Reactor Pattern
&
Event-Driven Programming

A scalable concurrent approach,
using EventMachine with Thin as an example

Lin Jen-Shin, http://godfat.org/

Reactor Pattern
&
Event-Driven Programming
2

A scalable concurrent approach,
using EventMachine with Thin as an example

Lin Jen-Shin, http://godfat.org/

Reactor Pattern
&
Event-Driven Programming
2

http://godfat.org/slide/2010-04-13-reactor-pattern-and-2.pdf

Lin Jen-Shin, http://godfat.org/

Table of Contents

- concurrency, why and how in network

- Event-Driven Programming explained in Flash with Ruby syntax

- Reactor Pattern in EventMachine with Thin

- how Thin works

- how EventMachine works

Event-Driven Programming

```
                          register method(:do_something)
loop{                     loop{
  # you control the flow    # event loop control the flow,
  do_something              # later it calls your callback
}                           event = pop_event_queue
                            dispatch event if event
                          }
```

Reactor Pattern

```
loop{
    data = read
    handle data
}
```

```
register method(:handle)
loop{
    data = partial_read
    event = process data
    dispatch event if event
}
```

Table of Contents

- how Thin works

- how EventMachine works

Table of Contents

- how Thin works

- how EventMachine works

Table of Contents

- how Thin works

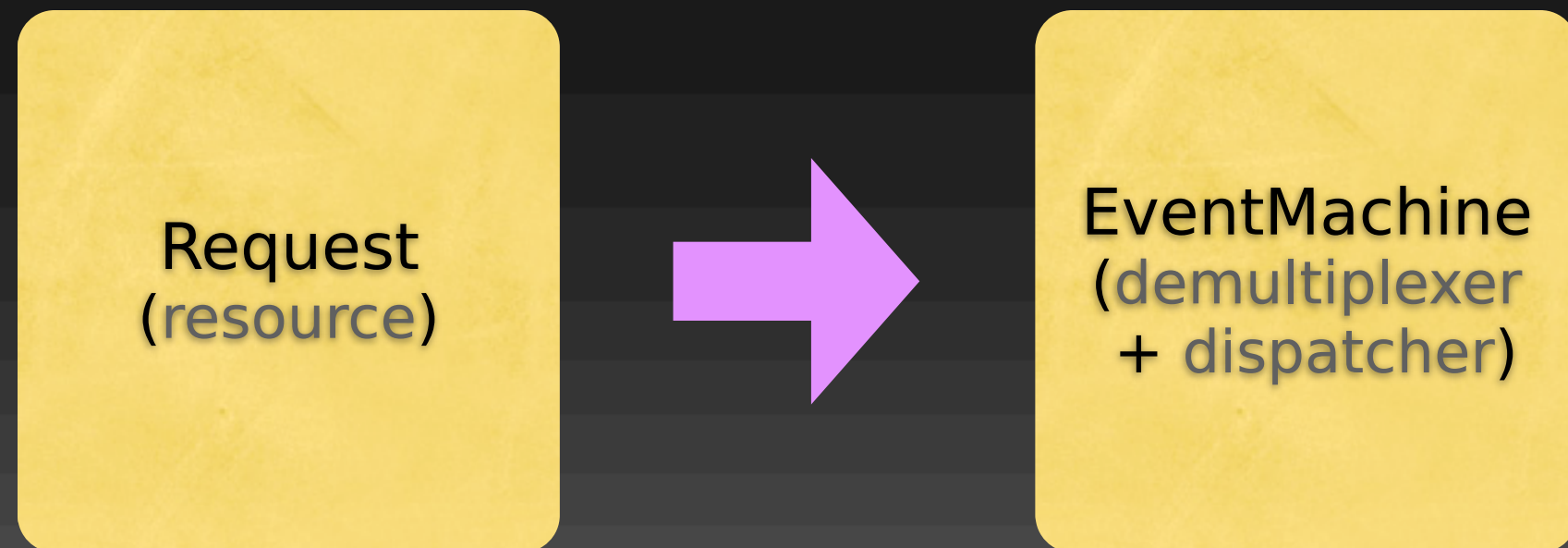- how EventMachine works

- how AMQP works

Table of Contents

- how Thin works

- how EventMachine works

- how AMQP works
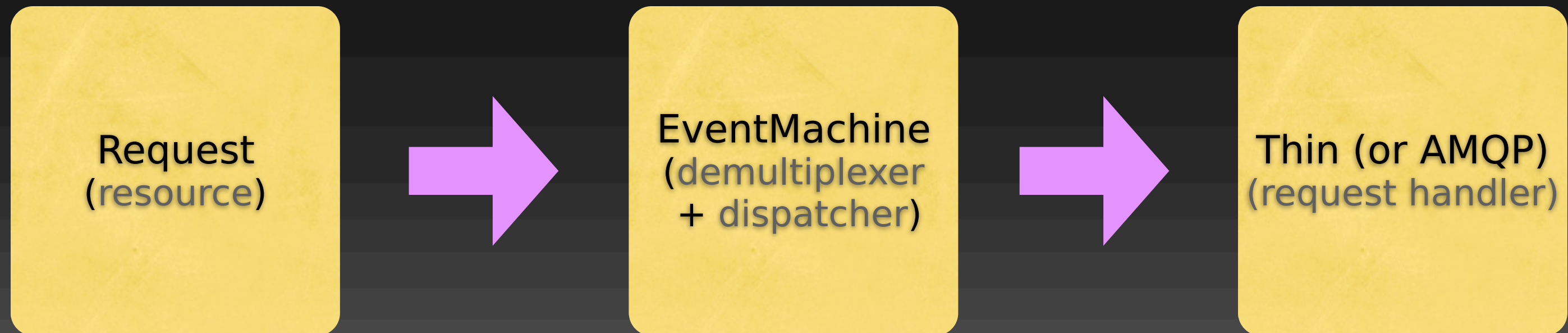
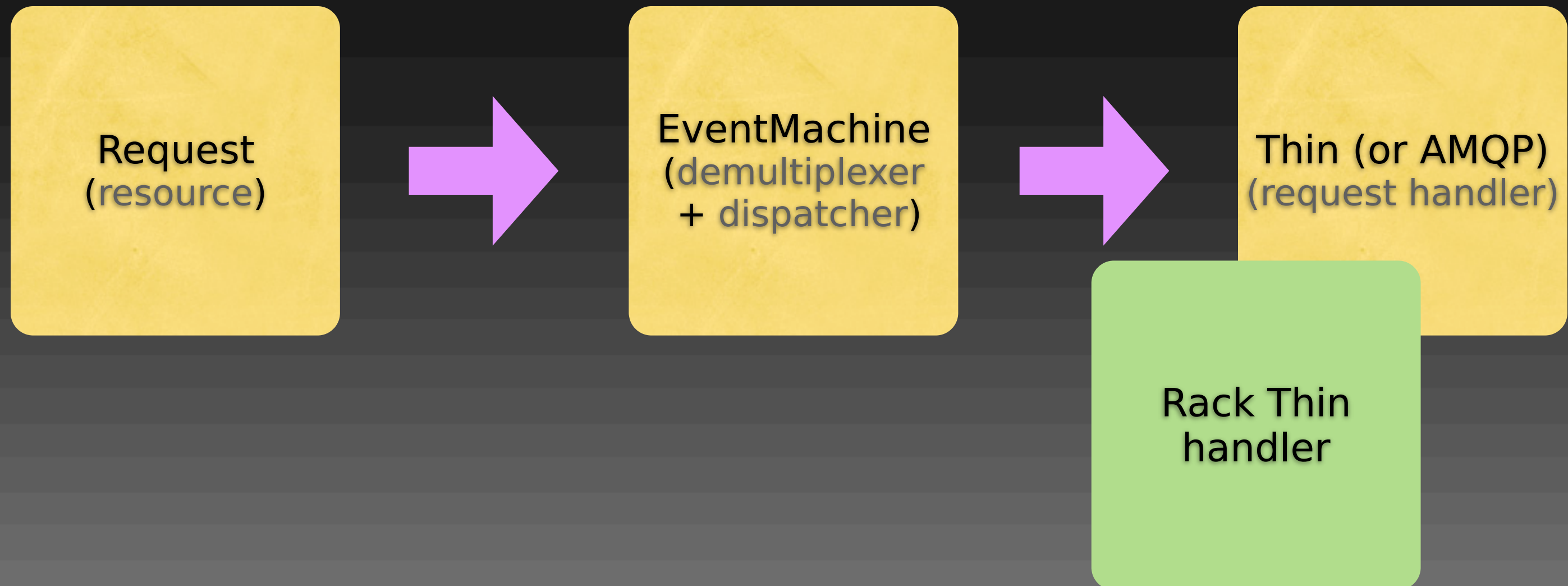- how Unicorn and Rainbows! works

Reactor Pattern

Request
(resource)

Reactor Pattern
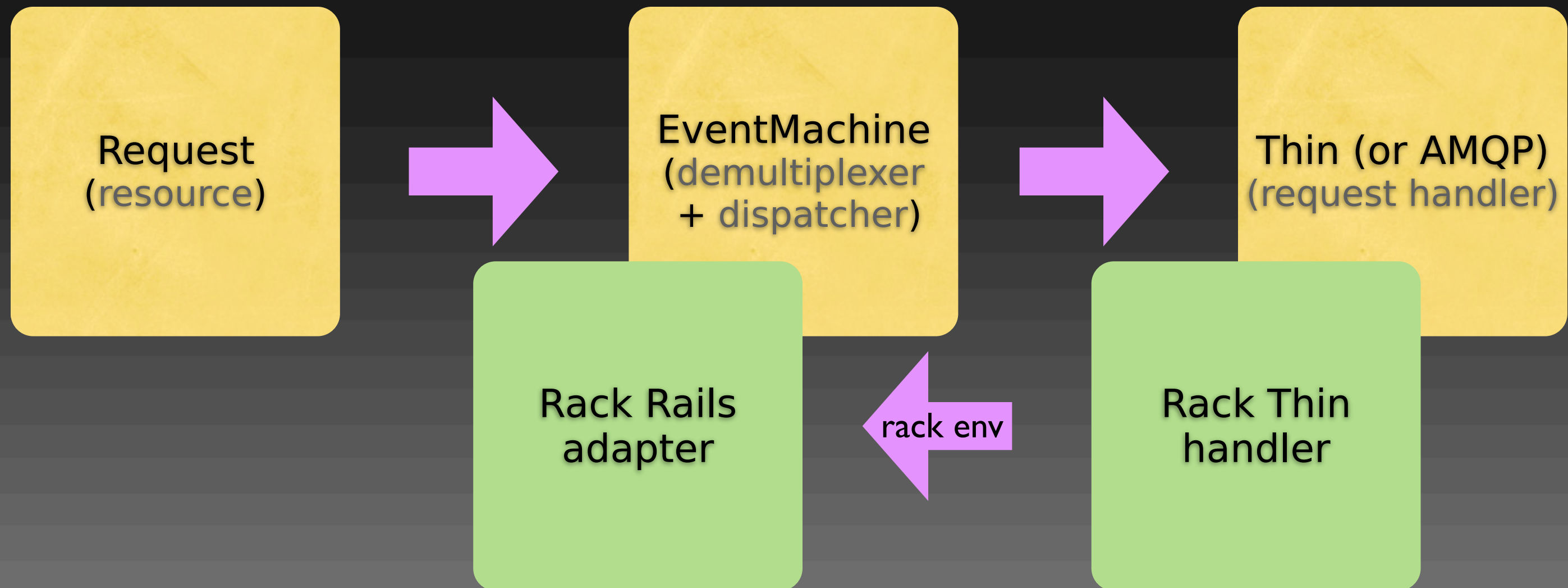
Request
(resource)

EventMachine
(demultiplexer
+ dispatcher)

Thin (or AMQP)
(request handler)

Reactor Pattern

Request
(resource)

→

EventMachine
(demultiplexer
+ dispatcher)

→

Thin (or AMQP)
(request handler)

Rack Thin
handler

Reactor Pattern

Request
(resource)

EventMachine
(demultiplexer
+ dispatcher)

Thin (or AMQP)
(request handler)

Rack Rails
adapter

rack env

Rack Thin
handler

Reactor Pattern

Request
(resource)

→

EventMachine
(demultiplexer
+ dispatcher)

→

Thin (or AMQP)
(request handler)

Rails

←

**Rack Rails
adapter**

← rack env

**Rack Thin
handler**

Reactor Pattern

your rails application

Request
(resource)

