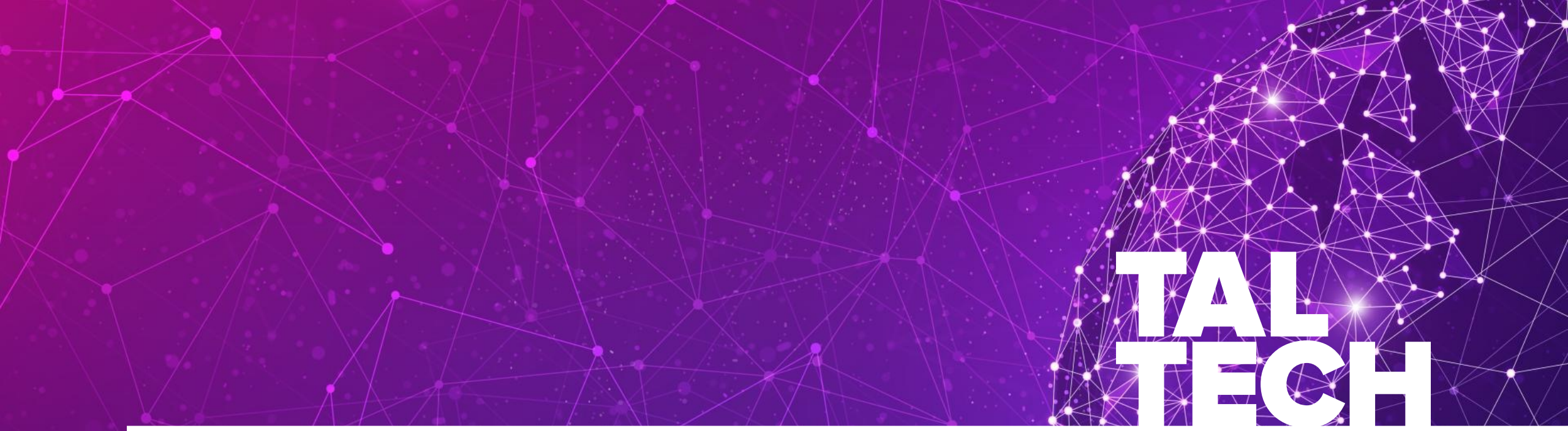


MICROPROCESSOR SYSTEMS (IAS0430)

Department of Computer Systems
Tallinn University of Technology

KERNEL VS USER

- Now that we know what makes a CPU.
  - What is the CPU made of?
  - How does it run?
  - What makes it operate?
  - How does it know what to operate on?
  - Where does it find things?
- Next is to learn how it interacts with **programs**:
  - More specifically, **which program is allowed to do what**?
  - In reality, the CPU must protect itself and its operations.
    - It must only allow certain programs to access certain operations
      - **Why?**

KERNEL VS USER

- Now that we know what makes a CPU.
  - What is the CPU made of?
  - How does it run?
  - What makes it operate?
  - How does it know what to operate on?
  - Where does it find things?
- Next is to learn how it interacts with **programs**:
  - More specifically, **which program is allowed to do what**?
  - In reality, the CPU must protect itself and its operations.
    - It must only allow certain programs to access certain operations
      - **Why?**
      - Because some operation may completely destroy everything!!!!!!

