北京師範大學
人工智能学院

BQ10214301：计算机系统导论

课程概览 / Course Overview

任课教师：计卫星　　**原作者：**Randal E. Bryant & David R. O'Hallaron

概览 Overview

- 课程理念 / Course theme
- 五个基本事实 / Five realities
- 与其它CS/ECE课程之间的关系 / How the course fits into the CS/ECE curriculum
- 课程目标 / Course Orientation

课程理念 Course Theme:

系统知识非常强大 (Systems) Knowledge is Power!

北京師範大學 人工智能学院

- **系统知识 Systems Knowledge**
  - 如何组合硬件（处理器、内存、磁盘驱动器、网络基础设施）加上软件（操作系统、编译器、库、网络协议）来支持应用程序的执行
  - How hardware (processors, memories, disk drives, network infrastructure) plus software (operating systems, compilers, libraries, network protocols) combine to support the execution of application programs
  - 作为一个程序员如何能够最好地使用这些资源
  - How you as a programmer can best use these resources

课程理念 Course Theme:
系统知识非常强大 (Systems) Knowledge is Power!

北京師範大學 人工智能学院

- **学习本课程的有益成果 Useful outcomes from taking course**
  - 成为更高效的程序员 Become more effective programmers
    - 能够有效发现和消除错误 Able to find and eliminate bugs efficiently
    - 能够理解和调优程序性能 Able to understand and tune for program performance
  - 为后续CS&ECE“系统”类课程奠定基础 Prepare for later “systems” classes in CS & ECE
    - 编译器、操作系统、网络、计算机体系结构、嵌入式系统、存储系统等
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, etc.

理解系统的工作原理非常重要
It’s Important to Understand How Things Work

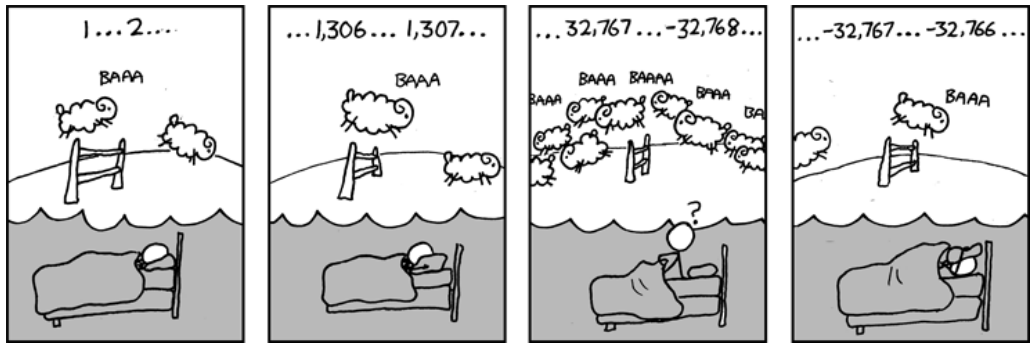
北京師範大學 人工智能学院

- **为何需要知道这些内容 Why do I need to know this stuff?**
  - 抽象很好，但是不要忘记现实 Abstraction is good, but don’t forget reality
- **大多数CS和CE课程强调抽象 Most CS and CE courses emphasize abstraction**
  - 抽象的数据类型 Abstract data types
  - 渐进分析（算法的时间复杂度） Asymptotic analysis
- **这些抽象有其限制 These abstractions have limits**
  - 特别是出现错误之时 Especially in the presence of bugs
  - 需要理解底层实现的具体细节 Need to understand details of underlying implementations
  - 有时抽象的接口不能提供所需要的控制或性能级别 Sometimes the abstract interfaces don’t provide the level of control or performance you need

重要事实#1/Great Reality #1:

int不是整数，float不是实数/Ints are not Integers, Floats are not Reals

- **示例1：x的平方总是大于等于0吗？Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!
  - Int's:
    - 40000 * 40000 = 1600000000
    - 50000 * 50000 = ??



- **示例2：加法结合率？Example 2: Is (x + y) + z = x + (y + z)?**
  - 对无符号和有符号整数正确 Unsigned & Signed Int's: Yes!
  - 浮点数： Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Source: xkcd.com/571

计算机算术 Computer Arithmetic

北京師範大學 人工智能学院

- **不会产生随机值/Does not generate random values**
  - 算术运算有重要的数学性质/Arithmetic operations have important mathematical properties
- **不能假设所有“通常”的数学性质成立/Cannot assume all “usual” mathematical properties**
  - 由于是有限位数的表示/Due to finiteness of representations
  - 整数运算满足“环”性质/Integer operations satisfy “ring” properties
    - 交换率、结合率和分配律/Commutativity, associativity, distributivity
  - 浮点运算满足“按序”性质/Floating point operations satisfy “ordering” properties
    - 单调性，符号的值/Monotonicity, values of signs
- **观察/Observation**
  - 需要理解哪些抽象适用于哪些上下文/Need to understand which abstractions apply in which contexts
  - 对编写编译器的人员和重要应用程序员重要的问题/Important issues for compiler writers and serious application programmers

重要事实#2 Great Reality #2:

你必须懂汇编语言You've Got to Know Assembly

北京師範大學 人工智能学院

- **你可能从来没有机会写汇编程序**
- **Chances are, you'll never write programs in assembly**
  - 编译器做的更好而且比你更有耐心 Compilers are much better & more patient than you are
- **但是理解汇编对机器级执行模式来说是关键**
- **But: Understanding assembly is key to machine-level execution model**
  - 出现错误时的程序行为 Behavior of programs in presence of bugs
    - 高级语言模型失灵 High-level language models break down
  - 调优程序性能 Tuning program performance
    - 理解编译器完成/没完成的优化工作 Understand optimizations done / not done by the compiler
    - 理解程序低效的原因 Understanding sources of program inefficiency

重要事实#2 Great Reality #2:

你必须懂汇编语言You've Got to Know Assembly

北京師範大學
人工智能学院

- **但是理解汇编对机器级执行模式来说是关键**
- **But: Understanding assembly is key to machine-level execution model**
  - 实现系统软件 Implementing system software
    - 编译器将机器代码作为目标 Compiler has machine code as target
    - 操作系统必须管理进程状态 Operating systems must manage process state
  - 创建/对抗恶意软件 Creating / fighting malware
    - x86汇编是首选的语言 x86 assembly is the language of choice!

北京師範大學 人工智能学院

重要事实#3：存储器很重要/Great Reality #3: Memory Matters

随机访问存储器是一种抽象/Random Access Memory Is an Unphysical Abstraction

- **内存不是无限界的 Memory is not unbounded**
  - 它必须进行分配和管理 It must be allocated and managed
  - 很多应用都受内存空间的限制 Many applications are memory dominated
- **内存引用错误非常致命 Memory referencing bugs especially pernicious**
  - 在时间和空间的影响都是滞后的 Effects are distant in both time and space
- **存储器性能并不一致 Memory performance is not uniform**
  - 高速缓冲和虚拟存储器可以显著影响程序性能 Cache and virtual memory effects can greatly affect program performance
  - 根据存储系统特点调整程序可以提升效率 Adapting program to characteristics of memory system can lead to major speed improvements

内存引用错误示例 / Memory Referencing Bug Example

北京師範大學
人工智能学院

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

```
fun(0)   =     3.14
fun(1)   =     3.14
fun(2)   =     3.1399998664856
fun(3)   =     2.0000006103515 6
fun(4)   =     3.14
fun(6)   =     Segmentation fault 段故障
```

