16.070

Introduction to Computers & Programming

Operating systems

Prof. Kristina Lundqvist
Dept. of Aero/Astro, MIT

- This chapter introduces the fundamental concepts associated with operating systems and networks. It begins with a historical look at operating systems, followed by discussions of operating system architecture and internal operation. An optional section covers semaphores and deadlock. Following this is an introduction to networks in general and the Internet in particular. There is an optional section that introduces the four level software hierarchy on which the Internet communication is based. the chapter closes with a discussion of network security issues.

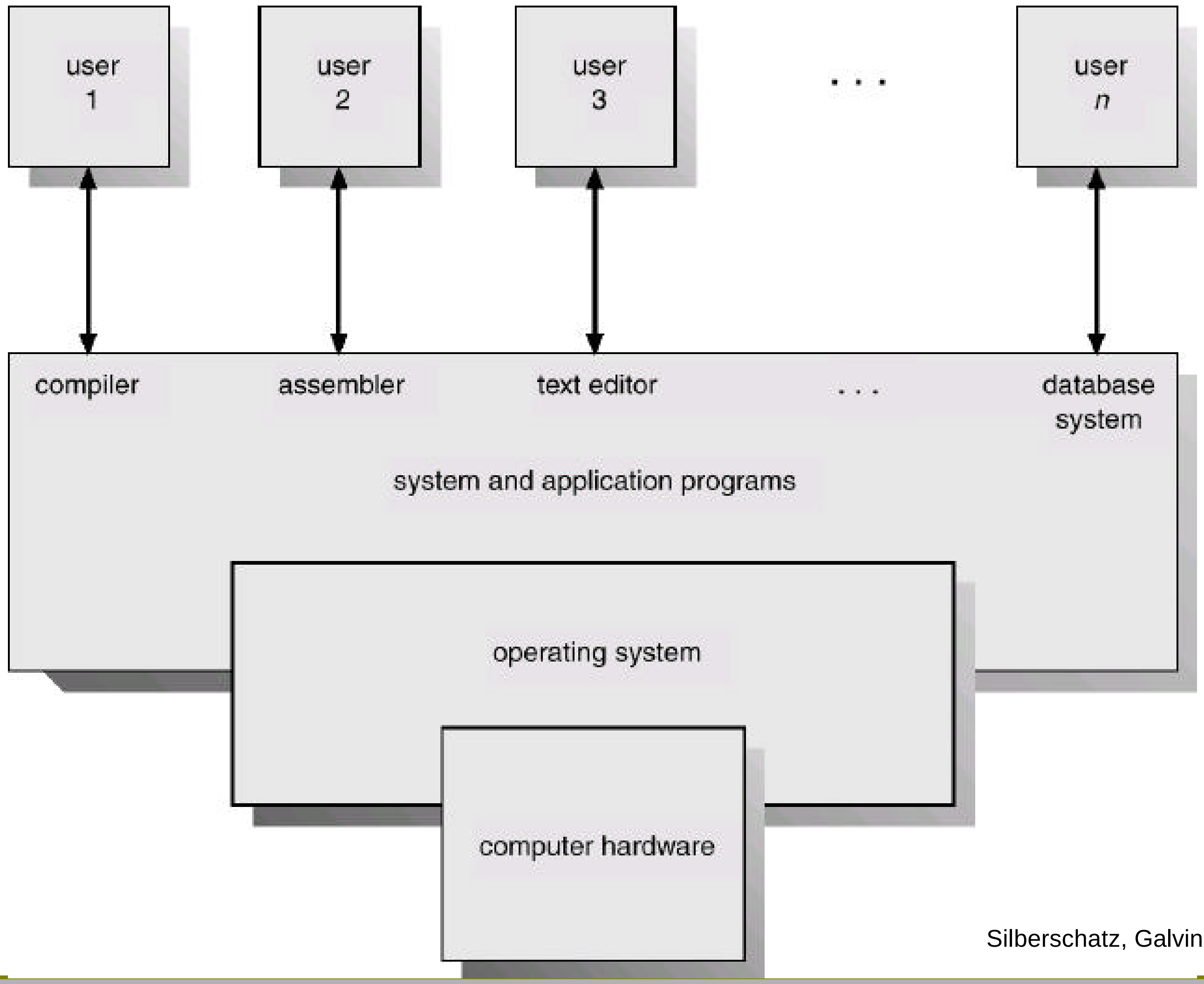
OS Fundamentals

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- Operating system goals:
  - Execute user programs and make solving user problems easier.
  - Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

Computer System

- Hardware – provides basic computing resources (CPU, memory, I/O devices).
- Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.
- Applications programs – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
- Users (people, machines, other computers).

