

# TorqueBox

## Enterprise-Grade Ruby Application Server

Bob McWhirter

**Asheville.rb**

Asheville, North Carolina

**2 December 2009**



Creative Commons BY-SA 3.0

**JBoss Community**

# Bob McWhirter

- Started in C++
- Moved to Java with JDK 1.1.8
- Began Ruby and Rails in 2005
- Lived in Asheville for 4 years
- Did some open-source along the way

# Open-Source



**TorqueBox**

# Currently

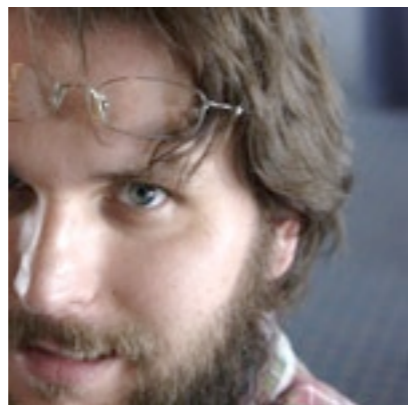


redhat®

← LINUX  
+



← JAVA  
+



← Bob

== Ruby!

# Actual R&D

I get to **actually research things,**  
and **develop new stuff.** Honest-  
to-goodness technology R&D  
**investment.**

# TorqueBox

...in a nutshell

# What is it?

A merging of the **Ruby** language with the **Java Virtual Machine**.

# Why bother?

As Ruby matures, it's quickly being used to solve larger, more difficult problems.

**Problems we've solved in the enterprise with Java already.**

# Such as...?

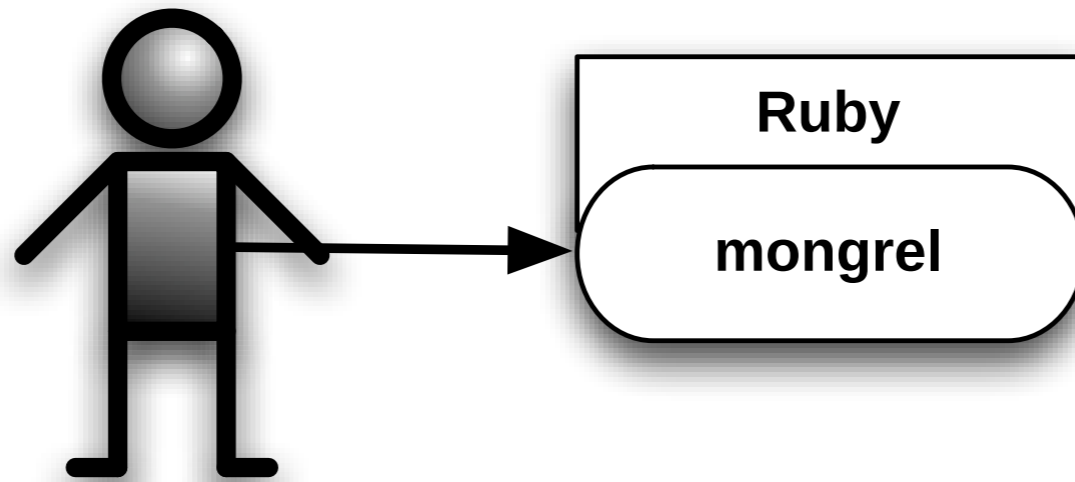
- Scalability
- Efficient & asynchronous messaging
- Scheduled jobs
- Telephony

# Scalability

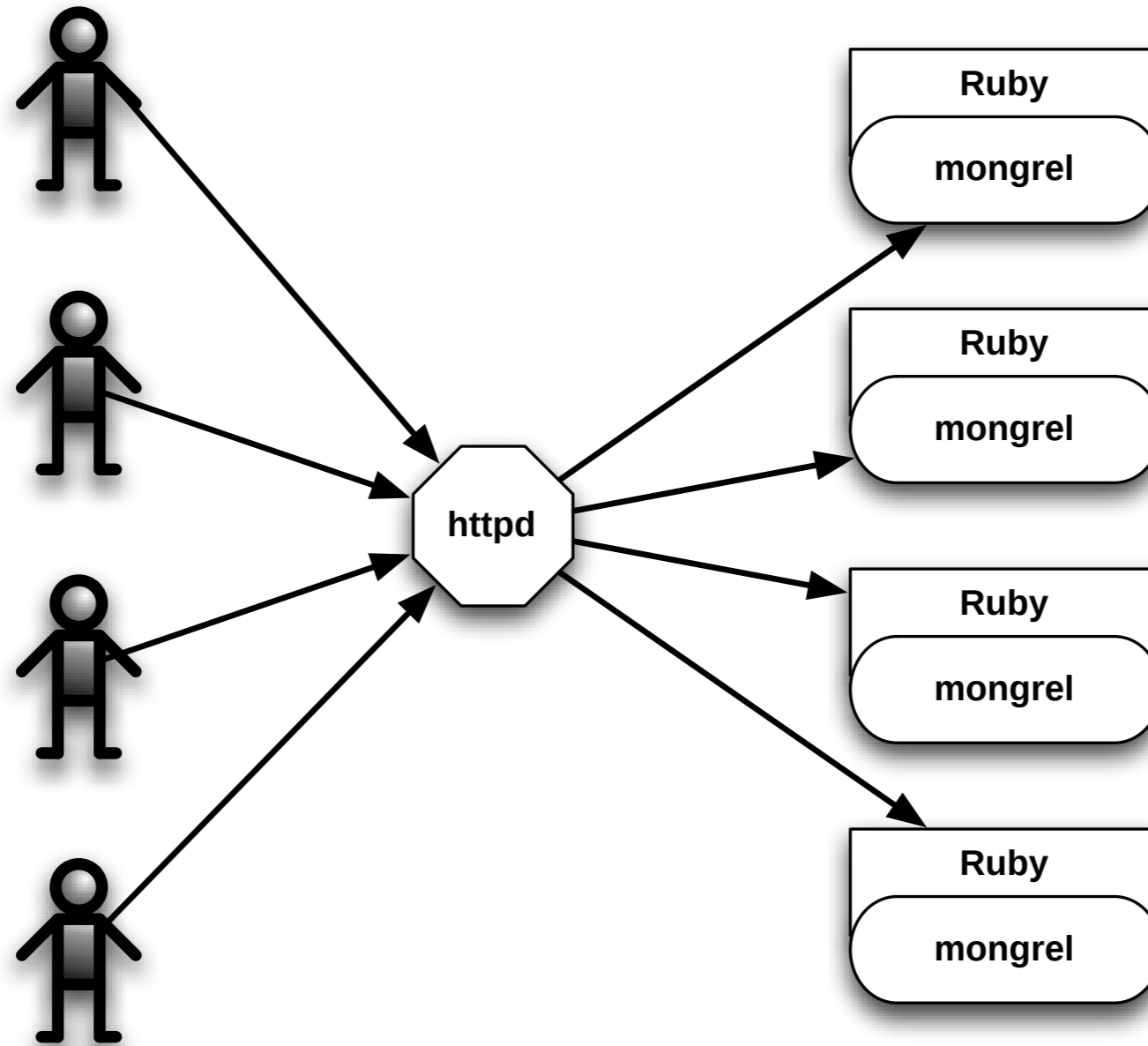
...because “*Rails doesn’t scale*”

