Hack Your Language!

CS164: Introduction to Programming Languages and Compilers

Lecture 1, Fall 2009

Instructor: **Ras Bodik**

GSI: **Joel Galenson**

UC Berkeley

Before we start

Lectures are not mandatory.

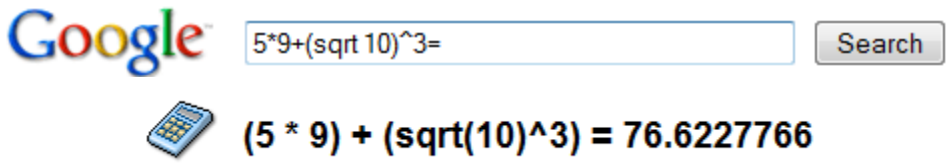
In case you decide to attend,

turn off your **cell phone** ringers, and
close your **laptops**.

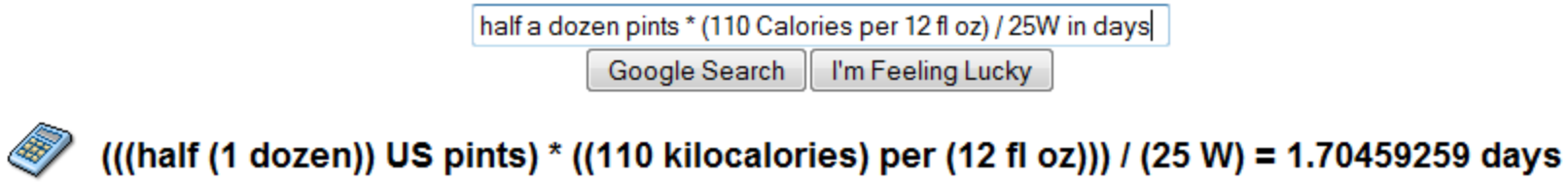
Seriously.

1. You work in a little web search company

Your boss says: *"We will conquer the world only if our search box answers all questions the user may ask."* So you build gcalc:



Then you remember cs164 and easily add unit conversion:



This is how long a brain can live on six beers, energy-wise.

CS164 helps you answer questions your boss really cares about.

2. Then you work in a tiny browser outfit

You observe JavaScript programmers and take pity. Instead of

```
var nodes = document.getElementsByTagName('a');
for (var i = 0; i < nodes.length; i++) {
    var a = nodes[i];
    a.addEventListener('mouseover', function(event) { event.target.style.backgroundColor='orange'; }, false );
    a.addEventListener('mouseout', function(event) { event.target.style.backgroundColor='white'; }, false );
}
```

iteration is hidden

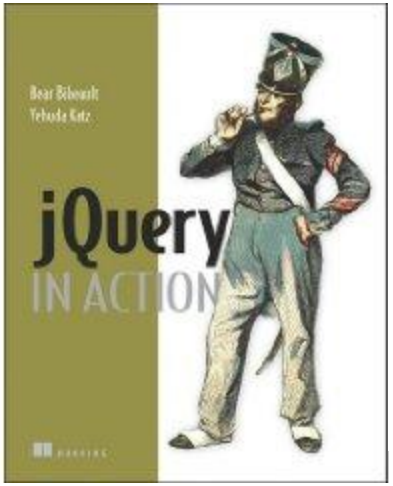
events are hidden

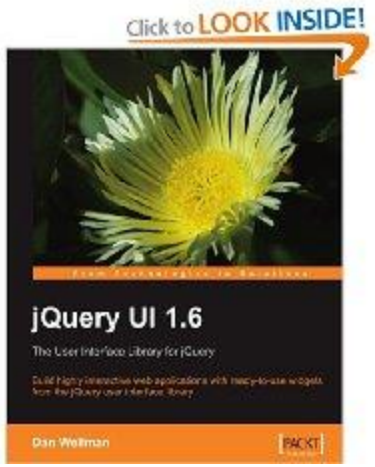
you let them write this, abstracting from iteration and events

