

Hi, I’m Rosa



github.com/rosa



rosa.codes

programmer at 37signals

Solid Queue: two years on Rails

SF Ruby, 11 November 2026


v1.0.0
Really ~~battle~~ Fun™-tested stuff
RAILS WORLD 2024


v1.0.0
Really battle Fun™-tested stuff
I didn’t know how much
Fun™ was yet to come!
RAILS WORLD

Solid Queue 1.0

Solid Queue 1.0

- Active Job backend
- Based on Active Record (relational DB)
- MySQL, PostgreSQL, SQLite
- No need for Redis

Solid Queue 1.0

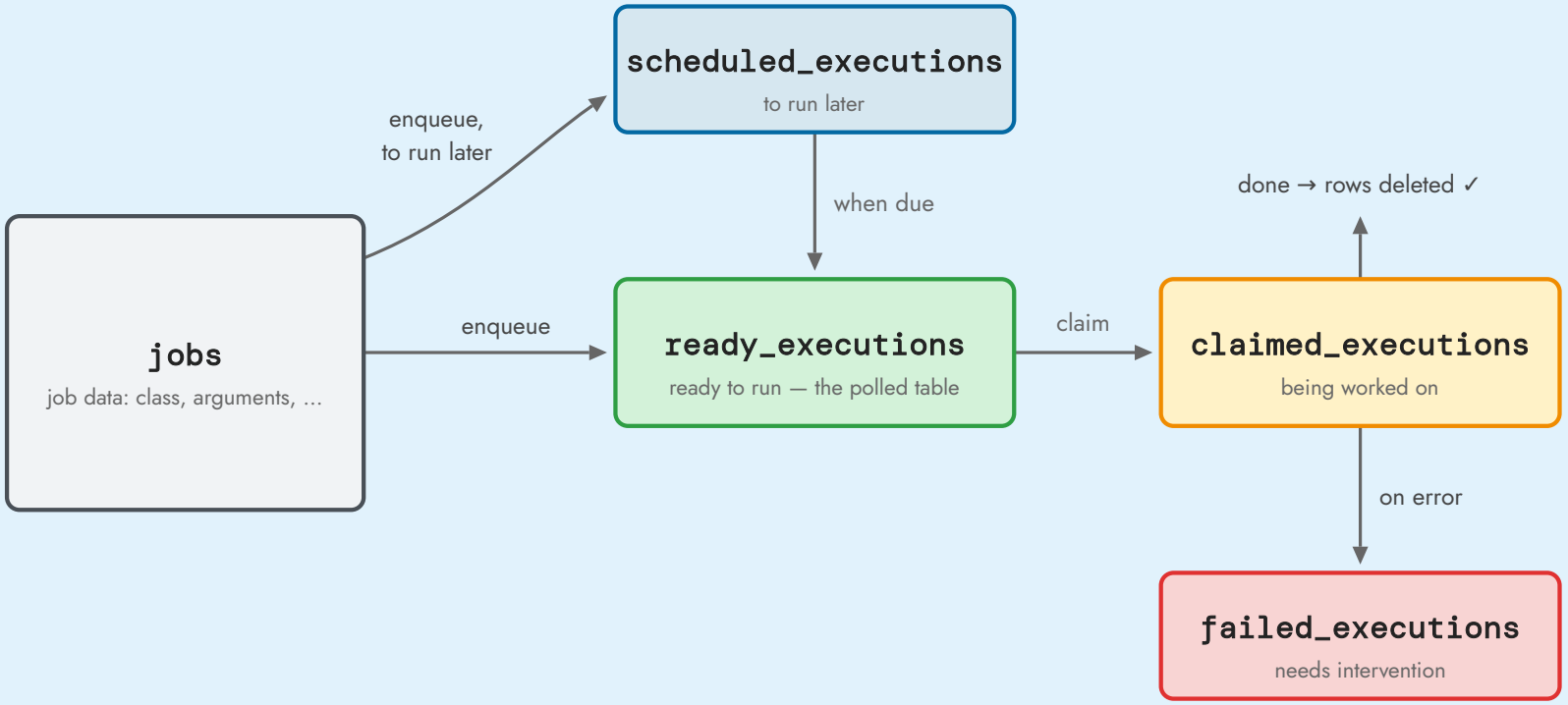
```
# Jobs
gem "resque"
gem "resque_supervised_fork"
gem "resque-pool"
gem "resque-scheduler"
gem "resque-pause"
gem "sequential_jobs"
gem "scheduled_job"
```



