Modernizing Systems

with Microservices, Hystrix and RxJava

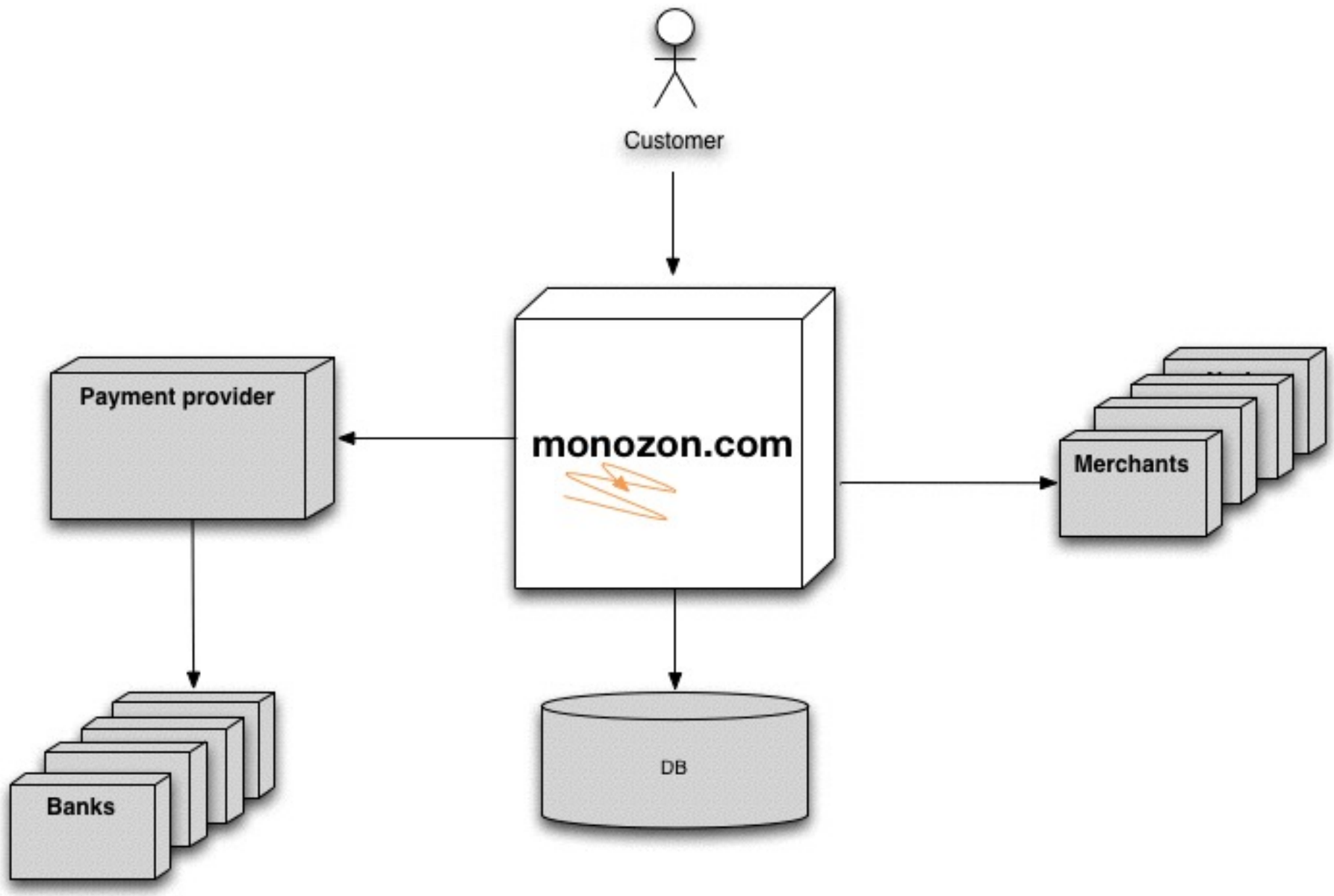
Holger Kraus

Javadays Kiev, Nov 4, 2015

A typical System

monozon.com

The context



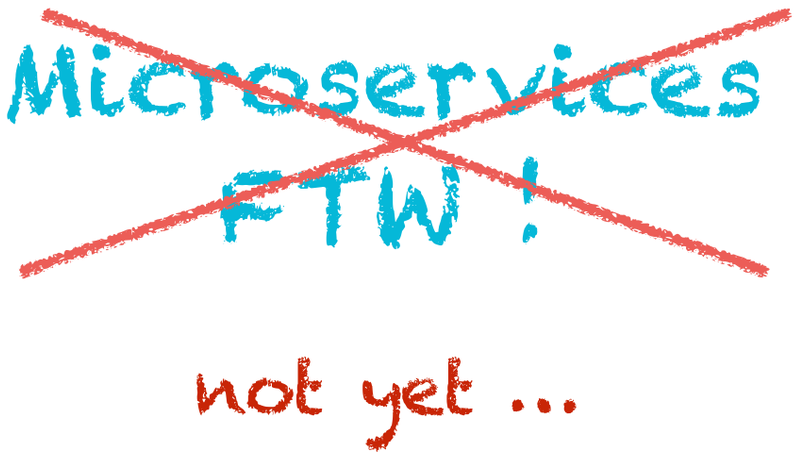
Current problems

- Maintenance is difficult
- New features need a lot of time
- Very unstable
- Outdated technology
- Doesn’t scale

+ frustrated developers :(

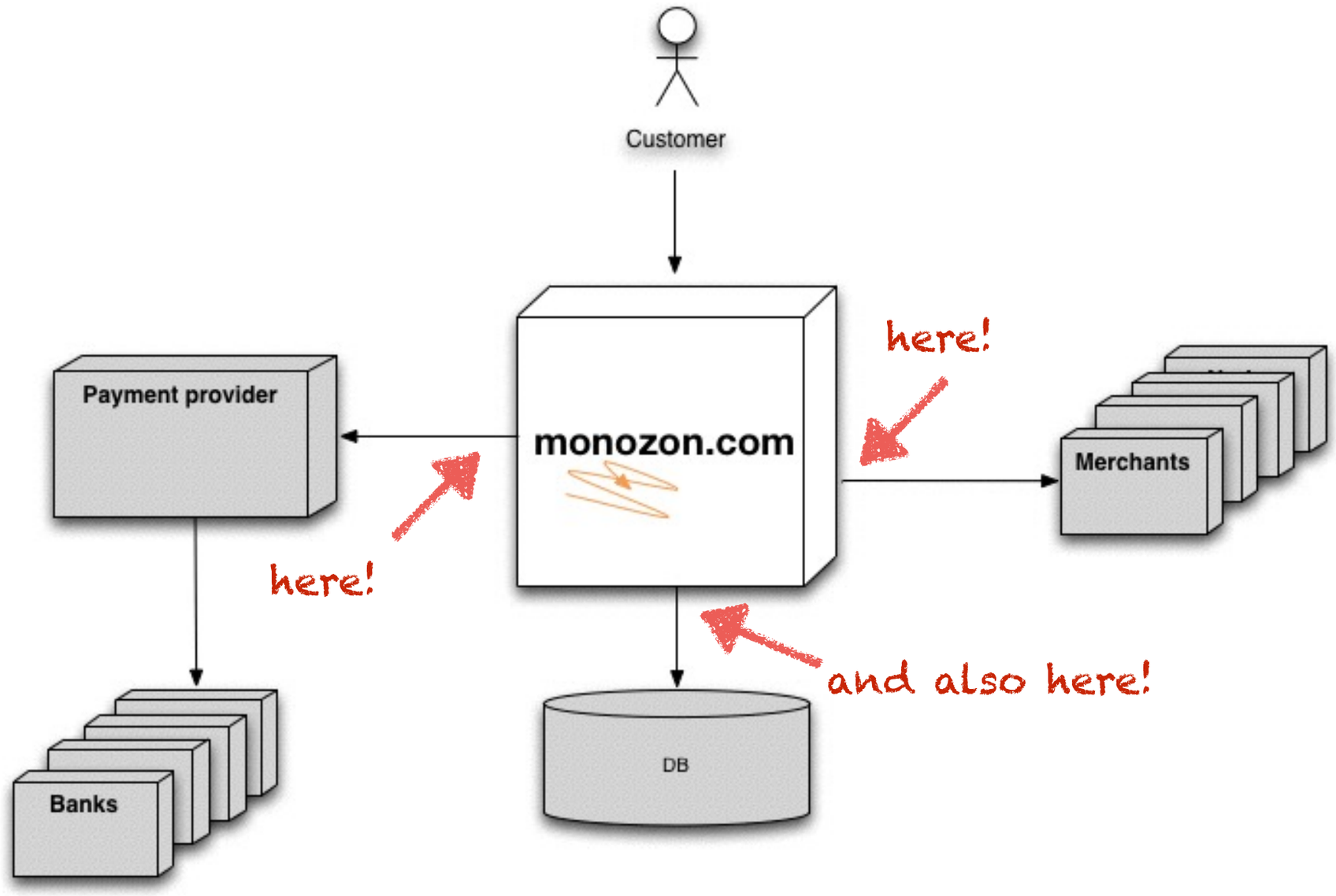
Current problems

- Maintenance is difficult
- New features need a lot of time
- Very unstable
- Outdated technology
- Doesn’t scale



Stabilize first!

