Fundamental concepts

Lecture 1

Module 1

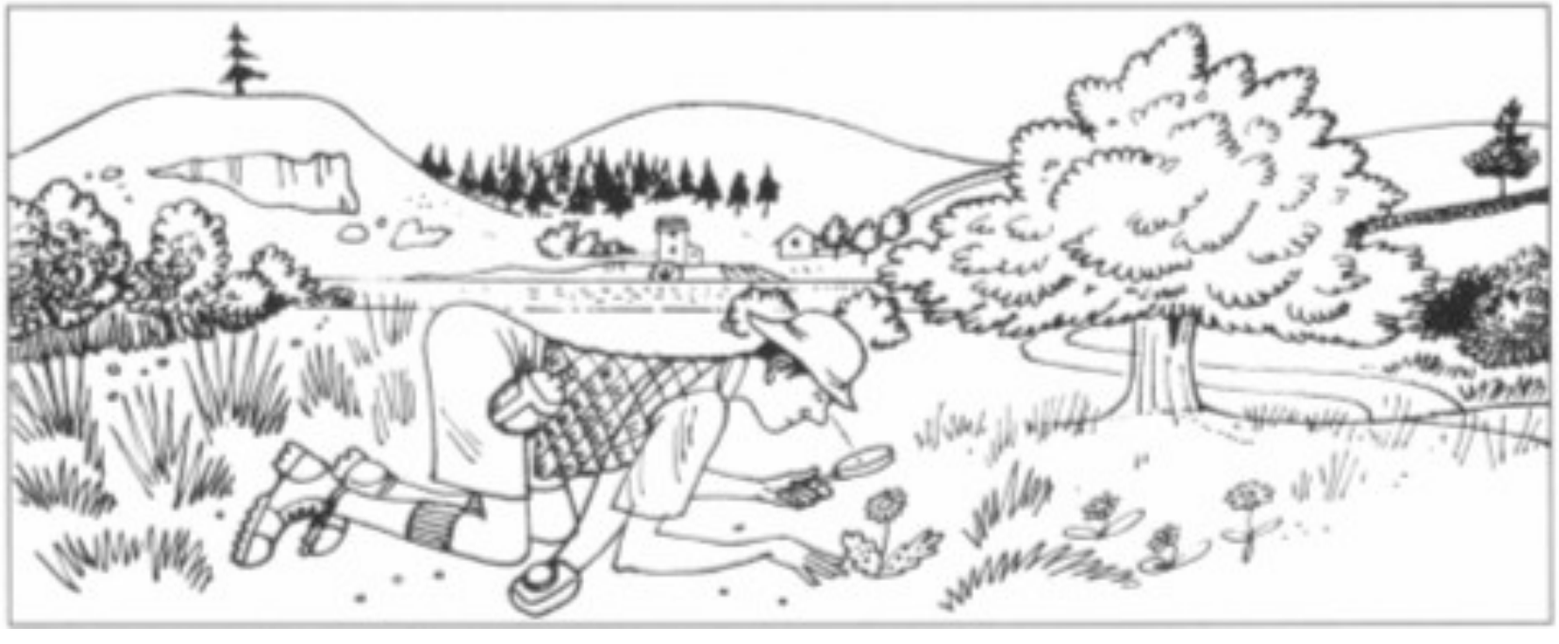
Operating systems 2018

1DT044 and 1DT096

2018-01-16 karl.marklund@it.uu.se Uppsala University

Learning and understanding

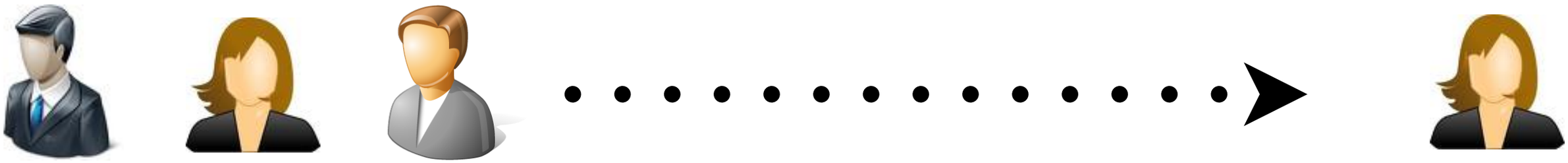
Top down



Bottom up

One top down view from lecture 0

User 1 User 2 User 3 User N



Compiler Spreadsheet Text Editor Pacman

System and Application Programs

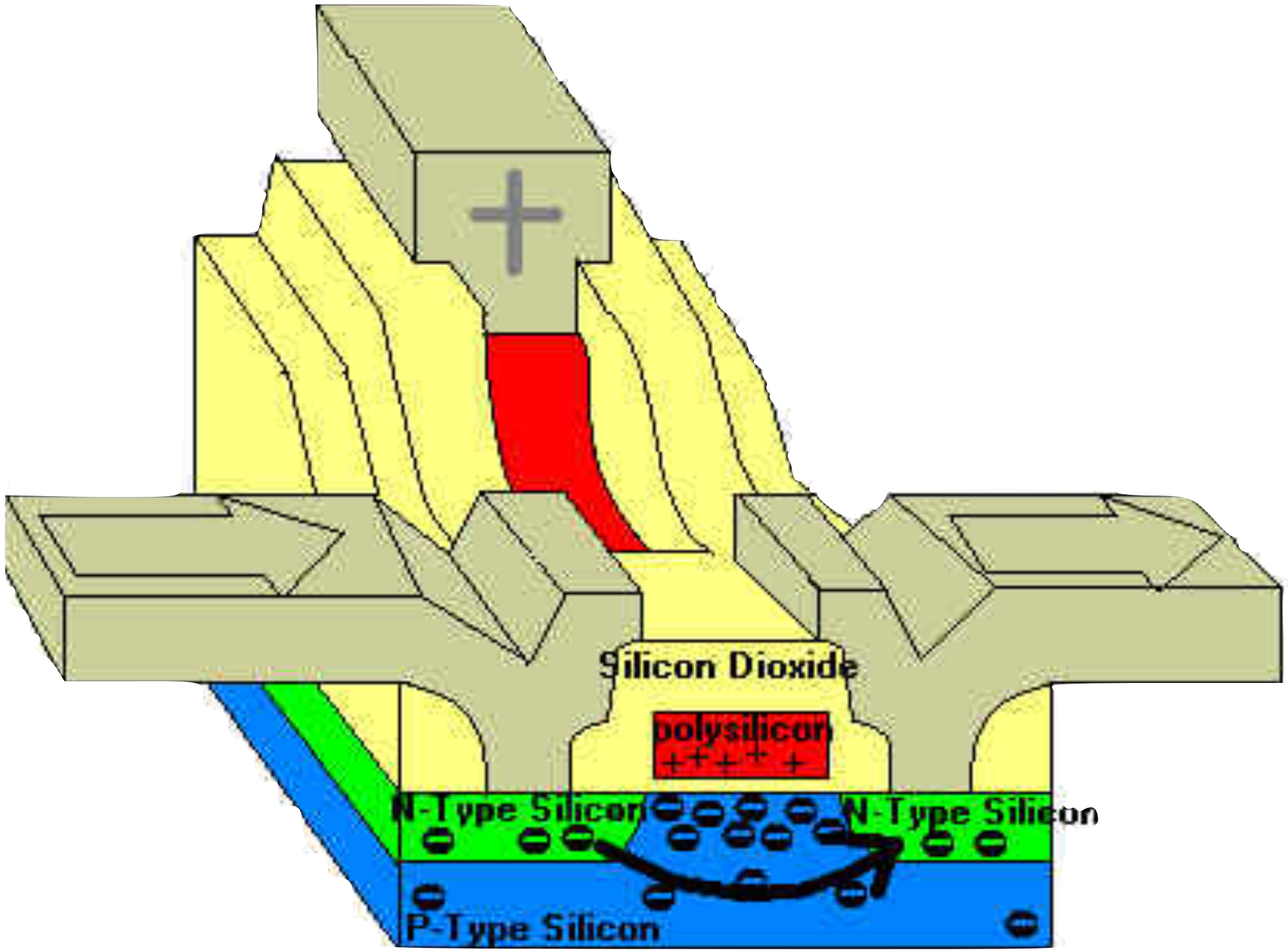
Operating System

Controls the hardware and coordinates its use among the various application programs for the various users.