EventMachine
(demultiplexer
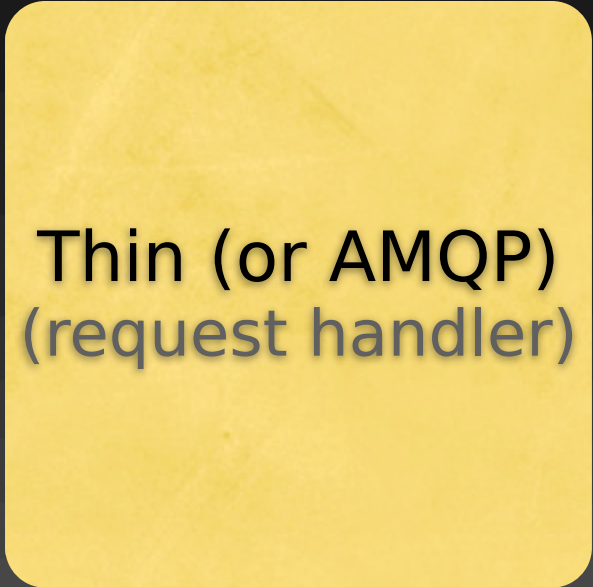+ dispatcher)

Thin (or AMQP)
(request handler)

Rails

Rack Rails
adapter

rack env

Rack Thin
handler

how Thin works

Thin (or AMQP)
(request handler)

