What's the scariest thing about serverless?

Long-running tasks?

Compliance?

Using binaries and large dependencies?

Cold start?

Cold start with VPC?

Local development and debugging?

Losing control?

Node.js?

But, what about...


BIG


BAD

VENDOR LOCK-IN

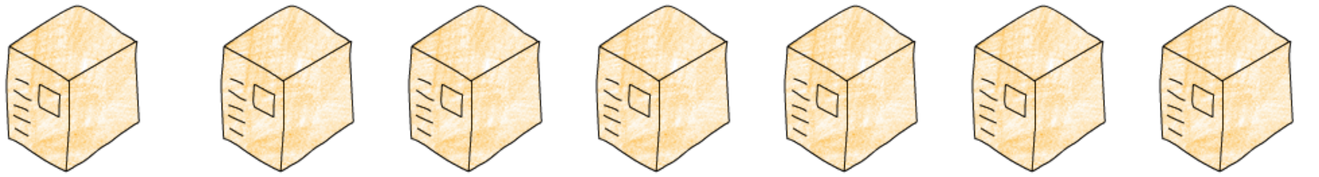
What is vendor lock-in?

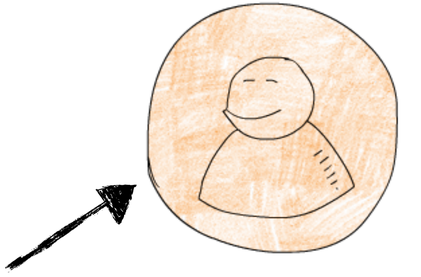
"In economics, vendor lock-in,
makes a customer dependent on a vendor
for products and services, unable to use another vendor
without substantial switching costs."


Hm, I think
I'll need
a Server


Should I buy it or rent it?

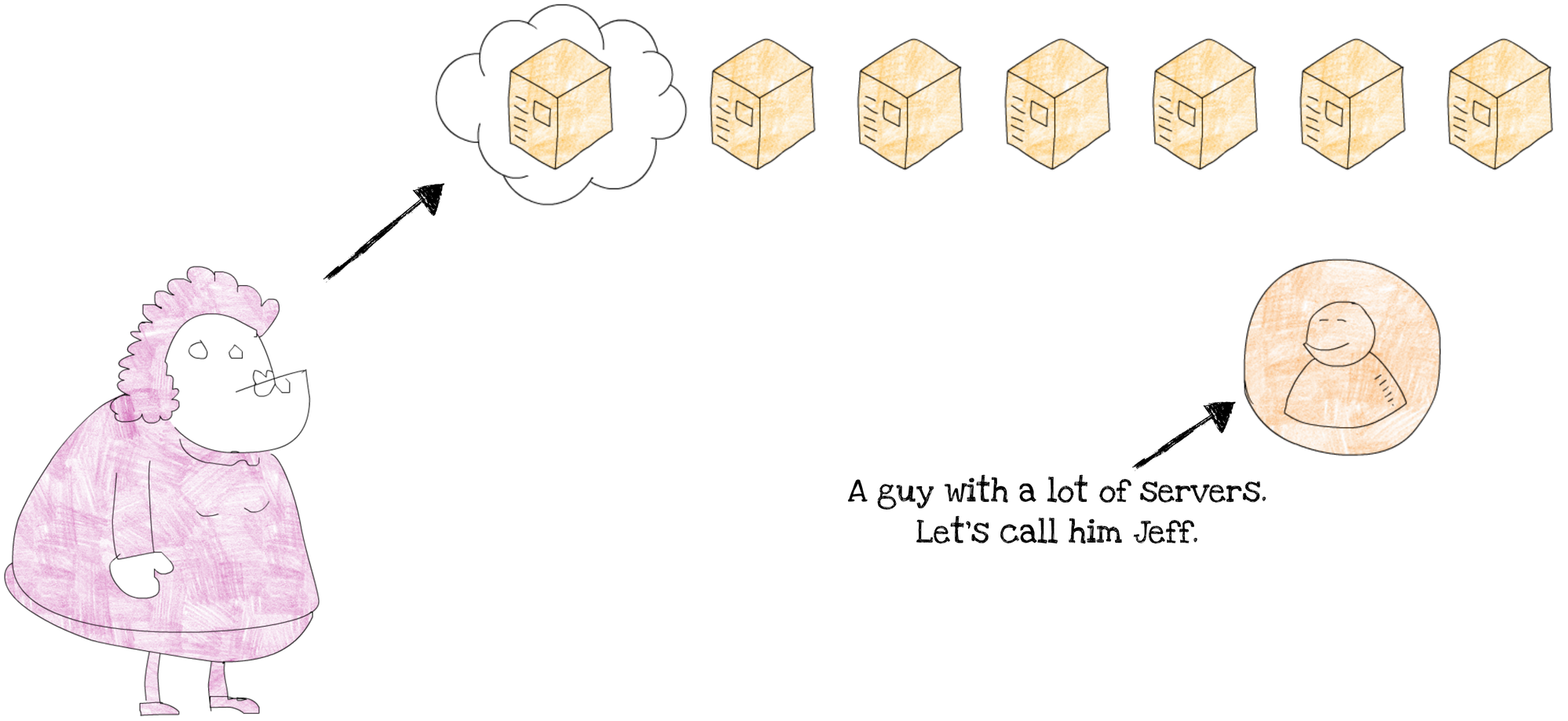
@slobodan_

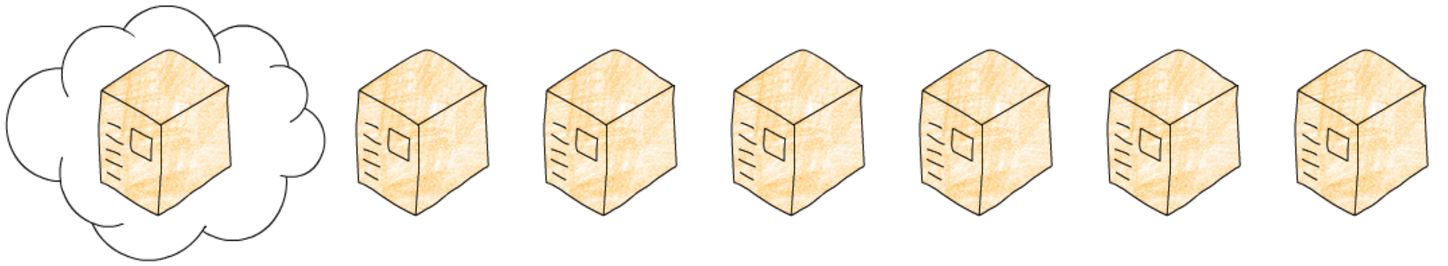


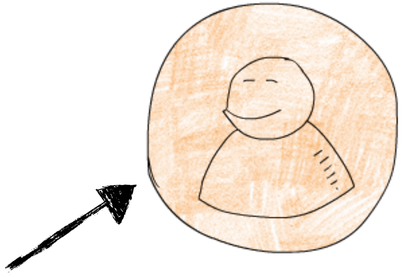


A guy with a lot of servers.
Let's call him Jeff.

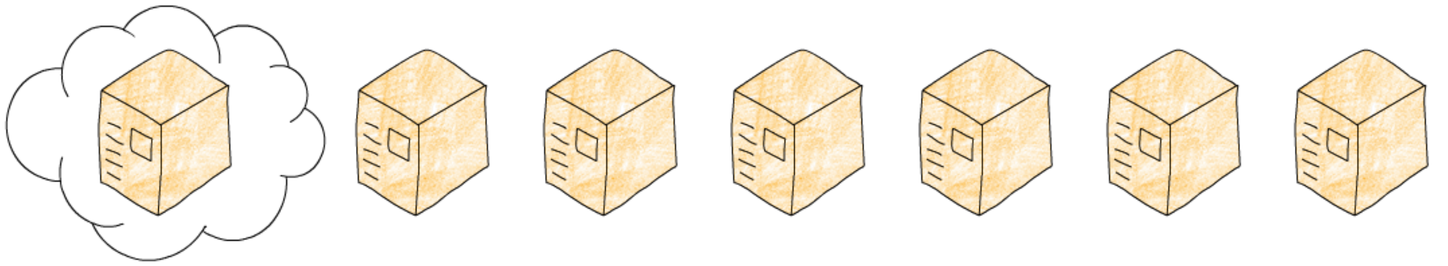
@slobodan_

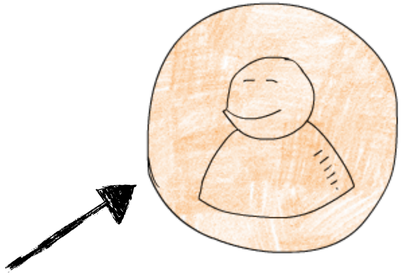

A guy with a lot of servers.
Let's call him Jeff.



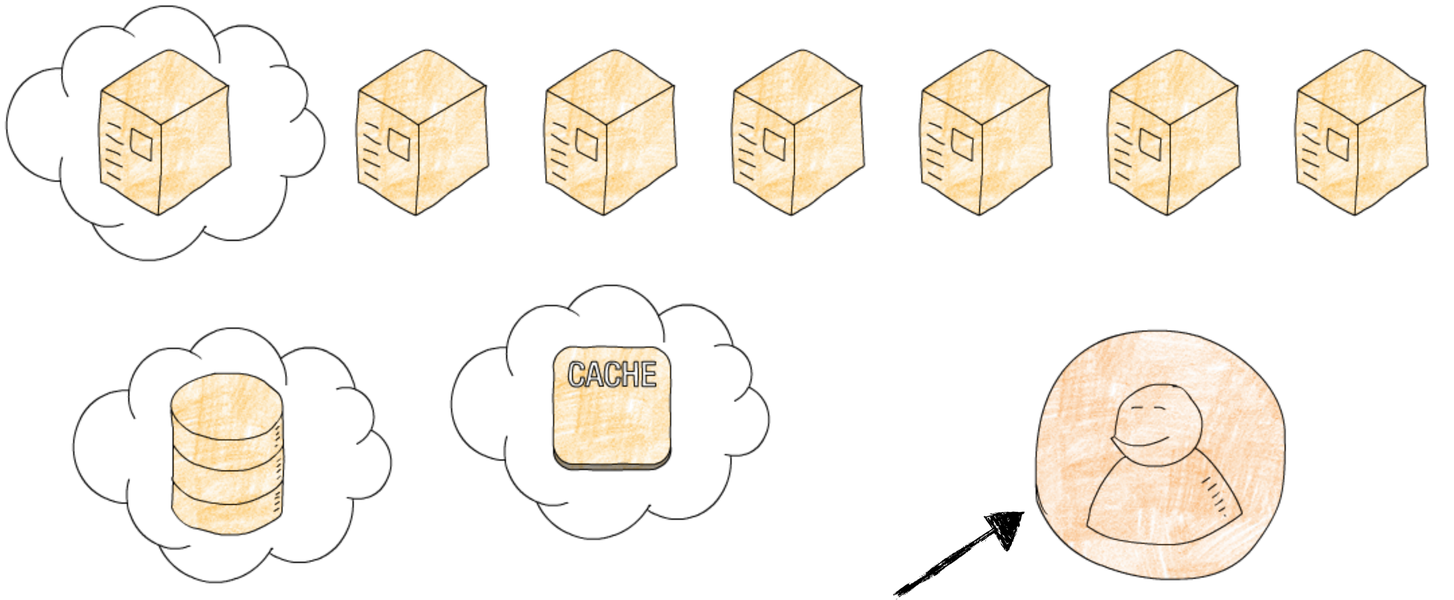


Jeff is smart, and he knows
how do you use his servers.

