Programming Languages Topic of Ultimate Mastery

Wes Weimer

EECS 590

http://web.eecs.umich.edu/~weimerw/590/

Reasonable Initial Skepticism

Today's Class

- Vague Historical Context
- Goals For This Course
- Requirements and Grading
- Course Summary

- Where is PL most useful?

Meta-Level Information

- Please interrupt at any time!
- Completely reasonable questions:
  - I don't understand: please say it another way.
  - Slow down, you talk too fast!
  - Wait, I want to read that!
  - I didn't get joke X, please explain.

What Have You Done For Us Lately?

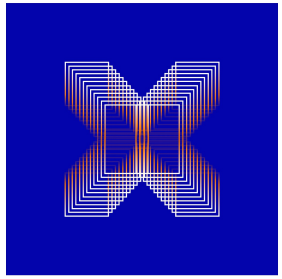
- PL is an old field within Computer Science
  - 1920's: "computer" = "person"
  - 1936: Church's Lambda Calculus (= PL!)
  - 1937: Shannon's digital circuit design
  - 1940's: first digital computers
  - 1950's: FORTRAN (= PL!)
  - 1958: LISP (= PL!)
  - 1960's: Unix
  - 1972: C Programming Language
  - 1981: TCP/IP
  - 1985: Microsoft Windows
  - 1992: Ultima Underworld / Wolfenstein 3D
  - 2001: 3G Cellphones

A Brief Tour

- … of PL research impact at companies

- Themes:
  - Multiple types of companies make languages
  - PL tools apply to many domains
  - PL research is embedded in other hardware and software
  - PL is interdisciplinary

OpenAI's Codex / Copilot



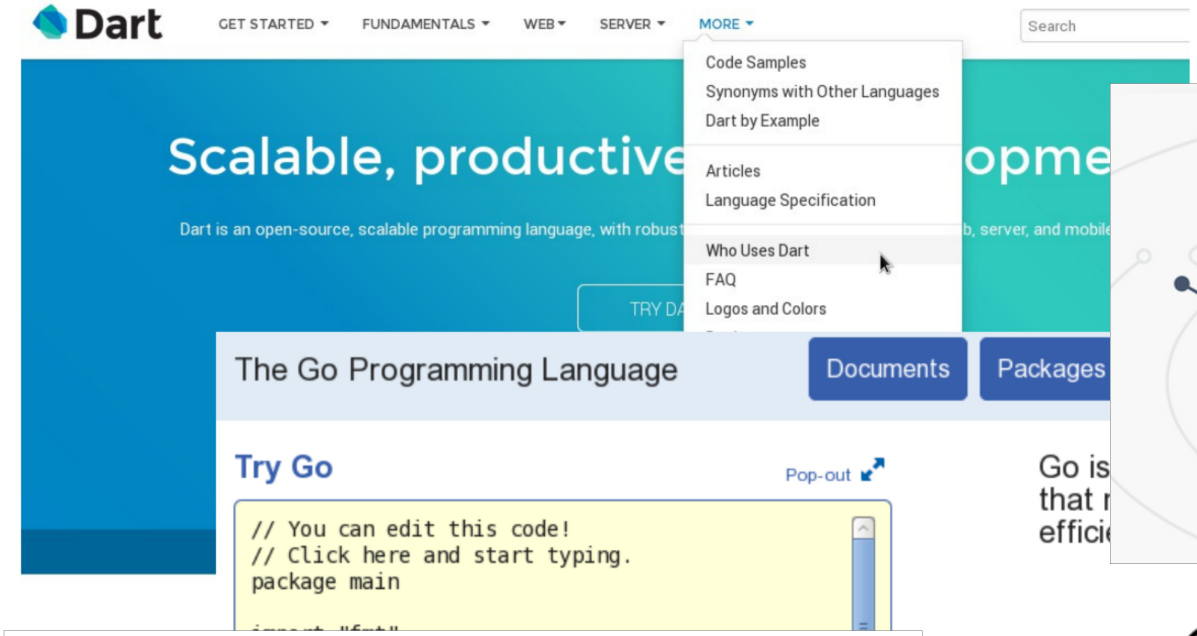
- (More detail on LLMs later in the course)

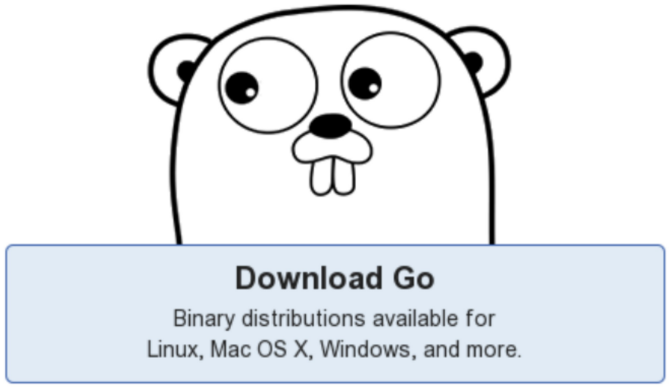
Codex is the model that powers GitHub Copilot, which we built and launched in partnership with GitHub a month ago. Proficient in more than a dozen programming languages, Codex can now interpret simple commands in natural language and execute them on the user's behalf—making it possible to build a natural language interface to existing applications. We are now inviting businesses and developers to build on top of OpenAI Codex through our API.

OpenAI Codex is a descendant of GPT-3; its training data contains both natural language and billions of lines of source code from publicly available sources, including code in public GitHub repositories. OpenAI Codex is most capable in Python, but it is also proficient in over a dozen languages including JavaScript, Go, Perl, PHP, Ruby, Swift and TypeScript, and even Shell. It has a memory of 14KB for Python code, compared to GPT-3 which has only 4KB—so it can take into account over 3x as much contextual information while performing any task.

Google

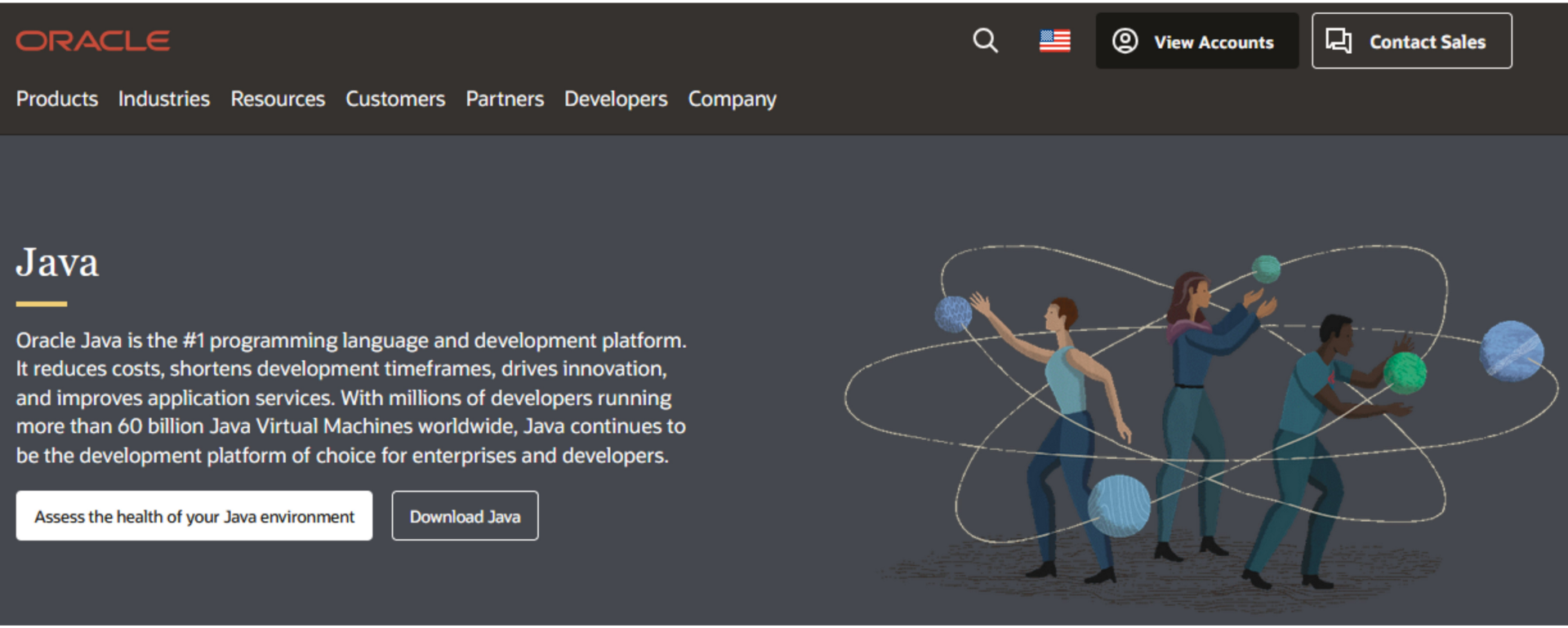
- Go, TensorFlow, Carbon, Dart, etc.





