Dear Reader,

I’m not really into lots of bulleted lists. Just not my thing. I like to make slides that are pictures I can point at and talk about. Many of the slides don’t stand on their own well without the pointing and talking bit.

But since you are reading my slides, I propose a game: Without watching my talk first, make up your own story about Rust and concurrency to go along with these slides. Then watch the video recording of my talk released by Strange Loop and compare. Have fun!

Love,

David

Level up your concurrency skills with Rust

Strange Loop 2017

David Sullins

@dvdsllns

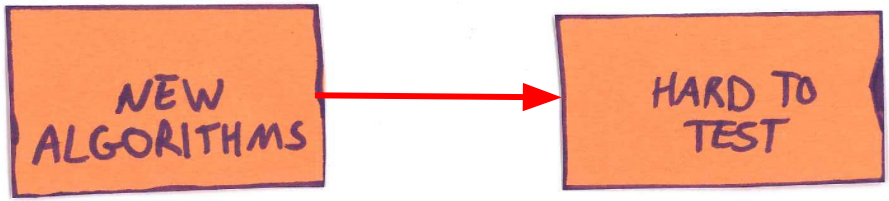
NVIDIA

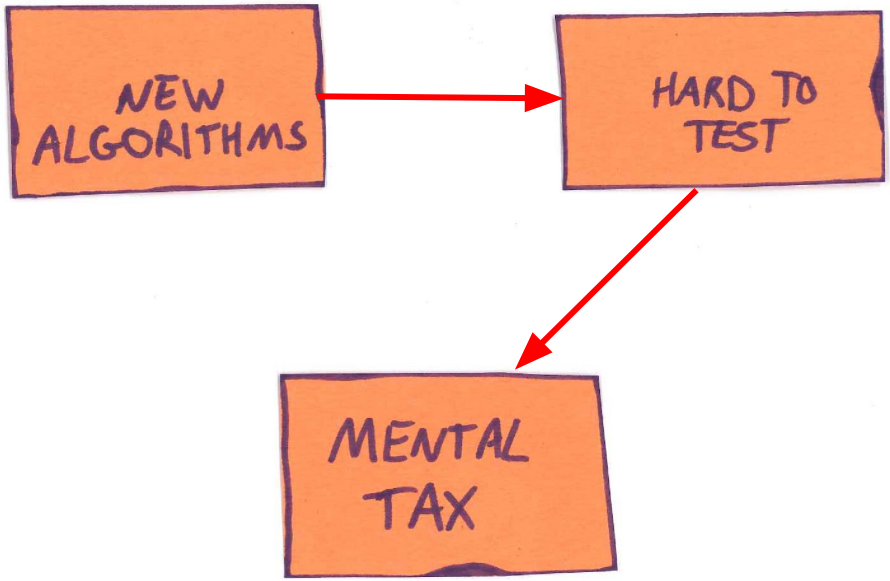
C++

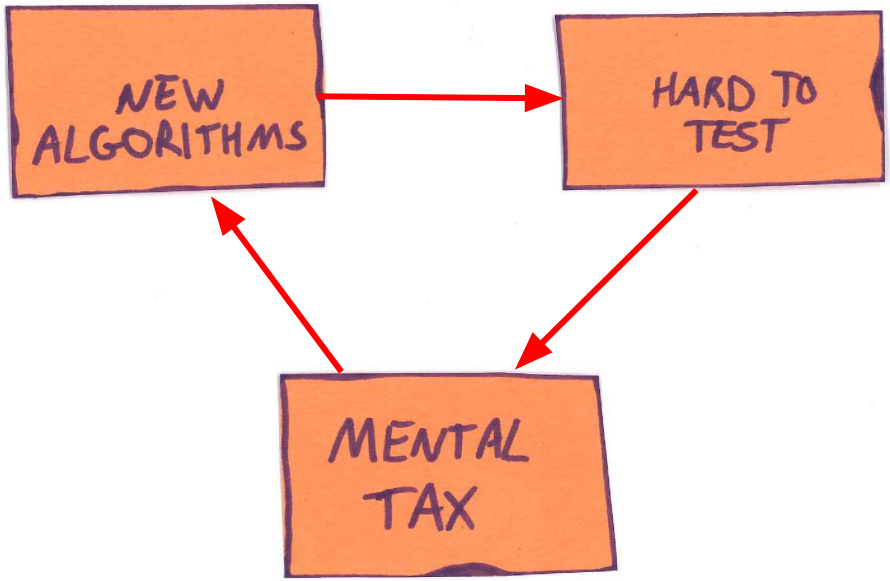
Why Rust?

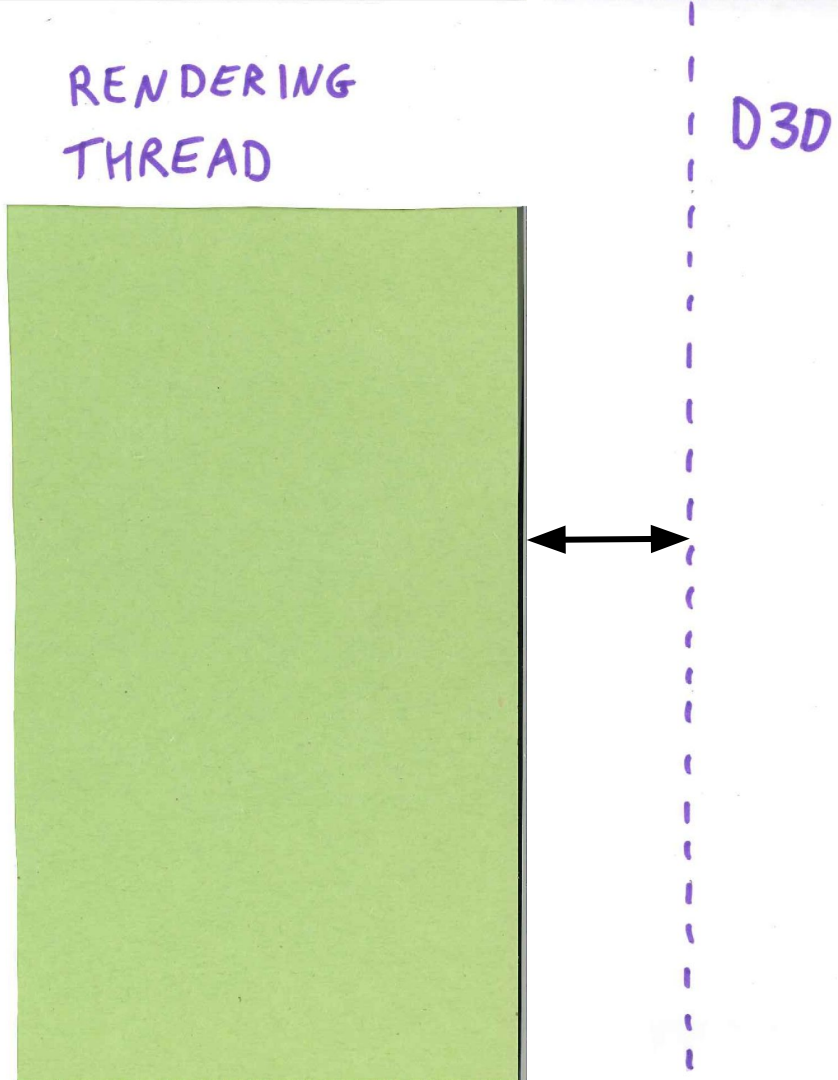
Why concurrency?

