香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Operating Systems
Lecture 1

Introduction

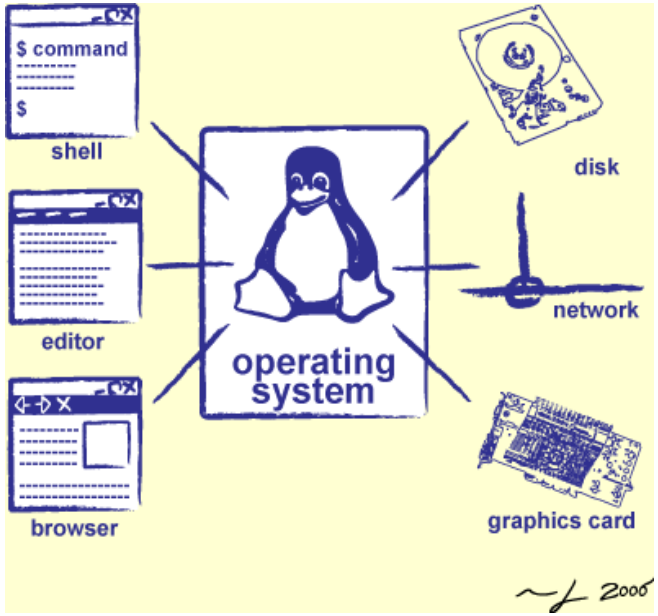
Prof. Yunming XIAO

School of Data Science

Acknowledgments: Ion Stoica and Matei Zaharia, Berkeley CS 162

Goals today

- Course logistics
- Why should you care?
- Why is it hard?
- What is an Operating System?



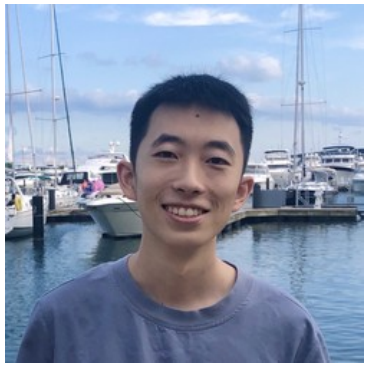
About me

- Background
  - Education: B.Eng. BUPT (2019) & Ph.D. Northwestern U (2024)
  - Work Experience
    - Intern: ByteDance, Bell Labs, HPE Labs, Google
    - Research Fellow (2024-2025), University of Michigan—Ann Arbor
    - Assistant Professor (2025.7-*Present*), SDS, CUHK-Shenzhen
  - Research: computer systems, networks, and security

- Contact
  - Email: yunmingxiao@cuhk.edu.cn
  - Office: Zhi Xin Building 403a
  - Office Hour: 2pm-3pm Tuesday (except holidays)

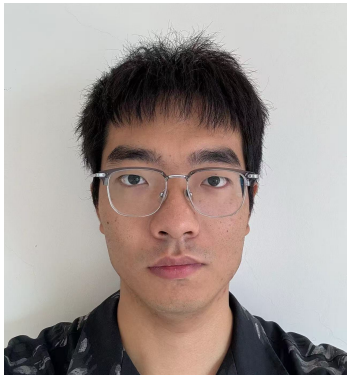
Course stuff

- Instructor: XIAO, Yunming



- Teaching Assistants:

- Li, Cheng



ZHENG, Yihao

- USTF:

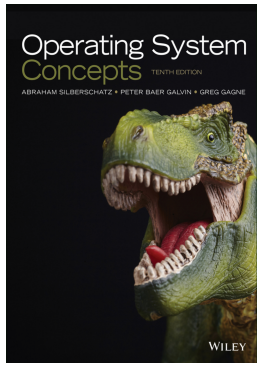
ZHENG Jiyan

Tutorials

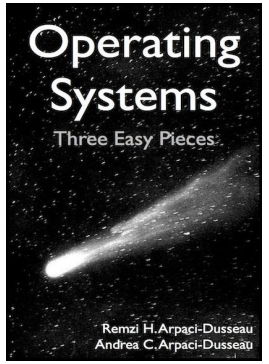
- Tutorials will be provided by TAs and USTF
- Very helpful for you to finish the projects!
- You should sign up for the tutorial session
- **Time: 6pm – 8pm, every Wednesday**

Course materials

- Infrastructures:
  - Website: https://yunmingxiao.github.io/courses/csc3150-26spring/syllabus.html
  - Blackboard: https://bb.cuhk.edu.cn/



- Recommended textbooks:
  - Operating System Concepts, by A. Silberschatz, P. Galvin, and G. Gangne (Available online)
  - Operating Systems: Three Easy Pieces, by Remzi and Andrea Arpaci-Dusseau (Available online)
  - The RISC-V Reader: An Open Architecture Atlas, by D. Patterson and A. Waterman, Strawberry Canyon (Available online)



Grading

- Participation: 5%
  - You need to attend at least 2 out of 3 roll calls (dates will be announced beforehand)

- Assignments/projects: 70%
  - 4 assignments: 25% each
  - Design (doc) + implementation (code)
  - Projects are done individually
  - We have **much less** workload:
    - Berkeley CS162: **5** labs + **7** programming assignments
    - MIT 6.1810: **8** labs + **23** homeworks

- Final exam: 25%

Collaboration policy

Explaining a concept to other students*

Discussing algorithms/testing strategies with other students

Searching online for generic algorithms (e.g., hash table)

Sharing code or test cases with other students

Copying or reading another student's code or test cases

Copying or reading online code or test cases

Uploading your solutions online (during AND after the course)

We compare all homework submissions against each other and online solutions and will take actions against offenders

* AI is considered as a student who has taken CSC3150 before

Assignments

Assignment Number	(Intended) Start Date	(Intended) Due Date
#1	2.3	3.9
#2	3.10	3.30
#3	3.31	4.20
#4	4.21	5.10

Course workloads

- Lectures
  - Core concepts and principles of OS, which form the foundation of computer systems in general (based on Berkeley CS 162)
  - Recent developments and frontiers in computer systems
    - Big data systems
    - Machine learning systems
    - Cloud computing
    - We will read interesting research papers in computer systems

- Programming assignments: VERY CHALLENGING
  - Design and implement your own operating systems
  - xv6 from MIT (used by OS courses at MIT)
  - But still, (much) reduced workloads

Preparing yourself for this class

- The assignments will require you to be very comfortable with programming and debugging C
  - Pointers (including function pointers, void*)
  - Memory management (malloc, free, stack vs heap)

- You will be working on a larger, more sophisticated code base than anything you've likely seen in other courses!

Goals

- Prepare students for advanced study and research in computer systems, by providing the necessary foundation and context **(learn core CS concepts)**

- Enable students to understand the internal core logic of the operating system in depth and have a chance to design and implement their own operating systems **(foundation for advanced programming)**

- Empower students to write and debug large programs, design and implement useful abstractions and especially practice their ability to write and debug multi- threading programs **(Get hands on industrial-scale codebase)**

Syllabus

- OS concepts: how to navigate as a system programmer
  - Process, I/O, networks
- Concurrency
  - Threads, scheduling, locks, deadlock, scalability, fairness
- Address Space
  - Virtual memory, address translation, protection, sharing
- File systems
  - I/O devices, file objects, storage, naming, caching, performance, paging
- Distributed systems
- Reliability and security
- Cloud infrastructure
- Modern computer system research

Lecture goal

Interactive!!!

Questions?

Why should you care?

