Cloud Computing Patterns @ OOP 2017

Case Study and Discussion

http://www.cloudcomputingpatterns.org/oop17/

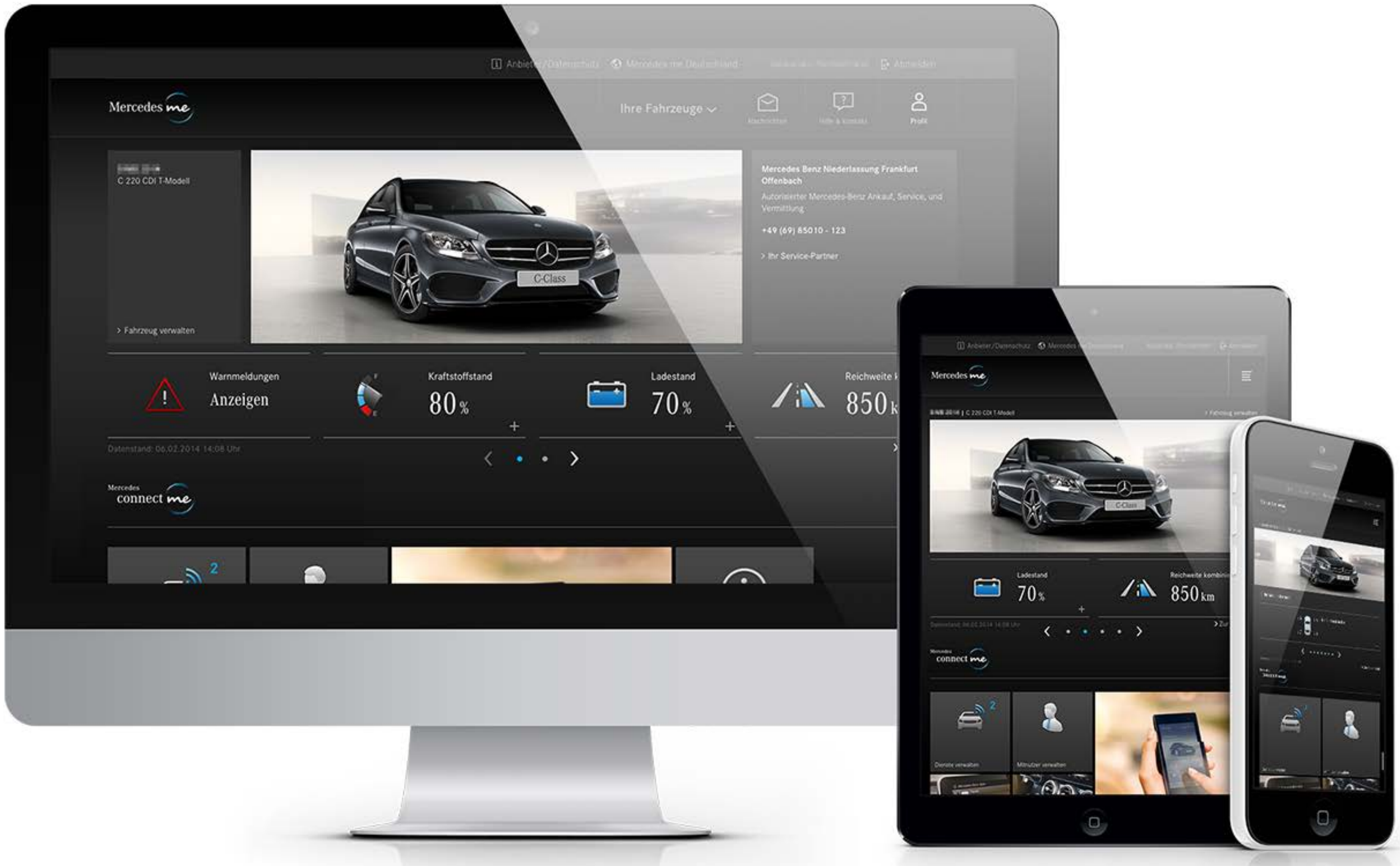
GET SLIDES HERE!

Dr. Christoph Fehling

christoph.fehling@daimler.com
Daimler AG

@ccpatterns

Mercedes Me


Mercedes me
Ihre Fahrzeuge
C 220 CDI T-Modell
Fahrzeug verwalten
C-Class
Mercedes Benz Niederlassung Frankfurt Offenbach
+49 (69) 85010 - 123
Ihr Service-Partner
Warnmeldungen
Anzeigen
Kraftstoffstand
80 %
Ladestand
70 %
850
Mercedes connect me
850 km

Agenda

How to compare cloud providers?
How do clouds affect application architectures?

How to run a private cloud?
How does a cloud affect an organization and processes?

2010 **2014** **2016** **OOP ☺**

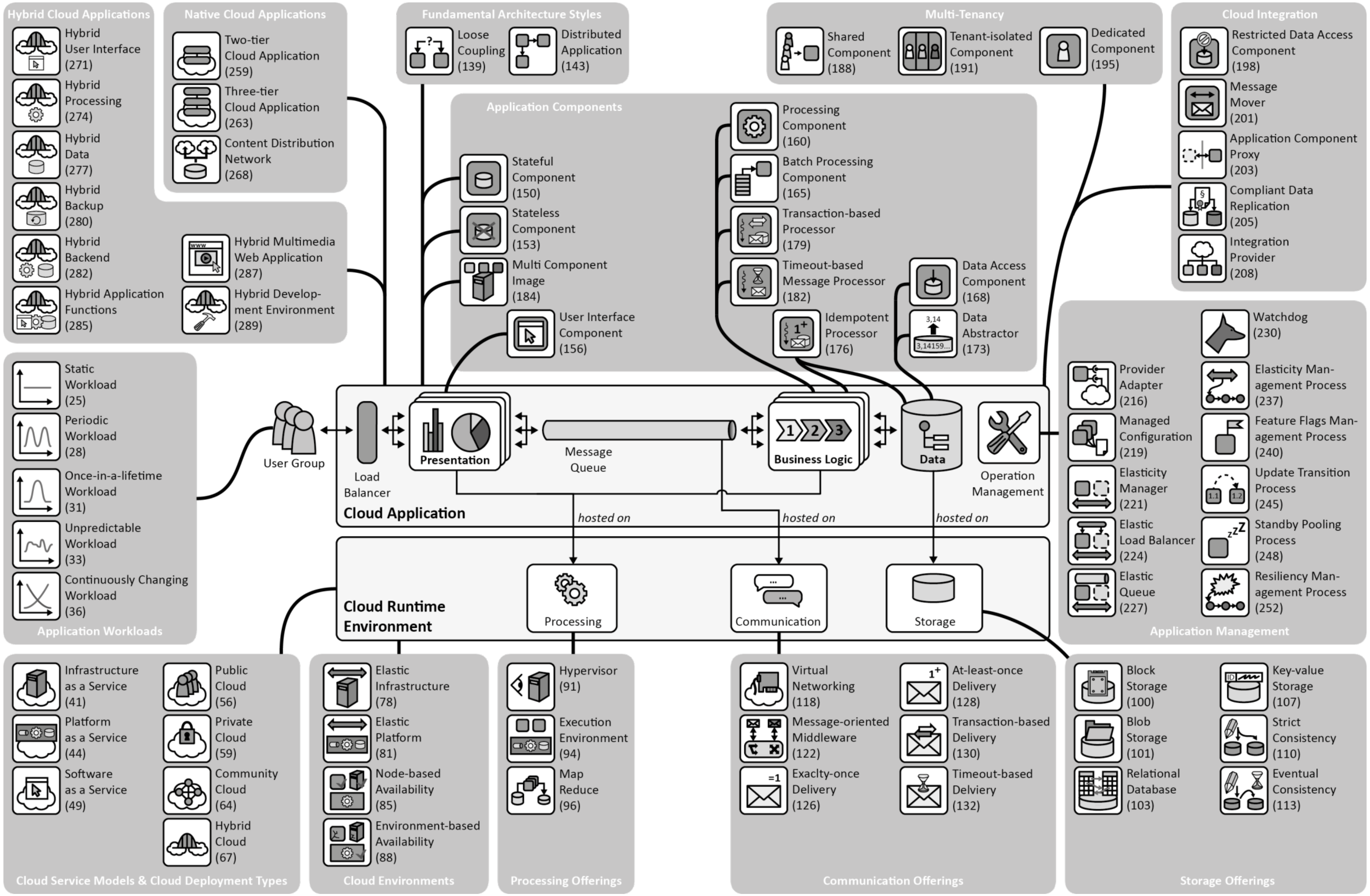
Part 1: Cloud Computing Patterns @ Mercedes Me

