

Introduction to Operating Systems

CPSC/ECE 3220 Spring 2022

Lecture Notes

Chapter 1

Lana Drachova

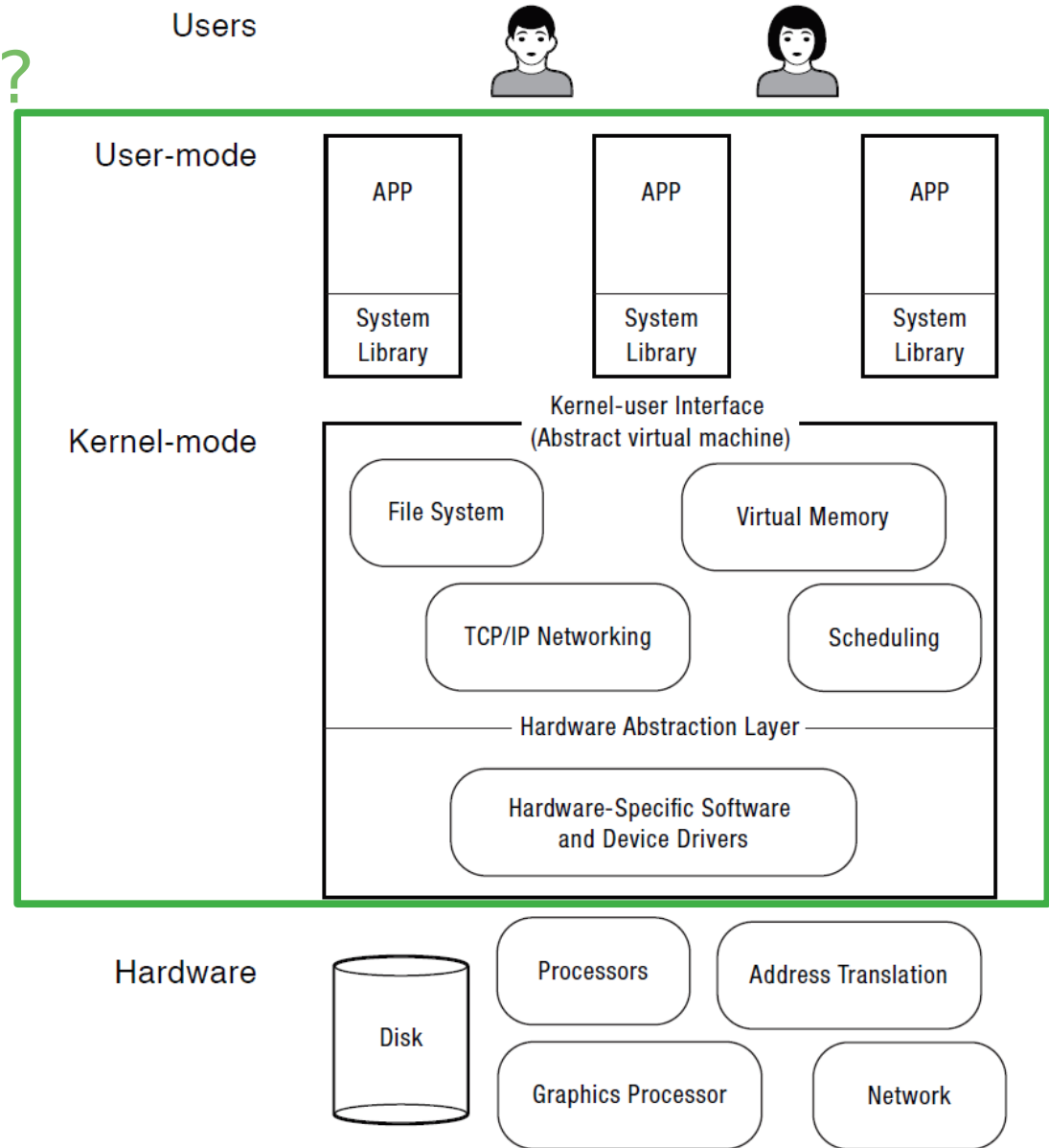
(adapted from Mark Smotherman's and Tom Anderson's slides)

Question

- What is an operating system?

