Lecture 2

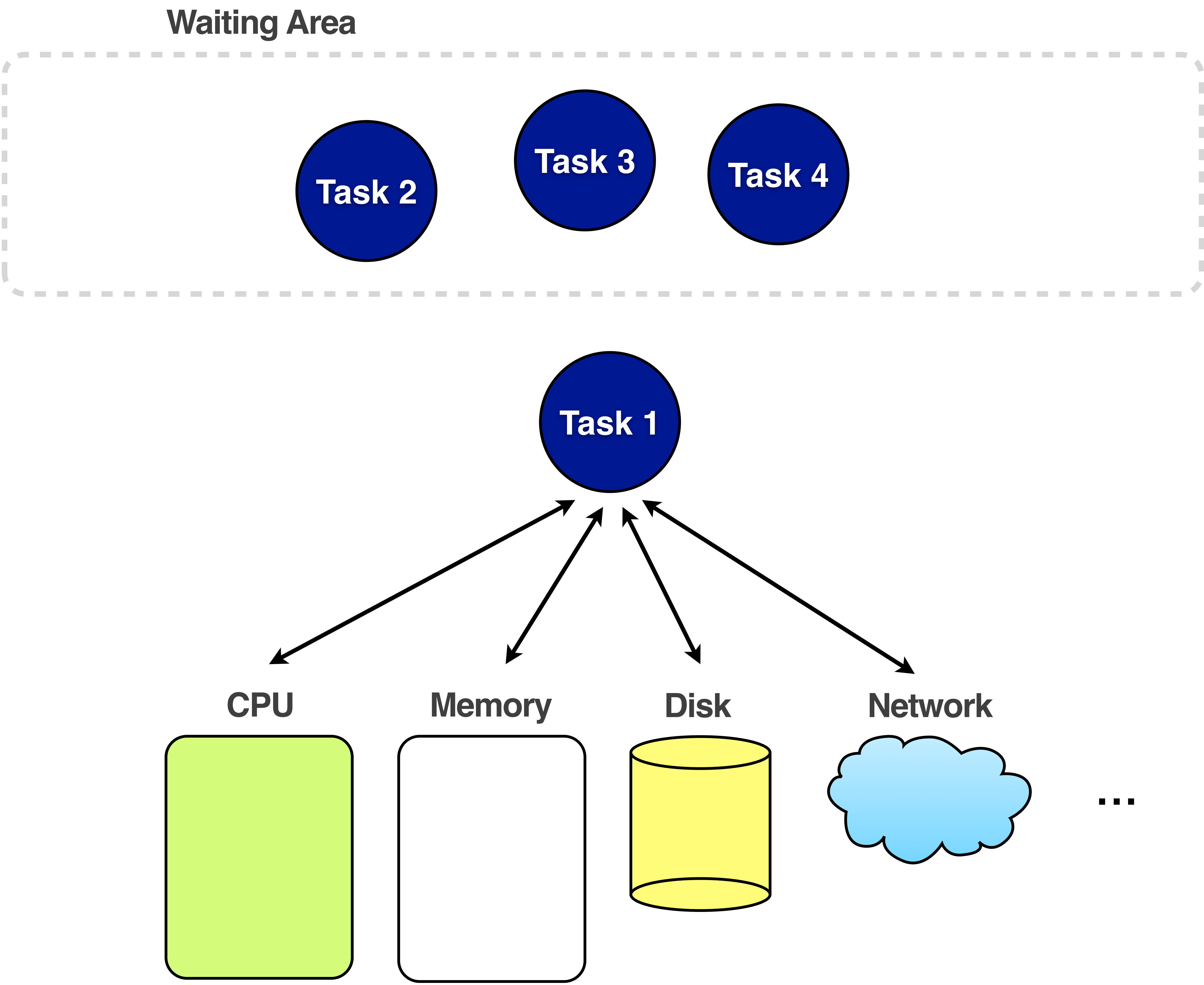
Abstractions and Interfaces

19 August, 2011

Overview of Basic Concepts in OS

process

Recall:
Time-Sharing



Waiting Area

Task 2

Task 3

Task 4

Task 1

CPU

Memory

Disk

Network

...

what do you need to remember in order to **resume** a task?