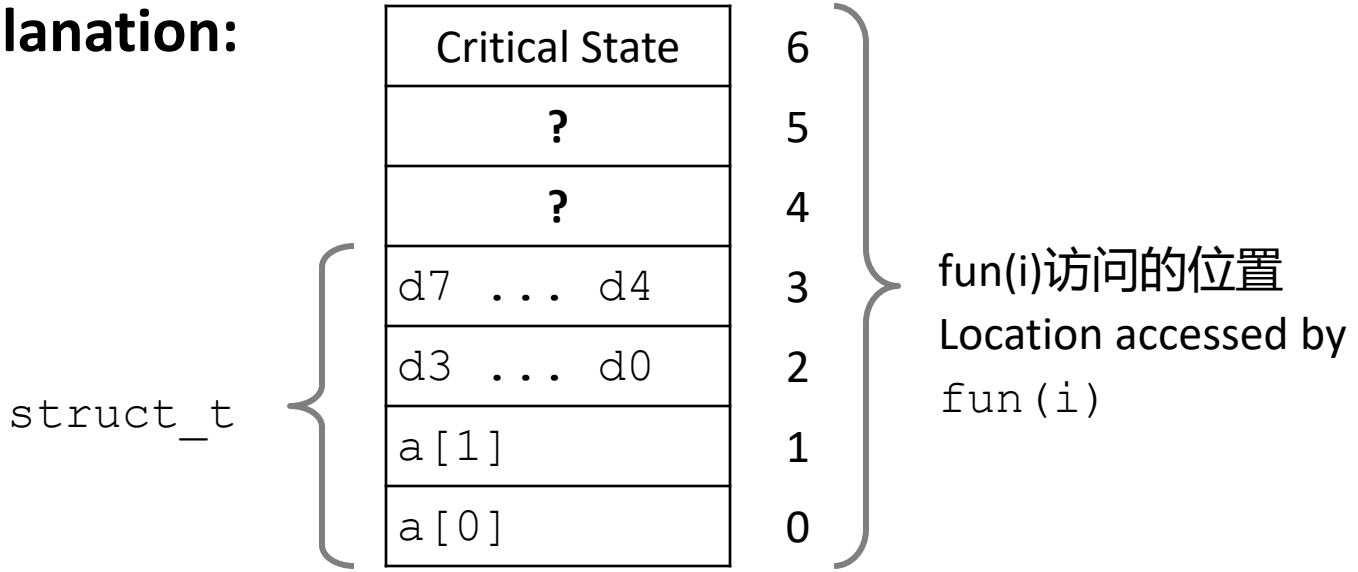
- 结果随着系统而不同 Result is system specific

内存引用错误示例
Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)   =     3.14
fun(1)   =     3.14
fun(2)   =     3.1399998664856
fun(3)   =     2.0000000061035156
fun(4)   =     3.14
fun(6)   =     Segmentation fault 段故障
```

解释 **Explanation:**



内存引用错误 Memory Referencing Errors

- **C语言和C++不提供任何内存保护 C and C++ do not provide any memory protection**
  - 数组引用超界 Out of bounds array references
  - 不合法的指针值 Invalid pointer values
  - 分配和释放内存滥用 Abuses of malloc/free
- **可能导致严重的错误 Can lead to nasty bugs**
  - 是否错误有任何影响取决于系统和编译器 Whether or not bug has any effect depends on system and compiler
  - 在远处产生影响 Action at a distance
    - 破坏的对象逻辑上和访问的对象毫不相关 Corrupted object logically unrelated to one being accessed
    - 错误的效果第一次观察到可能距离产生的时间很长 Effect of bug may be first observed long after it is generated
- **这种情况应该如何处理? How can I deal with this?**
  - 采用Java、Ruby、Python、ML等编程 Program in Java, Ruby, Python, ML, …
  - 理解可能会发生什么相互影响 Understand what possible interactions may occur
  - 使用或开发工具来检测引用错误（例如Valgrind） Use or develop tools to detect referencing errors (e.g. Valgrind)

重要的事实#4：性能不仅仅是渐进复杂度

Great Reality #4: There's more to performance than asymptotic complexity

北京師範大學
人工智能学院

- **常数因子也很重要 Constant factors matter too!**
- **甚至精确的操作计数都不能预测性能 And even exact op count does not predict performance**
  - 很容易发现10倍性能差异取决于如何编写代码 Easily see 10:1 performance range depending on how code written
  - 必须在多个级别进行优化：算法、数据表示、过程和循环 Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **必须理解系统才能优化性能 Must understand system to optimize performance**
  - 程序是如何编译和执行的 How programs compiled and executed
  - 如何测量程序性能和识别瓶颈 How to measure program performance and identify bottlenecks
  - 如何改进性能同时不破坏代码的模块性和通用性 How to improve performance without destroying code modularity and generality

内存系统性能示例
Memory System Performance Example

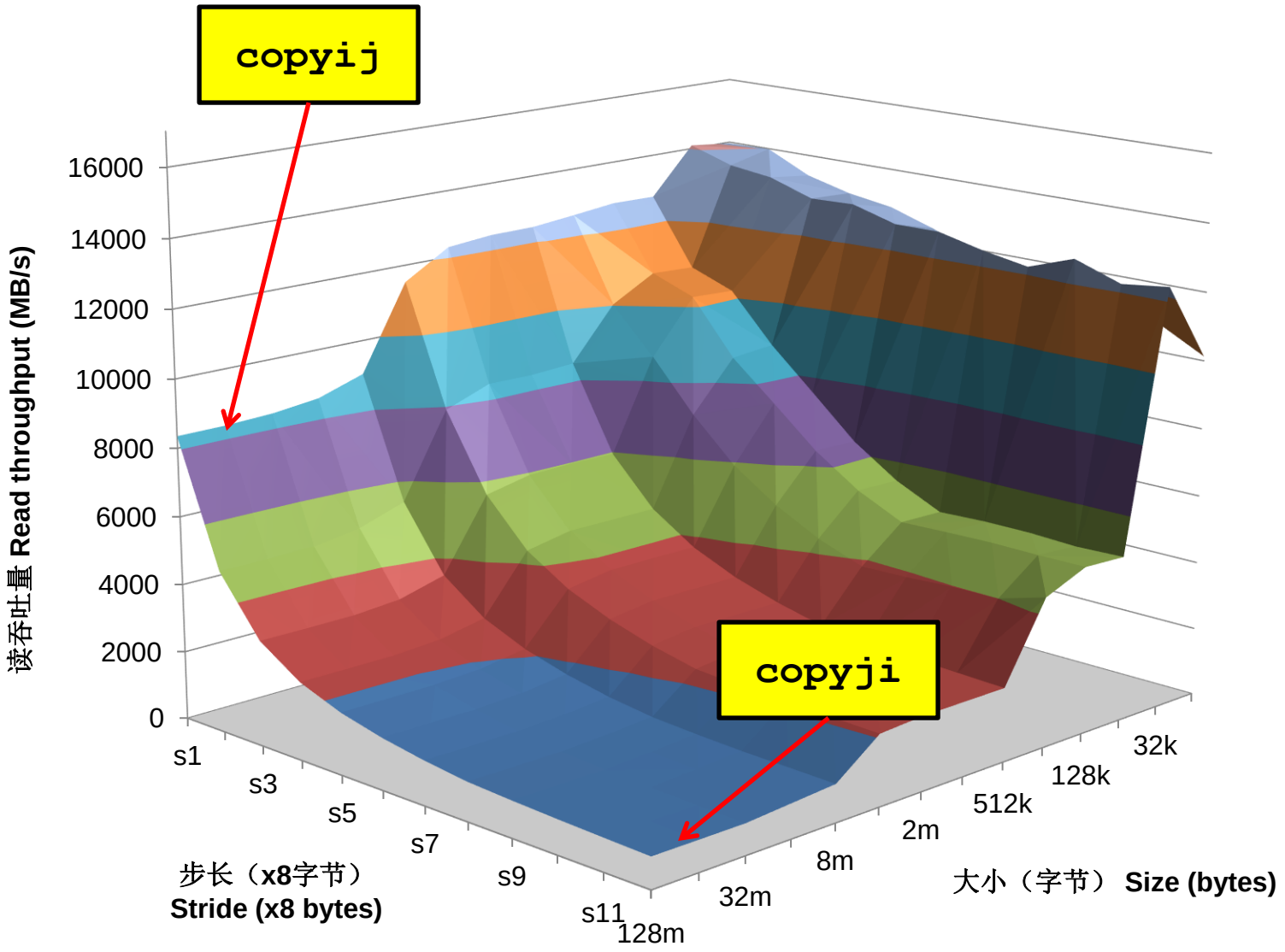
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms 2.0 GHz Intel Core i7 Haswell **81.8ms**

- 层次化存储器组织 Hierarchical memory organization
- 性能依赖于访问模式 Performance depends on access patterns
  - 包括如何设置步长遍历多维数组 Including how step through multi-dimensional array

为何出现性能差异/Why The Performance Differs



重要的事实#5：计算机不仅执行程序还做更多的事情

Great Reality #5: Computers do more than execute programs

- **计算机需要完成数据输入和输出 They need to get data in and out**
  - I/O系统对程序的可靠性和性能至关重要 I/O system critical to program reliability and performance
- **计算机通过网络进行彼此通信 They communicate with each other over networks**
  - 很多系统级问题由于网络引起Many system-level issues arise in presence of network
    - 自治进程的并发操作 Concurrent operations by autonomous processes
    - 处理不可靠的传输介质 Coping with unreliable media
    - 跨平台的兼容性 Cross platform compatibility
    - 复杂的性能问题 Complex performance issues

课程的视角 Course Perspective

北京師範大學 人工智能学院

- 大多数系统课程以构建者为中心 Most Systems Courses are Builder-Centric
  - 计算机体系结构 Computer Architecture
    - 用Verilog设计流水线处理器 Design pipelined processor in Verilog
  - 操作系统 Operating Systems
    - 实现操作系统的样例部分 Implement sample portions of operating system
  - 编译器 Compilers
    - 给简单的语言编写编译器 Write compiler for simple language
  - 网络 Networking
    - 实现和模拟网络协议 Implement and simulate network protocols

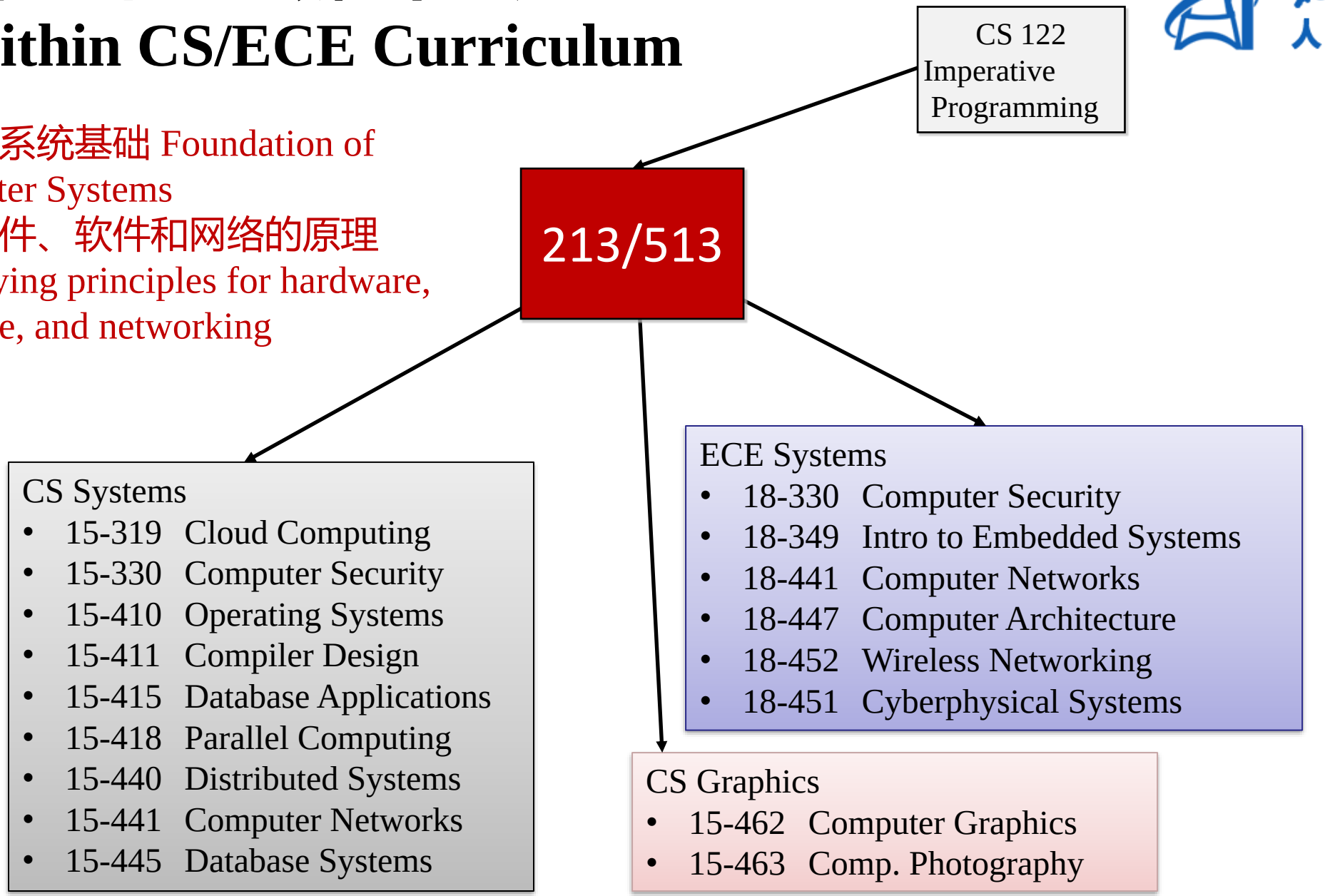
课程的视角（续）Course Perspective (Cont.)

北京師範大學 人工智能学院

- 我们的课程以程序员为中心 Our Course is Programmer-Centric
  - 目的是展示如果懂得更多底层系统的知识，程序员可以更加有效率 Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer
  - 使你能够 Enable you to
    - 编写的程序更加可靠和高效 Write programs that are more reliable and efficient
    - 包含需要在OS中嵌入钩子程序才能完成的功能 Incorporate features that require hooks into OS
      - 例如并发、信号处理程序 E.g., concurrency, signal handlers
  - 本课程覆盖的材料是独一无二的 Cover material in this course that you won't see elsewhere
  - 不仅仅是专门针对黑客的课程 Not just a course for dedicated hackers

本课程在CS/ECE教程中的角色
Role within CS/ECE Curriculum

计算机系统基础 Foundation of Computer Systems
底层硬件、软件和网络的原理
Underlying principles for hardware, software, and networking



主教材 Primary Textbooks

- Randal E. Bryant and David R. O'Hallaron,
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - http://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems

- 龚奕利 贺莲译,
  - *深入理解计算机系统*, 原书第3版, 机械工业出版社, 2017
  - https://course.educg.net 希冀在线平台
    - 用户名：bnu-学号，密码：学号
    - 课程材料，作业及提交，在线实验等

课程构成 Course Components

北京師範大學 人工智能学院

- 讲授 Lectures
  - 高级概念 Higher level concepts
- 习题课 Recitations
  - 概念应用、实验的重要工具和技巧、讲授内容的澄清、考试范围等 Applied concepts, important tools and skills for labs, clarification of lectures, exam coverage
- 实验 Labs
  - 课程的核心 The heart of the course
  - 每个1-2周 1-2 weeks each
  - 提供系统某一方面的深度理解 Provide in-depth understanding of an aspect of systems
  - 编程和度量 Programming and measurement
- 考试（期末）Exams (final)
  - 测试你对概念和数学原理的理解程度 Test your understanding of concepts & mathematical principles

策略：成绩评定 Policies: Grading

- 期末考试（60%） Final Exam (60%)
- 实验（30%） Labs (30%)
- 作业和出勤（10%） Homework and Attendance (10%)
- 最终成绩直接基于比例生成 Final grades based on a straight scale.

程序和数据 Programs and Data

北京師範大學 人工智能学院

- 主题 Topics
  - 位操作、运算和汇编语言程序 Bits operations, arithmetic, assembly language programs
  - C语言控制和数据结构的表示 Representation of C control and data structures
  - 包括体系结构和编译器方面的概念 Includes aspects of architecture and compilers

- 作业 Assignments
  - L1 (datalab): 操作比特位 Manipulating bits
  - L2 (bomblab): 拆除二进制炸弹 Defusing a binary bomb
  - L3 (attacklab): 代码注入攻击基础 The basics of code injection attacks

存储器层次结构 The Memory Hierarchy

北京師範大學 人工智能学院

- 主题 Topics
  - 存储器技术、存储器层次结构、高速缓冲存储器、磁盘和局部性 Memory technology, memory hierarchy, caches, disks, locality
  - 包括体系结构和操作系统方面的知识 Includes aspects of architecture and OS

- 作业 Assignments
  - L4 (cachelab): 构建一个cache模拟器并进行局部性优化 Building a cache simulator and optimizing for locality.
    - 学会如何开发程序的局部性 Learn how to exploit locality in your programs.

异常控制流 Exceptional Control Flow

北京師範大學 人工智能学院

- 主题 Topics
  - 硬件异常、进程、进程控制、Unix信号、非本地跳转 Hardware exceptions, processes, process control, Unix signals, nonlocal jumps
  - 包括编译器、操作系统和体系结构方面的知识 Includes aspects of compilers, OS, and architecture

- 作业 Assignments
  - L5 (tshlab): 编写你自己的Unix外壳程序 Writing your own Unix shell.
    - 对并发的第一次导入 A first introduction to concurrency

虚拟存储器 Virtual Memory

北京師範大學 人工智能学院

- 主题 Topics
  - 虚拟存储器、地址变换、动态存储分配
  - Virtual memory, address translation, dynamic storage allocation
  - 包括体系结构和操作系统方面的知识
  - Includes aspects of architecture and OS

- Assignments
  - L6 (malloclab): 编写你自己的内存分配软件包
  - Writing your own malloc package
    - 对系统级编程得到切实的感受
    - Get a real feel for systems-level programming

网络和并发 Networking, and Concurrency

北京師範大學 人工智能学院

- 主题 Topics
  - 高级和低级I/O、网络编程 High level and low-level I/O, network programming
  - 互联网服务、Web服务器 Internet services, Web servers
  - 并发、并发服务器设计、线程 concurrency, concurrent server design, threads
  - 带选择的I/O多路复用 I/O multiplexing with select
  - 包括网络、操作系统和体系结构方面的知识 Includes aspects of networking, OS, and architecture

- 作业 Assignments
  - L7 (proxylab): 编写你自己的Web代理 Writing your own Web proxy
    - 学会网络编程和更多有关并发和同步 Learn network programming and more about concurrency and synchronization.

优化流水线处理器的性能
Optimizing the Performance of a Pipelined Processor

北京師範大學
人工智能学院

- 主题 Topics
  - 指令集体系结构 Instruction Set Architecture
  - 逻辑设计、硬件控制语言HCL Logic Design， Hardware Control Language HCL
  - 顺序执行CPU实现 Sequential CPU Implementations
  - 流水线CPU实现 Pipelined CPU Implementations
  - 包括体系结构方面的知识 Includes aspects of architecture

- 作业 Assignments
  - L8 (archlab): 编写你自己的流水线CPU模拟器 Writing your own pipelined CPU simulator
    - 学会流水线处理器的设计与实现，优化其性能 learn about the design and implementation of a pipelined processor, optimizing the performance.

致谢 Acknowledgments

- 本文档基于CMU的15-213: Introduction to Computer Systems课程材料加工获得
- 感谢原作者Randal E. Bryant 和David R. O’Hallaron的辛苦付出

北京師範大學
人工智能学院

BQ10214301：计算机系统导论

第1章计算机系统漫游 / A Tour of Computer System

任课教师：计卫星　　**原作者：**Randal E. Bryant & David R. O'Hallaron

计算机系统漫游 / A Tour of Computer System

- 我们通过跟踪hello程序的生命周期来开始对系统的学习
- We begin our study of systems by tracing the lifetime of the hello program.

