OS HISTORY

Module 1.3

Richard Newman

University of Florida

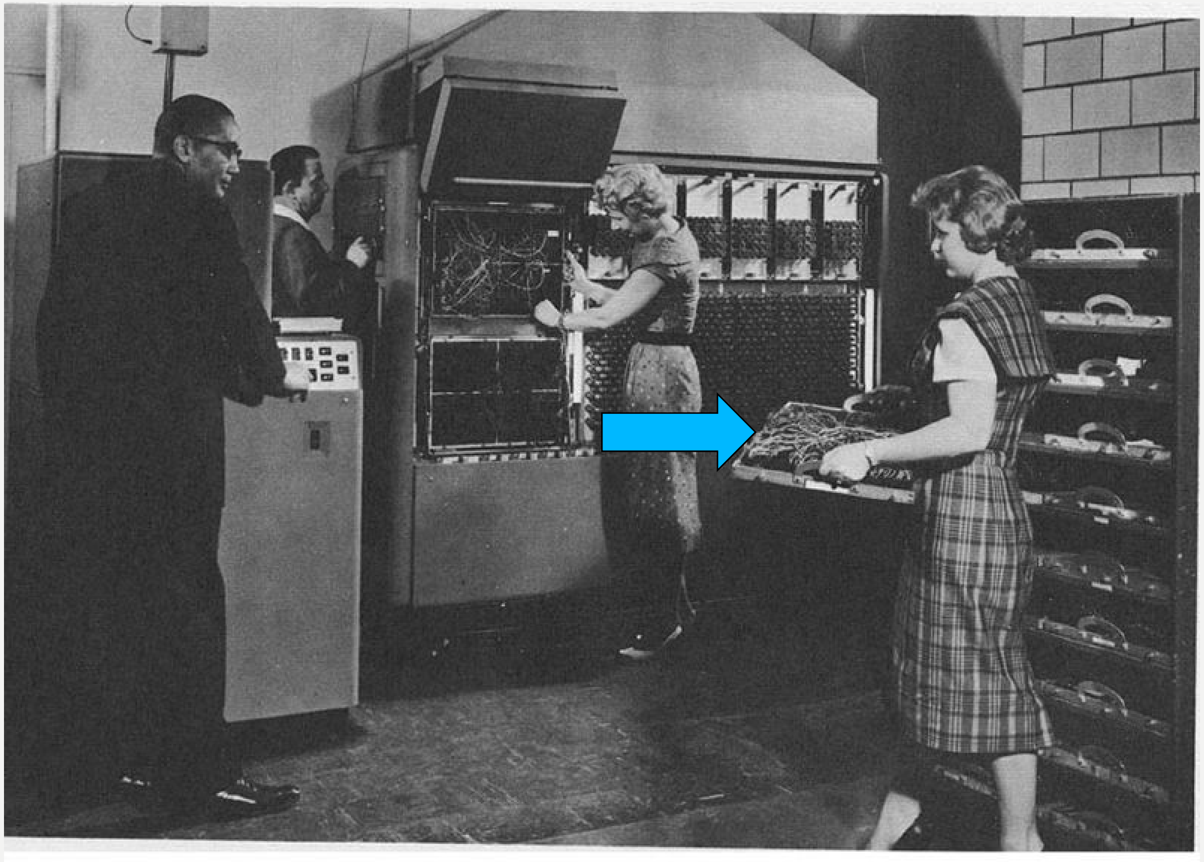
WHY WE CARE: ONTOGENY RECAPITULATES PHYLOGENY

- Each new “species” of computer
  - Goes through same development as “ancestors”
- Consequence of impermanence
  - Text often looks at “obsolete” concepts
  - Changes in technology may bring them back
- Happens with large memory, protection hardware, disks, virtual memory

HISTORY OF OPERATING SYSTEMS

- The first generation (1945–55)
  - vacuum tubes
- The second generation (1955–65)
  - transistors and batch systems
- The third generation (1965–1980)
  - ICs and multiprogramming
- The fourth generation (1980–present)
  - personal computers
- The fifth generation (1990–present)
  - mobile computers

Remmington Rand 409 (Univac)



Program is in plugboard!!!

