System Architecture

2 System Overview

Design, Structure, Interfaces

October 27 2008

Winter Term 2008/09

Gerd Liefländer

Agenda

- System Design
  - Criteria and Objectives
  - System Abstractions
  - Basic Concepts
- System Structure
  - Library
  - Kernel
  - System Call
  - Interfaces and “Virtual Machines”

http://www.osdata.com/kind/history.htm
http://www.armory.com/~spectre/tech.html
http://courses.cs.vt.edu/~cs1104/VirtualMachines/OS.1.html

System Design

Criteria and Basic Terms

Abstractions

Concepts

Interesting paper: Jan-Peter Richter et al.:

"Serviceorientierte Architektur (SOA)",

Informatik Spektrum, Oktober 2005

Design Parameters of an OS

- Size (Handheld, NC, NB, PC, WS, Super Computer)
- Price (low-, medium-, high-budget systems)
- Performance (slow, ..., ultra fast)
- Power consumption (low, ..., high)
- Scalability (non, slightly, ..., highly scalable)
- Versatility (dedicated ~, ..., general purpose systems)
- Security (open systems, ..., closed systems)
- Homogeneity (homogeneous, heterogeneous)
- Mobility (stationary, ..., fully mobile systems)

Terms: Policy & Mechanism

- Scheduling → Dispatching
- Paging → Replacement
- ...

Concept & Implementation

- Interaction → IPC
  - Cooperation → Shared memory
  - Communication → Pipe, socket, message

Process & Thread

Process:

- Application or system program inside the system waiting to be executed or executing
- Instance of a program active on a computer
- Standard system entity of resource ownership

Thread:

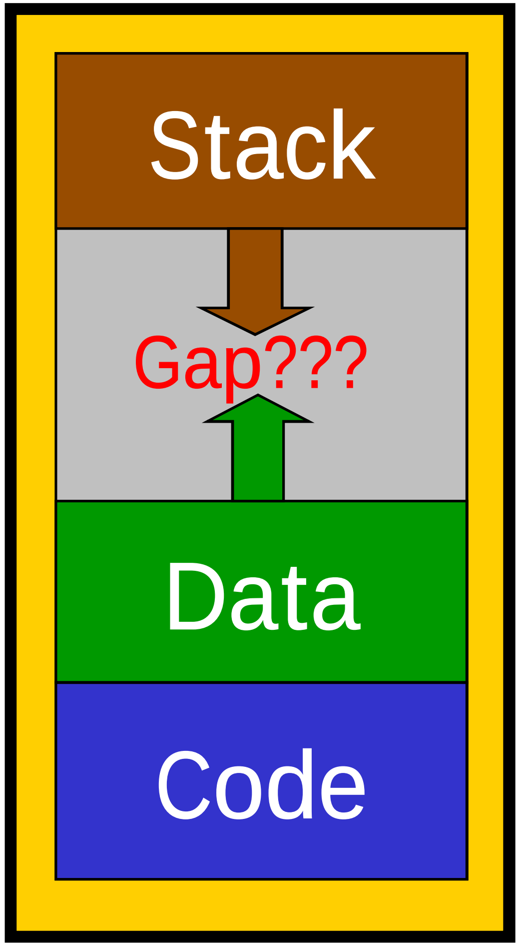
- Activity entity[1] assigned to or executed on a **CPU**

[1] *Smallest entity in a system?*

Process & Address Space

- Often three segments
  - Text (= code)
  - Data (global variables)
  - Stack
    - Local variables
    - Frame of procedure
- Note:
  - Data can dynamically grow up
  - Stack can dynamically grow down

Address Space



Question: How to guarantee a non-zero gap?

Concurrency

- Processes execute in parallel or concurrently on a single- or on a multi-processor system
- Threads of a multithreaded task can be executed in parallel or concurrently
  - Dependent on thread model
  - Dependent on underlying HW
- "Race conditions" can happen if you be lazy with your concurrency $\Rightarrow$
  - Synchronize threads/processes
  - Never rely on timing conditions during the tests

Memory Management

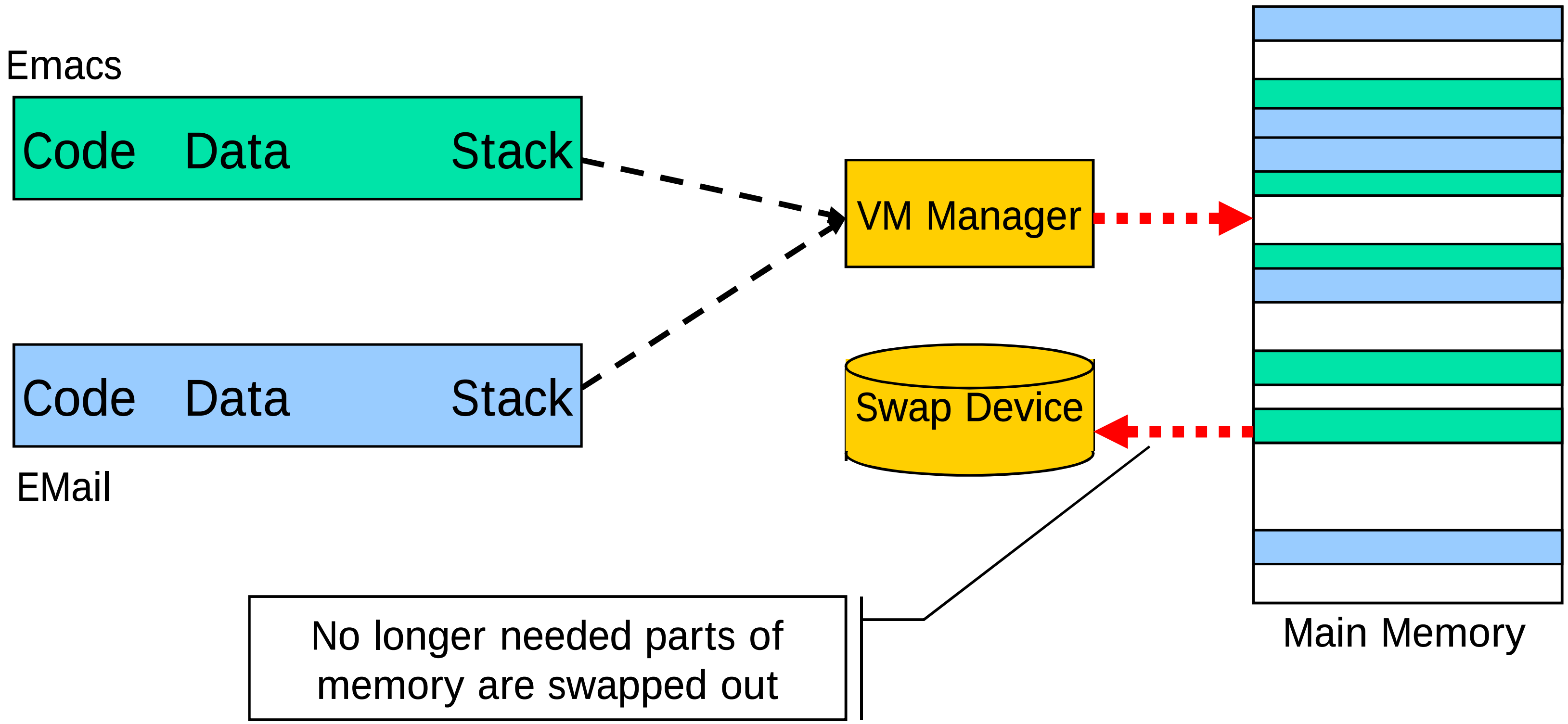
- Main (physical) memory (RAM) is limited
- Memory needs of all active tasks/processes can be larger than RAM
  - Already a Java applet might need some MBs
- Application programs do not want to know where they are located in RAM
  - Modern program code is relocatable, i.e., it can run anywhere in RAM