- What are the **cloud computing patterns**?
- What are the **properties of a cloud-native application**?
- How does a cloud-native application look like at **Mercedes Me**?

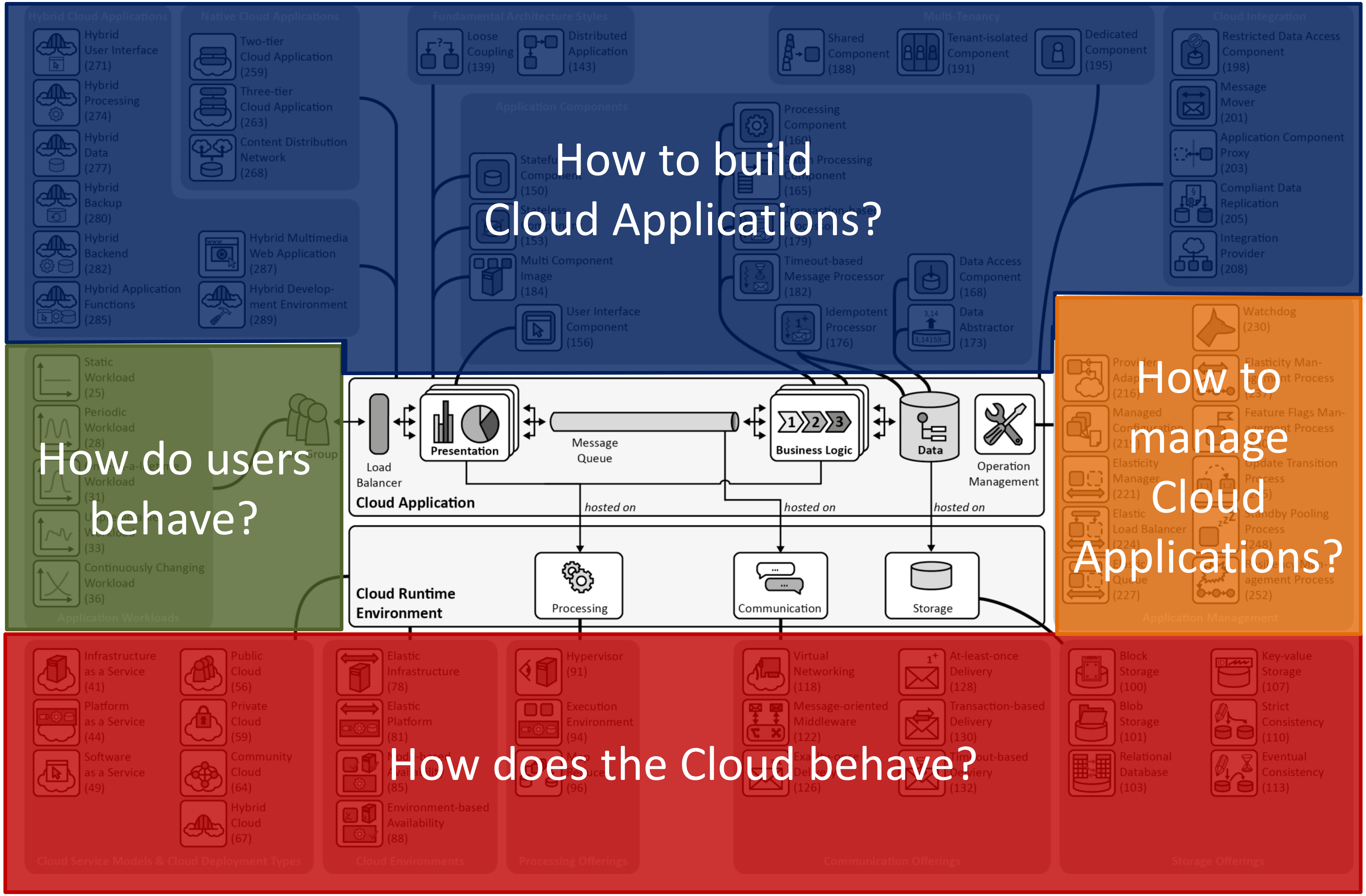
Part 2: the Non-technical “Stuff“ (Discussion)

- How does cloud computing affect **procurement processes**?
- Why are **licenses** of cloud products so problematic?
- How does cloud computing affect **organizational hierarchies**?

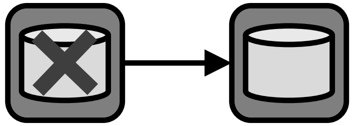
http://www.cloudcomputingpatterns.org



http://www.cloudcomputingpatterns.org



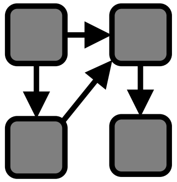
IDEAL Cloud Application Properties



Isolated State: most application components should be *stateless*. They do not handle:

Session State: state of the communication with the application.

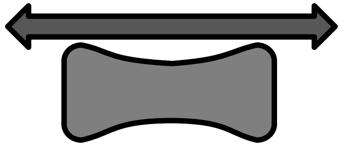
Application State: business data handled by the application.



Distribution: cloud applications are split up into multiple components…

… to utilize multiple cloud resources.

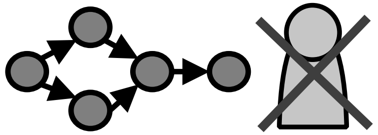
… because the cloud itself is a large distributed system.



Elasticity: cloud applications are scaled by adjusting resource numbers (*scaling out*) – not by *scaling up*:

Scale out: Increase performance by adding more resources.

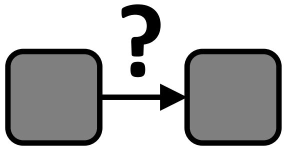
Scale up: Increase performance by improving existing resources.



Automated Management: management tasks during runtime have to be handled quickly.

Example: Cost reduction by adjusting pay-per-use resource numbers automatically.

Example: automatic reaction to resource failures.