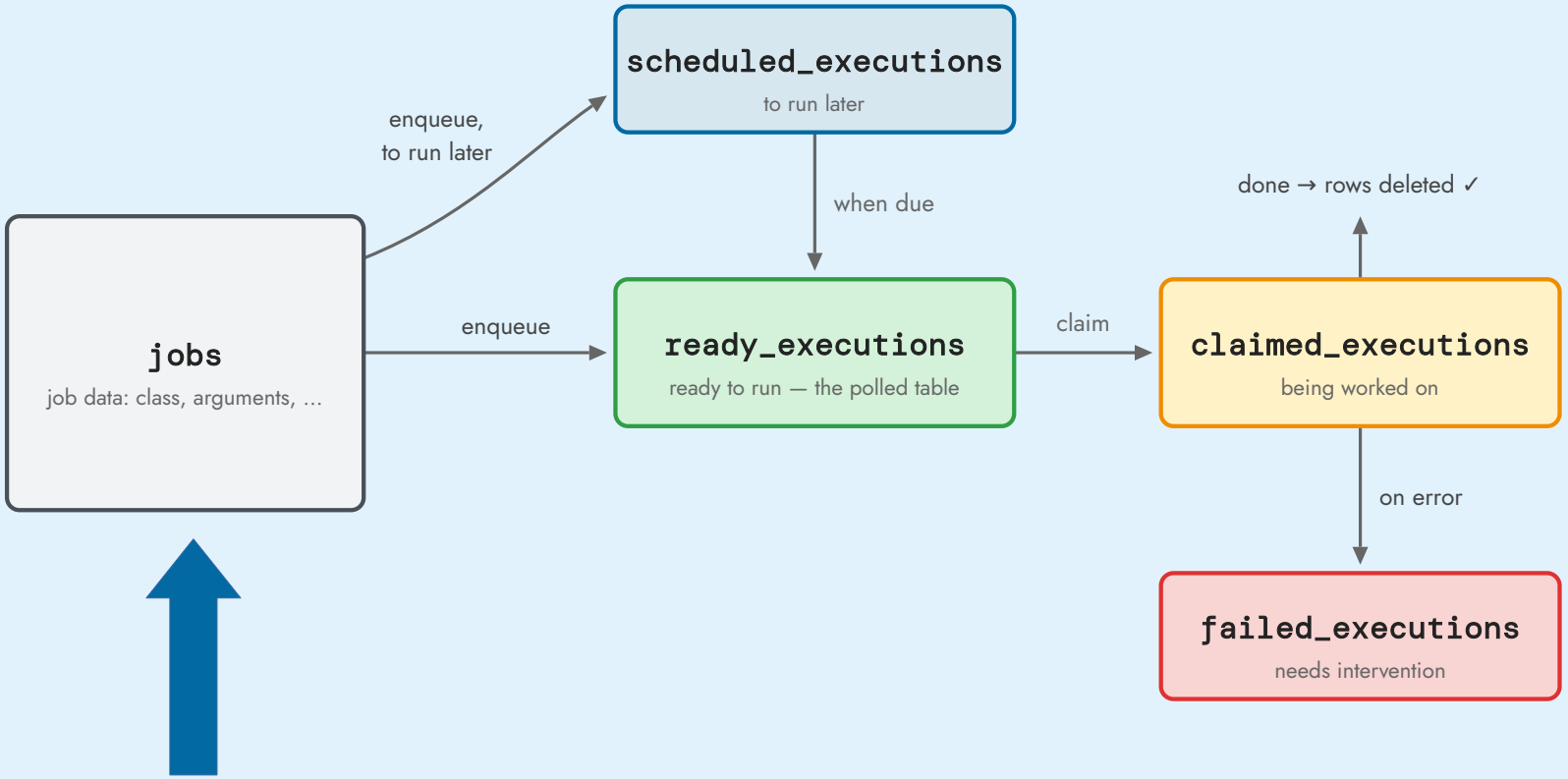

```
# Jobs
gem "solid_queue"
```



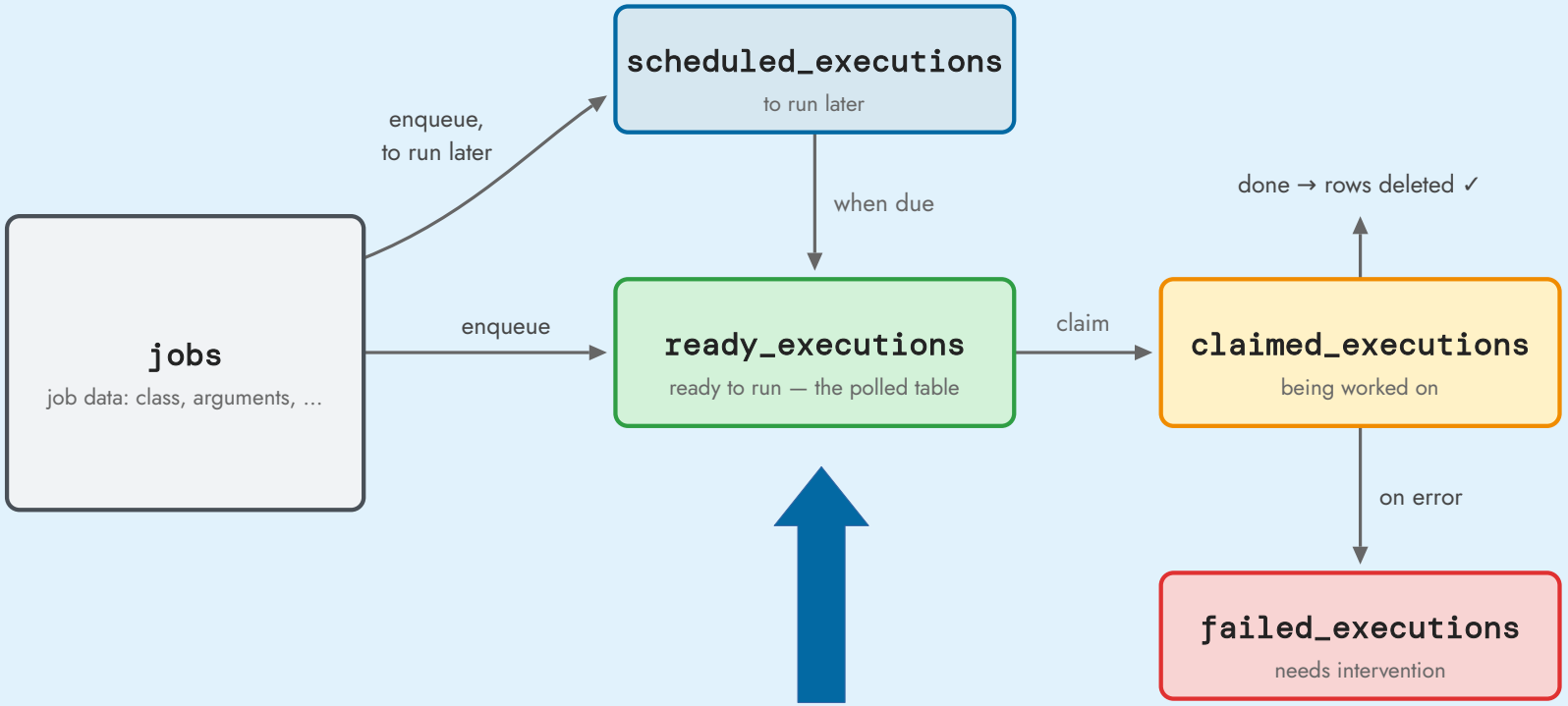
Solid Queue 1.0



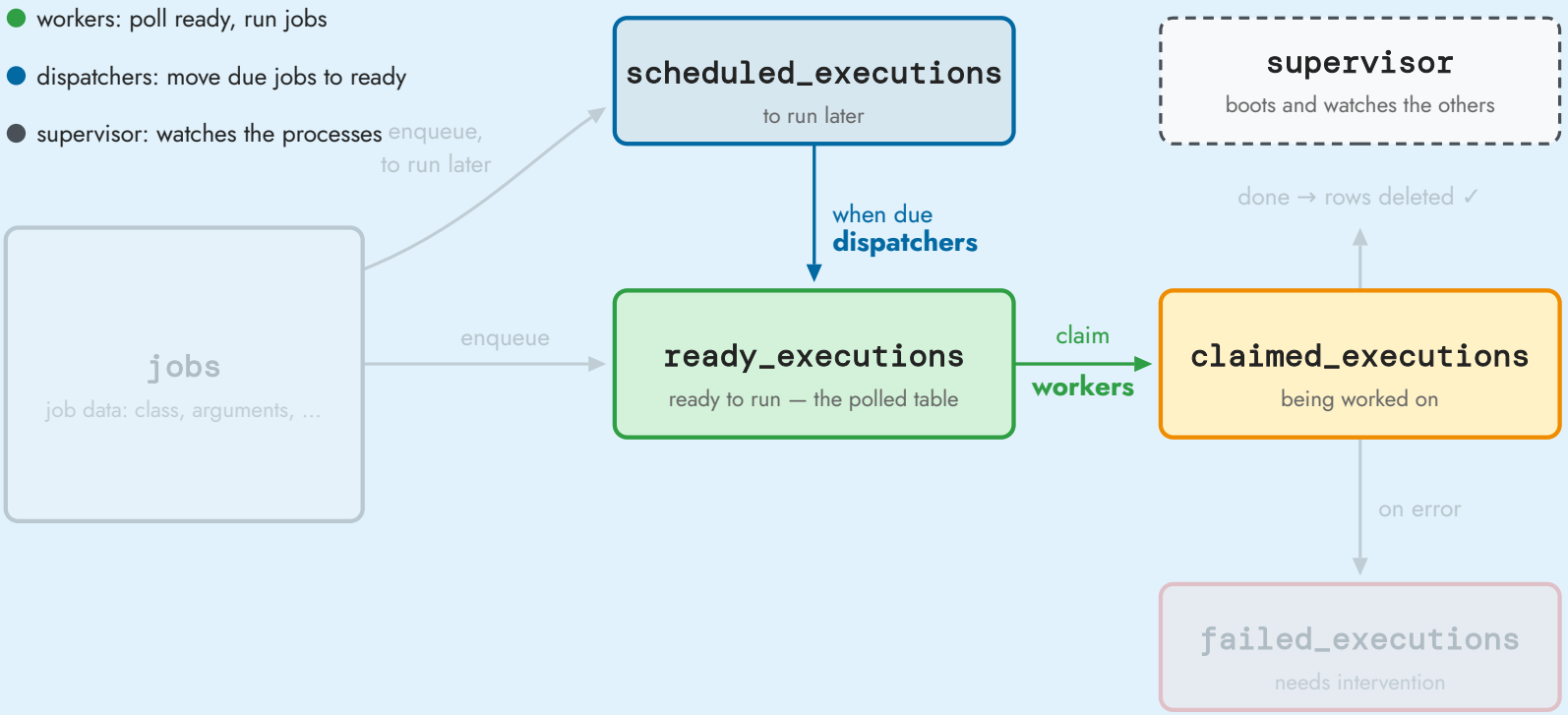
Solid Queue 1.0



Solid Queue 1.0



Solid Queue 1.0



Solid Queue 1.0

- Enqueue, poll and run jobs ✔
- Keep failed jobs for manual retry ✔
- Allow scheduling jobs in the future ✔

Solid Queue 1.0

- Enqueue, poll and run jobs ✔
- Keep failed jobs for manual retry ✔
- Allow scheduling jobs in the future ✔
- Sequential jobs?

Solid Queue 1.0

- Enqueue, poll and run jobs ✔
- Keep failed jobs for manual retry ✔
- Allow scheduling jobs in the future ✔
- ~~Sequential jobs?~~ Concurrency controls