Virtual Memory

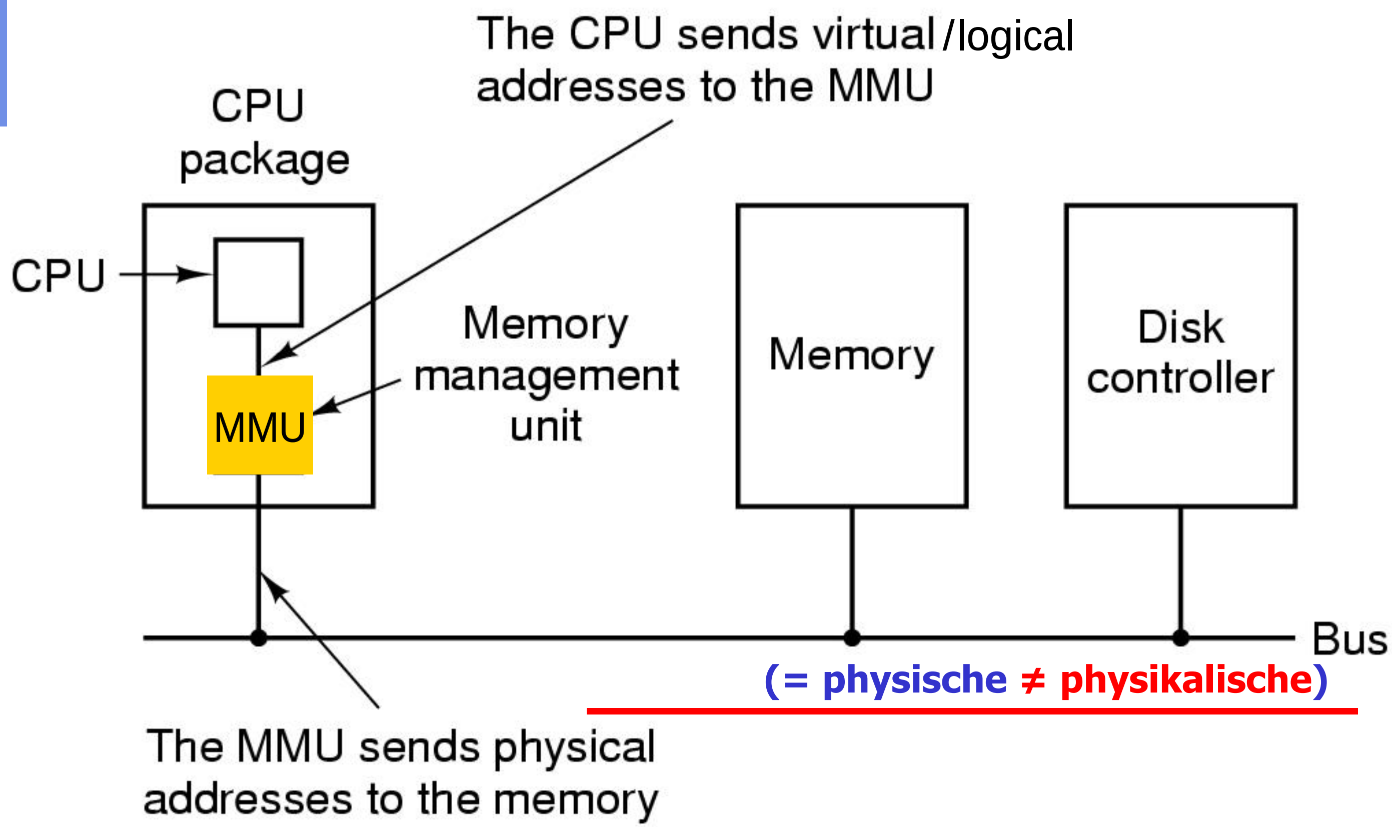
- Allows programmers to address memory in a reasonable fashion
  - Gives applications the illusion of having the total RAM for themselves
  - Address spaces (AS) are independent of each other, i.e. the same logical address in two different ASes is mapped to different locations in main memory
  - Automatic mapping of logical address space regions to appropriate physical memory portions
- Efficient virtual memory needs HW support

Virtual Memory

- Applications think they have a flat address space
- Physical memory is split into page frames
- AS regions do not have to be mapped to contiguous page frames
- However, ∃ specific regions that are mapped contiguously. *Why?*



Addressing Virtual Memory

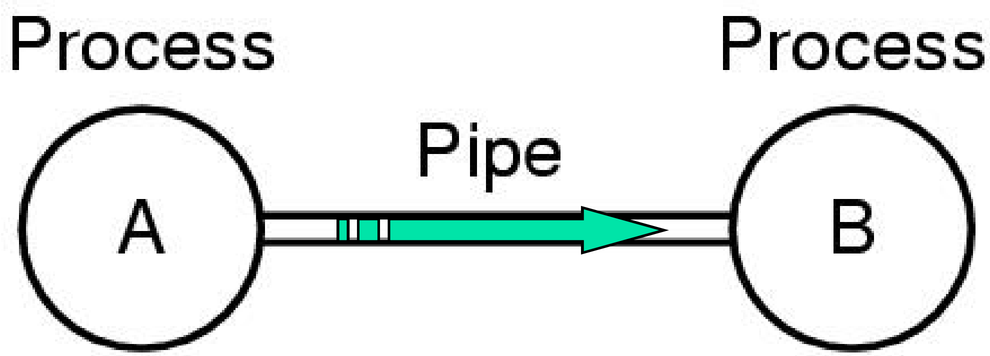


- Location and function of *MMU*

Scheduling

- Fairness
  - Give equal and fair access to all applications
- Differential responsiveness
  - Distinguish between different classes of jobs
    - Real-time processes versus interactive tasks
- Efficiency
  - Maximize throughput
  - Minimize response time
  - Accommodate as many users as possible

Communication (IPC)



A common example for IPC:
2 communicating UNIX processes A and B connected via a pipe

Semantics:
Process A stops writing to pipe when pipe is full
Process B stops reading from pipe when pipe is empty

I/O Device

- 3 major classes of I/O devices
  - Character devices
    - Serial port, keyboard, mouse
  - Block devices
    - Disks (IDE, SCSI)
    - CD-ROMs
    - Tape drives
  - Application specific devices
    - ...

File & Directory

- Implements long-term (persistent) storage
- Persistently stored data units, e.g.
  - files
  - directories
  - ...
- Traditional files
  - Accessed via specific system calls, e.g. **`read()`**
- Memory mapped files
  - Accessed like any other part of RAM

Protection and Security

- Access control
  - Regulate user access to the system as a whole or to individual system components (e.g. file system)

Only for files

- Information flow control
  - Regulate flow of information (data) within the system and its delivery to users
- Certification
  - Proving that access control & flow control perform according to the specifications of the system

Supporting Functions

- OS kernel executes “system calls”, i.e. basic software functions such as IPC
- Other high end supporting system code is not part of an OS, e.g.
  - Editor
  - Compiler
  - Assembler
  - Linker
  - Command interpreter (shell)

System Structure

Library

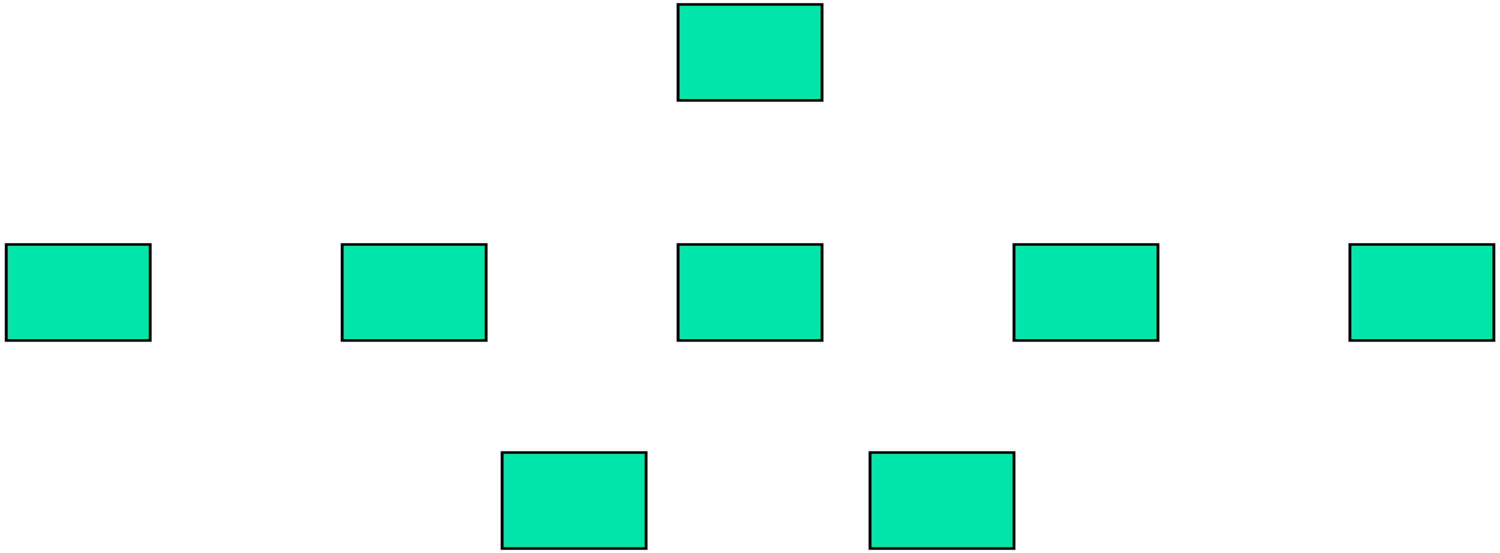
Kernel

System Call

Interfaces and “Virtual Machines”

System Structure (1)

Software System := set of components*



*Component based system = another buzzword in systems (meanwhile a bit outdated)

Potential System Components

- Applications
  - Simulating the traffic
  - Forecasting the weather
  - Editing a textbook, etc.