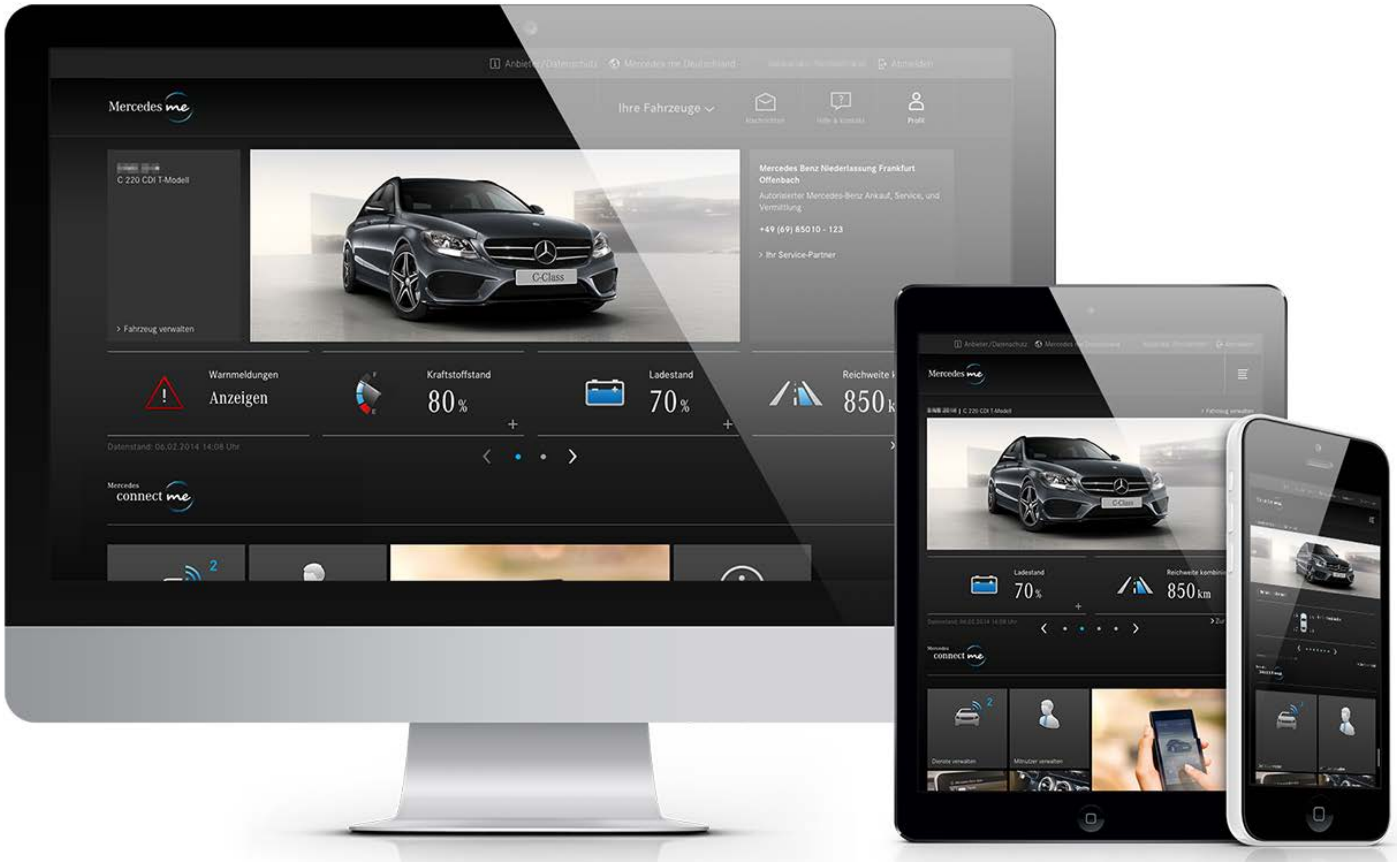


Loose Coupling: application components should not influence each other regarding factors such as availability, data format, data exchange rate.

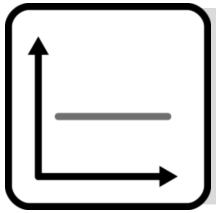
Example: failure of one application component does not cause failure of other components.

Part 1: Cloud Computing Patterns @ Mercedes Me

Mercedes Me

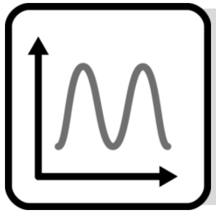

Mercedes me
Ihre Fahrzeuge
C 220 CDI T-Modell
> Fahrzeug verwalten
C-Class
Mercedes Benz Niederlassung Frankfurt Offenbach
+49 (69) 85010 - 123
> Ihr Service-Partner
Warnmeldungen
Anzeigen
Kraftstoffstand
80%
Ladestand
70%
850
Mercedes connect me

Workload Patterns @ Mercedes Me



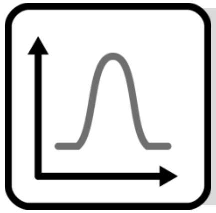
Static Workload

IT resources with an equal utilization over time experience static workload.



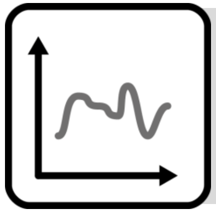
Periodic Workload

IT resources with a peaking utilization at reoccurring time intervals experience periodic workload.



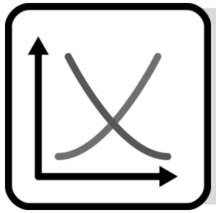
Once-in-a-Lifetime Workload

IT resources with an equal utilization over time disturbed by a strong peak occurring only once experience once-in-a-lifetime workload.



Unpredictable Workload

IT resources with a random and unforeseeable utilization over time experience unpredictable workload.



Continuously Changing Workload

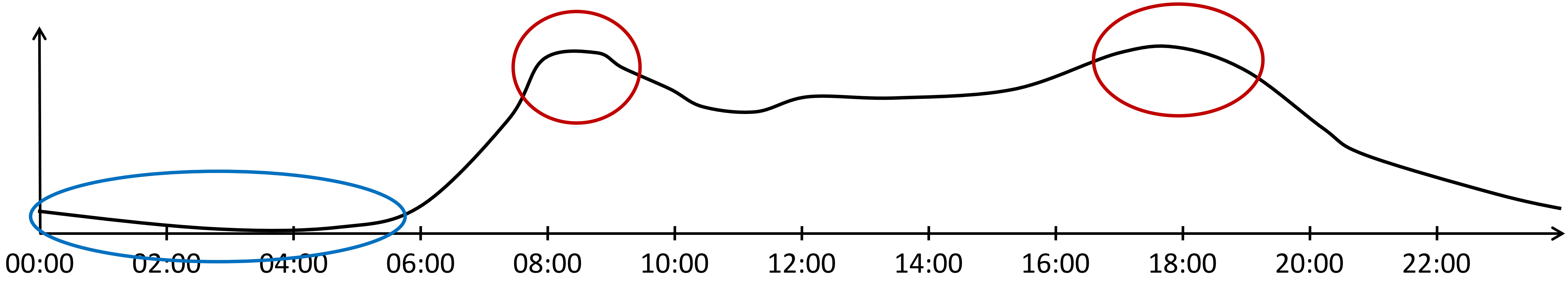
IT resources with a utilization that grows or shrinks constantly over time experience continuously changing workload.

IDEAL

Cloud-Native Application

- ○ Isolated State
- ○ Distribution
- ✔ Elasticity
- ✔ Automated Management
- ○ Loose Coupling

Car Status Updates during a single Day



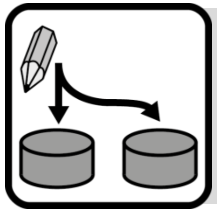
Peak workload: beginning and end of each day

→ Rush hour

Low workload: during each night

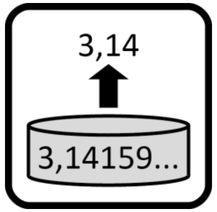
→ People are sleeping

Data Handling and Data Abstraction @ Mercedes Me



Strict Consistency
Data is stored at different locations to improve response time and to avoid data loss in case of failures while consistency of replicas is ensured at all times.

