Refactoring the Monolith

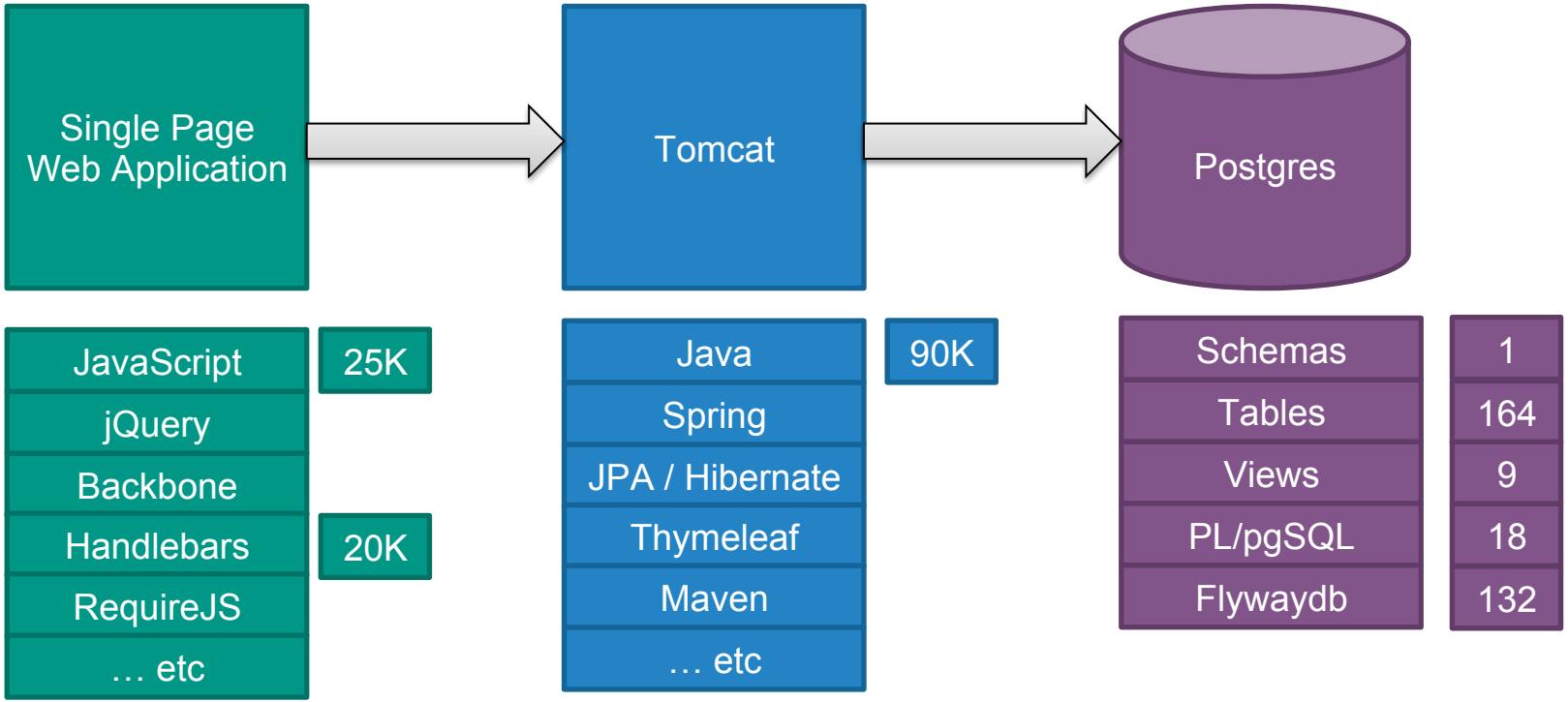
Typical Problems / Patterns / Solutions

BigMonolith Demo

Pivotal

BigMonolith Architecture

Circa March 2012 Technology Stack & Design

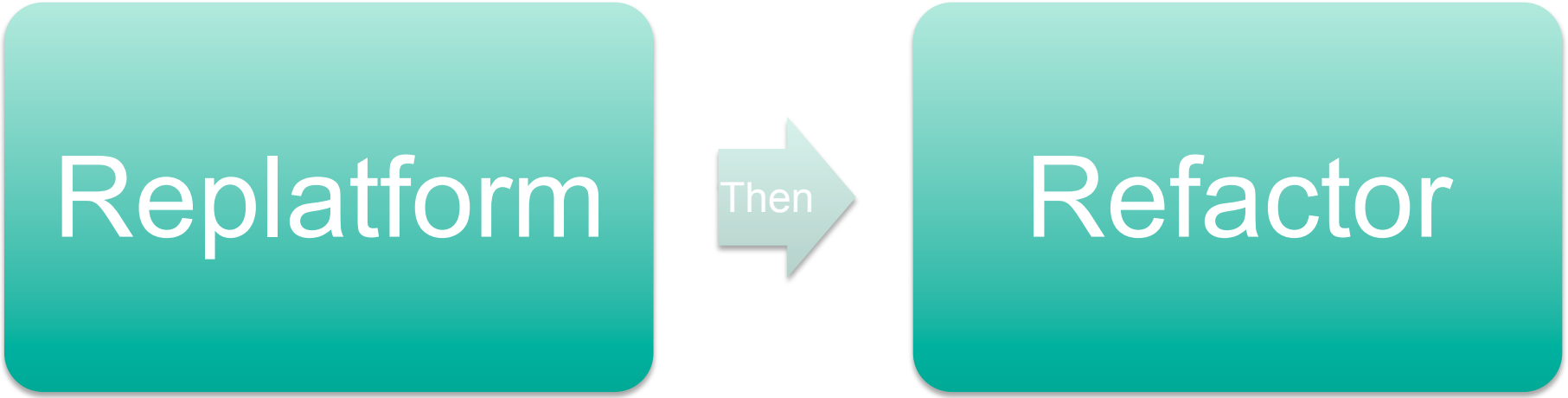


BigMonolith Code Base Tour

Pivotal

Migrating The Monolith

Always …



http://www.infoq.com/presentations/paas-migration

Let’s Replatform

Pivotal

Replatforming Obstacles

EASY

- Packaged as .war file
- No SpringBoot
- Uses JNDI to lookup database & mail server
- Environment specific settings in tomcat context.xml

Solution is to introduce a spring cloud profile to work around JNDI & context.xml settings

Pivotal

Refactoring #1: Introduce Spring Boot

Use Spring Boot Dependencies

- Introduce Spring Boot with .war packaging
- Modify the maven pom.xml files to be based on spring boot
- Newer versions of libraries brought in by SpringBoot might cause deprecation warnings or other issues
- *Minimal* refactor of current spring .xml and JavaConfig files to rely on Spring Boot configuration and conventions
- Keep deploying to current tomcat infrastructure (keep using JNDI in production)
- Lots of configuration changes minimal code changes extensive testing required

Refactoring #2: Introduce Cloud Profile

- Add spring cloud connectors to the project
- Introduce a cloud profile that can read configuration settings from VCAP_SERVICES
- Application should be able to run on PCF and on standalone tomcat

Refactoring #3: Make Jar Not War

One Code Base One Artifact

- Modify the spring boot packaging to use .jar
- Fix any issues caused by moving to a .jar format
- Eliminate any code / configuration that is no longer needed

Refactoring #4: Profile Cleanup

A separate profile for every environment / easy and clean

- Introduce/refactor a development profile that is used by developers that allows them to check source out of git and everything just works zero setup required

- Introduce/refactor an integration testing profile(s) so that integration tests can be easily run

- Introduce a production profile to capture any production specific settings that might not be cloud specific such as api access keys / logging configuration … etc

- Refactor cloud profile for any cleanup required

Refactoring #5: CI / CD Tuning

- Tune your CI / CD pipeline to take full advantage of Spring Boot

Ready to Refactor to Microservices

Pivotal

Identify Bounded Contexts

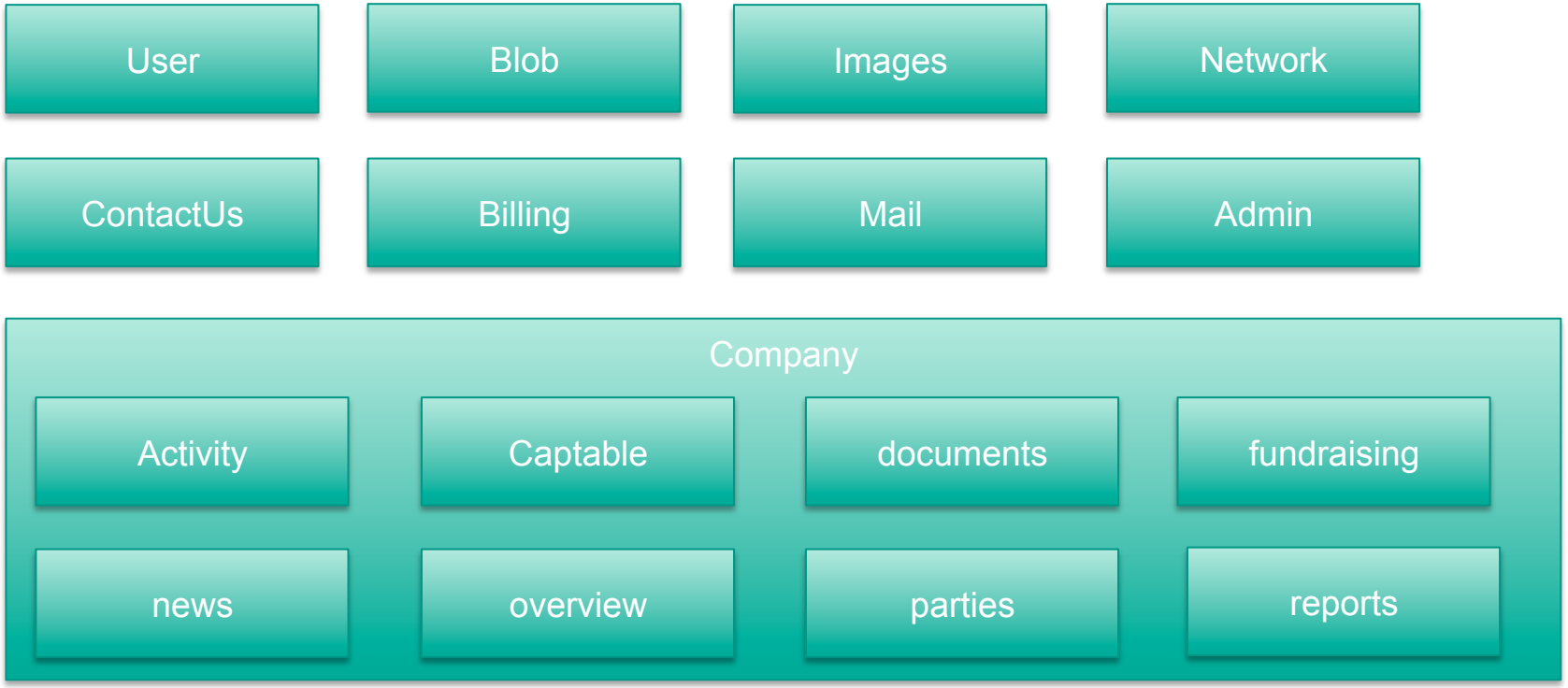
- Examine project structure
- Run static analysis tools that show how package dependencies
- Talk to developers working on the project
- Examine existing documentation / component / architecture diagrams
- Learn how to use the software and figure out carve it up into different areas from a user point of view
- Look at which parts change together

