

CSCI 2410 Systems Programming

Fall 2026

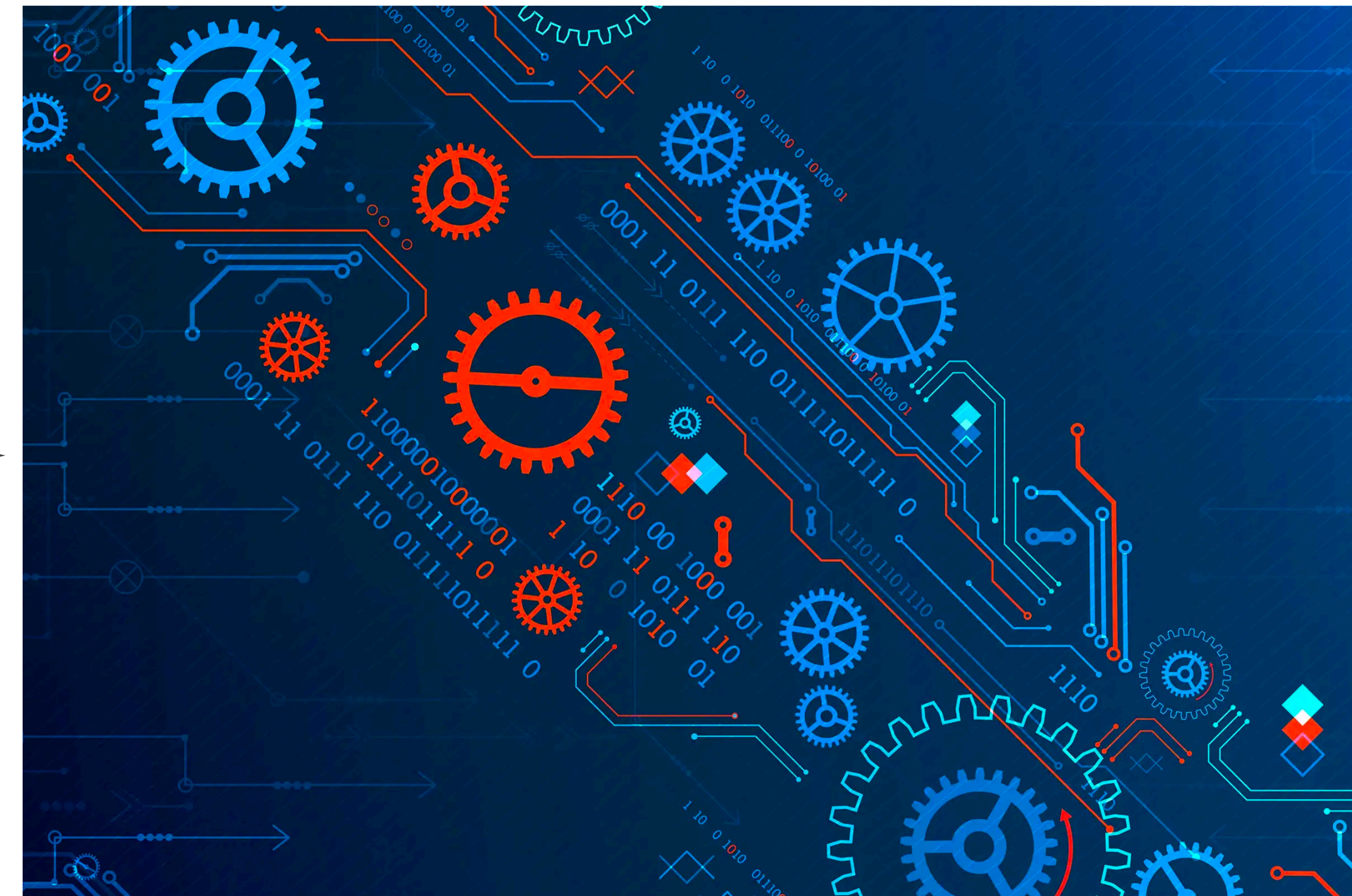
Instructor: Jie Zhou

Department of Computer Science
George Washington University



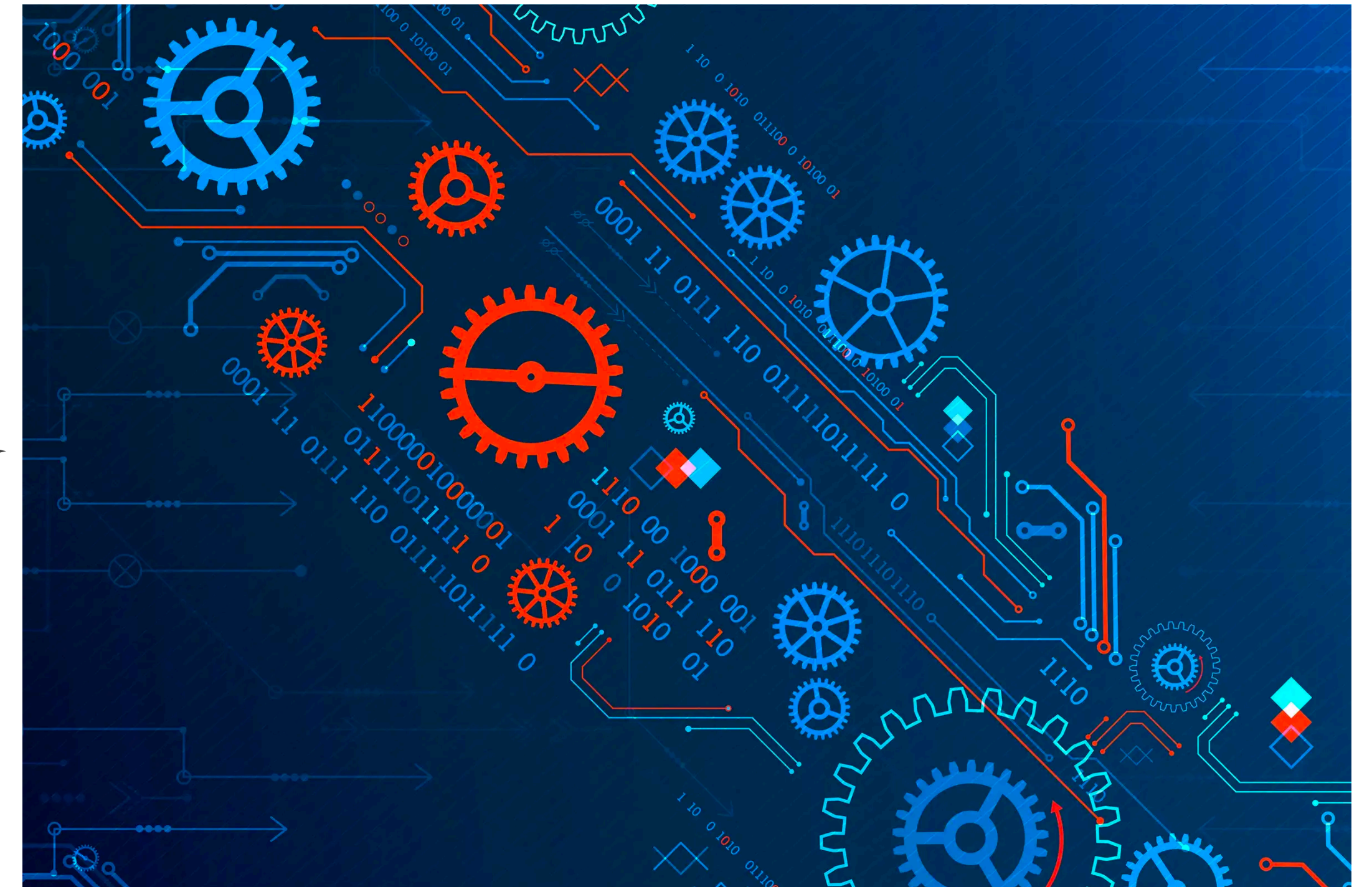
Slides materials are partially credited to Sibin Mohan and Gabriel Parmer at GWU.

