

CS4400
“How to be a serious programmer”

Computer Systems

John Regehr
Spring Semester 2004

Course Theme

- **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
- **Abstractions hide things from you**
 - Good abstractions hide things you don't care about
 - Bad abstractions hide things you do care about
 - What you care about changes over time!

Useful Outcomes

- **Become more effective programmers**
 - Able to find and eliminate bugs efficiently
 - Able to tune program performance
- **Become more resourceful**
 - Figure things out instead of asking people
 - Intuitive understanding of the machine at all levels of abstraction
- **Foundation for specialized “systems” classes in CS**
 - Compilers, Operating Systems, Networks, Computer Architecture

Why should you care?

- **This material is what makes the difference between average coder and serious programmer**
 - **Dirty secret: We sometimes let really bad programmers graduate**
 - **Don't be one of them – life's too short to be a computer scientist who sucks at programming**
- **Applicable to all areas of programming, not just systems**
- **CS traditionally has a sink-or-swim culture**
 - **“If it's important you'll figure it out yourself”**
 - **This material not taught until a few years ago**
 - **This class is intended as an antidote to that mentality**
 - » **To give you a head start on being a serious programmer**

Great Reality #1

- *Int's are not Integers, Float's are not Reals*

- **Examples**

- **Is $x^2 \geq 0$?**

- » **Math class: Yes!**

- » **Floats: Yes!**

- » **Int's:**

- **40000 * 40000 --> 1600000000**

- **50000 * 50000 --> ??**

- **Is $(x + y) + z = x + (y + z)$?**

- » **Math class: Yes!**

- » **Unsigned & Signed Int's: Yes!**

- » **Float's:**

- **$(1e20 + -1e20) + 3.14$ --> 3.14**

- **$1e20 + (-1e20 + 3.14)$ --> ??**

Computer Arithmetic

- **Cannot assume “usual” 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*
- **You'll probably never write much of it**
 - **Compilers are much better & more patient than you are**
- **Understanding assembly key to machine-level execution model**
 - **Behavior of programs in presence of bugs**
 - » **High-level language model breaks down**
 - » **Many bugs can only be understood at the assembly level**
 - **Tuning program performance**
 - » **Understanding sources of program inefficiency**
 - » **Getting accurate timing information**
 - **Implementing system software**
 - » **Compiler has machine code as target**
 - » **Operating systems must manage process state – important things cannot be “said” in C, Java, Perl, etc.**

Assembly Code Example

- **Time Stamp Counter**

- Special 64-bit register in x86 machines where $x \geq 5$ (Pentium and Athlon)
- Incremented every clock cycle after processor powers up
 - » How long is a cycle on a 3 GHz machine?
- Read with `rdtsc` instruction

- **Application**

- Measure time required by procedure
 - » In units of clock cycles

```
long long s = read_counter();  
P();  
long long t = read_counter() - s;  
printf("P required %lld 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
- What happens if you compile this using Visual C++?
- What happens if you compile this on a Sparc?

```
long long read_counter (void)
{
    long long d;
    __asm__ __volatile__ ("rdtsc" : "=A" (d));
    return d;
}
```

Measuring Time

- **Trickier than it Might Look**
 - Many sources of variation
- **Example**
 - Sum integers from 1 to n

n	Cycles	Cycles/n
100	961	9.61
1,000	8,407	8.41
1,000	8,426	8.43
10,000	82,861	8.29
10,000	82,876	8.29
1,000,000	8,419,907	8.42
1,000,000	8,425,181	8.43
1,000,000,000	8,371,2305,591	8.37

Great Reality #3

- *Memory Matters*
- **Memory is finite**
 - It must be allocated and managed
 - Many applications are memory dominated
- **The hardest bugs are memory related**
 - Effects are distant in both time and space
- **Not all memory is created equal**
 - 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

