RubyKaigi 2007

AP4R

Kiwamu Kato & Shun'ichi Shinohara

Future Architect, Inc.

- Introduction
- Demo
- Real world examples
- Technical topics

Introduction

Self Introduction

Kiwamu Kato

- Majored in Civil Engineering
- Has Enhanced and maintained messaging middleware in company

Kiwamu Kato

- Started Ruby since last year
- Leader of the AP4R team
- http://d.hatena.ne.jp/kiwamu/

shino

- Programmer, I love coding !
- The language of the year is C or Haskell

shino

- Last year, made a debut to the OSS world
- http://d.hatena.ne.jp/ita-wasa/

What is AP4R ?

Lightweight Messaging

What is AP4R ?

- Asynchronous Processing for Ruby
- Loose coupling by messaging

What can we do w/ AP4R ?

- Quicker response to users
- Load distribution

Press release 2006/09/05

http://itpro.nikkeibp.co.jp/article/NEWS/20060905/247218/

Key points are ..

Lightweight
and
Robust

When is it useful ?

Quicker response to users

If NO speedy response is needed,

execute it later

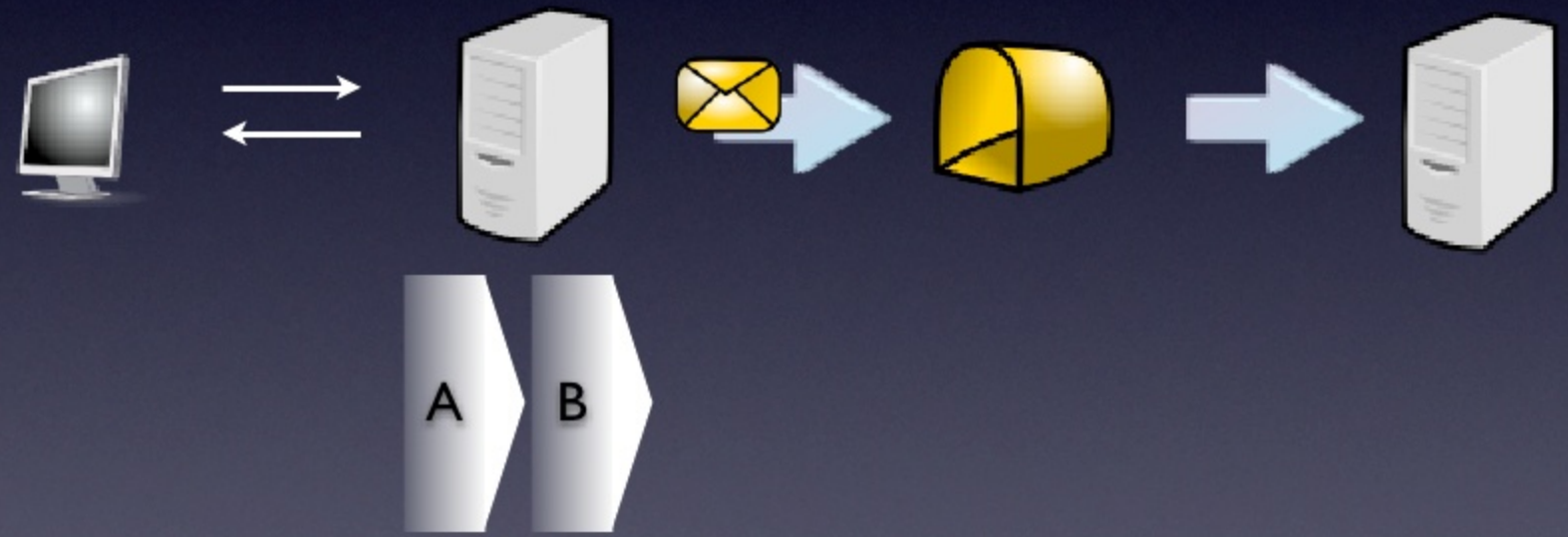
Before

Client

Server

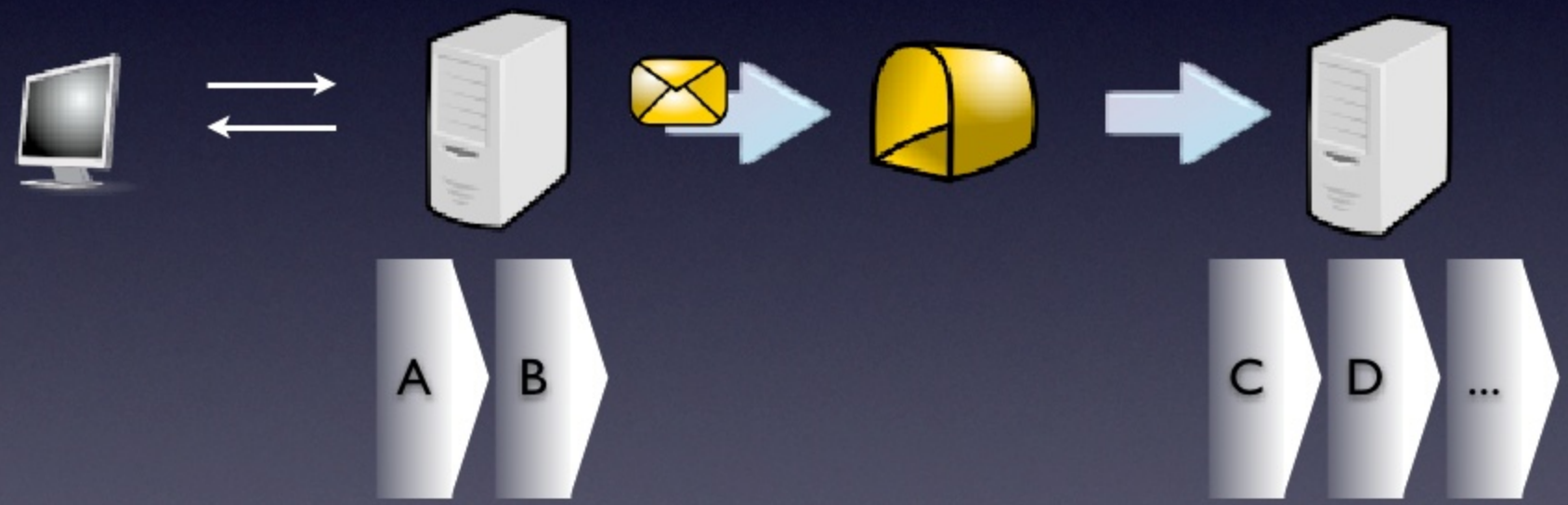
After

Client Server Messaging Server



After

Client Server Messaging Server



Useful cases

- Logging
- Sending mail
- Creating heavy summary data
- Collaboration with other systems

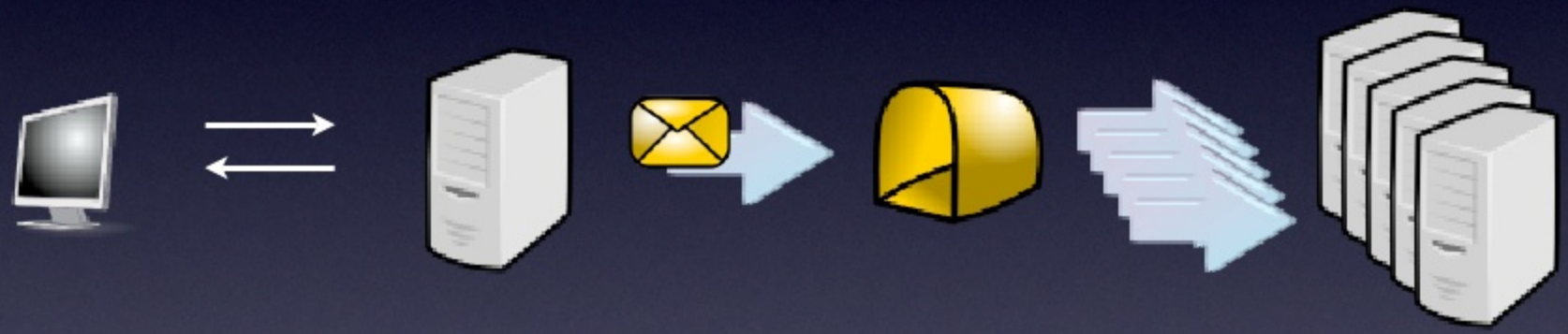
Moreover, load distribution

Easy to scale out

Client Server Messaging Server

Easy to scale out

Client Server Messaging Servers



Only good things ?

Sounds easy ?

(Maybe) No

If message deliveries are NOT guaranteed,...

it may lead to

message lost and message duplication

In other words...

- You can't receive the goods despite payment !
- Double withdrawals !

Messages should be delivered in a reliable way,

No matter what happens

- Database trouble
- Network trouble
- Server down
- Busy and timeout
- ... etc

Lightweight

and

Robust

In Future Architect ...

We use messaging a lot

- Developed own messaging library
- Named "RtFA"

We use messaging a lot

- Pure Java
- Its own protocol, API (≠ JMS)
- Time-proven on various systems

One of illustrative cases

Mission critical system in transportation business

LinuxToday 2006年12月28日 12:20

LinuxToday・バックナンバー

フューチャーシステムなど、佐川急便の基幹システムをオープン化

著者: 山形直子 プリンター用 記事を転送

▼2006年12月28日 12:20 付の記事

□国内internet.com発の記事

ブックマークする

IT コンサルティングのフューチャーシステムコンサルティングは2006年12月27日、 SG ホールディングス（SGH）と、 佐川急便の基幹システムのひとつ「貨物システム」をオープンシステムに刷新、稼動を開始した、と発表した。

これにより、メインフレーム以外では実現できないと言われてきた大量トランザクションを要する大規模システムを、オープン系の分散・並列処理で実現できることが実証された。

システムでは、一日最大1,000万個の貨物取扱個数に関わる1億件以上の貨物データを格納して、佐川急便の業務の中核となる貨物情報を一元管理、一般顧客と1万台以上の社内クライアントからの貨物データを利用できるようにした。

http://japan.internet.com/linuxtoday/20061228/5.html

PJ summary

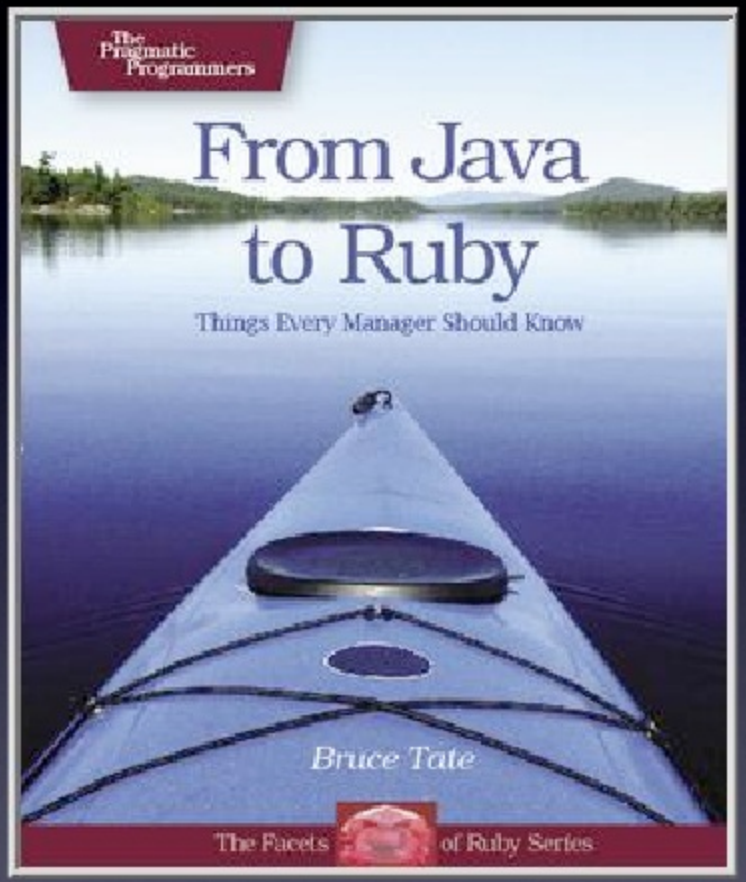
- Over 100 million package data per day
- Load distribution and parallel processing on 8 nodes of Oracle 10g RAC

PJ summary

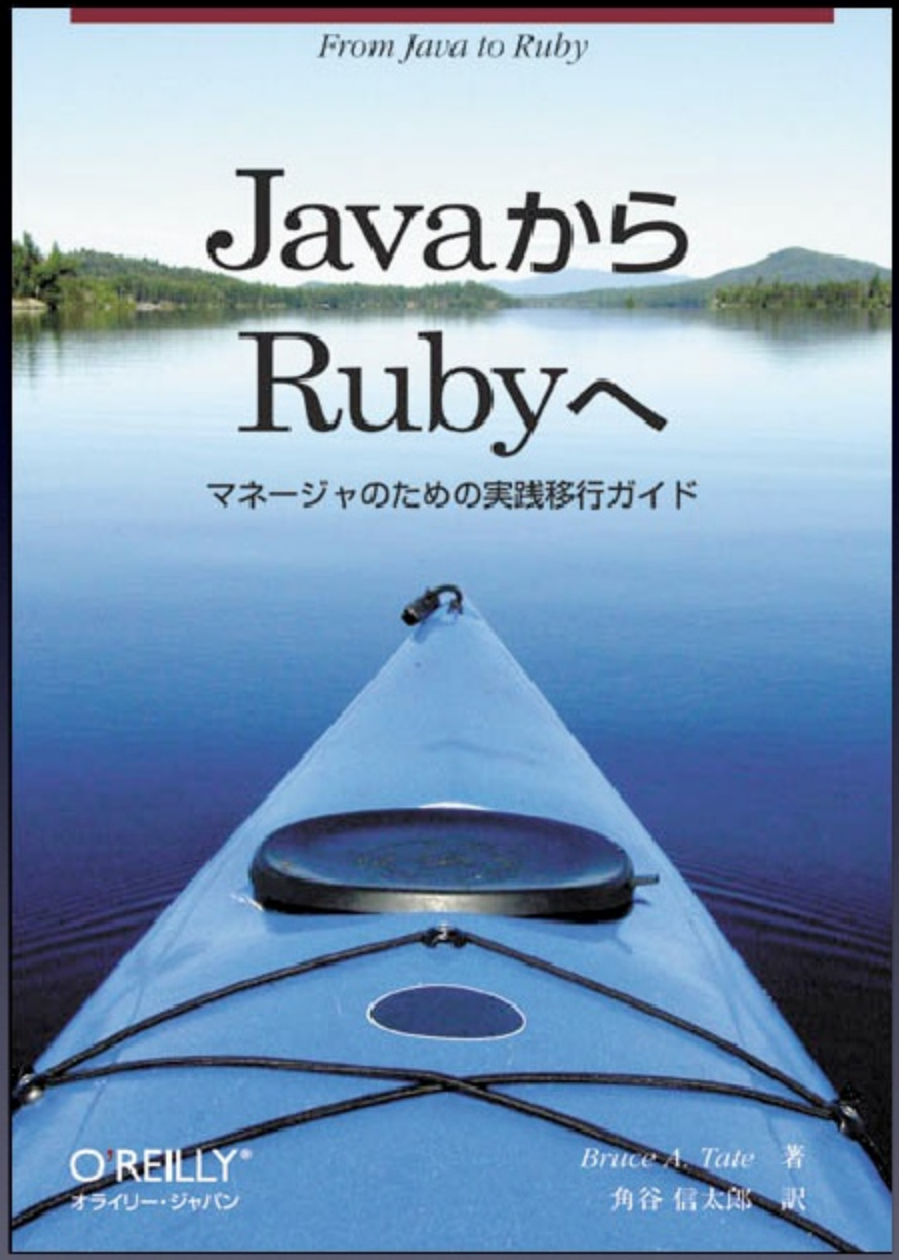
- Approx 100 servers
- Used "RtFA", an in-house middleware

Messaging is important

Lightweight Messaging



http://www.pragmaticprogrammer.com/titles/fr_j2r/



http://www.oreilly.co.jp/books/9784873113203/

From RtFA to AP4R

- Based on implementation/experiences with Java, kept good parts, improved bad parts
- Focus on ease of use（API, configuration）

Lightweight

and

Robust

Demo

Outline

- Shopping store application
- Order and payment

Shopping store app.

Client

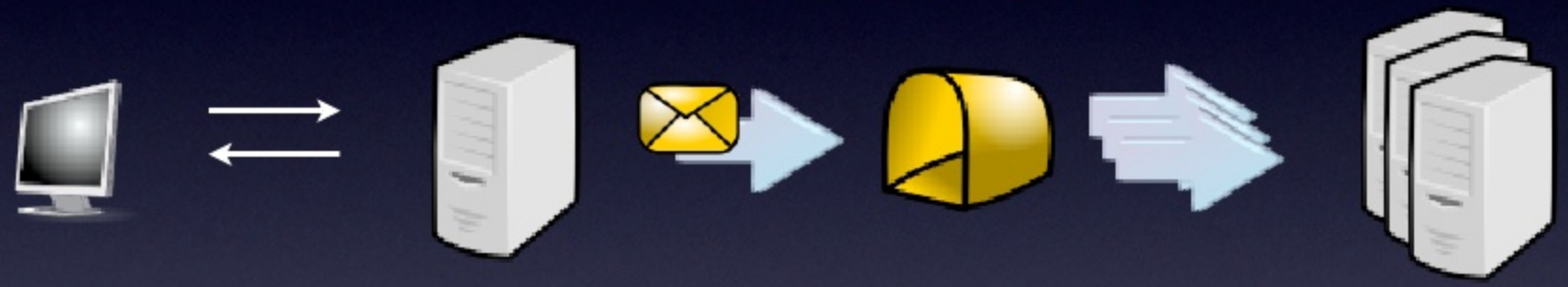
Rails

Order

Payment

Loose coupling

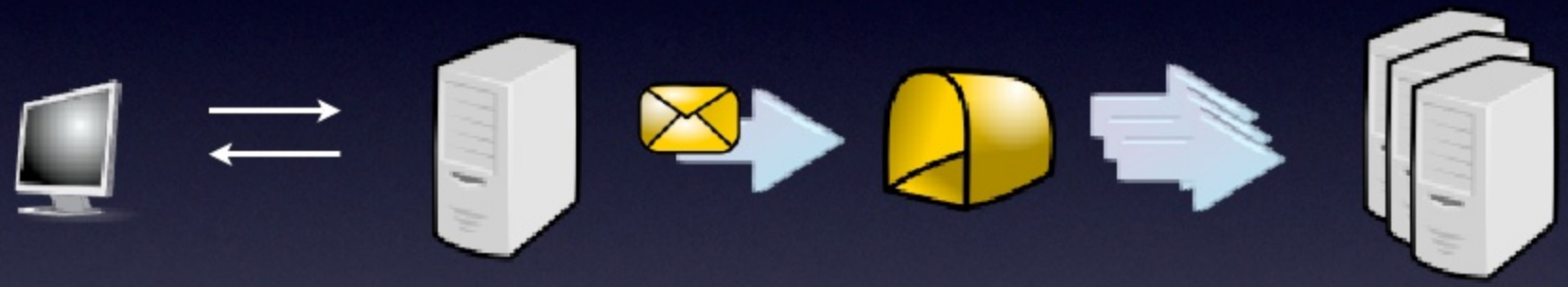
Client Rails AP4R Rails × 3



Order

Loose coupling

Client Rails AP4R Rails × 3

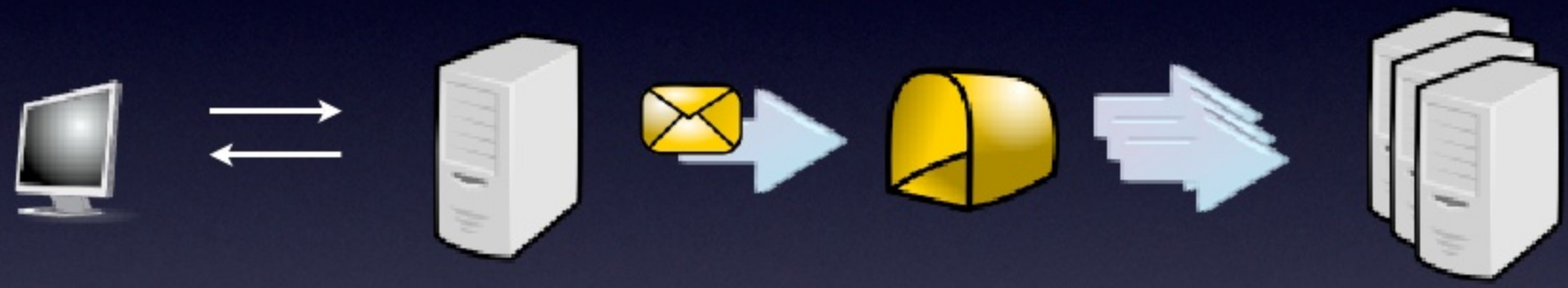


Order

Payment

Loose coupling

Client Rails AP4R Rails × 3



Order

Payment

Sleep 1

Outline

- Use mongrel cluster
- Stress by ab
- Monitor queue retaining status

Real world examples

Two examples

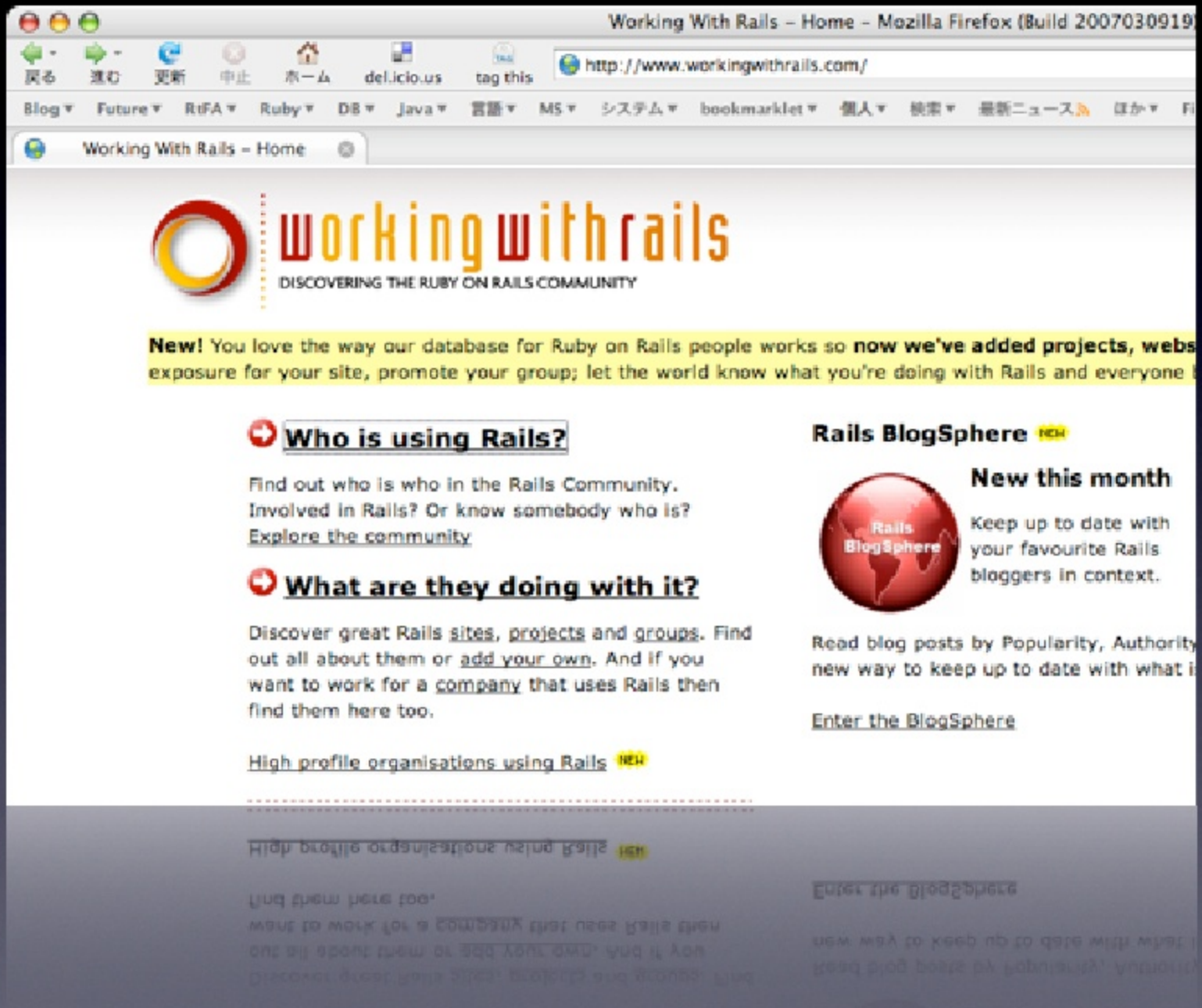
- Work With Rails
- Our internal project

Working With Rails

http://workingwithrails.com/

WWR

- is who's who for Rails developers
- listed 7,000 people
  from 104 countries

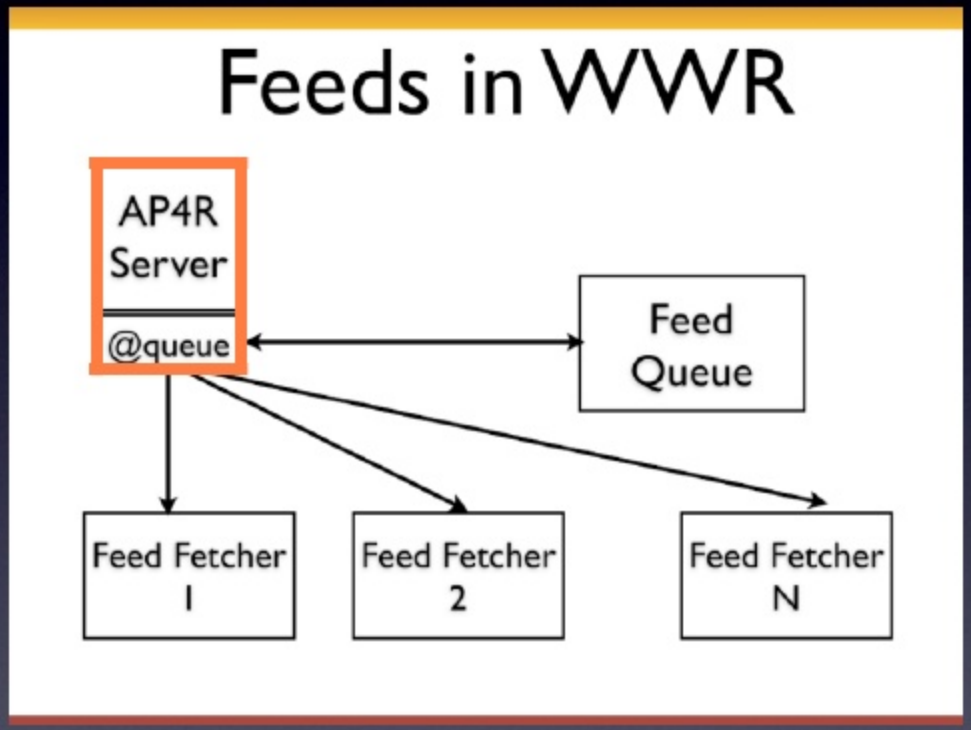

Working With Rails – Home – Mozilla Firefox (Build 2007030919)
http://www.workingwithrails.com/
Working With Rails – Home
working with rails
DISCOVERING THE RUBY ON RAILS COMMUNITY
New! You love the way our database for Ruby on Rails people works so now we've added projects, webs
exposure for your site, promote your group; let the world know what you're doing with Rails and everyone
Who is using Rails?
Find out who is who in the Rails Community.
Involved in Rails? Or know somebody who is?
Explore the community
What are they doing with it?
Discover great Rails sites, projects and groups. Find out all about them or add your own. And if you want to work for a company that uses Rails then find them here too.
High profile organisations using Rails
Rails BlogSphere
New this month
Keep up to date with your favourite Rails bloggers in context.
Read blog posts by Popularity, Authority
new way to keep up to date with what
Enter the BlogSphere

WWR

- aggregates feeds from user's blog
- has scaling problem in fetching them slow/stale

WWR solution

- Queue Feeds
- Fetch async'ly



"Getting Distributed with Ruby on Rails" slide #44 (from slideshare)

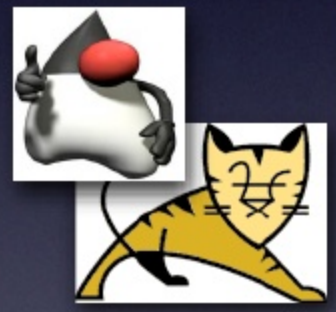
WWR solution

- Able to scale
- Flexible to feed types

Internal project

Briefing

- Work sheet management app.
  newly-building in Ruby
- Labor cost accounting app.
  existing in Java



Requirements

- Quick and rich UI
- Replicate data between Java and Ruby

With AP4R

- Separate loads from UI
- Invoke Java from Ruby async'ly

Separate loads from UI

Client Server Messaging Server

Separate loads from UI

Client Server Messaging Server

Separate loads from UI

Client Server Messaging Server

Separate loads from UI

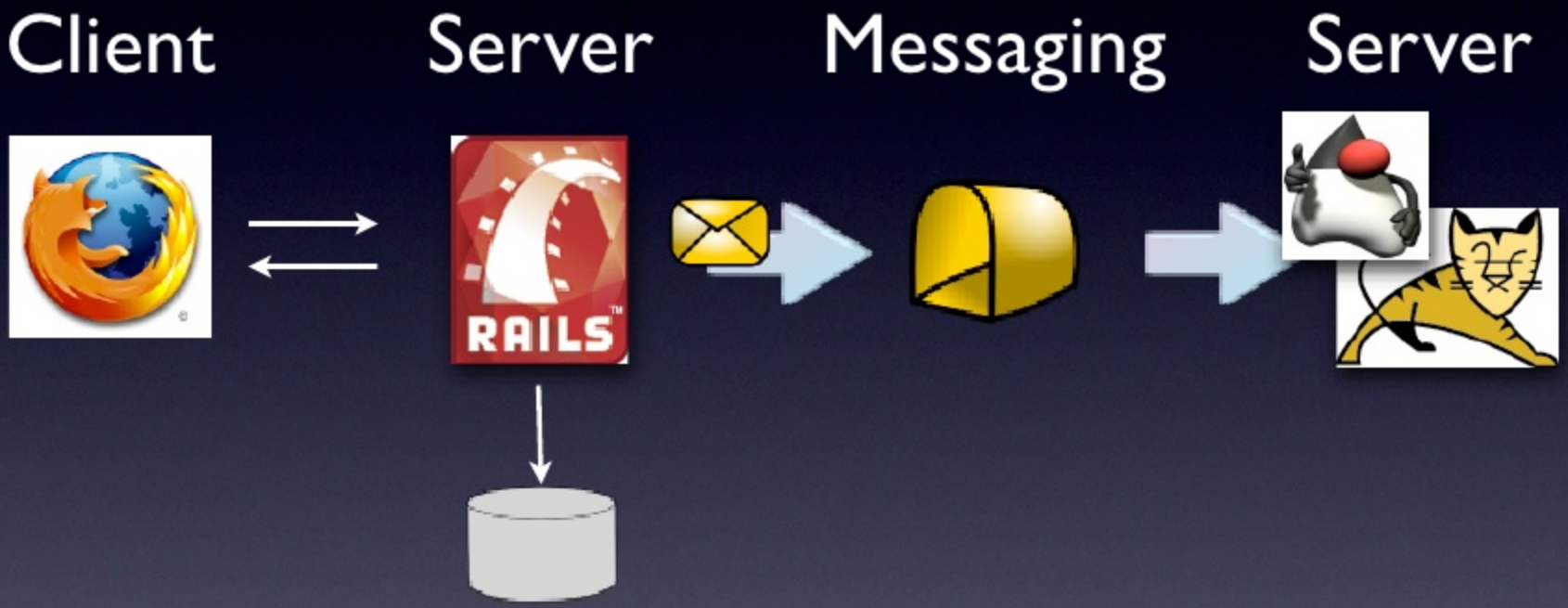
Client Server Messaging Server

Create summary data,
Send mails, etc.

