

Operating Systems

Lecture I

Why Learn Operating System? And How?

Prof. Mengwei Xu



Goals for Today

- About this course
- What is OS?
- Why learn OS?
- Challenges and how to overcome?
- Brief history of OS

- Warmups
 - Computer organization
 - CPU, ISA, assembly, etc
 - Lifetime of a “Hello World” program



Goals for Today

- **About this course**
- What is OS?
- Why learn OS?
- Challenges and how to overcome?
- Brief history of OS

- Warmups
 - Computer organization
 - CPU, ISA, assembly, etc
 - Lifetime of a “Hello World” program

About This Course

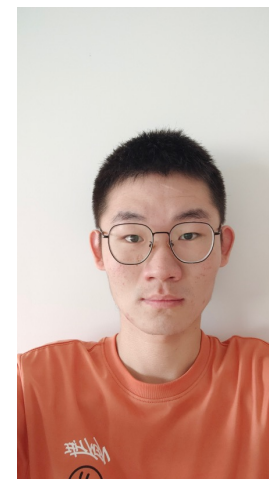
- Mengwei XU (徐梦炜) , 副教授, 硕/博导师
- Office: 科研楼1107
- BA and PhD from PKU, joined BUPT in 2020
- Personal page: <https://xumengwei.github.io/>
- System Software, Edge Computing, ML Systems
- TAs:



杨叶轩



王昕阁



杜嘉骏

About This Course

- Textbooks
 - <Operating System Principles & Practice> (2nd edition) Thomas Anderson et al
 - <Operating Systems: Three Easy Pieces> (version 1.1.0) Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
 - <Operating System Concepts> (7th Edition) Abraham Silberschatz et al
- Slides
 - Much of the contents are from Ion Stoica @ Berkeley!
- Course time
 - 8:00-9:35, Tuesday
 - 9:50-11:25, Friday
- Materials
 - There will be a website that holds everything about the course
 - <https://buptos.github.io/index.html>
- Discipline
 - Ask a question anytime, but do not disturb others from learning.

About This Course

- WeChat Group



Ahead-of-course Questionnaire

对本课程的感受和建议（欢迎长篇大论，褒贬都可以，有助于本人改进教学质量）

老师讲的太好了，收获很大

整体上比较难，需要一定的前置基础才能听懂，但是老师讲的很好，期末复习的时候整体回顾有恍然大悟的感觉

感觉通过老师的教授，学到了很多有益的知识，受益匪浅。

通过lab也学到了很多有用的知识。

但是由于缺少前驱课程的学习（比如北邮计算机专业与ai专业不开设汇编），在做lab时也遇到了一些困难。

我之前学完了南京大学蒋炎岩老师开设的『操作系统：设计与实践』这门课，感觉效果很好，老师可以去看看。但是感觉在北邮按照蒋炎岩老师的方式授课，由于培养方案设置问题，可能会使得课程的难度较大。

课程内容很好，希望lab讲解可以早一些更详细一些，有份简易的指导书

老师可以多布置一些跟进课堂内容的lab，每个规模更小一些，这样可以通过动手操作更好复习掌握知识。

感觉这个课更像一个提高课，初学操作系统的话第一次上课很难听懂，但是回头复习再看回放的话就发现老师讲的很好

从我个人角度讲，徐老师是我大学所有老师中可以排到前几名的好老师，授课思路清晰，备课认真，尤其是愿意聆听学生的困难和建议，这点几乎没几个老师做得好而且通过作业和录屏，我对OS有了切真实用的认识，代码能力也有显著提高如果以后有学弟学妹问我要不要选徐老师的课，我一定会强烈推荐！
谢谢老师！

来自末位学生的体验：讲课可以说是全场最佳，考试难度和考前复习也很到位。

美中不足就是Lab实在是太看得起我了，即使是东拼西凑也很难真正意义上吃透一些Lab任务，真正理想化地遵守学术规范独立完成就是难以想象了。

老师教学非常认真，是真正让人收获很大，喜欢上了操作系统，下学期还选老师的课所以假期我在自学rust。老师的PPT，录屏做的很好，复习特别方便。改进的建议也是有的，就是实验部分设置的有点难，因为我们培养方案没有计算机系统概论之类的体系结构课，所以课上汇编，还有实验要用的docker，gdb什么的希望老师在带下一届的时候可以引导他们去做一做，带着电脑一起操作，不至于两眼一抹黑。还有就是希望老师上课时候有一些部分可以通过敲代码来展示操作系统，要不有的东西太过于抽象了，比如老师可以用矩阵加速乘法这个例子贯穿始终。总的来说老师讲的非常好，助教们也很好，谢谢各位老师的辛苦付出。

我认为保持这样就很好了，感受到了老师和助教的认真，也确实学到了很多真正os的东西而不是应试教学，课堂也比较吸引人。建议：教学不应试作业可以稍微应试点，可以多布置点那种题目，你懂得。

我觉得老师讲的很好

个人感觉这学期os课程难度还是比较合理的。boslab设计的也很好。从os课中确实能学到很多东西。

给我带来一点不便的是课件中有一些地方的英文比较难以理解，我觉得如果课件中多一些中文注释就方便多了。

另外，lab的要求如果能再详细一些就更好了，比如可以录个视频阐述一下，或者课上讲一下。这学期只看网页上的要求有时会比较迷惑

最后，希望徐老师的os课越做越好！

继续保持

确实感觉学到了很多东西，对计算机体系有了一个更完整的认识。一些建议就是希望实践性的任务（lab和作业）布置的更详细一些；课上的ppt重点部分可以多标一些高亮。

上课方式能够充分调动思考的积极性，受益匪浅，布置lab时希望多给一些提示，有时候因为相关知识缺失导致没有方向从而花费过量时间

老师讲课出色

内容挺好，就是太难了，因为前置课程不足

课程内容非常细致，内容很丰富，但是考核的时候多选题很难做，可以考虑换一换题型，或者少选得半分

编程作业较多，但很有实际意义，建议给出参考材料，节约学生时间；lab布置最好是在相关知识都讲完以后；office hour很有意义；课程内容适中，但希望老师讲课能有侧重点；谢谢老师和助教的付出，你们辛苦了！

教学方式很好，教学质量很高，听老师的课感觉学的很明白，谢谢老师

Lab1难度太大了，因为我们没学过汇编，后面的Lab在网上学一学还能懂，汇编看不懂，看懂了也不会Linux写代码，课程方面内容实在太多了，所以复习周期很长，加上做题的时间，大概花十天左右，以后可以早点提醒同学复习。

挺好的

上了徐老师的课，最大的感受可以用一个例子来说明。关于内存管理方面cache的n路组相连计算机组成原理也讲过 但当时我直到考试之前也并没有学懂 然后上操作系统的时候就感觉理解得很轻松 其实关于操作系统这门课我个人觉得很有难度 尽管老师上课很认真讲解 但有些概念我认为第一次接触很难去花一遍去理解 对此我的做法是上课先尽量跟着听 然后回去看着录像将老师讲的每句话尽可能记下来 这样下来就感觉几乎没有弄不懂的点 所以希望老师可以将录屏坚持下去！非常感谢徐老师一年的付出 祝老师新年快乐！

老师讲的很好，但是就是lab对于没有基础的学生来讲有点难

感受超级好，理论和实践的的教学都很好的落实了，学到了很多，非常感谢老师和TAs!

可以拓展更多前沿内容

感觉学到了很多东西，老师也很用心，但是有个小建议是希望lab能给个qemu操作语法说明
谢谢老师撈我好开心嘿嘿嘿老师新年快乐

希望期末考题大部分和考研相似，小部分为开放题

好！

好！

老师讲的很好很细致，但是对于一些概念理解容易不透彻，可以多复习多提及重要知识点

lab好难，希望可以在前期做那个环境配置和一些基础步骤的指导书，或者有个示例代码

真挺好真挺好，能看的出来很用心🥰

About This Course - Grading

- Homework (30%)
 - <200 lines of code as designed
 - Reports, code, documents, paper reading, etc.
- Final (70%)
- Bonus (5%) – MIT JOS 6.828
 - <https://pdos.csail.mit.edu/6.828/2018/labs/lab1>
 - Let TAs know that you take the challenge within 2 weeks.
 - There will be face-to-face test at the end of semester
- Bonus (1%) – Find error in ChatGPT
 - If you find it answers wrongly about OS concepts (in English)
 - Report it immediately to TAs – only one student gets credits for the same error

Projects

| | Contents | Environment setups and debug | Grading policy | Time investment | Reason to choose |
|----------------------------------|---|------------------------------|--|---------------------|--|
| Simulation-based labs | Focus on the high-level concepts of OS | Very simple | • Complete all labs and write documents | ~ 15 hrs | Reasonable efforts, still learn some basic OS knowledge |
| JOS (MIT 6.868) | Implement a fully-fledged kernel from scratch by filling out the missing pieces | QEMU emulator; hard to debug | <ul style="list-style-type: none"> • Complete all labs and write documents • 1-to-1 interview in the end of semester | ~ 80 hrs | Probably the once-in-life chance to write your own kernel! |

Advise: do not take the JOS for better grades

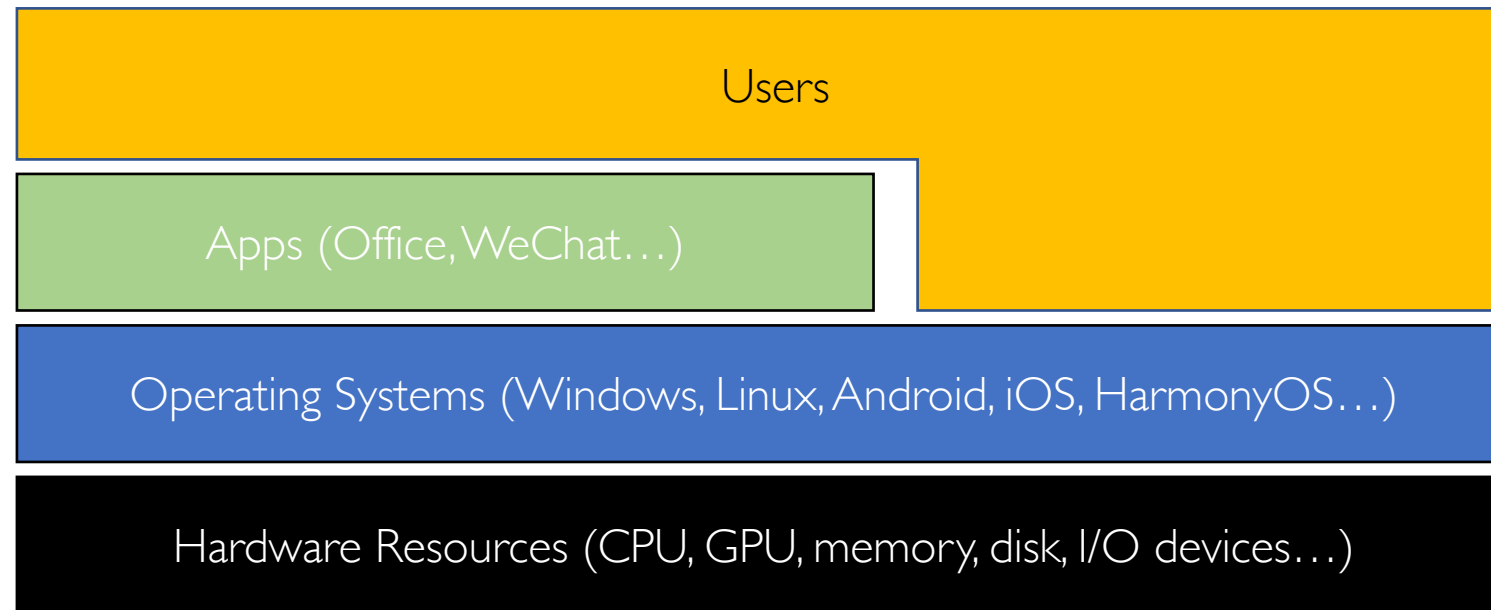
Goals for Today

- About this course
- **What is OS?**
- Why learn OS?
- Challenges and how to overcome?
- Brief history of OS

- Warmups
 - Computer organization
 - CPU, ISA, assembly, etc
 - Lifetime of a “Hello World” program

What is OS?

