# Operating Systems and Protection

Prof. David August

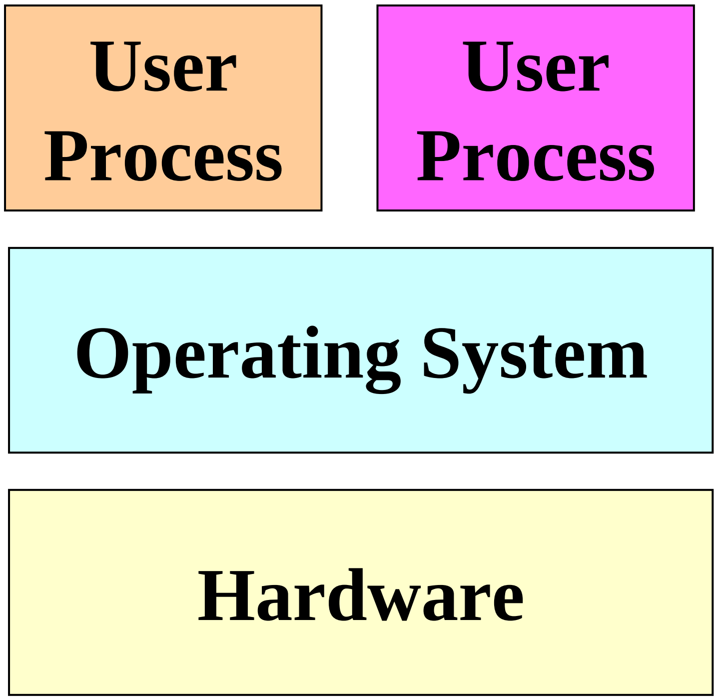
COS 217

# Goals of Today's Lecture

- How multiple programs can run at once
  - Processes
  - Context switching
  - Process control block
  - Virtual memory
- Boundary between parts of the system
  - User programs
  - Operating system
  - Underlying hardware
- Mechanics of handling a page fault
  - Page tables
  - Process ID registers
  - Page faults

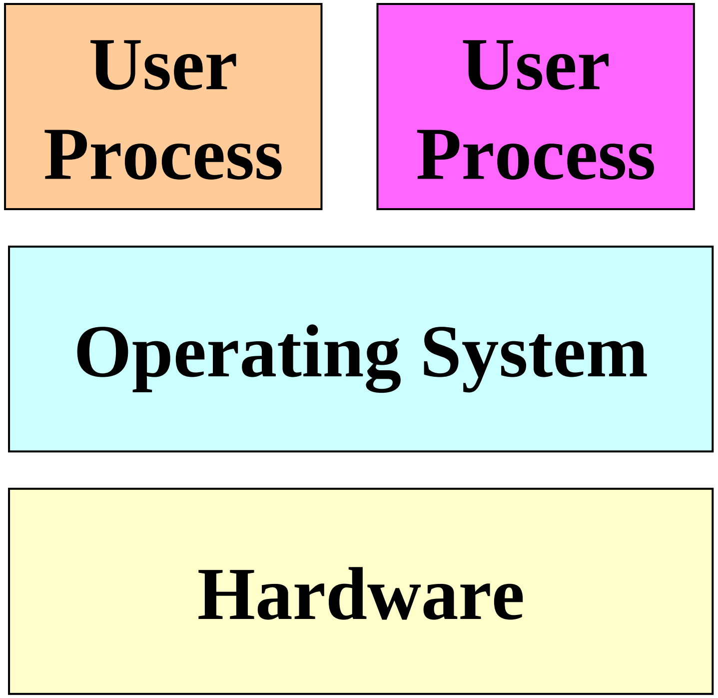
# Operating System

- Supports virtual machines
  - Promises each process the illusion of having whole machine to itself
- Provides services:
  - Protection
  - Scheduling
  - Memory management
  - File systems
  - Synchronization
  - etc.

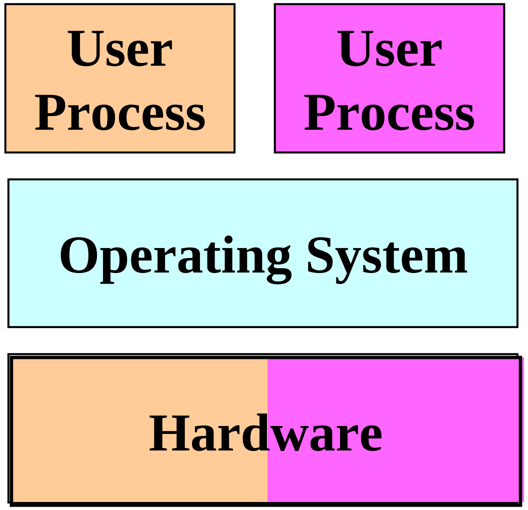


# What is a Process?

- A process is a running program with its own …
  - Processor state
    - EIP, EFLAGS, registers
  - Address space (memory)
    - Text, bss, data, heap, stack
- Supporting the abstraction
  - Processor
    - Saving state per process
    - Context switching
  - Main memory
    - Sharing physical memory
    - Supporting virtual memory
  - Efficiency, fairness, protection

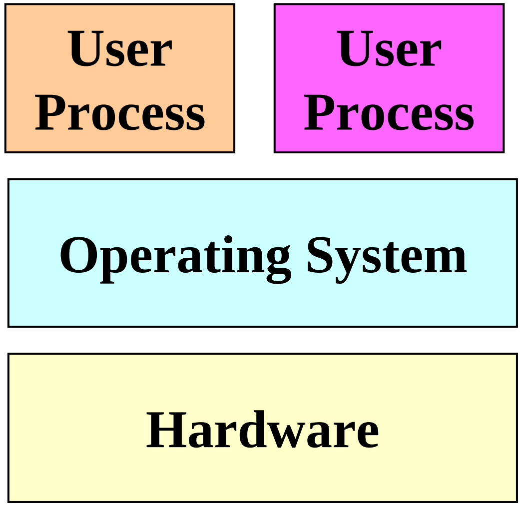


# Divide Hardware into Little Pieces?




- Idea: registers, memory, ALU, etc. per process
  - Pro: totally independent operation of each process
  - Con: lots of extra hardware;
    some parts idle at any given time;
    hard limit on the number of processes

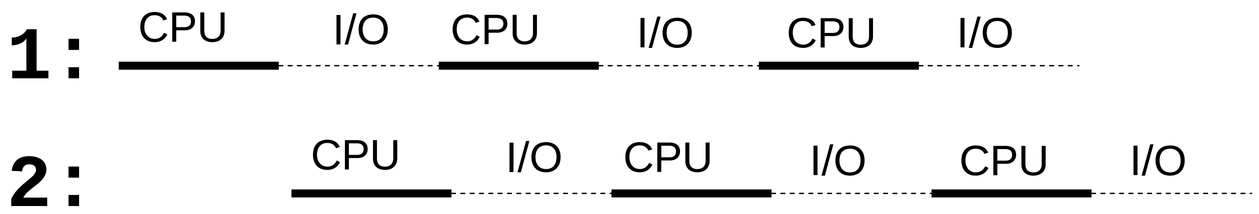
# Indirection, and Sharing in Time?




- Idea: swap processes in and out of the CPU;
  map references into physical addresses
  - Pro: make effective use of the resources by sharing
  - Con: overhead of swapping processes;
    overhead of mapping memory references

# When to Change Which Process is Running?

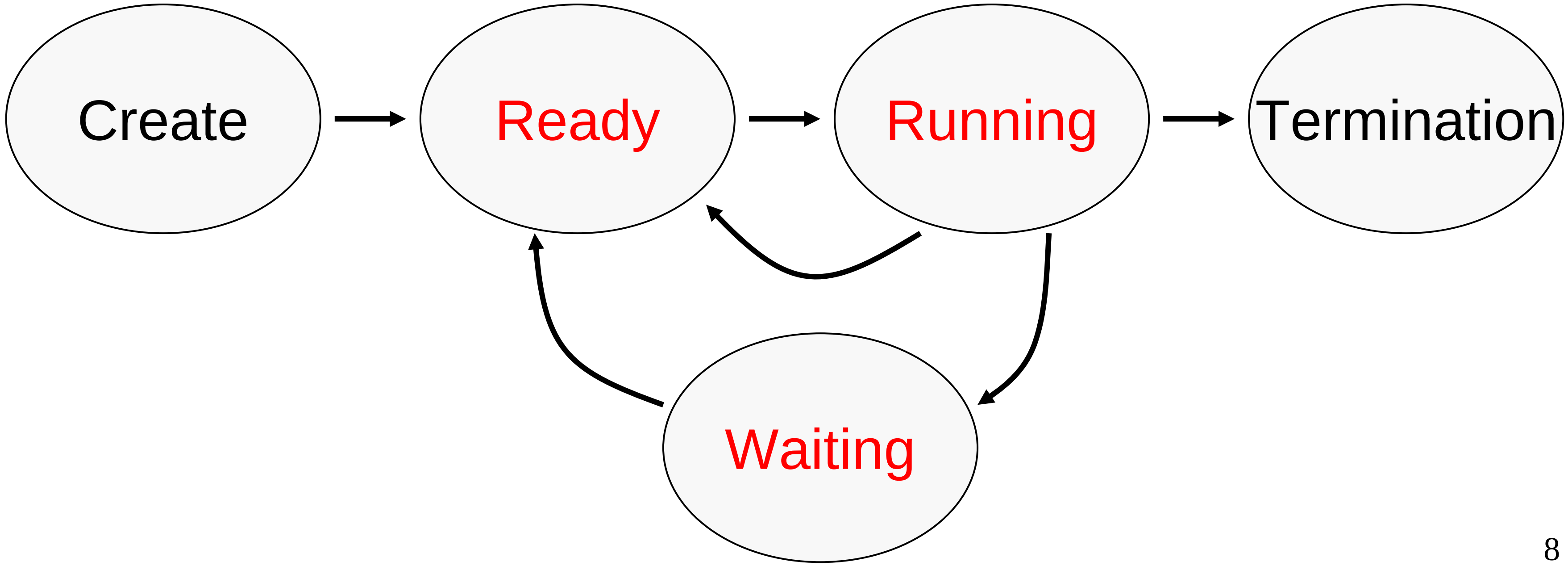
- When a process is stalled waiting for I/O
  - Better utilize the CPU, e.g., while waiting for disk access



