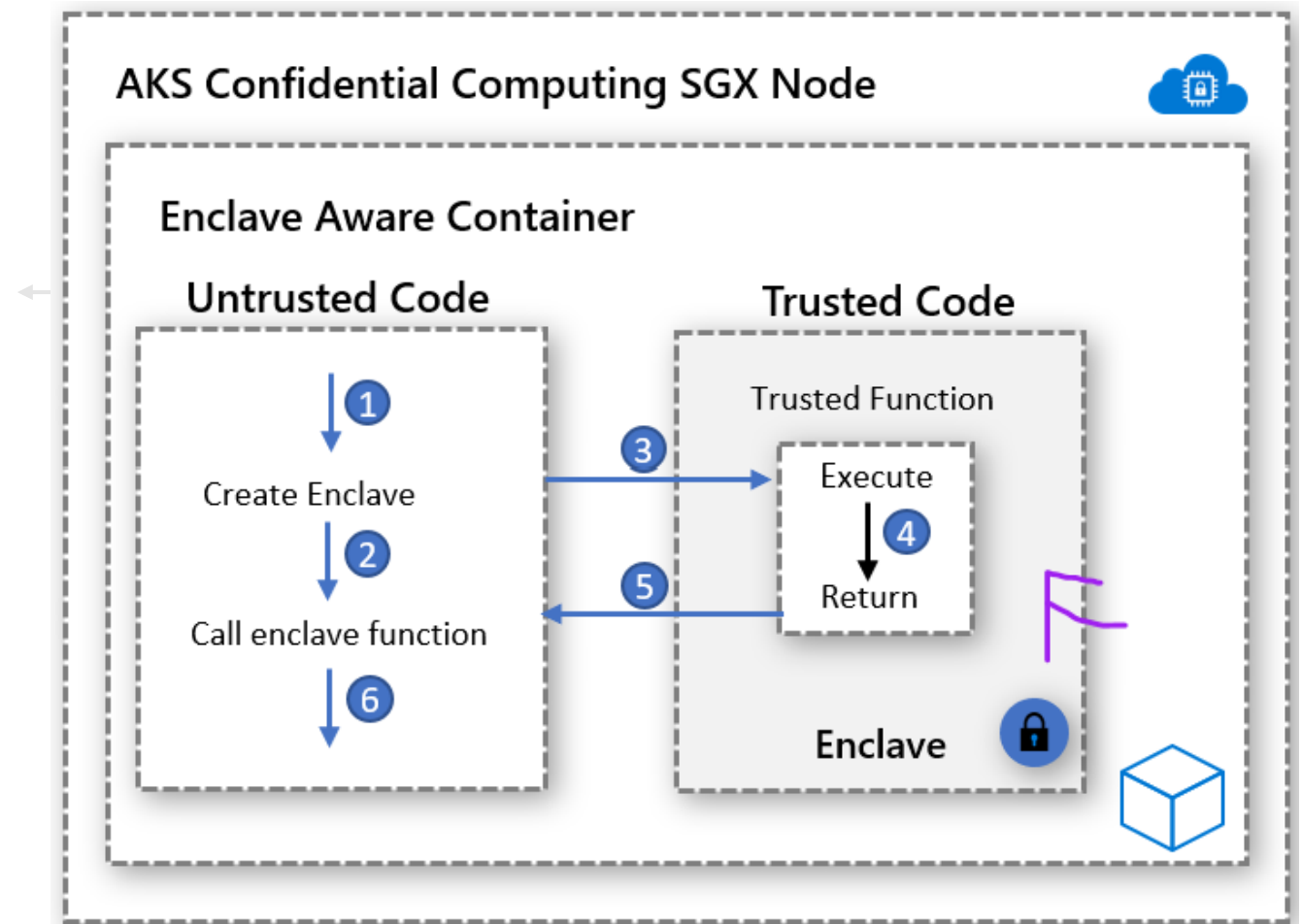
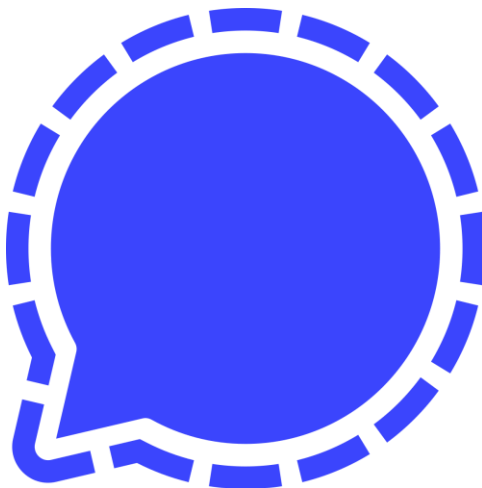
- Physical
- Temporal
- Logical
- ***Cryptographic***



