University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 391: Computer Systems Engineering

Introduction and Overview

Who are We?

Zbigniew Kalbarczyk

Res. Prof. ECE, CSL

At UIUC since … 1992

(MS from TU Warsaw, MS from TU Sofia, PhD from Bulgarian Acad. Sci.)

Research: Fault Tolerance, Reliable Computing, High-Performance Computing, Genomic Medicine

Education: 391, 542, and others

kalbarcz@illinois.edu

http://depend.csl.illinois.edu/

Steve Lumetta

Assoc. Prof. ECE, CS, CSL

At UIUC since 1998

(BS, MS, PhD Berkeley)

Research: Networks, Processors, Accelerators, High-Performance Computing, Genomics

Education: 3×CE core courses & others

lumetta@illinois.edu

http://lumetta.web.engr.illinois.edu/

And Your New Best Friends…

Xiang Li (head TA)

Hye Gang “Gregory” Jun

Huili Tao

Tianchen “Jerry” Wang

Linghao Zhang

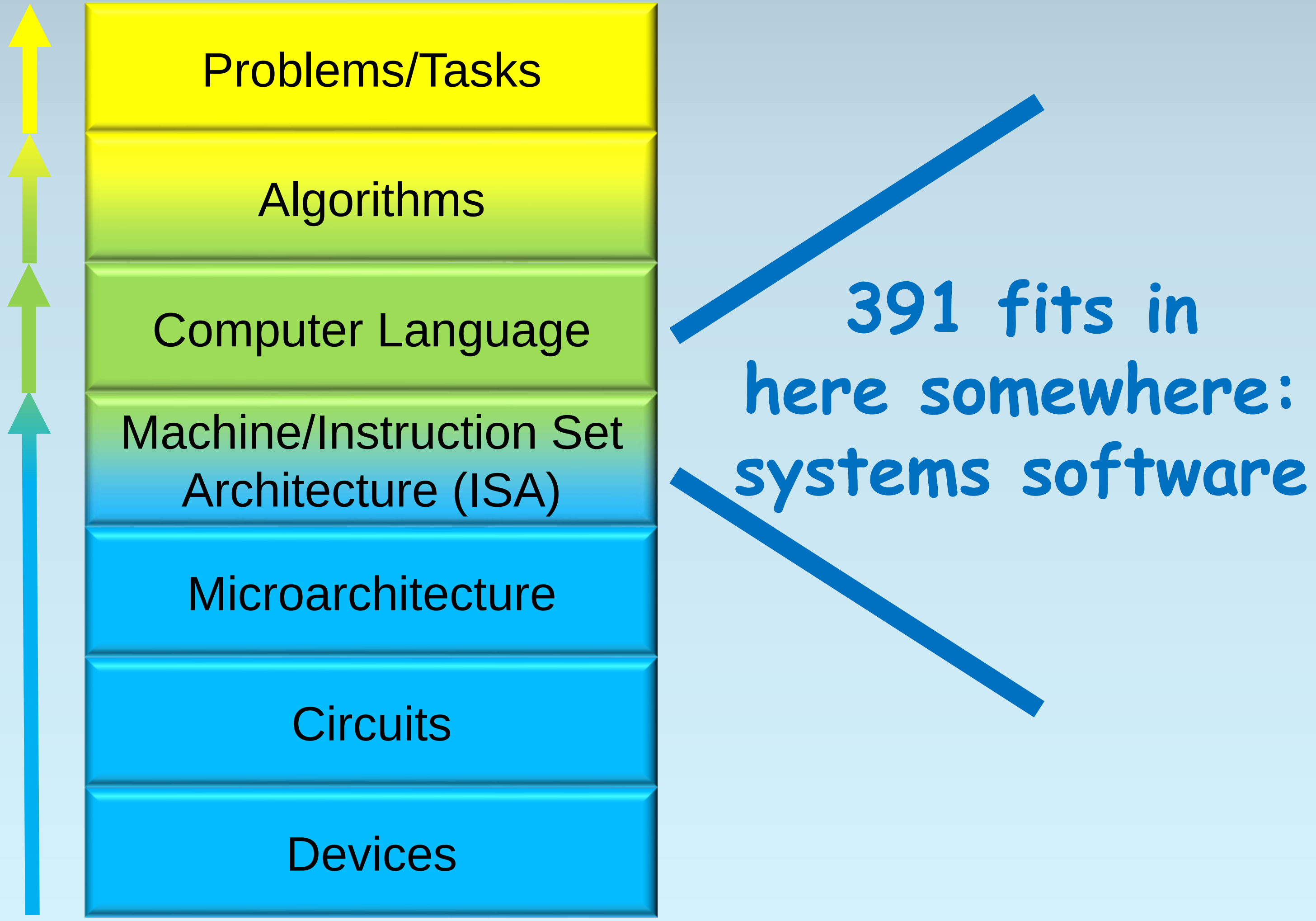
Yirui Zhou

plus 35 CAs!

All of these people **have done well in 391**—
and they **can help you!**

stolen from 220

~~Our Class Builds Upwards from ECE120~~



In ECE120, you learned to build a computer from bits and gates.

Let’s Start with the Knowledge Objectives

1. **understand the hardware-software interface in detail**
2. **understand how one abstracts physical resources in software**

three key roles for operating system (OS):

- **virtualization** – providing the illusion of larger or unlimited physical resources
- **abstraction** – providing simpler, more intuitive interfaces to hardware
- **protection** – ensuring that distinct programs and users do not interfere with one another’s operation

Operating Systems are Key to System Software

Why study operating systems (OSs)?

OSs are the **epitome of all software**: any idea tried in software has been tried in an OS.

Increased abstraction, even in embedded systems …

- 391 created before phones had OSs.
- Today, your watch may have an OS.
- Tomorrow, perhaps an OS will run in your shoe.

Is That a Computer in Your Shoe?

Sensors in sports shoes get all the attention, but other devices can identify you by how you walk and help Alzheimer's patients find their way home

Randy Rieland

391 Also Has Skill Objectives

1. **Tools—learn to use them! They make you happy.** (source control, compiler, dependency management, debugger, and so forth)
2. **learn team-based development and testing**
   - learn to communicate, leverage individual strengths, design workable interfaces for development and test
   - learn to create, implement, and execute testing plans
3. **learn to find necessary information**
   - reading specs and documentation
   - reading code

A Brief History of … This Class*

Some history:

- 391 was developed 18 years ago
- to fill a gap in CE curriculum
- consisting of two classes of material.
- We had one free semester course.
- It’s **meant to be your main class**.

Don’t take it with 374, 385, 411, or 445.

* Apologies to SH.

