Operating Systems

Lecture 3

Computer Security DD2395

Roberto Guanciale
robertog@kth.se

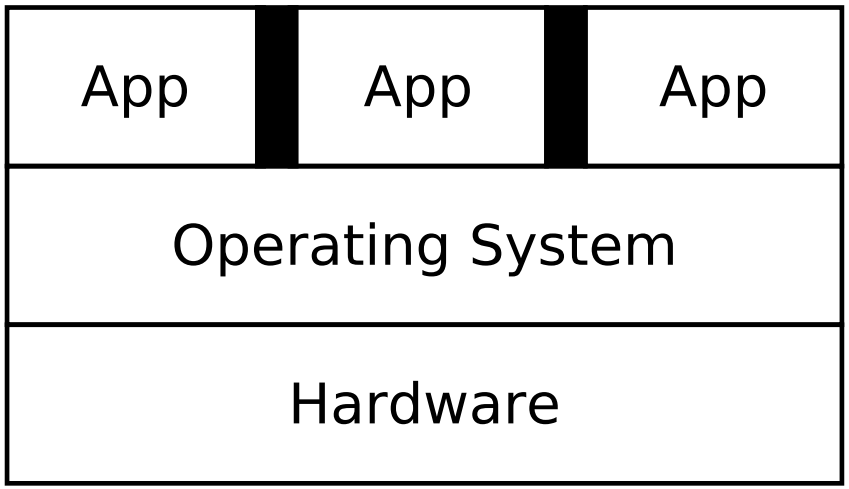
2017-11-03

Roberto Guanciale

- Assistant professor Computer Security
- Research interest in Formal Verification
  - Hack-proof Operating Systems
- Enthusiast software developer
- robertog@kth.se (use [DD2395] in mail subject)
- Office: Building D, level 5, room 4529
- You are welcomed (no booking required, no fee)
  - Tuesday 13:15-14:30

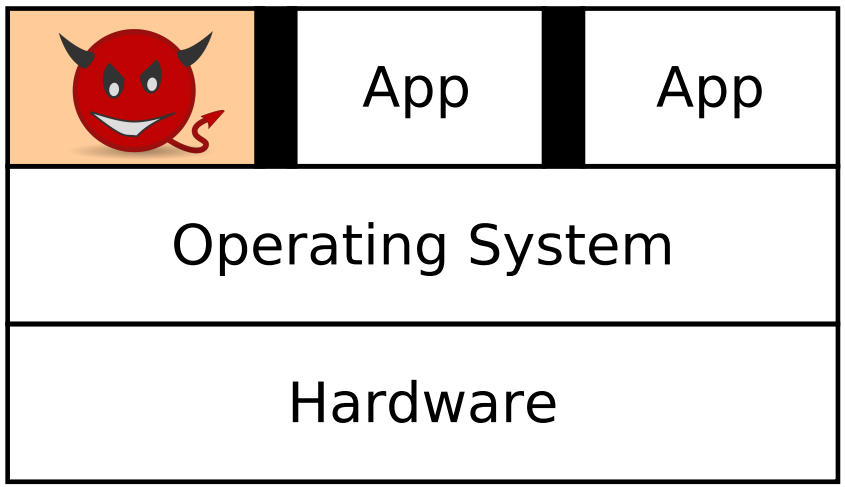
Operating Systems

- An OS is a program that acts an intermediary between applications and computer hardware
- Makes application software portable and versatile.
- Allows sharing of hardware and software resources.
- Provides isolation, security and protection among applications.



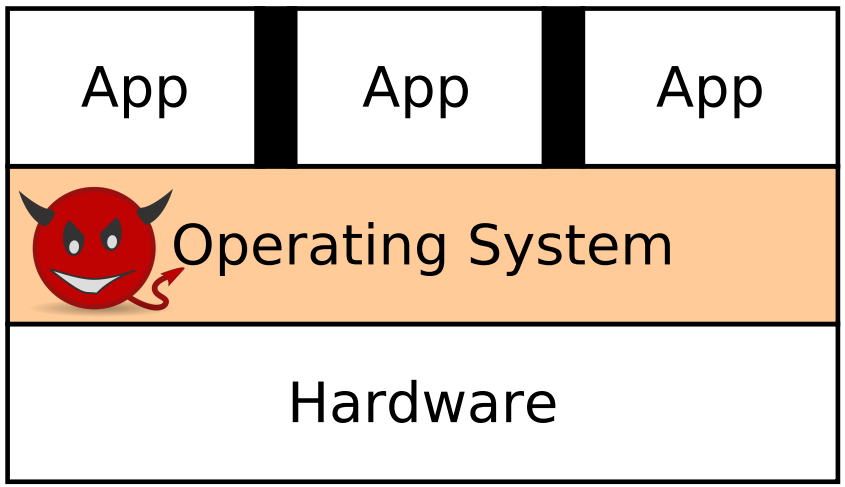
Operating Systems

- An OS is a program that acts an intermediary between applications and computer hardware
- Makes application software portable and versatile.
- Allows sharing of hardware and software resources.
- Provides isolation, security and protection among applications.



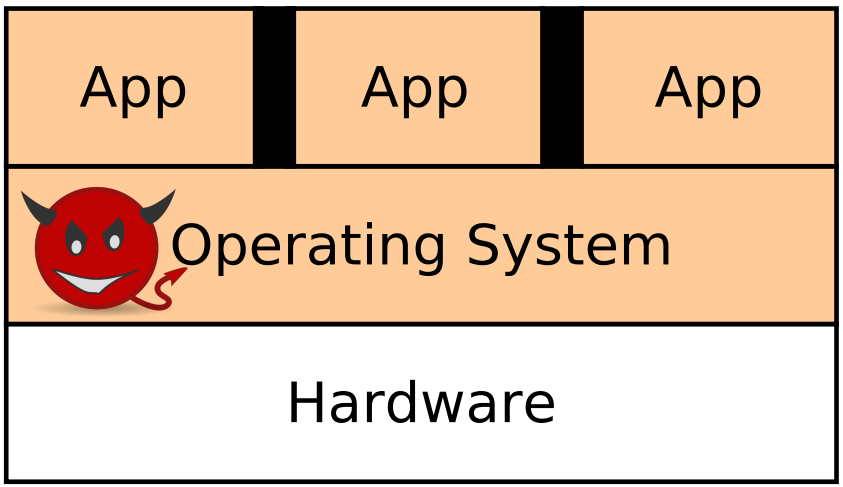
Operating Systems

- An OS is a program that acts an intermediary between applications and computer hardware
- Makes application software portable and versatile.
- Allows sharing of hardware and software resources.
- Provides isolation, security and protection among applications.



Operating Systems

- An OS is a program that acts an intermediary between applications and computer hardware
- Makes application software portable and versatile.
- Allows sharing of hardware and software resources.
- Provides isolation, security and protection among applications.



Main questions

- How can we run Chrome on a modern architecture?
- How can we isolate Chrome from LibreOffice?
- How can we run multiple instances of Chrome?
- How can we interact with peripherals?

Example of OSes

- Windows/Linux/OS X ($\sim$ 50 milions LOC)
- Android/iOS ($\sim$ 50 milions LOC)

Example of OSes

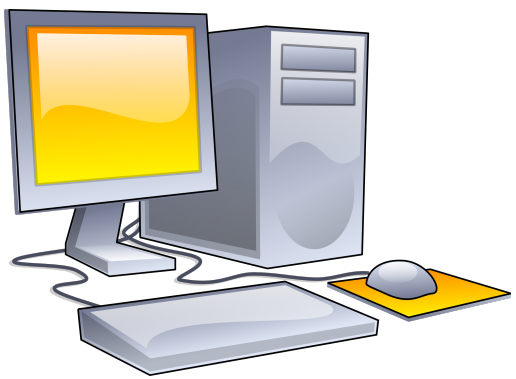
- Windows/Linux/OS X ($\sim$ 50 milions LOC)
- Android/iOS ($\sim$ 50 milions LOC)
- S40 ($\sim$ 1 milions LOC)

Example of OSes

- Windows/Linux/OS X ($\sim$ 50 milions LOC)
- Android/iOS ($\sim$ 50 milions LOC)
- S40 ($\sim$ 1 milions LOC)
- Free RTOS ($\sim$ 10 000 LOC)

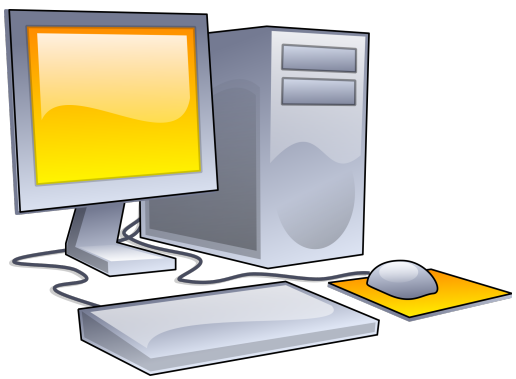
Example of OSes

- Windows/Linux/OS X ($\sim$ 50 milions LOC)
- Android/iOS ($\sim$ 50 milions LOC)
- S40 ($\sim$ 1 milions LOC)
- Free RTOS ($\sim$ 10 000 LOC)



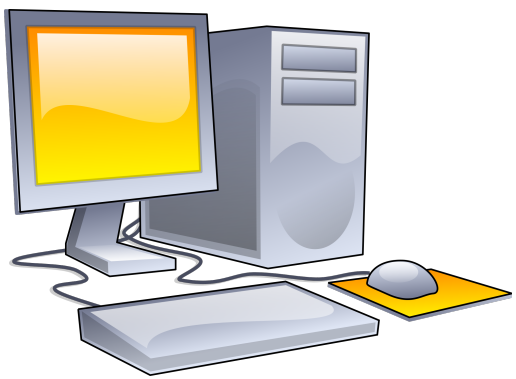
Example of OSes

- Windows/Linux/OS X ($\sim$ 50 milions LOC)
- Android/iOS ($\sim$ 50 milions LOC)
- S40 ($\sim$ 1 milions LOC)
- Free RTOS ($\sim$ 10 000 LOC)



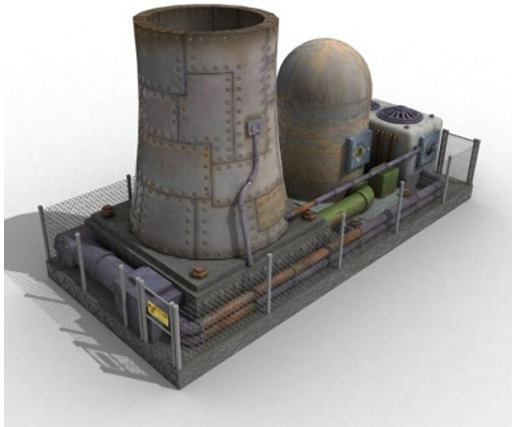
Example of OSes

- Windows/Linux/OS X ($\sim$ 50 milions LOC)
- Android/iOS ($\sim$ 50 milions LOC)
- S40 ($\sim$ 1 milions LOC)
- Free RTOS ($\sim$ 10 000 LOC)



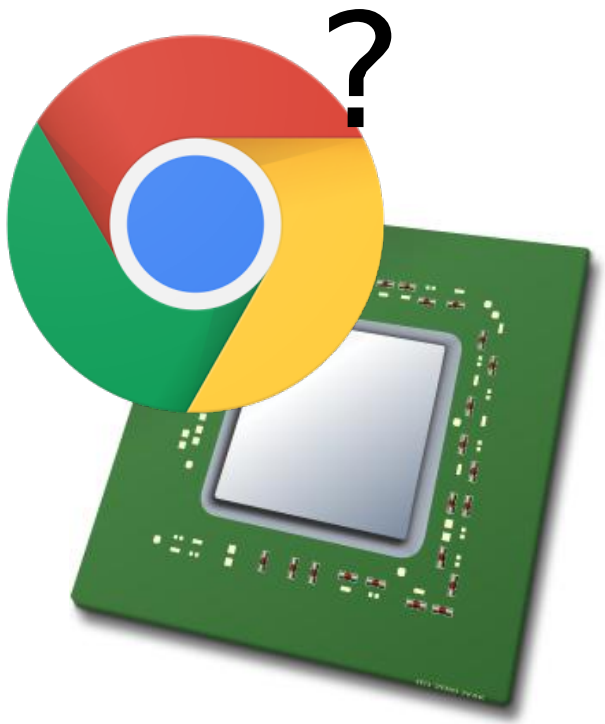
Example of OSes

- Windows/Linux/OS X ($\sim$ 50 milions LOC)
- Android/iOS ($\sim$ 50 milions LOC)
- S40 ($\sim$ 1 milions LOC)
- Free RTOS ($\sim$ 10 000 LOC)



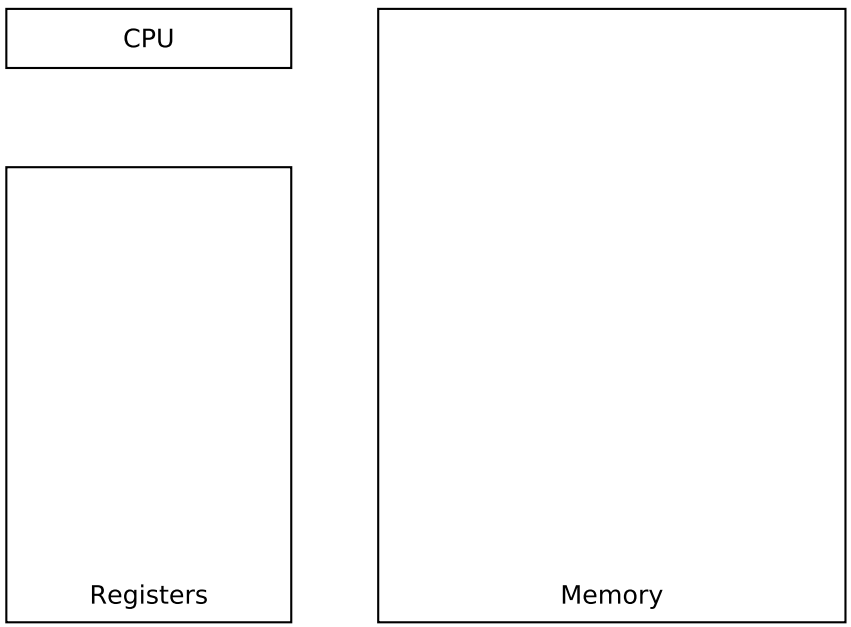
OS by example

How can we run Chrome on a modern CPU?



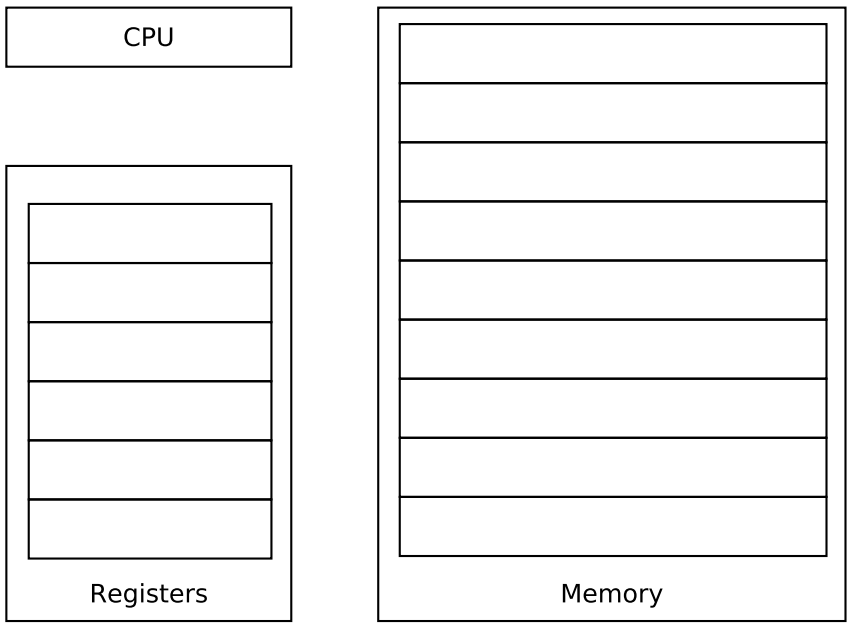
HW architecture

- Common architectures x86, x64, ARM, MIPS, PowerPC



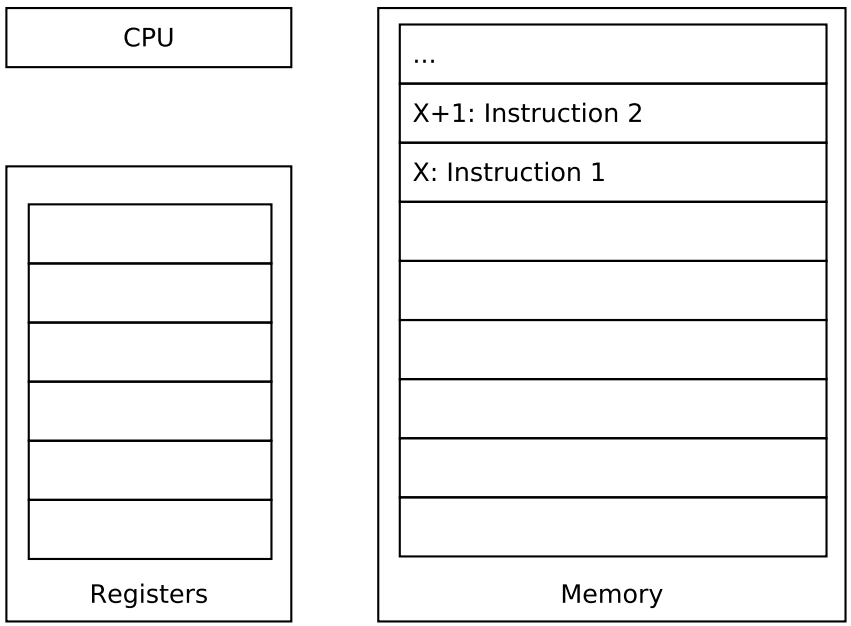
HW architecture

- Common architectures x86, x64, ARM, MIPS, PowerPC



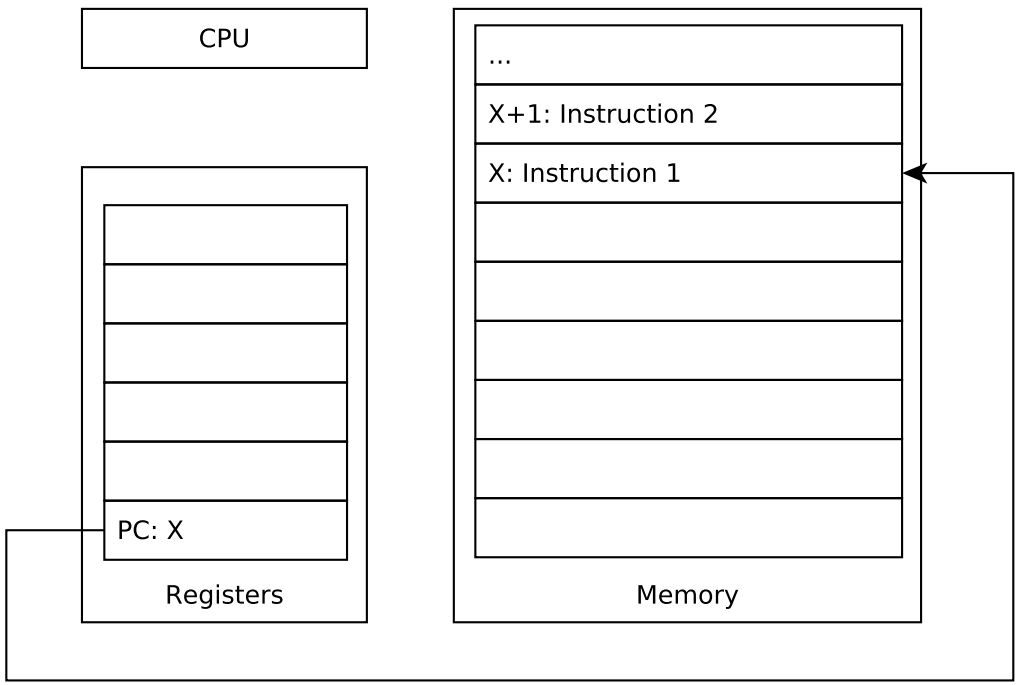
HW architecture

- Common architectures x86, x64, ARM, MIPS, PowerPC
- Program stored in memory



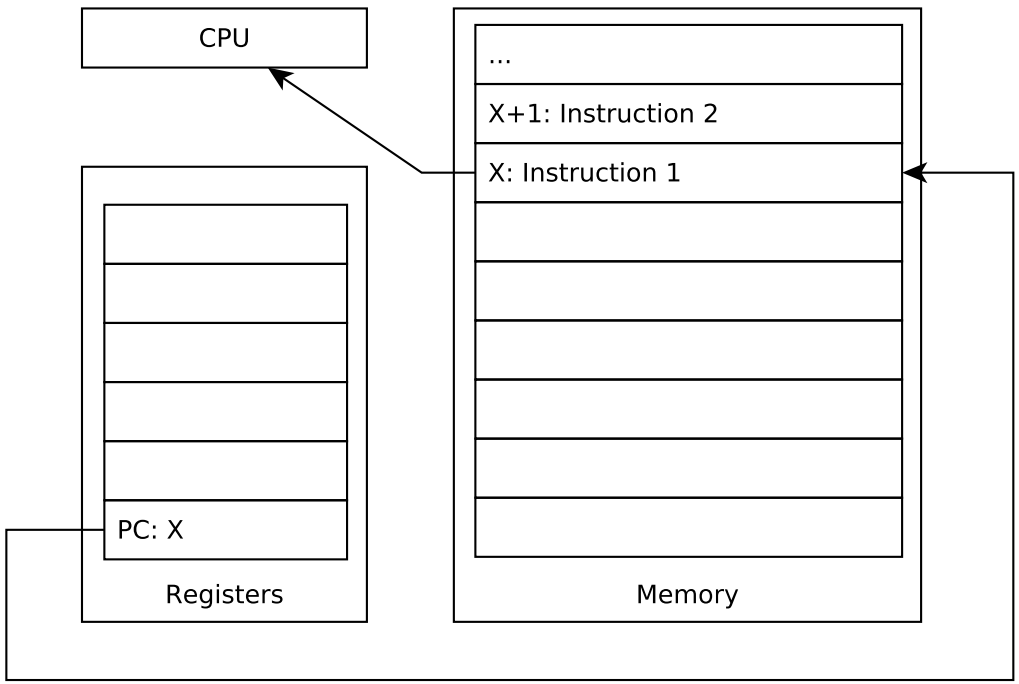
HW architecture

- Common architectures x86, x64, ARM, MIPS, PowerPC
- Program stored in memory
- Program Counter (PC): address of the current instruction