External dependencies



Cascading Failures

https://www.flickr.com/photos/benstassen/2991003141/

Stability patterns

- Timeouts
- Circuit Breaker
- Bulkhead

... Fail Fast, Steady State, Handshaking, Test Harness, Decoupling Middleware

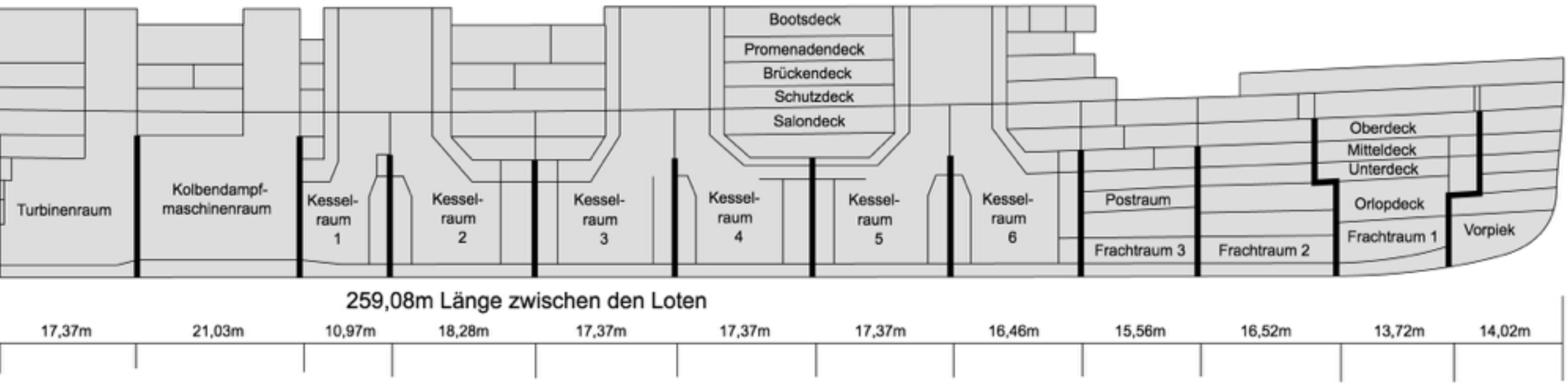
Timeout

https://www.flickr.com/photos/55293400@N07/15564061004

Bulkheads

https://www.flickr.com/photos/10413717@N08/6935206524

Ships



http://de.wikipedia.org/wiki/RMS_Titanic#/media/File:Titanic_Structure_de.svg

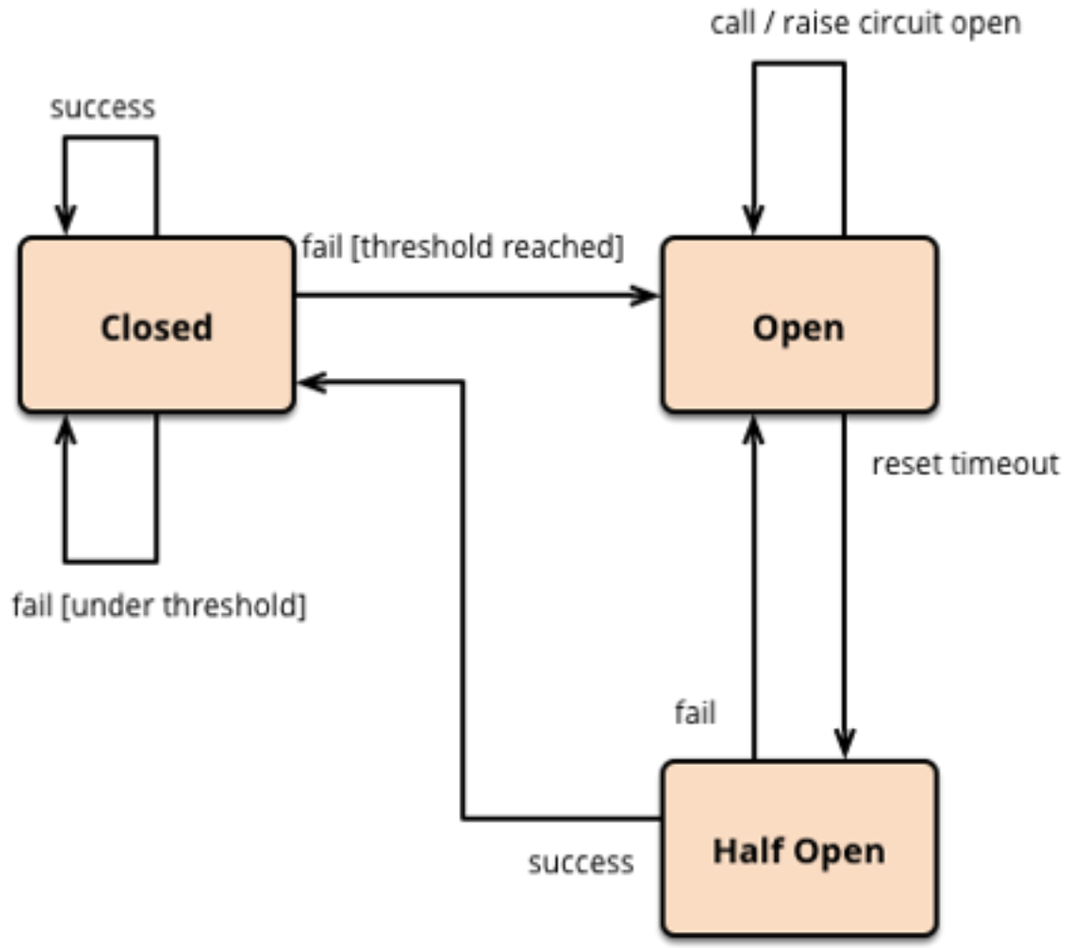
Bulkheads and IT

- Thread pools
- Database connection pools
- Instances
- Server
- Data center

Circuit Breaker

https://www.flickr.com/photos/leafbug/409950515/

Circuit Breaker

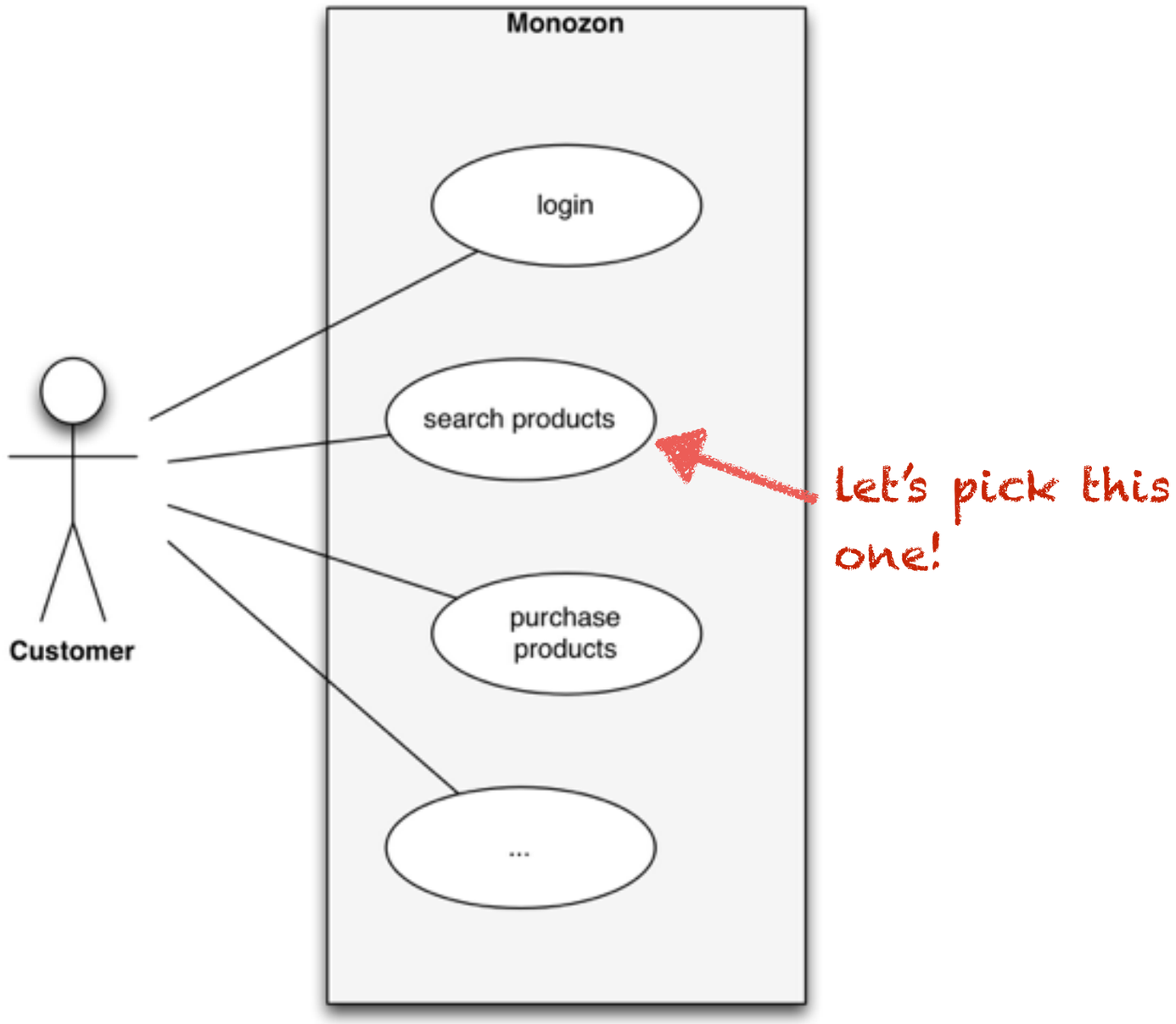


http://martinfowler.com/bliki/CircuitBreaker.html

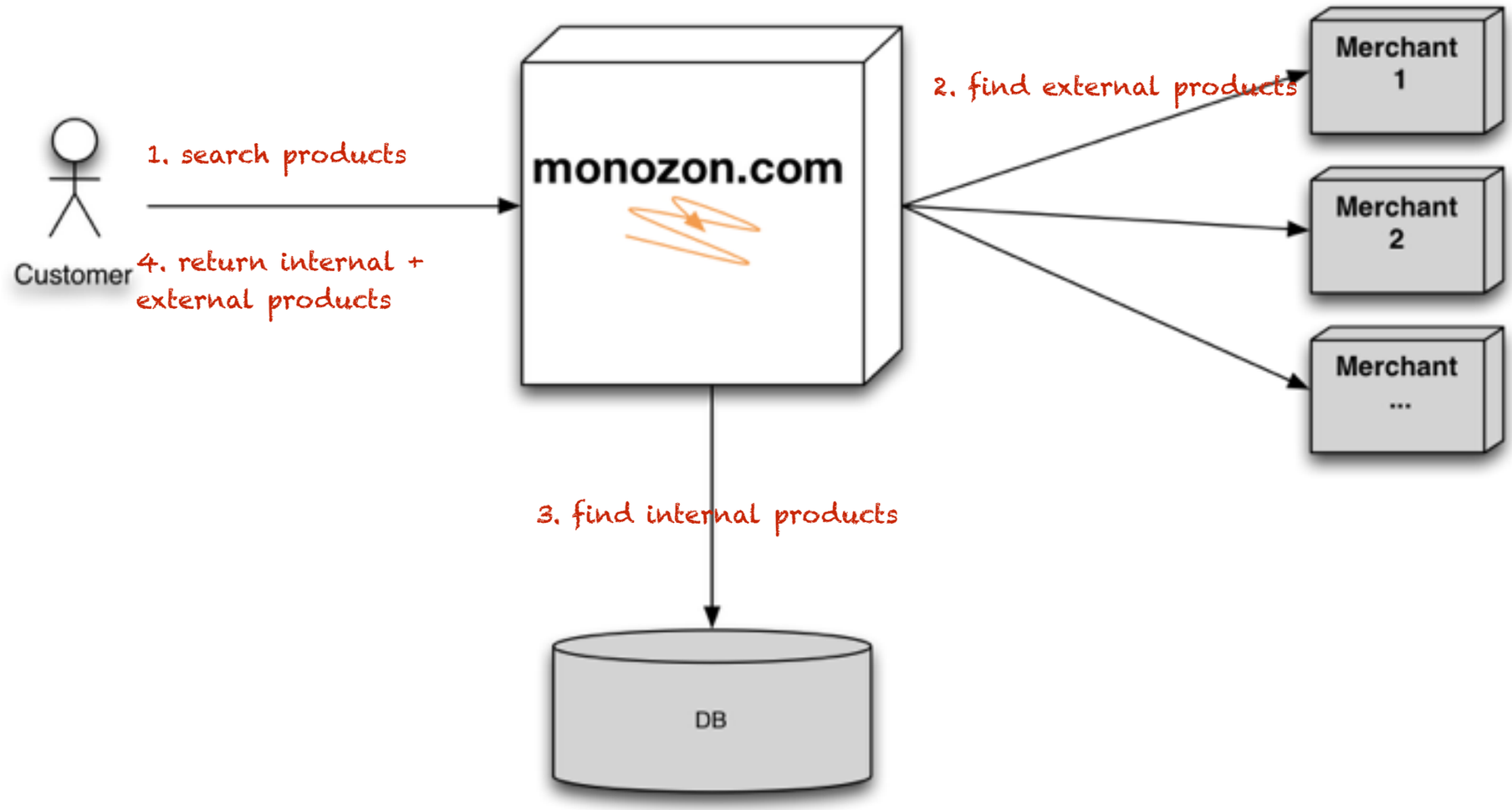
Hystrix

- Library from Netflix
- Resilience Library
- Command Pattern
- Metrics
- Dashboard

Use Cases


Monozon
login
search products
purchase products
...
Customer
let's pick this one!

Search Products



Search Products

```
➜ /  http GET http://monozon:8080/products | jq '.'
[
  "internalProduct_1",
  "internalProduct_2",
  "merchant_1",
  "merchant_2",
  "merchant_3",
  "merchant_4"
]
```

Call without Hystrix

```
private List<Product> findProducts(String query) {
    ClientResponse clientResponse =
            Client.create()
                    .resource("http://merchant1/products/")
                    .queryParam("query", query)
                    .get(ClientResponse.class);

    return toProduct(clientResponse);
}
```

cascading failures incoming!

