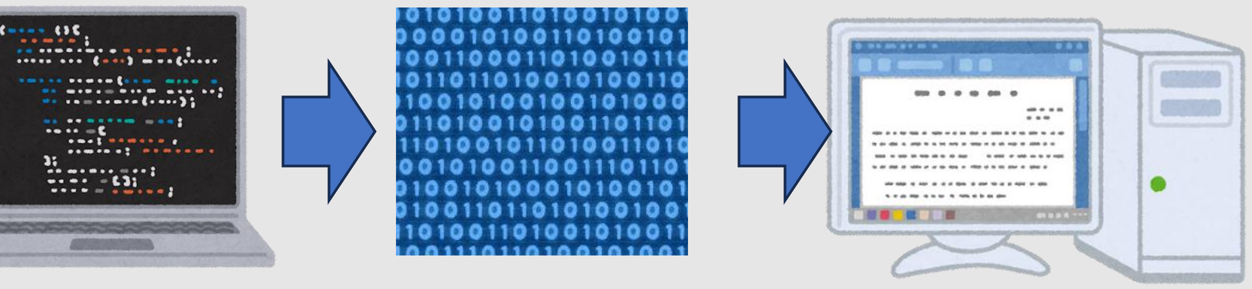
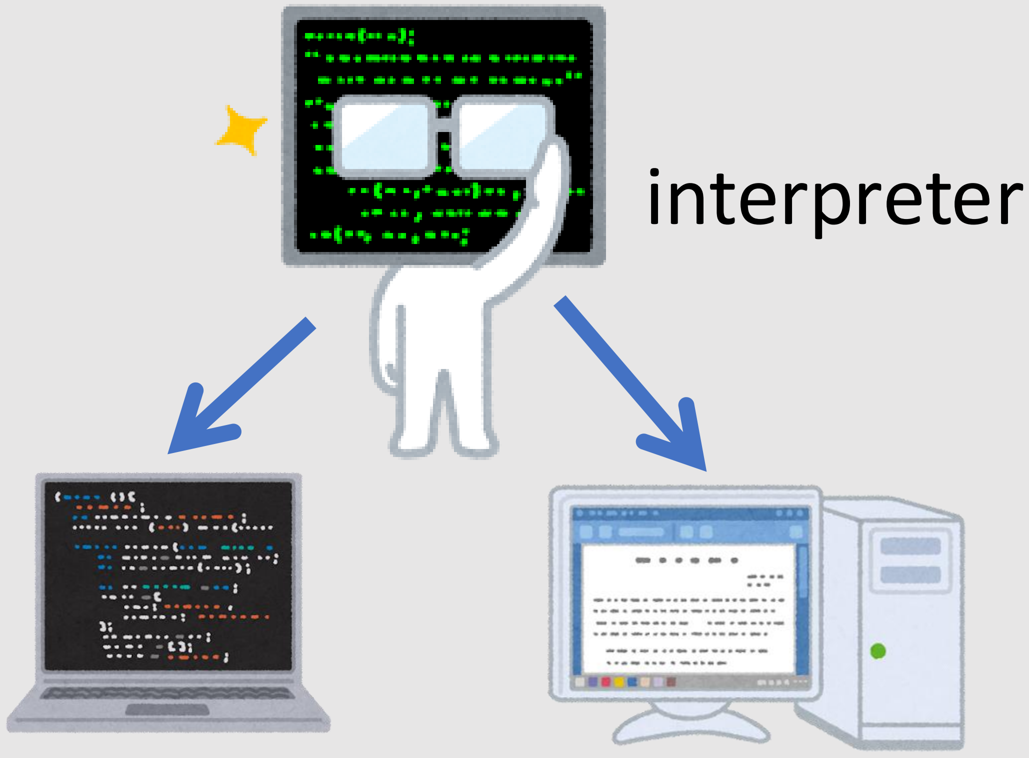
Resource Management in Programming Languages

Compiled and Interpreted Language

Compiled Language



Interpreted Language



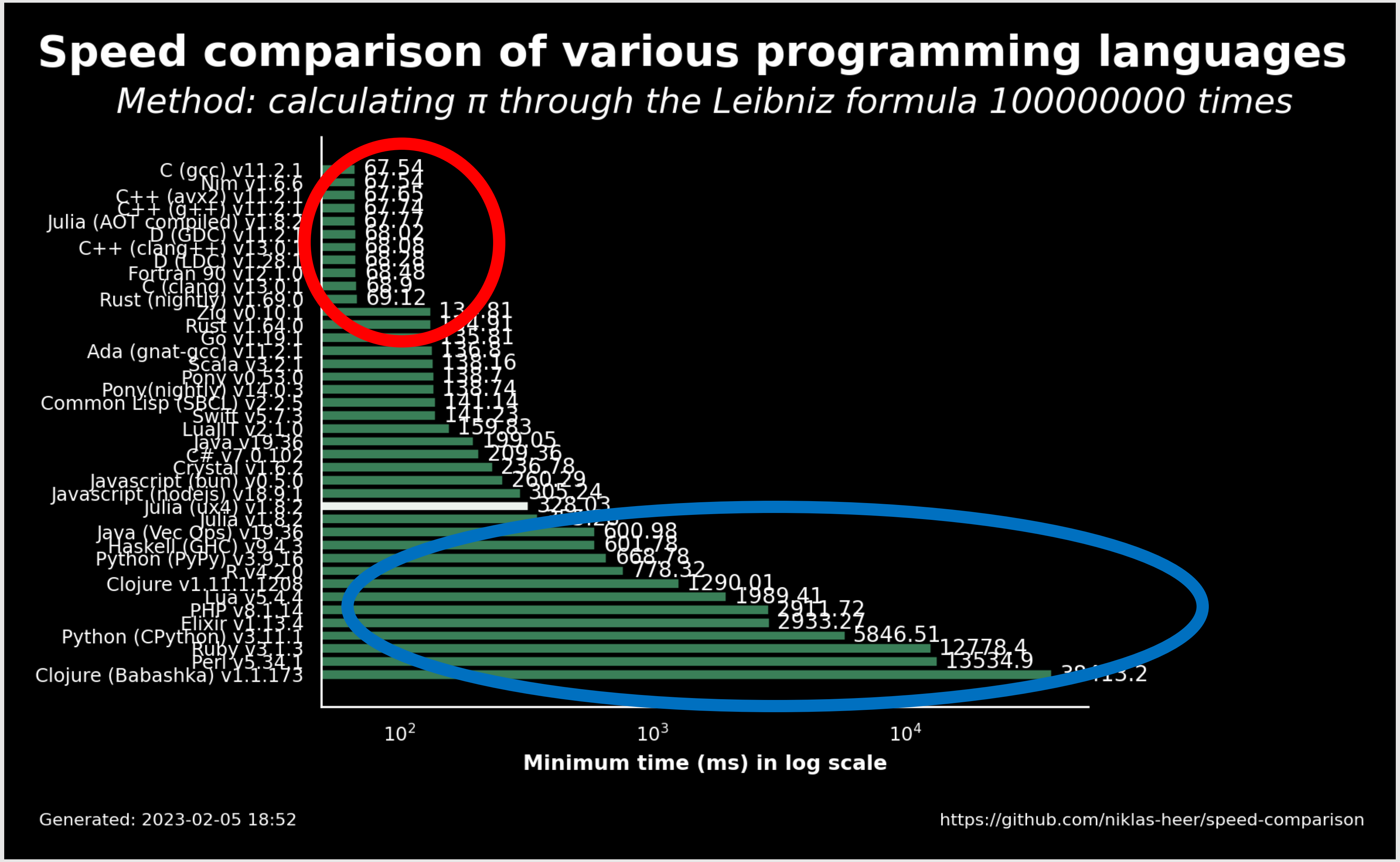
Compiled and Interpreted Language

Compiled Language	*Interpreted Language*
Static typing	Dynamic typing
Compile-time Memory management	Garbage collection