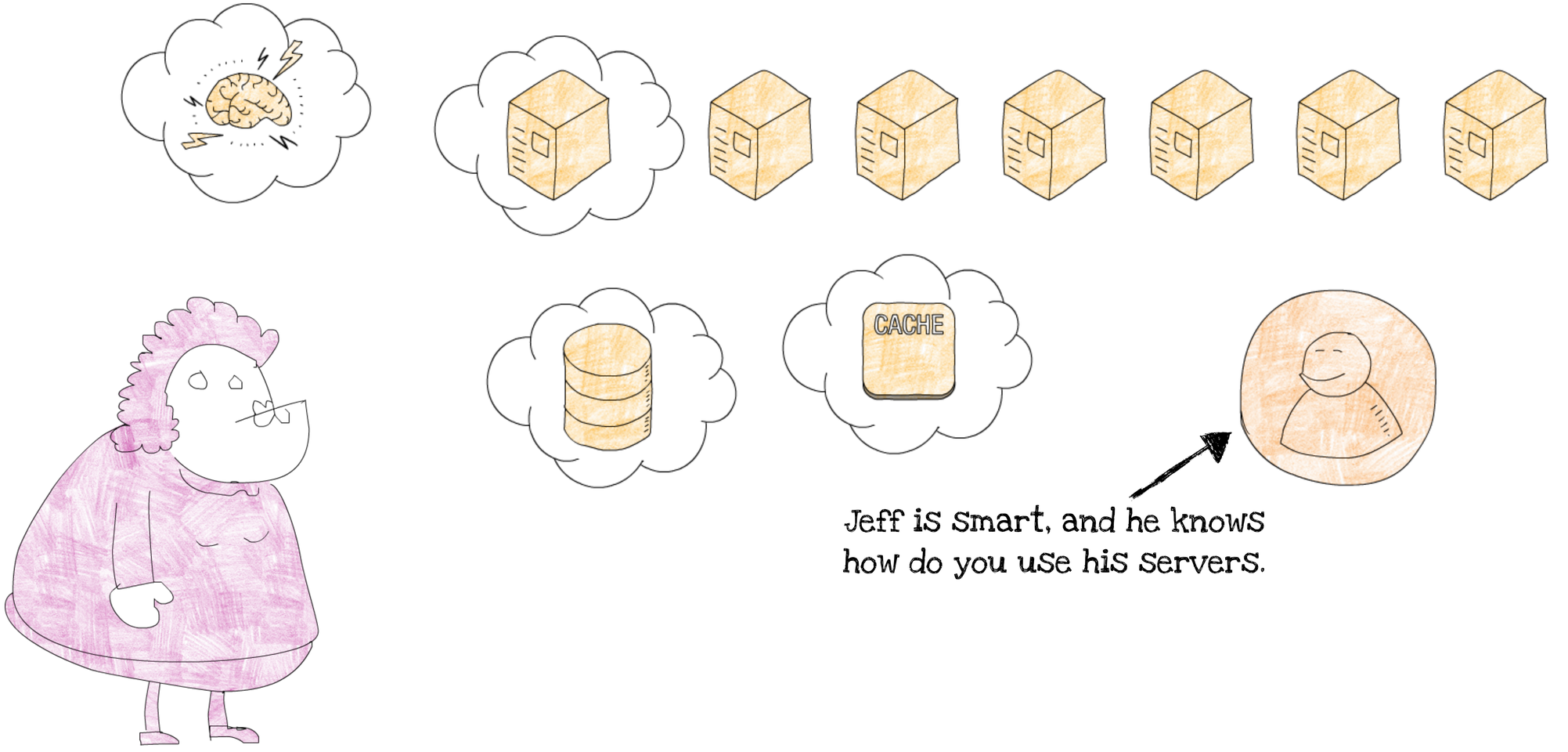


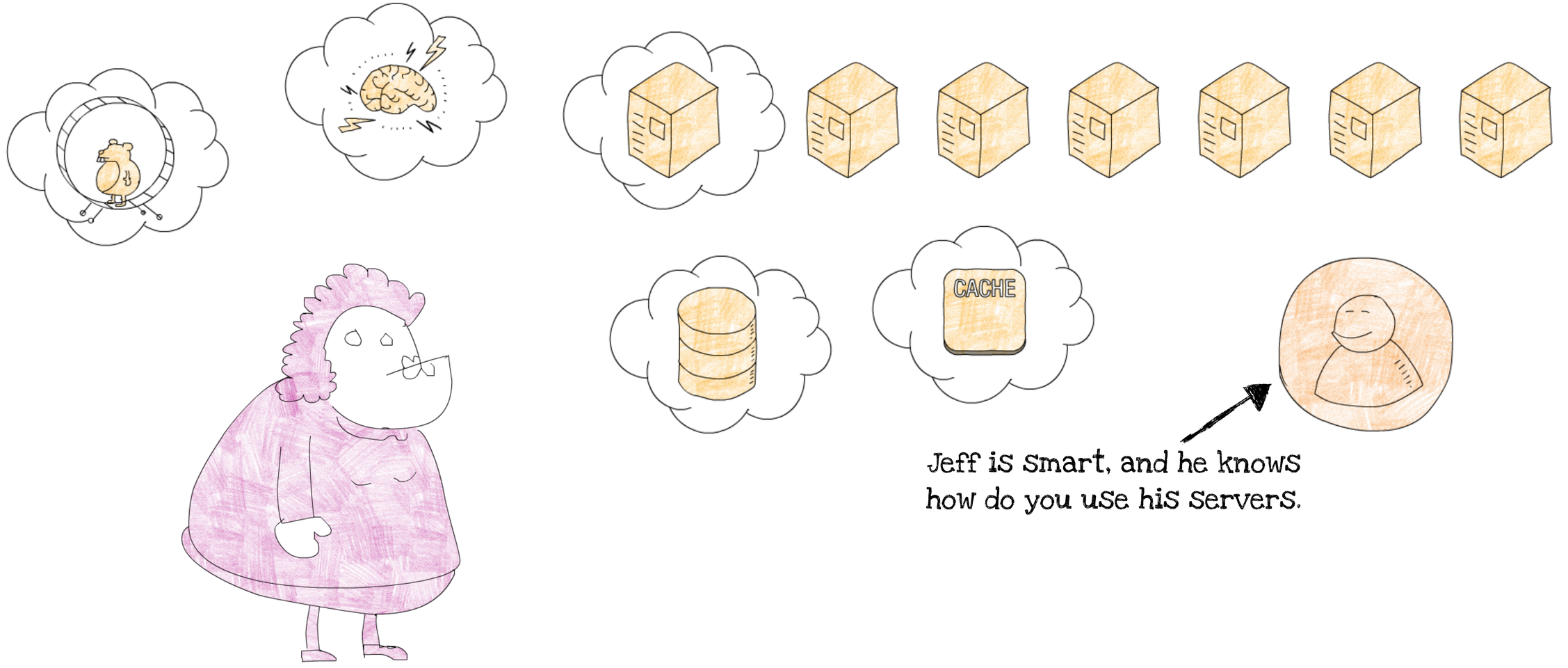


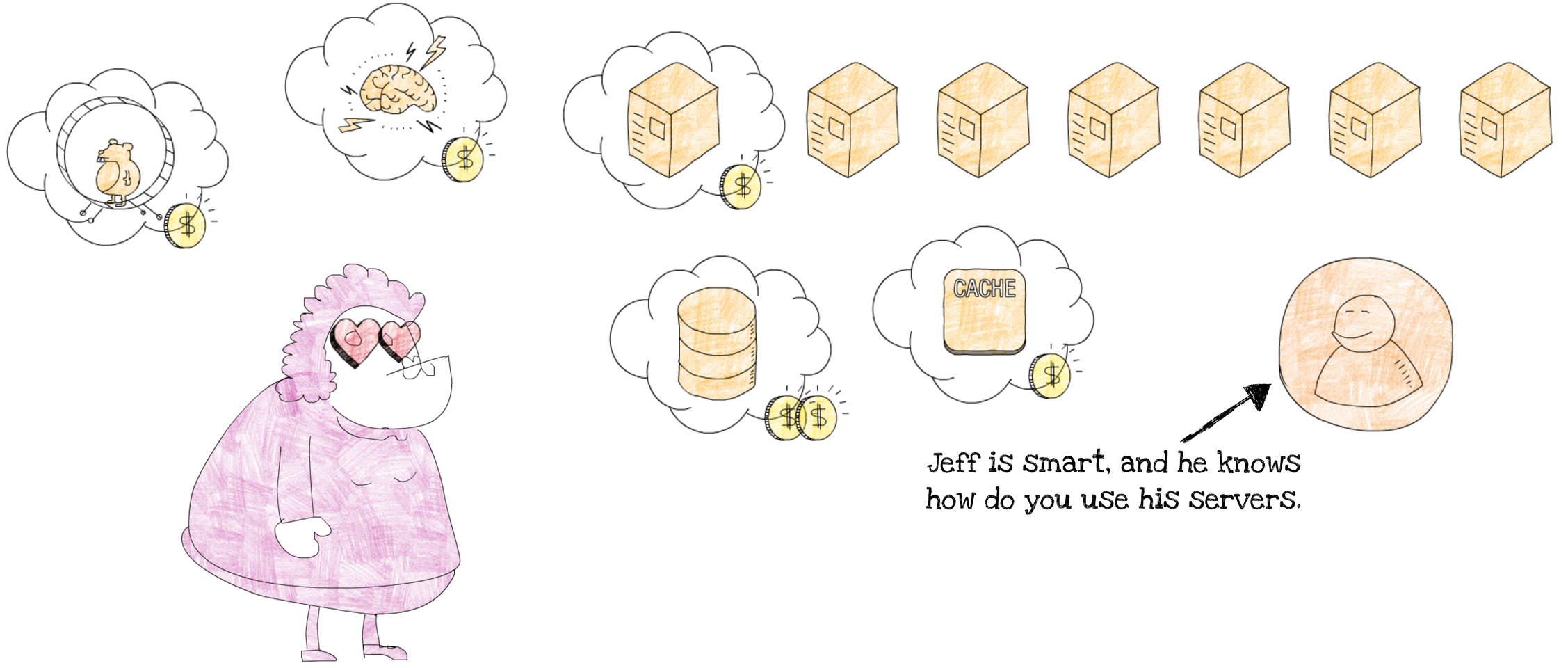
Jeff is smart, and he knows
how do you use his servers.

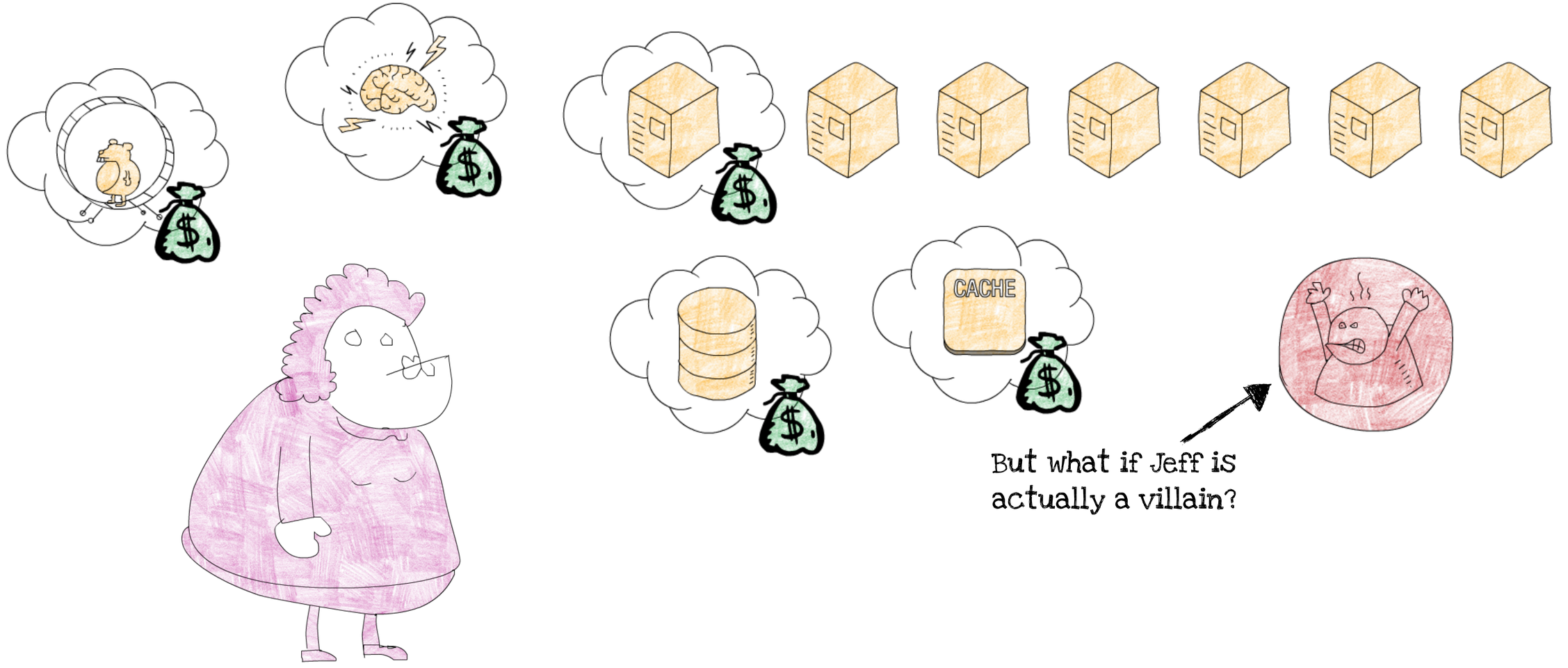


Jeff is smart, and he knows
how do you use his servers.

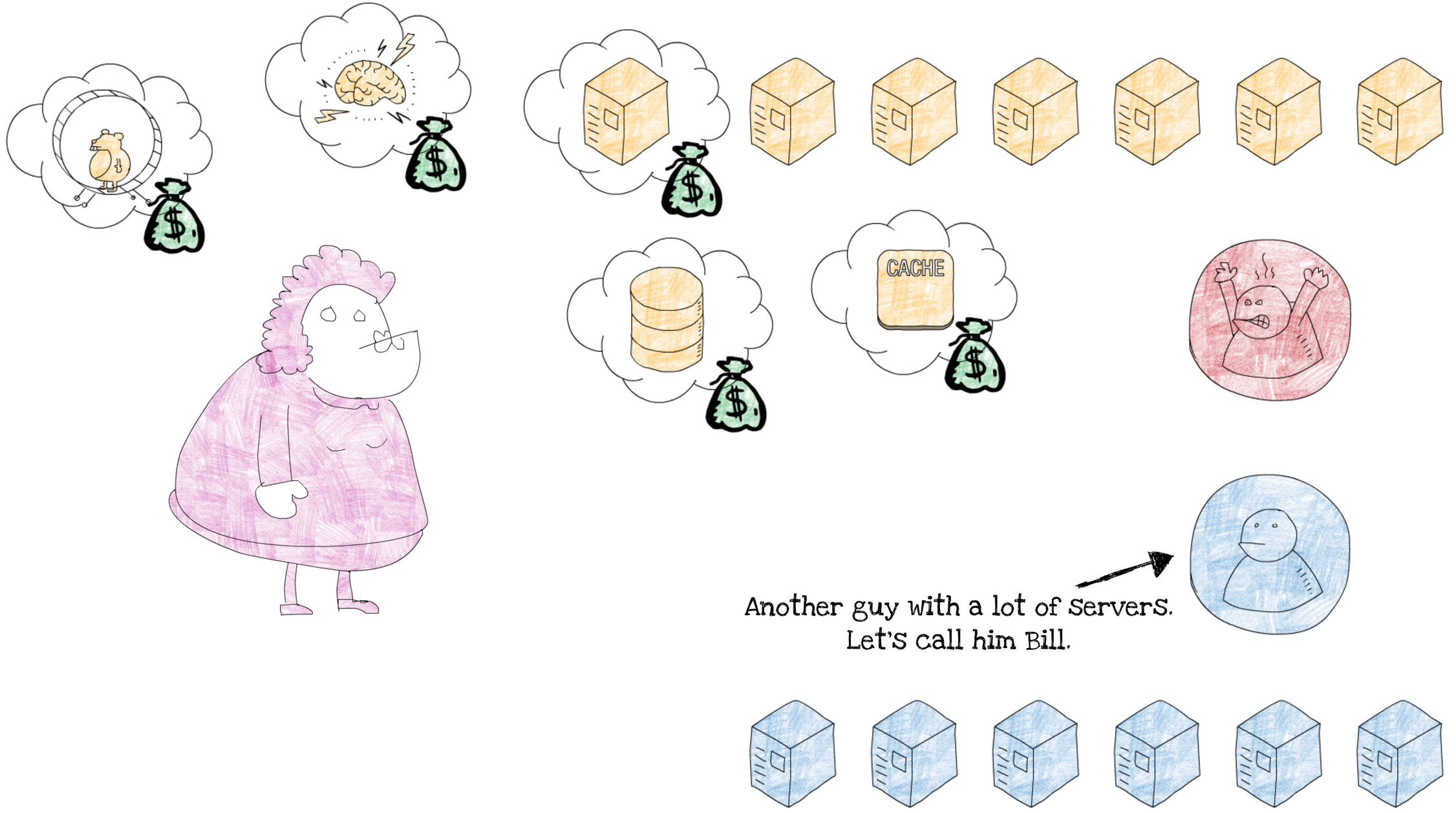

CACHE
Jeff is smart, and he knows
how do you use his servers.

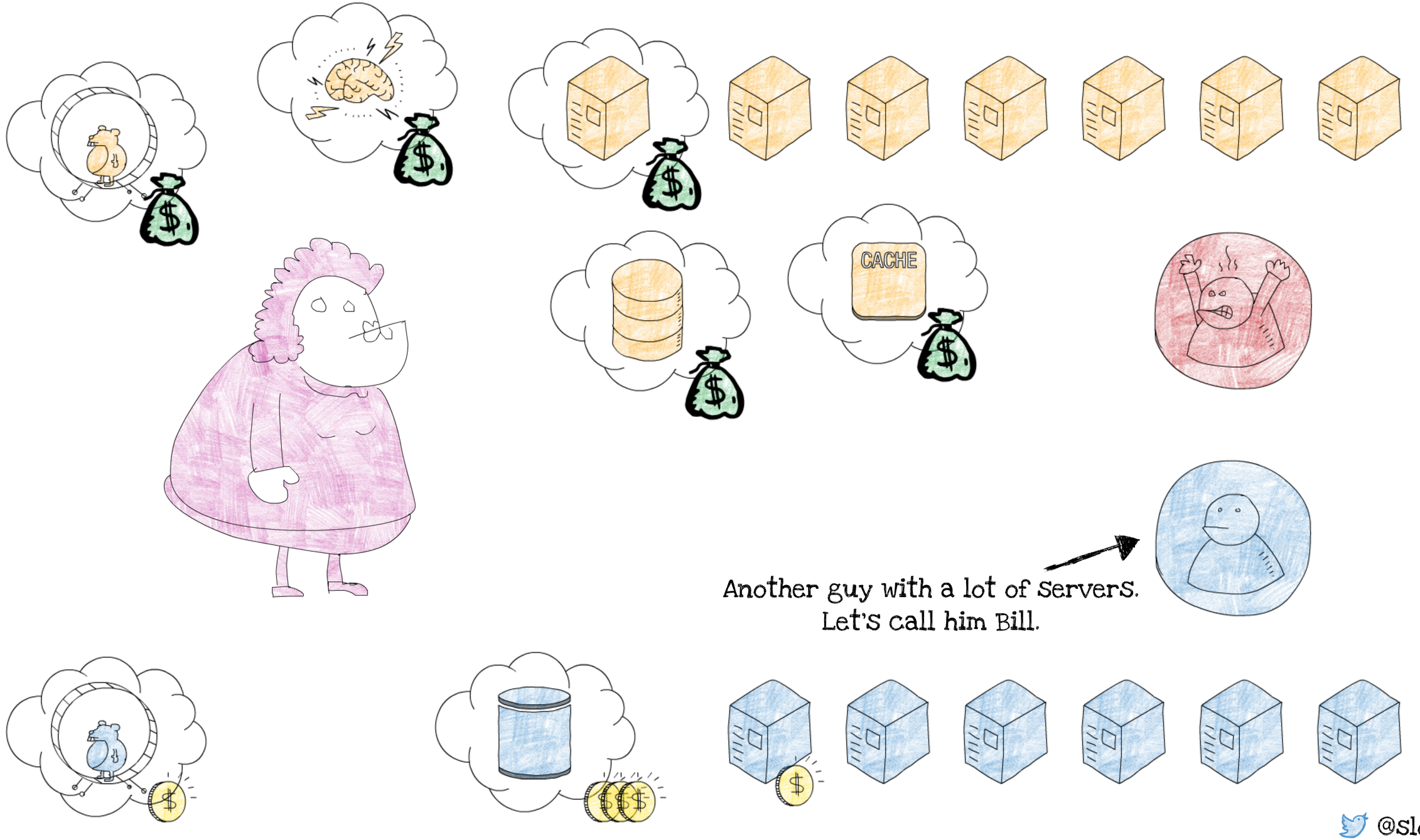

CACHE
Jeff is smart, and he knows
how do you use his servers.


CACHE
Jeff is smart, and he knows
how do you use his servers.

