Rust's Recipes for Code and Community

@qedunham

talks.edunham.net/djangoconau2019

Rust's Recipes for Code and Community

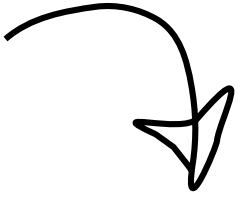
@qedunham

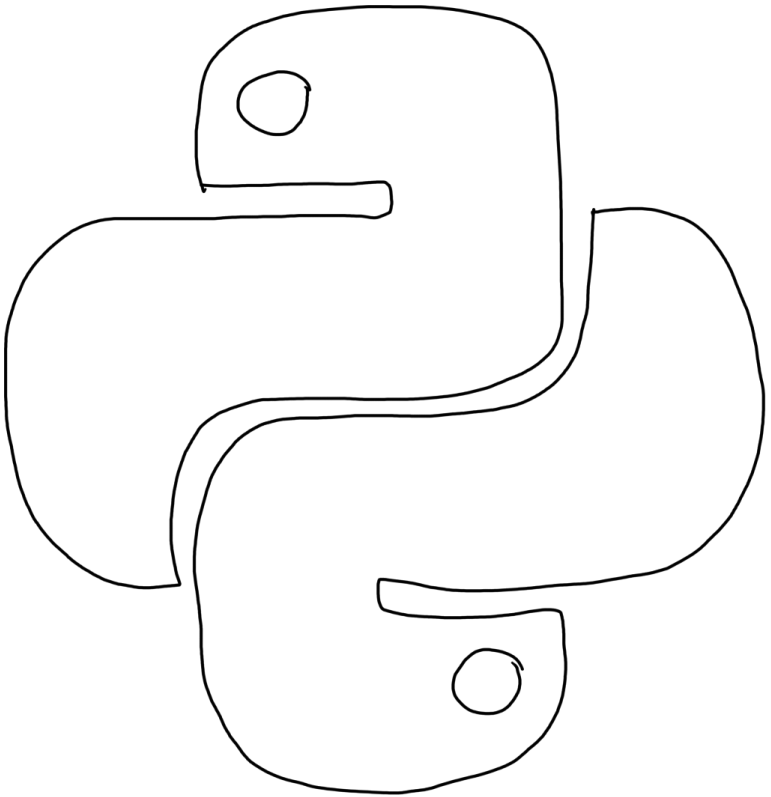
talks.edunham.net/djangoconau2019

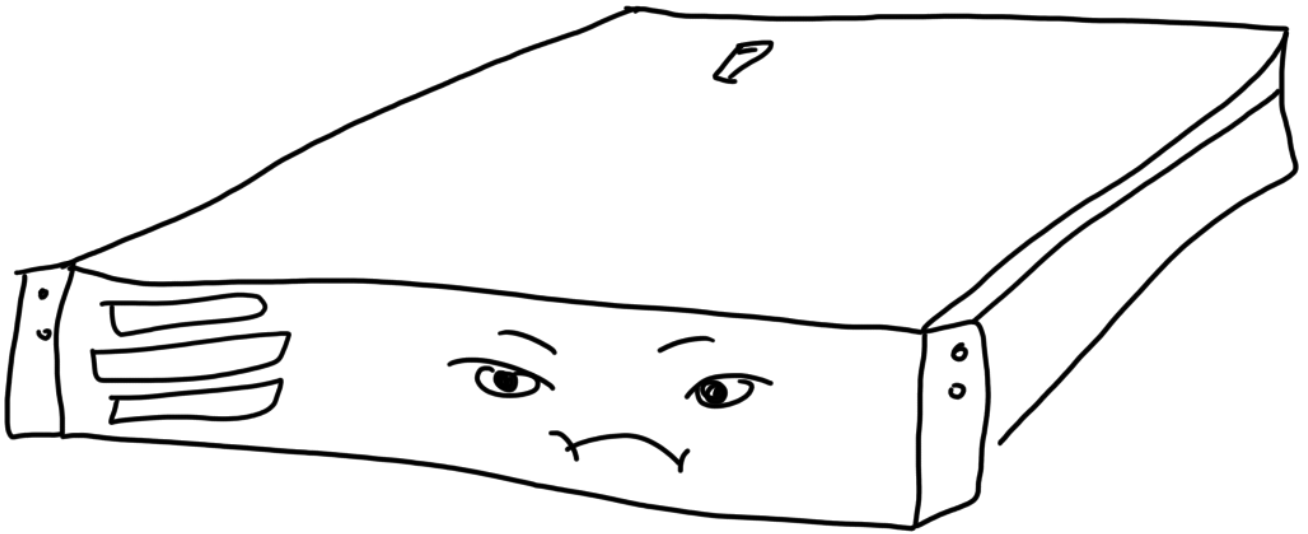
Rust's Recipes for Code and Community

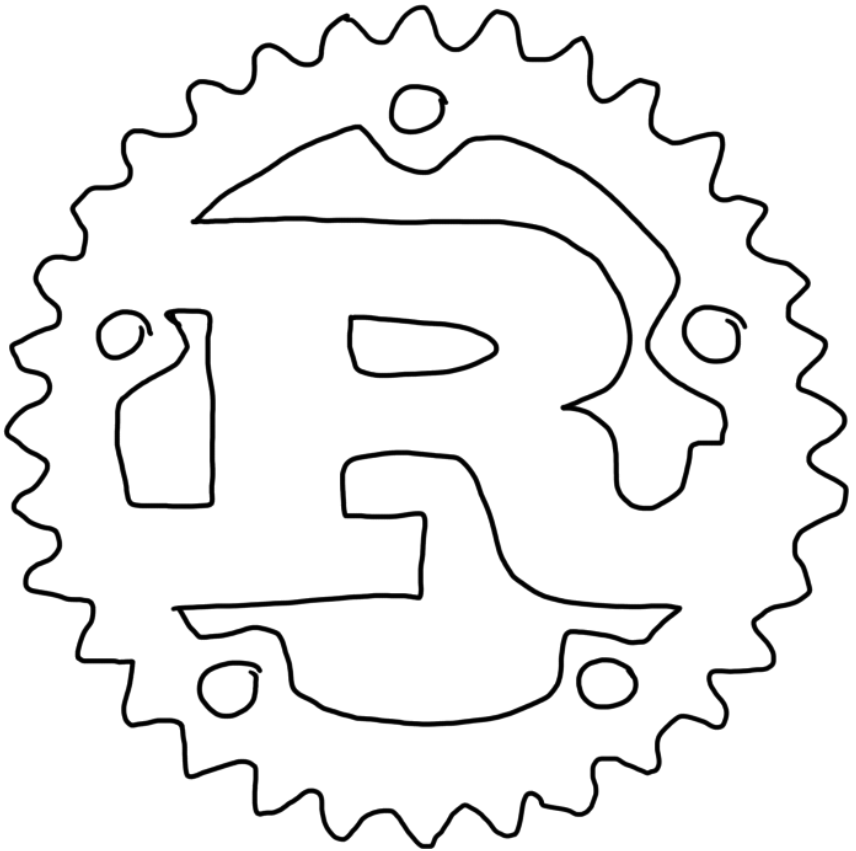
@qedunham

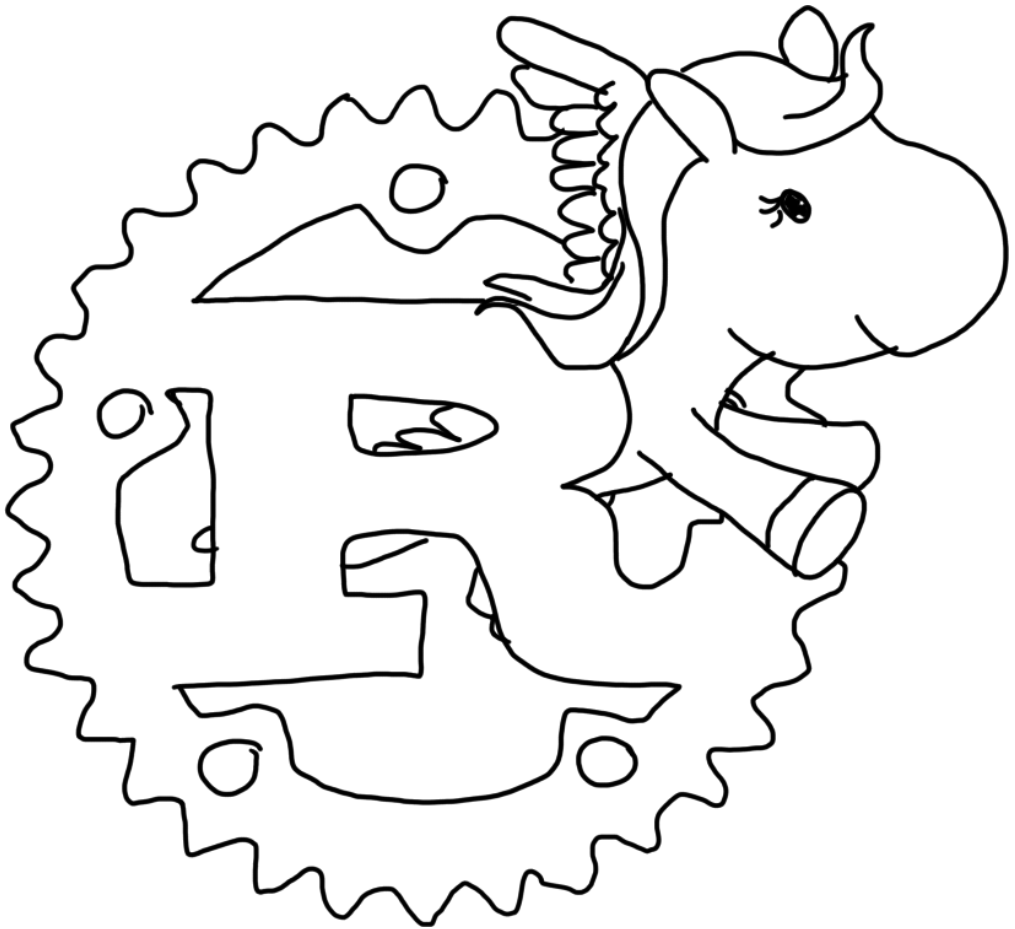
talks.edunham.net/djangoconau2019

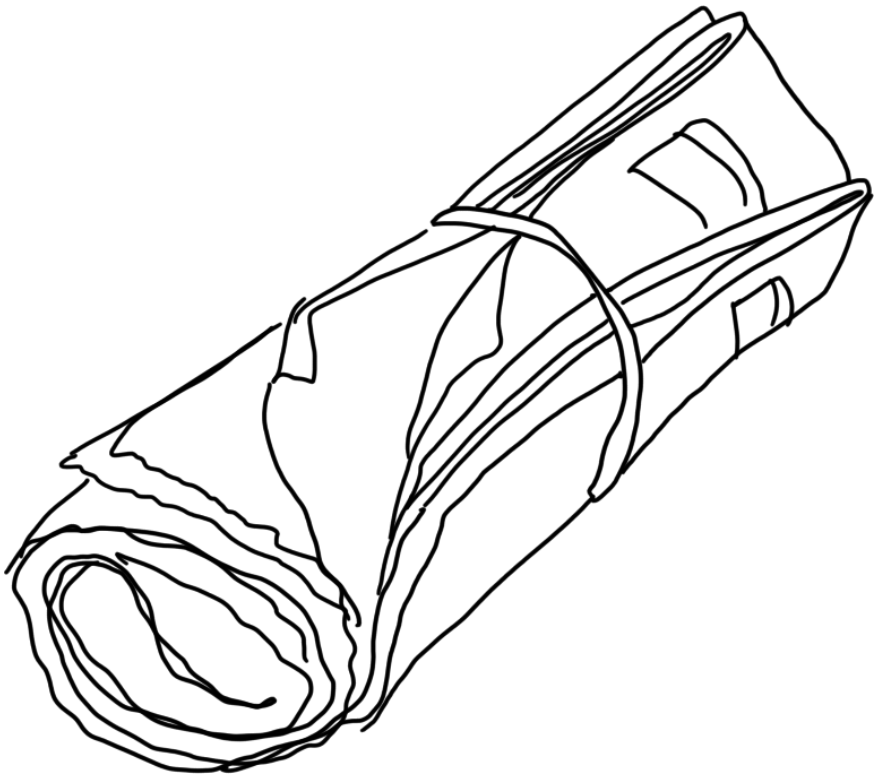


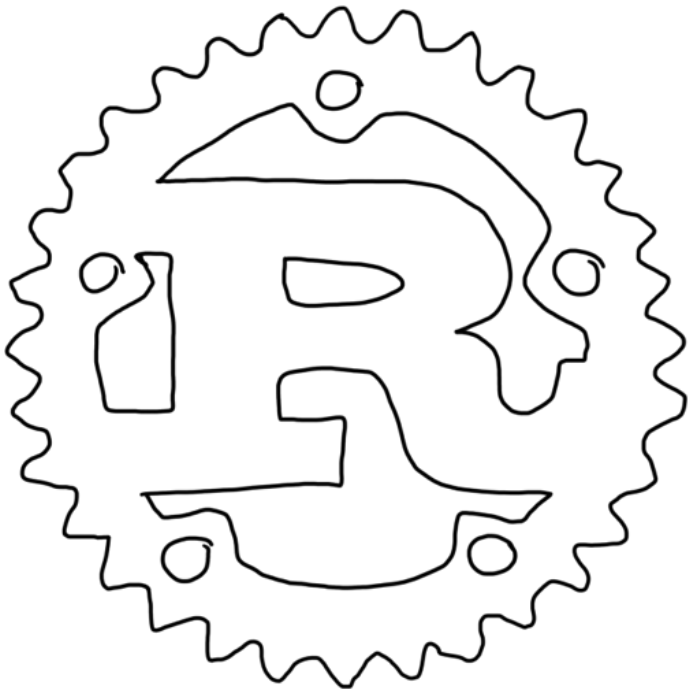




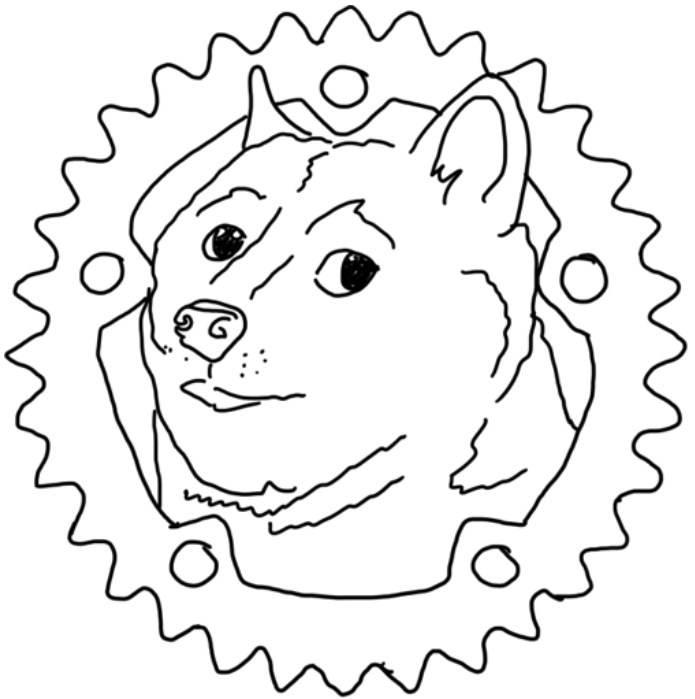


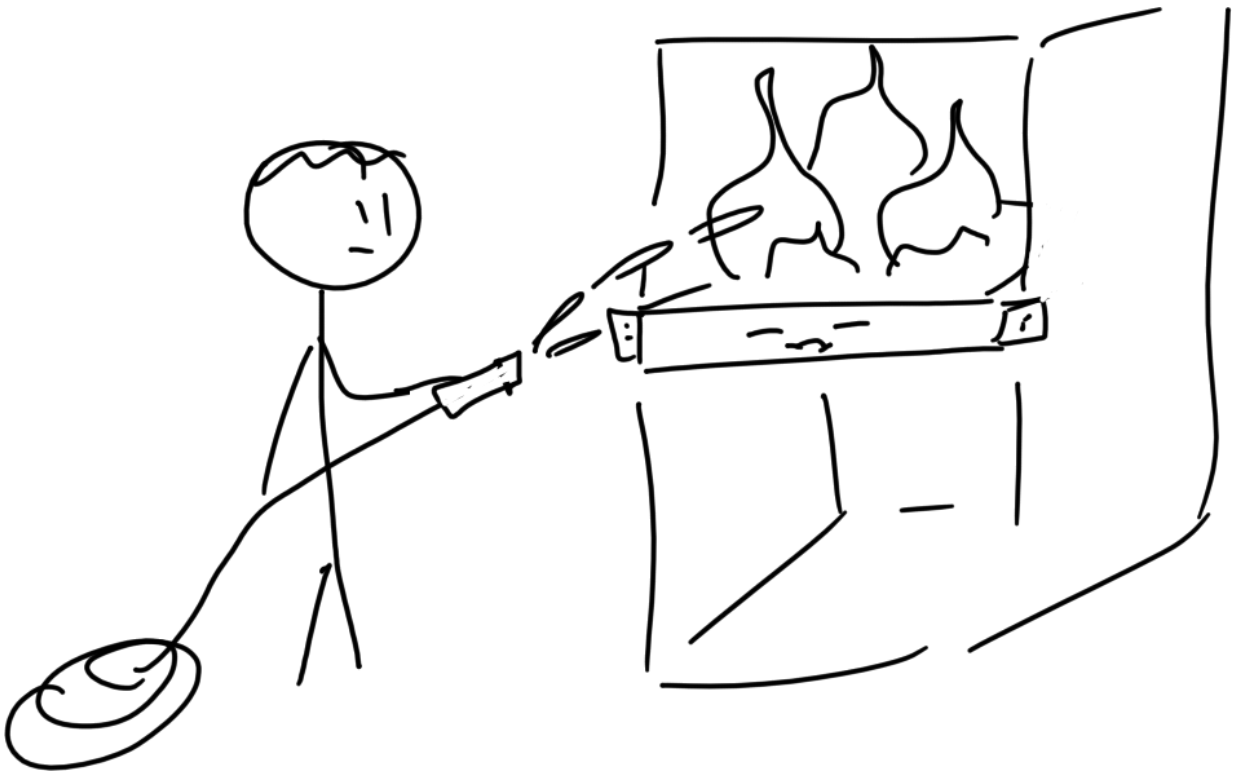


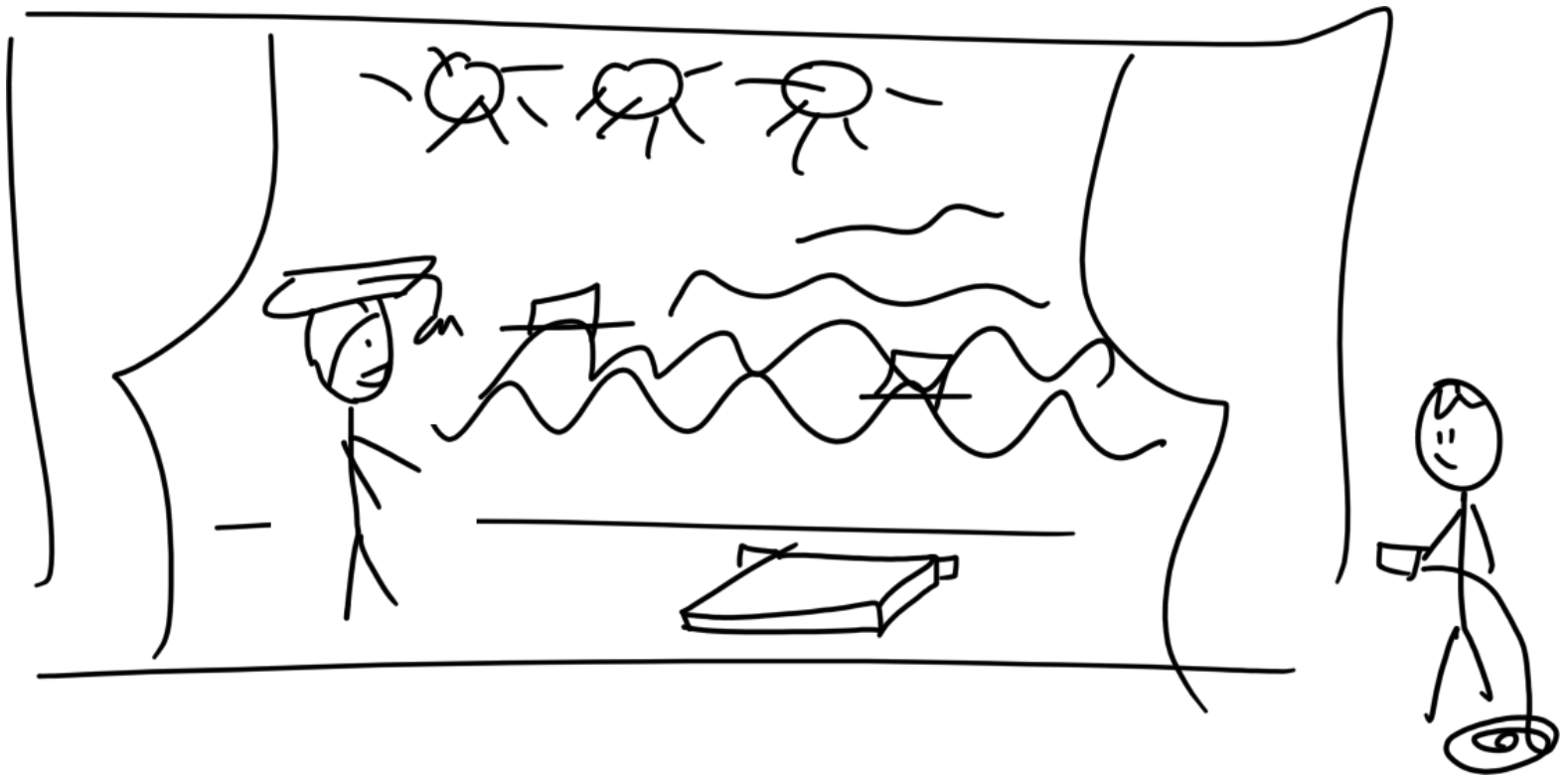


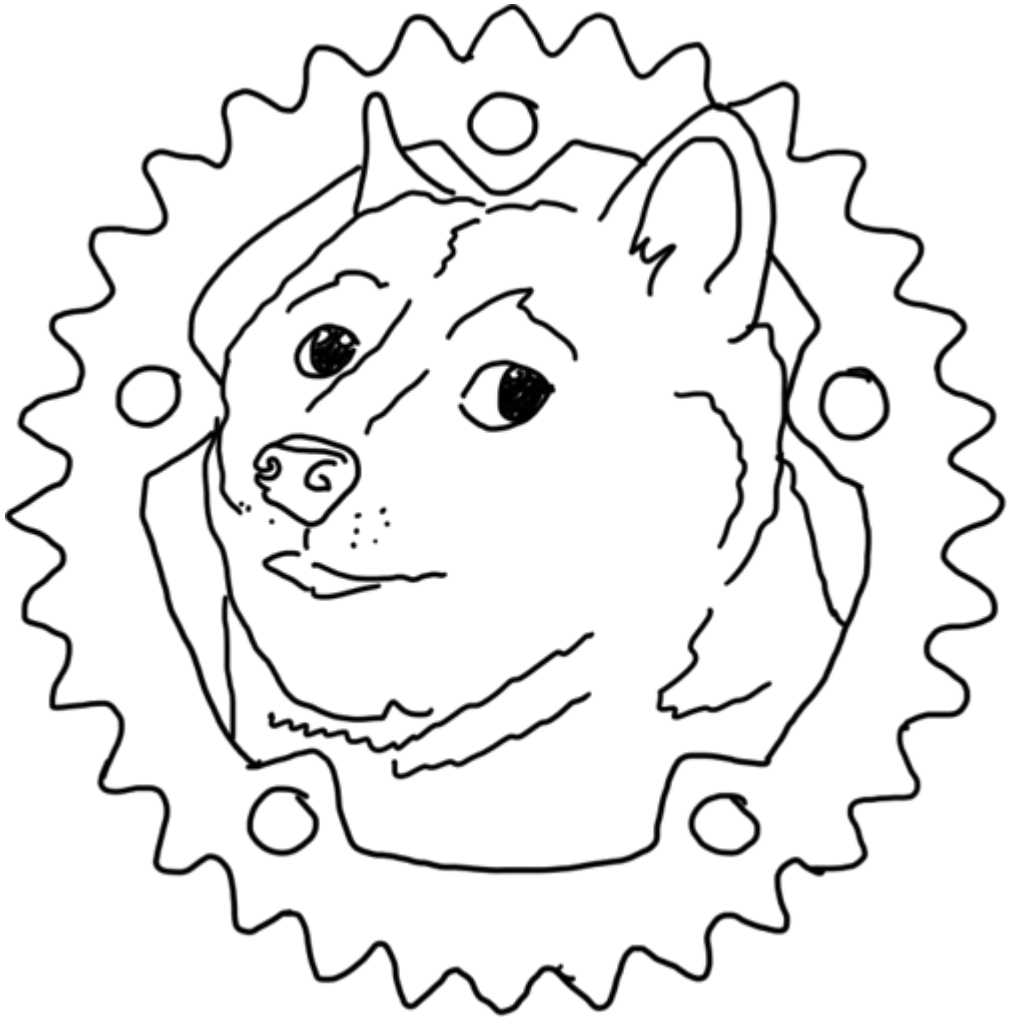


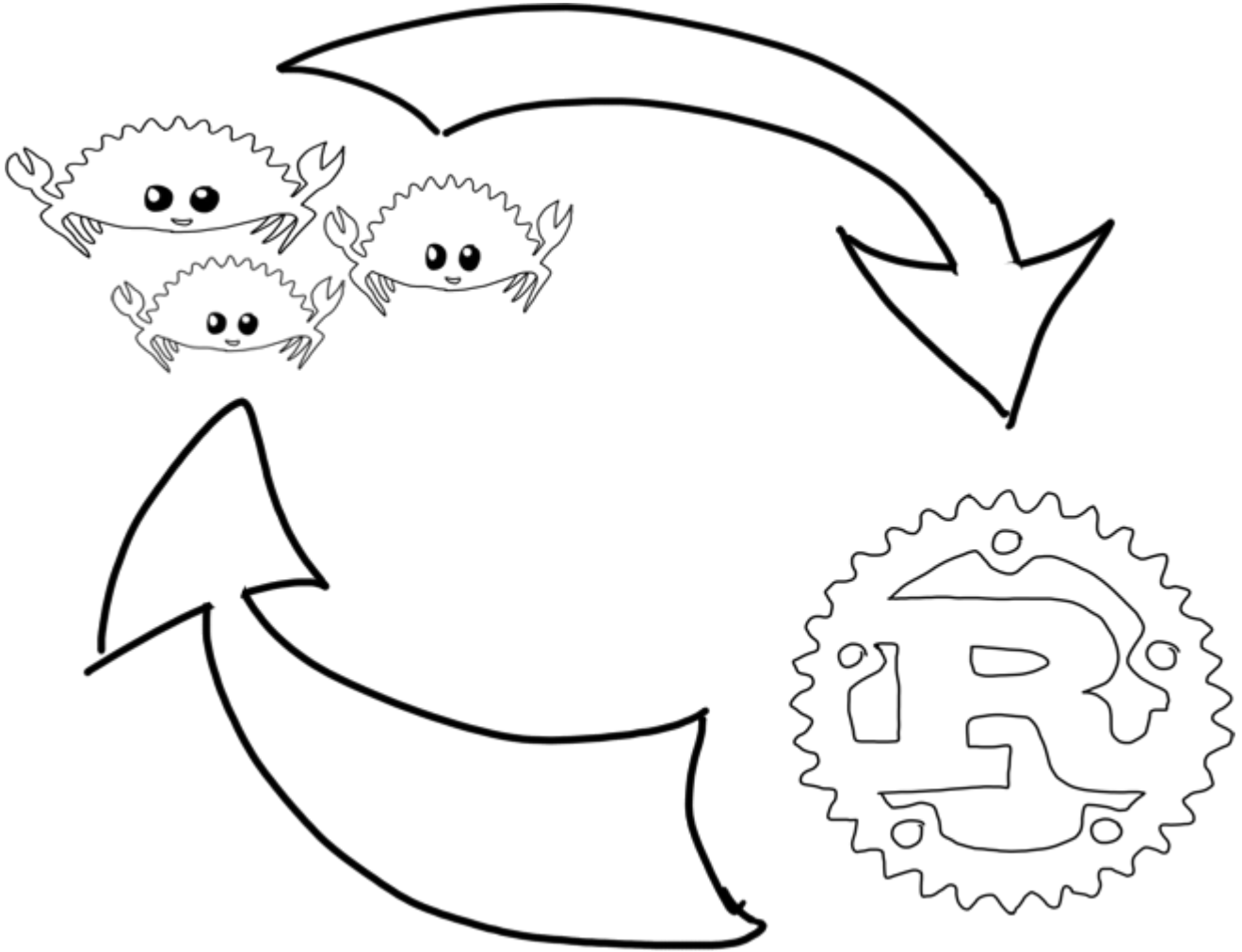
&

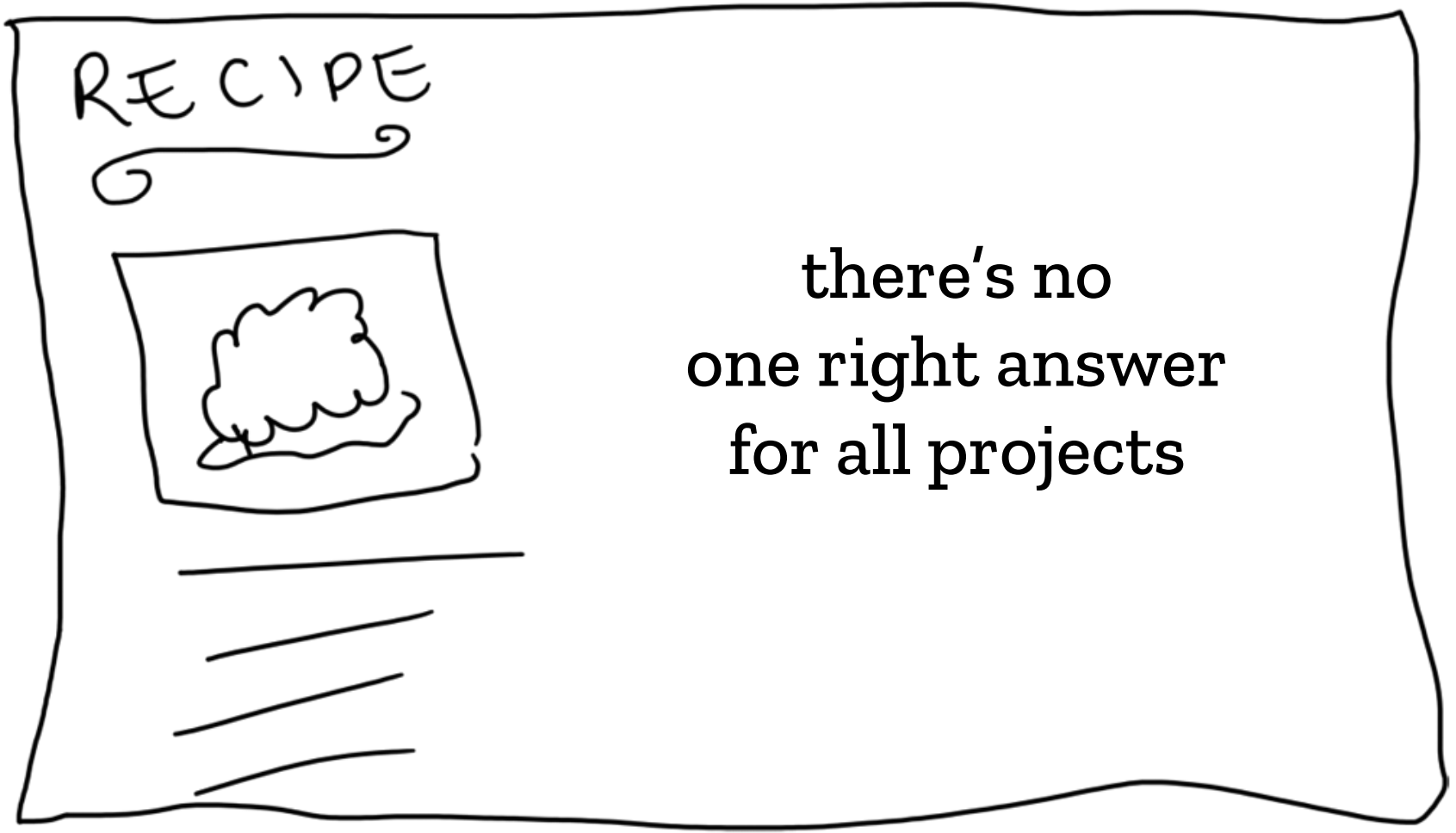






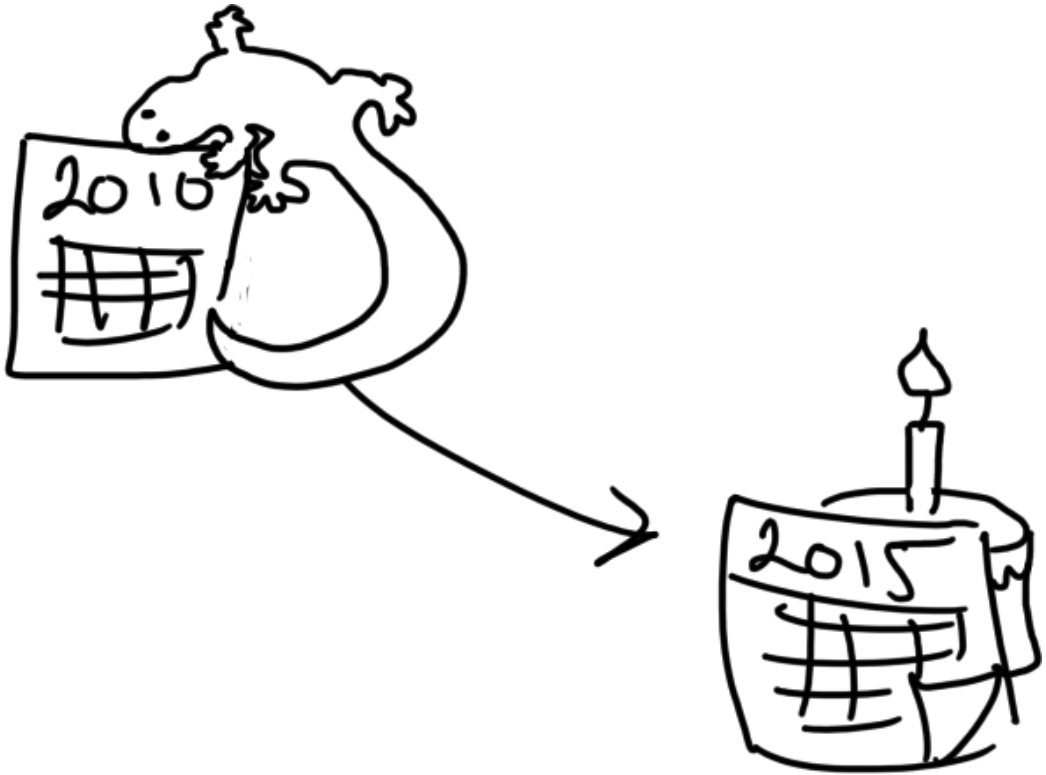


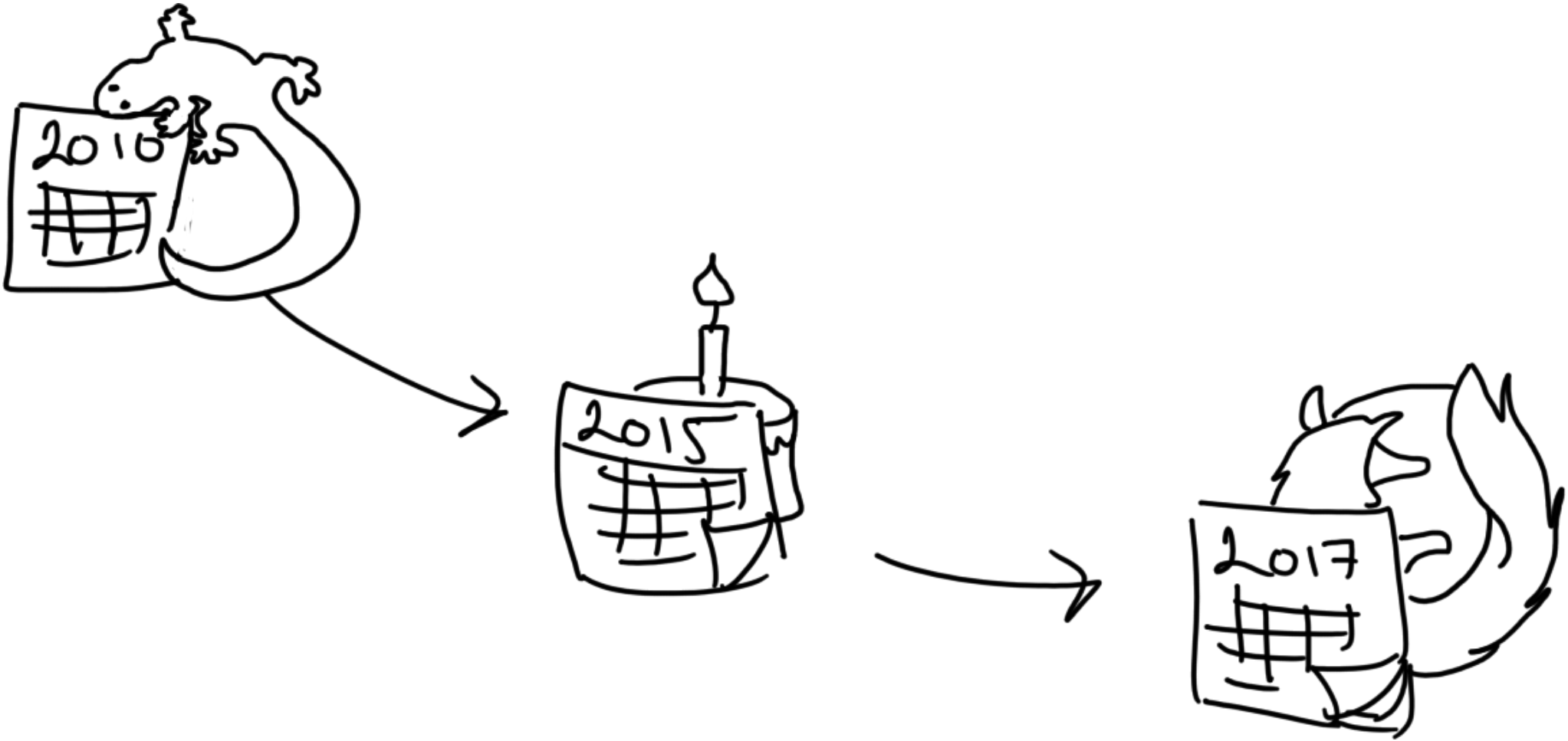


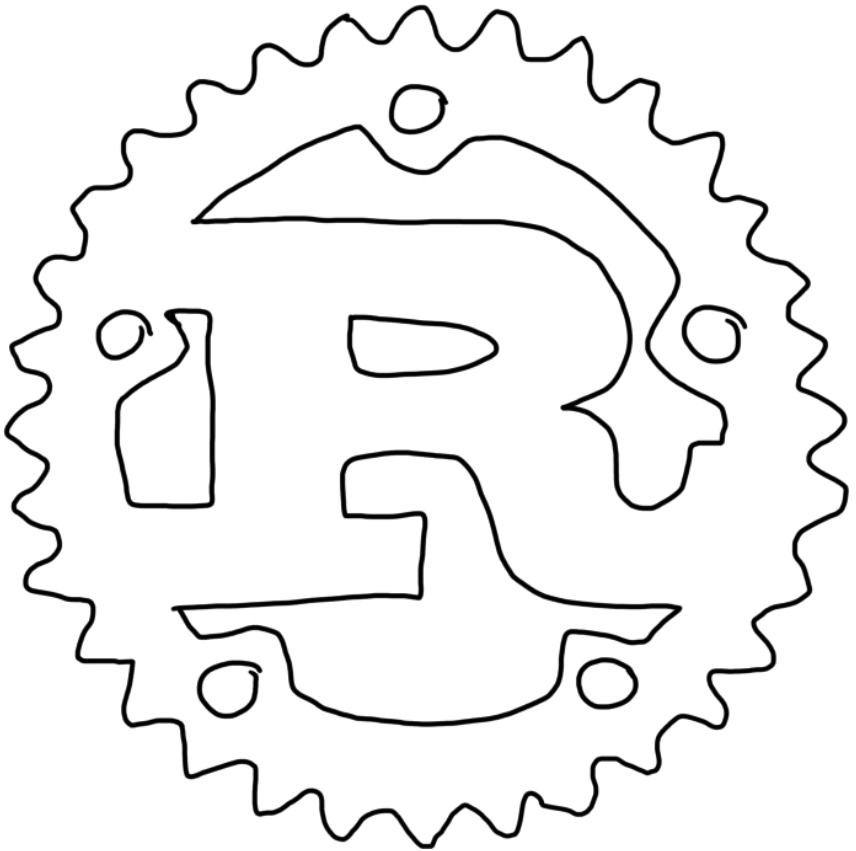


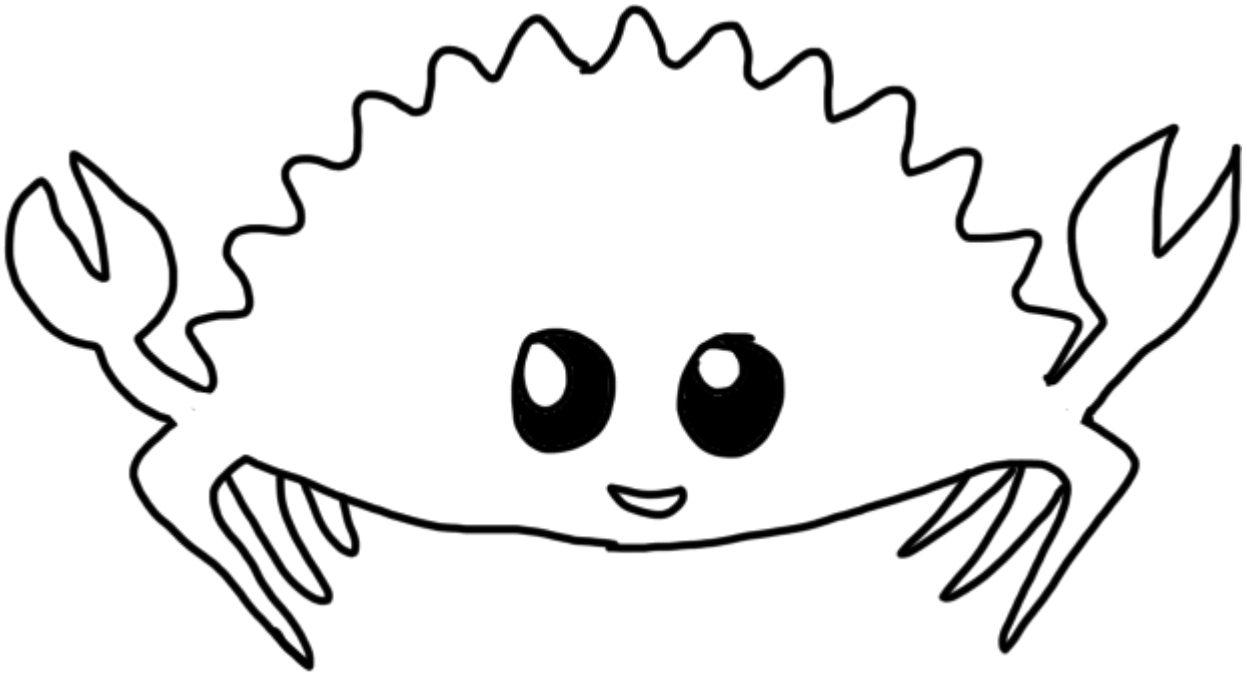
there's no
one right answer
for all projects

