Advanced Topics in Compilers

Welcome!

Simone Campanoni
simone.campanoni@northwestern.edu

Prerequisites

Source code (e.g., C++)

i	n	t		m	a	i	n		…

↓

Front-end — ***EECS 322: Compiler Construction***

↓ IR ↓

```
myVarX = 40
myVarY = myVarX + 2
```

Middle-end — ***EECS 323: Code analysis and transformation***

↓ IR ↓

```
myVarY = 42
```

Back-end — ***EECS 322: Compiler Construction***

↓

Machine code

01010111010101010101

Goal and mindset of ATC

- The goal is to expose you to compiler research
    - An example of how we identify problems that require novelty
    - How we create novelty
    - How we implement solutions
    - And how we test it

- It requires a lot of independence on your end
    - You don’t know something?
      Check the makefiles, sources, documentation (when it exists)
    - Is something not working?
      Understand why, debug it, fix it, send pull requests

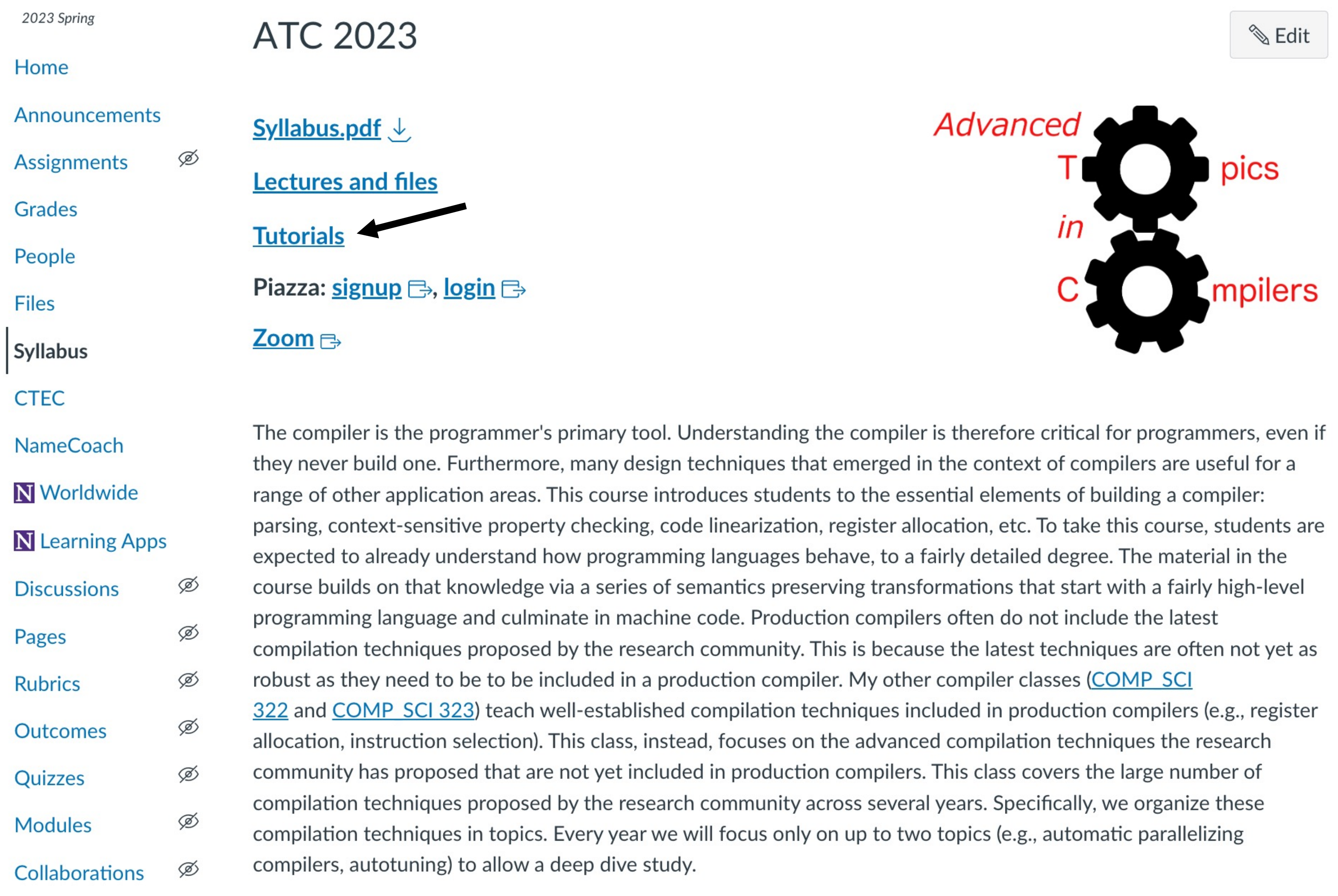
Outline

- Structure of the course
- This year’s topics
- Projects

ATC

- You learned the internals of modern production-quality compilers
  - E.g., Data-flow analysis, constant propagation, memory alias analysis (CS 323)
  - E.g., Instruction selection, register allocation (CS 322)
- Research labs include advance techniques not yet included in production-quality compilers
  - They are not (yet) as robust as production-quality compilers need to be
- ATC:
  - You will learn some of these advanced techniques
    - They are organized in topics
    - Each year we look at techniques of two topics
    - Each year ATC is different

ATC on Canvas



2023 Spring

- Home
- Announcements
- Assignments
- Grades
- People
- Files
- **Syllabus**
- CTEC
- NameCoach
- Worldwide
- Learning Apps
- Discussions
- Pages
- Rubrics
- Outcomes
- Quizzes
- Modules
- Collaborations

ATC 2023

Edit

Syllabus.pdf

Lectures and files

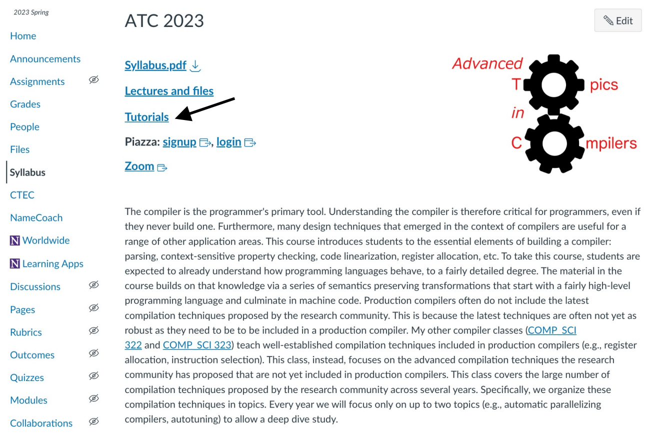
Tutorials

Piazza: signup, login

Zoom

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. Production compilers often do not include the latest compilation techniques proposed by the research community. This is because the latest techniques are often not yet as robust as they need to be to be included in a production compiler. My other compiler classes (COMP_SCI 322 and COMP_SCI 323) teach well-established compilation techniques included in production compilers (e.g., register allocation, instruction selection). This class, instead, focuses on the advanced compilation techniques the research community has proposed that are not yet included in production compilers. This class covers the large number of compilation techniques proposed by the research community across several years. Specifically, we organize these compilation techniques in topics. Every year we will focus only on up to two topics (e.g., automatic parallelizing compilers, autotuning) to allow a deep dive study.