Identifying BigMonolith Bounded Contexts

Pivotal

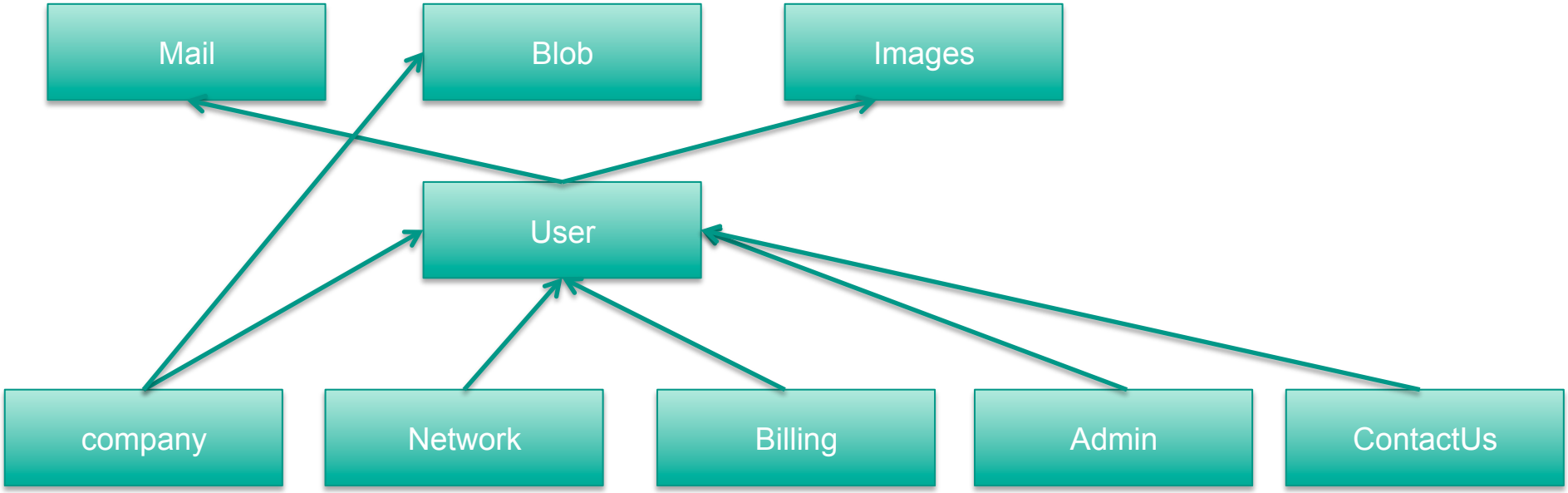
BigMonolith Bounded Contexts

First Pass / Best Guesses



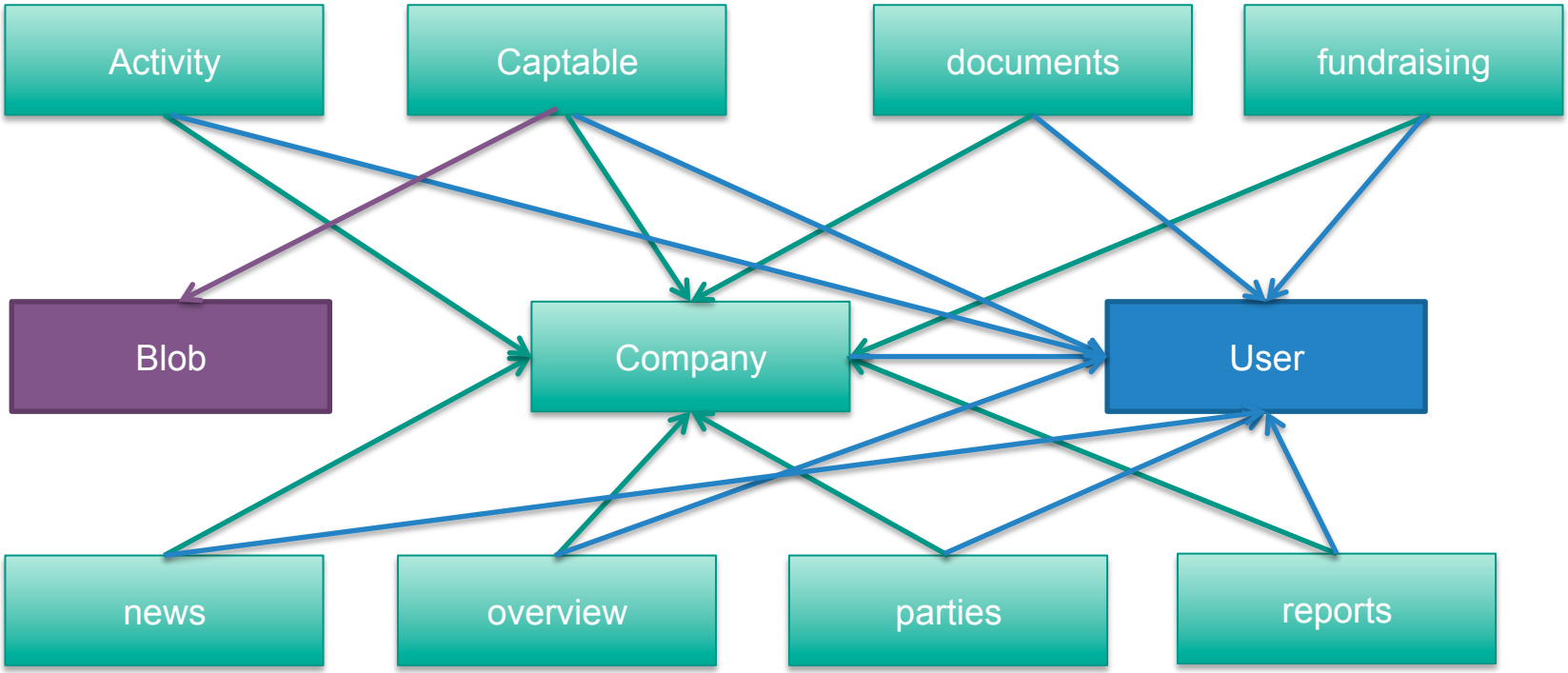
BigMonolith Bounded Contexts

First Pass / Best Guesses



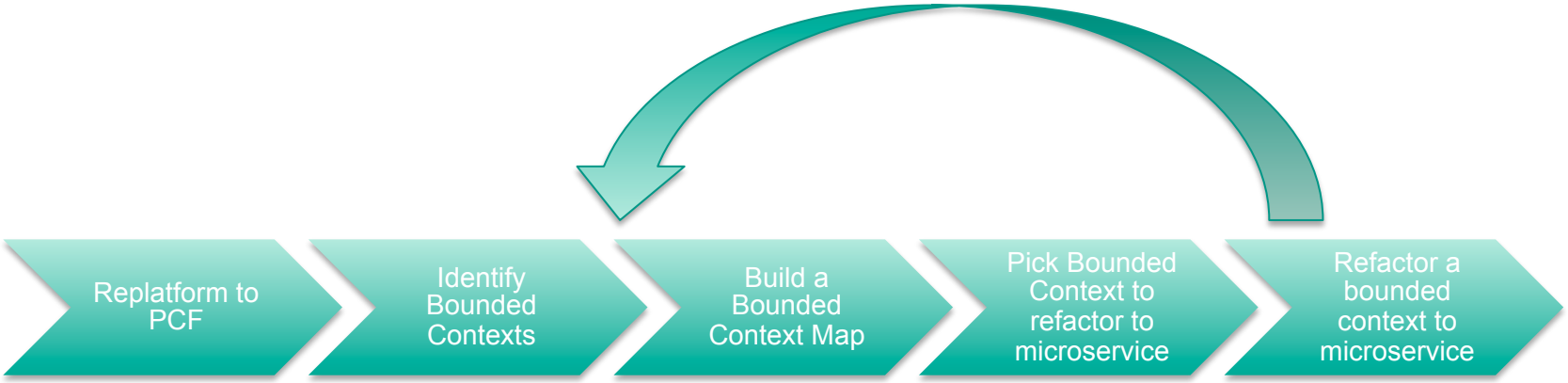
BigMonolith Core Domain

Keeping track of company data and sharing it is the core domain of BigMonolith



Pivotal

The Process So Far



Refactoring a Bounded Context

Generic Recipe

Pivotal

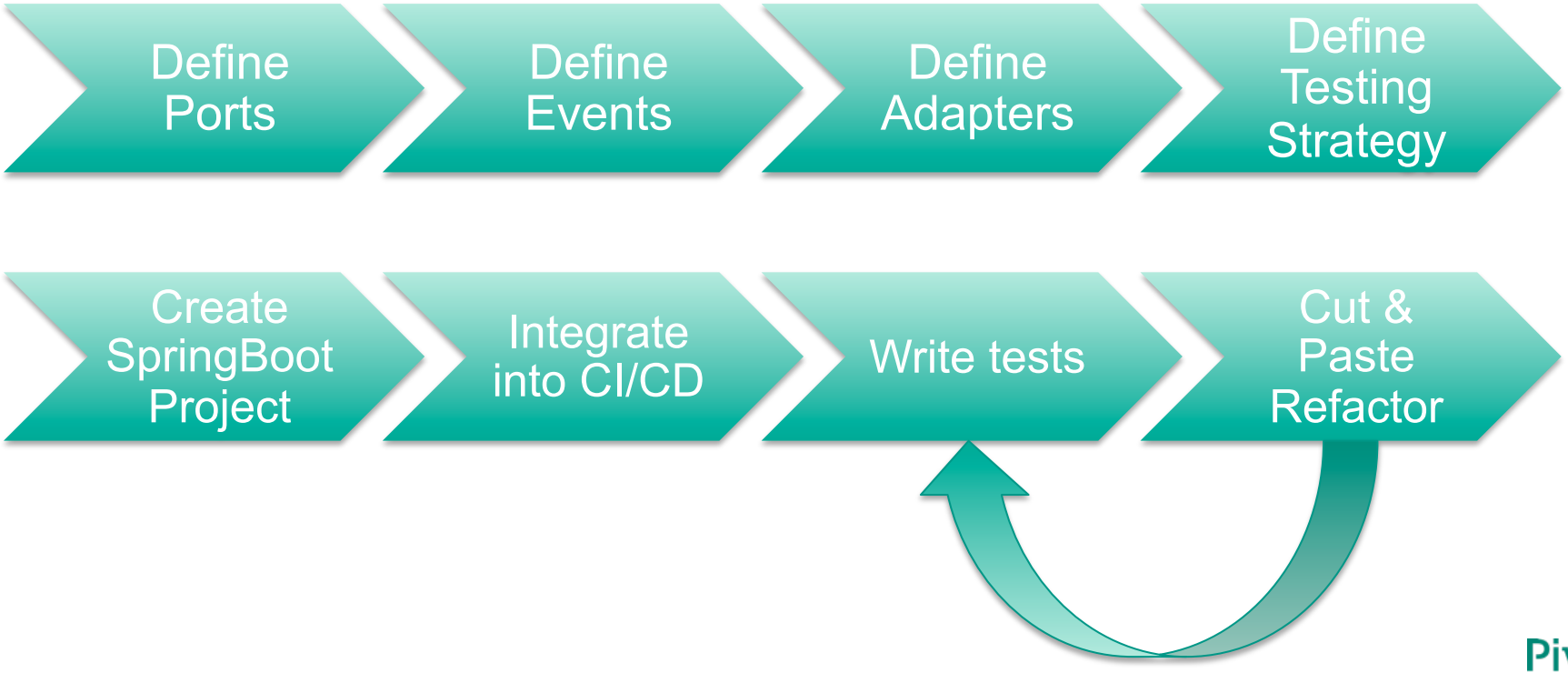
Generic Bounded Context Refactoring Recipe

SpringBoot a Hexagonal Architecture Application

- Search for all the call sites into the bounded context
- Analyze the current interfaces exposed by the bounded context
- Define the required ports for the bounded context
- Analyze external dependencies used by the bounded contexts
- Define the required adapters for the bounded context
- Analyze how the bounded context will stay in sync with the rest of the system
- Define what domain events the bounded context will emit
- Define what events the bounded context will listen for
- Copy and paste the code from the old project s
- Create a new spring boot project for hosting the refactored code
- Add the newly configured project to the CI / CD Pipeline
- Write unit and integration tests
- Cut and paste code and refactor it
- Iterate until done

Generic Bounded Context Refactoring Recipe

SpringBoot a Hexagonal Architecture Application



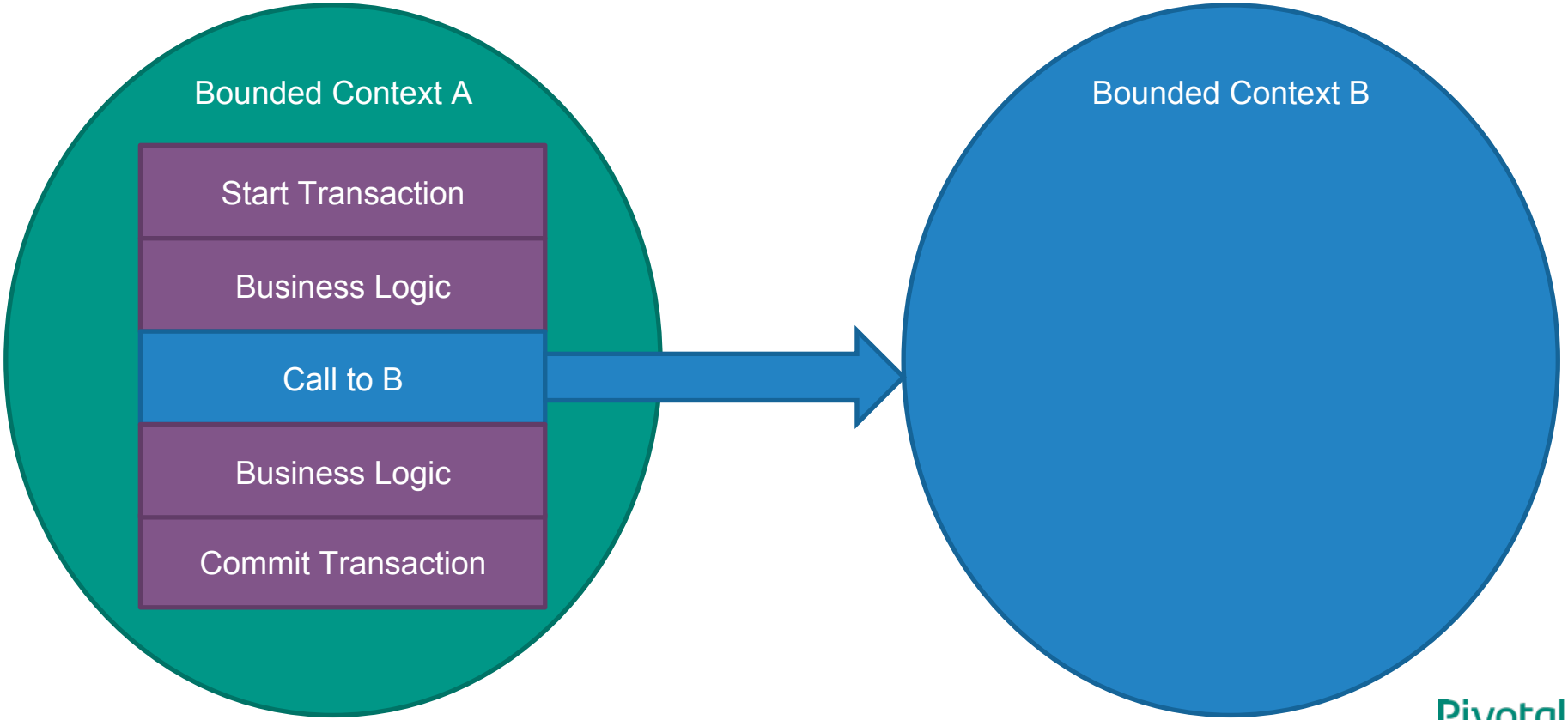
Pivotal

Refactoring Mail Bounded Context

Code Tour

Pivotal

Obstacle #1: Transaction Boundaries



Ladder #1: Distributed XA Transactions

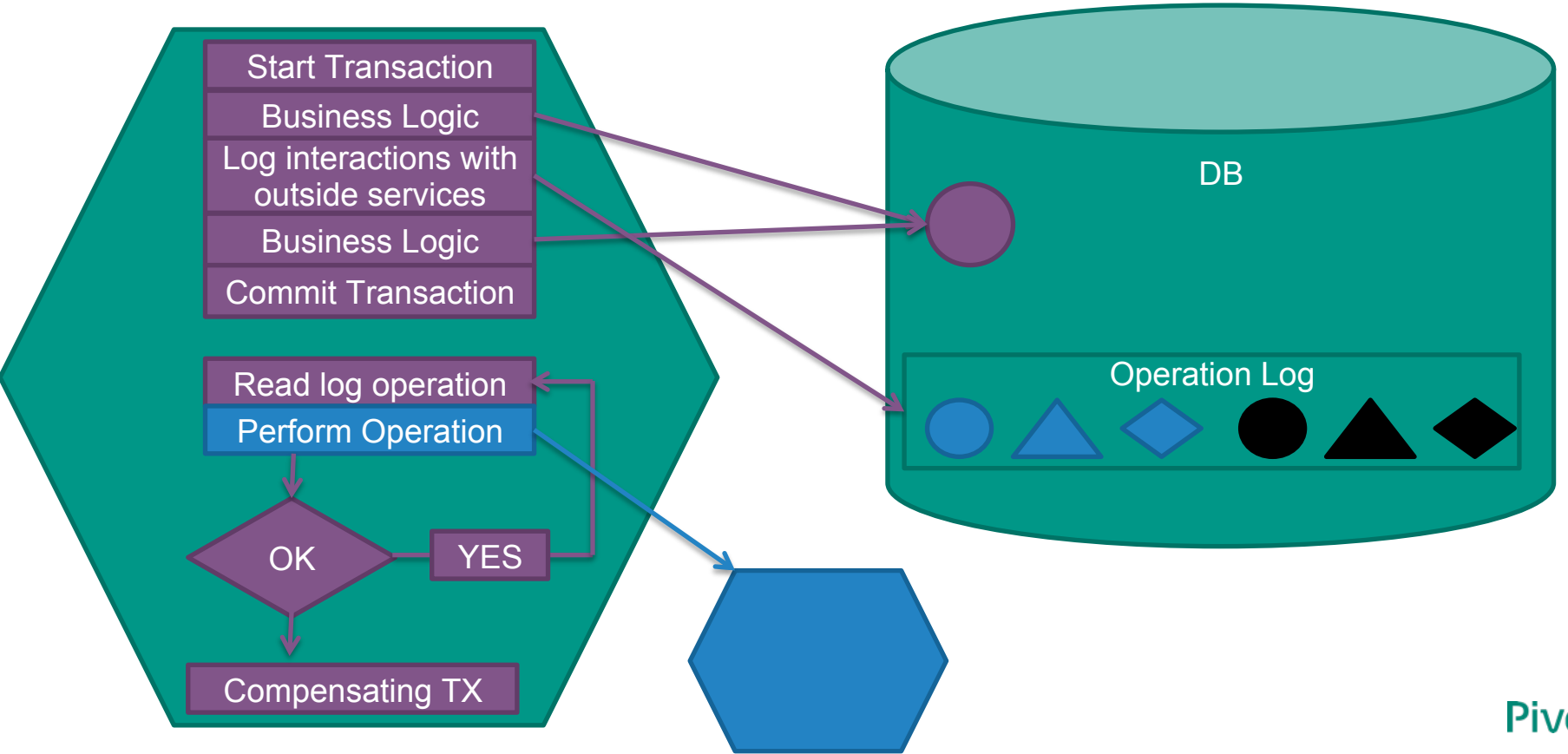
- Too brittle
- TX manager needs to run in a VM since it needs persistent disk BOSH release would be ideal
- too much work to be a good fit for PCF now

