RABBITMQ IS MORE THAN A SIDEKIQ REPLACEMENT

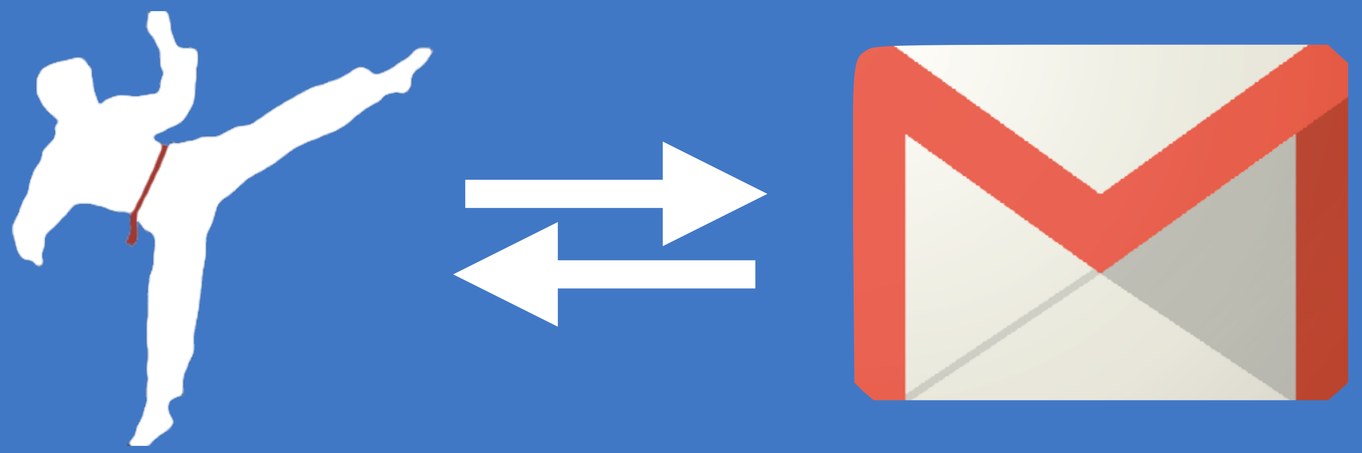
Stanko Krtalic Rusendic

github.com/stankec
@monorkin
hey@stanko.io
stanko.io

01

WORKERS

Asynchronous execution



???

```ruby
require_relative 'chat'

class App < Roda
  Chat.start_heartbeat

  plugin :streaming
  plugin :render, engine: 'slim'

  route do |r|
    r.root do
      view('root', layout: false)
    end

    r.get 'stream' do
      response['Content-Type'] = 'text/event-stream;charset=UTF-8'
      user_queue = Chat.subscribe
      callback = proc { Chat.unsubscribe(user_queue) }
      stream(loop: true, callback: callback) do |out|
        Chat.stream(out, user_queue)
      end
    end

    r.on 'messages' do
      r.post do
        name = r.params['name']
        message = r.params['message']
        Chat.message(name, message)
      end
    end
  end
end
```


"demo/01-simple_job_queue/app.rb" 33L, 684C written

➜ demo (master) ✗

```ruby
require_relative 'chat'

class App < Roda
  JOBS = Queue.new
  Thread.new do
    loop do
      JOBS.pop.call
    end
  end

  Chat.start_heartbeat

  plugin :streaming
  plugin :render, engine: 'slim'

  route do |r|
    r.root do
      view('root', layout: false)
    end

    r.get 'stream' do
      response['Content-Type'] = 'text/event-stream;charset=UTF-8'
      user_queue = Chat.subscribe
      callback = proc { Chat.unsubscribe(user_queue) }
      stream(loop: true, callback: callback) do |out|
        Chat.stream(out, user_queue)
      end
    end

    r.on 'messages' do
      r.post do
        name = r.params['name']
        message = r.params['message']
        if message.match?(%r{^\\broadcast\s+.*$})
          message = message.scan(%r{^\\broadcast\s+(.*)$}).flatten.first
          JOBS << proc do
            Chat.broadcast(message)
          end
        else
          Chat.message(name, message)
        end
```

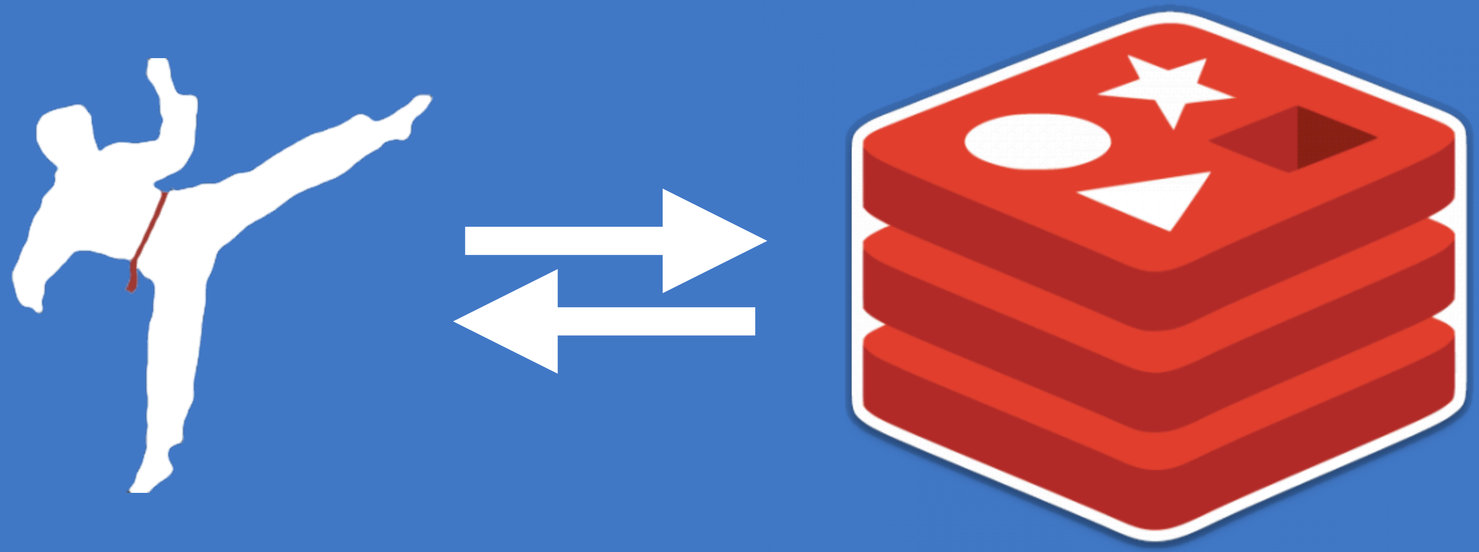
N… demo/01-simple_job_queue/app.rb rub… 76% ≡ 36/47 ㏑ : 33

```
^CGracefully stopping... (press Ctrl+C again to force)
Stopping demo_demo-01_1 ...
Killing demo_demo-01_1 ... done
➜ demo (master) ✗
```

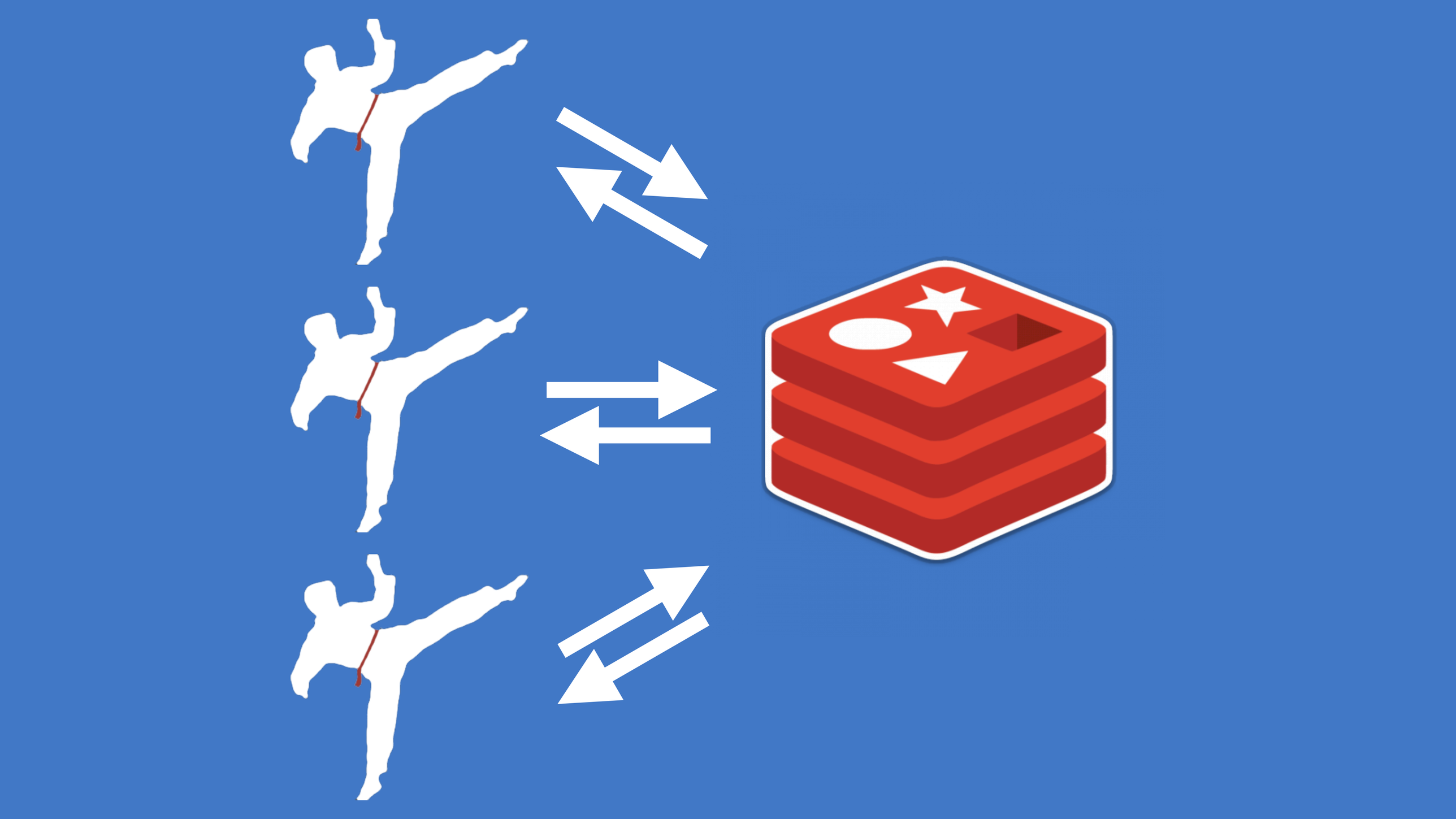
Downsides?

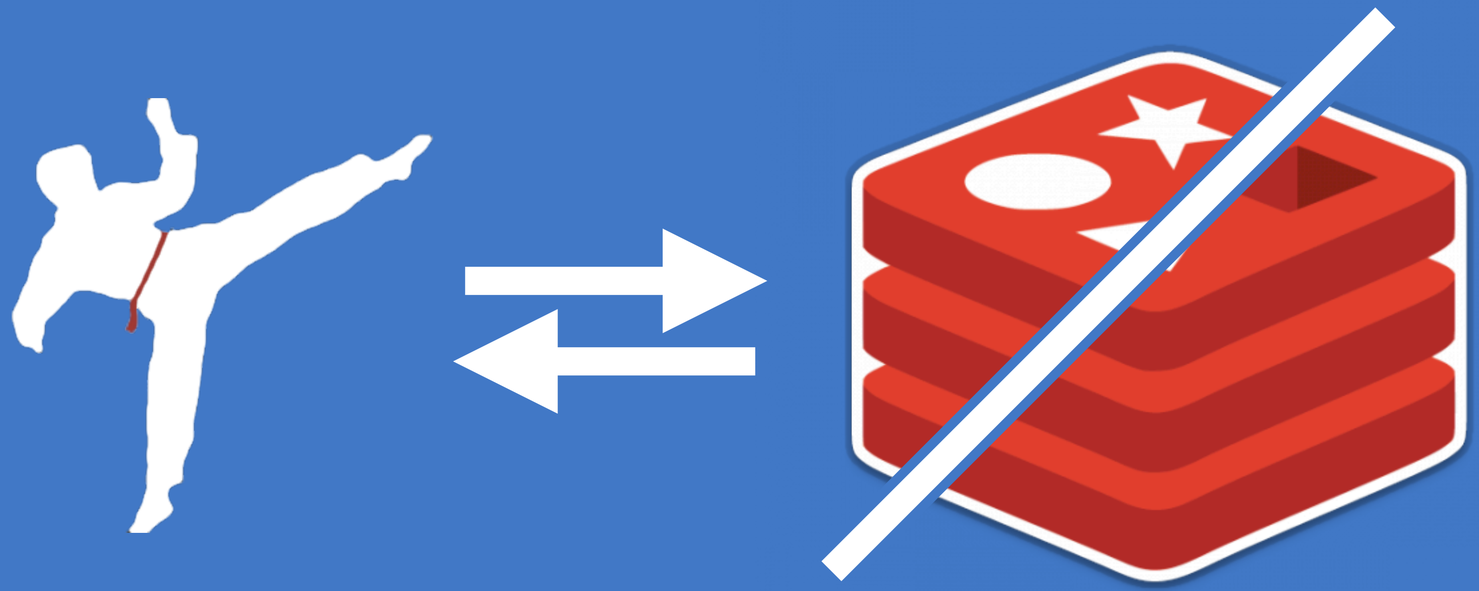
Debugging

Load balancing



```
127.0.0.1:6379>
127.0.0.1:6379> KEYS *
1) "stat:failed:push_notifs"
2) "dead"
3) "stat:failed"
4) "retry"
127.0.0.1:6379> ZRANGE retry 0 -1
1) "{\"retry_count\":10,\"retried_at\":1519677868,\"queue\":\"push_notifs\",\"max_retry_count\":10,\"jid\":\"374075687611975539 00\",\"finished_at\":null,\"failed_at\":1519660887,\"error_message\":\"no match of right hand side value: nil\",\"error_backtrace\":\"    (papa_pal) lib/papa_pal/workers/push_apns.ex:9: PapaPal.Workers.PushApns.perform/3\\n    (verk) lib/verk/worker.ex:35: Verk.Worker.handle_cast/2\\n    (stdlib) gen_server.erl:616: :gen_server.try_dispatch/4\\n    (stdlib) gen_server.erl:686: :gen_server.handle_msg/6\\n    (stdlib) proc_lib.erl:247: :proc_lib.init_p_do_apply/3\\n\",\"enqueued_at\":1519660886,\"class\":\"PapaPal.Workers.PushApns\",\"args\":[\"900185ef-5bc3-4481-ad0c-fb1ad4637d2a\",\"Remember to complete your visit!\",\"{\\\"payload\\\":null}\"]}"
127.0.0.1:6379>
```



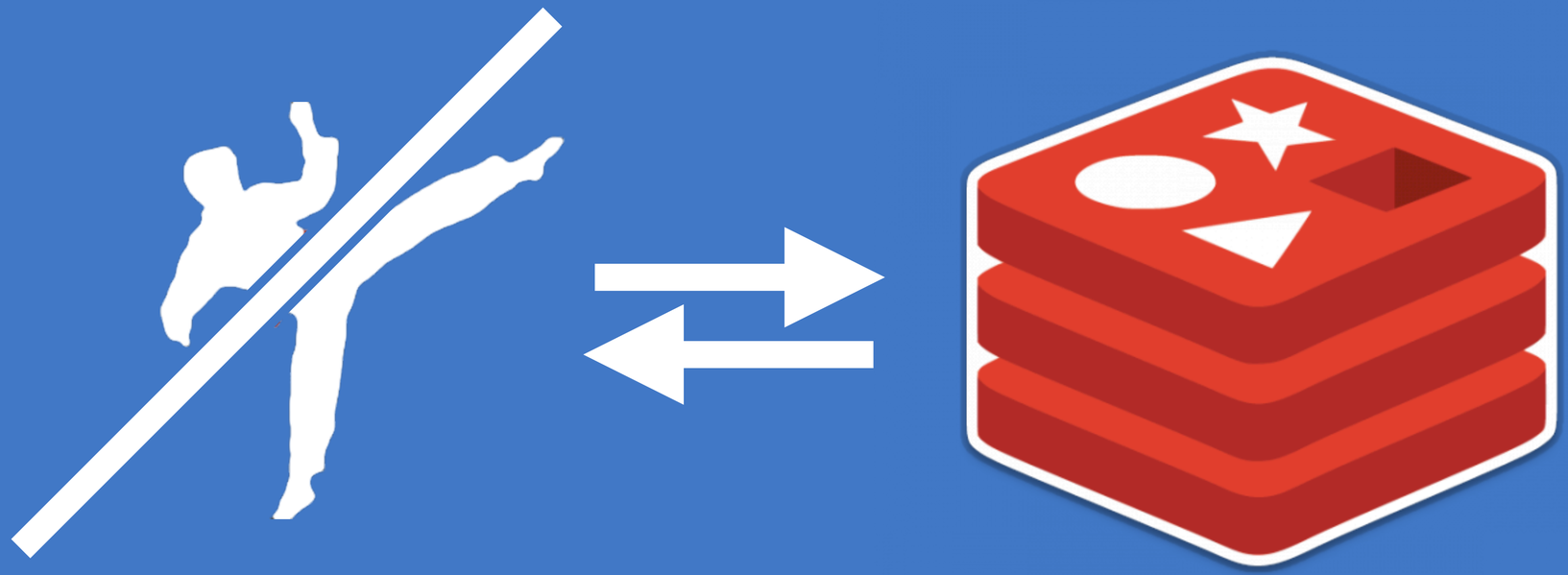


RDB disadvantages

- RDB is NOT good if you need to minimize the chance of data loss in case Redis stops working (for example after a power outage). You can configure different *save points* where an RDB is produced (for instance after at least five minutes and 100 writes against the data set, but you can have multiple save points). However you'll usually create an RDB snapshot every five minutes or more, so in case of Redis stopping working without a correct shutdown for any reason you should be prepared to lose the latest minutes of data.
- RDB needs to fork() often in order to persist on disk using a child process. Fork() can be time consuming if the dataset is big, and may result in Redis to stop serving clients for some millisecond or even for one second if the dataset is very big and the CPU performance not great. AOF also needs to fork() but you can tune how often you want to rewrite your logs without any trade-off on durability.

AOF disadvantages

- AOF files are usually bigger than the equivalent RDB files for the same dataset.
- AOF can be slower than RDB depending on the exact fsync policy. In general with fsync set to *every second* performances are still very high, and with fsync disabled it should be exactly as fast as RDB even under high load. Still RDB is able to provide more guarantees about the maximum latency even in the case of an huge write load.
- In the past we experienced rare bugs in specific commands (for instance there was one involving blocking commands like BRPOPLPUSH) causing the AOF produced to not reproduce exactly the same dataset on reloading. This bugs are rare and we have tests in the test suite creating random complex datasets automatically and reloading them to check everything is ok, but this kind of bugs are almost impossible with RDB persistence. To make this point more clear: the Redis AOF works incrementally updating an existing state, like MySQL or MongoDB does, while the RDB snapshotting creates everything from scratch again and again, that is conceptually more robust. However - 1) It should be noted that every time the AOF is rewritten by Redis it is recreated from scratch starting from the actual data contained in the data set, making resistance to bugs stronger compared to an always appending AOF file (or one rewritten reading the old AOF instead of reading the data in memory). 2) We never had a single report from users about an AOF corruption that was detected in the real world.



```ruby
class TestWorker
  include Sidekiq::Worker

  def perform(*args)
    # Do something
  rescue
    # Do something else
  end
end
```

```ruby
# frozen_string_literal: true

require_relative 'demo'
include Demo

begin
  Demo.halt_and_catch_fire
rescue
  puts 'All is good'
ensure
  puts "The VM didn't crash this time"
end
```

Process Crashes

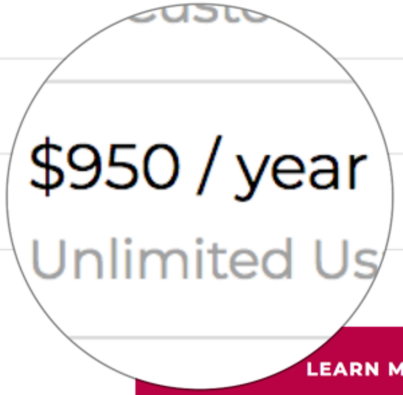
If the Sidekiq process segfaults or crashes the Ruby VM, any jobs that were being processed are lost. Sidekiq Pro offers a reliable queueing feature which does not lose those jobs.

Sidekiq

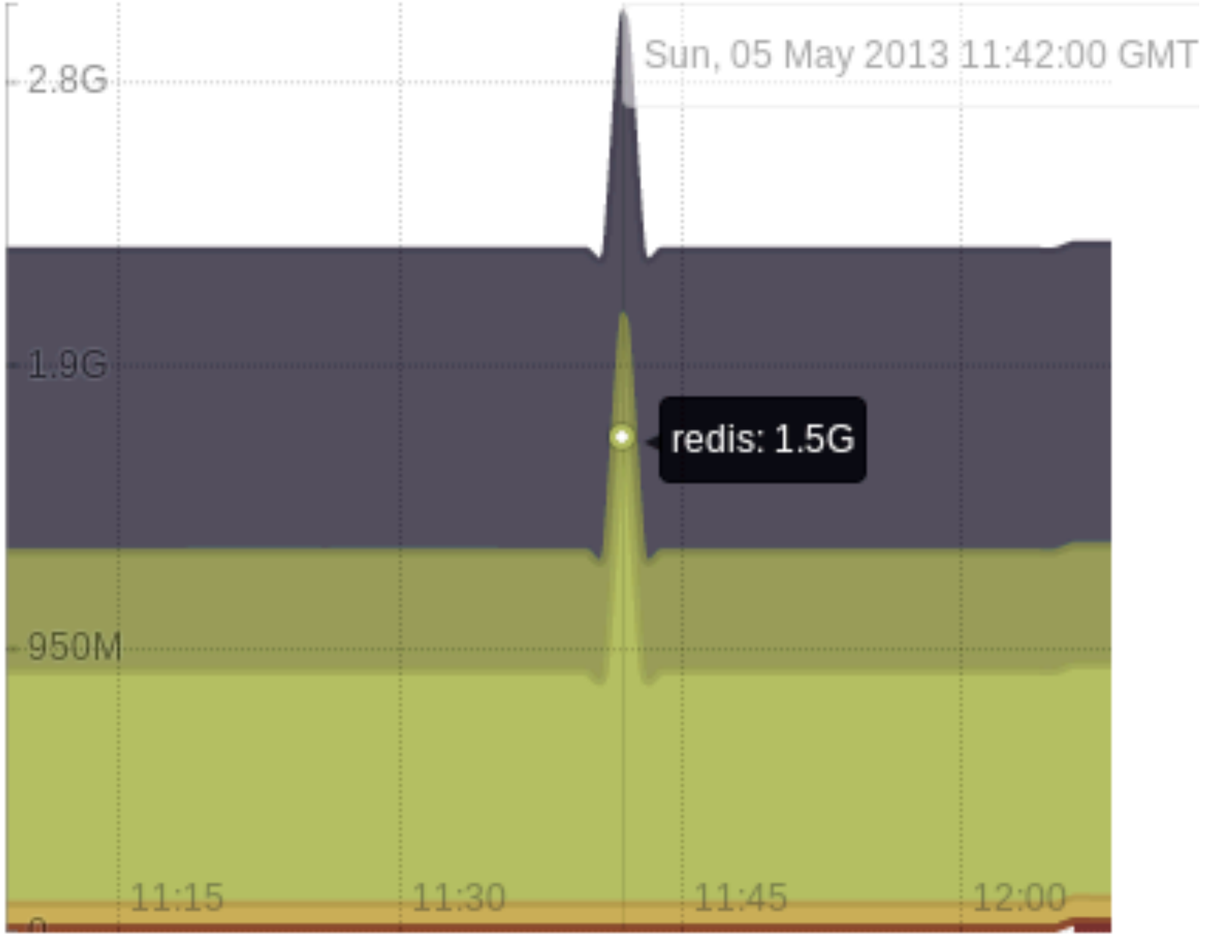
Simple, efficient background processing for Ruby.

