Course Introduction

Rust, in Practice and in Theory
Lecture 1

CAS CS 392 (M1)

Outline

- Discuss the expectations of this course
- Look at what this course is about
- **Workshop:** Install Rust
- **If you finish:** Set up a Cargo project

Course Information

Minutiae

Instructor: Nathan Mull

Course Webpage: https://nmmull.github.io/CS392-S25/index.html

Midterm Date: March 6

Grade Breakdown

30%	Assignments
40%	Final Project (4 parts, 10% each)
20%	Midterm Exam (in class)
10%	Participation

Disclaimer

Disclaimer

This is a new course. It's going to be a bit **disorganized.** I appreciate your patience

Disclaimer

This is a new course. It's going to be a bit **disorganized**. I appreciate your patience

I am *not* an expert of Rust. I'm learning a lot myself. But I'm an expert in **type theory**. My goal is to show you all how *I* learn PLs

Lectures

Lectures

This course will be something like a flipped classroom/workshop hybrid. You'll be expected to read *before* lecture and complete a short pre-lecture quiz

Lectures

This course will be something like a flipped classroom/workshop hybrid. You'll be expected to read *before* lecture and complete a short pre-lecture quiz

I ***will*** take attendance. You're allowed to miss 2-3 lectures

Lectures

This course will be something like a flipped classroom/workshop hybrid. You'll be expected to read *before* lecture and complete a short pre-lecture quiz

I ***will*** take attendance. You're allowed to miss 2-3 lectures

We'll spend the first part of the lecture reviewing the material you read about, and then we'll go into a workshop/lab-style meeting during which you'll work on the homework assignments or final projects or other in-class tasks

Lectures

This course will be something like a flipped classroom/workshop hybrid. You'll be expected to read *before* lecture and complete a short pre-lecture quiz

I ***will*** take attendance. You're allowed to miss 2-3 lectures

We'll spend the first part of the lecture reviewing the material you read about, and then we'll go into a workshop/lab-style meeting during which you'll work on the homework assignments or final projects or other in-class tasks

I want this course to be very collaborative. I'll expect that you're working in groups, pair/group programming, and one-on-one discussions with me and the other students

Assignments

Assignments

Assignments will consist of either programming exercises or larger programming tasks in Rust. We may have 1-2 written assignments

Assignments

Assignments will consist of either programming exercises or larger programming tasks in Rust. We may have 1–2 written assignments

There are 8 total, I'll drop your lowest two

Assignments

Assignments will consist of either programming exercises or larger programming tasks in Rust. We may have 1-2 written assignments

There are 8 total, I'll drop your lowest two

Even if you pair/group program a problem, try to type your own solution and cite who you worked with

Final Project

Final Project

The final project is an implementation of a subset of Rust in Rust

Final Project

The final project is an implementation of a subset of Rust in Rust

This will take up most of the second half of the course

Final Project

The final project is an implementation of a subset of Rust in Rust

This will take up most of the second half of the course

It has four parts: (1) parser, (2) evaluator, (3) type/borrow checker, (4) extension of your choosing (you cannot drop any part)

Final Project

The final project is an implementation of a subset of Rust in Rust

This will take up most of the second half of the course

It has four parts: (1) parser, (2) evaluator, (3) type/borrow checker, (4) extension of your choosing (you cannot drop any part)

By the end of the course, you should have a working interpreter

Participation

The participation part of the grade is made up of:

» Attendance

» Pre-lecture quizzes

» Participation in class discussion and online

Grading

Grading

You'll submit assignments via Gradescope, but there will be no autograders (or only very simple autograders)

Grading

You'll submit assignments via Gradescope, but there will be no autograders (or only very simple autograders)

I'll be looking at most of the code by hand and commenting on it (it's a small course and I want to see your code)

Grading

You'll submit assignments via Gradescope, but there will be no autograders (or only very simple autograders)

I'll be looking at most of the code by hand and commenting on it (it's a small course and I want to see your code)

(If you convince me you've learned something, you'll almost certainly get an A in the course)

Course Communication

Course Communication

We have a Piazza page for the course, please keep an eye on it for course-related updates

Course Communication

We have a Piazza page for the course, please keep an eye on it for course-related updates

I won't respond to emails regarding material, but you can email me if you have logistical questions

Course Webpage

https://nmmull.github.io/CS392-S25/index.html

The course webpage will have all material, links to reading, etcetera

Make sure to check it frequently

Questions?

If I miss anything, ask on Piazza

Remember: This is a *small, experimental* course. If you have suggestions on the course, please let me know

By continuing in this course you're agreeing to all these conditions

Last thing: What's your name?

Name:

Year:

Interest in CS:

Interest outside of CS:

Take <u>1 minute</u> to think about it, and then we'll go around the room

What is this course?

The Idea

If you took CS320 last semester, then the concept should be familiar:

1. **Learn Rust** (a somewhat difficult PL)

2. **Implement Rust** (in Rust, to check our understanding)

A Word of Warning

A Word of Warning

`This is` *`not`* `an introductory programming course`

A Word of Warning

This is *not* an introductory programming course

Some things will move very fast (I'll assume you'll be able to write simple programs within the first week)

A Word of Warning

This is *not* an introductory programming course

Some things will move very fast (I'll assume you'll be able to write simple programs within the first week)

Some things will move very slow (We'll dwell a bit on things like memory management)

What's Rust?

What's Rust?

```
Rust is a type-safe memory-safe PL
```

What's Rust?

Rust is a type-safe memory-safe PL

It's possible to write simple clean code that's *guaranteed* to be free of memory bugs

What's Rust?

Rust is a type-safe memory-safe PL

It's possible to write simple clean code that's *guaranteed* to be free of memory bugs

It's an **alternative to C or C++** which can be used in production settings for rapid development without fear of crashes or memory leaks

About Rust

Developed by Graydon Hoare out of Mozilla in the 2000s (originally implemented in OCaml)

It became stable (in particular with its type system) in the late 2010s

The Rust Foundation was started in 2021 and is the basis for Rust information and adoption today

It's community members are called **Rustacians**, which is the basis for the unofficial mascot **Ferris** the crab

Mozilla

Why Rust?

Why Rust?

1. Rust is *weird*

Why Rust?

1. Rust is *weird*

   It uses a unique type system to achieve its memory safety, which programmers often have to wrestle with

Why Rust?

1. Rust is *weird*

   It uses a unique type system to achieve its memory safety, which programmers often have to wrestle with

2. Rust is becoming *popular*

Why Rust?