The Difference of Speed

Compiled language

Interpreted language



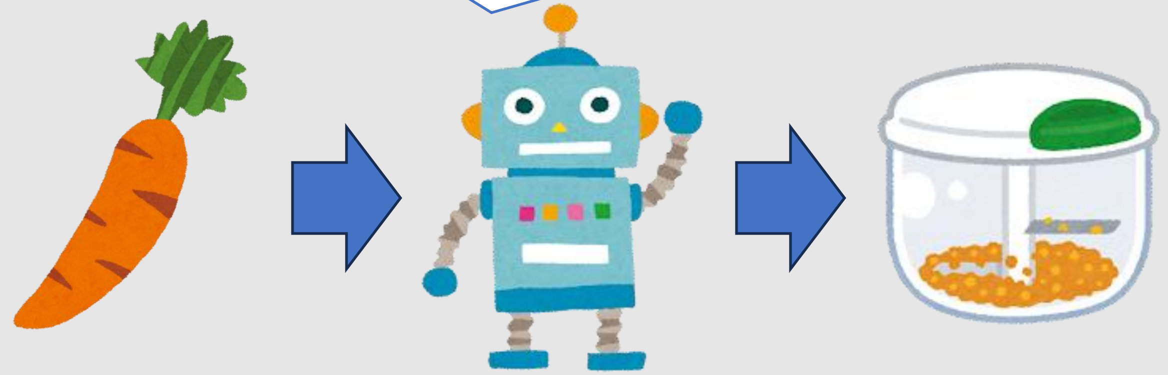
https://niklas-heer.github.io/speed-comparison/

Why Such Difference in Speed?

Static typing

```
int one_up(int a) {
  return a + 1;
}
```

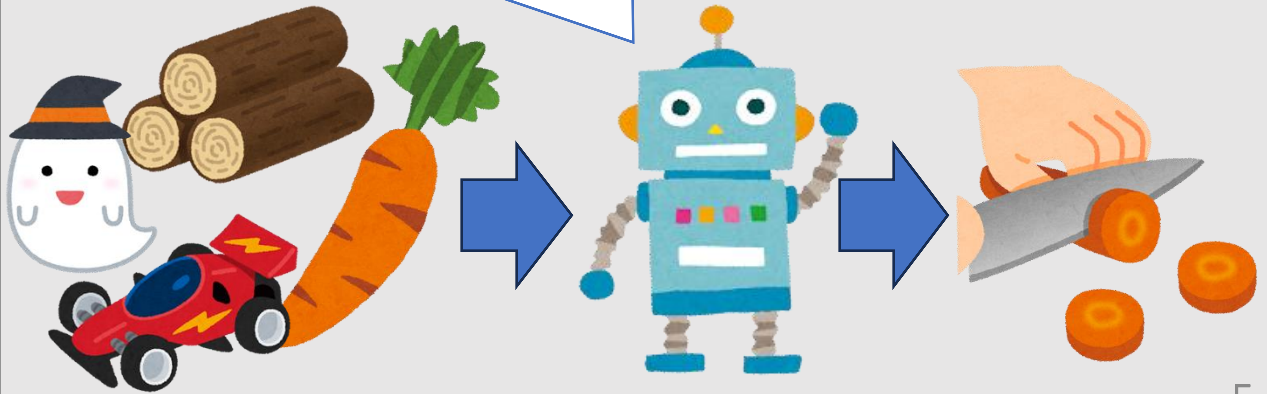
Let’s use a specialized tool for cutting I prepared beforehand.



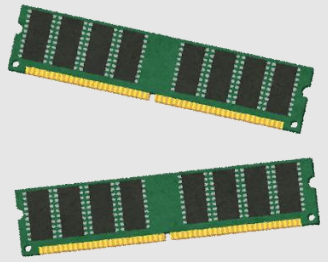
Dynamic typing

```
def one_up(a):
  return a + 1
```

No idea what’s coming.
Let’s search for a tool when it comes.



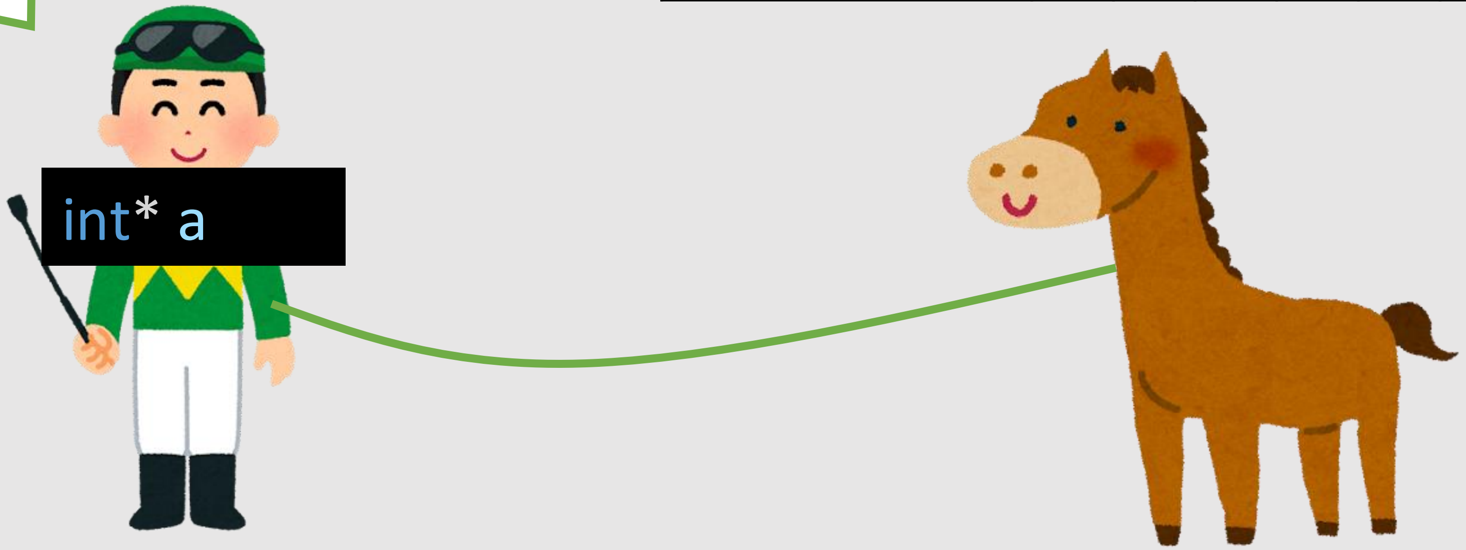
Variables and Resource



```
void main() {
    int* a = new char[4];
}
```

