Microservices Architecture

Memi Lavi
www.memilavi.com

History

- Problems with monolith and SOA led to a new paradigm
- Has to be modular, with simple API
- The term “microservices” first appeared in 2011
- In 2014 Martin Fowler and James Lewis published their “Microservices” article
  - Became the de-facto standard for Microservices definition

The Article


martinFowler.com
Refactoring Agile Architecture About ThoughtWorks
Microservices
a definition of this new architectural term
The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.
25 March 2014
CONTENTS
Characteristics of a Microservice Architecture
Componentization via Services
Organized around Business Capabilities
Products not Projects
Smart endpoints and dumb pipes
Decentralized Governance
Infrastructure Automation
James Lewis
James Lewis is a Principal Consultant at ThoughtWorks and member of the
https://martinfowler.com/articles/microservices.html

The Article


martinFowler.com
Refactoring Agile Architecture About ThoughtWorks
Microservices
a definition of this new architectural term
The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.
25 March 2014
CONTENTS
Characteristics of a Microservice Architecture
Componentization via Services
Organized around Business Capabilities
Products not Projects
Smart endpoints and dumb pipes
Decentralized Governance
Infrastructure Automation
James Lewis
James Lewis is a Principal Consultant at ThoughtWorks and member of the
https://martinfowler.com/articles/microservices.html

Characteristics of Microservices

- Componentization via Services
- Organized Around Business Capabilities
- Products not Projects
- Smart Endpoints and Dumb Pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for Failure
- Evolutionary Design

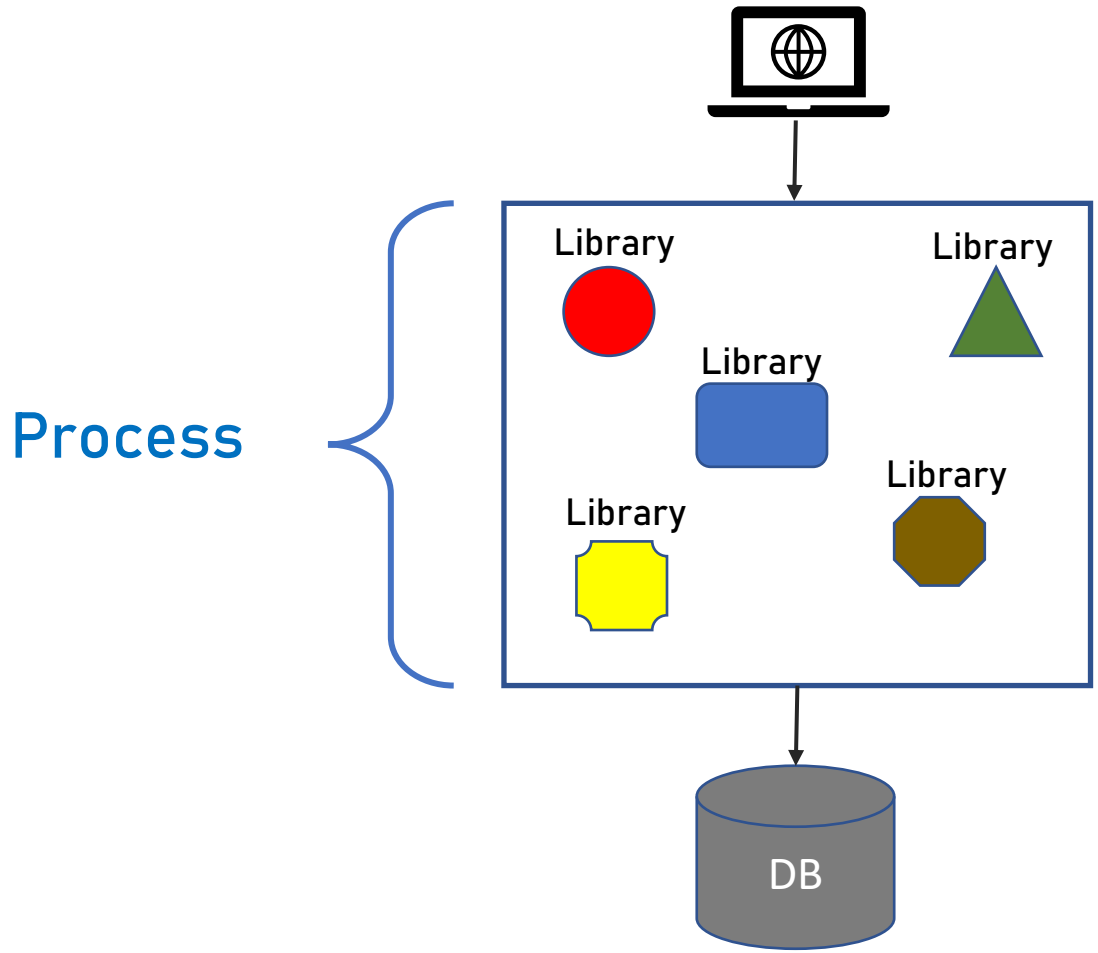
Componentization via Services

- Modular design is always a good idea
- Components are the parts that together compose the software
- Modularity can be achieved using:
  - Libraries – called directly within the process
  - Services – called by out-of-process mechanism (Web API, RPC)