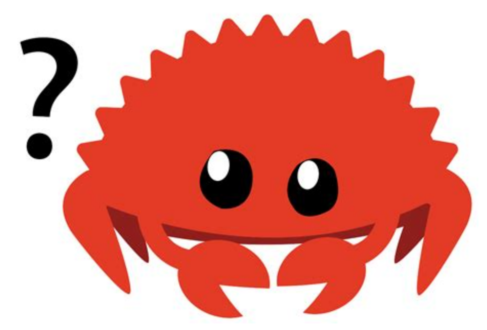
1. Rust is *weird*

   It uses a unique type system to achieve its memory safety, which programmers often have to wrestle with

2. Rust is becoming *popular*

   Firefox, Dropbox, Yelp, Amazon (along with lots of others) are all adopting Rust in large-scale projects

Example: Rust being Weird



```c
void swap(char **x, char **y) {
  char *z = *x;
  *x = *y;
  *y = z;
}
```

C

```rust
fn swap(x : &mut String, y : &mut String) {
  let z : String = *x;
  *x = *y;
  *y = z;
}
```

Rust

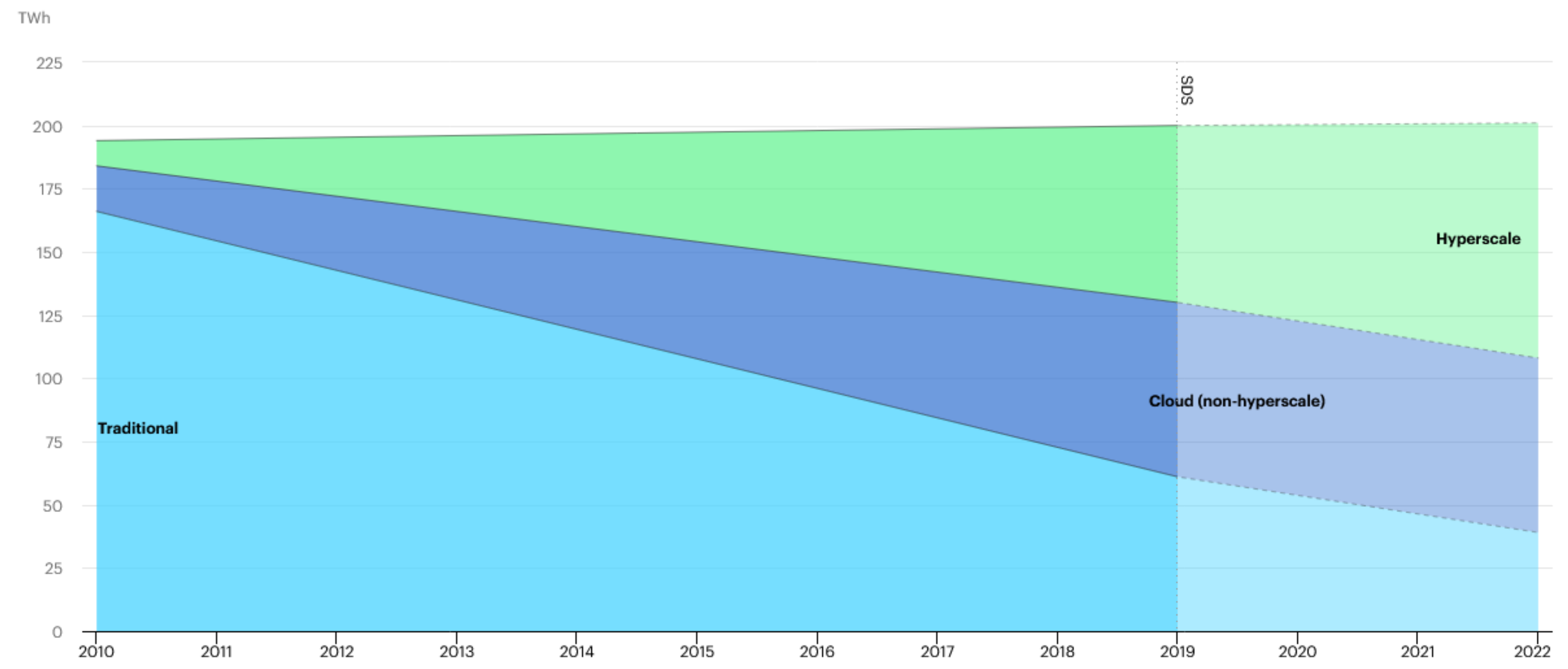
Rust has a notion of references, but it's *not possible* to write the swap-string-pointer function

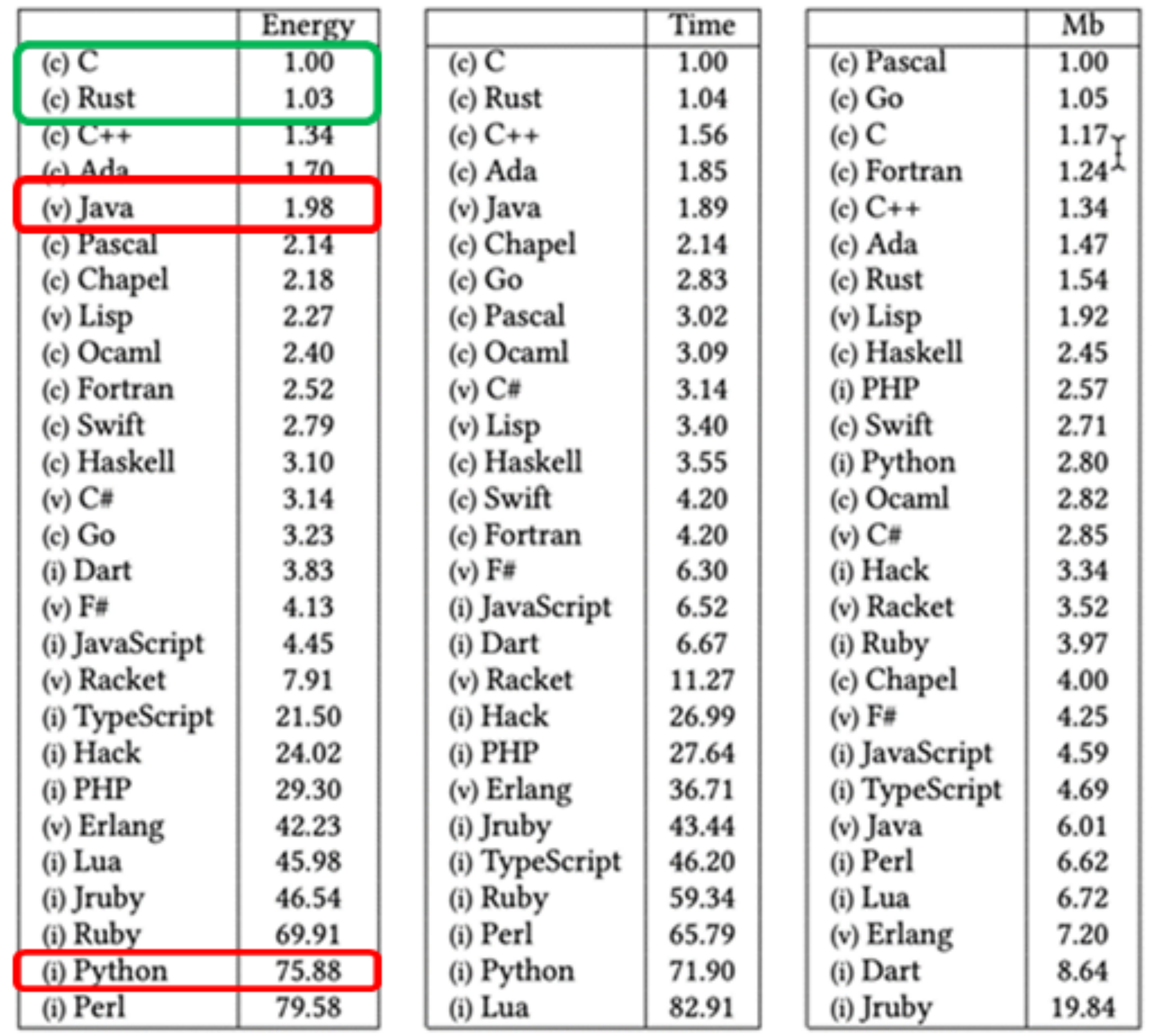
A badly defined pointer-swap could cause a memory leak. Rust's type system disallows this *by fiat*