ATOMIKOS

Pivotal

Ladder #2: Application Level Eventual Consistency

Persist operation state in the client microservice and track to success or compensate



Pivotal

Tools for Implementing Application Level Eventual Consistency

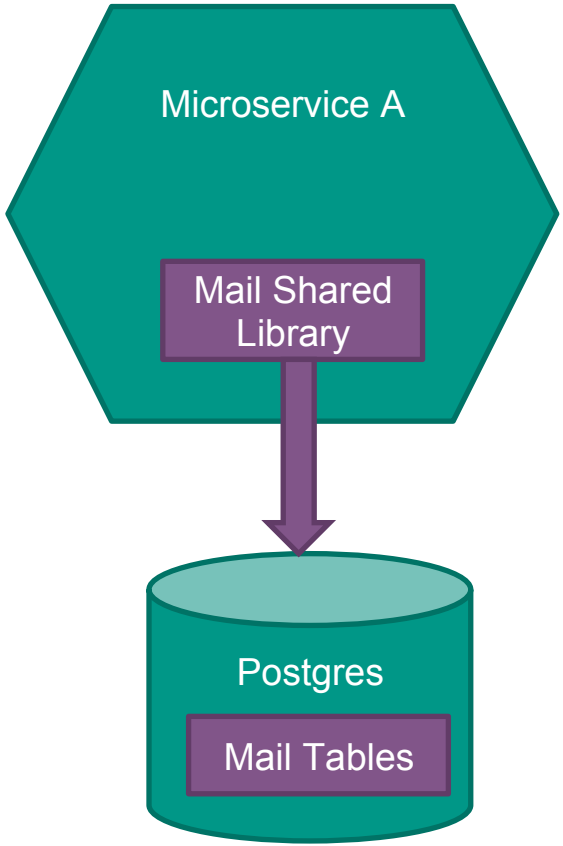
- Databases SQL or NoSQL Gemfire
- Queues (RabbitMQ, JMS, WebSphereMQ, Tibco .. etc)
- Spring State Machine
- Java 8 Completeable Futures

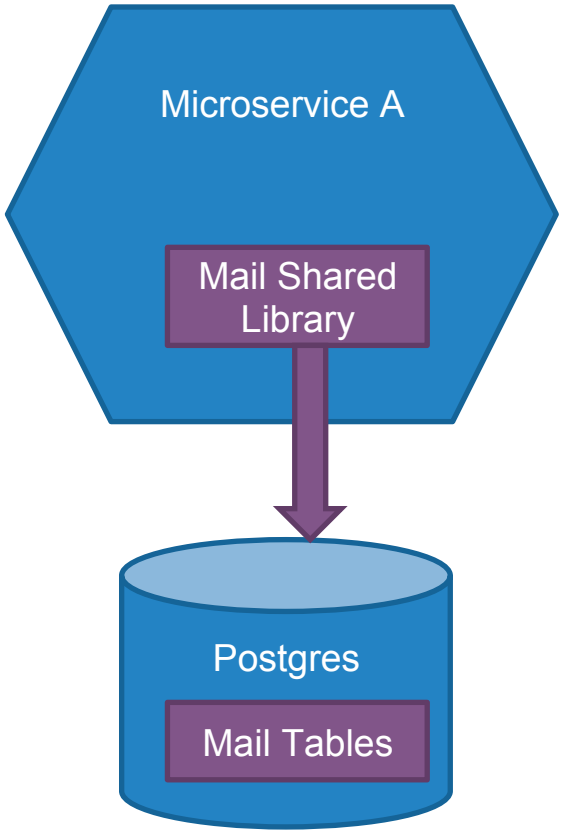
- “Distributed transactions in Spring, with and without XA” from Dave Syer http://www.javaworld.com/article/2077963/open-source-tools/distributed-transactions-in-spring--with-and-without-xa.html

Tip #1: Transaction Boundaries stay inside a Bounded Context

Mail Refactoring #1: Extract Shared Library

Extract mail as as a shared library and introduce it as a utility into any microservice