- OS subsystems (components) with a specific task
  - Initiating (e.g. bootstrap loader)
  - Controlling (e.g. shell)
  - Protecting (e.g. firewall)
  - Accounting (e.g. monitor)
  - Servicing (e.g. file server)

- Basic Functions (e.g. synchronization)

Potential System Components

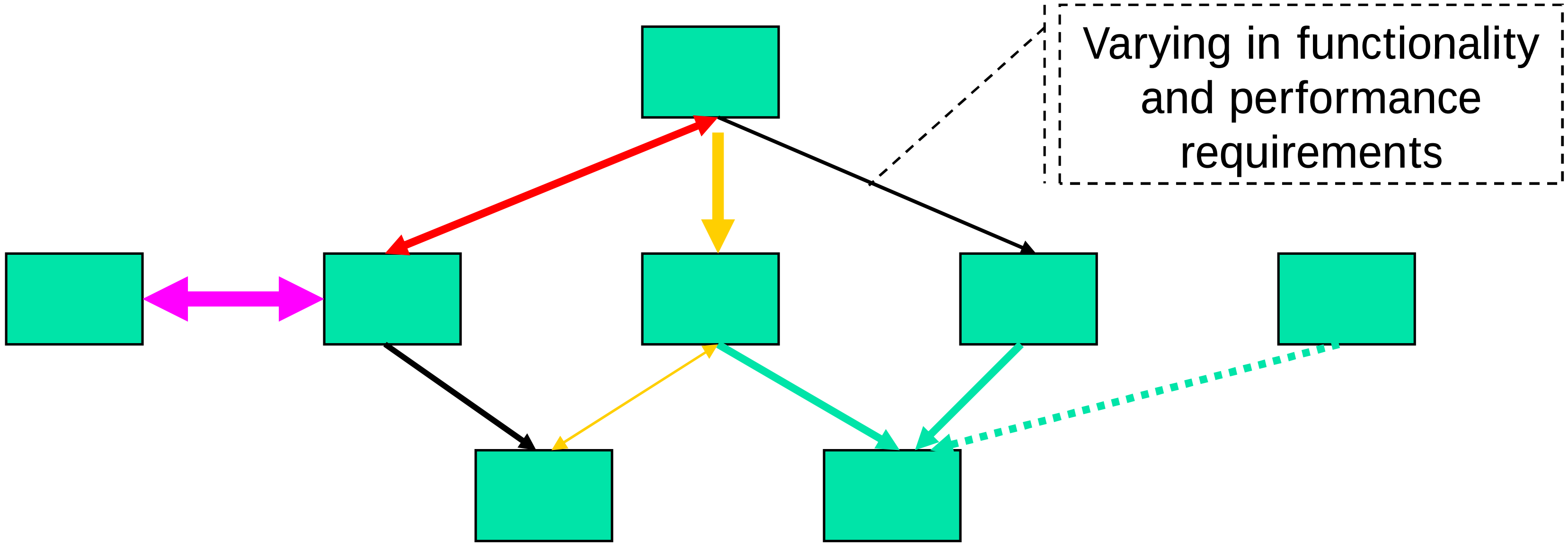
Component	Objects	Example Operation
GUI/shell	button, window	execute shell script, ...
Application	a.out	quit, kill, ...
File System	directories, files	open, close, read,
Devices	printer, display	open, write, ...
Communication	ports, channels	send, receive, ...
Virtual Memory	segments, pages	write, fetch
Secondary Storage	chunks, blocks	allocate, free,
Task	task queue	exit, create, ...
Process/Thread	ready queue, PCB/TCB	wakeup, execute, ...
Interrupts	interrupt handler	invoke, mask, ...

Unix System Software

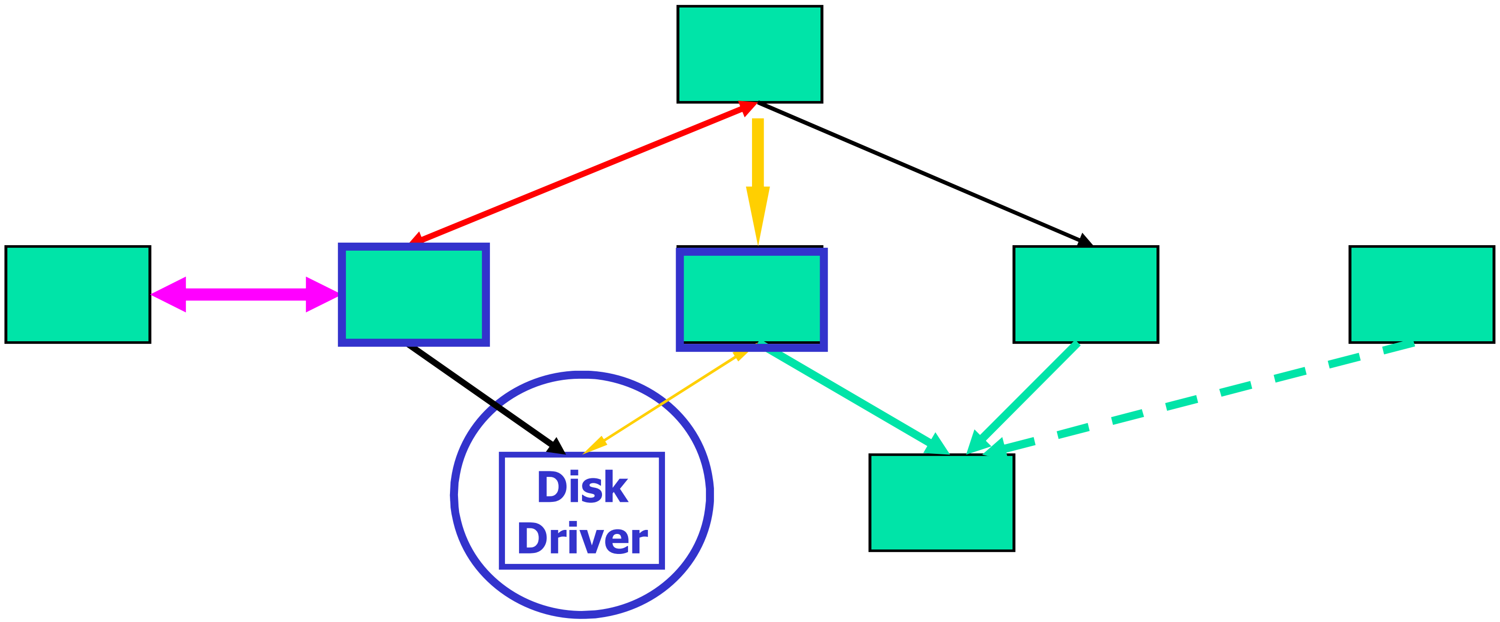
ar	build & maintain archives
cat	concatenate files → standard out
cc	compile C program
chmod	change protection mode
cp	copy file
echo	print argument
grep	file search including a pattern
kill	send a signal to a process
ln	link a file
lp	print a file
ls	list files and directories
mv	move a file
sh	start a user shell
tee	copy standard in to standard out and to a file
wc	word count

System *Structure* (2)

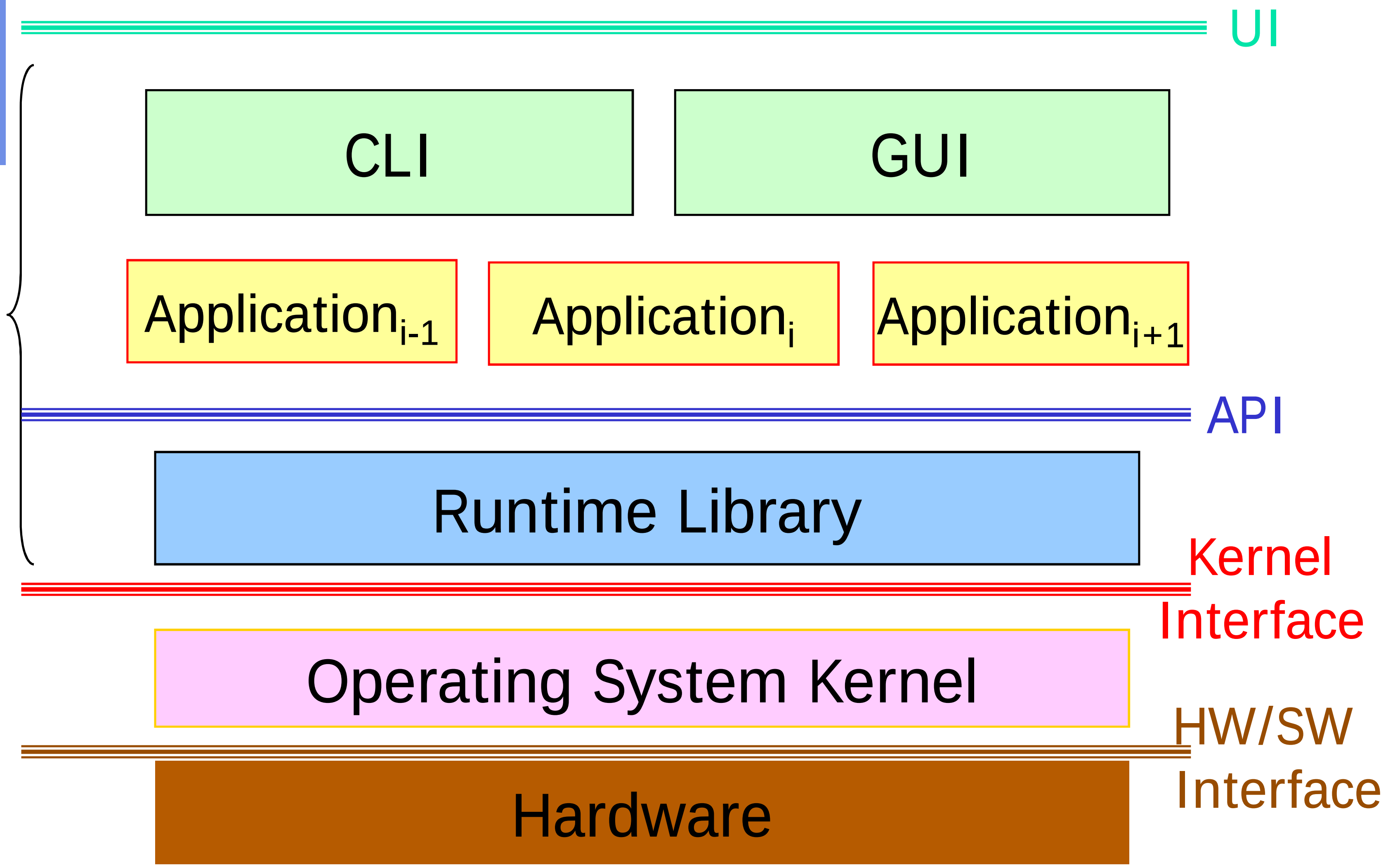
Software System := set of components & their interconnections with various *interdependencies*



