Making the Impossible

Impossible

Improving Reliability by Preventing Classes of Problems

@ChrisSinjo

Hi

~~Hi~~

Greetings

@ChrisSinjo

@ChrisSinjo

Infra Engineer

PlanetScale

Making the Impossible

Impossible

Improving Reliability by
Preventing Classes
of Problems

@ChrisSinjo

We are at

SREcon

We likely share:

- Job titles
- Skills
- Ways of thinking

Common ground/
"Best practices"

Some ideas have

outsized

impact

In SRE: SLOs

(Service Level Objectives)

A refresher:
Measuring the performance of a service as a percentage of successful operations

Example: HTTP requests

$$\frac{\text{Successful requests}}{\text{Total requests}} \times 100$$

$\geq 99.9\%$

So why am I here today?

O'REILLY

Site Reliability Engineering

HOW GOOGLE RUNS PRODUCTION SYSTEMS

Edited by Betsy Beyer, Chris Jones, Jennifer Petoff & Niall Richard Murphy

The perils of success

The way we measure

shapes

The way we think

The way we think

shapes

The solutions we explore

SLOs encourage **percentage** thinking

Instances go unhealthy

Instances go unhealthy

↓

Add health checks &

route traffic away

Regional network issues

Regional network issues

↓

Serve from multiple regions

Rare slow requests

Rare slow requests

⬇

Add timeouts to protect majority of traffic

Example: HTTP requests

$$\frac{\text{Successful requests}}{\text{Total requests}} \times 100 \geq 99.9\%$$

Reliability is a **percentage** game

We can
stack the odds
in our favour

Not all solutions look the **same**

Not all solutions are about percentages

Some solutions prevent problems entirely

Today's talk:

- Another lens for reliability

Today's talk:

- Another lens for reliability
- Examples in the wild

Today's talk:

- Another lens for reliability
- Examples in the wild
- How to spot problems of this shape

This is not:

- An attack on SLOs

This is not:

- An attack on SLOs
- One-size-fits all solution

This is not:

- An attack on SLOs
- One-size-fits all solution
- Possible if you can't edit software

Examples:

- State machines

Examples:

- State machines
- Memory safety

Examples:

- State machines
- Memory safety
- Database migrations

Example 1

State machines


Secretary of the Treasury
Treasurer of the United States.
100
$10,000
$10,000
$10,000

Collect from customer

Collect from customer

↓

Pay out to merchant

Payment

💸

Payment

Created

Submitted

Collected

Paid out

Failed

Simple model

id	description	state
1	Laptop	submitted
2	Phone	collected
3	Unused domain renewal	collected

Simple model

id	description	state
1	Laptop	submitted
2	Phone	collected
3	Unused domain renewal	collected

Simple model

id	description	state
1	Laptop	collected
2	Phone	collected
3	Unused domain renewal	collected

Simple model

id	description	state
1	Laptop	paid_out
2	Phone	collected
3	Unused domain renewal	collected

Simple model

id	description	state
1	Laptop	submitted
2	Phone	collected
3	Unused domain renewal	collected

Simple model

id	description	state
1	Laptop	failed
2	Phone	collected
3	Unused domain renewal	collected

Submitted ➡ Failed

Submitted ➡ Failed

Collected ➡ Failed?

Submitted ➡ Failed

Paid out ➡ Failed?

We want some

restrictions

State restriction pseudocode

```
class Payment
  def fail()
    state = "failed"
```

State restriction pseudocode

```
class Payment
  def fail()
    if state == "submitted"
      state = "failed"
    else
      raise "Cannot fail from state: #{state}"
```

State restriction pseudocode

```
class Payment
  def submit()
    if state == "created"
      state = "submitted"
    else
      raise "Cannot submit from state: #{state}"
```

Payment

Created
Submitted
Collected
Paid out
Failed

Payment

Created
Submitted
Collected
Payout submitted
Paid out
Failed

State restriction pseudocode

```
class Payment
  def fail()
    if state in ["submitted", "payout_submitted"]
      state = "failed"
    else
      raise "Cannot fail from state: #{state}"
```

An

ad-hoc

mess

Bugs 📈

Maintenance 📈

Computer Science has an answer

We can use a

state machine

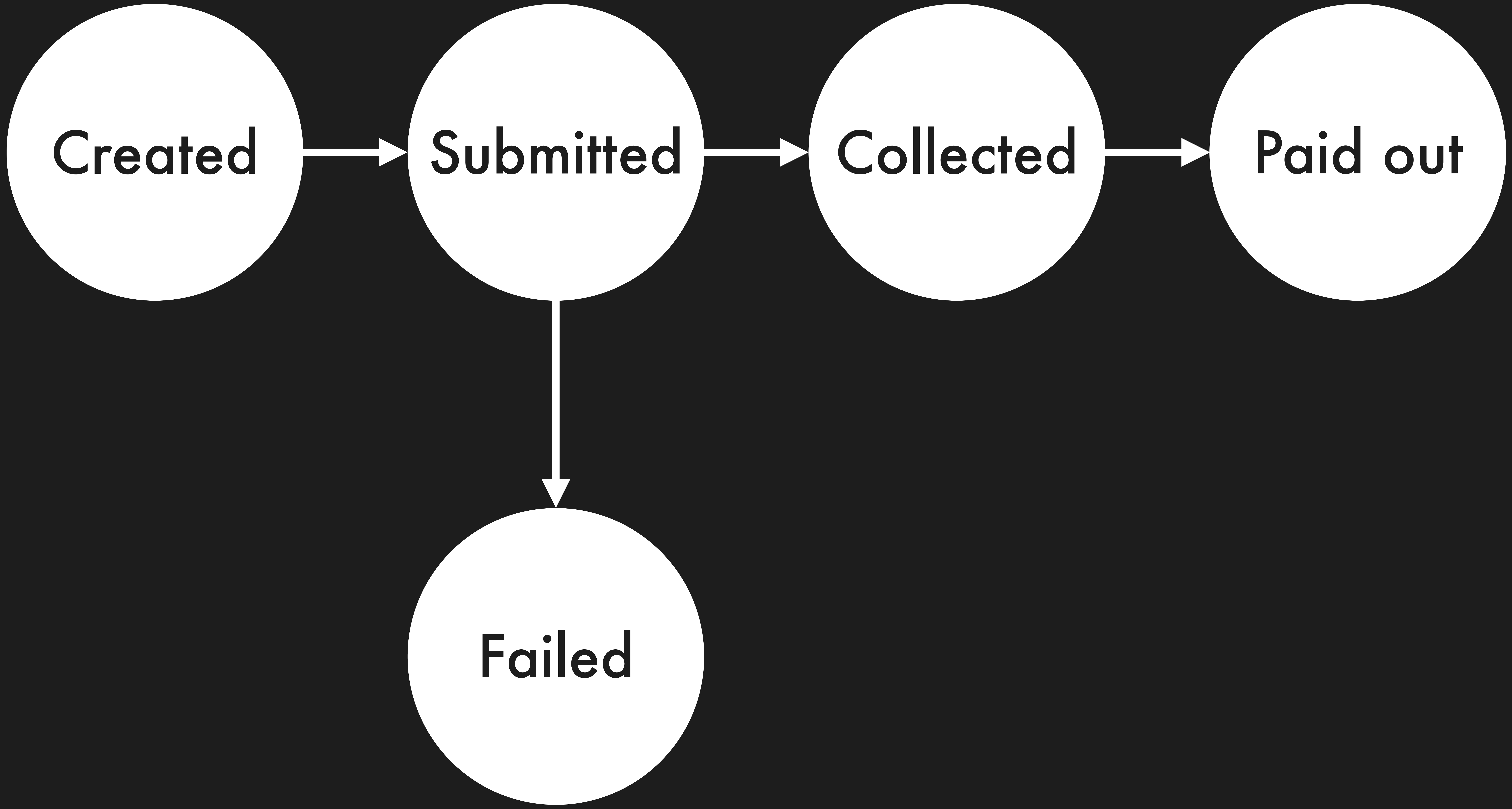
State machine:

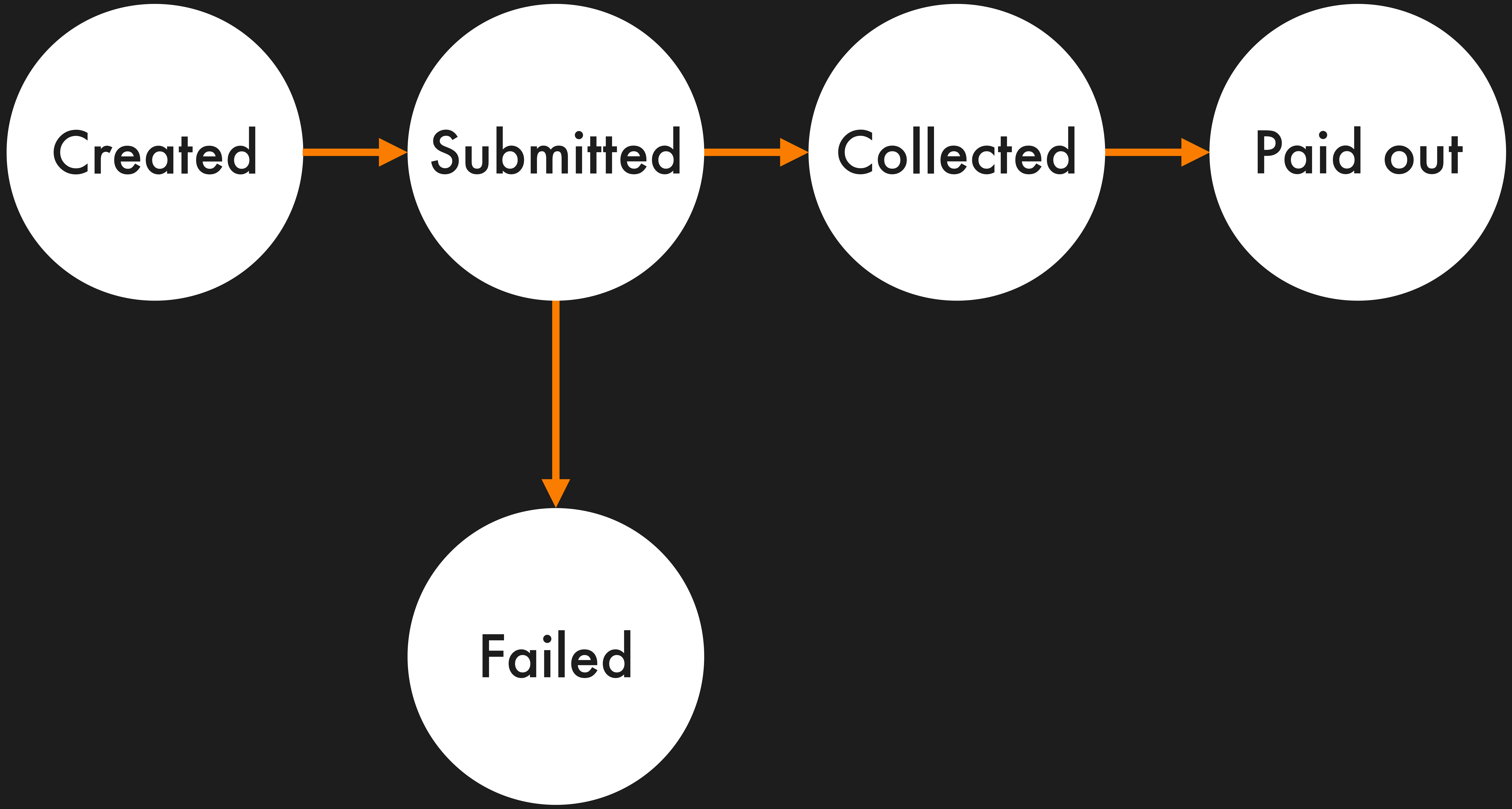
- A set of states
- A set of allowed transitions between those states

State machine pseudocode

```
class Payment
  states(["created", "submitted", ...])

  allow_transition("created", "submitted")
  allow_transition("submitted", "collected")
  allow_transition("submitted", "failed")
...
```


Created
Submitted
Collected
Paid out
Failed


Created
Submitted
Collected
Paid out
Failed

State machine pseudocode

```
class Payment
  states(["created", "submitted", ...])

  allow_transition("created", "submitted")
  allow_transition("submitted", "collected")
  allow_transition("submitted", "failed")
...
```

Error: cannot transition from "paid out" to "failed"

