CS 202
Advanced Operating Systems

Winter 26

Lecture 2: Historical Perspective

Instructor: Chengyu Song

Protection

- Operating systems implement some kind of protection system
  - Who can access a file
  - How they can access it
- More generally…
  - Objects are “what”, subjects are “who”, actions are “how”
- A protection system dictates whether a given action performed by a given subject on a given object should be allowed
  - You can read and/or write your files, but others cannot
  - You can read “/etc/motd”, but you cannot write to it

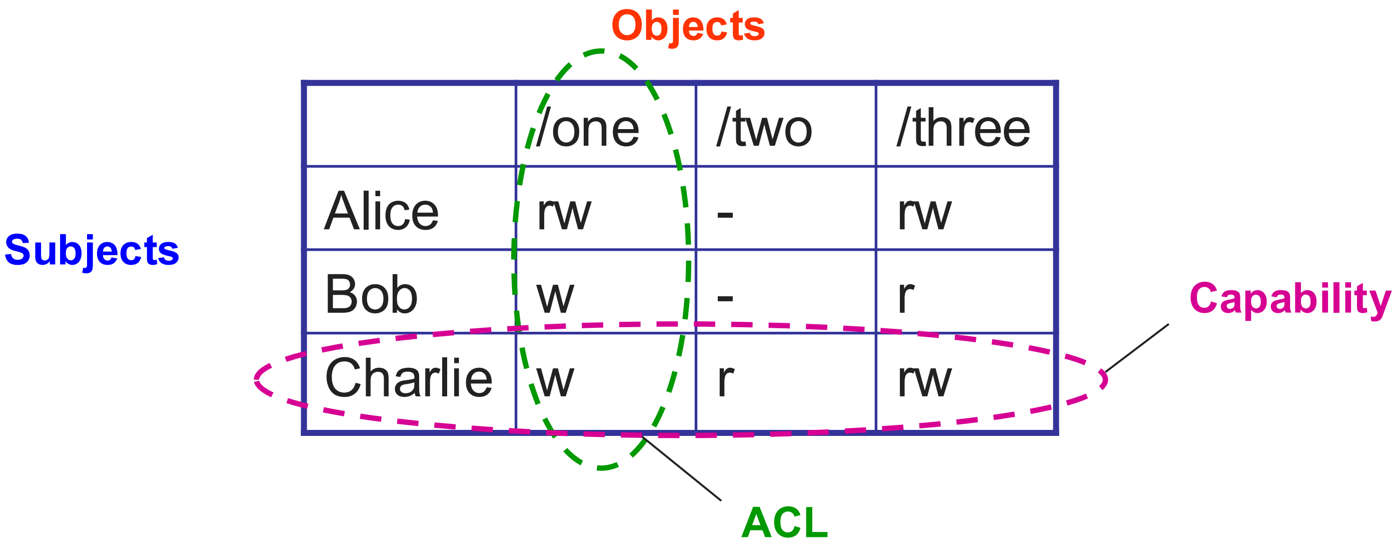
Representing Protection

Access Control Lists (ACL)

- For each object, maintain a list of subjects and their permitted actions

Capabilities

- For each subject, maintain a list of objects and their permitted actions



	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

ACLs and Capabilities

- The approaches differ only in how table is represented
  - What approach does Unix use?
  - What approach does Mobile permission system use?
- Capabilities are easier to transfer
  - They are like keys, can handoff, does not depend on subject
- In practice, ACLs are easier to manage
  - Object-centric, easy to grant, revoke
  - To revoke capabilities, have to keep track of all subjects that have the capability – a challenging problem
- ACLs have a problem when objects are heavily shared
  - The ACLs become very large

HYDRA

- **Protection** and **Extensibility**
- Philosophy: The Separation of **Mechanism** and **Policy**
- The Kernel as a Nucleus
  - HYDRA was built as a "kernel" or "nucleus" providing only a set of facilities with "universal applicability" and "absolute reliability”
- Architectural Flexibility
  - The designers argued that previous systems failed to experiment effectively because they didn't separate mechanisms (what the kernel provides) from policies (how those facilities are used)
- Meta-Environment
  - HYDRA’s goal was to provide a "meta-environment" that could host multiple, diverse operating systems simultaneously, allowing for the exploration of different user-visible environments

