CS440: Programming Languages and Translators

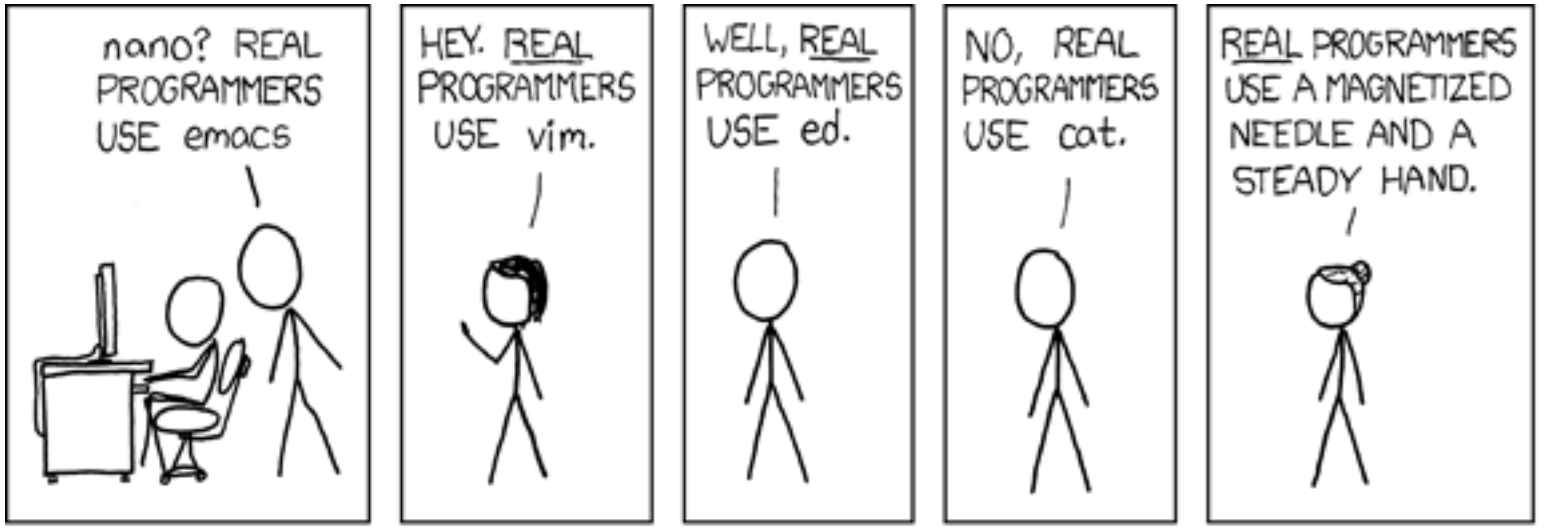
Lecture 0

Spring 2023

Outline

1. Programming Languages
2. Translators: Compilers and Interpreters
3. Types of Programming Languages
4. Syllabus

You can program without programming languages… if you really want



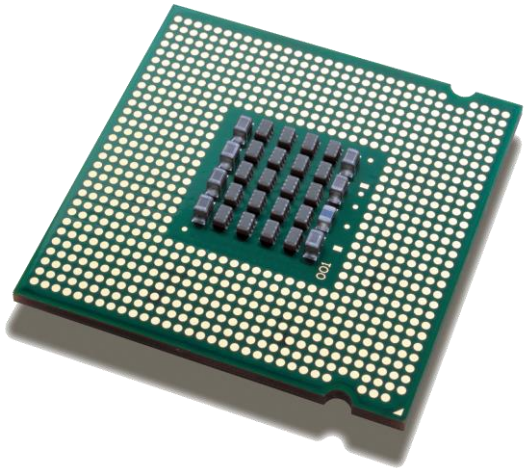
xkcd

Computer Architecture in One Slide

Memory
01100111

Program Counter

Op Code	Data 1	Data 2

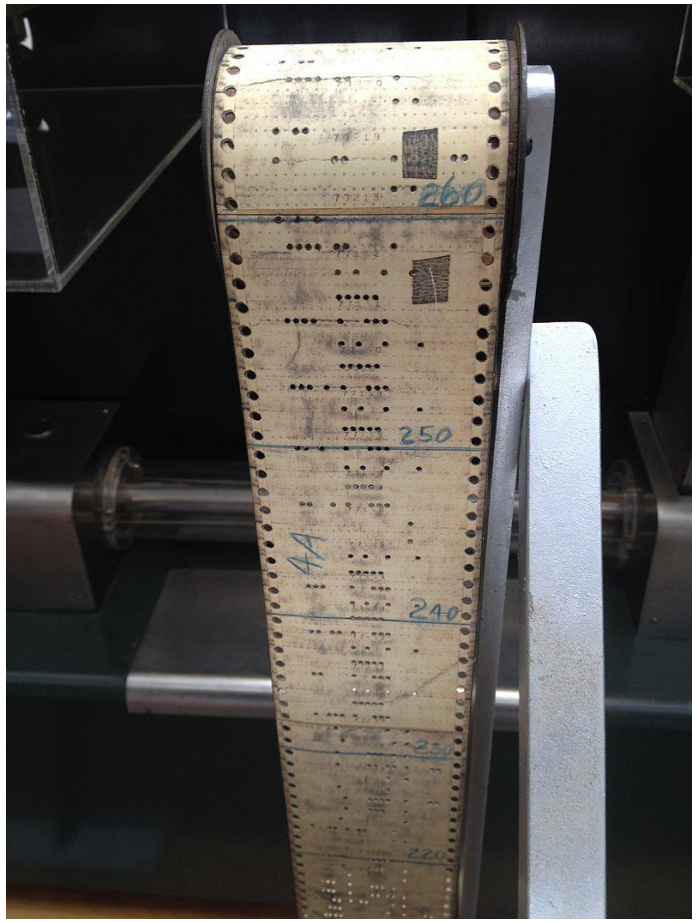


Registers
42
126
35
1
7
0
987
1024
2

You can program without programming languages… if you really want

Altair 8800
1974

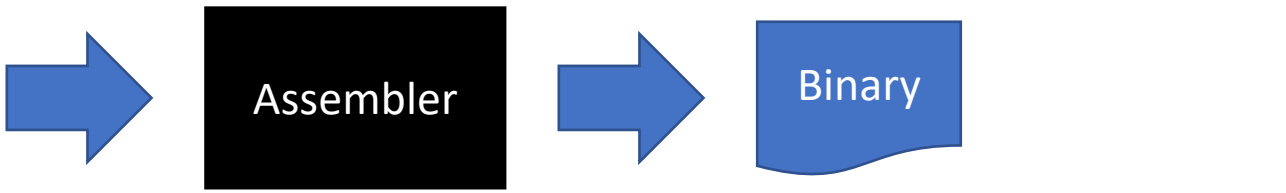
You can program without programming languages… if you really want



Instruction tape for Harvard Mark I
~1944