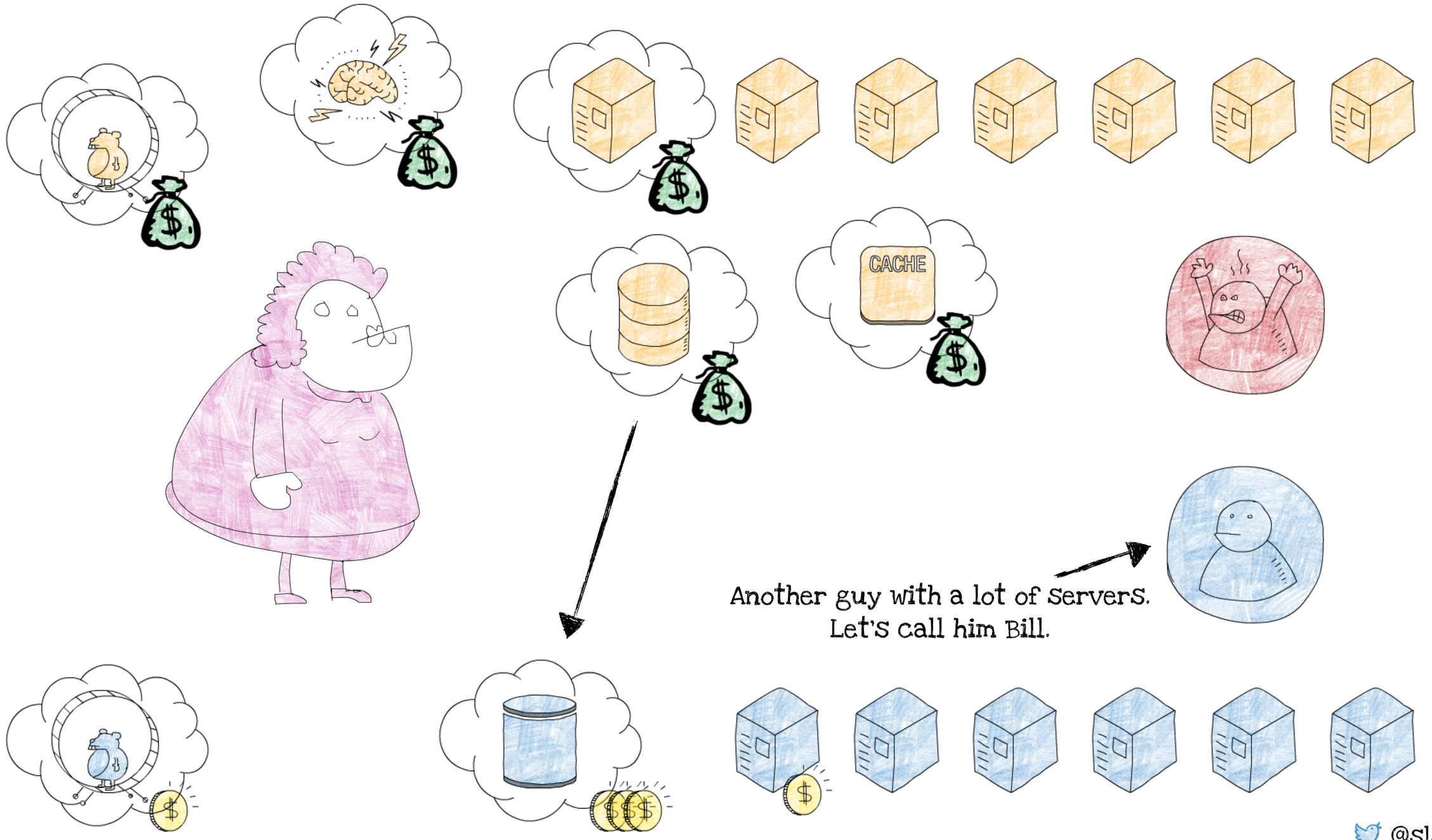

CACHE
But what if Jeff is actually a villain?

Your wallet would not be happy...

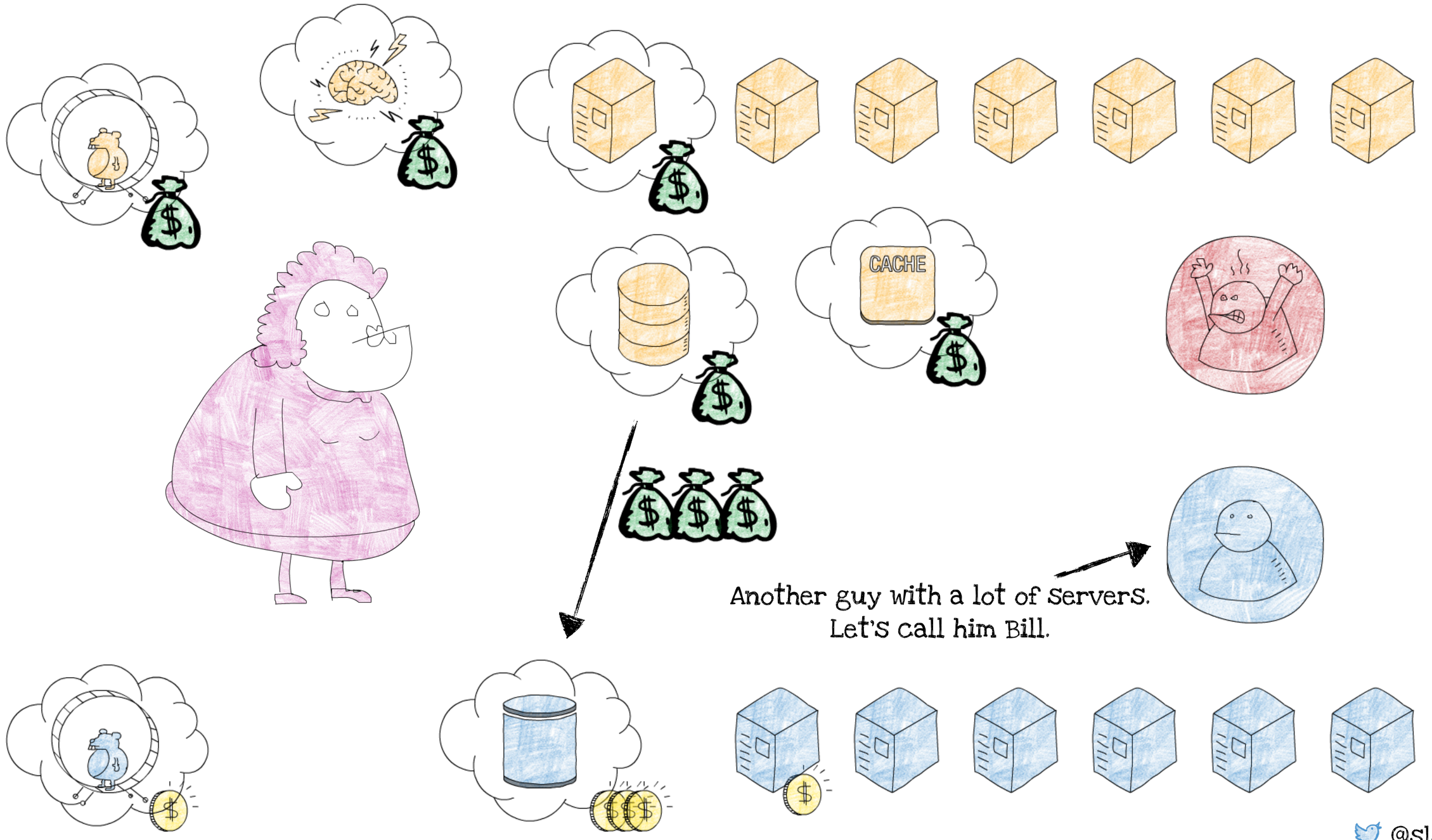




@slobodan_



@slobodan_


CACHE
Another guy with a lot of servers.
Let's call him Bill.
@slobodan_

That's vendor lock-in in the cloud

"My train of thought went like this:
the term "lock-in" is misleading.
We are really talking about switching costs."

Mark Schwartz

Enterprise Strategist at AWS

@slobodan_

"As soon as you commit yourself to
a platform or a vendor you will have
switching costs if you later decide to change."

Mark Schwartz

Enterprise Strategist at AWS

How to fight vendor lock-in?

Or, how to keep your
switching costs
reasonable?

How likely will I need to switch?
What would be the cost?

- Planning and analysis
- Good architecture
- Deployment procedures

That leads us to our topic...

Writing testable serverless apps and preventing vendor lock-in

USING HEXAGONAL ARCHITECTURE

But, before we continue...

Slobodan Stojanovic

CTO @ Cloud Horizon & CTO @ Vacation Tracker

co-author of Serverless Applications with Node.js book

AWS Serverless Hero

 @slobodan_

Writing testable serverless apps using hexagonal architecture

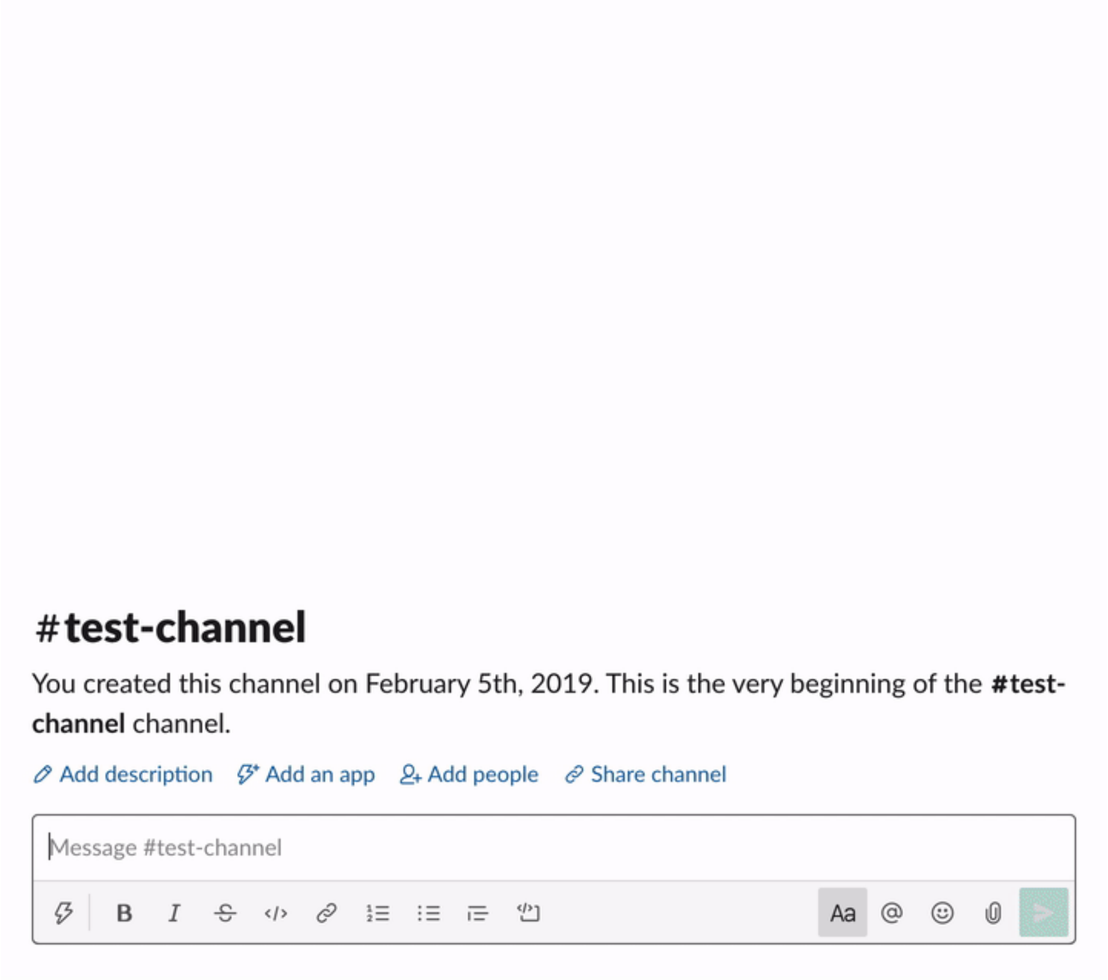
@slobodan_

Why is testing important for serverless apps?

Most of the time serverless apps
are not fully isolated monoliths
without integrations

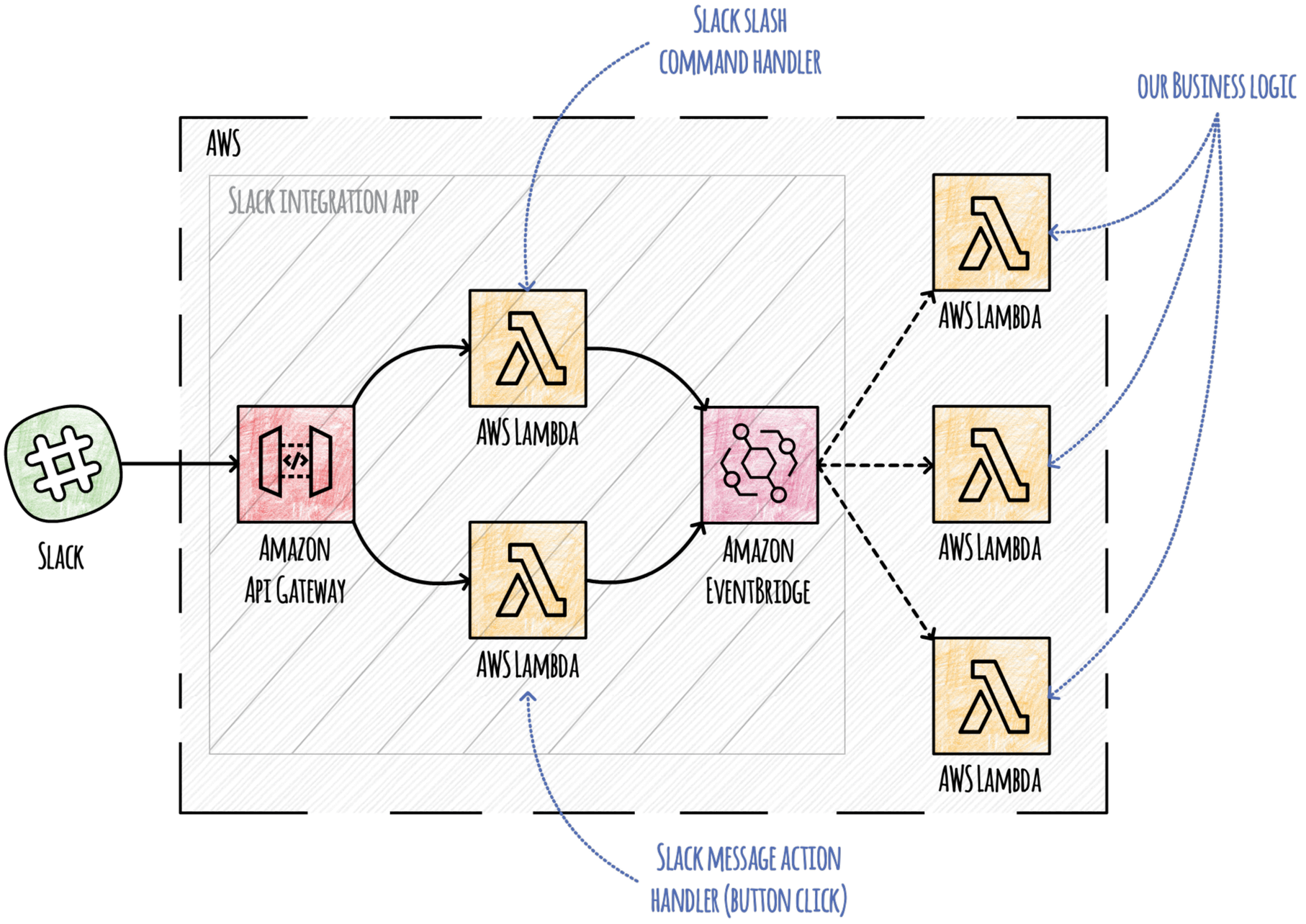
Instead, they contain many services
interacting with each other
and with external dependencies