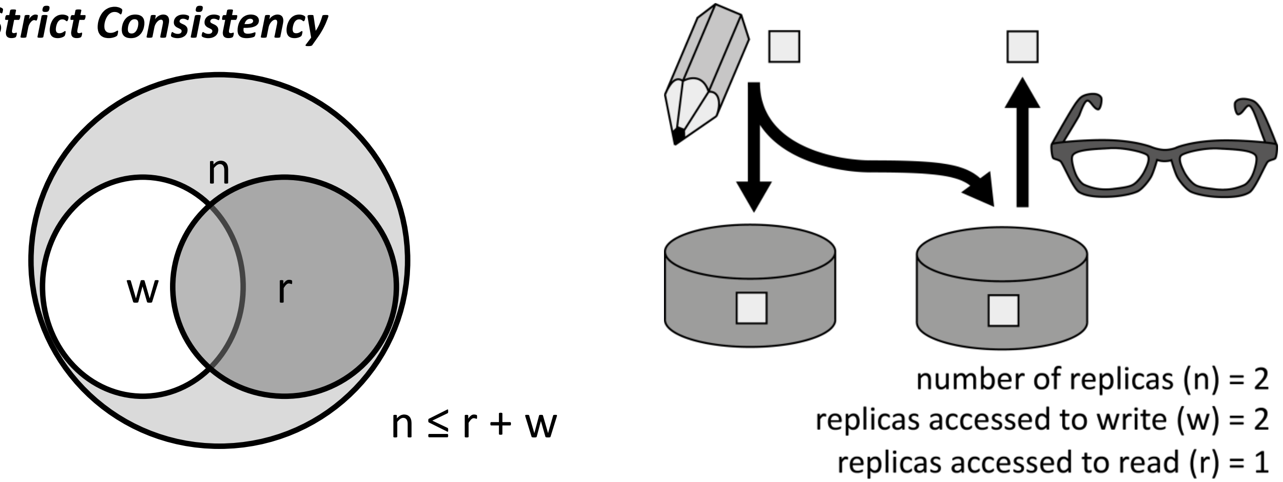
Eventual Consistency
Performance and availability of data in case of network partitioning are enabled by ensuring data consistency eventually and not at all times.



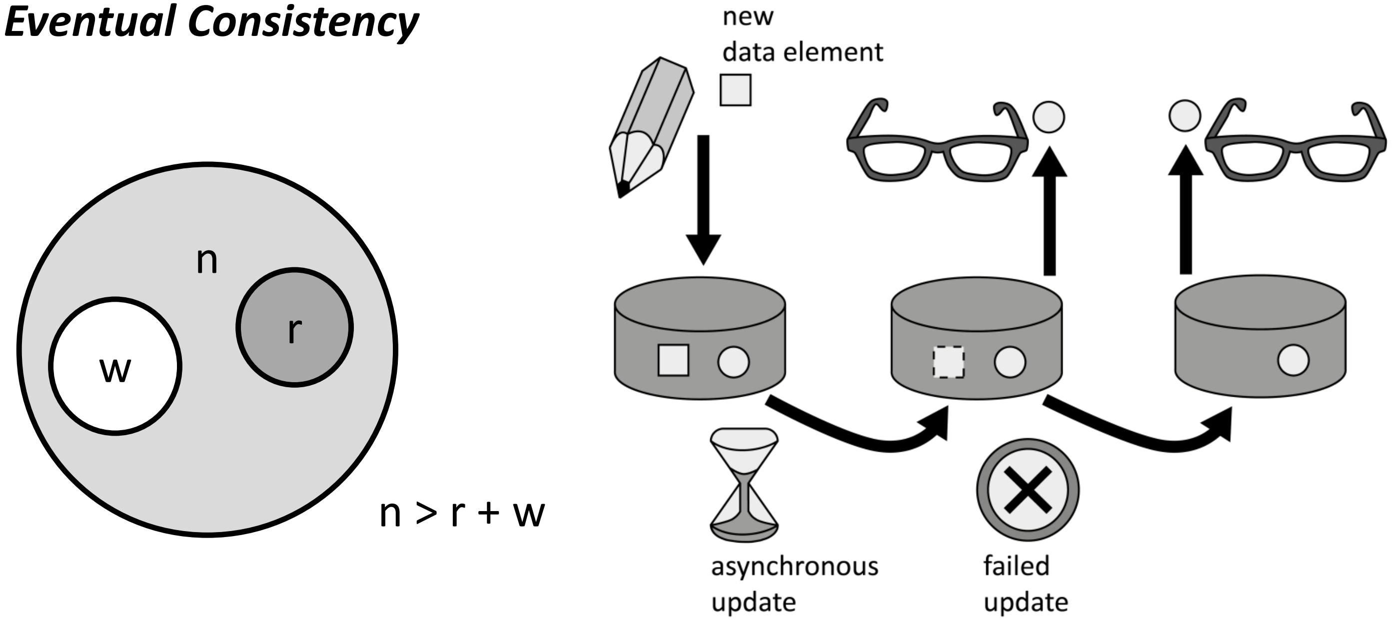
IDEAL
Cloud-Native Application

- [x] Isolated State
- [ ] Distribution
- [ ] Elasticity
- [ ] Automated Management
- [x] Loose Coupling

Strict Consistency



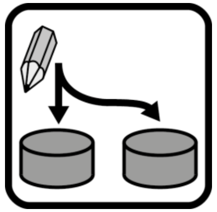
Eventual Consistency

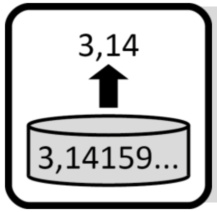


IDEAL

Cloud-Native Application

- ✔ Isolated State
- ○ Distribution
- ○ Elasticity
- ○ Automated Management
- ✔ Loose Coupling





Data Abstractor

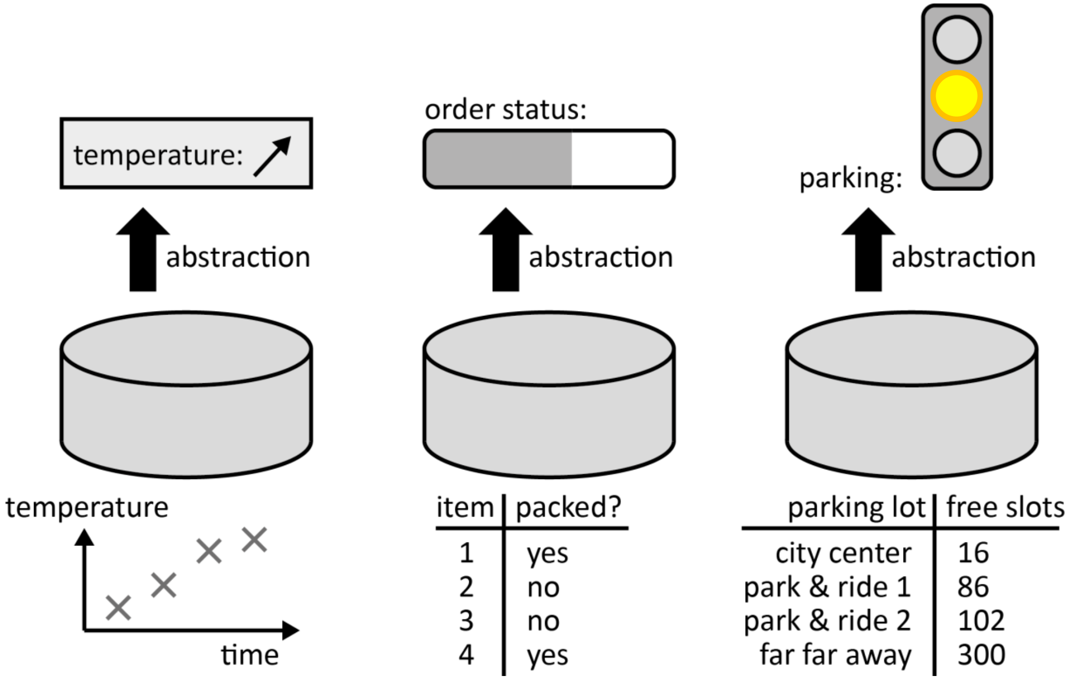
Data is abstracted to inherently support eventually consistent data storage through the use of abstractions and approximations.

