DataArt

Docker, microservices, reactive programming and all that Jazz in Java world

Denis Tsyplakov
Solution Architect, DataArt

Why I decided to speak about this topic

DataArt

As a Solution Architect I mostly doing the following types of work:

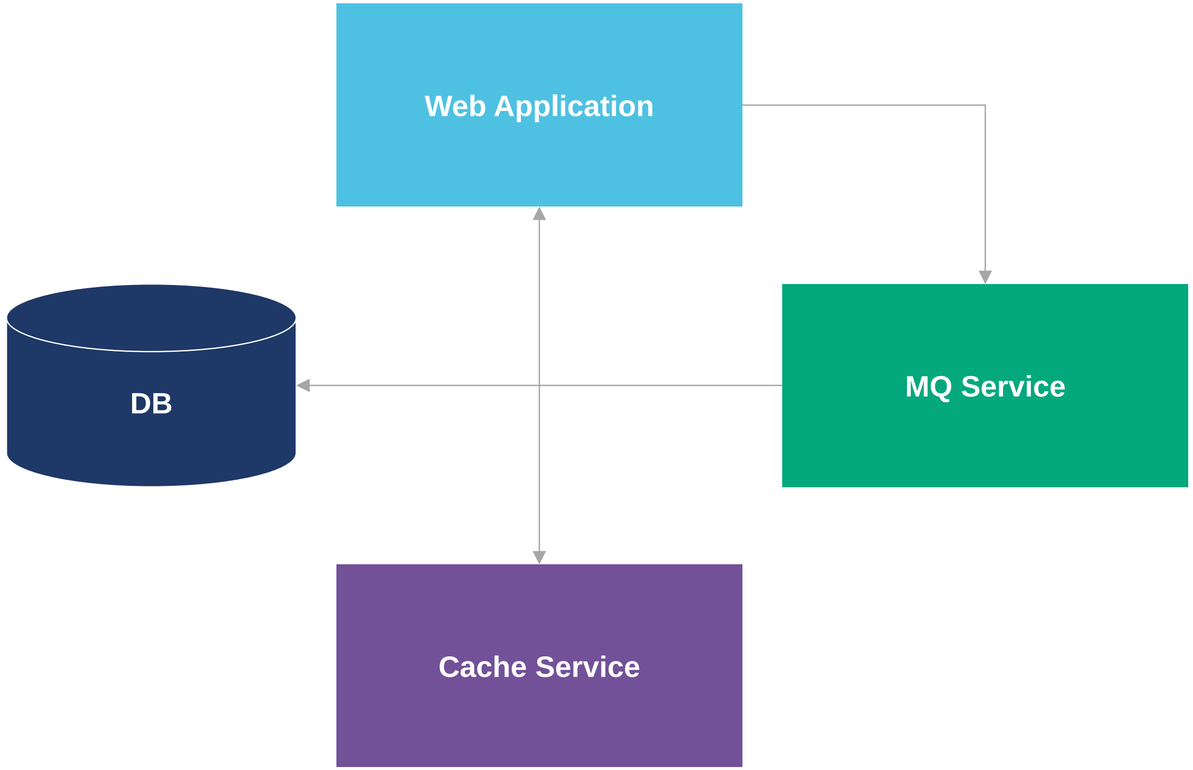
- Help people to fix broken application architecture. Usually it is about:
  - Performance
  - Scalability
  - Extensibility
  - Endless bug fixing
  - Total cost of ownership
- Help people to design application in a way that will allow them to avoid issues from the list above.

There are common anti-patterns, some of them I want to discuss today.