An Example: Vacation Tracker

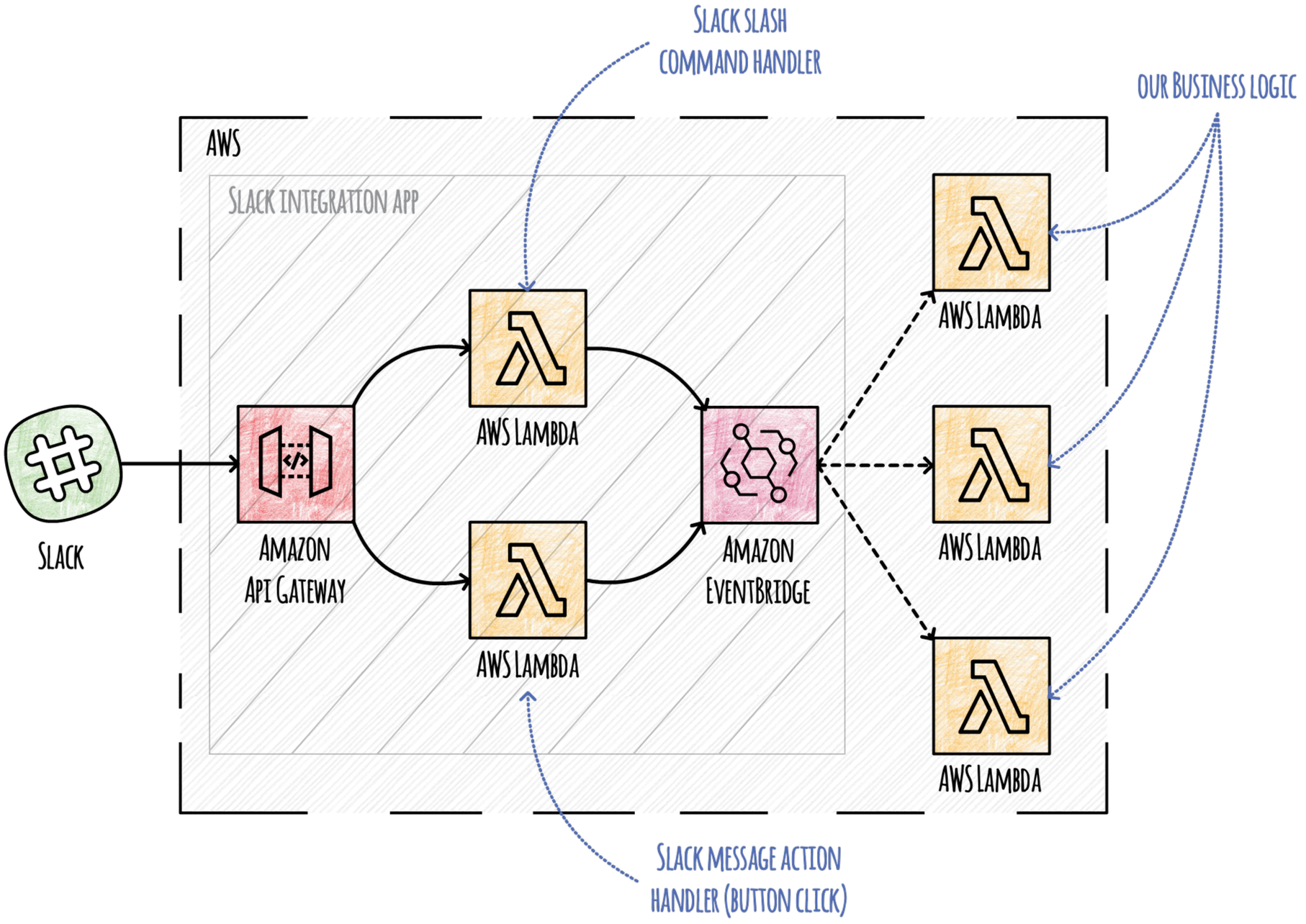


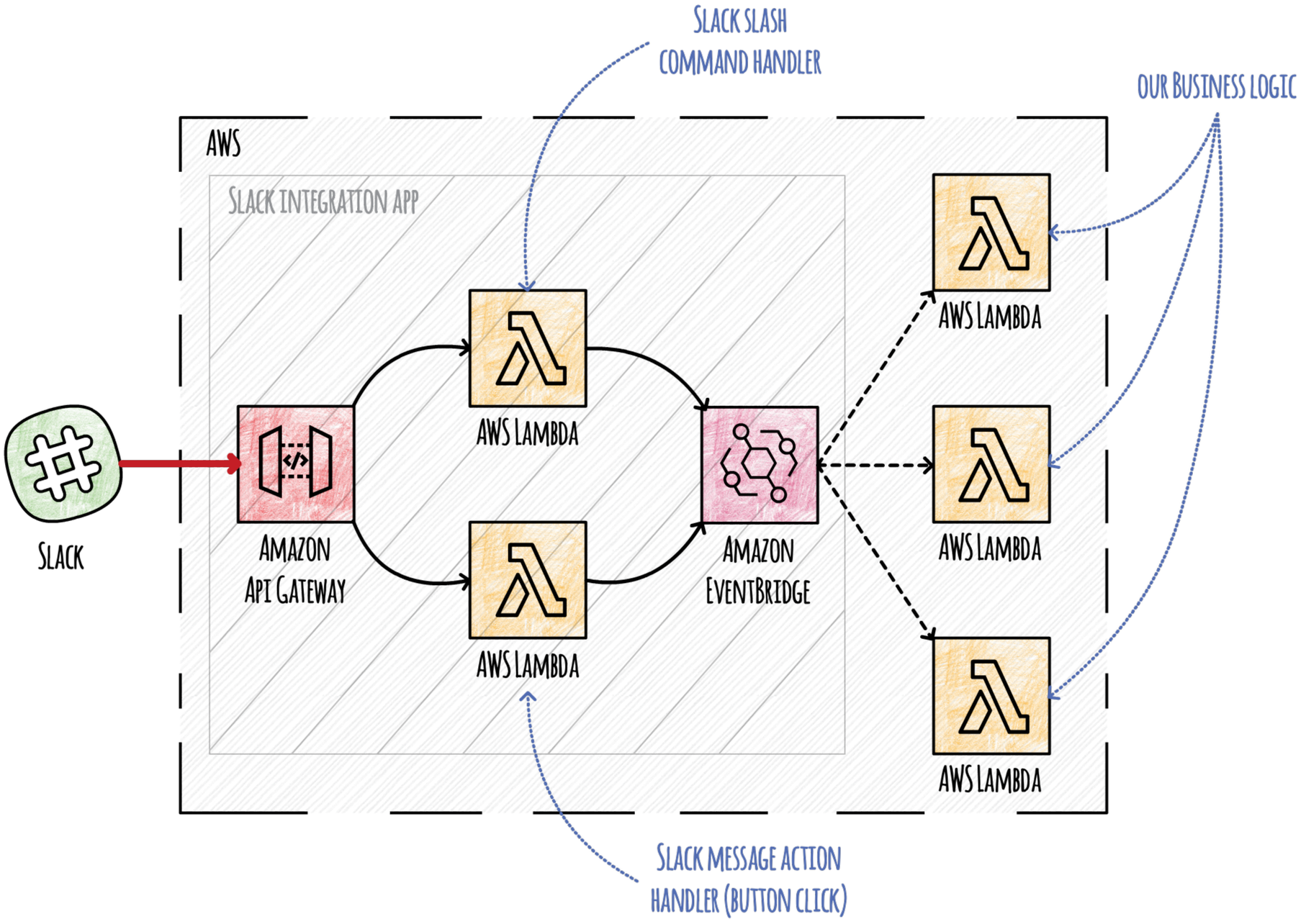
VacationTracker.io

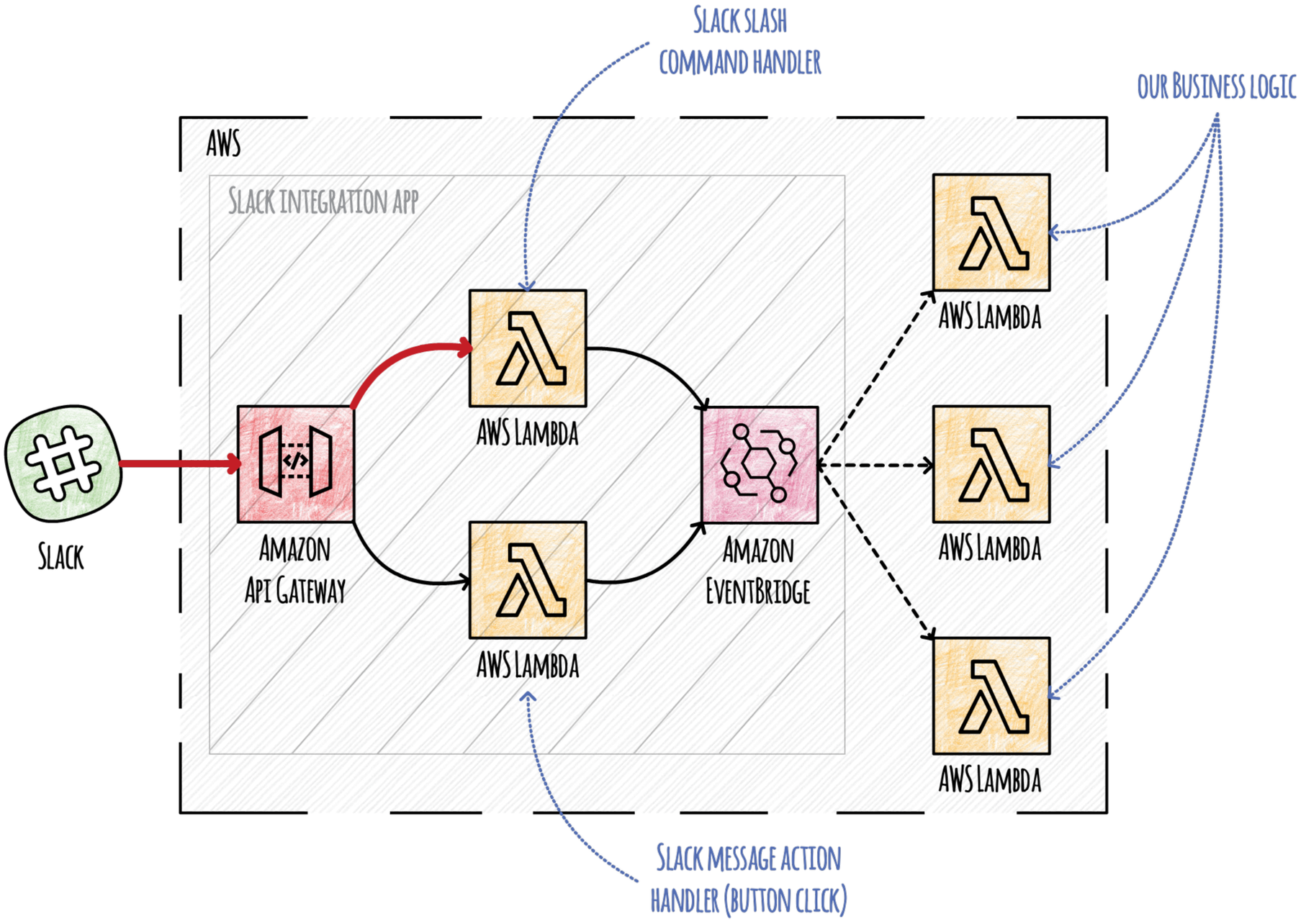
@slobodan_

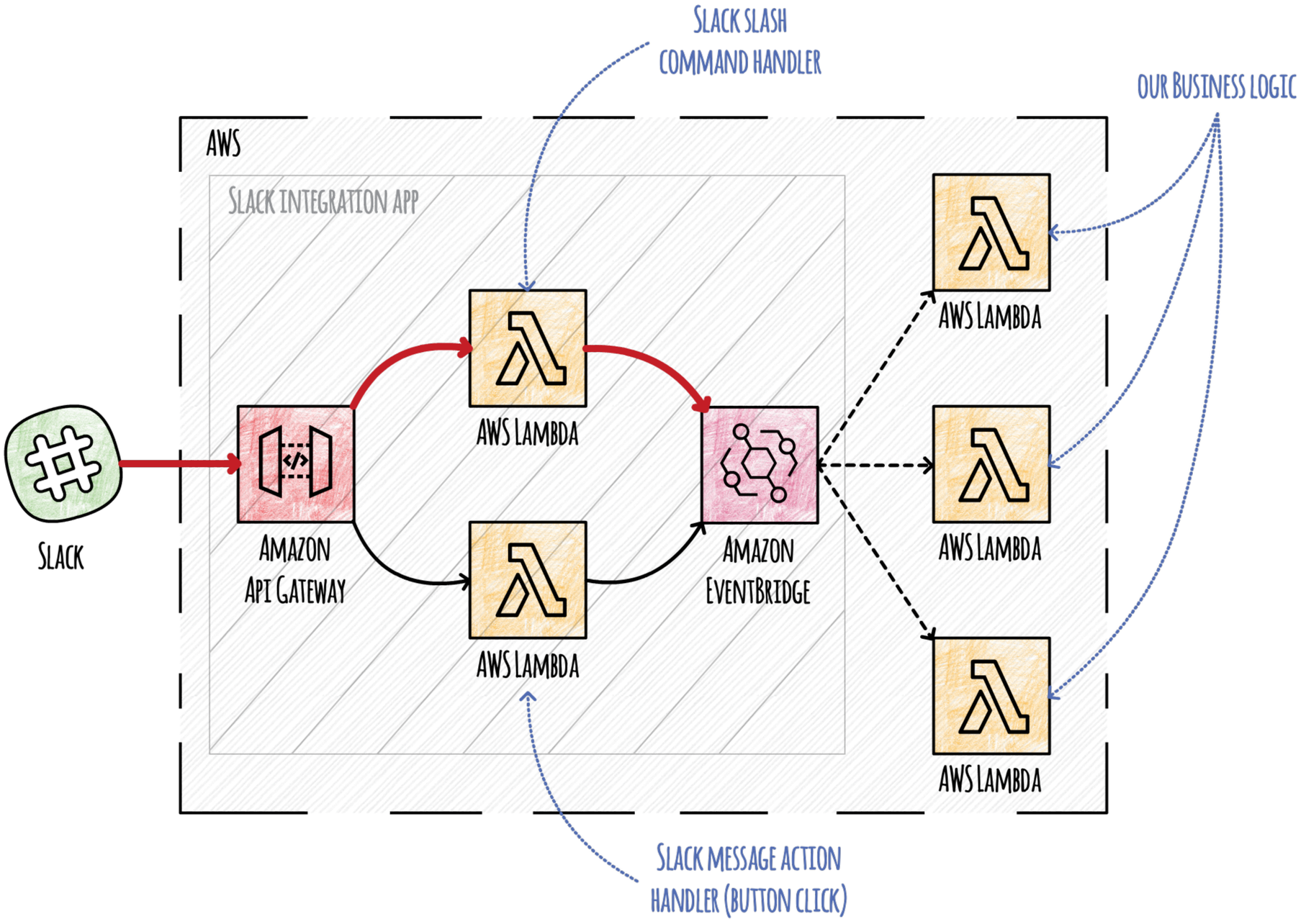

SLACK SLASH
COMMAND HANDLER
OUR BUSINESS LOGIC
AWS
SLACK INTEGRATION APP
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
SLACK
AMAZON
API GATEWAY
AMAZON
EVENTBRIDGE
SLACK MESSAGE ACTION
HANDLER (BUTTON CLICK)

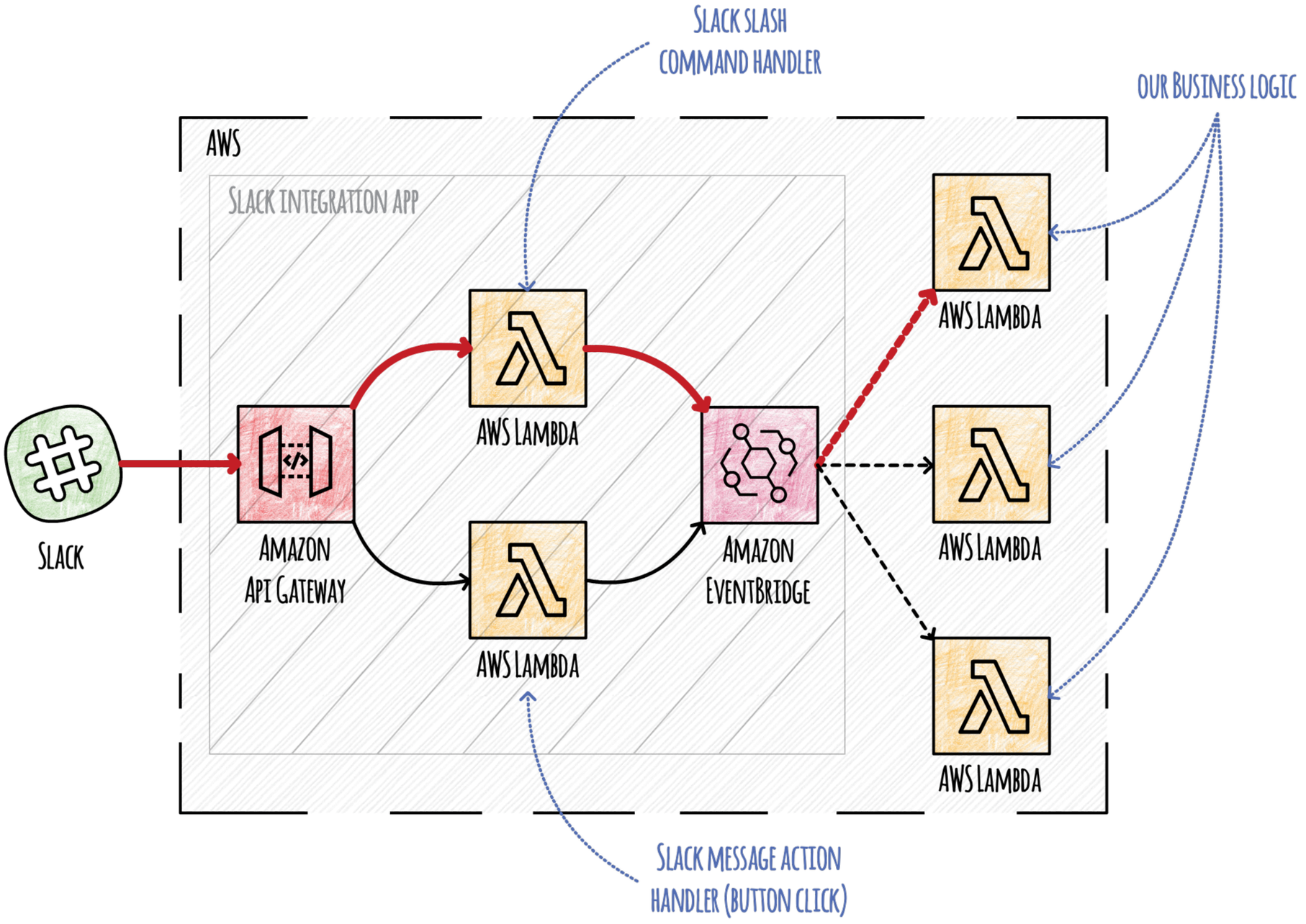
Integrations can change or fail every moment!