State machine pseudocode

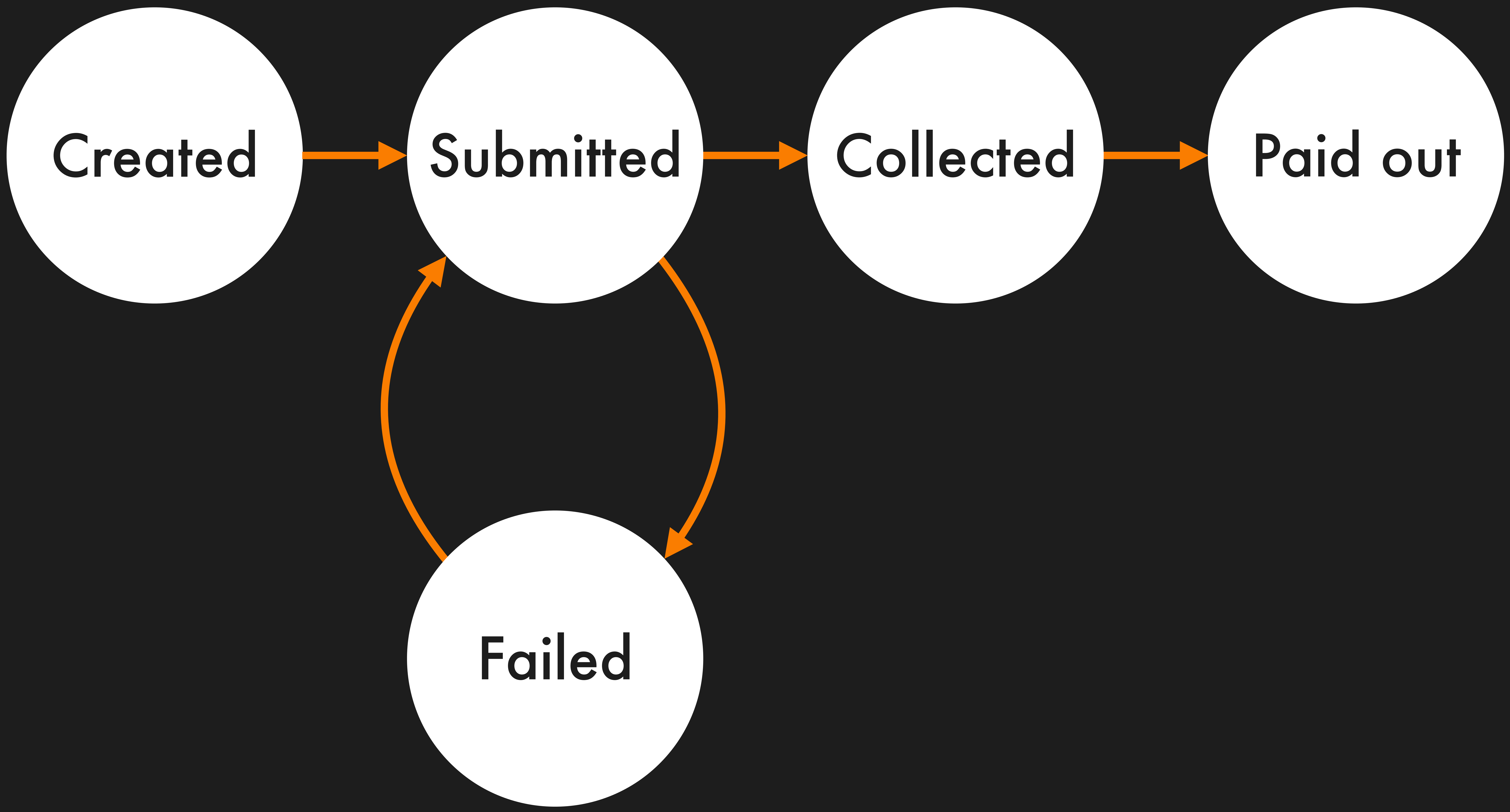
```
class Payment
  states(["created", "submitted", ...])

  allow_transition("created", "submitted")
  allow_transition("submitted", "collected")
  allow_transition("submitted", "failed")
...
```

State machine pseudocode

```
class Payment
  states(["created", "submitted", ...])

  allow_transition("created", "submitted")
  allow_transition("submitted", "collected")
  allow_transition("submitted", "failed")
  allow_transition("failed", "submitted")
...
```


Created
Submitted
Collected
Paid out
Failed

Often dismissed: "Too academic"

README.md

STATESMAN

A statesmanlike state machine library.

For our policy on compatibility with Ruby and Rails versions, see COMPATIBILITY.md.

gem version 10.0.0 circleci passing maintainability B gitter join chat

https://github.com/gocardless/statesman

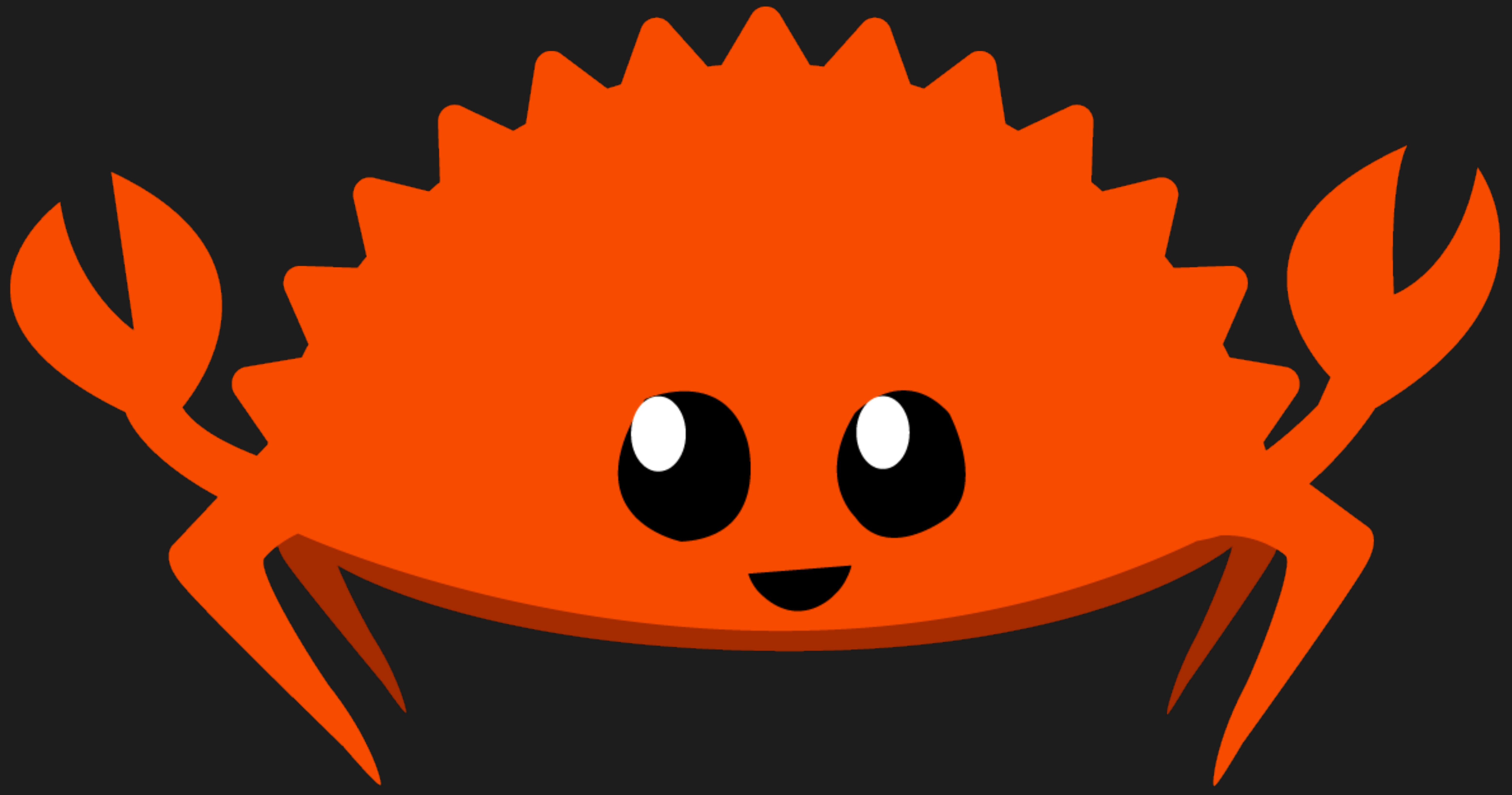
Make the problem

impossible

Example 2

Memory safety

Not here to sell you Rust



Something we

often

take for granted

But first, some C

Memory allocation in C

```
char *ptr = malloc(SIZE);
do_stuff(ptr);
free(ptr);
```

Use-after-free in C

```c
char *ptr = malloc(SIZE);
do_stuff(ptr);
free(ptr);
// Many lines more code
do_other_stuff(ptr);
```

Undefined behaviour

(You don't know what your program will do)

Undefined behaviour

(An attacker might be able to abuse it)

A non-scientific study

CVE List

Search CVE List Down

HOME > CVE > SEARCH RESULTS

Search Results

There are **534** CVE Records that match your search.