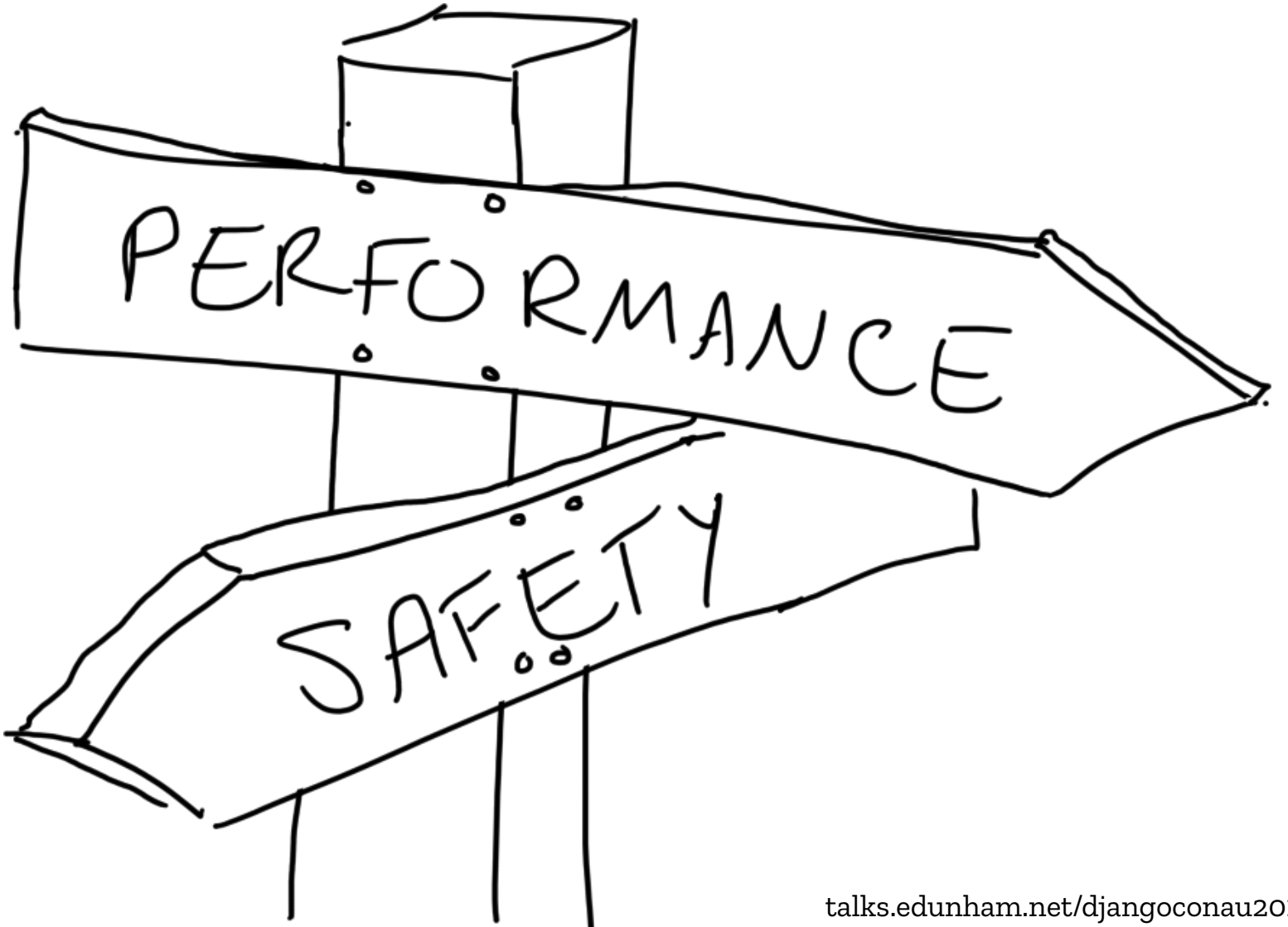

2010

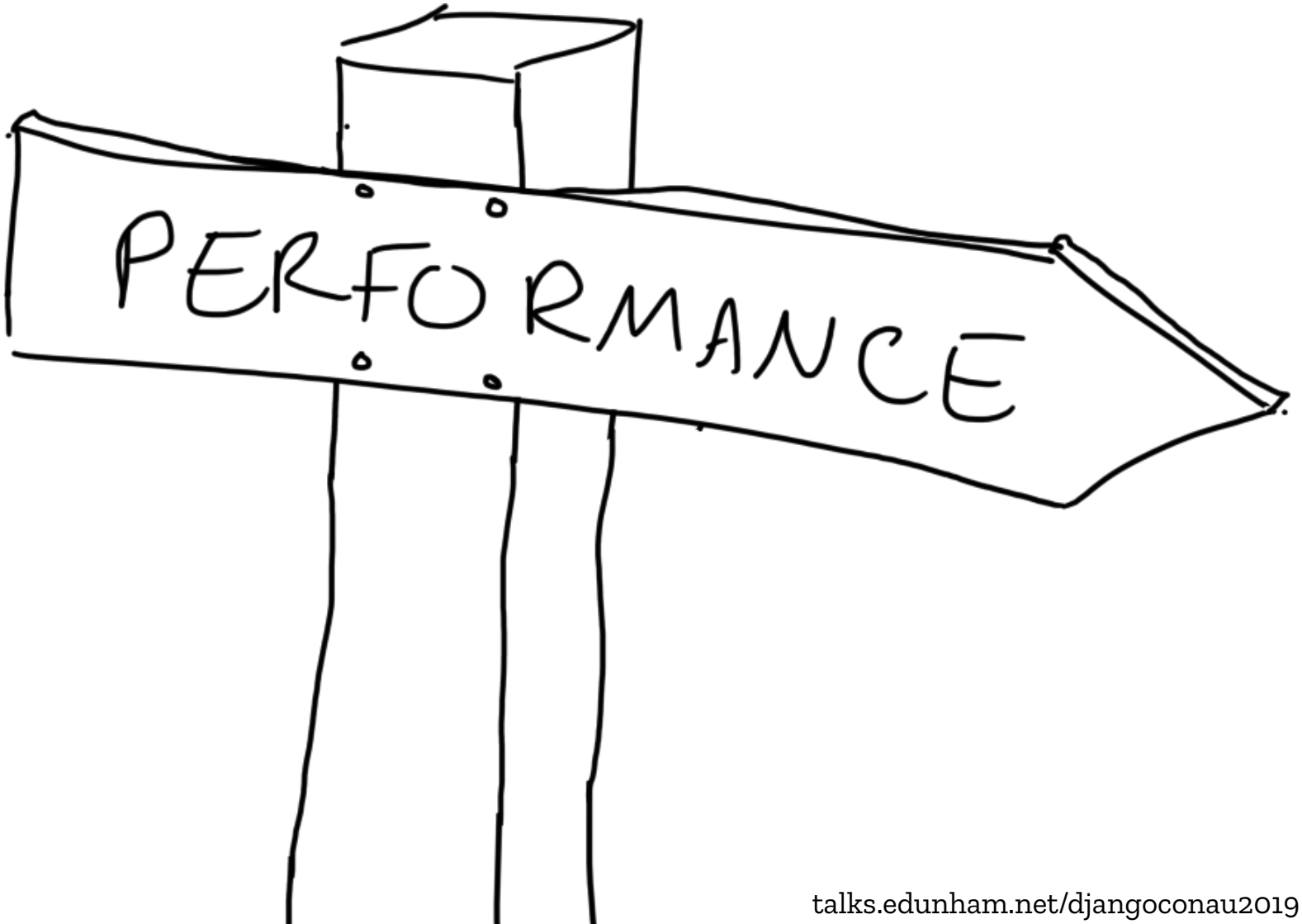

2016
2015


2010
2015
2017

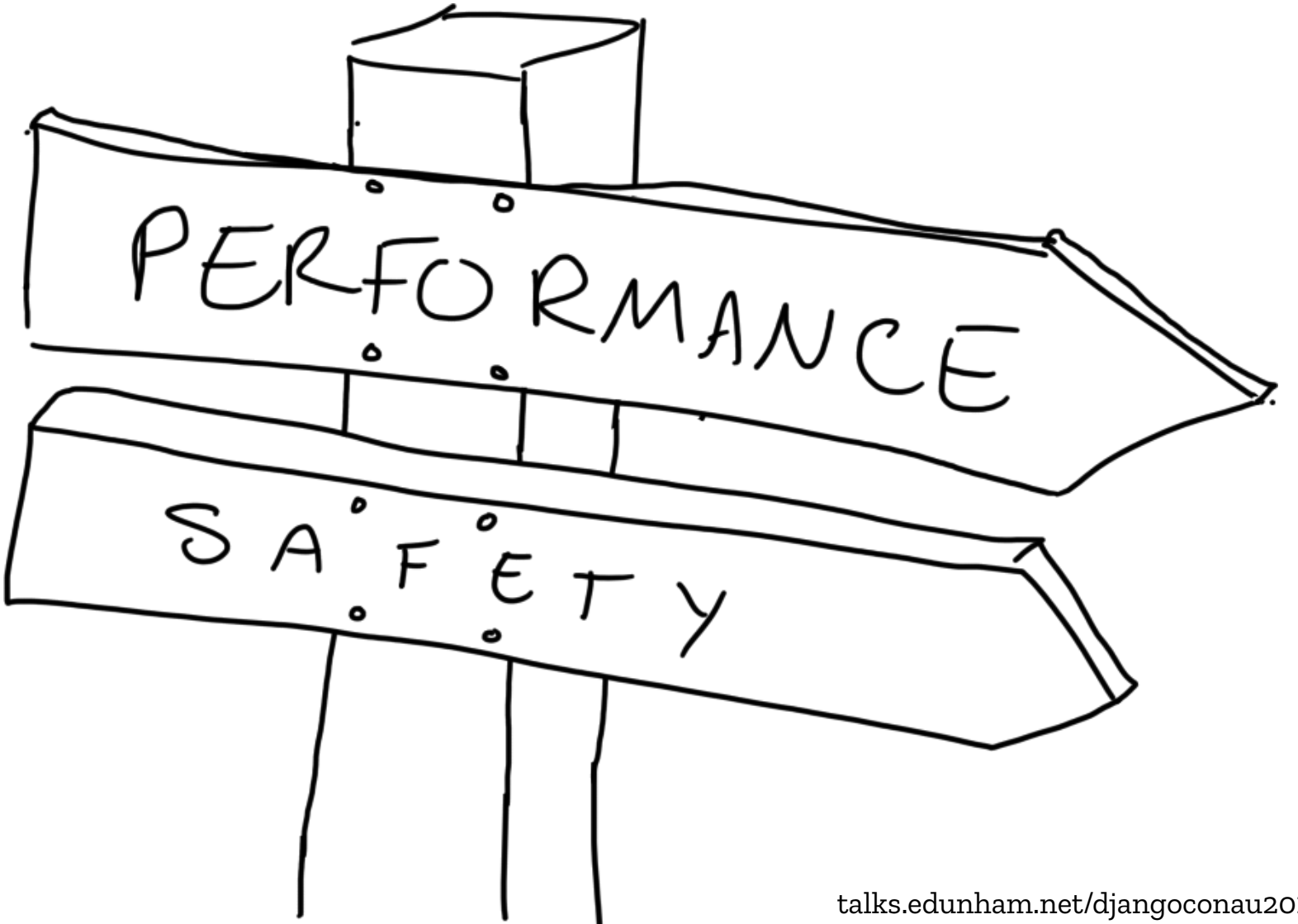


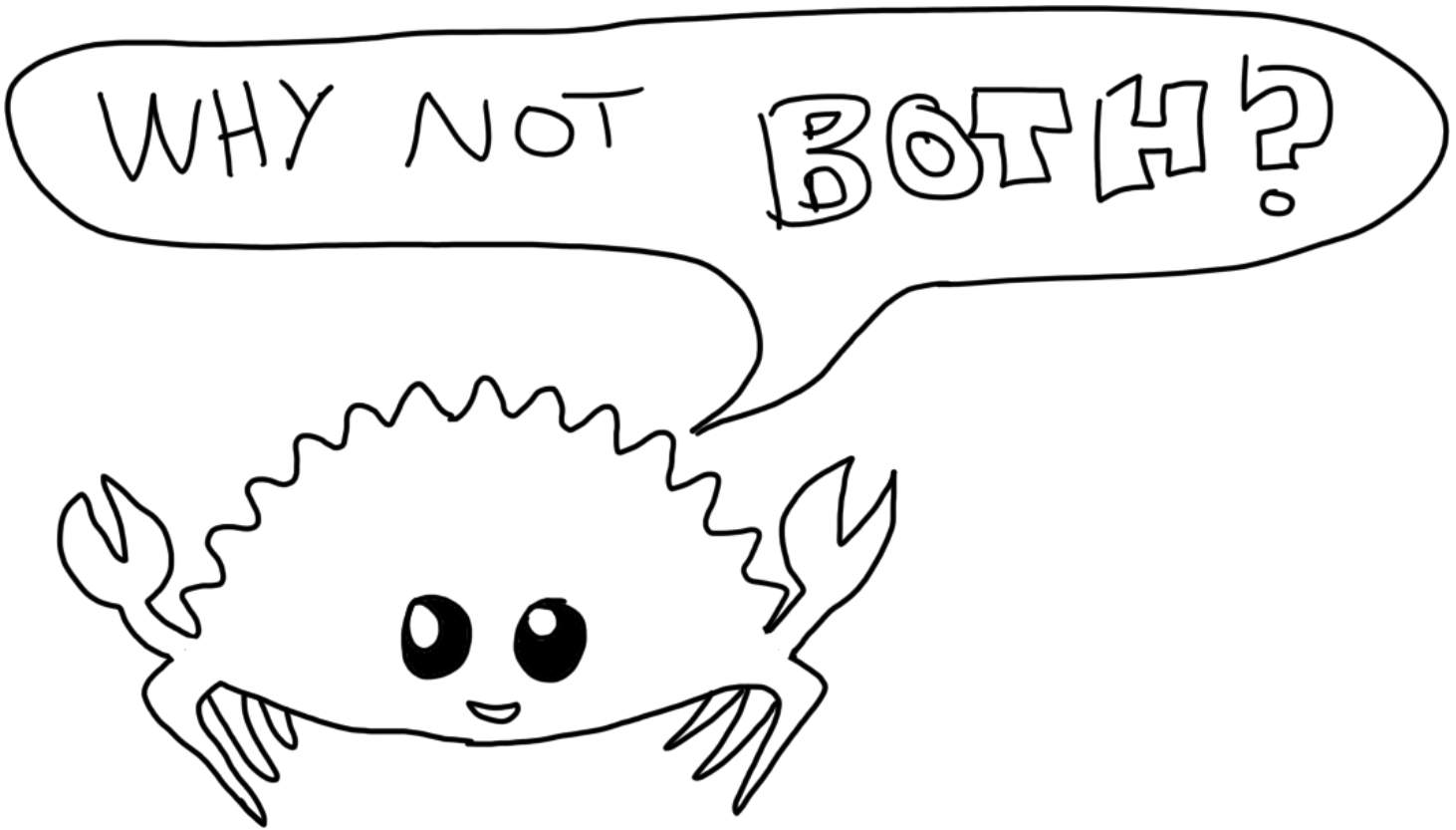


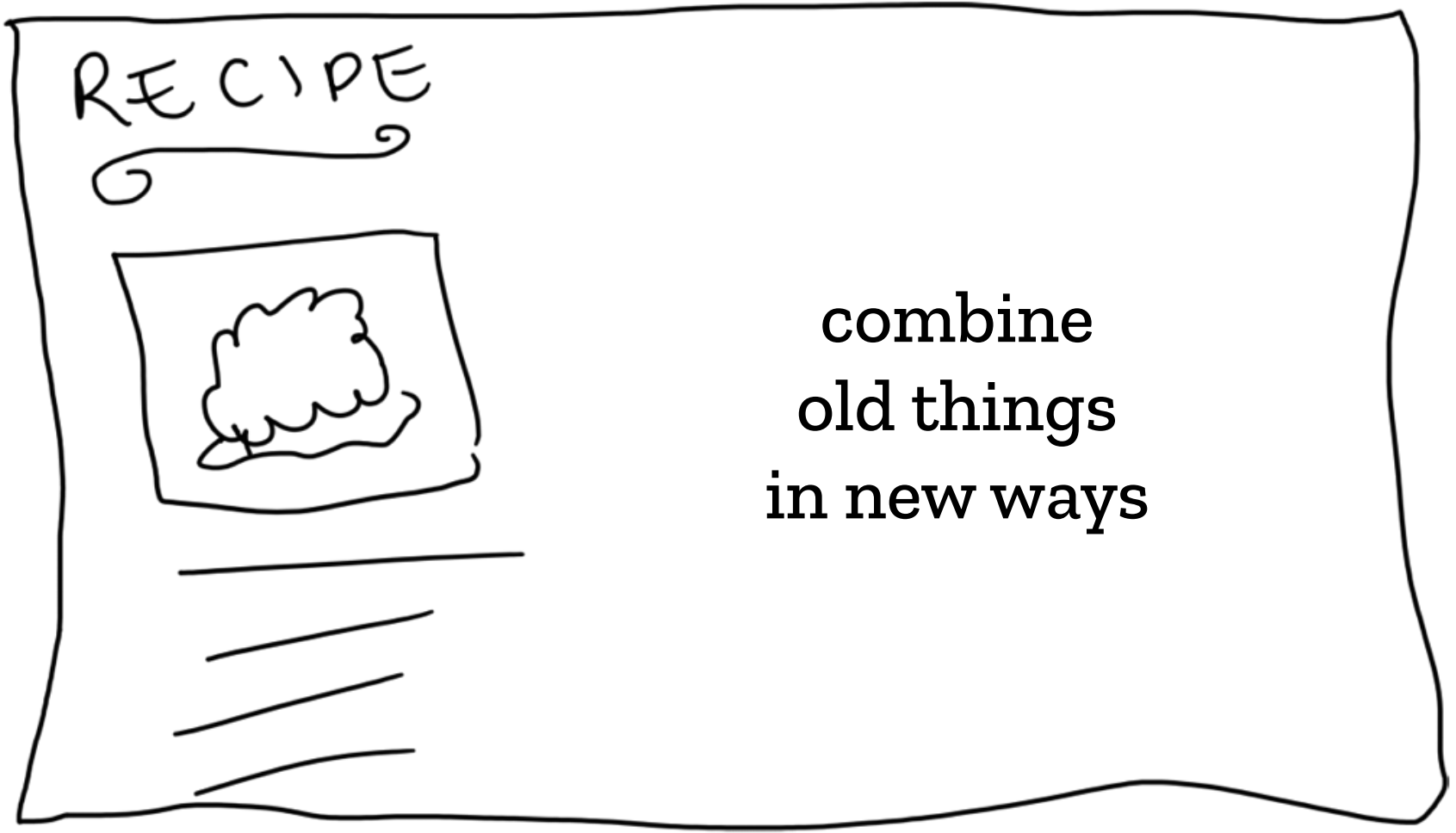

PERFORMANCE
SAFETY


PERFORMANCE

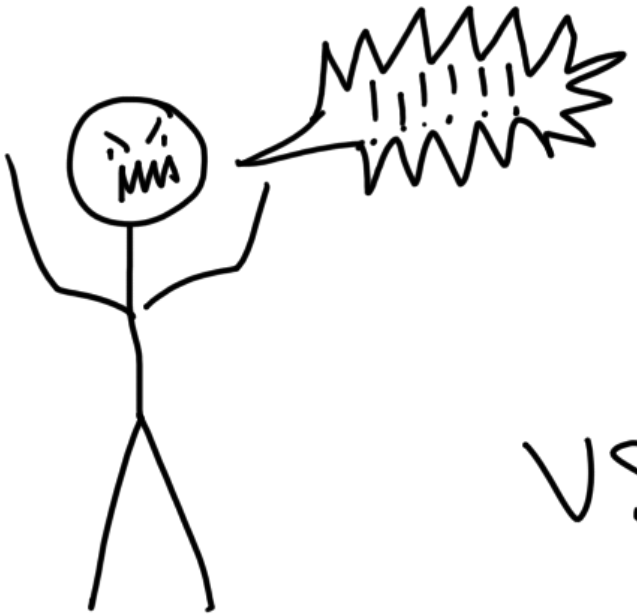

SAFETY


PERFORMANCE
SAFETY

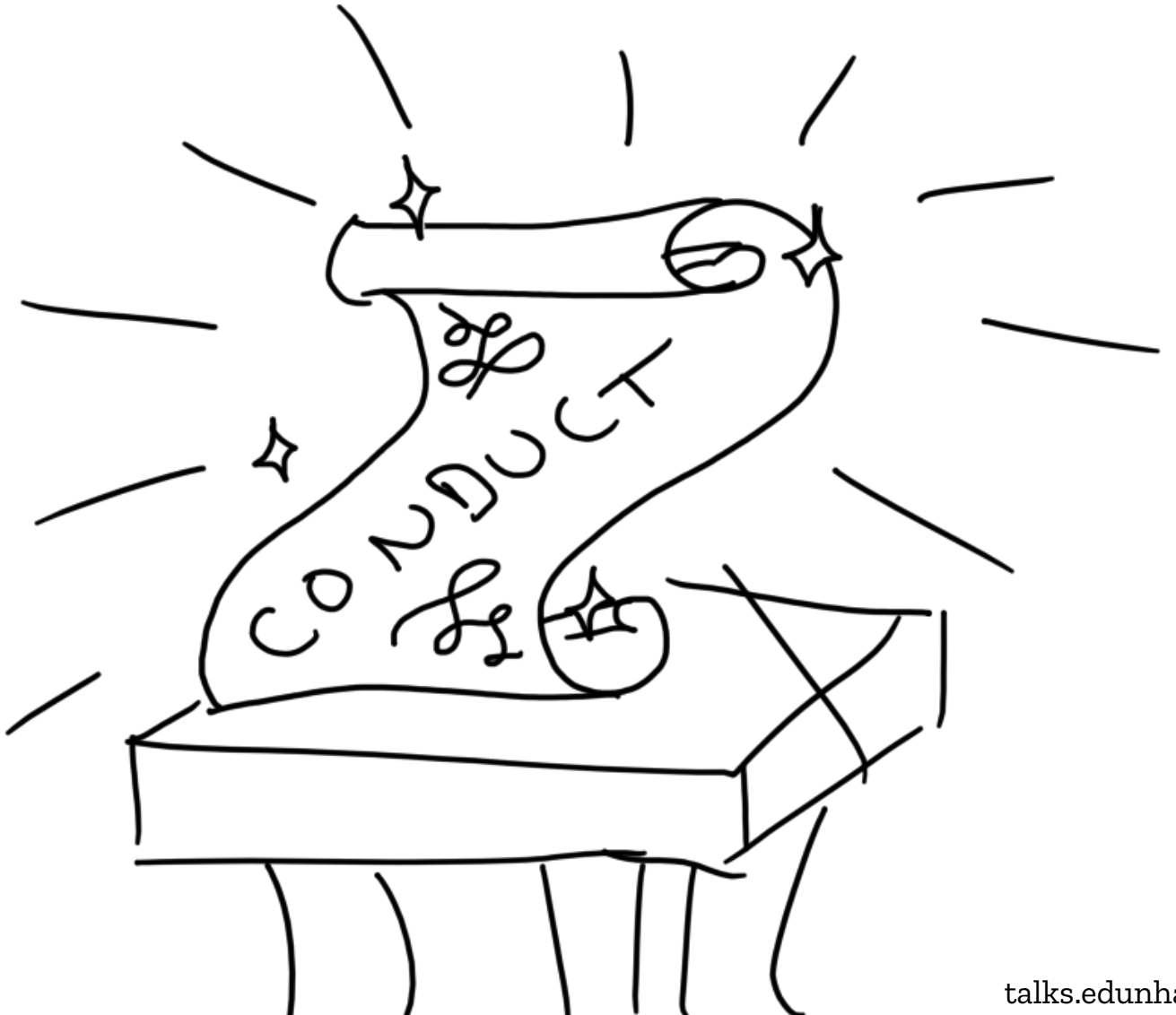

WHY NOT BOTH?



combine
old things
in new ways

