Systems 3

Principles

Marcel Waldvogel

Department of Computer and Information Science
University of Konstanz

Winter 2019/2020

Timetable

- Lecture: Thursday (15:15) && Friday (11:45)
  - Marcel Waldvogel (marcel.waldvogel@uni.kn)
- Exercise: Thursday (8:15 || 11:45 || 17:00)
  - Klaus Herberth (klaus.herberth@uni.kn)
  - Christian Löhle (christian.loehle@uni.kn)

Exam Admission

- Work in a group of two students
- Your solutions to every exercise need to show serious deliberation
- You are allowed to have two failures (three strikes and out)
- If you get 50% in total you will get a 0,3 exam bonus

Teaching materials

are available via git[1] on the University GitLab[2]. One member of the group forks the repository[3] into her personal space and

- **grants access** to the other member (at least 'Developer'),
- **shares** it with the tutors (at 'Owner' or 'Master' level).

You can then browse all files in the repository, which includes instructions on how to clone it into a local repository.

[1] https://git-scm.com/
[2] https://git.uni.kn
[3] https://git.uni.kn/disy/lectures/systeme-3-2019

Chapter Goals

- Operating Systems[4]: What and why?
- What are their functions and how did they evolve?
- First insights into structure and processes.
- Does the CPU[5] control the OS or vice versa?

[4] OS

[5] Central Processing Unit aka[6] 'the processor'

[6] "also known as"

What is an Operating System?

- Where do we find them?
- What do they do there?
- What are common operating systems (OS)?

What does an Operating System do?

Traditional goals

Hardware abstraction

1. Simplify application development
2. Portability
3. ...

Resource management

1. Share CPU among multiple processes
2. Share network cards among multiple services
3. ...

What does an Operating System do?

