leanovate
beyond product

MicroService Discovery

Bodo Junglas, York Xylander

Who is leanovate?

leanovate
beyond product

25 people, HQ in Kreuzberg

Mission: Build learning organizations

What do we do?

leanovate
beyond product

100

Dev

Meth

Doing

Coaching

Consulting

Training

Java,Scala, SPA,..

Kanban, Scrum, Lean*,..

Promises of µServices architectures

leanovate
beyond product

100

Challenges of µServices architectures

leanovate
beyond product

- Developing & Running
- Configuring
- Debugging
- Deploying
- Discovering
- Resilience
- …

Tough to learn & understand!

leanovate
beyond product

Microzon:

A lab for µServices

https://github.com/leanovate/microzon

Case Study: Microzon-Shop

leanovate
beyond product

1001

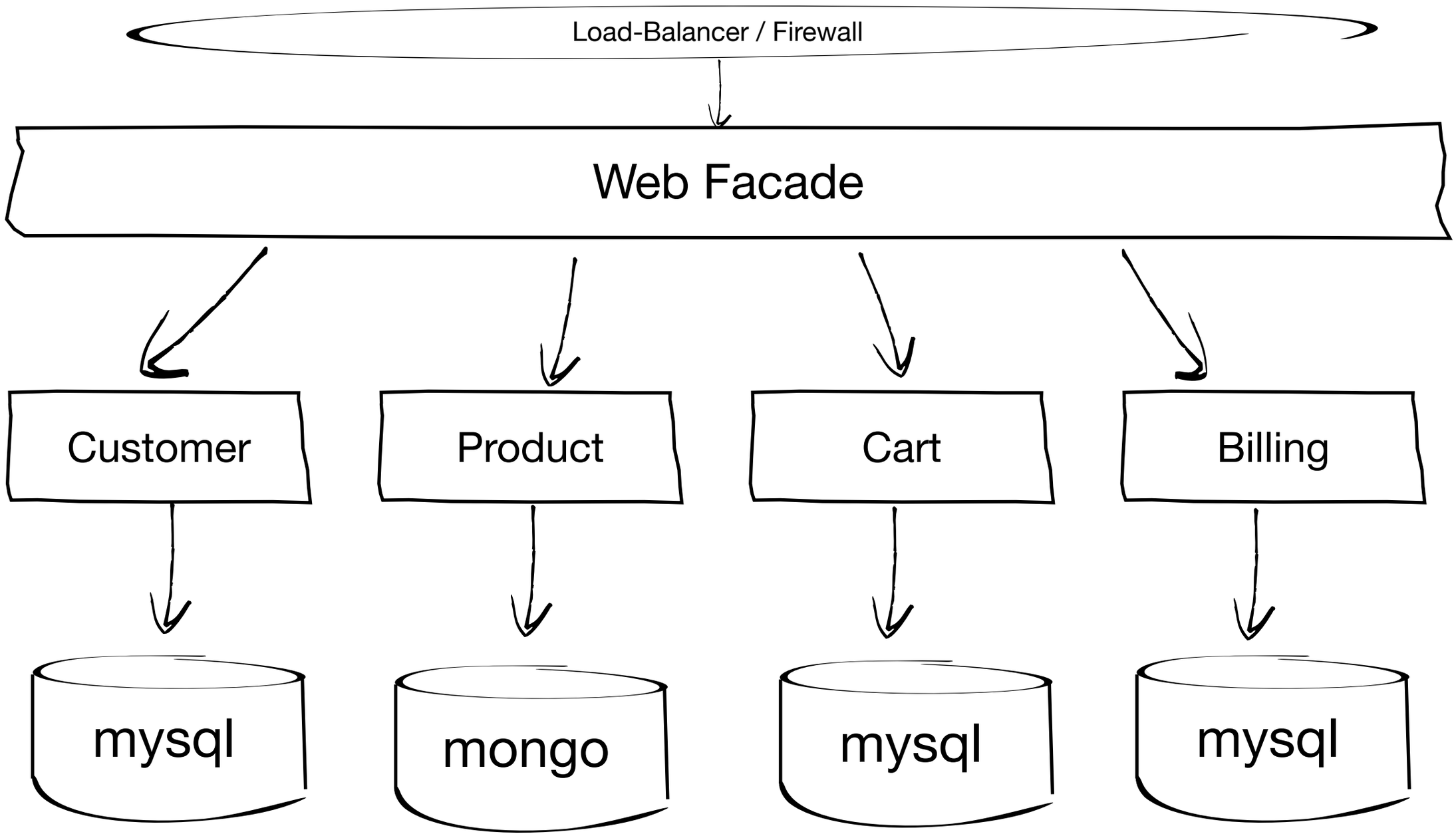
DEMO

- Browse through catalog
- Select product
- Put into cart
- Checkout

Case Study: Microzon-Shop

leanovate
beyond product

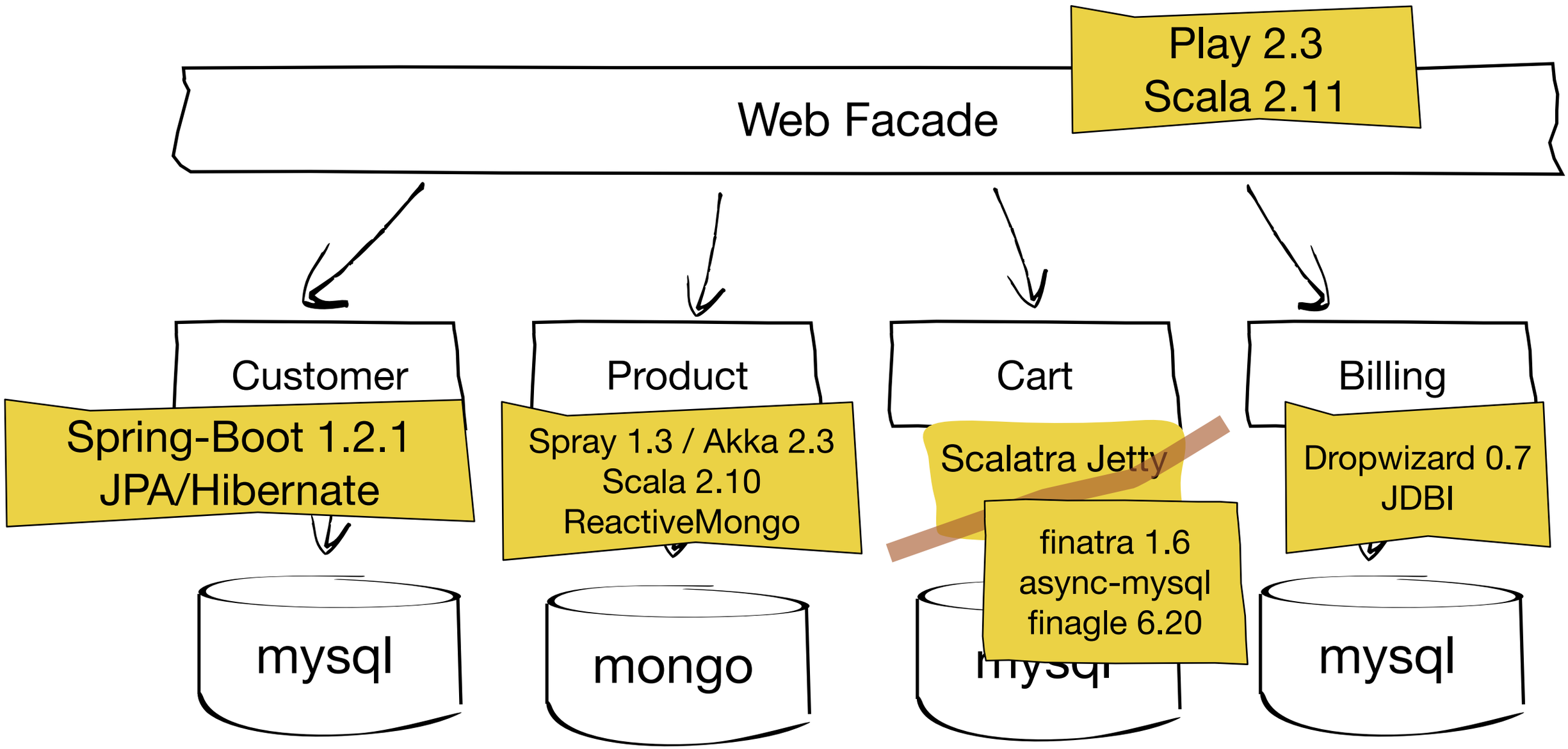
1001



Technology-Babylon

leanovate
beyond product

1001



Challenges:

- Running:
  - How long does it take to get a dev system up and running for a new team member?
    - How to run your ci system it in your favorite cloud?
- Configuring
- Debugging

Case Study: Microzon-Shop

leanovate
beyond product

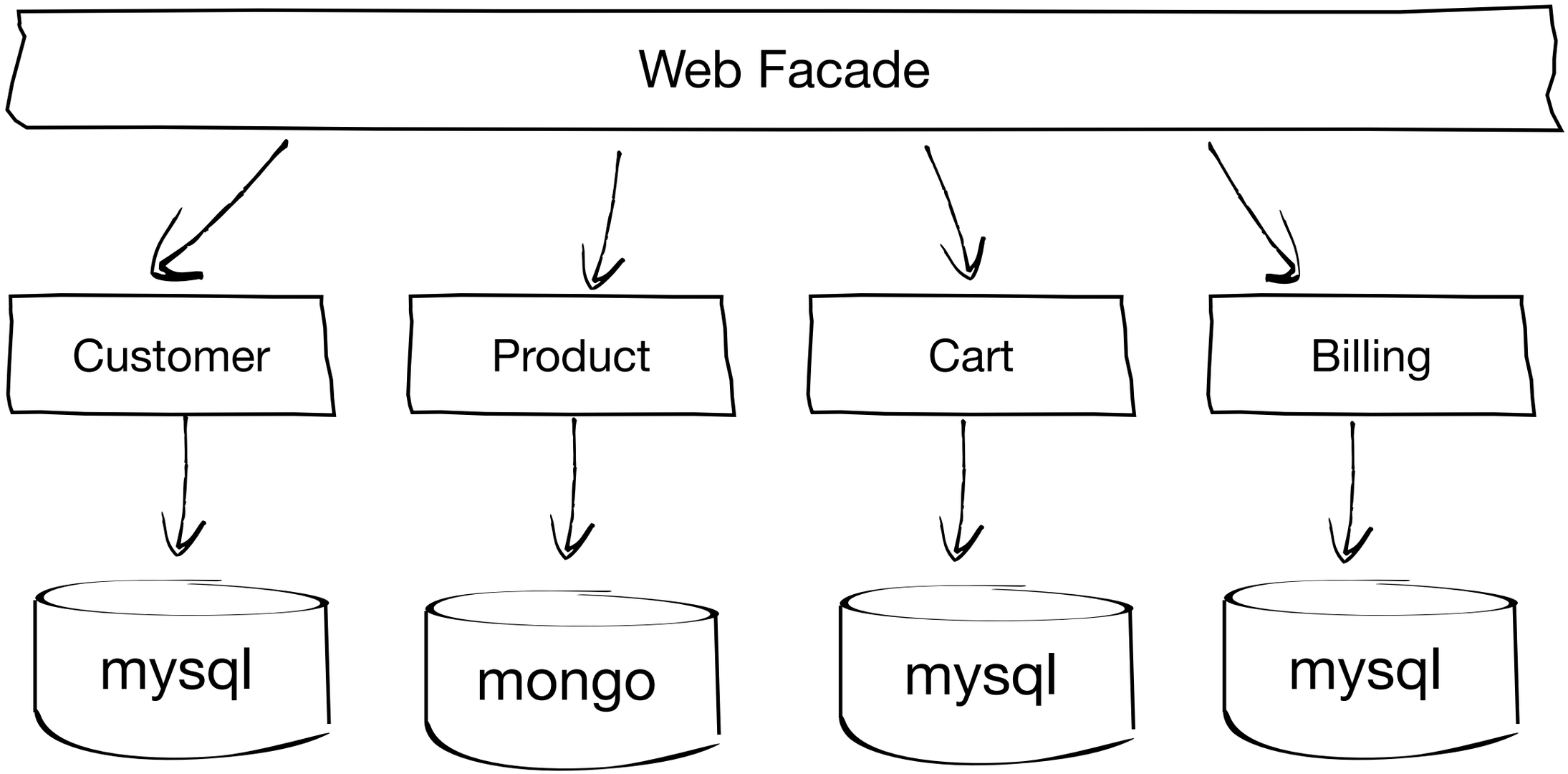
DEMO

- Start system with docker
- Create products
- Logstash
- Zipkin

Why is that a challenge?