HYDRA: Object Model

- Resources as **Objects**
  - HYDRA introduced a generalized notion of "resource" (physical or virtual) called an "object”
- Structure of an Object
  - Every object consists of a unique name, a type, and a representation that includes both a data part and a **capability** part
- Extensible Types
  - Users can create entirely new object types by defining a new "representative" object. This allows the system to support anything from new file systems to specialized synchronization primitives

HYDRA: Capabilities

- Unforgeable References:
  - Objects are accessed only via capabilities, which are references combined with **a list of access rights**
- Kernel-Only Manipulation
  - Capabilities are maintained only by the kernel; they cannot be modified or "forged" by user programs
- Uniform Protection
  - Unlike systems that only protect specific entities like files, HYDRA provides a protection facility for all entities in the system, including memory and execution domains
- The CALL Mechanism
  - When a procedure is invoked, the kernel creates a Local Name Space (LNS), which defines the totality of capabilities available for that specific invocation

Unix appears

- Ken Thompson, who worked on MULTICS, wanted to use an old PDP-7 laying around in Bell labs
- He and Dennis Richie built a system designed by programmers for programmers
- Originally in assembly. Rewritten in C
  - In their paper describing UNIX, they defend this decision!
  - However, this is a new and important advance: portable operating systems!
- Shared code with everyone (particularly universities)

Unix (cont'd)

- Berkeley added support for virtual memory for the VAX
    - Unix BSD
- DARPA selected Unix as its networking platform in ARPAnet
- Unix became commercial
    - …which eventually lead Linus Torvald to develop Linux

UNIX: OS for developers

- Simplicity and Elegance
  - The primary achievement of UNIX was demonstrating that a powerful, interactive system could be built with minimal resources (hardware under $40,000 and less than two man-years of effort)
- The "File as a Byte Stream" Model
  - UNIX abandoned the idea of the OS imposing structure on data. Files are simply strings of characters (bytes), and any structure is controlled by user programs, not the system
- Unified I/O and "Special Files”
  - UNIX treats almost everything as a file. Special files represent I/O devices (e.g., paper tape, disks), allowing them to be read and written using the same syntax and protection mechanisms as ordinary files
- Hierarchical File System and Removal Mounts
  - UNIX introduced directories (the tree) and mount/unmount (of tree nodes)

UNIX: OS for developers

- The Shell as a User Program
  - A major innovation was making the command line interpreter (the Shell) an ordinary, swappable user program. This allows users to replace the Shell easily and makes features like I/O redirection and background processes (using the & operator) trivial to implement
- Many built-in utility programs for programming
  - assembler, text, linking loader, symbolic debugger, C compiler, BASIC interpreter, text formatting program, Fortran compiler, Snobol interpreter, top-down compiler-compiler (TMG), bottom-up compiler-compiler (YACC), form letter generator, macro processor, permuted index program
- Self-Supporting System
  - From its early stages, UNIX was used to maintain itself; all UNIX software and documentation were generated and formatted using the system's own tools

UNIX: Engineering

- No Predefined Objectives
  - Paradoxically, UNIX succeeded because it was not designed to meet any predefined objectives. It began as a personal effort by programmers (Thompson and Ritchie) to create a "hospitable environment" for their own work, rather than a corporate project intended to satisfy external requirements
- Designed by Programmers for Programmers
  - Because the designers were the primary users, they focused on programming convenience. They prioritized an interactive environment, which they found much more productive than the "batch" systems prevalent at the time
- Self-Supporting and Self-Correcting
  - Almost from the start, UNIX was able to maintain itself. The designers were forced to use the system they were building, which made them immediately aware of any "functional and superficial deficiencies" and motivated them to fix them quickly

