CS 5460/6460
Operating Systems

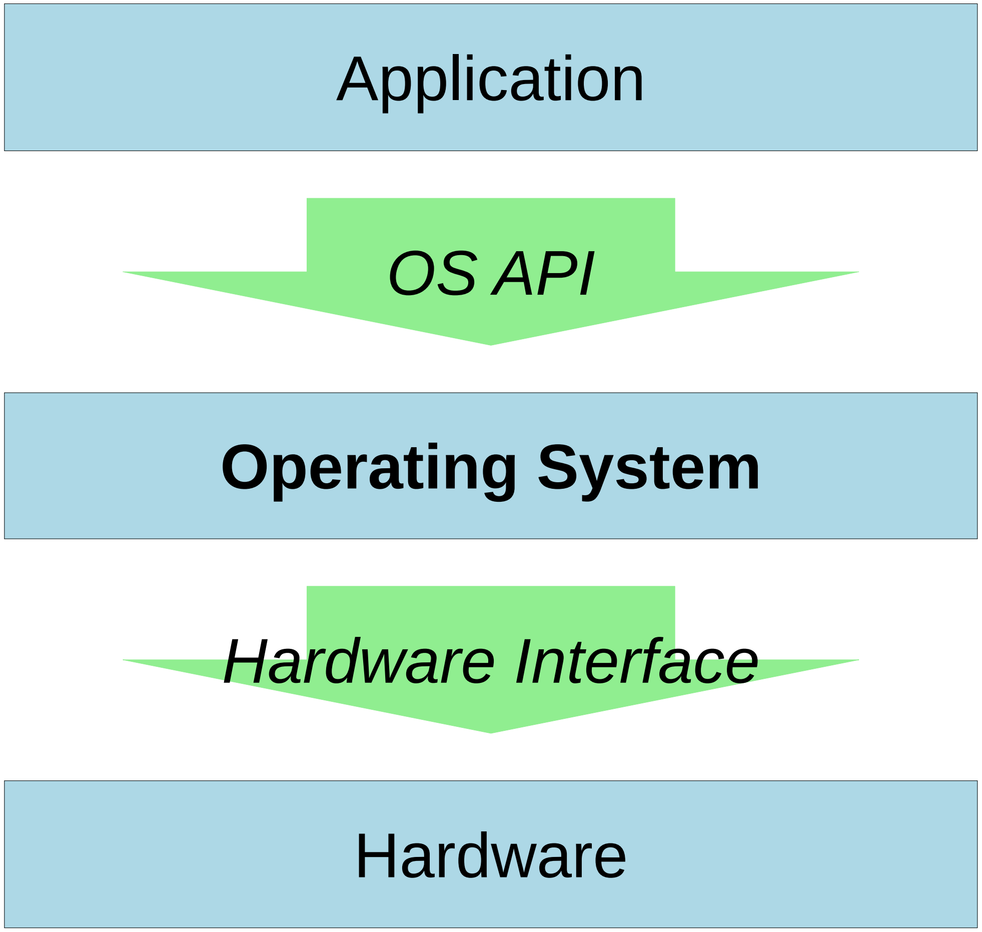
Fall 2009

Instructor: **Matthew Flatt**