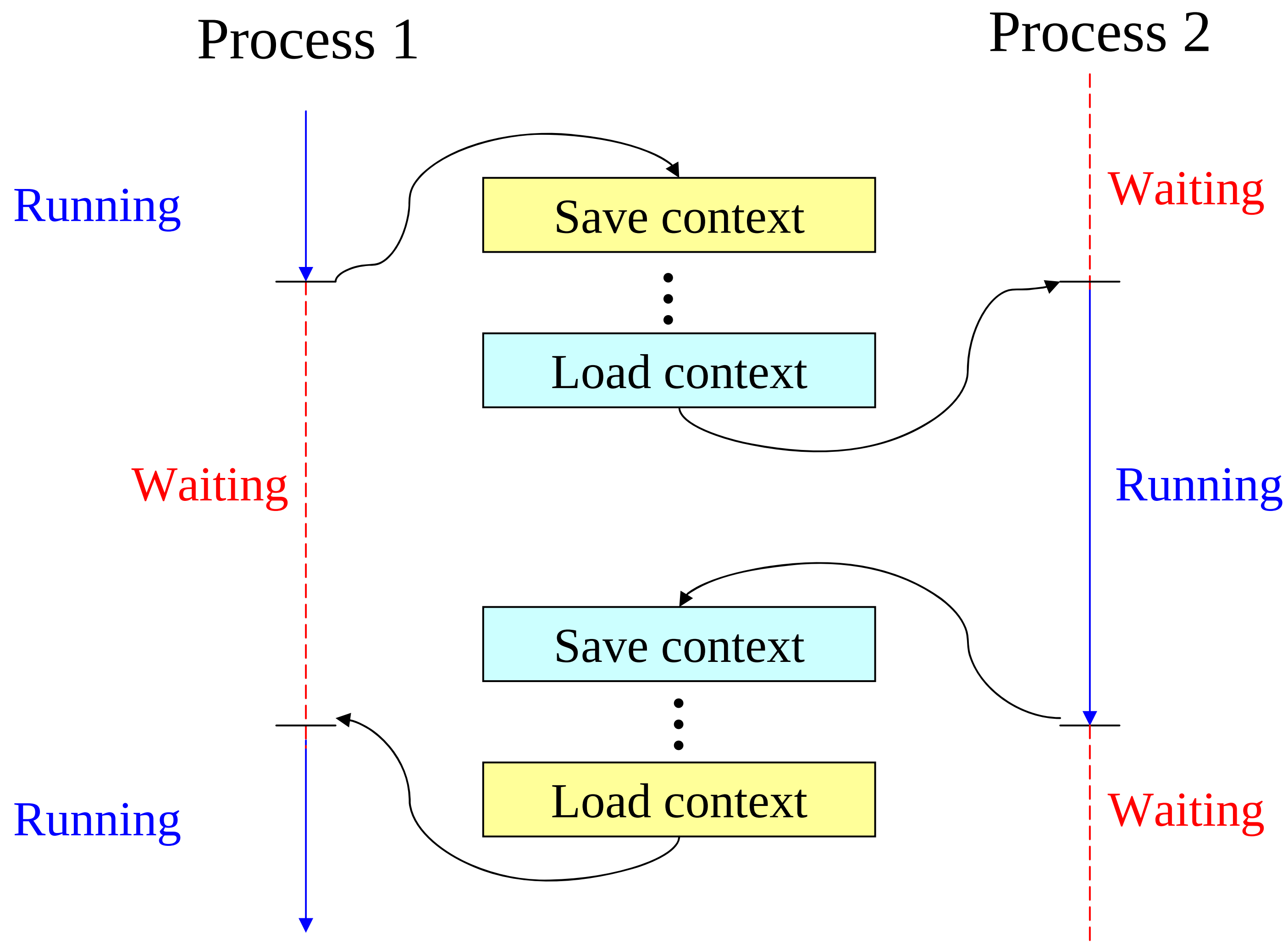

- When a process has been running for a while
  - Sharing on a fine time scale to give each process the illusion of running on its own machine
  - Trade-off efficiency for a finer granularity of fairness

# Life Cycle of a Process

- Running: instructions are being executed
- Waiting: waiting for some event (e.g., I/O finish)
- Ready: ready to be assigned to a processor



# Switching Between Processes

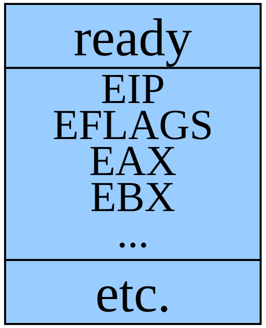


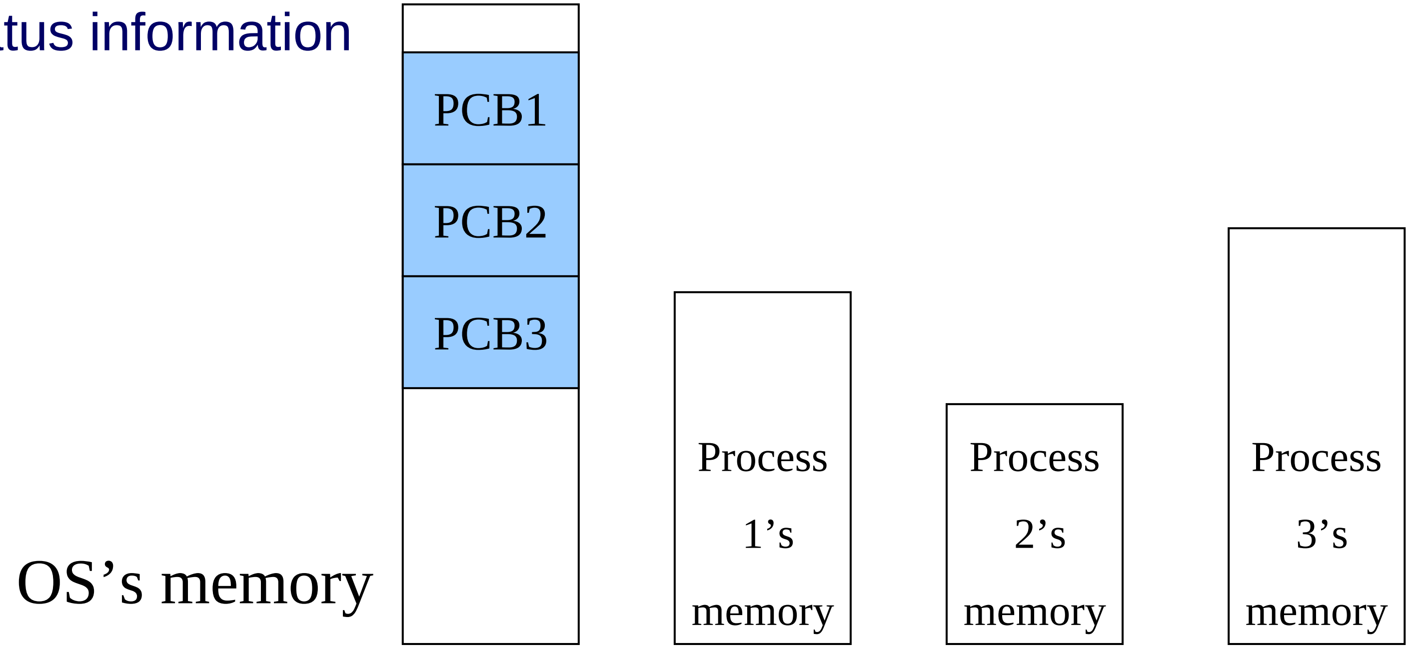
# Context Switch: What to Save & Load?

