

# SERVICE-ORIENTED ARCHITECTURE

a practical introduction to the  
service-oriented mind set

Thorben Schröder  
FH Salzburg 2014

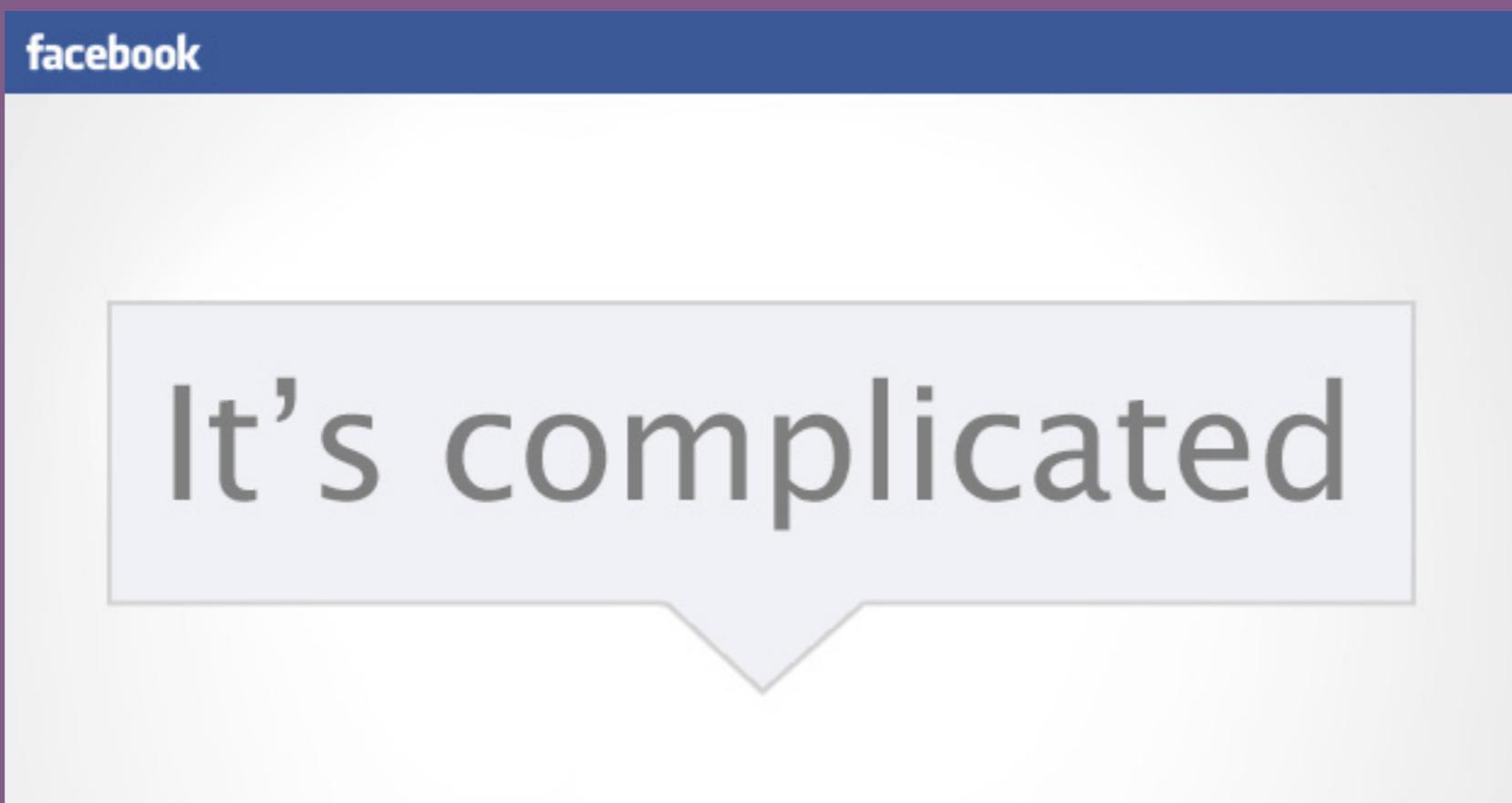
# HI THERE

I am Thorben.

Reach me at [info@thorbenschroeder.de](mailto:info@thorbenschroeder.de)

Or find me on Twitter, GitHub & Co as @walski.

# WHAT I DO



# WHAT I DO



**WHAT DO YOU DO?**

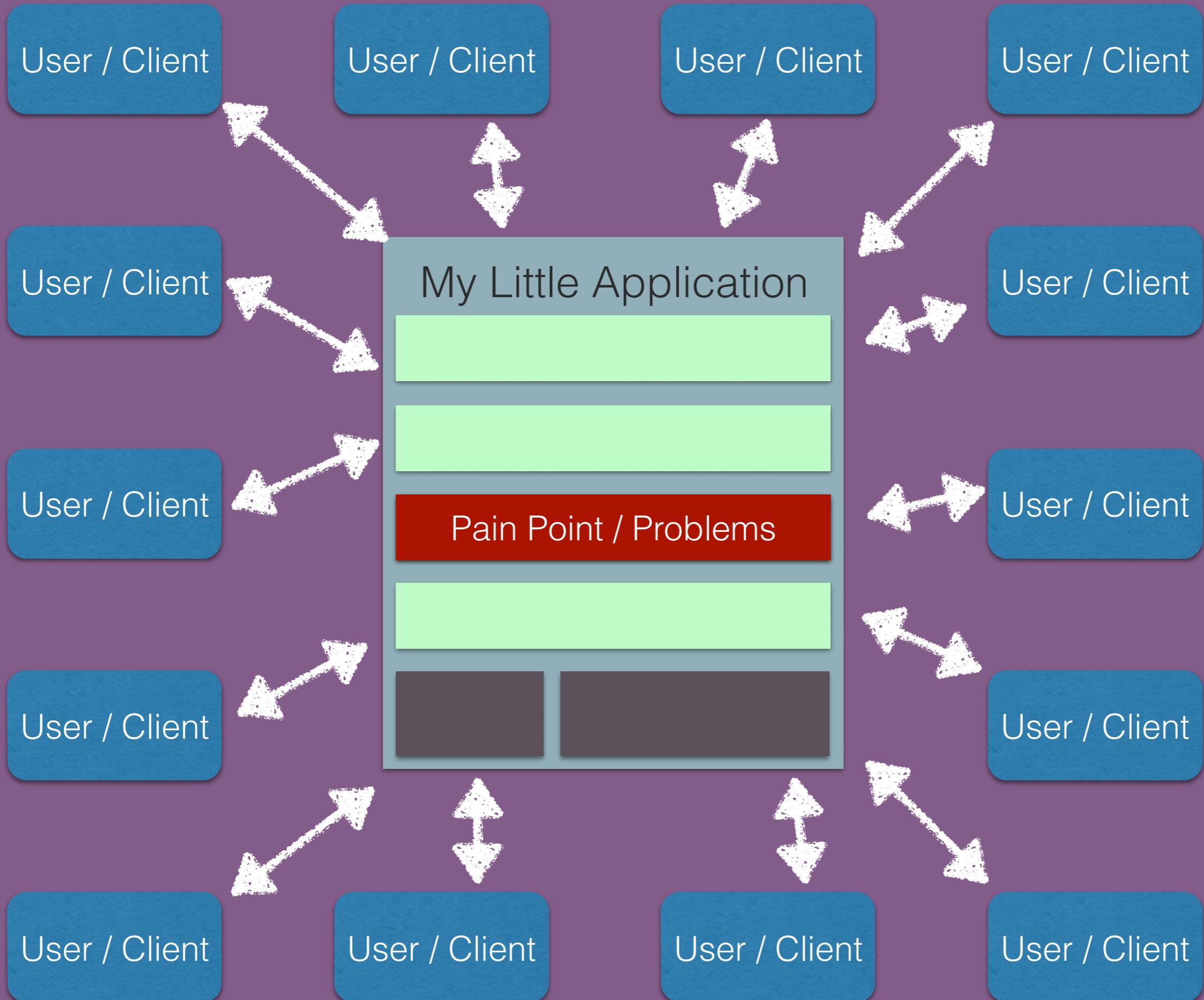
# GOOD TO KNOW

- **Basics**
  - **git, (a little) UNIX, vocabulary (deployments, ...)**
- **Ruby ( $\geq 2.0$ )**
- **Ruby ecosystem**
  - **RubyGems, bundler, RVM**
- **Test driven methodology**

# AGENDA

- 1. Intro to SOA**
- 2. Modular Web Applications**
- 3. Documentation**
- 4. Testing**
- 5. Development Setups**
- 6. Service Authentication**
- 7. Inter Service Communication**
- 8. Deployments**
- 9. Browsers as Clients**

