CS5460/6460: Operating Systems

Lecture 1: Introduction

Anton Burtsev
January, 2014

Class details

- Mixed undergraduate and graduate
  - 125 people (waiting list: 8 students)
- Instructor: Anton Burtsev
- 4 TAs
  - Saurav Singh, Sriraam Appusamy Subramanian
  - Scotty Bauer, Sarah Spall
- Web page
  - http://www.cs.utah.edu/~aburtsev/cs5460

This course

- Based on
  - MIT 6.828: Operating System Engineering
    http://pdos.csail.mit.edu/6.828/2012/overview.html
- We will use xv6
  - Relatively simple (9K lines of code)
  - Reasonably complete UNIX kernel
  - http://pdos.csail.mit.edu/6.828/2012/xv6/xv6-rev7.pdf
- xv6 comes with a book
  - http://pdos.csail.mit.edu/6.828/2012/xv6/book-rev7.pdf

Course organization

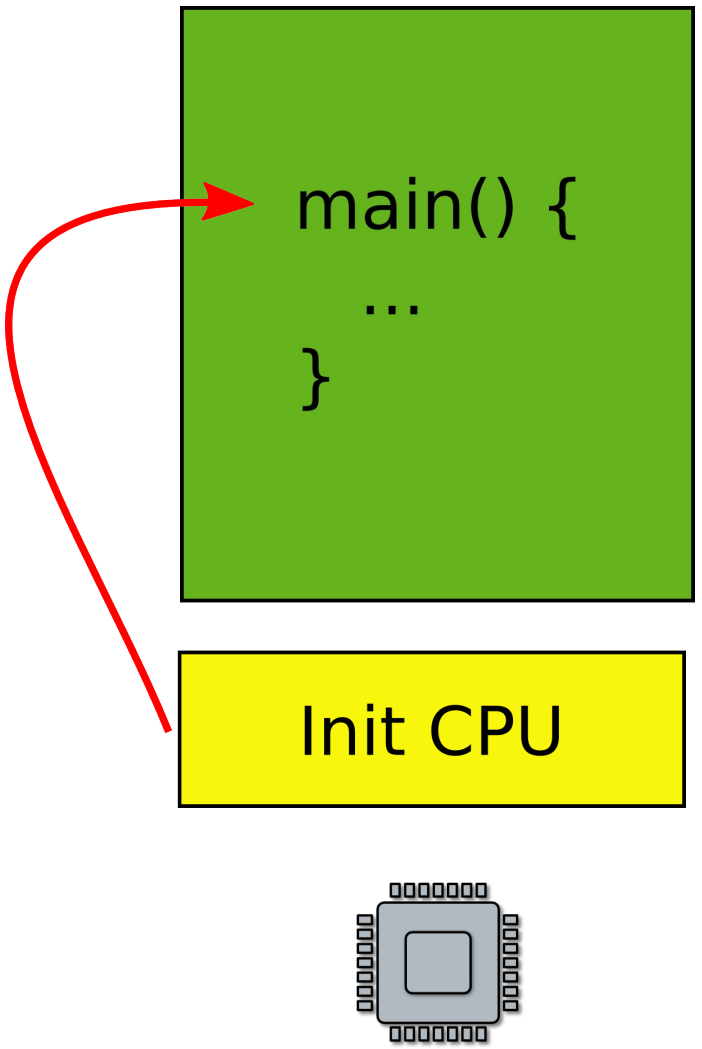
- Lectures
  - High level concepts and abstractions
- Reading
  - Xv6 book + source code
- Labs
  - Coding real parts of the xv6 kernel
- Design riddles
  - Understanding tradeoffs, explaining parts of xv6

Prerequisites

- Solid C coding skills
  - Xv6 is written in C
  - You need to read, code and debug
  - All labs are in C
  - Many questions about explaining xv6 code
- Be able to code in Linux
- Some assembly skills

What is an operating system?

- Want to run your code on a piece of hardware

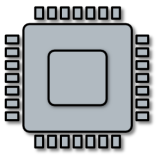


- GCC + some linking magic
  - Crashes....
- Read manual
- Code a tiny boot layer
  - Initialize CPU, memory
  - Jump to your code
- **This is your OS!**

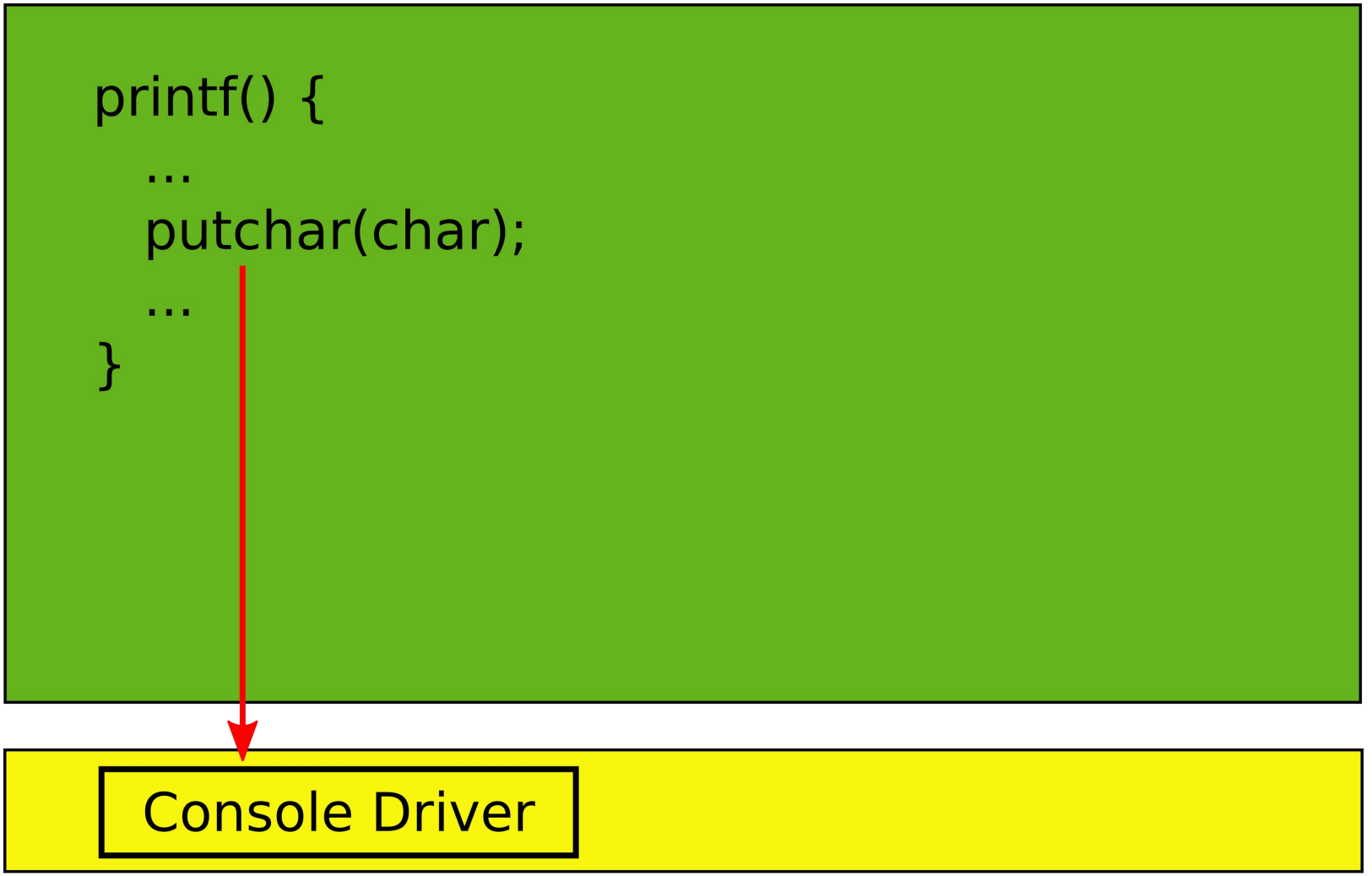
- Want to print out a string
  - On the screen or serial line