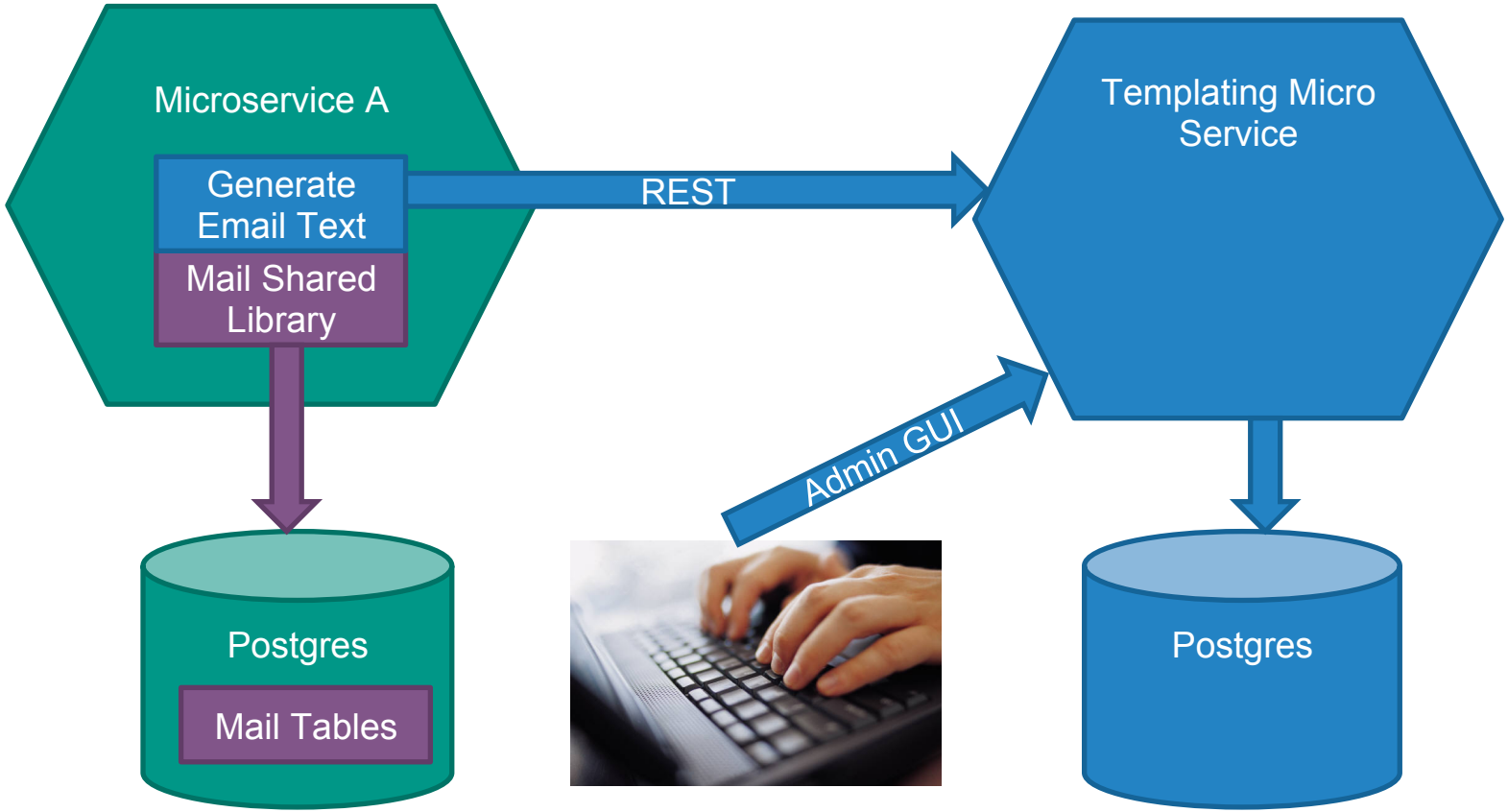




Pivotal

Mail Refactoring #2: Template Microservice

Extract templating into a microservice

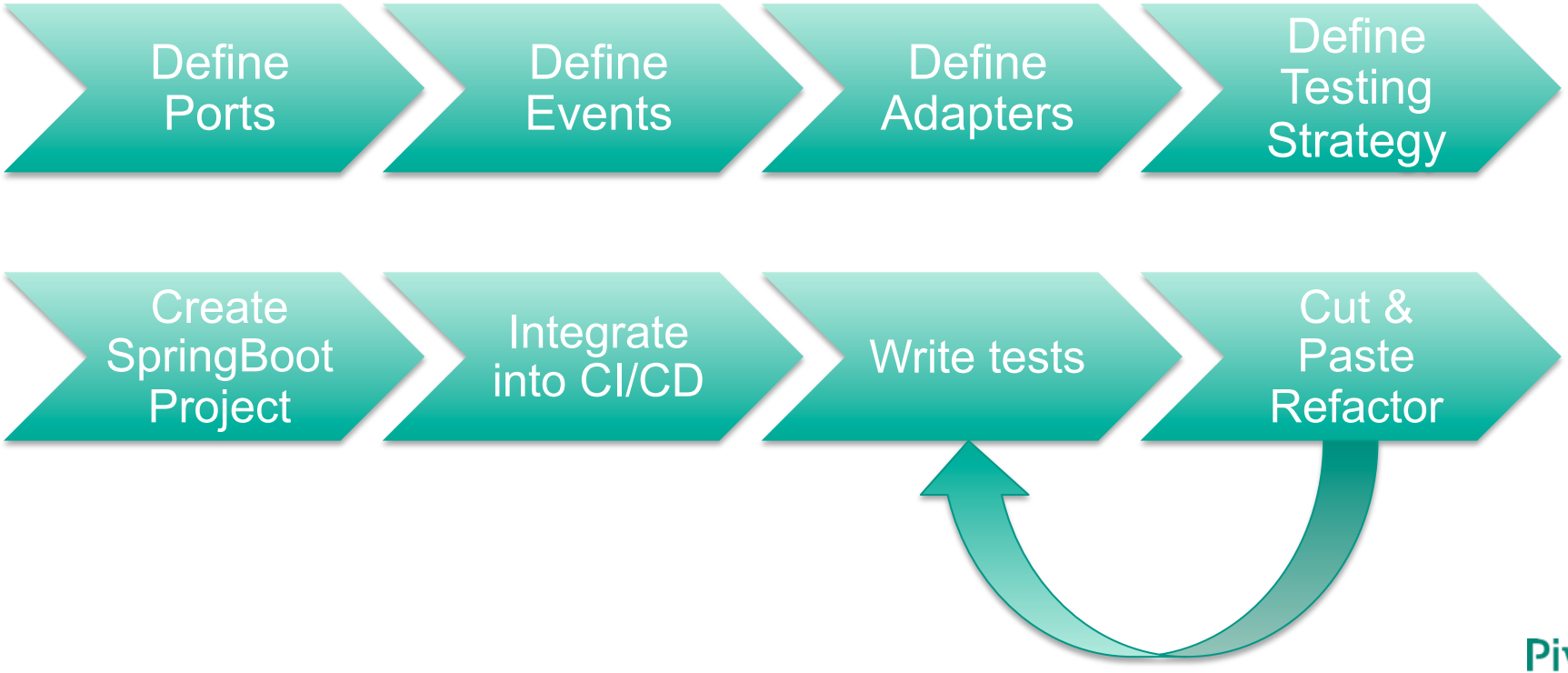


Refactor the Blob Bounded Context

Pivotal

Blob Bounded Context Refactoring

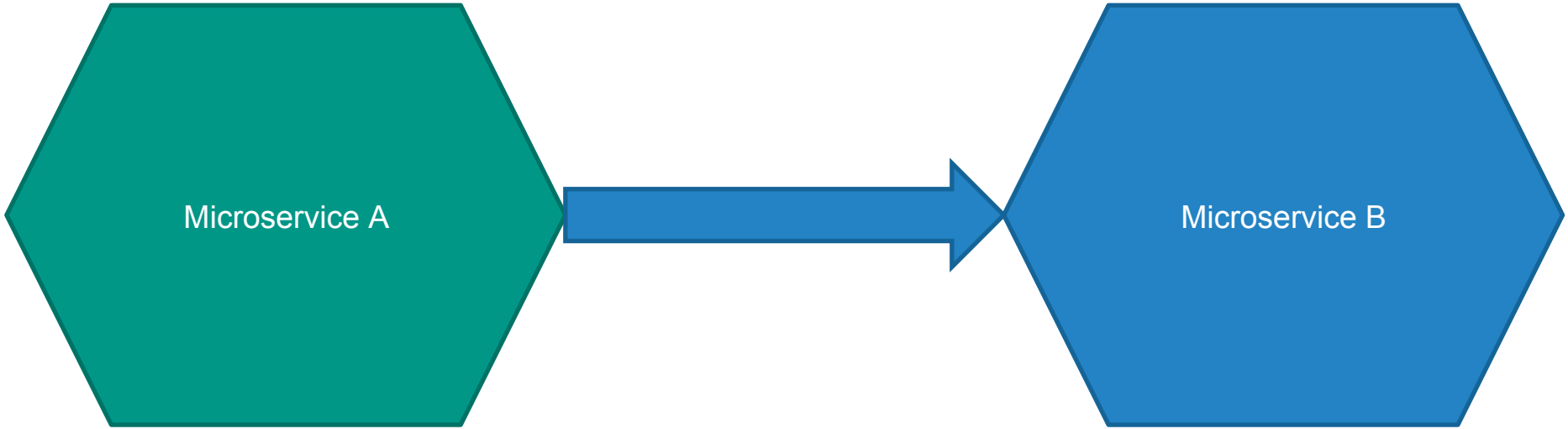
The generic Recipe Works



Pivotal

Obstacle #2: Security

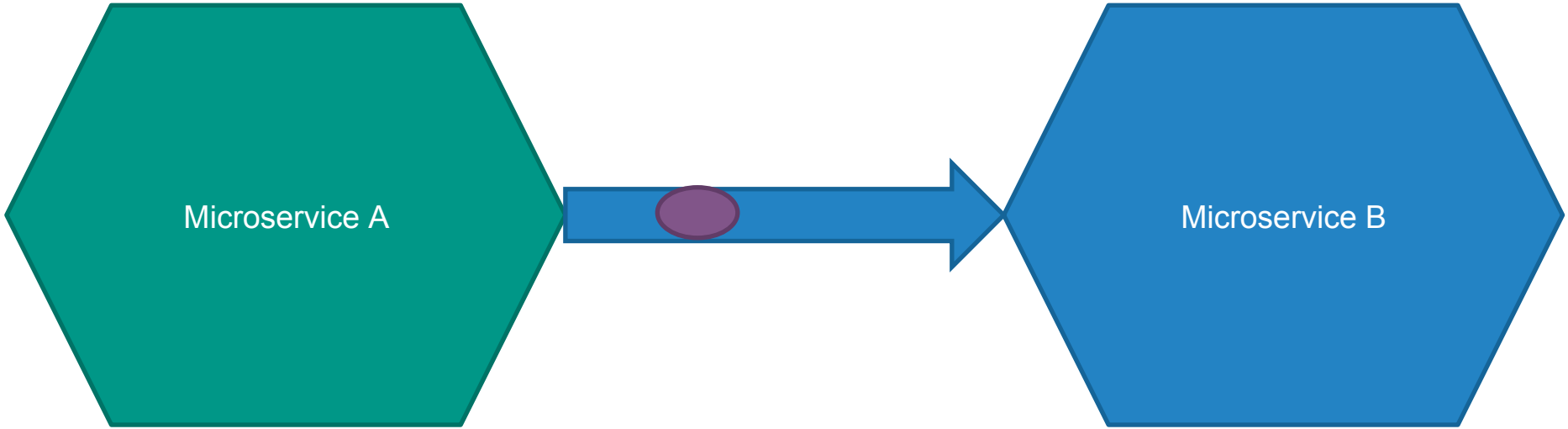
How to protect access to a microservice



Pivotal

Ladder #1: Api Keys

Service Admin Hand out API keys for use by callers

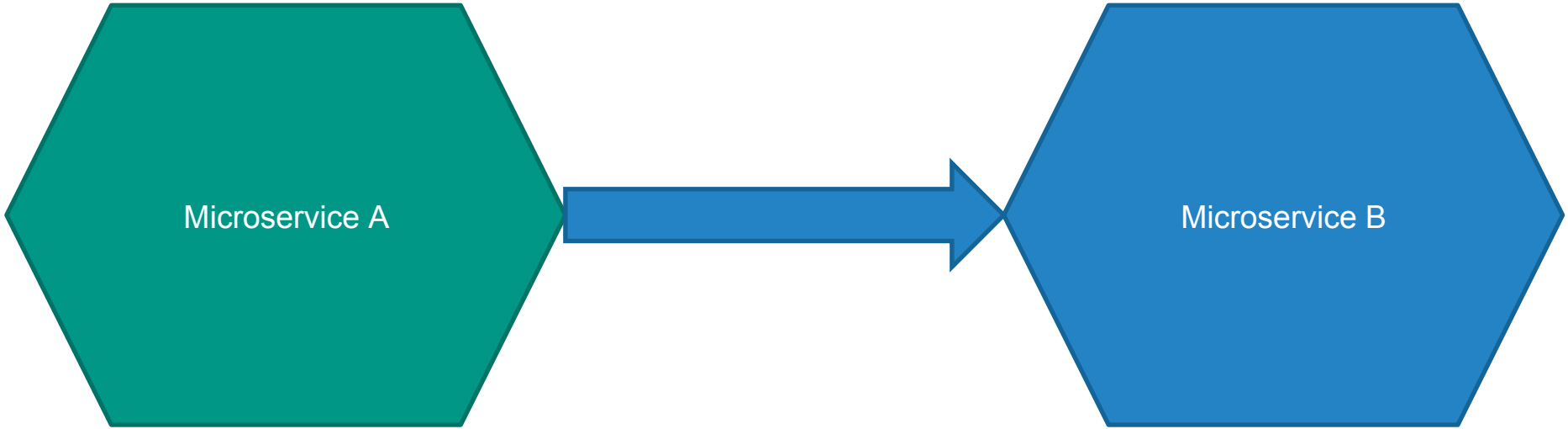


Simple to Implement and use but does not propagate user identity between services