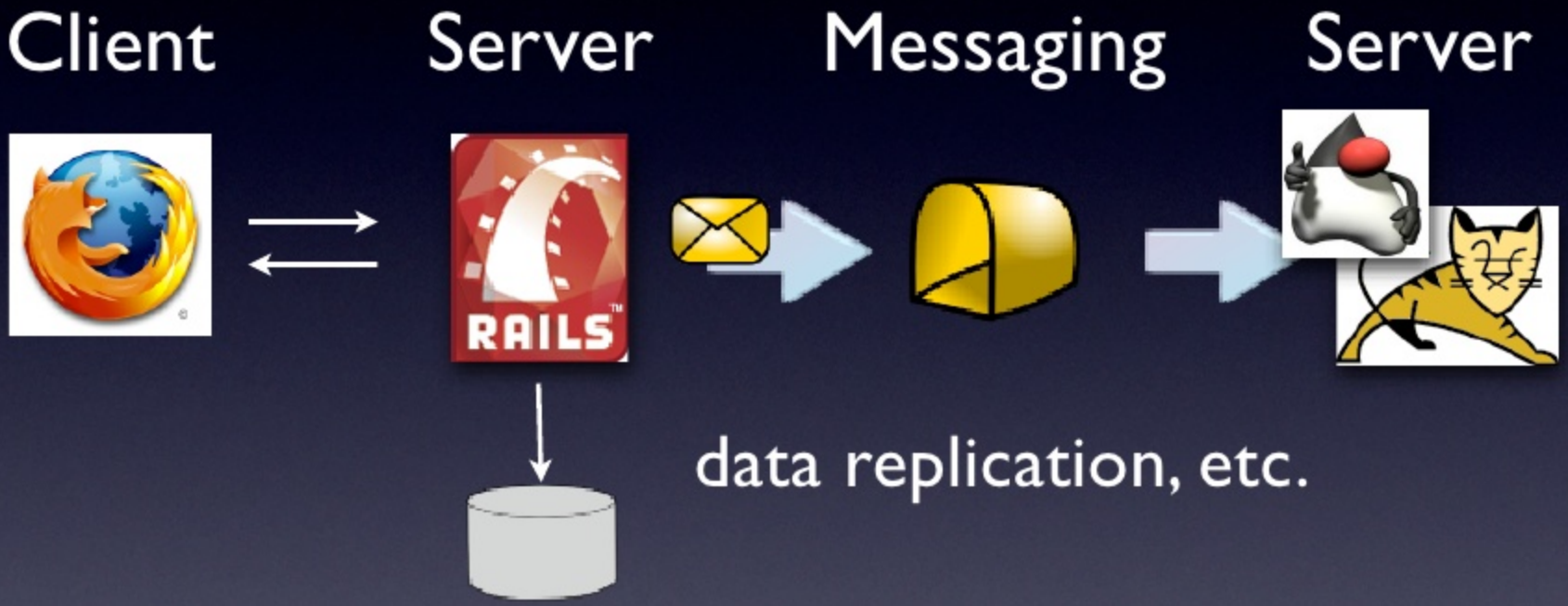
Async'ly invoke Java

Client Server Messaging Server

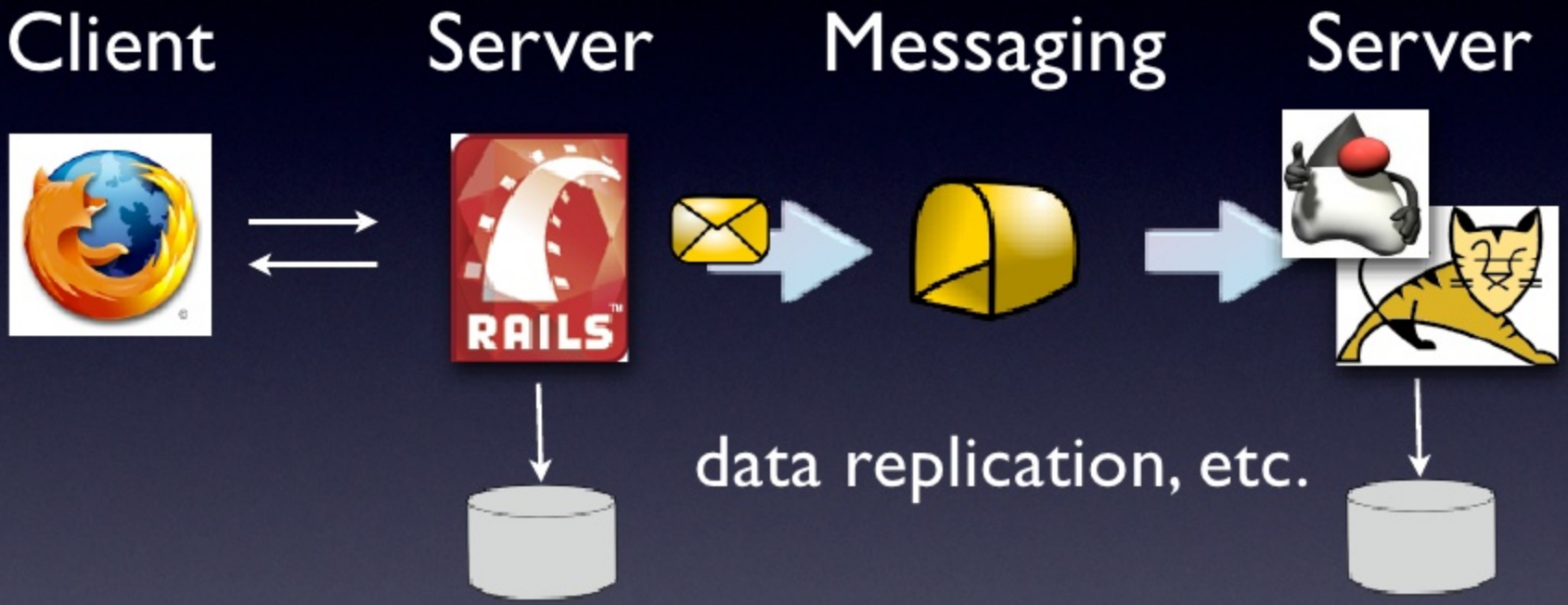
Async'ly invoke Java



Async'ly invoke Java



Async'ly invoke Java



Utilize Async! when

- Needless proc. for responce data
- Disjoined transactions are allowable
- External calls (often slow, unstable)

Technical topics

Two topics here

- API to put messages and process flow
- Stored and Forward

API to put messages and process flow

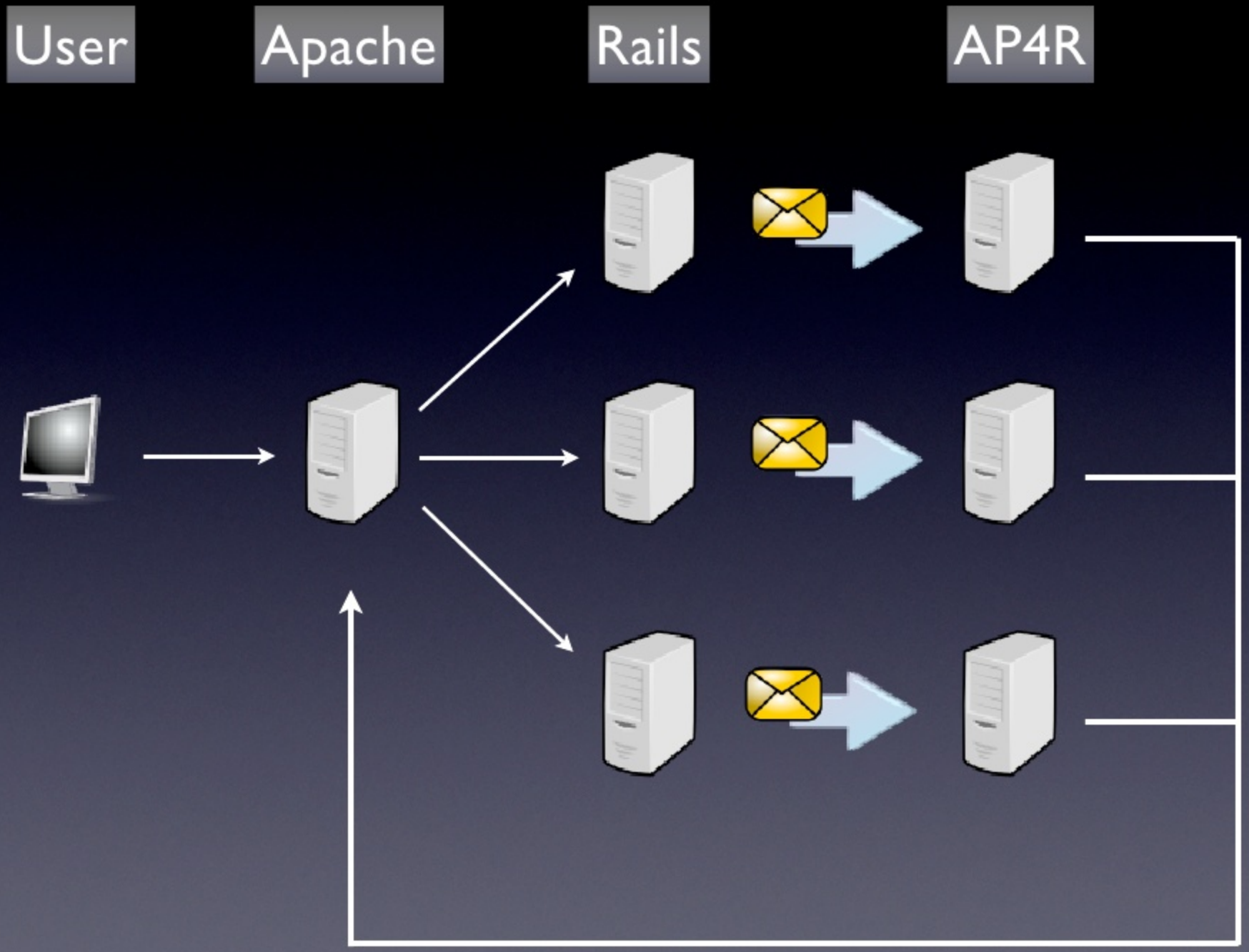
```
ap4r.async_to({specify next},
              {request data}
              [,{more opts}])
```

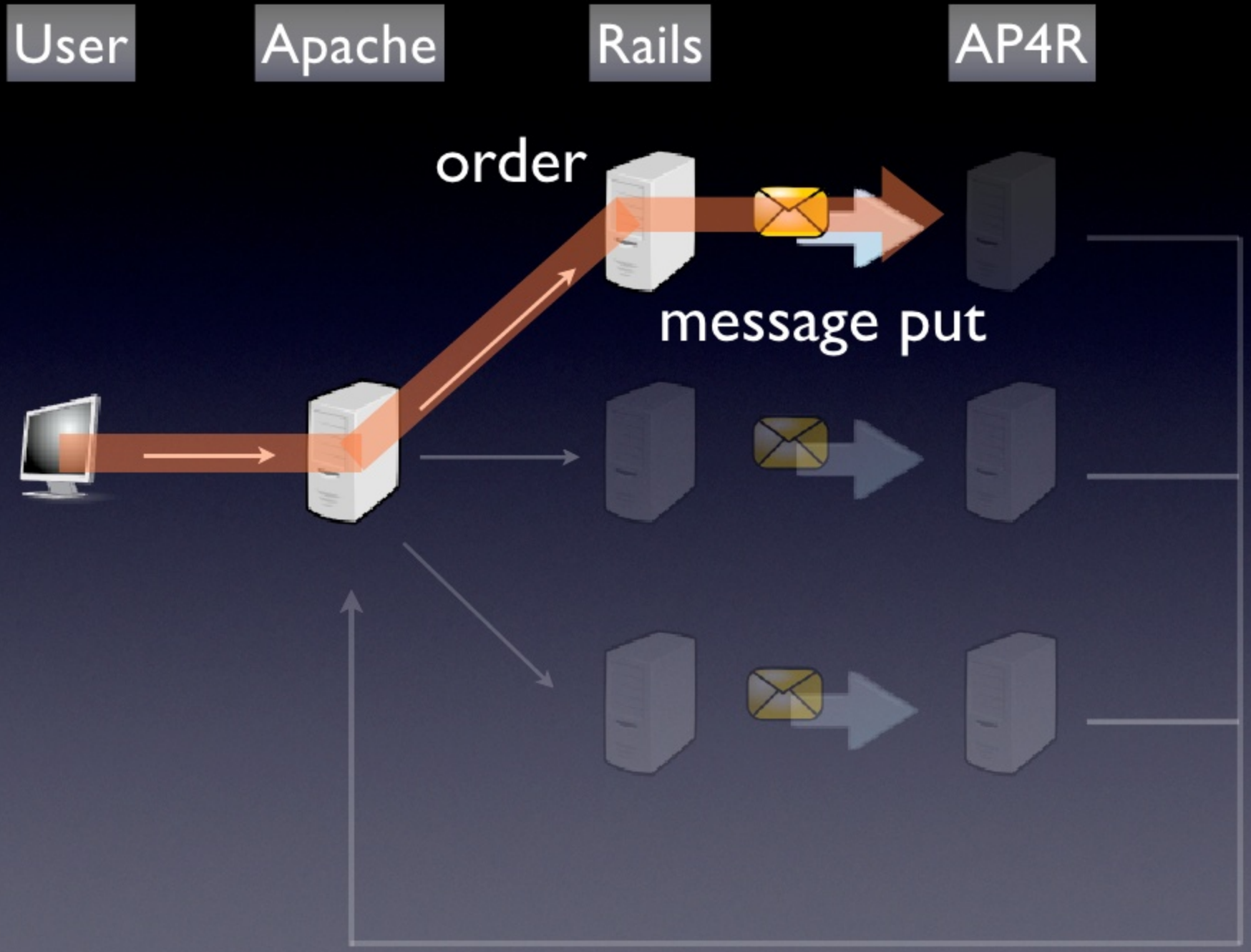
There's also block style

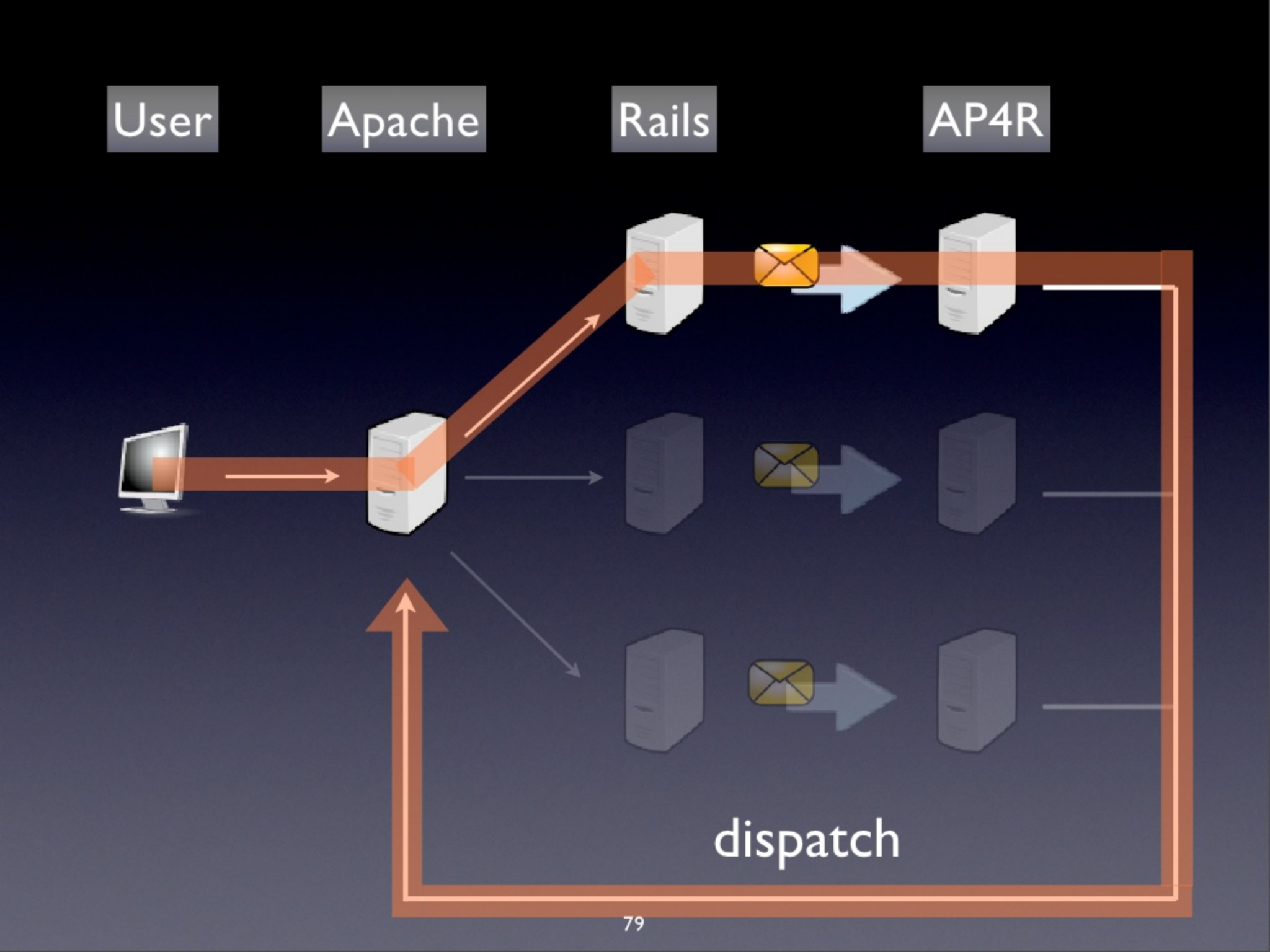
c.f. RDoc: http://ap4r.rubyforge.org/doc/

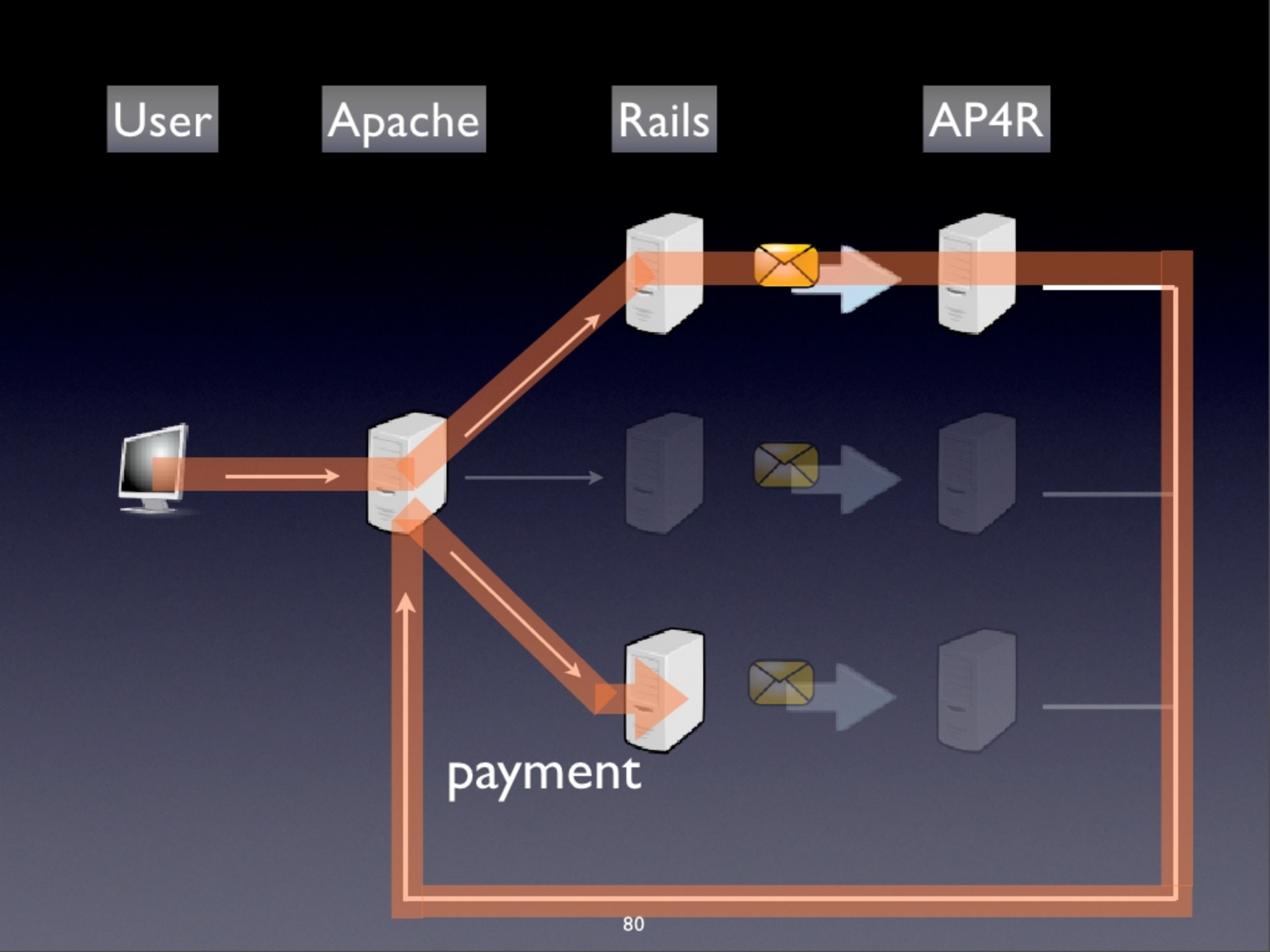
```
Class ShopController < ApplicationController

  def order                  # synchronous side
    ...
    ap4r.async_to({:action => 'payment'},
                  {:id => 1})

    redirect_to ...
  end

  def payment                # asynchronous side
    id = params[:id]
    ...
    render :text => "true"
  end

end
```