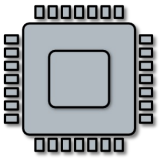
```
printf() {
    ...
    if (vga) {
        asm("mov <magic number 1>, char");
    } else if (serial) {
        asm("out <magic number 2>, char");
    }
    ...
}
```

OS



- Implement a general function!
  - First device driver

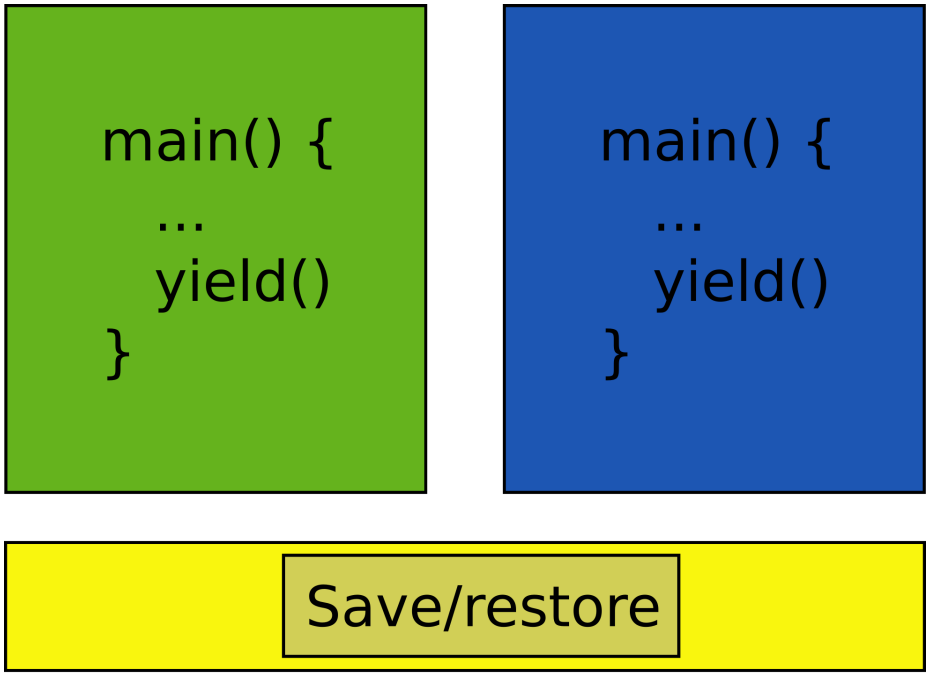




Device drivers

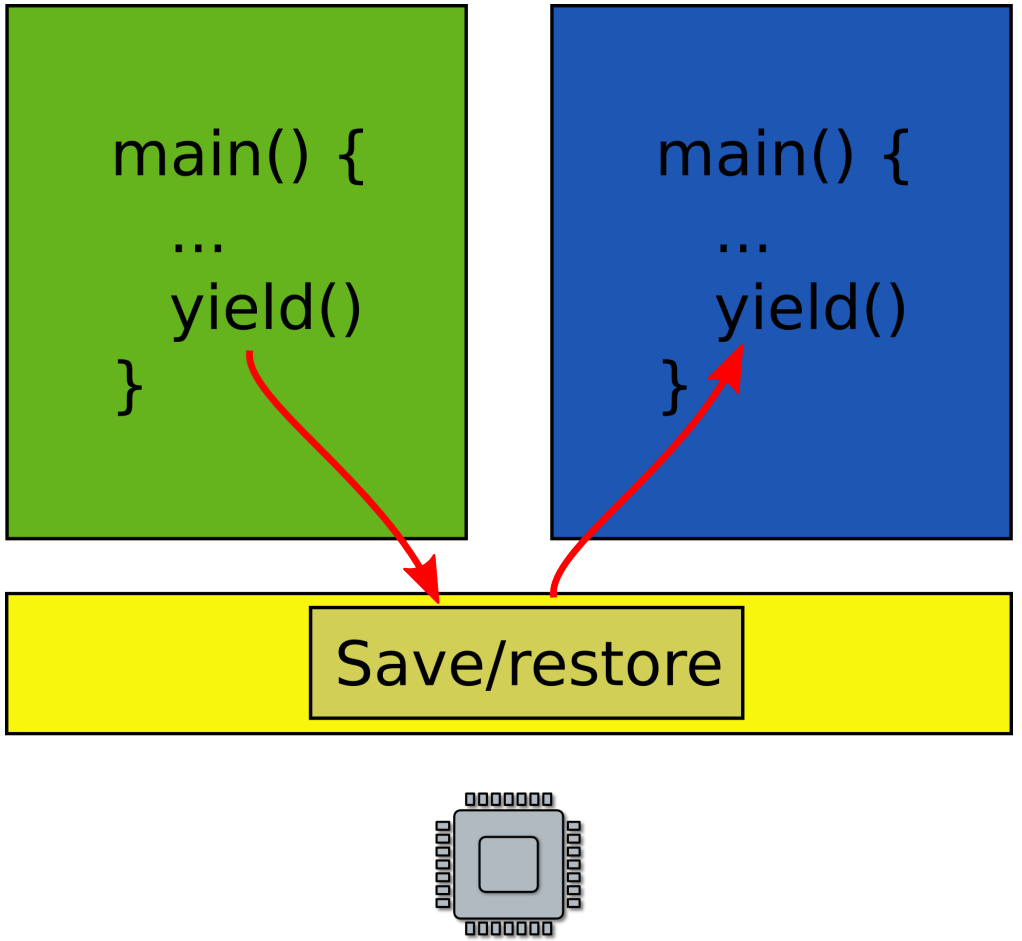
- Abstract hardware
  - Provide high-level interface
  - Hide minor differences
  - Implement some optimizations
    - Batch requests
- Examples
  - Console, disk, network interface
  - ...virtually any hardware you know