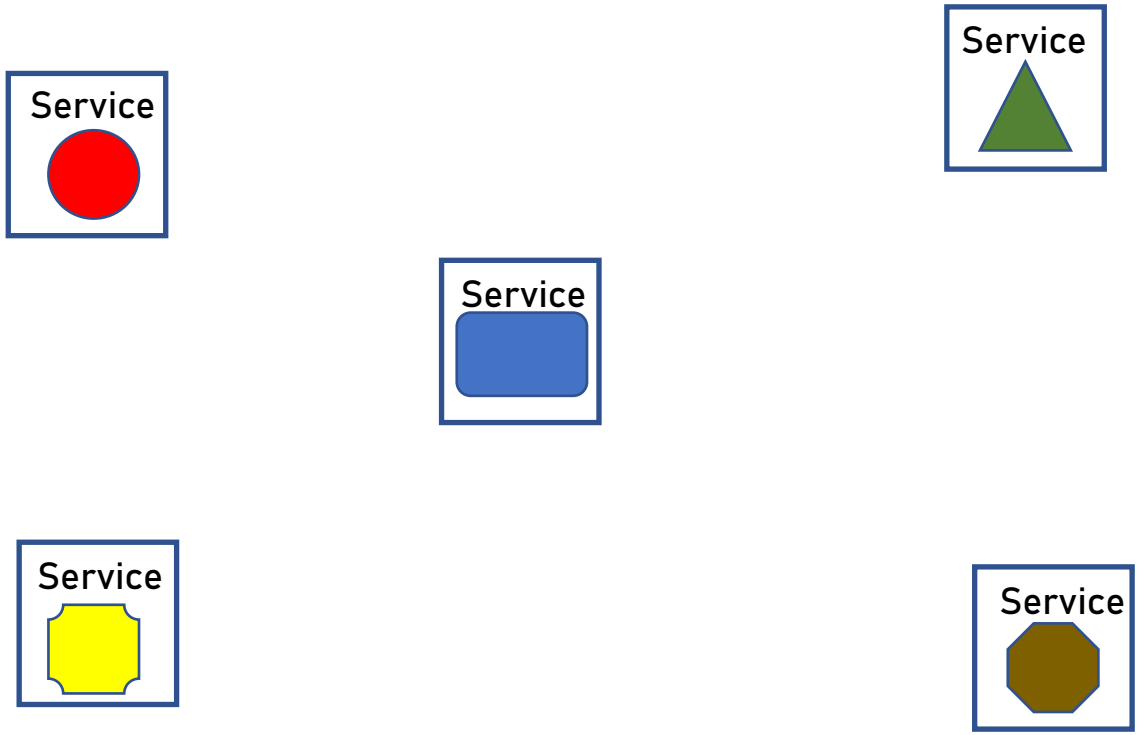
Componentization via Services

- In Microservices we prefer using Services for the componentization
- Libraries can be used inside the service