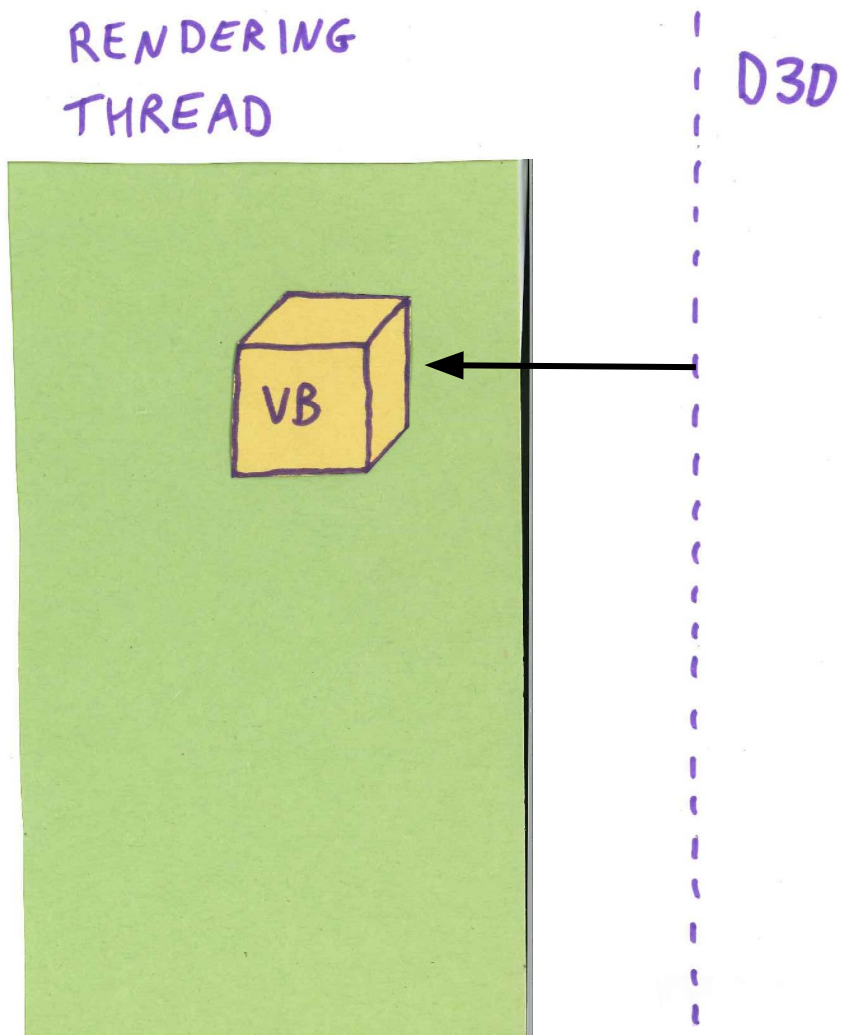

NEW
ALGORITHMS

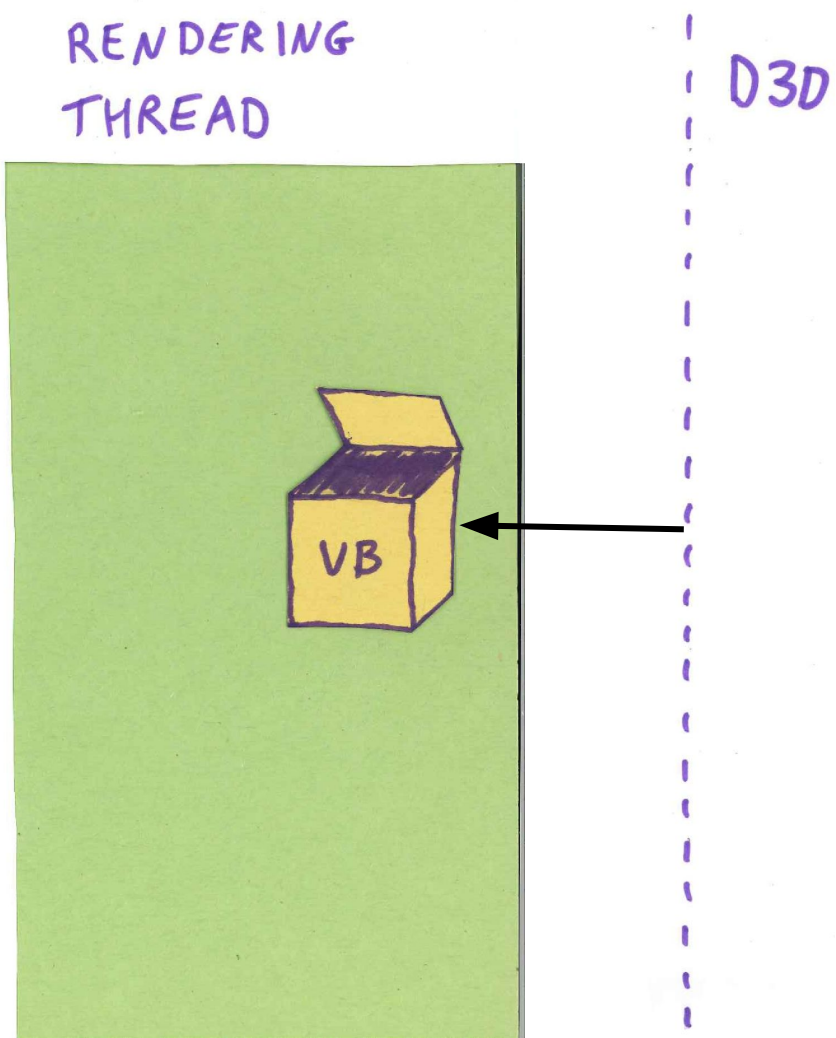

NEW ALGORITHMS
HARD TO TEST


NEW ALGORITHMS
HARD TO TEST
MENTAL TAX


NEW ALGORITHMS
HARD TO TEST
MENTAL TAX

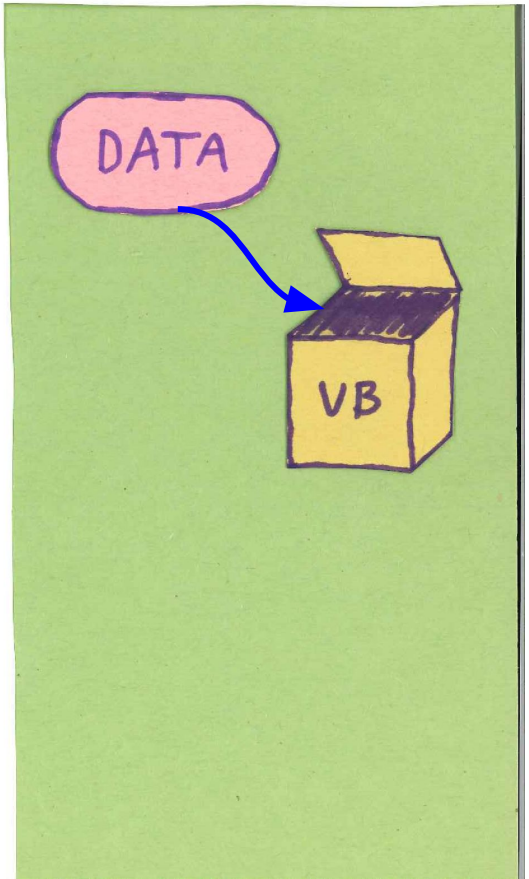

RENDERING
THREAD
D3D

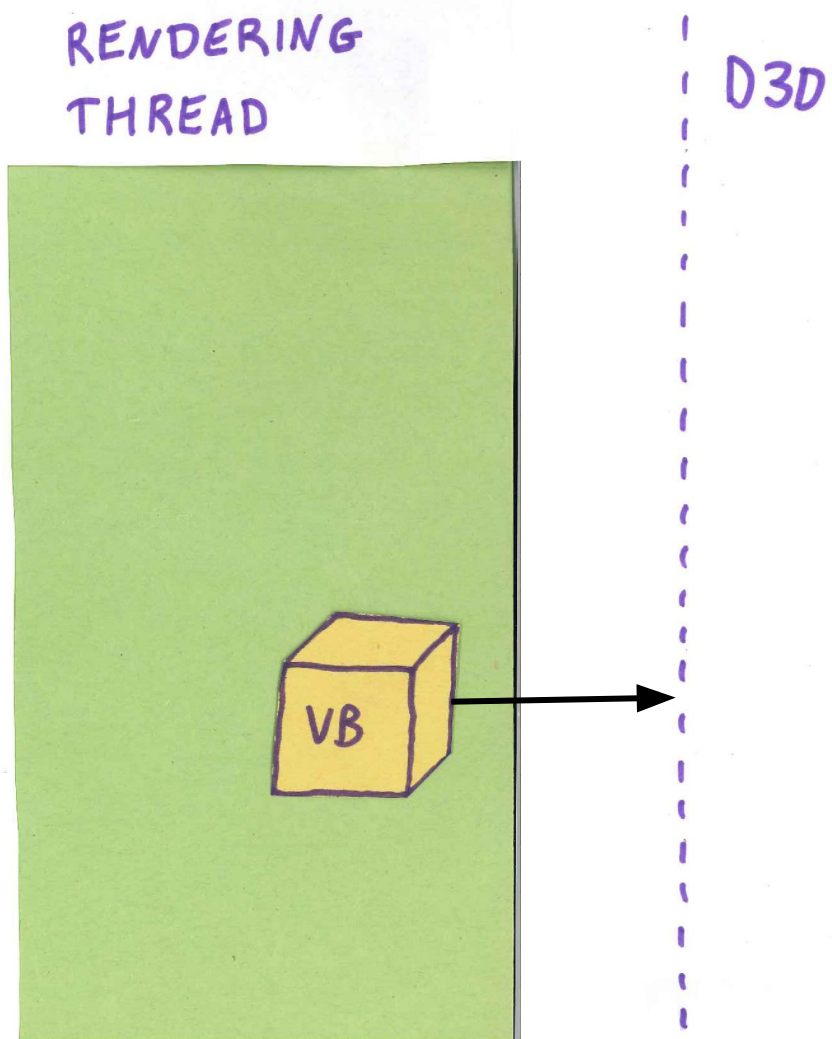

RENDERING
THREAD
D3D
VB

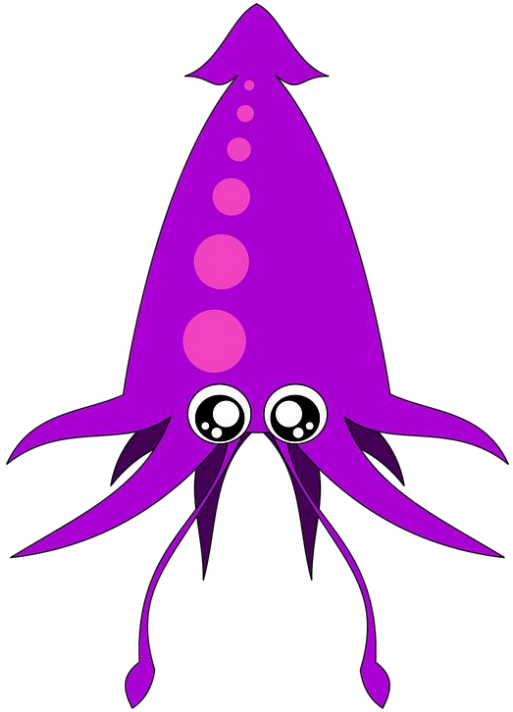

RENDERING
THREAD
D3D
VB

RENDERING THREAD

D3D

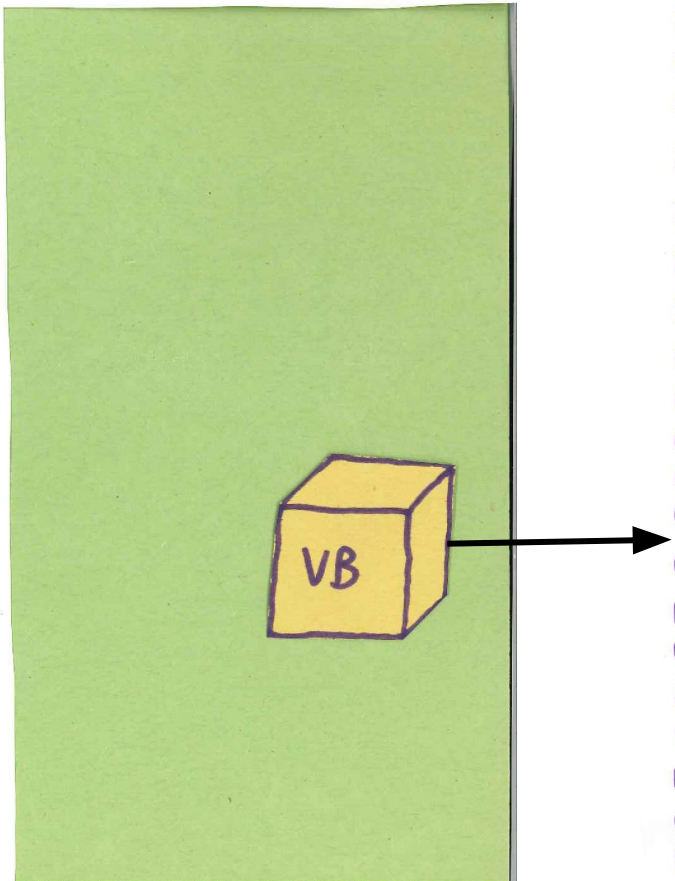


