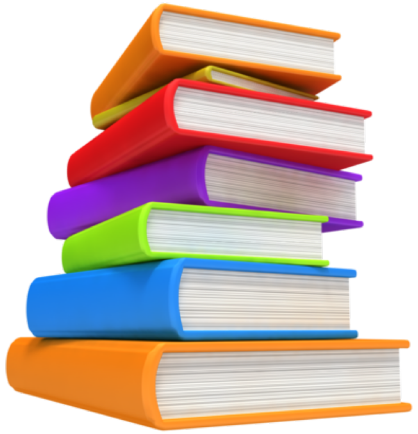


My teaching approach

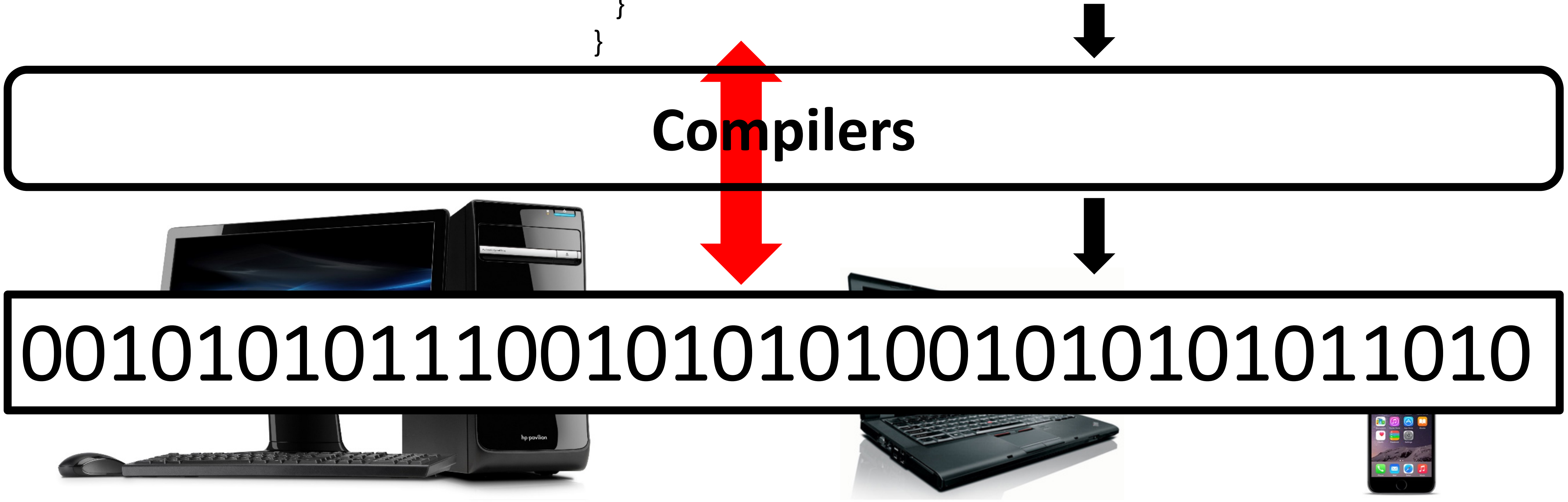
Simone Campanoni
simonec@eecs.northwestern.edu