```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 0;
}
```

信息是比特位+上下文
Information Is Bits + Context

北京師範大學 人工智能学院

- **hello.c的ASCII文本表示** ASCII text representation of hello.c

```
#    i    n    c    l    u    d    e    SP   <    s    t    d    i    o    .
35   105  110  99   108  117  100  101  32   60   115  116  100  105  111  46

h    >    \n   \n   i    n    t    SP   m    a    i    n    (    )    \n   {
104  62   10   10   105  110  116  32   109  97   105  110  40   41   10   123

\n   SP   SP   SP   SP   p    r    i    n    t    f    (    "    h    e    l
10   32   32   32   32   112  114  105  110  116  102  40   34   104  101  108

l    o    ,    SP   w    o    r    l    d    \    n    "    )    ;    \n   SP
108  111  44   32   119  111  114  108  100  92   110  34   41   59   10   32

SP   SP   SP   r    e    t    u    r    n    SP   0    ;    \n   }    \n
32   32   32   114  101  116  117  114  110  32   48   59   10   125  10
```

- 信息由一串比特位表示，区分不同数据对象是上下文
- Information is represented as a bunch of bits. distinguishing different data objects is the context.

实用程序 Utilities

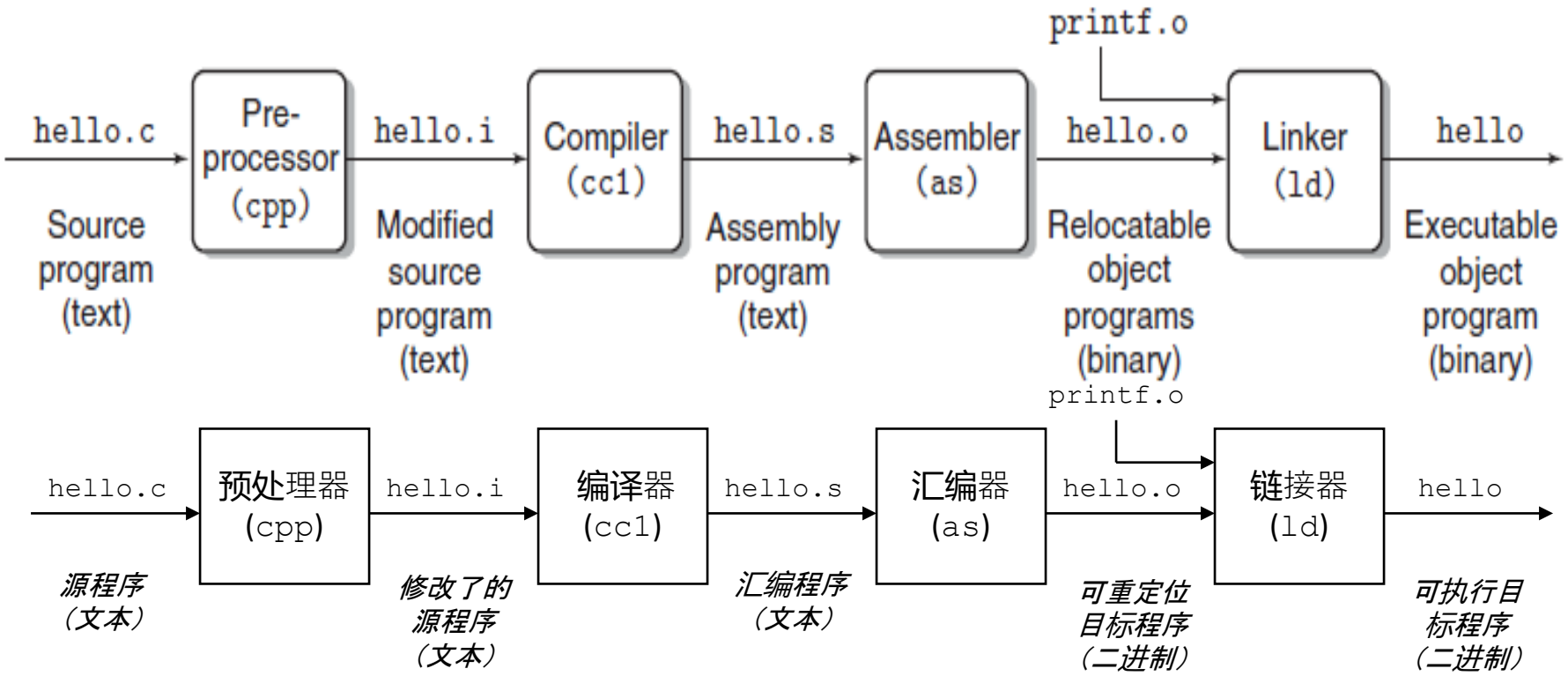
- 编程语言 Programming language
  - ANSI C
- 编译器 Compiler
  - GNU-gcc： GNU Complier Collection（GNU编译器套件）
- 工具 Tools
  - GNU tool chain 工具链

34

编译系统 Compilation System

北京師範大學
人工智能学院

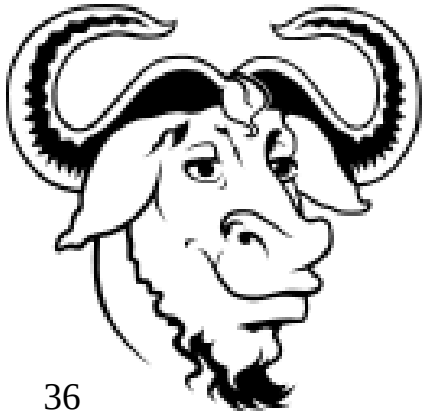
- Linux> gcc -o hello hello.c