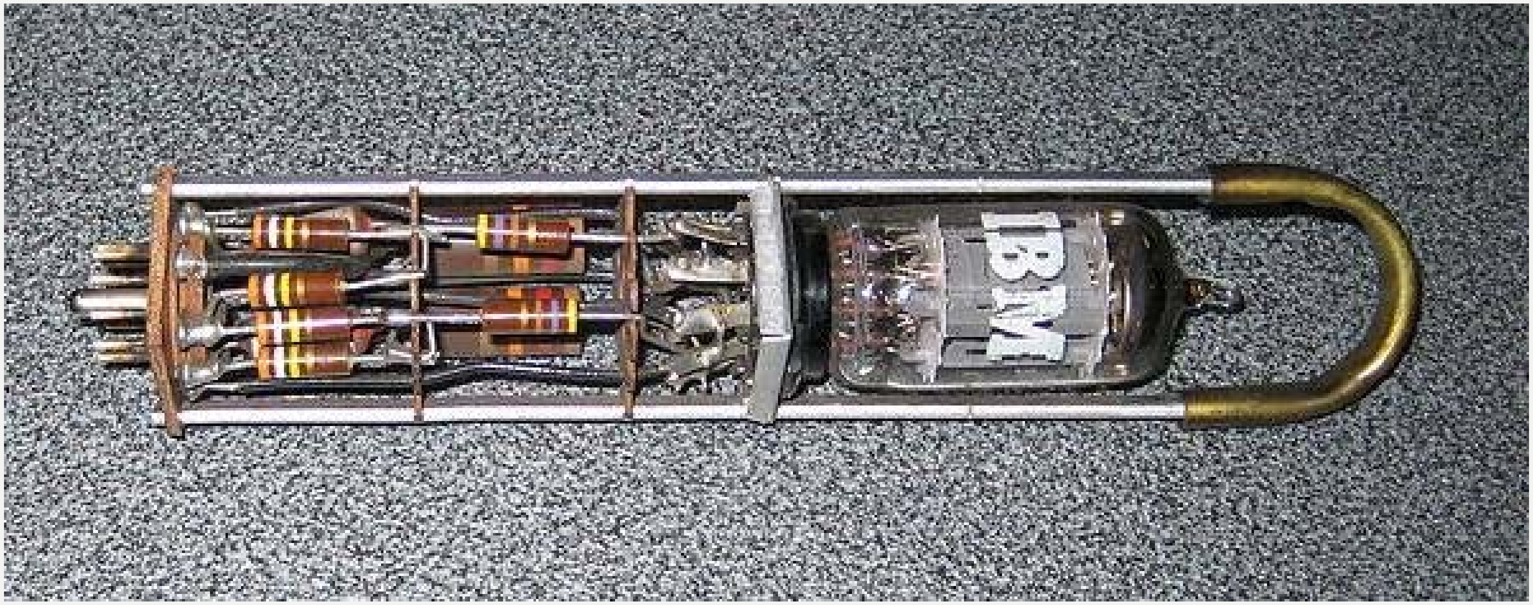
IBM 650

card sorter (L) reader/punch (C), computer (R)

IBM 650

front panel (L) drum (B), tubes (TL)

Vacuum Tube (IBM 650)



IBM 650 announced1953, sold until1962

Sperry-Univac AN-UYK/20

Program from toggle switches on front panel
Note 64K of core memory (bottom half)

OPERATING SYSTEMS GENERATIONS (Silberschatz and others – S/W)

- Generation 1 (50’s on)
  Resident Monitor – user = operator
  Single application program at a time
- Generation 2 (late 50’s – 60’s)
  Batch systems – user not operator, JCL
  2a – uniprogrammed batch
  2b – multiprogrammed batch
- Generation 3 (1960’s on)
  Timesharing systems
- Generation 4 (1980 – Present)
  Personal computers – user = operator

Did they improve productivity?