- Want to run two programs



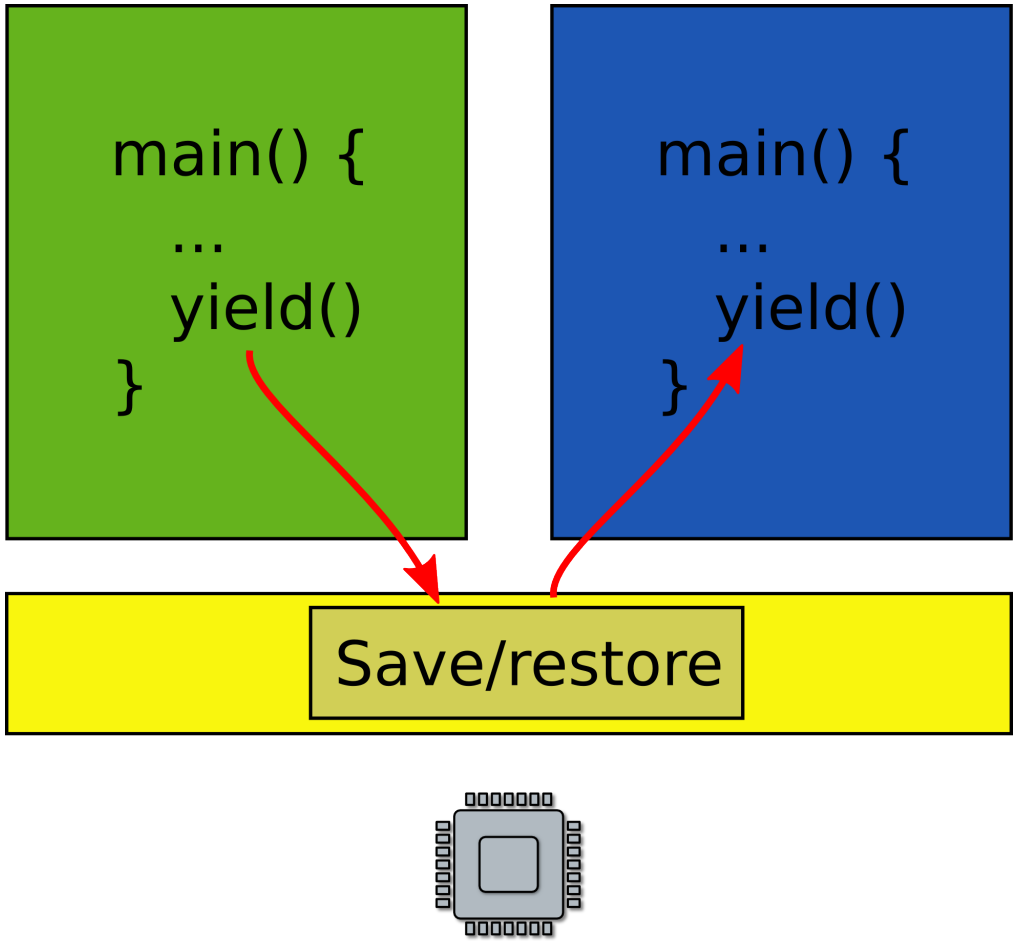
- What does it mean?
  - Only one CPU
- Run one, then run another one

- Want to run two programs



- Exit into the kernel periodically
- Context switch
  - Save and restore context
  - Essentially registers

- What! Two programs in one memory?

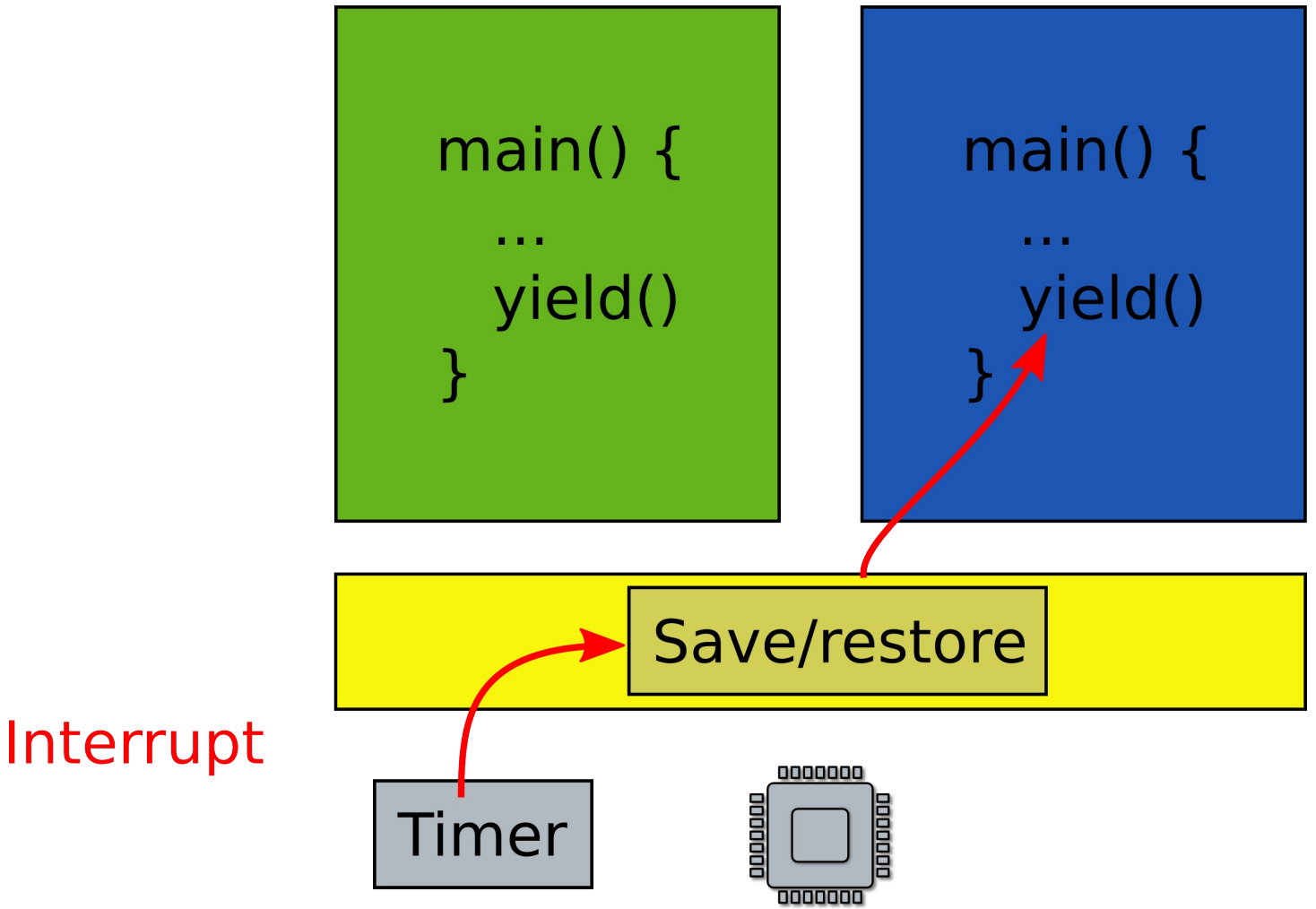


- Am I cheating with the linker?