- Process state
  - New, ready, waiting, halted
- CPU registers
  - EIP, EFLAGS, EAX, EBX, …
- I/O status information
  - Open files, I/O requests, …
- Memory management information
  - Page tables
- Accounting information
  - Time limits, group ID, ...
- CPU scheduling information
  - Priority, queues

# Process Control Block

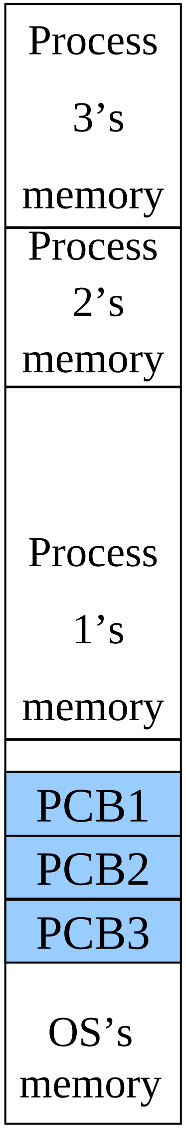
- For each process, the OS keeps track of ...
  - Process state
  - CPU registers
  - CPU scheduling information
  - Memory management information
  - Accounting information
  - I/O status information



# Sharing Memory

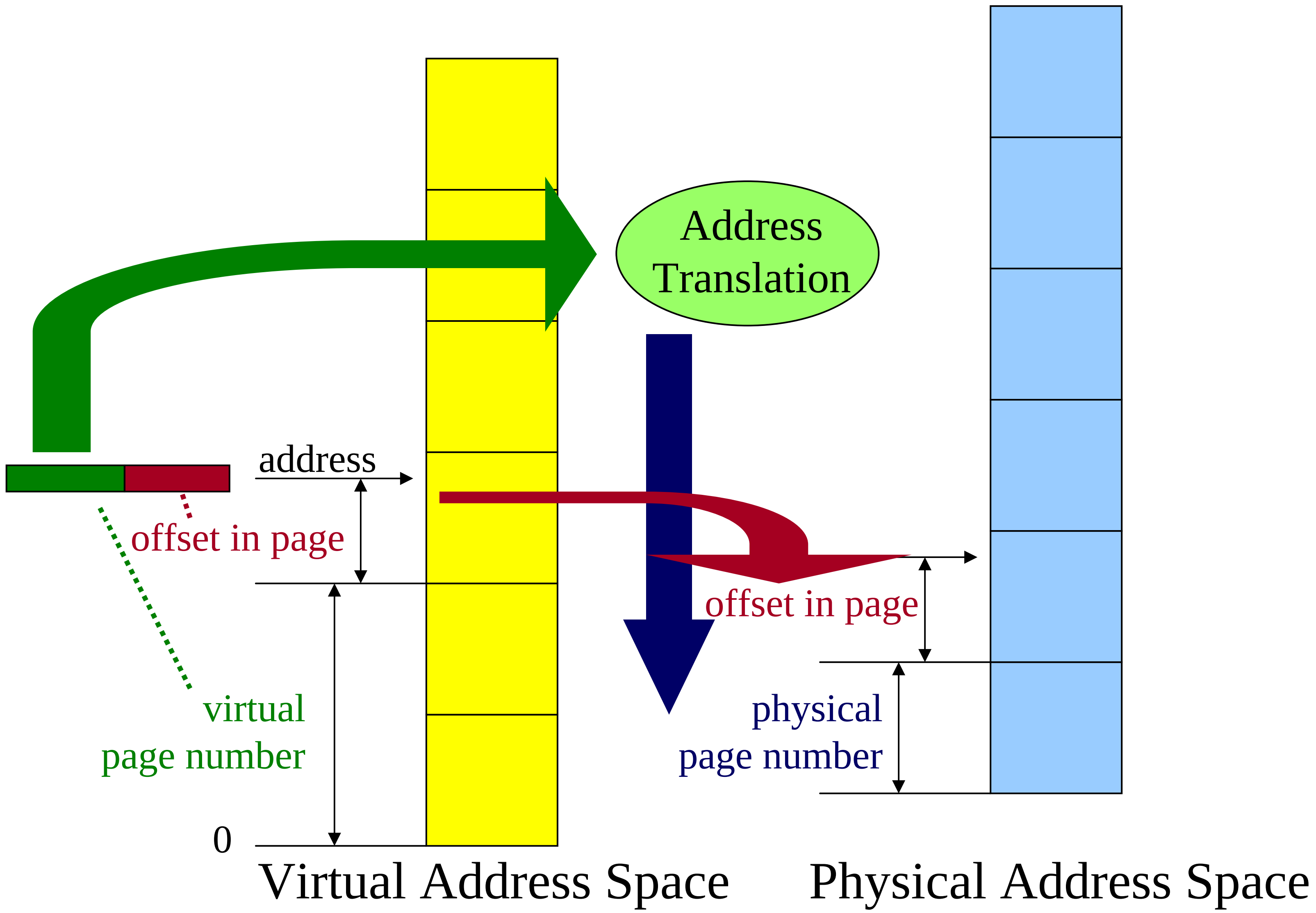
- In the old days…
  - MS-DOS (1990)
  - Original Apple Macintosh (1984)
- Problem: protection
  - What prevents process 1 from reading/writing process 3's memory?
  - What prevents process 2 from reading/writing OS's memory?
- In modern days, Virtual Memory protection
  - IBM VM-370 (1970)
  - UNIX (1975)
  - MS Windows (2000)



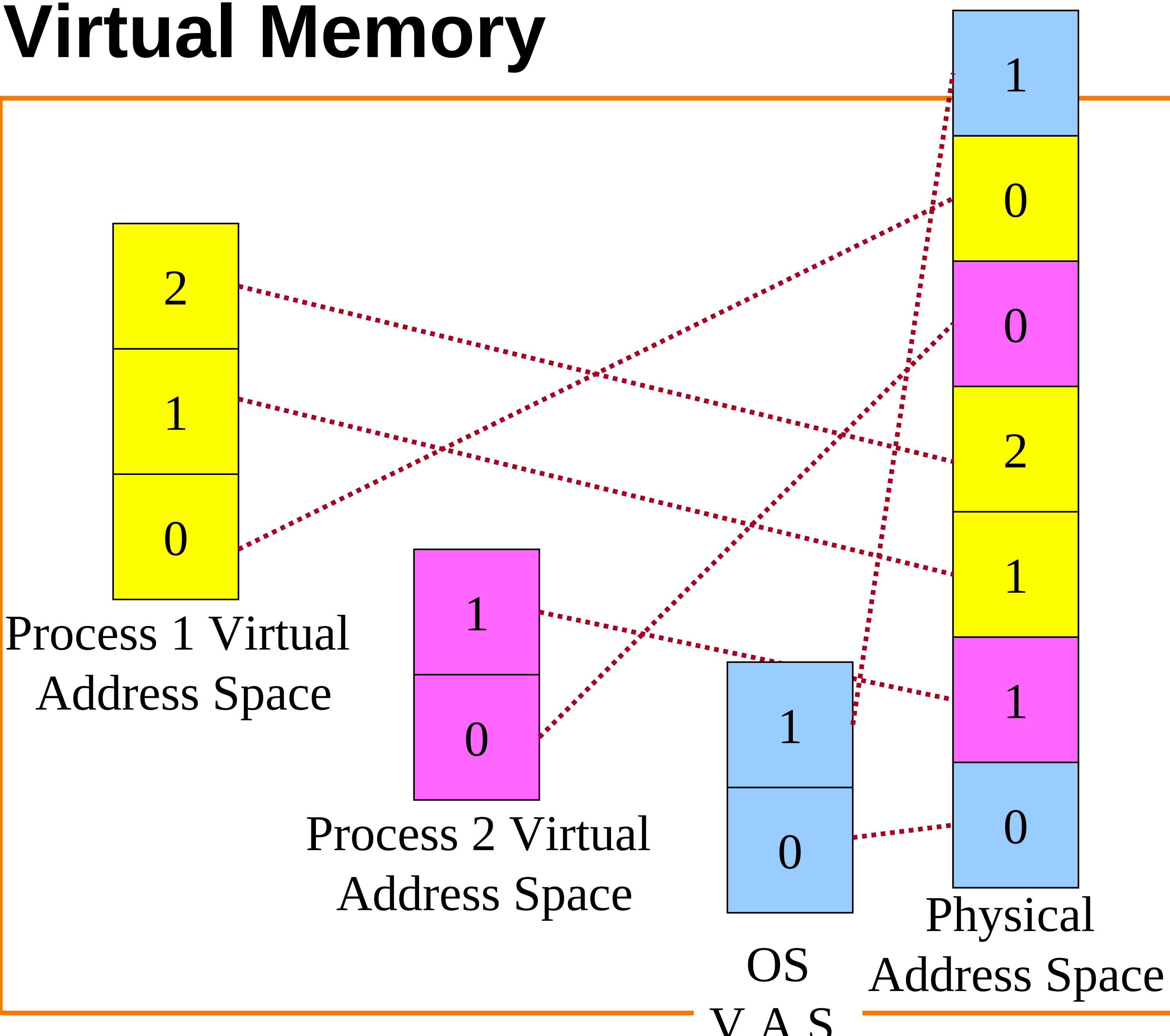
# Virtual Memory

- Give each process illusion of large address space
  - E.g., 32-bit addresses that reference 4 Gig of memory
- Divide the physical memory into fixed-sized pages
  - E.g., 4 Kilobyte pages
- Swap pages between disk and main memory
  - Bring in a page when a process accesses the space
  - May require swapping *out* a page already in memory
- Keep track of where pages are stored in memory
  - Maintain a page table for each process to do mapping
- Treat address as page number and offset in page
  - High-order bits refer to the page
  - Low-order bits refer to the offset in the page