Solid Queue 1.0

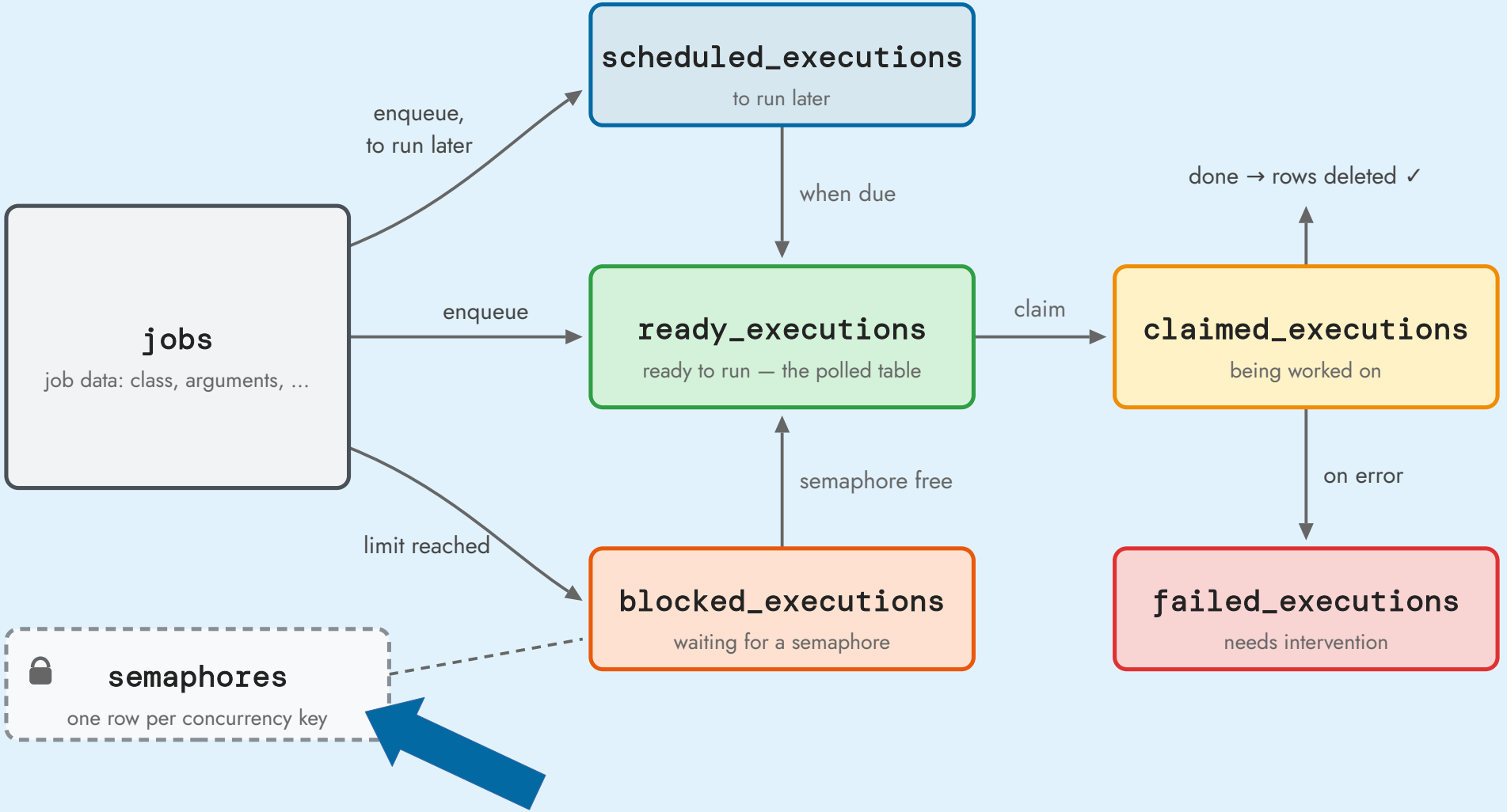
Concurrency controls

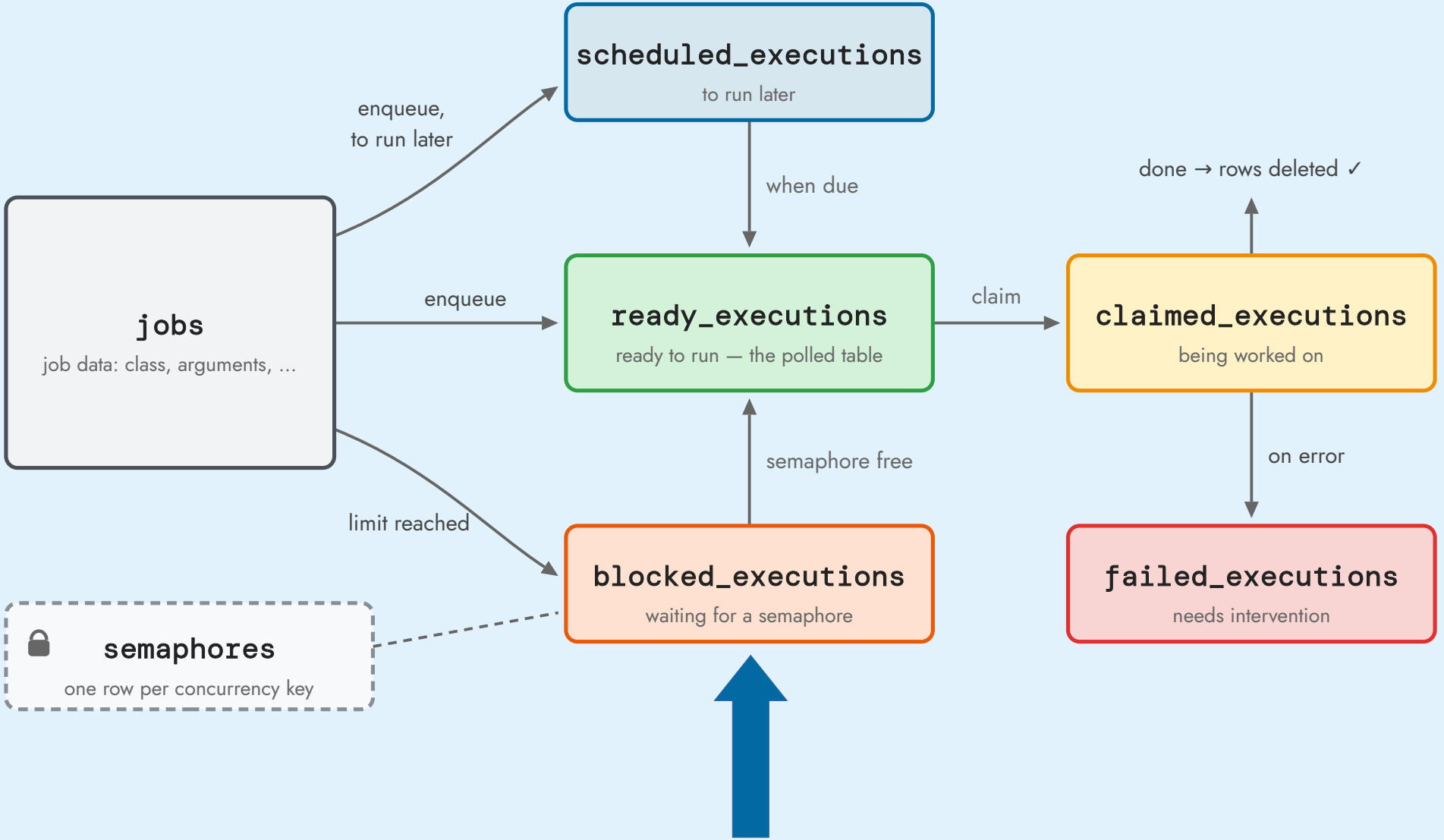
```
class Recording::StatusPropagationJob
  limits_concurrency limit: 1,
                     key: ->(recording, _, _) { recording.bucket_id },
                     duration: 1.day

  # ...
```

Solid Queue 1.0

Concurrency controls


scheduled_executions
to run later
enqueue,
to run later
when due
done → rows deleted ✓
jobs
job data: class, arguments, ...
enqueue
ready_executions
ready to run — the polled table
claim
claimed_executions
being worked on
semaphore free
on error
limit reached
blocked_executions
waiting for a semaphore
failed_executions
needs intervention
semaphores
one row per concurrency key


scheduled_executions
to run later
enqueue,
to run later
when due
done → rows deleted ✓
jobs
job data: class, arguments, ...
enqueue
ready_executions
ready to run — the polled table
claim
claimed_executions
being worked on
semaphore free
on error
limit reached
blocked_executions
waiting for a semaphore
failed_executions
needs intervention
semaphores
one row per concurrency key

Solid Queue 1.0

Concurrency controls



Solid Queue 1.0

- Enqueue, poll and run jobs ✔
- Keep failed jobs for manual retry ✔
- Allow scheduling jobs in the future ✔
- Concurrency controls ✔

Solid Queue 1.0

- Enqueue, poll and run jobs ✔
- Keep failed jobs for manual retry ✔
- Allow scheduling jobs in the future ✔
- Concurrency controls ✔
- Recurring jobs

