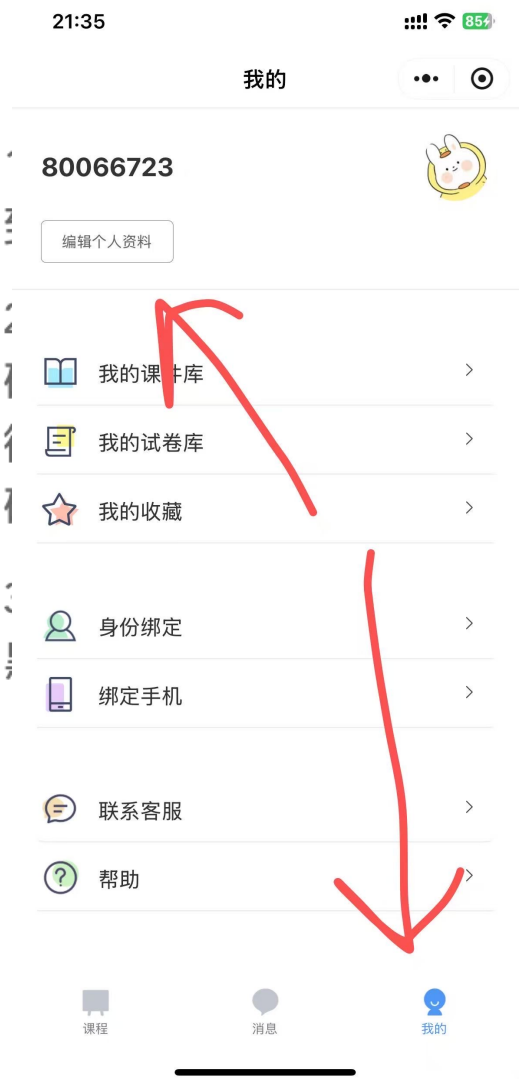
该二维码7天内(9月14日前)有效，重新进入将更新

计算机系统基础I

谢旻晖

2025.9

雨课堂编辑个人资料，修改姓名学号



目录

- 教师、助教信息
- 教材和成绩构成
- 学习动机

授课教师

谢旻晖

Email: xieminhui@ruc.edu.cn

电话/微信: 18801119418

办公室：信息楼505

研究方向：机器学习系统、存储系统

博士：清华大学计算机系（2019-2024）

本科：南京大学计算机系（2015-2019）

云计算与大数据系统实验室CDSLab（信息楼124/125/505/239）

2025-2026学年助教

助教信息

为了方便同学们联系助教，~~也为子避免大家在xlb上不知道该喷哪个助教。~~，我们整理了 2025 ICS I 的全部助教的名单。

需要注意的是，虽然 ICS I 一共有三个班级，但是今年为了更好地协调工作，助教并不是直接绑定到某一个班的。大家直接按照本文档中的信息联系助教即可。

助教联系方式

推荐直接通过微信联系助教，以下是各个助教的微信头像：

2025-2026学年助教

如果出于各种原因，需要通过邮箱联系助教，请参考如下助教邮箱列表：

助教姓名	邮箱
彭文博	2022201555@ruc.edu.cn
马亚辛	2023202293@ruc.edu.cn
李甘	2023202296@ruc.edu.cn
方言诚	2023202302@ruc.edu.cn
卢虹宇	2023202269@ruc.edu.cn
张昕跃	2023202300@ruc.edu.cn
文博正	2023202287@ruc.edu.cn
夏维汉	2023202226@ruc.edu.cn
方瑾茗	2023201243@ruc.edu.cn

如果需要联系某位助教但联系不上，请联系李甘（微信：nictheboy，手机：136 8158 2912）

2025-2026学年助教

各工作负责人

以下是助教们的分工列表，大家也可依照此表联系助教：

板块	工作	主要负责助教	协助助教
Data	Lab	**方瑾茗**	马亚辛
Data	作业	**文博正**	方瑾茗
Data	复习	**方言诚**	李甘
Asm	Lab (Bomblab)	**卢虹宇**	文博正
Asm	作业	**张昕跃**	李甘
Asm	复习	**马亚辛**	方言诚
Cache	Lab	**张昕跃**	方言诚
Cache	作业	**文博正**	卢虹宇
Cache	复习	**马亚辛**	夏维汉
Link	Lab	**彭文博**	张昕跃
Link	作业	**彭文博**	方瑾茗, 夏维汉
Link	复习	**李甘**	卢虹宇

注1. 一般环境问题请联系李甘

注2. 当你不知道该联系谁，或被联系的助教非常繁忙时，请联系夏维汉

- 往年，助教常常被大量重复的问题所困扰。
- 因此，他们不得不在群里反复发送消息，以广播某个常见问题——或者索性草率地回复。

为了避免该情况，助教们为大家提供了 *答疑吐槽灌水论坛*

匿名提问的方式既避免尴尬，也分享公共问题的回答

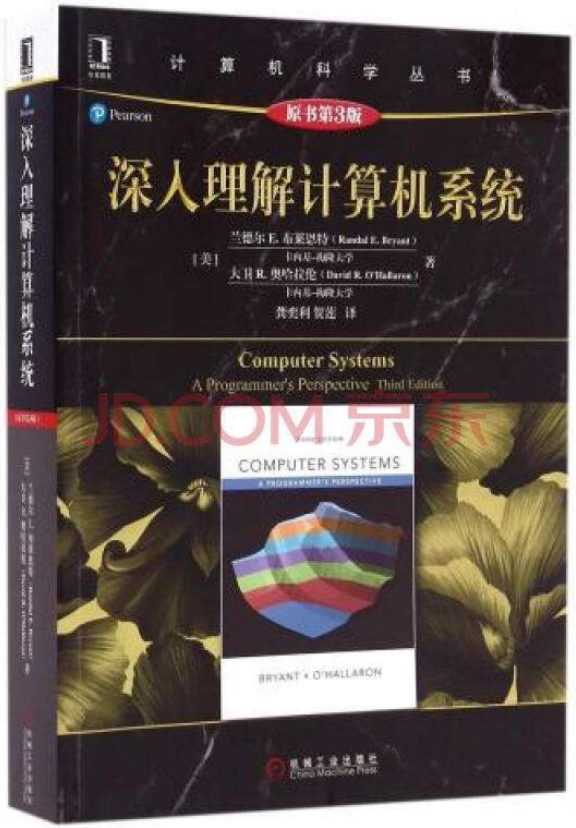
forum.rucics.tech 校园网访问 ruc邮箱注册

上机安排

- **周三11-13节：下午6:00～8:30，明德地下H**
  - 上机实验
  - 测试
  - 习题讲解/答疑
- 实验内容
  - Lab1: DataLab
  - Lab2: BombLab
  - Lab3: CacheLab
  - Lab4: LinkLab

教材

- Randy Bryant and David O'Hallaron,
  - Computer Systems: A Programmer's Perspective
  - 3nd Edition
  - 《深入理解计算机系统》（原书第3版）



成绩构成

- 平时成绩(70%)
  - 期中考试 (40%)
  - Labs (40%)
    - Don't cheat
    - Start early
  - Homework (10%)
  - 课堂表现 (10%)

- 期末考试(30%)