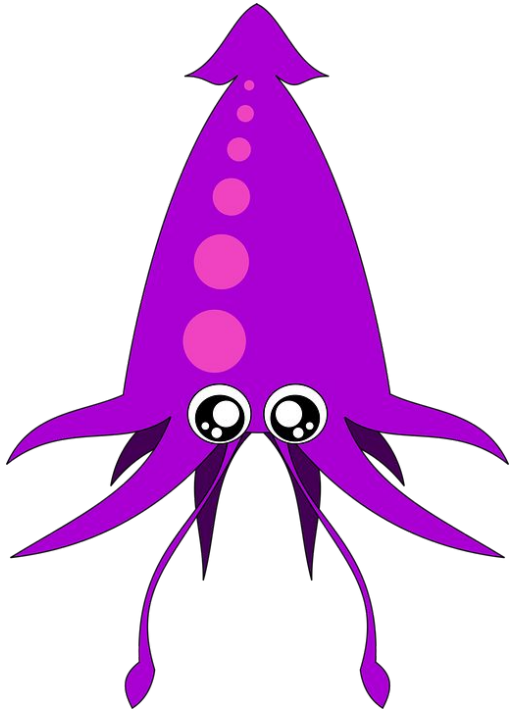

RENDERING
THREAD
D3D
VB



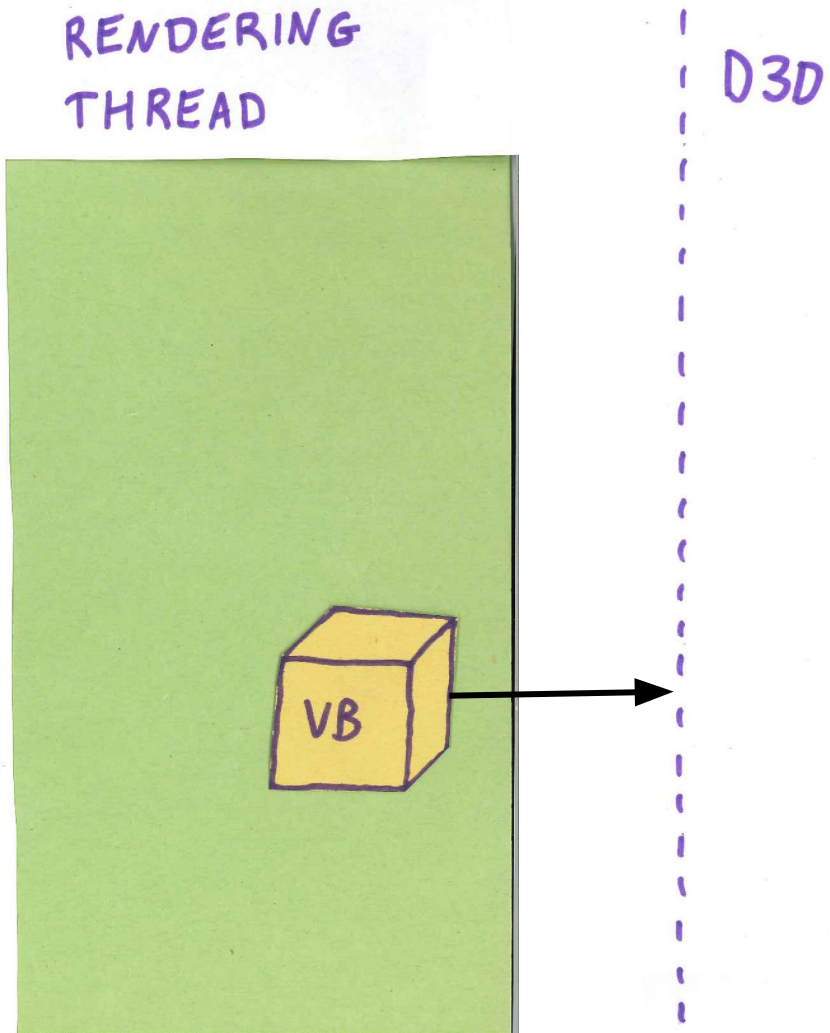
RENDERING
THREAD

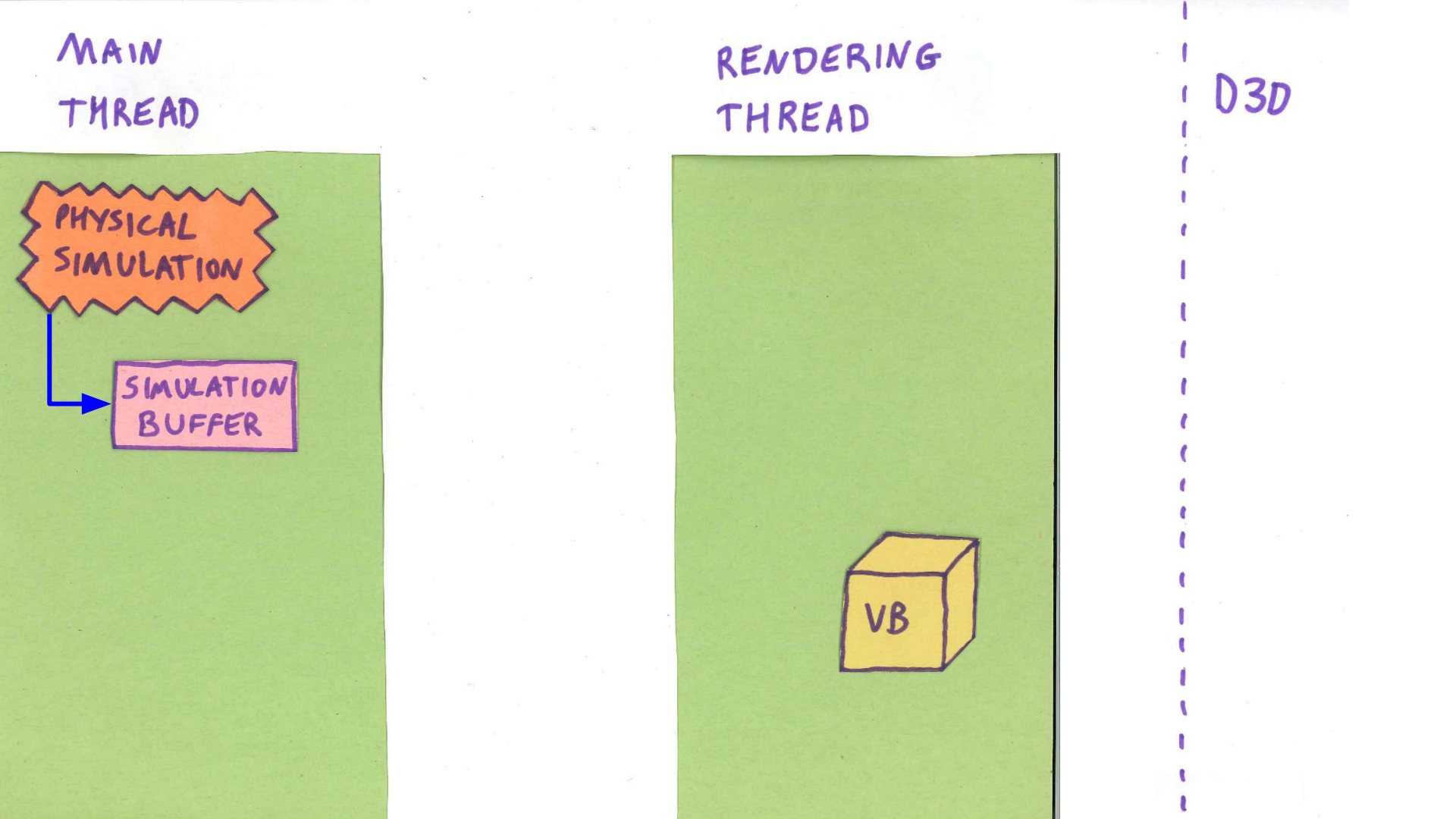
D3D

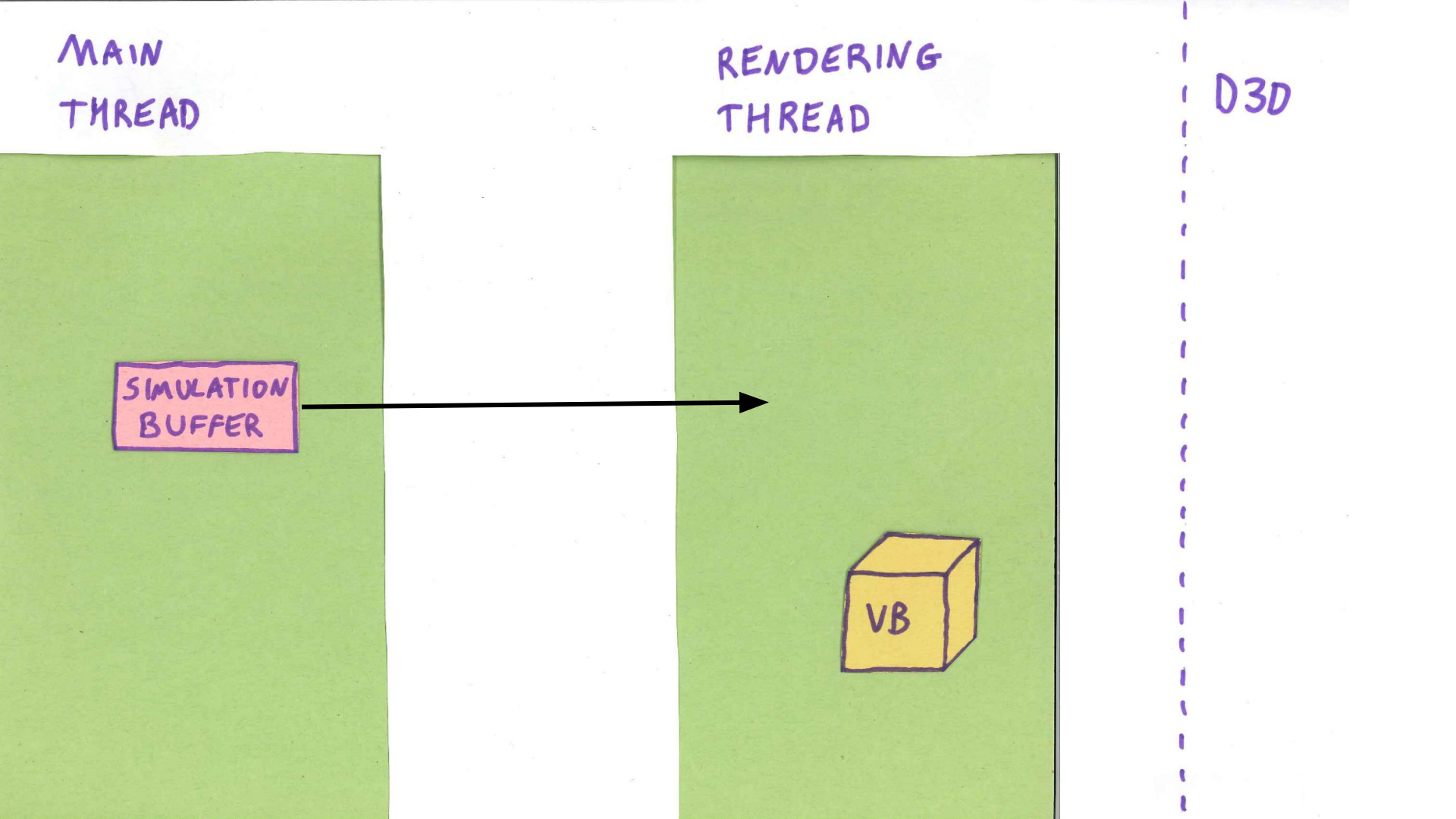




NOT AN ACTUAL SCREENSHOT



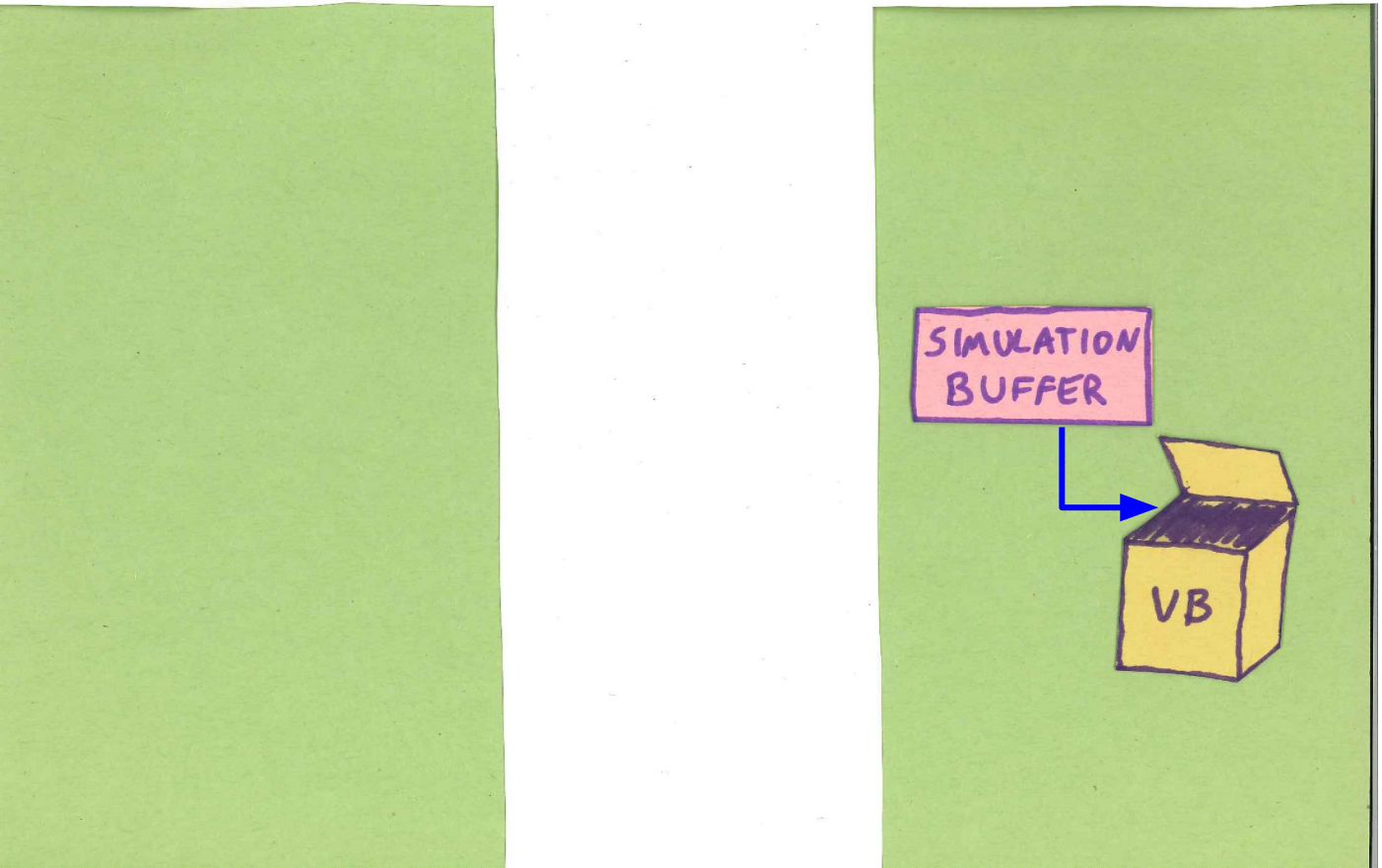

MAIN
THREAD
PHYSICAL
SIMULATION
SIMULATION
BUFFER
RENDERING
THREAD
VB
D3D


MAIN
THREAD
RENDERING
THREAD
D3D
SIMULATION
BUFFER
VB

MAIN THREAD

RENDERING THREAD

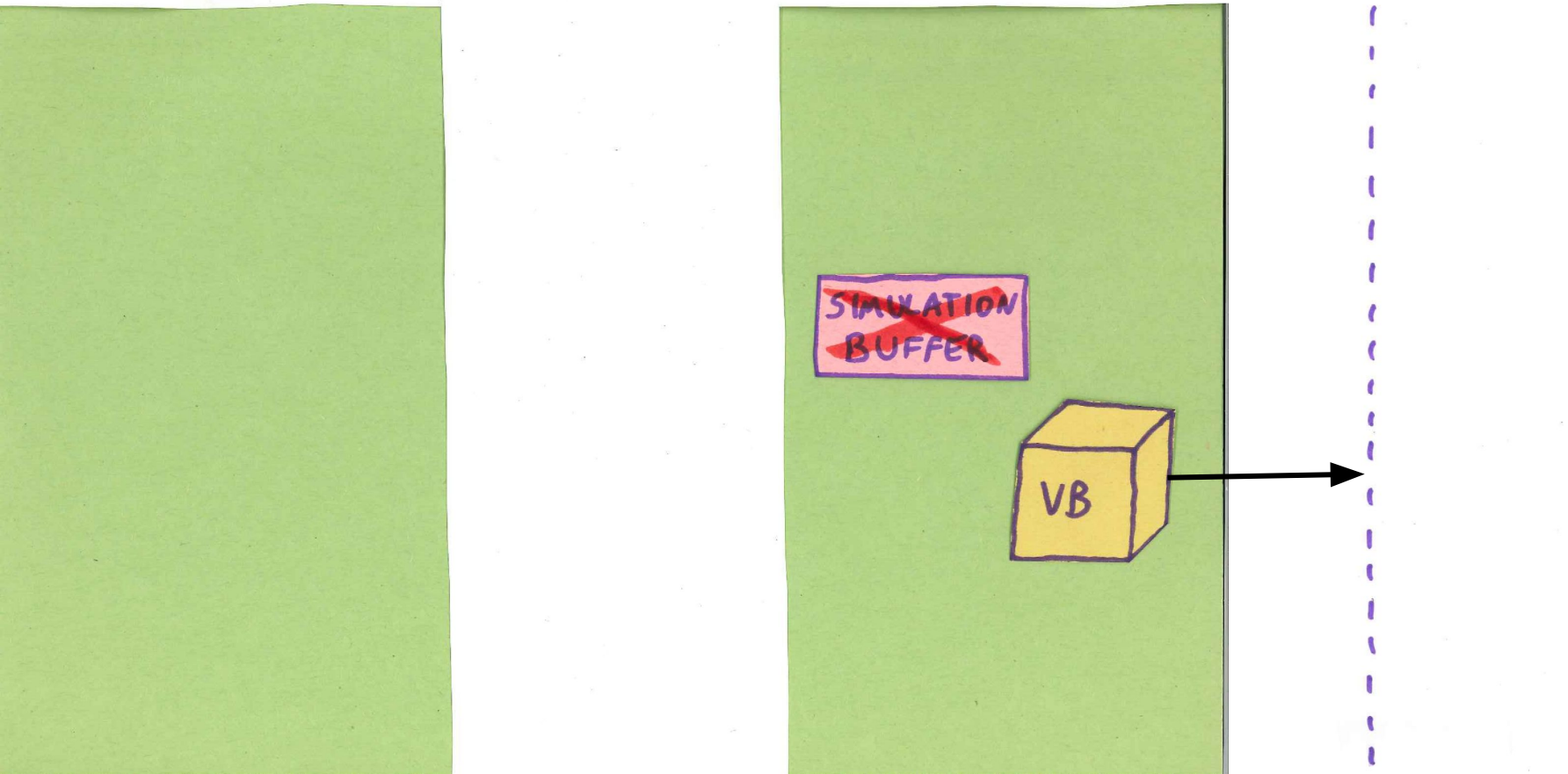
D3D