Address	Data							
0x7ffffffffffe2fb	0	0	0	0	0	0	0	0
0x7ffffffffffe2fc	0	0	0	0	0	0	0	1
0x7ffffffffffe2fd	0	0	0	0	0	0	0	0
0x7ffffffffffe2fe	0	0	0	0	0	0	0	0
0x7ffffffffffe2ff	0	0	0	0	0	0	0	0
0x7ffffffffffe300	0	0	0	0	0	0	0	0
0x7ffffffffffe301	0	0	0	0	0	0	0	0

I own a piece of memory.
My lifetime is between “{“ and “}”

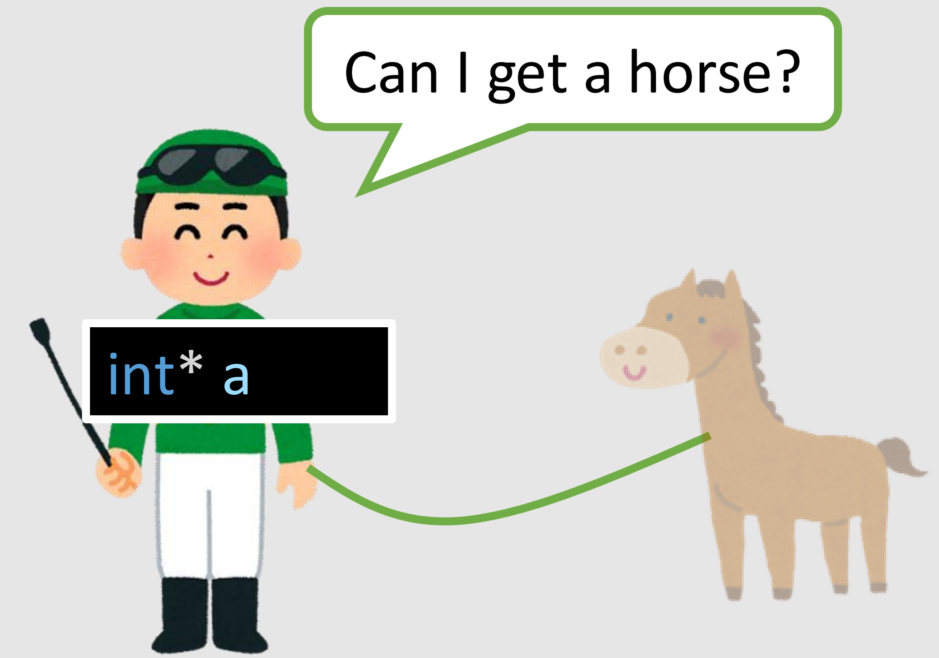


Resource Management

```cpp
void main() {
  int* a = new int[32];
  a[0] = 1;
  delete[] a;
}
```

Resource Management

```
void main() {
  int* a = new int[32];
  a[0] = 1;
  delete[] a;
}
```



Resource Management

```
void main() {
  int* a = new int[32];
  a[0] = 1;
  delete[] a;
}
```

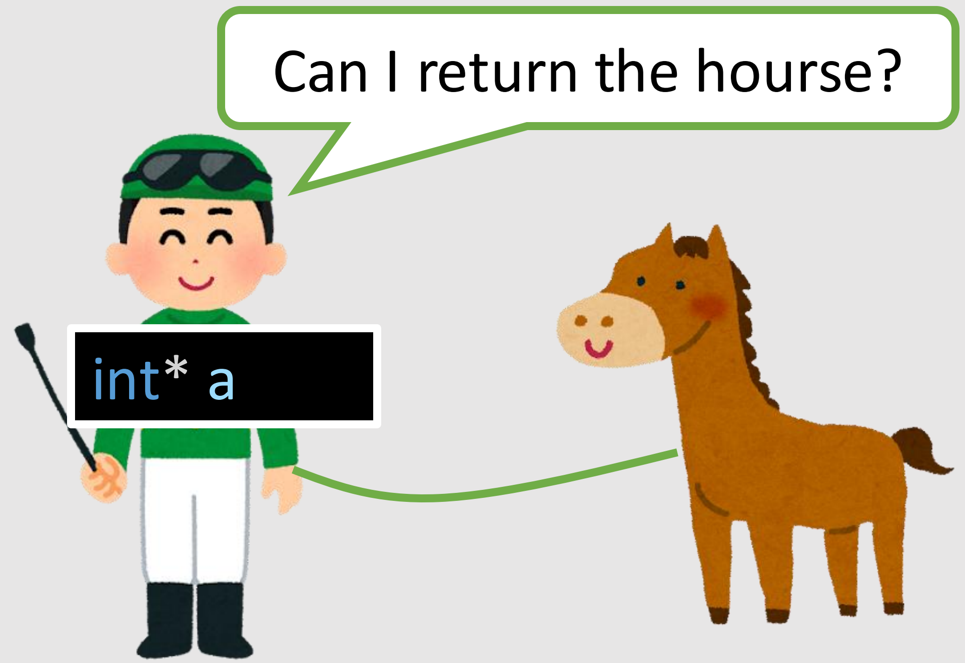
Have fun!
OS
Thanks!
int* a

Resource Management

```
void main() {
  int* a = new int[32];
  a[0] = 1;
  delete[] a;
}
```

