Operating Systems
Lecture 1

Why Learn Operating System?
And How?

August 22, 2022

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）， 特聘副研究员，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:

李弘宇

王博琛

张博闻

About This Course

- Textbooks
  - <Operating System Concepts> (7th Edition) Abraham Silberschatz et al
  - <Operating System Principles & Practice> (2nd edition) Thomas Anderson et al
- Course time
  - 13:00-14:25, Monday
  - 8:00-9:35, Friday
- Materials
  - There will be a website that holds everything about the course
- Discipline
  - Ask a question anytime, but do not disturb others from learning.

About This Course

- Office hour: TBD, 1 hour per week
  - There will at least be 2 teacher/TAs there
- WeChat Group
  - Will send slides there after each course

About This Course - Grading

- Homework (20%)
- Project (20%)
- Final (60%)

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	• 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!

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	• 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

Ahead-of-course Questionaire

“

- 希望老师认真讲课即可，不要搞一些过于花里胡哨的作业或者任务就行了
- 最重要的一点，要有习题课，至少有时间的话期末之前带着复习一下
- 希望能够学到知识
- 希望增加例子帮助理解相关概念
- 希望老师可以进行一些简单的实践操作 并多进行一些指导
- 顺利完成课程学时，深刻学到操作系统相关知识
- 可以多从代码中讲解理论
- 这课看着就难，希望简单点
- 希望设计实践的时间不要太早，也不要太晚以防期末压力太大
- 给分高点
- 希望能学有所用
- 希望老师可以让想要多学的同学有事可做，同时能力较弱的同学能跟上课程详细学习以备考研
- 希望能够结合实践来学习

”

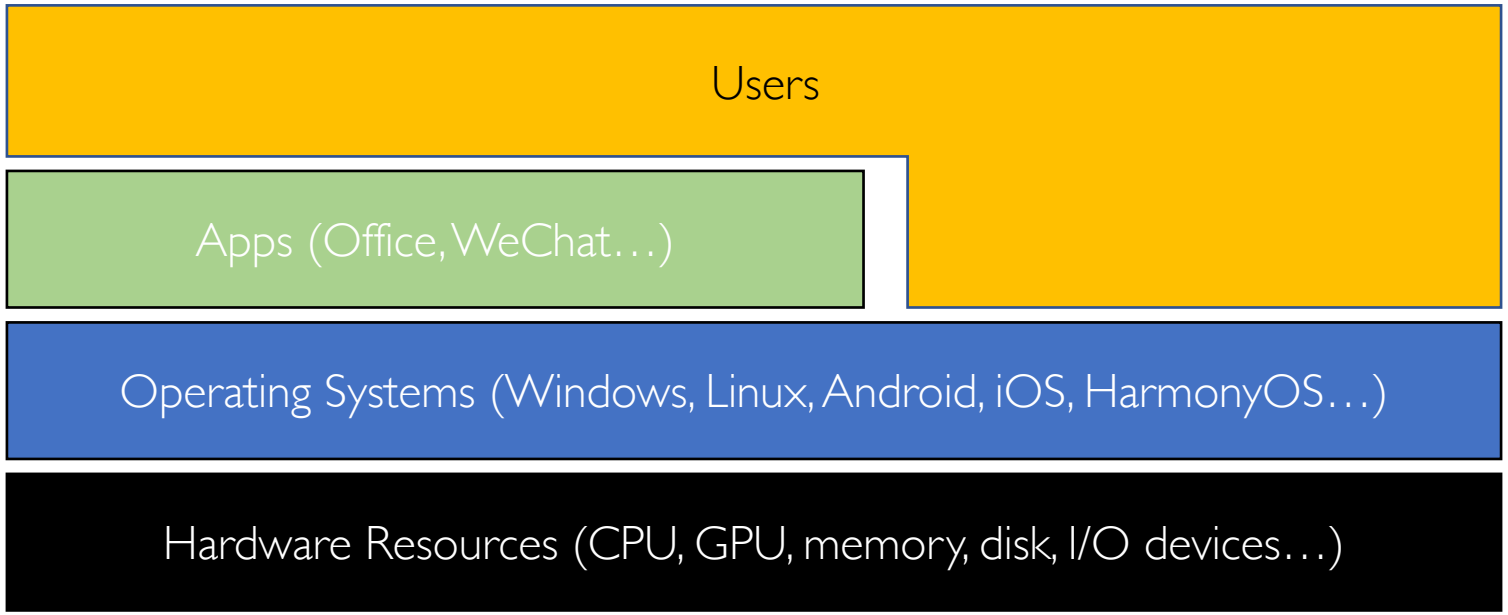
Goals for Today

- 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

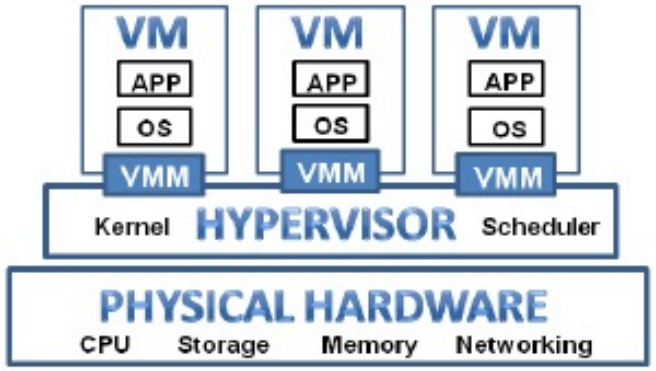
- Resource allocation
  - Multitasks on constrained resources
  - What if there is an infinite loop?
- Isolation
  - Fault in one app shall not disrupt others
  - Prevent malicious attackers
- Communication
  - Reliable, safe, and fast

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

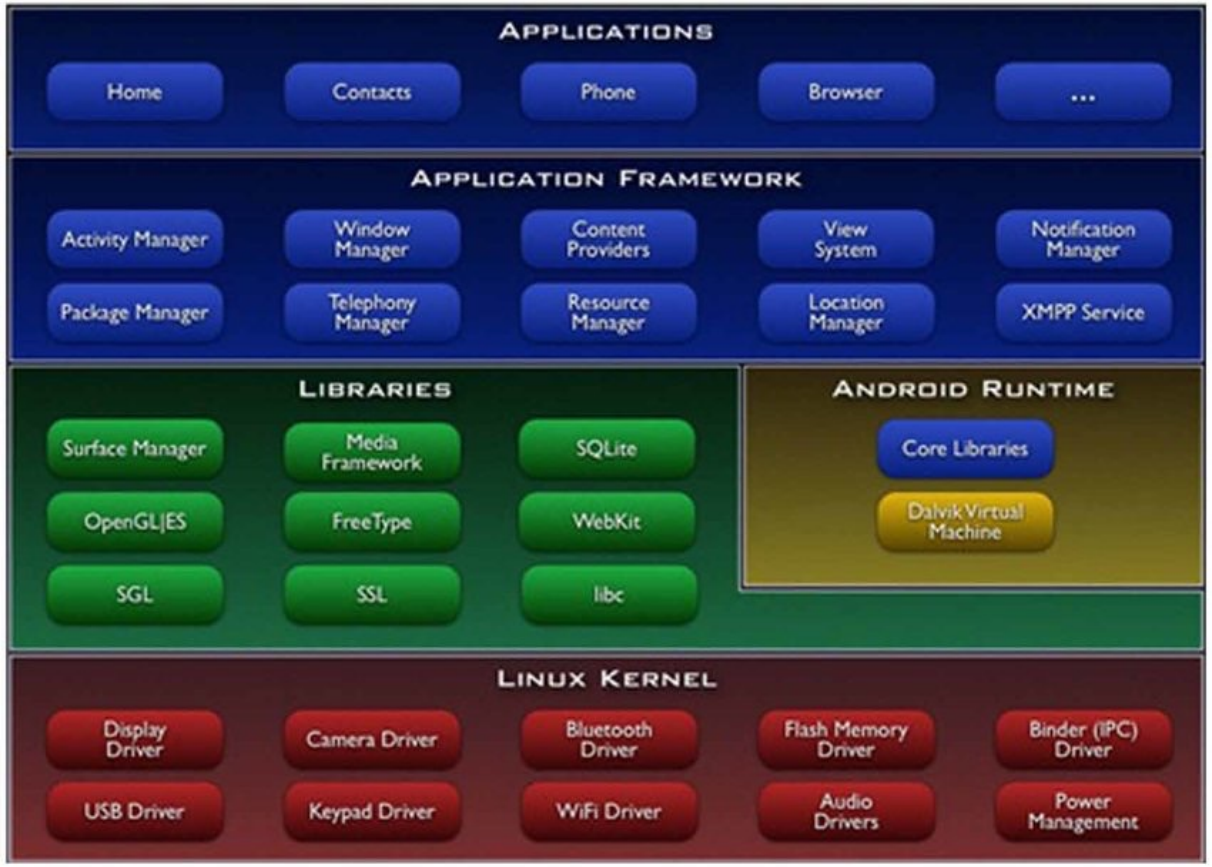
- Providing common services to facilitate resource sharing
  - read/write files
  - share memory
  - pass messages
  - UI
- Decouple HW and app development
  - As an interoperability layer for portability
  - Most applications can evolve independently with OS, unless those require very low-level programming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