MAIN THREAD

RENDERING THREAD

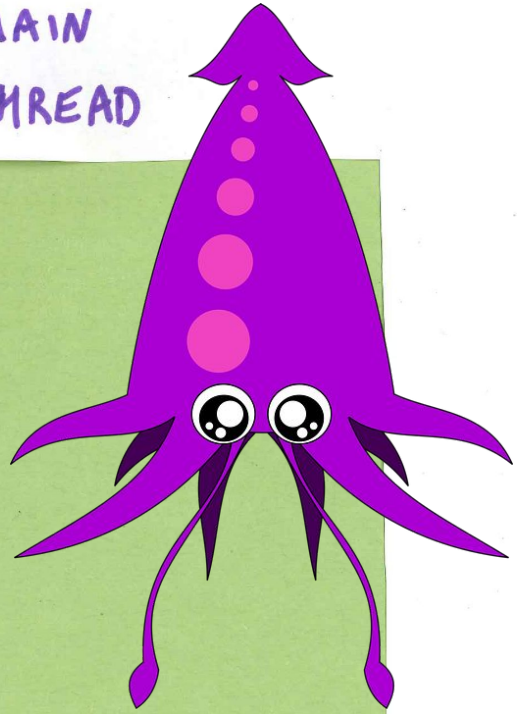
D3D



MAIN THREAD

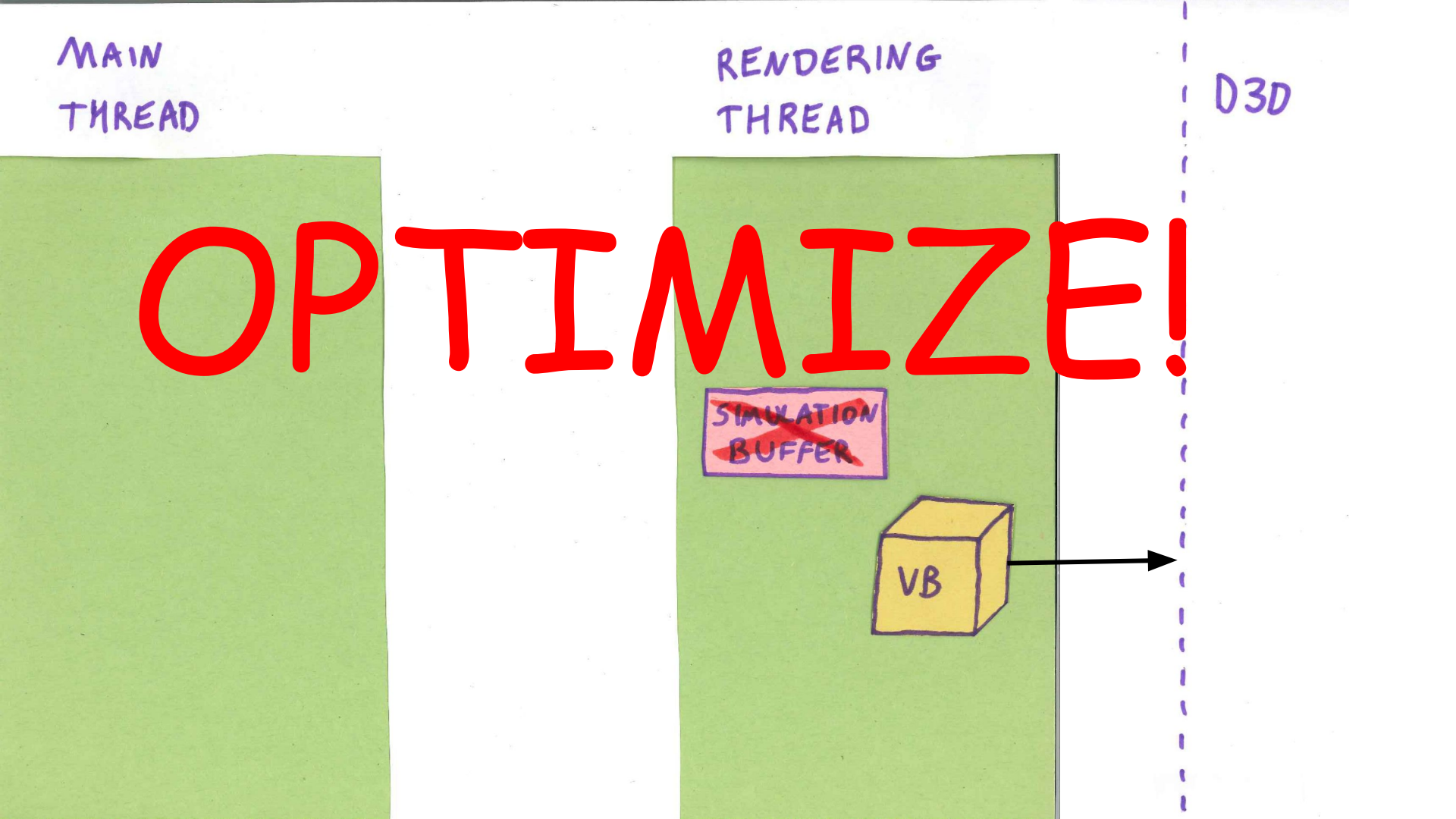
RENDERING THREAD

D3D



~~SIMULATION BUFFER~~

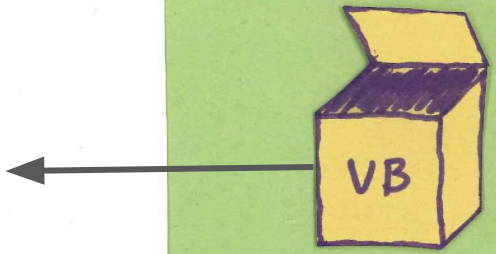
VB


MAIN THREAD
RENDERING THREAD
D3D
OPTIMIZE!
SIMULATION BUFFER
VB

MAIN THREAD

RENDERING THREAD

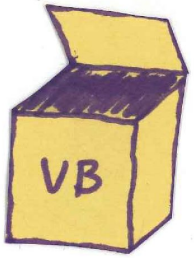
D3D

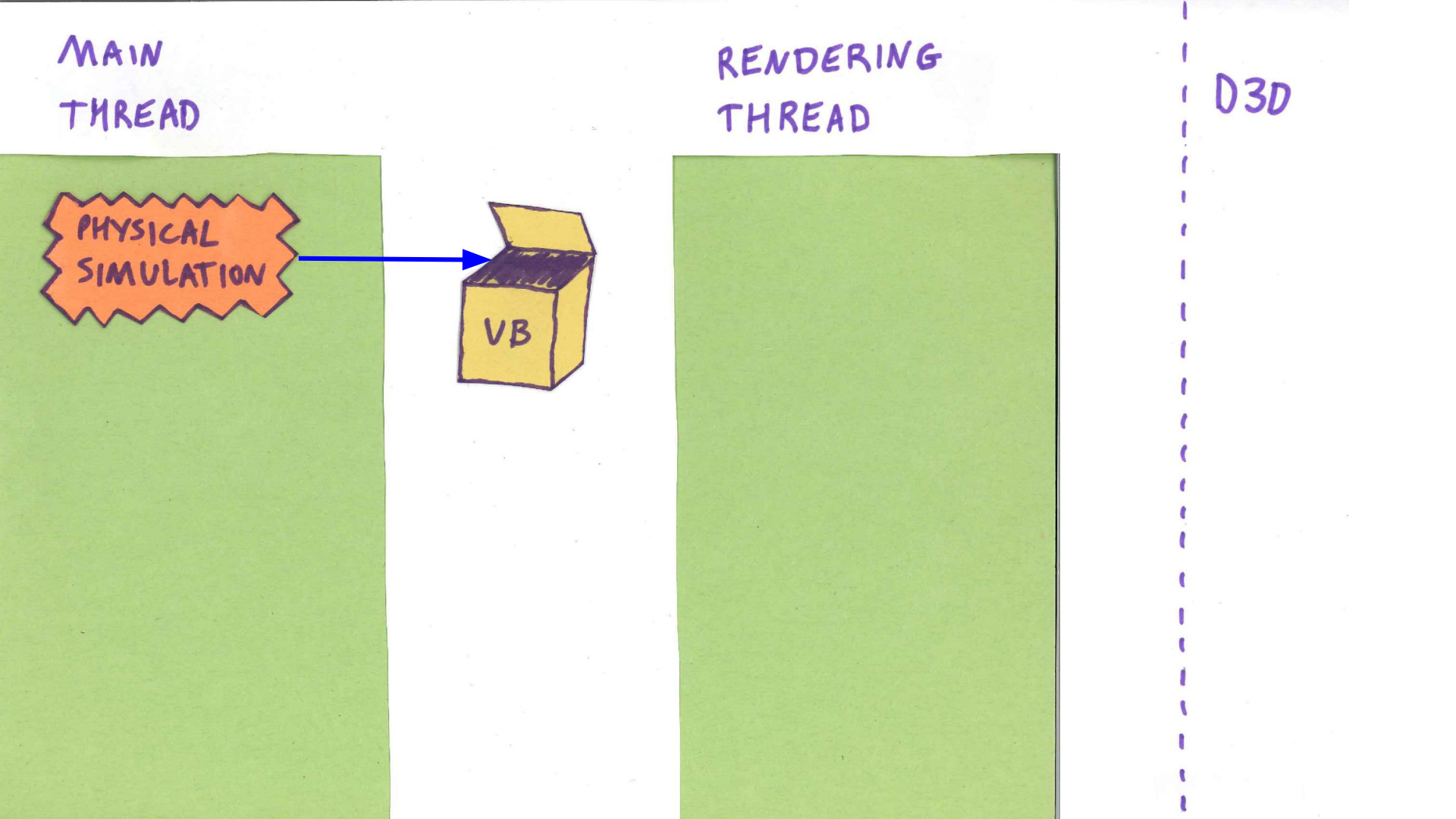


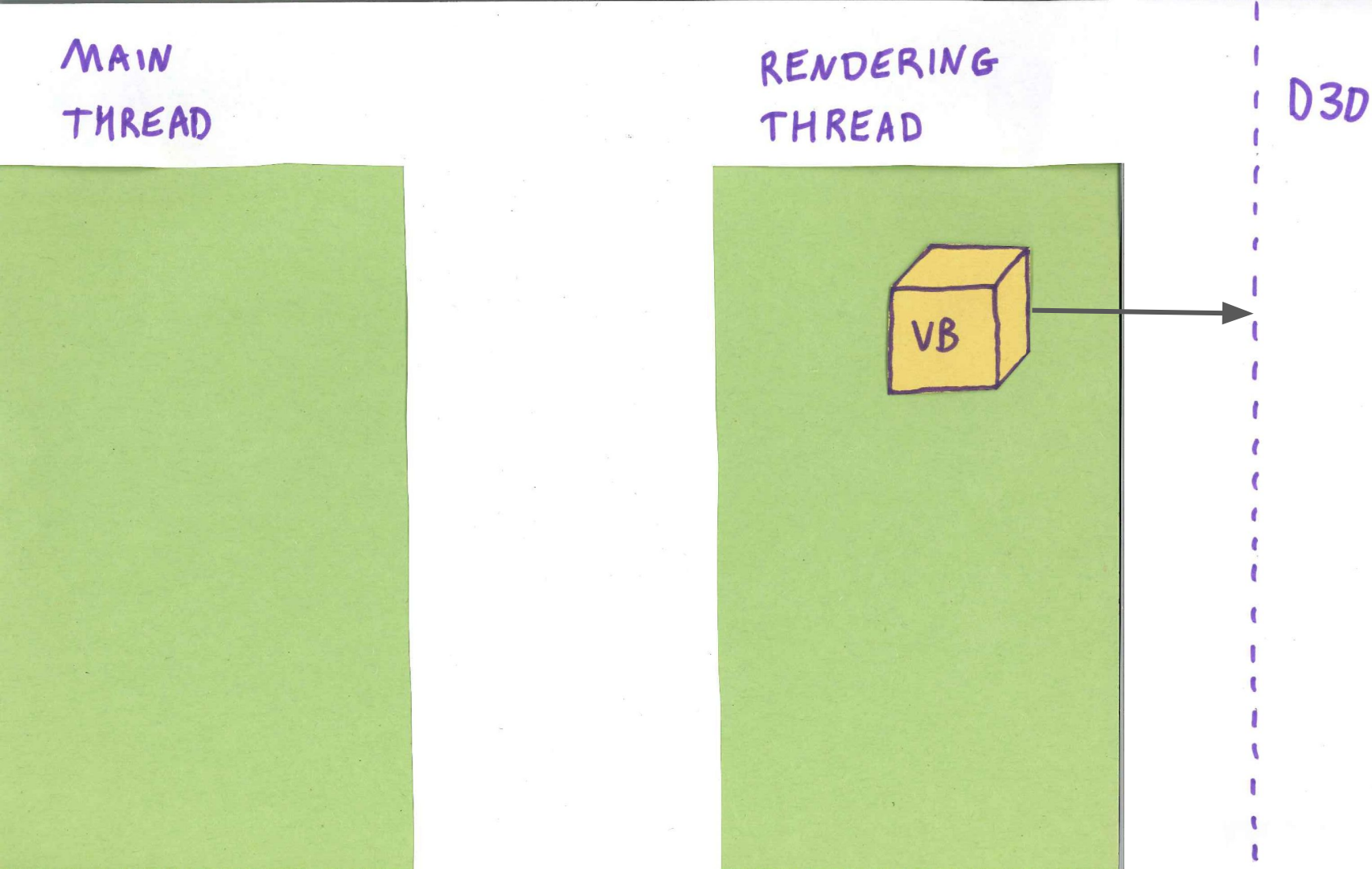
MAIN THREAD

RENDERING THREAD