Systemic View



Silberschatz, Galvin, and Gagne ©1999

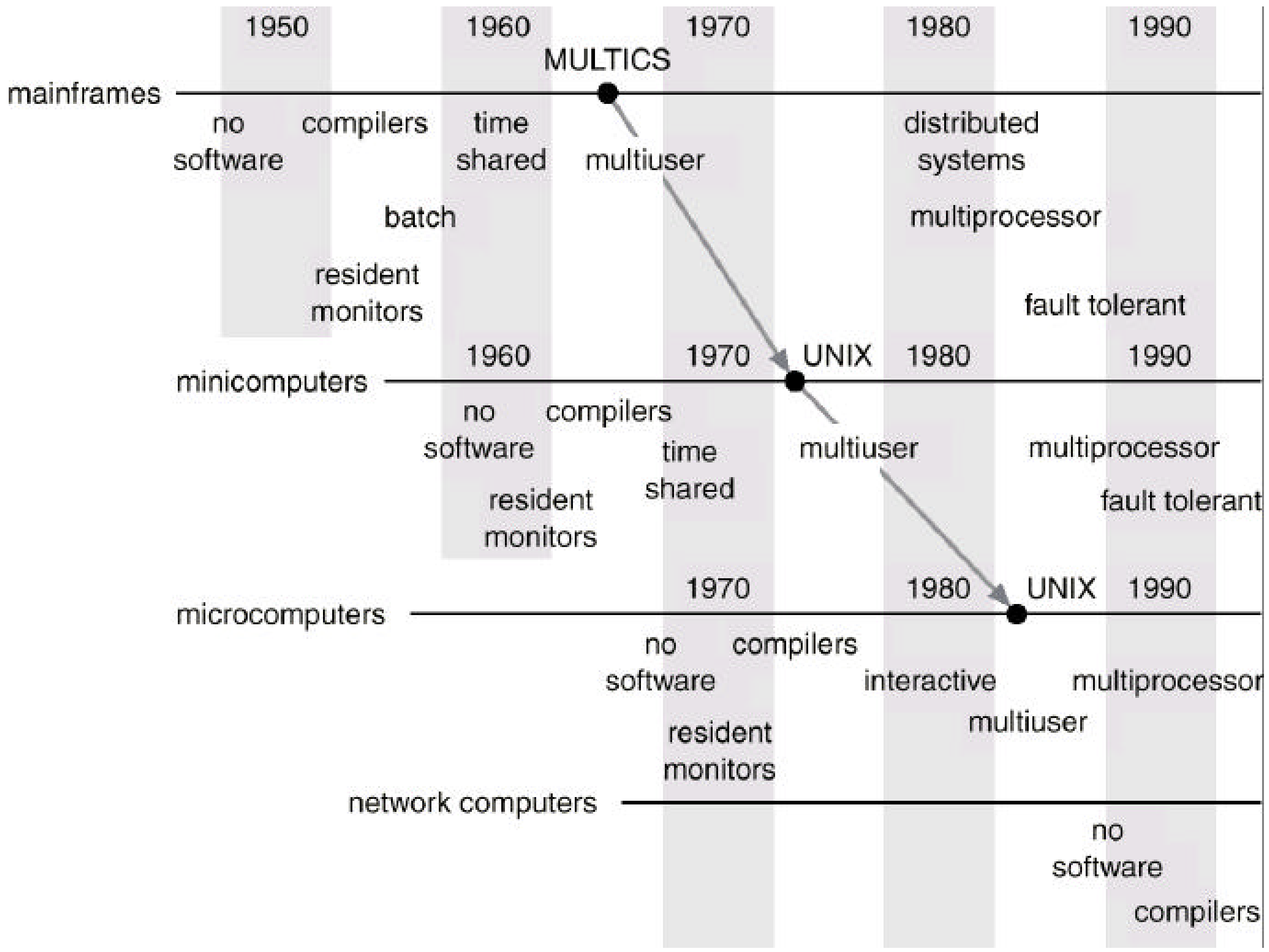
Types of OS

- Real-time operating system (RTOS)
- Single-user, single task
- Single-user, multi task
- Multi-user

Key Components of an OS

- Processor management
- Memory management
- Device management
- Storage management
- Application interface
- User interface