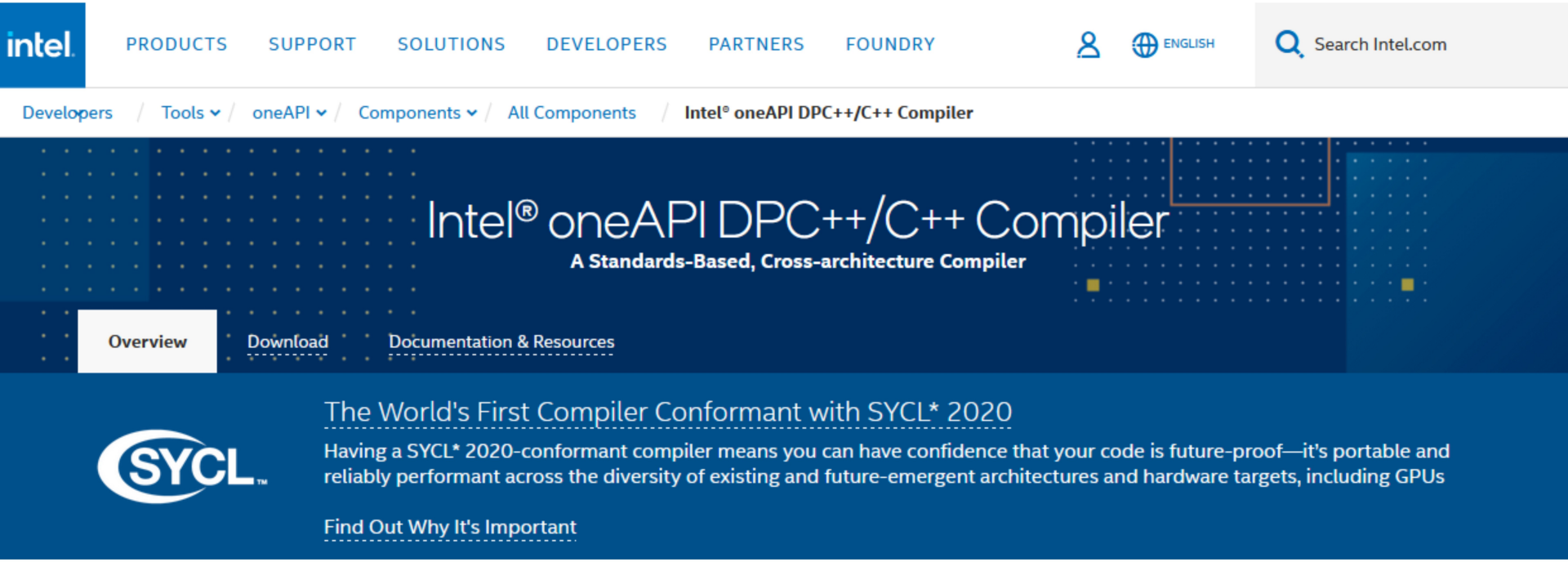
Oracle

- Java Compiler, Java Virtual Machine, …

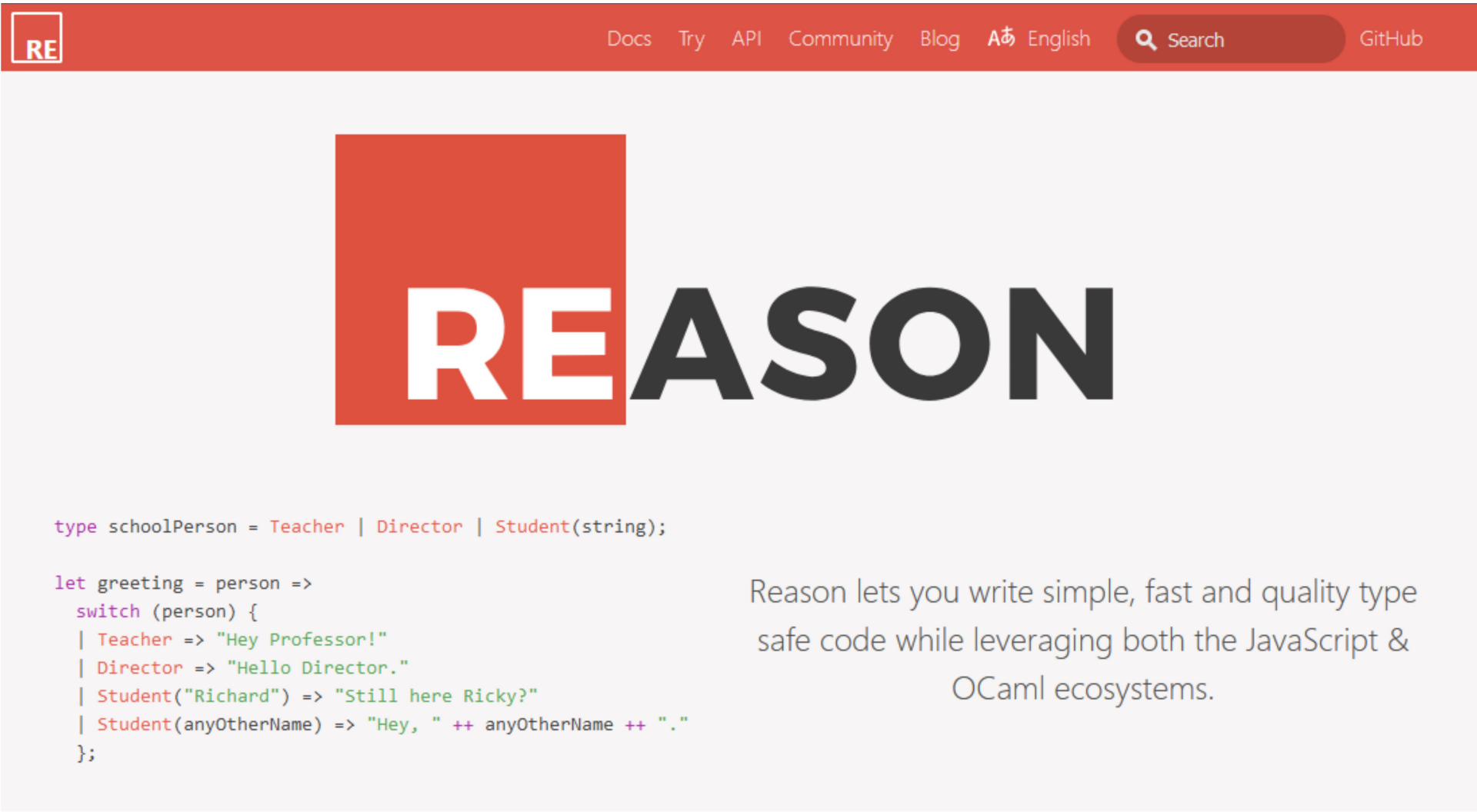


Intel

- Intel's C++ Compiler

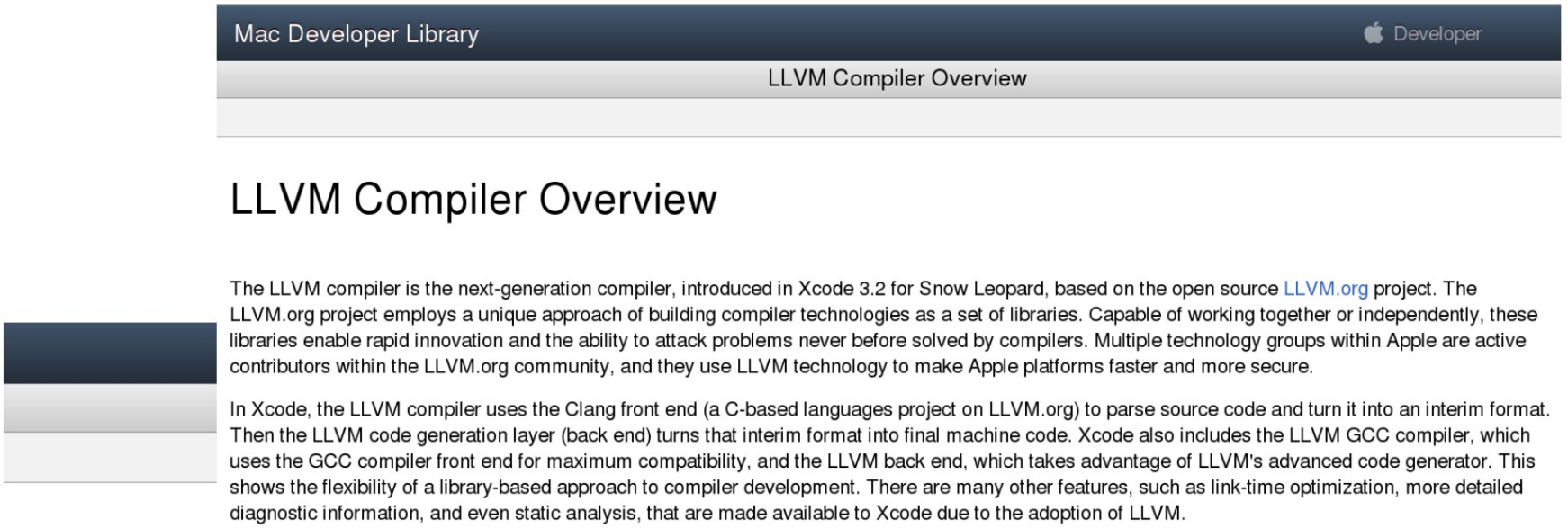


Facebook


RE
Docs Try API Community Blog Aあ English Search GitHub
REASON
type schoolPerson = Teacher | Director | Student(string);
let greeting = person =>
switch (person) {
| Teacher => "Hey Professor!"
| Director => "Hello Director."
| Student("Richard") => "Still here Ricky?"
| Student(anyOtherName) => "Hey, " ++ anyOtherName ++ "."
};
Reason lets you write simple, fast and quality type safe code while leveraging both the JavaScript & OCaml ecosystems.