Solutions that enable my teaching philosophy

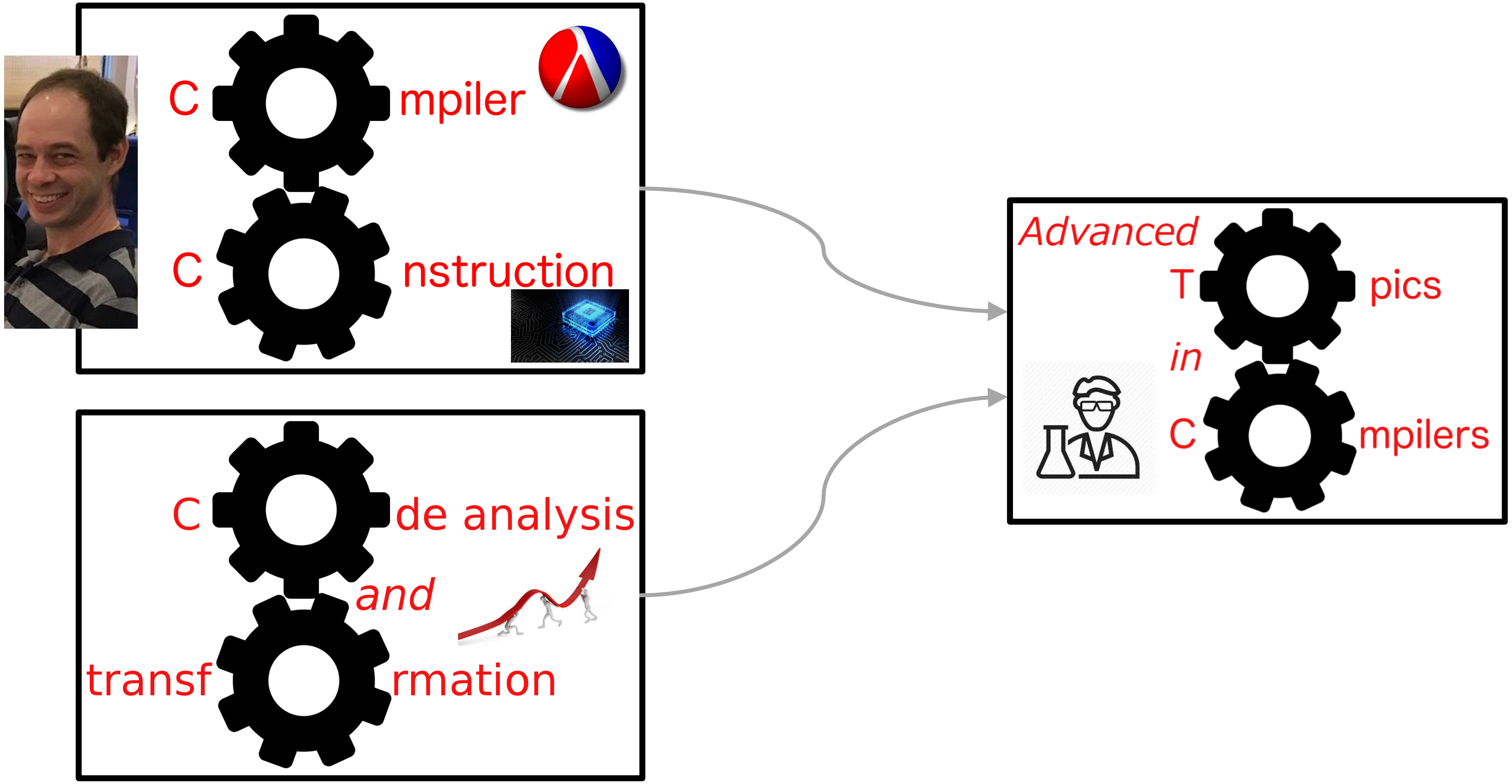
- are probably known by experienced teachers
- might not be directly transferrable to other classes
- work well (for me at least) for classes with 30-50 students
- worked because students are truly impressive
- are a great fit for my personality

But first, a gentle compiler introduction

```
while (somethingToDo){
  if (somethingToCode){
    code(myState);
  } else {
    read(papers);
  }
}
```