# Web Scalability

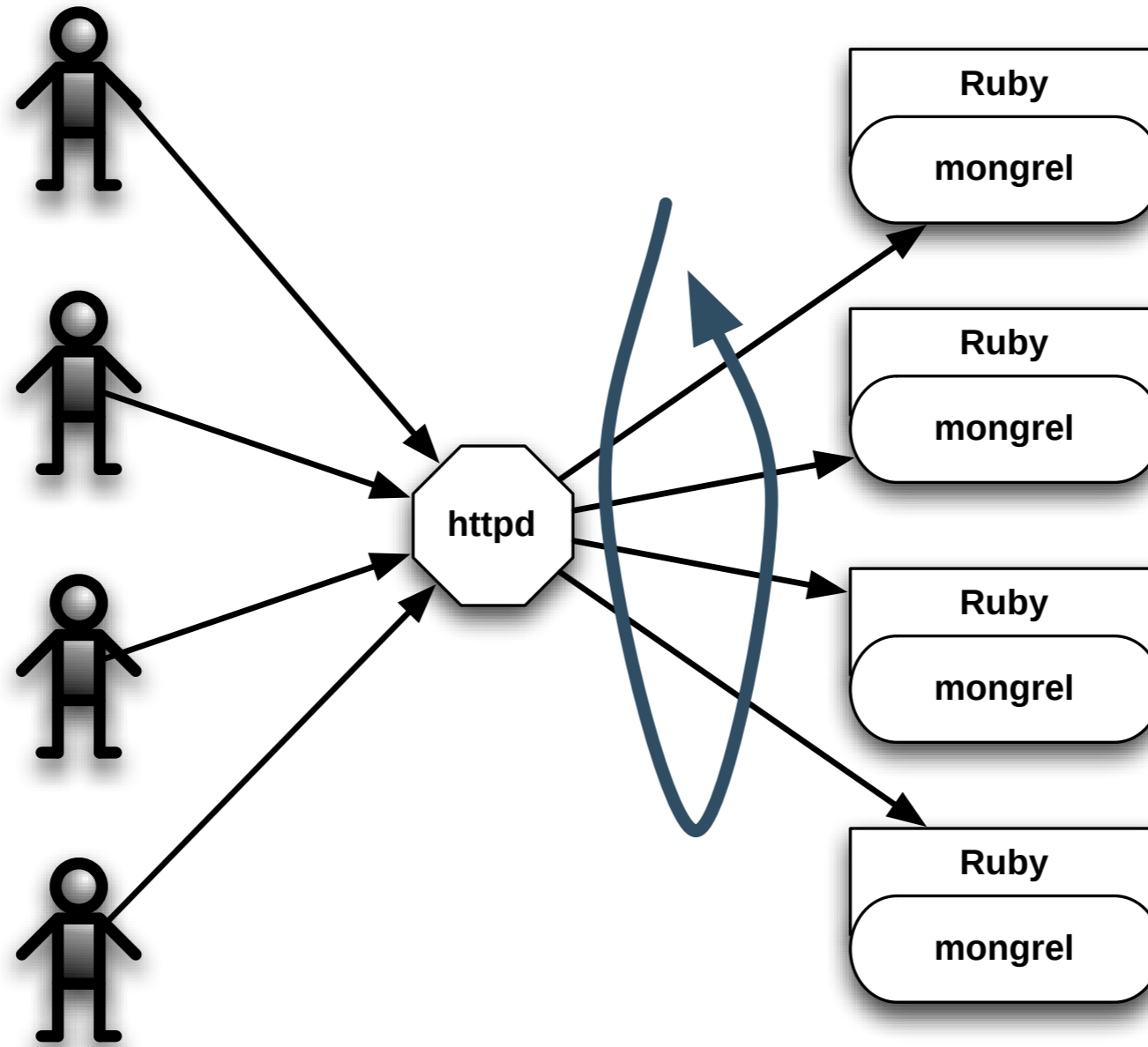
# mongrel-style



# mongrel-style

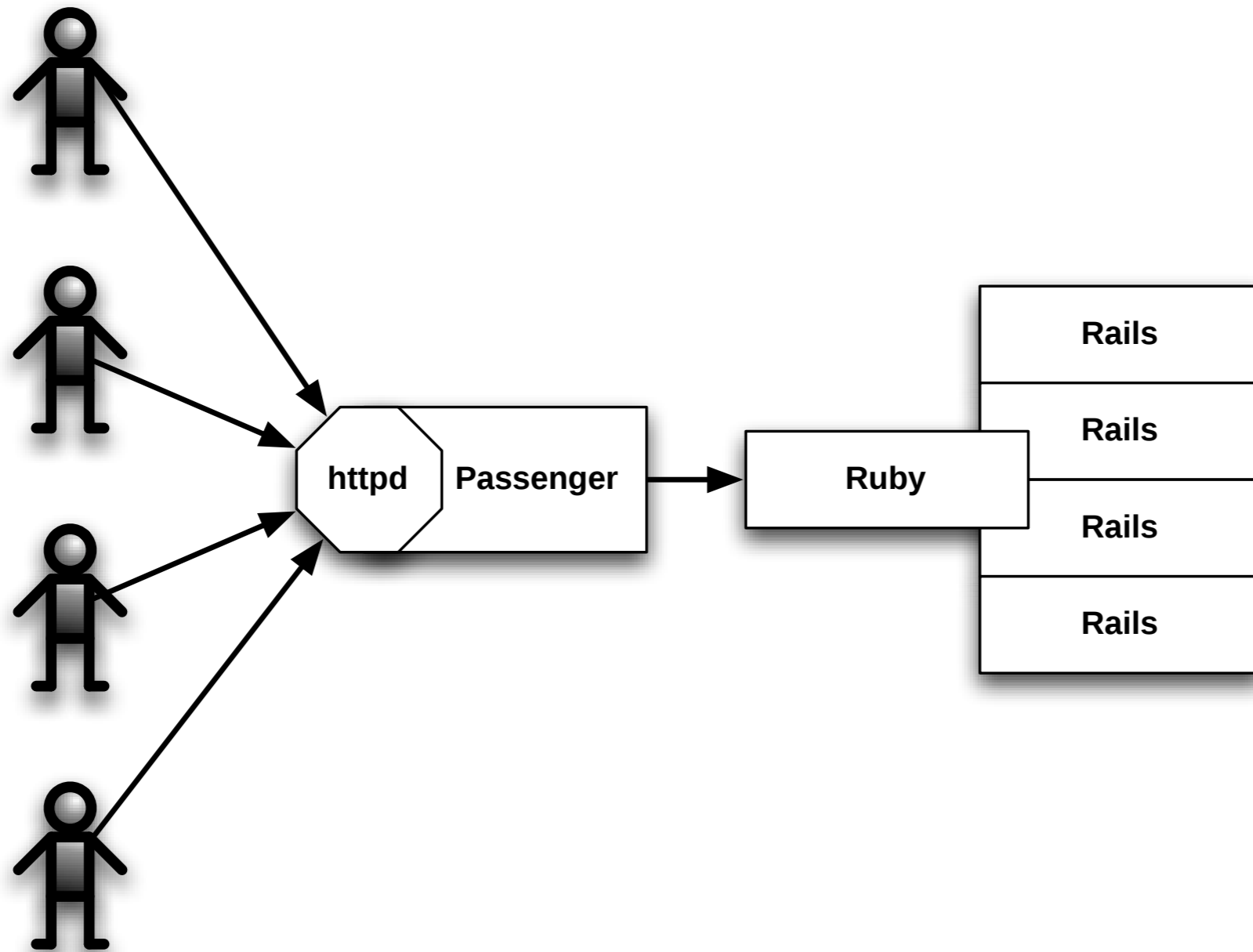


# mongrel-style



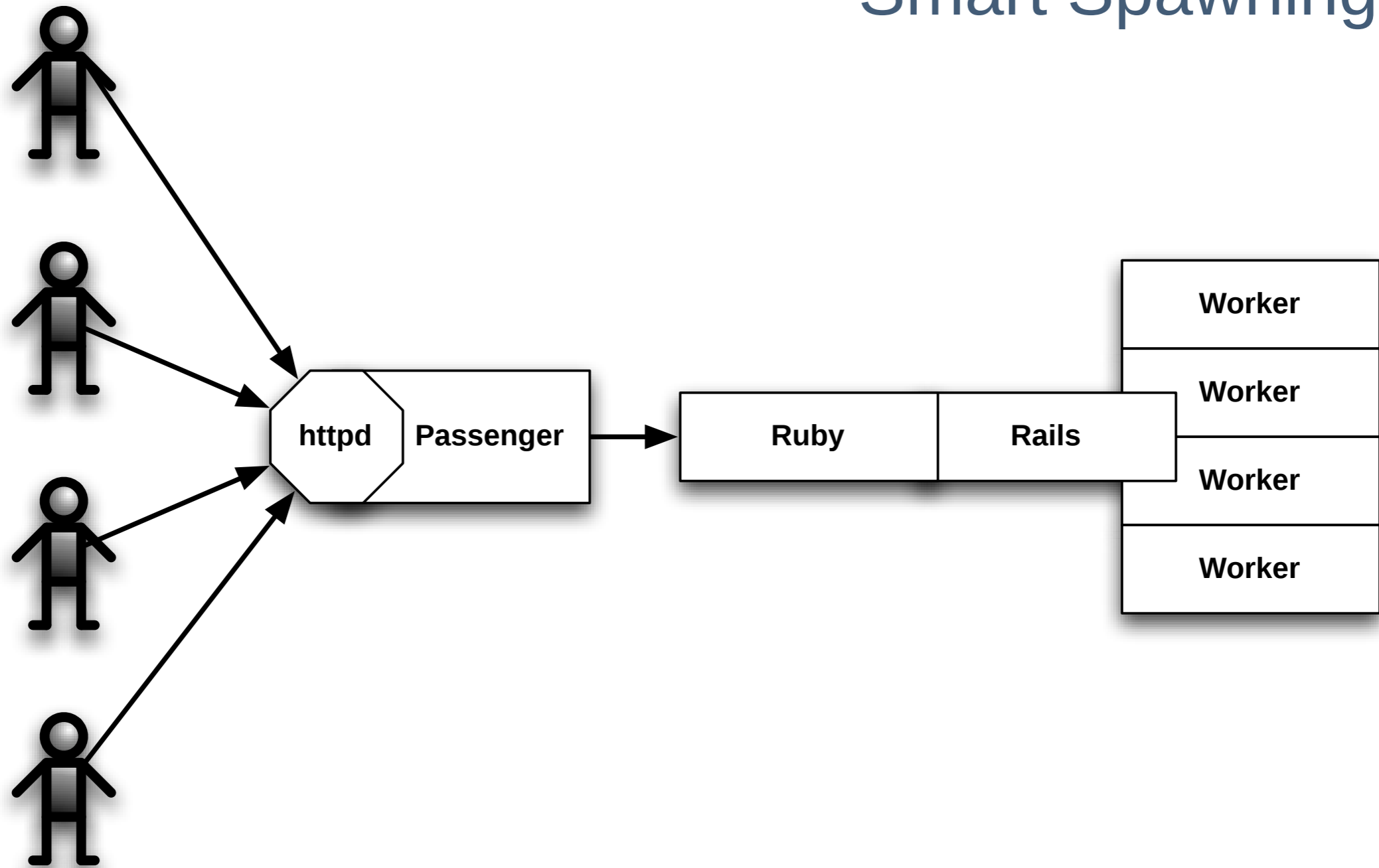
# passenger style

Conservative Spawning