Virtual address spaces

- Illusion of a private memory for each application
  - Keep a description of an address space
  - In one of the registers
- All normal program addresses are inside the address space
- OS maintains description of address spaces
  - Switches between them

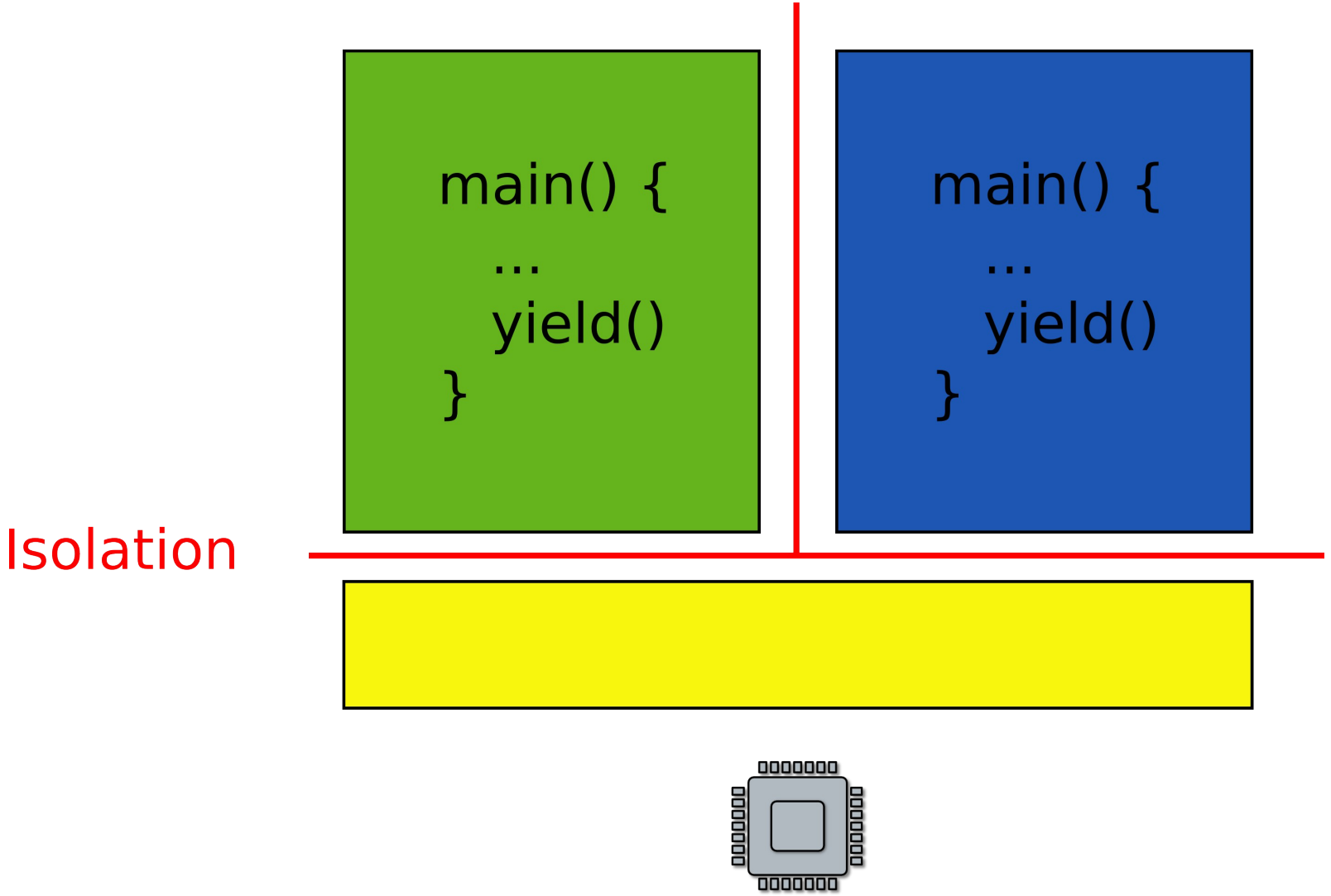
- What if one program fails to release the CPU?
- It run forever. Need a way to preempt it. How?



Scheduling

- Pick which application to run next
  - And for how long
- Illusion of a private CPU for each task
  - Frequent context switching

- What if one program corrupts the kernel? Or other programs?

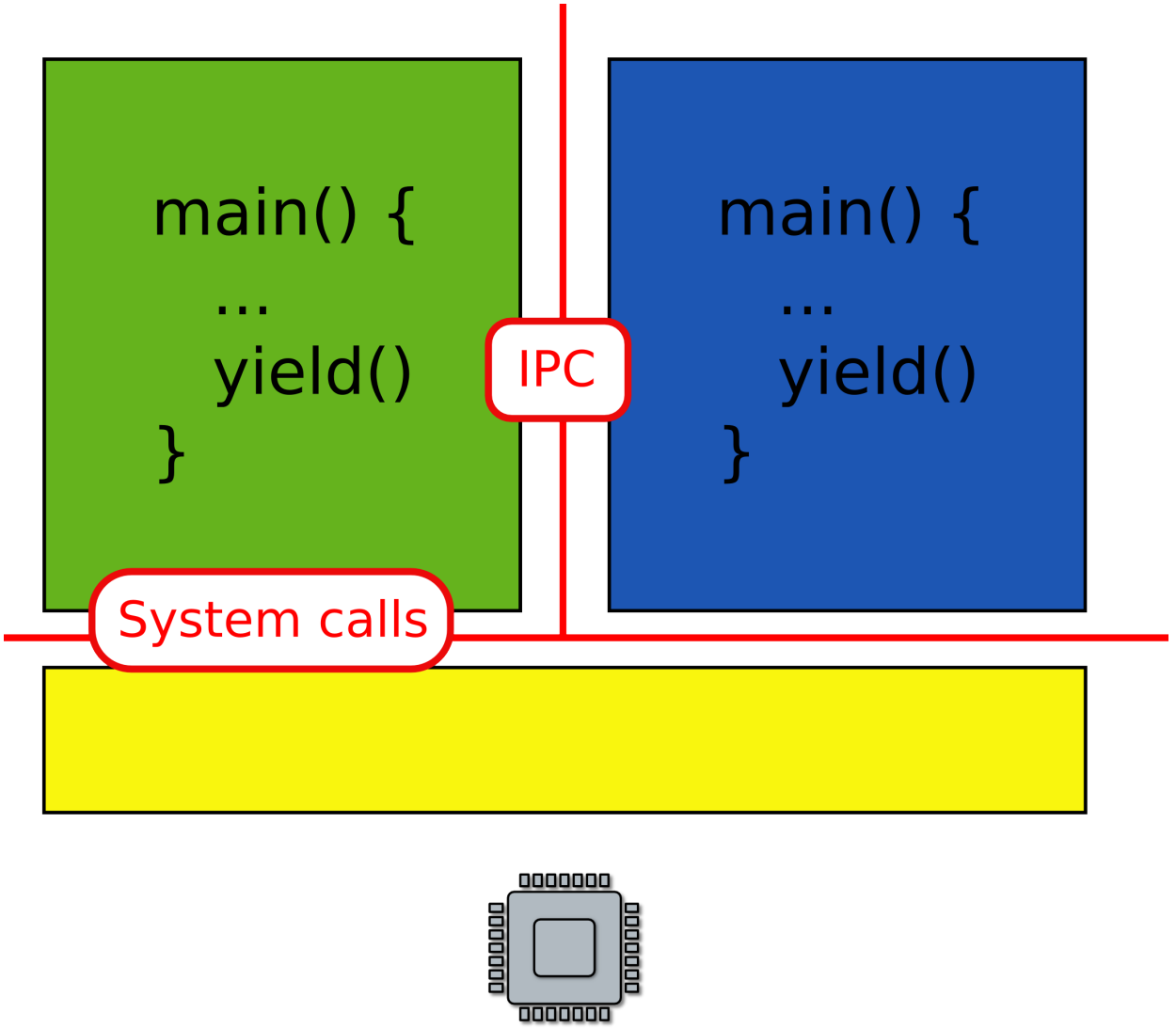


Isolation

- Today is done with address spaces in hardware
  - Many issues, e.g. shared device drivers, files, etc.