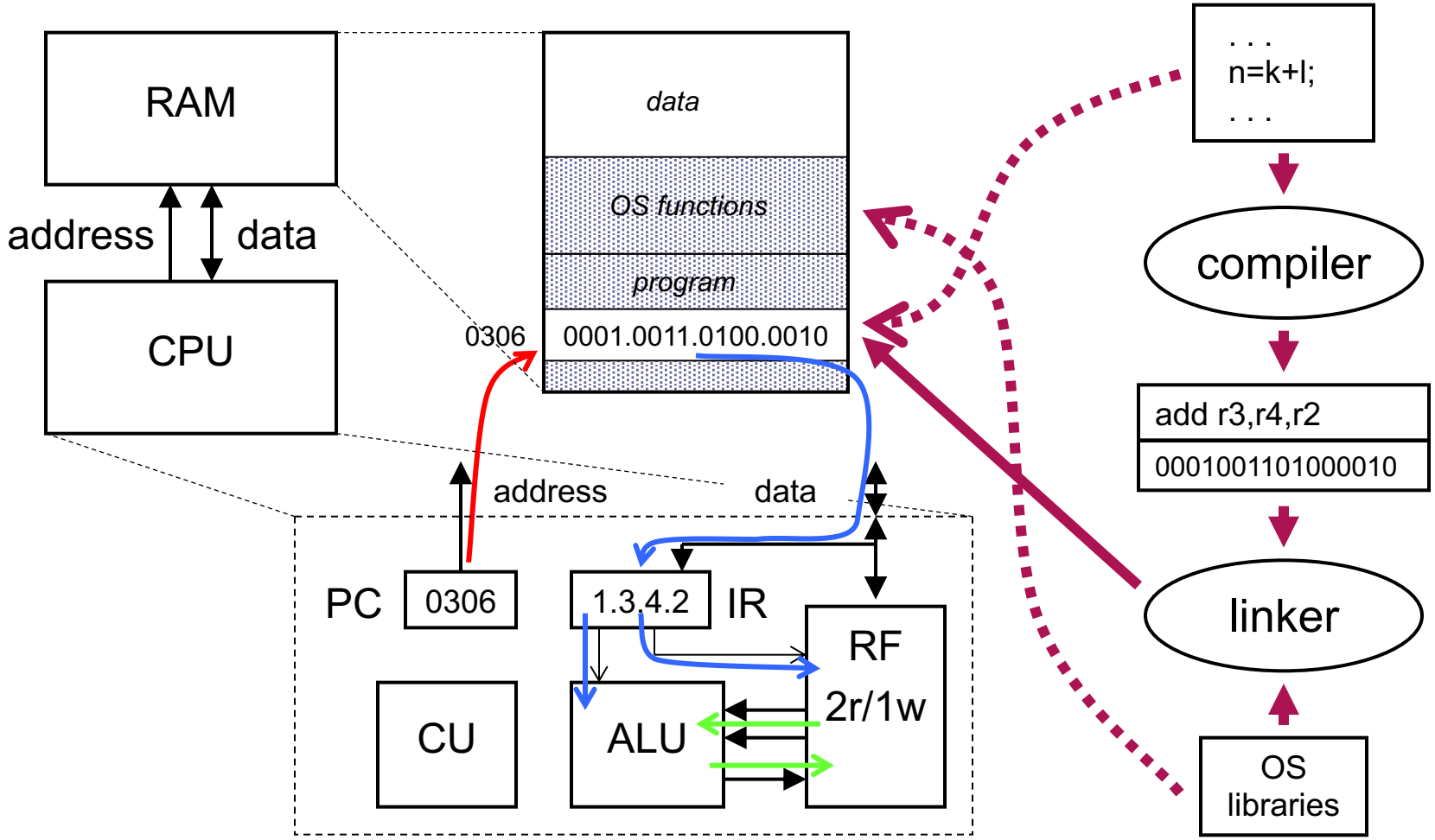
KERNEL VS USER

- Now that we know what makes a CPU.
  - What is the CPU made of?
  - How does it run?
  - What makes it operate?
  - How does it know what to operate on?
  - Where does it find things?
- Next is to learn how it interacts with **programs**:
  - More specifically, **which program is allowed to do what**?
  - In reality, the CPU must protect itself and its operations.
    - It must only allow certain programs to access certain operations
      - **Why?**
      - Because some operation may completely destroy everything!!!!!!
    - As well as only allow specific locations in memory to be accesses.
      - **Why?**

KERNEL VS USER

- Now that we know what makes a CPU.
  - What is the CPU made of?
  - How does it run?
  - What makes it operate?
  - How does it know what to operate on?
  - Where does it find things?
- Next is to learn how it interacts with **programs**:
  - More specifically, **which program is allowed to do what**?
  - In reality, the CPU must protect itself and its operations.
    - It must only allow certain programs to access certain operations
      - **Why?**
      - Because some operation may completely destroy everything!!!!!!
    - As well as only allow specific locations in memory to be accesses.
      - **Why?**
      - Some location in memory **contain sensitive information** that must not be changed/updated/altered/removed – **NOT A TOY TO PLAY WITH**

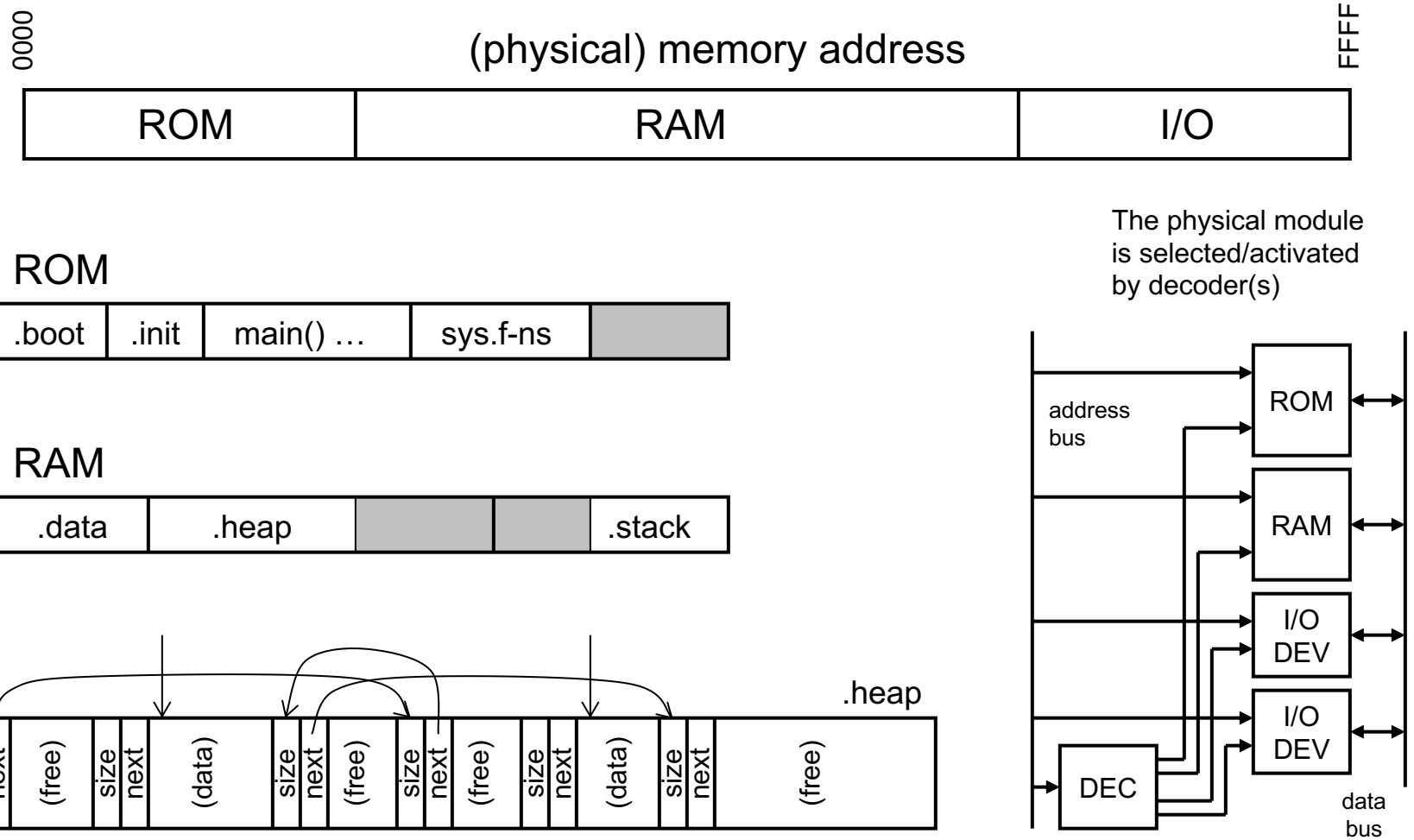
RUNNING A PROGRAM



PROGRAM IN MEMORY

- **Single** application/program
  - Memory layout
    - Program – application + system
    - Data – static / dynamic / stack
  - Program running
    - Starting up (booting)
      - Initializing processor
      - Initializing program
    - Program in loop
    - System calls

