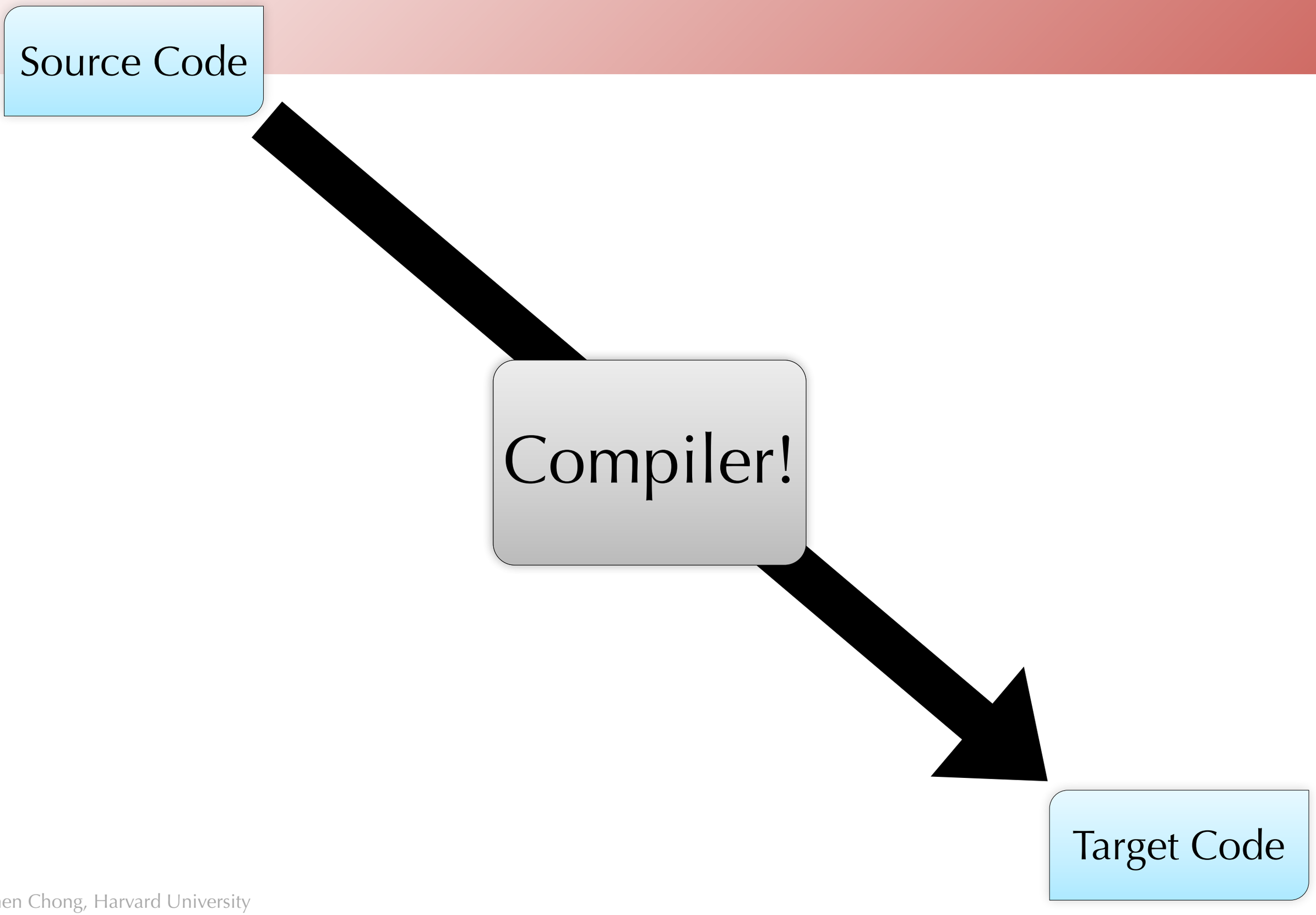
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

