Organisation und Architektur von Rechnern

Lecture 01

Instructor:

Reinhard v. Hanxleden

http://www.informatik.uni-kiel.de/rtsys/teaching/v-sysinf2

These slides are used with kind permission from the Carnegie Mellon University

Overview

- Logistics
- Abstraction Is Good But Don’t Forget Reality
- Course Overview

Why are these slides in English?

- … and not – for example – in German (which is the spoken language in class)?
  - Almost all of the publications in this field and most of the manuals and code documentations are in English – including, for example, the primary textbook used in this class
  - So being able to read English makes a vast amount of information accessible that would not be available otherwise.
  - Becoming acquainted with the English terminology is also a prerequisite to writing English documents – which in turn is a prerequisite to make your results globally available.
  - In short, English is the lingua franca of modern computer science – and you should try to practice it whenever you read or produce technical documentation!
- Instructor will try to bridge language gap as much as possible
- However, you must **ask immediately** if slide contents are unclear!

Teaching staff

- Instructor
  - Reinhard von Hanxleden
    (Office: CAP4, R. 1117; 880-7281;
    rvh@informatik.uni-kiel.de)

There are no nominal office hours. Come talk to us anytime! (Or phone or send email)

- Head Teaching Assistants
  - Christoph Daniel Schulze
    (Office: CAP4, R. 1112; 880-7297; cds@...)
  - Miro Spönemann
    (Office: CAP4, R. 1112; 880-7297; msp@...)

- Teaching Assistants
  - Lewe Andersen (lan@...)
  - Julia Krone (jkr@...)
  - Robin Mohr (rmo@...)
  - Nelson Tavares de Sousa (ntd@...)

Registering for this Class

- If you want to take this class, you must register for it electronically (StudiDB, iLearn)
- Access via class homepage
- Before registering, you should
  1. **Team up with a partner**
  2. Decide on which of the recitation classes you want to participate in (see below)

Übungsgruppen

WSP3 - Seminarraum 1, Donnerstag, 10:00 - 12:00
Gruppenleiter Lewe Andersen

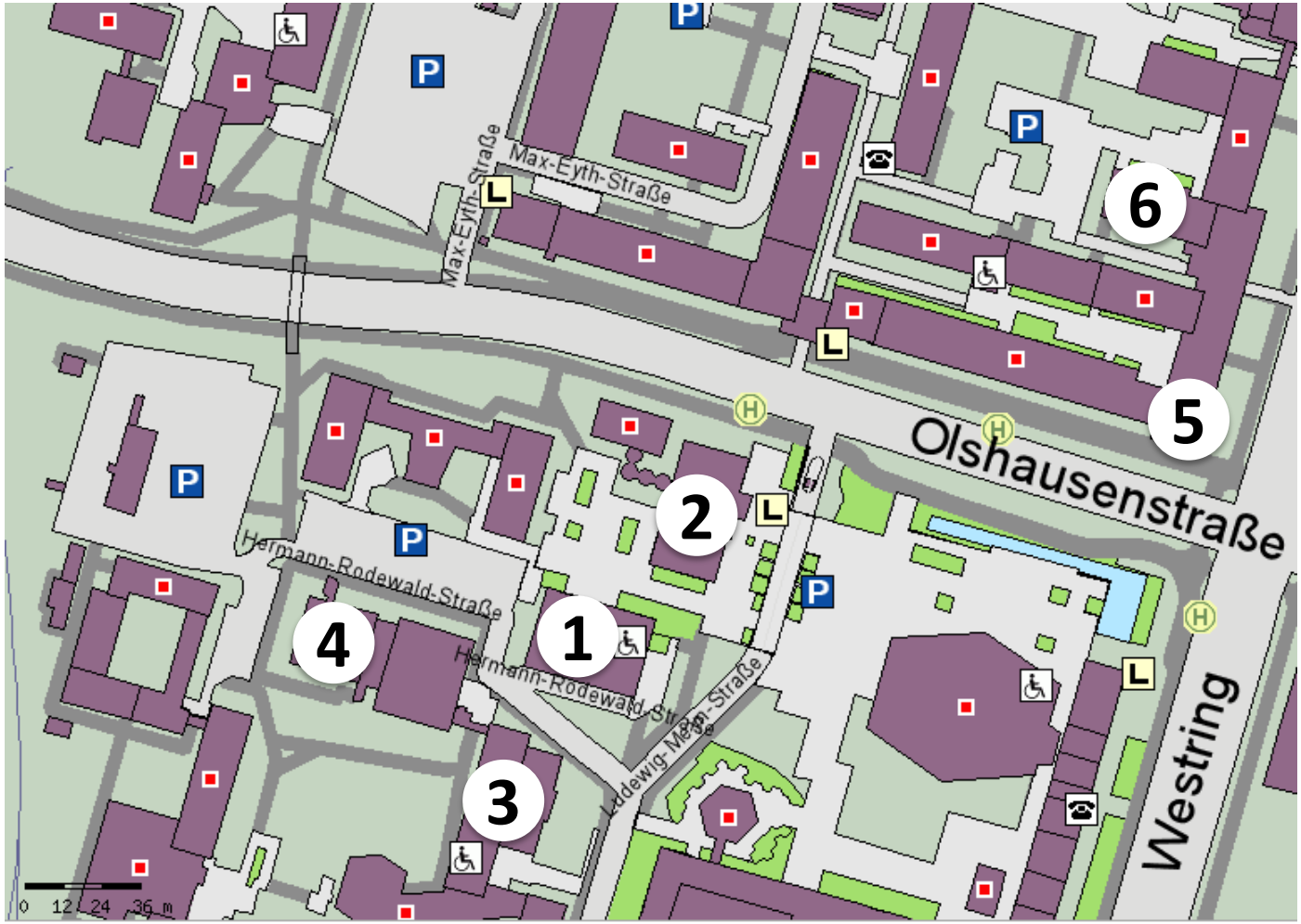
OS40 - R.13, Donnerstag, 10:00 - 12:00
Gruppenleiter Miro Spönemann