- A bridge between hardware and apps/users.
- Or, a special software layer that provides and manages the access from apps/users to hardware resources (CPU, memory, disk, etc).
 - *“We can solve any problem by introducing an extra level of indirection” – David Wheeler*





The Role of OS

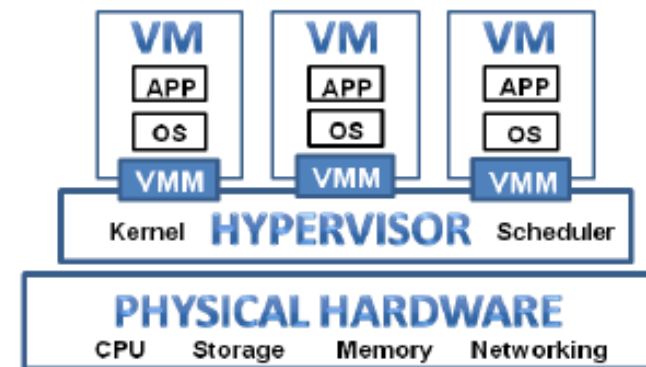
- OS as referee
- OS as illusionist
- OS as glue

The Role of OS

- OS as referee
- OS as illusionist
- OS as glue
- Resource allocation
 - Multitasks on constrained resources
 - What if there is an infinite loop?
- Isolation
 - Fault in one app shall not disrupt others
 - Prevent malicious attackers
- Communication
 - Reliable, safe, and fast

The Role of OS

- OS as referee
 - OS as illusionist
 - OS as glue
- Illusion of resources not physically present
 - Processor, memory, screen, disk..
 - Ease of debugging, portability, isolation
 - A step further: virtual machine



The Role of OS

- OS as referee
- OS as illusionist
- OS as glue
- Providing common services to facilitate resource sharing
 - read/write files
 - share memory
 - pass messages
 - UI
- Decouple HW and app development
 - As an interoperability layer for portability
 - Most applications can evolve independently with OS, unless those require very low-level programming such as databases



Code Example: CPU Virtualization

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/time.h>
4  #include <assert.h>
5  #include "common.h"
6
7  int
8  main(int argc, char *argv[])
9  {
10     if (argc != 2) {
11         fprintf(stderr, "usage: cpu <string>\n");
12         exit(1);
13     }
14     char *str = argv[1];
15     while (1) {
16         Spin(1);
17         printf("%s\n", str);
18     }
19     return 0;
20 }
```



Code Example: CPU Virtualization

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/time.h>
4 #include <assert.h>
5 #include "common.h"
6
7 int
8 main(int argc, char *argv[])
9 {
10     if (argc != 2) {
11         fprintf(stderr, "usage: cpu <string>\n");
12         exit(1);
13     }
14     char *str = argv[1];
15     while (1) {
16         Spin(1);
17         printf("%s\n", str);
18     }
19     return 0;
20 }
```

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
A
^C
```

Code Example: CPU Virtualization

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/time.h>
4 #include <assert.h>
5 #include "common.h"
6
7 int
8 main(int argc, char *argv[])
9 {
10     if (argc != 2) {
11         fprintf(stderr, "usage: cpu
12             exit(1);
13     }
14     char *str = argv[1];
15     while (1) {
16         Spin(1);
17         printf("%s\n", str);
18     }
19     return 0;
20 }
```

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
A
^C
```

```
prompt> ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 7353
[2] 7354
[3] 7355
[4] 7356
A
B
D
C
A
B
D
C
A
...
```

Code Example: Memory Virtualization

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "common.h"
5
6  int
7  main(int argc, char *argv[])
8  {
9      int *p = malloc(sizeof(int));           // a1
10     assert(p != NULL);
11     printf("(%d) address pointed to by p: %p\n",
12           getpid(), p);                     // a2
13     *p = 0;                                 // a3
14     while (1) {
15         Spin(1);
16         *p = *p + 1;
17         printf("(%d) p: %d\n", getpid(), *p); // a4
18     }
19     return 0;
20 }
```

Code Example: Memory Virtualization

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "common.h"
5
6  int
7  main(int argc, char *argv[])
8  {
9      int *p = malloc(sizeof(int));           // a1
10     assert(p != NULL);
11     printf("(2134) address pointed to by p: %p\n",
12           getpid(), p);                     // a2
13     *p = 0;                                 // a3
14     while (1) {
15         Spin(1);
16         *p = *p + 1;
17         printf("(2134) p: %d\n", getpid(), *p); // a4
18     }
19     return 0;
20 }
```

```
prompt> ./mem
(2134) address pointed to by p: 0x200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
```


Code Example: Memory Virtualization

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "common.h"
5
6  int
7  main(int argc, char *argv[])
8  {
9      int *p = malloc(sizeof(int));
10     assert(p != NULL);
11     printf("(%d) address pointed to
12           getpid(), p);
13     *p = 0;
14     while (1) {
15         Spin(1);
16         *p = *p + 1;
17         printf("(%d) p: %d\n", getp
18     }
19     return 0;
20 }
```

```
prompt> ./mem
(2134) address pointed to by p: 0x200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
```

```
prompt> ./mem & ./mem &
[1] 24113
[2] 24114
(24113) address pointed to by p: 0x200000
(24114) address pointed to by p: 0x200000
(24113) p: 1
(24114) p: 1
(24114) p: 2
(24113) p: 2
(24113) p: 3
(24114) p: 3
(24113) p: 4
(24114) p: 4
...
```

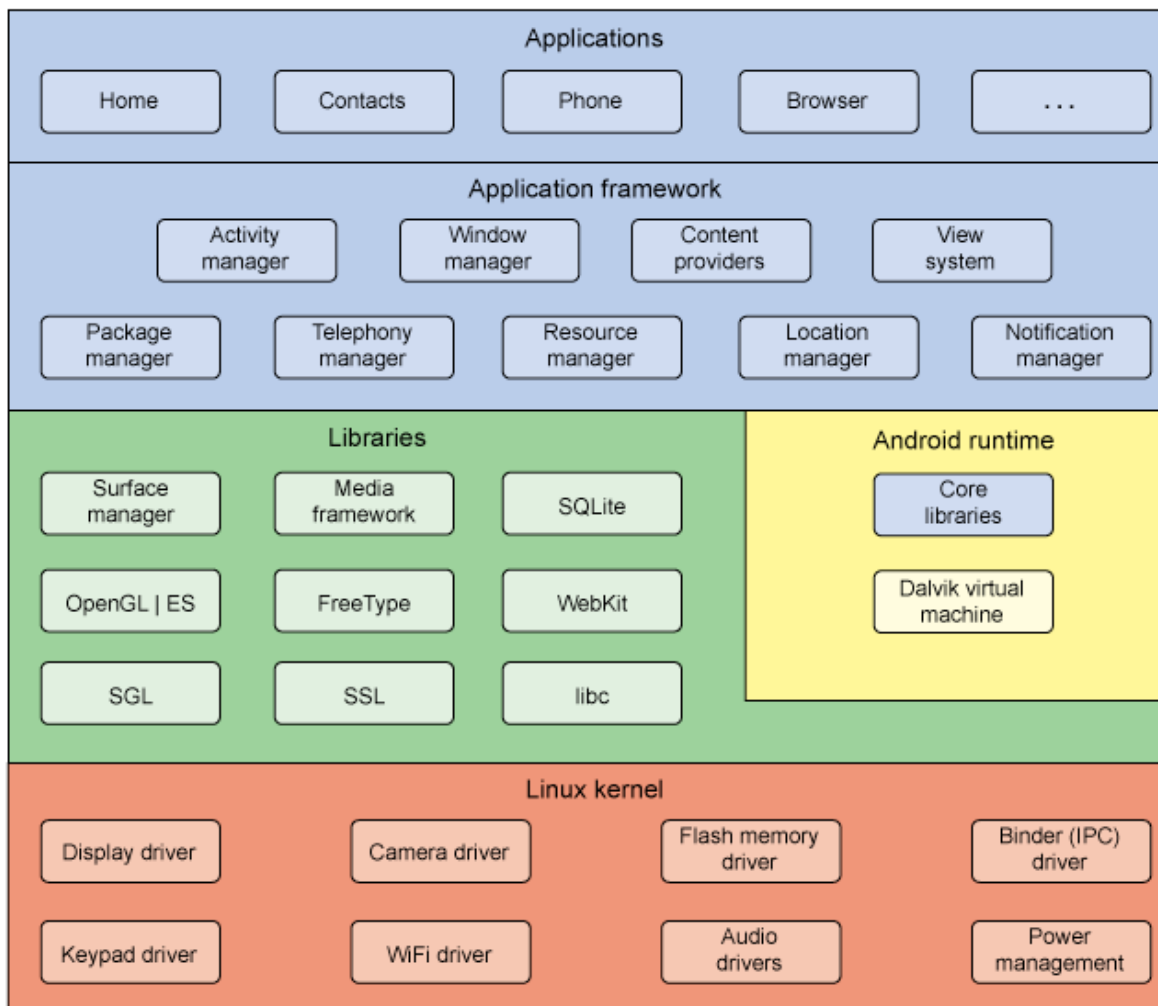
Why not 1,2,3,4..?

The Goal of OS

- Managing hardware resources
 - Allocating resources to concurrent tasks, who gets more?
 - Determining hardware status: CPU frequency, I/O device on/off, etc.
- Facilitate app developers
 - Illusion of resources not physically present
 - Large memory size
 - Isolation of apps
 - so a bug does not cause the machine down
 - Cross-apps communication
 - Without
- Facilitate users
 - UI components and rendering

The OS concept is expanding

- Android/TensorFlow/WeChat
- OS vs. kernel (内核)

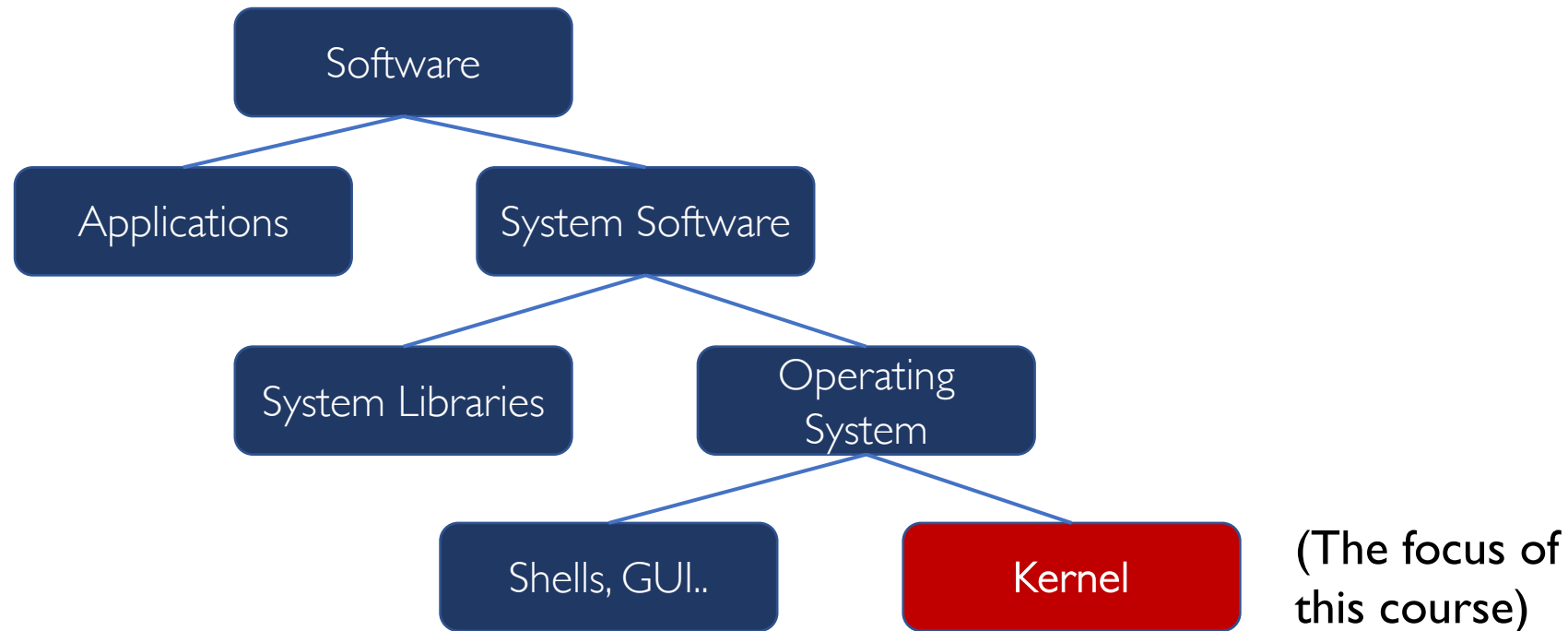


(The focus of this course)



The OS concept is expanding

- Android/TensorFlow/WeChat
- OS vs. kernel (内核)



Goals for Today

- About this course
- What is OS?
- **Why learn OS?**
- Challenges and how to overcome?
- Brief history of OS

- Warmups
 - Computer organization
 - CPU, ISA, assembly, etc
 - Lifetime of a “Hello World” program

Why Study OS? (1/3)

- OS is a fundamental piece of computer science.
 - It's everywhere and will be long-standing.
 - It extensively adopts the most important concepts in CS.
 - It extensively interacts with hardware, compiler, PLs, runtime, etc.



what are the top-5 important courses in computer science



Computer science is a diverse field with various sub-disciplines and specializations. The importance of specific courses can depend on your career goals and interests. However, here are five foundational and important courses in computer science:

1. **Data Structures and Algorithms:** This course is fundamental in computer science. It teaches you how to design, analyze, and implement efficient data structures (e.g., arrays, linked lists, trees) and algorithms (e.g., sorting, searching) essential for solving complex problems and optimizing software performance.
2. **Programming Languages:** A course on programming languages explores different programming paradigms (e.g., imperative, object-oriented, functional), syntax, semantics, and language design principles. It helps you become proficient in multiple programming languages, which is crucial for versatility as a developer.
3. **Databases:** Understanding databases and database management systems (DBMS) is essential for handling and storing data efficiently. This course covers database design, querying with SQL, transaction management, and database optimization.
4. **Operating Systems:** An operating systems course delves into the principles and components of modern operating systems, such as process management, memory management, file systems, and concurrency. It's crucial for understanding how software interacts with hardware.
5. **Computer Networks:** Computer networks are the backbone of modern computing. This course teaches the fundamentals of networking, including protocols, routing, security, and network design. Understanding how data moves across networks is vital for various applications, including web development and cybersecurity.