- Every device, from your smartwatch, your smart light bulb, to your mobile phone and laptop runs an operating system
- Every program you will ever write will run on an operating system
- Its performance and execution behavior will depend on the operating system

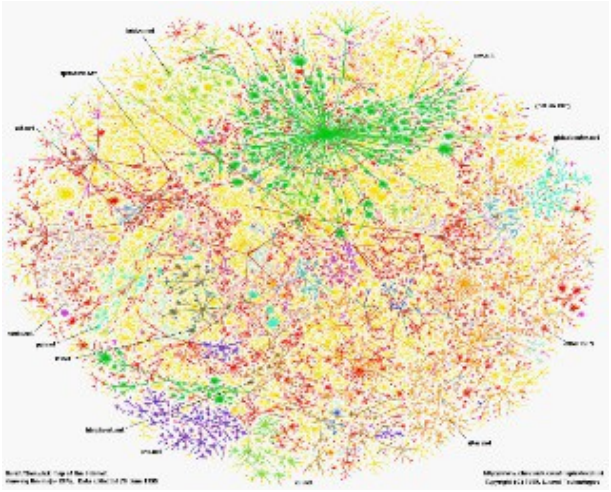
Why is designing an OS hard?

- Think: what do they have in common?

Why is designing an OS hard?

- OS is everywhere

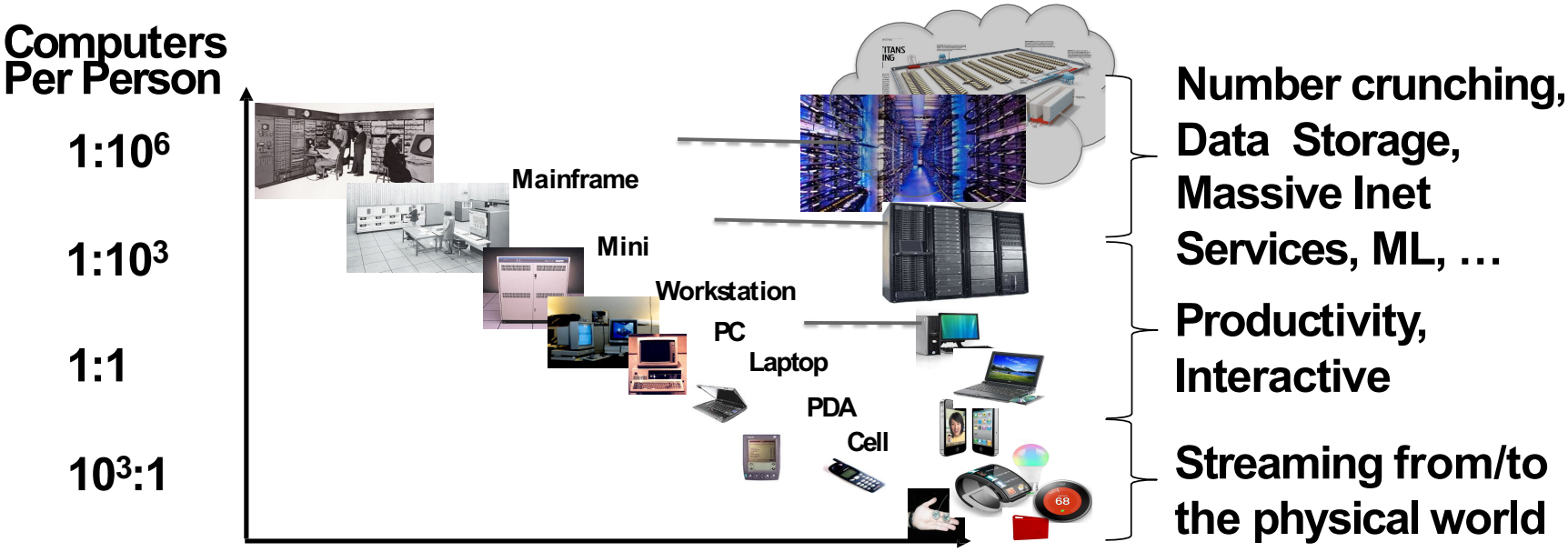
All have an operating system



Communication over Internet

Interface across huge diversity of devices

Bell's law



One new device class every 10 years

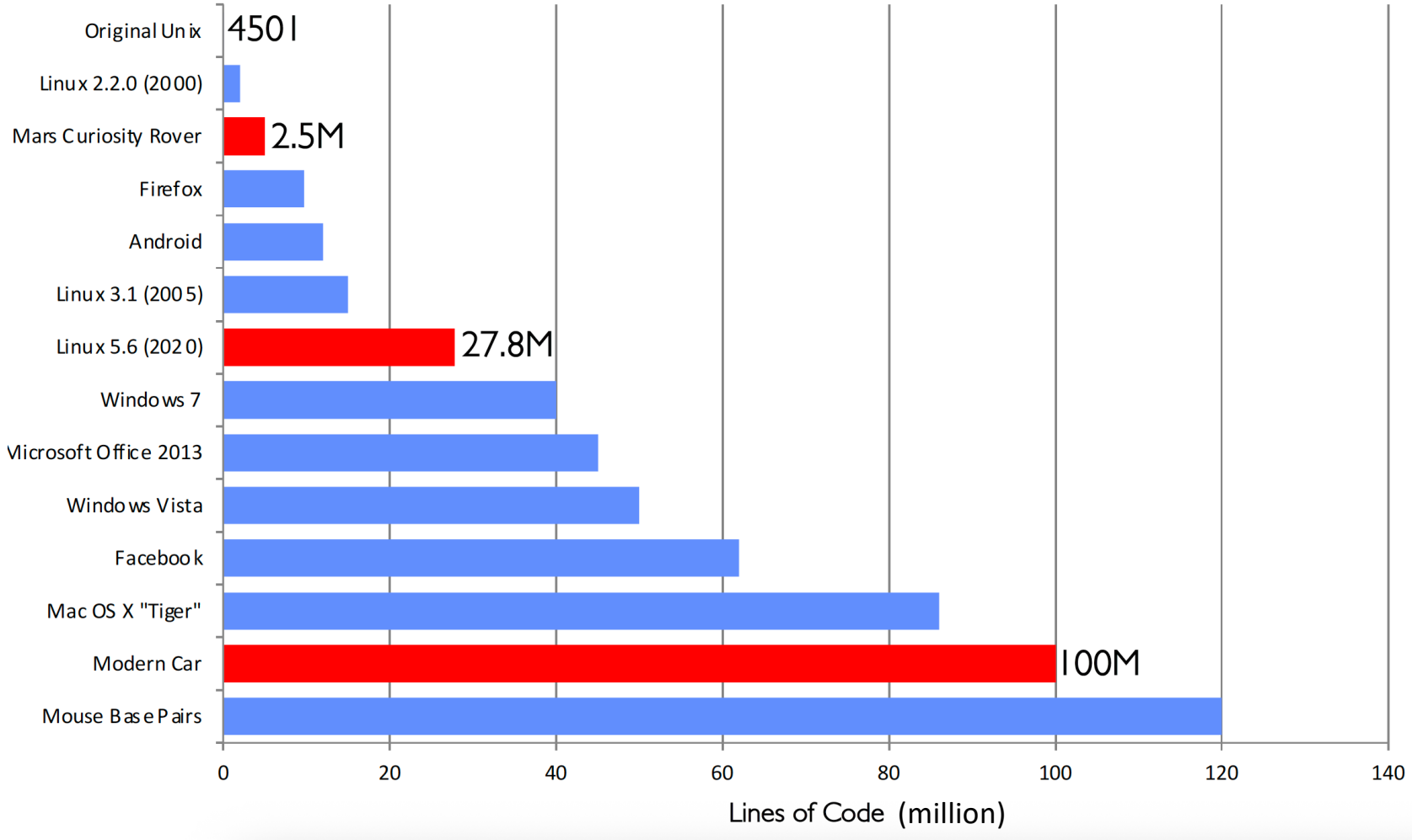
Across many timescales

- Jeff Dean's numbers every should know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

8 orders of magnitude!