系统能力课程徽章

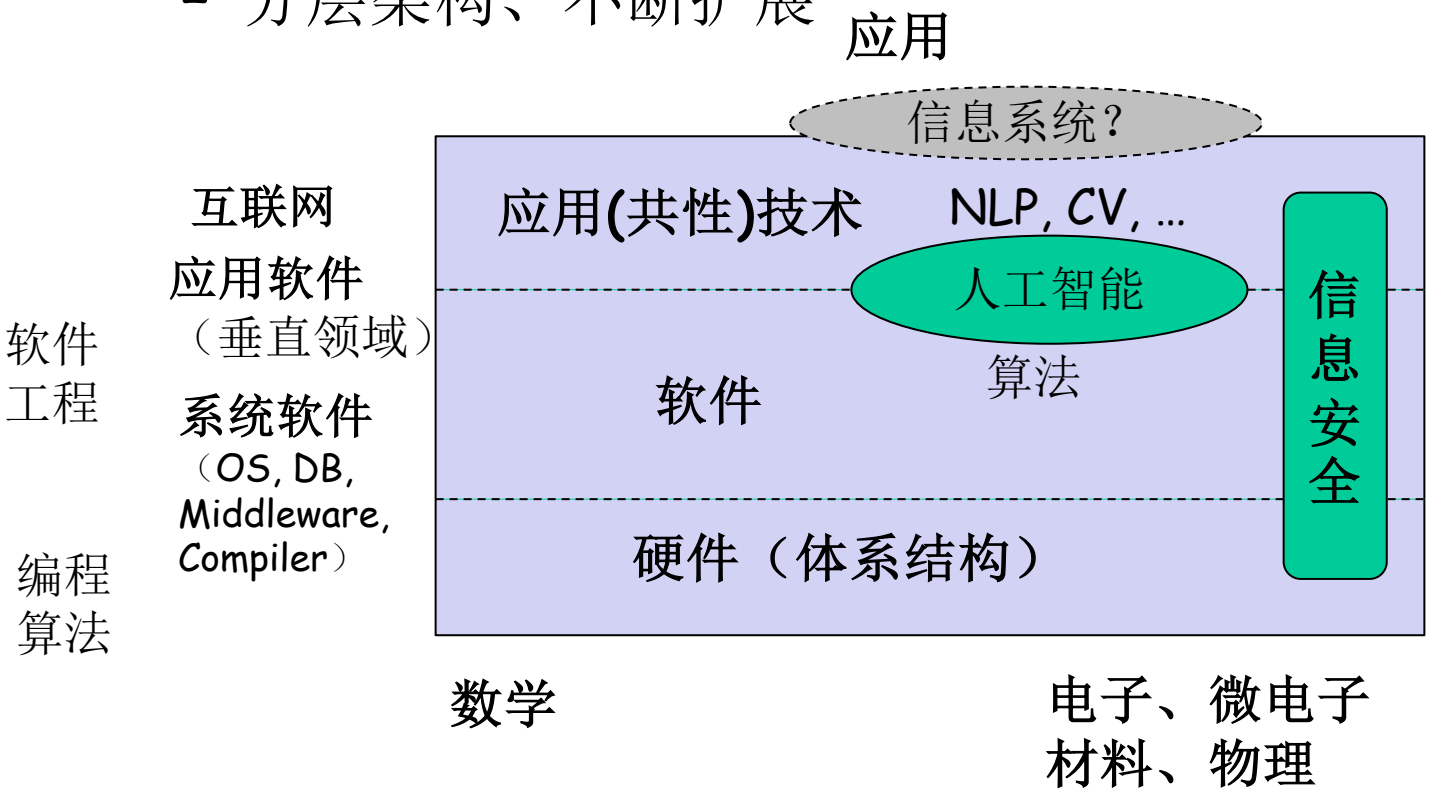

荣誉证书
张三同学参加并通过了信息学院“系统能力培养”系列课程7门，予以7枚系统能力培养荣誉徽章。
中国人民大学信息学院
2022年3月11日
徽章墙

Motivation

- 为什么要学习这么课？
  - 计算机大类（计算机/软工/大数据/信安/人工智能）专业三大基础课
    - 程序设计I和II (*)
    - 数据结构与算法I和II（->应用，含AI各类应用）
    - 计算机系统基础I和II（->复杂工程/系统）
    - **本科课程前半场**：以上 +数学（->理论）
    - **本科课程后半场**：更复杂/更细分的专业课
    - **外场**：竞赛、科研（拔尖人才）
  - 计算机底层/硬件相关课程已经大幅减少
    - 数电、汇编、组成、体系结构、微机原理（嵌入式）

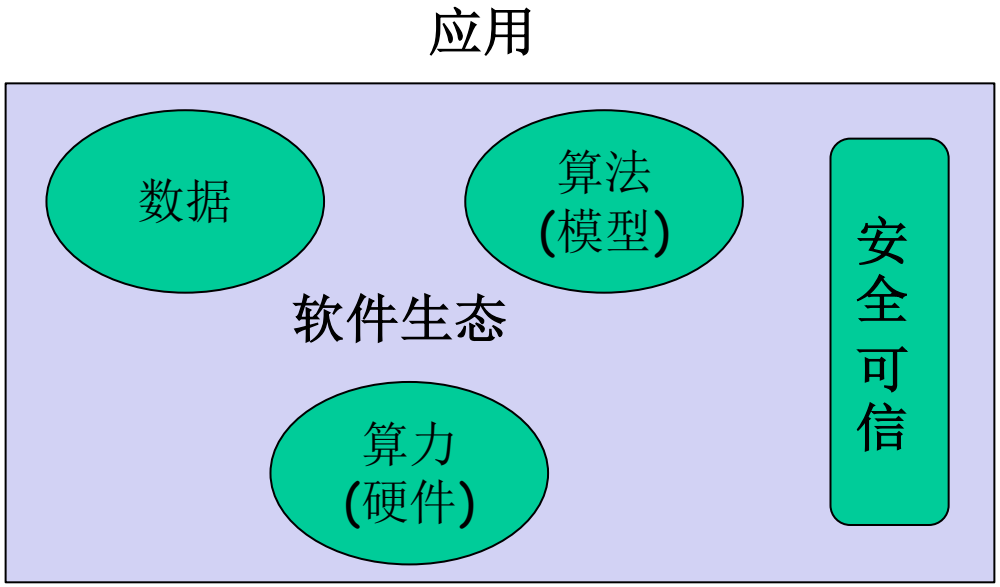
Motivation

- 怎么理解计算机科学与技术学科（!=专业）？
  - 分层架构、不断扩展



Motivation

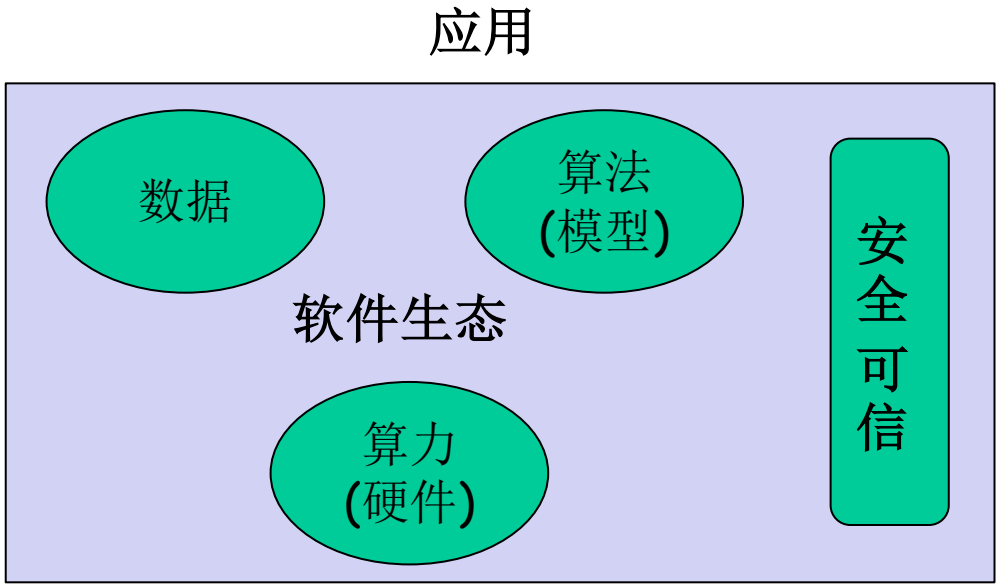
- 怎么理解计算机科学与技术学科（!=专业）？
  - 时代：计算->数据->智能
  - (另一种视角)三要素：算法、算力、数据