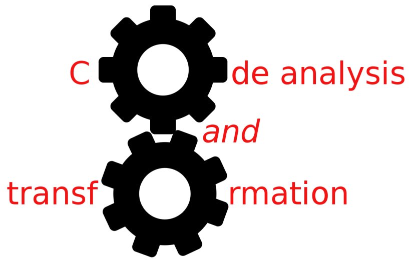


The compiler classes at Northwestern



A rough start

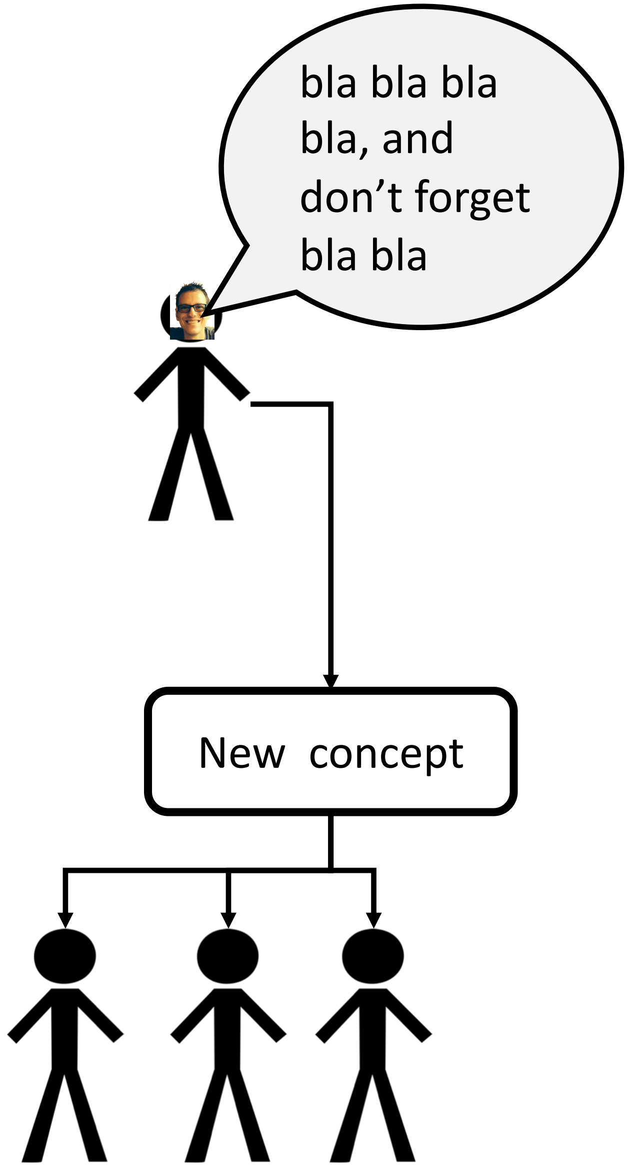
- End of my post-doc: July 2015
- Fall 2015: beginning of my first class designed from scratch



- Prior experience: ok, but not great
- My goal: show students why compilers are fascinating
  - Challenge: compilers isn’t a hot topic like Machine Learning
- Solution: transfer my passion

It’s all about passion

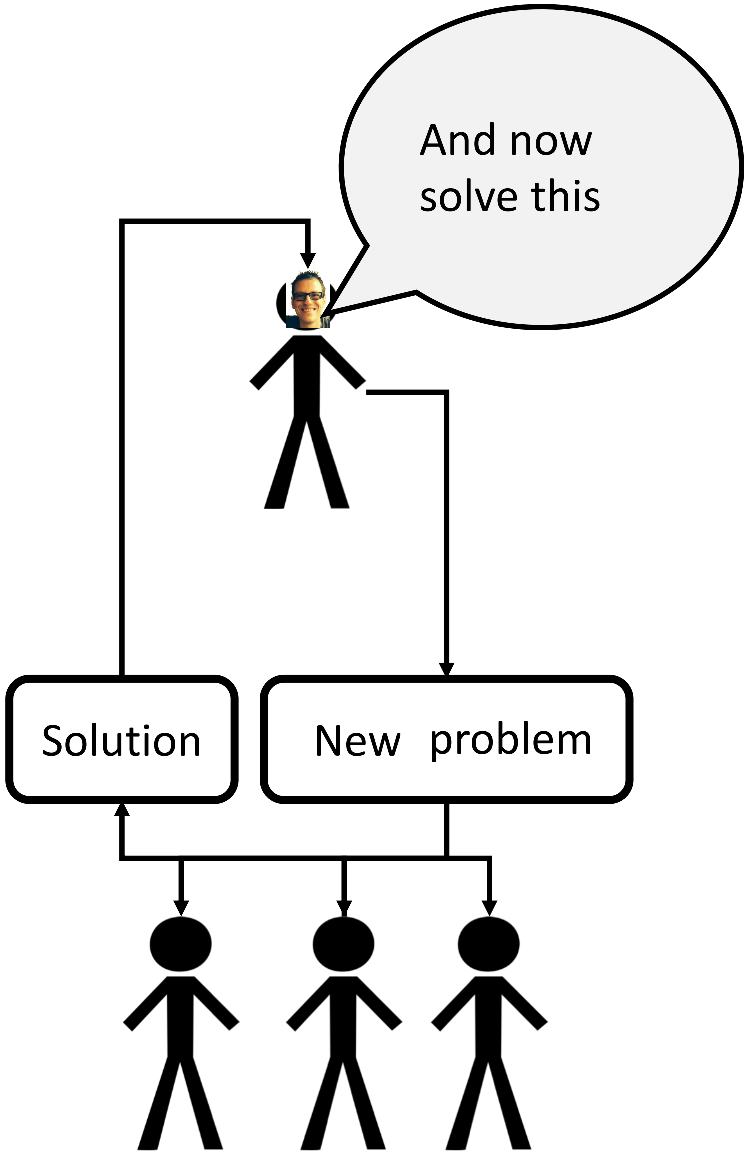
- My first passion (before compilers) is about learning
- It is because of the freedom you gain by the new knowledge
  - If you know more, you can do more
- **My 1st teaching goal:**
  make students feel this freedom (hopefully) like I do
  - Students need
    - to learn new concepts
    - to use new concepts to solve new problems