Process flow

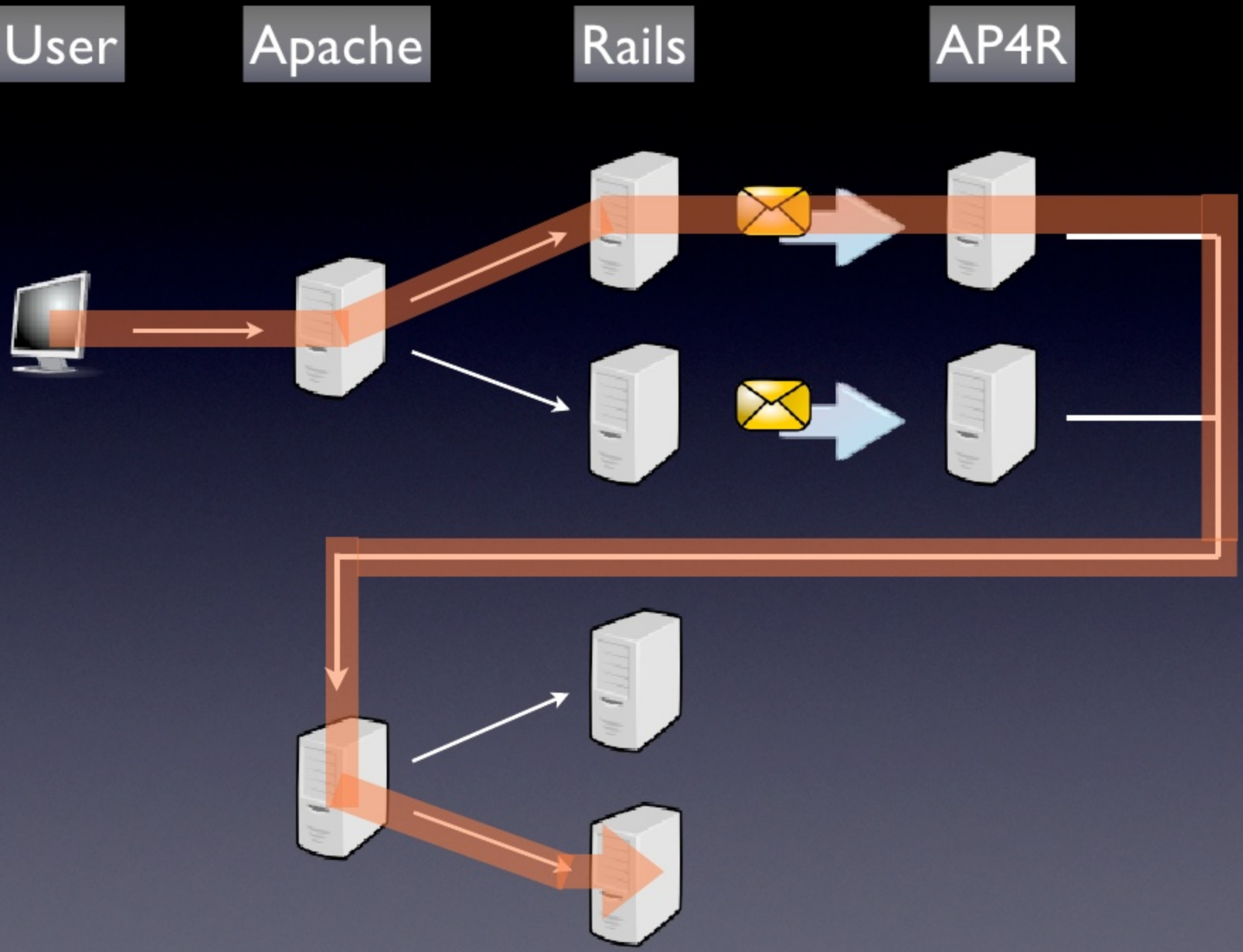

User
Apache
Rails
AP4R


User
Apache
Rails
AP4R
order
message put

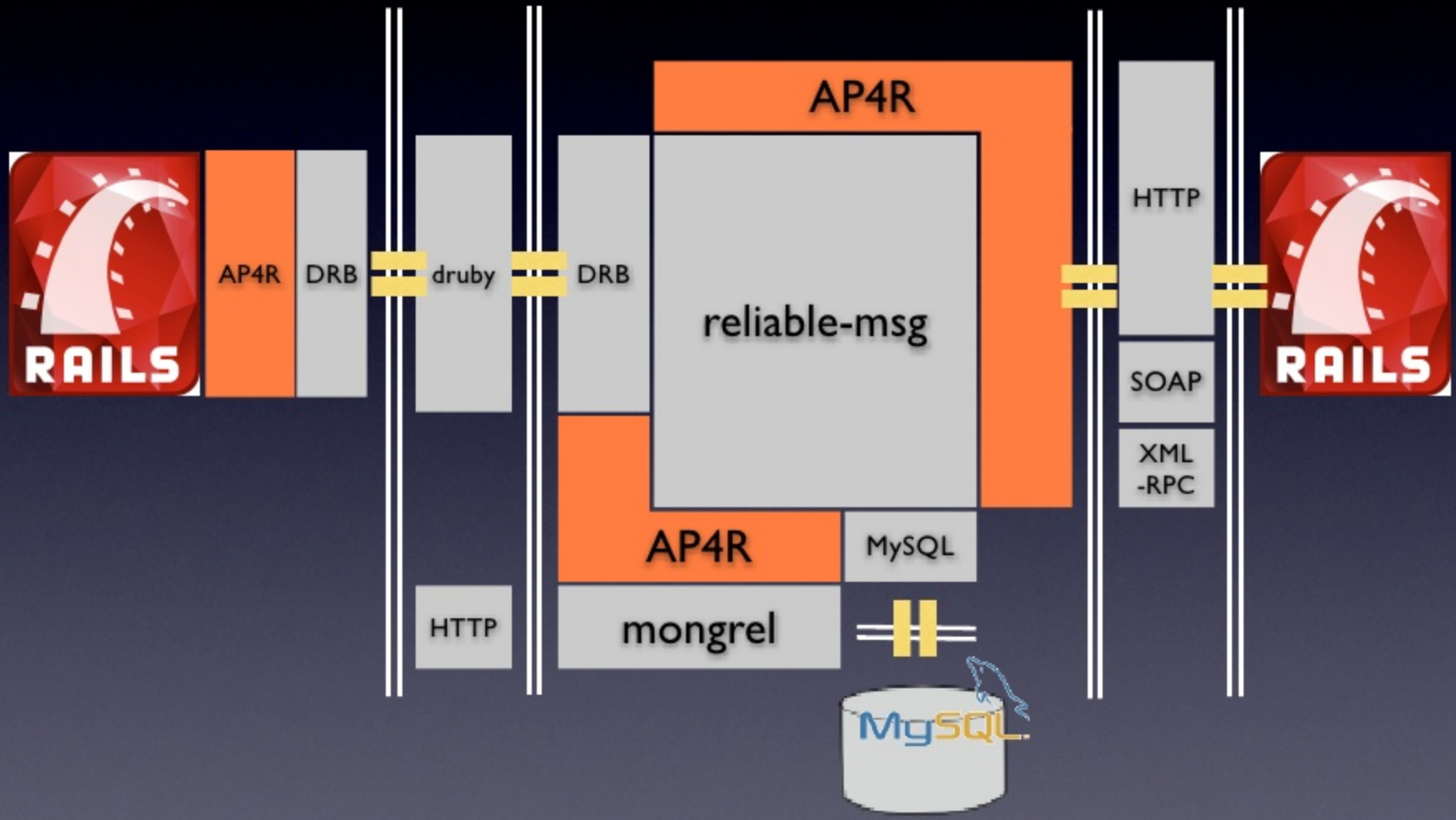

User
Apache
Rails
AP4R
dispatch


User
Apache
Rails
AP4R
payment

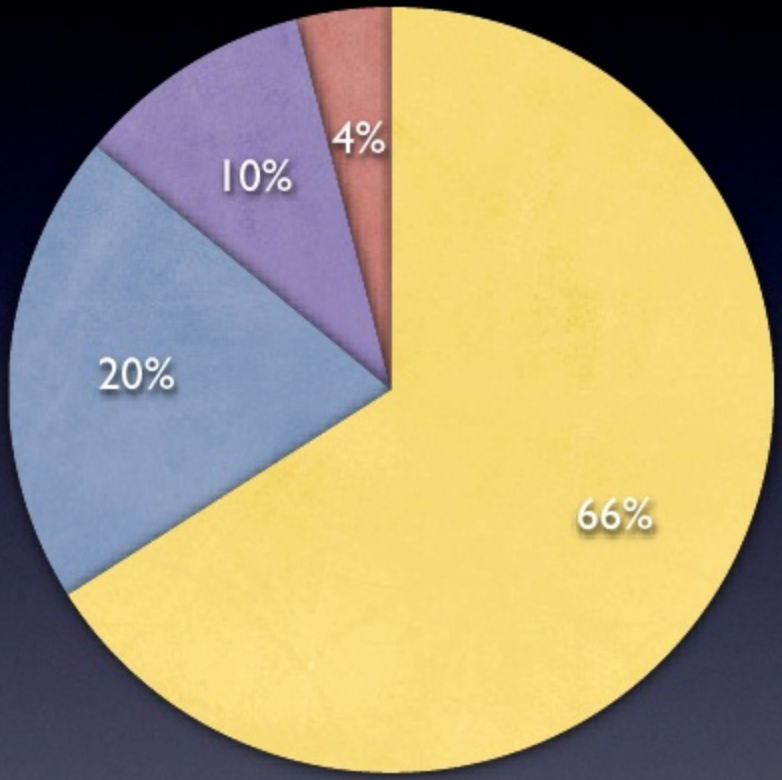
Able to separate sync/async servers


User
Apache
Rails
AP4R

Architecture



Thanks to



- DRb
- reliable-msg
- mongrel
- rails

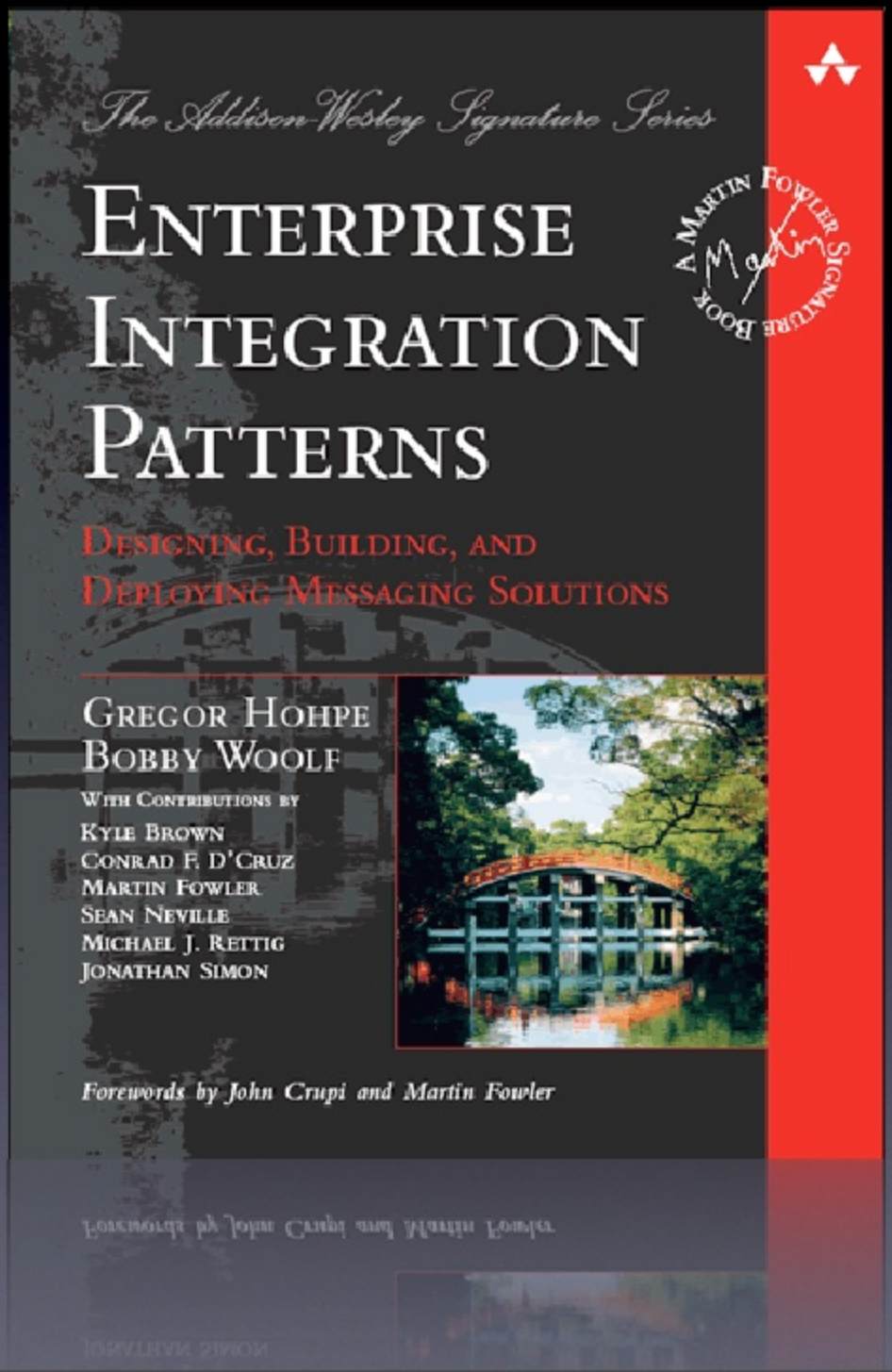
Stored And Forward

Stored ?

Forward ?

at-least-once

between Producer and Channel



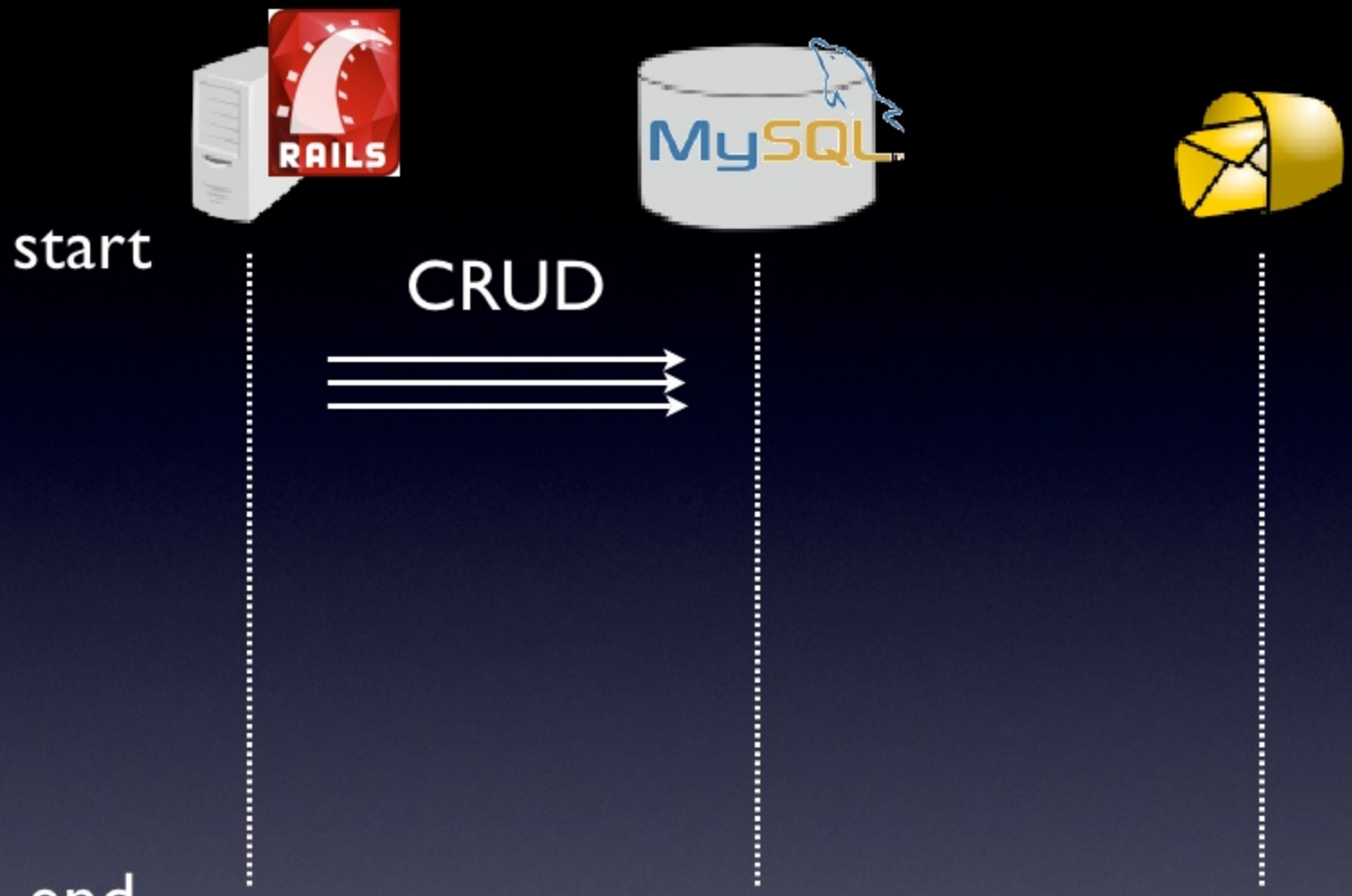
Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions by Gregor Hohpe,Bobby Woolf

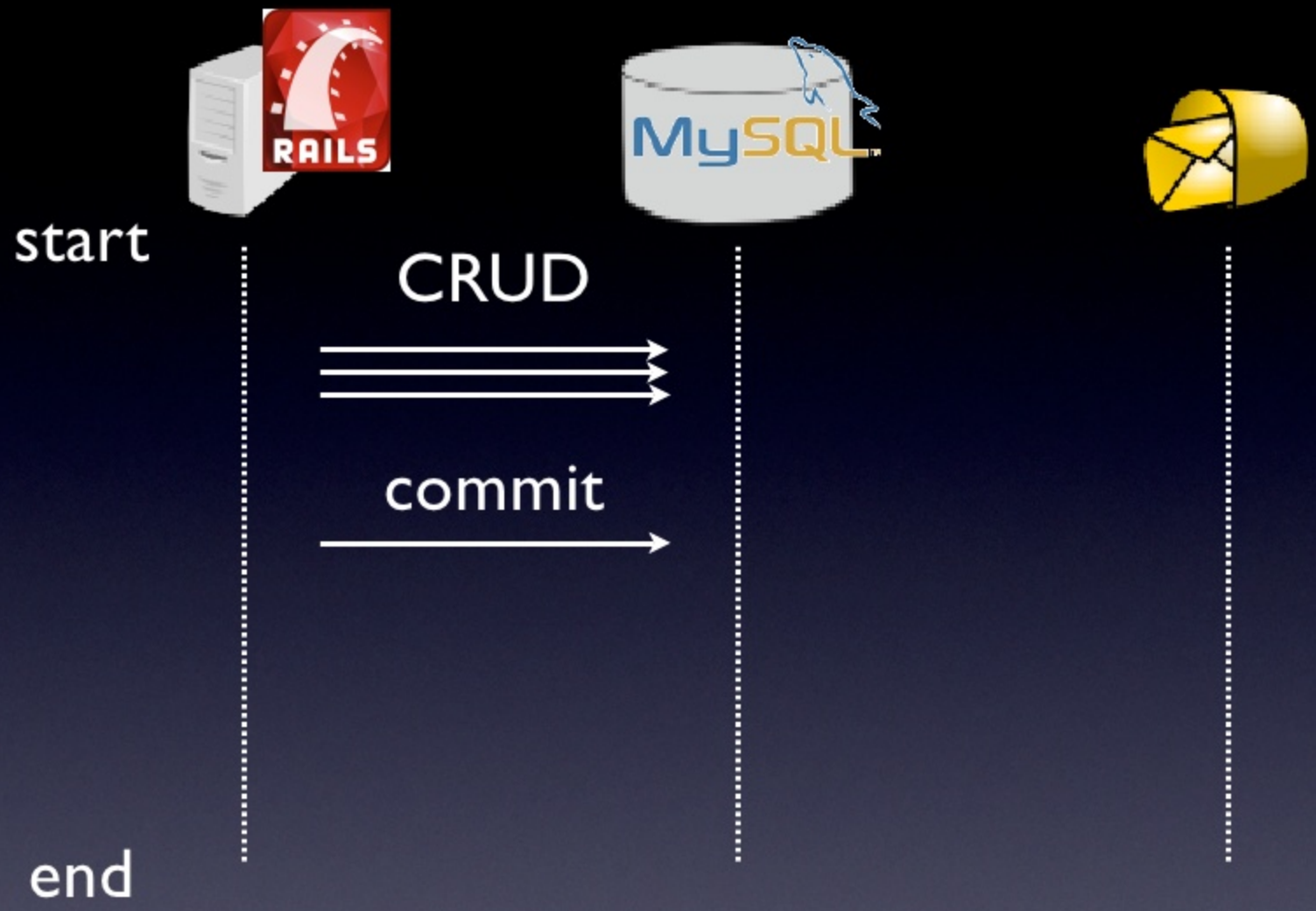
Guaranteed Delivery [122]

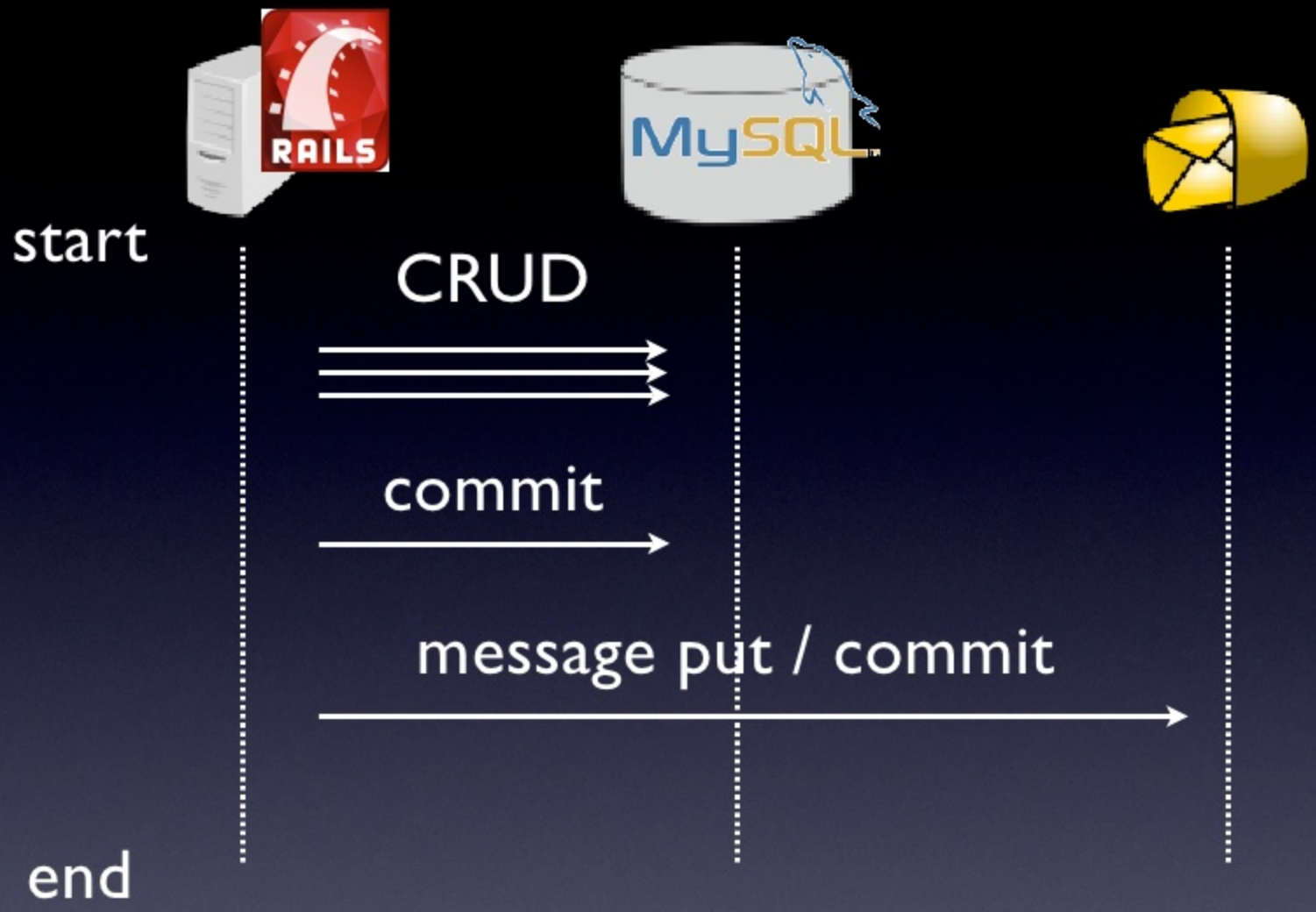
Typical flow
to create async. messages

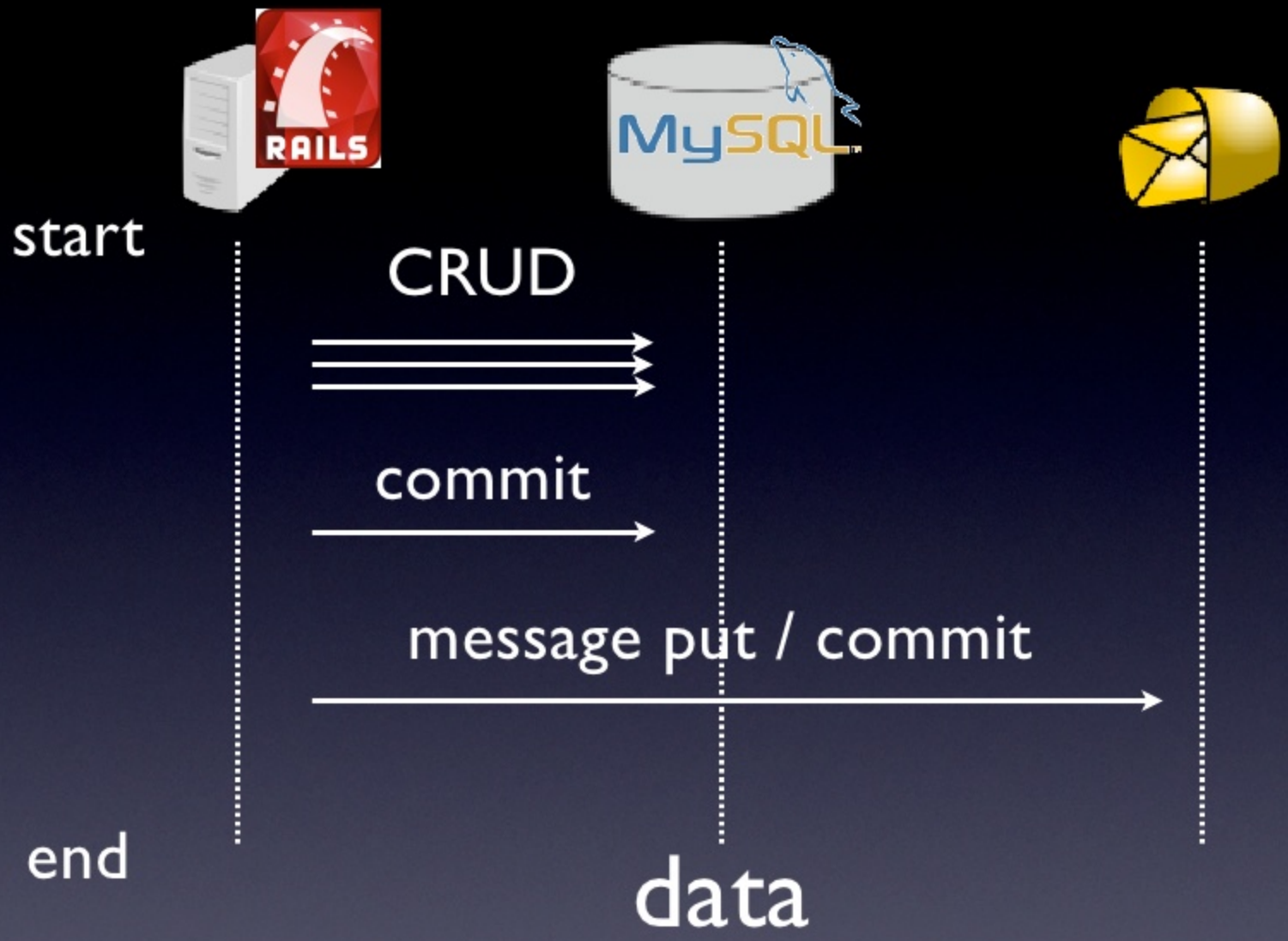
start

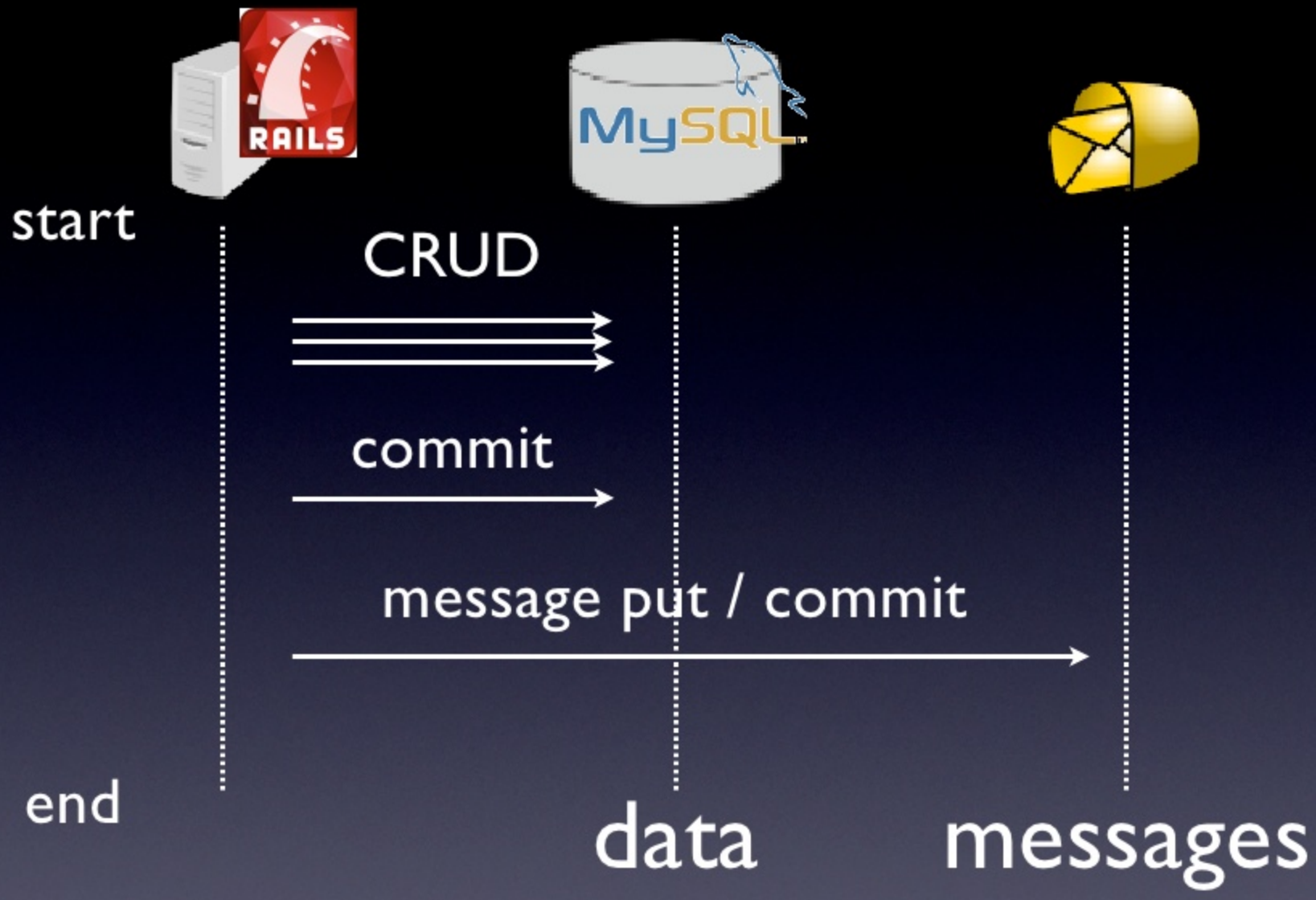
end


RAILS
MySQL
start
CRUD
end


RAILS
MySQL
start
CRUD
commit
end


RAILS
MySQL
start
CRUD
commit
message put / commit
end


RAILS
MySQL
start
CRUD
commit
message put / commit
end
data


RAILS
MySQL
start
CRUD
commit
message put / commit
end
data
messages

Ensure (some)

Atomicness

of

updating database and message creation

On updating database

- Error occurs,
- Any following messages must NOT be queued

On updating database

- Everything succeeded,
- All following messages must be queued

Relief

Lightweight

and

Robust

Ap4r::Client#transaction

```
Class ShopController < ApplicationController
  def order                    # synchronous side
    ap4r.transaction do
      ...                      # CRUD on database

      ap4r.async_to ...
    end

    redirect_to ...
  end
end
```

Process flow changes in queueing sequence

Behind ap4r.transaction

```
ap4r.transaction do
  ...                  # CRUD on database

  ap4r.async_to(...)

end
```

Behind ap4r.transaction

```
ap4r.transaction do
  ...                    # CRUD on database

  ap4r.async_to(...)
```

Insert into SAF table

```
end
```

Behind ap4r.transaction

```
ap4r.transaction do
  ...                    # CRUD on database

  ap4r.async_to(...)
```

Insert into SAF table

end

Commit database

Behind ap4r.transaction

```
ap4r.transaction do
  ...                    # CRUD on database

  ap4r.async_to(...)
```

Insert into SAF table

end

Commit database

Queue messages

Behind ap4r.transaction

```
ap4r.transaction do
  ...                    # CRUD on database

  ap4r.async_to(...)
```

Insert into SAF table

Store

end

Commit database

Queue messages

Behind ap4r.transaction

```
ap4r.transaction do
  ...                    # CRUD on database

  ap4r.async_to(...)
end
```