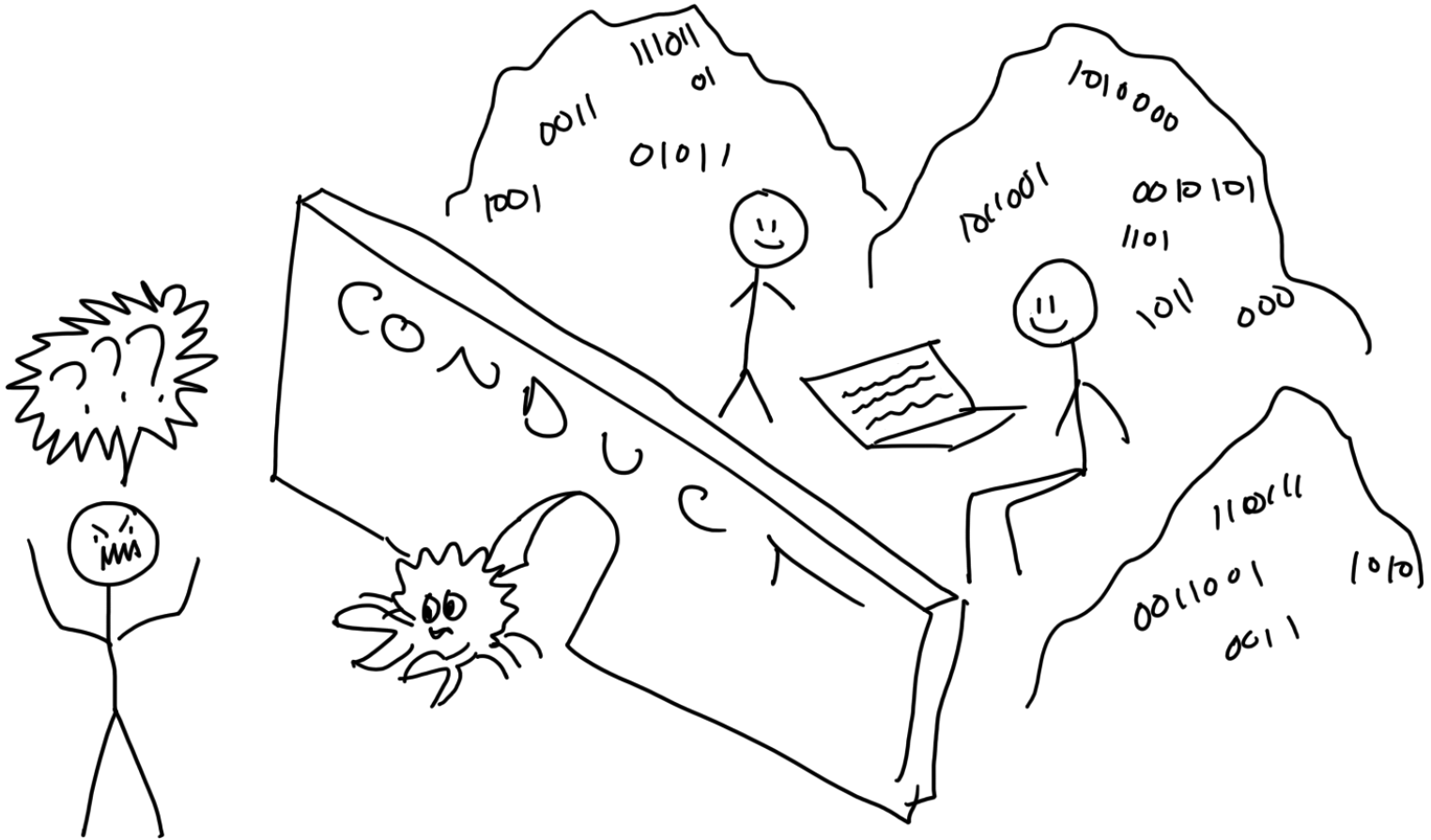


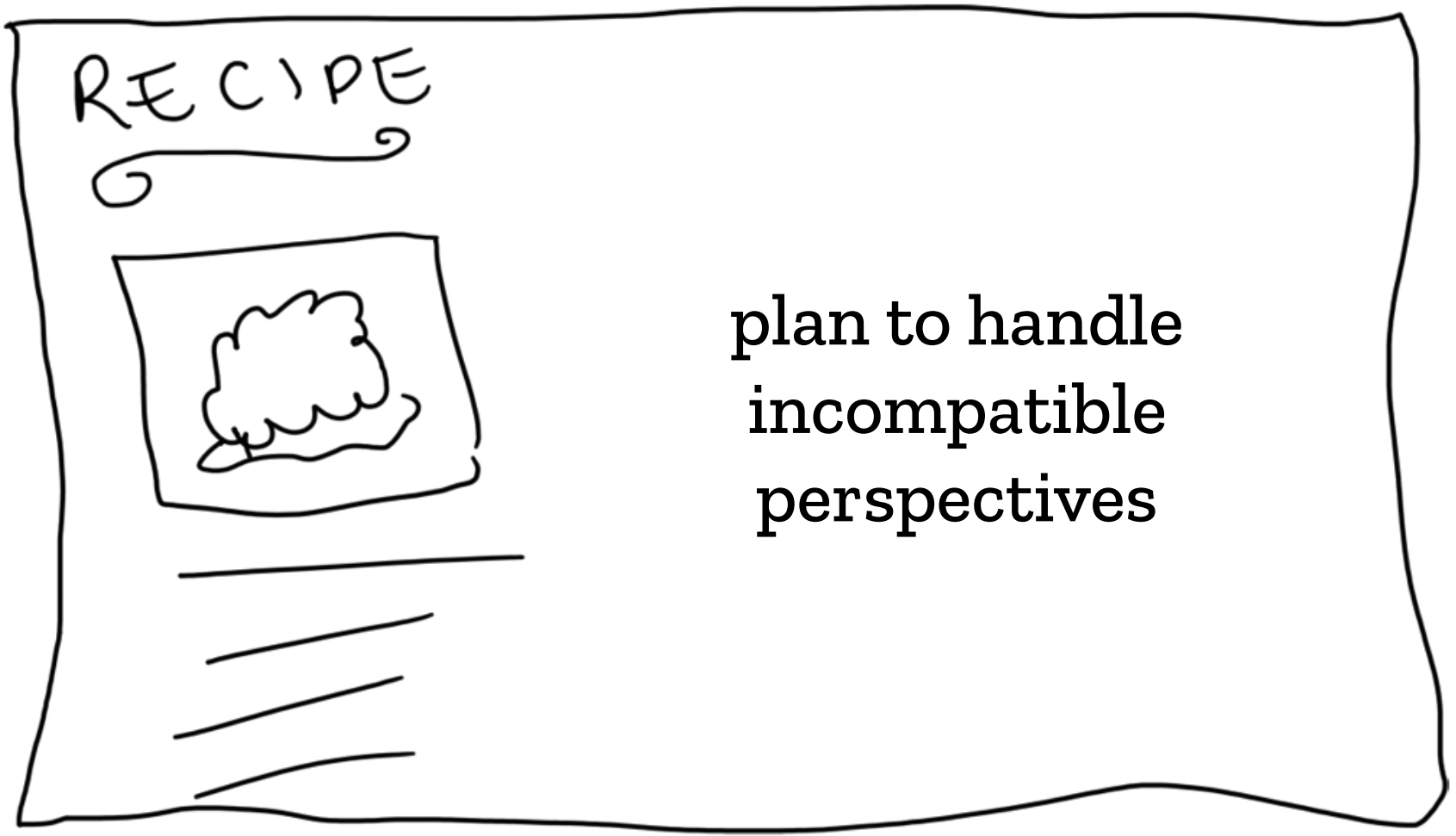
VS


CONDUCT

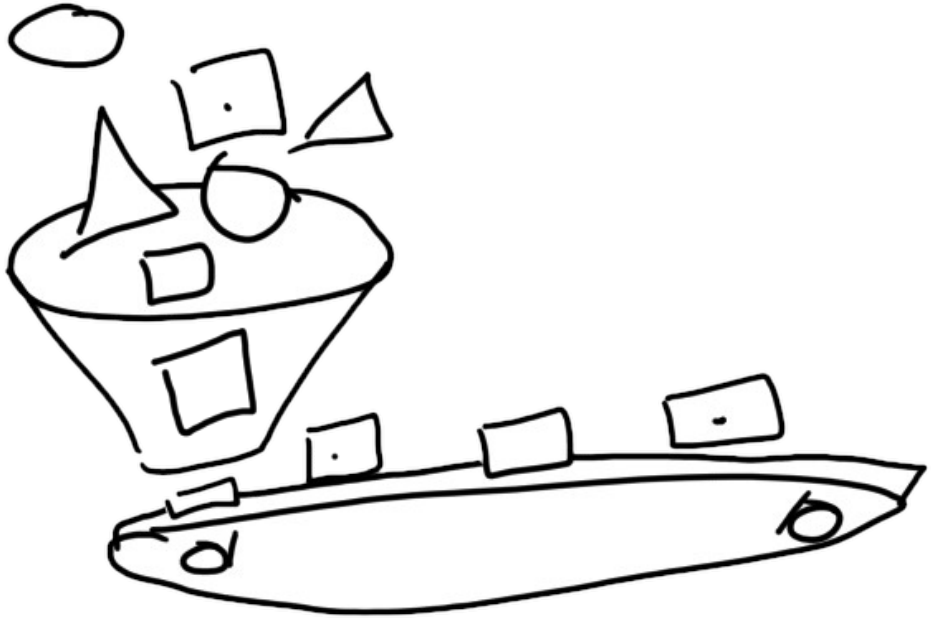
https://www.rust-lang.org/policies/code-of-conduct

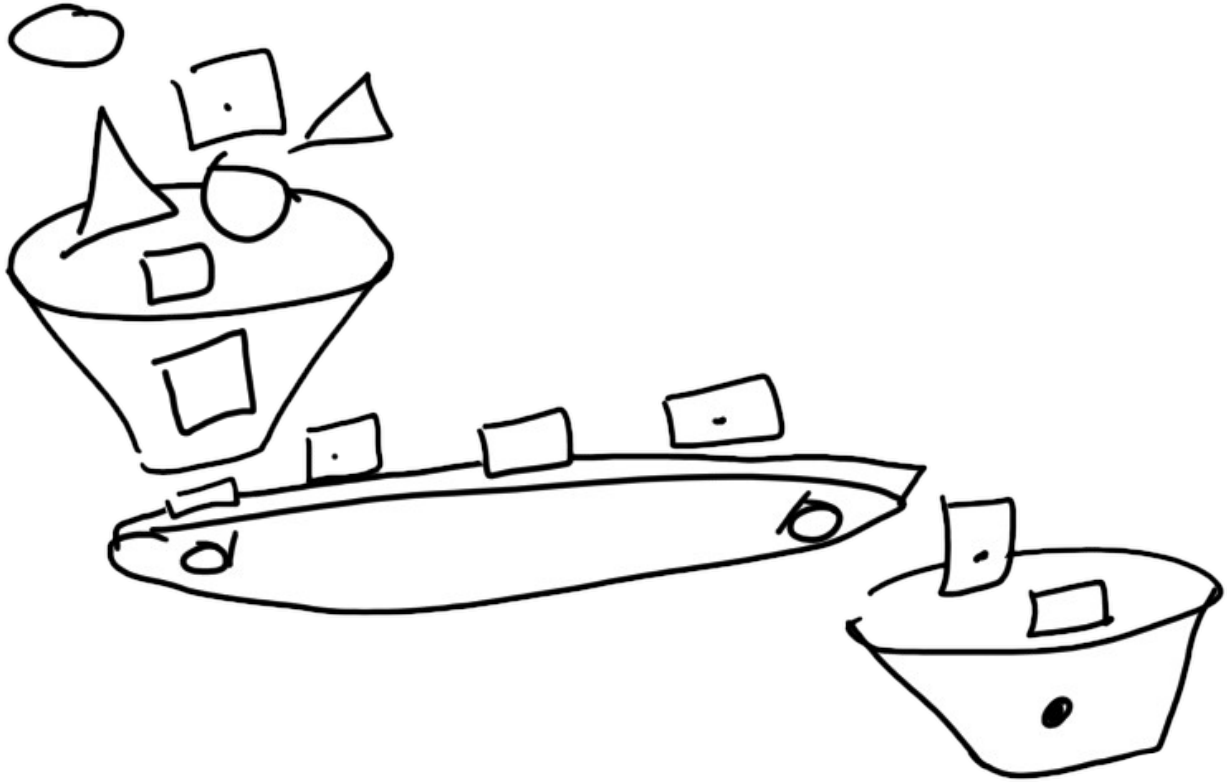
https://www.rust-lang.org/policies/code-of-conduct

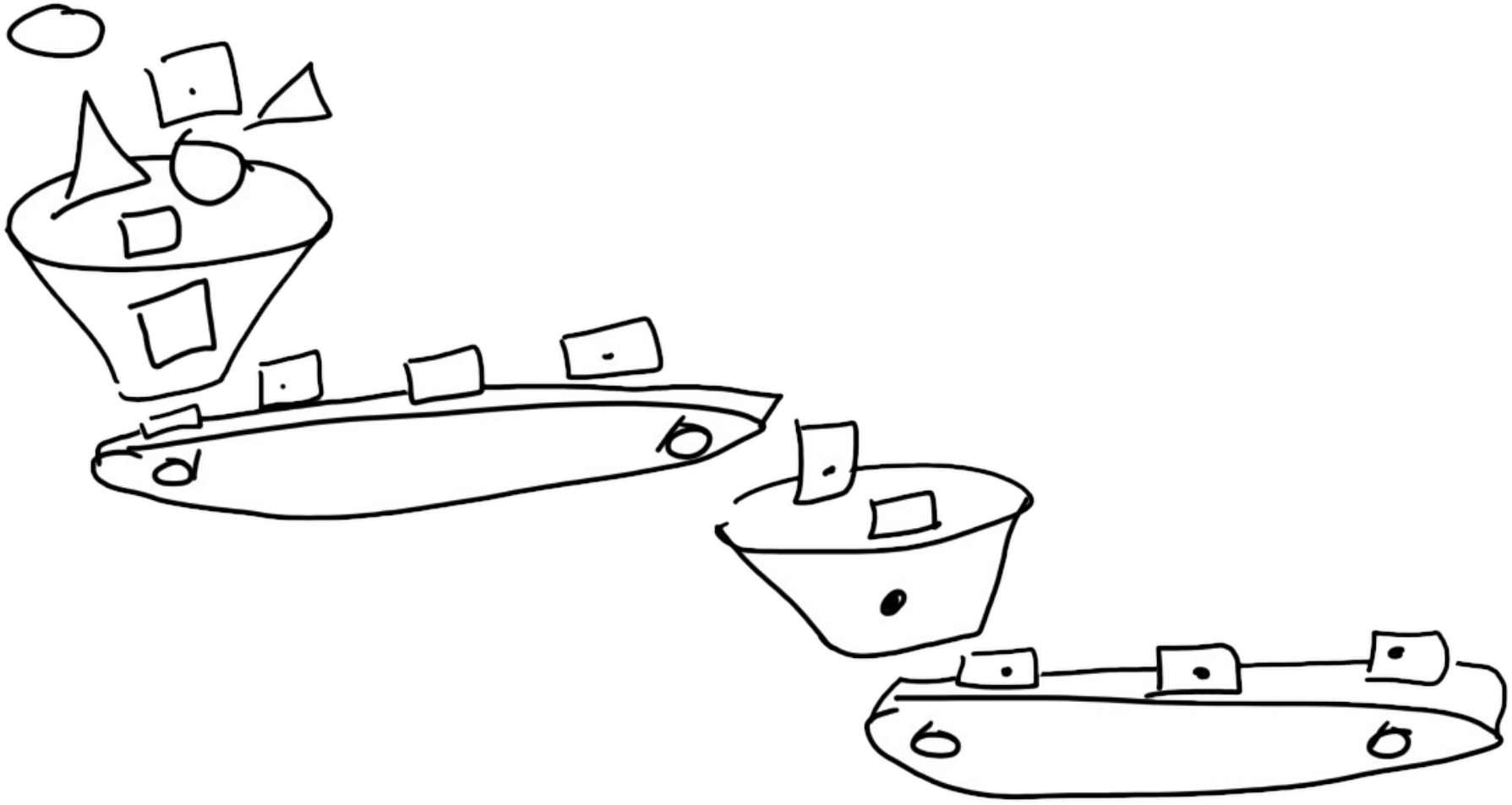

CONDUCT

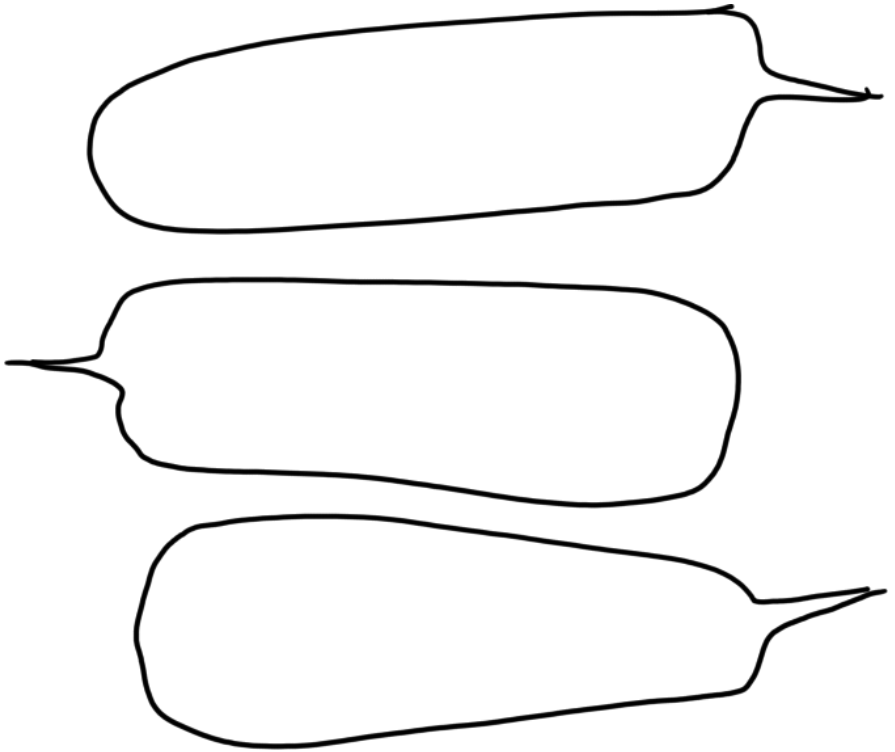


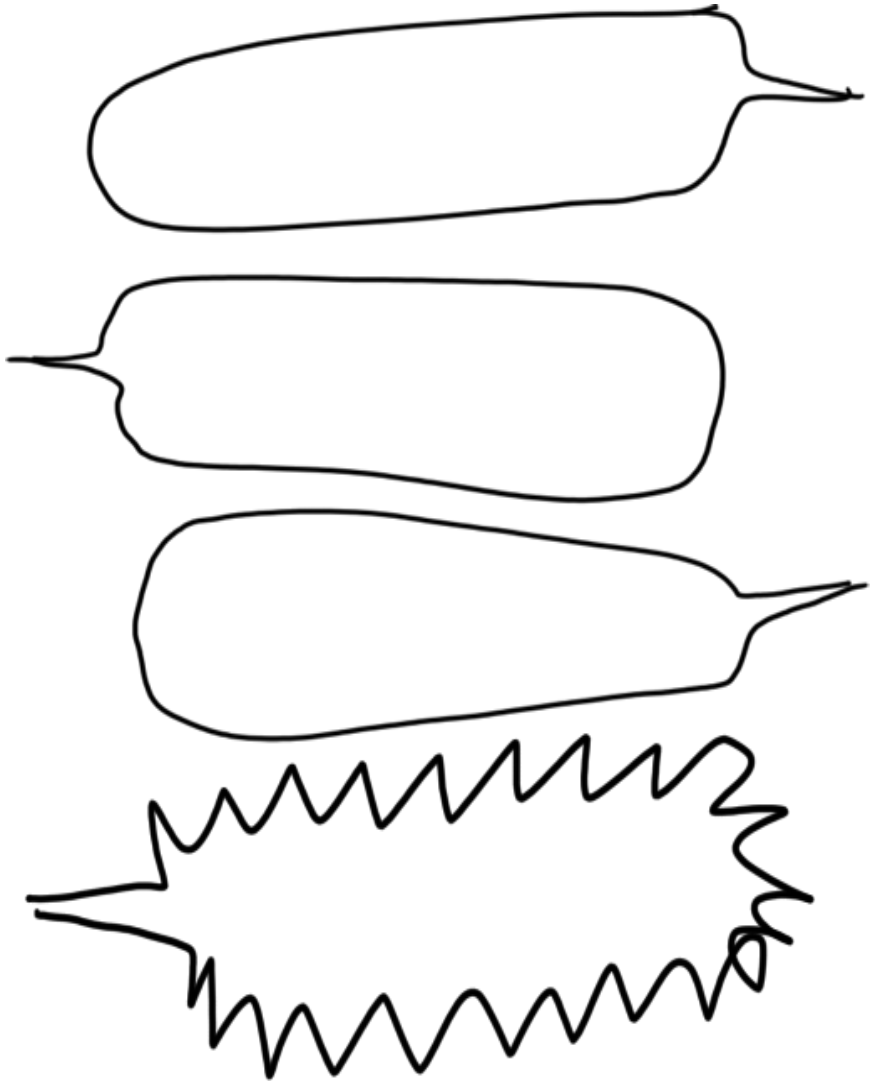
plan to handle
incompatible
perspectives