ATC on Canvas



ATC 2023

Syllabus.pdf

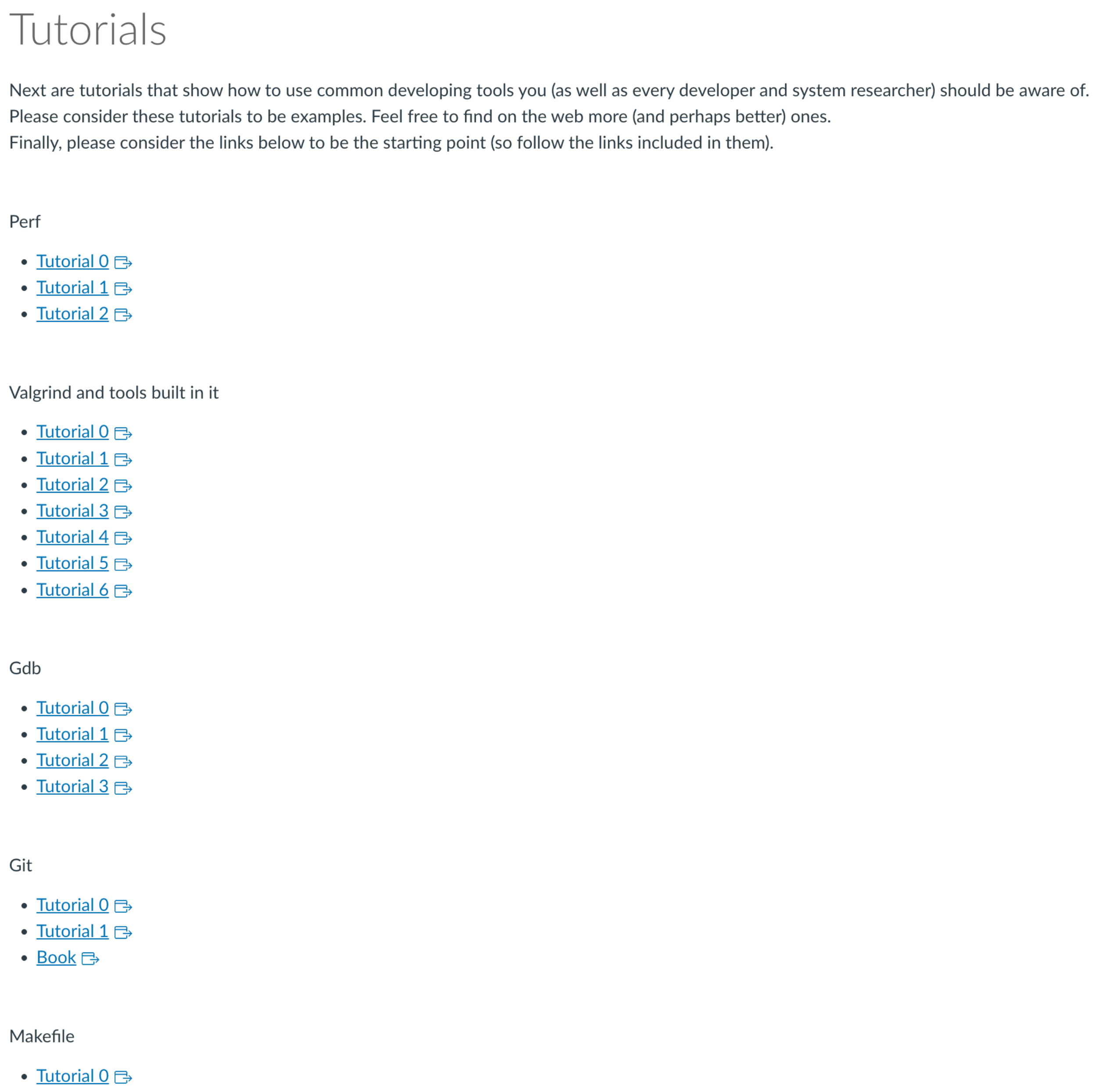
Lectures and files

Tutorials

Piazza: signup, login

Zoom

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. Production compilers often do not include the latest compilation techniques proposed by the research community. This is because the latest techniques are often not yet as robust as they need to be to be included in a production compiler. My other compiler classes (COMP_SCI 322 and COMP_SCI 323) teach well-established compilation techniques included in production compilers (e.g., register allocation, instruction selection). This class, instead, focuses on the advanced compilation techniques the research community has proposed that are not yet included in production compilers. This class covers the large number of compilation techniques proposed by the research community across several years. Specifically, we organize these compilation techniques in topics. Every year we will focus only on up to two topics (e.g., automatic parallelizing compilers, autotuning) to allow a deep dive study.



Tutorials

Next are tutorials that show how to use common developing tools you (as well as every developer and system researcher) should be aware of.
Please consider these tutorials to be examples. Feel free to find on the web more (and perhaps better) ones.
Finally, please consider the links below to be the starting point (so follow the links included in them).

