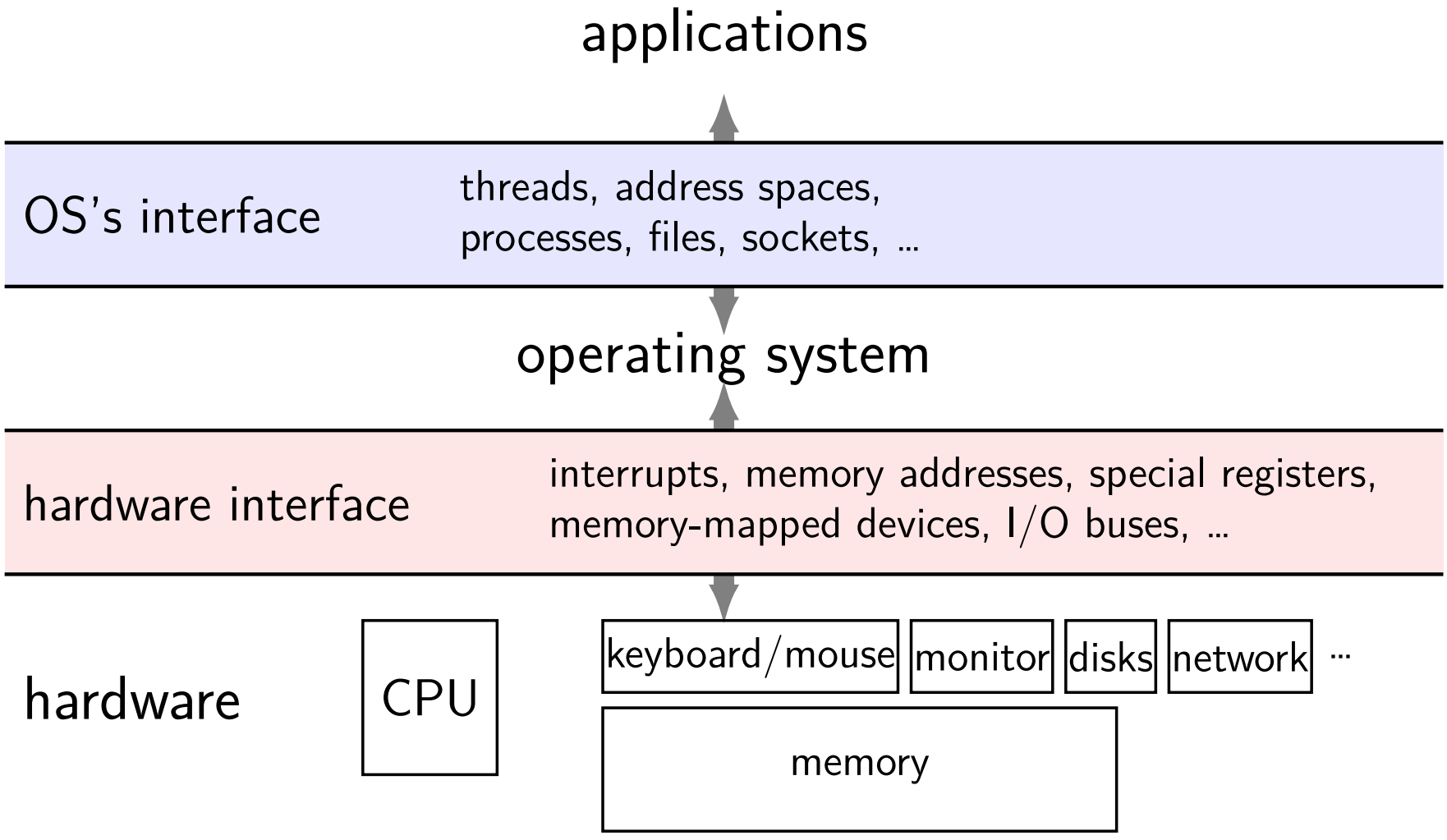
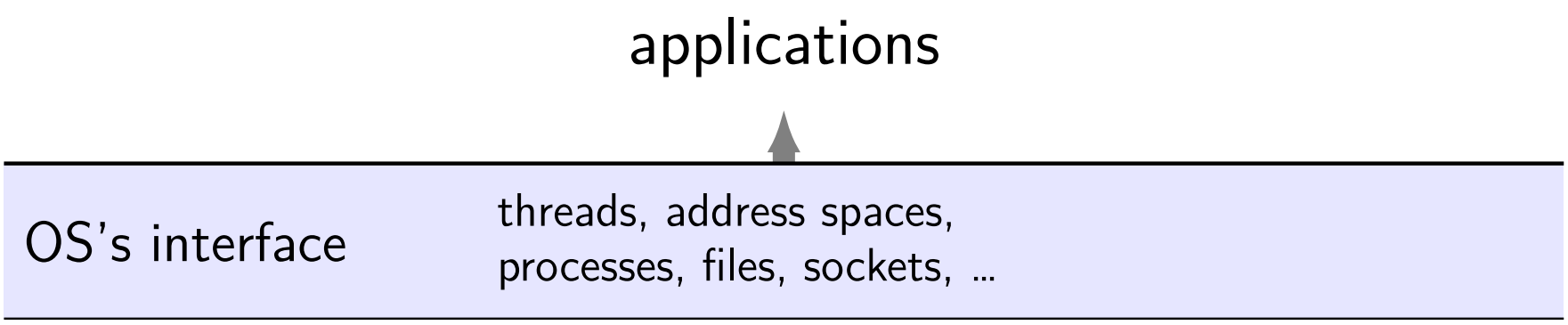
the abstract virtual machine