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

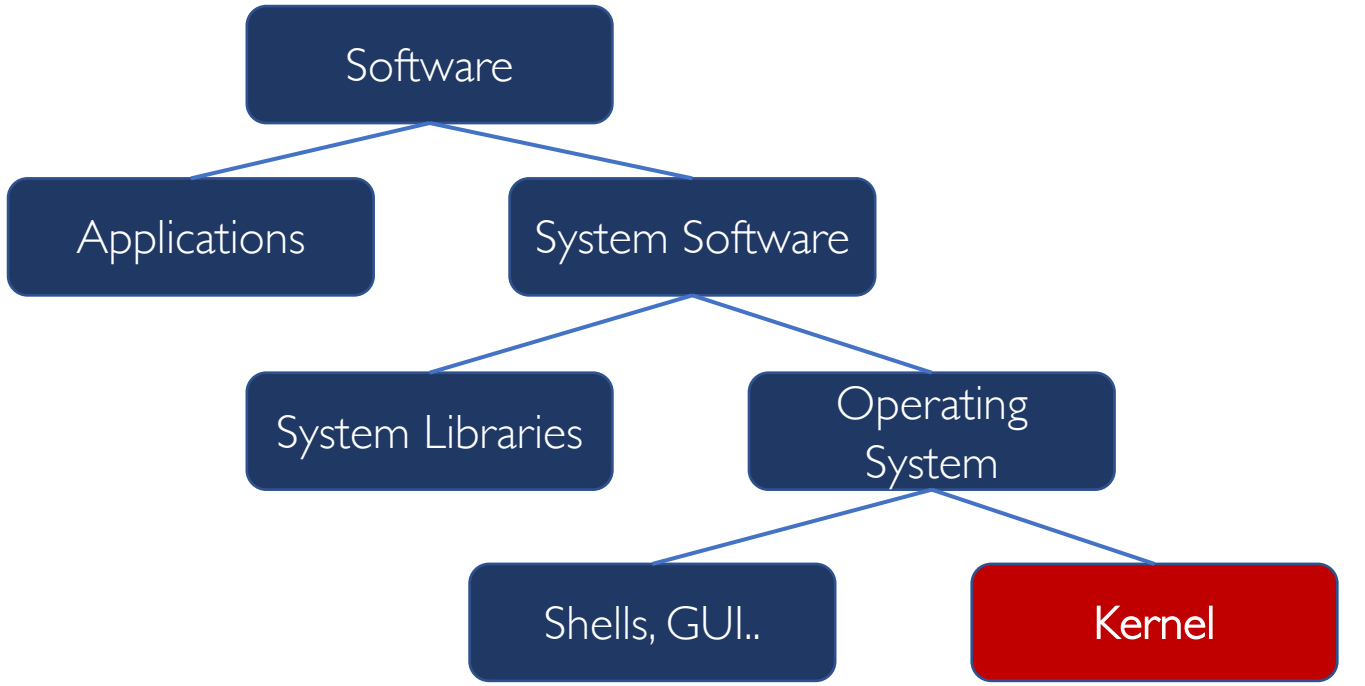
- Android/TensorFlow/WeChat
- OS vs. kernel (内核)



(The focus of this course)

The OS concept is expanding

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



(The focus of this course)

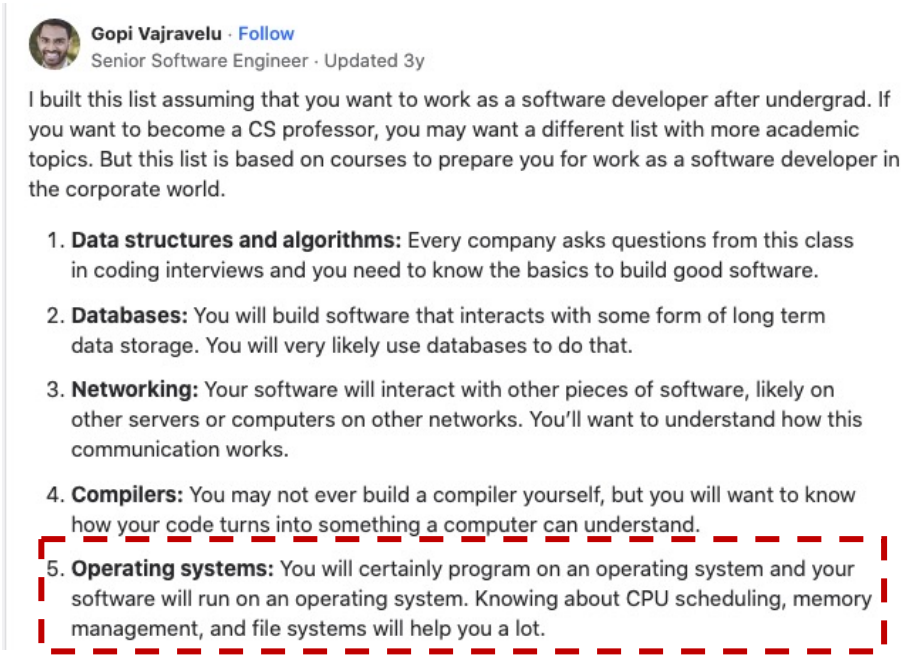
Goals for Today

- 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.



Gopi Vajravelu · Follow
Senior Software Engineer · Updated 3y

I built this list assuming that you want to work as a software developer after undergrad. If you want to become a CS professor, you may want a different list with more academic topics. But this list is based on courses to prepare you for work as a software developer in the corporate world.

1. **Data structures and algorithms:** Every company asks questions from this class in coding interviews and you need to know the basics to build good software.
2. **Databases:** You will build software that interacts with some form of long term data storage. You will very likely use databases to do that.
3. **Networking:** Your software will interact with other pieces of software, likely on other servers or computers on other networks. You'll want to understand how this communication works.
4. **Compilers:** You may not ever build a compiler yourself, but you will want to know how your code turns into something a computer can understand.
5. **Operating systems:** You will certainly program on an operating system and your software will run on an operating system. Knowing about CPU scheduling, memory management, and file systems will help you a lot.

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, ButlerW 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

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..

- 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

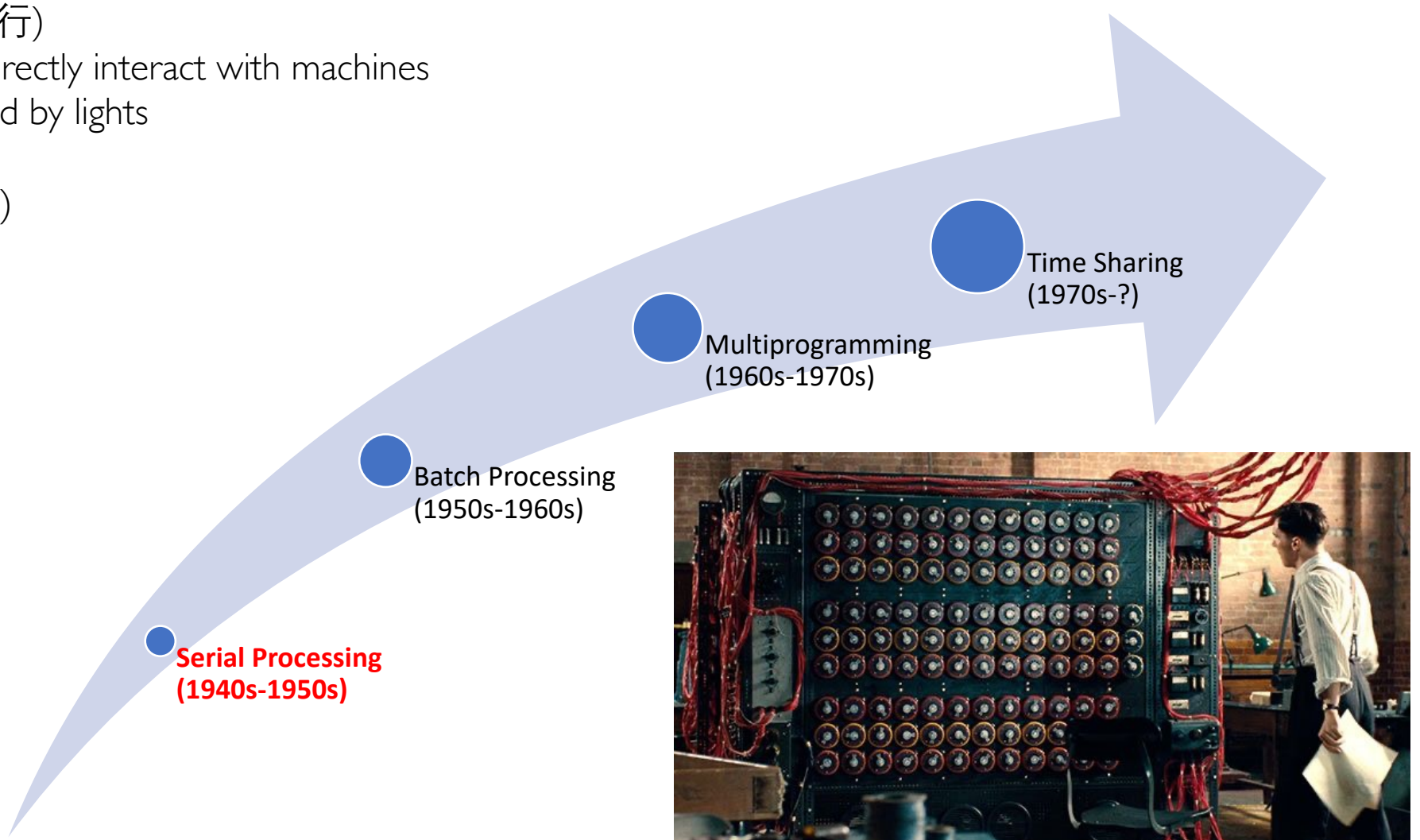
- About this course
- What is OS?
- Why learn OS?
- Challenges and how to overcome?
- Brief history of OS