OS Evolution

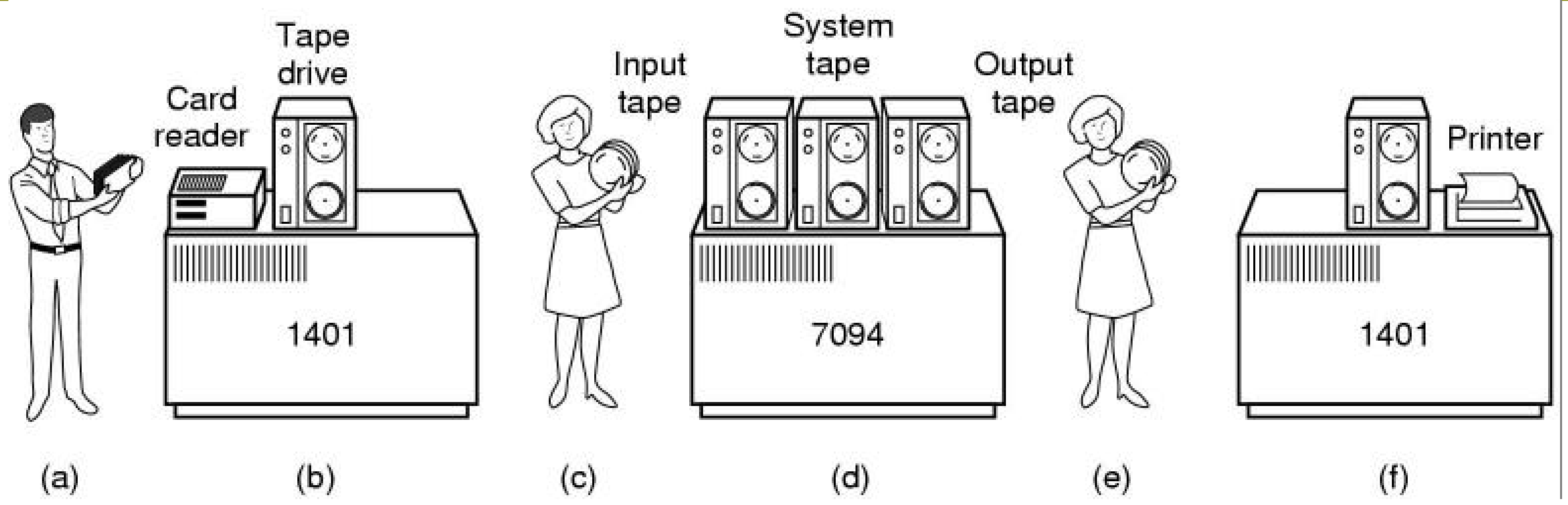


Silberschatz, Galvin, and Gagne ©1999

First Generation

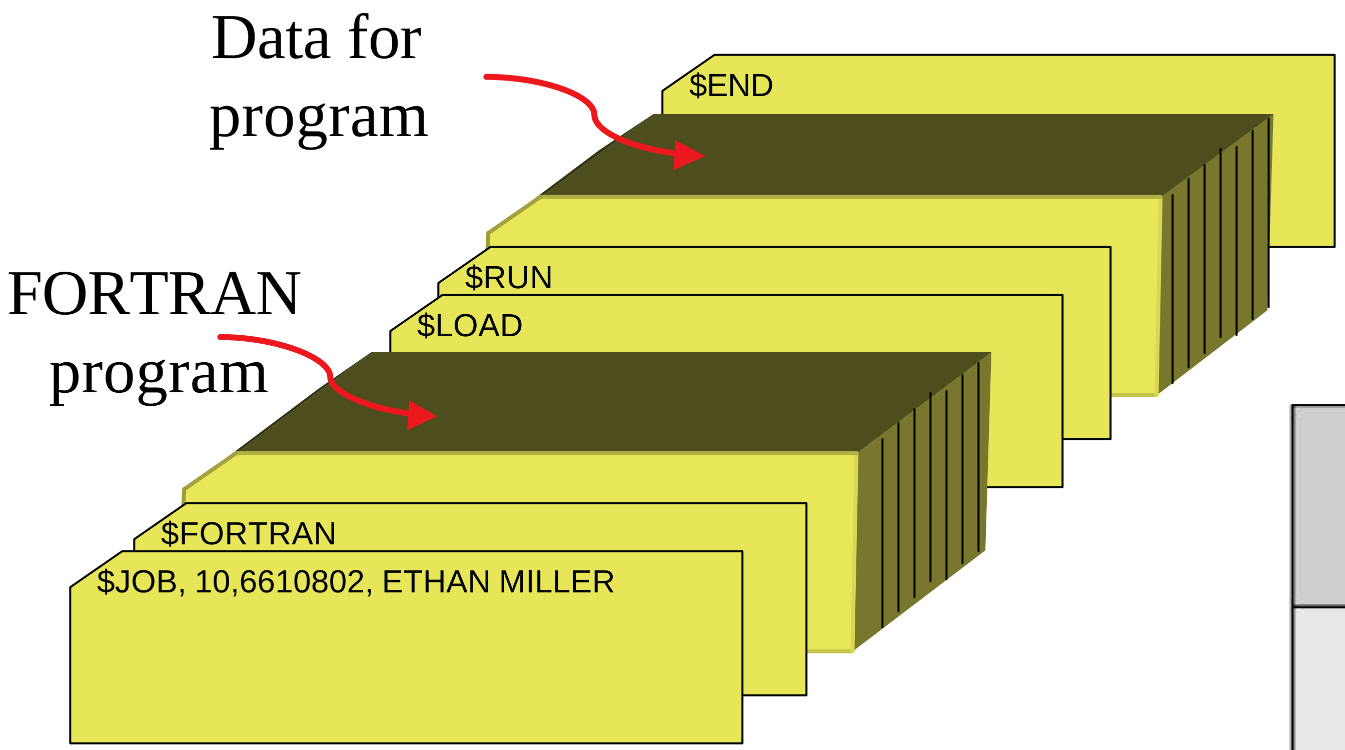
- Run one job at a time
  - Enter it into the computer (might require rewiring!)
  - Run it
  - Record the results
- Problem: lots of wasted computer time!
  - Computer was idle during first and last steps
  - Computers were ***very*** expensive!
- Goal: make better use of an expensive commodity: computer time

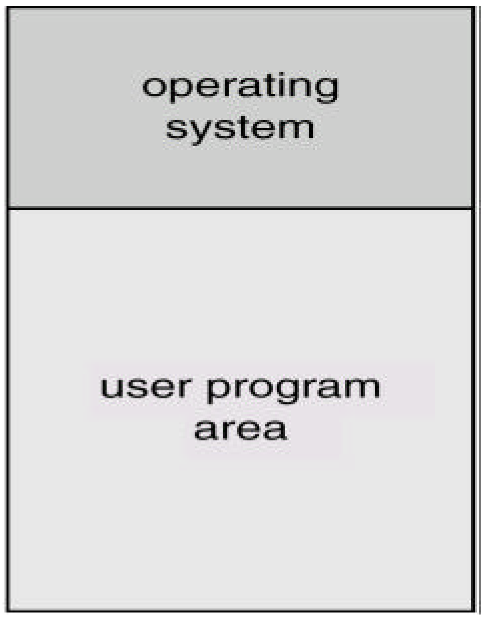
Batch Systems



- Bring cards to 1401
- Read cards onto input tape
- Put input tape on 7094
- Perform the computation, writing results to output tape
- Put output tape on 1401, which prints output