然后, 我觉得计算机专业最重要的5门课是:

- 1、数据结构与算法
- 2、计算机组成原理
- 3、操作系统
- 4、计算机网络
- 5、数据库

之所以把这五个作为最重要的课, 其实包含一点功利之心, 因为校招面试大厂的时候, 这五门基础课问的最多的, 至于其他的编译原理、汇编语言、数学基础课程不是说不重要, 只是相对于校招面试这些课程问题的比较少。

接下来, 我就跟大家分享下, 我看过数据结构与算法、计算机组成、操作系统、计算机网络、数据库这些课程的书和视频, 都是从入门再到进阶的路线。

这里推荐下我当初自学的书籍和视频。

▲ 赞同 2095 ▼

● 30 条评论

↗ 分享

★ 收藏

♥ 喜欢

...

收起 ^

Why Study OS? (2/3)

- OS is useful
 - Whatever you work at correlates to OS
 - HPC, mobile/edge/cloud computing, ML/AI, security, etc..
 - Helps you understand why your program does not work well
 - Helps you to get a good job

```
for i in xrange(4096):  
    for j in xrange(4096):  
        for k in xrange(4096):  
            C[i][j] += A[i][k] * B[k][j]
```

| Version | Implementation | Running time (s) | GFLOPS | Absolute speedup |
|---------|-----------------------------|------------------|---------|------------------|
| 1 | Python | 25,552.48 | 0.005 | 1 |
| 2 | Java | 2,372.68 | 0.058 | 11 |
| 3 | C | 542.67 | 0.253 | 47 |
| 4 | Parallel loops | 69.80 | 1.969 | 366 |
| 5 | Parallel divide and conquer | 3.80 | 36.180 | 6,727 |
| 6 | plus vectorization | 1.10 | 124.914 | 23,224 |
| 7 | plus AVX intrinsics | 0.41 | 337.812 | 62,806 |

Charles E Leiserson, Neil C Thompson, Joel S Emer, Bradley C Kuszmaul, Butler W Lampson, Daniel Sanchez, and Tao B Schardl. There's plenty of room at the top: What will drive computer performance after moore's law? Science, 368(6495), 2020.

Why Study OS? (3/3)

- OS is cool
 - Probably the most complex software you will deal with
 - The critical path to understand how computer *really* works



Why Study OS? – especially as AI students



shall students majoring artificial intelligence study operating system course

From ChatGPT



In an academic curriculum focused on Artificial Intelligence (AI), the Operating Systems course might not appear to be directly applicable, especially when compared to courses like Machine Learning, Natural Language Processing, or Computer Vision. However, there are several compelling reasons why students majoring in AI might still benefit from understanding operating systems:

1. **Resource Management:** Understanding how an operating system allocates resources can help AI developers optimize computational tasks, particularly important for machine learning models that may require substantial computational power.
2. **Concurrency:** AI applications often involve parallel or distributed computing. An understanding of processes, threads, and scheduling can be beneficial for implementing and optimizing these types of systems.
3. **System-Level Understanding:** A well-rounded comprehension of computing environments can help AI practitioners identify bottlenecks or issues that may arise during the development and deployment of AI applications.
4. **Versatility:** AI specialists often need to collaborate with engineers focused on other parts of a software stack. A knowledge of operating systems can make this collaboration more effective and can expand the range of roles that an AI major might be qualified for.
5. **Software Engineering Practices:** Operating Systems courses often delve into topics that improve one's overall software engineering skills, such as debugging, performance optimization, and understanding of system calls, which are beneficial when developing or maintaining larger AI projects.

In summary, while an Operating Systems course might not be as directly relevant to AI as other specialized courses, the foundational knowledge it provides can be invaluable for more advanced work and for broader career flexibility. Therefore, it's generally a good idea for students majoring in AI to study Operating Systems.



shall students majoring artificial intelligence study operating system course

From LLaMA-70B



Students majoring in Artificial Intelligence (AI) may not necessarily need to take an Operating System (OS) course, depending on their specific interests and career goals within AI. Here's why:

While OS is an important area of computer science that deals with the internal functions and operations of a computer system, it may not directly impact many areas of AI research or application. Many AI techniques involve working at higher levels of abstraction than the low-level details of an OS, focusing instead on machine learning models, natural language processing, computer vision, robotics, and other key subfields within AI.

That being said, there are some potential reasons why studying OS might still be beneficial for AI majors:

1. **Understanding system architecture:** An OS course provides an overview of the entire computer system, including hardware components, memory management, process scheduling, and I/O handling. These concepts can help AI students better understand how various software systems interact with each other and how they can be designed to support AI applications. For example, understanding how the OS manages memory and processes can inform strategies for efficient use of computational resources in deep learning frameworks.
2. **Networking and distributed computing:** As AI applications become more widespread and complex, they often require networking and distributed computing capabilities to function effectively. Learning about OS-related topics like network protocols, socket programming, and distributed systems can provide valuable insights into how AI systems can be designed to communicate and coordinate across different nodes and devices.
3. **Security considerations:** With the growing concern about data privacy and security in AI applications, having a basic understanding of OS-related security mechanisms (e.g., access control, cryptography) could prove useful for AI majors. This knowledge can help them design more secure AI systems and protect against potential cyber threats.

In summary, while an Operating System course may not be strictly necessary for all AI majors, it can complement their studies by providing a broader understanding of computer system architecture, networking, and security concerns that are relevant to many areas within AI research and development.

Who Cares OS?

- Academia: USENIX/ACM/IEEE
 - Conferences: SOSP/OSDI/USENIX ATC/EuroSys/ApSys..
- Companies
 - (direct) Microsoft, Google, Apple, Huawei..
 - (indirect) NVIDIA, Intel, Meta, Alibaba..
 - Hardware and OS always co-evolve!
- Developers who benefit from OS knowledge: everyone!



Goals for Today

- About this course
- What is OS?
- Why learn OS?
- **Challenges and how to overcome?**
- Brief history of OS

- Warmups
 - Computer organization
 - CPU, ISA, assembly, etc
 - Lifetime of a “Hello World” program

Challenges

- Complexity
 - Windows 10 has roughly 10,000,000 lines of code.
- OS directly deals with hardware
 - OS crash means...
- OS manages concurrency (并发)
 - A major source of software bugs
- OS cares many metrics
 - Reliability, availability, performance, energy, etc...
- Gap between concepts and code
- There are historical baggage

How to Overcome?

- Think more and ask why
 - What's the benefit? What if.. ?
- Be interactive
 - In and after course
- Be patient
 - Too many abstractions and indirections
- Get your hands dirty

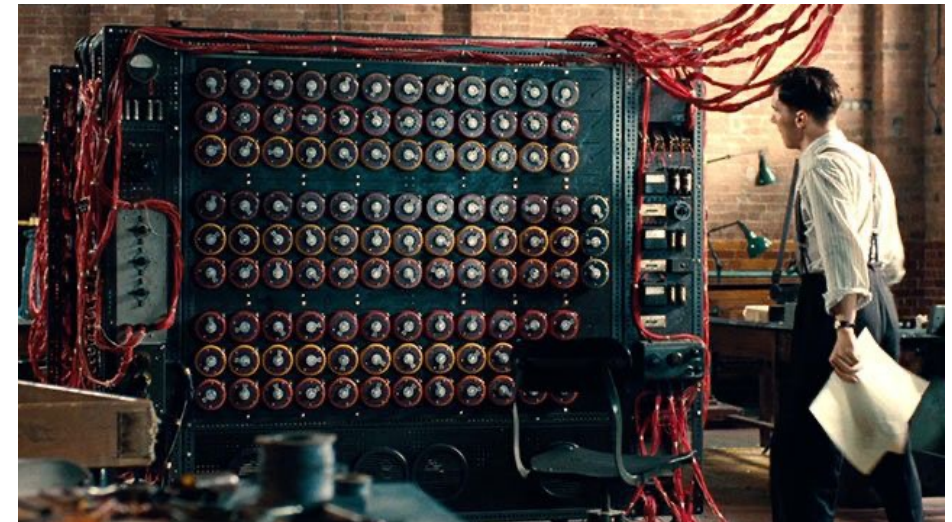
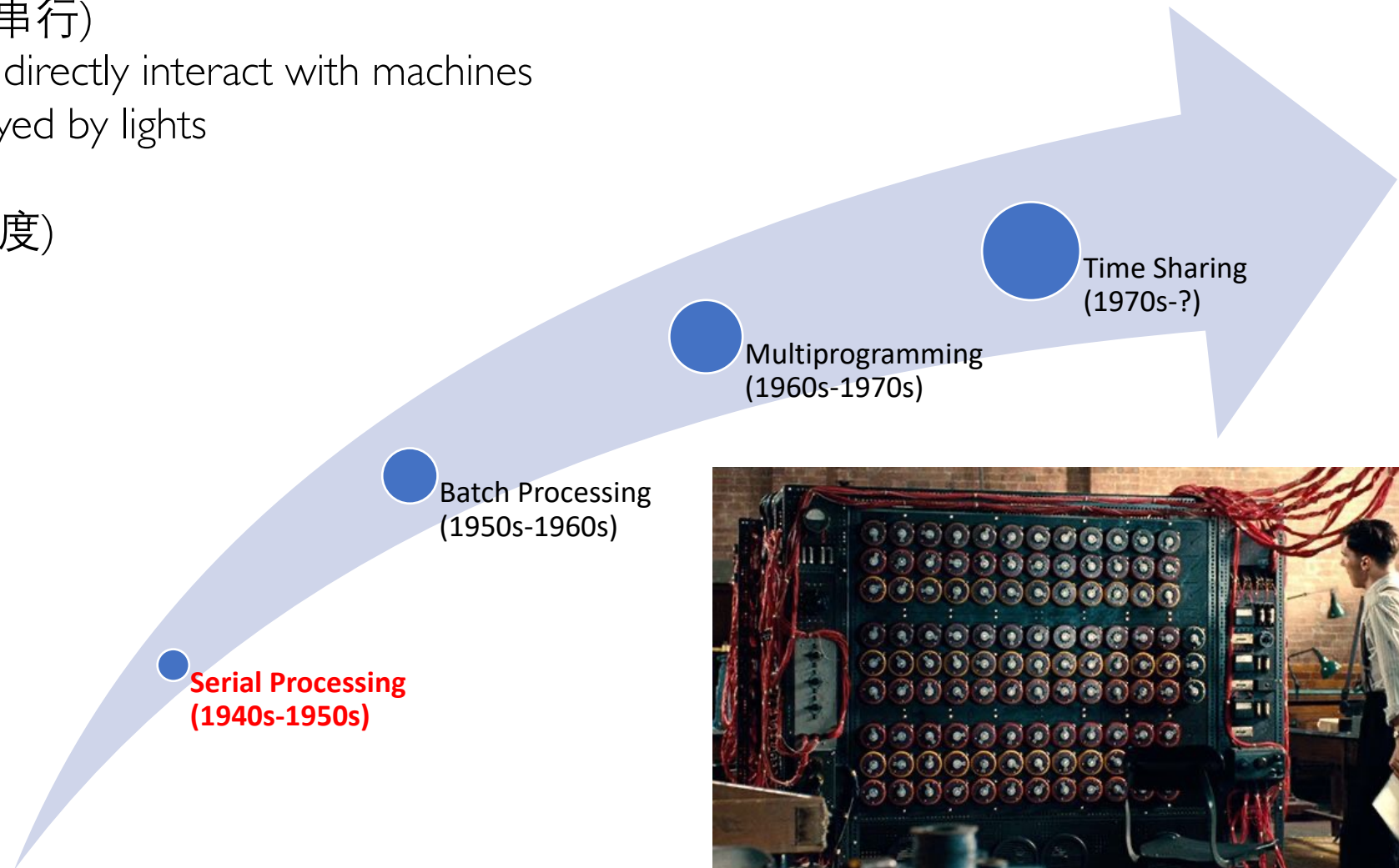
Goals for Today

- About this course
- What is OS?
- Why learn OS?
- Challenges and how to overcome?
- **Brief history of OS**

- Warmups
 - Computer organization
 - CPU, ISA, assembly, etc
 - Lifetime of a “Hello World” program