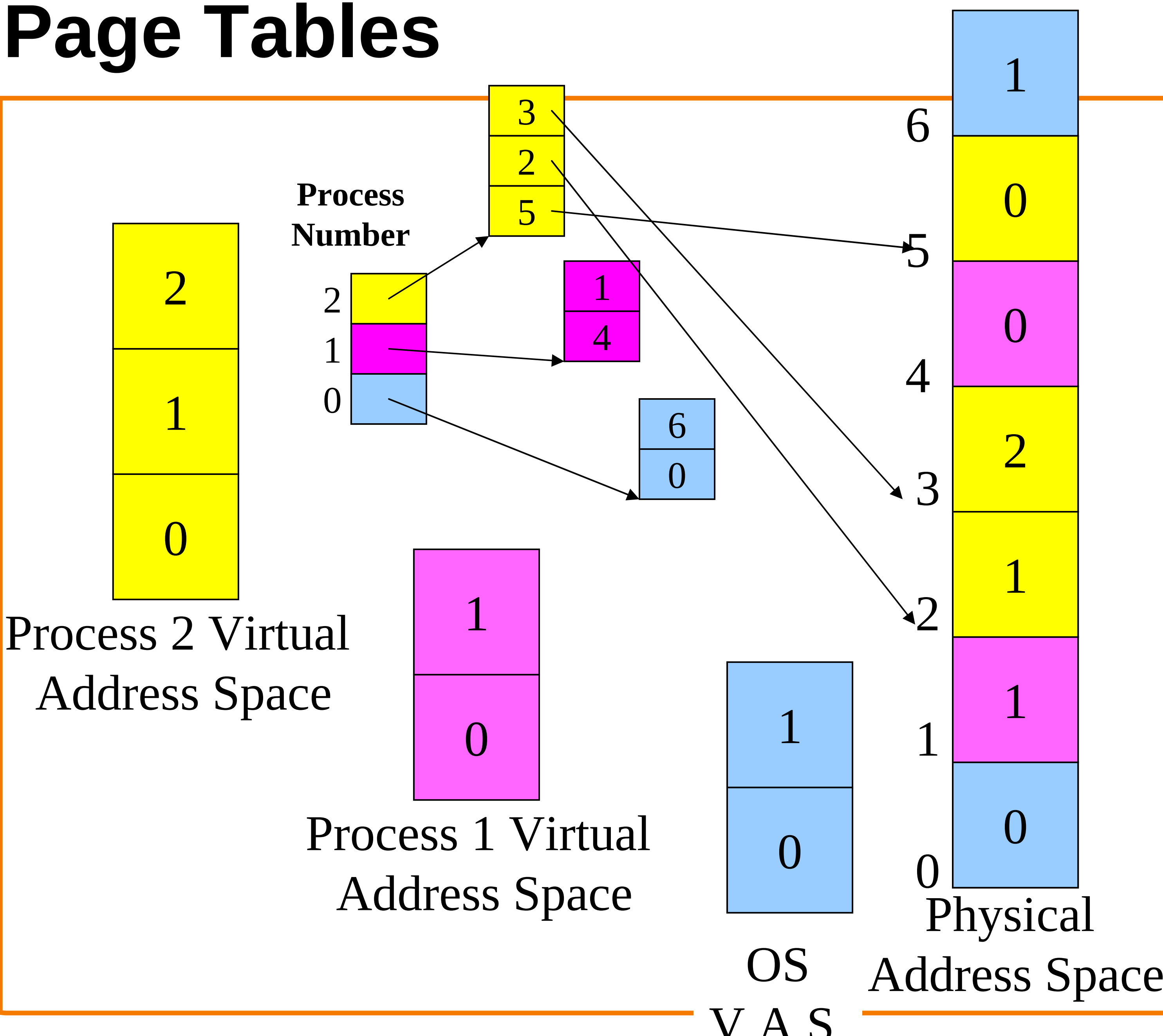
# Virtual Memory for a Process



# Virtual Memory

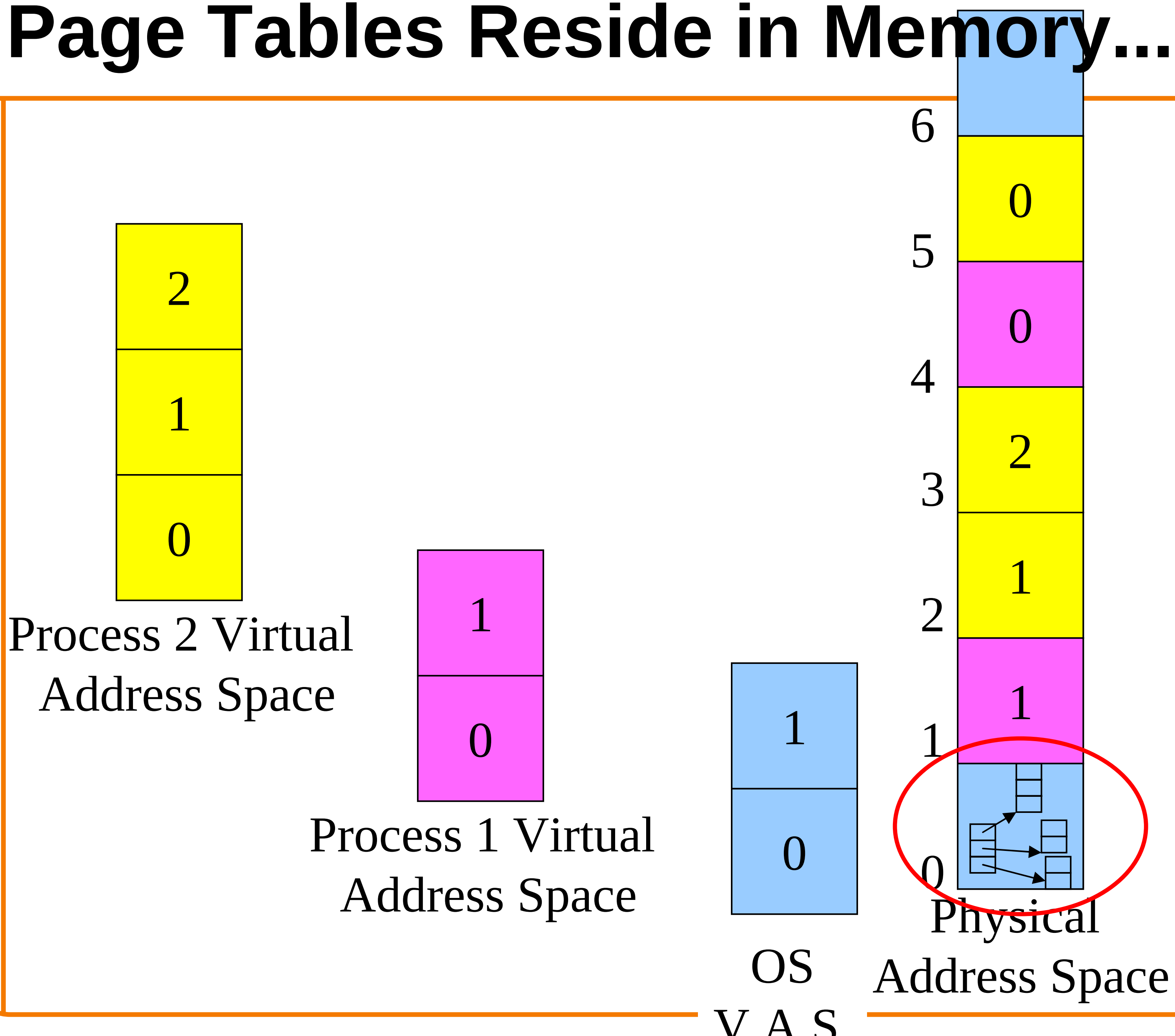


# Page Tables

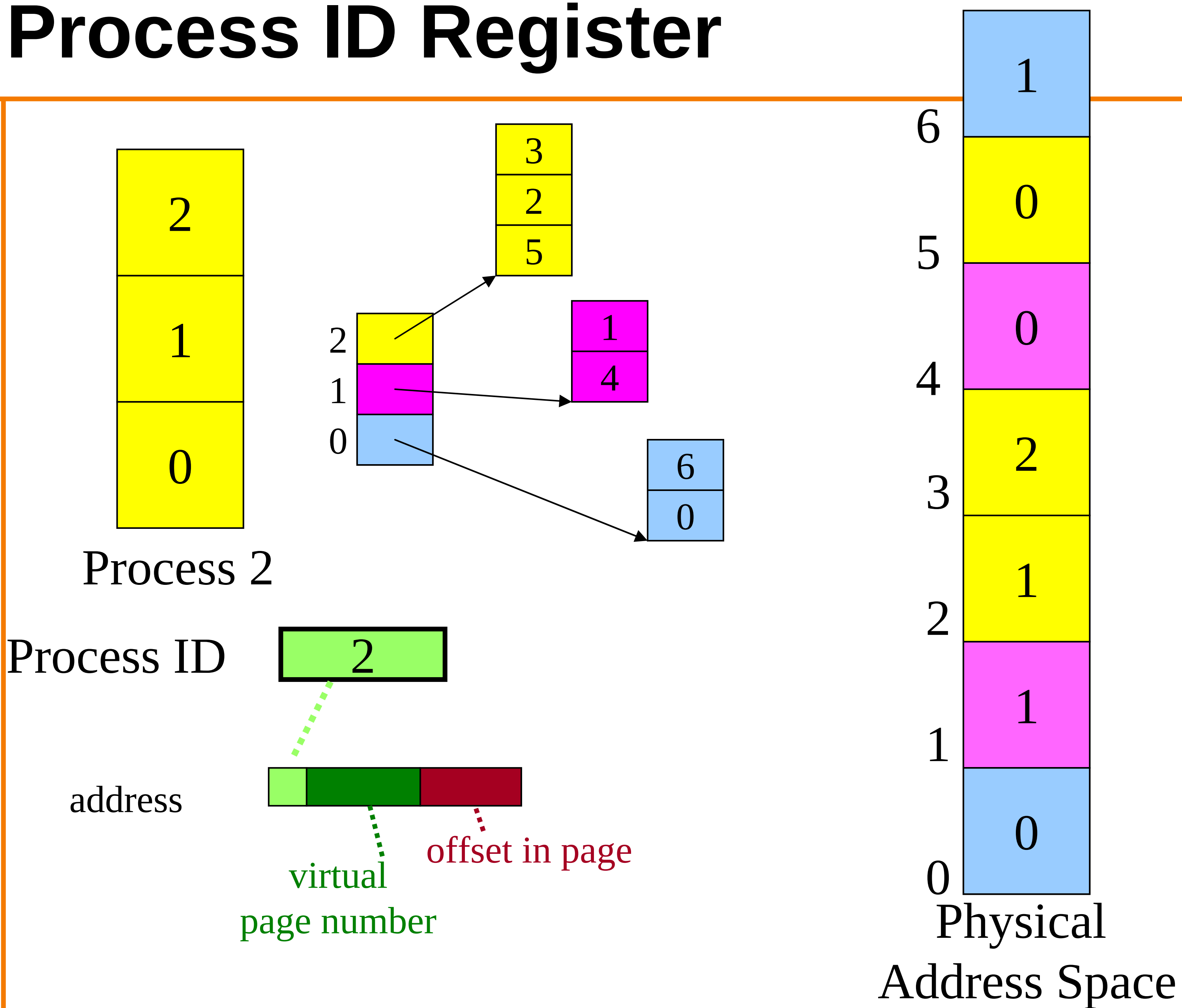


3
2
5
Process Number
2
1
0
1
4
6
0
2
1
0
Process 2 Virtual Address Space
1
0
Process 1 Virtual Address Space
1
0
OS V.A.S.
6
5
4
3
2
1
0
1
0
0
2
1
1
0
Physical Address Space