```
1   main:
2       subq    $8, %rsp
3       movl    $.LC0, %edi
4       call    puts
5       movl    $0, %eax
6       addq    $8, %rsp
7       ret
```

GNU’s not Unix

- 自由软件 Free software
- Richard Stallman, 1984
- 完整的类Unix系统，有源代码 A complete Unix-like system with source code
- 一套环境 An environment
  - Unix操作系统所有主要组件 All major components of a Unix operating system
  - 不包括内核 Except for kernel



36

C编程语言 The C Programming Language

- C语言创建于 C was developed
  - in 1969 to 1973
  - by Dennis Ritchie of Bell Laboratories（贝尔实验室）.
- 美国国家标准学会（ANSI）The American National Standards Institute (ANSI)
  - 1989年正式批准了ANSI C标准 ratified the ANSI C standard in 1989.
- 该标准定义了 The standard defines
  - the C language
  - 以及一组库函数，称为C语言标准库 and a set of library functions known as the *C standard library*.

C编程语言 The C Programming Language

北京師範大學 人工智能学院

- 在他们的经典著作中描述了 / ANSI C Kernighan and Ritchie describe ANSI C in their classic book
  - 被称为“K&R”which is known affectionately as “K&R” .
- 用Ritchie的话来说，C语言是 / In Ritchie’s words, C is
  - 古怪的 / quirky,
  - 有缺陷的 / flawed,
  - 同时也是巨大的成功 / and an enormous success.
- 为何会取得如此成功？ / Why the success?

38

C编程语言 The C Programming Language

北京師範大學 人工智能学院

- *C语言与Unix操作系统关系密切 C was closely tied with the Unix operating system*
  - C语言从一开始就是作为Unix系统编程语言而开发出来的
    C was developed from the beginning as the system programming language for Unix.
  - 大部分Unix内核以及所有支持工具和库函数都是用C语言编写的
    Most of the Unix kernel, and all of its supporting tools and libraries, were written in C.
  - 随着70年代后期到80年代早期Unix在大学的广泛流行，许多人开始接触C语言并喜欢上它
    As Unix became popular in universities in the late 1970s and early 1980s, many people were exposed to C and found that they liked it.
  - 由于Unix几乎全部是用C编写的，它可以很方便地移植到新的机器，这种特点为C和Unix赢得了更为广泛的支持
    Since Unix was written almost entirely in C, it could be easily ported to new machines, which created an even wider audience for both C and Unix.

39

C编程语言 The C Programming Language

- *C语言小而简单 C is a small, simple language.*
  - 设计是由一个人而非一个协会掌控的，因此是一个简洁，没有什么冗赘的设计
  - The design was controlled by a single person, rather than a committee, and the result was a clean, consistent design with little baggage.
  - K&R这本书用大量的例子和练习描述了完整的语言及标准库，全书不过261页
  - The K&R book describes the complete language and standard library, with numerous examples and exercises, in only 261 pages.
  - C语言的简单性使它相对易于学习和移植到不同的计算机上
  - The simplicity of C made it relatively easy to learn and to port to different computers.

C编程语言 The C Programming Language

- *C语言是为实践目的设计的*
- *C was designed for a practical purpose.*
  - C语言是设计用来实现Unix操作系统的
  - C was designed to implement the Unix operating system.
  - 后来，其他人发现能够用这门语言无障碍地编写他们想要的程序
  - Later, other people found that they could write the programs they wanted, without the language getting in the way.

C编程语言 The C Programming Language

- C语言是系统级编程的首选
- C is the language of choice for system-level programming
- 它也非常适用于应用级程序的编写
- There is a huge installed base of application-level programs as well.

42

C编程语言 The C Programming Language

- 然而，它也并非适用于所有程序员和所有情况
- However, it is not perfect for all programmers and all situations
  - C语言的指针是造成困惑和编程错误的一个常见原因
  - C pointers are a common source of confusion and programming errors
  - C语言也缺乏有用抽象的显式支持，例如类和对象
  - C also lacks explicit support for useful abstractions such as classes and objects
  - 像C++和Java这样针对应用级程序的新语言解决了这些问题
  - Newer languages such as C++ and Java address these issues for application-level programs

43

C语言标准化 Standardization of C

- 原始贝尔实验室C语言版本 The original Bell Labs version of C
  - K&R著作的第1版 the 1st edition of the book K&R
- 1989年发布ANSI C标准 The ANSI C standard in 1989
  - 美国国家标准学会 The American National Standards Institute
  - 修改了函数声明的方式 Modify the way functions are declared
  - K&R著作的第2版 the 2nd edition of the book K&R
  - ISO C90 (The International Standards Organization国际标准化组织)

44

C语言标准化 Standardization of C

北京師範大學 人工智能学院

- ISO C99
  - 引入了一些新的数据类型 Introduced some new data types
  - 对使用不符合英语语言字符的文本串提供了支持 Provided support for text strings requiring characters not found in the English language
- Gcc支持 Gcc supporting
  - Unix> gcc -std=c99 prog.c
  - -ansi and -std=c89 have the same effect 效果相同

45

C语言标准化 Standardization of C

C version gcc	命令行选项 command line option
GNU 89	*none, -std=gnu89*
ANSI, ISO C90	-ansi, -std=c89
ISO C99	-std=c99
GNU 99	-std=gnu99
ISO 11	-std=c11
GNU 11	-std=gnu11

- **ACM:**
  - 计算机协会 **Association of Computing Machinery**
- **IEEE:**
  - 电气与电子工程师协会 **Institute of Electrical and Electronics Engineers**[46]

了解编译系统如何工作是大有益处的
It Pays to Understand How Compilation Systems Work

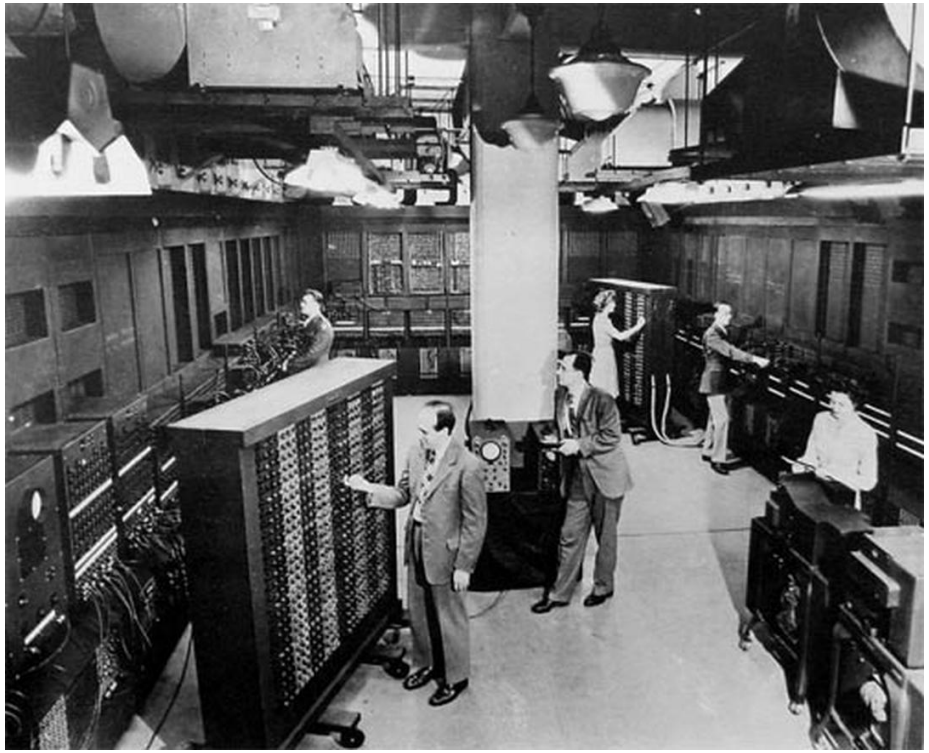
北京師範大學 人工智能学院

- 优化程序性能 Optimizing program performance.
  - 现代编译器通常生成很好的代码 Modern compilers usually produce good code.
  - 需要基本理解机器代码和编译器将不同的C语句转换成机器代码的方式 need a basic understanding of machine-level code and how the compiler translates different C statements into machine code.
- 理解链接时出现的错误 Understanding link-time errors.
  - 一些令人困惑的编程错误都与链接器操作有关，特别是构建大型软件 some of the most perplexing programming errors are related to the operation of the linker, especially trying to build large software systems.
- 避免安全漏洞 Avoiding security holes.
  - 缓冲区溢出漏洞是造成大多数网络和互联网服务器上安全漏洞的主要原因 buffer overflow vulnerabilities have accounted for many of the security holes in network and Internet servers.

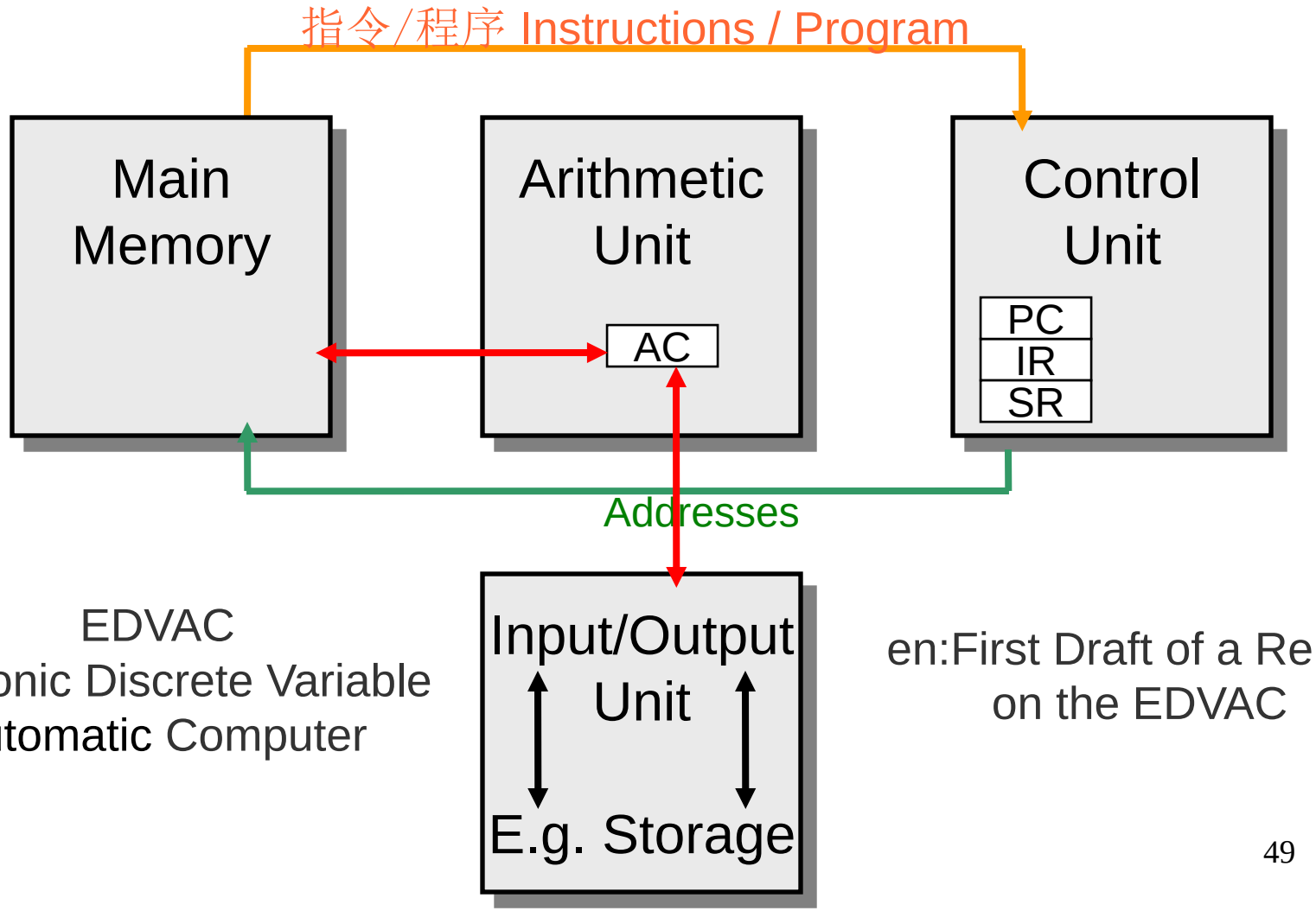
第一台通用目的计算机
The first general purpose computer

北京師範大學
人工智能学院

- ENIAC
  - 电子数字积分计算机 Electronic Numerical Integrator And Computer
  - 1946年宾夕法尼亚大学出品 Delivered by UPenn. on Feb. 14, 1946
  - John Mauchly、John Eckert
  -



计算机硬件-冯·诺依曼体系结构
Computer Hardware - Von Neumann Architecture



EDVAC
Electronic Discrete Variable Automatic Computer

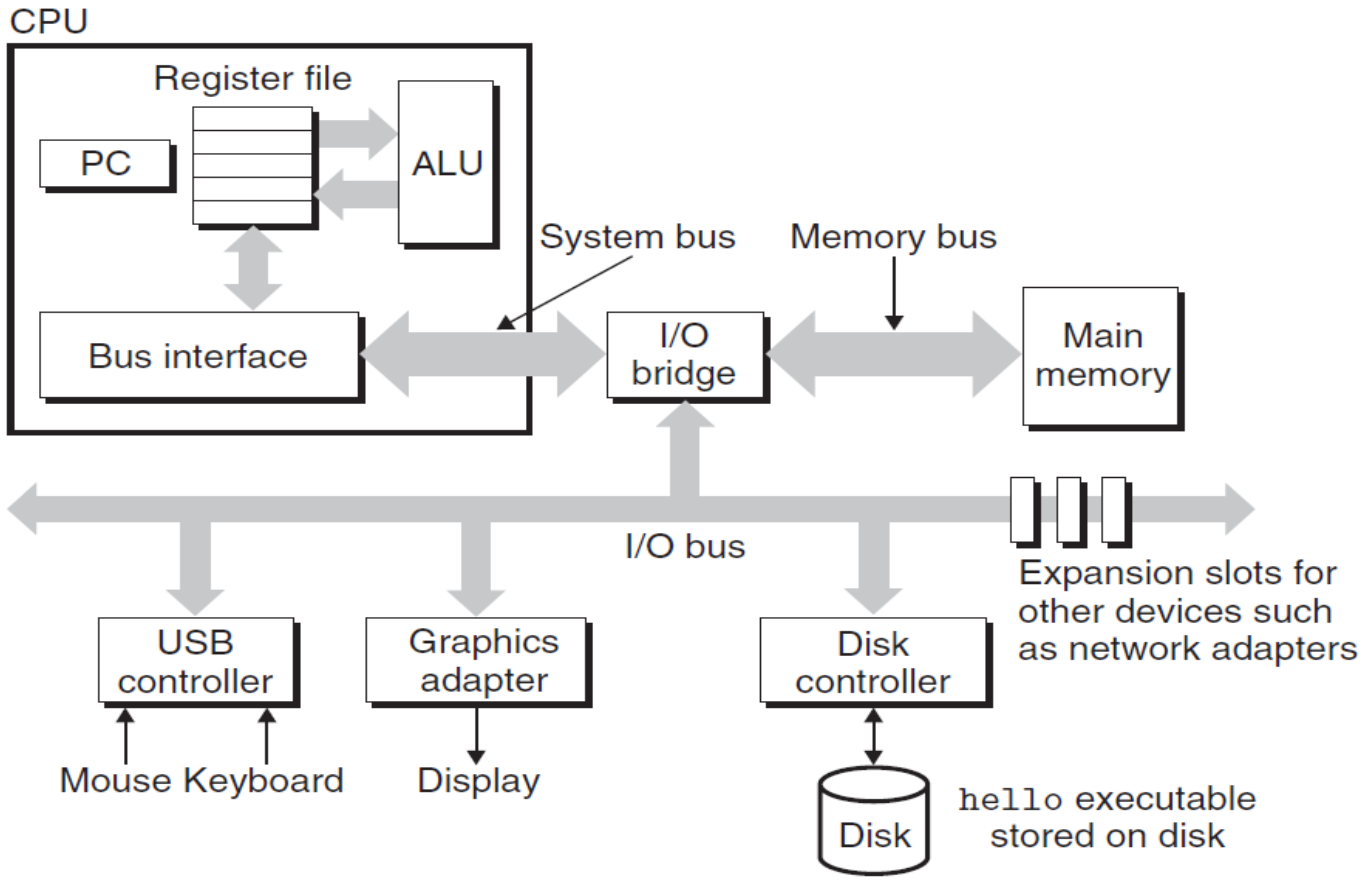
en:First Draft of a Report on the EDVAC

49

处理器读并解释存储在内存中的指令

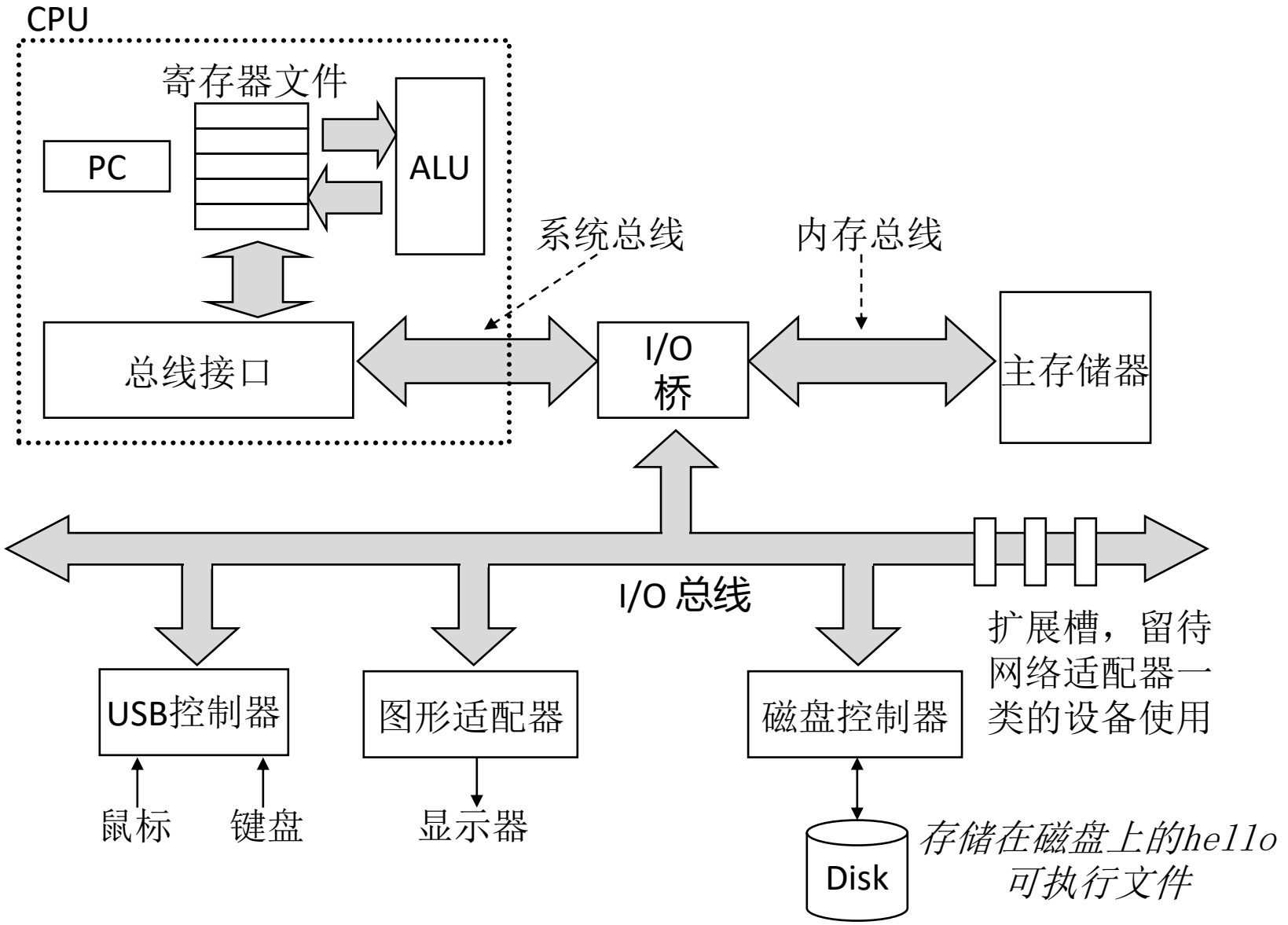
Processors Read and Interpret Instructions Stored in Memory