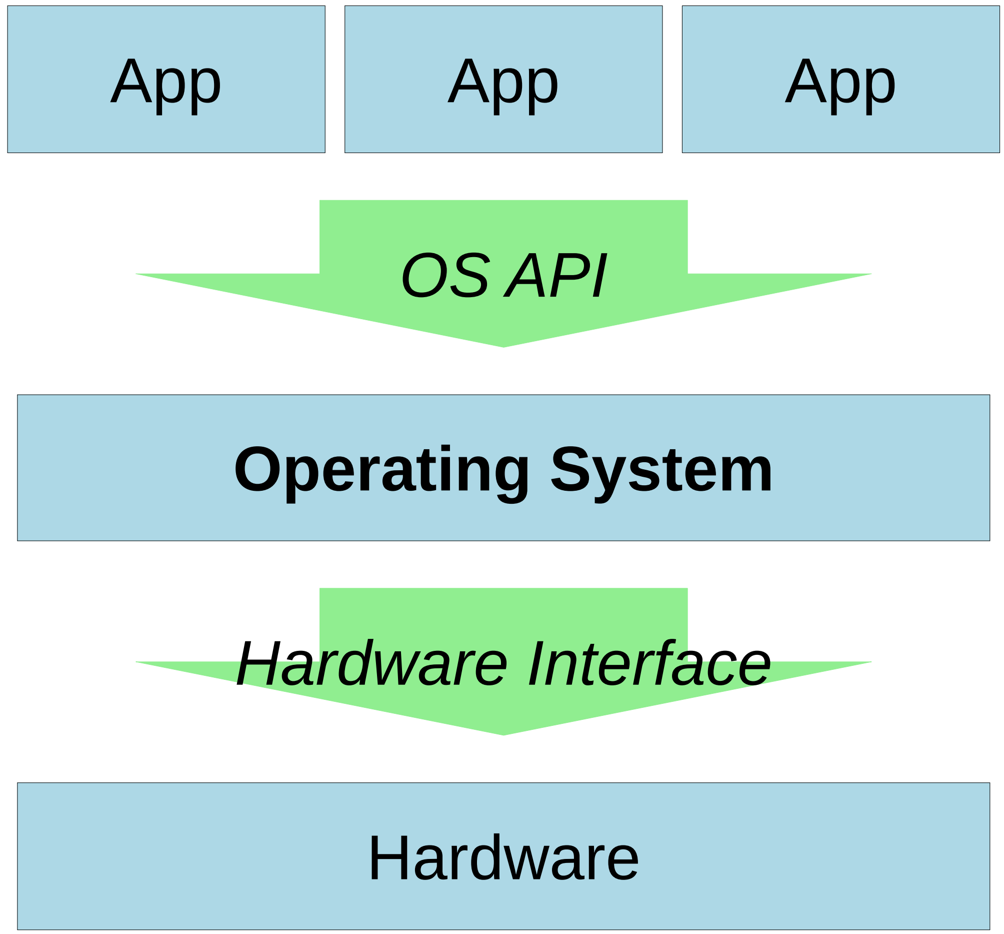
TAs: Bigyan Mukherjee, Amrish Kapoor

Part I – Course Overview

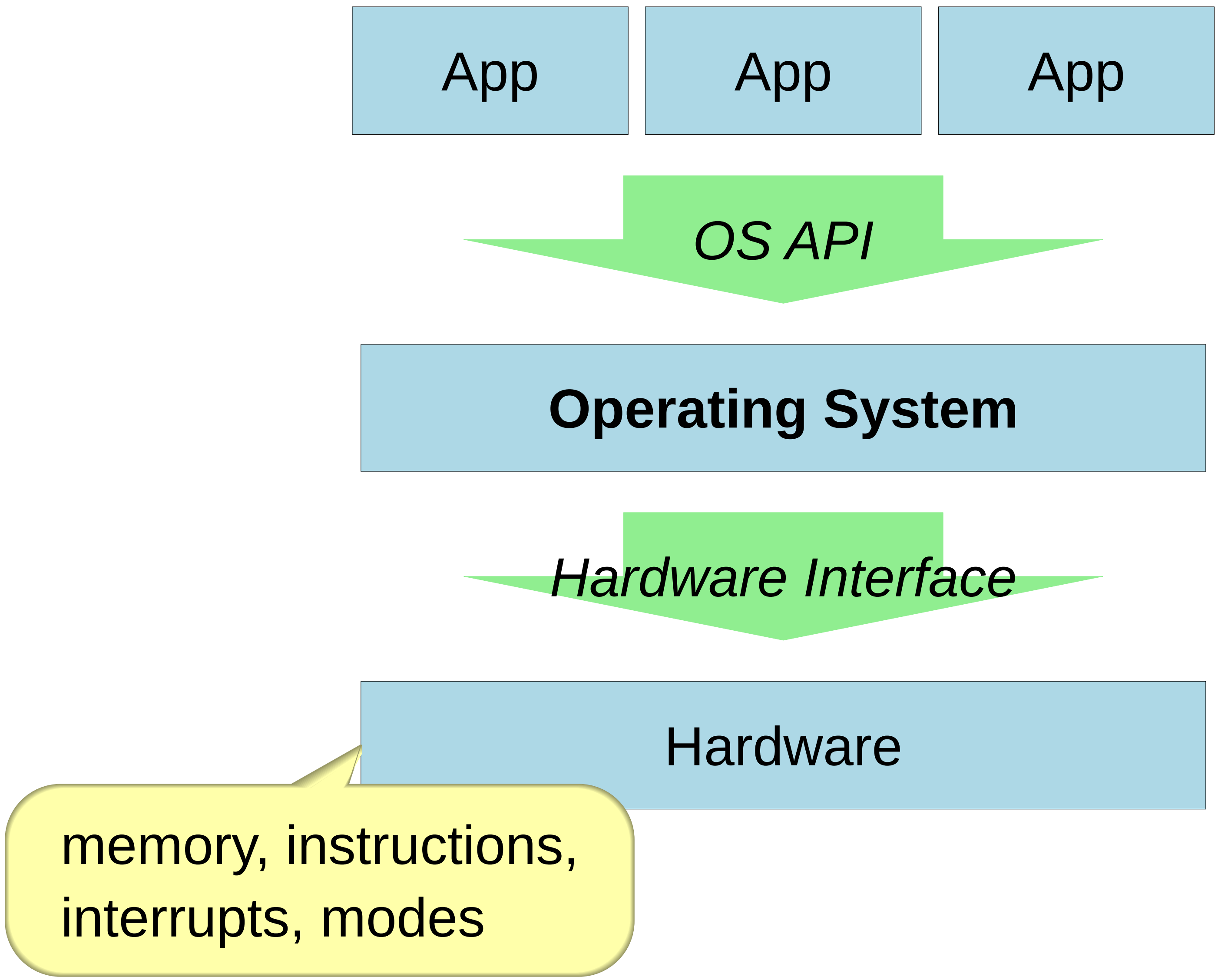