```
main ()
{
    long int a[2];
    double d = 3.14;
    a[2] = 1073741824; /* Out of bounds reference */
    printf("d = %.15g\n", d);
    exit(0);
}
```

	Alpha	MIPS	Linux
-g	5.30498947741318e-315	3.1399998664856	3.14
-O	3.14	3.14	3.14

(Linux version gives correct result, but implementing as separate function gives segmentation fault.)

Memory Referencing Errors

- **C and C++ provide zero 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, Ada, Python, Scheme, or ML
 - » problem: systems interface and systems programming
 - Program carefully and defensively
 - Use or develop tools to detect referencing errors

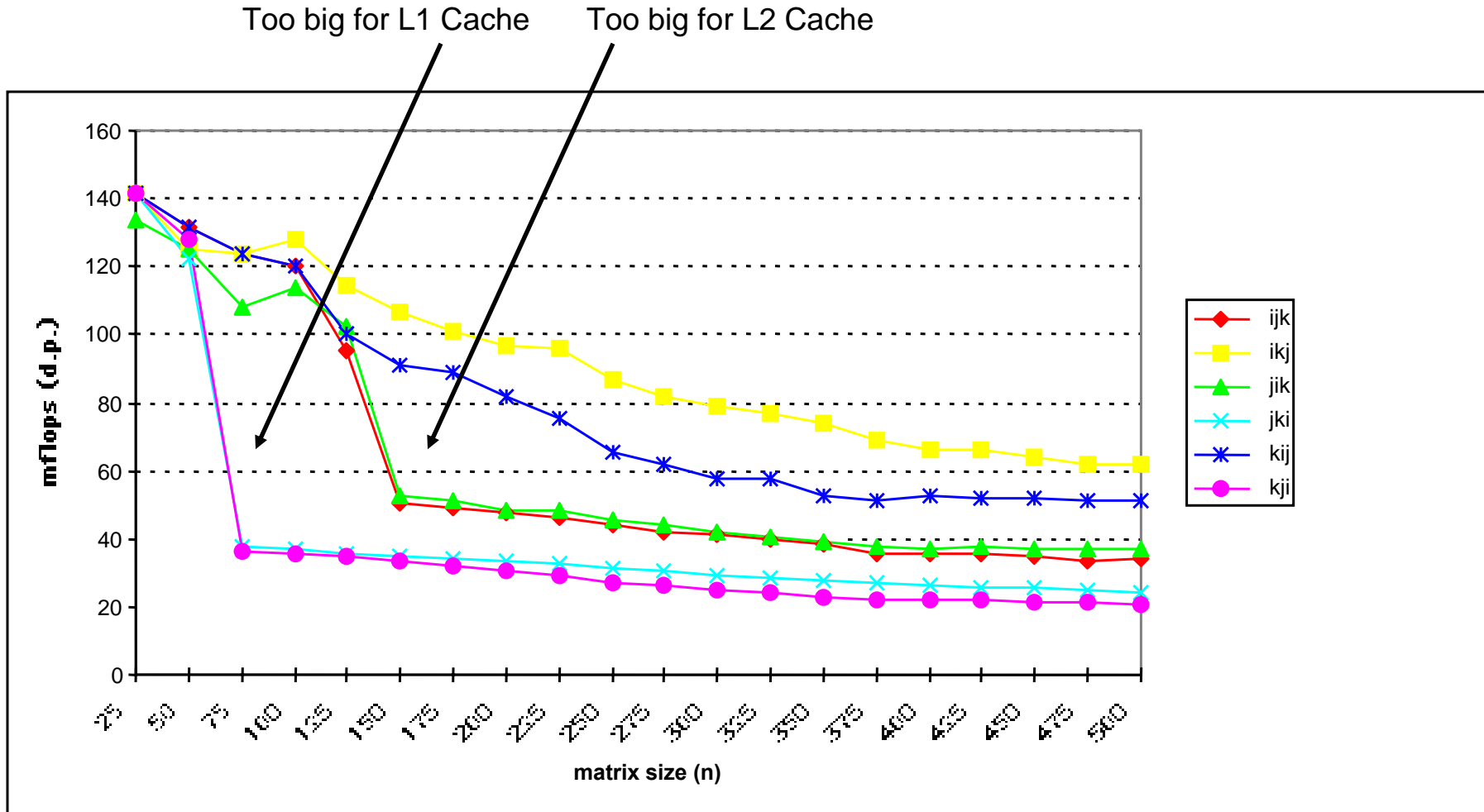
Memory Performance Example

- Implementations of Matrix Multiplication
 - Multiple ways to nest loops

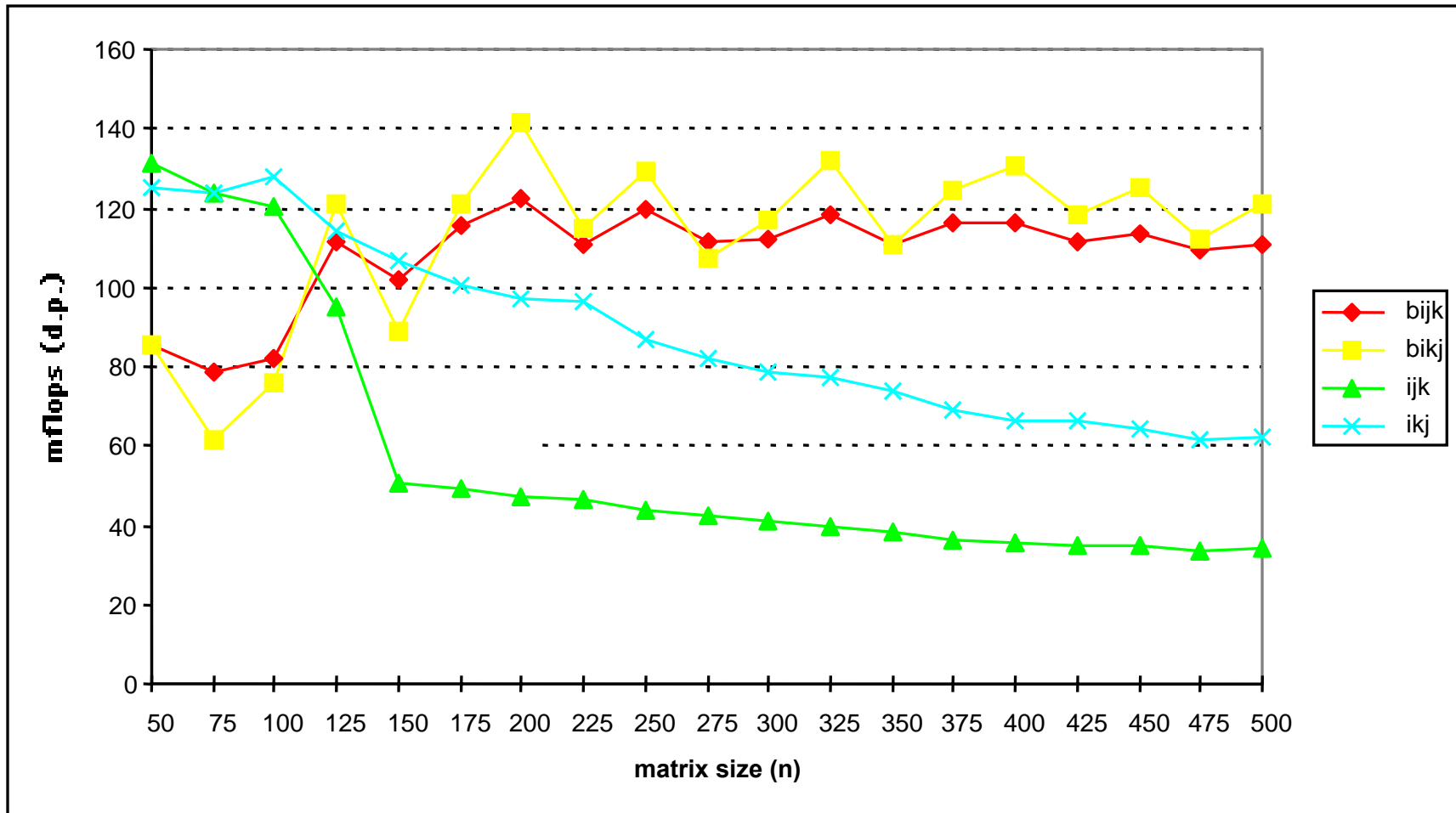
```
/* ijk */
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

```
/* jik */
for (j=0; j<n; j++) {
  for (i=0; i<n; i++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

Matmult Performance (Alpha 21164)



Blocked matmult perf (Alpha 21164)



Great Reality #4

- *There's more to performance than asymptotic complexity*
- **Constant factors matter too!**
 - Compiling with “-O3” gives ~2x performance increase
 - Can easily see 10:1 or 100:1 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

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 and analysis of processor
 - **Operating Systems**
 - » Implement large portions of operating system
 - **Compilers**
 - » Write compiler for simple language
 - **Networking**
 - » Implement and simulate network protocols
 - These courses are sort of silly because most people are not going to design a CPU, an OS, a compiler, etc.

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**
 - » **Goal is to start on the path to an elite “power programmer”**
 - a rare and highly sought talent
 - **Cover material in this course that you won’t see elsewhere**
 - » **performance analysis**
 - » **holistic treatment of the computer as a system**
 - and how you can write better programs by knowing about the details

Computer System

- **Tight ensemble of components**
 - **hardware**
 - » processor, caches, memory, I/O bridges & devices, network
 - **OS**
 - » resource management for files, processes, threads, and memory
 - » process protection
 - illusion of a single running process with access to everything
 - **Application program**
 - » your code
 - **Compiler (preprocessor, compiler, assembler, linker)**
 - » `cpp: foo.c → foo.i` (bring in all the include stuff)
 - » `cc1: foo.i → foo.s` (creates machine language text program)