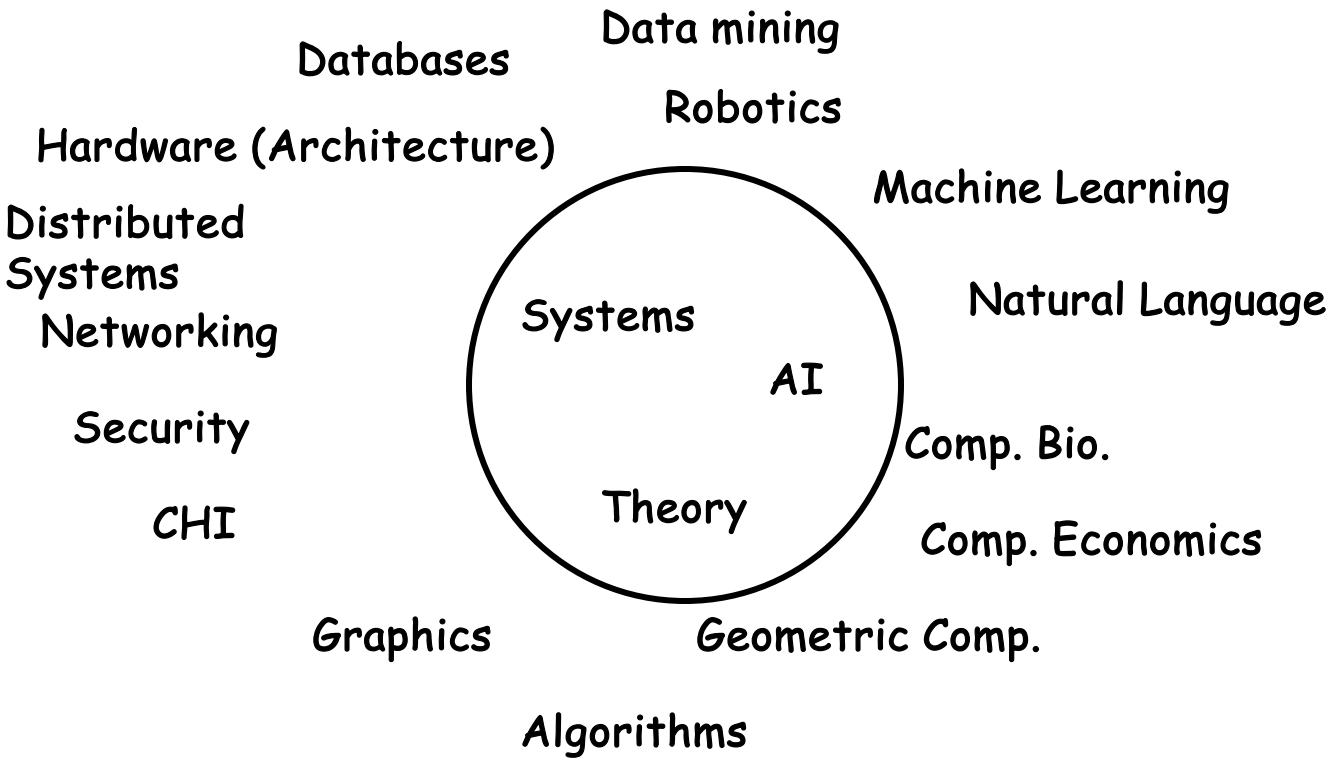
Motivation

- 怎么理解计算机科学与技术学科（!=专业）？
  - 时代：计算->数据->智能
  - (另一种视角)三要素：算法、算力、数据

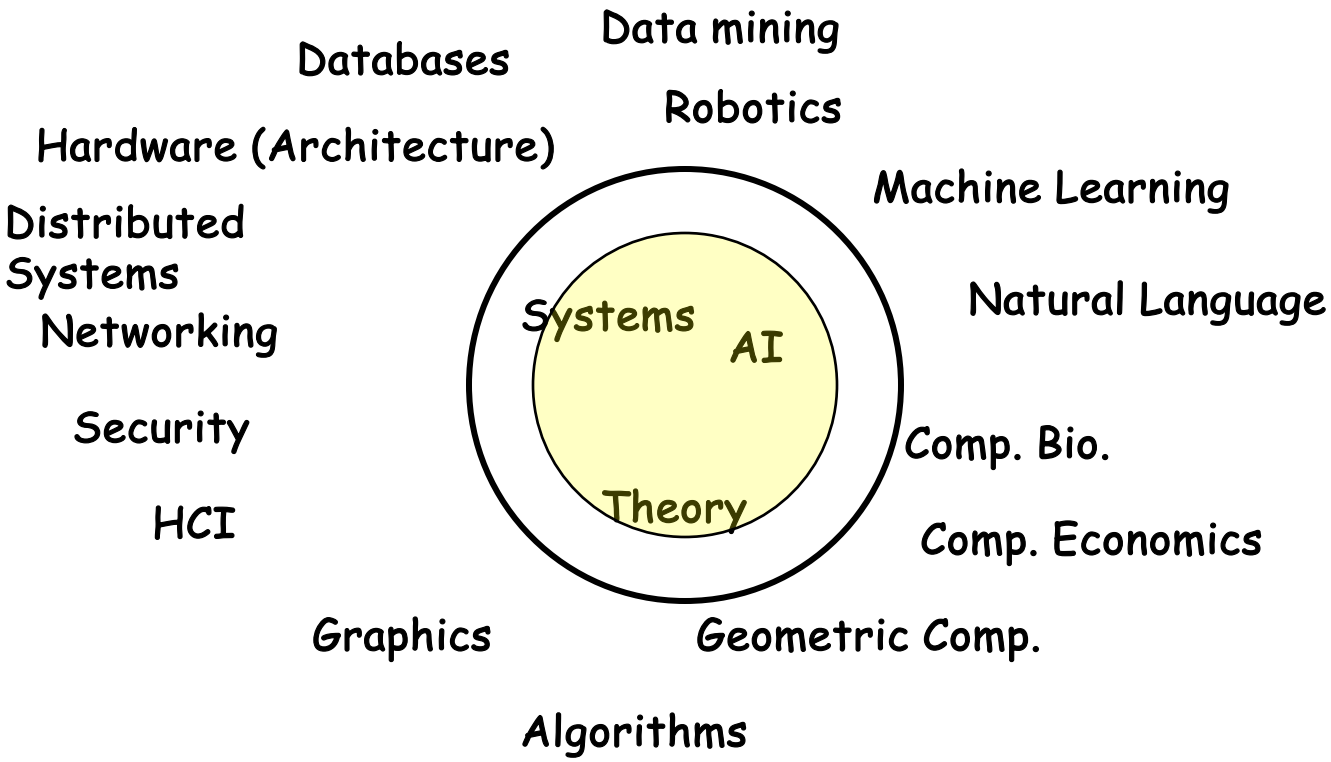
计算机系统
- 算力
- 软件生态(OS, 数据管理、AI软件栈)
- 服务于数据、AI