Refactor Contact Us Bounded Context

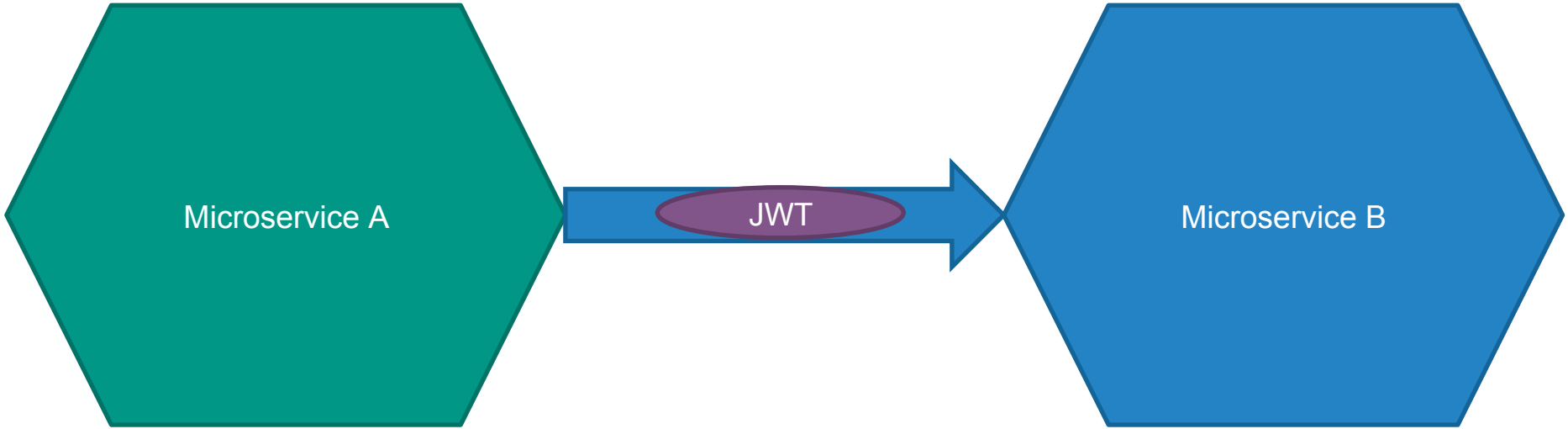
Obstacle #3: Propagating Identity

How to propagate user identity / profile between microservice calls?



Ladder #1: Use JWT Tokens

Use JSON Web Token to pass user info between microservices



JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.