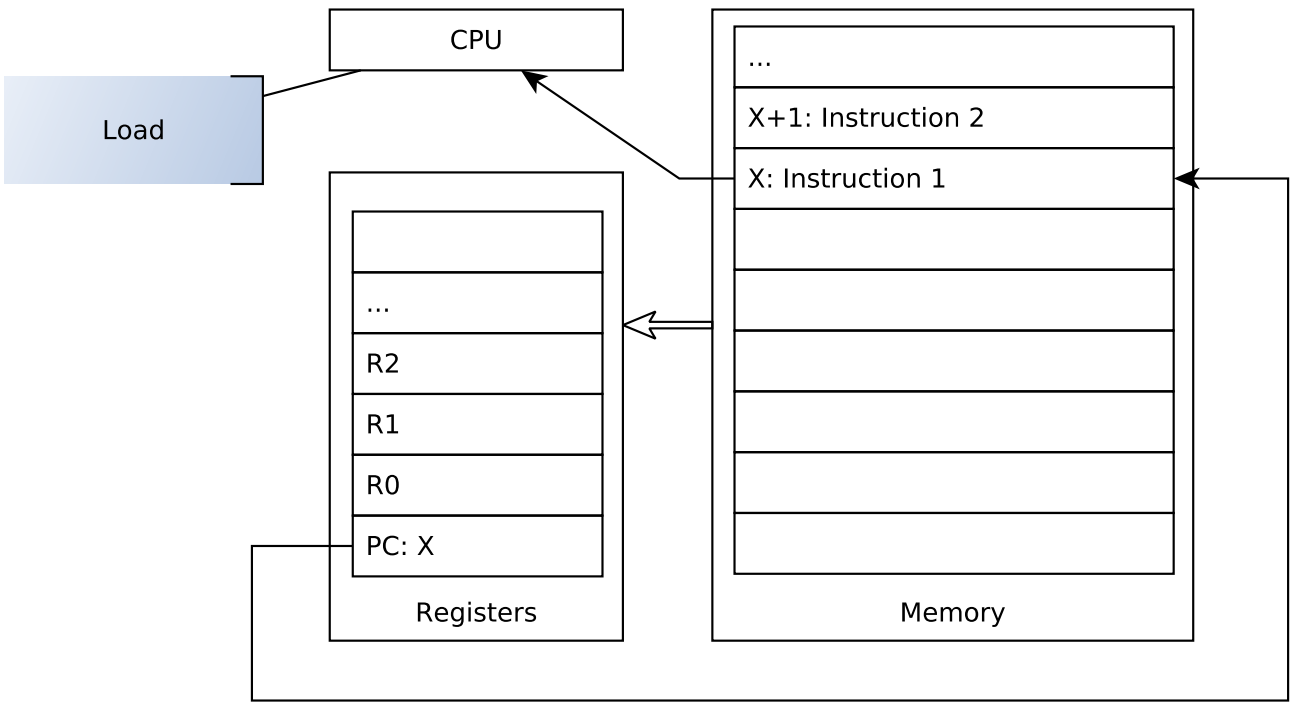
HW architecture

- Common architectures x86, x64, ARM, MIPS, PowerPC
- Program stored in memory
- Program Counter (PC): address of the current instruction



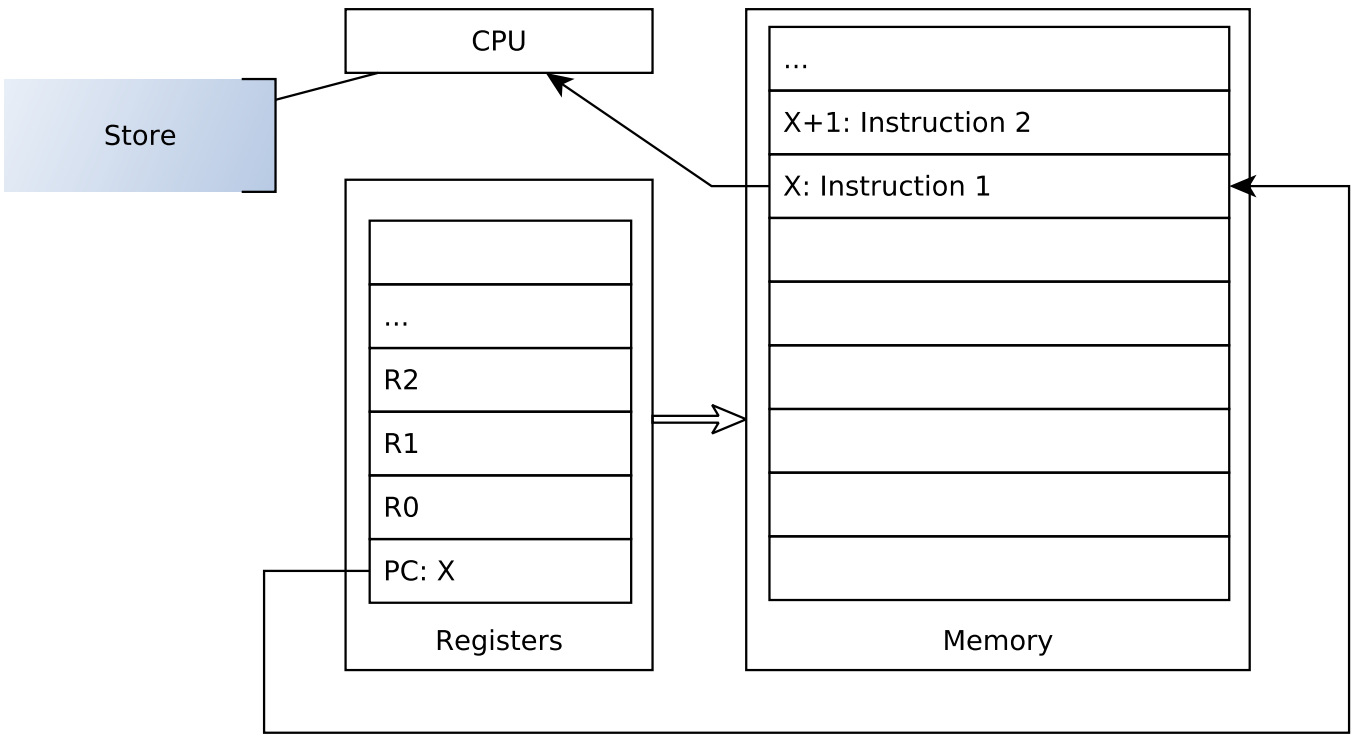
HW architecture

- Common architectures x86, x64, ARM, MIPS, PowerPC
- Program stored in memory
- Program Counter (PC): address of the current instruction



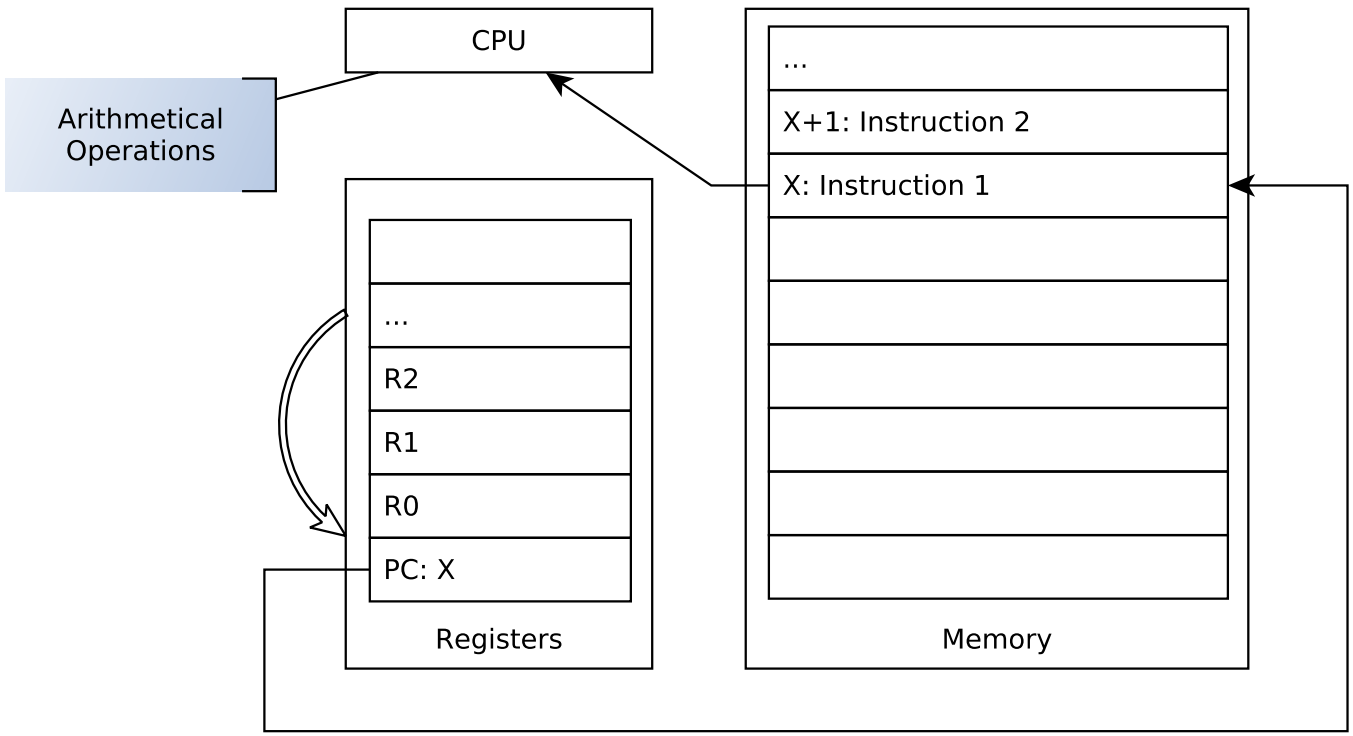
HW architecture

- Common architectures x86, x64, ARM, MIPS, PowerPC
- Program stored in memory
- Program Counter (PC): address of the current instruction



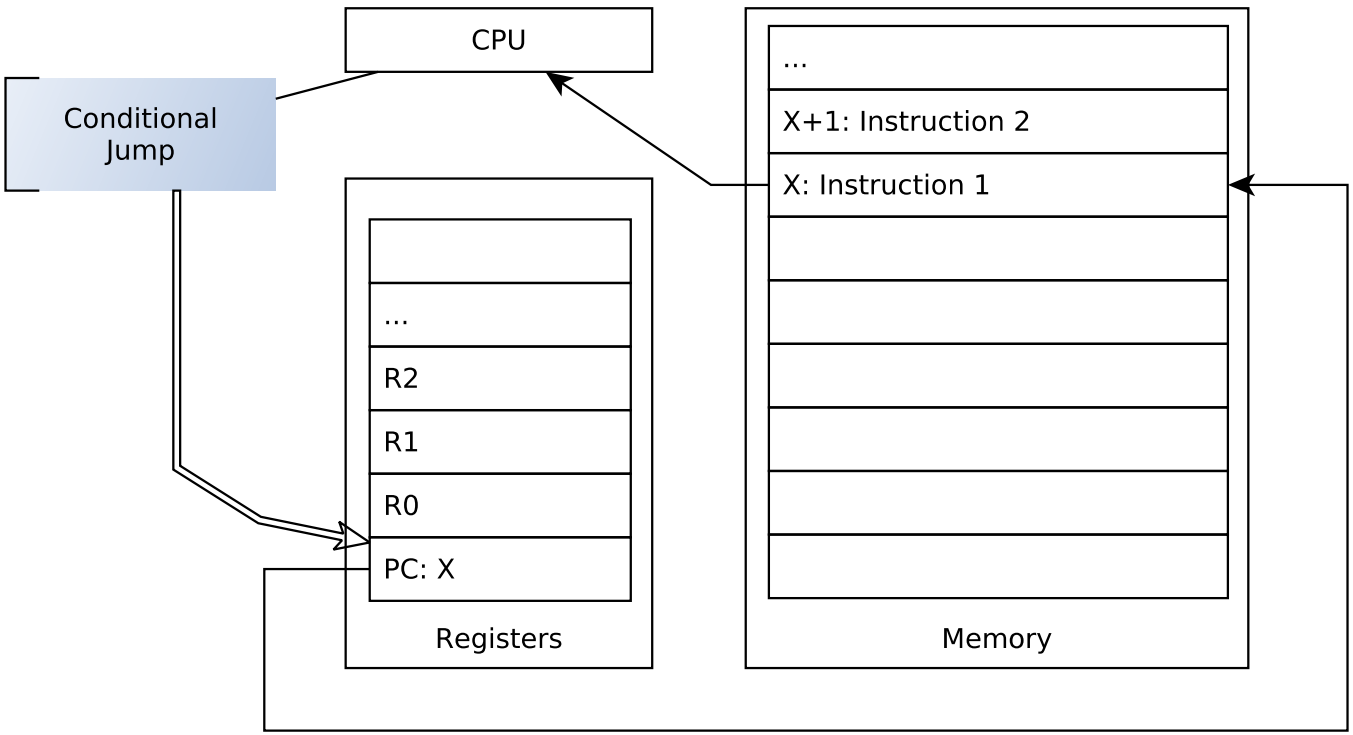
HW architecture

- Common architectures x86, x64, ARM, MIPS, PowerPC
- Program stored in memory
- Program Counter (PC): address of the current instruction



HW architecture

- Common architectures x86, x64, ARM, MIPS, PowerPC
- Program stored in memory
- Program Counter (PC): address of the current instruction



Application memory

```
int x;
int * hello(int y) {
  int * res = malloc(8);
  res[0] = x + 10;
  res[1] = y + x;
  return res;
}
```

Application memory

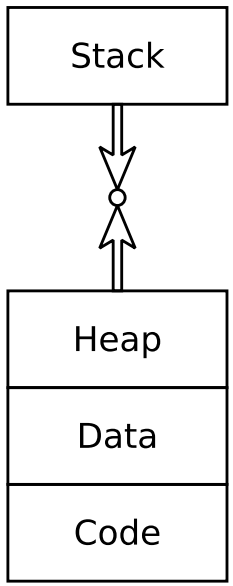
```
int x;
int * hello(int y) {
  int * res = malloc(8);
  res[0] = x + 10;
  res[1] = y + x;
  return res;
}
```

```
class Example {
   static int x;
   int[] hello(int y) {
      int[] res = new int[2];
      res[0] = x + 10;
      res[1] = y + x;
      return res;
   }
```

Application Memory

```
int x;
int * hello(int y) {
  int * res = malloc(8);
  res[0] = x + 10;
  res[1] = y + x;
  return res;
}
```

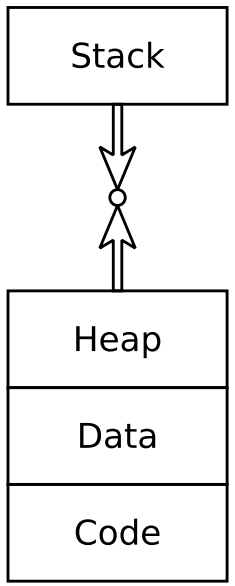
```
class Example {
   static int x;
   int[] hello(int y) {
      int[] res = new int[2];
      res[0] = x + 10;
      res[1] = y + x;
      return res;
   }
```



Application Memory

```
int x;
int * hello(int y) {
  int * res = malloc(8);
  res[0] = x + 10;
  res[1] = y + x;
  return res;
}
```

```
class Example {
   static int x;
   int[] hello(int y) {
      int[] res = new int[2];
      res[0] = x + 10;
      res[1] = y + x;
      return res;
   }
```

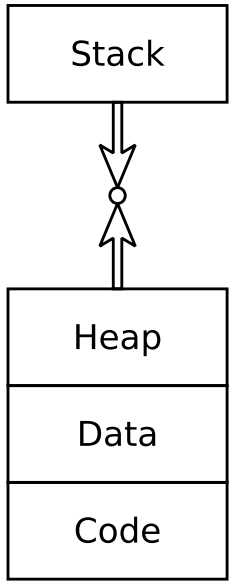


Discuss five minutes with your colleagues
Where each element is allocated?

Application Memory

```
int x;
int * hello(int y) {
  int * res = malloc(8);
  res[0] = x + 10;
  res[1] = y + x;
  return res;
}
```

```
class Example {
   static int x;
   int[] hello(int y) {
      int[] res = new int[2];
      res[0] = x + 10;
      res[1] = y + x;
      return res;
   }
```



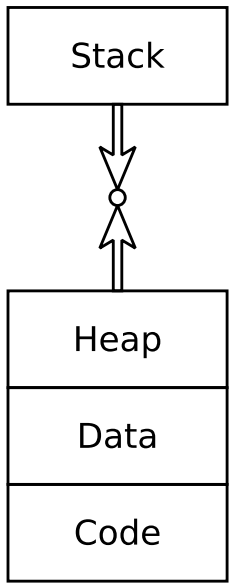
Question 2: Where does malloc allocate memory?

- A - Code
- B - Data
- C - Heap
- D - Stack

Application Memory

```
int x;
int * hello(int y) {
  int * res = malloc(8);
  res[0] = x + 10;
  res[1] = y + x;
  return res;
}
```

```
class Example {
   static int x;
   int[] hello(int y) {
      int[] res = new int[2];
      res[0] = x + 10;
      res[1] = y + x;
      return res;
   }
```



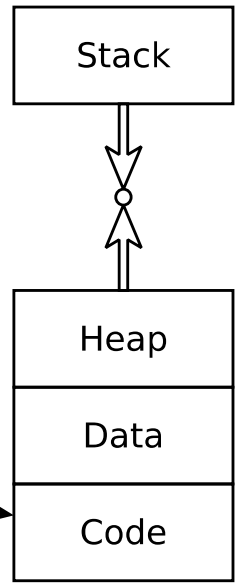
Question 3: Where is y allocated?