Operating Systems



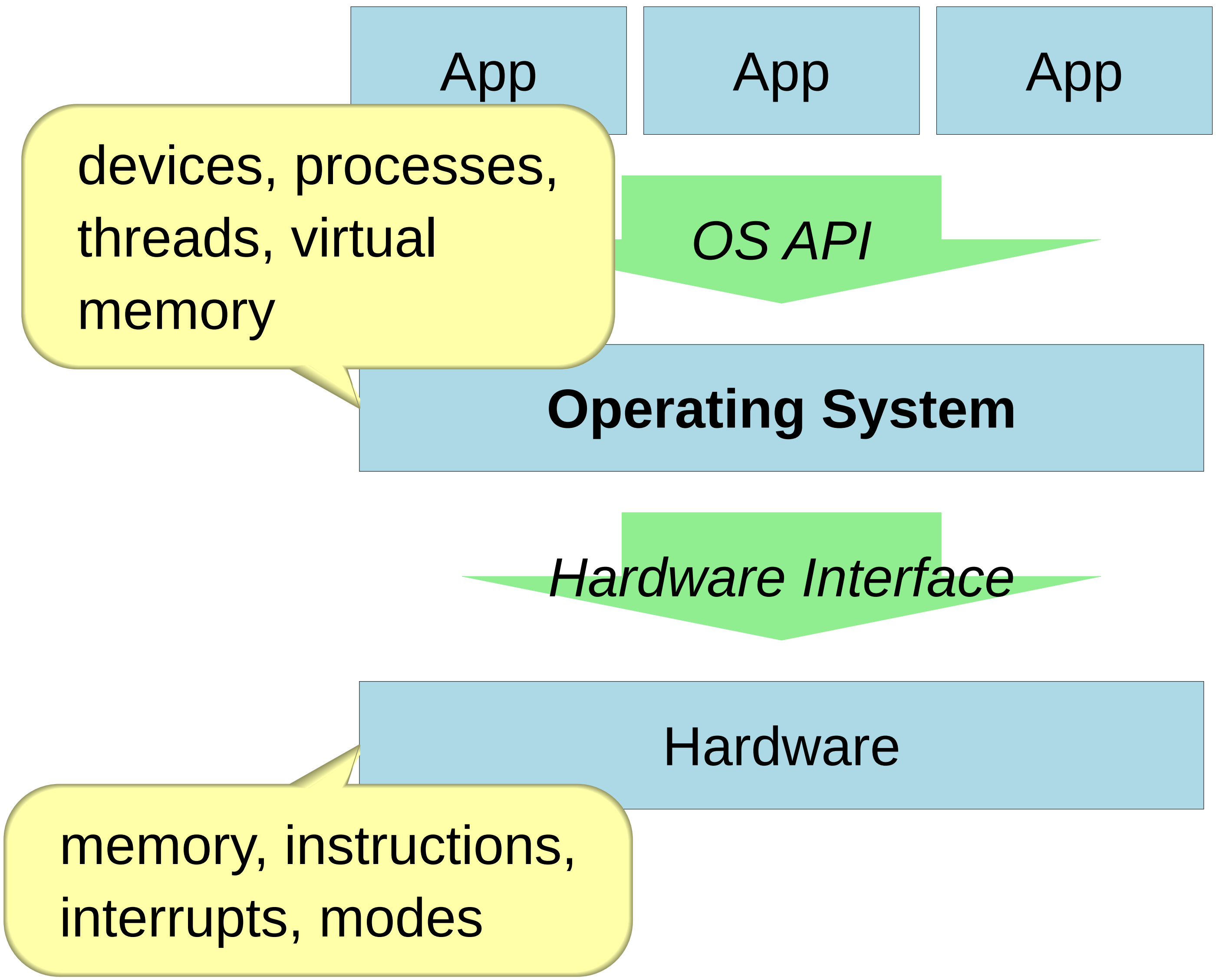
Operating Systems



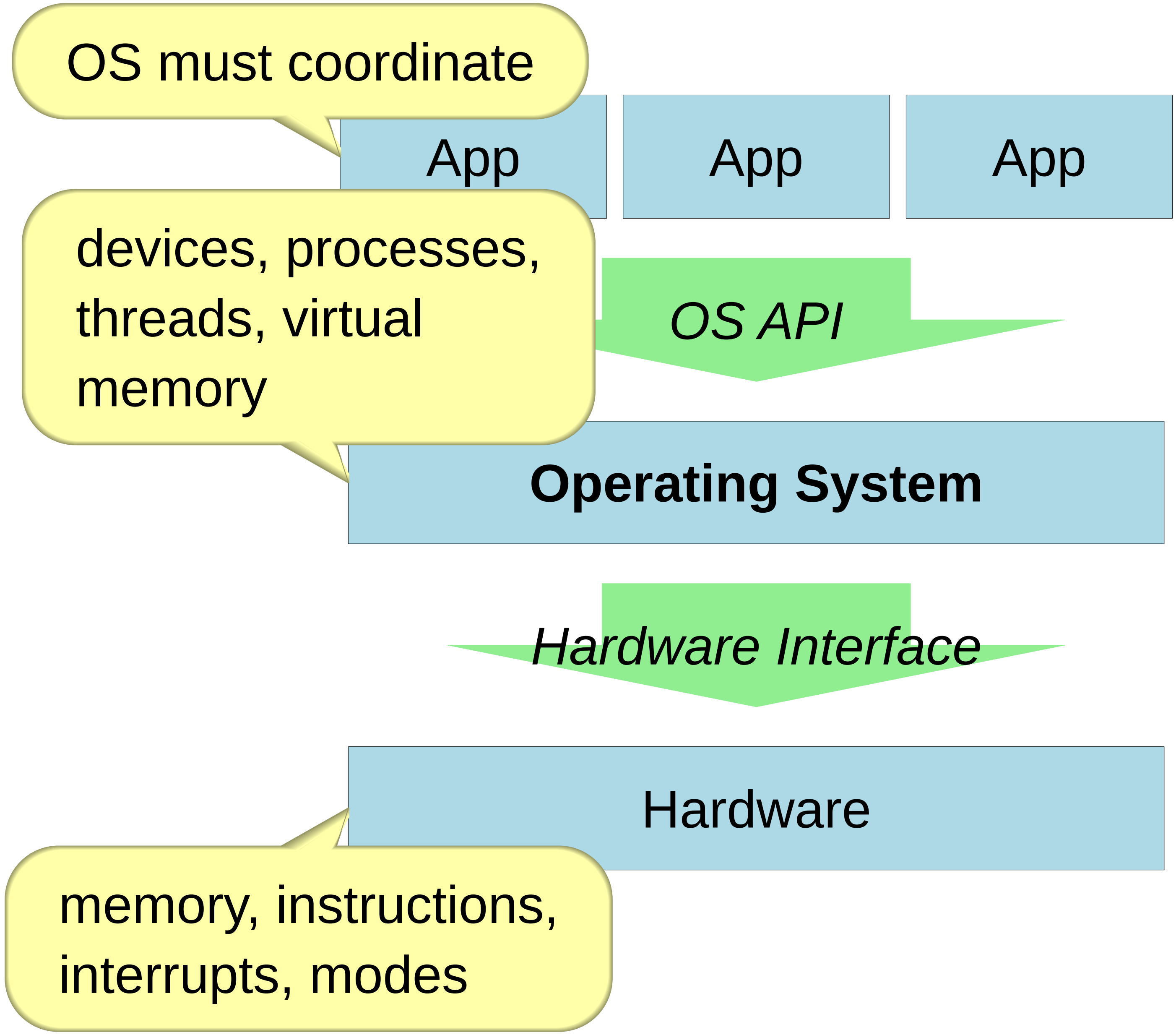