JWT Example

Visit JWT.IO

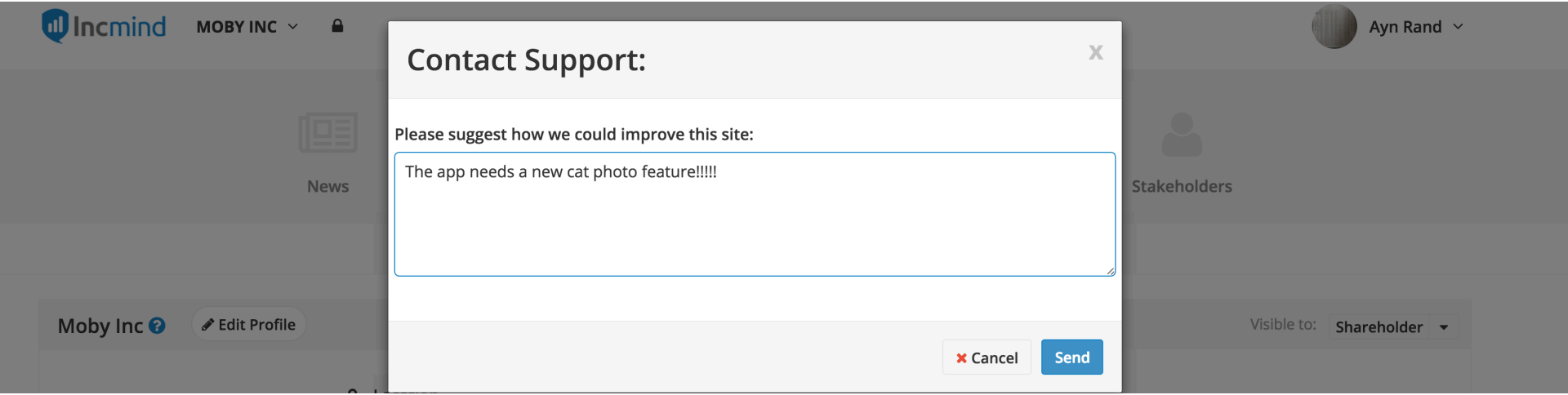
```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV
CJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwi
bmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4
iOnRydWV9.TJVA95OrM7E2cBab30
RMHrHDcEfxjoYZgeFONFh7HgQ
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Pivotal

Obstacle #4: The User Interface

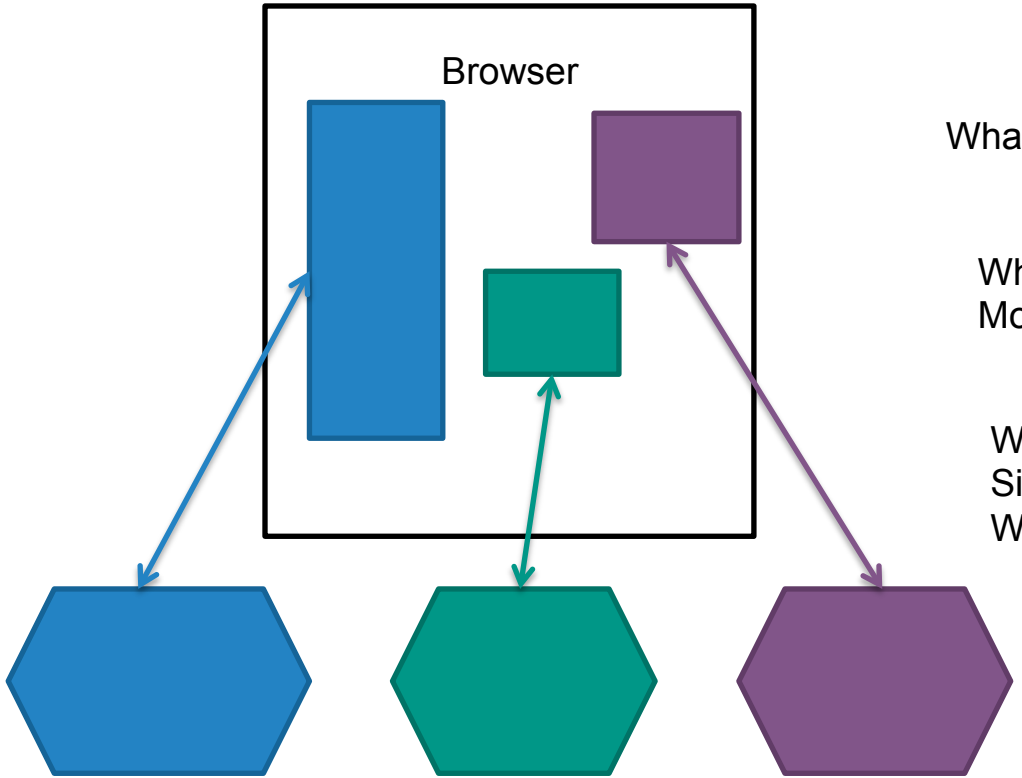
Where should the UI code live?



Pivotal

Obstacle #4: The User Interface

Where should the UI code live? How is it composed into a single UI each microservice?



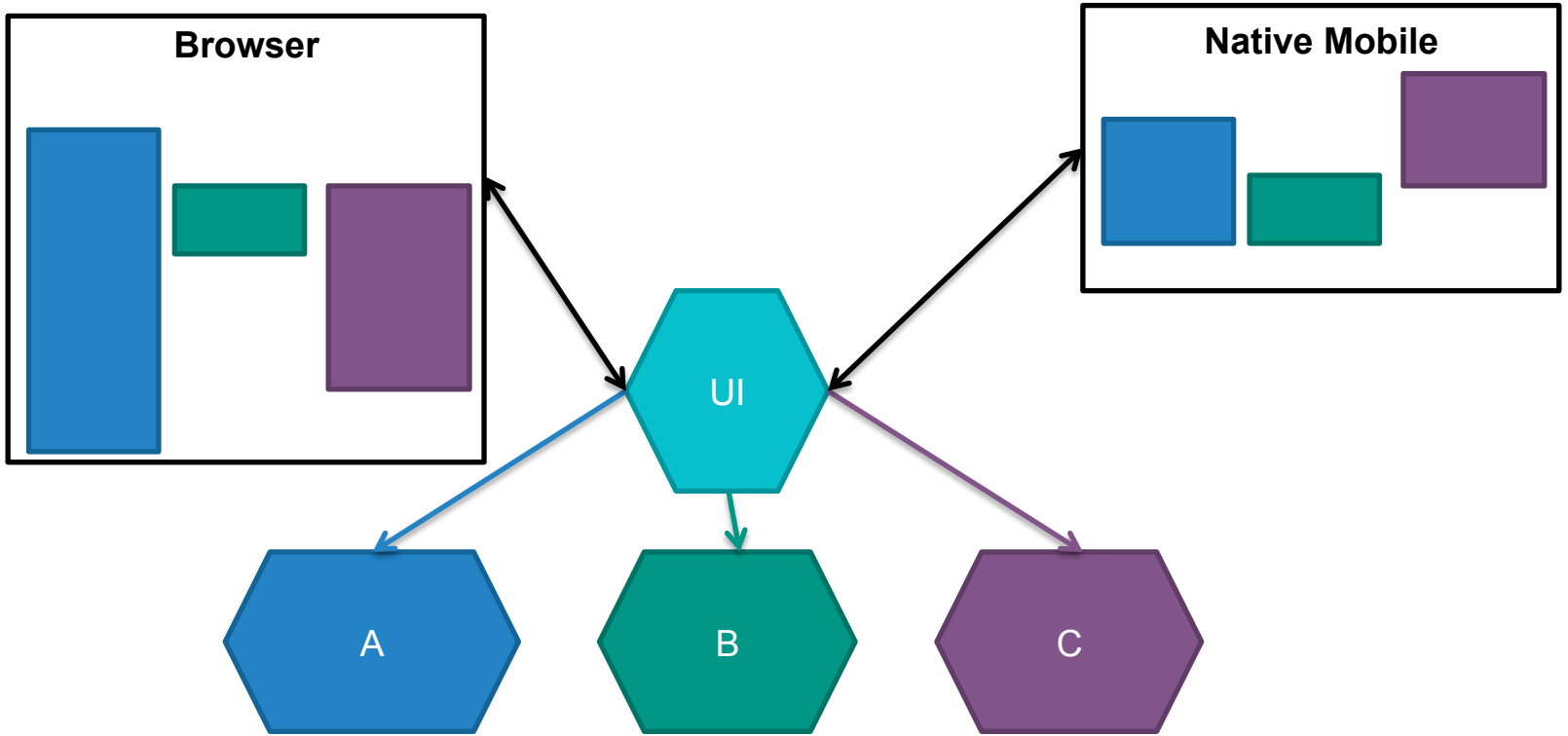
What about CORS?

What about a Native Mobile Clients?

What about Server Side Rendering for a Web UI?

Ladder #1: Monolithic Edge UI Gateway

Make a UI Microservice that is exposed to end users and have it serve up the UI?

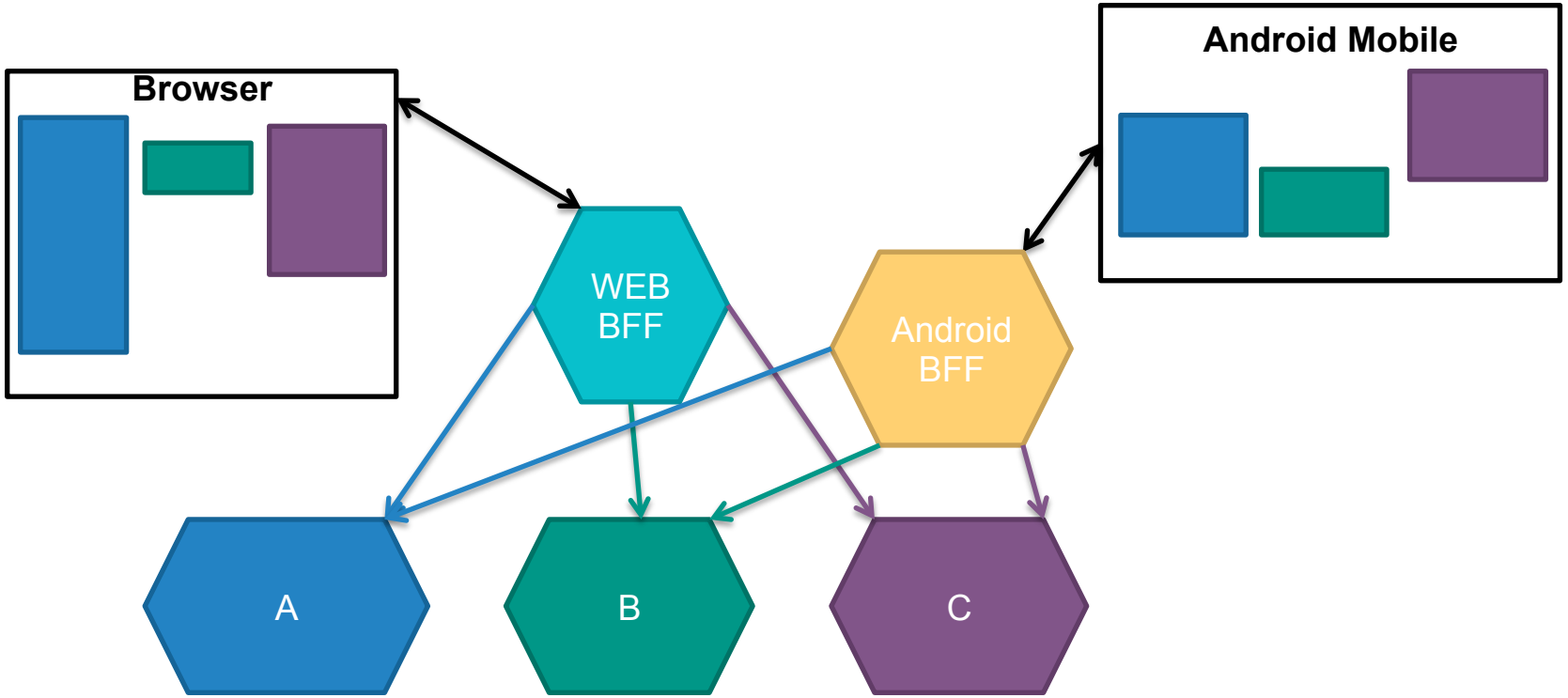


Pivotal

Ladder #2: Back End For Front End

Extend each UI experience with a dedicated backend component for UI

http://samnewman.io/patterns/architectural/bff/

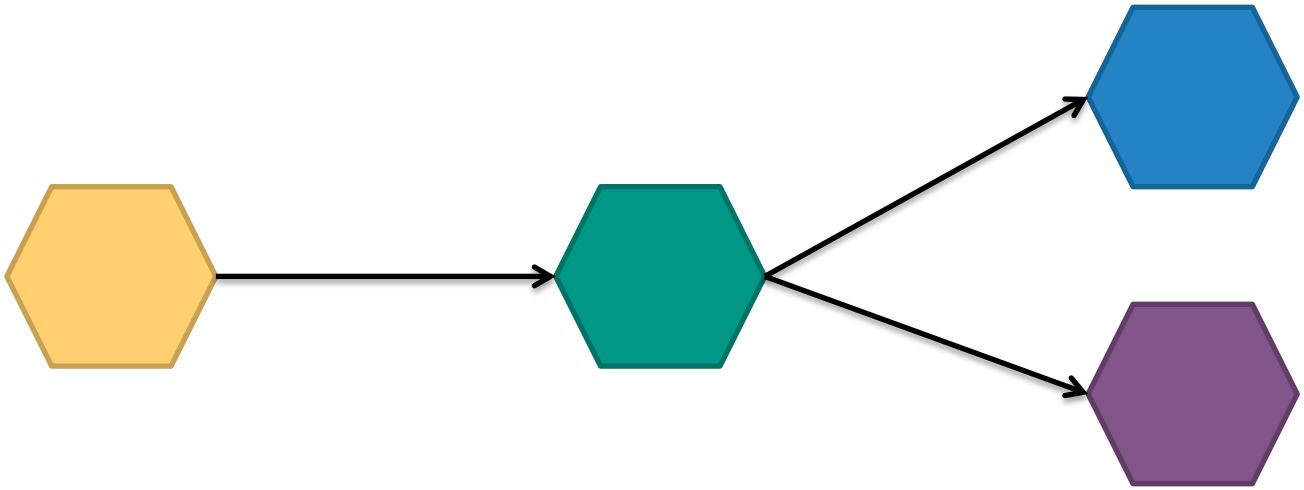


Pivotal

Refactor User Bounded Context

Obstacle #5: Centralized Authentication / Authorization

Does each microservice do it’s own authentication & Authorization