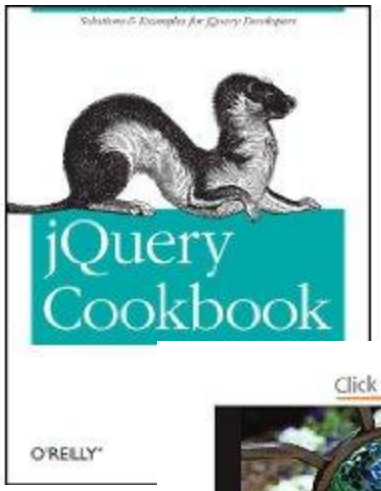
```
jQuery('a').hover( function() { jQuery(this).css('background-color', 'orange'); },
                   function() { jQuery(this).css('background-color', 'white'); } );
```

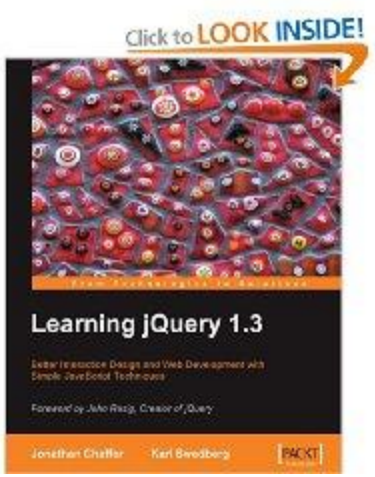
jQuery was developed by John Resig, now at Mozilla