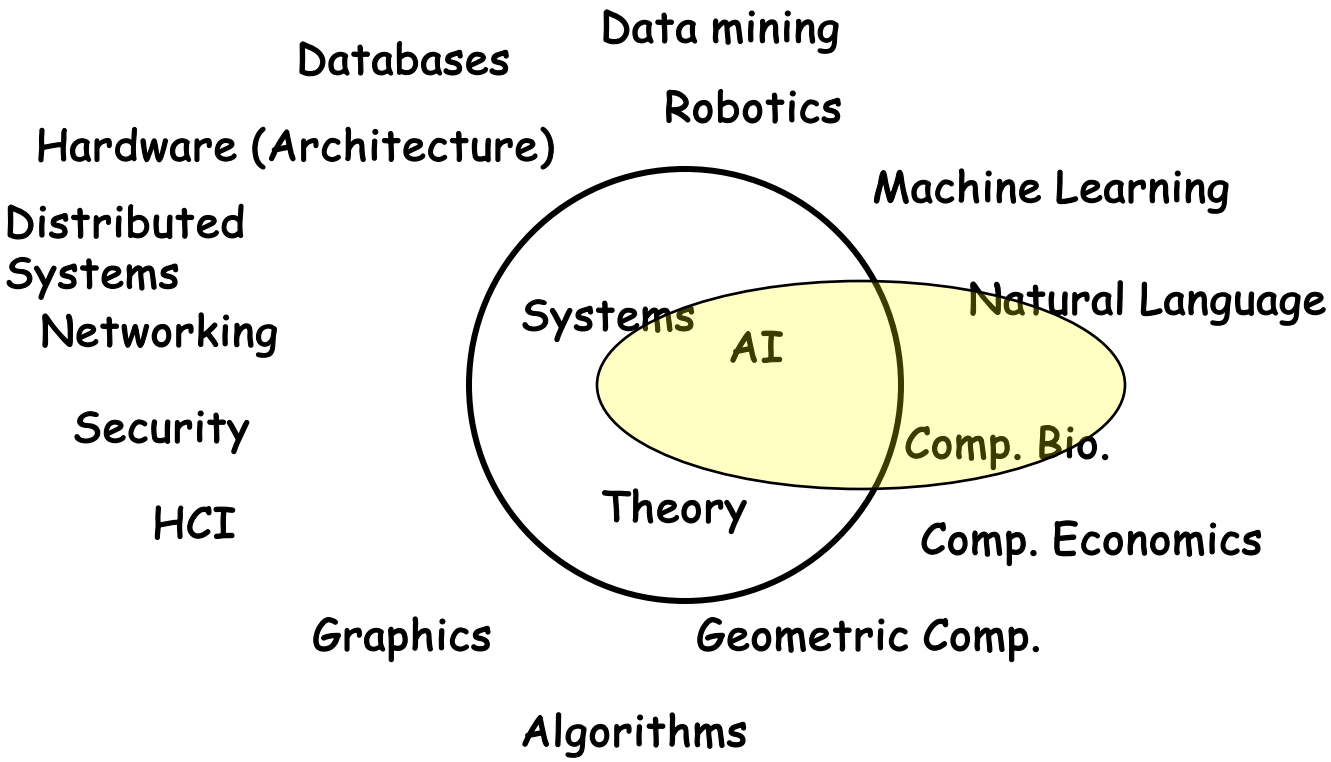
CS内涵


Data mining
Databases
Robotics
Hardware (Architecture)
Machine Learning
Distributed Systems
Natural Language
Systems
Networking
AI
Security
Comp. Bio.
Theory
CHI
Comp. Economics
Graphics
Geometric Comp.
Algorithms