Apple

- LLVM. Objective-C (iOS, etc.).



Mac Developer Library

Developer

LLVM Compiler Overview

LLVM Compiler Overview

The LLVM compiler is the next-generation compiler, introduced in Xcode 3.2 for Snow Leopard, based on the open source LLVM.org project. The LLVM.org project employs a unique approach of building compiler technologies as a set of libraries. Capable of working together or independently, these libraries enable rapid innovation and the ability to attack problems never before solved by compilers. Multiple technology groups within Apple are active contributors within the LLVM.org community, and they use LLVM technology to make Apple platforms faster and more secure.

In Xcode, the LLVM compiler uses the Clang front end (a C-based languages project on LLVM.org) to parse source code and turn it into an interim format. Then the LLVM code generation layer (back end) turns that interim format into final machine code. Xcode also includes the LLVM GCC compiler, which uses the GCC compiler front end for maximum compatibility, and the LLVM back end, which takes advantage of LLVM's advanced code generator. This shows the flexibility of a library-based approach to compiler development. There are many other features, such as link-time optimization, more detailed diagnostic information, and even static analysis, that are made available to Xcode due to the adoption of LLVM.

About Objective-C

Objective-C is the primary programming language you use when writing software for OS X and iOS. It's a superset of the C programming language and provides object-oriented capabilities and a dynamic runtime. Objective-C inherits the syntax, primitive types, and flow control statements of C and adds syntax for defining classes and methods. It also adds language-level support for object graph management and object literals while providing dynamic typing and binding, deferring many responsibilities until runtime.

Microsoft FlashFill

Our programming by example work (POPL 2011), also recognized as CACM Research Highlights (CACM 2012), ships as part of the Flash Fill feature in Excel in Office 2013. Here's a small video illustrating this feature. Here's another small video illustrating potential extensions.

Here's the inside story of how it came about: Flash Fill Gives Excel a Smart Charge

Here are some other videos on FlashFill

- You-tube: Excel 2013 Flash Fill: 23 Amazing Examples, Excel 2013- Flash Fill, Meet new Excel's Flash Fill, Dutch video, French Video, German video, Japanese video, Polish video, Romanian video, Urdu video, Musical
- Microsoft: Rick Rashid on FlashFill (in conversation with John Markoff of New York Times), Office Blog, Customer Preview Video (See the video segment from 0:35-0:40), Peter Lee on FlashFill (in his Keynote Speech on the 14th Computing in the 21st Century Conference – See the video segment from 22:22-27:20)
- CNet: Microsoft gives new Office a Windows 8 look (This video is at the bottom of the page. See the video segment from 2:00-3:01)

Here is what popular media says about this feature

- PC Magazine: My favorite new feature, because it saves a tremendous amount of time-wasting effort, is called Flash Fill, and it's one of many features where Excel acts as it it's using its brain, not just its raw number-crunching power. With some experimentation, you may find that Flash Fill is smarter than you expect.

People

Sumit Gulwani
Partner Research Manager

	A	B
1	SSN	Column2
2	542368978	542-36-8978
3	123145542	
4	121247543	
5	454545465	
6	642548745	
7	514852145	
8	152358834	
9	642974682	

Ctrl-E

	A	B
1	SSN	Column2
2	542368978	542-36-8978
3	123145542	123-14-5542
4	121247543	121-24-7543
5	454545465	454-54-5465
6	642548745	642-54-8745
7	514852145	514-85-2145
8	152358834	152-35-8834
9	642974682	642-97-4682

DARPA Cyber Grand Challenge

The ultimate test of wits in computer security occurs through open competition on the global Capture the Flag (CTF) tournament circuit. In CTF contests, experts reverse-engineer software, probe its weaknesses, search for deeply hidden flaws and create securely patched replacements.

What if a purpose-built computer systems could compete against the CTF circuit's greatest experts? DARPA has modeled the Cyber Grand Challenge on today's CTF tournaments to pave the way toward that future.

On August 4th, 2016, DARPA will hold the world's first all-computer Capture the Flag tournament in Las Vegas. Seven prototype systems will square off against each other, competing for nearly $4 million in prizes in a live network competition. The CGC Final Event will take place in conjunction with DEF CON, home of the longest-running annual CTF competition.

To learn more about the Cyber Grand Challenge, explore this site and visit DARPA's official CGC homepage, CGC Final Event announcement and CGC news articles.

DARPA Cyber Grand Challenge

The ultimate test of wits in computer security occurs through
Capture the Flag (CTF) tournament circuit. In CTF contests, ex
probe its weaknesses, search for deeply hidden flaws and crea

What if a purpose-built computer systems could compete agai
experts? DARPA has modeled the Cyber Grand Challenge on t
way toward that future.

On August 4th, 2016, DARPA will hold the world's first all-com
in Las Vegas. Seven prototype systems will square off against
million in prizes in a live network competition. The CGC Final
with DEF CON, home of the longest-running annual CTF comp

To learn more about the Cyber Grand Challenge, explore this s
homepage, CGC Final Event announcement and CGC news arti

Abstract

The automatic exploit generation challenge is given a program, automatically find vulnerabilities and generate exploits for them. In this paper we present AEG, the first end-to-end system for fully automatic exploit generation. We used AEG to analyze 14 open-source projects and successfully generated 16 control flow hijacking exploits. Two of the generated exploits (expect-5.43 and htget-0.93) are zero-day exploits against unknown vulnerabilities. Our contributions are: 1) we show how exploit generation for control flow hijack attacks can be modeled as a formal verification problem, 2) we propose preconditioned symbolic execution, a novel technique for targeting symbolic execution, 3) we present a general approach for generating working exploits once a bug is found, and 4) we build the first end-to-end system that automatically finds vulnerabilities and generates exploits that produce a shell.

DARPA Cyber Grand Challenge

AEG searches for bugs at the source code level by exploring execution paths. Specifically, AEG executes `iwconfig` using symbolic arguments

AEG performs dynamic analysis on the `iwconfig` binary using the concrete input generated in step 2.

AEG generates the constraints describing the exploit using the runtime information generated

Abstract

ploit generation challenge is given
cally find vulnerabilities and gener-
. In this paper we present AEG, the
m for fully automatic exploit gener-
to analyze 14 open-source projects
and successfully generated 16 control flow hijacking ex-
generated exploits (expect-5.43 and
-day exploits against unknown vul-
ontributions are: 1) we show how
r control flow hijack attacks can be
modeled as a formal verification problem, 2) we pro-
d symbolic execution, a novel tech-
ymbolic execution, 3) we present a
r generating working exploits once
4) we build the first end-to-end sys-
tem that automatically finds vulnerabilities and gener-
ates exploits that produce a shell.

Microsoft

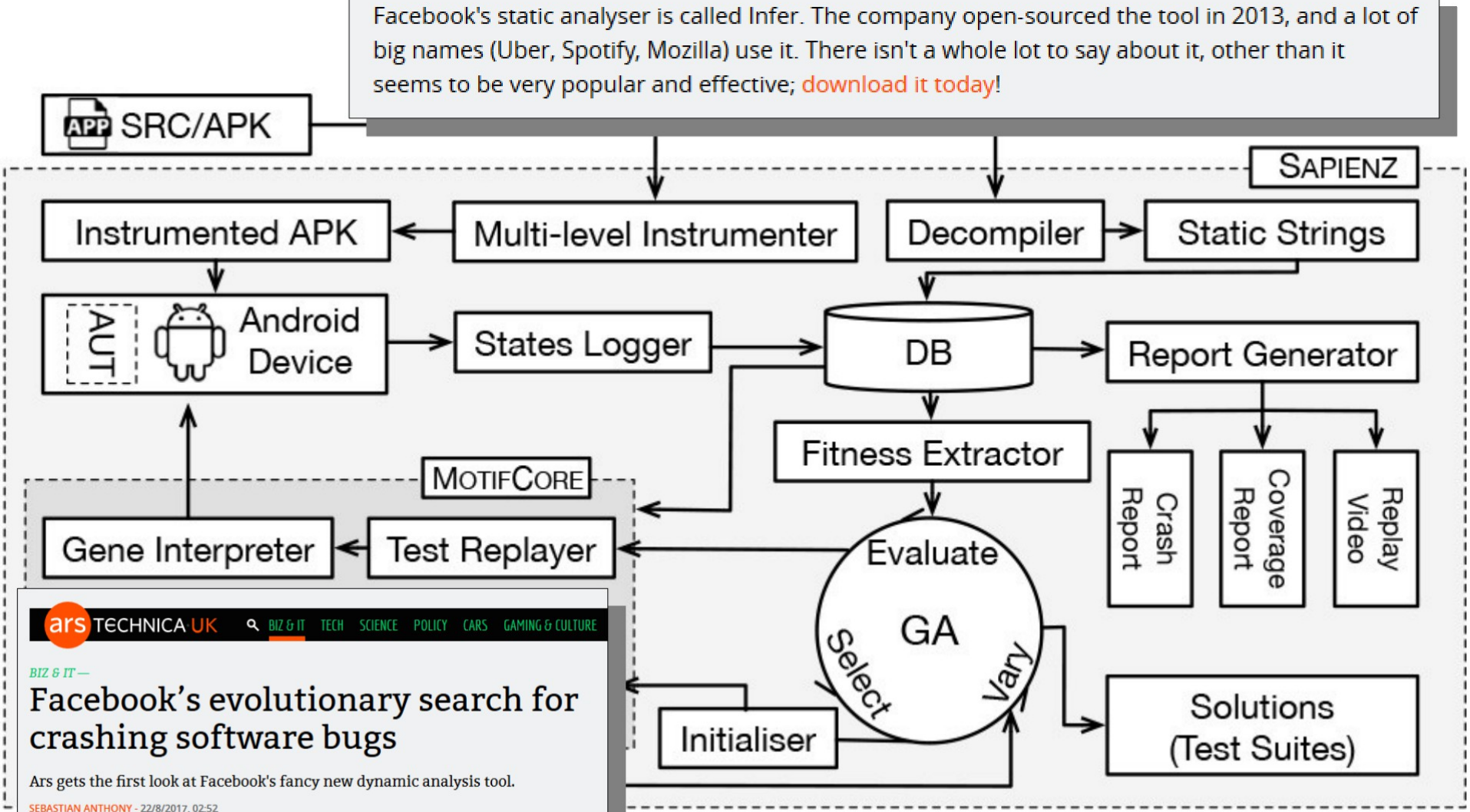
- (In addition to Visual Studio, MSVC++, etc.) Software, Languages, Analysis and Model Checking