- Can it be done in software?

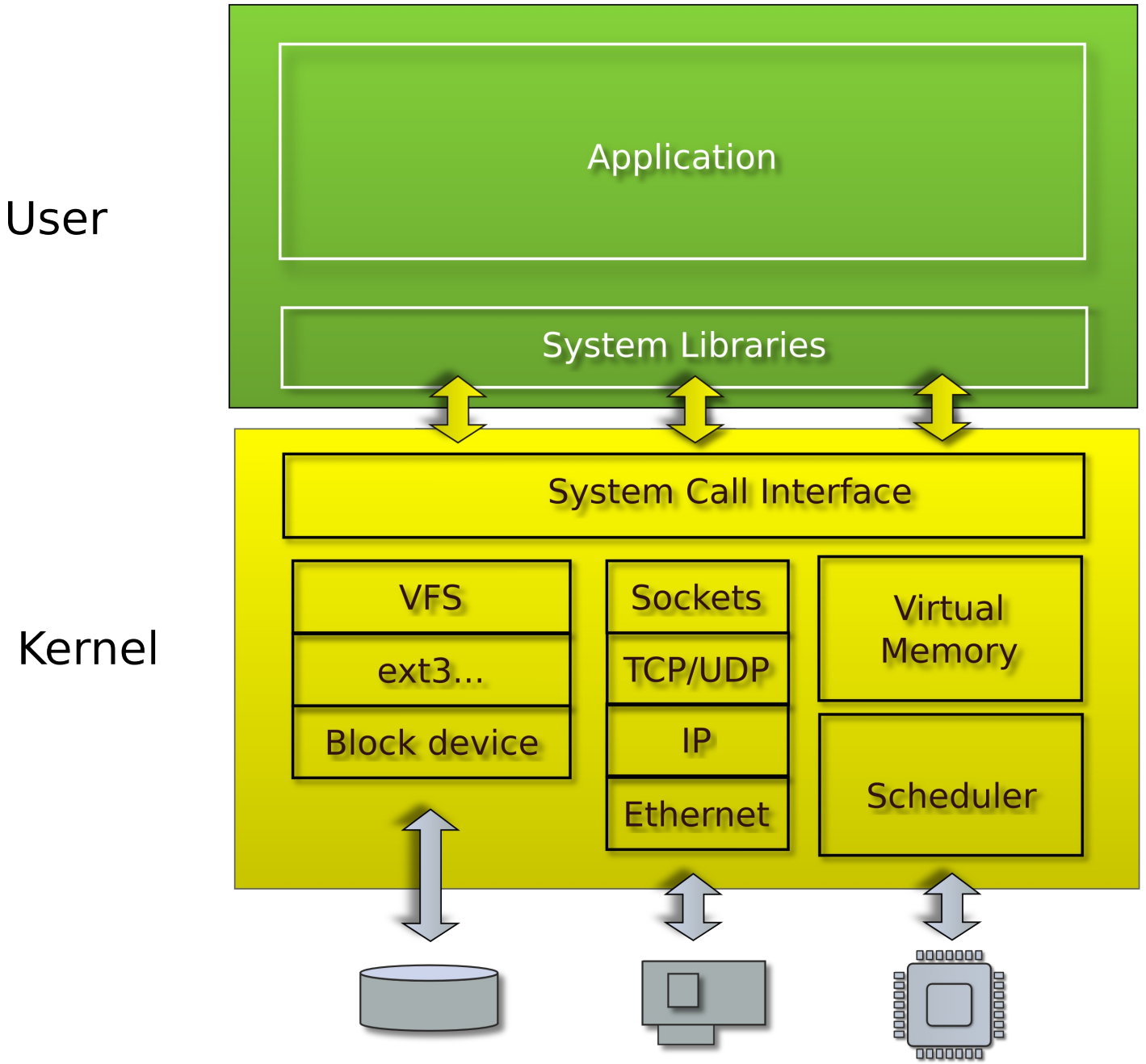
- What about communication?
- Can we invoke a function in a kernel?



- What if you want to save some data?
- Permanent storage
  - E.g., disks
- But disks are just arrays of blocks
  - wrtie(block_number, block_data)
- Files
  - High level abstraction for saving data
  - fd = open(“contacts.txt”);
  - fpritnf(fd, “Name:%s\n”, name);

- What if you want to send data over the network?
- Network interfaces
  - Send/receive Ethernet packets (Level 2)
  - Two low level
- Sockets
  - High level abstraction for sending data