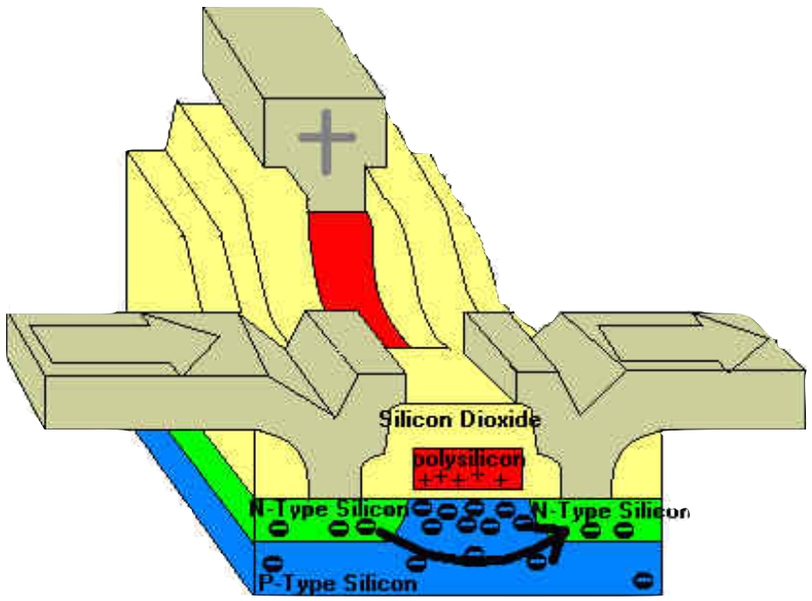
Computer Hardware

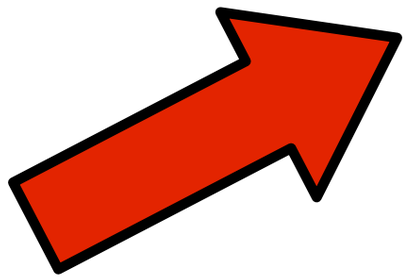
Now, let's start bottom up

The transistor

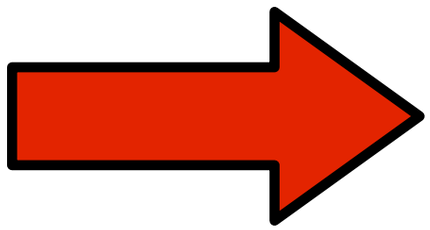


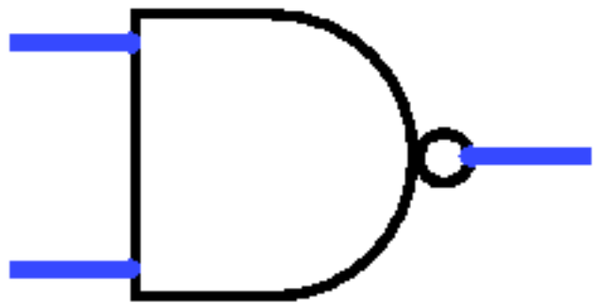
From transistors to logical gates



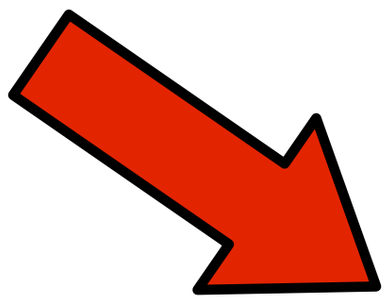


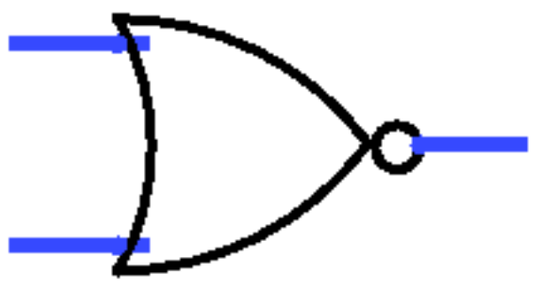
NOT





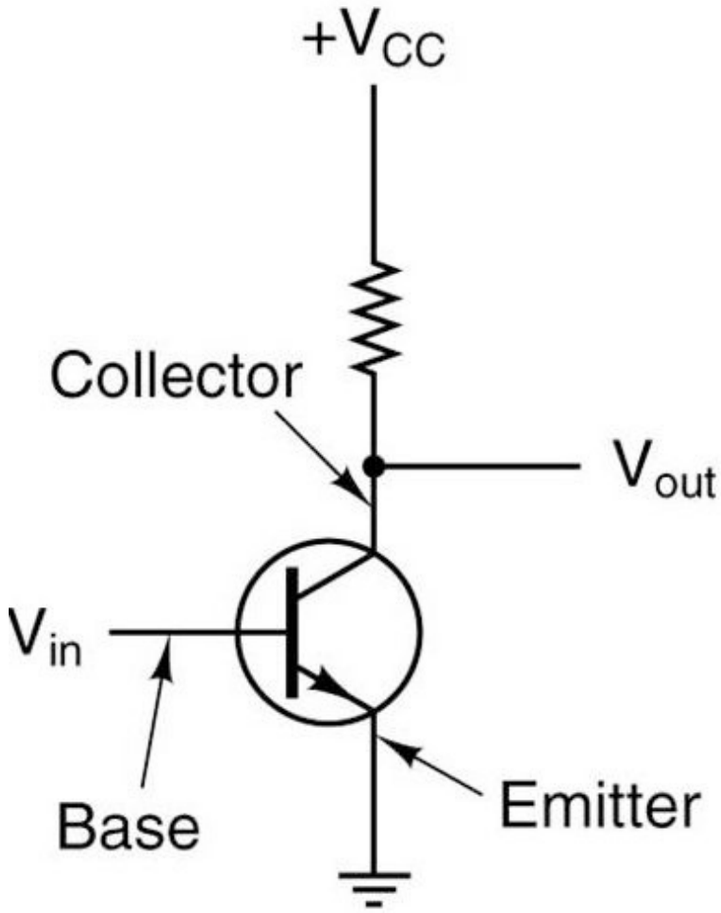
NAND

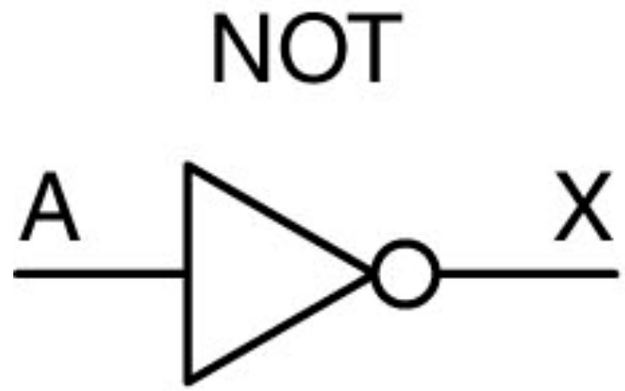




NOR

An inverter - logical NOT

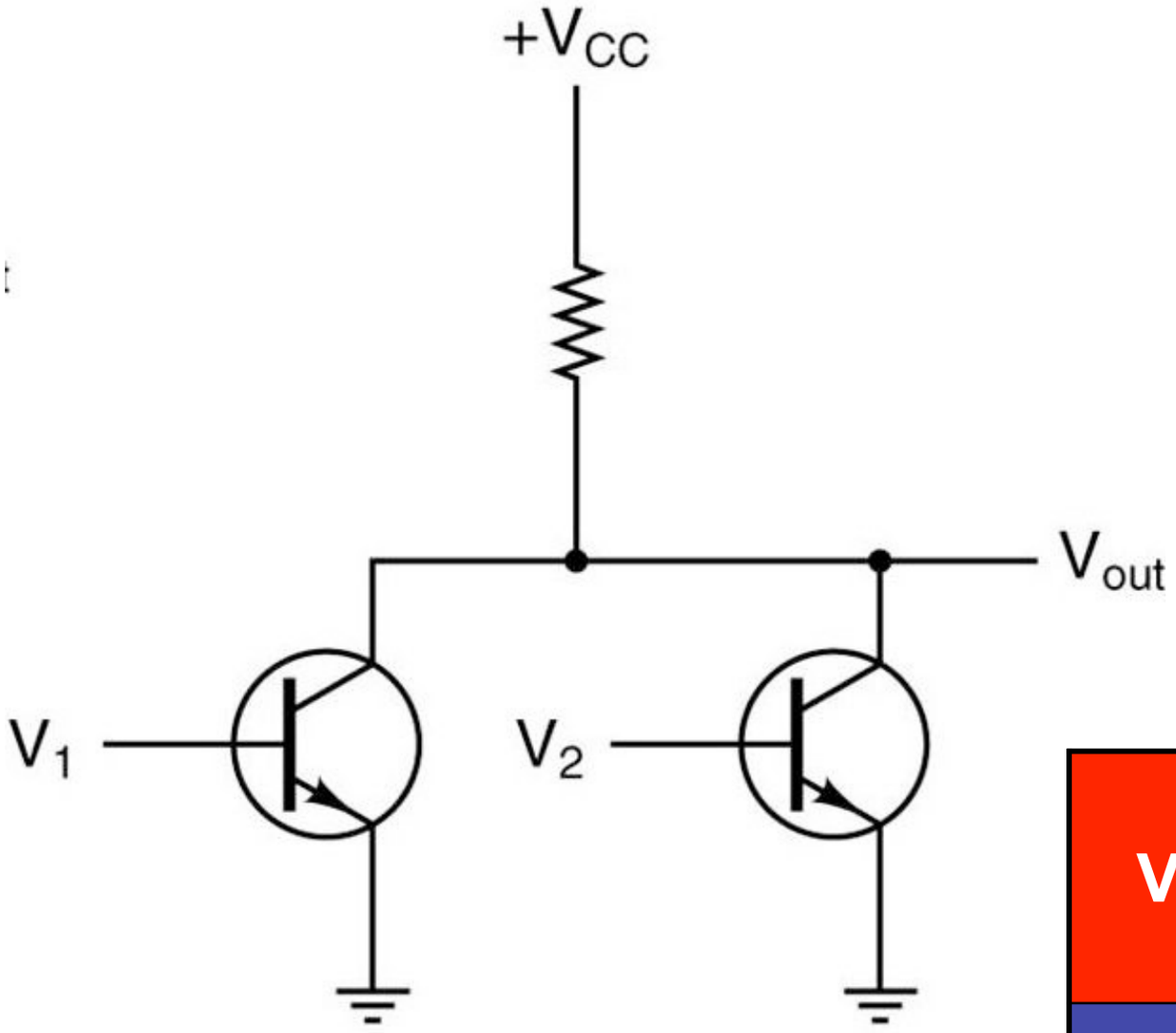


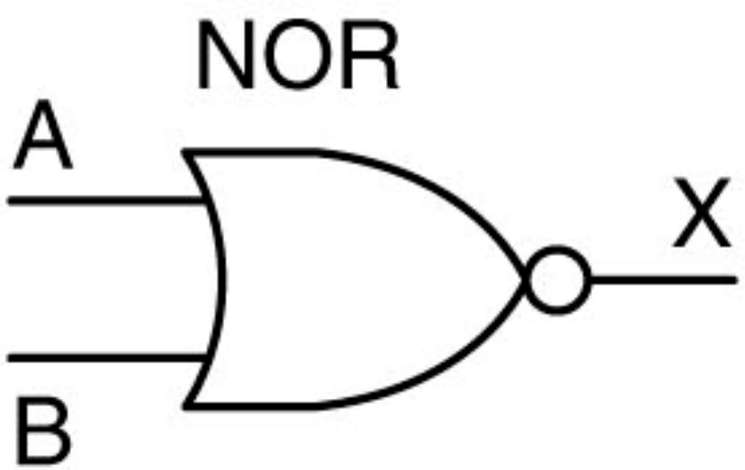


V_{in}	V_{out}
low	high
high	low

A	X
0	1
1	0

logical NOR

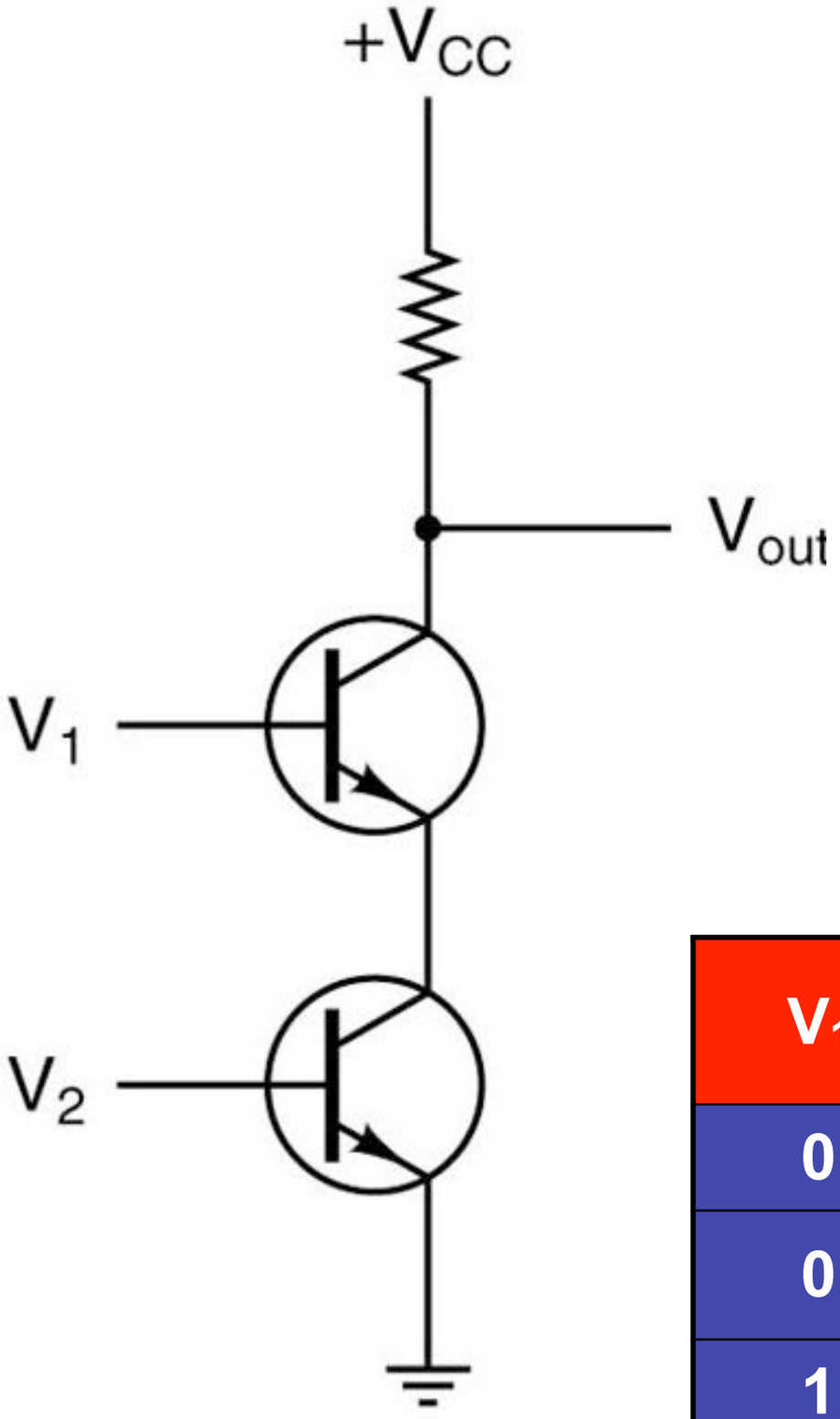


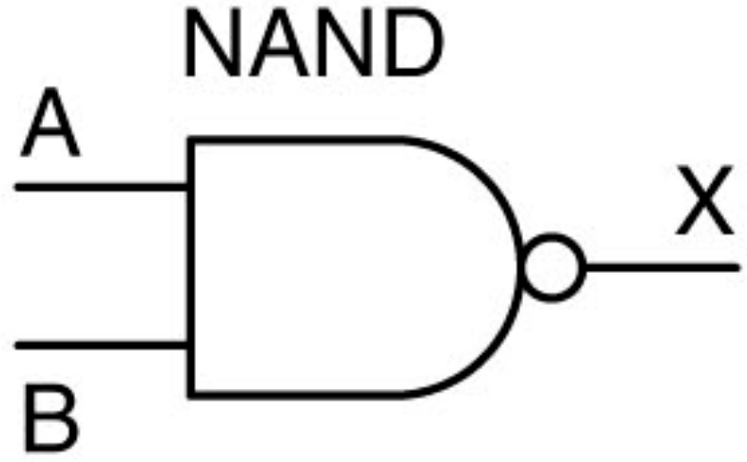


V_1	V_2	V_{out}	A or B	a nor b = not (A or B)
0	0	1	0	1
0	1	0	1	0
1	0	0	1	0
1	1	0	1	0