- A - Code
- B - Data
- C - Heap
- D - Stack

Application Memory

```
int x;
int * hello(int y) {
   int * res = malloc(8);
   res[0] = x + 10;
   res[1] = y + x;
   return res;
}
```

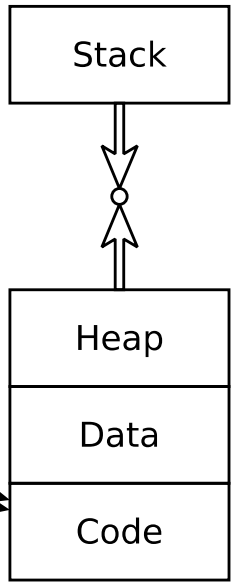
```
class Example {
   static int x;
   int[] hello(int y) {
      int[] res = new int[2];
      res[0] = x + 10;
      res[1] = y + x;
      return res;
   }
```



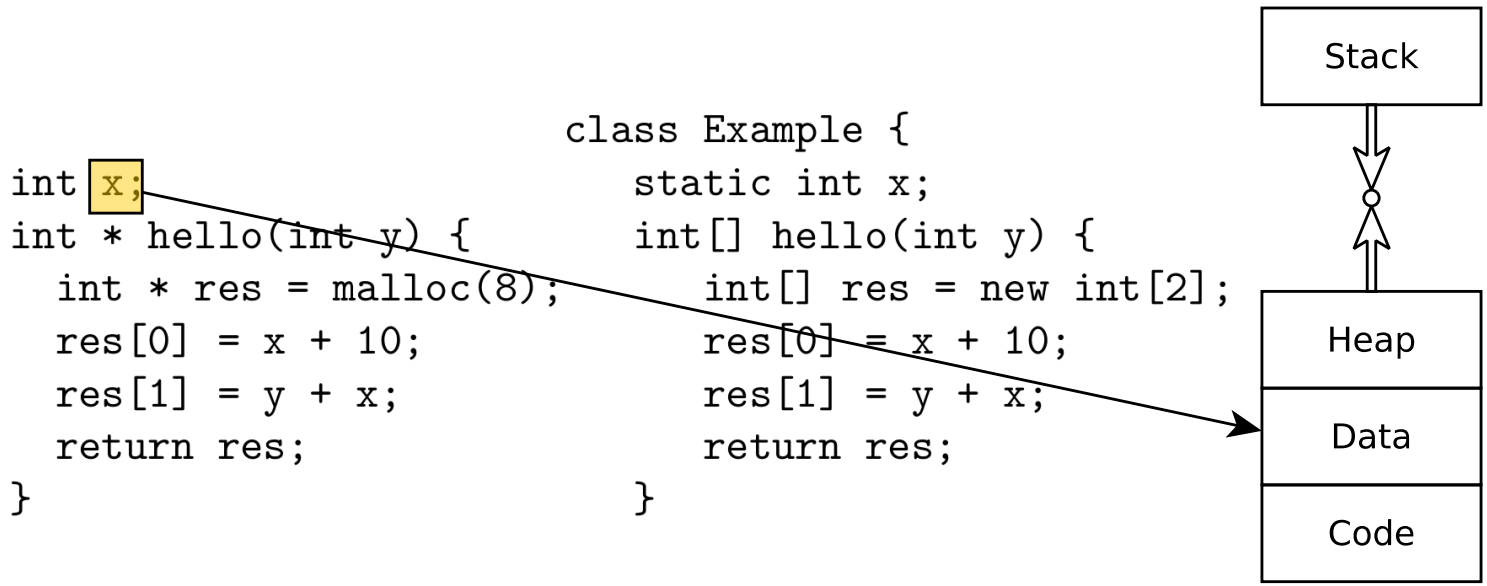
Application Memory

```
int x;
int * hello(int y) {
  int * res = malloc(8);
  res[0] = x + 10;
  res[1] = y + x;
  return res;
}
```

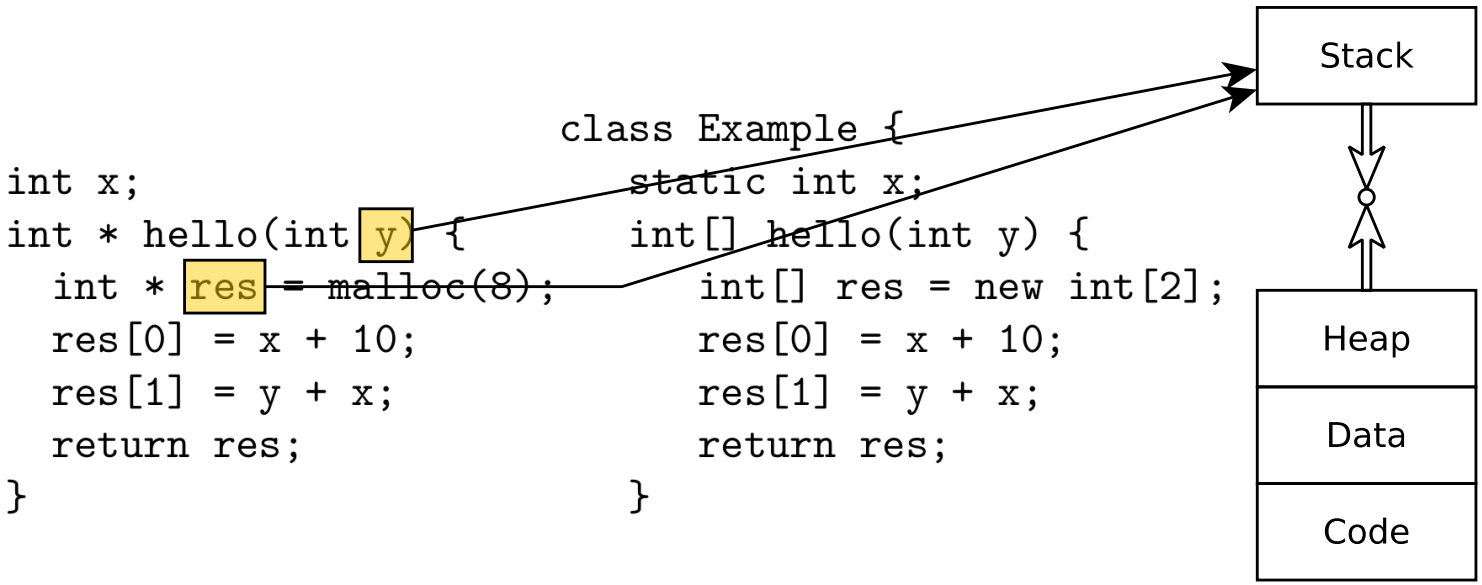
```
class Example {
   static int x;
   int[] hello(int y) {
      int[] res = new int[2];
      res[0] = x + 10;
      res[1] = y + x;
      return res;
   }
```



Application Memory



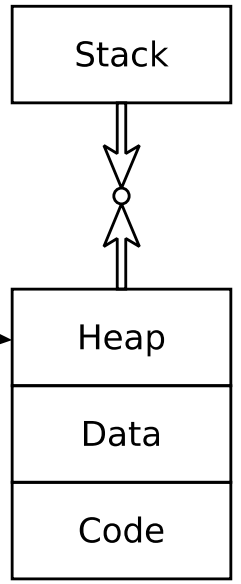
Application Memory



Application Memory

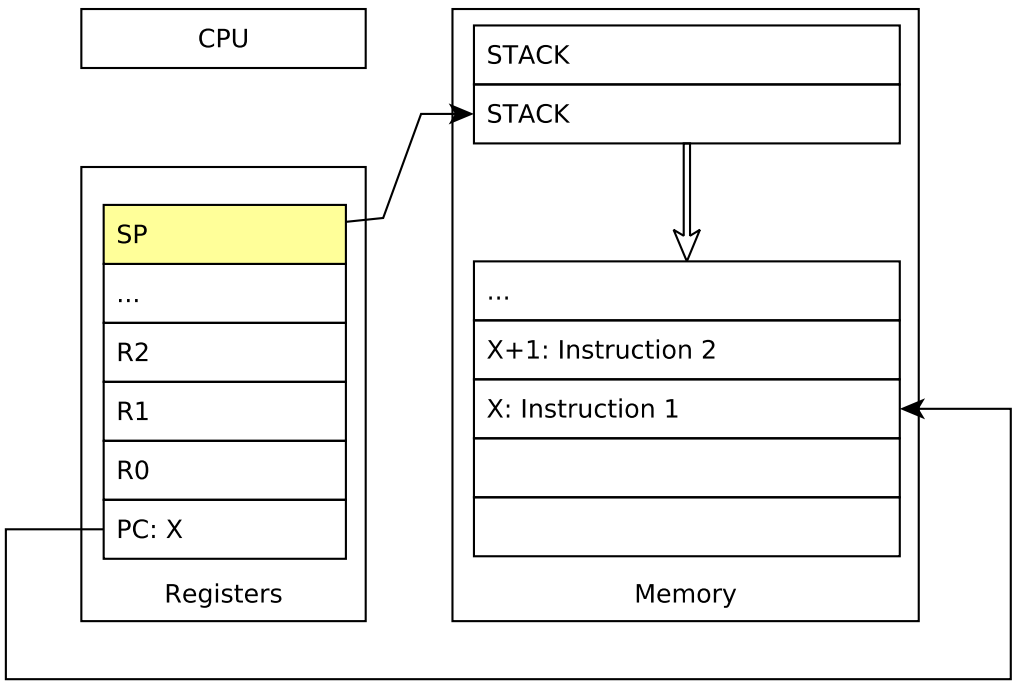
```
int x;
int * hello(int y) {
  int * res = malloc(8);
  res[0] = x + 10;
  res[1] = y + x;
  return res;
}
```

```
class Example {
   static int x;
   int[] hello(int y) {
      int[] res = new int[2];
      res[0] = x + 10;
      res[1] = y + x;
      return res;
   }
```



Application Memory

- Stack Pointer points to the bottom of the stack



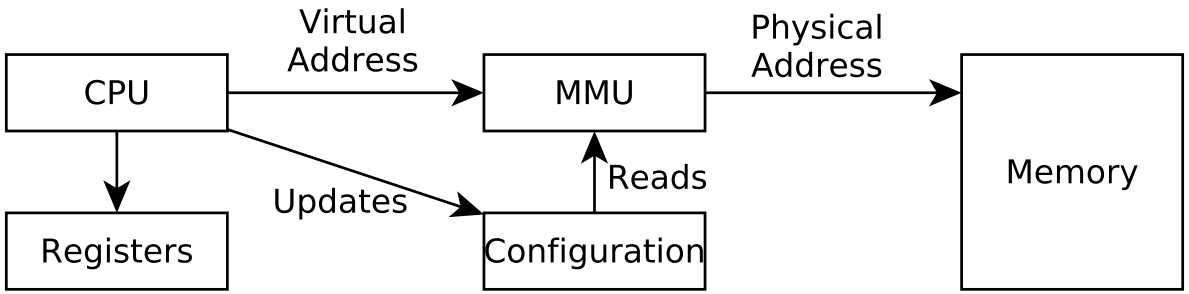
Virtual Memory

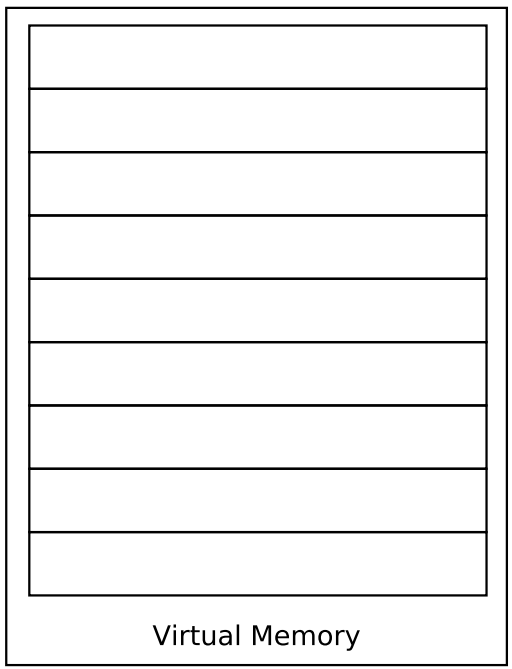
- Modern systems do not use directly physical addresses
  - Run the same software in systems with 1 GB or 2 GB of memory
  - Guarantee that two independent applications do not use the same addresses

Virtual Memory

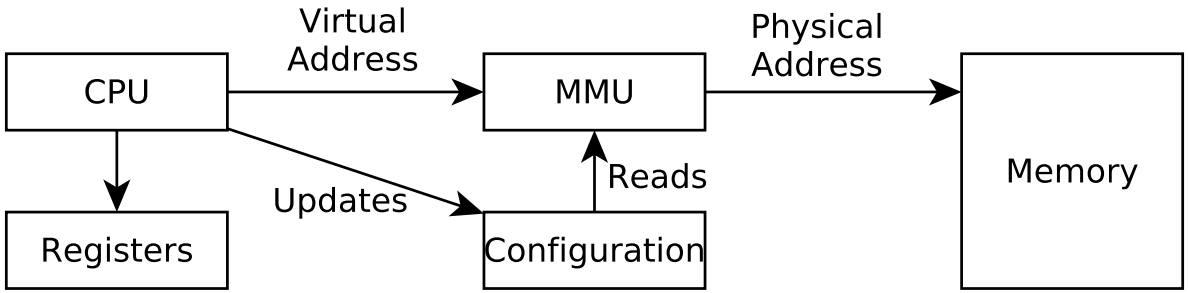
- Modern systems do not use directly physical addresses
  - Run the same software in systems with 1 GB or 2 GB of memory
  - Guarantee that two independent applications do not use the same addresses
- Virtual memory
  - SW uses virtual addresses
  - HW configuration maps virtual addresses to physical addresses
  - HW configuration updatable
  - HW mechanism is called Memory Management Unit