SLACK SLASH
COMMAND HANDLER
OUR BUSINESS LOGIC
AWS
SLACK INTEGRATION APP
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
SLACK
AMAZON
API GATEWAY
AMAZON
EVENTBRIDGE
SLACK MESSAGE ACTION
HANDLER (BUTTON CLICK)


SLACK SLASH
COMMAND HANDLER
OUR BUSINESS LOGIC
AWS
SLACK INTEGRATION APP
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
SLACK
AMAZON
API GATEWAY
AMAZON
EVENTBRIDGE
SLACK MESSAGE ACTION
HANDLER (BUTTON CLICK)


SLACK SLASH
COMMAND HANDLER
OUR BUSINESS LOGIC
AWS
SLACK INTEGRATION APP
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
SLACK
AMAZON
API GATEWAY
AMAZON
EVENTBRIDGE
SLACK MESSAGE ACTION
HANDLER (BUTTON CLICK)


SLACK SLASH
COMMAND HANDLER
OUR BUSINESS LOGIC
AWS
SLACK INTEGRATION APP
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
SLACK
AMAZON
API GATEWAY
AMAZON
EVENTBRIDGE
SLACK MESSAGE ACTION
HANDLER (BUTTON CLICK)


SLACK SLASH
COMMAND HANDLER
OUR BUSINESS LOGIC
AWS
SLACK INTEGRATION APP
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
SLACK
AMAZON
API GATEWAY
AMAZON
EVENTBRIDGE
SLACK MESSAGE ACTION
HANDLER (BUTTON CLICK)

Tests don't prevent changes.
They make sure your changes
are not accidental.

But how do we prevent changes?

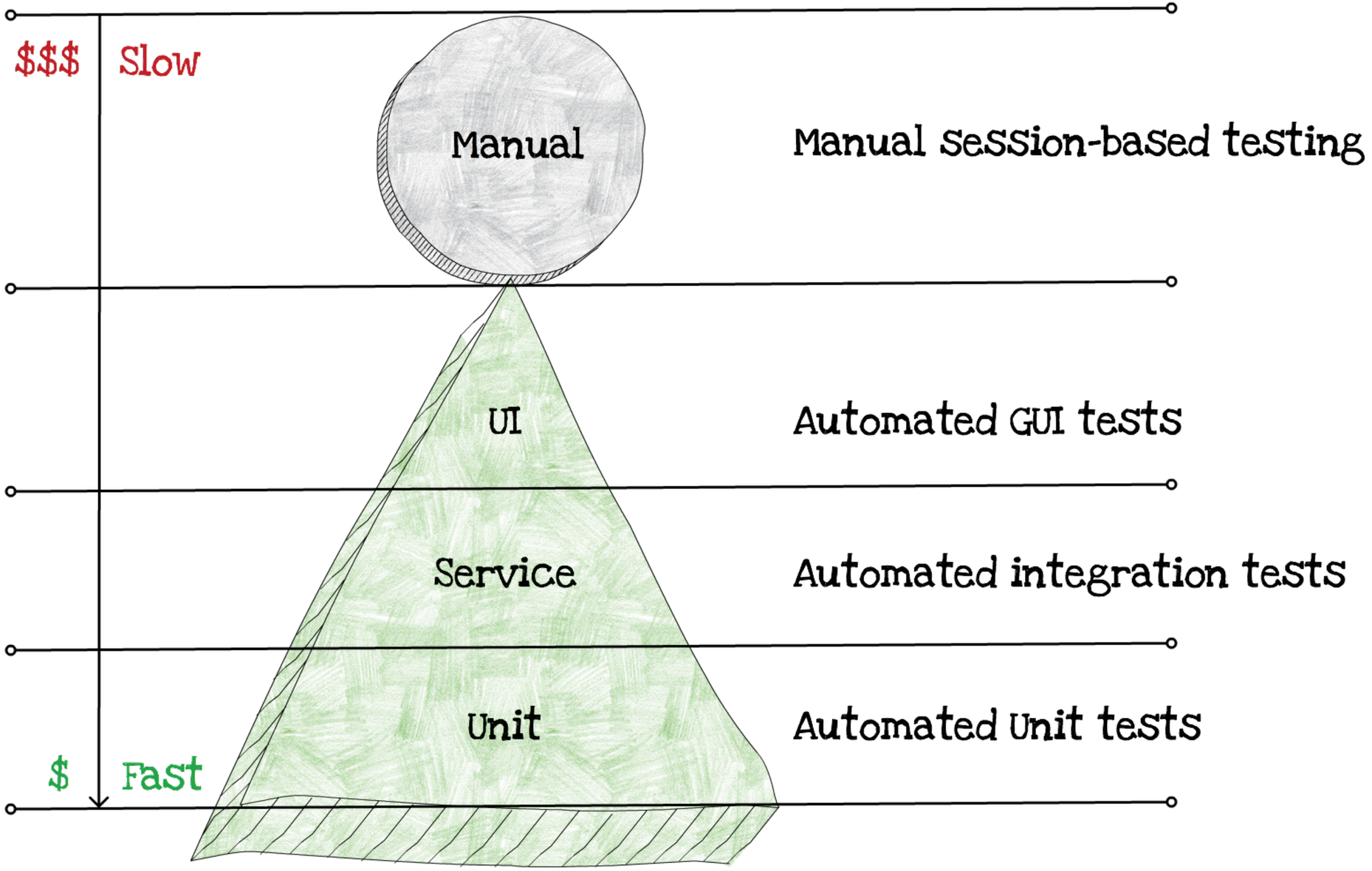
But we'll discuss that later today.

We can't.

Our app needs to adapt fast!

But how do you know
what should you test in a serverless app?

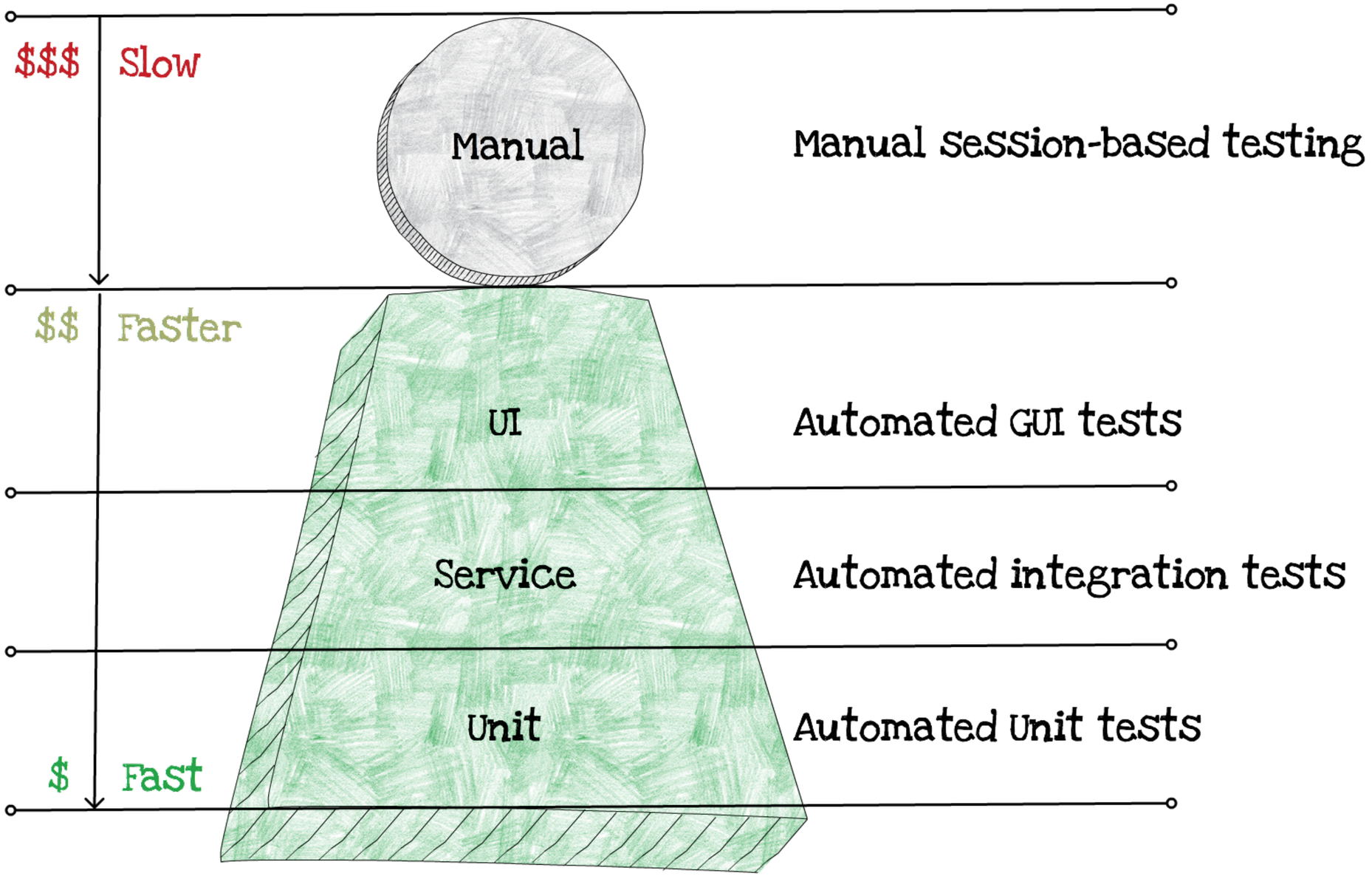
Testing pyramid


$$$
Slow
Manual
Manual session-based testing
UI
Automated GUI tests
Service
Automated integration tests
Unit
Automated Unit tests
$
Fast

Testing pyramid

vs

"Serverless testing pyramid"


$$$
Slow
Manual
Manual session-based testing
$$
Faster
UI
Automated GUI tests
Service
Automated integration tests
Unit
Automated Unit tests
$
Fast

Integration tests are cheaper,
but also more important,
because the common serverless app
is split into many small pieces

Writing testable serverless apps using hexagonal architecture

@slobodan_

Ok, so which architecture is the best for serverless apps?

Any architecture that
will let you test your serverless app easily
and keep switching costs low.

Because sooner or later you'll need to switch/migrate pieces of your app.

Not to another cloud vendor,
but to your new service,
new or changed integration...

Risks to consider when building a serverless app

- CONFIGURATION RISKS
- TECHNICAL WORKFLOW RISKS
- BUSINESS LOGIC RISKS
- INTEGRATION RISKS

One of the architectures that fits these needs is

Hexagonal Architecture

or

Ports and Adapters

Writing testable serverless apps using hexagonal architecture

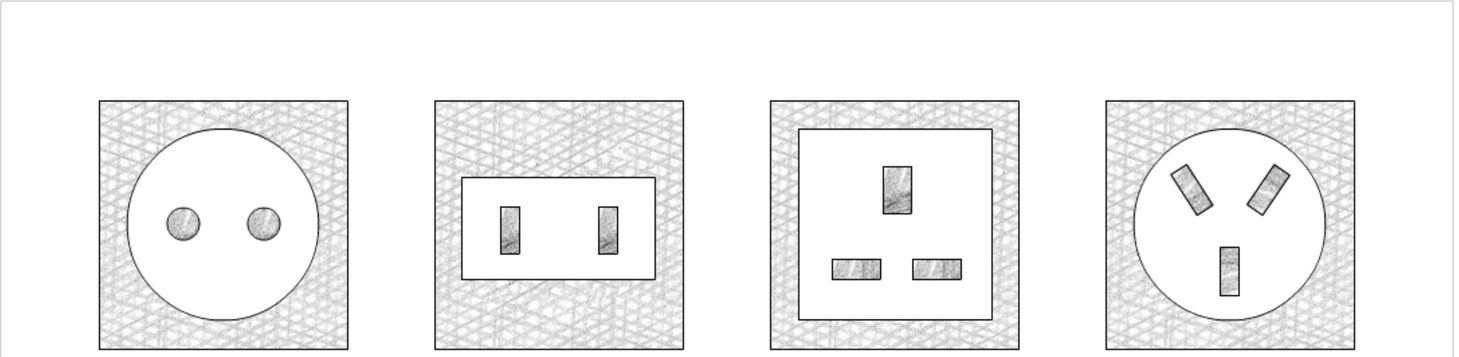
@Slobodan_

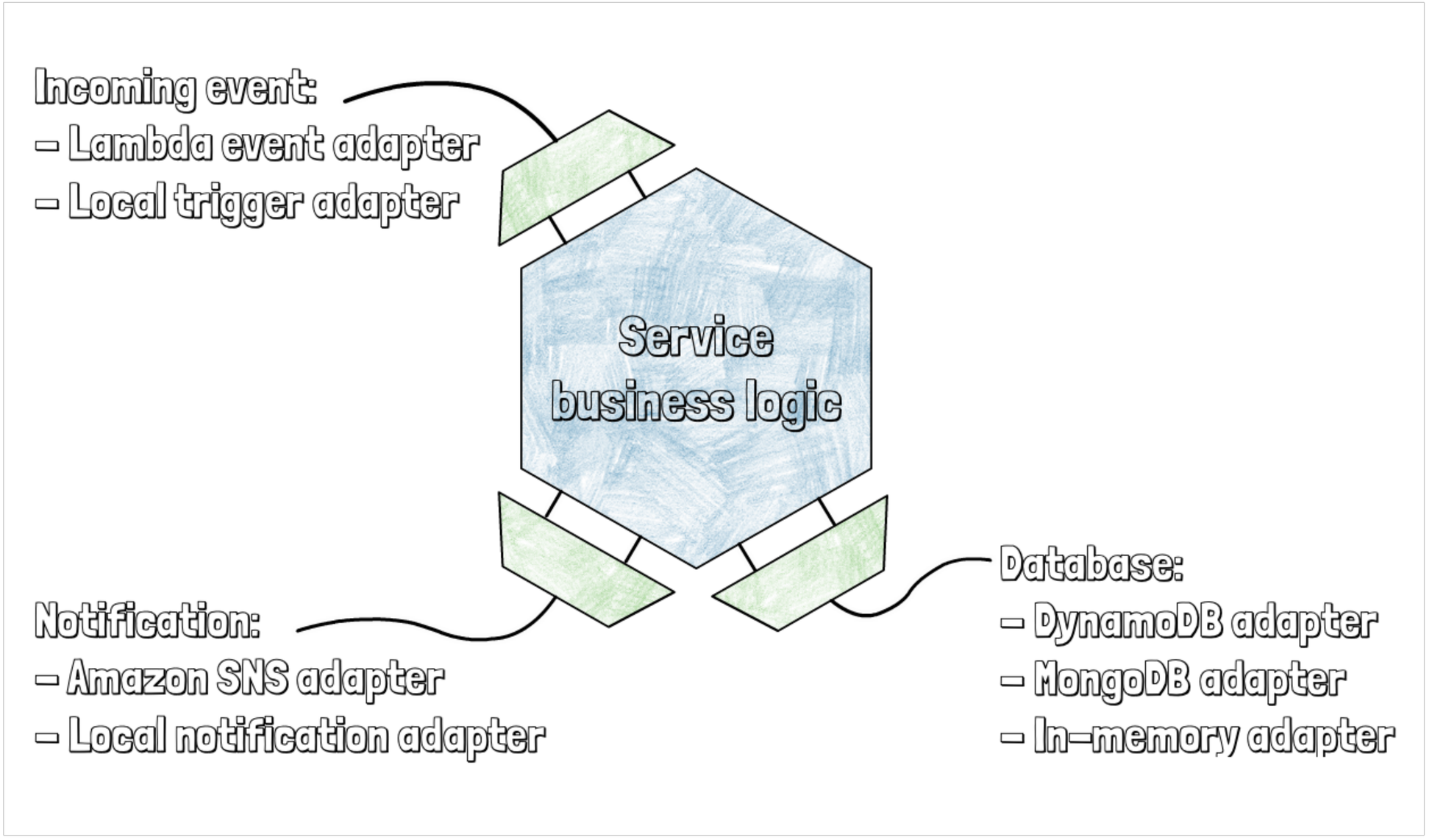
"Allow an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases."