Low → 0 High → 1

logical NAND



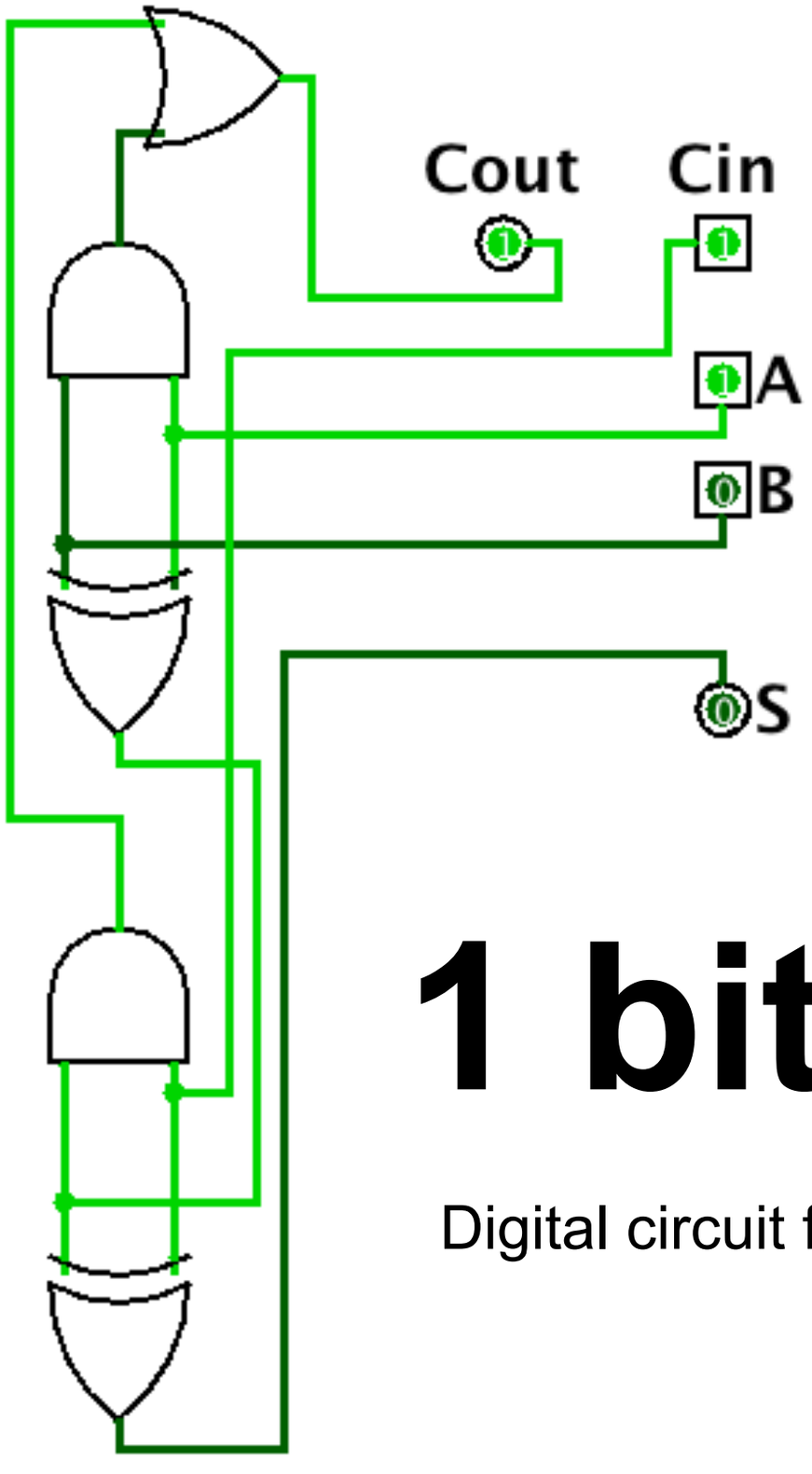


V_1	V_2	V_{out}	A and B	A nand B = not (A and B)
0	0	1	0	1
0	1	1	0	1
1	0	1	0	1
1	1	0	1	0

Low → 0 High → 1

One OR-gate to combine the two Halfadders

One Halfadder



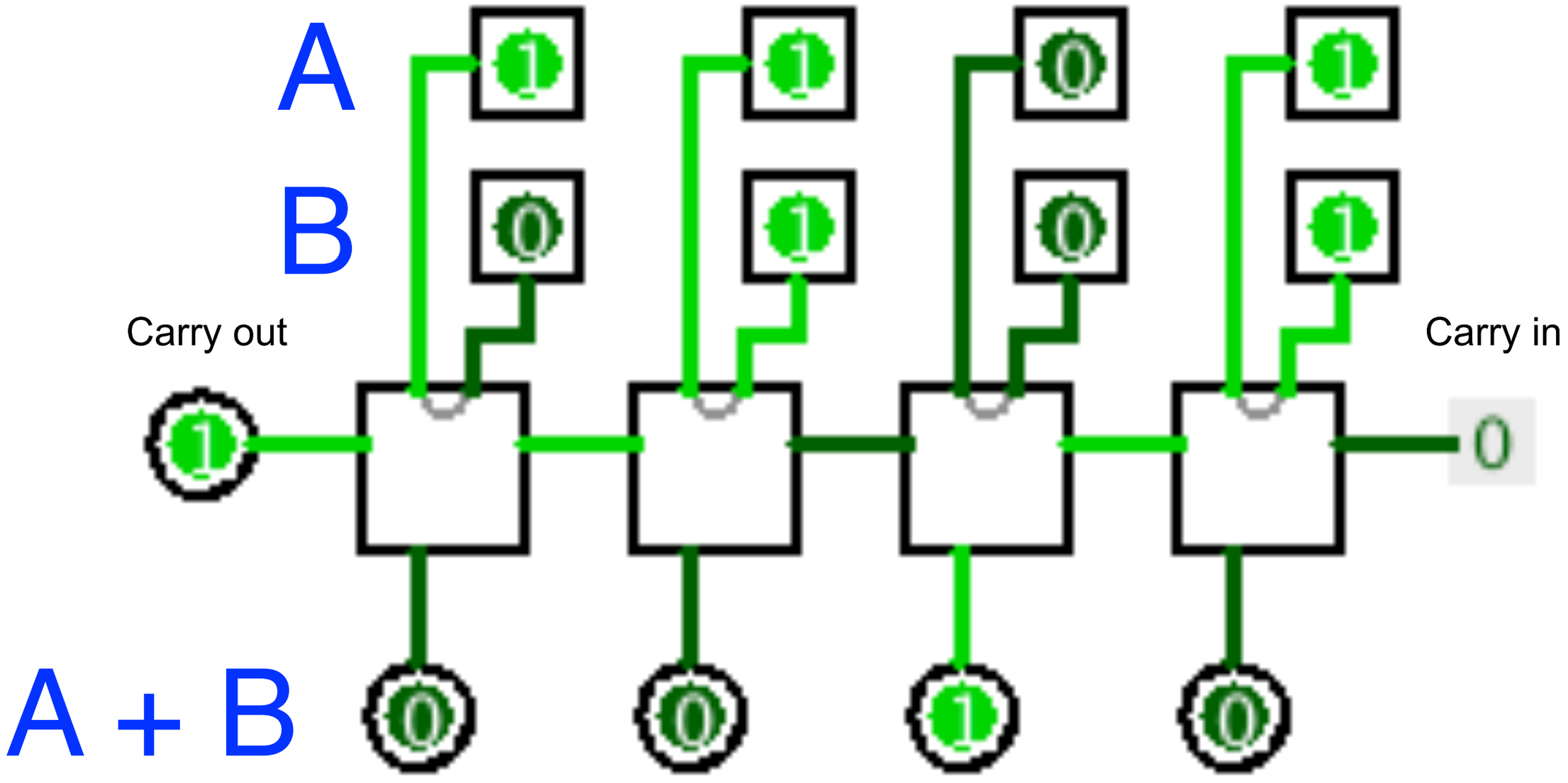
One Halfadder

1 bit full adder

Digital circuit for adding two 1 bit numbers A and B

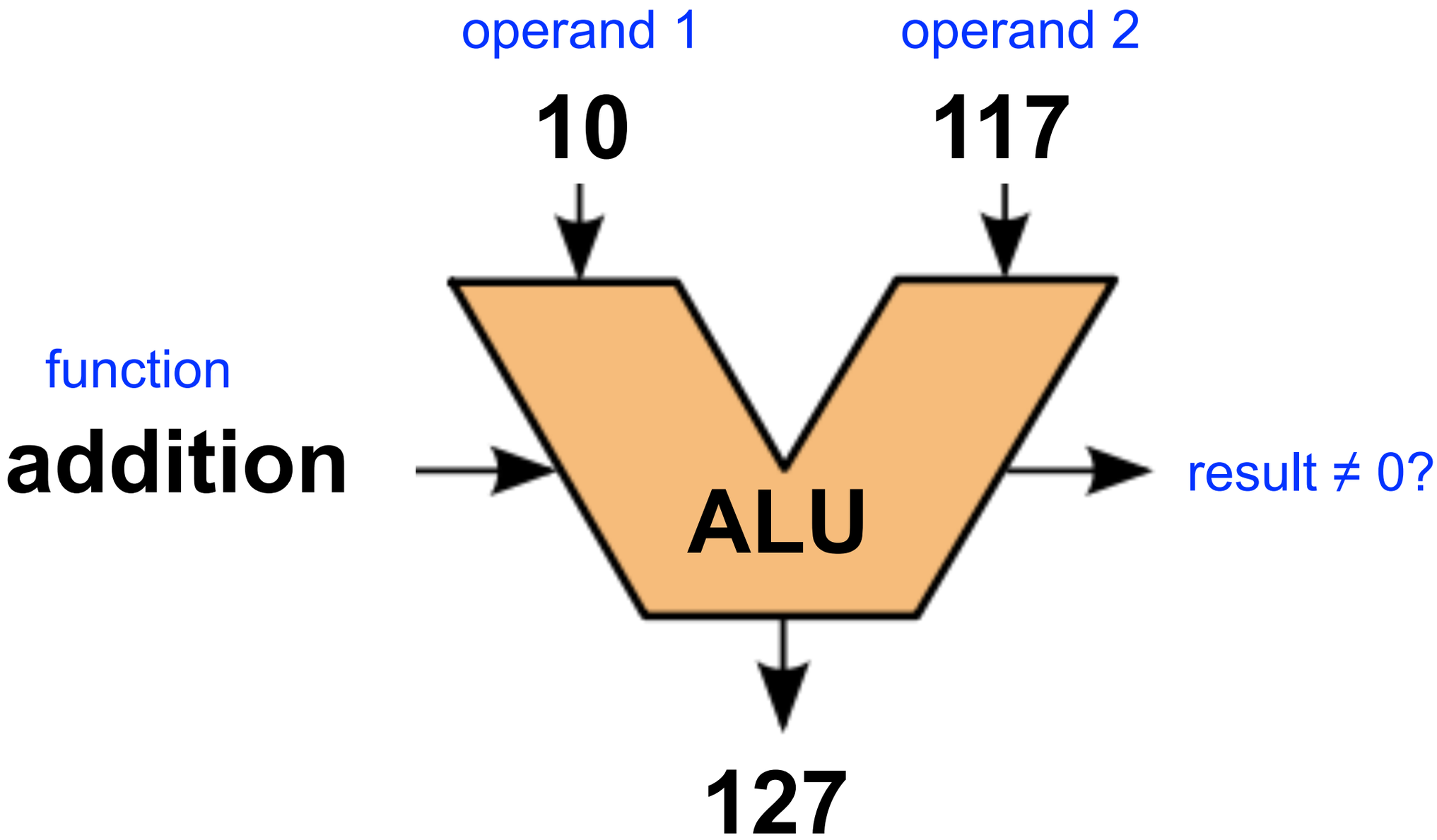
Ripple adder

Four 1 bit full adders used to add two four bit numbers



Arithmetic Logic Unit (ALU)

An adder is the first part in constructing a working ALU.