process:
code + data +
context of execution

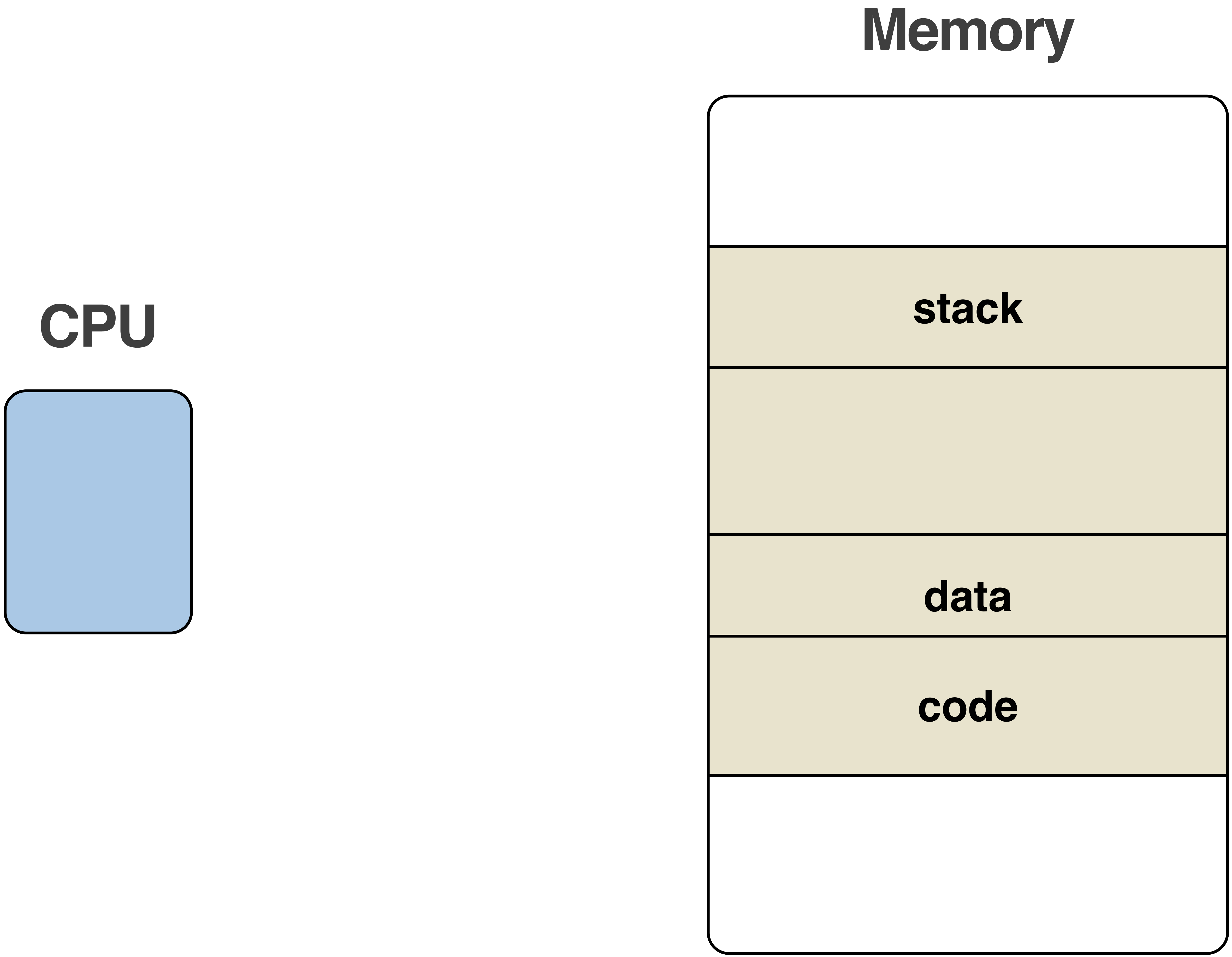
address space

how to prevent one process from corrupting the memory of another process?

every process has its own
address space

a process may occupy different physical memory location at different time but the executable code remains the same.

(e.g., for instruction:
load from address X to register Y
what should X be?)



Memory

CPU

stack

data

code

file and directory

Data on disks are organized into **files** and **directories** block special files

character special files

in

/dev

pipe

shell

User interfaces to OS:

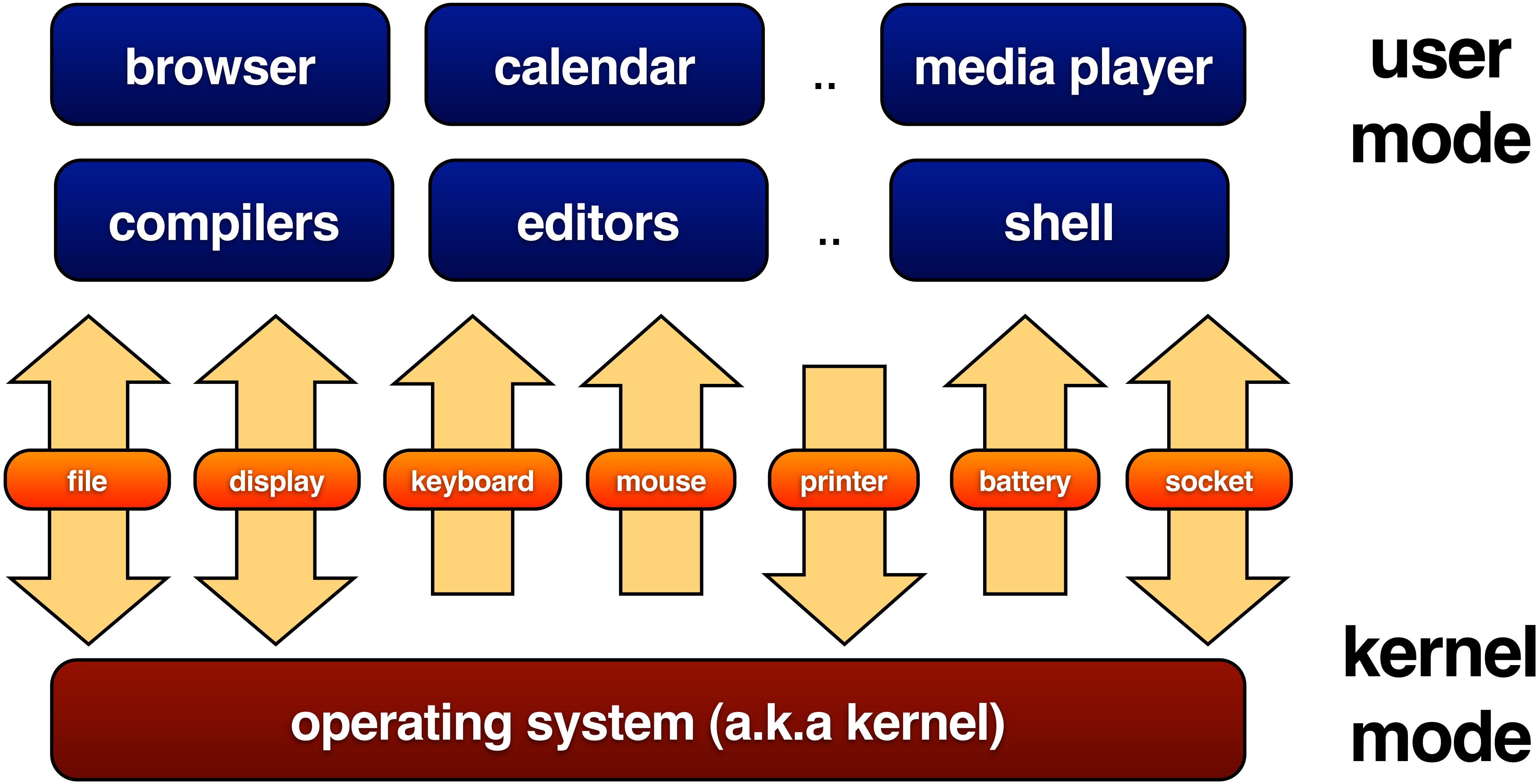
1. shell
2. window systems

process

address space

files and directories

shell



browser

calendar

..

media player

compilers

editors

..