leanovate
beyond product

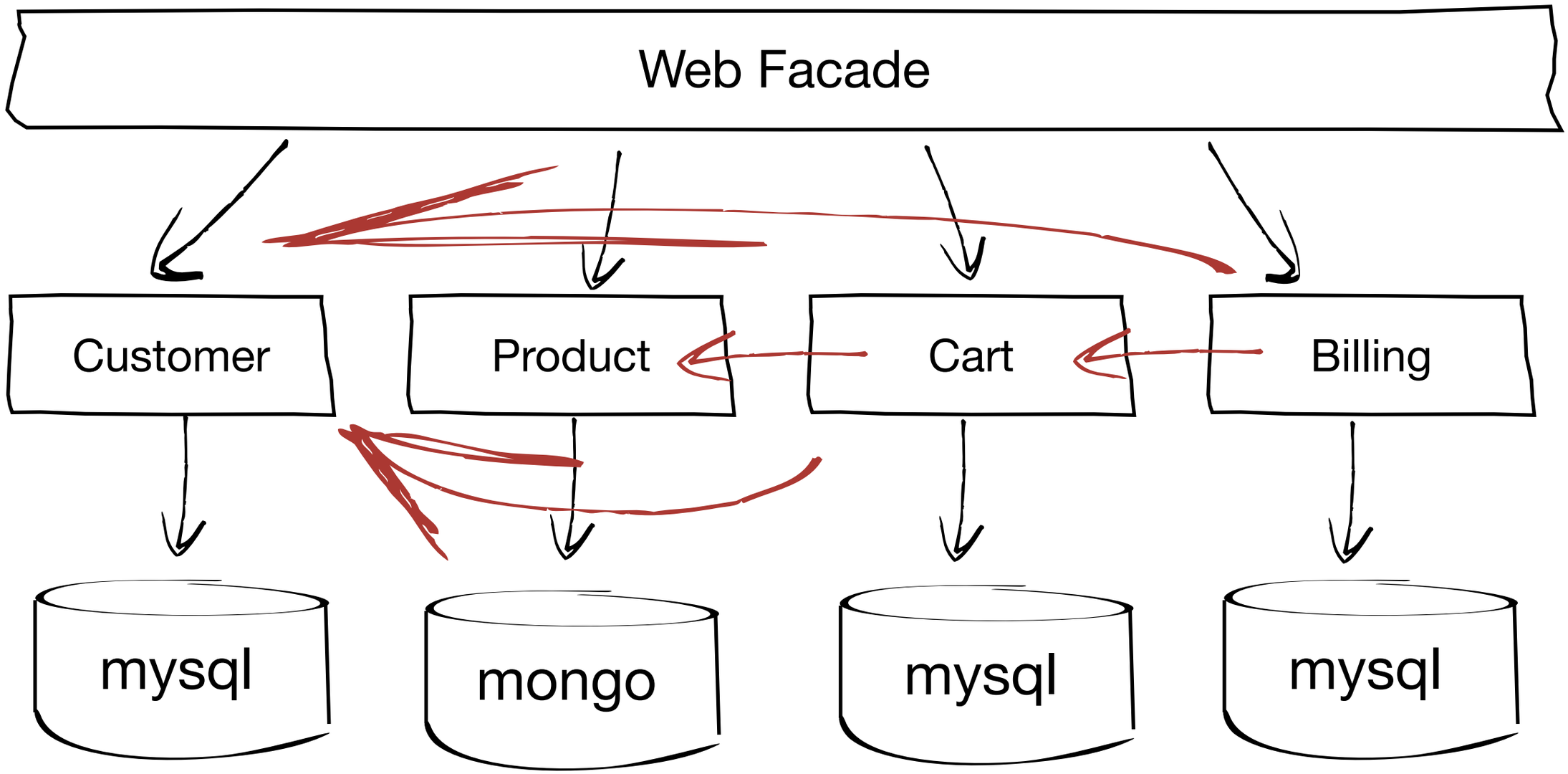
1001



Why is that a challenge?

leanovate
beyond product

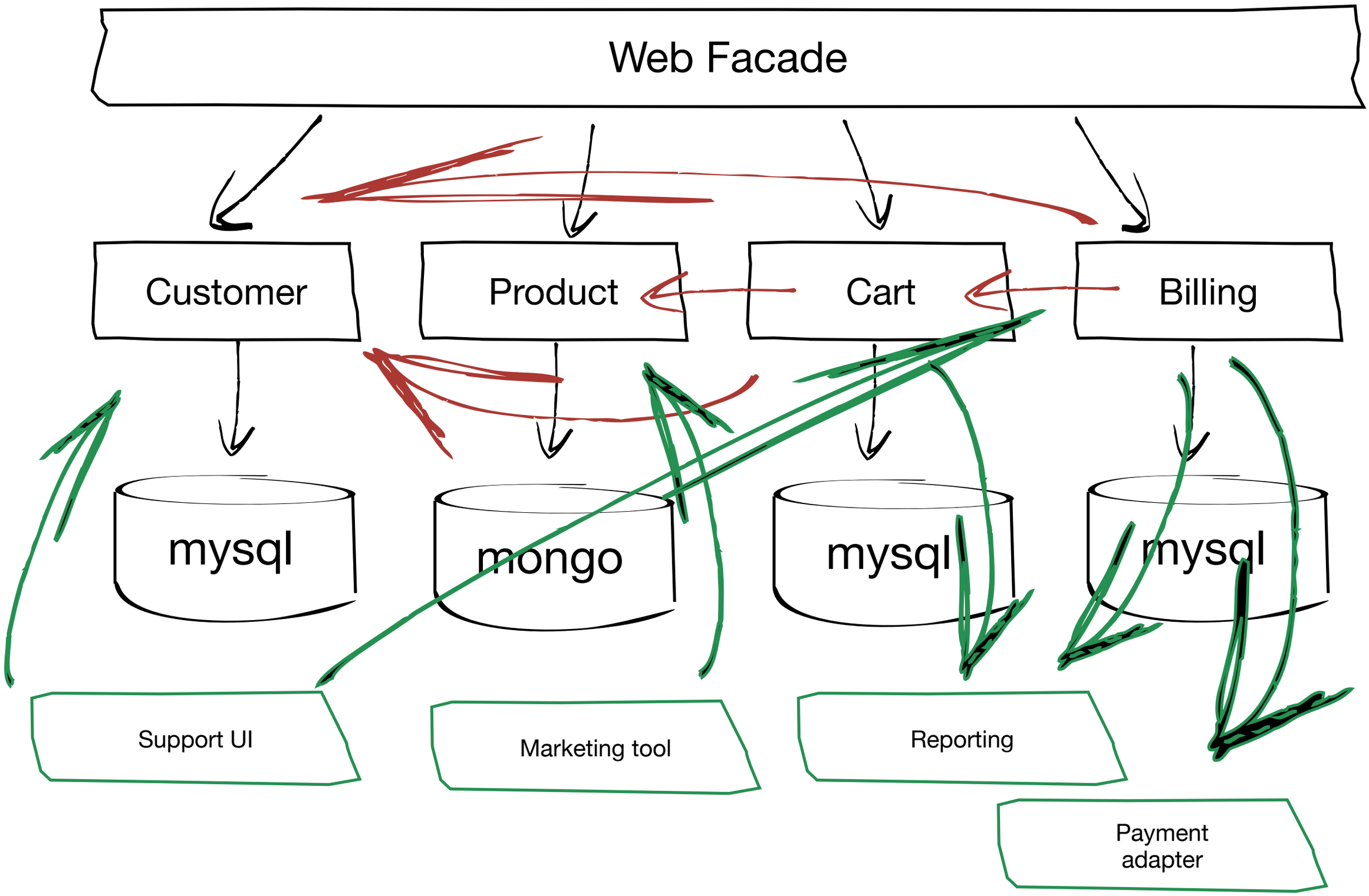
1001



Why is that a challenge?

