
Systems Programming for All!

What is Rust all about?

Elegant code that runs fast.

Code that **does the right thing.**

Two years old on May 15!





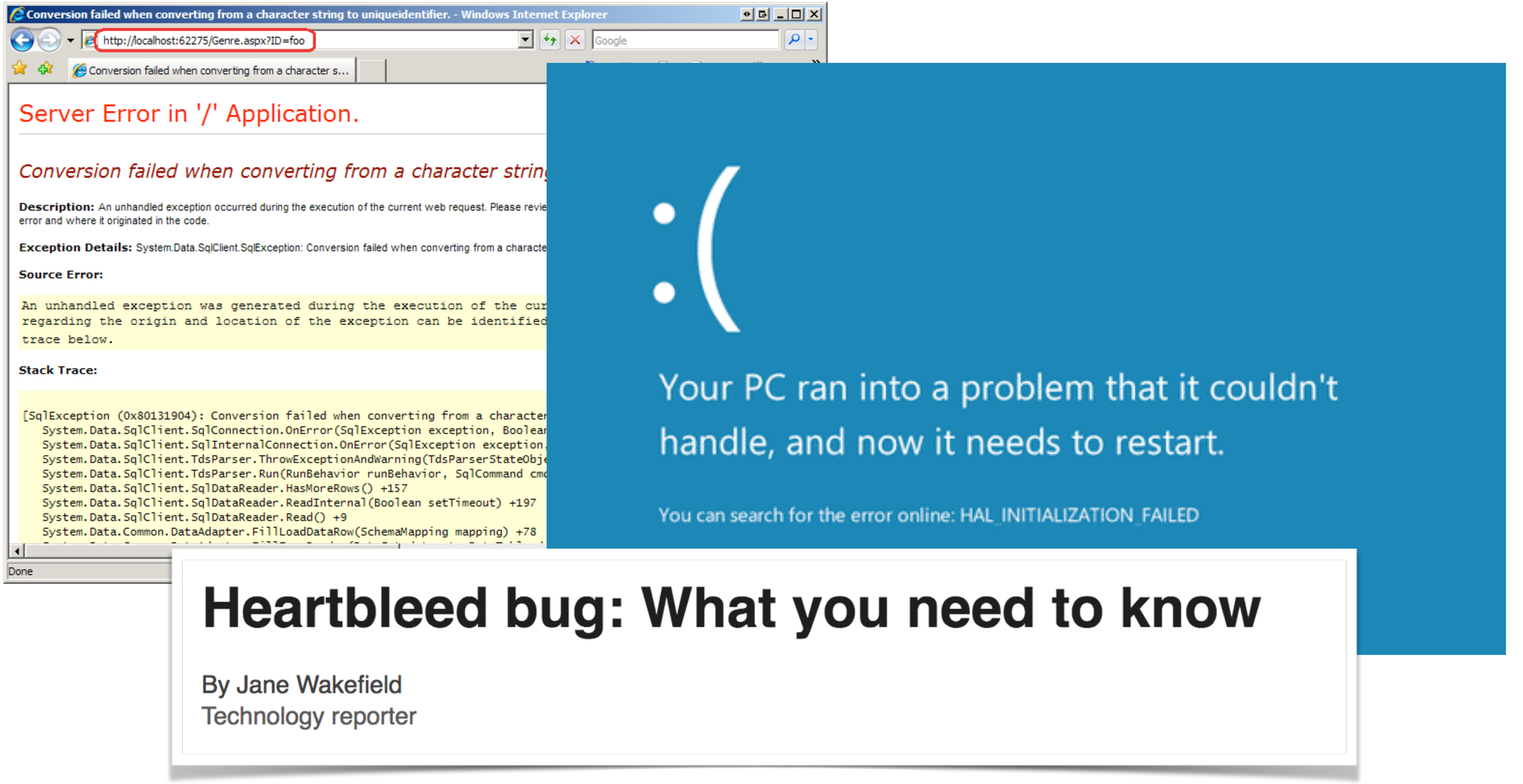
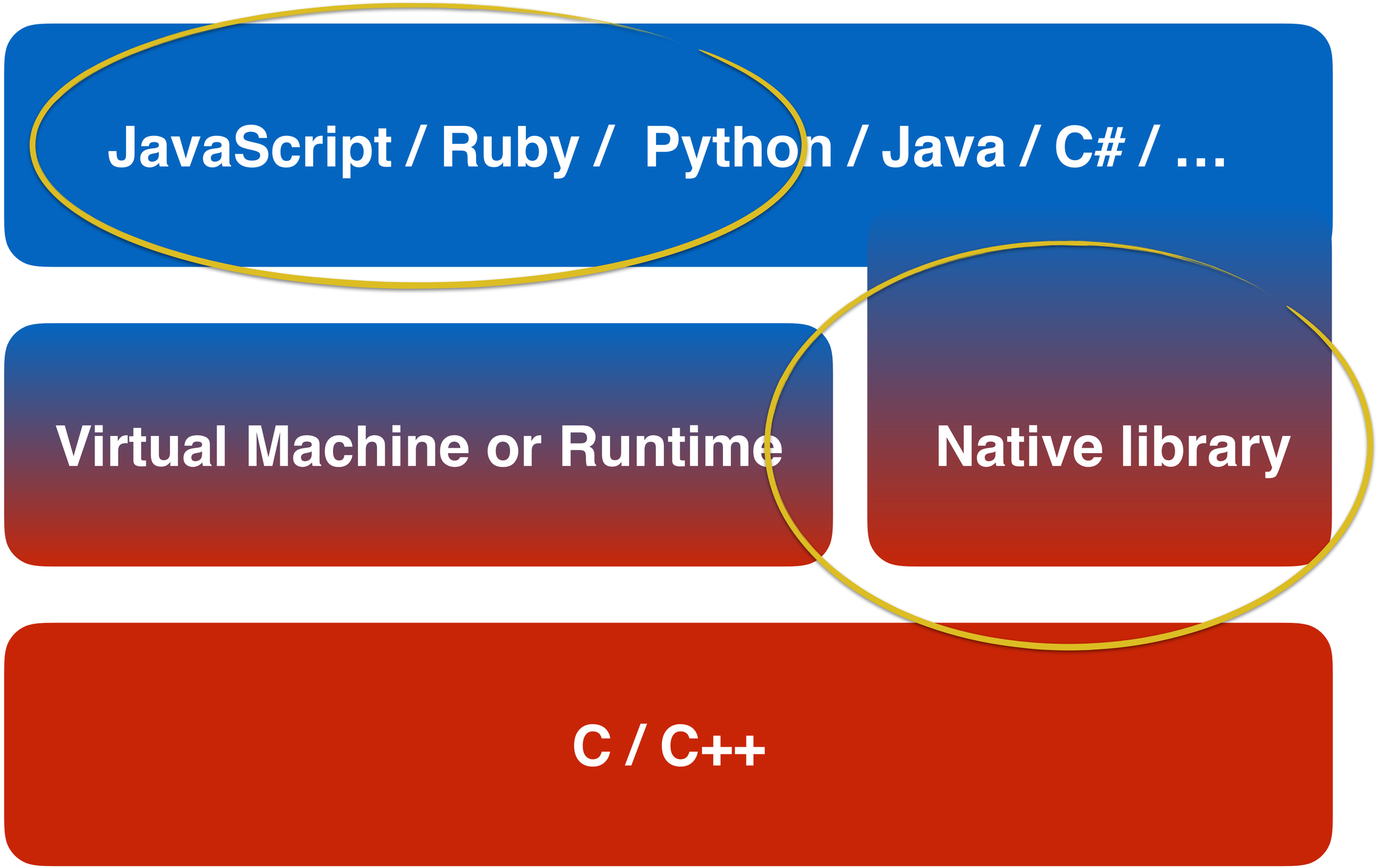
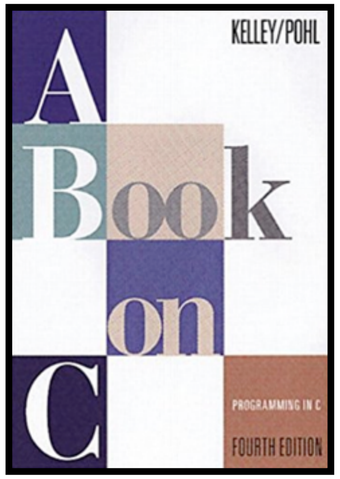
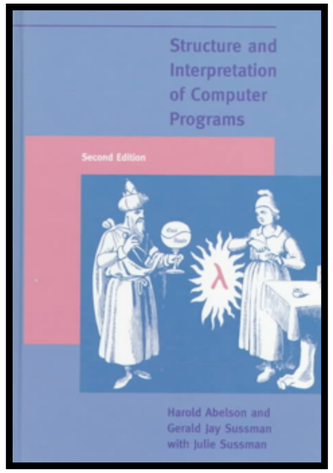