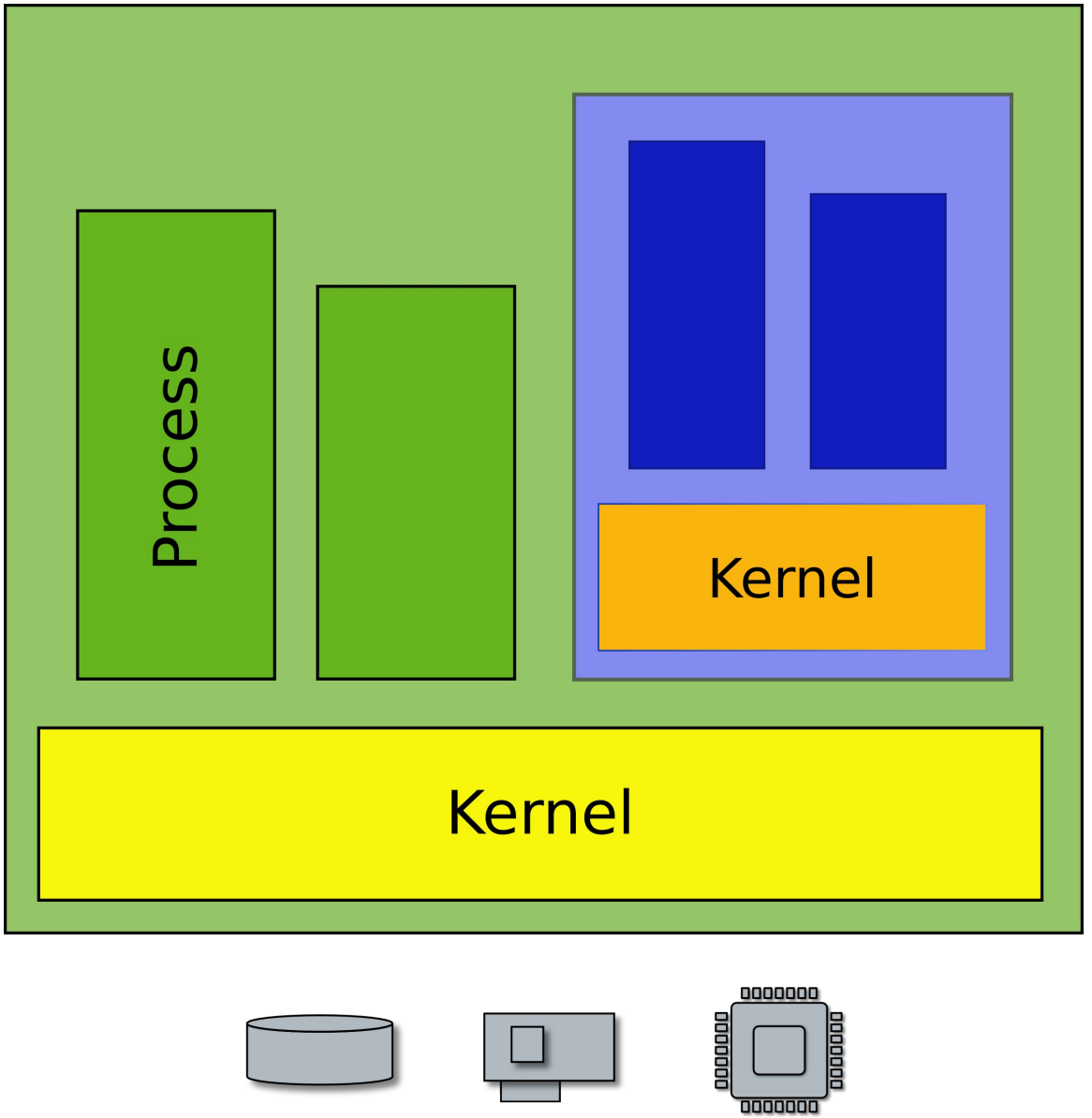
- Linux/Windows/Mac



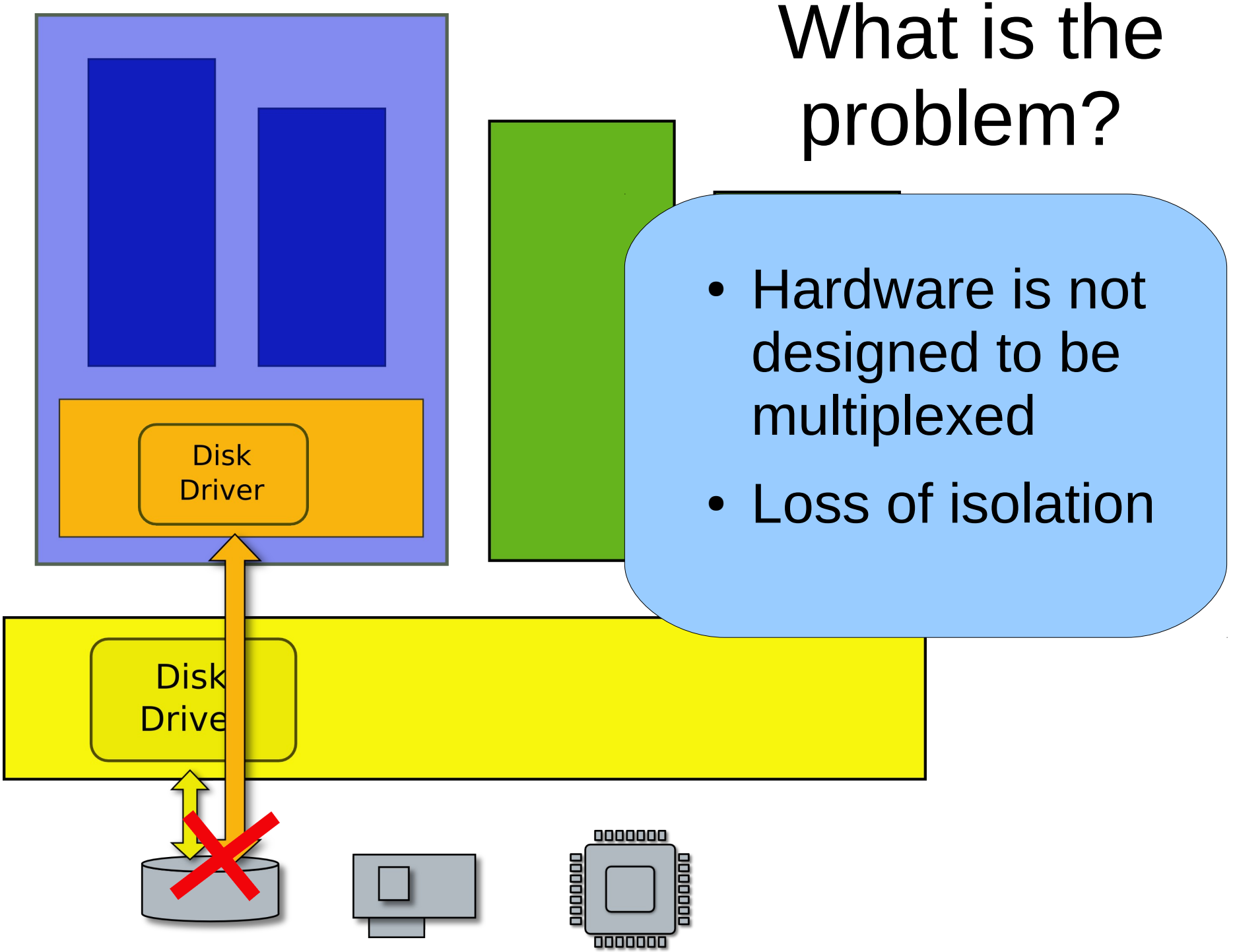
Multiple levels of abstraction

- Multiple programs
  - Each has illusion of a private memory and CPU
  - Context switching, scheduling, isolation, communication
- File systems
  - Multiple files, concurrent I/O requests
  - Consistency, caching
- Network protocols
  - Multiple virtual network connections
- Memory management

- Want to run a Windows application on Linux?