What is an operating system?

- Software to manage a computer's resources for its users and applications
- Can exceed 50 million lines of code



Question

- Where can you find a operating system?

Types of Operating Systems

- Mobile OS
 - Android, iOS
- Desktop / Server OS
 - Linux, Windows
- Embedded OS
 - L4, Tizen, QNX, webOS
- Real Time OS
 - VxWorks
- Network OS
 - Cisco IOS
- Mainframe OS
 - z/OS

Tizen Embedded OS

- 2012
- Linux-based mobile OS by Samsung Electronics.
- Open source software written in html5, C, C++.
- Tablets, smart TVs, wearables, refrigerators, etc.



Samsung Galaxy
Watch



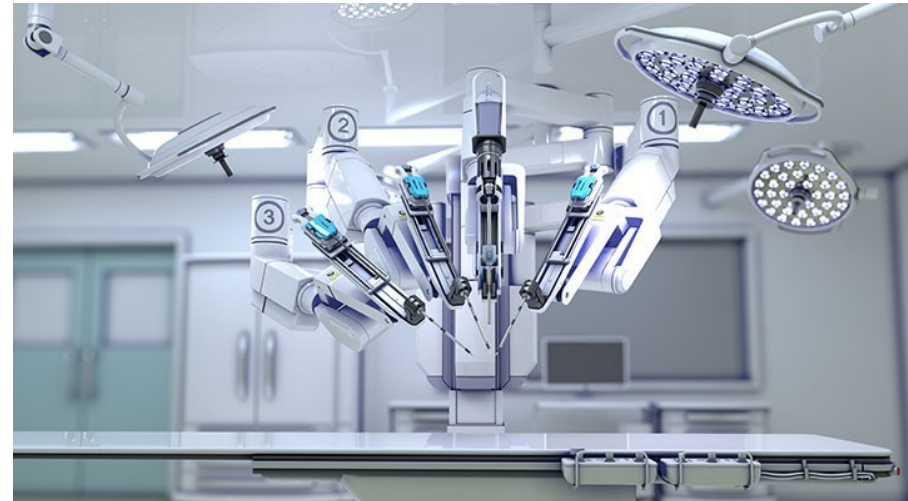
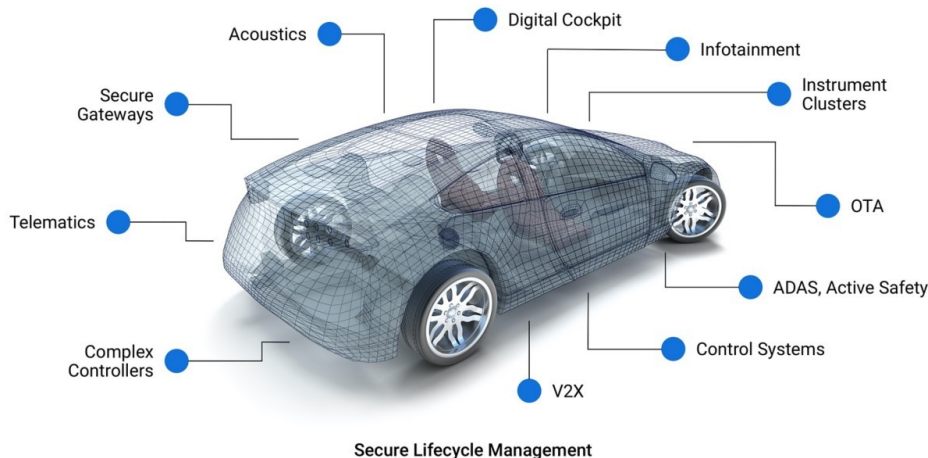
NX300 camera



Family Hub 3 Smart Fridge

QNX Embedded OS

- BlackBerry's Commercial closed-source Unix-like RTOS for embedded systems, 1980s
 - Advanced driver assistance systems, digital instrument clusters, connectivity modules, handsfree systems, and infotainment systems in:
- Audi, BMW, Ford, GM, Honda, Hyundai, Jaguar Land Rover, KIA, Maserati, Mercedes-Benz, etc.
- + Medical Devices