Insert into SAF table

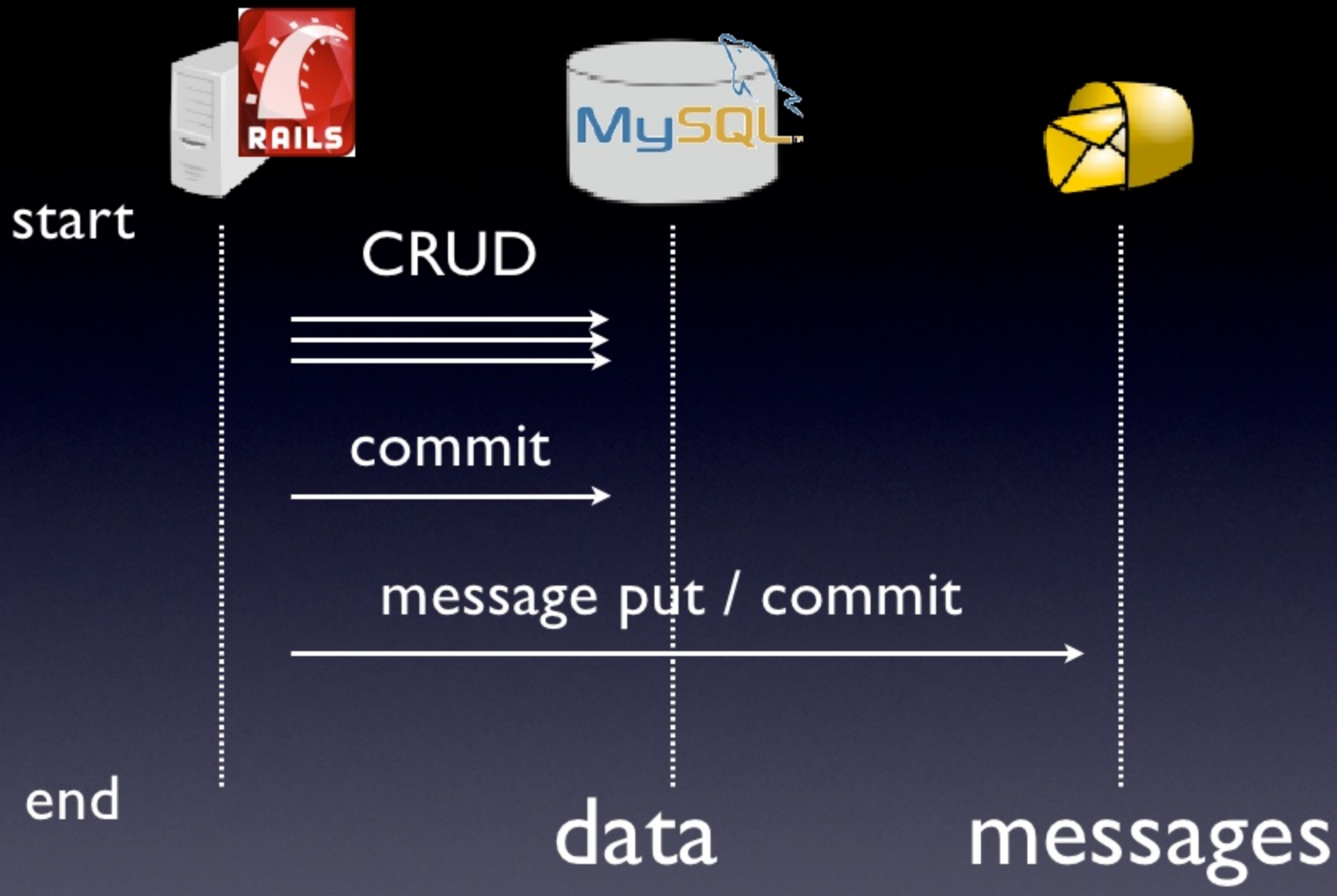
Store

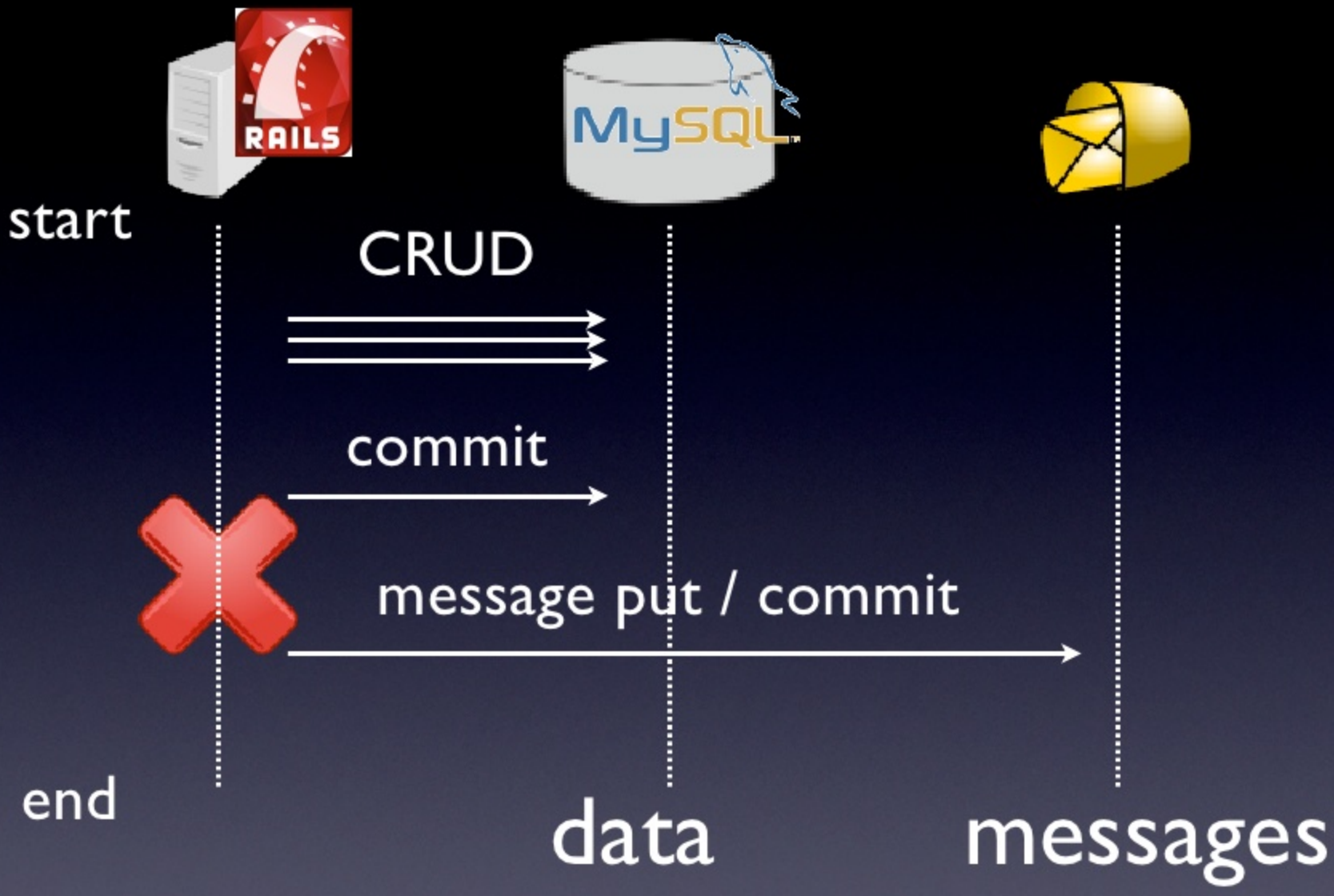
Commit database

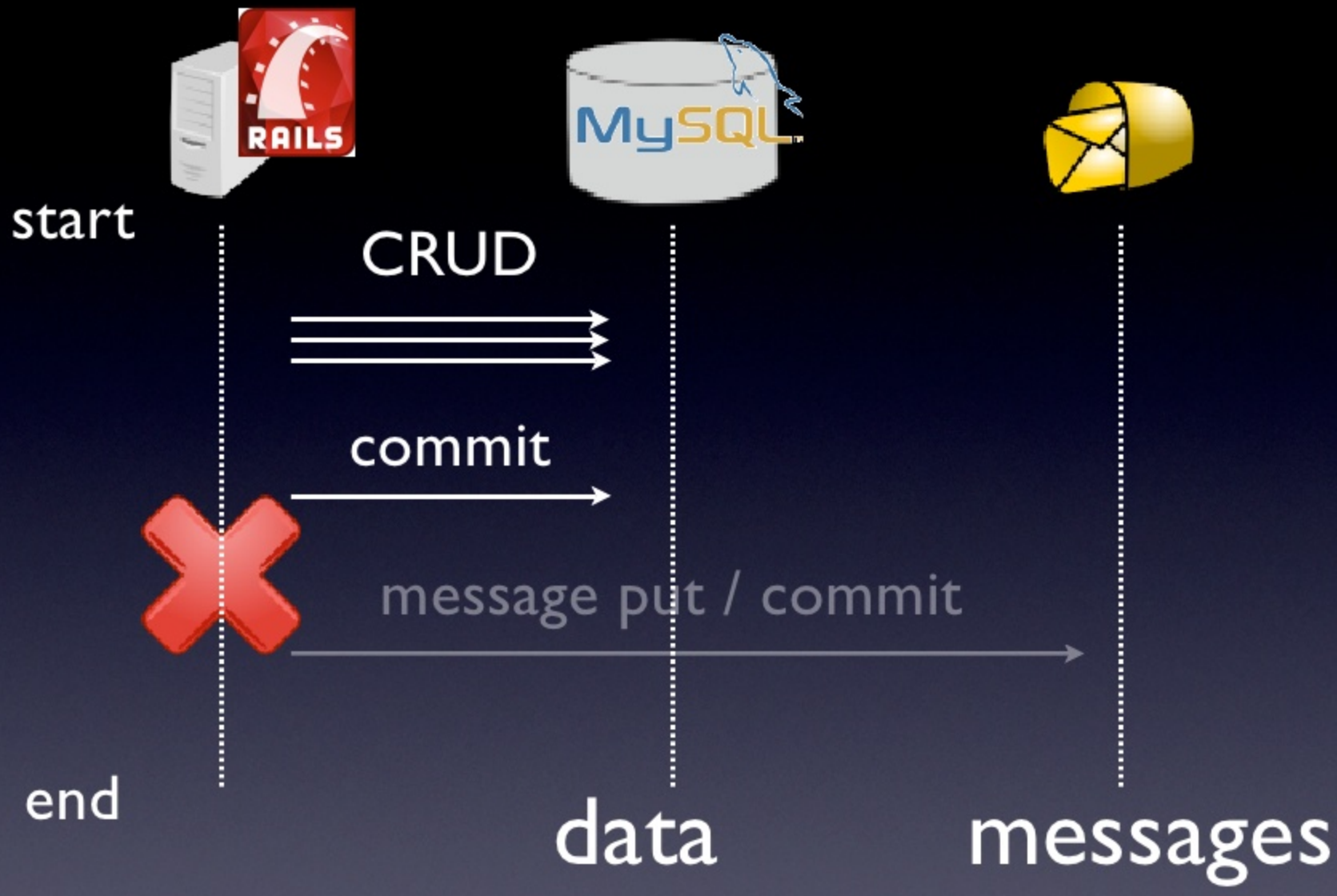
Queue messages

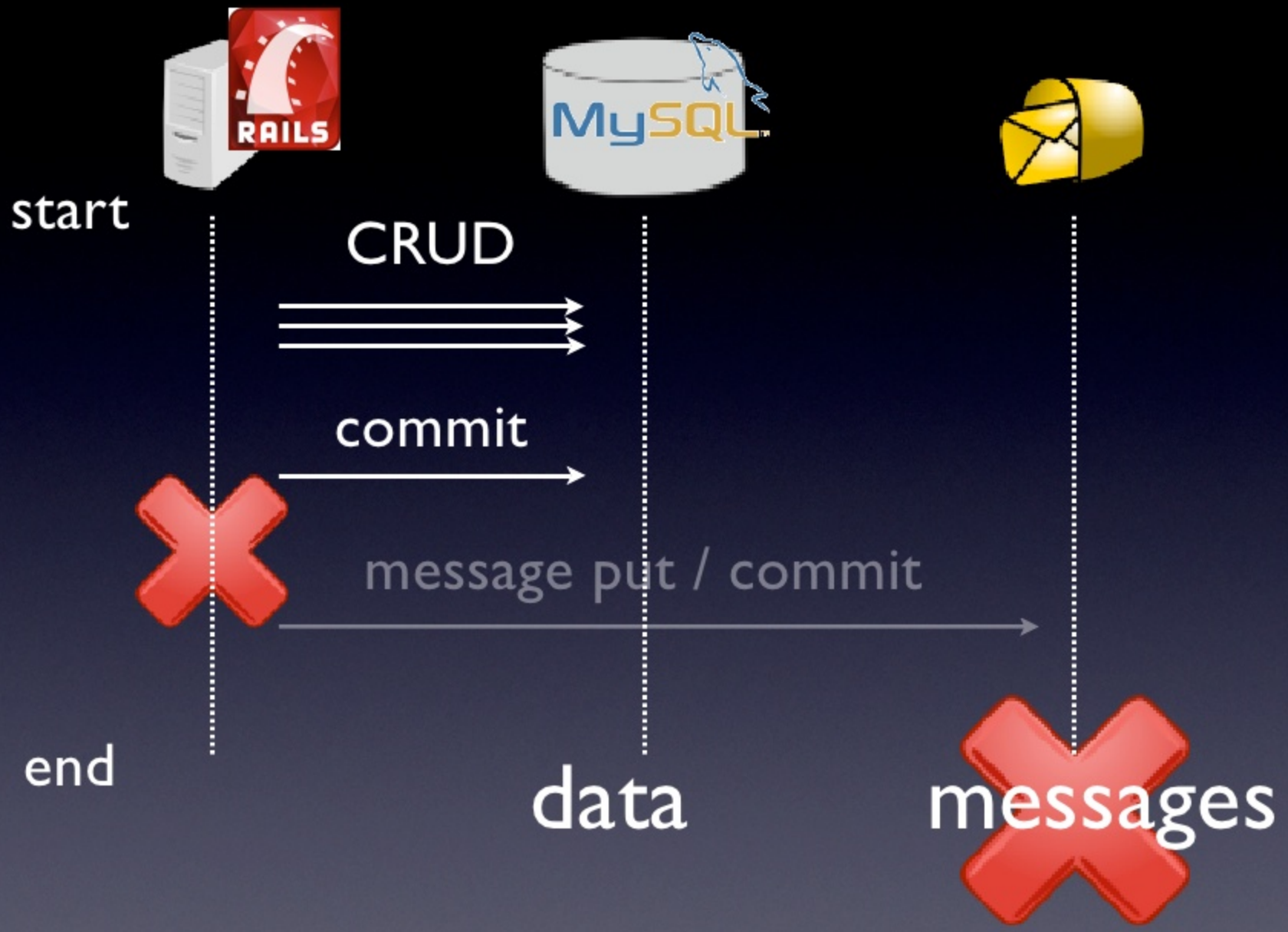
Forward

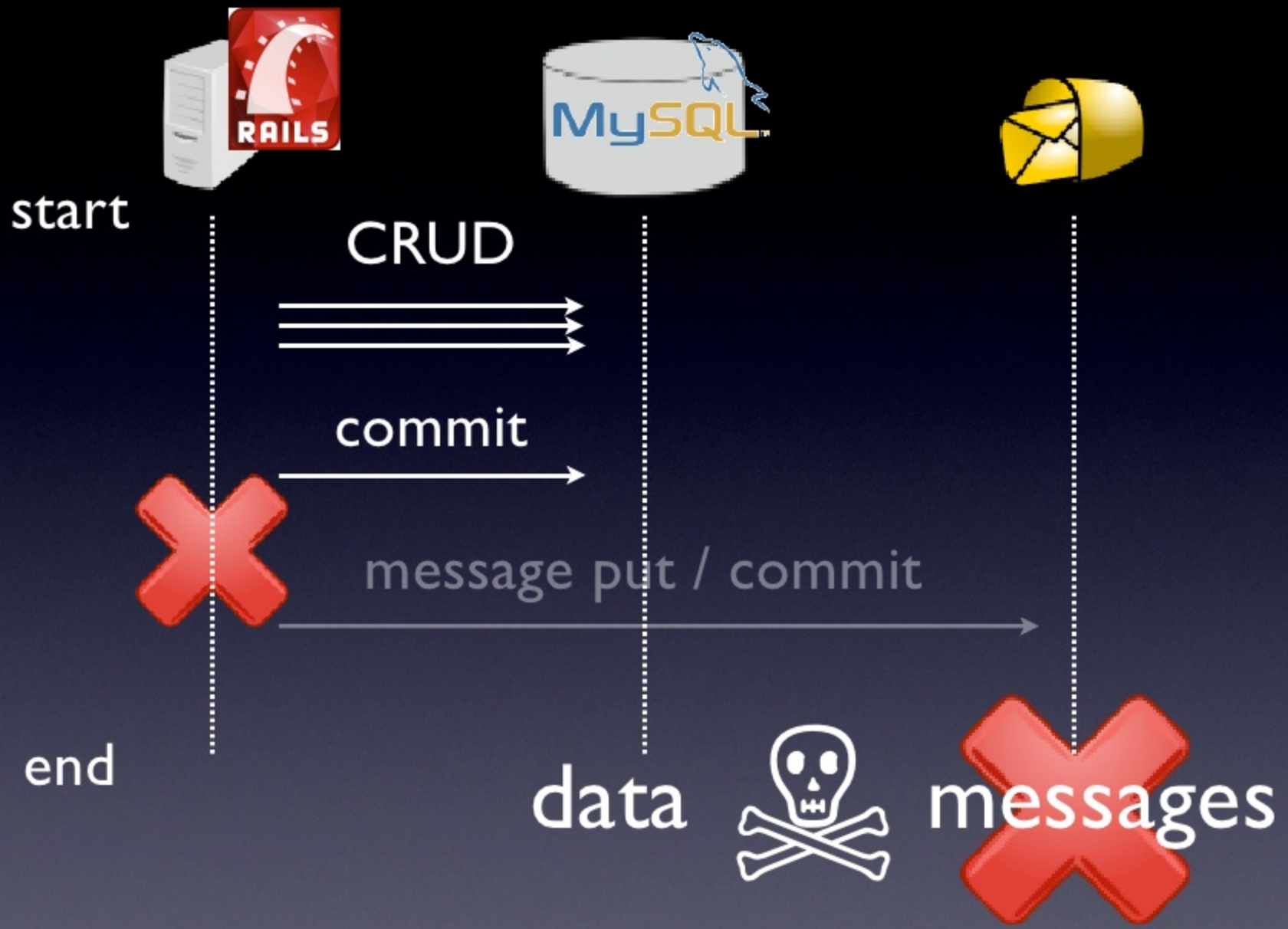
Sequence in detail
Without SAF

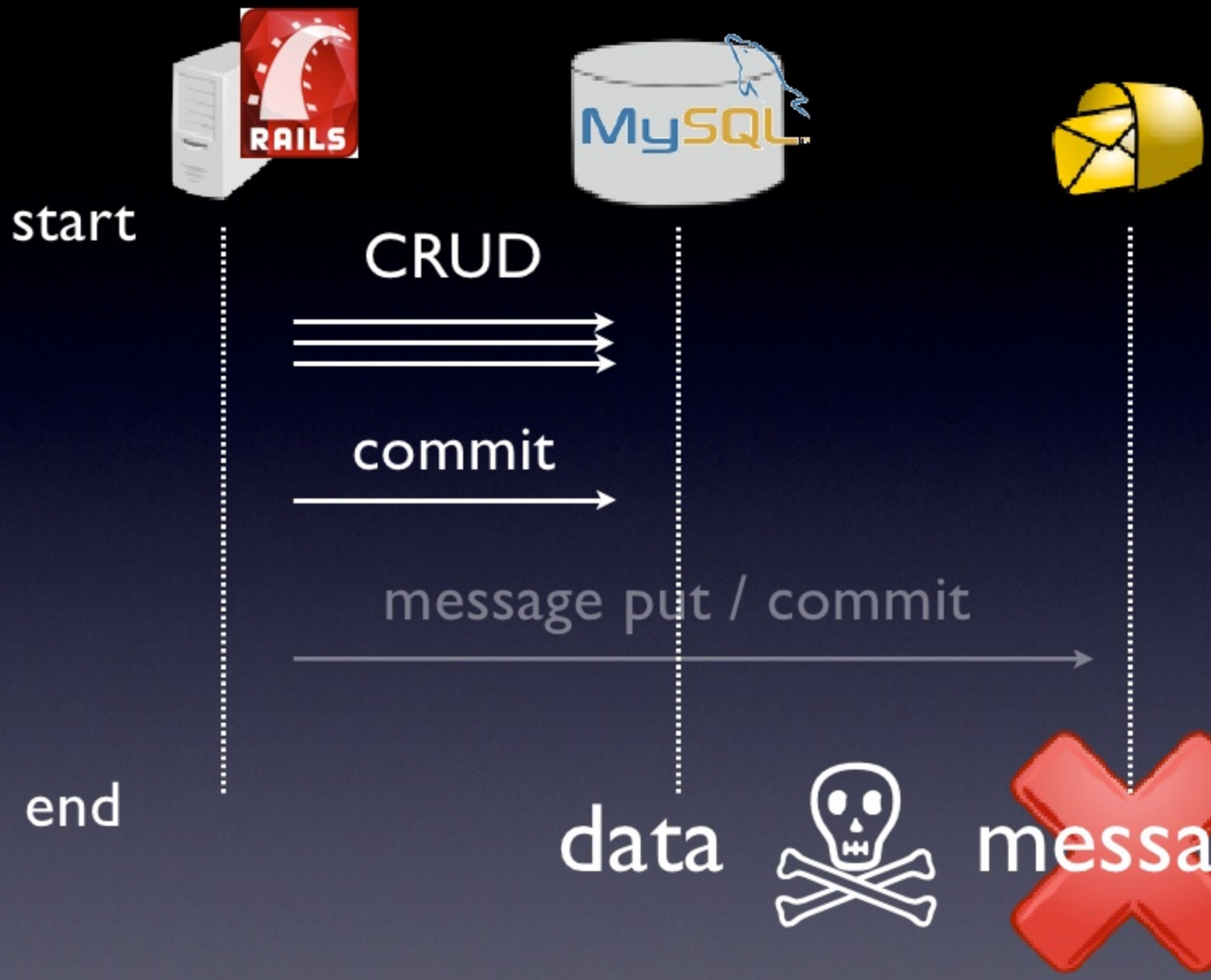

RAILS
MySQL
start
CRUD
commit
message put / commit
end
data
messages

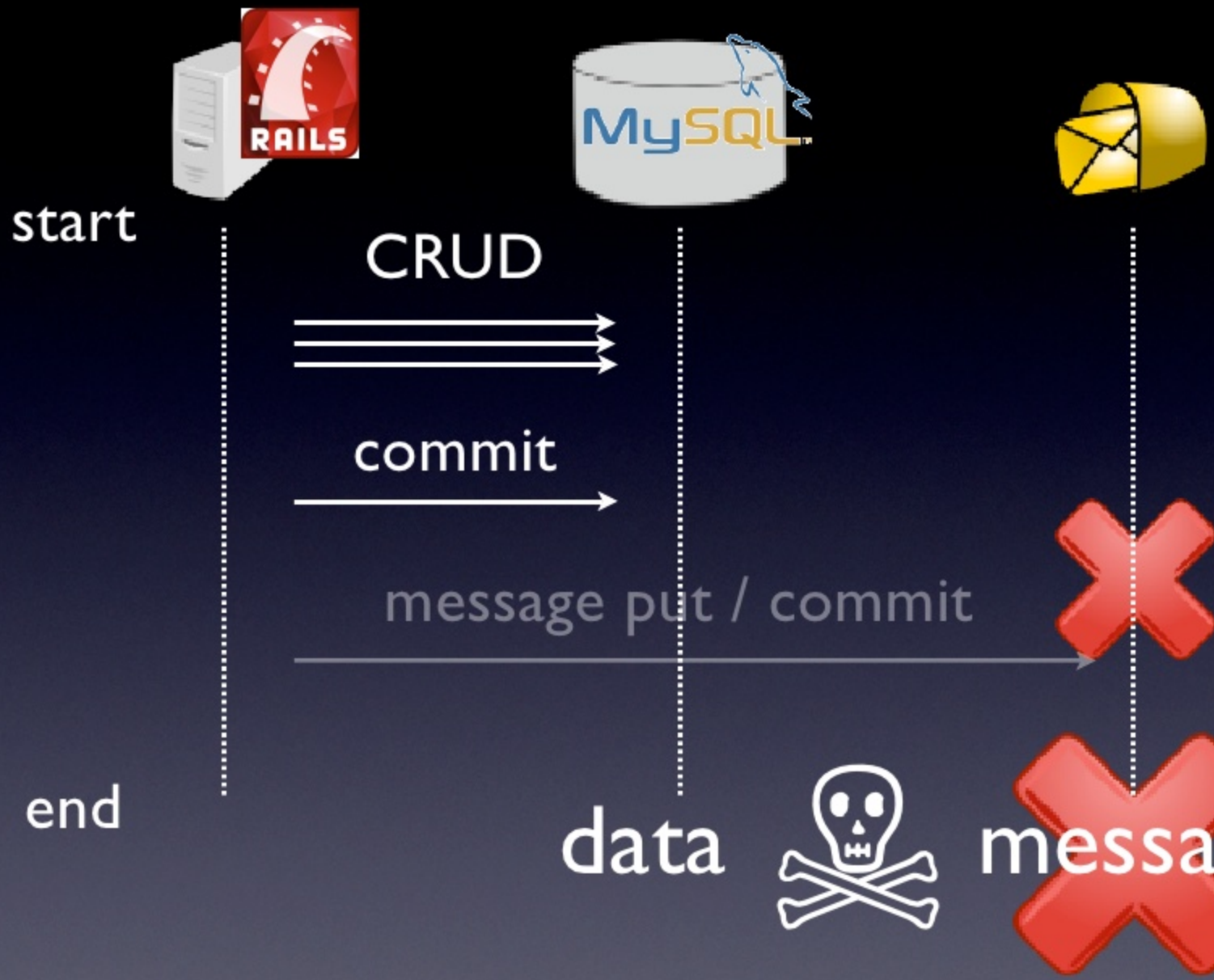

RAILS
MySQL
start
CRUD
commit
message put / commit
end
data
messages

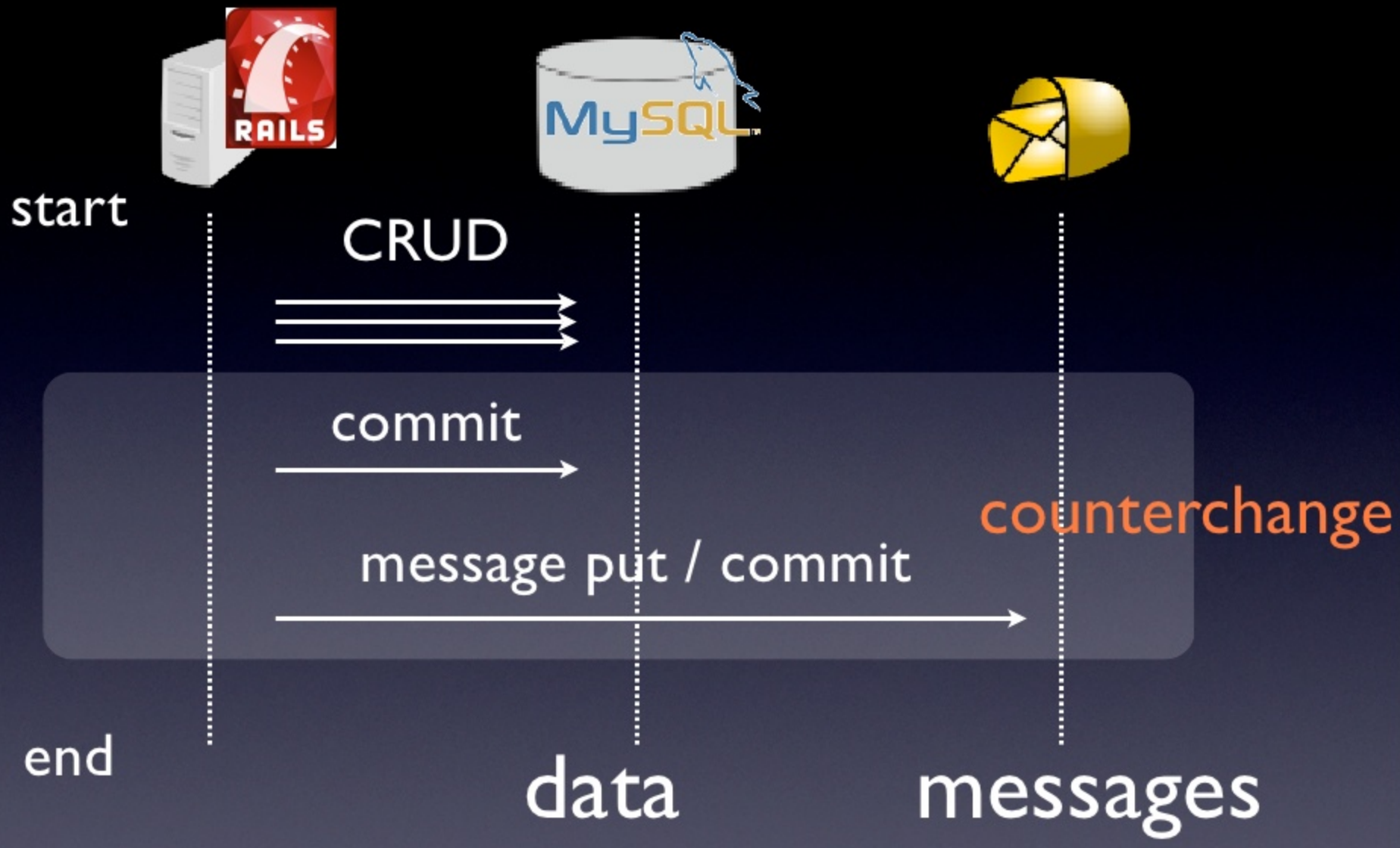

RAILS
MySQL
start
CRUD
commit
message put / commit
end
data
messages

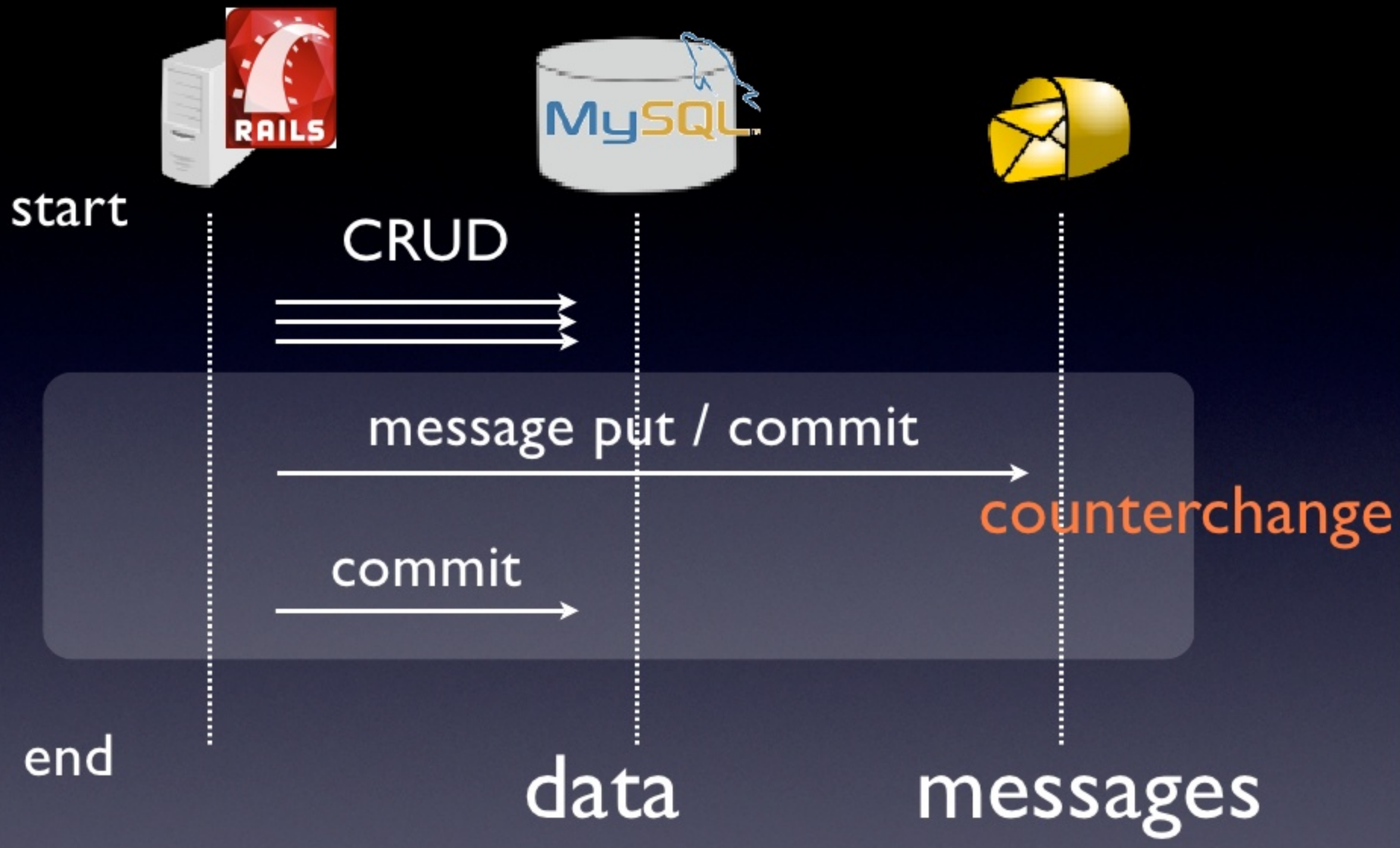

RAILS
MySQL
start
CRUD
commit
message put / commit
end
data
messages

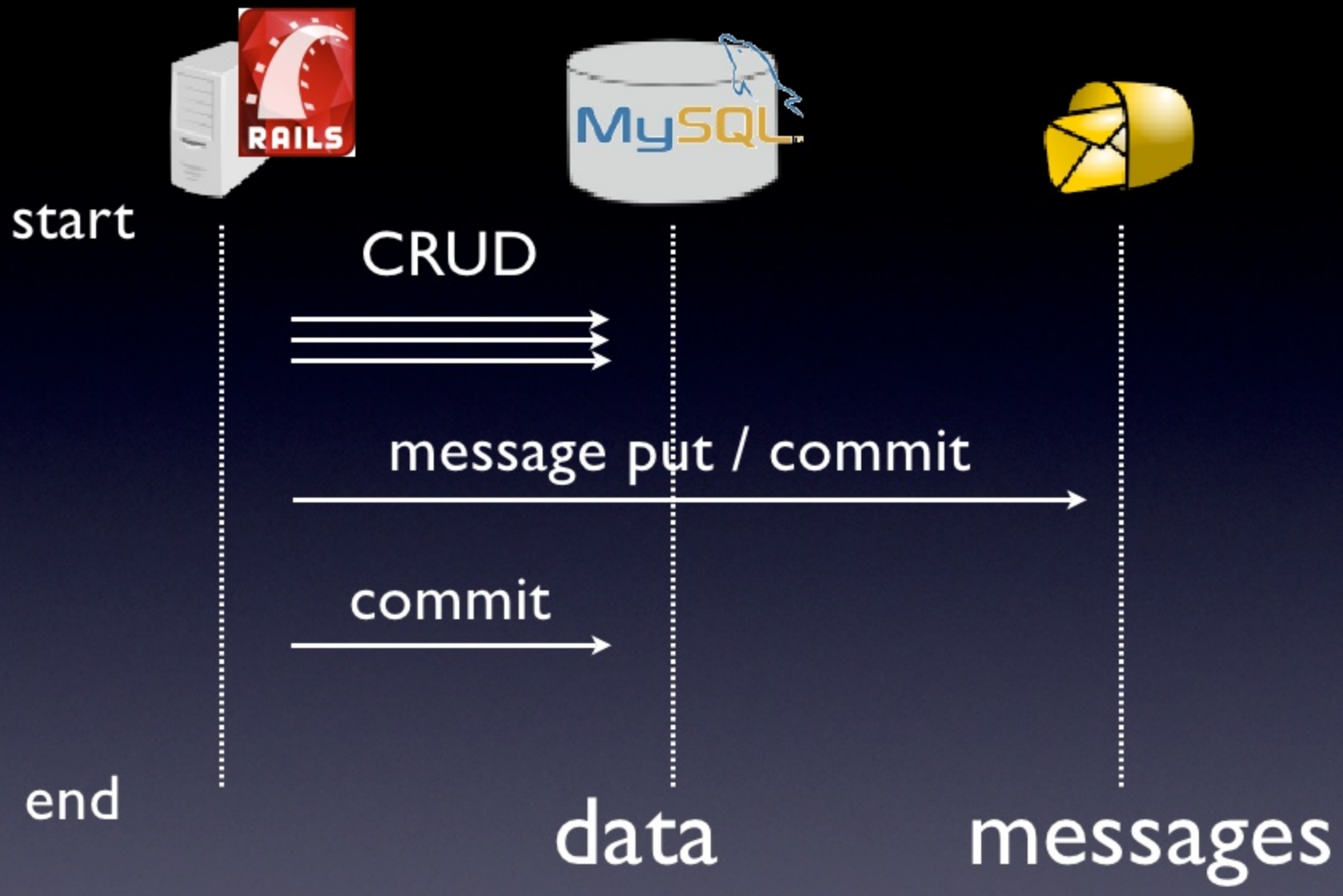

RAILS
MySQL
start
CRUD
commit
message put / commit
end
data
messages

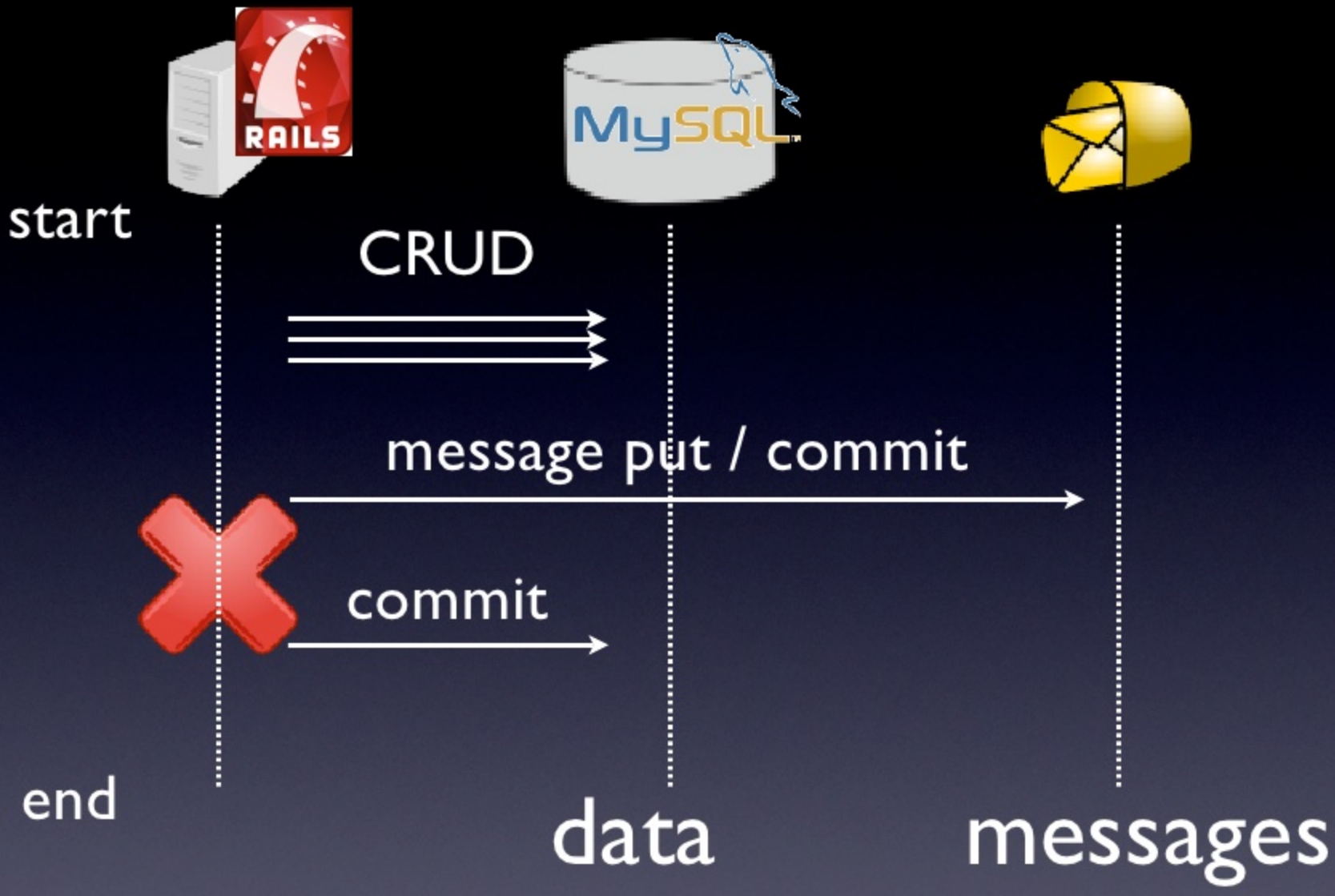

RAILS
MySQL
start
CRUD
commit
message put / commit
end
data
messages

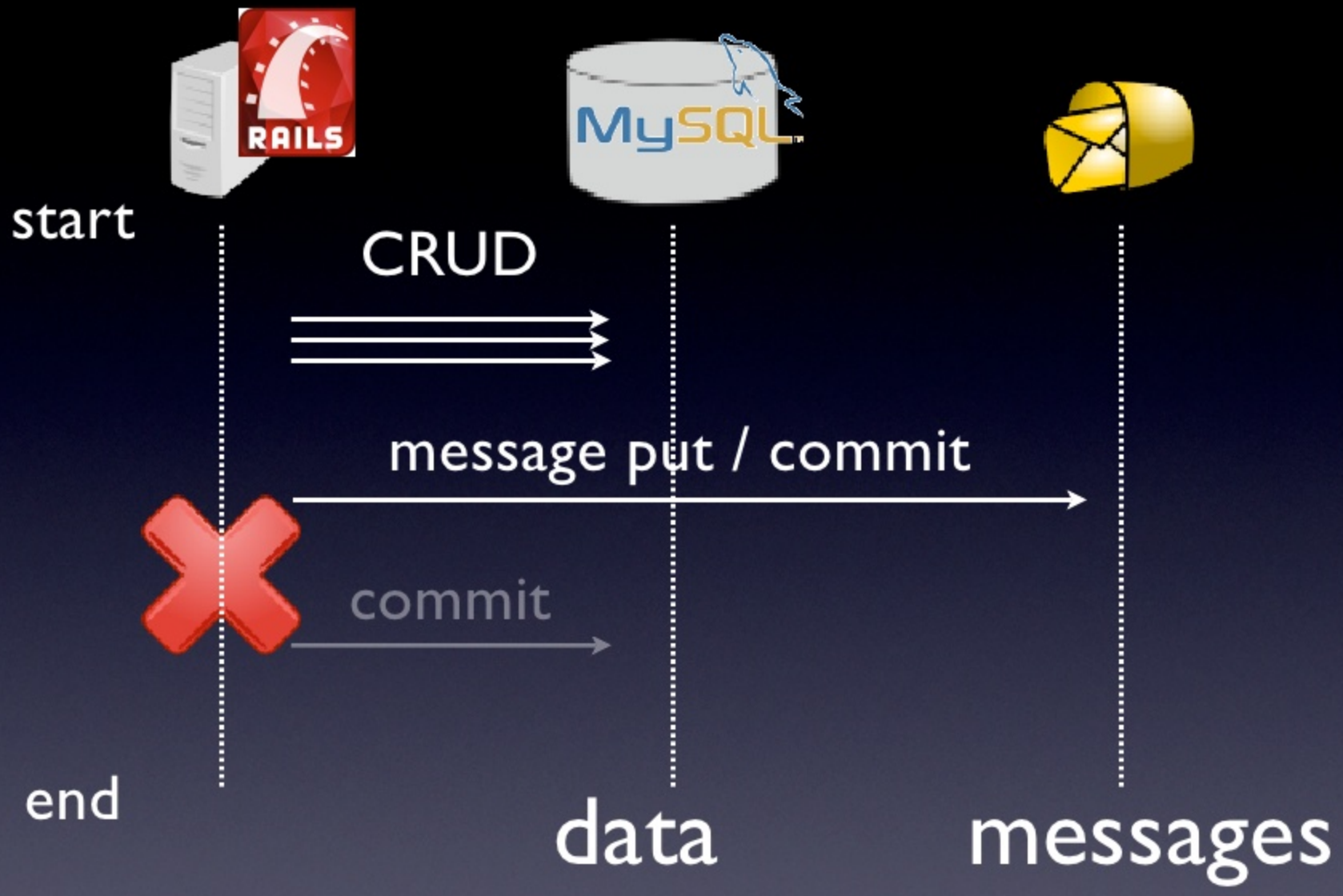

RAILS
MySQL
start
CRUD
commit
message put / commit
end
data
messages

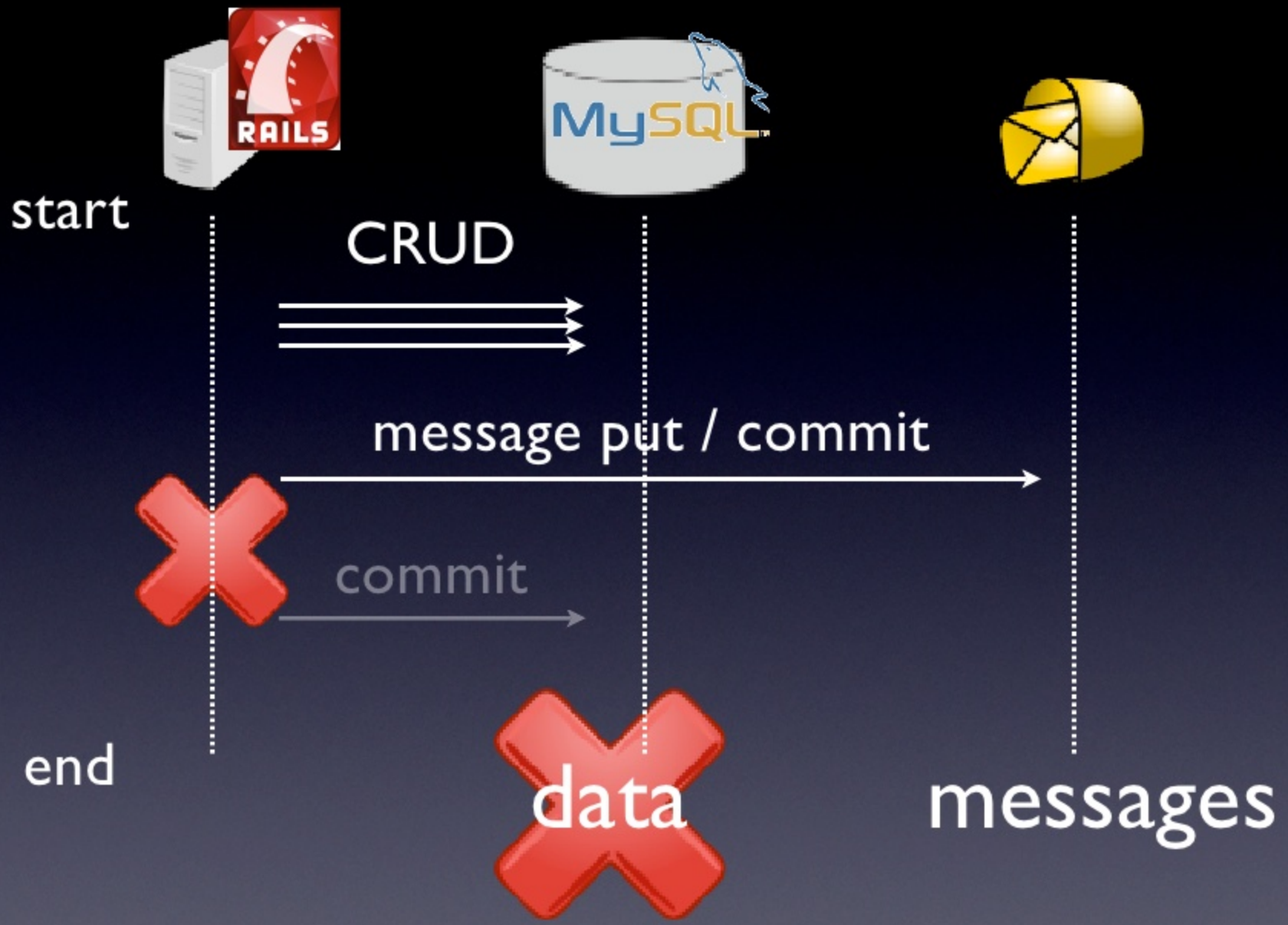

RAILS
MySQL
start
CRUD
commit
counterchange
message put / commit
end
data
messages

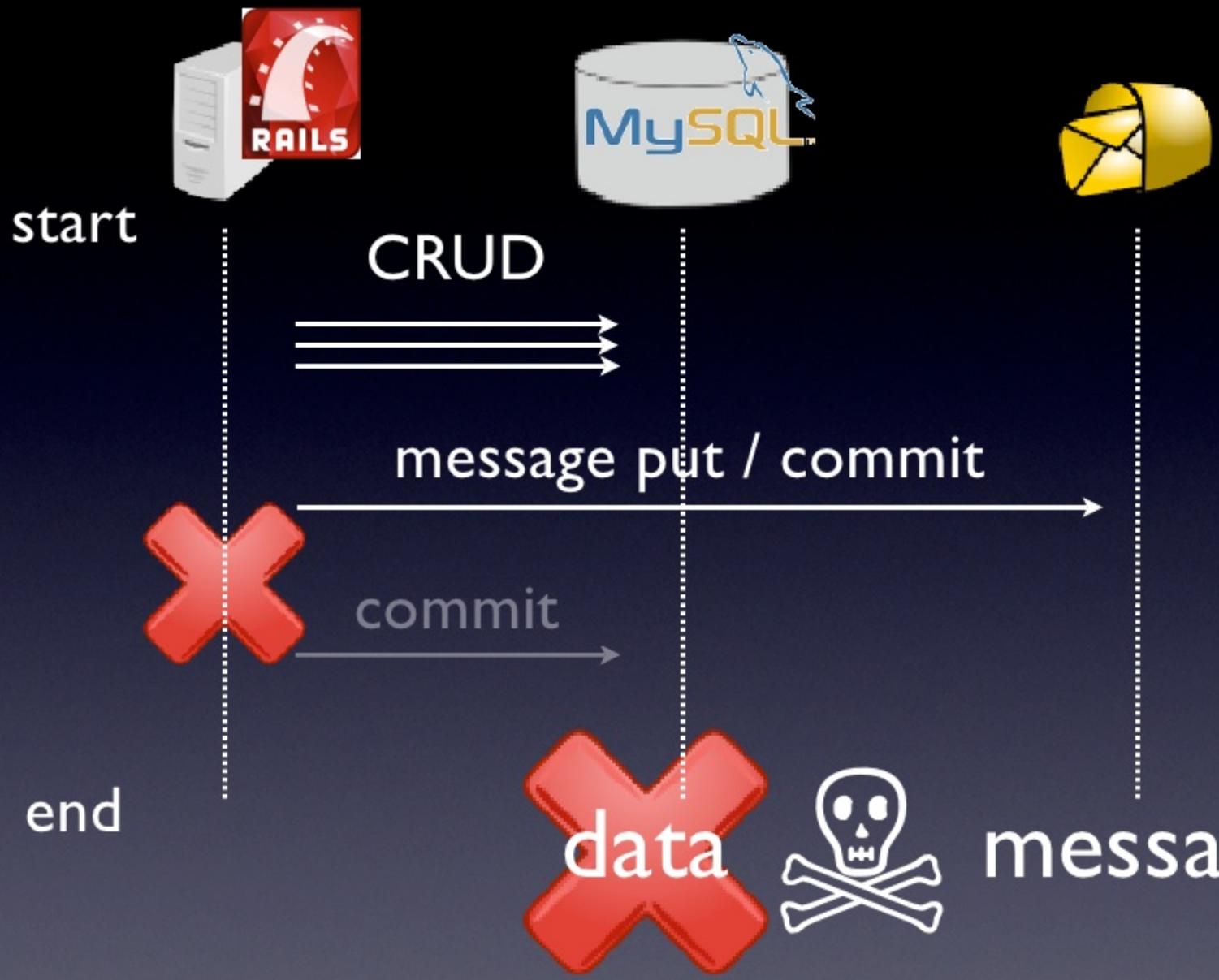

RAILS
MySQL
start
CRUD
message put / commit
counterchange
commit
end
data
messages