Conversion failed when converting from a character string to uniqueidentifier. - Windows Internet Explorer
http://localhost:62275/Genre.aspx?ID=foo
Google
Conversion failed when converting from a character s...
Server Error in '/' Application.
Conversion failed when converting from a character strin
Description: An unhandled exception occurred during the execution of the current web request. Please revie
error and where it originated in the code.
Exception Details: System.Data.SqlClient.SqlException: Conversion failed when converting from a characte
Source Error:
An unhandled exception was generated during the execution of the cur
regarding the origin and location of the exception can be identified
trace below.
Stack Trace:
[SqlException (0x80131904): Conversion failed when converting from a character
System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean
System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception,
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObj
System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmd
System.Data.SqlClient.SqlDataReader.HasMoreRows() +157
System.Data.SqlClient.SqlDataReader.ReadInternal(Boolean setTimeout) +197
System.Data.SqlClient.SqlDataReader.Read() +9
System.Data.Common.DataAdapter.FillLoadDataRow(SchemaMapping mapping) +78
Done
:(
Your PC ran into a problem that it couldn't handle, and now it needs to restart.
You can search for the error online: HAL_INITIALIZATION_FAILED
Heartbleed bug: What you need to know
By Jane Wakefield
Technology reporter

Photo credit: tao lin
https://www.flickr.com/photos/taolin/8639700222/


JavaScript / Ruby / Python / Java / C# / …
Virtual Machine or Runtime
Native library
C / C++

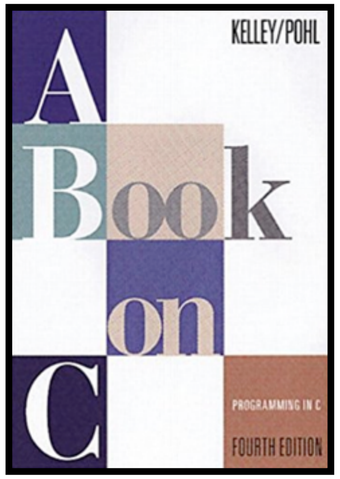


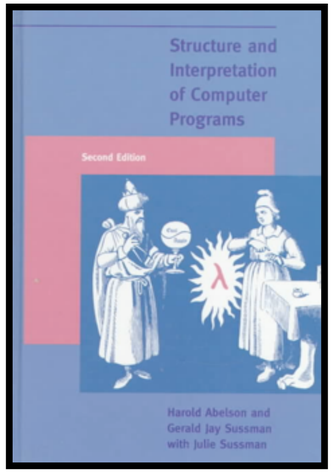


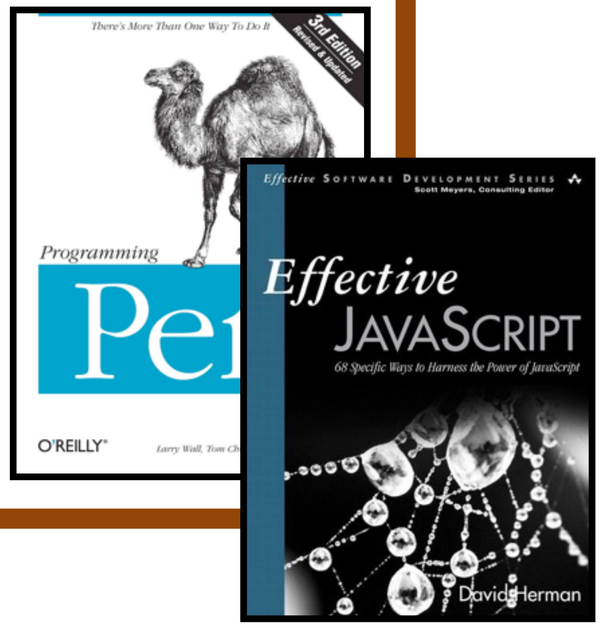
Images used without permission.