MEMORY LAYOUT – SINGLE APPLICATION

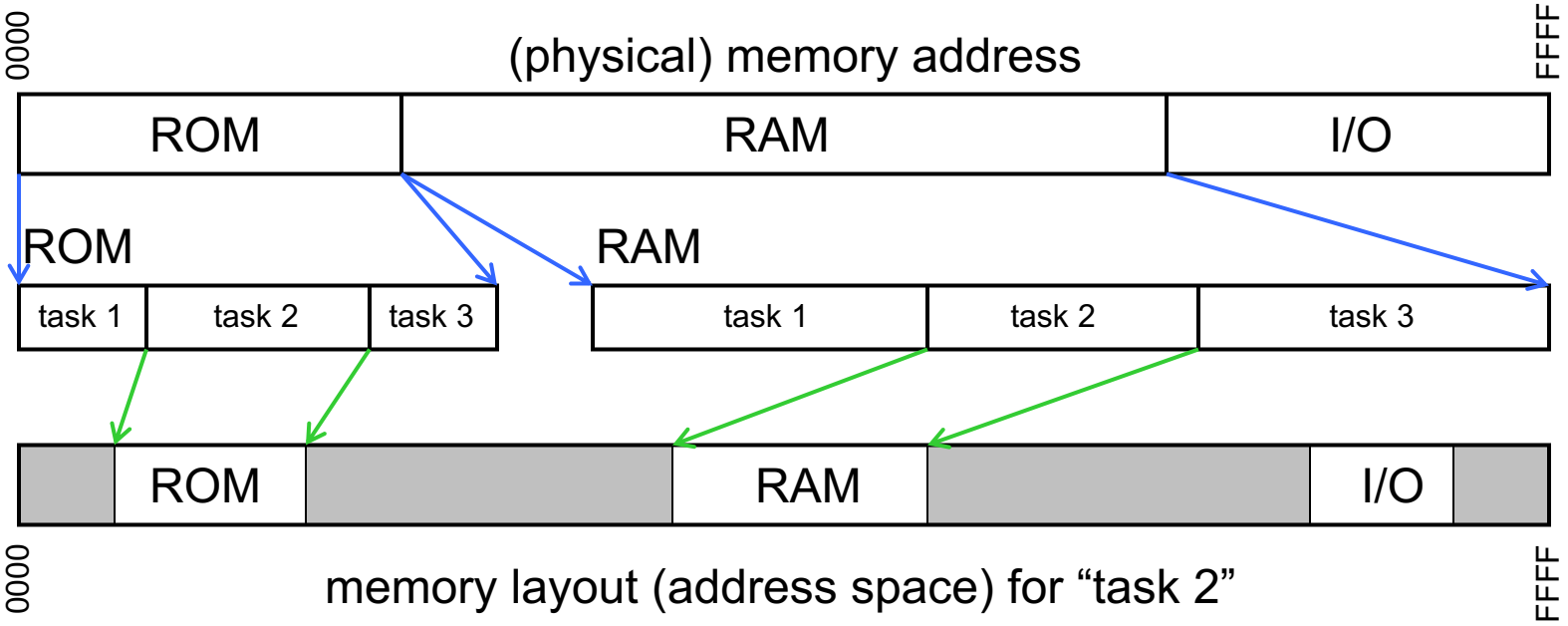


PROGRAMS IN MEMORY

- **Multiple** applications/programs
  - in addition to the single application
- Memory layout
  - Different programs & their data at different memory locations
  - Protecting data
    - MMU – Memory Management Unit
- Programs running
  - Initializing system
  - Changing applications/programs/tasks
    - scheduling, saving CPU status, …

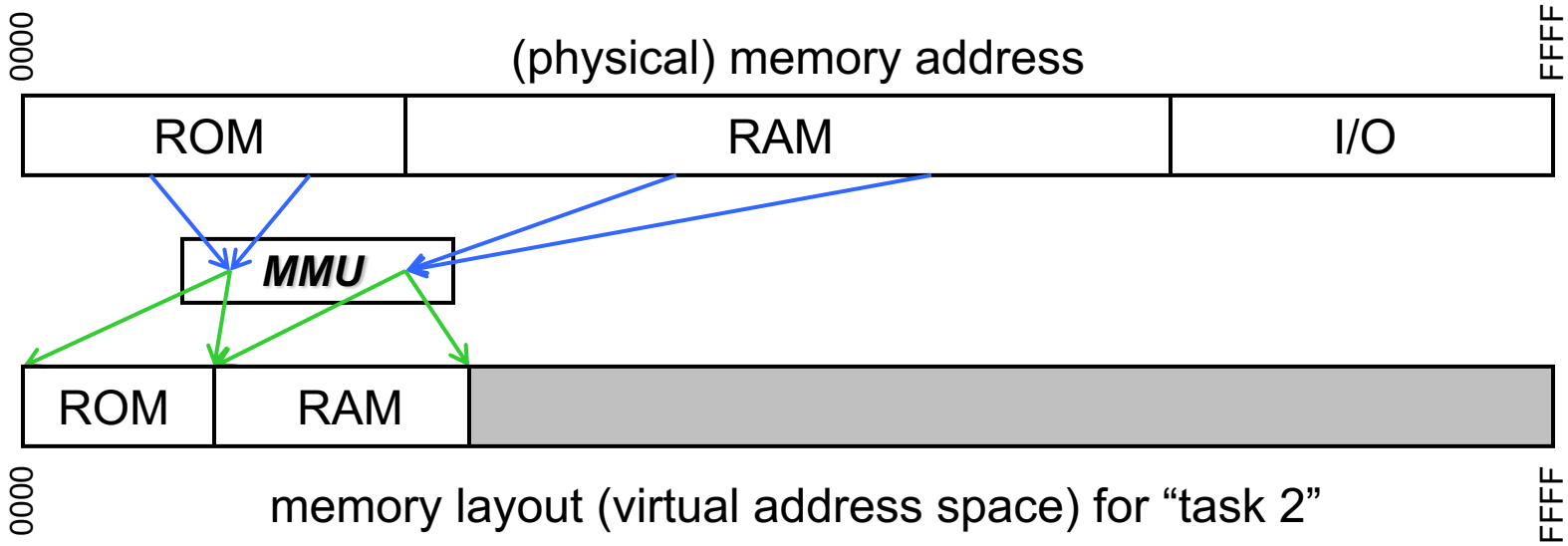
MEMORY LAYOUT – MULTIPLE APPLICATIONS

- **Every** application/program at different physical address
  - Internal memory layout is the same for every application (.data, .heap, etc.)
  - MMU is used to protect from wrong/faulty memory accesses
  - Linking is complicated – addresses must be defined separately



MEMORY LAYOUT – MULTIPLE APPLICATIONS

- Memory layout is the **same** for **all** applications/programs
  - virtual address space (virtual memory)
  - Internal memory layout is the same for every application (.data, .heap, etc.)
  - MMU is used to protect from wrong/faulty memory accesses and also to "translate" the virtual address into physical address
  - Linking is simple – same for all applications/programs



MEMORY LAYOUT – MULTIPLE APPLICATIONS

- MMU
  - https://www.youtube.com/watch?v=2quKyPnUShQ

KERNEL VS USER

- But how do we do that?
  - How can the CPU protect itself from bad programs?

