ECS 251: History of OS

Sam King

Administrative

Overview

- Trends and history of OS
- OS architectures
- OS interfaces
- Trends moving forward

History of operating systems

- OS history dominated by two trends
  - Expensive hardware becoming cheap
  - OS complexity increasing

- OS is simple, library of services, one “thread”
- Poor utilization

Mainframes and single operator

- Goal: just get it to work!
- Lots of wasted time where expensive computers idle

http://www.mainframes360.com/2009/06/what-is-mainframe-computer.html

Batch processing

Batch processing

- Goal: improve CPU and I/O utilization
- Solution: Remove human from the pipeline
  - Encourages a very different style of programming

- Still only one job at a time
  - Batched together to form a type of a pipeline
  - Input encoding stage, execution stage, output stage all decoupled
- OS is batch monitor and library of services
- Batch monitor loads, runs programs
- For the first time protection becomes an issue

- Why was protection not an issue for single operator?

Multi-programmed batch

- Goal: improve CPU and I/O utilization
- Solution: overlap disk I/O and CPU

- Allows multiple I/Os to happen simultaneously
- Allows CPU and disk to work simultaneously
- OS getting more complex
  - Switch between tasks, manage I/O
  - Now, must protect tasks for each other

Time sharing

- Goal: restore ability for humans to interact
- Insight: human is modeled as a (very slow) I/O
- Solution: switch when waiting for human
- OS getting more complex
  - Lots of jobs
  - Jobs coming from different places
  - Mechanisms and policies to cope with users

Personal computing

- Computer hardware cheap
  - Single operator at console
  - OS reverts to subroutine library
  - Do you need time-share between multiple jobs?

  - Do you need protection?

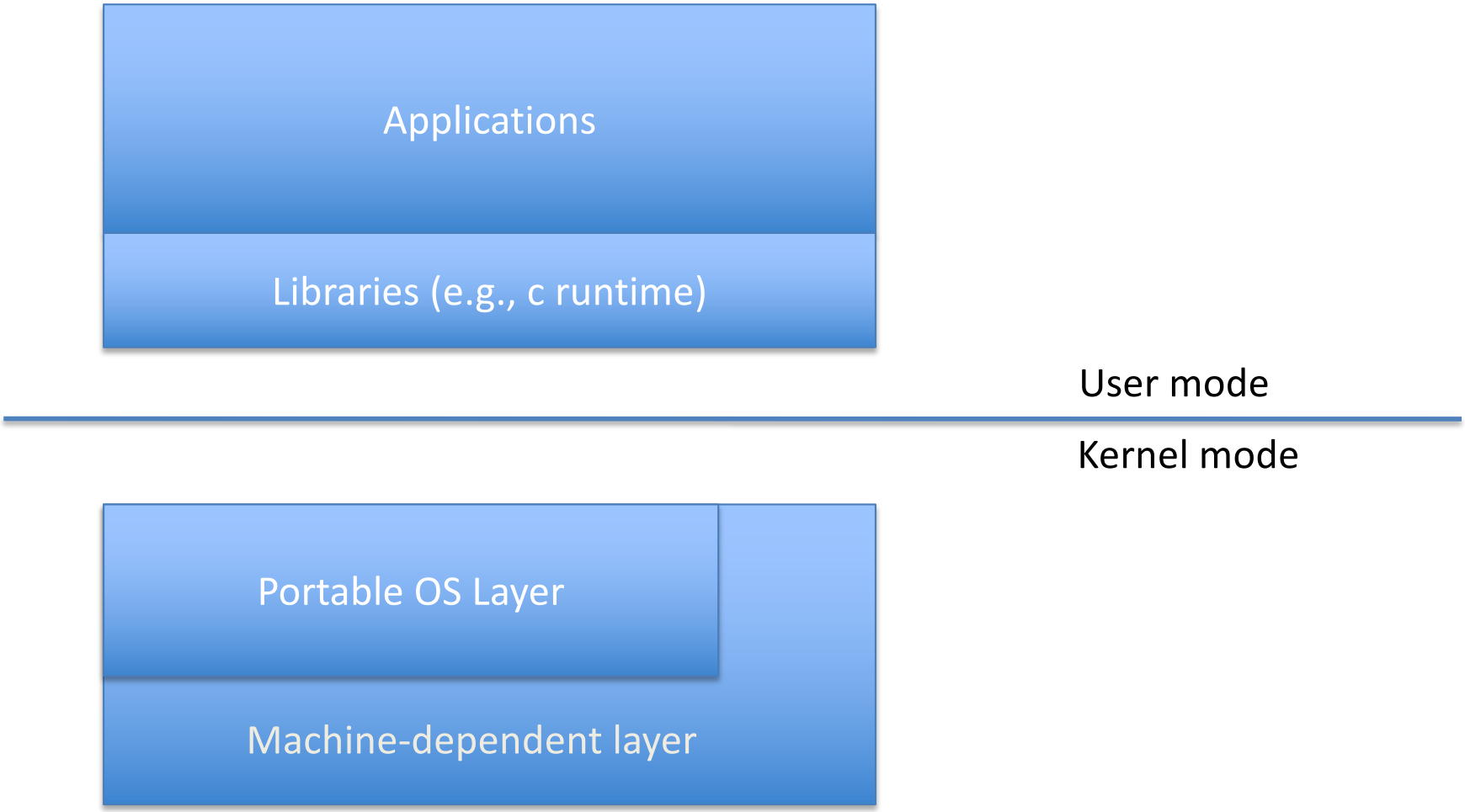
- PC operating systems have gradually added back in features from time-sharing systems