Arithmetic Logic Unit (ALU)

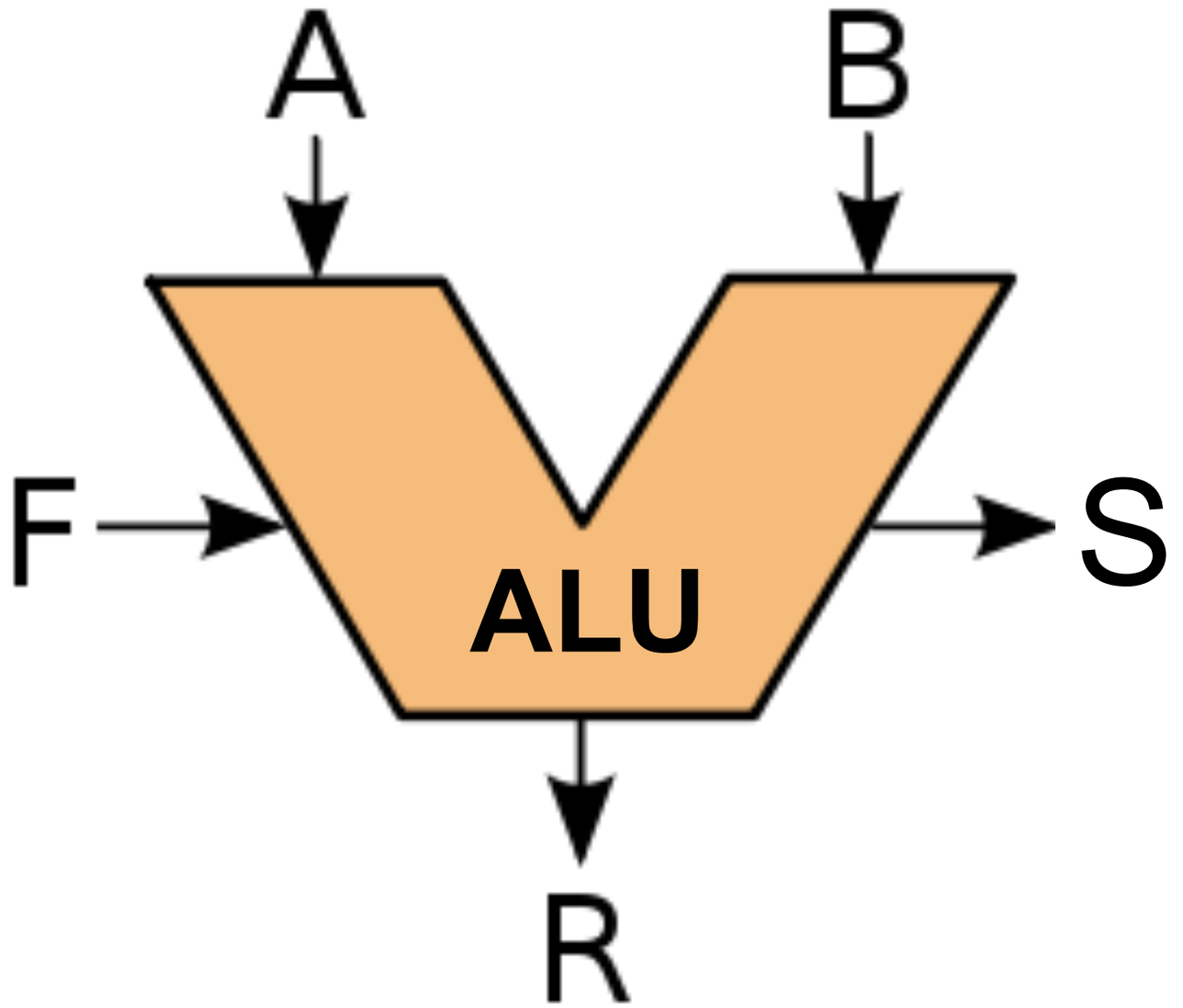
Similar to the addition circuit, circuits can be constructed for other ALU operations.

arithmetic

- A + B
- A − B
- A * B
- A / B

logic

- A < B
- A == B
- A AND B
- A OR B
- A XOR B



Input devices

CPU

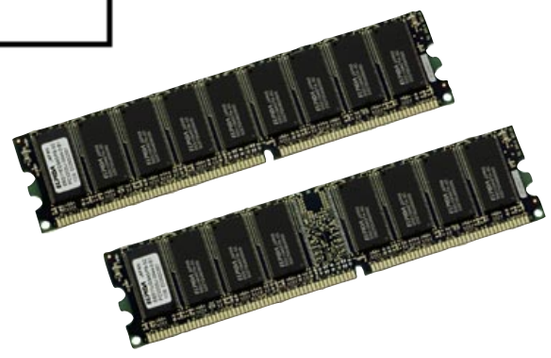
Control unit

Arithmetic logic unit

Output devices

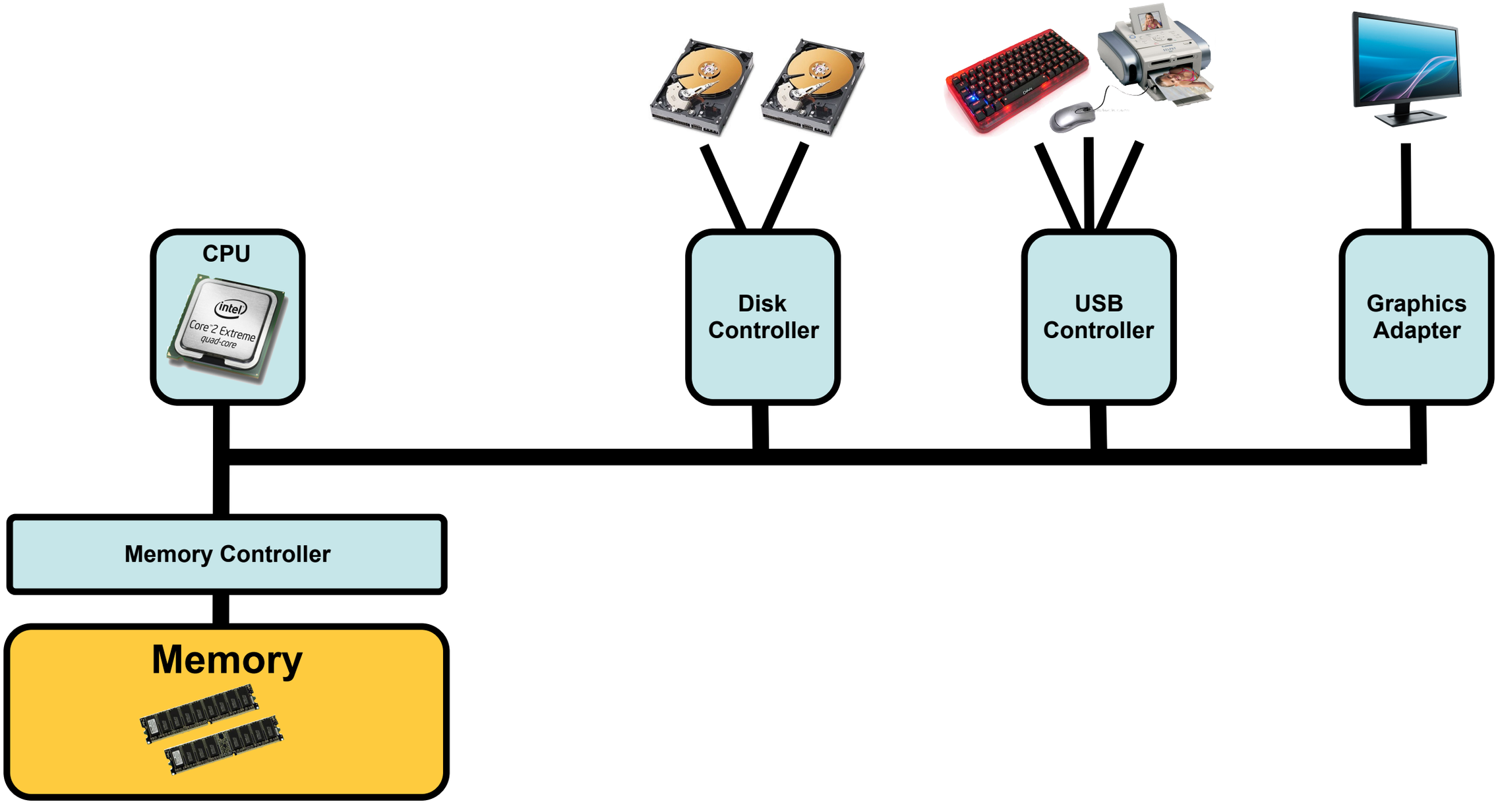
Memory

External storage



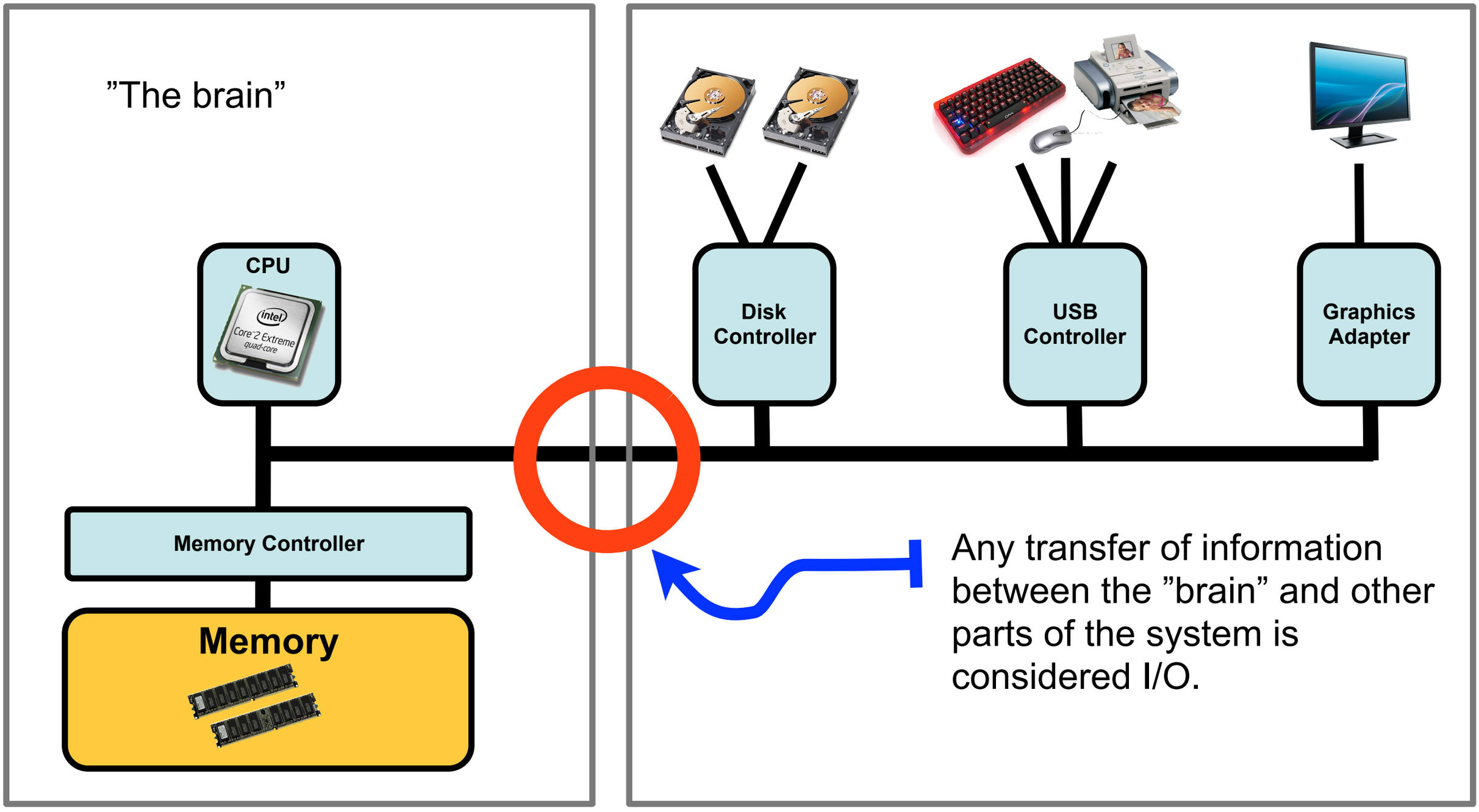
The von Neuman modell

A typical modern computer system

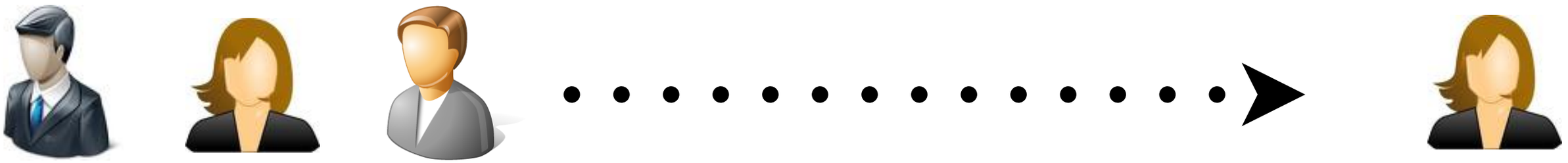


Input and Output (I/O)