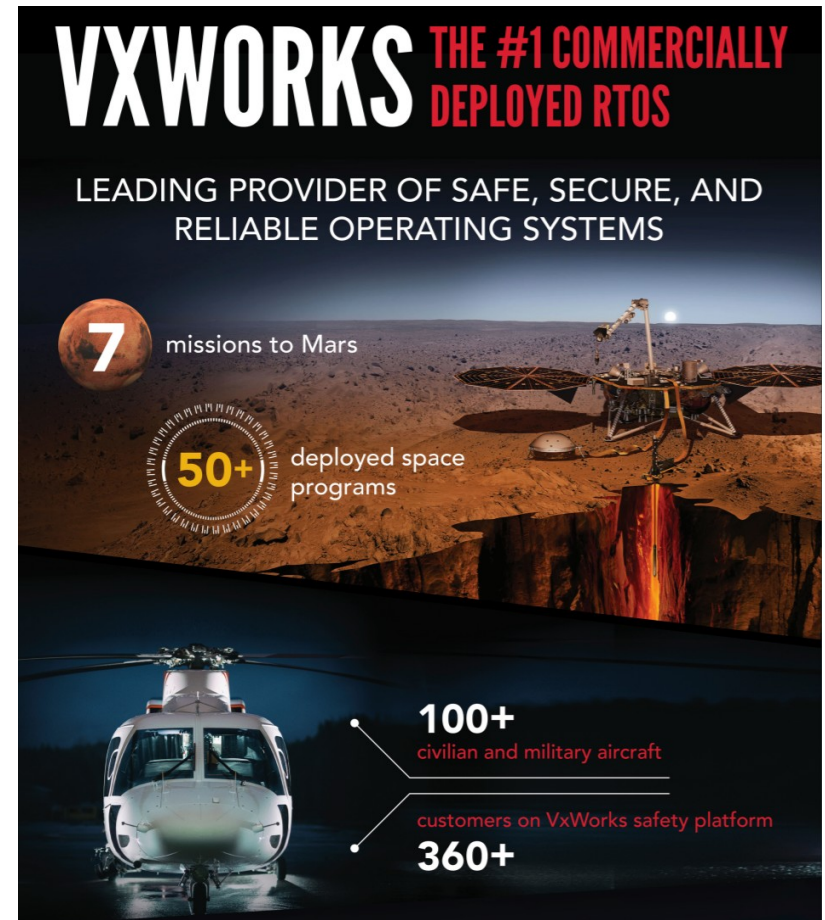


VXWorks OS

- 1987 Proprietary (by Wind River System) RTOS for embedded systems that require RT, deterministic performance, security and safety
- Aerospace and defense industries, medical devices, robotics, industrial equipment, automotive, transportation:
 - Mars Rover 2020
 - Honda's Asimo
 - Boeing 787
 - Brother printers
 - Radiation therapy devices
 - MRI machines



Img src: google.com

A promotional graphic for VxWorks OS. The top half features a Mars rover on a reddish-brown planet surface. The bottom half features a white helicopter. Text and statistics are overlaid on the image.