Simple Command

```
public class GetMerchant1Products
        extends HystrixCommand<List<Product>> {

    private final String query;

    public GetMerchant1Products(String query) {
        super(HystrixCommandGroupKey.Factory.asKey("merchant1"));
        this.query = query;
    }

    @Override
    protected List<Product> run() throws Exception {
        return findProducts(query);
    }
}
```

Execute it!

```
public List<Product> findExternalProducts(String query) {

    List<Product> productList =
            new GetMerchant1Products(query).execute();

    List<Product> merchant2Products =
            new GetMerchant2Products(query).execute();

    productList.addAll(merchant2Products);

    return productList;
}
```

Execute it asynchronously

```
public List<Product> findExternalProducts(String query) {

    Future<List<Product>> merchant1ProductsFuture =
            new GetMerchant1Products(query).queue();

    Future<List<Product>> merchant2ProductsFuture =
            new GetMerchant2Products(query).queue();

    List<Product> productList = new ArrayList<>();
    productList.addAll(merchant1ProductsFuture.get());
    productList.addAll(merchant2ProductsFuture.get());

    return productList;
}
```

Fallback

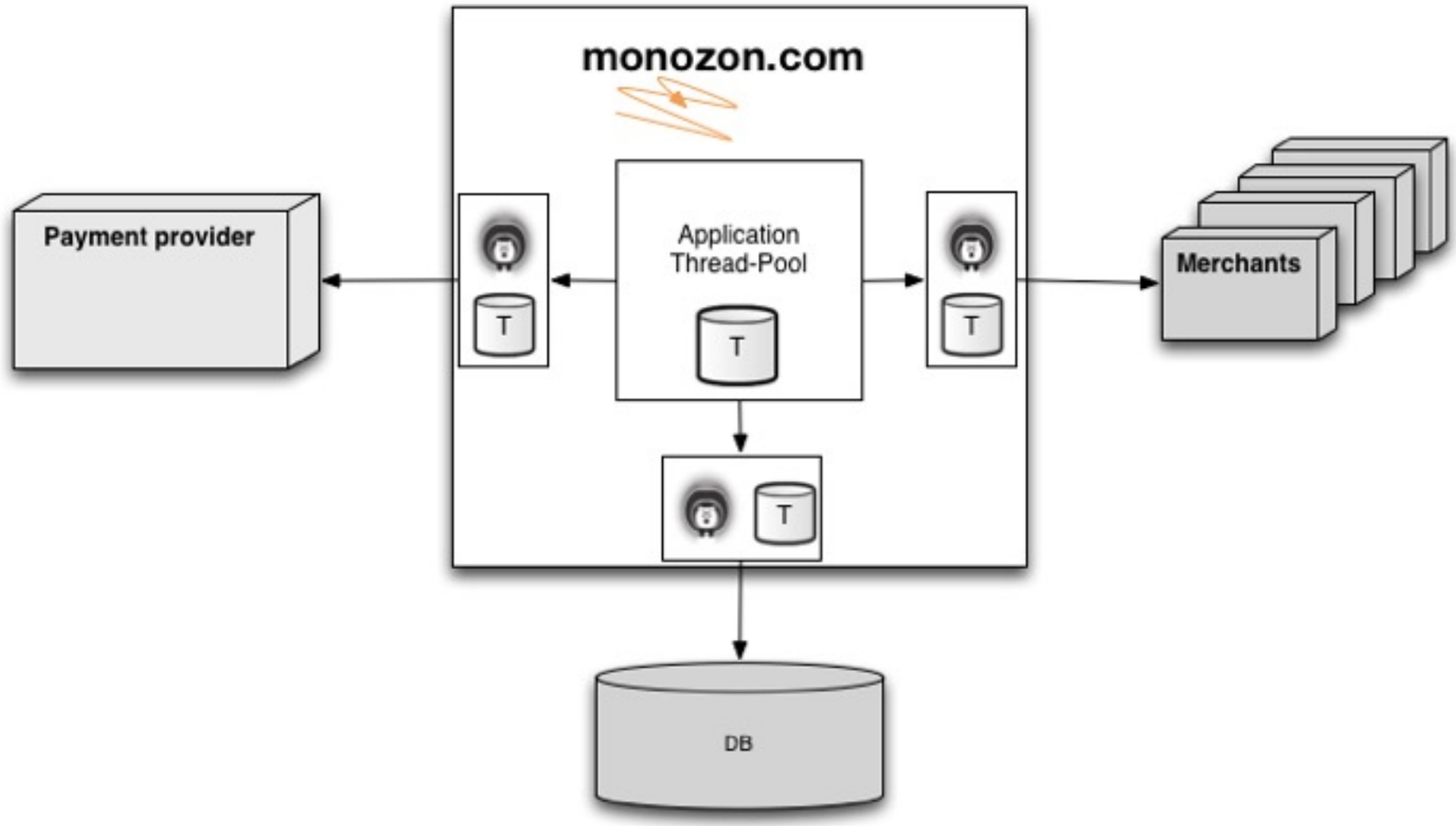
```
@Override
protected List<Product> run() throws Exception {
    return findProducts(query);
}

@Override
protected List<Product> getFallback() {
    return Collections.emptyList();
}
```

In case Merchant 2 is down

```
➜ /  http GET http://monozon:8080/products | jq '.'
[
  "internalProduct_1",
  "internalProduct_2",
  "merchant_1",
  "merchant_3",
  "merchant_4"
]
```

something is missing here

The stabilized system



Demo

And now?

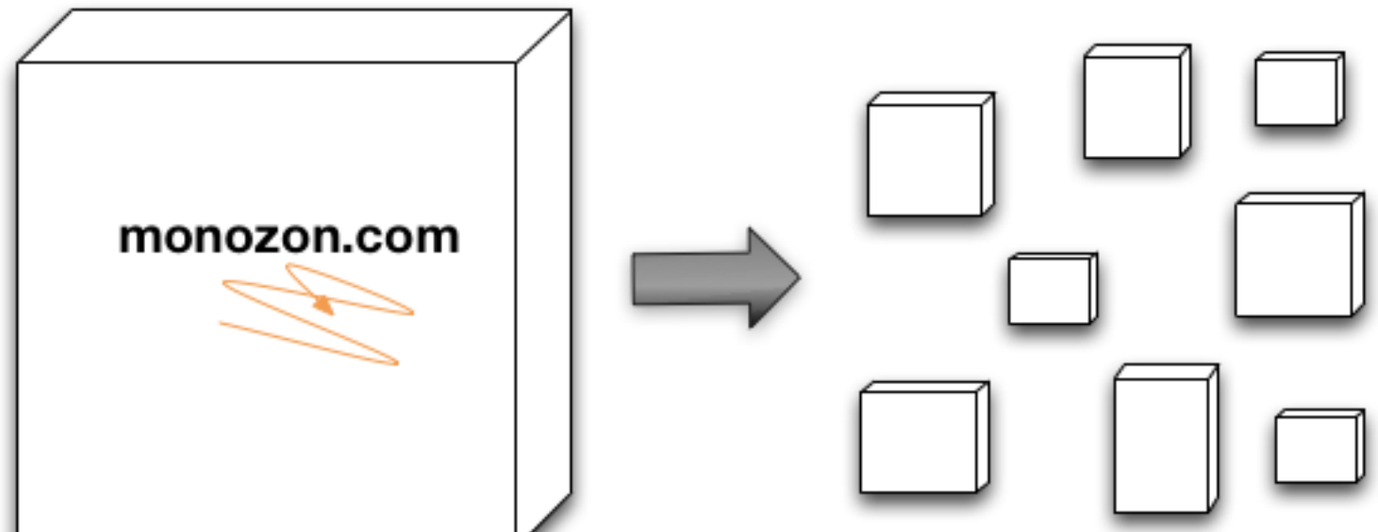
Current Problems

- Maintenance is difficult
- New features need a lot of time
- ~~Very unstable~~ => enables further distribution
- Outdated technology
- Doesn’t scale

We need a clear cut!

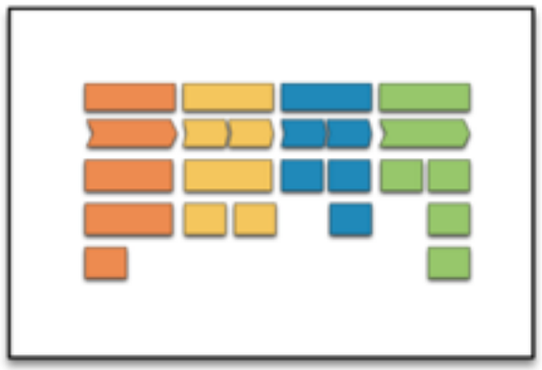
https://www.flickr.com/photos/taefit/8528632756

Microservices!



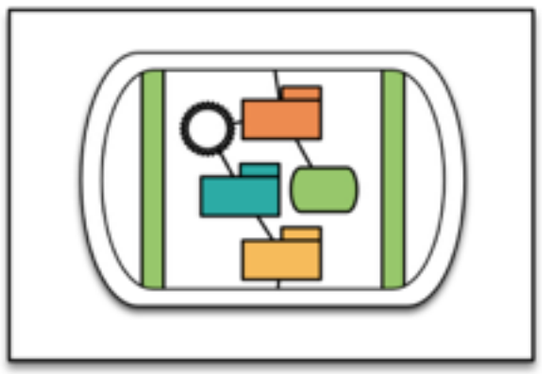
but how to get started ????