My teaching goals

- **My 1st teaching goal:**
  make students feel this freedom (hopefully) like I do
  - Students need
    - to learn new concepts
    - to use new concepts to solve new problems
- **My 2nd teaching goal:** show why compilers are fascinating
  - Optimization
  - Abstraction
  - Scale

Elegant, low-cost optimizations

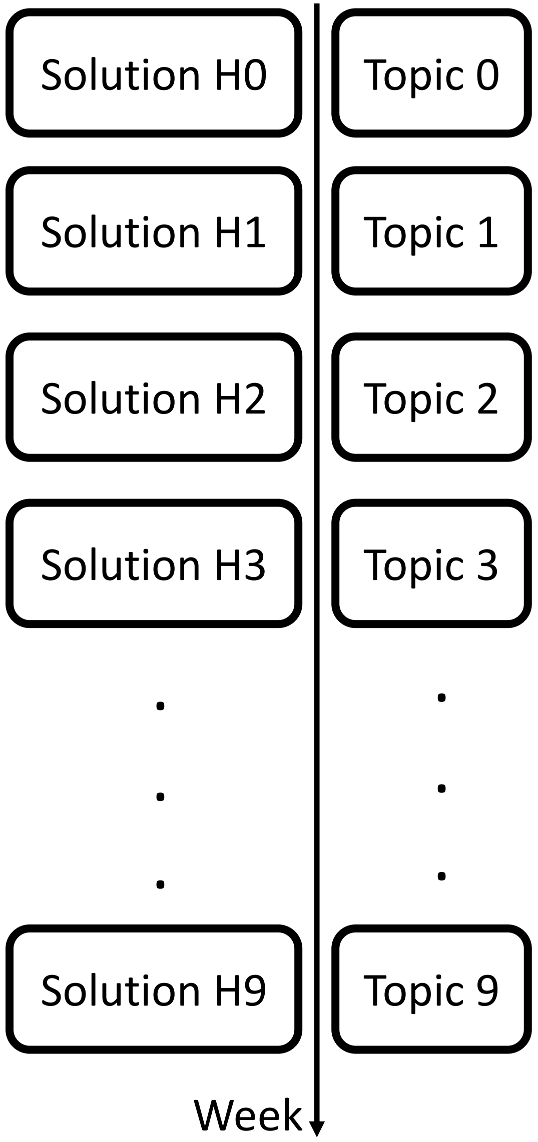


My teaching goals

- **My 1st teaching goal:**
  make students feel this freedom (hopefully) like I do
  - Students need
    - to learn new concepts
    - to use new concepts to solve new problems
- **My 2nd teaching goal:** show why compilers are fascinating
  - Optimization
  - Abstraction
  - Scale

Elegant, low-cost optimizations

- **My 3rd teaching goal:** a bad software/solution is expensive

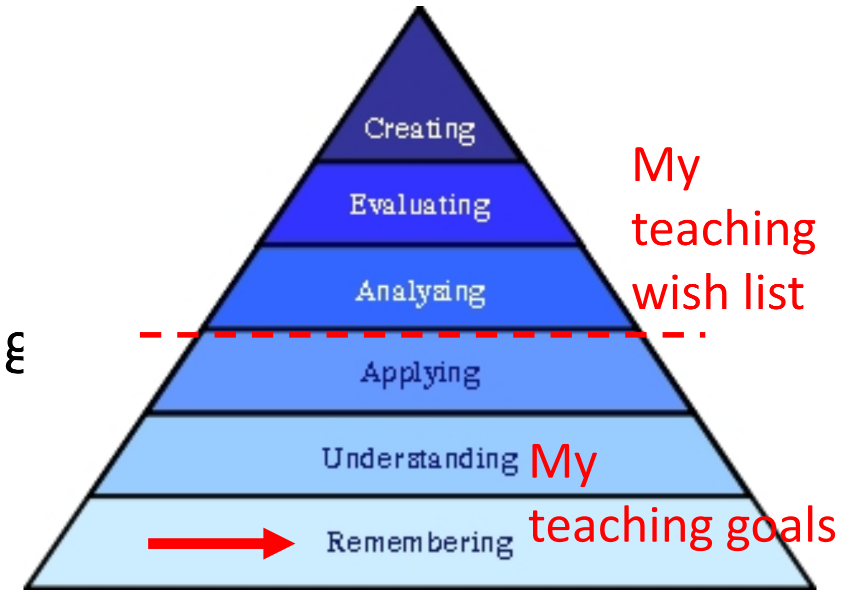


My teaching goals

- **My 1st teaching goal:**
  make students feel this freedom (hopefully) like I do