PRO ENTERPRISE SUPPORT ABOUT

	OSS	PRO	ENTERPRISE
AUDIENCE	Hobbyists	Small Business	Med–Large Companies
DEDICATED SUPPORT	None		Email
LICENSE	LGPL		Commercial with Custom Terms
PRICING	Free		$1950 / year per 100 Workers
PURCHASING	n/a		Credit Card, Invoice
	GET STARTED	LEARN MORE ›	LEARN MORE ›
		BUY ›	BUY ›

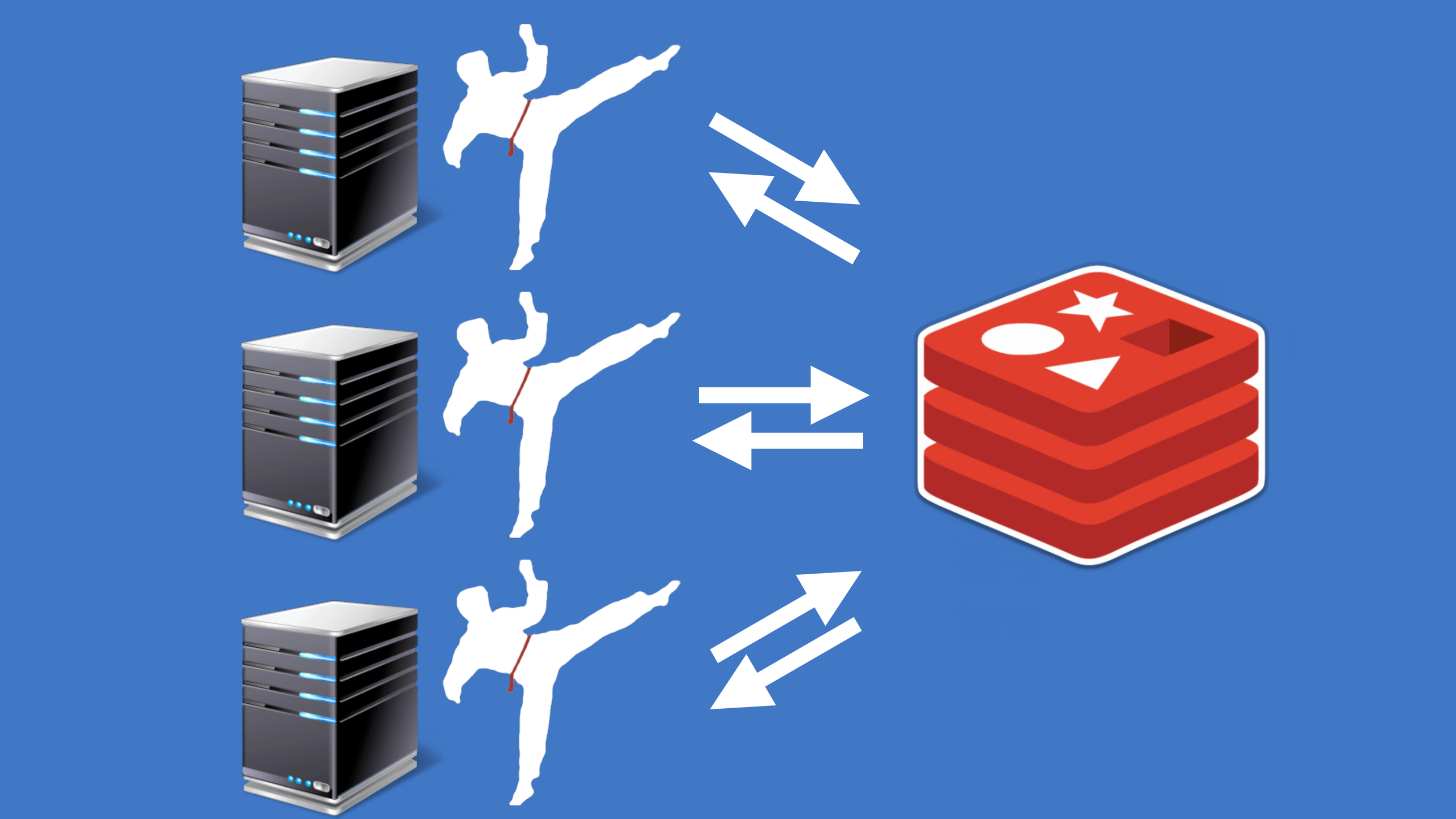


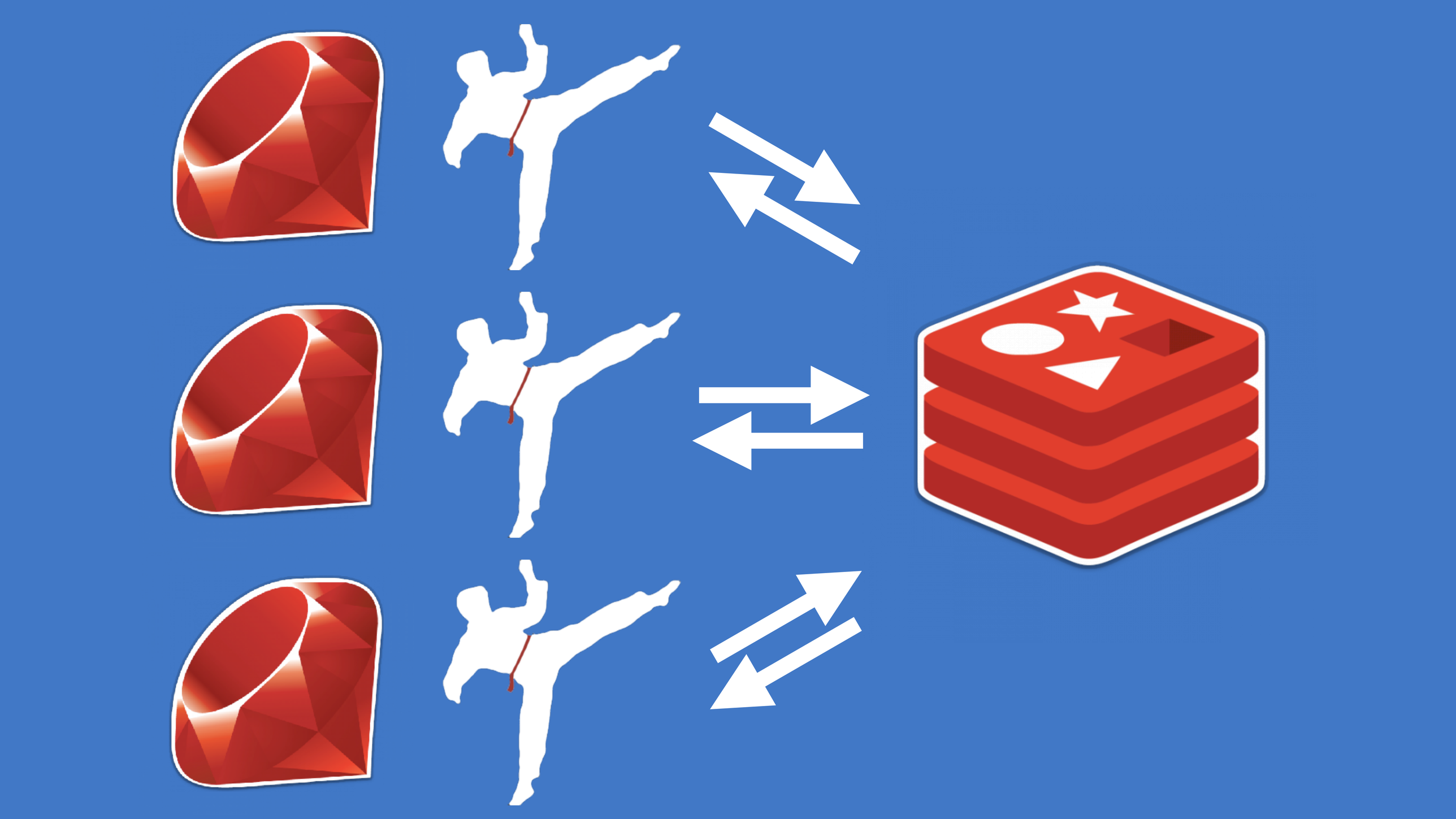
proc mem prod

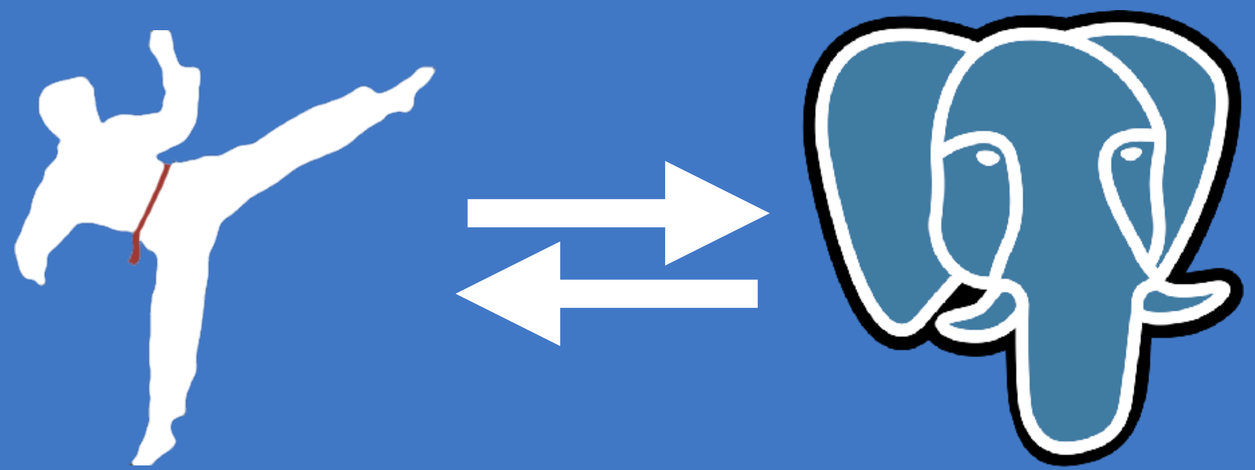


main process memory usage on production server

???







02

RABBITMQ


RabbitMQ

jondot / sneakers

A fast background processing framework for Ruby and RabbitMQ http://sneakers.io

394 commits 9 branches 19 releases 66 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

michaelklishin committed on Jan 18 Merge pull request #335 from kke/patch-1 ... Latest commit c647e34 on Jan 18

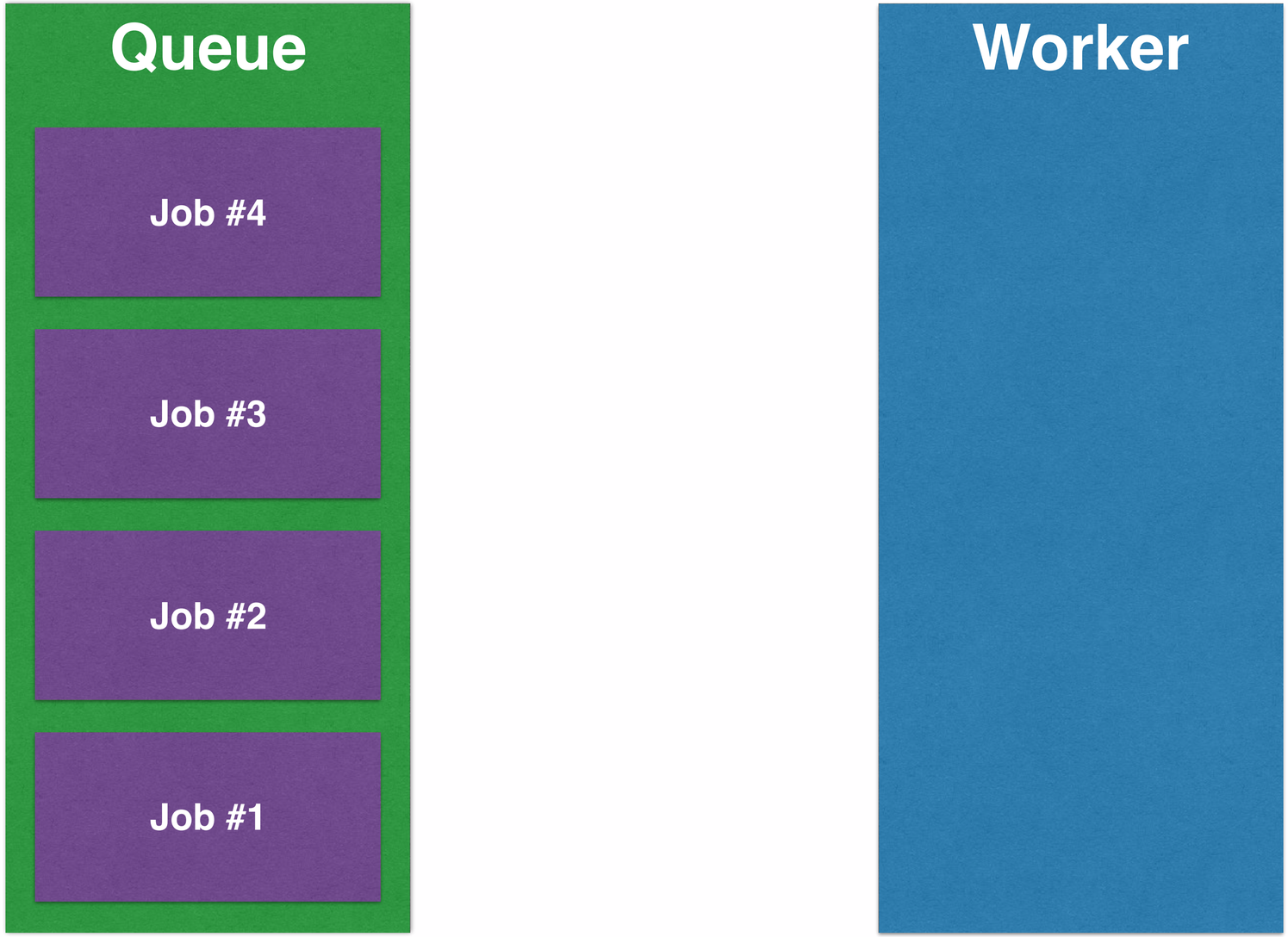
bin	Allow setting logger and provide server configuration convenience block.	3 years ago
examples	Indicate that worker is a background task for NewRelic	11 months ago
lib	Merge branch 'fix_connected_session_bug' of https://github.com/webit-...	4 months ago
scripts	Add a fat and slim docker builds	2 years ago
spec	Wording	4 months ago
.gitignore	Maxretry Handler	4 years ago
.travis.yml	Test against the same set of Rubies as Bunny	11 months ago
Dockerfile	Add a fat and slim docker builds	2 years ago

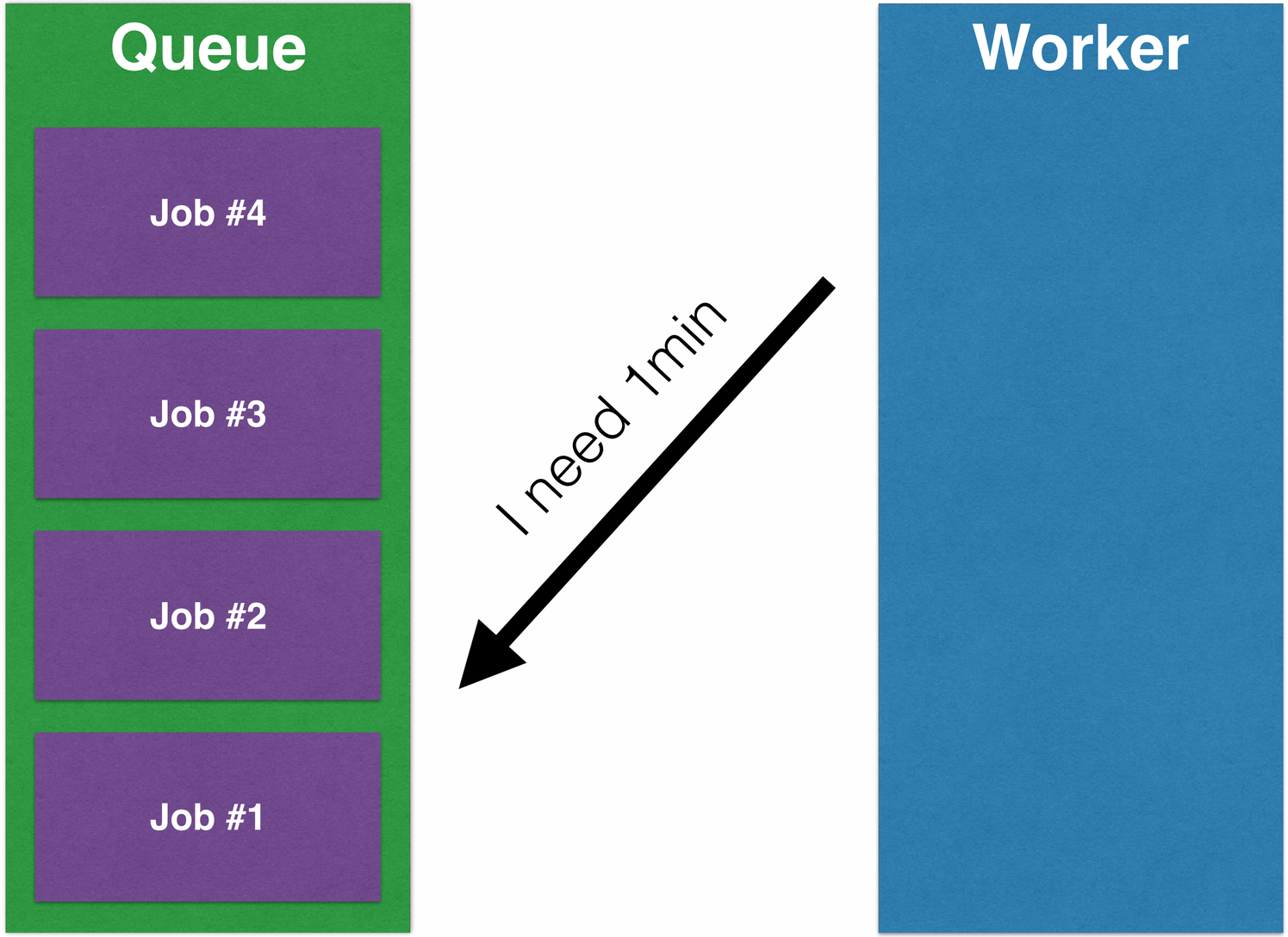
```
1  class TitleScraper
2    include Sneakers::Worker
3
4    from_queue 'downloads'
5
6    def work(msg)
7      title = extract_title(msg)
8      logger.info "FOUND <#{title}>"
9      ack!
10   end
11
12   private
13
14   def extract_title(html)
15     html.scan(/<title>(.*?)<\/title>/).flatten.first
16   end
17 end
```

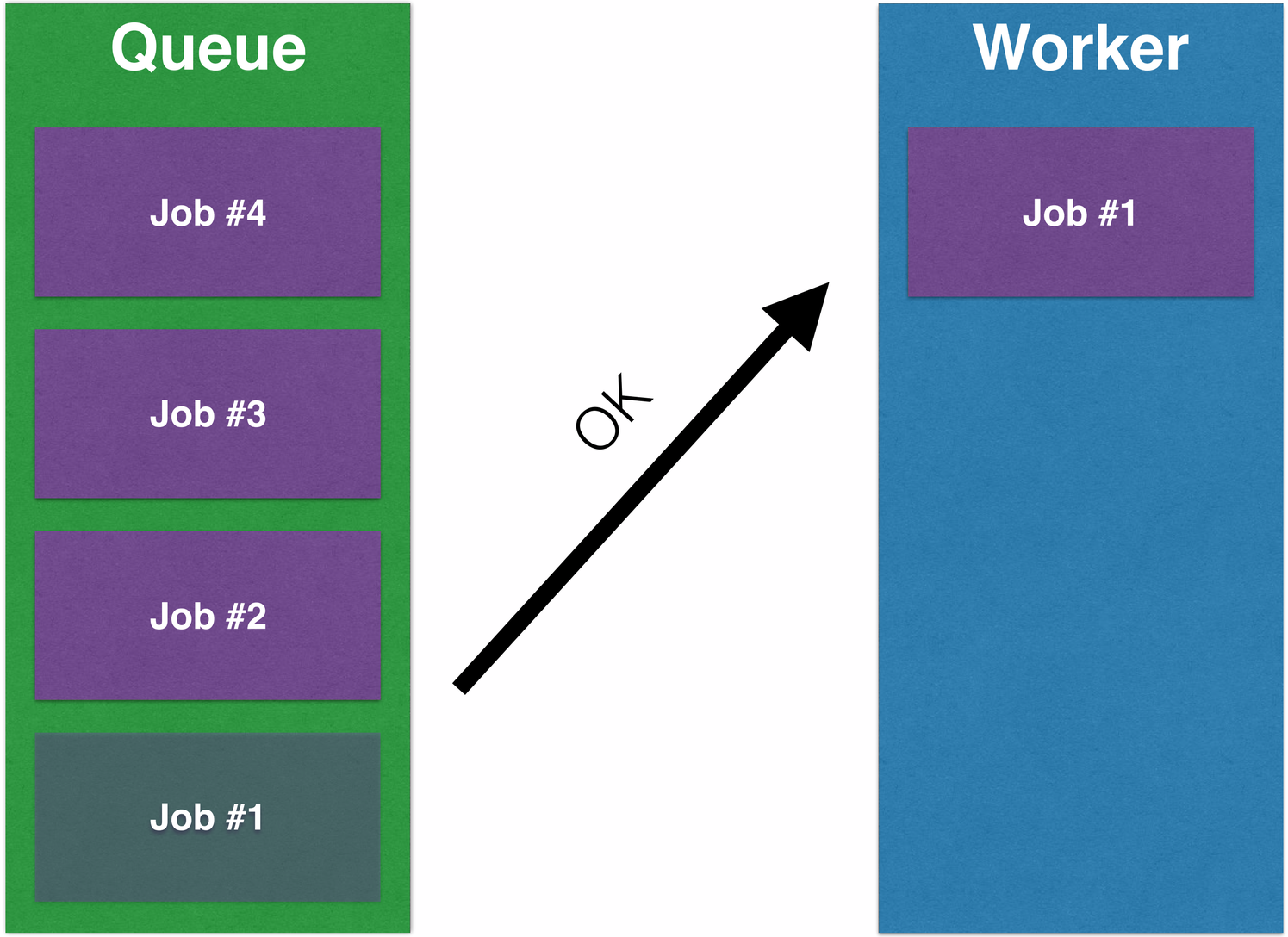
```
i  1  class TestWorker
   2    include Sidekiq::Worker
   3
   4    def perform(*args)
   5      # Do something
   6    end
   7  end
+  8
+  9
+  10
+  11
+  12
+  13
+  14
+  15
+  16
+  17
```

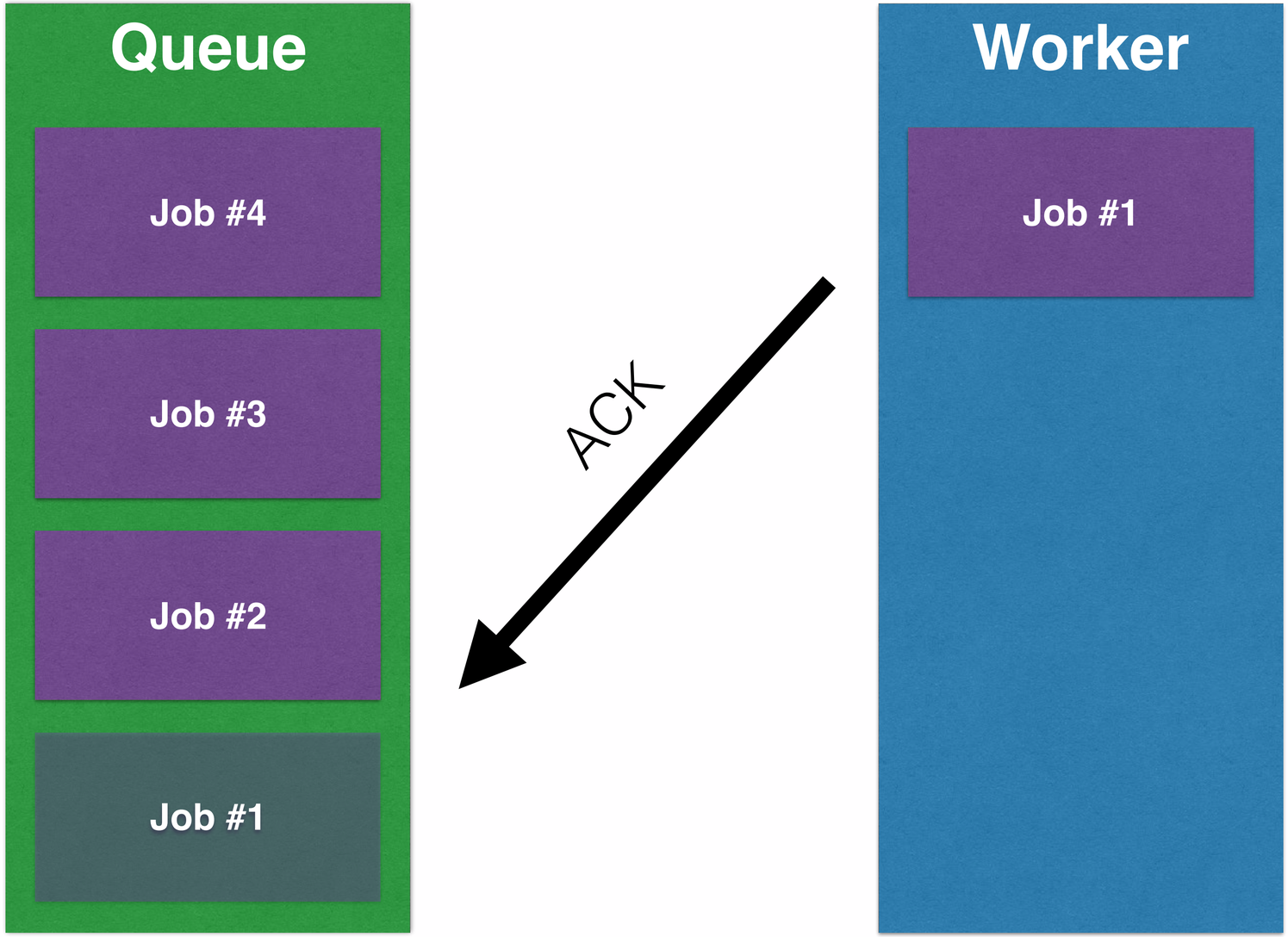
```ruby
class TitleScraper
  include Sneakers::Worker

  from_queue 'downloads'

  def work(msg)
    title = extract_title(msg)
    logger.info "FOUND <#{title}>"
    ack!
  end

  private

  def extract_title(html)
    html.scan(/<title>(.*?)<\/title>/).flatten.first
  end
end
```