UNIX: Engineering

- Simplicity and Elegance
  - The system demonstrated that a powerful OS did not need to be expensive or require massive human effort; it was built with less than two man-years of software effort. Its core strength lies in its simplicity, providing a unified interface where "everything is a file" and complex tasks are handled by a swappable user-mode program called the Shell
- Severe Size Constraints
  - Unlike "larger systems," UNIX faced strict hardware limitations. These constraints encouraged economy and elegance of design, forcing the creators to develop a small yet powerful set of "fertile ideas"

Unix Access Control Model

- Unix uses ACLs
  - Rights are associated with objects
    - Essentially files, everything is a file in Unix
- Three sets of rights
  - Owner, Group, Others
  - Use groups to limit the length of ACL
    - Files (objects) can only belong to one group
    - Process (subjects) can belong to multiple group

```
$ ls -l fs.pdf
-rw-r--r--  1 csong  csprofs  fs.pdf
$ id
uid=123(csong) gid=12(csprofs) groups=12(csprofs),34(csoffice)
```

Unix Access Rights

- Files
  - Literal: read, write, execute (rwx)
- Directories
  - Read: read/list file names, but not metadata
  - Write: create, rename, and delete files
  - Execute: read file metadata and change current directory to it
- Who can set the access rights and group?
  - Owner!

Discretionary Access Control

- Both ACLs and capabilities are DAC
- DAC has a special user: root/Administrator
  - root is not subject to access rights
    - They can read/write/execute any file
  - root can change the owner/group of any file
  - root can change the access rights of any file
  - Why?
- This is why attackers like root

Users and Processes

- Although ACLs use users as subjects, the OS actually uses processes as subjects
  - Processes act on the behalf of users, like a **trusted** proxy
- Then how to decide the identity (user/group) of a process?
  - R1: by default, a newly created process inherits its parent process' identity, unless R2 or R3
  - R2: a process can change its identity through setuid()/setgid() system calls
  - R3: exec a setuid/setgid program will change the identity of the process to the owner/group of the executable file

Boot and login

- Process tree in Unix
  - The first process is init
    - Created with root privilege
    - Spawns other daemons (services), including the login daemon
  - The login daemon
    - Authenticates the user
    - Forks itself, changes its identity to the authenticated user
    - Starts a shell (desktop GUI)

Phase 3: 1980s

- Computers are cheap, people expensive
  - Put a computer in each terminal
  - CP/M from DEC first personal computer OS (for 8080/85) processors
  - IBM needed software for their PCs, but CP/M was behind schedule
  - Approached Bill Gates to see if he can build one
  - Gates approached Seattle computer products, bought 86-DOS and created MS-DOS
  - Goal: finish quickly and run existing CP/M software
  - OS becomes subroutine library and command executive

Phase 4: 1990s to now?

- Its all about connectivity
- Enables parallelism but performance is not goal
- Goal is communication/sharing/power consumption/...
  - Requires high speed communication
  - We want to share data not hardware
- Networked applications drive everything
  - Web, email, messaging, social networks, …
  - Chromebook

New problems

- Large scale
  - Google file system, map-reduce, …
- Parallelism on the desktop (multicores)
- Heterogeneous systems, IoT
  - GPU, FPGA, …
  - Real-time; energy efficiency
- Security and Privacy