OPERATING SYSTEMS GENERATIONS (Silberschatz and others – S/W)

- Generation 5 (1980 – present)
  Network Operating Systems

  Remote commands, interoperability key
- Generation 6 (late 80's – present)
  Distributed Operating Systems

  Transparency key – homogenous systems
- Generation 7 (1990's on)
  Cooperative Autonomous Systems – independent systems – integration key

  Resource discovery, negotiation, etc.

  e.g., Service Oriented Architectures
- Generation 8 (2000's on)
  Cloud systems – computing as a utility

RESIDENT MONITOR

- Users reserved time on machine (hour scale)
  If not done when time up, too bad – try again later

- Useful to have a user interface and RM to allow you to
  - Read program and data in from card reader, tape drive, etc.;
  - Run program
  - Print results

RESIDENT MONITOR

- Also had utility programs
  - Compiler
  - Linker
  - Loader
  - Editor
- Also had libraries
  Sets of convenient procedures reused often
  - Math functions
  - Print formatting
  - I/O access

TRANSISTORS AND BATCH SYSTEMS

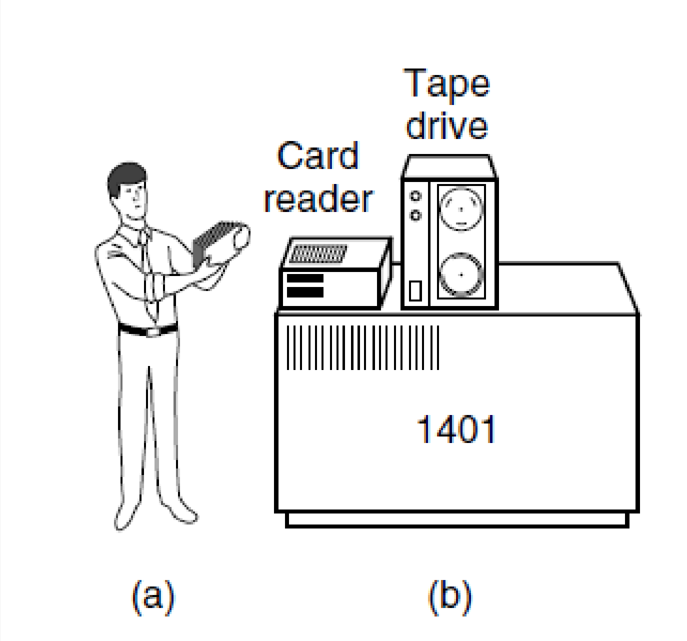


Figure 1-3. An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape.