Ladder #1: OAuth2 / JWT & Friends

Leverage the CloudFoundry User Account and Authentication (UAA) Server

- Will Tran Microservices security Talk
http://www.infoq.com/presentations/microservices-security-spring-cloud

- FreddyBBQ UAA Example https://github.com/william-tran/microservice-security

- UAA GitHub https://github.com/cloudfoundry/uaa

- Pivotal PCF SSO Service

- Spring Security OAuth project

Other Common Obstacles

Obstacle #6: Foreign Keys Constraints

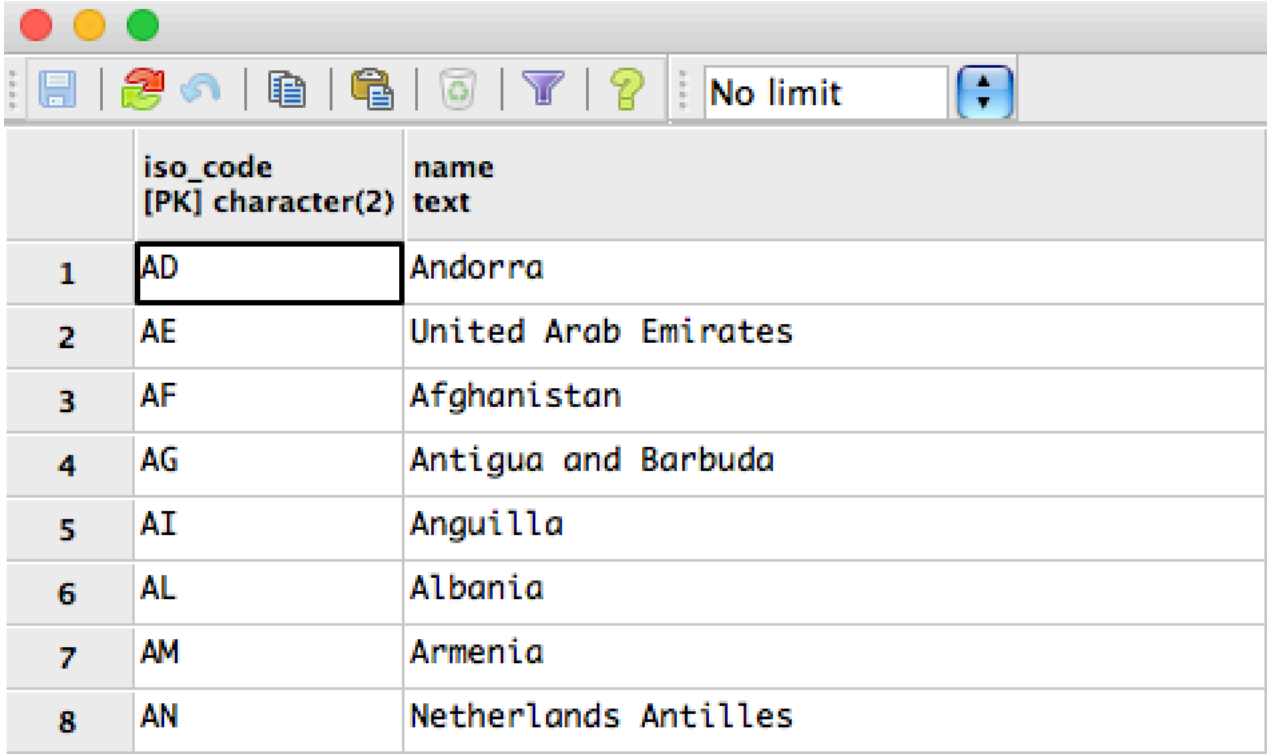
How are Foreign Key Constraints Validated Across Table is Different Bounded Contexts

- Enforcing Foreign Key Constraints between microservices becomes an application level problem to be handled by the microservices rather than the database

- Usage of Immutable Stable URI's to identify Foreign keys can be helpful

Obstacle #7: Shared Static Data

How to deal with common reference data in the database?



	iso_code [PK] character(2)	name text
1	AD	Andorra
2	AE	United Arab Emirates
3	AF	Afghanistan
4	AG	Antigua and Barbuda
5	AI	Anguilla
6	AL	Albania
7	AM	Armenia
8	AN	Netherlands Antilles

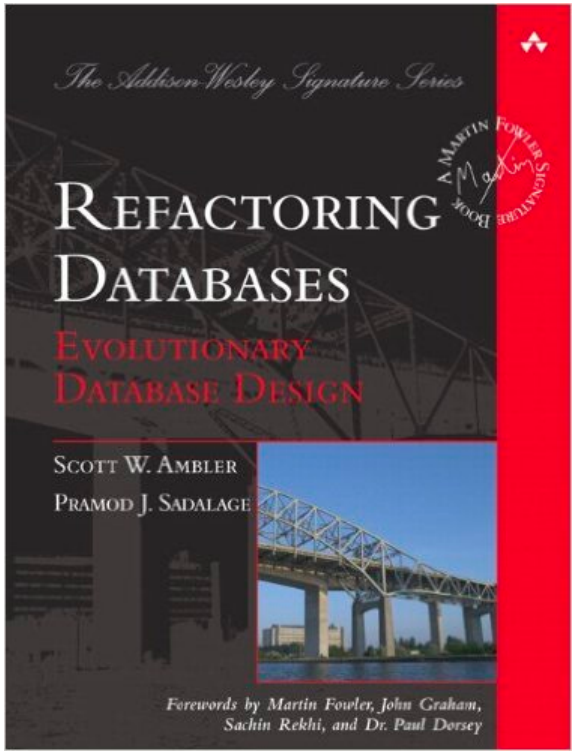
Ladder #1: Shared Static Data Becomes Code

Turn Static Shared Data into Code Accessible via dependency manager