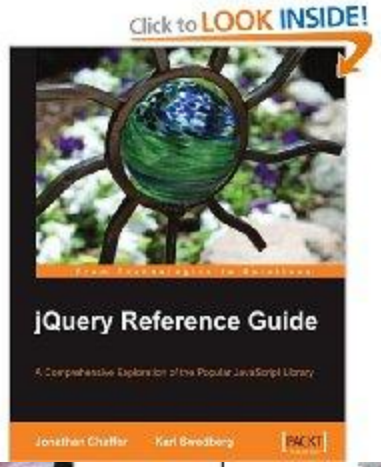
… and the fame follows

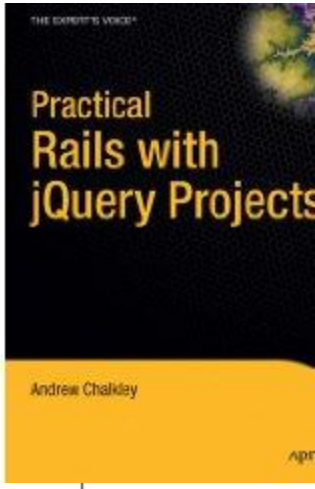


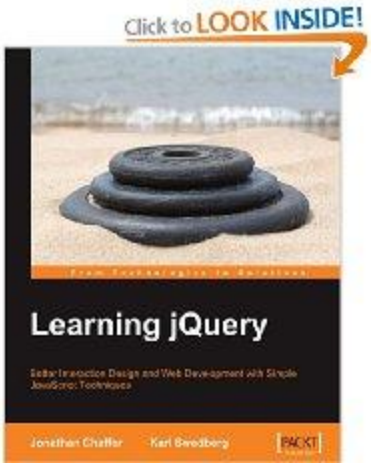


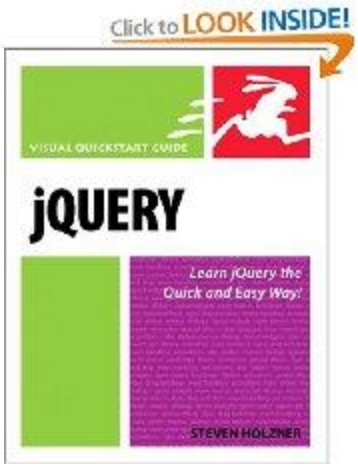


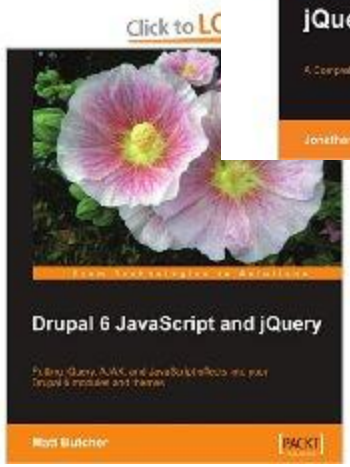






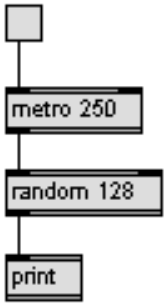






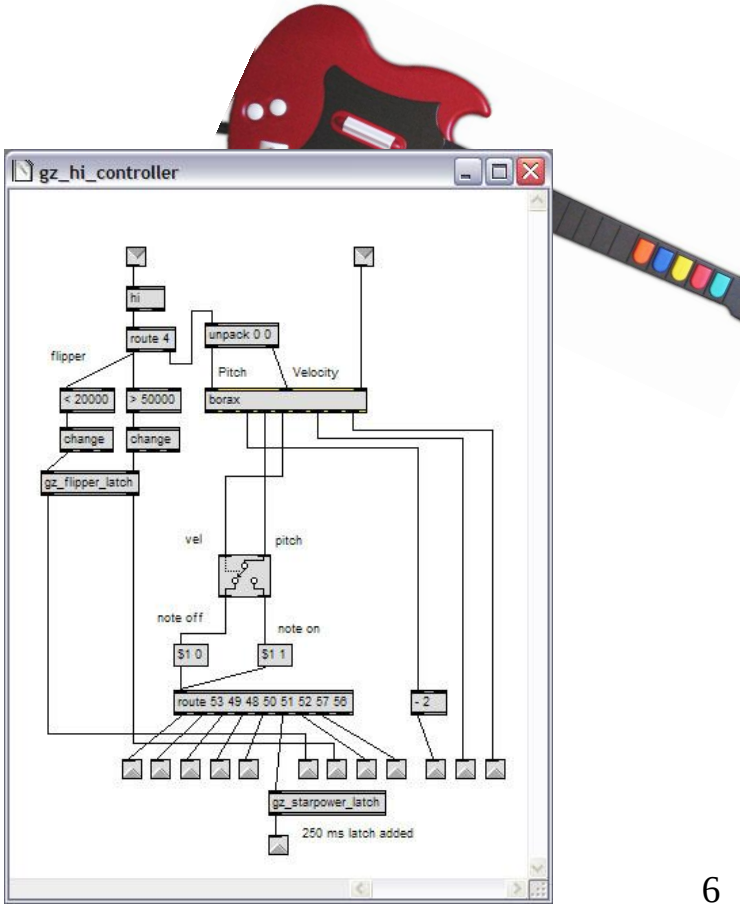
3. Then you write visual scripting for musicians

Allowing non-programmers produce interactive music by "patching" visual metaphors of electronic blocks:



watch this youTube video

http://www.youtube.com/watch?v=uxzPCt7Pbds

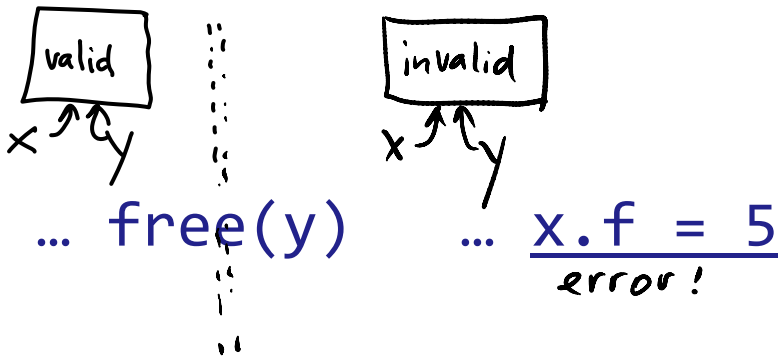


Guitair Zeros: a S.F. band enabled by the Max/MSP language.

Max/MSP was created by Miller Puckette and is now developed by Cycling '74.

4. Then you live in open source

You see Linux developers suffer from memory bugs, such as buffer overruns, and dangling pointers (i.e., accesses to freed memory).



You design a tool that associates each bit in memory with a valid bit, set by new and reset by free. When a location is accessed, you check its validity.