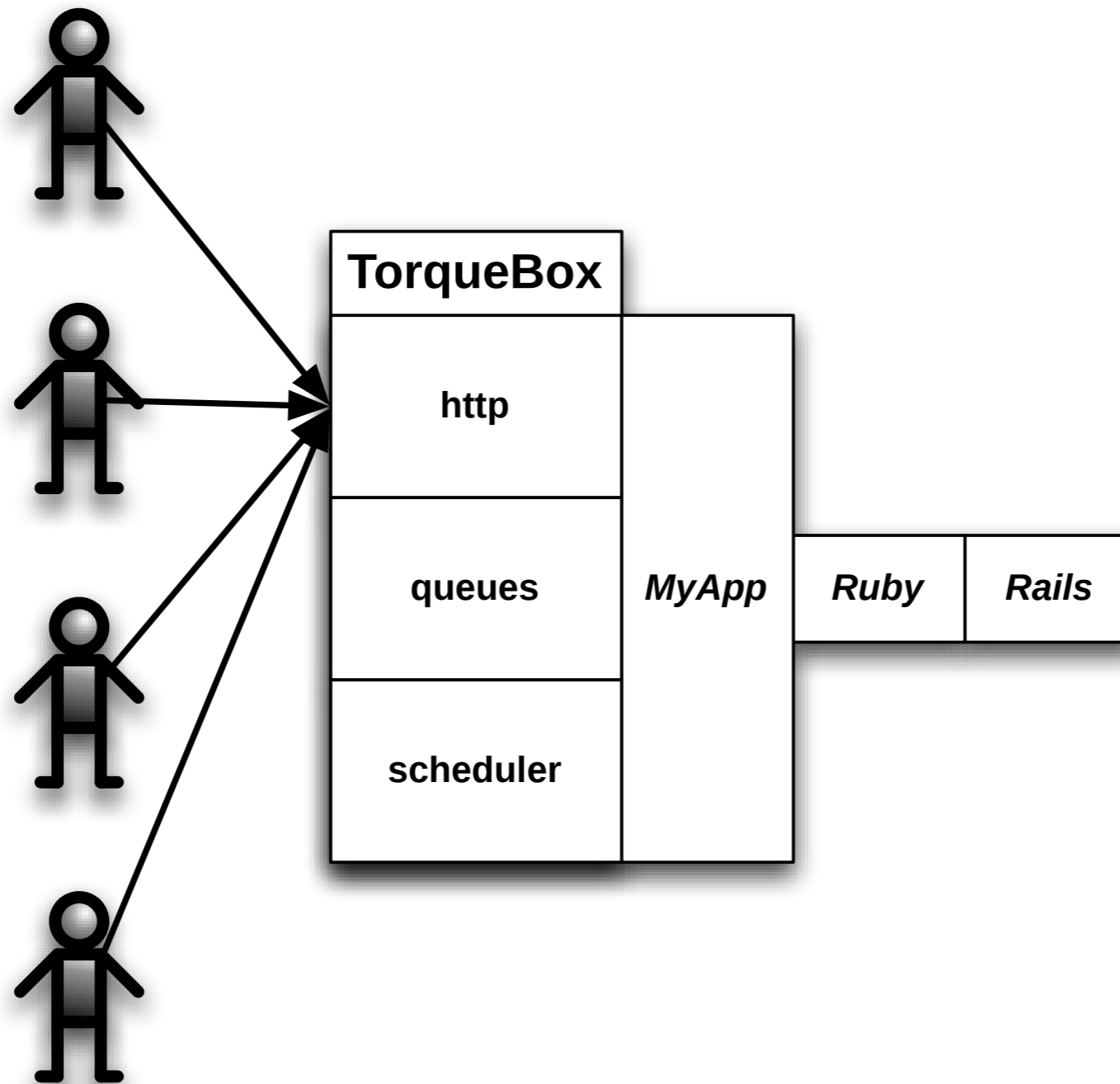


# passenger style

Smart Spawning



# TorqueBox style



# Clustering

Multiplicity rocks.

# Concerns

When clustering, we care about **sessions** and **load balancing**.

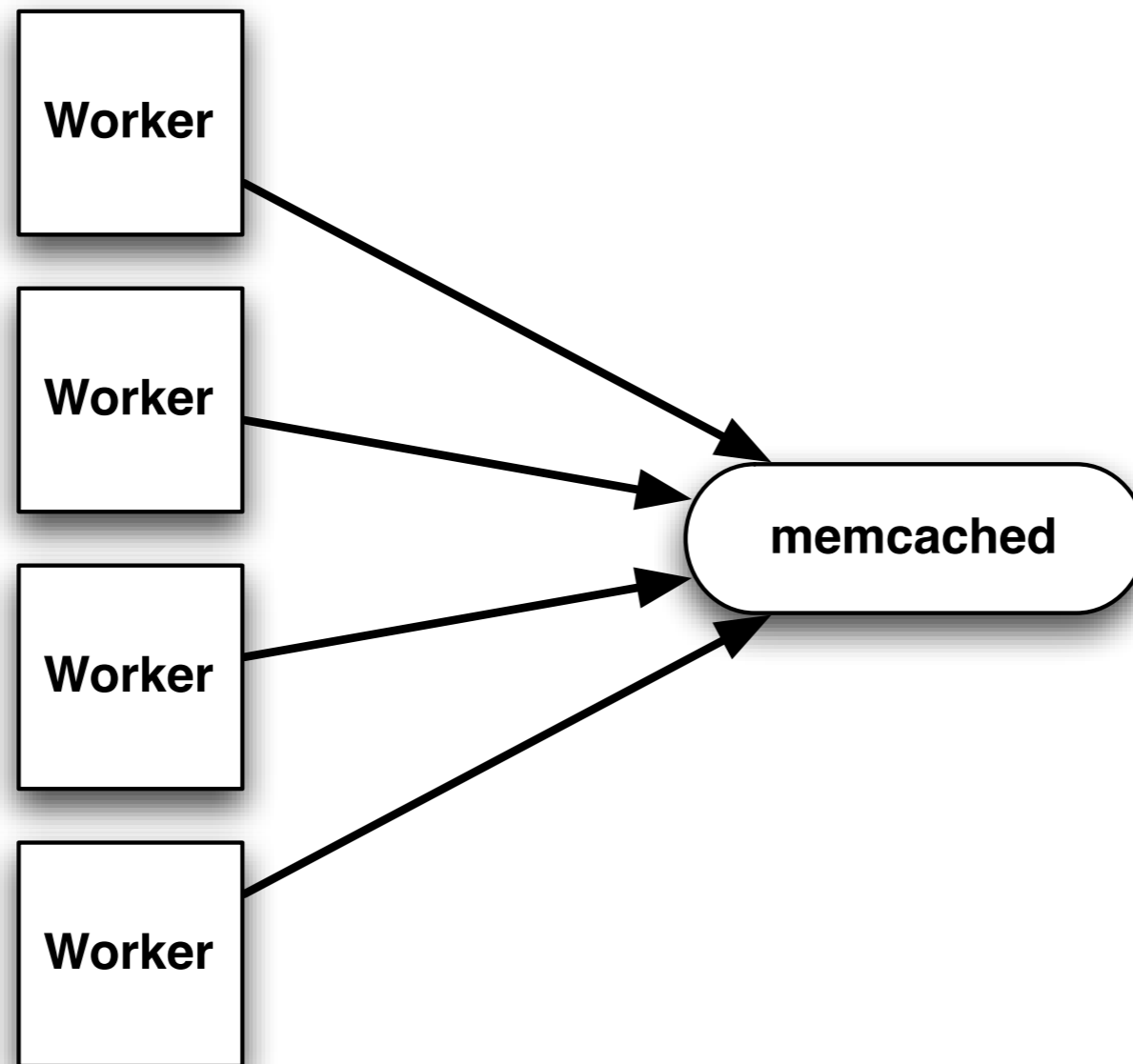
# Sessions

With multiple application servers, sessions must be **shared**, or **session affinity** must be used.

# Session Sharing

Sharing sessions requires using a central store, such as **memcached** or **ActiveRecord** session stores.