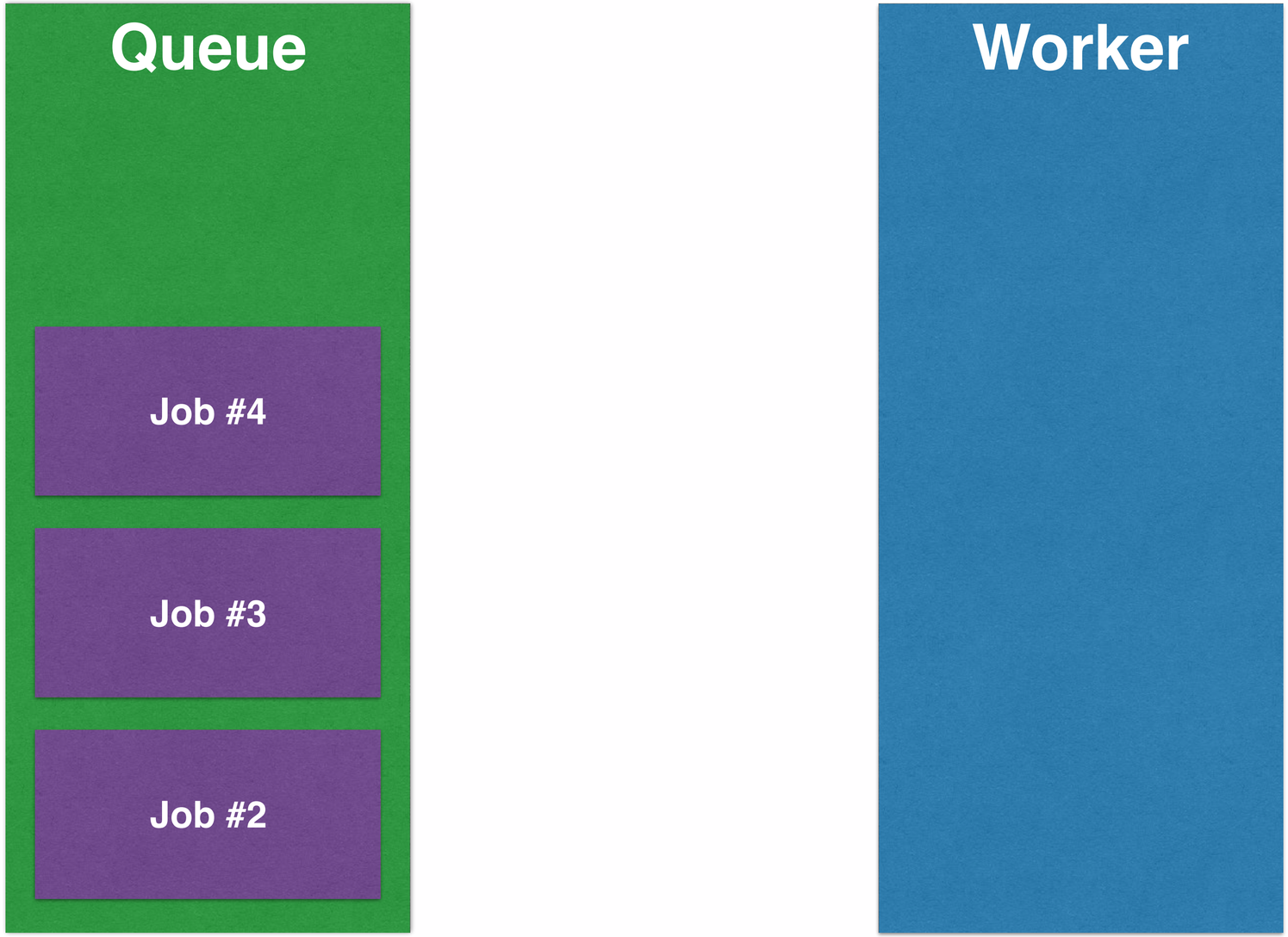
```ruby
class TitleScraper
  include Sneakers::Worker

  from_queue 'downloads'

  def work(msg)
    title = extract_title(msg)
    logger.info "FOUND <#{title}>"
    ack!
  end

  private

  def extract_title(html)
    html.scan(/<title>(.*?)<\/title>/).flatten.first
  end
end
```