In computer architecture, the combination of the CPU and main memory (i.e. memory that the CPU can read and write to directly, with individual instructions) is considered the ”brain” of a computer



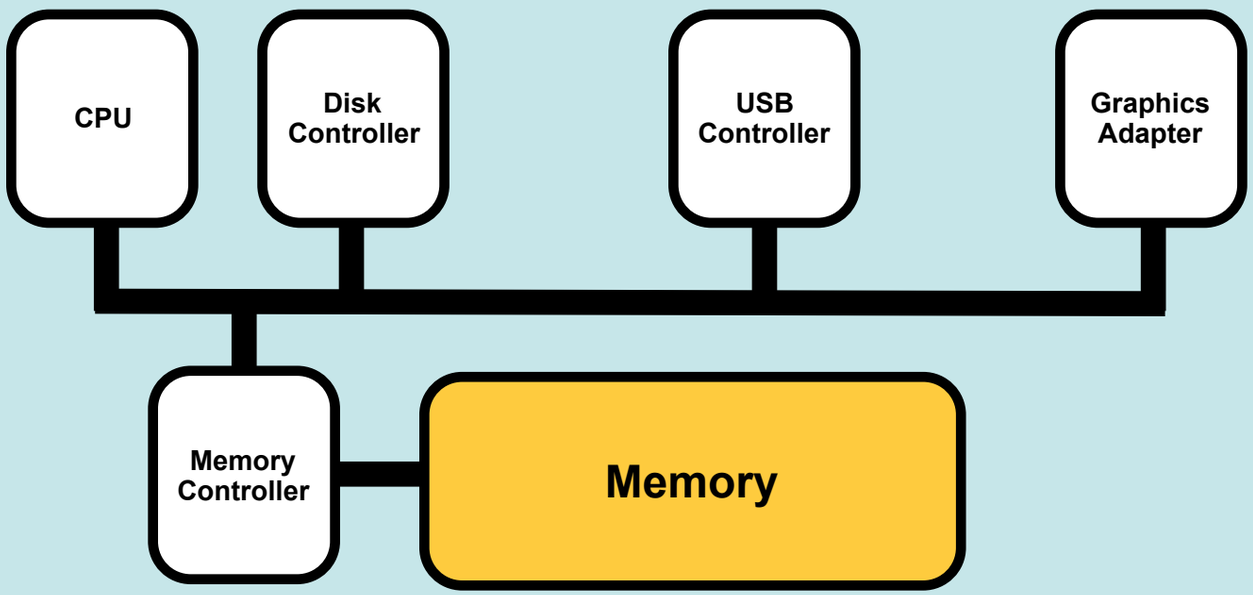
User 1 User 2 User 3 User N



Compiler Spreadsheet Text Editor Pacman

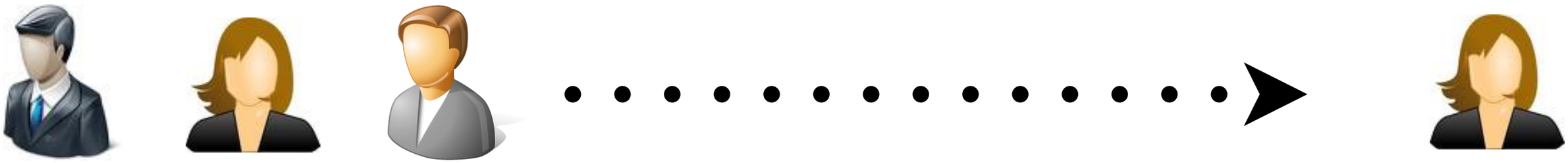
System and Application Programs

Operating System



Computer Hardware

User 1 User 2 User 3 User N



Compiler Spreadsheet Text Editor Pacman

System and Application Programs

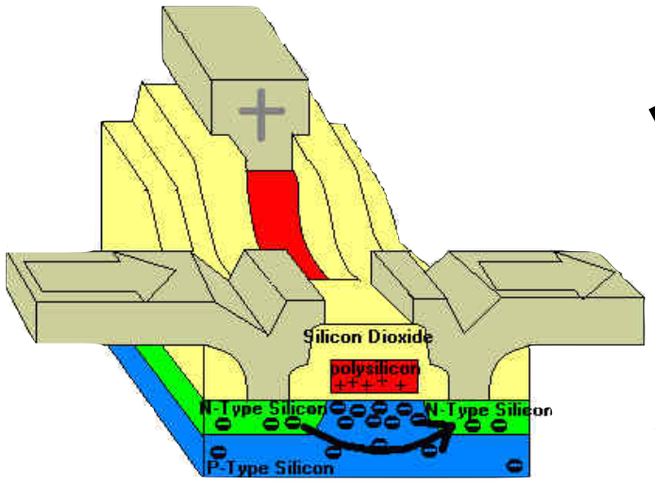
Operating System

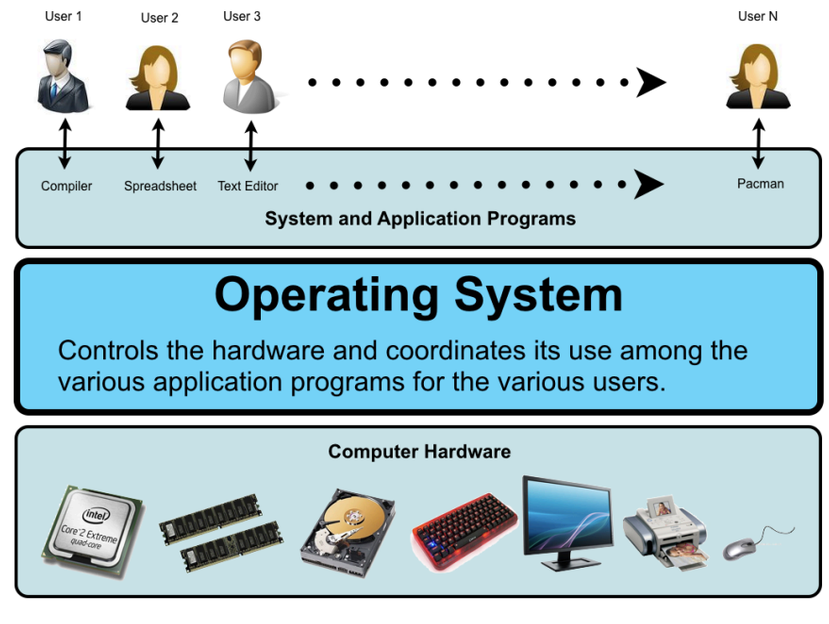
Controls the hardware and coordinates its use among the various application programs for the various users.

Computer Hardware

Bottom up

Top down





The cognitive domain

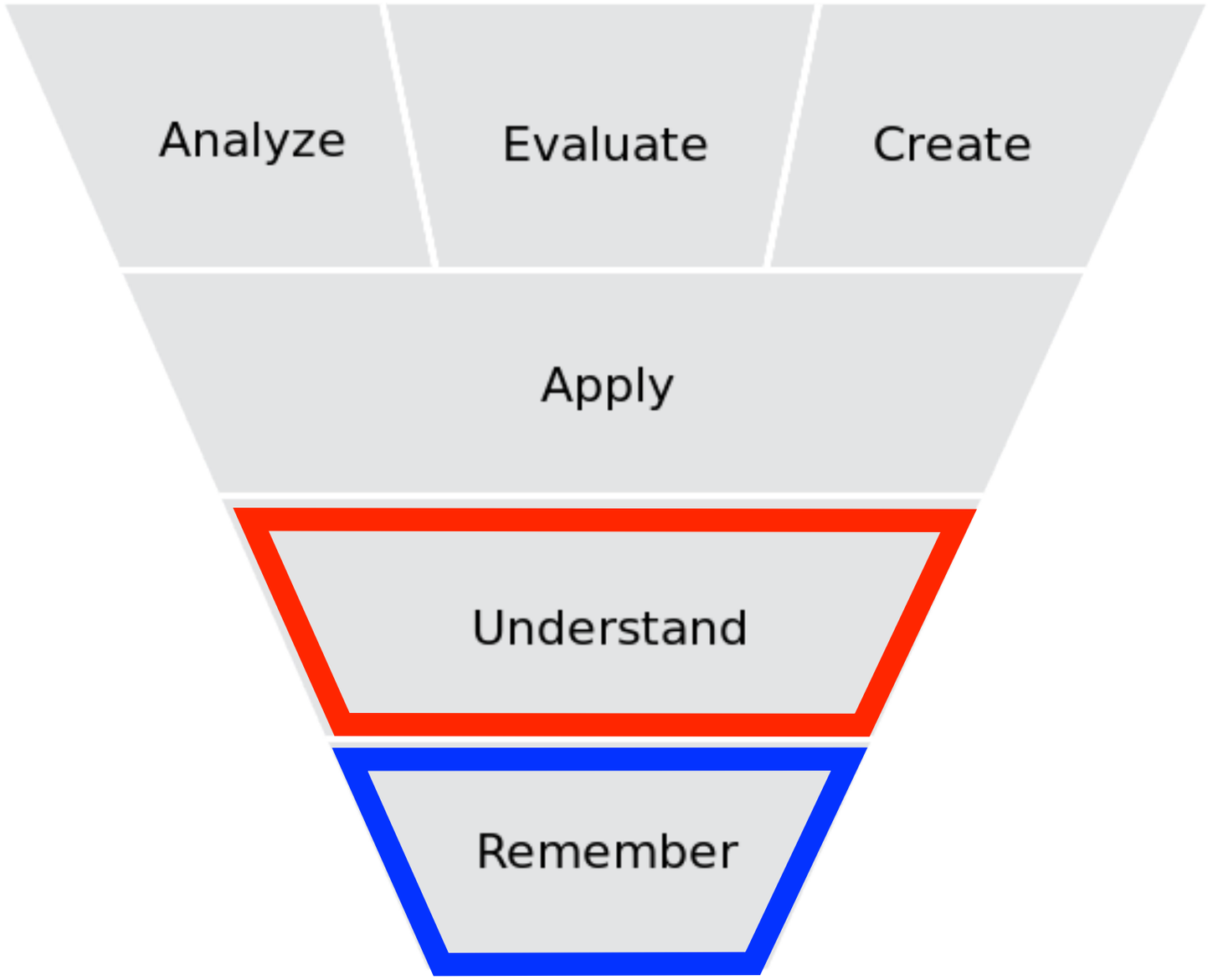


Image: http://commons.wikimedia.org/wiki/File:BloomsCognitiveDomain.svg 2016-01-17

Important definitions

CPU

The central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

Register

A processor register is a quickly accessible location available to a computer's central processing unit (CPU). Registers usually consist of a small amount of fast storage. A CPU only has a small number of registers.

Memory

Memory refers to the computer hardware integrated circuits that store information for immediate use in a computer; it is synonymous with the term "primary storage". The memory is much slower than the CPU register but much larger in size.

Sources https://en.wikipedia.org/wiki/Central_processing_unit 2018-01-16
https://en.wikipedia.org/wiki/Processor_register 2018-01-16
https://en.wikipedia.org/wiki/Computer_memory 2018-01-16

CPU context

At any point in time, the values of all the registers in the CPU defines the CPU context. Sometimes CPU state is used instead of CPU context.

Program

A set of instructions which is in human readable format. A passive entity stored on secondary storage.

Executable

A compiled form of a program including machine instructions and static data that a computer can load and execute. A passive entity stored on secondary storage.

Process

A program loaded into memory and executing or waiting. A process typically executes for only a short time before it either finishes or needs to perform I/O (waiting). A process is an active entity and needs resources such as CPU time, memory etc to execute.