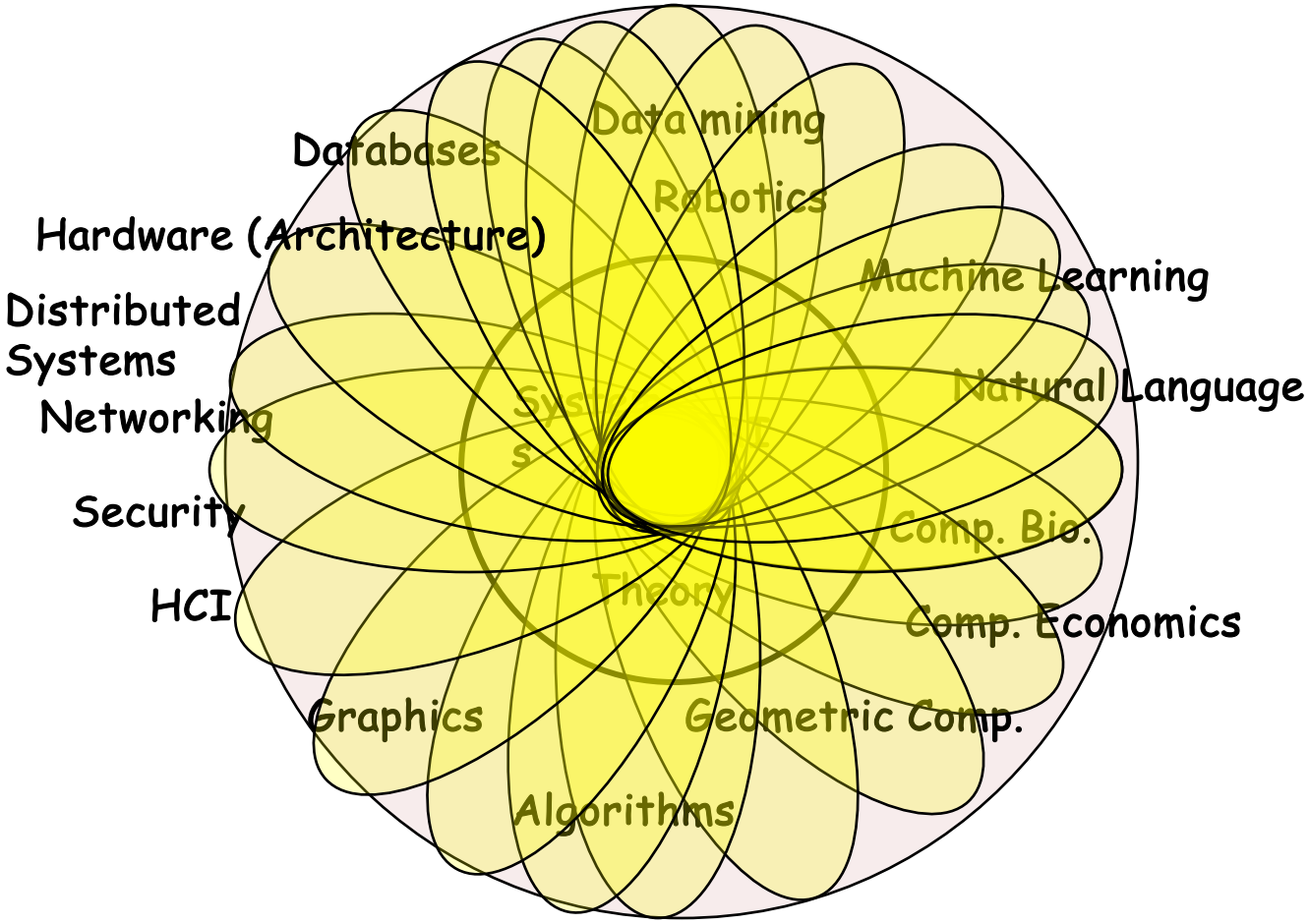
三大基础（传统计算机教育）



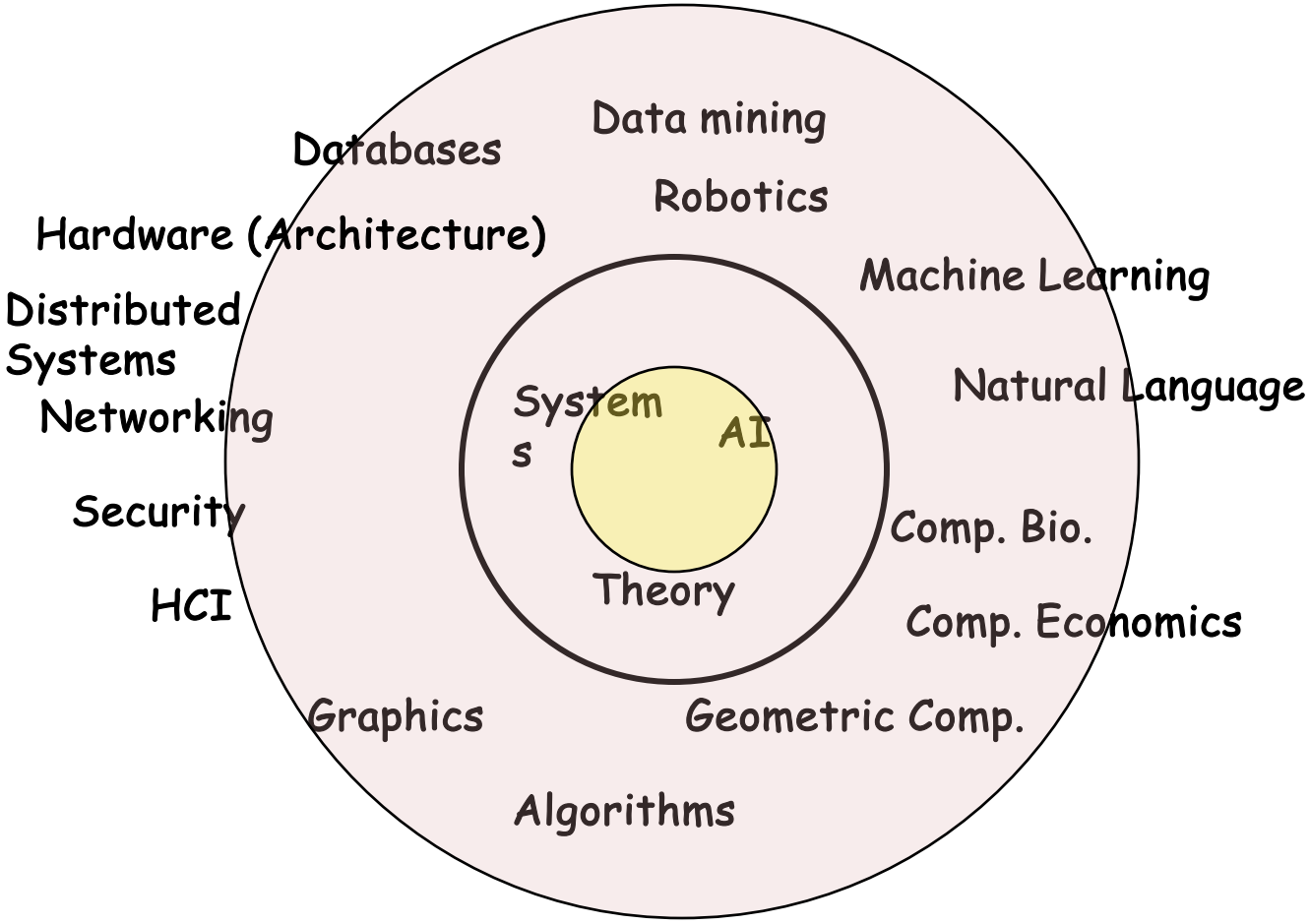
各具体方向越来越成熟、重要



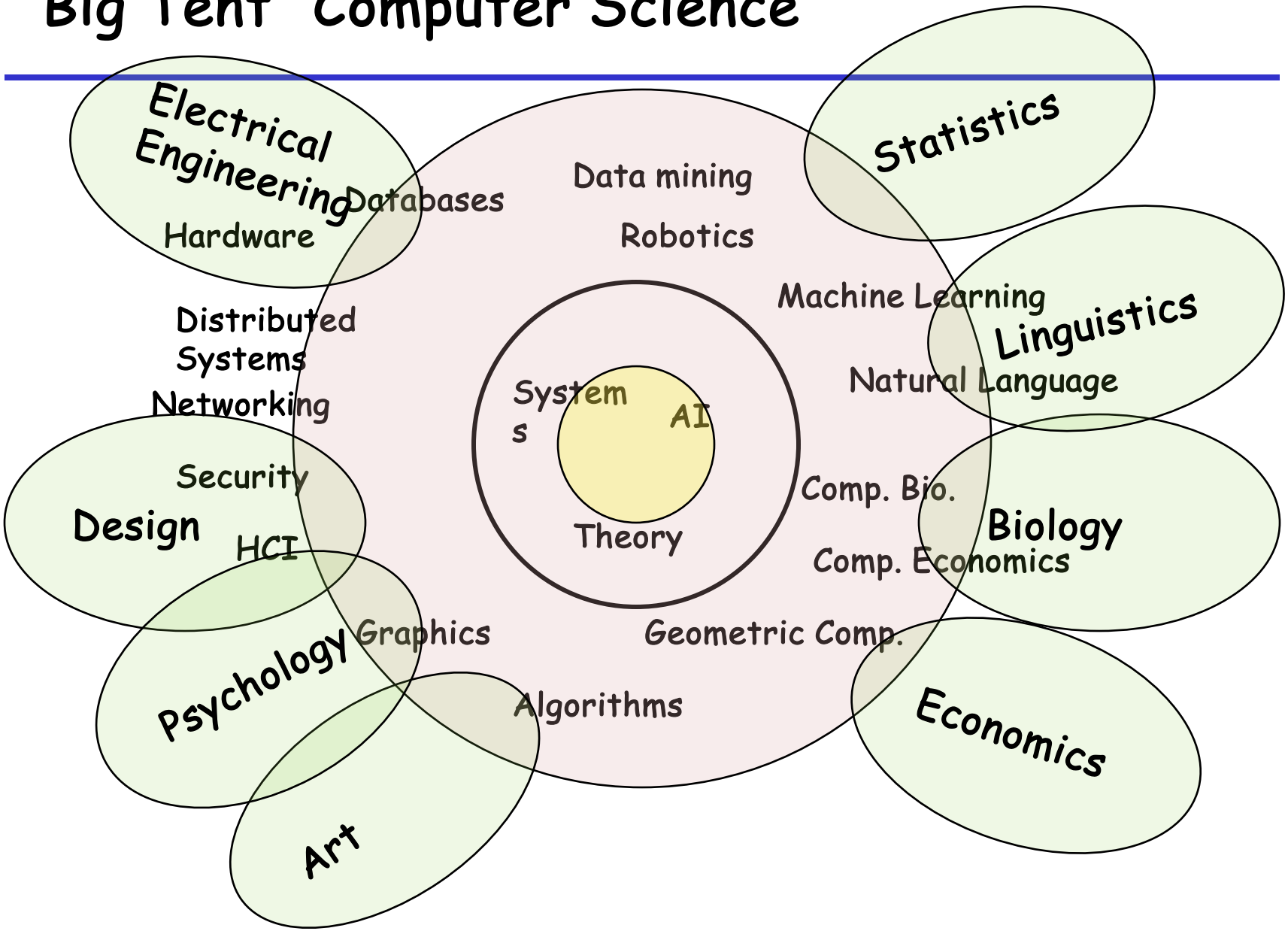
现代计算机教育范围已然扩大



Total Potential Footprint is Larger

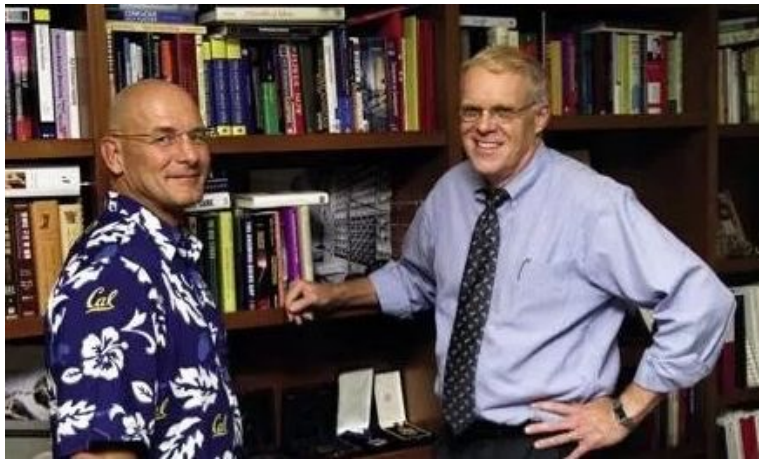


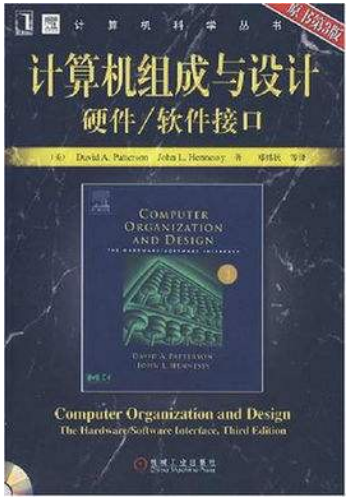
"Big Tent" Computer Science



Systems方向

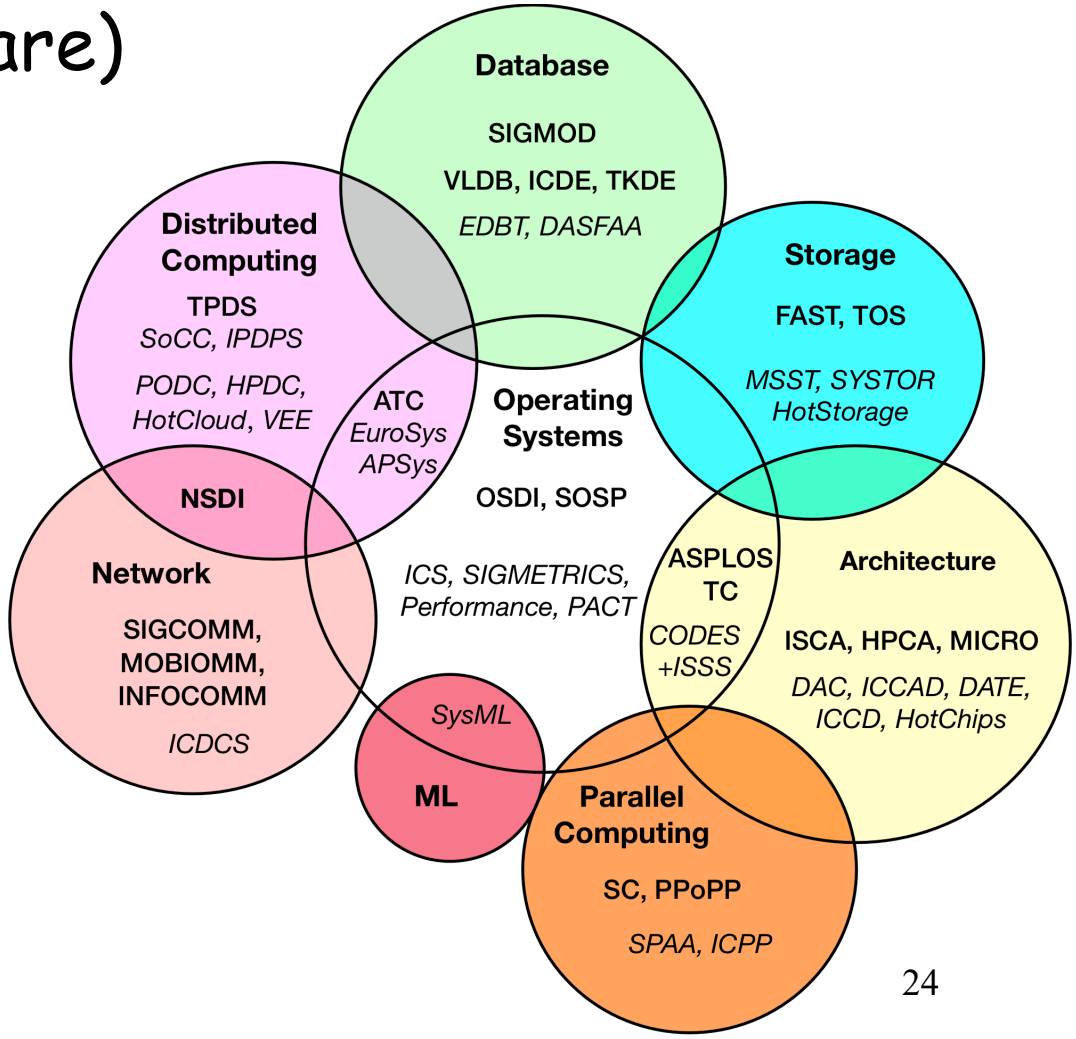
- Systems (Software)
- Architecture (Hardware)
  - 计算机组成原理（计算机，尤其是CPU的构造）
  - 数字电路
  - （模拟电路、电路原理）
  - （计算机系统结构）





Systems方向

- Systems (Software)
  - 单机系统：
    - 汇编语言
    - 操作系统
    - 编译原理
    - 数据库系统
  - 分布式系统：
    - 计算机网络
    - 分布式系统
    - 并行计算



为什么要学习计算机系统？

- 从Programmer角度看
  - 计算机是一个黑盒，接受指令（C, Java, Python, …）
  - 学会编程和算法就可以了，有些公司面试也只考这些(软工)
  - 以上认识建立在一个假设基础上：
    - 计算机黑盒是完美的，高性能、很智能、不出错
- 但实际上计算机系统并不完美（Abstraction Is Good But Don't Forget Reality）
  - 学习计算机系统课程，主要是要了解其局限性（程序出Bug、性能出问题等）
  - 成为更好的Programmer
  - 例1. HAWQ, TiKV（国内主导的大型开源项目；汇编调试）
  - 例2. 磁盘/SSD/NVM的性能特征

为什么要学习计算机系统？