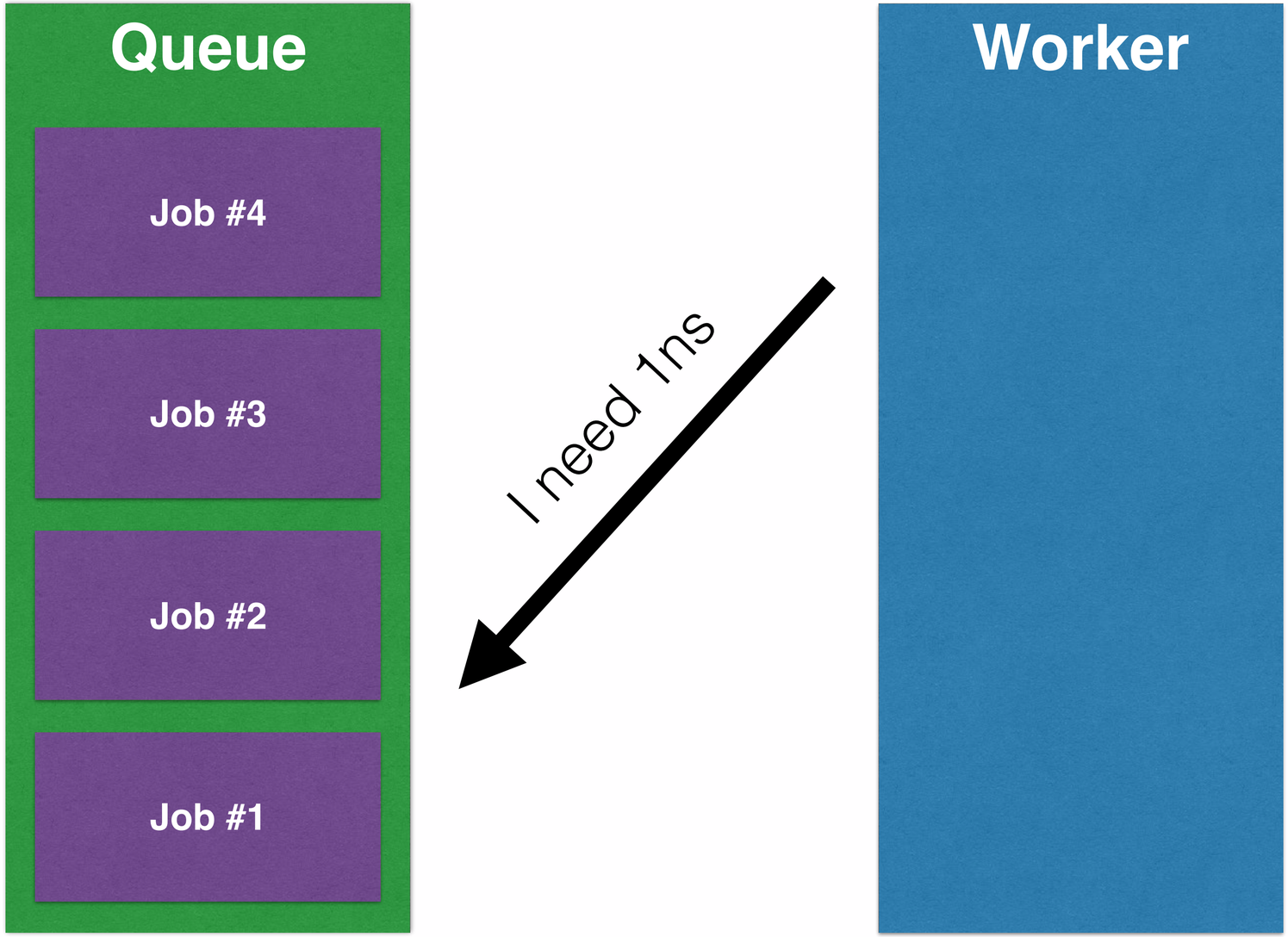

Queue
Job #4
Job #3
Job #2
Job #1
Worker

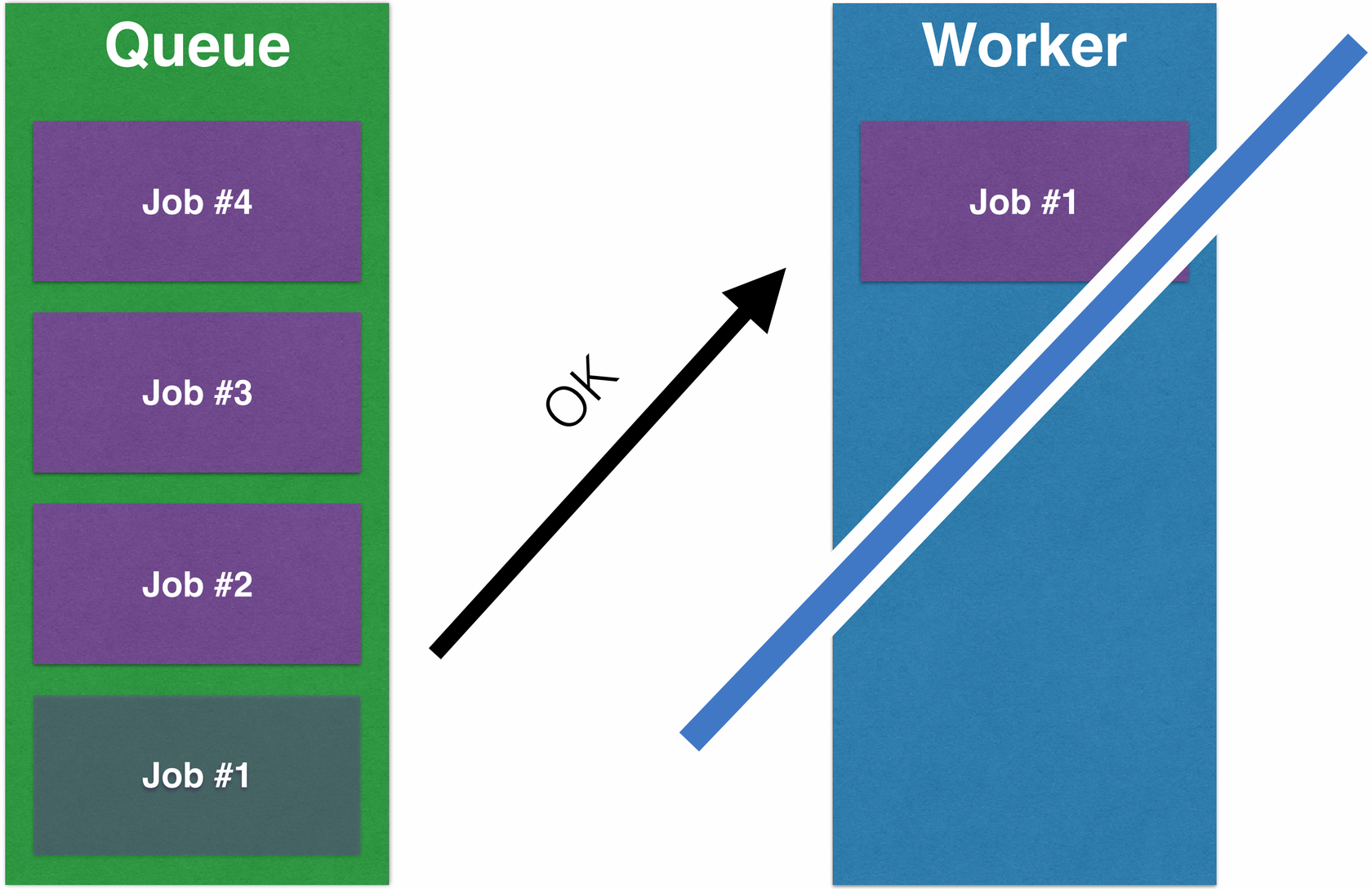

Queue
Job #4
Job #3
Job #2
Job #1
I need 1min
Worker

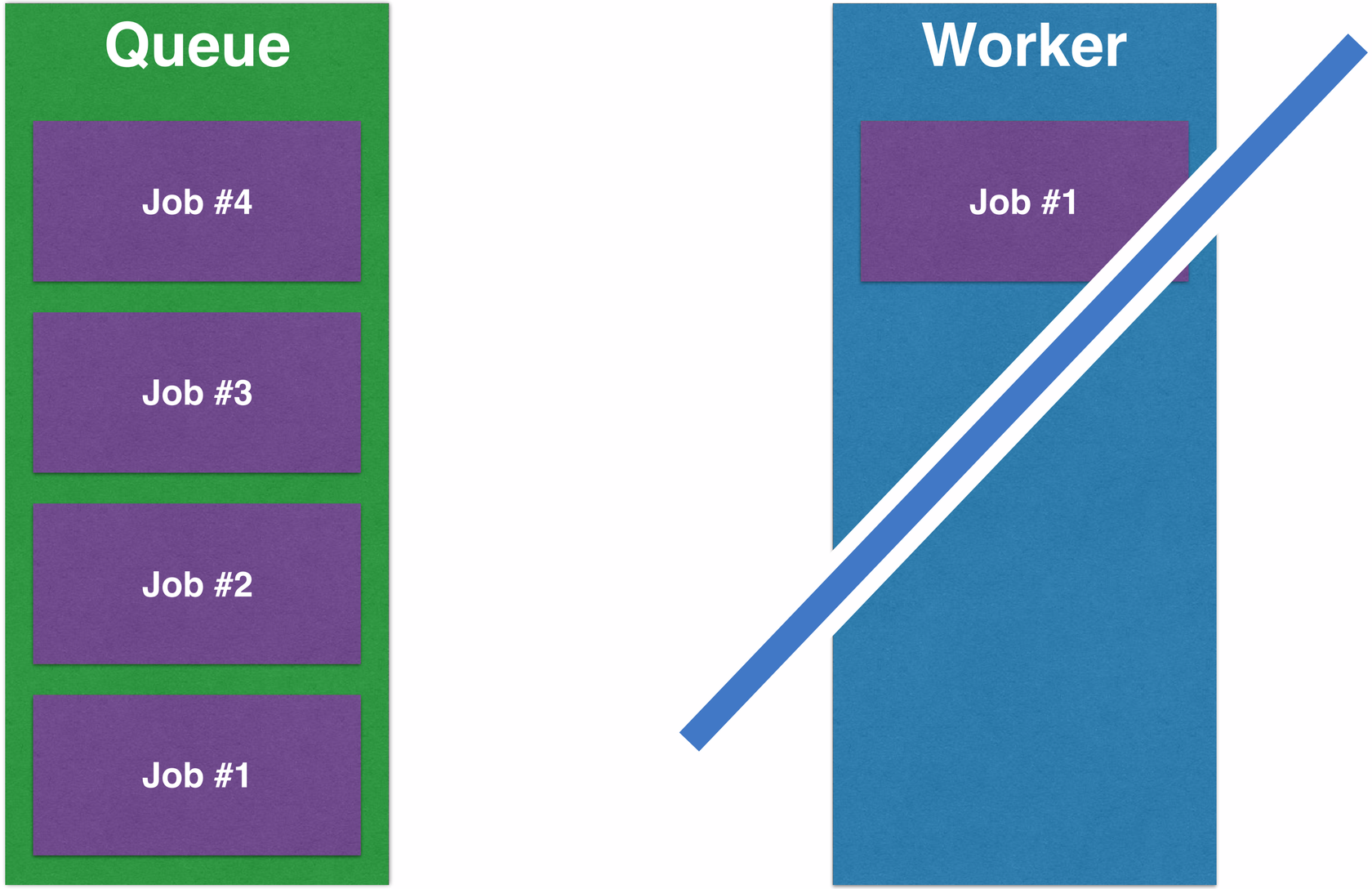

Queue
Job #4
Job #3
Job #2
Job #1
OK
Worker
Job #1

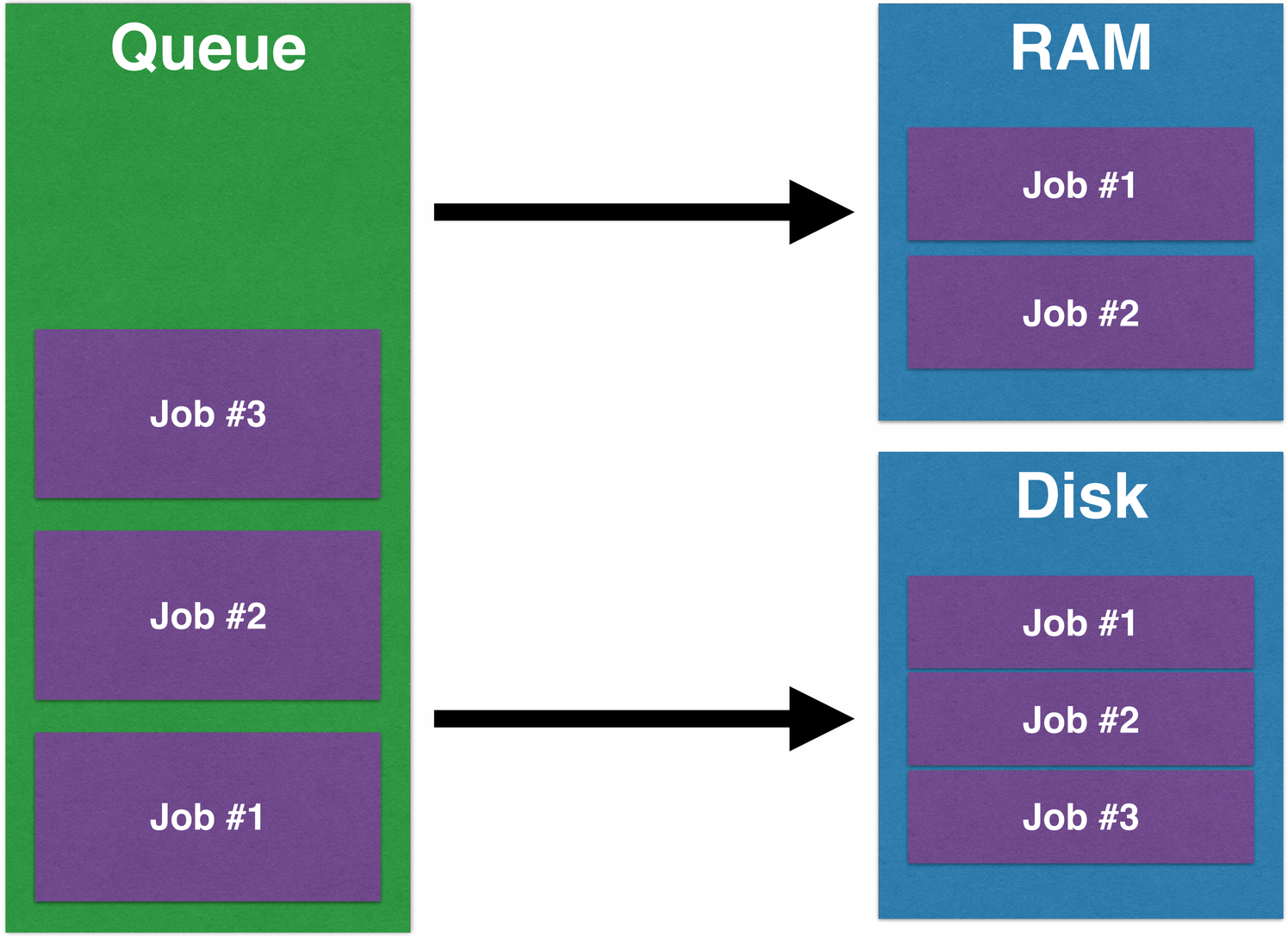

Queue
Job #4
Job #3
Job #2
Job #1
ACK
Worker
Job #1

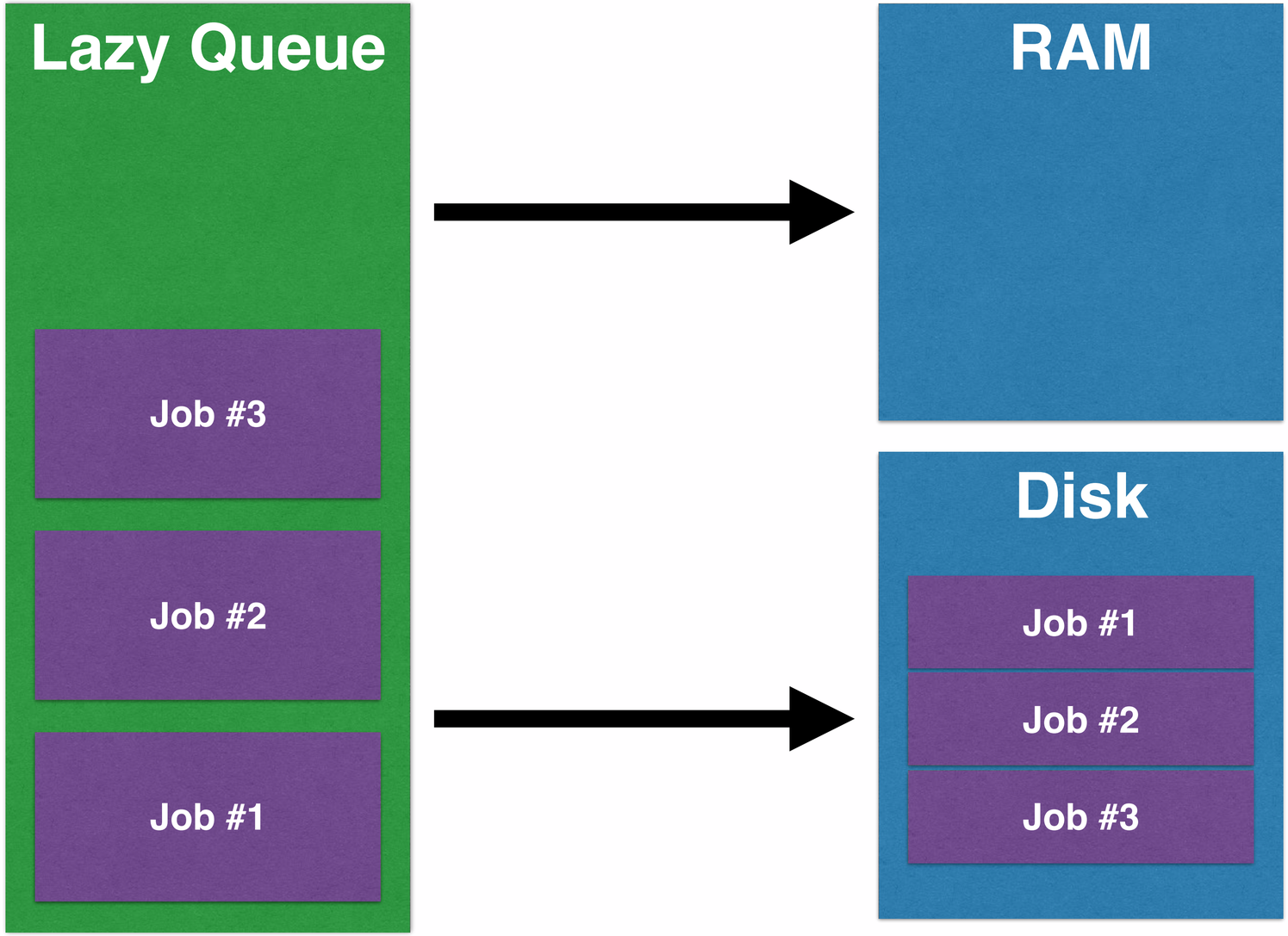

Queue
Job #4
Job #3
Job #2
Worker

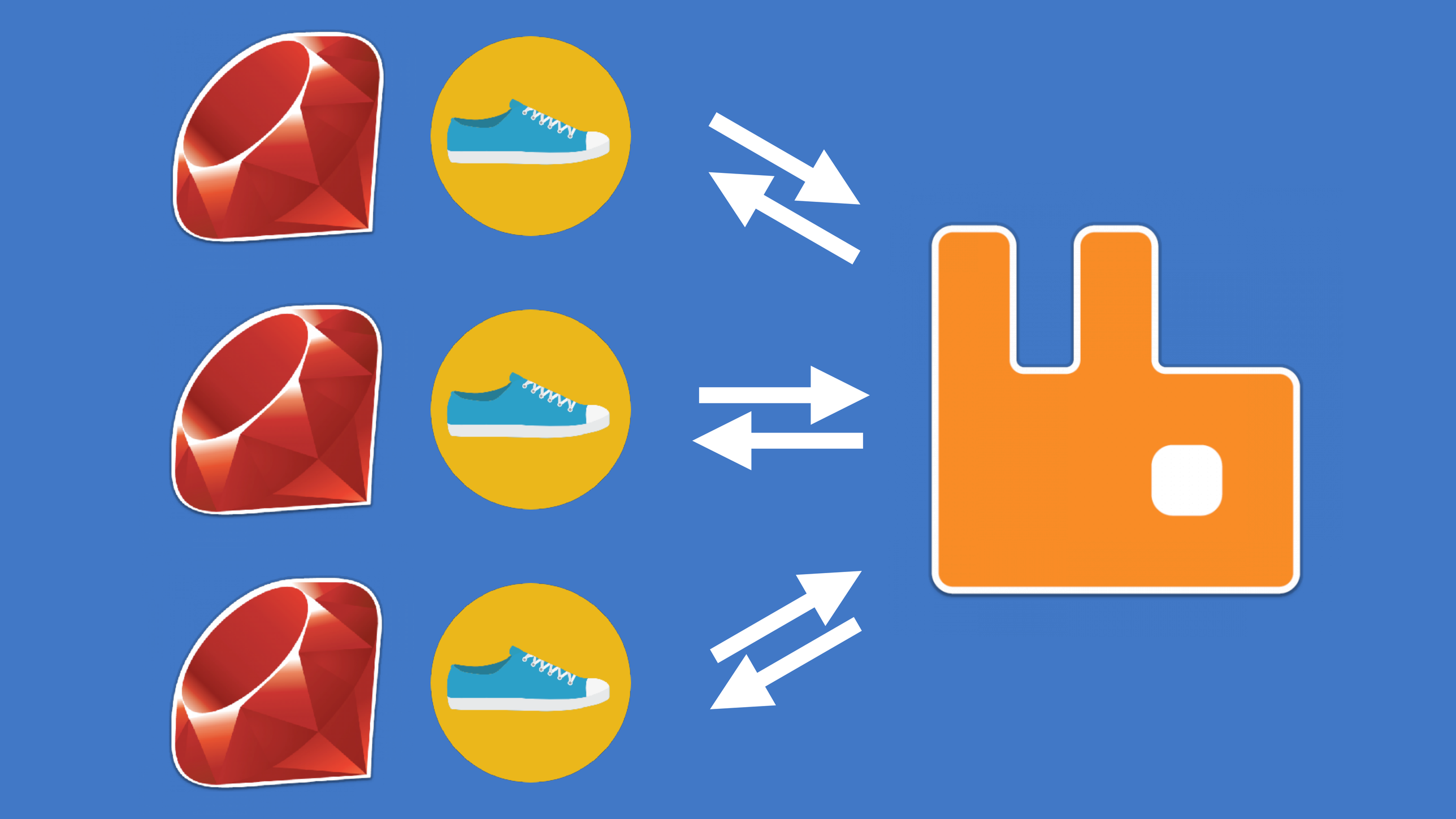

Queue
Job #4
Job #3
Job #2
Job #1
I need 1ns
Worker


Queue
Job #4
Job #3
Job #2
Job #1
OK
Worker
Job #1


Queue
Job #4
Job #3
Job #2
Job #1
Worker
Job #1


Queue
Job #3
Job #2
Job #1
RAM
Job #1
Job #2
Disk
Job #1
Job #2
Job #3

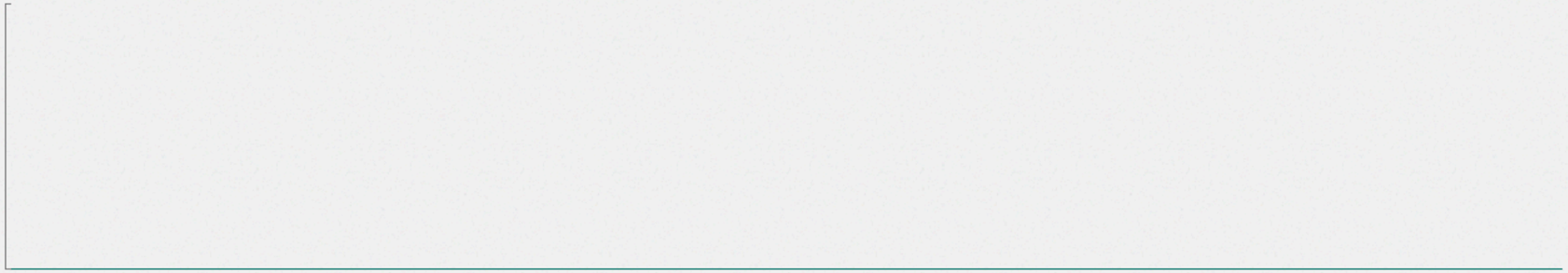

Lazy Queue
Job #3
Job #2
Job #1
RAM
Disk
Job #1
Job #2
Job #3



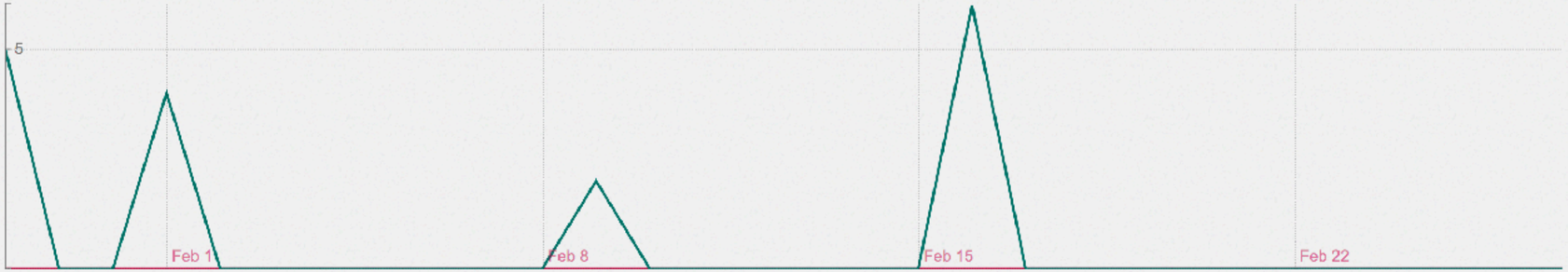
18	0	0	0	0	0	0
Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead

Dashboard

Polling interval: 5 sec



History 1 week **1 month** 3 months 6 months



Redis

4.0.8	10	30	1.69M	3.15M
Version	Uptime (days)	Connections	Memory Usage	Peak Memory Usage

Username: *

Password: *

Login

WEB UI AUTHORIZATION	⊖	⊖	✓
DEDICATED SUPPORT	None	Email	Email
LICENSE	LGPL	Commercial No Custom Terms	Commercial with Custom Terms
PRICING	Free	$950 / year Unlimited Usage	$1950 / year per 100 Workers
PURCHASING	n/a	Credit Card	Credit Card, Invoice

RabbitMQ

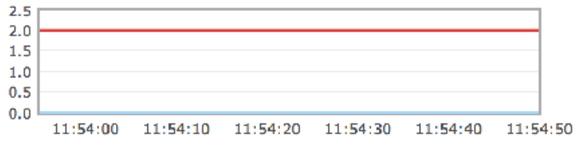
User: Log out
Cluster: rabbit@46e1483dee20 (change)
RabbitMQ 3.6.10, Erlang 19.3

Overview | Connections | Channels | Exchanges | Queues | Admin

Overview

Totals

Queued messages (chart: last minute) (?)



Ready	2
Unacked	0
Total	2

Message rates (chart: last minute) (?)

Publish	0.00/s	Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s	Disk read	0.00/s
Publisher confirm	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s	Disk write	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Return	0.00/s		

Global counts (?)

Connections: 12 | Channels: 12 | Exchanges: 17 | Queues: 8 | Consumers: 12

Node

Node: rabbit@46e1483dee20 **(More about this node)**

File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Rates mode	Info	Reset stats DB	+/-
39 1048576 available	12 943626 available	474 1048576 available	77MB 800MB high watermark	25GB 48MB low watermark	basic	Disc 1	Reset	

Reset stats on all nodes

Paths

Config file	/etc/rabbitmq/rabbitmq.config
Database directory	/var/lib/rabbitmq/mnesia/rabbit@46e1483dee20
Log file	tty
SASL log file	tty

Ports and contexts

RabbitMQ

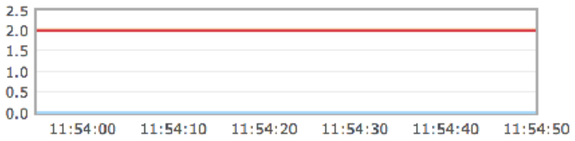
User: [redacted] Log out
Cluster: rabbit@46e1483dee20 (change)
RabbitMQ 3.6.10, Erlang 19.3

Overview | Connections | Channels | Exchanges | Queues | Admin

Overview

Totals

Queued messages (chart: last minute) (?)



Ready	2
Unacked	0
Total	2

Message rates (chart: last minute) (?)

Publish	0.00/s	Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s	Disk read	0.00/s
Publisher confirm	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s	Disk write	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Return	0.00/s		

Global counts (?)

Connections: 12 | Channels: 12 | Exchanges: 17 | Queues: 8 | Consumers: 12

Node

Node: rabbit@46e1483dee20 **(More about this node)**

File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Rates mode	Info	Reset stats DB +/-
39 1048576 available	12 943626 available	474 1048576 available	77MB 800MB high watermark	25GB 48MB low watermark	basic	Disc 1	Reset

Reset stats on all nodes

Paths

Config file	/etc/rabbitmq/rabbitmq.config
Database directory	/var/lib/rabbitmq/mnesia/rabbit@46e1483dee20
Log file	tty
SASL log file	tty

Ports and contexts

RabbitMQ

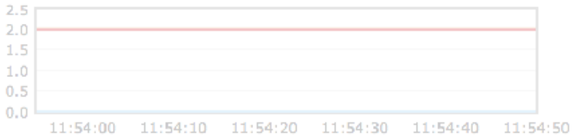
User:
Log out
Cluster: rabbit@46e1483dee20 (change)
RabbitMQ 3.6.10, Erlang 19.3

Overview | Connections | Channels | Exchanges | Queues | Admin

Overview

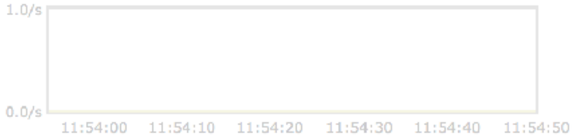
Totals

Queued messages (chart: last minute) (?)



Ready	2
Unacked	0
Total	2

Message rates (chart: last minute) (?)



Publish	0.00/s	Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s	Disk read	0.00/s
Publisher confirm	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s	Disk write	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Return	0.00/s		

Global counts (?)

Connections: 12 | Channels: 12 | Exchanges: 17 | Queues: 8 | Consumers: 12

Node

Node: rabbit@46e1483dee20 **(More about this node)**

File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Rates mode	Info	Reset stats DB	+/-
39 1048576 available	12 943626 available	474 1048576 available	77MB 800MB high watermark	25GB 48MB low watermark	basic	Disc 1	Reset	

Reset stats on all nodes

Paths

Config file	/etc/rabbitmq/rabbitmq.config
Database directory	/var/lib/rabbitmq/mnesia/rabbit@46e1483dee20
Log file	tty
SASL log file	tty

Ports and contexts

RabbitMQ

User: Log out
Cluster: rabbit@46e1483dee20 (change)
RabbitMQ 3.6.10, Erlang 19.3

Overview | Connections | Channels | Exchanges | Queues | Admin

Queues

All queues (8)

Pagination

Page 1 of 1 - Filter: Regex (?)(?)

Displaying 8 items , page size up to: 100

Overview			Messages			Message bytes		Message rates			+/-
Name	Features	State	Ready	Unacked	Total	In Memory	Persistent	incoming	deliver / get	ack	
chatbot_messages	D DLX DLK	idle	0	0	0	0B	0B				
chatbot_messages.error	D	idle	0	0	0	0B	0B				
default	D DLX	running	0	0	0	0B	0B	0.00/s	0.00/s	0.00/s	
default-error	D	idle	2	0	2	2.0kB	2.0kB	0.00/s			
default-retry	D TTL DLX	idle	0	0	0	0B	0B				
mailers	D DLX	idle	0	0	0	0B	0B	0.00/s	0.00/s	0.00/s	
mailers-error	D	idle	0	0	0	0B	0B				
mailers-retry	D TTL DLX	idle	0	0	0	0B	0B				

Add a new queue

HTTP API | Command Line

Update every 5 seconds

Last update: 2018-02-27 12:04:46

RabbitMQ

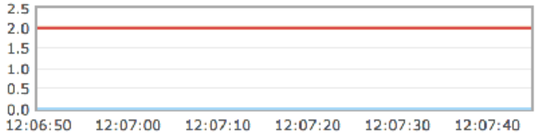
User: [ ] Log out
Cluster: rabbit@46e1483dee20 (change)
RabbitMQ 3.6.10, Erlang 19.3

Overview | Connections | Channels | Exchanges | Queues | Admin

Queue default-error

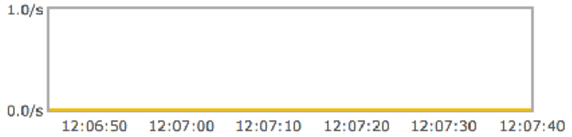
Overview

Queued messages (chart: last minute) (?)



Ready	2
Unacked	0
Total	2

Message rates (chart: last minute) (?)



Publish	0.00/s

Details

					Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Features	durable: true	State	idle	Messages (?)	2	2	0	2	2	0
Policy		Consumers	0	Message body bytes (?)	2.0kB	2.0kB	0B	2.0kB	2.0kB	0B
		Consumer utilisation (?)	0%	Process memory (?)	15kB					

Consumers

Bindings

Publish message

Get messages

Move messages

Delete

Purge

Runtime Metrics (Advanced)

Reductions (per second) (chart: last minute) (?)

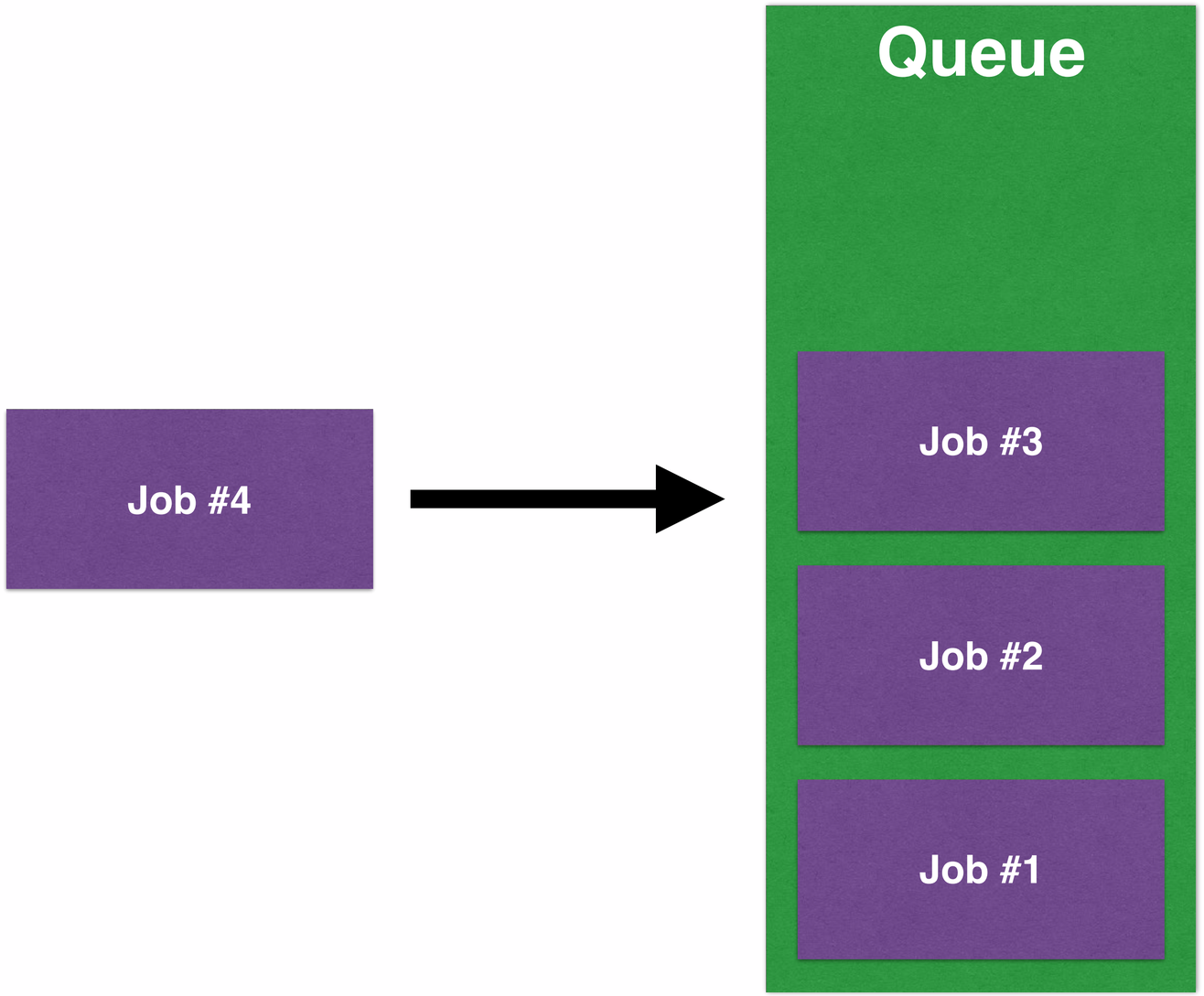
User:

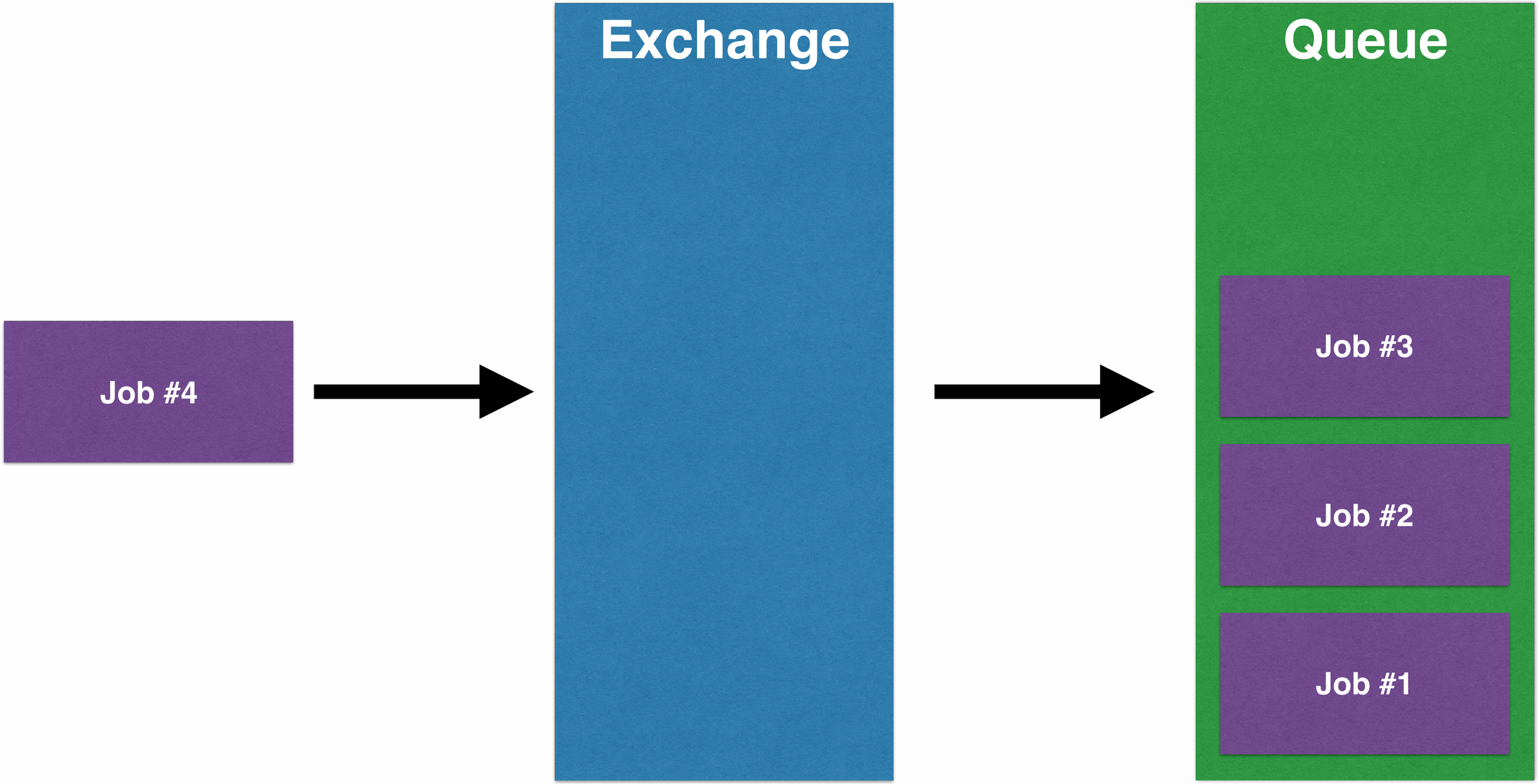
Log out

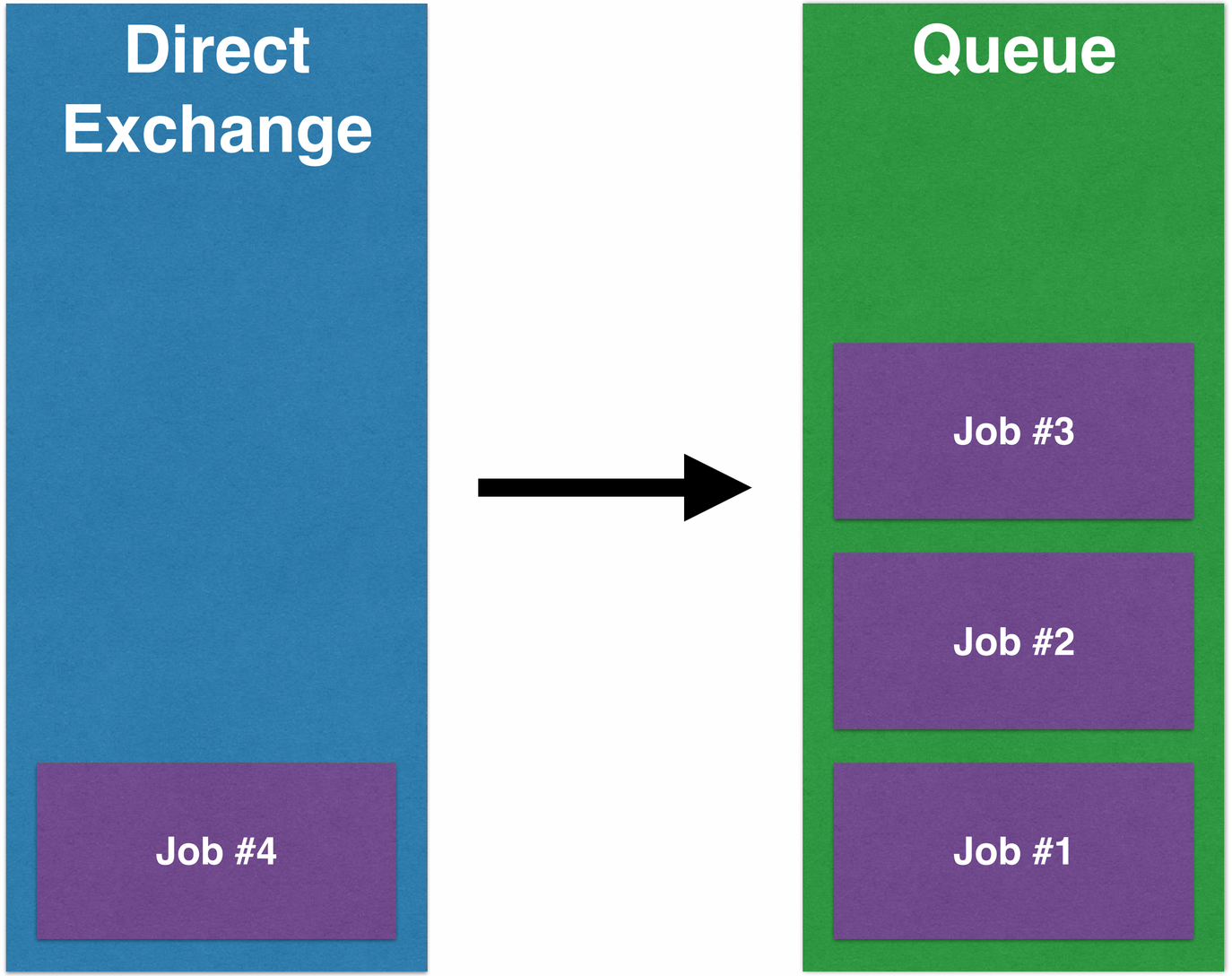
Cluster: rabbit@46e1483dee20 (change)