Perf

- Tutorial 0
- Tutorial 1
- Tutorial 2

Valgrind and tools built in it

- Tutorial 0
- Tutorial 1
- Tutorial 2
- Tutorial 3
- Tutorial 4
- Tutorial 5
- Tutorial 6

Gdb

- Tutorial 0
- Tutorial 1
- Tutorial 2
- Tutorial 3

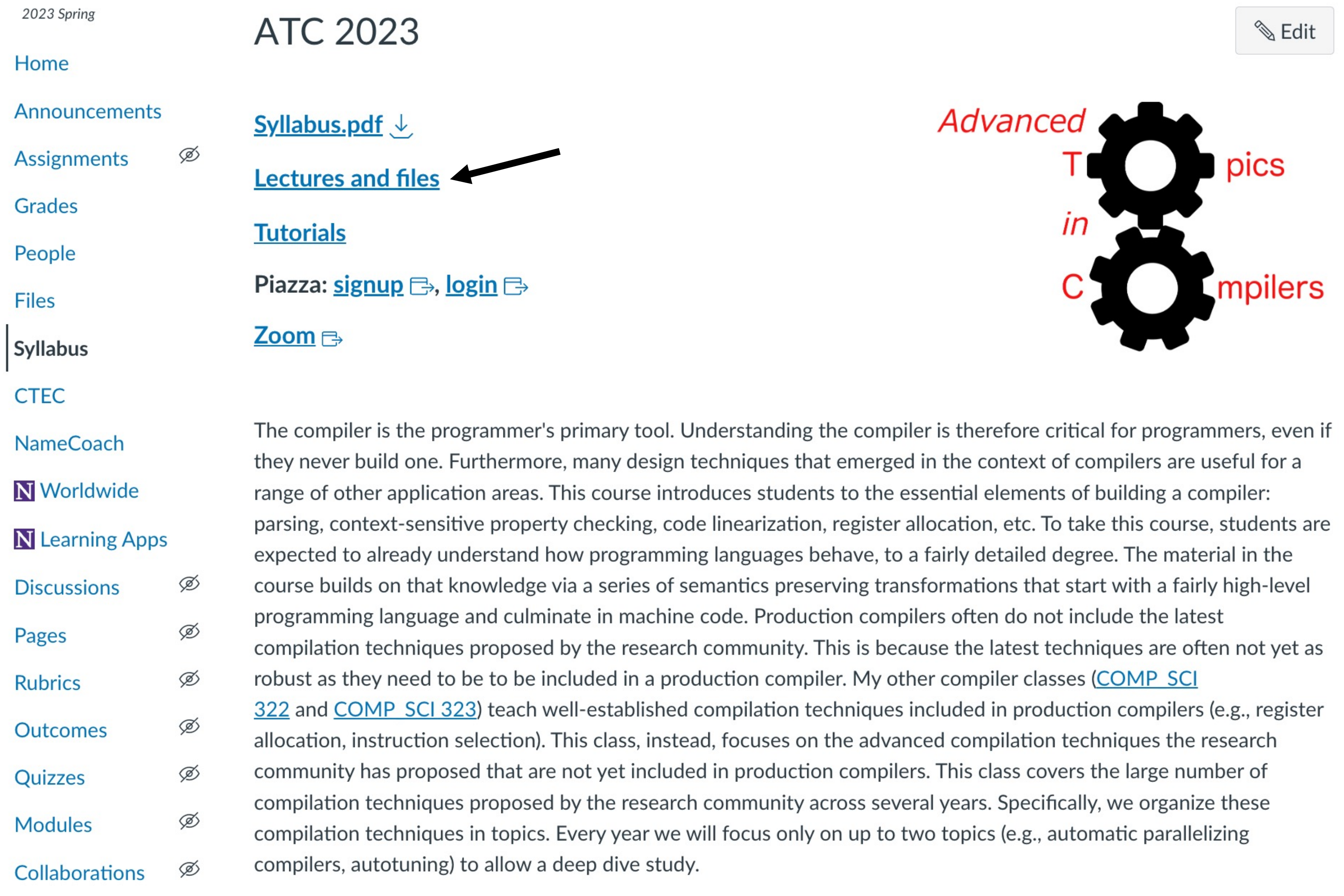
Git

- Tutorial 0
- Tutorial 1
- Book

Makefile

- Tutorial 0

ATC on Canvas



2023 Spring

- Home
- Announcements
- Assignments
- Grades
- People
- Files
- Syllabus
- CTEC
- NameCoach
- Worldwide
- Learning Apps
- Discussions
- Pages
- Rubrics
- Outcomes
- Quizzes
- Modules
- Collaborations

ATC 2023

Edit

Syllabus.pdf

Lectures and files

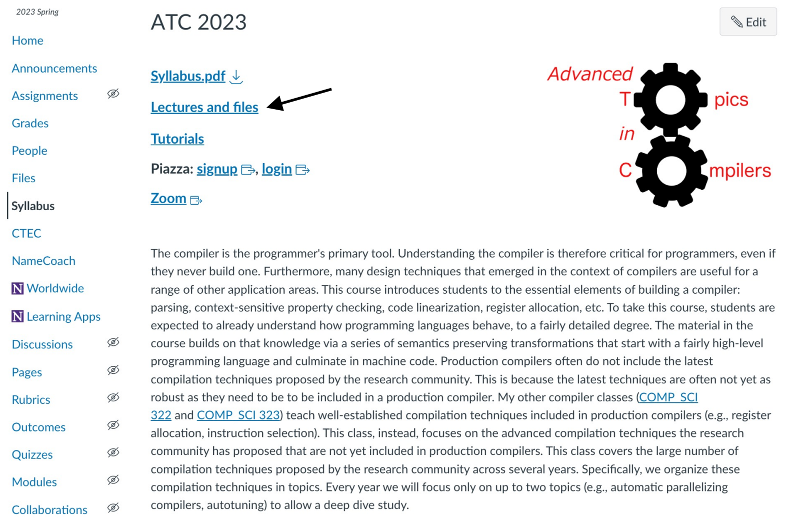
Tutorials

Piazza: signup, login

Zoom

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. Production compilers often do not include the latest compilation techniques proposed by the research community. This is because the latest techniques are often not yet as robust as they need to be to be included in a production compiler. My other compiler classes (COMP_SCI 322 and COMP_SCI 323) teach well-established compilation techniques included in production compilers (e.g., register allocation, instruction selection). This class, instead, focuses on the advanced compilation techniques the research community has proposed that are not yet included in production compilers. This class covers the large number of compilation techniques proposed by the research community across several years. Specifically, we organize these compilation techniques in topics. Every year we will focus only on up to two topics (e.g., automatic parallelizing compilers, autotuning) to allow a deep dive study.