shell

file

display

keyboard

mouse

printer

battery

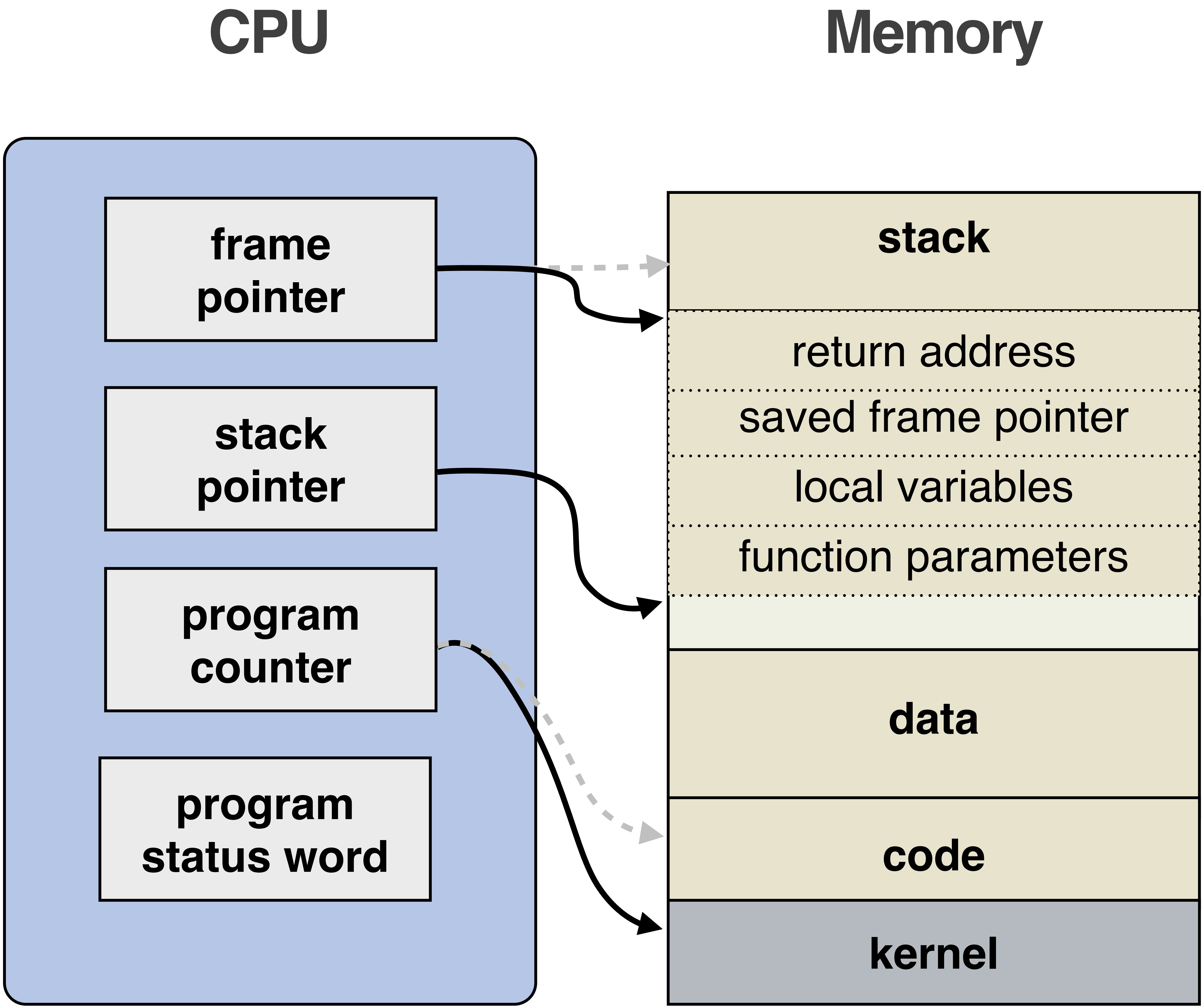
socket

operating system (a.k.a kernel)

user mode

kernel mode

Interfaces provided by OS
are known as **system calls.**



CPU

Memory

frame pointer

stack pointer

program counter

program status word

stack

return address

saved frame pointer

local variables

function parameters

data

code

kernel

Invoking system calls involved:

1. a common setup/cleanup routine

2. actual doing the work of the system call

system call setup:

1. switch to kernel mode (PSW bit)
2. save context of current process
3. pass arguments to kernel
4. determine which system call service routine to call

system call clean/up:

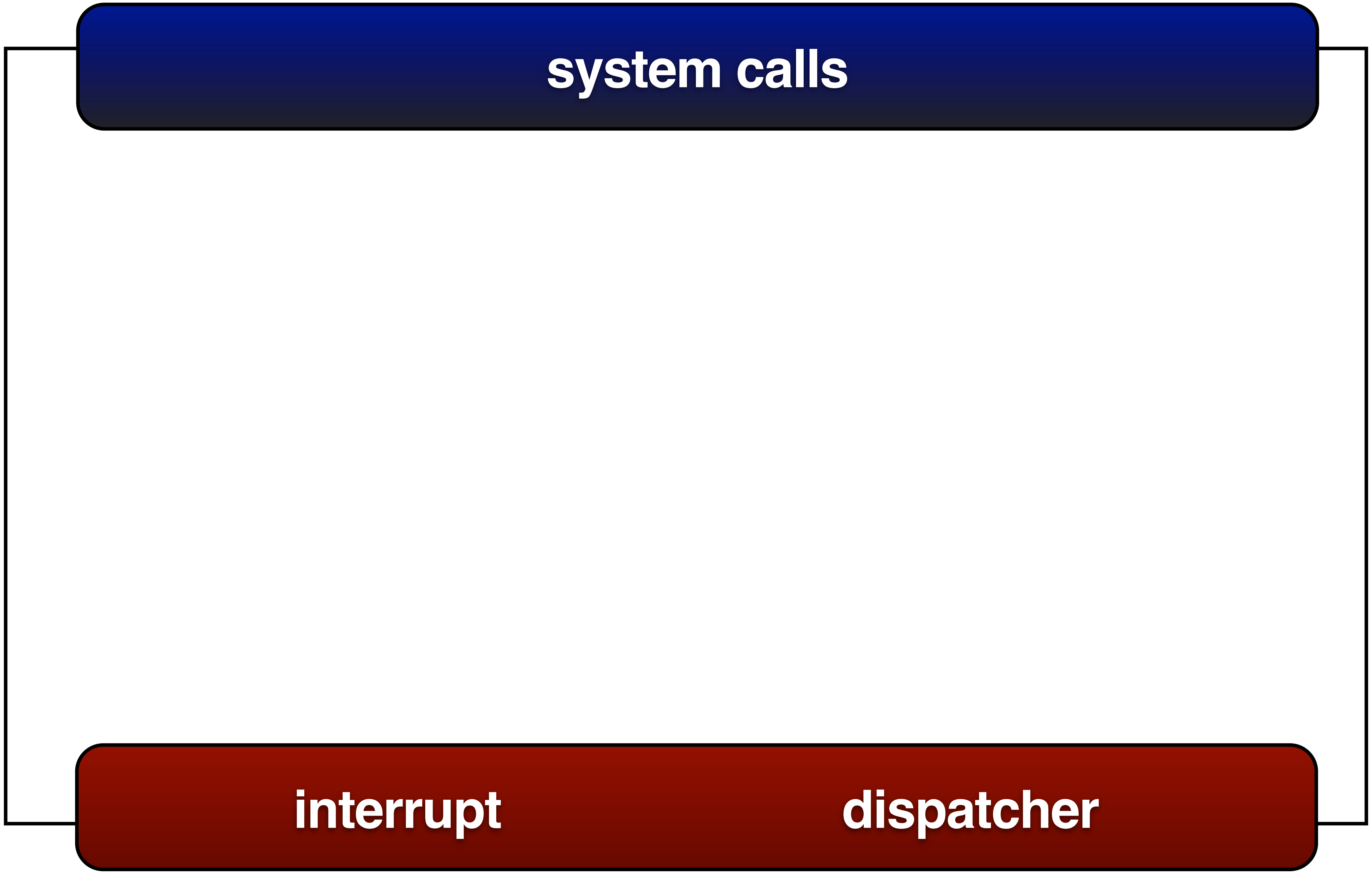
1. indicate error code, if any
2. check if need to switch to another process
3. restore process context
4. return to user mode

System calls are typically triggered by an interrupt instruction (e.g., TRAP or INT)