Docker

DataArt

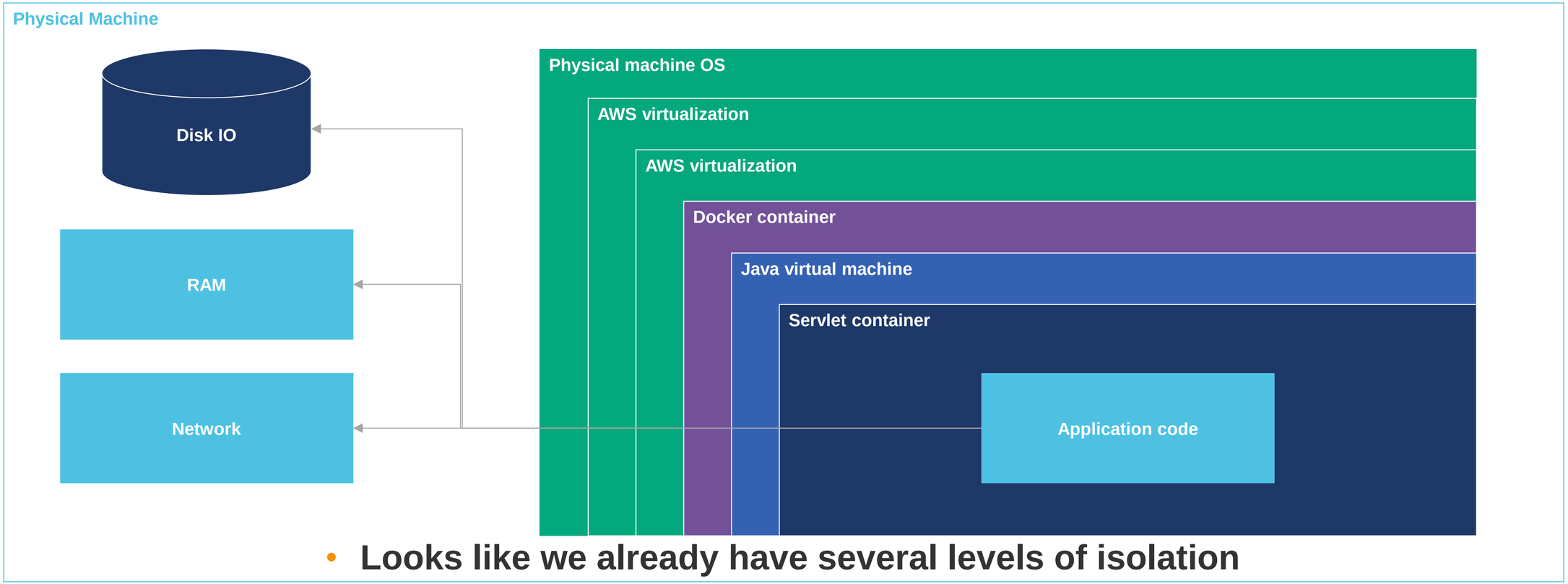
- Does isolation of an application from other applications/runtimes and libraries running / installed on host OS.
- Allows you to package code, libraries, settings, into one container.
- Docker compose allows you to run system that consists of several services in one command.



Looks great!
Should we use Docker with Java?

docker

Dockerized web application deployed on AWS EC2

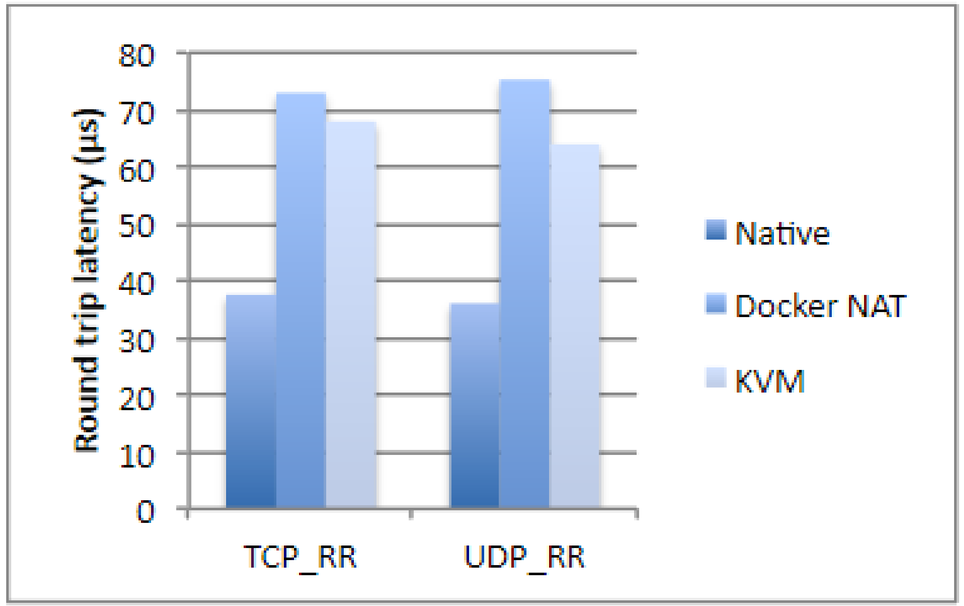


- **Looks like we already have several levels of isolation**

Some numbers from IBM research

DataArt

- Disk IO, CPU, RAM access – show speed almost identical to native.
- Also Docker increases
  - HDD space usage
  - Memory usage
  - Startup time.