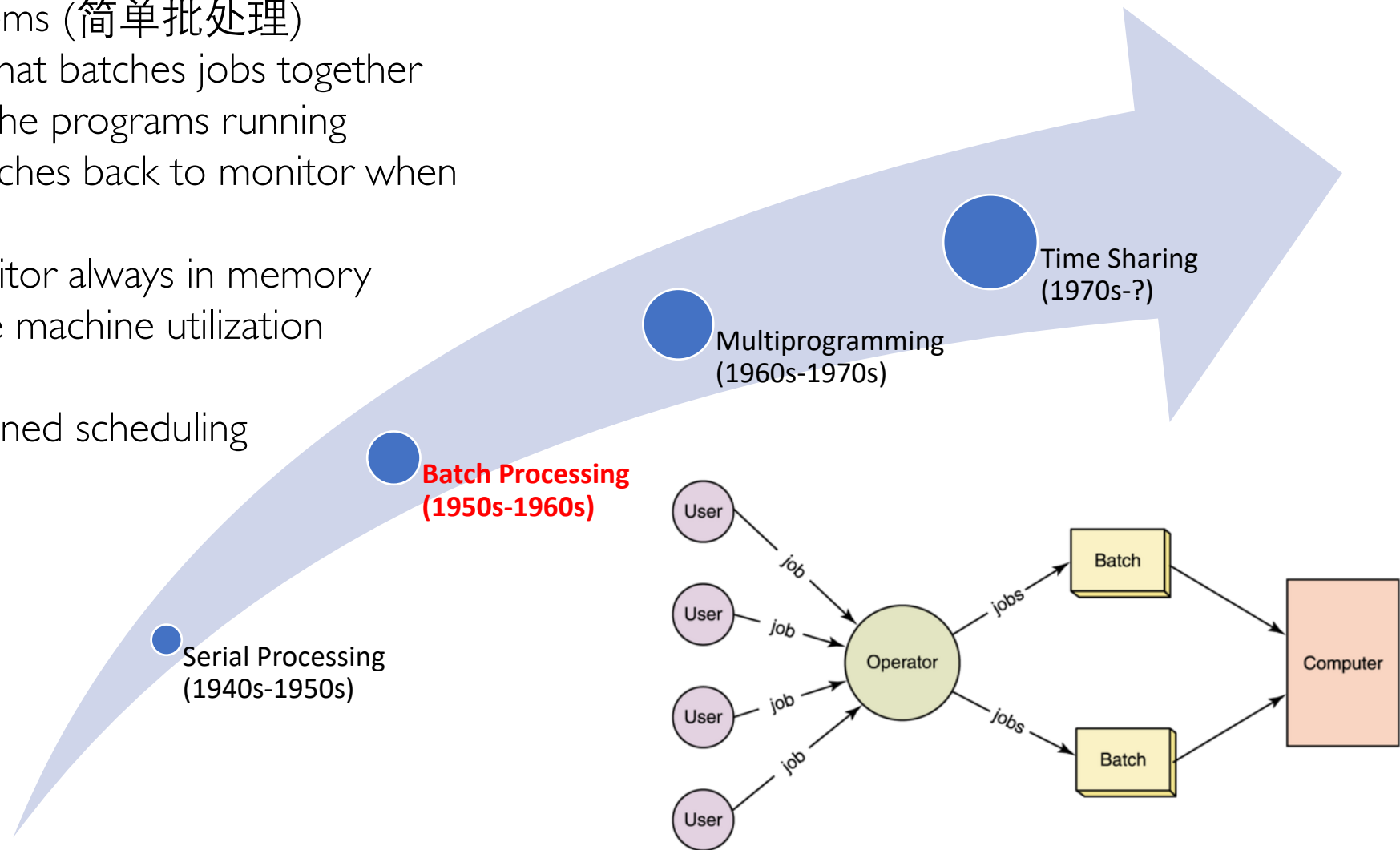
OS history

- Serial Processing (串行)
 - No OS, users directly interact with machines
 - Output displayed by lights
- No scheduling (调度)
- Huge setup time



OS history

- Simple Batch Systems (简单批处理)
 - A "monitor" that batches jobs together and controls the programs running
 - Program branches back to monitor when finished
 - (part of) Monitor always in memory
 - Maximizes the machine utilization
- Simple, coarse-grained scheduling
- Uniprogramming



OS history

- Multiprogrammed Batch Systems (多道批处理)
 - Processor can switch among different jobs when they are running (waiting for I/O)
 - Memory management, scheduling, save and restore
 - Optimized for job throughput (vs. speed)
- Not enough for user interactions

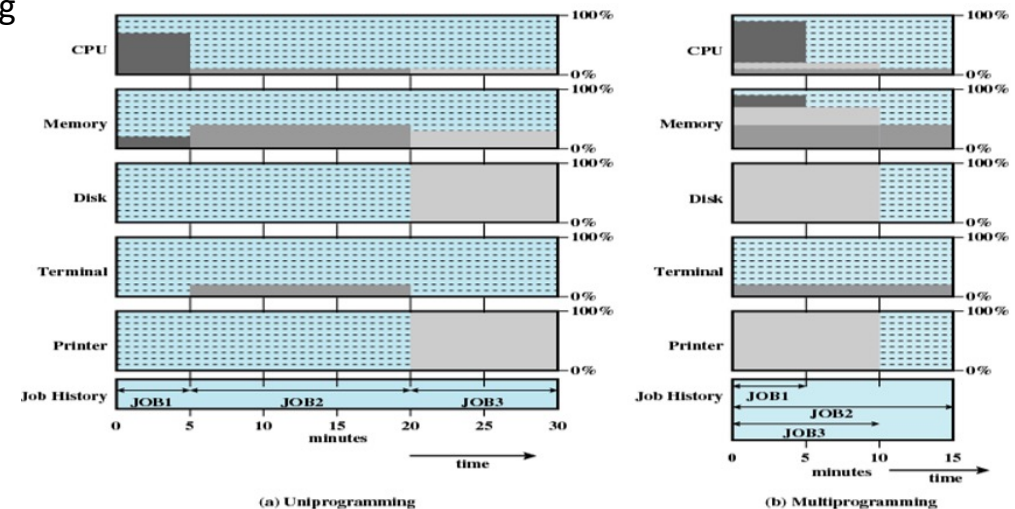
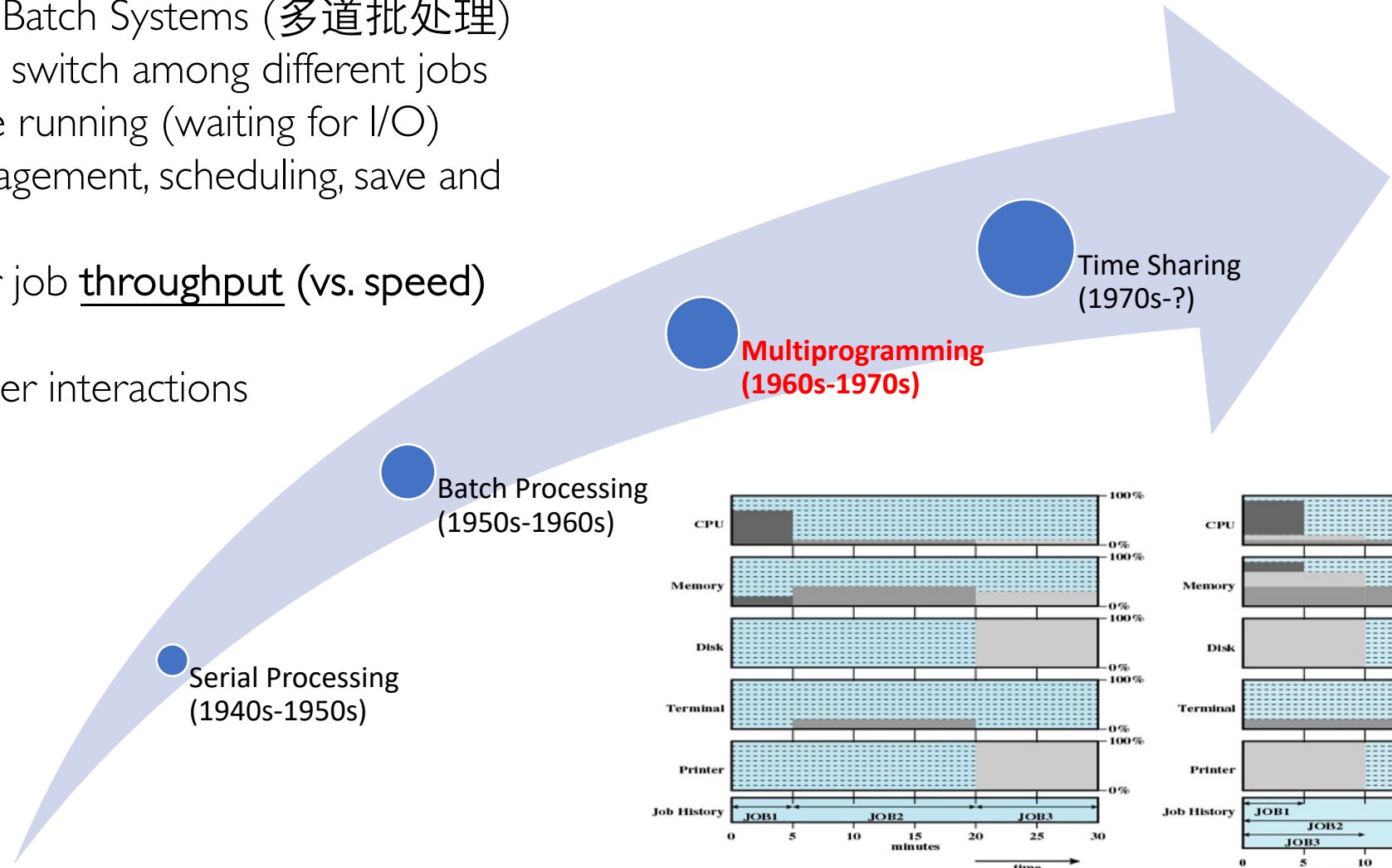
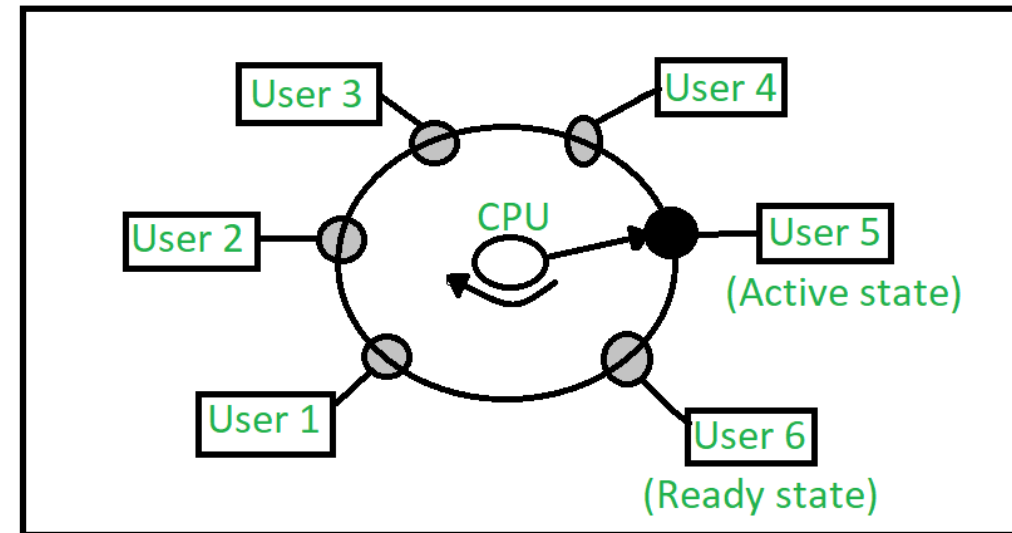
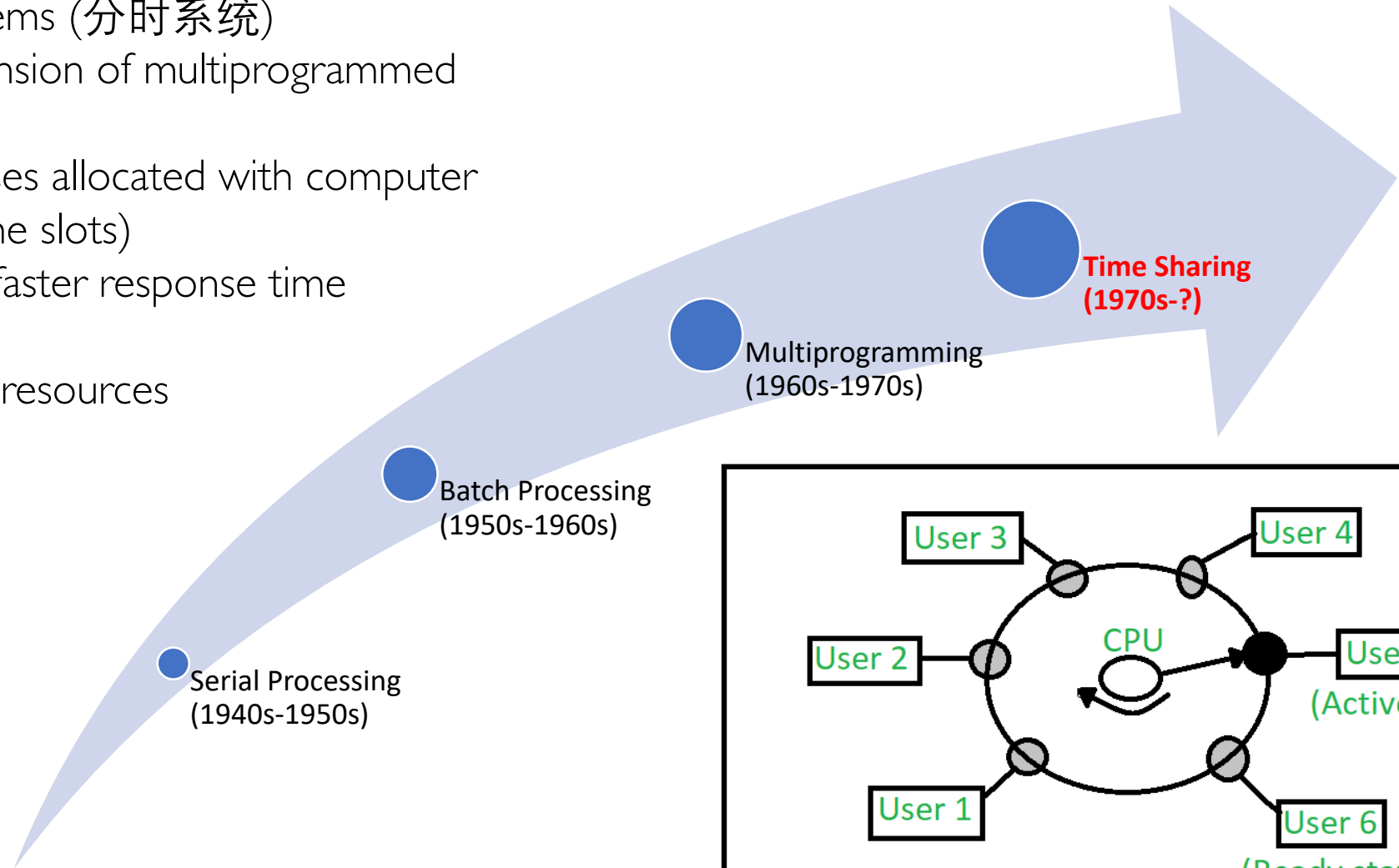