Name	
CVE-2022-42703	mm/rmap.c in the Linux kernel before 5.19.7 has a use-after-free

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=use+after+free+2022

A non-scientific study

CVE-ID	
CVE-2022-41849	[Learn more at National Vulnerability Database (NVD)] • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description
drivers/video/fbdev/smscufx.c in the Linux kernel through 5.19.12 has a race condition and resultant use-after-free if a physically proximate attacker removes a USB device while calling open(), aka a race condition between ufx_ops_open and ufx_usb_disconnect.

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-41849

You don't know which one will be serious

The assertion that we can simply code better is nonsense

Something we

often

take for granted

Garbage
collected
languages

Garbage collection pseudocode

```
def main()
  name = "Chris"
  greet(name)

def greet(name)
 puts("Hello #{name}")
```

Garbage collection pseudocode

```
def main()
  name = "Chris"
  greet(name)

def greet(name)
 puts("Hello #{name}")
```

Falls out of scope

The computer

does it

for you

Garbage collection is outrageously successful

Java

Go

Ruby

Python

JavaScript

C#

Haskell

Lisp

PHP

Erlang

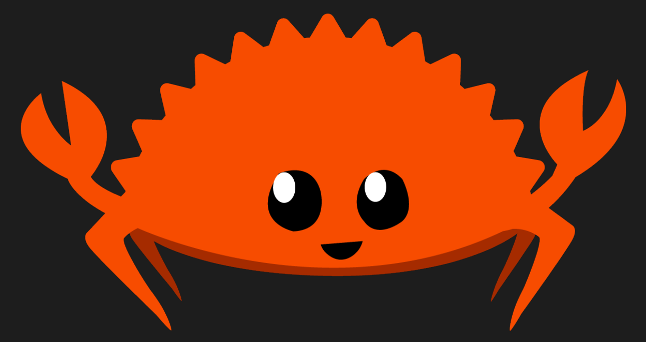
But what about…

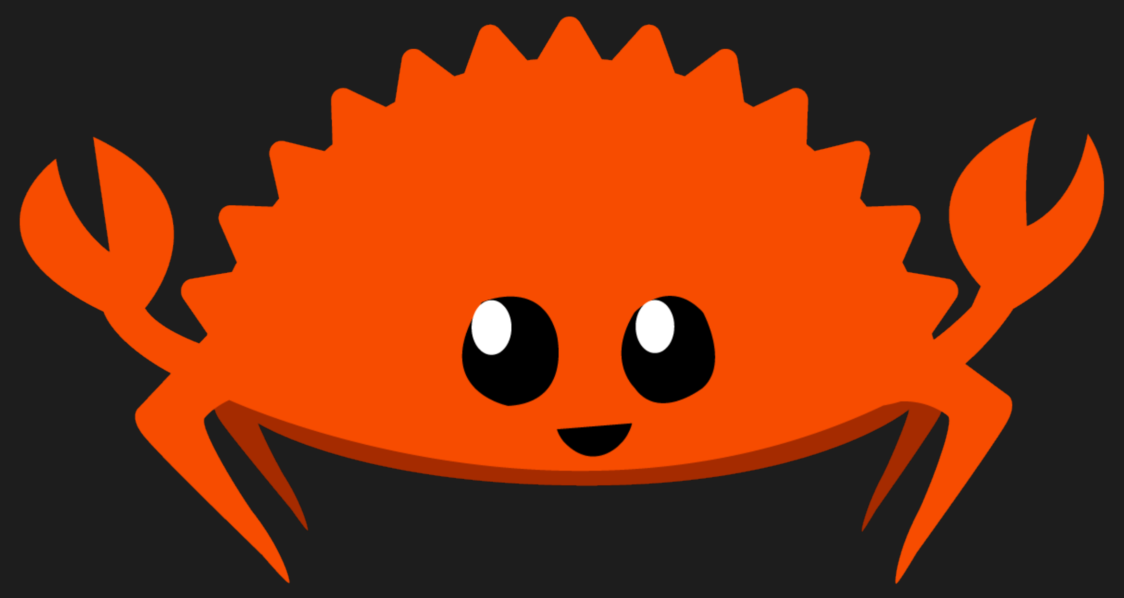
You don't always want a **runtime**