Design & Implementation Problems



Major Interfaces of a System



User Interface (UI)

CLI	GUI

CLI allows direct command input

- Sometimes implemented in kernel, sometimes by system processes outside the kernel
- Sometimes multiple flavors implemented, e.g. in Unix ∃ different shells (**sh, bsh, csh, ksh,** …)
- Shell fetches a command from user, interprets, and executes it
  - Sometimes commands are built-ins
  - Sometimes just names of executable files
    - In the latter case adding new features does not require a complete modification of the shell
    - You only have to add another case to the central switch statement

User Interface (UI)

CLI	GUI

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc.
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
  - Invented at Xerox PARC
- Today's systems include both CLI & GUI interfaces
  - Microsoft Windows is GUI with CLI "command" shell
  - Apple Mac OS X is "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Solaris offers a CLI with optional GUI interfaces (Java Desktop, KDE)

Applic. Program Interface (API)

- ∃ two programming interfaces to the services provided by the OS kernel
  - Kernel interface, i.e., a list of system calls
  - API, typically written in C or C++
- Three common APIs are:
  - Win32 API for Windows
  - POSIX API (offered by virtually all versions of UNIX, Linux, and Mac OS X)
  - Java API for the Java virtual machine (JVM)

Example1: Win32 Standard API

- Consider the **ReadFile()** function in the Win32 API

return value

```
BOOL   ReadFile c  (HANDLE        file,
                    LPVOID        buffer,
                    DWORD         bytes To Read,
                    LPDWORD       bytes Read,
                    LPOVERLAPPED  ovl);
```

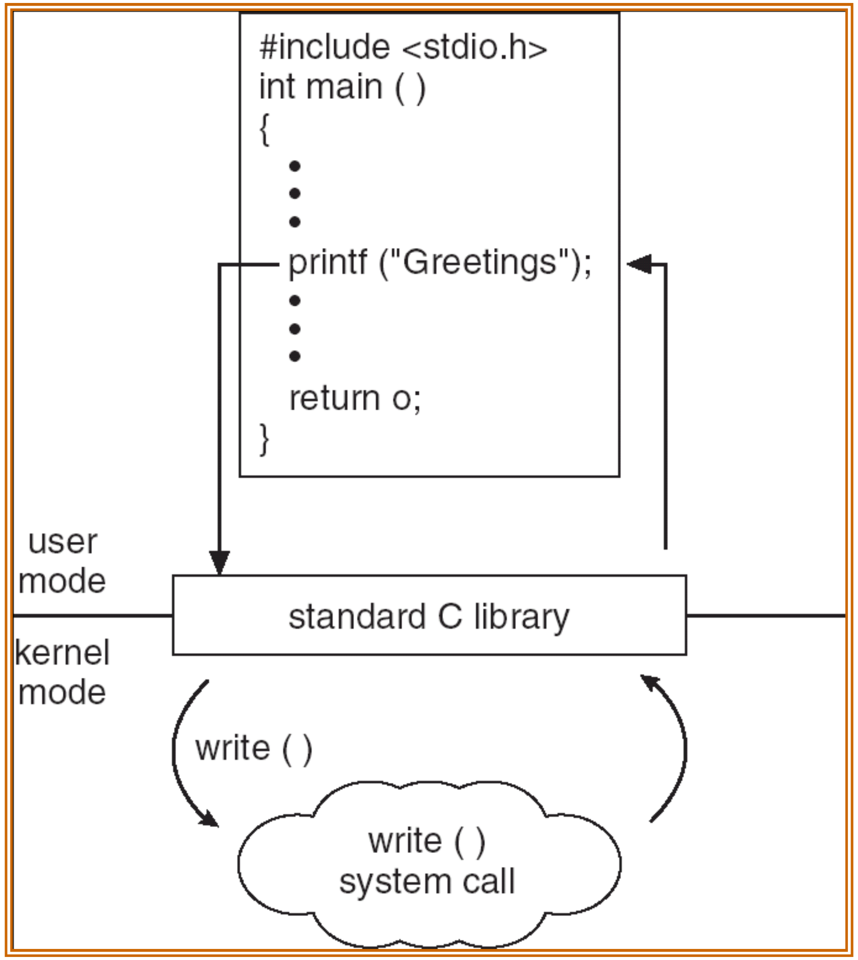
function name

parameters

- A description of the parameters passed to **ReadFile()**
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be stored
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

Example2: Standard C Library

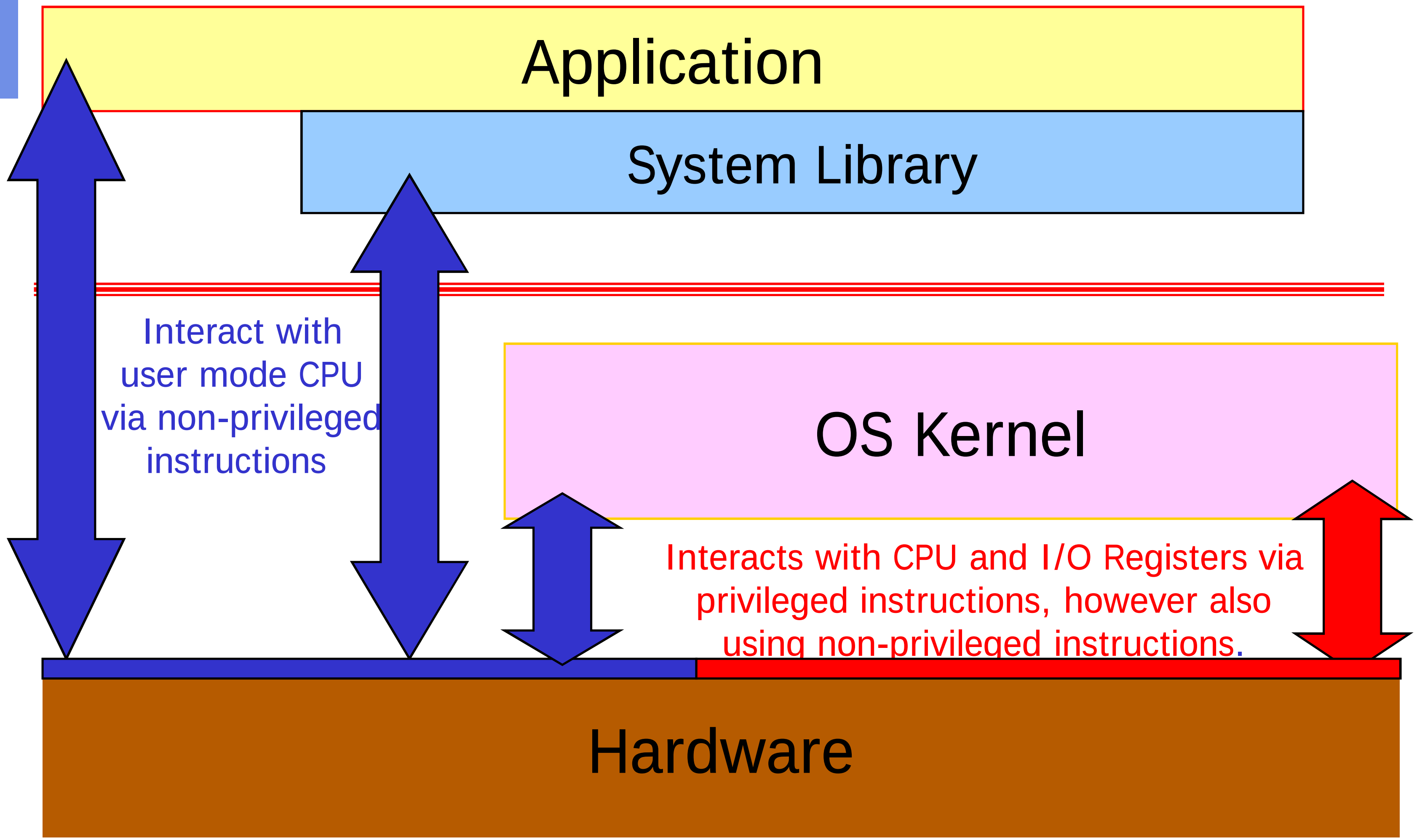
- C program invoking **`printf()`** library call, which performs **`write()`** system call