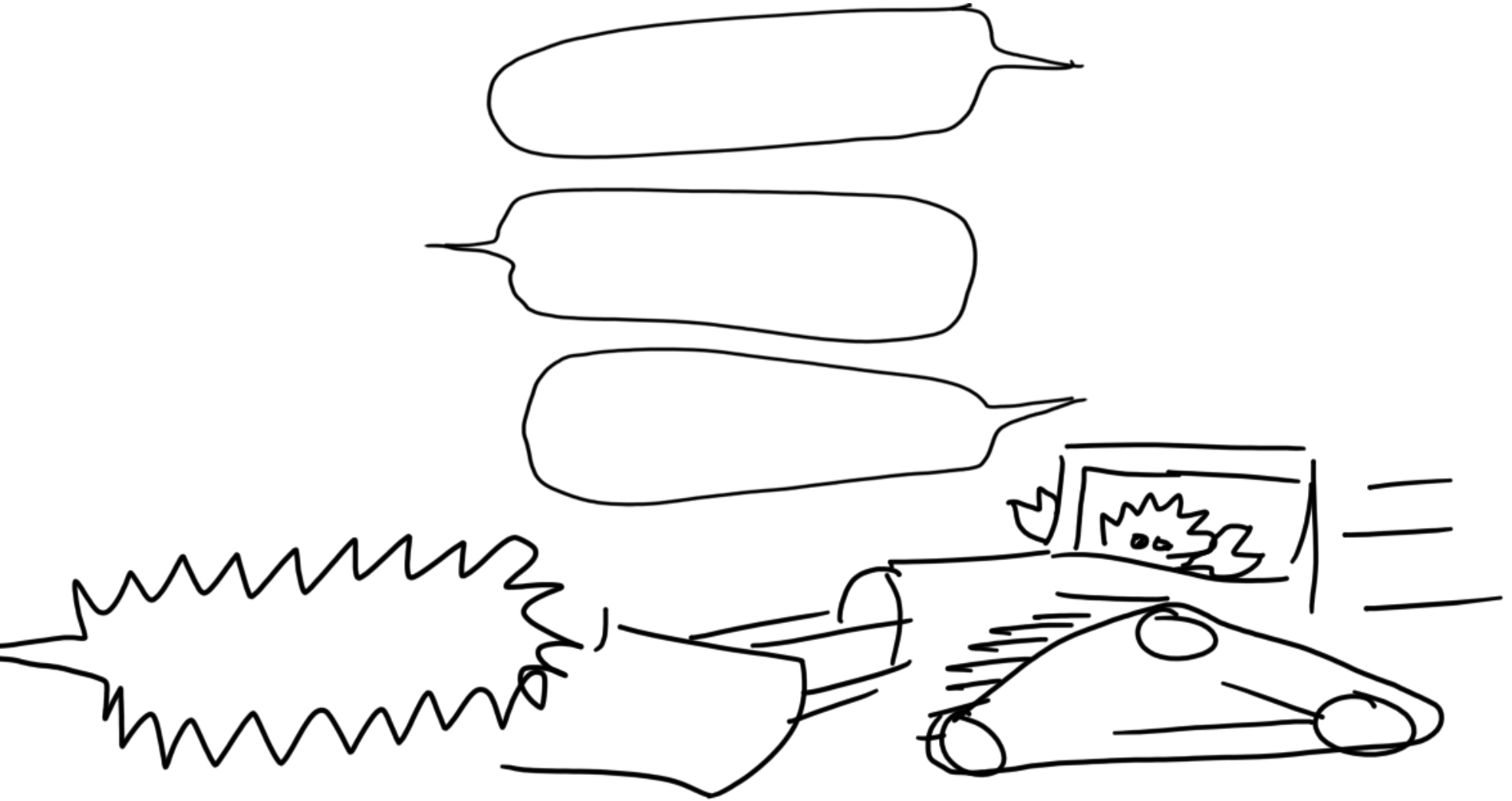


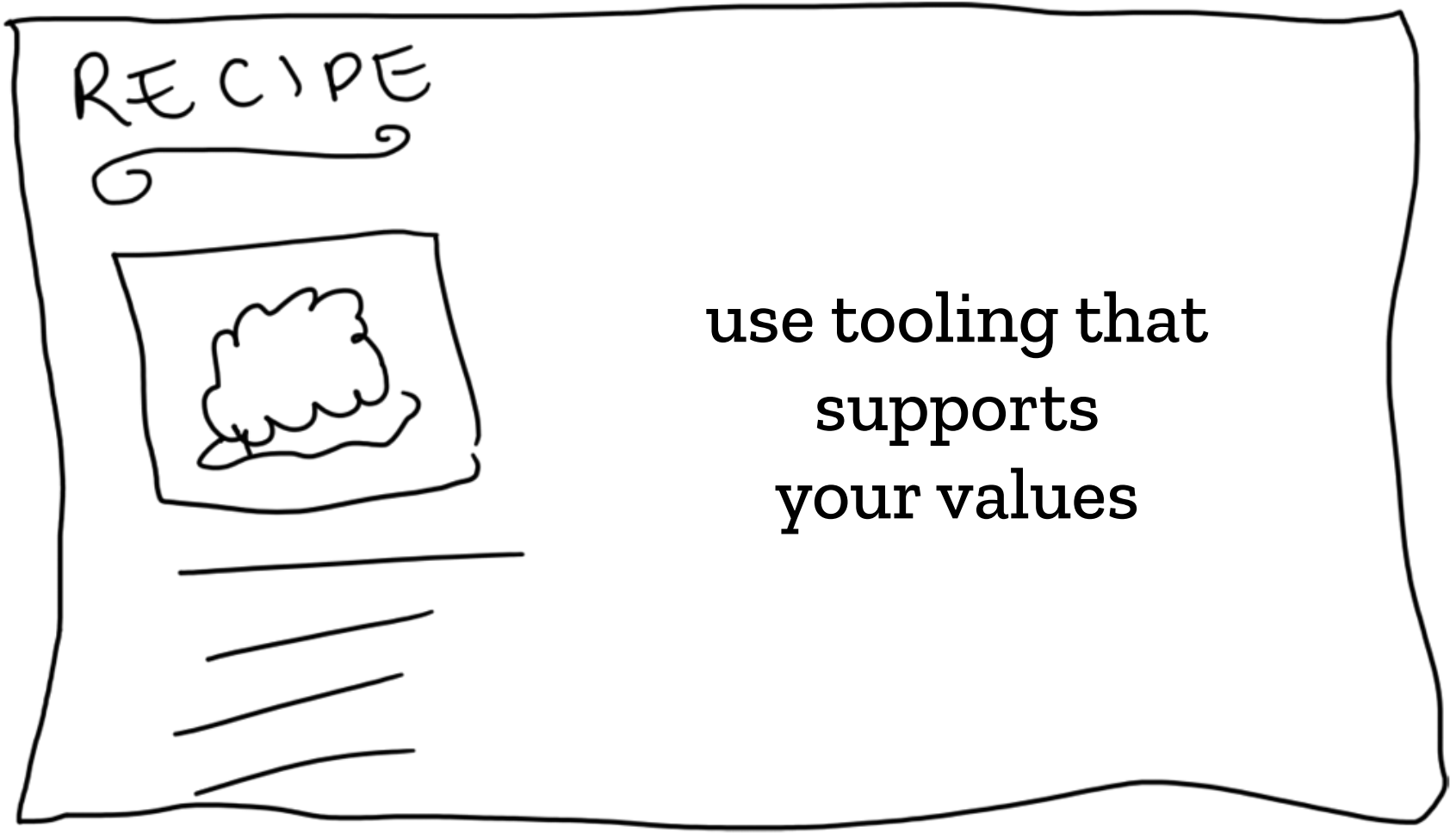




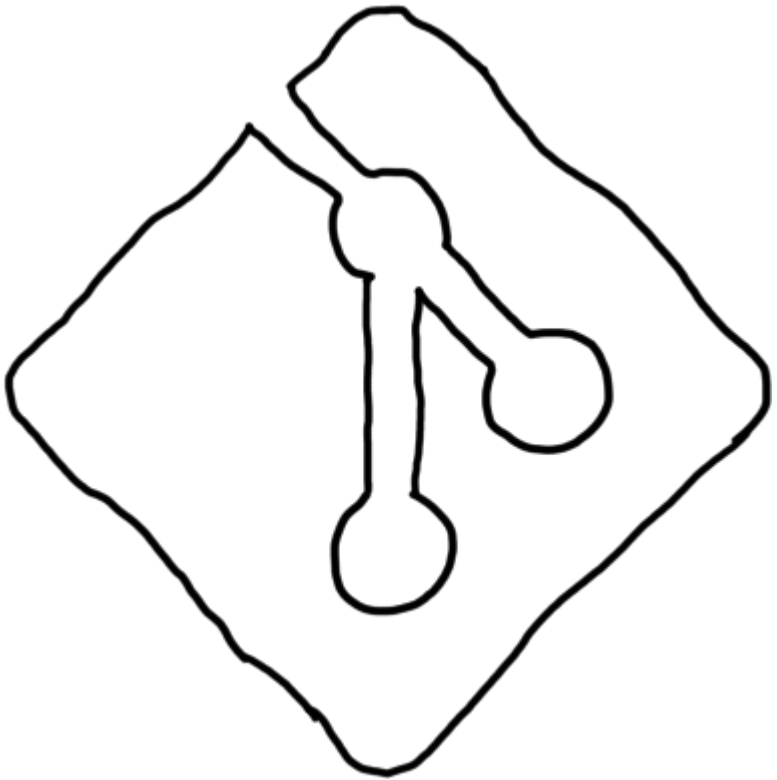




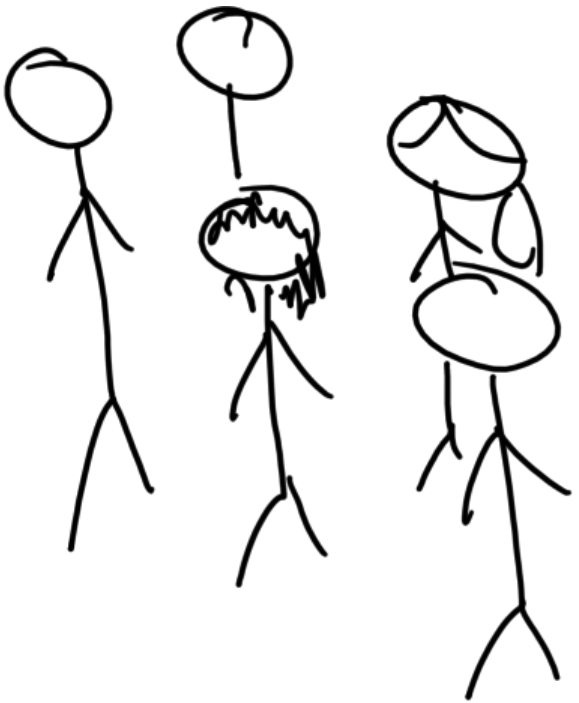


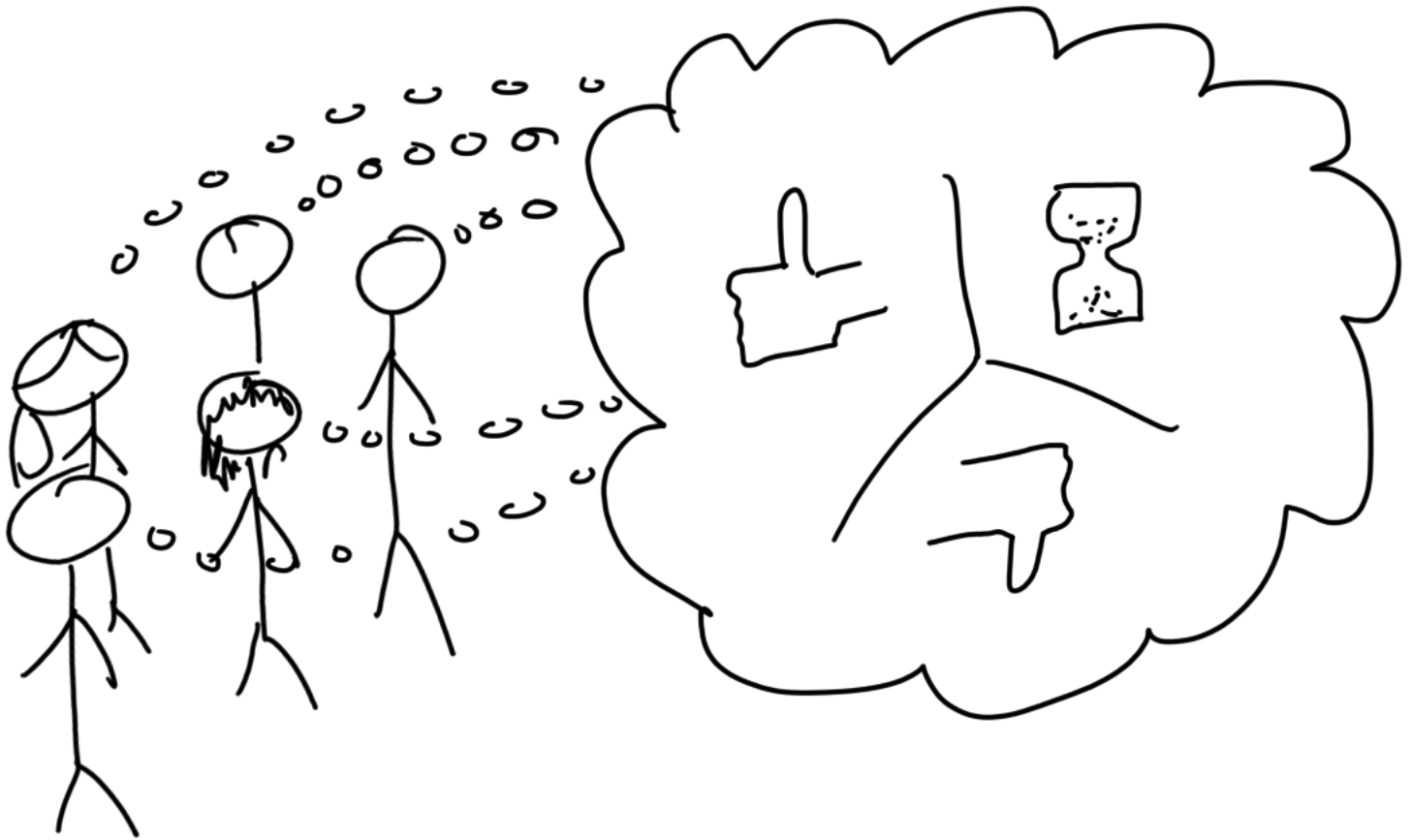


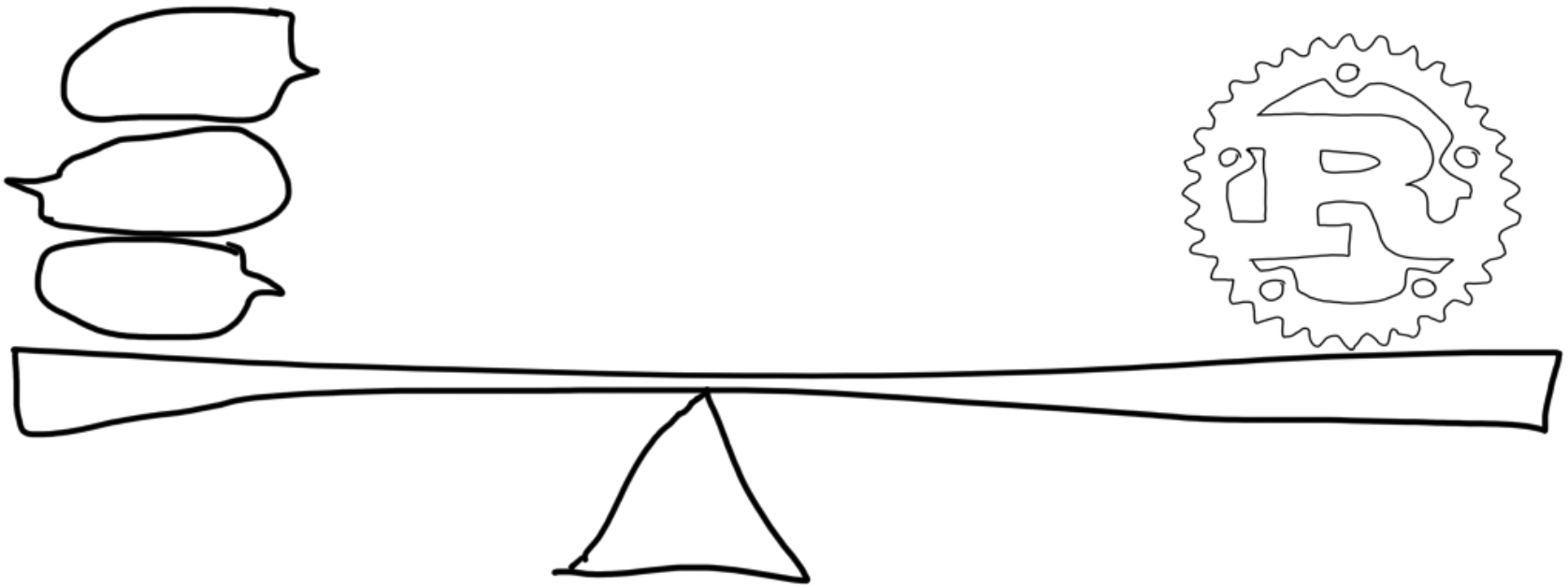
use tooling that
supports
your values

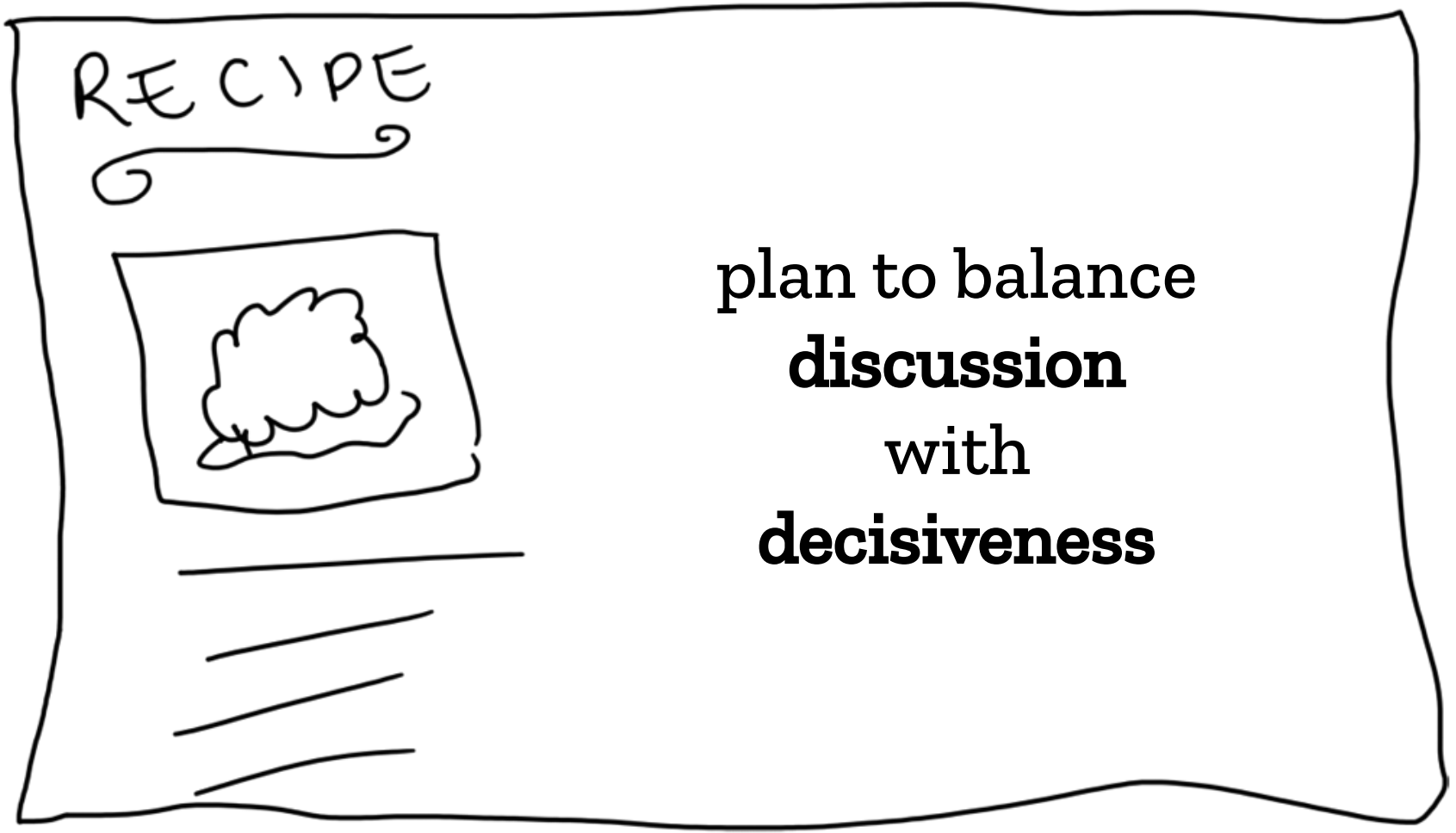



RFC

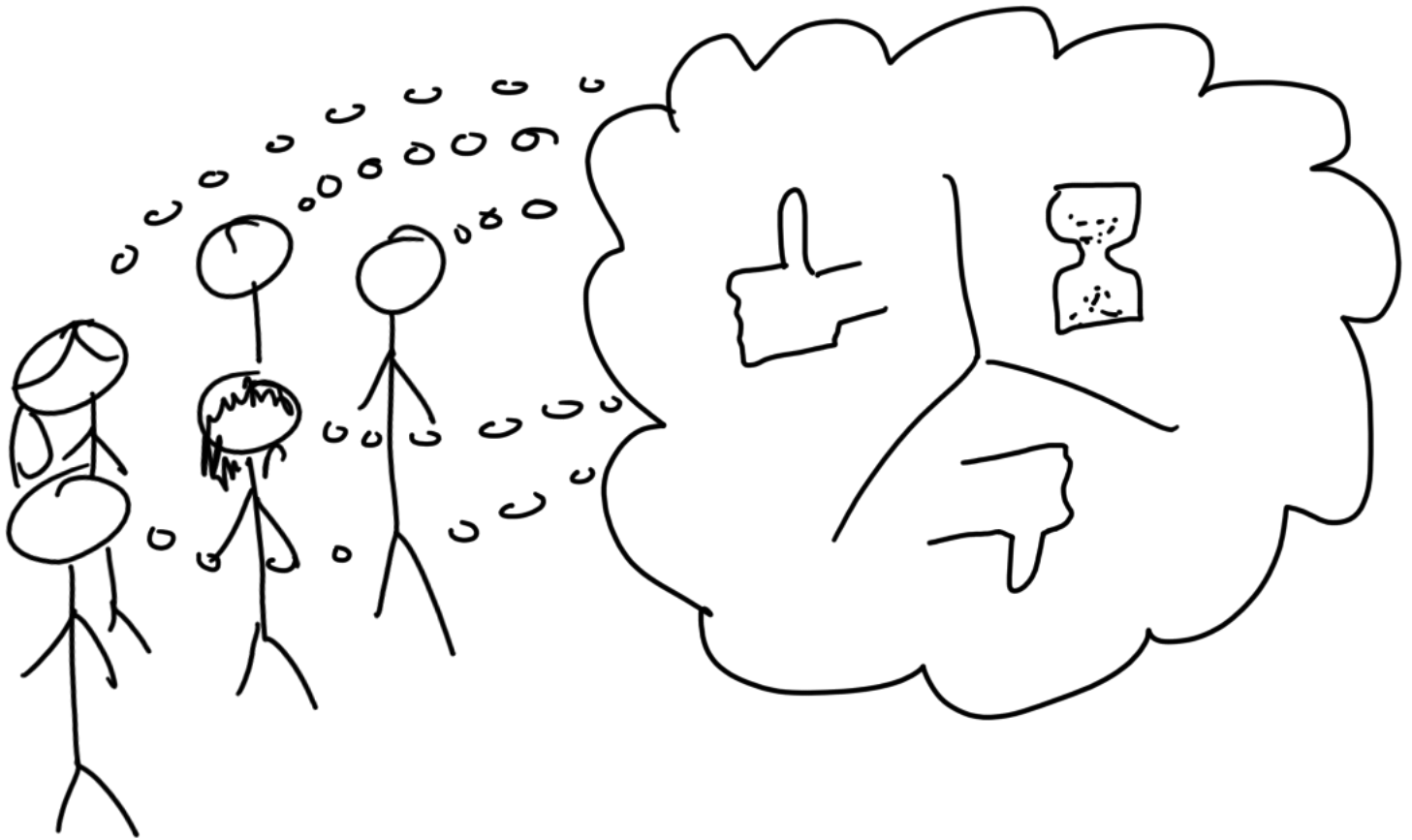


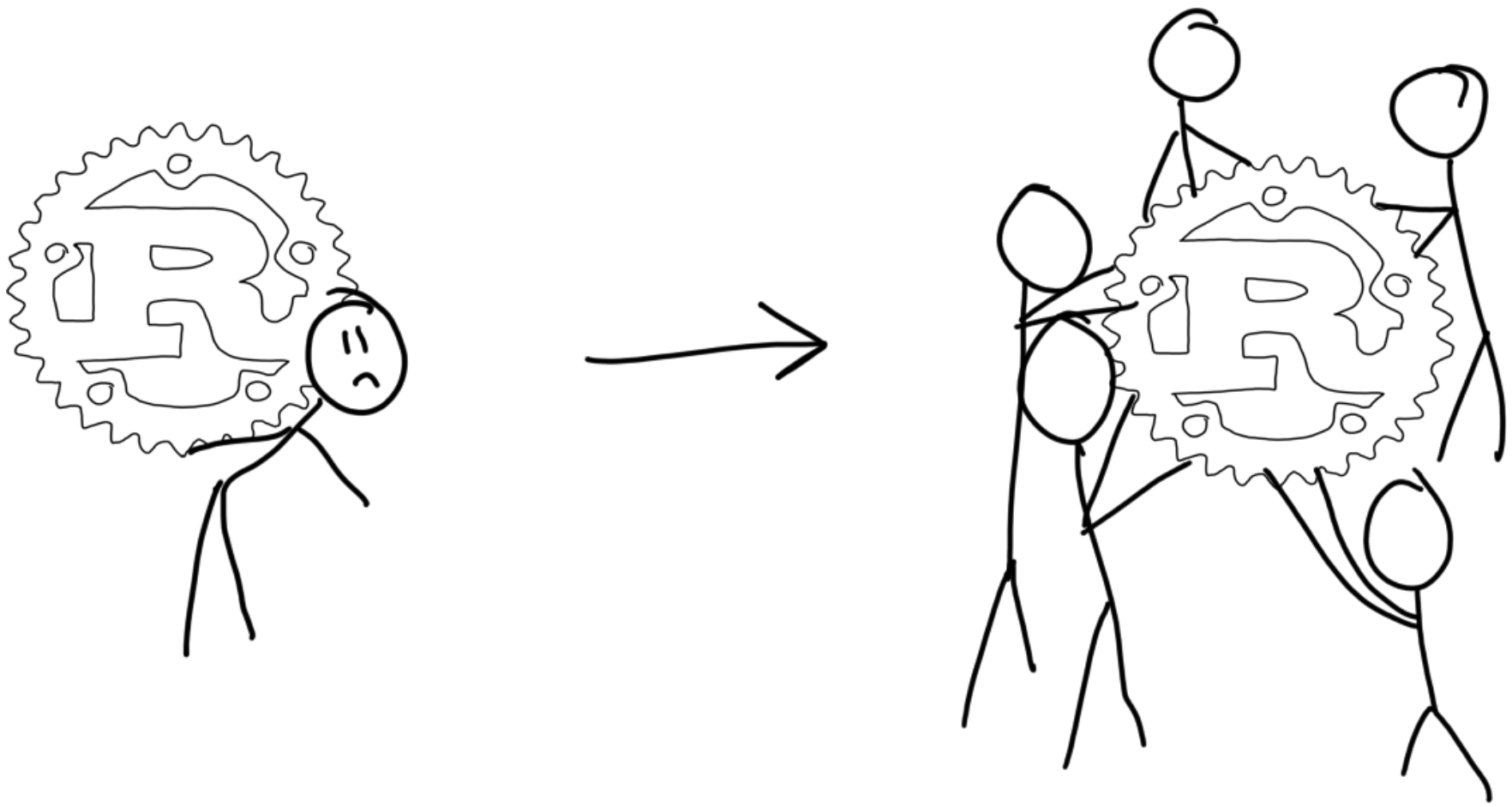


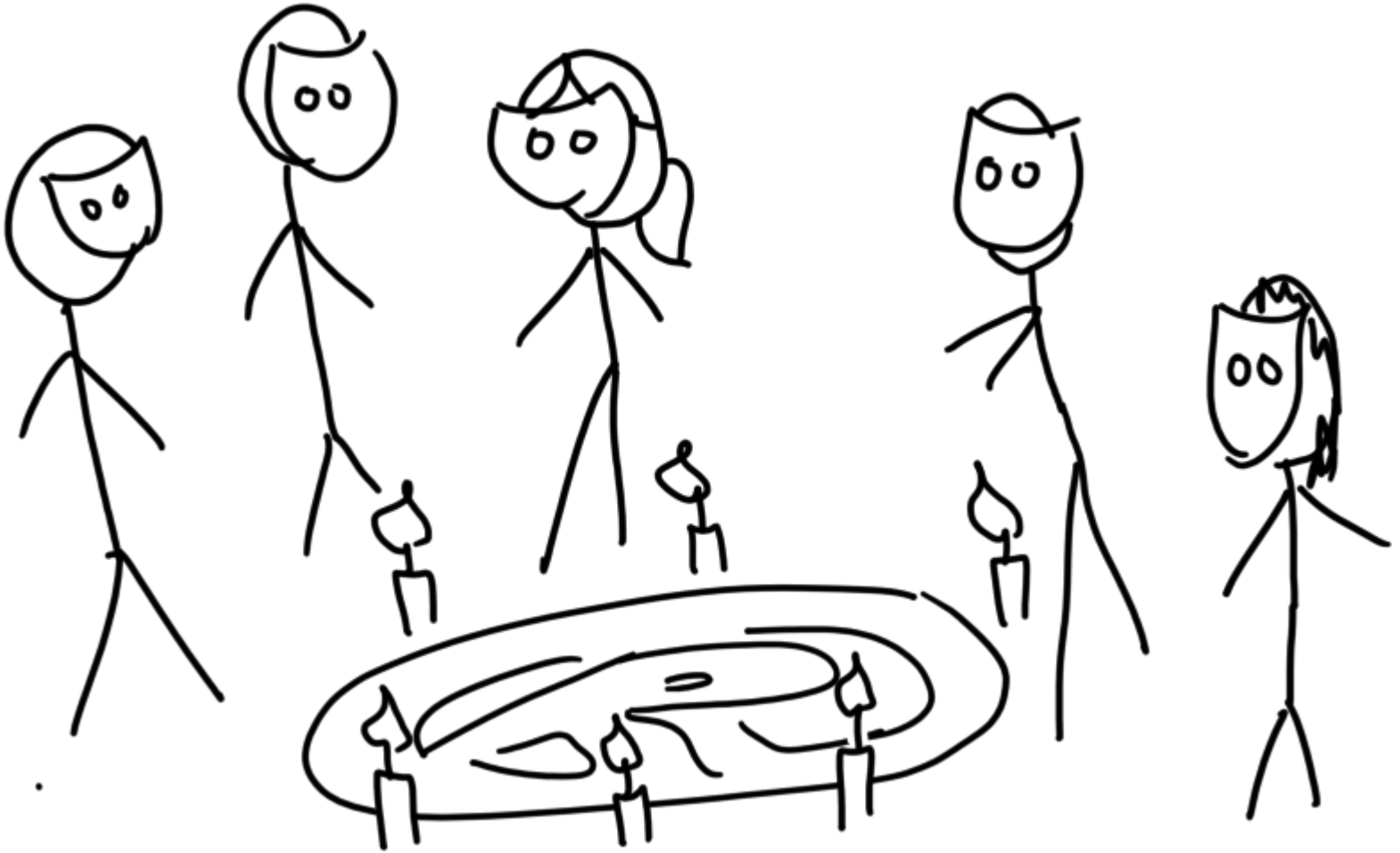




plan to balance
discussion
with
decisiveness

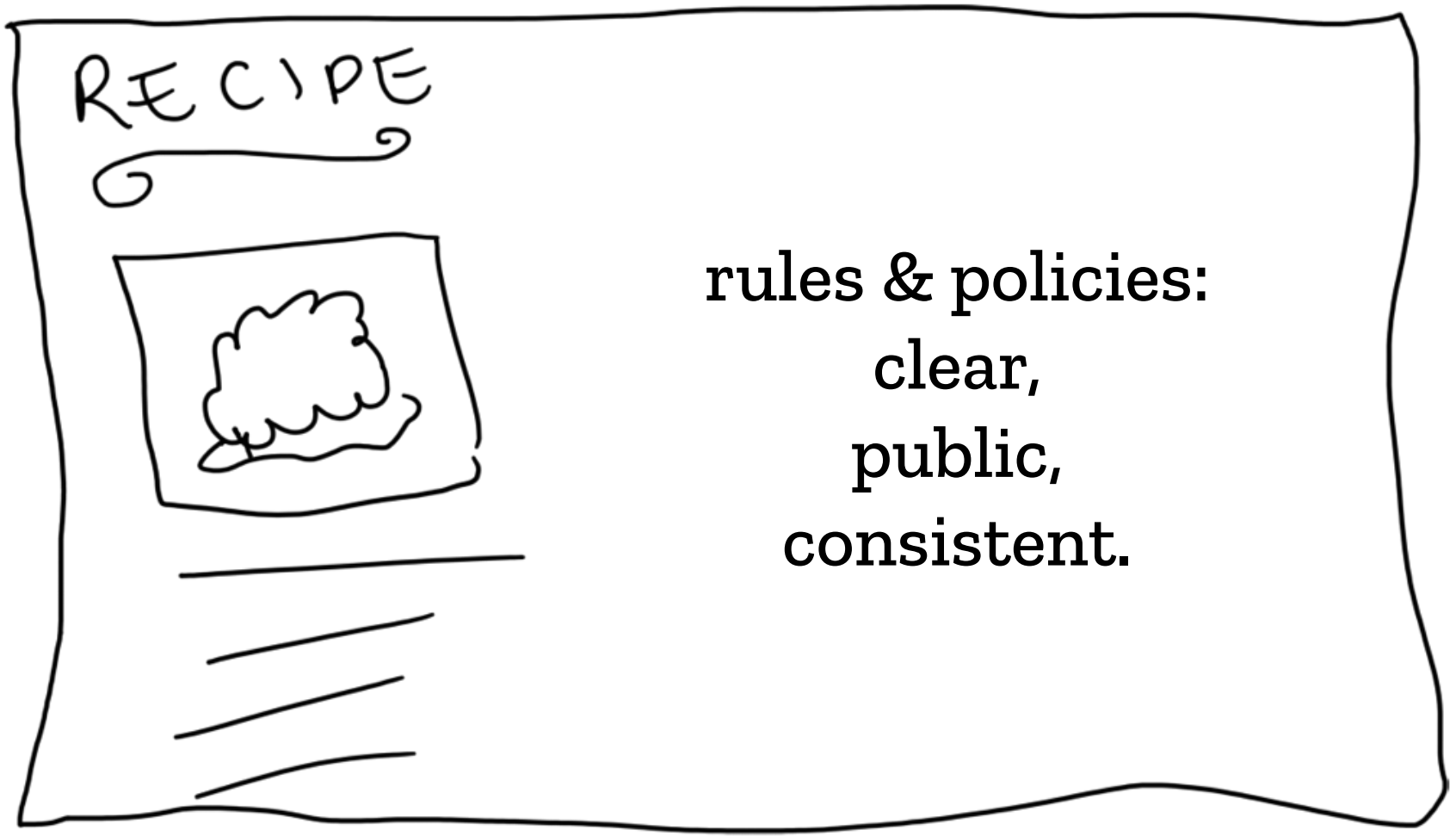




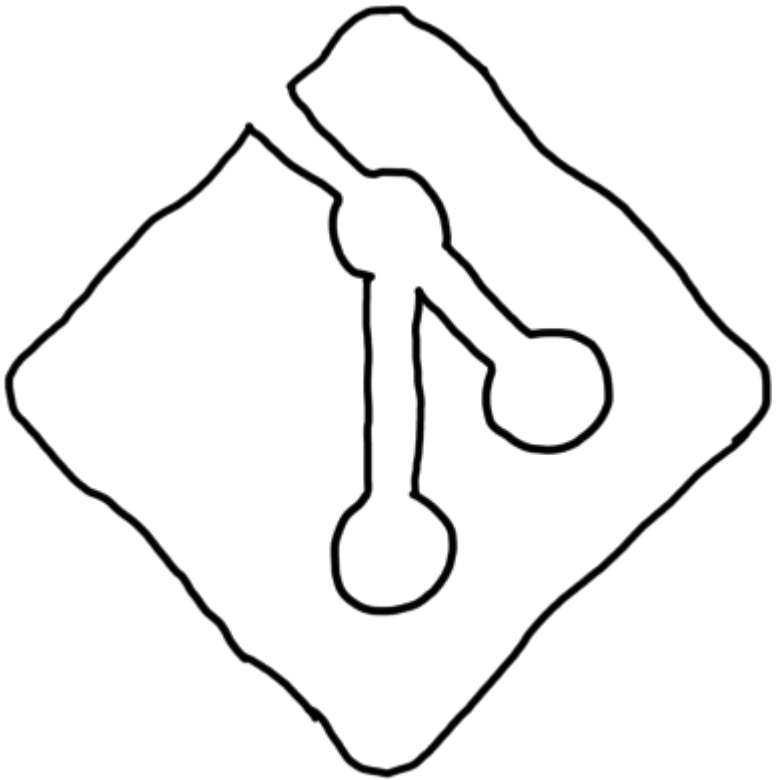