VXWORKS THE #1 COMMERCIALY DEPLOYED RTOS

LEADING PROVIDER OF SAFE, SECURE, AND RELIABLE OPERATING SYSTEMS

7 missions to Mars

50+ deployed space programs

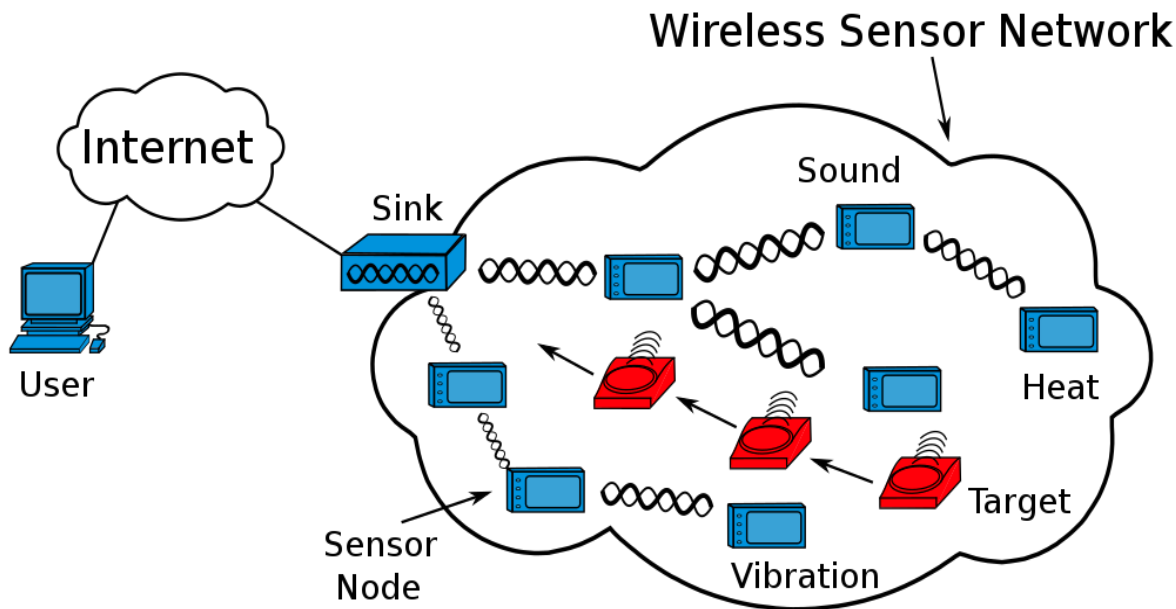
100+ civilian and military aircraft

customers on VxWorks safety platform

360+

TinyOS

- Embedded open-source component-based operating system and platform for low-power wireless devices written in nesC
- Used in wireless sensor networks (WSNs), smartdust, building automation, and smart meters.
- UC in Berkley, Intel and Crossbow Tech in 2000.



Img src: wikipedia.com



Can OS be built in hardware?

Yes!

- 1966 Fairchild Symbol IIR Computer by Fairchild Camera and Lens
- Both Compiler and OS were built in HW.
- Simple set of English Instructions replaced hundreds of machine instructions
- Incorporated 20,000 ICs, handled up to 32 terminals
- Compilation rate: up to 75,000 stmts/min
- Memory control and reclamation, compilation, and job control were controlled by individual processors

1966 Fairchild Symbol IIR Computer Cont'd

- Hardware mechanisms were based on software techniques.
- one-pass translator generated a symbol table in reverse Polish code as in conventional software interpretive languages.
- Compiler operated at disk transfer speeds and was so fast there was no need to keep and store object code - it could be quickly regenerated on-the-fly.
- The hardware-implemented job controller performed conventional operating system functions.
- The memory controller provided a virtual memory for variable-length strings.

1966 Fairchild Symbol IIR Computer



Src: google.com

OS Concepts

- Most complex and technical concepts in CS
- Concepts and abstractions needed for reliable, portable, efficient and secure software are the same as in
 - for cloud computing,
 - secure web browsers,
 - game consoles,
 - graphical user interfaces,
 - media players,
 - databases, etc.

Operating System Roles

- Referee (“keep it more constrained”)
 - Resource allocation among users and applications
 - Isolation of users and applications from each other
 - Communication between users and applications
- Illusionist (“more powerful”)
 - Each application appears to have a machine to itself
 - Physical limitations and details are masked
 - Higher-level objects are provided, such as files
- Glue (“more useful”)
 - Execution environment with common set of services
 - Files written by one app can be read by another

Example: File Systems

- Referee
 - Prevent users from accessing each other's files without permission
- Illusionist
 - Files and directories
 - Files can grow (nearly) arbitrarily large
 - Disk details such as sector size are hidden
- Glue
 - `open()`, `fprintf()`, `fscanf()`

Question

- How should an operating system allocate processing time between competing uses?
 - Give the CPU to the first to arrive?
 - To the one that needs the least resources to complete
 - To the one that needs the most resources?
- Design choices represent trade-offs