ATC on Canvas



2023 Spring

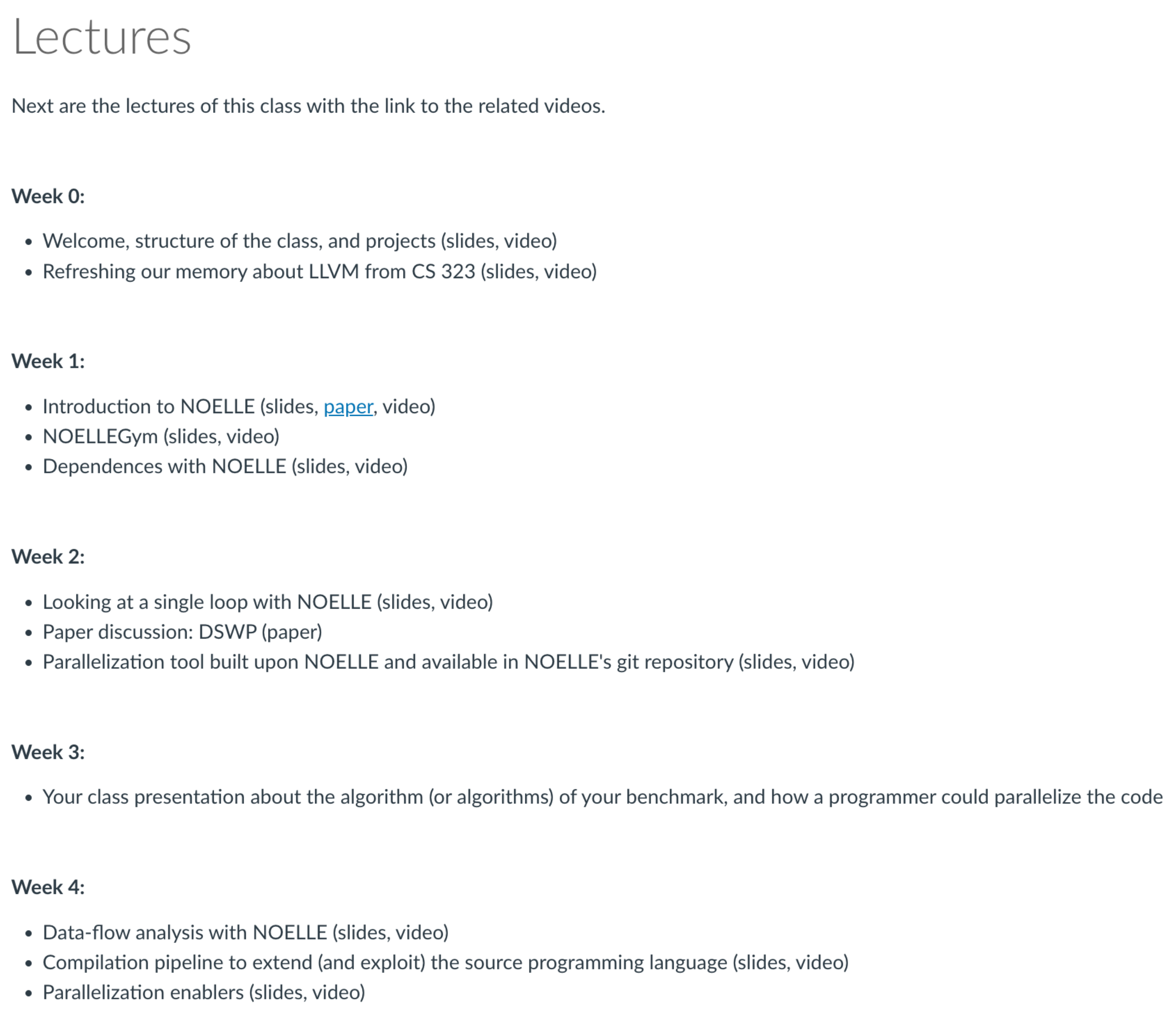
Home
Announcements
Assignments
Grades
People
Files
Syllabus
CTEC
NameCoach
Worldwide
Learning Apps
Discussions
Pages
Rubrics
Outcomes
Quizzes
Modules
Collaborations

ATC 2023

Edit

Syllabus.pdf
Lectures and files
Tutorials
Piazza: signup, login
Zoom

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. Production compilers often do not include the latest compilation techniques proposed by the research community. This is because the latest techniques are often not yet as robust as they need to be to be included in a production compiler. My other compiler classes (COMP_SCI 322 and COMP_SCI 323) teach well-established compilation techniques included in production compilers (e.g., register allocation, instruction selection). This class, instead, focuses on the advanced compilation techniques the research community has proposed that are not yet included in production compilers. This class covers the large number of compilation techniques proposed by the research community across several years. Specifically, we organize these compilation techniques in topics. Every year we will focus only on up to two topics (e.g., automatic parallelizing compilers, autotuning) to allow a deep dive study.



Lectures

Next are the lectures of this class with the link to the related videos.

Week 0:

- Welcome, structure of the class, and projects (slides, video)
- Refreshing our memory about LLVM from CS 323 (slides, video)

Week 1:

- Introduction to NOELLE (slides, paper, video)
- NOELLEGym (slides, video)
- Dependences with NOELLE (slides, video)

Week 2:

- Looking at a single loop with NOELLE (slides, video)
- Paper discussion: DSWP (paper)
- Parallelization tool built upon NOELLE and available in NOELLE's git repository (slides, video)

Week 3:

- Your class presentation about the algorithm (or algorithms) of your benchmark, and how a programmer could parallelize the code

Week 4:

- Data-flow analysis with NOELLE (slides, video)
- Compilation pipeline to extend (and exploit) the source programming language (slides, video)
- Parallelization enablers (slides, video)

ATC assignments

- Project: you will do a project you choose from a set
  - The set of projects are related to the topics of the current year
  - You can work in a team (maximum 2 people per team) only for 397
  - You will develop your project during the quarter
  - Each team will meet me weekly to discuss progress and roadblocks
  - You will present your project to the rest of the class 3 times
    at the 3 milestones (described next)

ATC assignments

- Project: you will do a project you choose from a set
  - The set of projects are related to the topics of the current year
  - You can work in a team (maximum 2 people per team) only for 397
  - You will develop your project during the quarter
  - Each team will meet me weekly to discuss progress and roadblocks
  - You will present your project to the rest of the class 3 times
    at the 3 milestones (described next)

Papers: you will learn some advanced compilation techniques

- You need to read and learn research papers (available on Canvas)
- You will present a few of them and you will need to defend them (as they are your papers)
- If you are not presenting a paper,
  then you'll need to poke the idea to find potential limitations

ATC assignments

- Project: you will do a project you choose from a set
  - The set of projects are related to the topics of the current year
  - You can work in a team (maximum 2 people per team) only for 397
  - You will develop your project during the quarter
  - Each team will meet me weekly to discuss progress and roadblocks
  - You will present your project to the rest of the class 3 times
    at the 3 milestones (described next)