Example: Sustainability with Rust

An interesting (slightly dated) article out of AWS

Rust is performant, energy efficient *and a whole lot more interesting than many other options*





	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

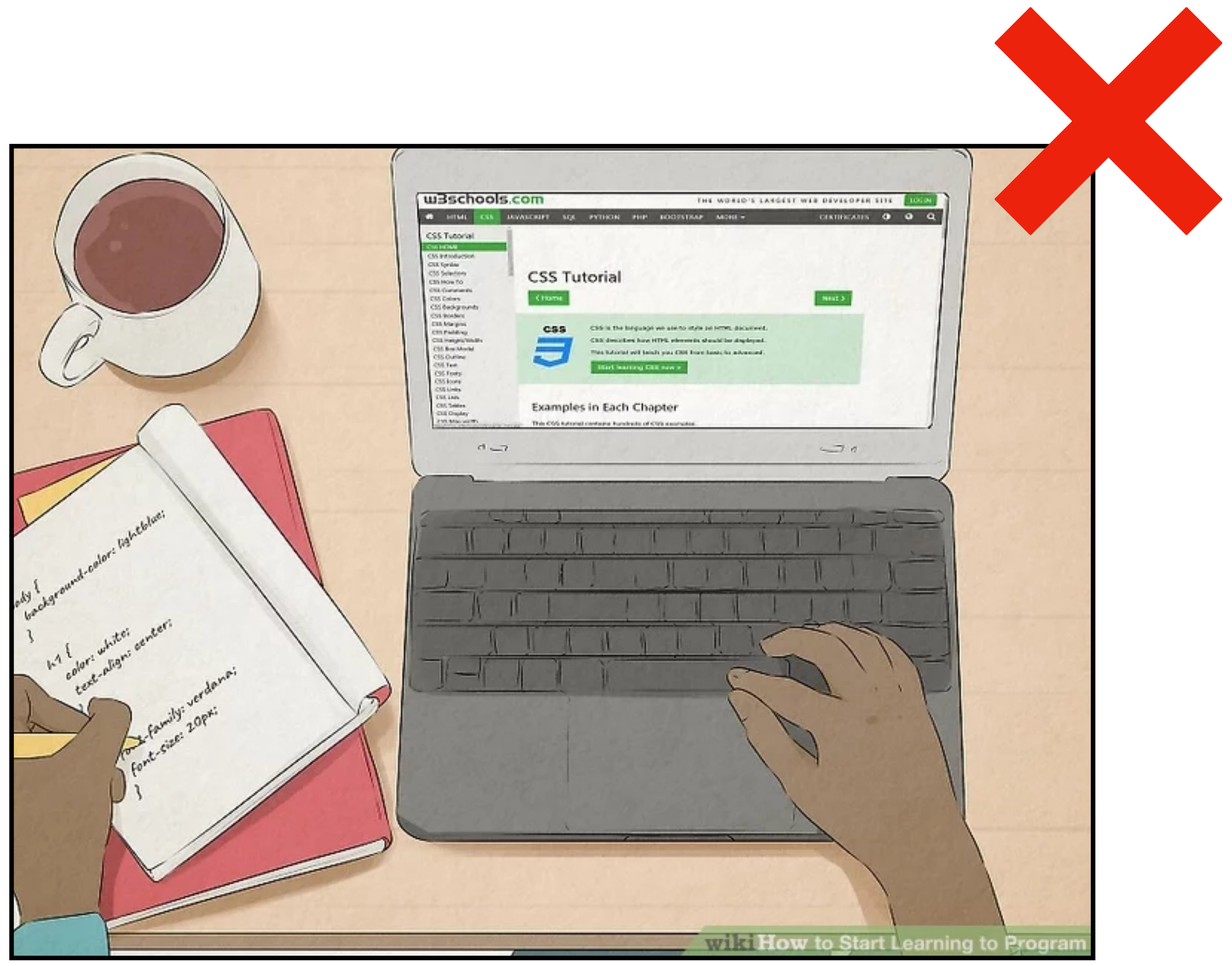
	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

https://aws.amazon.com/cn/blogs/opensource/sustainability-with-rust/

Aside: How to learn a PL

Aside: How to learn a PL

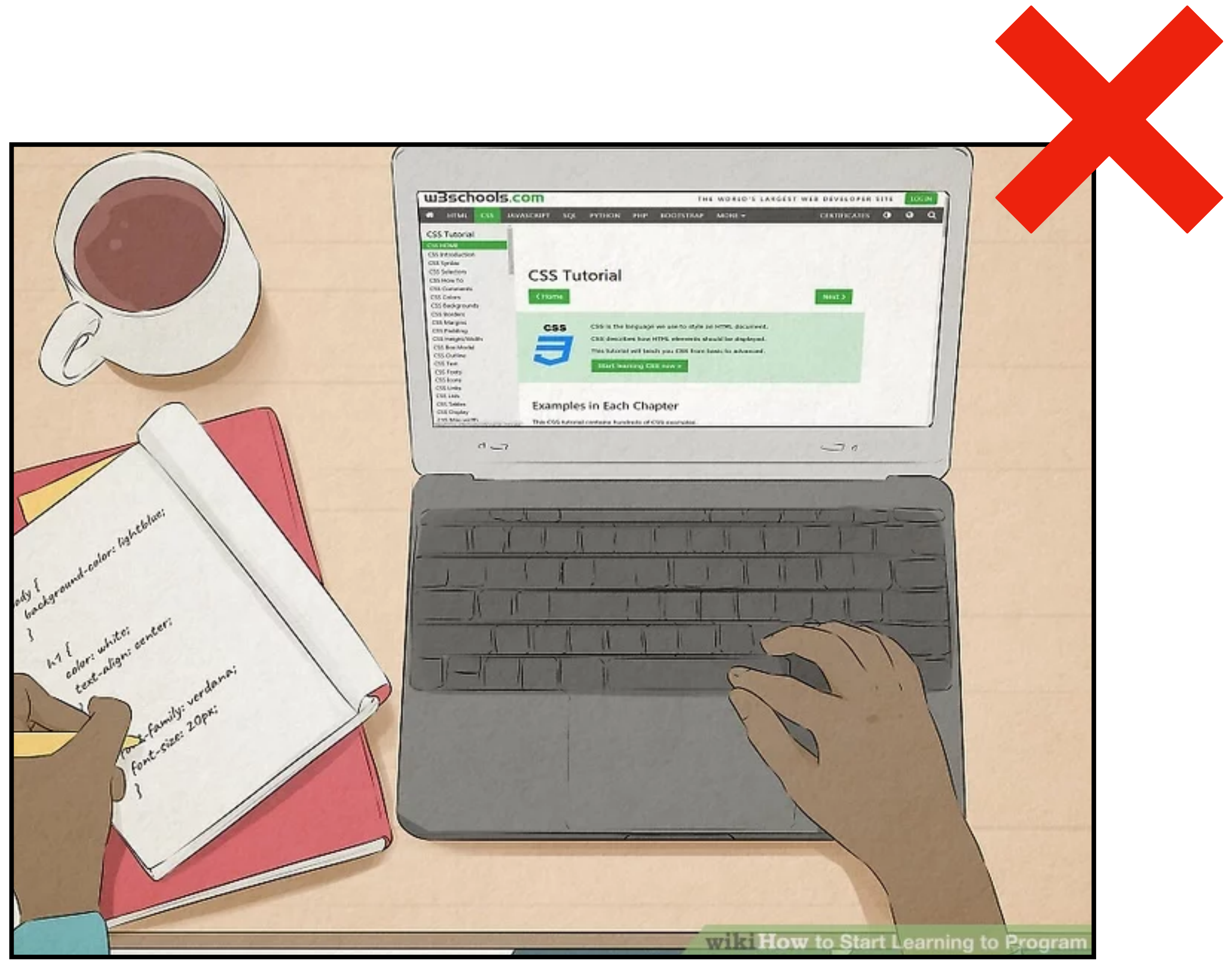
We tend to learn PLs the "wrong" (fast) way, i.e., reading tutorials and doing examples