Assembly code makes instructions more human-readable

```
push   %rbp
mov    %rsp,%rbp
sub    $0x30,%rsp
mov    %rdi,-0x28(%rbp)
mov    %fs:0x28,%rax

mov    %rax,-0x8(%rbp)
xor    %eax,%eax
mov    -0x28(%rbp),%rax
mov    (%rax),%rax
mov    %rax,-0x10(%rbp)
cmpq   $0x0,-0x10(%rbp)
je     7c2 <MergeSort+0x88>
mov    -0x10(%rbp),%rax
mov    0x8(%rax),%rax
test   %rax,%rax
je     7c2 <MergeSort+0x88>
lea    -0x18(%rbp),%rdx
lea    -0x20(%rbp),%rcx
mov    -0x10(%rbp),%rax
mov    %rcx,%rsi
mov    %rax,%rdi
callq  877 <FrontBackSplit>
lea    -0x20(%rbp),%rax
mov    %rax,%rdi
callq  73a <MergeSort>
lea    -0x18(%rbp),%rax
mov    %rax,%rdi
callq  73a <MergeSort>
mov    -0x18(%rbp),%rdx
mov    -0x20(%rbp),%rax
mov    %rdx,%rsi
mov    %rax,%rdi
callq  7d9 <SortedMerge>
mov    %rax,%rdx
```