```
public enum CurrencyCode
{
    CAD("CAD"), USD("USD"), EUR("EUR");

    private final String isoCode;
    public final Currency currency;
    public final MathContext displayContext;
    public final RoundingMode ROUNDING_MODE = RoundingMode.HALF_EVEN;
    public final MathContext computeContext = new MathContext(6, ROUNDING_MODE);

    private CurrencyCode(String isoCode)

    public String getCode()

    public static CurrencyCode parse(String isoCode)

    public BigDecimal round(BigDecimal amount)
}
```

Obstacle #8: Refactoring Database

How to split monolithic databases / refactor databases Safely?



Pivotal

Obstacle #9: Reporting

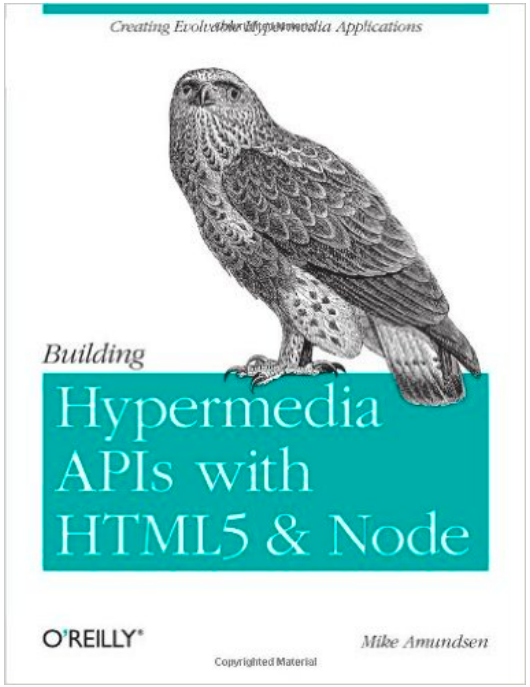
My data is scattered across many databases how do I get an integrated view?

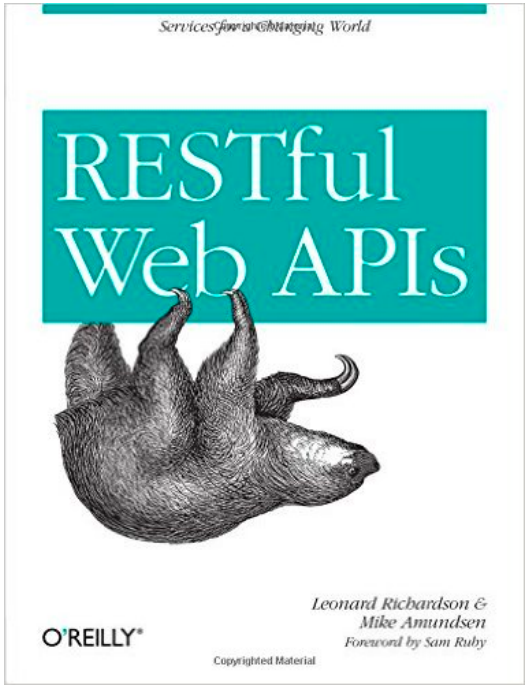
Spring Cloud Data Flow

Talk to the Pivotal Data Engineers!

Obstacle #10: Versioning

How to Evolve API versions without breaking the world





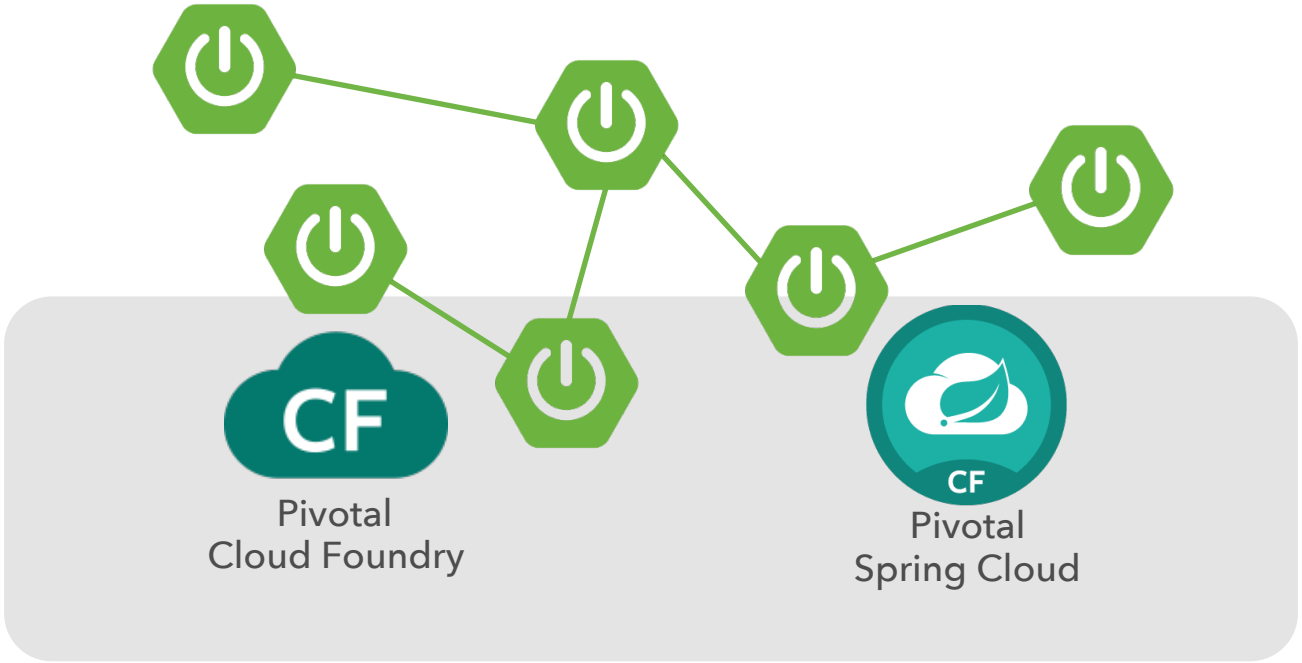
Consumer-Driven Contracts: A Service Evolution Pattern

http://martinfowler.com/articles/consumerDrivenContracts.html

Pivotal

Obstacle #11: Distributed Systems are Hard!

Pivotal's got you covered!



Building Planes in the Sky

Not as Hard as it look with Pivotal on Your Side ☺

Pivotal


Pivotal®
Transforming How The World Builds Software