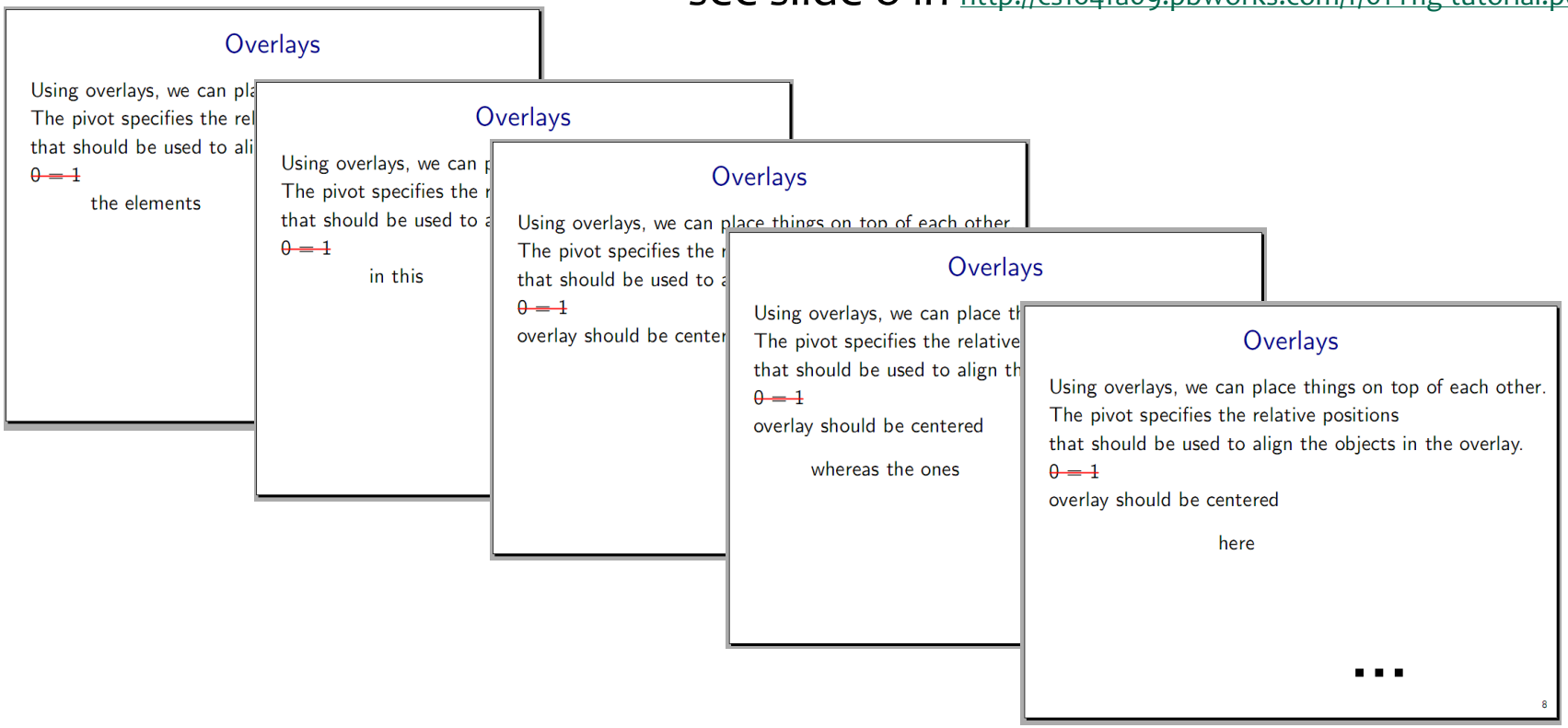
To add these checks, the implementation rewrites the binary of the program, and adds shadow memory.

Valgrind was developed by Julian Seward

5. Then you decide to get a PhD

You get tired of the PowerPoint and its animations.
Or you simply are not a WYSIWIG person.
You embed a domain-specific language (DSL) into Ruby.

see slide 8 in http://cs164fa09.pbworks.com/f/01-rfig-tutorial.pdf



The animation in rfig, a Ruby-based language

```
slide!('Overlays',

  'Using overlays, we can place things on top of each other.',

  'The pivot specifies the relative positions',

  'that should be used to align the objects in the overlay.',

  overlay('0 = 1', hedge.color(red).thickness(2)).pivot(0, 0),

  staggeredOverlay(true, # True means that old objects disappear
    'the elements', 'in this', 'overlay should be centered', nil).pivot(0, 0),

  cr, pause,           # pivot(x, y): -1 = left, 0 = center, +1 = right

  staggeredOverlay(true,
    'whereas the ones', 'here', 'should be right justified', nil).pivot(1, 0),

  nil) { |slide| slide.label('overlay').signature(8) }
```

rfig was developed by Percy Liang, now a Berkeley student

More examples of how cs164 will help

6. Roll your own make/ant in Python (Bill McCloskey)
7. Ruby on Rails (another system on top of Ruby)
8. Choose the right language (reduce lines 10x)
9. Custom scripting languages (eg for testing)
10. Custom code generators (eg for new hardware)

Take cs164. Become unoffshorable.