Operating Systems



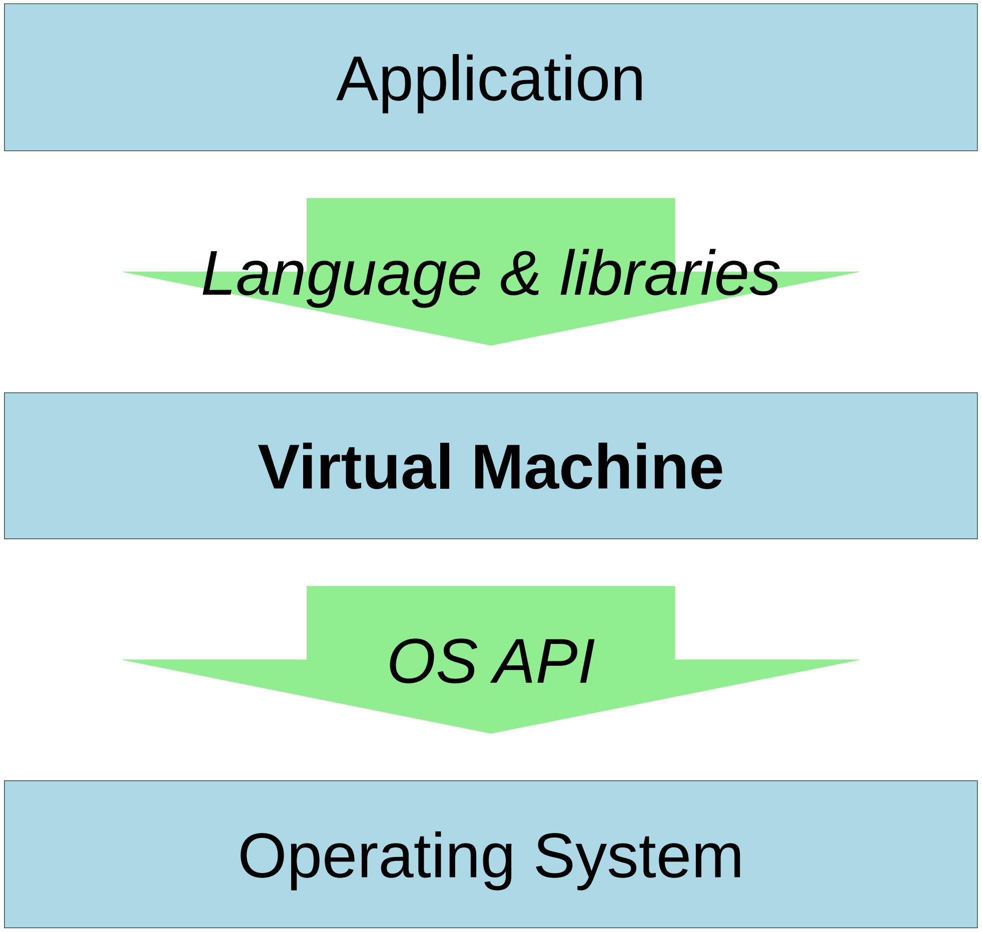
Operating Systems



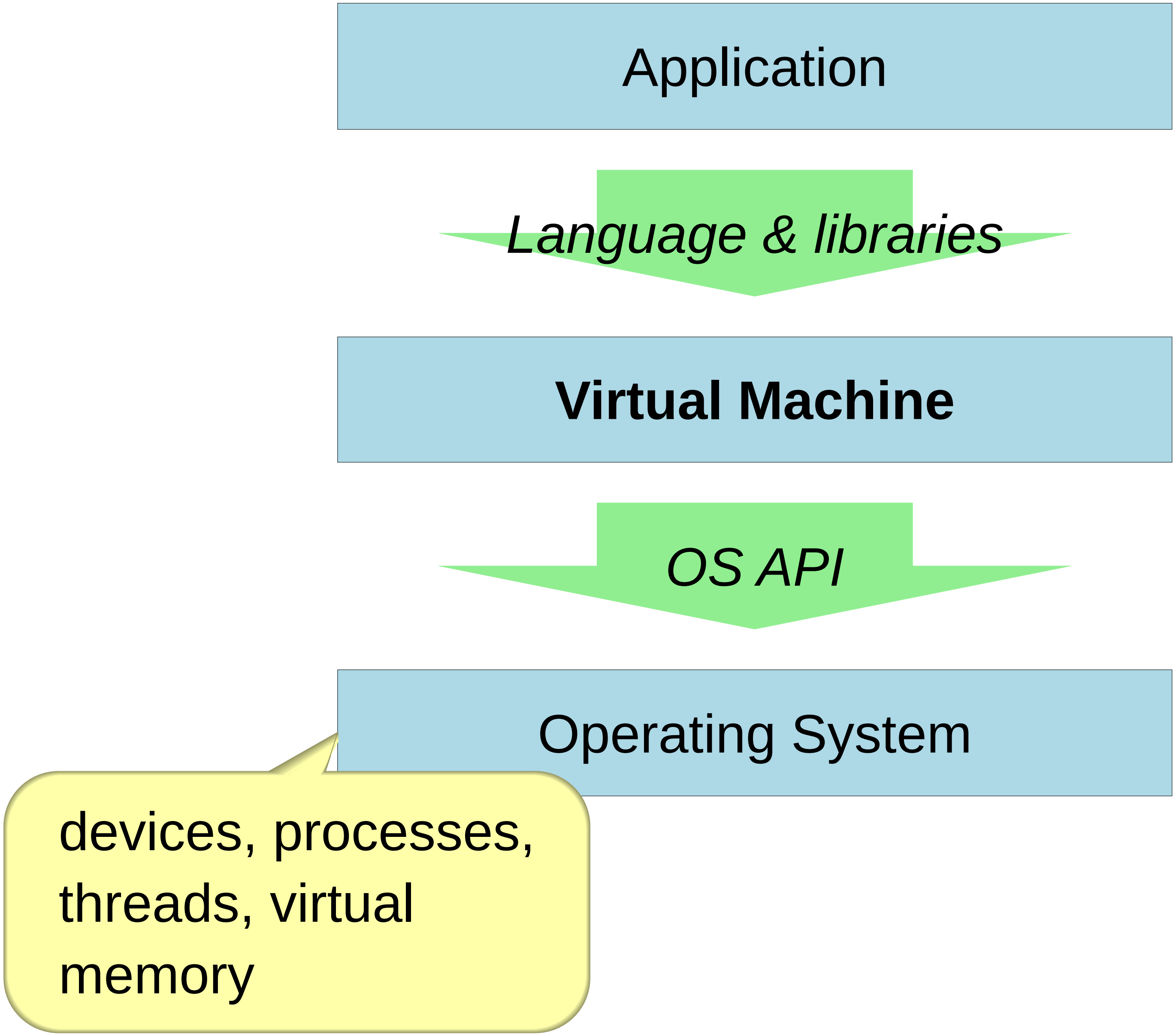