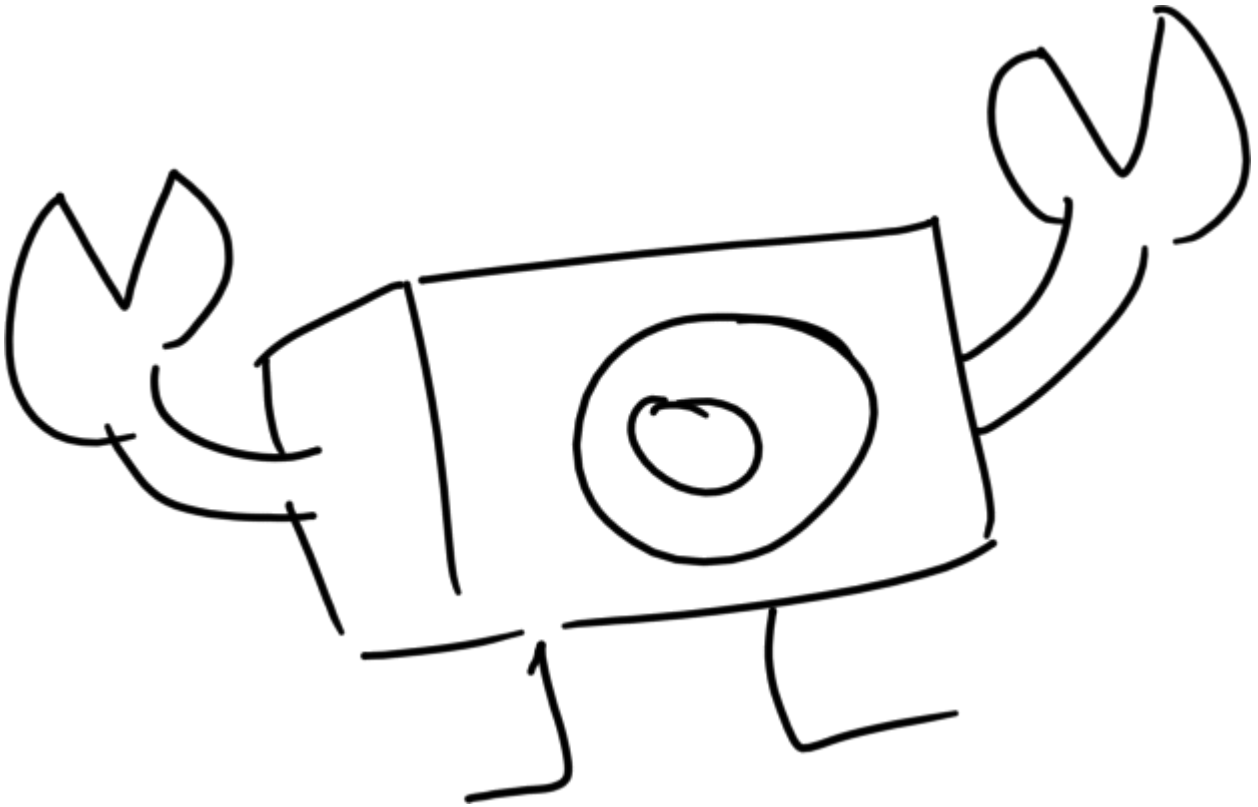
"the **argument supporting the disposition on the RFC**
needs to have **already been clearly articulated**,
and there should **not be a strong consensus *against* that position**
outside of the subteam."

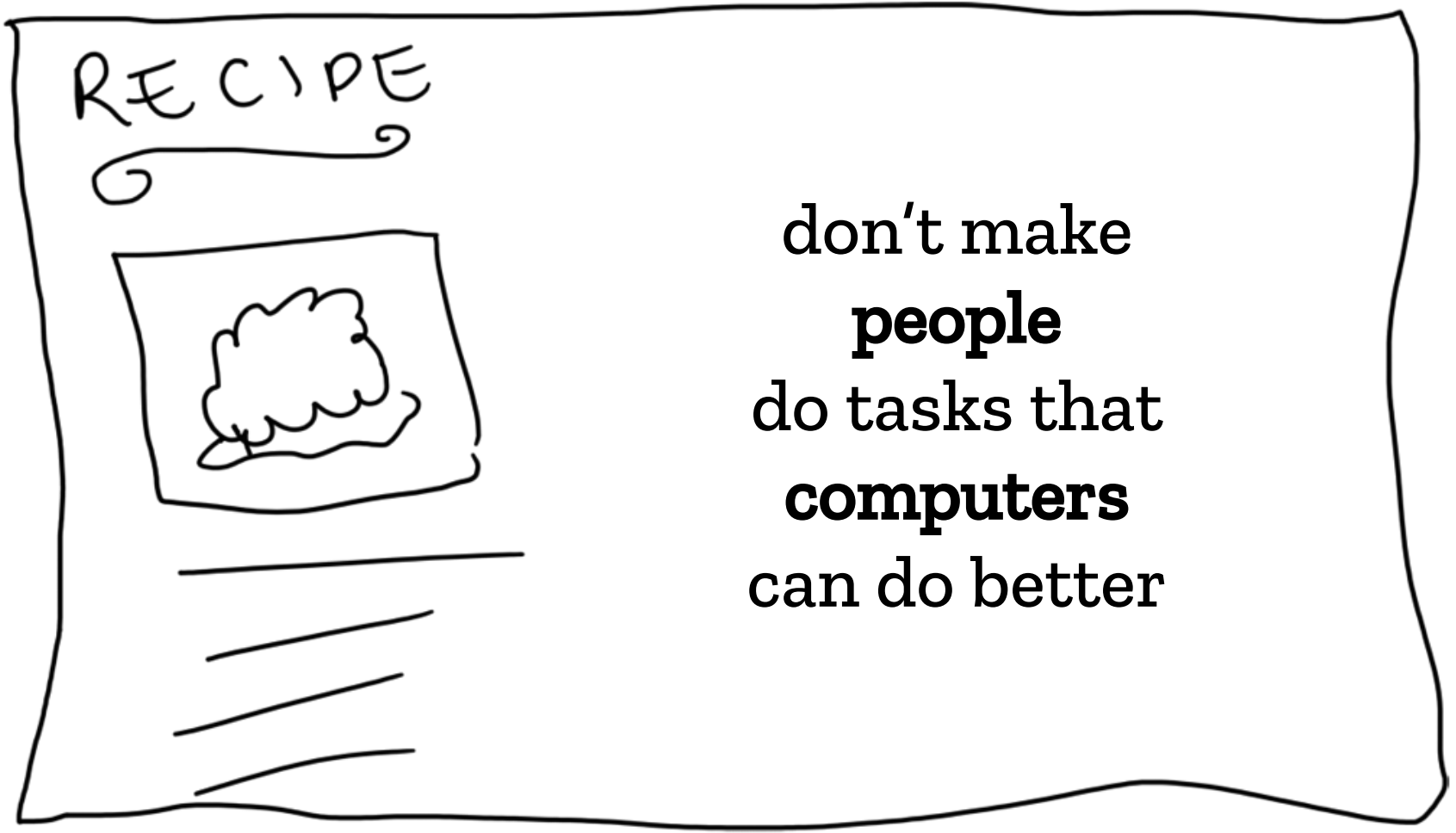
– github.com/rust-lang/rfcs README



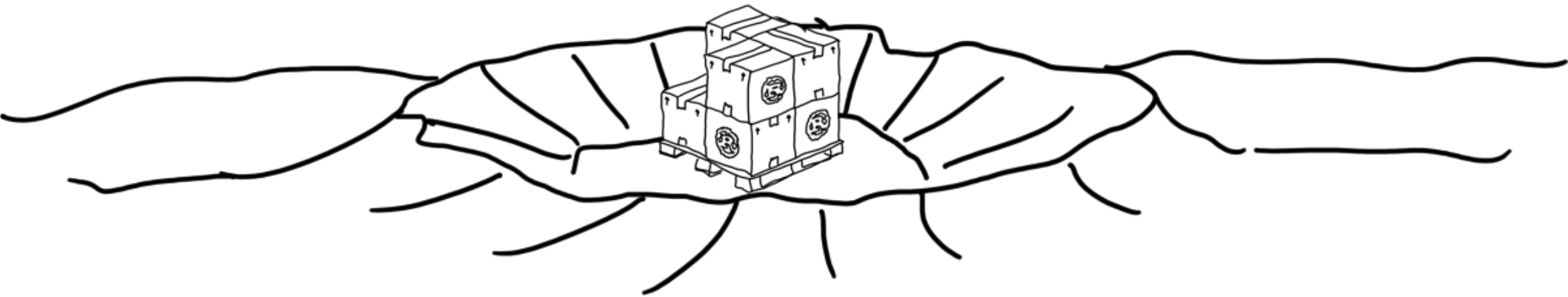
rules & policies:
clear,
public,
consistent.



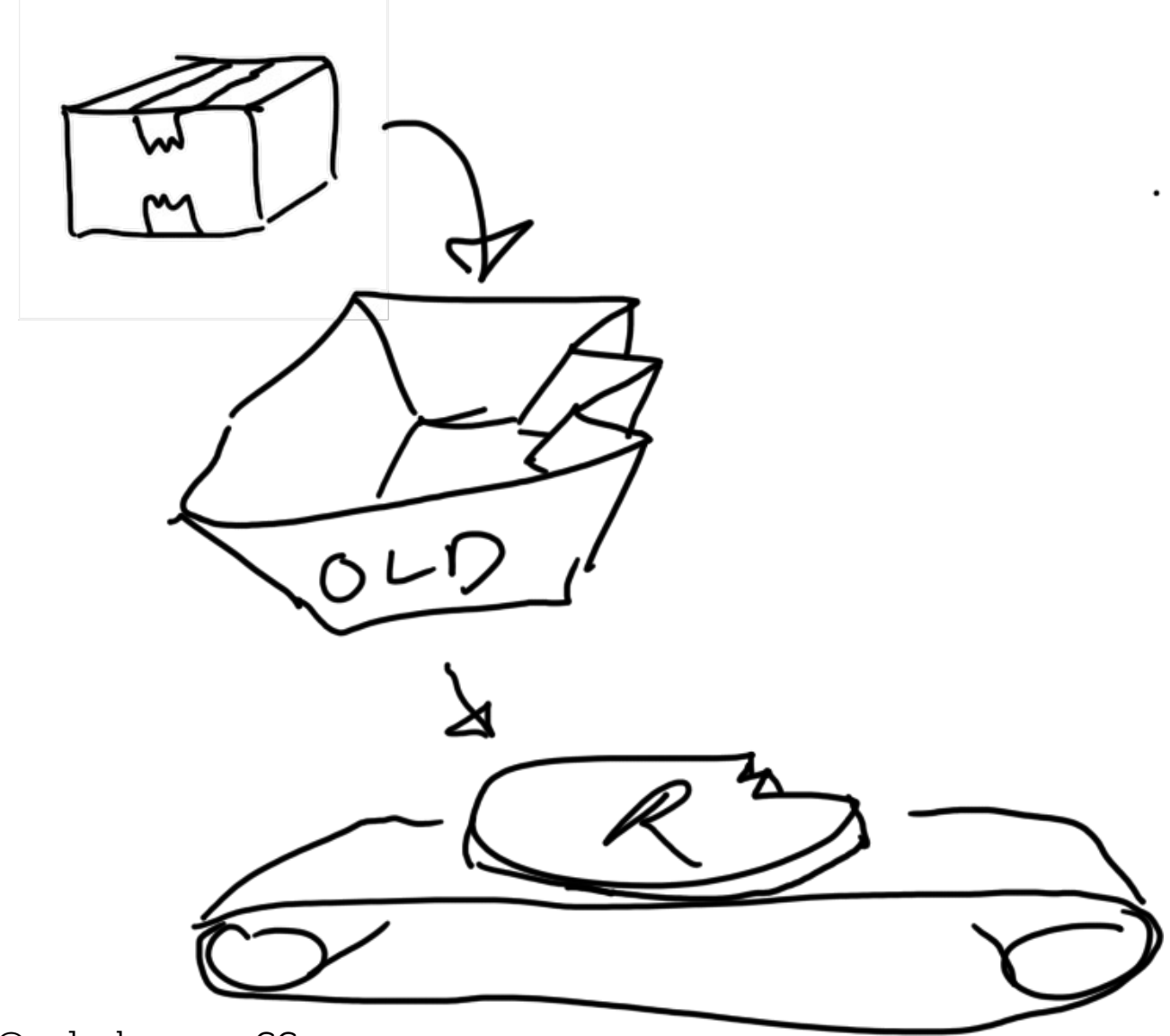


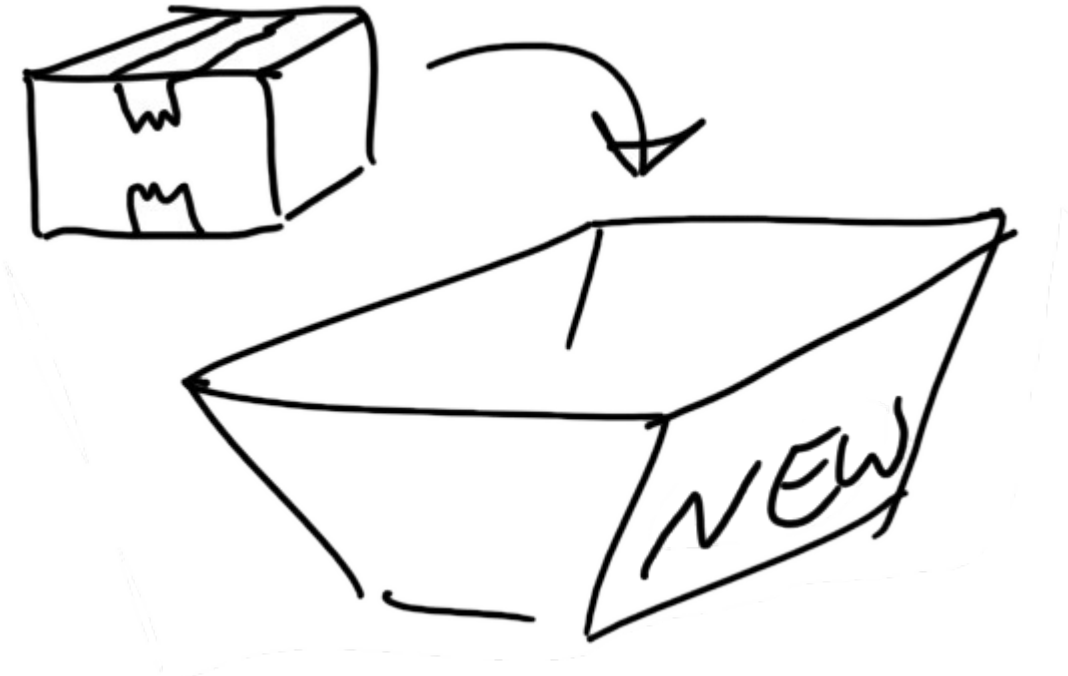


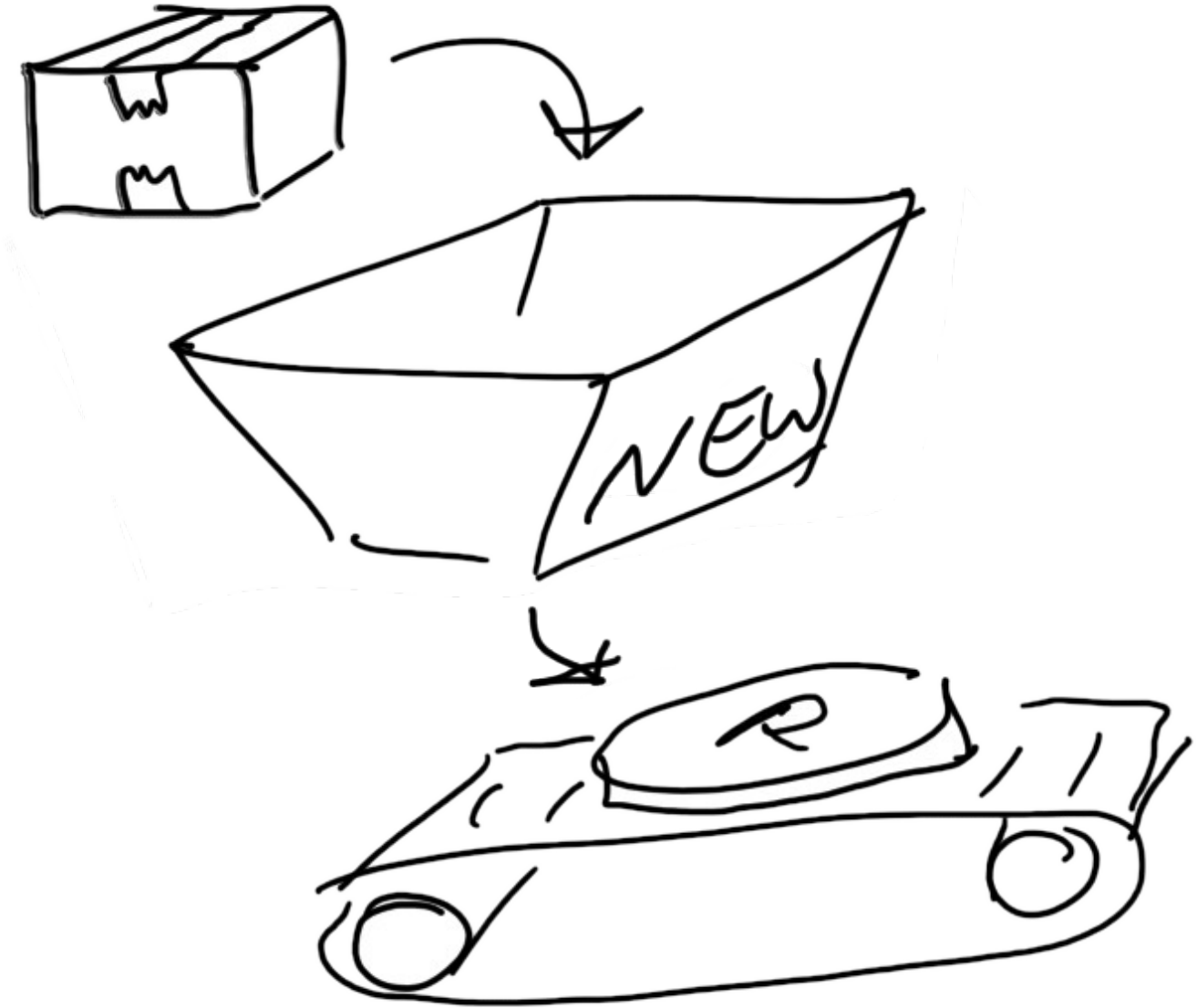
don't make
people
do tasks that
computers
can do better