KERNEL VS USER

- But how do we do that?
  - How can the CPU protect itself from bad programs?
    - **By adding a force field – duh**

KERNEL VS USER

- But how do we do that?
  - How can the CPU protect itself from bad programs?
    - **By adding a force field – duh**
  - This force field, is simply called the **Operating System.**
    - We will not talk about the **OS** today, that's for next week.
  - Today we will talk about what makes the **OS** so …… **OS**y

  - Today we will talk about **Kernel** and **User** Modes

  - A **Kernel** is the soft part of a nut – **Yummy**
    - But what we care about is that it is **contained within a shell**
  - That is why we call the core of the computer – a Kernel
    - What is the shell of a **Computer Kernel**?

KERNEL VS USER

- But how do we do that?
  - How can the CPU protect itself from bad programs?
    - **By adding a force field – duh**
  - This force field, is simply called the **Operating System.**
    - We will not talk about the **OS** today, that's for next week.
  - Today we will talk about what makes the **OS** so …… **OS**y

  - Today we will talk about **Kernel** and **User** Modes

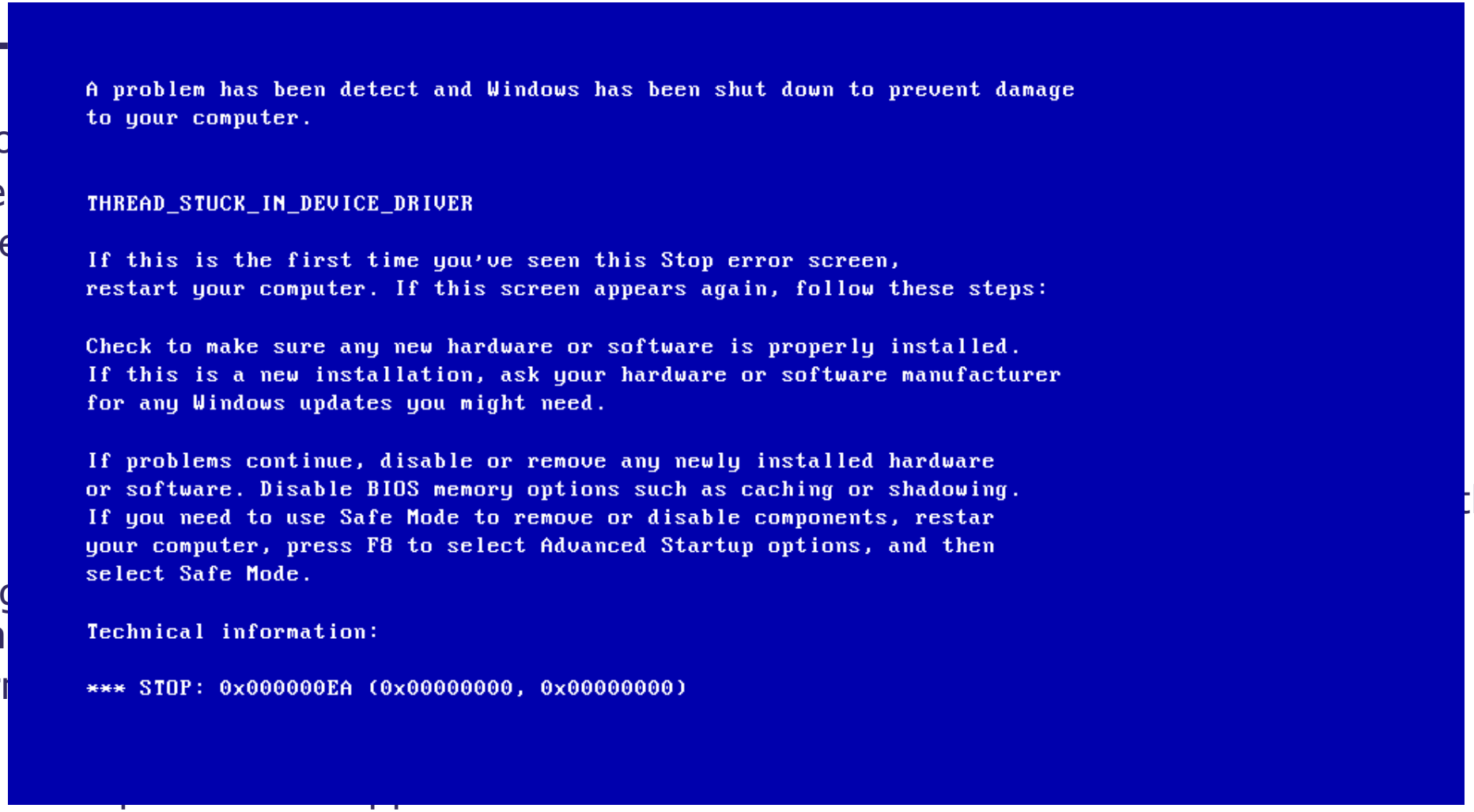
  - A **Kernel** is the soft part of a nut – **Yummy**
    - But what we care about is that it is **contained within a shell**
  - That is why we call the core of the computer – a Kernel
    - What is the shell of a **Computer Kernel**?
    - The **OS**

KERNEL VS USER

- **Kernel mode:**
  - Also known as the **super user** mode
  - The CPU is full on operation work. All is granted and all can be done.
  - When a program runs in Kernel mode
    - It means that the program is being executed with the CPU is unrestricted!
    - It has complete access to the **hardware** (CPU, ports, memory, etc.)
      - Can perform any operation allowed by the architecture.
      - Can signal any I/O device.
    - It can execute any **CPU instruction** and reference any **memory location** that it may need.
  - Programs that run in Kernel mode are usually the most **trusted and essential functions and services** of the OS.
  - Kernel mode crashes (programs that fail to execute properly) are catastrophic.
    - Will almost always result in complete system halt
    - Example of this happens in Windows?

KERNEL

- **Kernel**
  - Also
  - The
  - Whe
    -
    -
    -
    -
  - Prog **fun**
  - Kern
    -
    -
      - The blue screen of death



KERNEL VS USER

- **User Mode:**
  - When a program runs in user mode, also known as **slave mode**…
    - It has no direct communication with the **hardware** (CPU, ports, memory, etc.)
    - It can only reference **memory locations** relevant to its own execution (its own memory space).
    - It must communicate with the system **Application Program Interface (API)** in order to access hardware and memory.
  - Programs that run in User mode are usually software the interact with the user of the system directly (power point, chrome, etc..)