Alistair Cockburn

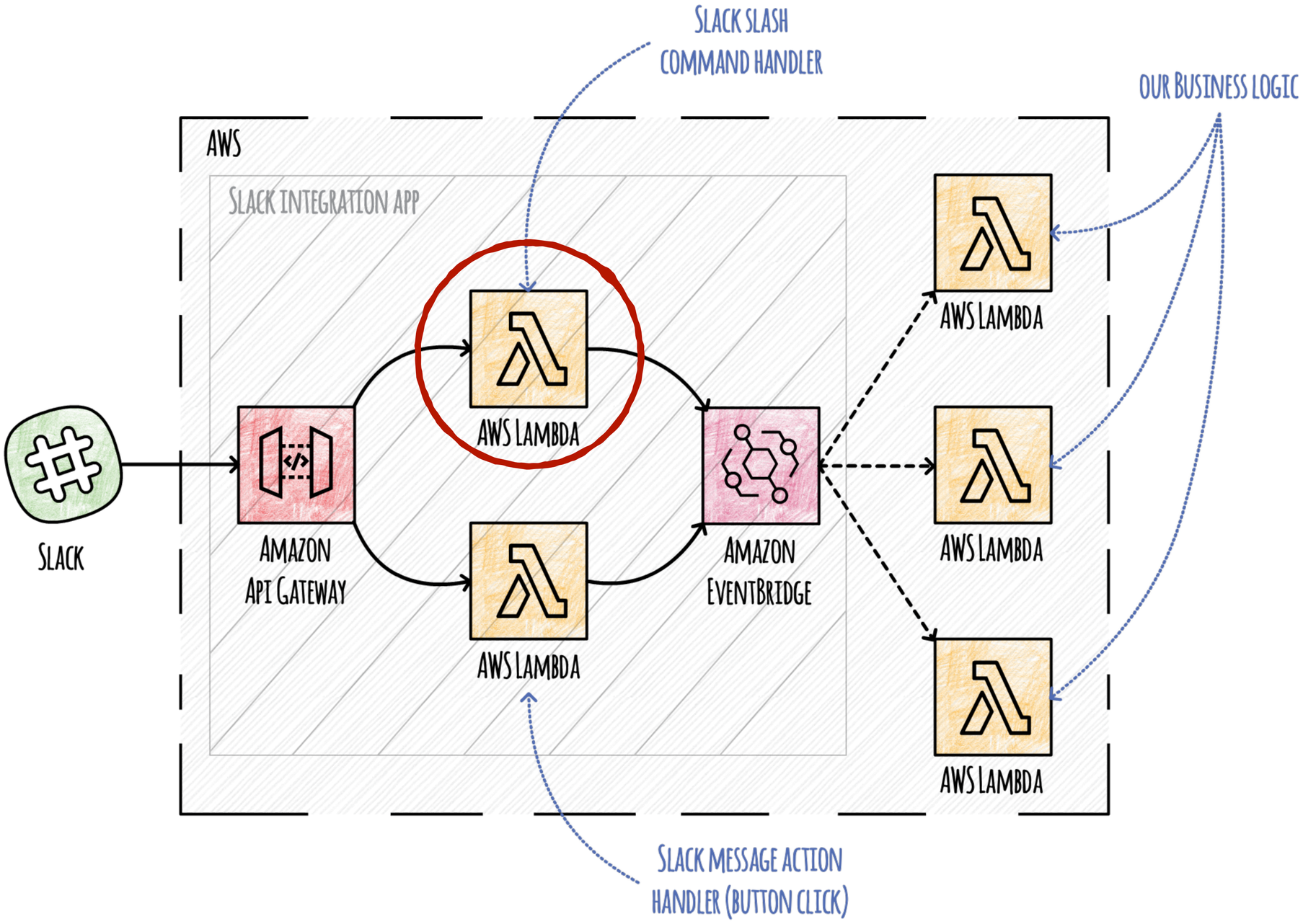
Creator of Hexagonal architecture

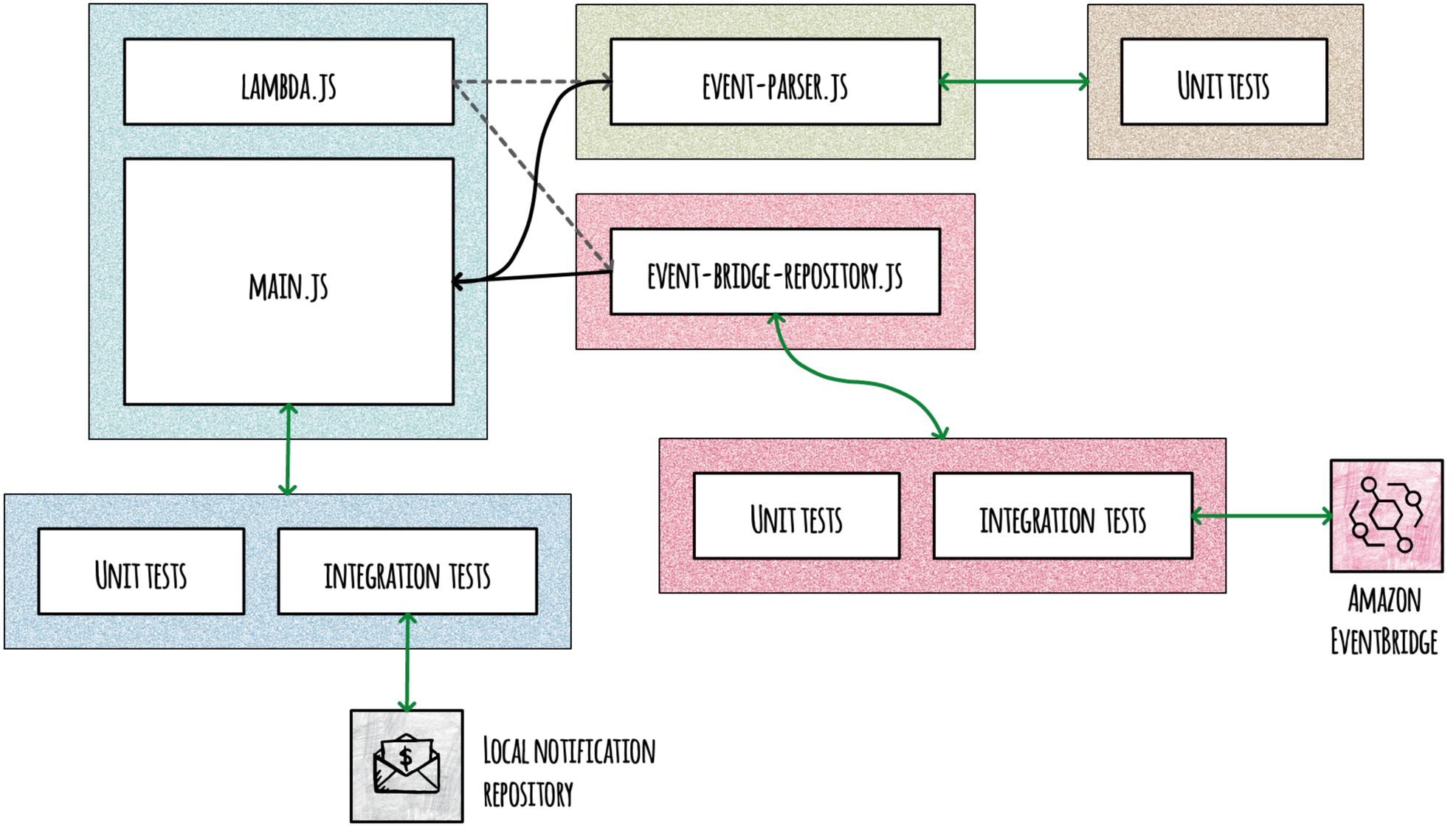
@slobodan_

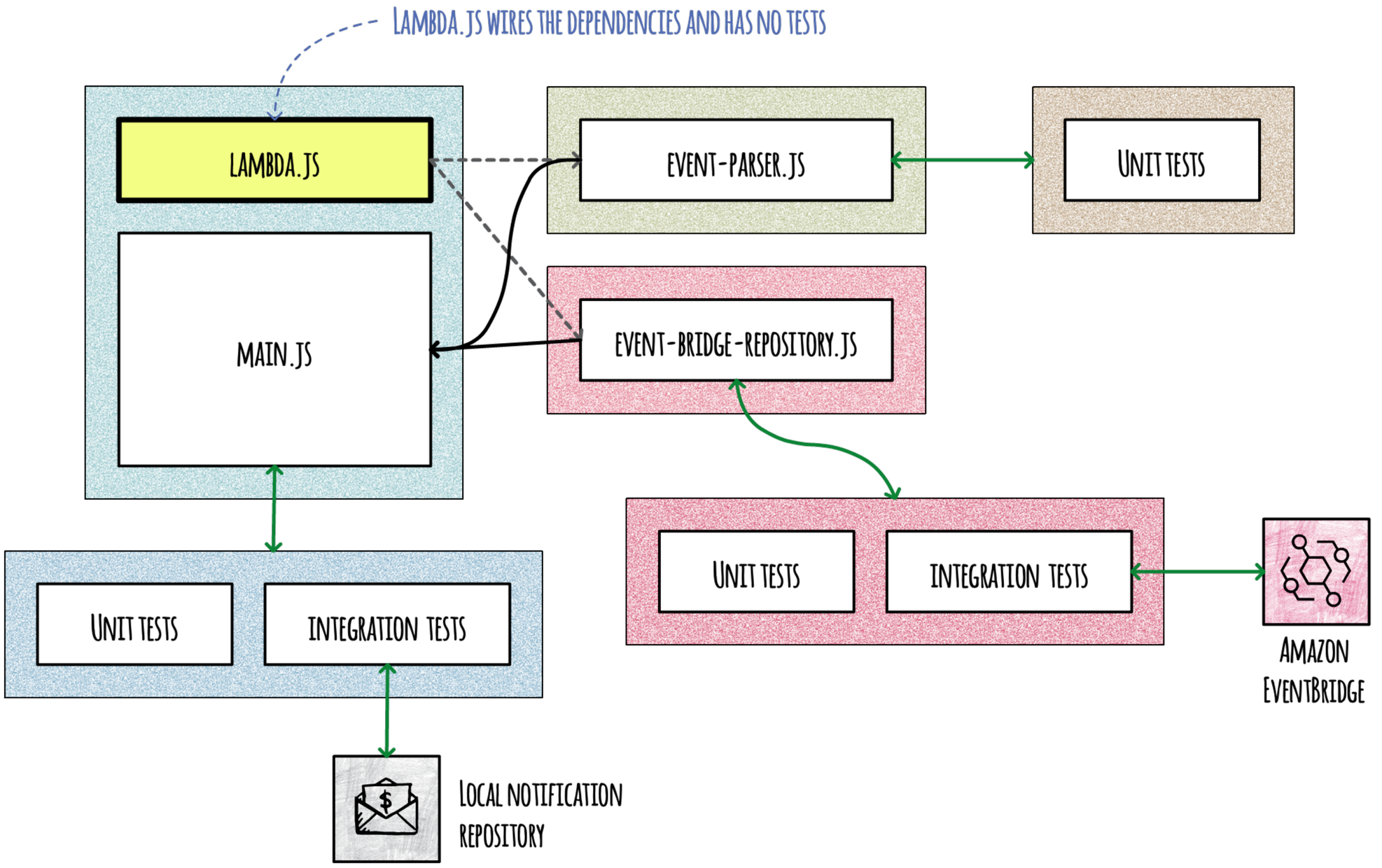


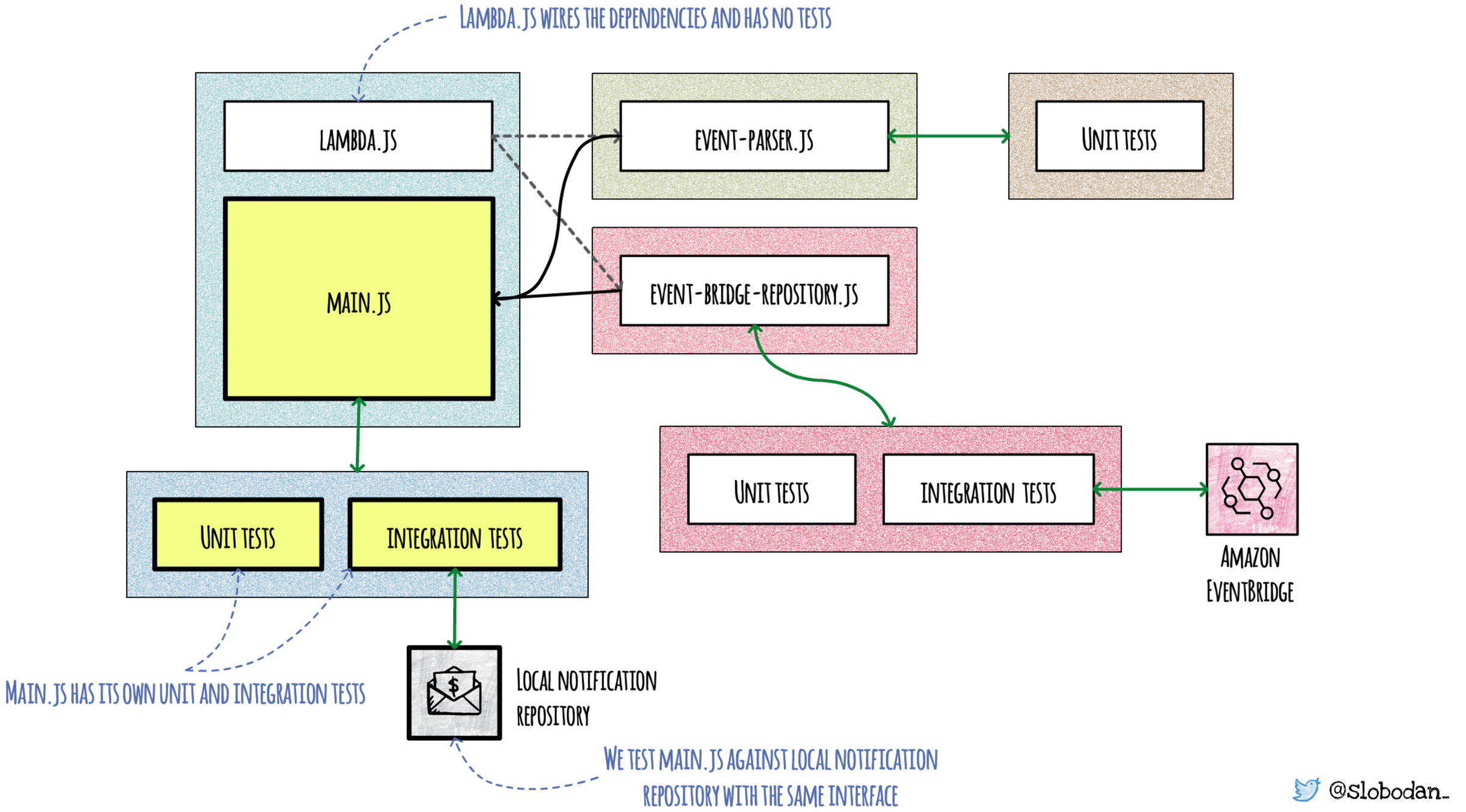

Incoming event:
– Lambda event adapter
– Local trigger adapter
Service
business logic
Notification:
– Amazon SNS adapter
– Local notification adapter
Database:
– DynamoDB adapter
– MongoDB adapter
– In-memory adapter

Let's go back to Vacation Tracker for an example

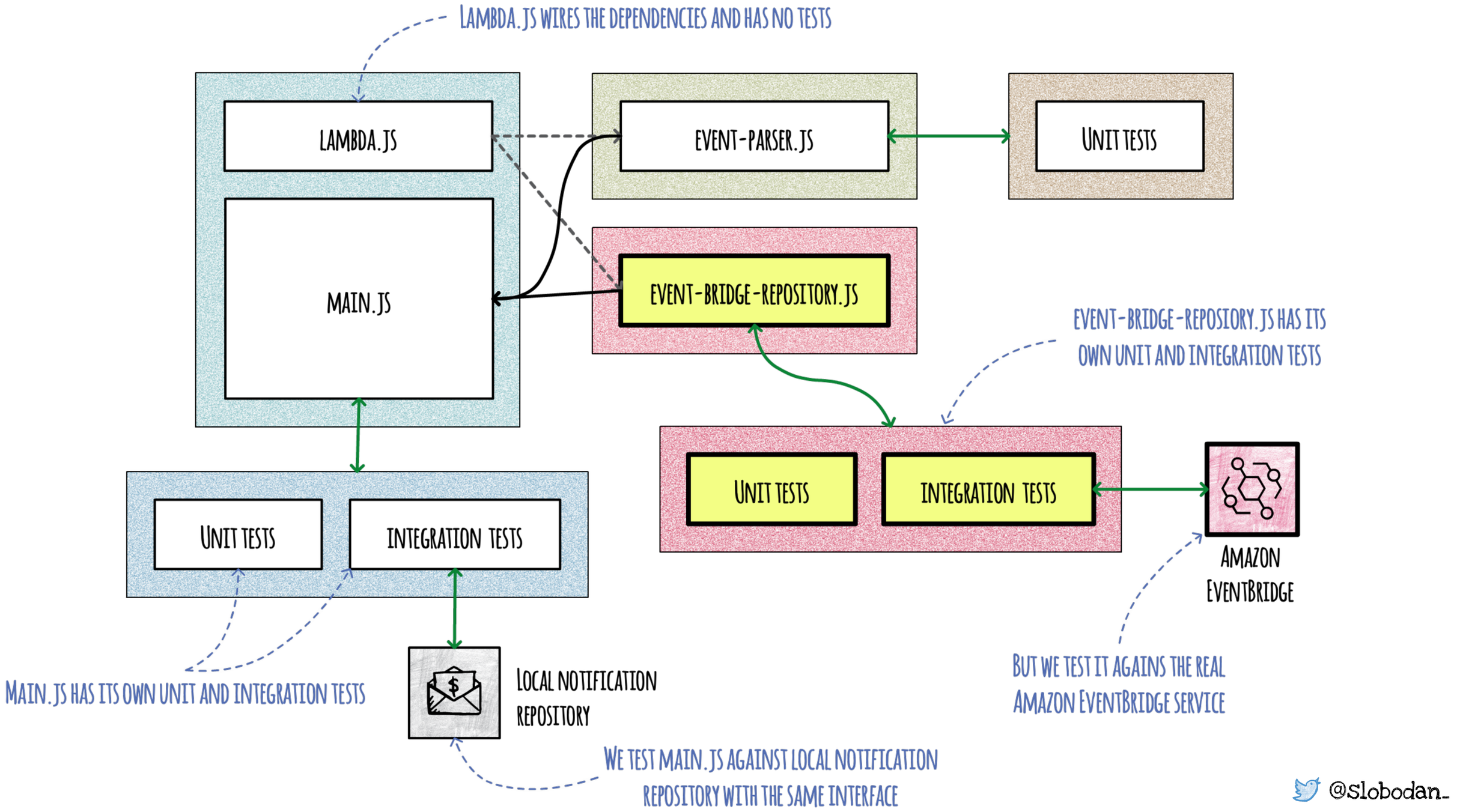

SLACK SLASH
COMMAND HANDLER
OUR BUSINESS LOGIC
AWS
SLACK INTEGRATION APP
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
AWS LAMBDA
SLACK
AMAZON
API GATEWAY
AMAZON
EVENTBRIDGE
SLACK MESSAGE ACTION
HANDLER (BUTTON CLICK)