Solid Queue 1.0

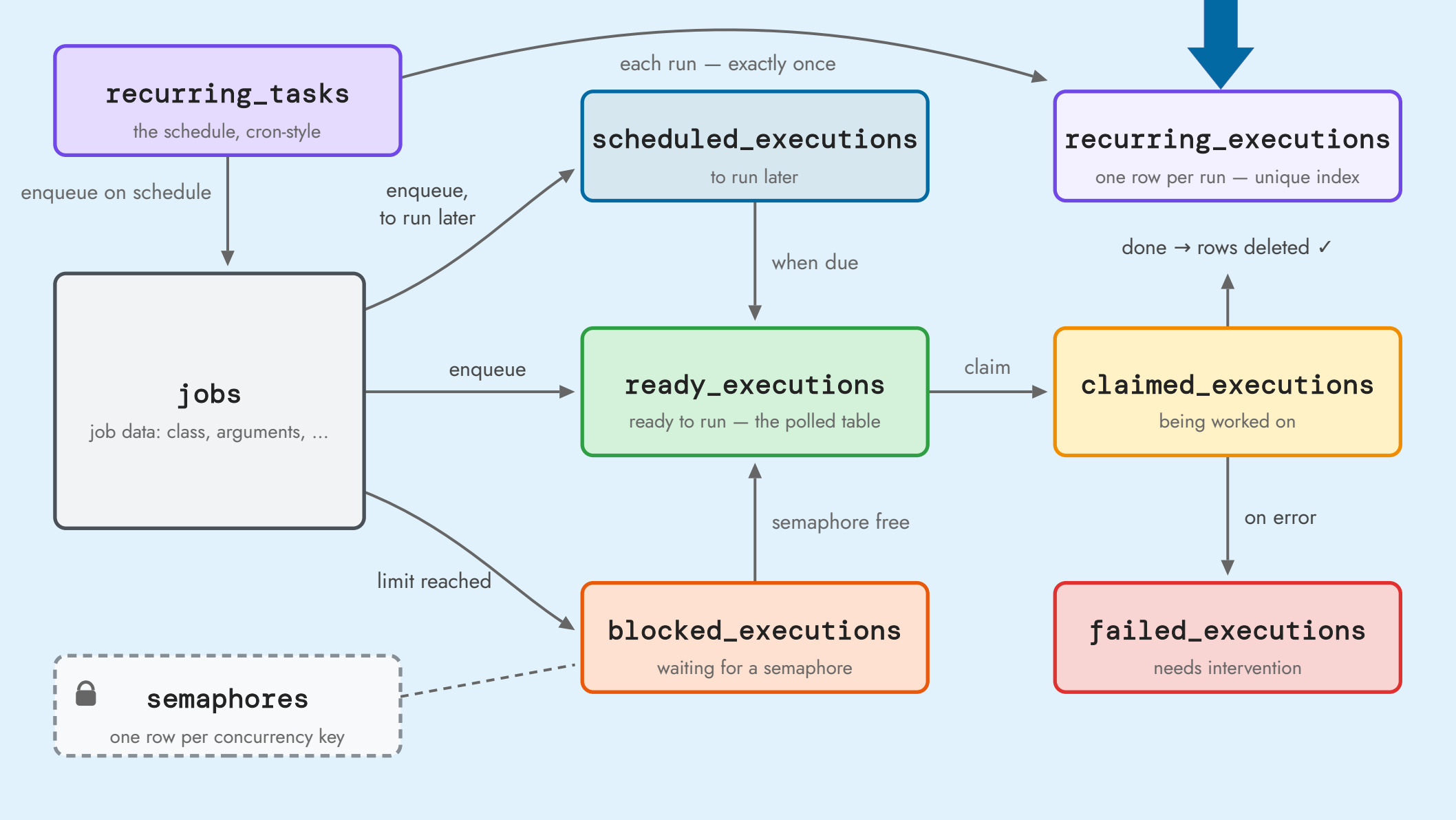
Recurring jobs

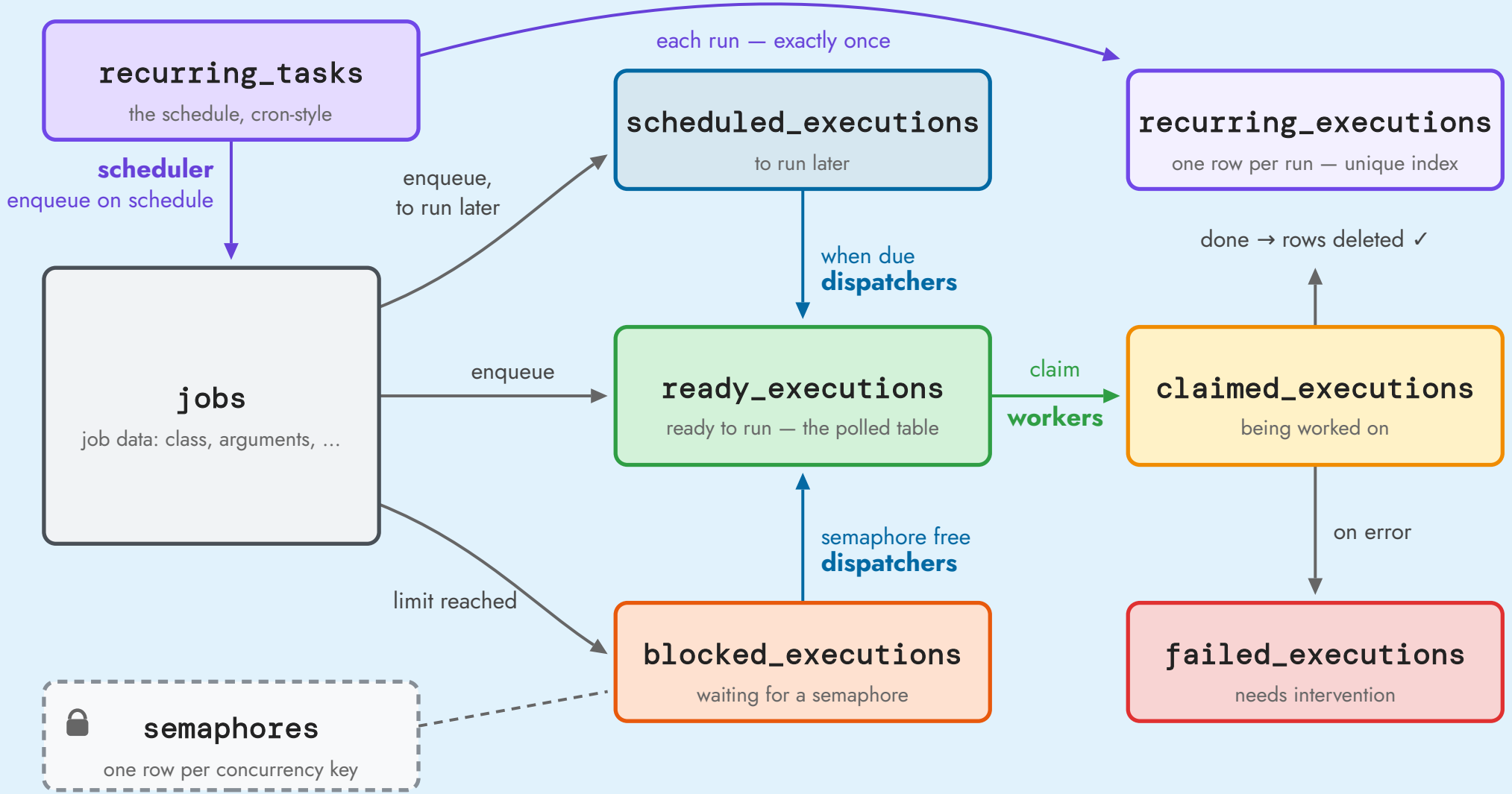
```yaml
# config/recurring.yml
production:
  deliver_bundled_notifications:
    command: "Notification::Bundle.deliver_all_later"
    schedule: every 30 minutes

  delete_unused_tags:
    class: DeleteUnusedTagsJob
    schedule: every day at 04:02

  clear_solid_queue_finished_jobs:
    command: "SolidQueue::Job.clear_finished_in_batches"
    schedule: every hour at minute 12

  # ...
```


recurring_tasks
the schedule, cron-style
each run — exactly once
recurring_executions
one row per run — unique index
enqueue on schedule
enqueue,
to run later
scheduled_executions
to run later
when due
done → rows deleted ✓
jobs
job data: class, arguments, ...
enqueue
ready_executions
ready to run — the polled table
claim
claimed_executions
being worked on
semaphore free
on error
limit reached
blocked_executions
waiting for a semaphore
failed_executions
needs intervention
semaphores
one row per concurrency key


recurring_tasks
the schedule, cron-style
each run — exactly once
recurring_executions
one row per run — unique index
scheduler
enqueue on schedule
enqueue,
to run later
scheduled_executions
to run later
when due
dispatchers
done → rows deleted ✓
jobs
job data: class, arguments, ...
enqueue
ready_executions
ready to run — the polled table
claim
workers
claimed_executions
being worked on
on error
semaphore free
dispatchers
limit reached
blocked_executions
waiting for a semaphore
failed_executions
needs intervention
semaphores
one row per concurrency key
scheduler: enqueue recurring tasks
workers: poll ready, run jobs
dispatchers: move due jobs, release blocked

Solid Queue 1.0

- Enqueue, poll and run jobs ✔
- Keep failed jobs for manual retry ✔
- Allow scheduling jobs in the future ✔
- Concurrency controls ✔
- Recurring jobs ✔

Solid Queue 1.0

built for one app…



- ~20M jobs / day
- ~800 workers, 74 supervisors
- Big, dedicated MySQL instance

… but adopted by others

- PostgreSQL at a large scale
- Single server, SQLite
- LLM-heavy jobs

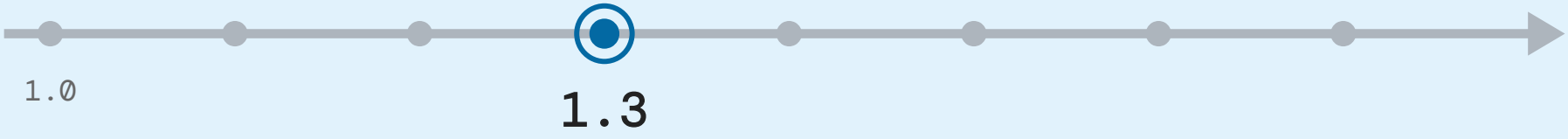
From 1.0 to 2.0 

- 23 releases
- 176 PRs merged
- 127 contributors, ~80 new since 1.0
- ~28.6M downloads