# SERVICES!



My Little This

Another Thing

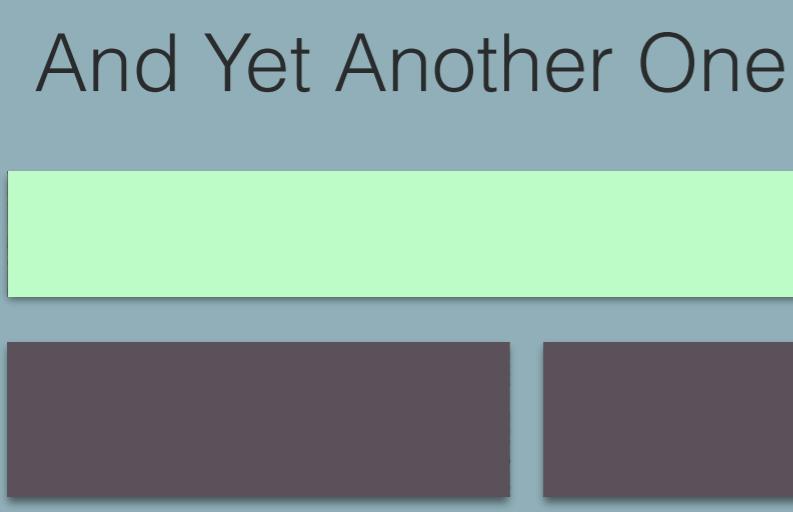
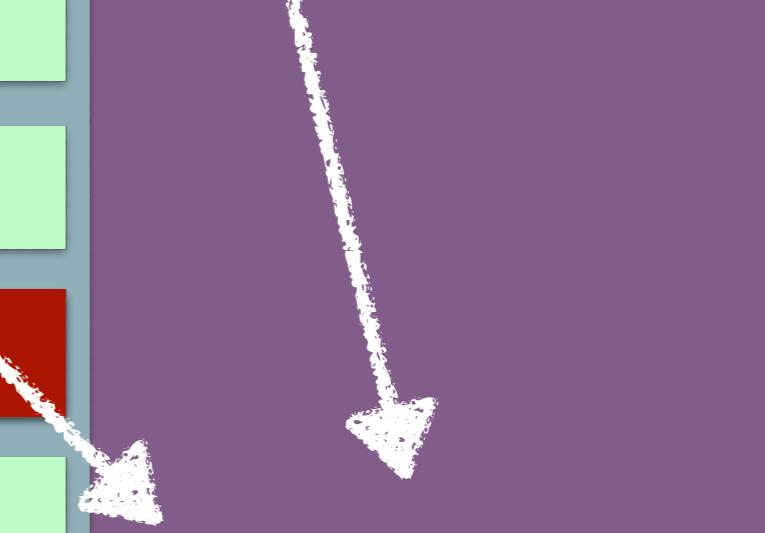
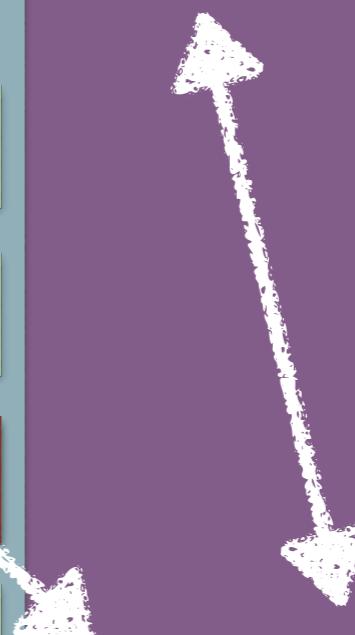
Pain Point / Problems