Algorithm “Learn All The Things”:

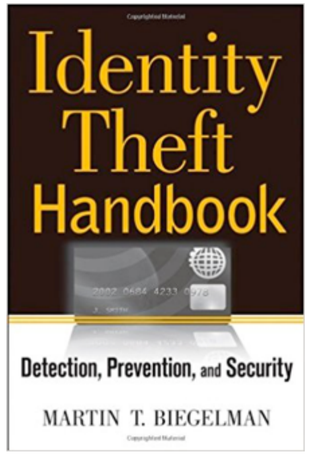
```
Find shelf S with computer books.
Let N = 1.
While less than N books on shelf S {
    Borrow the Nth book from shelf S.
    Read it and return it.
    N = N + 1.
}
```

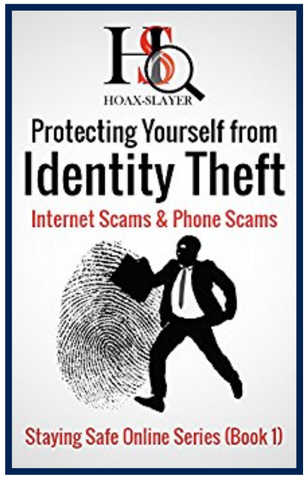


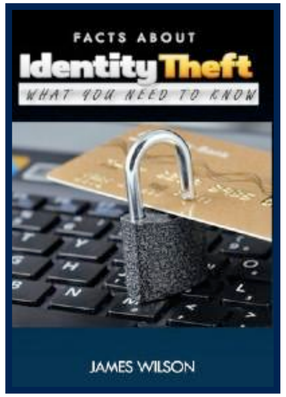


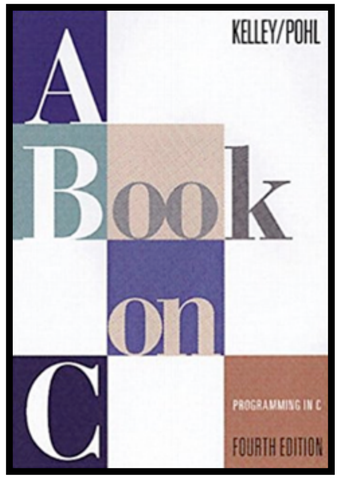


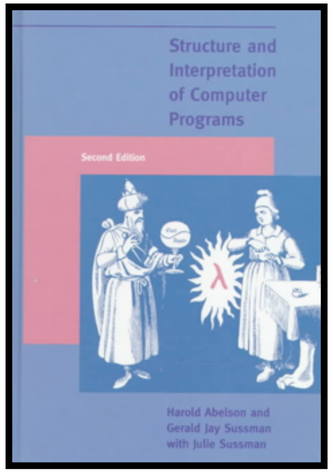
Images used without permission.