TRANSISTORS AND BATCH SYSTEMS

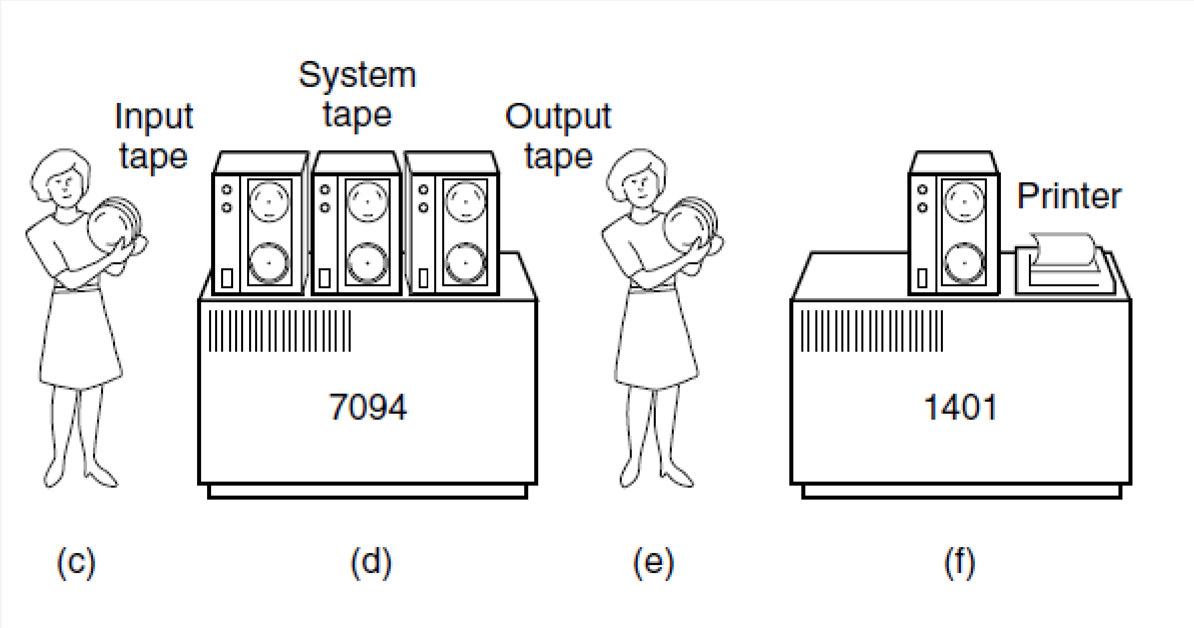


Figure 1-3. An early batch system. (c) Operator carries input tape to 