IDEAL

Cloud-Native Application

- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling

Data Abstractor

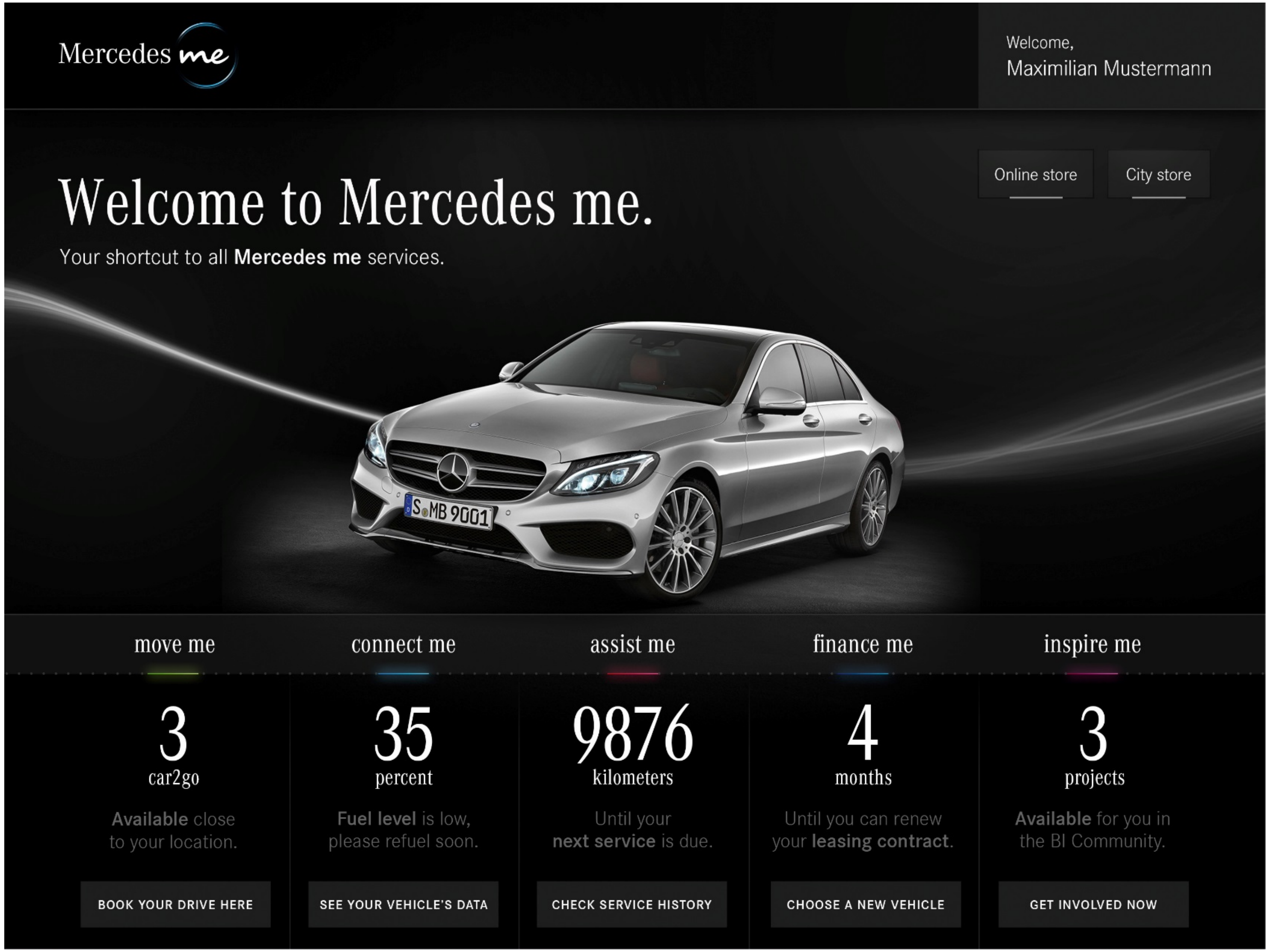


IDEAL

Cloud-Native Application

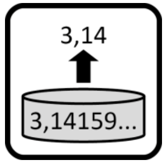
- ✔ Isolated State
- ○ Distribution
- ○ Elasticity
- ○ Automated Management
- ✔ Loose Coupling

Mercedes Me Portal



A lot of eventual consistent data:

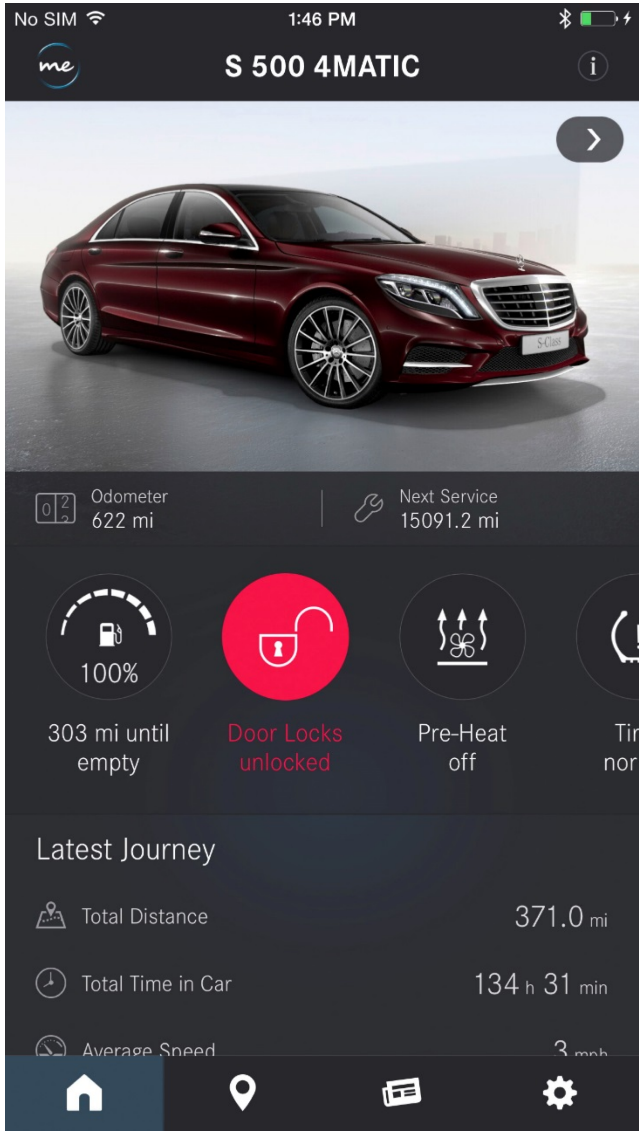
- Odometer
- Fuel level
- Service intervals
- Errors shown in the car



If the connectivity of the car is limited (parking garage), an obsolete status is displayed!