- 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

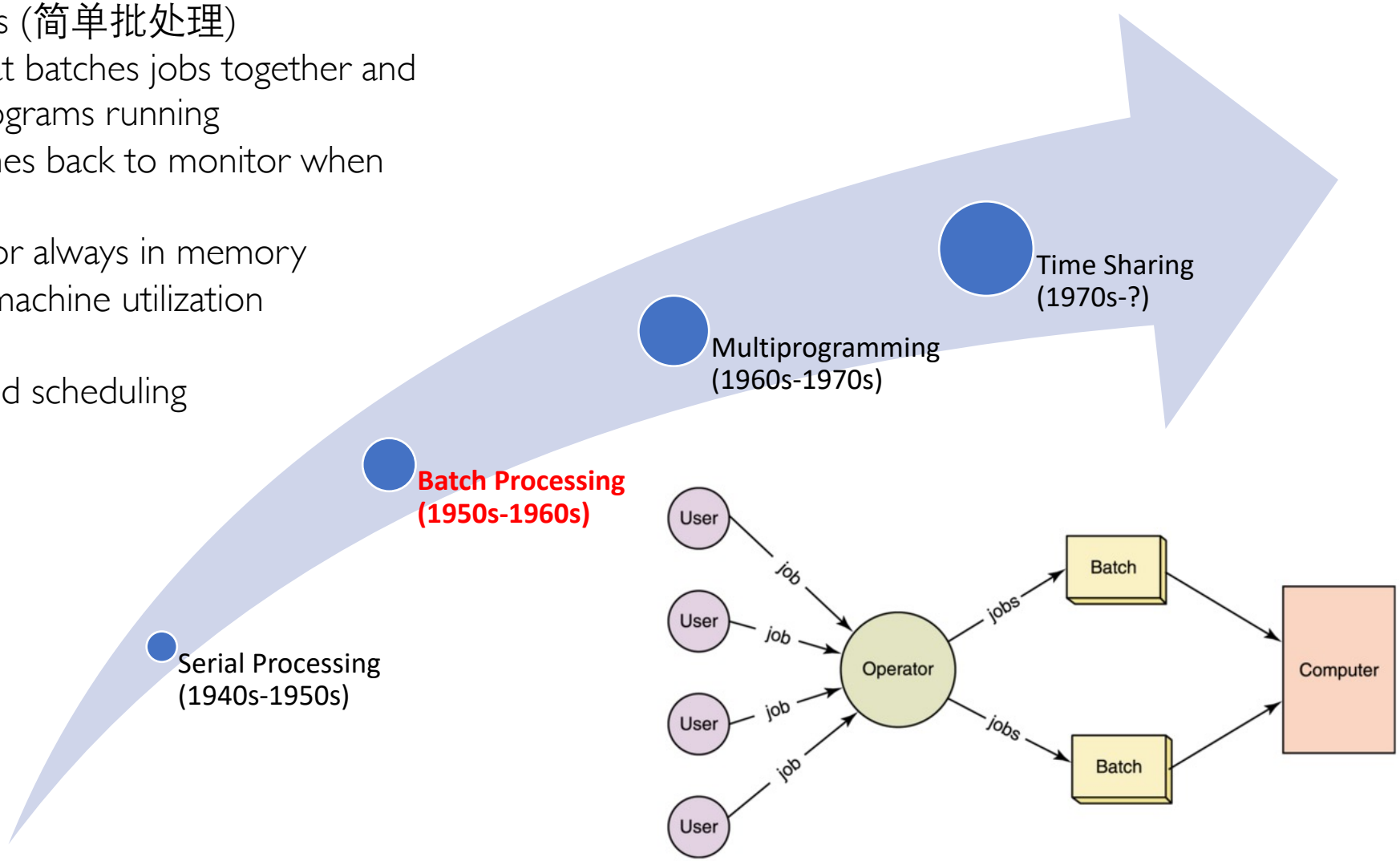
- No scheduling (调度)
- Huge setup time



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
- Not enough for user interactions