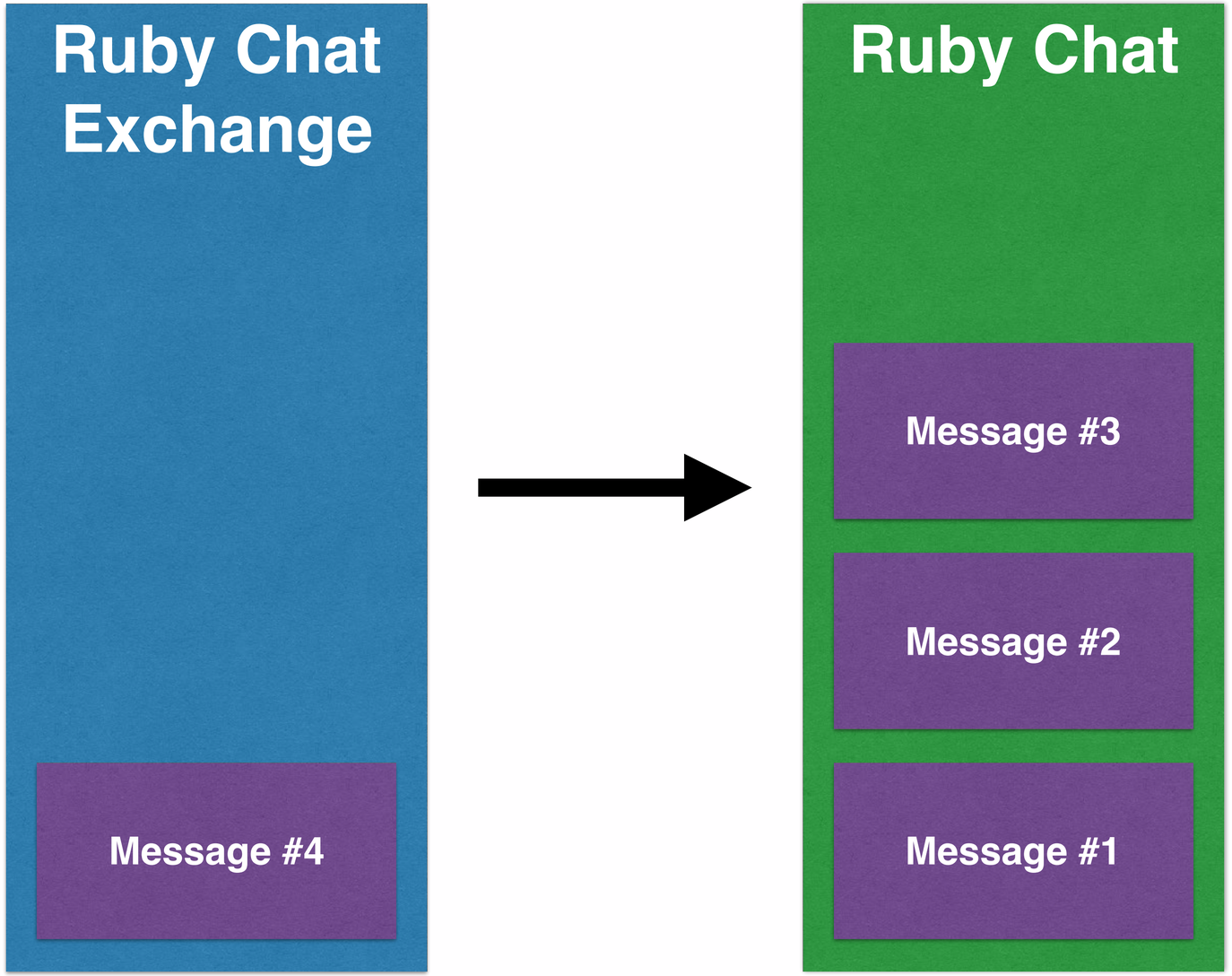
RabbitMQ 3.6.10, Erlang 19.3

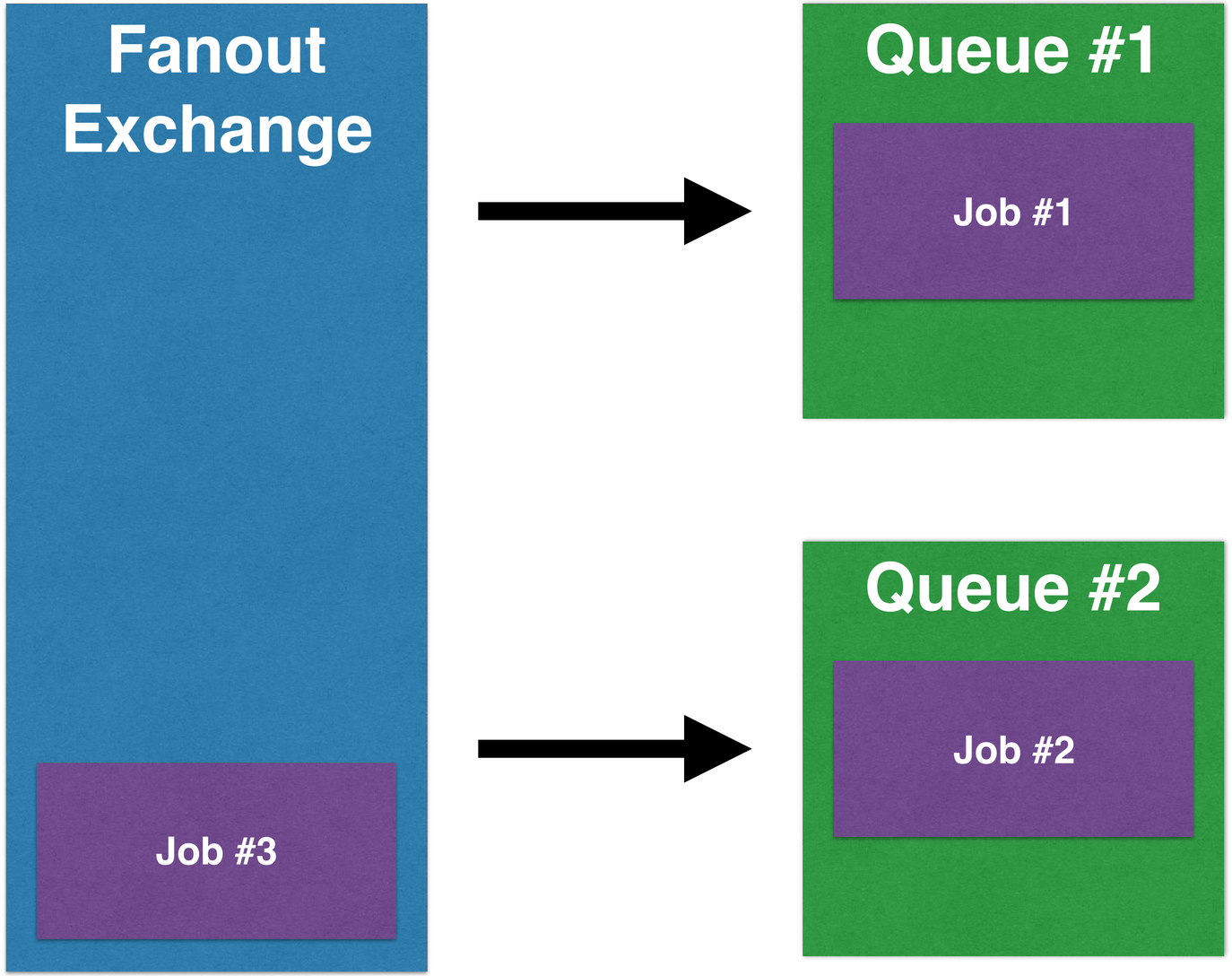
Overview | Connections | Channels | **Exchanges** | Queues | Admin