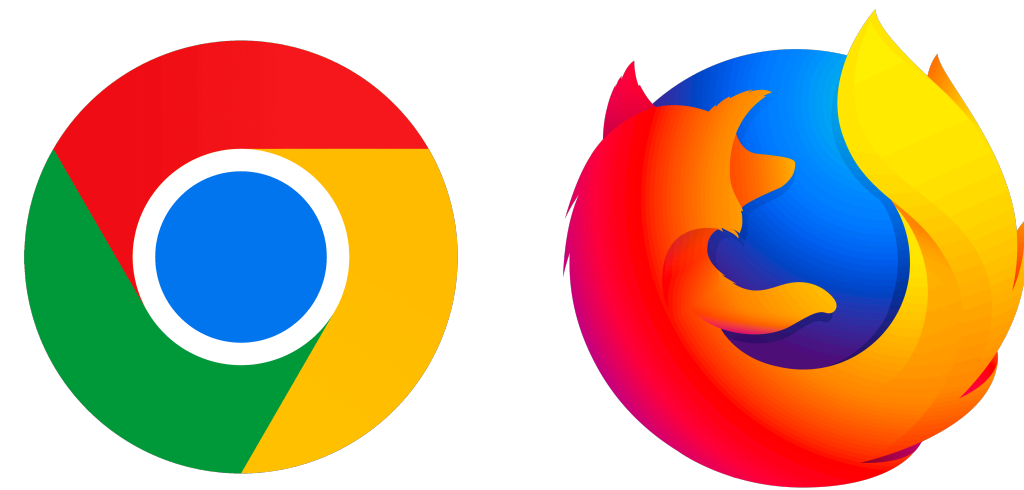
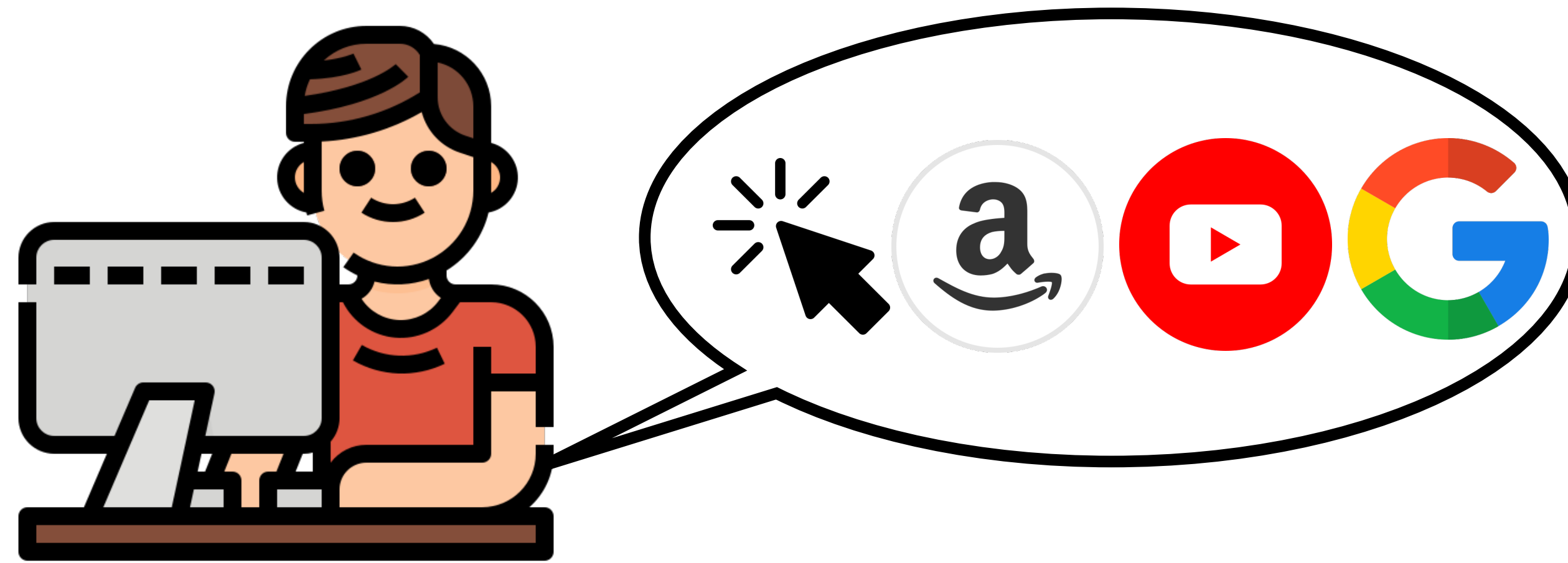
Why learn Systems Programming?





Systems Software





Browsers



Operating Systems



Web Servers

The digital world runs on systems software.

Teaching Staff

- Instructor: Jie Zhou
 - Assistant Professor in Computer Science
 - Research area: systems and software security, compilers, program analysis
- Two GTA



Gustavo Londono



Vineetha Paruchuri

Teaching Staff

- Two UTAs



Mack Rusing



Lydia Miller

Teaching Staff

- Two LAs



Sonia Lerner



George Cole

Course Resources

- Website: <https://jiezhoucs.github.io/courses/csci-2410/fall-2026/>
- Discord channel (sent in Blackboard)
- Each TA holds two 2-hr office hours weekly
- All teaching staff available in Discord
- Linux server: <http://ubuntu-vlab01.seas.gwu.edu/>
 - You can access to this machine using your GW credentials; needs VPN.

Course Structure

- Lectures
 - Mix of C/programming/systems concepts
 - You need to spend time to learn C syntax.
 - In-class programming exercises
- Labs
 - Hands-on exploration of specific topics
 - tools (gcc, valgrind, gdb)
 - interfaces (git, ssh, etc.)
 - Concepts related to homework assignments
 - Two sessions on Wed and Fri. Go to either one.

Homework Assignments

- Three main take-home programming homework
- Three warm-up homework to get you up to speed with the main ones
- Due 8 pm EST of the due day
- Grading:
 - Script-based auto grading
 - Extra credit for finishing early
 - You should test your code carefully!
 - Contact Gustavo for grading issues; if unresolved, contact the professor.
 - Provide very specific reasons for why you think the grading is incorrect.
 - Unacceptable reasons: forgot to submit, submitted the wrong version, “It works on my computer”, etc.

Assessments

- One coding assessment for each concluded homework
- No Internet access
- Must be completed in lab sessions

In-class Programming Exercises

- Similar to in-lab assessments, but the teaching staff will help you.
- Five in total
- Due at the end of the lecture, but you can submit it by the end of the day with 20% penalty.

Quizzes

- In-class, 11 in total
- 5–10 minutes at the beginning of the lecture
- Multi-choice questions and short-answer questions on paper
- Open notes, but no electronic devices
- Lowest one dropped
 - If you miss a quiz, you can use the drop quota.

Grading

- Homework: 30%
- In-class exercise: 10%
- Quizzes: 20%
- Lab assessments: 30%
- Class Participation: 10% + 5%

Class Participation

- Actively engaging in lectures is important.
 - Ask and answer questions
- Engagement outside of lectures is also strongly encouraged.
 - The professor and TAs/LAs will remember you, and you will be rewarded.
- There are no stupid questions, as long as they show your genuine thinking.

Lateness Policy

- Request for extension must be submitted *before* deadline.
- Only acceptable excuses:
 - medical and family issues
 - must-attend events such as sports—need proof from your coach/related party
- Unacceptable excuses:
 - computer failing
 - lots of other assignments
 - having completed the work but forgetting to submit in time
 - others

Office Hour Decorum

- Be specific about your problems.
- Avoid vague questions/statements such as
 - “It does not work.”
 - “Does it work?”
 - “How to solve this assignment?”
- For programming questions, make sure
 - it compiles.
 - come with prepared tests.
 - properly comment the code and use good variable/function names.

Use of Generative AI



- Permitted for learning purposes (BE CAREFUL!)
- ***Prohibited*** from generating contents for assignments

Who to Contact for What

- Professor: lecture materials, general questions, challenges/problems, etc.
- All TAs/LAs: office hours & Discord for lecture materials and assignments
- Gustavo: homework grading
- Vineetha: quiz grading

Questions?

What Makes Up a “Computer”?

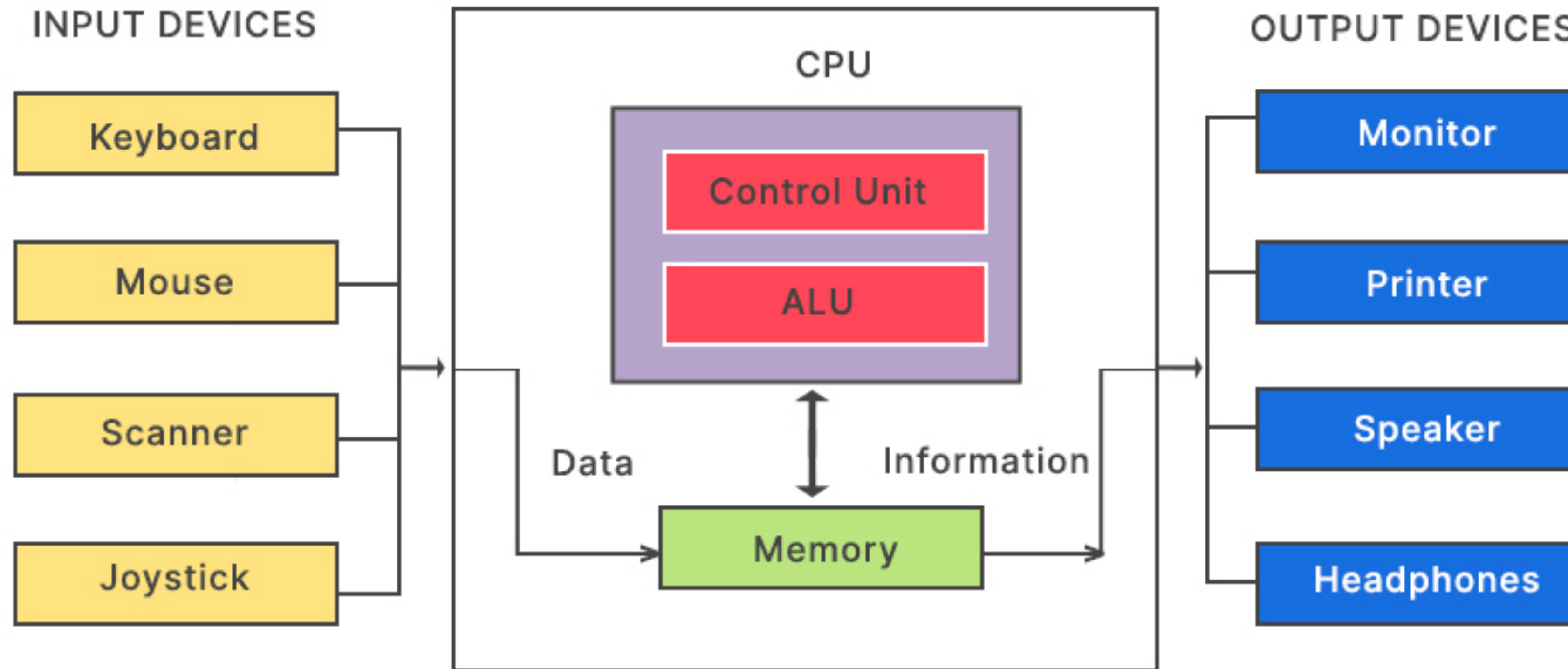
- Hardware

- CPU
- memory (RAM)
- hard disk
- monitor
- keyboard/mouse/trackpad
- GPU
- network card
- A lot more