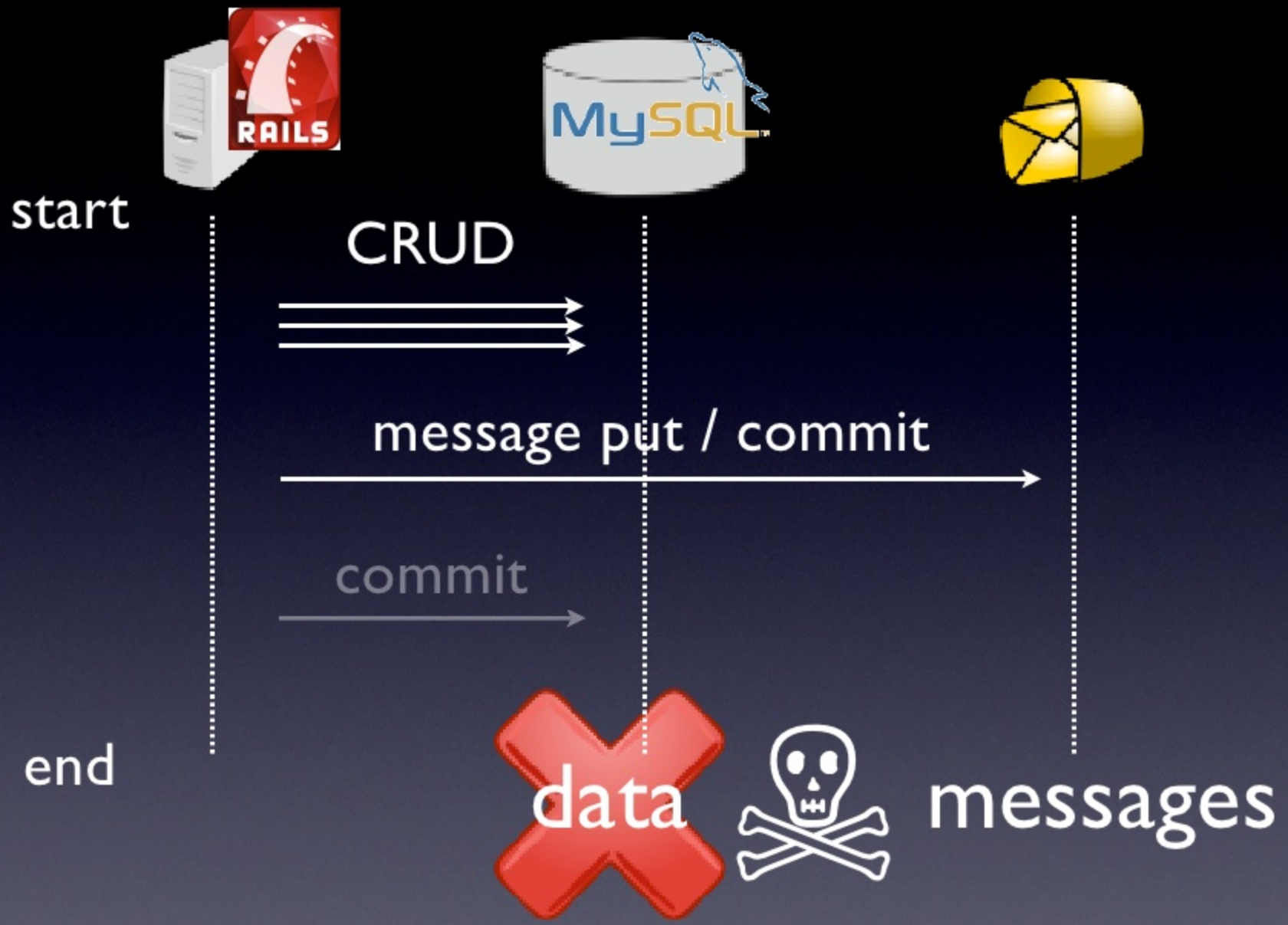

RAILS
MySQL
start
CRUD
message put / commit
commit
end
data
messages

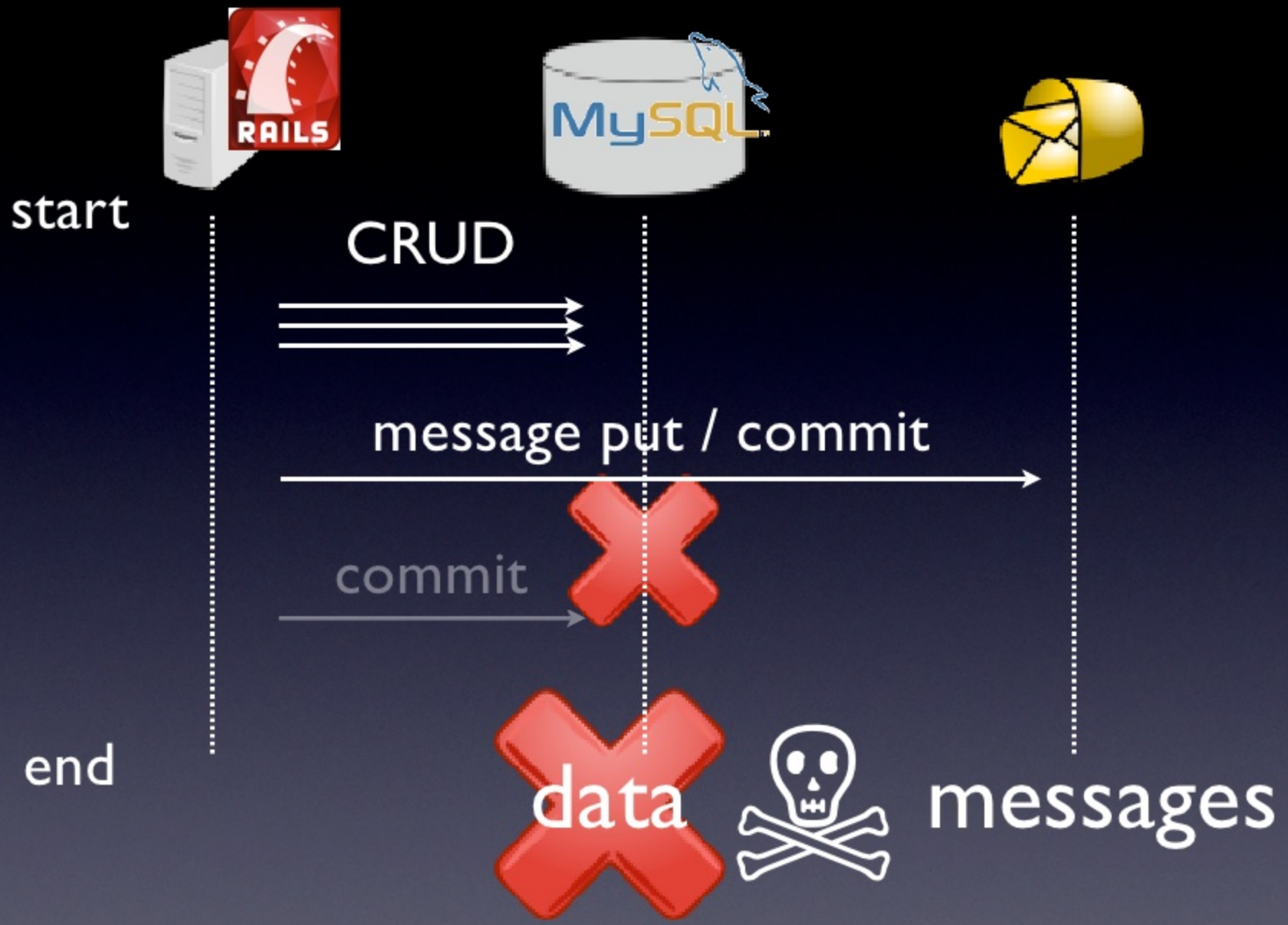

RAILS
MySQL
start
CRUD
message put / commit
commit
end
data
messages


RAILS
MySQL
start
CRUD
message put / commit
commit
end
data
messages


RAILS
MySQL
start
CRUD
message put / commit
commit
end
data
messages


RAILS
MySQL
start
CRUD
message put / commit
commit
end
data
messages


RAILS
MySQL
start
CRUD
message put / commit
commit
end
data
messages

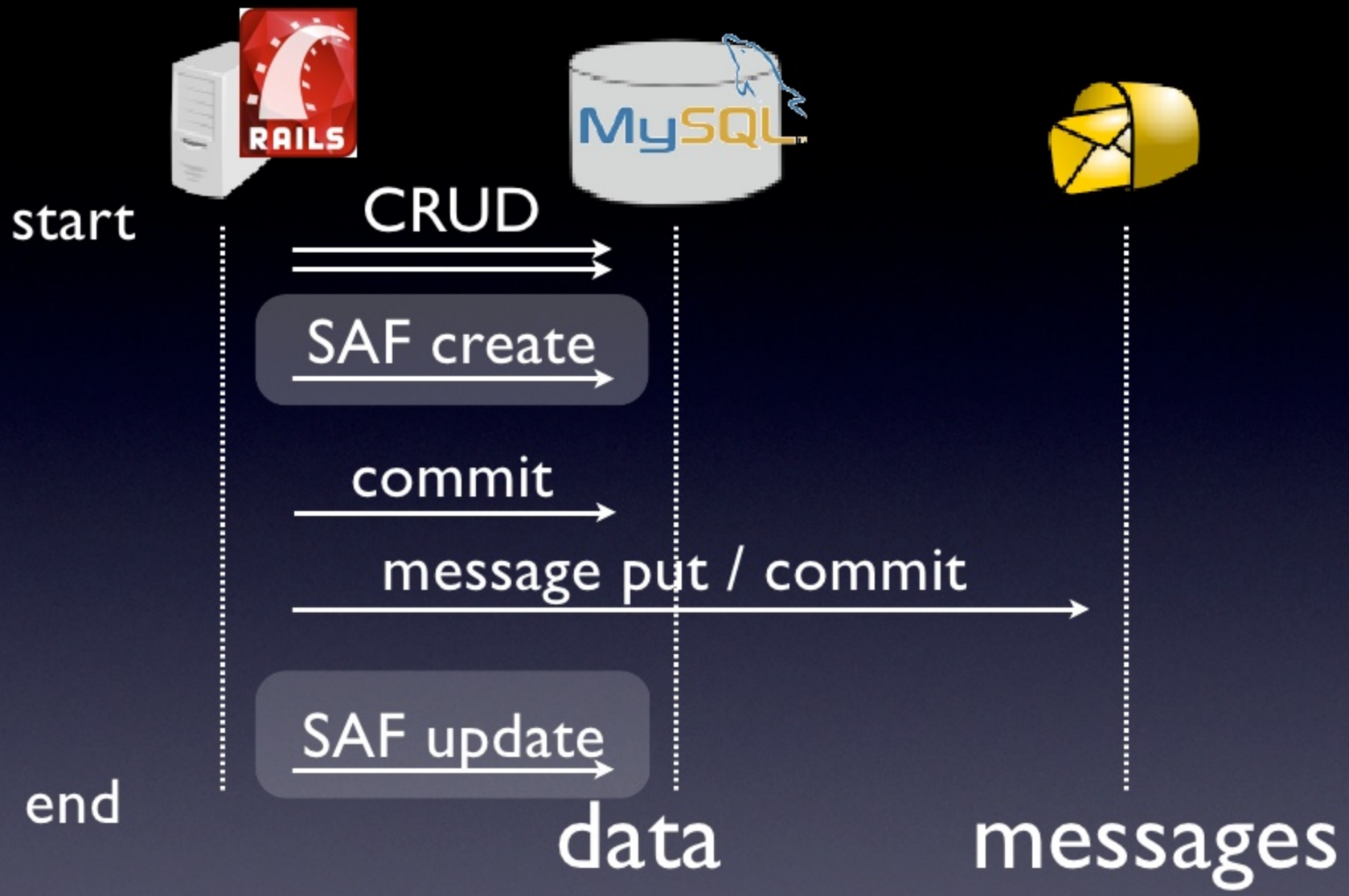

RAILS
MySQL
start
CRUD
message put / commit
commit
end
data
messages

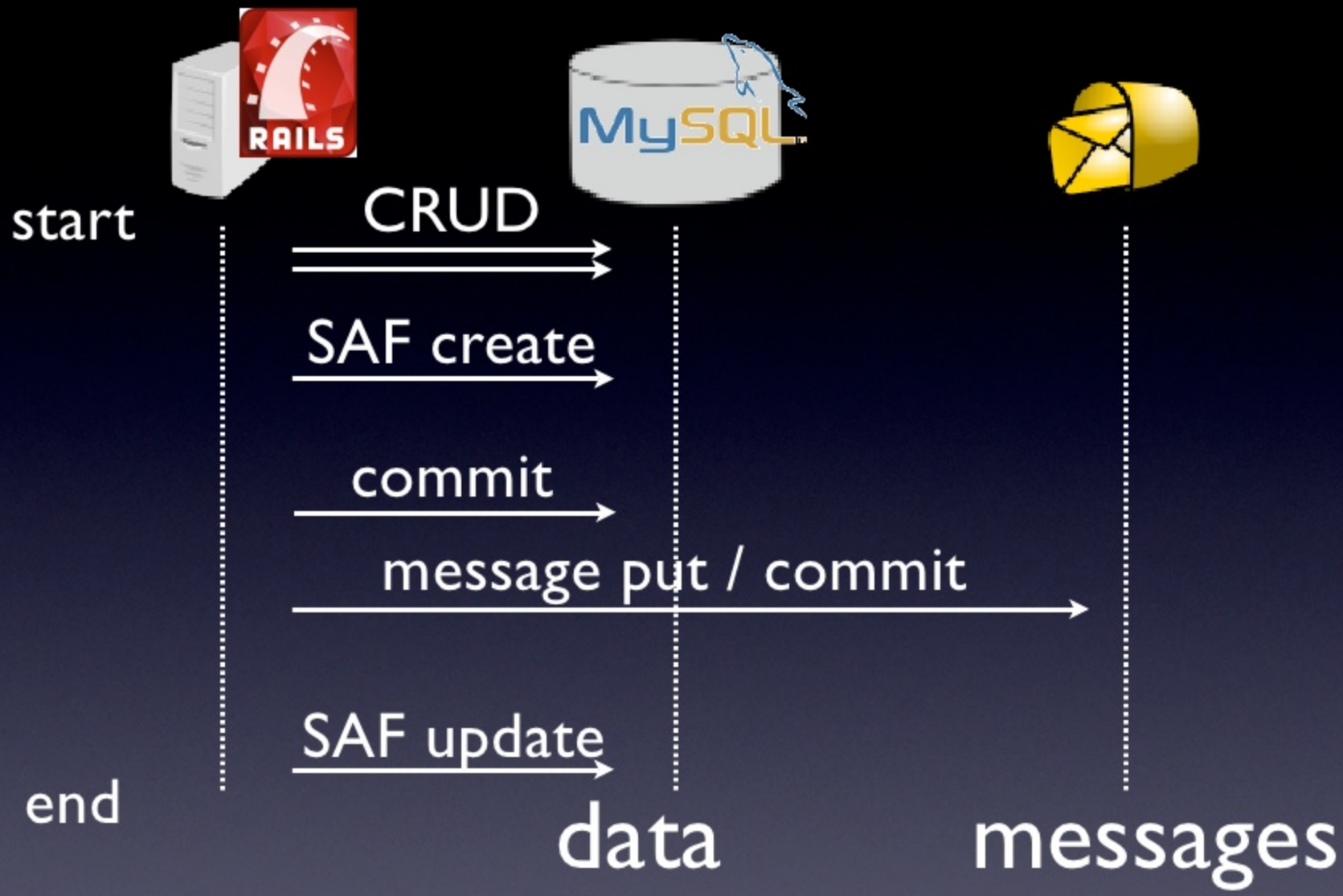
Better Safe than Sorry

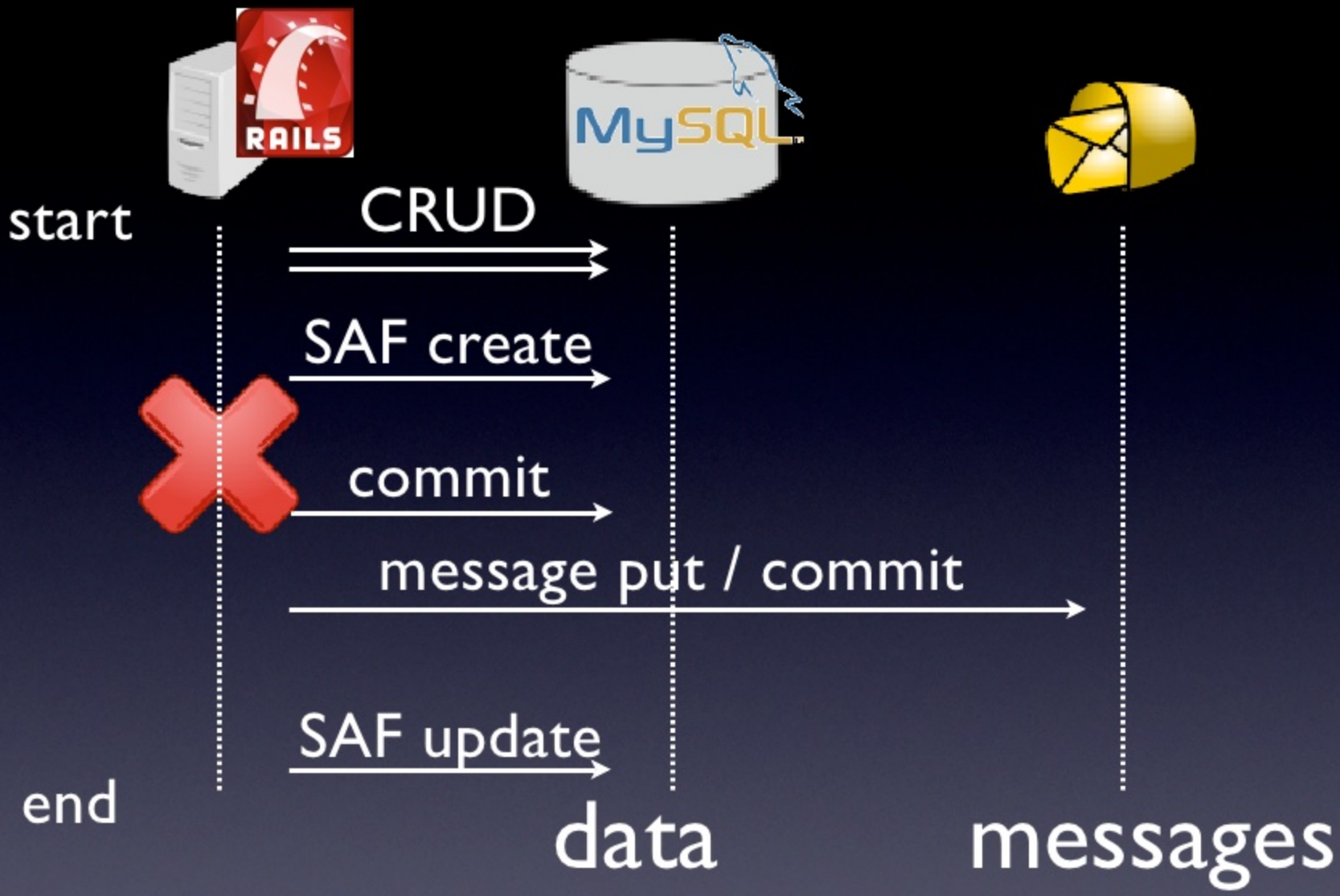
- Database trouble
- Network trouble
- Server down
- busy and timeout
- ... etc