CS153: Compilers
Lecture 1: Introduction

Stephen Chong

https://www.seas.harvard.edu/courses/cs153

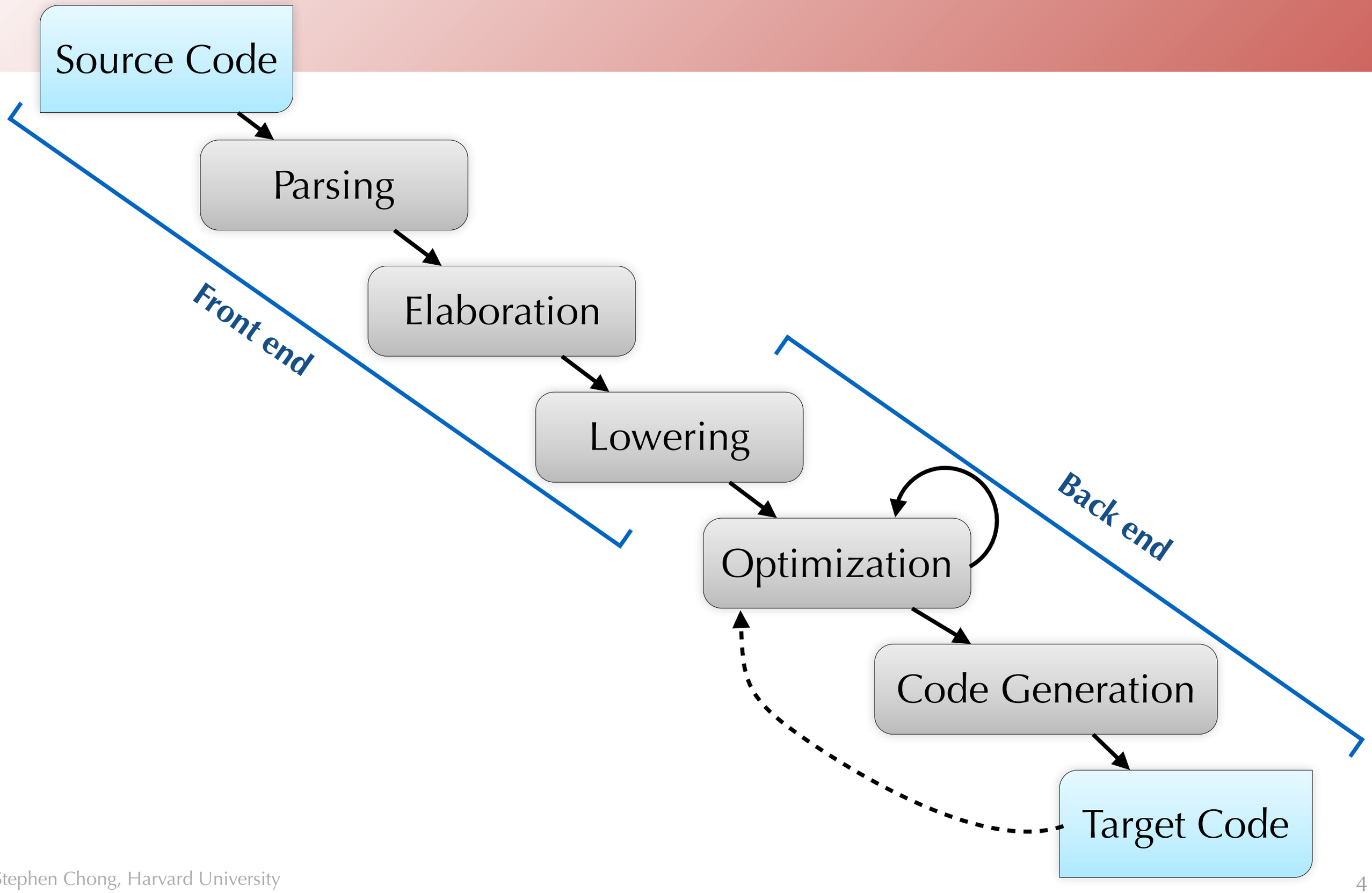
What is this course about?



What is this course about?