Figure 2.6 Utilization Histograms

OS history

- Time Sharing Systems (分时系统)
 - A logical extension of multiprogrammed systems
 - Users/processes allocated with computer resources (time slots)
 - Designed for faster response time
- It consumes many resources



Other OS types

- Real-time OS (实时操作系统)
 - vs. General-purpose OS (通用操作系统)
 - Used in robots, satellites, airplanes, etc..
- Distributed OS (分布式操作系统)
 - Could be atop traditional OS
 - C/S, P2P
- Microkernel
 - vs. Monolithic kernels
 - Small size: moving drivers, filesystems, etc to user space
 - Securer, but slower
 - Many OSes are hybrid (Mac OS)

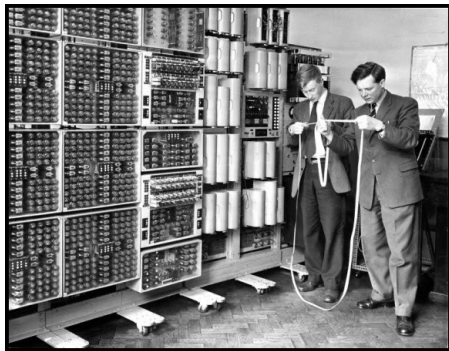
Summary of OS Evolution

- Adapting to new hardware
 - Multi-core CPU, GPU, NPU, NVM, RDMA..
- Adapting to new workloads
 - Machine learning serving/training, distributed programs, AR/VR..
- Adapting to new user demands
 - Faster, more responsive, lower power..

- Hardware is cheaper, yet humans are more expensive

Summary of OS Evolution

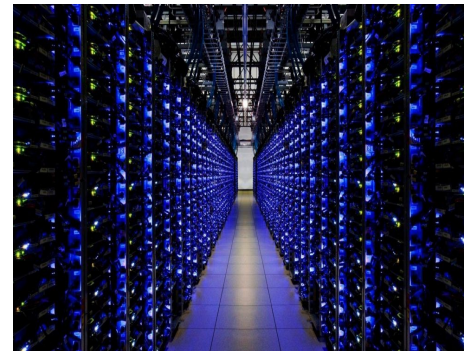
- 分久必合，合久必分



Centralized



Decentralized



Centralized



Decentralized

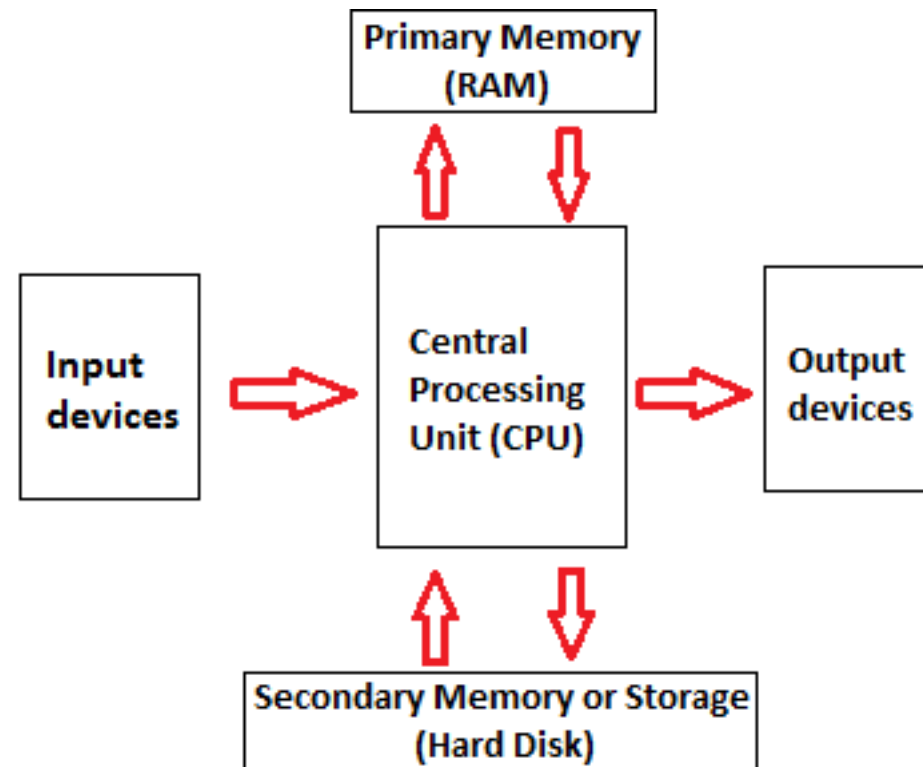
Goals for Today

- About this course
- What is OS?
- Why learn OS?
- Challenges and how to overcome?
- Brief history of OS

- **Warmups**
 - Computer organization
 - CPU, ISA, assembly, etc
 - Lifetime of a “Hello World” program

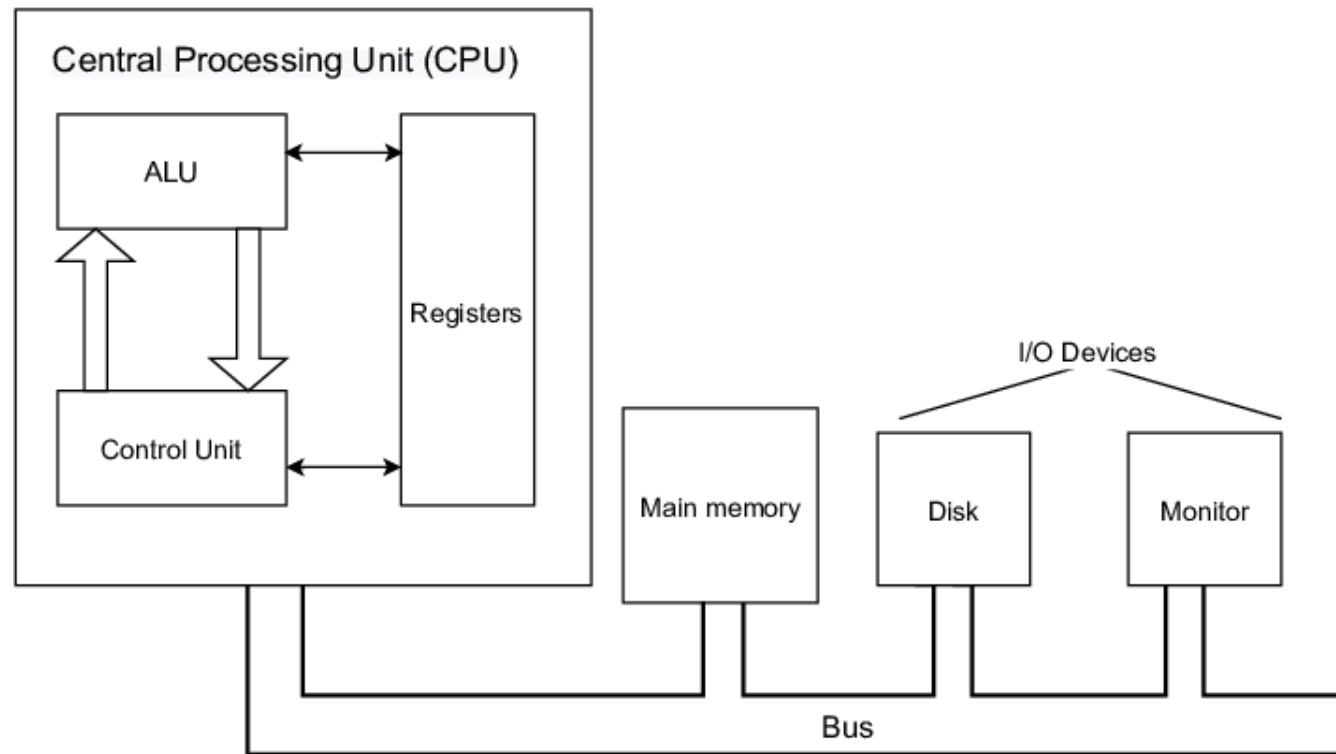
Computer Components

- CPU, Memory, Disk, Input devices, Output devices
 - This is a very rough understanding



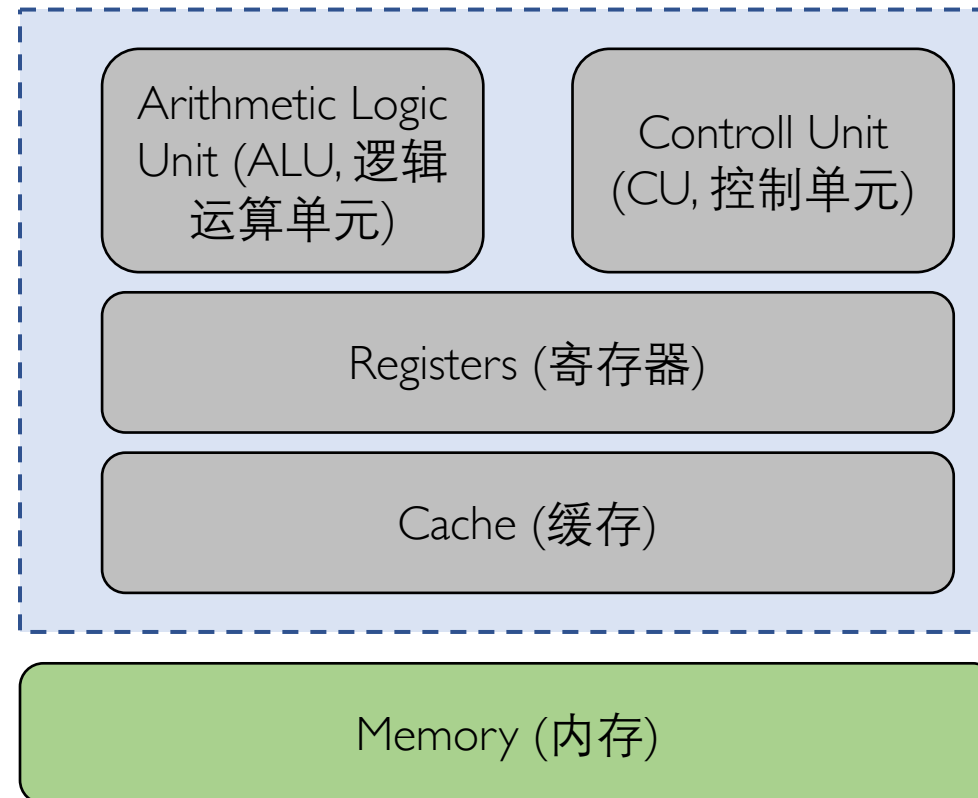
Computer Architecture

- Direct Memory Access (DMA, 直接存储器访问)
- CPU directly accesses data in cache and memory, but not disk.