Structure of a Second Generation Job



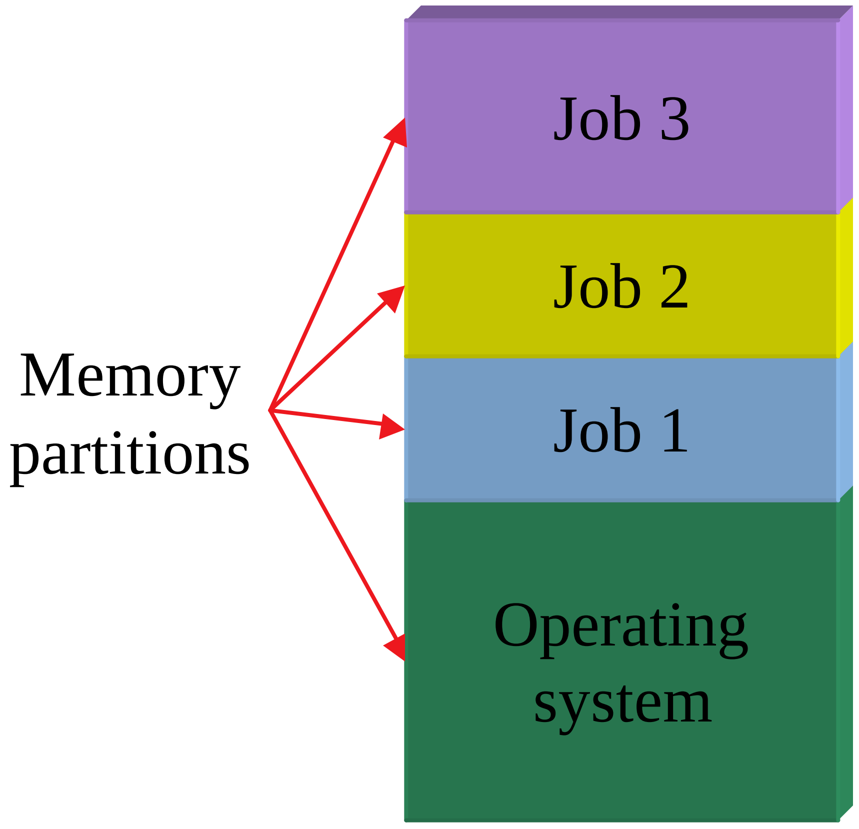


Silberschatz, Galvin, and Gagne ©1999

Spooler

- Original batch systems used tape drives
- Later batch systems used disks for buffering
  - Operator read cards onto disk attached to the computer
  - Computer read jobs from disk
  - Computer wrote job results to disk
  - Operator directed that job results be printed from disk
- Disks enabled simultaneous peripheral operation on-line (spooling)
  - Computer overlapped I/O of one job with execution of another
  - Better utilization of the expensive CPU
  - Still only one job active at any given time

Third Generation: Multiprogramming



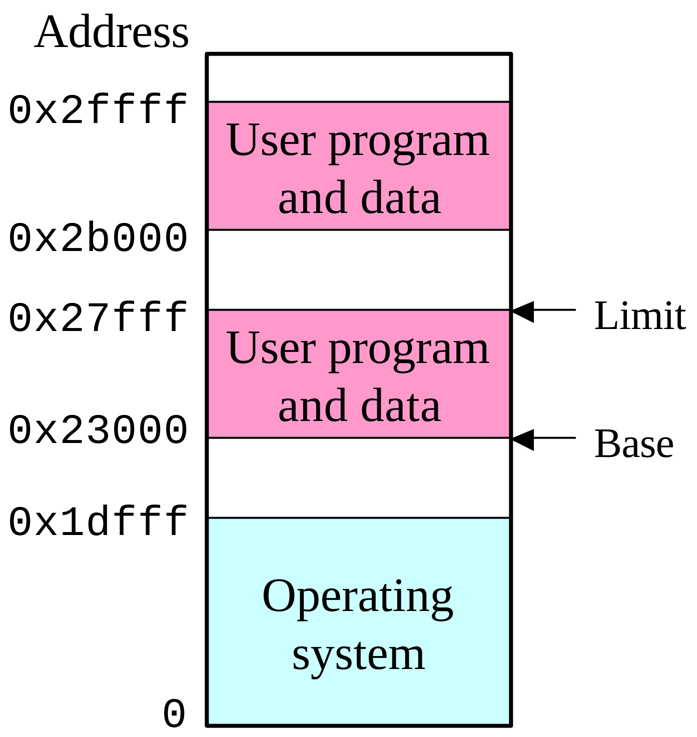
- Multiple jobs in memory
  - Protected from one another
- Operating system protected from each job as well
- Resources (time, hardware) split between jobs
- Still not interactive
  - User submits job
  - Computer runs it
  - User gets results minutes (hours, days) later