how Thin works

- Thin::Server

how Thin works

- Thin::Server

- Thin::Backends::TcpServer
  # communicate with EventMachine

how Thin works

- `Thin::Server`

- `Thin::Backends::TcpServer`
  `# communicate with EventMachine`

- `Thin::Connection`
  `# EventMachine event handler`

how Thin works

- Thin::Server

- Thin::Backends::TcpServer
  # communicate with EventMachine

- Thin::Connection
  # EventMachine event handler

- Thin::Request
  # partial HTTP request parsing
  # Rack env builder

how Thin works

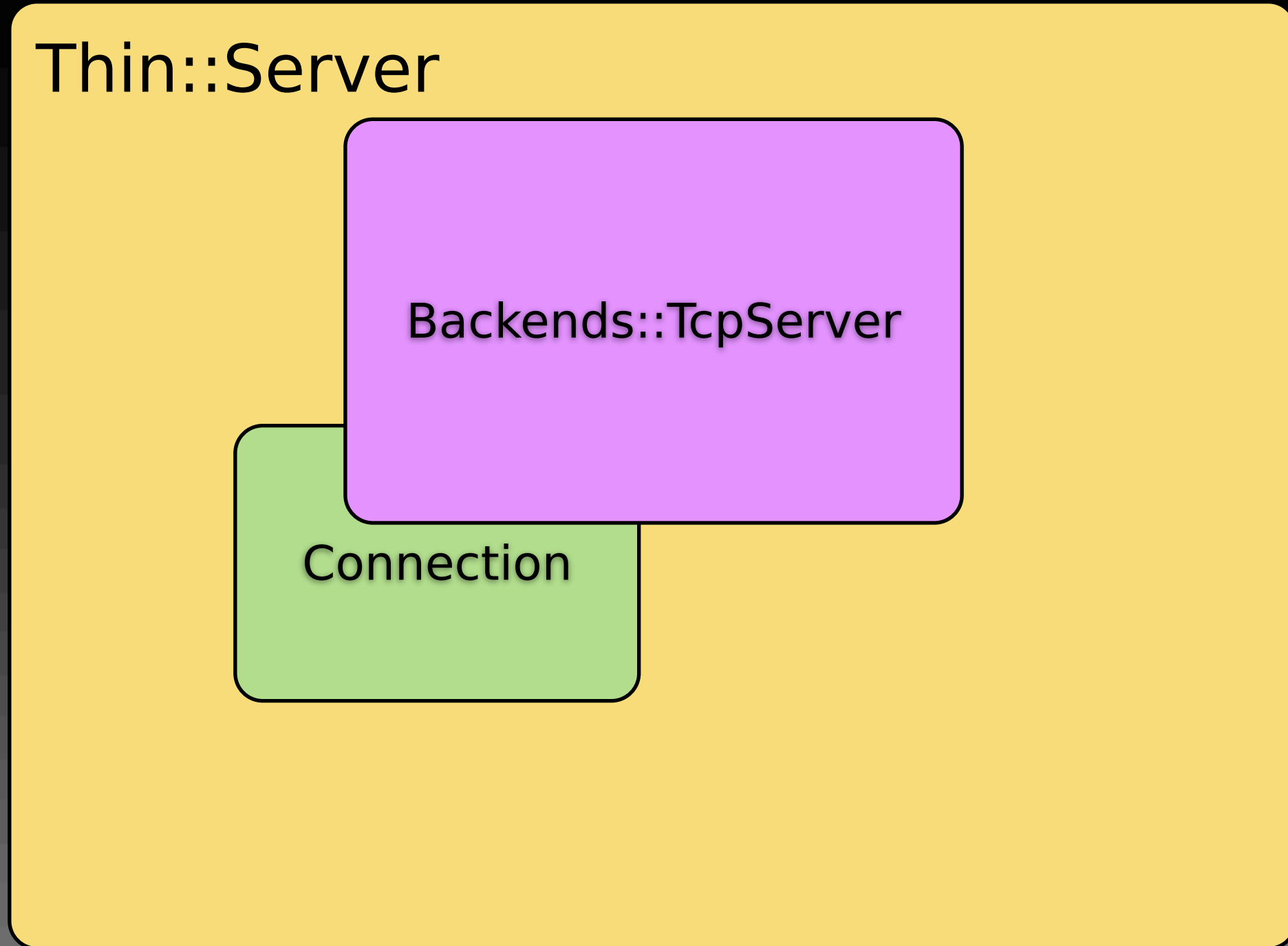Thin::Server

how Thin works

Thin::Server

Backends::TcpServer

how Thin works

Thin::Server

Backends::TcpServer

Connection | ection | ection
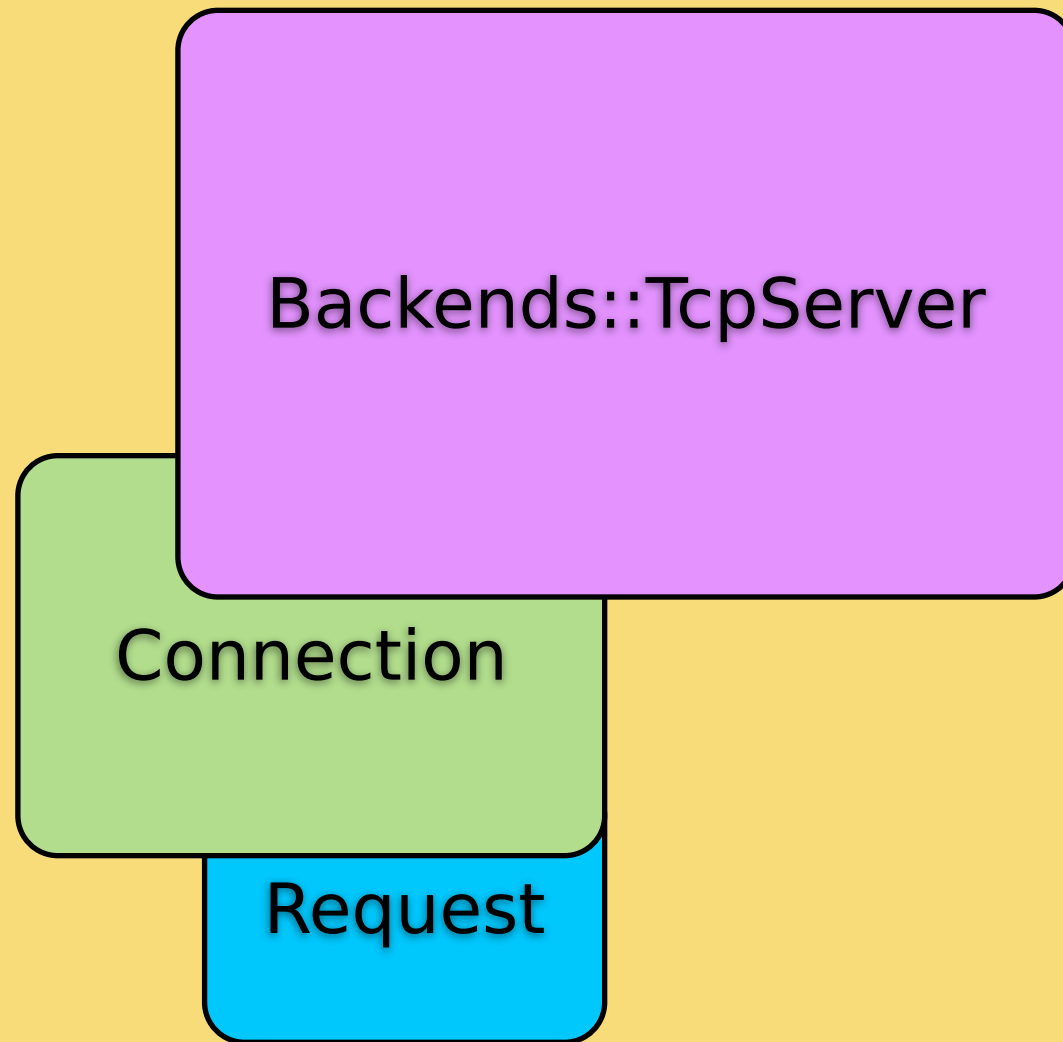
Request | Request | Request