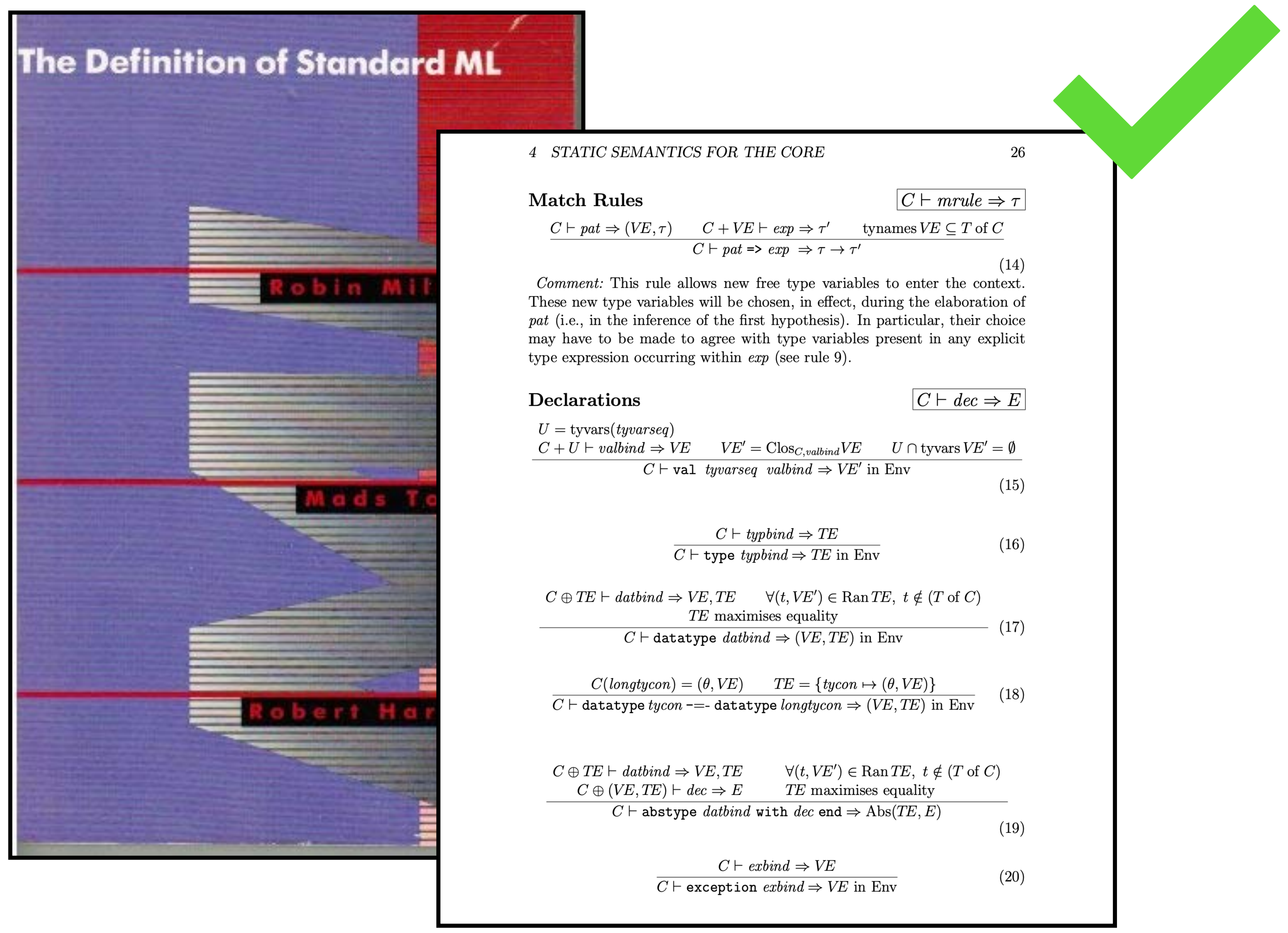


Aside: How to learn a PL

We tend to learn PLs the "wrong" (fast) way, i.e., reading tutorials and doing examples

In this course we want to learn Rust the "right" (slow) way, i.e., formally describe what Rust is doing




The Definition of Standard ML
Robin Mil
Mads To
Robert Har

4 STATIC SEMANTICS FOR THE CORE 26

Match Rules $\boxed{C \vdash mrule \Rightarrow \tau}$

$$\frac{C \vdash pat \Rightarrow (VE, \tau) \qquad C + VE \vdash exp \Rightarrow \tau' \qquad \text{tynames}\, VE \subseteq T \text{ of } C}{C \vdash pat \texttt{ => } exp \Rightarrow \tau \to \tau'} \quad (14)$$

Comment: This rule allows new free type variables to enter the context. These new type variables will be chosen, in effect, during the elaboration of *pat* (i.e., in the inference of the first hypothesis). In particular, their choice may have to be made to agree with type variables present in any explicit type expression occurring within *exp* (see rule 9).