- Ready Player One
  - 抽象vs.现实

Examples

Is $x^2 \geq 0$?

Float's: Yes!

Int's:

40000 * 40000 = [填空1]

50000 * 50000 = [填空2]

Is (x + y) + z = x + (y + z)?

Unsigned & Signed Int's: Yes!

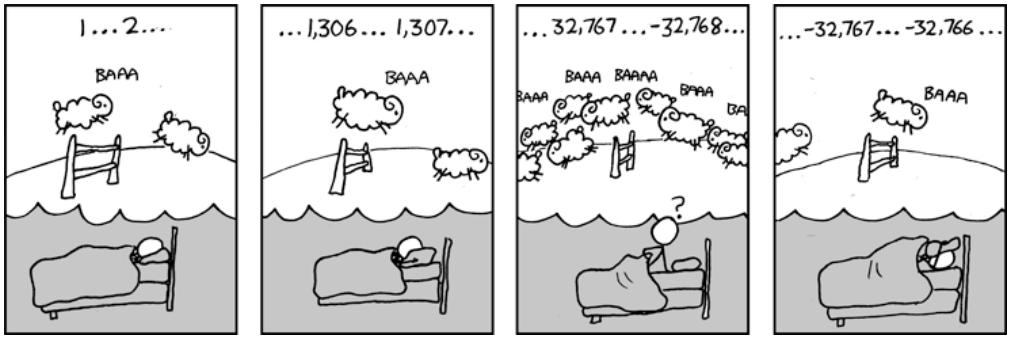
Float's:

(1e20 + -1e20) + 3.14 = [填空3]

1e20 + (-1e20 + 3.14) = [填空4]

Great Reality #1: Ints are not Integers, Floats are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!



  - Int's:
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??
- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Great Reality #2: You've Got to Know Assembly

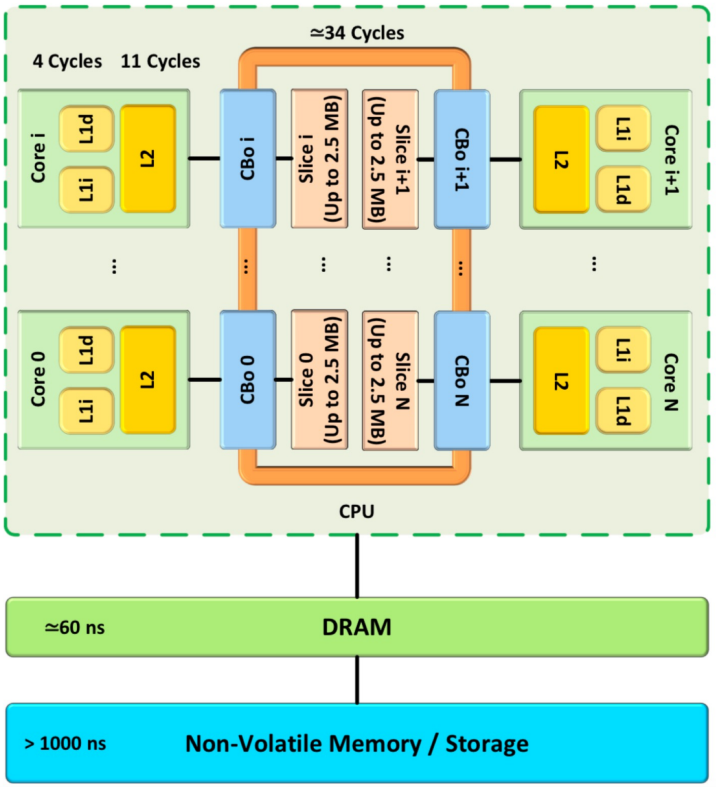
- 一般很少有机会用汇编编程（不超过5行）
  - Compilers are much better & more patient than you are
- 但是：理解汇编是进阶的关键，理解机器级别的执行模型
  - 理解很多Bug的原因
    - High-level language models break down
  - 系统性能调优
    - 编译器/人工智能的局限性
    - 理解系统低效的原因
  - 实现系统软件（OS、Storage、DB）
    - 系统软件不能只用高层抽象，要管理到具体的进程、线程、共享内存等系统底层的状态，甚至控制硬件Cache刷回内存（clflush, mfence）
  - 创造/防御恶意软件
    - 需要使用x86汇编（Intel CPU普及）