Today's computing

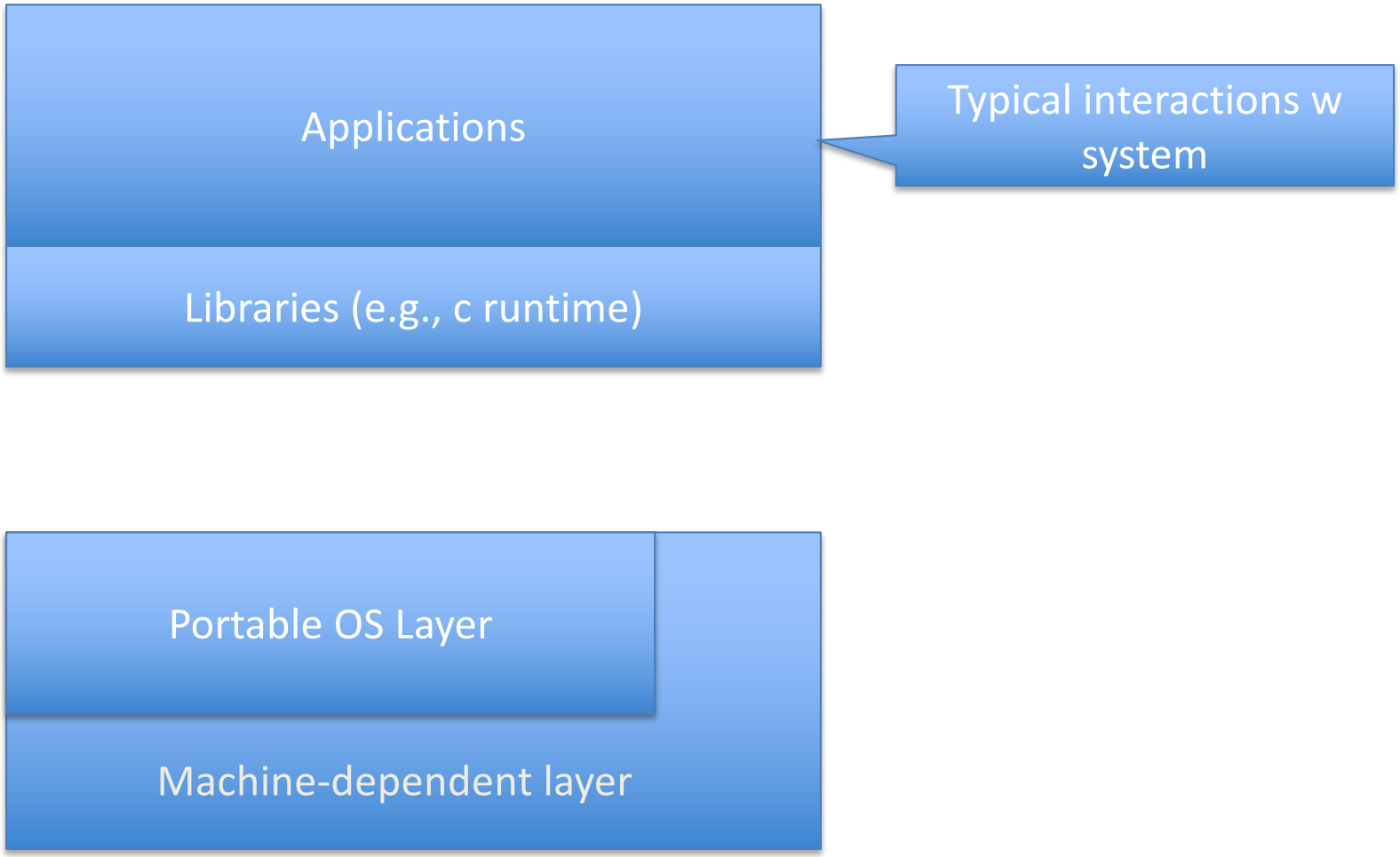
- OS are massively complex
  - Windows XP kernel had 45M loc https://www.facebook.com/windows/posts/155741344475532
- APIs and practical distributed systems
  - E.g., Spanner
- OS abstractions are getting *harder* to work with
  - Mobile devices for security and power saving