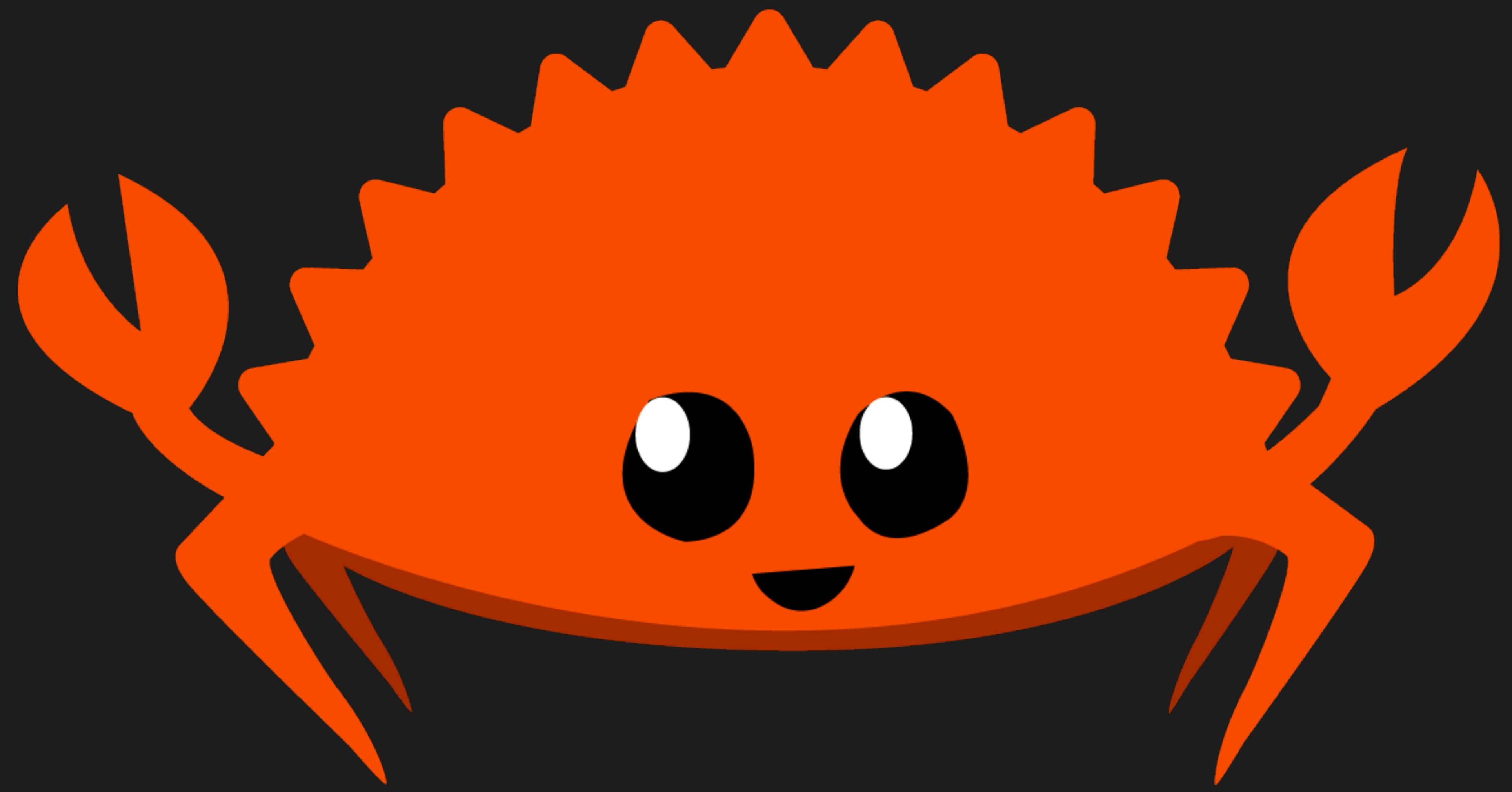

M50222
818Z05
163
CC
J.M. Turnauckas Photography
Highland Square, Akron

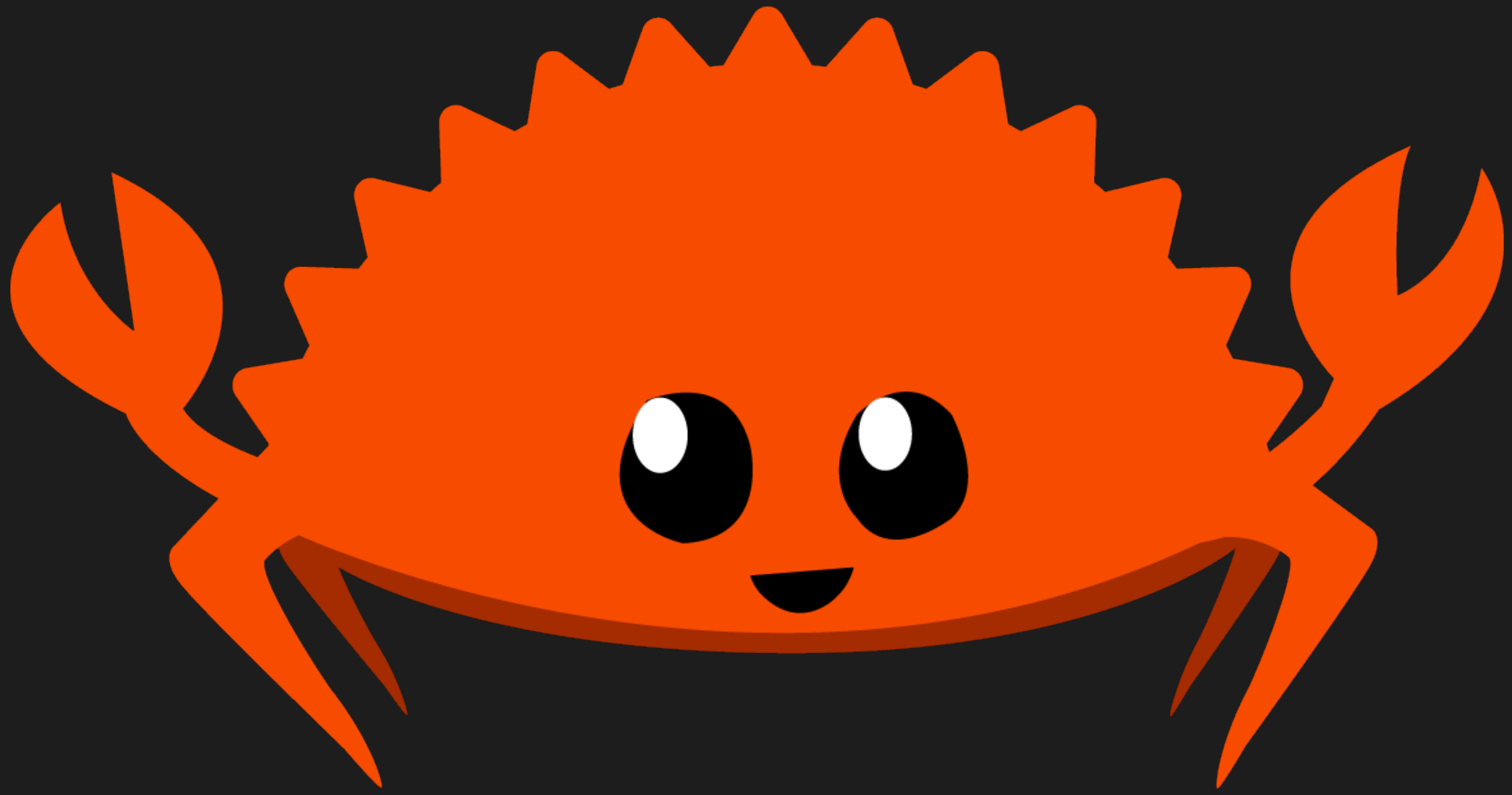
Stuck with manual memory management

Until...









Okay so hear me out

Ownership & borrow-checking

Tl;dr:

Every value in memory has **at most one** owner

Garbage collection pseudocode

```
def main()
  name = "Chris"
  greet(name)

def greet(name)
 puts("Hello #{name}")
```

Rust greetings

```rust
fn main() {
   let name = String::from("Chris");
   greet(name);
}

fn greet(name: String) {
  println!("Hello {}", name);
}
```

Rust greetings

```
fn main() {
   let name = String::from("Chris");
   greet(name);
}

fn greet(name: String) {
 println!("Hello {}", name);
}
```

Owner transferred

Rust greetings

```
fn main() {
  let name = String::from("Chris");
  greet(name);
}

fn greet(name: String) {
 println!("Hello {}", name);
}
```

Owner transferred

Falls out of scope

Owner out-of-scope

↓

Value dropped

Rust greetings

```
fn main() {
   let name = String::from("Chris");
   greet(name);
   say_goodbye(name);
}

fn greet(name: String) {
 println!("Hello {}", name);
}
```

Compiler error

Rust greetings

```
fn main() {
   let name = String::from("Chris");
   greet(&name);
   say_goodbye(name);
}

fn greet(name: &String) {
  println!("Hello {}", name);
}
```