how Thin works
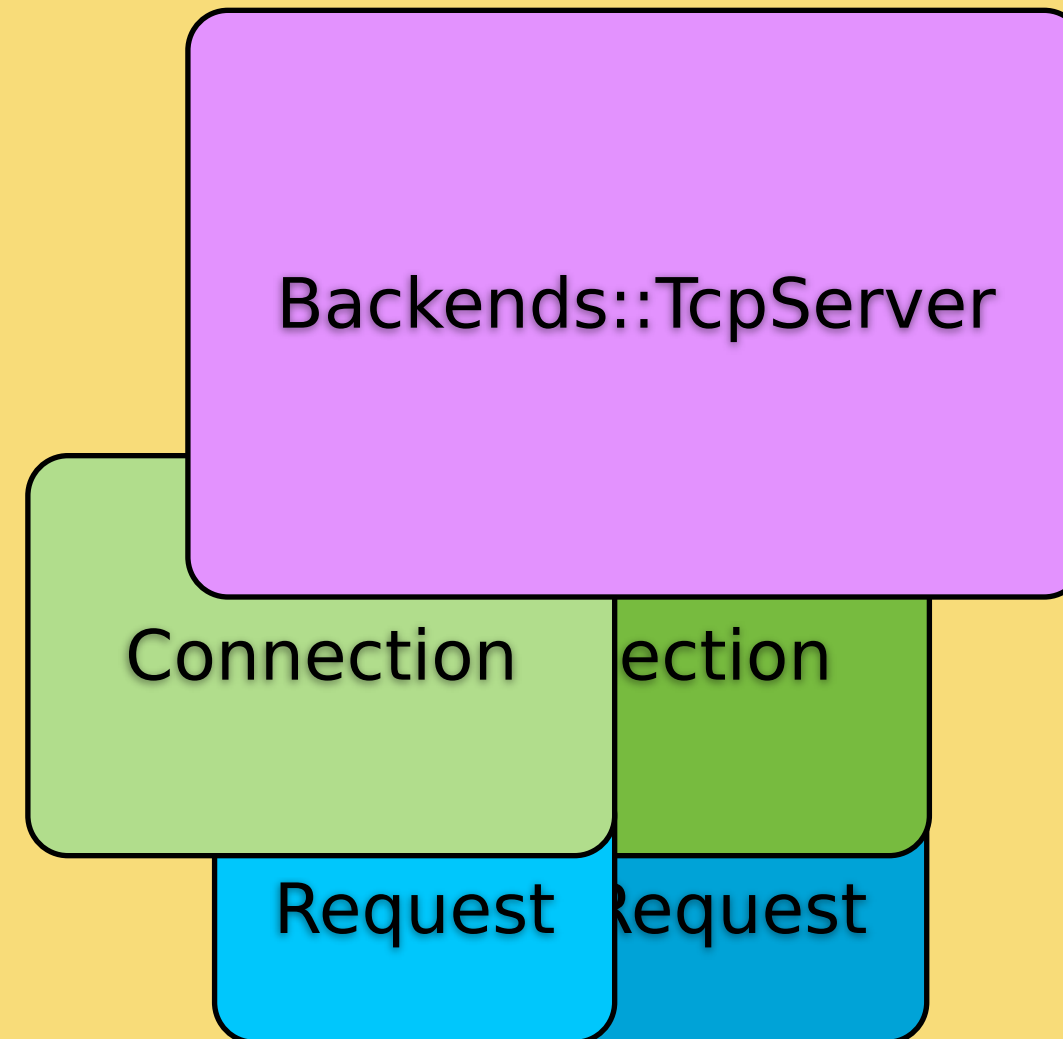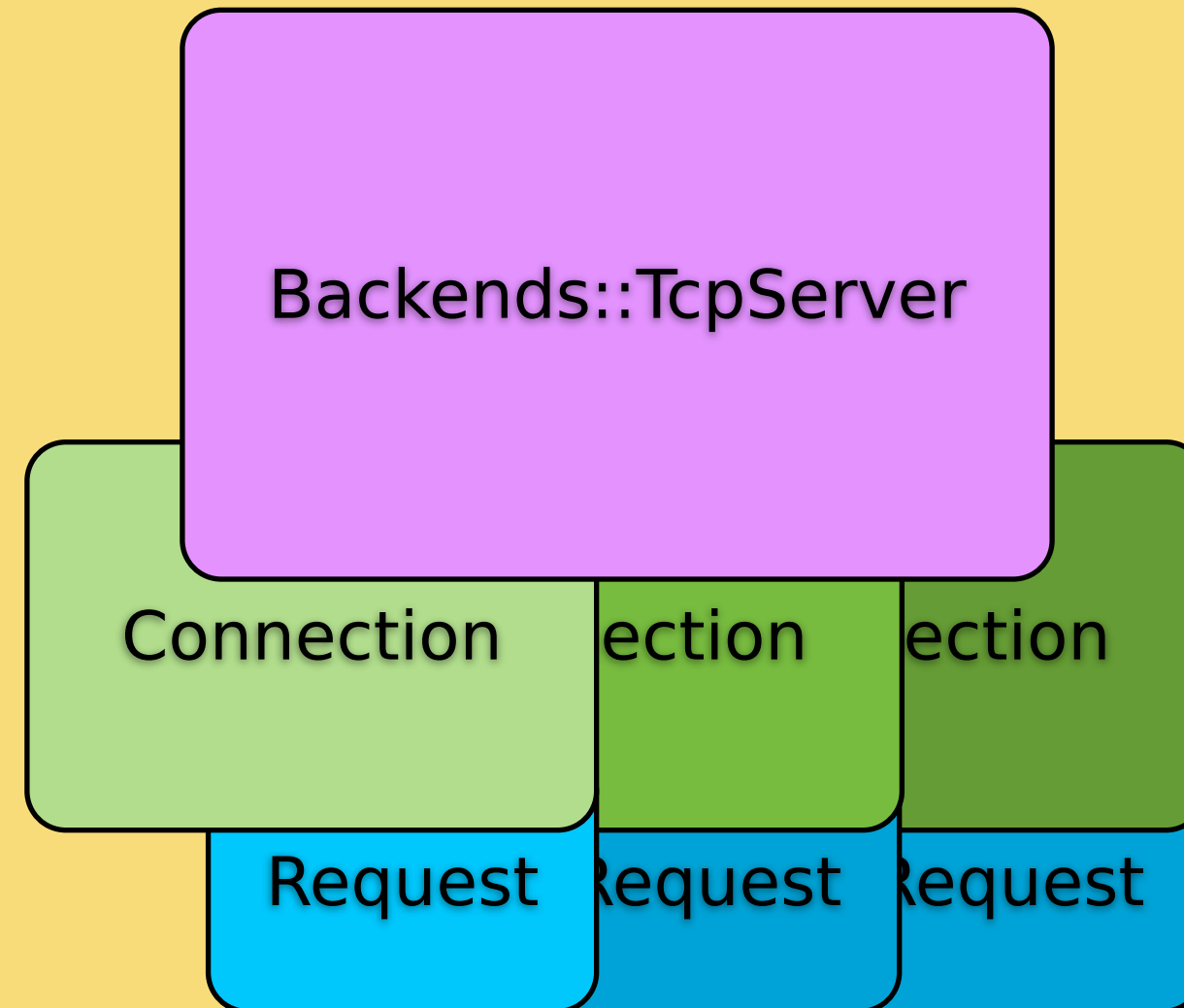
thin 1.2.7 codename No Hup

how Thin works
thin 1.2.7 codename No Hup

```ruby
# in lib/thin/backends/tcp_server.rb:16
# in Thin::TcpServer#connect

EventMachine.start_server(
  @host, @port,
  Thin::Connection,
  &method(:initialize_connection))

              # rack app, backend ref, timeout, etc
```

how Thin works
thin 1.2.7 codename No Hup

```ruby
# in lib/thin/connection.rb:42
# in Thin::Connection#receive_data



process if @request.parse(data)



          #  true: parsed, so process!
          # false:  we need more data!
```

how Thin works
thin 1.2.7 codename No Hup

```ruby
# in lib/thin/request.rb:82
# in Thin::Request#parse


@request = @parser.execute(@env, @data, @nparsed)



                        #     @env: Rack env
                        #    @data: HTTP header buffer
                        # @nparsed: index of parsed data
```

how Thin works

```
// in ext/thin_parser/thin.c:335
// in thin.c#Thin_HttpParser_execute


thin_http_parser_execute(http, dptr, dlen, from);



                      // http: HTTP parser pointer
                      // dptr: HTTP header data pointer
                      // dlen: HTTP header data length
                      // form: previous @nparsed
```

how Thin works

```
// in ext/thin_parser/parser.rl:102
// in parser.rl#thin_http_parser_execute
// (it's mongrel's http parser)

size_t thin_http_parser_execute(
  http_parser *parser, const char *buffer,
  size_t len, size_t off)
```

how Thin works

Ragel is a finite state machine compiler with output support for C, C++, Objective-C, D, Java and Ruby source code.

how Thin works

Ragel is a finite state machine compiler with output support for C, C++, Objective-C, D, Java and Ruby source code.