HRS3 - R.218b [Schulungsraum], Donnerstag, 14:00 - 16:00
Gruppenleiter Christoph Daniel Schulze

LMS2 - R.Ü1, Freitag, 12:00 - 14:00
Gruppenleiter Robin Mohr

CAP4 - R.1304 a, Freitag, 12:00 - 14:00
Gruppenleiter Christoph Daniel Schulze

Locations



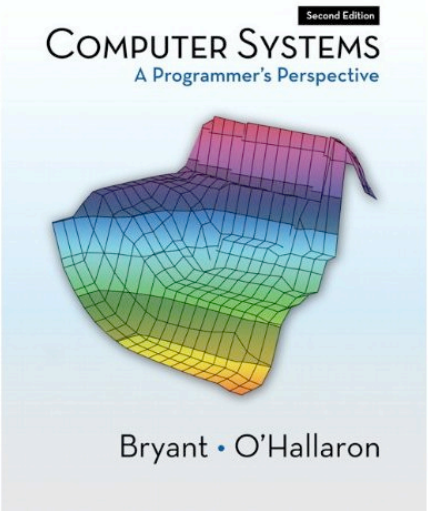
Class
1 CAP3 R.2

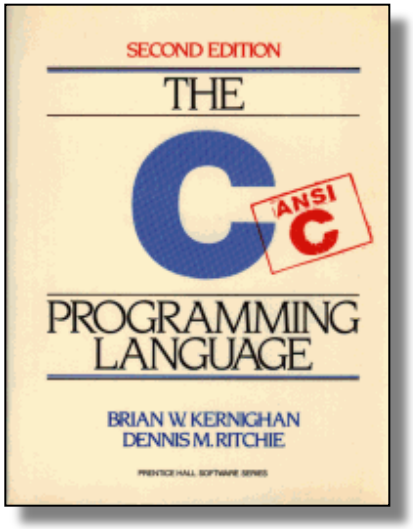
Prof + Head TAs
2 CAP4, R.1112

Recitations
2 CAP4, R.1304a
3 LMS2, Ü1
4 HRS3, R.218b
5 OS40, R.13
6 WSP3, R.1

Textbooks

- **Randal E. Bryant and David R. O'Hallaron,**
  - "Computer Systems: A Programmer's Perspective", Prentice Hall 2nd Edition, 2010. http://csapp.cs.cmu.edu
  - **Chapters 1-6 will be "Skript" for this course**
  - Based on introductory course taught at Carnegie-Mellon University
  - Copies available at library
- **Brian Kernighan and Dennis Ritchie,**
  - "The C Programming Language, Second Edition", Prentice Hall, 1988
  - A classic





Textbooks 2

- **“Die Programmiersprache C. Ein Nachschlagewerk”**
  - Script of RRZN (Regionales Rechenzentrum für Niedersachsen)
  - 16th Edition, 2007
  - Introduction to and reference of C for beginners, 160 pages
  - German language
  - Copies can be obtained by the “Kopierstelle” at the main university library
    - 3,60 Eur soft cover

Prerequisites

- **Basic programming skills in C**
  - Should know how to write, compile, and run simple programs
  - See also class home page
  - Use the problem assignments series 1 to learn or check your programming skills
- **Access to Linux System**
  - Some of the labs require Linux
  - Preferably, you – or your group partner – should have your own PC, with a Linux system
  - If that is not possible, you can work on the ThinLinc systems (not tested yet)
  - There is also a VM available with a Linux installation that will work wonderfully for this lecture (see the lecture's iLearn page)
  - See also class home page

