西安交通大學
XI'AN JIAOTONG UNIVERSITY

ANTS
Advance Your Network

# 计算机系统导论

**李昊**

西安交通大学计算机学院

智能网络与网络安全教育部重点实验室

# 教学团队

李昊 副教授

hao.li@xjtu.edu.cn

兴庆校区 彭康楼234

单丹枫 副教授

dfshan@xjtu.edu.cn

创新港 泓理楼6078

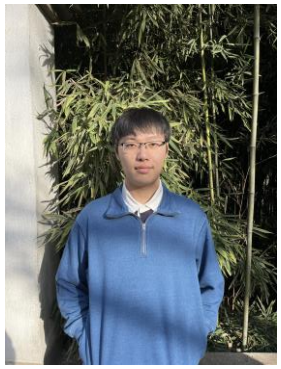
# 助教团队


**党一涵 研二 助教组长**

USENIX NSDI 学生一作
研究生学分绩3/157
本科总学分绩3/32

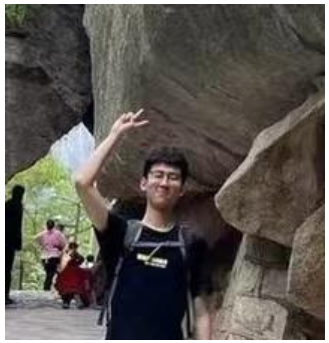
**杨舜磊 研一**

本科总学分绩6/171
校级一等奖学金
深交所奖学金



**夏泽 研0**

2次ACM ICPC银牌
1次ACM CCPC银牌
CCP认证400分



**李云广 研0**

本科前三年学分绩90+

# 教材与参考书

核心教材与原始Slides

深入理解计算机系统 第三版

Computer Systems: A Programmer's Perspective 3rd

# 教材与参考书

**核心教材与原始Slides**

**深入理解计算机系统 第三版**

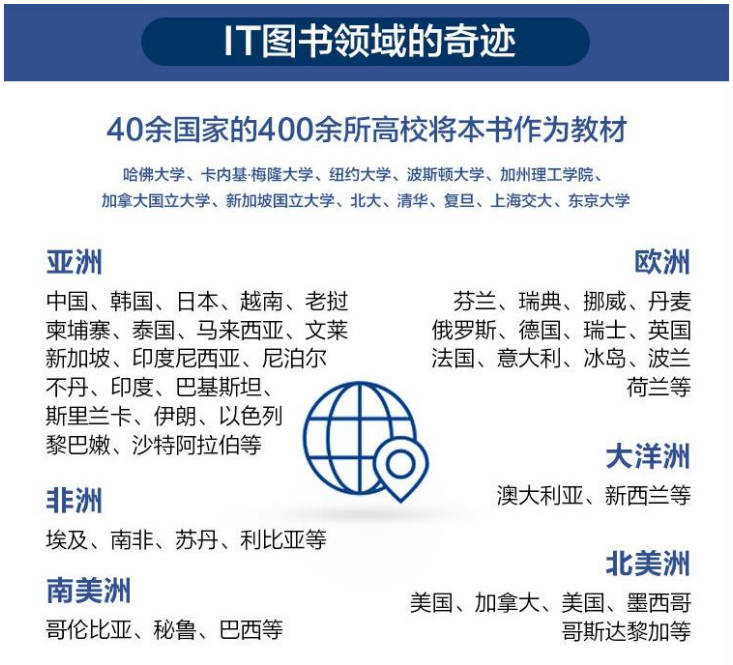
**Computer Systems: A Programmer's Perspective 3rd**