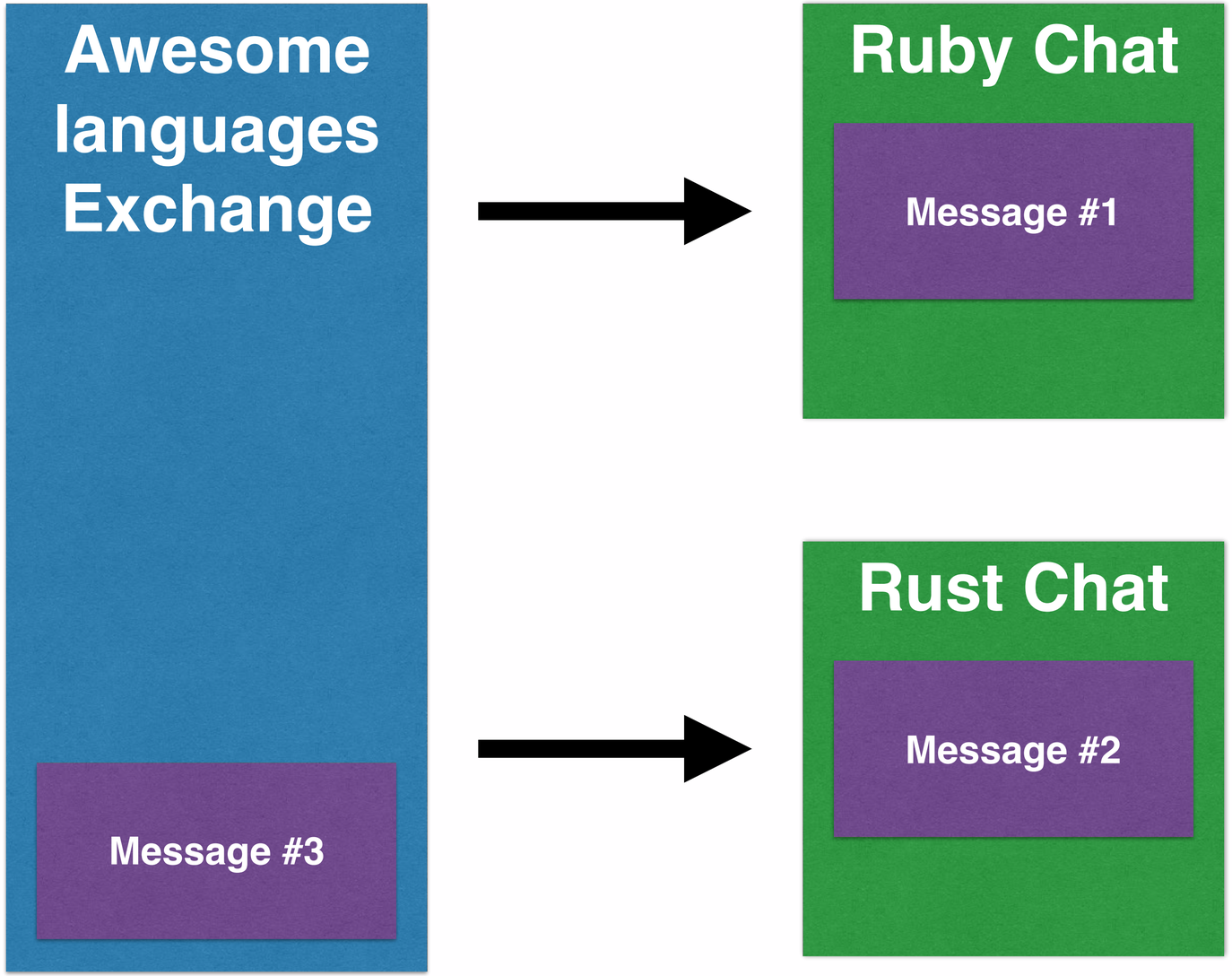

Queue
Job #4
Job #3
Job #2
Job #1

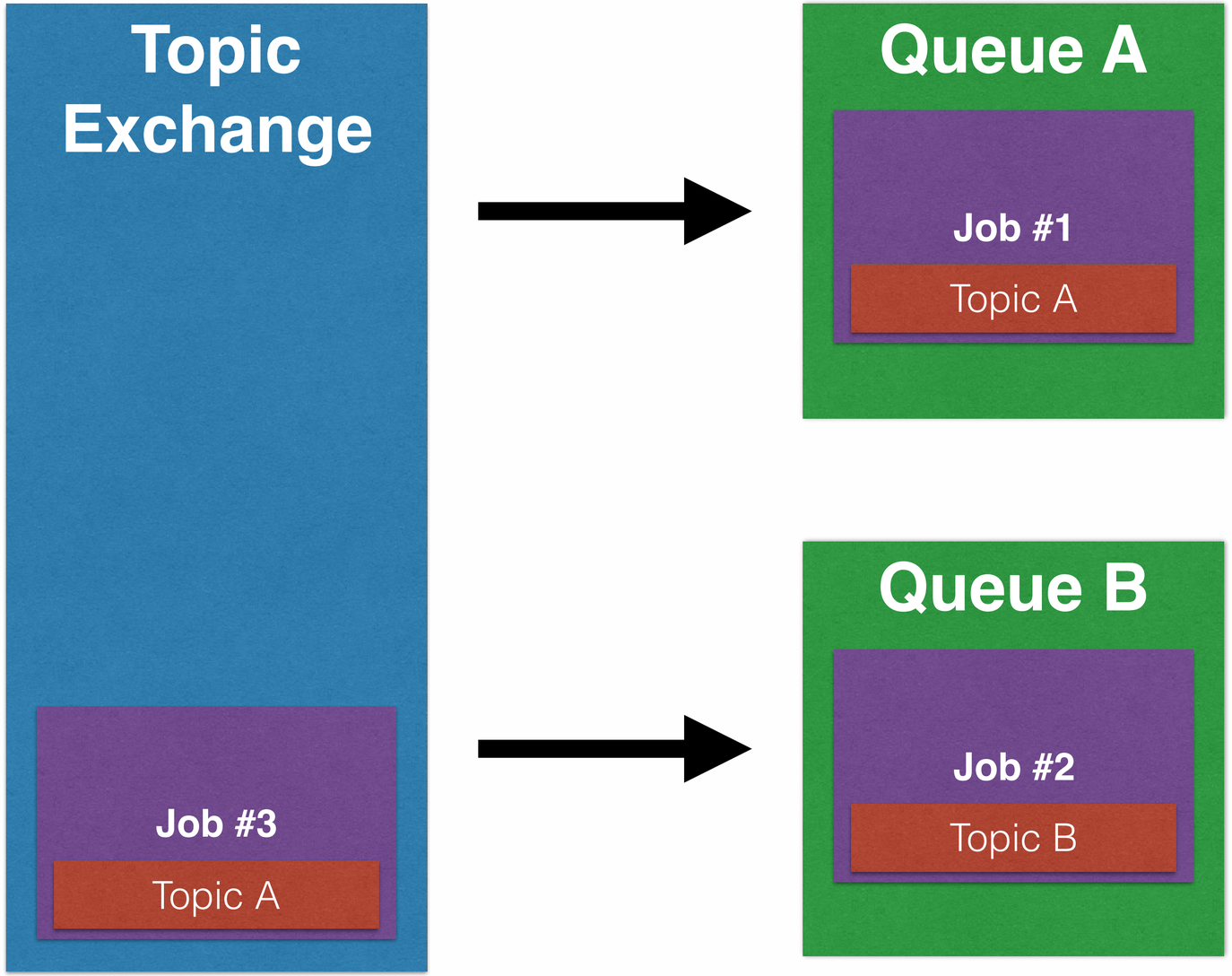

Exchange
Queue
Job #4
Job #3
Job #2
Job #1

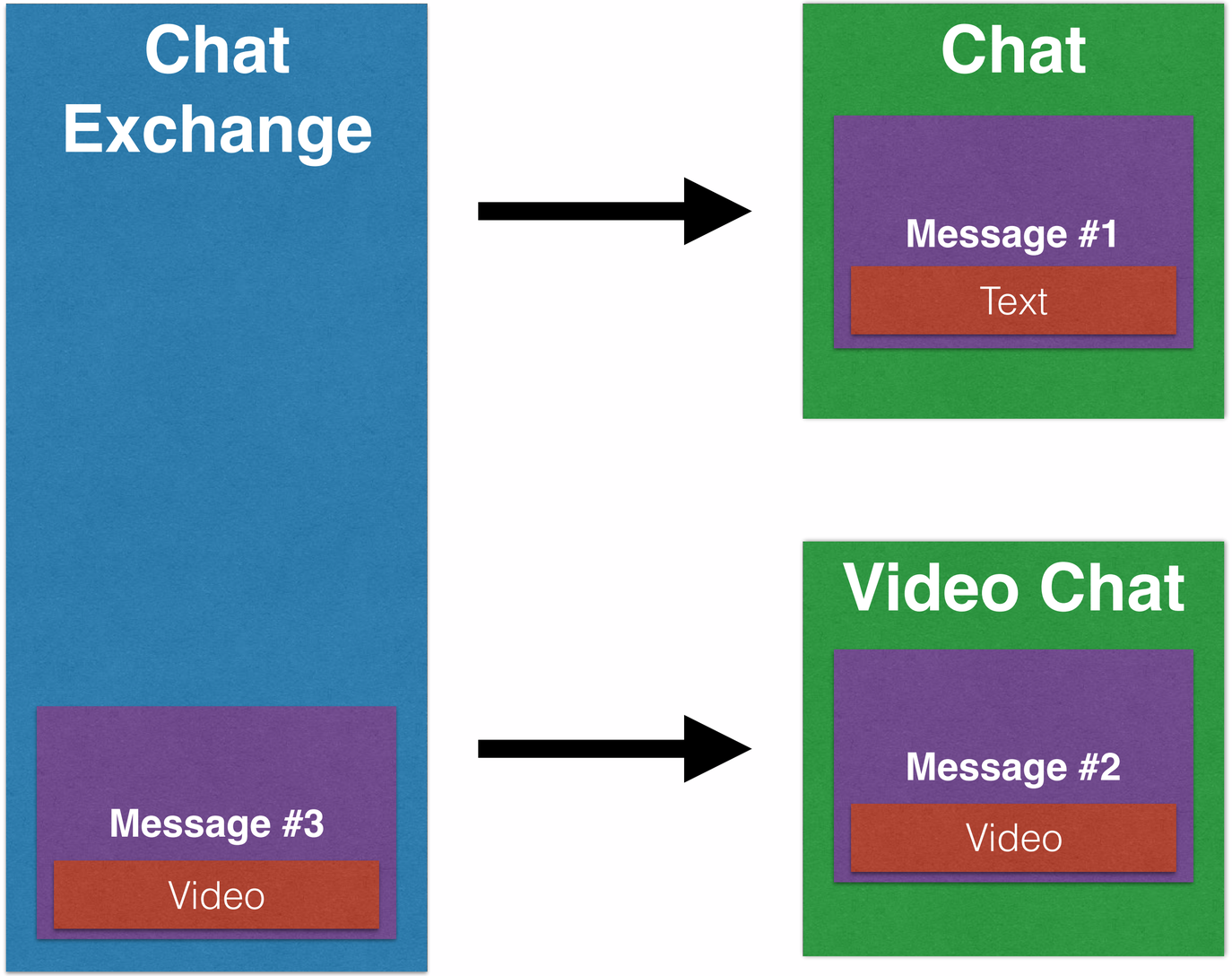

Direct
Exchange
Job #4
Queue
Job #3
Job #2
Job #1

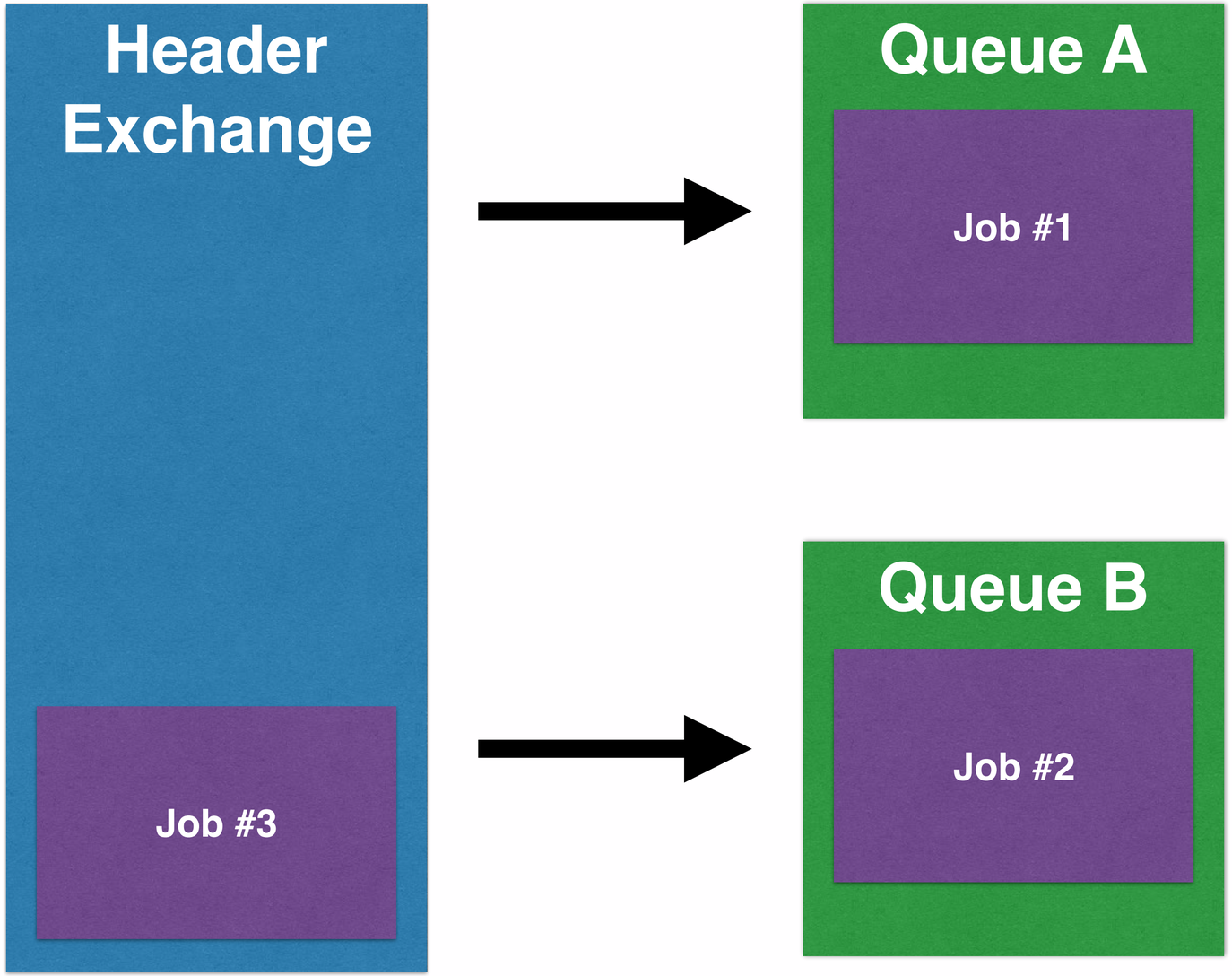

Ruby Chat Exchange
Message #4
Ruby Chat
Message #3
Message #2
Message #1


Fanout
Exchange
Job #3
Queue #1
Job #1
Queue #2
Job #2


Awesome languages Exchange
Message #3
Ruby Chat
Message #1
Rust Chat
Message #2


Topic Exchange
Job #3
Topic A
Queue A
Job #1
Topic A
Queue B
Job #2
Topic B


Chat Exchange
Message #3
Video
Chat
Message #1
Text
Video Chat
Message #2
Video

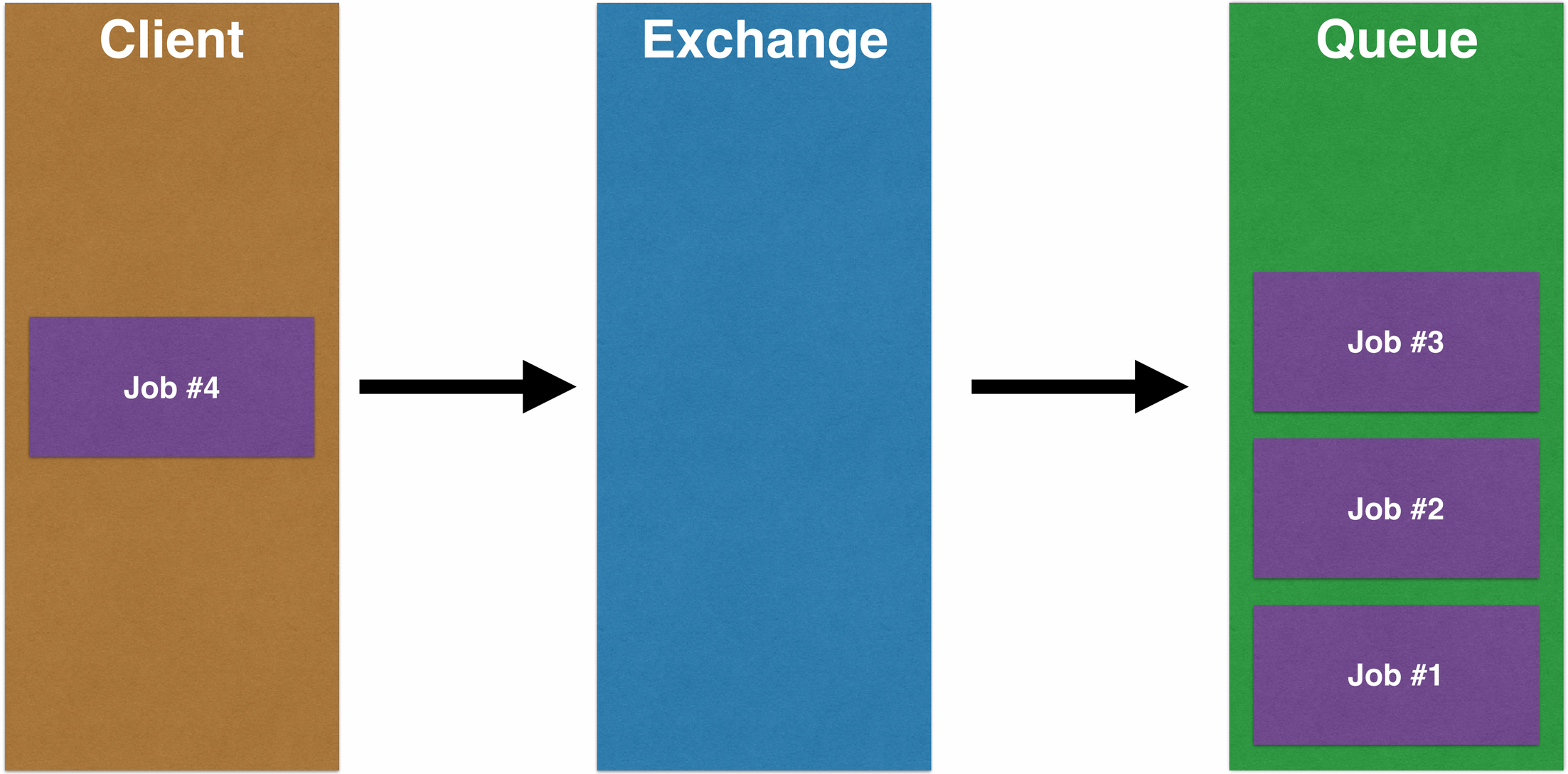

Header
Exchange
Job #3
Queue A
Job #1
Queue B
Job #2

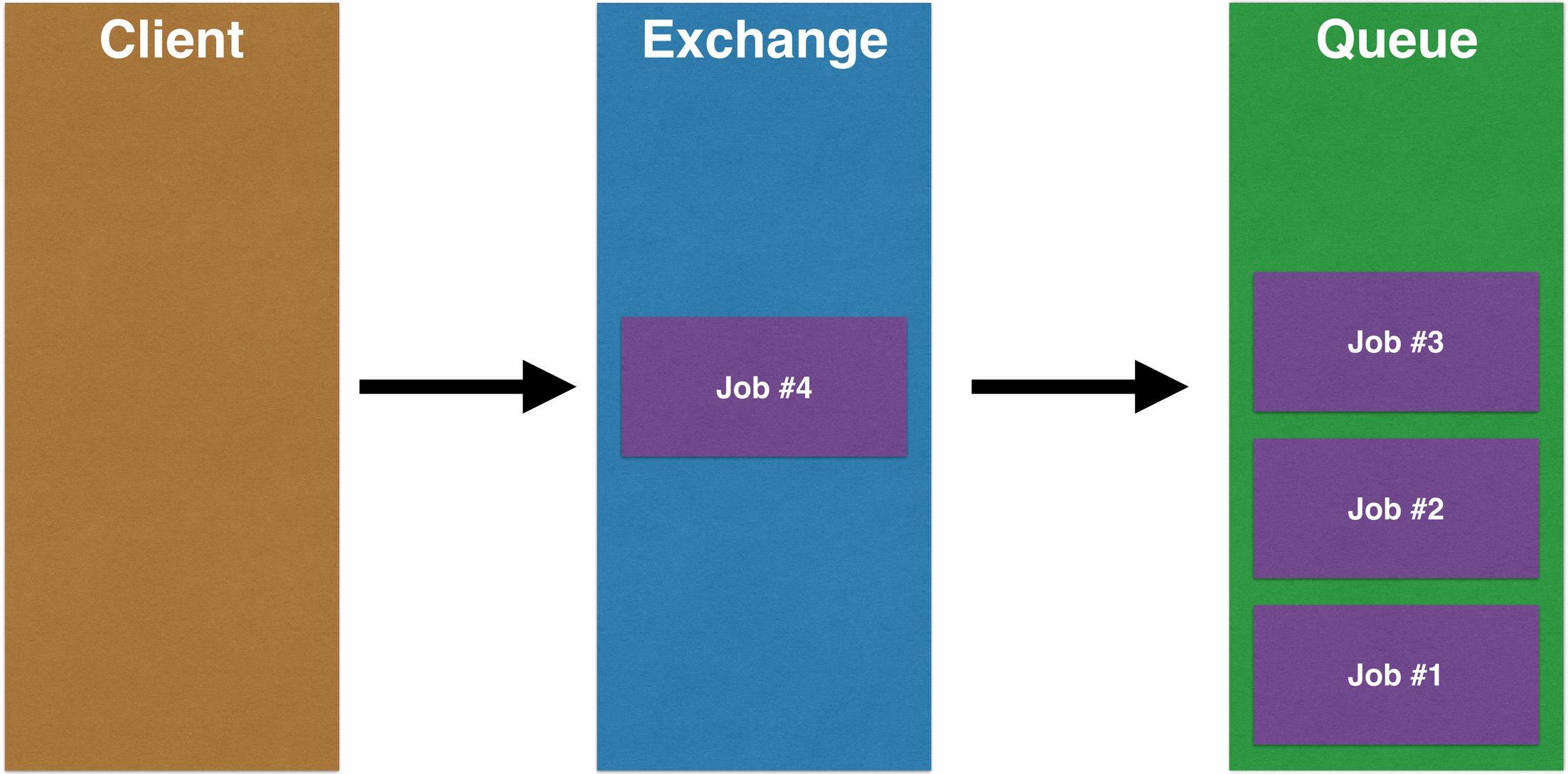
But what would you use the?