 - » `as: foo.s → foo.o` (relocatable object/binary file)
 - » `ld: foo.o → foo` (creates an executable binary file)
 - **Example**
 - » read Chapter 1 – it's an excellent step by step system view

Course Logistics

- **Best source of information is the course web page**
 - » www.cs.utah.edu/classes/cs4400
 - » contains these slides for example
- **TAs**
 - » Swaminathan Pichumani (Swami)
 - » Sonjong Hwang
 - » plus hopefully 1 more given the class size!
 - » office hours and locations will be posted on the web soon
- **Office Hours**
 - » use them
 - » John's office hours are right after class until the line is empty

Textbooks

- **Randal E. Bryant and David R. O'Hallaron,**
 - “Computer Systems: A Programmer’s Perspective,” Prentice Hall 2003
 - csapp.cs.cmu.edu
 - Required

- **Brian Kernighan and Dennis Ritchie,**
 - “The C Programming Language, Second Edition,” Prentice Hall, 1988
 - Optional, but most serious programmers own this

Course Components

- **Lectures – Higher level concepts**
- **Text Problems**
 - **two types of problems in the book**
 - » **practice – you should do all of these as you read**
 - answers are at the end of each chapter
 - » **homework – these will appear on the web syllabus**
 - they are optional but you should do them as they are the best preparation for the exams
- **Office Hours**
 - **Applied concepts, important tools and skills for labs, clarification of lectures, help with homework problems**
- **Labs – the heart of the course**
 - **1 or 2 weeks each**
 - **Provide in-depth understanding of some system aspect**
 - **Programming and measurement**
 - » **ANSI C, Unix, and x86**
 - » **nothing else will do**

FAQ: How do I do my labs at home?

- **Chances are**
 - you have a Windows machine at home
 - this is unlikely to work
- **Options**
 - **get a terminal emulator and ssh to labnixY**
 - » Putty is free and works (do a Google search)
 - » SecureCRT is better but costs \$\$
 - **run Unix emulation on top of Windows**
 - » VMware is excellent but pricey
 - » Cygwin is free but has some rough edges
 - **turn your machine into a Linux box**
 - » Not to be taken lightly!
 - » If you already run your own Linux box parts of this course will be easy or boring for you

Labs

- **Before you turn your lab in**
 - **make sure it runs on the Linux boxes in EMCB 210**
 - » **which one doesn't matter**
 - **why?**
 - » **because we run the grading program on these machines**
 - » **if your code bombs then you lose**

Getting Help

- **Web**
 - www.cs.utah.edu/classes/cs4400
 - » read the syllabus