SLAM is a project for checking that software satisfies critical behavioral properties of the interfaces it uses and to aid software engineers in designing interfaces and software that ensure reliable and correct functioning. Static Driver Verifier is a tool in the Windows Driver Development Kit that uses the SLAM verification engine.

"Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability."
Bill Gates, April 18, 2002. Keynote address at WinHec 2002

Facebook: Infer and Sapienz

Facebook's static analyser is called Infer. The company open-sourced the tool in 2013, and a lot of big names (Uber, Spotify, Mozilla) use it. There isn't a whole lot to say about it, other than it seems to be very popular and effective; download it today!



ars TECHNICA UK BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE

BIZ & IT —

Facebook's evolutionary search for crashing software bugs

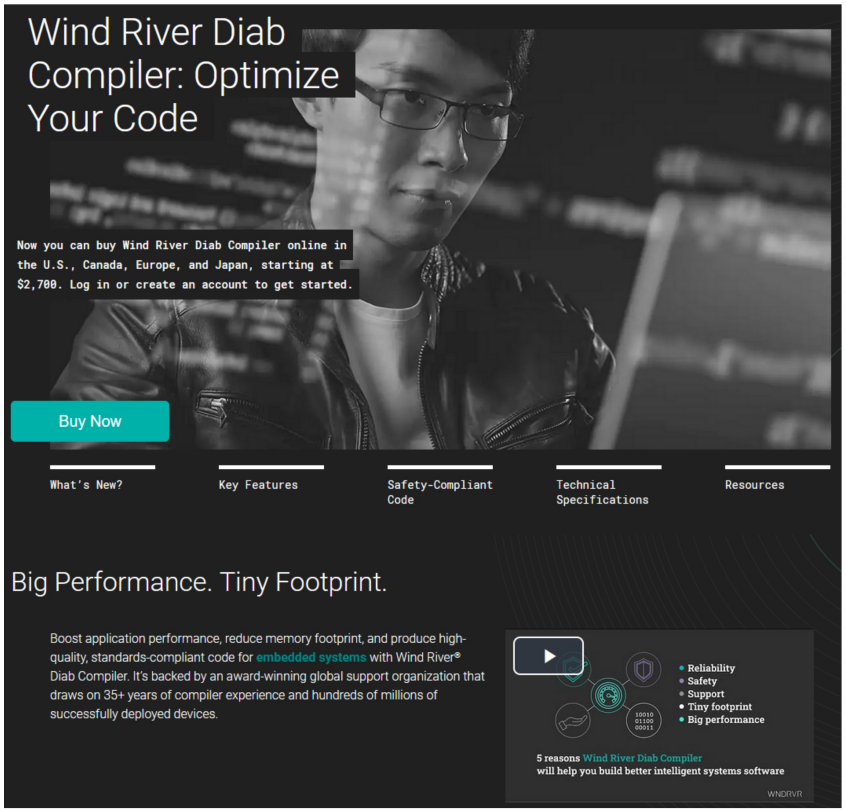
Ars gets the first look at Facebook's fancy new dynamic analysis tool.

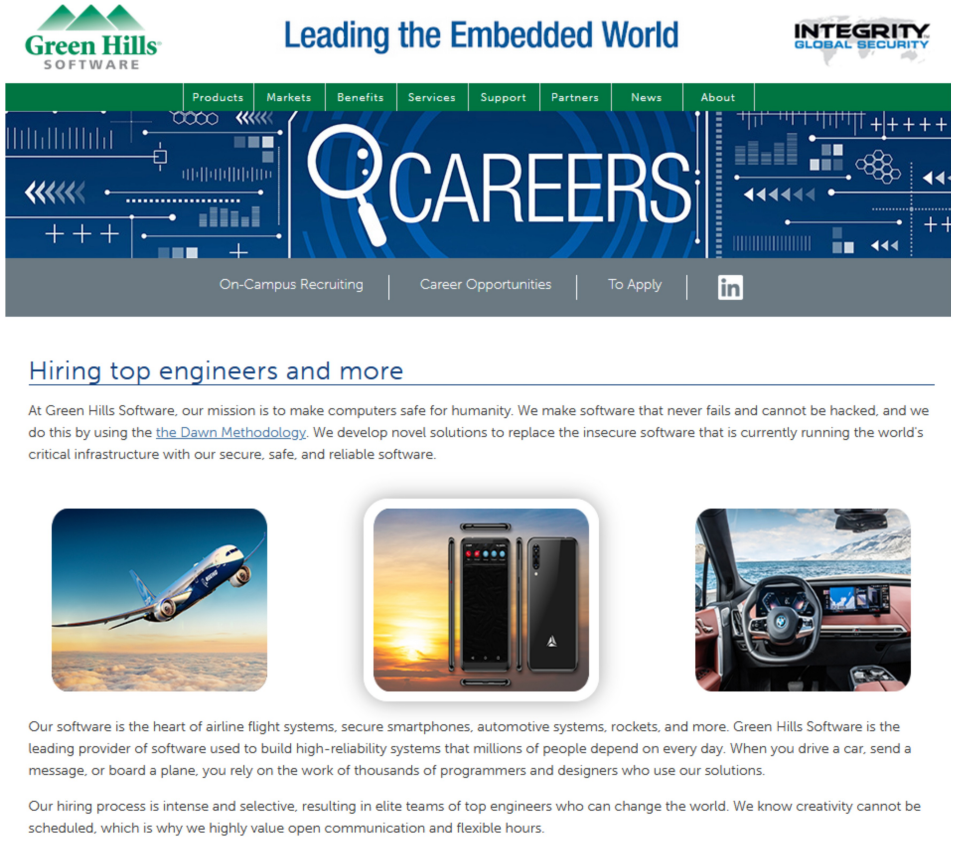
SEBASTIAN ANTHONY - 22/8/2017, 02:52

Figure 1: Sapienz workflow.

Wind River, Green Hills

- Embedded Systems





Founded 1981; 1,800+ employees; $400M+ revenue/year

Wait, what? Embedded?

- Curiosity Mars Rover, Cell Phones, Satellites, Engine Control Modules, Computed Radiology, Fighter Jets, Digital Cameras, Turbines, Anti-Lock Brakes, Switch Game Console, UAVs, ...

Eight years ago, NASA Jet Propulsion Laboratory (JPL) first began its work on the Mars Science Laboratory rover, Curiosity. Because of its long record of success with Wind River® on more than 20 JPL missions, NASA chose VxWorks® for the most technologically advanced autonomous robotic spacecraft and geologist set ever to be deployed by any space venture. Wind River VxWorks powered the craft's controls from the second the rocket left Earth on November 26, 2011, to its successful landing in the Gale Crater on Mars on August 5, 2012, and will support Curiosity's exploratory capability throughout the life of the mission.

Stay tuned for future updates as Wind River VxWorks continues to play a strategic role in NASA's groundbreaking mission to determine whether Mars is or has ever been capable of supporting life and to assess the planet's habitability for future human missions.

The Astrée Static Analyzer

- Astrée was able to prove, completely automatically, the absence of any RTE in a C version of the automatic docking software of the Jules Vernes Automated Transfer Vehicle (ATV) enabling ESA to transport payloads to the International Space Station.

Adobe

- Photoshop contains interpreters

2 Photoshop Scripting Basics

This chapter provides an overview of scripting for Photoshop, describes scripting support for the scripting languages AppleScript, VBScript, and JavaScript, how to execute scripts, and covers the Photoshop object model. It provides a simple example of how to write your first Photoshop script.

If you are familiar with scripting or programming languages, you most likely will want to skip much of this chapter. Use the following list to locate information that is most relevant to you.

- For more information on the Photoshop object model, see "Photoshop Object Model" on page 11.
- For information on selecting a scripting language, refer to the *Introduction to Scripting* guide.
- For examples of scripts created specifically for use with Photoshop, see Chapter 3, "Scripting Photoshop" on page 21.
- For detailed information on Photoshop objects and commands, please use the reference information in the three reference manuals provided with this installation: *Adobe Photoshop CC 2015 AppleScript Scripting Reference*, *Adobe Photoshop CC 2015 Visual Basic Scripting Reference*, and *Adobe Photoshop CC 2015 JavaScript Scripting Reference*.