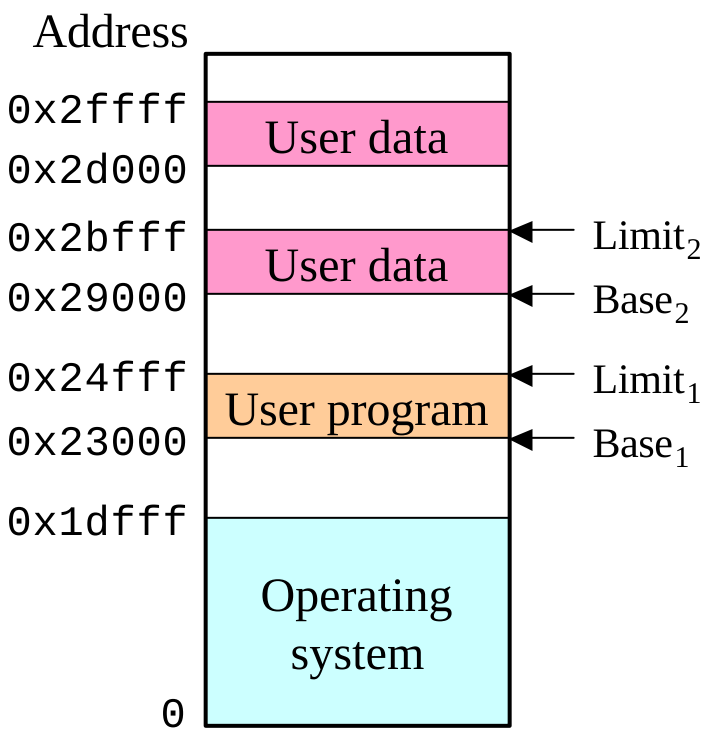
Silberschatz, Galvin, and Gagne ©1999

Time Sharing

- Multiprogramming allowed several jobs to be active at one time
  - Initially used for batch systems
  - Cheaper hardware terminals -> interactive use
- Computer use got much cheaper and easier
  - No more “priesthood”
  - Quicker turnaround meant quick fixes for problems

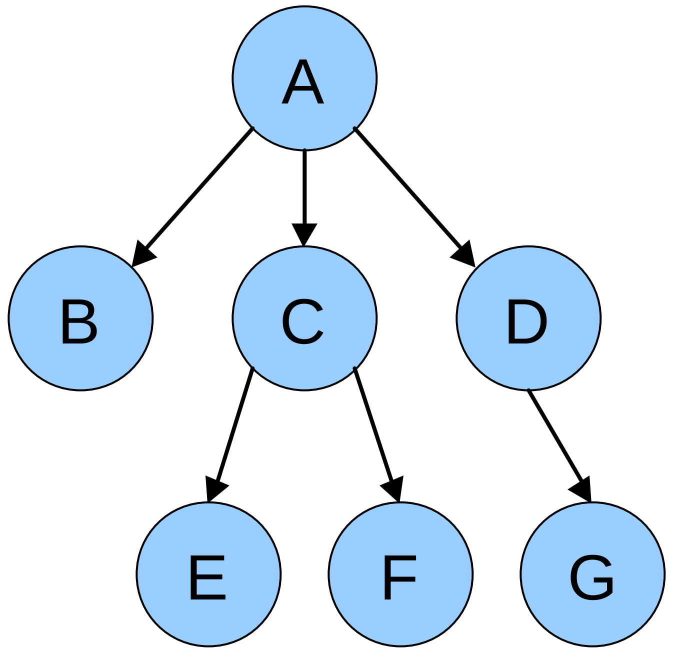
Memory





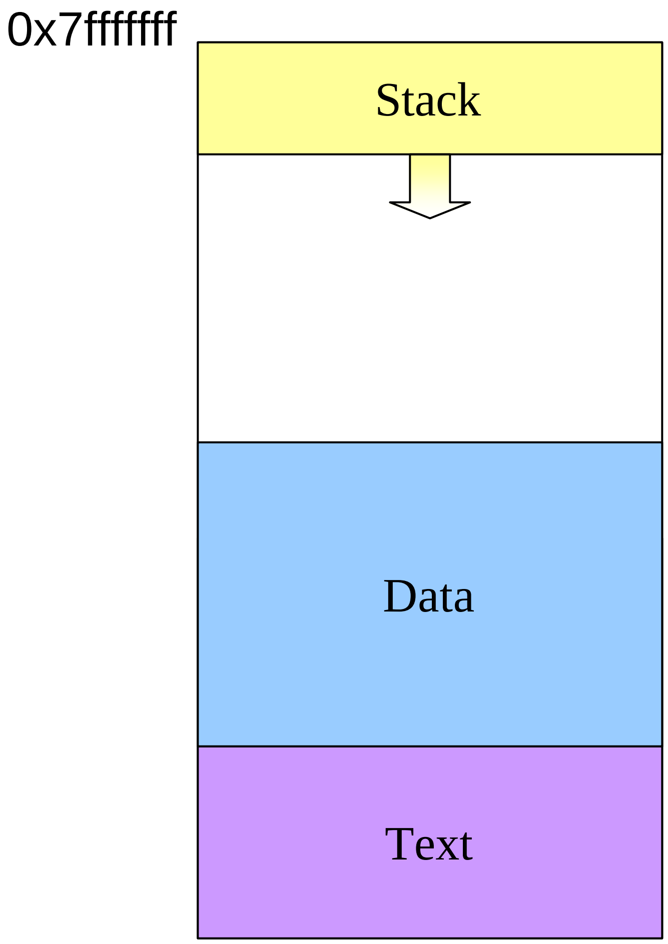
- Single base/limit pair: set for each process
- Two base/limit registers: one for program, one for data