  - User mode crashes are **manageable** and **easy to recover** since a user mode program has no direct access to the system services or resources. Thus, does not effect the overall all functionalities of the system.

  - Kernel mode aims to **increase stability of the system by managing program access to volatile and sensitive resources.**

KERNEL VS USER

- **So to summarize:**
- **Kernel Mode**
  - Only specific system services and functions can run on CPU Kernel mode
    - Because they are important for performance
    - And they make little threat to system resources
    - Are not controlled by the User!
  - Kernel mode helps protect the core of the system functionality.

- **User Mode**
  - Programs can only access resources allocated to them by the system.
    - This is for keeping the system stable
    - Prevents programs from accessing hardware that does not belong to them
    - Allow interaction between the system hardware and the User without putting the system in jeopardy.

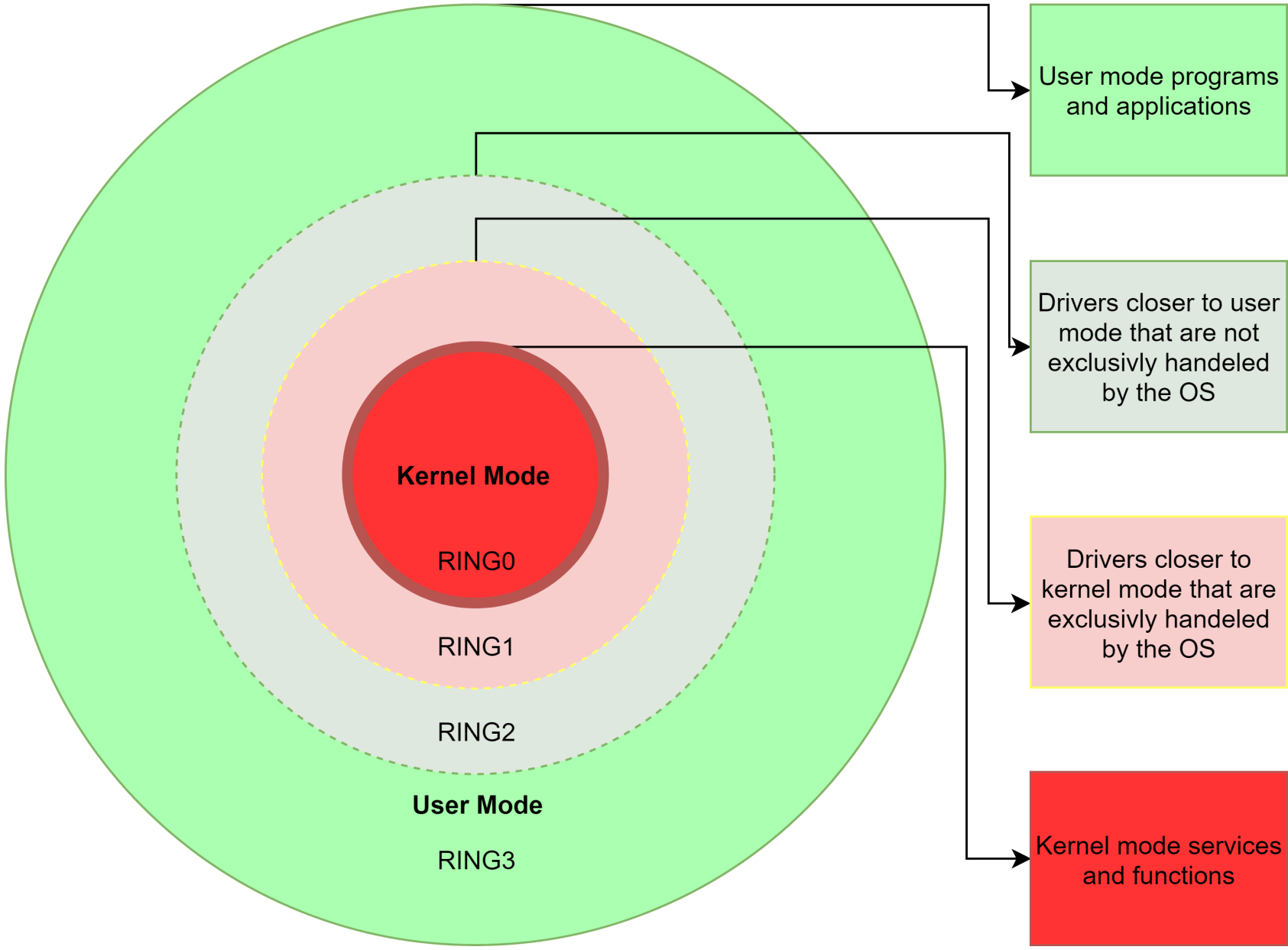
KERNEL VS USER

- **But**, what about when user mode needs to access hardware that is not managed by the system?
- There are a way to manage this. The system is divided into rings of influence

KERNEL VS USER

- **But**, what about when user mode needs to access hardware that is not managed by the system?
- There are a way to manage this. The system is divided into rings of influence
- **RING0**

Exclusive Kernel Mode programs. **Hardware management software** and **device controllers communication software.** This also includes **OS services** that need direct access to hardware