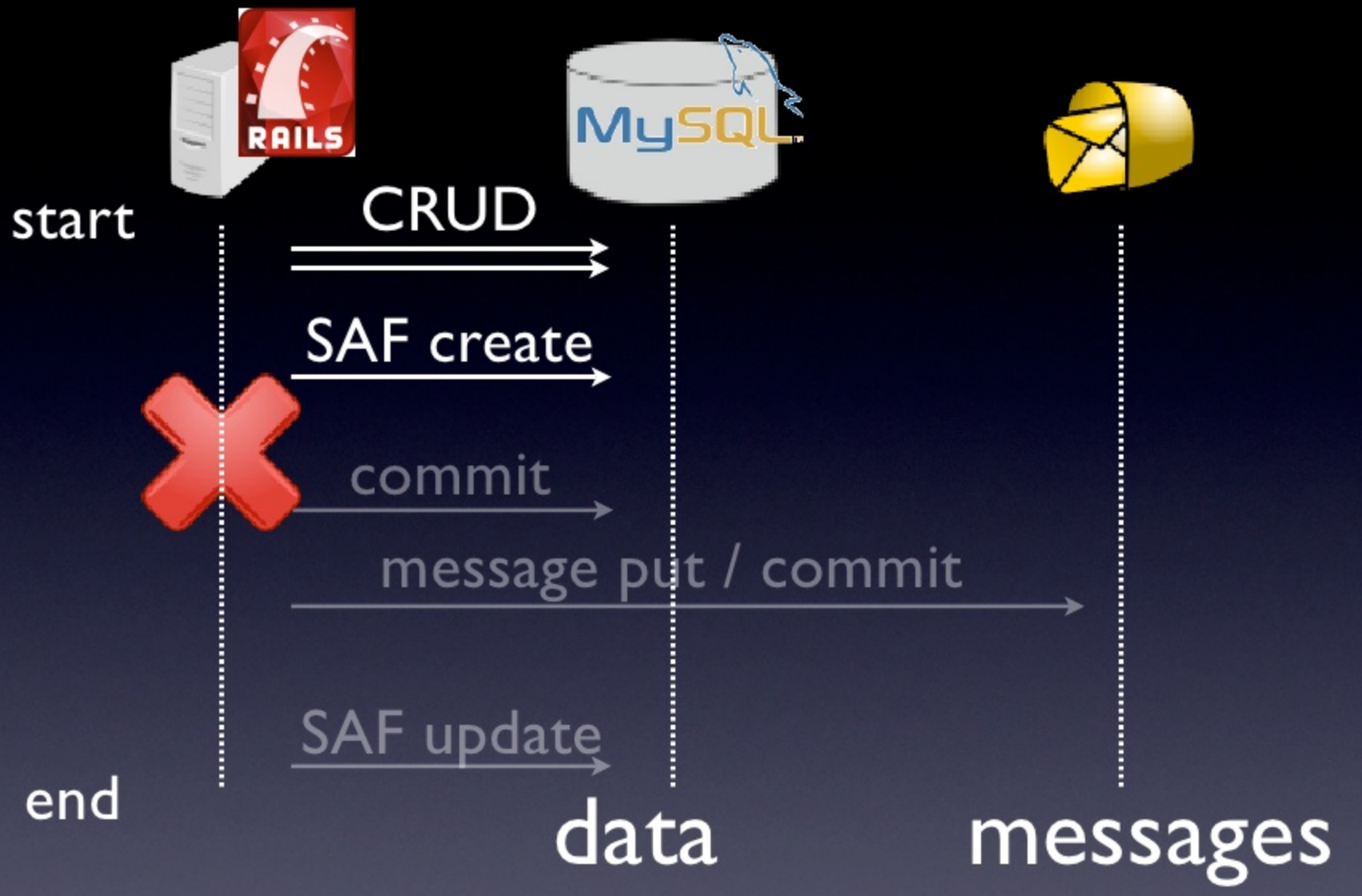
Embrace every exceptions

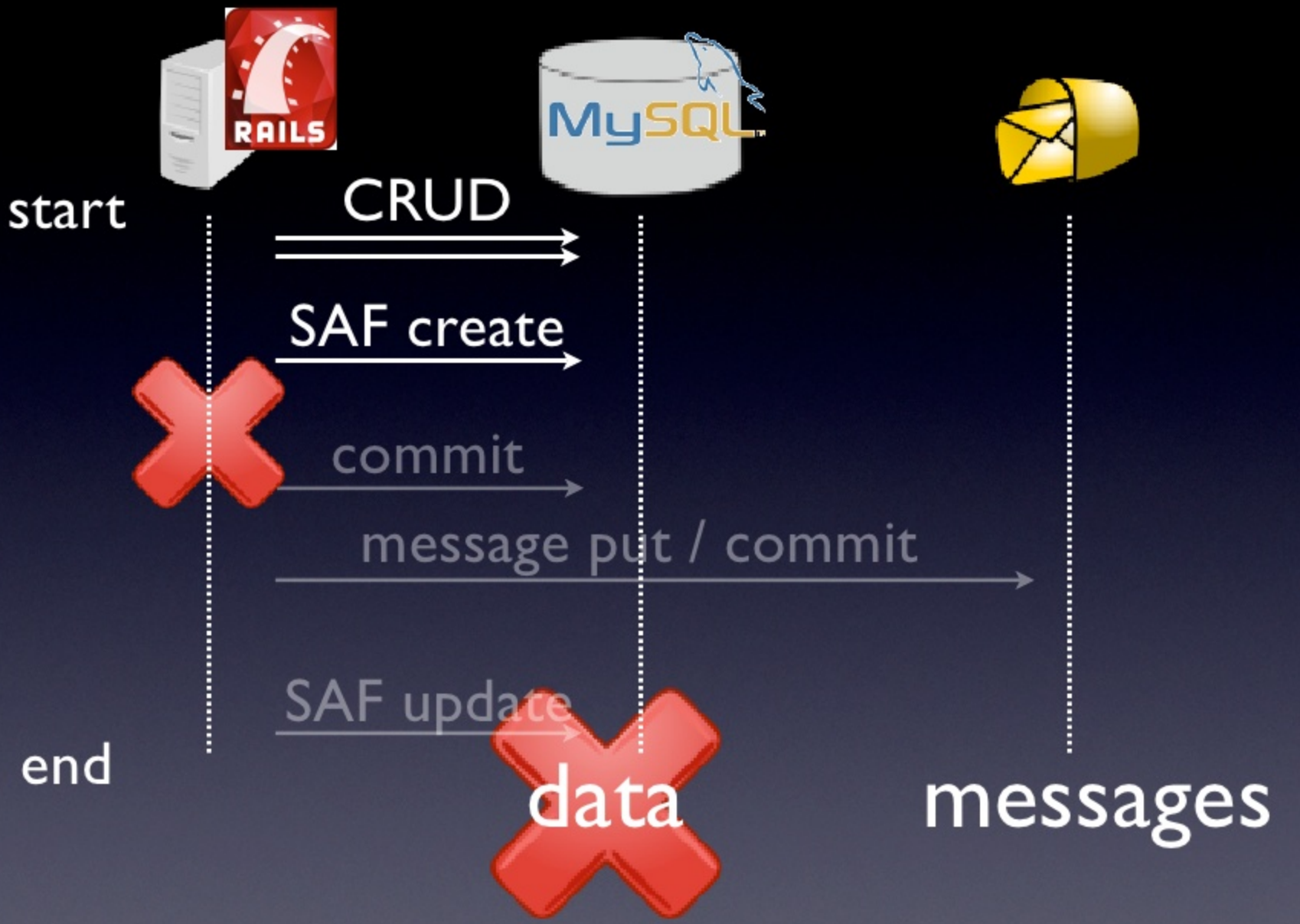
Sequence in detail
With SAF

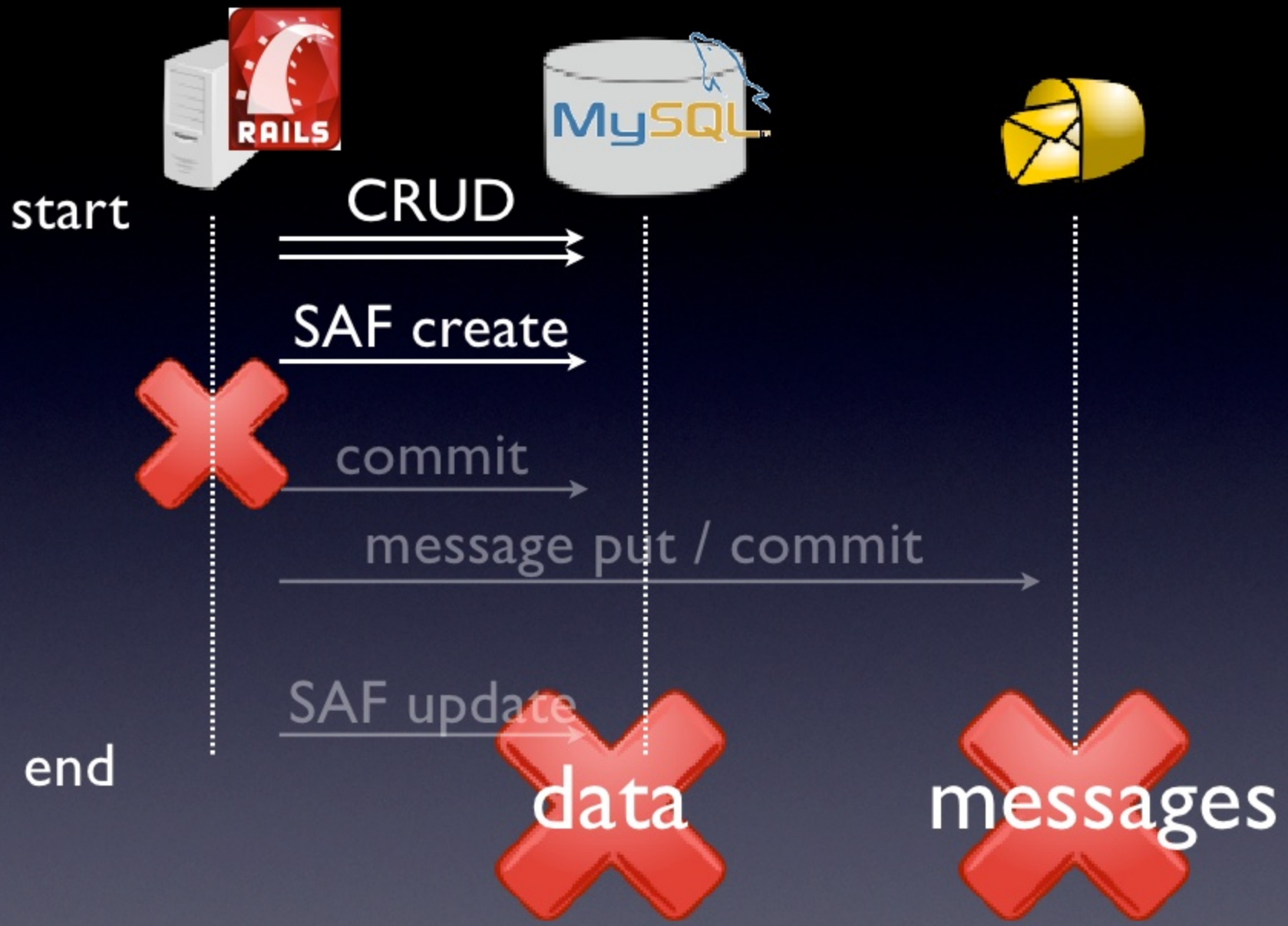

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

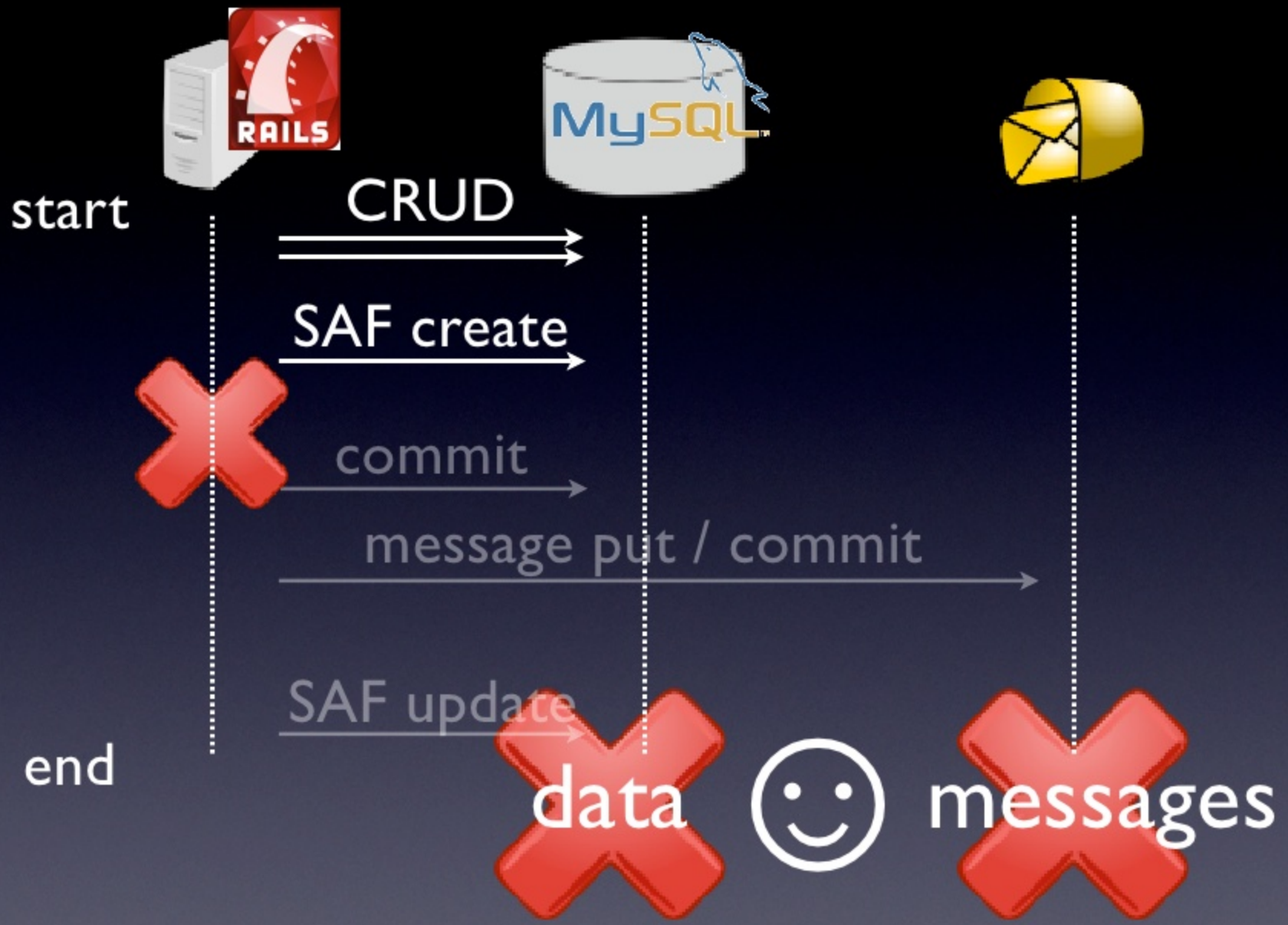

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

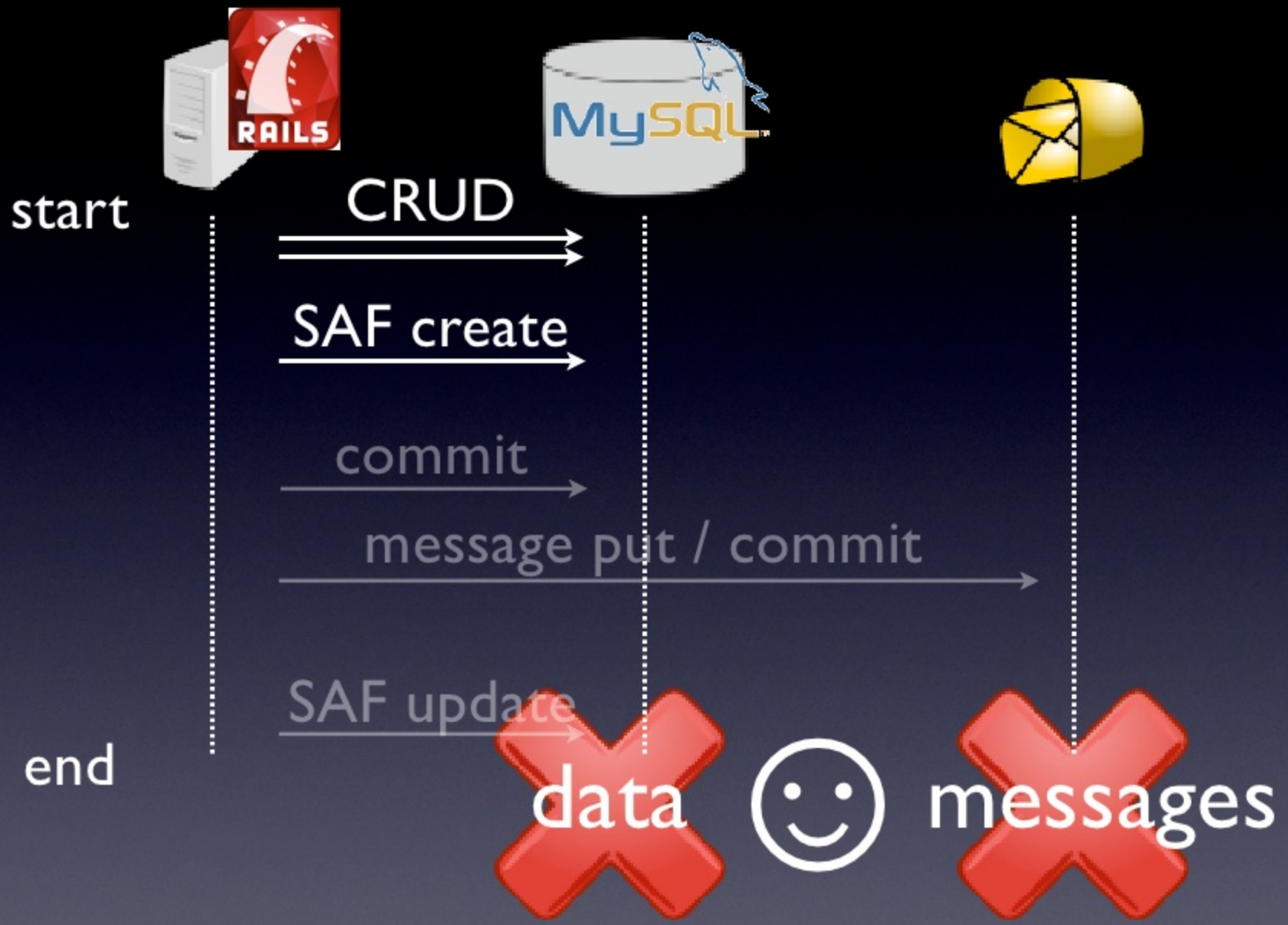

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

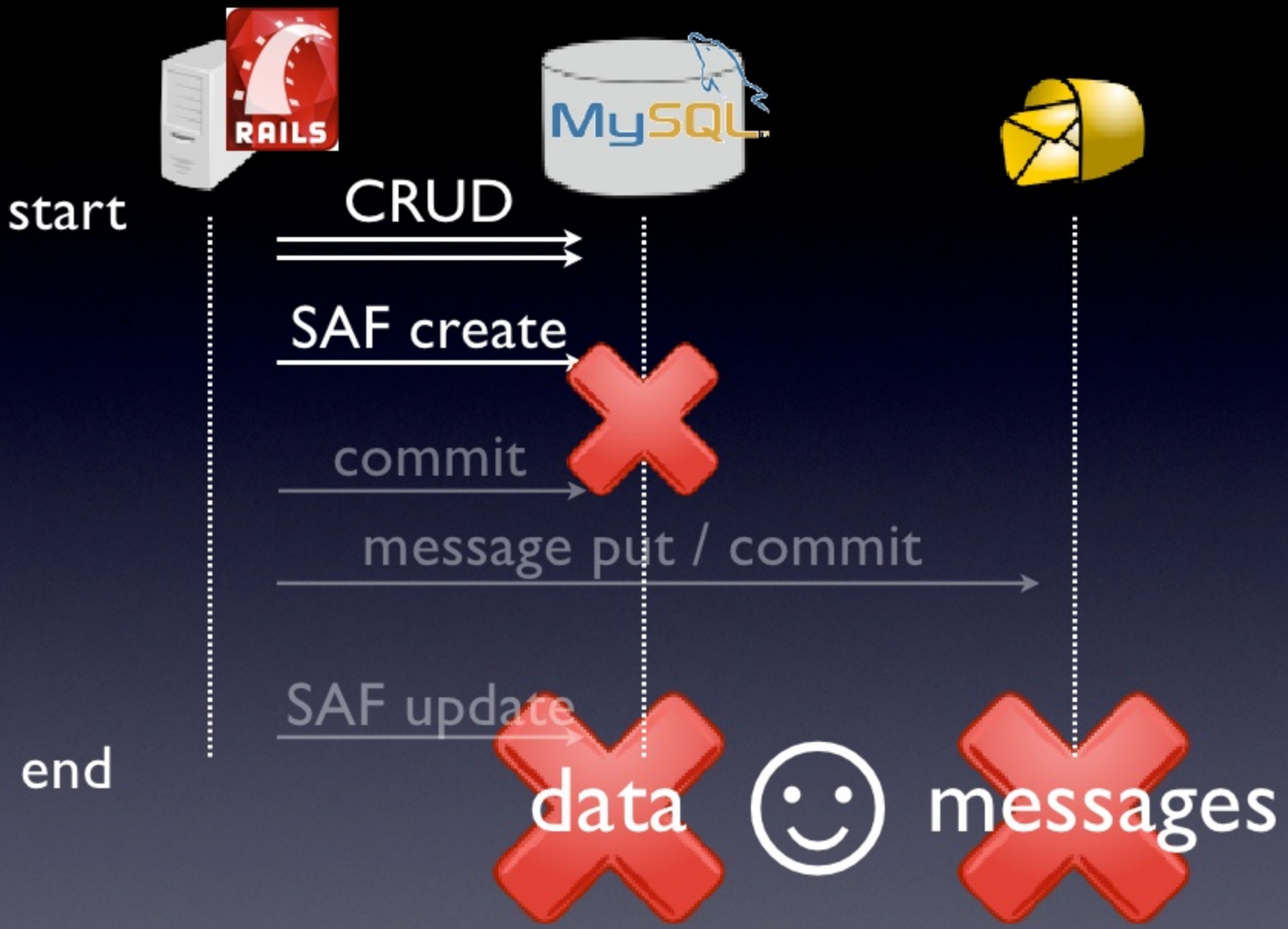

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

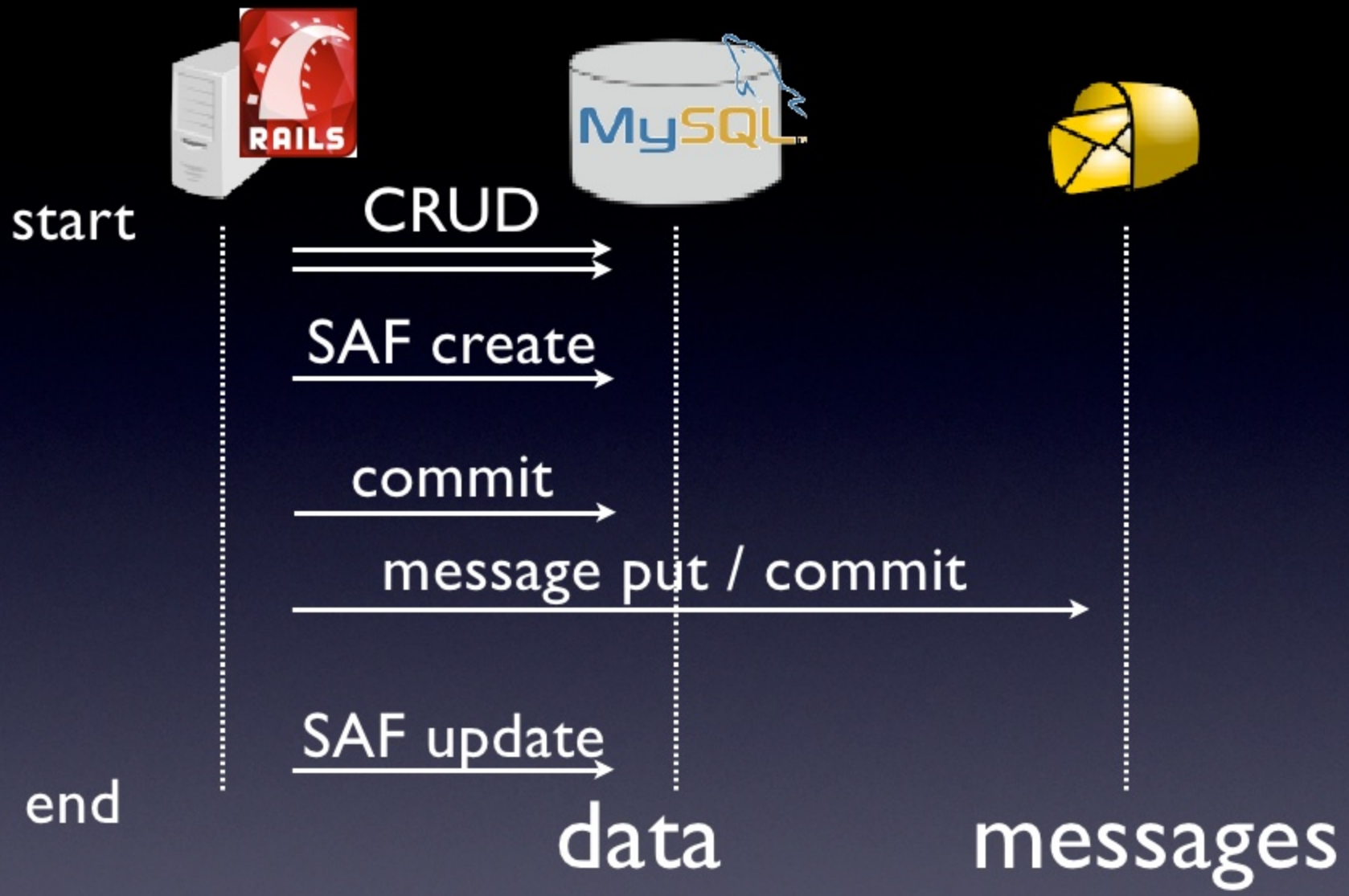

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

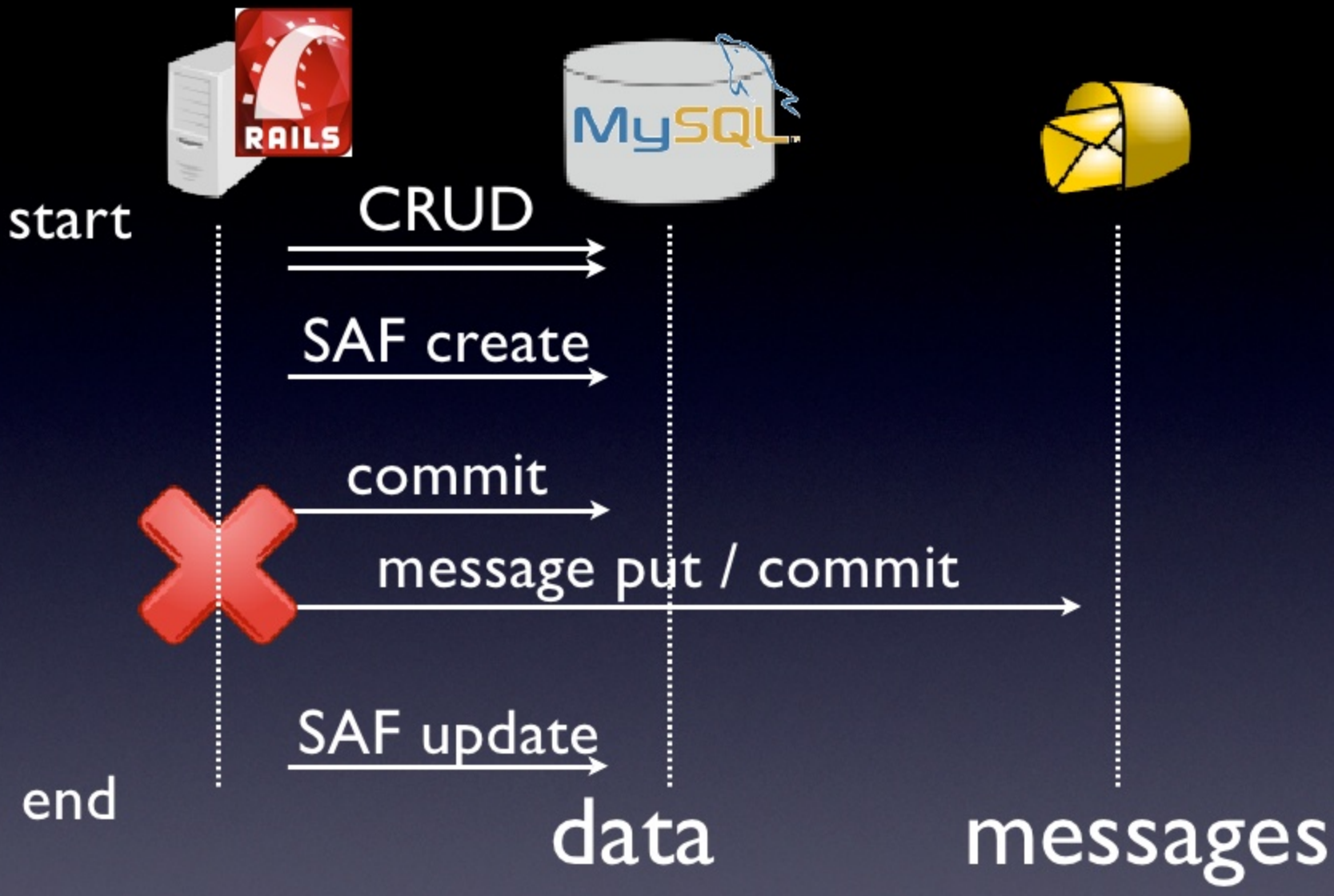

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

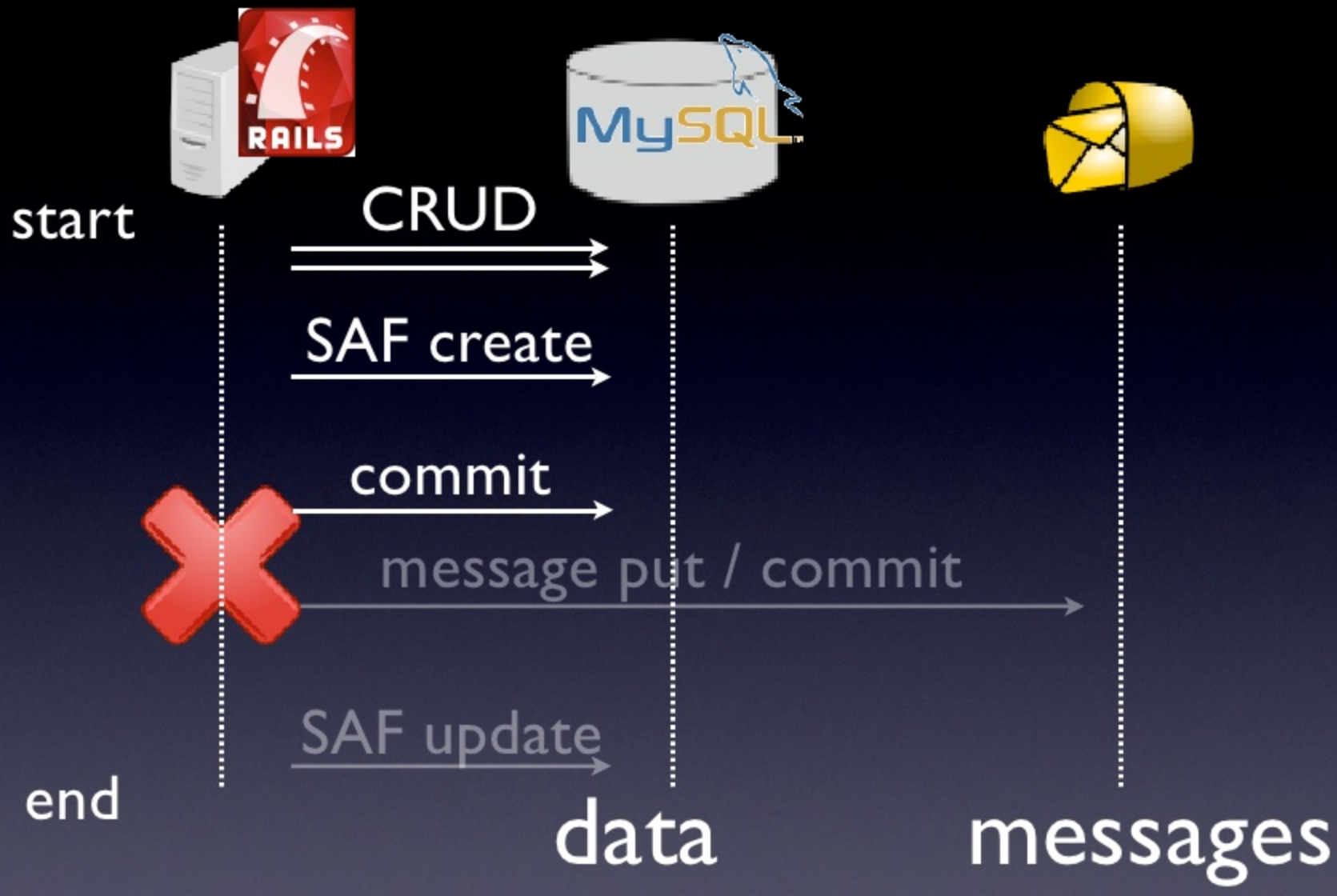

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

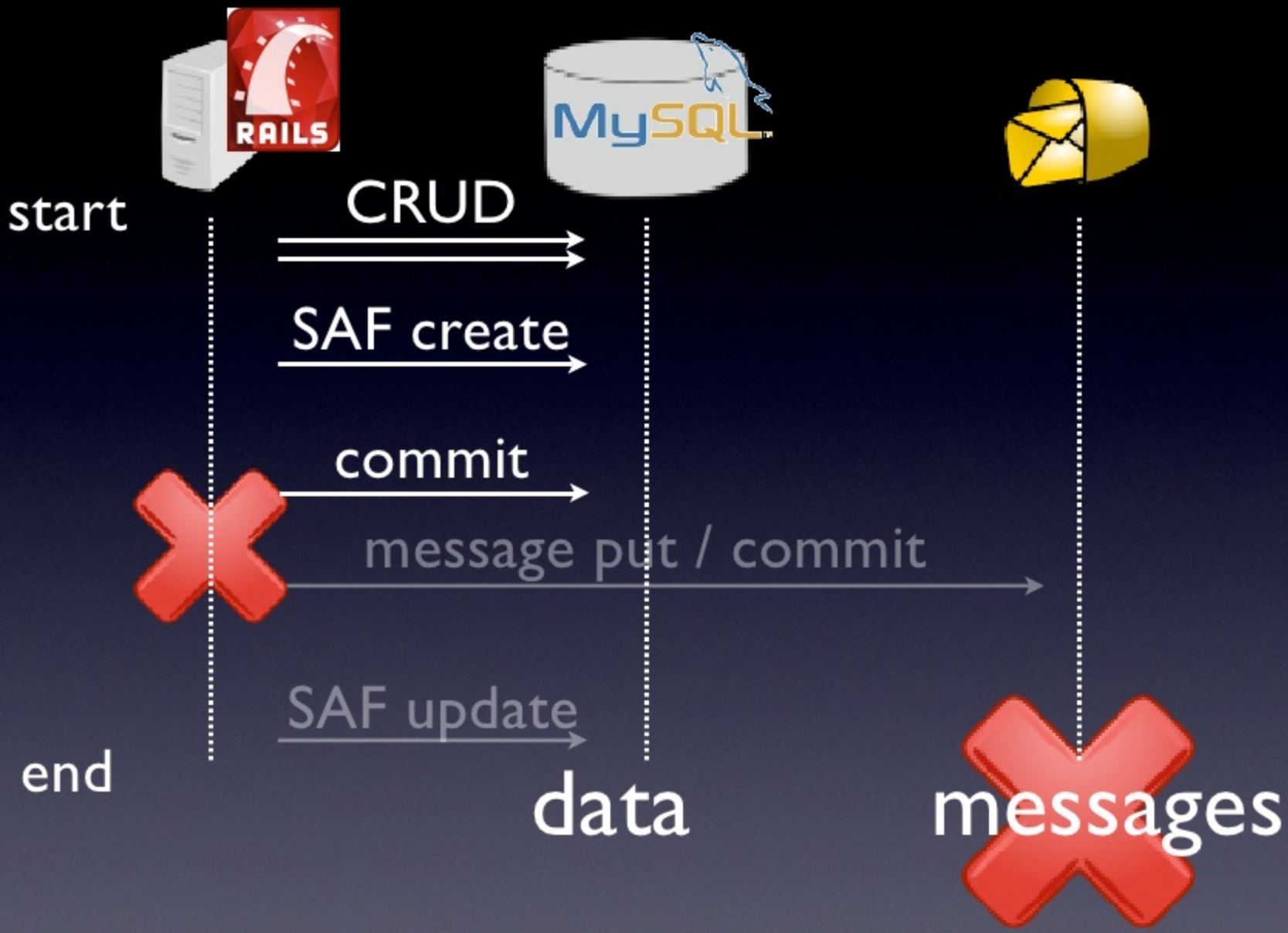

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

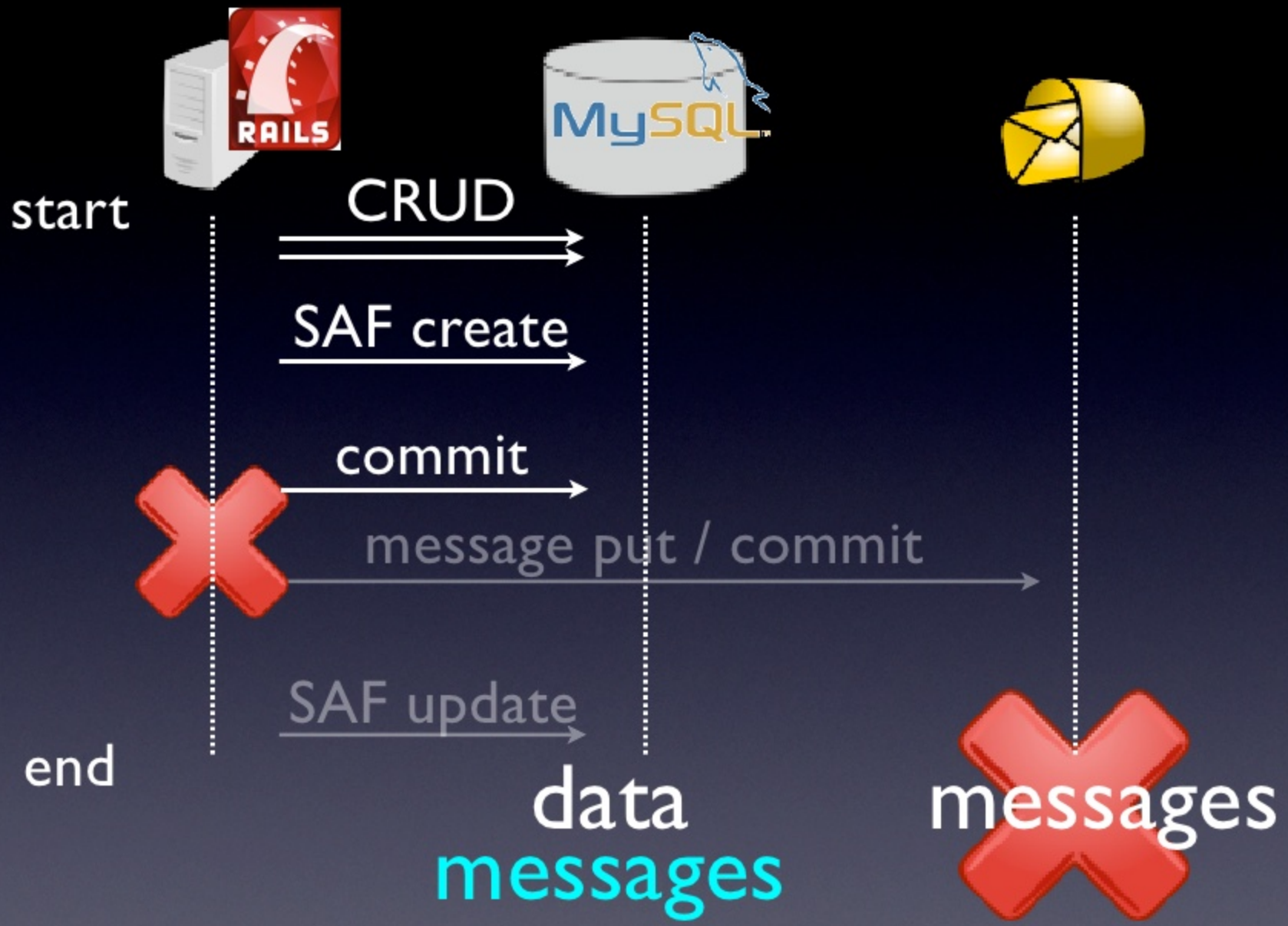

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

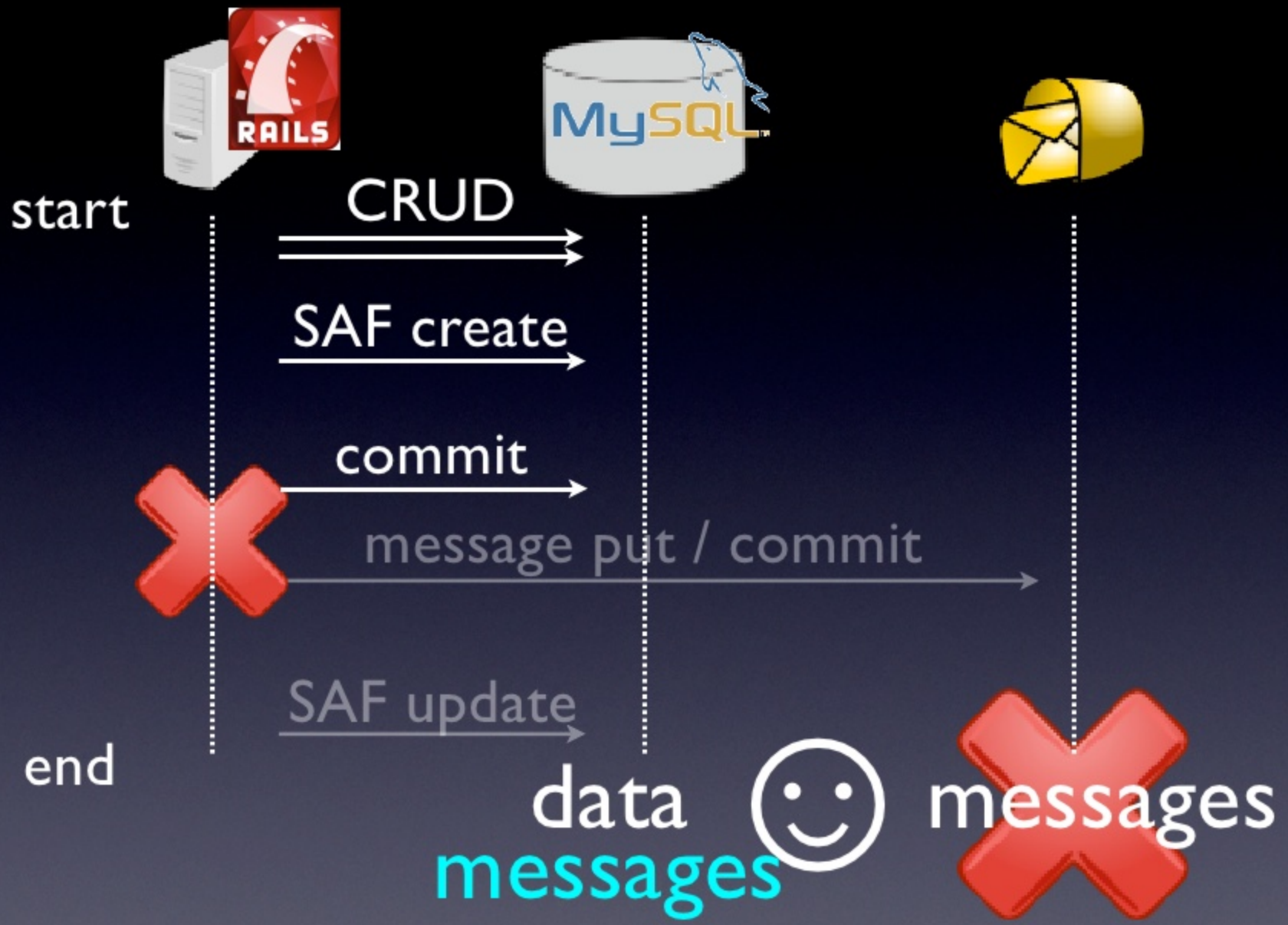

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages


RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

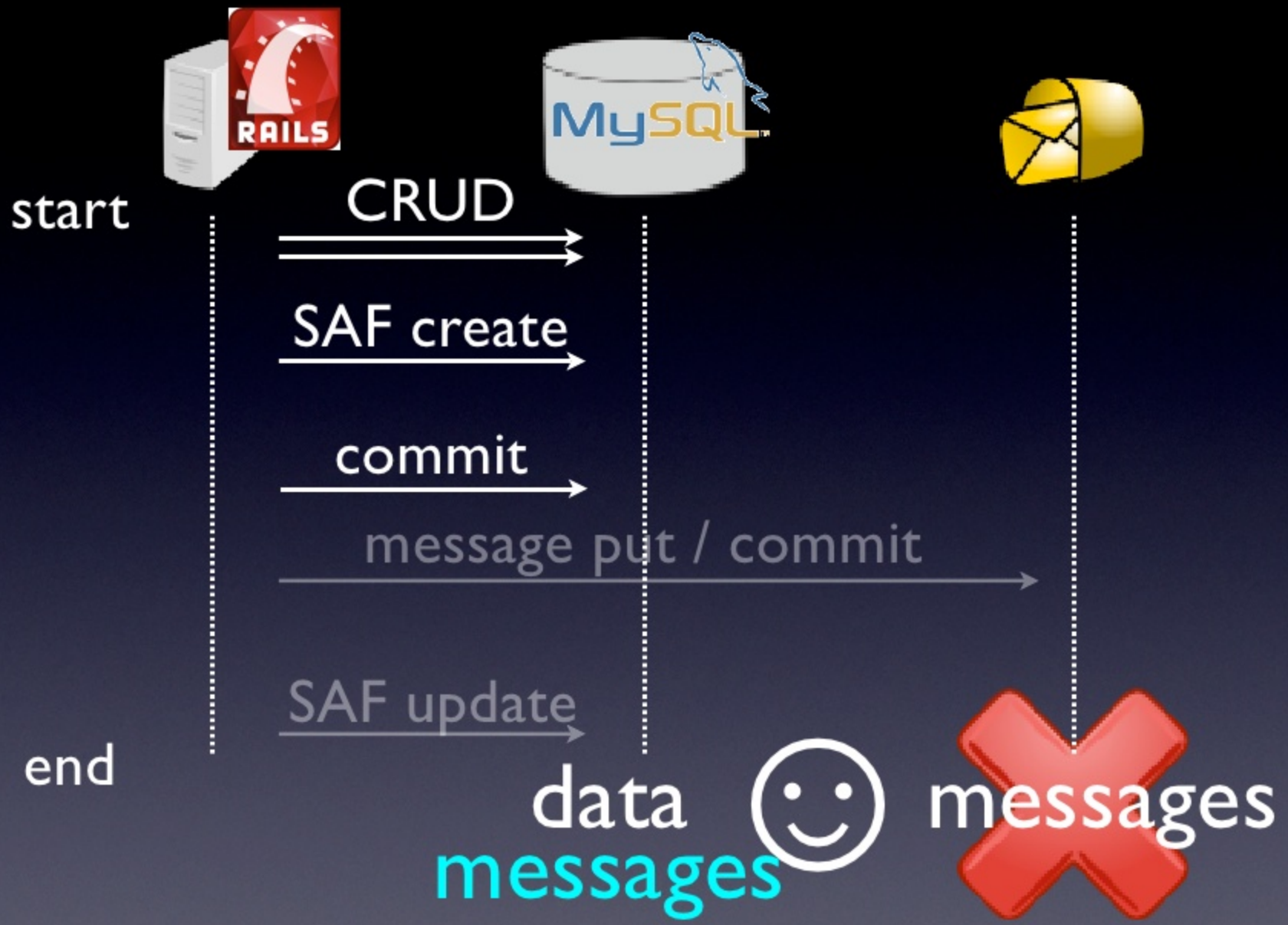

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