abstract VM: application view

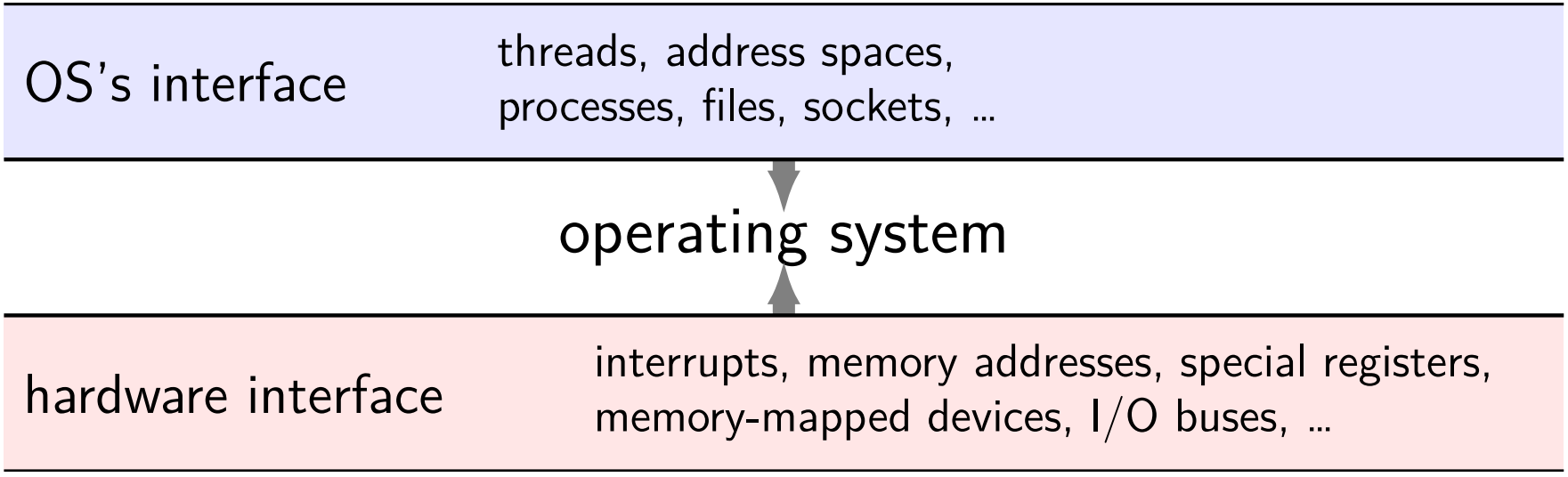


the application's "machine" is the operating system

no hardware I/O details visible — future-proof

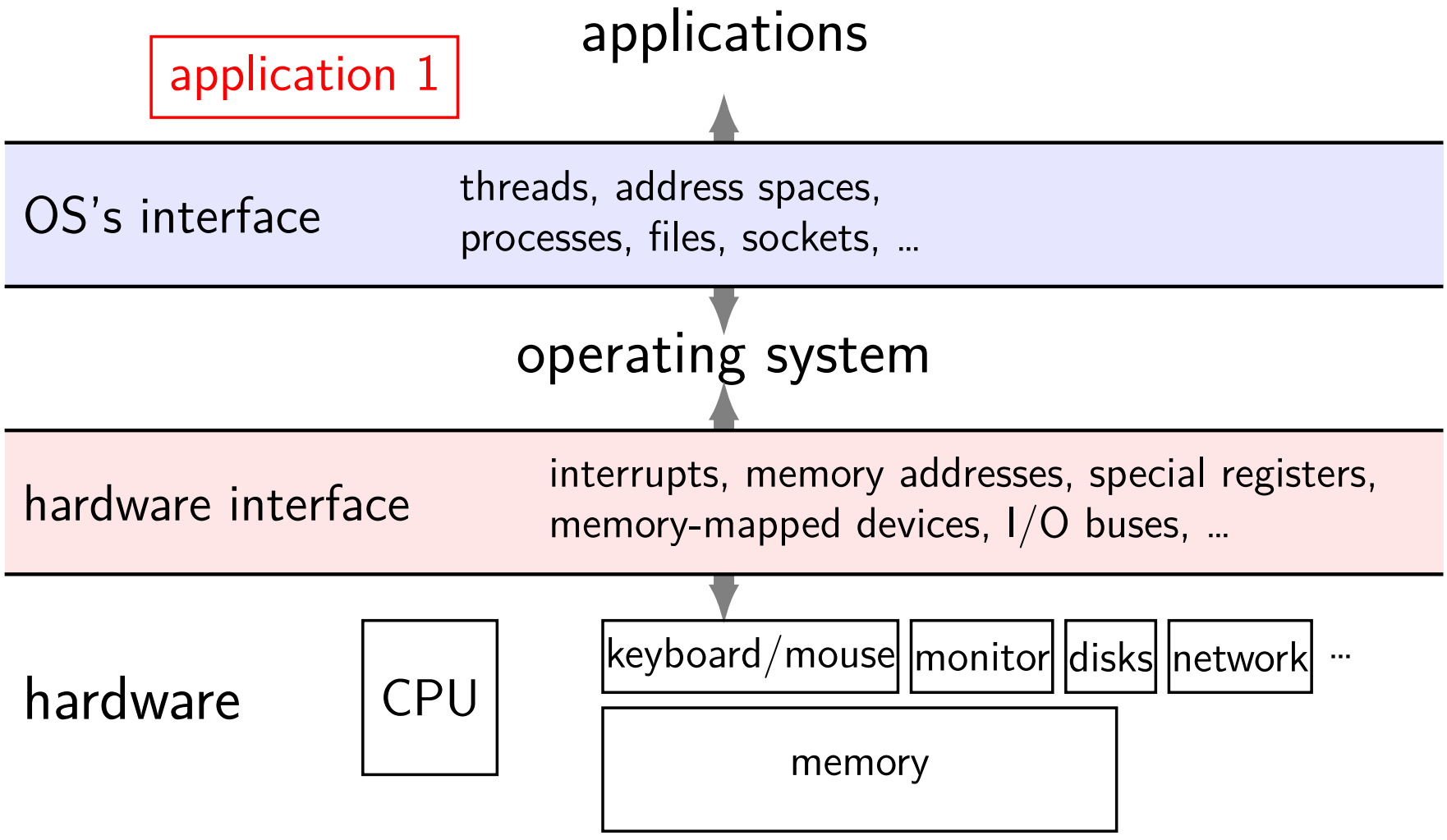
more featureful interfaces than real hardware