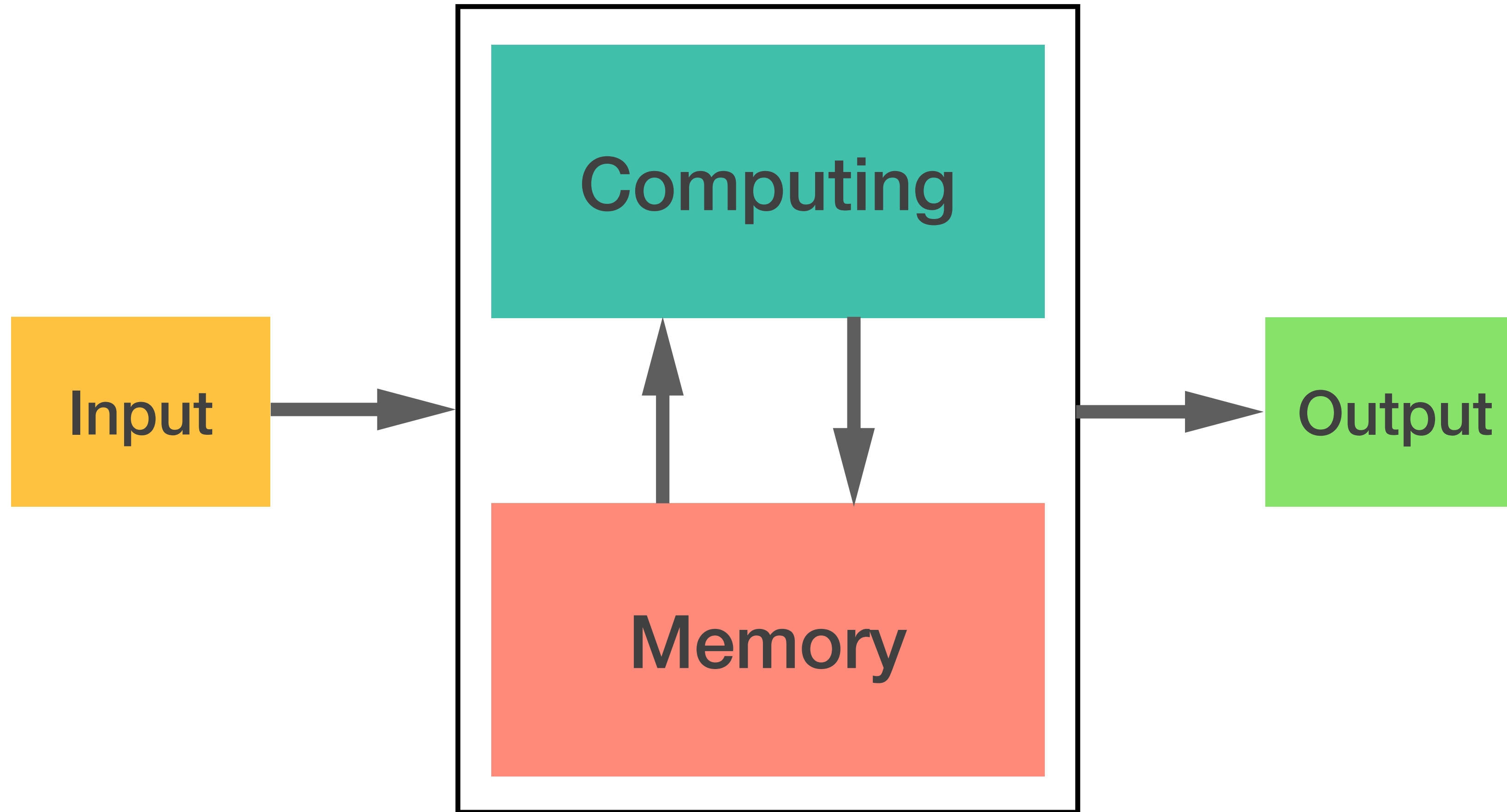
- Software

- operating systems (OS)
- device drivers
- libraries
- application software

Computer Organization



Architecture of Modern Computers



What Makes Up a “Computer”?

- Hardware

- CPU
- memory (RAM)
- hard disk
- monitor
- keyboard/mouse/trackpad
- GPU
- network card
- A lot more

- Software

- operating systems (OS)
- device drivers
- libraries
- application software

Operating Systems

- A software system—a “manager” software for other software
- Allows users/programs to
 - use hardware resources, e.g., CPU, GPU, disks
 - share resources, such as memory, across programs
 - interact with external devices, e.g., keyboard, joysticks,
- Additional features: security, reliability/fault handling, etc.

Goal: Ensure the entire system behaves *correctly* and *efficiently*.

Libraries

- Code written/packaged by others
- So we can reuse code/functionality.
- E.g., `java.util`, Python's `math`, `numpy`
- System libraries
 - provide core/common functionality
 - e.g., write to screen, read a file, copy data verbatim
 - relied by the standard libraries of most programming languages



Why do we have libraries?

Device Drivers

- How to use hardware?
 - e.g., GPU, keyboard
- Device drivers are software written to interact with hardware.
- Specific to a piece of hardware
 - manufacturer, OS, version, etc.

Linux x86_64/AMD64/EM64T

Latest Production Branch Version: 535.104.05

Latest New Feature Branch Version: 530.41.03

Latest Beta Version: 535.43.02

Latest Legacy GPU version (470.xx series): 470.199.02

Latest Legacy GPU version (390.xx series): 390.157

Latest Legacy GPU version (340.xx series): 340.108

Latest Legacy GPU version (304.xx series): 304.137

Latest Legacy GPU Version (71.86.xx series): 71.86.15

Latest Legacy GPU Version (96.43.xx series): 96.43.23

Latest Legacy GPU Version (173.14.xx series): 173.14.39

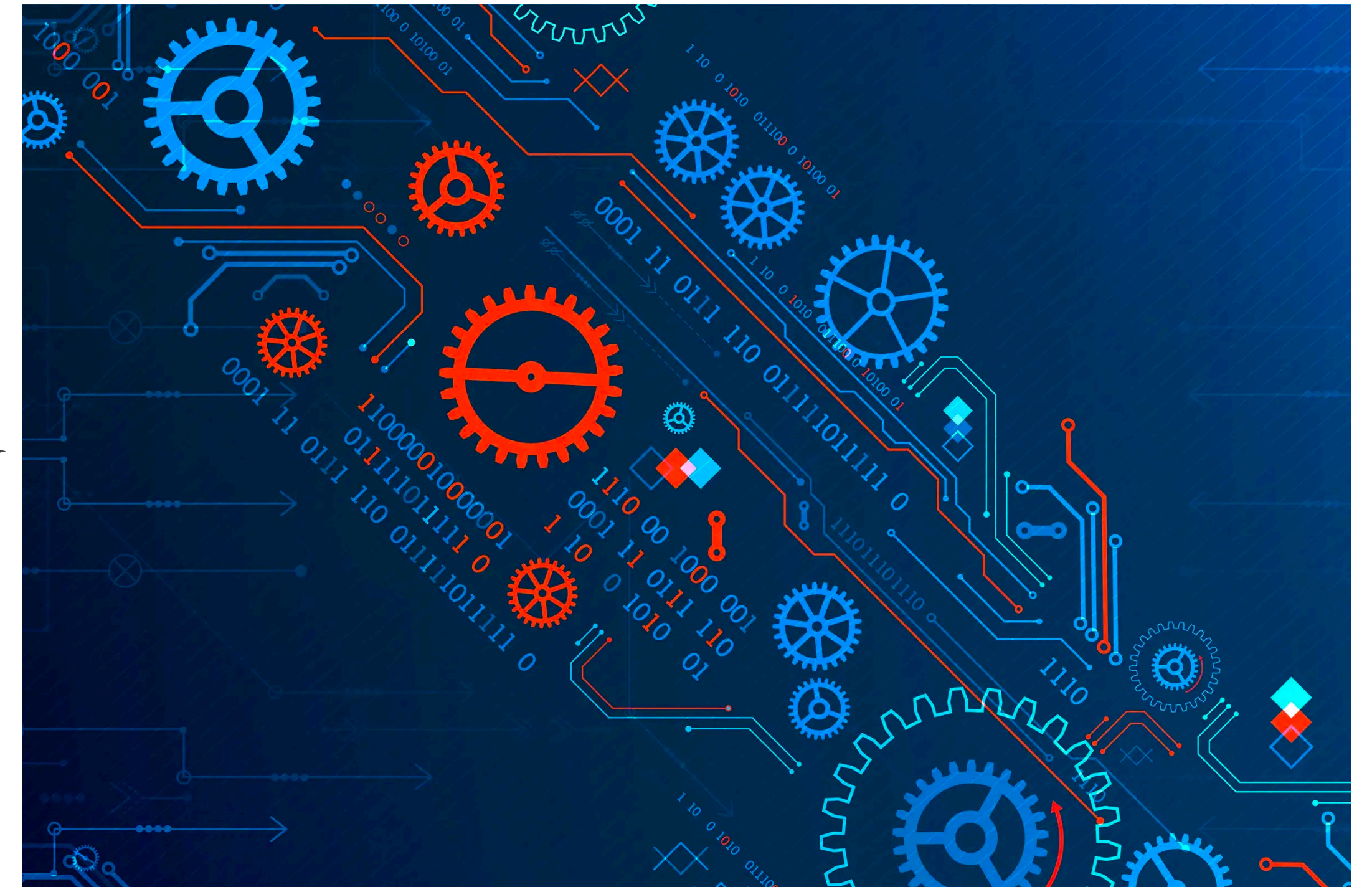
[Archive](#)

Applications

- User programs that we write/download
 - e.g., Slack, Chrome, games, vim, etc.