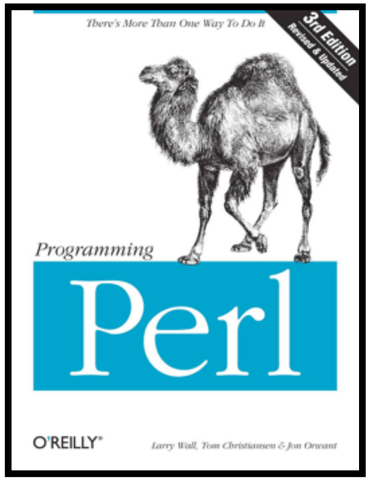


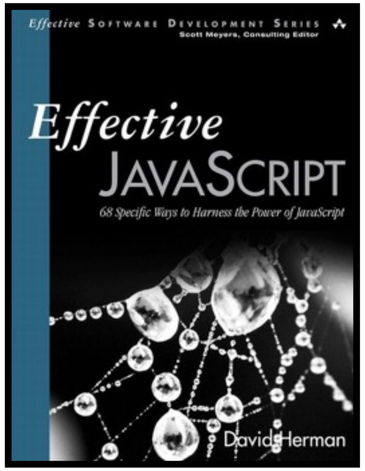






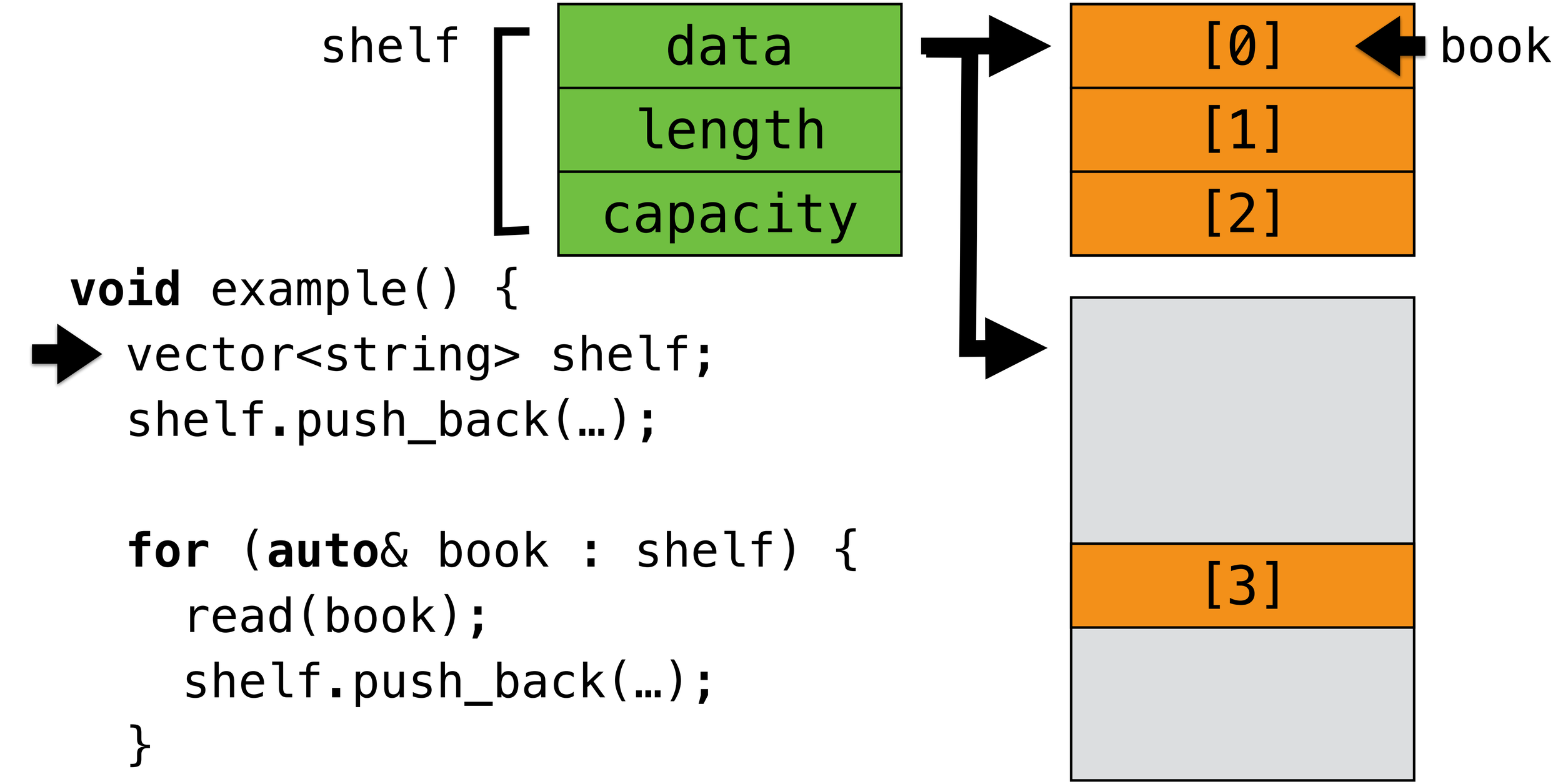






Images used without permission.

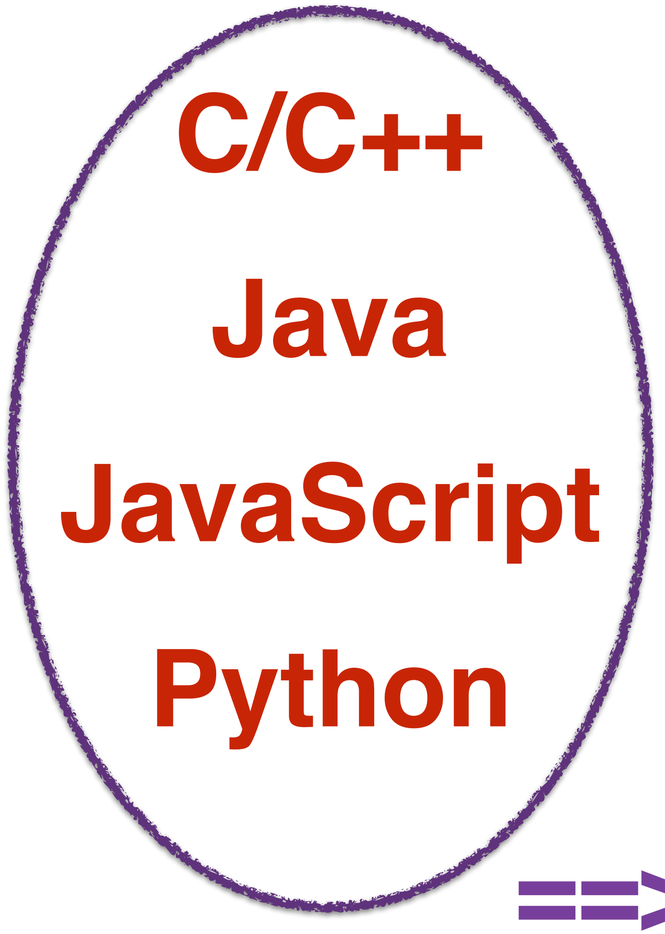
C / C++




```
void example() {
  vector<string> shelf;
  shelf.push_back(…);

  for (auto& book : shelf) {
    read(book);
    shelf.push_back(…);
  }
}
```