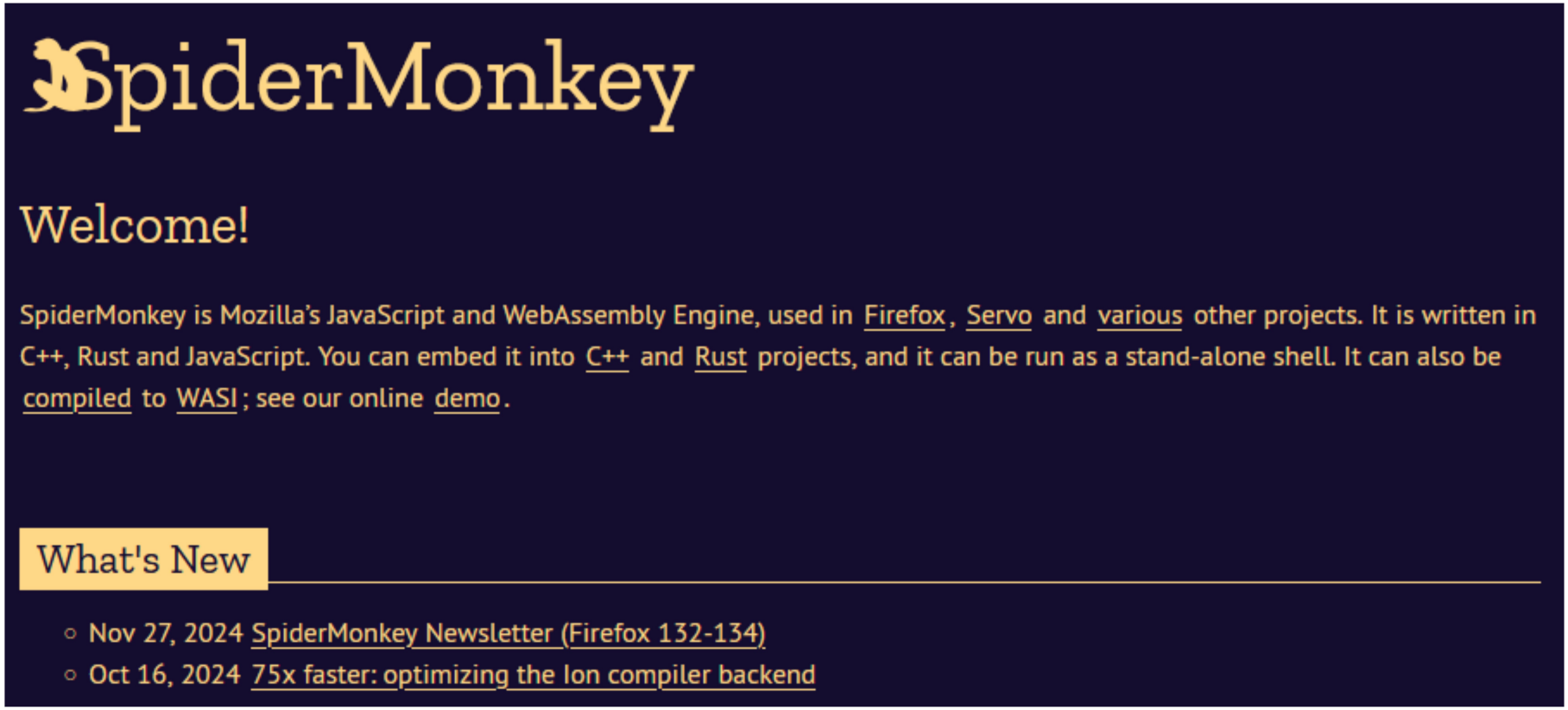
NOTE: You can also view information about the Photoshop objects and commands through the object browsers for each of the three scripting languages. See "Viewing Photoshop Objects, Commands, and Methods" on page 21.

Scripting Overview

A script is a series of commands that tells Photoshop to perform a set of specified actions, such as applying different filters to selections in an open document. These actions can be simple and affect only a single object, or they can be complex and affect many objects in a Photoshop document. The actions can call Photoshop alone or invoke other applications.

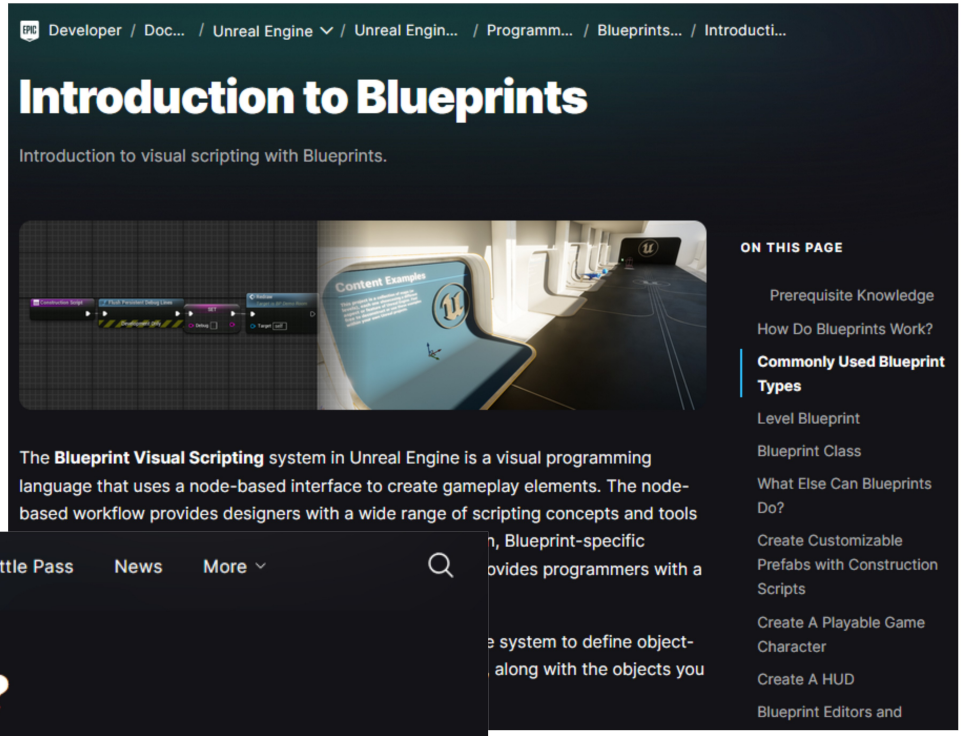
Mozilla

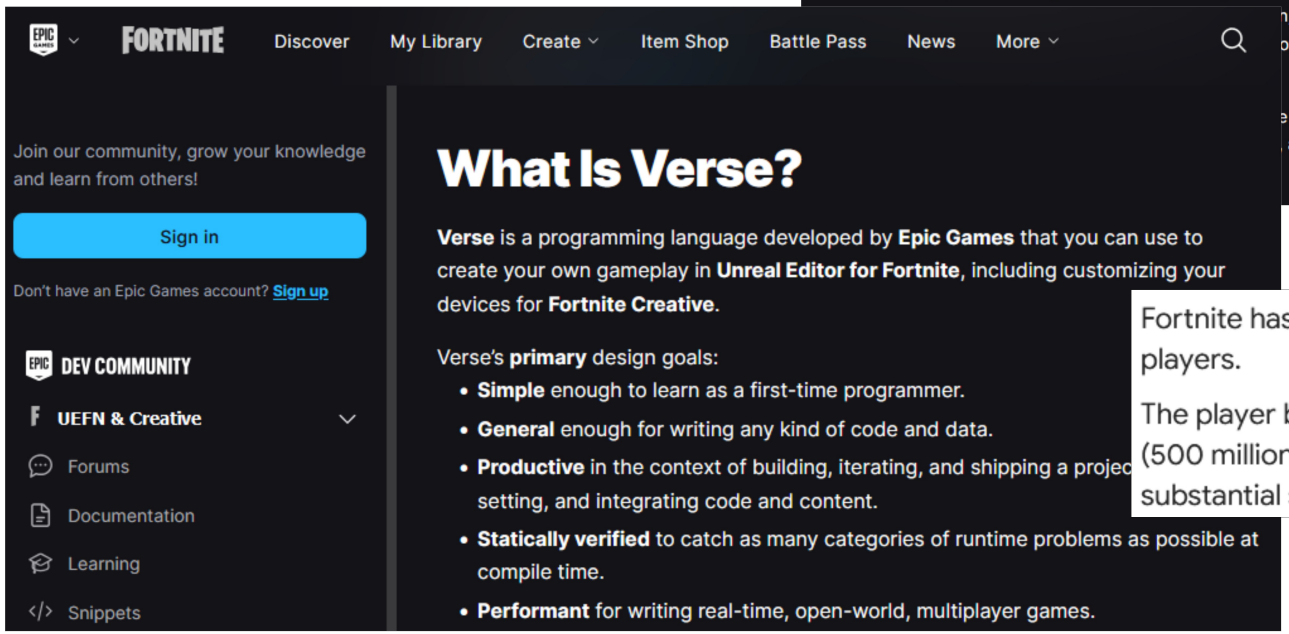
- SpiderMonkey JavaScript / WebAssembly



Epic Games

- Blueprints (Unreal)
- Verse (Fortnite)





Fortnite has over 650 million registered players.