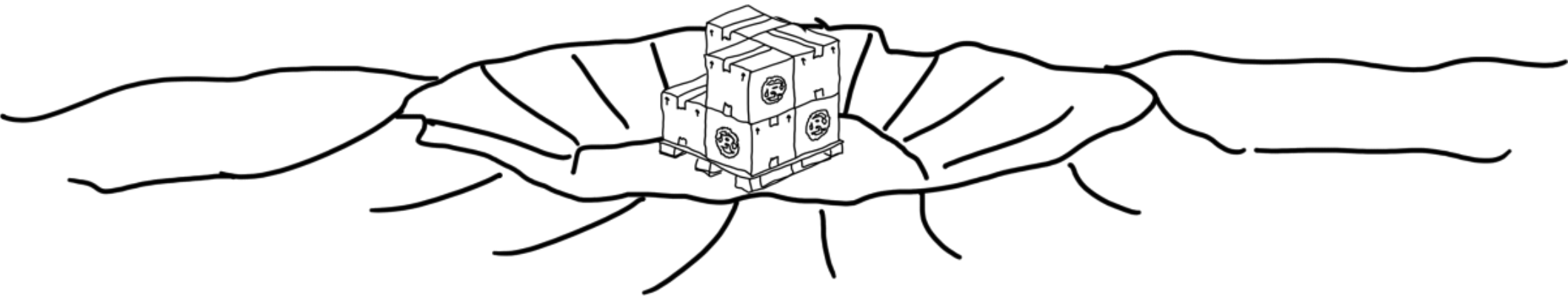

OLD

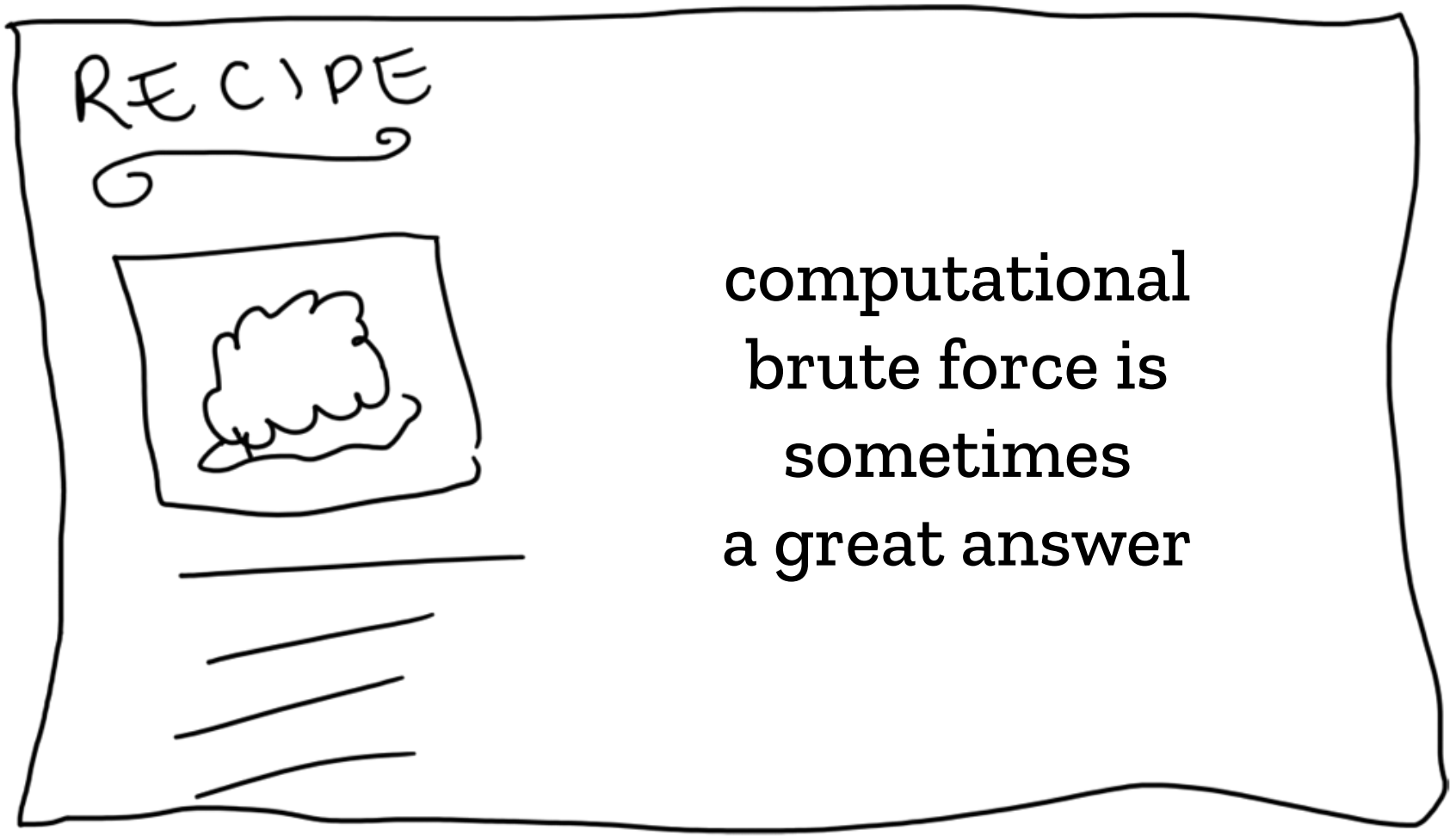

OLD
R


NEW

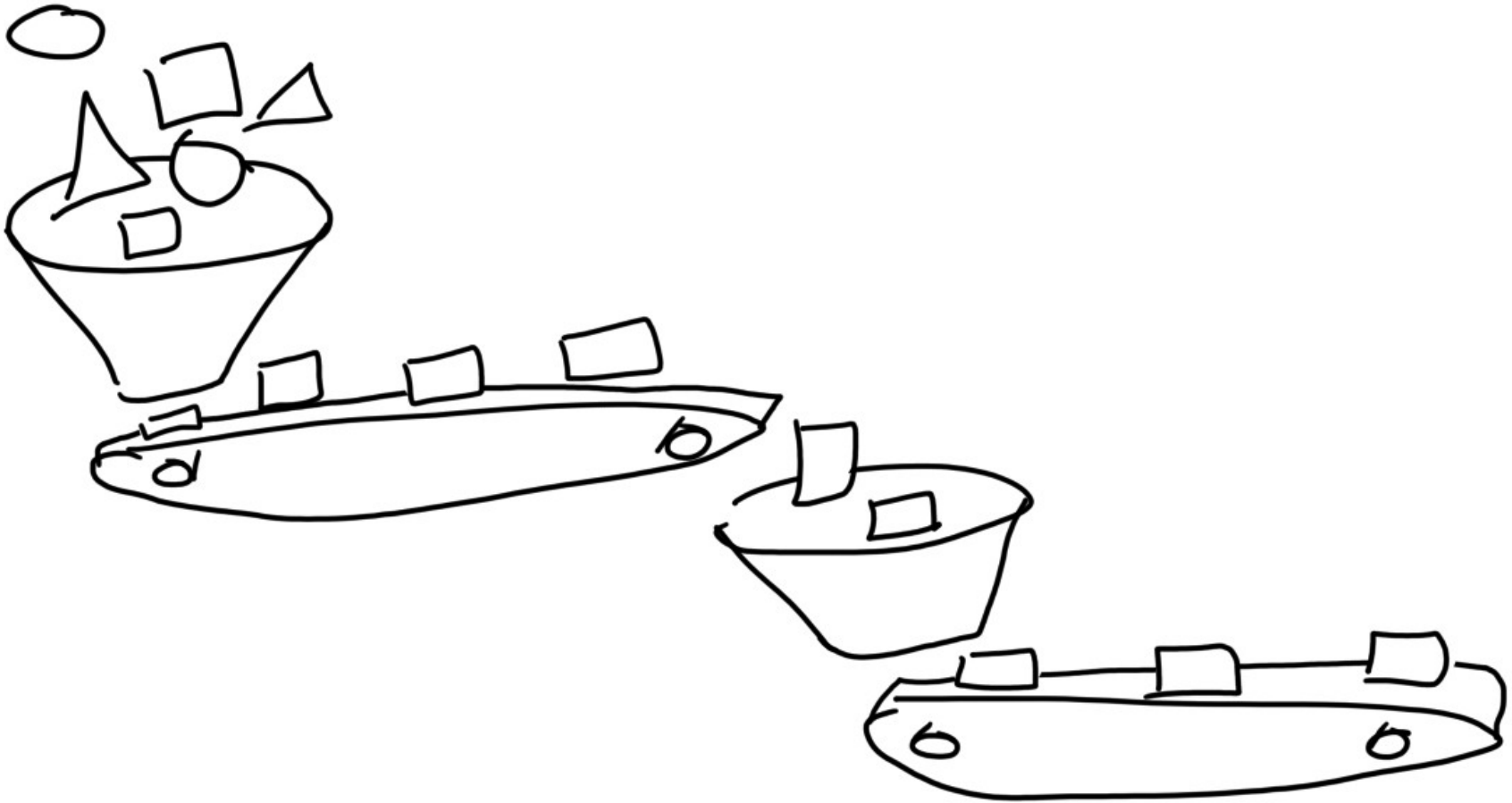

NEW

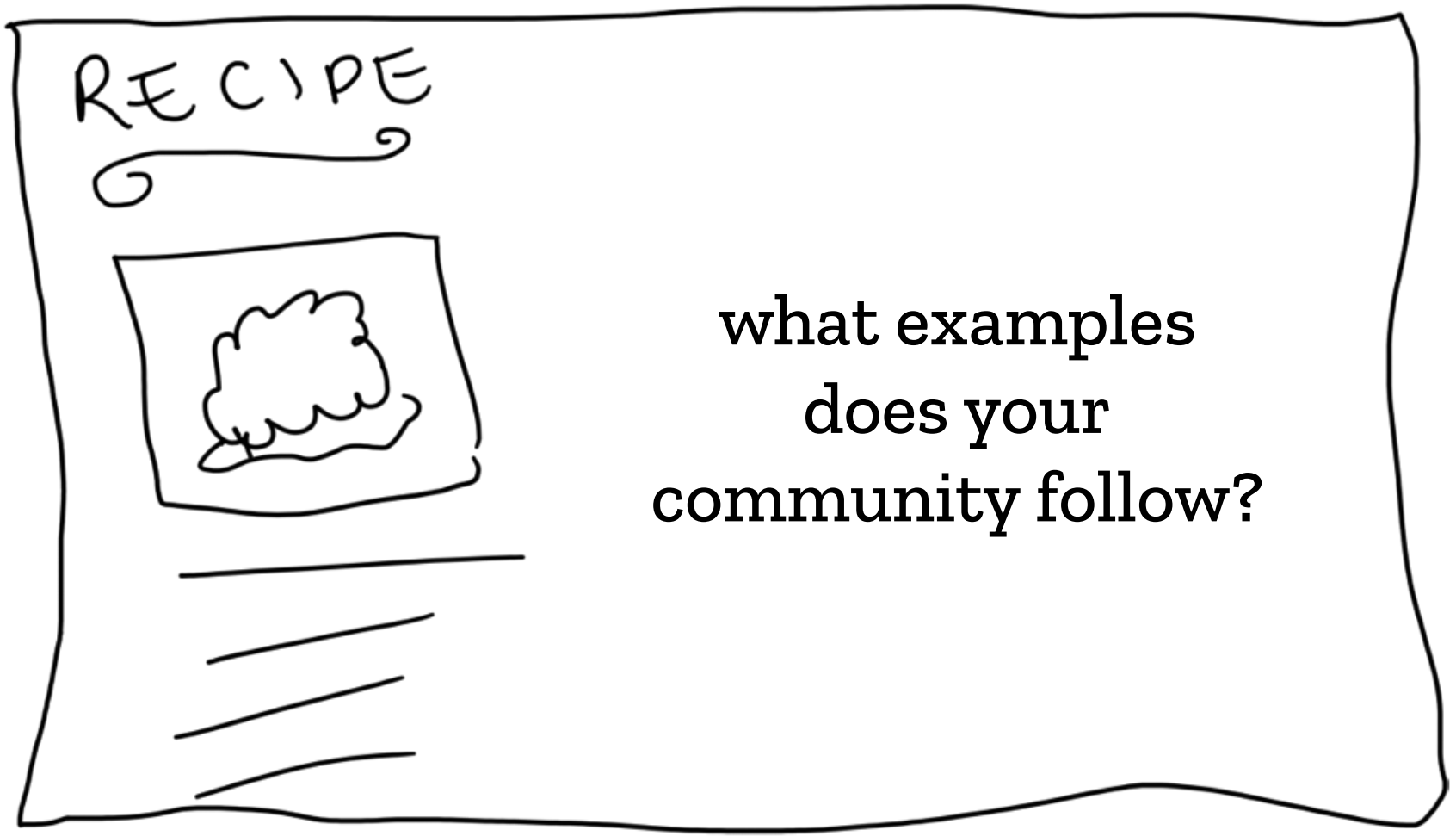

VS



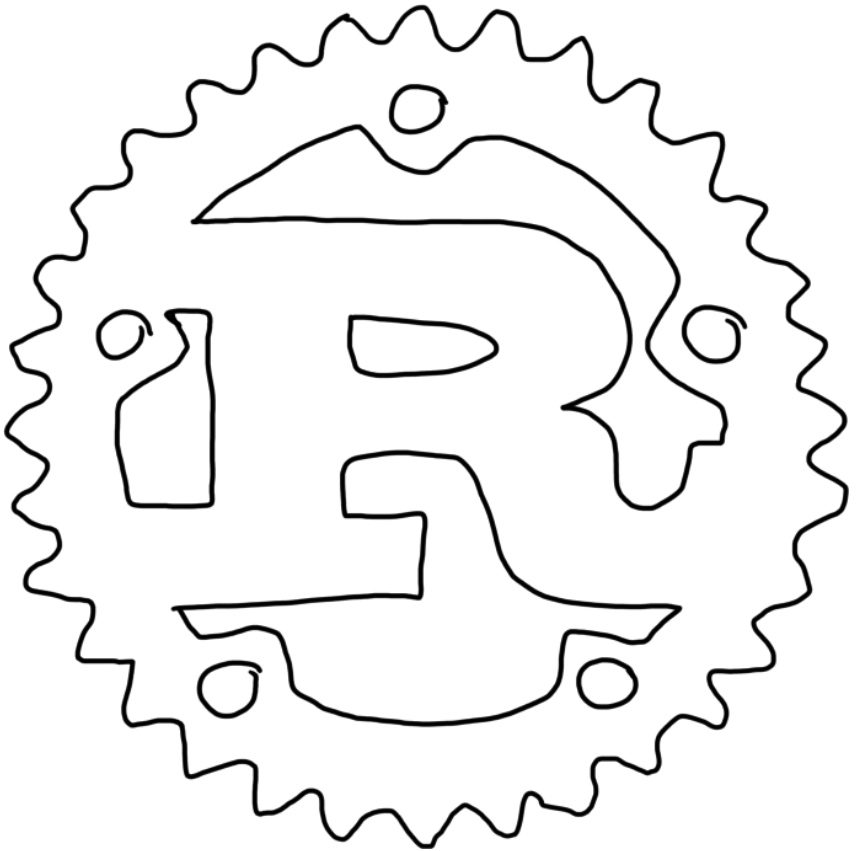


computational
brute force is
sometimes
a great answer

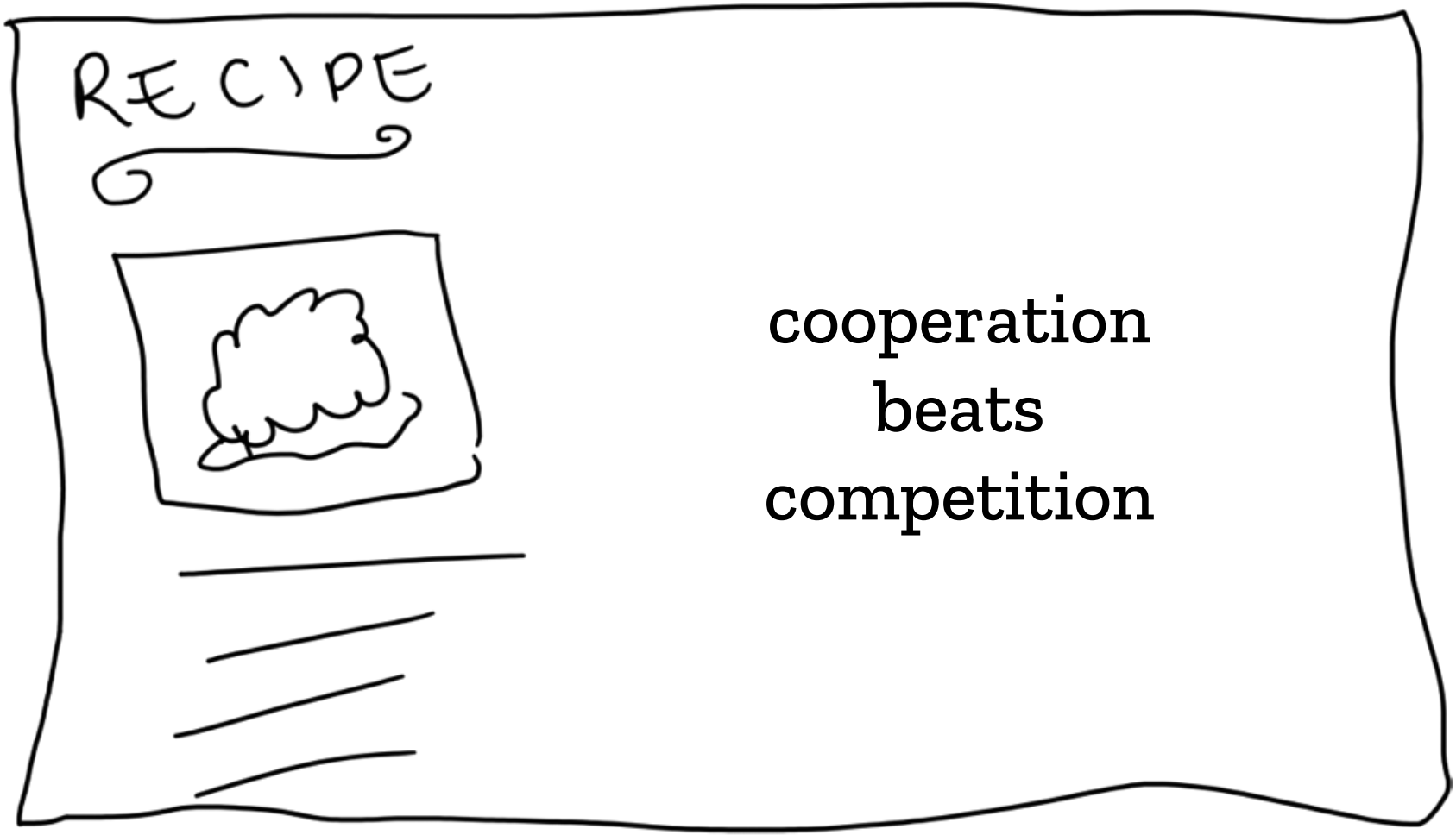




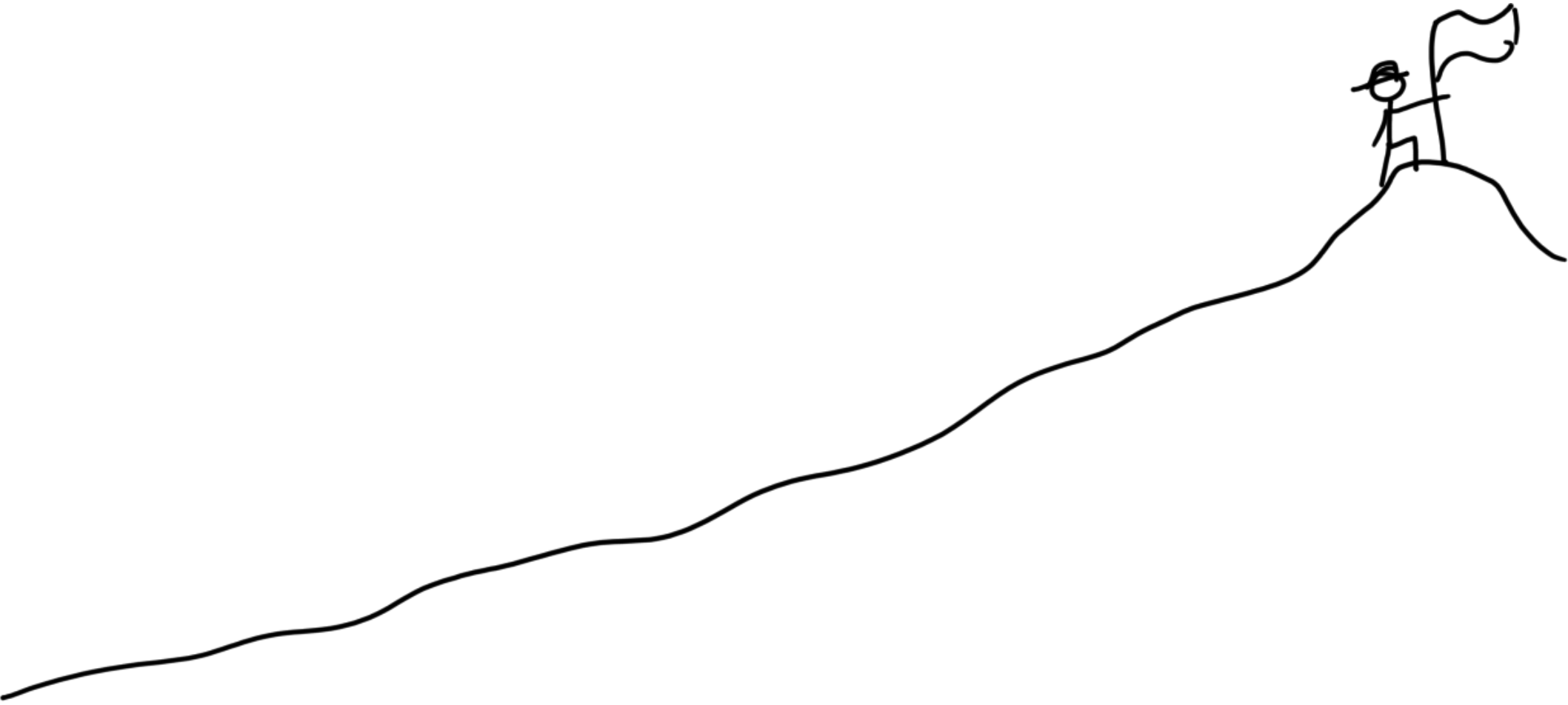
what examples
does your
community follow?

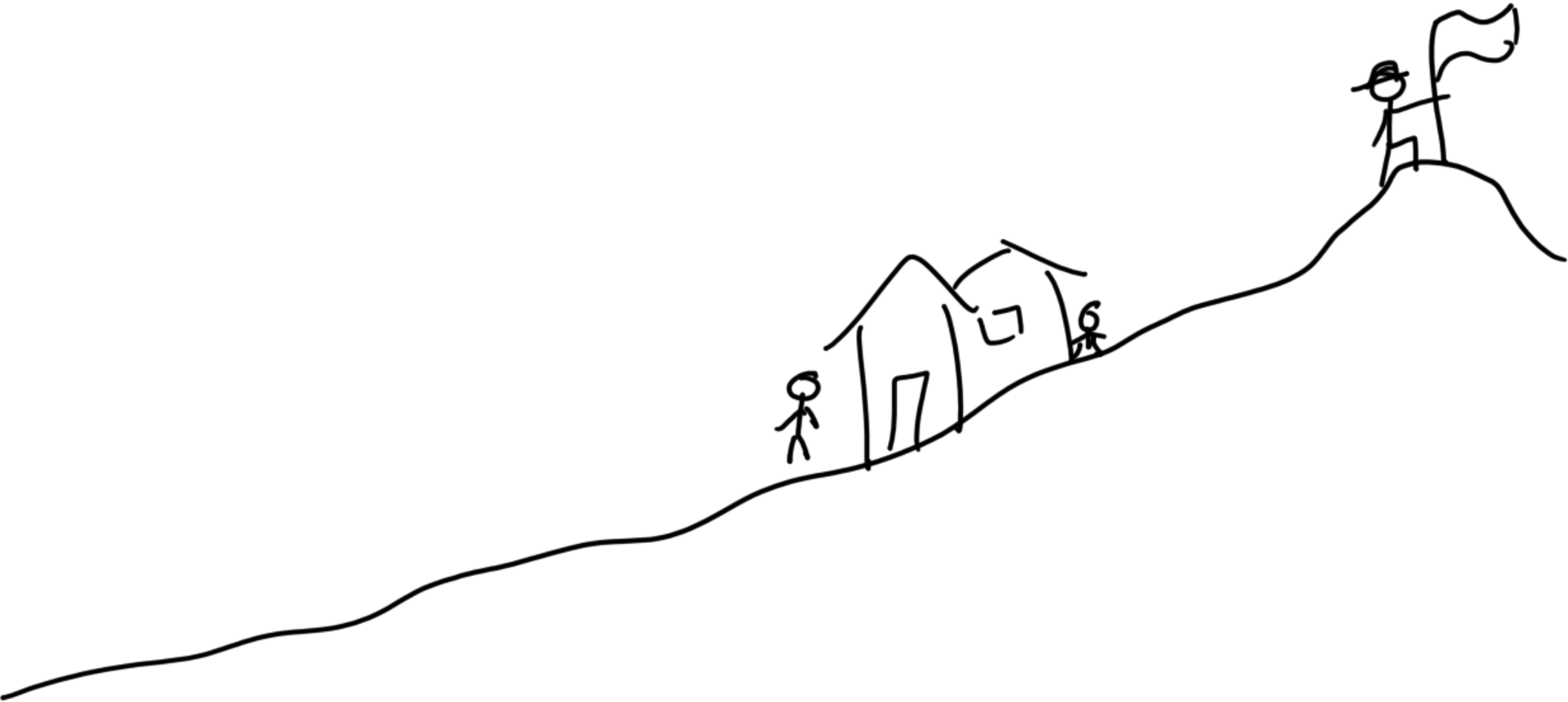


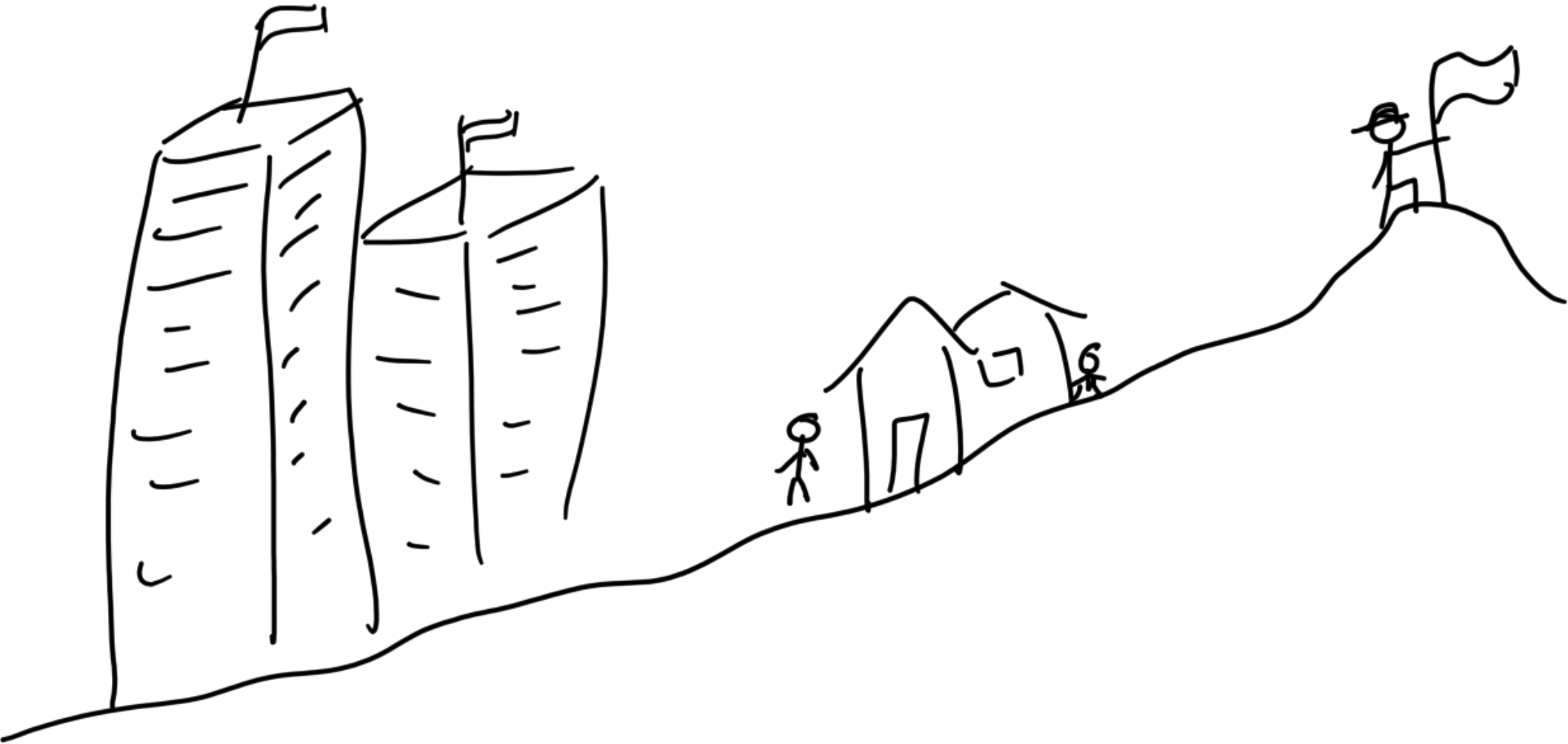

LLVM

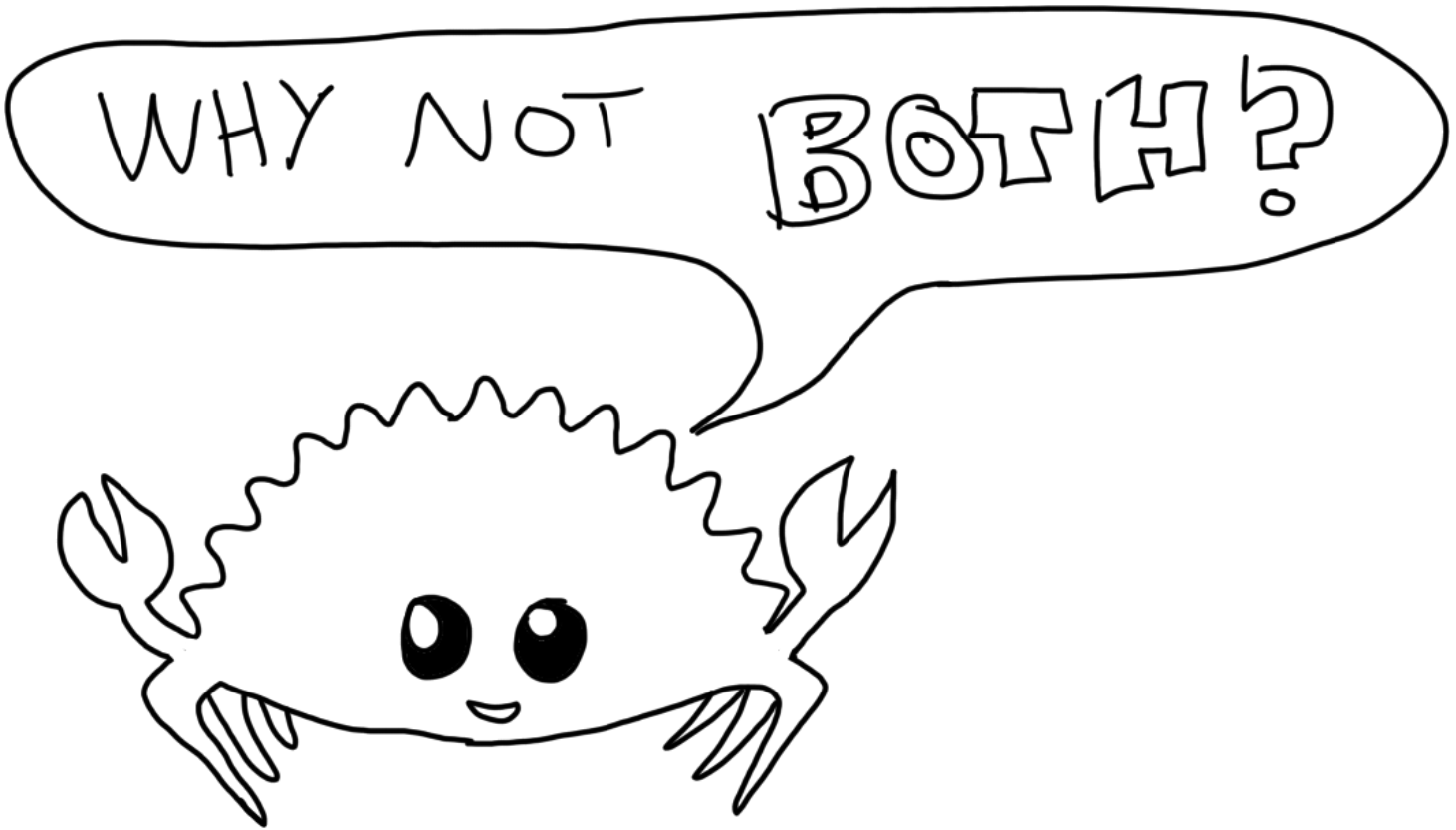


cooperation
beats
competition

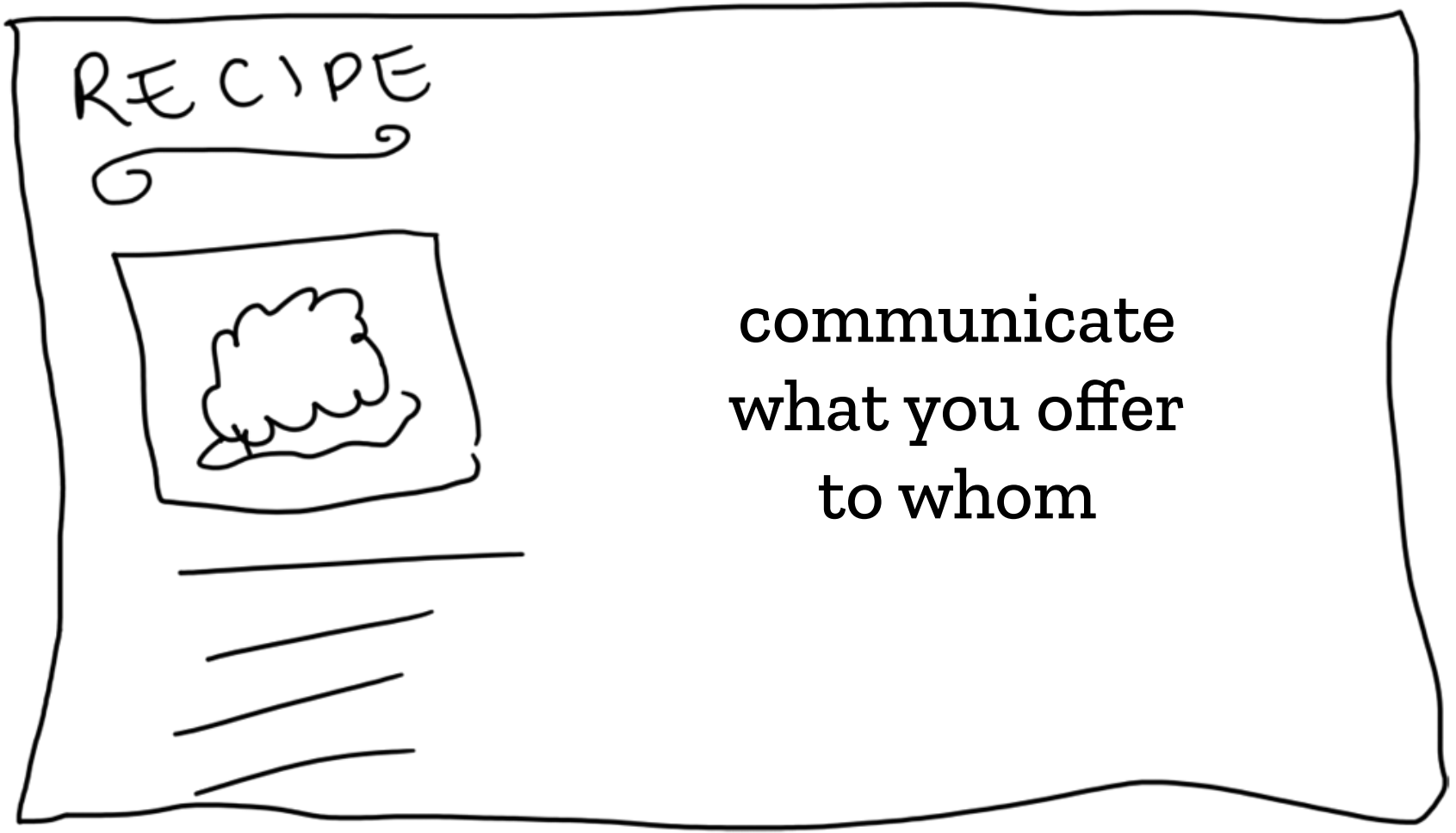




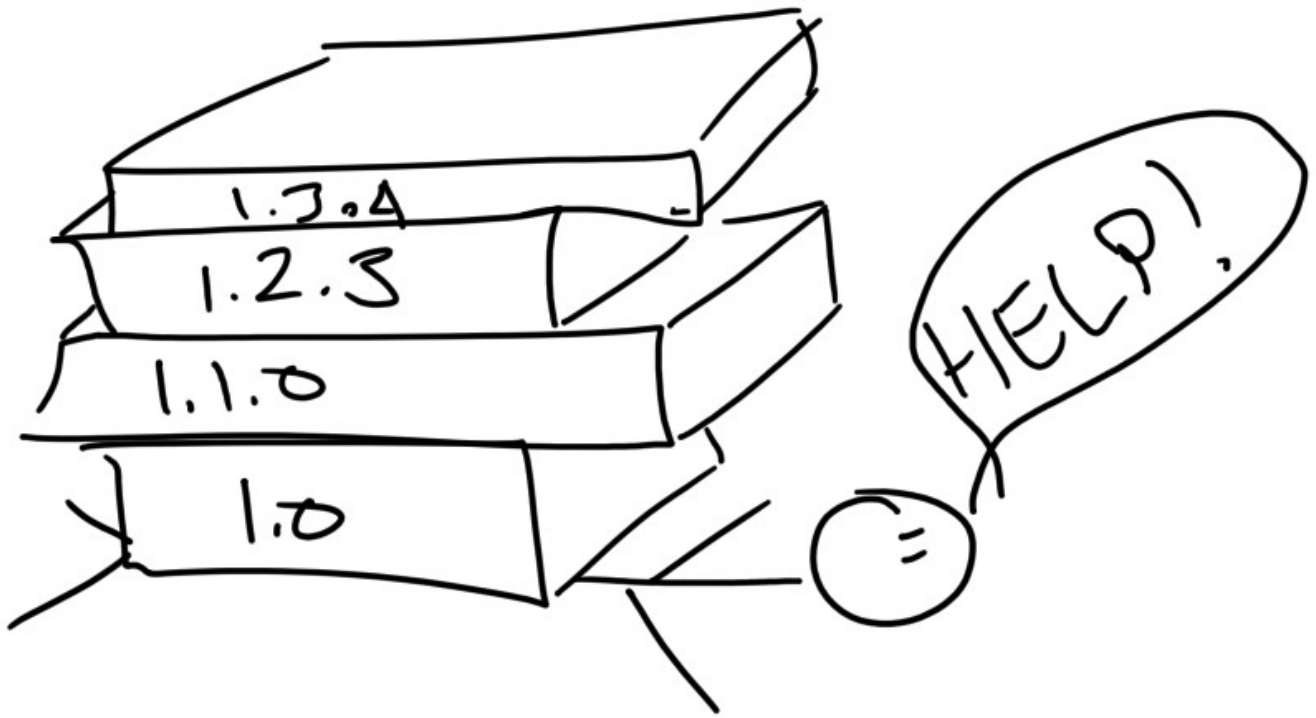



WHY NOT BOTH?