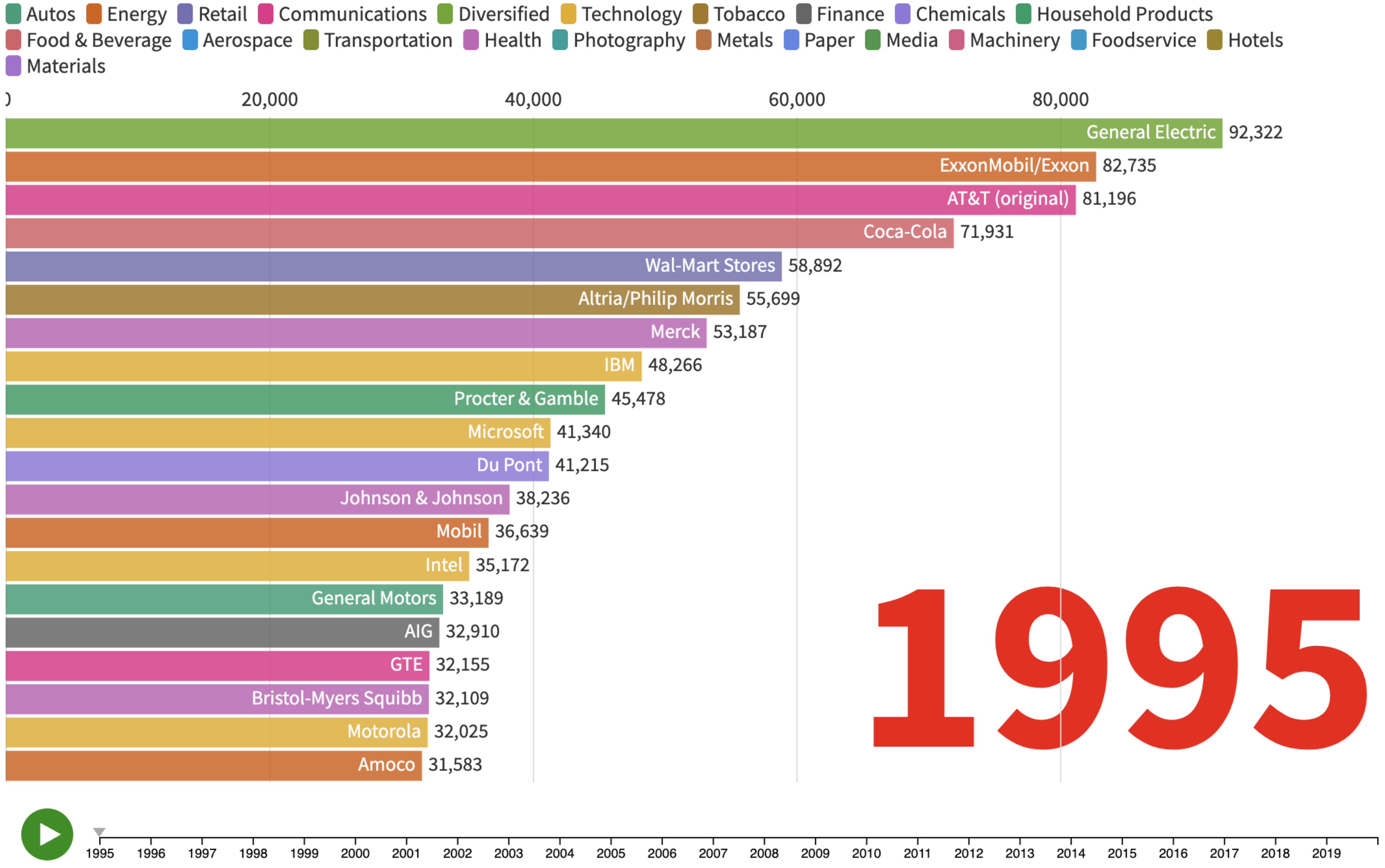
Phase 5

- New generation?

- Computing evolving beyond networked systems
  - Cloud computing, edge computing, IoT, wearable devices, drones, cyber-physical systems, autonomous cars, computing everywhere
  - But what is it?
  - … and what problems will it bring?