With increased complexity



Why so much complexity?

- Hardware is becoming smarter!
- Better reliability and security
- Better performance (more efficient code, more parallel code)
- Better energy efficiency
- Legacy

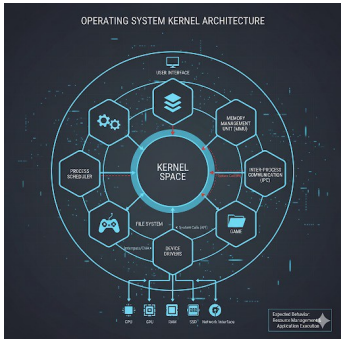
What is an Operating System?

Operating

Manages multiple tasks and users

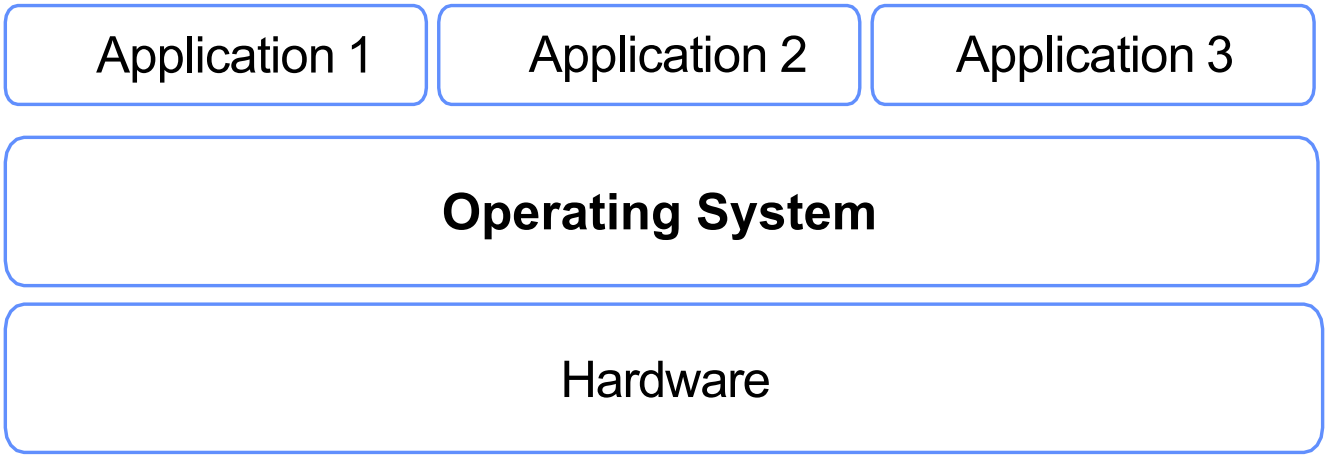
System

A set of interconnected components with an expected behavior observed at the interface with its environment



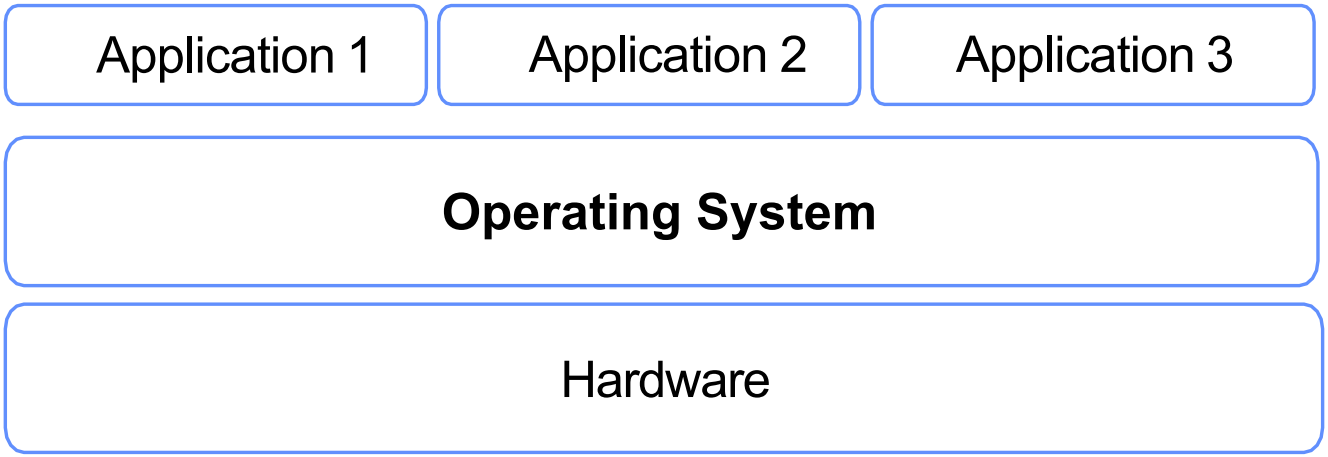
Operating System (v1)

- An operating system is the layer of software that interfaces (many) applications running on a machine with (diverse) hardware resources of that machine



Operating System (v2)

- An operating system implements a virtual machine for the application whose interface is more convenient than the raw hardware interface
- (convenient = portability, reliability, security)