# Session Sharing

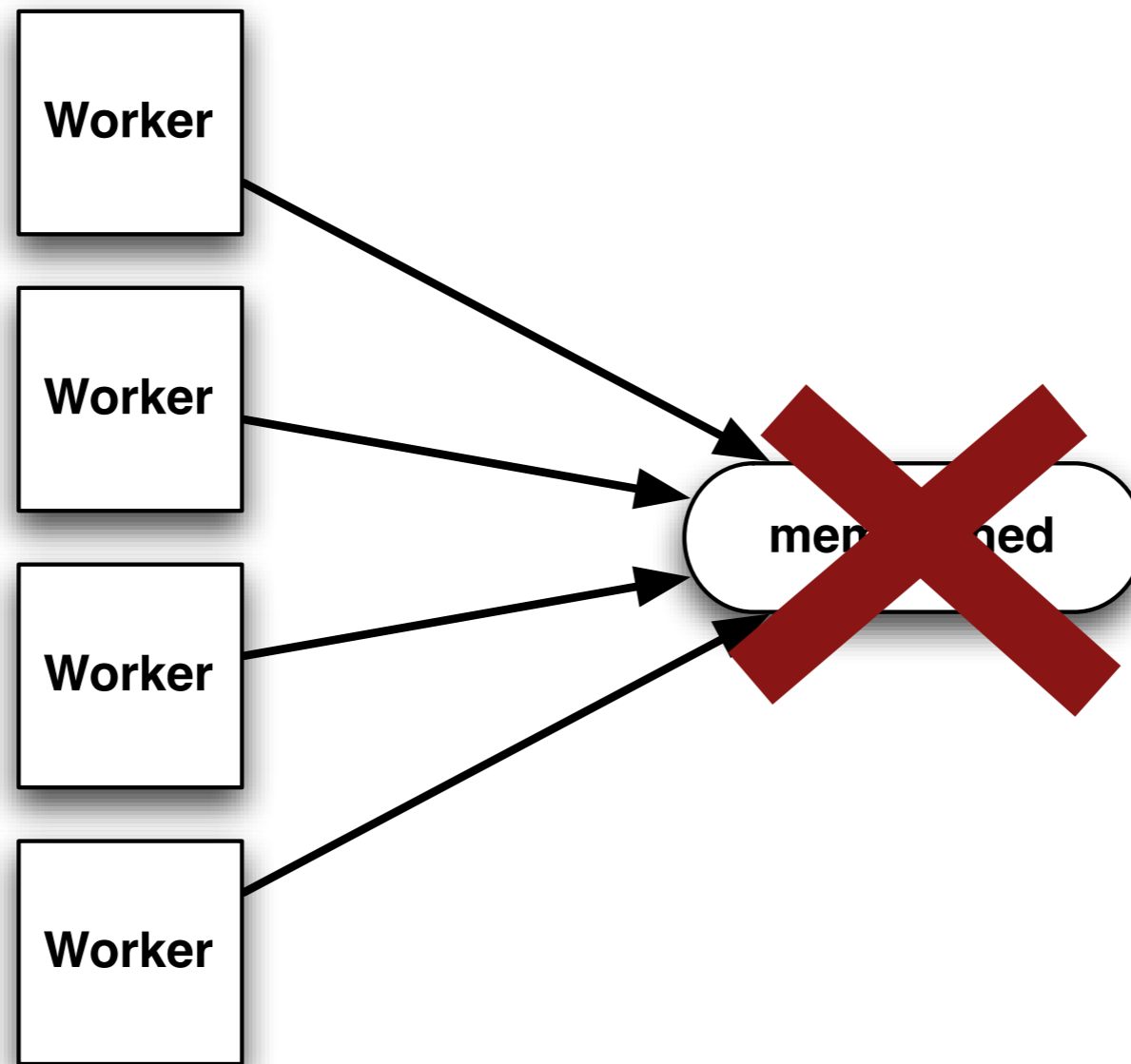


# Session Sharing

But what if the central session store is **unavailable**, or **overwhelmed**?