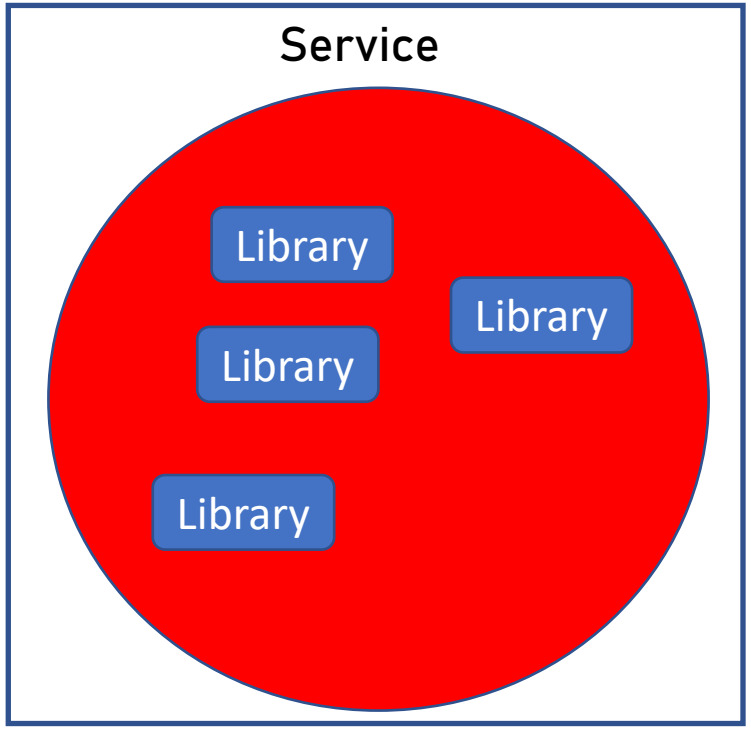
Componentization via Services



Componentization via Services



Componentization via Services



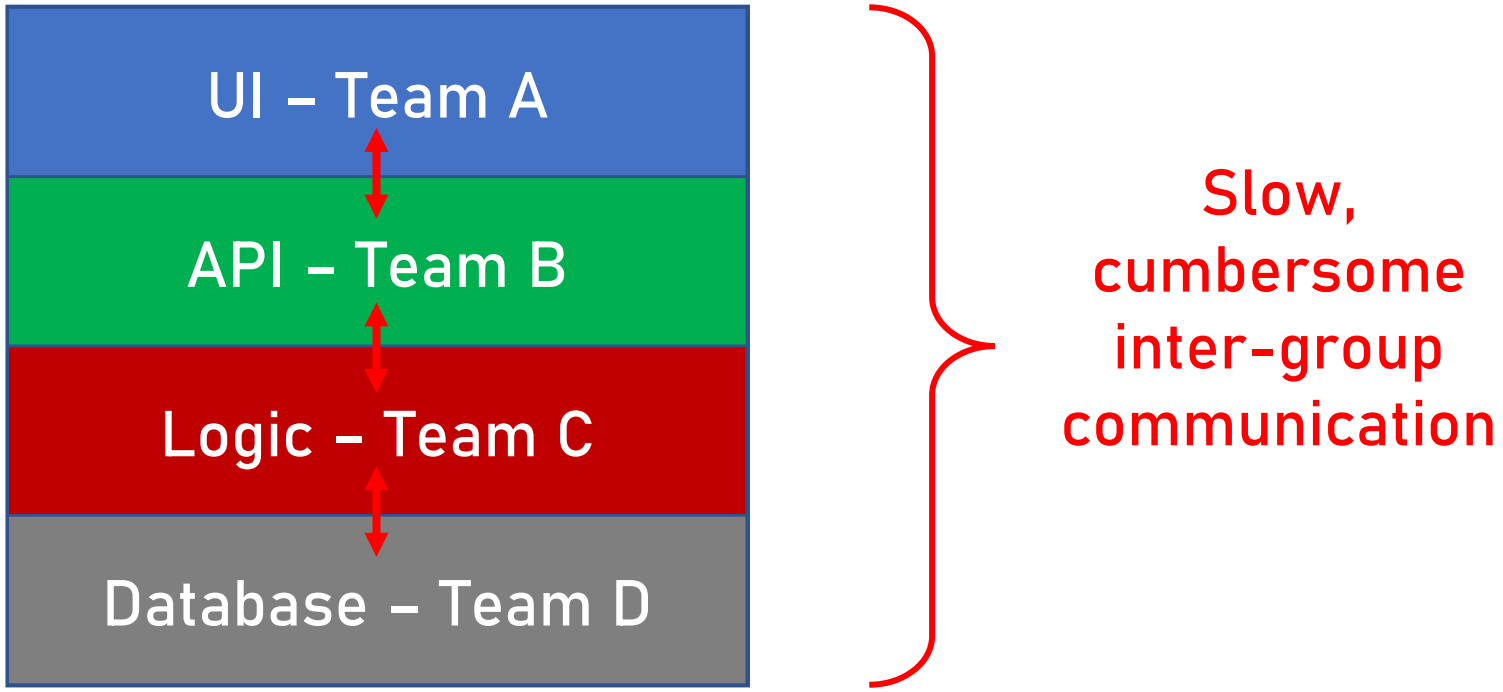
Componentization via Services

- Motivation:
  - Independent deployment
  - Well defined interface

Organized Around Business Capabilities

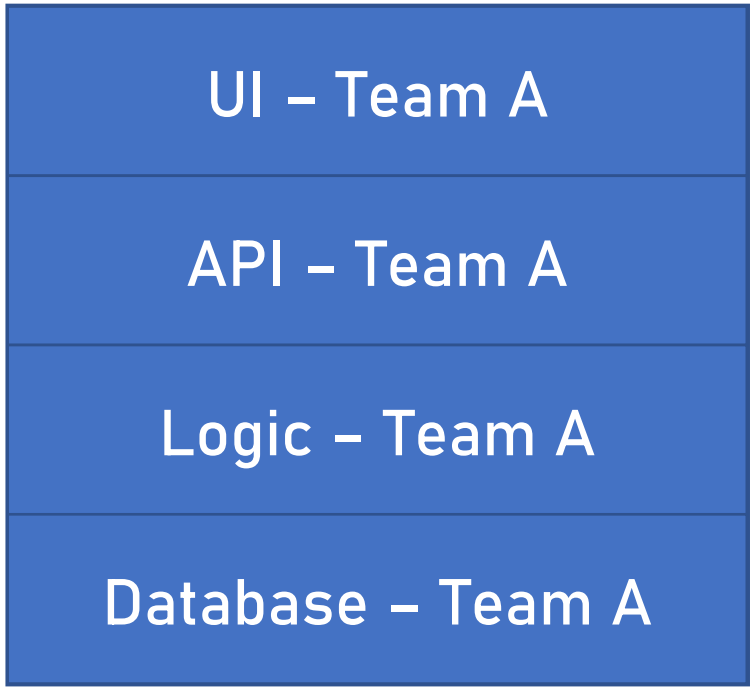
- Traditional projects have teams with horizontal responsibilities –