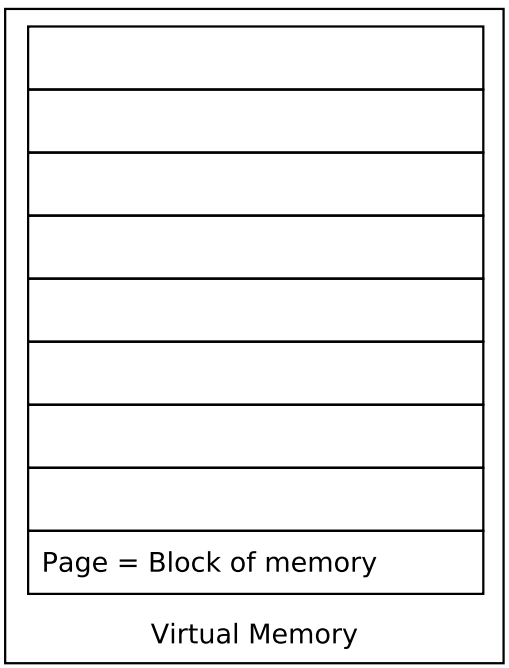
Virtual Memory



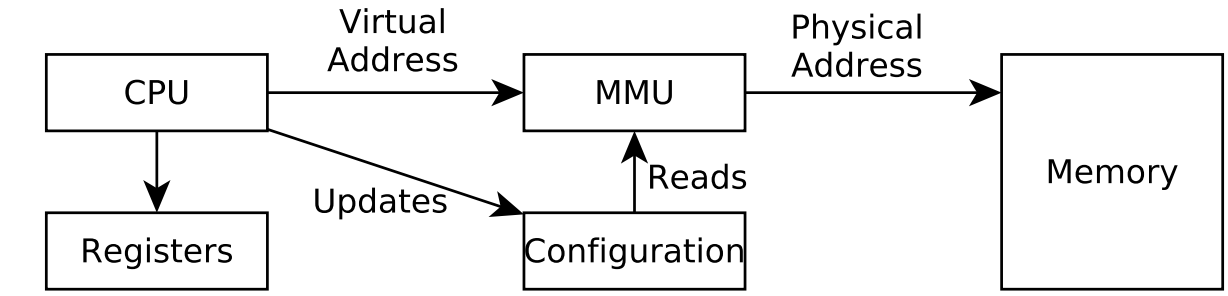


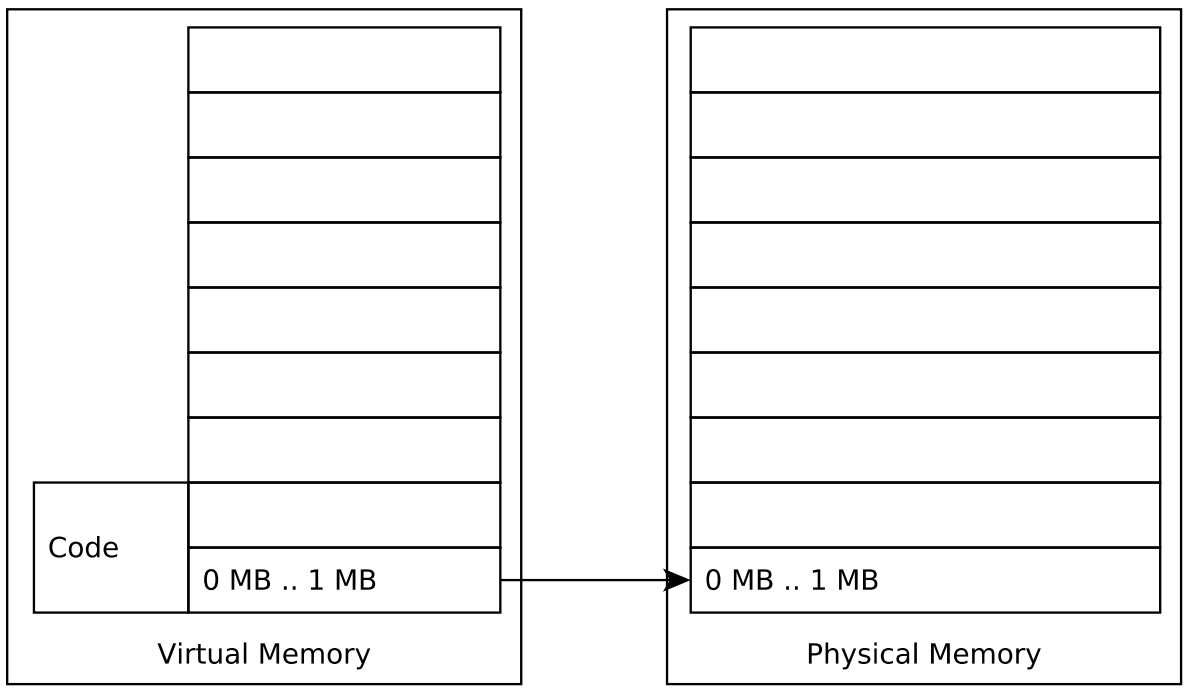
Virtual Memory



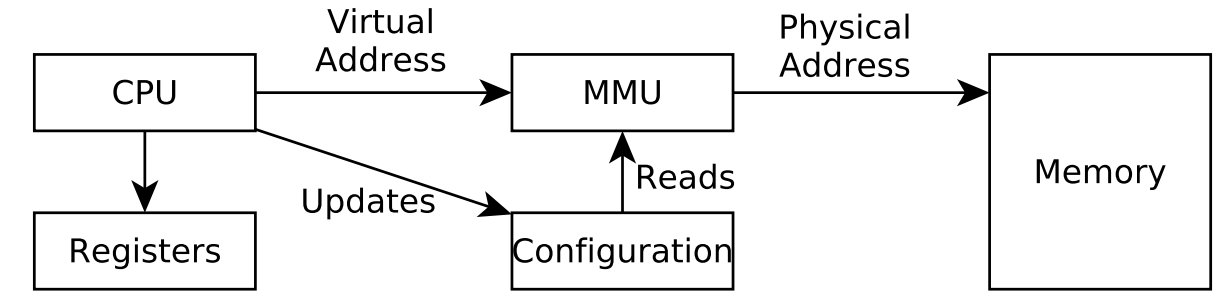


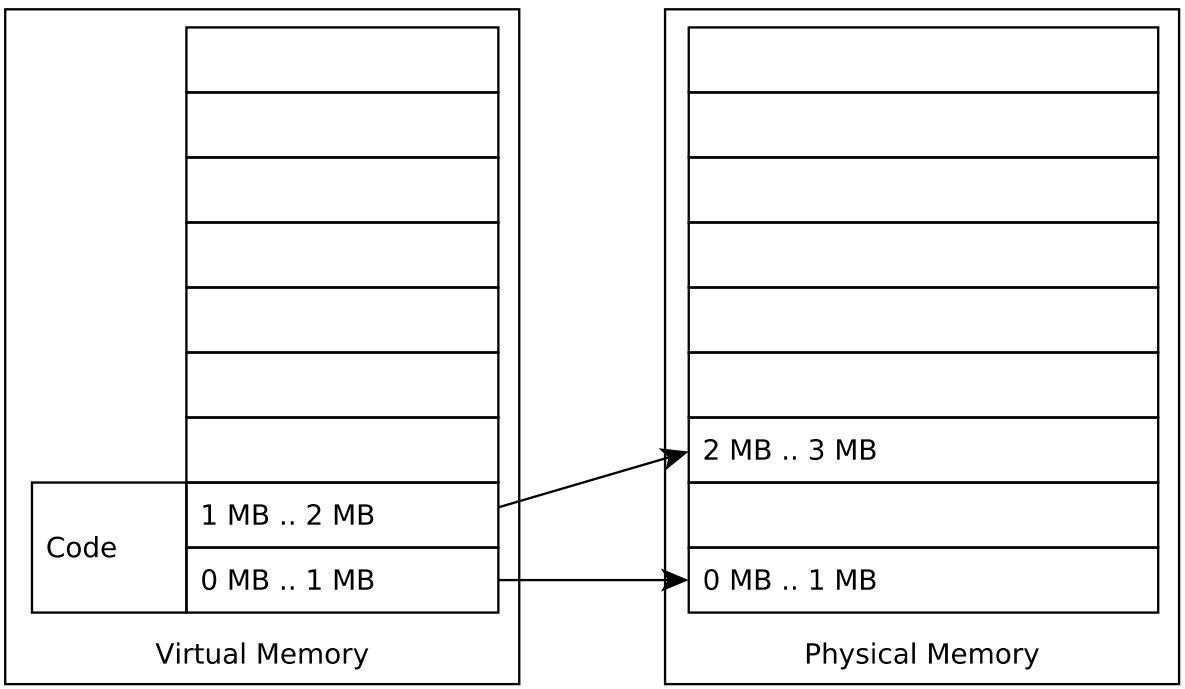
Virtual Memory



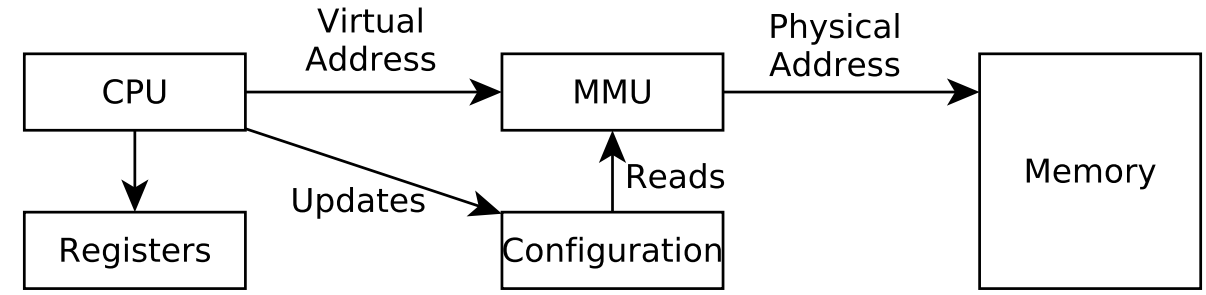


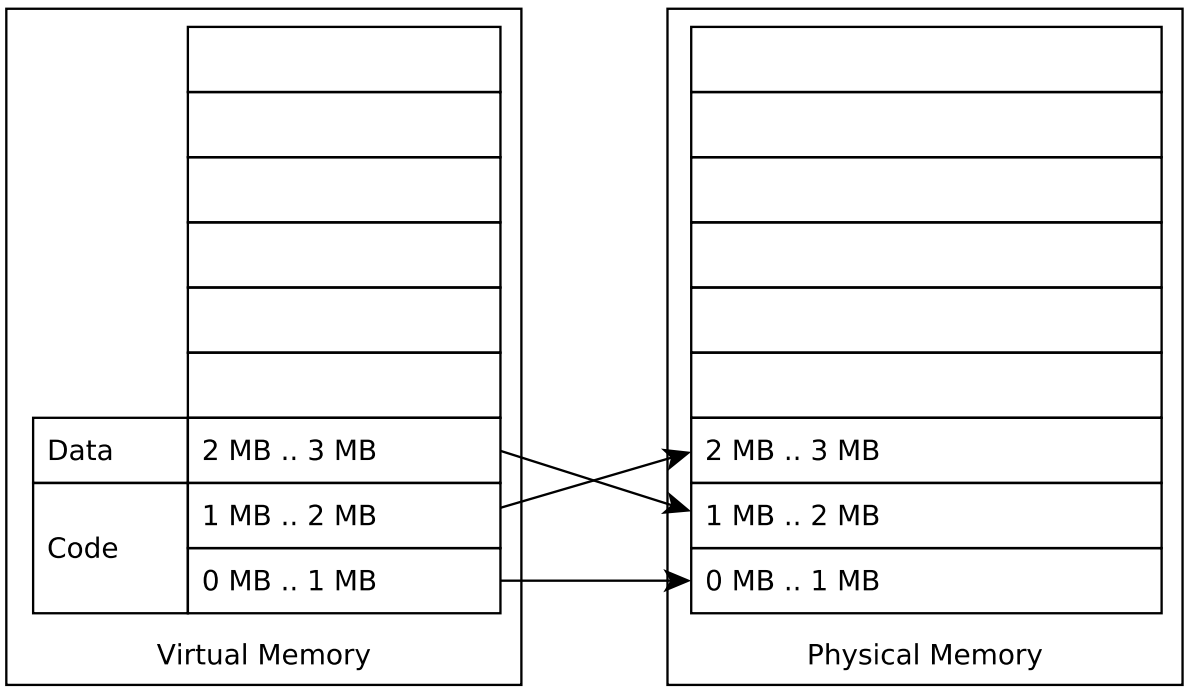
Virtual Memory



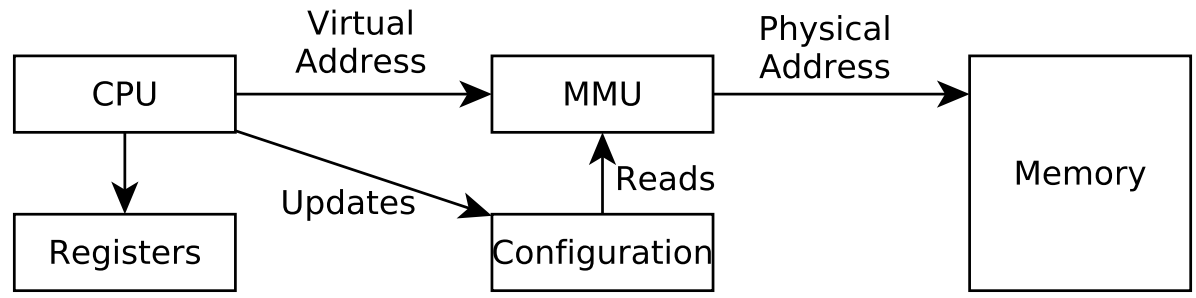


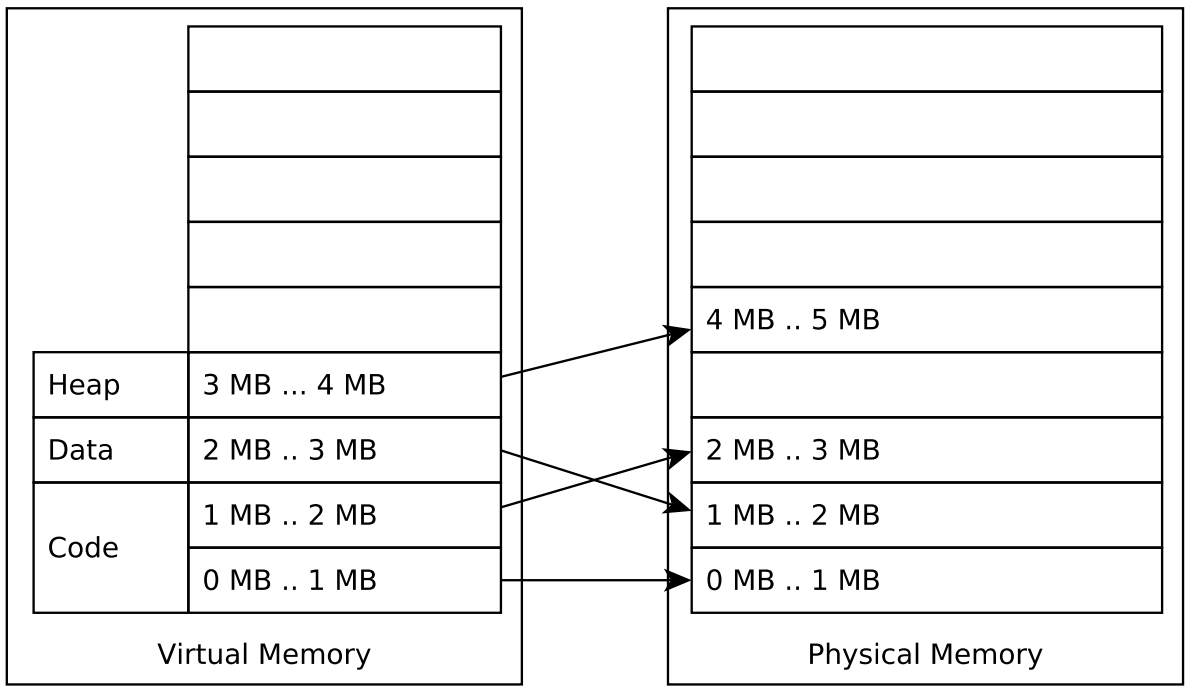
Virtual Memory



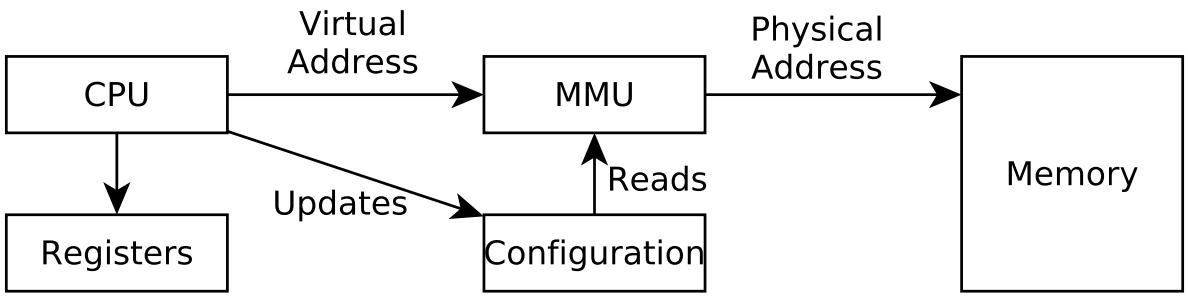


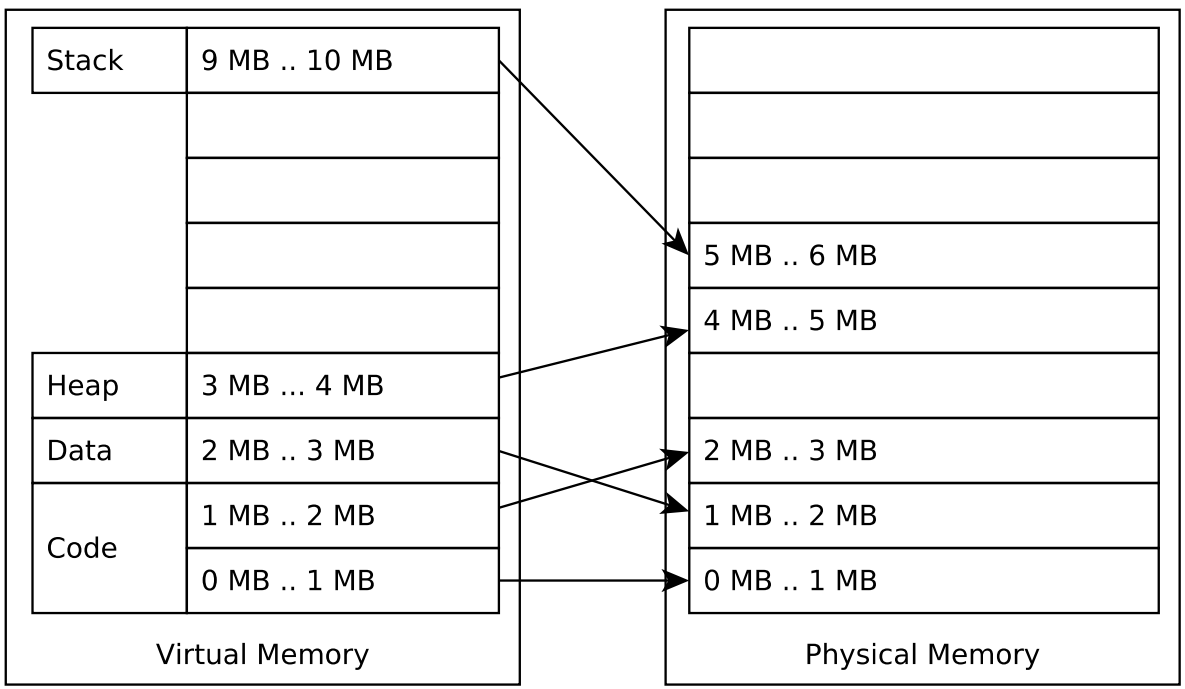
Virtual Memory



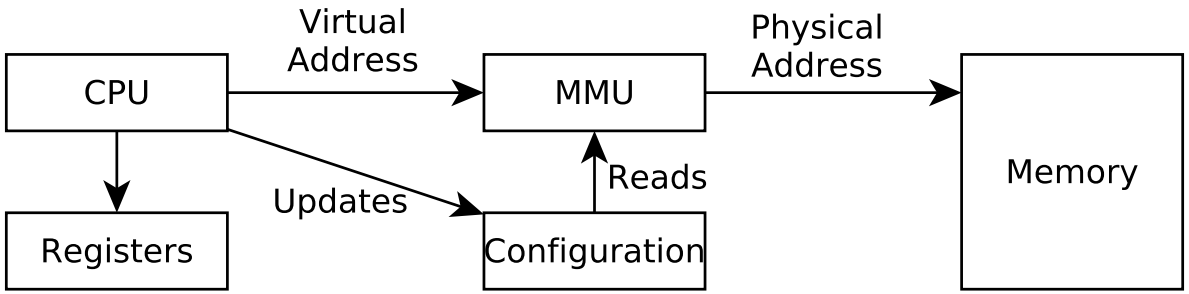


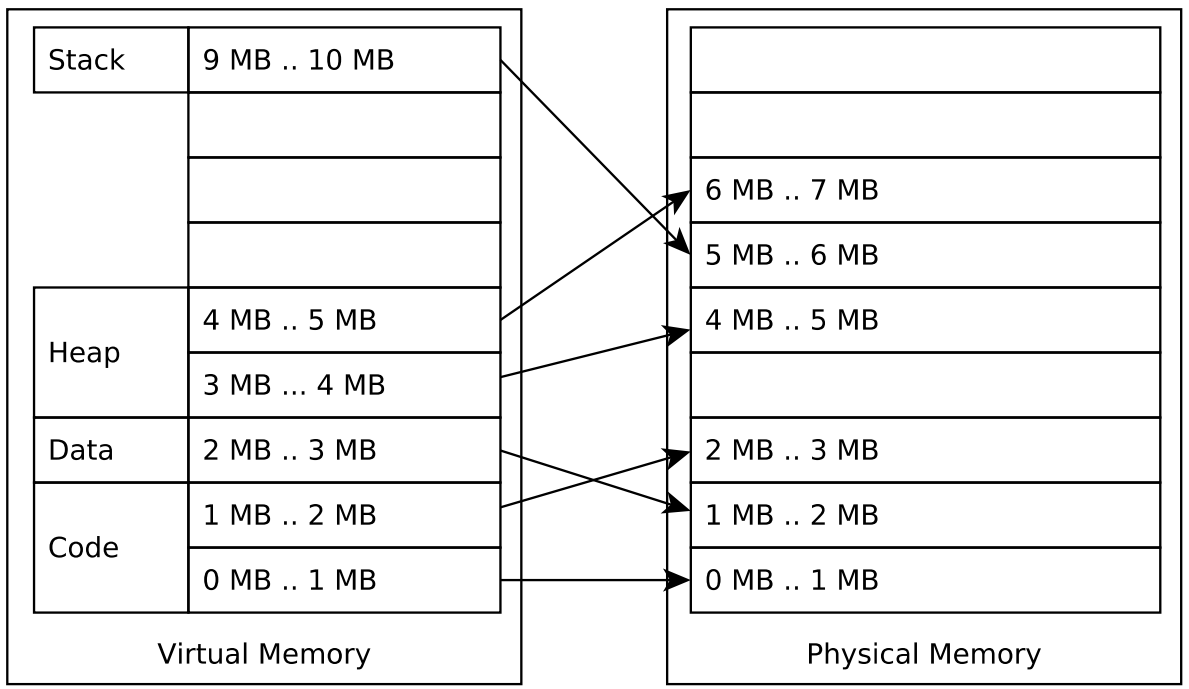
Virtual Memory



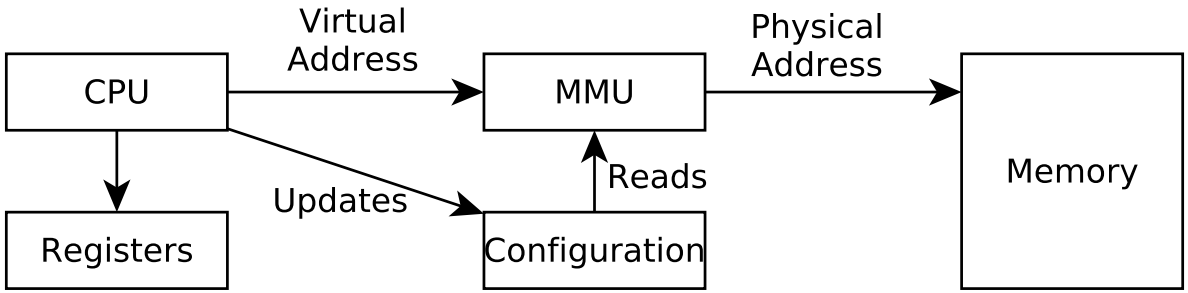


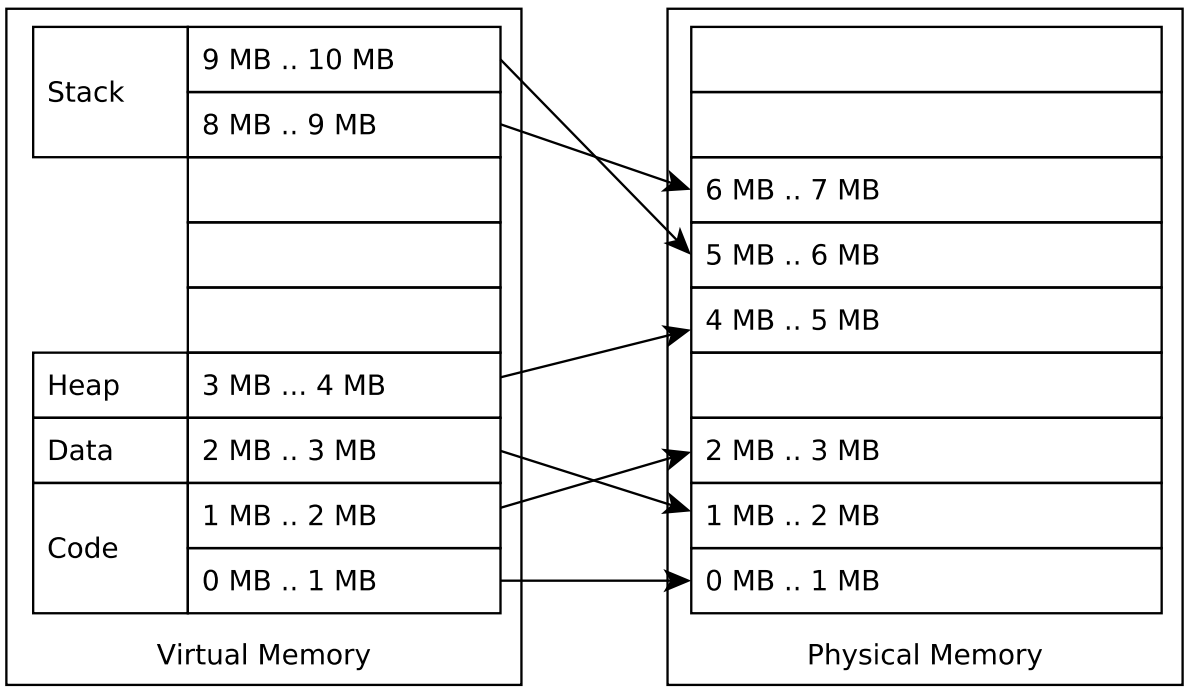
Virtual Memory





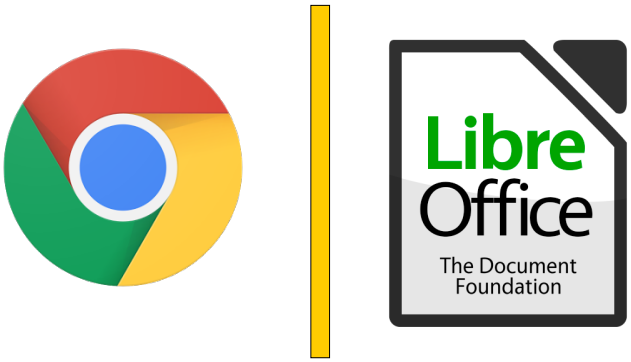
Virtual Memory





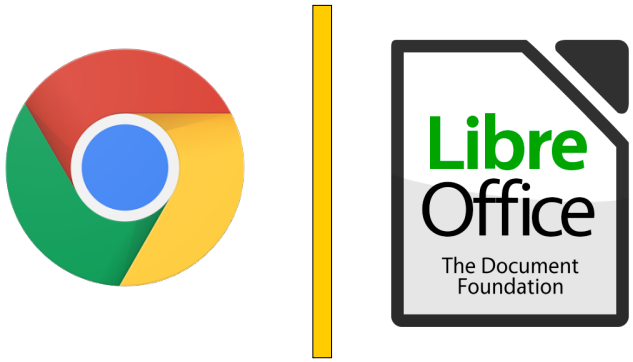
OS by example: 2

- How can we isolate Chrome from LibreOffice?



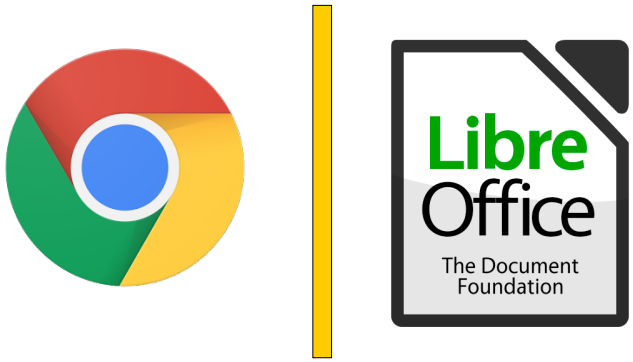
OS by example: 2

- How can we isolate Chrome from LibreOffice?
- Two types of resources:
  - registers
  - memory