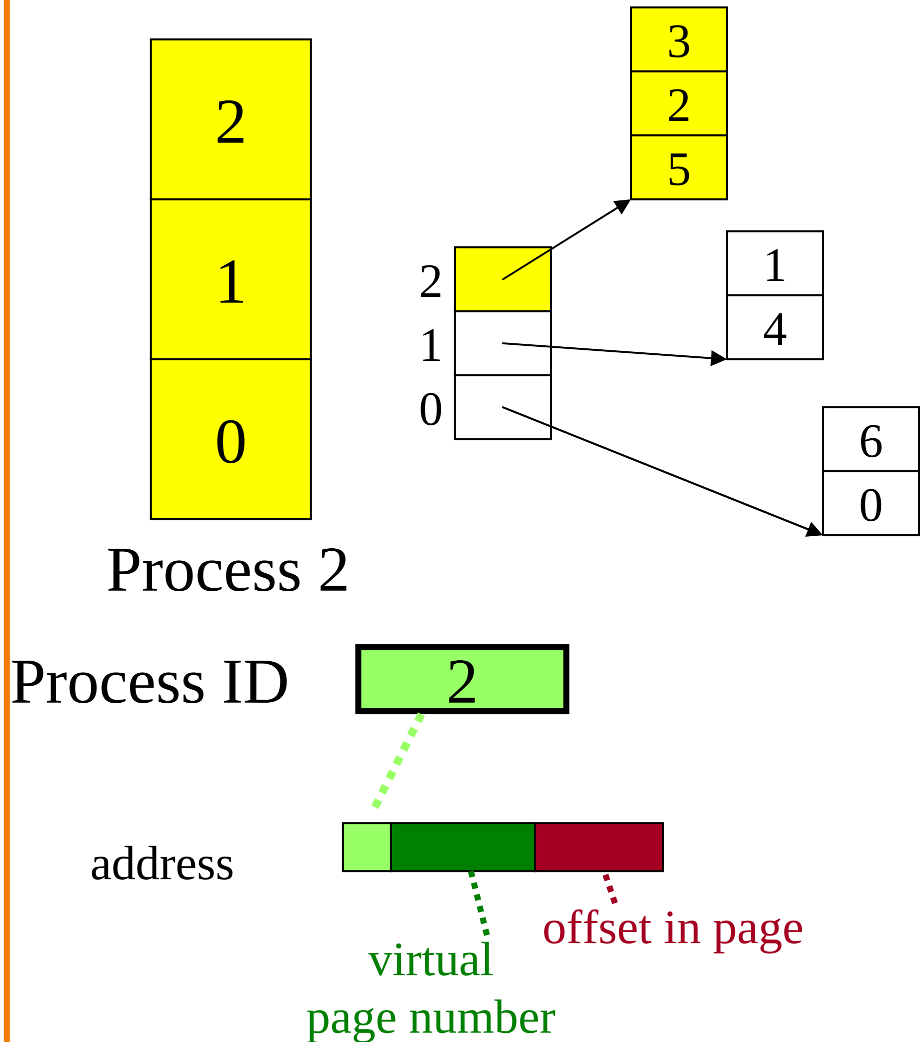
# Page Tables Reside in Memory...