Processes



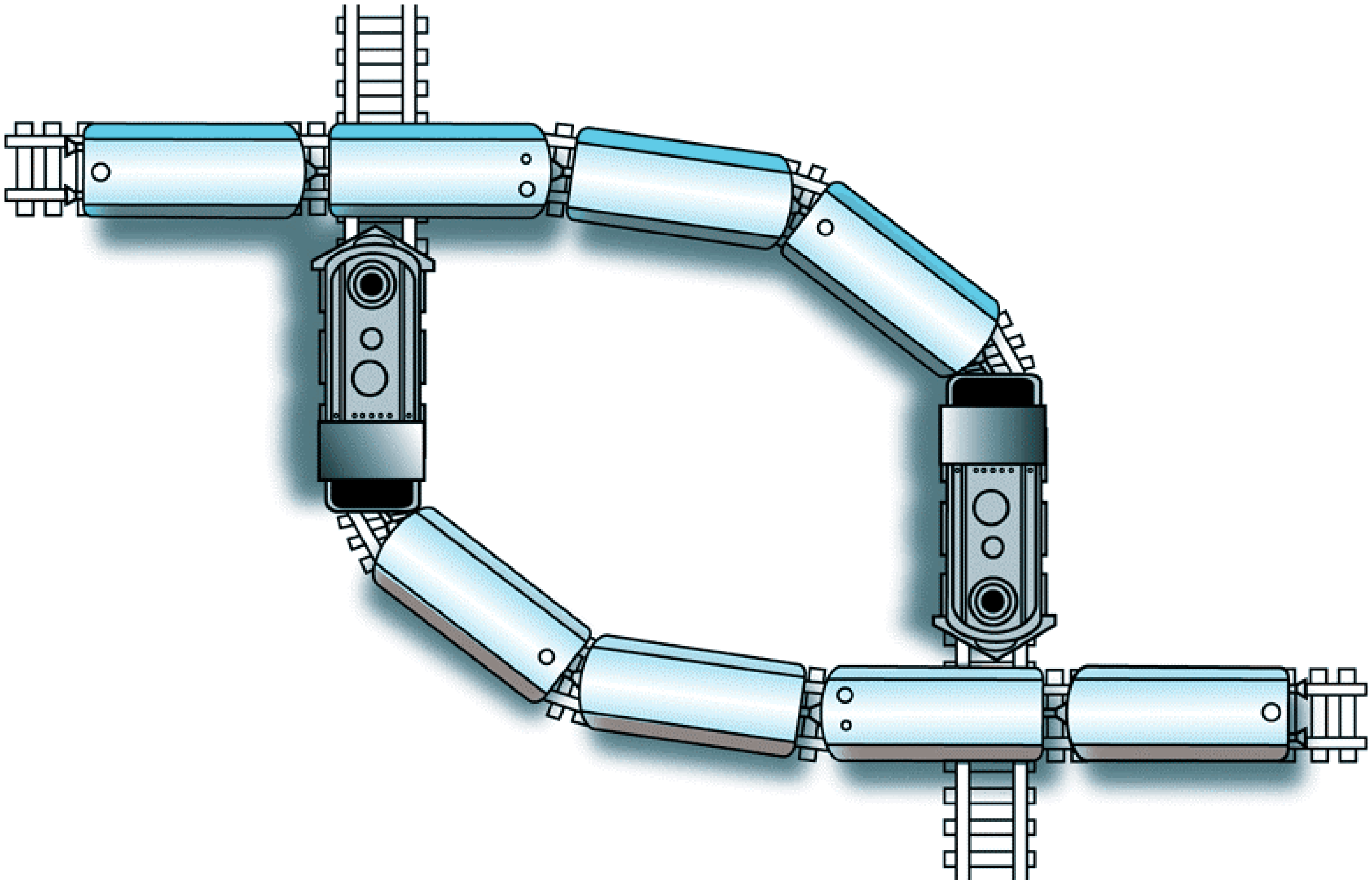
- Process: program in execution
  - Address space (memory) the program can use
  - State (registers, including program counter & stack pointer)
- OS keeps track of all processes in a *process table*
- Processes can create other processes
  - Process tree tracks these relationships
  - A is the *root* of the tree
  - A created three child processes: B, C, and D
  - C created two child processes: E and F
  - D created one child process: G