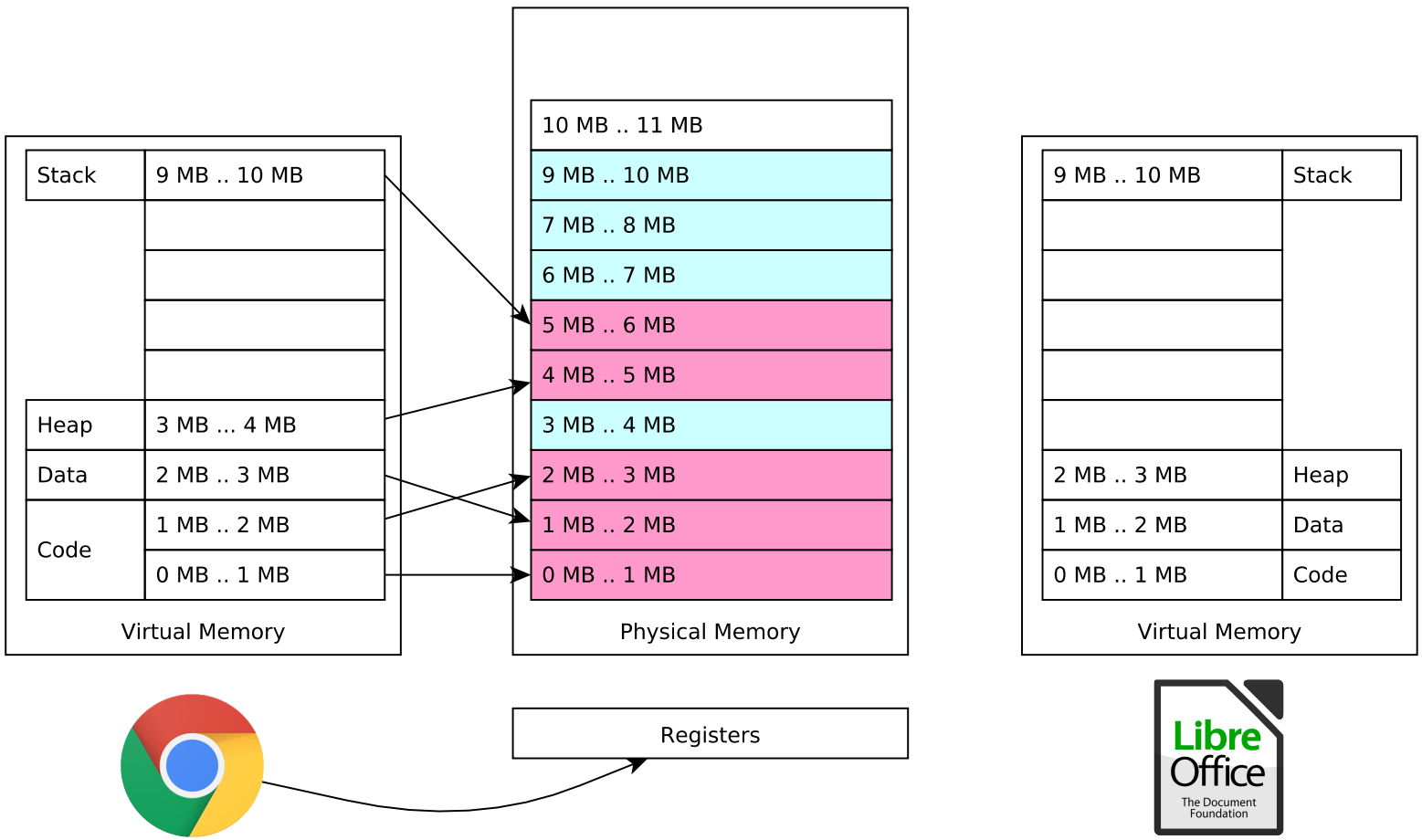


OS by example: 2

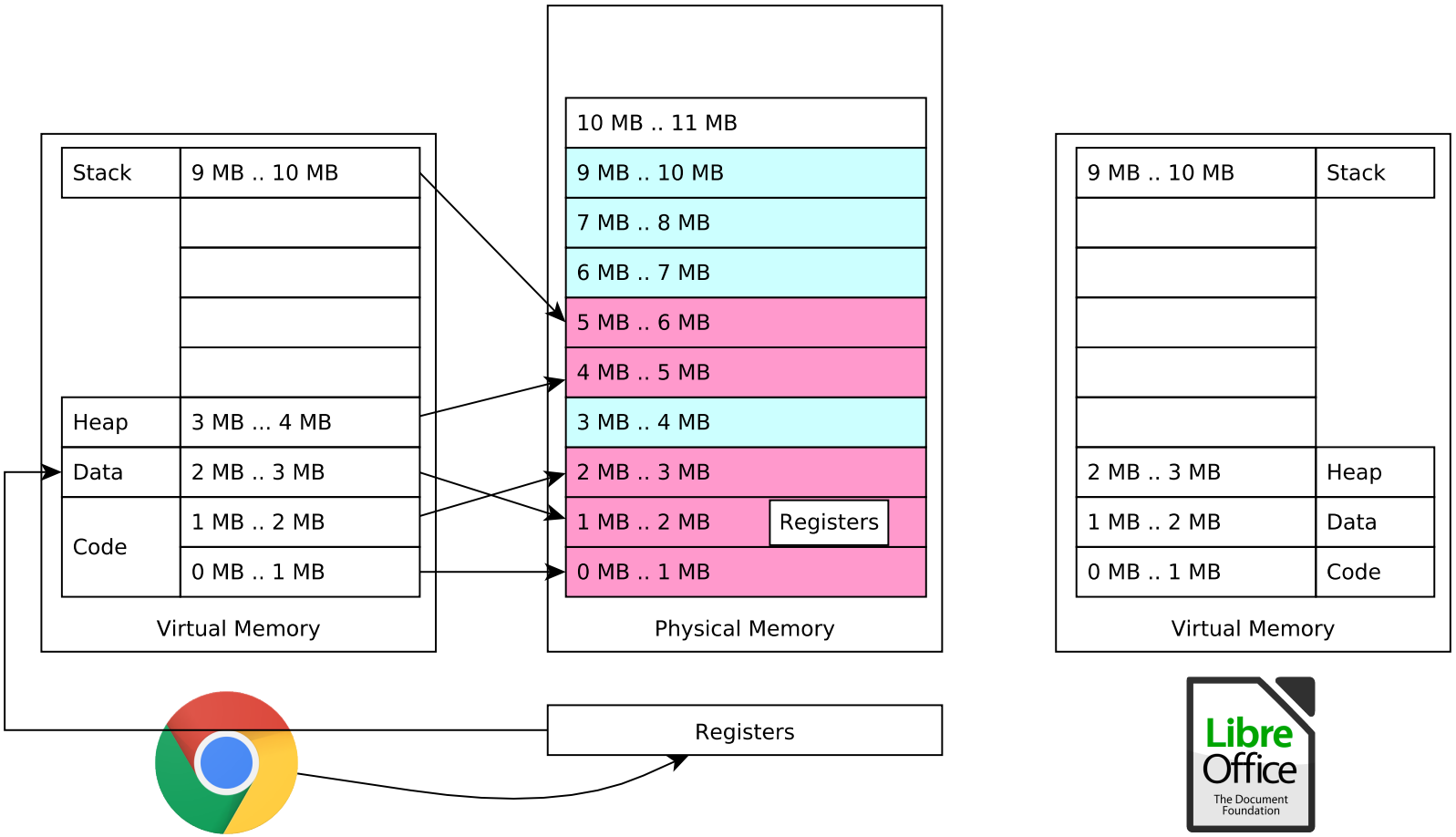
- How can we isolate Chrome from LibreOffice?
- Two types of resources:
  - registers $\Rightarrow$ multiplex
  - memory $\Rightarrow$ isolation
- One of the main tasks of an OS



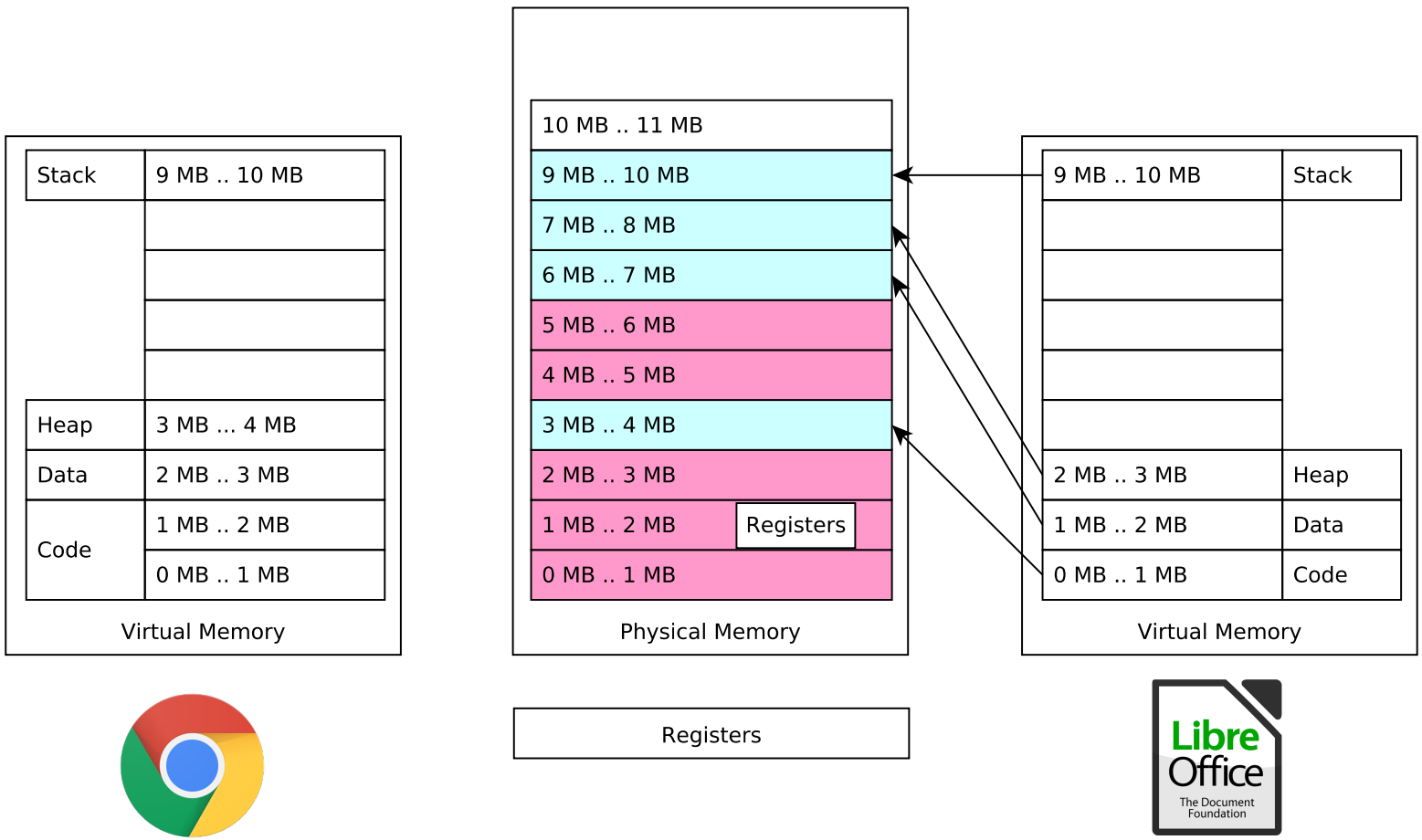
Context switch



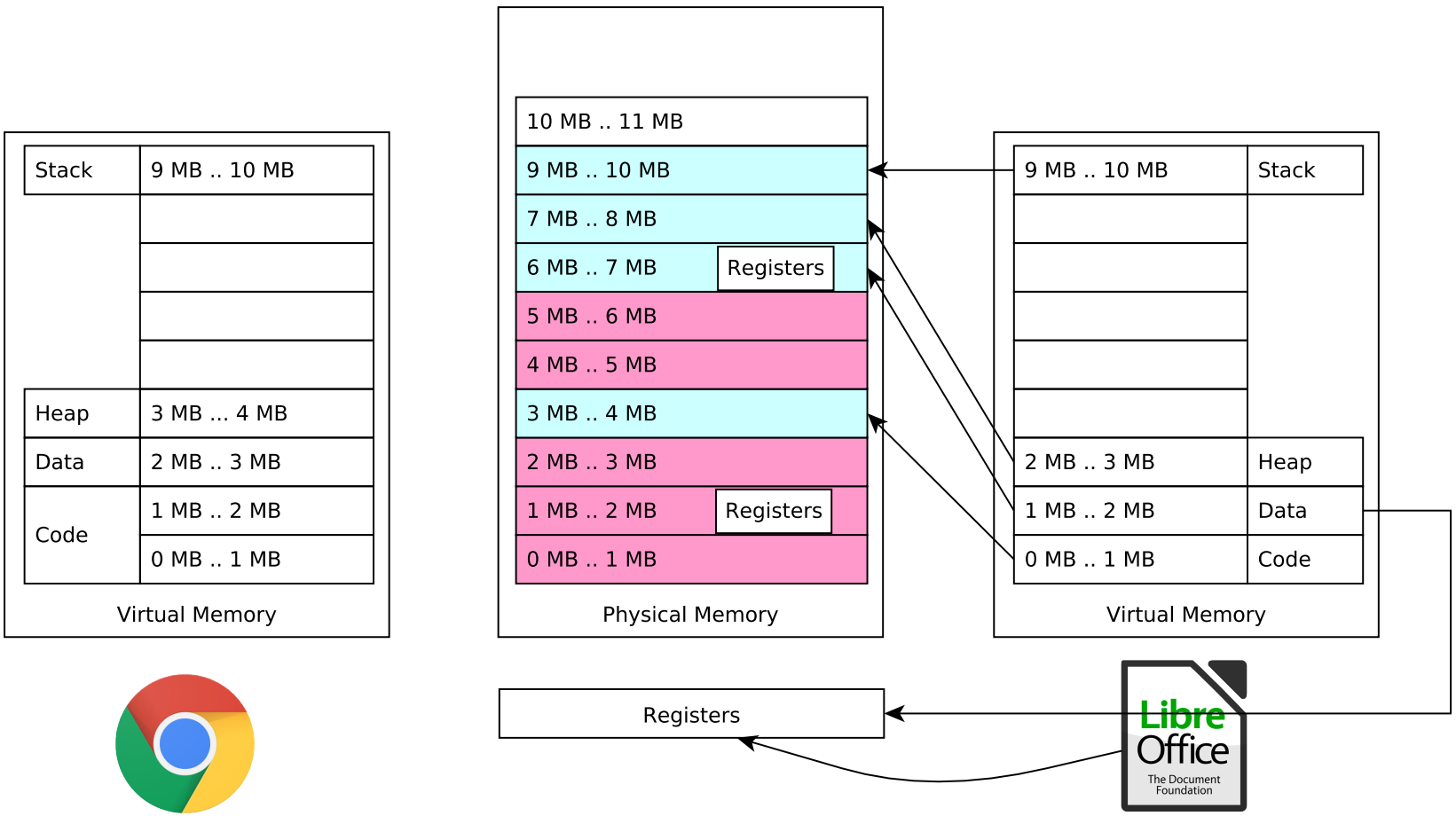
Context switch



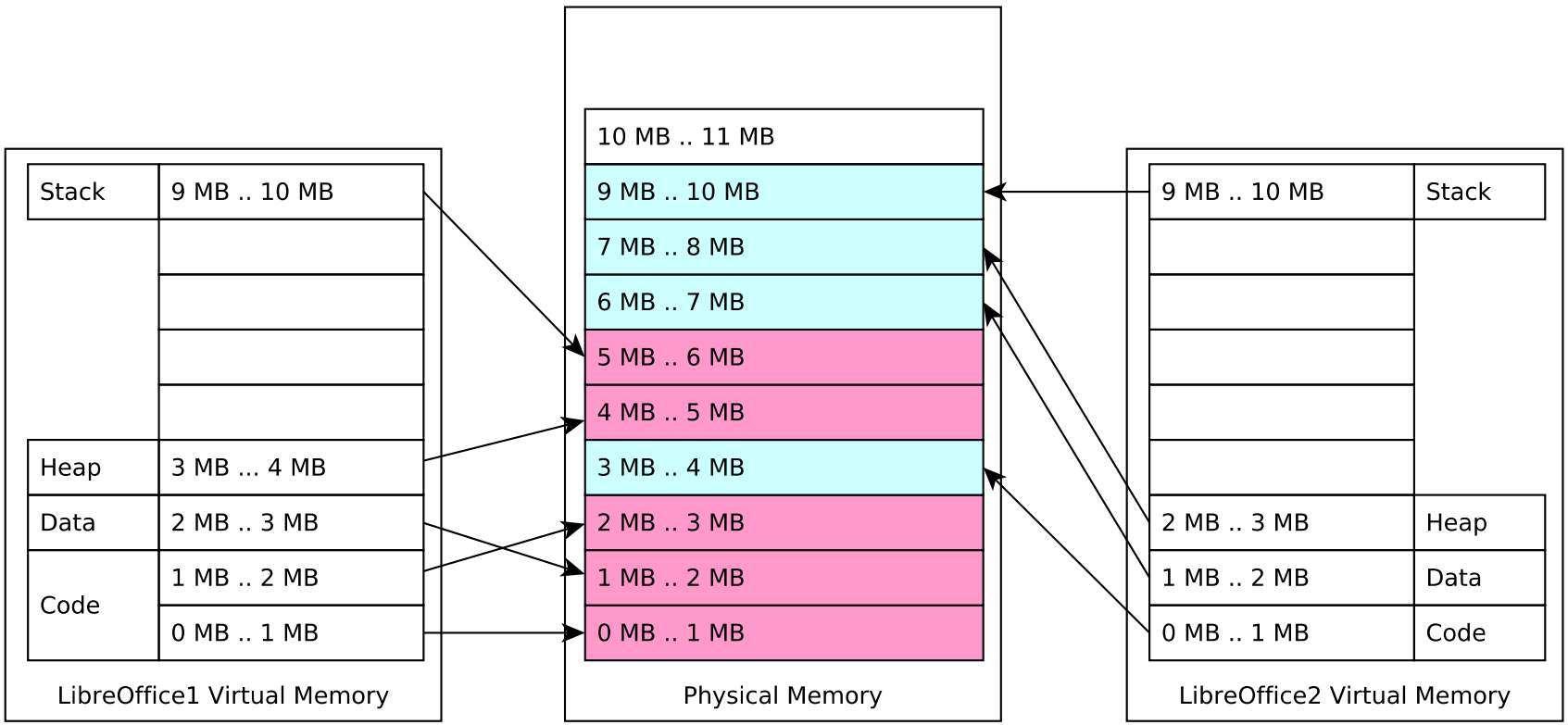
Context switch



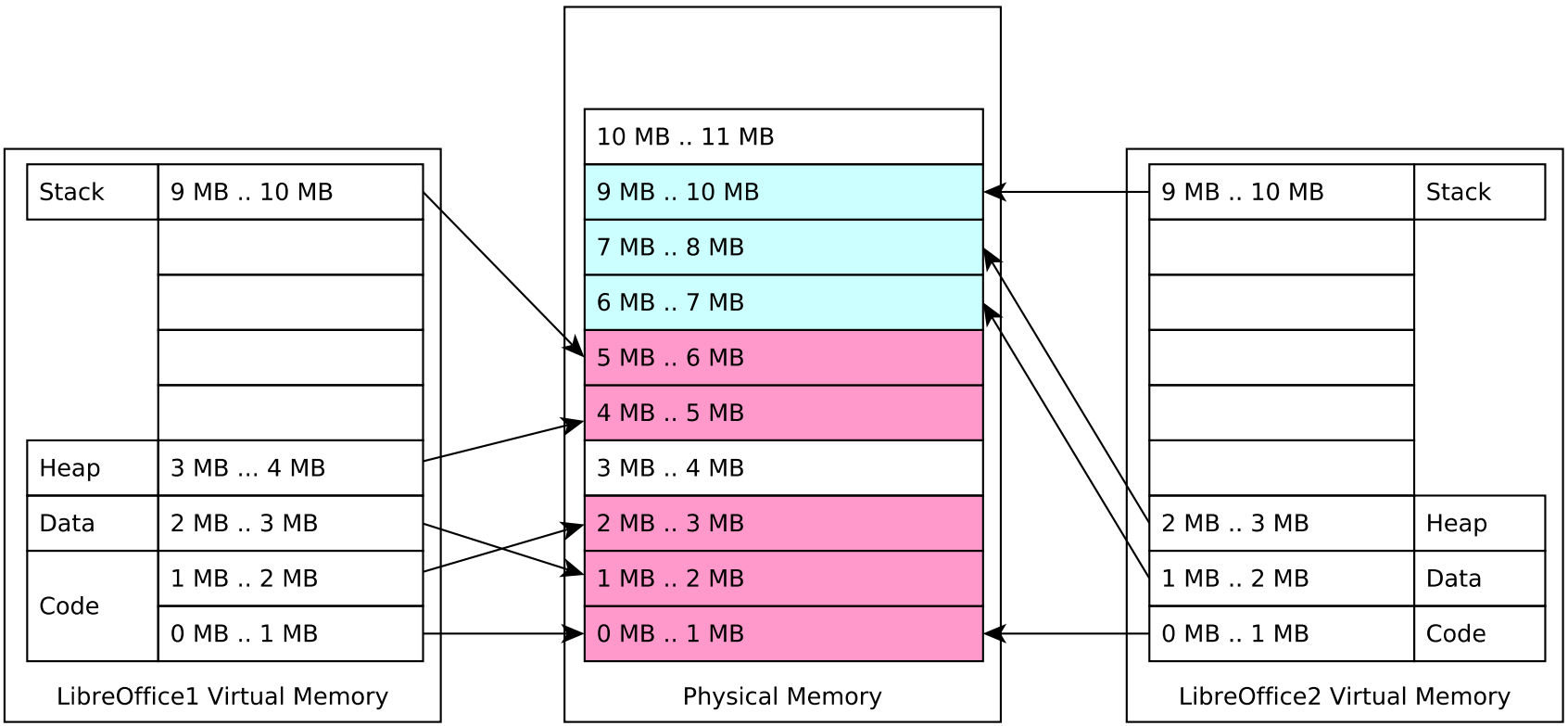
Context switch



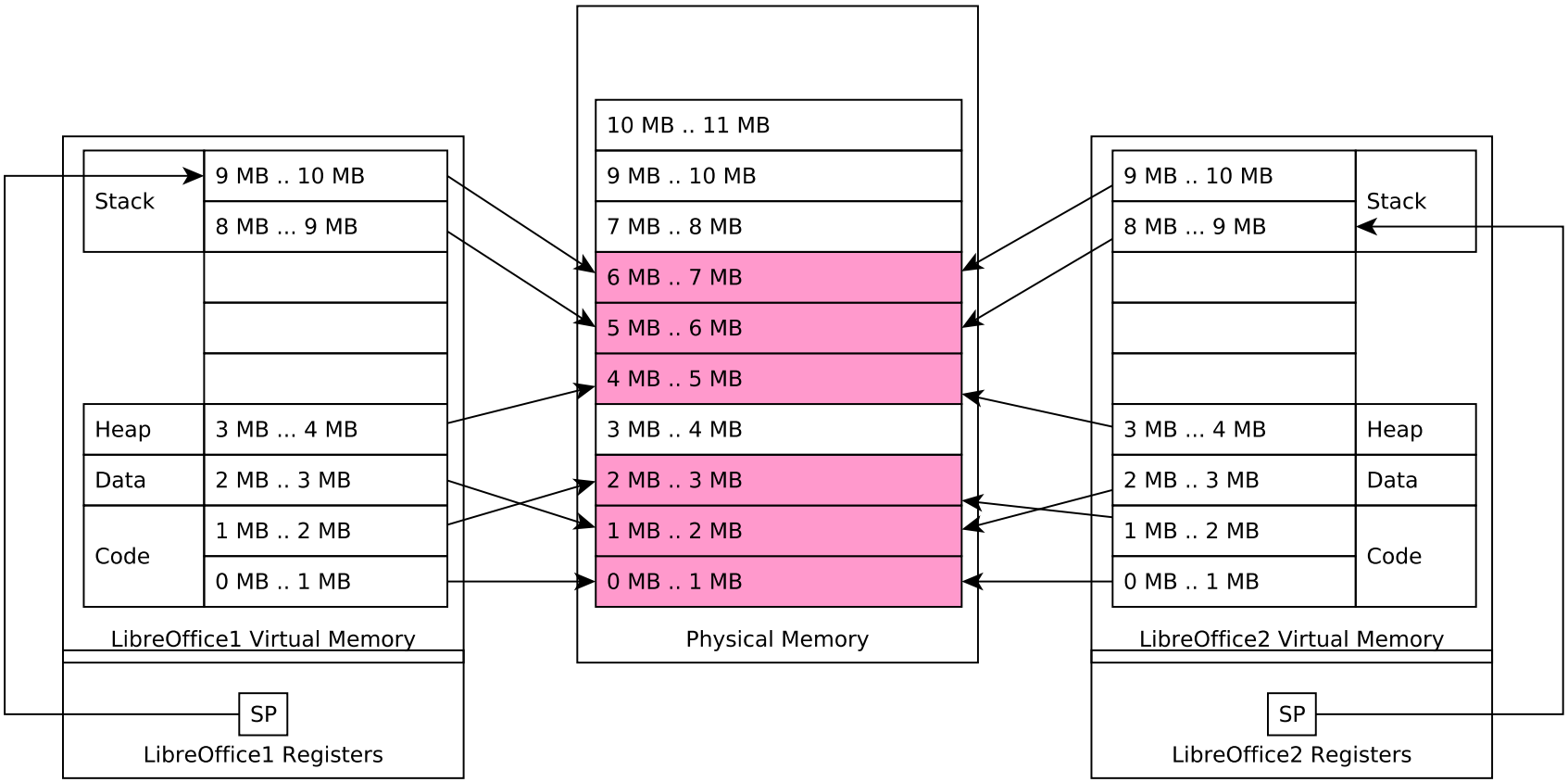
Processes


Stack
9 MB .. 10 MB
Heap
3 MB ... 4 MB
Data
2 MB .. 3 MB
Code
1 MB .. 2 MB
0 MB .. 1 MB
LibreOffice1 Virtual Memory
10 MB .. 11 MB
9 MB .. 10 MB
7 MB .. 8 MB
6 MB .. 7 MB
5 MB .. 6 MB
4 MB .. 5 MB
3 MB .. 4 MB
2 MB .. 3 MB
1 MB .. 2 MB
0 MB .. 1 MB
Physical Memory
9 MB .. 10 MB
Stack
2 MB .. 3 MB
Heap
1 MB .. 2 MB
Data
0 MB .. 1 MB
Code
LibreOffice2 Virtual Memory