Architectural Decisions



› Domain Architecture

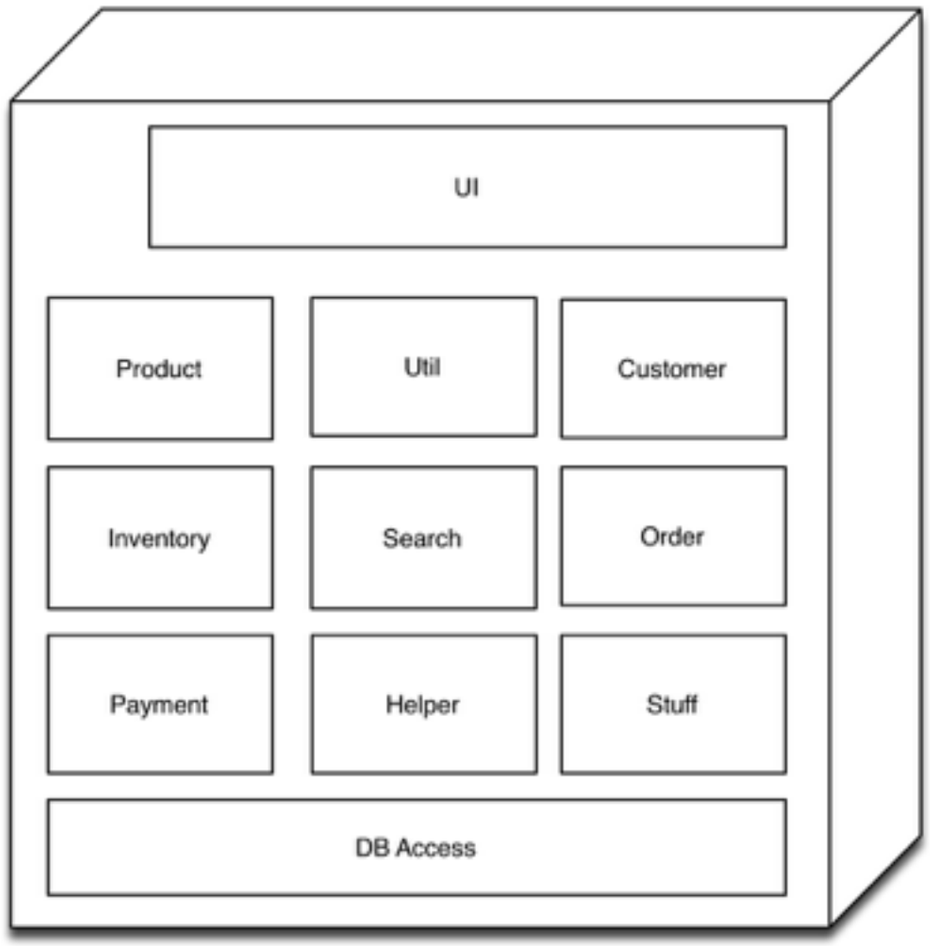
› Macro Architecture

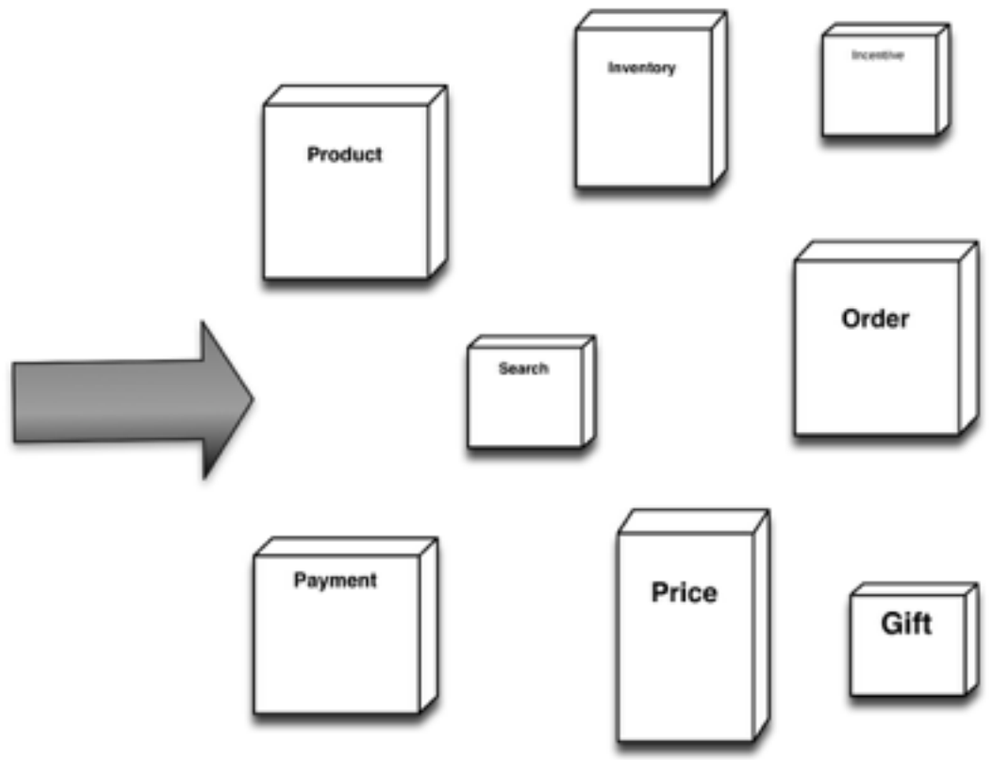


› Micro Architecture

Domain Architecture

which boxes do we need ?





let the monolith guide you!

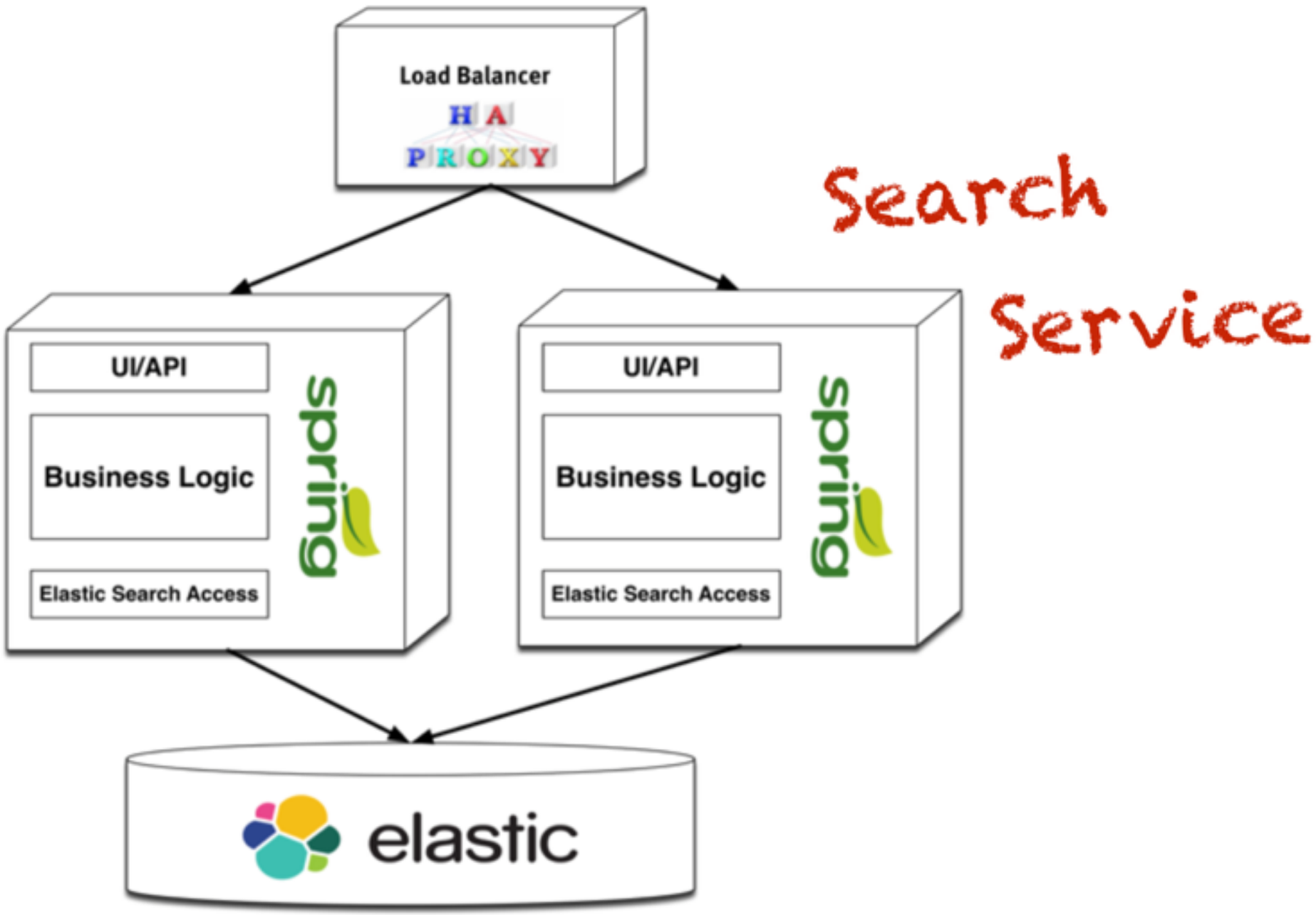
Macro Architecture

what's the same for all boxes ?

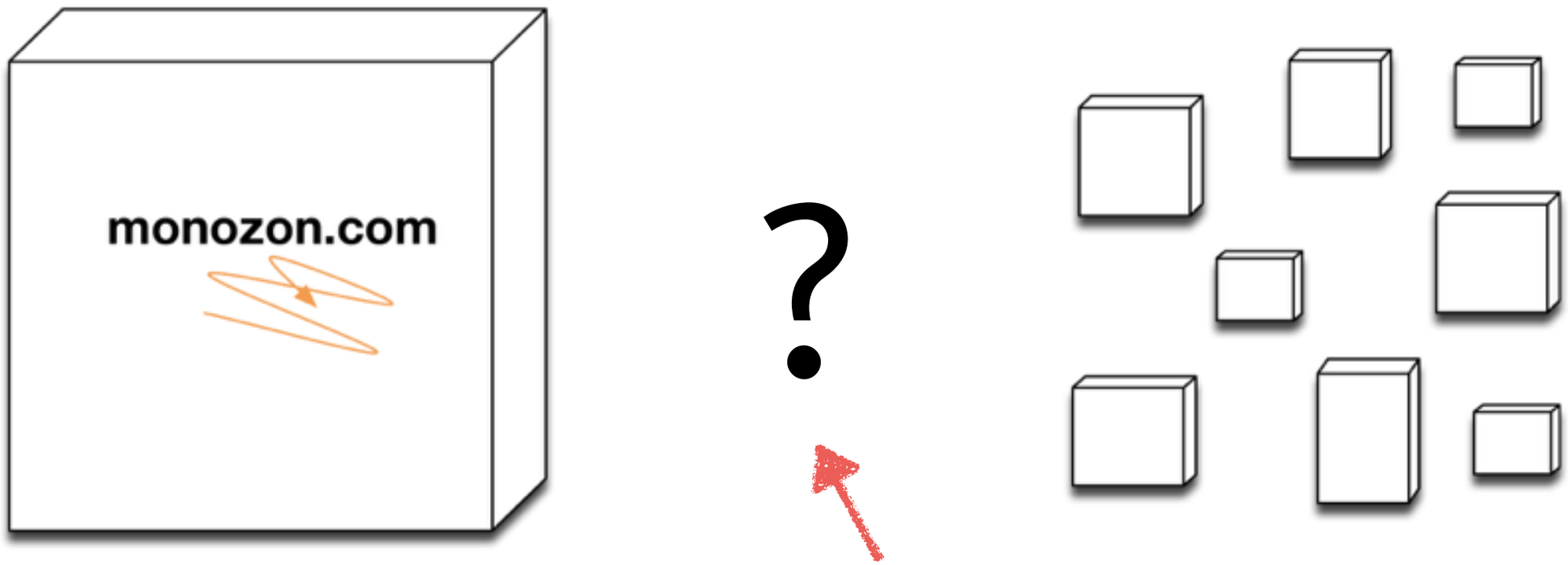
	Monozon:
› Integration	= API + UI
› Deployment	= Docker
› Formats	= JSON
› Protocols	= HTTP + AMQP
› Reduce Choices	= Java, Go

pick your own!

Micro Architecture



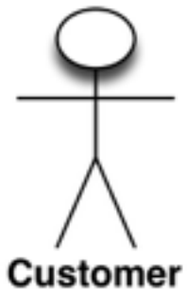
Migration Path



Big Bang

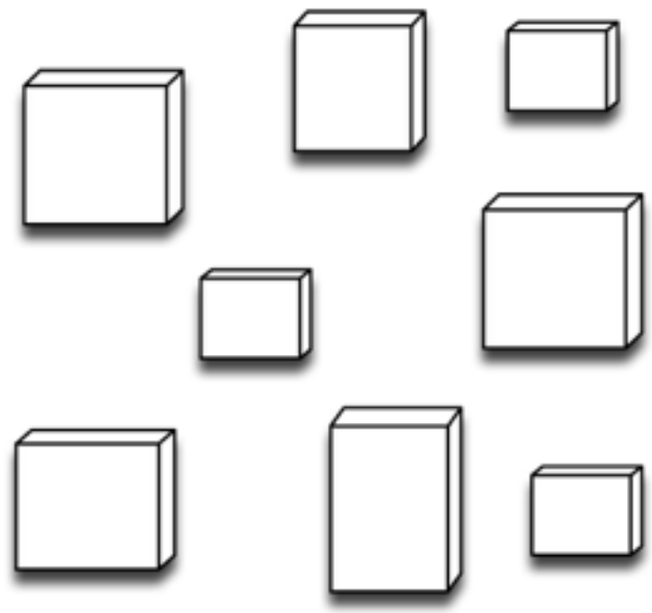
a.k.a REVOLUTION

Step 1



only call old monolith

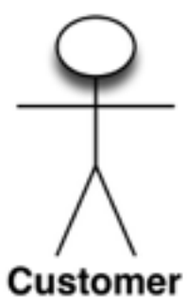
meanwhile build new systems:



Big Bang

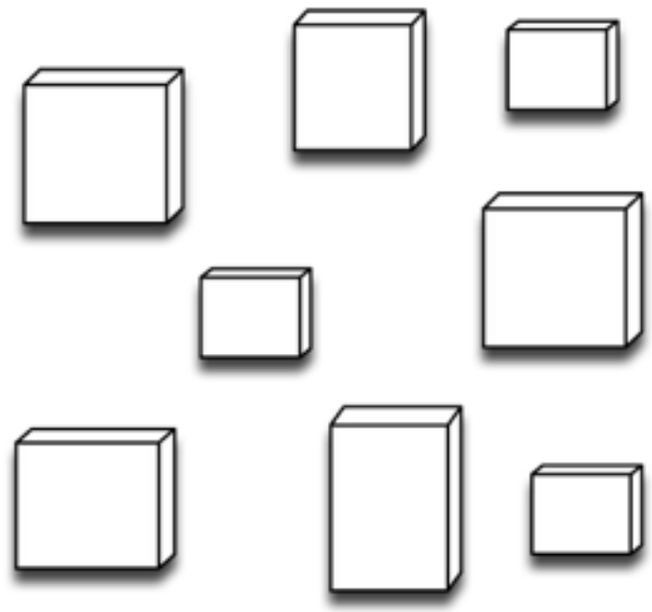
a.k.a REVOLUTION

Step2



only call new shiny systems

delete this one!