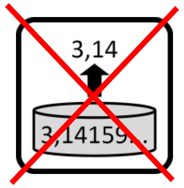
But can all data be treated this way?

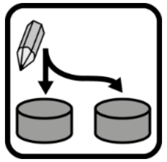
Mercedes Me App



Same as in the portal:

- Odometer
- Fuel level
- Service intervals
- Errors shown in the car



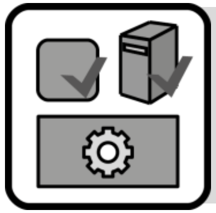


But certain interactions **must not be abstracted!**

- Door lock status
- Heating status
- Engine status (cooling)

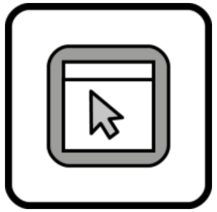
→ **Data abstraction and eventual consistency has to be evaluated for each data element!**

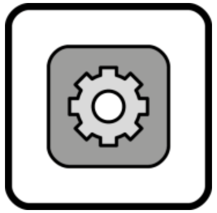
Mercedes Me Microservice Template



Node-based Availability
A cloud provider guarantees the availability of nodes, such as individual virtual servers, middleware components or hosted application components.

Environment-based Availability
A cloud provider guarantees the availability of the environment hosting individual nodes, such as virtual servers or hosted application components.

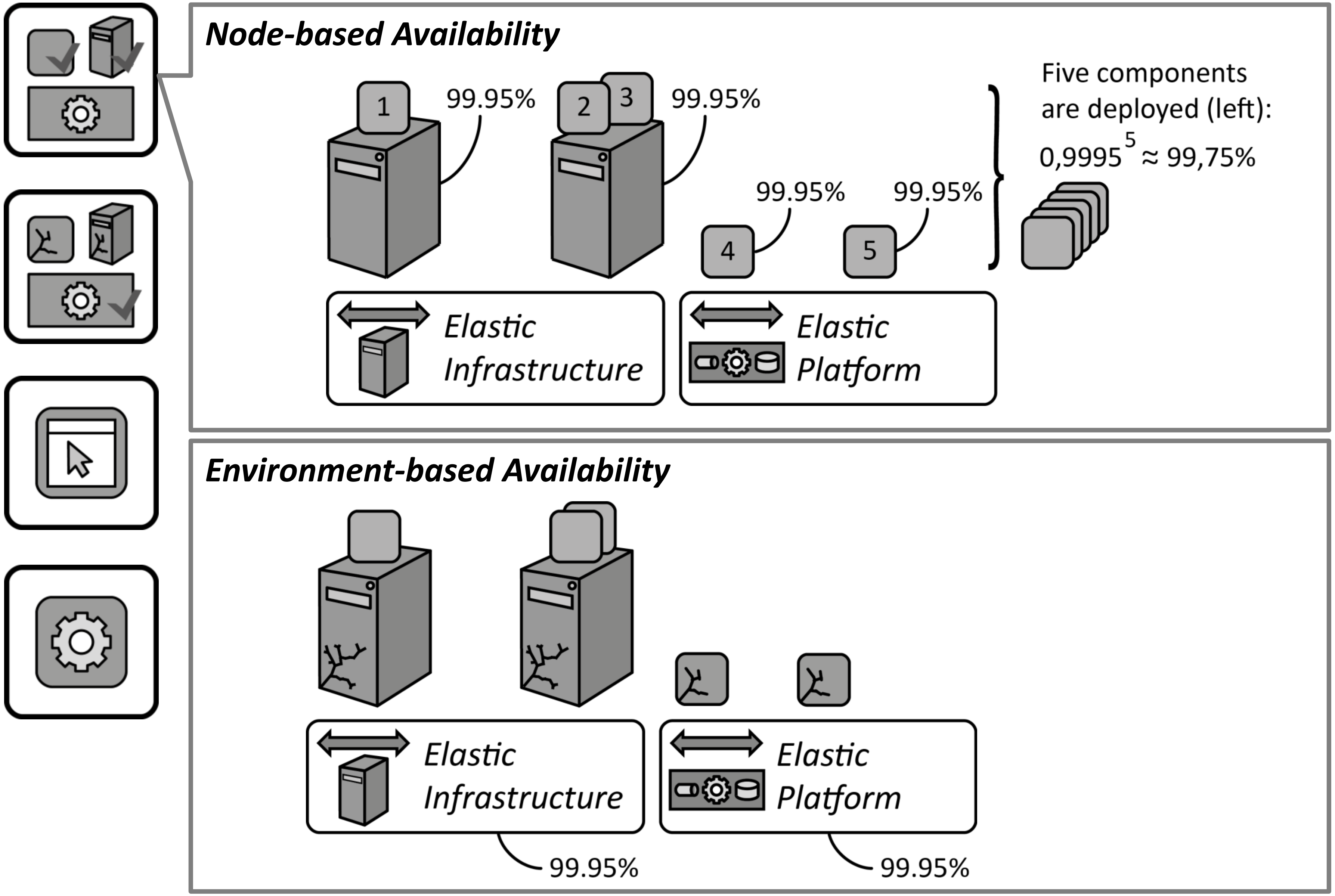




IDEAL

Cloud-Native Application

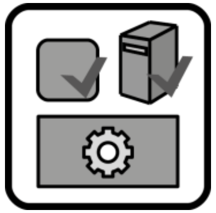
- ○ Isolated State
- ✓ Distribution
- ○ Elasticity
- ✓ Automated Management
- ○ Loose Coupling

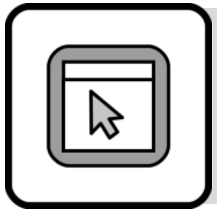


IDEAL

Cloud-Native Application

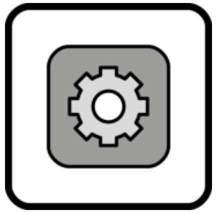
- [ ] Isolated State
- [x] Distribution
- [ ] Elasticity
- [x] Automated Management
- [ ] Loose Coupling





User Interface Component

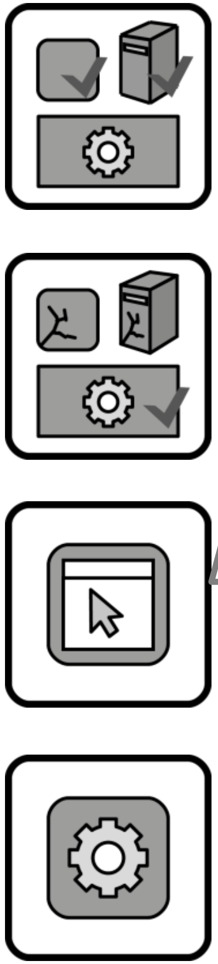
Synchronous user interfaces are accessed by humans, while application-internal interaction is realized asynchronously to ensure loose coupling.