**Single point of failure!**

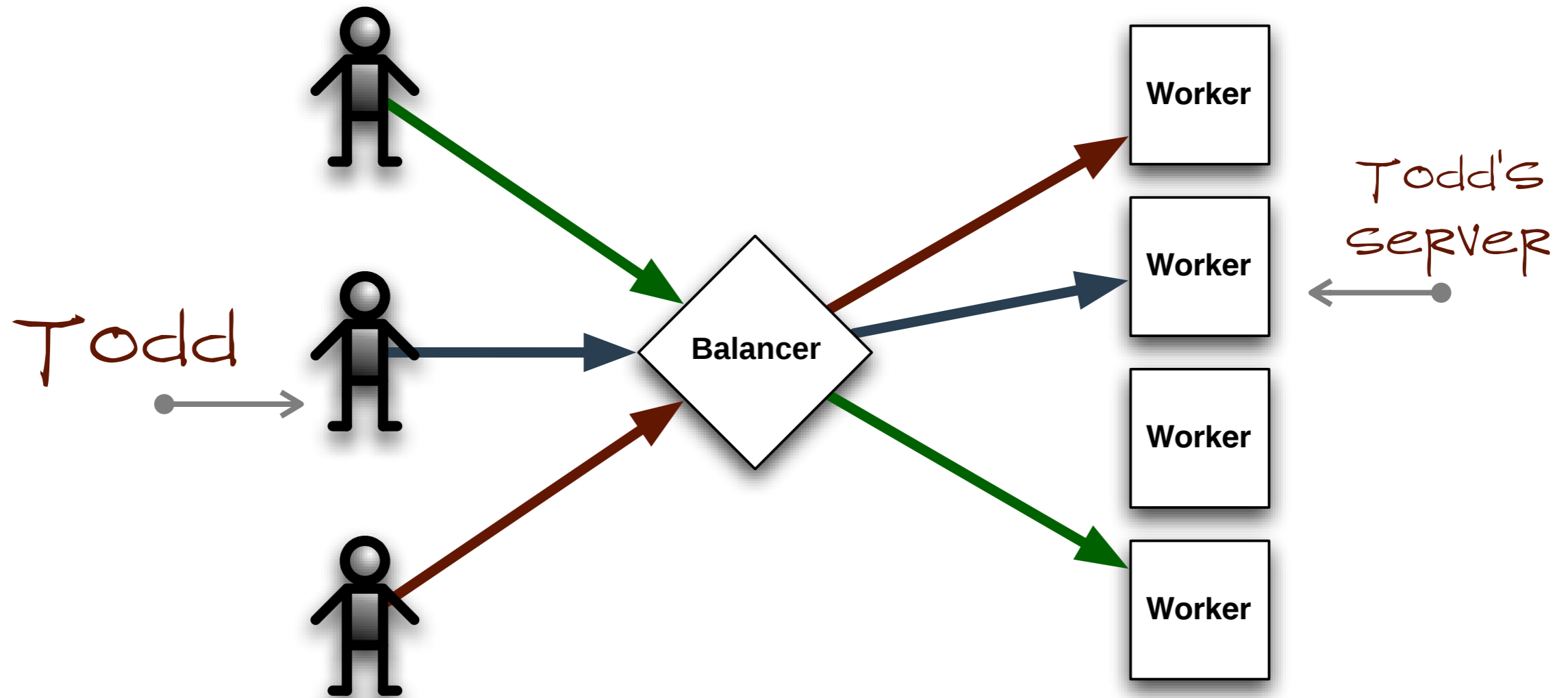
# Session Sharing



# Session Affinity

**Session affinity** is making sure the same back-end server handles all of **Todd's** requests.

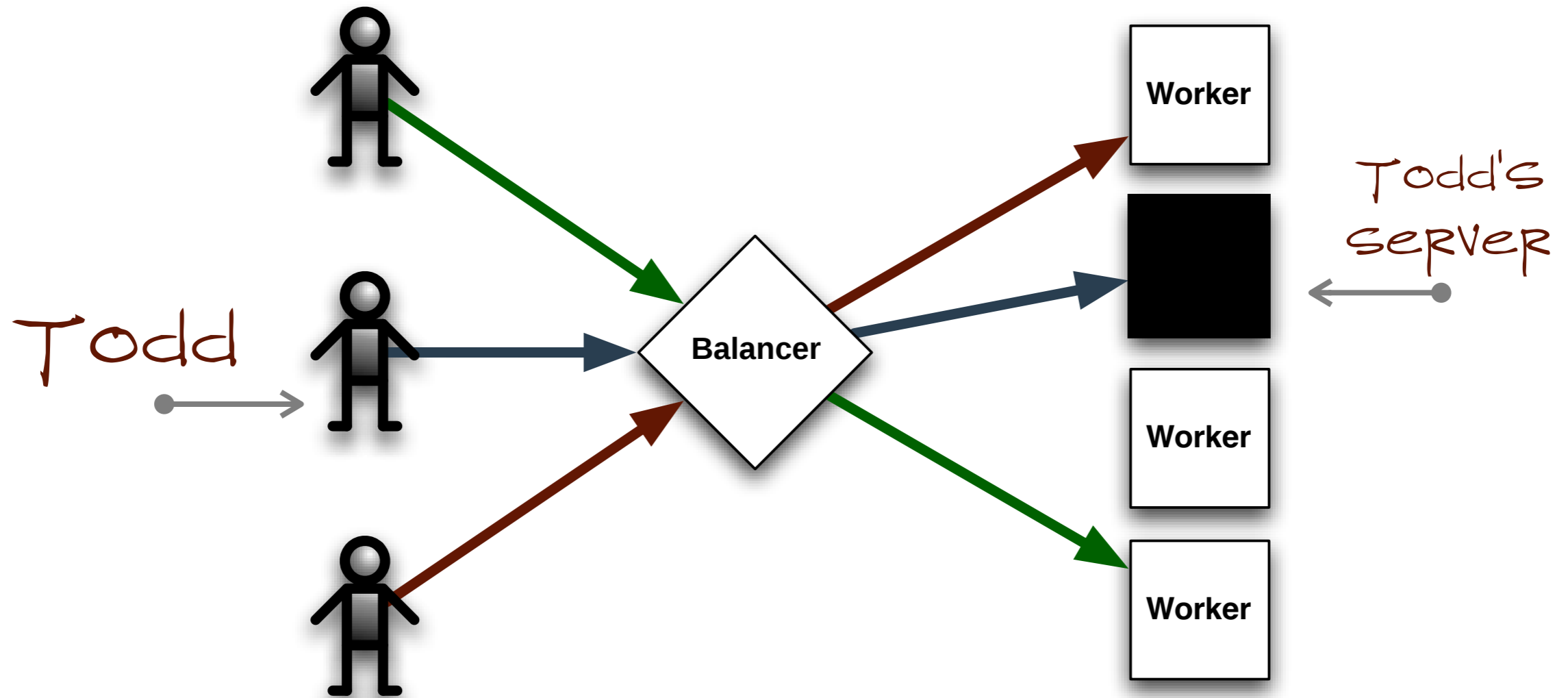
# Session Affinity



# Session Affinity

But what if the back-end server handling Todd's requests becomes **unavailable**?