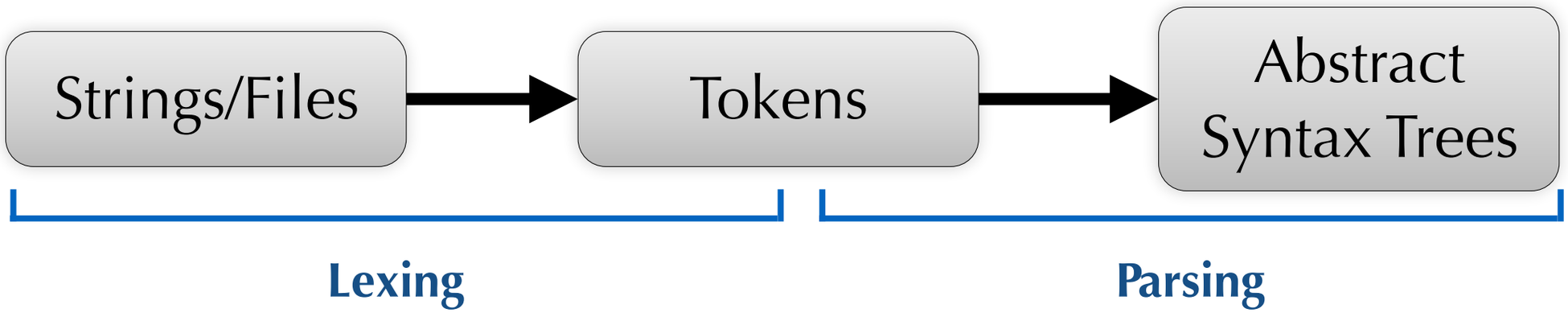
- How are programs written in a high-level language transformed into machine code?
- How can we ensure a program is somewhat meaningful?
- How is the program's memory managed?
- How can we analyze programs to discover invariant properties and improve their runtime performance?

Basic Architecture



Front end

- Lexing & Parsing
  - From strings to data structures
  - Usually split into two phases:

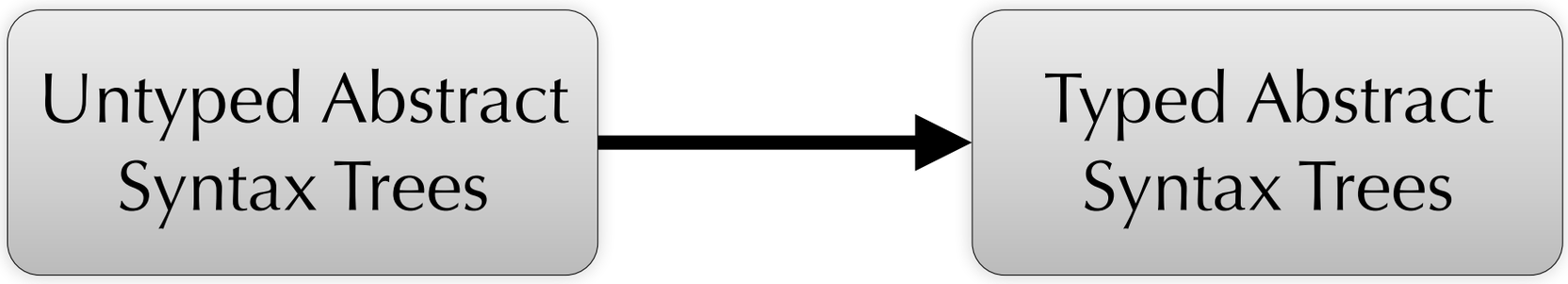


Parsing tools

- Parsing is something that happens in essentially all applications.
  - E.g., Google search bar, calendar, etc.
  - First step in going from raw data to information
- Lots of CS theory (121) to help out
  - Regular expressions (finite state automata)
  - Context-free grammars (push-down automata)
  - These abstractions show up in optimization too!
- Lots of tool support
  - E.g., Lex and Yacc; Antlr, parsing combinators; etc.

Elaboration

- Type-checking
  - Resolve variables, modules, etc.
  - Check that operations are given values of the right types
  - Infer types for sub-expressions
  - Check for other safety/security problems