Iterator invalidation: modify while iterating



C/C++ — Crash, read random data, wev

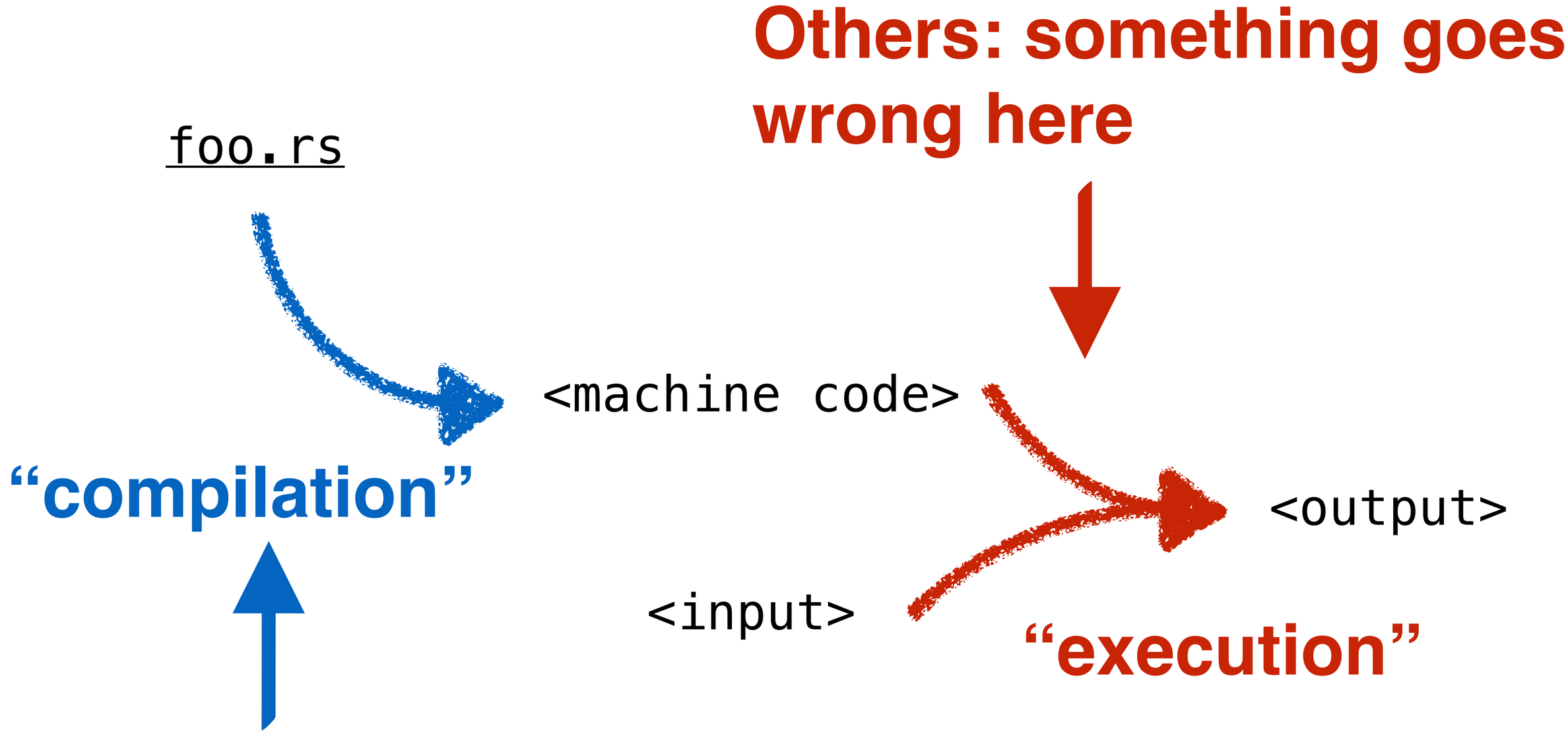
Java — Throw exception

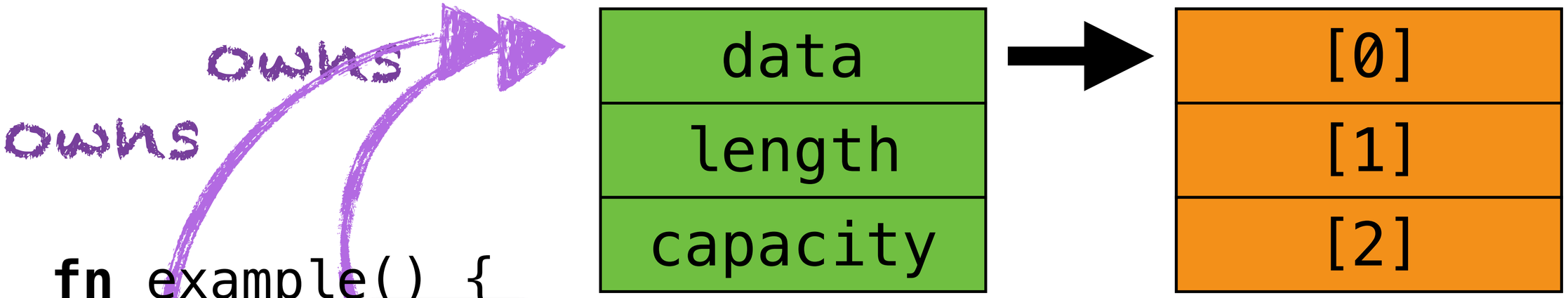
JavaScript — Do… something

Python — Something… slightly different

==> Runtime Error

Rust? — Detect problem when compiling


Others: something goes wrong here
foo.rs
<machine code>
“compilation”
<output>
<input>
“execution”
Rust: check ownership + borrowing




```
fn example() {
    let mut shelf = Vec::new();
    shelf.push(…);

    let shelf2 = shelf;
    shelf.push(…);
    print(shelf2.len());
    shelf2.push(…);
}
```

moved

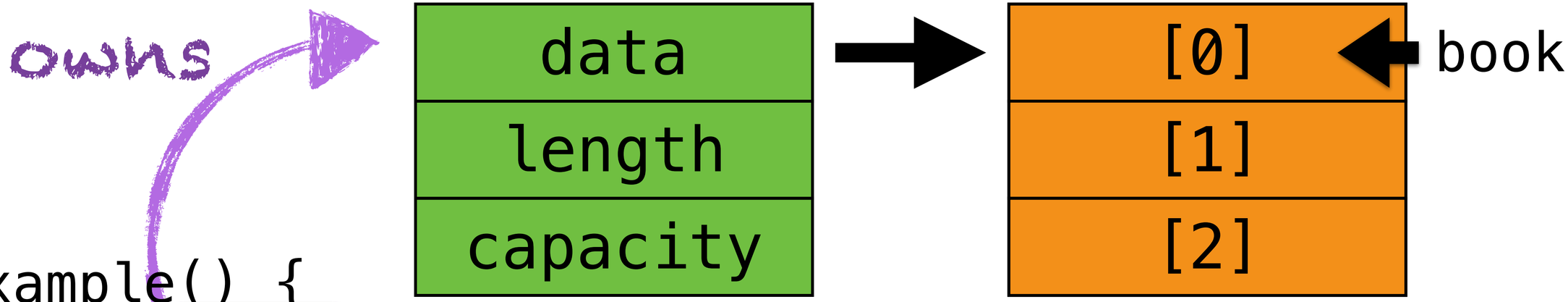
❌ "moved"

✅

❌ "not mutable"

1. Every value is owned by a single variable.

(a) Owner gets to decide whether it is mutable.



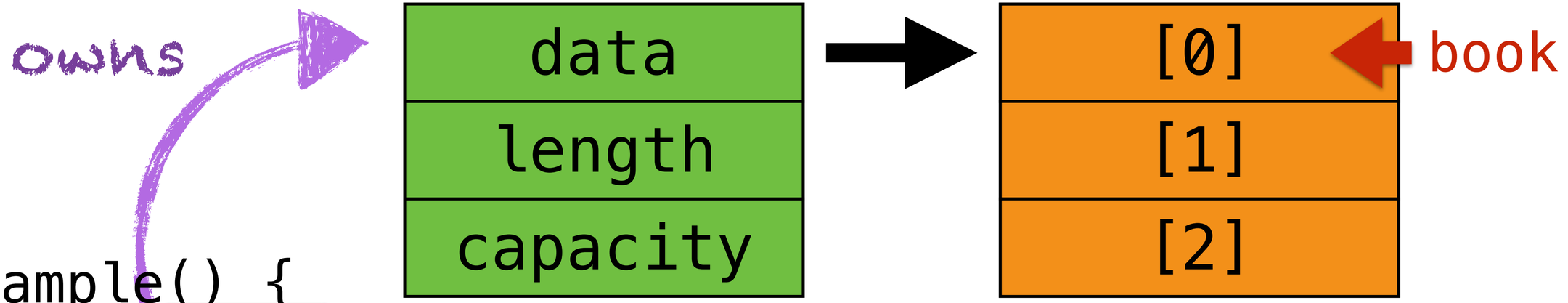

```
fn example() {
    let mut shelf = Vec::new();
    shelf.push(…);

    for book in &shelf {
        read(book);
        shelf.push(…); ❌ "immutable while borrowed"
    }
    `shelf` borrowed here

    shelf.push(…); ✅
}
```