OS Challenges

- Reliability
 - Does the system do what it was designed to do?
 - App failure should not bring entire OS down.
 - OS works in hostile environment
- Availability
 - What portion of the time is the system working?
 - Affected by
 - Frequency of failure: Mean Time To Failure (MTTF)
 - Time to repair it: Mean Time to Repair (MTTR)
 - $\text{Availability} = \text{MTTF} / (\text{MTTF} + \text{MTTR})$
 - Improved by increasing MTTF and reducing MTTR
- Ideally, OS should be both reliable and available

OS Challenges

- Security
 - ~ Cannot be compromised by an attacker
- Privacy
 - ~ Aspect of security
 - ~ Data is accessible only to authorized users
 - ~ Can system be perfectly secure?
 - + Hackers, bad admin, back doors, viruses
- Need the following:
 - ~ enforcement mechanism - how do you ensure only permitted actions are allowed
 - ~ Security policy - dictated who can do what

OS Challenges

- Portability
 - ~ Apps see underlying hardware abstraction
 - ~ Changes to HW should be transparent to the user
 - ~ OS itself can be derived from one platform and carried to another
 - iOS derived from Mac OS x code
 - ~ Os should support future apps
 - ~ Linux is portable and has long history!

OS Challenges

- Performance (measured in many ways)
 - Latency/response time
 - How long does an operation take to complete?
 - Throughput
 - How many operations can be done per unit of time?
 - Rate at which system completes tasks
 - Overhead - extra work is done by the OS
 - Efficiency - lack of overhead
 - Fairness - equal performance for different users
 - Predictability - consistent performance over time