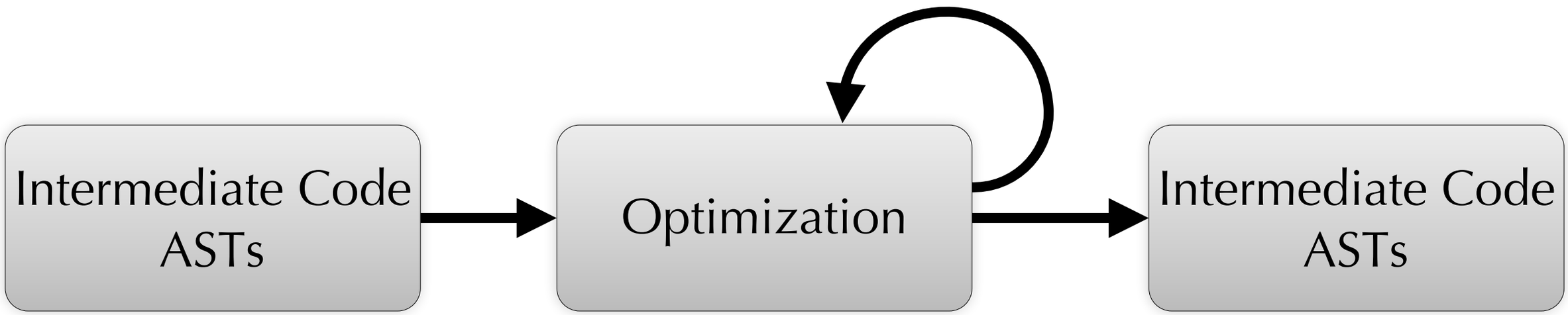


Lowering

- Translate high-level features into a small number of target-like constructs
  - e.g., `while`, `for`, `do`-loops all compiled to code using `goto`'s
  - e.g., objects, closures to records and function pointers
  - e.g., make type-tests, array-bounds checks, etc. explicit

Optimization

- Rewrite expensive sequences of operations into less expensive
  - e.g., constant folding: 3+4 → 7
  - e.g., lift an invariant computation out of a loop
  - e.g., parallelize a loop



Code generation

- Translate intermediate code into target code
  - Register assignment
  - Instruction selection
  - Instruction scheduling
  - Machine-specific optimization

Who should take CS153?

- People fascinated by languages & compilers
  - Why does[n't] this language include this feature?
  - Systems programmers
- Know the one tool that you use every day.
  - What can['t] a compiler do well?
- Architecture designers
  - Interdependence between compiler and architecture
  - See Intel iAPX 432 and Intel Itanium (compiler-related) failures; register windows
  - These days, architecture folks use compilers too!
- API designers
  - A language is the ultimate API
  - c.f., Facebook

Suggested prerequisites

- Ideally CS51 **and** CS61
  - CS51 alone is likely enough
- We assume
  - Knowledge of OCaml
  - A bit about how machines work
    - e.g., 2's complement arithmetic
- First project is a good time to get up to speed
- If you don't know something, ask!

Course Staff

Aaron Bembenek

Andrew Wong-Rolle

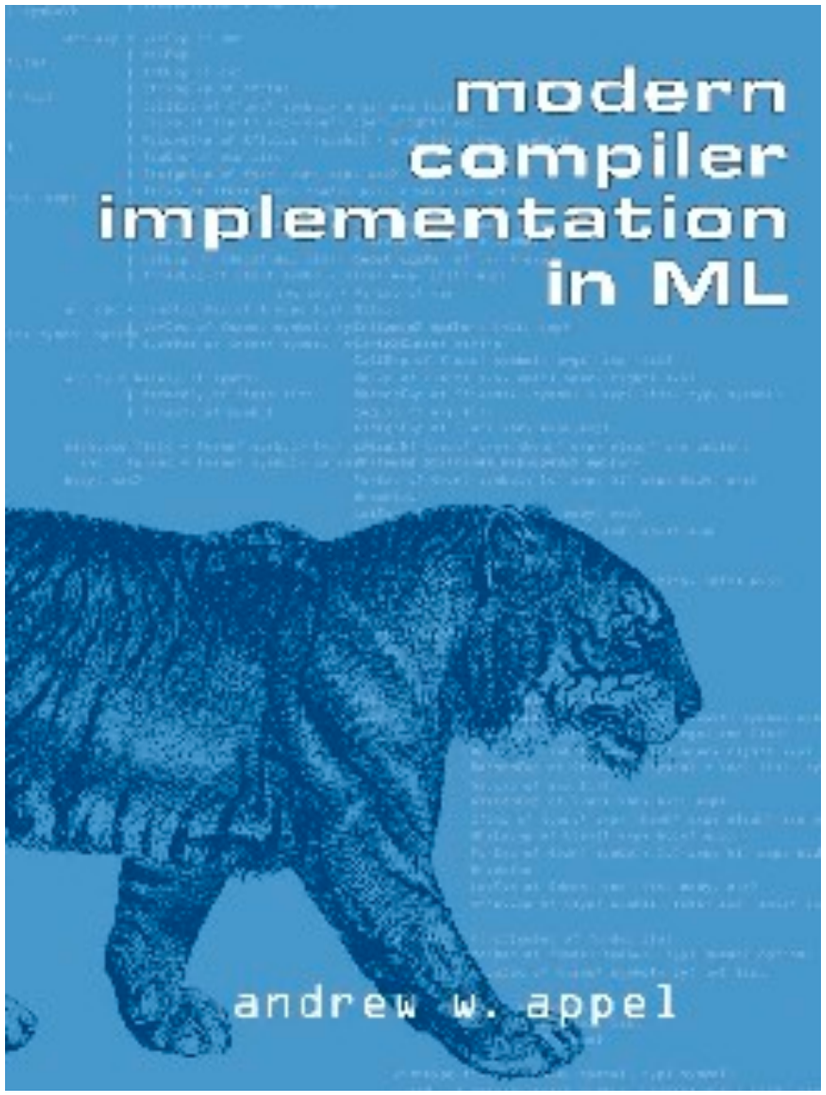
- Contact course staff at
  cs153-staff@seas.harvard.edu
  or via Piazza
- Don't send email to individual course staff
  (slower response, all of us need to be informed)