```
linux> ./hello
hello, world
linux>
```

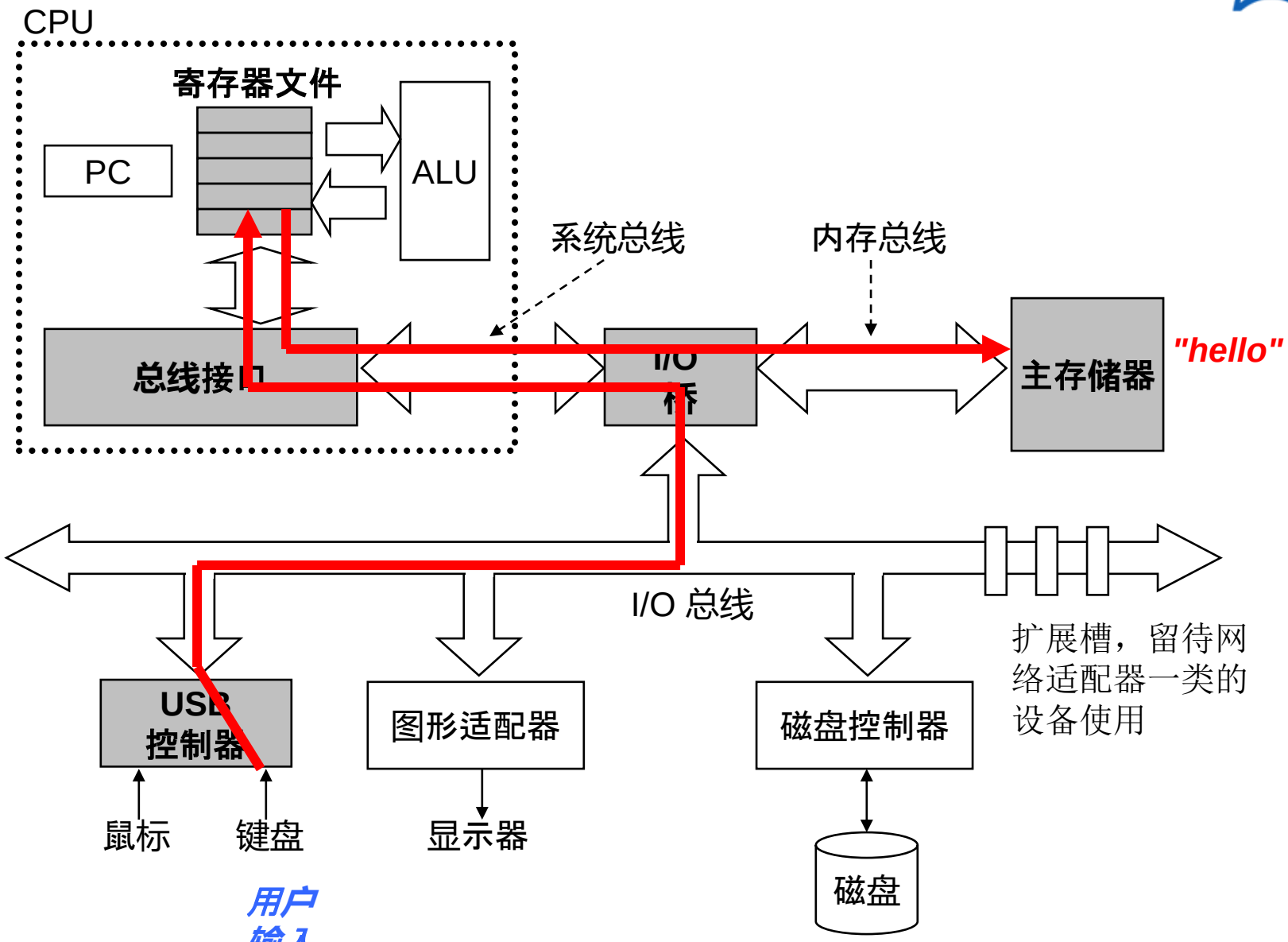


系统的硬件组成

Hardware organization of system



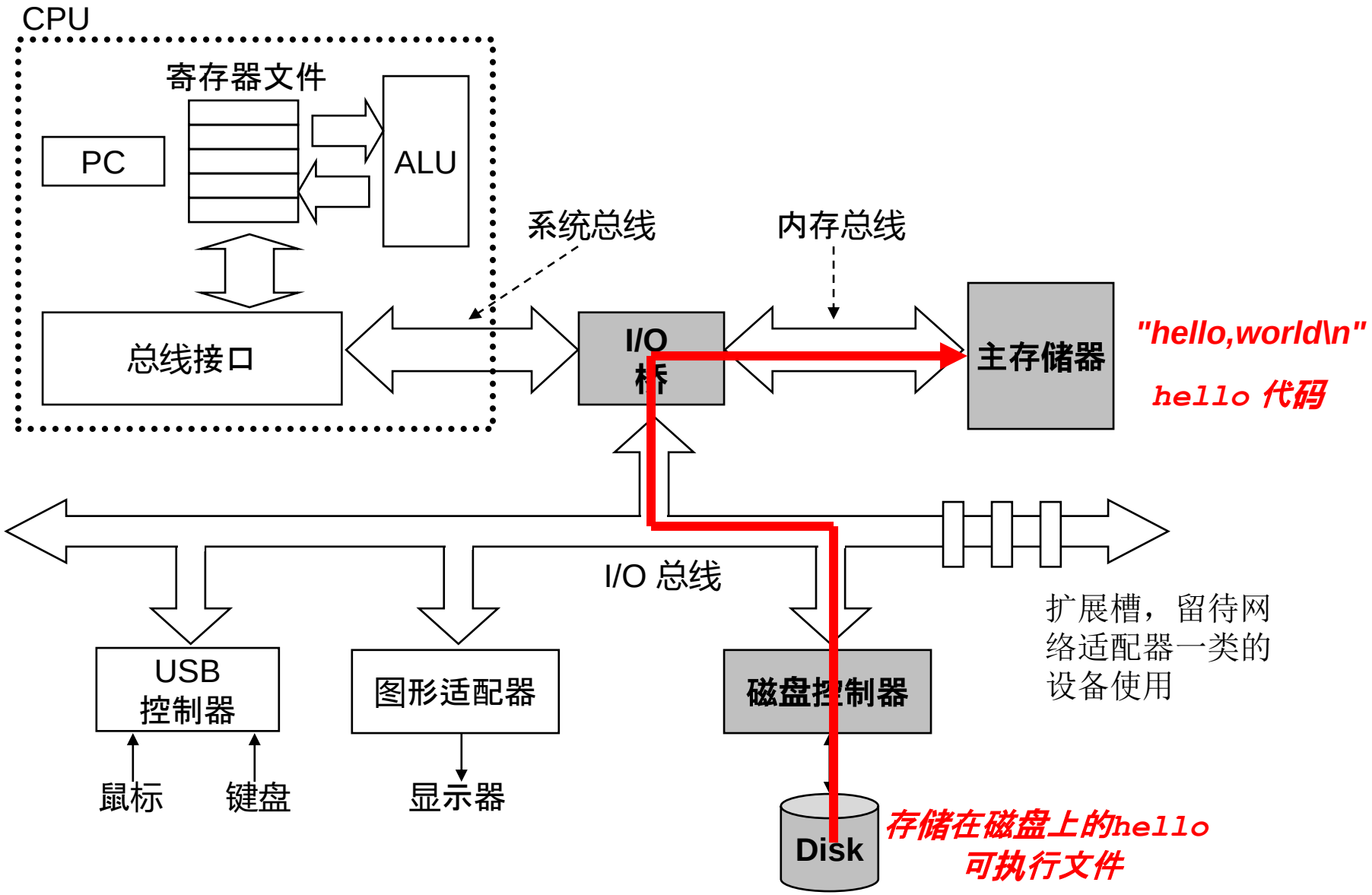
运行hello程序 Running the hello Program



将可执行文件从磁盘加载到主存中

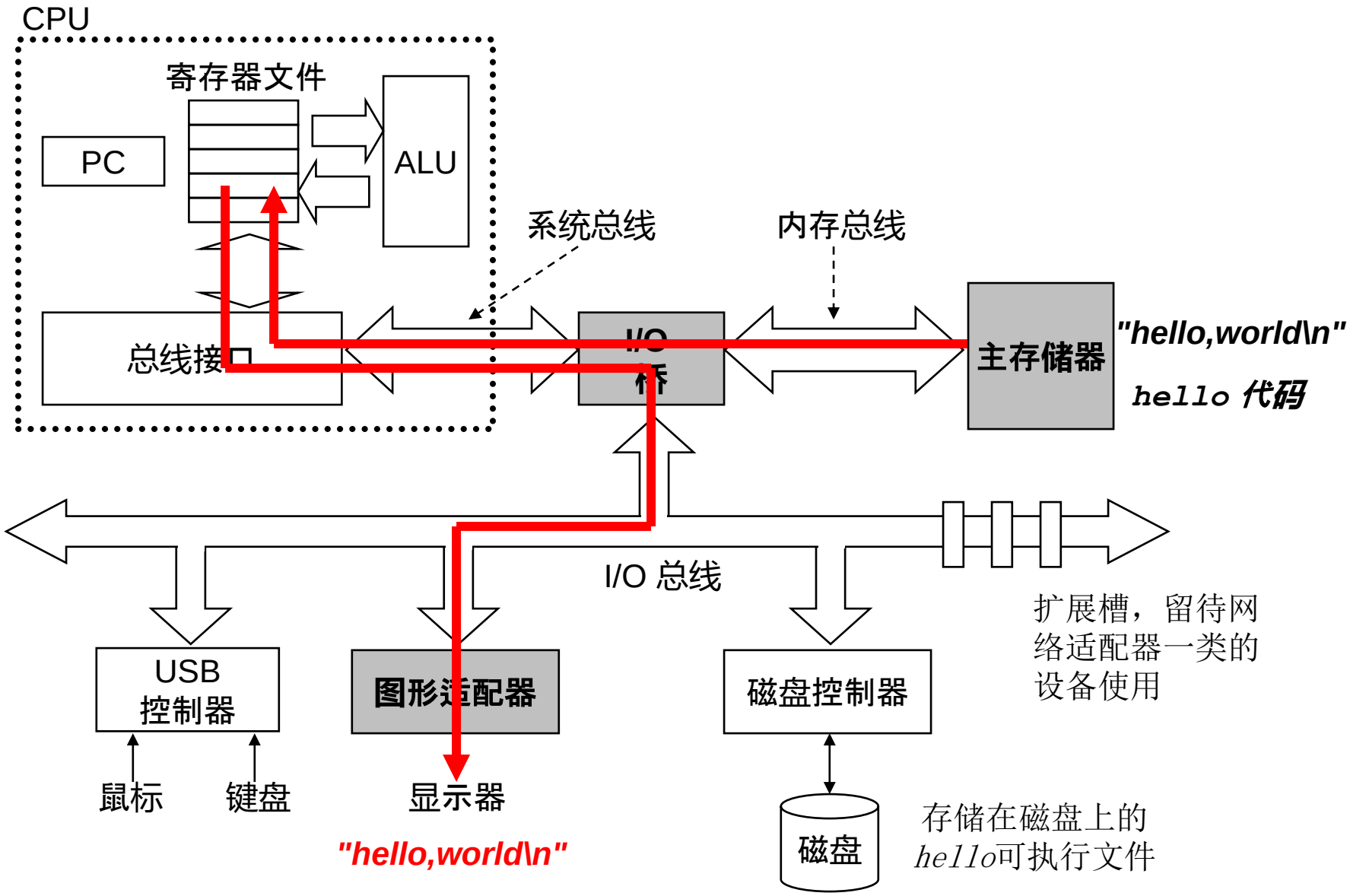
Loading the executable from disk into main memory

北京師範大學
人工智能学院



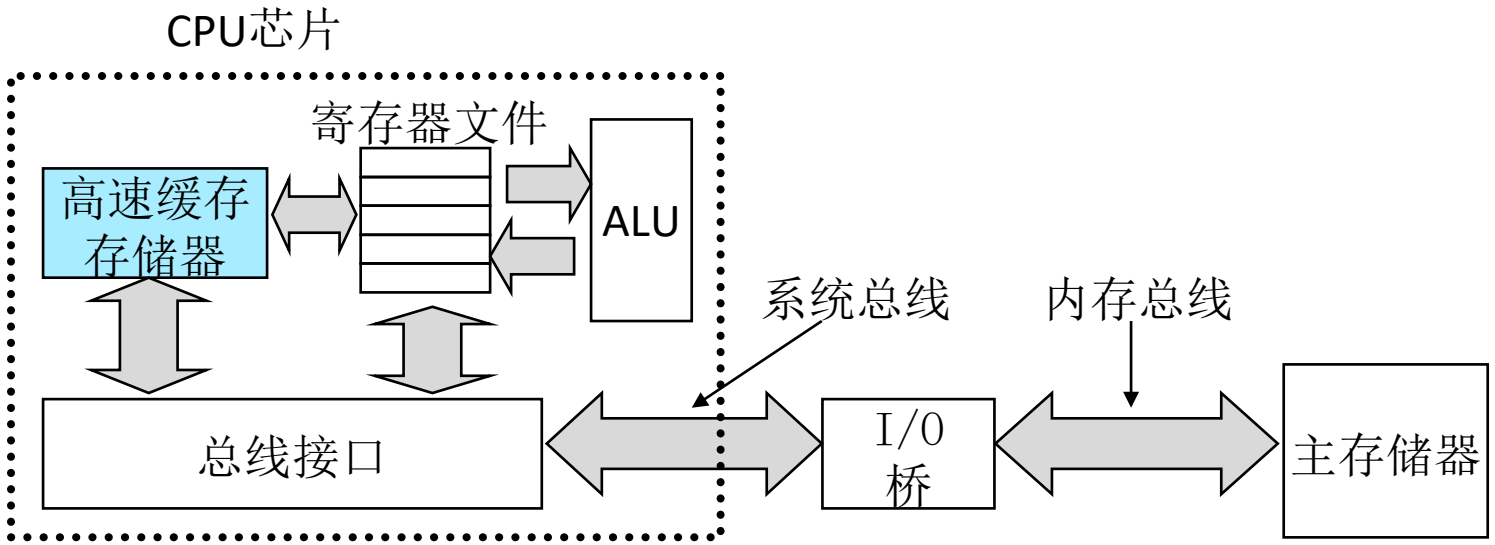
将输出字符串从内存写入显示器

Writing the output string from memory to the display

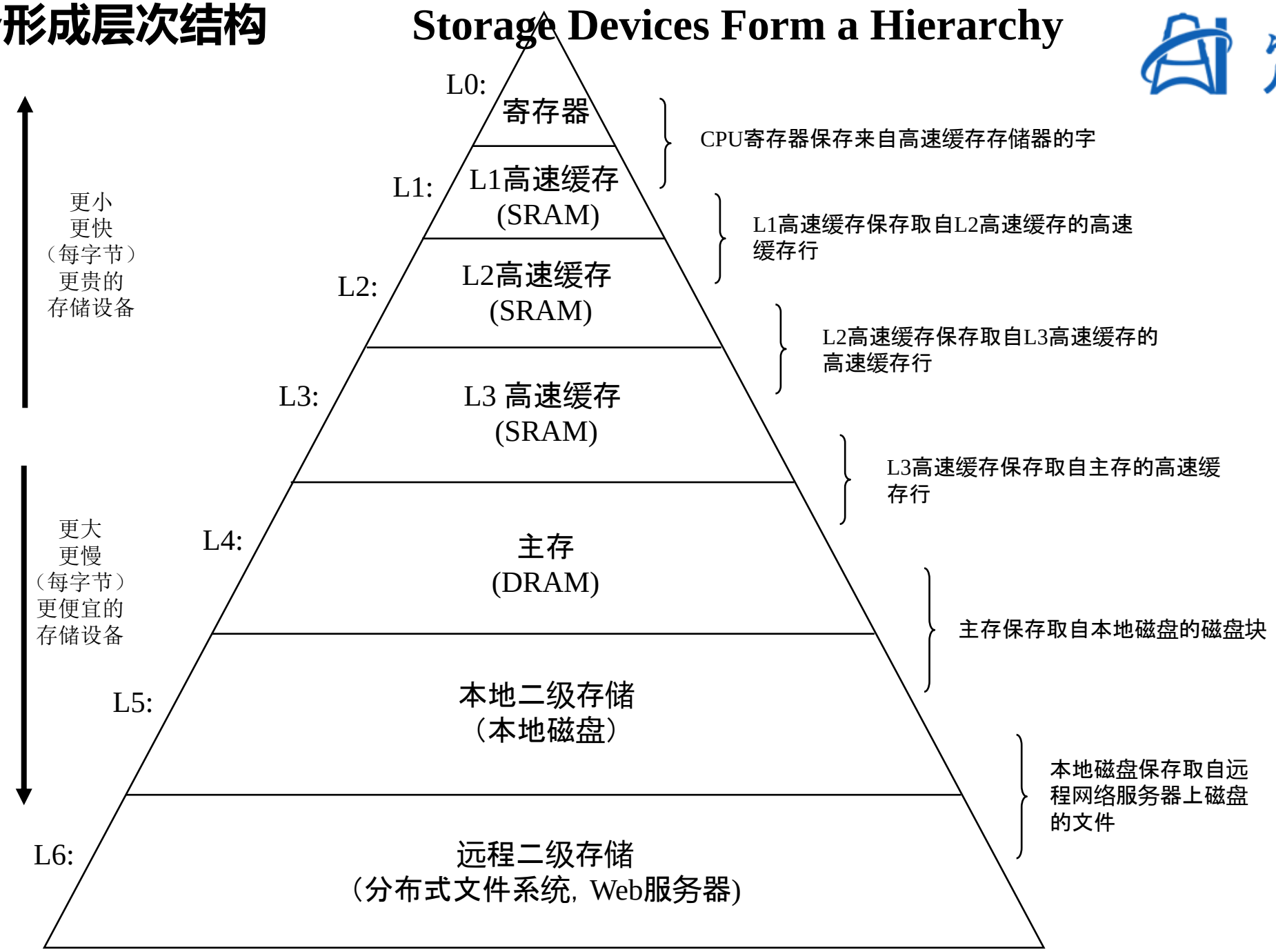


高速缓存至关重要/Caches Matter

- 处理器和内存速度鸿沟持续增大 processor-memory gap continues to increase
- 更小和快速的存储设备称为cache存储器（简称cache） smaller, faster storage devices called cache memories (or simply caches)
- 用称为静态随机访问存储器硬件技术实现 implemented with a hardware technology known as static random access memory (SRAM).

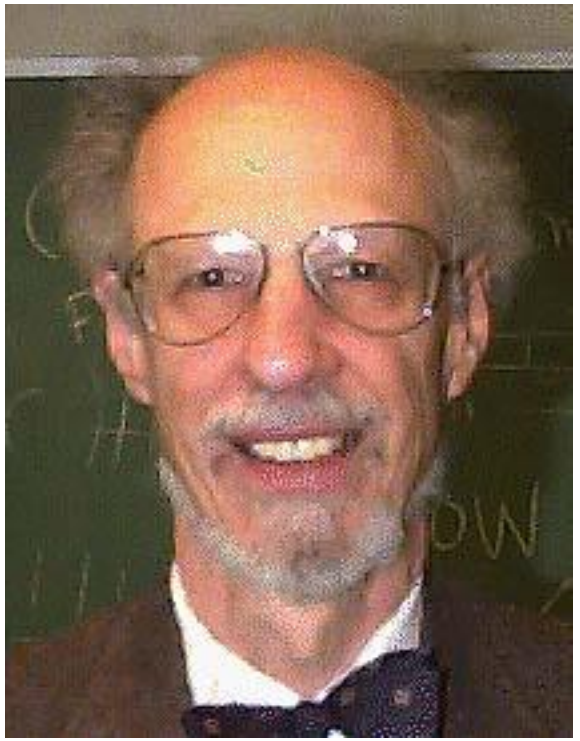


存储设备形成层次结构 Storage Devices Form a Hierarchy



操作系统
Operating Systems

- 1960’s
  - IBM OS/360, Honeywell Multics,
- Fernado Jose Corbató
  - IEEE Computer Pioneer Award, 1982
  - ACM Turing Award, 1990



57

操作系统/Operating Systems

- Unix
  - Bell Lab, DEC PDP-7, 1969
  - Ken Thompson, Dennis Ritchie, Doug Mcllroy, Joe Ossana
  - 1970 Brian Kernighan dubbed the system “Unix”
  - Rewritten in C in 1973, announced in 1974
  - BSD (UC, Berkeley), System V(Bell lab)
  - Solaris (Sun Microsystem)
- Posix standard
- Ken Thompson, Dennis Ritchie
  - ACM Turing Award, 1983

58

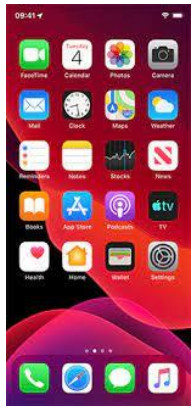
Linux

- 1991, Linus Torvalds
- Unix-like operating systems
- 386(486)AT, bash(1.08), gcc(1.40)
- Posix complaint version of Unix operating system
- Available on a wide array of computers
  - From handheld devices to mainframe computers
  - wristwatch

We have seen a bunch of Operating Systems

北京師範大學
人工智能学院

Linux



北京師範大學
人工智能学院

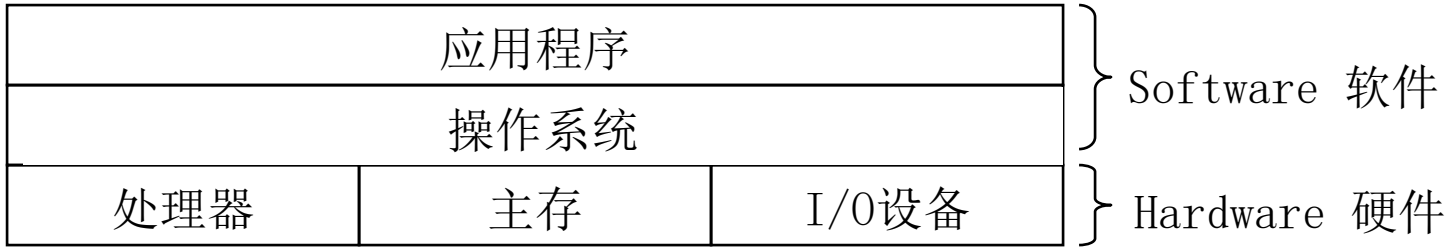
Operating Systems are everywhere





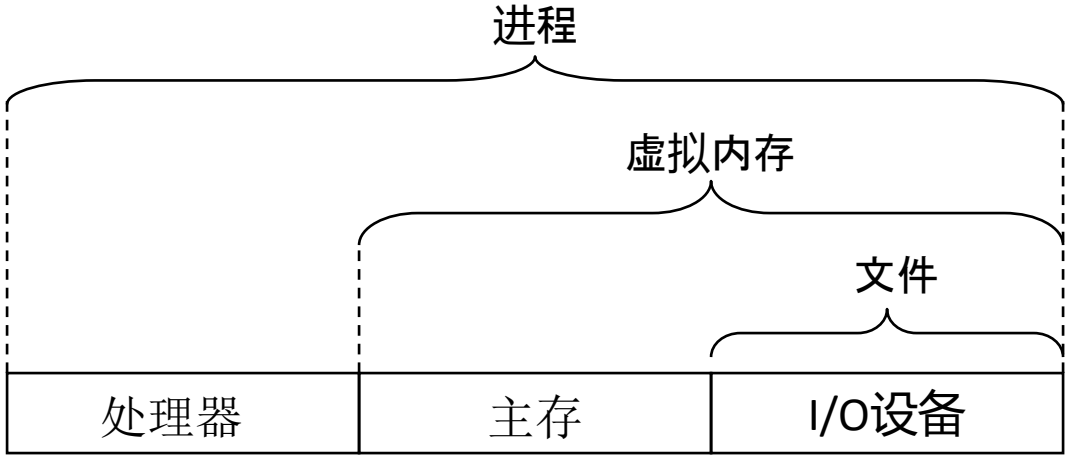


操作系统管理硬件
Operating System Manages Hardware



Layered view of a computer system

计算机系统的分层视图



Abstractions provided by an operating system

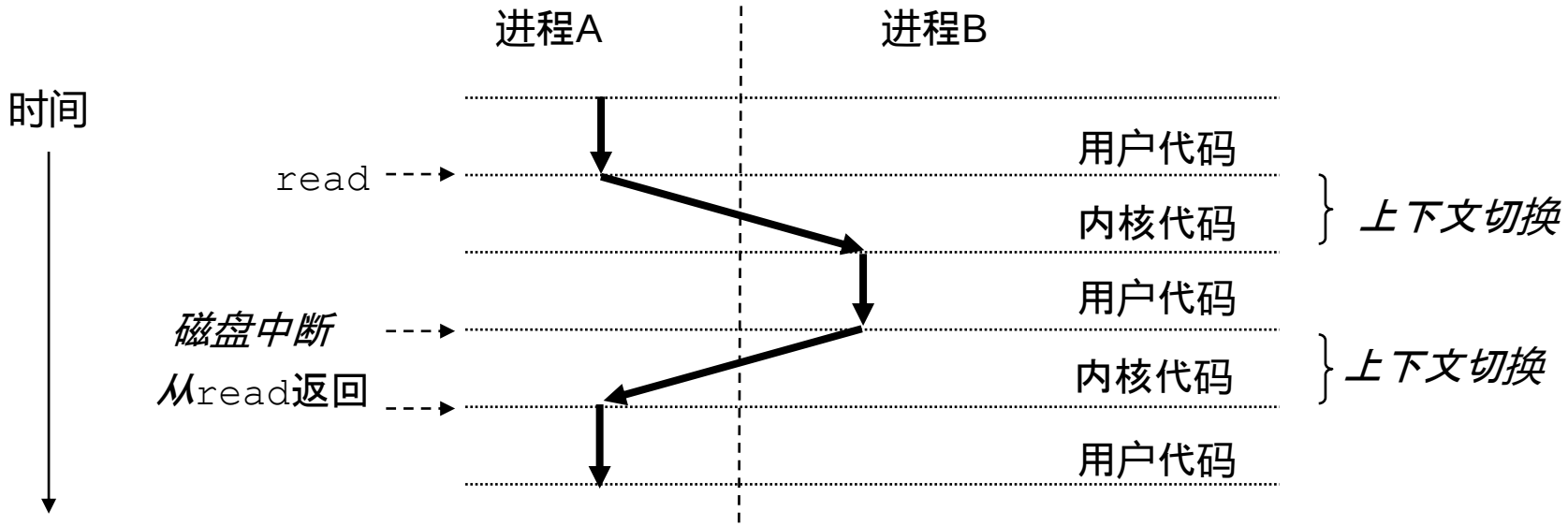
操作系统提供的抽象表示

操作系统管理硬件
Operating System Manages Hardware

北京師範大學 人工智能学院

- 将操作系统看作应用程序和硬件之间插入的一个软件层 think of the operating system as a layer of software interposed between the application program and the hardware
- 两个基本目的： two primary purposes:
  - 保护硬件防止被失控的应用误用 to protect the hardware from misuse by runaway applications.
  - 给应用提供简单统一的机制操作复杂和差异巨大的低层硬件设备 to provide applications with simple and uniform mechanisms for manipulating complicated and often wildly different low-level hardware devices.
- 基本抽象：进程、虚存和文件 fundamental abstractions: processes, virtual memory, and files.
  - 文件是I/O设备的抽象 files are abstractions for I/O devices,
  - 虚拟存储器是主存和磁盘I/O设备的抽象 virtual memory is an abstraction for both the main memory and disk I/O devices
  - 进程是处理器、主存和I/O设备的抽象 processes are abstractions for the processor, main memory, and I/O devices.