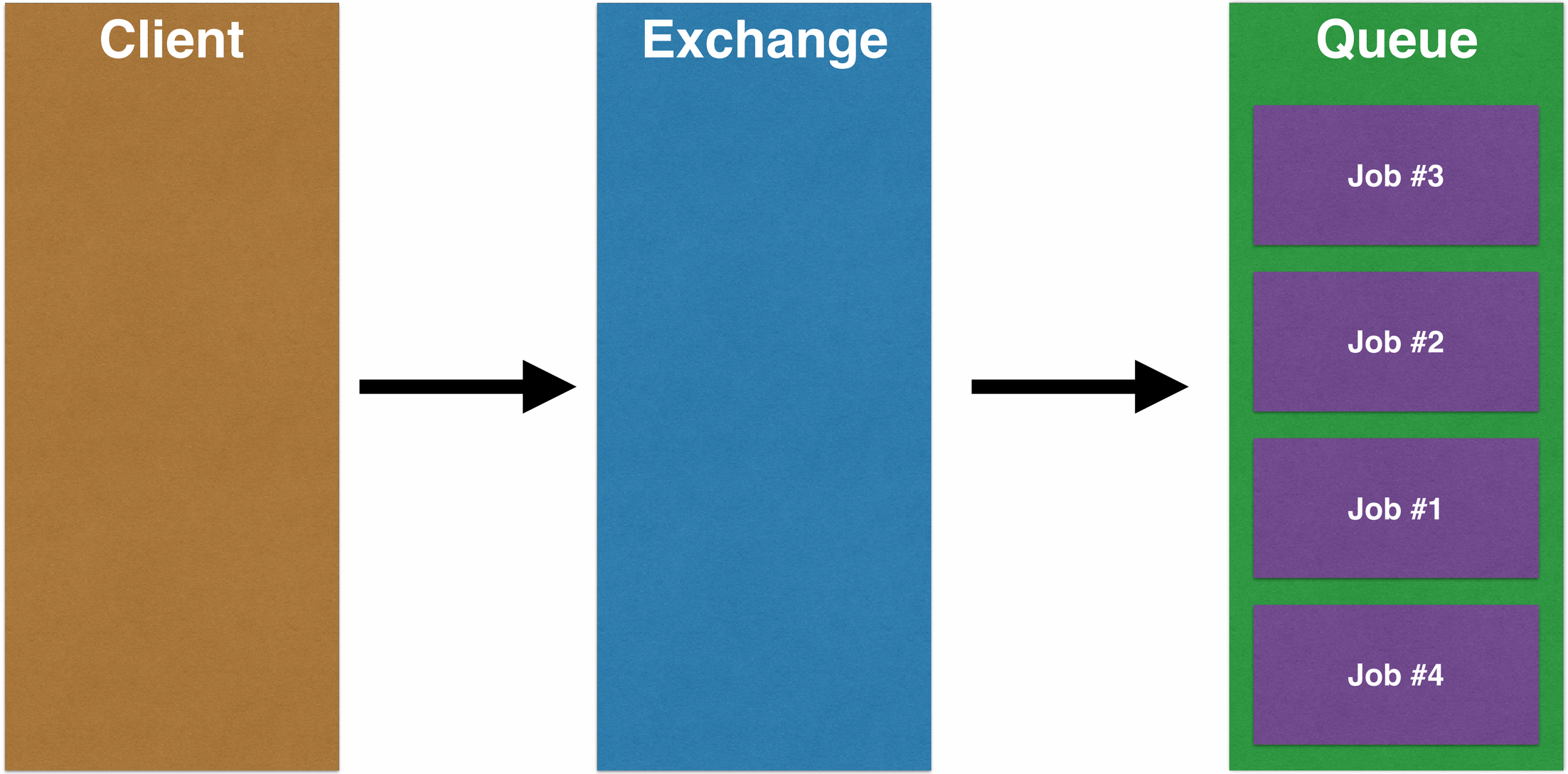
03

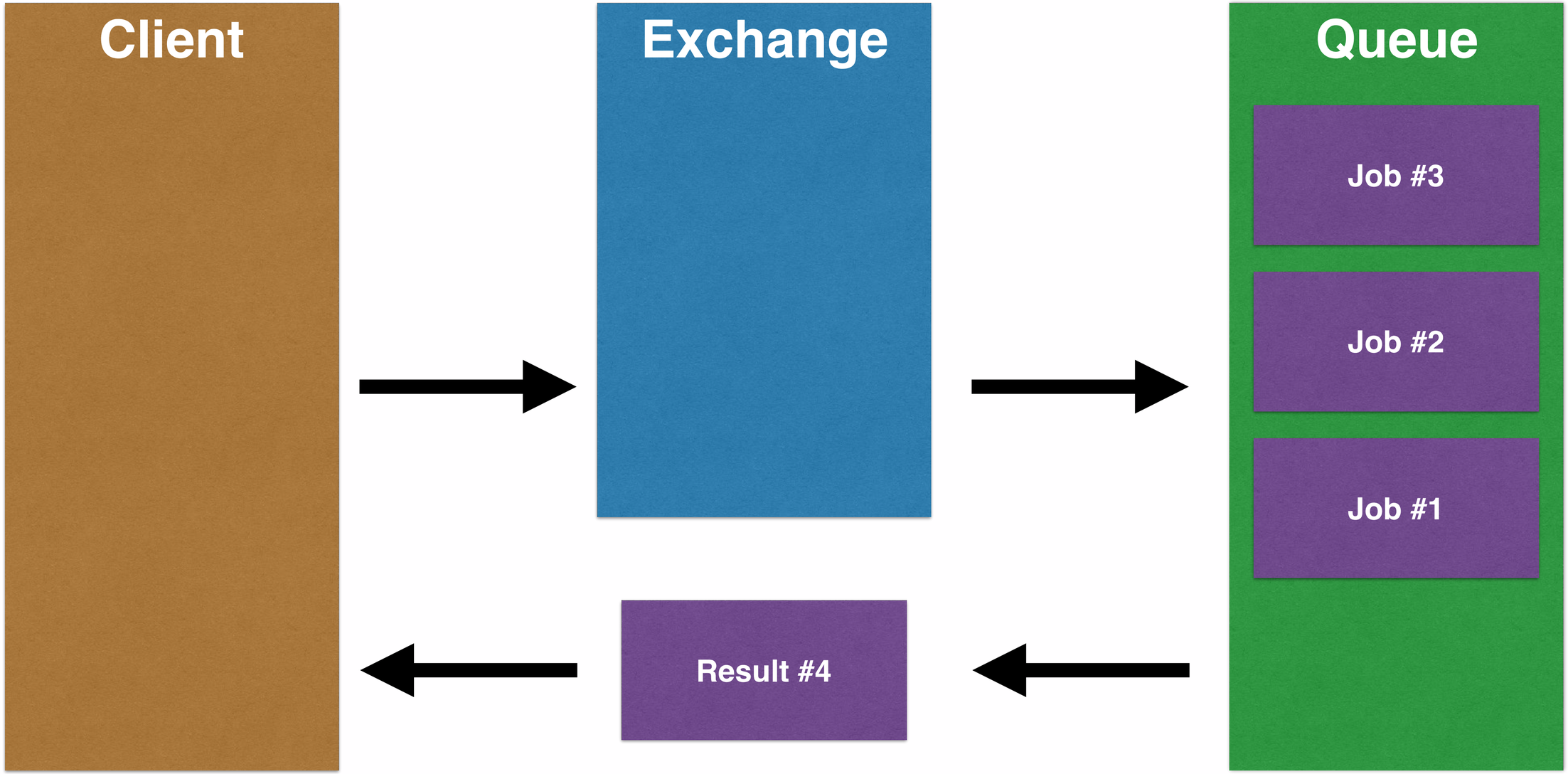
SPECIAL FEATURES

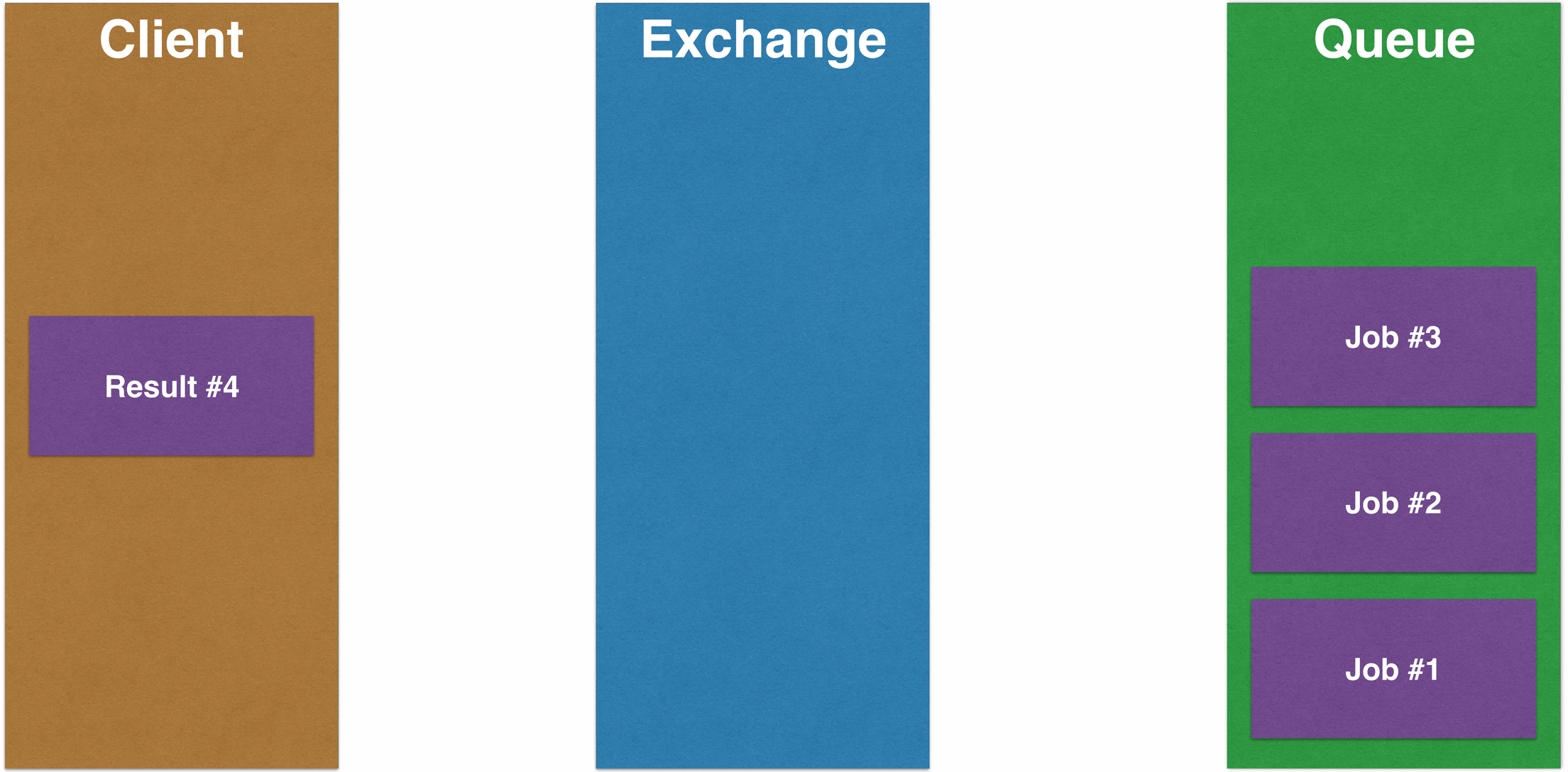
Direct reply-to

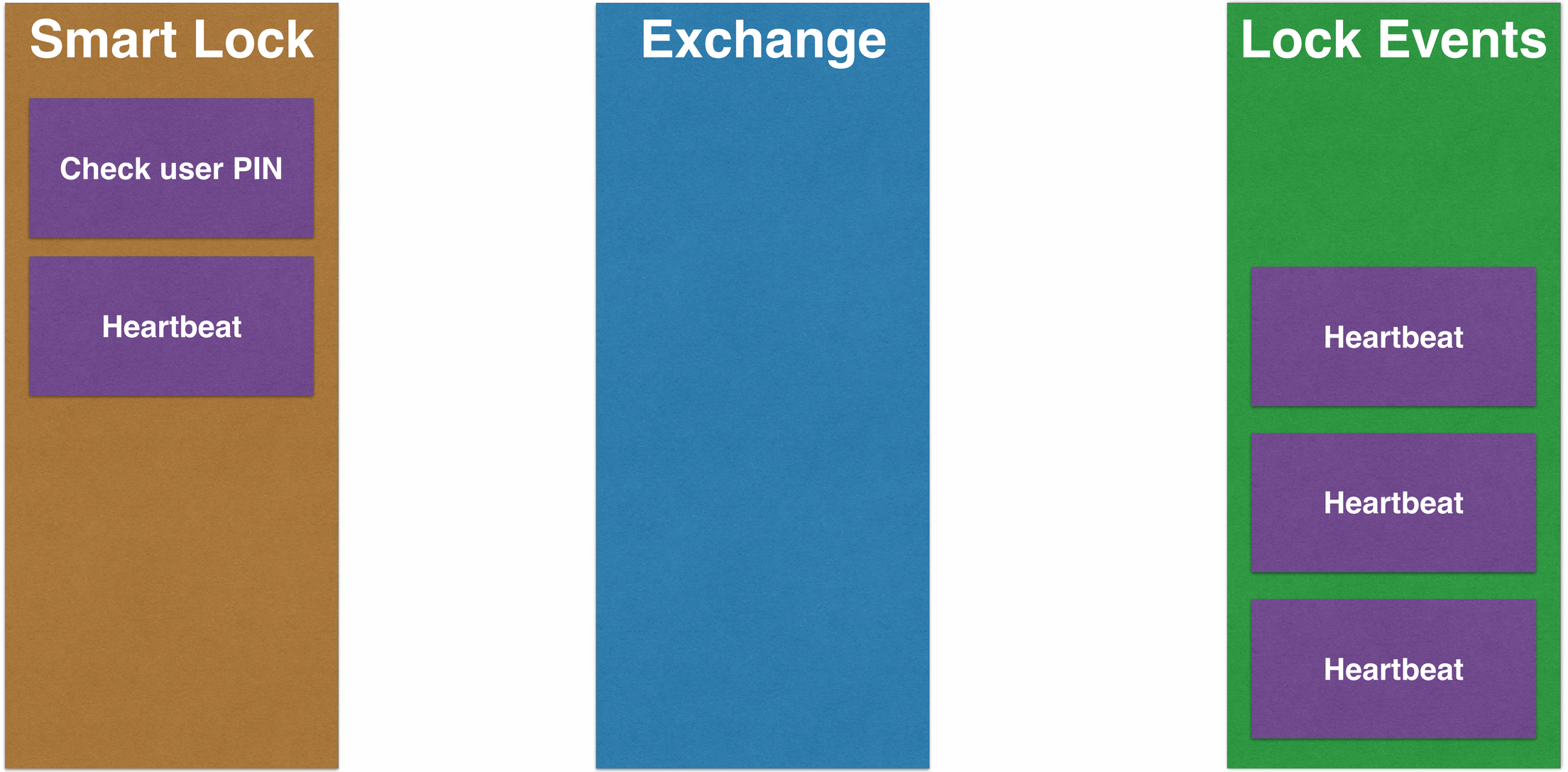

Client
Job #4
Exchange
Queue
Job #3
Job #2
Job #1

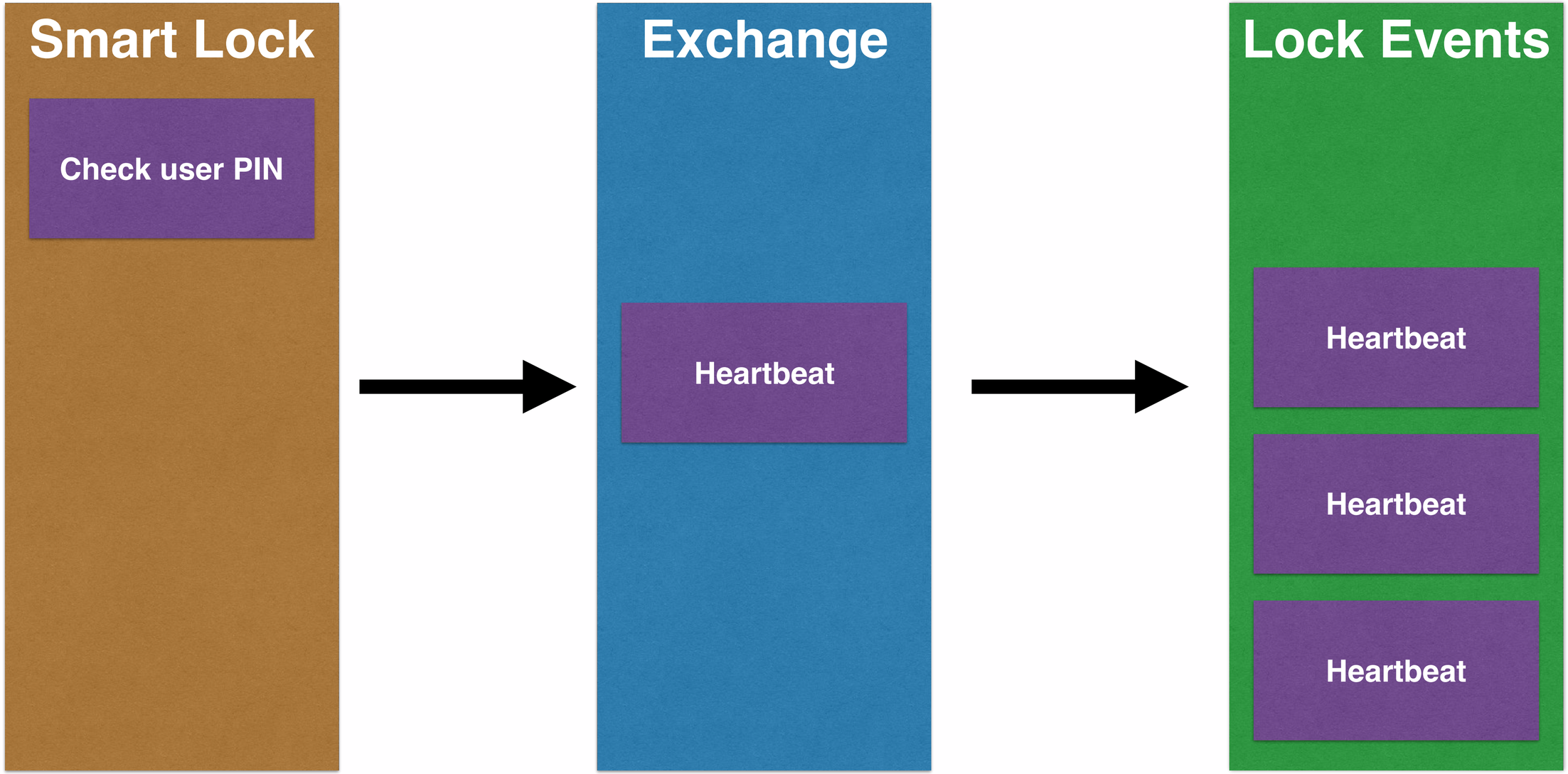

Client
Exchange
Job #4
Queue
Job #3
Job #2
Job #1

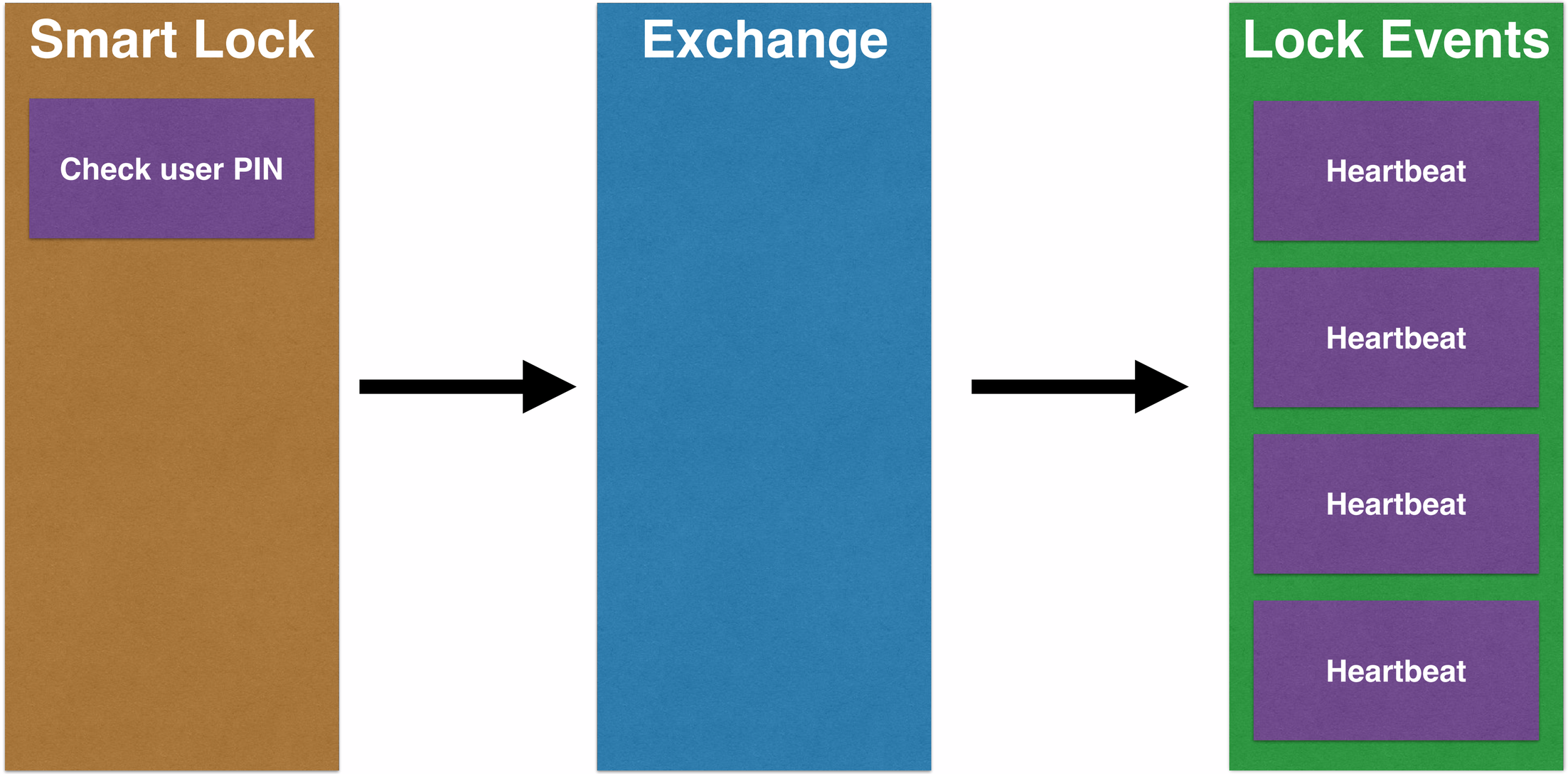

Client
Exchange
Queue
Job #3
Job #2
Job #1
Job #4

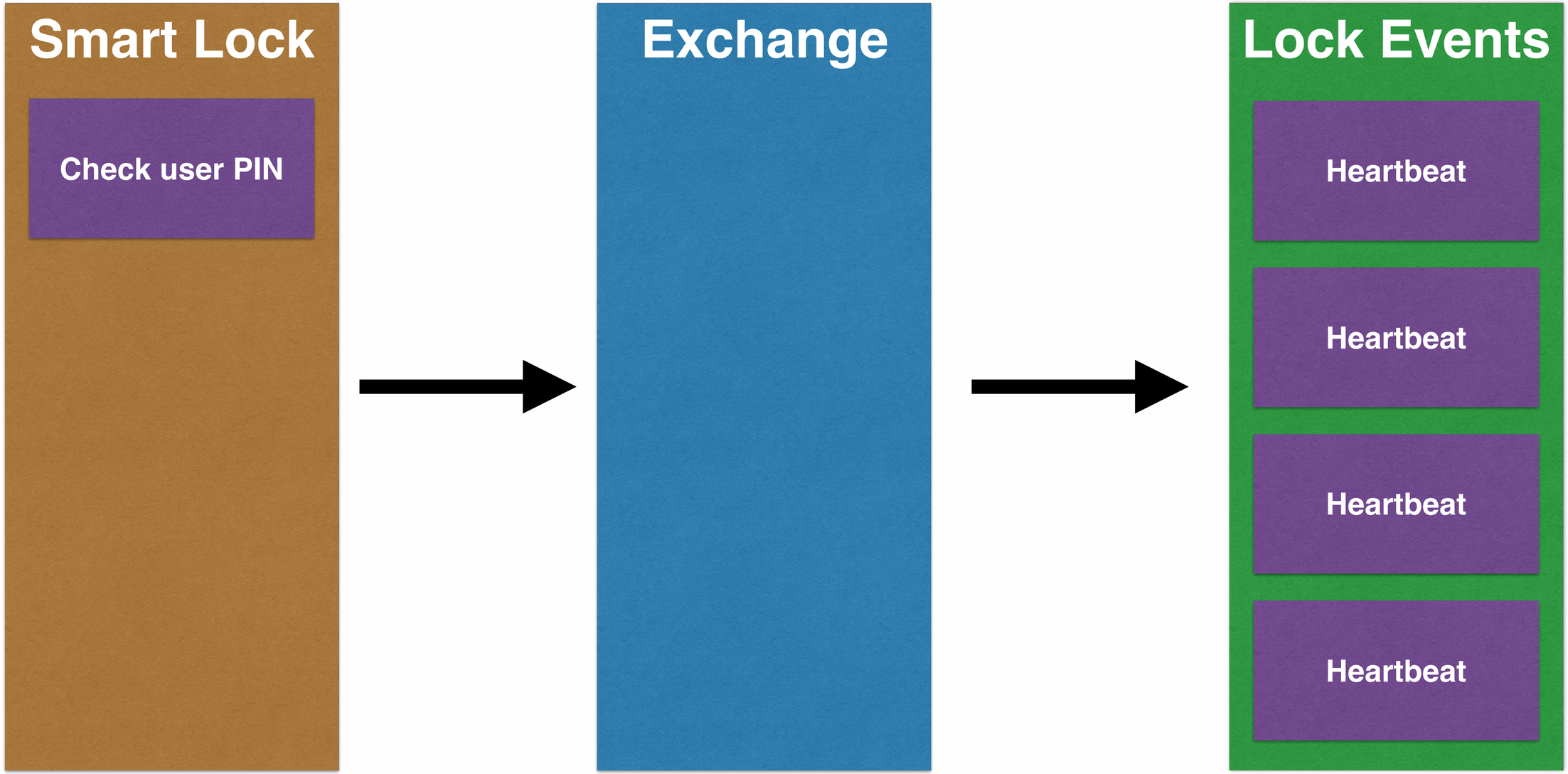

Client
Exchange
Queue
Job #3
Job #2
Job #1
Result #4

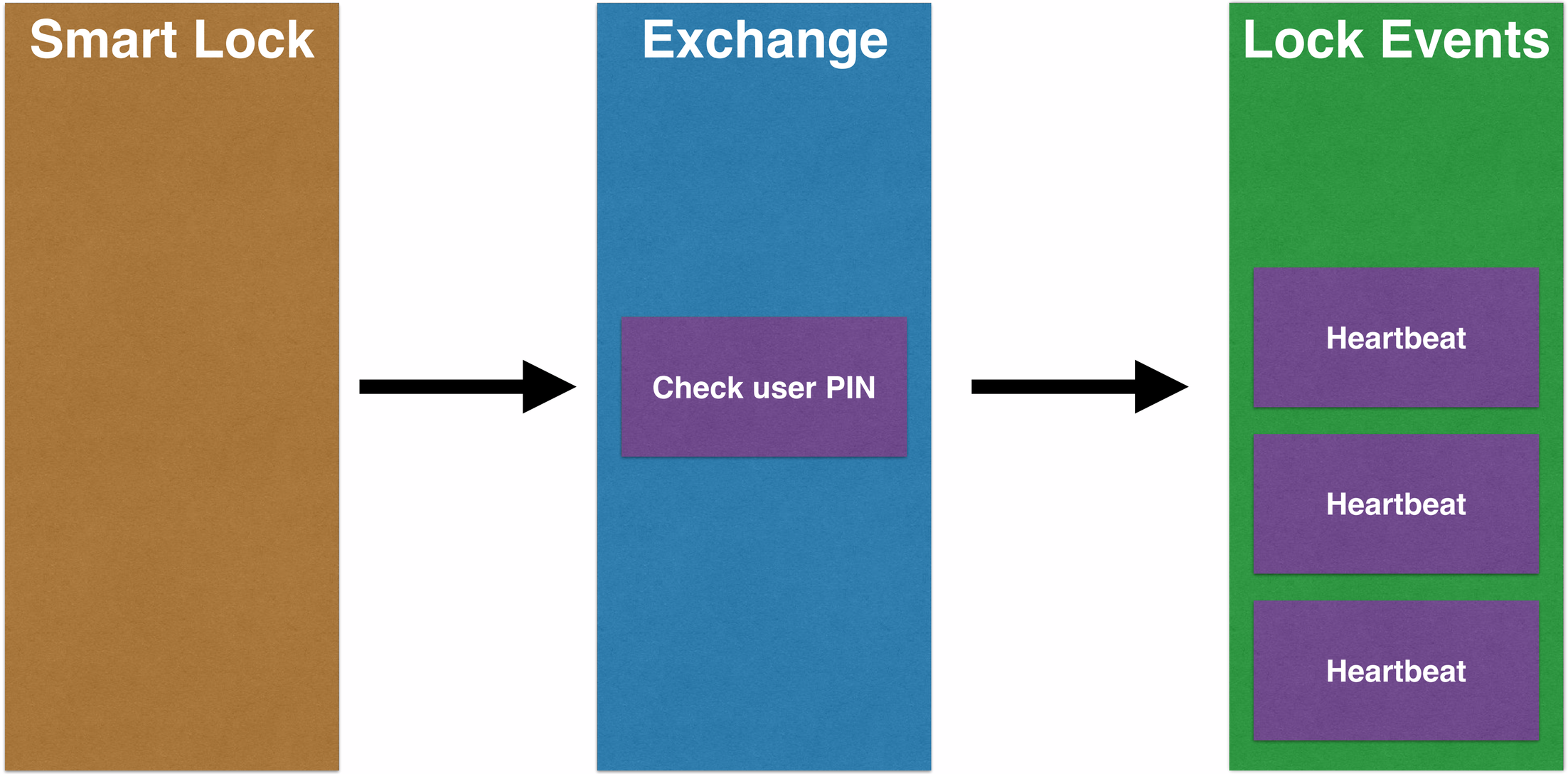

Client
Result #4
Exchange
Queue
Job #3
Job #2
Job #1

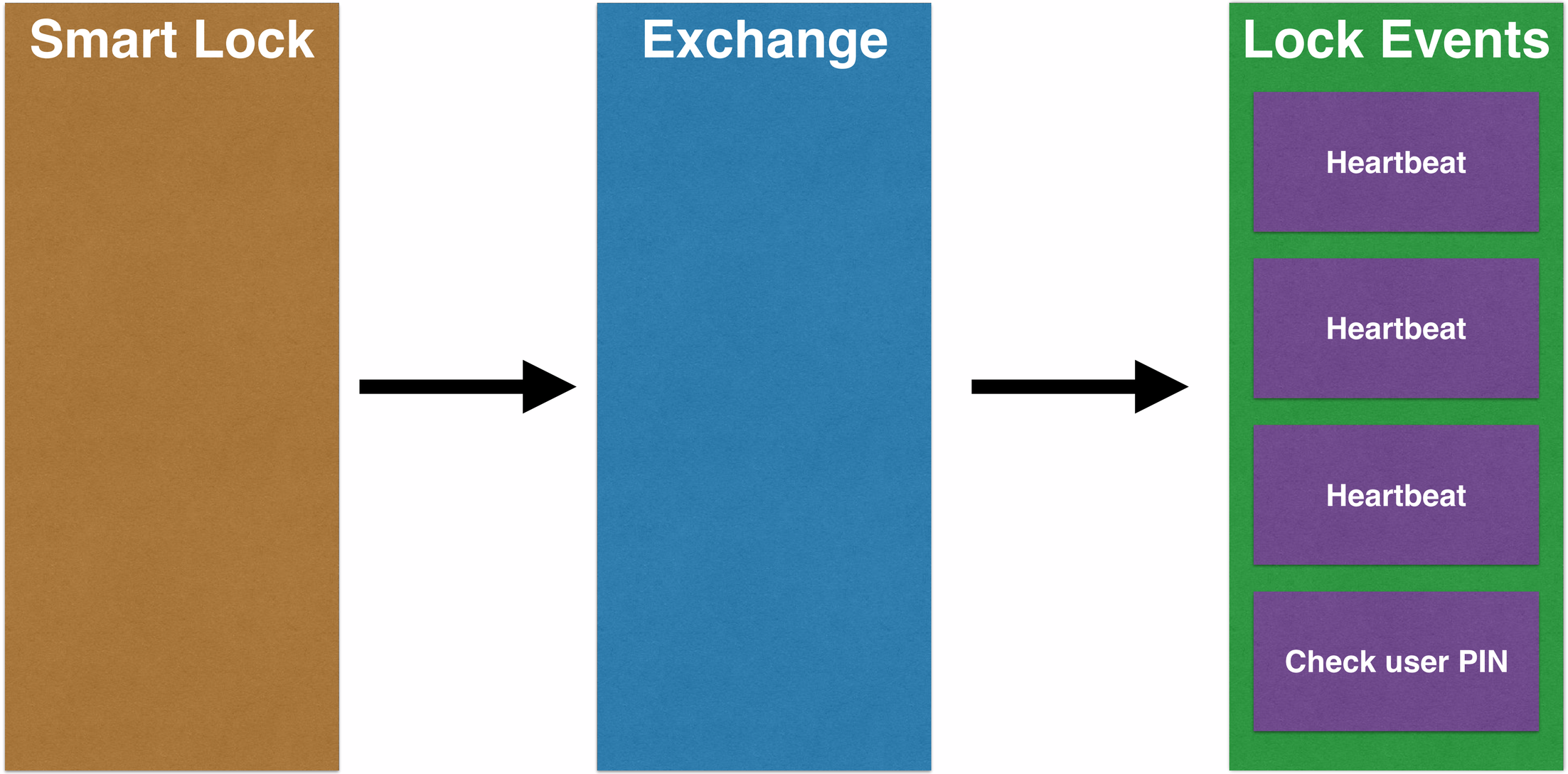

Smart Lock
Check user PIN
Heartbeat
Exchange
Lock Events
Heartbeat
Heartbeat
Heartbeat

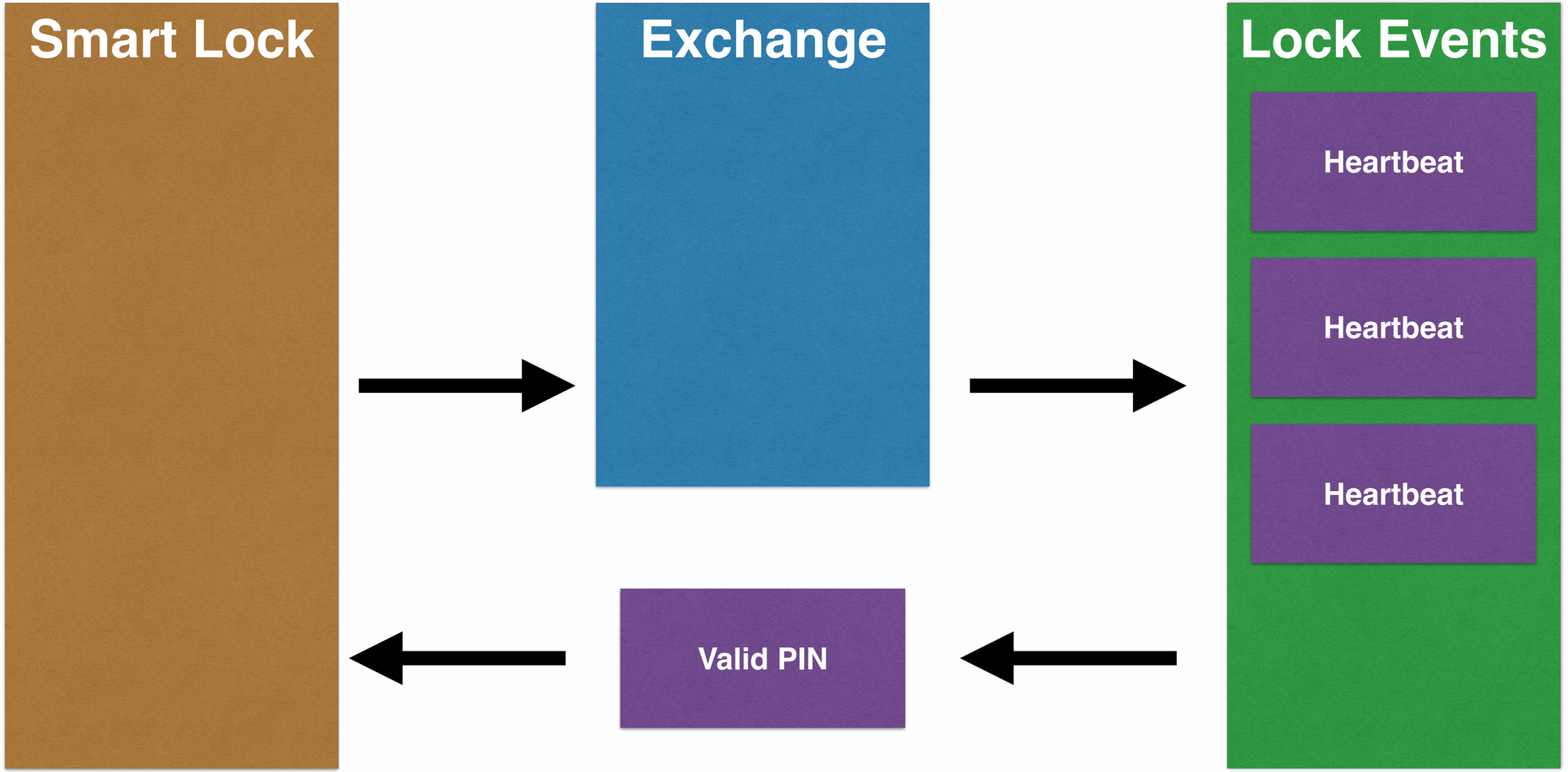

Smart Lock
Check user PIN
Exchange
Heartbeat
Lock Events
Heartbeat
Heartbeat
Heartbeat

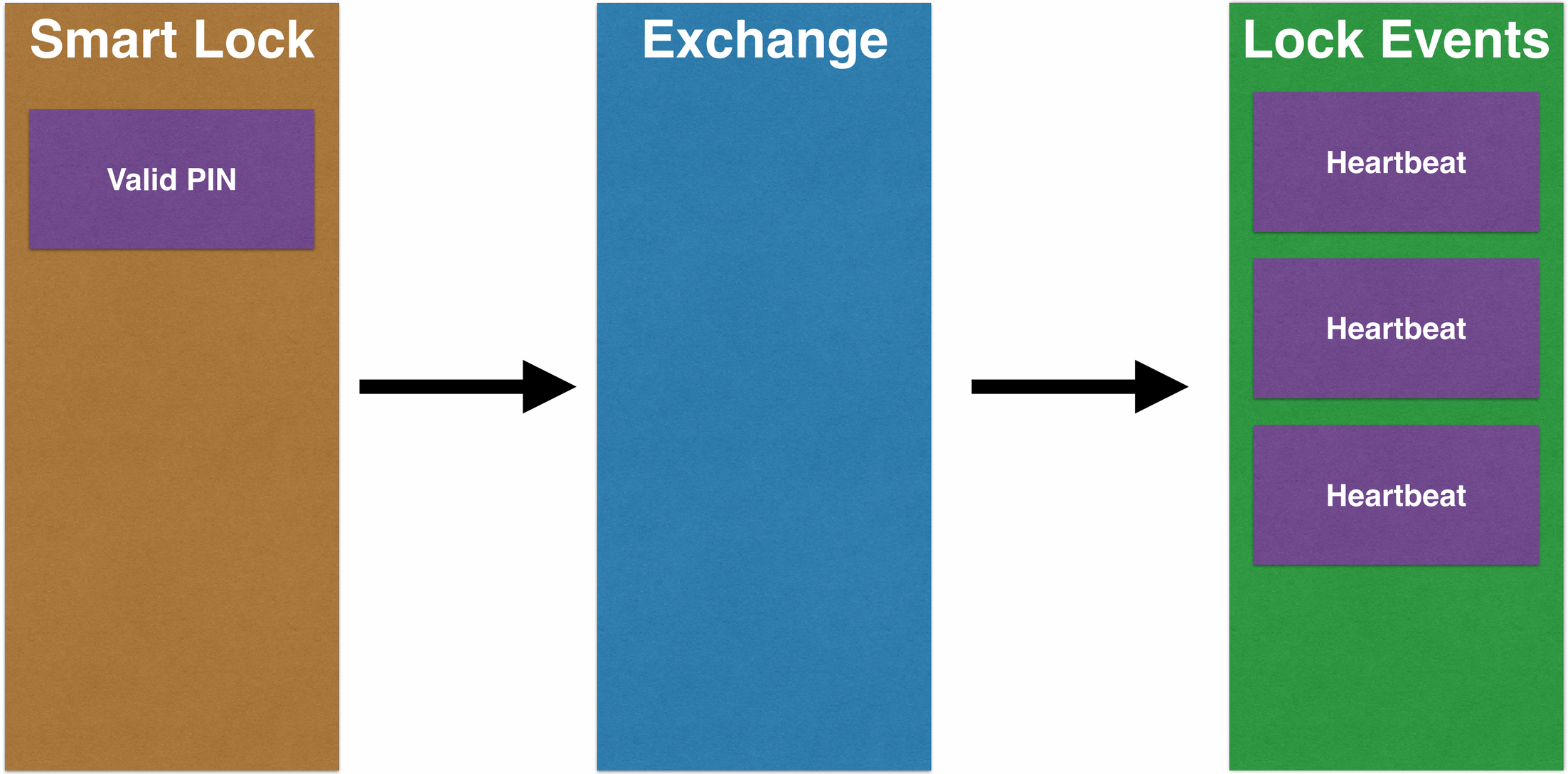

Smart Lock
Check user PIN
Exchange
Lock Events
Heartbeat
Heartbeat
Heartbeat
Heartbeat


Smart Lock
Check user PIN
Exchange
Lock Events
Heartbeat
Heartbeat
Heartbeat
Heartbeat