Administrivia

- Website
  - https://www.seas.harvard.edu/courses/cs153
- Piazza
  - https://piazza.com/class#fall2018/cs153
    - In general, post questions publicly unless you think question or answer might be inappropriate sharing of project solutions
    - Can post anonymously to classmates if you wish
- Office hours
  - Will be announced later. Start next week
  - Look on course website
- No section
  - TF hours will be spent on office hours instead

Textbook

- *Modern Compiler Implementaton in ML* by Andrew W. Appel
  - Recommended but not required.
- In most cases, class materials should suffice
- Lecture slides posted after the lecture
  (or before if they are ready in time)



Programming environment

- OCaml
  - Ideal for writing compilers!
- SPIM
  - MIPS simulator
  - Ideal target for compilers.
- GitHub
- ...

Assessment

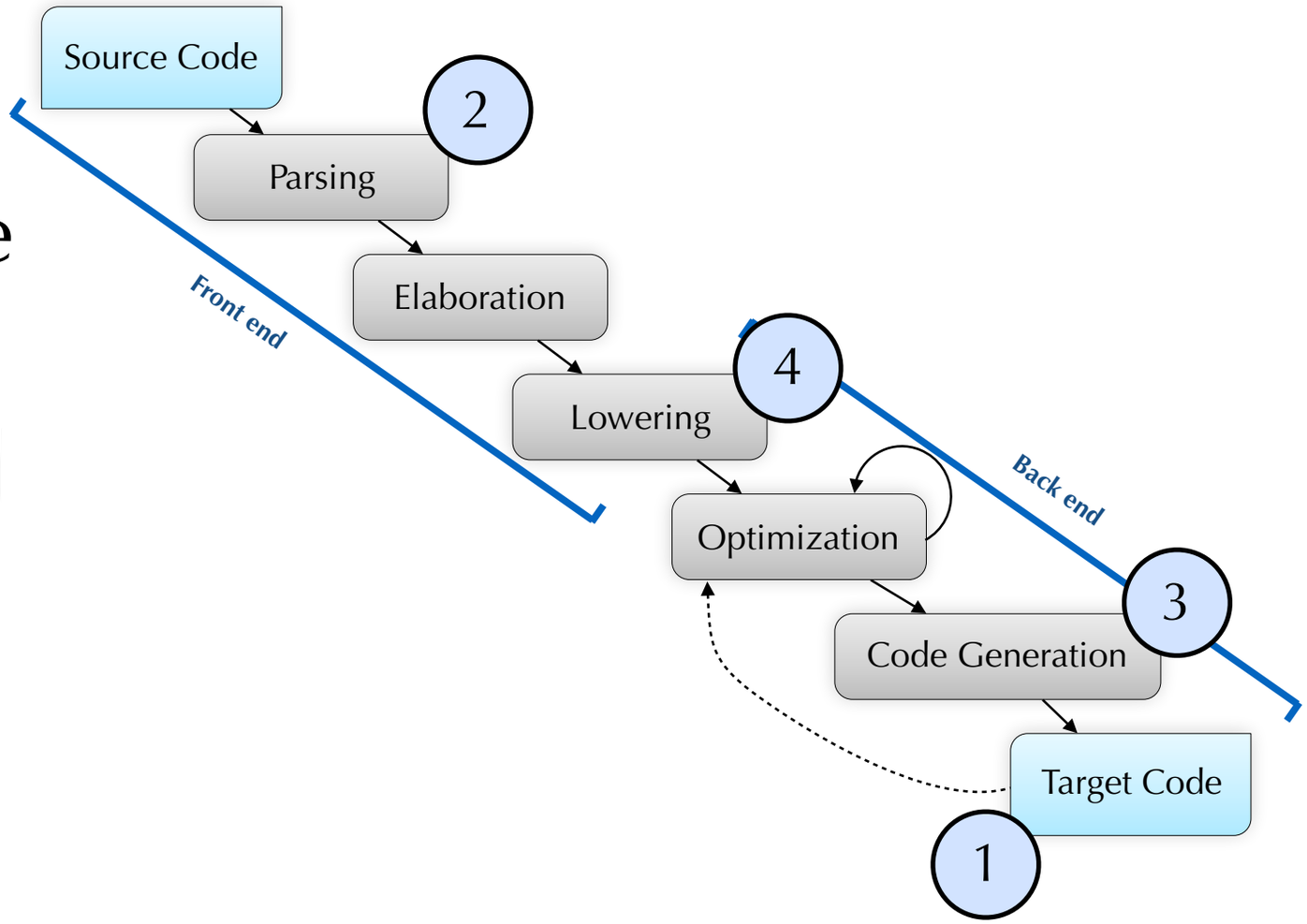
- Roughly
  - 70% projects (~8 projects total)
  - 25% final exam
  - 5% participation
    - E.g., Piazza posting/answering, attend lectures and engage in discussion, attend office hours, ...