  UI, API, Logic, DB etc



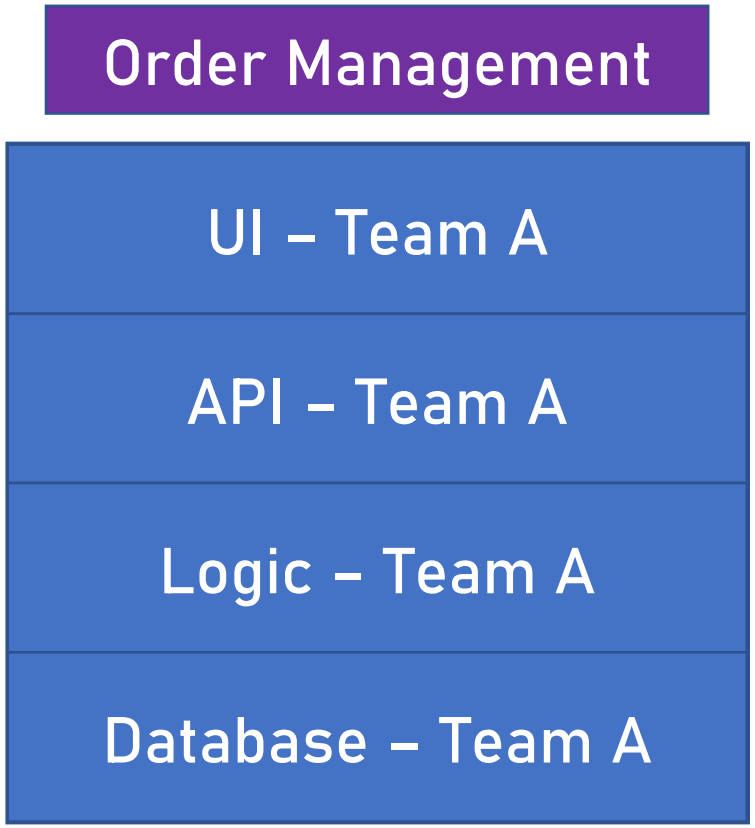
Organized Around Business Capabilities

- With Microservices, every service is handled by a single team, responsible for all aspects.



Organized Around Business Capabilities

- With Microservices, every service handles a well-defined business capability.



Organized Around Business Capabilities

- Motivation:
  - Quick development
  - Well-defined boundaries

Products not Projects

- With traditional projects, the goal is to deliver a working code
- No lasting relationship with the customer
- Often no acquaintance with the customer
- After delivering – the team moves on to the next project

Products not Projects

- With Microservices – the goal is to deliver a working product
- A product needs ongoing support and requires close relationship with the customer
- The team is responsible for the Microservice after the delivery too

> *"You build it, you run it"*
>
> Werner Vogels, AWS CTO

Products not Projects

- Motivation:
  - Increase customer's satisfaction
  - Change developers' mindset

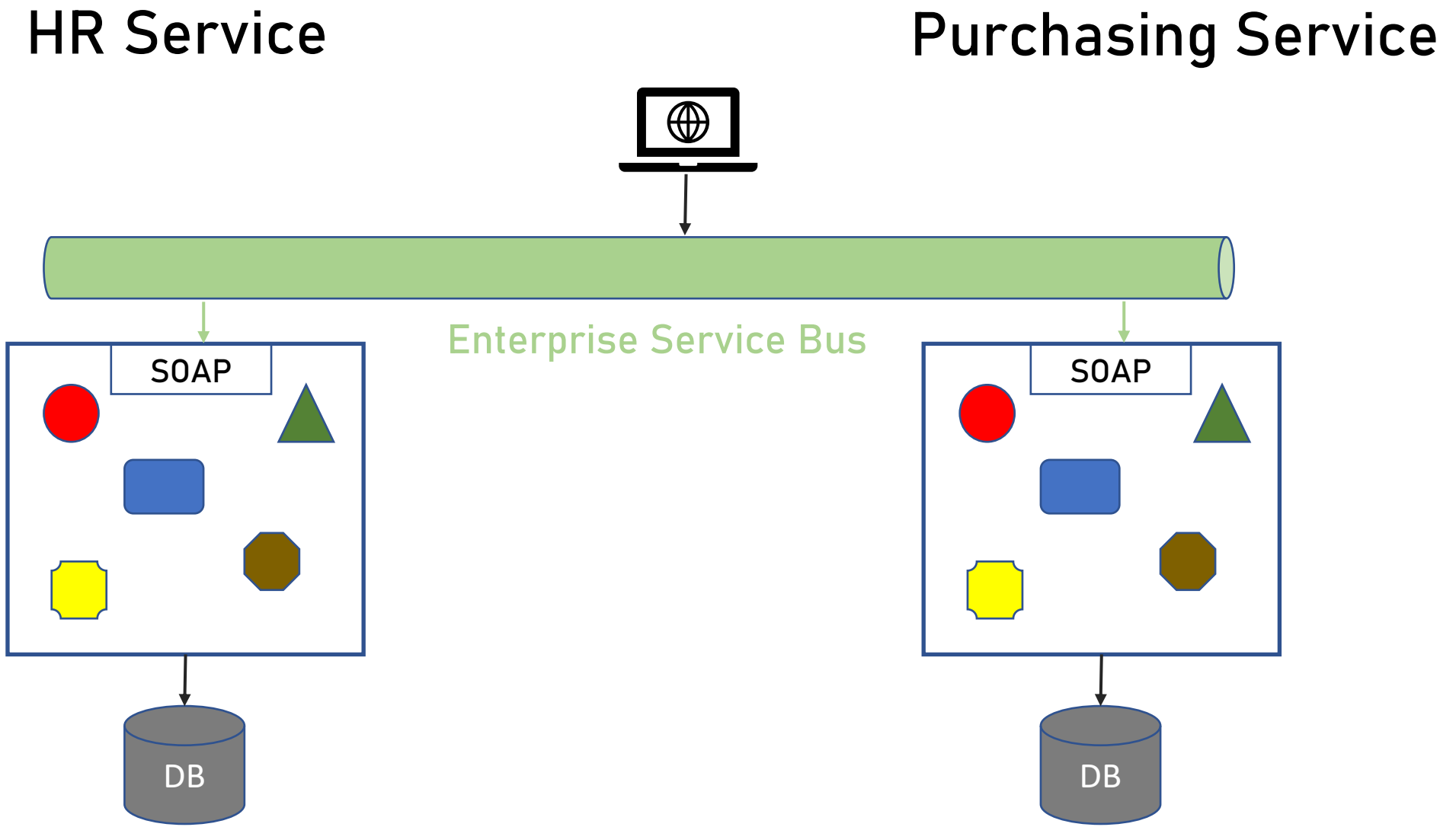
Smart Endpoints and Dumb Pipes

- Traditional SOA projects used two complicated mechanisms:
  - ESB
  - WS-* protocol
- Made inter-service communication complicated and difficult to maintain