OS architectures

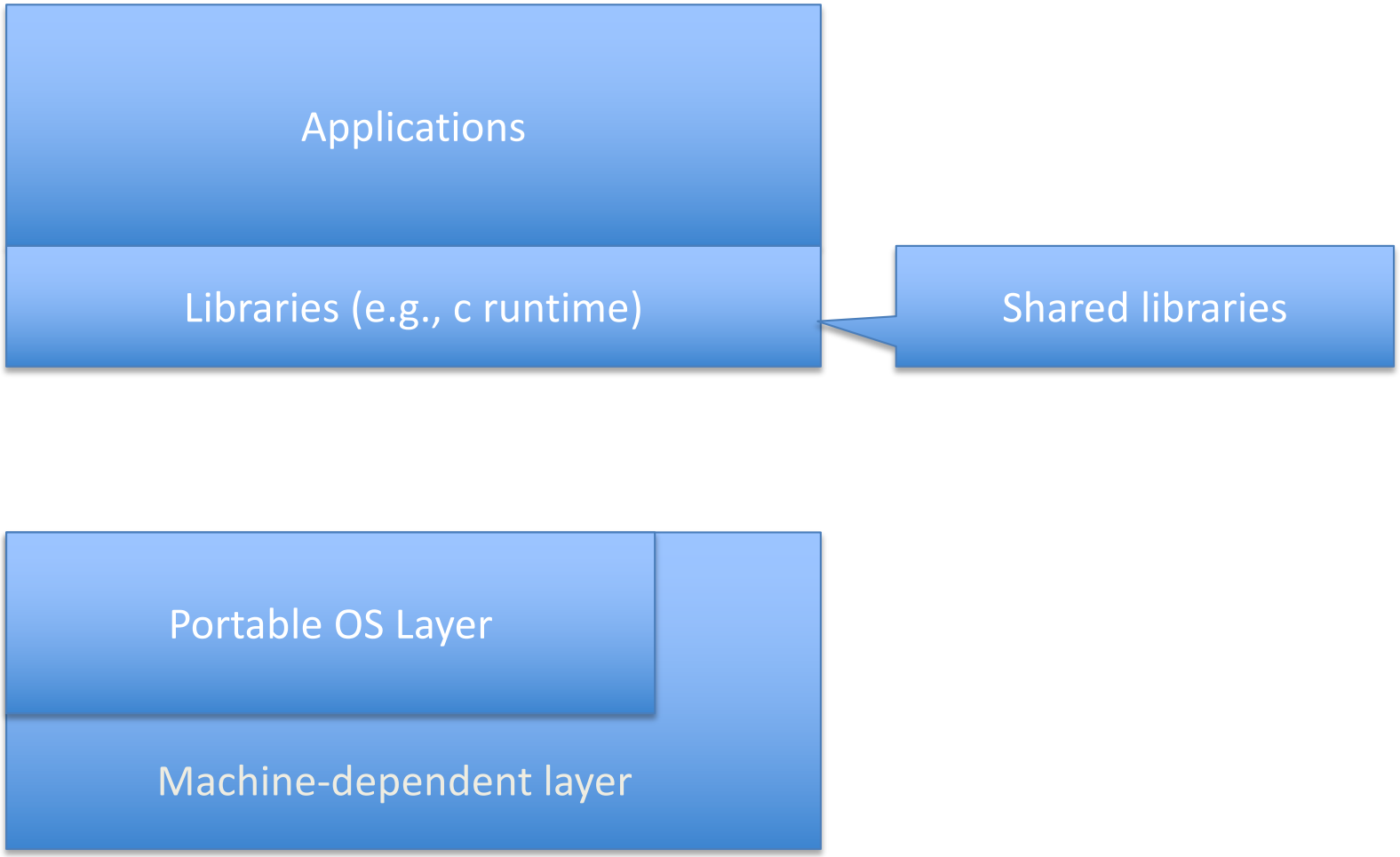
A peek into Unix/Linux



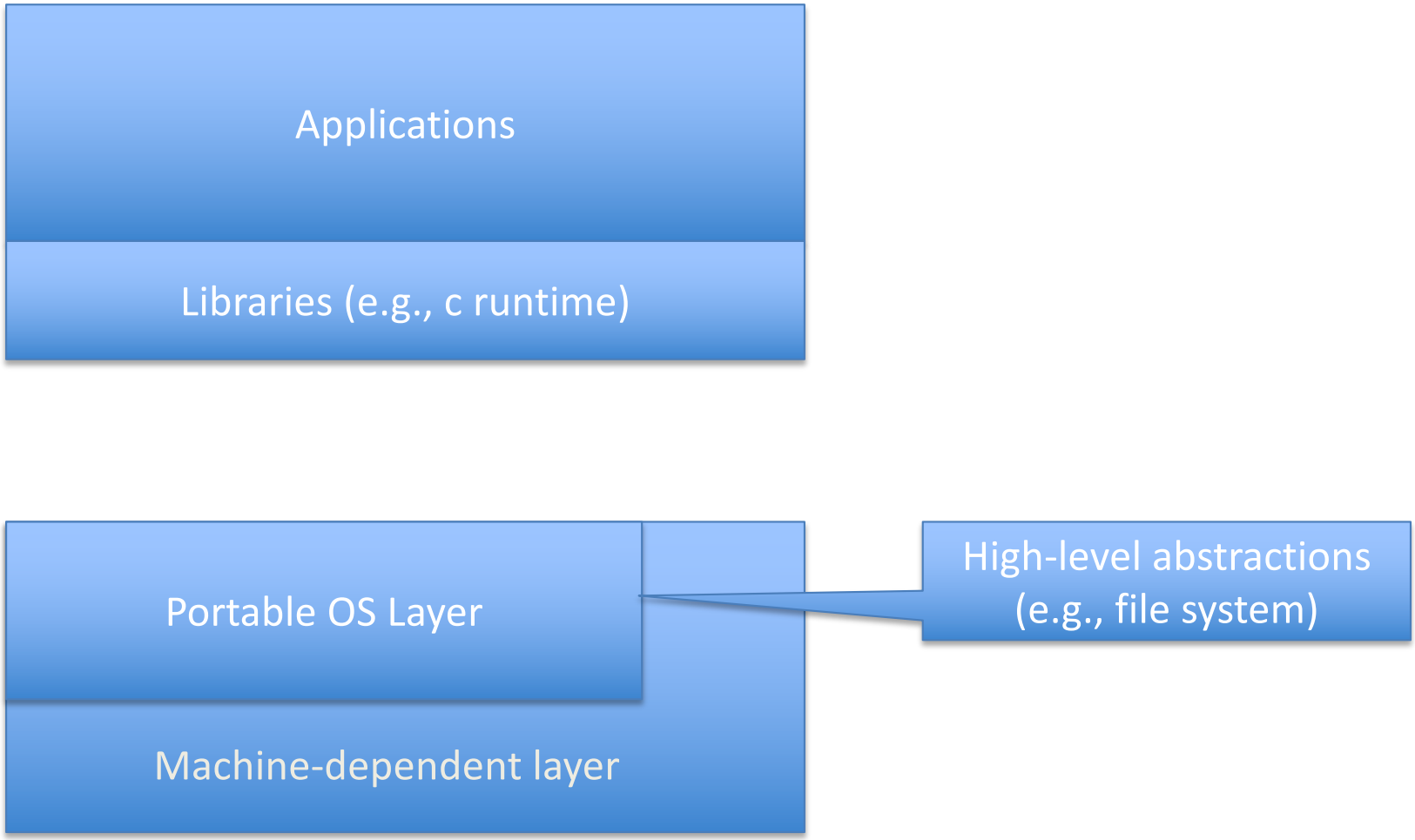
A peek into Unix/Linux



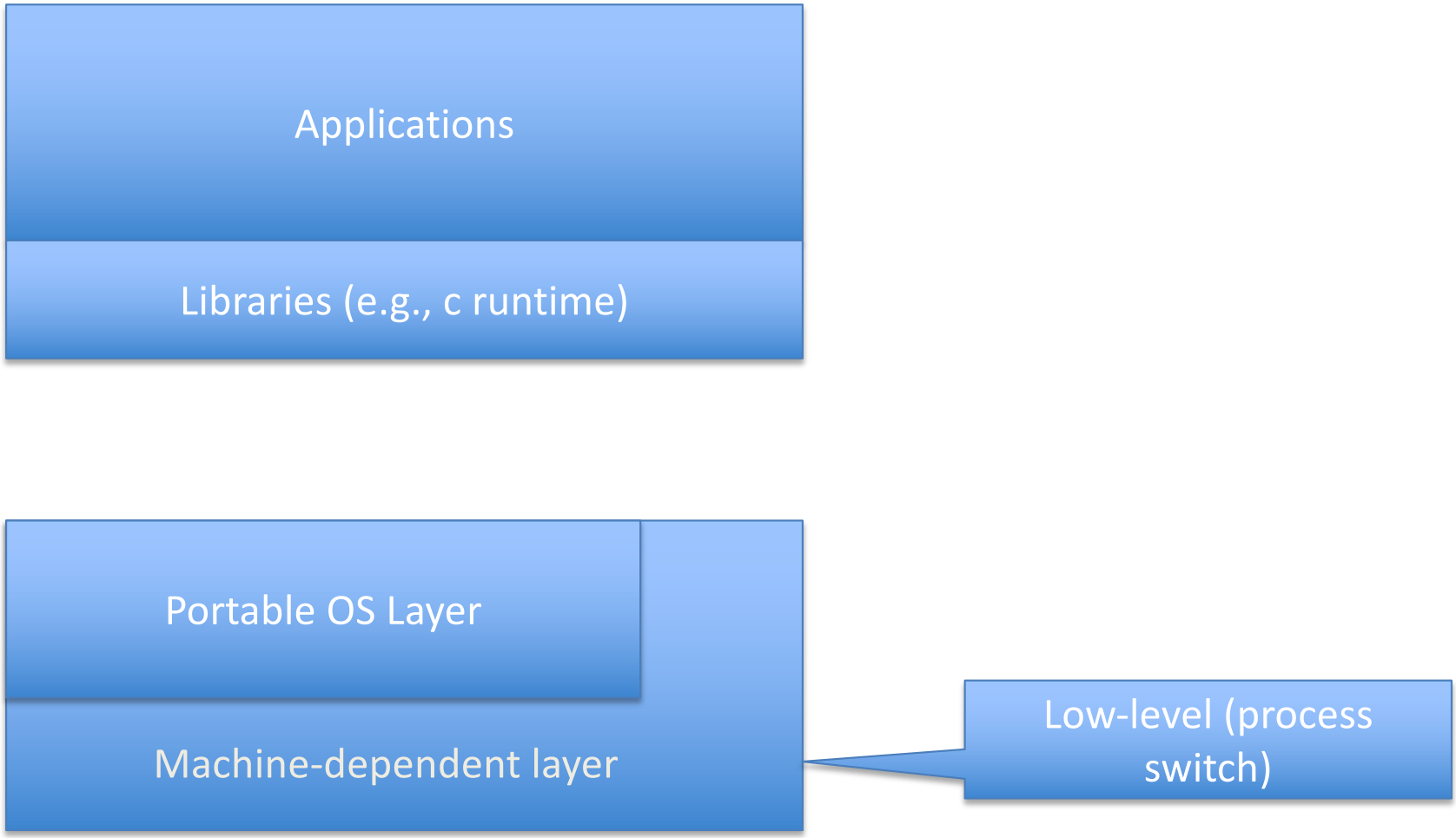