Today: Communications gateway

Enable communications

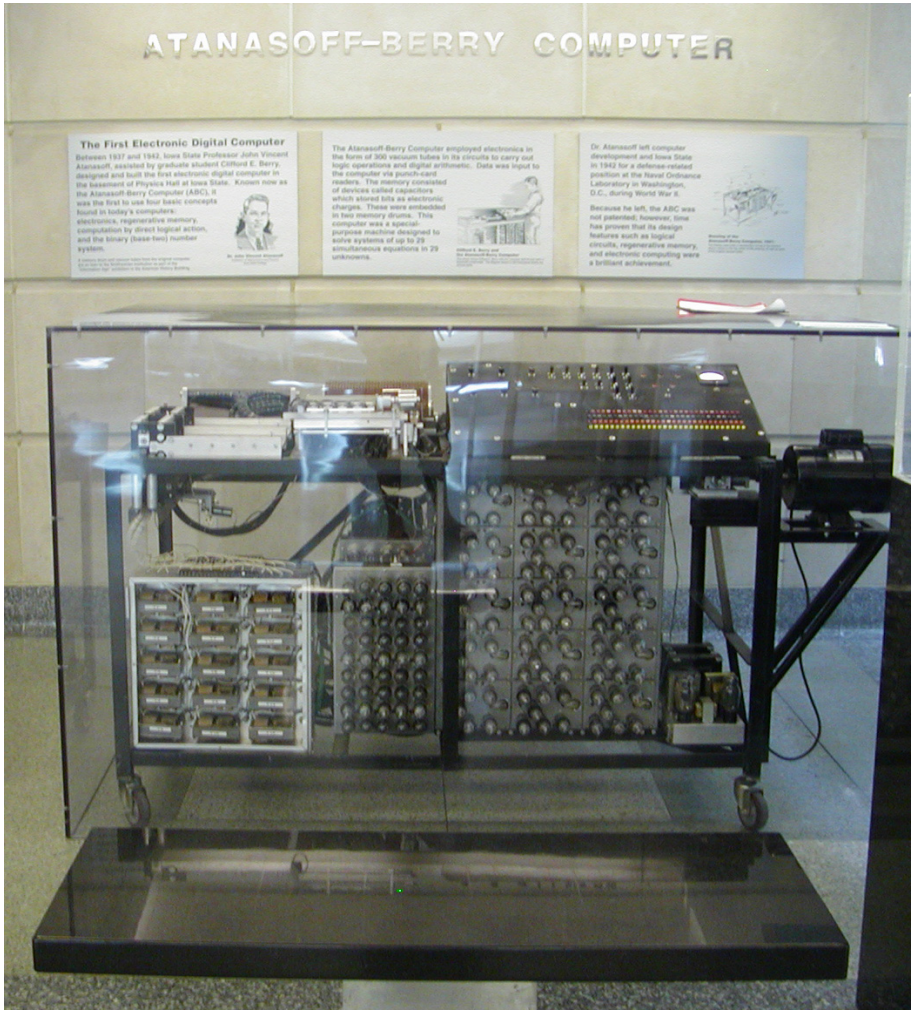
1. Between processes
2. Between systems

Restrict/select communications

1. Not everyone should be able to interfere with everything
2. Only when a process is ready
3. Only what a process expects