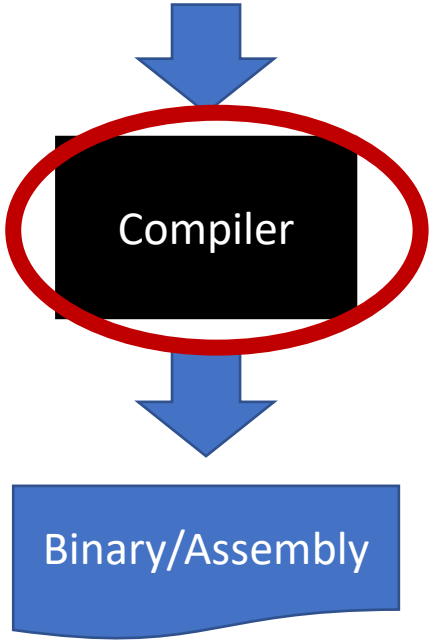
10101010100100010001001111001010100100010000011111010110000110000…

If we can turn text into binaries, why not easier-to-write text?

Rear Admiral Grace Hopper
(1906-1992)

COBOL
(1959)

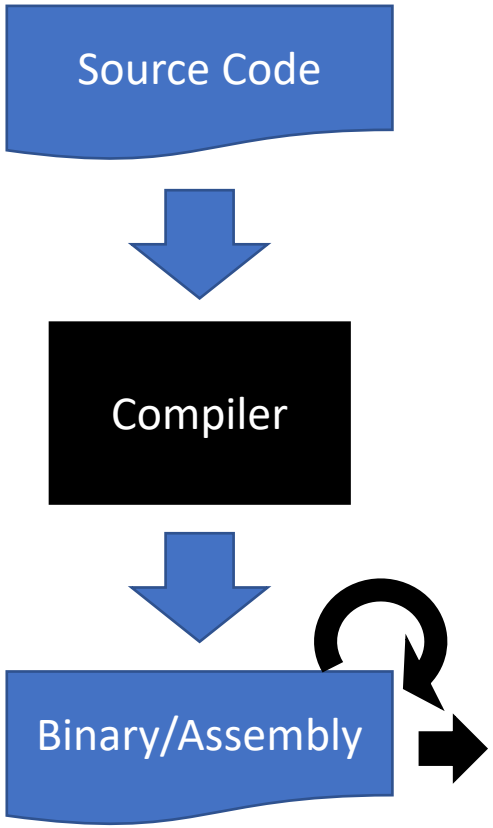
```
ADD 1 TO x
ADD 1, a, b TO x ROUNDED, y, z ROUNDED

ADD a, b TO c
    ON SIZE ERROR
        DISPLAY "Error"
END-ADD

ADD a TO b
    NOT SIZE ERROR
        DISPLAY "No error"
    ON SIZE ERROR
        DISPLAY "Error"
```



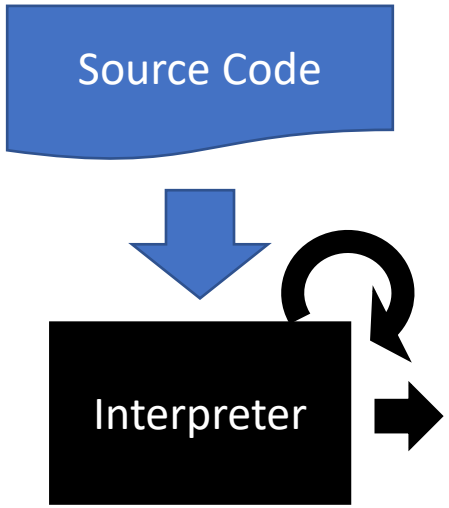
Outline

1. Programming Languages
2. Translators: Compilers and Interpreters
3. Types of Programming Languages
4. Syllabus

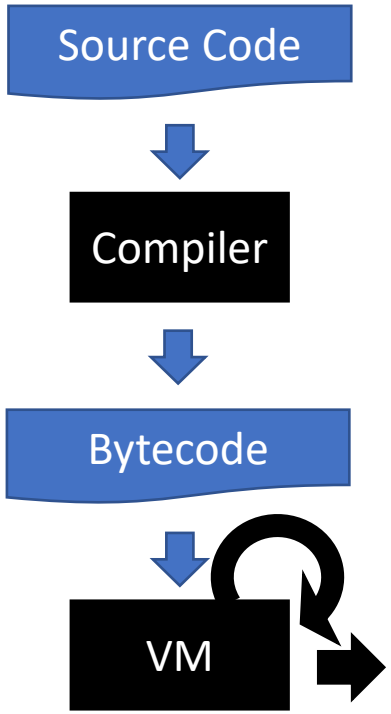
There are different ways of translating a programming language