What You are Expected to Already Know

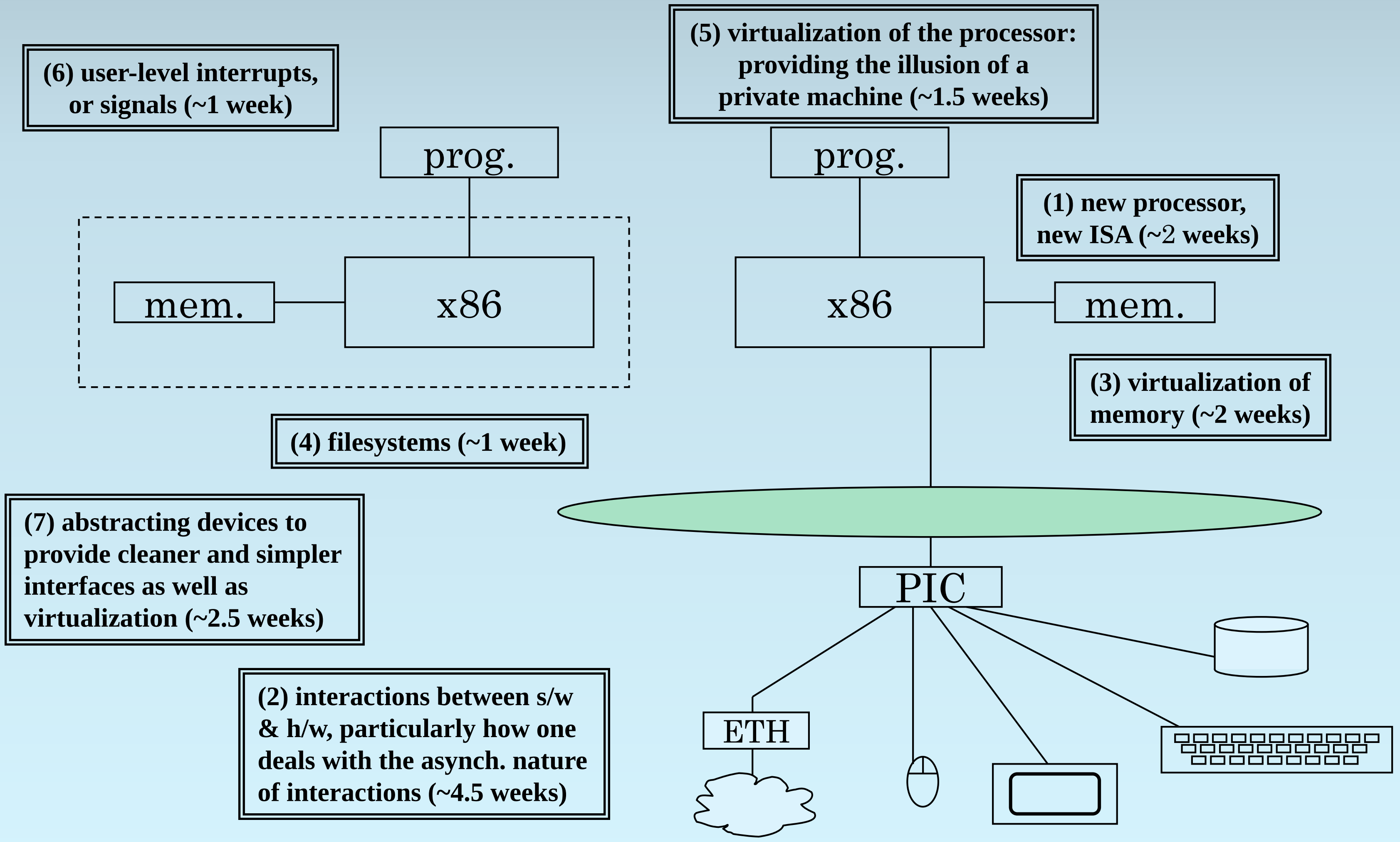
Things you should already know how to do (ECE220):

- **code in C**: significant experience writing small programs (hundreds of lines)
- **code in assembly**: median of 700 lines when Lumetta teaches 220

We **will use a few basic data structures**.

If you've finished CS225, you may find such discussions easier to follow, so we encourage (not require) that class first.

Big Picture of the Material



(6) user-level interrupts, or signals (~1 week)

(5) virtualization of the processor: providing the illusion of a private machine (~1.5 weeks)

(1) new processor, new ISA (~2 weeks)

(3) virtualization of memory (~2 weeks)

(4) filesystems (~1 week)

(7) abstracting devices to provide cleaner and simpler interfaces as well as virtualization (~2.5 weeks)

(2) interactions between s/w & h/w, particularly how one deals with the asynch. nature of interactions (~4.5 weeks)

Let's Hear from the Audience

(quotes about 391 from alumni…)

LMGTFY

ECE 391 provided the foundation necessary to be succeed in my CPU design verification internship at Apple. Even though I wrote ARM assembly, the **fundamental ideas taught in lecture and the hours reading dense technical documentation (Intel Manual) made it easy to learn quickly** from the corresponding ARM manuals and Apple documents.

Eric Clark, ECE 391 in Fall of '13, now at Apple

Coding in Practice

However, the value of the ECE 391 to me extends beyond memorizing particular concepts; it was in ECE 391 that I got my first real experience working in unfamiliar codebases and debugging challenging problems. **In my professional career, I have found that much of my time is spent reading new code and figuring out how to implement new features that that my org needs**; these skills have transferred well from ECE 391.