进程上下文切换
Process context switching



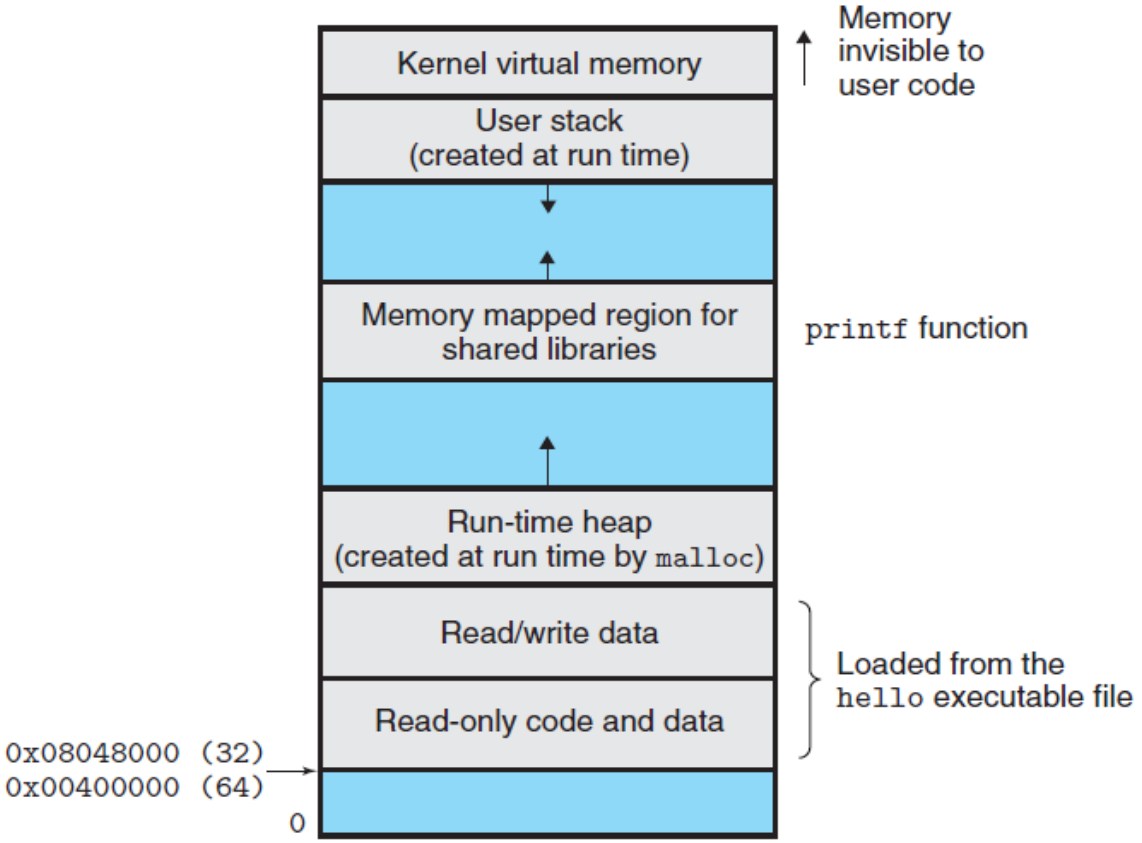
A process is the operating system's abstraction for a running program.

进程是操作系统对运行程序的抽象

虚拟存储器/Virtual Memory

北京師範大學 人工智能学院

- 每个进程有同样一致的内存视图，称为进程的虚地址空间 Each process has the same uniform view of memory, known as its virtual address space.

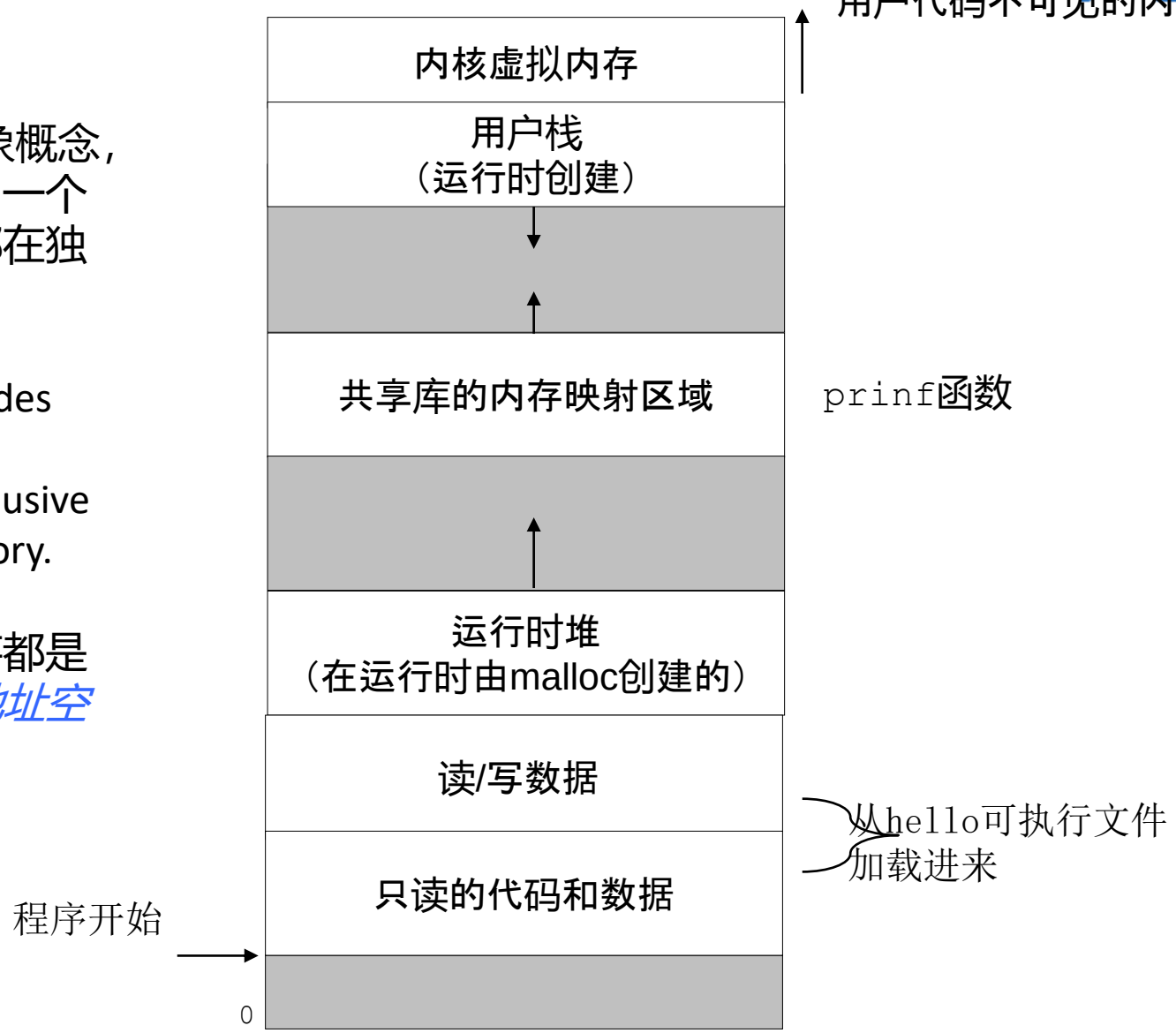


进程的虚拟地址空间

*虚拟内存*是一个抽象概念，它为每个进程提供了一个假象，即每个进程都在独占地使用主存。

Virtual memory is an abstraction that provides each process with the illusion that it has exclusive use of the main memory.

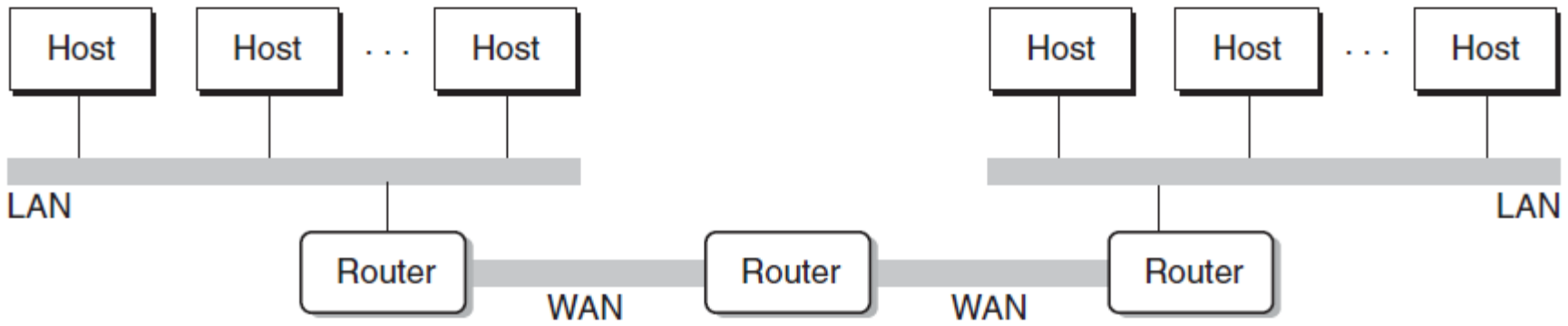
每个进程看到的内存都是一致的，称为*虚拟地址空间*。



系统之间使用网络通信

Systems Communicate with Other Systems Using Networks

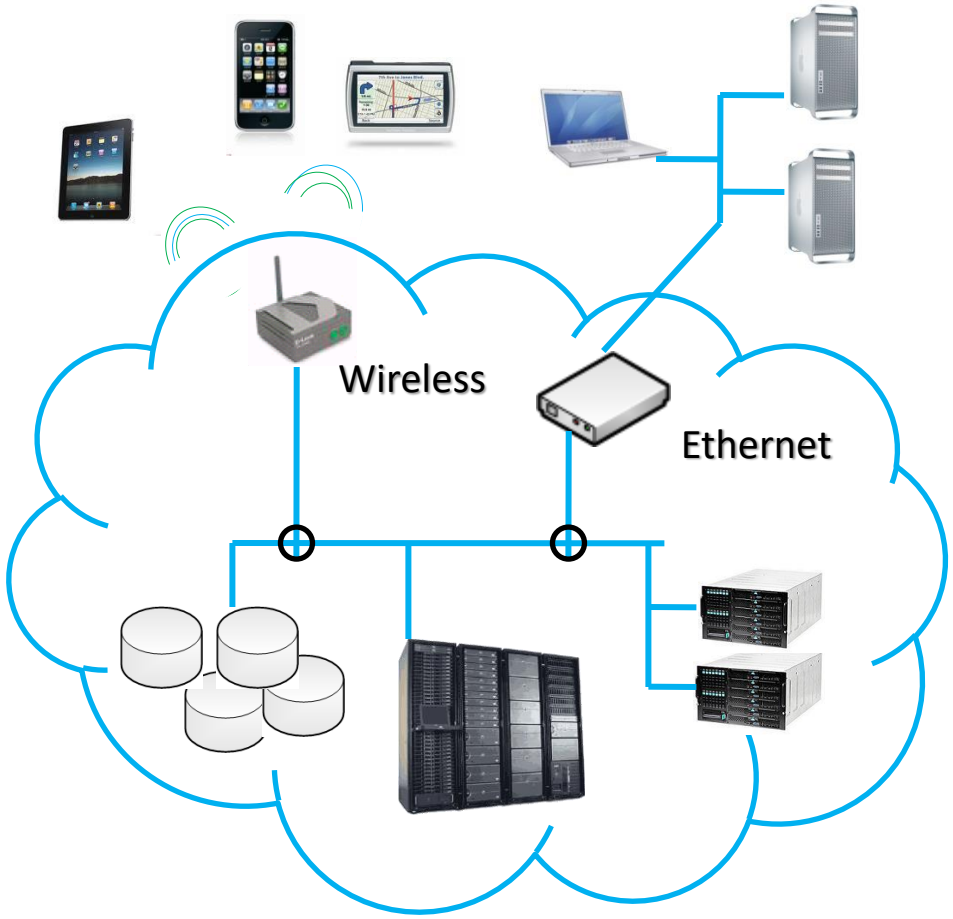
- 网络也是一种I/O设备，计算机之间使用网络进行连接 A network is another I/O device, Computers are connected by networks



云计算 Cloud Computing

北京師範大學 人工智能学院

- 计算机系统支持云计算 Computer Systems support the cloud computing



68

重要主题
Important Themes

北京師範大學 人工智能学院

- Amdahl定律 Amdahl's Law
  - 对系统的某个部分加速时，其对系统整体性能的影响取决于该部分的重要性和加速程度 speed up one part of a system, the effect on the overall system performance depends on both how significant this part was and how much it sped up.

$$S = \frac{1}{(1-\alpha) + \alpha/k}$$

  - 该部分时间占比为α requires a fraction α of this time
  - 该部分性能改进提高k倍 improve its performance by a factor of k.
  - 必须提升在总体系统中占比非常大的部分的速度 must improve the speed of a very large fraction of the overall system.

$$S_\infty = \frac{1}{(1-\alpha)}$$

重要主题/Important Themes

北京師範大學 人工智能学院

- 并发和并行 Concurrency and Parallelism
  - 并发指同时具有多个活动的系统这个通用概念 concurrency refer to the general concept of a system with multiple, simultaneous activities
  - 并行指用并发使系统运行更快 parallelism refer to the use of concurrency to make a system run faster.
  - 并行可以在计算机系统的多个抽象层次上运用 Parallelism can be exploited at multiple levels of abstraction in a computer system.
  - 三个层次，在系统层次结构中从最高到最低级 three levels, working from the highest to the lowest level in the system hierarchy
- 线程级并发 Thread-Level Concurrency
  - 在进程抽象基础上，多个程序同时执行，导致并发 building on the process abstraction, multiple programs execute at the same time, leading to concurrency.
  - 使用线程可以在单一进程中有多个控制流 With threads, have multiple control flows executing within a single process.

重要主题/Important Themes

北京師範大學 人工智能学院

- 线程级并发 Thread-Level Concurrency
  - 从单处理器系统到多处理器系统，最近多核和超线程 from uniprocessor system to multiprocessor system. recently multi-core processors and hyperthreading.
  - 超线程称为同时多线程，是一项允许单一CPU执行多个控制流的技术 Hyperthreading, called simultaneous multi-threading, is a technique that allows a single CPU to execute multiple flows of control.
  - 要求程序必须以多线程方式编写 the program is expressed in terms of multiple threads.
- 指令级并行 Instruction-Level Parallelism
  - 现代处理器可以一次执行多条指令，称为指令级并行 modern processors can execute multiple instructions at one time, known as instruction-level parallelism.
  - 流水线的使用，接近一个时钟周期一条指令的执行速率 use of pipelining, an execution rate close to 1 instruction per clock cycle.