Ex.: C, C++



Ex.: Python



Ex.: Java

Outline

1. Programming Languages
2. Translators: Compilers and Interpreters
3. Types of Programming Languages
4. Syllabus

All programming languages are the same… in a deep sense

“Turing completeness”

But the choice of language still matters in a very real sense—languages are tools!

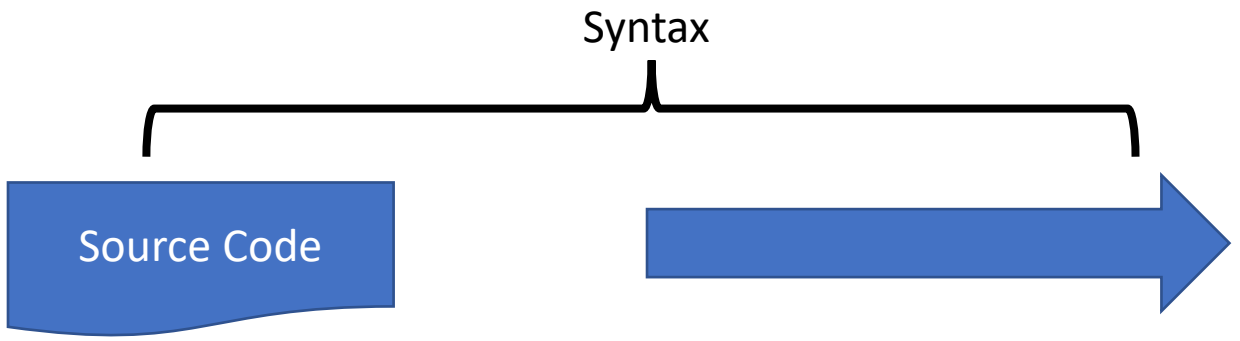
Programming Language =

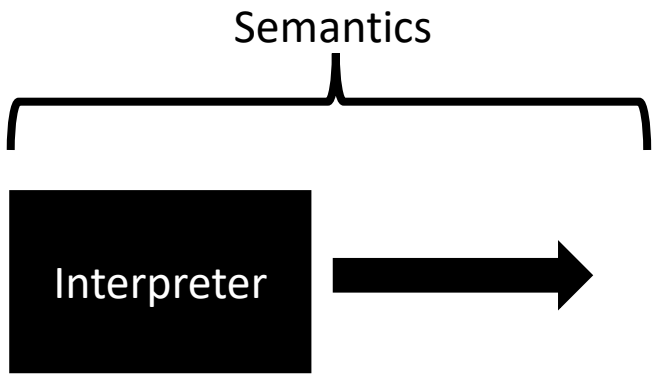
Syntax — What programs *look like*

+

Semantics — What programs *mean*

Syntax vs. semantics: Python



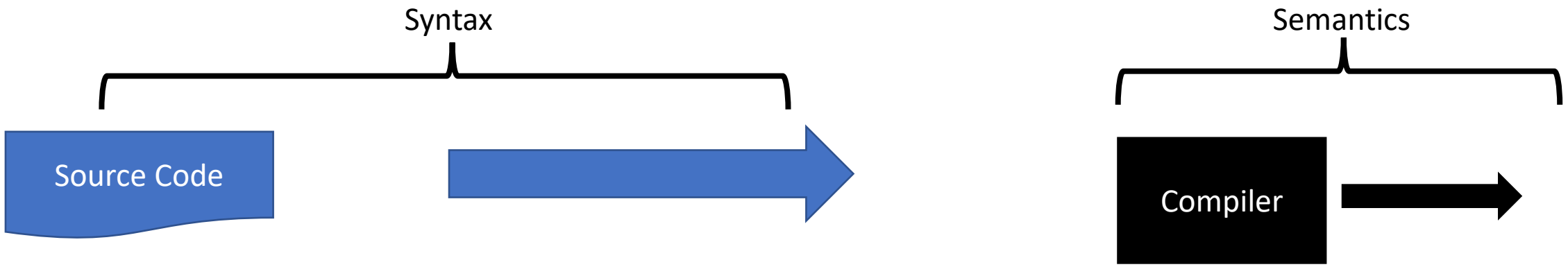



```
def func ():
  return 5 + “hello”
```

```
File "main.py", line 1
   def func ();
               ^
SyntaxError: invalid syntax
```

```
File "main.py", line 2, in func
    return 5 + "hello"
TypeError: unsupported operand
type(s) for +: 'int' and 'str'
```