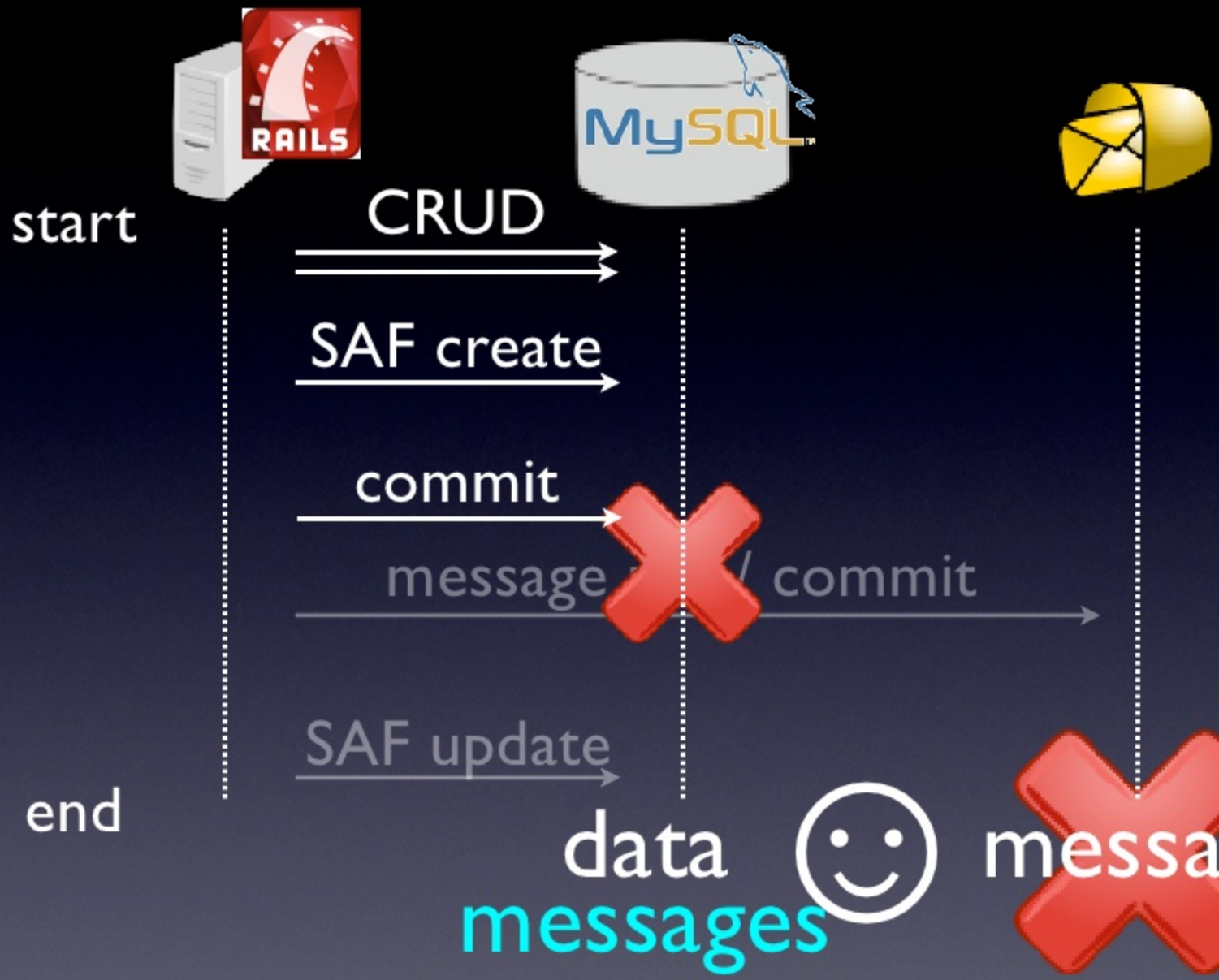

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

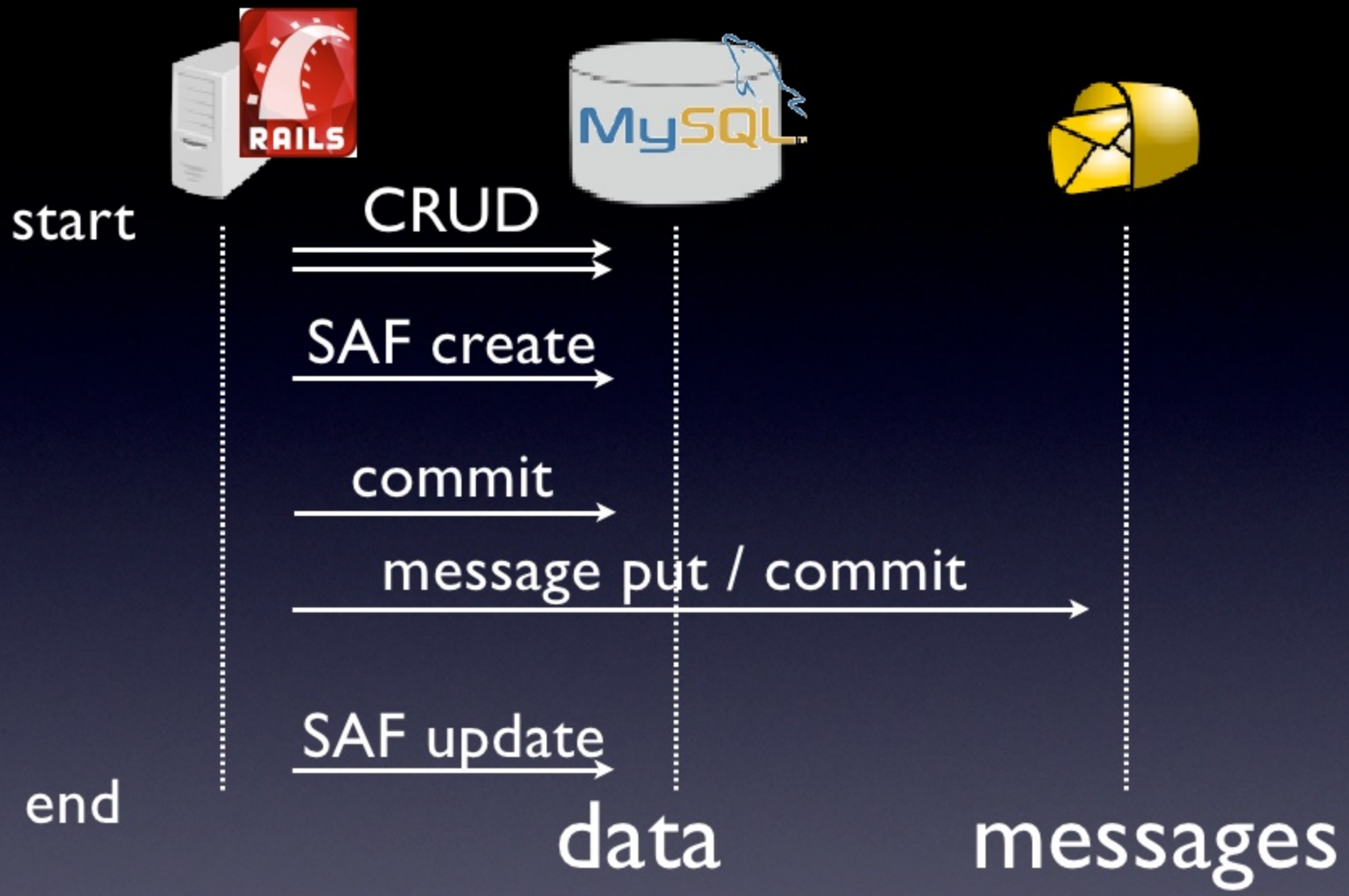

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages
messages

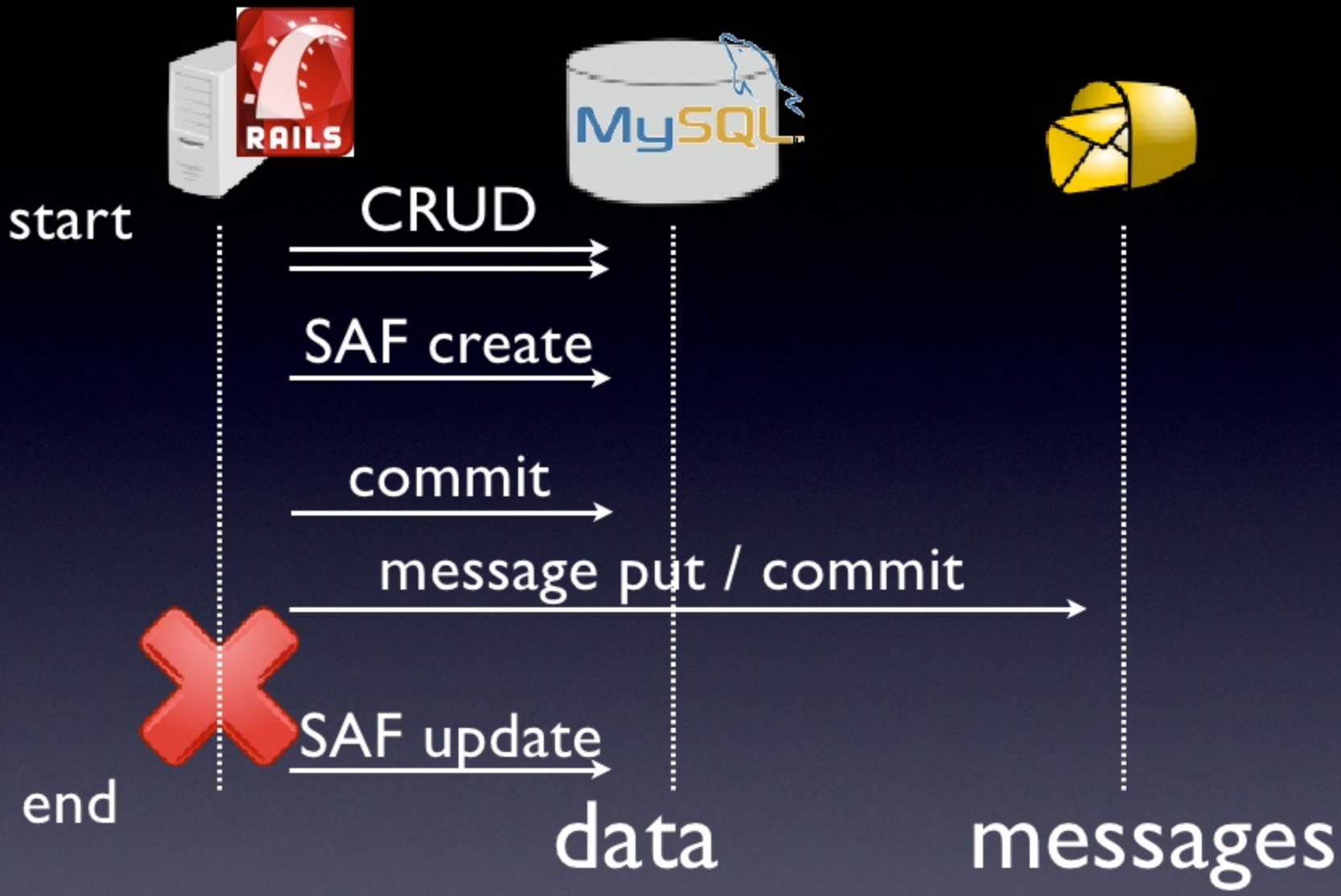

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages
messages

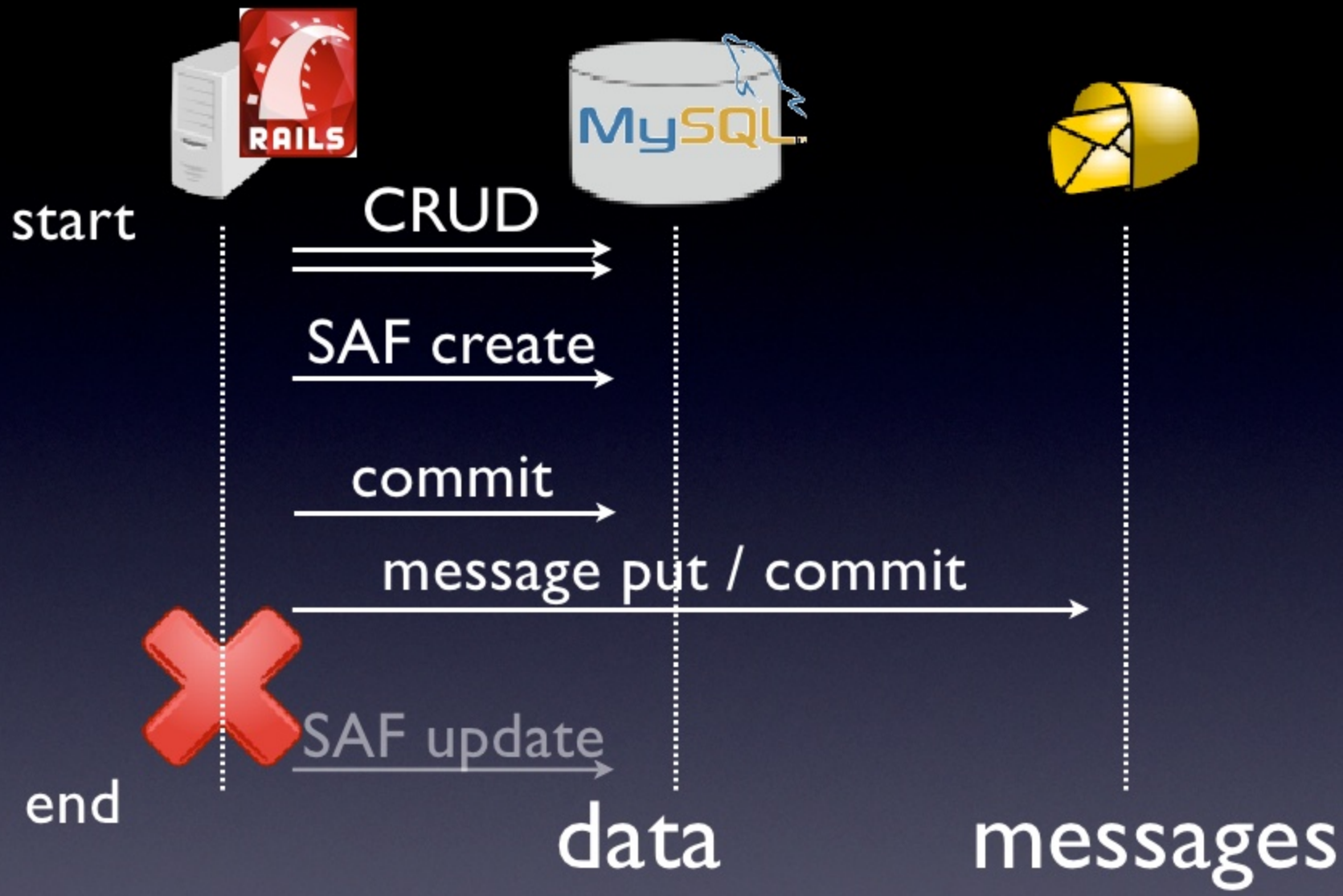

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
Recoverable
SAF update
end
data
messages
messages

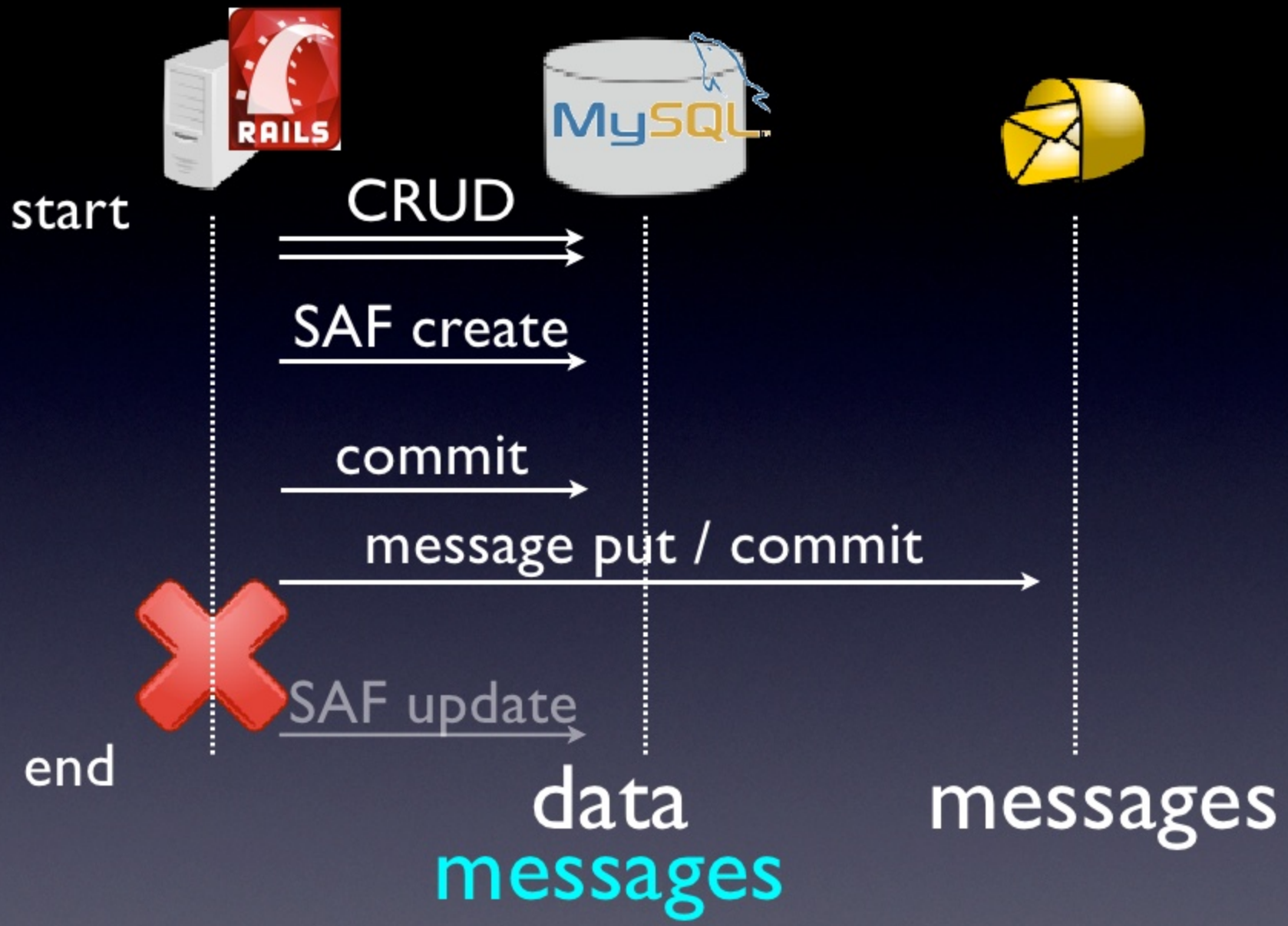

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages
messages

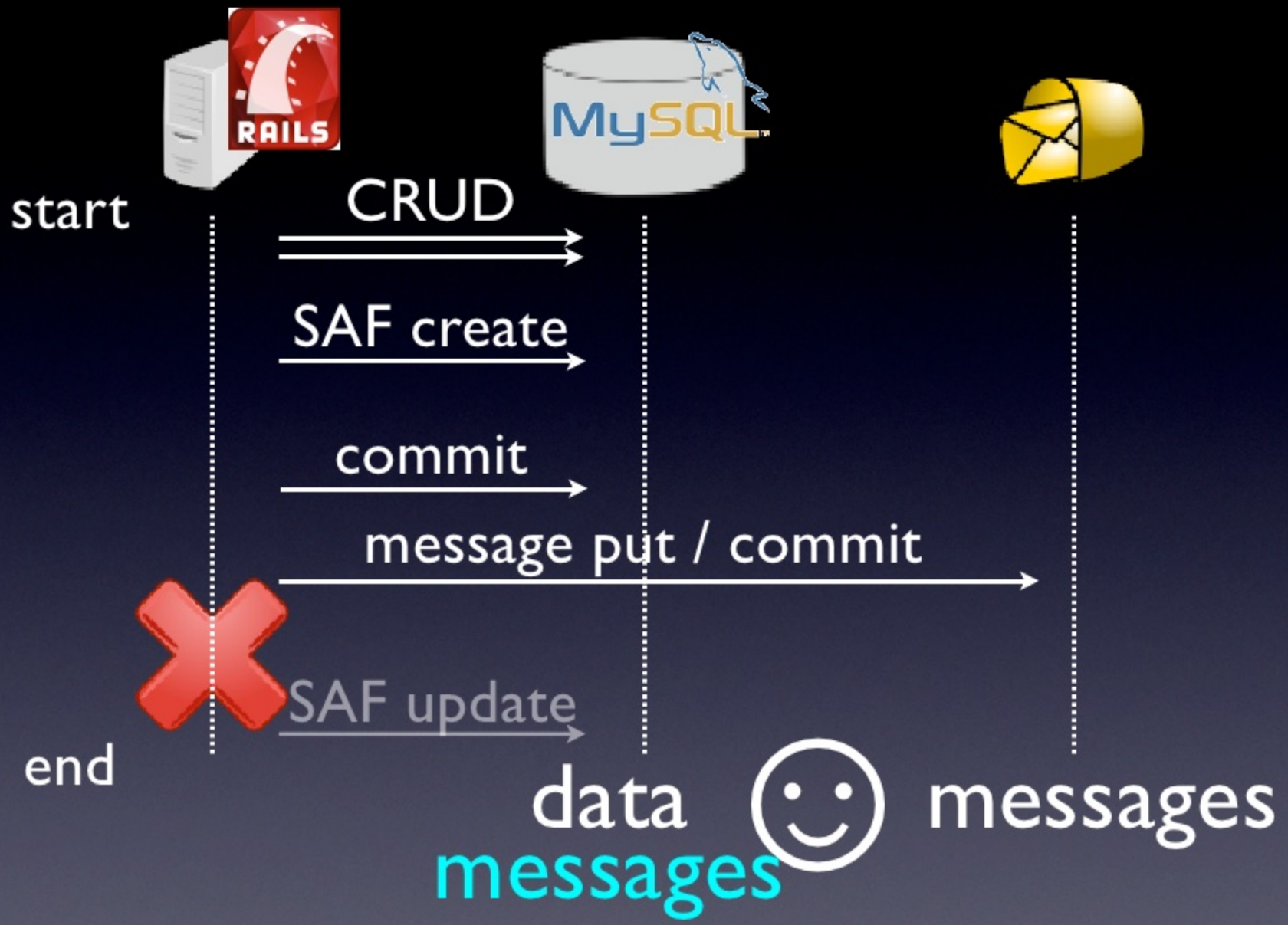

RAILS
MySQL
start
CRUD
SAF create
commit
commit
SAF update
end
data
messages
messages

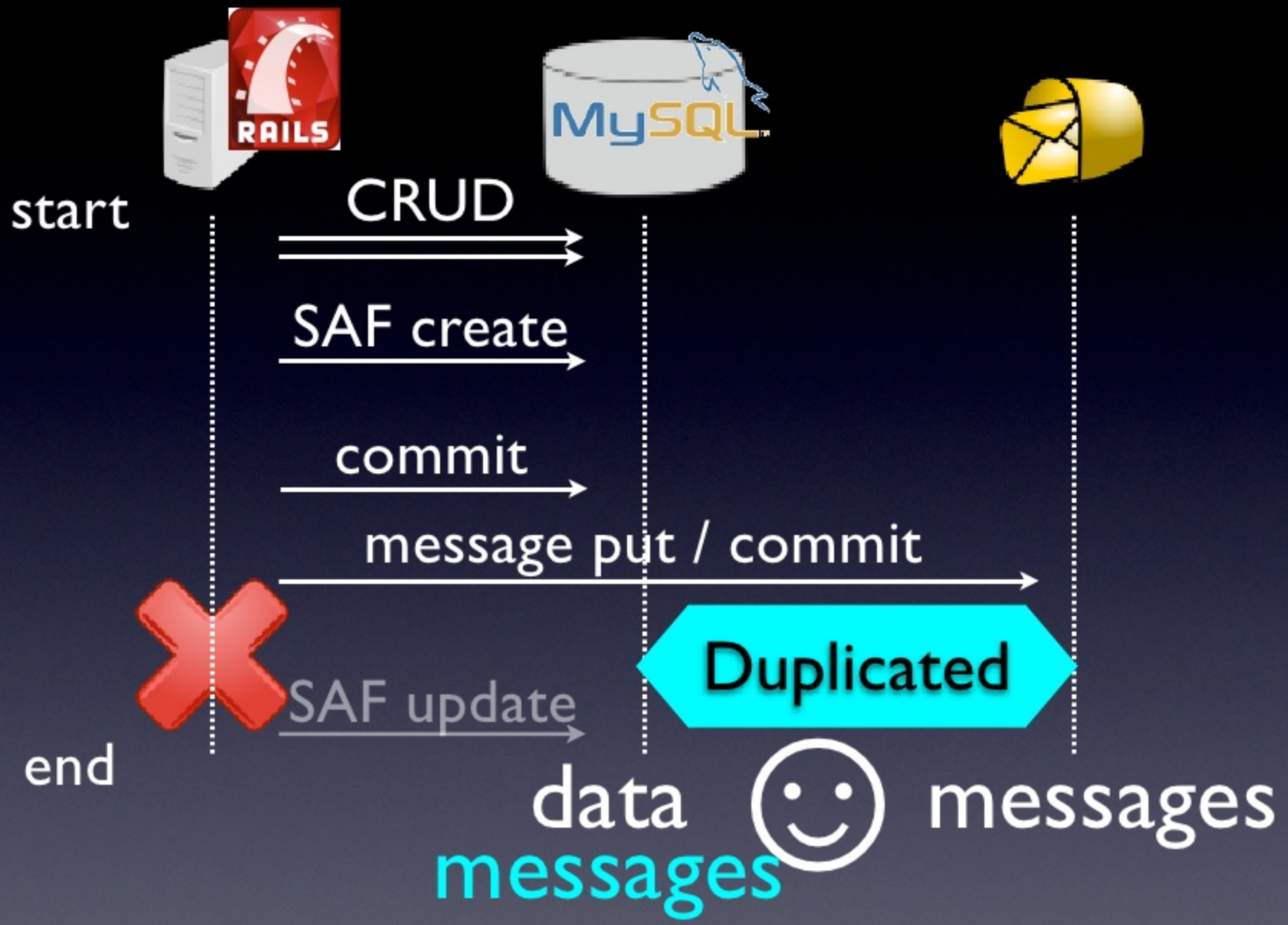

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

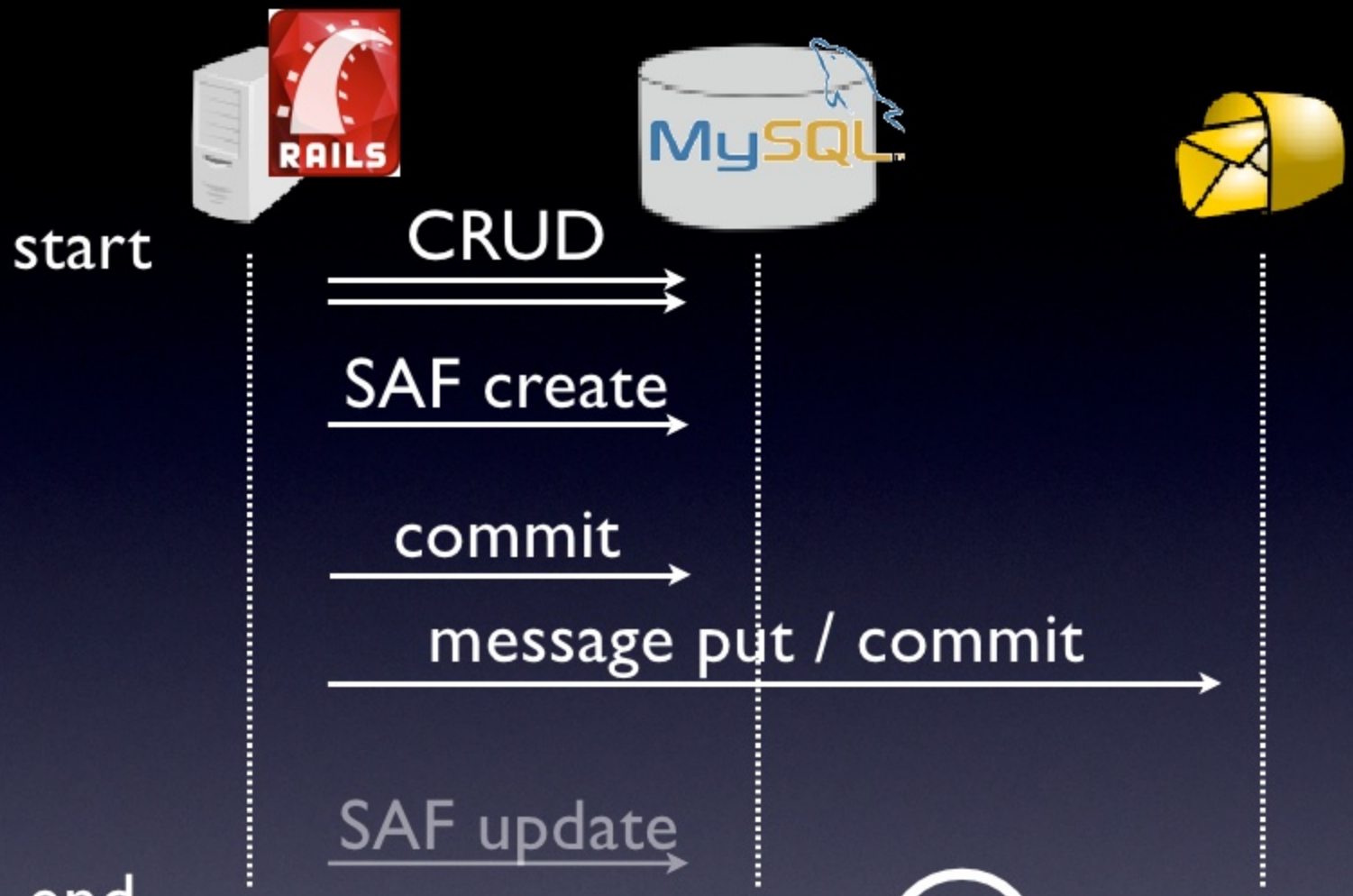

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages

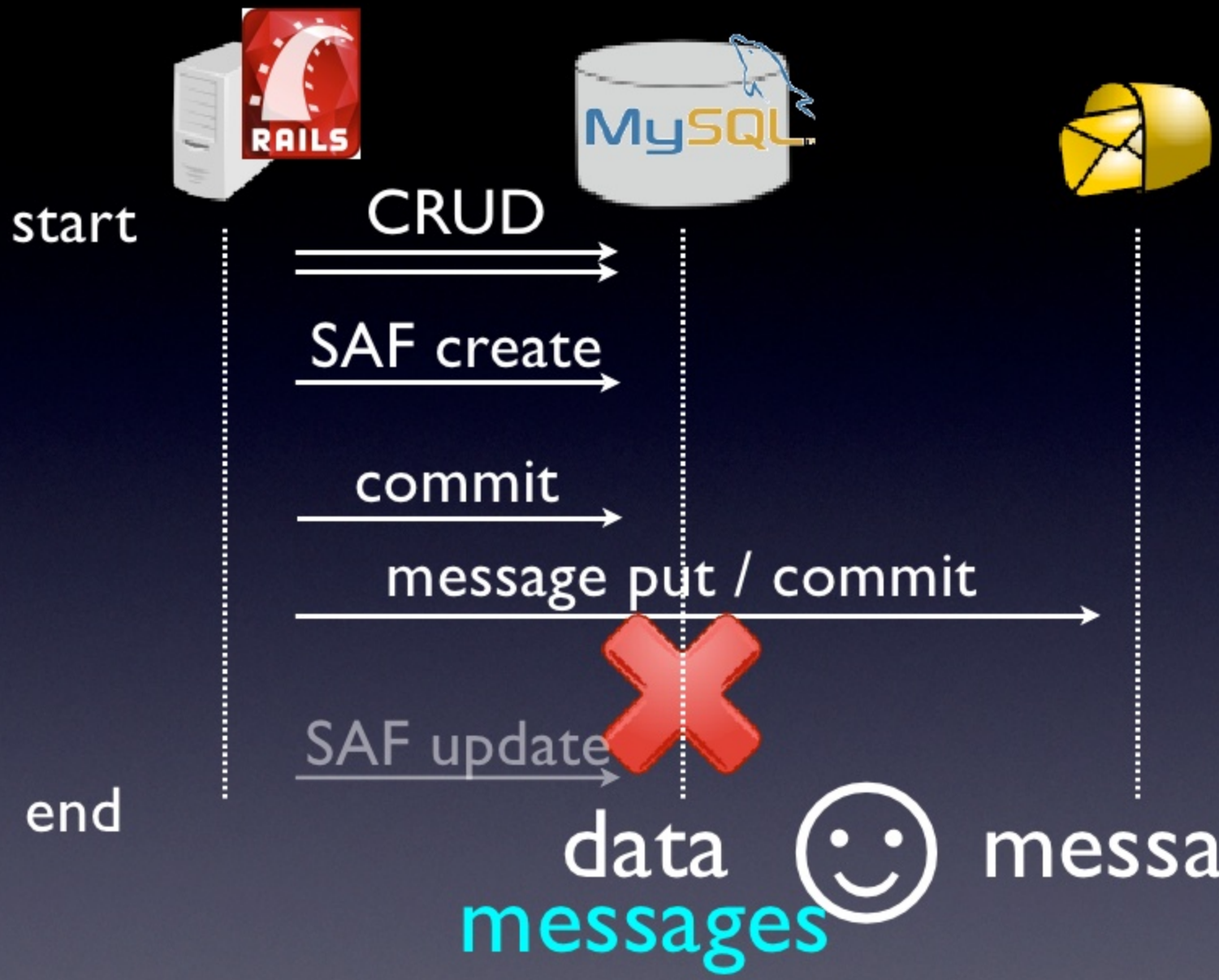

RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages


RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages
messages


RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages
messages


RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
Duplicated
SAF update
end
data
messages
messages


RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages
messages


RAILS
MySQL
start
CRUD
SAF create
commit
message put / commit
SAF update
end
data
messages
messages

Ensure (some)

Atomicness

of data and messages

at-least-once

between Producer and Channel

Summary

- Messaging with simple API, simple configuration.
- Seamless collaboration with Rails

- Automatic invocation of async proc.
- Guaranteed delivery by SAF

Lightweight
and
Robust

Future Plans

Smooth operation ver.0.3.x

- ☑ Daemonize
- ☑ URL-rewrite filter
- ☐ DLQ / SAF recovery
- ☐ Protocols: Stomp, HTTP

Monitoring ver.0.4.x

- Coordinate with Cacti, Nagios, etc.
- Many AP4R processes
- Interactive management tool

Step to large scale ver.0.5.x

- Dynamic configuration
- Automatic recovery
- Long running

Others

- Security
- Application testing support
- Blocking queues

AP4R@RubyForge

- wiki: http://ap4r.rubyforge.org/wiki/wiki.pl?HomePage
- source: svn co svn://rubyforge.org/var/svn/ap4r

Your patches are welcome! Thank you.

- Future Architect logo belong to Future Architect, Inc. Japan. All rights reserved.
- Rails logo are trademarks of David Heinemeier Hansson. All rights reserved.
- The Ruby logo is copyright c 2006, Yukihiro Matsumoto. It is released under the terms of the Creative Commons Attribution-ShareAlike 2.5 License.