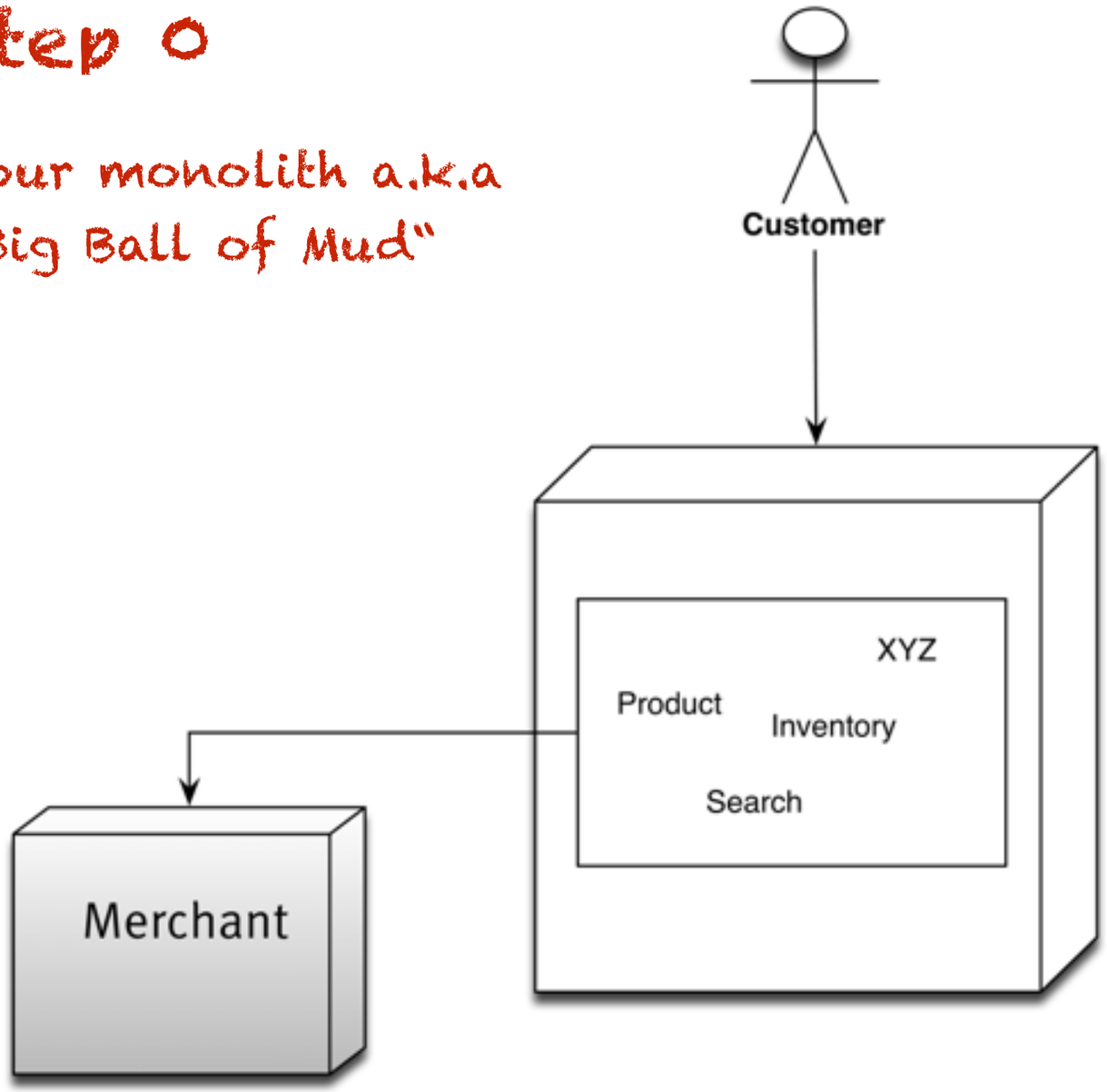


Strangler

a.k.a EVOLUTION

Step 0

your monolith a.k.a
„Big Ball of Mud“



Strangler

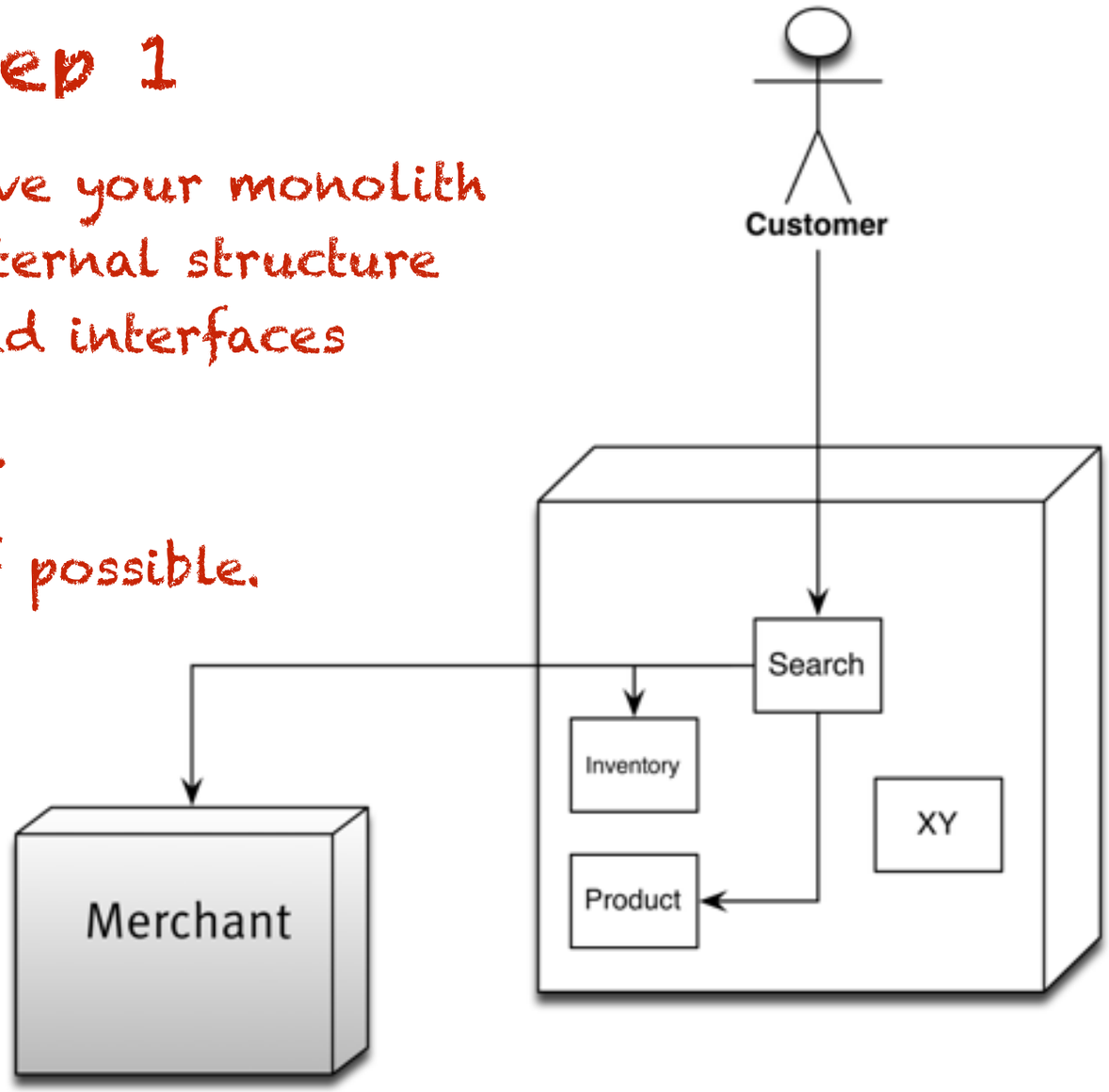
a.k.a EVOLUTION

Step 1

give your monolith
internal structure
and interfaces

...

if possible.