leanovate
beyond product

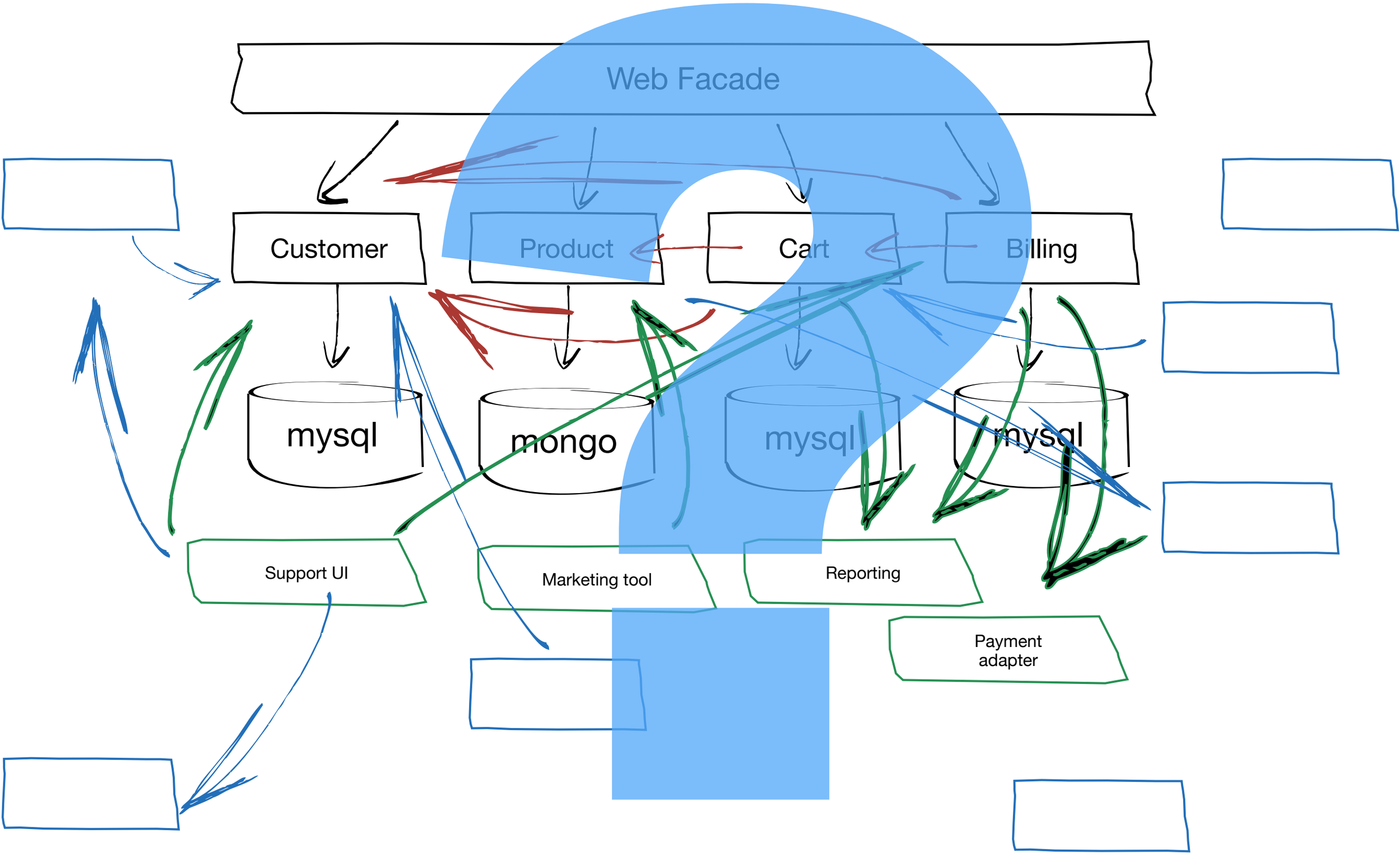
1001



Why is that a challenge?

leanovate
beyond product

1001



Takeaway

Running/Configuring/Debugging

- … will quickly become a non-trivial matter
- We have chosen …
  - **docker** for development system
  - **puppet** for »production«
  - **elasticsearch/logstash/kibana** for distributed logging
  - **zipkin** for request tracing
- … but that is not this focus of this talk

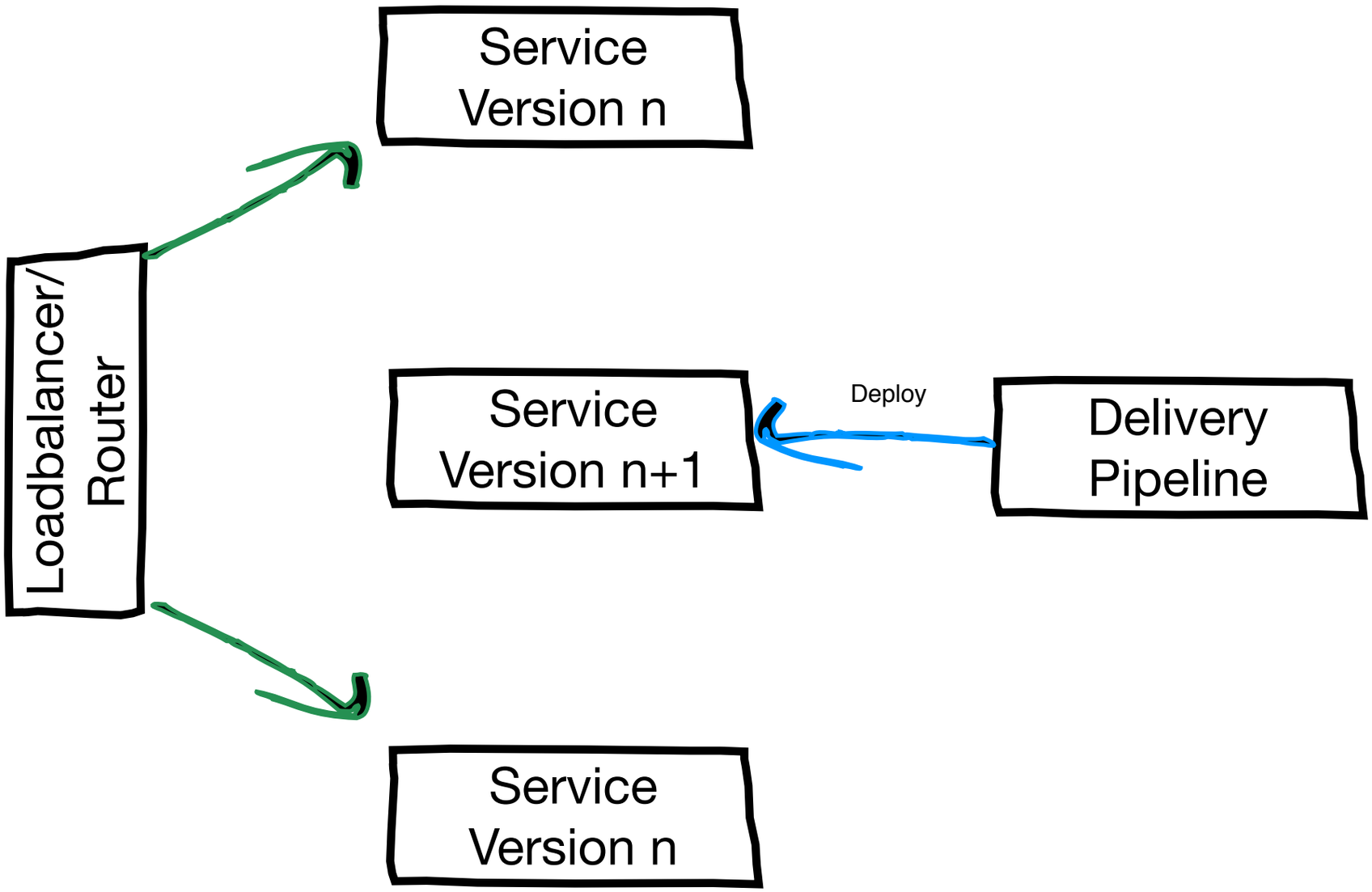
Challenge: Deploying

leanovate
beyond product

1001

Zero-downtime deployment strategies/variants:

Blue/Green, Wave, Canary,…



Challenge: Service discovery

leanovate
beyond product

1001

- How to remove service nodes from the cluster or take them temporary offline?
- How to add new ones or take them online again?