Styles of Separation



- Physical
- Temporal
- Logical
- ***Cryptographic***



Memory Protection Mechanisms

OS Provided Memory Protection Mechanisms



- fence
- base--bounds registers
- segment addressing

Memory Fence

Physical Separation



fence



user
code

0xFF...FFF

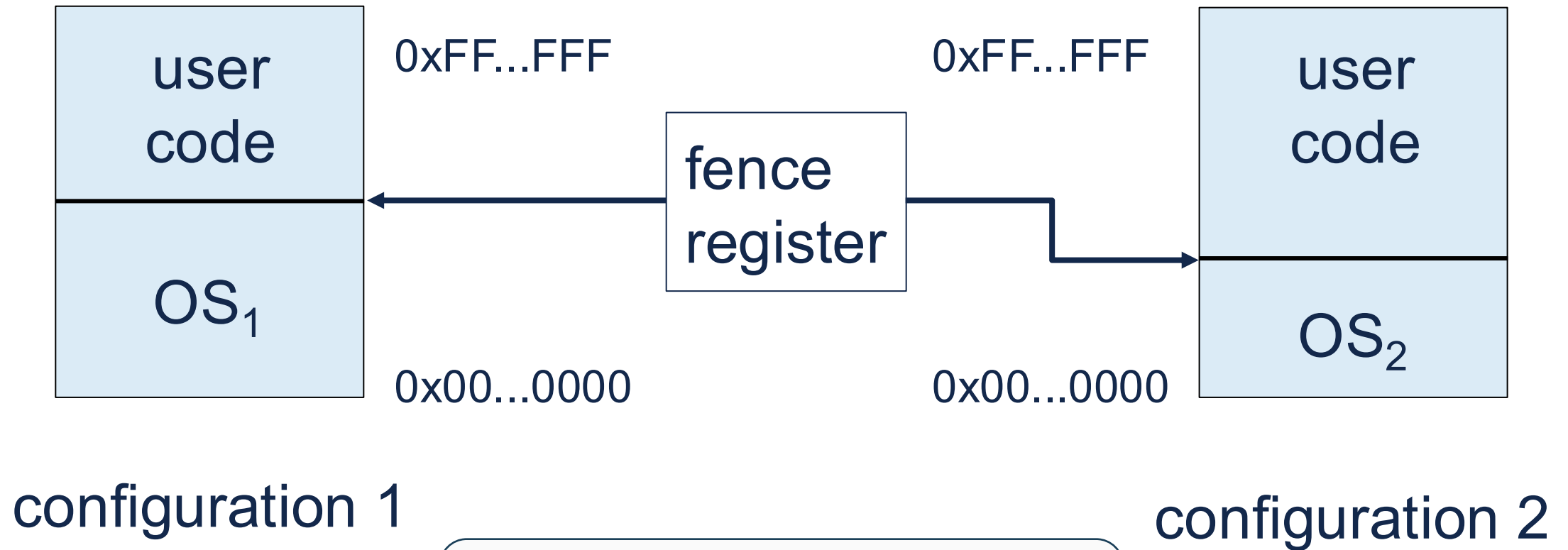
OS

0x00...0000

Fence: hardcoded
address in processor that
denotes starting point of
OS code

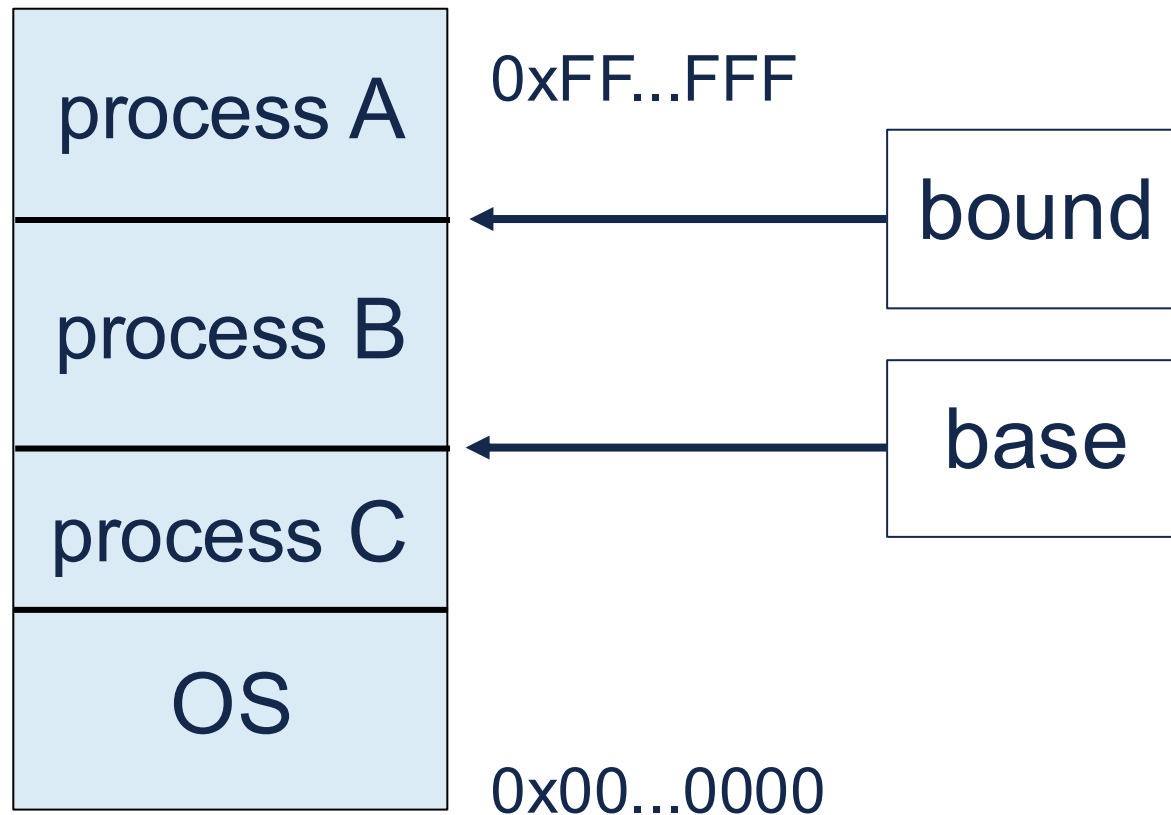
Memory Fence

Logical Separation



Fence now configurable!

Base–Bounds Registers



***Base & Bounds
Registers:*** prevent code
from one process from
interfering with w/ another

Segment Addressing



0	base addr	length	R	W	X	M	F
1	base addr	length	R	W	X	M	F
n	base addr	length	R	W	X	M	F

segment descriptors
for process A

Introduces Virtual Memory: gives each process an isolated view of memory

Allows for separate access permissions (read, write, execute) for a specified memory region