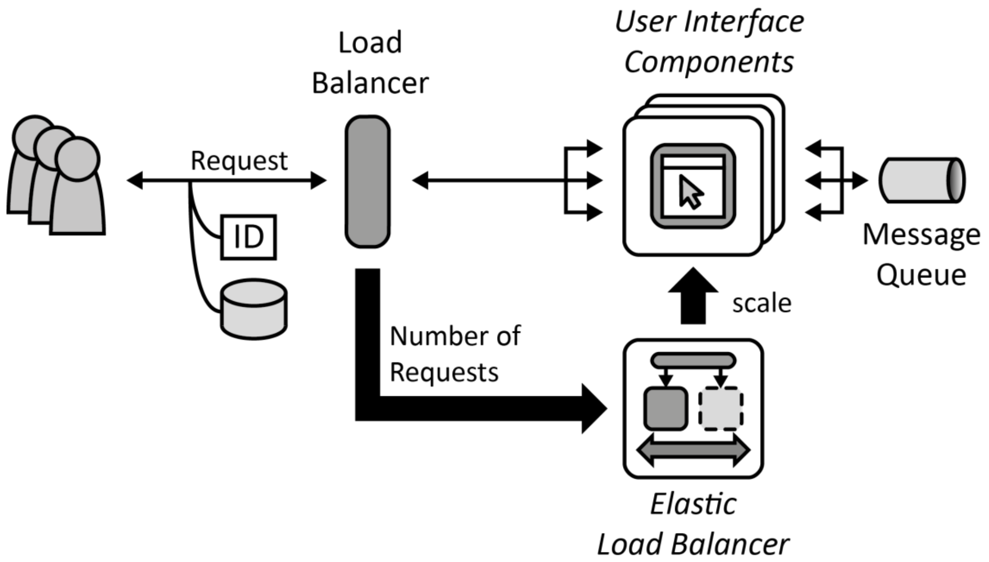
IDEAL

Cloud-Native Application

- [ ] Isolated State
- [x] Distribution
- [x] Elasticity
- [x] Automated Management
- [ ] Loose Coupling

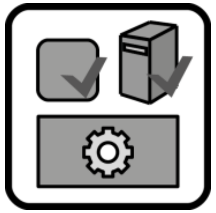


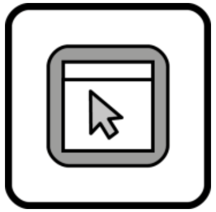
User Interface Component

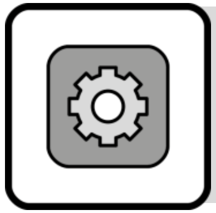


IDEAL
Cloud-Native Application

- Isolated State
- ✓ Distribution
- ✓ Elasticity
- ✓ Automated Management
- Loose Coupling







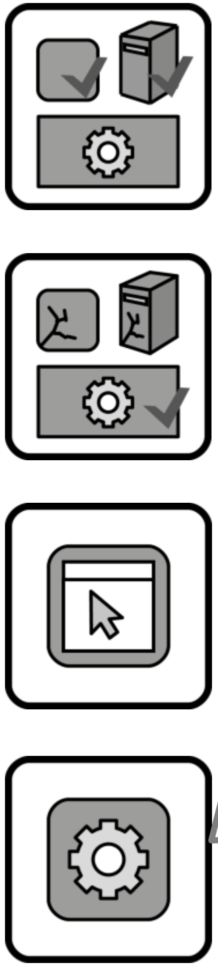
Processing Component

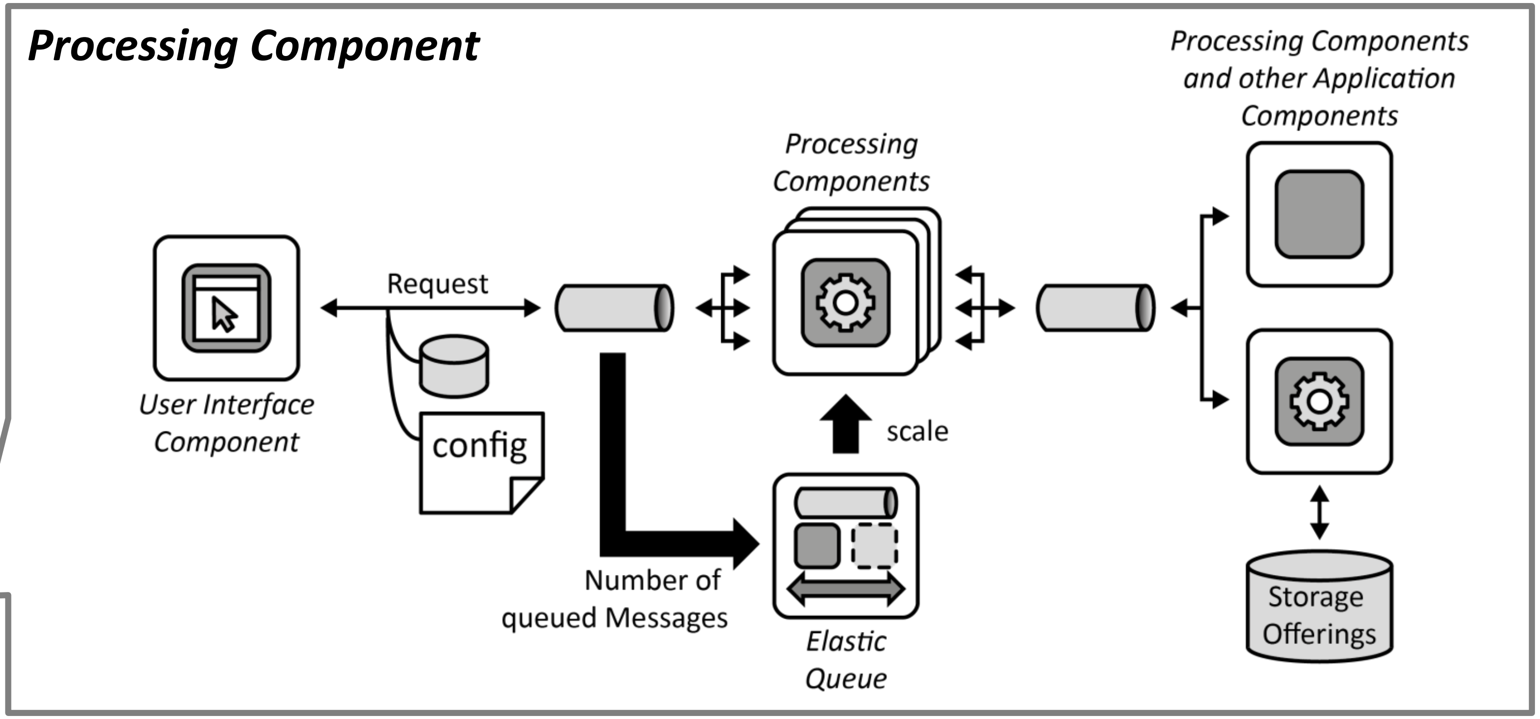
Processing functionality is handled by elastically scaled components.