Declarations $\boxed{C \vdash dec \Rightarrow E}$

$$\frac{\begin{array}{l} U = \text{tyvars}(tyvarseq) \\ C + U \vdash valbind \Rightarrow VE \qquad VE' = \text{Clos}_{C,valbind} VE \qquad U \cap \text{tyvars}\, VE' = \emptyset \end{array}}{C \vdash \texttt{val}\ tyvarseq\ valbind \Rightarrow VE' \text{ in Env}} \quad (15)$$

$$\frac{C \vdash typbind \Rightarrow TE}{C \vdash \texttt{type}\ typbind \Rightarrow TE \text{ in Env}} \quad (16)$$

$$\frac{\begin{array}{c} C \oplus TE \vdash datbind \Rightarrow VE, TE \qquad \forall (t, VE') \in \text{Ran}\, TE,\ t \notin (T \text{ of } C) \\ TE \text{ maximises equality} \end{array}}{C \vdash \texttt{datatype}\ datbind \Rightarrow (VE, TE) \text{ in Env}} \quad (17)$$

$$\frac{C(longtycon) = (\theta, VE) \qquad TE = \{tycon \mapsto (\theta, VE)\}}{C \vdash \texttt{datatype}\ tycon\ \texttt{-=-}\ \texttt{datatype}\ longtycon \Rightarrow (VE, TE) \text{ in Env}} \quad (18)$$

$$\frac{\begin{array}{ll} C \oplus TE \vdash datbind \Rightarrow VE, TE & \forall (t, VE') \in \text{Ran}\, TE,\ t \notin (T \text{ of } C) \\ C \oplus (VE, TE) \vdash dec \Rightarrow E & TE \text{ maximises equality} \end{array}}{C \vdash \texttt{abstype}\ datbind\ \texttt{with}\ dec\ \texttt{end} \Rightarrow \text{Abs}(TE, E)} \quad (19)$$

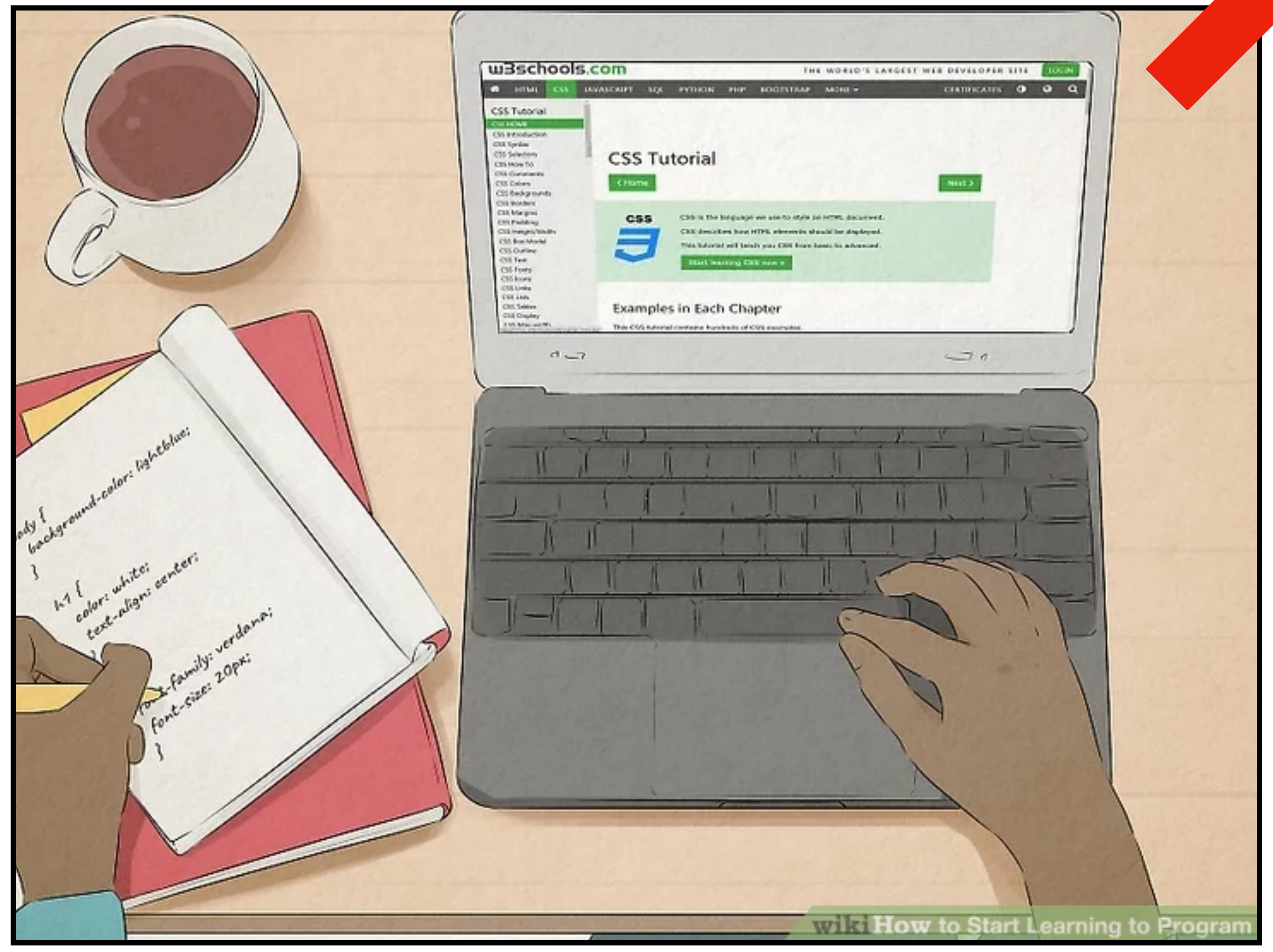
$$\frac{C \vdash exbind \Rightarrow VE}{C \vdash \texttt{exception}\ exbind \Rightarrow VE \text{ in Env}} \quad (20)$$