Course Components

- **Lectures**
  - Higher level concepts
- **Problem Assignments**
  - Relatively small practical and theoretical homework problems
  - Typically one week to solve
- **Lab Assignments**
  - Practical problems – the heart of the course
  - Typically two weeks to solve – may overlap with assignments
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- **Recitation classes (*"Übungsstunden"*)**
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam coverage
  - Discussion of Assignments and Labs
  - **Recitation classes start next week**

Lab Rationale

- **Each lab should have a well-defined goal such as solving a puzzle or winning a contest.**
  - Use minimal number of operators for computing expression
  - Defusing a binary bomb.
- **Doing a lab should result in new skills and concepts**
  - Data Lab: computer arithmetic, digital logic.
  - Bomb Labs: assembly language, using a debugger, understanding the stack
  - Architecture Lab: processor design
- **We try to use competition in a fun and healthy way.**
  - Set a threshold for full credit.
  - Post intermediate results (anonymized) on Web page for glory!

Getting Help

- **Web**
  - http://www.informatik.uni-kiel.de/rtsys/teaching
  - Register for the class via StudiDB
  - Register for the recitations (Übungsgruppen) via iLearn
  - Copies of lectures + assignments
  - Clarifications to assignments
- **Personal help**
  - Professor, Head TA: door open means come on in (no appt necessary)
  - TAs: please mail first.

e-Mail

- **Current information about the class will be disseminated**
  - Within the class and the recitation classes
    - If you can't make it to class, you should make sure that at least your team partner participates in class
  - Via e-Mail
    - READ YOUR E-MAIL!
    - Preferred e-Mail address is your account at the department (@informatik.uni-kiel.de) or at the university (@email.uni-kiel.de)
    - If you rather read mails from another account (e.g., from your personal provider), should *forward* your university e-Mail to that account
    - It's easy to forward mail, see http://www.informatik.uni-kiel.de/rbg/email/email-weiterleitung/
    - Send mail from your informatik-address only. How to do that from home, see http://trac.rtsys.informatik.uni-kiel.de/trac/rtsys/wiki/Sending_Mail_Externally
    - All of the above apply not only to this class, but throughout your studies!
    - Individual matters are sent to you personally
    - Other matters are sent to mailing list of the whole class
    - READ YOUR E-MAIL!

Policies: Assignments

- **Team work**
  - You should work in groups of two at all labs and assignments
  - Splitting work within group is ok – but *both* group members must fully understand *complete* solution
  - Choosing the right team partner is critical
  - If a team does not work out, inform us as soon as possible, and we'll try to work out a solution
- **Handins**
  - Assignments due at 11:59pm on Mondays.
  - Electronic hand-ins only: iLearn System
  - Can write formulas ($x^2 = \ldots$) in Pseudo-TeX (*x^2 = …*)
  - Discussion in recitation class same week Thursday/Friday