Operating Systems



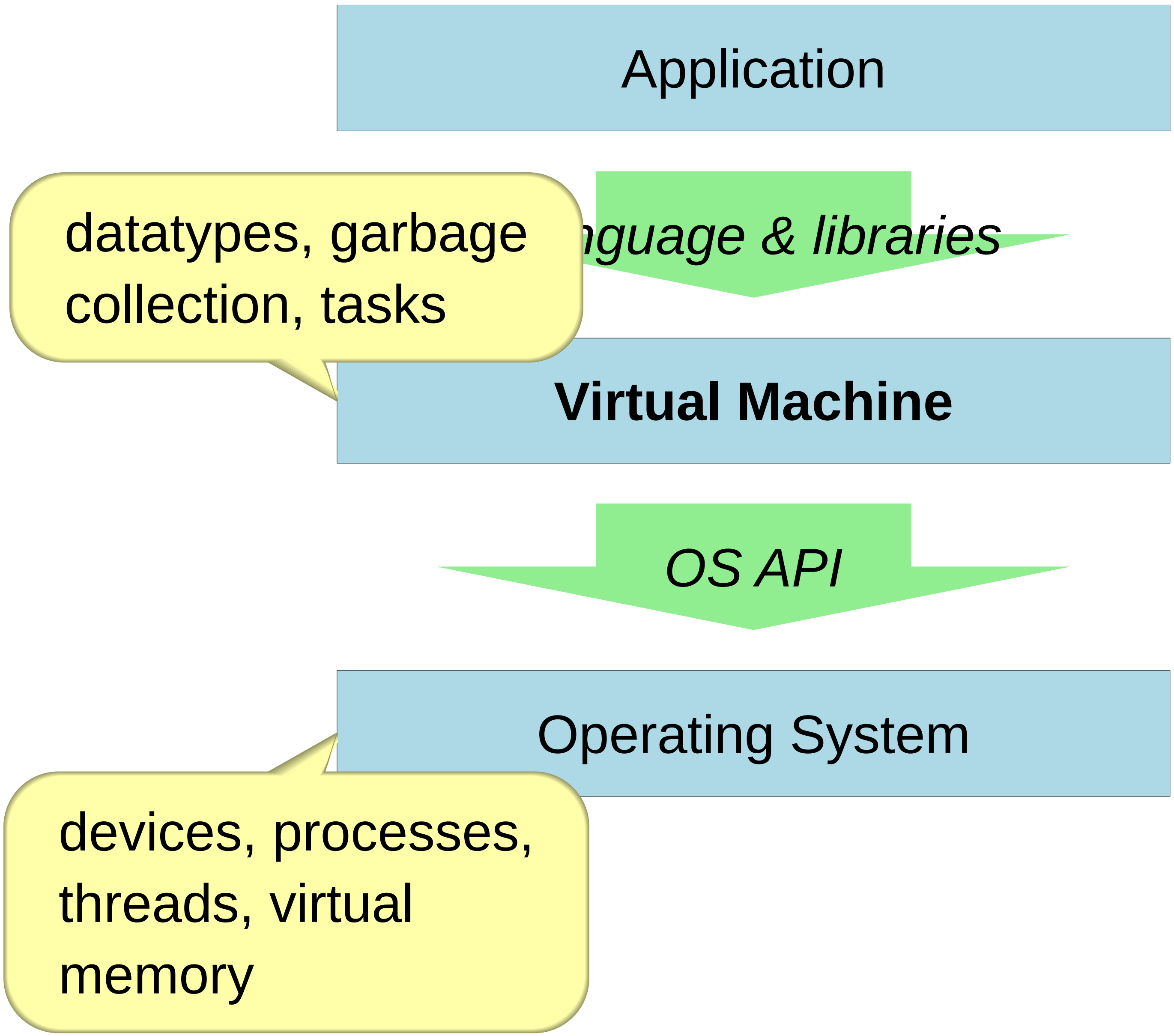
Virtual Machines



Virtual Machines



Virtual Machines