Matt Tischer, Spring '12, Software Engineer at Microsoft

Real systems are messy

ECE 391 is pretty much the only class in college that challenged my coding skills. ECE 190 … and CS 225 gave me some exposure to coding, but at that point I was still learning. 391 was the first time that I got to actually work on a large code. **In my job today, I don't have the luxury of the answers already being figured out… There are no instructions, no hints, no TA office hours, no nothing**. It's just me and hours of banging my head against a wall. But I'd like to think that 391 prepared me well and so I suffer from much less head-banging than I likely would have otherwise.

Eric Carl of House Badger, the First of His Name, The Unallocated, King of the Stackman, the Bitdiddle and the First Ben, King of Execute, Kerneleesi of the Great Language C, Protector of the Stack, Lord Regnant of the Seven Syscalls, Breaker of Segments and Father of Pages, Spring 2012, Hadoop Code Monkey at Oath: the company formerly known as Yahoo!

Writing good code

There were so many things I learnt in ECE 391, but the most important one is **how to write good code and why do we need to do so**. ECE 391 prepared me really well for my job, aside from the point mentioned above, it also taught me how to think as an computer system engineer.

Fei Deng, Spring'14, Software Development Engineer at Yahoo! Inc.

Systems Thinking

Computer systems and cars are similar machines: in the sense that you don't have to know how to build a car in order to drive or fix one, nor will it make you an expert driver or technician (software engineering and I.T. in the computer world), but **if you understand the fundamental physics you will never be surprised with something you can't understand or learn**.

Daniel Fernandes, Fall of '13, EE PhD Student at Stanford University

Systems Thinking II

391 benefited me greatly in my career … Even with architectural differences of working on Windows, **391 helped teach me how to approach and solve system-level problems**. I also spend a significant amount of time debugging crash dumps, so the debugging experience from the MPs was very helpful, as well as being able to look at disassembly with the knowledge of x86 and calling conventions.

Drew Kluemke, Fall 2015, Microsoft

Debugging

I think taking ECE 391 has earned me an enormous amount of respect in the workplace … one other thing I think is hugely important in ECE 391 is that you learn to be very methodical in how you debug and write code … I think this skill, perhaps more than any other, will impress your coworkers and bring value. **I've had cases where people asked me "how did you find that bug?" when the answer was simply I opened the debugger and stepped through the code.**

Dennis Liang, Spring '14, Systems Engineer at Cloudflare

Groups

ECE 391 gave me experience in working on a large **coding project within a group**… The course also taught me how to better **collaborate with group members** and divide work in ways so group/team members don’t “step on each other's toes.”

Michael F., Fall 2015, now Software Engineer at Microsoft.

Let’s Talk about the Course Elements

lecture—concepts and some of the practical side

discussion—help with concepts and labs

recorded to Media Space

pre-labs—alternate with labs; helps to prepare you for labs

labs (machine problems, MPs)—hands-on experience with tools and real systems

tests (mostly non-cumulative)

- some emphasis on lab material
- some emphasis on conceptual material
- see web page for guides and previous exams

Course Materials: Notes and References

class notes—terse and to the point

books—more detail than needed, but good as references and introductory material

- *The C Programming Language*
- *Advanced Unix Programming*, 2nd ed.
- LDD (*Linux Device Drivers*, 3rd ed.
- ULK (*Understanding the Linux Kernel*), 3rd ed.

See web page for more resources.

Information about the Class

Piazza: Go here first! (piazza.com/illinois/fall2022/ece391)

- Announcements, errata, and discussion all go here.
- **Read it every day.**
- Anything not personal? **Post it to Piazza**:
  - defaults to anonymous to other students, but
  - mark “private” and they won’t see at all.
- **DO NOT POST SOLUTIONS.**
- **Don’t repeat questions.**

Check the web page for things that don’t change often:

https://courses.grainger.illinois.edu/ece391/fa2022/

Use e-mail only as a last resort.

Workload and Grading

Workload includes

- four programming assignments, two with pre-lab,
- two midterms, and one final.

Grading:

- **50% on labs** (5%, 10%, 10%, 25%) (prelabs worth 10% of associated lab grade),
- **15%** on each test, and
- 5% subjective.

What About Discussion Section?

No points for discussion section?

Technically, no.

But they were **created in response to student demand**.

They **will help you do better** in the course.