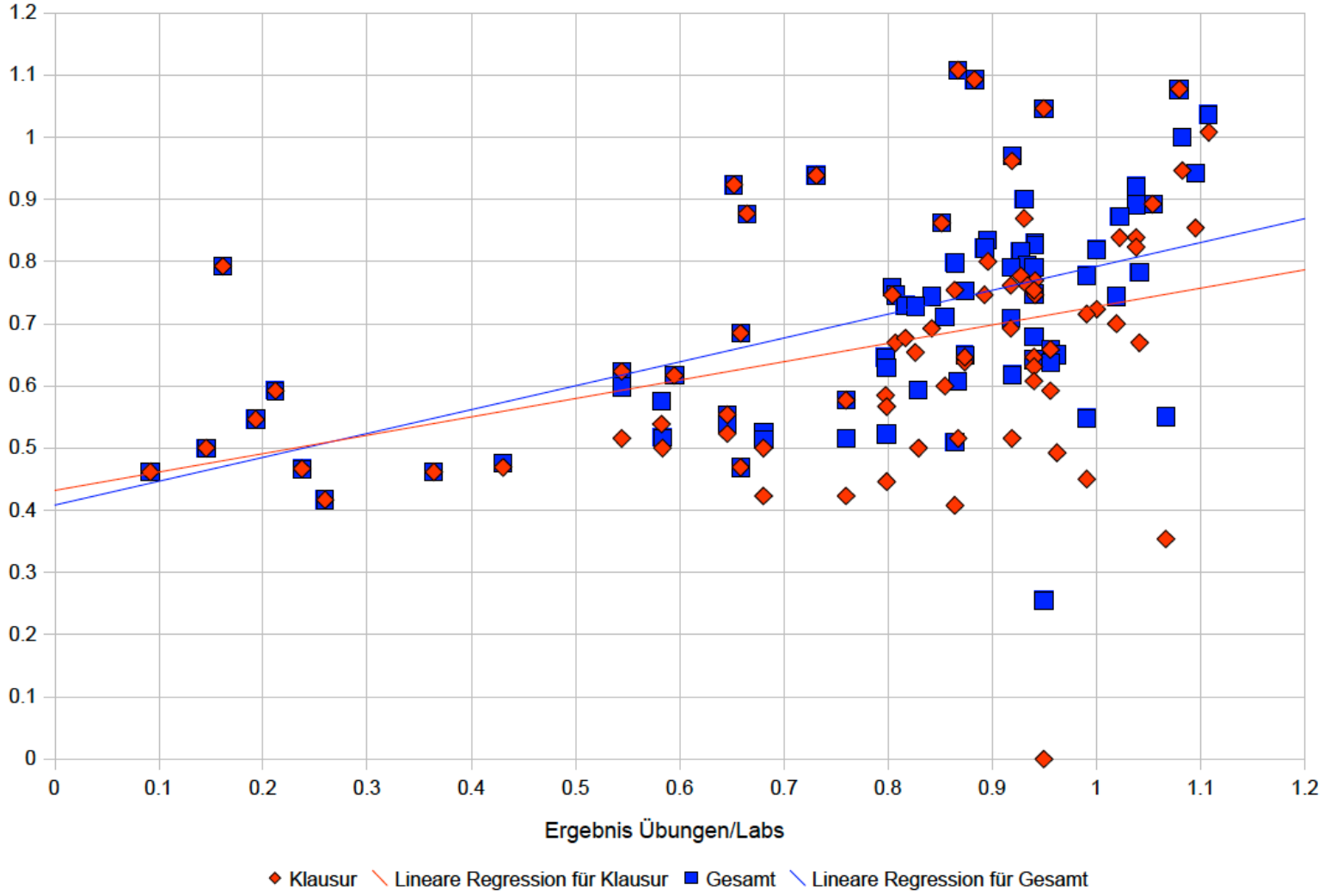
Policies: Assignments, Grading

- **Assignments**
  - Can receive bonus points for outstanding solutions
  - Can reject or deduct points for late submissions – or anything else that makes the graders' life more miserable than necessary … this includes sabotaging things in any way, eg, by submitting solutions to the Data Lab that use illegal operators.
- **Your final grade depends on the final written exam**
  - Need at least 50% to pass
- **Allowed to take final exam if:**
  - Received at least 50% of available homework assignment points
  - Missed at most two recitation classes

Cheating

- **What is cheating?**
  - Sharing code: either by copying, retyping, looking at, or supplying a copy of a file
- **What is NOT cheating?**
  - Helping others use systems or tools
  - Helping others with high-level design issues
  - Helping others debug their code
- **Penalty for cheating:**
  - Removal from course with failing grade