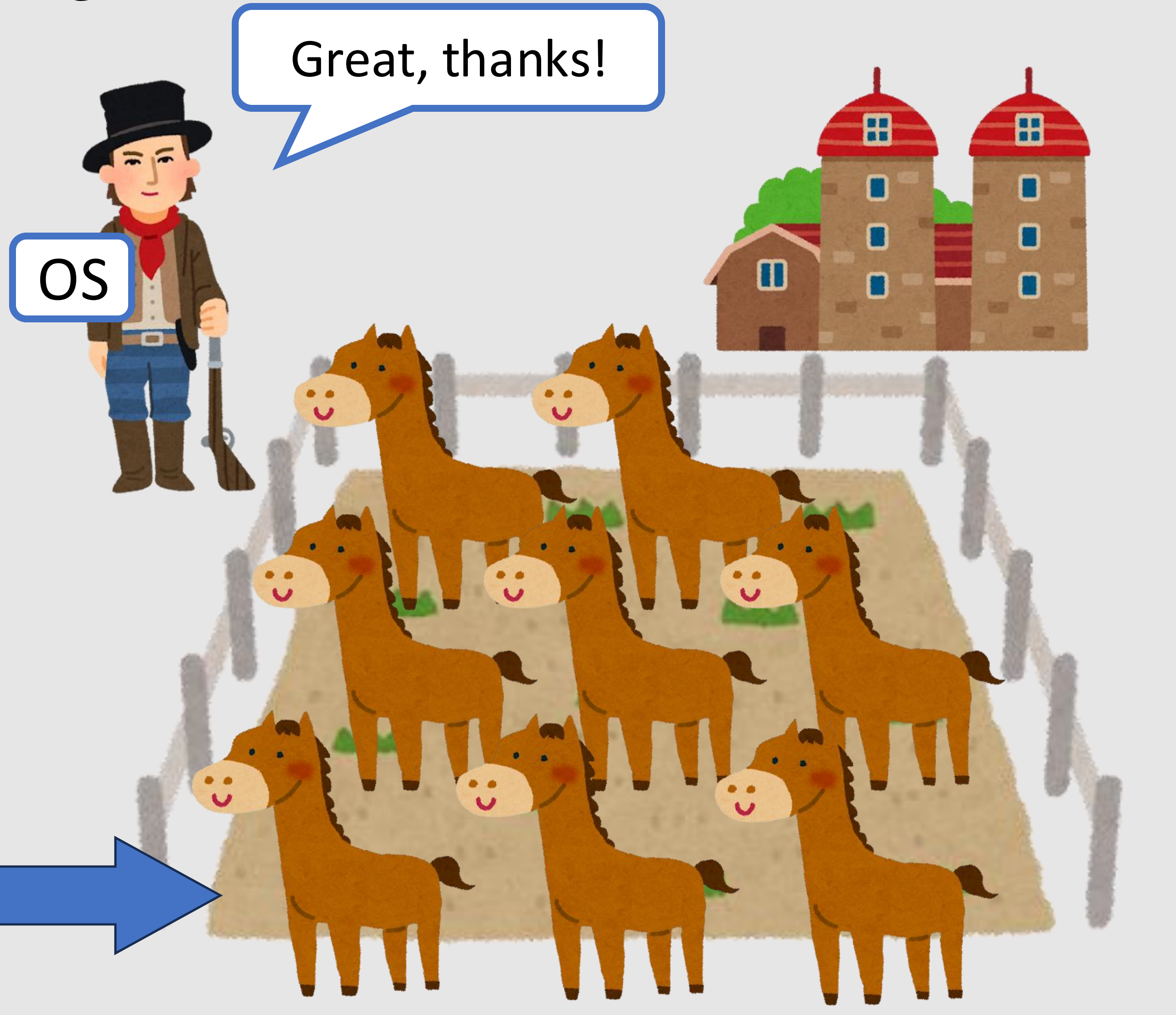
Resource Management

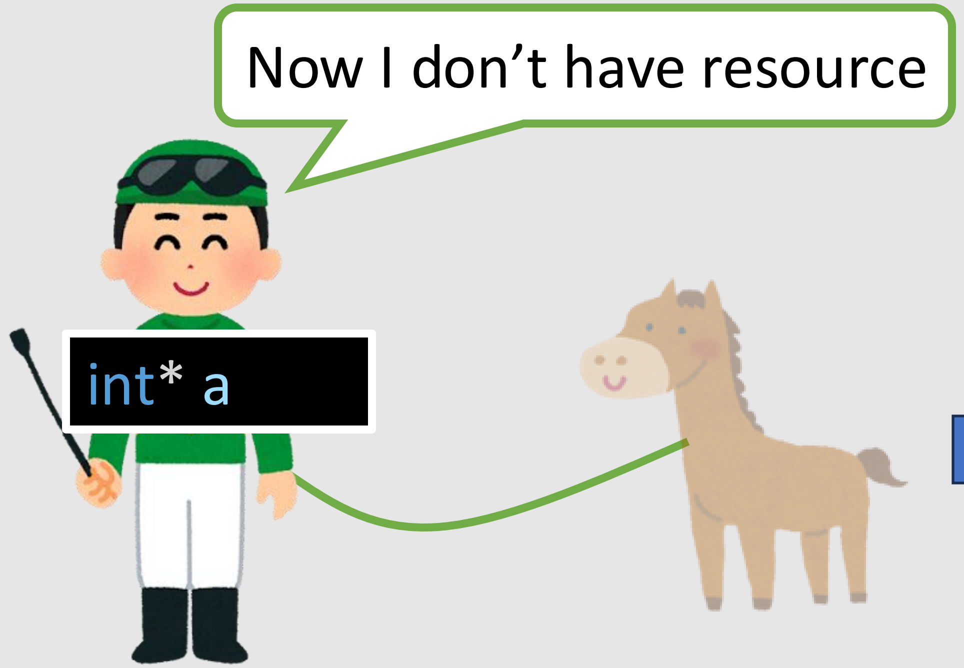
```
void main() {
  int* a = new int[32];
  a[0] = 1;
  delete[] a;
}
```



Resource Management

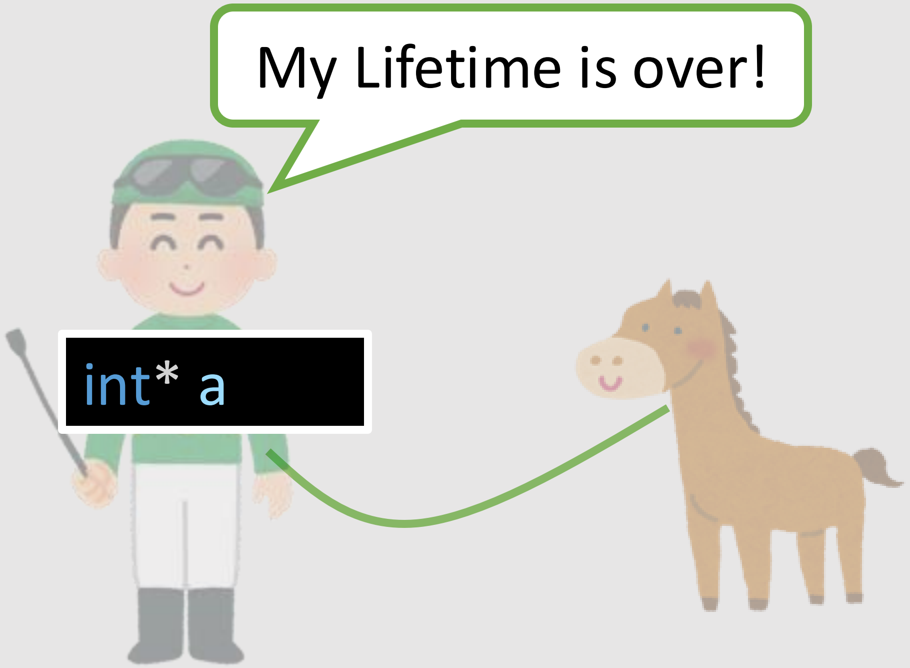
```
void main() {
  int* a = new int[32];
  a[0] = 1;
  delete[] a;
}
```





Resource Management

```
void main() {
  int* a = new int[32];
  a[0] = 1;
  delete[] a;
}
```



Problems in Manual Memory Management

Dangling pointer

```
void main() {
  int* a = nullptr;
  a[0] = 1;
  int* a = new int[10];
  delete[] a;
  a[0] = 1
}
```