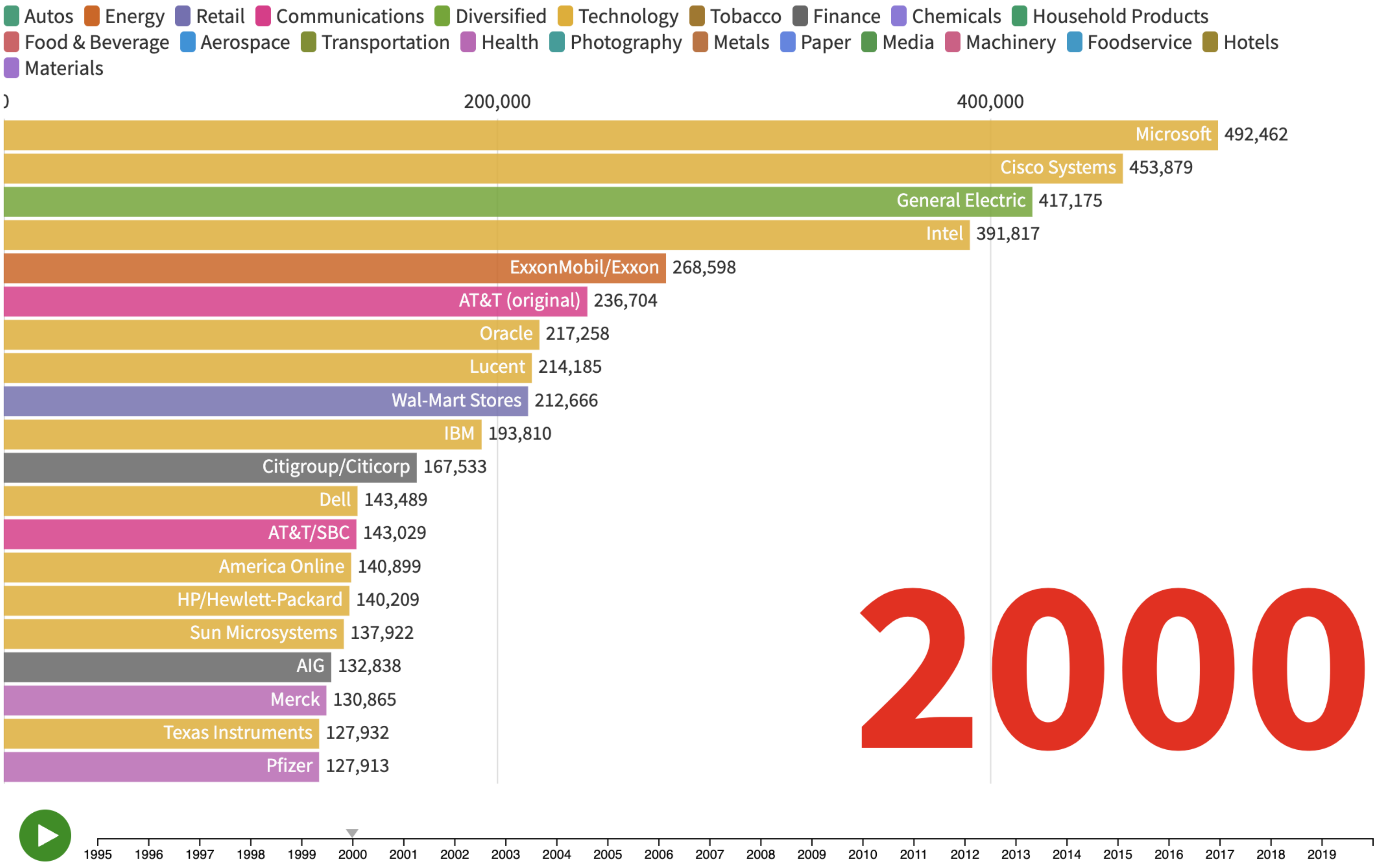
20 MOST VALUABLE FORTUNE 500 COMPANIES 1995-2020