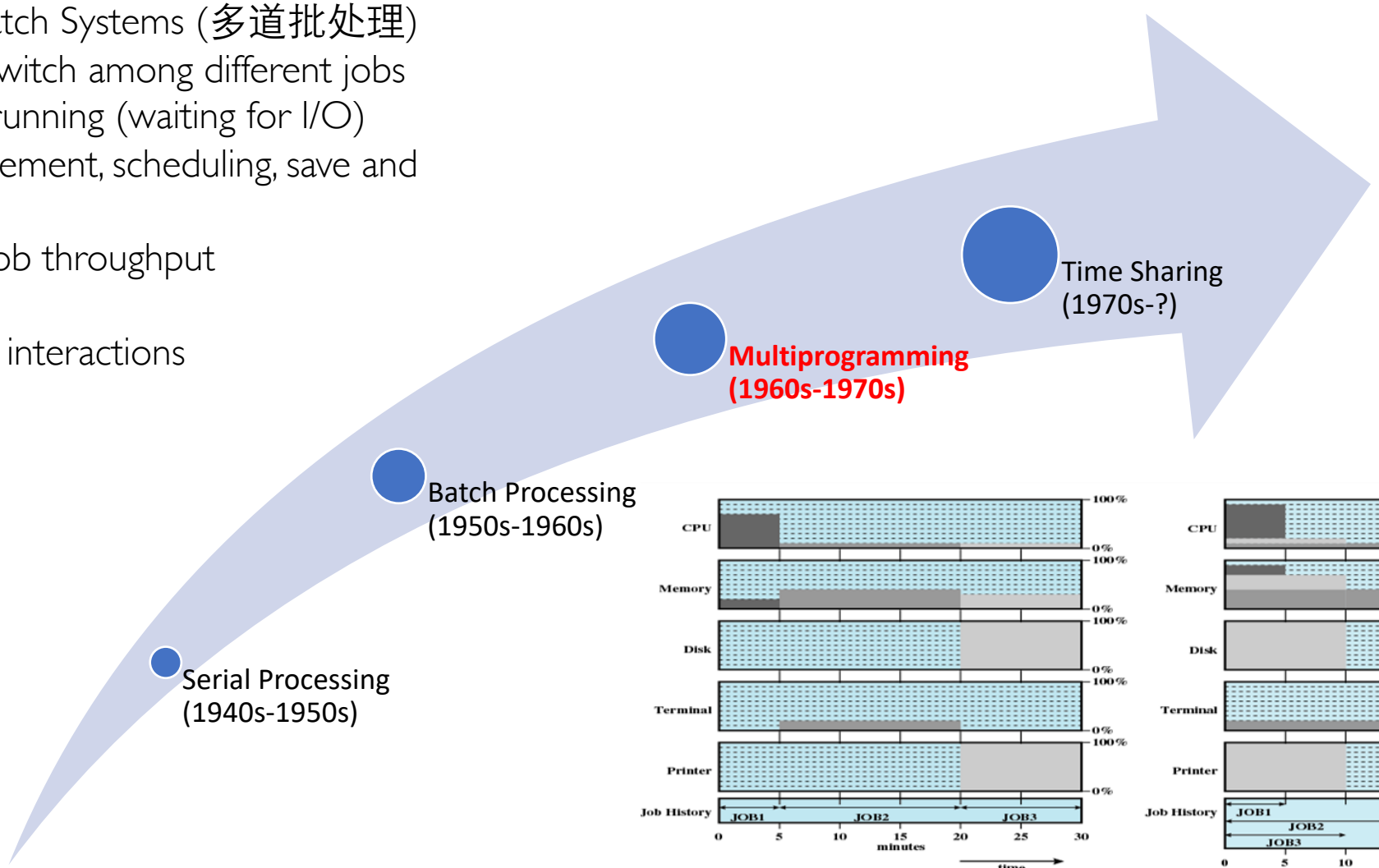
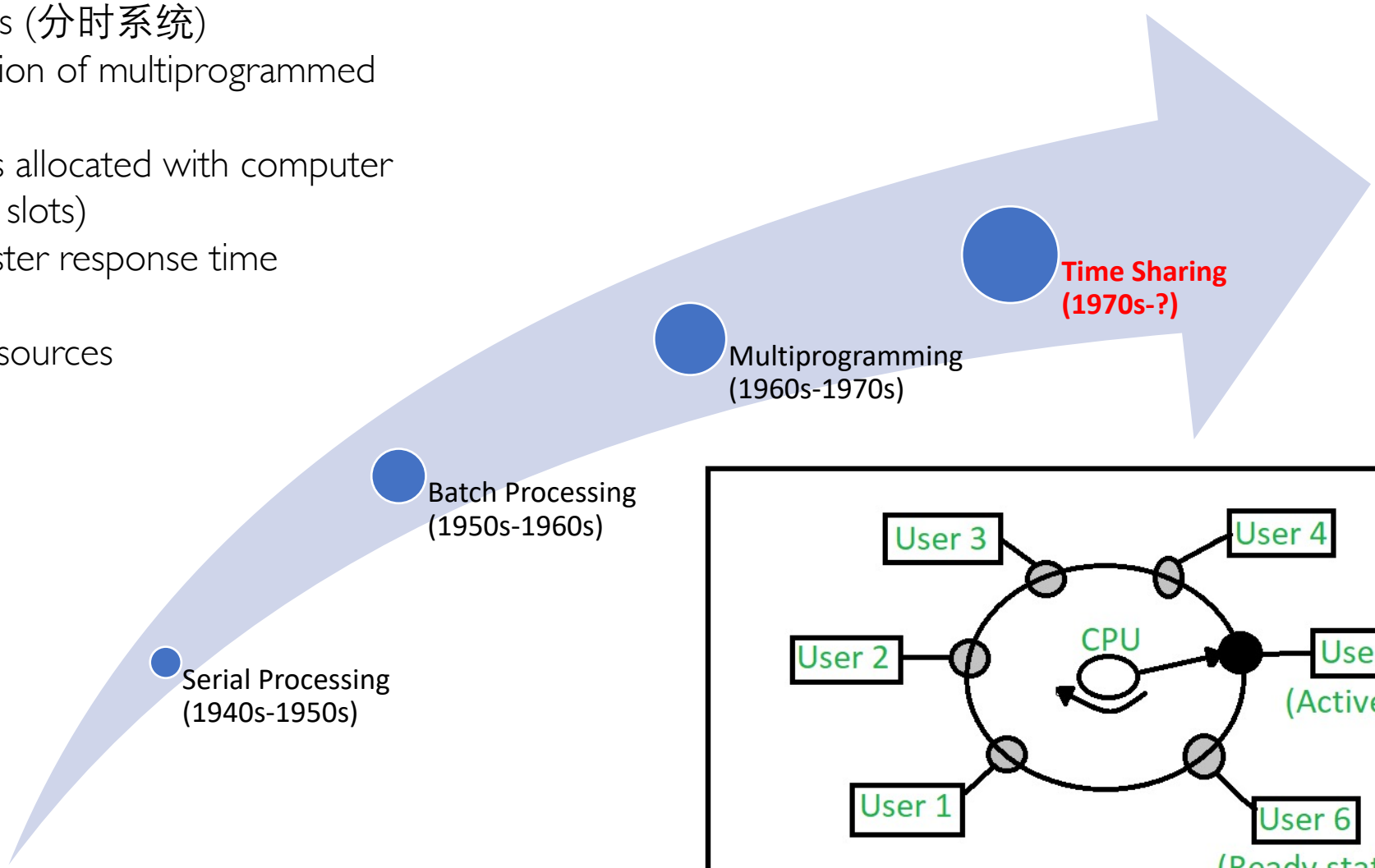


Figure 2.6 Utilization Histograms

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)