"We design them here, but the labor is cheaper in Hell."

How is this compiler class different?

Not a compiler class at all.

It's about:

a) foundations of programming langauges
b) but also how to design your own languages
c) how to implement them
d) and about PL tools, such as analyzers
e) Ans also learn about some classical C.S. algorithms.

How is this PL/compiler class different?

Not intended for future compiler engineers

- there are few among our students

... but for software developers

- raise your hand if you plan to be one

But why does a developer or hacker need a PL class?

- we'll take a stab at this question today

Why a software engineer/developer needs PL

New languages will keep coming

- Understand them, choose the right one.

Write code that writes code

- Be the wizard, not the typist.

Develop your own language.

- Are you kidding? No.

Learn about compilers and interpreters.

- Programmer's main tools.

Overview

- how many languages does one need?
- trends in programming languages
- ... and why they matter to you
- course logistics

New Languages will Keep Coming

A survey: how many languages did you use?

- Let's list them here:

in the browser alone:

- HTML / XHTML
- XML
- JavaScript / Actionscript / Flash
- CSS
- jQuery
- Prototype
- Xpath
- Silverlight

JSON

Be prepared to program in new languages

Languages undergo constant change

- FORTRAN 1953
- ALGOL 60 1960
- C 1973
- C++ 1985
- Java 1995

Evolution steps: 12 years per widely adopted language

- are we overdue for the next big one?

... or is the language already here?

- *Hint:* are we going through a major shift in what computation programs need to express?
- your answer here: JavaScript

- programs are distributed
- more interactive
- "installed" as part of web page

Develop your own language

Are you kidding? No. Guess who developed:

- PHP
- Ruby
- JavaScript
- perl

Done by smart hackers like you

- in a garage
- not in academic ivory tower

Our goal: learn good academic lessons

- so that your future languages avoid known mistakes

Trends in Programming Languages

Trends in programming languages

programming language and its interpreter/compiler:

- programmer's primary tools
- you must know them inside out

languages have been constantly evolving ...

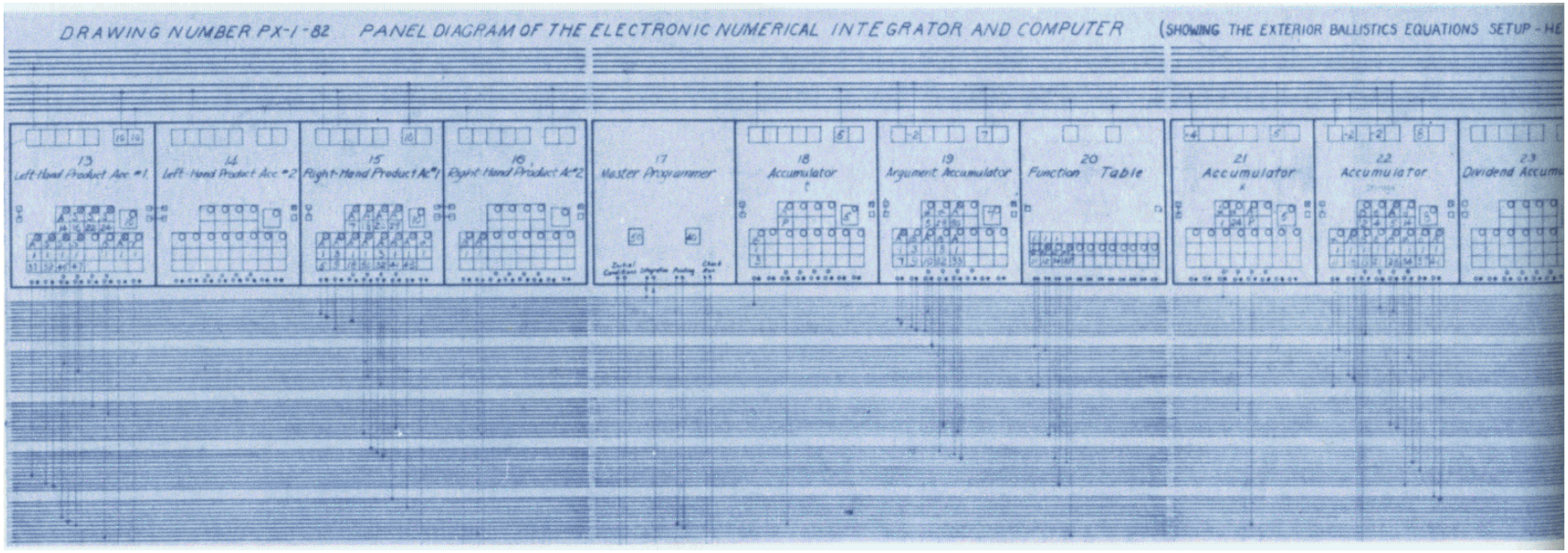
- what are the forces driving the change?