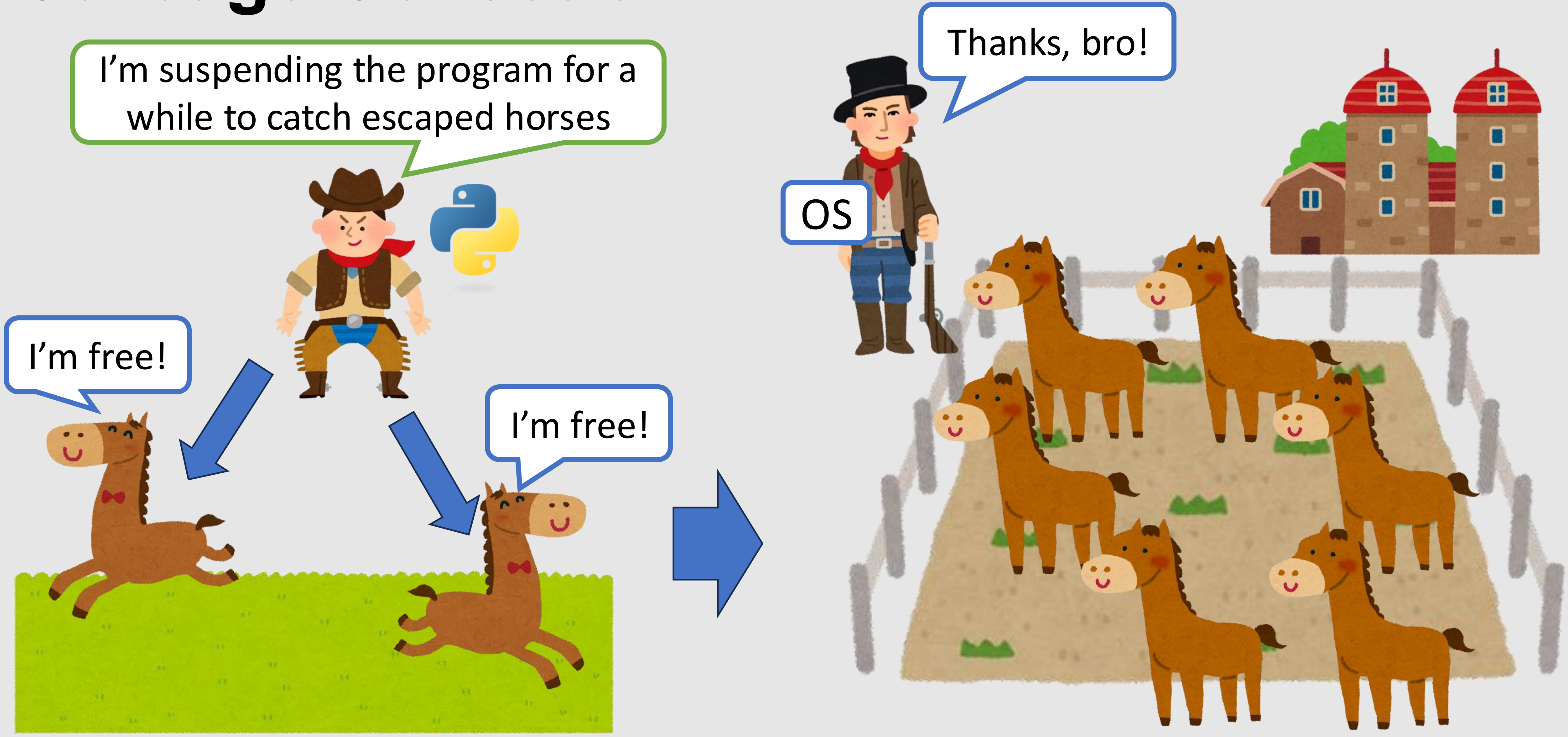
Multiple ownership

```
void main() {
  int* a = new int[10];
  int* b = a;
  delete[] a;
  delete[] b;
}
```

Memory leak

```
void main() {
  int* a = new int[10];
  a[0] = 1;
}
```

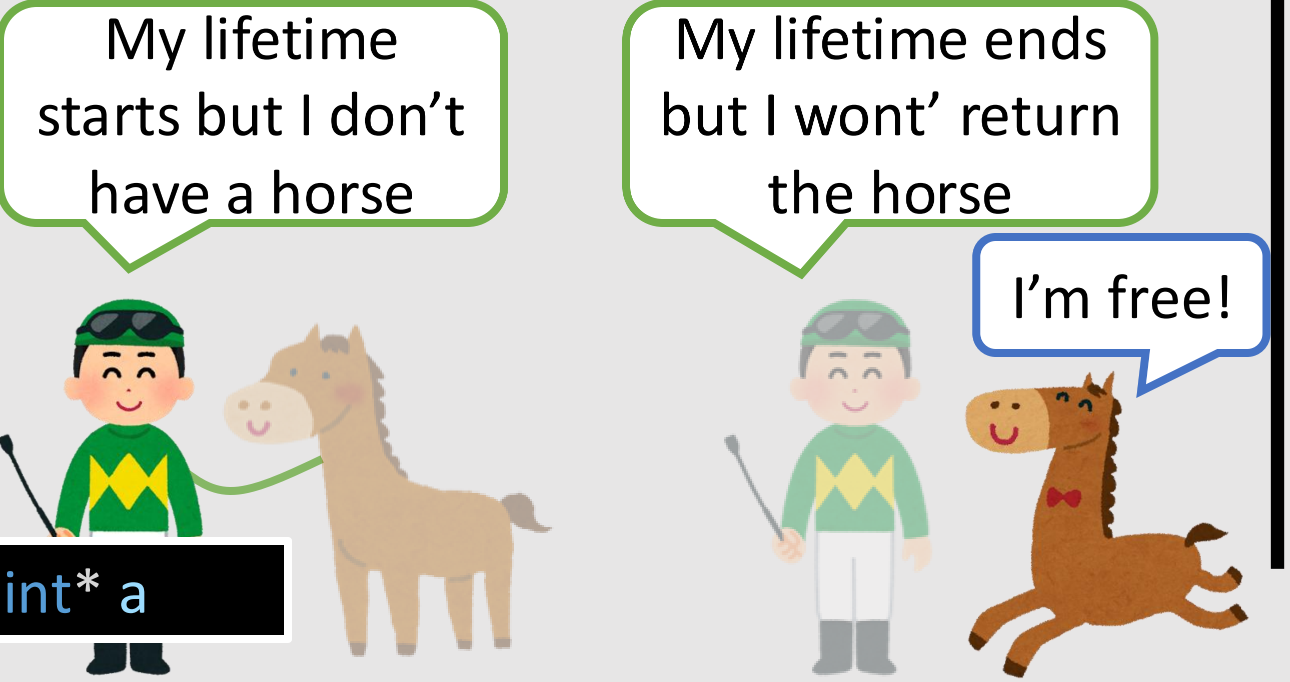
Garbage Collection


I’m suspending the program for a while to catch escaped horses
I’m free!
I’m free!
OS
Thanks, bro!