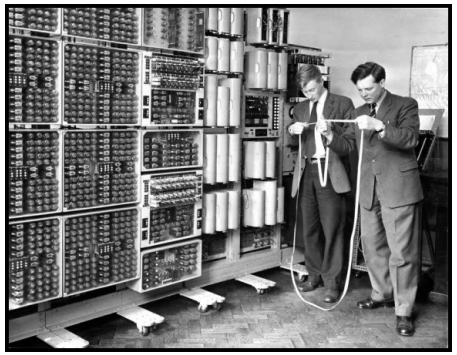
Summary of OS Evolution

- 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

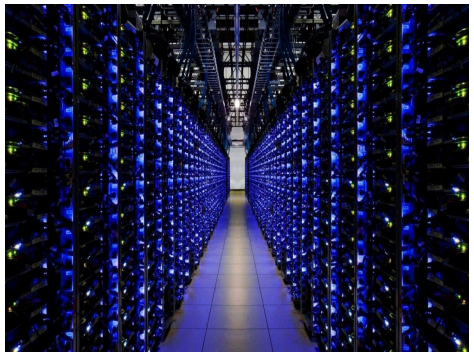
Summary of OS Evolution

- 分久必合，合久必分



Centralized

Decentralized



Centralized

Decentralized

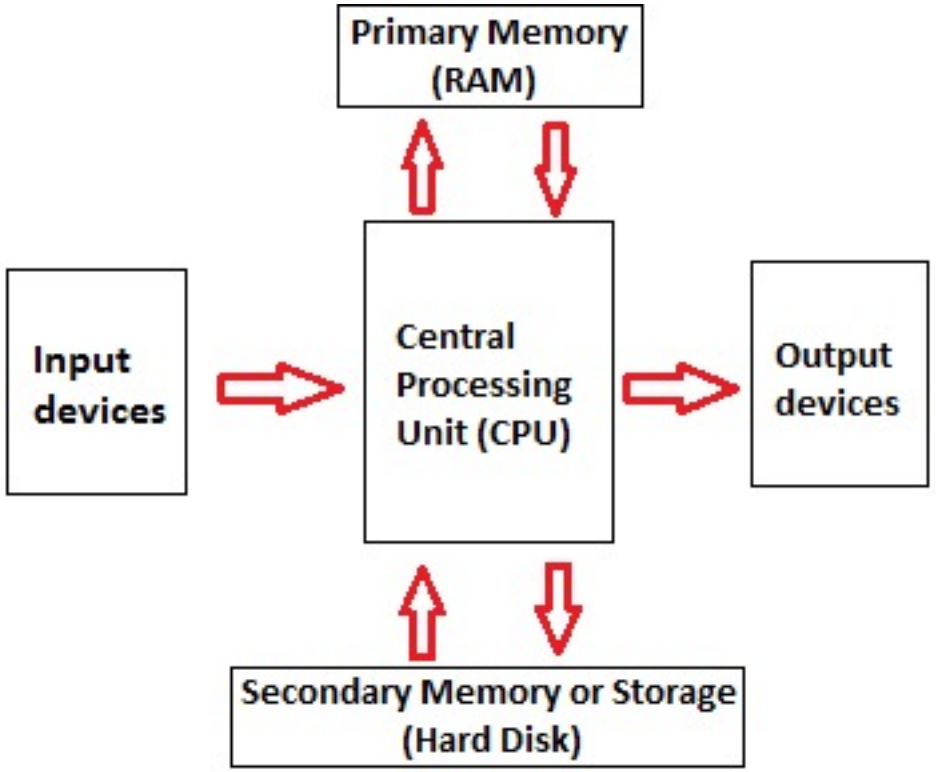
Goals for Today

- 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

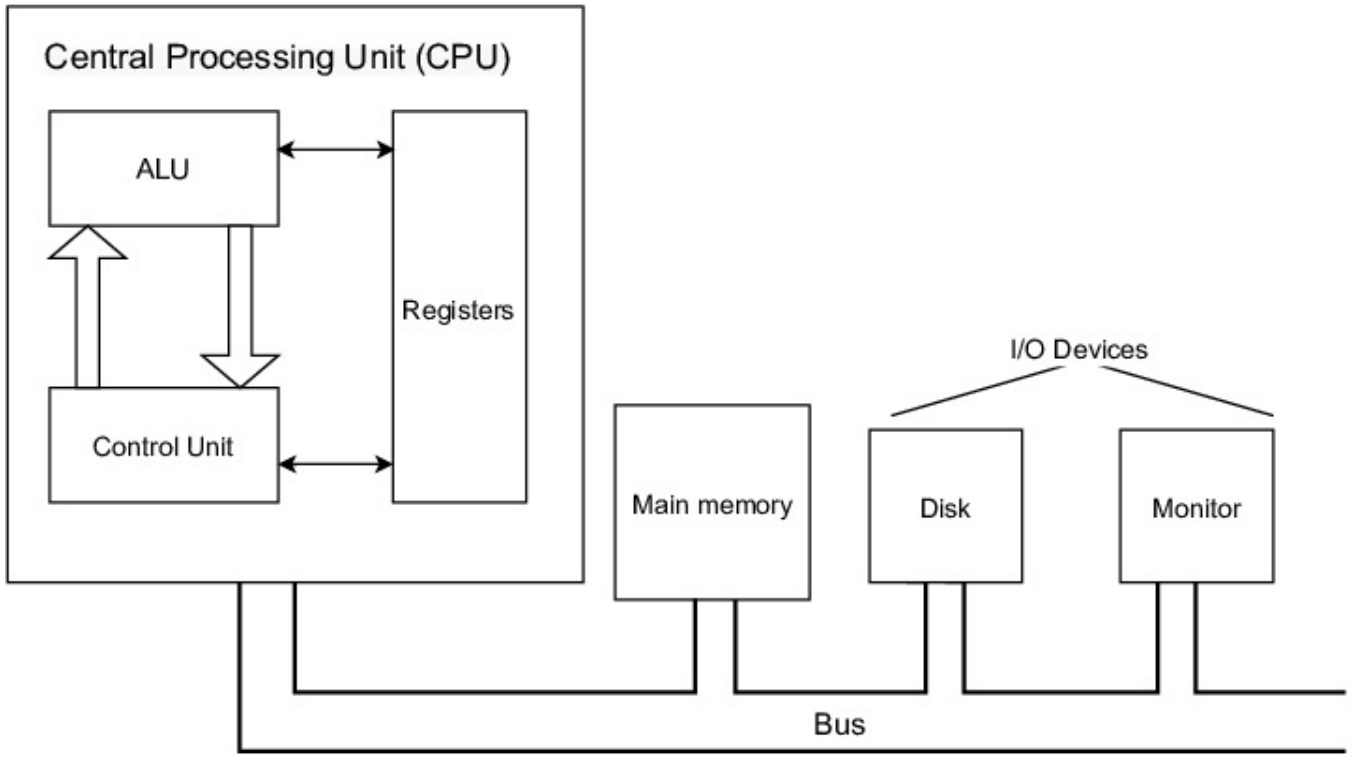
Computer Components

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



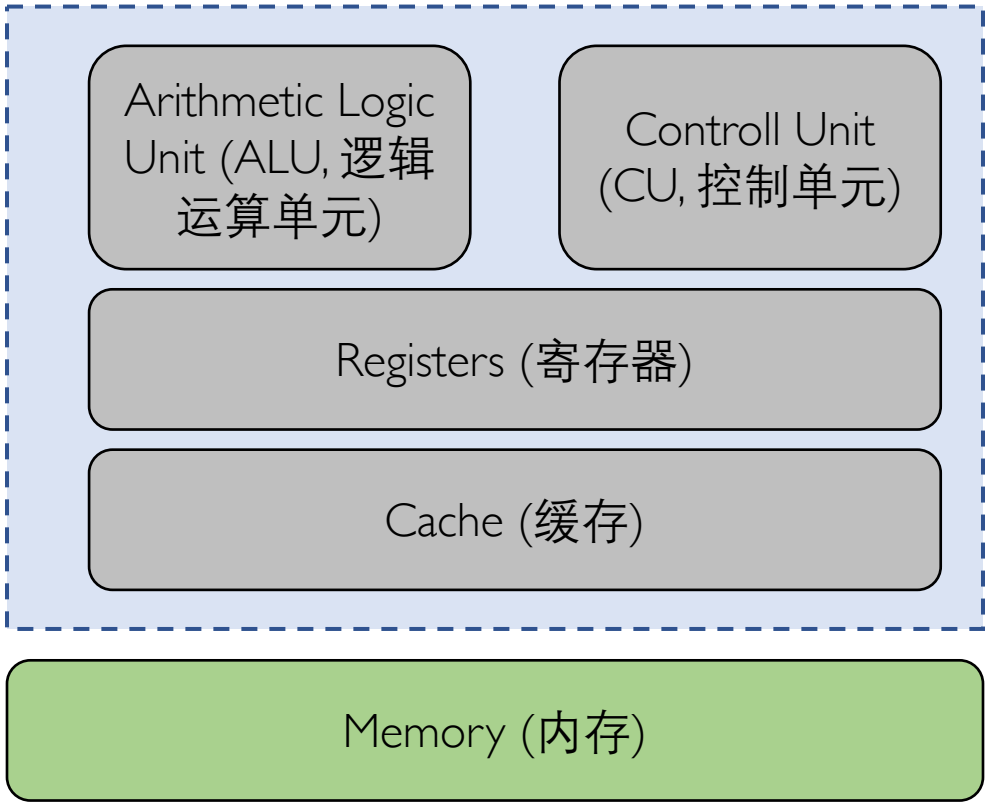
Computer Architecture

- 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

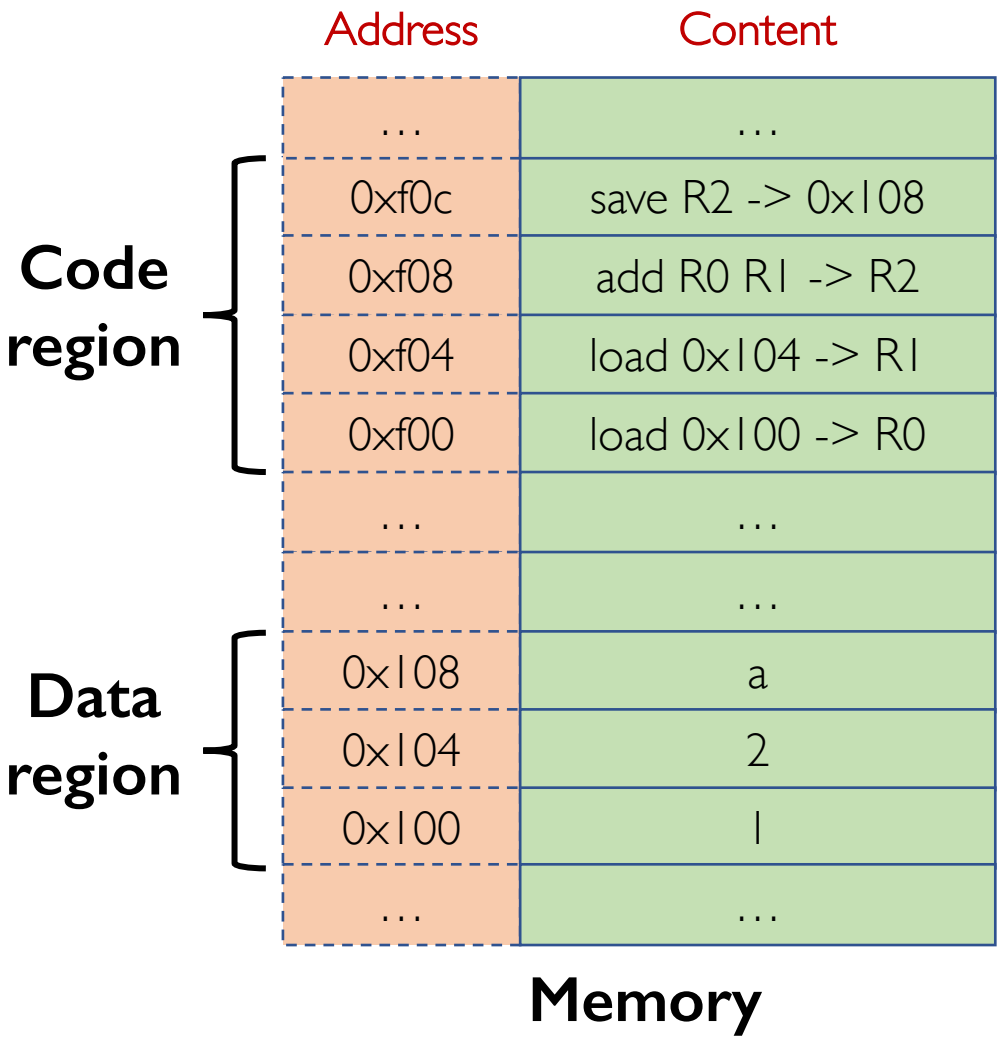
64-bit	32-bit	16-bit	8-bit (low)
RAX	EAX	AX	AL
RBX	EBX	BX	BL
RCX	ECX	CX	CL
RDX	EDX	DX	DL
RSI	ESI	SI	SIL
RDI	EDI	DI	DIL
RBP	EBP	BP	BPL
RSP	ESP	SP	SPL
R8	R8D	R8W	R8B
R9	R9D	R9W	R9B
R10	R10D	R10W	R10B
R11	R11D	R11W	R11B
R12	R12D	R12W	R12B
R13	R13D	R13W	R13B
R14	R14D	R14W	R14B
R15	R15D	R15W	R15B

- 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

- 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)