7094. (d)7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

TRANSISTORS AND BATCH SYSTEMS

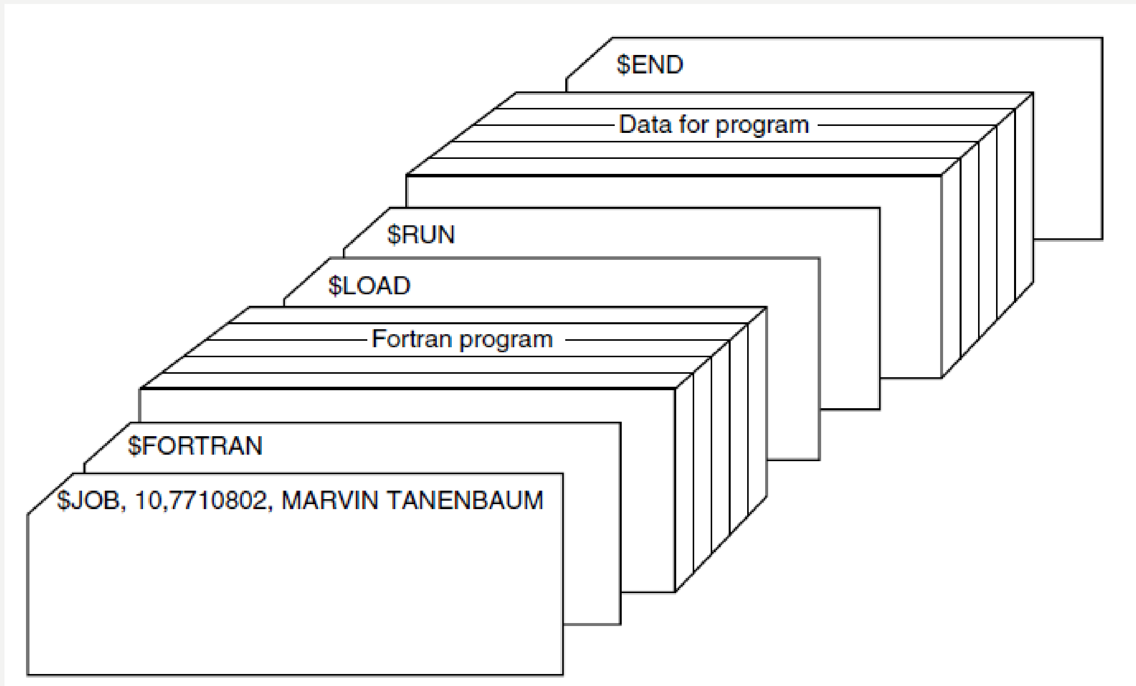
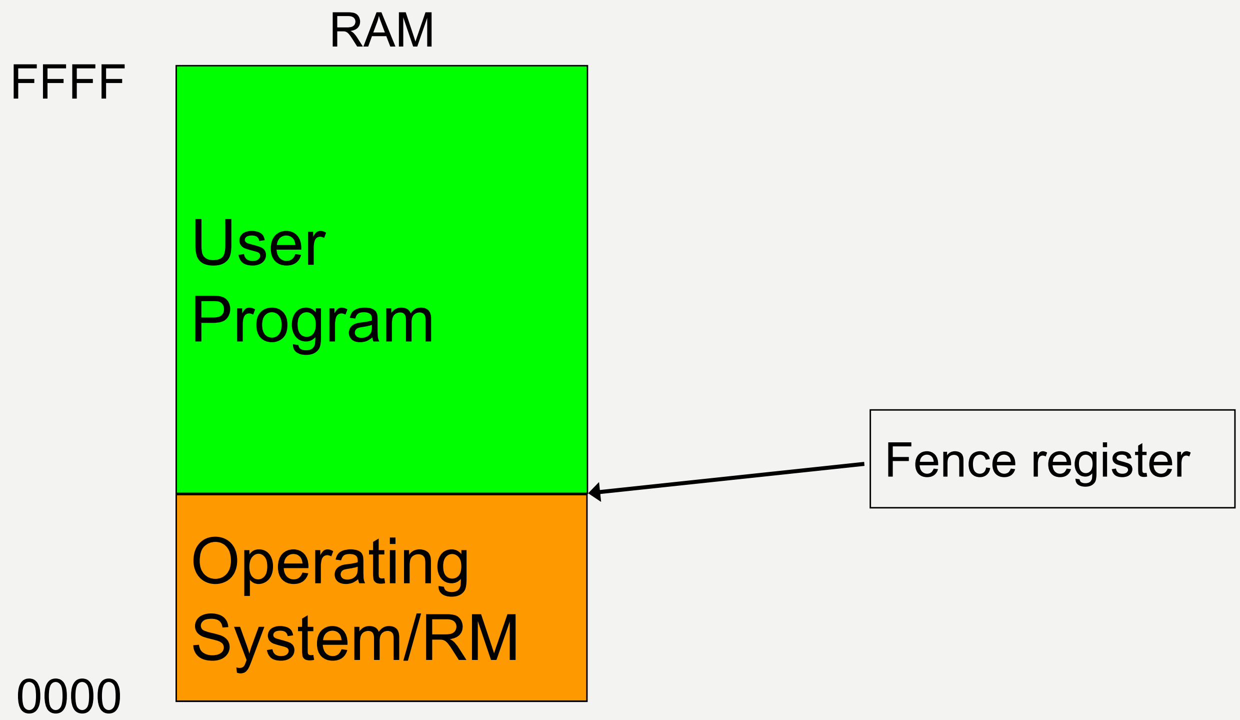