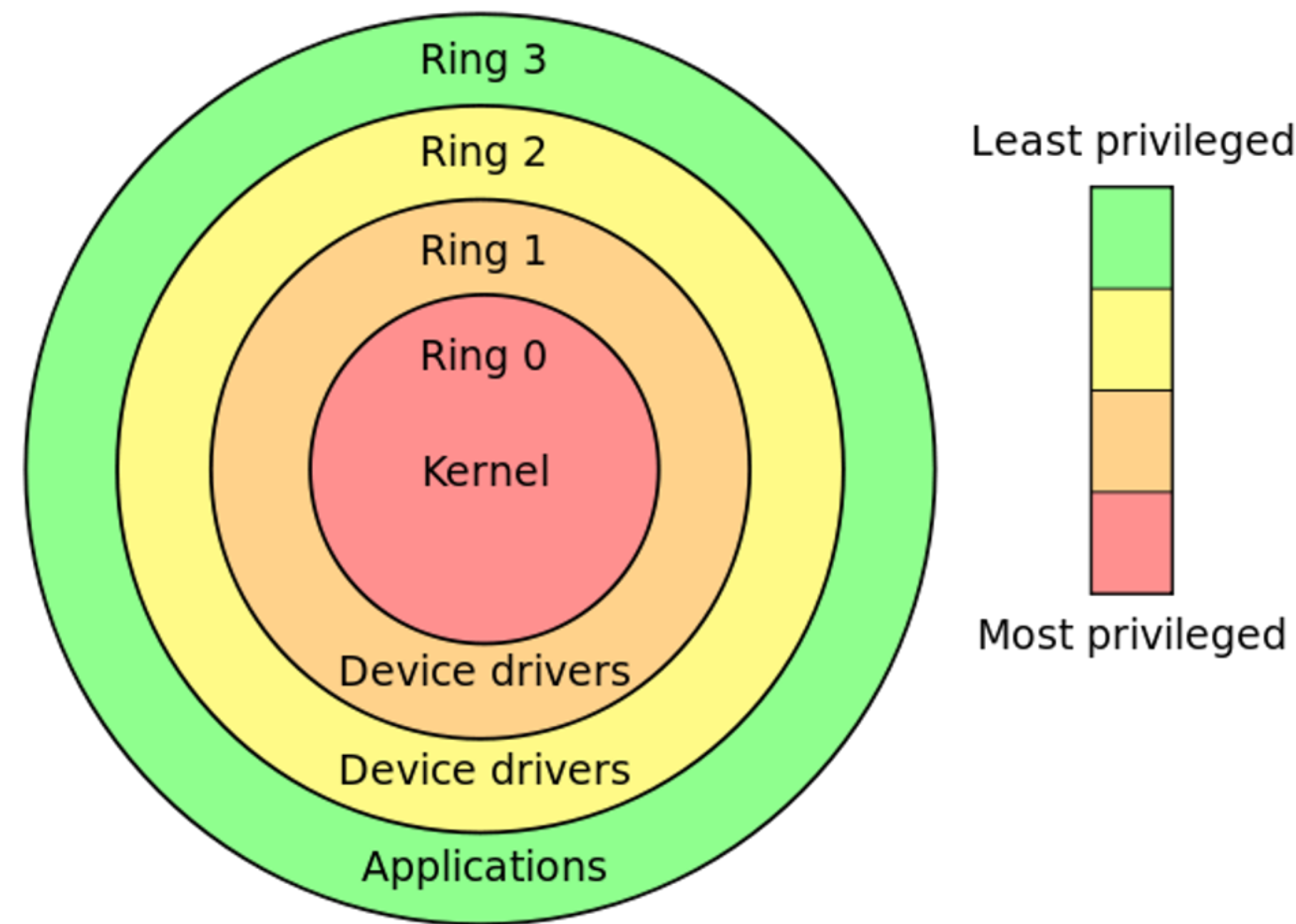


Systems Software



Privilege Rings

- CPUs provide multiple privilege levels, separating privileged OS/kernel execution from unprivileged applications.



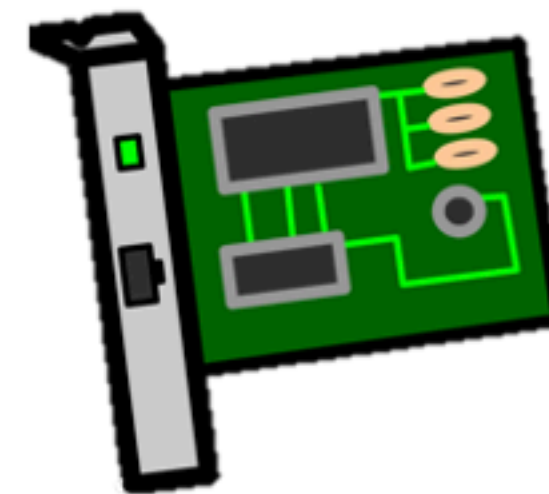
Privilege rings for x86

Hardware-Software Stack



How It Works in Practice?

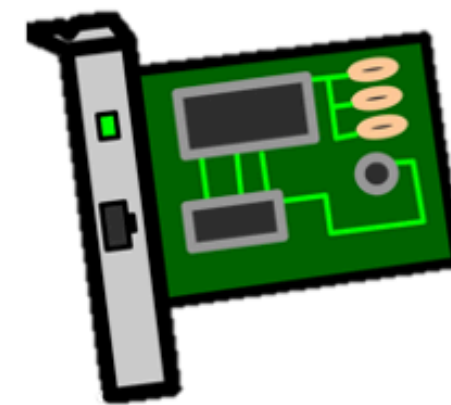
KERNEL [Ring 0]



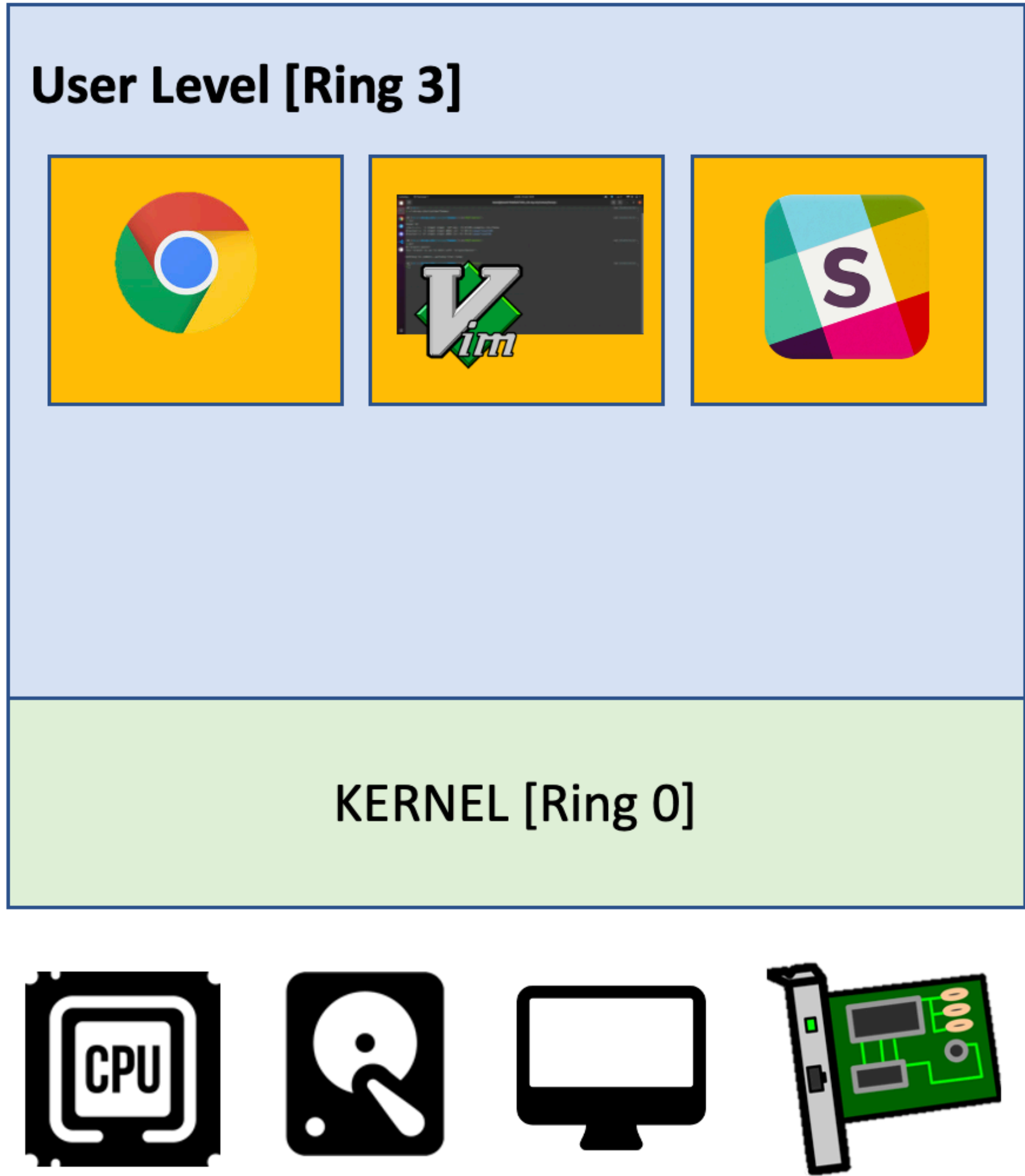
How It Works in Practice?

User Level [Ring 3]

KERNEL [Ring 0]



How It Works in Practice?



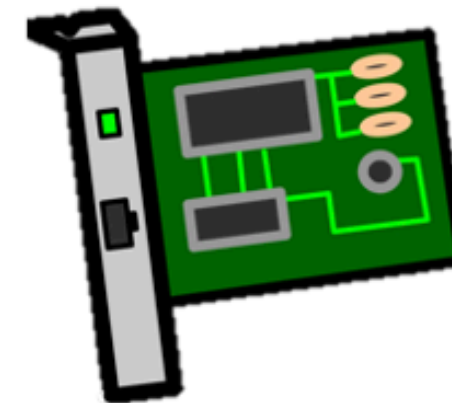
How It Works in Practice?