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

Instruction set for a Relatively Simple CPU

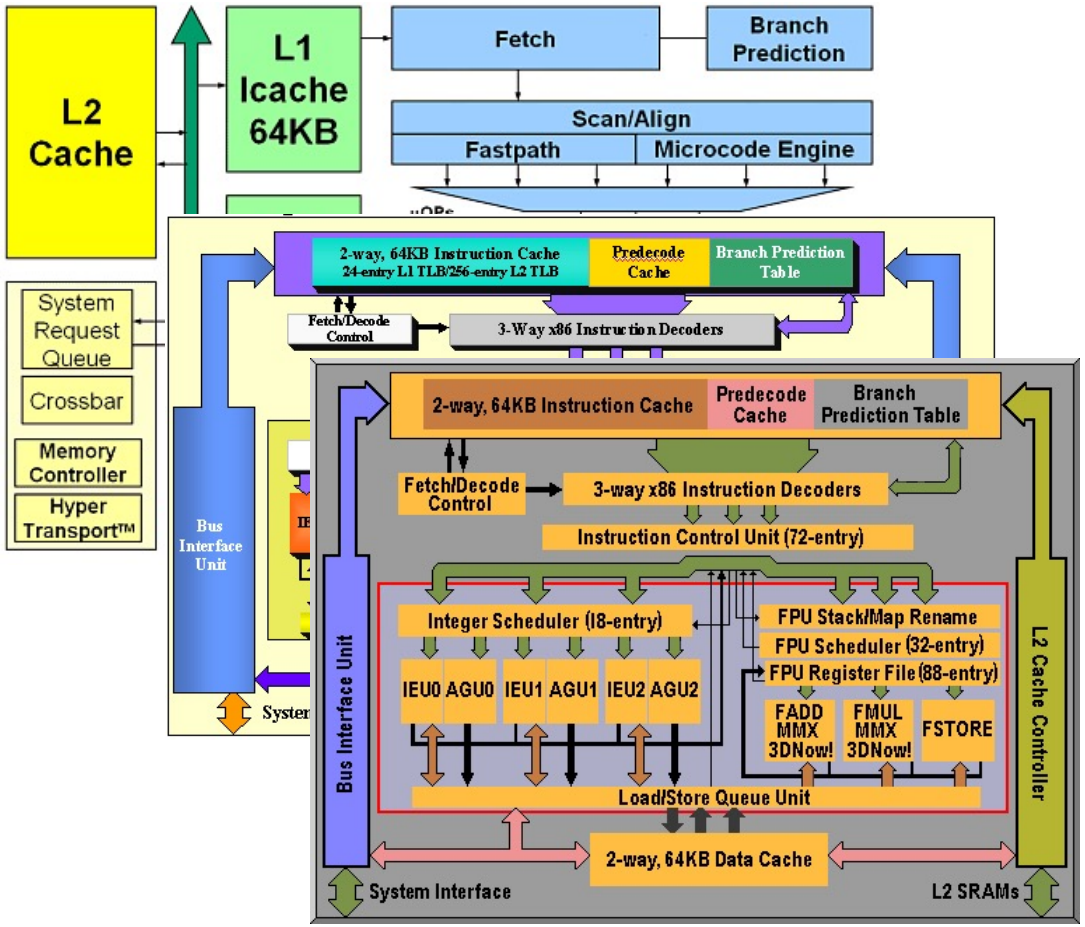
Instruction	Instruction Code	Operation
NOP	0000 0000	No operation
LDAC	0000 0001 Γ	*AC*←M[Γ]
STAC	0000 0010 Γ	M[Γ]←*AC*
MVAC	0000 0011	*R*←*AC*
MOVR	0000 0100	*AC*←*R*
JUMP	0000 0101 Γ	GOTO Γ
JMPZ	0000 0110 Γ	IF (Z=1) THEN GOTO Γ
JPNZ	0000 0111 Γ	IF (Z=0) THEN GOTO Γ
ADD	0000 1000	*AC*←*AC* + *R*, IF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

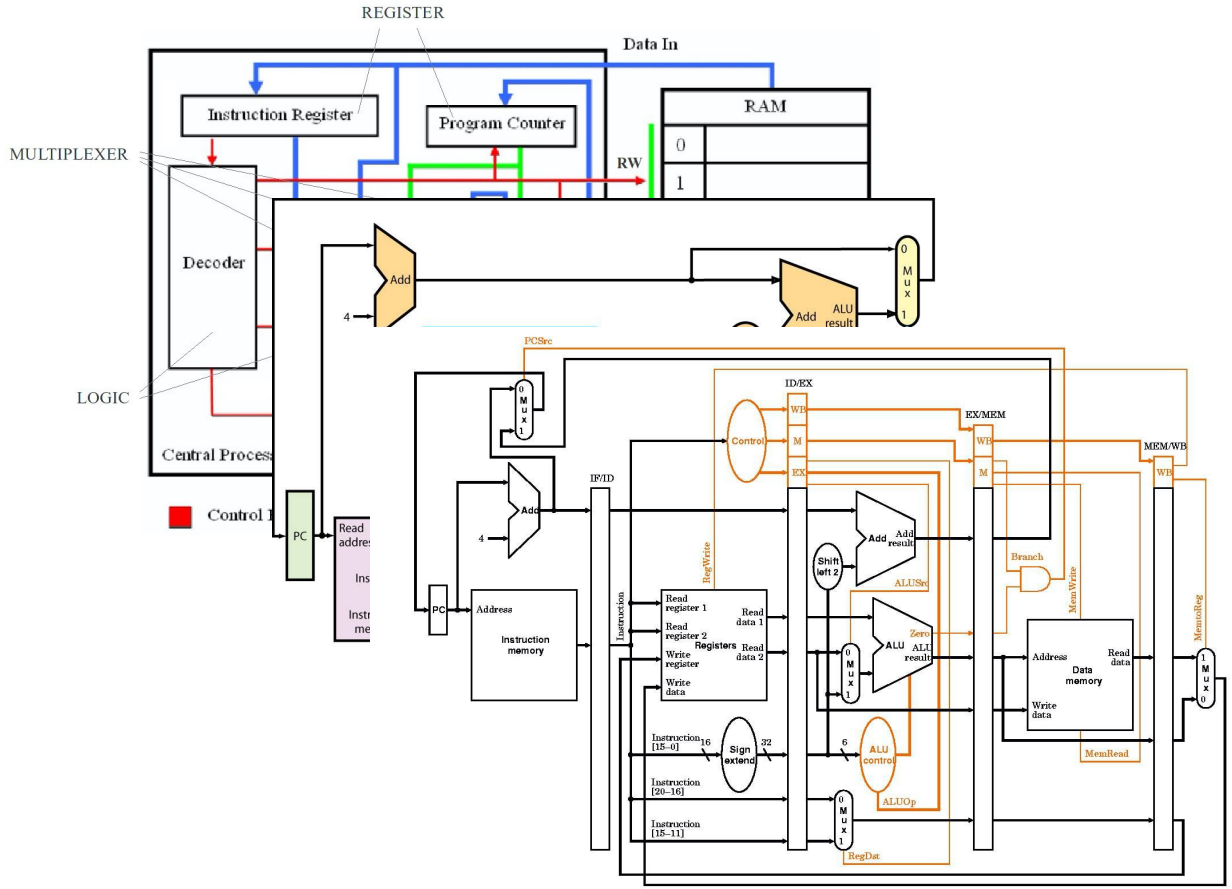
- How CPU understands (decodes) instructions?
  - Opcode (操作码), Operands (操作数)
  - Instruction Set Architecture (ISA, 指令集架构)
    - How many you know?