CPU

- ALU operates on registers
- CU loads/saves data from/to memory



CPU

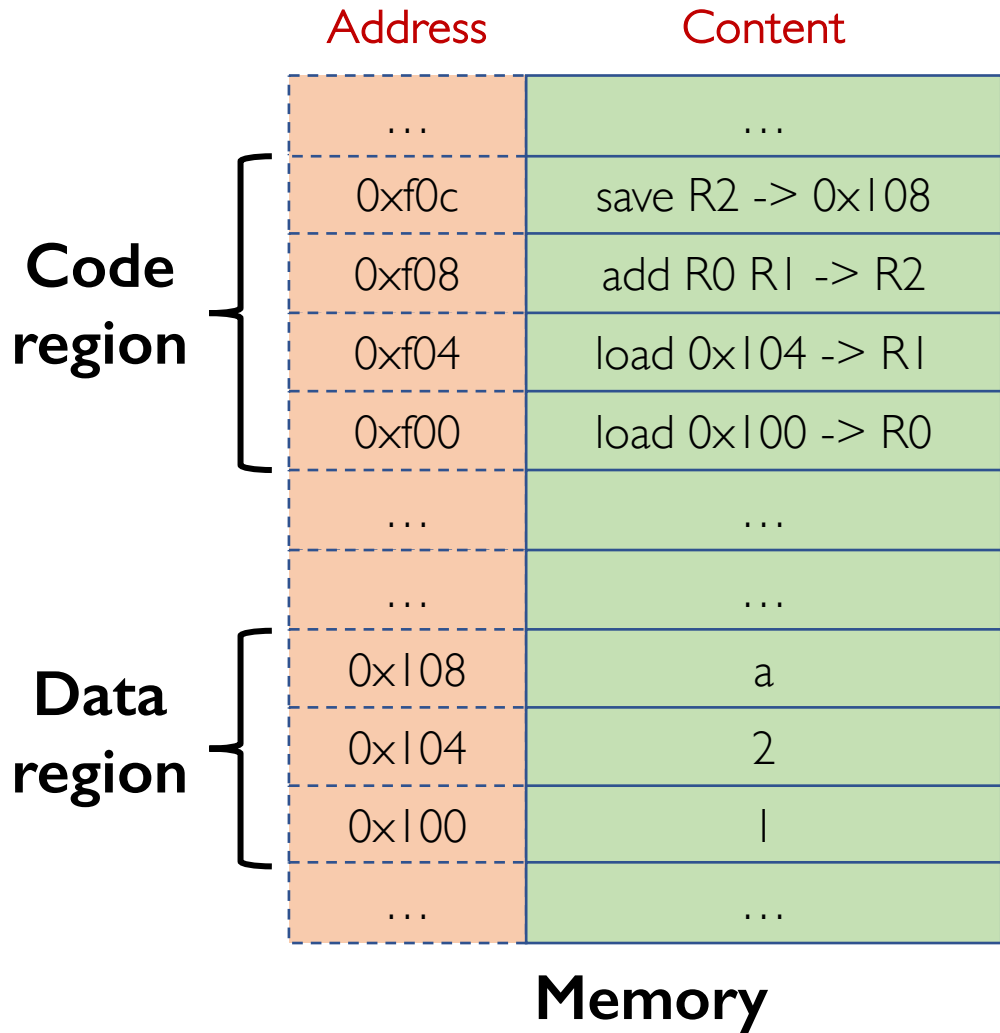
- In x86-64 CPUs, there are 16 64-bit general-purpose registers and many other special purpose registers

| 64-bit | 32-bit | 16-bit | 8-bit (low) |
|--------|--------|--------|-------------|
| RAX | EAX | AX | AL |
| RBX | EBX | BX | BL |
| RCX | ECX | CX | CL |
| RDX | EDX | DX | DL |
| RSI | ESI | SI | SIL |
| RDI | EDI | DI | DIL |
| RBP | EBP | BP | BPL |
| RSP | ESP | SP | SPL |
| R8 | R8D | R8W | R8B |
| R9 | R9D | R9W | R9B |
| R10 | R10D | R10W | R10B |
| R11 | R11D | R11W | R11B |
| R12 | R12D | R12W | R12B |
| R13 | R13D | R13W | R13B |
| R14 | R14D | R14W | R14B |
| R15 | R15D | R15W | R15B |

- Instruction pointer (EIP/IP)
- Status register (RFLAGS)
- Segment registers (CS, SS, DS, ES, FS, GS)
- Control registers (CR0-CR15)
 - Last bit of CR0 indicates if CPU is in protected mode or real mode
 - 31st bit of CR0 indicates if paging is enabled
 - CR2 indicates the page fault address
- SIMD and FP registers (AVX..)
- Etc..
- Those special registers determine how CPU operates

CPU

- An example of “a=1+2”
 - Von Neumann architecture (冯诺依曼架构)
 - Program Counter (PC, 程序计数器)
 - Also known as instruction pointer (IP)
 - Assembly code (汇编程序)



CPU

<http://ref.x86asm.net/coder32.html>

- An example of “a=1+2”
 - Von Neumann architecture (冯诺依曼架构)
 - Program Counter (PC, 程序计数器)
 - Also known as instruction pointer (IP)
 - Assembly code (汇编程序)
- How CPU understands (decodes) instructions?
 - Opcode (操作码), Operands (操作数)
 - Instruction Set Architecture (ISA, 指令集架构)
 - How many you know?

Instruction set for a Relatively Simple CPU

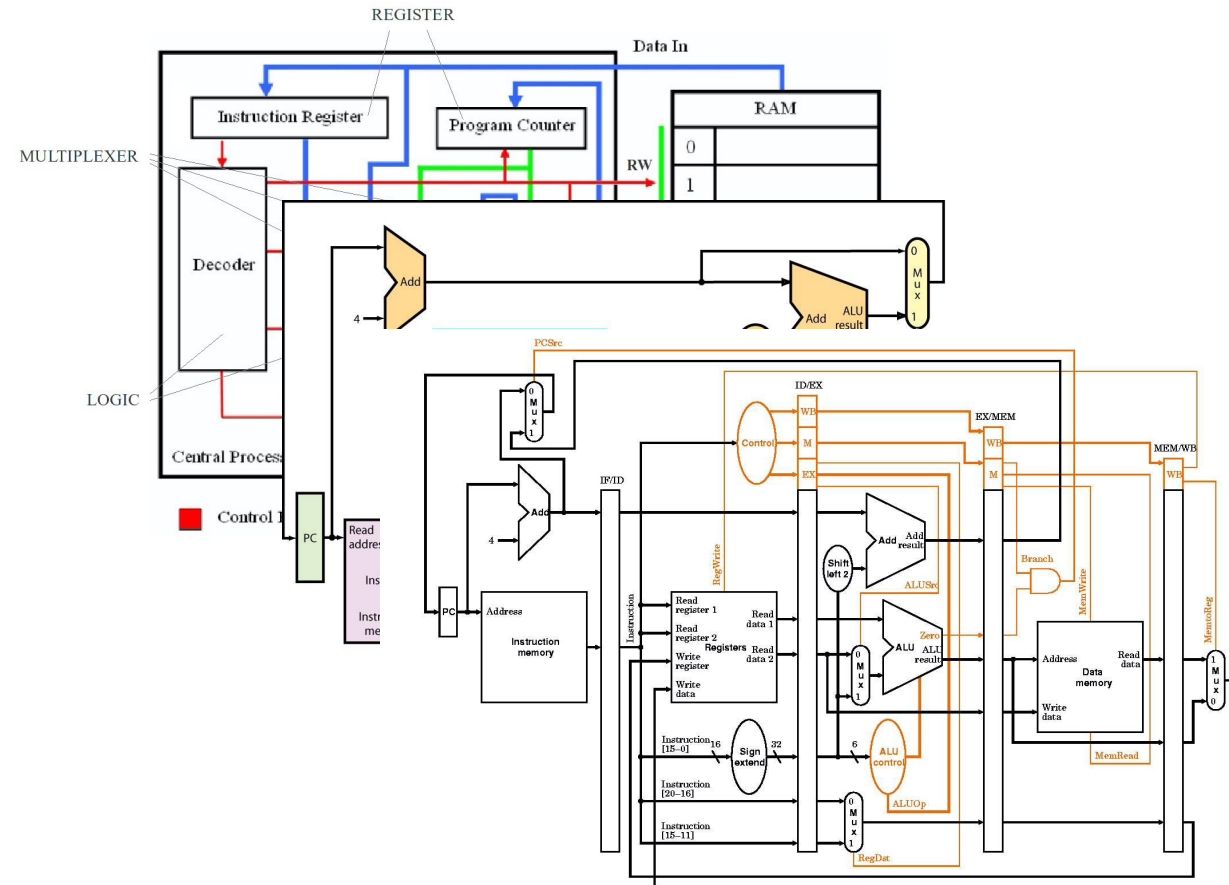
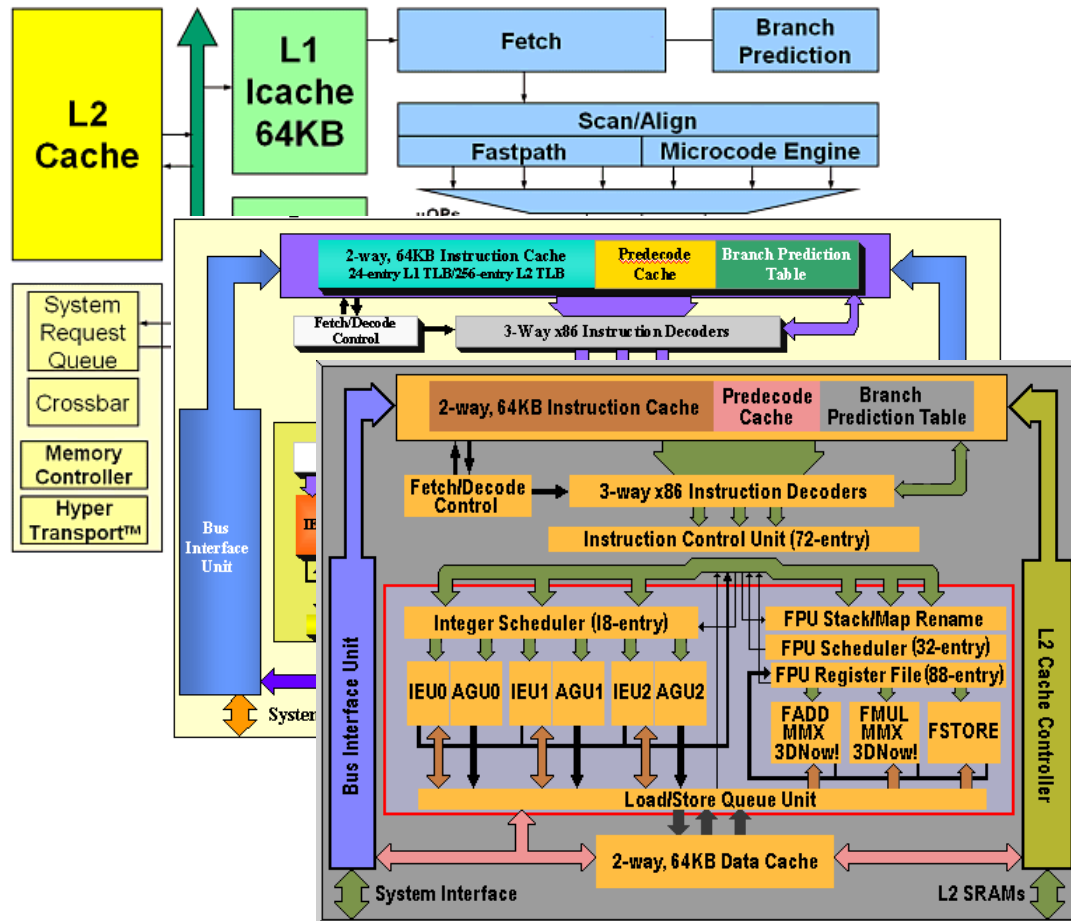
| Instruction | Instruction Code | Operation |
|-------------|--------------------|--|
| NOP | 0000 0000 | No operation |
| LDAC | 0000 0001 Γ | $AC \leftarrow M[\Gamma]$ |
| STAC | 0000 0010 Γ | $M[\Gamma] \leftarrow AC$ |
| MVAC | 0000 0011 | $R \leftarrow AC$ |
| MOVR | 0000 0100 | $AC \leftarrow R$ |
| JUMP | 0000 0101 Γ | GOTO Γ |
| JMPZ | 0000 0110 Γ | IF (Z=1) THEN GOTO Γ |
| JPNZ | 0000 0111 Γ | IF (Z=0) THEN GOTO Γ |
| ADD | 0000 1000 | $AC \leftarrow AC + R$, IF (AC + R = 0) THEN Z←1 ELSE Z←0 |
| SUB | 0000 1001 | $AC \leftarrow AC - R$, IF (AC - R = 0) THEN Z←1 ELSE Z←0 |
| INAC | 0000 1010 | $AC \leftarrow AC + 1$, IF (AC + 1 = 0) THEN Z←1 ELSE Z←0 |
| CLAC | 0000 1011 | $AC \leftarrow 0$, Z←1 |
| AND | 0000 1100 | $AC \leftarrow AC \wedge R$, IF (AC \wedge R = 0) THEN Z←1 ELSE Z←0 |
| OR | 0000 1101 | $AC \leftarrow AC \vee R$, IF (AC \vee R = 0) THEN Z←1 ELSE Z←0 |
| XOR | 0000 1110 | $AC \leftarrow AC \oplus R$, IF (AC \oplus R = 0) THEN Z←1 ELSE Z←0 |
| NOT | 0000 1111 | $AC \leftarrow AC'$, IF (AC' = 0) THEN Z←1 ELSE Z←0 |