Kernel

The kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system.

Source https://en.wikipedia.org/wiki/Kernel_(operating_system)

System and Application Programs

Operating System

Controls the hardware and coordinates its use among the various application programs for the various users.

Bootstrap program

Kernel

Computer Hardware

System and Application Programs

Operating System

Controls the hardware and coordinates its use among the various application programs for the various users.

Bootstrap program

Kept on chip (ROM or EEPROM), aka **firmware**.

Small program executed on power up or reboot.

Initializes all aspects of the system, from CPU register to device controllers to memory content.

Locates and **loads the kernel into memory** for execution.

Kernel

The part of the operating system that is running at all times.

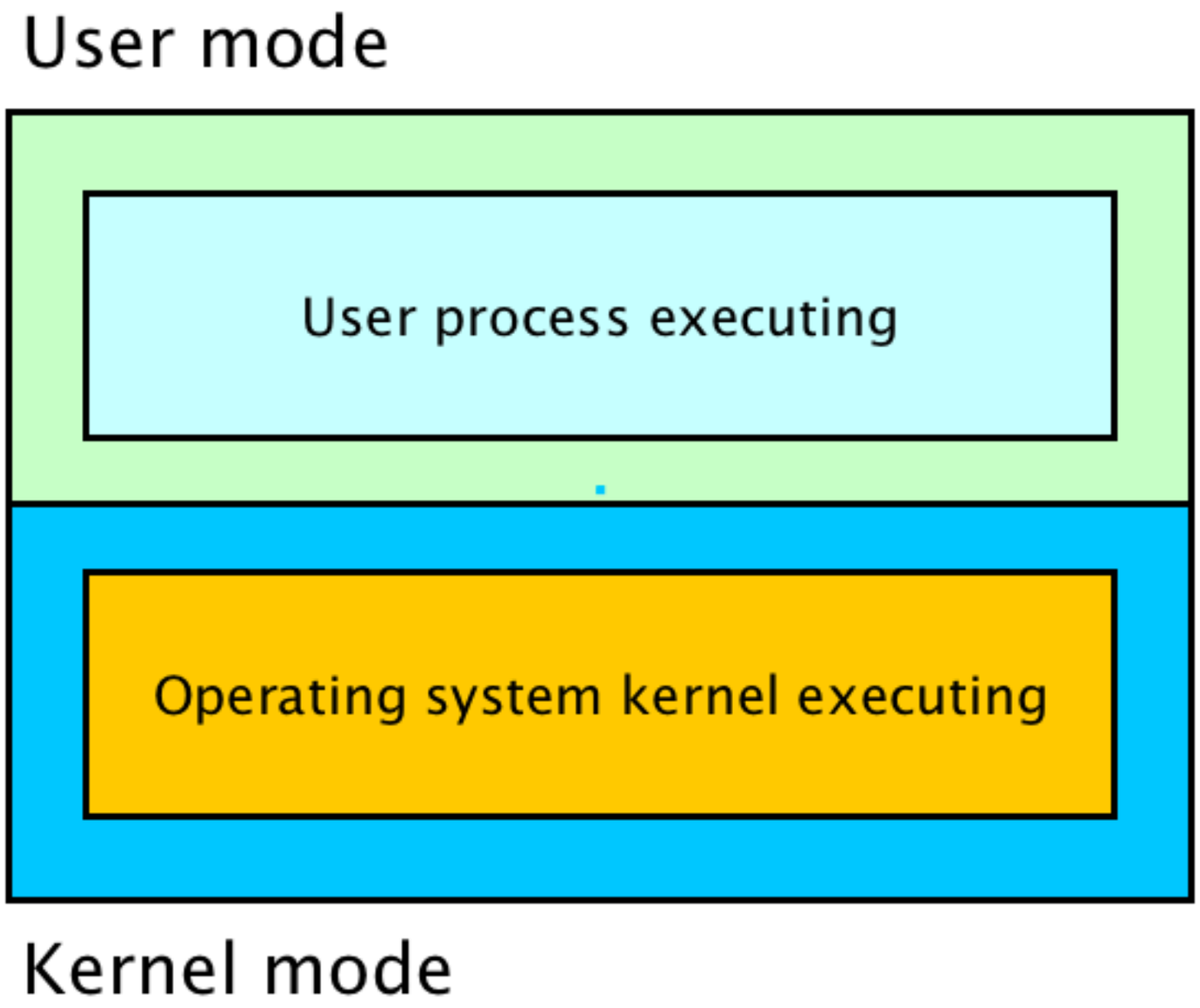
On boot, starts executing the first process such as **init**.

Waits for some **event** to occur...

Computer Hardware

Dual mode operation

In order to protect the operating system from user processes and protect user processes from each other, two modes are provided by the hardware: **user mode** and **kernel mode**.



Dual mode operation place restrictions on the type and scope of operations that can be executed by the CPU. This design allows the operating system kernel to execute with more privileges than user application processes.

Synchronous and asynchronous events

Synchronous means happening, existing, or arising at precisely the same time. [1]

Asynchronous simply means “not synchronous”.

If an event occurs at the same instruction every time the program is executed with the same data and memory allocation, the event is synchronous. An synchronous event is directly related to the instruction currently being executed by the CPU.

On the other hand, an asynchronous event is not directly related to the instruction currently being executed by the CPU.

1) https://www.merriam-webster.com/dictionary/synchronous

Exceptions and interrupts

Interrupts and exceptions are used to notify the CPU of events that needs immediate attention during program execution.

Exceptions are internal and synchronous

- ★ Exceptions are used to handle internal program errors.
- ★ Overflow, division by zero and bad data address are examples of internal errors in a program.
- ★ Another name for exception is trap. A trap (or exception) is a software generated interrupt.
- ★ Exceptions are produced by the CPU control unit while executing instructions and are considered to be synchronous because the control unit issues them only after terminating the execution of an instruction.

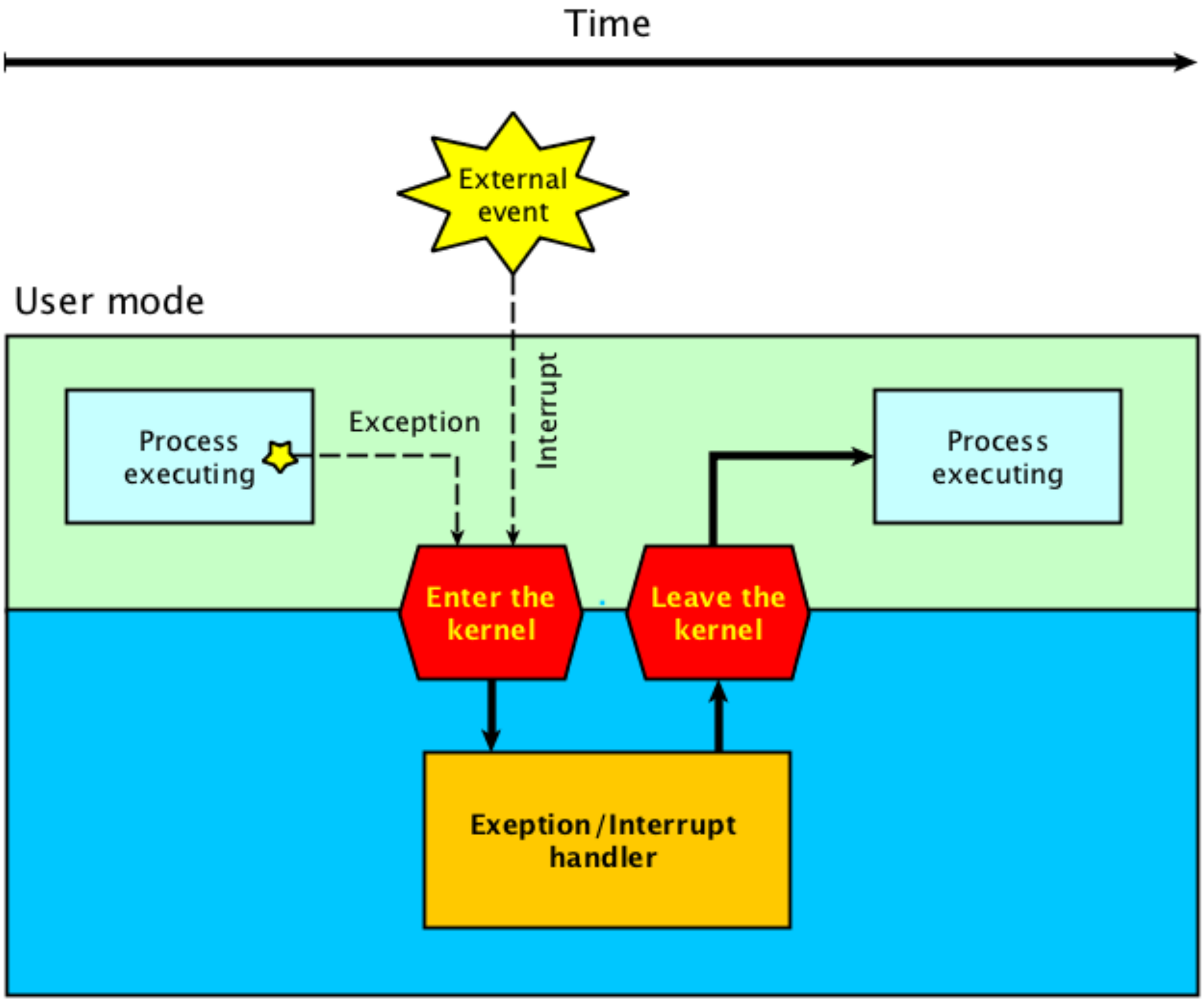
Interrupts are external and asynchronous

- ★ Interrupts are used to notify the CPU of external events.
- ★ Interrupts are generated by hardware devices outside the CPU at arbitrary times with respect to the CPU clock signals and are therefore considered to be asynchronous.
- ★ Key-presses on a keyboard might happen at any time. Even if a program is run multiple times with the same input data, the timing of the key presses will most likely vary.

- ★ Read and write requests to disk is similar to key presses. The disk controller is external to the executing process and the timing of a disk operation might vary even if the same program is executed several times.

Exception and interrupt handling

- ★ When an exception or interrupt occurs, execution **transition** from user mode to kernel mode.
- ★ The **cause** of the interrupt or exception is determined.
- ★ The exception or interrupt is **handled**.
- ★ When the exception or interrupt has been handled execution **resumes** in user space.


Time
External event
User mode
Process executing
Exception
Interrupt
Process executing
Enter the kernel
Leave the kernel
Exeption/Interrupt handler
Kernel mode