My Little Application

Pain Point / Problems

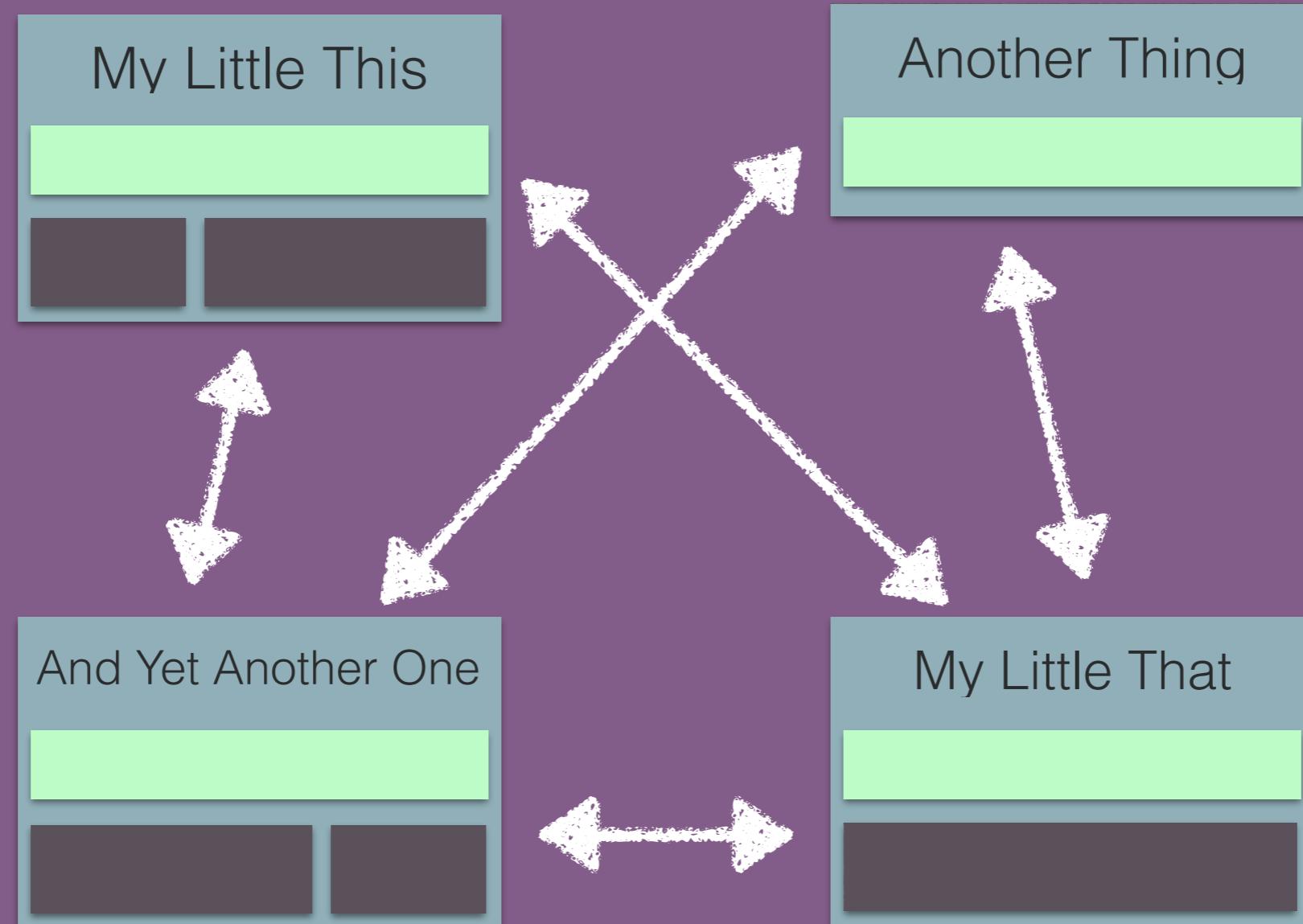
And Yet Another One

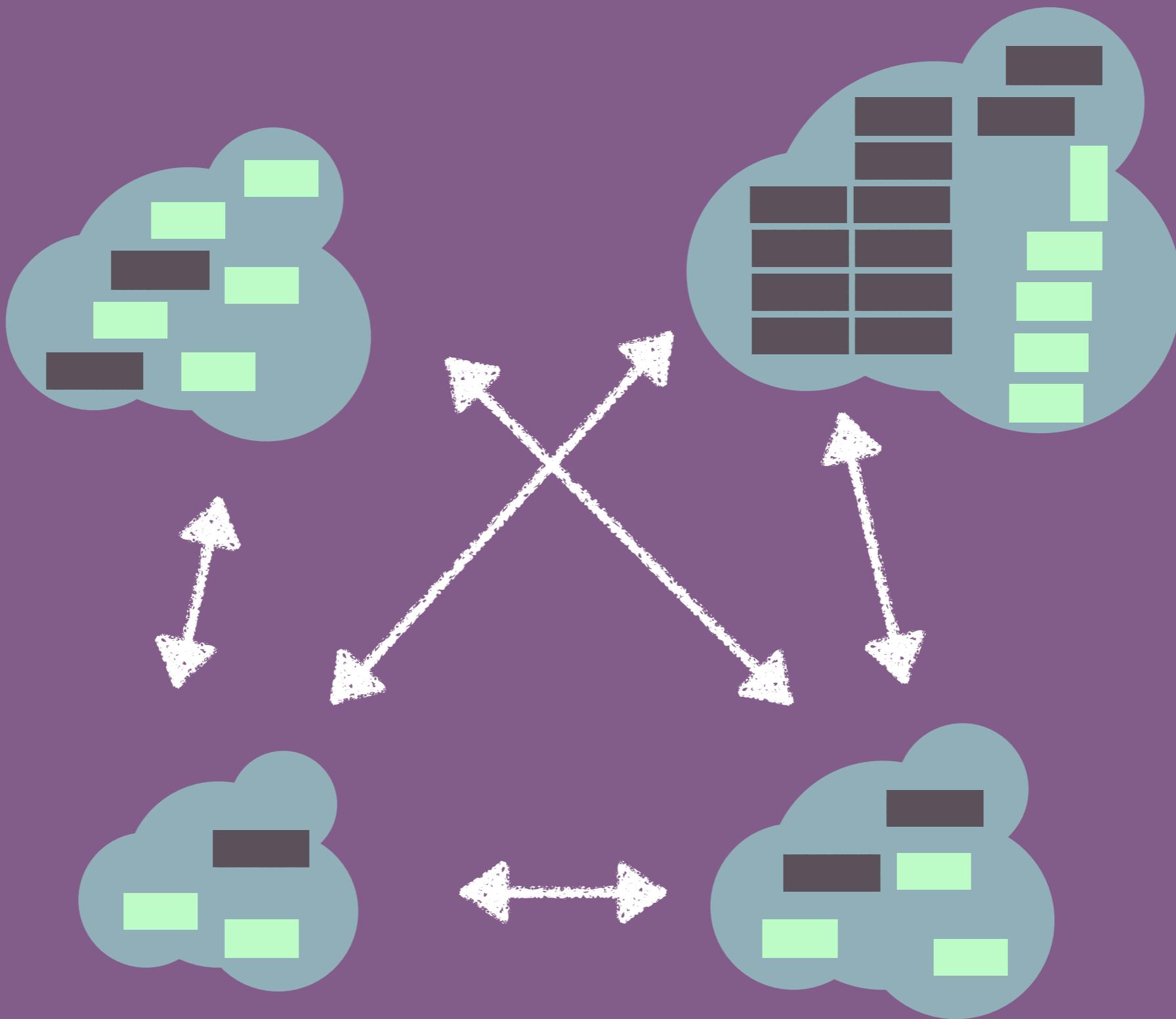
My Little That

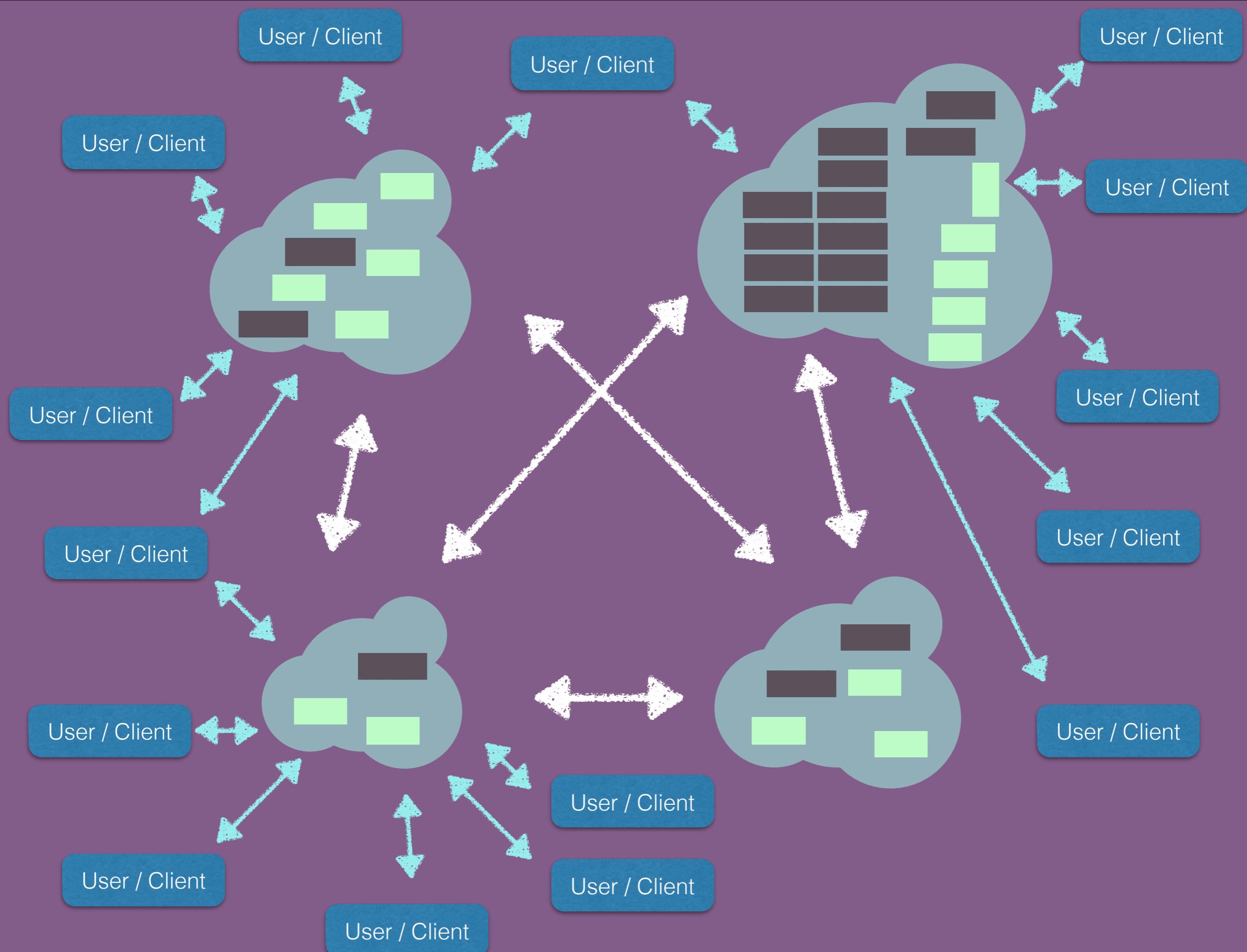


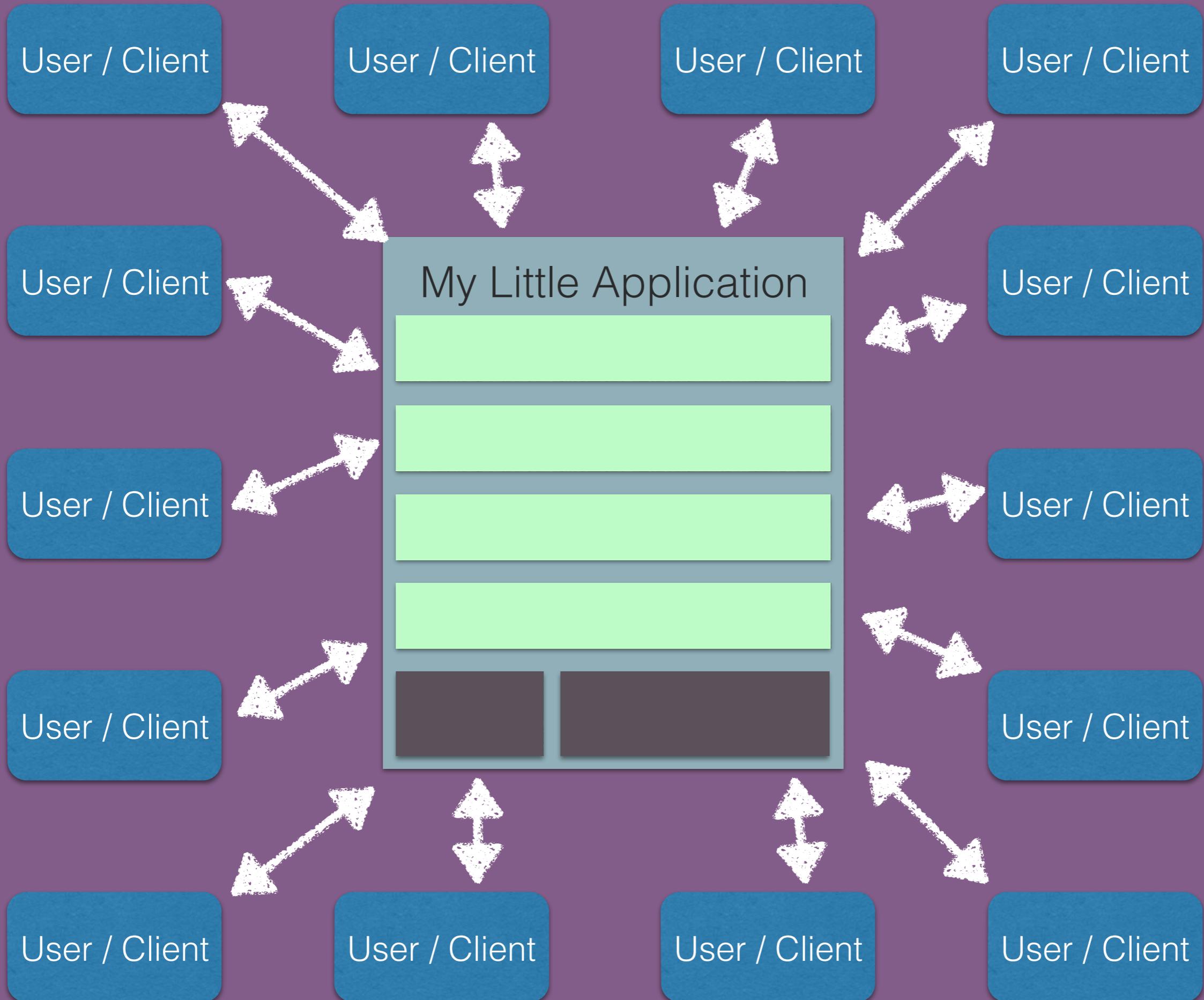
# WHY?

- Applications' requirements keep growing
- Big applications are hard to maintain & debug
- Splitting up helps to focus!
- Bonus points: An internal API can be used externally at very little extra cost









# SERVICE ORIENTED ARCHITECTURE IN BIG STROKES

- Move away from monolithic applications
- Change to clusters of communicating apps
- UNIX philosophy:  
“Make each program do one thing well.”