Figure 1-4. Structure of a typical FMS job.

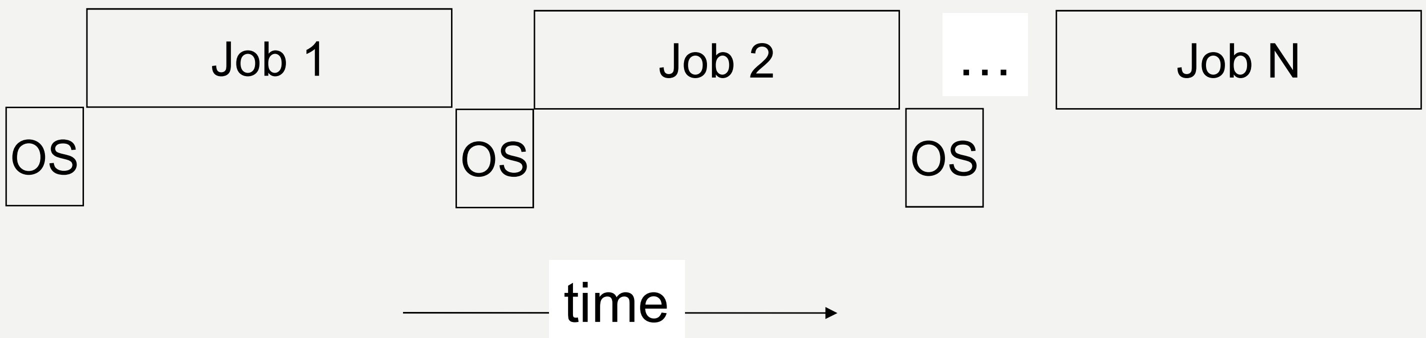
UNIPROGRAMMED BATCH SYSTEM



Uniprogramming – single process runs at a time

Protect OS/RM using fence register & test vs. address

UNIPROGRAMMED BATCH SYSTEM



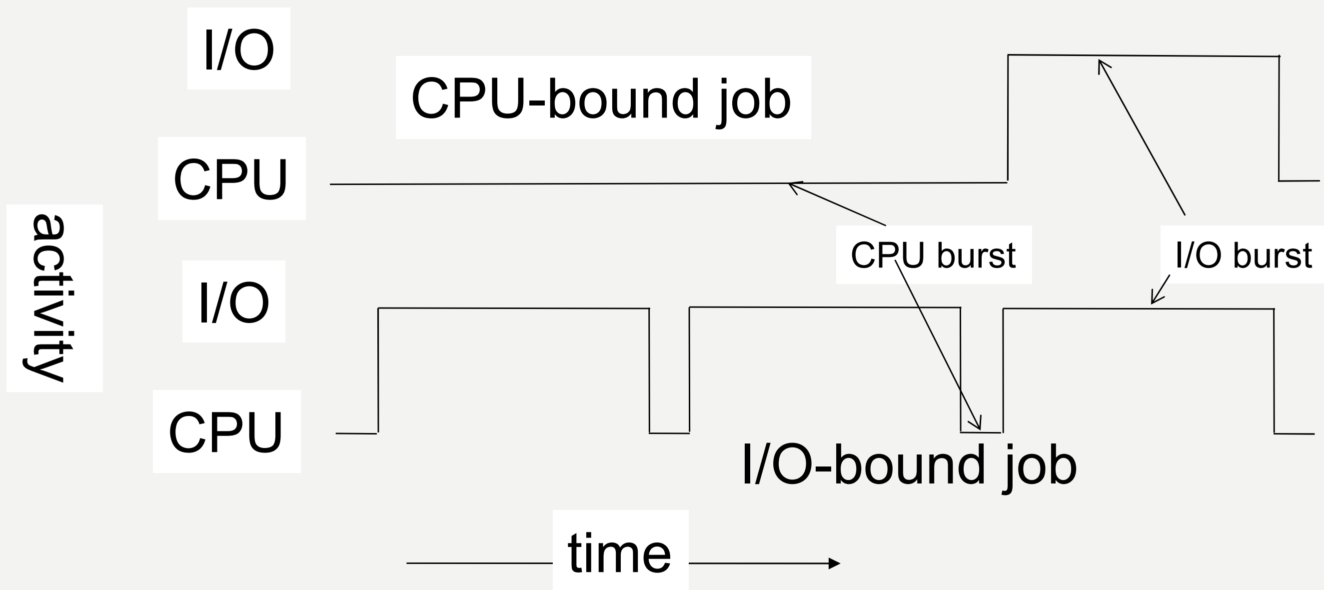
While there is another job

  Read control card for next job

  Load job

  Run job

UNIPROGRAMMED BATCH SYSTEM



I/O-bound jobs underutilize CPU

Even CPU-bound jobs use little CPU when doing I/O

Desirable to overlap CPU use of one job with I/O of other job(s)