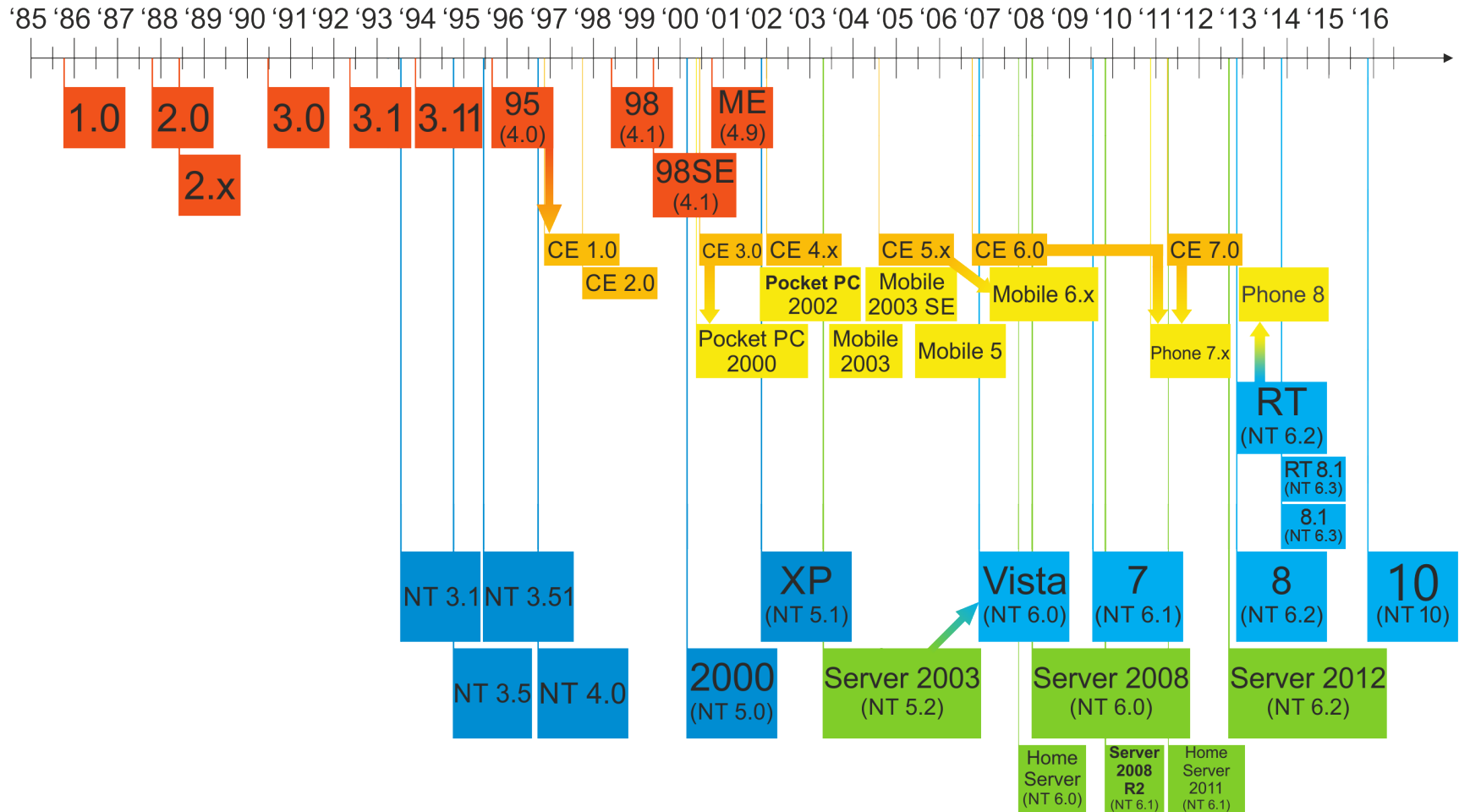
OS Challenges

- Adoption
 - ~ App availability
 - ~ Hw support availability
- Proprietary or open source?
 - Is source code available?
 - Linux is open source
 - Windows or Mac OS are proprietary
- Support vs cost