... and will keep doing so

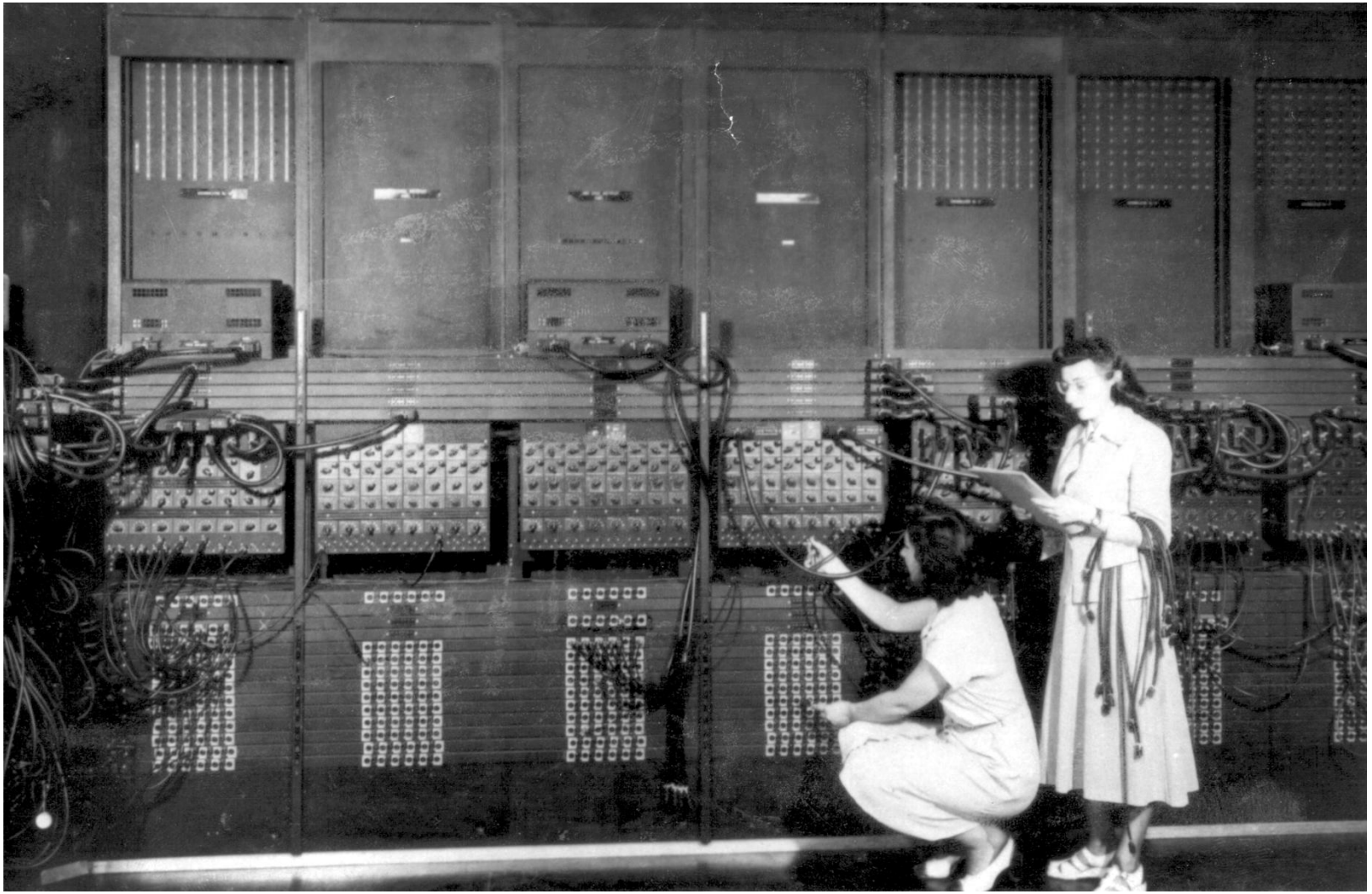
- to predict the future, let's examine the history…