Papers: you will learn some advanced compilation techniques

- You need to read and learn research papers (available on Canvas)
- You will present a few of them and you will need to defend them
  (as they are your papers)
- If you are not presenting a paper,
  then you'll need to poke the idea to find potential limitations

Tutorials: you will learn how to use advanced compilation techniques

- You will learn how to use a real codebase that implements them
- You need to watch tutorial videos before class
- You will collaborate in class to solve new problems using this new knowledge

ATC assignments

- Project: you will do a project you choose from a set
  - The set of projects are related to the topics of the current year
  - You can work in a team (maximum 2 people per team) only for 397
  - You will develop your project during the quarter
  - Each team will meet me weekly to discuss progress and roadblocks
  - You will present your project to the rest of the class 3 times
    at the 3 milestones (described next)

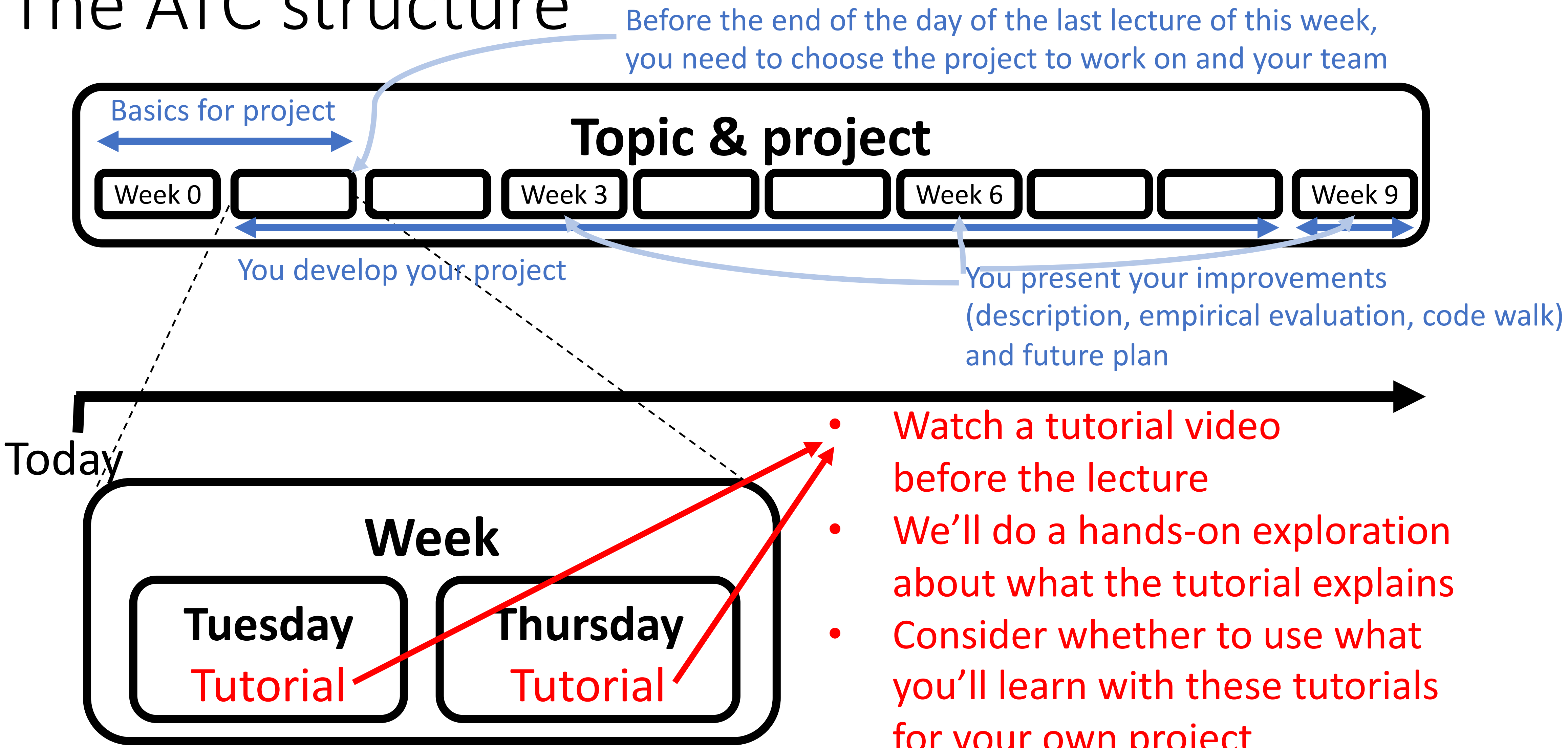
Tutorials: you will learn how to use advanced compilation techniques

- You will learn how to use a real codebase that implements them
- You need to watch tutorial videos before class
- You will collaborate in class to solve new problems using this new knowledge