Unix Process

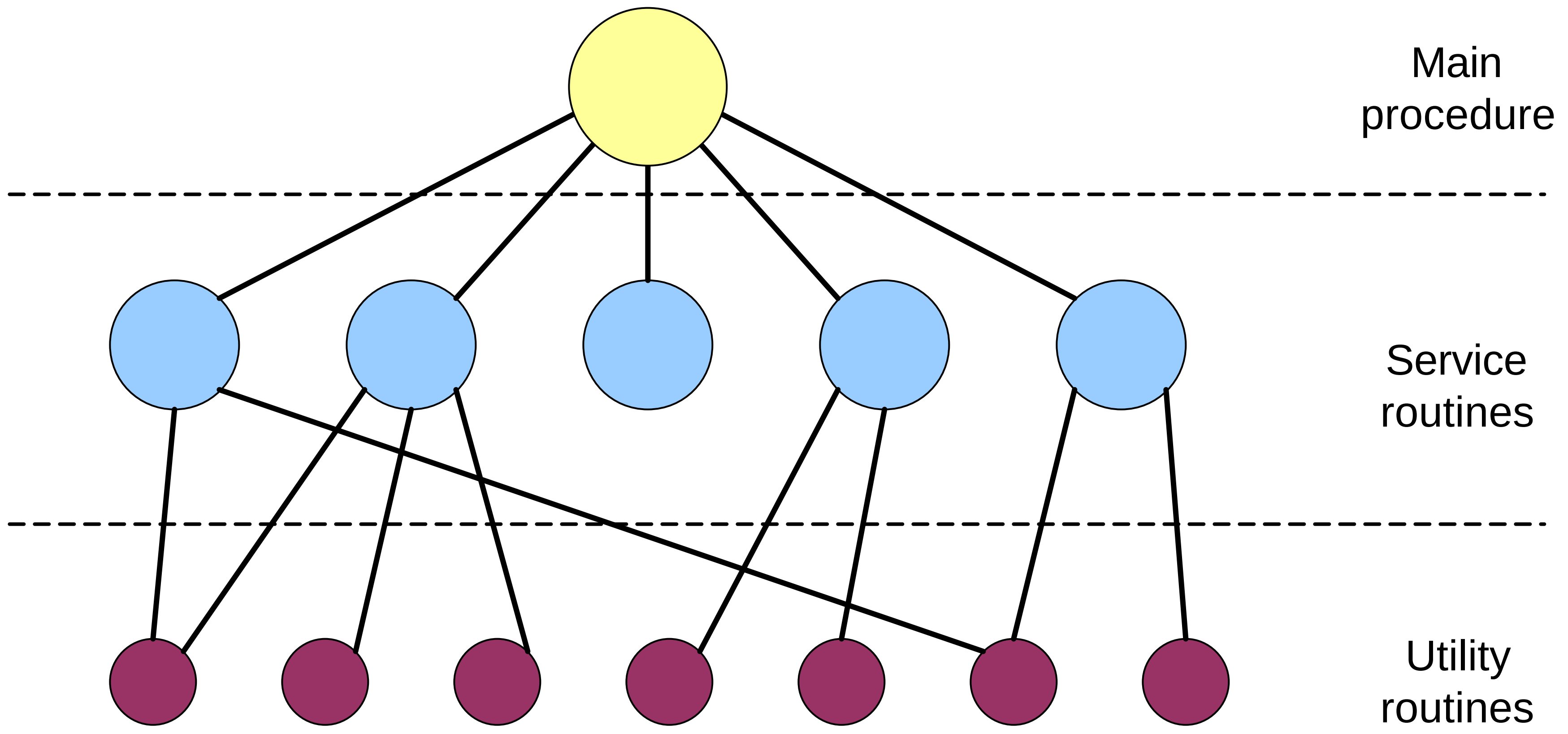


- Processes have three segments
  - Text: program code
  - Data: program data
    - Statically declared variables
    - Areas allocated by new
  - Stack
    - Automatic variables
    - Procedure call information
- Address space growth
  - Text: doesn’t grow
  - Data: grows “up”
  - Stack: grows “down”