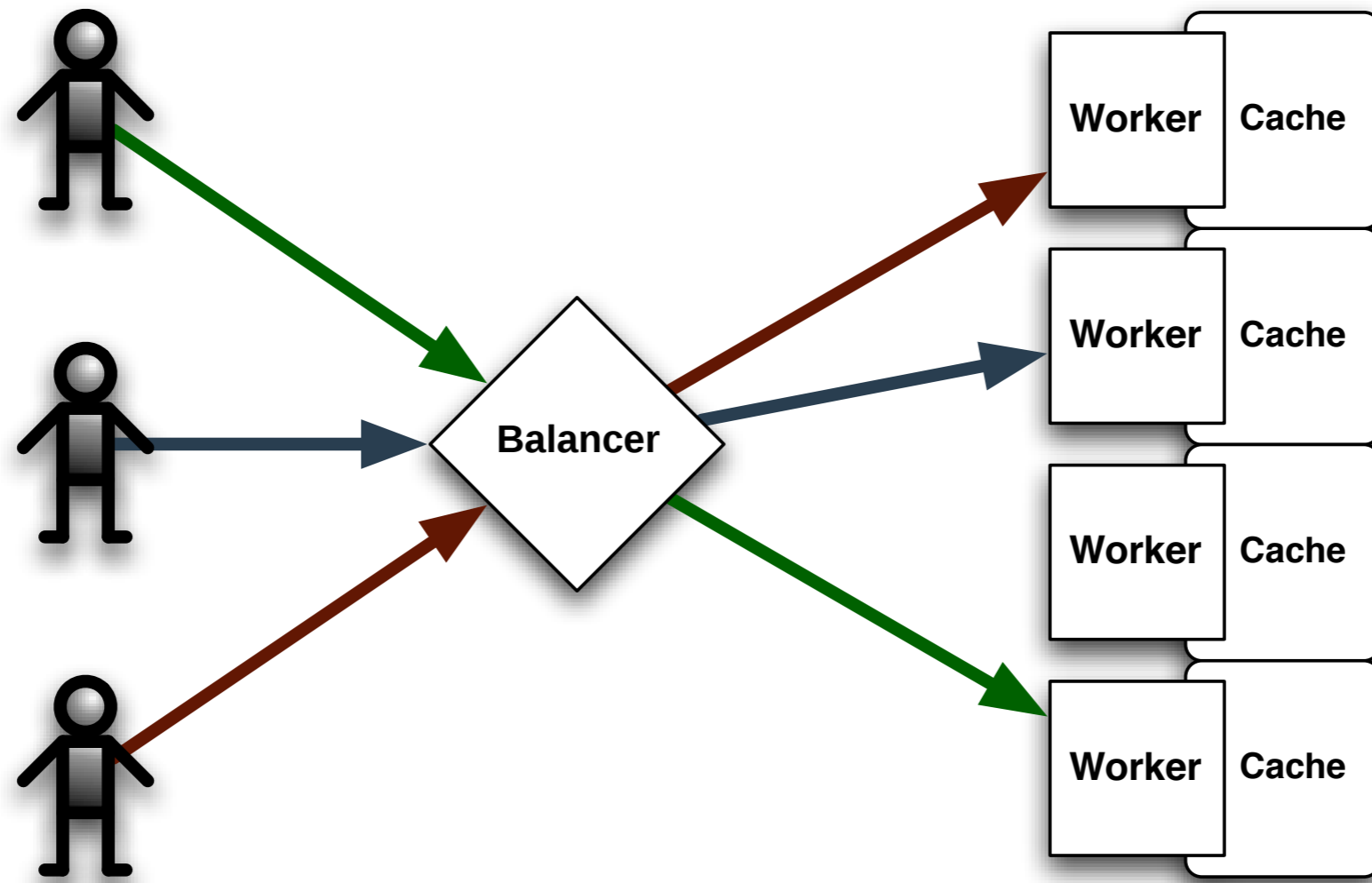
# Session Affinity



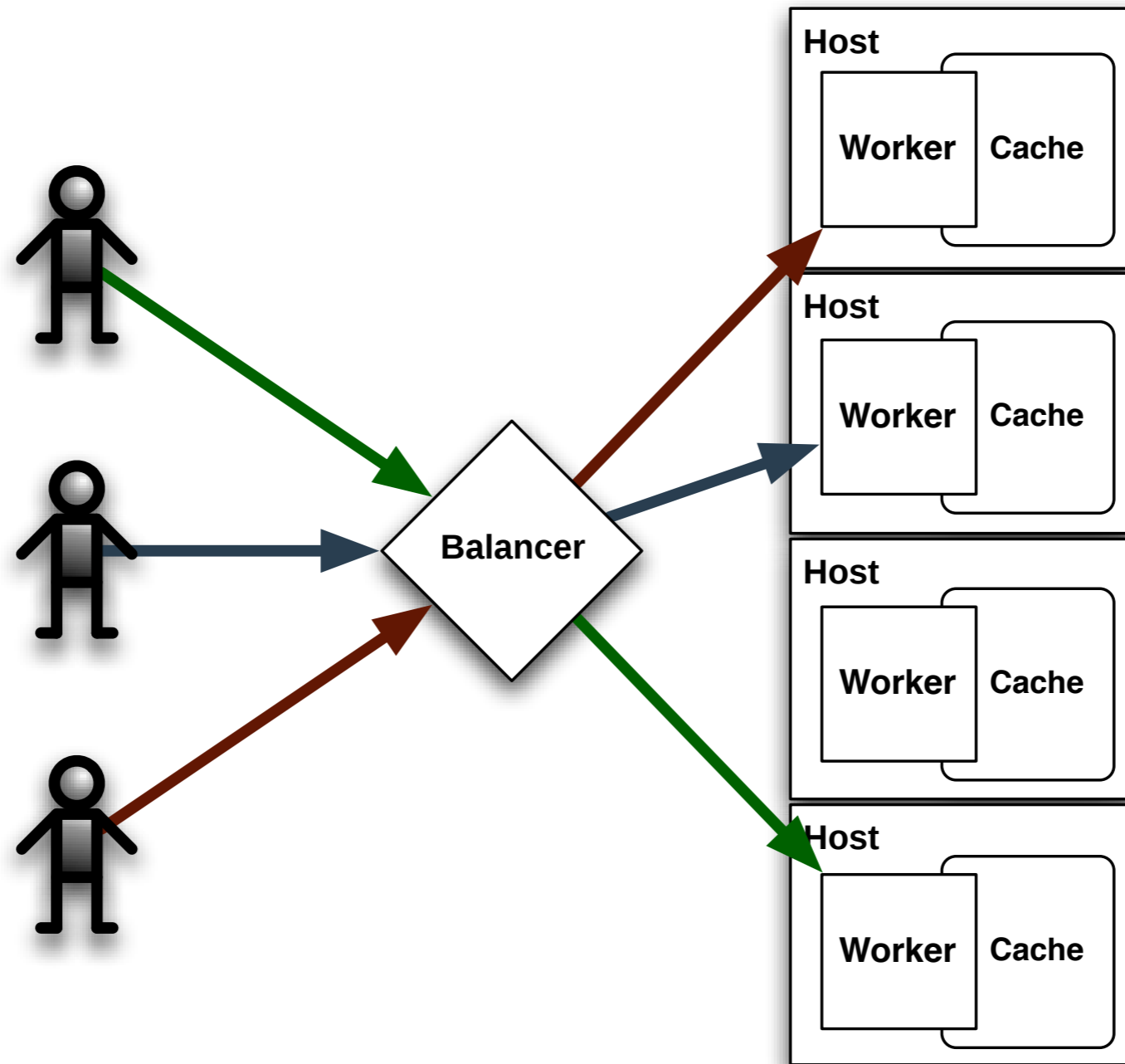
# Use Both!

Use **session affinity** when you can, but allow workers to **replicate** session information when necessary.

# Use Both!



# Spanning Hosts



# Balancer in the cluster

Using **mod\_cluster** for Apache **httpd**, the balancer participates in the cluster view.

# mod\_cluster

As server nodes join and leave the cluster, the balancer (**mod\_cluster**) is aware of these changes.

# mod\_proxy\_ajp

The **Apache JServ Protocol (AJP)** is an efficient pipeline for shuffling HTTP requests to a Java application-server for handling.

# Oh yeah...

- **Multiple apps** per server or cluster.
- **Virtual-host** support baked in.
- **Context-path** support.
- **Full Rack** support (bare, Sinatra, etc).
- **Full Java-EE** stack.

# And bundles

**myapp.rails**

**myapp.rack**

# And bundles

Single **ZIP file bundle** of an app.  
Easily move specific artifact  
between **staging** and **production**.

*Required* for cluster **auto-farming**  
deployment.

# Rails scales!

...but there's more than just port 80.

# Task Queues

Simply install Erlang, set up RabbitMQ, use an AMQP gem...

**Or just use TorqueBox**

...and just write a Ruby class.

# app/queues/\*\*\*\_queue.rb

- A class per queue.
- A method per task handled by the queue.
- A simple API to toss stuff around.