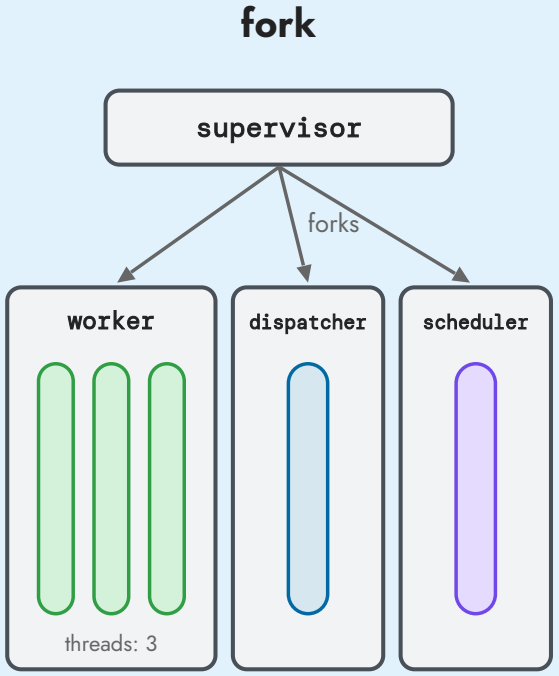


1.0

Async/single-process mode



Async/single-process mode

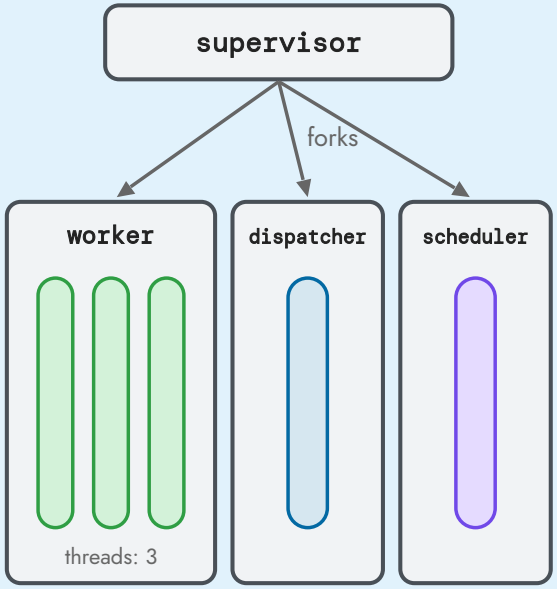


a separate process for each one —
good isolation, more memory

Async/single-process mode

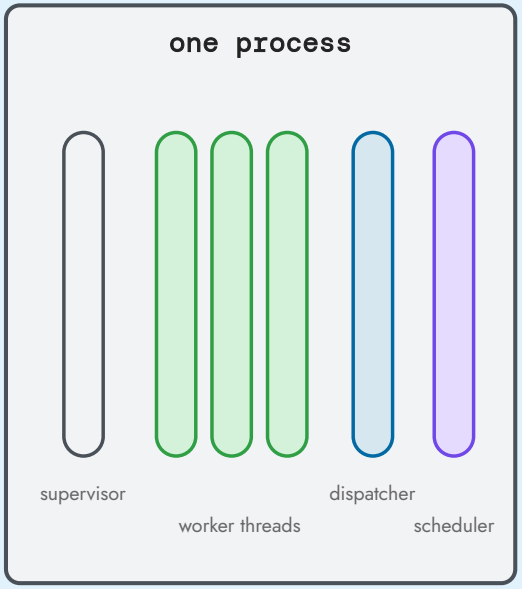


fork



a separate process for each one —
good isolation, more memory

async

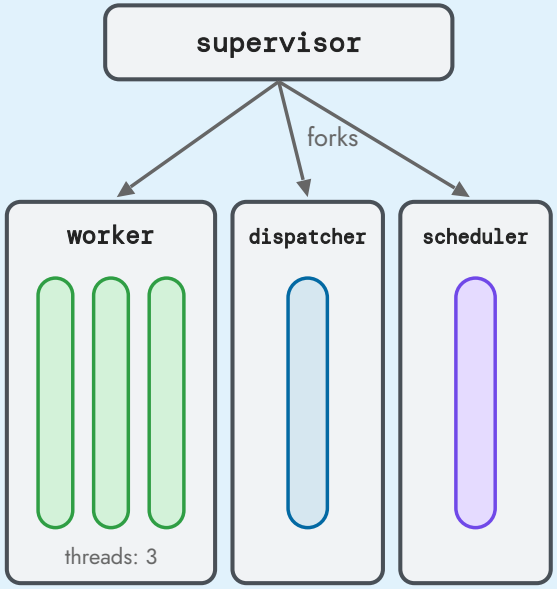


everything runs on threads,
in a single process

Async/single-process mode

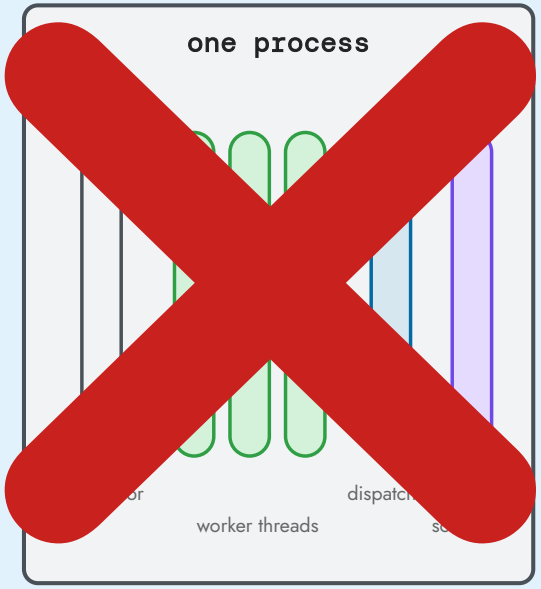


fork



a separate process for each one —
good isolation, more memory

async



everything runs on threads,
in a single process

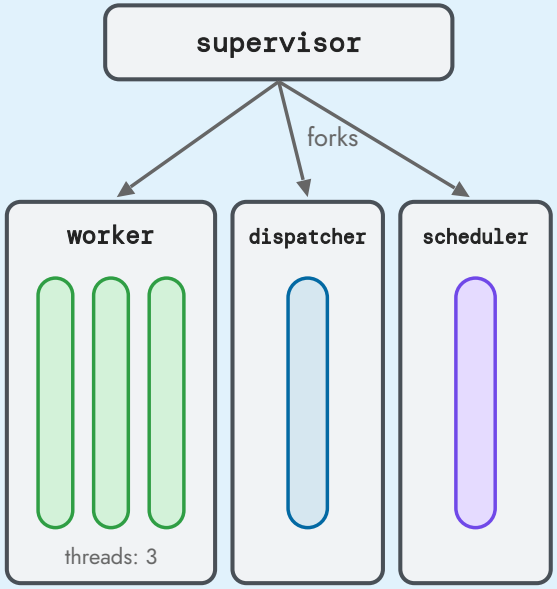


@bensheldon

Async/single-process mode

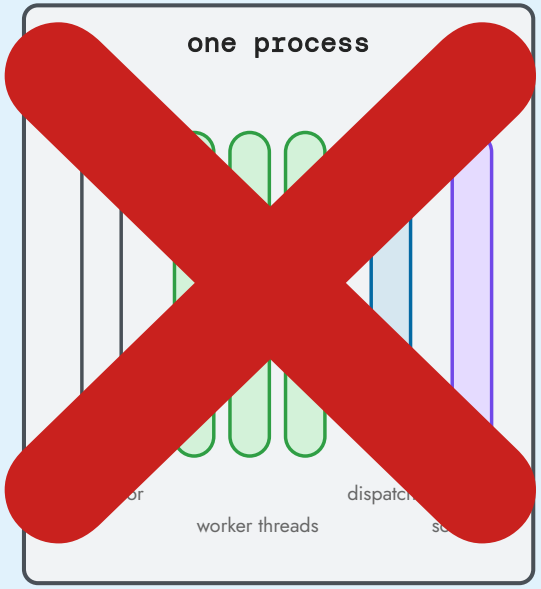


fork



a separate process for each one —
good isolation, more memory

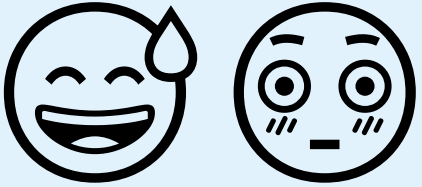
async



everything runs on threads,
in a single process



@bensheldon



Async/single-process mode 

```
# single-process mode
$ bin/jobs --mode async
```



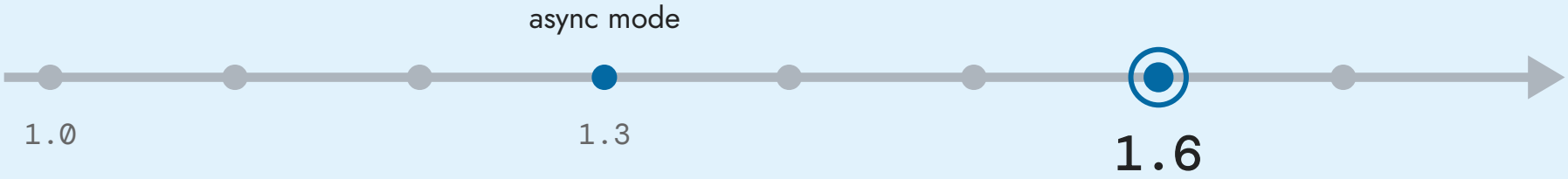
@joshleblanc

~700MB → ~270MB



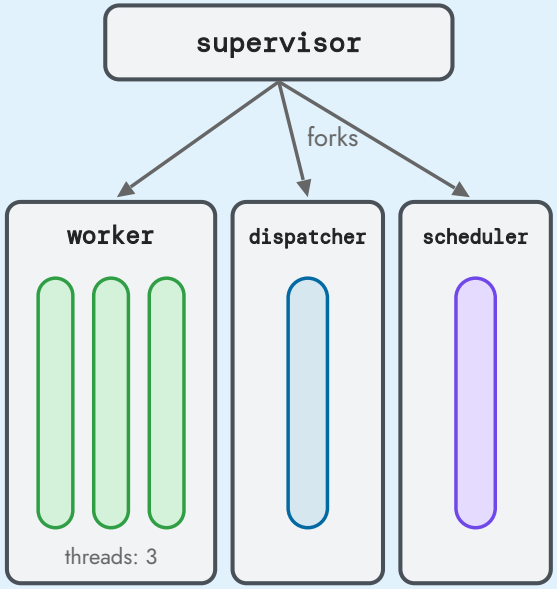


Fibers 



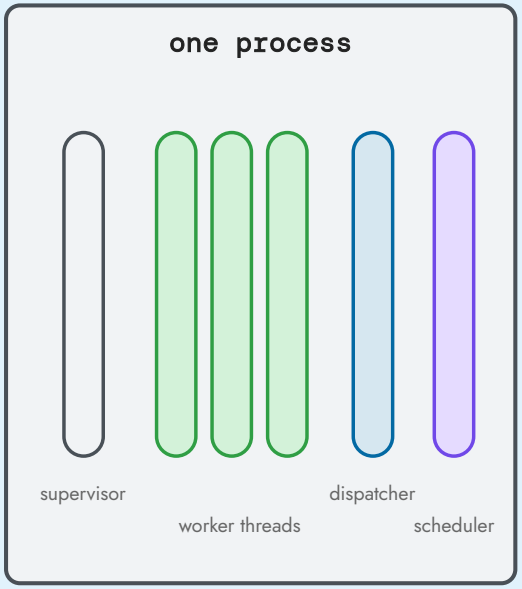
Fibers

fork



a separate process for each one —
good isolation, more memory

async



everything runs on threads,
in a single process



@crmne



Fibers



@crmne

```yaml
# config/queue.yml
workers:
  - queues: "api*"
    threads: 3
    polling_interval: 0.05
```

Fibers



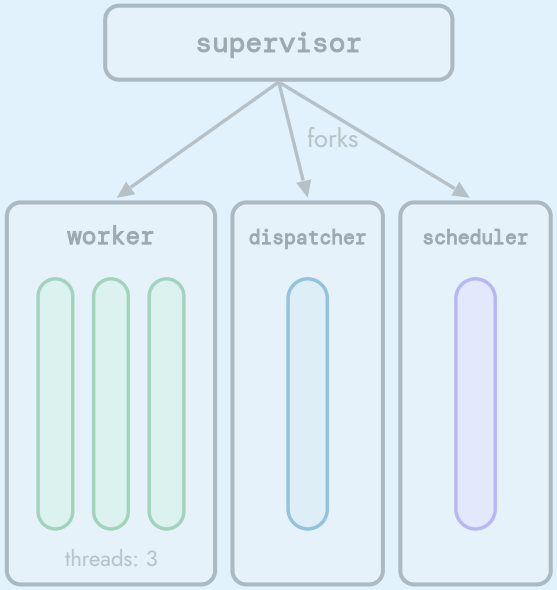
@crmne

```yaml
# config/queue.yml
workers:
  - queues: "api*"
    fibers: 100
    polling_interval: 0.05
```