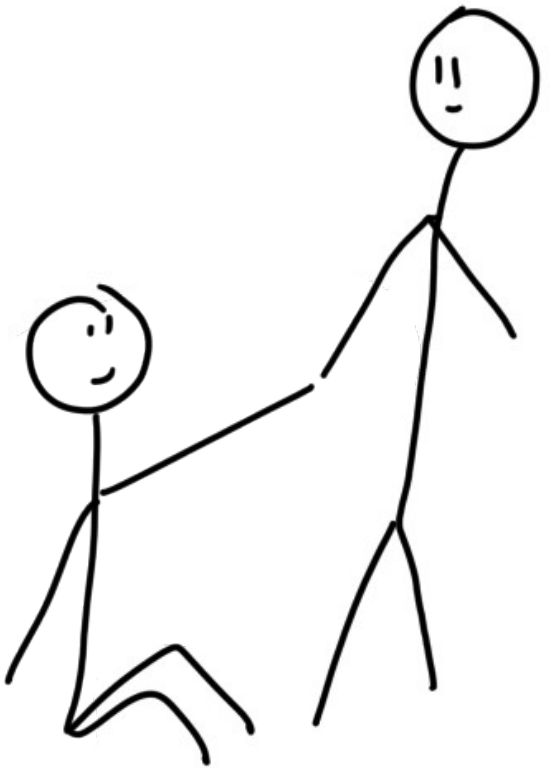

STABLE
BETA
NIGHTLY



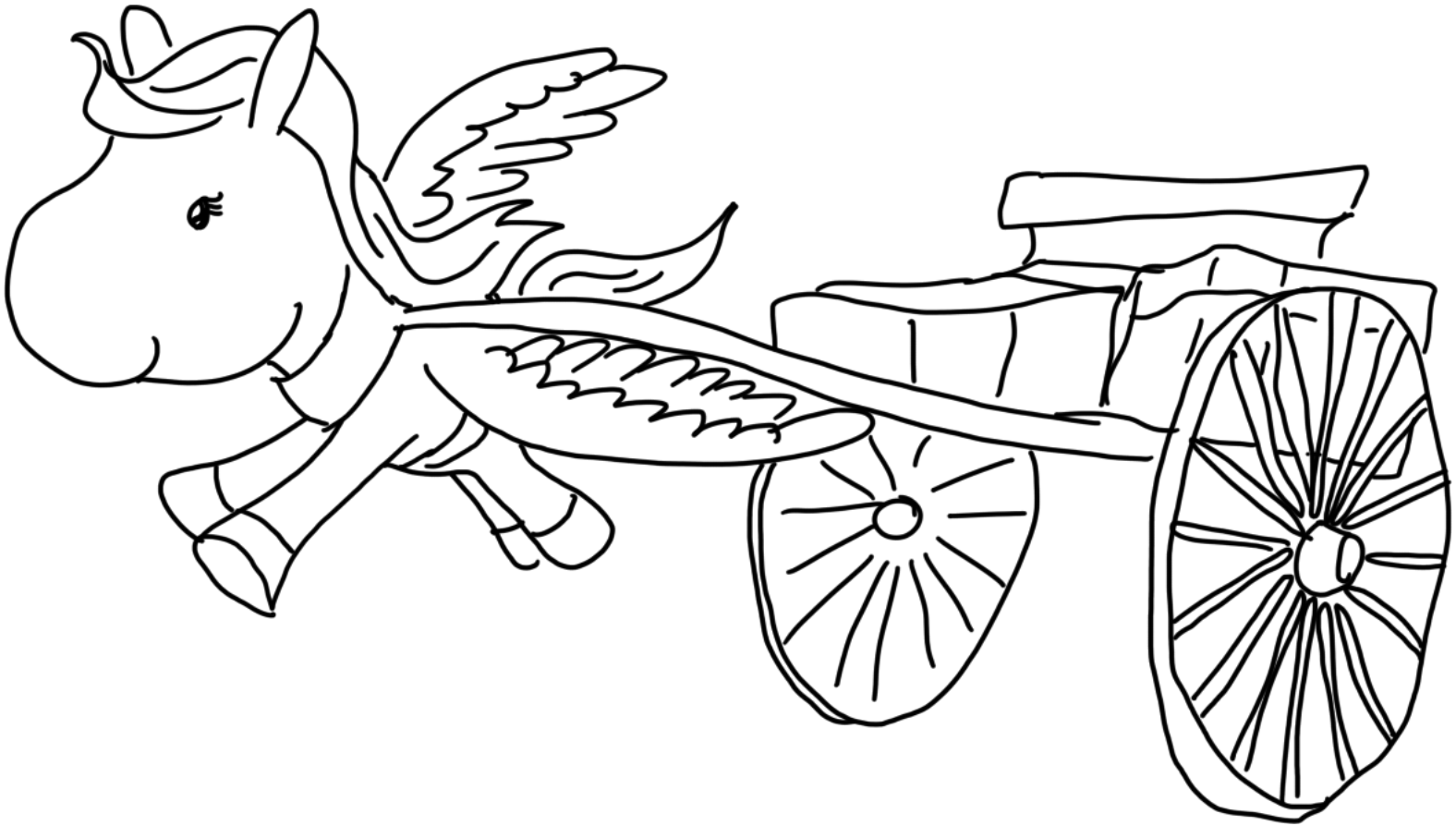
communicate
what you offer
to whom


1.3.4
1.2.3
1.1.0
1.0
HELP!

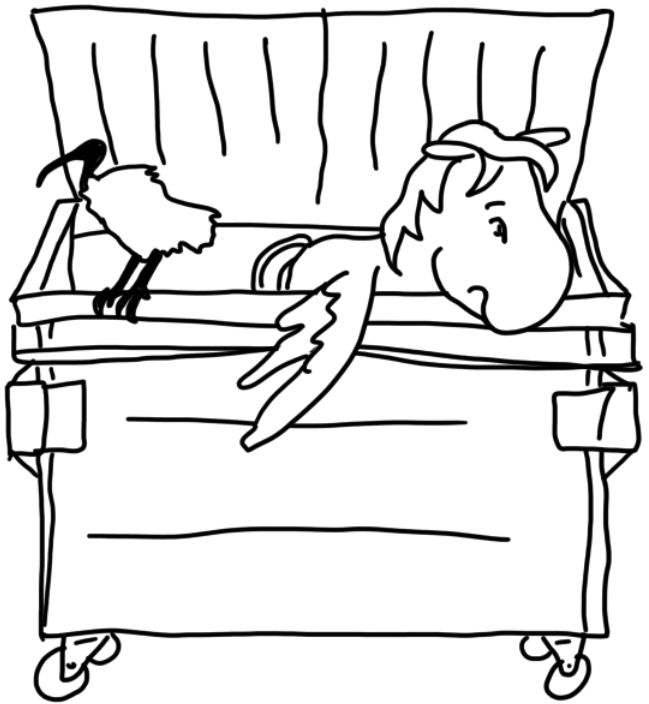

1.3.4
1.2.3
1.1.0
1.0

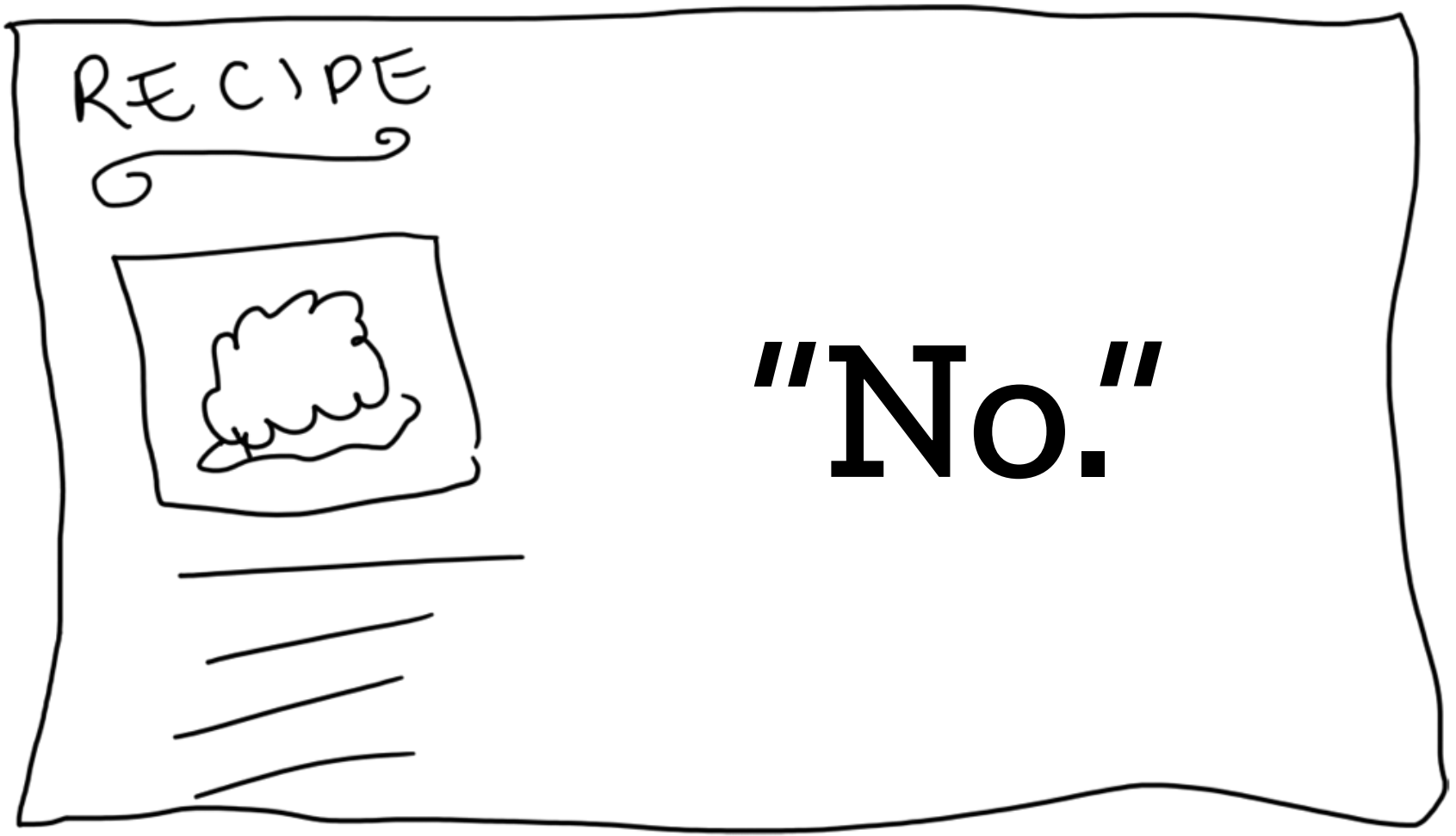


prepare to address
unforeseen
challenges

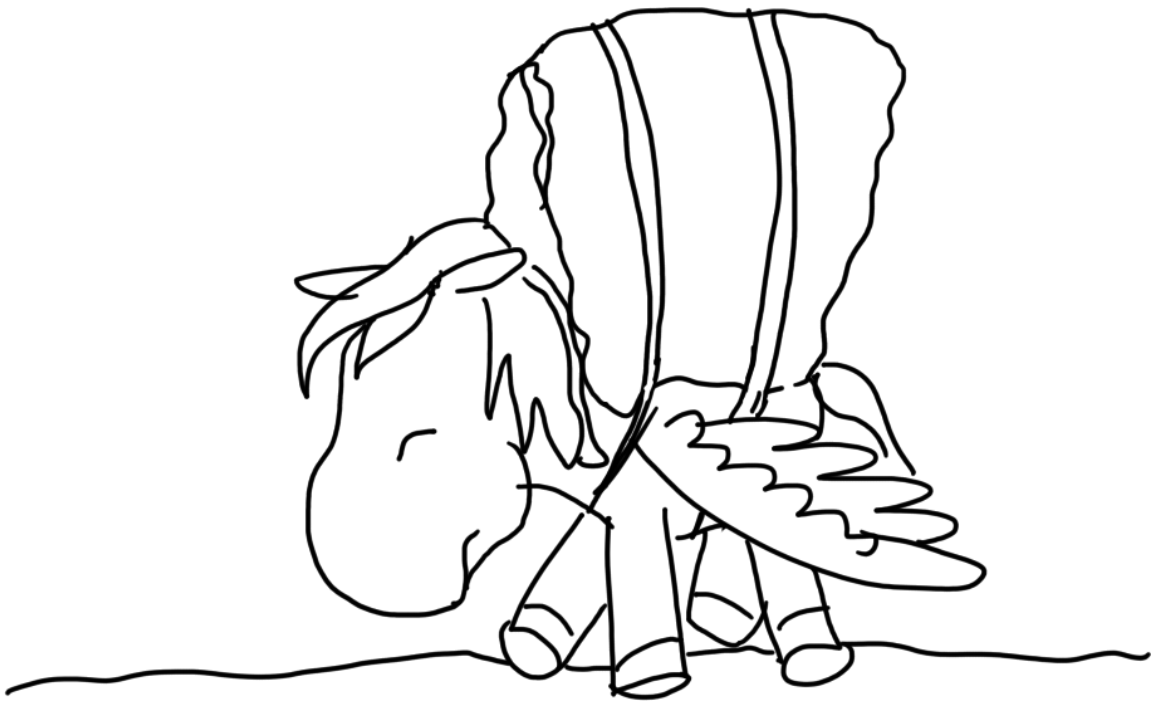


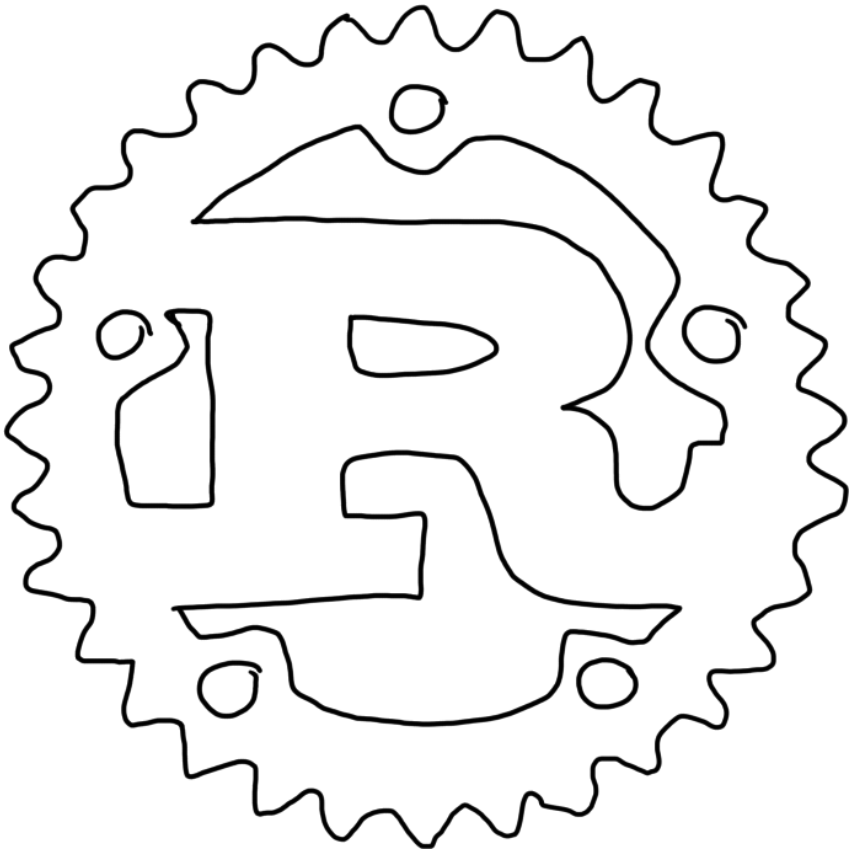
Rewrite it all?



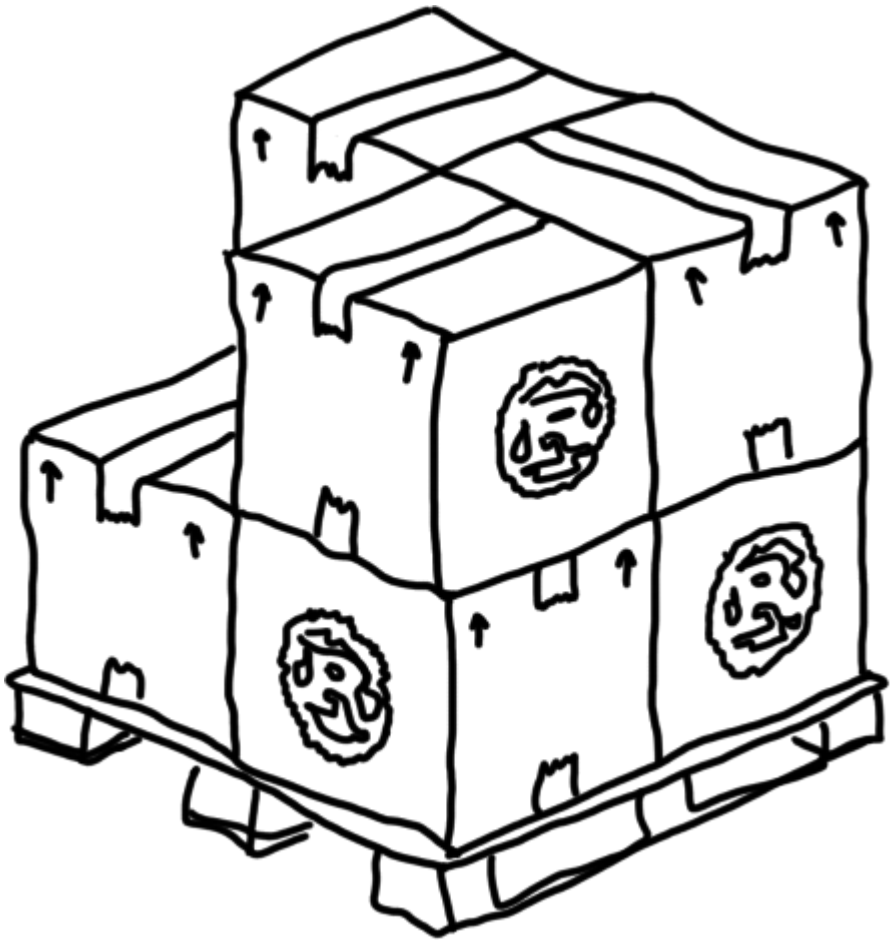


"No."

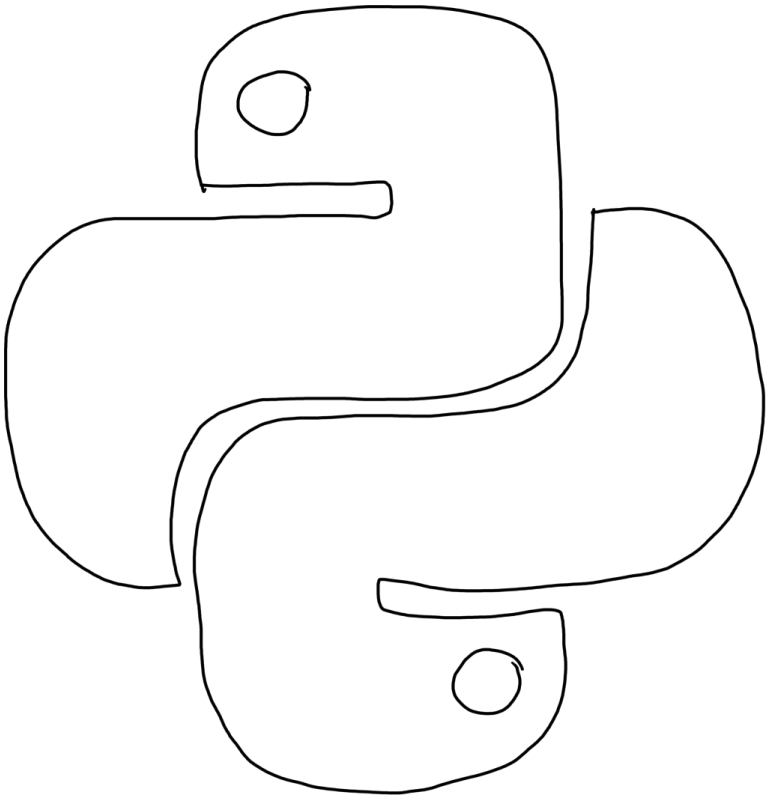




rustup.rs

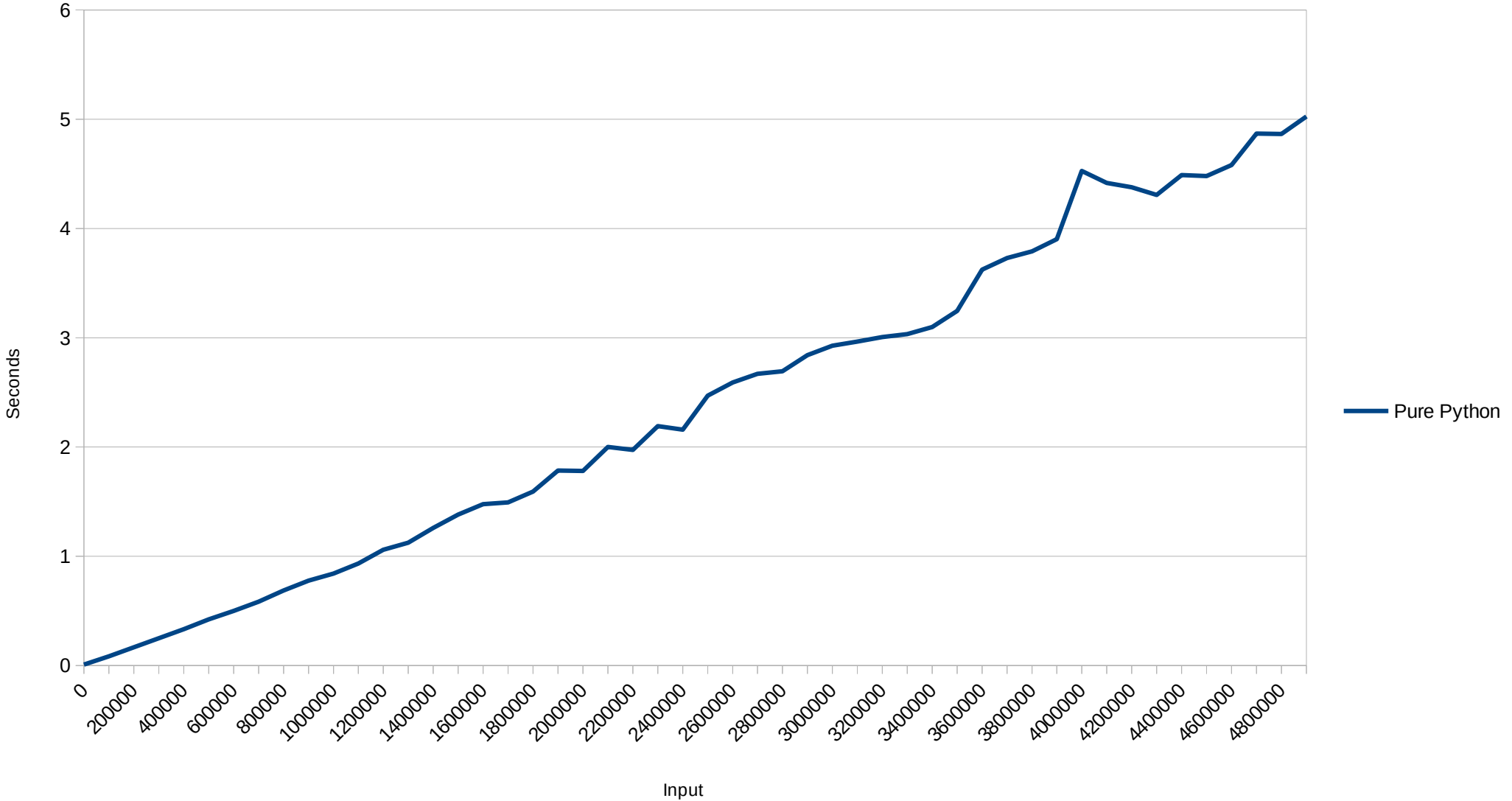


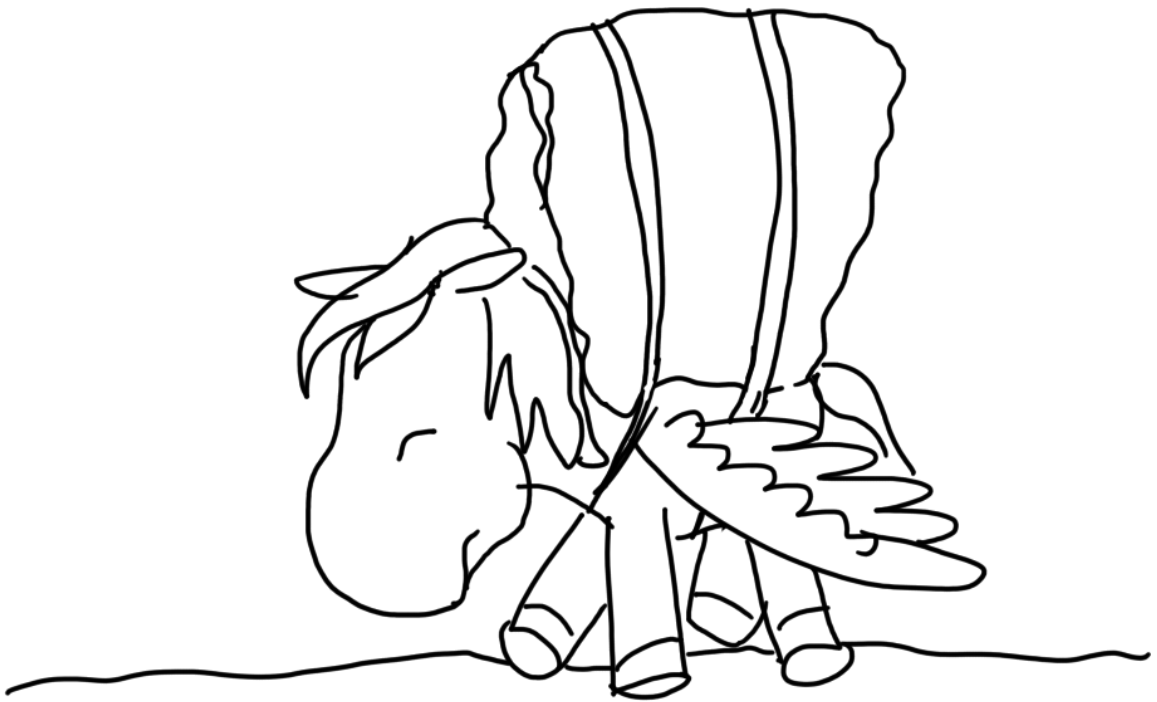
areweideyet.com



github.com/edunham/python-rust-ffi-timing

```
#!/usr/bin/env python3
from sys import argv
def sum_digits(sumthing):
    total = 0
    while sumthing > 0:
        total += sumthing % 10
        sumthing /= 10
    if total >= 10:
        total = sum_digits(total)
    return total

def sum_up(upto):
    total = 0
    for n in range(0, upto + 1):
        total = sum_digits(total + n)
    return total
lucky = sum_up(int(argv[1]))
print("your lucky digit calculated by python is {}".format(lucky))
```


Seconds
0
1
2
3
4
5
6
0
200000
400000
600000
800000
1000000
1200000
1400000
1600000
1800000
2000000
2200000
2400000
2600000
2800000
3000000
3200000
3400000
3600000
3800000
4000000
4200000
4400000
4600000
4800000
Input
Pure Python