We **STRONGLY ENCOURAGE you to attend**.

Go to one, go to two, watch the video, whatever you like.*

*We do have to obey fire marshal rules about people in any space.

Exam Timing

When are the exams?

MT1: Tuesday 27 September, 7:00-9:00 p.m.

MT2: Tuesday 1 November 7:00-9:00 p.m.

FIN: Tuesday 13 December 7:00-10:00 p.m.

Let us know about any exam **conflict** ASAP, but **no later than a week before the exam**.

Collaboration Allowed for Prelabs

OK to talk to staff at any time.

Pre-labs are completely open—work with anyone

- We encourage you to do so.
- **Don’t screw yourself** by putting your name on something you didn’t do.
- **Turn in one copy with all names and net ids** to be credited (save us time on grading)
- (if your name is on >1 submission, you earn the minimum grade over all those with your name)

Collaboration NOT Allowed for Labs and Exams

machine problems (labs)

- **MPs 0, 1, and 2 are individual work**
- **MP3 is done with a team of four**
- **collaboration is NOT allowed**
- OK to talk with others about ideas, strategies.

tests—all by yourself

Academic Integrity Violations…Just Don't Do It

Any unauthorized provision or use of code

- not written by you (or your partners, for MP3)
- **is a violation of academic integrity**.

Such violations will result, at a minimum, of

- a **0 on the assignment**,
- **loss of one full letter grade**, and
- a report to the college through FAIR (which may be FOIAble, and may result in your expulsion if you have or develop a history of such behavior)

Additional violations will result in failing the class.

Examples of Prohibited Actions

- **Turning in any portion of another's code as your own** (whether from a current student, a former student, a Github repo, a Co-Pilot session, a Git-a-Coder payment, or anything else).
- **Making your solution code available to a current student** (whether by printing it, posting it to Github, sending by email, or any other means).
- **Using another's code** (see list above) **as a hint or template** to develop your own solution.

MP 3 Offers a Design Competition

In **MP3**, in the **second half of** the **semester**,
- you'll work with a **team of four** (not 3, not 5)
- **to develop your own OS**.

For the hyper-motivated,
- we have a **design competition,**
- **sponsored by Microsoft** since 2006.

Who’s on Your Team?

So do you want to …

- win the competition,
- get an A, or
- just pass the class?

(Good to have teammates with similar answers.)

Let’s Talk about the Environment

No, not that environment.

The *class* environment.

Let’s Talk about the Environment

It’s natural to be intimidated
by kernel (OS) work.

What happens if you mess up?

- Could lose all of your homework.
- Or, worse, your photos and music!
- Maybe you’ll have to buy a new computer.

We can’t let you work on lab machines…

…but you all have laptops, right?

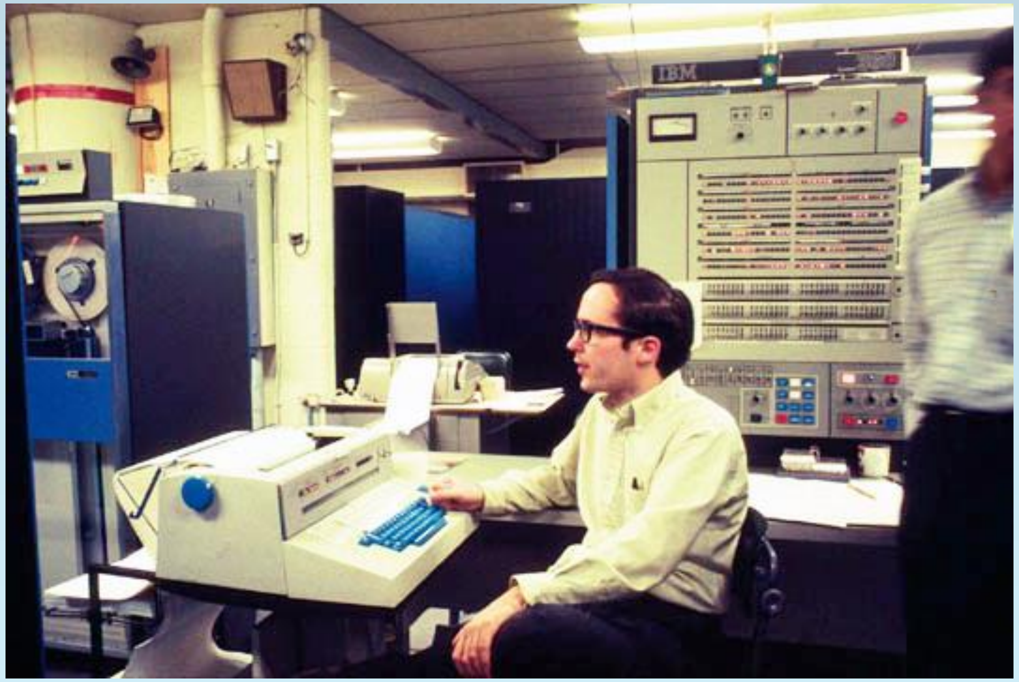
Change of Topic … Simulators!