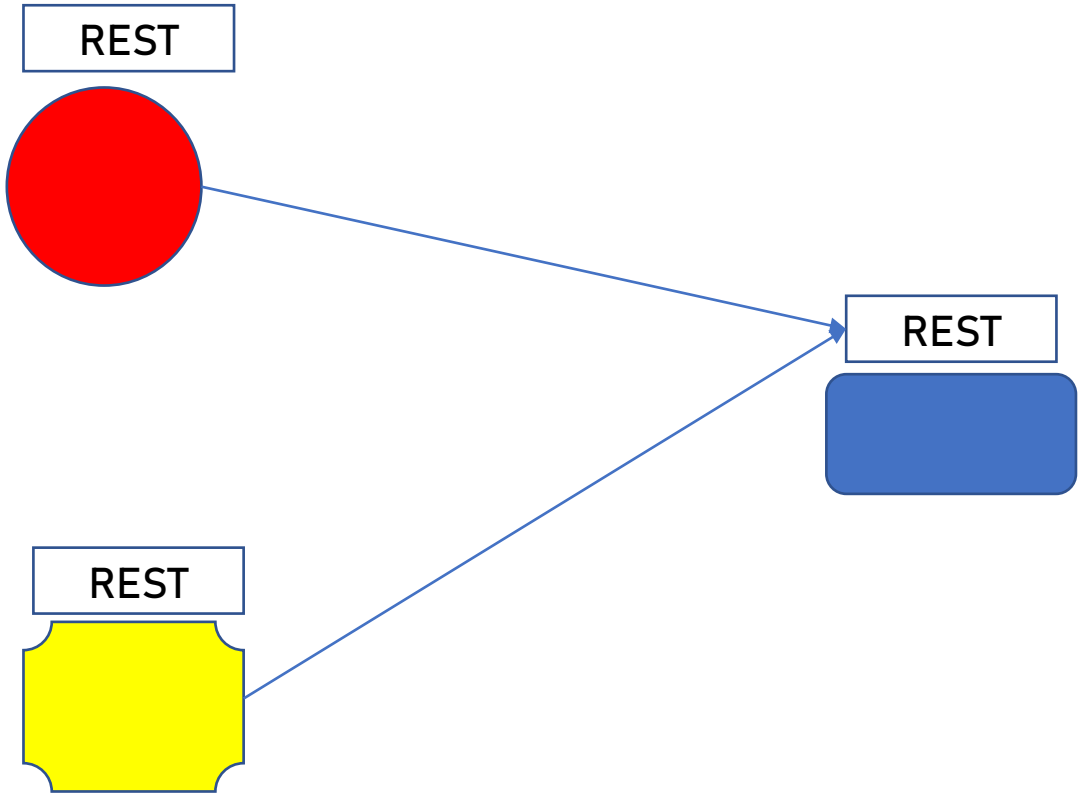
Smart Endpoints and Dumb Pipes

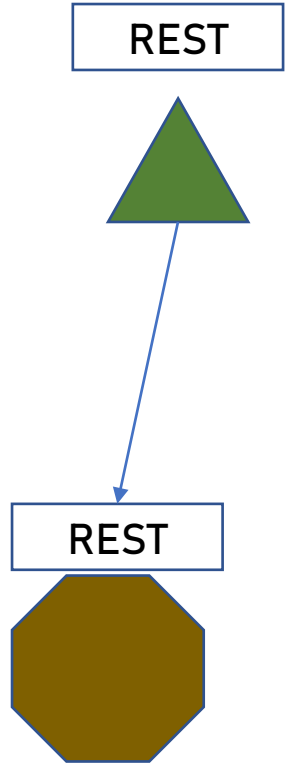
- Microservices systems use “dumb pipes” – simple protocols
- Strive to use what the web already offers
- Usually – REST API, the simplest API in existence

Smart Endpoints and Dumb Pipes



Smart Endpoints and Dumb Pipes





Smart Endpoints and Dumb Pipes

- Important notes:
  - Direct connections between services is not a good idea
  - Better use discovery service or a gateway
  - In recent years more protocols were introduced (GraphQL, gRPC), some of them quite complex

Smart Endpoints and Dumb Pipes

- Motivation:
  - Accelerate development
  - Make the app easier to maintain

Decentralized Governance

- In traditional projects there is a standard for almost anything:
  - Which dev platform to use
  - Which database to use
  - How logs are created
  - And more…

Decentralized Governance

- With Microservices each team makes its own decisions:
  - Which dev platform to use
  - Which database to use
  - How logs are created
  - And more…