# 教材与参考书

## 核心教材与原始Slides

## 深入理解计算机系统 第三版

## Computer Systems: A Programmer's Perspective 3rd

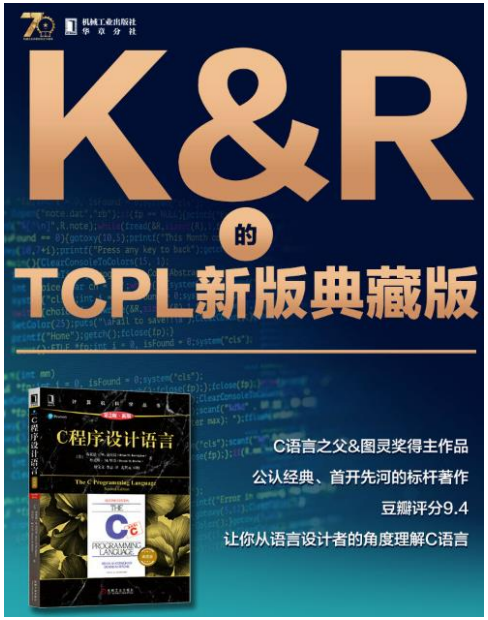



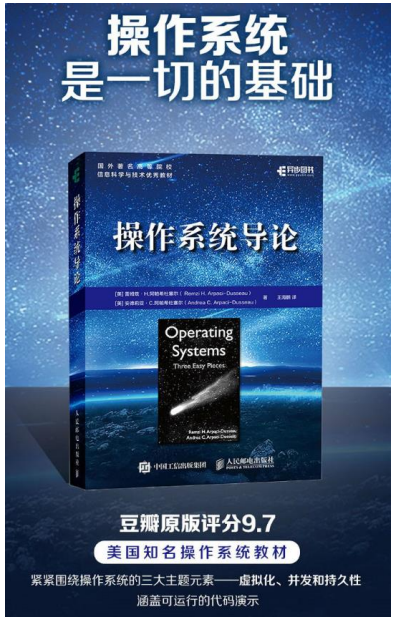
# 教材与参考书

## 参考材料

C语言程序设计 第二版 （K&R）

操作系统导论（OSTEP）



# 教学规划 – 授课

| 内容 | 参考学时 |
| --- | --- |
| 信息的处理与表示 | 4 |
| 程序的机器级表示 | 8 |
| 存储器体系结构 | 4 |
| 程序优化 | 4 |
| 程序链接 | 4 |
| 异常控制流 | 4 |
| 虚拟存储器 | 6 |
| 系统级I/O | 4 |
| 网络编程 | 4 |
| 并发编程 | 4 |

# 教学规划 – 授课

| 内容 | 参考学时 |
| --- | --- |
| 信息的处理与表示 | 4 |
| 程序的机器级表示 | 8 |
| 存储器体系结构 | 4 |
| 程序优化 | 4 |
| 程序链接 | 4 |
| 异常控制流 | 4 |
| 虚拟存储器 | 6 |
| 系统级I/O | 4 |
| 网络编程 | 4 |
| 并发编程 | 4 |

# 教学规划 – 授课

| 内容 | 参考学时 |
| --- | --- |
| 信息的处理与表示 | 4 |
| 程序的机器级表示 | 8 |
| 存储器体系结构 | 4 |
| 程序优化 | 4 |
| 程序链接 | 4 |
| 异常控制流 | 4 |
| 虚拟存储器 | 6 |
| 系统级I/O | 4 |
| 网络编程 | 4 |
| 并发编程 | 4 |

# 教学规划 – 授课

| 内容 | 参考学时 |
| --- | --- |
| 信息的处理与表示 | 4 |
| 程序的机器级表示 | 8 |
| 存储器体系结构 | 4 |
| 程序优化 | 4 |
| 程序链接 | 4 |
| 异常控制流 | 4 |
| 虚拟存储器 | 6 |
| 系统级I/O | 4 |
| 网络编程 | 4 |
| 并发编程 | 4 |

# 教学规划 – 项目实践（Lab）

| 内容 | 参考学时 |
| --- | --- |
| datalab | 1 |
| bomblab | 1 |
| attacklab | 1 |
| cachelab | 1 |
| linkerlab | 8 |
| netlab | 4 |