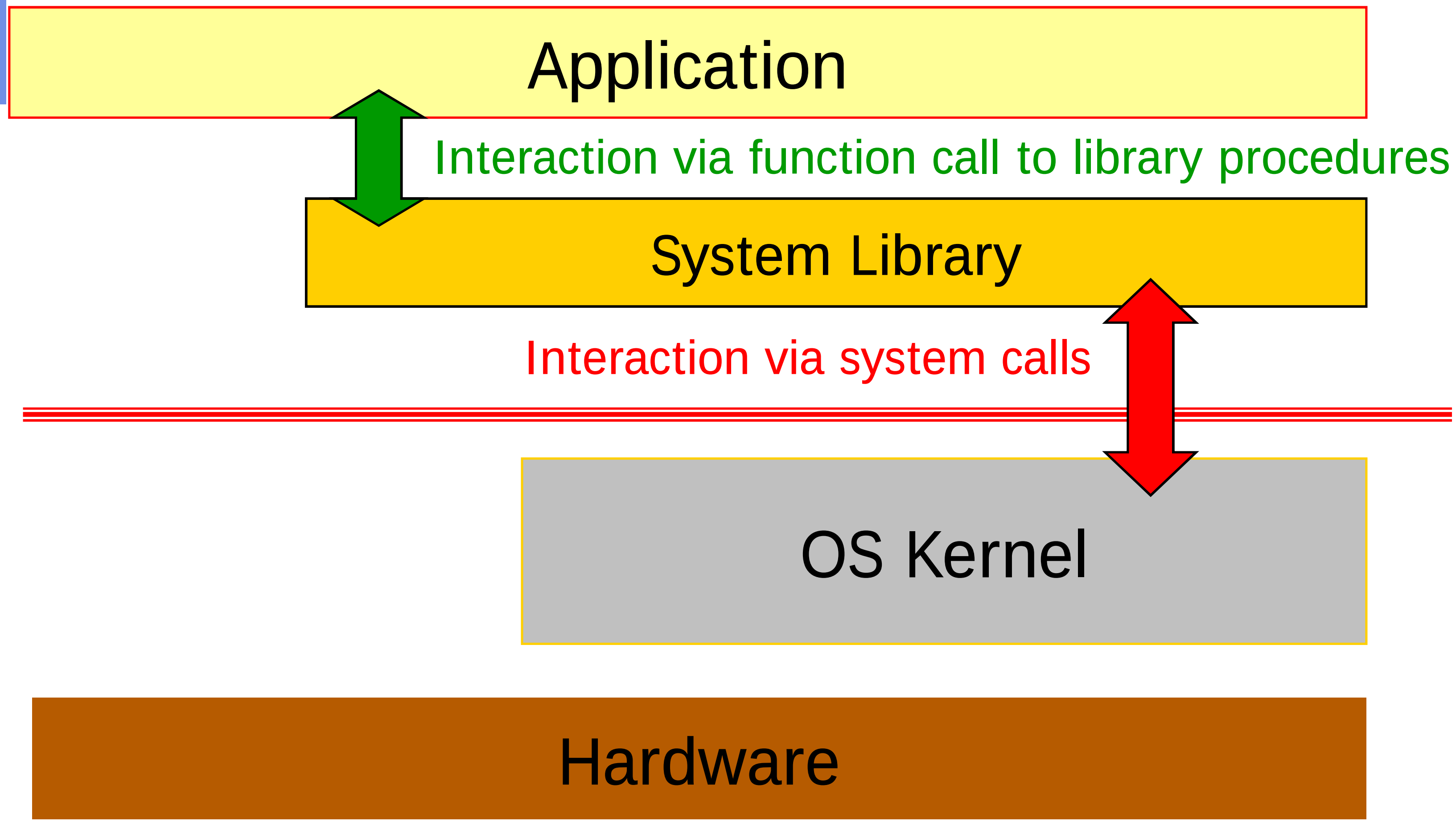
Protection

- Challenge: OS must support multiple protection domains
  - OS acts as “law enforcement” (~ role of police)
- Goals
  - Buggy applications cannot crash the system
  - Malicious applications cannot take control
  - User data is protected from “non trusted” users or programs

Interaction of System Components



Interaction of System Components



System Libraries

- In most OSes ∃ different system libraries supporting
  - programming languages (e.g., C library)
  - graphics
  - mathematics
- **Not** every library function implies a system call
  - `strcmp()`, `memcpy()` are pure user-level functions
  - Mathematical functions are often pure user-level functions
- `fopen()`, `fscanf()` et al. imply system calls

Traditional Kernel

- All kernel programs run in privileged mode
- Often the complete kernel is resident in RAM
- Kernel contains basic functions
  - whatever is required to offer services
  - whatever is required to provide security
  - ...
- Also called Nucleus, Monitor*, Supervisor, ...

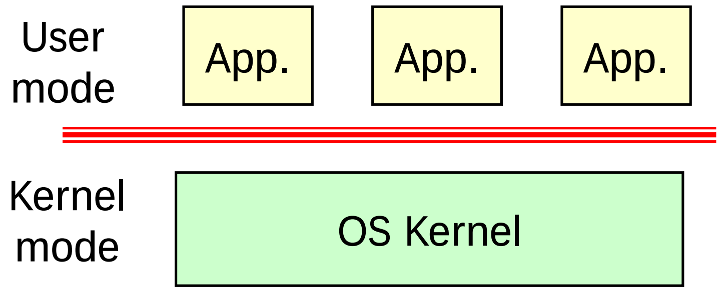
* Computer scientists like this term

Triggering the Kernel

- Before a kernel program can run on a CPU, it must be triggered by some event, e.g. by
  - a system call
  - an exception
  - an interrupt

User/Kernel Boundary

- Implemented in HW
- Allows the OS to execute privileged instructions
- Applications enter kernel by executing a system call