It pays to participate …



Previous results

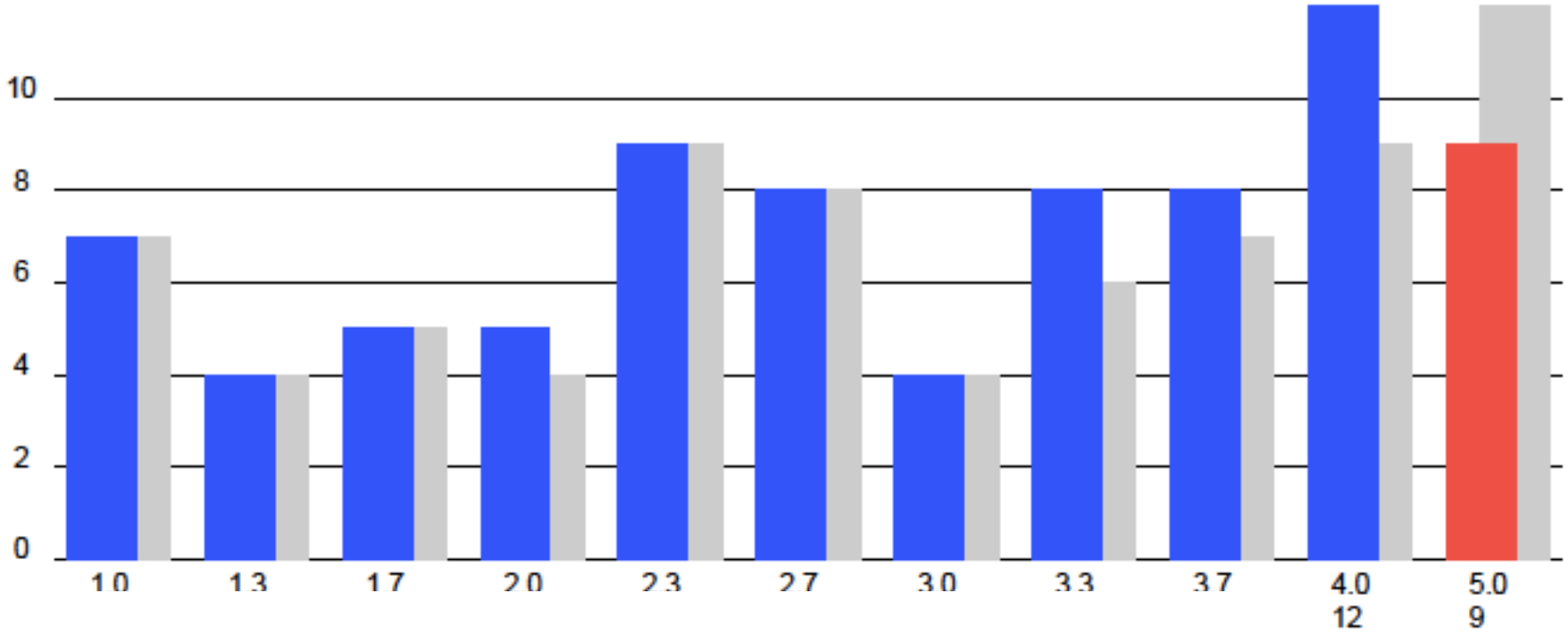
Conduct in Class

- **I:**
  - Start and finish on time
  - Try to deliver high-quality lectures
  - Listen to your concerns
  - Make sure you have a fair chance of passing this class

- **You:**
  - Are punctual (should you arrive late, please use rear entrance)
  - Do not disturb others (**no talking with neighbor, no laptops**)
  - Ask if things are unclear (during lecture, or afterwards)
  - Actively participate

Shared goal: Make you pass this class

- This class requires work!
- But there are rewards: around 80% - 90% of participants will pass



Previous results

Overview

- Logistics
- Abstraction Is Good But Don't Forget Reality
- Course Overview

Course Theme:
Abstraction Is Good But Don't Forget Reality

- **Most CS courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
- **Useful outcomes**
  - 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
    - Compilers, Operating Systems, Networks, Computer Architecture, Real-Time/Embedded Systems

Four realities of computer systems

- **Abstraction is good, but don't forget reality!**
- **Courses to date emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand underlying implementations
- **Useful outcomes**
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to tune program performance
  - Prepare for later "systems" classes in CS
    - Operating Systems, Communication Systems, Computer Architecture, Compilers, Real-Time Systems

Great Reality #1:
Int's are not Integers, Float's 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) --> ??

Code Security Example

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

- Similar to code found in FreeBSD's implementation of getpeername
- There are legions of smart people trying to find vulnerabilities in programs

Typical Usage

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, MSIZE);
    printf("%s\n", mybuf);
}
```

Malicious Usage

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, -MSIZE);
    . . .
}
```

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 program in assembly**
  - Compilers are much better & more patient than you are
- **But: Understanding assembly key to machine-level execution model**
  - Behavior of programs in presence of bugs
    - High-level language model breaks 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!

Assembly Code Example

- **Time Stamp Counter**
  - Special 64-bit register in Intel-compatible machines
  - Incremented every clock cycle
  - Read with rdtsc instruction
- **Application**
  - Measure time (in clock cycles) required by procedure

```
double t;
start_counter();
P();
t = get_counter();
printf("P required %f clock cycles\n", t);
```

Code to Read Counter

- Write small amount of assembly code using GCC's asm facility
- Inserts assembly code into machine code generated by compiler

```
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;

/* Set *hi and *lo to the high and low order bits
   of the cycle counter.
*/
void access_counter(unsigned *hi, unsigned *lo)
{
    asm("rdtsc; movl %%edx,%0; movl %%eax,%1"
       : "=r" (*hi), "=r" (*lo)
       :
       : "%edx", "%eax");
}
```

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

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

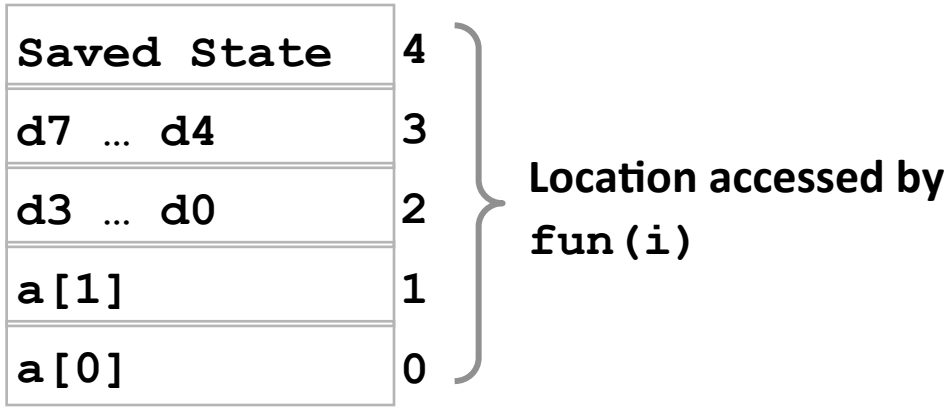
```
fun(0)  ->    3.14
fun(1)  ->    3.14
fun(2)  ->    3.1399998664856
fun(3)  ->    2.00000061035156
fun(4)  ->    3.14, then segmentation fault
```