The player base grew by 150 million from 2022 (500 million) to 2023 (650 million), showing a substantial surge in engagement. From 2017 to

Surprise: PDF Files

PostScript (**PS**) is a page description language and dynamically typed, stack-based programming language. It is most commonly used in the electronic publishing and desktop publishing realm, but as a Turing complete programming language, it can be used for many other purposes as well. PostScript was created at Adobe Systems by John Warnock, Charles Geschke, Doug Brotz, Ed Taft and Bill Paxton from 1982 to 1984. The most recent version, PostScript 3, was released in 1997.

History [edit]

The concepts of the PostScript language were seeded in 1976 by John Gaffney at Evans
John V

The development of PDF began in 1991 when John Warnock wrote a paper for a project then code-named Camelot, in which he proposed the creation of a simplified version of PostScript called Interchange PostScript (IPS).[6] Unlike traditional PostScript, which was tightly focused on rendering print jobs to output devices, IPS would be optimized for displaying pages to any screen and any platform.[6]

PostScript

PostScript 3 logo

Paradigm	Multi-paradigm: concatenative (stack-based), procedural
Designed by	John Warnock, Chuck Geschke, Doug Brotz, Ed Taft, Bill Paxton
er	Adobe Systems

PostScript language [edit]

PostScript is a page description language run in an interpreter to generate an image.[6] It can handle graphics and has standard features of programming languages such as branching and looping.[6] PDF is a subset of PostScript, simplified to remove such

Ubiquity

- Your cellphones, browsers, games, PDF files, spreadsheets, etc., all contain interpreters for (or were built using compilers for) **specialized programming languages**
- These may be in places you would not expect, using languages you may not know, from companies you may not have heard of … but PL is a big business.

Wait …

- But weren't most of those examples mixtures of PL and some other discipline?
  - Mars Rover, Intel = PL + Hardware
  - FlashFill = PL + Machine Learning
  - Cyber Grand Challenge = PL + Security
  - Gaming Languages, PDF = PL + Graphics
  - Codex, Sapienz = PL + AI
  - SLAM = PL + Model Checking
- Yes! That's the point!

Parts of Computer Science

- CS = (Math $\times$ Logic) + Engineering
  - Science (from Latin *scientia* - knowledge) refers to a system of acquiring knowledge - based on empiricism, experimentation, and methodological naturalism - aimed at finding out the truth.
- We rarely actually do this in CS
  - "CS theory" = Math (logic)
  - "Systems" = Engineering (bridge building)

Programming Languages

- Best of both worlds: Theory and Practice!
  - Only pure CS theory is more primal
- Touches most other CS areas
  - Theory: DFAs, PDAs, TMs, language theory (e.g., LALR)
  - Systems: system calls, assembler, memory management
  - Arch: compiler targets, optimizations, stack frames
  - Numerics: FORTRAN, IEEE FP, Matlab, loop nest optim.
  - AI: theorem proving, machine learning, GenAI coding
  - DB: SQL, persistent objects, modern linkers
  - Networking: packet filters, protocols, even Ruby on Rails
  - Graphics: OpenGL, LaTeX, PostScript, even Logo (= LISP)
  - Security: buffer overruns, .net, bytecode, PCC, …
  - Software Engineering: bug finding, refactoring, types, ...

Overarching Theme

- I assert (and shall argue) that
- PL is one of the more vibrant and active areas of CS research today
  - It has theoretical and practical meatiness
  - It intersects most other CS areas
- This course teaches you how to interpret and conduct PL research in your own projects

Goal #1

- Learn to **use** advanced PL techniques

Useful Complex Knowledge

- A proof of the fundamental theorem of calculus
- A proof of the max-flow min-cut theorem
- Nifty Tree node insertion (e.g., B-Trees, AVL, Red-Black)
- The code for the Fast Fourier Transform
- And so on …

No Useless Memorization

- I will not waste your time with “useless” memorization
- This course will cover complex subjects
- I will teach their details to help you understand them the first time
- But you will not have to memorize anything low-level
- Rather, learn to apply broad concepts

Goal #2

- When you design a language, it will avoid the mistakes of the past and you'll be able to describe it formally

Story 1: JavaScript Objects?

- JavaScript is object-oriented, but originally (1995) it used Prototypes
  - "The most controversial feature of the language is the way it does inheritance, which is radically different than virtually all other modern languages. Most languages use classes – I call them 'classical languages' – JavaScript does not. JavaScript is class free. It uses prototypes. For people who are classically trained who look at the language, they go: well, this is deficient. You don't have classes, how can you get anything done? How can you have any confidence that the structure of your program's going to work? And they never get past that."
    - Douglas Crockford, inventor of JavaScript Object Notation
- It took 20 years to add Classes (2015)

Story 2: Java Saves Space?

- Java bytecode programs contain subroutines (jsr) that run in the caller's stack frame *(why?)*
- jsr complicates the formal semantics of bytecodes
  - Several verifier bugs were in code implementing jsr
  - 30% of typing rules, 50% of soundness proof due to jsr
- It is not worth it:
  - In 650K lines of Java code, 230 subroutines, saving 2427 bytes, or 0.02%
  - 13 times more space could be saved by renaming the language back to Oak
    - [In 1994], the language was renamed "**Java**" after a trademark search revealed that the name "Oak" was used by a manufacturer of video adapter cards.

Story 3: C++ Inheritance?

- C++ supports multiple inheritance (1983)
  - "I think my most obvious mistake was not to introduce templates before multiple inheritance"
  - "One problem with introducing MI before templates was that it encouraged further overuse of class hierarchies"
  - "To get efficiency and type safety for containers, you need templates (and I didn't have an implementation supporting templates until 1988 or 1989). However, templates are not enough, you also need a design for containers and uses of containers that can deliver that safety. We didn't have such an architecture until Alex Stepanov came along with the STL."
    - Bjarne Stroustrup, Designer of C++
- Stepanov introduced the Standard Template Library in 1993

Recall Goal #2

- When (not if) you design a language, it will avoid the mistakes of the past and you'll be able to describe it formally

Goal #3

- Understand current PL research (PLDI, POPL, OOPSLA, TOPLAS, …) and technology transfer (MS, Intel, …)

Final Goal: Fun

Changeups and Trivia

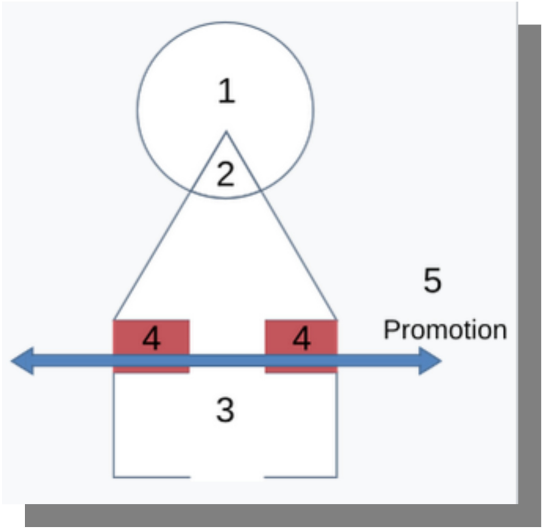
- "[Professors who] deliberately and consistently interspersed their lectures with … some other form of deliberate break … usually commanded a better attention span from the class, and these deliberate variations had the effect of postponing or even eliminating the occurrence of an attention break"

[Johnstone and Percival. *Attention breaks in lectures.* Education in Chemistry, 13. 49-50, 1976.]

[Middendorf and Kalish. *The "Change-up" in Lectures.* TRC Newsletter, 8:1 (Fall 1996).]

Q: Popular Media

- This Korean dystopian survival thriller involves hundreds of contestants playing fatal children's games for a chance at a large payout. The name is based on 오징어 (ojingŏ), a game in which the playing field is said to resemble an animal.



Q: Books (730 / 842)

- This 1960 Daniel Keyes sci-fi novel is told as a *"progris riport"* from the point-of-view of Charlie Gordon as he takes an experimental intelligence-enhancing treatment. The treatment is temporary. The book won the Hugo and Nebula awards.