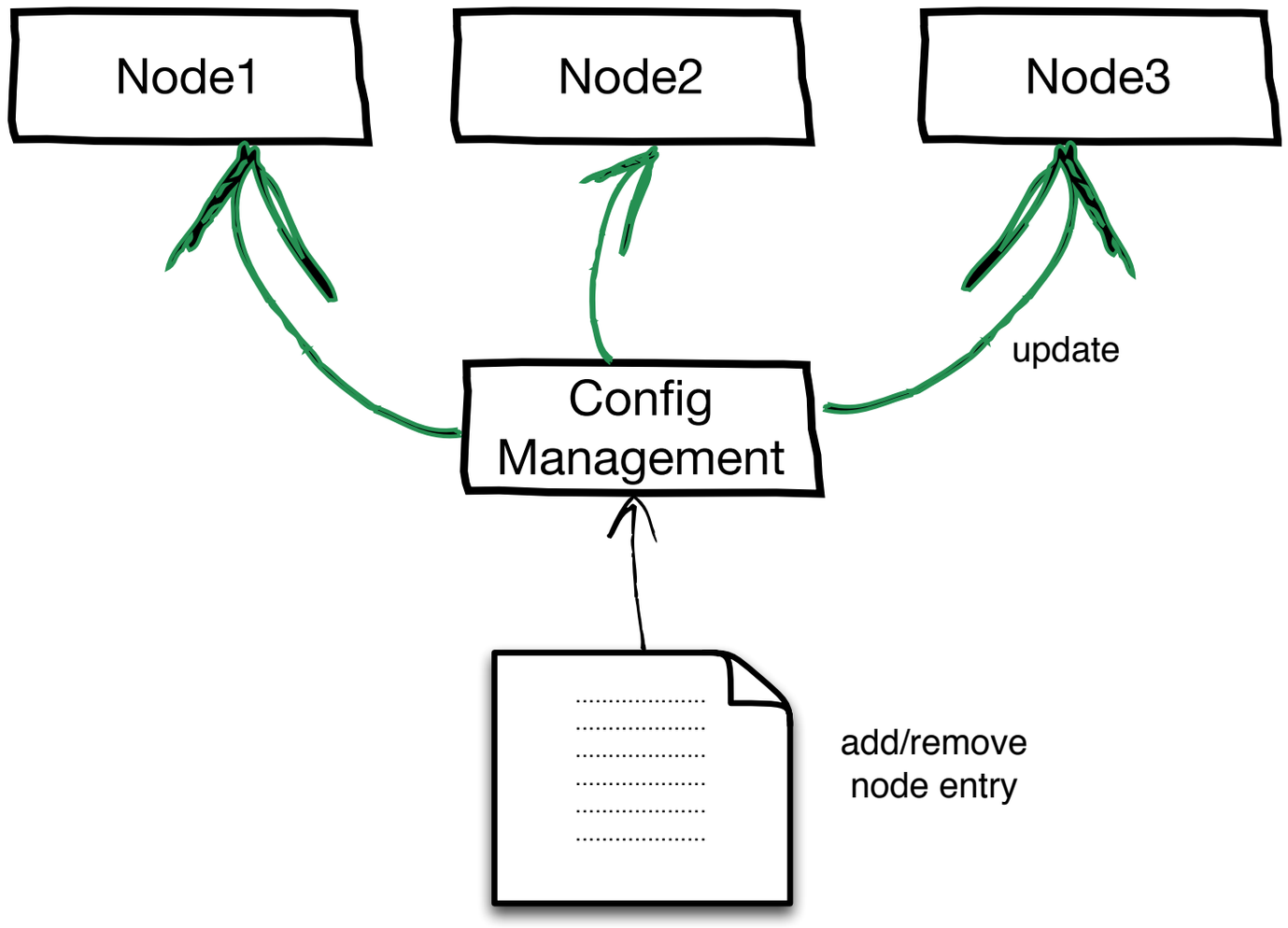
How do services find each other?
=> ServiceDiscovery

Service Discovery by configuration

leanovate
beyond product

1001

You (mis)use your configuration management tool (puppet/chef) to generate service configuration with explicit endpoints



Service Discovery by configuration

leanovate
beyond product

Pros:

Simply works

No extra technology involved (and thereby no extra point of failure)

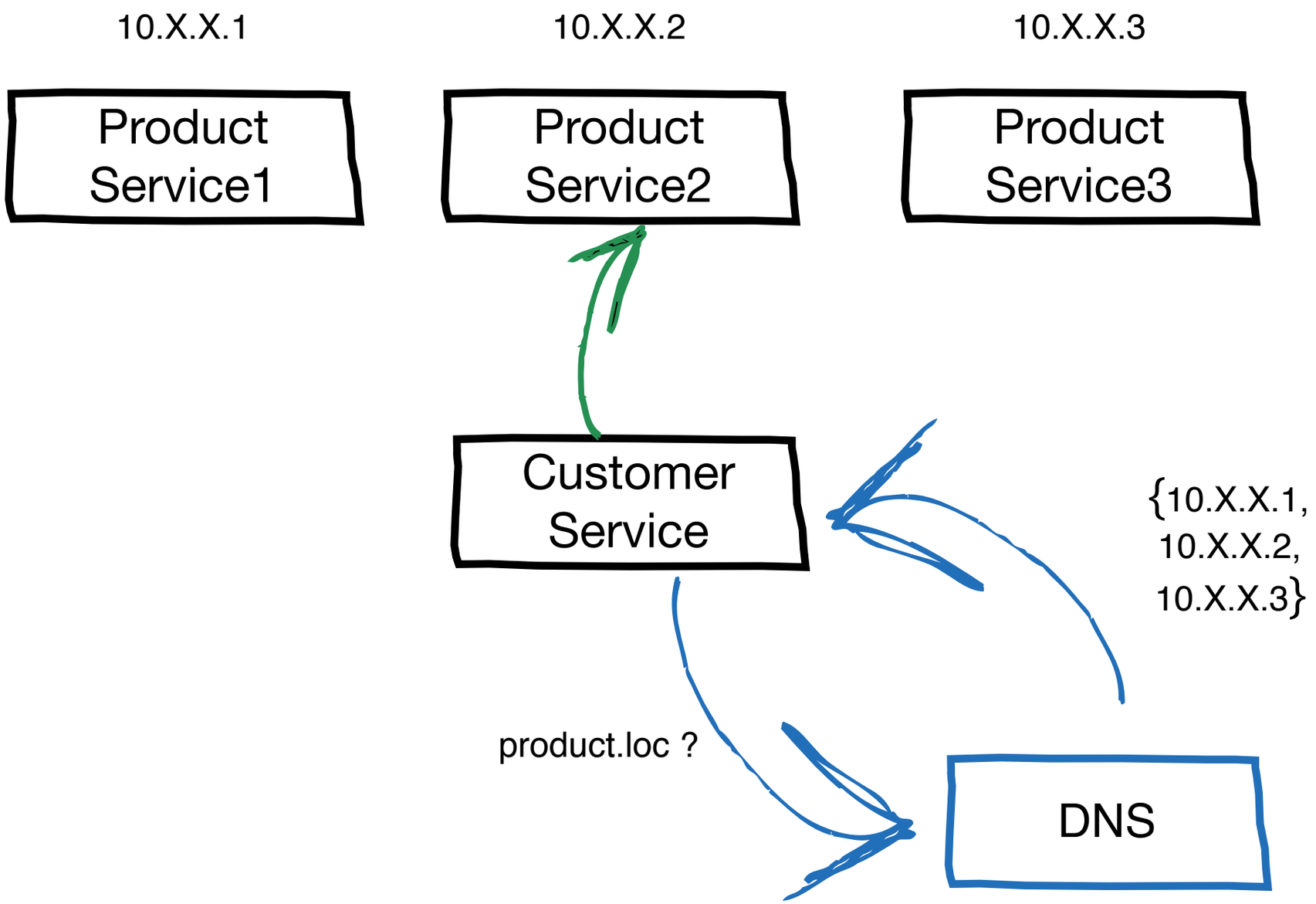
Cons:

To take a service node offline one has to update all of its consumers (or wait for them to be updated)

All consumers have to be able to reload their configuration without restart

Service Discovery by DNS

DNS is actually a service discovery



Pros:

- Old technology that is proven to work on a very very large scale
- Supported by almost everyone

Cons:

- Rather crude (and very inconvenient) interface (especially for updates)
- Resolved service names might be cached on multiple levels
- Focusses purely on service nodes, not on the services itself

Also: DNS might lead to wrong assumptions

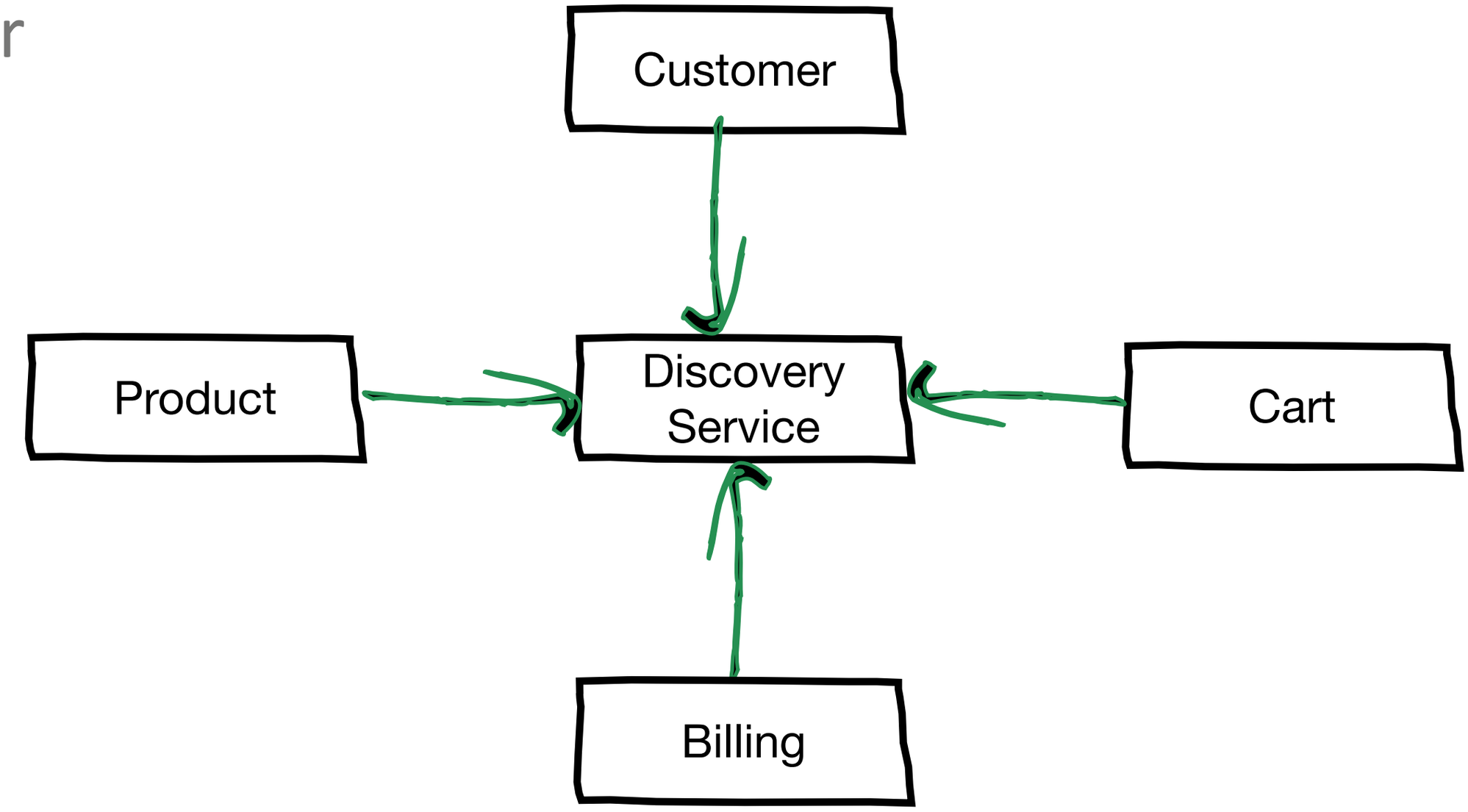
Apache HttpClient 4.3:
org.apache.http.impl.conn.HttpClientConnectionOperator

```
public void connect( ... ) throws IOException {
    ...
    final InetAddress[] addresses = this.dnsResolver.resolve(host.getHostName());
    ...
    for (int i = 0; i < addresses.length; i++) {
        final InetAddress address = addresses[i];
        final boolean last = i == addresses.length - 1;

        Socket sock = sf.createSocket(context);
        ...
        try {
            sock.setSoTimeout(socketConfig.getSoTimeout());
            ...
            conn.bind(sock);
            return;
        } catch (final SocketTimeoutException ex) {
            ...
        } catch (final ConnectException ex) {
            ...
        }
    }
}
```