# HISTORY

- term coined by 1996
- got popular with the web as it's foundation by around 2001
- not very well defined

# FOUR TENETS OF SOA

Don Box of Microsoft:

1. **Boundaries are explicit**
2. **Services are autonomous**
3. **Services share schema and contract, not class**
4. **Service compatibility is based on policy**

**DOESN'T SAY MUCH...**

BUT

# My Little Application

Shop

Billing

Warehouse

Users

Redis

Database

My Billing

Billing

Database

My Shop

Shop

Warehouse

Redis

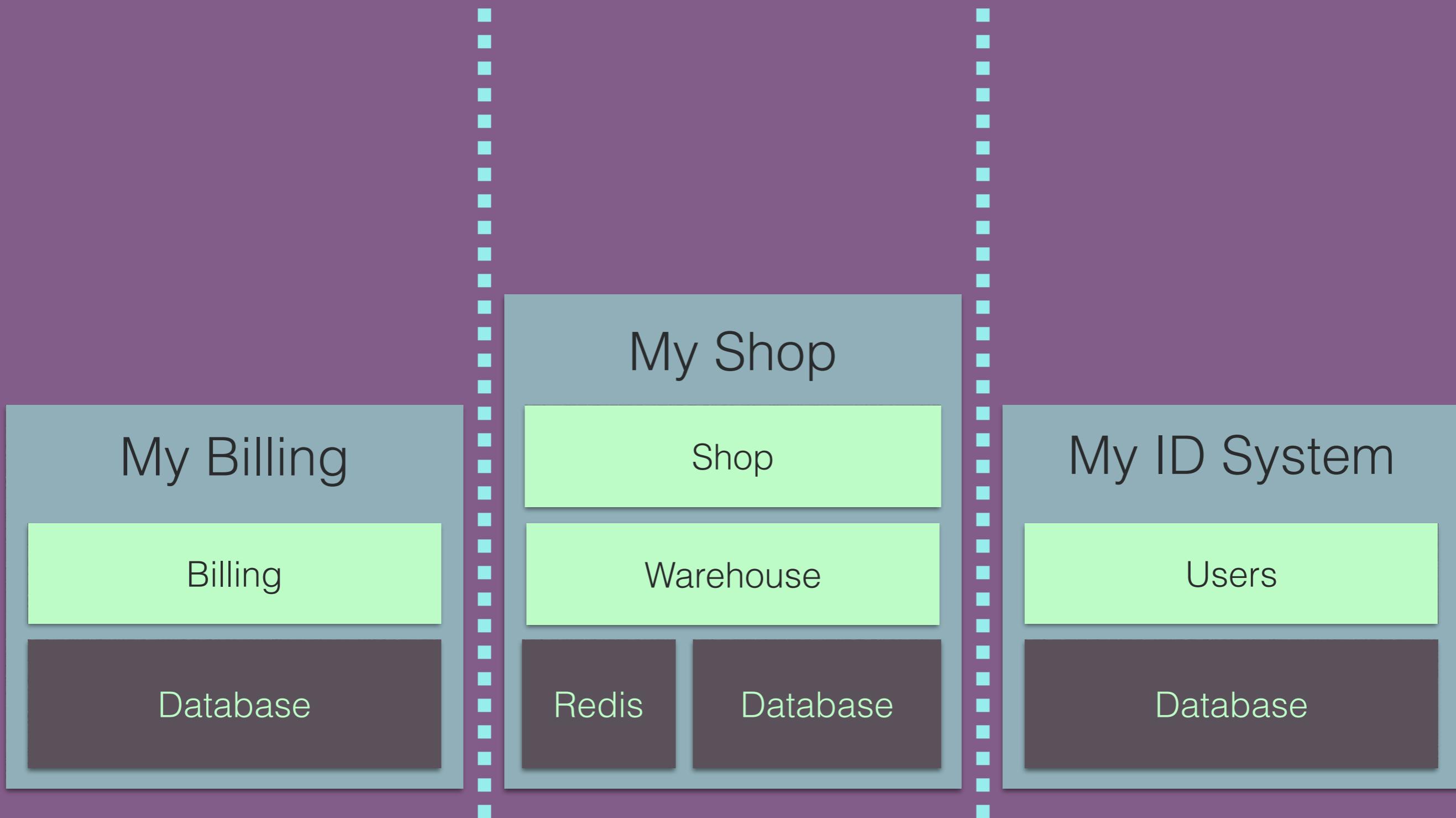
Database

My ID System

Users

Database

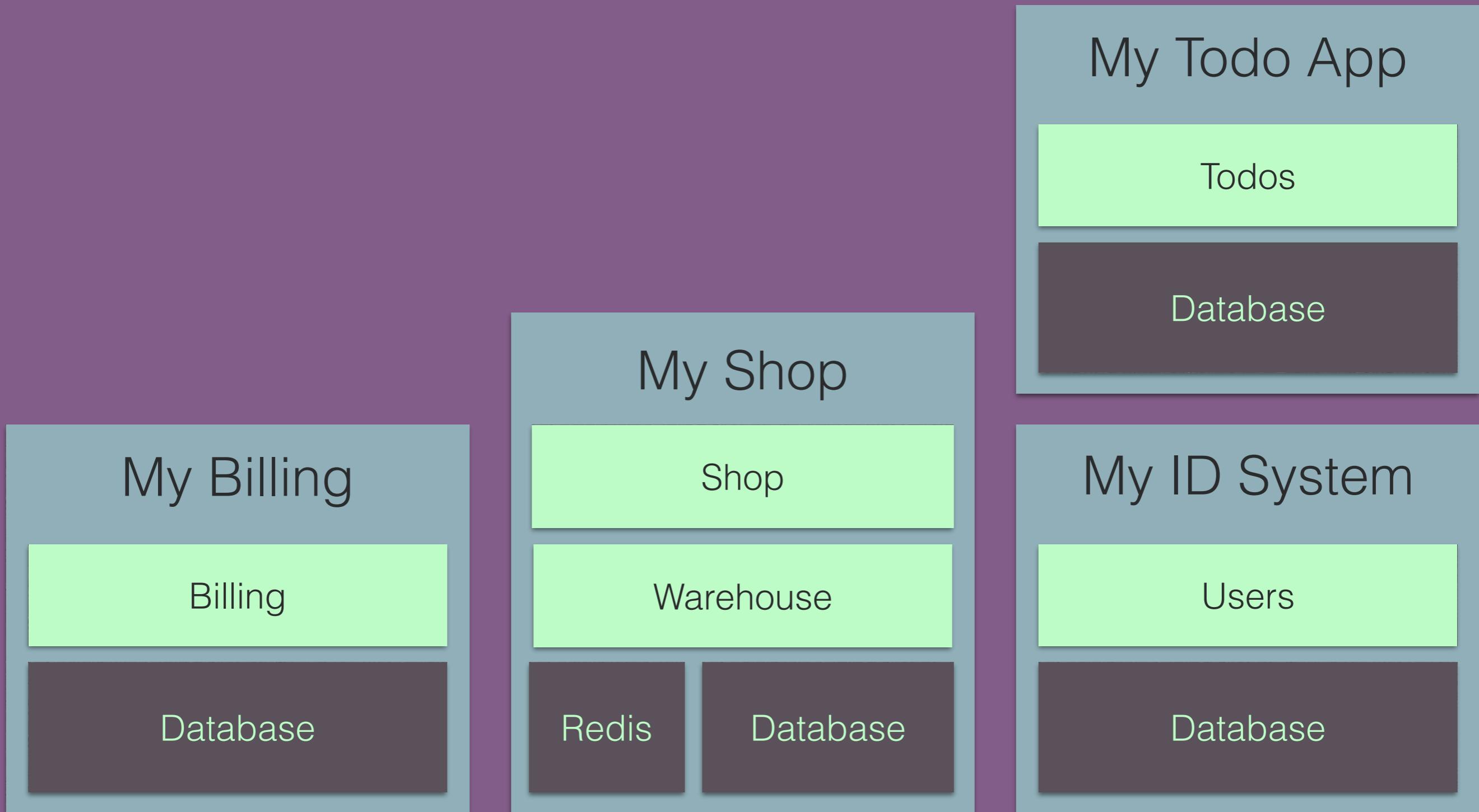
# 1. BOUNDARIES ARE EXPLICIT



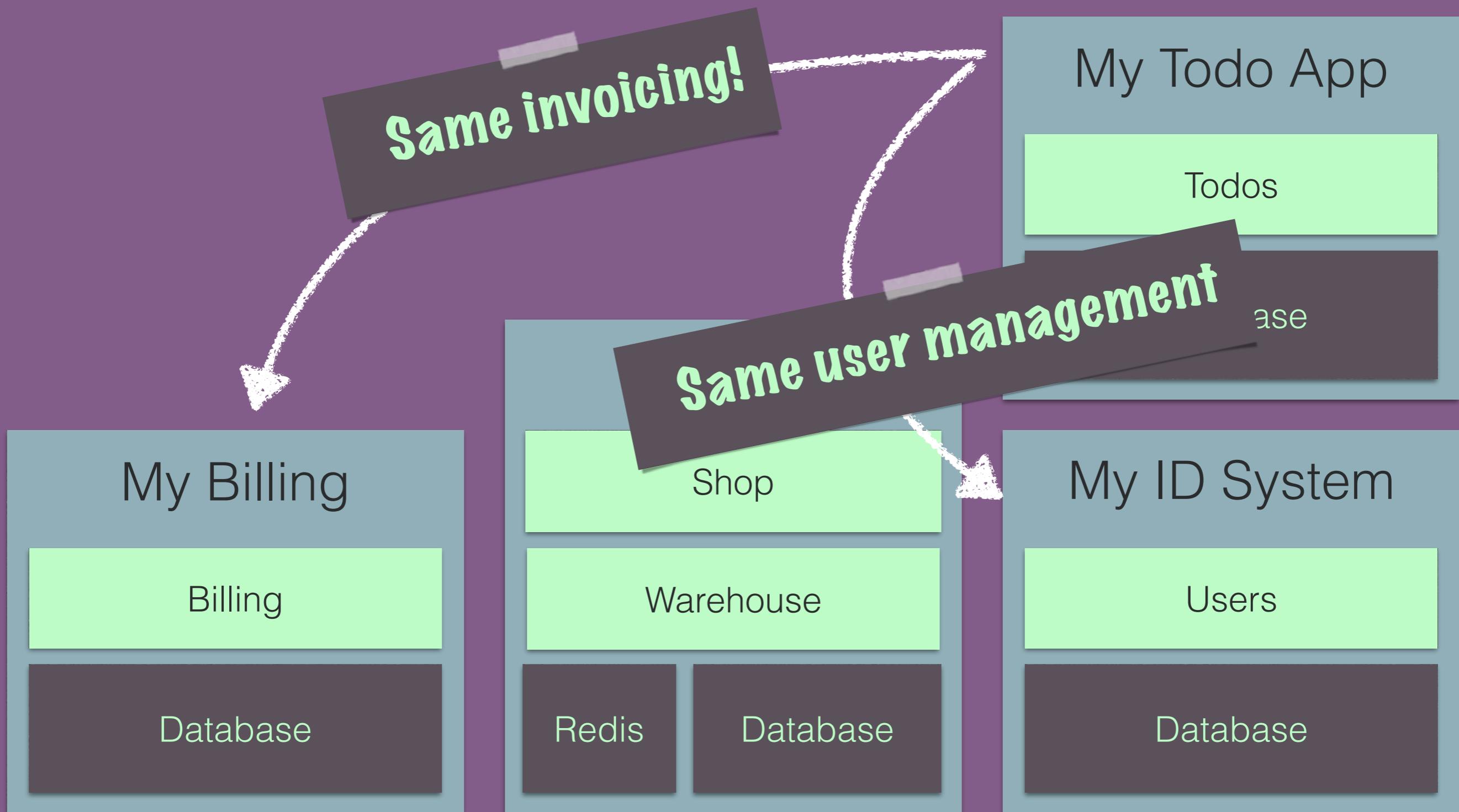
## 2. SERVICES ARE AUTONOMOUS



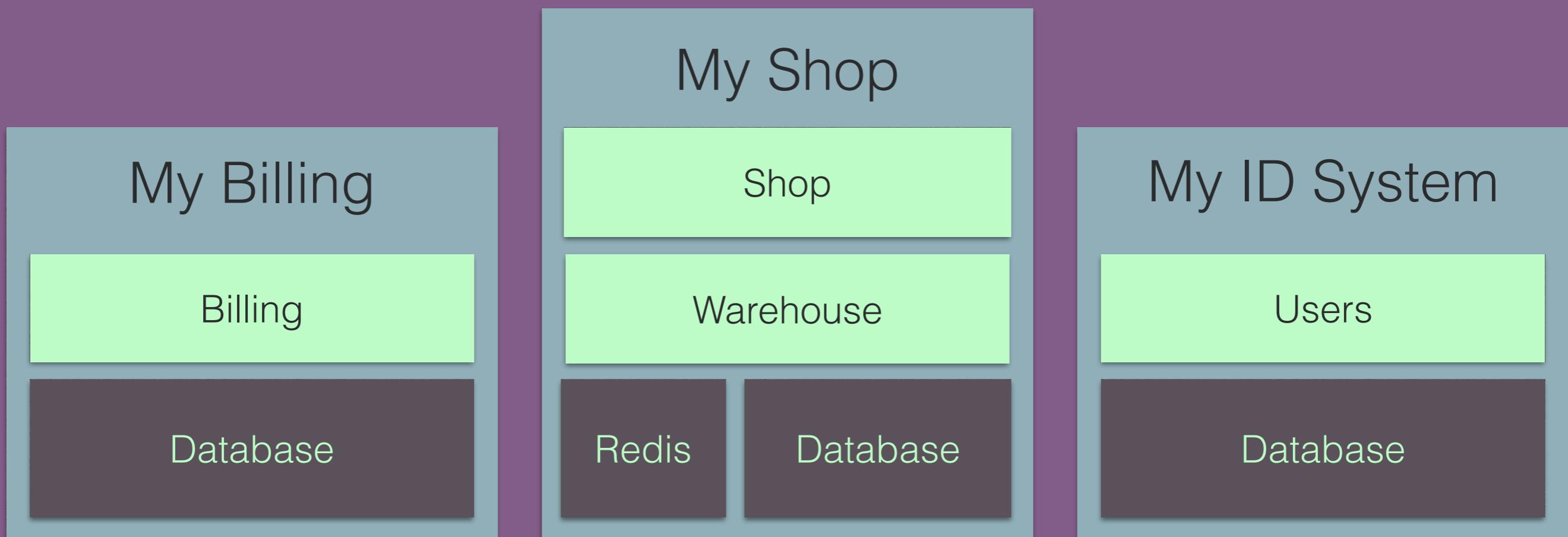
## 2. SERVICES ARE AUTONOMOUS



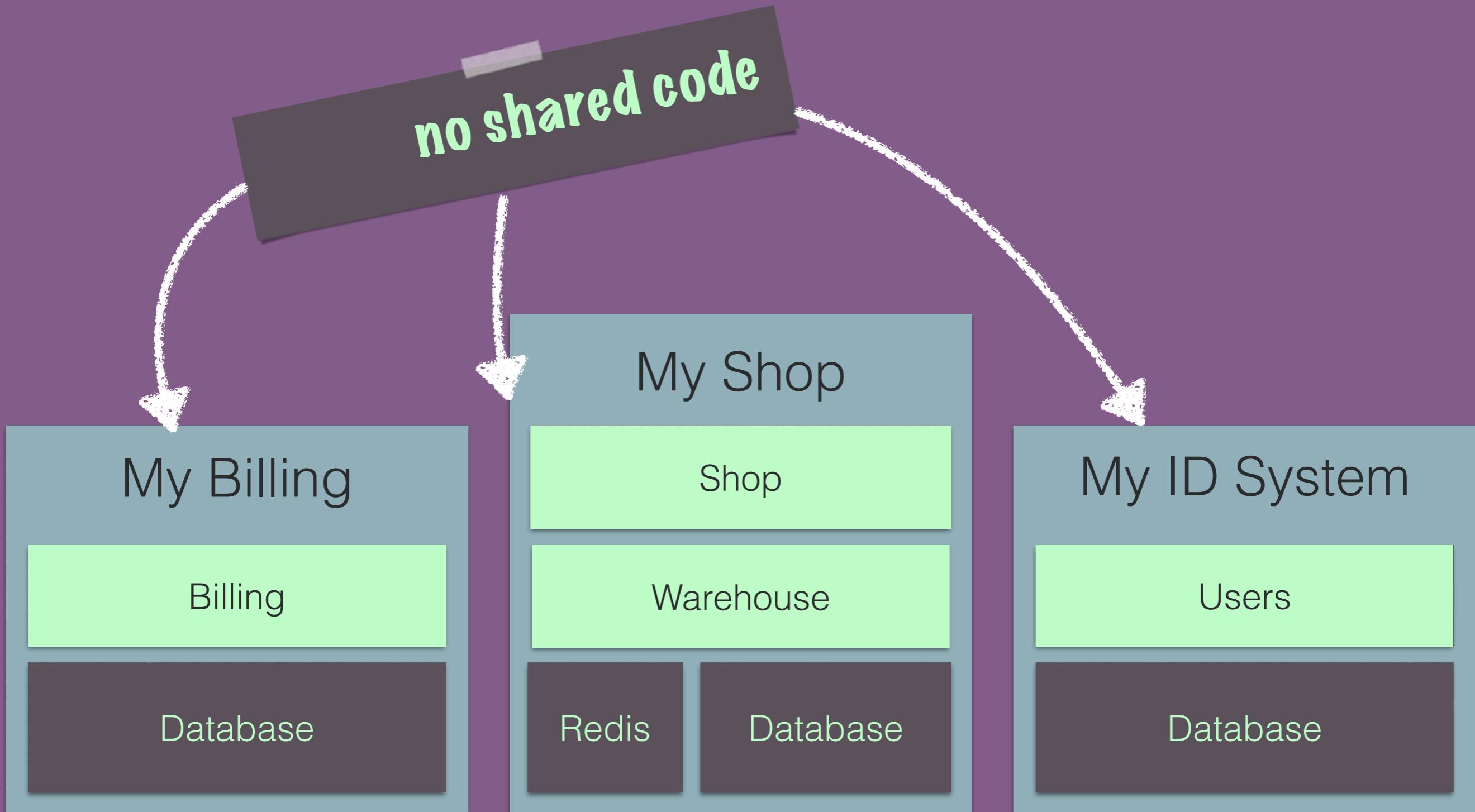
## 2. SERVICES ARE AUTONOMOUS



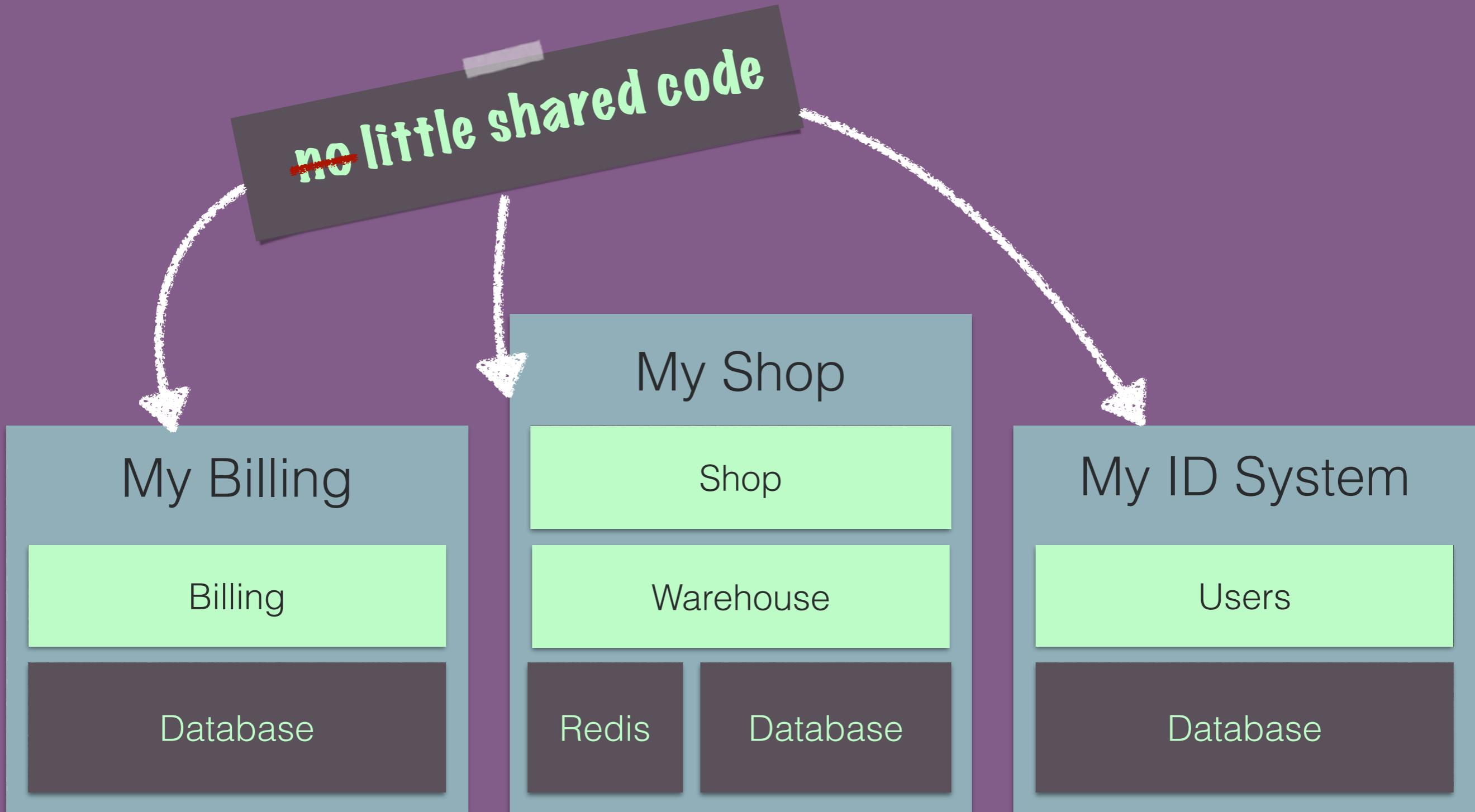
### 3. SERVICES SHARE SCHEMA AND CONTRACT, NOT CLASS



### 3. SERVICES SHARE SCHEMA AND CONTRACT, NOT CLASS



### 3. SERVICES SHARE SCHEMA AND CONTRACT, NOT CLASS



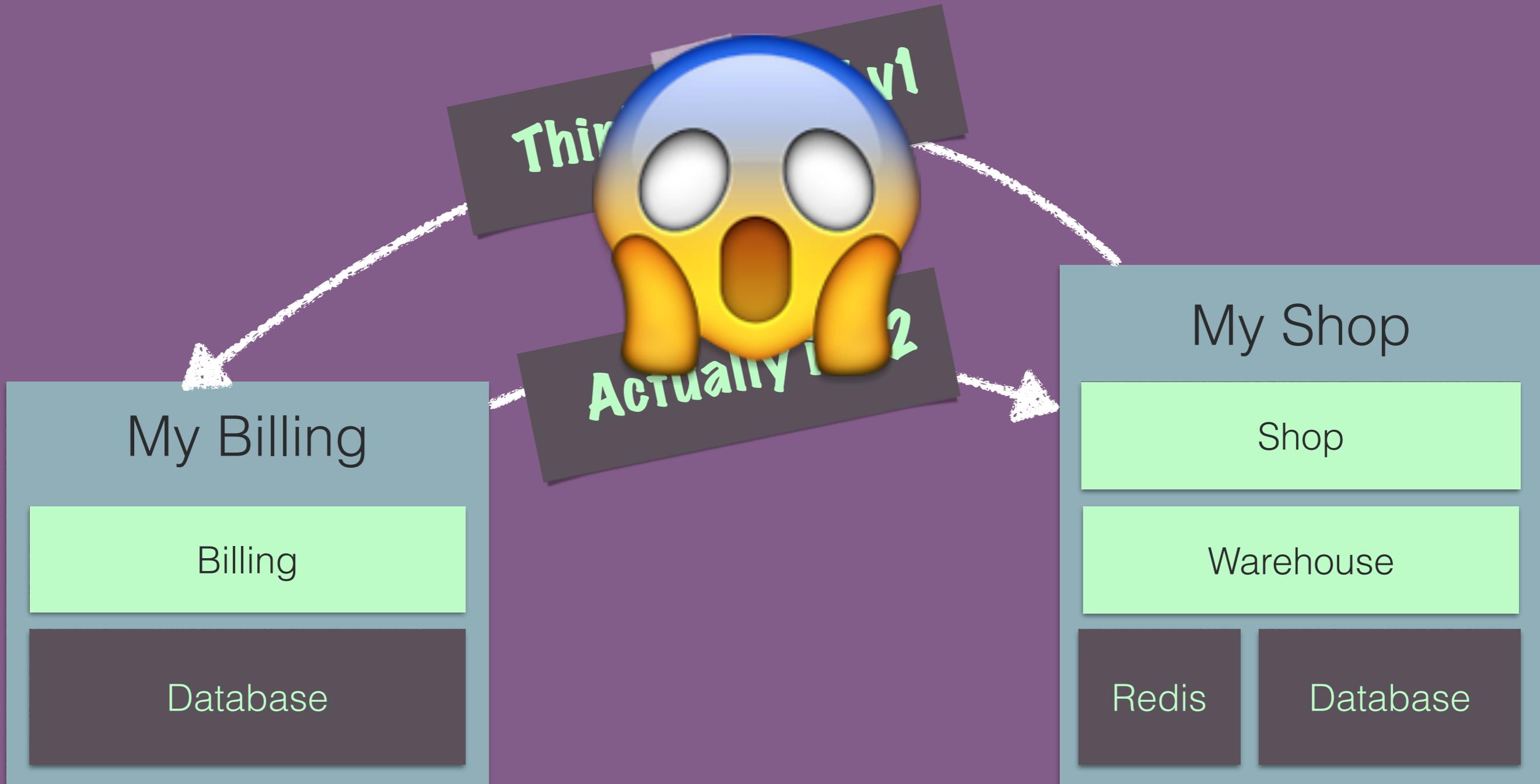
# 4. SERVICE COMPATIBILITY IS BASED ON POLICY



# 4. SERVICE COMPATIBILITY IS BASED ON POLICY



# 4. SERVICE COMPATIBILITY IS BASED ON POLICY



# FOUR TENETS OF SOA

Thorben's version:

1. Think in autonomous apps
2. Make them communicate by messages only