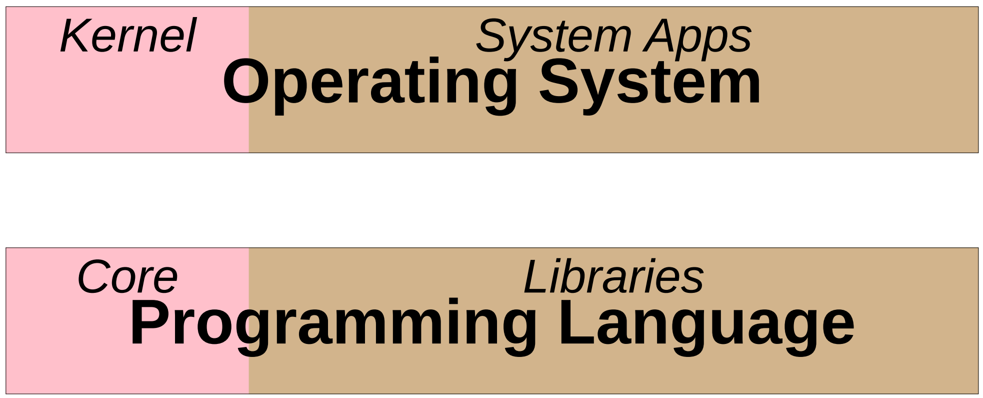


OS vs. Kernel

Operating System

Programming Language

OS vs. Kernel


Kernel
System Apps
Operating System
Core
Libraries
Programming Language

OS Design

- Convenience
- Performance

Why Study OS?