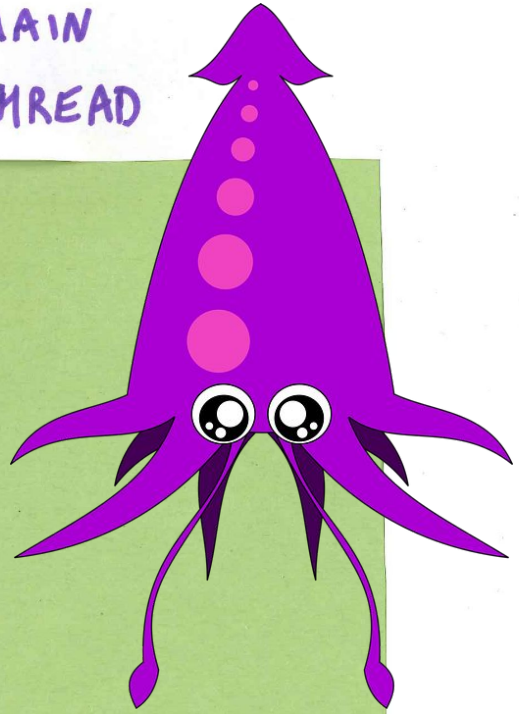
D3D




MAIN
THREAD
RENDERING
THREAD
D3D
PHYSICAL
SIMULATION
VB


MAIN
THREAD
RENDERING
THREAD
D3D
VB

MAIN THREAD



RENDERING THREAD

VB

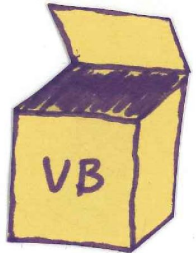
D3D

MAIN THREAD

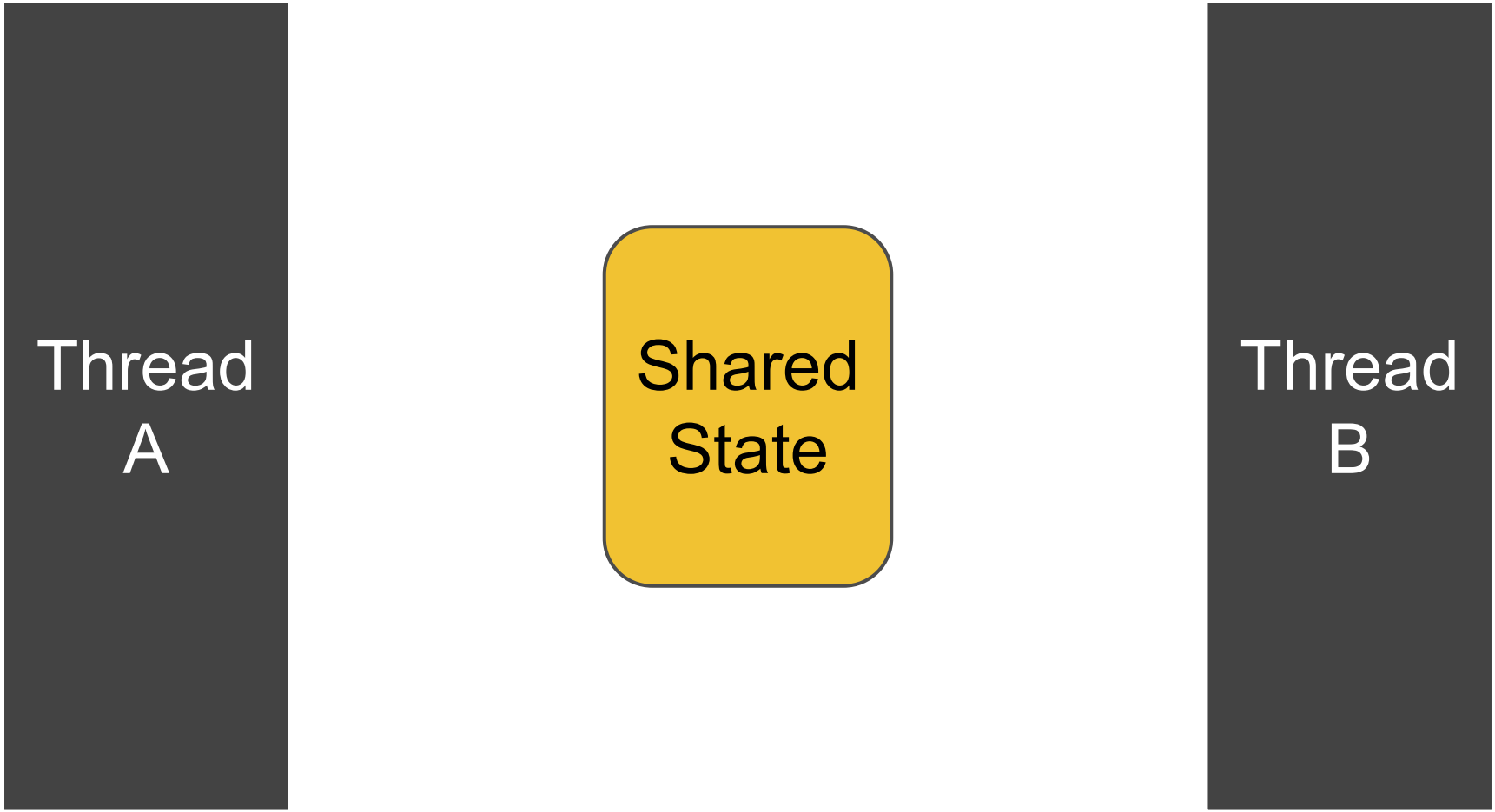
FAIL

RENDERING THREAD

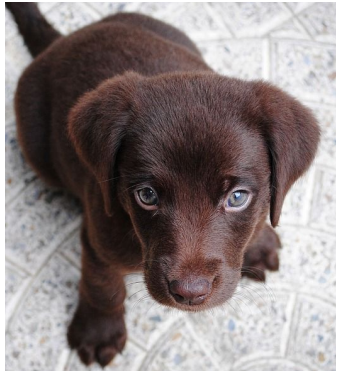
D3D



DATA RACE


Thread A
Shared State
Thread B

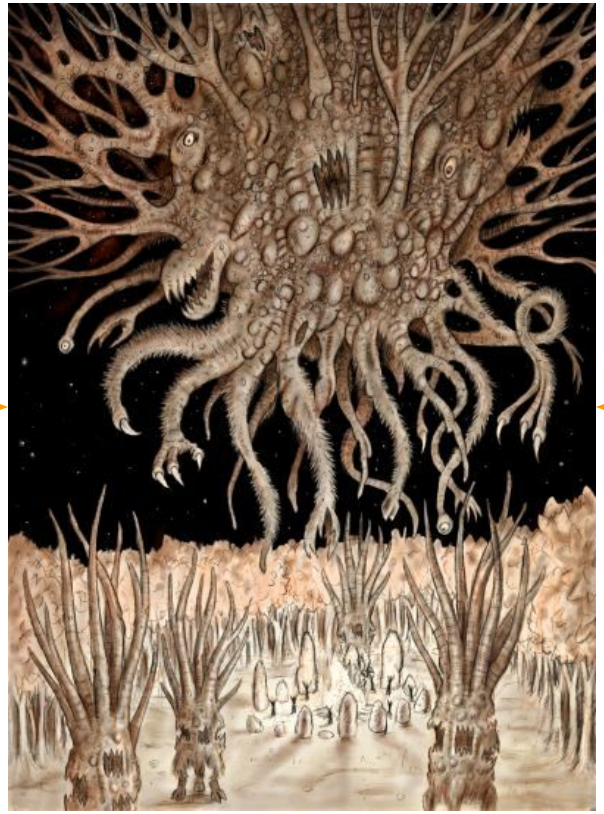
A



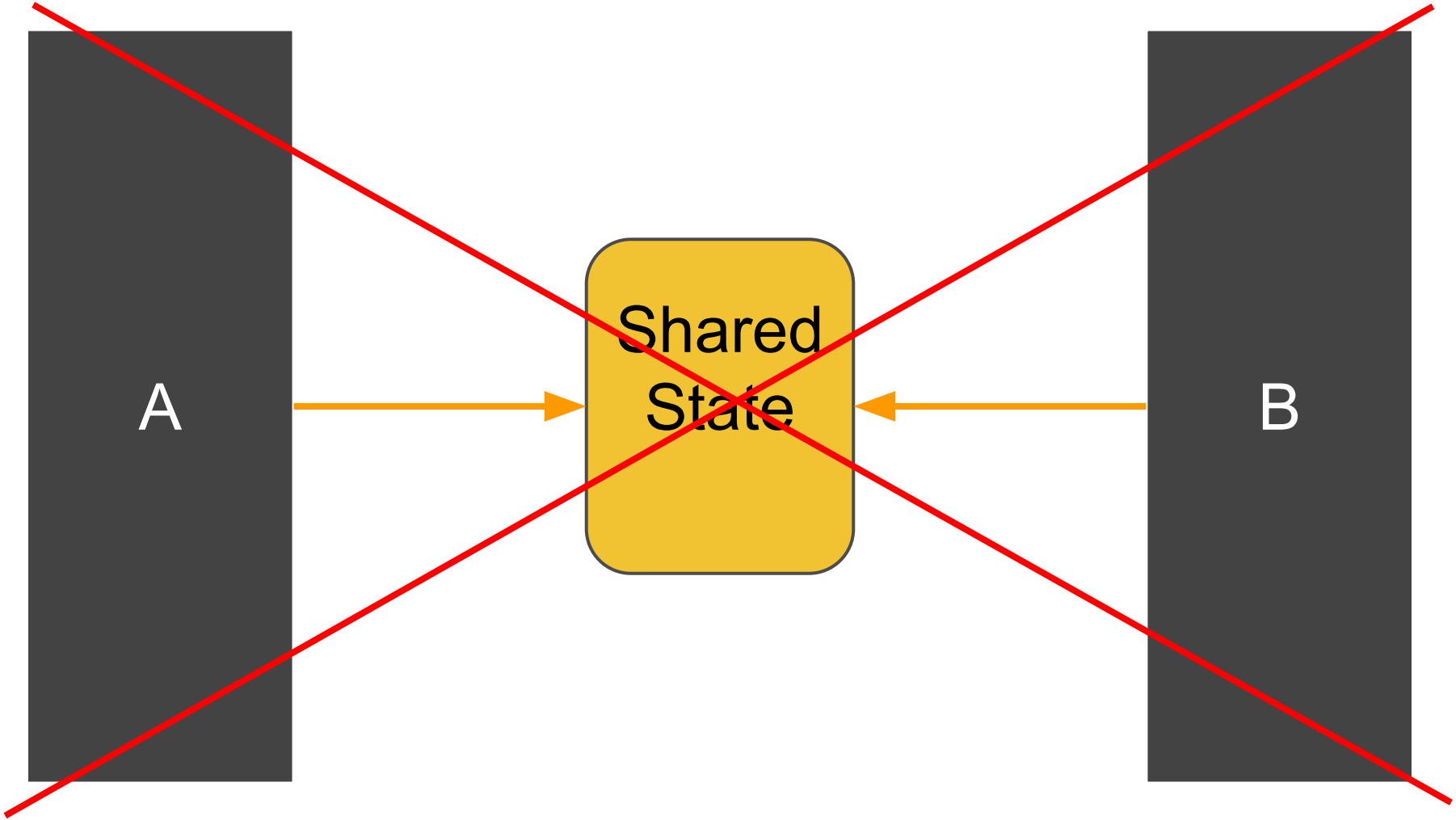
B

AAAAAAA!