abstract VM: OS view

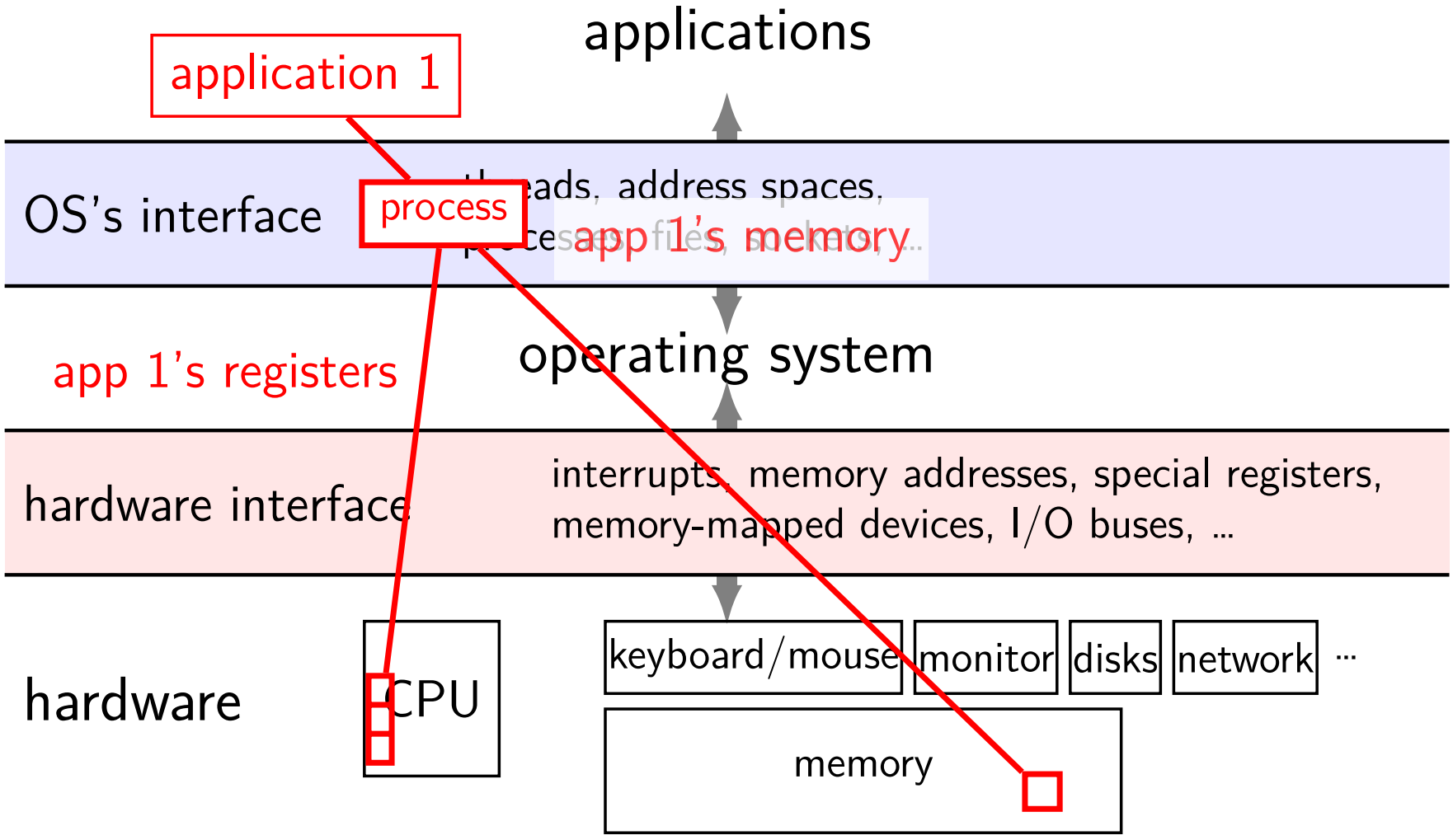


operating system's job: translate one interface to another

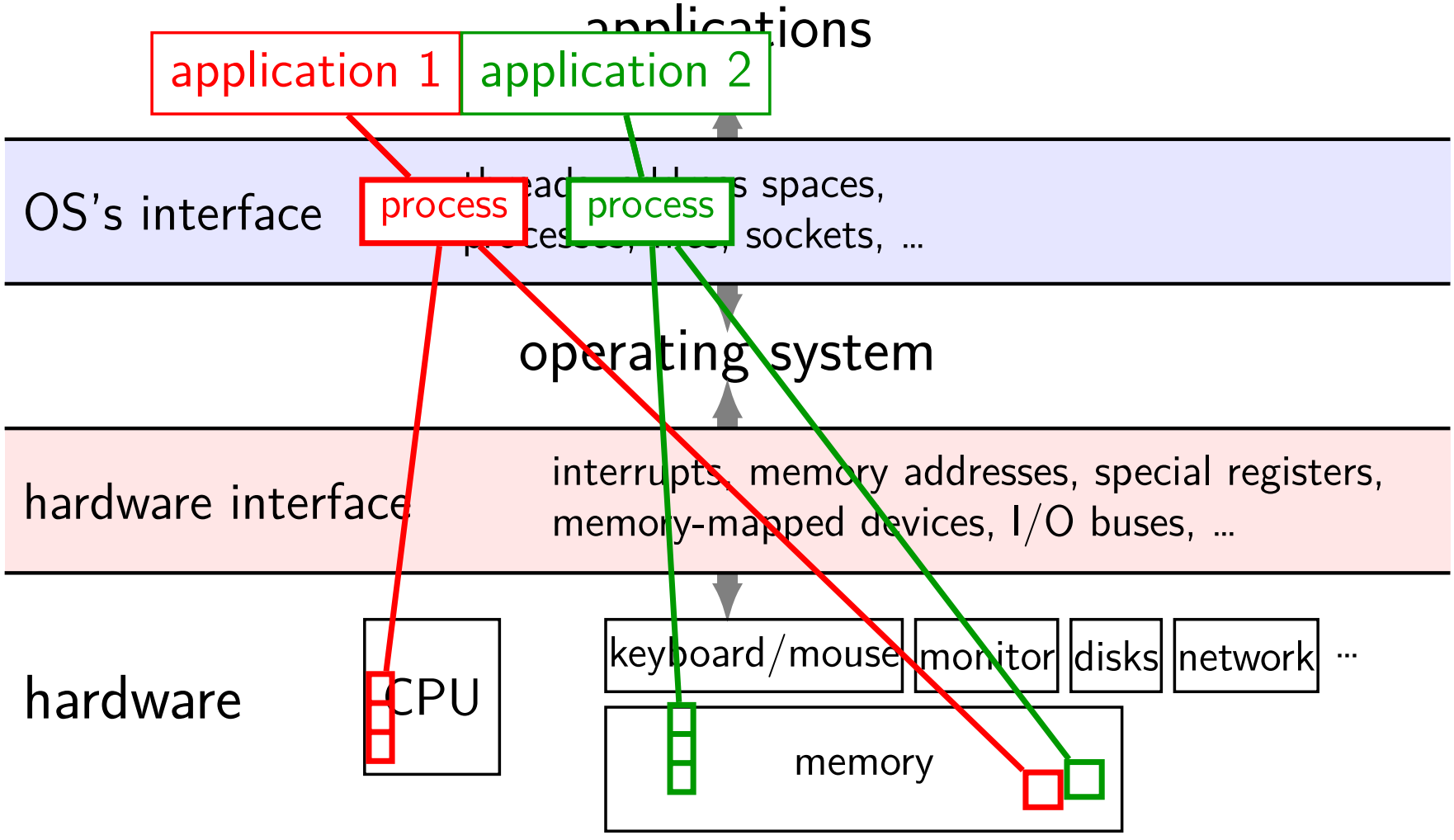
program → process → CPU and memory



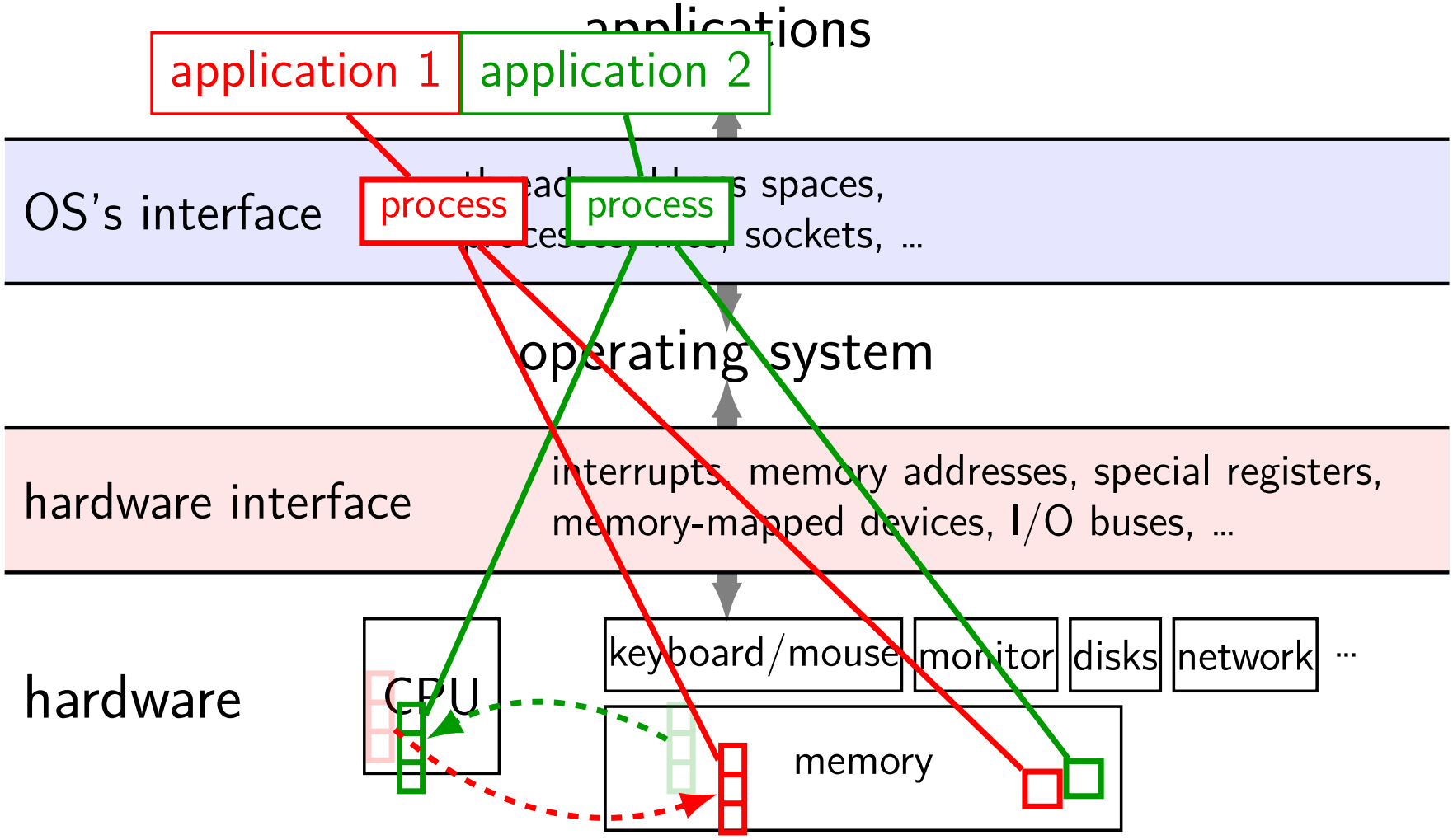
program → process → CPU and memory



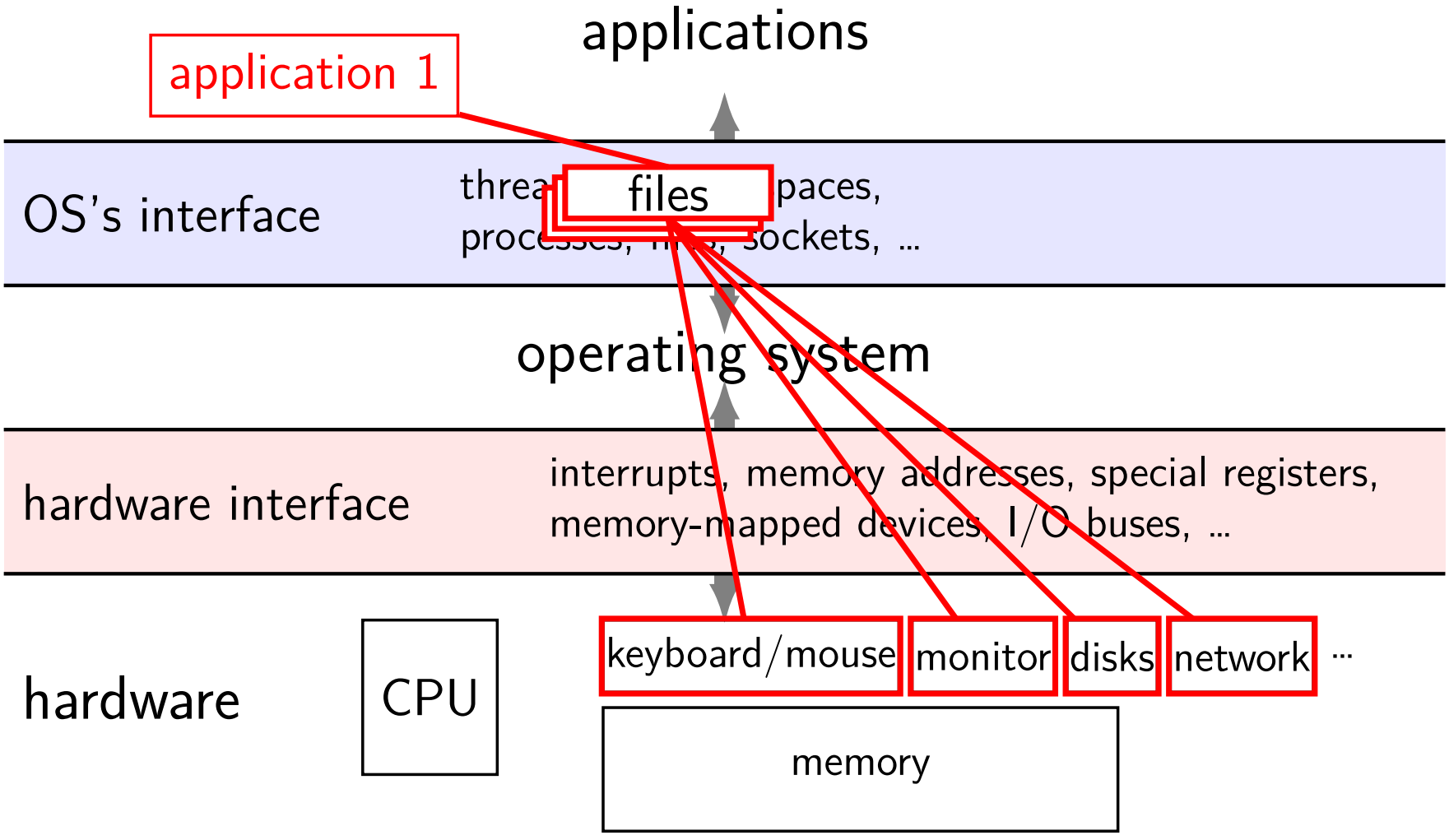
program → process → CPU and memory

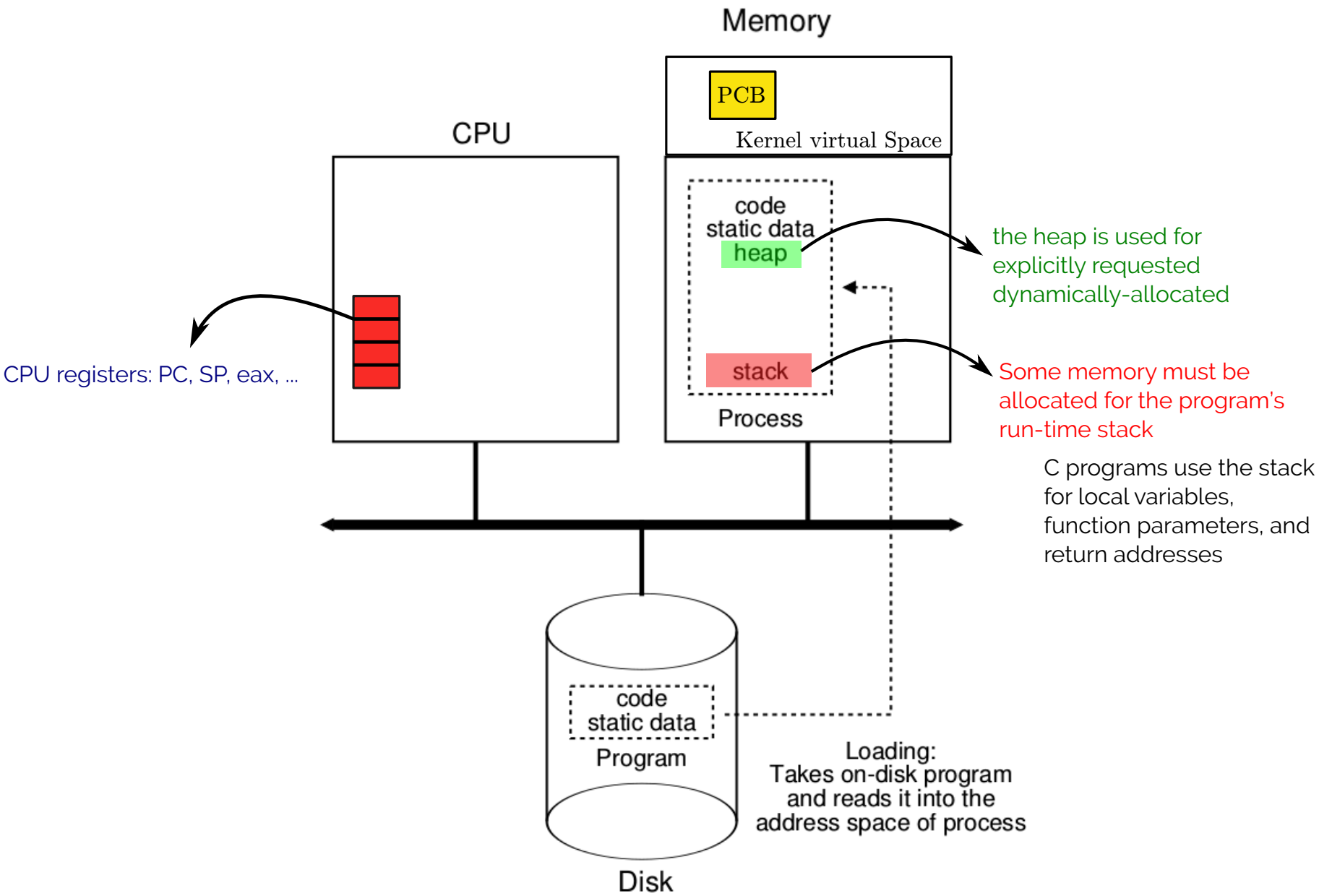


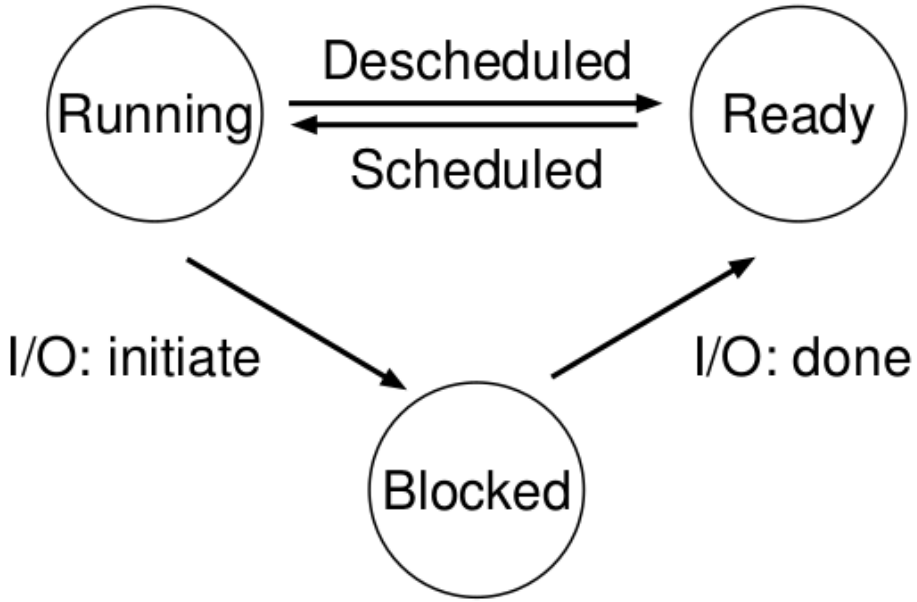
program → process → CPU and memory



files → input/output




Memory
PCB
Kernel virtual Space
CPU
code
static data
heap
stack
Process
CPU registers: PC, SP, eax, ...
the heap is used for explicitly requested dynamically-allocated
Some memory must be allocated for the program's run-time stack
C programs use the stack for local variables, function parameters, and return addresses
code
static data
Program
Loading:
Takes on-disk program and reads it into the address space of process
Disk



- running: its state word is contained in a processor which is running.
- ready: it could be placed in execution by a processor if one were free
- suspend: awaiting activation by an external event, such as the completion of an i/o function.

Example

Time	Process_0	Process_1	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	Process_0 now done
5	–	Running	
6	–	Running	
7	–	Running	
8	–	Running	Process_1 now done

Tracing Process State: CPU Only

Example

Time	$Process_0$	$Process_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	$Process_0$ initiates I/O
4	Blocked	Running	$Process_0$ is blocked,
5	Blocked	Running	so $Process_1$ runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	$Process_1$ now done
9	Running	–	
10	Running	–	$Process_0$ now done

Tracing Process State: CPU and I/O

Aside: xv6

In this course, instead of practical OS kernels (with millions of code lines), we use xv6 (just few thousands of code lines) for Kernel hacking

- A teaching OS developed at MIT
- simplified Unix version 6 (main change: from PDP-11 to Intel x86)
- well structured and documented
- github link: https://github.com/mit-pdos/xv6-public
- A commentary book: https://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf
- visit https://pdos.csail.mit.edu/6.828/2012/xv6.html

```
// the registers xv6 will save and restore
// to stop and subsequently restart a process
struct context {
  int eip;
  int esp;
  int ebx;
  int ecx;
  int edx;
  int esi;
  int edi;
  int ebp;
};

// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                  RUNNABLE, RUNNING, ZOMBIE };

// the information xv6 tracks about each process
// including its register context and state
struct proc {
  char *mem;                  // Start of process memory
  uint sz;                    // Size of process memory
  char *kstack;               // Bottom of kernel stack
                              // for this process
  enum proc_state state;      // Process state
  int pid;                    // Process ID
  struct proc *parent;        // Parent process
  void *chan;                 // If !zero, sleeping on chan
  int killed;                 // If !zero, has been killed
  struct file *ofile[NOFILE]; // Open files
  struct inode *cwd;          // Current directory
  struct context context;     // Switch here to run process
  struct trapframe *tf;       // Trap frame for the
                              // current interrupt
};
```

The xv6 Proc Structure

(**Process Control Block (PCB)/process descriptor**)