Papers: you will learn some advanced compilation techniques

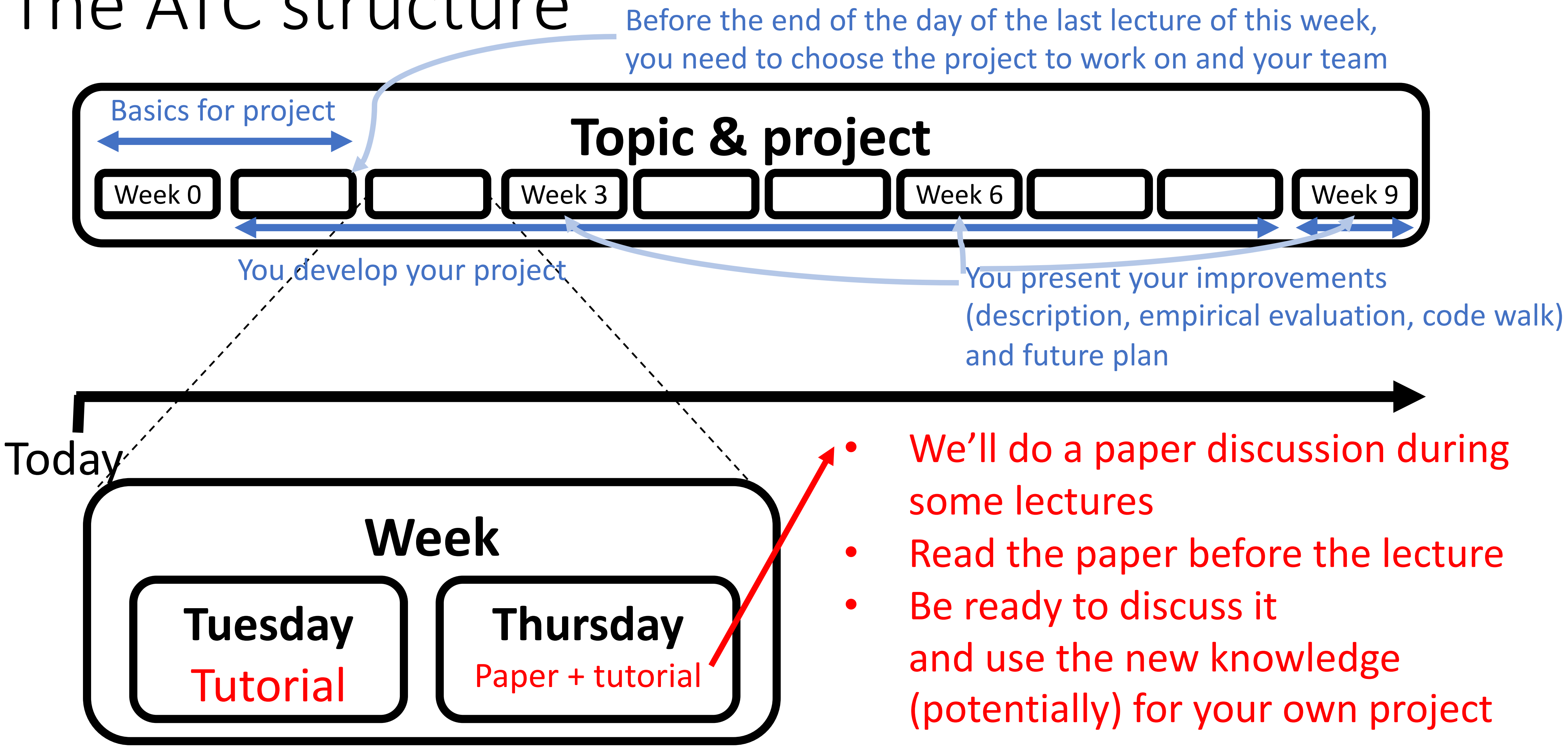
- You need to read and learn research papers (available on Canvas)
- You will present a few of them and you will need to defend them
  (as they are your papers)
- If you are not presenting a paper,
  then you'll need to poke the idea to find potential limitations

The ATC structure


Before the end of the day of the last lecture of this week, you need to choose the project to work on and your team
Basics for project
Topic & project
Week 0
Week 3
Week 6
Week 9
You develop your project
You present your improvements (description, empirical evaluation, code walk) and future plan
Today
Week
Tuesday
Tutorial
Thursday
Tutorial

- Watch a tutorial video before the lecture
- We'll do a hands-on exploration about what the tutorial explains
- Consider whether to use what you'll learn with these tutorials for your own project

The ATC structure


Before the end of the day of the last lecture of this week, you need to choose the project to work on and your team
Basics for project
Topic & project
Week 0
Week 3
Week 6
Week 9
You develop your project
You present your improvements (description, empirical evaluation, code walk) and future plan
Today
Week
Tuesday
Tutorial
Thursday
Paper + tutorial

- We’ll do a paper discussion during some lectures
- Read the paper before the lecture
- Be ready to discuss it and use the new knowledge (potentially) for your own project

Papers

- Distributed (or linked) through Canvas
- Each paper will be discussed as following
  - Half of the class will be the presenters
    - They will briefly present the paper (5 minutes)
  - Half of the class will be the skeptical ones
    - They need to ask questions or make claims about their (objective) skepticism
- Everyone needs to read all papers before the class and participate in the discussion
  - When somebody asks a question (including myself), I might use the “name wheel” to pick who is going to answer it
    - A name wheel will choose randomly a name from a pool

Mindset to use for paper reading

- In this class: you will read 1 paper for a lecture
  - Not 2-4

- Expectation: you need to take the time to read the single paper very well
  - You need to understand it in depth
    Simple test: can you implement what is described?
      - Yes: you understood it
      - No: please re-read it

Materials

- Software:
  - NOELLE: it can be downloaded from here
    - our set of abstractions/transformations/analyses that can be used by an LLVM middle-end pass
    - set of tools including a parallelizing compiler
  - NOELLEGym: it can be downloaded from here
    infrastructure to test NOELLE-based optimizations
    on benchmarks typically used in research venues
  - VIRGIL: it can be downloaded from here
    our task engine that is used by (for example) the NOELLE-based parallelizing compiler
  - Not ours: from the web
- Documentation:
  - Tutorial video: links on Canvas
  - Papers: either from the web (when available) or from Canvas
  - Software docs: from the web (when they exist)