 - Copies of lectures, assignments, exams, solutions
 - Clarifications to assignments
- **Email**
 - cs4400@cs.utah.edu
 - » you must be on this list, use mailman to sign up
 - » contains clarifications to assignments, general discussion
 - teach-cs4400@cs.utah.edu
 - » goes to John + TAs
 - » questions on labs, requests for an appointment, etc.
- **Personal help**
 - Office hours – if you can't make it, email us for an appointment

Policies: Assignments

- **Work groups**
 - You must work alone on all labs (possibly except for the last lab)
- **Handins**
 - Assignments due by midnight on specified due date
 - Electronic handins only.
 - » details will be in the lab assignments
- **Makeup exams and assignments**
 - Only by prior arrangement and it will need to be a documented emergency
 - Too busy doesn't cut it
 - Sounds harsh but with 110 students the other way doesn't work
- **Appealing grades**
 - Within 7 days of grades being posted on the Web
 - Labs: Talk to the lead TA on the assignment
 - Exams: Talk to John

Policies: 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.**
 - » you have a consent form that you'll need to sign on this issue
 - **Appeals to the SoC Director**
 - » there will be no warning and no appeal at the course level
 - **Note**
 - » we will be using automatic cheating detectors that are very good
 - » you'll be asked to explain suspicious results

Policies: Lecture Etiquette

- Don't talk while I'm talking
- Turn off phones
- If you arrive late, come in quietly and sit in the back
- If you're doing something besides listening to the lecture, do it quietly and unobtrusively
- Basically:
 - Whether you pay attention or not is your own business
 - But you must not be disruptive – it's not fair
- Do ask questions: If you're confused chances are other people are too
 - Even big classes like this one can be interactive

Policies: Grading

- **Exams (50%)**
 - Two in class exams (15% each)
 - Final (20%)
 - All exams are open book / closed notes
 - » you may scribble notes / comments in your book however
- **Labs (50%)**
 - 7 labs (8-12% each)
- **Grading Characteristics**
 - Lab scores tend to be high
 - » Serious handicap if you don't hand a lab in
 - Tests typically have a wider range of scores

Normalized Grading

- **For each Lab or exam**
 - you will be graded against the best student in the class
 - best student by definition gets 100%
 - if the best student is too good then you'll be normalized against the Nth student (N is often 2 or 3)
- **Letter grades**
 - 90-100% of the best student \approx A
 - 80-89% \approx B
 - 70-79% \approx C