Projects

- 8(ish) projects
- Mostly implementing parts of a compiler
- Typically 1-2 weeks per project
- In OCaml
- Plan ahead!
  - No late days! No late submissions!
    - But plenty of time to work on it
  - All submissions must type-check and compile
  - Multiple projects out at same time
- Implementation heavy course!
- **Strongly** encourage you to work with a partner!
  - Next lecture will give Google form if you are looking for project partner

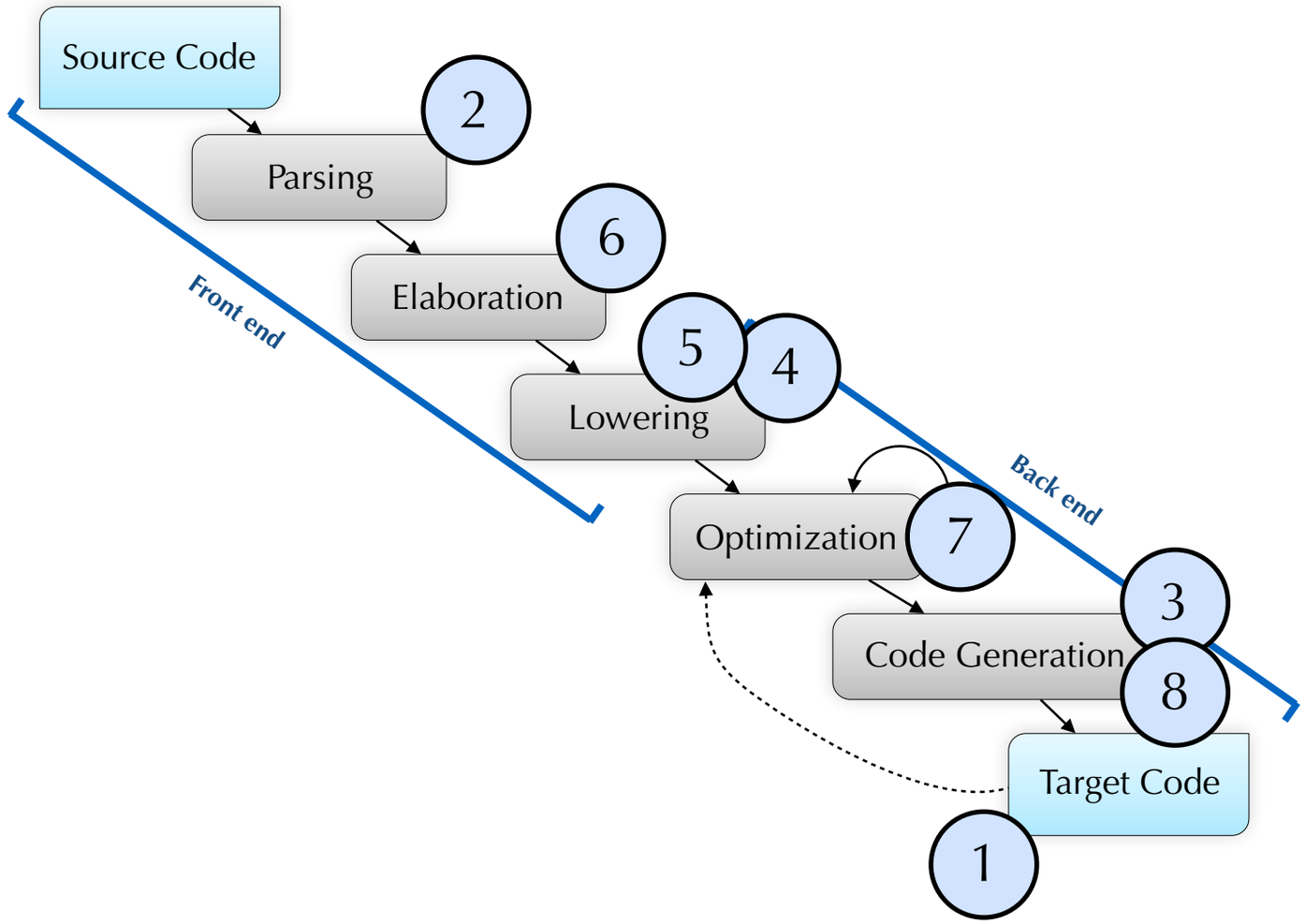
Projects

- 1. MIPS simulator
  - Understand the machine we are targeting
- 2. Fortran-ish front end
  - Parsing and lexing
- 3. Fortran-ish → MIPS
  - Simple front-end
  - Simple lowering & code generation
- 4. C-ish → MIPS
  - 1st-order procedures, structs, arrays



Projects

- 5. Scheme-ish → C-ish
  - Higher-order procedures, type-tests
- 6. ML-ish → Scheme-ish
  - Type inference
- 7. Algebraic optimization
  - Tree-based rewriting
- 8. Control-flow graphs
  - Rip out naïve C-ish backend and replace with one based on control-flow graphs.
  - Implement liveness analysis
  - Implement graph-coloring register allocation



Learning outcomes

- Understand how compilers work
  - Parsing, type-checking & inference, lowering, analysis, optimization, code generation.
  - How to rewrite your code so that the compiler can do its job better.
- Programming experience
- Ability to abstract and improve an API with a formal language

Collaboration/Academic integrity

- Discussion and exchange of ideas good!
- Consult with your classmates as you work on problem sets.
- **But:** work you submit for evaluation should be result only of your and your project partner's efforts