Hardware abstraction, resource management, and communications are independent dimensions (more or less).

Gen1 ∼1950: Tubes, plugboards, core memory



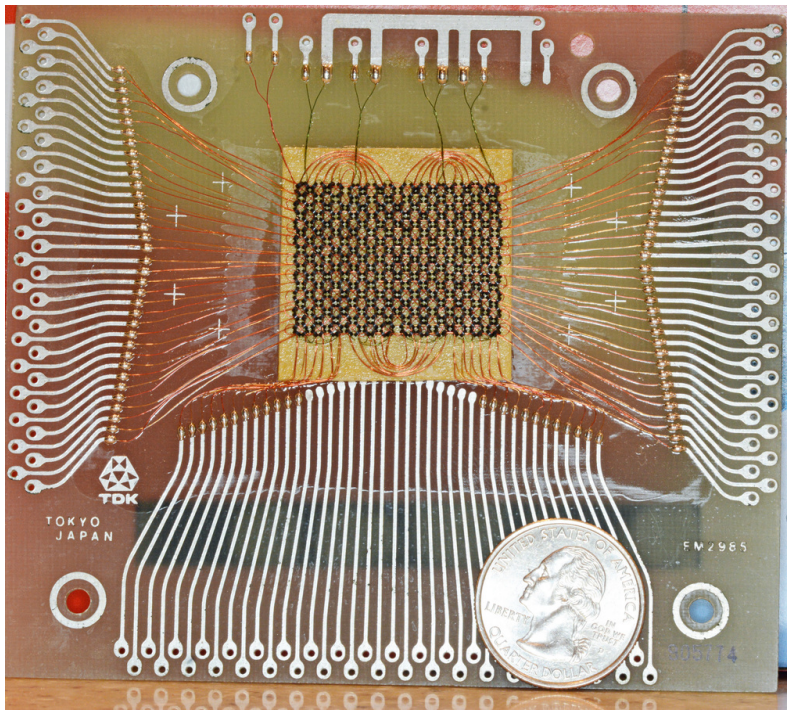
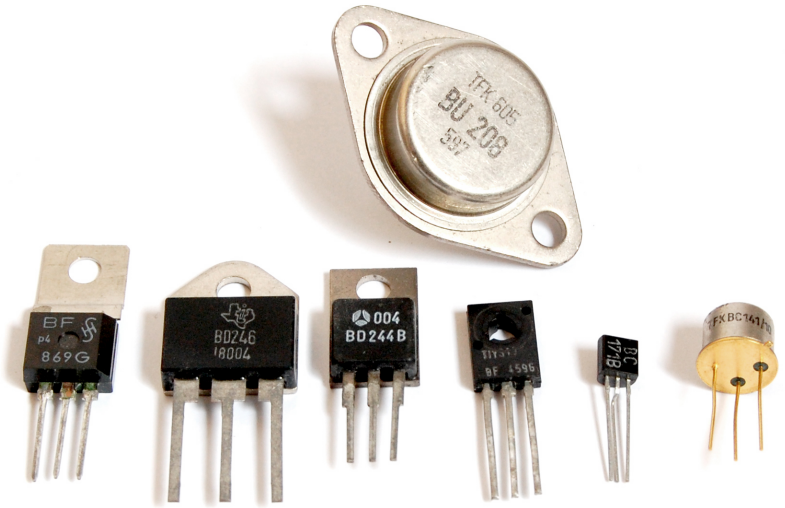


Photo credits: Wikpedia users Manop, CC BY-SA 3.0;
Stefan Riepl (Quark48), CC BY-SA 2.0 de;
Bubba73 (Jud McCranie), CC BY-SA 4.0.