# Process ID Register

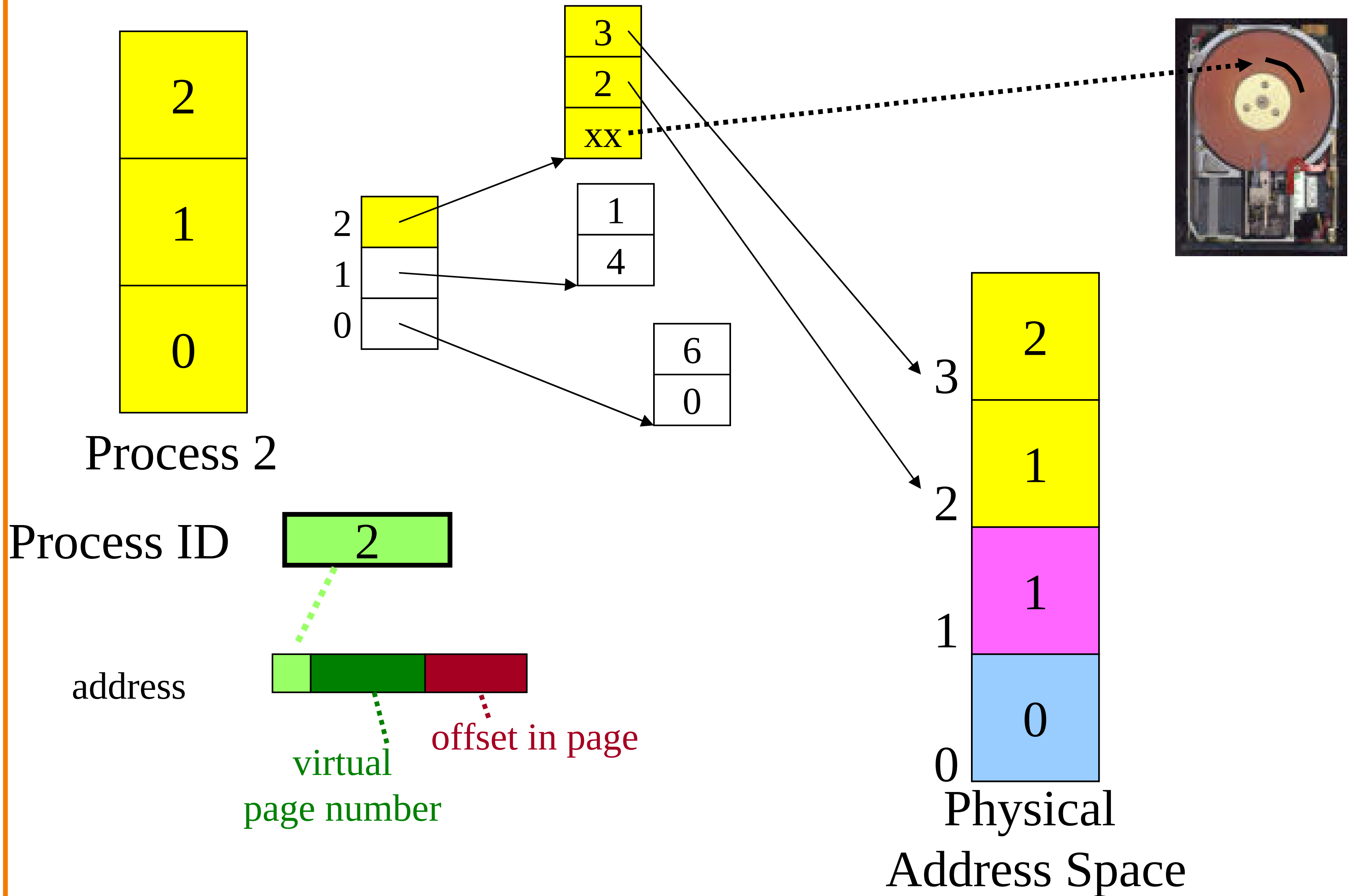


# Protection Between Processes



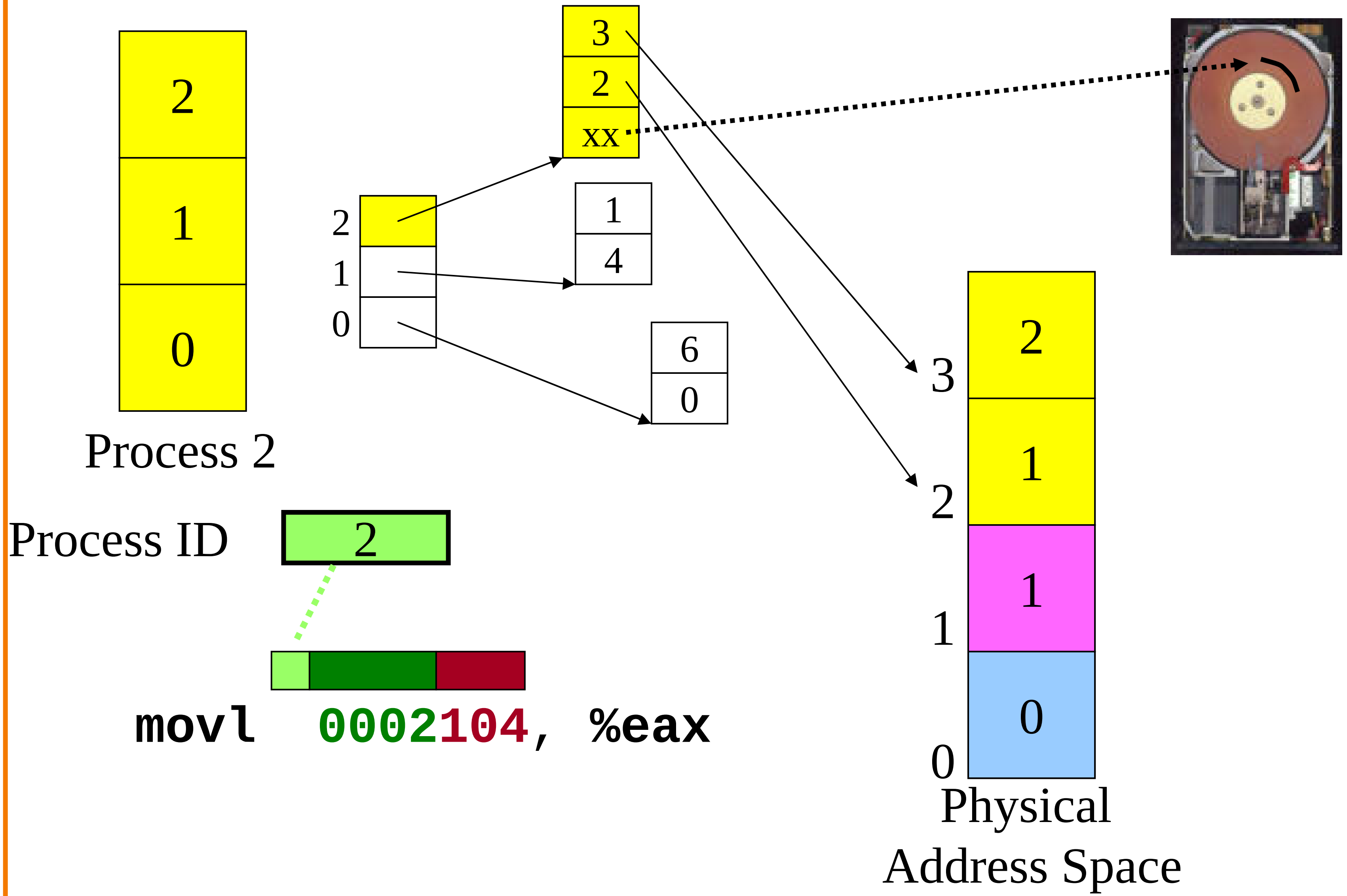

- User-mode (unprivileged) process *cannot* modify Process ID register
- If page tables are set up correctly, process #1 can access *only* its own pages in physical memory
- The operating system sets up the page tables

# Paging



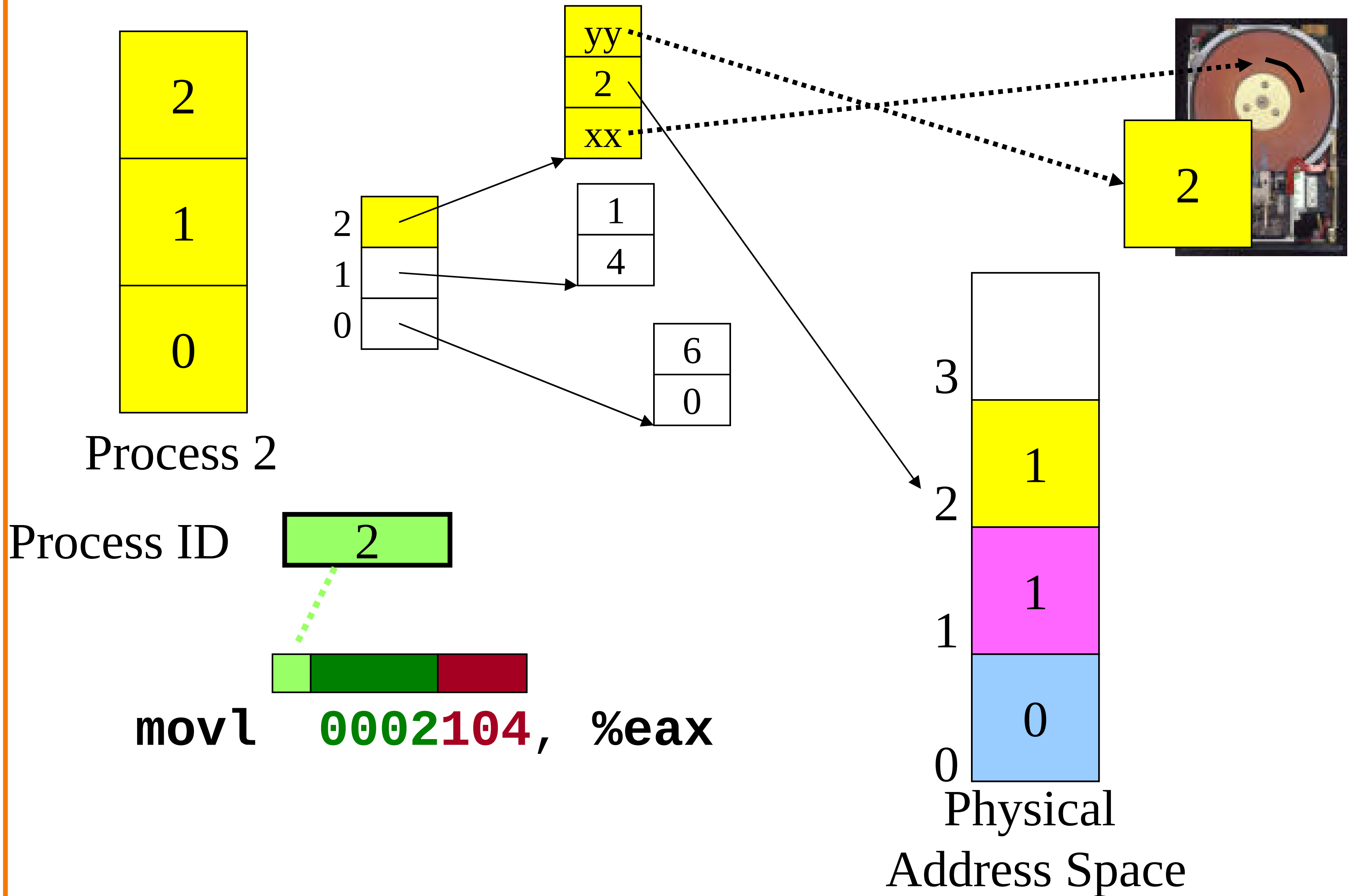
2
1
0
Process 2
2
1
0
3
2
xx
1
4
6
0
Process ID
2
address
virtual
page number
offset in page
3
2
1
0
2
1
1
0
Physical
Address Space

# Page Fault!



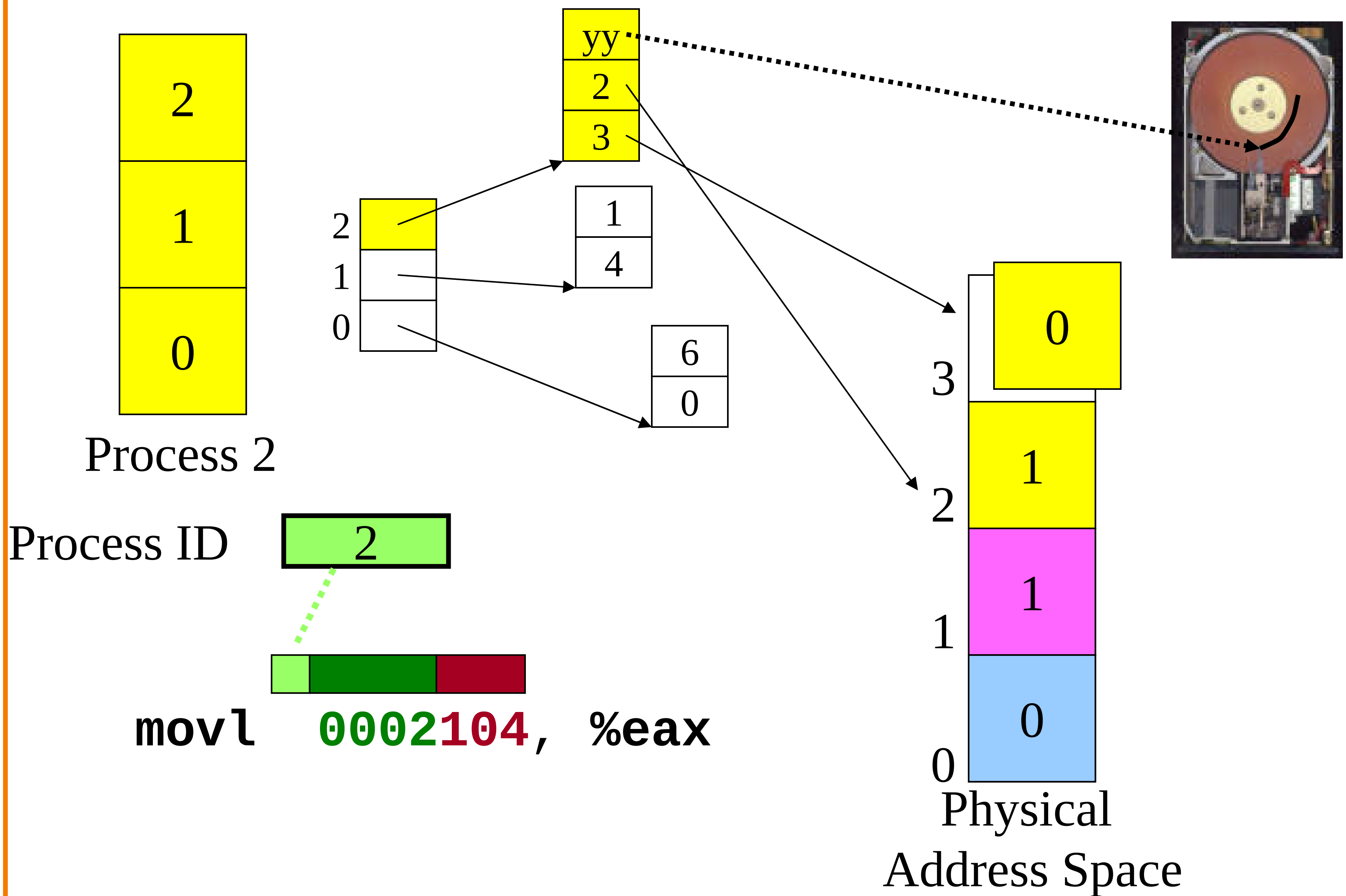
2
1
0
Process 2
2
1
0
3
2
xx
1
4
6
0
Process ID
2
movl 0002104, %eax
3
2
1
0
2
1
1
0
Physical
Address Space