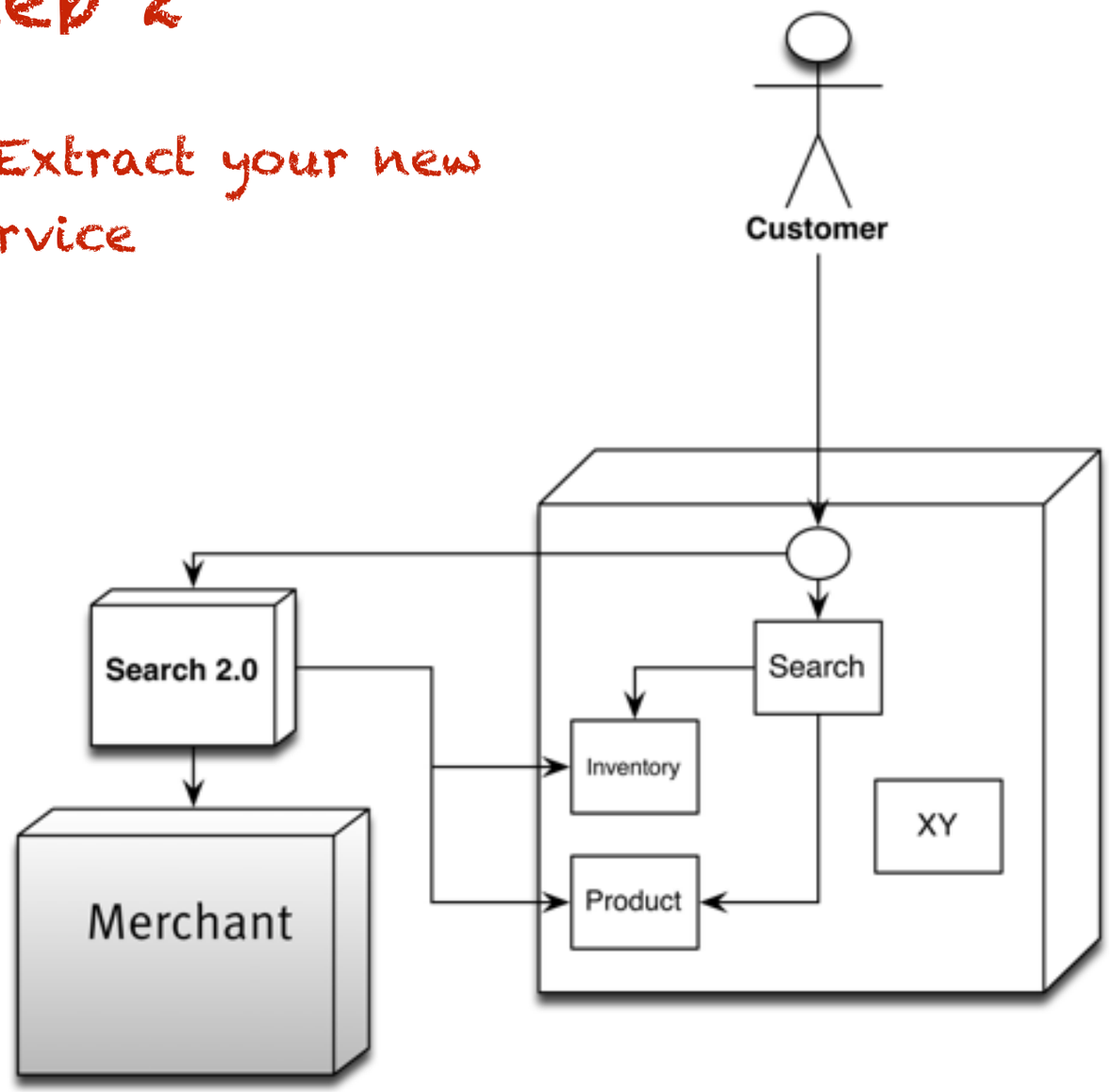


Strangler

a.k.a EVOLUTION

Step 2

- Extract your new Service

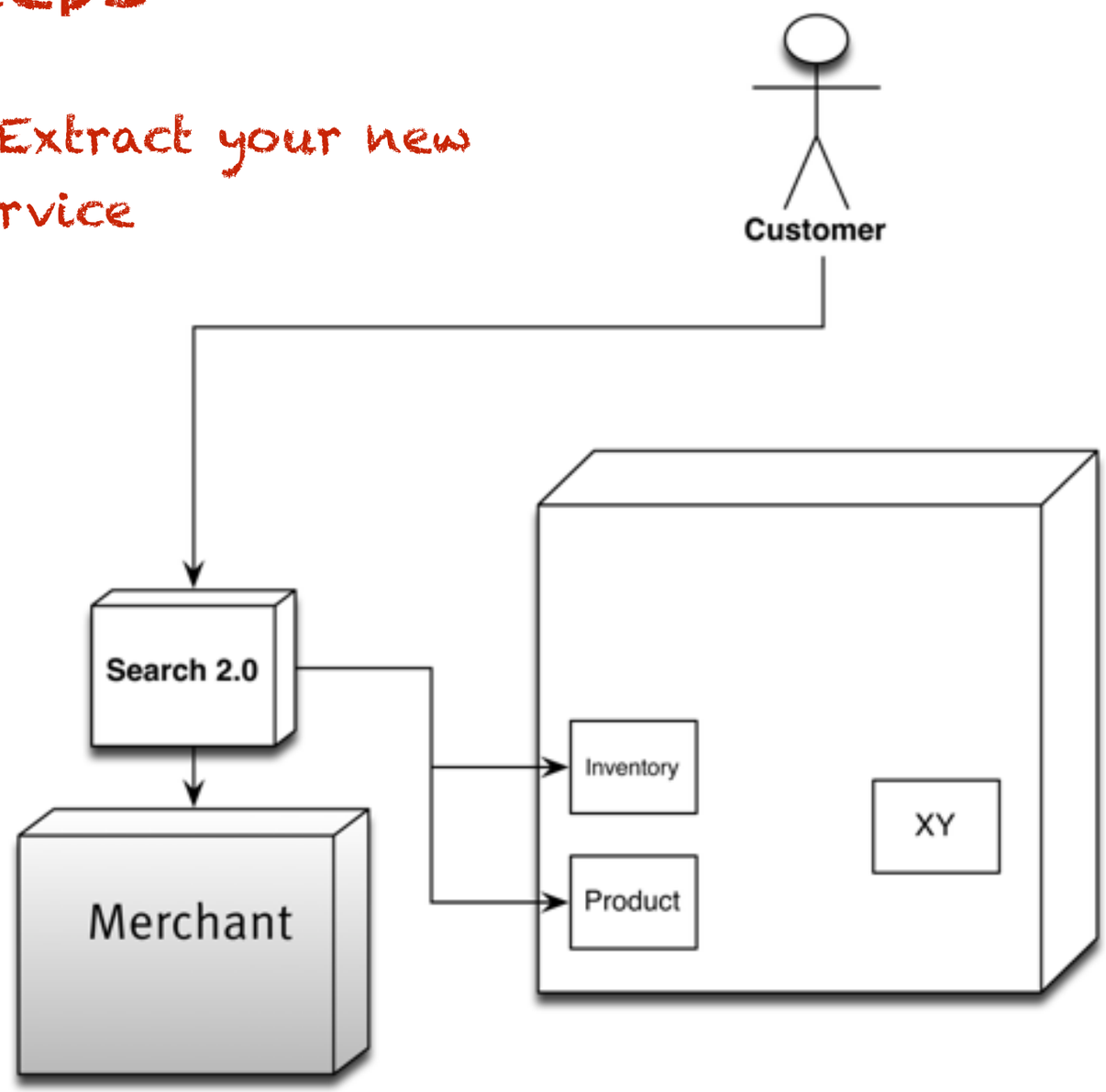


Strangler

a.k.a EVOLUTION

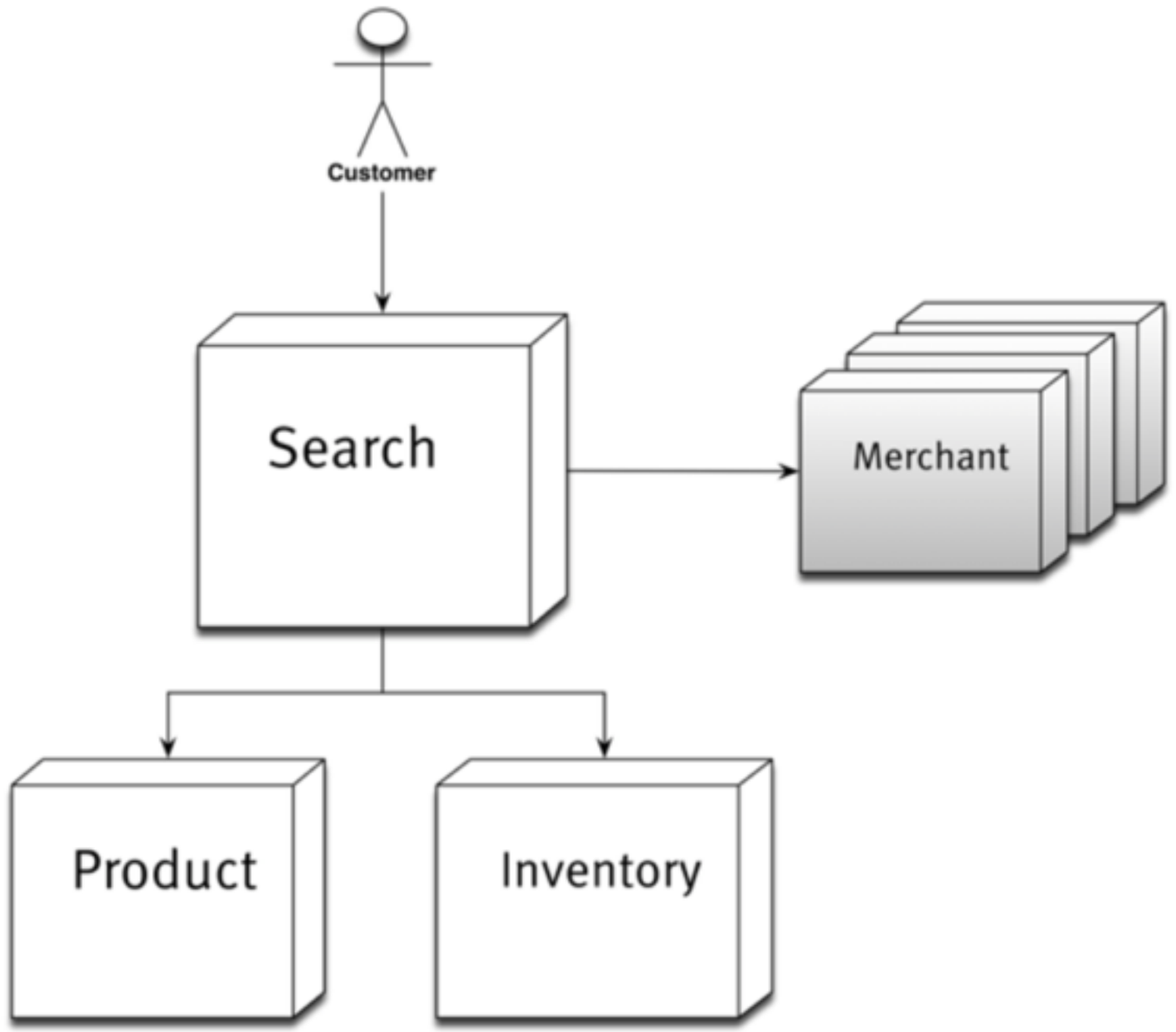
Step3

- Extract your new Service

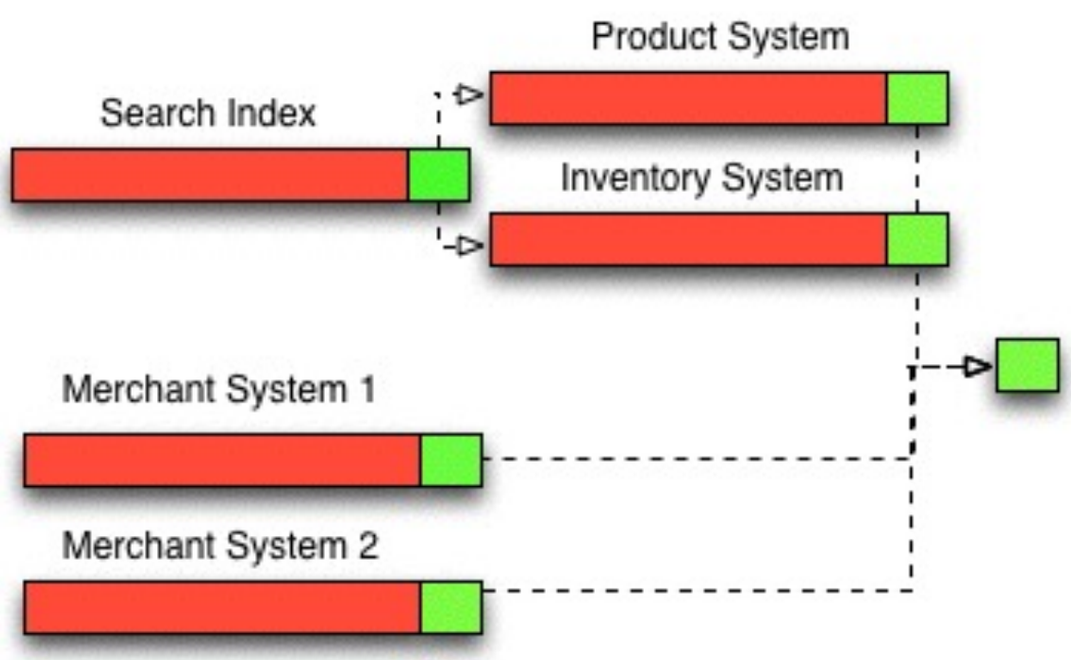


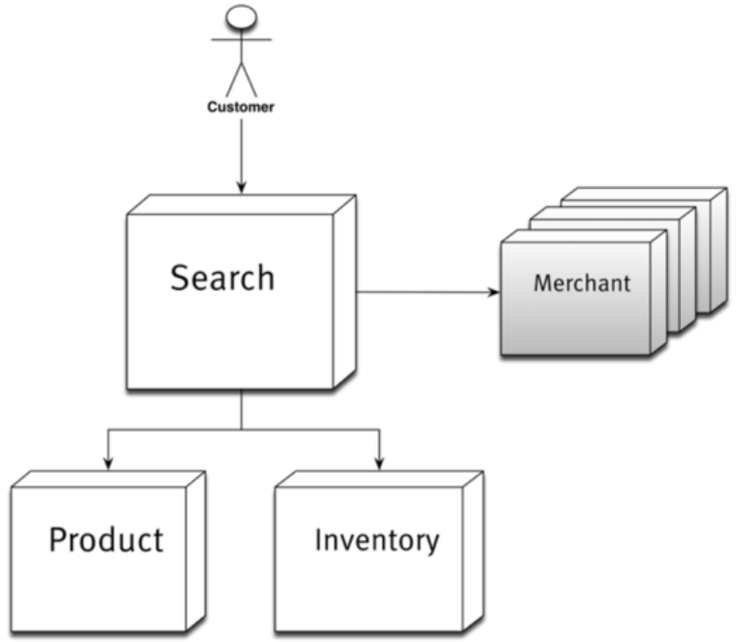
Is it really so easy?