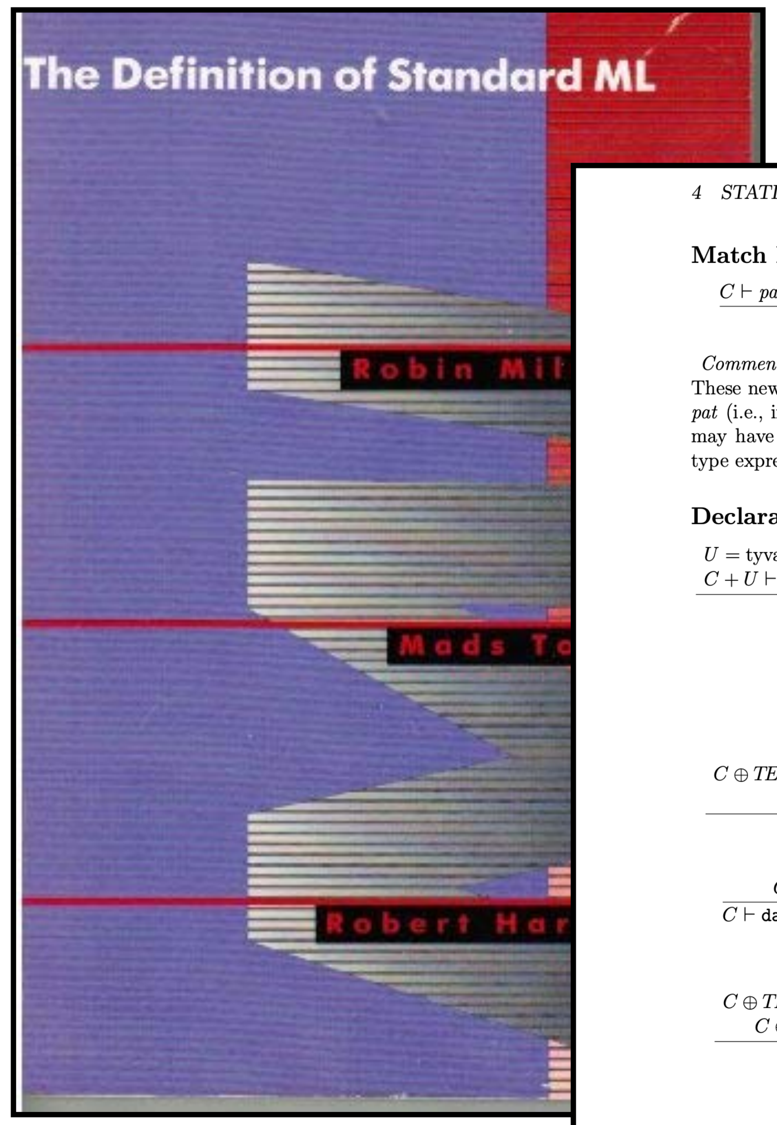
Aside: How to learn a PL

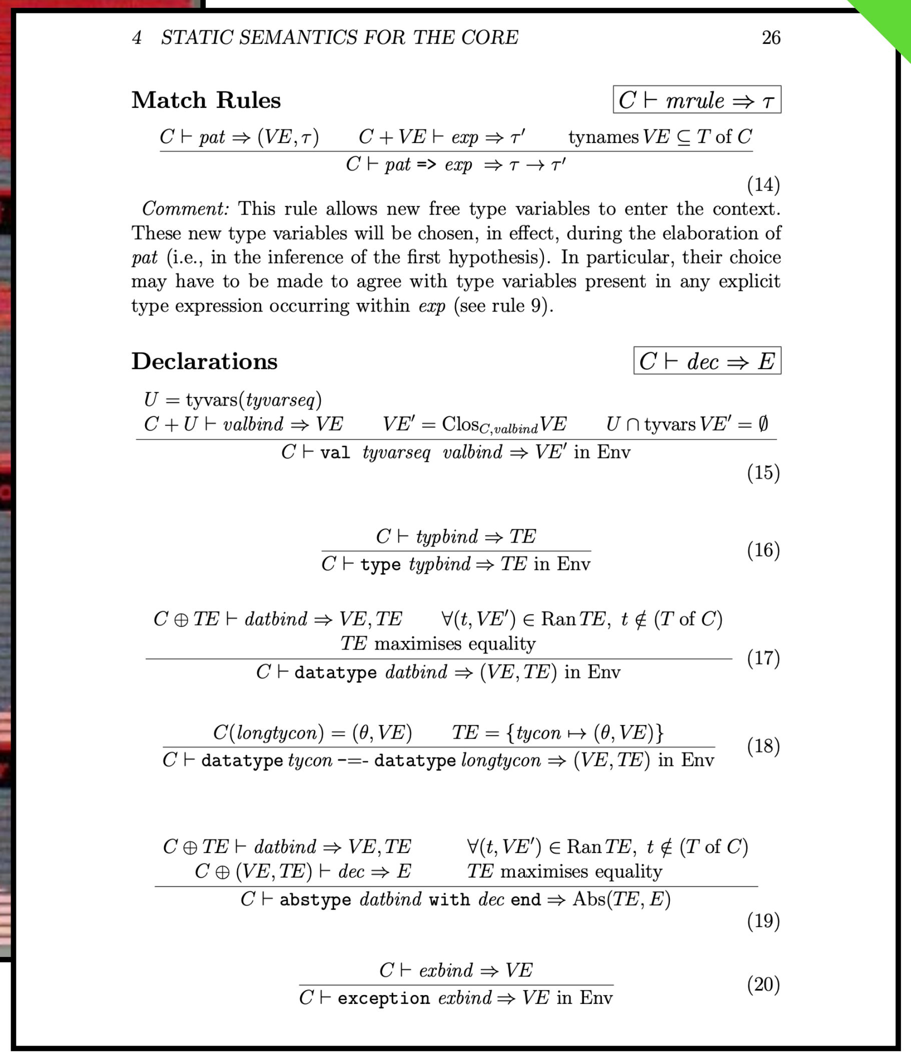
We tend to learn PLs the "wrong" (fast) way, i.e., reading tutorials and doing examples

In this course we want to learn Rust the "right" (slow) way, i.e., formally describe what Rust is doing

We *won't* learn many cool, advanced features of Rust that are useful in practice






4 STATIC SEMANTICS FOR THE CORE 26

Match Rules $\boxed{C \vdash mrule \Rightarrow \tau}$

$$\frac{C \vdash pat \Rightarrow (VE,\tau) \qquad C + VE \vdash exp \Rightarrow \tau' \qquad \text{tynames}\, VE \subseteq T \text{ of } C}{C \vdash pat \;\texttt{=>}\; exp \Rightarrow \tau \to \tau'} \tag{14}$$

Comment: This rule allows new free type variables to enter the context. These new type variables will be chosen, in effect, during the elaboration of *pat* (i.e., in the inference of the first hypothesis). In particular, their choice may have to be made to agree with type variables present in any explicit type expression occurring within *exp* (see rule 9).

Declarations $\boxed{C \vdash dec \Rightarrow E}$

$$\frac{\begin{array}{l} U = \text{tyvars}(tyvarseq) \\ C + U \vdash valbind \Rightarrow VE \qquad VE' = \text{Clos}_{C,valbind} VE \qquad U \cap \text{tyvars}\, VE' = \emptyset \end{array}}{C \vdash \texttt{val}\; tyvarseq\; valbind \Rightarrow VE' \text{ in Env}} \tag{15}$$

$$\frac{C \vdash typbind \Rightarrow TE}{C \vdash \texttt{type}\; typbind \Rightarrow TE \text{ in Env}} \tag{16}$$

$$\frac{\begin{array}{c} C \oplus TE \vdash datbind \Rightarrow VE, TE \qquad \forall (t, VE') \in \text{Ran}\, TE,\ t \notin (T \text{ of } C) \\ TE \text{ maximises equality} \end{array}}{C \vdash \texttt{datatype}\; datbind \Rightarrow (VE, TE) \text{ in Env}} \tag{17}$$

$$\frac{C(longtycon) = (\theta, VE) \qquad TE = \{tycon \mapsto (\theta, VE)\}}{C \vdash \texttt{datatype}\; tycon \;\texttt{-=-}\; \texttt{datatype}\; longtycon \Rightarrow (VE, TE) \text{ in Env}} \tag{18}$$

$$\frac{\begin{array}{ll} C \oplus TE \vdash datbind \Rightarrow VE, TE & \forall (t, VE') \in \text{Ran}\, TE,\ t \notin (T \text{ of } C) \\ C \oplus (VE, TE) \vdash dec \Rightarrow E & TE \text{ maximises equality} \end{array}}{C \vdash \texttt{abstype}\; datbind\; \texttt{with}\; dec\; \texttt{end} \Rightarrow \text{Abs}(TE, E)} \tag{19}$$

$$\frac{C \vdash exbind \Rightarrow VE}{C \vdash \texttt{exception}\; exbind \Rightarrow VE \text{ in Env}} \tag{20}$$