Gen2 ∼1960: Transistors and batch systems

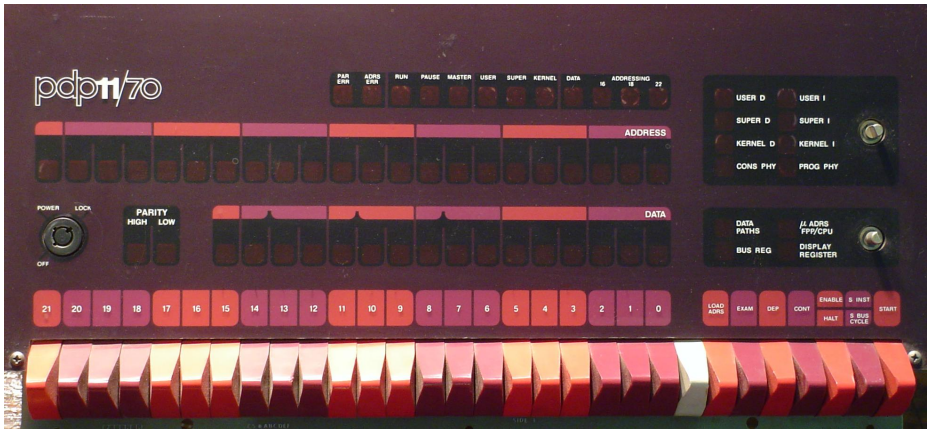


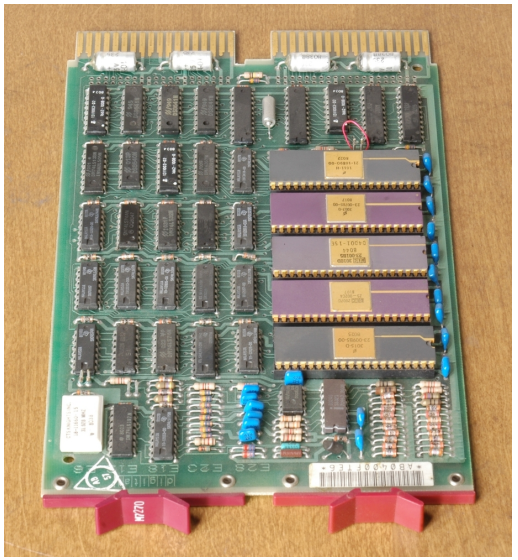
Batch processing

Programs can be run quickly one after another to reduce turn-around time and increase utilization.
Today used for e.g. HPC/Supercomputing, end-of-day processing.

Photo credits: Wikpedia users Benedikt.Seidl, public domain; Grkauls, public domain.

Gen3 ∼1970–1980: ICs and multiprogramming





Multiprogramming

Multiple programs (of different users) can be run seemingly in parallel.
Today used for e.g. personal computers, mobile devices, even IoT.

Photo credits: Wikpedia users Stefan_Kögl, CC BY-SA 3.0; Retro-Computing Society of Rhode Island, CC BY-SA 3.0 (2) .

Gen4 1980+: Personal computers

Single-user

Computers were getting cheap enough to be used by single users, instead of using terminals to a central computer. First without networking.

Today many devices are used by mostly a single user, but still run multi-user operating systems. Reasons: Security, compatibility, ...

Photo credits: Wikpedia users Hydrargyrum, CC BY-SA 3.0; Ruben de Rijcke, CC BY-SA 3.0; Rama, CC BY-SA 2.0 fr.

Gen5 1990+: Mobile devices etc.