- Built on the Async gem
- Fork or *async* mode

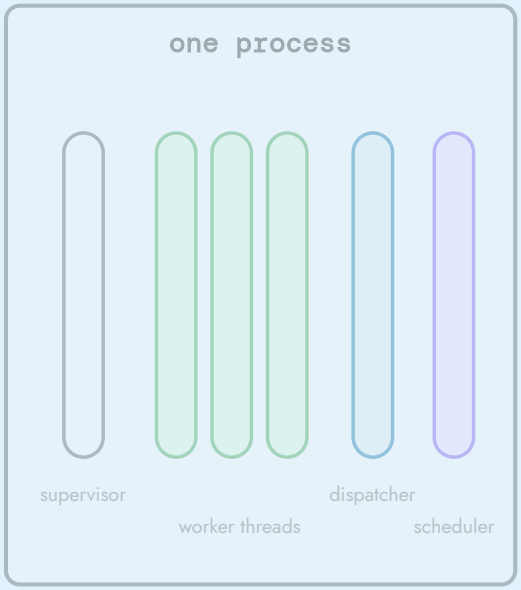
Fibers

fork



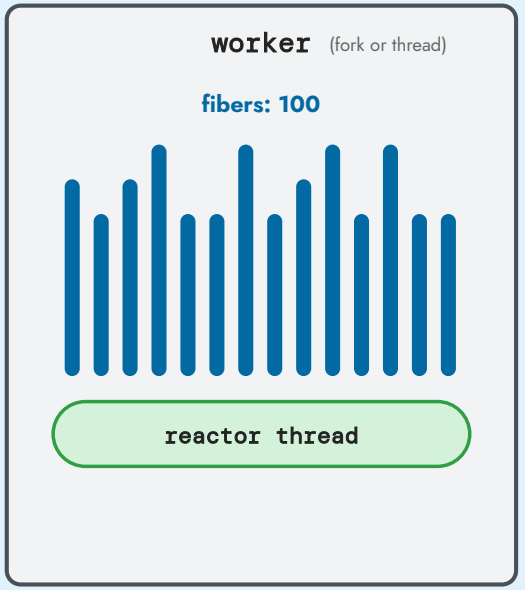
a separate process for each one —
good isolation, more memory

async



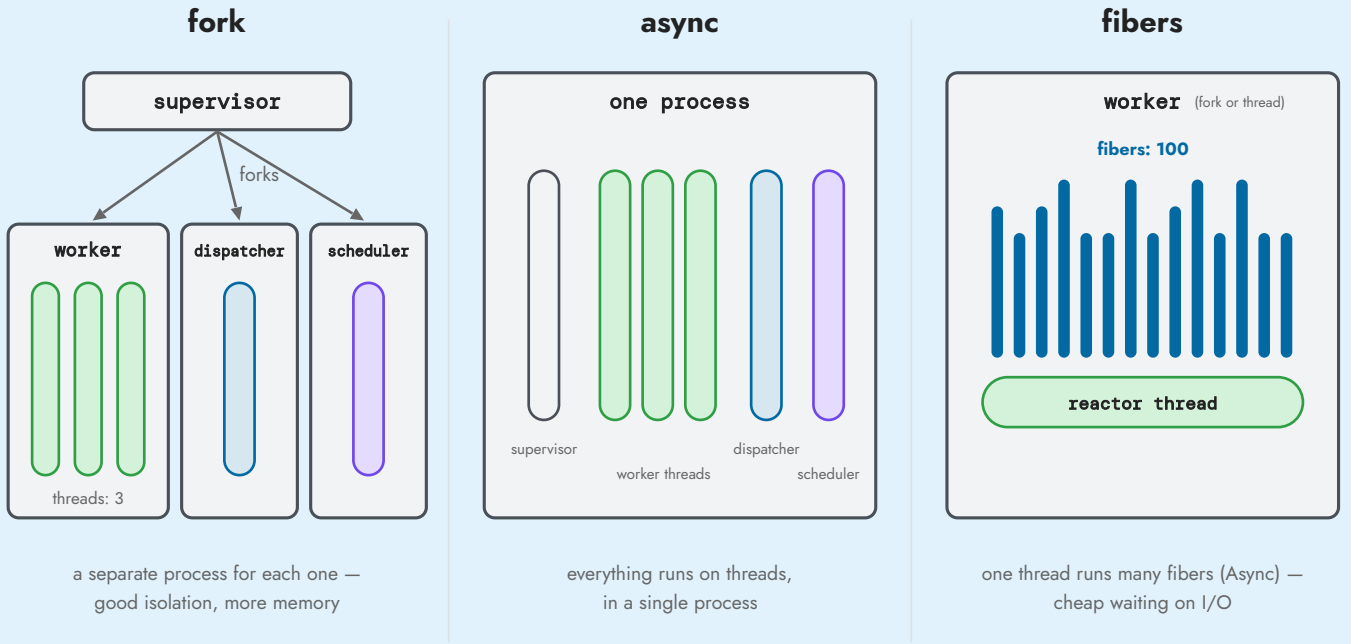
everything runs on threads,
in a single process

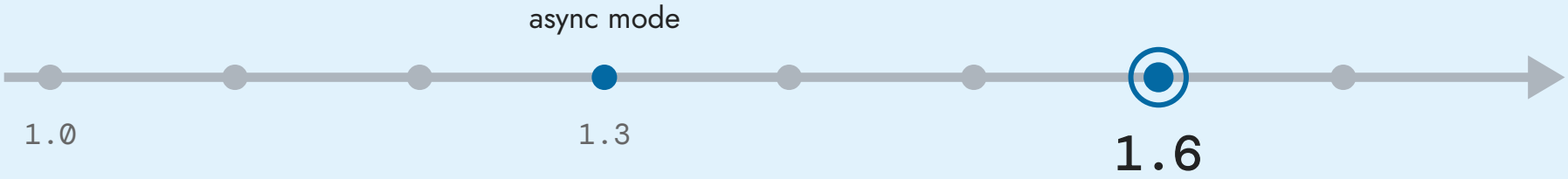
fibers



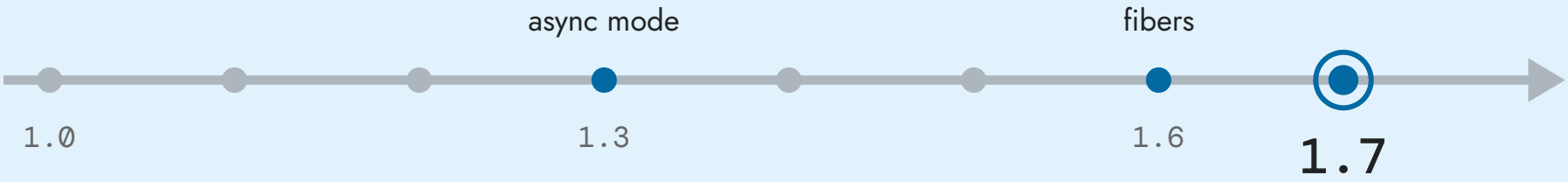
one thread runs many fibers (Async) —
cheap waiting on I/O

Fibers

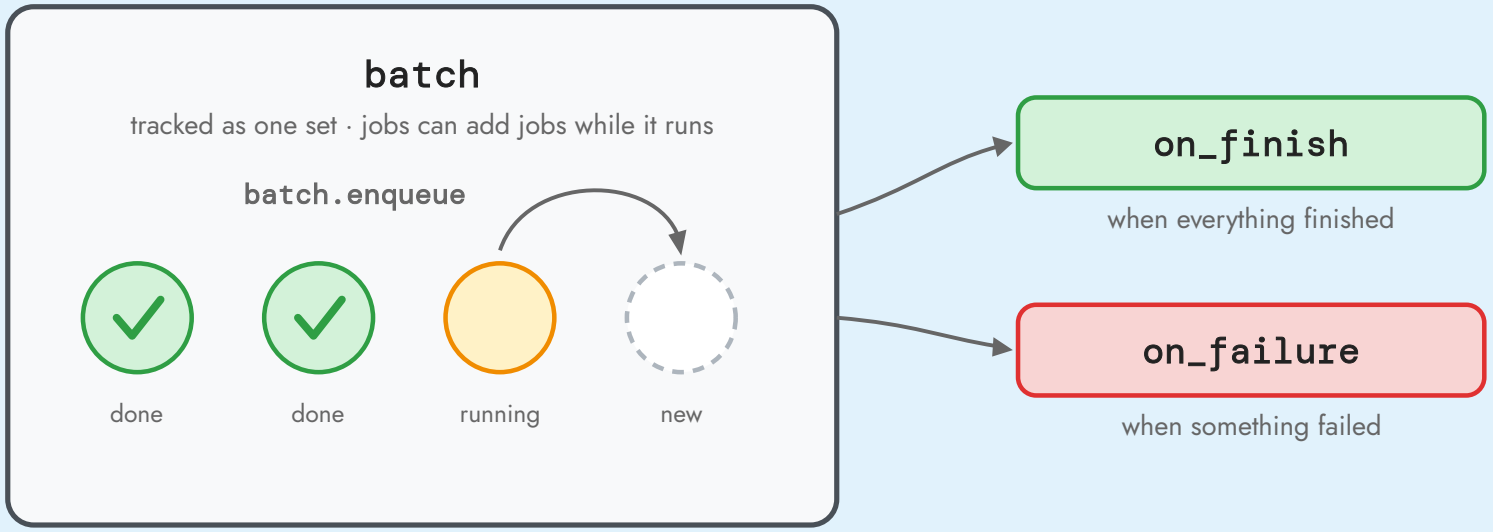

fork
supervisor
forks
worker
dispatcher
scheduler
threads: 3
a separate process for each one —
good isolation, more memory
async
one process
supervisor
worker threads
dispatcher
scheduler
everything runs on threads,
in a single process
fibers
worker (fork or thread)
fibers: 100
reactor thread
one thread runs many fibers (Async) —
cheap waiting on I/O