User Level [Ring 3]

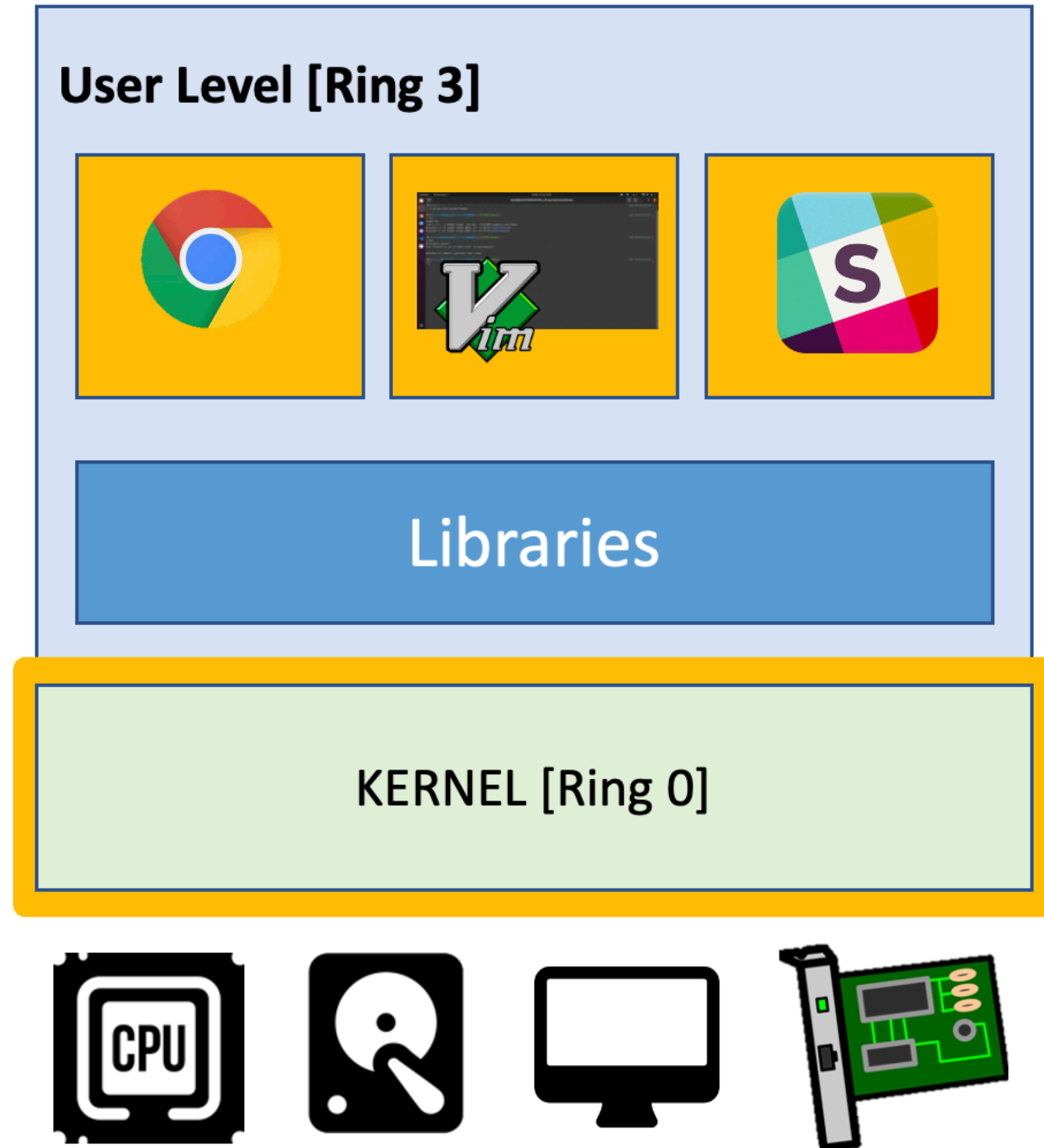


Libraries

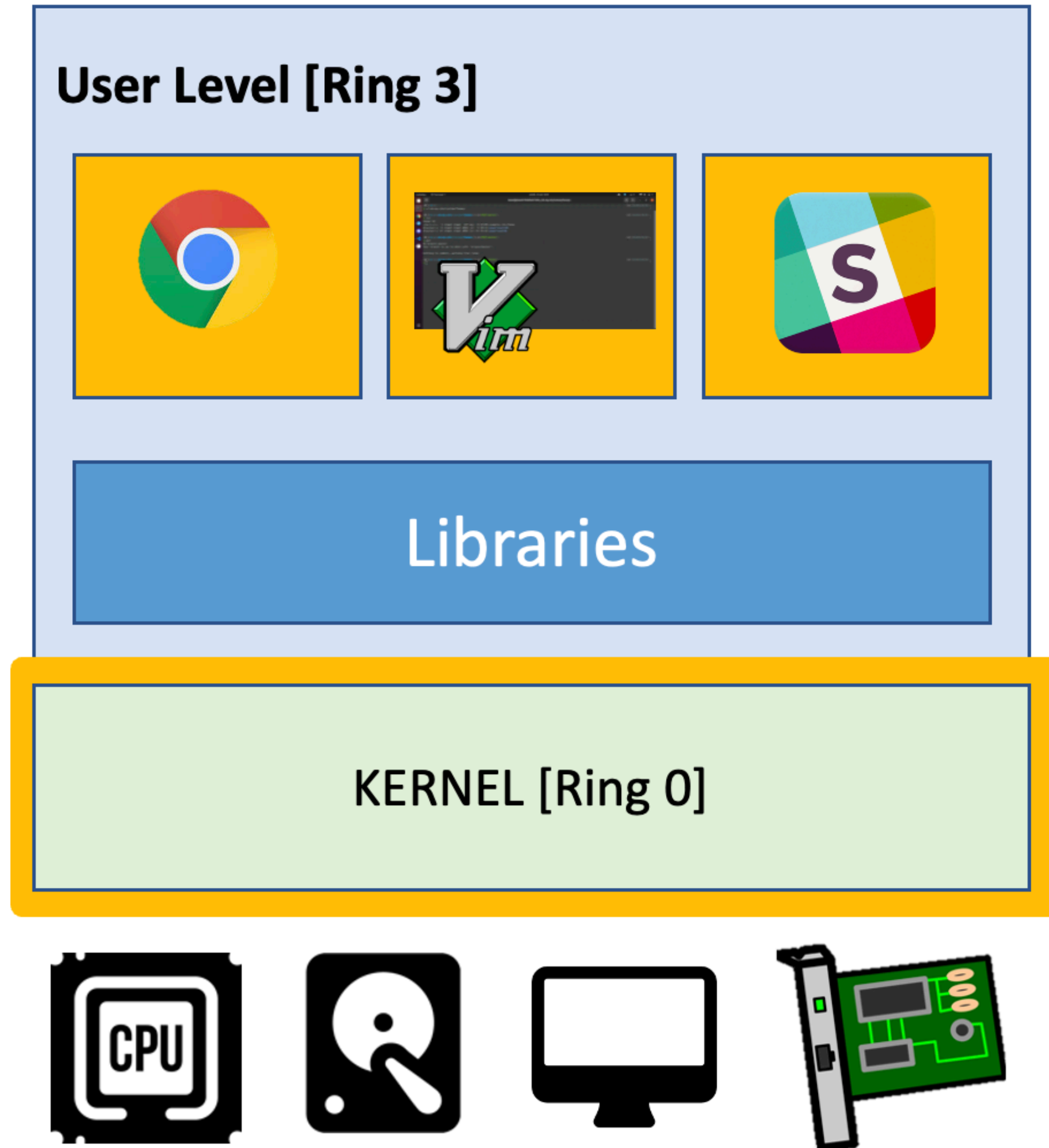
KERNEL [Ring 0]



Kernel (ring 0)

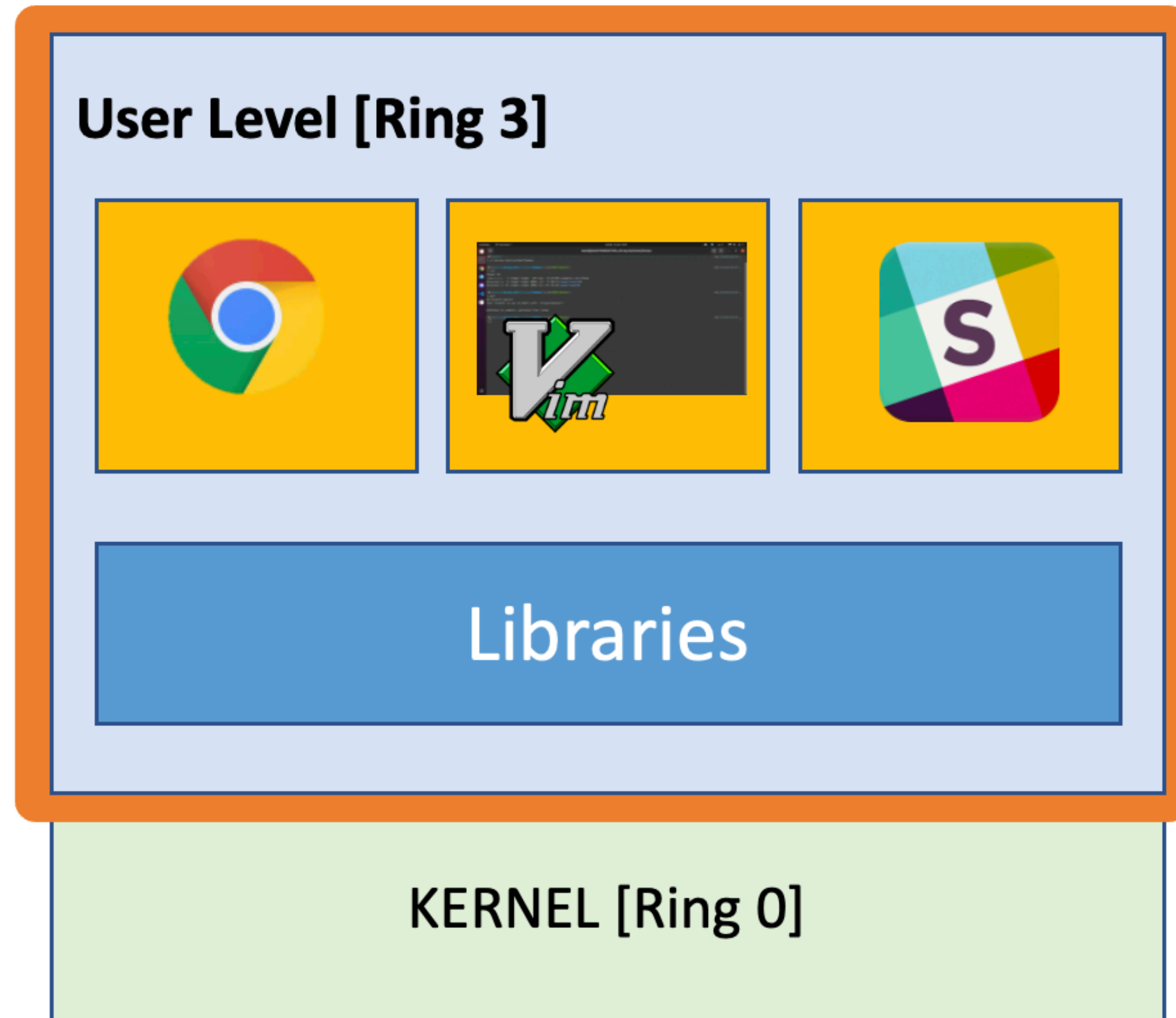


Kernel (ring 0)