User Mode versus Kernel Mode

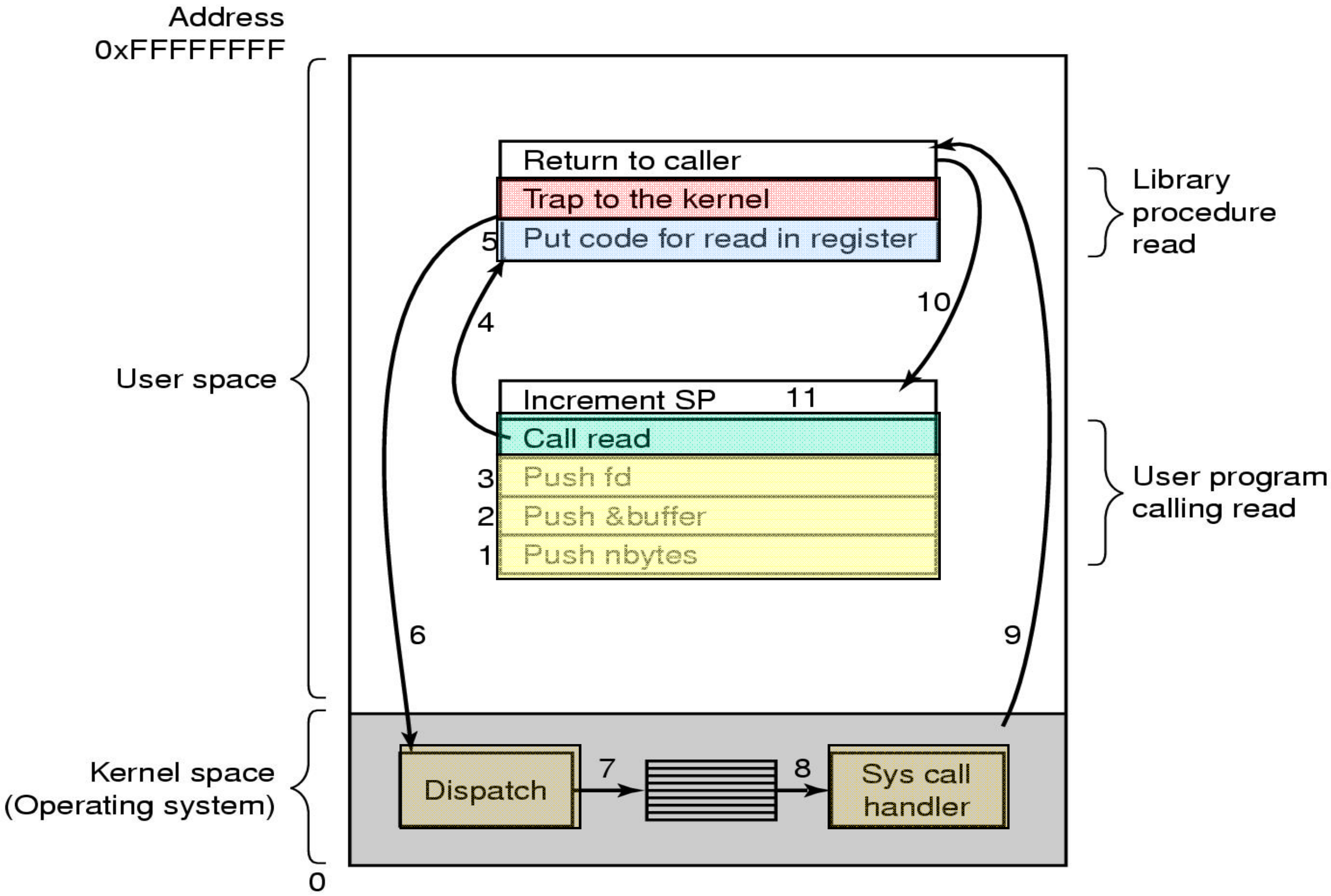
- Only the kernel can *execute privileged* instructions, i.e. if an application tries to execute a privileged instruction, CPU raises an exception
- Examples of privileged instructions
  - access to I/O registers
    - poll for I/O, perform DMA, catch HW interrupt
  - manipulate MMU and memory states
    - set up page tables, load/flush TLB, etc.
  - configure various “mode bits”
    - interrupt priority level, software trap vector, etc.
  - call HALT instruction
    - put CPU into low-power/idle state until next HW interrupt
- A system call is one way to enter the kernel

System Call Overview

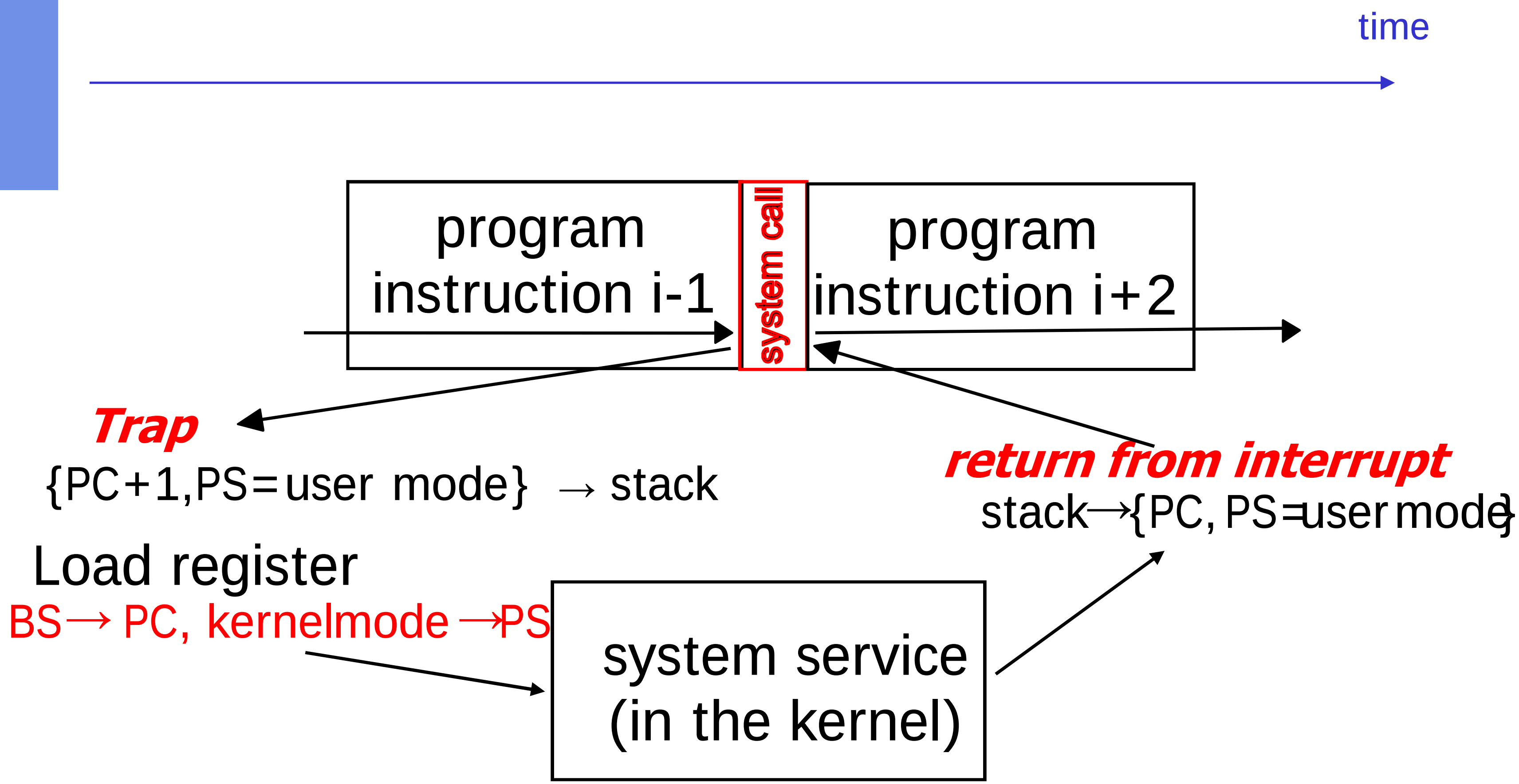
- Application invokes a helper procedure (e.g. a library function)
  - `read, write, gettimeofday, …`
- Helper passes control to the OS
  - Indicates the system call number
  - Loads arguments into “registers”
  - Issues a trap (software interrupt)
- OS saves user state (registers)
- OS invokes appropriate system call handler
- OS returns control to the user application

Trigger Example 1: System Call

```
count = read(fd, buffer, nbytes)
```



System Call at Instruction i



System Calls: Traps & Interrupts

- Synchronous indirect method invocation ***(Trap)***

```
...
Move A,R1
Trap 7
Move R1, A
```

interrupt vector table

RAM-Address	
0017	PS für ISR 8
0016	Address of ISR 8
0015	**PS für ISR 7**
0014	**Address of ISR 7**
0013	PS für ISR 6
0012	Address of ISR 6

- Asynchronous HW-**Interrupt**-Signal 7

ISR = Interrupt Service Routine ~ driver

PS = Processor Status Word (prio, mode,..)

Parameter Passing

- Often, more information than just the system call number is needed
  - type & amount of info vary according to system call and OS
- 3 general methods used to pass multiple parameters
  - Pass the parameters in registers
    - Often you have more parameters than you have registers
  - Parameters stored in a block in memory, and the address of the block is passed as a parameter in a register
  - Parameters are pushed to the stack by the calling program and popped by the kernel
- Both, block and stack methods do not limit the number or the length of the parameters

Insecure System Call

- Consider a hypothetical system call **`zeroFill()`**, which fills a user buffer with zeroes
  `zeroFill(char* buffer, int bufferSize)`
- The following kernel implementation of **`zeroFill`** contains a security flaw. *What is the vulnerability, and how would you fix it?*

```
void zeroFill(char* buffer, int bufferSize){
    for (int i=0; i < bufferSize; i++){
            buffer[i] = 0;
}}
```

Summary: System Call

- Kernel must verify the parameters
- *How does application pass data to the kernel?*
  - Example: **`write()`** passes in a pointer to a buffer to be written to a file
- *How does the kernel return kernel state to the application?*
  - Example: **`read()`** returns an **`int`** indicating the number of bytes actually read

How to read from a file?