Resource Acquasition is Initialization (RAII)

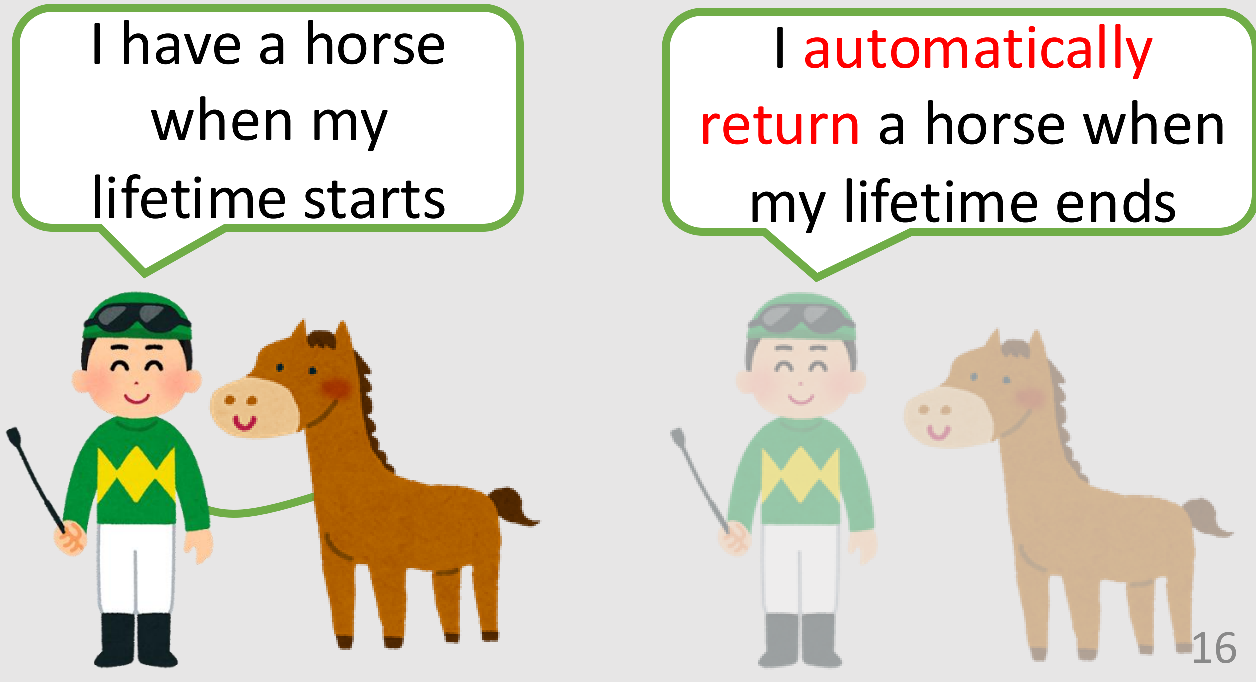
****Not** RAII***

```
void main() {
  int* a = nullptr;
  a = new int[10];
}
```

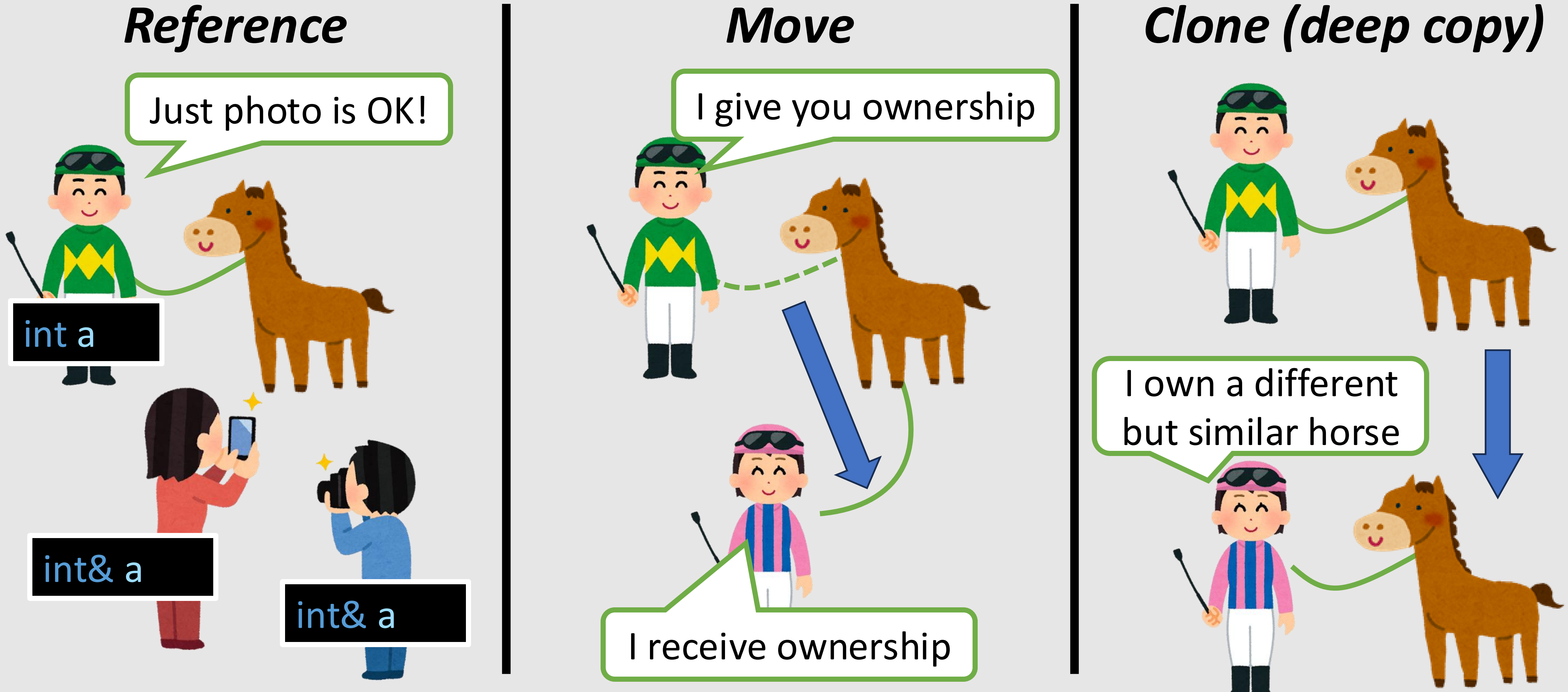

My lifetime starts but I don't have a horse
int* a
My lifetime ends but I wont' return the horse
I'm free!

RAII

```
#include <vector>
void main() {
  auto a = std::vector<int>(10);
}
```


I have a horse when my lifetime starts
I automatically return a horse when my lifetime ends

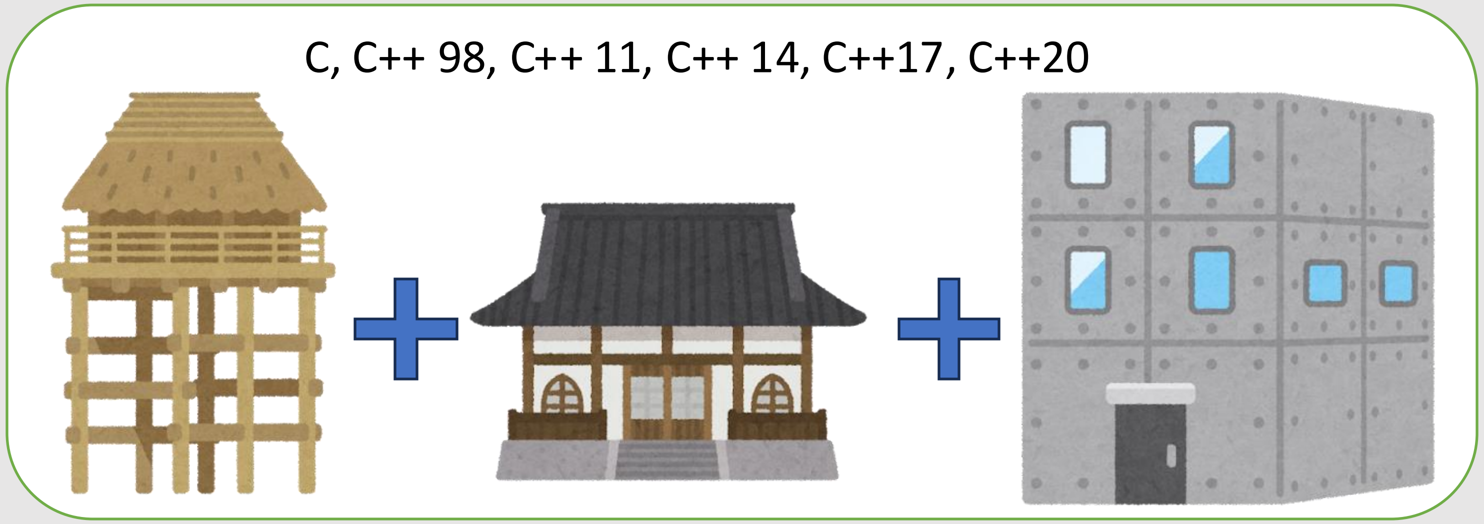
Differences in Reference, Move and Clone