- Run at the highest privilege level
- Manage hardware resources
- Manage processes/apps
- Enforce protection and isolation
- Trusted base

Kernel (ring 0)

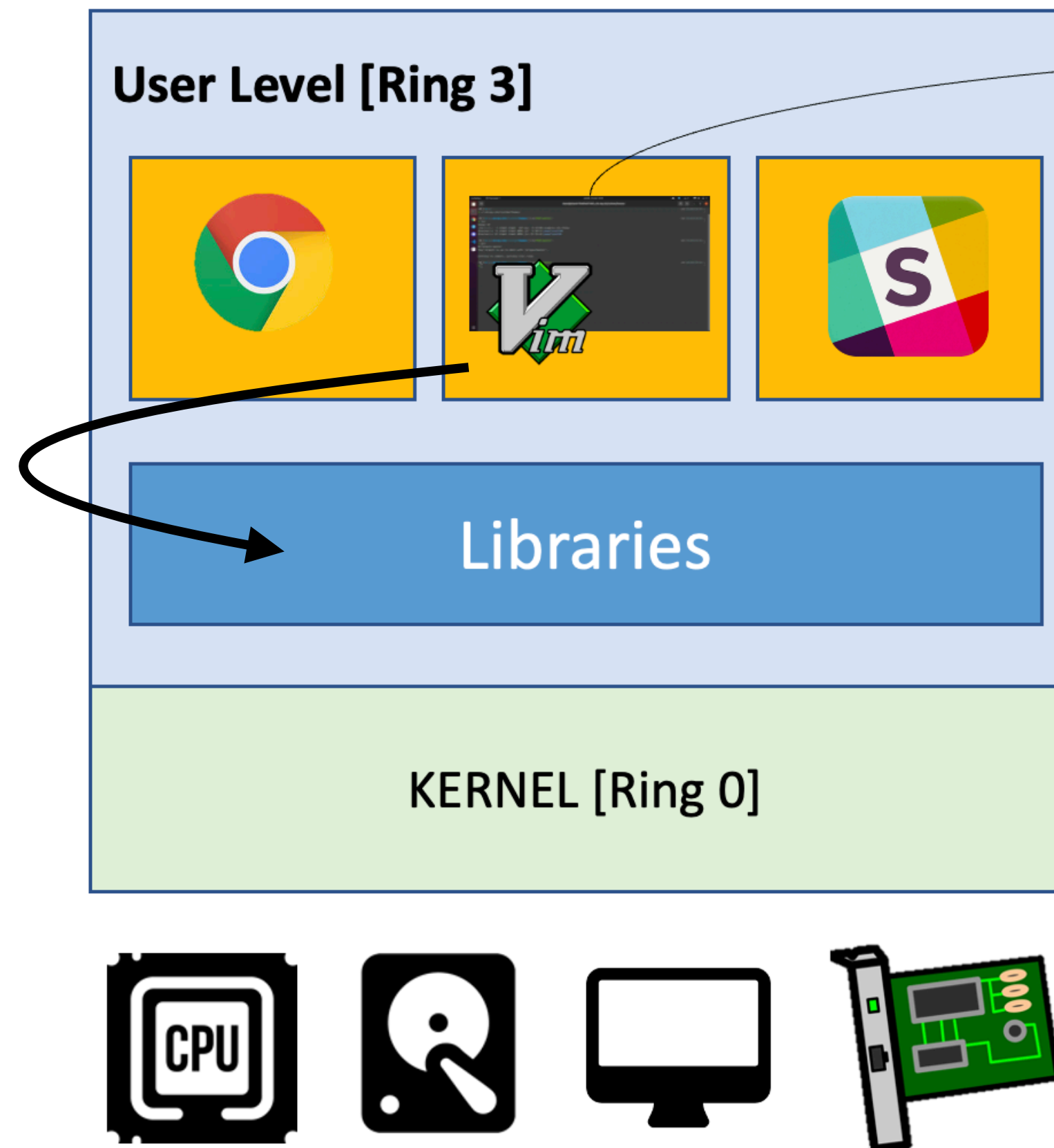


- Restricted privileges
- Applications run at ring 3
- Cannot access kernel memory
- Cannot directly access hardware
- Kernel mediates accesses.



Example: printf()

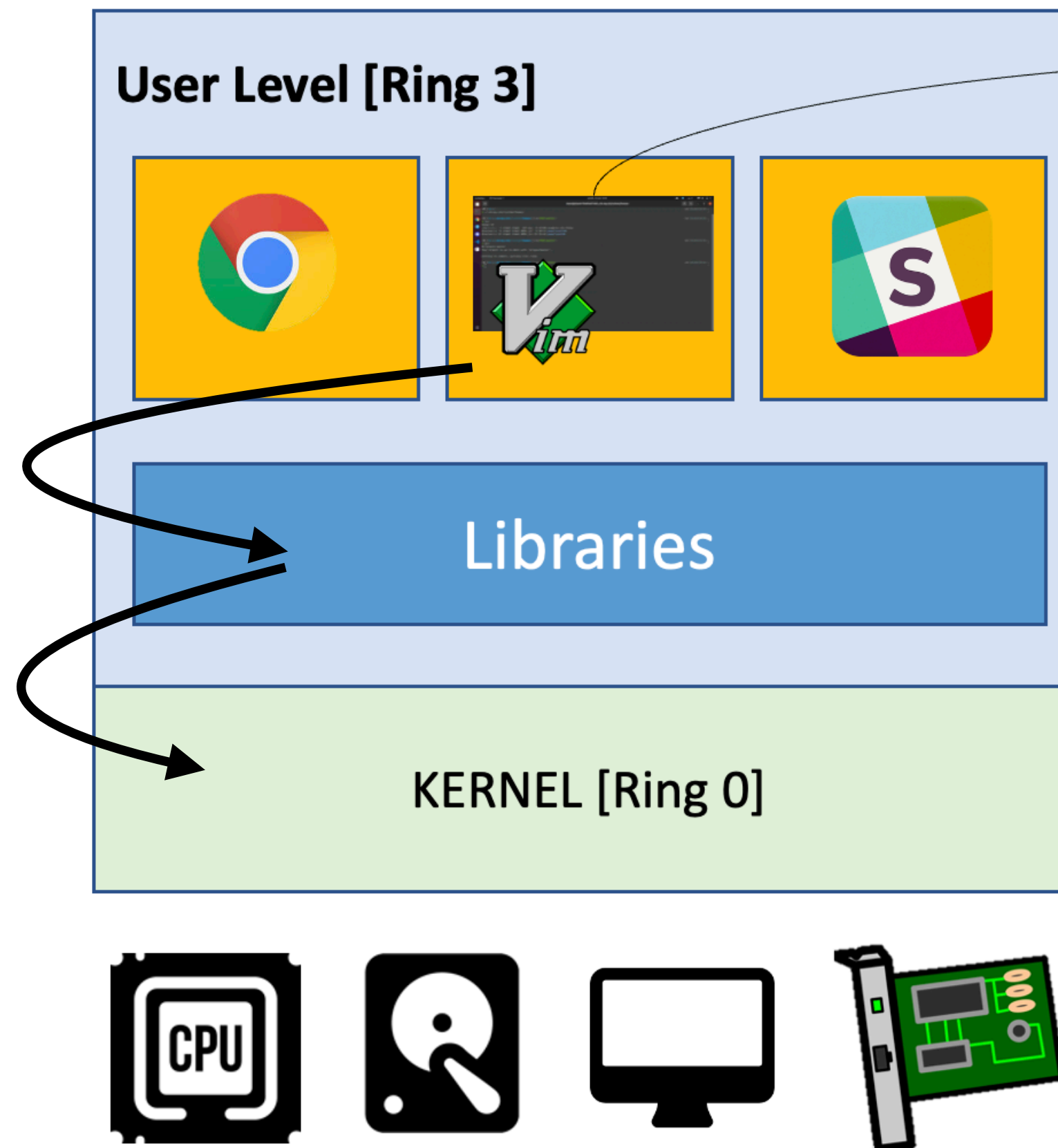
- What happens when you call printf()?
- printf() is a standard C library API.



```
int main() {  
    printf( "gwu" ) ;  
}
```


Example: printf()