Borrow

No

manual memory

management

The computer
does it
for you

No

GC

Make the problem impossible

Example 3

Database migrations

MySQL

(but also true in Postgres)

```sql
-- Create a table
CREATE TABLE payments (
  id int NOT NULL,
  ...
)
```

```sql
-- Create a table
CREATE TABLE payments (
  id int NOT NULL,
  ...
)

-- Realise `int` isn't large enough (2^32)
-- You're going to run out of IDs
ALTER TABLE payments MODIFY id bigint;
```

```sql
-- Create a table
CREATE TABLE payments (
  id int NOT NULL,
  ...
)

-- Realise `int` isn't large enough (2^32)
-- You're going to run out of IDs
ALTER TABLE payments MODIFY id bigint;
```

Blocks all other queries

The migrations reviewer

Add a new column

or

Recreate the table

The migrations

reviewer

🕵️

The migrations reviewer

😰

The migrations

reviewers

🕵️ 🕵️ 🕵️

The migrations

reviewers

😰 😰 😰

It doesn't

scale

and it's still

not enough

Seemingly innocuous

```sql
ALTER TABLE payments ADD COLUMN refunded boolean;
```

But can
still
be dangerous

```sql
-- Slow transaction
START TRANSACTION;
SELECT * FROM payments;
```

```sql
-- Slow transaction
START TRANSACTION;
SELECT * FROM payments;

-- Forces this to queue
ALTER TABLE payments ADD COLUMN refunded boolean;
```

```sql
-- Slow transaction
START TRANSACTION;
SELECT * FROM payments;

-- Forces this to queue
ALTER TABLE payments ADD COLUMN refunded boolean;

-- Which blocks these
SELECT * FROM payments WHERE id = 123;
```

PlanetScale

Vitess

Tl;dr:

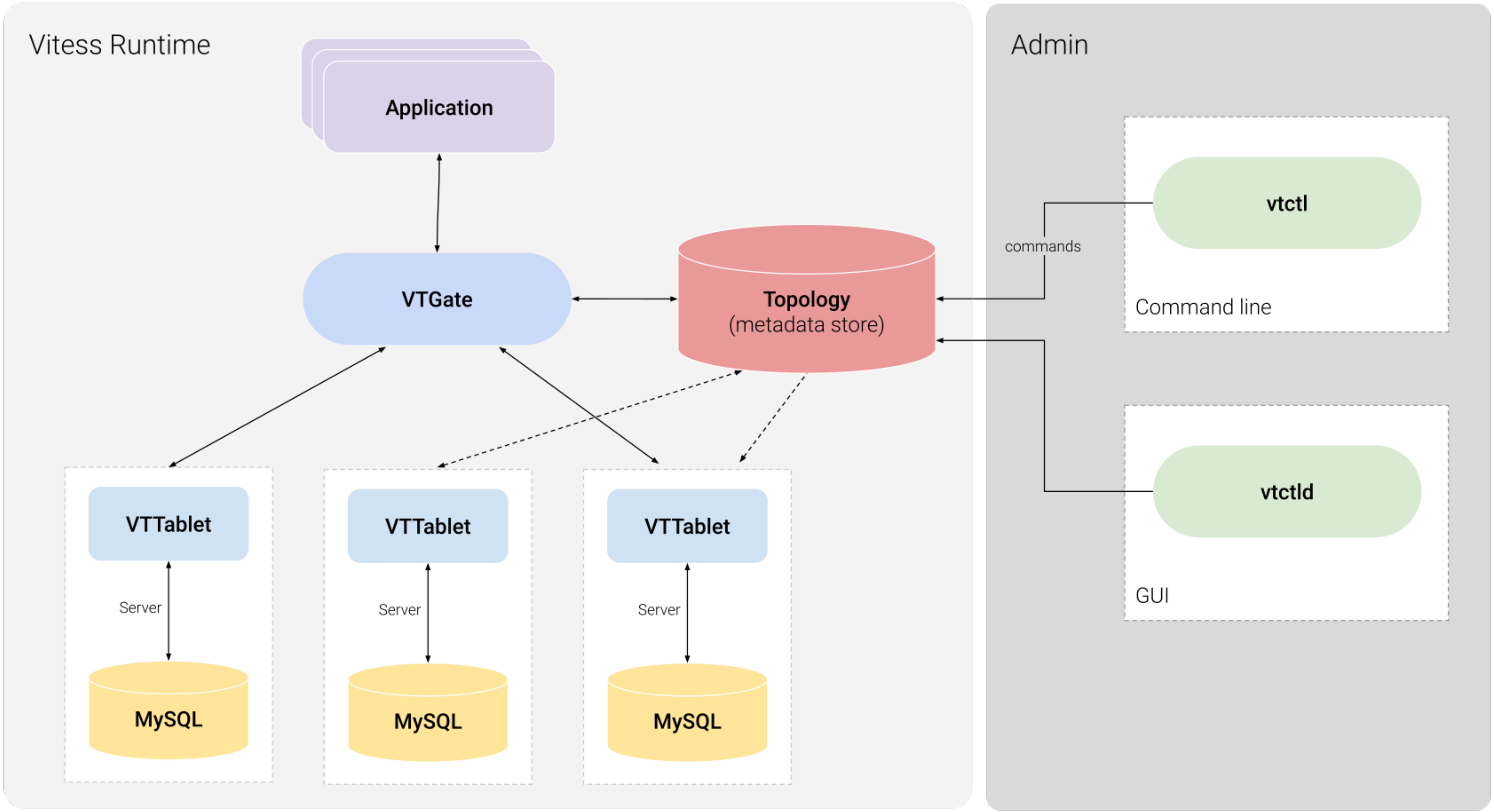
- MySQL-compatible

Tl;dr:

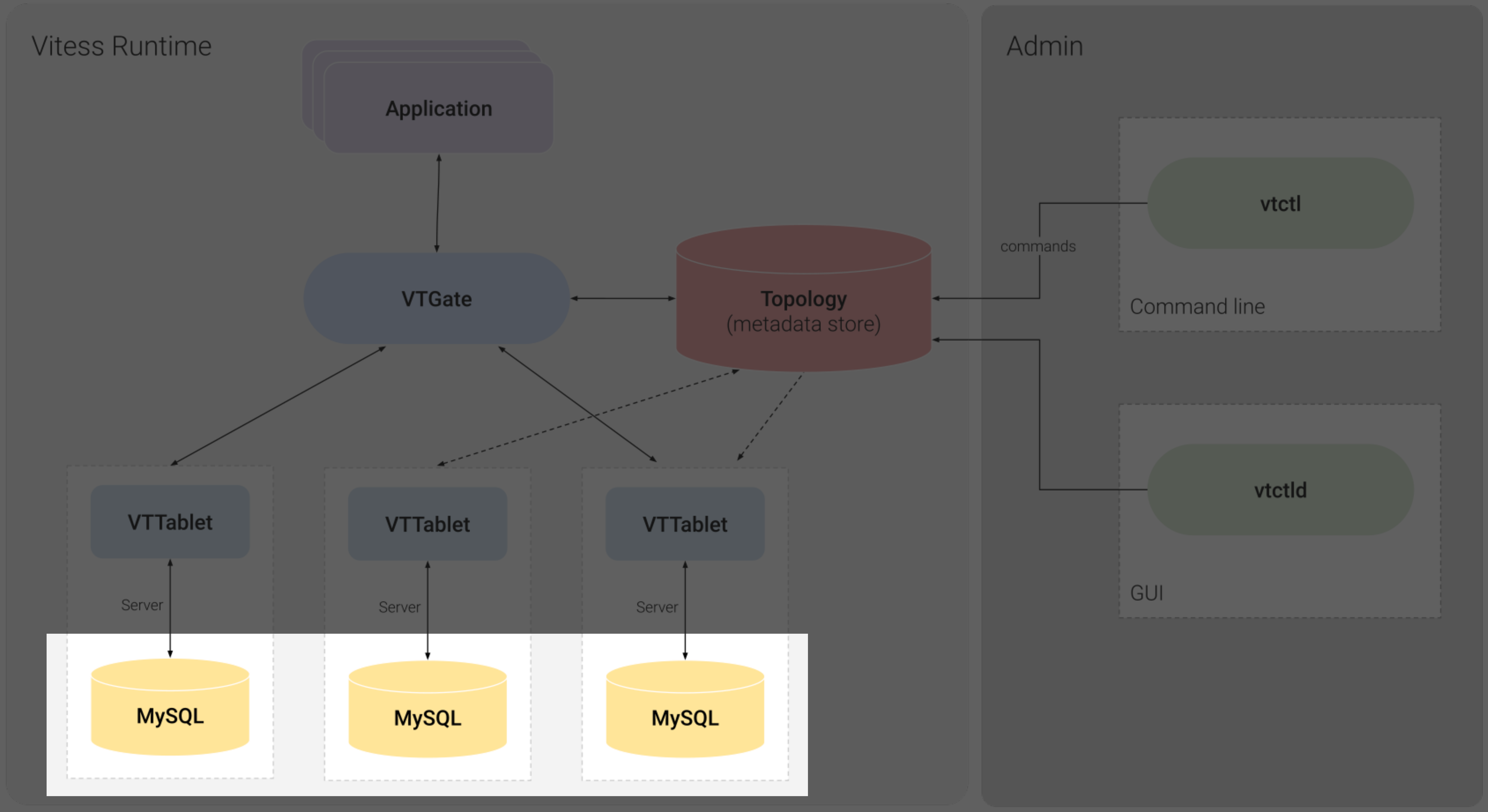
- MySQL-compatible
- Scalability (sharding)

Tl;dr:

- MySQL-compatible
- Scalability (sharding)
- High-availability

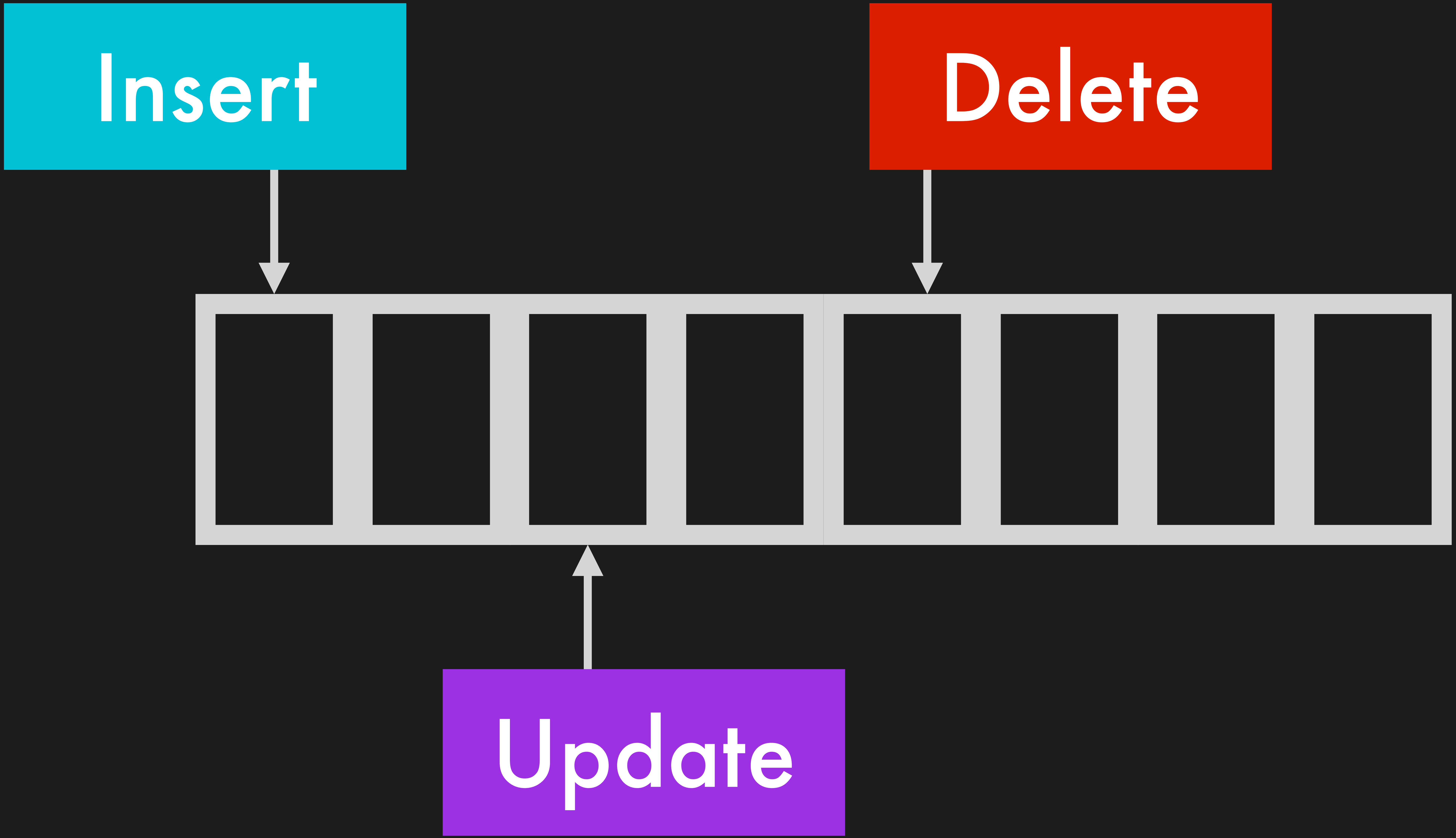


Vitess Runtime
Application
VTGate
Topology
(metadata store)
VTTablet
Server
MySQL
VTTablet
Server
MySQL
VTTablet
Server
MySQL
Admin
commands
vtctl
Command line
vtctld
GUI