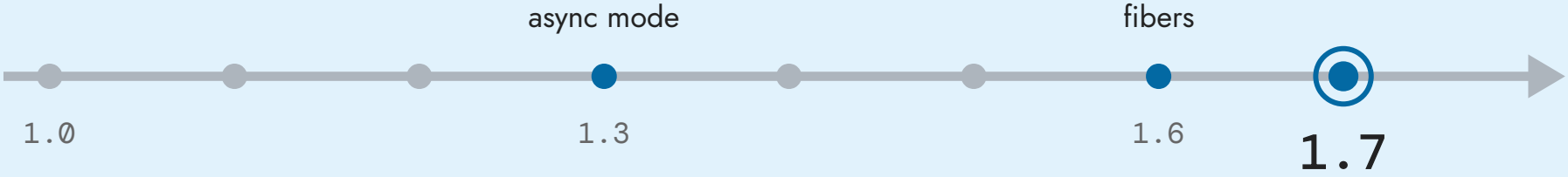

async mode
1.0
1.3
1.6

Batches



Batches


batch
tracked as one set · jobs can add jobs while it runs
batch.enqueue
done
done
running
new
on_finish
when everything finished
on_failure
when something failed


async mode
fibers
1.0
1.3
1.6
1.7

Batches



@jpcamara

2025

2026

PR #142

opened 2 Feb 2024 — @jpcamara

async mode

fibers

1.0

1.3

1.6

1.7

Batches



@jpcamara

when will you merge this

please merge it

PR #142

opened 2 P

the big reshape

2026

we need this!

any updates?

async mode

fibers

1.0

1.3

1.6

1.7

Batches



@jpcamara

2025

API design back and forth

"is this ready yet?"

PR #142

opened 2 Feb 2024 — @jpcamara

async mode

fibers

1.0 1.3 1.6 1.7

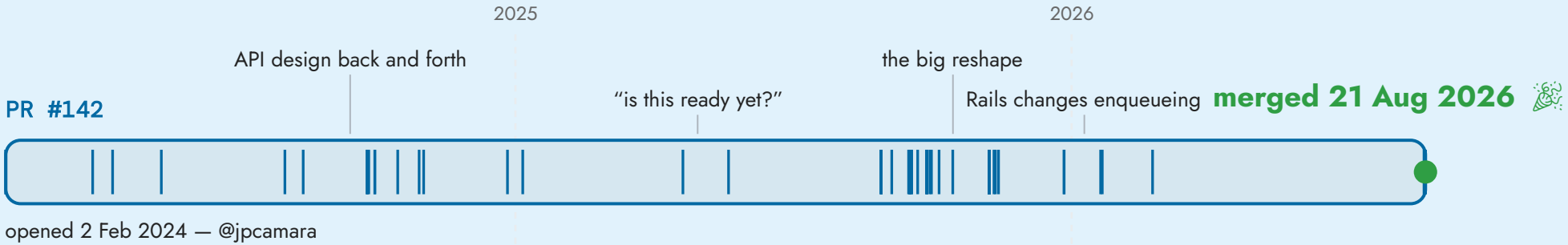
Batches



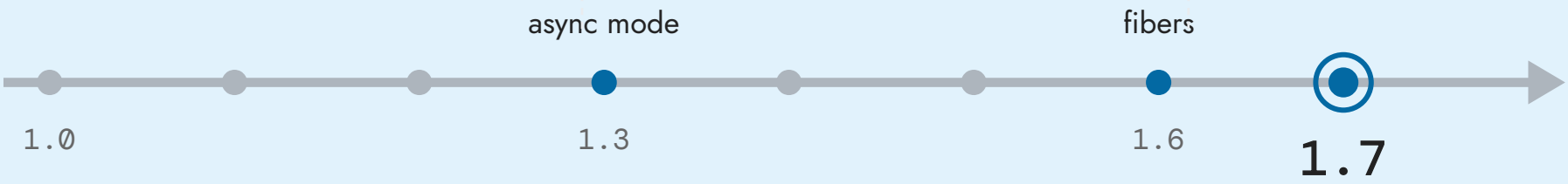




@jpcamara



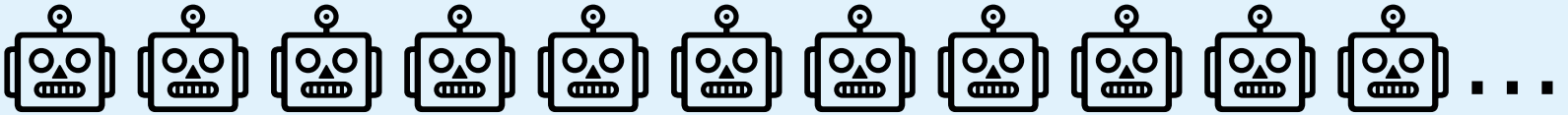
2 years, 6 months and 19 days later



On the new reality…



- Taste still matters (correct vs *right*)

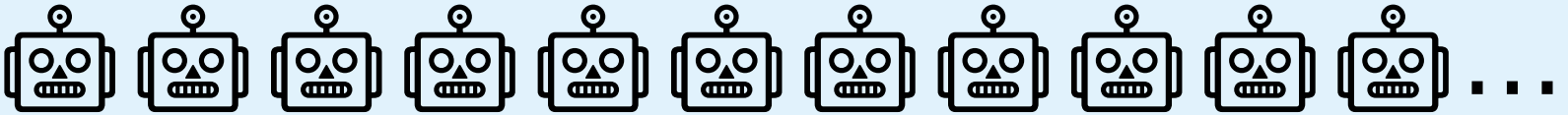




On the new reality…

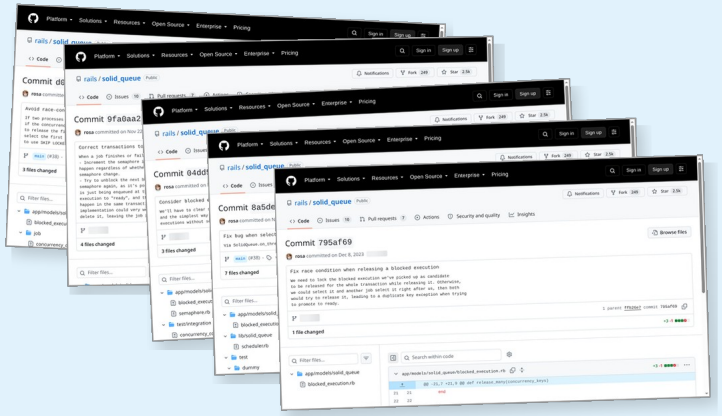


- Taste still matters (correct vs *right*)
- Understanding your code still (mostly) matters