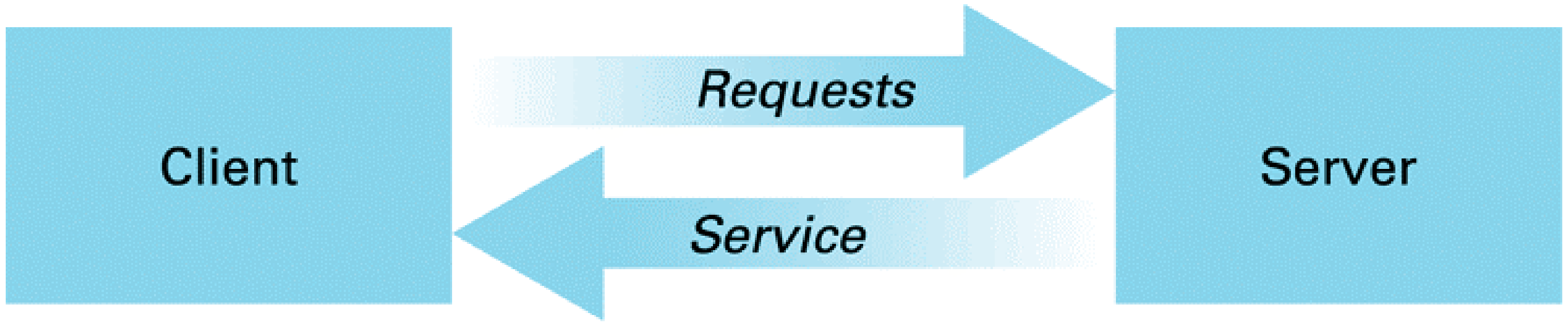
Deadlock

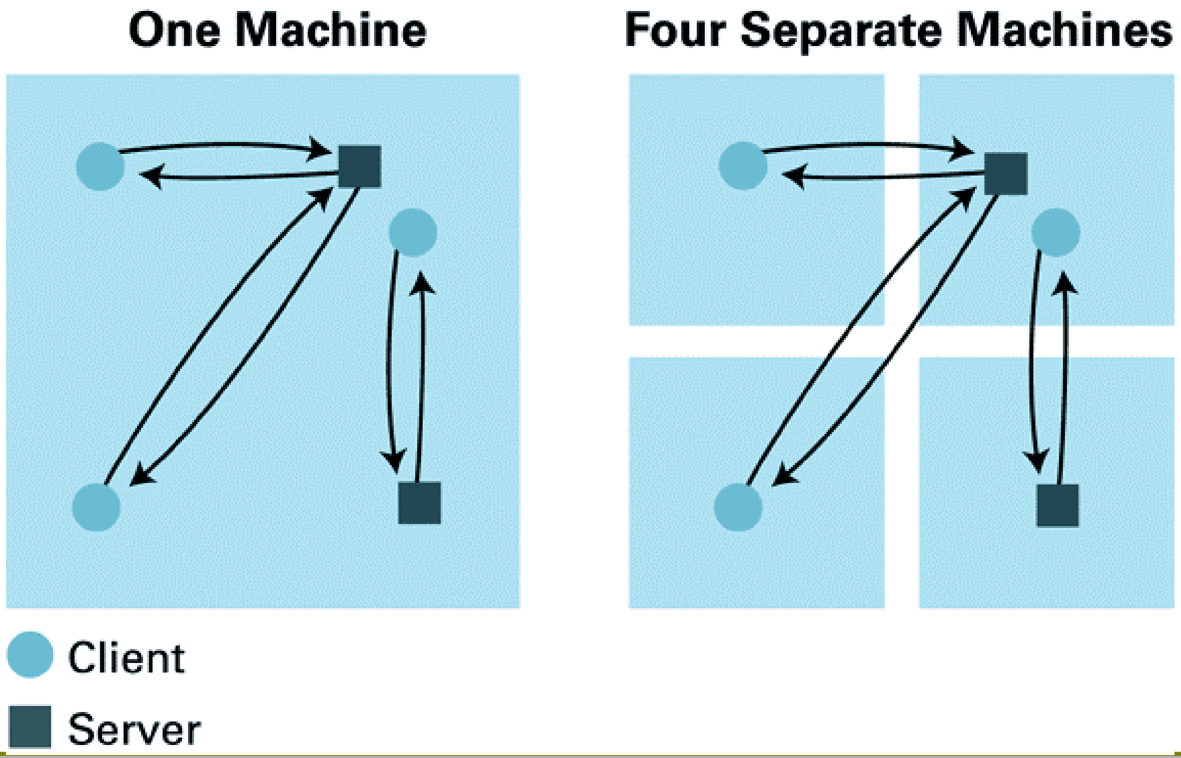


Monolithic OS

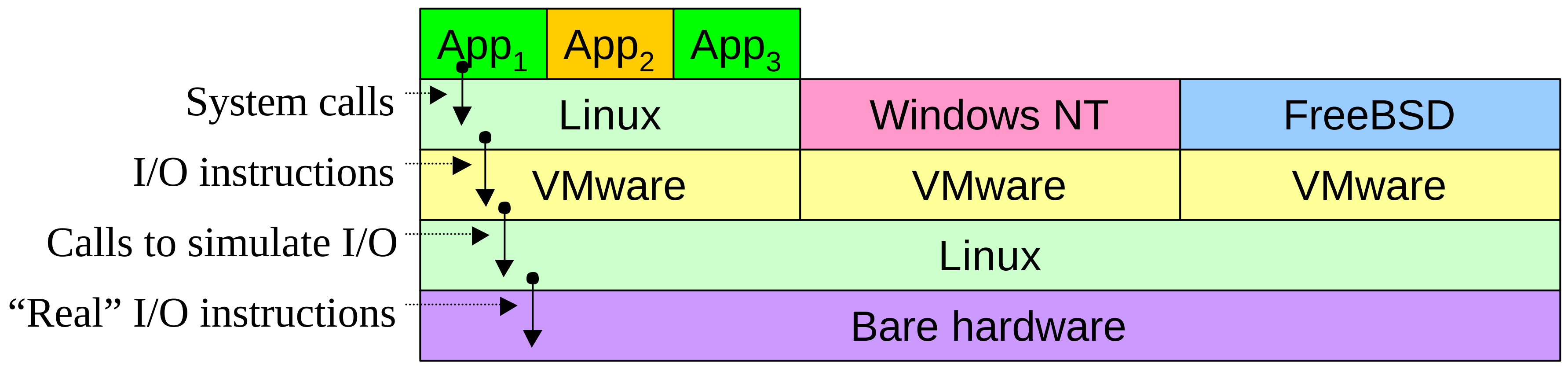


Client Server Model



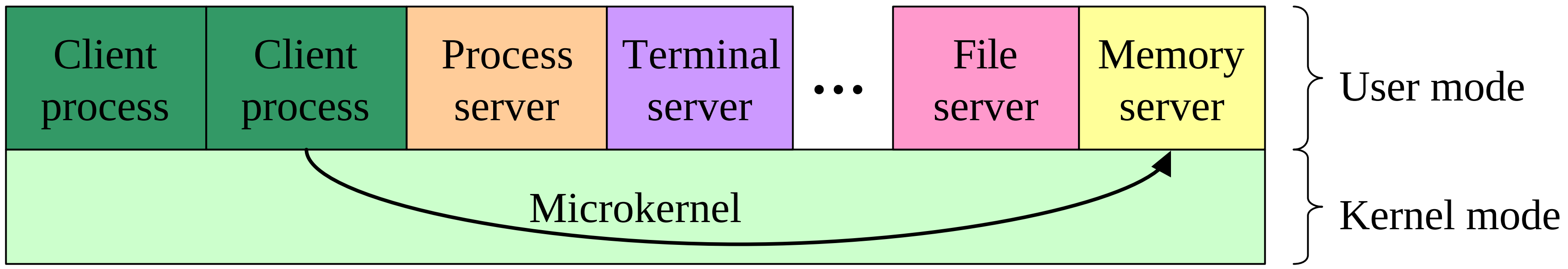


Virtual Machines



- First widely used in VM/370 with CMS
- Available today in VMware
  - Allows users to run any x86-based OS on top of Linux or NT
- "Guest" OS can crash without harming underlying OS
  - Only virtual machine fails—rest of underlying OS is fine
- "Guest" OS can even use raw hardware
  - Virtual machine keeps things separated

Client Server Model



- Processes (clients and OS servers) don't share memory
  - Communication via message-passing
  - Separation reduces risk of "byzantine" failures
- Examples include Mach