Syntax vs. semantics: OCaml




```
let func () = 5 + “hello”
```

```
Line 5, characters 18-25:
Error: This expression has
type string but an
expression was expected of
type int
```

Semantics =

Static — Analyzed at compile time

+

Where do we check types?

Dynamic — Happens at run time

We can divide programming languages by whether they have *static* or *dynamic* types

- Static languages: types checked at compile time: *no type errors* at runtime
- Dynamic languages: types checked at run time, can have type errors
- (Weakly typed languages): types checked at compile time, but can be avoided, resulting in unexpected behavior or type errors at run time

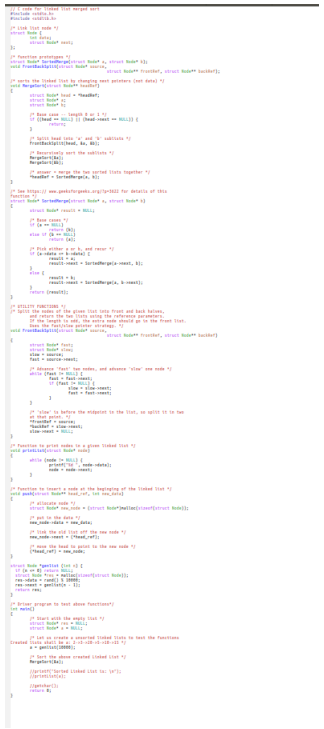
We can also divide programming languages based on *paradigm* (how you think about programming)

- Imperative: *tell computer what to do*
- Functional: *describe the computation mathematically*
- Object-oriented: *objects perform computation and carry data*
- Scripting
- Relational
- Domain-specific
- Logic

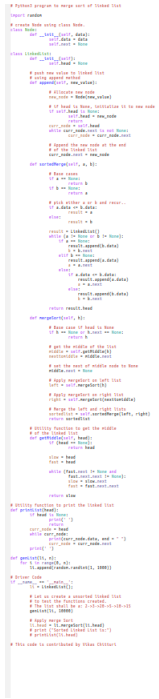
	Static	Dynamic
Imperative	Typescript, Pascal	Python, Javascript
Functional	Haskell, OCaml	Scheme, Racket
Object-oriented	C++, C#, Java	

Knowing the right paradigm to use can make programming easier

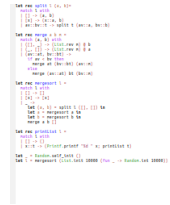
Task: Sort a linked list (using merge sort)