A



B


A
Shared State
B

A

strangeloop

B

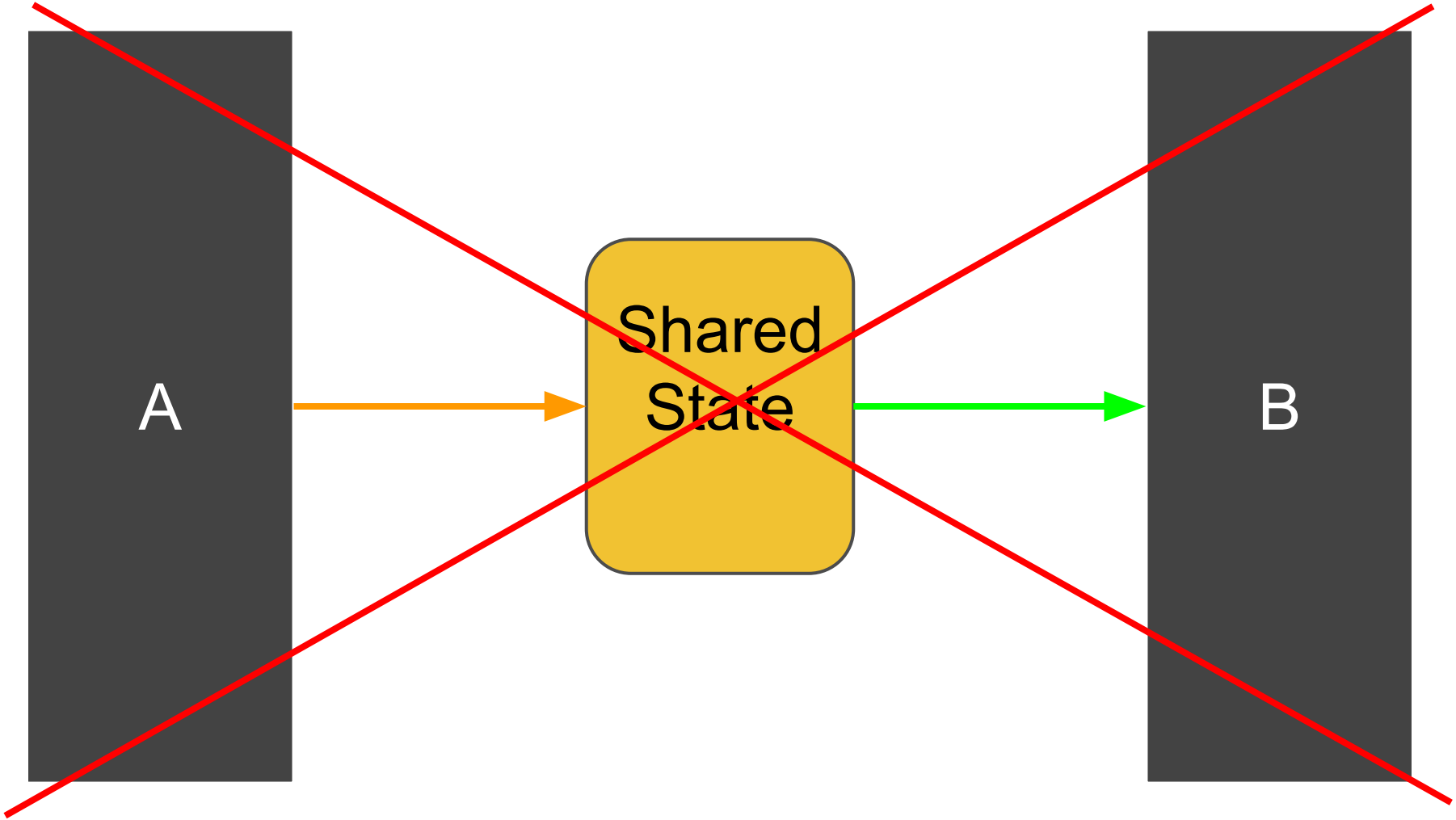
A

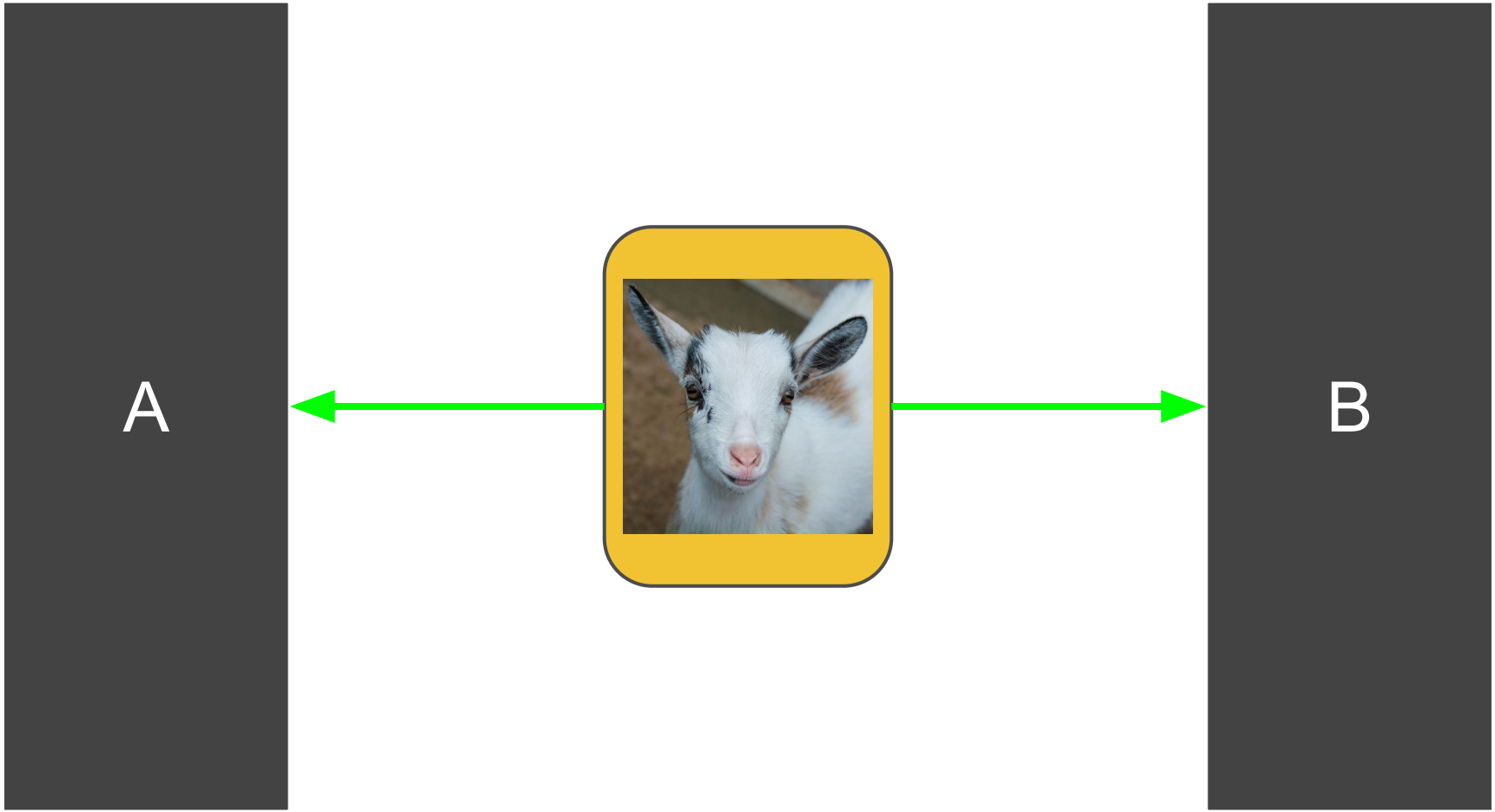
B

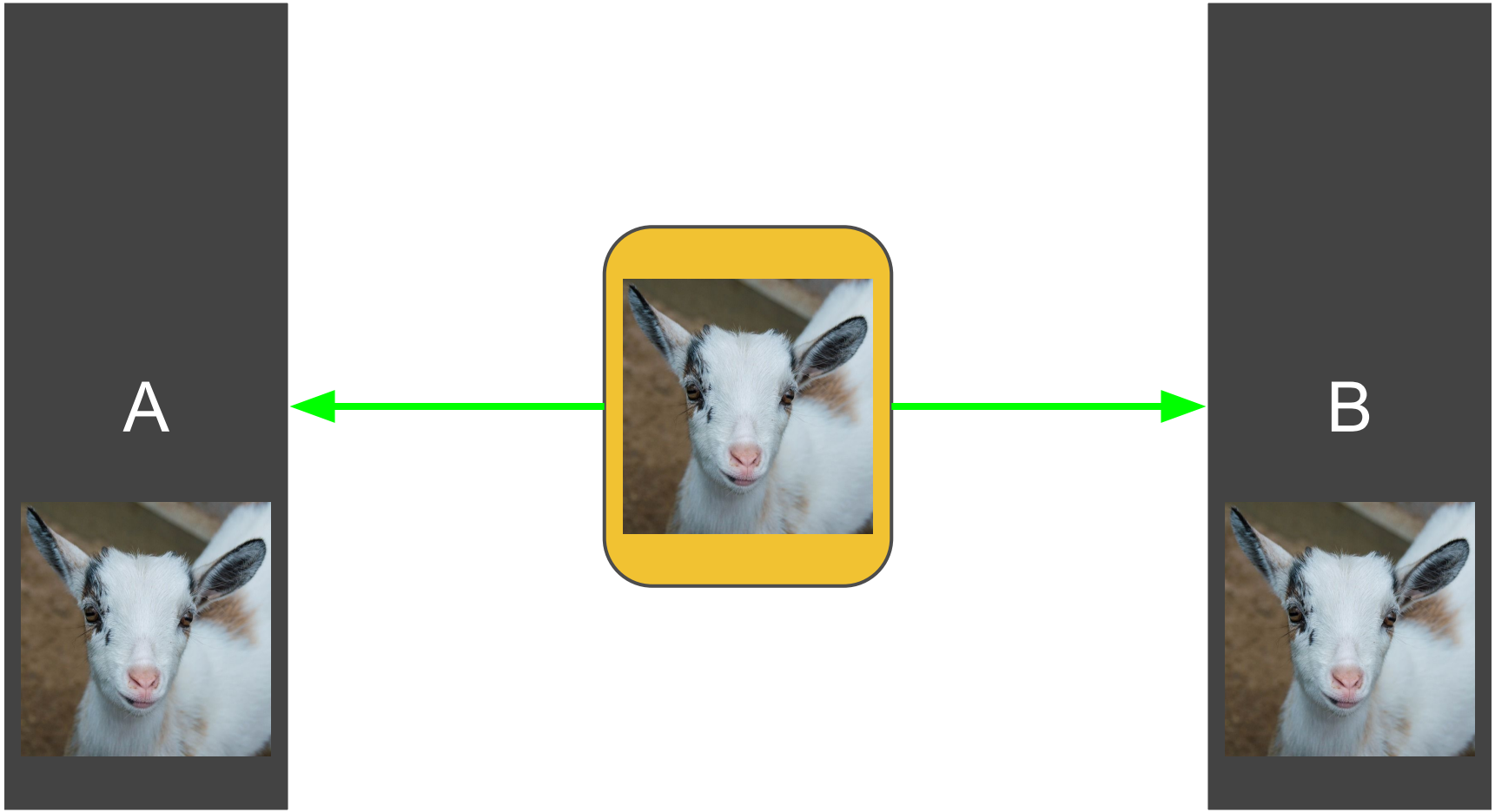
A

B

Atomic: can be accessed by 1 thread with no possibility of another thread encountering an inconsistent state


A
Shared State
B


A
B


A
B

Immutable: state doesn’t change
Mutable: state could change

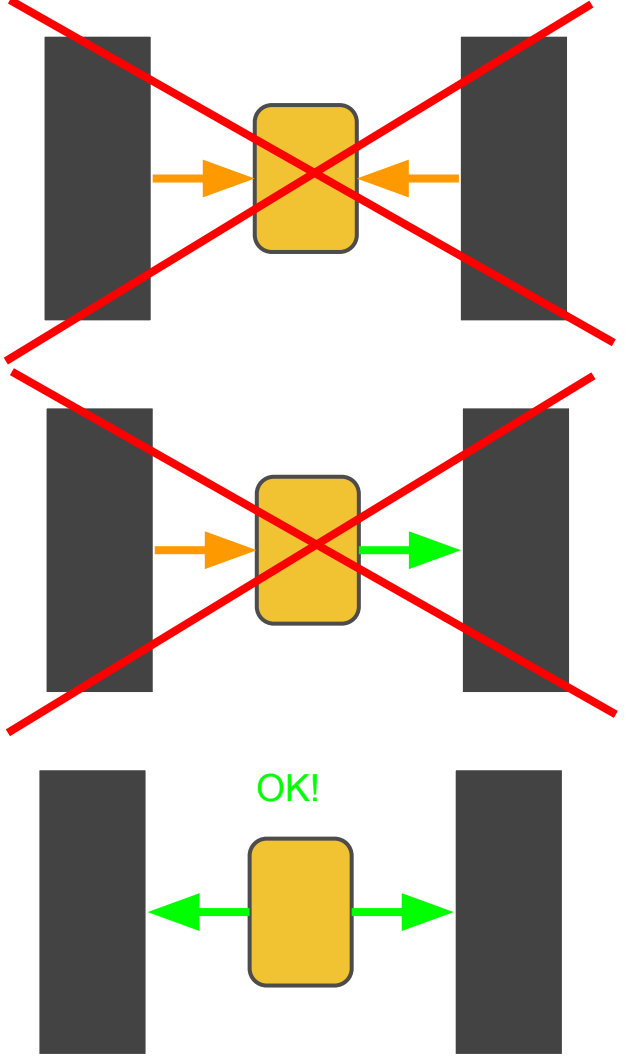
A

B


Immutability bedevils
garbage collection.
entures in the vBuffer | @superSGP

Race condition

- More than one access to shared state
- At least one access is a write
- Access isn’t synchronized/atomic



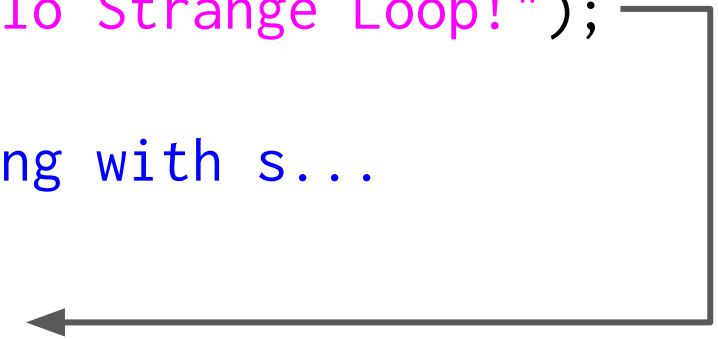
```rust
fn main() {
    println!("Hello Strange Loop!");
}
```

```rust
fn main() {
    println!("Hello Strange Loop!");
}
```

```rust
fn sum_of_squares(slice: &[i32]) -> i32 {
    slice.iter().map(|x| x * x).sum()
}
```

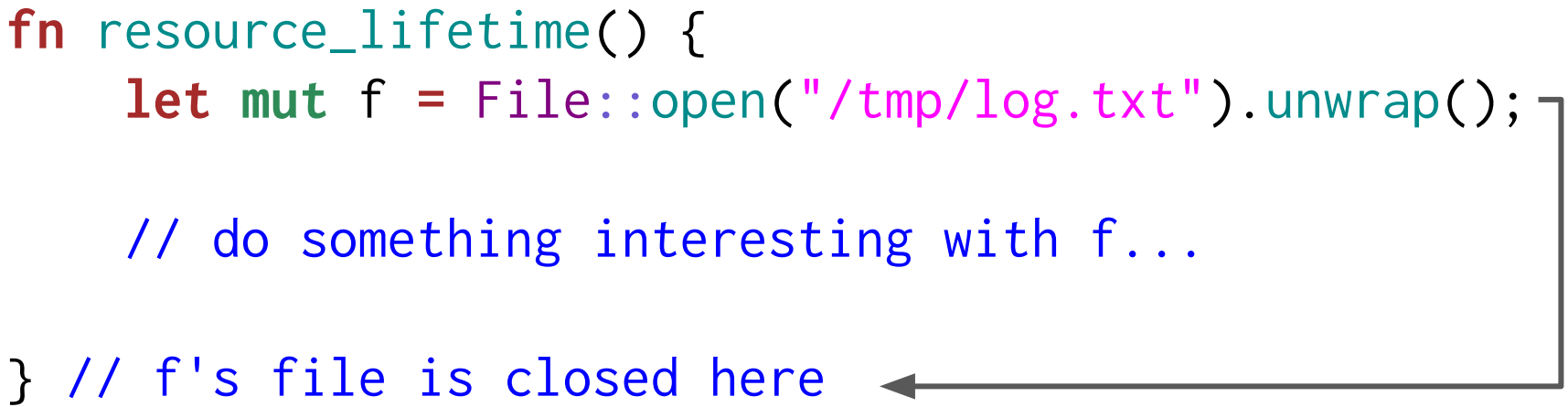
```rust
fn string_lifetime() {
    let s = String::from("Hello Strange Loop!");

    // do something interesting with s...

} // s's memory is freed here
```

```rust
fn string_lifetime() {
    let s = String::from("Hello Strange Loop!");

    // do something interesting with s...

} // s's memory is freed here
```



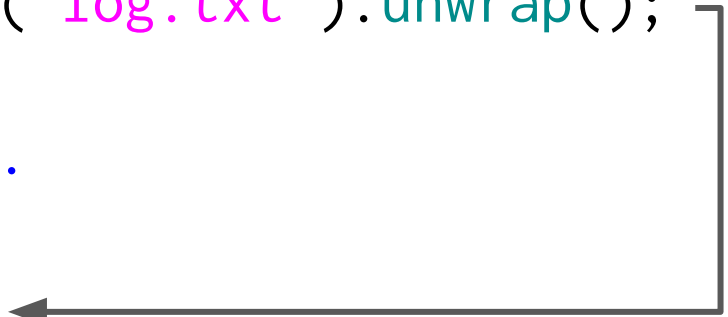
```
fn resource_lifetime() {
    let mut f = File::open("/tmp/log.txt").unwrap();

    // do something interesting with f...

} // f's file is closed here
```

```
fn resource_lifetime() {
    let mut f = File::open("/tmp/log.txt").unwrap();

    // do something interesting with f...

} // f's file is closed here
```


fn resource_lifetime() {
let mut f = File::open("/tmp/log.txt").unwrap();
// do something interesting with f...
} // f's file is closed here

```rust
fn string_lifetime() {
    let s = String::from("Hello Strange Loop!");

    // do something interesting with s...

} // s's memory is freed here

fn resource_lifetime() {
    let mut f = File::open("/tmp/log.txt").unwrap();

    // do something interesting with f...

} // f's file is closed here
```

```rust
fn resource_lifetime(enable_logging: bool)  {
    let s = String::from("Hello Strange Loop!");

    if enable_logging {
        let mut f = File::create("log.txt").unwrap();

        // log some stuff to f...

    } // f's file is closed here

    // do more interesting things with s...

} // s's memory is freed here
```



```rust
fn get_strange() -> String  {
    let s = String::from("Hello Strange Loop!");

    // do interesting things here...

    s
} // s's memory is NOT freed here...

fn example() {
    let strange = get_strange();
    // ...s's memory is now owned by strange...
} // ...and finally gets freed here
```

```rust
fn hello() {
    let s = String::from("Hello Strange Loop");

    hype(s);

    println!("{}", s);
}

fn hype(s: String) {
    println!("{}!!!", s);
}
```

```rust
fn hello() {
    let s = String::from("Hello Strange Loop");

    hype(s);

    println!("{}", s);
}
```

Compiler says “NO”

```
error[E0382]: use of moved value: `s`
  --> src/main.rs:72:20
   |
70 |     hype(s);
   |          - value moved here
71 |
72 |     println!("{}", s);
   |                     ^ value used here after move
```

```rust
fn hello() {
    let s = String::from("Hello Strange Loop");

    hype(s.clone());

    println!("{}", s);
}

fn hype(s: String) {
    println!("{}!!!", s);
}
```

This compiles!

Output:
Hello Strange Loop!!!
Hello Strange Loop

```rust
fn hello() {
    let s = String::from("Hello Strange Loop");

    hype(&s);

    println!("{}", s);
}


fn hype(s: &String) {
    println!("{}!!!", s);
}
```

```
fn hello() {
    let s = String::from("Hello Strange Loop");

    hype(&s);

    println!("{}", s);
}


fn hype(s: &String) {
    println!("{}!!!", s);
}
```

hype “borrows” s

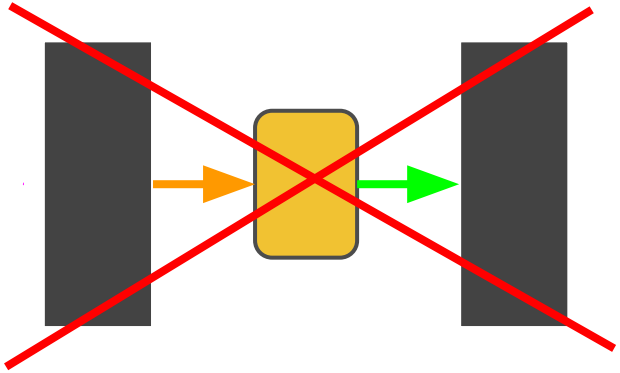
immutable String reference

```rust
fn immutable_borrow() {
    let mut s = String::from("Hello Strange Loop");

    {
        let borrowed = &s;
        println!("{}", s);  // read access, ok!
        s.truncate(5);      // write access, error
    }
}
```

```
fn immutable_borrow() {
    let mut s = String::from("Hello Strange Loop");

    {
```


```
}
```

```
fn immutable_borrow() {
    let mut s = String::from("Hello

    {
```



```
error[E0502]: cannot borrow `s` as mutable because it is also borrowed as immutable
   --> src/main.rs:117:9
    |
115 |         let borrowed = &s;
    |                        - immutable borrow occurs here
116 |         println!("{}", s);    // ok!
117 |         s.truncate(5);        // error
    |         ^ mutable borrow occurs here
118 |     }
    |     - immutable borrow ends here
}
```

```rust
fn immutable_borrow() {
    let mut s = String::from("Hello Strange Loop");

    {
        let borrowed = &s;
        println!("{}", s);  // read access, ok!
        //s.truncate(5);
    }
    s.truncate(5);      // write access, ok!
    println!("{}", s); // prints Hello
}
```

```rust
fn hello() {
    let mut s = String::from("Hello Strange Loop");

    hype(&mut s);

    println!("{}", s);
}


fn hype(s: &mut String) {
    s.push_str("!!!");
}
```

```
fn hello() {
    let mut s = String::from("Hello Strange Loop");

    hype(&mut s);

    println!("{}", s);
}


fn hype(s: &mut String) {
    s.push_str("!!!");
}
```

hype “mutably borrows” s

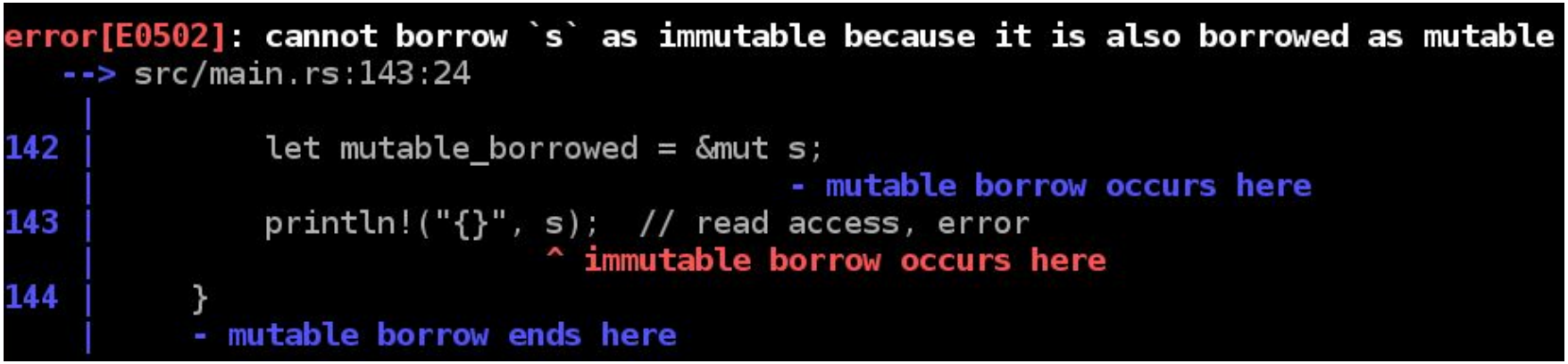
mutable String reference

```rust
fn hello() {
    let mut s = String::from("Hello Strange Loop");

    hype(&mut s);

    println!("{}", s);
}


fn hype(s: &mut String) {
    s.push_str("!!!");
}
```

Output:
Hello Strange Loop!!!