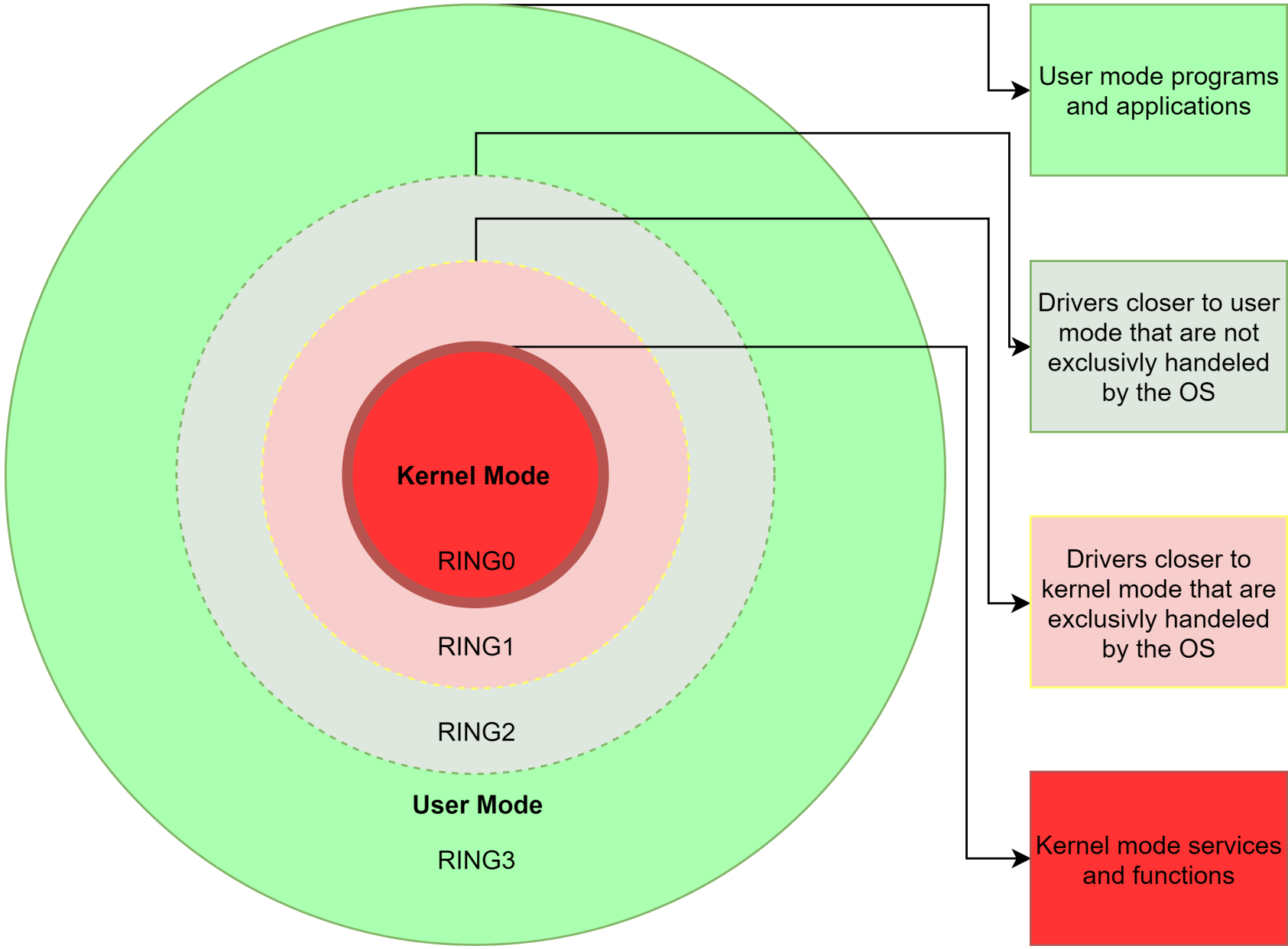


KERNEL VS USER

- **RING1**

Kernel Mode related Drivers. These are drivers such as the video card and LAN card that need driver controllers (**software**) to manage the physical drivers (**hardware**) using the OS services.

Those drivers do not get full access to kernel mode, but they have certain level of privilege that allows them to communicate on kernel mode-like way. This mostly managed by the OS, but also the system kernel manage them as well.

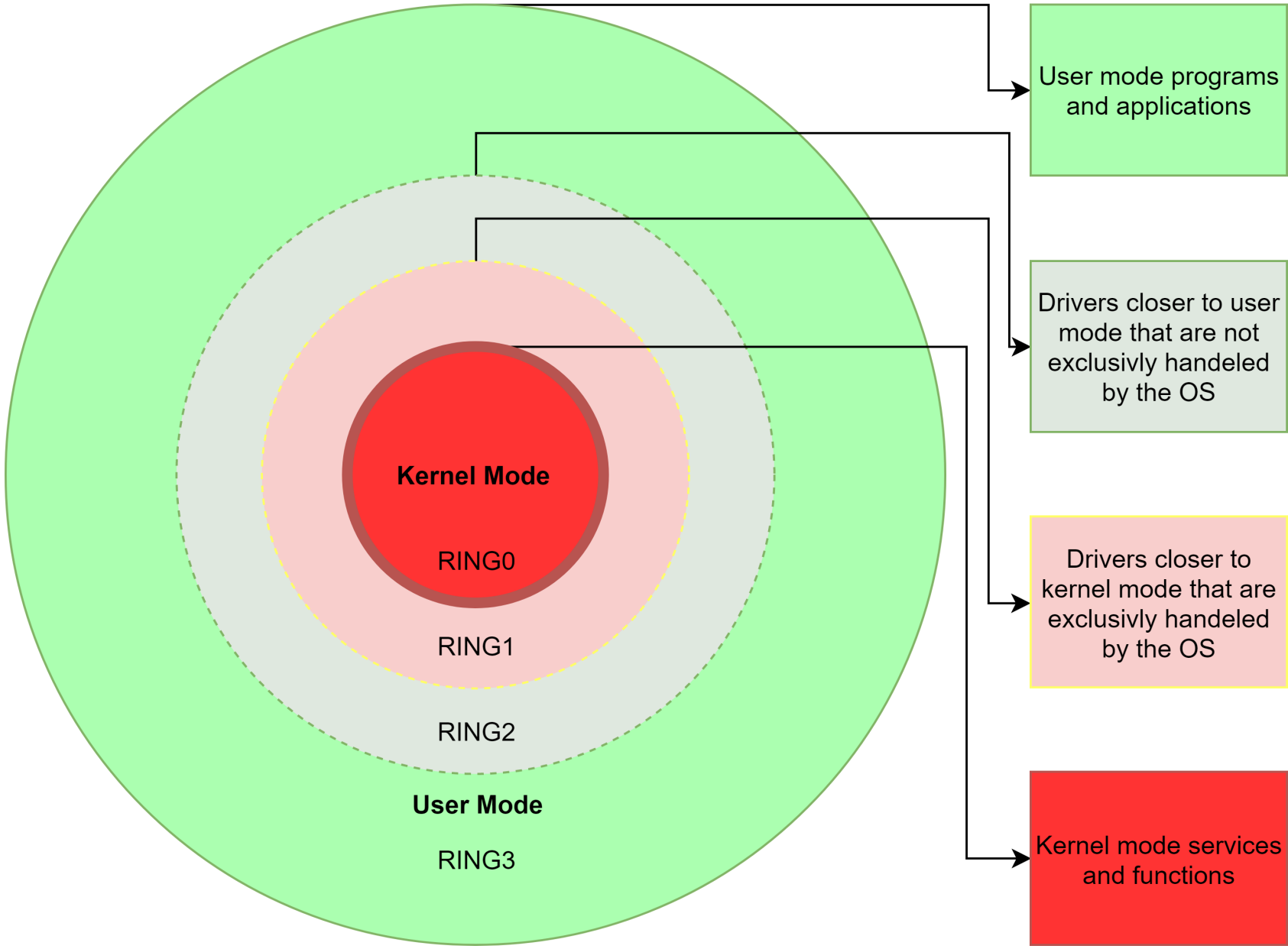


KERNEL VS USER

- **RING2**

User mode related Drivers. Drivers that do not necessarily need OS services and can be managed by software independently than the OS. Such as **mouse**, **keyboard**, sometimes **USB**, **printers, etc….**