Service discovery service

Zookeeper
Curator
consul
etcd
doozerd
SkyDNS
Eureka
…



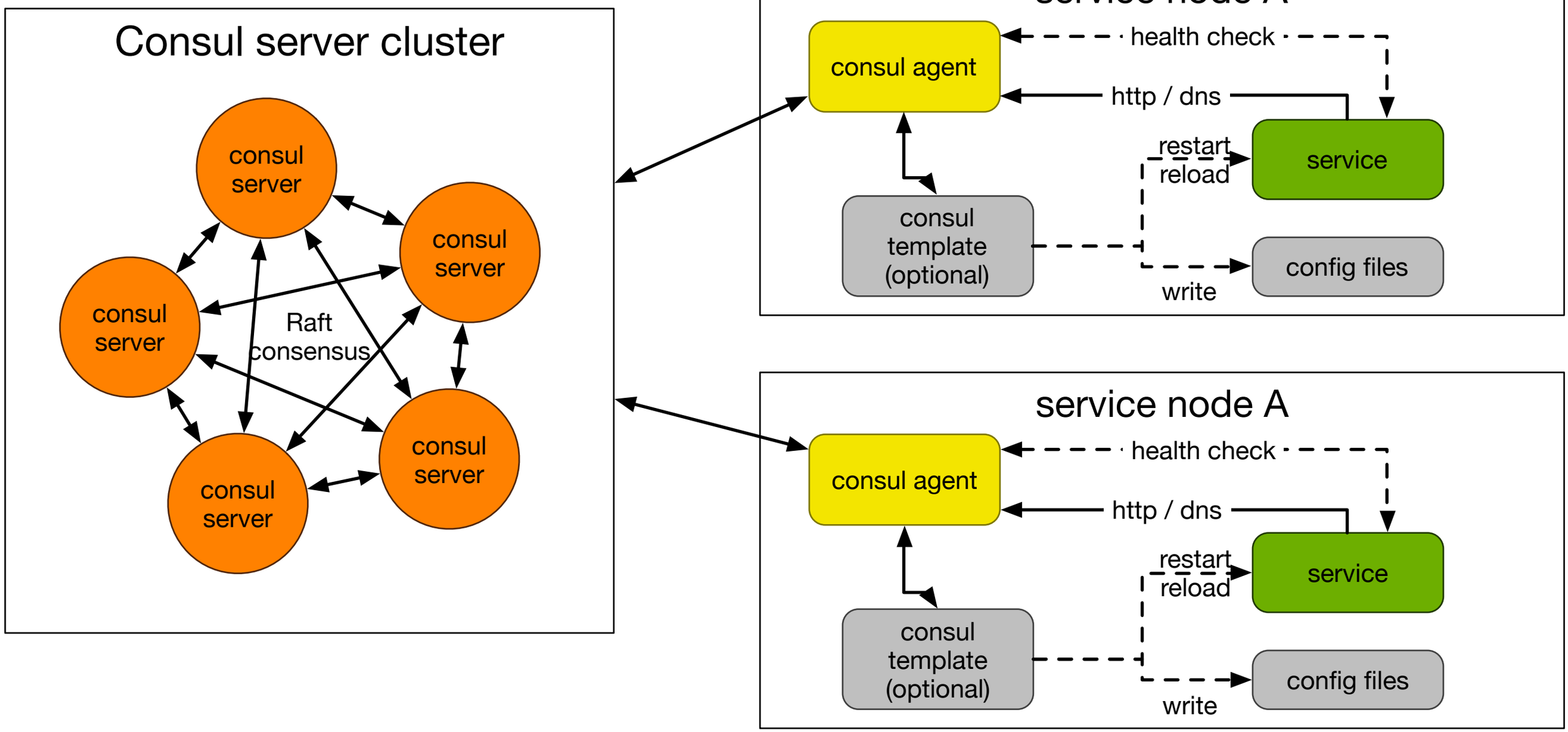
Evaluation criteria

leanovate
beyond product

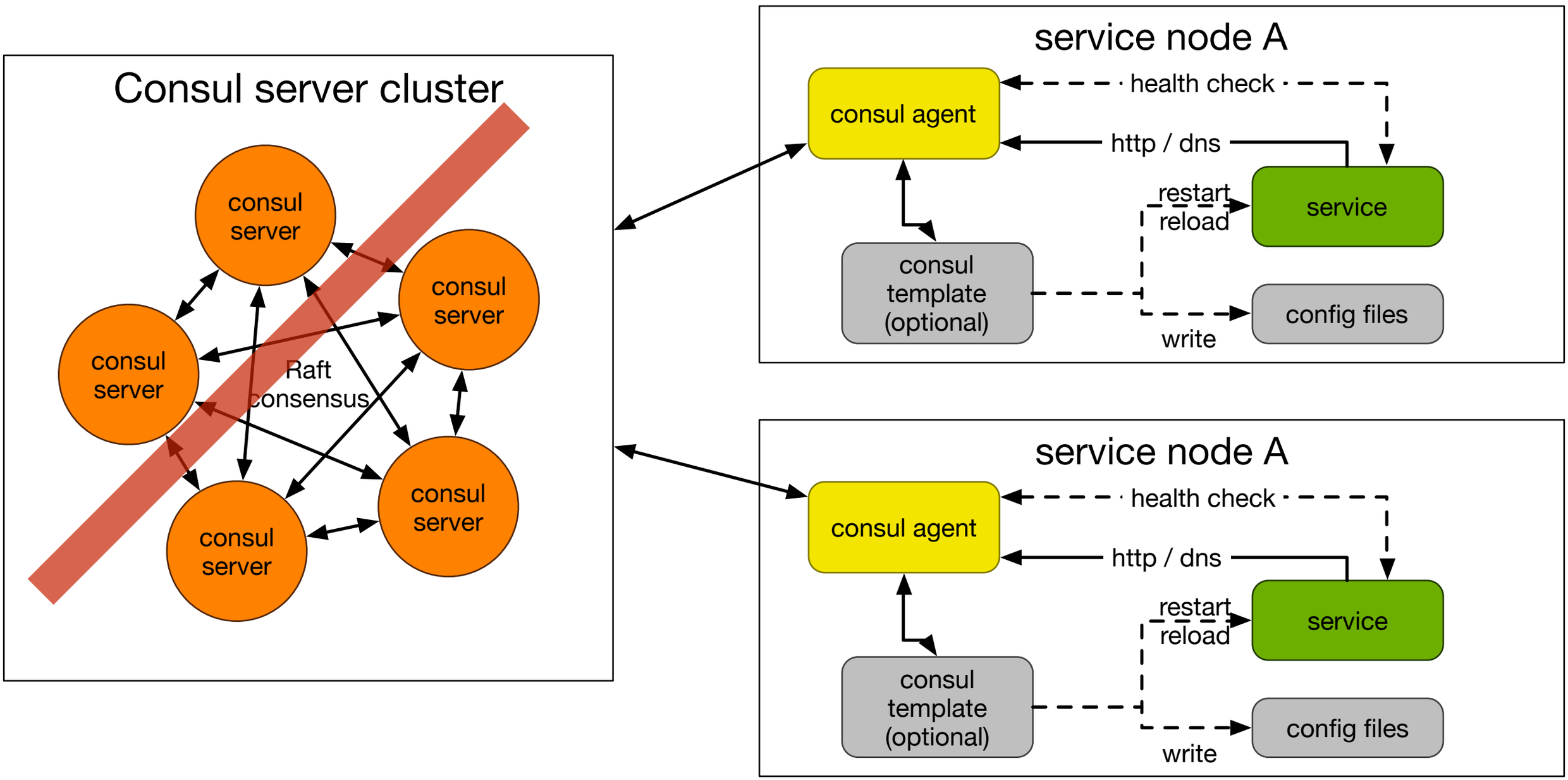
1001

- High availability
- Consistency (consistency over availability?)
- Support for service-level checks
- API (http, DNS?, …)
- Footprint (Memory/CPU)
- Multiple datacenter support
- UI
- Template engine (for config files)

Consul



Consul


Consul server cluster
consul server
consul server
consul server
consul server
consul server
Raft consensus
service node A
consul agent
health check
http / dns
restart
reload
service
consul template (optional)
config files
write
service node A
consul agent
health check
http / dns
restart
reload
service
consul template (optional)
config files
write

Consul

leanovate
beyond product

1001

DEMO

- Show consul UI
- Show web-service status page
- Add cart/remove cart

Challenge: Resilience

leanovate
beyond product

1001

Every system has a retry!