How I view the OS

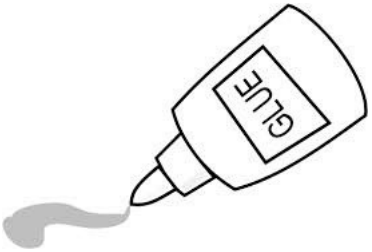
Three main hats

Referee

Manage protection, isolation, and sharing of resources

Illusionist

Provide clean, easy-to-use abstractions of physical resources

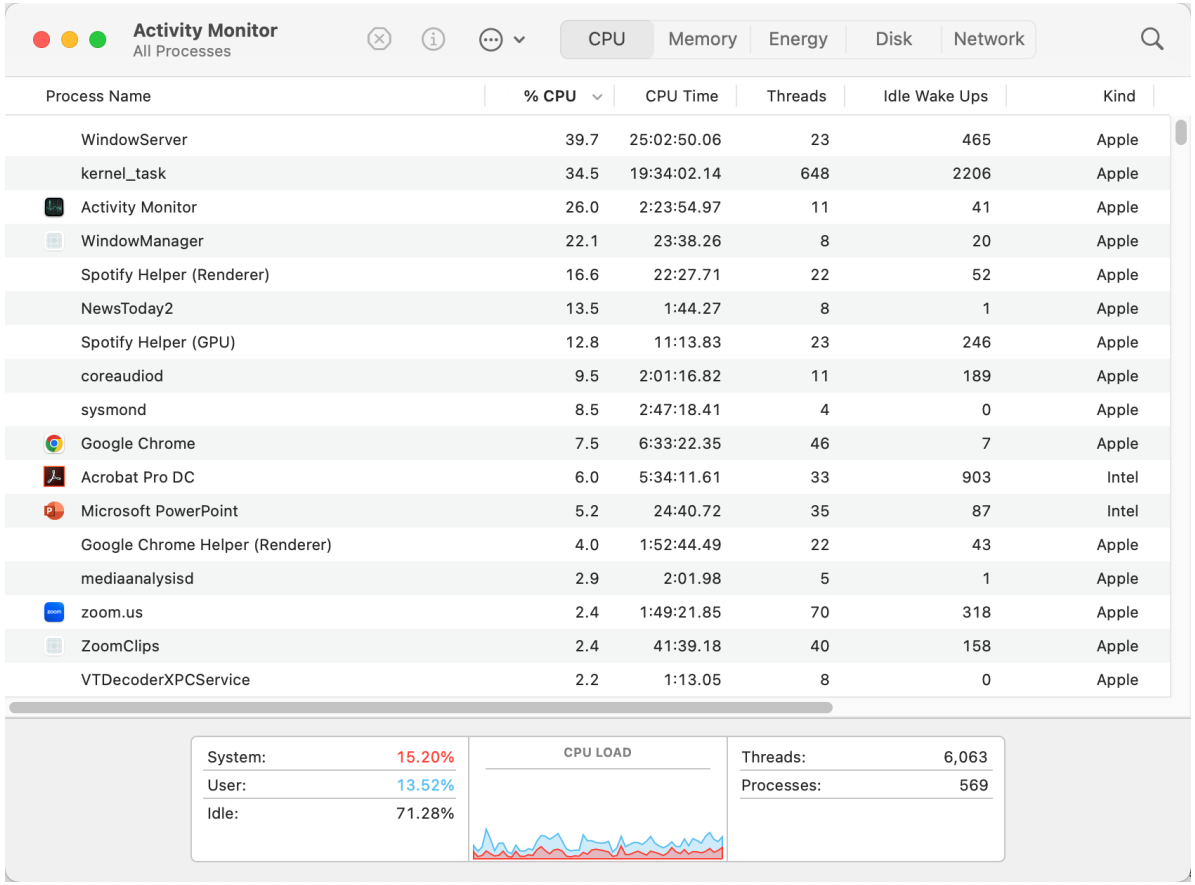


Glue

Provides a set of common services

OS as a referee

- Allow multiple (untrusted) applications to run concurrently



OS as a referee

- Allow multiple (untrusted) applications to run concurrently

Fault Isolation	**Resource Sharing**	**Communication**
Isolate programs from each other	How to choose which task to run next?	How can OS support communication to share results?
Isolate OS from other programs	How to split physical resources?	
Process	Scheduling	Pipes/Sockets
Dual Mode Execution		

What does this program do?

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>

int main(int argc, char *argv[]){
  char *str = argv[1];
  while (1) {
    printf("%s\n", str);
  }
  return 0;
}
```

```
xiao@laptop> gcc -o cpu cpu.c -Wall
xiao@laptop> ./cpu A
```

```
A
A
A
A
…
```

```
xiao@laptop> ./cpu A & ./cpu B & ./cpu C
```

a)

```
A
A
A
A
…
```

b)

```
A
B
C
A
B
C
…
```

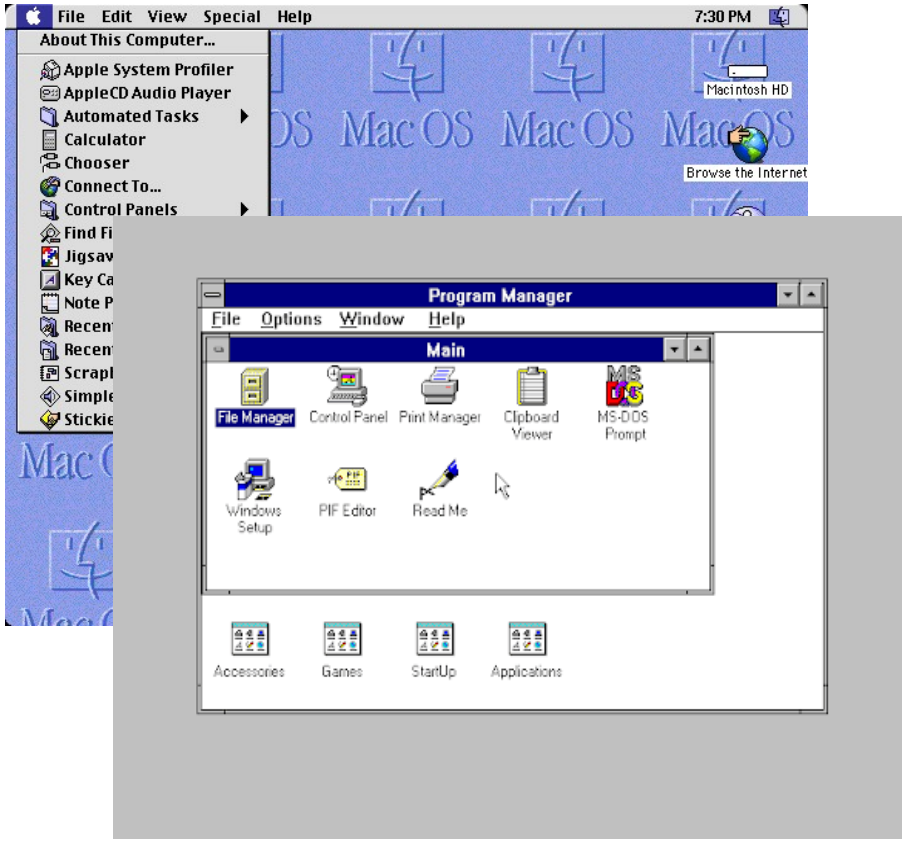
c)

```
A
B
B
A
C
B
C
…
```

```
xiao@laptop> ./cpu & ; ./cpu B
```

```
Segmentation Fault
B
…
```

Refereeing is hard!



Up to MacOS 9x and Windows 3.1

OS cannot force program to give up control!

```
xiao@very-old-laptop>
./cpu A & ./cpu B & ./cpu C
A
A
A
A
A
A
A
...
```

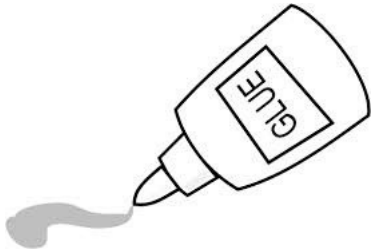
Three main hats

Referee

Manage protection, isolation, and sharing of resources

Illusionist

Provide clean, easy-to-use abstractions of physical resources



Glue

Provides a set of common services

OS as an illusionist

- Mask the restrictions inherent in computer hardware through virtualization

All alone	**All powerful**	**All expressive**
Provide illusion that application has exclusive use of resources	Provide illusion that hardware resources are infinite	Provide illusion of hardware capabilities that are not physically present

What does this program do?

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]){
  int *p = malloc(sizeof(int));
  printf("(%d) p: %p\n", getpid(), p);
  *p = 0;
  while (1) {
    *p = *p + 1;
    printf("(%d) p: %d\n", getpid(), *p);
  }
  return 0;
}
```

```
xiao@laptop> gcc -o memory memory.c -Wall
xiao@laptop> ./memory
```

```
(120)   p:  0x200000
(120)   p:  1
(120)   p:  2
(120)   p:  3
(120)   p:  4
```

```
xiao@laptop> ./memory & ./memory
(120) p: 0x200000
(254) p: 0x200000
```

a)
```
(120)  p:  1
(254)  p:  2
(120)  p:  3
(254)  p:  4
(120)  p:  5
(254)  p:  6
```

b)
```
(120)  p:  1
(254)  p:  1
(120)  p:  2
(254)  p:  2
(120)  p:  3
(254)  p:  3
```

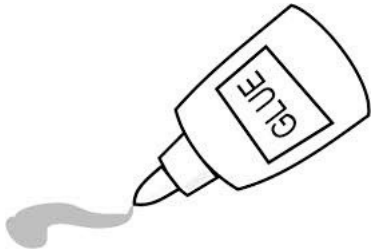
Three main hats

Referee

Manage protection, isolation, and sharing of resources

Illusionist

Provide clean, easy-to-use abstractions of physical resources



Glue

Provides a set of common services

OS as a glue

- Provide set of common, standard services to applications to simplify and regularize their design

Make sharing easier

Simpler if all assume same basic primitives

Maximize reuse

Avoid re-implementing functionality from scratch.