Vitess Runtime
Application
VTGate
Topology
(metadata store)
VTTablet
VTTablet
VTTablet
Server
Server
Server
MySQL
MySQL
MySQL
Admin
commands
vtctl
Command line
vtctld
GUI

VReplication

A stream of changes


Insert
Delete
Update

```
ALTER TABLE payments MODIFY id bigint;
```

ALTER TABLE payments MODIFY id bigint;

id (int)	description
1	Laptop
2	Phone

ALTER TABLE payments MODIFY id bigint;

id (int)	description
1	Laptop
2	Phone

id (bigint)	description

ALTER TABLE payments MODIFY id bigint;

id (int)	description
1	Laptop
2	Phone

id (bigint)	description
1	Laptop

ALTER TABLE payments MODIFY id bigint;

id (int)	description
1	Laptop
2	Phone
3	Unused domain renewal

id (bigint)	description
1	Laptop

ALTER TABLE payments MODIFY id bigint;

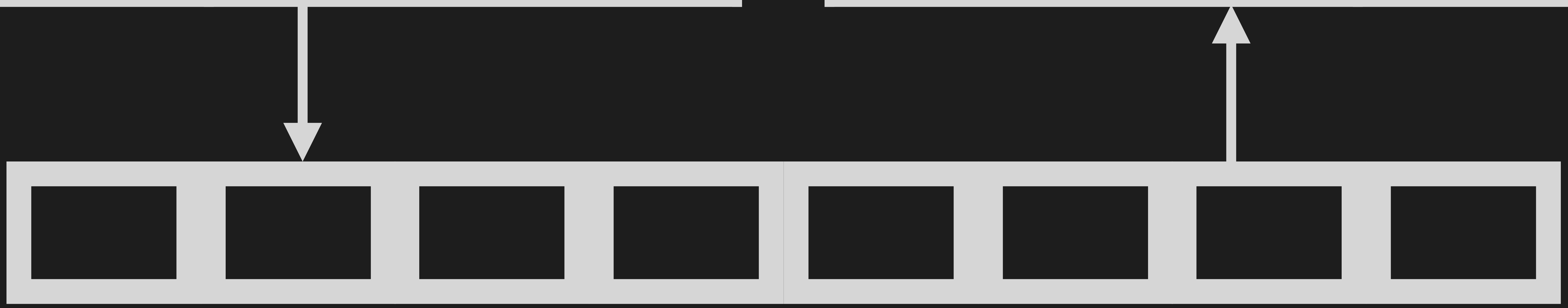
id (int)	description
1	Laptop
2	Phone
3	Unused domain renewal

id (bigint)	description
1	Laptop
2	Phone

ALTER TABLE payments MODIFY id bigint;

id (int)	description
1	Laptop
2	Phone
3	Unused domain renewal

id (bigint)	description
1	Laptop
2	Phone
3	Unused domain renewal



ALTER TABLE payments MODIFY id bigint;

id (int)	description
1	Laptop
2	Phone
3	Unused domain renewal

id (bigint)	description
1	Laptop
2	Phone
3	Unused domain renewal

ALTER TABLE payments MODIFY id bigint;

id (int)	description
1	Laptop
2	Phone
3	Unused domain renewal

id (bigint)	description
1	Laptop
2	Phone
3	Unused domain renewal

User queries (via proxy)

ALTER TABLE payments MODIFY id bigint;

id (int)	description
1	Laptop
2	Phone
3	Unused domain renewal

id (bigint)	description
1	Laptop
2	Phone
3	Unused domain renewal

User queries (via proxy)

ALTER TABLE payments MODIFY id bigint;

id (int)	description
1	Laptop
2	Phone
3	Unused domain renewal

id (bigint)	description
1	Laptop
2	Phone
3	Unused domain renewal

User queries (via proxy)

Fully-online

schema

migrations

The migrations

reviewers

😰 😰 😰

People doing
their actual job
😀 😀 😀

Make the problem

impossible

Examples

Example 1

State machines

Example 2

Memory safety

Example 3

Database migrations

Take aways:

- Complementary technique

SLOs

are alive

and well

Percentage

solutions

are too

Percentage solutions

Instances go unhealthy

↓

Add health checks & route traffic away

Regional network issues

↓

Serve from multiple regions

Rare slow requests

↓

Add timeouts to protect majority of traffic

A complementary technique

GoCardless

GoCardless

Resources → Technology →

Fear-free PostgreSQL migrations for Rails

Written by James Turley
Last edited Mar 2020

https://gocardless.com/blog/fear-free-postgresql-migrations-for-rails/

Take aways:

- Complementary technique
- You have to write software

No code changes

Instances go unhealthy
↓
Add health checks & route traffic away

Regional network issues
↓
Serve from multiple regions

Rare slow requests
↓
Add timeouts to protect majority of traffic

This is

not

one of them

Sometimes BIG

Sometimes small

Not everyone can build a database

README.md

★ STATESMAN ★

A statesmanlike state machine library.

For our policy on compatibility with Ruby and Rails versions, see COMPATIBILITY.md.

gem version 10.0.0 | circleci passing | maintainability B | gitter join chat

https://github.com/gocardless/statesman

Maybe

someone

already solved it

Take aways:

- Complementary technique
- You have to write software
- It's not easy to spot

Take aways:

- Complementary technique
- You have to write software
- It's not easy to spot
  - But there are some tells

The migrations reviewer

🕵️

Smug internet comments

🙄

Smug internet comments

🙄

Examples:

- State machines
- Memory safety
- Database migrations

Smug comments:

- State machines
- Memory safety
- Database migrations

Smug comments:

Add more unit tests

- State machines
- Memory safety
- Database migrations

Smug comments:

- State machines
- Memory safety
- Database migrations

Add more unit tests

Write better C

Smug comments:

Add more unit tests

- State machines
- Memory safety
- Database migrations

Write better C

Just hire a DBA

There's probably more to it

The assertion that we can simply code better is nonsense

We

can

do better

Thank you

✌🏻❤️

@ChrisSinjo
@planetscaledata

Image credits

- Poker Winnings - slgckgc - CC-BY - https://www.flickr.com/photos/slgc/42157896194/
- Thinking Face - Twemoji - CC-BY - https://github.com/twitter/twemoji
- Ferris (Extra-cute) - Unofficial Rust mascot - Copyright waived - https://rustacean.net/
- A350 Board - Mark Turnauckas - CC-BY - https://www.flickr.com/photos/marktee/17118767669/
- Play - Annie Roi - CC-BY - https://www.flickr.com/photos/annieroi/4421442720/

Image credits

- White jigsaw puzzle with missing piece - Marco Verch Professional Photographer - CC-BY - https://www.flickr.com/photos/30478819@N08/50605134766/
- Hedge maze - claumoho - CC-BY - https://flickr.com/photos/claudiah/3929921991/
- photo_1405_20060410 - Robo Android - CC-BY - https://www.flickr.com/photos/49140926@N07/6798304070/
- Gears - Mustang Joe - Public Domain - https://www.flickr.com/photos/mustangjoe/20437315996/

Questions?

✌🏻❤️

@ChrisSinjo
@planetscaledata