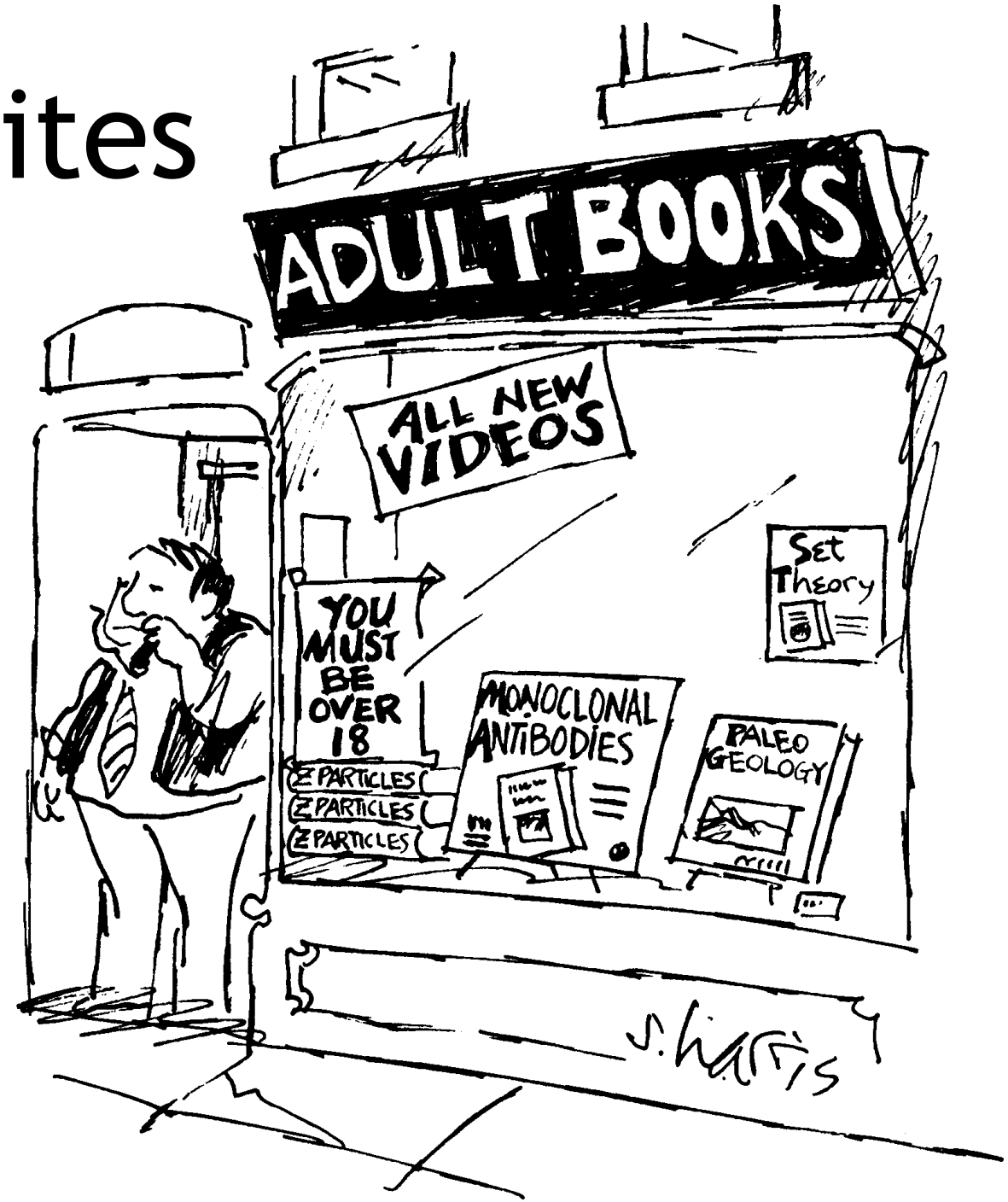
Q: Computer Science

- This Sri Lanka-born, British computer scientist is best known for his development of *QuickSort*, a logic for verifying program correctness, the *monitor* approach to mutual exclusion, and the formalism of *Communicating Sequential Processes*. In 2009 he apologized for inventing the *null reference*:
  - I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

How Hard Is This Class?

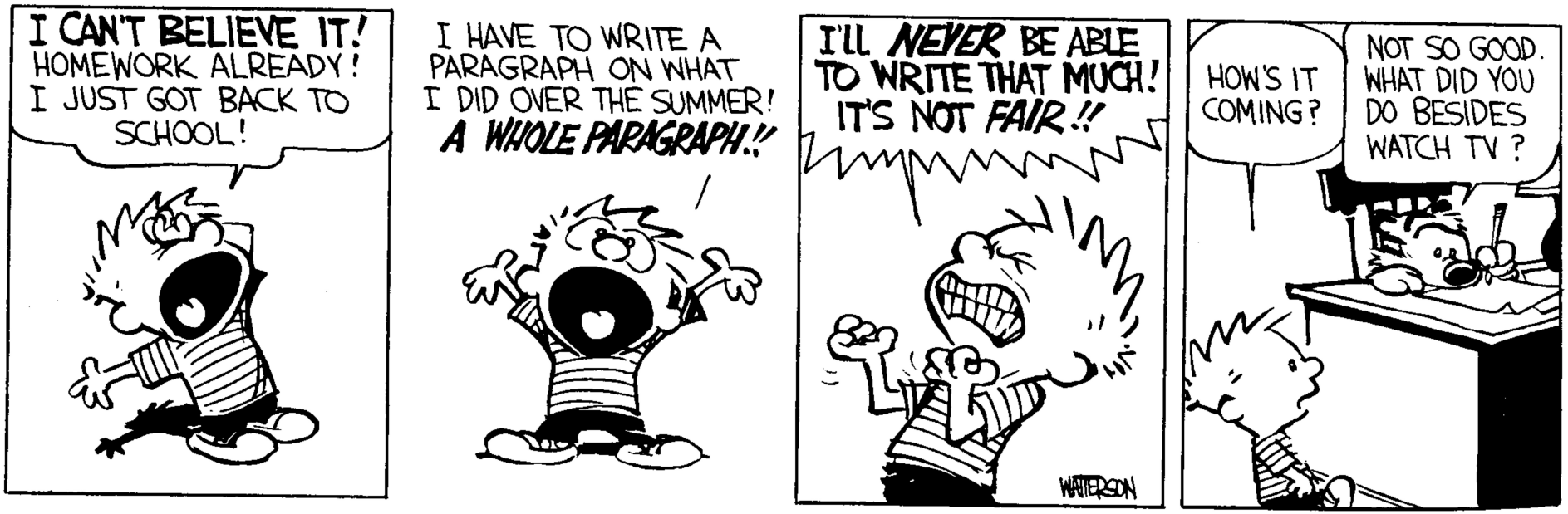
Prerequisites

- Undergraduate PL/compilers course?
  - **No**
- "Mathematical maturity"



Assignments

- Homework Assignments (6+1)
- Peer Review Activities (6)
- Daily Reading (1-2 papers per class)
- Quizzes and Participation
- ~~Final Project~~



Homework Problem Sets

- Some material can be “mathy”
- Much like Calculus, practice is handy
- Short: ~3 theory + 1 coding per HW
- You have one week to do each one
  - Available in advance ...
- Long: analysis of real C programs
- We will review and comments on your English prose.

Peer Review Motivation

- A key outcome of this class is being able to interpret PL research and papers
  - You may have to read through a few PL papers to decide which one to cite for a related work section, for example
  - You may have to convince your manager to use tool X instead of tool Y based on claims
- You *write* your own homeworks, but we also want more practice *reading*

Anonymous Peer Review

- For the formal written homeworks
  - 1st page: your name, email, etc.
  - Other pages: no name, no email, just text
- We will take the *anonymous* parts of submissions and shuffle them around
- On some days (see schedule) we will go over the "answer key" together
- You will then write up feedback via Gradescope within 48 hours

Peer Review

- I believe in feedback for X **before** X+1 is due (both peer review and grades)
- Assignments are all due Day X at midnight
  - Sometimes a Friday!
- Peer review is 1 or 2 lectures later
- Implication → we can't accept late homework once we've gone over the answer key

Reading Quizzes

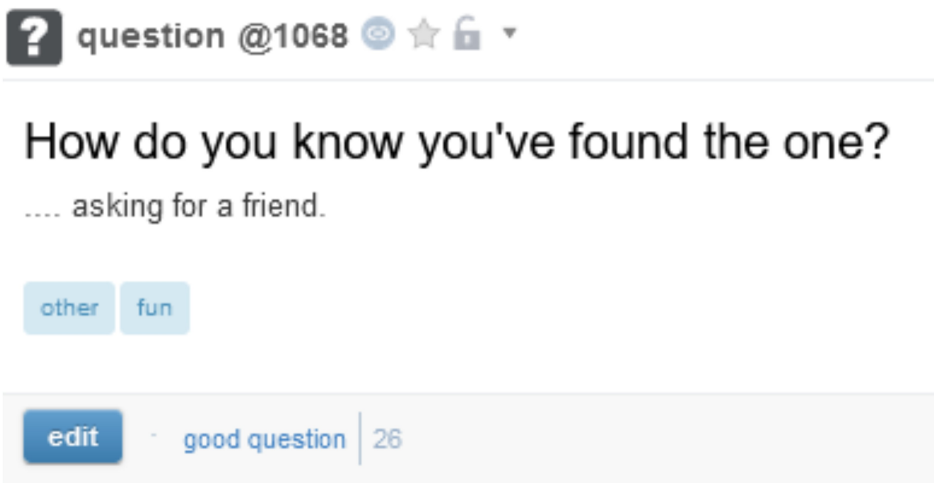
- A key problem:
  - If I never check, graduate students will not do the reading.
- A related desire:
  - Graduate students often wish that someone would make them do the reading.
- Implementation:
  - Very short answer or multiple choice quizzes
  - Mostly to keep you on track
  - (Who benefits if you Ctrl-F instead of reading?)