Opcode	Operands/Address
add	R0 R1 R2

CPU

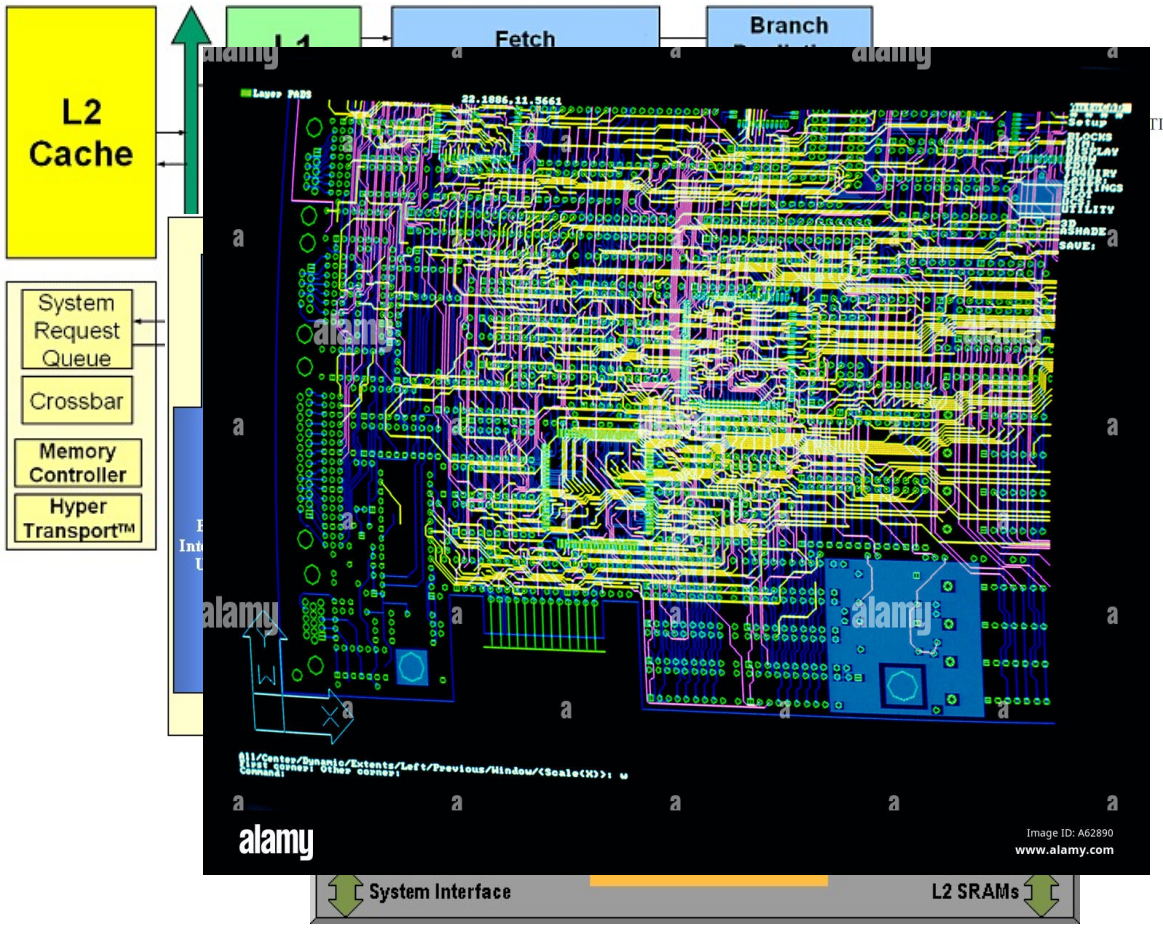
- Internally, it's about circuit and very complex

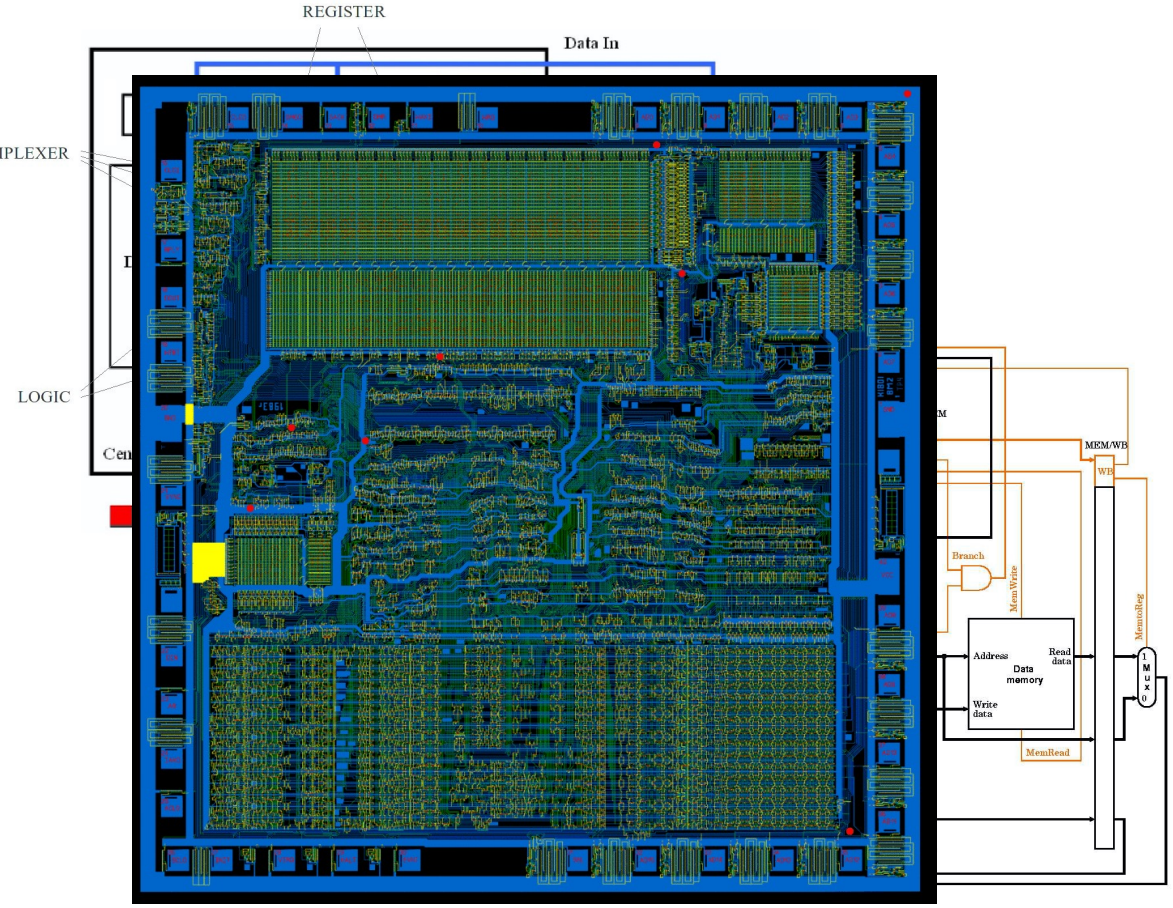




CPU

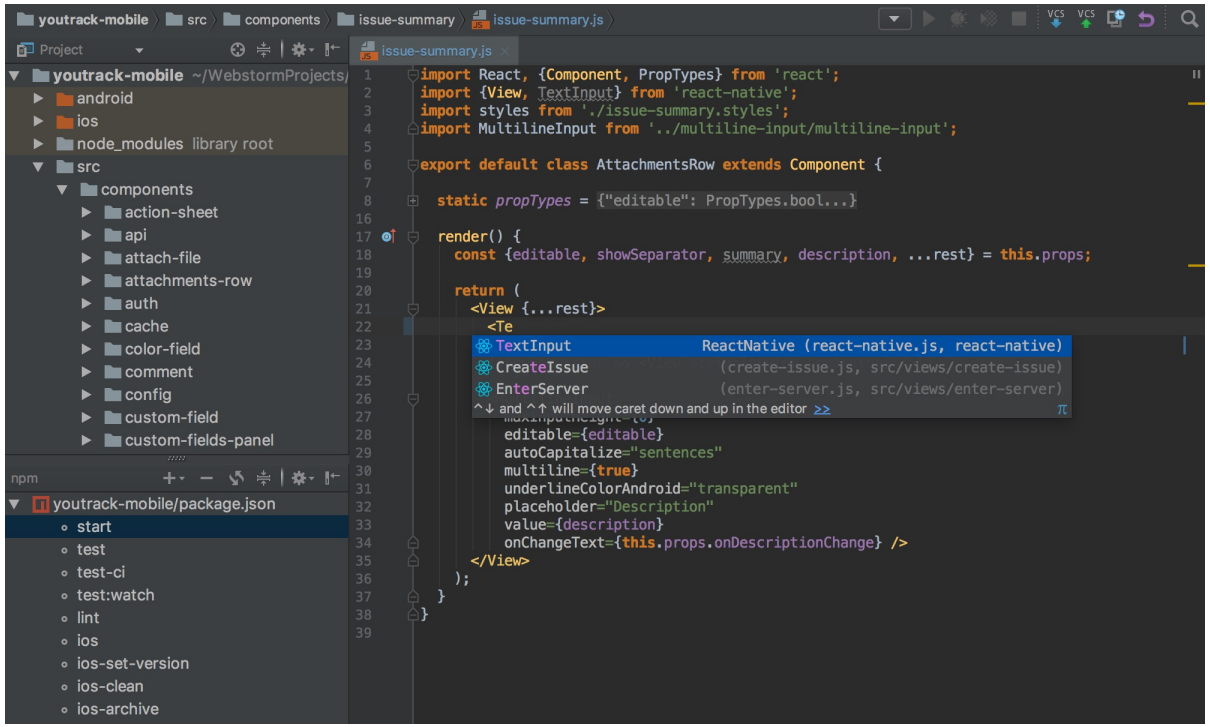
- Internally, it's about circuit and very complex