# Write Some Other Page to Disk




```
movl  0002104, %eax
```

# Fetch Current Page, Adjust Page Tables



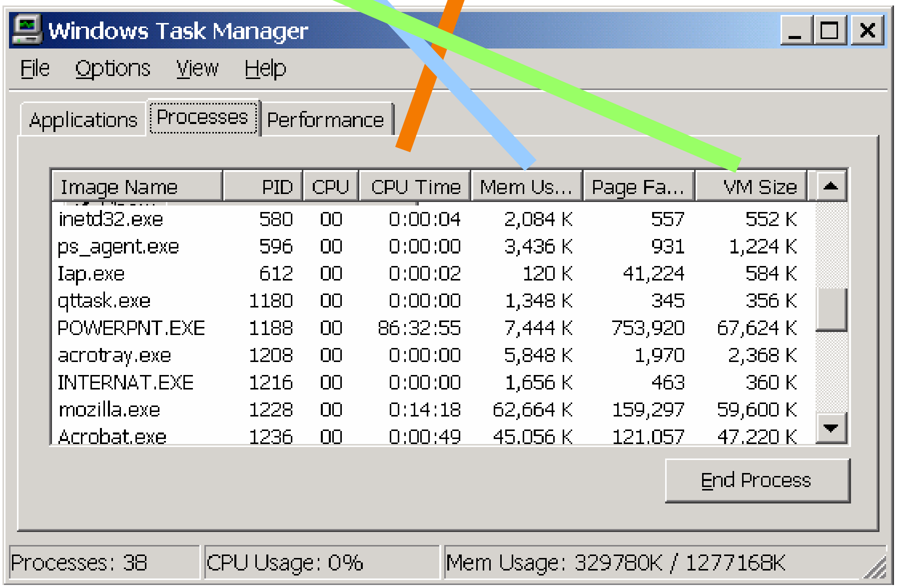
# Measuring the Memory Usage

Virtual memory usage
Physical memory usage ("resident set size")
CPU time used by this process so far

Unix

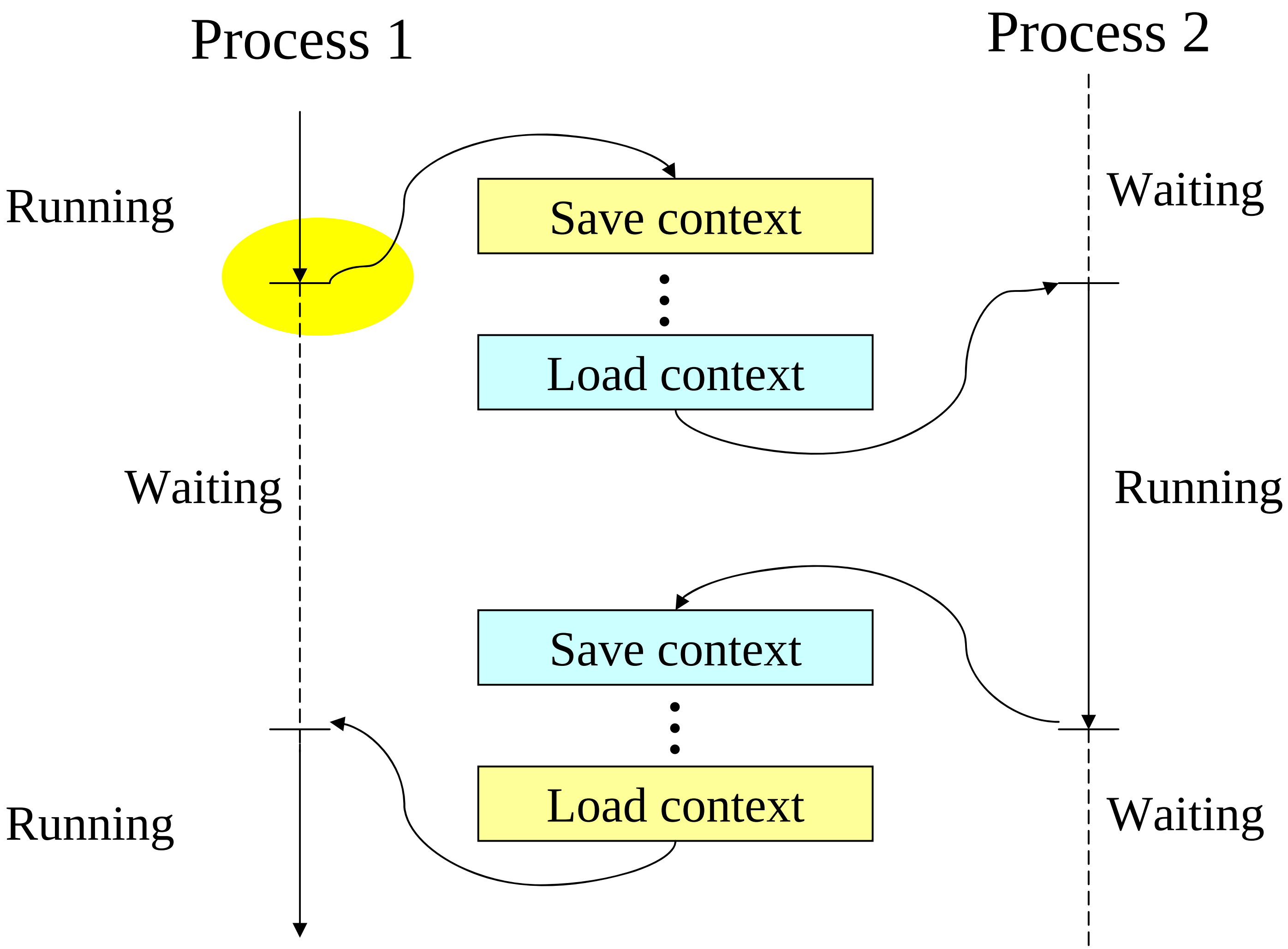
```
% ps l
F  UID   PID  PPID PRI   VSZ   RSS STAT  TIME COMMAND
0  115  7264  7262  17  4716  1400  SN   0:00 -csh
0  115  7290  7264  17 15380 10940  SN   5:52 emacs
0  115  3283  7264  23  2864   812  RN   0:00 ps l
```

Windows




| Image Name | PID | CPU | CPU Time | Mem Us... | Page Fa... | VM Size |
|---|---|---|---|---|---|---|
| inetd32.exe | 580 | 00 | 0:00:04 | 2,084 K | 557 | 552 K |
| ps_agent.exe | 596 | 00 | 0:00:00 | 3,436 K | 931 | 1,224 K |
| Iap.exe | 612 | 00 | 0:00:02 | 120 K | 41,224 | 584 K |
| qttask.exe | 1180 | 00 | 0:00:00 | 1,348 K | 345 | 356 K |
| POWERPNT.EXE | 1188 | 00 | 86:32:55 | 7,444 K | 753,920 | 67,624 K |
| acrotray.exe | 1208 | 00 | 0:00:00 | 5,848 K | 1,970 | 2,368 K |
| INTERNAT.EXE | 1216 | 00 | 0:00:00 | 1,656 K | 463 | 360 K |
| mozilla.exe | 1228 | 00 | 0:14:18 | 62,664 K | 159,297 | 59,600 K |
| Acrobat.exe | 1236 | 00 | 0:00:49 | 45,056 K | 121,057 | 47,220 K |