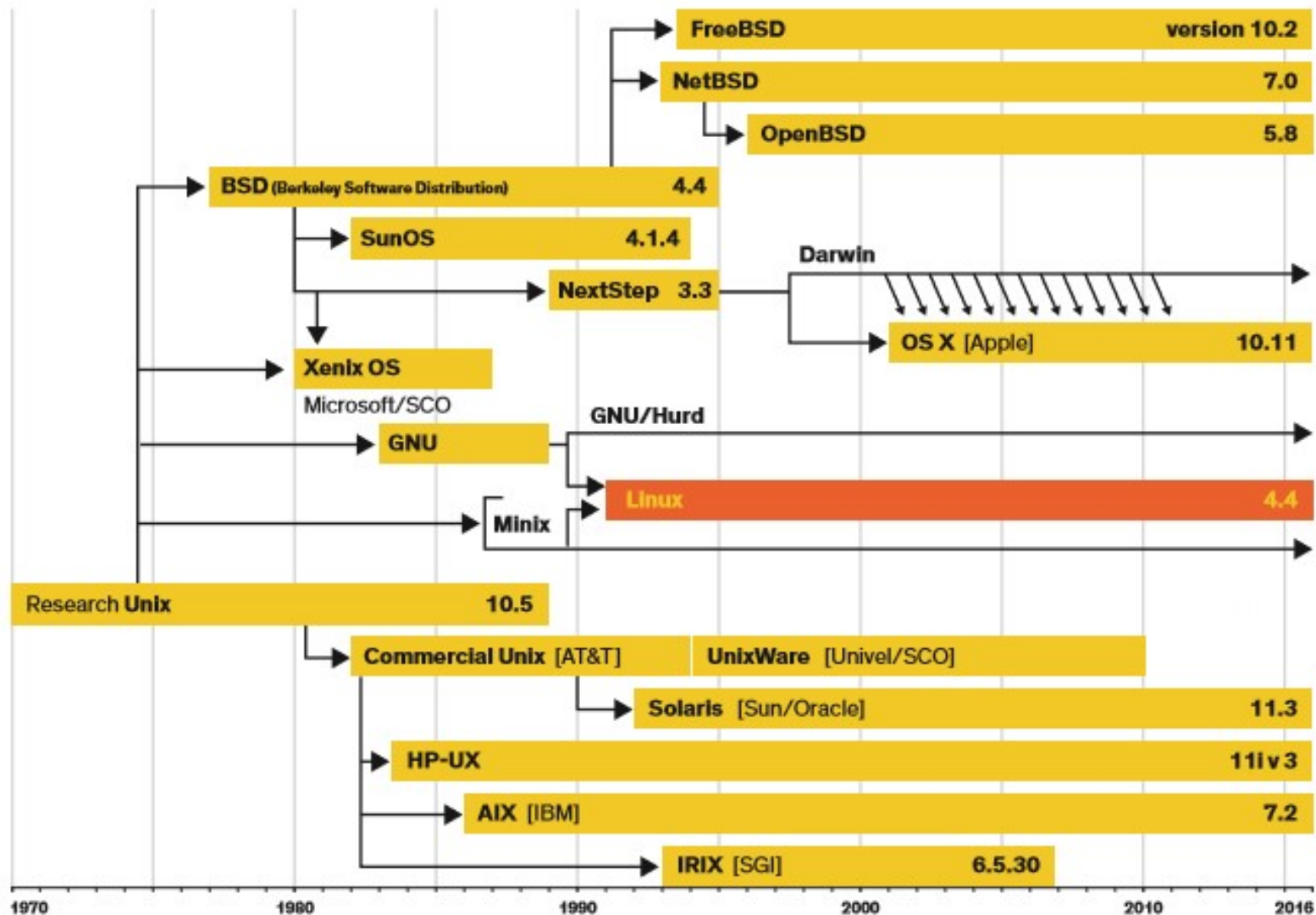
Selected OS Developers

- 1960s
 - Fred Brooks (manager) – IBM OS/360
 - Fernando Corbató (project leader) – Multics
- 1970s
 - Dave Cutler – DEC VMS (and later Windows NT)
 - Dennis Ritchie and Ken Thompson – UNIX
- 1980s
 - Rick Rashid – Mach (Carnegie Mellon Unix-like kernel)
- 1990s
 - Linus Torvalds – Linux kernel

Microsoft Timeline



Unix Evolution



SOURCE: WIKIMEDIA COMMONS

Image source: *Linux at 25* paper, by Chris Tozzi

Computer Performance Over Time

	1981	1997	2014	Factor (2014/1981)
Uniprocessor speed (MIPS)	1	200	2500	2.5K
CPUs per computer	1	1	10+	10+
Processor MIPS/\$	\$100K	\$25	\$0.20	500K
DRAM Capacity (MiB)/\$	0.002	2	1K	500K
Disk Capacity (GiB)/\$	0.003	7	25K	10M
Home Internet	300 bps	256 Kbps	20 Mbps	100K
Machine room network	10 Mbps (shared)	100 Mbps (switched)	10 Gbps (switched)	1000
Ratio of users to computers	100:1	1:1	1:several	100+

KiB, MiB, GiB, etc????

Unit of measure	Bytes
Kilobyte (KB)	$1000^1 = 1,000$
Megabyte (MB)	$1000^2 = 1,000,000$
Gigabyte (GB)	$1000^3 = 1,000,000,000$
Terabyte (TB)	$1000^4 = 1,000,000,000,000$
Petabyte (PB)	$1000^5 = 1,000,000,000,000,000$
Kibibyte (KiB)	$1024^1 = 1,024$
Mebibyte (MiB)	$1024^2 = 1,048,576$
Gibibyte (GiB)	$1024^3 = 1,073,741,824$
Tebibyte (TiB)	$1024^4 = 1,099,511,627,776$
Pebibyte (PiB)	$1024^5 = 1,125,899,906,842,624$

Early Operating Systems: Computers Were Very Expensive

- One application at a time
 - Had complete control of hardware
 - Runtime library was a primitive OS
 - Users would stand in line to use the computer
- Batch systems
 - Keep CPU busy by having a queue of jobs
 - OS would load next job while current one runs
 - Users would submit jobs and wait

Time-Sharing Operating Systems: Computers and People Both Expensive

- Multiple users on computer at same time
 - Multiprogramming: run multiple programs at same time
 - Interactive performance: try to complete everyone's tasks quickly
 - As computers became cheaper, more important to optimize for user time, not computer time

Today's Operating Systems: Computers Are Cheap

- Smartphones
- Embedded systems
- Laptops
- Tablets
- Virtual machines
- Data center servers

Tomorrow's Operating Systems

- Giant-scale data centers
- Increasing numbers of processors per computer
- Increasing numbers of computers per user
- Very large scale storage
- What else will the future bring?