- In same cases it is not a big deal, but in some cases it is important.

When do we need Docker with Java stack?

DataArt

- When we need different JRE versions (it is painful, but it happens) or similar case when we need to package JRE together with the code.
- When we have external binary dependencies (JNI, etc).
- When we need to compose our host system from several different services (processes) created with different tech stack (nginx proxies 1 Node.JS application + Go library + Java application + Python ML service).
- When we use Kubernetes (or some other Docker based orchestration system).
- **In simple cases packaging with Spring Boot is enough.**

OK, if our system includes several dependent services we need Kubernetes to manage it

DataArt

In general this is correct,

DataArt

… but in case if you have moderate amount of services / instances, simpler tool can do the job:

- Puppet
- Chef
- CFEngine
- SaltStack

My favorite tool – Ansible:

- Agent-less (uses SSH)
- Minimalistic
- Simple

```
---
- hosts: webservers
  remote_user: root

  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- hosts: databases
  remote_user: root

  tasks:
  - name: ensure postgresql is at the latest version
    yum:
      name: postgresql
      state: latest
  - name: ensure that postgresql is started
    service:
      name: postgresql
      state: started
```

Microservices

DataArt

Advantages of microservices:

Lower resource consumption and better flexibility while scaling

API Versioning

Fault isolation

Dev process isolation for big teams, clean API contracts between teams

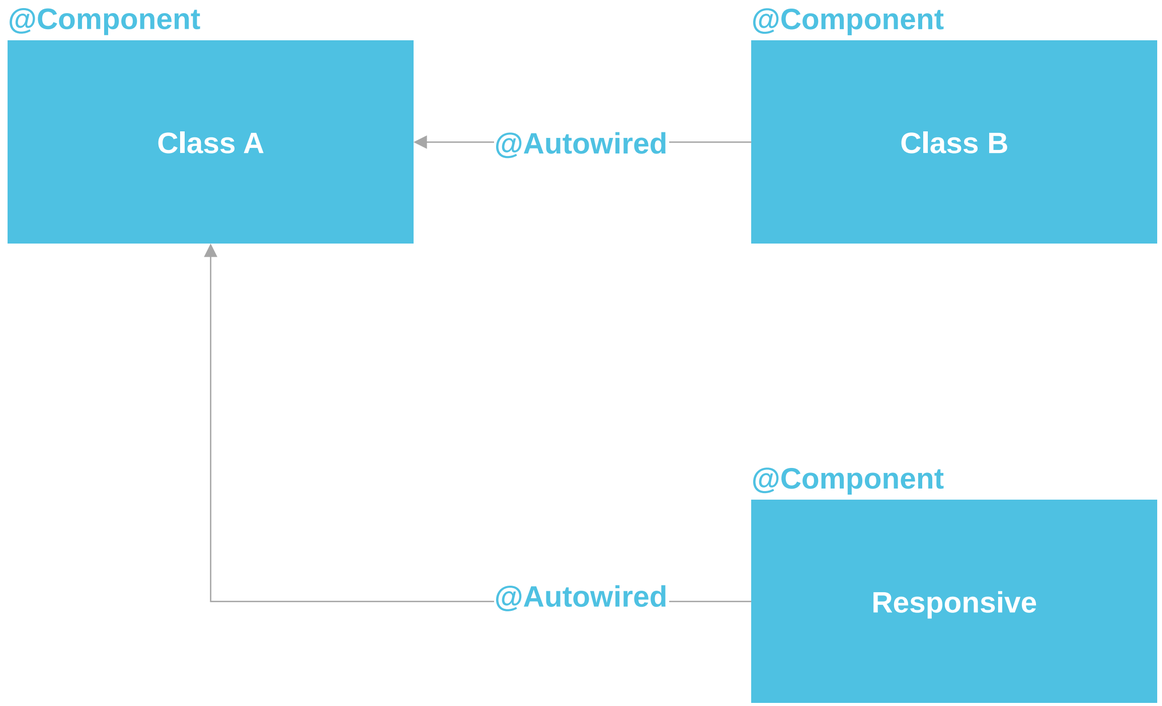
Flexible tech stack

What we have in Java

DataArt

In Java we have:

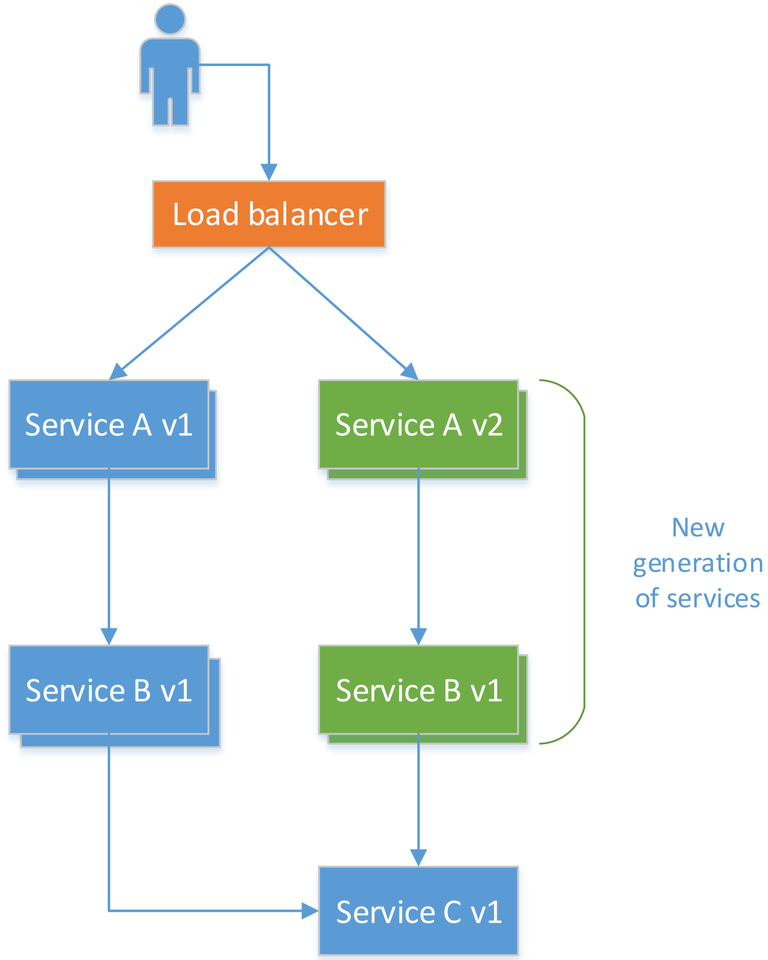
- Class API contracts
- Implementation isolation
- Dependency injection (service discovery)
- Modular projects (team scope isolation, deployment profiles)