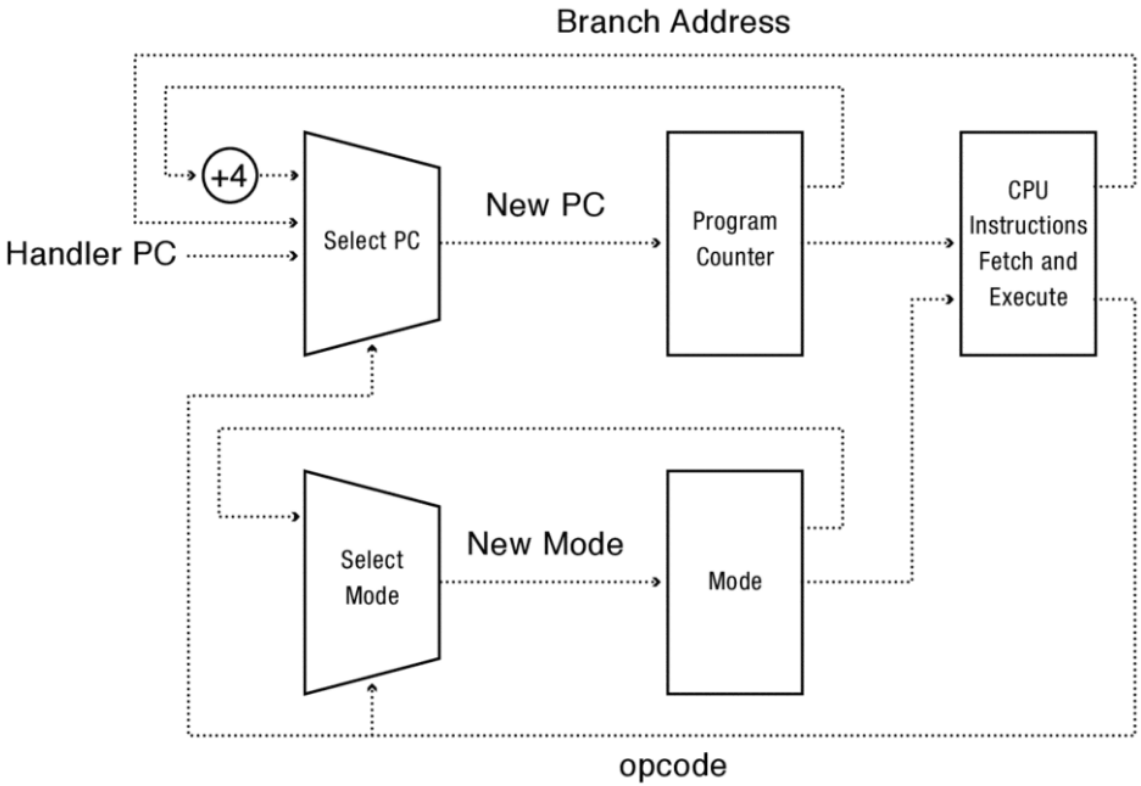
Hardware Support

Dual Mode

Exception

Memory Protection

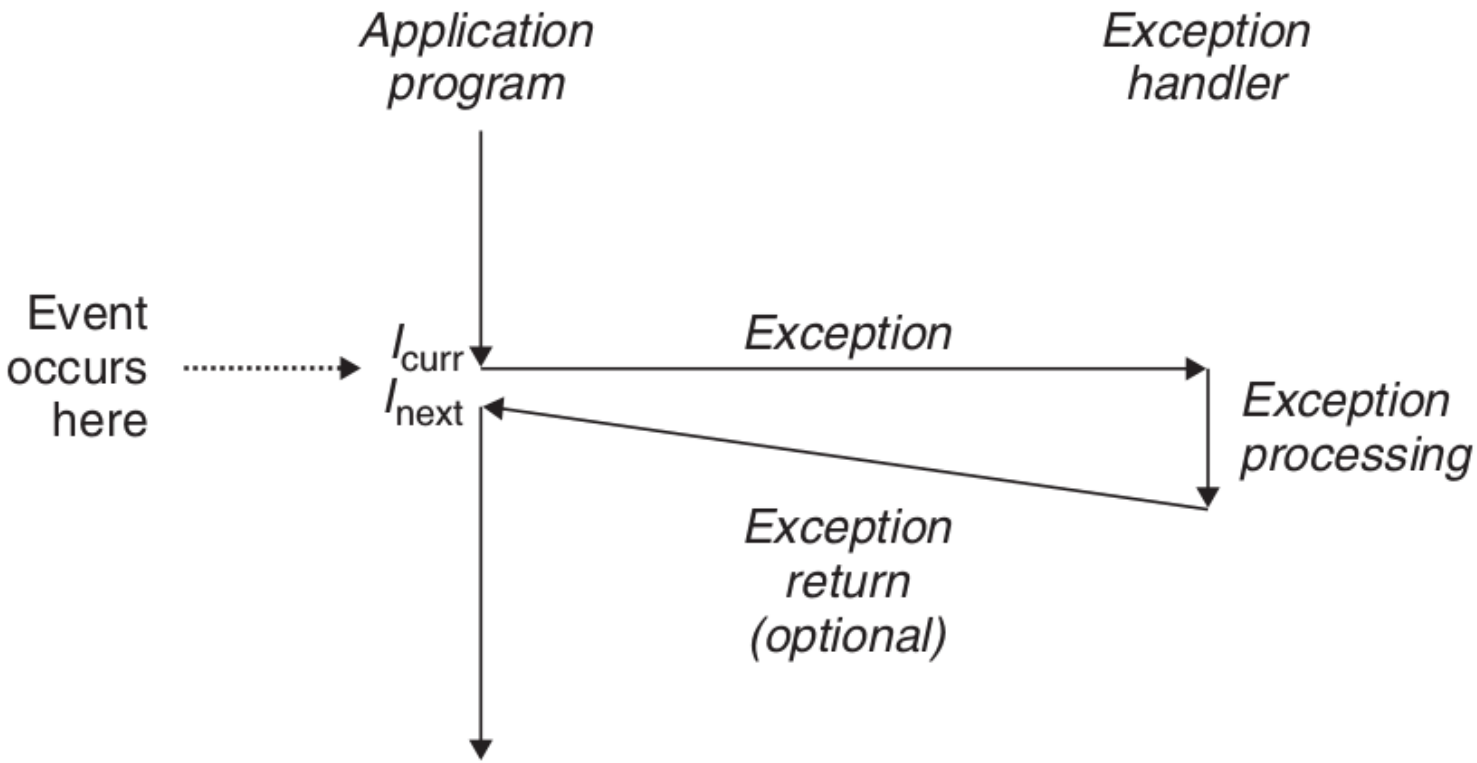
The operation of a CPU with kernel and user modes