ENIAC (1946, University of Philadelphia)

ENIAC program for external ballistic equations:



Programming the ENIAC



ENIAC (1946, University of Philadelphia)

programming done by

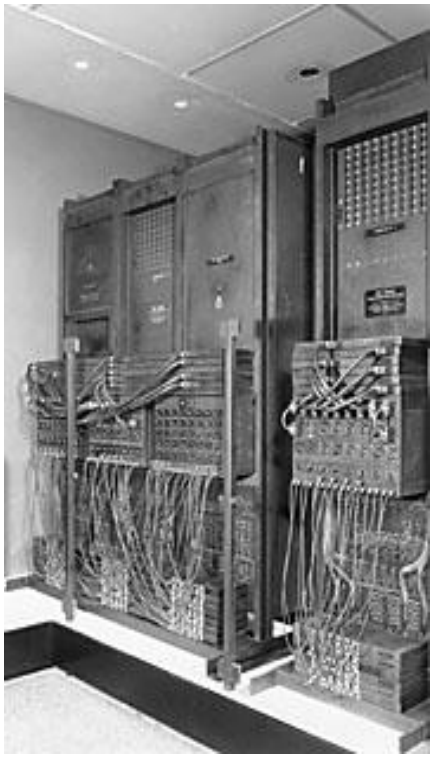
- rewiring the interconnections
- to set up desired formulas, etc

Problem (what's the tedious part?)

- programming = rewiring
- slow, error-prone

solution:

- **store the program in memory!**
- birth of ***von Neuman*** paradigm



UDSAC (1947, Cambridge University)

the first real computer

- large-scale, fully functional, **stored-program** electronic digital computer (by Maurice Wilkes)

problem: Wilkes realized:

- "a good part of the remainder of my life was going to be spent in finding errors in ... programs"

solution: procedures (1951)

- procedure: abstracts away the implementation
- reusable software was born

Assembly – the language (UNIVAC 1, 1950)

Idea: mnemonic (assembly) code

- Then translate it to machine code by hand (no compiler yet)
- write programs with mnemonic codes (add, sub), with symbolic labels,
- then assign addresses by hand

Example of symbolic assembler

clear-and-add a

add b

store c

translate it by hand to something like this (understood by CPU)

B100 A200

C300

Assembler – the compiler (Manchester, 1952)

- it was assembler nearly as we know it, called AutoCode
- a loop example, in MIPS, a modern-day assembly code:

```
loop: addi $t3, $t0, -8
      addi $t4, $t0, -4
      lw $t1, theArray($t3)        # Gets the last
      lw $t2, theArray($t4)        #  two elements
      add $t5, $t1, $t2            # Adds them together...
      sw $t5, theArray($t0)        # ...and stores the result
      addi $t0, $t0, 4             # Moves to next "element"
                                   # of theArray
      blt $t0, 160, loop           # If not past the end of
                                   # theArray, repeat
      jr $ra
```

Assembly programming caught on, but

Problem: Software costs exceeded hardware costs!

John Backus: "Speedcoding"

- An interpreter for a high-level language
- Ran 10-20 times slower than hand-written assembly
  - way too slow

FORTRAN I (1954-57)

Langauge, and the first compiler

- Produced code almost as good as hand-written
- Huge impact on computer science (laid foundations for cs164)
- Modern compilers preserve its outlines
- FORTRAN (the language) still in use today