重要主题/Important Themes

北京師範大學 人工智能学院

- 指令级并行 Instruction-Level Parallelism
  - 比一个周期一条指令更快的执行速率，称为超标量处理器 execution rates faster than 1 instruction per cycle, known as superscalar processors.
- 单指令流多数据流（SIMD）并行 Single-Instruction, Multiple-Data (SIMD) Parallelism
  - 单条指令引起并行执行多个操作 a single instruction to cause multiple operations to be performed in parallel
  - 某些编译器尝试自动从C程序抽取SIMD并行性 some compilers attempt to automatically extract SIMD parallelism from C programs,
  - 自己使用编译器支持的特殊向量数据类型编写程序 write programs using special vector data types supported in compilers.

重要主题/Important Themes

北京師範大學 人工智能学院

- 计算机系统中抽象的重要性 The Importance of Abstractions in Computer Systems
  - 抽象的使用是计算机科学中最为重要的概念之一 The use of abstractions is one of the most important concepts in computer science.
- 计算机系统中使用的几个抽象 several of the abstractions in computer systems
  - 指令集体系结构提供实际处理器硬件的抽象 the instruction set architecture provides an abstraction of the actual processor hardware.
  - 操作系统提供三个抽象：文件作为I/O设备抽象、虚存作为程序内存的抽象、进程作为运行程序的抽象 OS provides three abstractions: files as an abstraction of I/O devices, virtual memory as an abstraction of program memory, and processes as an abstraction of a running program.
  - 新抽象：虚拟机提供整个计算机的抽象，包括OS、处理器和程序 a new one: the virtual machine, providing an abstraction of the entire computer, including the operating system, the processor, and the programs.

响应时间和吞吐量
Response Time and Throughput

- 响应时间 Response time
  - 做一项任务所需时间 How long it takes to do a task
- 吞吐量 Throughput
  - 每单位时间完成的总工作量Total work done per unit time
    - 例如任务/事务/...每小时 e.g., tasks/transactions/… per hour
- 响应时间和吞吐量如何受以下因素影响 How are response time and throughput affected by
  - 替换处理器成更快的版本 Replacing the processor with a faster version?
  - 增加更多的处理器？ Adding more processors?
- 现在我们将聚焦关注响应时间 We’ll focus on response time for now…

相对性能 / Relative Performance

北京師範大學 人工智能学院

- 定义性能为：1/执行时间 Define Performance = 1/Execution Time
- “X比Y快n倍” “X is *n* time faster than Y” X的性能/Y的性能 = Y的执行时间/X的执行 = n

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

- 例如：运行程序的时间 / Example: time taken to run a program
  - 10s on A, 15s on B
  - B执行时间/A执行时间 / Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - 所以，A的性能是B的1.5倍 / So A is 1.5 times faster than B

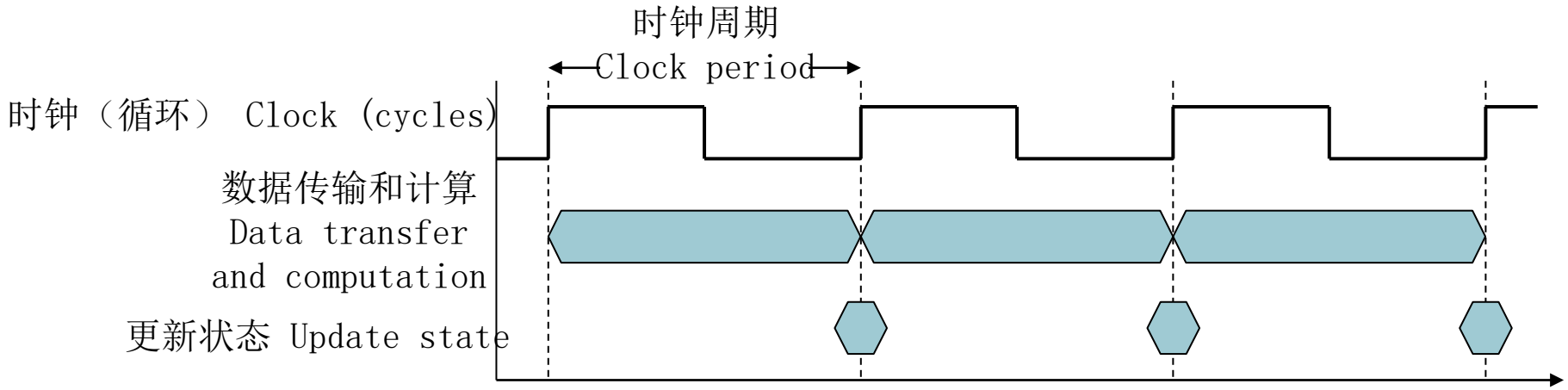
测量执行时间 / Measuring Execution Time

- 经历时间 / Elapsed time
  - 总响应时间，包括所有方面 / Total response time, including all aspects
    - 处理、I/O、操作系统开销、空闲时间 / Processing, I/O, OS overhead, idle time
  - 决定了系统性能 / Determines system performance
- CPU时间 / CPU time
  - 对于指定的作业花费在处理上的时间 / Time spent processing a given job
    - 减掉I/O时间，其它作业共享时间 / Discounts I/O time, other jobs' shares
  - 由用户CPU时间和系统CPU时间构成 / Comprises user CPU time and system CPU time
  - 不同程序受CPU和系统性能的影响不同 / Different programs are affected differently by CPU and system performance

CPU时钟周期 / CPU Clocking

北京師範大學
人工智能学院