- Mongrel HTTP parser

- Hpricot HTML/XML parser

- JSON parser

how Thin works
thin 1.2.7 codename No Hup

```ruby
# in lib/thin/connection.rb:42
# in Thin::Connection#receive_data



process if @request.parse(data)



                      #  true: parsed, so process!
                      # false:  we need more data!
```

how Thin works
thin 1.2.7 codename No Hup

```ruby
# in lib/thin/connection.rb:52
# in Thin::Connection#process

if threaded?
  @request.threaded = true
  EventMachine.defer(method( :pre_process),
                     method(:post_process))
else
  @request.threaded = false
  post_process(pre_process)
end
```

how EventMachine works

eventmachine 0.12.10

```ruby
# in lib/eventmachine.rb:1045
# in EventMachine.defer

unless @threadpool
  require 'thread'
  @threadpool  = []
  @threadqueue = ::Queue.new
  @resultqueue = ::Queue.new
  spawn_threadpool
end
@threadqueue << [op||blk,callback]
```

how Thin works
thin 1.2.7 codename No Hup

```ruby
        # in lib/thin/connection.rb:68
        # in Thin::Connection#pre_process

@request.async_callback = method(:post_process)
# ...
response = AsyncResponse
catch(:async) do
  # Process the request calling the Rack adapter
  response = @app.call(@request.env)
end
response
```

how Thin works
thin 1.2.7 codename No Hup

```ruby
# in lib/thin/connection.rb:95
# in Thin::Connection#post_process
        @response.status,
        @response.headers,
        @response.body = *result
        # ...
        @response.each do |chunk|
          trace { chunk }
          send_data chunk
        end
```

Reactor Pattern

- resources

- synchronous event demultiplexer

- dispatcher

- request handler (Thin::Connection)

by wikipedia

Table of Contents

- how Thin works

- how EventMachine works

- how AMQP works

- how Unicorn and Rainbows! works

how EventMachine works

eventmachine 0.12.10

how EventMachine works

eventmachine 0.12.10

```ruby
# in lib/eventmachine.rb:571
# in EventMachine.start_server

s = if port
      start_tcp_server server, port
    else
      start_unix_server server
    end
@acceptors[s] = [klass,args,block]
          #     s: server (in Reactor) uuid
          # klass: Thin::Connection
          #  args: []
          # block: method(:initialize_connection)
```

how EventMachine works

```
# in lib/eventmachine.rb:50

case $eventmachine_library
  when :pure_ruby
    require 'pr_eventmachine'
  when :extension
    require 'rubyeventmachine'
  when :java
    require 'jeventmachine'
```

how EventMachine works

eventmachine 0.12.10

```ruby
# in lib/pr_eventmachine.rb:318
# in EventMachine.run

loop {
  @current_loop_time = Time.now
  break if @stop_scheduled
  run_timers # timer event
  break if @stop_scheduled
  # epoll, kqueue, etc
  crank_selectables
  break if @stop_scheduled
  # close scheduling if client timeout
  run_heartbeats
}
```

how EventMachine works

```ruby
    # in lib/eventmachine.rb:1445
    # in EventMachine.event_callback

elsif opcode == ConnectionData
  c = @conns[conn_binding] or raise ConnectionNotBound,
    "received data #{data} for unknown signature:" \
    "#{conn_binding}"
  c.receive_data data
elsif opcode == LoopbreakSignalled

                        #            opcode: event enum (int)
                        # conn_binding: connection uuid
                        #              data: received data
```

how Thin works
thin 1.2.7 codename No Hup

```ruby
# in lib/thin/connection.rb:42
# in Thin::Connection#receive_data



    process if @request.parse(data)



                    #  true: parsed, so process!
                    # false:  we need more data!
```

how EventMachine works

eventmachine 0.12.10

```ruby
    # in lib/eventmachine.rb:1427
    # in EventMachine.event_callback
elsif opcode == ConnectionAccepted
  accep,args,blk = @acceptors[conn_binding]
  raise NoHandlerForAcceptedConnection unless accep
  c = accep.new data, *args
  @conns[data] = c
  blk and blk.call(c)
  c # (needed?)
elsif opcode == ConnectionCompleted
                # conn_binding: server uuid
                #        data: connection uuid
```

how Thin works
thin 1.2.7 codename No Hup

```ruby
# in lib/thin/backends/tcp_server.rb:16
# in Thin::TcpServer#connect

EventMachine.start_server(
    @host, @port,
    Thin::Connection,
    &method(:initialize_connection))


                    # rack app, backend ref, timeout, etc
```

how EventMachine works

eventmachine 0.12.10

```ruby
# in lib/pr_eventmachine.rb:256

module EventMachine
  TimerFired           = 100
  ConnectionData       = 101
  ConnectionUnbound    = 102
  ConnectionAccepted   = 103
  ConnectionCompleted  = 104
  LoopbreakSignalled   = 105
end
```

Table of Contents

- how Thin works

- how EventMachine works

- how AMQP works

- how Unicorn and Rainbows! works

how AMQP works

- AMQP::BasicClient
  # extend to AMQP::Client

how AMQP works

- AMQP::BasicClient
  # extend to AMQP::Client

- AMQP::Client
  # included into EventMachine::Connection

how AMQP works

amqp 0.6.7

how AMQP works
amqp 0.6.7

```ruby
# in lib/amqp.rb:79
# in AMQP.start

EM.run{
  @conn ||= connect *args
  @conn.callback(&blk) if blk
  @conn
}
```

how AMQP works
amqp 0.6.7

```
# in lib/amqp.rb:18
# in AMQP.connect

Client.connect *args
```

how AMQP works
amqp 0.6.7

```ruby
# in lib/amqp/client.rb:188
# in AMQP::Client.connect

opts = AMQP.setting.merge(opts)
EM.connect opts[:host], opts[:port], self, opts
```

how Thin works
thin 1.2.7 codename No Hup

```
# in lib/thin/backends/tcp_server.rb:16
# in Thin::TcpServer#connect

EventMachine.start_server(
    @host, @port,
    Thin::Connection,
    &method(:initialize_connection))


            # rack app, backend ref, timeout, etc
```

how EventMachine works

```ruby
  # in lib/eventmachine.rb:1571
  # in EventMachine.klass_from_handler
klass = if handler and handler.is_a?(Class)
  raise ArgumentError,
    "must provide module or #{klass.name}" unless
    klass >= handler
  handler
elsif handler
  Class.new(klass){ include handle }
else
  klass      #   klass: EventMachine::Connection
end
  # handler: Thin::Connection or AMQP::Client
```