Structure of OSes

Structure of Linux Kernel

user program

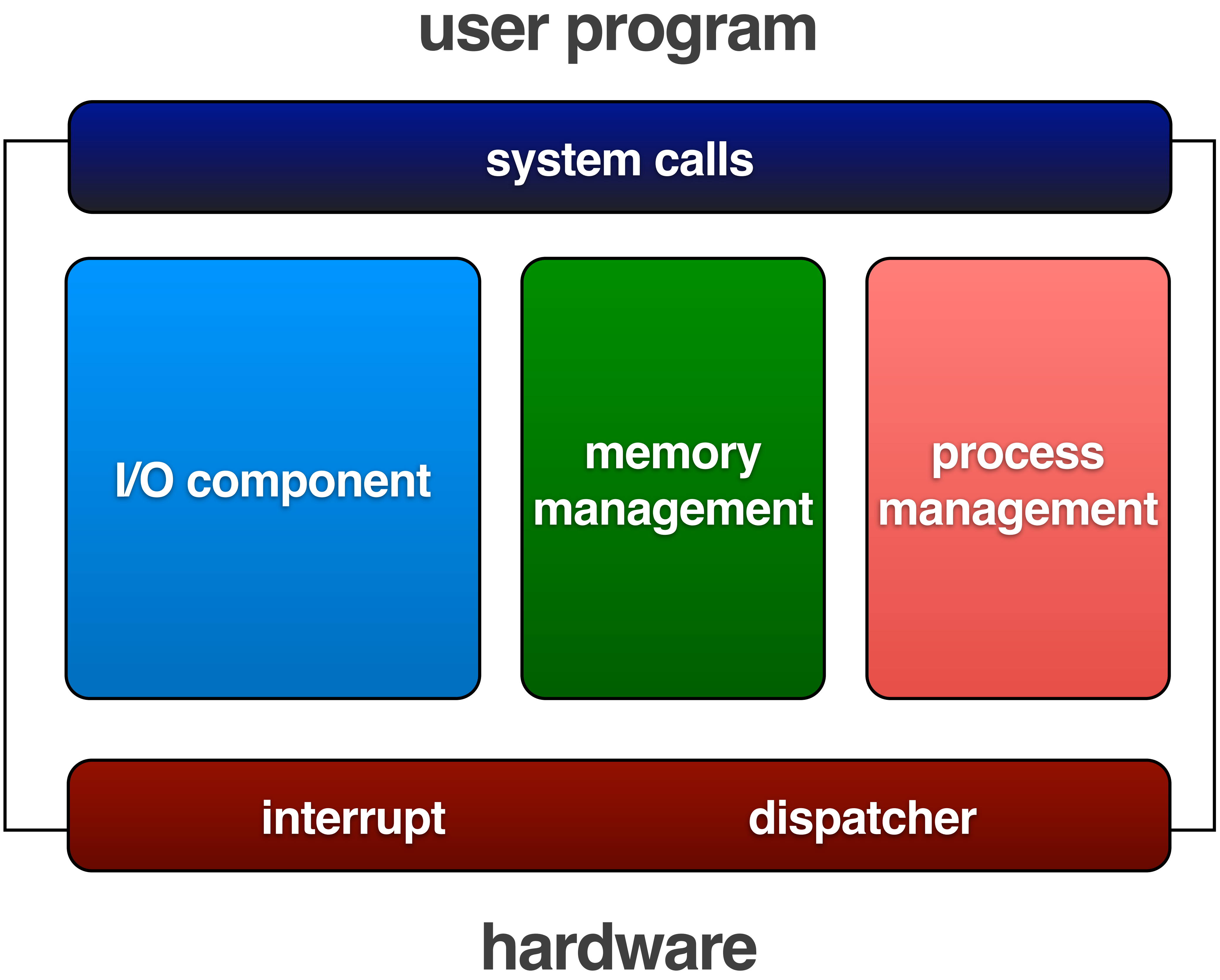


system calls

interrupt

dispatcher

hardware



user program

system calls

I/O component

memory management

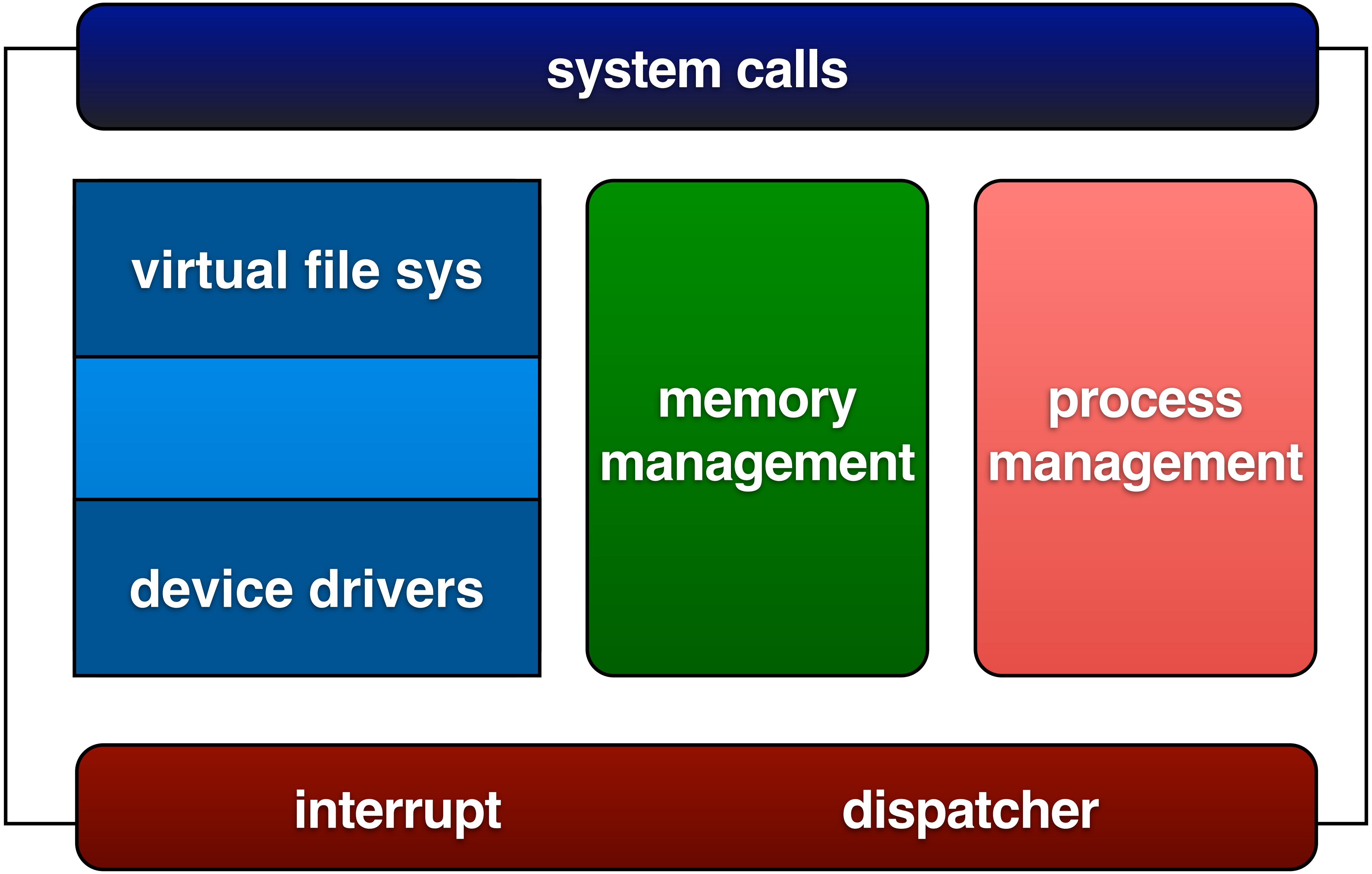
process management

interrupt

dispatcher

hardware

user program



system calls

virtual file sys

device drivers

memory management

process management

interrupt

dispatcher

hardware

such design is called **monolithic kernel**

dynamically loadable kernel module

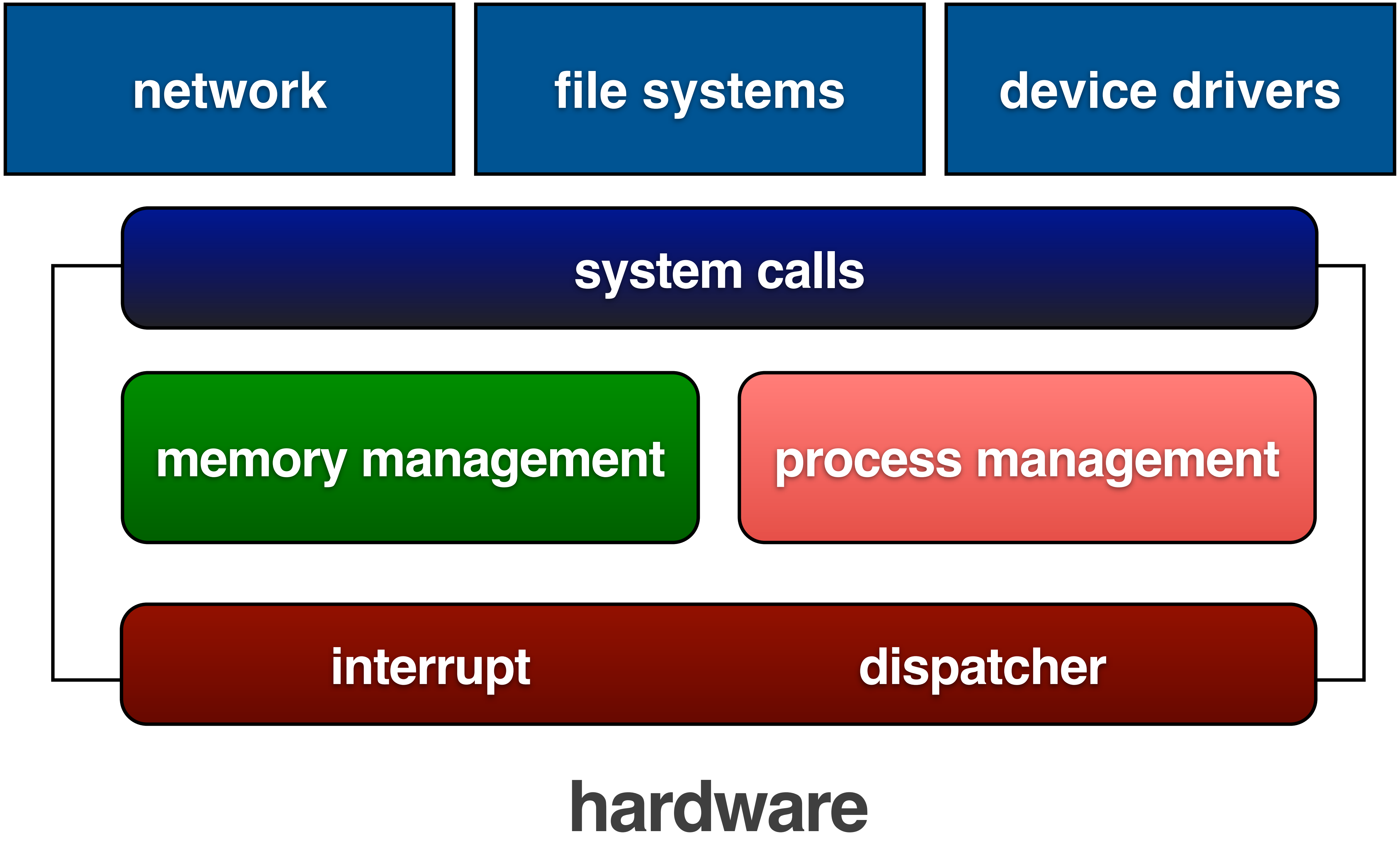
keeps the kernel small, extensible

(also in MS Windows, Mac OS X)

Alternative to monolithic design:

microkernel

An example microkernel architecture



network

file systems

device drivers

system calls

memory management

process management

interrupt

dispatcher

hardware

keep minimal functions in kernel

example:

Microsoft
Copyright © 1985-1996
Microsoft Corporation
Microsoft
Windows NT