Memory Referencing Bug Example

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

```
fun(0)  ->     3.14
fun(1)  ->     3.14
fun(2)  ->     3.1399998664856
fun(3)  ->     2.00000061035156
fun(4)  ->     3.14, then 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 or ML
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors

Memory System Performance Example

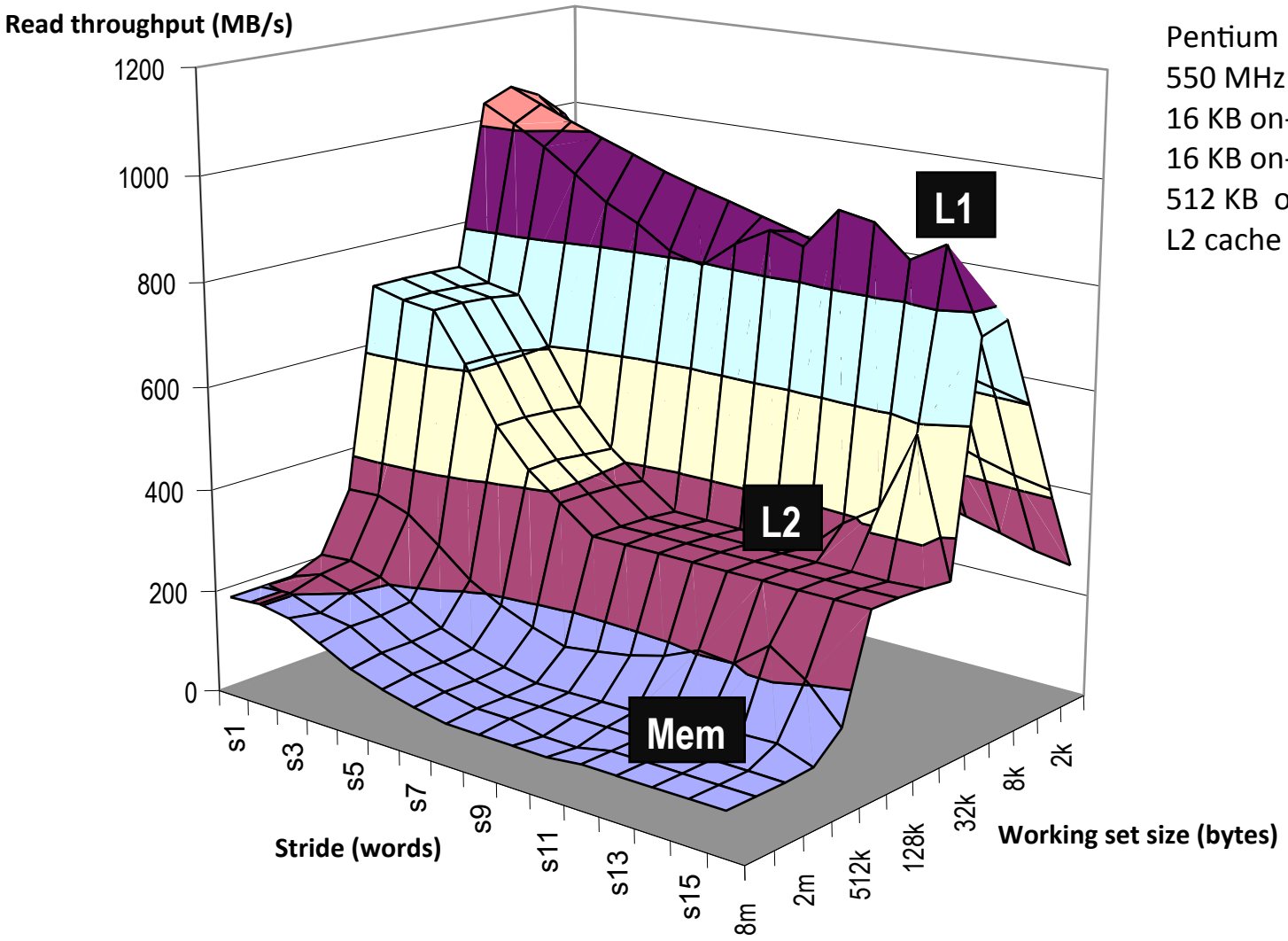
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
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];
}
```

21 times slower (Pentium 4)