Processes

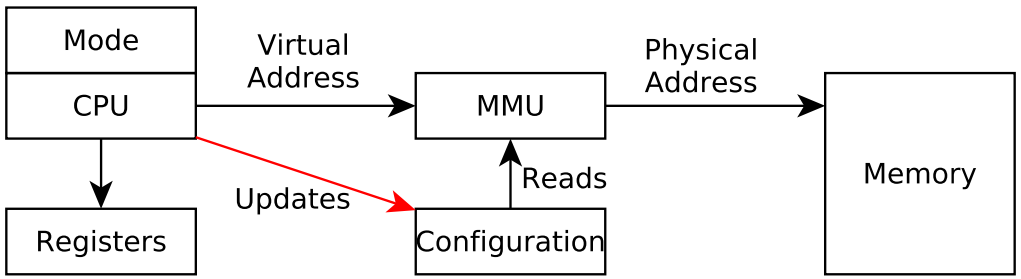

Stack
9 MB .. 10 MB
Heap
3 MB ... 4 MB
Data
2 MB .. 3 MB
Code
1 MB .. 2 MB
0 MB .. 1 MB
LibreOffice1 Virtual Memory
10 MB .. 11 MB
9 MB .. 10 MB
7 MB .. 8 MB
6 MB .. 7 MB
5 MB .. 6 MB
4 MB .. 5 MB
3 MB .. 4 MB
2 MB .. 3 MB
1 MB .. 2 MB
0 MB .. 1 MB
Physical Memory
9 MB .. 10 MB
Stack
2 MB .. 3 MB
Heap
1 MB .. 2 MB
Data
0 MB .. 1 MB
Code
LibreOffice2 Virtual Memory

Threads


10 MB .. 11 MB
9 MB .. 10 MB
7 MB .. 8 MB
6 MB .. 7 MB
5 MB .. 6 MB
4 MB .. 5 MB
3 MB .. 4 MB
2 MB .. 3 MB
1 MB .. 2 MB
0 MB .. 1 MB
Physical Memory
Stack
9 MB .. 10 MB
8 MB ... 9 MB
Heap
3 MB ... 4 MB
Data
2 MB .. 3 MB
Code
1 MB .. 2 MB
0 MB .. 1 MB
LibreOffice1 Virtual Memory
SP
LibreOffice1 Registers
9 MB .. 10 MB
8 MB ... 9 MB
Stack
3 MB ... 4 MB
Heap
2 MB .. 3 MB
Data
1 MB .. 2 MB
0 MB .. 1 MB
Code
LibreOffice2 Virtual Memory
SP
LibreOffice2 Registers

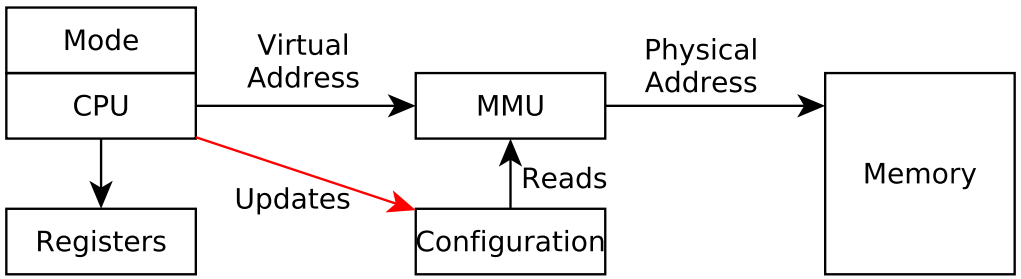
HW architecture

- Execution mode (i.e. restricted/un-restricted)
- OS takes control of un-restricted mode
- Everything else uses restricted mode



HW architecture

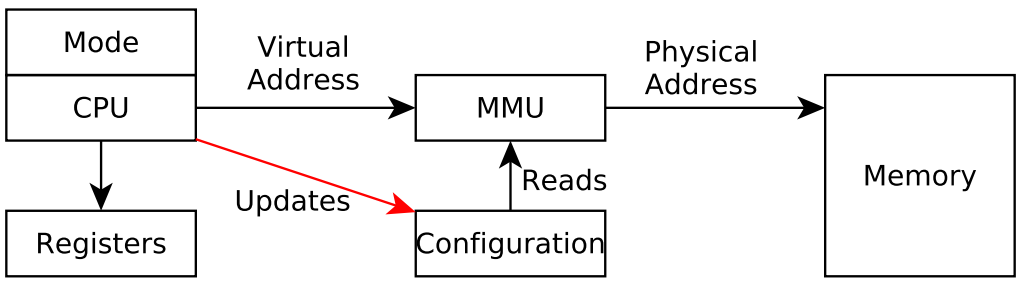
- Execution mode (i.e. restricted/un-restricted)
- OS takes control of un-restricted mode
- Everything else uses restricted mode



- From un-restricted to restricted = exiting the OS
  - instruction to change mode

HW architecture

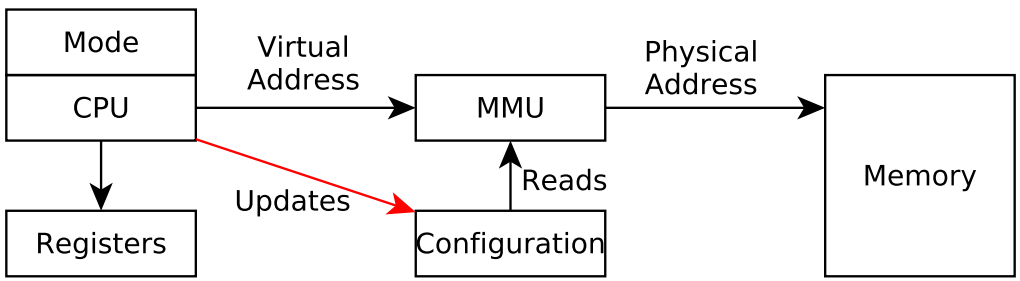
- Execution mode (i.e. restricted/un-restricted)
- OS takes control of un-restricted mode
- Everything else uses restricted mode



- From un-restricted to restricted = exiting the OS
  - instruction to change mode
- From restricted to un-restricted = entering the OS?

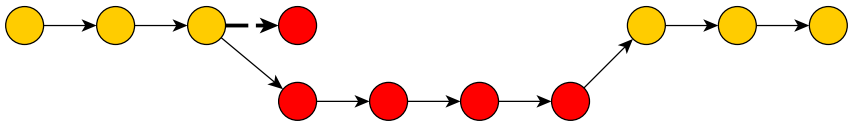
HW architecture

- Execution mode (i.e. restricted/un-restricted)
- OS takes control of un-restricted mode
- Everything else uses restricted mode



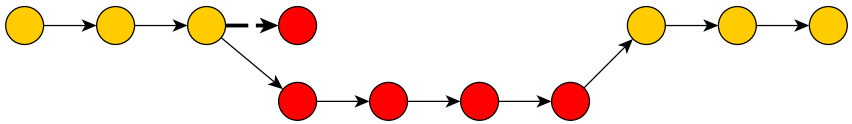
- From un-restricted to restricted = exiting the OS
  - instruction to change mode
- From restricted to un-restricted = entering the OS?
  - interrupts

SW interrupt



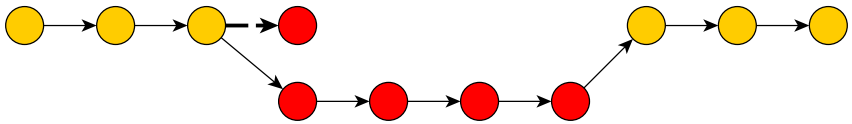
1. Application executes the Software Interrupt Instruction

SW interrupt



1. Application executes the Software Interrupt Instruction
2. HW
   - Mode switches to un-restricted
   - Program counter jumps to the OS vector table

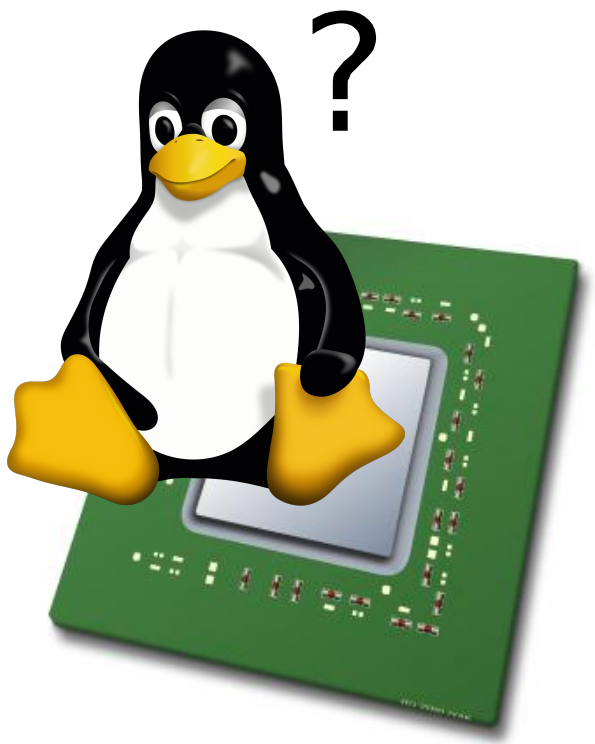
SW interrupt



1. Application executes the Software Interrupt Instruction
2. HW
   - Mode switches to un-restricted
   - Program counter jumps to the OS vector table
3. OS
   - Saves the registers (context) in the process memory
   - Jumps to the exception handler
   - ...
     (e.g. change MMU configuration and "logical" active process)
   - Loads the registers (context) from the memory of the active process
   - Switches the mode to restricted

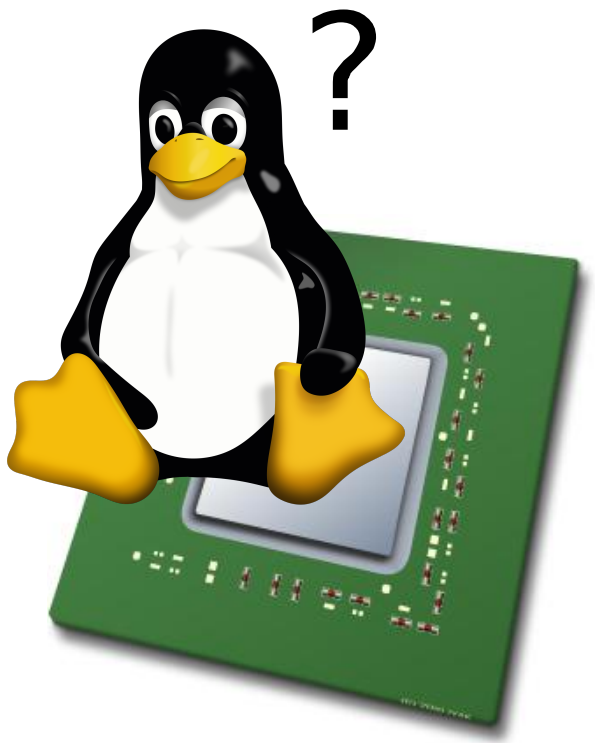
OS Memory

1. Question 4: How does the OS access the memory?
   - A - Using directly physical addresses
   - B - Using the same virtual addresses of processes
   - C - Using dedicated virtual addresses



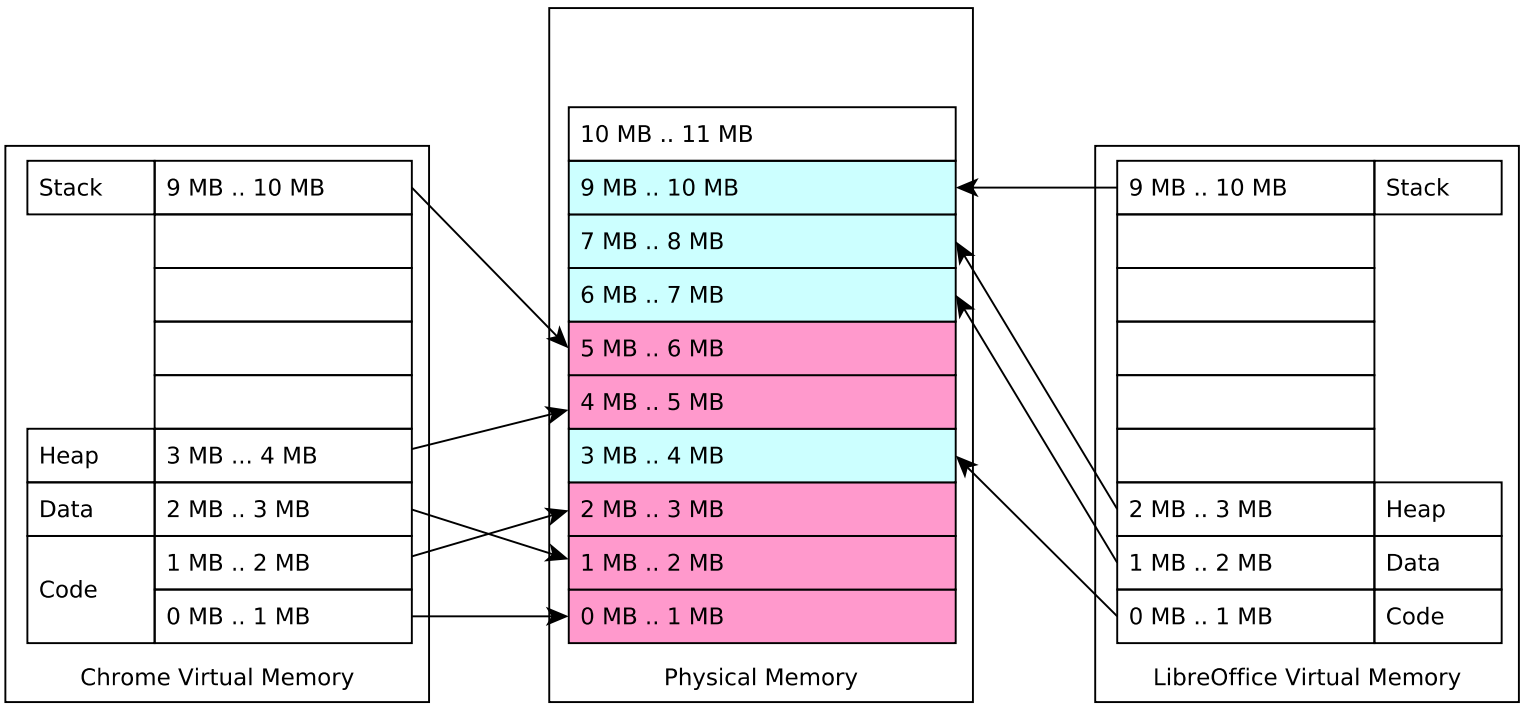
OS Memory

1. Question 4: How does the OS access the memory?
   - A - Using directly physical addresses
   - B - Using the same virtual addresses of processes
   - C - Using dedicated virtual addresses