Workstation 4.0
with Microsoft Internet Explorer
This product is protected by US and international copyright laws as described in the About Box.

A brief introduction to

C

C vs Java

(highlights)

Java: set of classes

C: set of functions + structures

C:

no `byte` datatype

no `boolean` datatype

no `String` class

(use `char`, `int`, and array of `char` respectively)

C:

no “`new`” operator

must explicit declare as pointer for reference variables

must explicitly `malloc()` and `free()` memory

Java: all variables are reference except boolean and numeric types

C: all variables are primitive types (holds the value of that exact type)

Java: external classes must be imported

C: external functions and types must be declared

(made easy with `#include` statement)

```
int main()
{
  return 0;
}
```

```
#include <stdio.h>

int main()
{
   printf(“Hello World!\n”);
   return 0;
}
```

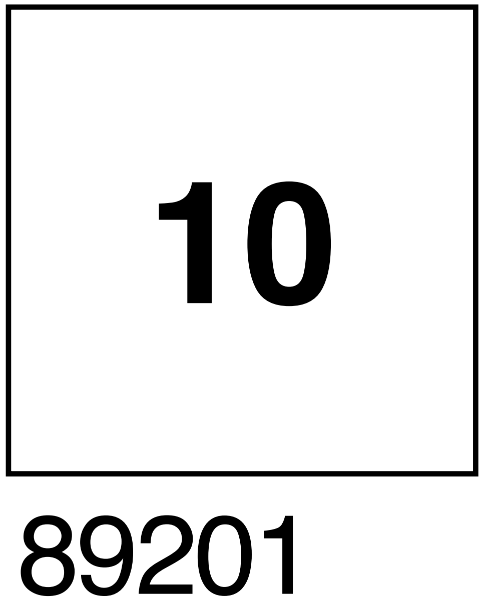
```
#include <stdio.h>
#include <stdlib.h>

void say_hello(int times)
{
  int i;
  for (i = 0; i < times; i++)
    printf(“Hello World\n”);
}

int main(int argc, char *argv[])
{
  say_hello(atoi(argv[1]));
  return 0;
}
```

every variable has an **address** and can store a **value.**

```
int x;
```



10

89201

assignment operator takes the **value** on the **right**, store into the **address** on the **left**.

```
int x;
int y;
x = 1;
y = 2;
x = y;
```

```
int *x;
int y;
y = 2;
x = &y;
*x = 3;
y = *x;

// what is *y, &x ?
```

```
int *x;
int y = 2;

*x = 1;
```

```
int y = 2;

f(y);



int f(int a)
{
   a = 2;
}
```

```
int y[3];

*y = 7;
*(y+1) = 9
```

```
char *name;

name = malloc(10);
strcpy(name, “cs2106”);
 :
 :
free(name);
```

```
struct student {
    int student_id;
    char *name;
    int age;
}
typedef struct student std;
```

```
std* create_student()
{
   std *s = malloc(sizeof(std));
   // initialize student
    :
    :
   return s;
}
```