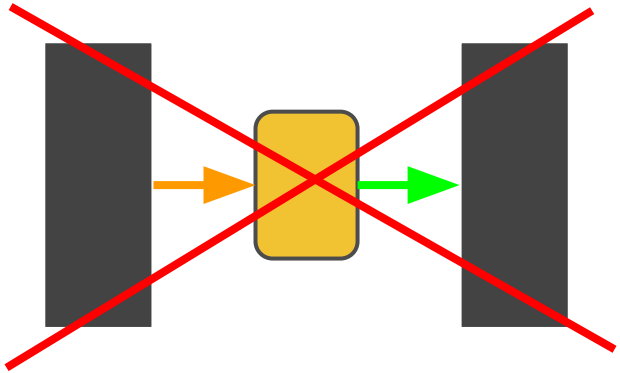
```rust
fn hello() {
    let mut s = String::from("Hello Strange Loop");

    {
        let mutable_borrowed = &mut s;
        println!("{}", s);  // read access, error
    }

    println!("{}", s);  // read access, ok!
}
```

```
fn hello() {
    let mut s = String::from("Hello Strange Loop");

    {
```




```
    println!("{}", s);  // read access, ok!
}
```

```
fn hello() {
    let mut s = String::from("Hello

    {
```

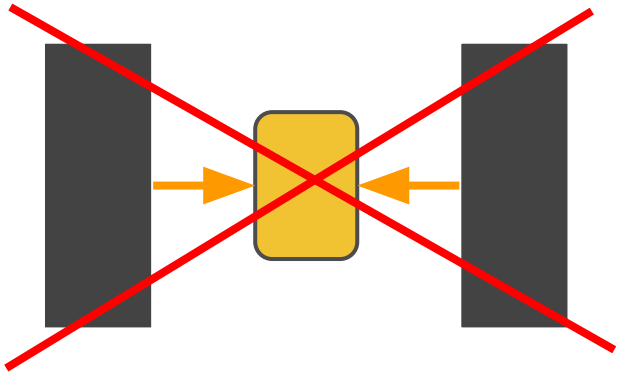


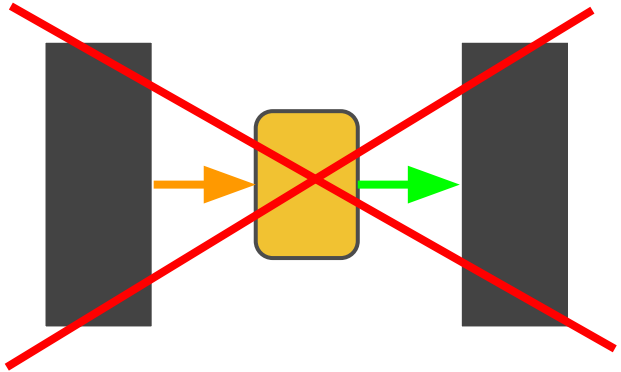
```
error[E0502]: cannot borrow `s` as immutable because it is also borrowed as mutable
   --> src/main.rs:143:24
    |
142 |         let mutable_borrowed = &mut s;
    |                                - mutable borrow occurs here
143 |         println!("{}", s);  // read access, error
    |                        ^ immutable borrow occurs here
144 |     }
    |     - mutable borrow ends here
```

```
    println!("{}", s);  // read access, ok!
}
```

Rust borrowing/ownership summary

- If you can **read** something, no one else can be **writing** it
- If you can **write** something, no one else can be **reading** or **writing** it
- For borrows, this is guaranteed at compile time - zero runtime cost!





Rust borrowing/ownership summary

- If you can **read** something, no one else can be **writing** it
- If you can **write** something, no one else can be **reading** or **writing** it
- For borrows, this is guaranteed at compile time - zero runtime cost!

- Efficient memory safety without garbage collection
- Prevents all data races!

Learning Rust

Compiler makes you care a lot about:

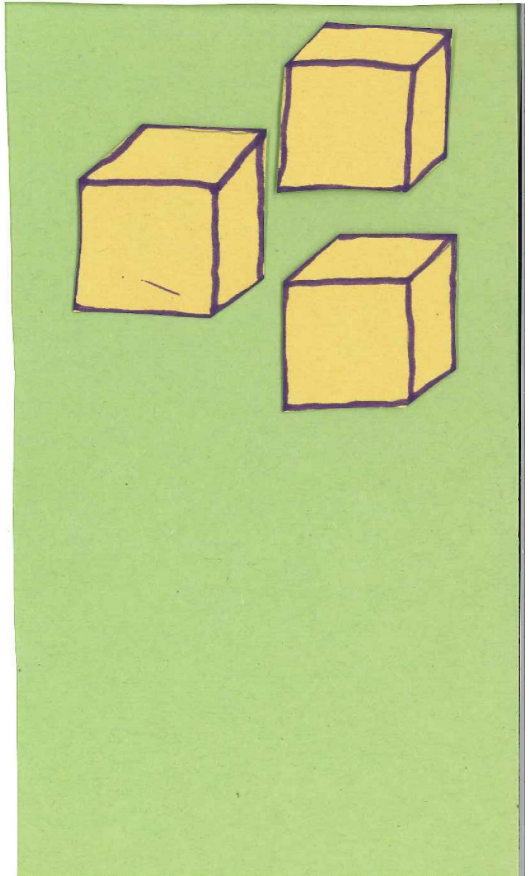
- Resources
- Ownership
- Lifetimes

Safe concurrent programming

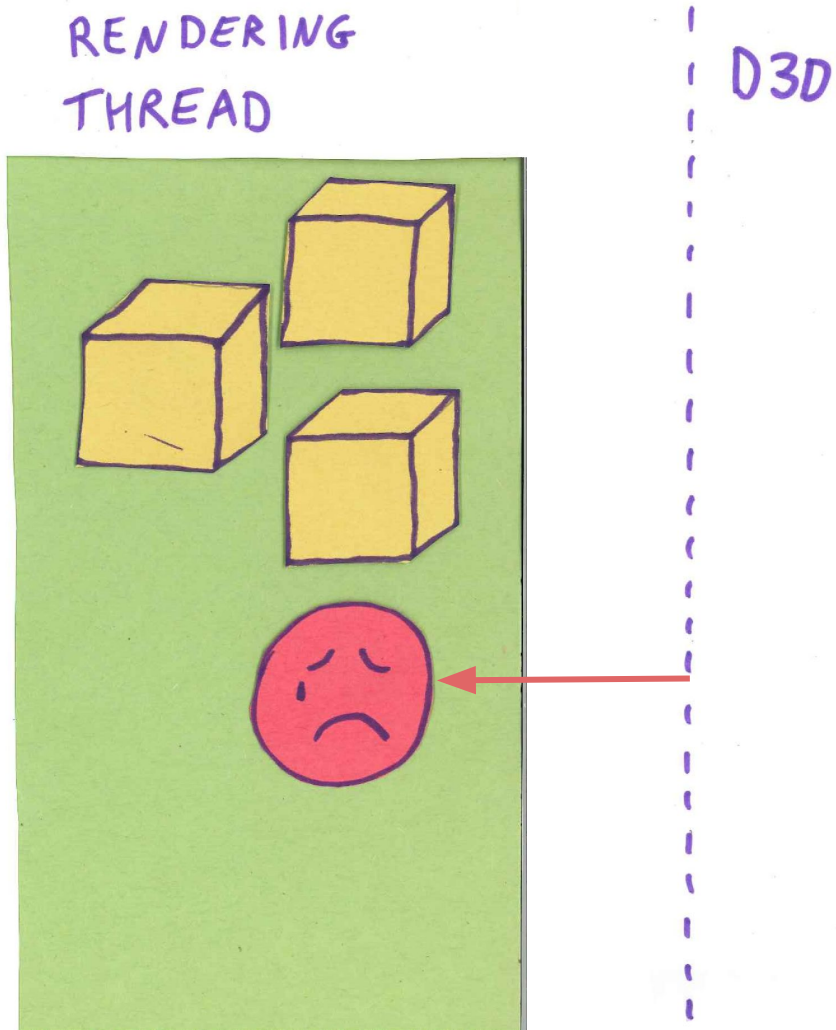
You should care a lot about:

- Resources
- Ownership
- Lifetimes

RENDERING THREAD

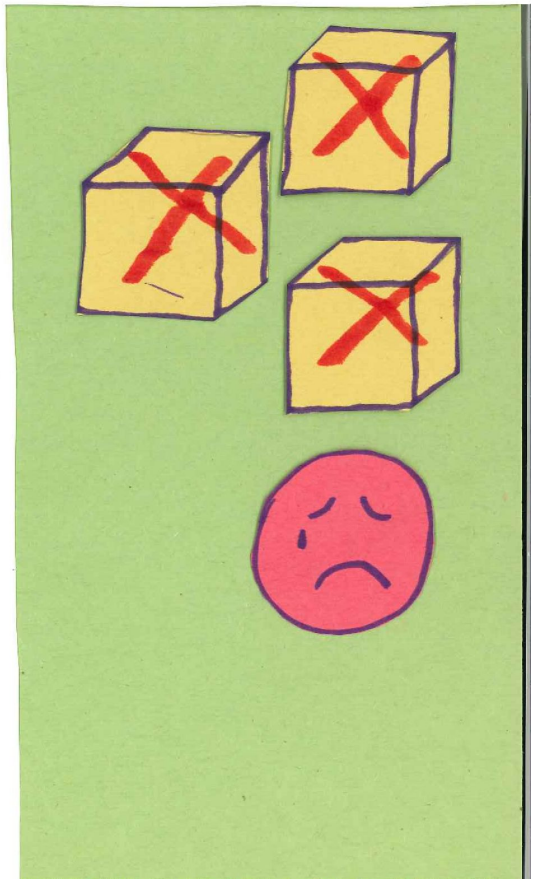


D3D


RENDERING
THREAD
D3D

RENDERING THREAD

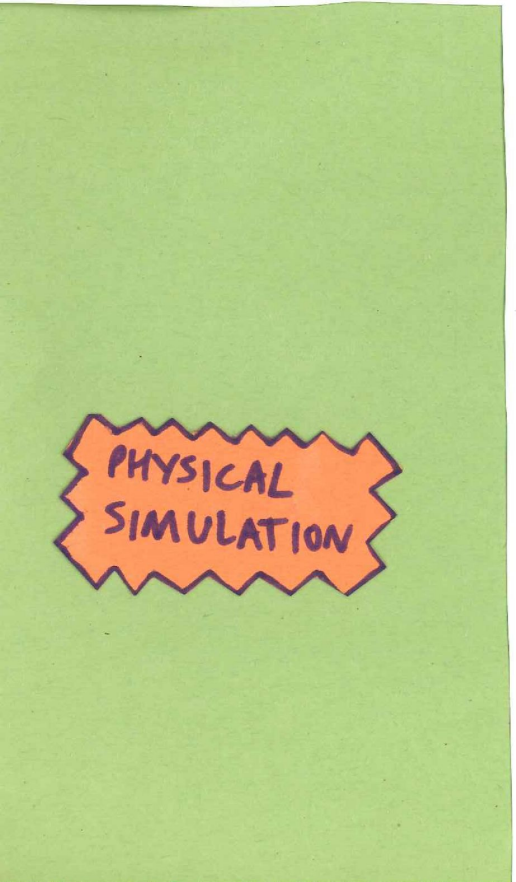
D3D



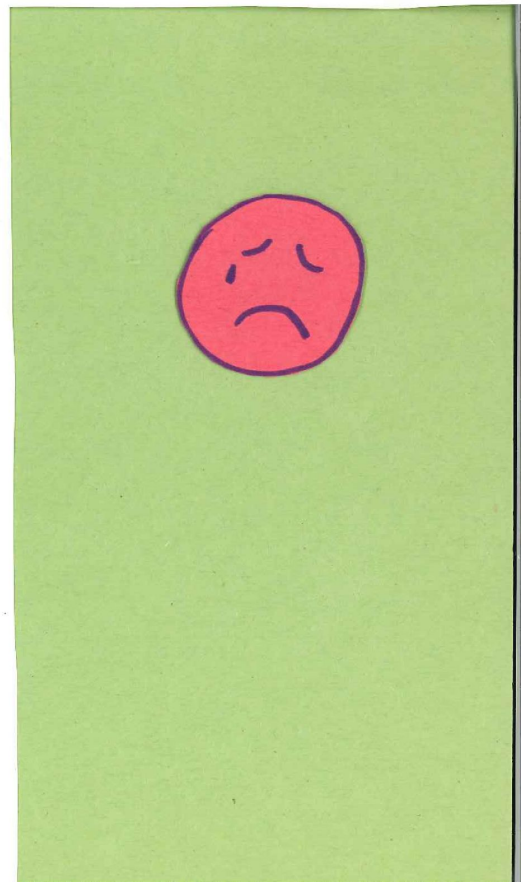
MAIN THREAD

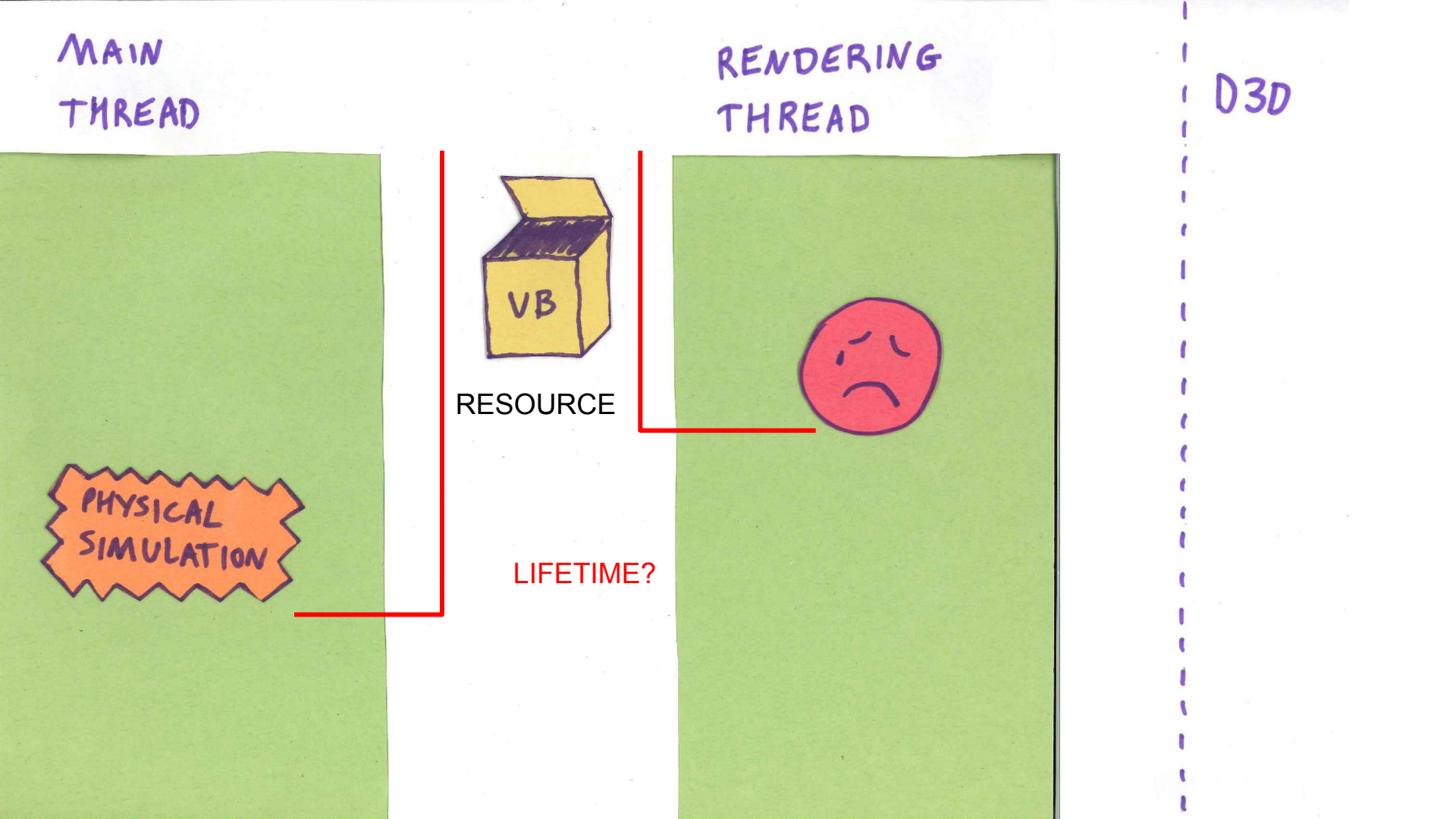
RENDERING THREAD

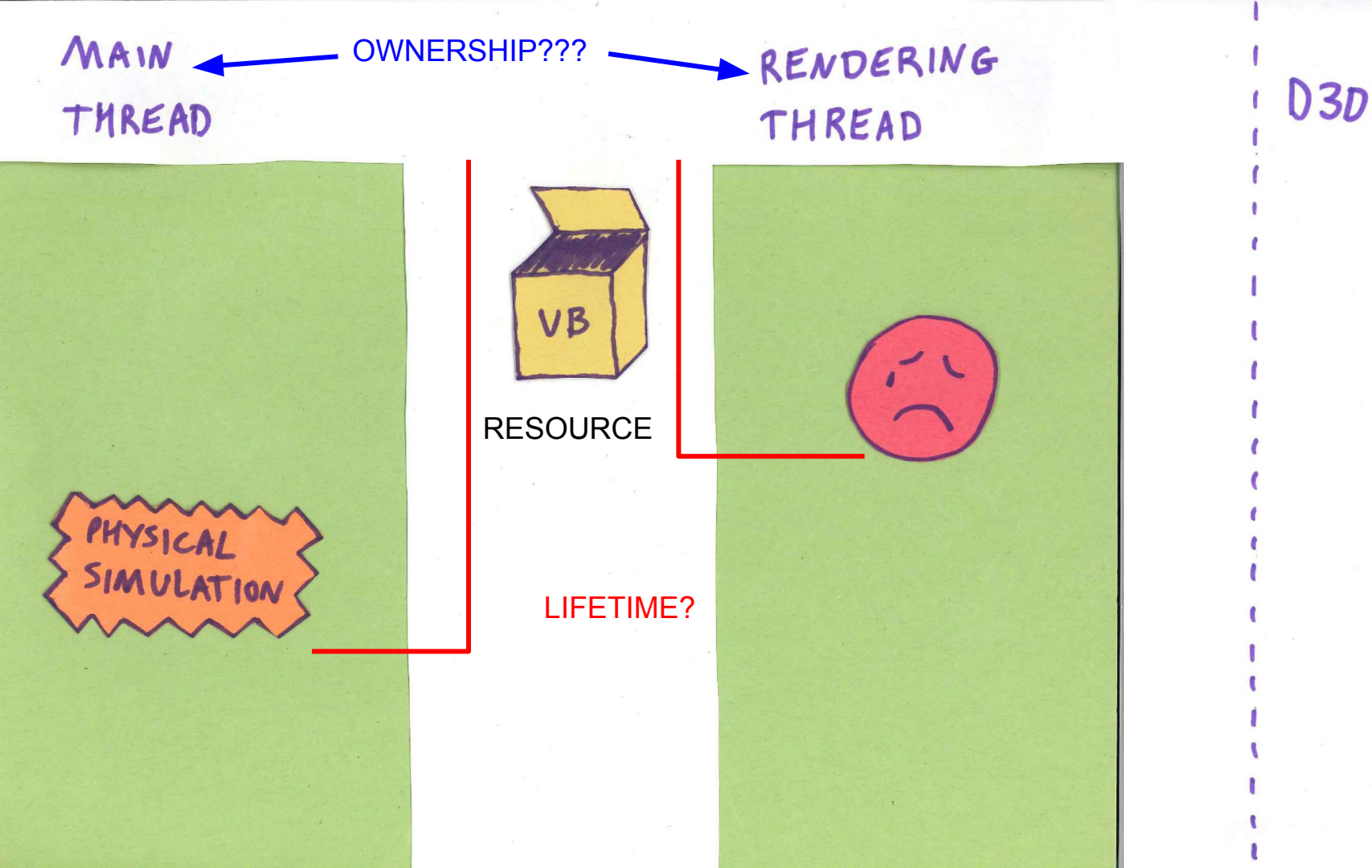
D3D

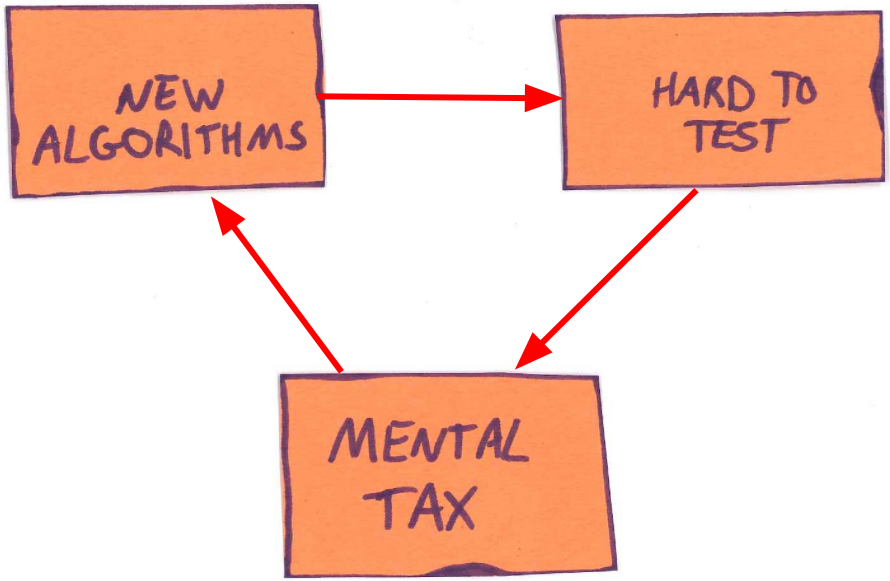


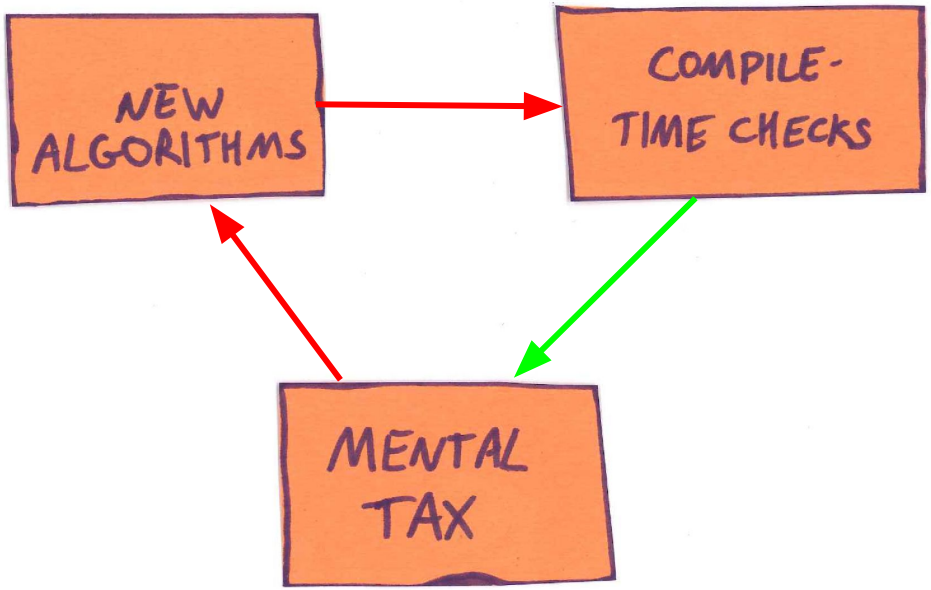
RESOURCE

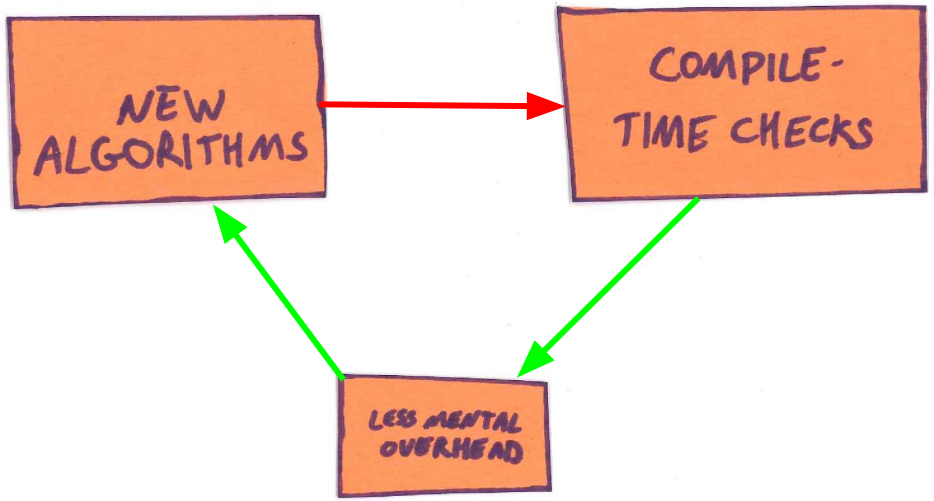


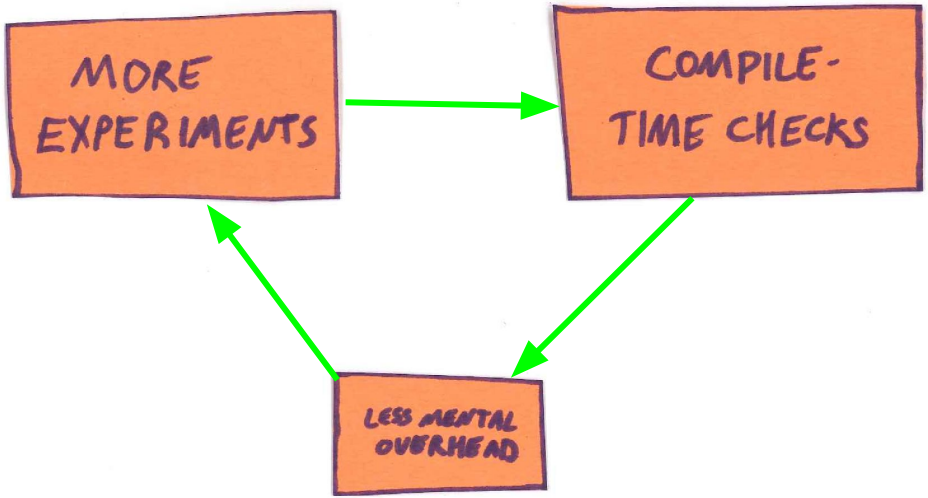

MAIN THREAD
RENDERING THREAD
D3D
VB
RESOURCE
PHYSICAL SIMULATION
LIFETIME?


MAIN THREAD
OWNERSHIP???
RENDERING THREAD
D3D
VB
RESOURCE
PHYSICAL SIMULATION
LIFETIME?


NEW ALGORITHMS
HARD TO TEST
MENTAL TAX


NEW ALGORITHMS
COMPILE-TIME CHECKS
MENTAL TAX


NEW ALGORITHMS
COMPILE-TIME CHECKS
LESS MENTAL OVERHEAD


MORE EXPERIMENTS
COMPILE-TIME CHECKS
LESS MENTAL OVERHEAD

5 Languages Every Programmer Should Learn

1.

2.

3.

4.

5.

5 Languages Every Programmer Should Learn

1. ??
2. ??
3. ??
4. ??
5. ??

6 Languages Every Programmer Should Learn

1. ??
2. ??
3. ??
4. ??
5. ??
6. Rust

Resources to learn Rust

- https://doc.rust-lang.org/stable/book - Learn by the book
- https://rustbyexample.com - Learn by example
- https://play.rust-lang.org - Try Rust in a browser
- http://intorust.com - Screencasts: ownership and borrows
- https://users.rust-lang.org - Forums
- #rust-beginners on irc.mozilla.org

Credits

These wonderful people helped me make this talk better:

- @jessitron
- Jingyu Sullins
- Gina Willard

I blatantly stole a slide from Sarah Groff Hennigh-Palermo’s Strange Loop talk “Adventures in the vBuffer” which you should watch

Title slide image:

- Cedar Apple Rust (CC BY 2.0) https://flic.kr/p/bzTpSu, Mike Lewinski