 - Etc.

Facilities

- **Machines**

- **16 x86 Linux machines in the “NT” Lab – EMCB210**
 - » can ssh to them as labnix1-labnix16.cs.utah.edu
- **You can also:**
 - » Use a CADE lab Linux machine
 - » Log into a Windows machine and ssh to a Linux machine
 - » Run Linux at home
 - » Etc.
- **But again: Projects must run on the EMCB 210 machines**
- **I’m working on getting some faster server boxes for the class**
- **You should all have accounts in the NT lab and in the teaching lab**

Programs and Data (8)

- **Topics**
 - Bits operations, arithmetic, assembly language programs, representation of C control and data structures
 - Includes aspects of architecture and compilers
- **Assignments**
 - L1: Manipulating bits
 - L2: Defusing a binary bomb
 - L3: Hacking a buffer bomb

Performance (3)

- **Topics**
 - High level processor models, code optimization (control and data), measuring time on a computer
 - Includes aspects of architecture, compilers, and OS
- **Assignments**
 - L4: Optimizing Code Performance

The Memory Hierarchy (2)

- **Topics**
 - Memory technology, memory hierarchy, caches, disks, locality
 - Includes aspects of architecture and OS.
- **Assignments**
 - L4: Optimizing Code Performance

Linking and Exceptional Control Flow (3)

- **Topics**

- Object files, static and dynamic linking, libraries, loading
- Hardware exceptions, processes, process control, Unix signals, nonlocal jumps
- Includes aspects of compilers, OS, and architecture

- **Assignments**

- L5: Writing your own shell with job control

Virtual memory (4)

- **Topics**
 - Virtual memory, address translation, dynamic storage allocation
 - Includes aspects of architecture and OS
- **Assignments**
 - L6: Writing your own malloc package

I/O, Networking, and Concurrency (6)

- **Topics**
 - High level and low-level I/O, network programming, Internet services, Web servers
 - concurrency, concurrent server design, threads, I/O multiplexing with select.
 - Includes aspects of networking, OS, and architecture.
- **Assignments**
 - L7: Writing your own Web proxy

Lab Rationale

- **Each lab should have a well-defined goal such as solving a puzzle or winning a contest.**
 - Defusing a binary bomb.
 - Winning a performance contest.
- **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
 - Perf Lab: profiling, measurement, performance debugging.
 - Shell Lab: understanding Unix process control and signals
 - Malloc Lab: understanding pointers and nasty memory bugs.
 - Proxy Lab: network programming, server 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!

Good Luck!