Problem

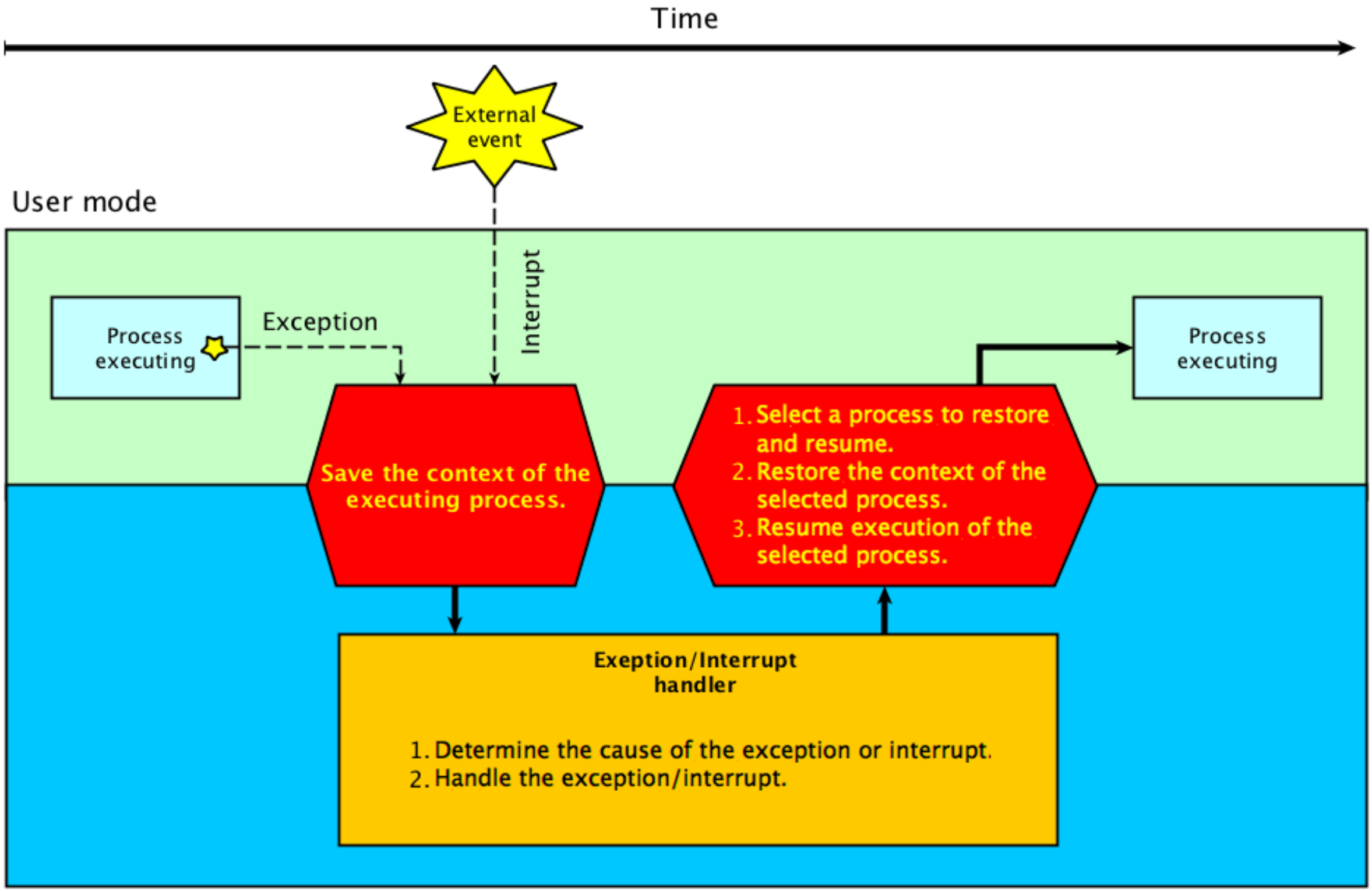
The exception/interrupt handler uses the same CPU (including the program counter and all registers) as the user processes.

Saving context

When entering the exception/interrupt handler, values in all registers must be saved to memory before the kernel can use the registers.

Restoring context

Before resuming execution of a user process, the values of all registers must be restored using the values saved to memory earlier.


Time
External event
User mode
Process executing
Exception
Interrupt
Process executing
Save the context of the executing process.
1. Select a process to restore and resume.
2. Restore the context of the selected process.
3. Resume execution of the selected process.
Exeption/Interrupt handler
1. Determine the cause of the exception or interrupt.
2. Handle the exception/interrupt.
Kernel mode

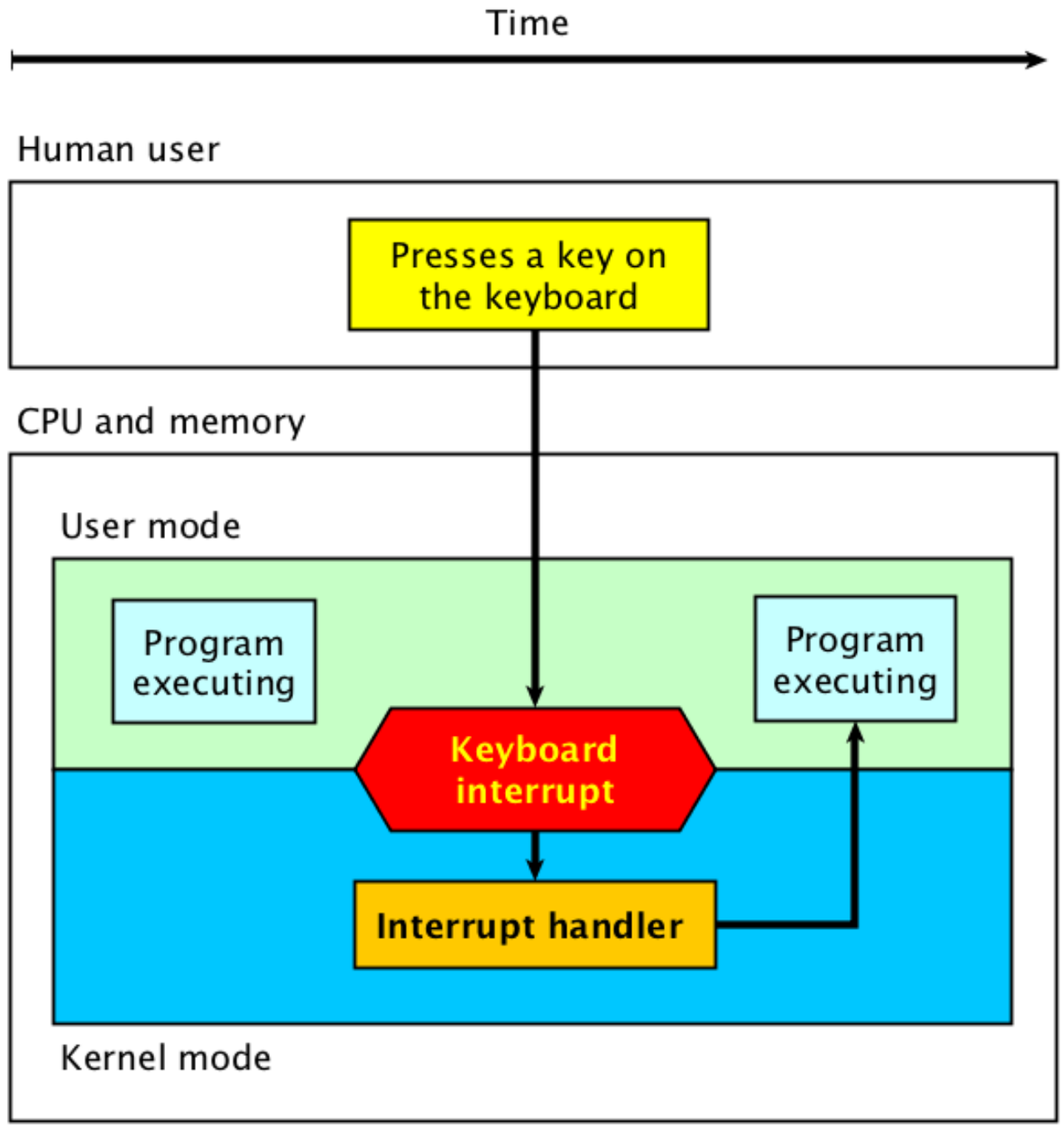
Waiting for keyboard input

- Humans are very slow compared to the CPU. No matter how fast you are, the CPU will be able to execute a huge number of instructions between every key-press you make.

- Is it possible to make the CPU do something useful while waiting for user input?

Interrupt driven input

Using interrupts makes it possible for the CPU to do something useful, for example execute another program, while waiting for user input.



Multiprogramming

Job

In a multiprogramming system a program loaded to memory and ready to execute is called a job.

Execute another job while waiting for I/O

The simple idea is to make a job wait for I/O "outside" the CPU. When a job makes a request for I/O the CPU will execute another job while the the first job waits for the/O request to complete.

Interrupts

Interrupts are used to notify the system when an I/O request is completed.

States

In a multiprogramming system, a job can be in one of three states.

Running

The job is currently executing on the CPU. At any time, at most one job can be in this state.

Ready

The job is ready to run but currently not selected to do so.

Waiting

The job is blocked from running on the CPU while waiting for an I/O request to be completed.

Memory

Job 1

Ready

Job 2

Ready

Job 3

Ready

Job 4

Ready

CPU

In a multiprogramming system, several jobs are kept in memory at the same time. Initially, all jobs are int the ready state.

Memory

CPU

Job 1

Running

Job 2

Ready

Job 3

Ready

Job 4

Ready

One of the ready jobs is selected to execute on the CPU and changes state from ready to running. In this example, job 1 is selected to execute.

Memory

CPU

Job 1

Waiting

Job 2

Ready

Job 3

Ready

Job 4

Ready

Eventually, the running job makes a request for I/O and the state changes from running to waiting.

Memory

CPU

Job 1

Waiting

Job 2

Ready

Job 3

Running

Instead of idle waiting for the I/O request to complete, one of the ready jobs is selected to execute on the CPU and have its state change from ready to running. In this example job 3 is selected to execute.

Job 4

Ready

Memory

CPU

Job 1

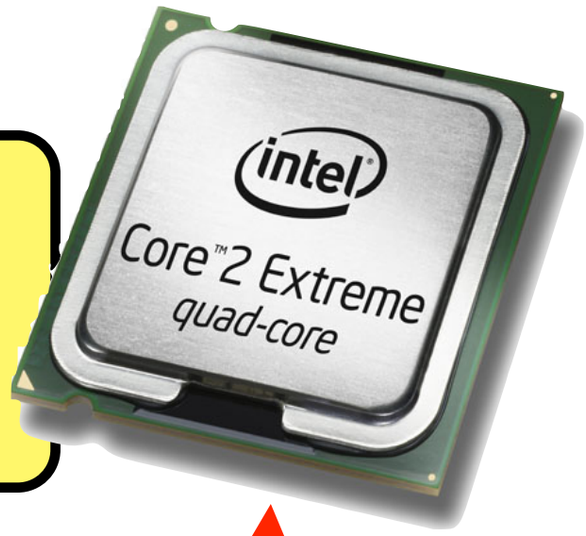
Waiting

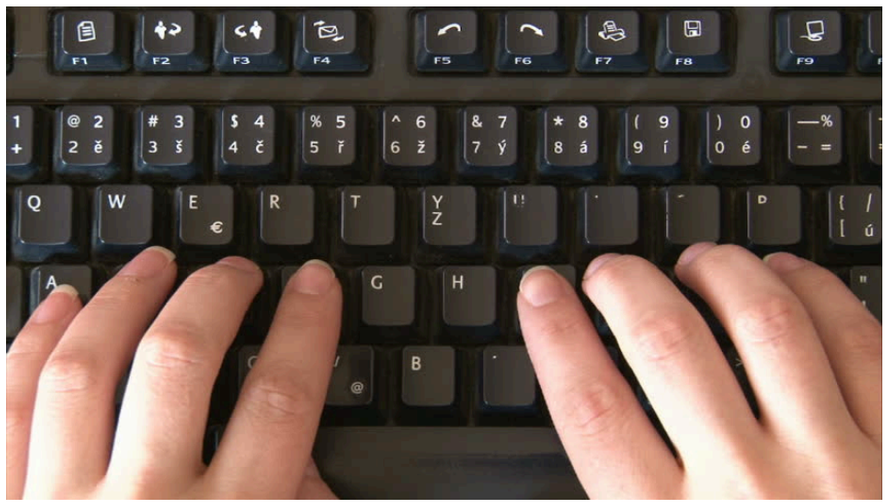
Job 2

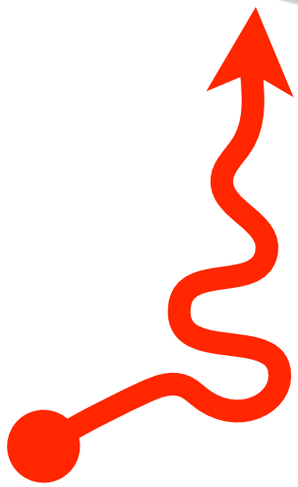
Ready

Job 3

Running







Interrupt

Job 4

Ready

Eventually the I/O request job 1 is waiting for will complete and the CPU will be notified by an interrupt. In this example, job 1 was waiting for a keypress on the keyboard.