how AMQP works
amqp 0.6.7

```ruby
# in lib/amqp/client.rb:115
# in AMQP::Client#receive_data

while frame = Frame.parse(@buf)
  log 'receive', frame
  process_frame frame
end
```

how AMQP works

- AMQP::Frame
  # basic building block of AMQP data stream

how AMQP works

- AMQP::Frame
  # basic building block of AMQP data stream

- AMQP::Buffer
  # frame buffer and parser

how AMQP works

- AMQP::Frame
  # basic building block of AMQP data stream

- AMQP::Buffer
  # frame buffer and parser

- AMQP::Protocol::Connection
  # used in BasicClient#process_frame

how AMQP works

- MQ
  # easy to use, high level wrapper

how AMQP works

- MQ
  # easy to use, high level wrapper

- MQ::Queue
  # the entities which receive messages

how AMQP works

- `MQ`
  `# easy to use, high level wrapper`

- `MQ::Queue`
  `# the entities which receive messages`

- `MQ::Exchange`
  `# the entities to which messages are sent`

how AMQP works

- `MQ`
  `# easy to use, high level wrapper`

- `MQ::Queue`
  `# the entities which receive messages`

- `MQ::Exchange`
  `# the entities to which messages are sent`

by wikipedia

how AMQP works

```
# default connection
MQ.new.queue('name')

# default exchange (direct)
MQ.new.publish('name')

#-- convenience wrapper (read: HACK)
#    for thread-local MQ object
MQ.queue('name')
MQ.publish('name')
```

how AMQP works

```
MQ.queues     # all created queues
MQ.exchanges  # all created exchanges
MQ.direct     #  direct exchange
MQ.fanout     #  fanout exchange
MQ.topic      #   topic exchange
MQ.headers    # headers exchange
```

Table of Contents

- how Thin works

- how EventMachine works

- how AMQP works

- how Unicorn and Rainbows! works

Unicorn?

Unicorn?

- is not event-driven!

Unicorn?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

Unicorn?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

Unicorn?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

Unicorn? Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

Unicorn? Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- could be event-driven

Unicorn?   Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- could be event-driven

- also pure Ruby, except...

Unicorn? | Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- could be event-driven

- also pure Ruby, except...

- *any* concurrency model

Unicorn? Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- could be event-driven

- also pure Ruby, except...

- *any* concurrency model

- provide network concurrency

Unicorn?

Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- RevFiberSpawn

Unicorn? Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- RevFiberSpawn

- Revactor

Unicorn?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

Rainbows!?

- RevFiberSpawn

- Revactor

- ThreadPool

Unicorn? Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- RevFiberSpawn

- Revactor

- ThreadPool

- Rev

Unicorn?

Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- RevFiberSpawn

- Revactor

- ThreadPool

- Rev

- ThreadSpawn

Unicorn?

Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- RevFiberSpawn

- Revactor

- ThreadPool

- Rev

- ThreadSpawn

- EventMachine

Unicorn?

Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- RevFiberSpawn
  - RevThreadSpawn
- Revactor

- ThreadPool

- Rev

- ThreadSpawn

- EventMachine

Unicorn? Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- RevFiberSpawn
- RevThreadSpawn
- Revactor
- FiberSpawn
- ThreadPool
- Rev
- ThreadSpawn
- EventMachine

Unicorn?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

Rainbows!?

- RevFiberSpawn
  - RevThreadSpawn
- Revactor
  - FiberSpawn
- ThreadPool
  - FiberPool
- Rev
- ThreadSpawn
- EventMachine

Unicorn? Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- RevFiberSpawn
  - RevThreadSpawn
- Revactor
  - FiberSpawn
- ThreadPool
  - FiberPool
- Rev
  - NeverBlock
- ThreadSpawn
- EventMachine

Unicorn?   Rainbows!?

- is not event-driven!

- except Mongrel HTTP parser, all written in Ruby

- yet *super fast* for fast client

- preforking worker with blocking I/O

- RevFiberSpawn
  - RevThreadSpawn
- Revactor
  - FiberSpawn
- ThreadPool
  - FiberPool
- Rev
  - NeverBlock
- ThreadSpawn
  - RevThreadPool
- EventMachine

Unicorn   Rainbows!

```
unicorn master
\_ unicorn worker[0]
|  \_ client[0]
\_ unicorn worker[1]
|  \_ client[1]
\_ unicorn worker[2]
|  \_ client[2]
...
\_ unicorn worker[M]
   \_ client[M]
```

Unicorn

```
unicorn master
\_ unicorn worker[0]
|  \_ client[0]
\_ unicorn worker[1]
|  \_ client[1]
\_ unicorn worker[2]
|  \_ client[2]

...
\_ unicorn worker[M]
   \_ client[M]
```

Rainbows!

```
rainbows! master
\_ rainbows! worker[0]
|  \_ client[0,0]
|  \_ client[0,1]
|  ...
|  \_ client[0,N]
\_ rainbows! worker[1]
|  \_ client[1,0]
|  ...
|  \_ client[1,N]
...
\_ rainbows! worker[M]
   \_ client[M,0]
   ...
   \_ client[M,N]
```
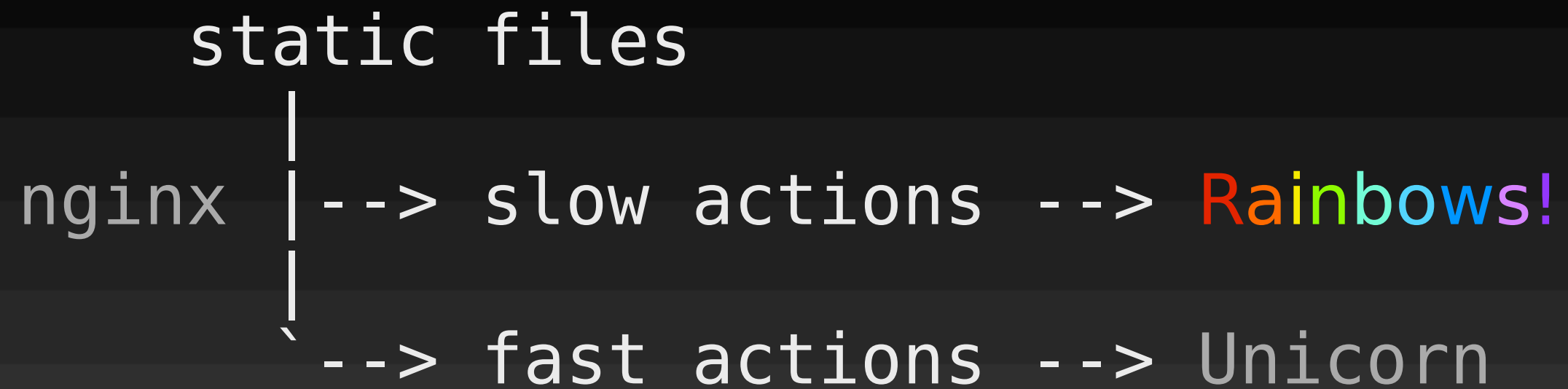
Unicorn