ICs AND MULTIPROGRAMMING

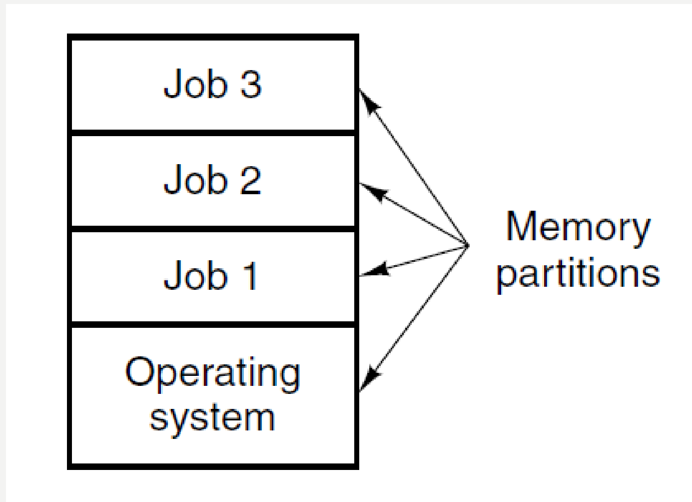
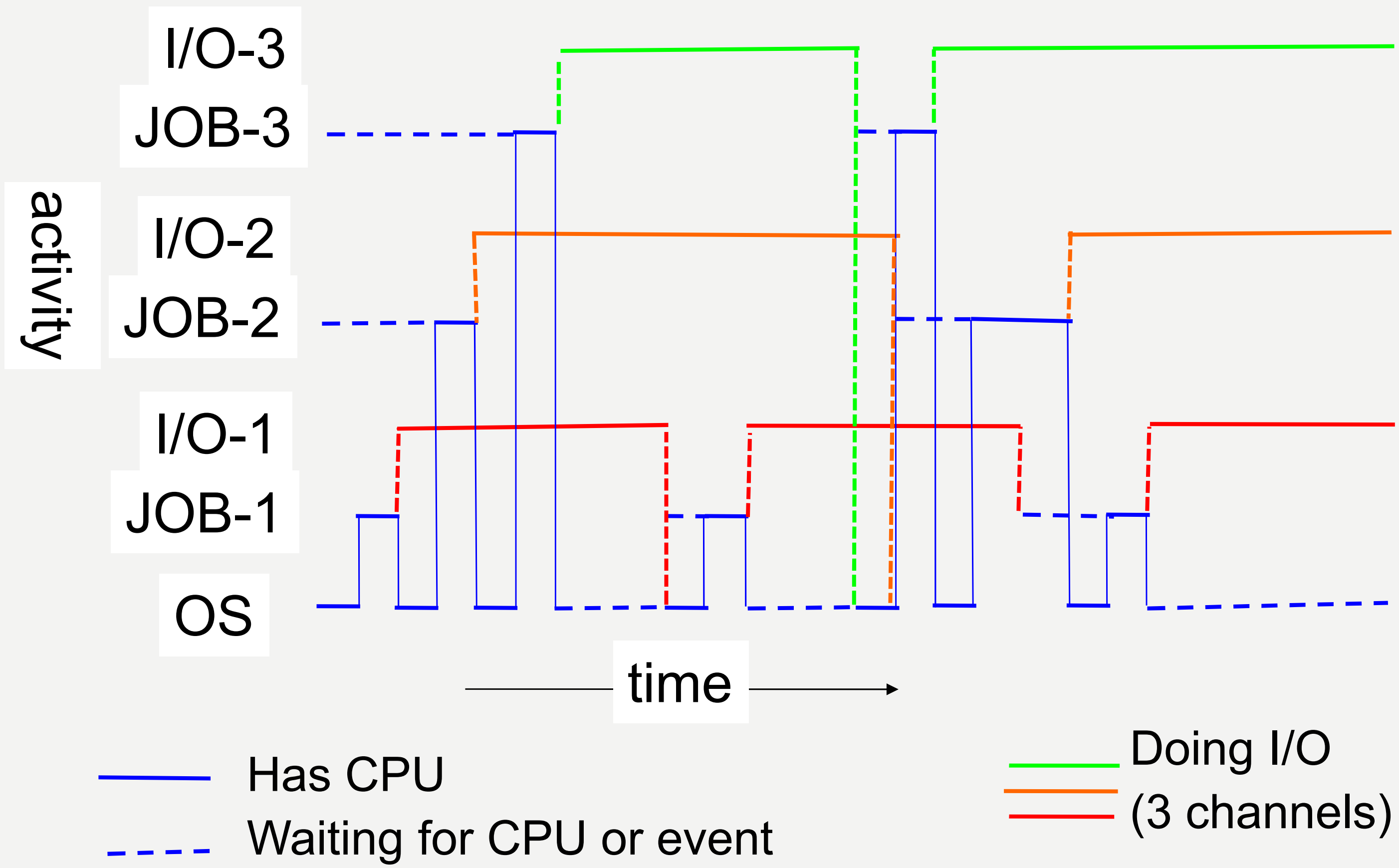


Figure 1-5. A multiprogramming system with three jobs in memory.