- Abstractions
  - Many tried and true constructs
- Know your tools
  - Driving a shell
  - Composing and controlling processes
- Know your environment
  - Space of possibilities for applications
  - Performance implications
- Build your own OS/VM?

Prerequisites

- CS 4400 *really is* a prerequisite
- C programming
- General familiarity with Unix-style OSes

Course Details

`http://www.eng.utah.edu/~cs5460/`

Expect about 5 C programming tasks as homework

Example Concepts

Services	Processes
Threads	CPU Scheduling
I/O Redirection	Pipes
Concurrency	Synchronization
Deadlock	Memory Management
Paging	Segmentation
Virtual Memory	Page Replacement
File Systems	I/O Systems
Distributed Systems	Networks
RPC	Distributed Filesystems
Security	Embedded Systems

Part II – OS History

OS History

Phase 1: No or Minimal OS

Hardware: *expensive*

Humans: *cheap*

- One user at a time on the console
- One function at a time (no overlap of computation and I/O)
- User must be on the console to debug

OS History

Phase 2: Batch Processing

Hardware: *expensive* Humans: *cheap*

- Users give their program (on cards or tape) to a human, who then schedules the jobs (e.g., Fortran and Pascal programs)
- OS loads, runs, and dumps user jobs
- Batch processing makes better use of the hardware, but debugging is much more difficult

OS History

Phase 3: Overlap of I/O and Computation

Hardware: *expensive*

Humans: *cheap*

- Buffering and interrupt handling in OS
- Spool jobs on drum
- No protection $\Rightarrow$ One job at a time
- Performance improves, because I/O and processing happen concurrently

OS History

Phase 4: Memory Protection and Relocation

Hardware: *expensive*

Humans: *cheap*

- Multiprogramming — several programs run at the same time
- One job runs until it performs I/O, then another job gets the CPU
- OS decides which spooled jobs to start, protects one program's memory from other programs, decides which process to resume when one gives up the CPU

First OS failures:

- Multics announced in 1963, released in 1969
- OS/360 released with 1000 known bugs

OS History

Phase 5: Interactive Timesharing

Hardware: *cheap*

Humans: *expensive*

- Terminals are cheap
  - All users interact with the system at once, debugging becomes a lot easier, process switching occurs much more frequently
- Memory is cheap — programs and data go on-line
- UNIX simplifies Multics so it can be built
- New OS services: shell, filesystems, rapid process switching, virtual memory

New problems: response time & thrashing

OS History

Phase 6: Personal Computing

Hardware: *very cheap*

Humans: *expensive*

- Computers are cheap, so put one in each terminal
- Make the OS simple (again) by getting rid of support for multiprogramming, concurrency, and protection...
  - Did not really work; e.g., Microsoft had to put all this functionality back into its OS
  - With distributed computing & networking, we still want to share resources, but now we want to share across machines

OS History

Phase 7a: Parallelism

Hardware: *very cheap* Humans: *expensive*

- Increased processing demands lead to parallelism
- In parallel systems, multiple processors are in the same machine, sharing memory, I/O devices, clock, ...
- In distributed systems, multiple processor communicate via network

Advantages: increased performance, increased reliability, sharing of specialized resources

OS History

Phase 7b: Real-Time Systems

Hardware: *very cheap*

Humans: *expensive*

- Computers control physical machines or provide high-quality interaction
  - e.g., virtual reality
- Timing requirements provide deadlines by when tasks must be accomplished
- Hard vs. soft real-time:
  - Hard real-time OS must meet timing requirements (so omit features)
  - Soft real-time OS allows deadlines to be missed

OS History

Phase 7c: Embedded Systems

Hardware: *very, very cheap*   Humans: *expensive*

- Your car may have dozens of processors
- Severe processing constraints $\Rightarrow$ no OS...
- Programmer creates own OS abstractions

History Conclusions

- Different environments demand different designs
- Many abstractions nevertheless work across many environmets