Decentralized Governance

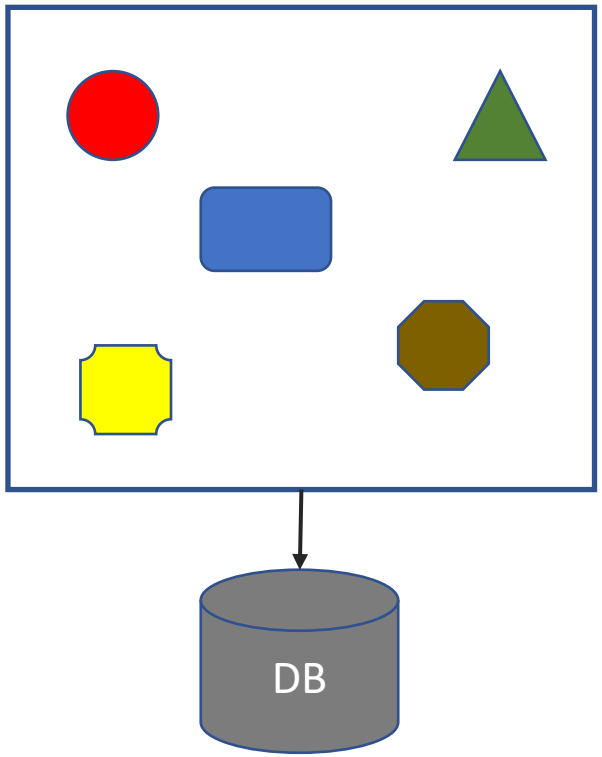
- Each team is fully responsible for its service
  - *“You build it, you run it”*
- …and so will make the optimal decisions
- Enabled by the loosely coupled nature of Microservices
- Multi dev platform is called *Polyglot*

Decentralized Governance

- Motivation:
  - Enables making the optimal technological decisions for the specific service

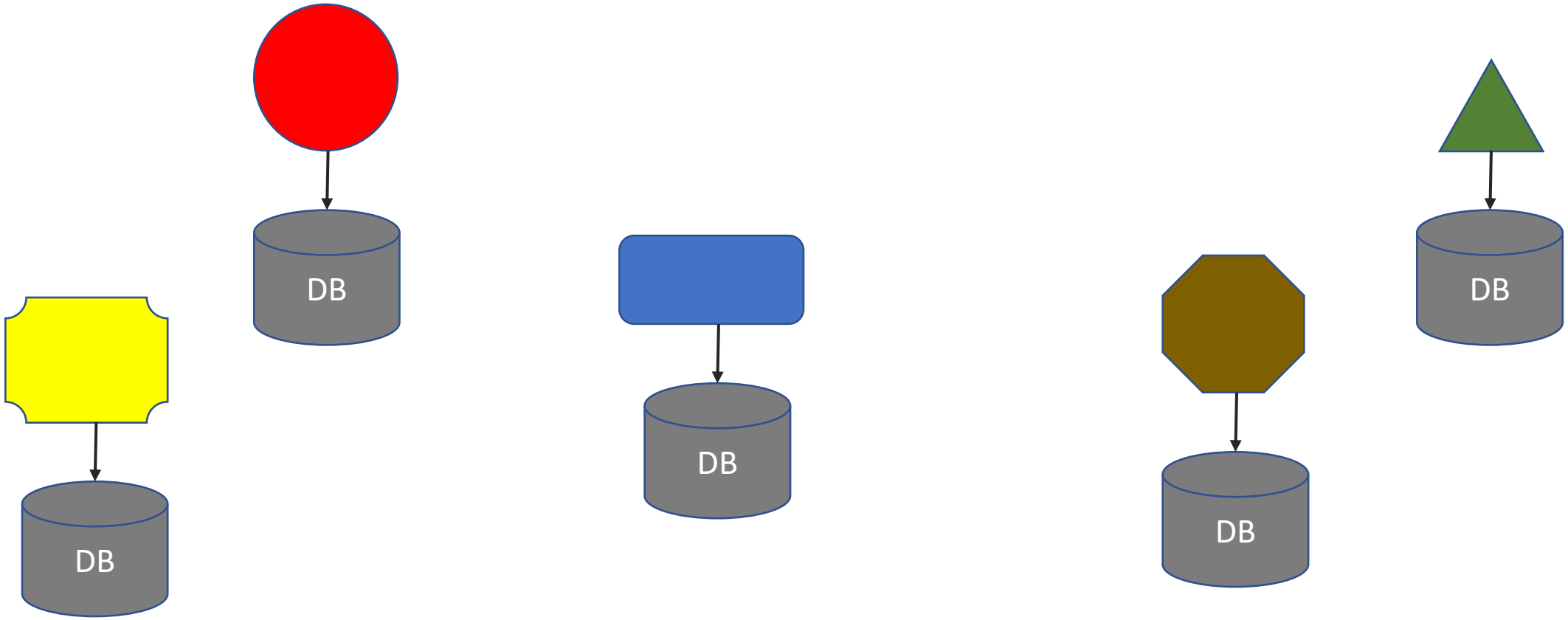
Decentralized Data Management

- Traditional systems have a single database
- Stores all the system's data from all the components



Decentralized Data Management

- With Microservices each service has its own database



Decentralized Data Management

- Important notes:
  - This is the most controversial attribute of Microservices
  - Not always possible
  - Raises problems such as distributed transactions, data duplication and more
  - Don't insist on it