3. Integration test everything

# MODULAR WEB APPLICATIONS

# THE TWELVE-FACTOR APP



<http://12factor.net/>

# THE TWELVE-FACTOR APP

- “**a methodology for building software-as-a-service apps**”
- **made for “cloud” deployments**
- **scalability in mind**
- **very practical approach to SOA development**

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII. Admin processes

# THE TWELVE-FACTOR APP

I. **Codebase**

**Use git**

II. **Dependencies**

III. **Config**

IV. **Backing Services**

V. **Build, release, run**

VI. **Processes**

VII. **Port binding**

VIII. **Concurrency**

IX. **Disposability**

X. **Dev/prod parity**

XI. **Logs**

XII.

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

**Use bundler**

Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

Sposability

X. Dev/prod parity

XI. Logs

XII.

Configuration is part  
of the environment

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. — Disposability

**Make services  
configurable**

XI. Logs

XII.

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

Turn Code & Config  
into a deployment

XII.

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

Processes are stateless

Any “state” belongs in  
a backing service

# THE TWELVE-FACTOR APP

**Apps are self-contained**

**Communicate over a  
configurable port**

**III. Config**

**IV. Backing Services**

**V. Build, release, run**

**VI. Processes**

**Port binding**

**II. Concurrency**

**Disposability**

**X. Dev/prod parity**

**XI. Logs**

**XII.**

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Configuration

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

III. Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

*Scale by processes  
Define process types*

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

Disposability

X. Dev/prod parity

XI. Logs

XII.

**Quick startup  
Graceful shutdown**

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Configuration

IV. Backing services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

**Development == Production**  
**Same backing services**  
**Continuous deployments**

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

**Log to STDOUT**

# THE TWELVE-FACTOR APP

I. Codebase

II. Dependencies

III. Config

IV. Backing services

V. Build, release, and deployment keys

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

Dev/prod parity

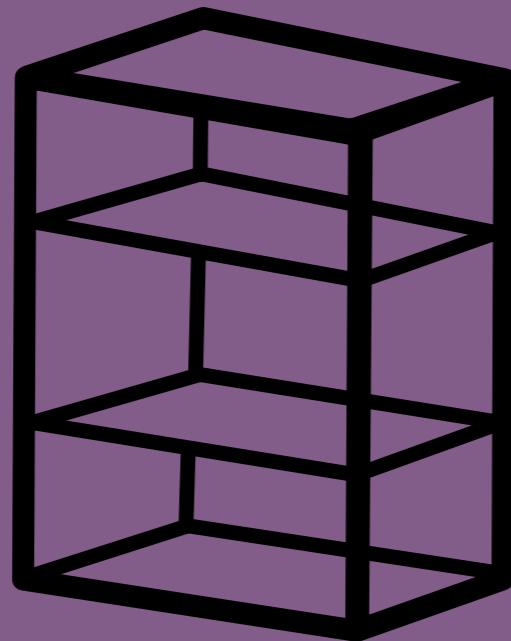
Logs

Admin processes

Same execution as  
app's processes

REPL should be part  
of the app

# A RACK 101



**rack**  
powers web applications

<http://rack.github.io/>

# WHAT IS RACK?

Rack provides a minimal interface between web servers supporting Ruby and Ruby frameworks.