# 考核方法

# 考核方法

平时成绩 10%

考勤、课堂纪律、上课回答问题、课堂测验

# 考核方法

平时成绩 10%

考勤、课堂纪律、上课回答问题、课堂测验

项目实践 50%

Auto-Grading系统对代码自动打分

Anti-Cheating系统自动检测代码抄袭

抄袭是高压线，一经核实双方项目实践分数为0

# 考核方法

平时成绩 10%

考勤、课堂纪律、上课回答问题、课堂测验

项目实践 50%

Auto-Grading系统对代码自动打分

Anti-Cheating系统自动检测代码抄袭

抄袭是高压线，一经核实双方项目实践分数为0

期末考试 40%

以课堂和Lab内容为主

# The Big Picture

# Course Theme: (Systems) Knowledge is Power!

- **Systems Knowledge**
  - How hardware (processors, memories, disk drives, network infrastructure) plus software (operating systems, compilers, libraries, network protocols) combine to support the execution of application programs
  - How you as a programmer can best use these resources
- **Useful outcomes from taking XJTU-ICS**
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later "systems" classes in CS, ECE, INI, ...
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, Computer Security, etc.

# It's Important to Understand How Things Work

- **Why do I need to know this stuff?**
  - Abstraction is good, but don't forget reality
- **Most CS courses emphasize abstraction**
  - **(CE courses less so)**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
  - Sometimes the abstract interfaces don't provide the level of control or performance you need

# Great Reality #1:

# Ints are not Integers, Floats are not Reals

Source: xkcd.com/571

# Great Reality #1:

# Ints are not Integers, Floats are not Reals

- **Example 1: Is $x^2 \geq 0$?**



Source: xkcd.com/571

# Great Reality #1:

# Ints are not Integers, Floats are not Reals

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

Source: xkcd.com/571

# Great Reality #1:

# Ints are not Integers, Floats are not Reals

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

Source: xkcd.com/571

# 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

Source: xkcd.com/571

# 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 --> ?

Source: xkcd.com/571

# 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)?**



Source: xkcd.com/571

# 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!

Source: xkcd.com/571

# 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:



Source: xkcd.com/571

# 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

Source: xkcd.com/571

# Great Reality #1:

# Ints are not Integers, Floats are not Reals

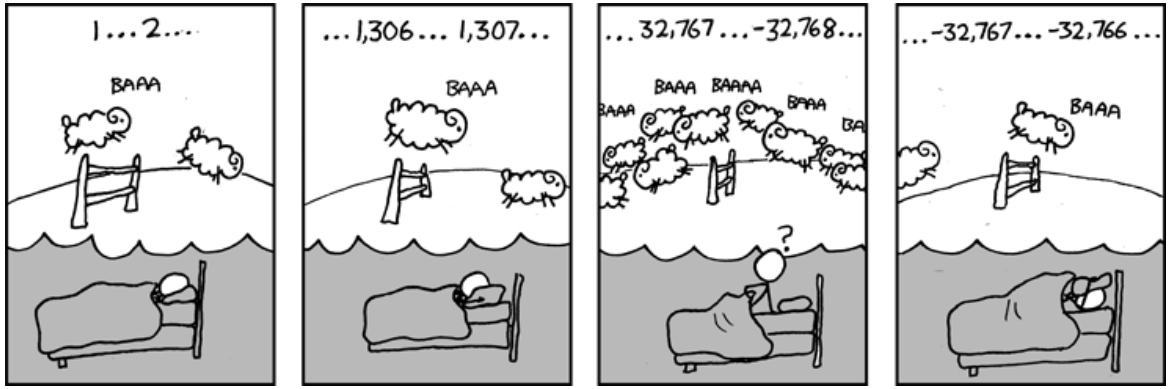
- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!
  - Int's:
    - 40000 * 40000 --> 1600000000
    - 50000 * 50000 --> ?
- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Source: xkcd.com/571

# 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) --> ??

Source: xkcd.com/571

# Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all "usual" mathematical properties**
  - Due to finiteness of representations
  - Integer operations satisfy "ring" properties
    - Commutativity, associativity, distributivity
  - Floating point operations satisfy "ordering" properties
    - Monotonicity, values of signs
- **Observation**
  - Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmers

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

- **Chances are, you'll never write programs in assembly**
  - Compilers are much better & more patient than you are
- **But: Understanding assembly is key to machine-level execution model**
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Creating / fighting malware
    - x86 assembly is the language of choice!