- Do not share code nor accept code from other students
- Do not post course materials (including projects, solutions, exams, etc.) to websites (including public GitHub repositories, and similar) or course-content archives
- Don't look on the web for solutions
- **If you are ever in doubt, ask the course staff to clarify what is and isn't appropriate.**

Diversity and Inclusion

- Aim: create a learning environment that supports a diversity of thoughts, perspectives and experiences, and honors your identities (including race, gender, class, sexuality, religion, ability, etc.)

Diversity and Inclusion

- To help accomplish this:
  - If you have a name and/or set of pronouns that differ from those that appear in your official Harvard records, please let me know!
  - If you feel like your performance in the class is being impacted by your experiences outside of class, please don't hesitate to come and talk with me. I want to be a resource for you. If you prefer to speak with someone outside of the course, members of the SEAS Committee on Diversity, Inclusion, and Belonging are excellent resources.
  - I (like many people) am still in the process of learning about diverse perspectives and identities. If something was said (by anyone) in class, office hours, Piazza, or project group work that made you feel uncomfortable, please talk to me about it.
  - As a participant in course discussions, office hours, and group projects, you should also strive to honor the diversity of your classmates.
- If you ever are struggling and just need someone to talk to, feel free to stop by office hours, or to reach out to me and we can arrange a private meeting.

Questions/comments?