LAMBDA.JS
MAIN.JS
EVENT-PARSER.JS
UNIT TESTS
EVENT-BRIDGE-REPOSITORY.JS
UNIT TESTS
INTEGRATION TESTS
UNIT TESTS
INTEGRATION TESTS
AMAZON EVENTBRIDGE
LOCAL NOTIFICATION REPOSITORY

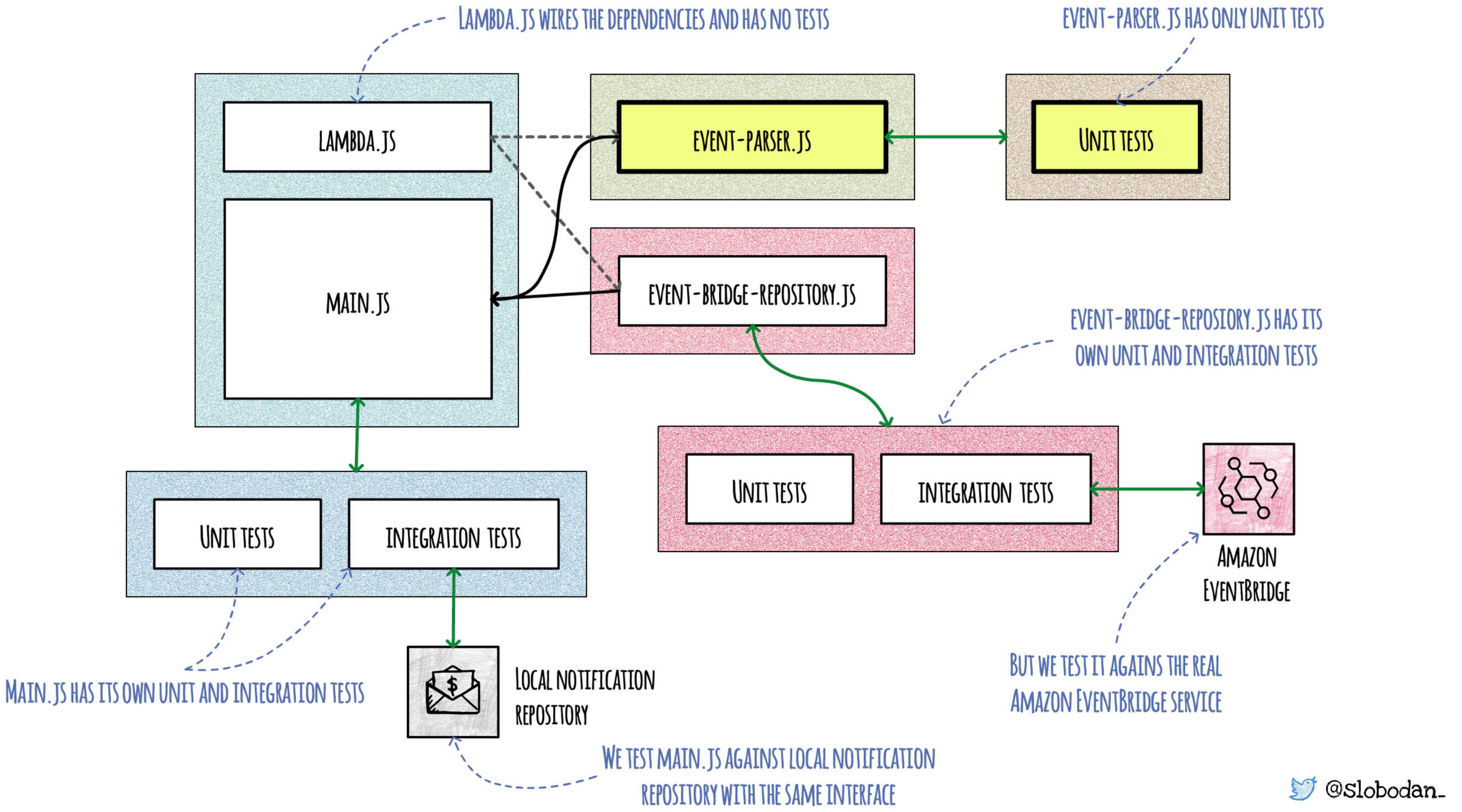

LAMBDA.JS WIRES THE DEPENDENCIES AND HAS NO TESTS
LAMBDA.JS
MAIN.JS
EVENT-PARSER.JS
UNIT TESTS
EVENT-BRIDGE-REPOSITORY.JS
UNIT TESTS
INTEGRATION TESTS
AMAZON EVENTBRIDGE
UNIT TESTS
INTEGRATION TESTS
LOCAL NOTIFICATION REPOSITORY


LAMBDA.JS WIRES THE DEPENDENCIES AND HAS NO TESTS
LAMBDA.JS
MAIN.JS
EVENT-PARSER.JS
UNIT TESTS
EVENT-BRIDGE-REPOSITORY.JS
UNIT TESTS
INTEGRATION TESTS
UNIT TESTS
INTEGRATION TESTS
AMAZON EVENTBRIDGE
MAIN.JS HAS ITS OWN UNIT AND INTEGRATION TESTS
LOCAL NOTIFICATION REPOSITORY
WE TEST MAIN.JS AGAINST LOCAL NOTIFICATION REPOSITORY WITH THE SAME INTERFACE

@slobodan_


LAMBDA.JS WIRES THE DEPENDENCIES AND HAS NO TESTS
LAMBDA.JS
MAIN.JS
EVENT-PARSER.JS
UNIT TESTS
EVENT-BRIDGE-REPOSITORY.JS
EVENT-BRIDGE-REPOSIORY.JS HAS ITS OWN UNIT AND INTEGRATION TESTS
UNIT TESTS
INTEGRATION TESTS
UNIT TESTS
INTEGRATION TESTS
AMAZON EVENTBRIDGE
MAIN.JS HAS ITS OWN UNIT AND INTEGRATION TESTS
LOCAL NOTIFICATION REPOSITORY
BUT WE TEST IT AGAINS THE REAL AMAZON EVENTBRIDGE SERVICE
WE TEST MAIN.JS AGAINST LOCAL NOTIFICATION REPOSITORY WITH THE SAME INTERFACE

@slobodan_


LAMBDA.JS WIRES THE DEPENDENCIES AND HAS NO TESTS
EVENT-PARSER.JS HAS ONLY UNIT TESTS
LAMBDA.JS
MAIN.JS
EVENT-PARSER.JS
UNIT TESTS
EVENT-BRIDGE-REPOSITORY.JS
EVENT-BRIDGE-REPOSIORY.JS HAS ITS OWN UNIT AND INTEGRATION TESTS
UNIT TESTS
INTEGRATION TESTS
UNIT TESTS
INTEGRATION TESTS
AMAZON EVENTBRIDGE
MAIN.JS HAS ITS OWN UNIT AND INTEGRATION TESTS
LOCAL NOTIFICATION REPOSITORY
BUT WE TEST IT AGAINS THE REAL AMAZON EVENTBRIDGE SERVICE
WE TEST MAIN.JS AGAINST LOCAL NOTIFICATION REPOSITORY WITH THE SAME INTERFACE

@slobodan_

Code, please!

```
const {
  httpResponse,
  parseApiEvent,
  EventBridgeRepository
} = require('../common')
const main = require('./main')

export async function handler(event) {
  // Create instance of SNS notification repository
  const notification = new EventBridgeRepository(
    process.env.topic
  )
  // Invoke main function with all dependencies
  await main(event, parseApiEvent, notification)
  return httpResponse()
}
```

```
await main(event, parseApiEvent, notification)
```

Unit tests

Some static values

Parser mock

```
await main(event, parseApiEvent, notification)
```

Mock notification repository instance

Integration tests

Some static values

Parser function

```
await main(event, parseApiEvent, notification)
```

Local notification adapter, using JS events for example

```
await main(event, parseApiEvent, notification)
```

EventBridge notification adapter
has its own integration tests

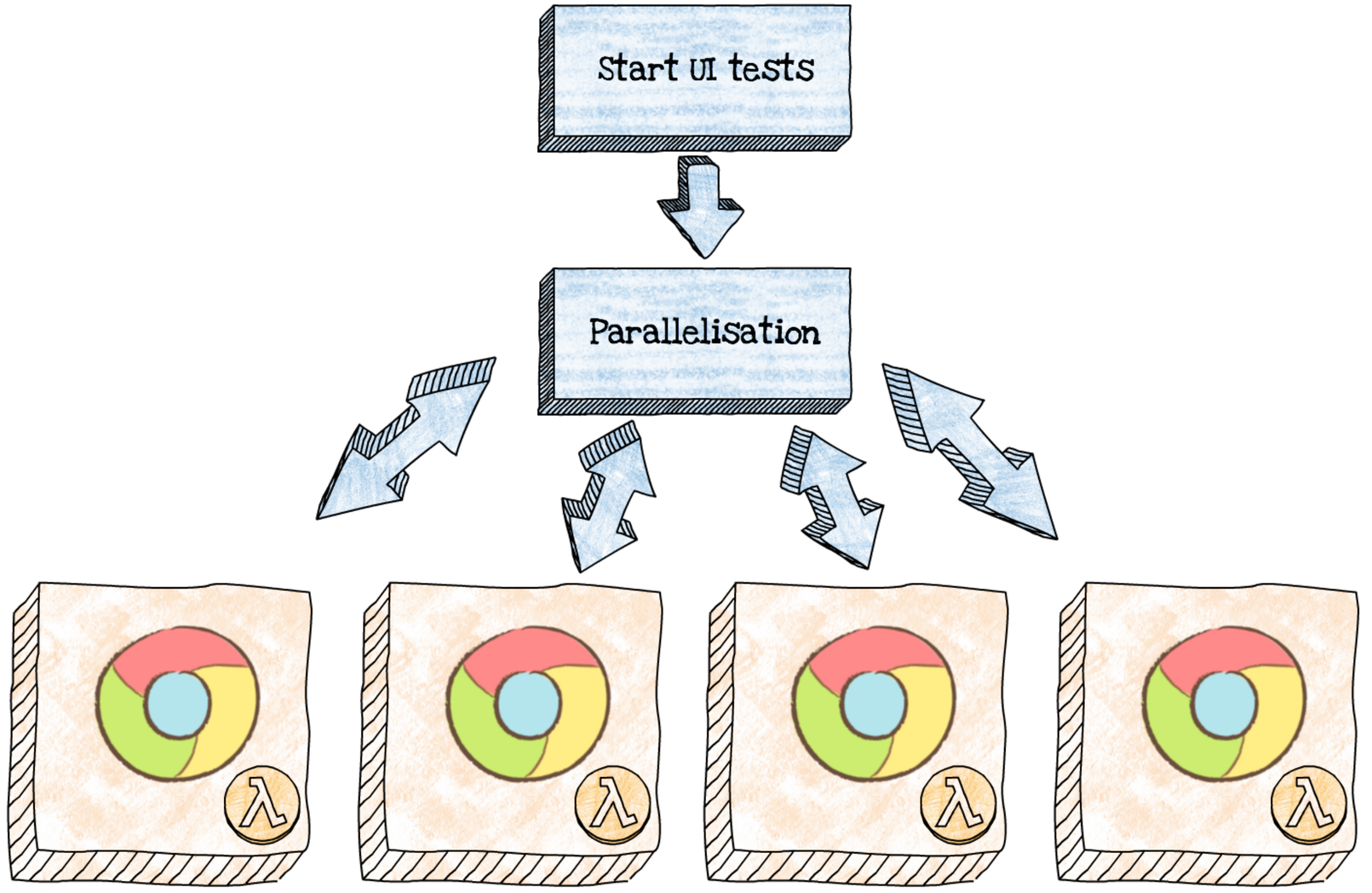
Simple and nice

What about End-to-End and UI tests?

Serverless is mainly
a back end thing,
but it can help with UI tests!

UI TESTS ARE
SLOW AND EXPENSIVE

Benefits of serverless are
cheap infrastructure
and easy/fast parallelization


Start UI tests
Parallelisation
λ
λ
λ
λ

Or, you can use your favorite tool,
such as cypress.io

But, do you remember...


BIG


BAD

VENDOR LOCK-IN

How does hexagonal architecture help you fighting vendor lock-in?

How does hexagonal architecture help you to keep switching costs reasonable?

Story time

Vacation Tracker

VacationTracker.io

- Serverless prototype
- Small team (1 fulltime developer)
- Initial product was Serverless chatbot + Express.js and MongoDB
- Growing fast (200+ teams using it)

+ A FEW BAD DECISIONS AS A BONUS :)

We did a few migrations in past few months.

For example:

- Express API -> Serverless API migration
- MongoDB -> DynamoDB migration

Let's talk about
MongoDB -> DynamoDB
switch

We defined an interface for our MongoDB Repository.

For example, this:

```
const db = new MongoDbRepository(something)
const user = db.getUser(userId)
```

Returns a single user with its properties.

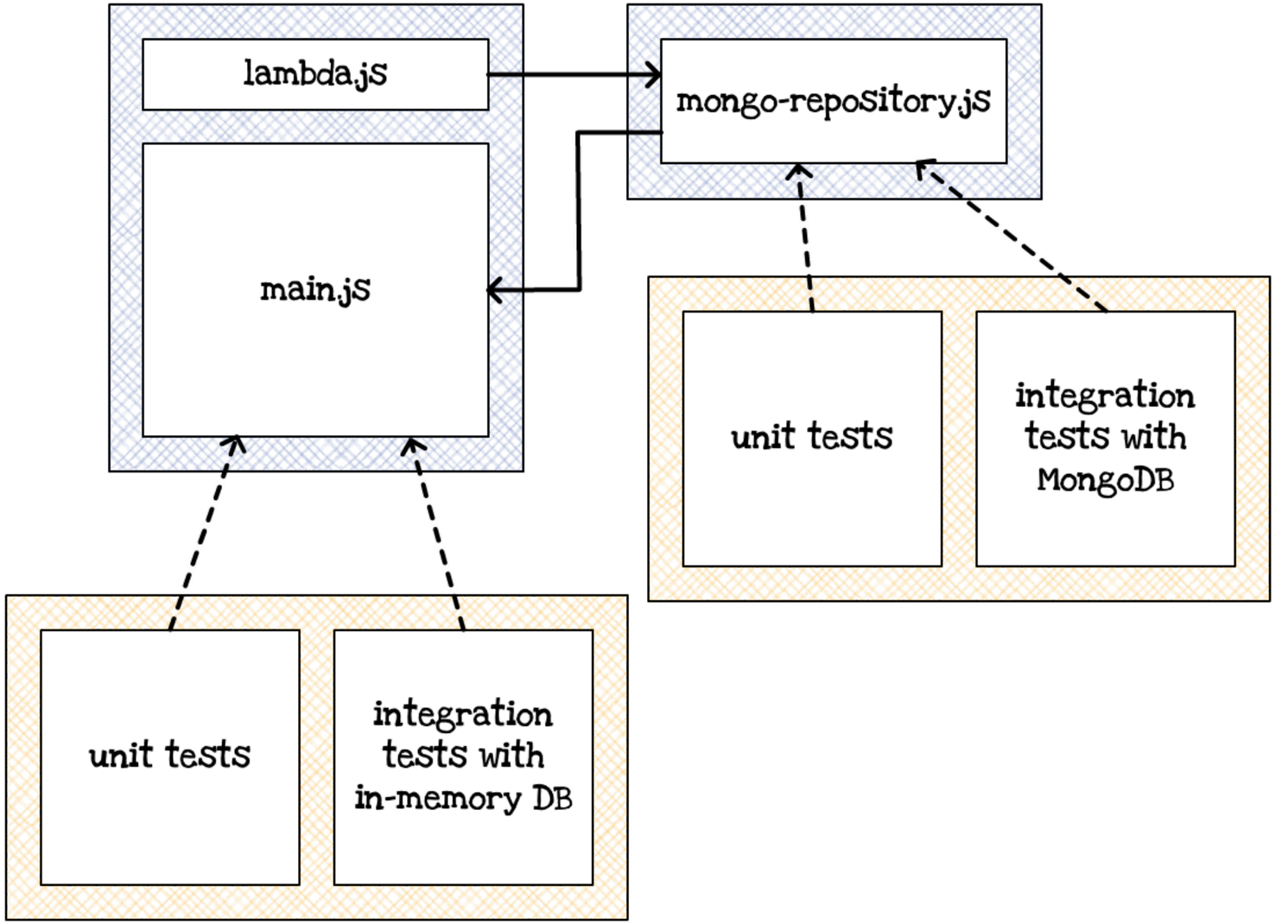
We created DynamoDB Repository with the same interface.