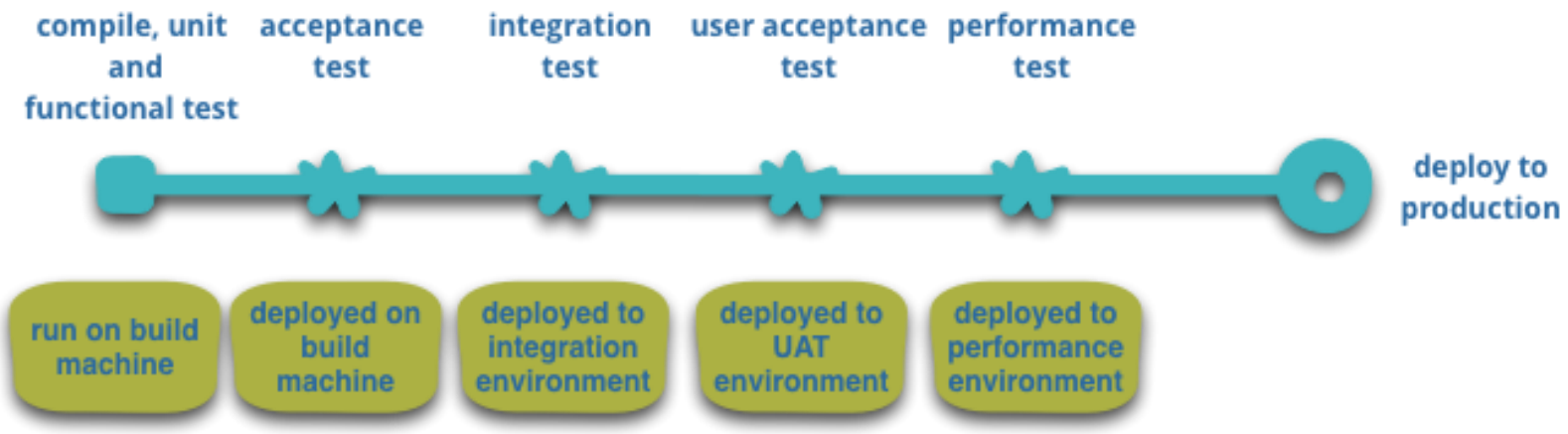
Decentralized Data Management

- Motivation:
  - Right tool for the right task – having the right database is important
  - Encourages isolation

Infrastructure Automation

- The SOA paradigm suffered from lack of tooling
- Tooling greatly helps in deployment using:
  - Automated Testing
  - Automated Deployment

Infrastructure Automation



Source: https://martinfowler.com/articles/microservices.html

Infrastructure Automation

- For Microservices automation is essential
- Short deployment cycles are a must
- Can't be done manually
- There are a lot of automation tools:

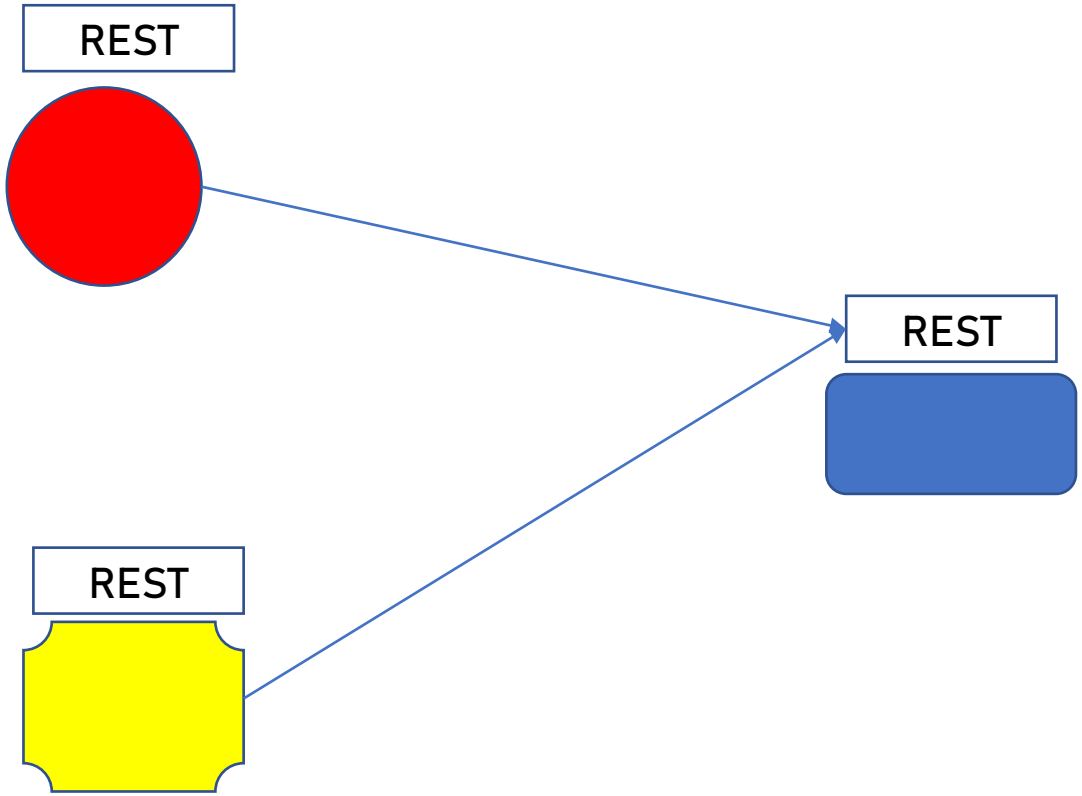
Infrastructure Automation

- Motivation:
  - Short deployment cycles

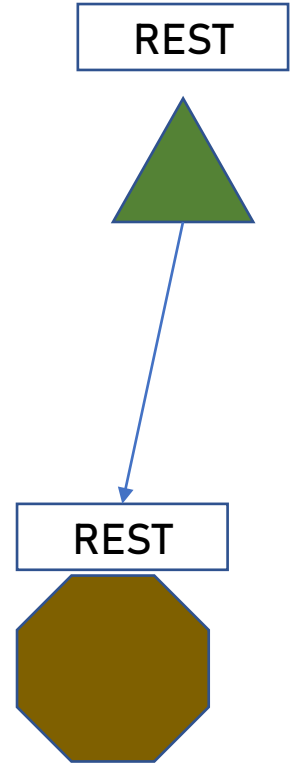
Design for Failure

- With Microservices there are a lot of processes and a lot of network traffic
- A lot can go wrong
- The code must assume failure can happen and handle it gracefully
- Extensive logging and monitoring should be in place

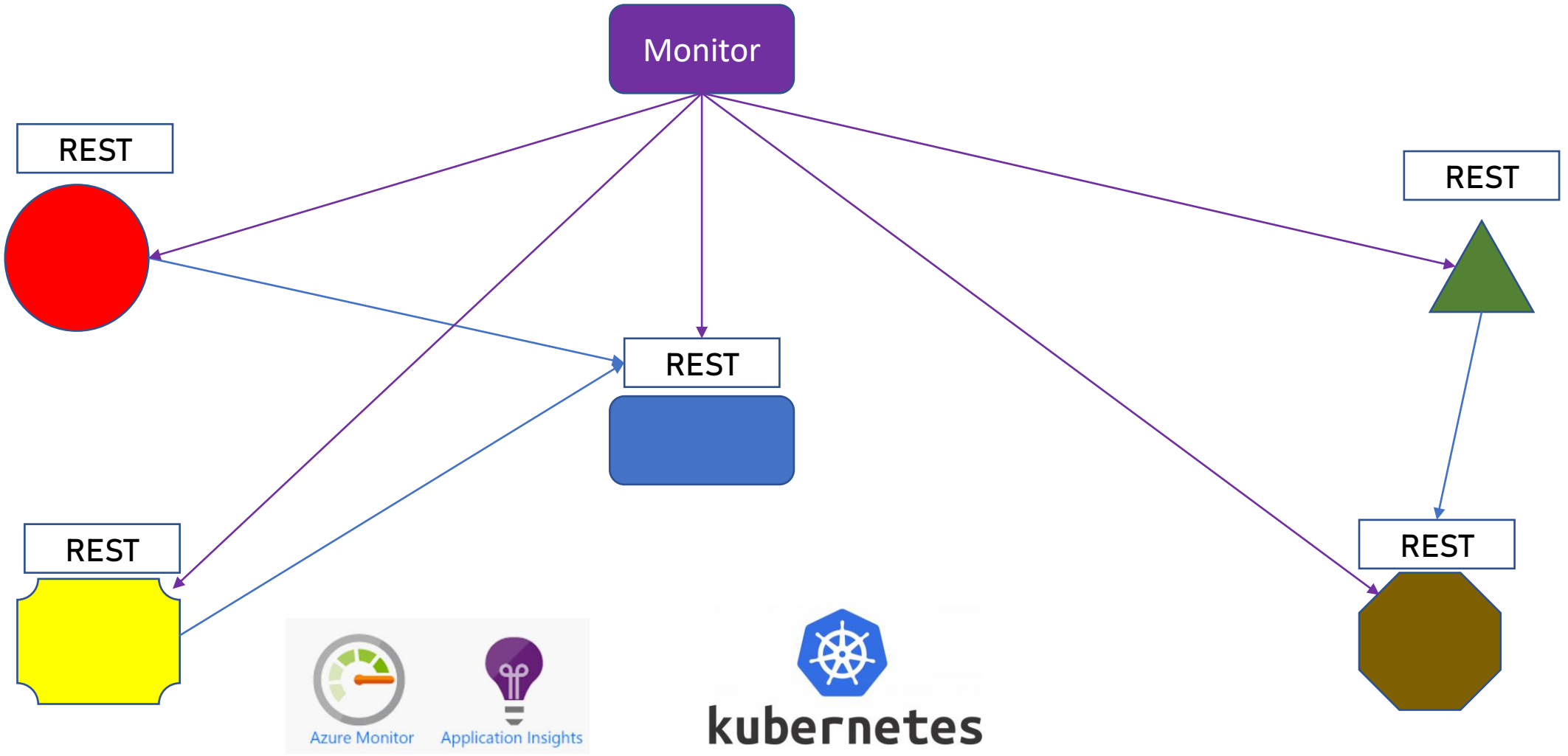
Design for Failure



- Catch the Exception
- Retry
- Log the Exception



Design for Failure



Design for Failure

- Motivation:
  - Increase system's reliability

Evolutionary Design

- The move to Microservices should be gradual
- No need to break everything apart
- Start small and upgrade each part separately

Summary

- These are guidelines, not mandatory instructions
- Adopt what works for you
- The Microservices world is rapidly changing
  - Follow new APIs, monitoring, cloud services etc.

Summary

- The most important attributes:
  - Componentization
  - Organized around business capabilities
  - Decentralized governance
  - Decentralized data management (when possible)
  - Infrastructure automation