Time to connect boxes



Synchronous vs. Async/Parallel





Try this with futures

```
Future<List<Long>> productIndexFuture =
        getProductIndexFuture(query);

List<Long> productIdList = productIndexFuture.get();

List<FuturePair> futurePairList = new ArrayList<>();
for (Long productId : productIdList) {
    Future<Product> productFuture = retrieveProductFromProductSystem(productId);
    Future<Long> quantityFuture = retrieveQuantityFromInventoryService(productId);
    futurePairList.add(new FuturePair(productFuture, quantityFuture));
}

List<SearchResult> searchResultList = new ArrayList<>();
for (FuturePair futurePair : futurePairList) {
    Product product = futurePair.getProductFuture().get();
    Long quantity = futurePair.getQuantityFuture().get();
    SearchResult searchResult = new SearchResult(product, quantity);
    searchResultList.add(searchResult);
}
```

Time for RxJava

- Reactive Extensions for the JVM
- Asynchronous streams

- Elements of
  - Iterator pattern
  - Observable pattern
  - Functional programming

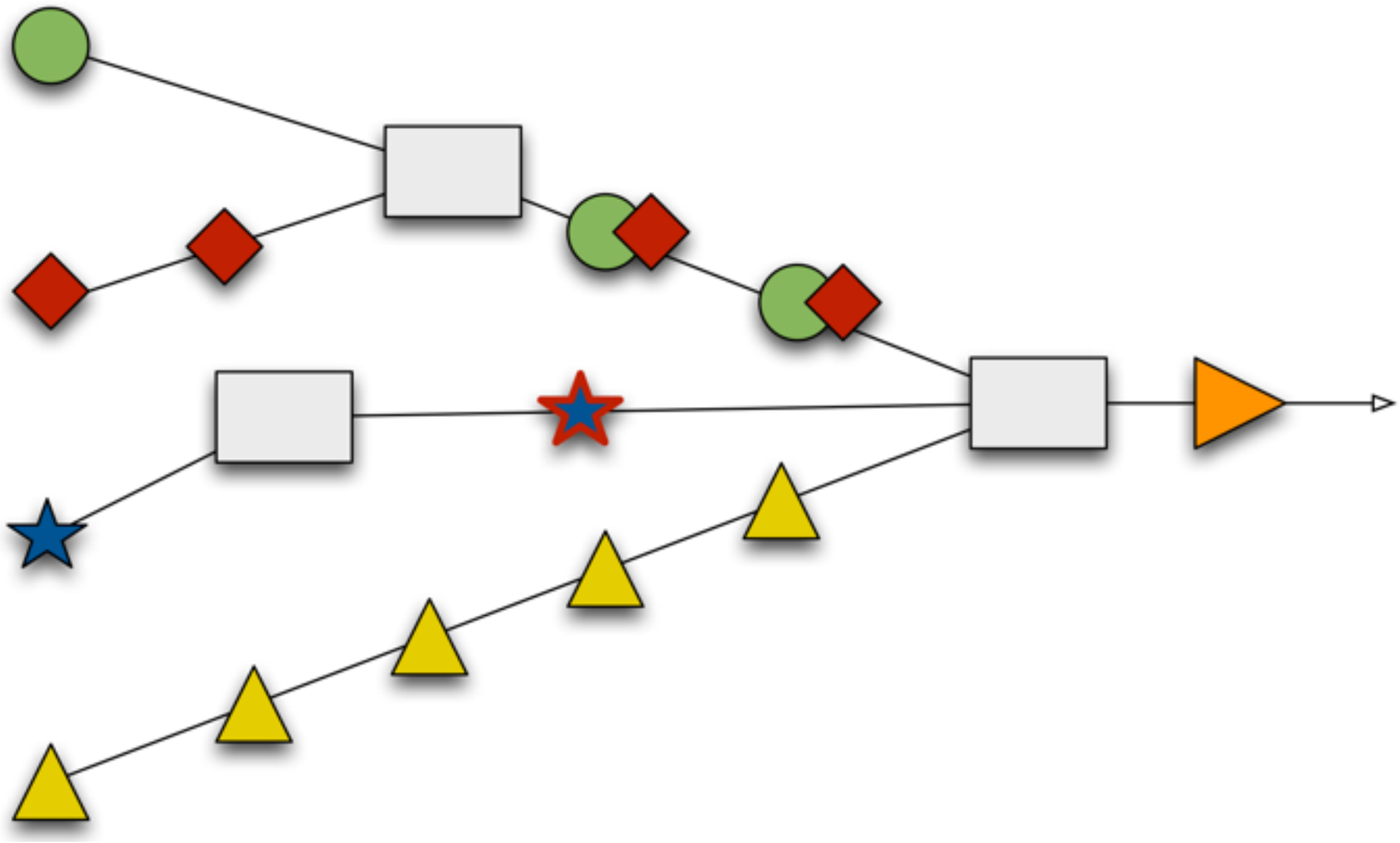
Iterable	Observable
pull	*push*
T next()	onNext(T)
throws Exception	onError(Exception)
returns;	onCompleted()

https://speakerdeck.com/benjchristensen/functional-reactive-programming-with-rxjava-javaone-2013

Everything is a stream!

https://www.flickr.com/photos/gruesome/3010529489

RxJava in one picture



https://speakerdeck.com/benjchristensen/functional-reactive-programming-with-rxjava-javaone-2013

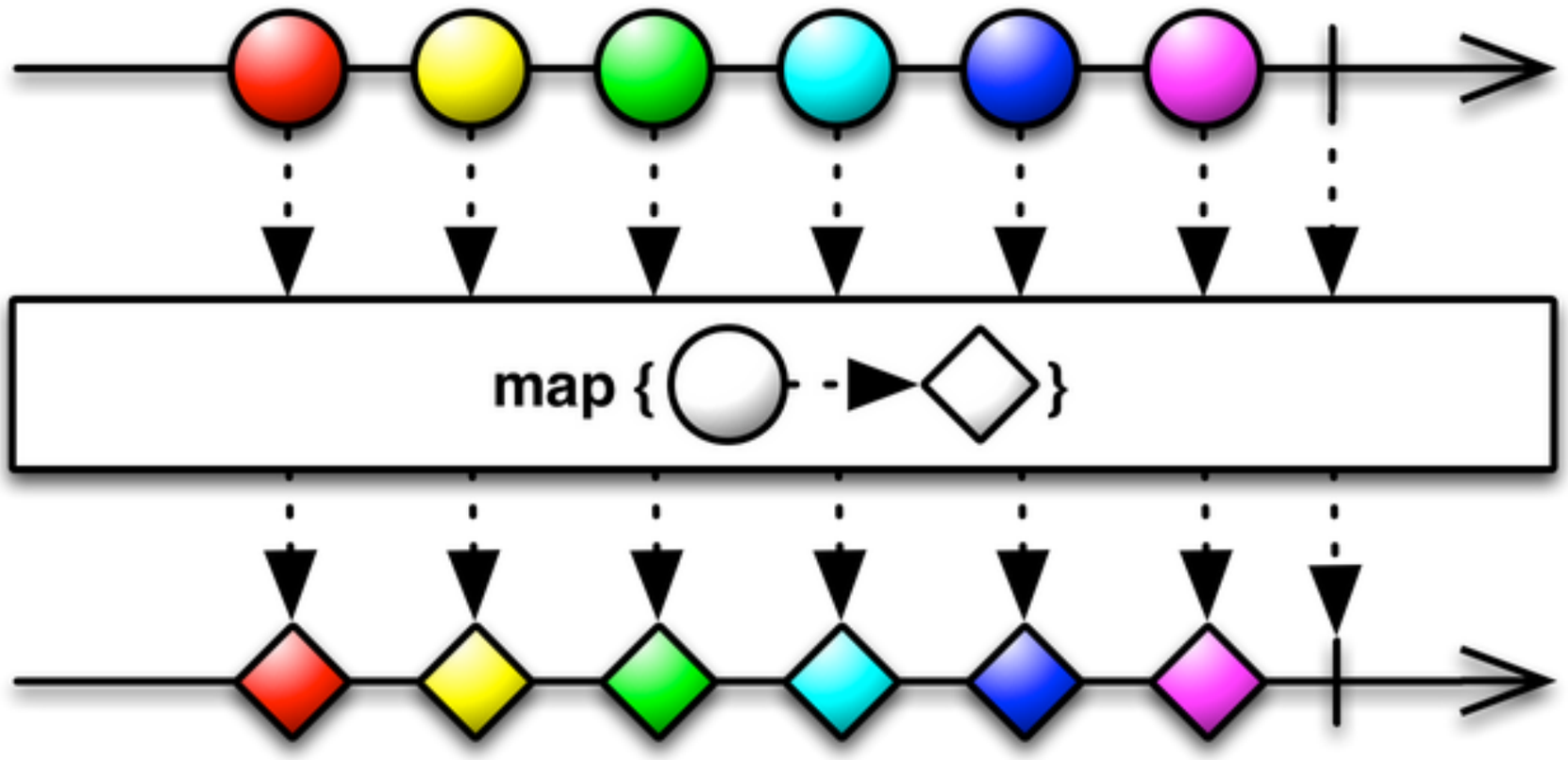
Creating Observables

```
Observable.just("Book A", "Book B", "Book C");
```

```
Observable.from(findProducts(query));
```

```
Observable.create(o -> {
    for (Merchant2Product product : findProducts(query)) {
        o.onNext(product);
    }
    o.onCompleted();
});
```