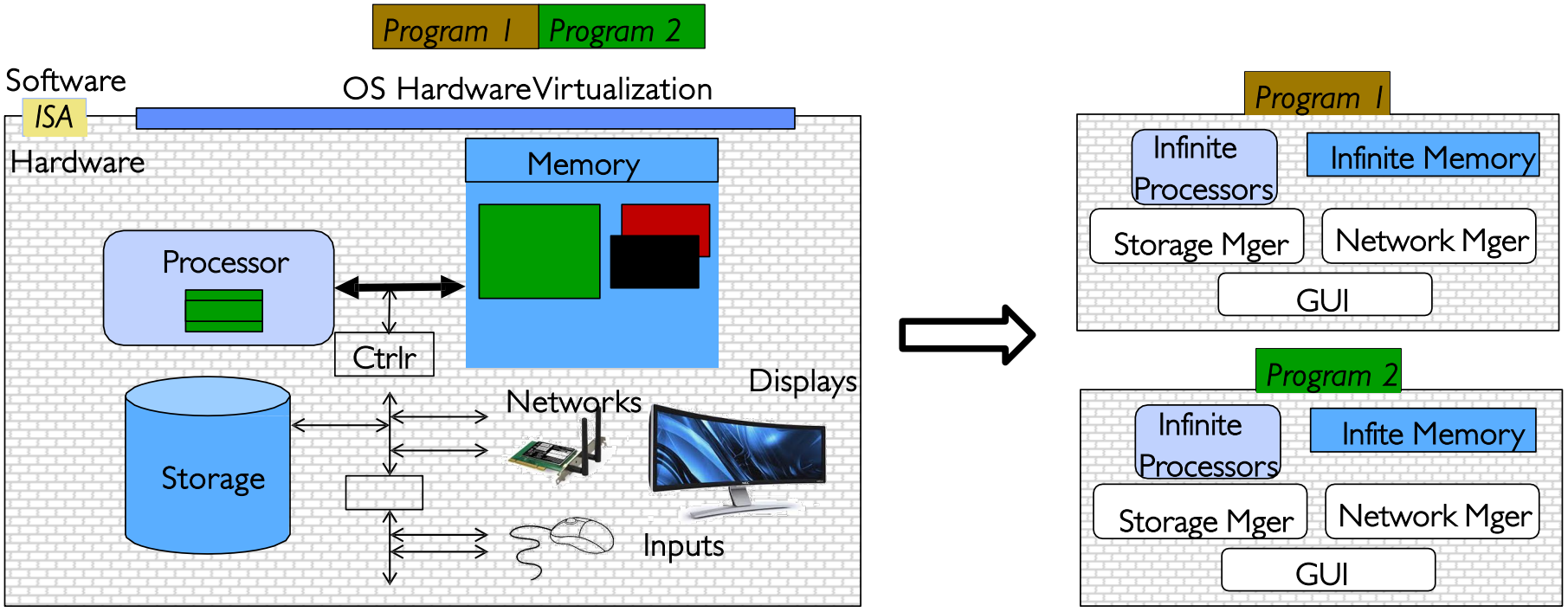
Evolve components independently

File system, user interface, network, etc

Putting it all together

Referee + illusionist + Glue

=> Easy to use virtual machine

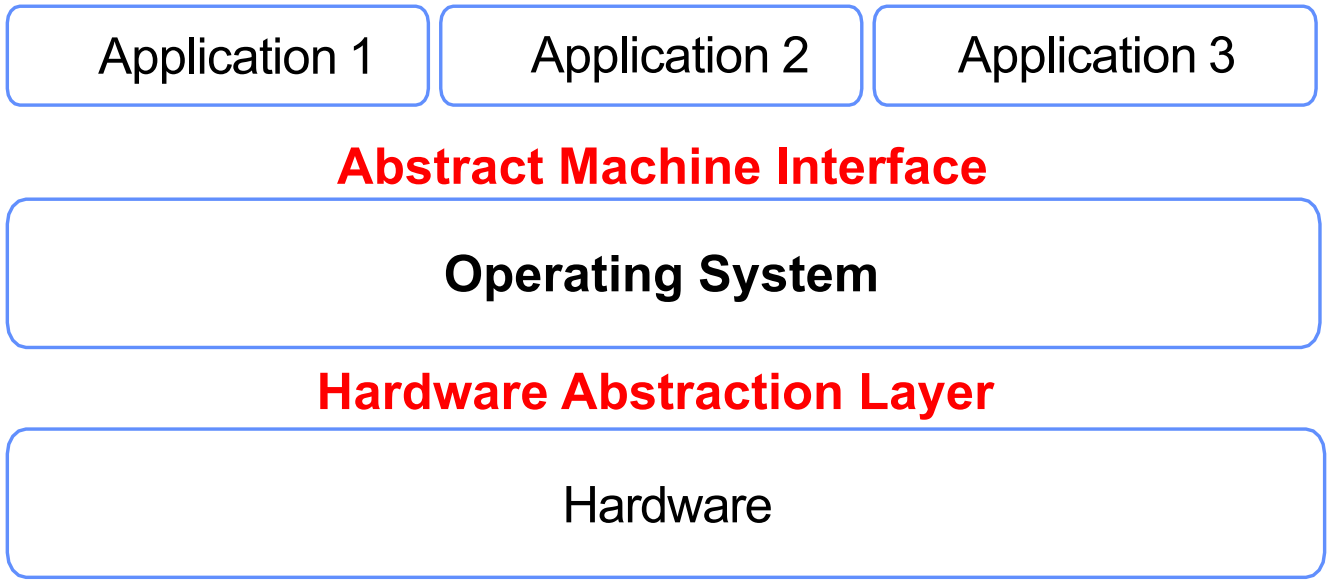


Evaluation criteria

- OS must implement the abstraction efficiently, with low overhead, and equitably
- Overhead: Added resource cost of implementing an abstraction
  Fairness: How equitable are resources distributed across applications?
- Portability: How much you need to change the application when hardware changes?
- Reliability: Does the system do what is supposed to do?
- Security: Maintain expected functionality in face of attacks
  Performance: meet user's and system administrator's expectations in terms of response time, throughput, and predictability

Evaluation criteria: portability

- A portable abstraction **does not** change as the hardware changes
- Can't rewrite application (or OS!) every time
- Must plan for hardware that does not exist yet!



Evaluation criteria: reliability

- System does what it is supposed to do
- OS failures are catastrophic!