Memory

Job 1

Ready

Job 2

Ready

Job 4

Ready

CPU

Job 3

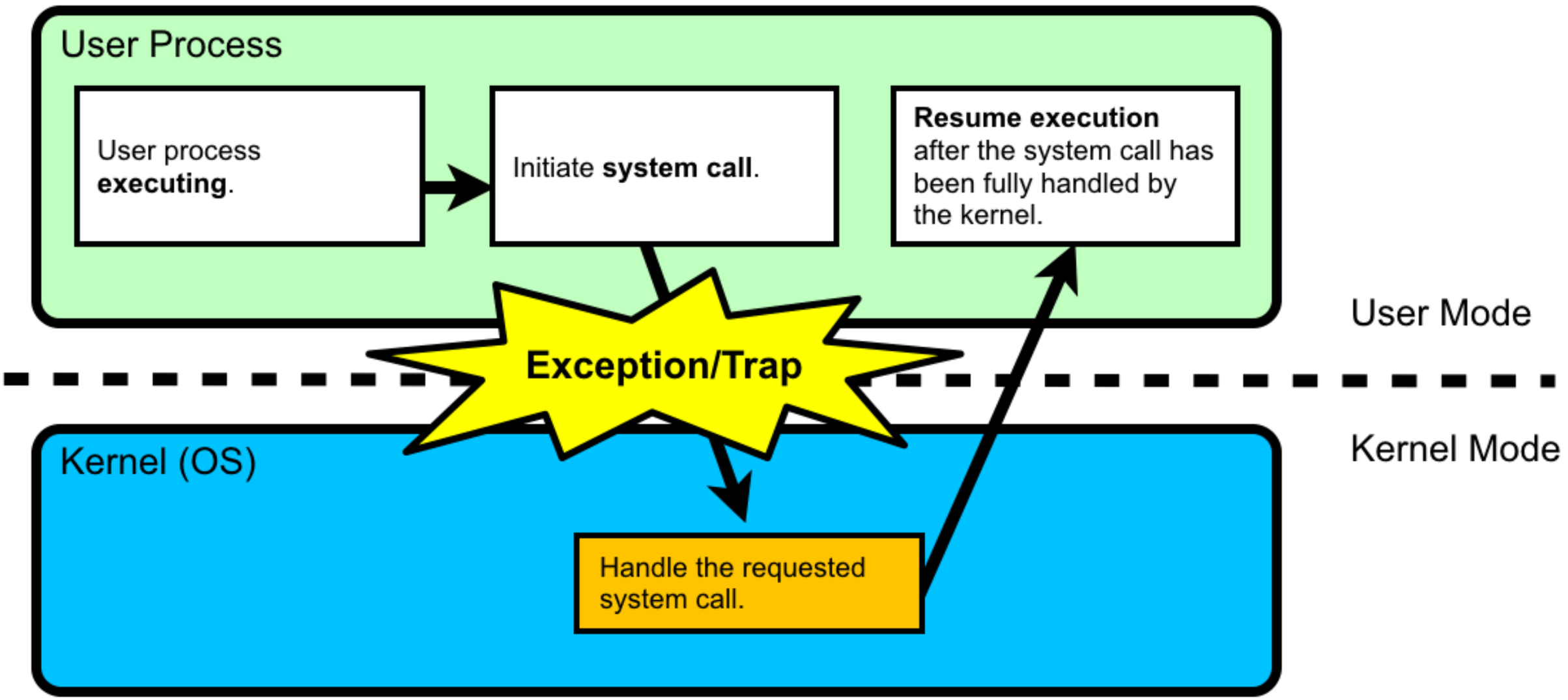
Running

The state of the waiting job (job 1) will change from waiting to ready.

Job 3 will execute until making a request for I/O.

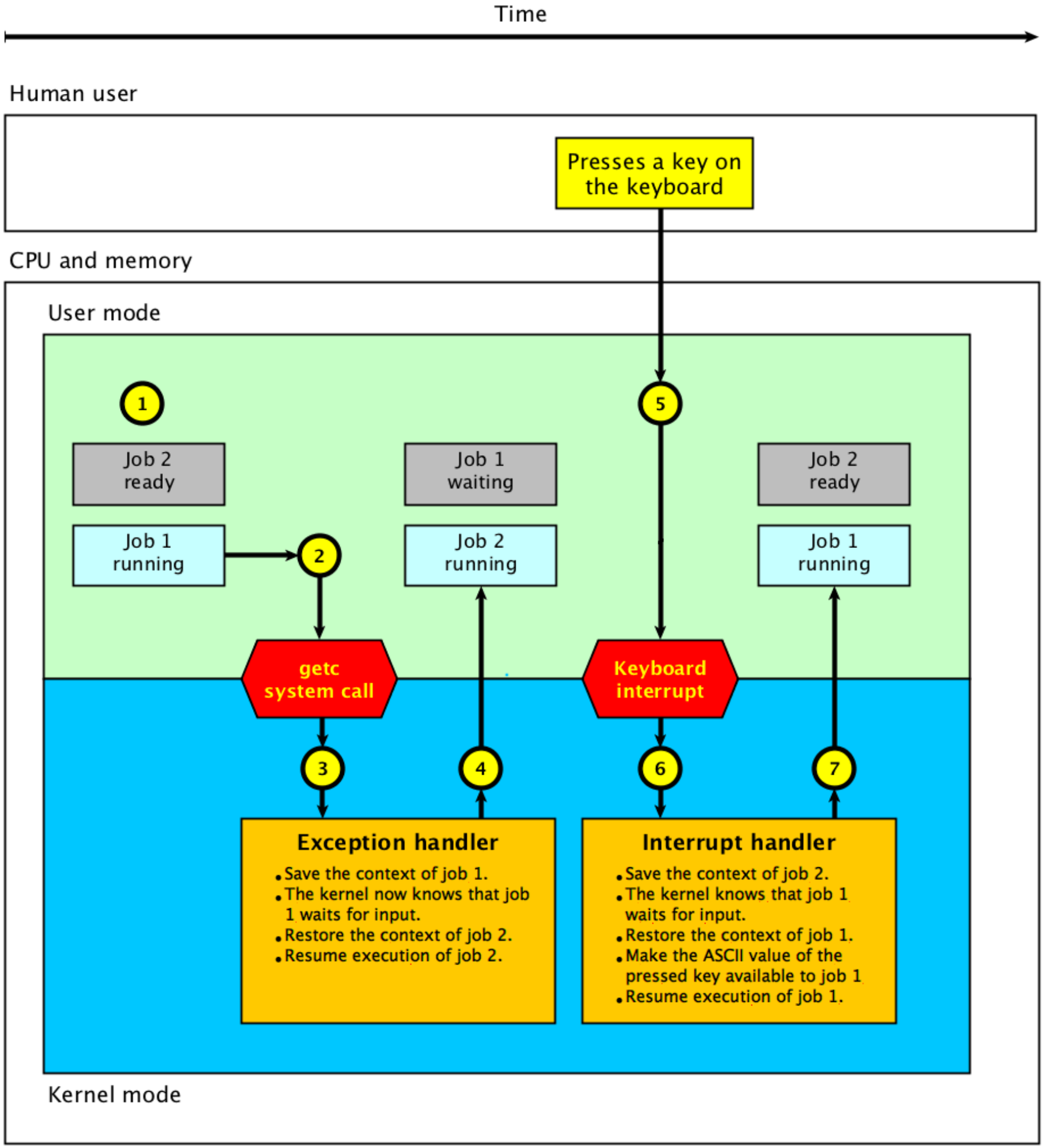
System call

A user program requests service from the operating system using system calls. System calls are implemented using a special system call exception. Another name for exception is trap. System calls forms an interface between user programs and the operating system.



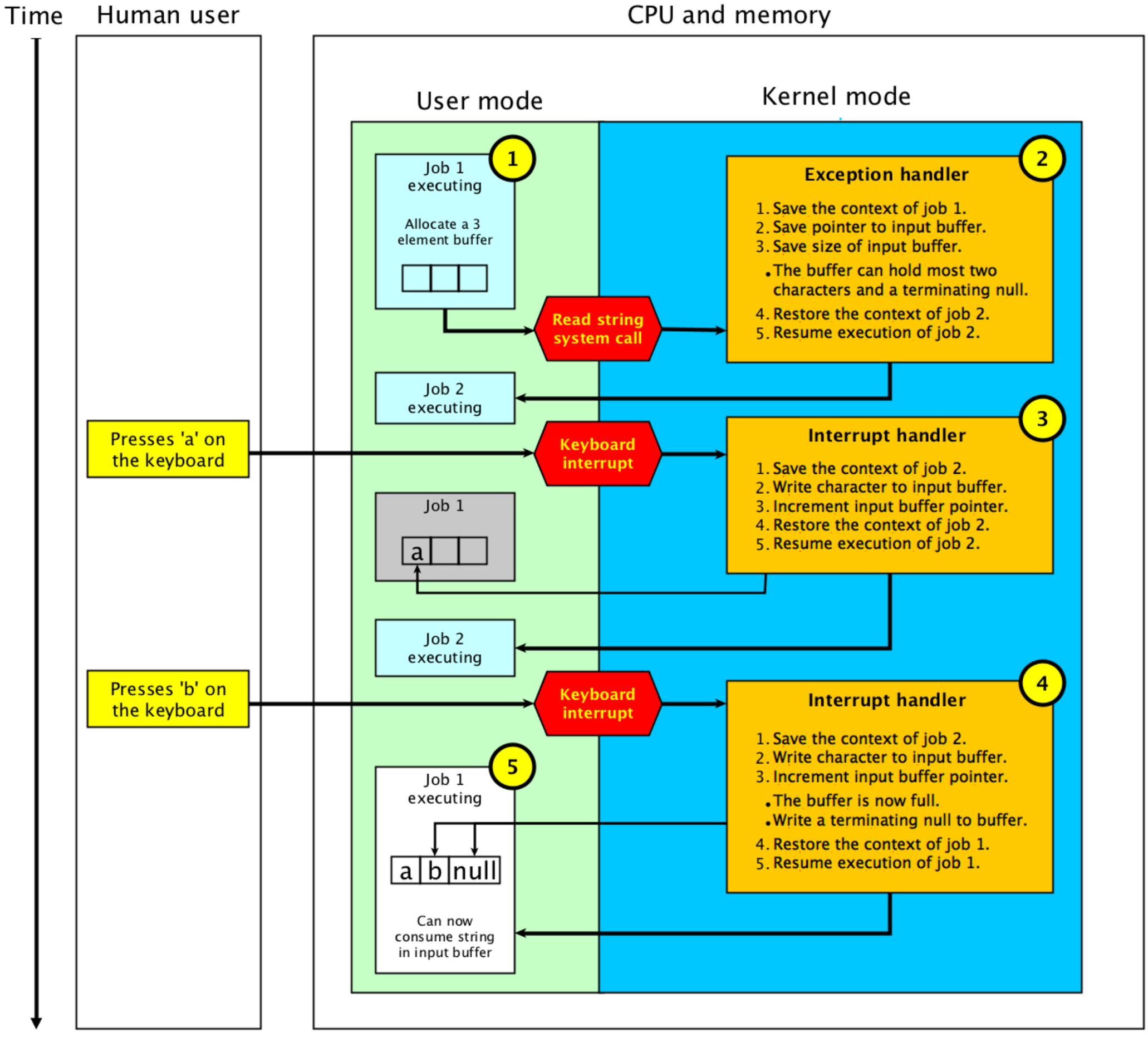
Read character system call design

Let’s sketch the design for a system call similar to the C library function **`getc`** that allows a program to read a single character typed by a human user on the keyboard.


Time
Human user
Presses a key on the keyboard
CPU and memory
User mode
1
Job 2 ready
Job 1 running
2
Job 1 waiting
Job 2 running
5
Job 2 ready
Job 1 running
getc system call
Keyboard interrupt
3
4
6
7
Exception handler
• Save the context of job 1.
• The kernel now knows that job 1 waits for input.
• Restore the context of job 2.
• Resume execution of job 2.
Interrupt handler
• Save the context of job 2.
• The kernel knows that job 1 waits for input.
• Restore the context of job 1.
• Make the ASCII value of the pressed key available to job 1
• Resume execution of job 1.
Kernel mode

Read string system call design

Let’s sketch the design for a system call similar to the C library function **`gets`** that allows a program to read a string typed by a human on the keyboard.


Time
Human user
CPU and memory
User mode
Kernel mode
Job 1 executing
Allocate a 3 element buffer
1
Exception handler
2
1. Save the context of job 1.
2. Save pointer to input buffer.
3. Save size of input buffer.
• The buffer can hold most two characters and a terminating null.
4. Restore the context of job 2.
5. Resume execution of job 2.
Read string system call
Job 2 executing
Presses 'a' on the keyboard
Keyboard interrupt
Interrupt handler
3
1. Save the context of job 2.
2. Write character to input buffer.
3. Increment input buffer pointer.
4. Restore the context of job 2.
5. Resume execution of job 2.
Job 1
a
Job 2 executing
Presses 'b' on the keyboard
Keyboard interrupt
Interrupt handler
4
1. Save the context of job 2.
2. Write character to input buffer.
3. Increment input buffer pointer.
• The buffer is now full.
• Write a terminating null to buffer.
4. Restore the context of job 1.
5. Resume execution of job 1.
5
Job 1 executing
a b null
Can now consume string in input buffer