- **My 2nd teaching goal:** show why compilers are fascinating

  Elegant, low-cost optimizations



- **My 3rd teaching goal:** a bad software/solution is expensive

Challenges

- My classes: one topic per week
- Every week, students have to
    1. Learn new concepts and related abstractions
    2. Understand new problems
    3. Decide which existing abstractions to reuse
       and whether new ones are needed
    4. Create new abstractions or tune existing ones
    5. Create a solution that actually works in real systems
    6. Learn how to test their solutions
- Very challenging and it requires
  a significant commitment of time and mental effort
    - Despite my efforts to simplify my classes, they are still challenging
    - I was scared (and still am) nobody would take my classes

- Challenge 1: knowing how to improve
- Challenge 2: learning new concepts quickly
- Challenge 3: motivating students
  to keep pushing theirself

Challenge 1: knowing how to improve

I ask feedbacks/criticisms during the last day of my classes

- Todos.txt keeps growing during that day
- Goal: everything in todos.txt will be done before the subsequent year

Challenge 2: learning new concepts quickly

Typical teaching flow:

1. Description of new concept
2. Description of possible implementations of this new concept
3. Description of how this concept
   is implemented in systems used in production
4. Description of when/how this concept implementation is used in practice

The enabling part is at the end

Too late:

- Exciting part is at the end
- Motivation of learning a new concept is at the end
- Students have lost some attention going through the first part

Challenge 2: learning new concepts quickly

My teaching flow

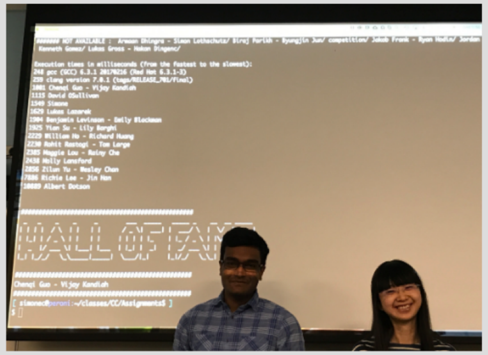
1. Demo: you want X (e.g., face detection in real time), you don't get it
2. Dig deep on why we didn't get it
3. Problem: this is the issue, we need to solve it to get X
4. Solution: we need information Y and Z to solve the issue
5. Description of a new concept that captures Y and Z
6. Use it to solve the original problem
7. Possible implementations of this new concept
8. How this concept is actually implemented in systems used in production
9. Other practical uses of this new concept