# Context Switch, in More Detail



# Context Switch, in More Detail

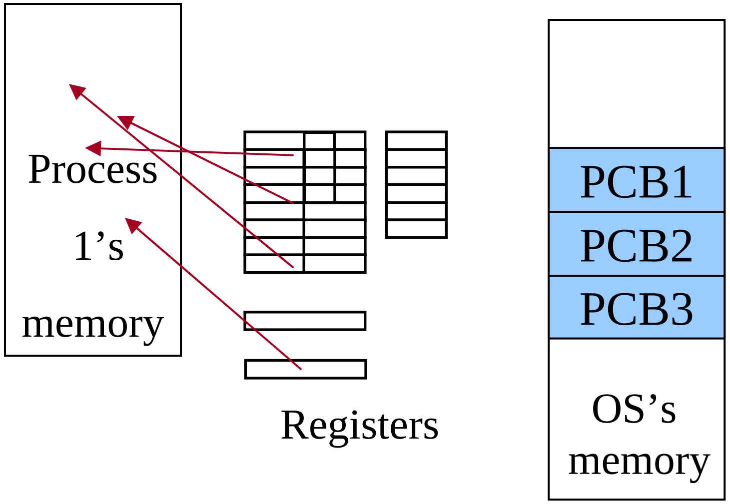
Process 1

Running

page fault

```
addl %eax, %ecx
movl -8(%ebp), %eax
addl %eax, %ecx
.
.
.
```

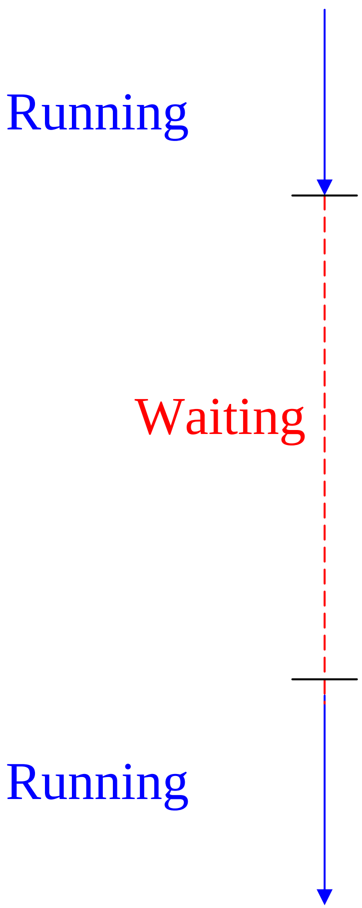
Waiting




Running

# Context Switch, in More Detail

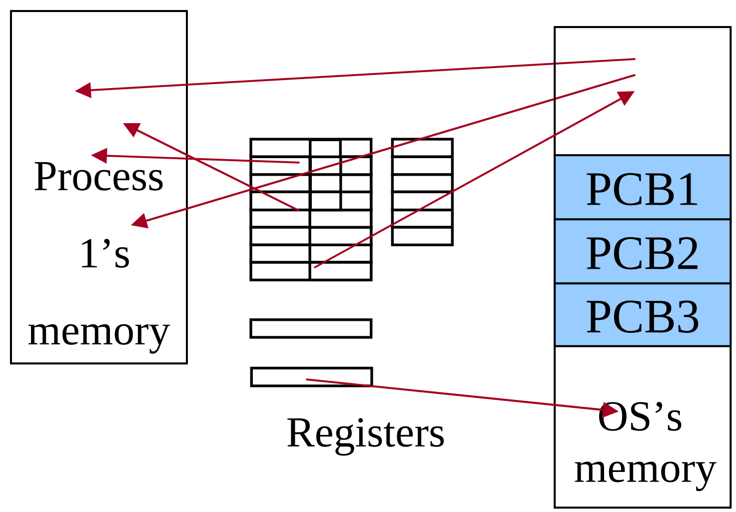
Process 1




```
addl %eax, %ecx
movl -8(%ebp), %eax
addl %eax, %ecx
.
.
.
```

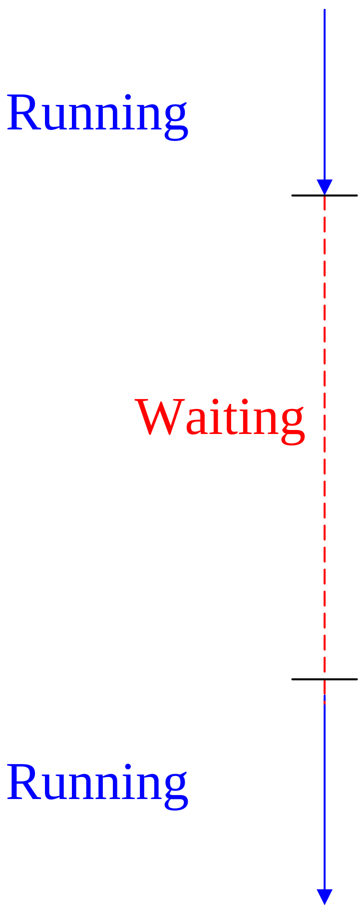
## Fault-handler hardware

1. Enters privileged mode
2. Sets EIP to specific location in operating system
3. Sets ESP to operating-system stack in OS memory
4. Pushes old (process 1) EIP and ESP on OS stack



# Context Switch, in More Detail

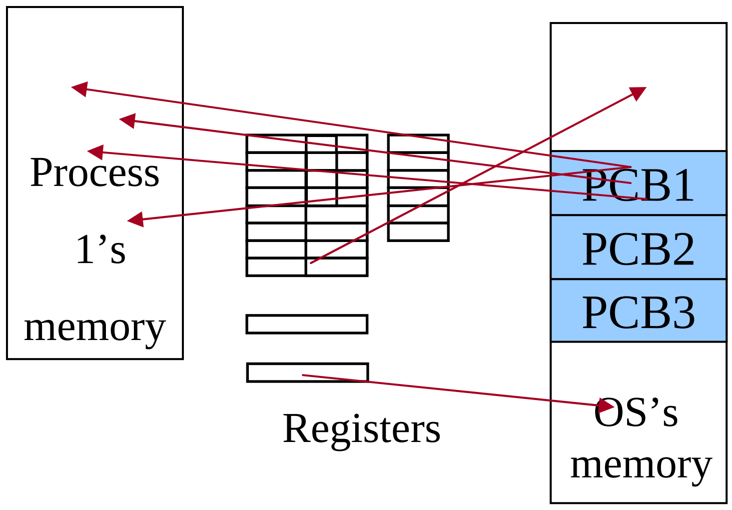
Process 1




```
addl %eax, %ecx
movl -8(%ebp), %eax
addl %eax, %ecx
.
.
.
```

## OS software

5. Pops saved EIP,ESP into PCB1
6. Copies rest of registers into PCB1
7. Sends instructions to disk drive to fetch page



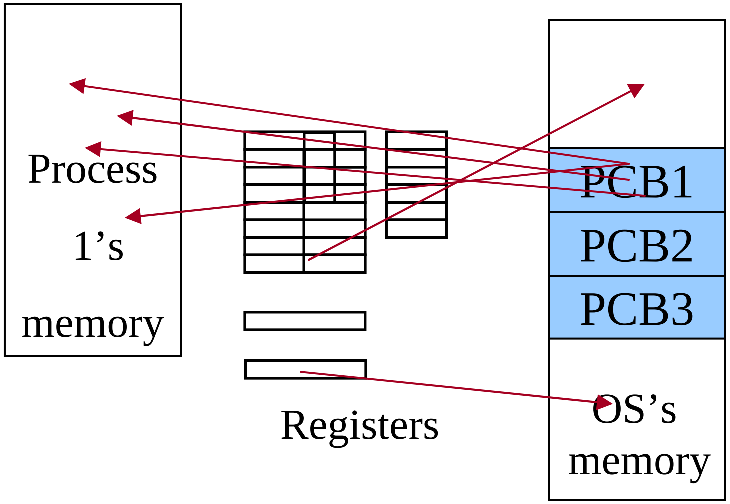
# Resuming Some Other Process

## OS software

8. Sets process-ID register to 2
9. Pushes saved EIP,ESP from PCB2 onto OS stack
10. Copies rest of registers from PCB2
11. Executes “return from interrupt” instruction

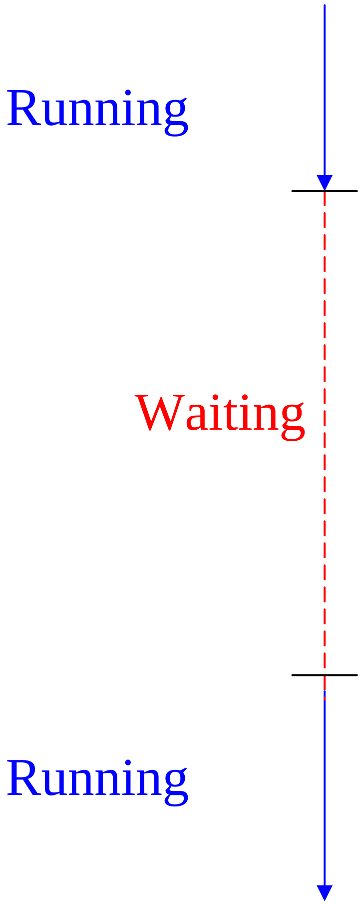
## Hardware

12. Pops EIP,ESP into registers
13. Switches back to unprivileged mode
14. Resumes where process 2 left off last time



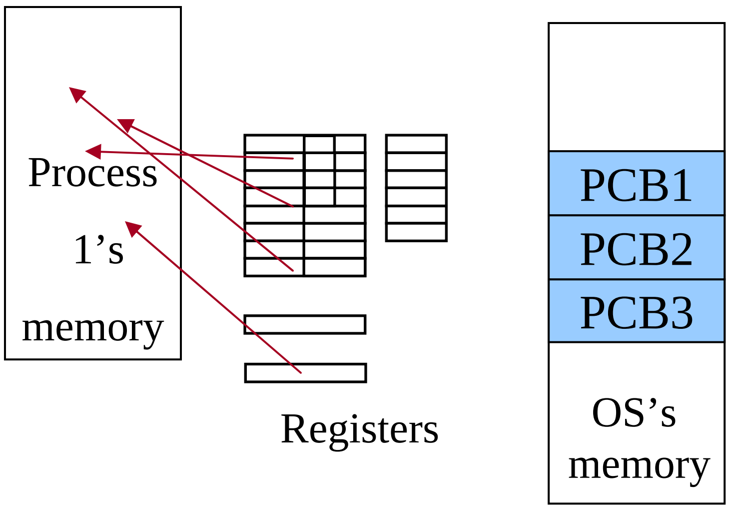
# System call, just another kind of fault

Process 1




```
mov     $4,%eax
int     $0x80
addl %eax, %ecx
.
.
.
```

system call
(privileged instruction)



# Summary

- Abstraction of a "process"
  - CPU: a share of CPU resources on a small time scale
  - Memory: a complete address space of your own
- OS support for the process abstraction
  - CPU: context switch between processes
  - Memory: virtual memory (VM) and page replacement
  - Files: open/read/write, rather than "move disk head"
  - Protection: ensure process access only its own resources
- Hardware support for the process abstraction
  - Context switches, and push/pop registers on the stack
  - Switch between privileged and unprivileged modes
  - Map VM address and process ID to physical memory