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

The Memory Mountain

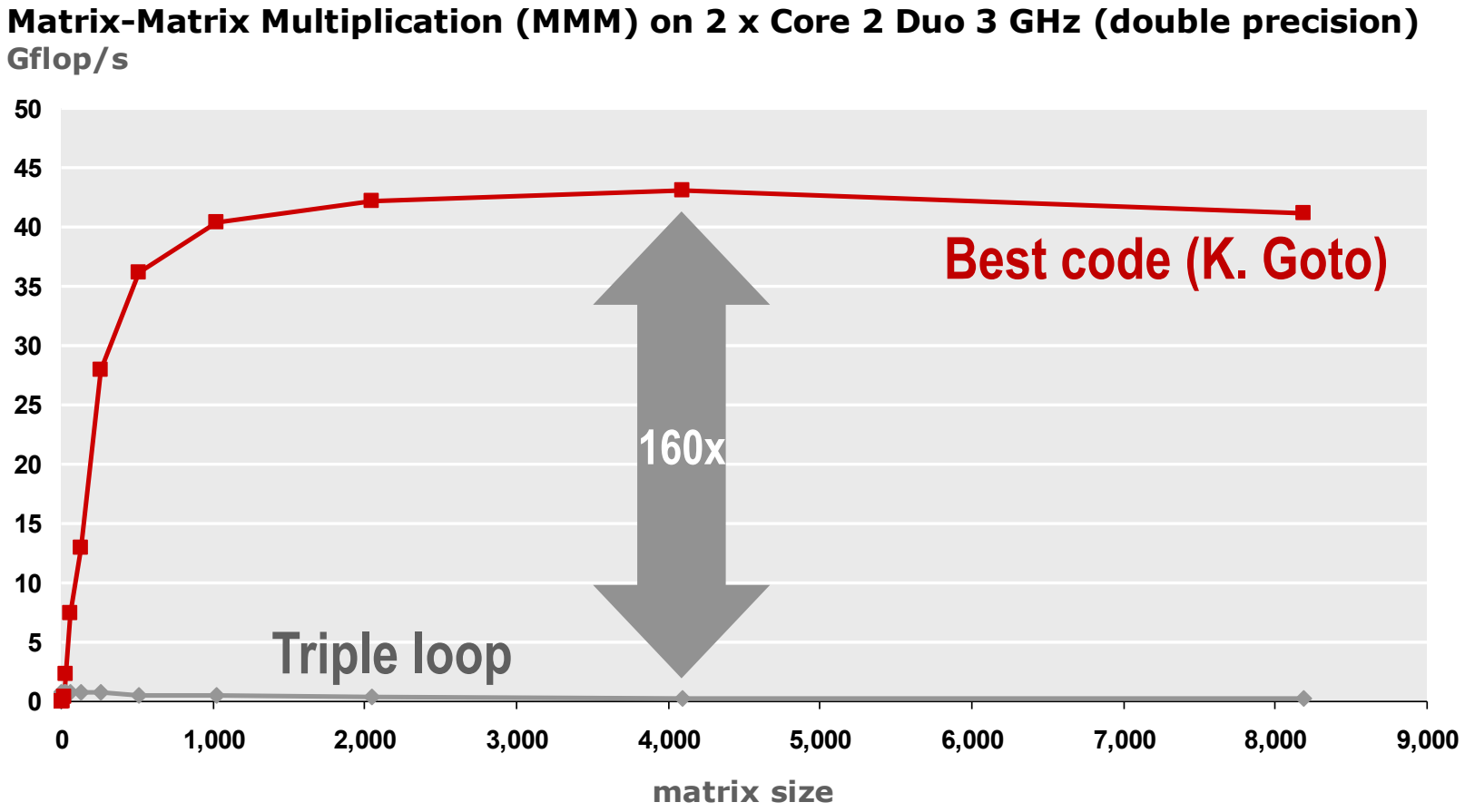


Pentium III Xeon
550 MHz
16 KB on-chip L1 d-cache
16 KB on-chip L1 i-cache
512 KB off-chip unified
L2 cache