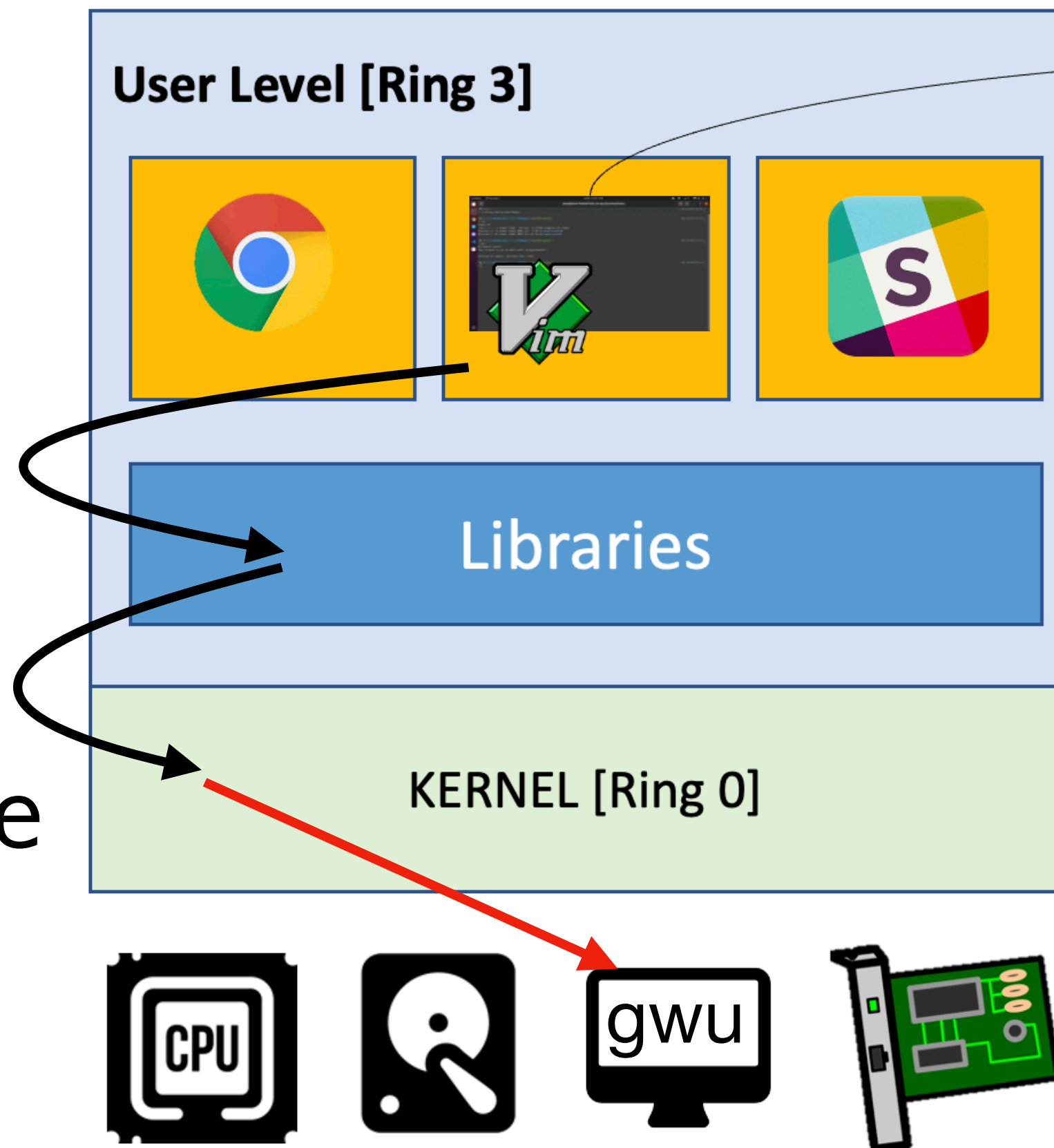
- What happens when you call printf()?
- printf() is a standard C library API.
- write(1, "gwu", 3) system call
- Switch from ring to ring 0



```
int main() {  
    printf( "gwu" ) ;  
}
```

Example: printf()

- What happens when you call printf()?
- printf() is a standard C library API.
- write(1, "gwu", 3) system call
- Switch from ring to ring 0
- Kernel invokes writing procedures, e.g. ksys_write
- Terminal receives and shows.

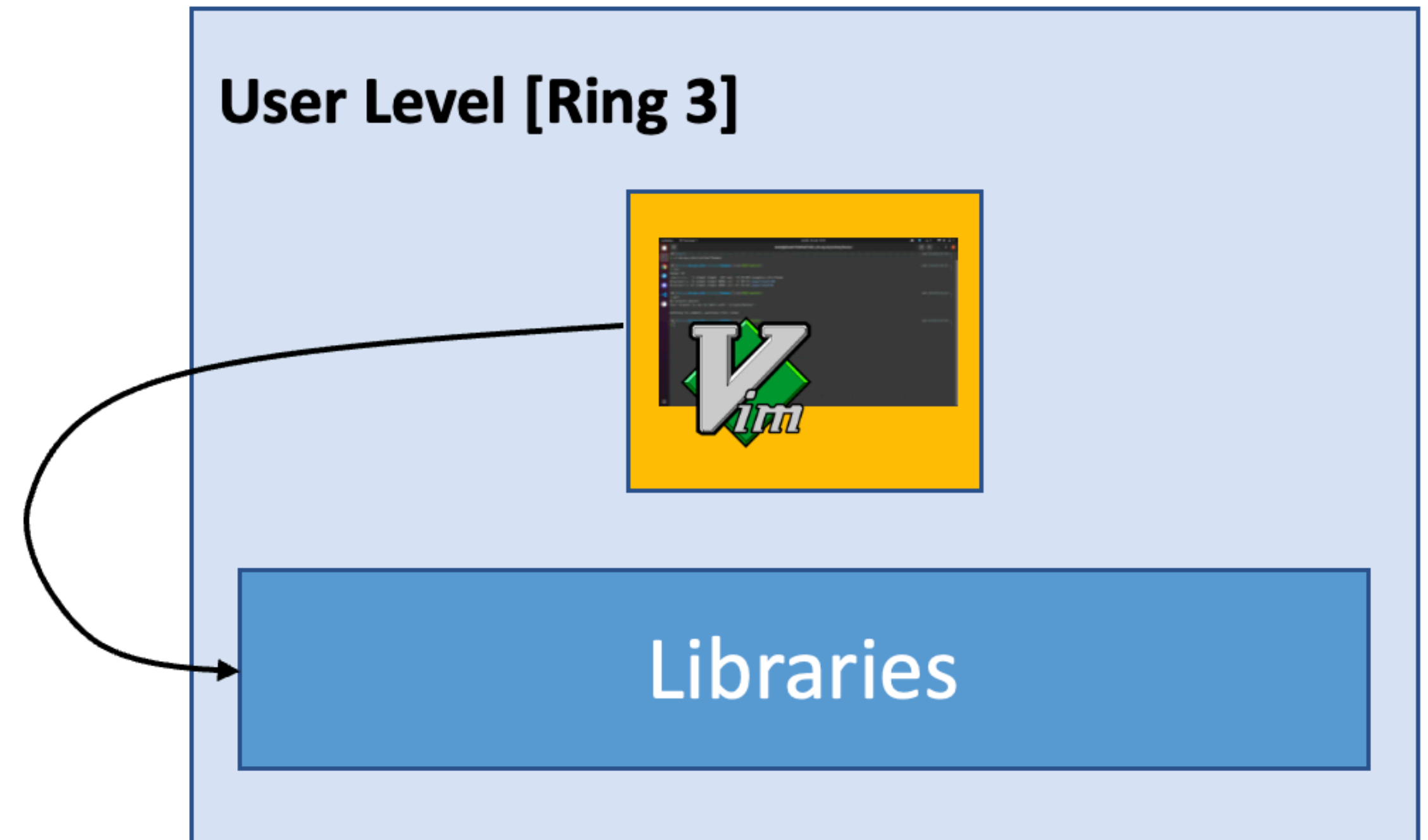


```
int main() {  
    printf( "gwu" ) ;  
}
```

Library Calls

- App function call
- Uses app memory space
- e.g., `printf()`, `strcpy()`
- ring 3 -> ring 3

`printf("gwu")`
library call in ring 3



Library Code Can Dominate Execution

```
#include<stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");

    return 0;
}
```



*How many machine instructions
run for program?*

```
jie@fedora: /home/jie/courses/csci2410
[$ gcc -o3 hello.c -o hello
jie@fedora: /home/jie/courses/csci2410
[$ ./hello
Hello, world!
```

```
jie@fedora: /home/jie/courses/csci2410
[$ perf stat -e instructions ./hello
Hello, world!
```

Performance counter stats for './hello':

```
126,812      instructions:u

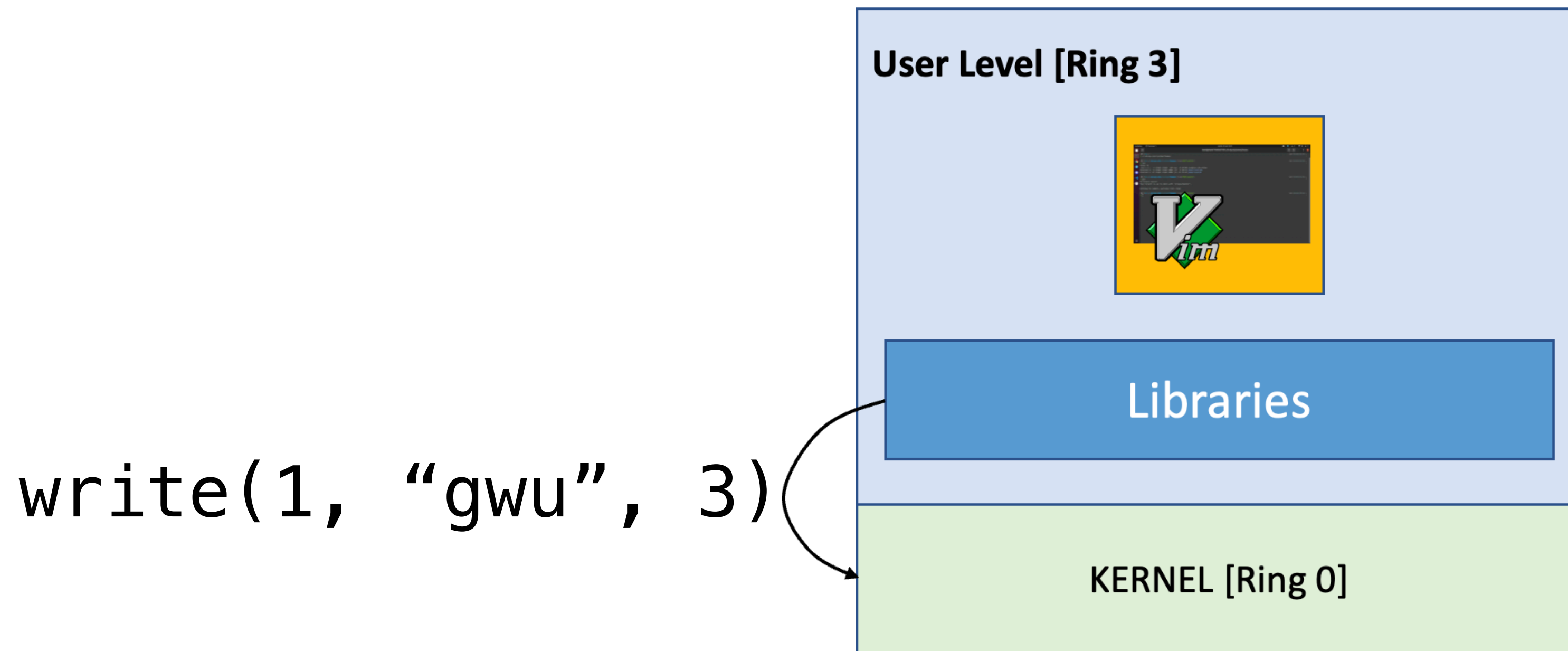
0.002079883 seconds time elapsed

0.000000000 seconds user
0.002408000 seconds sys
```


System Calls



A system call (syscall) is the programmatic way in which a computer program requests a service from the operating system on which it is executed.