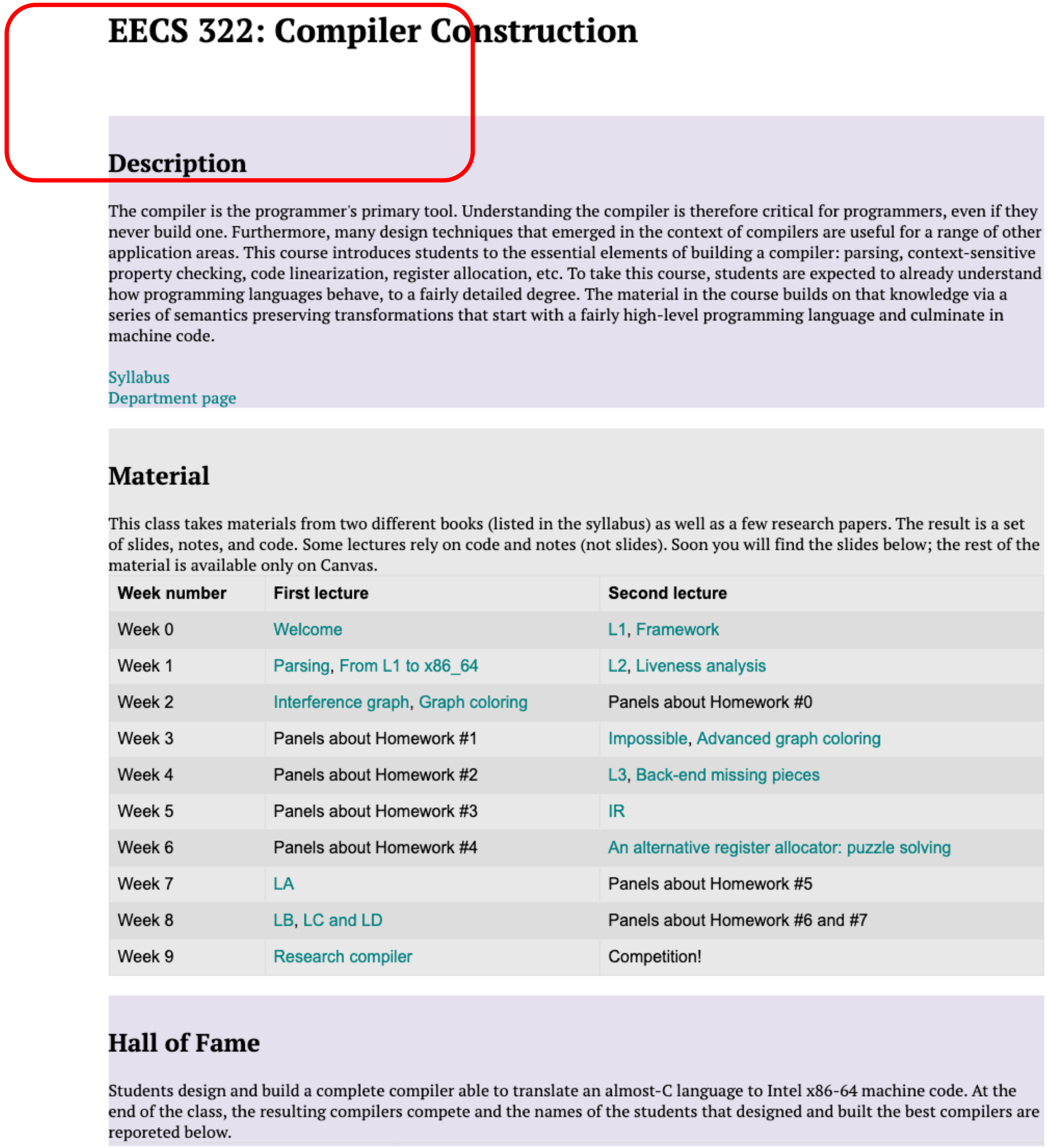
More steps, but each one is significantly quicker
(you don’t need to keep repeating yourself, the flow feels more natural)
Students stay more engaged

Challenge 3: motivate students to keep pushing theirself

- I gamified my classes as much as possible
- One assignment per week: range of points available
- Extra assignments for more points
    - Advance uses of topics learn in class
- End goal: competition during the last day of the class
    - Live
    - Cars/students will compete (ncurses-based framework)
    - They will compete against me as well
        - Something I didn’t expect: they really want to beat my solution

Hall of fame

Year	Name	Picture
2018 - 2019	Vijay Kandiah and Chenqi Guo	
2017 - 2018	Matt C. Cheung	
2016 - 2017	Zhiping Xiu	



EECS 322: Compiler Construction

Description

The compiler is the programmer's primary tool. Understanding the compiler is therefore critical for programmers, even if they never build one. Furthermore, many design techniques that emerged in the context of compilers are useful for a range of other application areas. This course introduces students to the essential elements of building a compiler: parsing, context-sensitive property checking, code linearization, register allocation, etc. To take this course, students are expected to already understand how programming languages behave, to a fairly detailed degree. The material in the course builds on that knowledge via a series of semantics preserving transformations that start with a fairly high-level programming language and culminate in machine code.

Syllabus
Department page

Material

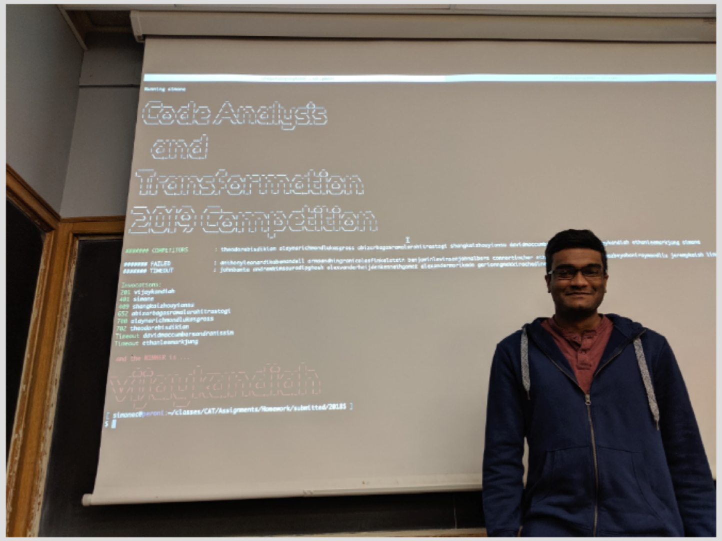
This class takes materials from two different books (listed in the syllabus) as well as a few research papers. The result is a set of slides, notes, and code. Some lectures rely on code and notes (not slides). Soon you will find the slides below; the rest of the material is available only on Canvas.