# Great Reality #3: Memory Matters

## Random Access Memory Is an Unphysical Abstraction

- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space
- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

# 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;
}
```

- Result is system specific

# 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;
}
```

- Result is system specific

# 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
```

- Result is system specific

# 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
```

- Result is system specific

# 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
```

- Result is system specific

# 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) -->     3.13999998664856
fun(3) -->     2.00000061035156
```

- Result is system specific

# 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
```

- Result is system specific

# 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
```

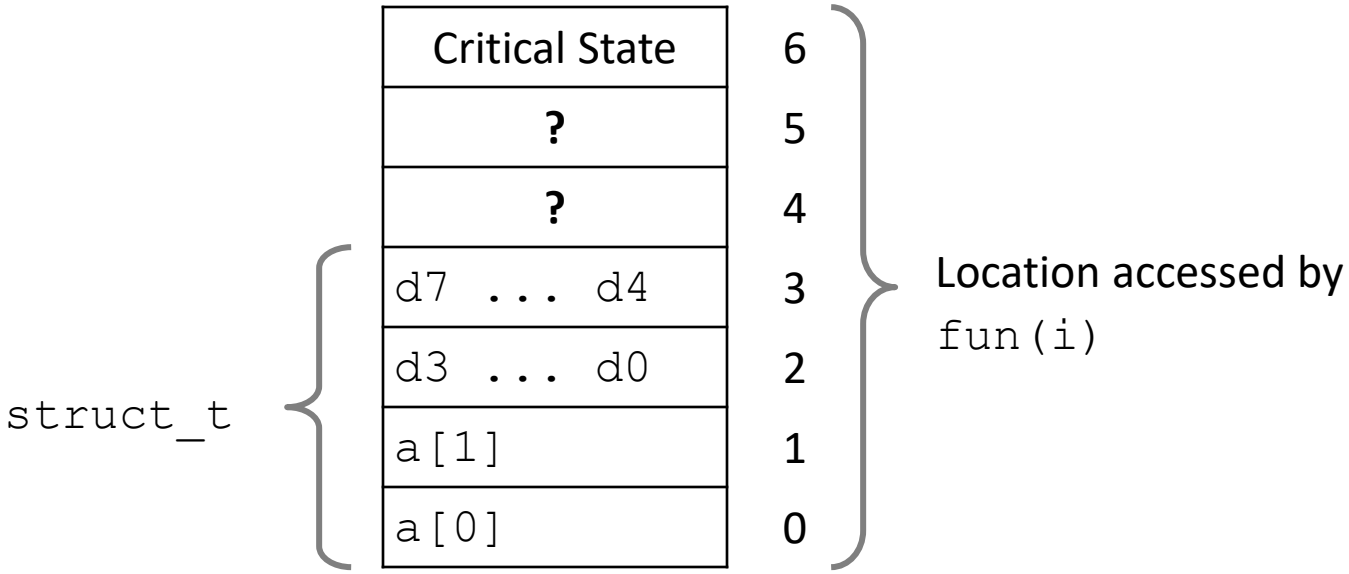
- Result is system specific

# 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 and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
    - Effect of bug may be first observed long after it is generated
- **How can I deal with this?**
  - Program in Java, Ruby, Python, ML, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

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

- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

# 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];
}
```

# 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];
}
```

# 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];
}
```

# 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**

# 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];
}
```