Compare `read( )` and `fread( )`

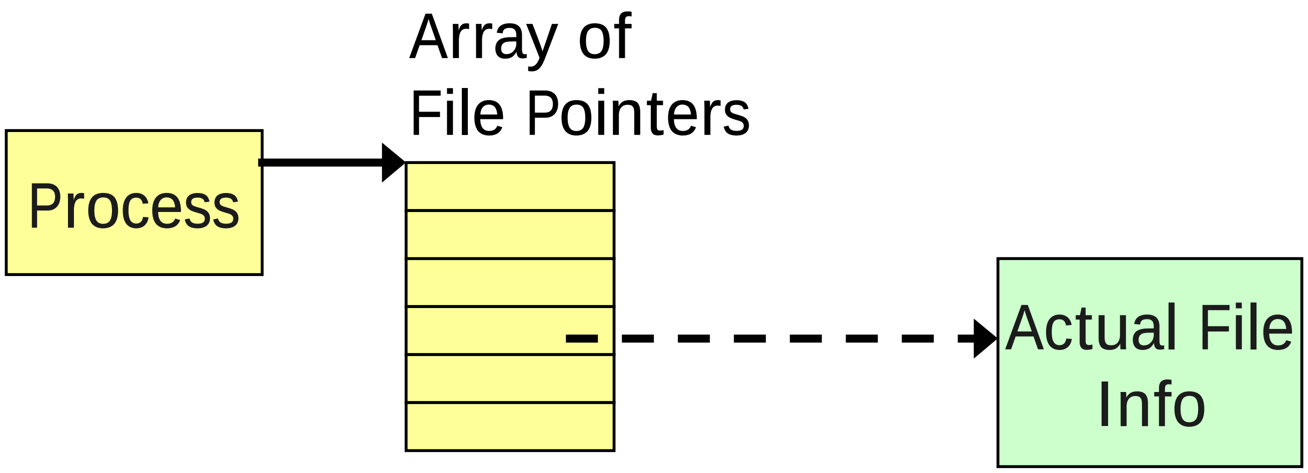
- `read(int d, void *buf, size_t nbytes)`

  `read( )` attempts to read `nbytes` of data from the object referenced by descriptor `d` into the buffer `buf`.

- `fread(void *ptr, size_t size,
        size_t nmemb, FILE *stream)`

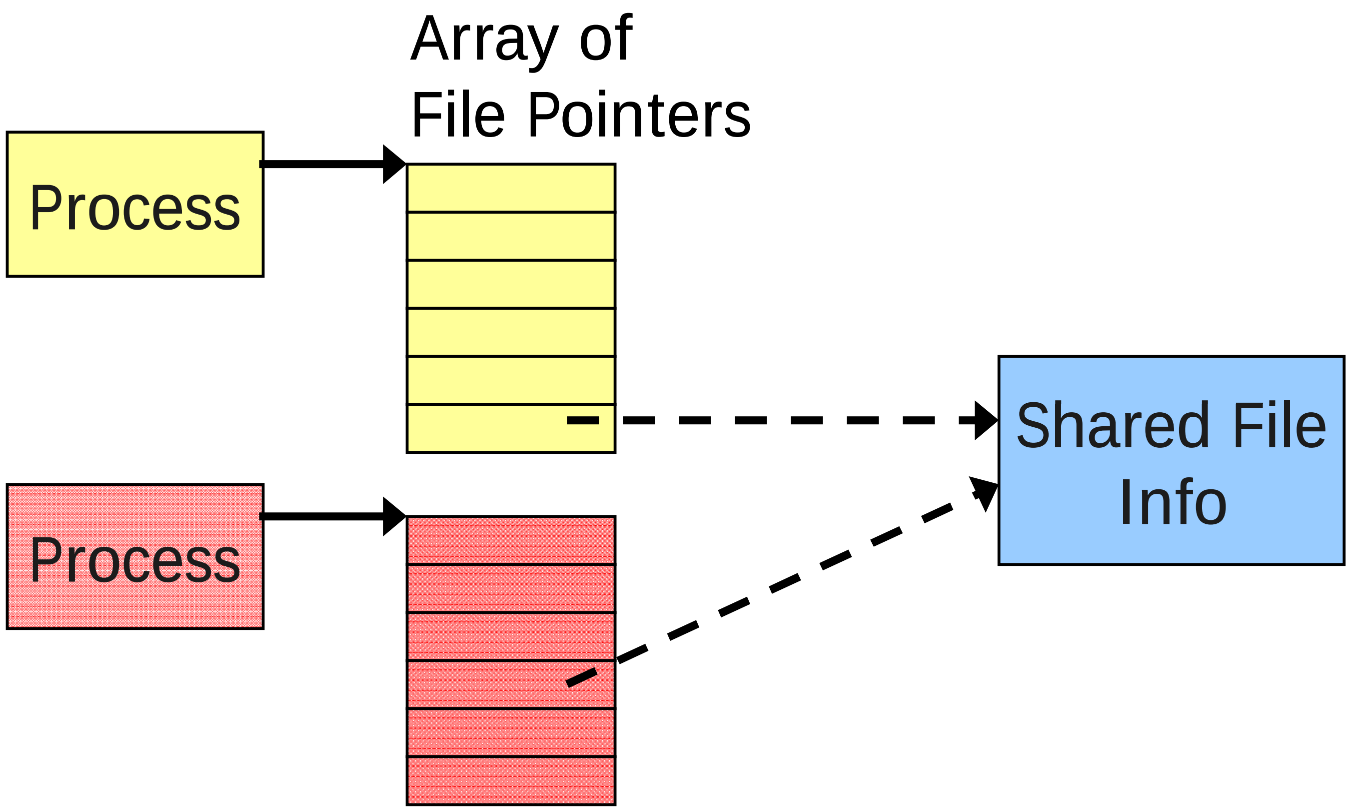
  `fread( )` reads `nmemb` objects, each `size` bytes long, from the stream pointed to by `stream`, storing them at the location given by `ptr.`

First Insight into OS Concepts (1)



What do we gain from this?

First Insight into OS Concepts (2)



Each process can have a different file pointer to a shared file.

System Calls for Processes

Process Management	
Call	Description
pid = fork()	Create child process
waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate execution + return status

Hint: We expect that you will be familiar with the POSIX System Calls at the end of this course.

See: http://www.opengroup.org/onlinepubs/7908799/xshix.html

System Calls for Files

File Management	
Call	Description
fd = open(file, how, ...)	Open file for reading, writing, ...
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position= lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get the file's status information

System Calls for Directories

Directory Management	
Call	**Description**
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create new entry name2 → name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

System Calls for Miscellaneous

Miscellaneous Management	
Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get elapsed time since Jan. 1, 1970

Unix versus Win32 System Calls

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Trigger Example 2: Time Slice IR

- Timer to interrupt infinite loops (avoids that a process can hog the CPU)
  - Set timer interrupt after specific period of time
  - When counter = zero, the timer unit generates a timer interrupt

Interface: An Example

Draw a rectangle of length dx and width dy.

DrawRectangle(float dx, float dy)

Method: DrawRectangle

Data: float dx, float dy

Protocol:

- initialize module „graphics“
- set scales
- set origin
- draw rectangle

uses method drawLine with data x0, y0, x1, y1

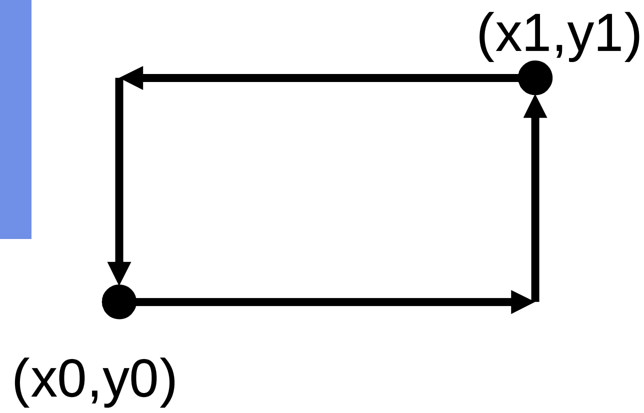
Interfaces: A Generalization

An interface consists of

- provided data and functions or methods (in OOD)
- protocols for usage of functions and data, with which the object has to do some service (export interface)
- required data, functions, and protocols for use by the module to deliver its services (import interface)