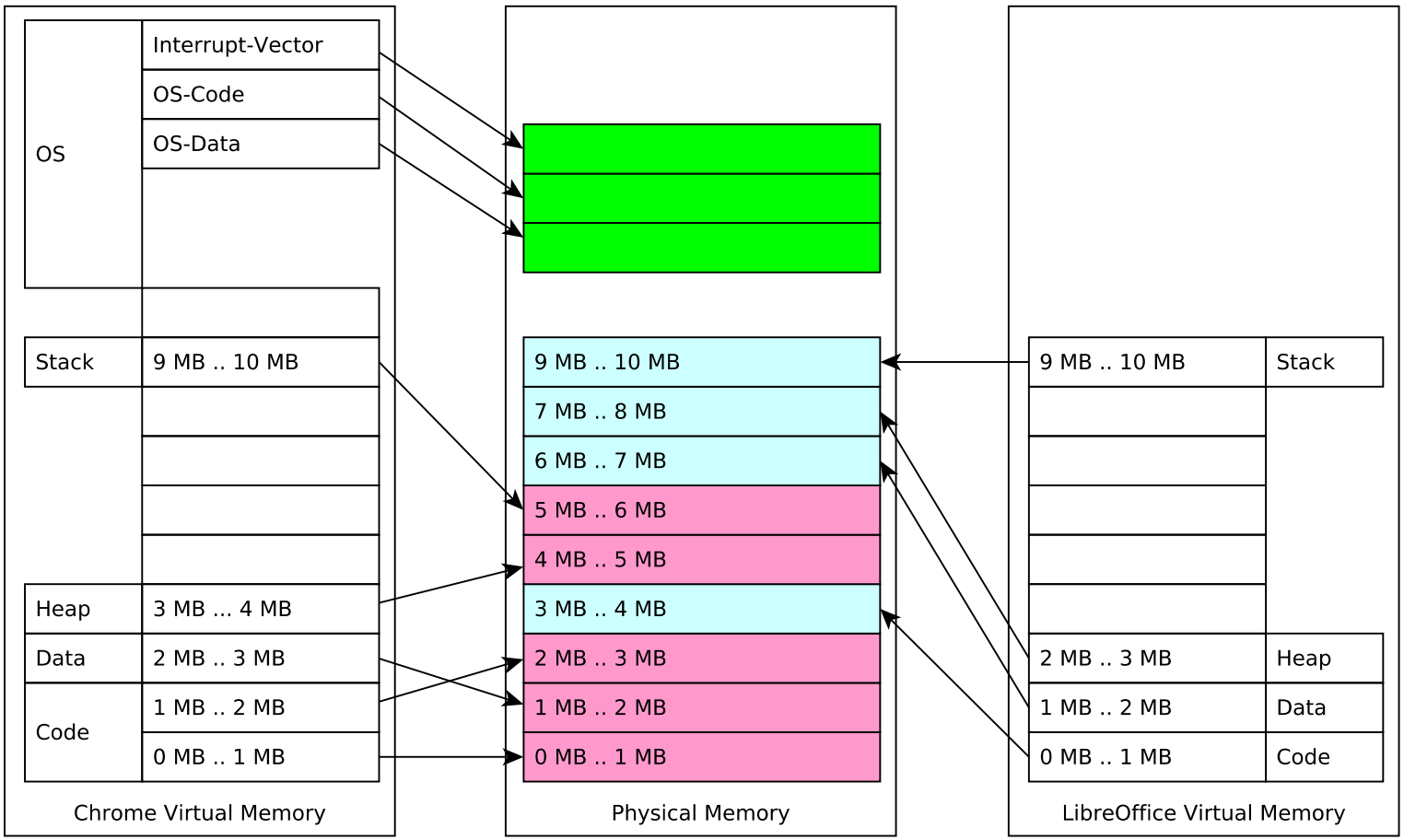
OS Memory

1. Question 4: How does the OS access the memory?



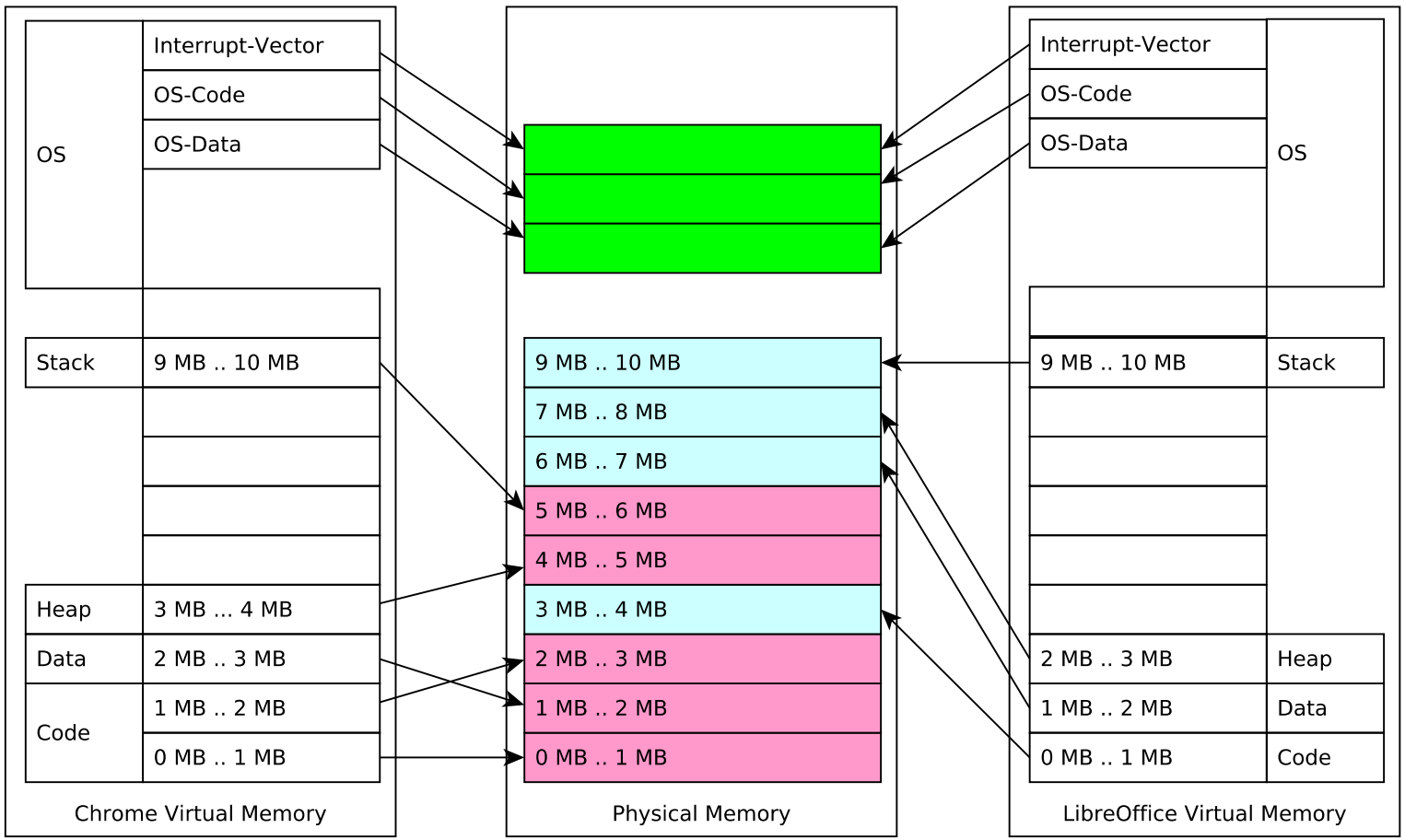
OS Memory

1. Question 4: How does the OS access the memory?



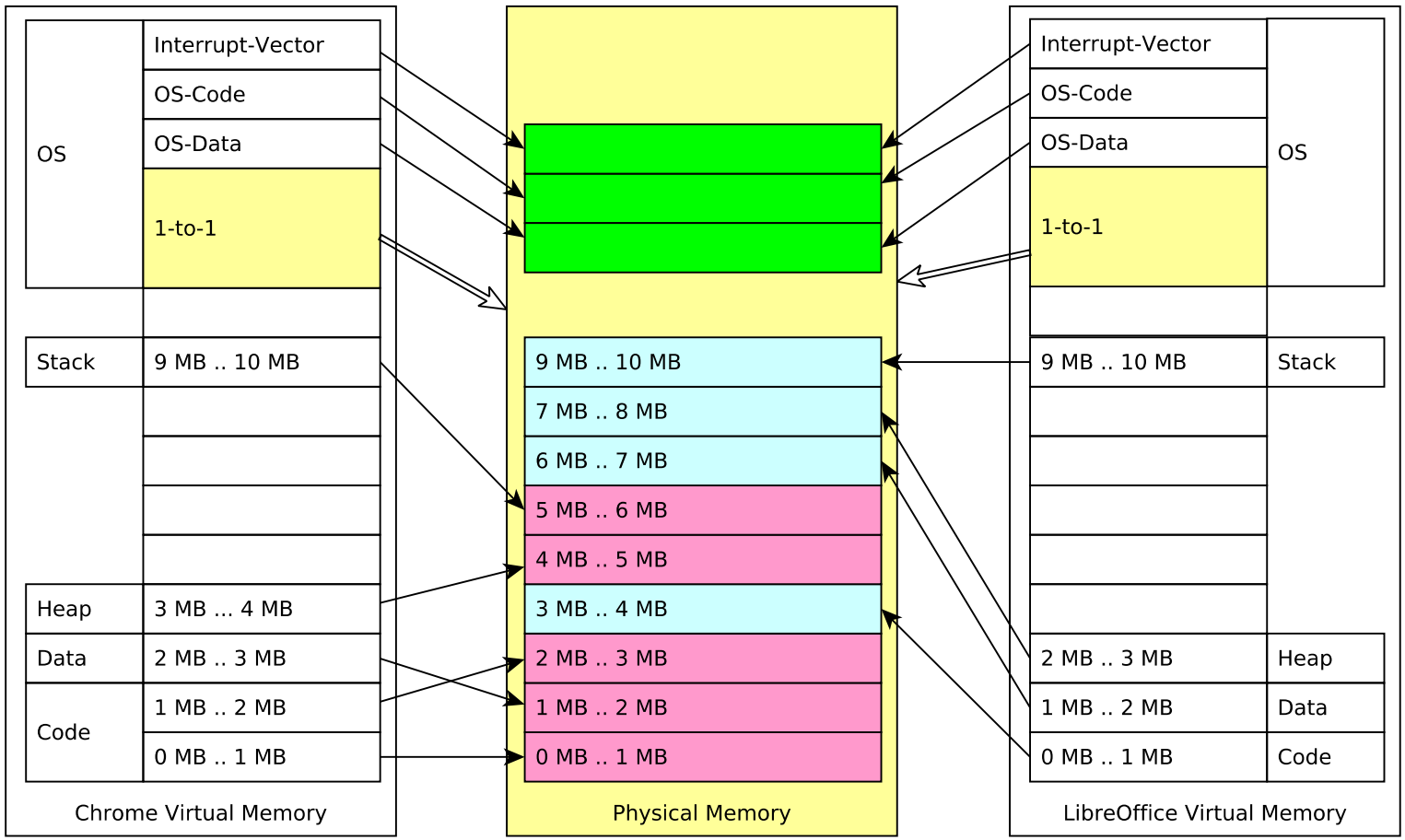
OS Memory

1. Question 4: How does the OS access the memory?



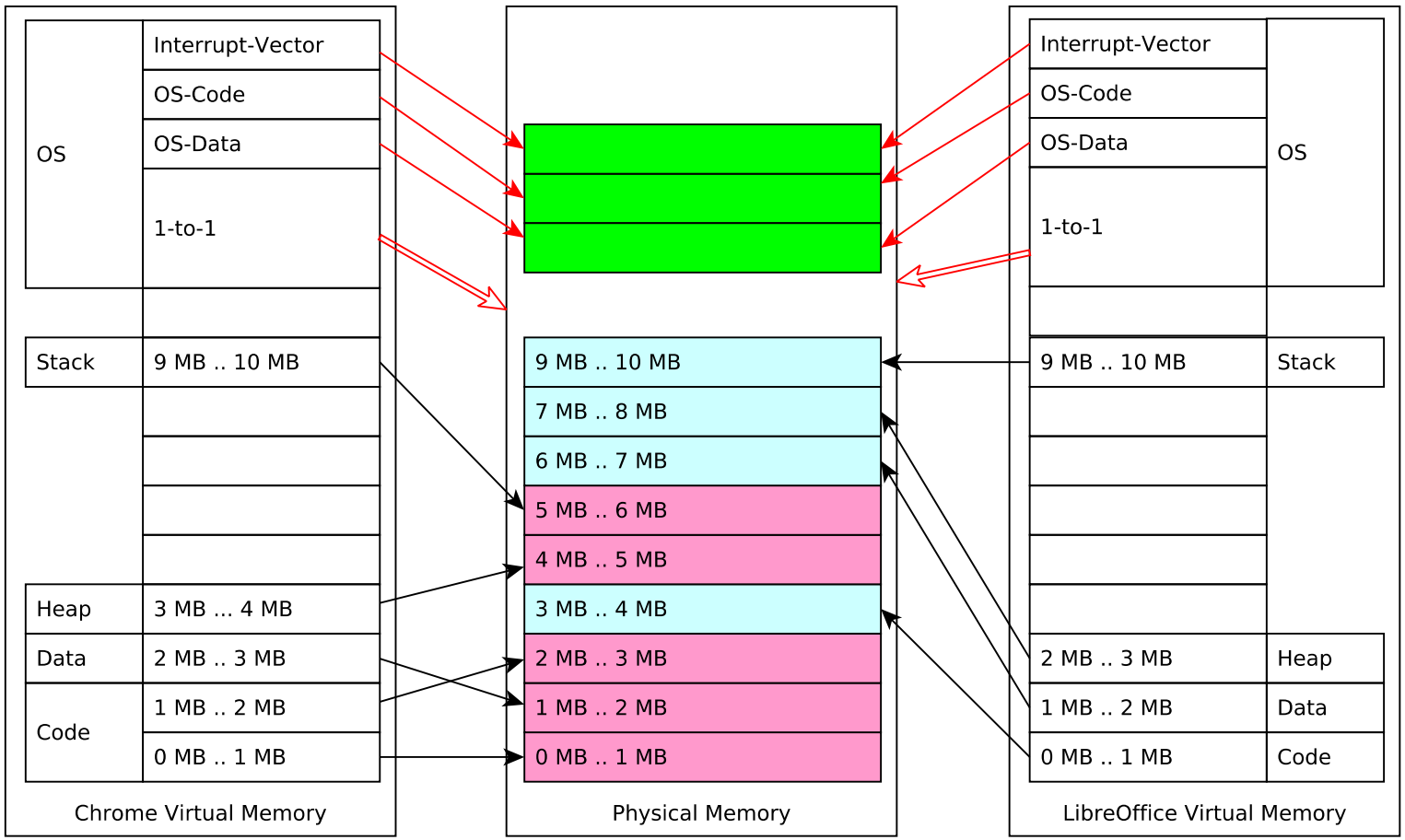
OS Memory

1. Question 4: How does the OS access the memory?



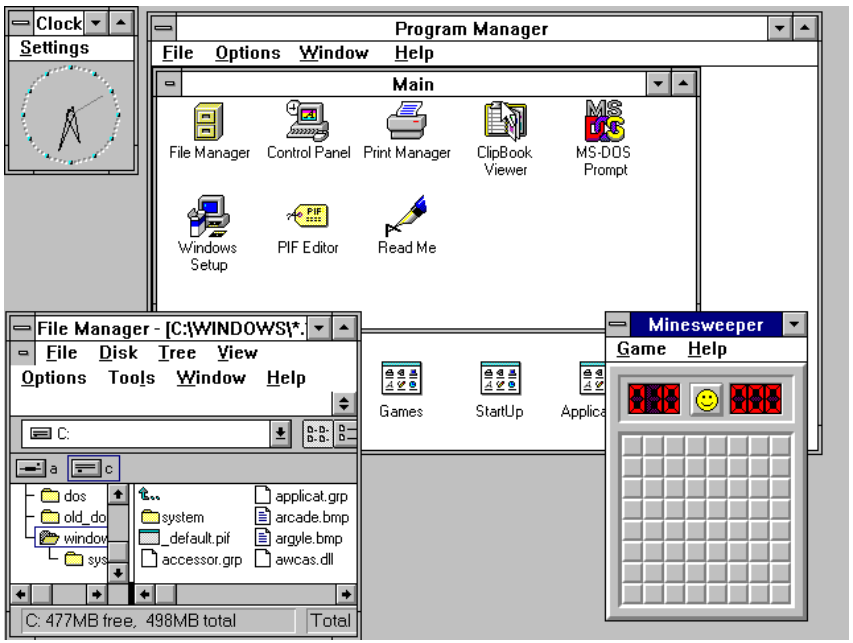
OS Memory

1. Question 4: How does the OS access the memory?
2. MMU enforces access right

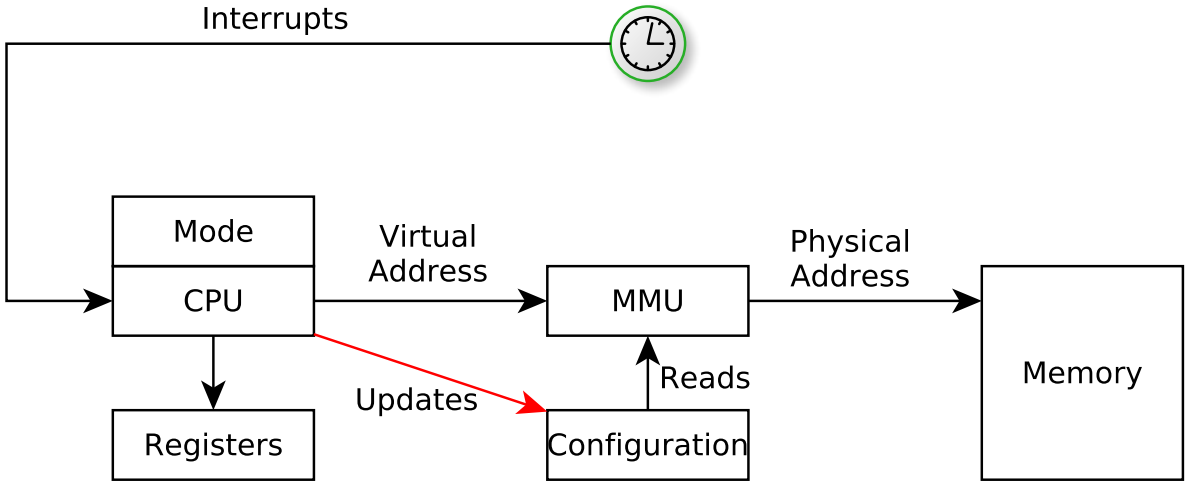


Non-preemptive multitasking

- Context switch driven by software interrupt
- Cooperative/non-preemptive multitasking

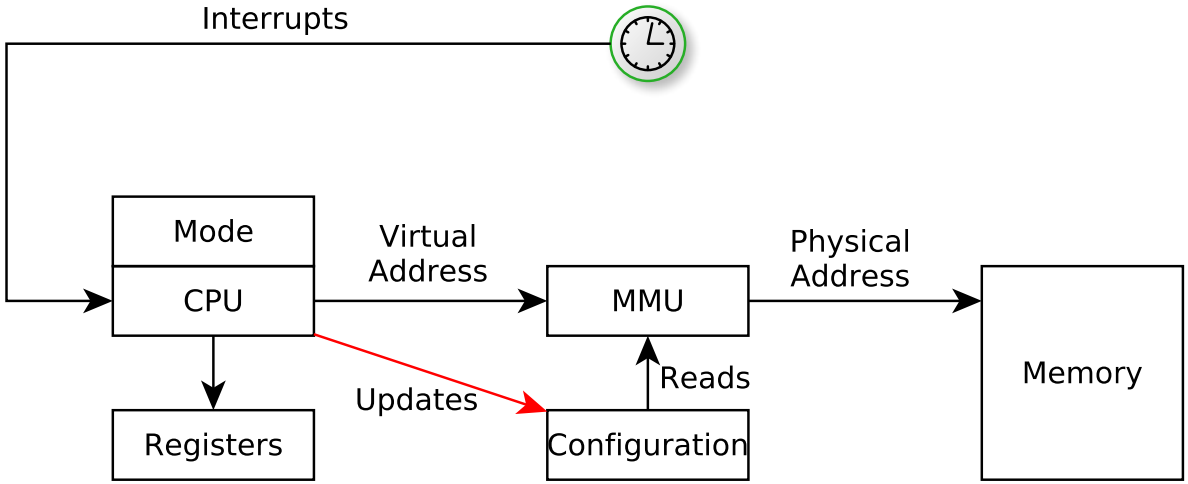


HW Interrupt



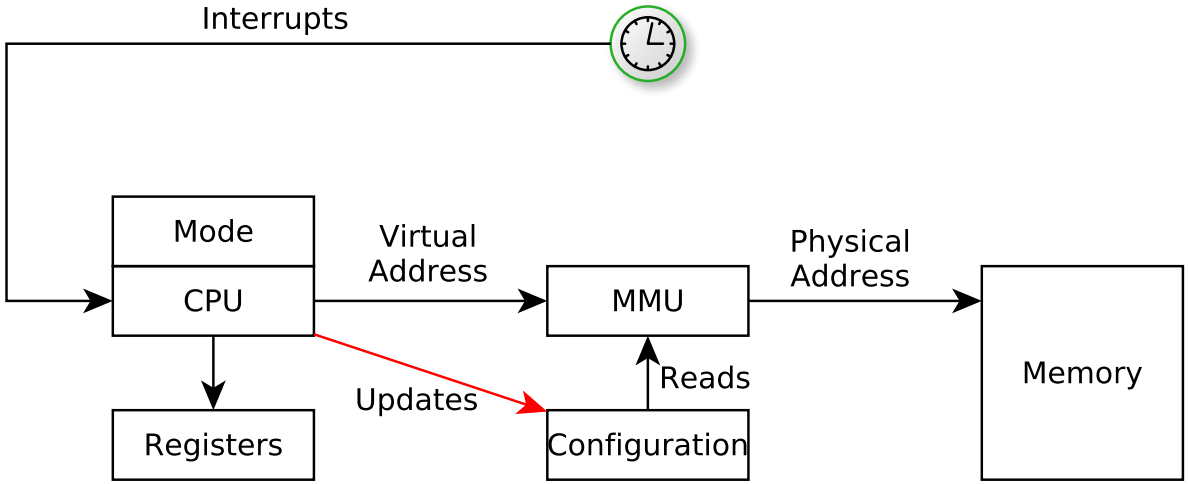
1. Peripheral (timer) delivers an Hardware Interrupt

HW Interrupt



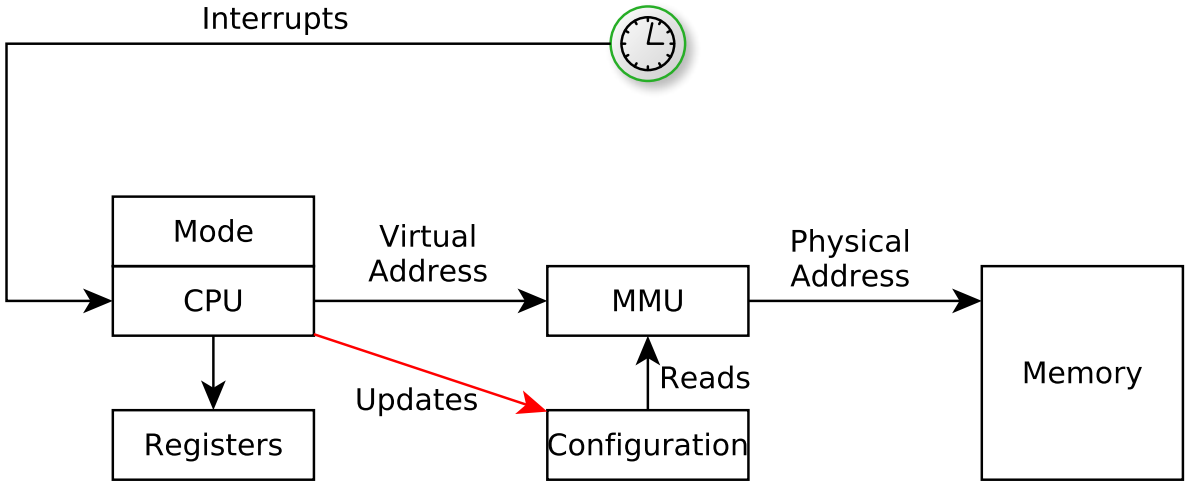
1. Peripheral (timer) delivers an Hardware Interrupt
2. HW
   - Mode switches to un-restricted
   - Program counter jumps to the OS "vector table"

HW Interrupt



1. Peripheral (timer) delivers an Hardware Interrupt
2. HW
   - Mode switches to un-restricted
   - Program counter jumps to the OS "vector table"
3. OS implements context switch

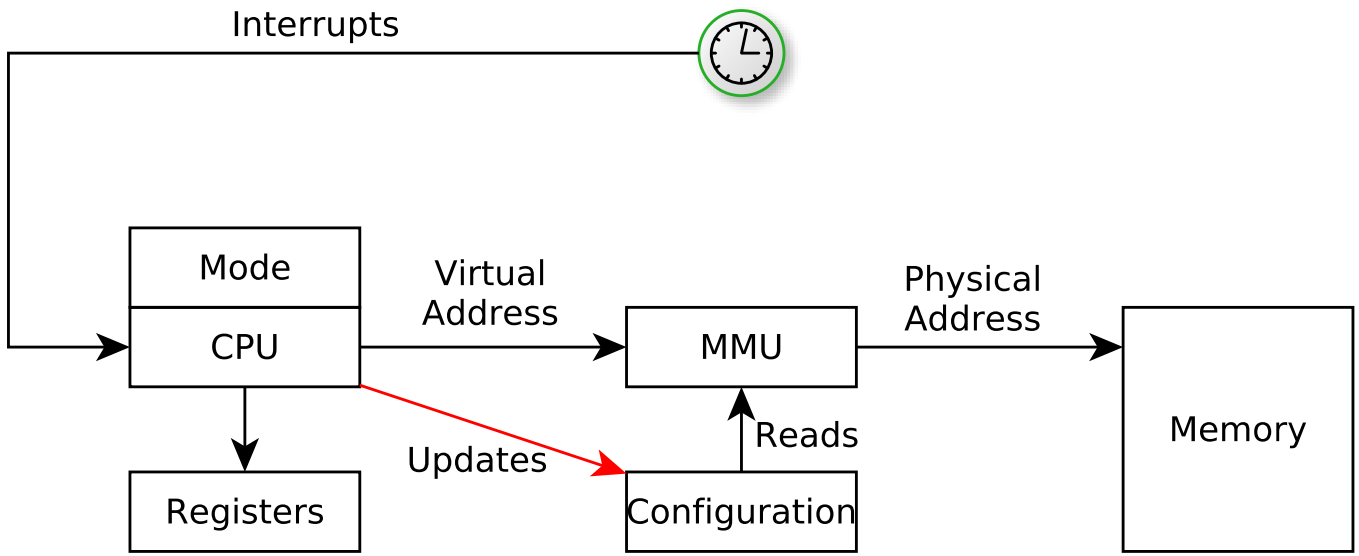
HW Interrupt



1. Peripheral (timer) delivers an Hardware Interrupt
2. HW
   - Mode switches to un-restricted
   - Program counter jumps to the OS "vector table"
3. OS implements context switch
4. Preemptive multitasking