Segment Addressing

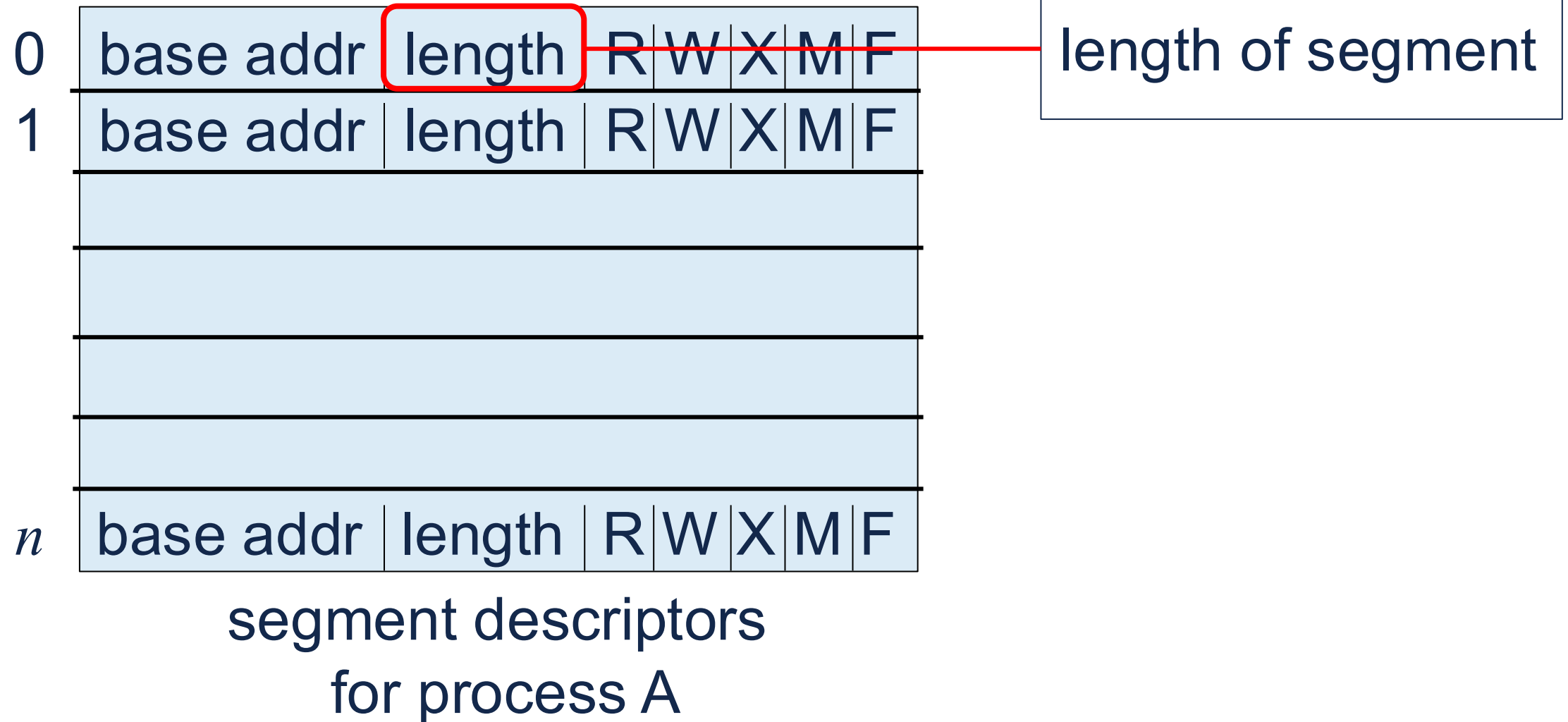


0	base addr	length	R	W	X	M	F
1	base addr	length	R	W	X	M	F
n	base addr	length	R	W	X	M	F

physical starting
address of segment

segment descriptors
for process A

Segment Addressing



Segment Addressing



0	base addr	length	R	W	X	M	F
1	base addr	length	R	W	X	M	F
n	base addr	length	R	W	X	M	F

access control
indicators

There is a bit for read,
write, execute, mode
(execute as supervisor?),
fault

segment descriptors
for process A

Segment Addressing



0	base addr	length	R	W	X	M	F
1	base addr	length	R	W	X	M	F
n	base addr	length	R	W	X	M	F

segment descriptors
for process A

<segment #, offset>

addressing

Worksheet time! Q1-3





Race conditions

A type of vulnerability!

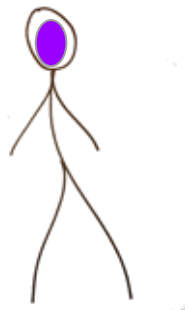


Race Condition

Def'n: Concurrent access of a resource is not serializable

Occurs when:

- Two or more processes have access to **the same resource**
- The final state of the resource depends on the **relative timing** of the **two processes**



ATM



Bank



balance = 900

withdraw \$100



get_balance()



balance = 900



\$100



set_balance(800)



balance = 800



balance = 900

w/d \$100



get_bal()



bal = 900



get_bal()



bal = 900



w/d \$50



set_bal(800)



set_bal(850)



\$50



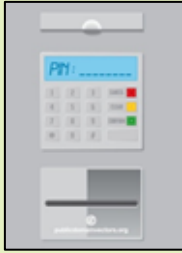
\$100



balance = ??



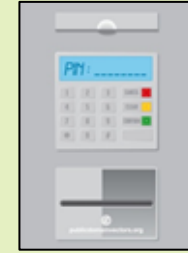
ATM



Bank



ATM



balance = 900

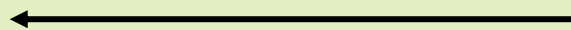
w/d \$100



get_bal()



bal = 900



\$100



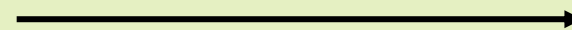
set_bal(800)



get_bal()



bal = 900



w/d \$50



set_bal(850)



\$50



balance = ??

Problem!!



- The bug is **not**: “two users accessed the account at once”
- The bug is that the supposedly single operation **withdraw** was implemented as multiple interleavable operations.
 - Caused inconsistent behavior!!
- **Potential solution:**
 - Use locking/transactions so only one withdrawal updates the balance at a time