A peek into Unix/Linux



A peek into Unix/Linux



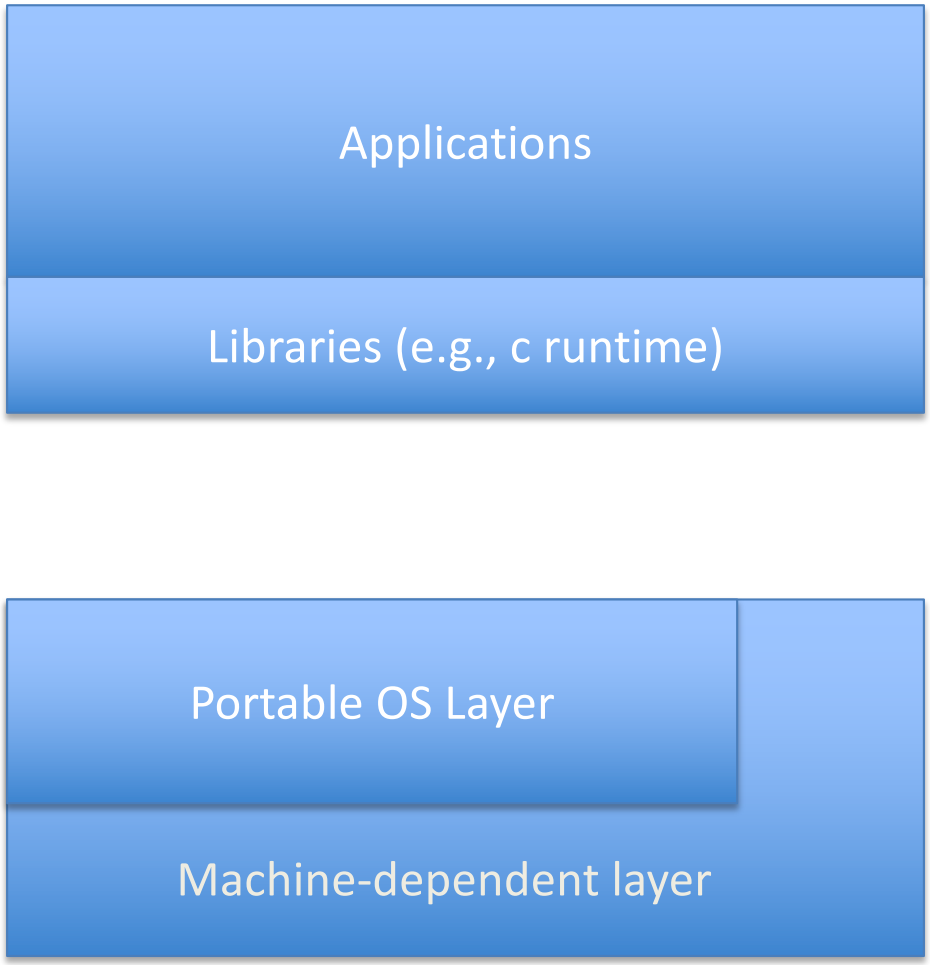
A peek into Unix/Linux



OS architectures

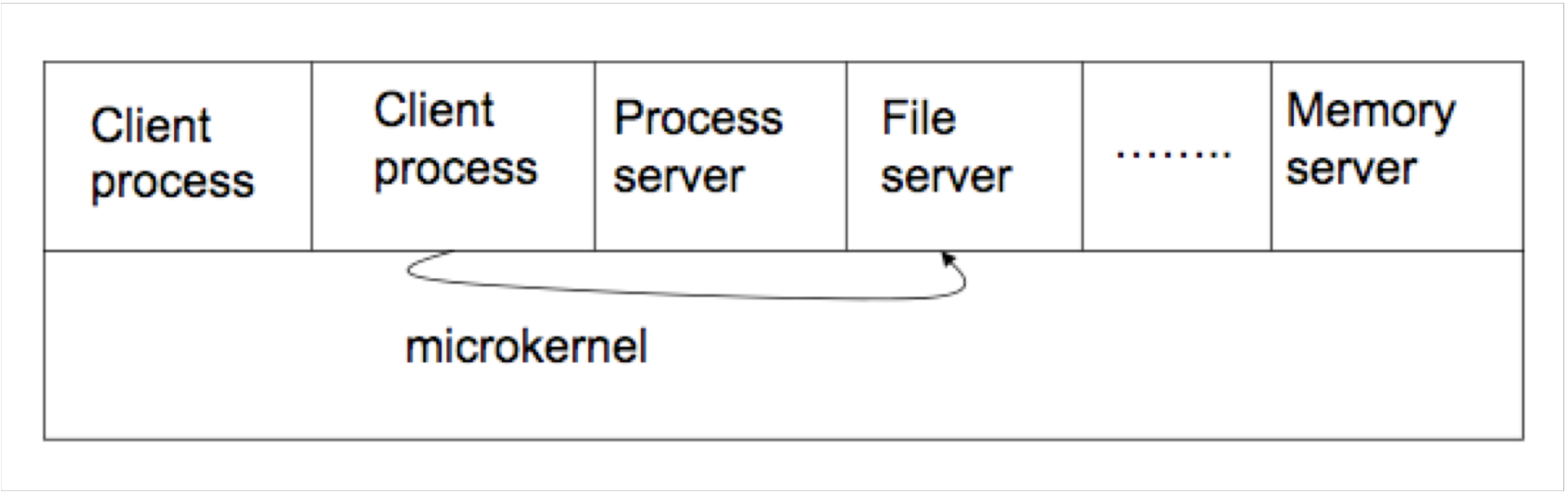
- OS developers paranoid
  - Buggy software
  - Unreliable hardware
  - Users cannot be trusted
- OS developers are engineers
  - Faster is better

Monolithic operating system



Alternative architectures: microkernel

- Protection, but how much?
- At cost of performance?
- Windows NT and Mac OS X have microkernel roots
- More common on embedded systems, might see resurgence in the years to come