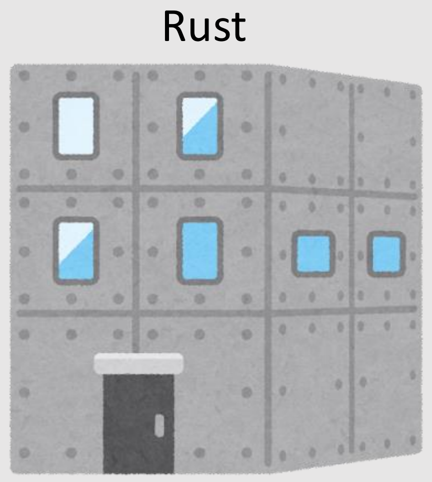


Rust

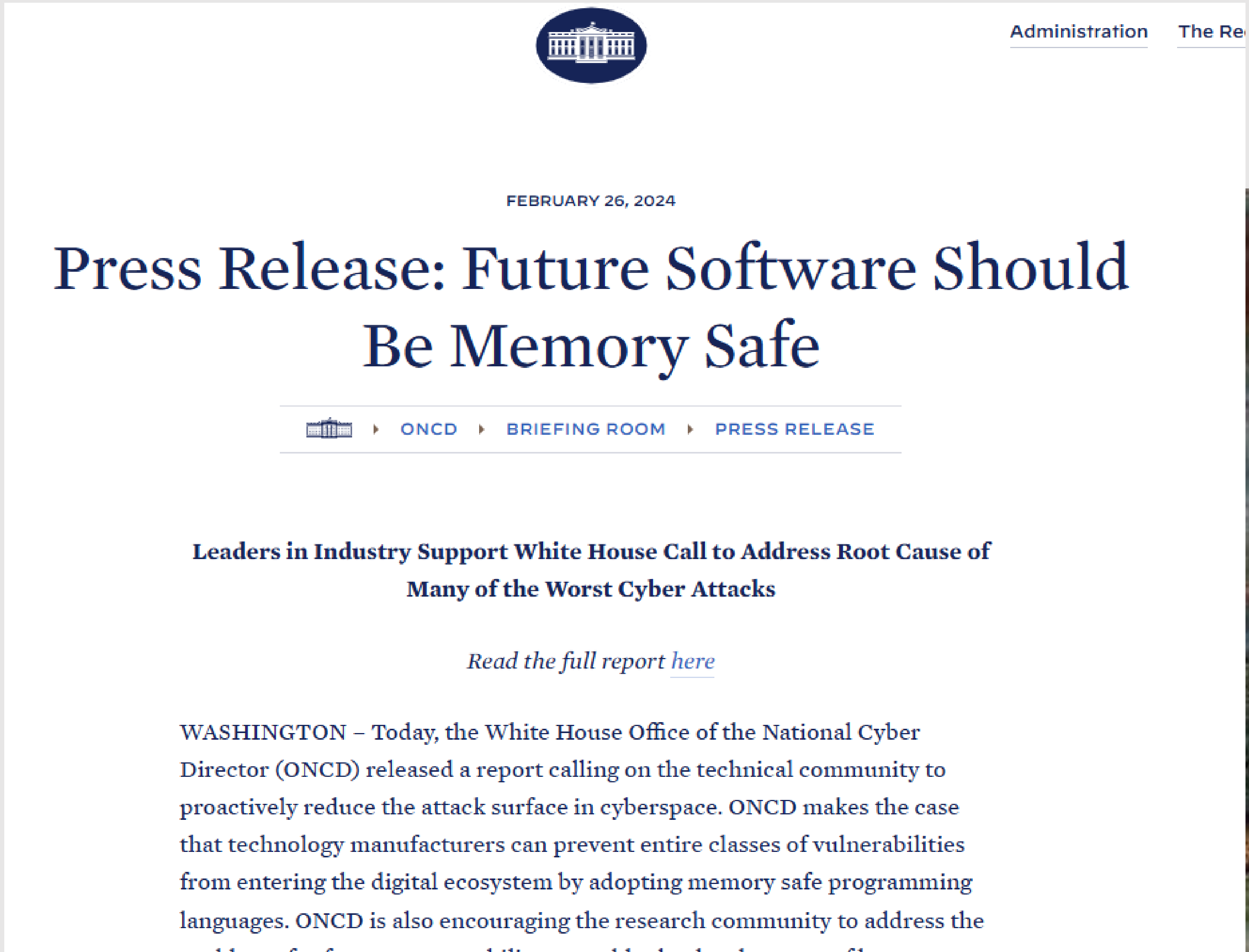
Why Rust (not C++)?

- Safety
  - Strictly implementing RAII
- Old languages just keep getting complecated for backward compatibility. Modern language can keep only the good stuffs in the old languages.