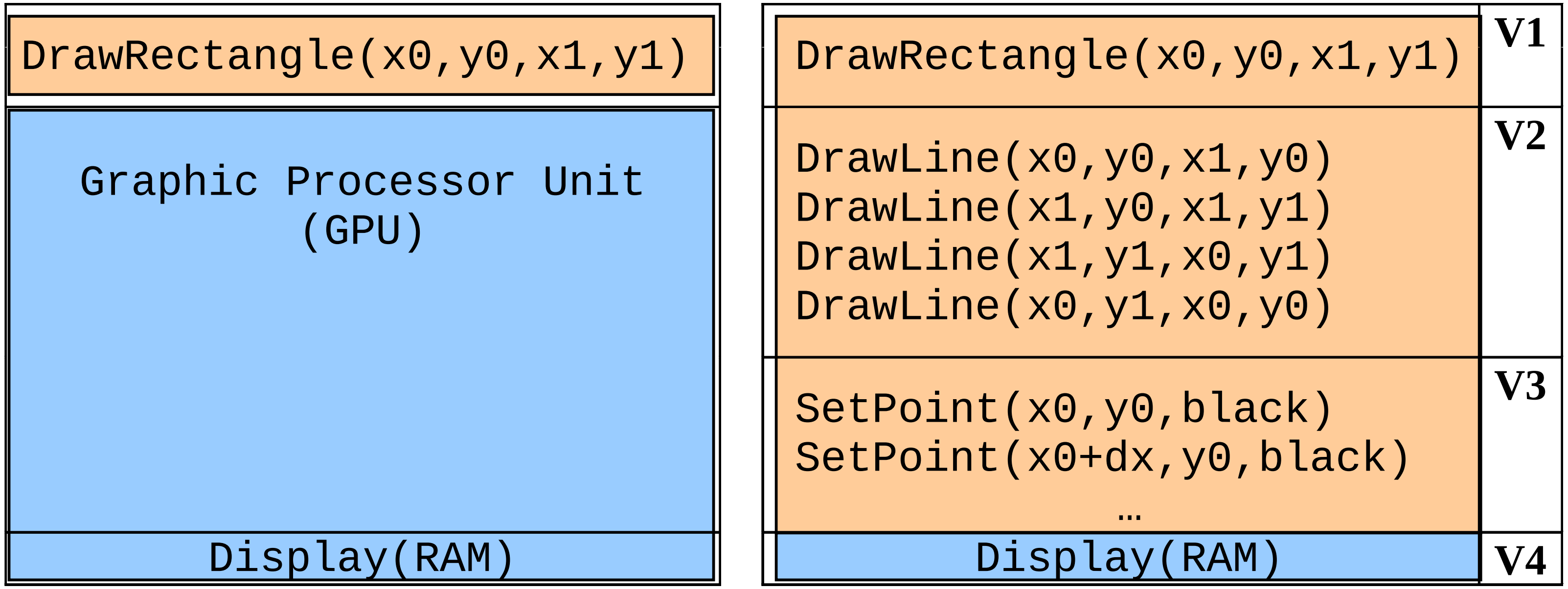
→ Virtual Machines

Virtual Machines: An Example

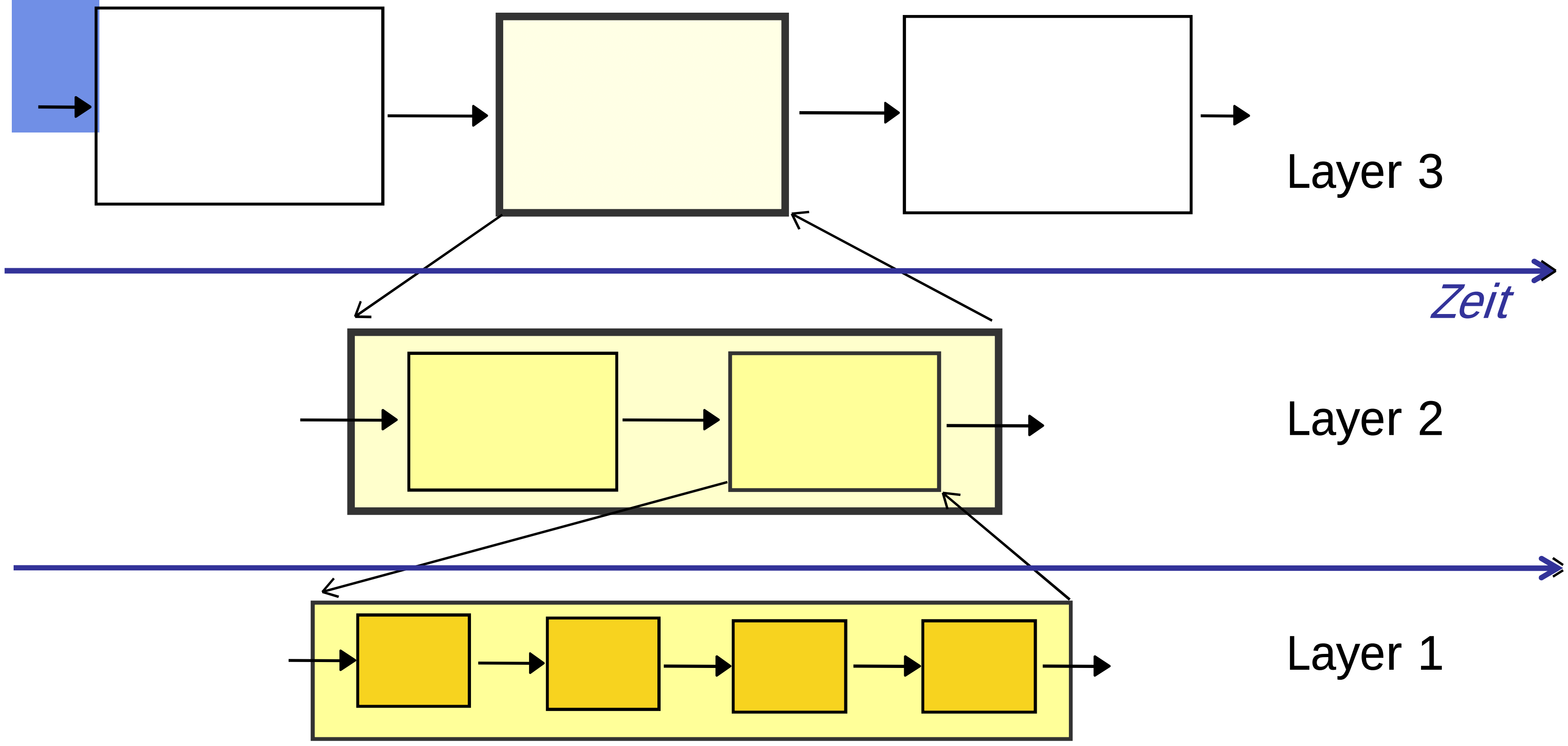


Draw a rectangle

DrawRectangle(x0,y0,x1,y1)



Virtual Machines: Idea



Virtual Processor: An Example

- Software-Hardware-Migration
- Via virtual CPU

Program in Java-Code
Java-Code / Native Code
CPU- Hardware

Program in Java-Code
Microcode- and CPU-Hardware

Virtual, Logical, Physical Devices

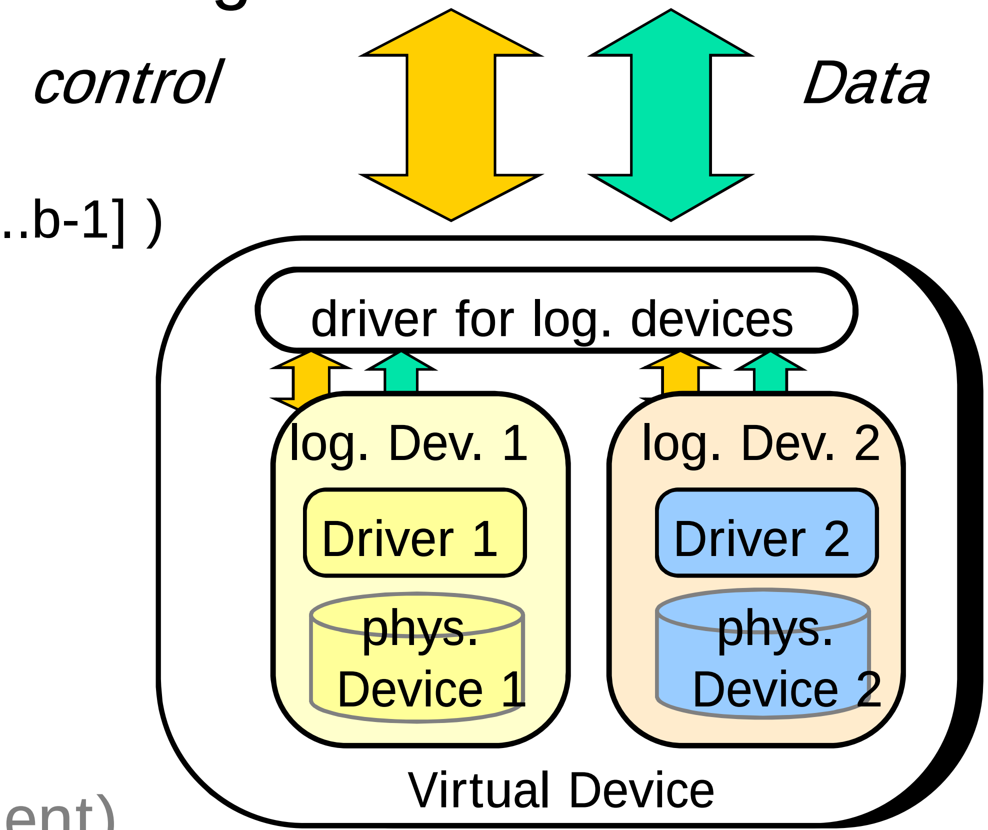
Example: Virtual Disk Storage

- Logical Device (block[0...b-1])

  = physical device & hardware driver

- Virtual device

  = logical device & logical driver (storage management)



Virtual Mass Storage: An Example

Storage Area Network SAN | *asymmetric pooling*