Rainbows!

```
unicorn master
  \_ unicorn worker[0]
  |   \_ client[0]
  \_ unicorn worker[1]
  |   \_ client[1]
  \_ unicorn worker[2]
  |   \_ client[2]

  ...

  \_ unicorn worker[M]
      \_ client[M]
```

```
rainbows! master
  \_ rainbows! worker[0]
  |   \_ client[0,0]------\      ___app[0]
  |   \_ client[0,1]-------\    /___app[1]
  |   \_ client[0,2]------->--<       ...
  |   ...                   ___/  `---app[P]
  |   \_ client[0,N]----/
  \_ rainbows! worker[1]
  |   \_ client[1,0]------\      ___app[0]
  |   \_ client[1,1]-------\    /___app[1]
  |   \_ client[1,2]------->--<       ...
  |   ...                   ___/  `---app[P]
  |   \_ client[1,N]----/
  \_ rainbows! worker[M]
      \_ client[M,0]------\      ___app[0]
      \_ client[M,1]-------\    /___app[1]
      \_ client[M,2]------->--<       ...
      ...                   ___/  `---app[P]
      \_ client[M,N]----/
```

Unicorn   Rainbows!

```
        static files
           |
           |
nginx  |--> slow actions --> Rainbows!
           |
           |
           `--> fast actions --> Unicorn
```

Unicorn

http://unicorn.bogomips.org/

Rainbows!

http://rainbows.rubyforge.org/

```
        static files
             |
             |
nginx  |--> slow actions --> Rainbows!
             |
             |
       `--> fast actions --> Unicorn
```
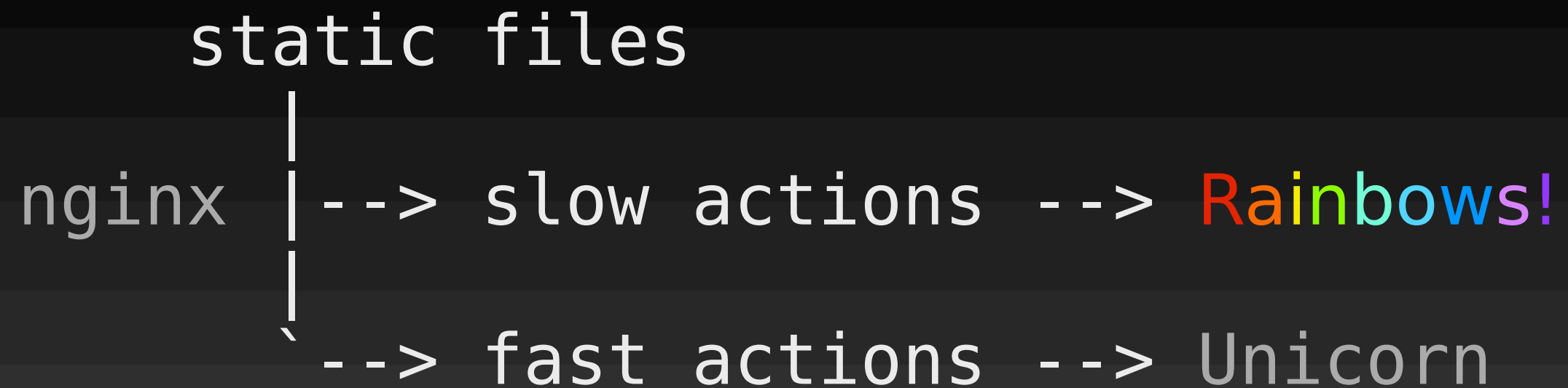
how Unicorn works

unicorn 0.97.0

how Unicorn works

unicorn 0.97.0

```
# in lib/unicorn.rb:270
# in Unicorn::HttpServer#start
```

maintain_worker_count

how Unicorn works

unicorn 0.97.0

```
# in lib/unicorn.rb:602
# in Unicorn::HttpServer#maintain_worker_count

(off = WORKER.size - worker_process) == 0 and return
 off < 0 and return spawn_missing_workers
```

how Unicorn works

unicorn 0.97.0

```ruby
# in lib/unicorn.rb:591
# in Unicorn::HttpServer#spawn_missing_workers

worker = Worker.new(worker_nr, Unicorn::Util.tmpio)
before_fork.call(self, worker)
WORKERS[fork {
  ready_pipe.close if ready_pipe
  self.ready_pipe = nil
  worker_loop(worker)
}] = worker
```

how Unicorn works

unicorn 0.97.0

```ruby
# in lib/unicorn.rb:705
# in Unicorn::HttpServer#worker_loop

ready.each do |sock|
  begin
    process_client(sock.accept_nonblock)
    # workers load balancing here!! ^^
```

how Unicorn works

unicorn 0.97.0

```ruby
      # in lib/unicorn.rb:630
      # in Unicorn::HttpServer#process_client

  # read request, call app, write app response
  def process_client(client)
    client.fcntl(Fcntl::F_SETFD, Fcntl::FD_CLOEXEC)
    response = app.call(env = REQUEST.read(client))
    # [...]
    HttpResponse.write(client, response,
      HttpRequest::PARSER.headers?)
```

how Unicorn works

```ruby
    # in lib/unicorn/http_request.rb:31
    # in Unicorn::HttpRequest#read

# Does the majority of the IO processing.
# It has been written in Ruby using about 8
# different IO processing strategies.
# [...]
# Anyone who thinks they can make it faster is
# more than welcome to take a crack at it.
```

how Rainbows! works

rainbows 0.91.0

how Rainbows! works

rainbows 0.91.0

Sorry! To be continued......

how Rainbows! works

rainbows 0.91.0

Sorry! To be continued......

?