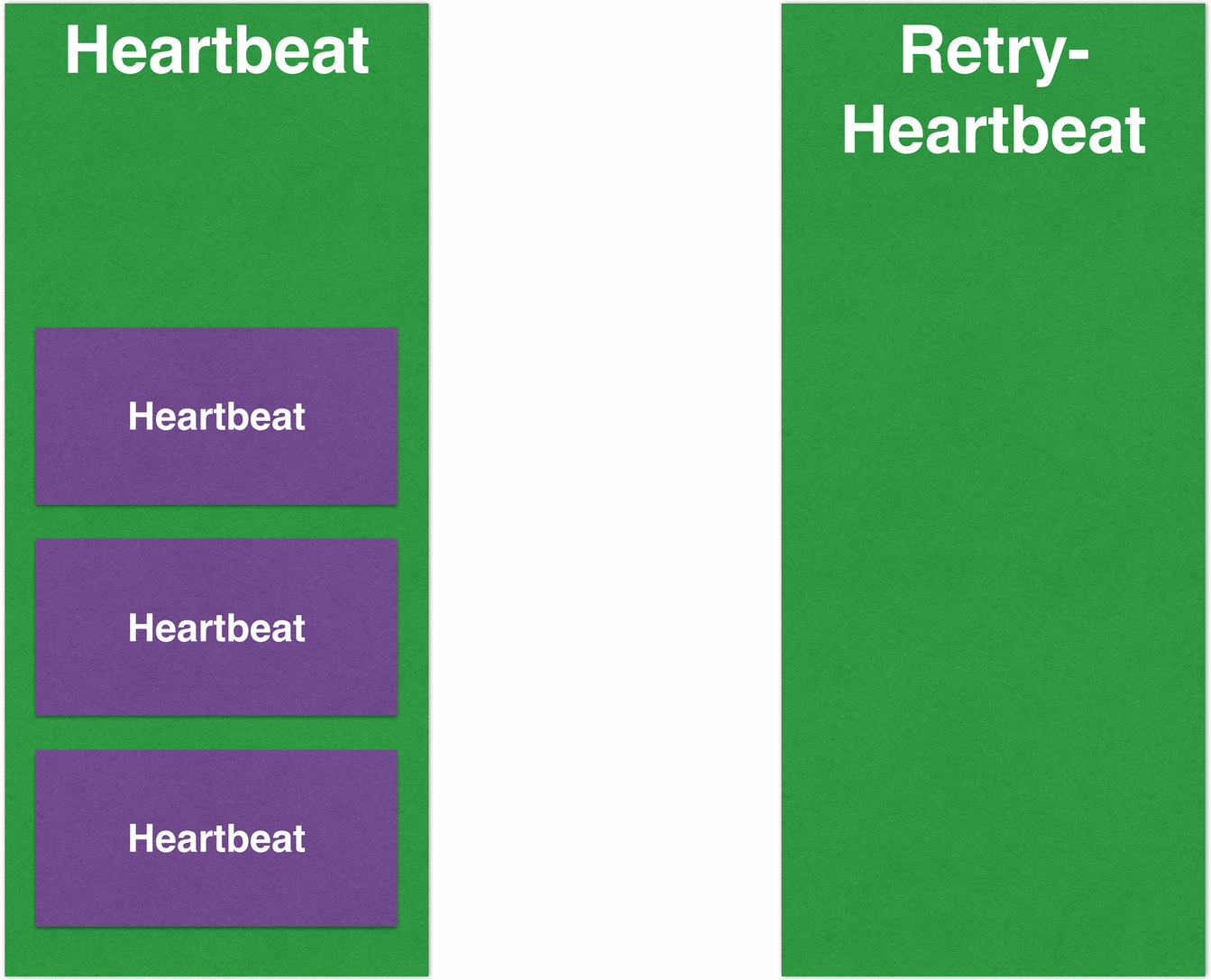

Smart Lock
Exchange
Check user PIN
Lock Events
Heartbeat
Heartbeat
Heartbeat

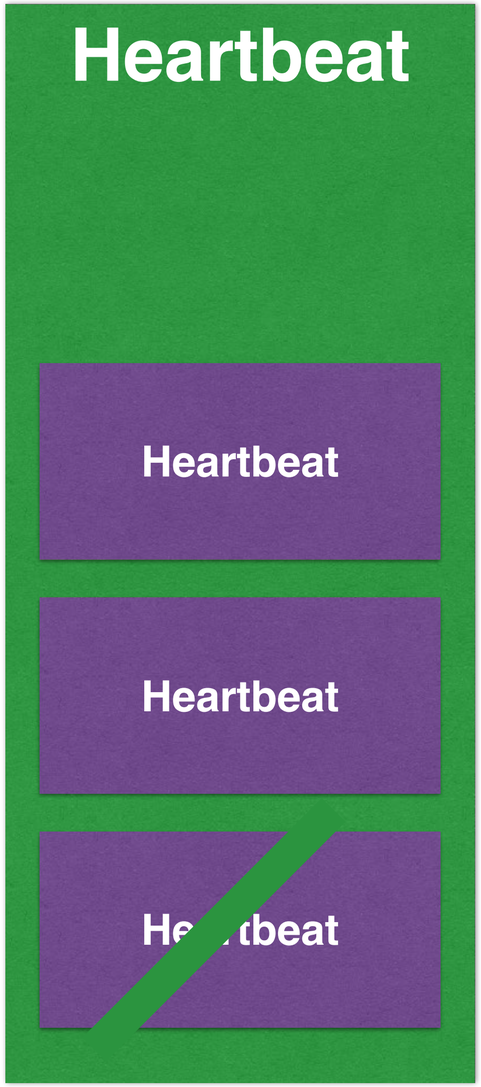

Smart Lock
Exchange
Lock Events
Heartbeat
Heartbeat
Heartbeat
Check user PIN

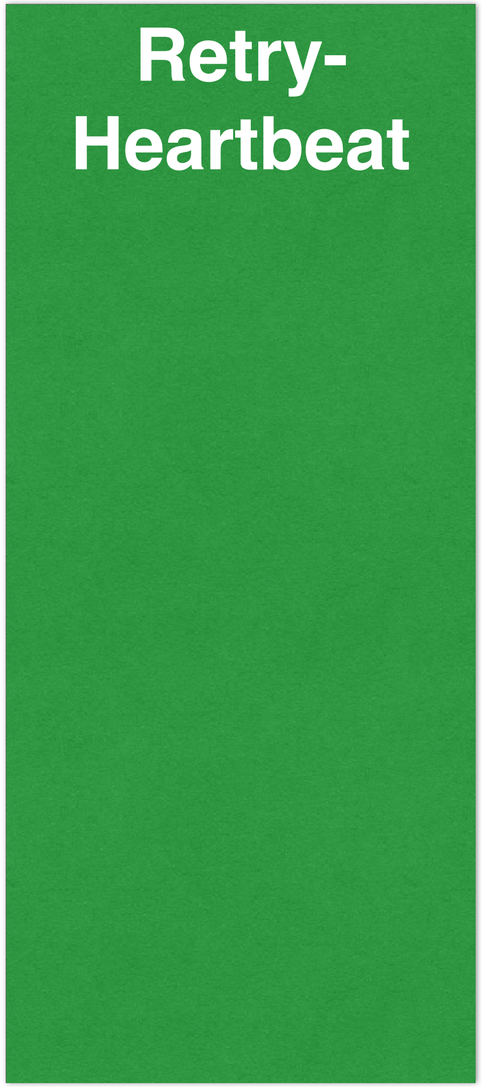

Smart Lock
Exchange
Lock Events
Heartbeat
Heartbeat
Heartbeat
Valid PIN

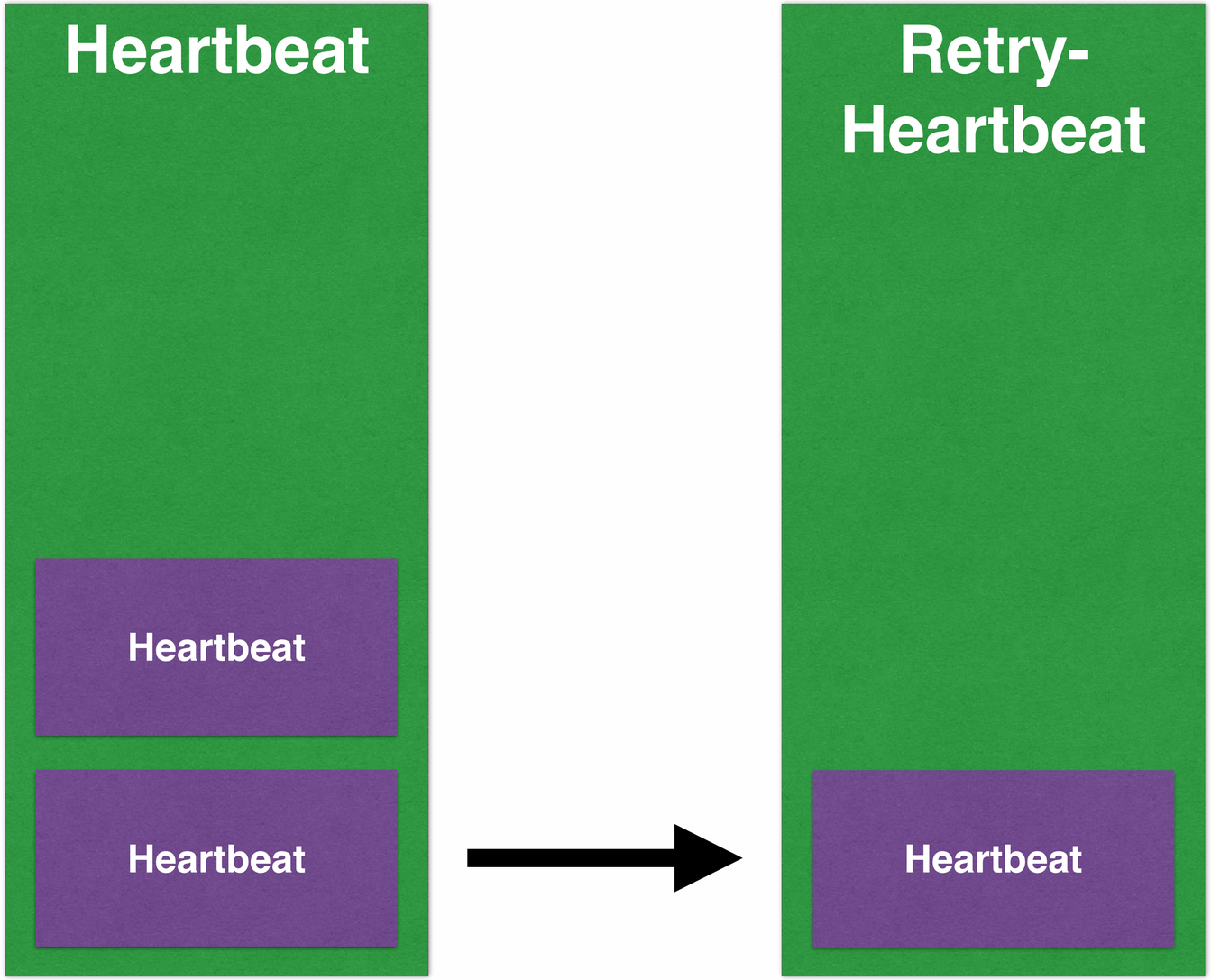

Smart Lock
Valid PIN
Exchange
Lock Events
Heartbeat
Heartbeat
Heartbeat

Dead lettering

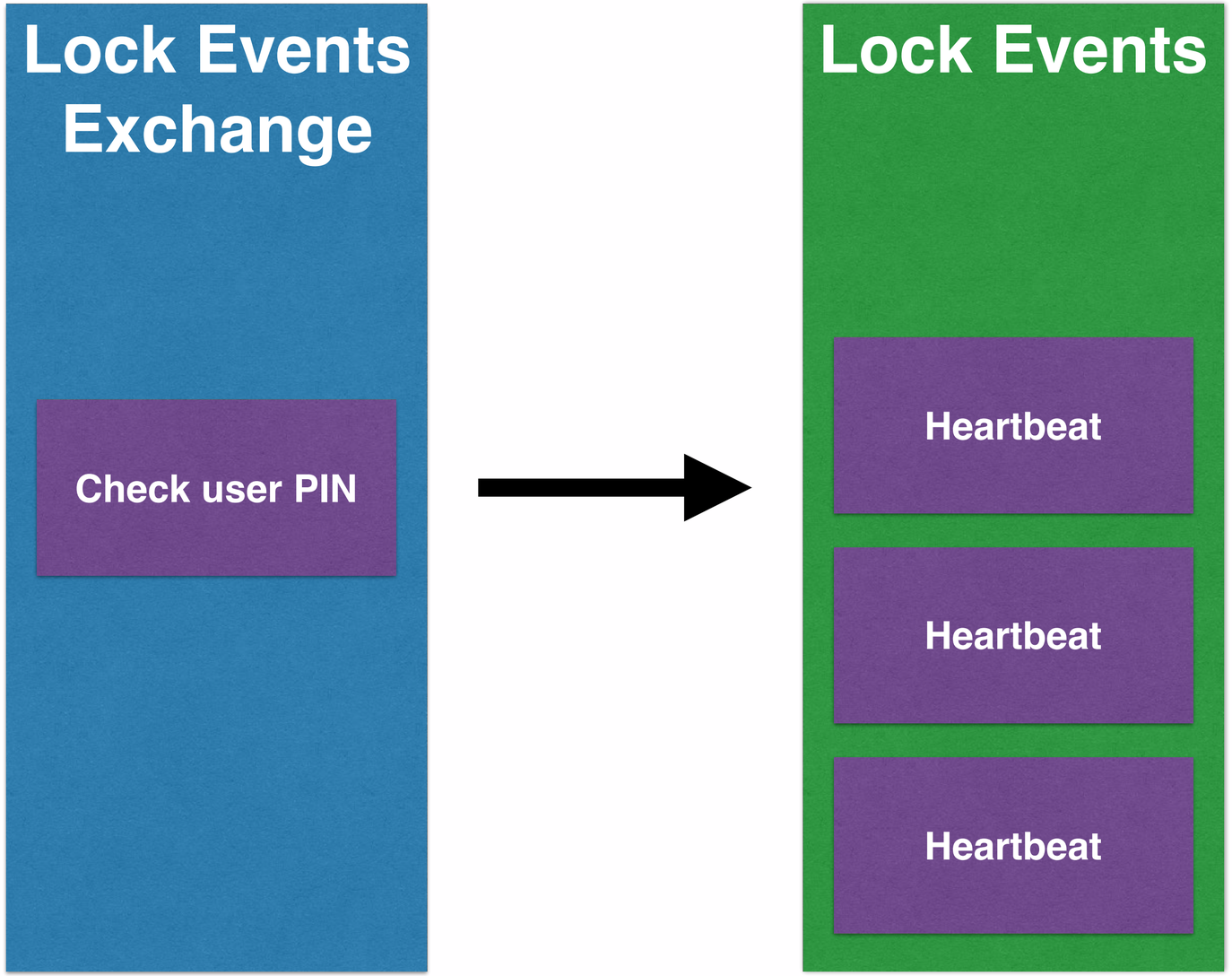

Heartbeat
Heartbeat
Heartbeat
Heartbeat
Retry-
Heartbeat

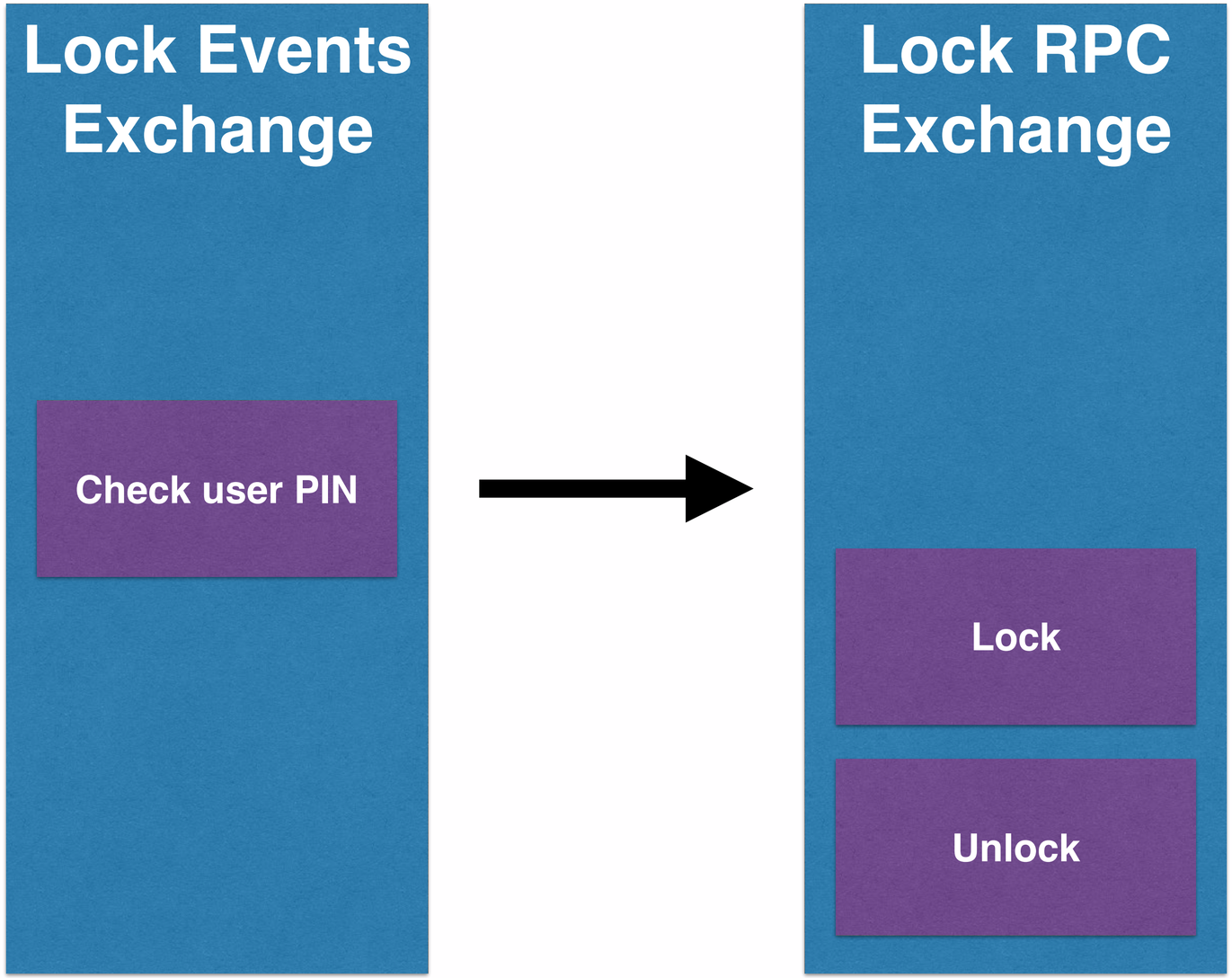

Heartbeat
Heartbeat
Heartbeat
Heartbeat

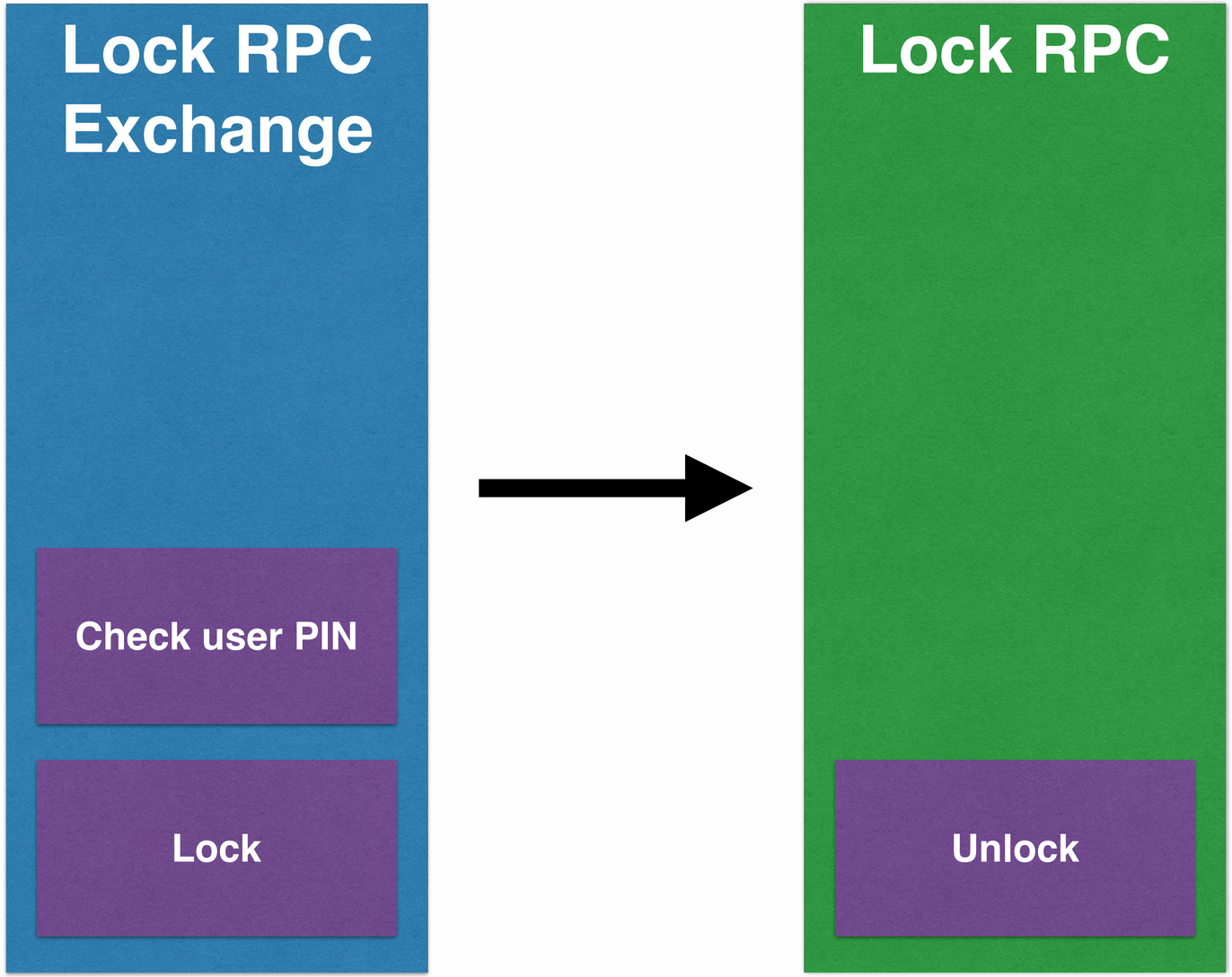

Retry-
Heartbeat


Heartbeat
Heartbeat
Heartbeat
Retry-
Heartbeat
Heartbeat

Alternate exchanges


Lock Events Exchange
Check user PIN
Lock Events
Heartbeat
Heartbeat
Heartbeat


Lock Events Exchange
Check user PIN
Lock RPC Exchange
Lock
Unlock


Lock RPC Exchange
Check user PIN
Lock
Lock RPC
Unlock

Priority consumers

Priority queues

TTL

04 PLUGINS

Management

RabbitMQ

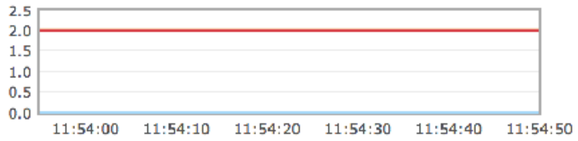
User: Log out
Cluster: rabbit@46e1483dee20 (change)
RabbitMQ 3.6.10, Erlang 19.3

Overview | Connections | Channels | Exchanges | Queues | Admin

Overview

Totals

Queued messages (chart: last minute) (?)



Ready	2
Unacked	0
Total	2

Message rates (chart: last minute) (?)

Publish	0.00/s	Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s	Disk read	0.00/s
Publisher confirm	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s	Disk write	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Return	0.00/s		

Global counts (?)

Connections: 12 | Channels: 12 | Exchanges: 17 | Queues: 8 | Consumers: 12

Node

Node: rabbit@46e1483dee20 **(More about this node)**

File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Rates mode	Info	Reset stats DB	+/-
39 1048576 available	12 943626 available	474 1048576 available	77MB 800MB high watermark	25GB 48MB low watermark	basic	Disc 1	Reset	

Reset stats on all nodes

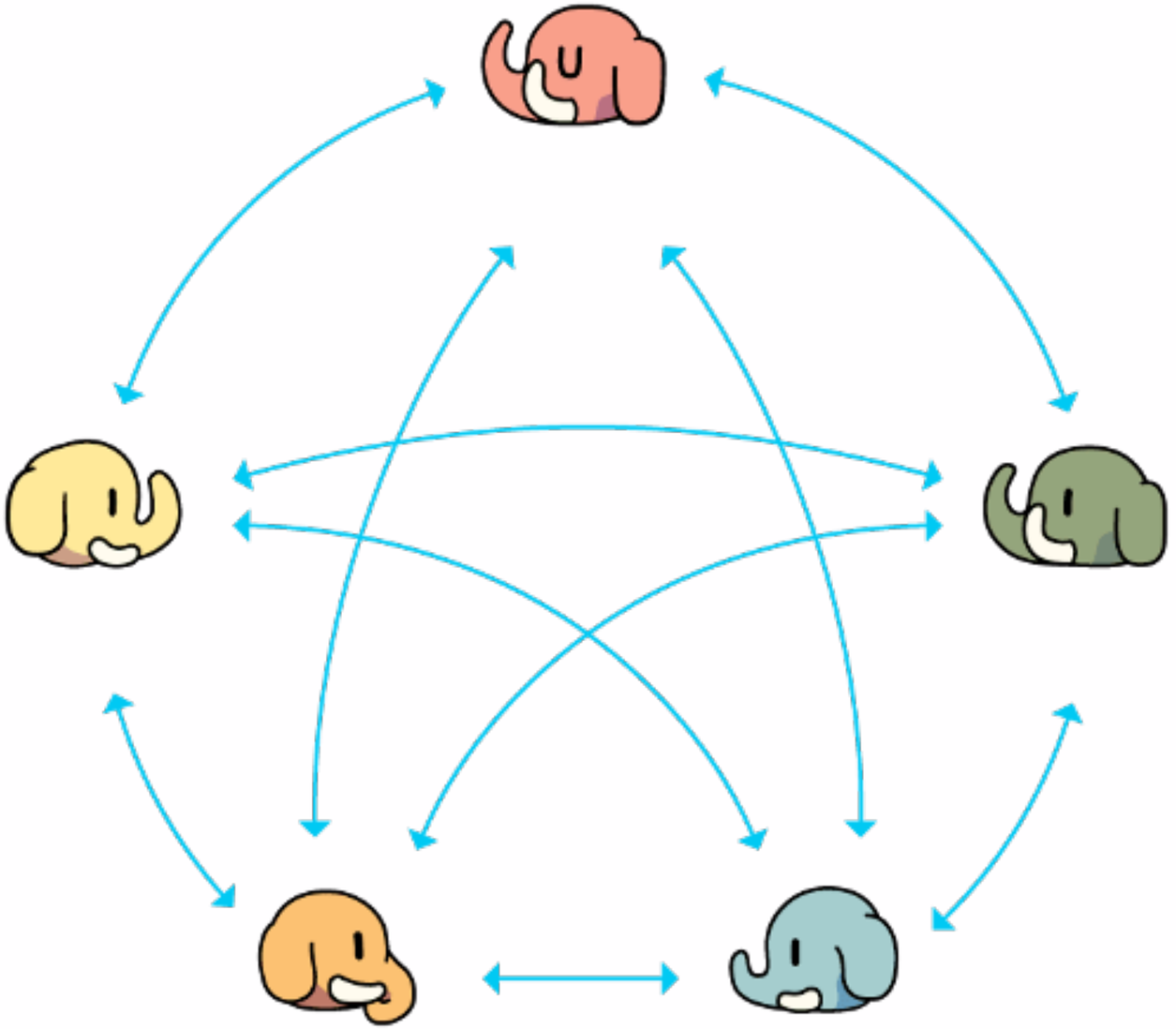
Paths

Config file	/etc/rabbitmq/rabbitmq.config
Database directory	/var/lib/rabbitmq/mnesia/rabbit@46e1483dee20
Log file	tty
SASL log file	tty

Ports and contexts

MQTT, STOMP and WebSockets

Federation



05

CONCLUSION

Debugging

Load balancing / Scaling

Memory consumption

Should I use it on my next project?

Monolith?
Probably not…

Services?

YES!

QUESTIONS

 github.com/stankec

 @monorkin

 hi@stanko.io