C

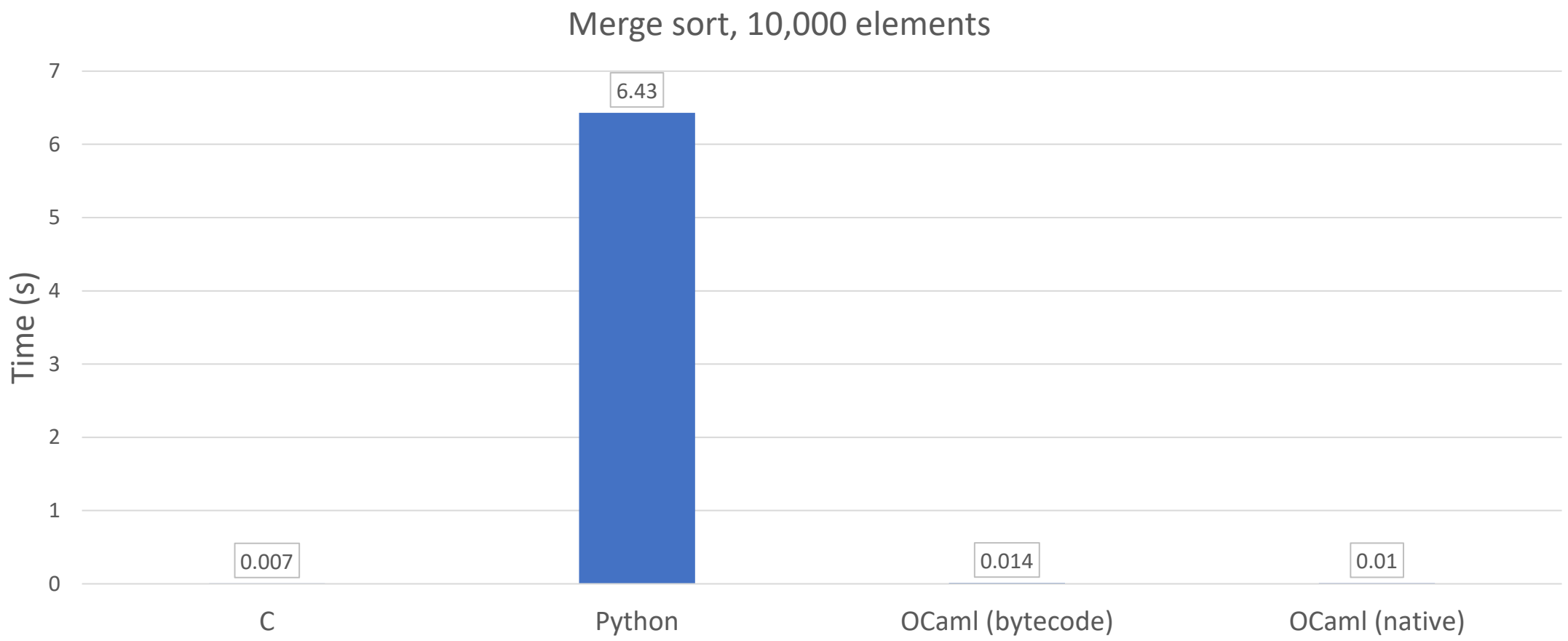


Python

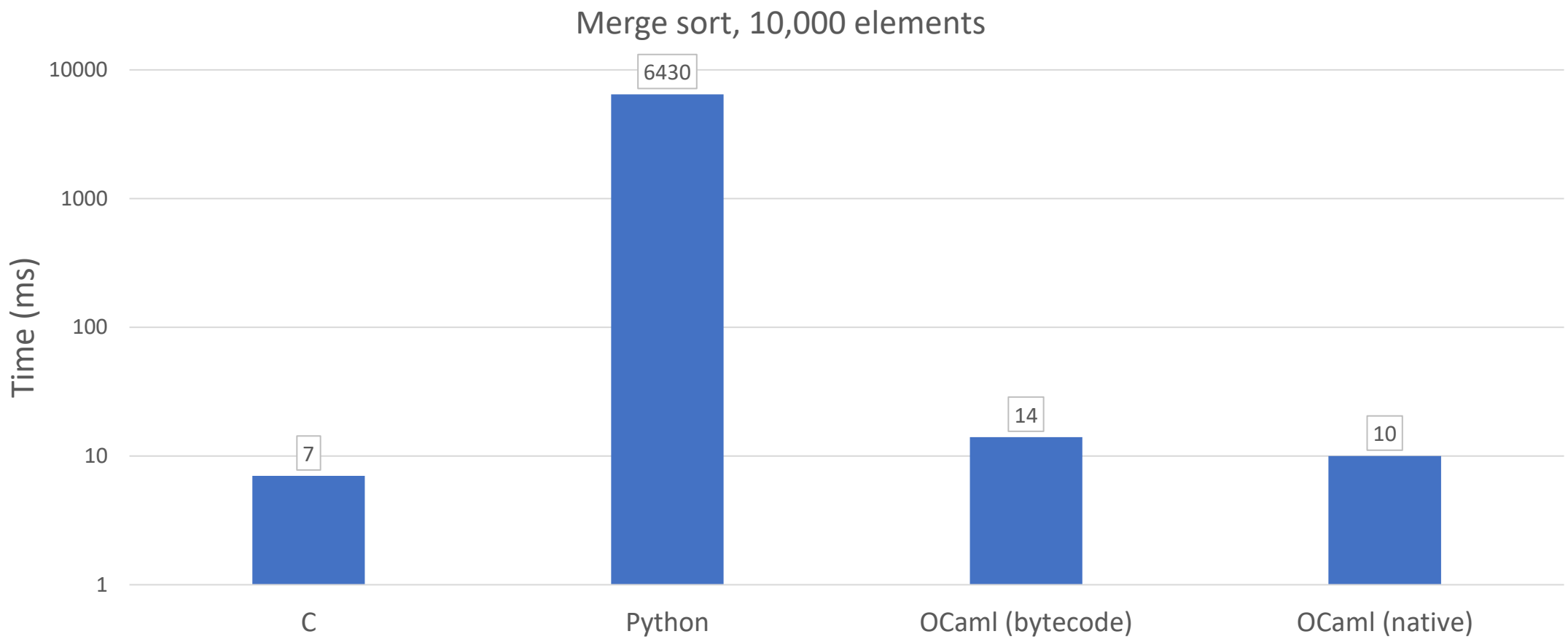


OCaml

Knowing about the language and how it's translated can help you write faster code



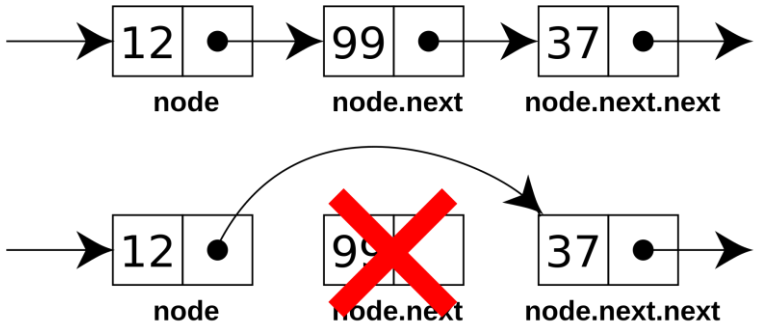
Knowing about the language and how it's translated can help you write faster code



Type systems can express different levels of guarantees

- C
```
node *mergesort(node *list)
```
  - Takes a pointer to a node and returns a pointer to a node.
- OCaml
```
mergesort : int list -> int list
```
  - Takes an integer list and returns an integer list.
- Haskell
```
mergesort :: IO ([int] -> [int])
```
  - Takes an integer list, returns an integer list and performs I/O (e.g., printing).
- Coq
```
mergesort : forall (l1 : list int), exists (l2: int list),
                    Sorted l2 /\ Permutation l1 l2
```
  - Takes an integer list and returns a sorted permutation of it.

Different languages are up to different tasks



OCaml

OCaml ?

C?

Rust?

Outline

1. Programming Languages
2. Translators: Compilers and Interpreters
3. Types of Programming Languages
4. Syllabus

Course Goals

- Learn to evaluate and discuss programming languages
  - Learn the lingo (impressing people with jargon isn't the point, but is a side effect)

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

- Reason precisely about what programs mean
  - Inject mathematical rigor into programming

- Become a *creator* of PLs, not just a *consumer*

Sections/Attendance

- Section 01: In-person
  - Attendance not recorded, but attendance/participation may be used to "break ties"
- Section 02: Online
  - Lecture videos will be posted to Blackboard after each lecture

Course Staff

- Instructor: Stefan Muller
  - Office Hours: Mon., 11am-12pm (Online – link to come)
    Thur., 2-3pm (SB 218E)

- TA: Xincheng Yang
  - Office Hours: Tue., 2-4pm (Online – appt. link to come)
    Wed., 2-4pm (SB 004)

Course Website: http://cs.iit.edu/~cs440/
Important info, notes, etc.