Great Reality #3: Memory Matters
Random Access Memory Is an Unphysical Abstraction

- 内存不是自来水
  - 内存需要分配和管理（存在开销）
  - 很多应用是memory-dominated
- 内存访问的bug尤其有害
  - 在时间和空间上的表现可能很不相同
- 内存的性能也不是固定不变的
  - Cache, virtual memory, NUMA对程序的性能很大的影响
  - 程序如果适应硬件内存系统，性能会有很大的提升（系统优化的重要方面）

Great Reality #3: Memory Matters
Random Access Memory Is an Unphysical Abstraction

- An Example
  - 网络速度越来越快，DRAM速度已经跟不上了
    - 例如100G的高速网络中，接受64B的网络包，处理时间只有5.2ns，而DRAM的访问延迟一般是在60ns。这时候只有LLC的延迟可以匹配，34 cycles
  - LLC的访问延迟并不是均匀的，由若干个slice组成（每个slice 2.5MB），各个core访问不同的slice延迟有所不同



Memory Referencing Bug Example

```
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)  →  2.00000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

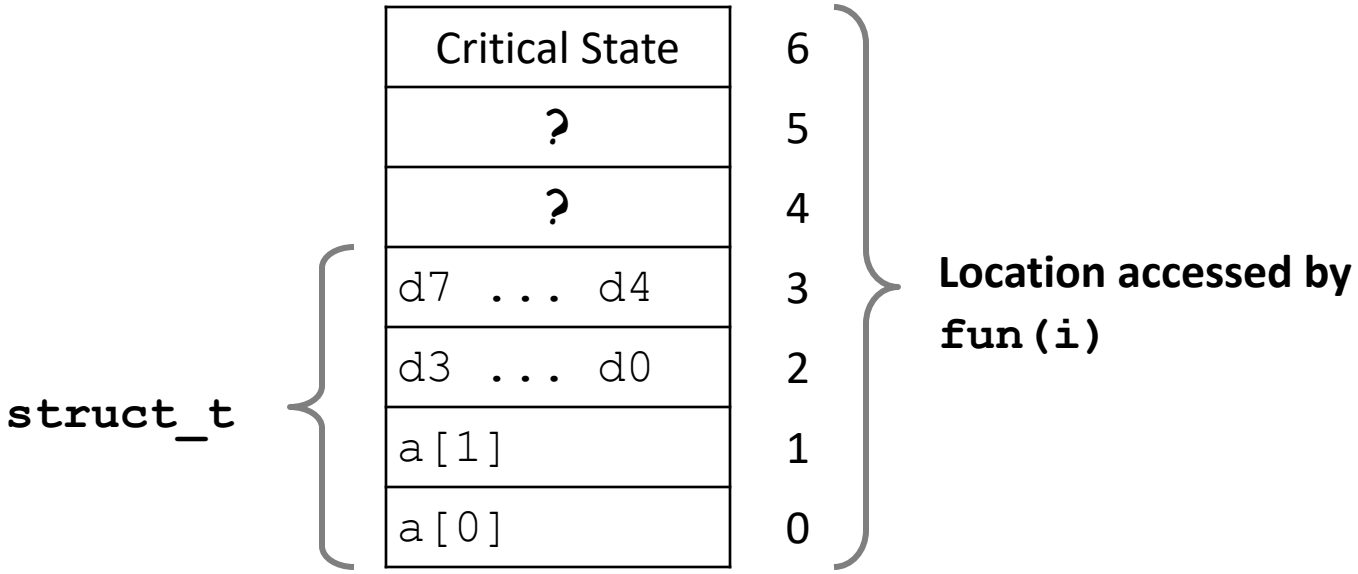
- Result is system specific

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)  →  2.00000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

Explanation:



Memory Referencing Errors

- C和C++不提供内存保护
  - 经常发生数组越界
  - 非法指针
  - 滥用malloc/free
- 可能导致严重的bug
  - 同一段代码，是否发生bug，还取决于系统和编译器
  - 而且可能过一段代码才发生bug，不易调试
- 怎么办？
  - 用更高级的语言编程：Java, Ruby, Python, Go, …（但效率和控制力也在下降）
  - 理解可能会发生什么
  - 使用或者开发工具来调试内存错误 (e.g. Valgrind)

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

- Constant factors matter too!
- 知道op数量，也很难准确预测系统性能
  - 代码写的好不好，系统性能差10倍很正常（每年提升30%）
  - 必须在多个层次优化性能：algorithm, data representations, procedures, and loops
- 必须理解系统才能优化性能
  - 理解程序如何编译和执行
  - 理解如何测量系统性能，找出瓶颈（perf, DTrace等）
  - 理解如何在不破坏代码模块性和通用性的前提下优化性能

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];
}
```

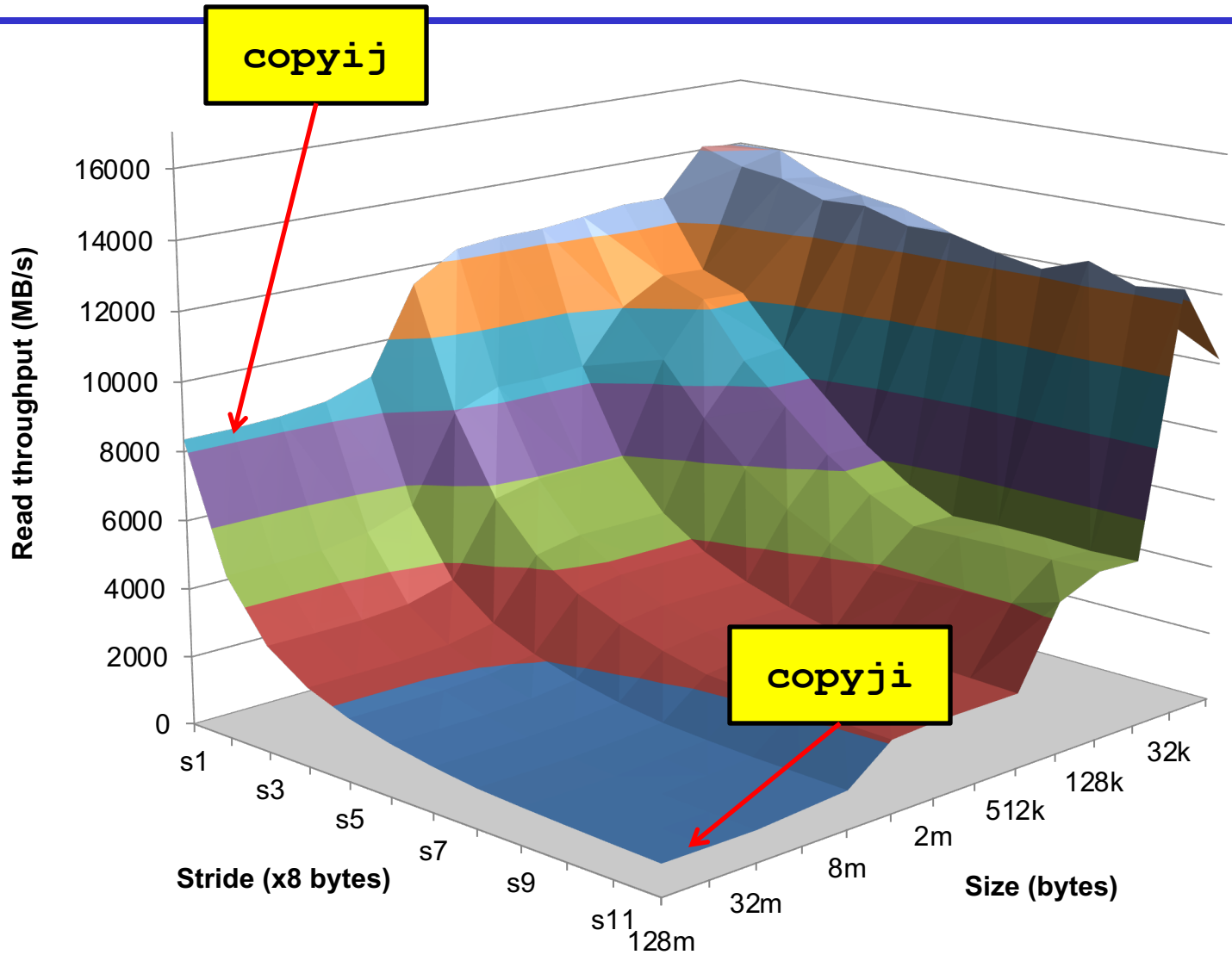
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms **81.8ms**