# my\_first\_queue.rb

```
class MyFirstQueue
  include TorqueBox::Queues::Base

  def task_one(payload={})
    o = Thing.find_by_id( payload[:thing_id] )
    o.frob!
    o.save!
  end
end

end
```

# my\_first\_queue.rb

```
class MyFirstQueue
  include TorqueBox::Queues::Base

  def task_one(payload={})
    o = Thing.find_by_id( payload[:thing_id] )
    o.frob!
    o.save!
  end
end
```

# my\_first\_queue.rb

```
class MyFirstQueue
  include TorqueBox::Queues::Base

  def task_one(payload={})
    o = Thing.find_by_id( payload[:thing_id] )
    o.frob!
    o.save!
  end
end
```

end

# my\_first\_queue.rb

```
class MyFirstQueue
  include TorqueBox::Queues::Base

  def task_one(payload={})
    o = Thing.find_by_id( payload[:thing_id] )
    o.frob!
    o.save!
  end

end

end
```

# my\_first\_queue.rb

```
class MyFirstQueue
  include TorqueBox::Queues::Base

  def task_one(payload={})
    o = Thing.find_by_id( payload[:thing_id] )
    o.frob!
    o.save!
  end
end

end
```

# Enqueue

```
TorqueBox :: Queues . enqueue (  
  'MyFirstQueue',  
  :task_one,  
  { :thing_id=>42 }  
)
```

# Enqueue

```
TorqueBox::Queues.enqueue(  
  'MyFirstQueue',  
  :task_one,  
  { :thing_id=>42 }  
)
```

# Enqueue

```
TorqueBox::Queues.enqueue(  
  'MyFirstQueue',  
  :task_one,  
  { :thing_id=>42 }  
)
```

# Enqueue

```
TorqueBox::Queues.enqueue(  
  'MyFirstQueue',  
  :task_one,  
  { :thing_id=>42 }  
)
```

# Enqueue

Use from your **controllers**,  
ActiveRecord **observers**, or  
other **queues**.

# Enqueue

```
class User < ActiveRecord::Base
  has_attached_file :avatar, ...
  after_save do |user|
    TorqueBox::Queues.enqueue(
      :image_processor_queue,
      :process_user_avatar,
      user.id
    ) if user.avatar_needs_processing?
  end
end
```

# Enqueue

```
class User < ActiveRecord::Base
  has_attached_file :avatar, ...
  after_save do |user|
    TorqueBox::Queues.enqueue(
      :image_processor_queue,
      :process_user_avatar,
      user.id
    ) if user.avatar_needs_processing?
  end
end
```

# Enqueue

```
class User < ActiveRecord::Base
  has_attached_file :avatar, ...
  after_save do |user|
    TorqueBox::Queues.enqueue(
      :image_processor_queue,
      :process_user_avatar,
      user.id
    ) if user.avatar_needs_processing?
  end
end
```

# Enqueue

```
class User < ActiveRecord::Base
  has_attached_file :avatar, ...
  after_save do |user|
    TorqueBox::Queues.enqueue(
      :image_processor_queue,
      :process_user_avatar,
      user.id
    ) if user.avatar_needs_processing?
  end
end
```

# Queue Benefits

- No additional systems or languages to setup and maintain.
- Clustering/HA available (including durable messages).
- No explicit queue configuration required.

# Scheduled Jobs

Just write a script, and a crontab, and deploy it with your app. Don't forget to undeploy and update it also. And check cron.log for errors.

**Or just use TorqueBox**

...and just write a Ruby class.

# app/jobs/\*\*.rb

- A class per queue.
- A method called **run()**
- Configure it via **jobs.yml**

# my\_first\_job.rb

```
class MyFirstJob
  include TorqueBox::Jobs::Base

  def run()
    subscriptions = Subscription.find(:all)
    subscriptions.each do |e|
      send_newsletter( e )
    end
  end
end
```

# my\_first\_job.rb

```
class MyFirstJob
  include TorqueBox::Jobs::Base

  def run()
    subscriptions = Subscription.find(:all)
    subscriptions.each do |e|
      send_newsletter( e )
    end
  end
end
```

# my\_first\_job.rb

```
class MyFirstJob
  include TorqueBox::Jobs::Base

  def run()
    subscriptions = Subscription.find(:all)
    subscriptions.each do |e|
      send_newsletter( e )
    end
  end
end
end
```

# my\_first\_job.rb

```
class MyFirstJob
  include TorqueBox::Jobs::Base

  def run()
    subscriptions = Subscription.find(:all)
    subscriptions.each do |e|
      send_newsletter( e )
    end
  end
end
```

# config/jobs.yml

```
subscription.mailer:  
  description: send out newsletters  
  job: MyFirstJob  
  cron: 20 7 1 * * ?
```

# config/jobs.yml

## subscription.mailer:

description: send out newsletters

job: MyFirstJob

cron: 20 7 1 \* \* ?

# config/jobs.yml

subscription.mailer:

**description: send out newsletters**

job: MyFirstJob

cron: 20 7 1 \* \* ?

# config/jobs.yml

```
subscription.mailer:  
  description: send out newsletters  
  job: MyFirstJob  
  cron: 20 7 1 * * ?
```

# config/jobs.yml

```
subscription.mailer:  
  description: send out newsletters  
  job: MyFirstJob  
  cron: 20 7 1 * * ?
```

# config/jobs.yml

20 7 1 \* \* ?  
sec min hour d.o.m. MON d.o.y

# config/jobs.yml

20 7 1 \* \* ?

EVERY day,

at 1:07:20 AM

# Scheduler Benefits

- Uses existing Ruby interpreter when jobs fire. No waiting for Ruby/Rails to load.
- No external system (**crontab**) to maintain.
- No additional daemons to monitor (**daemonize**)
- Full Rails environment (AR models, queues, lib/)

# Telephony

Just... um... something... with Asterisk?

**Or just use TorqueBox**

...and just write a Ruby class.

# Initiate Calls

By doing **SIP** magic, your application can initiate calls, or connect multiple parties.

# Press 2 to STFU

Using the **Media Server**, automated messages, interactive voice response (**IVR**) and other advanced systems can be built.

# OMGWTFBBQPONIES!

Have your app (for instance, upon successful job completion) fire off an **SMS**.

# Telephony Benefits

- It's just freakin' cool.

# Manageability

Okay, sorta speculative, forward-looking dreaming, but...

# Manageable

Everything ultimately maps to a **managed object** within the Java application-server.

We have the **JOPR** management console.

# But not yet

But we haven't exposed everything in a super nice fashion. **Yet.**

# All-in-One

Everything's in the tin.

Everything works together.

# More information...

<http://torquebox.org/>

<http://github.com/torquebox>

<http://twitter.com/torquebox>

# Thanks

+

# Q&A