CS440: Programming Languages and Translators, Spring 2023

Instructor: Stefan Muller, smuller2@iit.edu
Office Hours: TBA
TA: TBA
Lectures:

Section 01: Tue, Thur 10:00-11:15 AM, SB 104
Section 02: Online only

Schedule Resources Policies

Schedule

Note: this schedule is tentative and subject to change.

For the readings posted:
"PDB" = "Purple Dragon Book" (Aho et al.)
"FPO" = "Functional Programming in OCaml" (linked below)
"TAPL" = *Types and Programming Languages* (Pierce)
"PFPL" = *Practical Foundations for Programming Languages* (Harper)

		Topic		Readings	Notes
January	10	Intro	Languages and course overview		
	12		Compiler structure, interpreters, OCaml		
	17	OCaml	OCaml evaluation, types, expressions		
	19		Functions and recursion		
	24		More on types		
	26		Lists and tail recursion		
	31		Records and algebraic data types		
February	2		Higher-order functions		
	7		Higher-order functions		
	9		Side effects		
	14	Interpreters	Building an interpreter		
	16		Closures		

Other ways to get help

- Discord: IIT CS server, #cs440 channel
  - If you're not on it, we'll send an invitation

- Academic Resource Center (ARC): www.iit.edu/arc
  - FREE subject matter tutoring and academic coaching

	Discord	**Office Hours**	**Email**	**ARC**
General questions about lectures, logistics, etc.	✓	✓		
General discussion, clarifications, about HW questions	✓	✓		
Specific questions about your HW answers		✓		✓
More in-depth personal tutoring				✓
Personal matters (accommodations, other requests, etc.)			✓	

Collaboration and Academic Honesty

- Discussing general concepts is encouraged
- Discussing broad strategies for doing lab tasks is OK – don't discuss actual answers or code
  - If, after your discussion, you don't take any notes/pictures and write up your code/solutions by yourself, you're probably OK
  - Cite collaborators and any other resources in your write-up
- Not allowed:
  - Working together
  - Sharing answers
  - Looking for answers on the internet

This is the short version: read the details on the course website

(Tentative) Schedule

- Intro (1 week – you are here)
- Learn OCaml (~4 weeks)
- Interpreters (~2 weeks)
- *Midterm*
- Type checking (~2 weeks)
- *Spring break*
- Formal semantics (~2 weeks)
- Formal type systems (~2 weeks)
- Other topics and wrap-up (~3 weeks)

Labs/Projects/Homeworks/Problem sets

- 6-7 homeworks, ~2 weeks each
  - Lab 0 Out ~Thursday, Due 1/26
- Written and programming
- Work individually

Late Days:

- 7 per student, extend deadline 24 hours
- No more than 2 per assignment
- If no more late days, 10% late penalty per day
- No work accepted >48 hours late

Exams

- Midterm (tentatively Mar. 2)
- Final (finals week)

- Details TBA

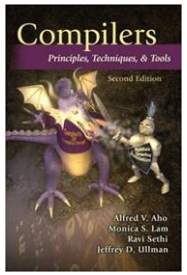
- (No using late days, sorry)

Grading

- 50% Homeworks
- 20% Midterm
- 30% Final

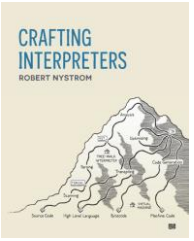
>=90	A
80-90	B
70-80	C
60-70	D
<60	E

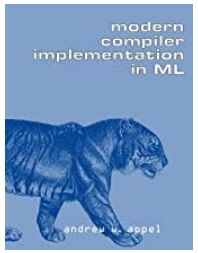
Textbooks



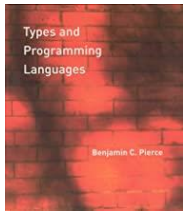
On Compilers/interpreters:

- "Purple Dragon Book": Aho et al. *Compilers: Principles, Techniques and Tools (2nd ed.)*
- Appel. *Modern Compiler Implementation in ML*
- Nystrom. *Crafting Interpreters*





For more math-y details:



- Pierce. *Types and Programming Languages*
- Harper. *Practical Foundations for Programming Languages*

For Thursday: Bring laptops if you can!