Mobile

Faster processors, higher integration, cheaper prices, and lower power consumption made computers more portable and almost ubiquitous (IoT).

Wireless/mobile networking, cloud computing, isolation between processes of the same user (sandboxing), virtualization.

Photo credits: Wikpedia users Snowmanradio, CC BY-SA 3.0; Blake Patterson from Alexandria, VA, USA; Source (WP:NFCC#4), Fair use.

OS History

1960s

CTSS Time-sharing (processes)

VM/CMS Virtualization (isolation)

Multics Single-level store, hierarchical file system, dynamic linking, IPC, hot-swapping

1970s

Unix ← Multics; modularity (everything is a file, user-level shell, byte stream, text files, pipes), portability, documentation

CP/M (← Unix); OS modularity (BIOS, BDOS, CCP), drive letters, OS for microcomputers/personal computers, Ctrl-Z for EOF

VMS File versions, file databases, multiprocessing/partitioning

Networking, email

OS History (cont'd)

1980s

MS-DOS ← CP/M

MINIX ← Unix; teaching

GNU Unix as free software[7]

Mach Micro-kernel (modularity in the kernel)

GUI Xerox, Apple, Digital Research/Atari, Amiga, Microsoft, X11

Networking Multiple protocols; "OSI"; TCP/IP, WWW

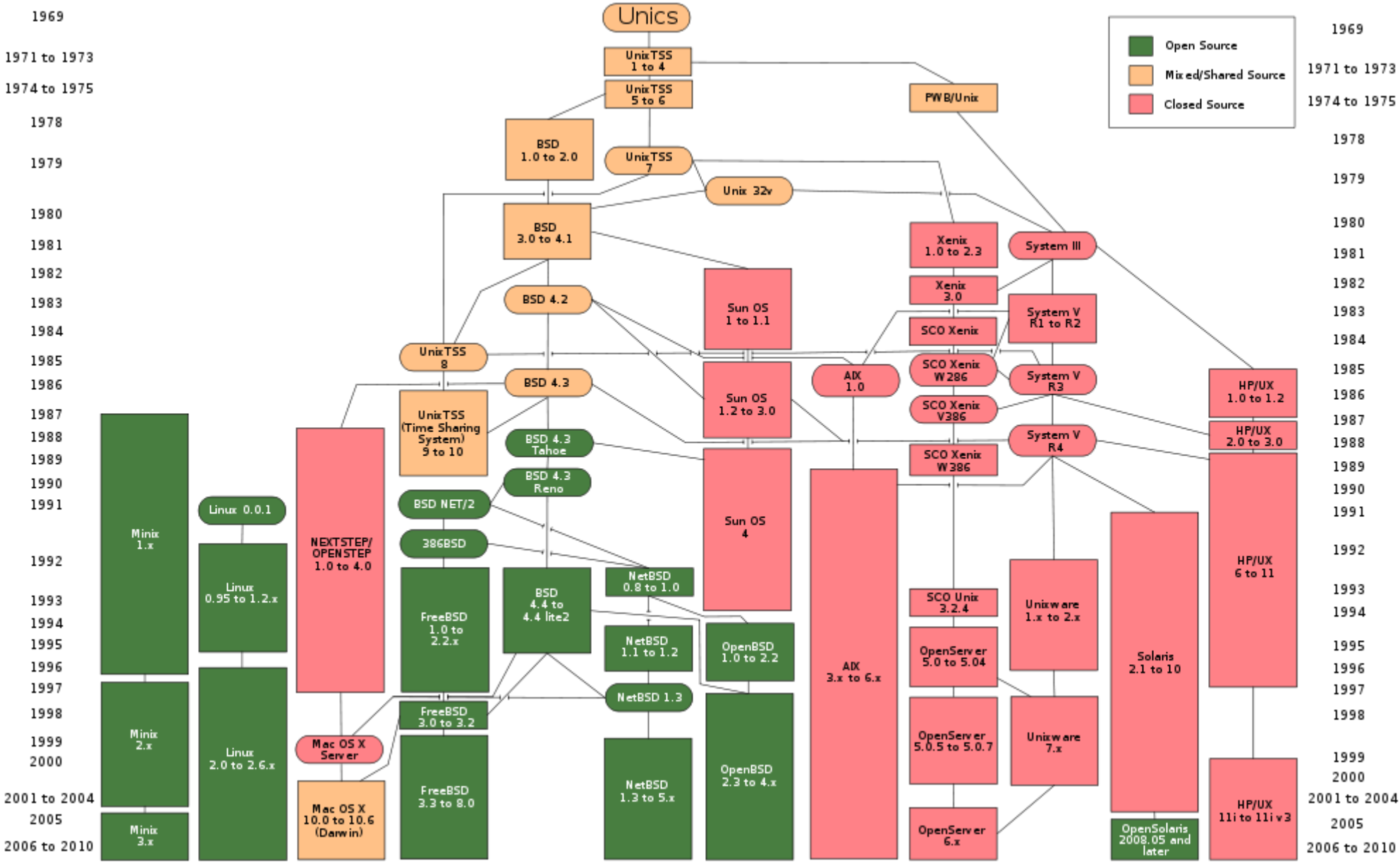
1990s

Linux Free software Unix-compatible kernel[8]

Browser Browser as GUI

[7] user-level software, kernel (Hurd) stalled

[8] User-level tools from GNU project, hence, often called *GNU/Linux* in combination



Source: Dr. Adamkó Attila https://gyires.inf.unideb.hu/GyBITT/20/ch01s02.html, crediting https://www.levenez.com/unix/

Market shares

Mobile

- Android (76%) ← Linux
- iOS (22%) ← Unix

Tablet

- iOS (73%) ← Unix
- Android (27%) ← Linux

Desktop

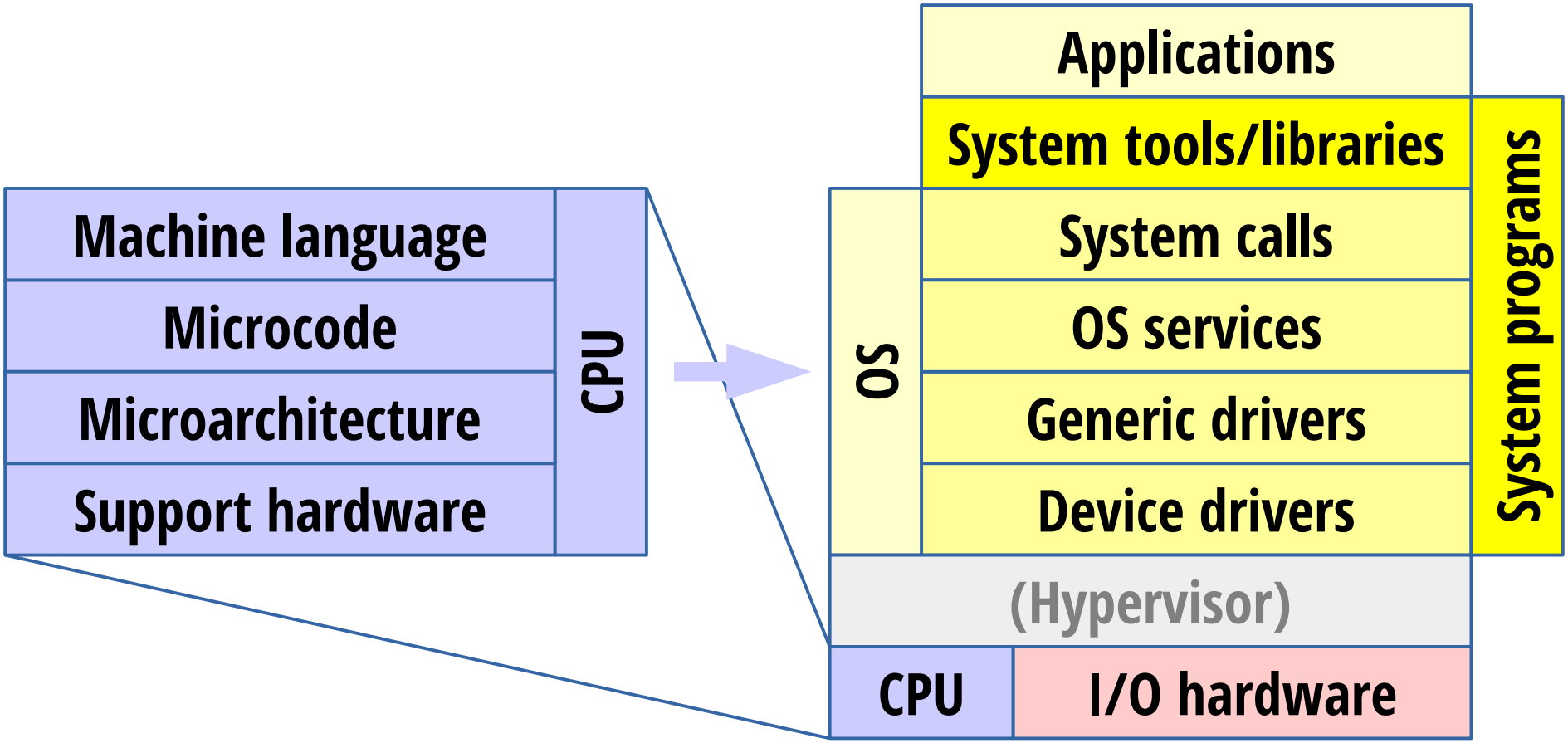
- Windows (80%) ← MS-DOS, VMS
- OS X (15%) ← Unix
- Linux (2%) ← Unix