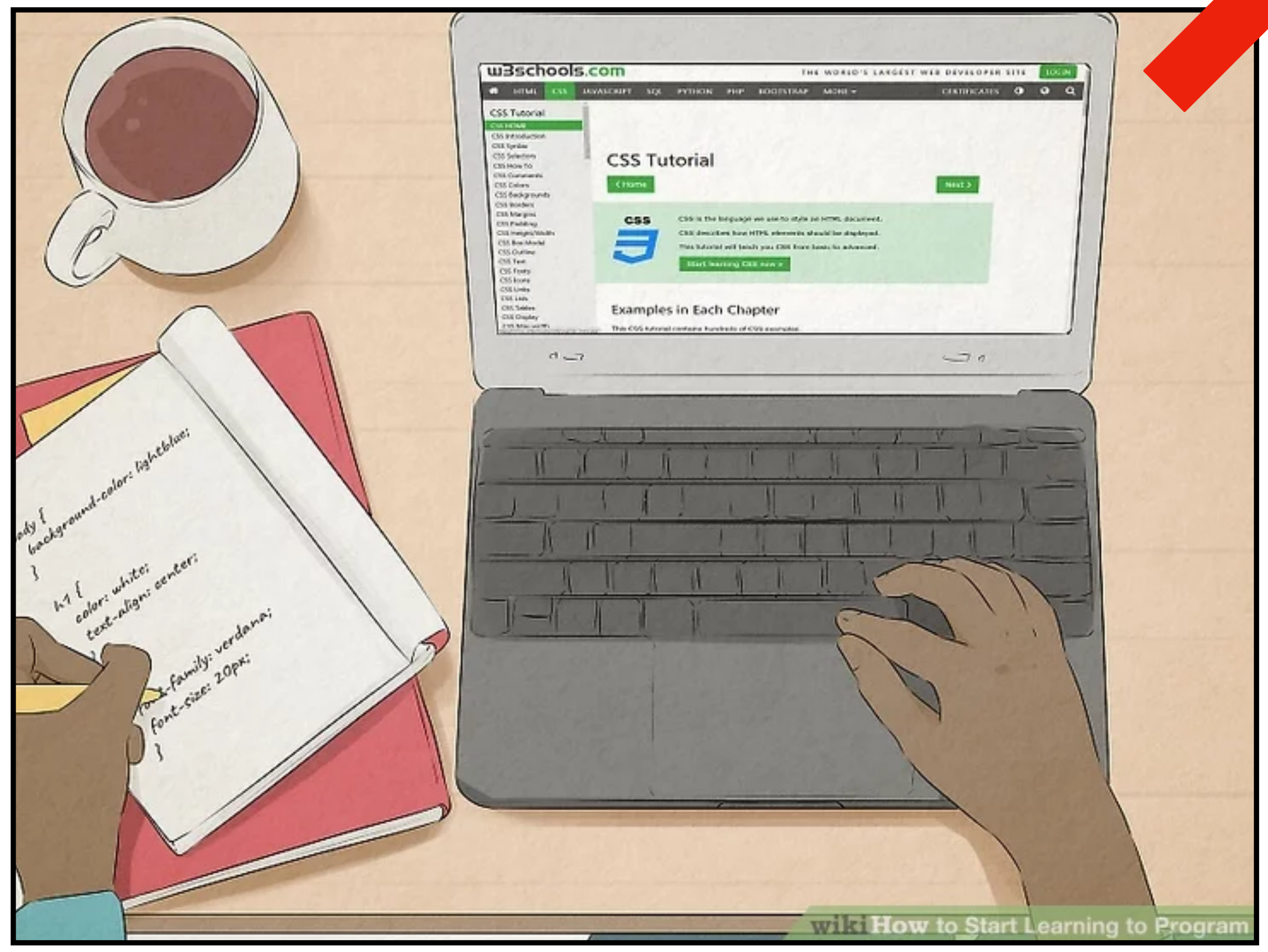
Aside: How to learn a PL

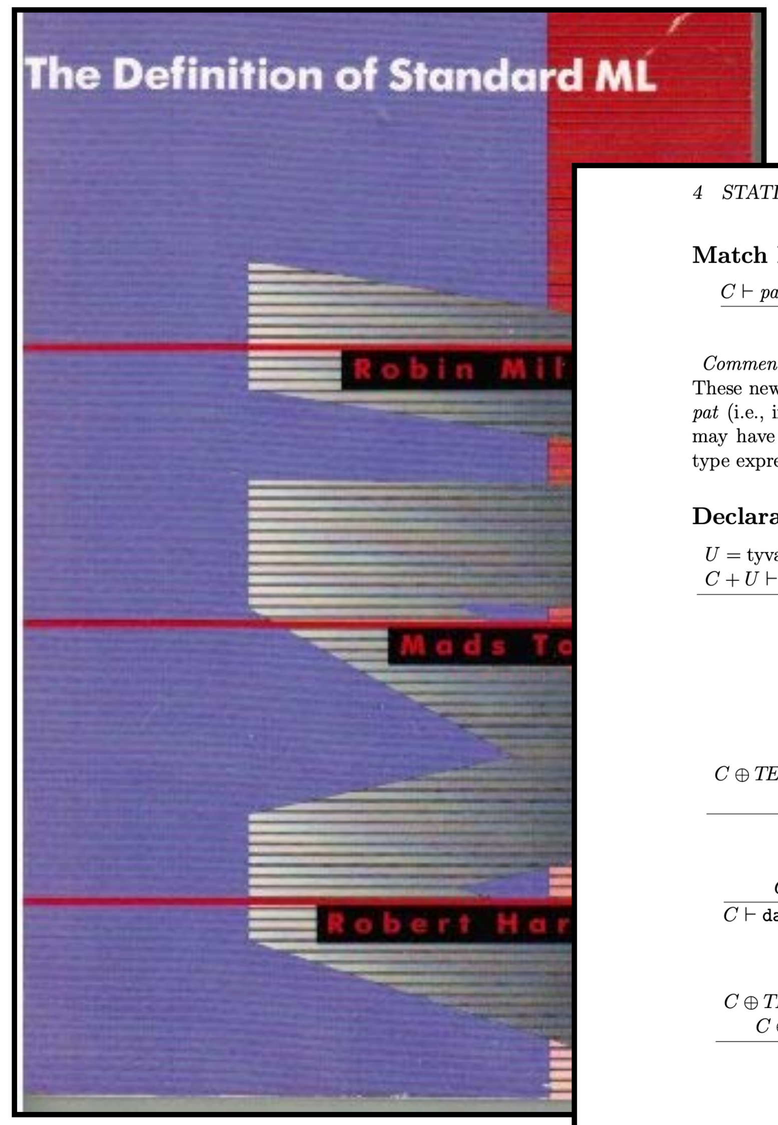
We tend to learn PLs the "wrong" (fast) way, i.e., reading tutorials and doing examples