Who has written a simulator?

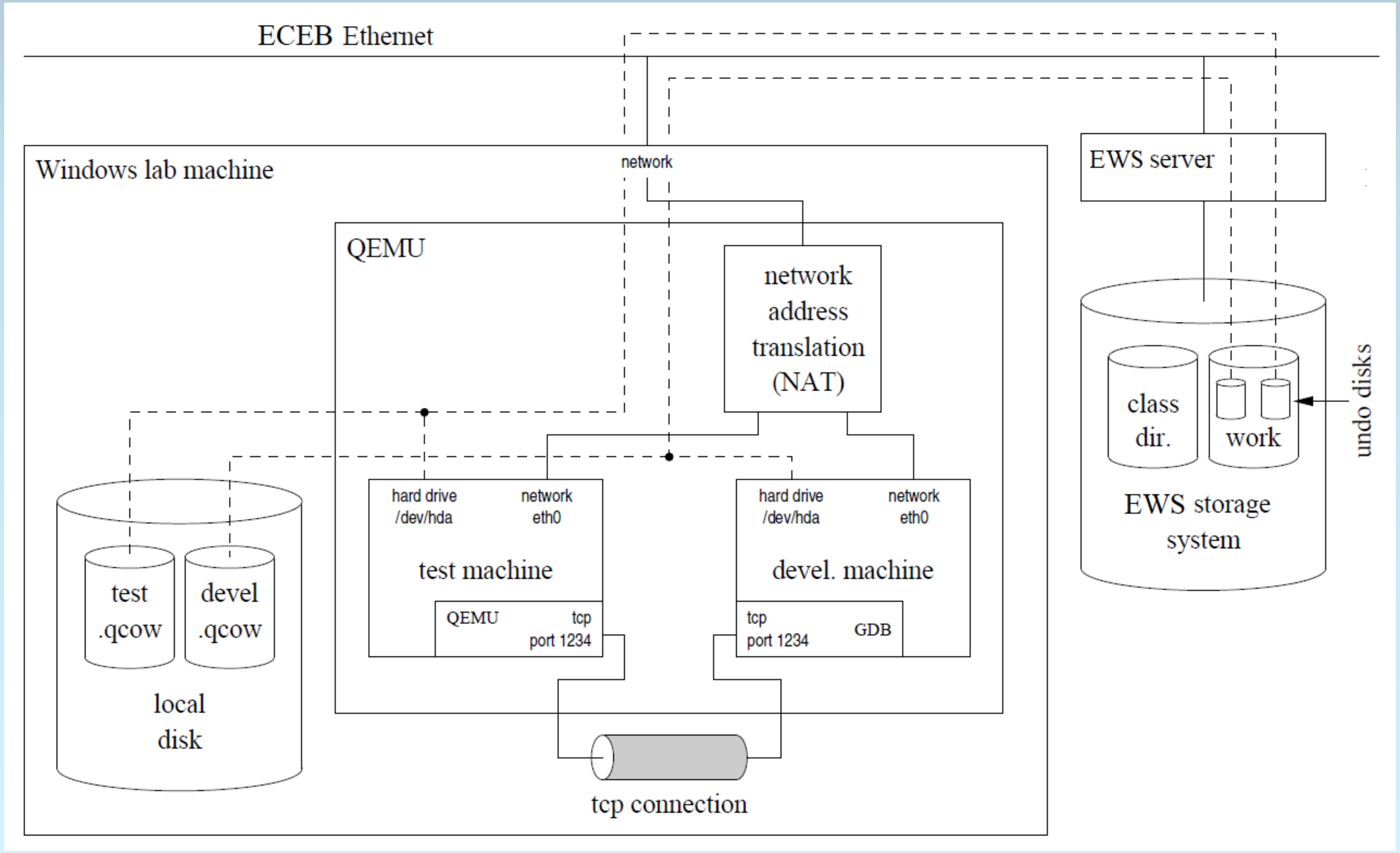
(An LC-3 simulator was one of two MPs in ECE190, the predecessor of ECE220.)