Transforming with map

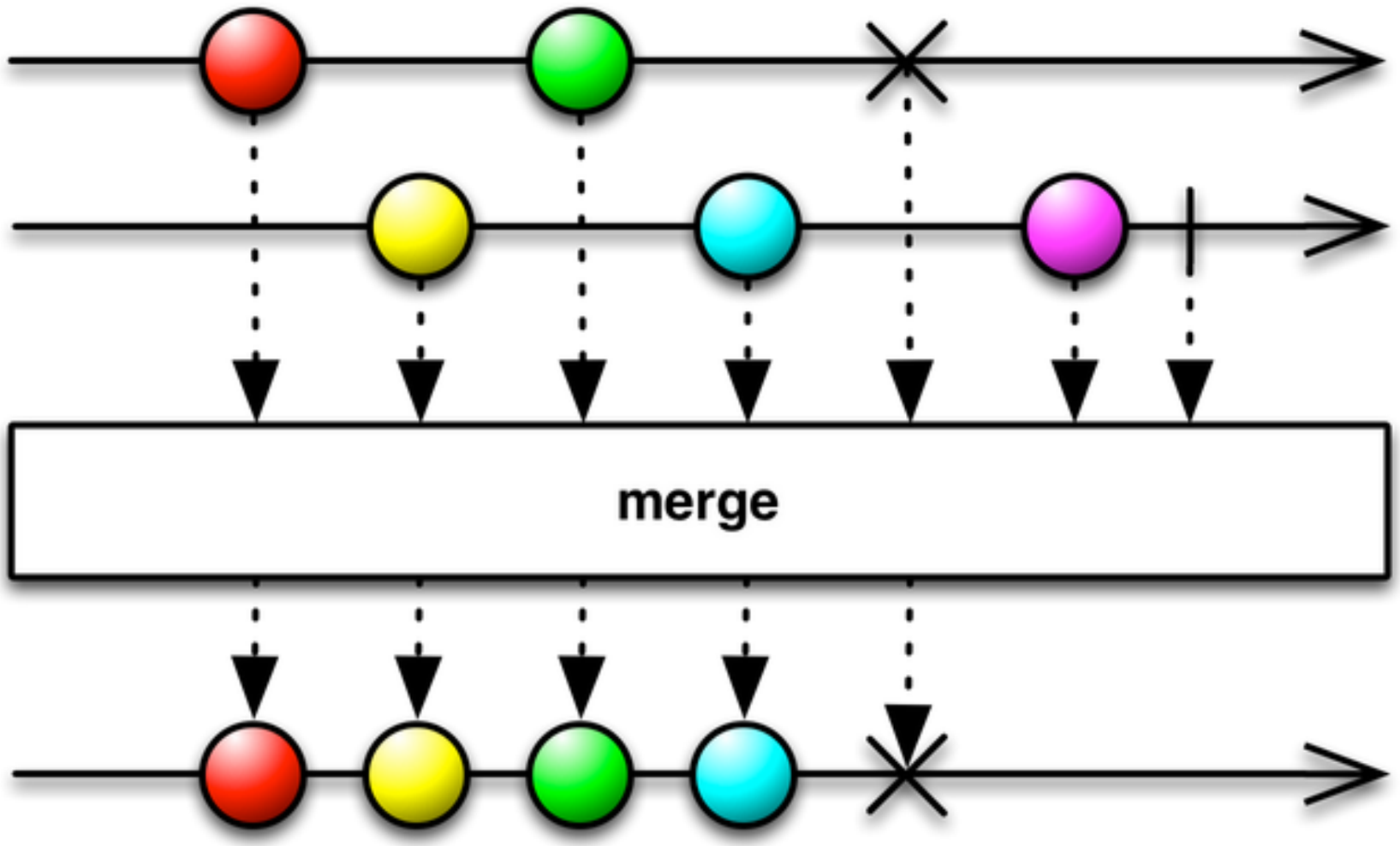


http://reactivex.io/RxJava/javadoc/rx/Observable.html

map in action

```
Observable<Merchant2Product> productM2Observable =
        Observable.from(findProducts(query));

Observable<SearchResult> searchResultM2Observable =
        productM2Observable.map(this::toSearchResult);
```

Combining with merge

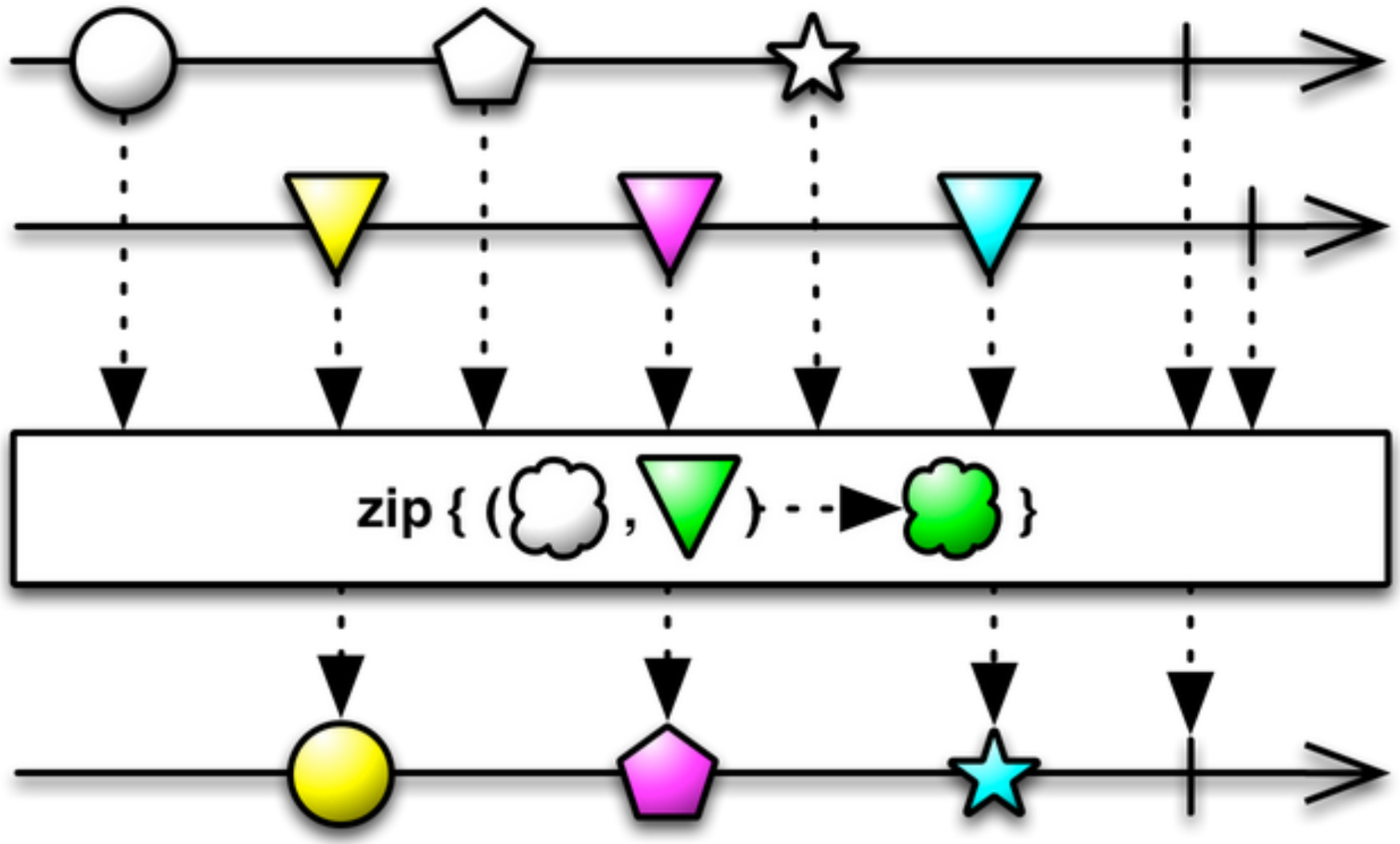


http://reactivex.io/RxJava/javadoc/rx/Observable.html

merge in action

```
Observable<SearchResult> mergedSearchResultObservable =
    searchResultM1Observable.mergeWith(searchResultM2Observable);
```

Combining streams with zip

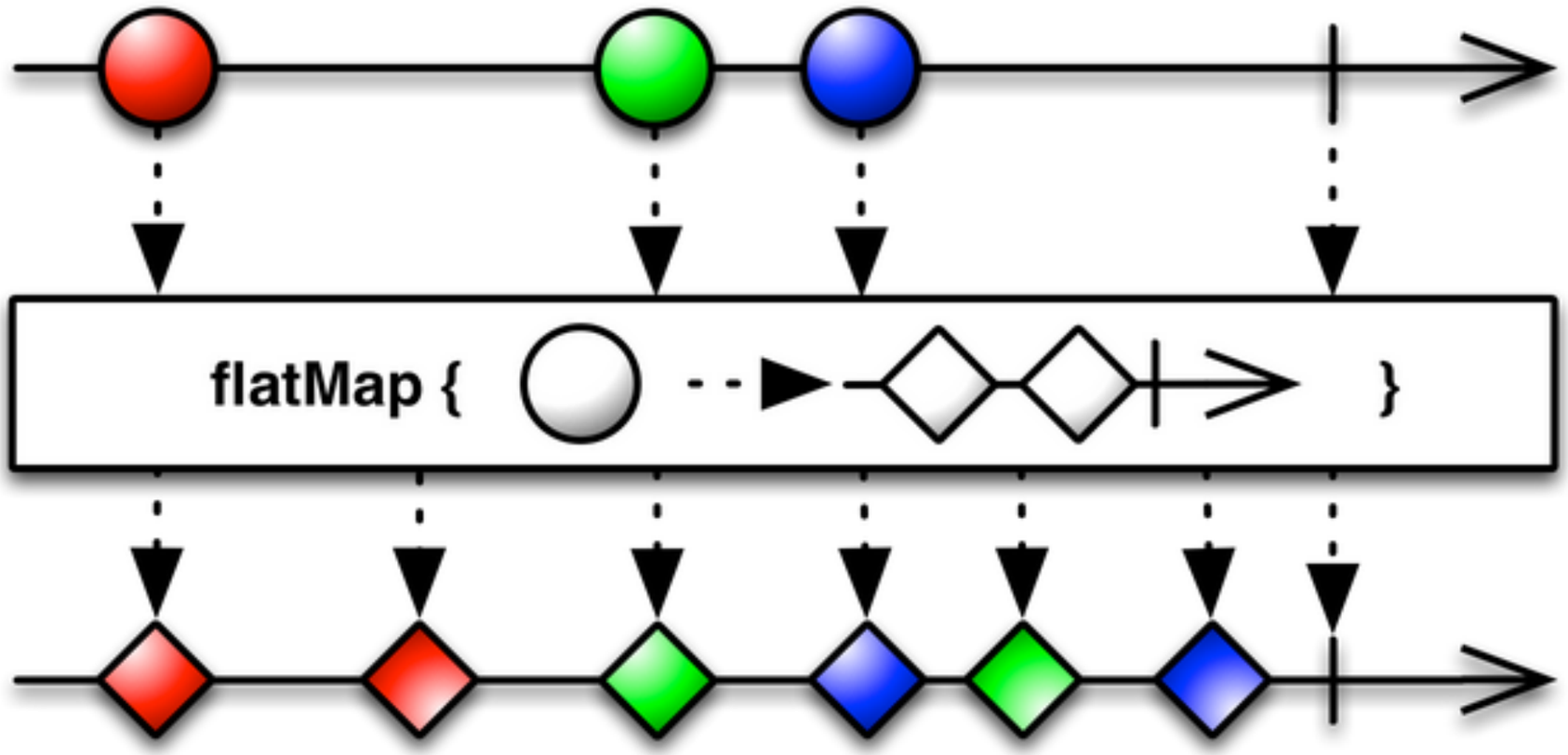


http://reactivex.io/RxJava/javadoc/rx/Observable.html

zip in action

```
private Observable<SearchResult> productDetails(Long productId) {

    Observable<Product> productObservable =
            retrieveProductFromProductSystem(productId);

    Observable<Long> quantityObservable =
            retrieveQuantityFromInventoryService(productId);

    return Observable.zip(productObservable,
            quantityObservable, SearchResult::new);
}
```

Collecting details with flatMap



http://reactivex.io/RxJava/javadoc/rx/Observable.html

flatMap in action

```
public Observable<SearchResult> findInternalProducts(String query) {

    Observable<Long> productIndexObservable =
            getProductIndexObservable(query);

    return productIndexObservable
            .flatMap(this::productDetails);

}
```

Why not map?

```
public Observable<Observable<SearchResult>> findInternalProducts(String query) {

    Observable<Long> productIndexObservable =
            getProductIndexObservable(query);

    return productIndexObservable
            .map(this::productDetails);

}
```

Concurrency

```
Observable<Merchant2Product> productM2Observable =
        Observable.from(findProducts(query))
        .subscribeOn(Schedulers.io());
```

Easier with Hystrix

```
public class GetMerchant2Products
        extends HystrixCommand<List<Merchant2Product>> {

    private final String query;

    public GetMerchant2Products(String query) {
        super(HystrixCommandGroupKey.Factory.asKey("merchant2"));
        this.query = query;
    }

    @Override
    public List<Merchant2Product> run() {
        return findProducts(query);
    }
```

Converting into a stream

```
Observable<List<Merchant2Product>> productM2ListObservable =
        new GetMerchant2Products(query).observe();

Observable<Merchant2Product> productM2Observable =
        productM2ListObservable.flatMap(Observable::from);
```

Returning a result

```
public List<SearchResult> getSearchResults(String query) {
    Observable<SearchResult> searchResults =
        searchService.findInternalProducts(query)
            .mergeWith(searchService.findExternalProducts2(query));

    Iterator<SearchResult> searchResultIterator
            = searchResults.toBlocking().getIterator();

    return Lists.newArrayList(searchResultIterator);
}
```

Summary

- Use Hystrix to stabilize your system!
- Use RxJava to increase the amount of async/ parallel processes in an easy way!
- Introduce Microservices to get control over your system again!
- Have fun :)

Thank you!

holger.kraus@innoq.com

Always worth a visit: innoq.com