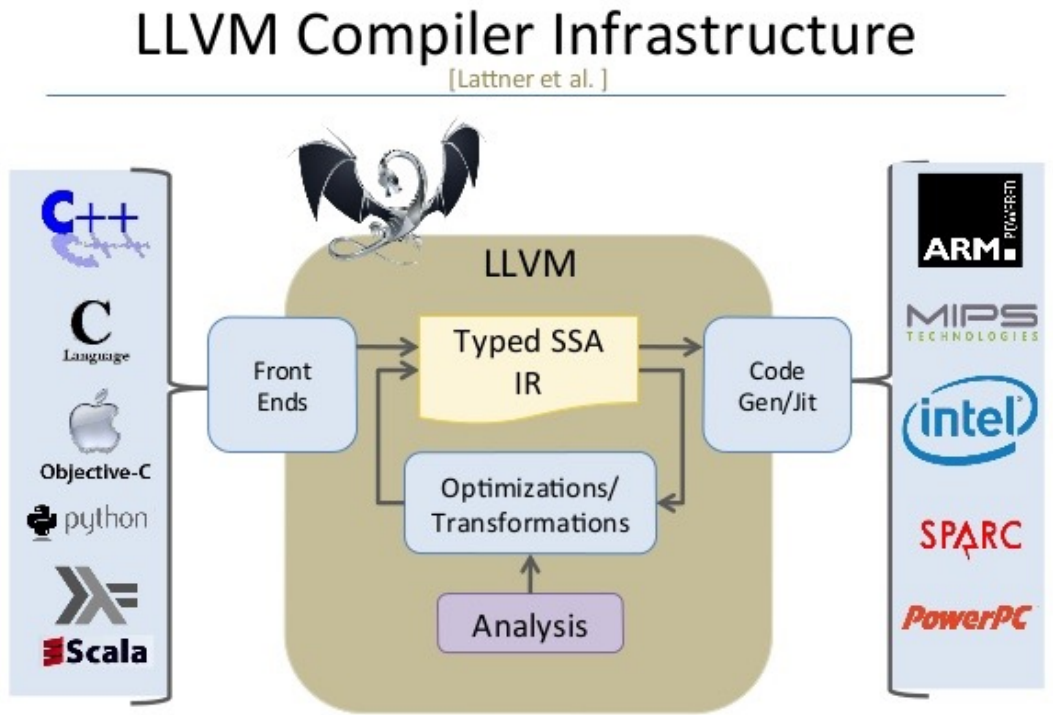
Lifetime of Hello World

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



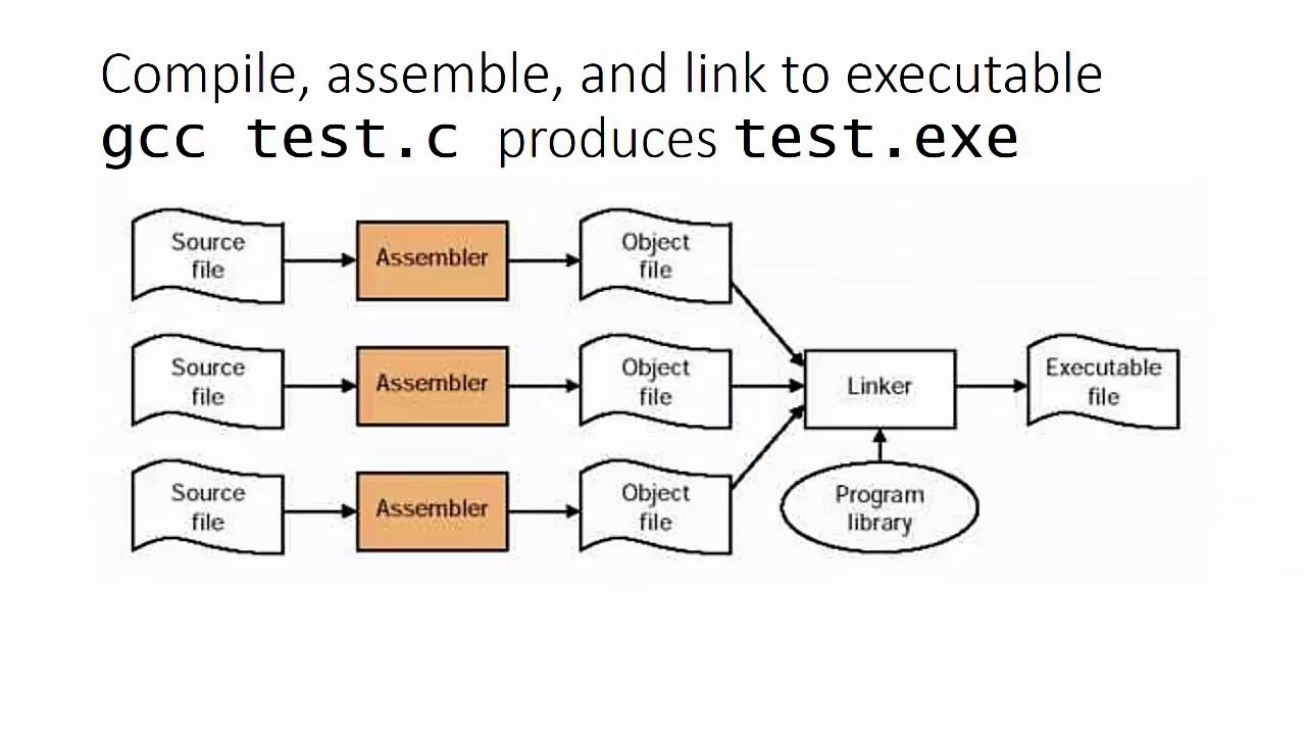
Lifetime of Hello World

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



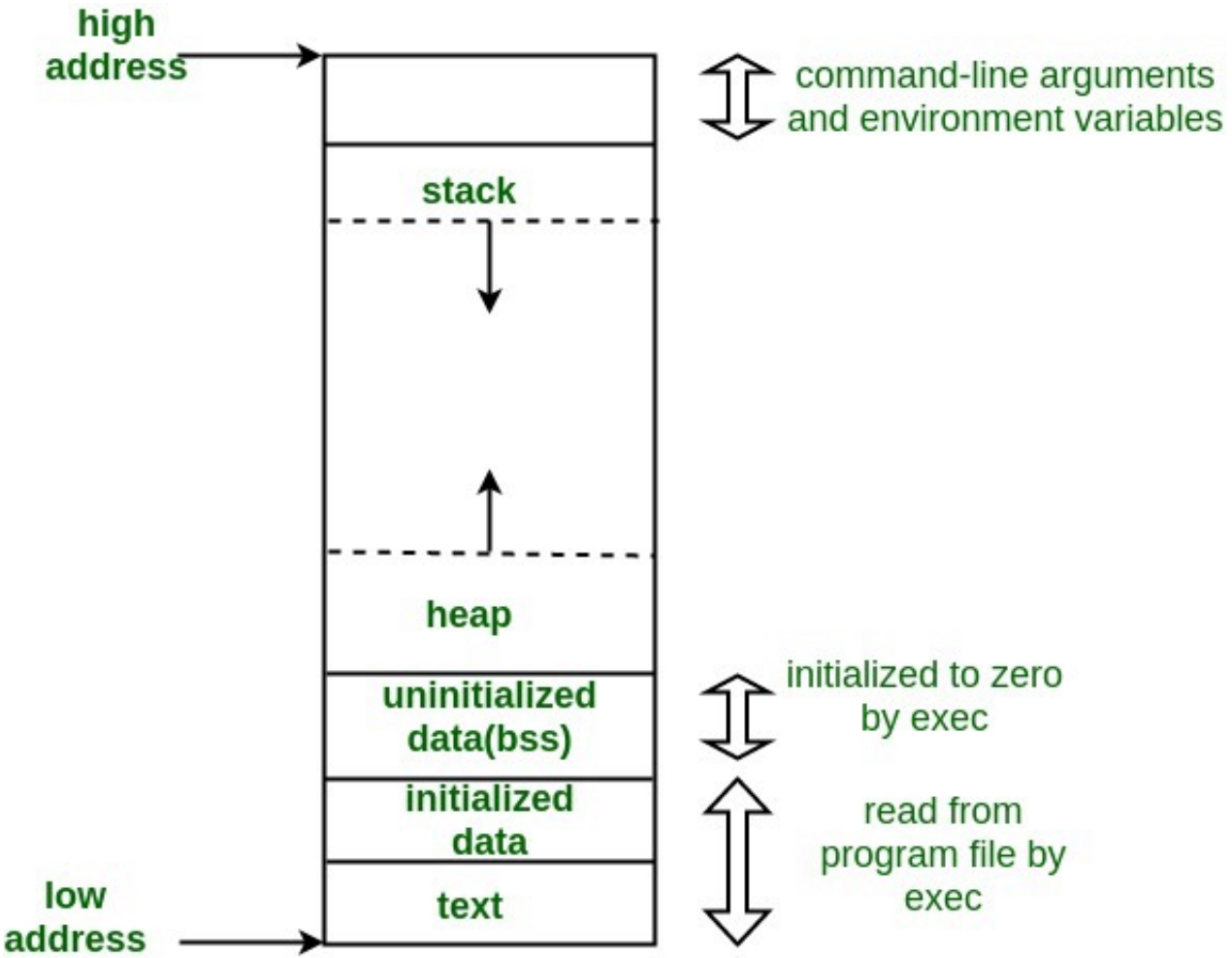
Lifetime of Hello World

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



Misc

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