borrows

2. *Shared borrows* make the value temporarily *immutable*.




```
fn example() {
    let mut shelf = Vec::new();
    shelf.push(…);

    for book in &mut shelf {
        edit(book);
        print(shelf.len());
    }

    shelf.push(…);
}
```

borrows mutably

✅ make changes

❌ "mutably borrowed"

`shelf` mutably borrowed here

3. *Mutable borrows* have *unique* access to the value.

1. Every value is owned by a single variable.

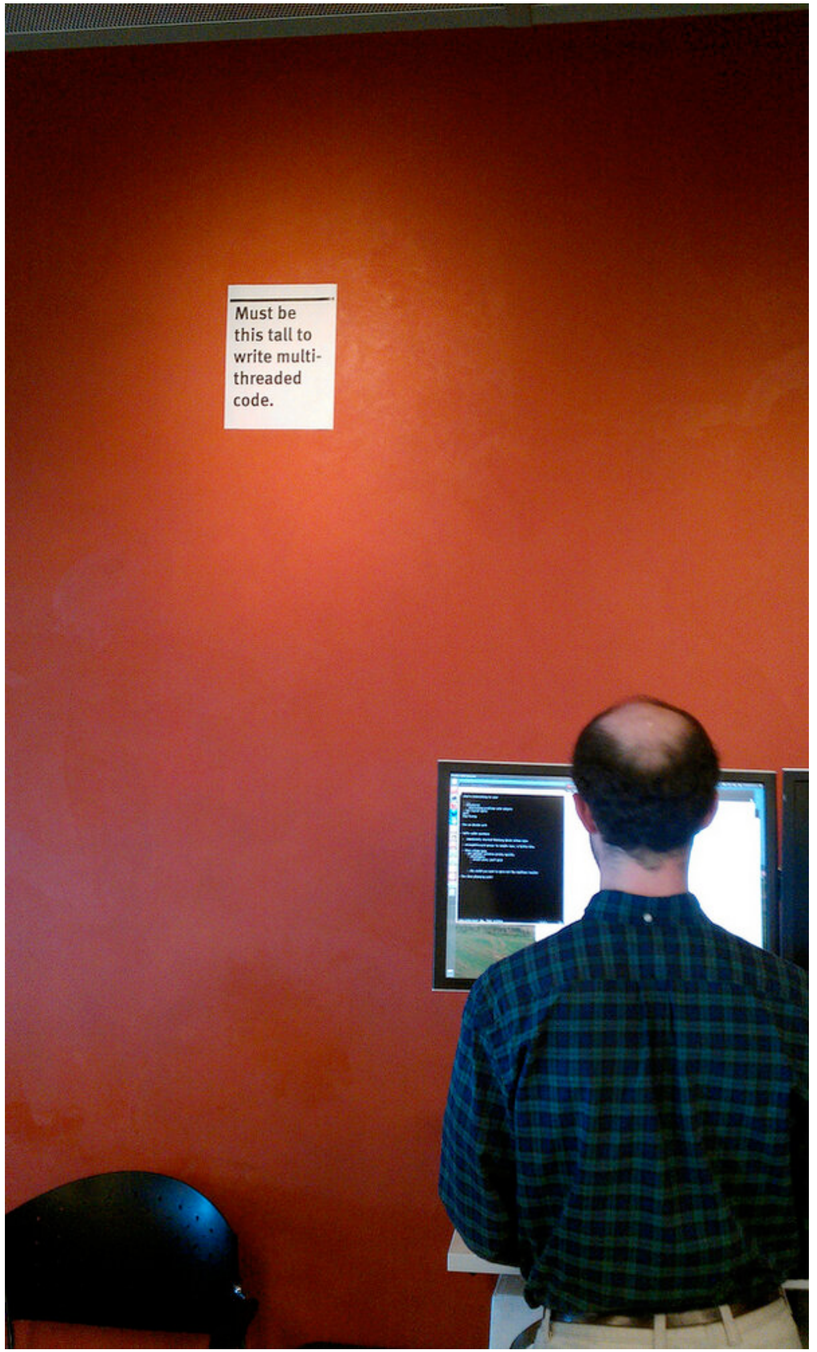
2. *Shared borrows* make the value temporarily *immutable*.

3. *Mutable borrows* have *unique* access to the value.

Recap: Mutation while iterating

C/C++	Maximally efficient
	Maximally dangerous 🌪️🌪️🌪️
Others	Less efficient
	Risky ⛈️🏊⛈️
Rust	Maximally efficient
	Perfectly safe 😎😎😎

“Must be this tall to write multi-threaded code”



Idea credit: David Baron
Mozilla Engineer Extraordinaire and All-Around Great Guy

Message-passing

Erlang, Go, JavaScript, … Rust

Futures and Async I/O

JavaScript, C#, … Rust

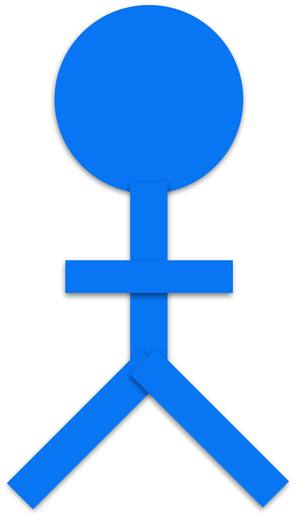
Data parallelism

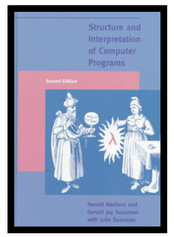
C and C++, Java, C#, … Rust

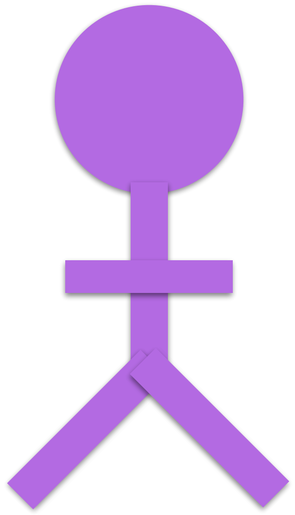
Shared state and threads

C and C++, Java, C#, … Rust

Message-passing







moved

```
let book = …;
channel.send(book);
```

```
let book = channel.receive();
```

```
read(book);
```