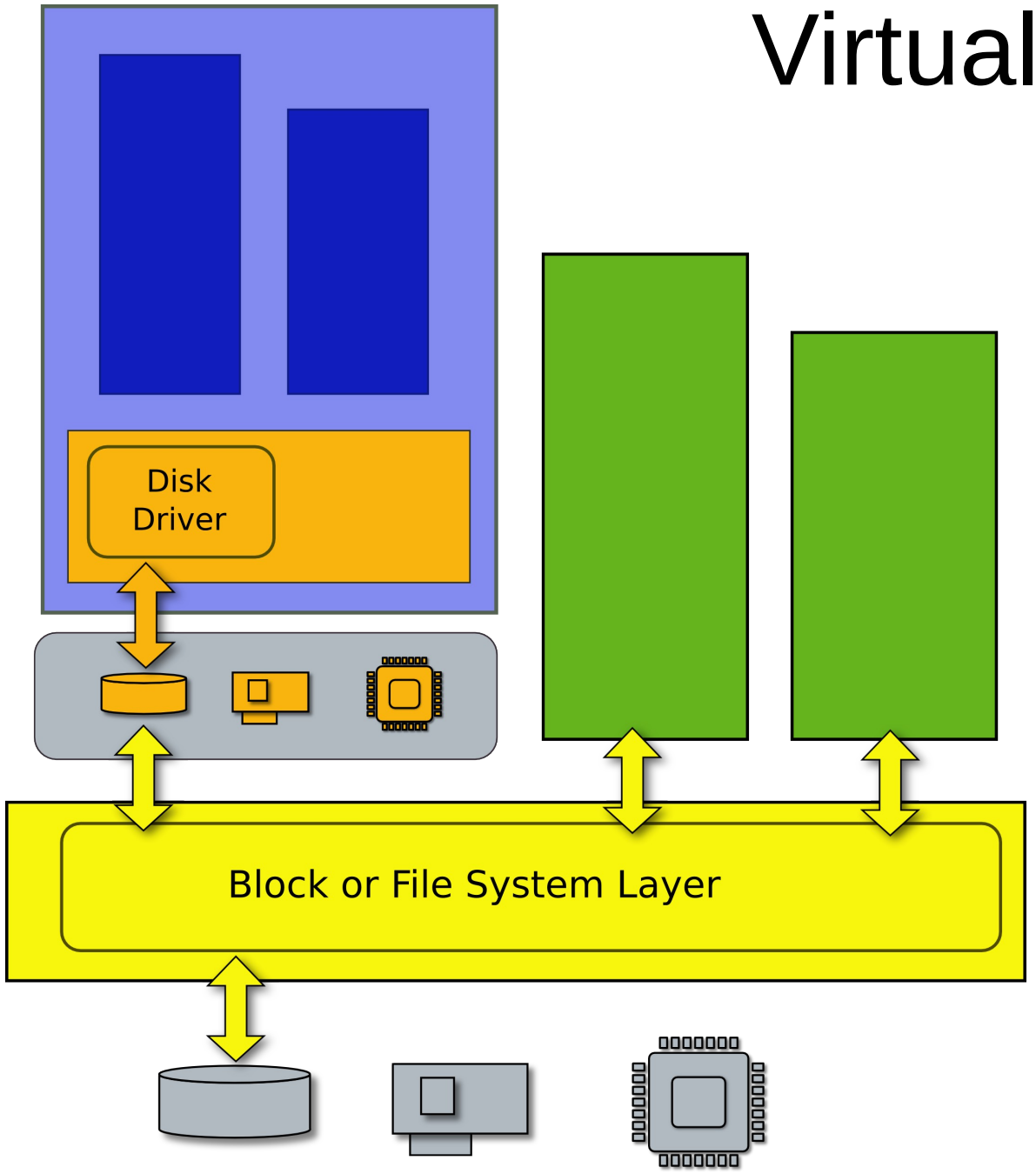


What is the problem?