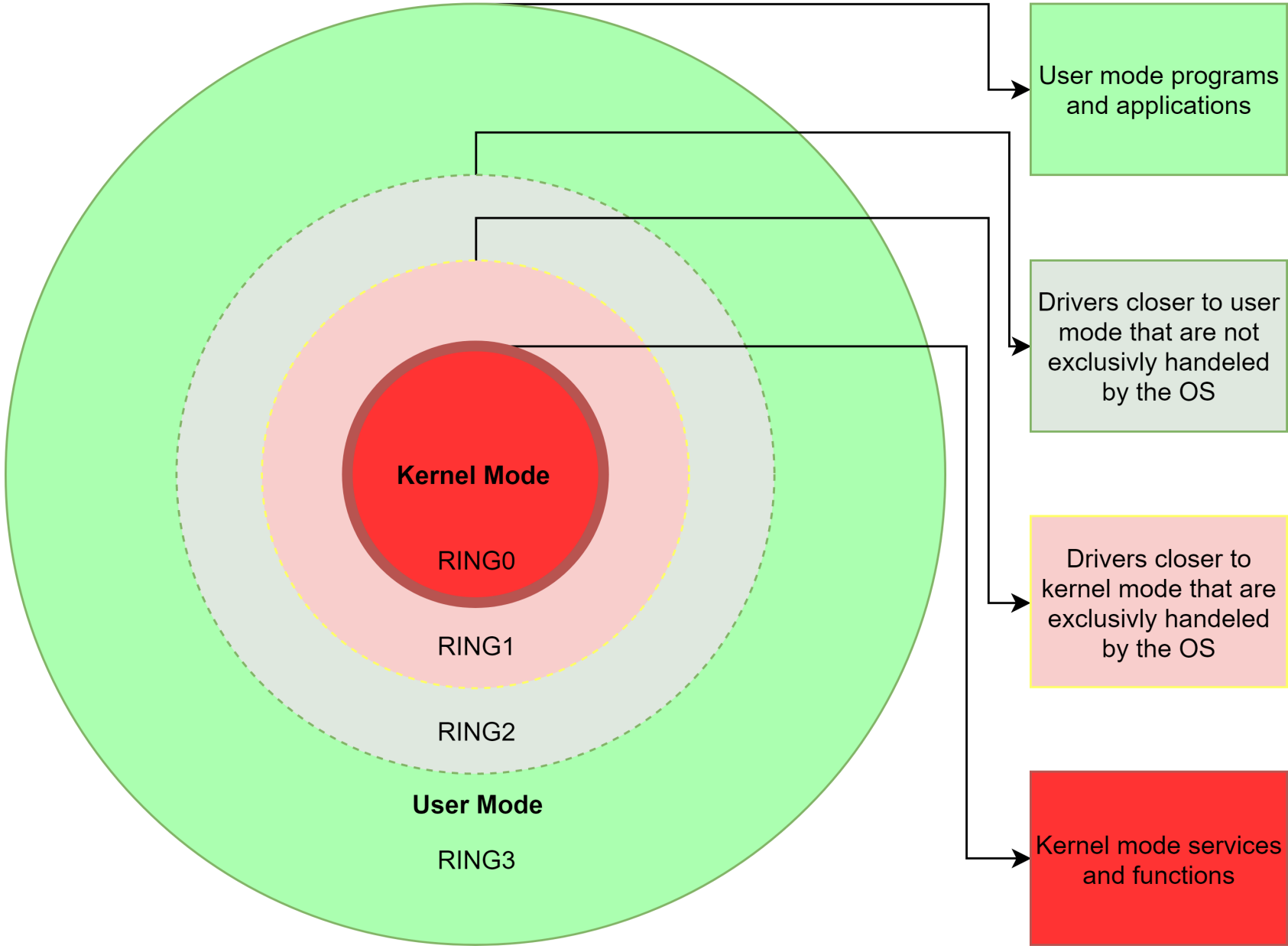
Those drivers do not get any privileges, but they will get some restricted access to kernel mode when needed. This is mostly managed by the OS.



KERNEL VS USER

- **RING3**

Includes all other types of programs and applications that run on User mode. These programs must directly communicate with API if access to Hardware was needed.

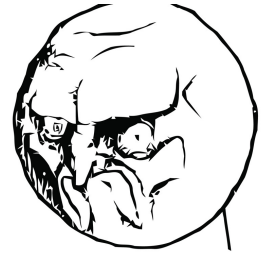


KERNEL VS USER

- Kernel:
  - https://www.youtube.com/watch?v=mycVSMyShk8&t=500s

KERNEL VS USER

- **But again**, why do we need all this mess, I mean, is it not easier if we can just run things on computers without having to worry about stuff….



NO.

WHY WE NEED THOSE MODES

- **Kernel and User modes provide features to the system**
  - **Security:**
    - Create layers of protection around data, user data, memory, and internal system operations
    - Preventing bad programs from interfering with system performance and OS
    - Provide a recovery protocol in case of process or program scheduling failure.
    - Different parts of the system run separately, allowing modification to be done when necessary without intervening with other parts.
  - **Privileges**:
    - We need clarify which programs can do what.
    - What resources can a program use.
    - Control which processes can have more authority on resources.
      - Memory
      - Registers
      - I/O devices
      - CPU time and internal system functions

WHY WE NEED THOSE MODES

- **Kernel and User modes provide features to the system**
  - **Control of execution flow:**
    - Control how execution is handled and how programs are scheduled.
    - Kernel mode programs **<u>can</u>** be of higher importance
      - Thus, they must run on higher priority.
    - Giving more priority to **system calls** or **syscalls**
      - When a user mode program requires access to HW, it issues a syscall.
      - A user mode program with a syscall must be handled before other user mode programs without a syscall

SWITCHING

- **Switching in between**
  - Ok, we know what is kernel and user modes and we know why we need them, but how does a program go from user mode to kernel mode and the other way around.