Market Capitalization in March of each year in Millions of Dollars



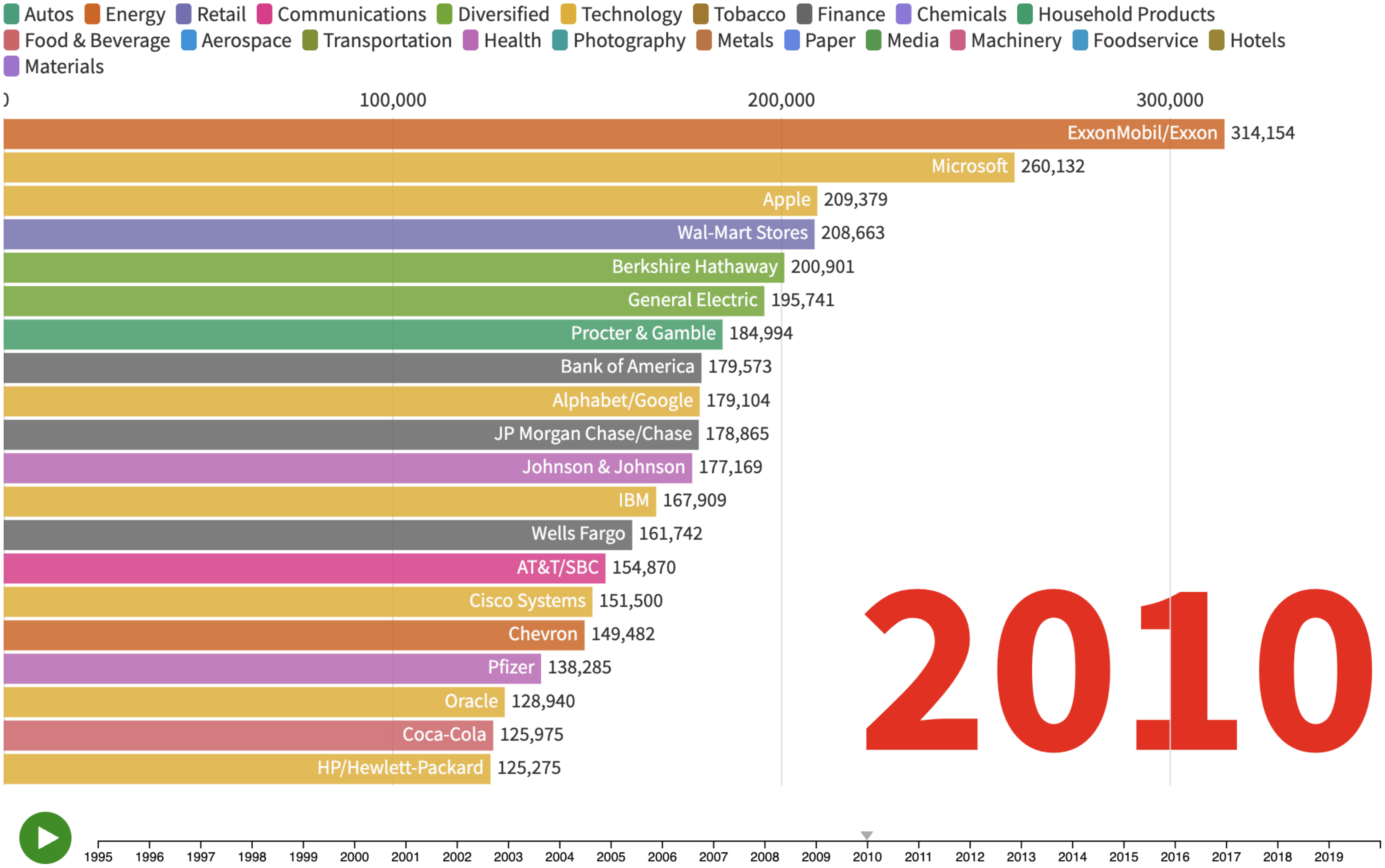
20 MOST VALUABLE FORTUNE 500 COMPANIES 1995-2020