Outline

- Structure of the course

- This year’s topics

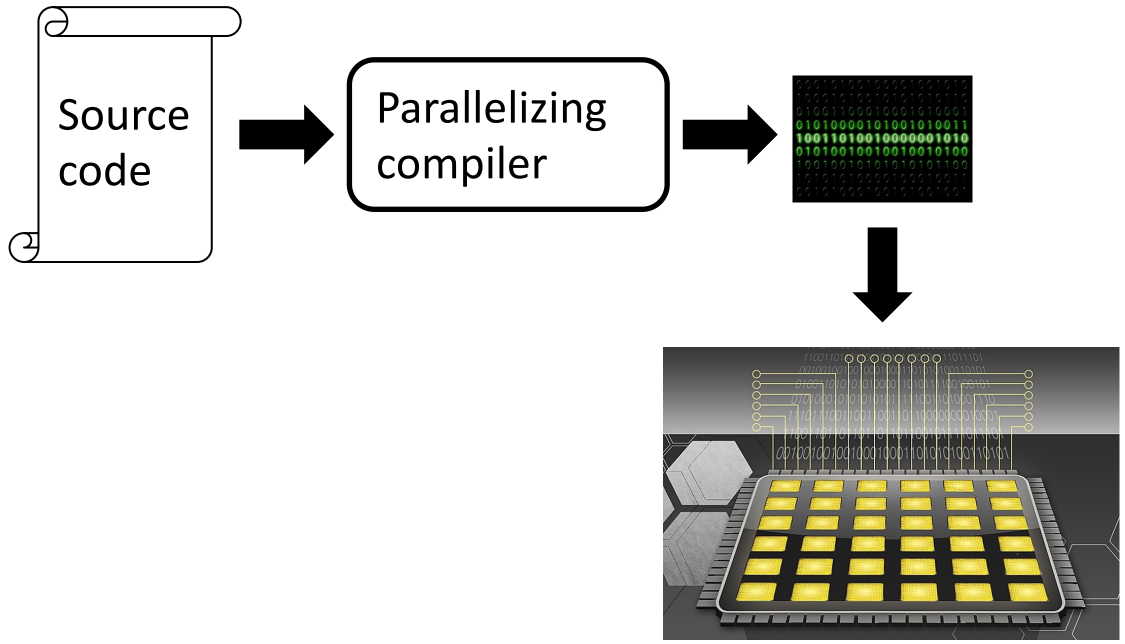
- Projects

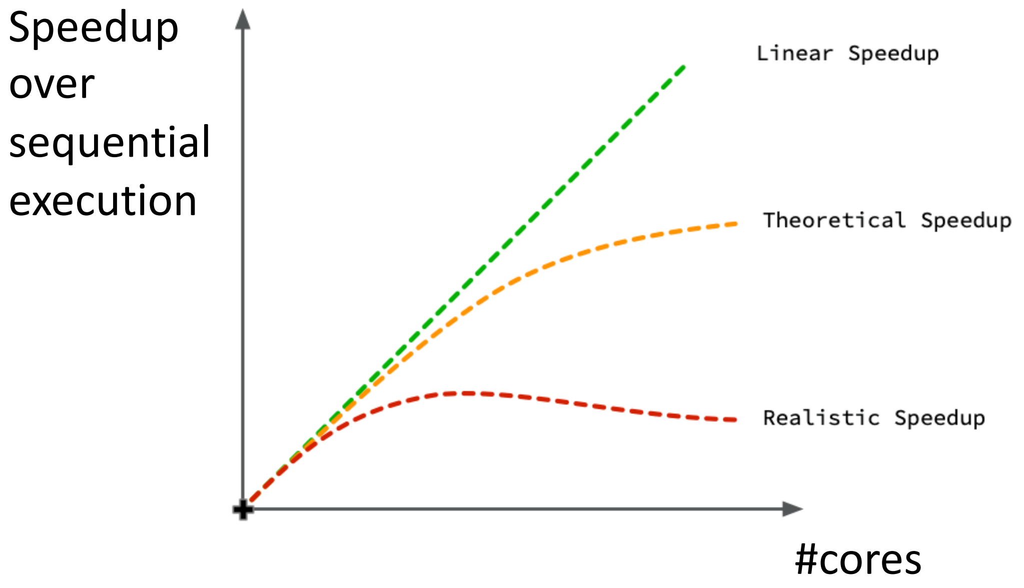
Topics

- Every year ATC covers different topics

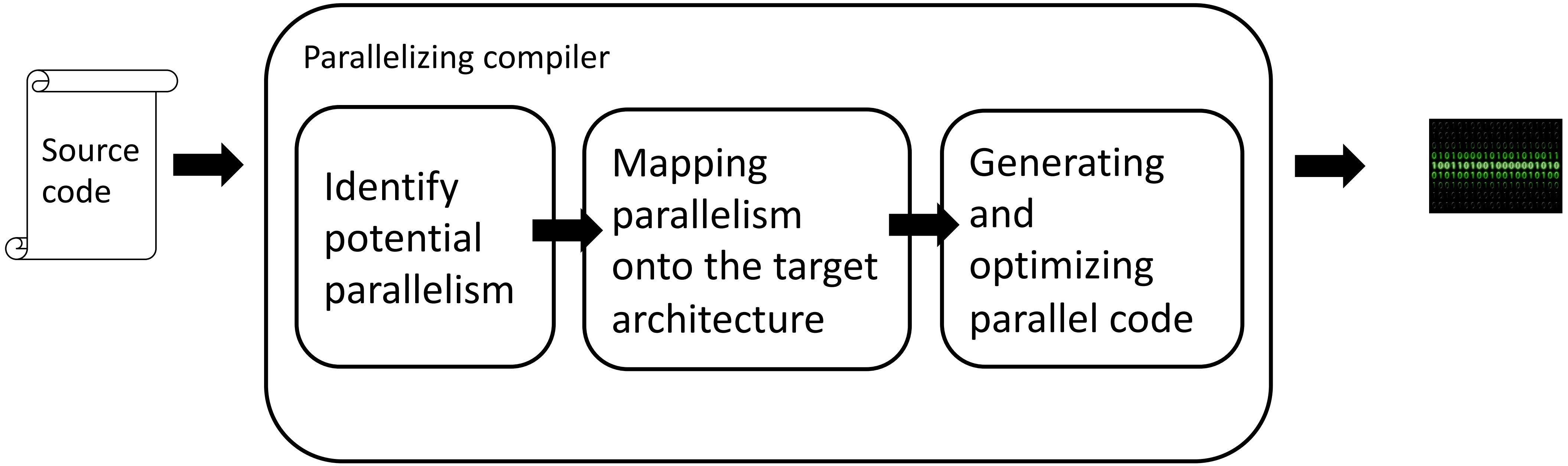
- Topics we will cover this year:
    - Parallelizing compilers