- 数字硬件的操作受固定时钟速率控制 / Operation of digital hardware governed by a constant-rate clock



- 时钟周期：一次时钟循环持续时间 / Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- 时钟频率（速率）：每秒钟的周期数 / Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU时间 CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- CPU时间=CPU时钟周期数X时钟周期时间=CPU时钟周期数/时钟速率

- 性能改进手段 Performance improved by
  - 减少时钟周期数量 Reducing number of clock cycles
  - 增加时钟速率 Increasing clock rate
  - 硬件设计师必须经常在时钟速率和周期数之间进行折中 Hardware designer must often trade off clock rate against cycle count

CPU时间示例 CPU Time Example

- 计算机A：2GHz时钟频率，10秒CPU时间 Computer A: 2GHz clock, 10s CPU time
- 设计计算机B / Designing Computer B
  - 目标达到6秒CPU时间 Aim for 6s CPU time
  - 可以做到更快的时钟，但是会导致需要1.2倍的时钟周期数 Can do faster clock, but causes 1.2 × clock cycles
- 计算机B的时钟必须达到多快？ How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

指令数和CPI Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- 时钟周期数=指令数 X 每条指令的时钟周期数
- CPU时间=指令数 X CPI X 时钟周期时间=指令数X CPI /时钟速率
- 程序的指令数 Instruction Count for a program
  - 由程序、ISA和编译器决定 Determined by program, ISA and compiler
- 平均CPI Average cycles per instruction
  - 由CPU硬件决定 Determined by CPU hardware
  - 如果不同指令有不同的CPI If different instructions have different CPI
    - 平均CPI受指令混合程度影响 Average CPI affected by instruction mix

CPI示例 CPI Example

北京師範大學 人工智能学院

- 计算机A：时钟周期=250ps，CPI=2.0 Computer A: Cycle Time = 250ps, CPI = 2.0
- 计算机B：时钟周期=500ps，CPI=1.2 Computer B: Cycle Time = 500ps, CPI = 1.2
- 同样的 ISA Same ISA
- 哪个更快，快多少？Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

…by this much

CPI更多的细节 CPI in More Detail

北京師範大學 人工智能学院

- 如果不同类型指令需要不同的时钟周期数/If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- 加权平均CPI为 Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{相对比率 Relative frequency}} \right)$$

CPI示例 CPI Example

北京師範大學 人工智能学院

- 替换编译的代码序列使用A、B、C类指令/Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- 序列1： Sequence 1: IC = 5
  - 时钟周期 Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- 序列2： Sequence 2: IC = 6
  - 时钟周期 Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

性能小结 Performance Summary

北京師範大學 人工智能学院

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructio ns}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instructio n}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- CPU时间=每个程序的指令数 X 每条指令的时钟周期数 X 每个时钟周期的时间

- 性能取决于 Performance depends on
  - 算法：影响IC，可能影响 CPI Algorithm: affects IC, possibly CPI
  - 编程语言：影响IC、CPI Programming language: affects IC, CPI
  - 编译器：影响IC、CPI Compiler: affects IC, CPI
  - 指令集体系结构：影响IC、CPI和时钟周期 Instruction set architecture: affects IC, CPI, T_c

MIPS和性能度量
MIPS and Performance Metric

北京師範大學 人工智能学院

- MIPS：每秒钟执行百万条指令 MIPS: Millions of Instructions Per Second
  - 并没有计算 Doesn't account for
    - 不同计算机之间的ISA不同 Differences in ISAs between computers
    - 不同指令之间的复杂度不同 Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- 在特定的CPU上在不同程序之间CPI是变化的 CPI varies between programs on a given CPU