In the 1960s, IBM developed a solution!

In our class, **we will use a simulator** that executes most code natively (the original meaning of "**virtual machine**").



Details of the ECE391 Lab Environment



MP 0 … Have You Started Yet?

Set up your personal environment in **MP0**,

- **due** in (any TA/CA's) office hours **by Wednesday 31 August**.
- **Demo in person in 3026 ECEB.**

You may also want to work on your own machines:

- https://courses.grainger.illinois.edu/ece391/sp2022/secure/references/doc-workhome_novpn.pdf
- allows you to develop and test MP code
- without competing for lab resources.

Lab Guidelines for MP 0 and Beyond

There will be contention for lab resources, so **start early**!

For MP 0,
- you may **lock** one lab computer for **up to 4 hours**,
- if you **leave a note** with expected return time.

After MP 0, the lock limit is **30 minutes.**

Hand-in/demo takes precedence for machine use and TA/CA time.

Working from home is encouraged, but **demos must happen on the lab machines**.

Please be respectful to others.

More Advice from the Age~~d~~s

I think ECE 391 can be a bit of a grind for some students because of its difficulty. However, consider this: **it's *worth* learning because it's difficult.** You shouldn't shy away from learning difficult things, you should welcome it. Learning doesn't stop when you're done with ECE 391, …..

It is an important class, and a difficult one for many students, but **you don’t have to be a genius to do well**. The same hard work and dedication that got you here will carry you through this class, too.

More Advice from the Ages

My advice is, and has always been, **RTDC (Read the Documentation Carefully)**. Read whatever the class provides at least twice before starting to solve the problem or write code. There were so many times students ask questions that can be answered with a direct quote from the docs we gave out.

More Advice from the Ages

Go to discussions, pay attention to the TA since they've all done well in ECE 391. **Go to office hours, ask good questions** but don't expect the TA to help you debug.

Talk to the TAs and your classmates. **Make sure you really understand what the class is asking you to do and why** you're doing it.

Make use of office hours. The 391 staff is relatively large and hosts frequent office hours in the lab. Maximize the value of the questions you ask; "It looks like I'm clobbering my stack somewhere in lines 10-20" is more likely to get useful results than "something is wrong with my code".

More Advice from the Ages

Get comfortable with your debugging setup--you'll be using it a lot.

- Learn to use GDB, don't rely on "printf" to debug, that's what newbies do.

Breaking down large tasks into smaller segments of code and **testing** each segment **early and often** really makes debugging easier as there are less items that can go wrong at each step.

More Advice from the Ages

Starting early is the most obvious but also most helpful piece of advice. If you spend a few hours a day on 391, you'll be a lot more effective than trying to complete a checkpoint within a 48 hour timespan. As a TA, I saw so many groups that were trying to rush to complete checkpoints in a day or two, and all of them looked miserable.

My advice to ECE 391 students is to **start early** on assignments and stay ahead of deadlines. Bugs will crop up at the worst possible moment

Everything will **take longer than you think**, so you need to plan ahead if you want to sleep around finals week.

Time management in ECE 391 is critical. While it may be possible to complete MPs in a single night in some other classes, ECE 391 is not among them.

Plan ahead. This sounds so simple, but it’s actually the hardest thing to do, because of the temptation of cutting corners. It may feel like a waste of time, but it will save you countless hours of debugging in the future.

We're All Done for Today

Any questions?