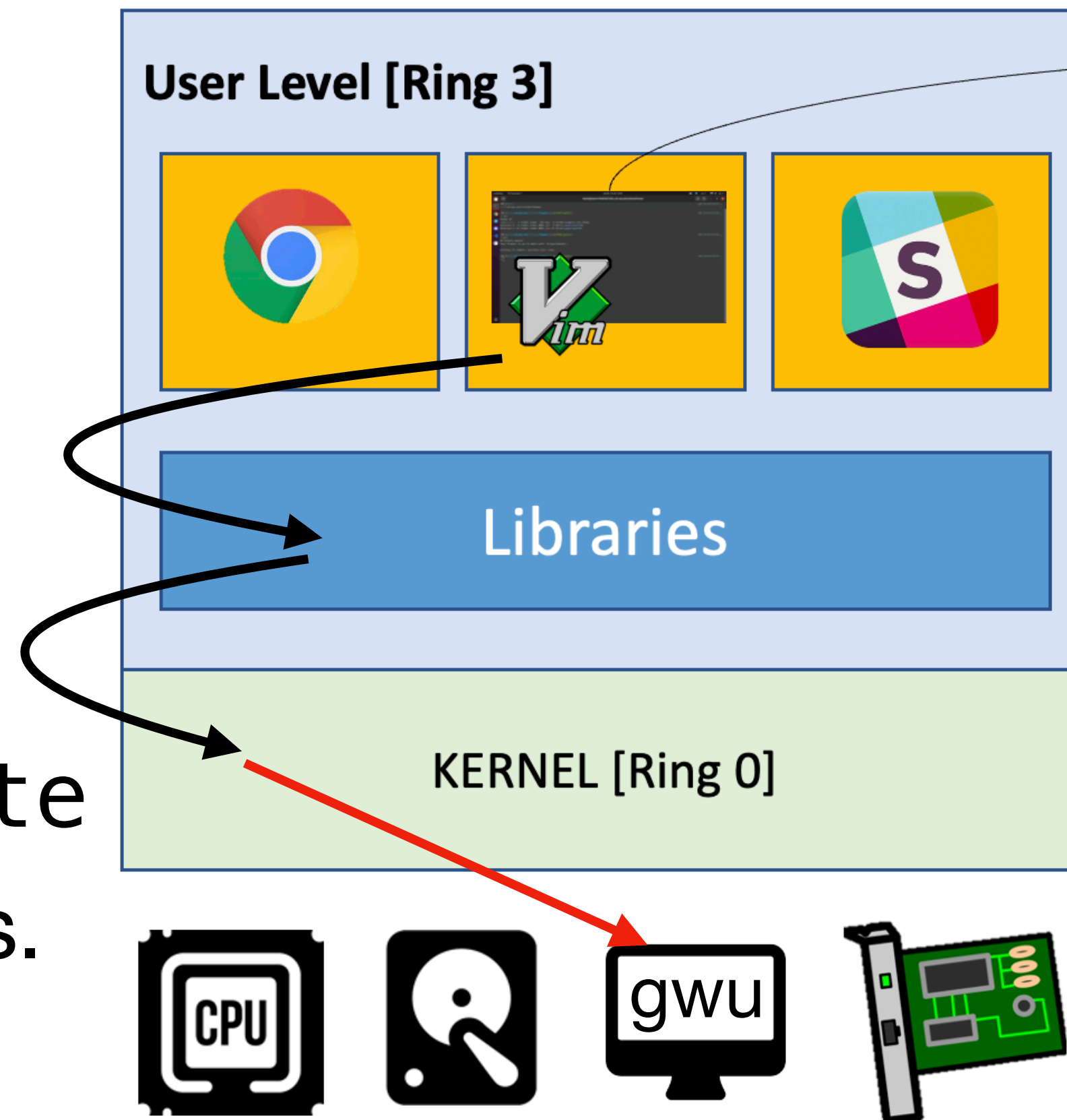


- Function call from app
- Request OS services
- System API
- E.g., I/O, networking
- ring 3 -> ring 0

How a user program asks the kernel to do something privileged on its behalf.

Example: printf()

- What happens when you call printf()?
- printf() is a standard C library API.
- write(1, "gwu", 3) system call
- Switch from ring to ring 0
- Kernel invokes writing procedures, e.g. ksys_write
- Terminal receives and shows.

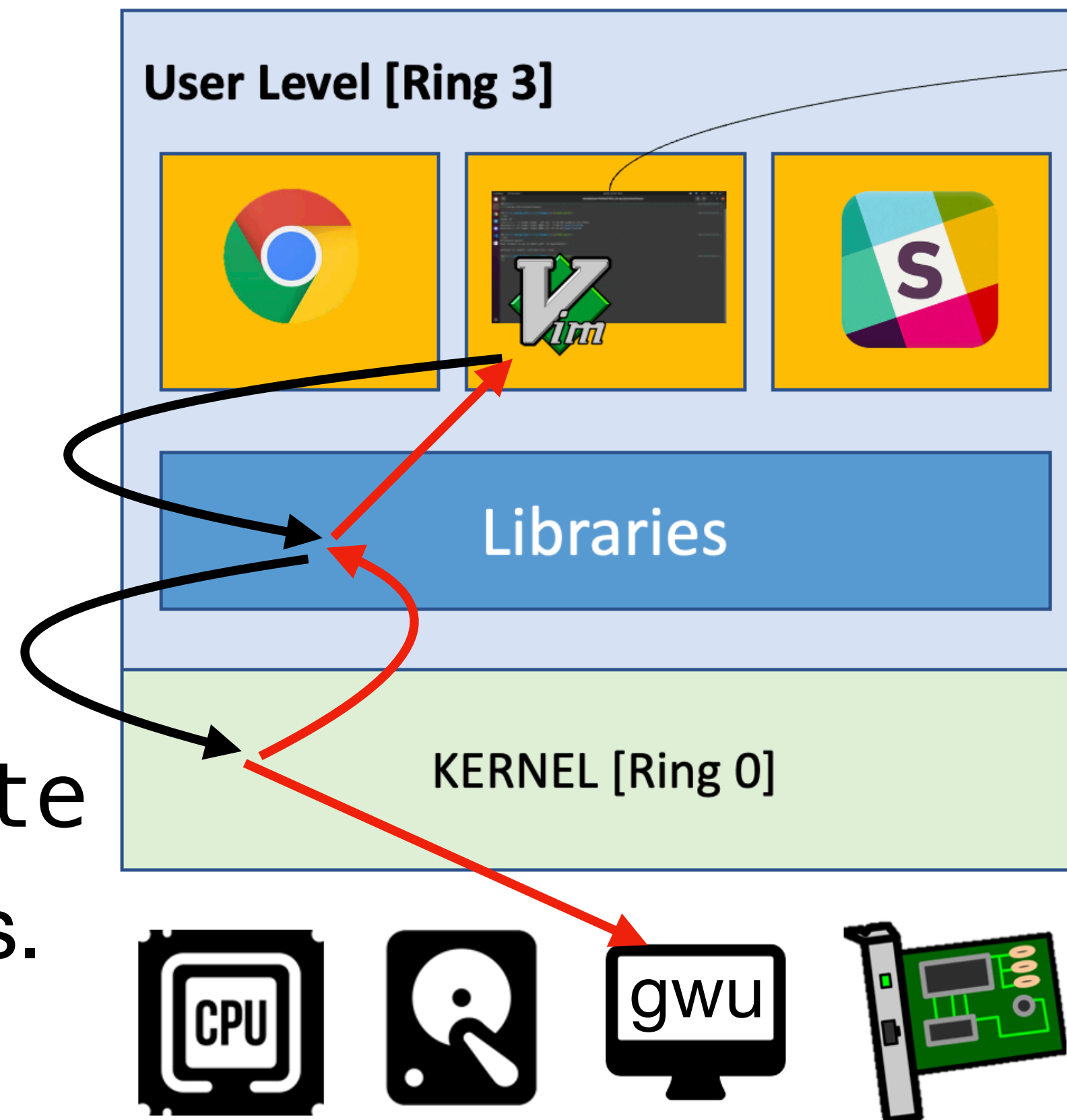


```
int main() {  
    printf( "gwu" ) ;  
}
```

We are still in the kernel!

Example: printf()

- What happens when you call printf()?
- printf() is a standard C library API.
- write(1, "gwu", 3) system call
- Switch from ring to ring 0
- Kernel invokes writing procedures, e.g. ksys_write
- Terminal receives and shows.
- Return from kernel to user;
Ring 3 -> ring 0
- Return from user lib to user app



```
int main() {  
    printf( "gwu" ) ;  
}
```




ELF: Executable and Linkable Format

- Common standard file format for binary code
 - executables, object code, shared libraries, etc.
- Standard binary file format for UNIX-like (including Linux) systems on x86