Server

- Cloud: Linux 90%
- Home router: Linux ∼100%
- HPC: Linux ∼100%
- Web servers: Linux 70. . . 97%
- IoT: Linux 72%
- Business: Windows-mostly(?)

Source: Clients StatCounter, September 2019; servers MakeUseOf, March 2018, James Playfair on Quora, IT Pro Today/Eclipse Foundation

The Modern Computer System



1. A main program that initializes the system and the first process(es)
2. A set of service procedures that carry out the process' system calls
3. A set of utility procedures that help the service procedures

The Process

Properties

- Environment for user programs
- Unit for the OS to manage running programs (=processes)
- Created by other processes

Most programs are written to run in processes. It becomes our little world with which we interact with the OS using system calls. The OS remains a black box with a defined API.

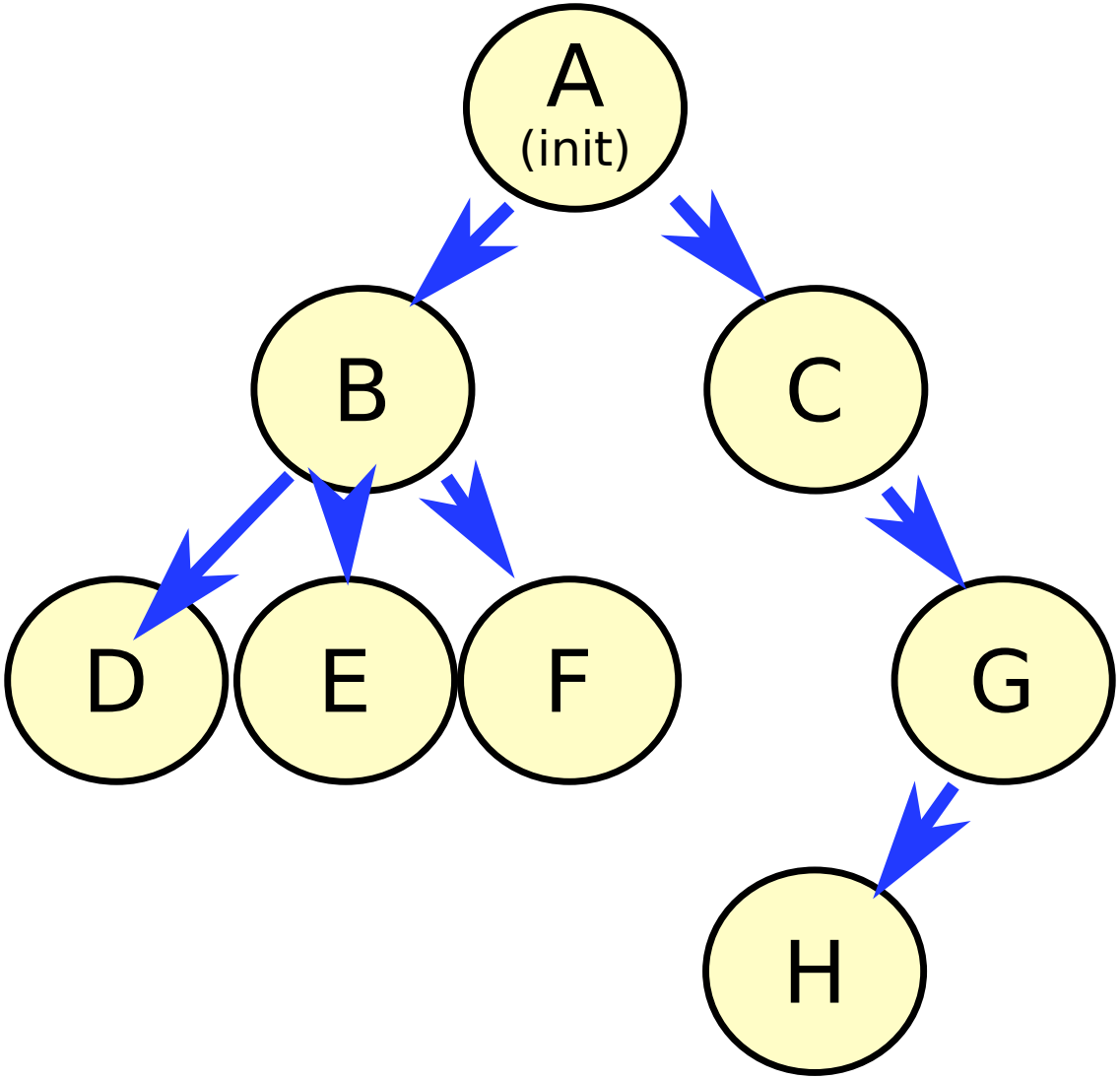
Process vs. program

Program The (static) code to be executed.

Process The environment for running a program (processor state, OS state (open files etc.), . . .)

Multiple processes may run the same program at the same time.
On Unix, a single process may run multiple programs in turn.

Process Tree



System Calls

OS is a black box

- How to get into this box?

Entry options

- Function call opcode? (Defeats isolation!)
- Special opcode for system calls! (Gatekeeper)

Goals for separation

- Isolation
- Security
- Abstraction

→ How can we call into an isolated environment with elevated rights?

System Calls

Example system calls

Files	open(), close(), read(), write()
Processes	fork(), exec(), exit(), getpid(), getppid()
Networking	socket(), bind(), listen(), accept(), ← Files
Memory	brk(), sbrk(), mmap(), munmap(). Not malloc(), free()!
IPC	pipe(), shmat()
Resources	setrlimit()
Users	getuid(), setuid(), getgid(), setgid()

Schedule

Week	Thursday	Friday
0	Principles	**C Basics**
1	**Pointers**	—
0	Principles	**C Introduction**
1	**C Basics**	**Pointers**
2	I/O	Memory
3	**Memory in C**	Memory Expansion
4	Process Model	Process Synchronization
5	**Threads**	Scheduling
6	**Big Programs**	Filesystem
7	OS Structure	OS Design
8	Vulnerabilities	Cryptography
9	**Libraries**	Virtualization
10	Meltdown	Security
11	Data security	Privacy
12	Defensive programming	Spare
13	Review	Q&A
14	Exam	—

Literature

- Andrew S. Tanenbaum, Albert S. Woodhull. *Operating systems: design and implementation*. Uni-KN kid 346/t16a + lbs 843/t16b
- Brian W. Kernighan, Dennis M. Ritchie. *The C Programming Language*. Uni-KN kid 248 k27.
- Peter van der Linden. *Expert C Programming — Deep C Secrets*. Uni-KN kid 248 v16.