Participation

- It is easier for engaged students to retain the formal material in this class
  - Some classes: just watch recordings at 1.5x
  - This may not be one of those classes
  - You'll want to be able to ask questions, etc.
- We will record attendance in class
  - Typically via writing your UM email on a notecard

Piazza Forum

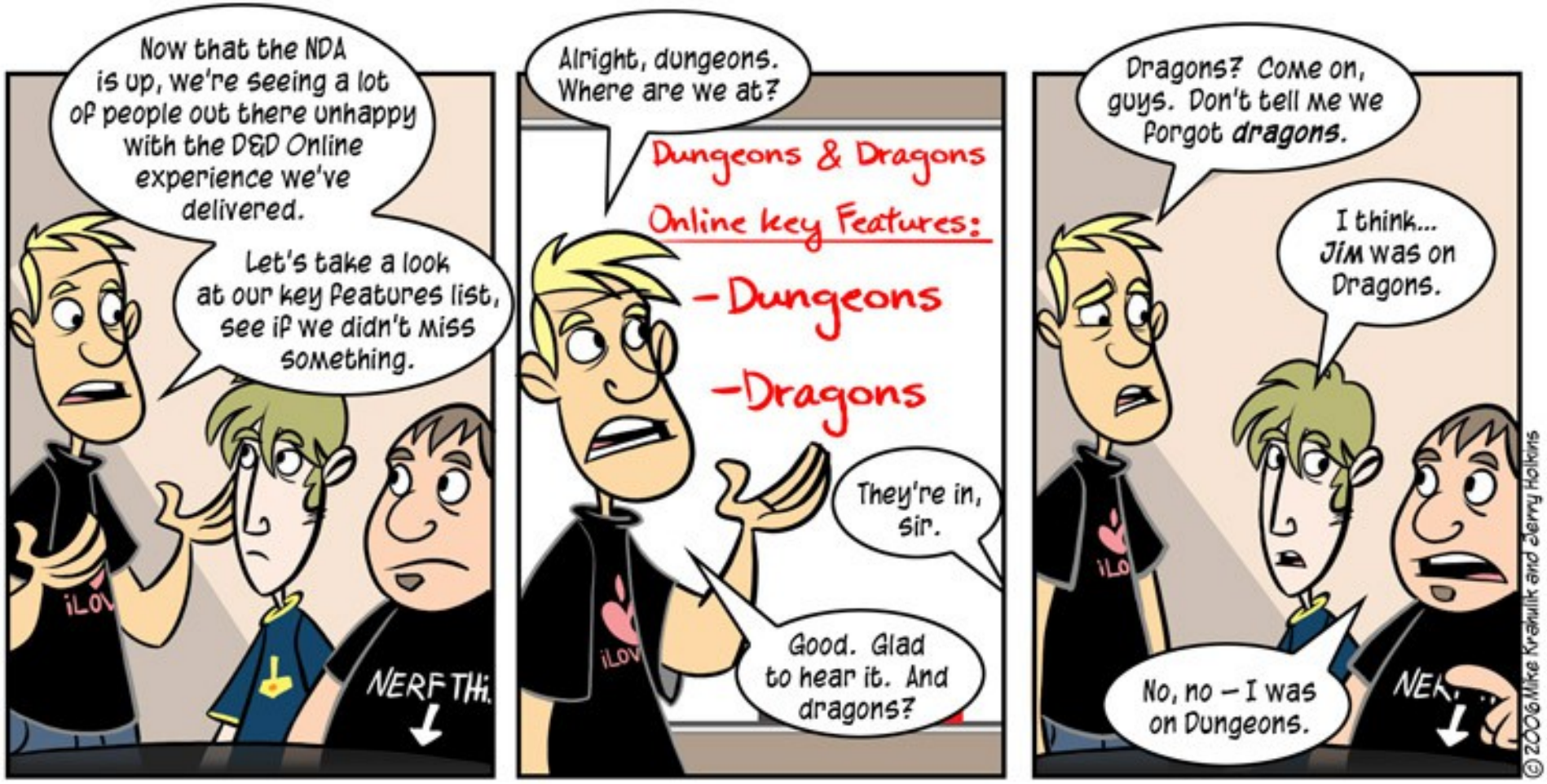
- Ask questions about assignments
- Compare concerns about papers
- Ask for more details about references from class
- Previous years: fun papers, memes, moral quandaries, etc.



Who Benefits?

- This research-focused class is most useful to students who will be reading and writing formal PL research papers
  - Ph.D. students in PL
  - Master's degree students pursuing PL work
- If you signed up because the name made it look like a Programming course or because you needed credits, email me and I'll help you find a better-fitting elective based on your interests

Key Features of PL



Programs and Languages

- Programs
  - What are they trying to do?
  - Are they doing it?
  - Are they making some other mistake?
  - Were they hard to write?
  - Could we make it easier?
  - Should you run them?
  - How should you run them?
  - How can I run them faster?

Programs and Languages

- Languages
  - Why are they annoying?
  - How could we make them better?
  - What tasks can they make easier?
  - What cool features might we add?
  - Can we stop mistakes before they happen?
  - Do we need new paradigms?
  - How can we help out My Favorite Domain?

Common PL Research Tasks

- Design a new language feature
- Design a new type system / checker
- Design a new program analysis
- Find bugs in programs
- (Help people to) Fix bugs in programs
- Transform programs (source or assembly)
- Interpret and execute programs
- Prove things about programs
- Optimize programs

Grand Unified Theory

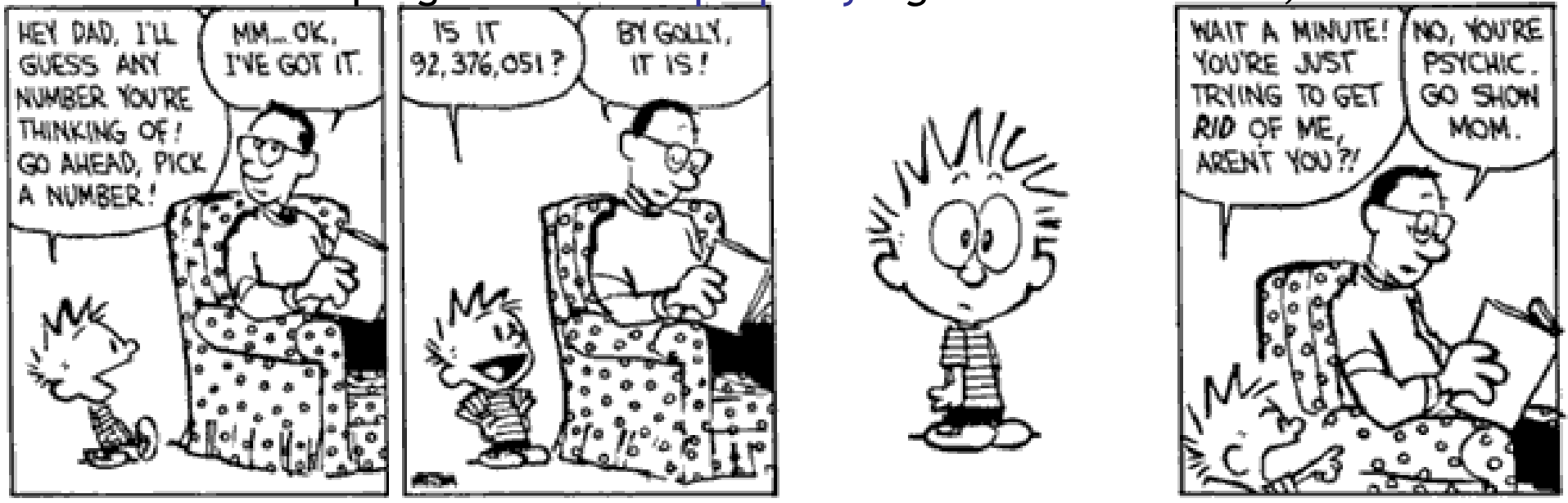
- Design a new type system
- Your type-checker becomes a bug-finder
  - No type errors $\Rightarrow$ proof that program is safe
  - Type error $\Rightarrow$ bug may exist in program
    - Fault localization and automated program repair
- Design a new language feature
  - To prevent the sort of mistakes you found
- Write a source-to-source transform
  - Your new feature now works on existing code

EECS 590 - Core Topics

- Model checking
- Operational semantics
- Neurosymbolic approaches
- Provers and proofs
- Verification conditions
- Type theory
- Symbolic execution
- Abstract interpretation
- Invariant detection
- Lambda calculus

First Topic: Model Checking

- Verify critical properties of software or find bugs
- Take an important program (e.g., a device driver)
- Merge it with a property (e.g., no deadlocks, asynchronous IRP handling, BSD sockets, database transactions, …)
- Transform the result into a *boolean program*
  - Same control flow, but only boolean variables
- Use a model checker to explore the resulting *state space*
  - Result 1: program provably satisfies property
  - Result 2: program violates property *right here on line 92,376!*



Example Program

```
Example ( ) {
    do{
        lock();
        old = new;
        q = q->next;
        if (q != NULL){
            q->data = new;
            unlock();
            new ++;
        }
    } while(new != old);
    unlock();
    return;
}
```

Is this program correct?

Example Program

```
Example ( ) {
    do{
        lock();
        old = new;
        q = q->next;
        if (q != NULL){
            q->data = new;
            unlock();
            new ++;
        }
    } while(new != old);
    unlock();
    return;
}
```

Is this program correct?

What does correct mean?

Doing no evil?

Doing some good?

How do we determine if a program is correct?

Verification by Model Checking

```
Example ( ) {
1: do{
       lock();
       old = new;
       q = q->next;
2:     if (q != NULL){
3:        q->data = new;
          unlock();
          new ++;
       }
4: } while(new != old);
5:  unlock();
    return;
}
```

1. (Finite State) Program
2. State Transition Graph
3. Reachability

- Pgm → Finite state model
- State explosion

\+ State Exploration

\+ Counterexamples

Precise [SPIN, SMV, Bandera, JPF]

For Our Next Exciting Episode

- See webpage under “Lectures”
- Read the two articles
- Peruse the HW page and details