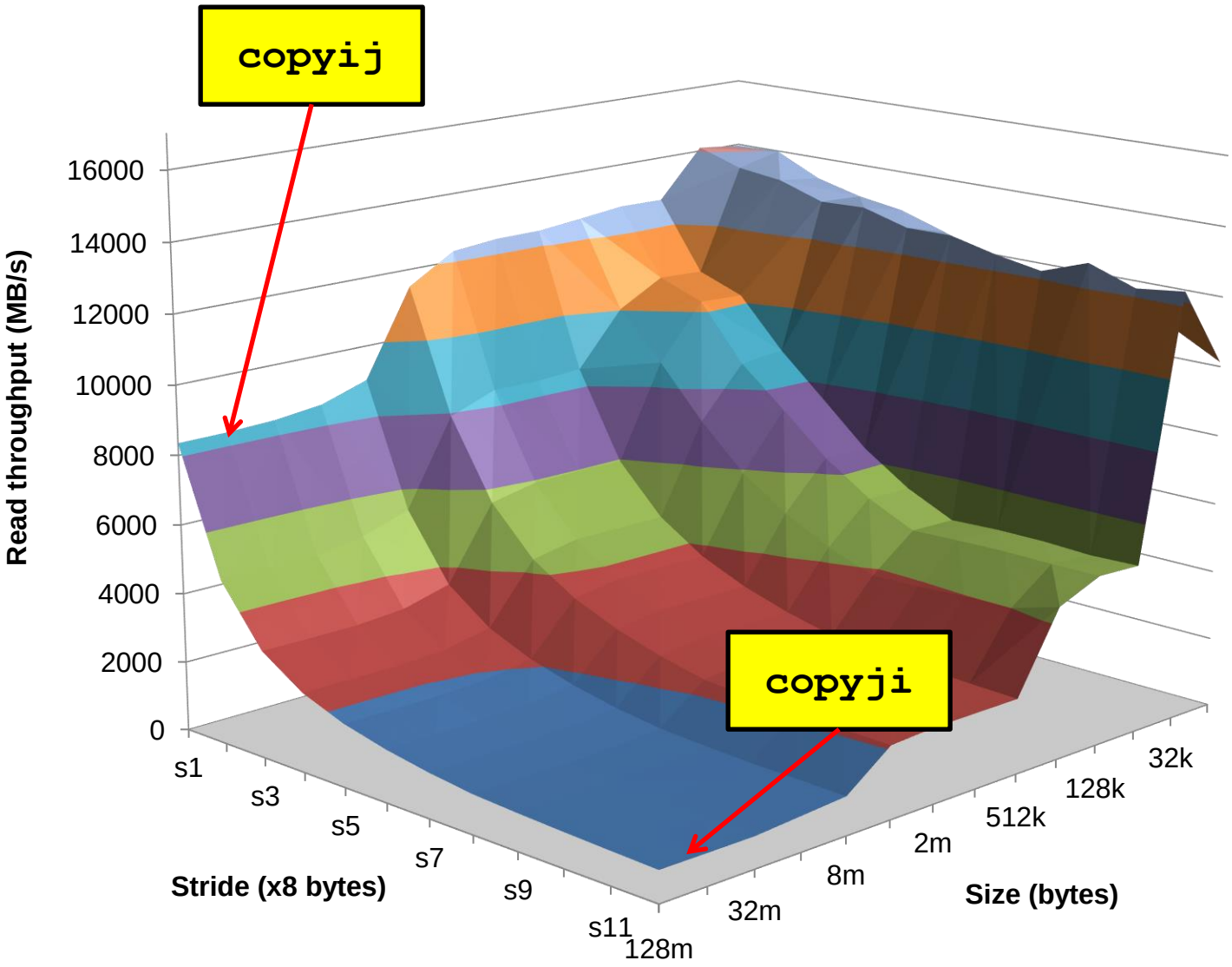
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

**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



# Great Reality #5:
# Computers do more than execute programs

- **They need to get data in and out**
  - I/O system critical to program reliability and performance

- **They communicate with each other over networks**
  - Many system-level issues arise in presence of network
    - Concurrent operations by autonomous processes
    - Coping with unreliable media
    - Cross platform compatibility
    - Complex performance issues

# Course Perspective

- **Most Systems Courses are Builder-Centric**
  - Computer Architecture
    - Design pipelined processor in Verilog
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols

# Course Perspective (Cont.)

- **Our Course is Programmer-Centric**
  - By knowing more about the underlying system, you can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
      - E.g., concurrency, signal handlers
  - Cover material in this course that you won't see elsewhere
  - Not just a course for dedicated hackers
    - **We bring out the hidden hacker in everyone!**

*Welcome*
*and Enjoy!*