Do not do failover/retry on connection

leanovate
beyond product

1001

GET ok

DELETE ok

PUT ok

POST ???

Duplicates might be ok (e.g. create new shopping cart)

… or not (e.g. register new customer)

might be solved by a request token (e.g. the xsrf token from the web) as long as the service supports this

GET really ok? What about streaming?

Failover should be part of the business

leanovate
beyond product

1001

Usually the failover strategy depends on the concrete use-case

Handling failover on the protocol layer (http-client) might hide error scenarios from the programmer

It can be difficult to distinguish between technical and business error on the protocol layer

As a rule of thumb: You want to retry all technical error, but not the business errors

… though even that is discussable in some cases

How »not« to do service failover

leanovate
beyond product

1001

```
public class ServiceFailover {
    private static final Random RANDOM = new Random(System.currentTimeMillis());

    public static <E, R> R retry(final List<E> endpoints, final Requester<E, R> requester)
                                throws IOException {
        final int size = endpoints.size();

        if (size == 0) throw new RuntimeException("No active endpoints found");

        final int offset = RANDOM.nextInt(size);
        IOException lastException = null;

        for (int idx = 0; idx < size; idx++) {
            final E endpoint = endpoints.get((idx + offset) % size);

            try {
                return requester.performTry(endpoint);
            } catch (IOException e) {
                lastException = e;
            }
        }
        throw lastException;
    }

    @FunctionalInterface
    public interface Requester<E, R> {
        R performTry(E endpoint) throws IOException;
    }
}
```

Resilience

leanovate
beyond product

1001

DEMO

- Kill 2 consul nodes
- Kill one cart node
-

Hystrix

leanovate
beyond product

1001

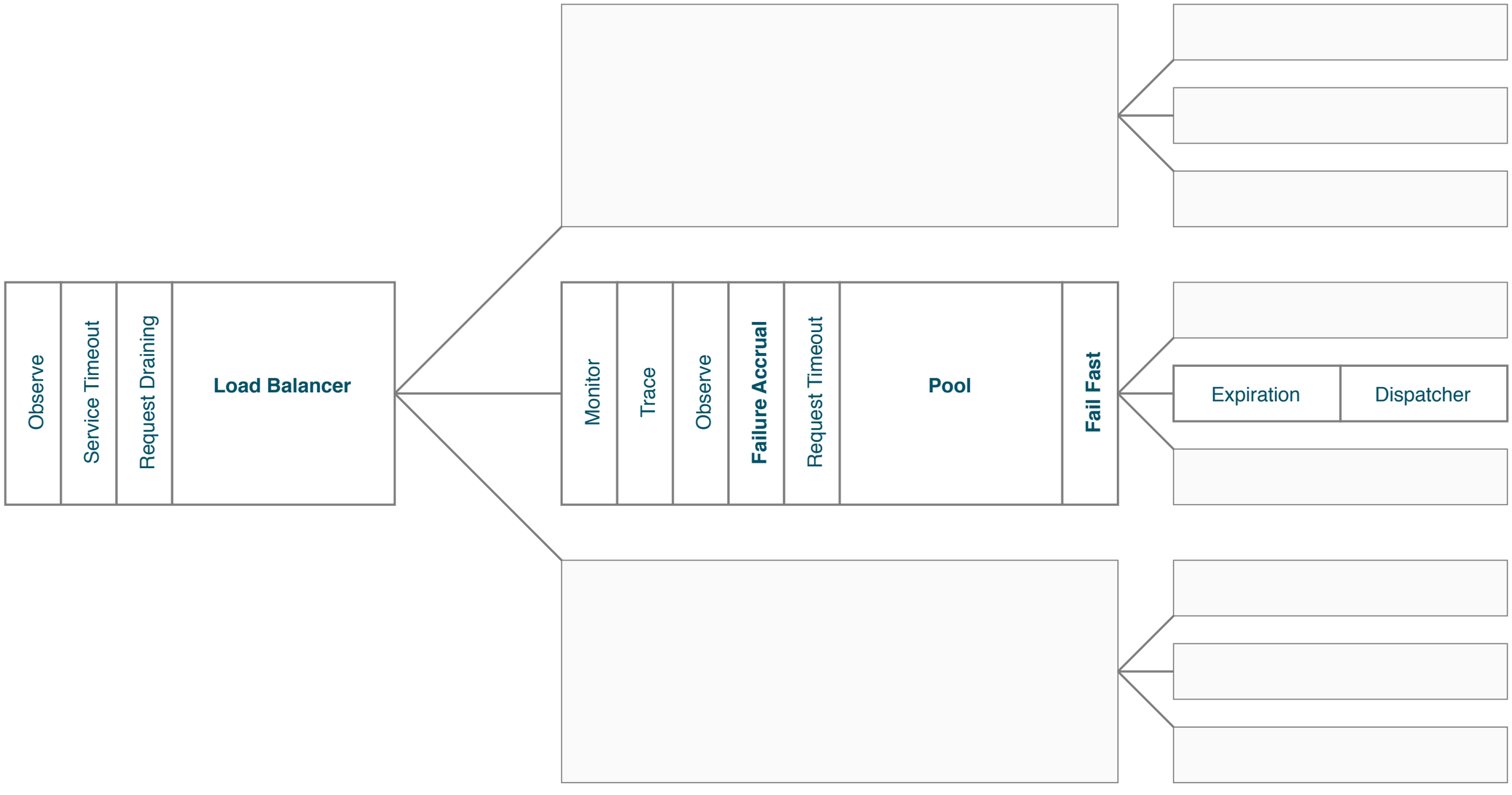
Circuit-Breaking

Fail-Early

Developer’s are »forced« to think in commands with potential fallback result rather than REST-calls

How it actually should look like

according to finagle



What other people do

- Netflix
  - Many libraries and tools that build up on top of each other
  - RxJava/ReactiveX: Reactive extension/Reactive streaming
    - Based on netty: RxNetty
  - Hystrix: Basic command system for circuit-breaking/fail-early
  - Eureka: Service registry
  - Ribbon: REST-Client with failover/service discovery based on Hystrix/Eureka
  - …

What other people do

- Twitter
  - Services based on finagle (scala)
    - … which is itself based on netty
    - … which contains als the basics for for failover/retry/service-discovery/monitoring

Service-Discovery done via zookeeper, but can be adapted/extended to other tools

Several frameworks/connectors build on top: finatra, async-mysql-connector …

Takeaway

Service discovery

- Helps a lot to realize …
  - … any kind of zero-downtime deployment strategy
  - … a self-healing micro-service jungle
- Does not create a fully resilient system by itself, even though it is the basis of it
- Might conflict with your existing configuration system (when creating config files via templates)
- Might be just another central component that fails

Failover/Retry

- The failover strategy usually depends on the business case
- A full failover stack is quite a piece of work
- Emerging frameworks might make life easier or at least provide a reference implementation