- Hardware is not designed to be multiplexed
- Loss of isolation

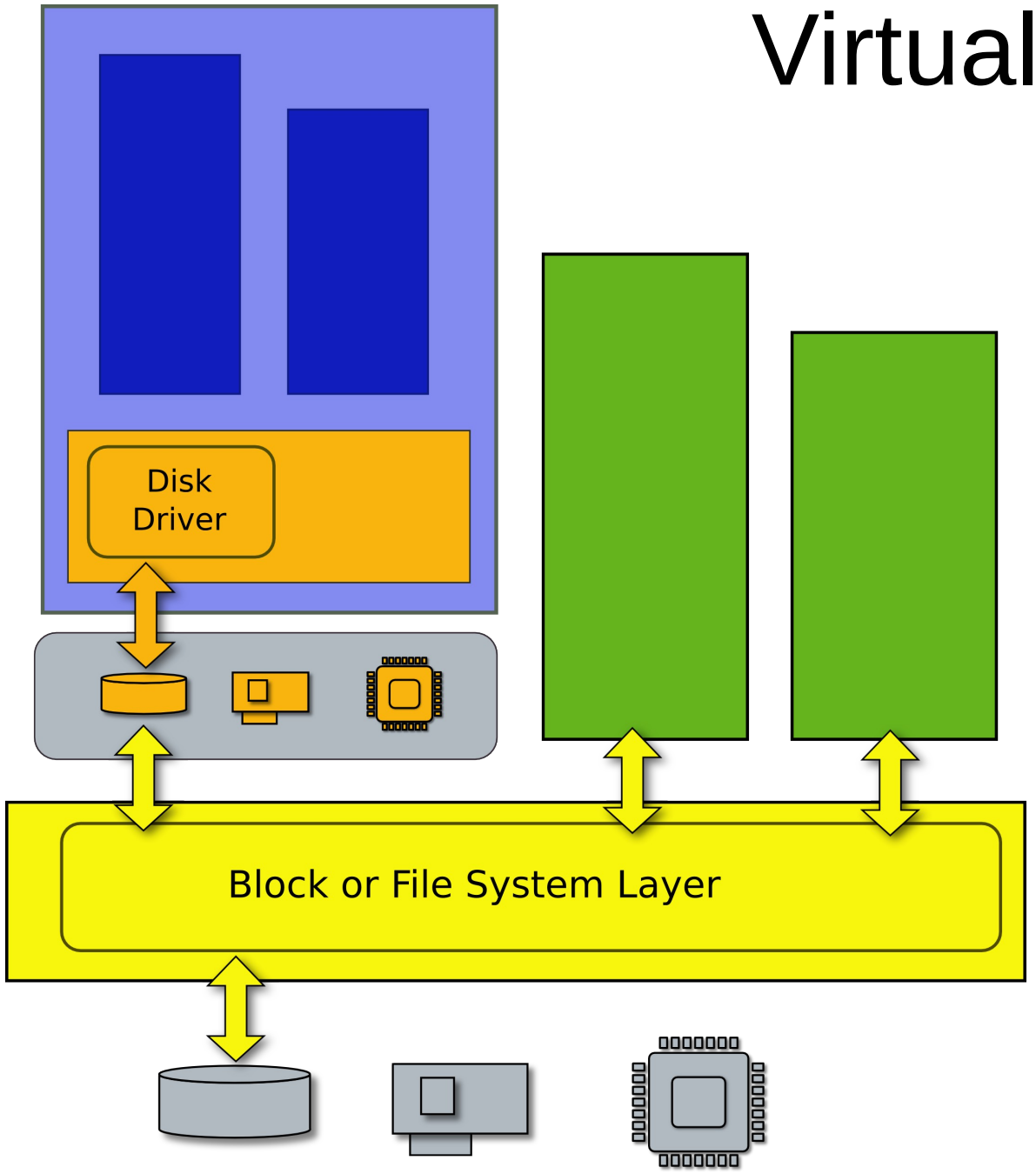
Virtual machine



Efficient duplicate of a real machine

- Compatibility
- Performance
- Isolation

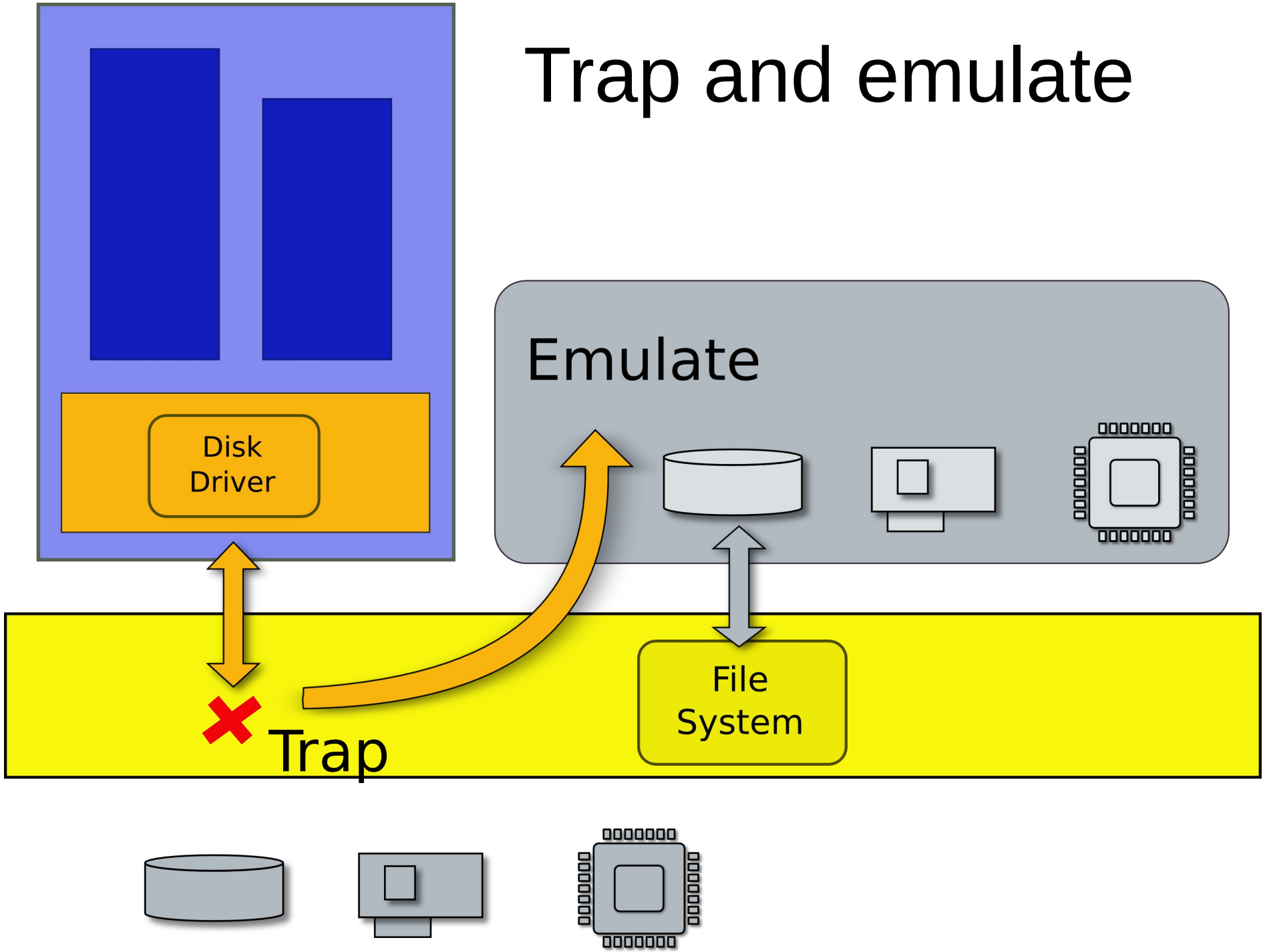
Virtual machine



Efficient duplicate of a real machine

- Compatibility
- Performance
- Isolation

Trap and emulate



What hardware do we have today?

- 8 cores (2 Ghz)
  - Each runs two threads
- 64 Gig of RAM
  - 20MB Level 3 cache
- 10 Gbps network controllers

Intel supports 80 millions of packets per second

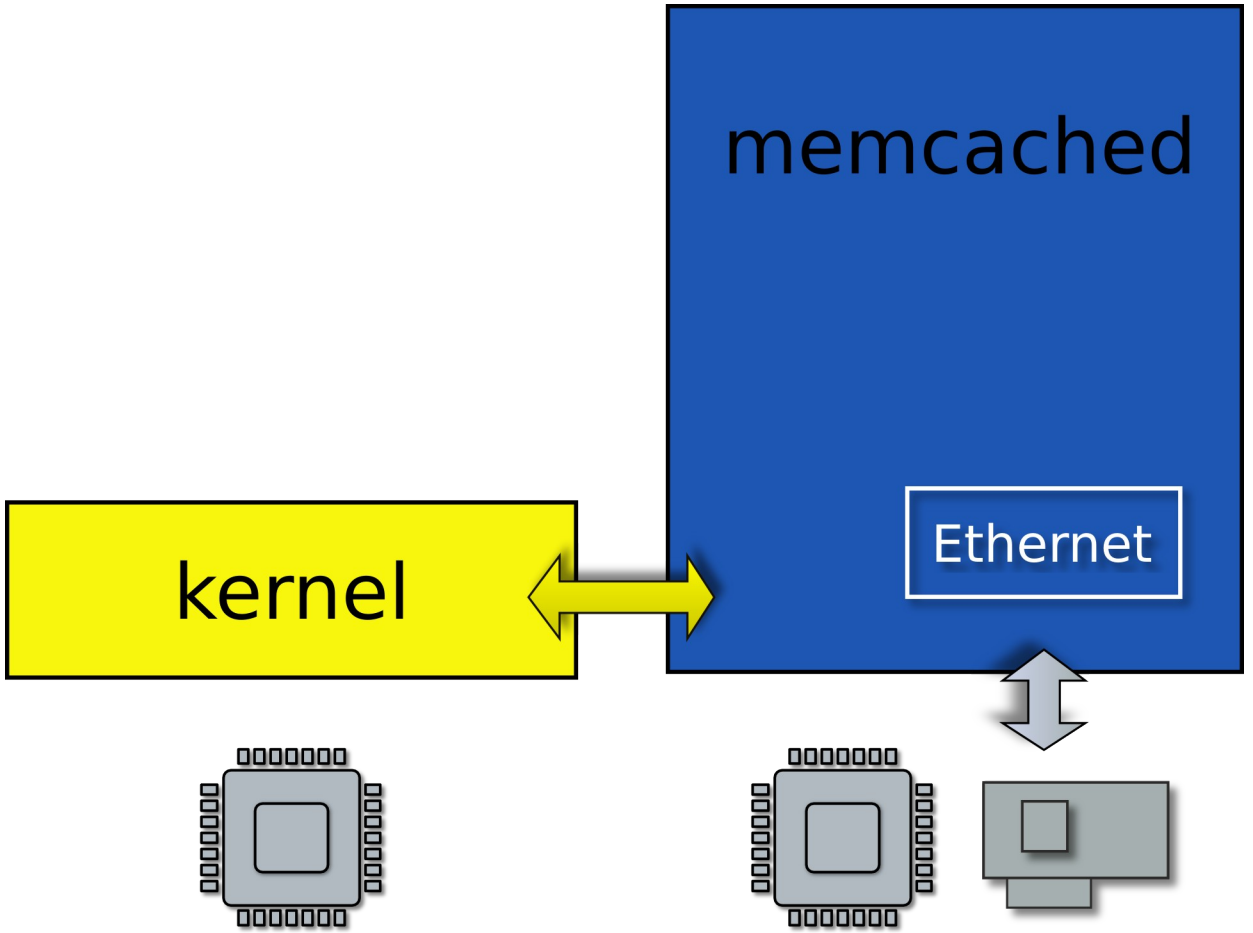
- L3 forwarding of 64 byte packets

Lets do the math:

- How many cycles per packet?
  - 1/80 000 000 = 1.25e-80 seconds per packet on 1 core
- Or in nanoseconds:
  - 12.5 nanoseconds per packet
- Or in cycles (remember 2Ghz, or 2 cycles per nanosecond)
  - 25 cycles per packet
- But we have 8 cores, so
  - 25*8 = 200 cycles per packet

Can you allow an OS on the critical path?

- Layer 2 → IP → TCP → Sockets → User-level?
- No! You want to run bare metal!



Ok, so what is an operating system?

References

- Impressive Packet Processing Performance Enables Greater Workload Consolidation. Intel. http://www.intel.com/content/dam/www/public/us/en
- Intel Data Plane Development Kit (Intel DPDK)

  http://www.intel.com/content/dam/www/public/us/en