| Opcode | Operands/Address |
|--------|------------------|
| add | R0 R1 R2 |



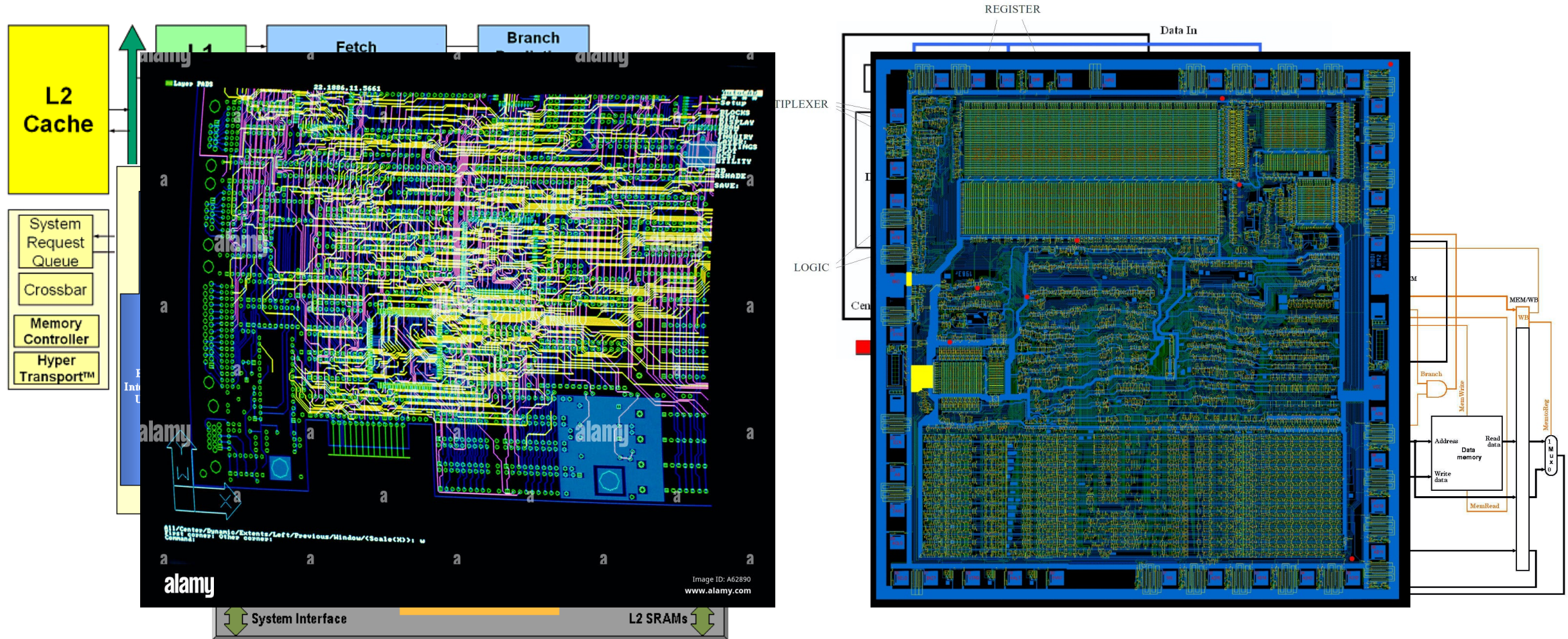
CPU

- Internally, it's about circuit and very complex



CPU

- Internally, it's about circuit and very complex



Lifetime of Hello World

1. Coded with PL and IDE

- By developers

2. Compiled to assembly

- By compilers like GCC/G++/LLVM

3. Linked to other libs

- By linkers like ld

4. Loaded to memory

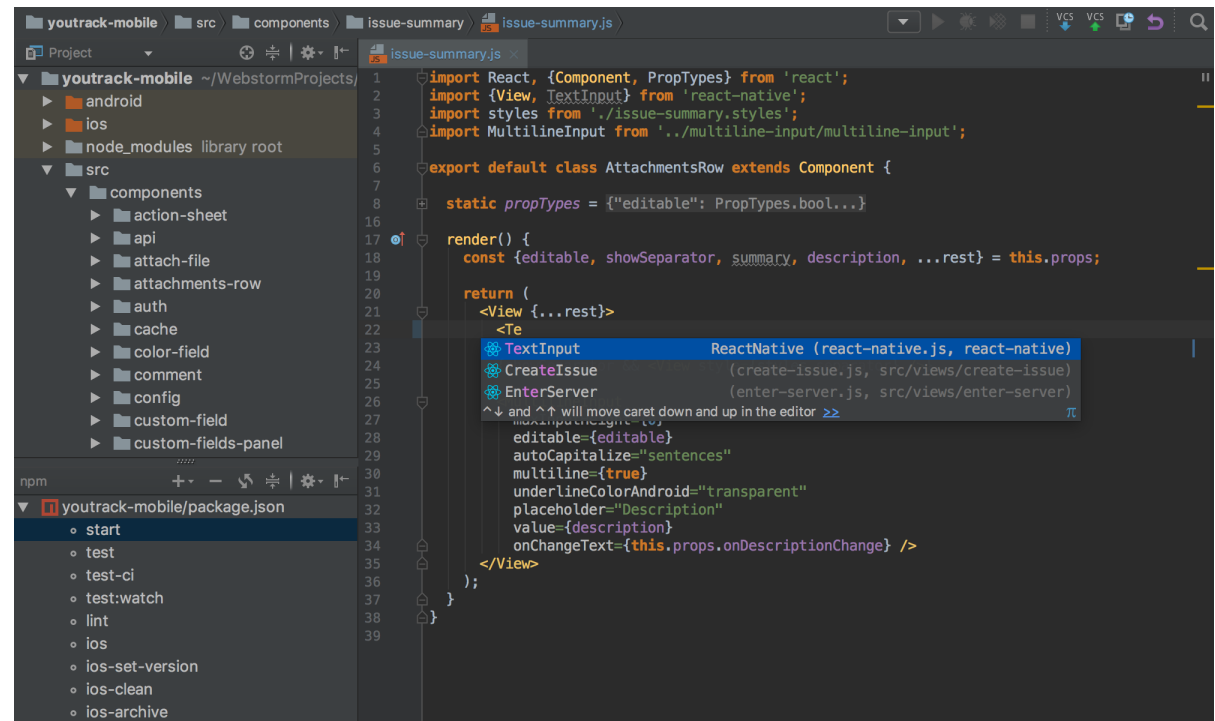
- By OS
- Allocated with memory at right position

5. Executed

- By OS
- See previous slide

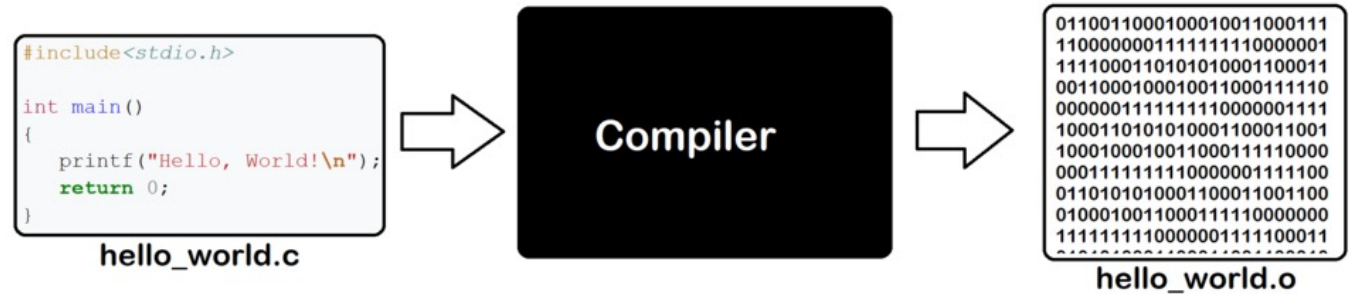
6. Terminated

- By OS with *return* command
- Memory released



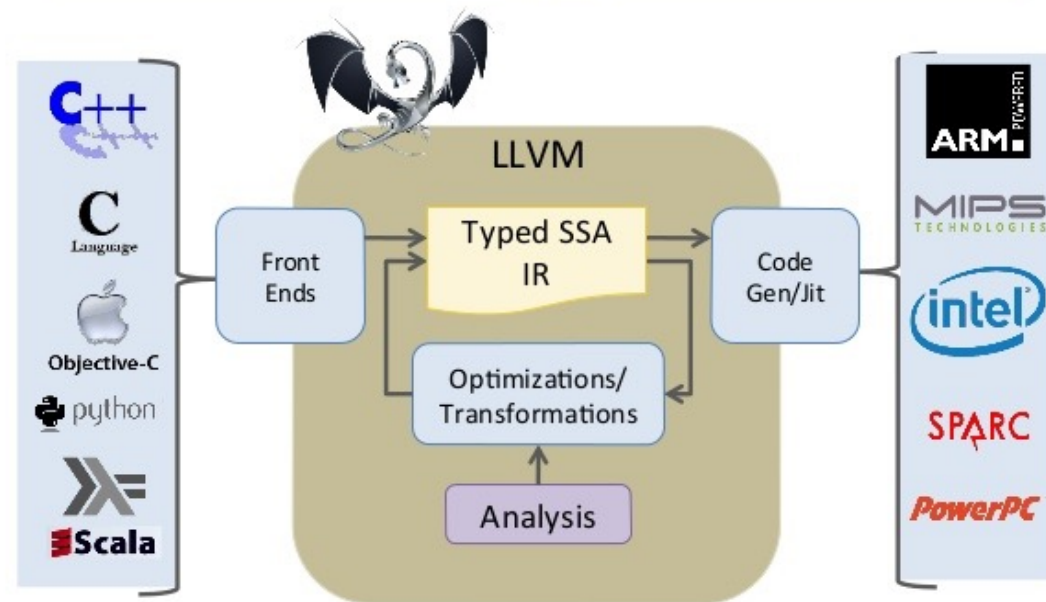
Lifetime of Hello World

1. Coded with PL and IDE
 - By developers
2. Compiled to assembly
 - By compilers like GCC/G++/LLVM
3. Linked to other libs
 - By linkers like ld
4. Loaded to memory
 - By OS
 - Allocated with memory at right position
5. Executed
 - By OS
 - See previous slide
6. Terminated
 - By OS with `return` command
 - Memory released



LLVM Compiler Infrastructure

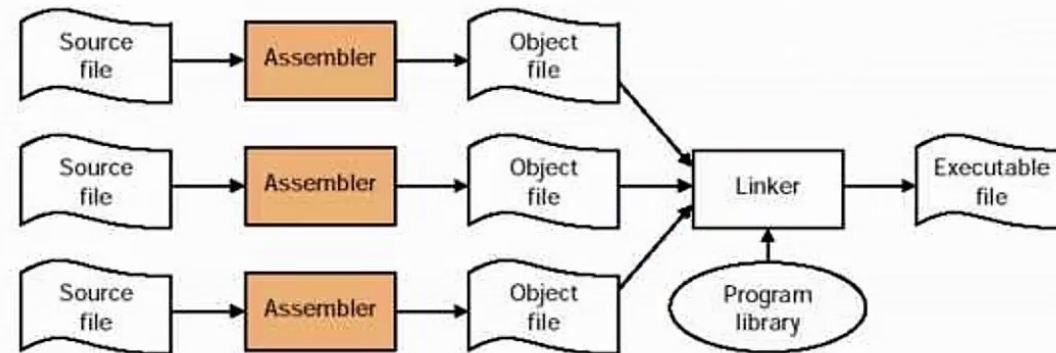
[Lattner et al.]