By 1958, >50% of all software is in FORTRAN

Cut development time dramatically

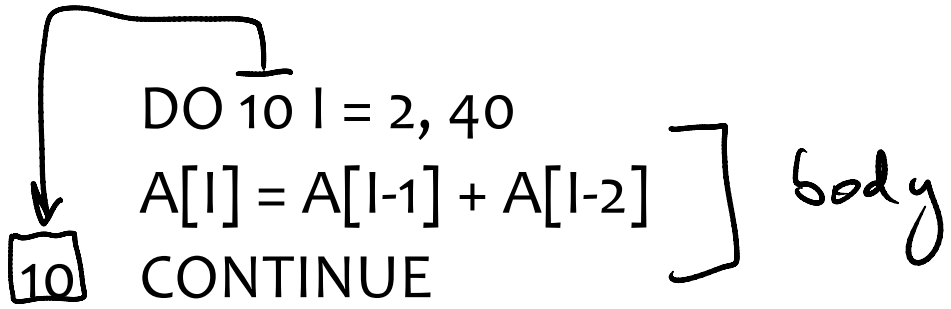
- 2 weeks → 2 hrs
- that's more than 100-fold

FORTRAN I (IBM, John Backus, 1954)

Example: nested loops in FORTRAN

- a big improvement over assembler,
- but annoying artifacts of assembly remain:
  - labels and rather explicit jumps (CONTINUE)
  - lexical columns: the statement must start in column 7

- The MIPS loop from previous slide, in FORTRAN:



Side note: designing a good language is hard

Good language protects against bugs, but lessons take a while.
An example that caused a failure of a NASA planetary probe:

buggy line:

```
DO 15 I = 1.100
```

what was intended (a dot had replaced the comma):

```
DO 15 I = 1,100
```

because Fortran ignores spaces, compiler read this as:

```
DO15I = 1.100
```

which is an assignment into a variable DO15I, not a loop.

This mistake is harder to make (if at all possible) with the modern lexical rules (white space not ignored) and loop syntax

```
for (i=1; i < 100; i++) { … }
```

Goto considered harmful

L1: statement

if expression goto L1

statement

Dijkstra says: gotos are harmful

- use structured programming
- lose some performance, gain a lot of readability

how do you rewrite the above code into structured form?

Object-oriented programming (1970s)

The need to express that more than one object supports draw() :

```
draw(2DElement p) {
    switch (p.type) {
      SQUARE: … // draw a square
                break;
       CIRCLE: … // draw a circle
                break;
    }
}
```

Problem:

unrelated code (drawing of SQUARE and CIRCLE) mixed in same procedure

Solution:

Object-oriented programming with inheritance

Object-oriented programming

In Java, the same code has the desired separation:

```
class Circle extends 2DElement {
  void draw() { <draw circle> }
}
class Square extends 2DElement {
  void draw() { <draw circle> }
}
```

the dispatch is now much simpler:

```
p.draw()
```

Review of historic development

- wired interconnects → stored program (von Neuman machines)
- lots of common bugs in repeated code → procedures
- machine code → symbolic assembly (compiled by hand)
- tedious compilation → assembler (the assembly compiler)
- assembly → FORTRAN I
- gotos → structured programming
- hardcoded "OO" programming → inheritance, virtual calls

Do you see a trend?

- Removal of boilerplate code
  - also called **plumbing**, meaning it conveys no application **logic**
- Done by means of new abstractions, such as procedures
  - They abstract away from details we don't wish to reason about

Where will languages go from here?

The trend is towards higher-level abstractions

- express the algorithm concisely!
- which means hiding often repeated code fragments
- new language constructs hide more of these low-level details.

Also, detect more bugs when the program is compiled

- with stricter type checking
- with tools that loom for bugs in eth program or in its execution

… and why the trends matter to you

Take cs164. Become unoffshorable.

"We design them here, but the labor is cheaper in Hell."

Don’t be a boilerplate programmer

Build tools for others

- libraries
- frameworks
- code generators
- small languages (such as configuration languages)
- big languages

course logistics

(see course info on the web for more)

Administrativia

Wed section at 5pm

- We'll try to move it
- Can you suggest times that would work for you?

Waitlist

- Hopefully all of you will get in

Academic (Dis)honesty

Read the policy at:

- http://www.eecs.berkeley.edu/Policies/acad.dis.shtml

We'll be using a state-of-the art plagiarism detector.

- Before you ask: yes, it works very well.

You are allowed to discuss the assignment

- but you must acknowledge (and describe) help in your submission.

Grading

- Grades will follow department guidelines
  - course average GPA will be around 2.9 (before extra credit for the optimization contest)
  - more at http://www.eecs.berkeley.edu/Policies/ugrad.grading.shtml
  - this has proven to be fair
- A lot of grade comes from a project
  - form a strong team
  - use the course newsgroup to find a partner

Conclusion