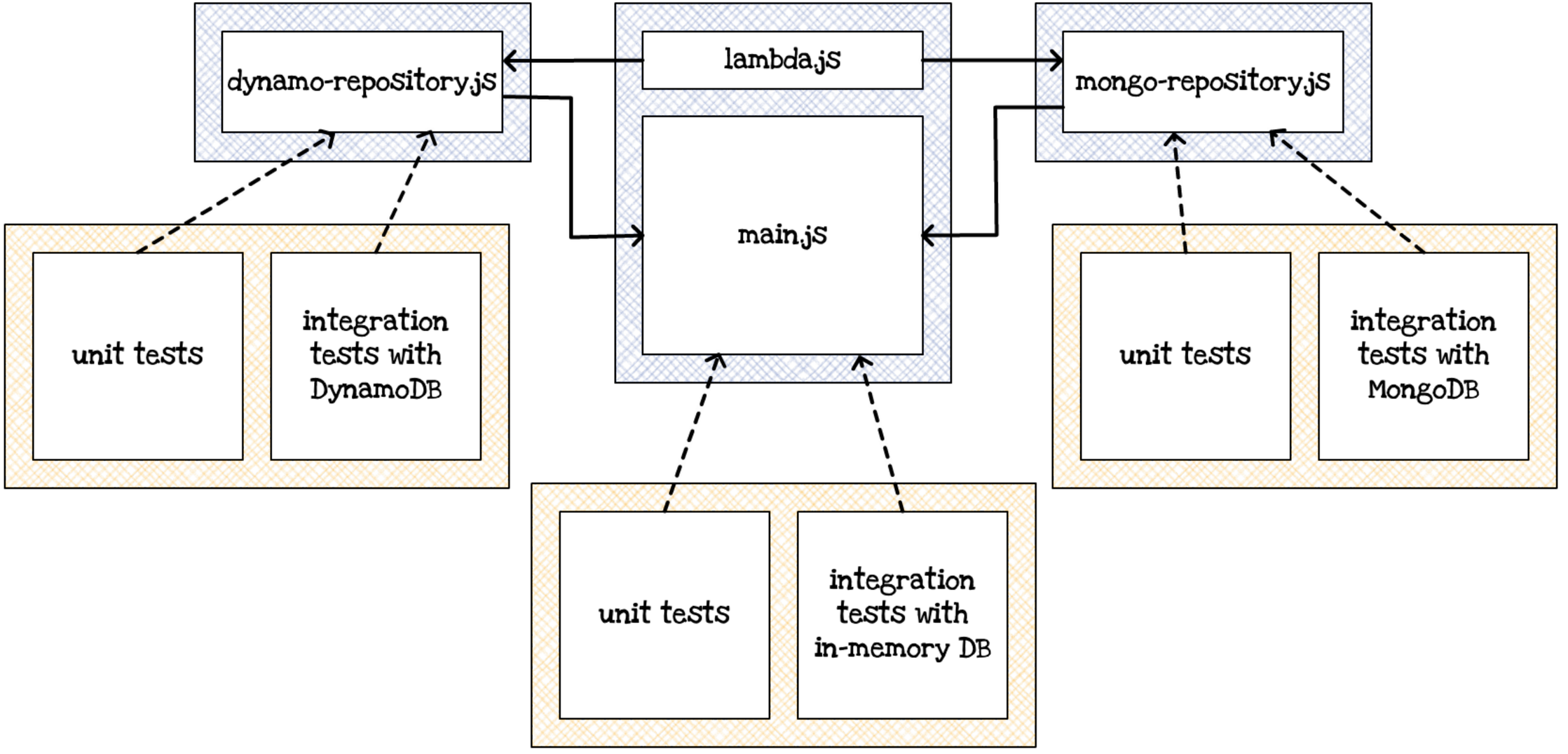
For example, this:

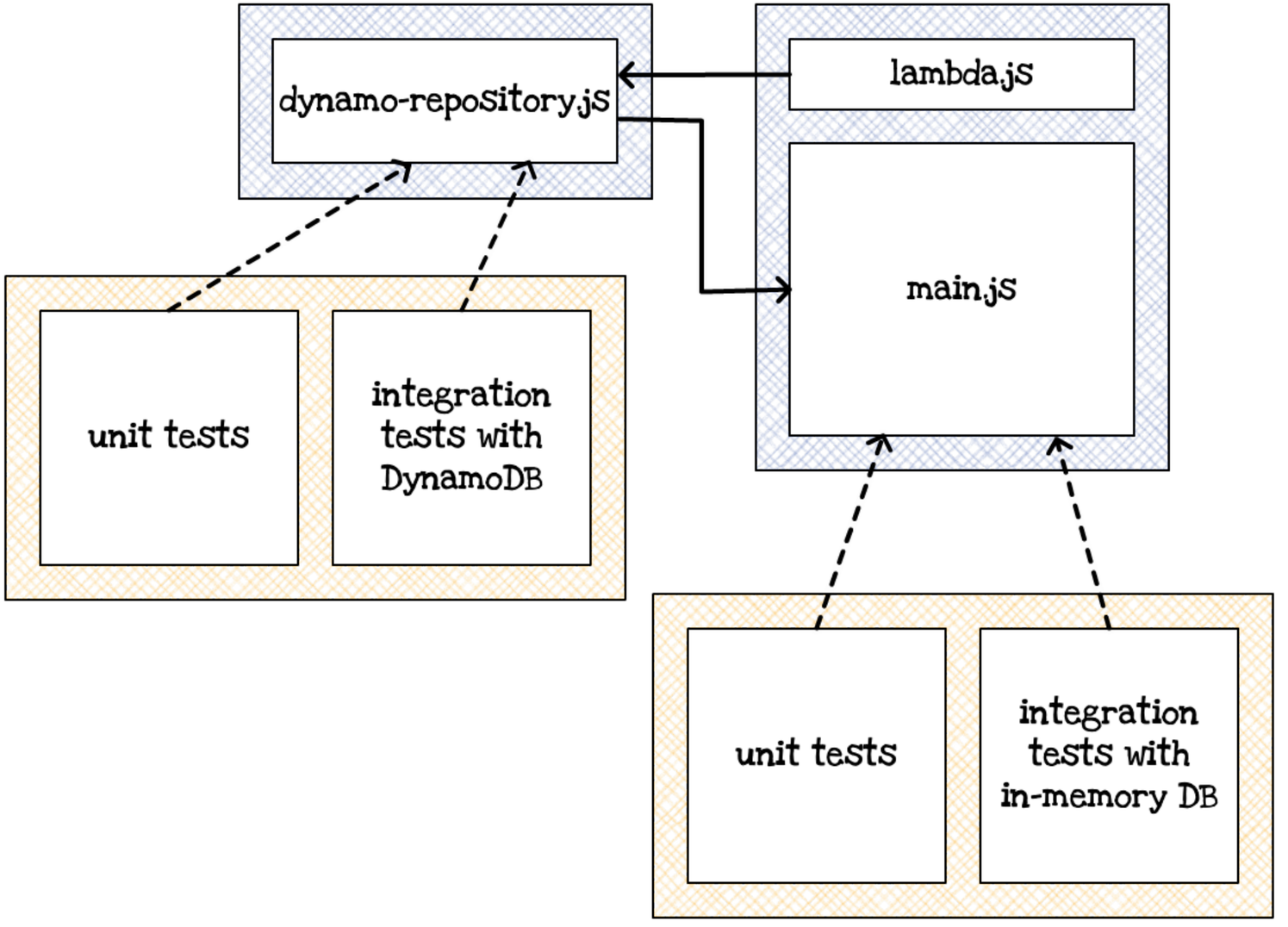
```
const mdb = new MongoDbRepository(something)
const ddb = new DynamoDbRepository(somethingElse)

const user1 = mdb.getUser(userId)
const user2 = ddb.getUser(userId)

expect(user1).toEqual(user2) // They are equal!
```

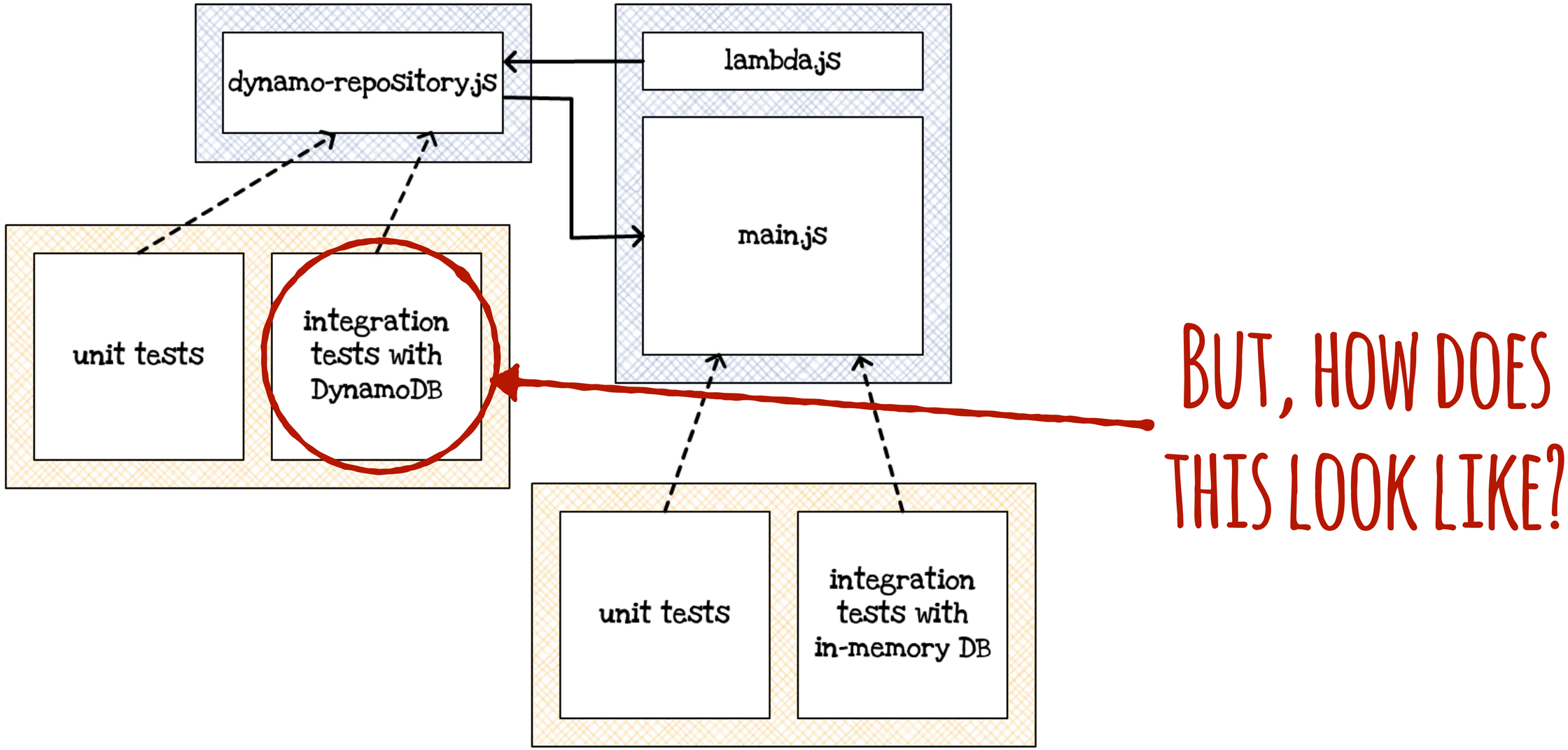
Returns a single user with the same properties.

So we simply did the following:


lambda.js
main.js
mongo-repository.js
unit tests
integration tests with MongoDB
unit tests
integration tests with in-memory DB


dynamo-repository.js
lambda.js
mongo-repository.js
main.js
unit tests
integration tests with DynamoDB
unit tests
integration tests with MongoDB
unit tests
integration tests with in-memory DB


dynamo-repository.js
lambda.js
main.js
unit tests
integration tests with DynamoDB
unit tests
integration tests with in-memory DB


dynamo-repository.js
lambda.js
main.js
unit tests
integration tests with DynamoDB
unit tests
integration tests with in-memory DB
BUT, HOW DOES THIS LOOK LIKE?

```
describe('DynamoDB repository', () => {
  describe('unit', () => { ... })

  describe('integration', () => {
    beforeAll(() => {
      // Create test DB
    })

    afterAll(() => {
      // Destroy test DB
    })

    // Tests
  })
})
```

```
beforeAll(async () => {

  const params = { ... }

  await dynamoDb.createTable(params).promise()

  await dynamoDb.waitFor('tableExists', {
    TableName: tableName
  }).promise()

})
```

```
afterAll(async () => {

  await dynamoDb.deleteTable({
    TableName: tableName
  }).promise()

  await dynamoDb.waitFor('tableNotExists', {
    TableName: tableName
  }).promise()

})
```

And we lived happily ever after...

Writing testable serverless apps using hexagonal architecture

BEYOND TESTING

@slobodan_

What should we do with things that can't be tested?

For example, Slack changes an API while your app is in production

Make sure you are monitoring your app and tracking errors

Monitoring/error-tracking tools

- Built-in tools (CloudWatch, X-Ray)
- Epsagon
- Thundra
- New Relic (they bought IOpipe)
- Lumigo
- and many others

Serverless apps often heavily relies on front end, make sure you track front end errors too!

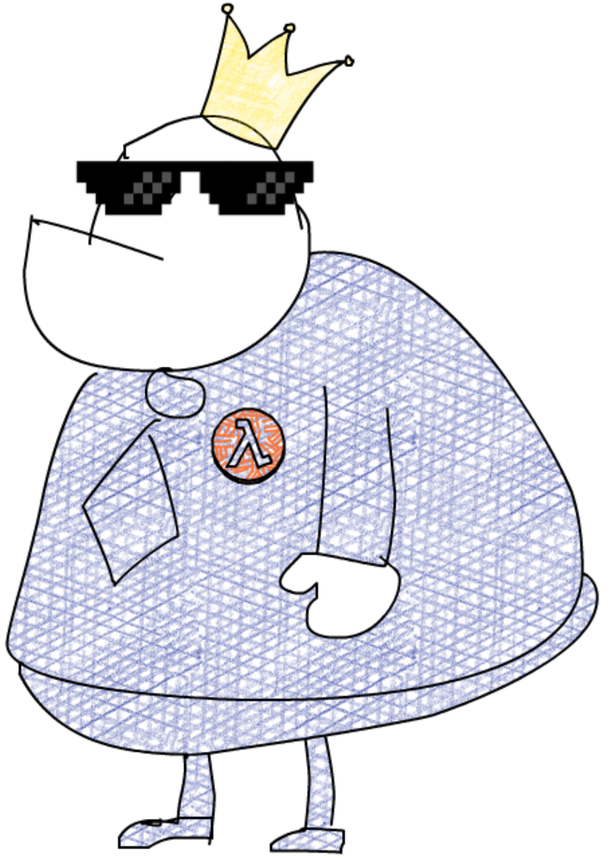
Also, services are smaller and smaller,
but integrations require fine grained permissions.

There are tools that can help you to improve your permissions and keep your app secure. *

* Protego, Puresec, and others.

But sometimes even monitoring can't help you!

You'll need a direct communication with your end users all the time!



Summary

- Good architecture helps you to maintain your switching costs low (or at least reasonable)
- Hexagonal architecture is a nice fit for serverless apps
- Test your integrations (and app in general)
- Testing is not enough, you'll need monitoring and error tracking for your serverless apps