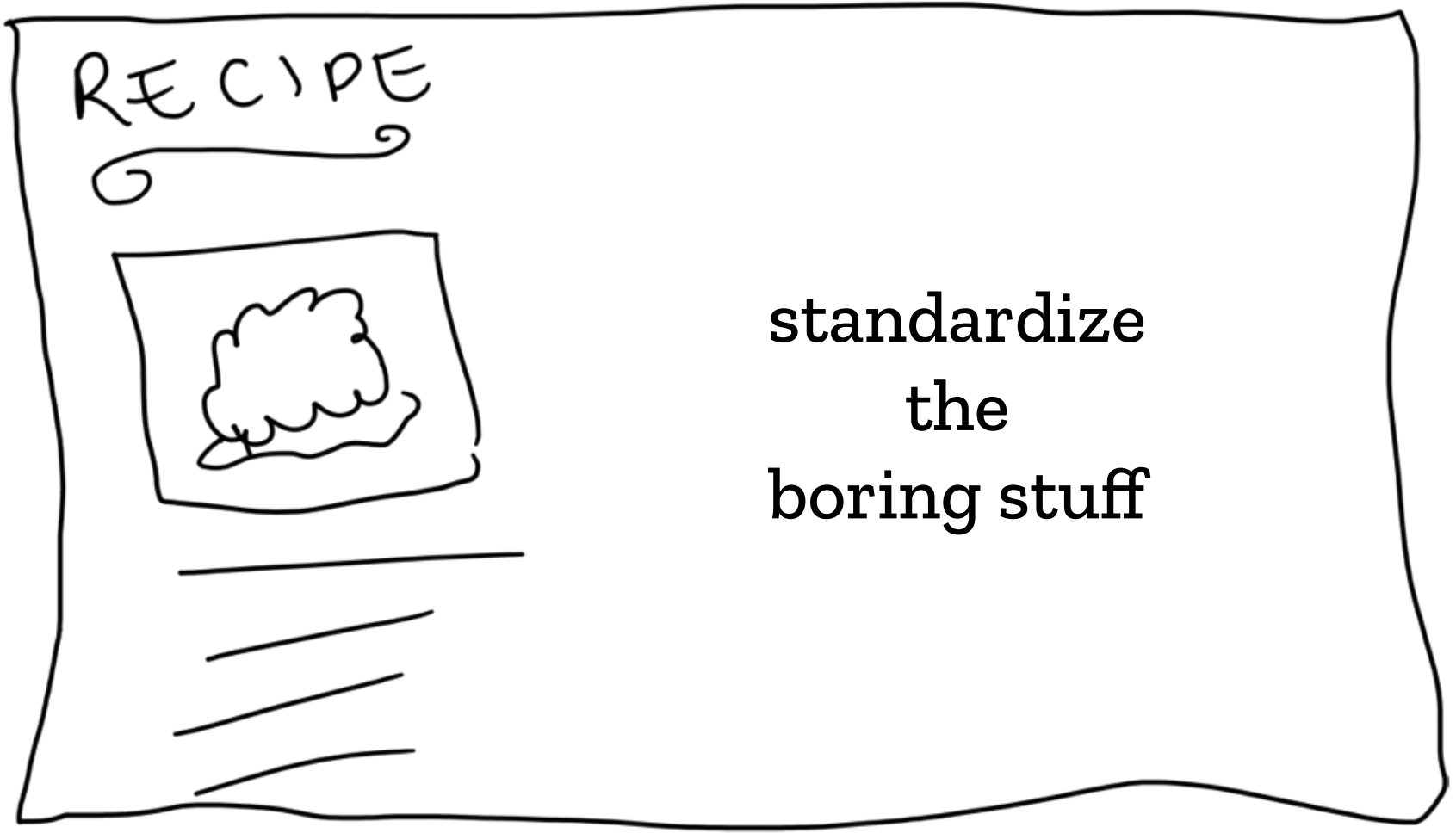
```python
def sum_digits(sumthing):
    total = 0
    while sumthing > 0:
        total += sumthing % 10
        sumthing /= 10

    if total >= 10:
        total = sum_digits(total)

    return total


def sum_up(upto):
    total = 0
    for n in range(0, upto + 1):
        total = sum_digits(total + n)

    return total
```

```rust
fn sum_digits(mut sumthing: u64) -> u64 {
    let mut total = 0;
    while sumthing > 0 {
        total += sumthing % 10;
        sumthing /= 10;
    }
    if total >= 10 {
        total = sum_digits(total)
    }
    return total;
}

fn sum_up(upto: u64) -> u64 {
    let mut total = 0;
    for n in 0..=upto {
        total = sum_digits(n + total);
    }
    return total;
}
```

```
fn sum_digits(mut sumthing: u64) -> u64 {
    let mut total = 0;
    while n > 0 {
        total += sumthing % 10;
        sumthing /= 10;
    }
    if total >= 10 {
        total = sum_digits(total)
    }
    return total;
}

fn sum_up(upto: u64) -> u64 {
    let mut total = 0;
    for n in 0..=upto {
        total = sum_digits(n + total);
    }
    return total;
}
```

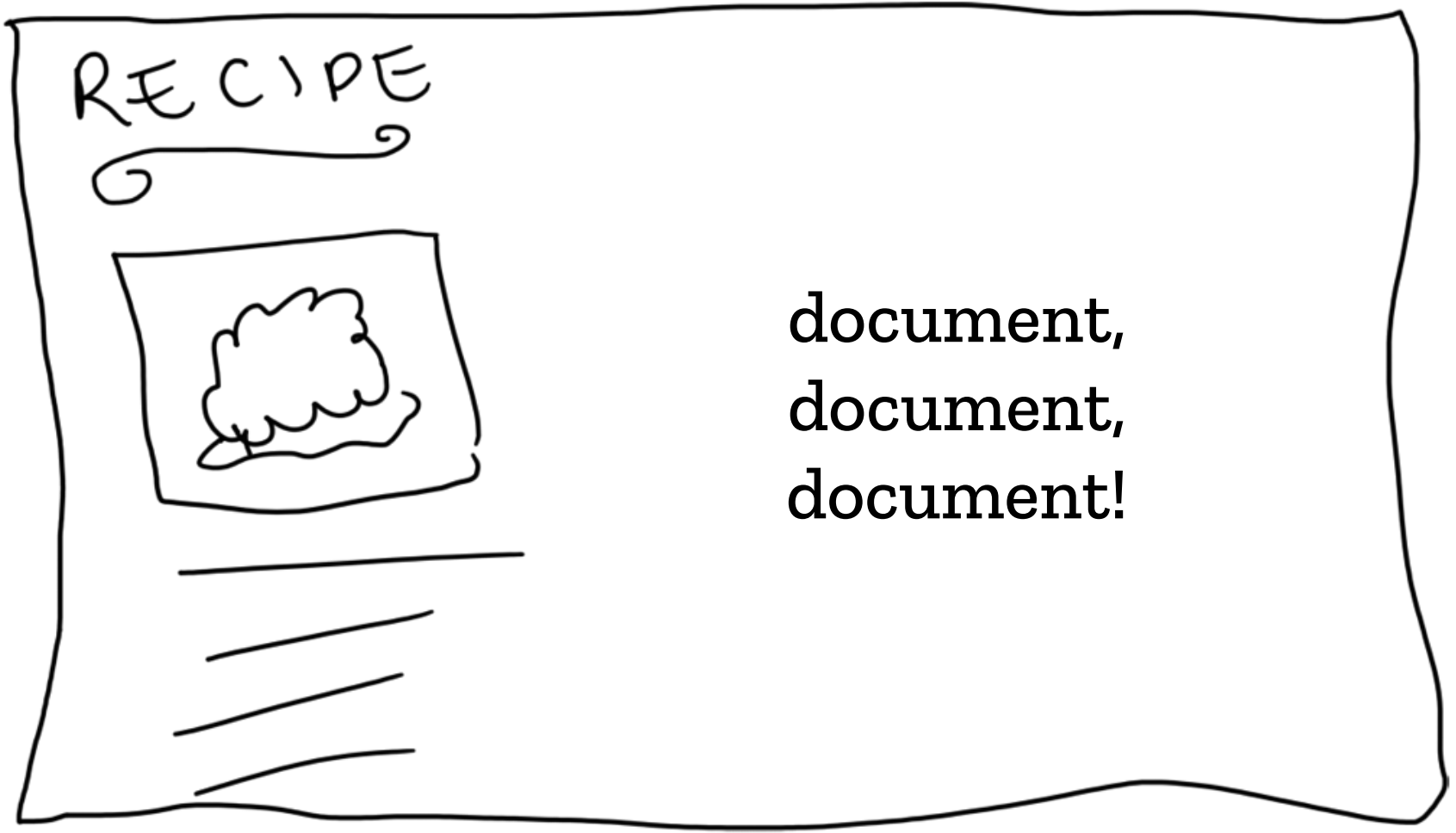
```
fn sum_digits(mut
sumthing:u64)->
u64{let mut
total=0;let
;while sumthing
>0{total
+=sumthing%10
;sumthing/=10;}if
total>=10 {total=
sum_digits(total
)}return total
;} fn sum_up (
upto:
u64)->u64{let
mut total=0
;for n in 0..=
upto{total
=sum_digits
(n+total);
}return total
;}
```

rustfmt



standardize
the
boring stuff


```
fn sum_digits(sumthing: u64) -> u64 {
    let mut total = 0;
    while sumthing > 0 {
        total += sumthing % 10;
        sumthing /= 10;
    }
    ...
```



document,
document,
document!

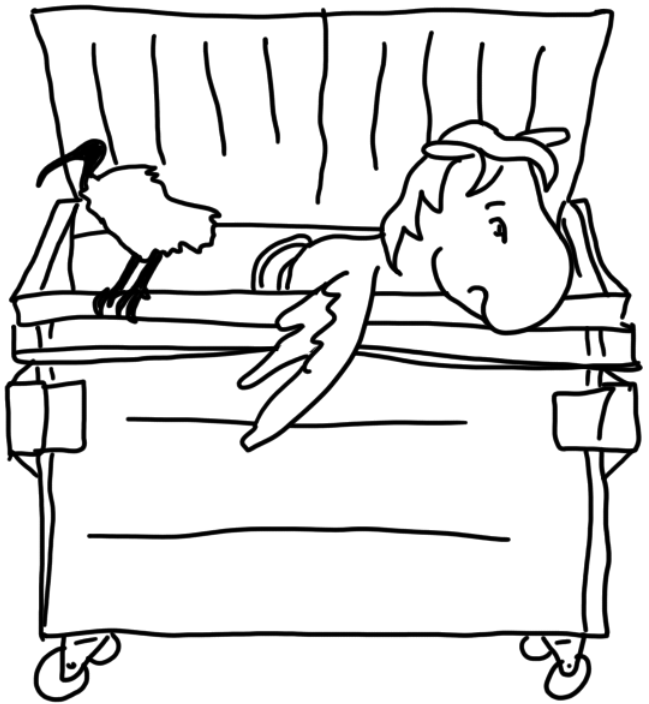

```
fn sum_digits(sumthing: u64) -> u64 {
    let mut total = 0;
    while sumthing > 0 {
        total += sumthing % 10;
        sumthing /= 10;
    }
    ...
```

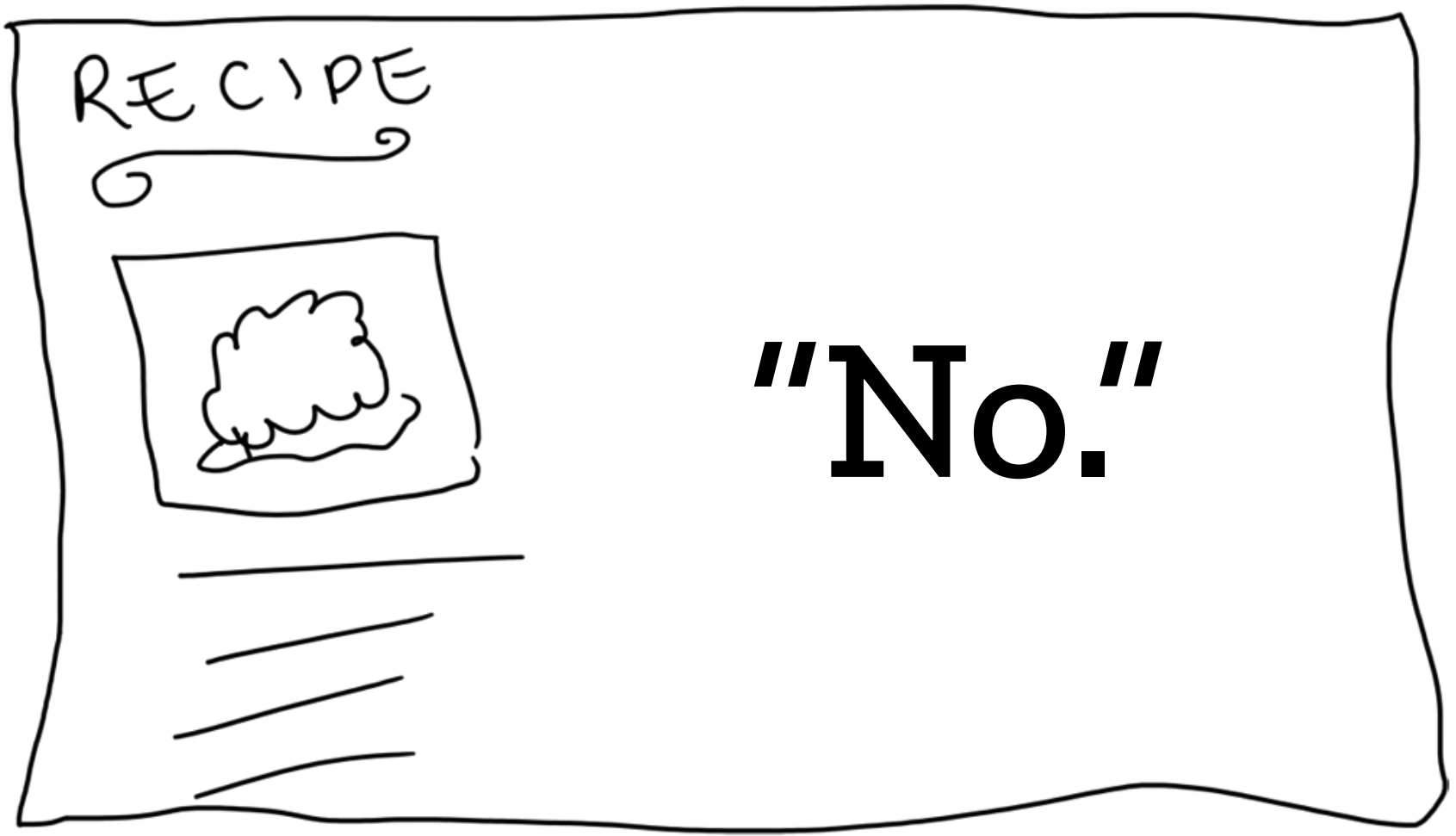
```
$ cargo build
   Compiling python-to-rust v0.2.0 (/home/edunham/repos/python-rust-ffi-timing)
error[E0384]: cannot assign to immutable argument `sumthing`
 --> src/lib.rs:5:9
  |
1 | fn sum_digits(sumthing: u64) -> u64 {
  |               -------- help: make this binding mutable: `mut sumthing`
...
5 |         sumthing /= 10;
  |         ^^^^^^^^^^^^^^ cannot assign to immutable argument

error: aborting due to previous error
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `python-to-rust`.
To learn more, run the command again with --verbose.
```

```python
def sum_digits(sumthing):
    total = 0
    while sumthing > 0:
        total += sumthing % 10
        sumthing /= 10

    if total >= 10:
        total = sum_digits(total)

    return total


def sum_up(upto):
    total = 0
    for n in range(0, upto + 1):
        total = sum_digits(total + n)

    return total
```

```rust
fn sum_digits(mut sumthing: u64) -> u64 {
    let mut total = 0;
    while sumthing > 0 {
        total += sumthing % 10;
        sumthing /= 10;
    }
    if total >= 10 {
        total = sum_digits(total)
    }
    return total;
}

fn sum_up(upto: u64) -> u64 {
    let mut total = 0;
    for n in 0..=upto {
        total = sum_digits(n + total);
    }
    return total;
}
```

Rewrite it all?





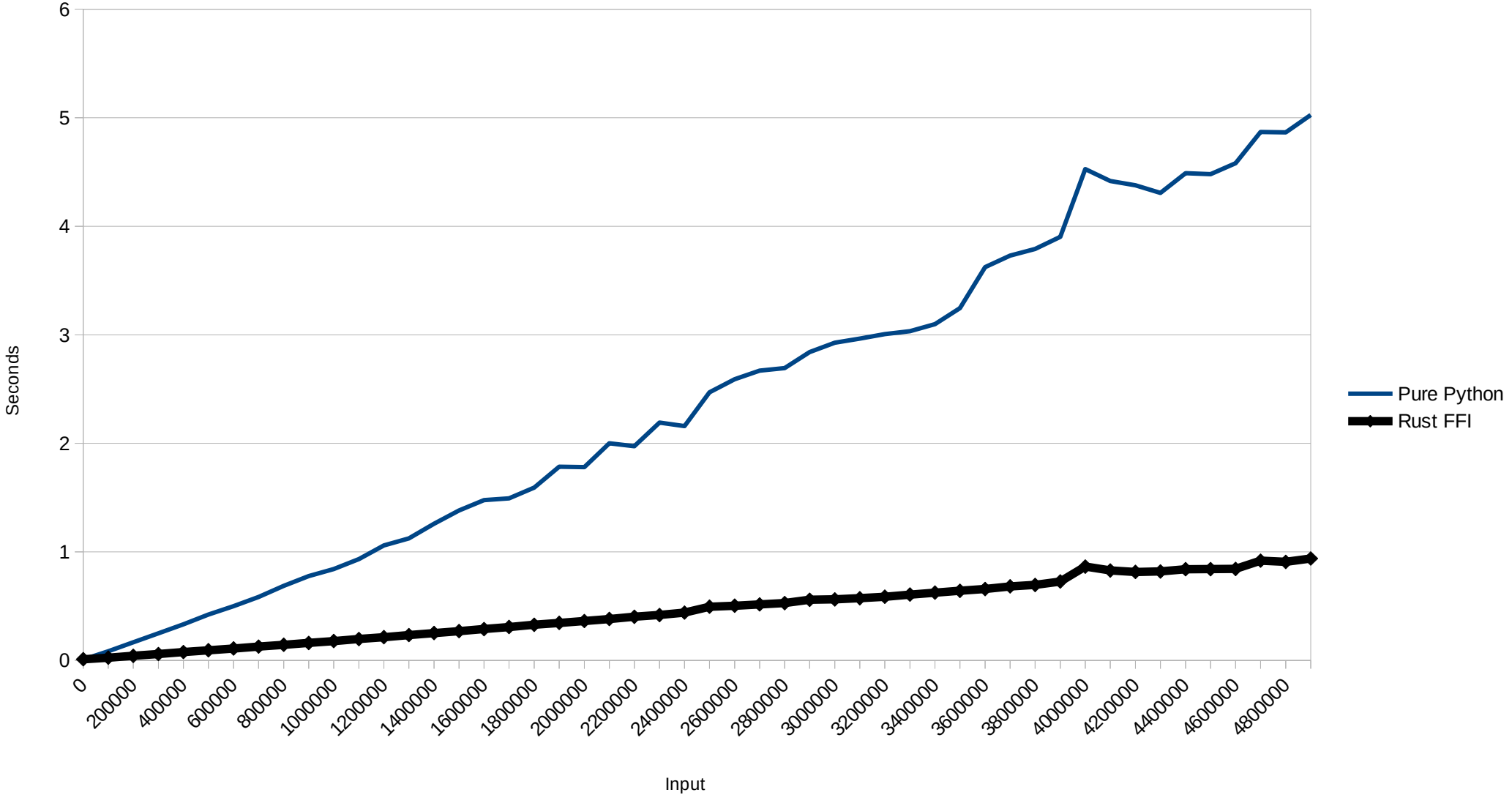
"No."

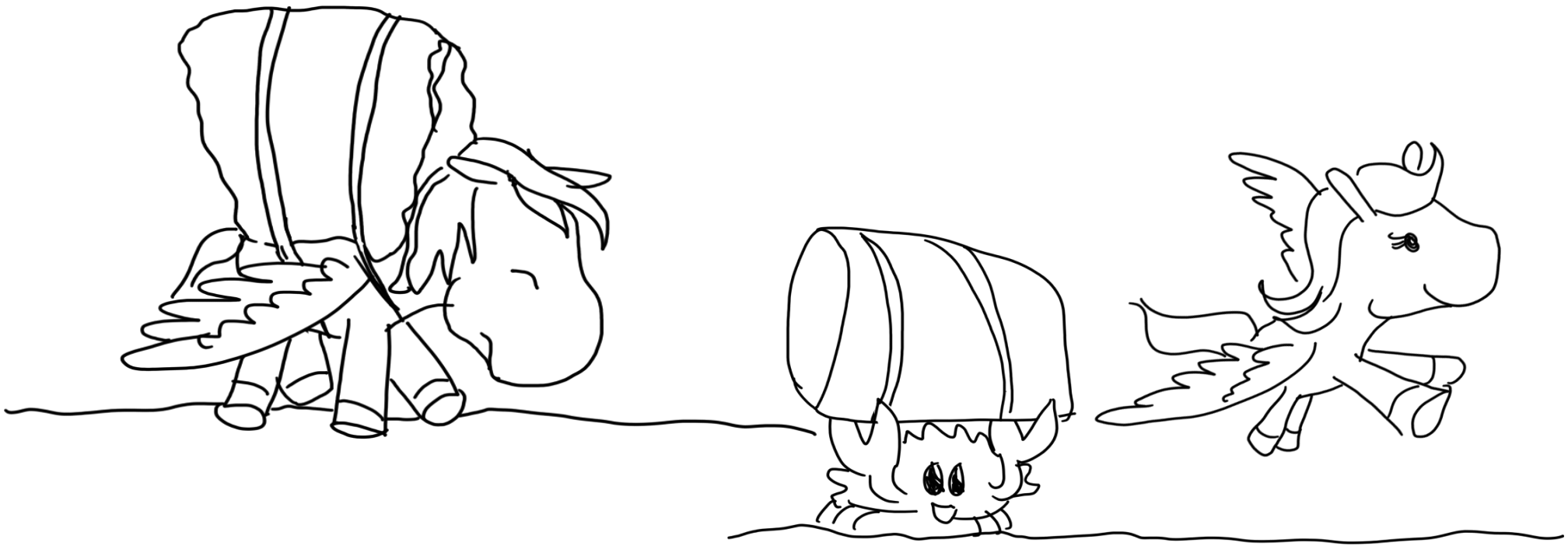
```
#!/usr/bin/python3
from ctypes import cdll
from sys import platform, argv

if platform == 'darwin':
    prefix = 'lib'
    ext = 'dylib'
elif platform == 'win32':
    prefix = ''
    ext = 'dll'
else:
    prefix = 'lib'
    ext = 'so'

lib = cdll.LoadLibrary('target/debug/{}sum_up.{}'.format(prefix, ext))
sum_up = lib.sum_up

sum_to = int(argv[1])
output = sum_up(sum_to)
print('your lucky digit calculated by rust is {}'.format(output))
```

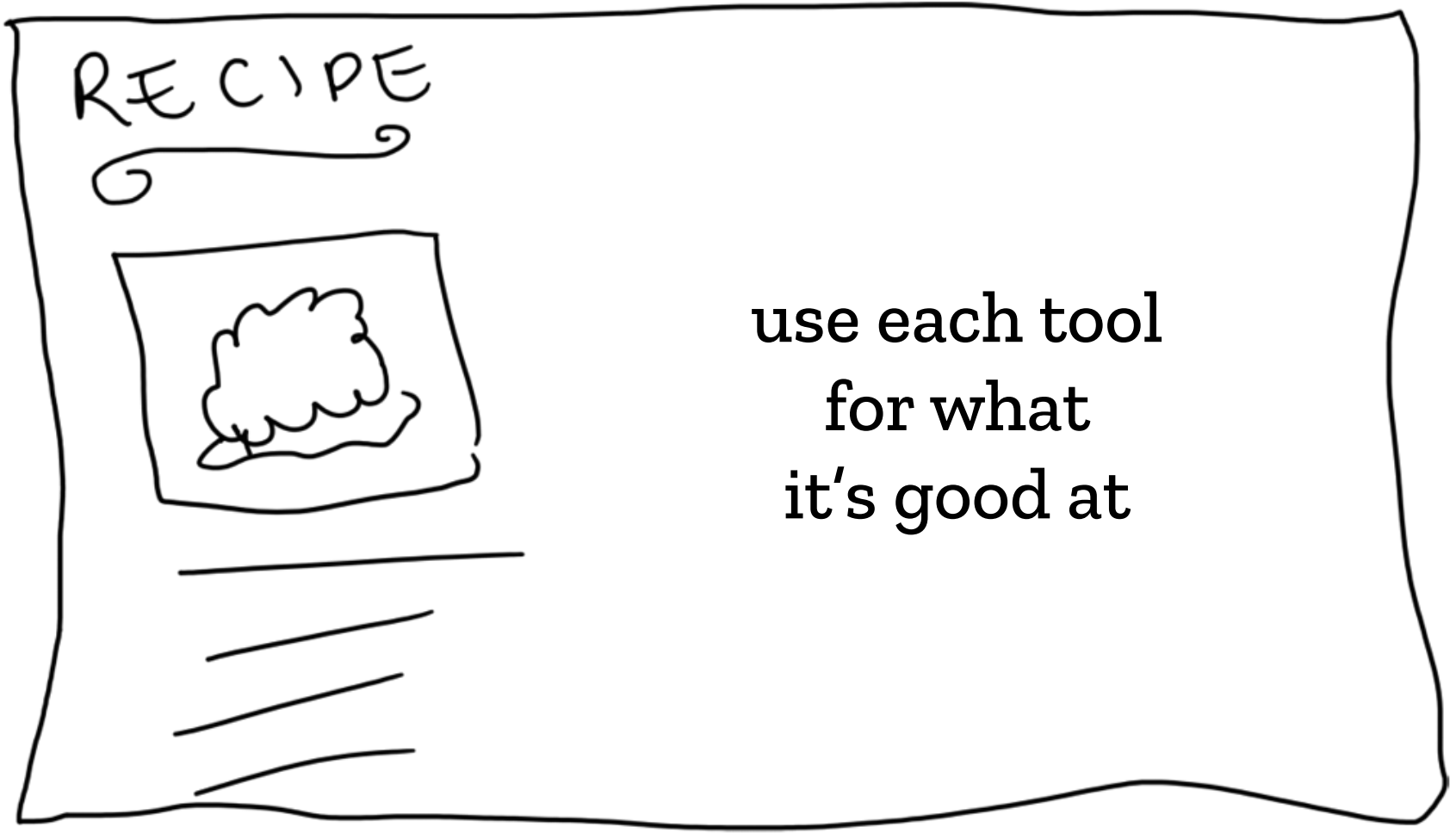
```
$ cat src/lib.rs

fn sum_digits(mut sumthing: u64) -> u64 {
    let mut total = 0;
    while sumthing > 0 {
        total += sumthing % 10;
        sumthing /= 10;
    }
    if total >= 10 {
        total = sum_digits(total)
    }
    return total;
}
#[no_mangle]
pub extern "C" fn sum_up(upto: u64) -> u64 {
    let mut total = 0;
    for n in 0..=upto {
        total = sum_digits(n + total);
    }
    return total;
}
```

```
$ cat Cargo.toml
[package]
name = "lucky-digit"
version = "0.2.0"
Authors = ["…”]
edition = "2018"

[lib]
name = "sum_up"
crate-type = ["dylib"]

$ cargo build
   Compiling lucky-digit v0.2.0 (/home/edunham/repos/python-rust-ffi-timing)
   Finished dev [unoptimized + debuginfo] target(s) in 0.26s
```

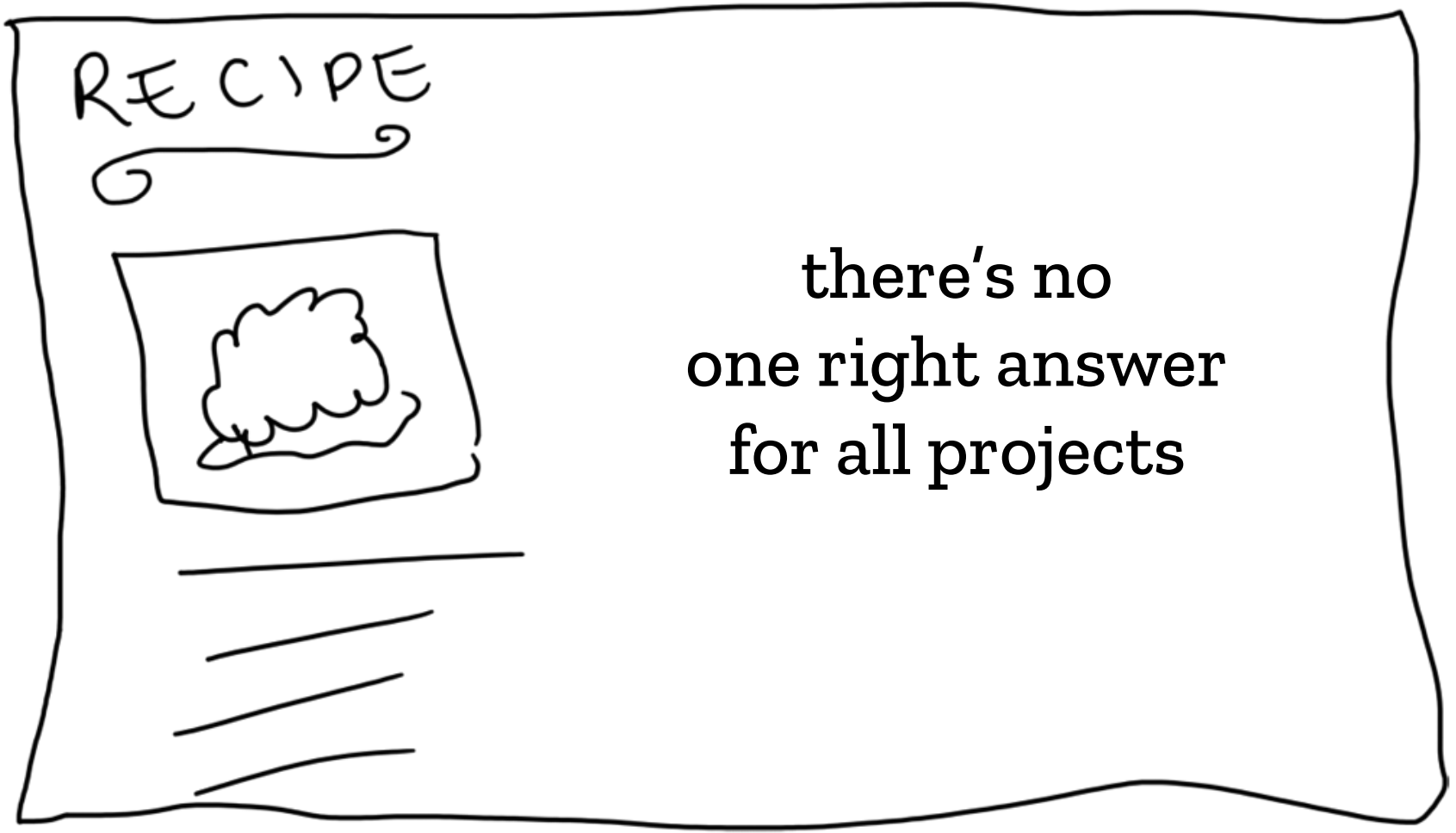
```
$ cat timer.py

import os
import timeit

outfile = "data.csv"
start = 0
datapoints = 50
step = 100000
for i in range(start,datapoints*step,step):
    print i
    pre = timeit.default_timer()
    os.system("python sum.py {}".format(i))
    mid = timeit.default_timer()
    os.system("python src/main.py {}".format(i))
    post = timeit.default_timer()
    with open(outfile, "a") as f:
        f.write("{}, {}, {}\n".format(i, mid - pre, post - mid))
```


6
5
4
3
2
1
0
Seconds
0
200000
400000
600000
800000
1000000
1200000
1400000
1600000
1800000
2000000
2200000
2400000
2600000
2800000
3000000
3200000
3400000
3600000
3800000
4000000
4200000
4400000
4600000
4800000
Input
Pure Python
Rust FFI





use each tool
for what
it's good at



there's no
one right answer
for all projects