❌ "moved"

1. Every value is owned by a single variable.

Data parallelism

borrowed from caller

```
fn load_images(paths: &[PathBuf]) -> Vec<Image> {
    paths.iter()
        .map(|path| {
            Image::load(path)
        })
        .collect()
}
```

For each path…

…load an image…

…create and return a vector.

paths = [“a.jpg”, “b.png”, …, “c.jpg”]

```
extern crate rayon;
```

Third-party library

```
fn load_images(paths: &[PathBuf]) -> Vec<Image> {
    paths.par_iter()
        .map(|path| {
          Image::load(path)
        })
        .collect()
}
```

Make it parallel

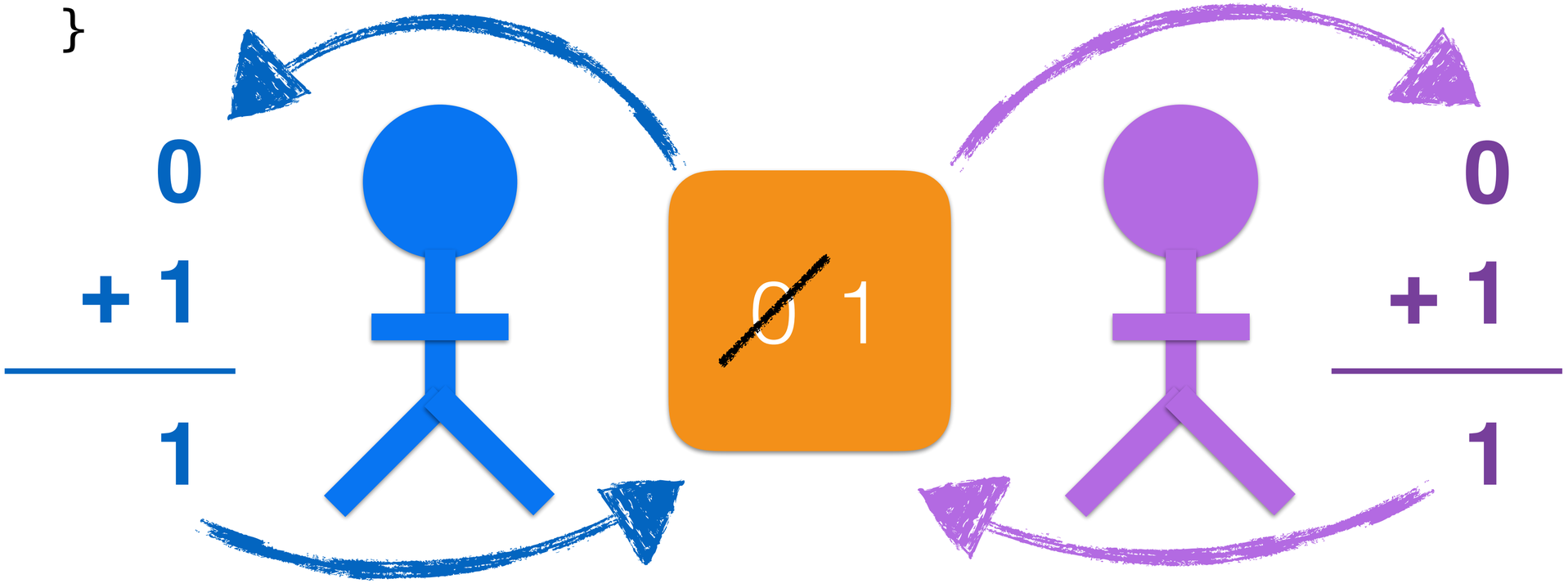
```
paths = [   “a.jpg”,        “b.png”,    …,      “c.jpg”     ]
```

```
fn load_images(paths: &[PathBuf]) -> Vec<Image> {
    let mut jpgs = 0;
    paths.par_iter()
        .map(|path| {
            if path.ends_with(“.jpg”) { jpgs += 1; }
            Image::load(path)
        })
        .collect()
}
```

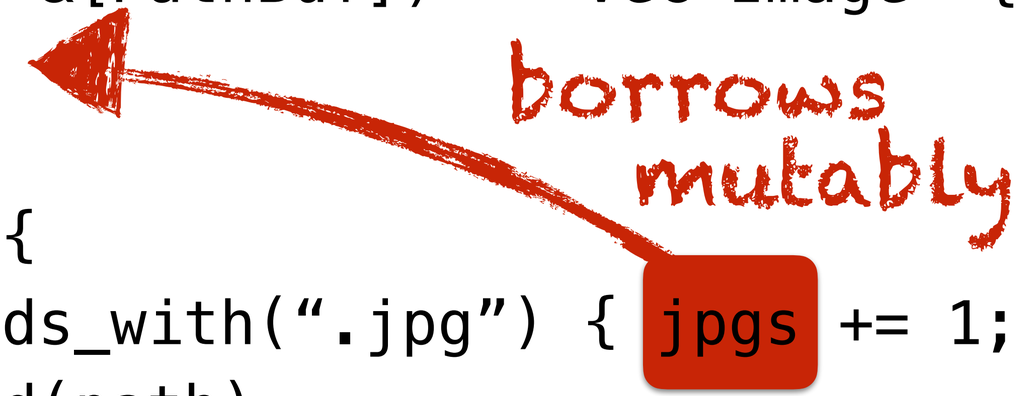
How many jpgs seen so far?