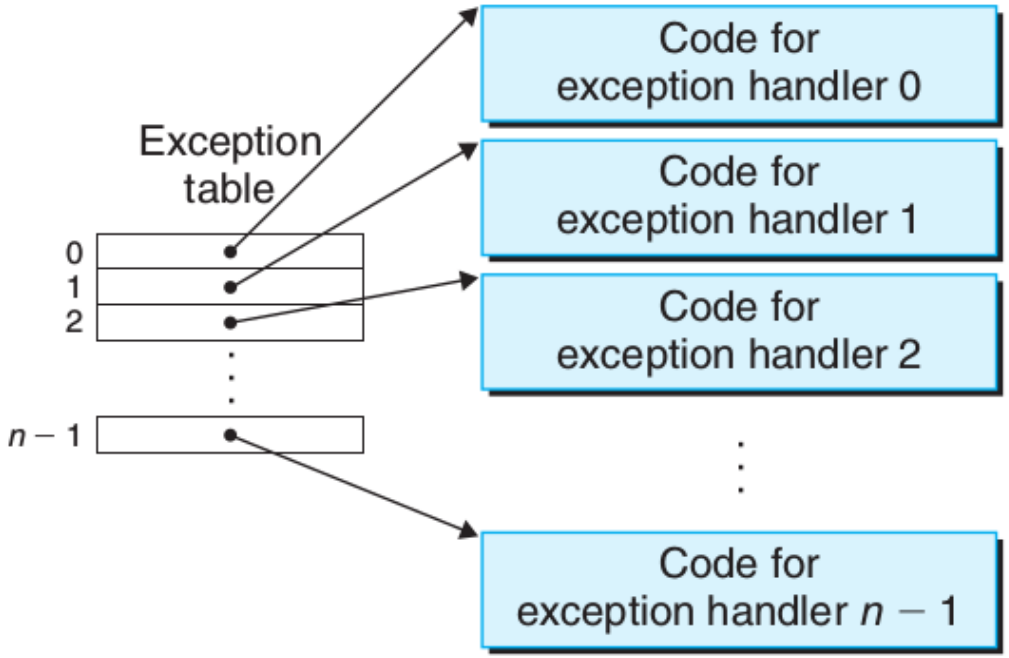


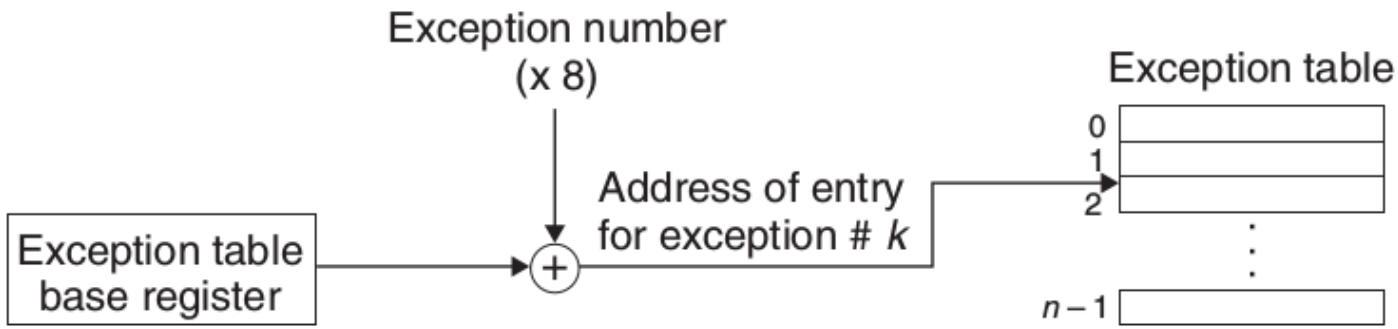
The kernel/user mode bit is one flag in the processor **status register**, set whenever the kernel is entered and reset whenever the kernel switches back to user mode.

Exception

An exception is an abrupt change in the control flow in response to some change in the processor's state.




Exception table
0
1
2
n – 1
Code for exception handler 0
Code for exception handler 1
Code for exception handler 2
Code for exception handler n – 1


Exception number (x 8)
Exception table base register
Address of entry for exception # k
Exception table
0
1
2
n – 1

Classes of Exceptions

Asynchronous (Interrupts)

Synchronous: Traps, Faults, Aborts

Exercise: How many exceptions?

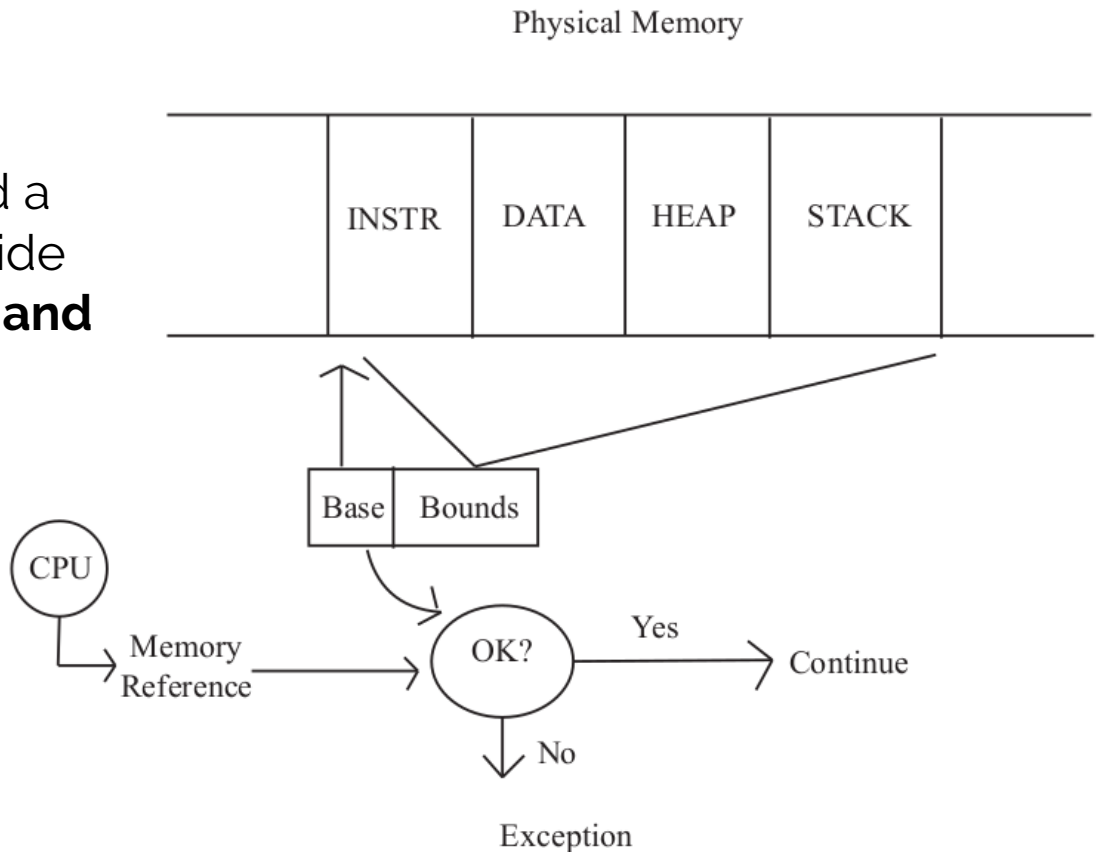
Single core, 3 processes A, B, and C

- Running process A
- A prompts for input then
- A waits to read a keypress
- While A is waiting for the keypress, OS runs B, then C
- then keypress happens, and OS switches to A immediately
- then A exits

Memory Protection

All memory accesses outside of a process's valid memory region are prohibited when executing in user mode.

Early computers pioneered a simple mechanism to provide protection, known as **base and bounds**.



More elaborated methods in the next chapter ....