A parallelizing compiler

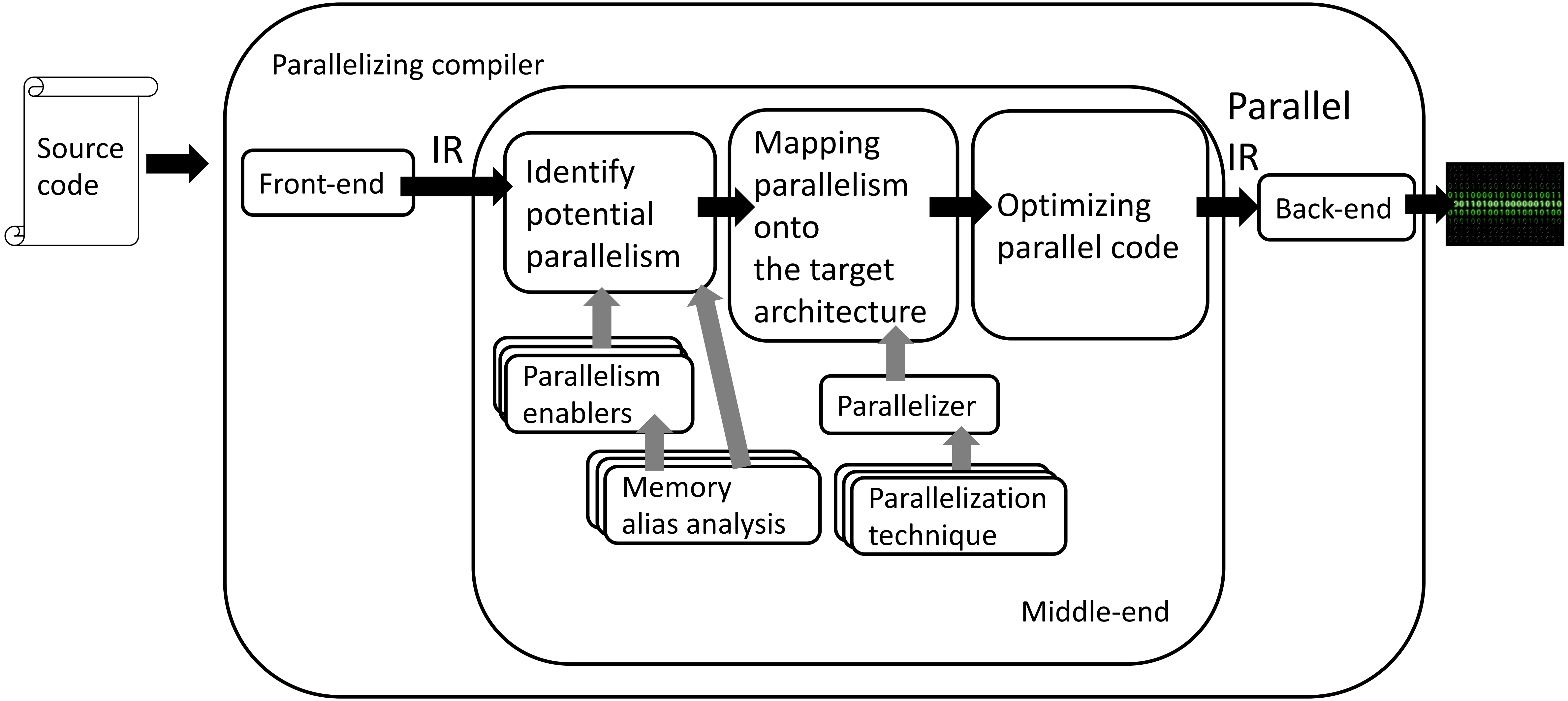




A parallelizing compiler


Source code
Parallelizing compiler
Identify potential parallelism
Mapping parallelism onto the target architecture
Generating and optimizing parallel code

A typical parallelizing compiler



Outline

- Structure of the course
- This year’s topics
- Projects

Goal of your project

- Parallelize benchmark X (each project has a different X)
- You can choose X from a set (next described)
- Working infrastructure: NOELLEGym
    - It automatically download NOELLE and VIRGIL

Milestones for your project

Milestone 0:
you reach this milestone when you can describe how to parallelize the benchmark (manually) enough to get a high speedup

- Expectation:
  you prepare a talk where you describe
  - the algorithm (or algorithms) used in benchmark X
  - How to parallelize the code (manually) to get high speedups
  - Try to support your claims with as much data as you can
    (e.g., use perf and other tools, try to parallelize it manually)
- Deadline:
  forth week (week 3 as we start counting from 0 ☺)

Milestones for your project

Milestone 1:
you extended the paralleling compiler built upon NOELLE to parallelize your benchmark X

- Expectation: you prepare a talk where you describe
  - The roadblocks that block the parallelizing compiler (before your extensions) to parallelize X
  - The extensions you have designed and implemented to overcome some of these roadblocks
  - The speedup you've obtained
  - Your plan to overcome the remaining roadblocks
- Deadline:
  seventh week (week 6 as we start counting from 0 ☺)

Milestones for your project

Milestone 2:
you complete the extensions for the paralleling compiler built upon NOELLE to fully parallelize your benchmark X

- Expectation: you prepare a talk where you describe
  - The roadblocks that you've worked on from Milestone 1 that blocked the parallelizing compiler at Milestone 1 to fully parallelize X
  - The extensions you have designed and implemented to overcome ALL roadblocks
  - The final speedup you've obtained
  - Thoughts about future improvements to parallelize X even more
- Deadline:
  last week (week 9 as we start counting from 0 ☺)

Final suggestions for your work

- Do not work on milestone Y while you didn’t complete milestone Y-1
- But keep an eye to milestone Y while working on milestone Y-1
- All changes are allowed (changes to the parallelizing compiler, to NOELLE’s abstractions, to NOELLE’s runtime VIRGIL, to NOELLEGym, to the code of the benchmark, to the Programming Language (PL) used to implement the benchmark), but not all changes are equal
  - Prefer changes to the parallelizing compiler
  - Only if these are not enough, then consider changing NOELLE’s abstractions that power the parallelizing compiler
  - Only when the above changes are not enough, then change the code of the benchmark
  - Only when the above changes are not enough, then change the PL

Final suggestions for your work (2)

- Think about Milestone 0 to be the detailed description of
  your plan to parallelize your benchmark X
  - E.g., to parallelize X, we must understand these 2 pointers don’t alias
  - E.g., to parallelize X, we must understand this variable is an Induction Variable
  - E.g., to parallelize X, the compiler must be aware of the concept Y of the code
  - E.g., to parallelize X, we must prevent the programmer/compiler to premature lower this concept into code

- Think about Milestone 1 to be the minimum-valuable-product that demonstrates the goodness of your plan to parallelize your benchmark X
  - Highest return should be aimed first

- Think about Milestone 2 to be your final product

Your project

- This class requires paper reading and a project (397/497)

- You will work at the frontier of compiler research
  - Edges at the frontier are sharp
  - I’ll guide you through your project to avoid getting cut

- You will provide weekly update on your progress
  - Where: Zoom
  - When: please use the doodle link at my website
    to find a 30-minutes time slot

Select one benchmark from this set

- MiBench **Strongly suggested!**
  - sha, search, basicmath, djpeg, qsort, cjpeg, rawdaudio, toast, untoast, crc, rawcaudio
- PARSEC
  - x264, fluidanimate, bodytrack
- SPEC CPU2017 **Only if you have prior knowledge of AND experience using SPEC**
  - Anyone

Select one benchmark from this set

- Synchronize with others
  - Use Piazza to make sure you are the only one that will work on the benchmark you chose

- When every team has chosen the benchmark they will target,
  and no other team has chosen that benchmark,
  then send me an email declaring your benchmark
  - Deadline: Midnight (CST) of 4/6/2023 (next Thursday)

Always have faith in your ability

Success will come your way eventually

Best of luck!