2.0 GHz Intel Core i7 Haswell

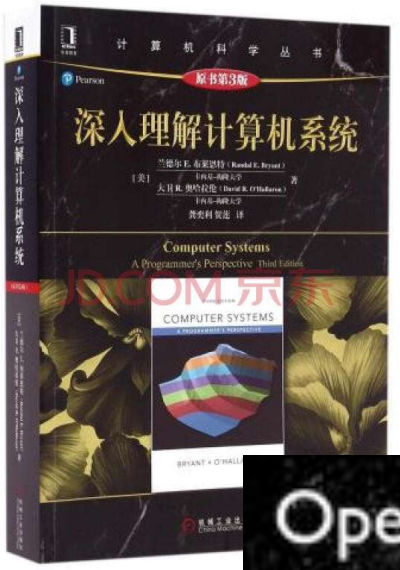
- Hierarchical memory organization
- Performance depends on access patterns
  - Including how step through multi-dimensional array

Why The Performance Differs



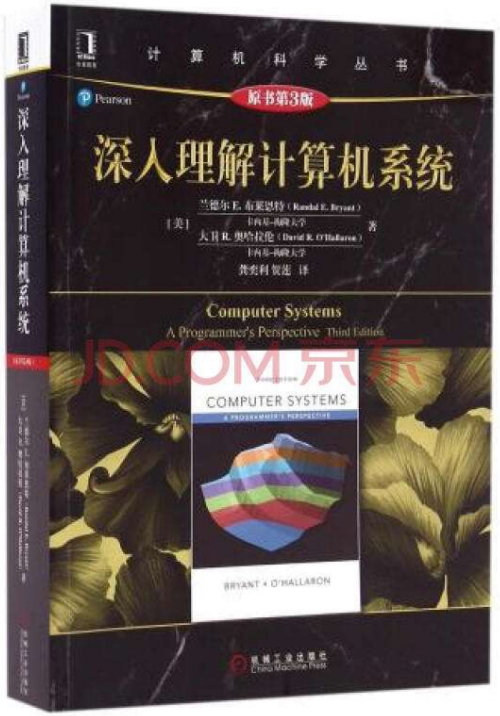
CS:APP

- Computer Systems: A Programmer's Perspective
  - 理解C语言下面的内容
  - 数字的表示和运算方法
  - 汇编语言（与C的对应，性能优化）
  - CPU的工作原理
  - 程序性能优化
  - 操作系统（硬件虚拟化、持久存储、并发）
    - + OS:TEP
  - 网络（TCP/IP）



CS:APP

- Computer Systems: A Programmer's Perspective
  - 两学期，7学分 (3+4)
  - 计算机系统基础、操作系统荣誉课程
  - 两大特点
    - 从程序员角度理解，有趣、有用
    - 平台课程，上面可以进一步深入学习具体方向，自由组合课程
      - 现代处理器设计
      - 编译原理
      - 并行与分布式系统
  - 其他高校经验（教育部101计划）
    - CMU, 北大、清华姚班、复旦、上交、浙大



Systems推荐书合辑

- 底层基础：CSAPP
  - 深入：Patterson&Hennessy，Hennessy&Patterson
  - CPU设计，CA量化研究方法
- 操作系统：OSTEP+CSAPP
  - 深入：MIT 6.828
- 分布式系统：DDIA+MIT 6.824
- AI系统：UCB CS294，…

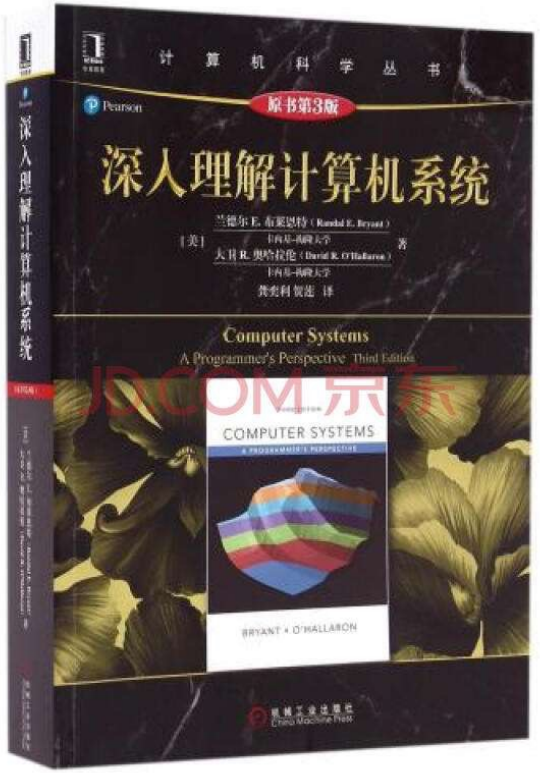
“计算机系统入门”

CS:APP

- 1. 计算机系统概述
- 2. 信息的表示和处理
- 3. 程序的机器级表示（汇编语言）+ CPU构造
- 4. 存储层次结构
- 5. 程序性能优化
- 6. 链接*
- 7. 进程和CPU调度
- 8. 虚拟内存
- 9. I/O和文件系统
- 10. 计算机网络
- 11. 并发编程

CS:APP

- 计算机系统基础I
  - Lab1: DataLab
    - C语言位操作实现算数逻辑运算
  - Lab2: BombLab
    - 汇编语言，二进制炸弹
  - Lab3: CacheLab
    - 缓存模拟和程序面向缓存优化
  - Lab4: LinkLab
    - 链接器实验



CS:APP

- 计算机系统基础II
  - Lab5: ShellLab
    - 写一个自己的Unix Shell，考虑多进程并发
  - Lab6: ScheLab
    - CPU调度策略设计（排名）
  - Lab7: MallocLab
    - 写自己的malloc包，比较性能和效率
  - Lab8: FSLab
    - 写能实际使用的FUSE文件系统
  - Lab9: NetLab
    - Socket编程、并发编程

注意学习长效的知识

- 计算机软硬件系统的具体实现不断变化
- 但是背后的有很多知识和观点是长期稳定的
- 采用类似原理的系统，运行的性能是差不多的
- 例子：
  - 各种Cache加速；
  - 以存代算、以算代存；
  - 冗余来提高可用性、可靠性；
  - Leader-Follower保持数据一致性

学做系统架构师/设计者

- 学生很少有机会设计和构造一个复杂系统
  - 成本太高（CPU、OS、编译器）
  - 没有积累和经验，也很难设计出好的系统
- Case-by-case的学习方式，参考经典系统
  - 操作系统
    - 文件系统
  - CPU
  - 分布式系统
  - 开源项目

学习方法

- 理解（听课/阅读教材和参考资料）
- 练习（习题/程序段）
- 实验

- 不是一门轻松的课程，但是本科阶段最有用的少数几门课程之一

Good luck!