MULTIPROGRAMMED BATCH SYSTEM



Modeling Multiprogramming

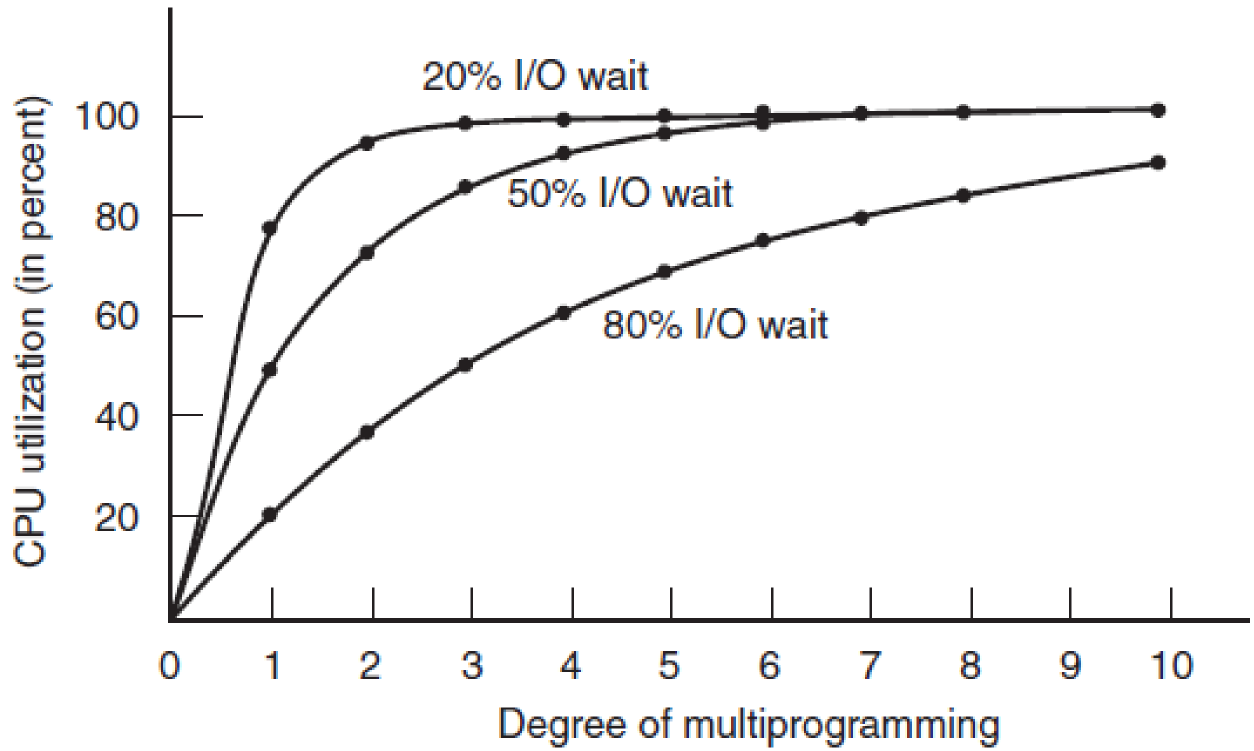


Figure 2-6. CPU utilization as a function of the number of processes in memory.

QUESTIONS

- How are jobs loaded?
- Where do we put them?
- How do we keep them from violating the space of other jobs?
- How do we keep them from violating the OS?
- How do we decide which job gets the CPU?
- How do we keep one job from hogging the CPU?
- How do we manage access to I/O devices?
- How can we provide common services to the jobs?
- What about persistent storage?

Summary

- Why we care about OS history
  - Cyclic nature of evolution
  - Develop solutions from simple to complex
- OS Generations
  - By hardware (Tanenbaum)
  - By purpose and capability (Silberschatz)
- Multiprogramming
  - Overlap I/O and computing
- Protection
  - Memory
  - Instructions