# WHAT IS RACK?

- **Rack Application**
- **Rack Middleware**
- **That is it**

# WHAT IS RACK?

- Hash in
- Array out

# WHAT IS RACK?

- Hash in
- Array out

Rack environment

```
{  
  'REQUEST_METHOD' => 'POST',  
  'PATH_INFO' => '/api/v1/todos',  
  #...  
  'HTTP_AUTHORIZATION' => 'Basic bGFzdDp1bmljb3Ju',  
}
```

# WHAT IS RACK?

in  
**HTTP Header**

- Array out

```
{  
  'REQUEST_METHOD' => 'POST',  
  'PATH_INFO' => '/api/v1/todos',  
  ...  
  'HTTP_AUTHORIZATION' => 'Basic bGFzdDp1bmljb3Ju',  
}
```

**Rack environment**

# WHAT IS RACK?

- Hash in
- Array out

Iterable  
to allow  
streaming

Rack response

```
[200, {"Content-Type" => "text/plain"}, ['Body']]
```

HTTP Status

HTTP Response Header

Response Body

# WHAT IS RACK?

- Hash in
- Array out

Iterable  
to allow  
streaming

Rack response

```
[200, {"Content-Type" => "text/plain"}, ['Body']]
```

HTTP Status

HTTP Response Header

Response Body

# A RACK APP

- must respond to “call” with the Rack environment as the only argument

# A RACK APP

Rack environment

```
class MyApp
  def call(env)
    [200, {'Content-Type' => 'text/plain'}, ['Hey there']]
  end
end
```

Rack response array

**BUT WAIT...**

**CALL YOU SAID?**

```
$ ri Proc#call
```

```
= Proc#call
```

```
(from ruby core)
```

```
-----  
prc.call(params,...)    -> obj  
prc[params,...]         -> obj  
prc.(params,...)        -> obj  
-----
```

Invokes the block, setting the block's parameters to the values in params using something close to method calling semantics....

**PROCS ARE RACK APPS!**

# ALSO A RACK APP

Rack environment

```
-> (env) { [200, {'Content-Type' => 'text/plain'}, ['...']] }
```

Rack response array

# CHALLENGE ACCEPTED



# SETTING UP OUR LAB

```
$ git clone https://github.com/walski/soa-workshop.git
$ cd soa-workshop/box
$ vagrant up
$ vagrant ssh
```

```
vagrant@...$ cd /vagrant/challenges/01
vagrant@...$ bundle exec rake
```

# SETTING UP OUR LAB

- **edit code on your machine (box/challenges)**
- **run the specs in Vagrant**

# CHALLENGE ACCEPTED



RACK'S MIGHTY FRIENDS

# RACK::REQUEST

---

# RACK::RESPONSE

---

# RACK::BUILDER

---

```
class MyAppA
  def call(env)
    [200, {'Content-Type' => 'text/plain'}, ['Hey A']]
  end
end

class MyAppB
  def call(env)
    [200, {'Content-Type' => 'text/plain'}, ['Hey B']]
  end
end

app = Rack::Builder.new {
  map("/a") { run MyAppA }
  map("/b") { run MyAppB }
  [200, {'Content-Type' => 'text/plain'}, ['What?']]
}
```

# RACKUP

---

CONFIG.RU

1. Create a rackup config

```
Bundler.require  
require 'myapp'
```

```
run MyApp.new
```

2. Run your web app

```
$ rackup
```

# RTFM

---

[rack.rubyforge.org/doc/README\\_rdoc.html](http://rack.rubyforge.org/doc/README_rdoc.html)

# CHALLENGE ACCEPTED



# A SINATRA PRIMER

---



# WHAT IS SINATRA?

- **simple web framework**
- **routing / templates / “all we need”**
- **Sinatra apps are Rack apps**
- **good example**

# WHAT IS SINATRA?

- **simple web framework**
- **routing / templates / “all we need”**
- **Sinatra apps are Rack apps**
- **good example**

# WHAT IS SINATRA?

```
require 'sinatra/base'

class MyApp < Sinatra::Base
  get '/hossa' do
    'cool!'
  end
end
```

**SINATRA CAN DO SO MUCH MORE.  
BUT SERIOUSLY...**

# RTFM

---

[www.sinatrarb.com/intro.html](http://www.sinatrarb.com/intro.html)

# CHALLENGE ACCEPTED



# A RACK MIDDLEWARE

- **is a wrapper around an app**
- **gets the app passed as the only argument to its constructor**
- **must respond to “call” with the Rack environment as the only argument**

# A RACK MIDDLEWARE

```
class MyMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    if env['PATH_INFO'] = '/yay'
      [200, {'Content-Type' => 'text/plain'}, ['yay']]
    return
  end
  @app.call(env)
end
end
```

**Parent Rack app**



# CHALLENGE ACCEPTED



# I HATE RUBY!

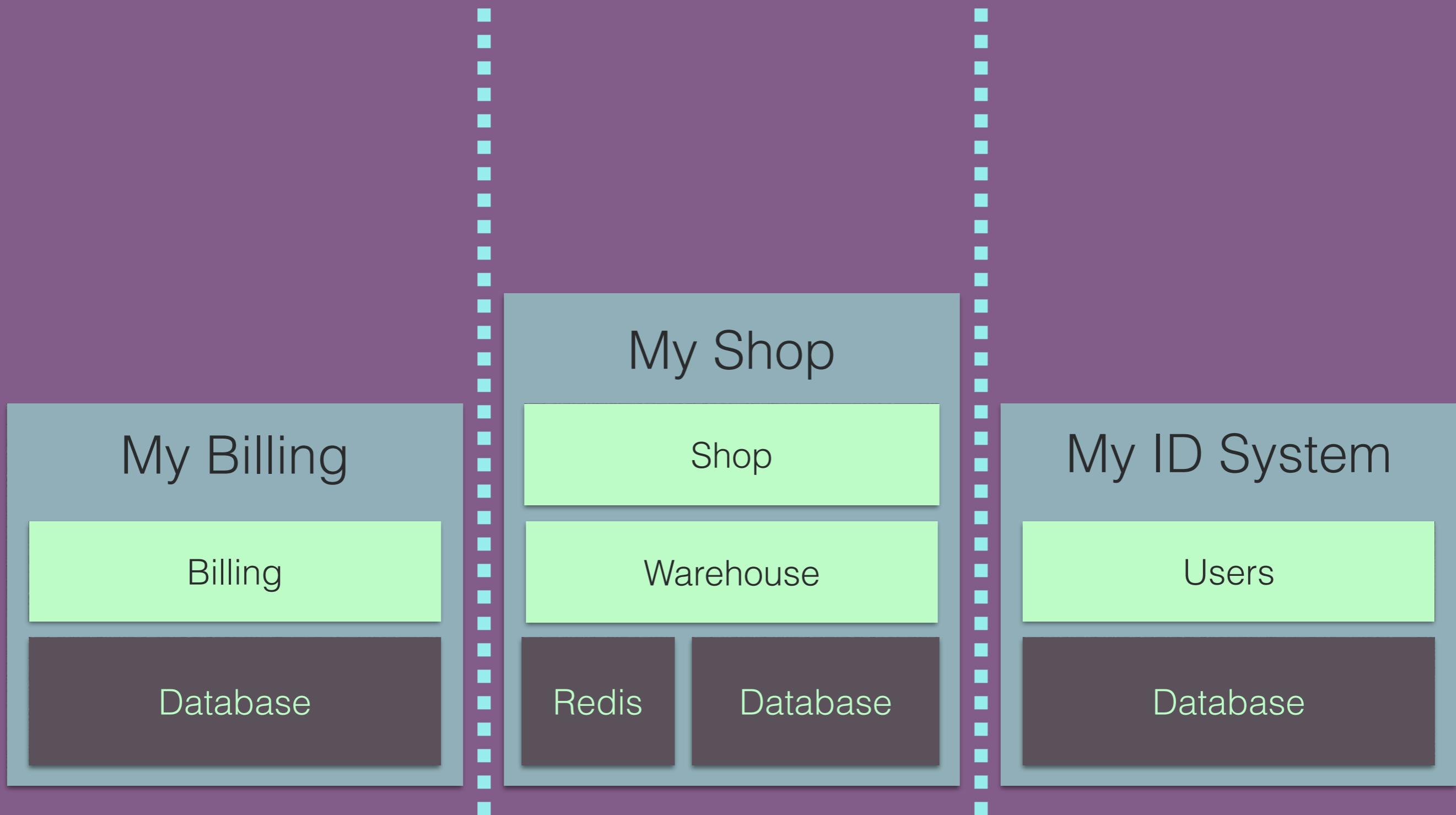
- **Don't you worry:**
  - **Python WSGI**
  - **Perl Web Server Gateway Interface**
  - **JavaScript Gateway Interface**
  - ...