Market Capitalization in March of each year in Millions of Dollars



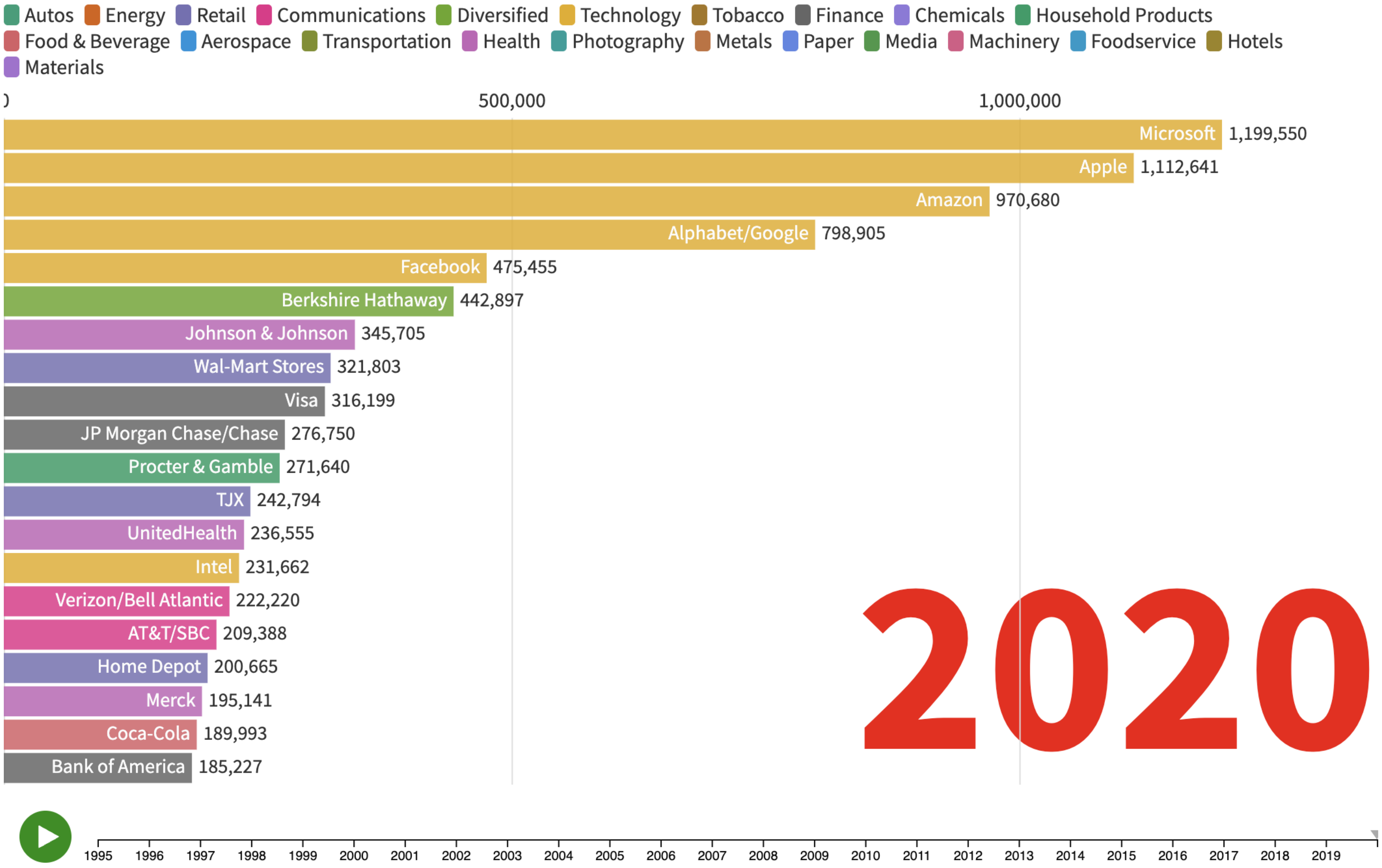
20 MOST VALUABLE FORTUNE 500 COMPANIES 1995-2020

Market Capitalization in March of each year in Millions of Dollars



20 MOST VALUABLE FORTUNE 500 COMPANIES 1995-2020

Market Capitalization in March of each year in Millions of Dollars


Autos
Energy
Retail
Communications
Diversified
Technology
Tobacco
Finance
Chemicals
Household Products
Food & Beverage
Aerospace
Transportation
Health
Photography
Metals
Paper
Media
Machinery
Foodservice
Hotels
Materials
Microsoft 1,199,550
Apple 1,112,641
Amazon 970,680
Alphabet/Google 798,905
Facebook 475,455
Berkshire Hathaway 442,897
Johnson & Johnson 345,705
Wal-Mart Stores 321,803
Visa 316,199
JP Morgan Chase/Chase 276,750
Procter & Gamble 271,640
TJX 242,794
UnitedHealth 236,555
Intel 231,662
Verizon/Bell Atlantic 222,220
AT&T/SBC 209,388
Home Depot 200,665
Merck 195,141
Coca-Cola 189,993
Bank of America 185,227