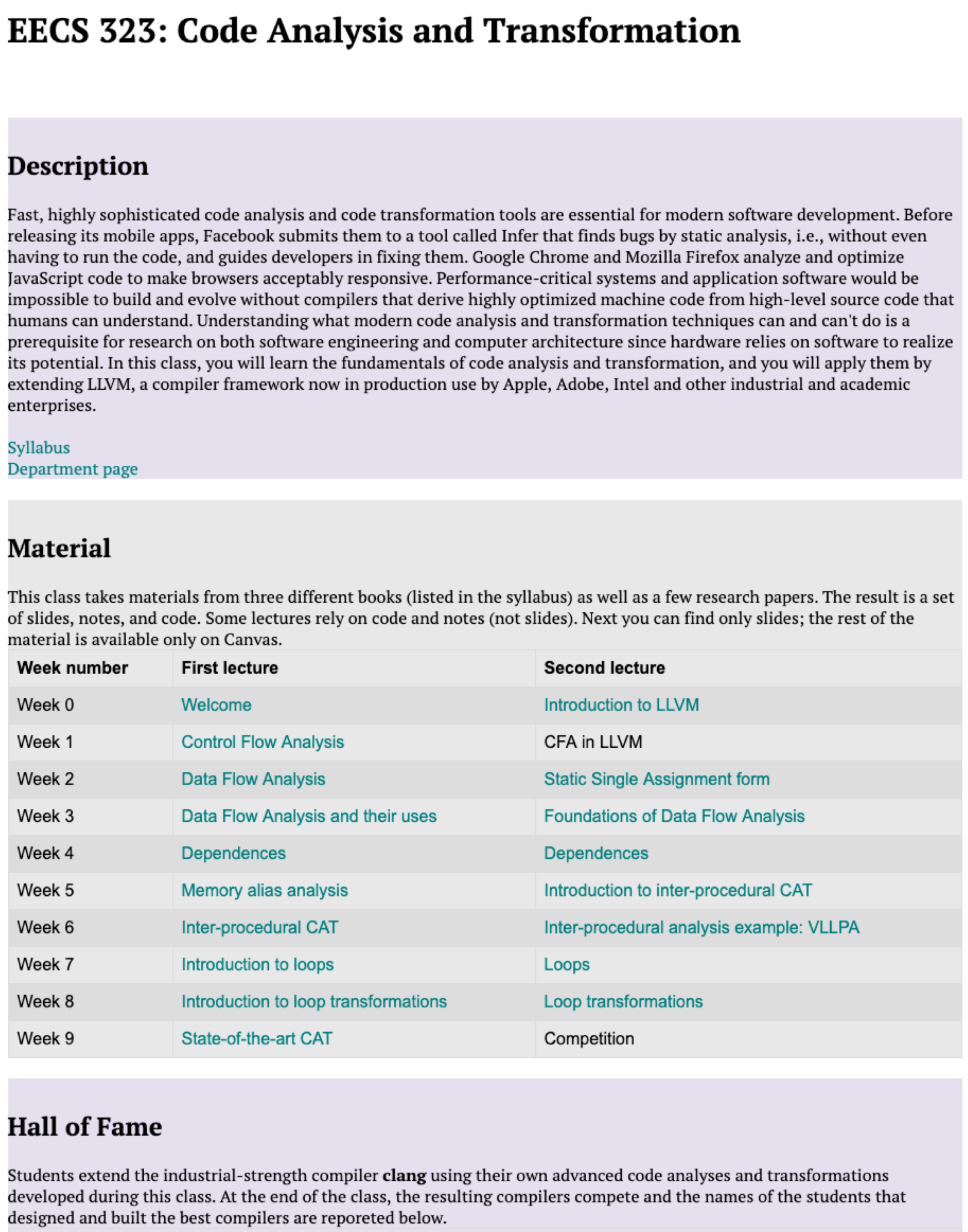
Week number	First lecture	Second lecture
Week 0	Welcome	L1, Framework
Week 1	Parsing, From L1 to x86_64	L2, Liveness analysis
Week 2	Interference graph, Graph coloring	Panels about Homework #0
Week 3	Panels about Homework #1	Impossible, Advanced graph coloring
Week 4	Panels about Homework #2	L3, Back-end missing pieces
Week 5	Panels about Homework #3	IR
Week 6	Panels about Homework #4	An alternative register allocator: puzzle solving
Week 7	LA	Panels about Homework #5
Week 8	LB, LC and LD	Panels about Homework #6 and #7
Week 9	Research compiler	Competition!