Lifetime of Hello World

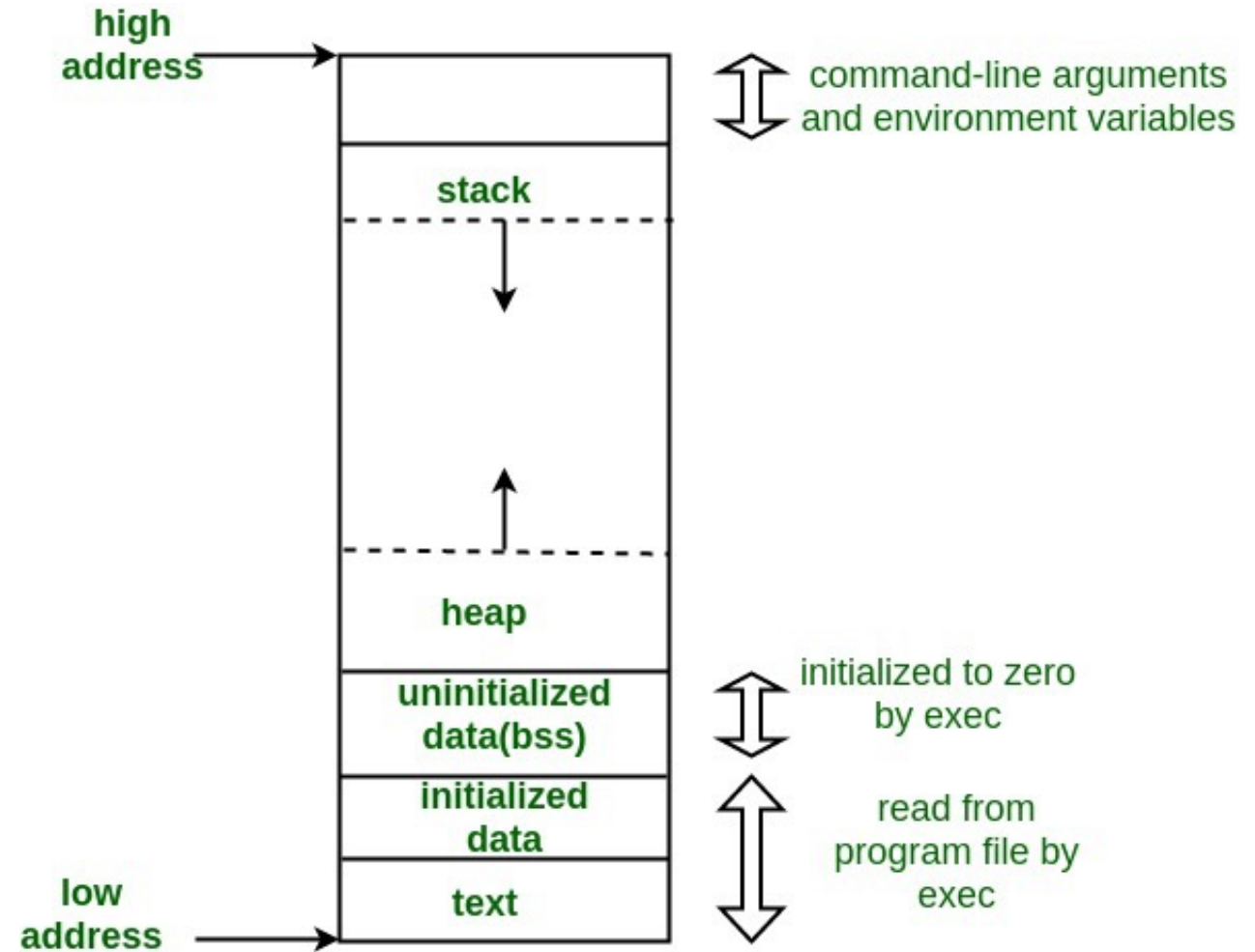
1. Coded with PL and IDE
 - By developers
2. Compiled to assembly
 - By compilers like GCC/G++/LLVM
3. **Linked to other libs**
 - **By linkers like ld**
4. Loaded to memory
 - By OS
 - Allocated with memory at right position
5. Executed
 - By OS
 - See previous slide
6. Terminated
 - By OS with *return* command
 - Memory released

Compile, assemble, and link to executable
`gcc test.c` produces `test.exe`



Lifetime of Hello World

1. Coded with PL and IDE
 - By developers
2. Compiled to assembly
 - By compilers like GCC/G++/LLVM
3. Linked to other libs
 - By linkers like ld
4. Loaded to memory
 - By OS
 - Allocated with memory at right position
5. Executed
 - By OS
 - See previous slide
6. Terminated
 - By OS with *return* command
 - Memory released



Linux Command-Line Interfaces

Linux Cheat sheet
by A.Mahouachi

1 File Commands

ls [options] file
options
-a: show hidden files
-A: show hidden files except . and ..
-d: only show directories
-h: human readable size
-i: inode info
-l: long list format
-m: output as csv
-n: numeric uid and guid
-r: sort in reverse order
-S: sort by file size
-t: sort by modification time

tree [options] dir
options
-d: only directories
-f: show full paths
-P: pattern: only matching pattern
-I: pattern: except matching pattern
-h: print sizes in human readable format
-C: use colors
-L: max: max level depth

cp [options] source dest

options
-b: backup dest before overwrite
-r: recursive
-f: force
-l: link files instead of copy
-P: dont follow sym links
-i: interactive
-u: copy only if source newer than dest

mv [options] source dest

options
-b: backup dest before overwrite
-f: force
-i: interactive
-u: move only if source newer than dest

ln [options] file link

options
-s: sym link (hard by default)
-f: overwrite link if exists
-b: backup old link before overwrite

rm [options] file

options
-f: force
-i: interactive
rm -r -foo if file name is -foo

chmod [options] mode file(s)

options
-R: recursive
symbolic mode
format: [ugoa][+|=|][perms]...
example: u+x,o-w,x,g-w
u: owner
g: group

o: others
a: all
+: add mode
-: remove mode
=: exact mode
r: read
w: write
x: execute files and search for dirs
X: search for dirs
s: setuid or setgid
t: sticky bit
numeric mode
format: [0-7]1,4
example: 755
first digit: setuid(4), setgid(2)
second digit: owner perms
third digit: group perms
fourth digit: others perms
read: 4
write: 2
execute: 1

find path [options] [tests] [actions]

options:
-mindepth: start from min level in hierarchy
-maxdepth: end with max level in hierarchy
tests:
-name "xyz*": name like xyz*
-iname "xyz*": like -name but case insensitive
-type d: only directories
-type f: only files
-mtime 0: modified < 1 day
-mtime -x: modified < x days
-mtime +x: modified > x days
-mmin: like -mtime but in minutes
-size +100M: size > 100mb
-size -100M: size < 100mb (k for kb, G for gb)
-perm /o+w: writable by others
! -perm /o+r: not readable by others

actions:
-print: print matching
-delete: rm matching files
-exec cmd '{ }' ; : run cmd for every match
-exec cmd '{ }' + : run cmd at the end of search
-exec rm -rf " : rm -rf matching items
-fprint /tmp/result: write matches to /tmp/result

diff [options] files
options
-r: recursive
-w: ignore whitespaces
-B: ignore blank lines
-q: only show file names
-x "sync*": exclude files with path like .sync*

grep [options] pattern files
options
-i: ignore case

-P: pattern is a perl regex
-m: stop after m matches
-n: also show matching line number
-R: recurse directories
-c: only show matching lines count
-exclude=glob : exclude these
-include=glob : only consider these

cat [options] file(s)

options
-v: non ascii chars except tab and eol
-T: show tabs
-t: equivalent to -vT
-E: show eol end of line
-e: equivalent to -vE
-A: equivalent to -vET
-s: remove repeat empty lines

tail [options] file

options
-f: show end of file live
-35: show last 35 lines
-q: be quiet

head [options] file

options
-35: show first 35 lines
-q: be quiet

tac file(s)

print files starting from last line
cut [options] file
options
-d char: use char as delimiter
-f 1,3,5: print fields 1, 3 and 5

uniq [options] input output

options
-c: prefix lines by number of occurrences
-d: only print duplicate lines
-u: only print unique lines

sort [options] file

options
-n: numeric sort
-b: ignore blank lines
-f: ignore case
-r: reverse order

tar [options] file

options
-f file: archive file
-c: create
-t: list
-x: extract
-C DIR: cd to DIR
-z: gzip
-j: bzip2

du [options] file

options:
-c: a grand total
-h: human readable
-L: dereference sym links
-P: no dereference of sym links
-s: total for each argument
-exclude=pattern
-max-depth=N: dont go deeper than N

df [options] file

options:
-h: human readable
-i: list inodes info
-P: no dereference of sym links

2 Process Commands

ps [options]

options:
-e: all processes
-f: full listing
-H: show hierarchy
-p pid: this process pid
-C cmd: this command name cmd
-w: wide output
-ww: to show long command lines
-l: long listing, including wchan
-o x,y,z: show columns x y z
-o user,pid,cmd: show columns user, pid command
-N: negation
-u user: processes owned by user
-u user -N: processes not owned by user
-sort=x,y: x y are columns in ps output
-sort=user: sort by user
-sort=+time: sort by cpu time asc
-sort=-time: sort by cpu time desc
-sort=size: sort by memory size
-sort=vsize: sort by vm size

top [options]

options
-d x: refresh every x seconds
-p pid1 -p pid2: only processes with pid1 pid2
-c: show command lines
interactive commands
space: update display
n: change number of displayed processes
up and down: browse processes
k: kill a process
o: change order
T: sort by time
A: sort by age
P: sort by cpu
M: sort by memory
c: display/hide command line
m: display/hide memory
t: display/hide cpu
f: manage list of displayed columns
up and down: move between columns
d: display/hide the selected column
q: apply and quit the field mgmt screen

pgrep [options] pattern

options
-l: show pid and process name
-a: show pid and full command line
-n: if more than one show newest
-o: if more than one show oldest
-u uid: show only processes of uid
-c: count results

3 Network & Remote

ssh [options] user@host ["cmd1;cmd2"]

options:
-2: force protocol 2
-o StrictHostKeyChecking=no: ignore warnings due to remote host key change
-X: forward X11 display

wget [options] url

options:
-b: run in background
-o file: print wget output in file
-o /dev/null: suppress wget output
-q: be quiet
-d: debug
-O file: save response to file
-c: resume file download
-S: print server headers
-T N: timeout after N seconds
-user=user: basic http auth user
-password=password: basic http auth password
-save-cookies file: save cookies to file
-load-cookies file: use file as cookies
-post-data=string
-post-file=file
-no-check-certificate: ignore ssl certificate

curl [options] url

options:
-H header: like -H "Host: st.com"
-u <user:password>: basic http auth
-s: be silent
-S: show errors if silent mode
-L: follow new location in case 301
-data "field=value": x-www-form-urlencoded query
-data-binary data: post data as is without encoding
-data-binary @filename: post filename content as is
-X method: use PUT, GET, POST etc.
-request method: use PUT, GET, POST etc.

mail [options] to-address

options:
-s subject: email with subject
-c address1,address2: cc copy
-b address1,address2: bcc copy
mail -s 'hello there' 'joe@st.com' < somefile

4 Terminal

Ctrl+C: halt current command
Ctrl+Z: pause current command
bg %1: resume paused command in background
fg %1: resume paused command in foreground
Ctrl+D: logout
Ctrl+W: remove a word from current line
Ctrl+U: remove current line
Ctrl+A: go to beginning of current line
Ctrl+E: go to end of current line

the command "help"

help gives you information about commands that comes with T-UI. click on the input area on the bottom of the screen and type "help".

```
$ help
```

```
A
about airplane alias aliasfile apps
B
...
```

"help" has also an other important feature: it works like Linux's man, therefore it tells you how to use a command. For example:

```
$ help search
```

will prints this:

Search something somewhere
Usage:
search [option] something

[option] can be:

```
-g -> (Google Search)
-p -> (Google Play Store)
-f -> (Files)
-y -> (YouTube)
```

Example:
>>search -p T-UI

< - -

- - >

Linux Command-Line Interfaces

- CLI vs GUI (graphical user interface)

Graphical User Interface (GUI)

- Most common interface
 - Windows, OS X, some versions of Linux and Unix
- Frees you from memorizing and typing text commands
- Uses a mouse to control objects
- Shortcuts open programs or documents

Command Line Interface

- Older interface
 - DOS, Linux, UNIX
- With a command-line interface, a user uses the keyboard to enter data and instructions
- User types commands at a prompt
- User must remember all commands
- Included in all GUIs

Linux Command-Line Interfaces

- Can large language models (LLMs) help?



convert the following instruction to linux command line instructions: move the file named "XX.cpp" to my home folder, change its permission to executable, and execute it with output dumped to YY.out



To accomplish the stated tasks, you can use the following Linux command line instructions:

1. **Move the file "XX.cpp" to your home folder:** Assuming you are in the directory containing "XX.cpp", you can use the `mv` command to move it to your home folder.

```
bash Copy code  
  
mv XX.cpp ~/
```

2. **Change its permission to executable:** Once the file is in your home directory, you can change its permissions to make it executable using the `chmod` command.

```
bash Copy code  
  
chmod +x ~/XX.cpp
```

3. **Execute it with output dumped to "YY.out":** You can execute the file and redirect its output to a file named "YY.out" in your home directory using the following command:

```
bash Copy code  
  
~/XX.cpp > ~/YY.out
```

Putting it all together, you can run these commands one after another in the same terminal session:

```
bash Copy code  
  
mv XX.cpp ~/  
chmod +x ~/XX.cpp  
~/XX.cpp > ~/YY.out
```

This sequence of commands will move the file "XX.cpp" to your home folder, make it executable, and then execute it, redirecting the output to a file named "YY.out" in your home directory.

Calling Conventions

```
1. void func_A() {  
2.     int a = 1 + 2;  
3.     int c = func_B(a);  
4.     print(c);  
5. }
```

```
1. void func_B(int x) {  
2.     int y = x + 3;  
3.     return y;  
4. }
```

- How does func_A goes to func_B?
- How does func_B return correctly back to func_A?

} stack

Search for “calling conventions” and try to understand what happens at assembly/instruction level



Misc

- Many OS features need architecture support
- Byte, bit, word
- C/C++ pointers

Homework

- The first homework – using large language models (LLMs) to build a command line tool helper
 - DDL: two weeks before the last course
 - Local LLMs (LLaMA 2) or remote (ChatGPT or 文心一言), you chose
 - Add if-else if needed
 - Chinese support is cool and useful!
 - Test with a clean environment (VM, docker) so it won't mess up your computers
- Submission: code + detailed documents + testing results
 - The best ones will be demonstrated in the last course
 - Don't copy from other projects!