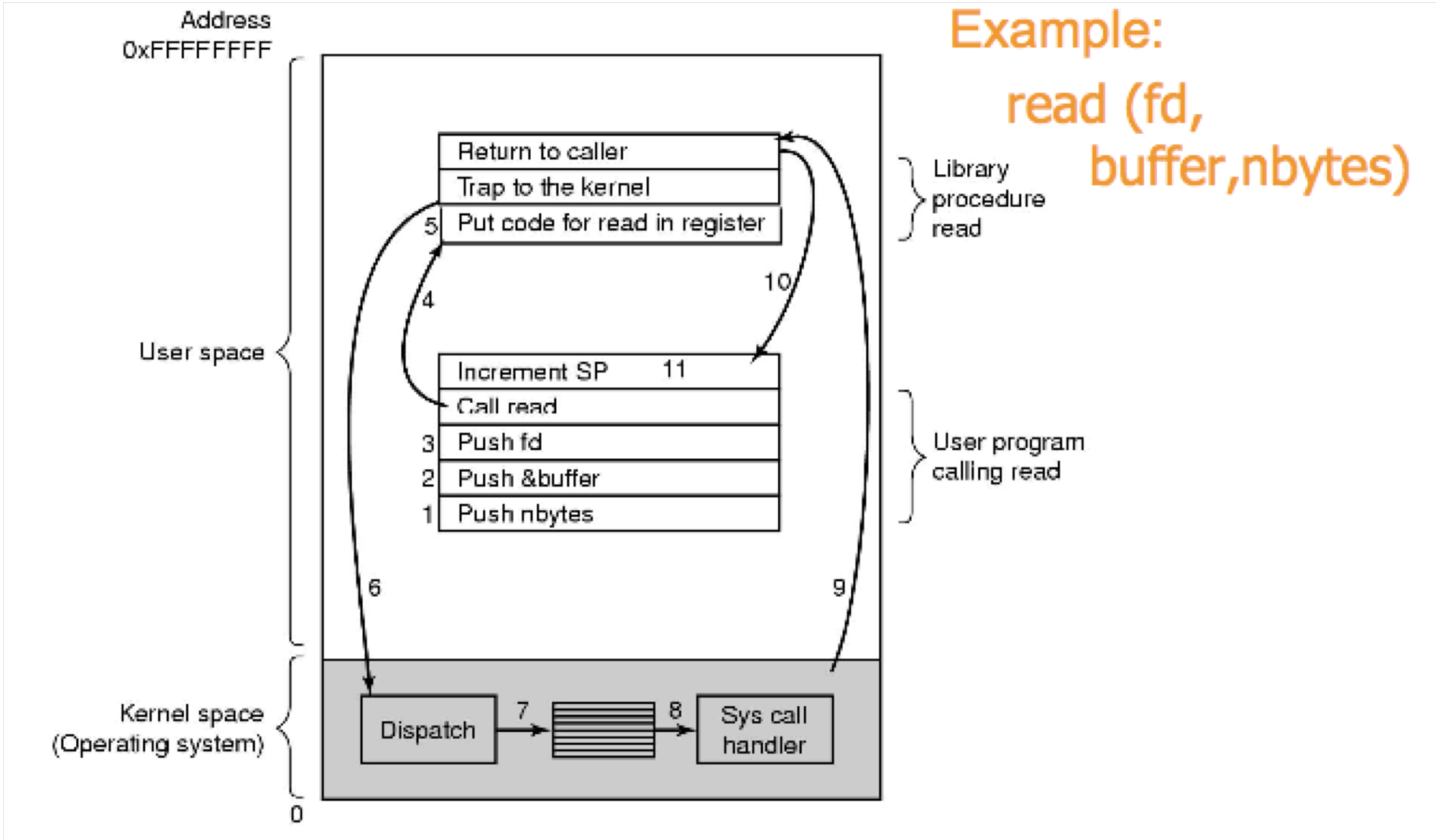


Alternative architecture: VMM

- VMM is like microkernel with uncreative interface
- But what about performance?
- Today's VMMs starting to resemble Microkernel

Abstractions and interfaces

- Key abstractions: process and file system
- Key interface: system call



Unix system calls

- Process management
  - Fork, exec, wait
- File handle
  - Open, read, write
- File namespace
  - Readdir, stat, unlink, link, rename

Implement a shell

- Goal: create child process and wait for it to return
- Fork() – return pid of child or 0 if you are the child
- Exec(argv) – replace current process with arguments argv
- Wait(pid) – wait for process pid to return

Implement a shell

```
while(1) {
    user_input = display_prompt();
    parse_input(user_input, &argv);









}
```

Process abstraction

- Illusion – infinite number of processes
- Reality – fixed process table sizes, finite memory and CPU

- Demo…

Fork bomb

- How can you fix this?

- CS comes from two different backgrounds: math and ee
  - Is it worth it to fix it?