If current file name ends in “jpg”…

…add 1 to the counter.



```
fn load_images(paths: &[PathBuf]) -> Vec<Image> {
    let mut jpgs = 0;
    paths.par_iter()
         .map(|path| {
           if path.ends_with(“.jpg”) { jpgs += 1; }
           Image::load(path)
         })
         .collect()
}
```



1. Every value is owned by a single variable.

3. *Mutable borrows* have *unique* access to the value.

```
fn load_images(paths: &[PathBuf]) -> Vec<Image> {
    let mut jpgs = 0;
    paths.par_iter()
         .map(          )
         .collect()
}
```

borrows mutably

borrows mutably

```
|path| {
  if path.ends_with(“.jpg”) {
    jpgs += 1;
  }
  Image::load(path)
}
```

```
|path| {
  if path.ends_with(“.jpg”) {
    jpgs += 1;
  }
  Image::load(path)
}
```

3. *Mutable borrows* have *unique* access to the value.

Recap: Parallel execution

Others	Flexible	✅✅✅
	Maximally dangerous	🌪️🌪️🌪️
	Restrictive	🚫🚫🚫
	Perfectly safe	😎😎😎
Rust	Flexible	✅✅✅
	Perfectly safe	😎😎😎

More examples…

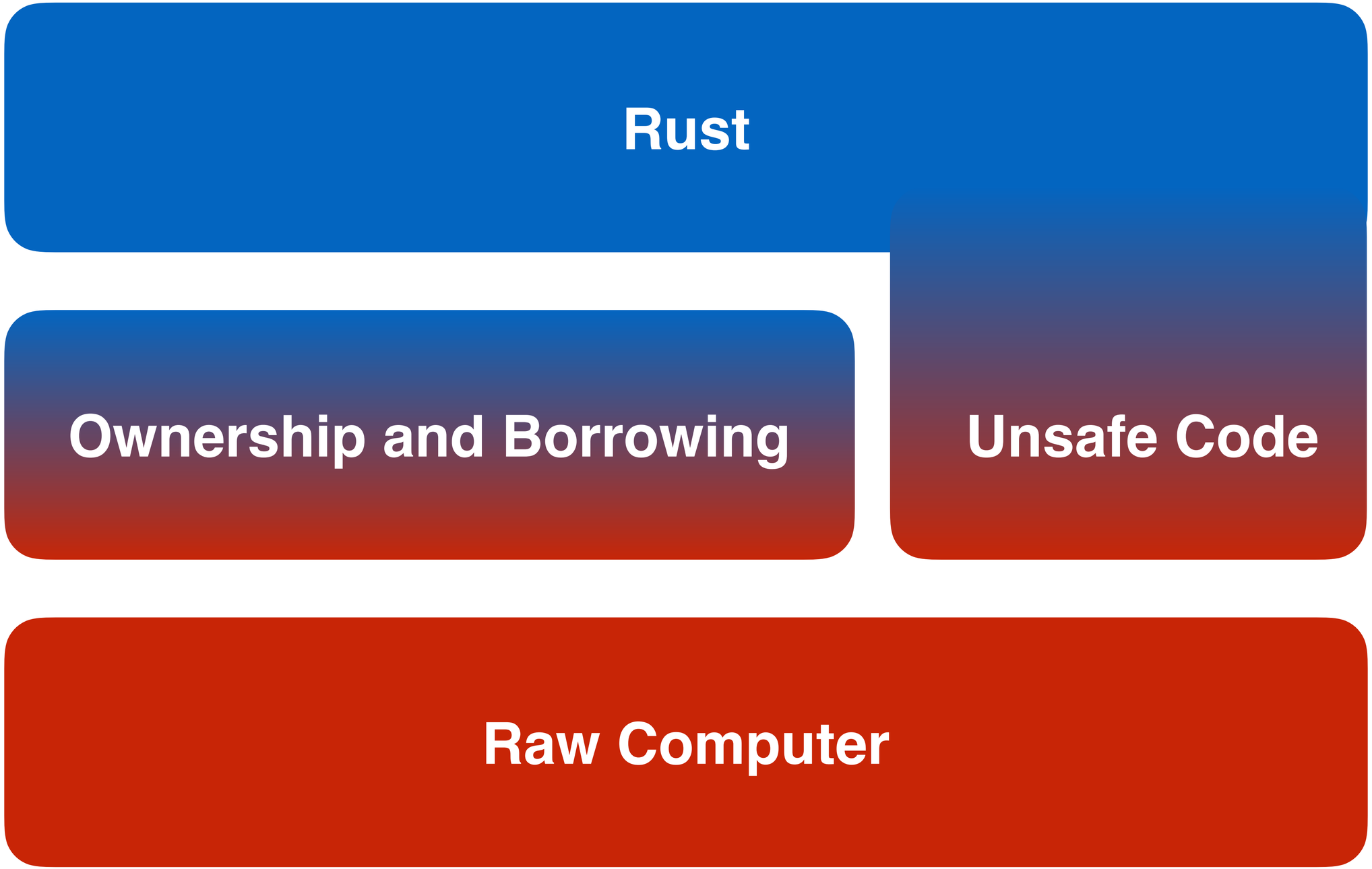
No null pointers (Foo *vs* Option<Foo>)

Generic programming

Robust error handling

…

Unsafe


Rust
Ownership and Borrowing
Unsafe Code
Raw Computer

Safe abstractions

```
fn pass_safely(…) {
  if !car_coming() {
    unsafe {
      drive_in_left_lane();
    }
  }
}
```

Validates input, etc.

Trust me.

Community

Photo credit: David McSpadden
https://www.flickr.com/photos/familyclan/15535822737/

CARGO
packages for Rust

Click or press 'S' to search...

Browse All Crates | Docs ▾ | Log in with GitHub

Fork me on GitHub

The Rust community's crate host

Install Cargo Getting Started

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

148,961,595 Downloads

9,190 Crates in stock

New Crates

- catflap (1.0.0)
- arbitrary (0.1.0)

Most Downloaded

- libc (0.2.22)
- winapi (0.2.8)

Just Updated

- mml (0.1.3)
- trackable (0.1.7)

Open and welcoming

From http://jvns.ca/blog/2016/09/11/rustconf-keynote/

Worldwide

rust-lang.org

irc.mozilla.org, #rust, #rust-beginners

doc.rust-lang.org/nightly/book/second-edition/

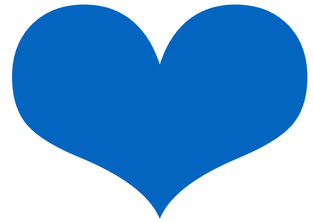
intorust.com

Boston area

https://www.meetup.com/BostonRust/

Next meetup: June 28 in Cambridge

http://boston-rust.herokuapp.com/



Also, stickers!