Java

When do we need microservices: Flexible versioning

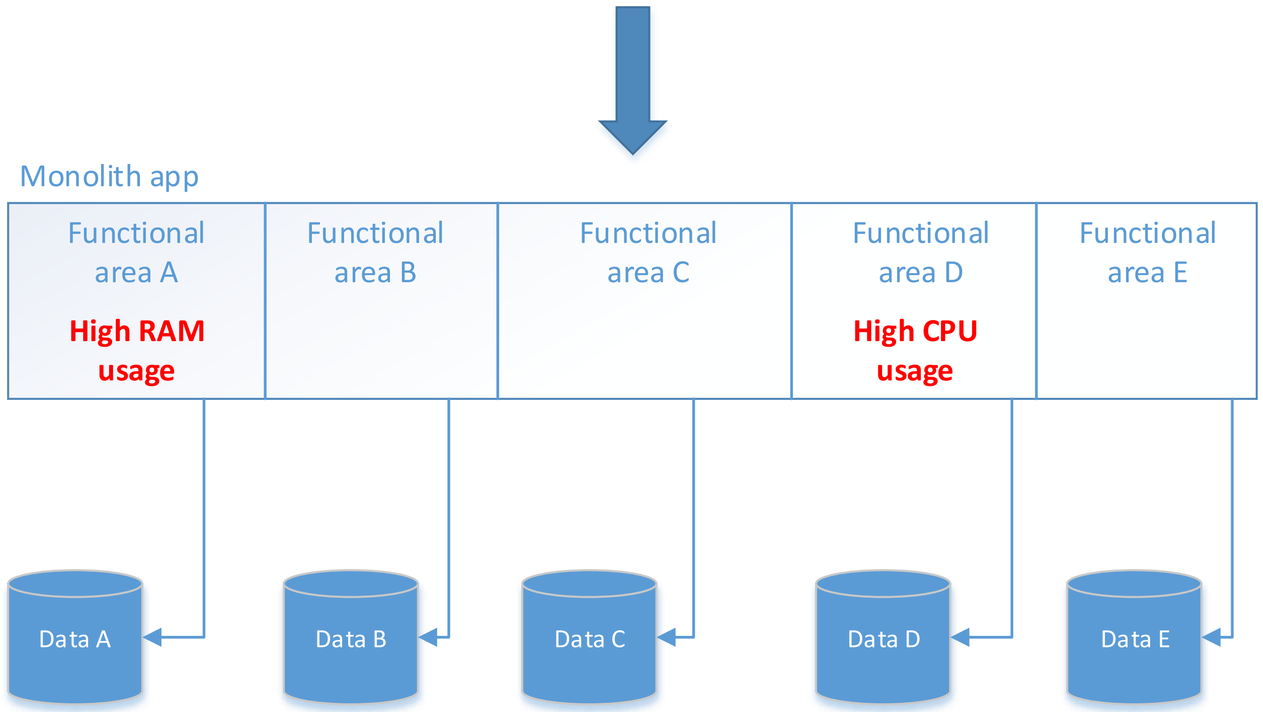
DataArt



When do we need microservices: …

DataArt

- Unbalanced resource usage
- Extreme scaling of some narrow functional area
- Usage of non Java tech for some specific area
- Fault isolation (it is always better to avoid faults, but it is not always possible)
- High management complexity requires area isolation inside service



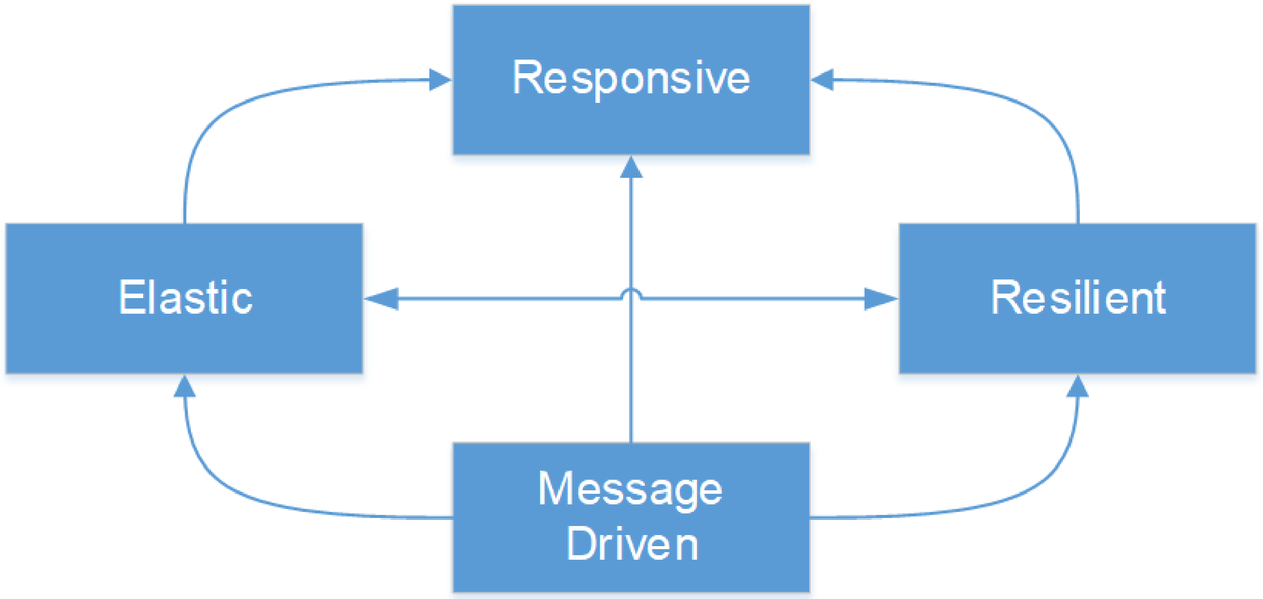
In more generic case monolith application can easily do the job and can be split into several services later.