Availability: mean time to failure + mean time to repair

Evaluation criteria: security

- Minimize vulnerability to attack
- Integrity: Computer's operation cannot be compromised by a malicious attacker
- Privacy: data stored on computer accessible to authorized users

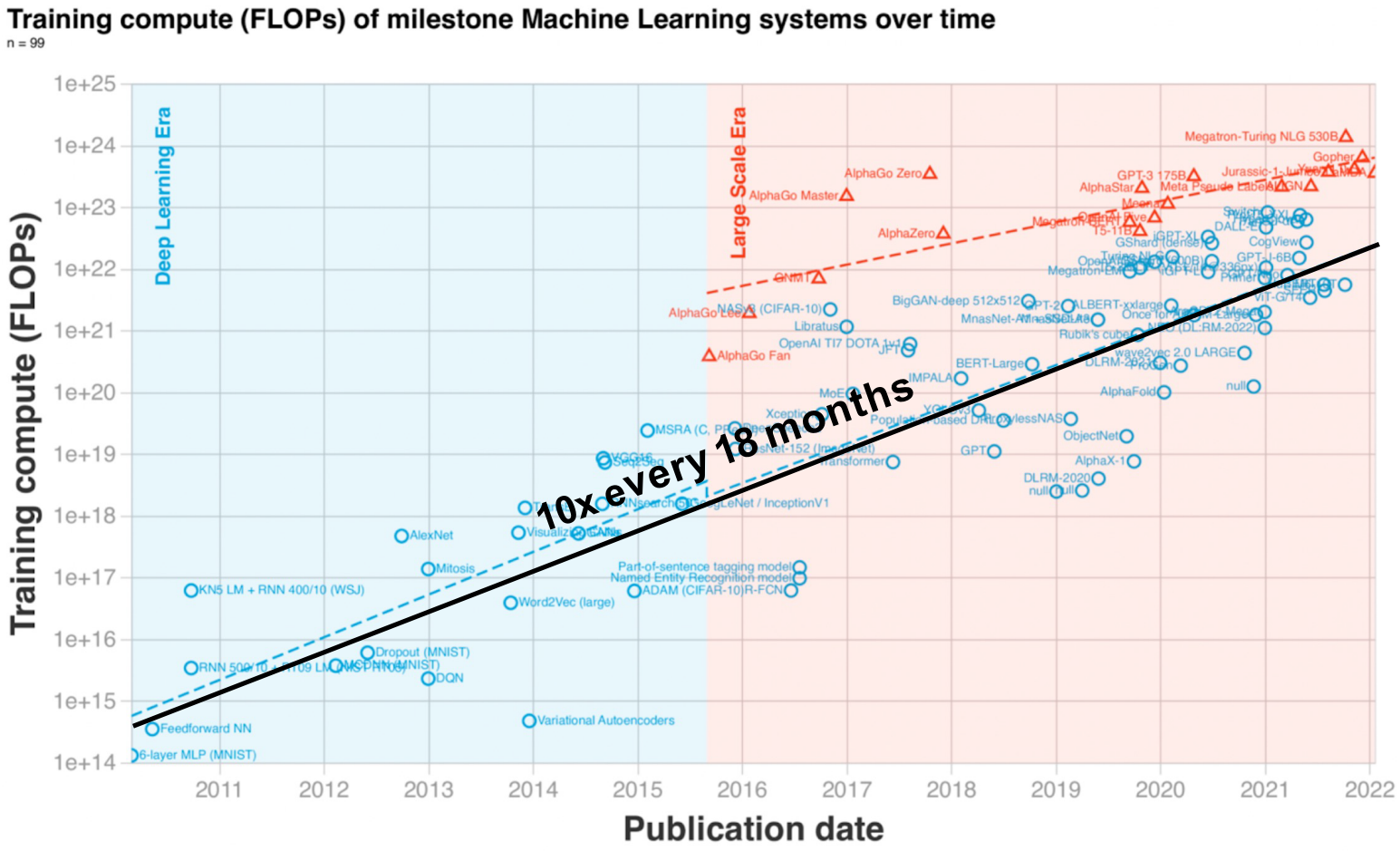
Evaluation criteria: performance

- Response time: how long does it take for a task to complete
- Throughput: Rate at which group of tasks can be completed
- Predictability: Are performance metrics constant over time?

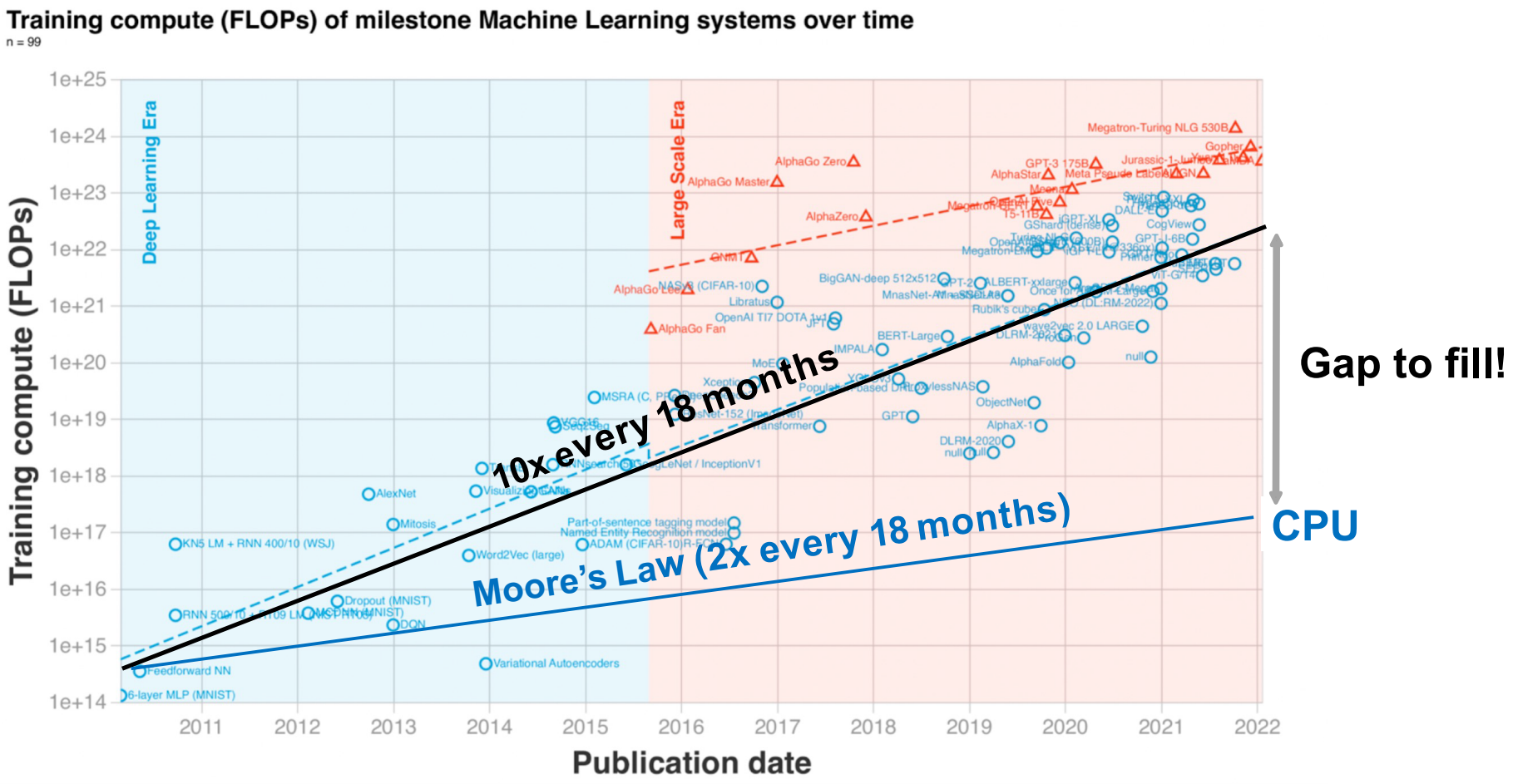
One last note…

- Operating systems in the age of AI
- Everything gets reinvented → this class more relevant than ever