The White House Says You Shoud Use Rust



Administration The Re

FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

ONCD › BRIEFING ROOM › PRESS RELEASE

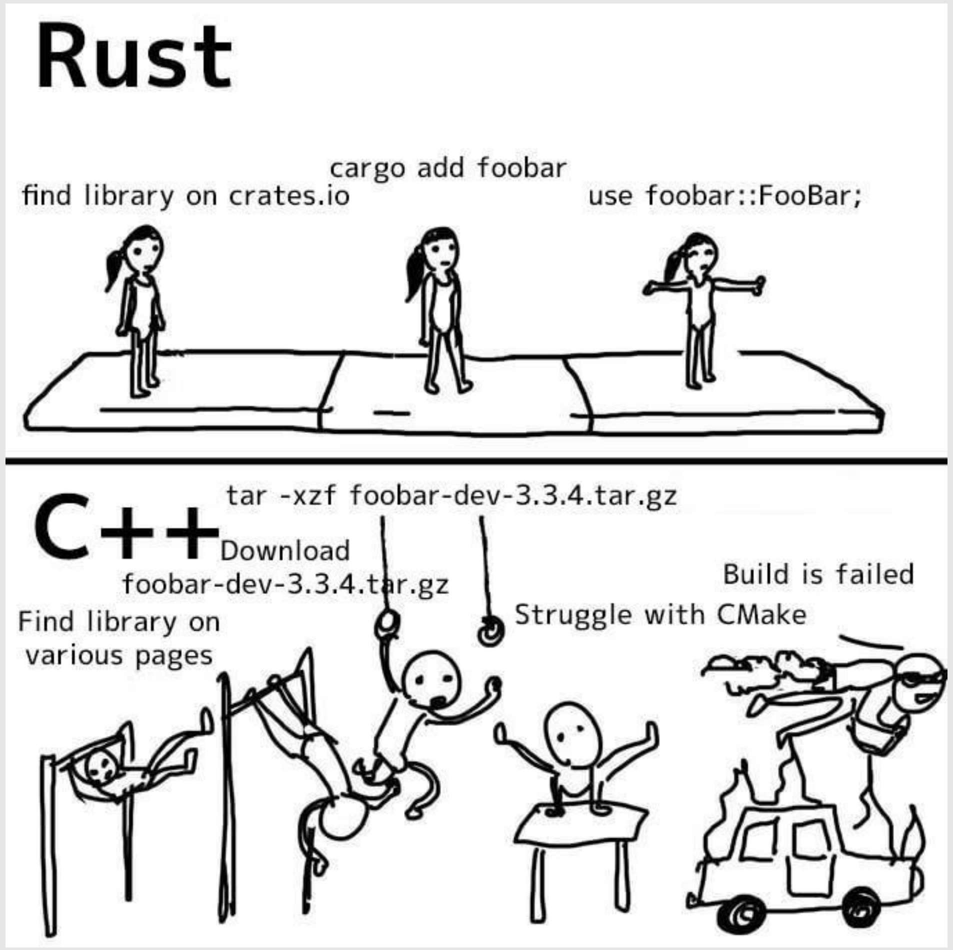
Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report here

WASHINGTON – Today, the White House Office of the National Cyber Director (ONCD) released a report calling on the technical community to proactively reduce the attack surface in cyberspace. ONCD makes the case that technology manufacturers can prevent entire classes of vulnerabilities from entering the digital ecosystem by adopting memory safe programming languages. ONCD is also encouraging the research community to address the

Why Rust (not C++)?

- It is easy to use other libraries (a.k.a. crates)


Rust
find library on crates.io
cargo add foobar
use foobar::FooBar;
C++
tar -xzf foobar-dev-3.3.4.tar.gz
Download foobar-dev-3.3.4.tar.gz
Find library on various pages
Struggle with CMake
Build is failed

https://www.reddit.com/r/ProgrammerHumor/comments/1hnfuvk/why
idliketoavoidusingcpp/?rdt=41480

Grammer Basics: Primitive Types

- Rust type name is short, but explains its size on memory

C/C++	Rust
int	i32
unsigned int	uint32
unsigned char	u8
float	f32
double	f64

Grammer Basics: Declaring Variables

Declaring variable

type

```
let a: f32 = 1.;
let b = 2i32;
```

I own this horse,
and I can just look at it

Declaring mutual variable

```
let mut a = 1u32;
a += 1;
```

I own this horse
& I can ride on it

Grammer Basics: Static/Dynamic Array

```
let b: [f32;2] = [5.0, 6.0];

let idx: usize = 0;
let b0 = b[idx];


let c: Vec<u32> = vec![1.0, 2.0];
let d = vec![3u32; 100];
```

Declaration of a static array

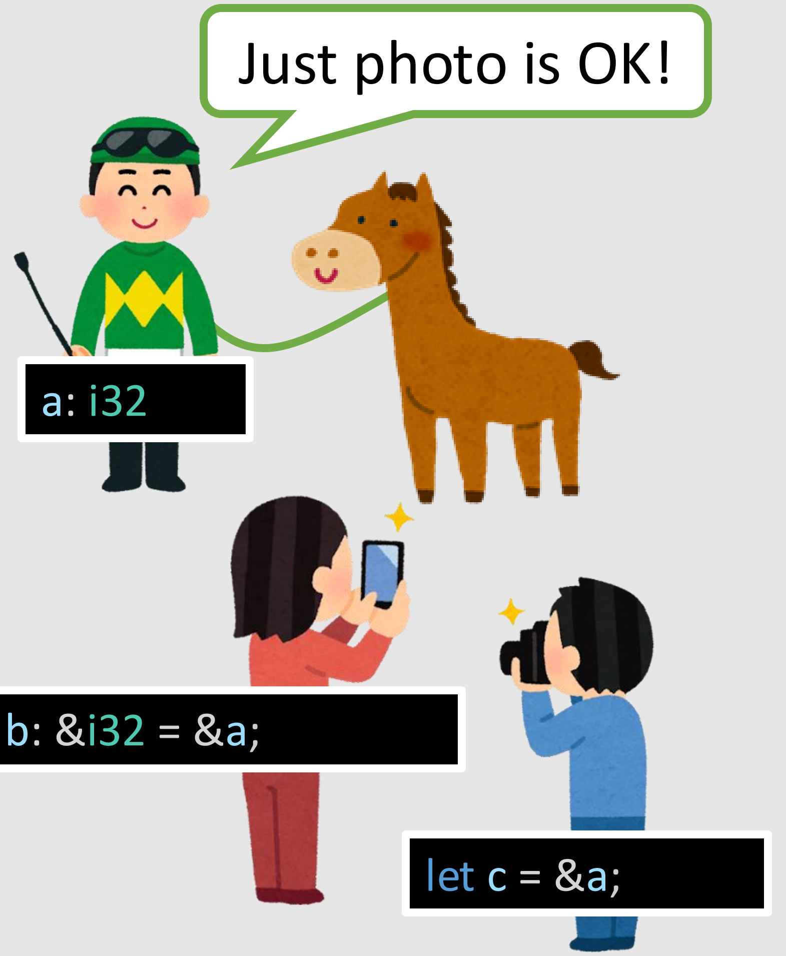
Index of the array should has `usize` type.
usize is 64-bit in 64 bit OS.

Declaration of a dynamic array

Another declaration of a dynamic array

Grammer Basics: Reference & Slice

Reference



```
a: i32
```

```
et b: &i32 = &a;
```

```
let c = &a;
```

Mutable reference

```
a: i32
```

```
let c = & mut a;
```

Slice (reference to array)

```
let b = vec![1.0, 2.0];
let c: &[f32] = &b;
```

This reference is called ”slice”

```
dbg!(c.len()); // 2
```

Slice has length

Grammer Basics: Functions

```
fn one_up(a: &mut u32) {
  a += 1;
}

let b = 1;
one_up(&mut b);
dbg!(b); // 2
```

Declaration of a function

Explicitly giving mutual reference to the function

How to use Rust

Project structure

```
Project_folder\
 ├ src\
 |   └ main.rs
 └Cargo.toml
```

main.rs

```rust
fn main() {
    println!("Hello, world!");
}
```

Cargo.toml

```toml
[package]
name = "task00"
version = "0.1.0"
edition = "2021"

[dependencies]
anyhow = "1.0.97"
del-canvas = “0.1.3”
```

Running project

```
➢ cargo build

➢ cargo run
=
➢ cargo fmt

> cargo clippy
```

- cargo build — Build project
- cargo run — Build & run project
- cargo fmt — Format code
- cargo clippy — Improve code

Integrated Development Environment (IDE)

- Code editor with linter, suggestion, jumps
- Static program analysis
- Debugger

Visual Studio Code

RustRover