Reactive approach

DataArt

Reactive Systems is a new, positive trend in modern the IT.

Should every Java application be based on Akka use Play Framework for web or at least use Netty?



Reactive Programming != Reactive System

DataArt

You can do this:

```
manager.getCampaignById(id)
  .flatMap(campaign ->
    manager.getCartsForCampaign(campaign)
      .flatMap(list -> {
        Single<List<Product>> products = manager.getProducts(campaign);
        Single<List<UserCommand>> carts = manager.getCarts(campaign);
        return products.zipWith(carts,
            (p, c) -> new CampaignModel(campaign, p, c));})
     .flatMap(model -> template
        .rxRender(rc, "templates/fruits/campaign.thl.html")
        .map(Buffer::toString))    )
    .subscribe(content -> rc.response().end(content),
     err -> {
      log.error("Unable to render campaign view", err);
      getAllCampaigns(rc); });
```

And you still have !Reactive application

Reactive system -> Reactive architecture

Reactive architecture does not necessarily means reactive programming

When you need to use reactive frameworks?

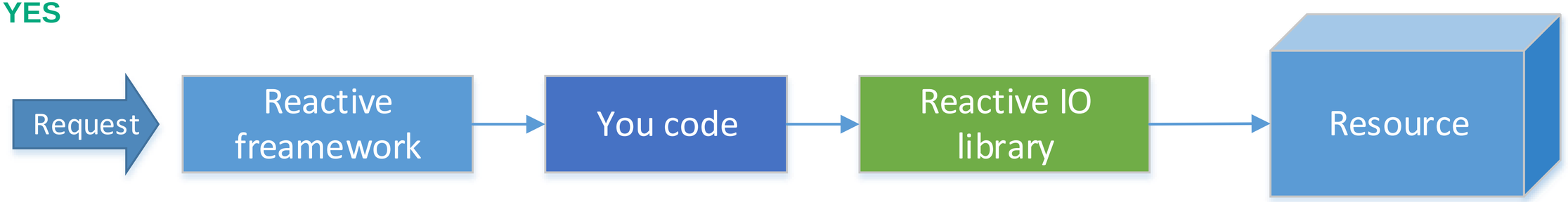
DataArt

- When you need to handle more requests with less threads.
- When do you need this?
- When you can save CPU time on thread context switch.
  Mostly in case when you have many requests and serving each request takes just a few CPU ticks
  AND this requests do not use blocking services.

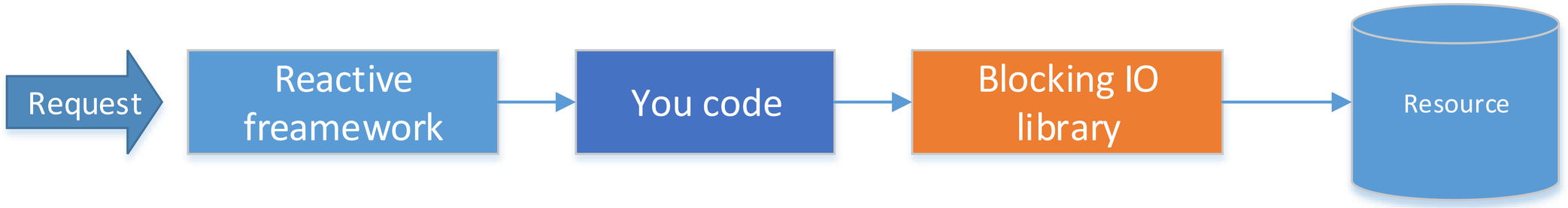
When reactive framework helps

DataArt

YES



NO



OK. Then how to design reactive system?

DataArt

- Start with business requirements that allows to do reactive design.
- Then design your data in a reactive-friendly way.
- Use asynchronous API where this is applicable.
- Shard data.
- Design fine-grained components with clean atomic functionality (component is not necessarily microservice, it can be a class, a function or even a method).
- Use queues and messages where it is applicable.

DataArt

Questions?

DataArt

Thank you!