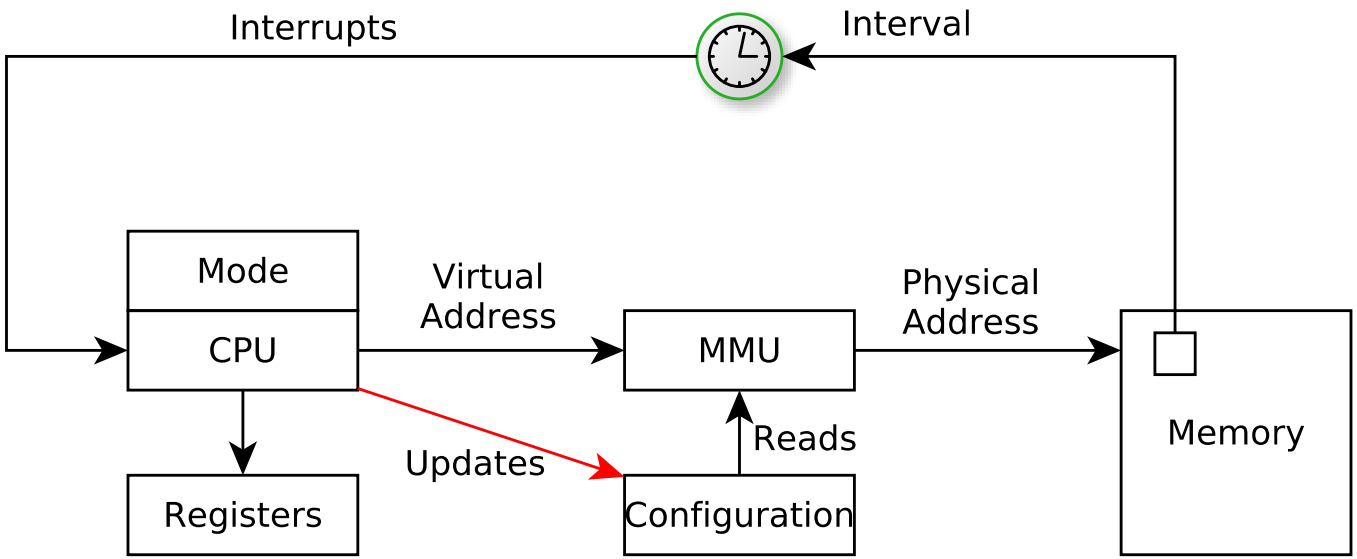
Peripherals

1. Interrupts



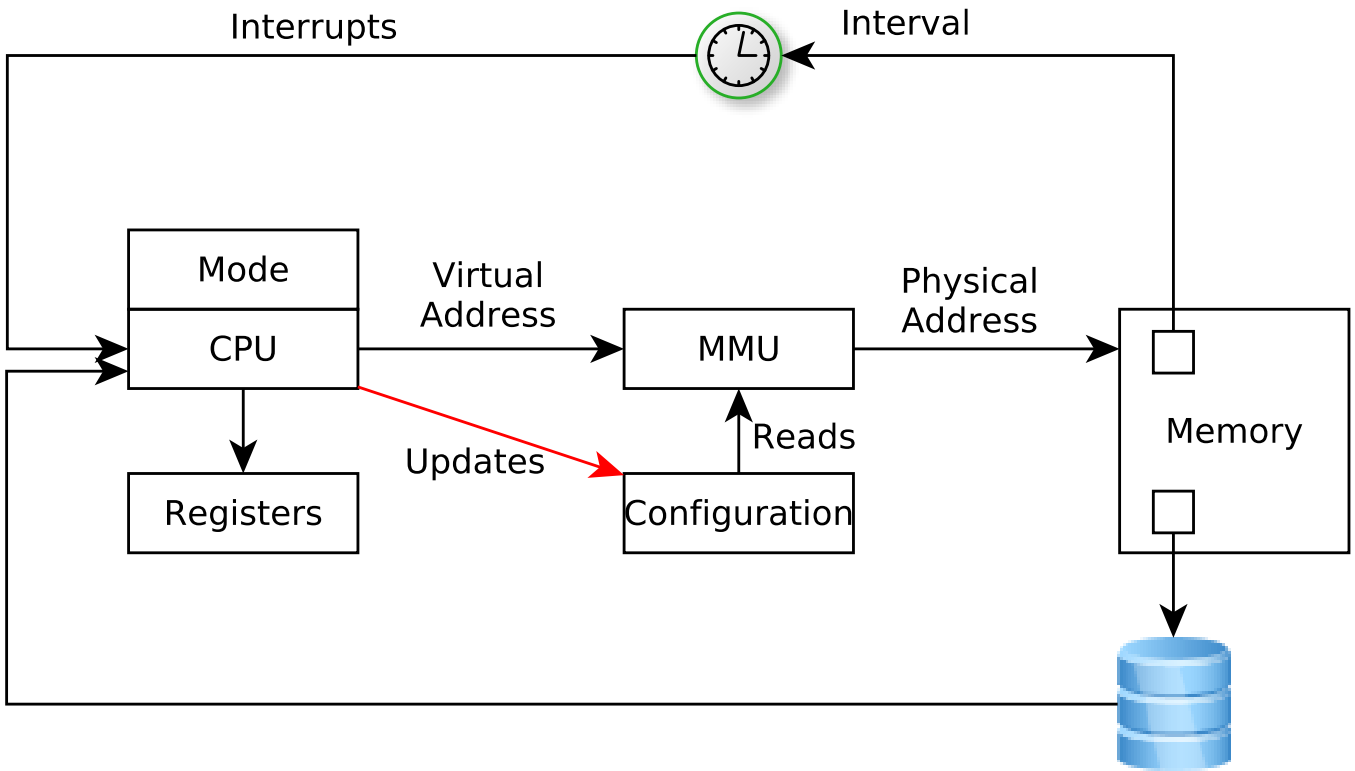
Peripherals

1. Interrupts
2. Memory mapped devices



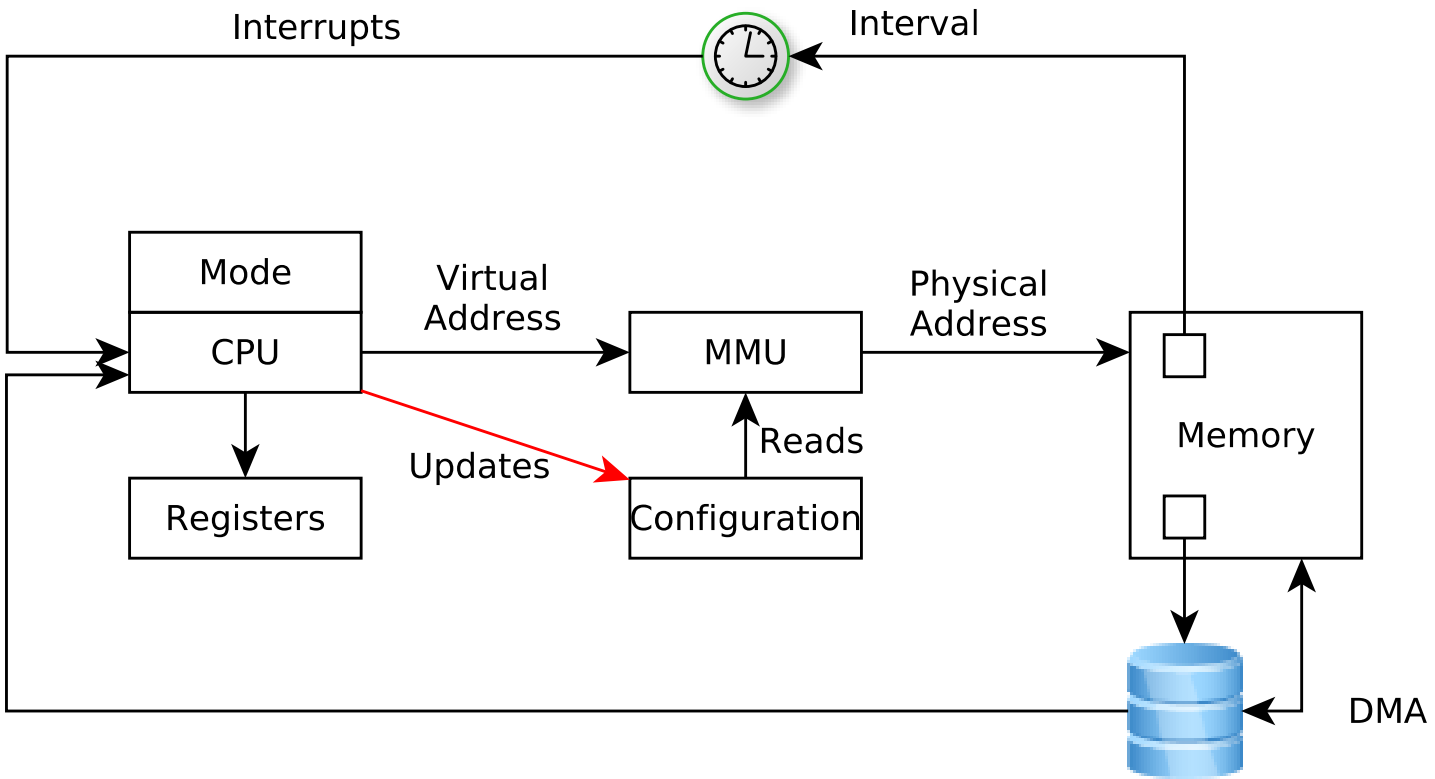
Peripherals

1. Interrupts
2. Memory mapped devices



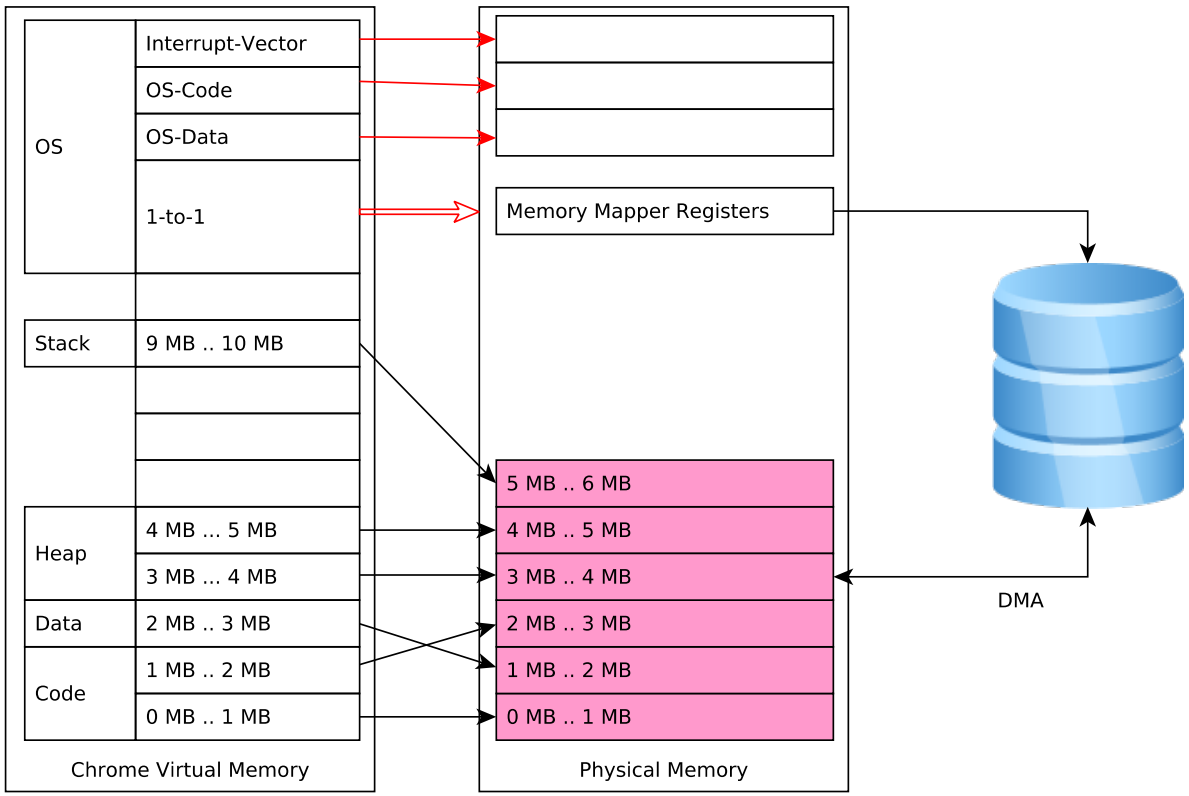
Peripherals

1. Interrupts
2. Memory mapped devices
3. Direct Memory Access

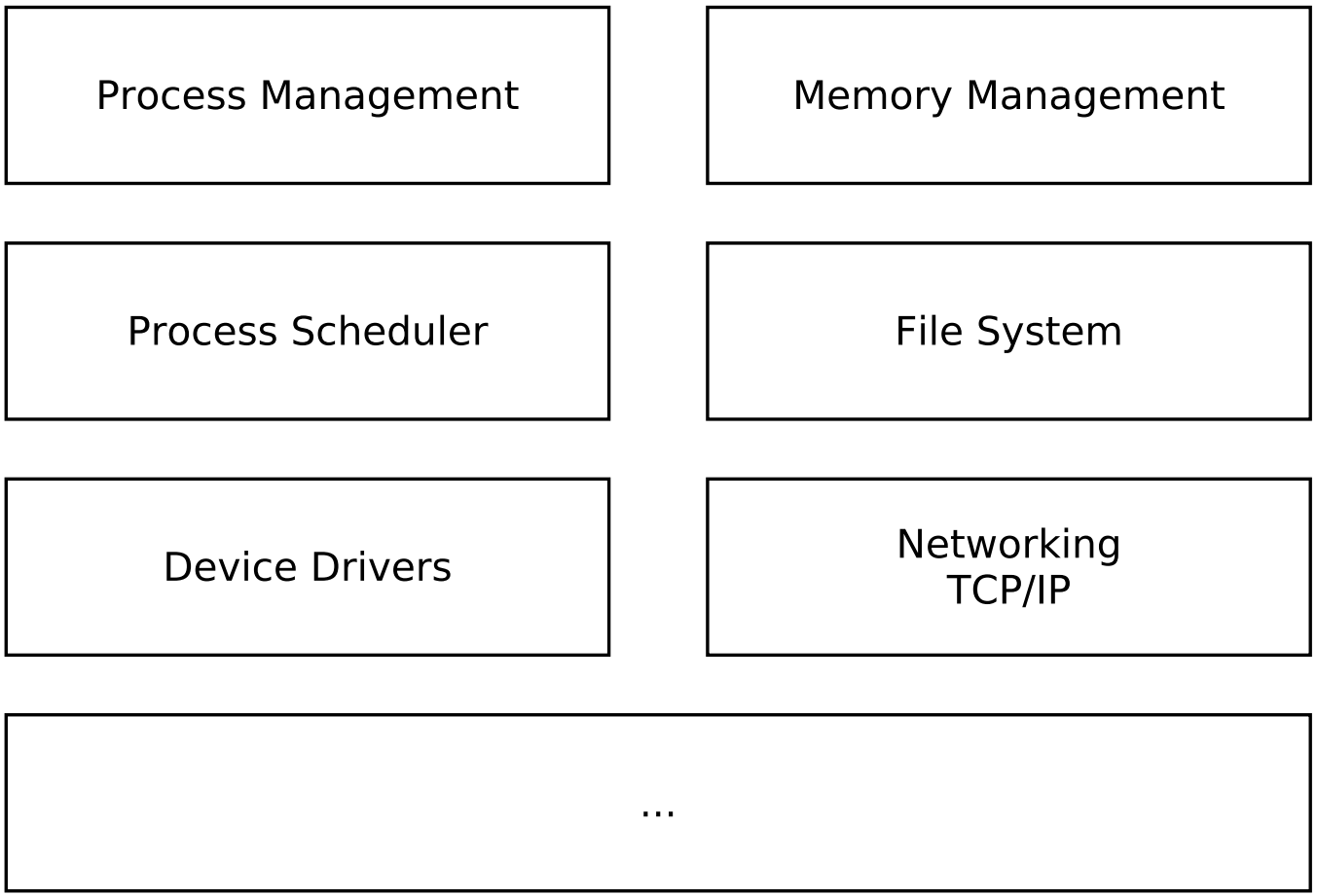


Peripherals

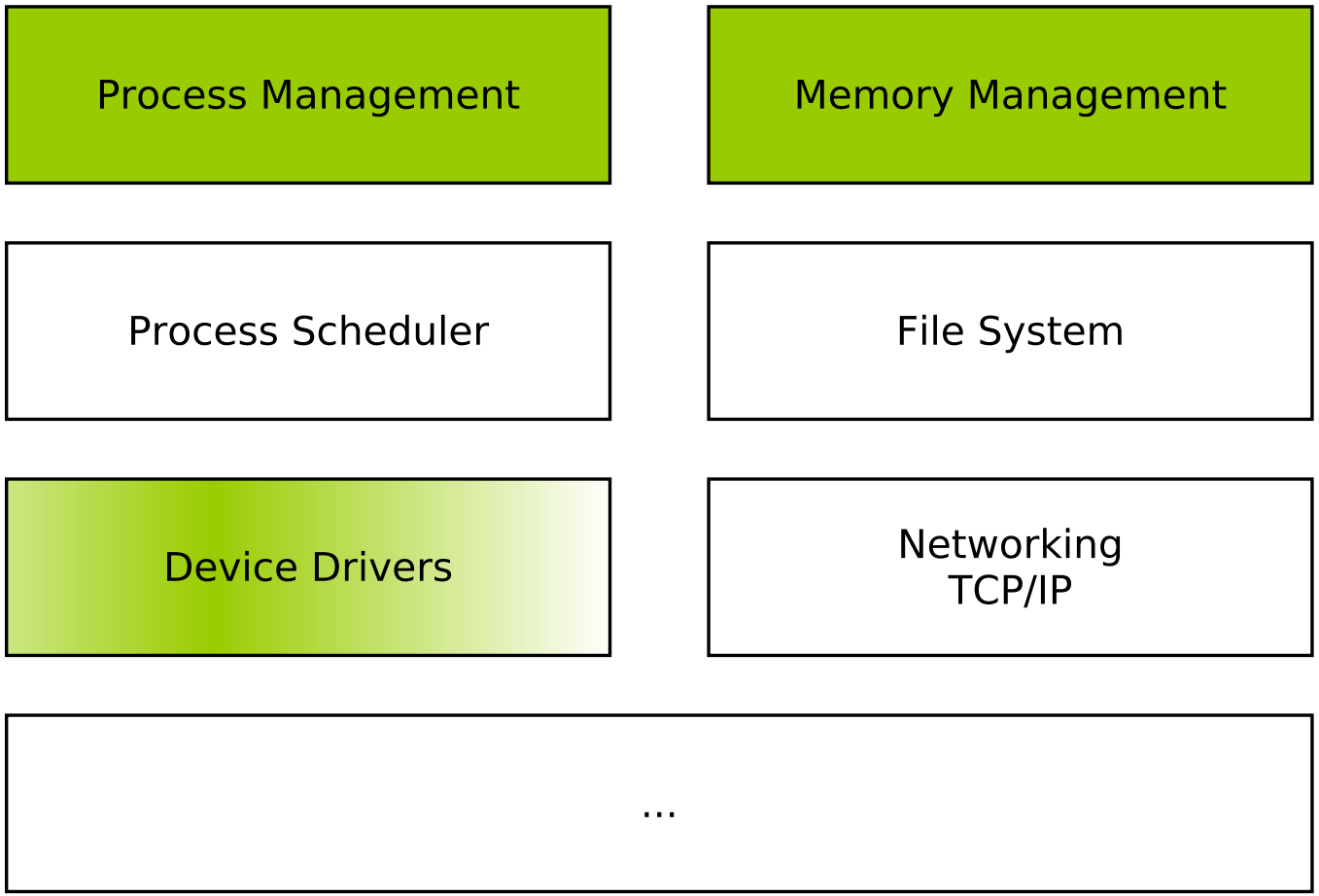
1. Interrupts
2. Memory mapped devices
3. Direct Memory Access



OS blocks


Process Management
Memory Management
Process Scheduler
File System
Device Drivers
Networking
TCP/IP
...

OS blocks


Process Management
Memory Management
Process Scheduler
File System
Device Drivers
Networking
TCP/IP
...

System calls

- Request OS services
  - File management
  - Device Management
  - Memory allocation
  - Process Control
  - Communication
  - Print to the terminal
- Calling convention
  - Usually invoked by Software Interrupts
  - HW-OS dependent

OS API

- Syscall low level (and non-portable)
- Library or API that sits between normal programs and the operating system
- Unix-like: libc, glibc
- Windows NT, that API is part of the Native API, in the ntdll.dll library
- in POSIX: open, read, write, close, wait, exec, fork, exit, mmap

Summary

- Process memory layout
- Processes vs Thread
- OS is key part of the Trusting Computing Base
- OS allows sharing resources
- OS monitors resources

Questions?

Questions?