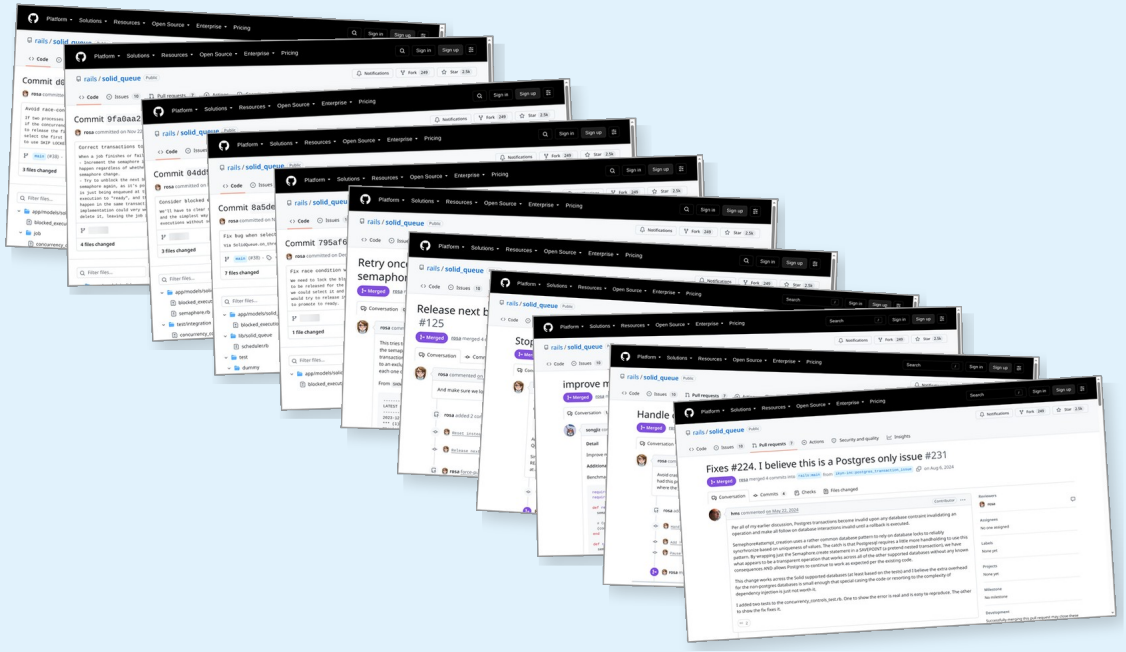




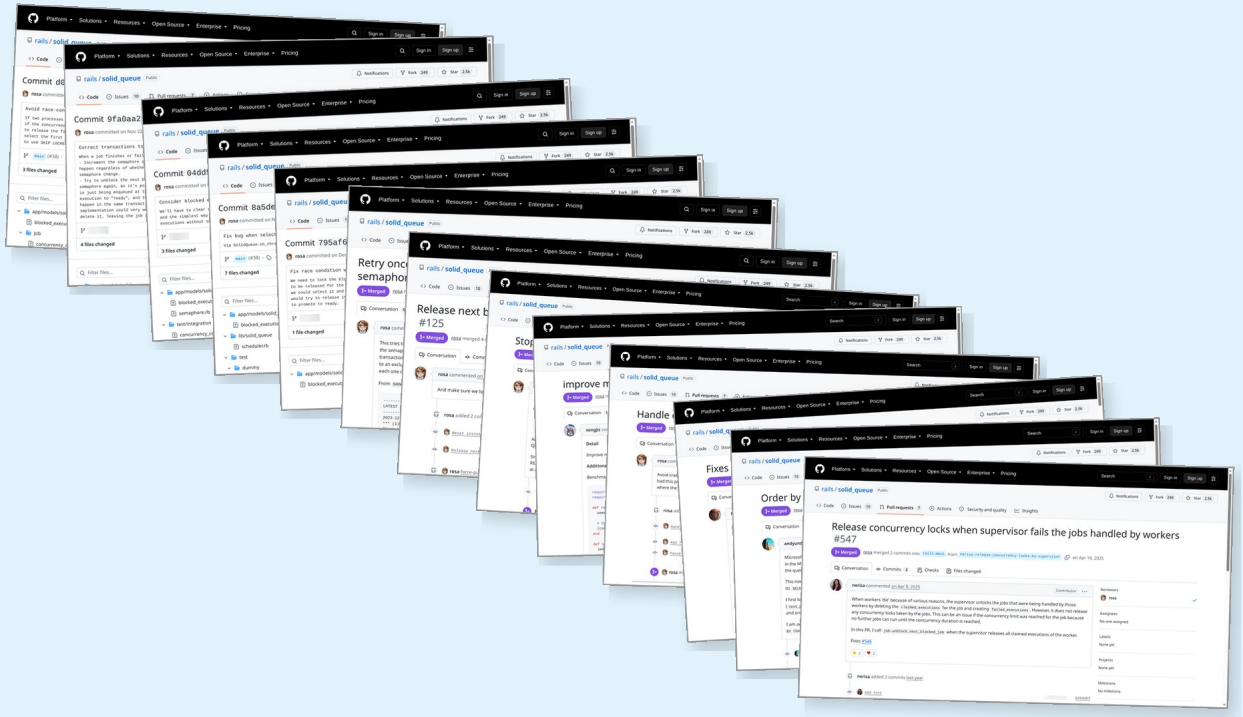
My most regretted feature 💔



My most regretted feature 💔



My most regretted feature 💔



My most regretted feature 💔

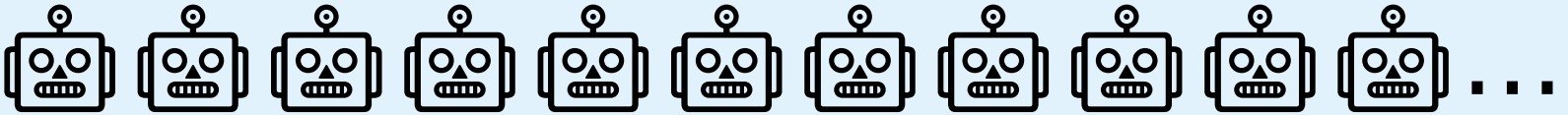


is also one of the
most Fun™ tested ones 🥹

On the new reality…

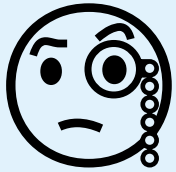


- Taste still matters
- Understanding your code still (mostly) matters
- Production is the definitive test



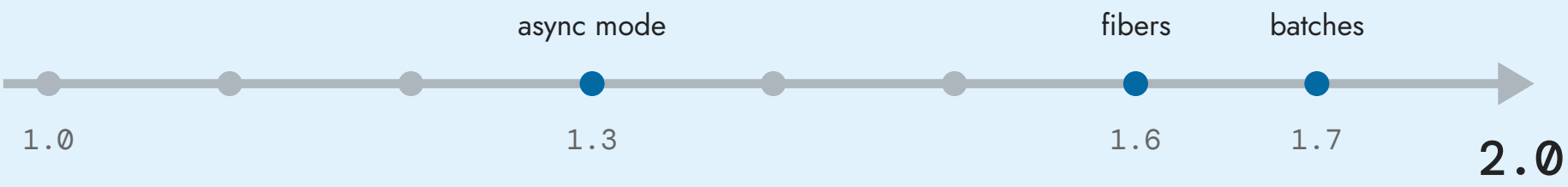


“Talk is cheap.
Show me the code.”

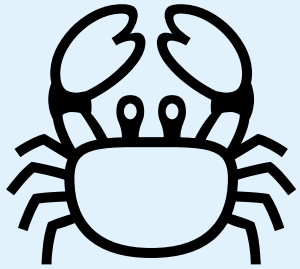
“*Code* is cheap. 
Show me *production*.”

“*Code* is cheap. 
Show me *your uptime*.”

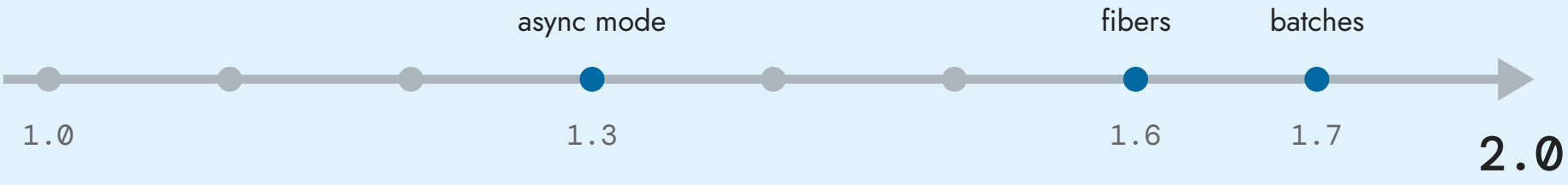
The path to 2.0 SOON



Sharding 

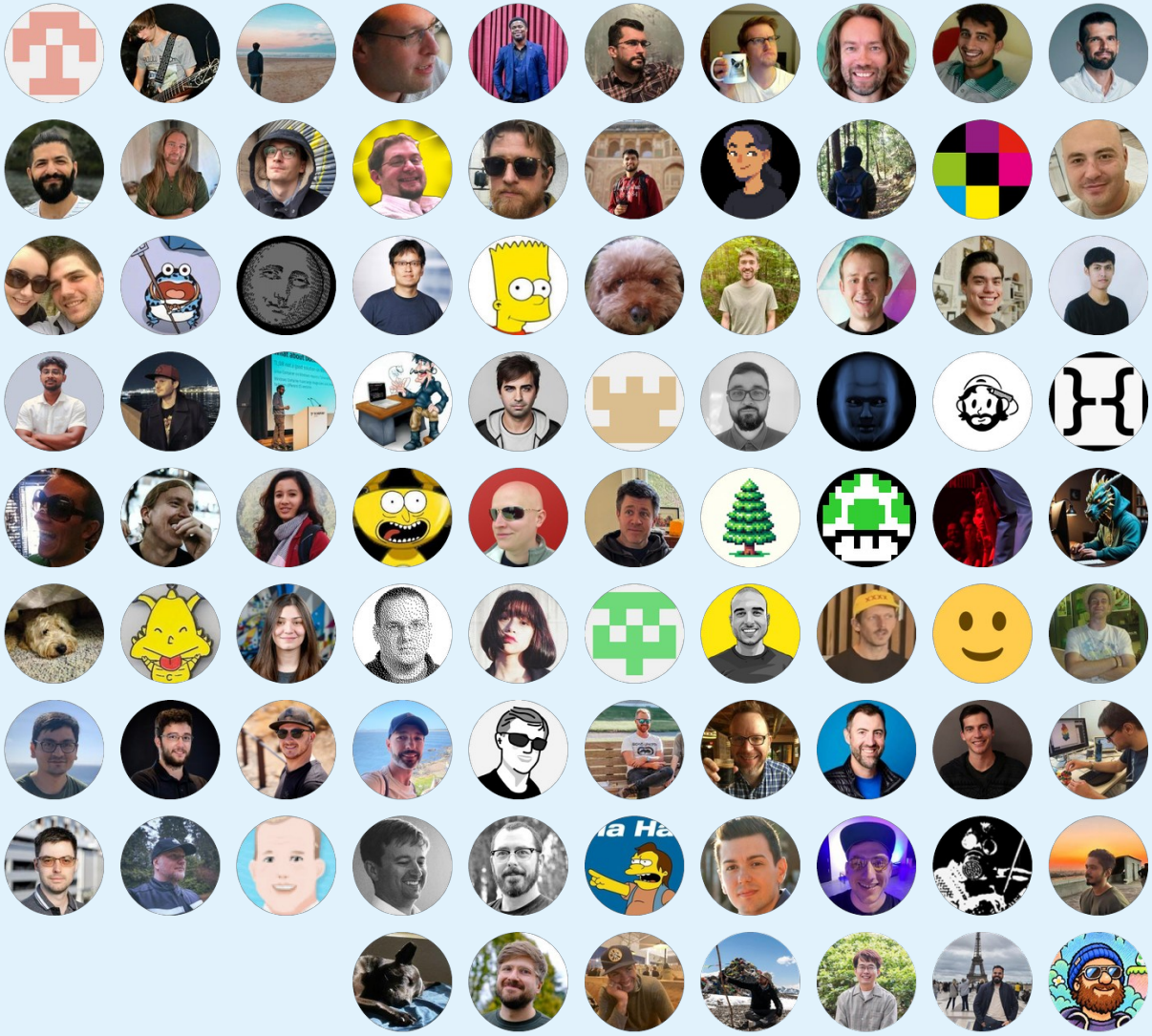


- Horizontal scaling
- A single DB has a ceiling
- Different from multi-tenancy



Onward





Onward

Thank you!

Slides & more