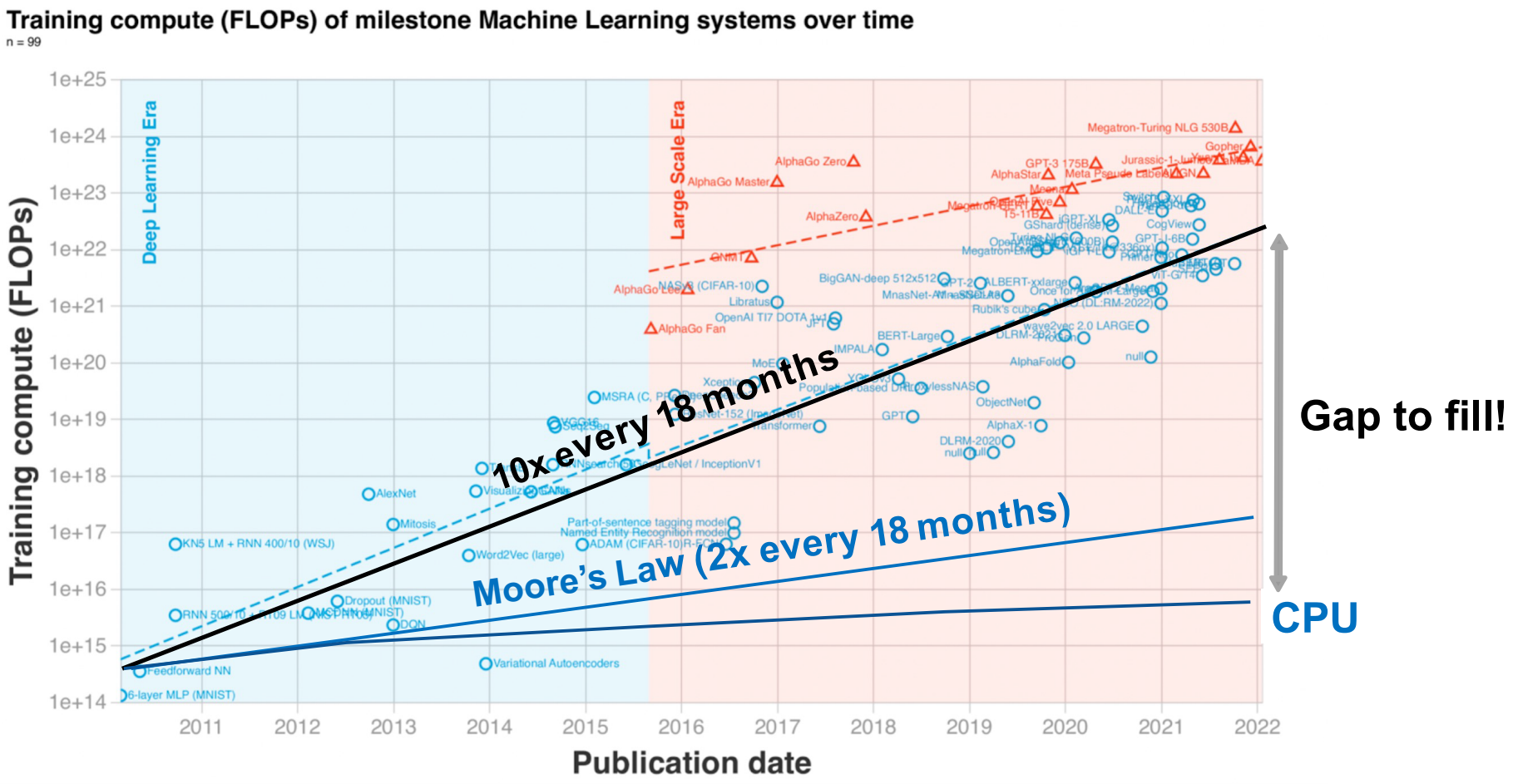
AI demands are exploding



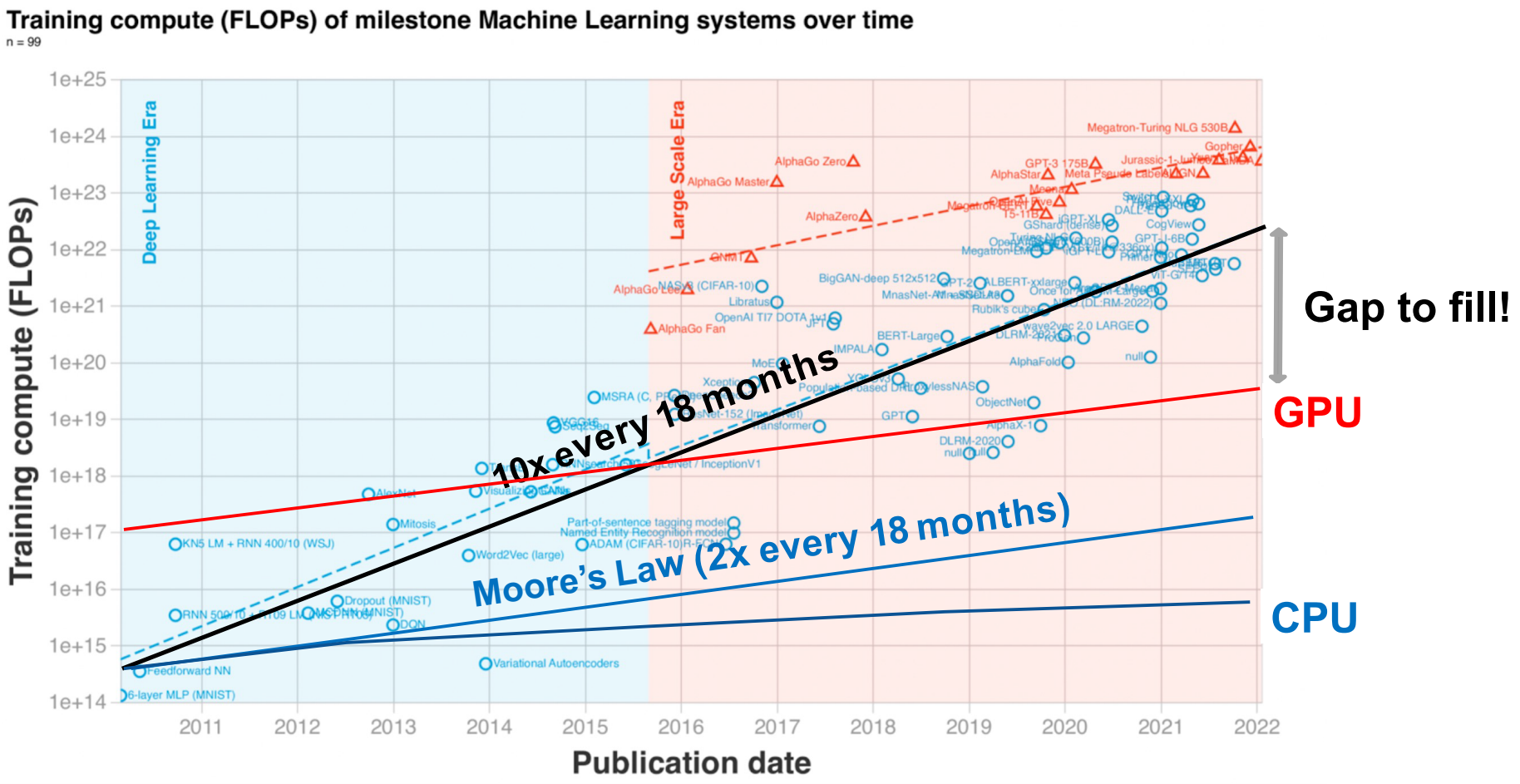
Growing gap between demand and supply



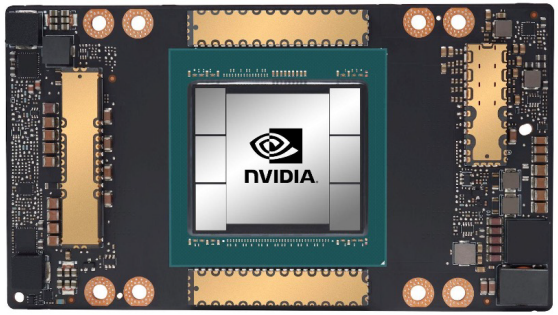
Growing gap between demand and supply

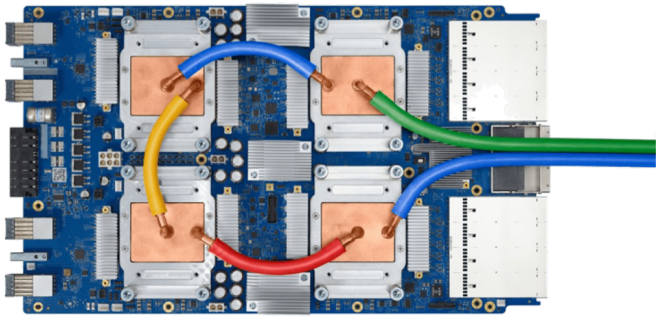


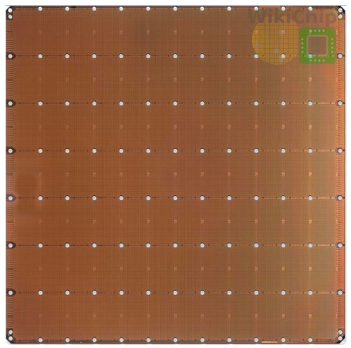
Specialized hardware not enough

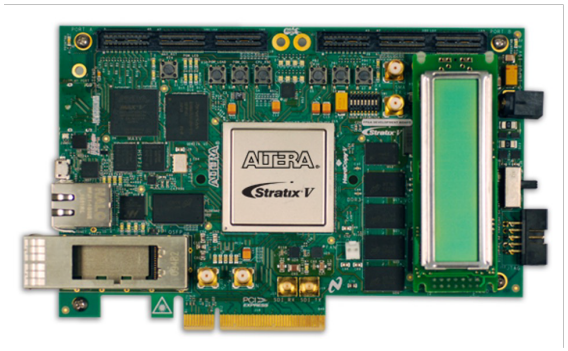


Specialized processors to the rescue → more complexity

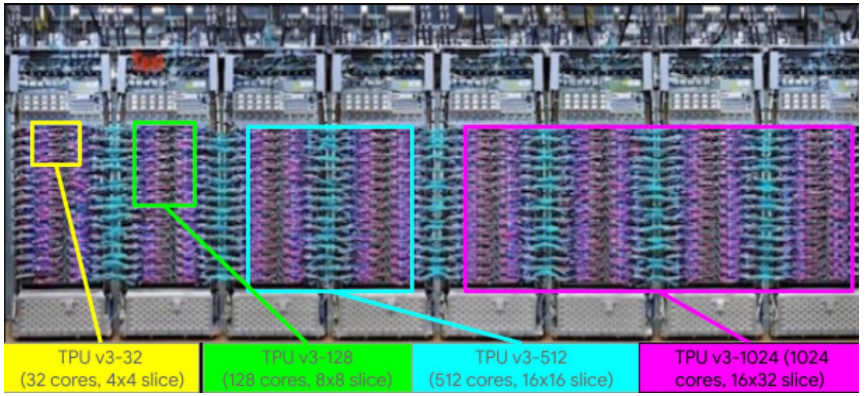




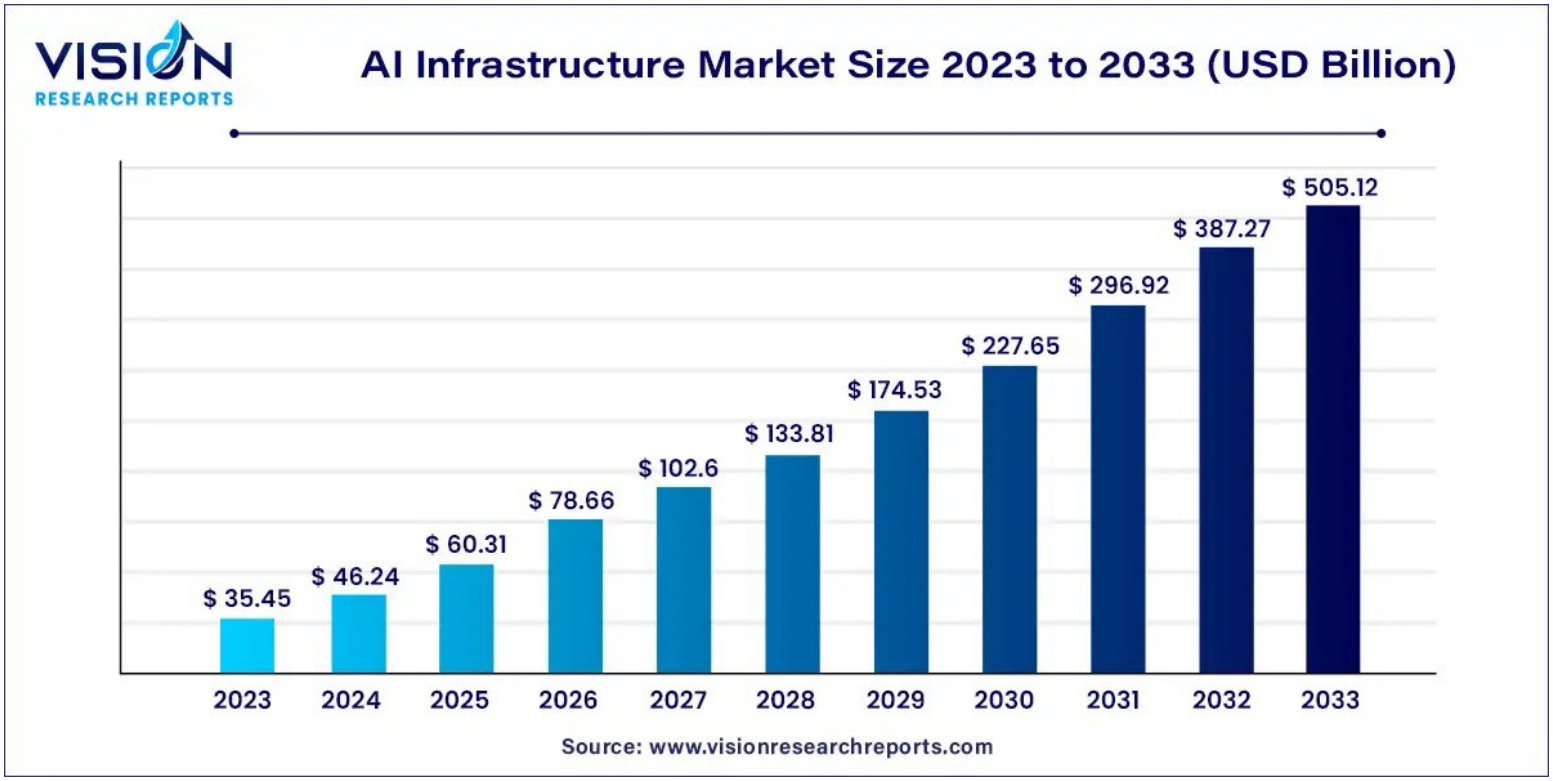




From servers to pods → more complexity



Huge spends on the AI infrastructure



All the infrastructures need to be managed:
Scale changes, hardware changes, application changes → OS will change!

Goals today

- Why should you care?
  - The OS is everywhere
- Why is it hard?
  - Deal with many different devices, many different time scales. Safety-critical
- What is an Operating System?
  - Provides abstraction of a simple, infinite virtual machine
  - Three roles: referee, illusionist, and glue
  - A good OS cares about performance, reliability, security and portability

Thanks