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

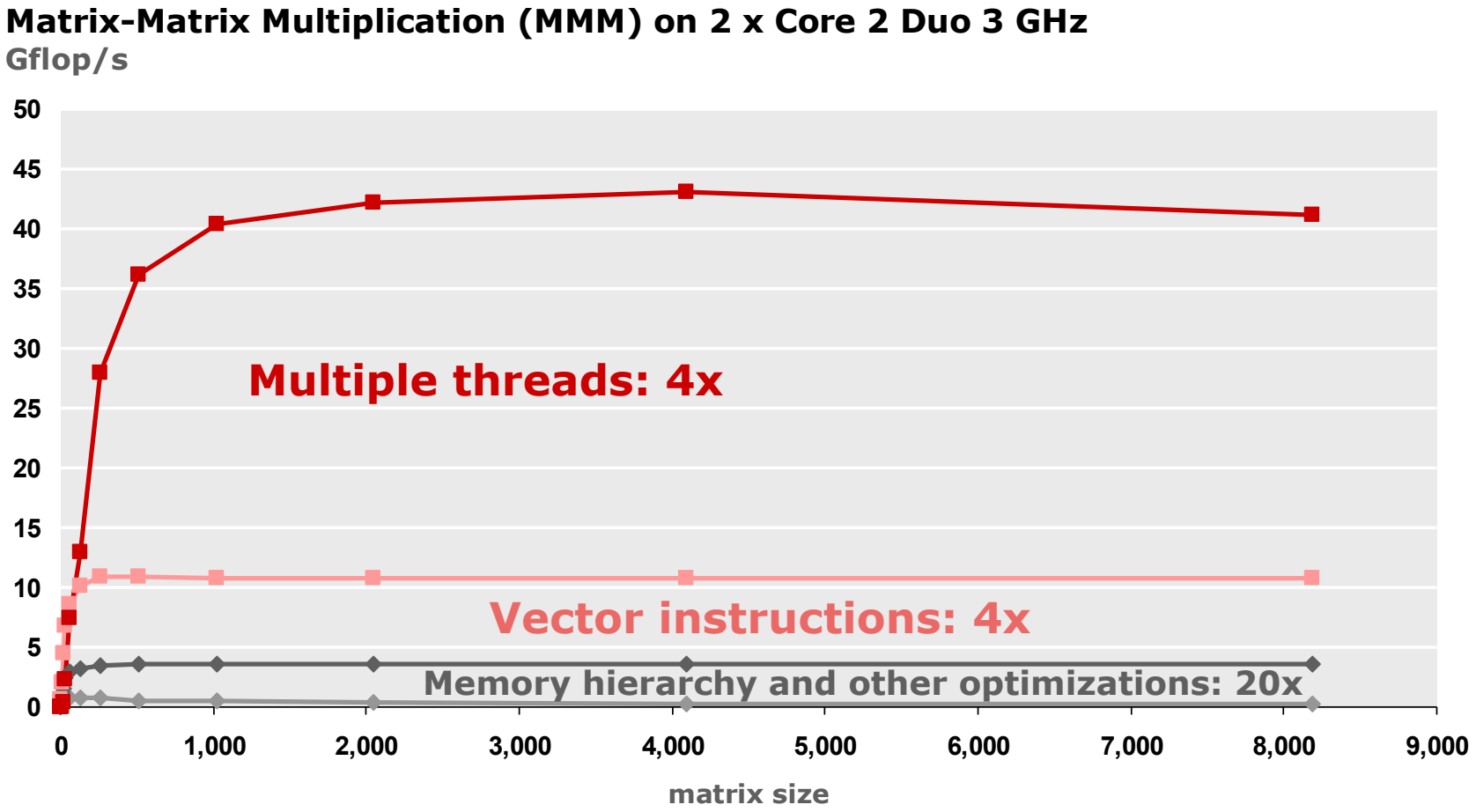
- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**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

Example Matrix Multiplication



- Standard desktop computer, vendor compiler, using optimization flags
- Both implementations have **exactly** the same operations count ($2n^3$)
- *What is going on?*

MMM Plot: Analysis



- Reason for 20x: Blocking or tiling, loop unrolling, array scalarization, instruction scheduling, search to find best choice
- *Effect: less register spills, less L1/L2 cache misses, less TLB misses*

Overview

- Logistics
- Abstraction Is Good But Don’t Forget Reality
- Course Overview

Course Perspective

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

Course Perspective (Cont.)

- **Our Course is Programmer-Centric**
  - Purpose is to show how by knowing more about the underlying system, one 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
  - Not just a course for dedicated hackers
    - We bring out the hidden hacker in everyone
  - Cover material in this course that you won't see elsewhere

Programs and Data

- **Topics**
  - Bits operations, arithmetic, assembly language programs, representation of C control and data structures
  - Includes aspects of architecture and compilers

- **Assignments**
  - L1 (datalab): Manipulating bits
  - L2 (bomblab): Defusing a binary bomb

Architecture

- **Topics**
  - Instruction set architecture
  - Logic design
  - Sequential implementation
  - Pipelining and initial pipelined implementation
  - Modern processor design

- **Assignments**
  - L3: Architecture Lab

The Memory Hierarchy

- **Topics**
  - Memory technology, memory hierarchy, caches, disks, locality
  - Includes aspects of architecture and OS.

Performance

- **Topics**
  - Optimization (control and data), measuring time on a computer
  - Includes aspects of architecture, compilers, and OS

- **Assignments:**
  - L4 (Perflab): Optimize the runtime of a routine

Lab Rationale

- **Each lab should have a well-defined goal such as solving a puzzle or winning a contest.**
- **Doing a lab should result in new skills and concepts**
- **We try to use competition in a fun and healthy way.**
  - Set a reasonable threshold for full credit.
  - Post intermediate results (anonymized) on Web page for glory!

Have Fun!