**SOA**

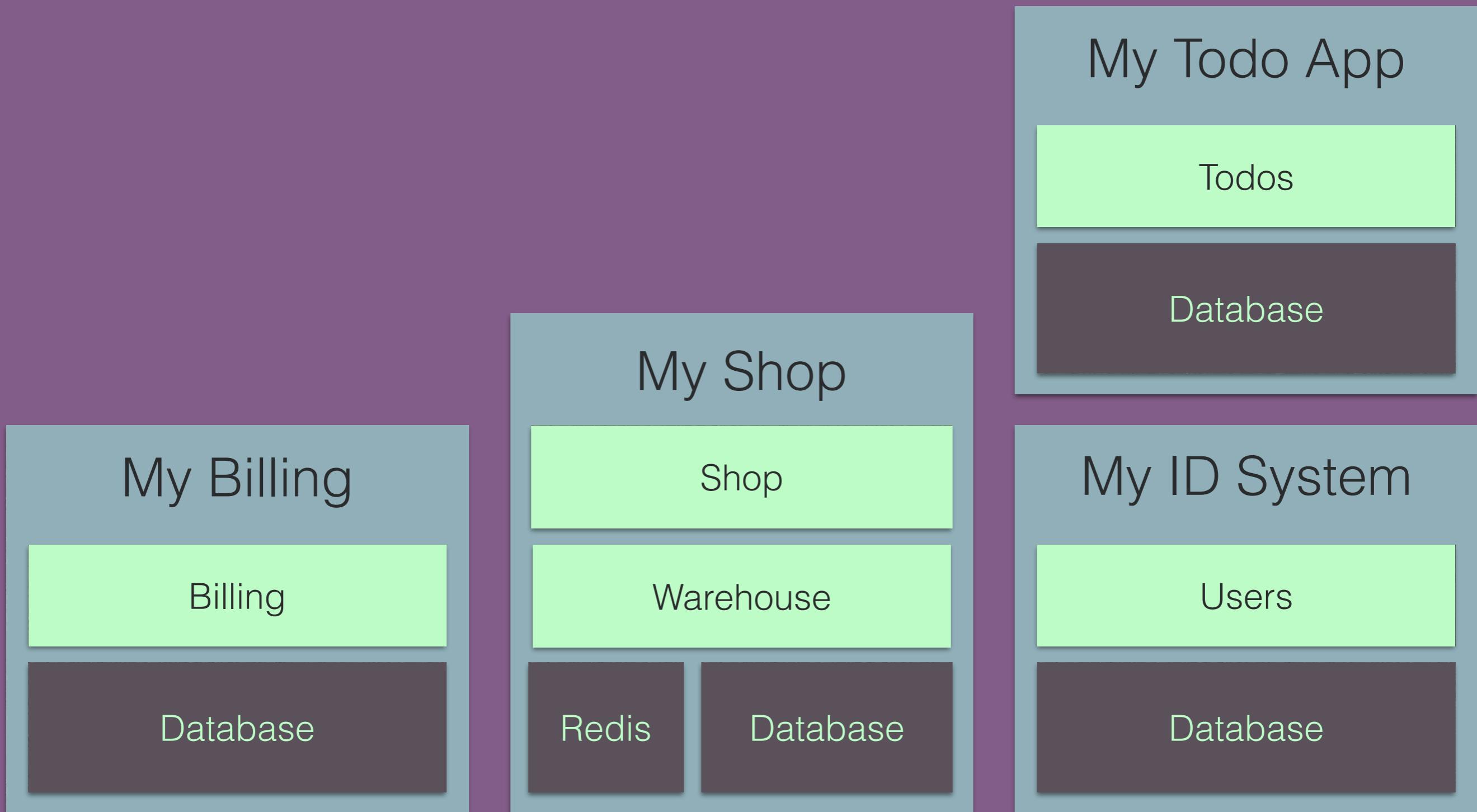
**DOCUMENTATION**

# QUICK RECAP

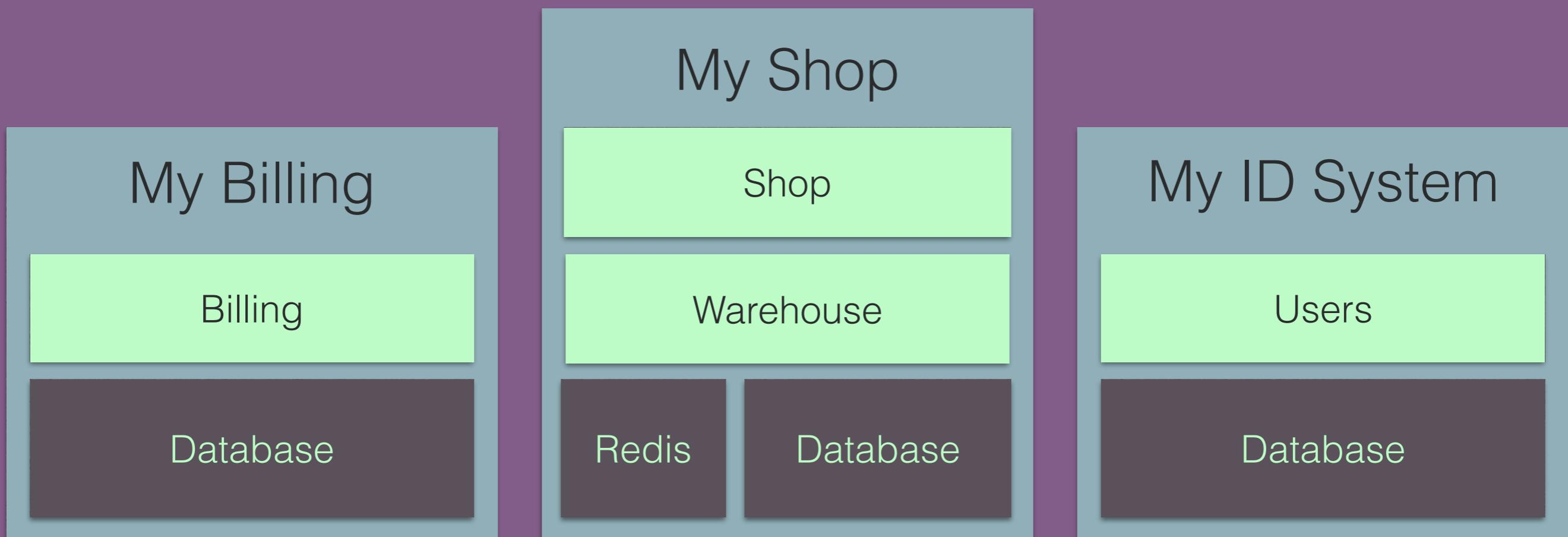
# 1. BOUNDARIES ARE EXPLICIT



## 2. SERVICES ARE AUTONOMOUS



### 3. SERVICES SHARE SCHEMA AND CONTRACT, NOT CLASS



4. SERVICE COMPATIBILITY IS BASED ON

## 4. SERVICE

COMPATIBILITY IS



# **4. SERVICE COMPATIBILITY IS BASED ON POLICY**

# DOCUMENT YOUR APIs

- APIs are your policy / contract
- Helps to frame your mindset

# DOCUMENT YOUR APIs

- APIs are your policy / contract
- Helps to frame your mindset

# **README DRIVEN DEVELOPMENT**

---

- **Write your Readme before writing code**
- **Before ANY code!**
- **Forces you to think what you are about to build**
- **Enables team discussions**

# READMES IN SOA

- The Readme can be / include your API documentation
- Good for cross functional teams

# WHAT NEEDS TO BE DOCUMENTED?

- Take advantage of the simplicity of HTTP:
  - Request
  - Response

# WHAT NEEDS TO BE DOCUMENTED?

- Request
- Verb
- Path
- Headers
- Body

# WHAT NEEDS TO BE DOCUMENTED?

- Response
- Status
- Headers
- Body
- One documented request might lead to several different responses depending on the request specifics (unauthenticated vs. authenticated requests...)

# WHAT NEEDS TO BE DOCUMENTED?

- **Miscellaneous**
- **Versioning**
- **Authentication**
- **Pagination**
- ...

# SAAAAAAAMPLES



[developer.github.com/v3/](https://developer.github.com/v3/)

# SAAAAAAAMPLES



<https://devcenter.heroku.com/articles/platform-api-reference>

# HOW TO DOCUMENT?

- Write it yourself
- Generated from code / comments
- Generated from tests / specs

# HOW TO DOCUMENT?

- Write it yourself
- Generated from code / comments
- Generated from tests / specs



No documentation  
challenge :/

**CHALLENGE DENIED.**

# SOA TESTING

**WHY IS THIS EVEN A THING?**

# INTEGRATION TESTS

vs.

# UNIT TESTS

Monolithic apps  
can be tested with  
unit tests

SOA apps can't

They rely on the  
integration with  
each other!

# YOUR OPTIONS

1. Have a mock for every single service
2. Find a way to run the real systems

# MANAGE DEPENDENCIES

Each app is a gem

# CREATING GEMS

```
$ bundle gem my_gem
  create  my_gem/Gemfile
  create  my_gem/Rakefile
  create  my_gem/LICENSE.txt
  create  my_gem/README.md
  create  my_gem/.gitignore
  create  my_gem/hello-world.gemspec
  create  my_gem/lib/my_gem.rb
  create  my_gem/lib/my_gem/version.rb
```

# CREATING GEMS

Pretty much empty

```
$ bundle gem my_gem
  create  my_gem/Gemfile
  create  my_gem/Rakefile
  create  my_gem/LICENSE.txt
  create  my_gem/README.md
  create  my_gem/.gitignore
  create  my_gem/hello-world.gemspec
  create  my_gem/lib/my_gem.rb
  create  my_gem/lib/my_gem/version.rb
```

# CREATING GEMS

Place to track your  
dependencies, edit name,  
description, etc...

```
$ bundle gem my_gem
  create  my_gem/Gemfile
  create  my_gem/Rakefile
  create  my_gem/LICENSE.txt
  create  my_gem/README.md
  create  my_gem/.gitignore
  create  my_gem/hello-world.gemspec
  create  my_gem/lib/my_gem.rb
  create  my_gem/lib/my_gem/version.rb
```

# CREATING GEMS

```
Gem::Specification.new do |spec|  
  ...  
  spec.add_development_dependency "bundler", "~> 1.3"  
  spec.add_development_dependency "rake"  
end
```



Use `spec.add_dependency`  
for runtime dependencies

# CHALLENGE ACCEPTED



# COMMUNICATION

*Wrap your communication  
between apps*

# FARADAY

```
conn = Faraday.new(:url => 'http://example.com')
response = conn.get '/'
puts response.body
```

<https://github.com/lostisland/faraday>

# CHALLENGE ACCEPTED