Hall of Fame

Students design and build a complete compiler able to translate an almost-C language to Intel x86-64 machine code. At the end of the class, the resulting compilers compete and the names of the students that designed and built the best compilers are reporeted below.

Hall of fame

Year	Name	Picture
2018 - 2019	Vijay Kandiah	
2017 - 2018	Angelo Matni	



EECS 323: Code Analysis and Transformation

Description

Fast, highly sophisticated code analysis and code transformation tools are essential for modern software development. Before releasing its mobile apps, Facebook submits them to a tool called Infer that finds bugs by static analysis, i.e., without even having to run the code, and guides developers in fixing them. Google Chrome and Mozilla Firefox analyze and optimize JavaScript code to make browsers acceptably responsive. Performance-critical systems and application software would be impossible to build and evolve without compilers that derive highly optimized machine code from high-level source code that humans can understand. Understanding what modern code analysis and transformation techniques can and can't do is a prerequisite for research on both software engineering and computer architecture since hardware relies on software to realize its potential. In this class, you will learn the fundamentals of code analysis and transformation, and you will apply them by extending LLVM, a compiler framework now in production use by Apple, Adobe, Intel and other industrial and academic enterprises.

Syllabus
Department page

Material

This class takes materials from three different books (listed in the syllabus) as well as a few research papers. The result is a set of slides, notes, and code. Some lectures rely on code and notes (not slides). Next you can find only slides; the rest of the material is available only on Canvas.

Week number	First lecture	Second lecture
Week 0	Welcome	Introduction to LLVM
Week 1	Control Flow Analysis	CFA in LLVM
Week 2	Data Flow Analysis	Static Single Assignment form
Week 3	Data Flow Analysis and their uses	Foundations of Data Flow Analysis
Week 4	Dependences	Dependences
Week 5	Memory alias analysis	Introduction to inter-procedural CAT
Week 6	Inter-procedural CAT	Inter-procedural analysis example: VLLPA
Week 7	Introduction to loops	Loops
Week 8	Introduction to loop transformations	Loop transformations
Week 9	State-of-the-art CAT	Competition

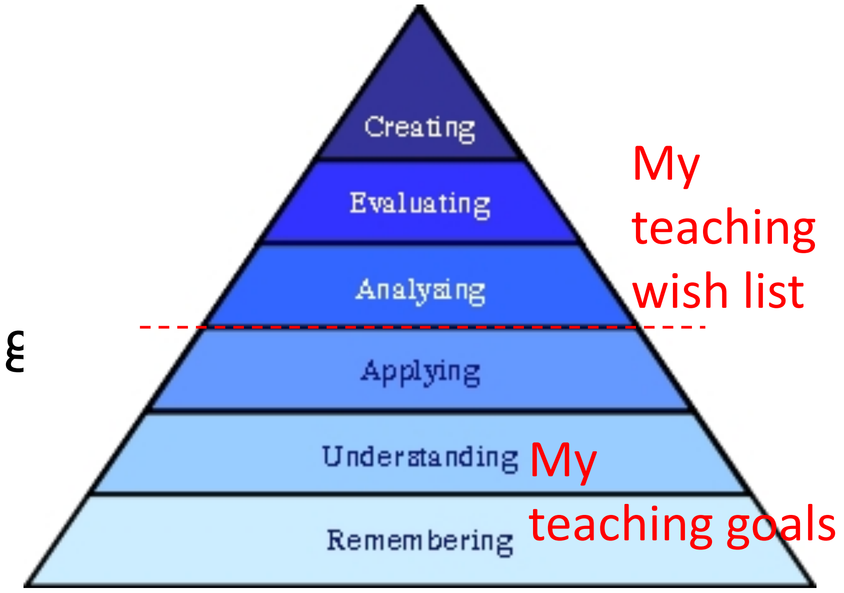
Hall of Fame

Students extend the industrial-strength compiler **clang** using their own advanced code analyses and transformations developed during this class. At the end of the class, the resulting compilers compete and the names of the students that designed and built the best compilers are reporeted below.

- **My 1st teaching goal:**
  make students feel this freedom (hopefully) like I do

- **My 2nd teaching goal:** show why compilers are fascinating

  Elegant, low-cost optimizations



- **My 3rd teaching goal:** a bad software/solution is expensive