SWITCHING

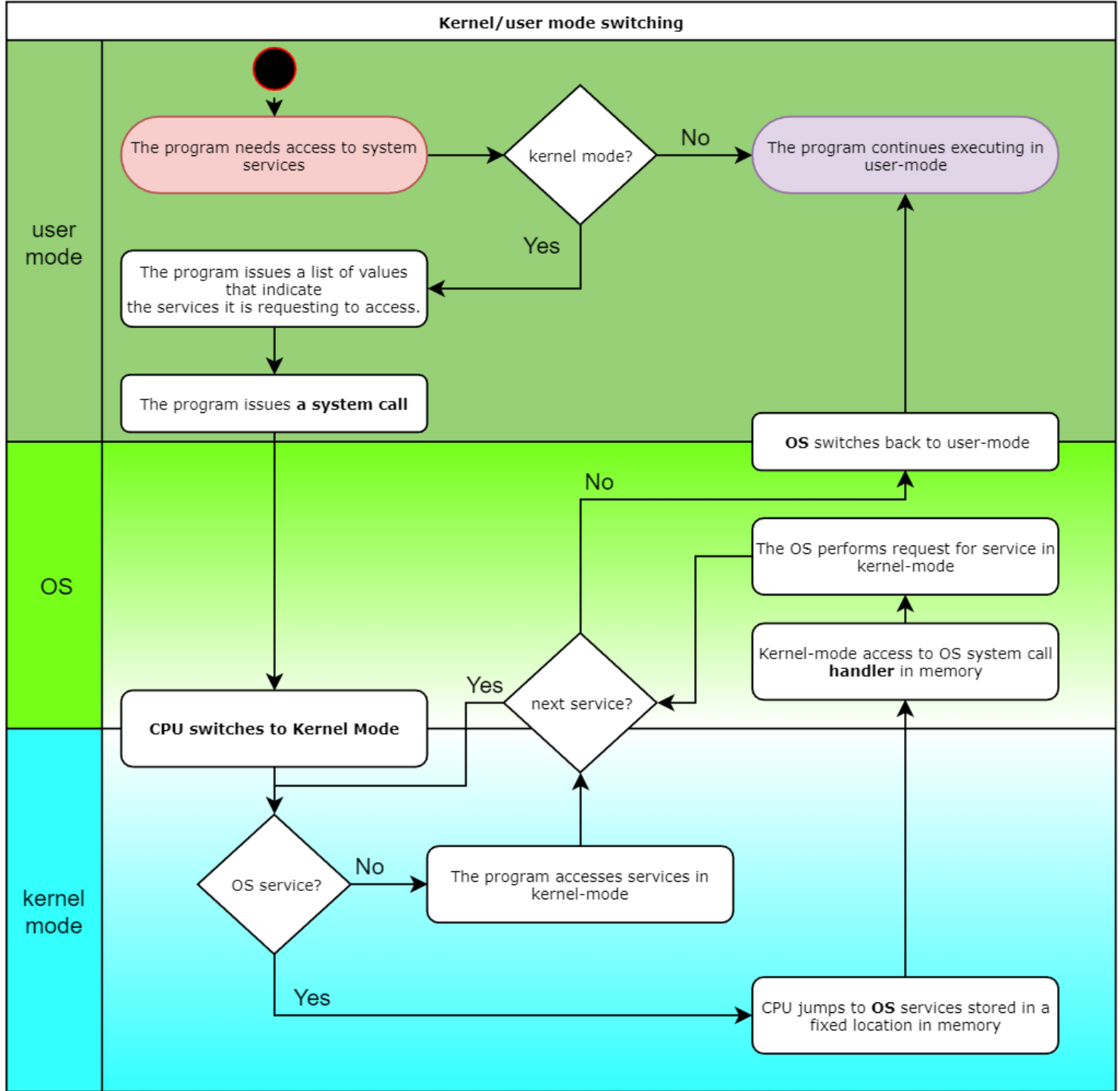
- **Switching in between**
  - Ok, we know what is kernel and user modes and we know why we need them, but how does a program go from user mode to kernel mode and the other way around.
  - Case: access to keyboard by **notepad**
    - Normally, **the CPU runs a program like notepad in User mode**, but it also switches to kernel mode when needed.
    - When a notepad needs access to the keyboard, it reaches out to the **OS API**.
      - It sends a **trap instruction** to the OS.
        - A trap is a special function instruction that requests access to an OS service.
        - Also known as a **syscall** instruction in MIPS ISA.
      - When the OS received the **trap**, it signals the CPU to switch to Kernel Mode to run the OS service that deals with keyboard accesses – **the trap handler**.
      - Once the CPU is in kernel mode, it jumps directly to a fixed location in memory.
        - This location is part of the OS that can only be accessed in kernel mode.
        - The location is **the trap handler** program of the OS.

SWITCHING

- **Switching in between**
  - The trap handler starts by jumping to **the location in memory** where the keyboard device driver is – again, only accessible in kernel mode.
  - Now that we are where the driver is, the **OS** (using the driver program) **accesses the keyboard on-behalf of the notepad program**.
    - It reads the input of the keyboard and sends it to the CPU (still in kernel mode) to be handled.
    - The input from the keyboard is then processed.
  - Once the input from the keyboard is processed, the CPU signals the OS that the trap sent by the notepad program is fulfilled.
  - The OS replies to the notepad that the trap is handled.
    - Then it switches the CPU back to User mode.
  - The notepad program continues executing in user mode.
    - In reality, the notepad itself (program instructions) are never at any case are executed in kernel mode.
    - To the note pad, the trap instruction is a **magical instruction**!
      - It appears that the whole system call was done in one instruction.

SWITCHING

- **Switching in between**



MORE IS MORE

- **This not about kernel mode and user mode:**
  - There is no single kernel mode!
    - There are a variety of modes that "kernel mode" refers to.
      - Depend on the program permissions, system functions, and OS services.
    - Same for user mode.
  - Programs other than the OS rarely run in kernel mode
    - Some exceptions like anti virus programs, Device control BIOSes (Nvidia, Intel, etc.)
      - That is why those require a direct access before the OS boots.
      - Otherwise, the OS will block them from accessing kernel mode.
  - Almost no program (be it OS service or BIOS service) can run on full kernel mode.
    - No program can control everything!
    - So even in kernel mode there are restrictions in some cases.