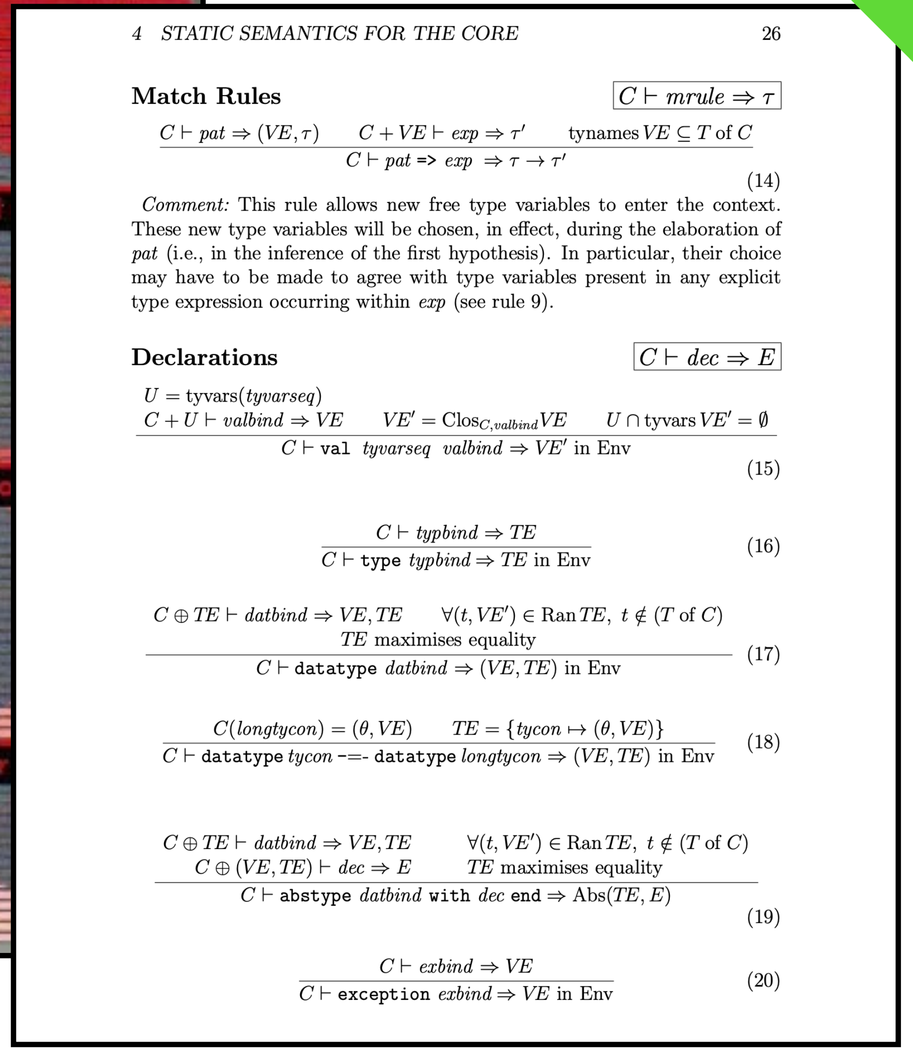
In this course we want to learn Rust the "right" (slow) way, i.e., formally describe what Rust is doing

We *won't* learn many cool, advanced features of Rust that are useful in practice

We *will* learn why Rust makes us tackle with the type system, and how it works







4 STATIC SEMANTICS FOR THE CORE 26

Match Rules $\boxed{C \vdash mrule \Rightarrow \tau}$

$$\frac{C \vdash pat \Rightarrow (VE, \tau) \qquad C + VE \vdash exp \Rightarrow \tau' \qquad \mathrm{tynames}\, VE \subseteq T \text{ of } C}{C \vdash pat \texttt{ => } exp \Rightarrow \tau \to \tau'} \quad (14)$$

Comment: This rule allows new free type variables to enter the context. These new type variables will be chosen, in effect, during the elaboration of *pat* (i.e., in the inference of the first hypothesis). In particular, their choice may have to be made to agree with type variables present in any explicit type expression occurring within *exp* (see rule 9).

Declarations $\boxed{C \vdash dec \Rightarrow E}$

$$\frac{\begin{array}{l} U = \mathrm{tyvars}(tyvarseq) \\ C + U \vdash valbind \Rightarrow VE \qquad VE' = \mathrm{Clos}_{C,valbind} VE \qquad U \cap \mathrm{tyvars}\, VE' = \emptyset \end{array}}{C \vdash \texttt{val}\ tyvarseq\ valbind \Rightarrow VE' \text{ in Env}} \quad (15)$$

$$\frac{C \vdash typbind \Rightarrow TE}{C \vdash \texttt{type}\ typbind \Rightarrow TE \text{ in Env}} \quad (16)$$

$$\frac{\begin{array}{c} C \oplus TE \vdash datbind \Rightarrow VE, TE \qquad \forall (t, VE') \in \mathrm{Ran}\, TE,\ t \notin (T \text{ of } C) \\ TE \text{ maximises equality} \end{array}}{C \vdash \texttt{datatype}\ datbind \Rightarrow (VE, TE) \text{ in Env}} \quad (17)$$

$$\frac{C(longtycon) = (\theta, VE) \qquad TE = \{tycon \mapsto (\theta, VE)\}}{C \vdash \texttt{datatype}\ tycon\ \texttt{-=-}\ \texttt{datatype}\ longtycon \Rightarrow (VE, TE) \text{ in Env}} \quad (18)$$

$$\frac{\begin{array}{ll} C \oplus TE \vdash datbind \Rightarrow VE, TE & \forall (t, VE') \in \mathrm{Ran}\, TE,\ t \notin (T \text{ of } C) \\ C \oplus (VE, TE) \vdash dec \Rightarrow E & TE \text{ maximises equality} \end{array}}{C \vdash \texttt{abstype}\ datbind\ \texttt{with}\ dec\ \texttt{end} \Rightarrow \mathrm{Abs}(TE, E)} \quad (19)$$

$$\frac{C \vdash exbind \Rightarrow VE}{C \vdash \texttt{exception}\ exbind \Rightarrow VE \text{ in Env}} \quad (20)$$

How *does* it work?

How *does* it work?

Answer: Linear types (Affine types really), which is based on **Linear logic** of Girard (1980s)

How *does* it work?

Answer: Linear types (Affine types really), which is based on **Linear logic** of Girard (1980s)

Rough idea:

How *does* it work?

Answer: Linear types (Affine types really), which is based on **Linear logic** of Girard (1980s)

Rough idea:

A function of type $A \to B$ takes an A and gives a B

How *does* it work?

Answer: Linear types (Affine types really), which is based on **Linear logic** of Girard (1980s)

Rough idea:

A function of type $A \to B$ takes an A and gives a B

A function of type $A \multimap$ B **consumes** an A and gives a B

How *does* it work?

Answer: Linear types (Affine types really), which is based on **Linear logic** of Girard (1980s)

Rough idea:

A function of type $A \to B$ takes an A and gives a B

A function of type $A \multimap$ B **consumes** an A and gives a B

Roughly speaking, this ensures that data is never unnecessarily duplicated or thrown away

How *does* it work?

Answer: Linear types (Affine types really), which is based on **Linear logic** of Girard (1980s)

Rough idea:

A function of type $A \to B$ takes an A and gives a B

A function of type $A \multimap$ B **consumes** an A and gives a B

Roughly speaking, this ensures that data is never unnecessarily duplicated or thrown away

This is exactly the kind of information we need to know to make sure that, e.g., there are no dangling pointers **without actually specifying it** (Rust doesn't work exactly like this)

Workshop:

Install Rust

Set up a Cargo Project

The Task

Follow the in The Rust Programming Language (RPL) on installing **rustup.** If you're using windows I *highly recommend* using WSL. If you finish, then follow the tutorial in RPL called **Hello, Cargo!**

Note: This is how I'll take attendance, so please make sure to talk to me before the end of lecture