IDEAL

Cloud-Native Application

- ◯ Isolated State
- ✔ Distribution
- ✔ Elasticity
- ✔ Automated Management
- ◯ Loose Coupling



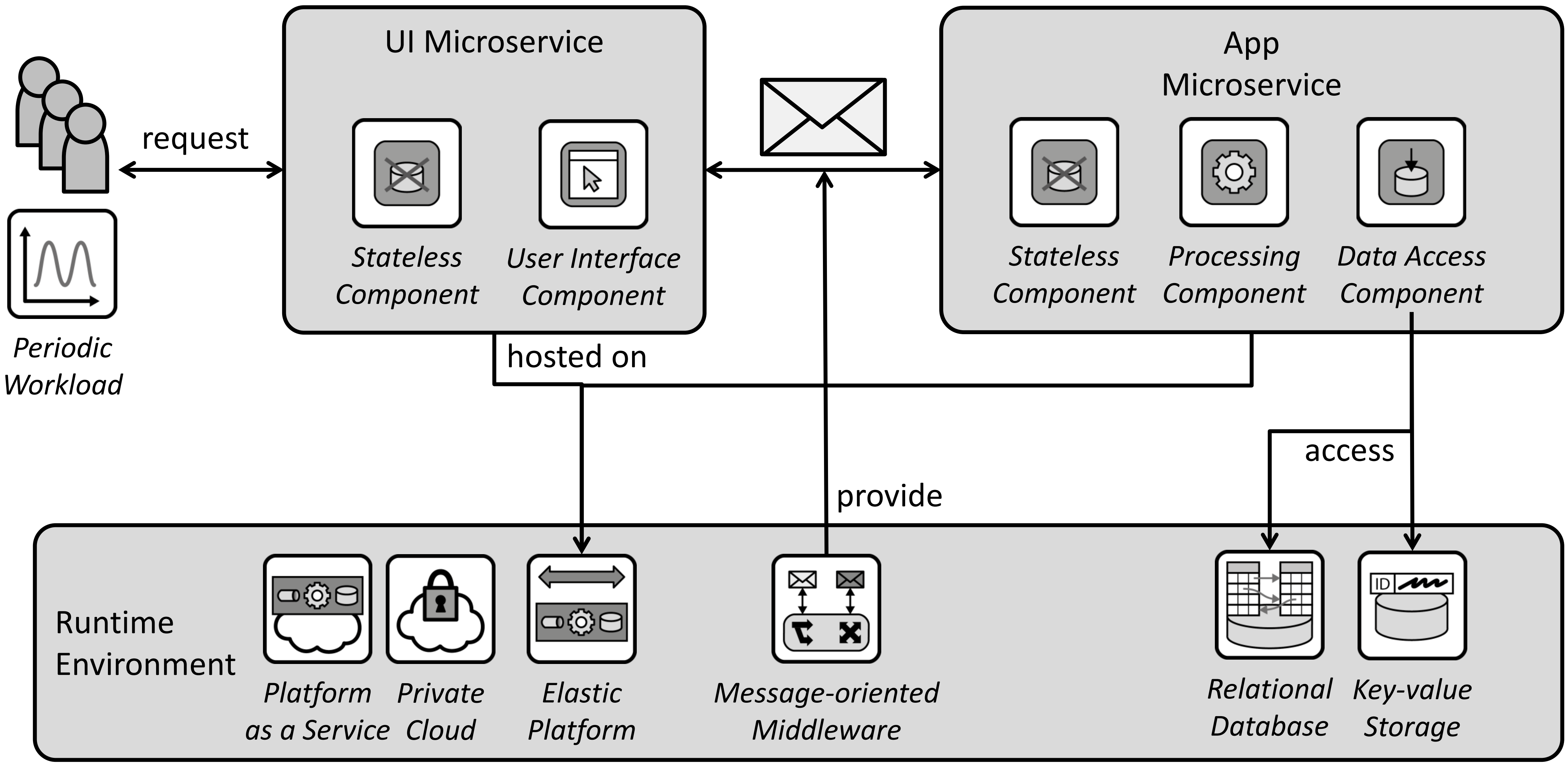


IDEAL

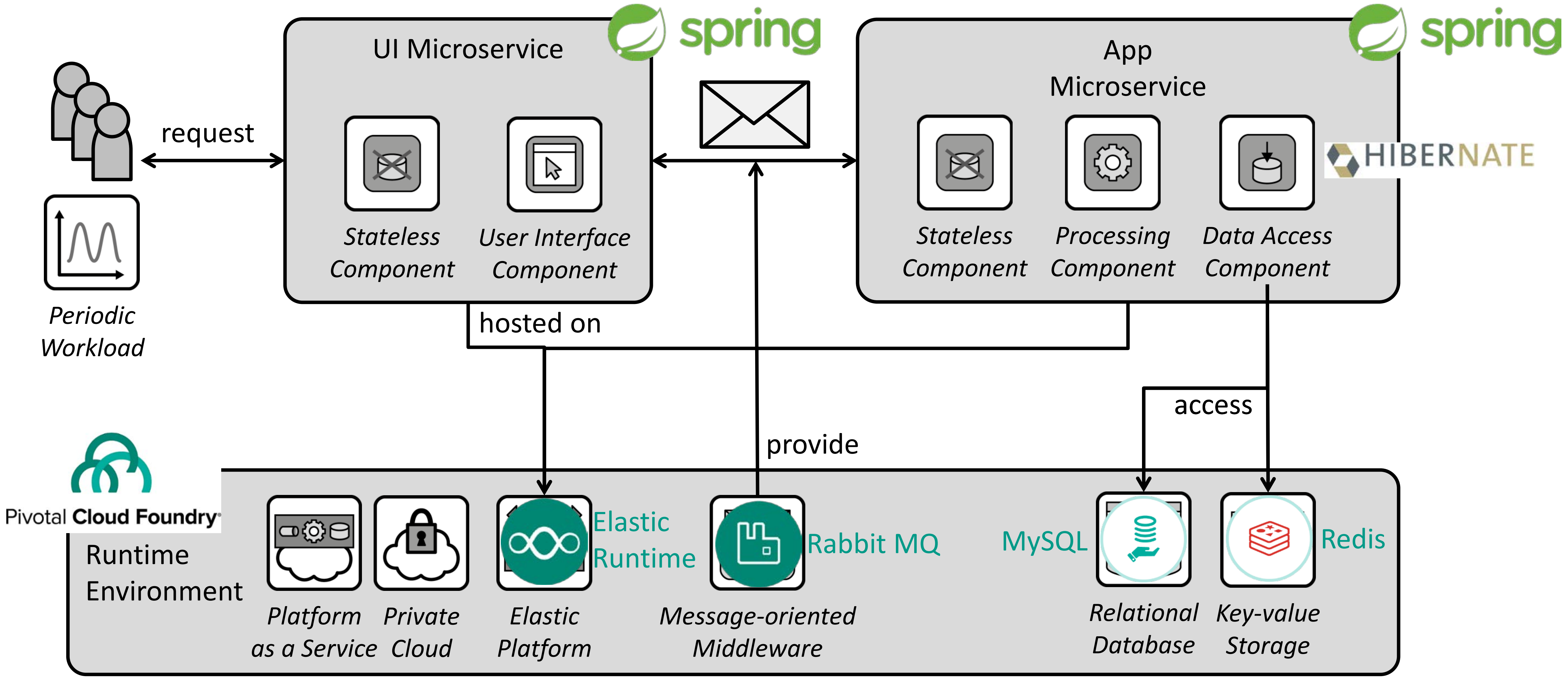
Cloud-Native Application

- Isolated State
- ✓ Distribution
- ✓ Elasticity
- ✓ Automated Management
- Loose Coupling

Mercedes Me Microservice Template



Mercedes Me Microservice Template



Lessons Learned

Moving from **Node-based Availability to Environment-based Availability** was the most **challenging factor**.

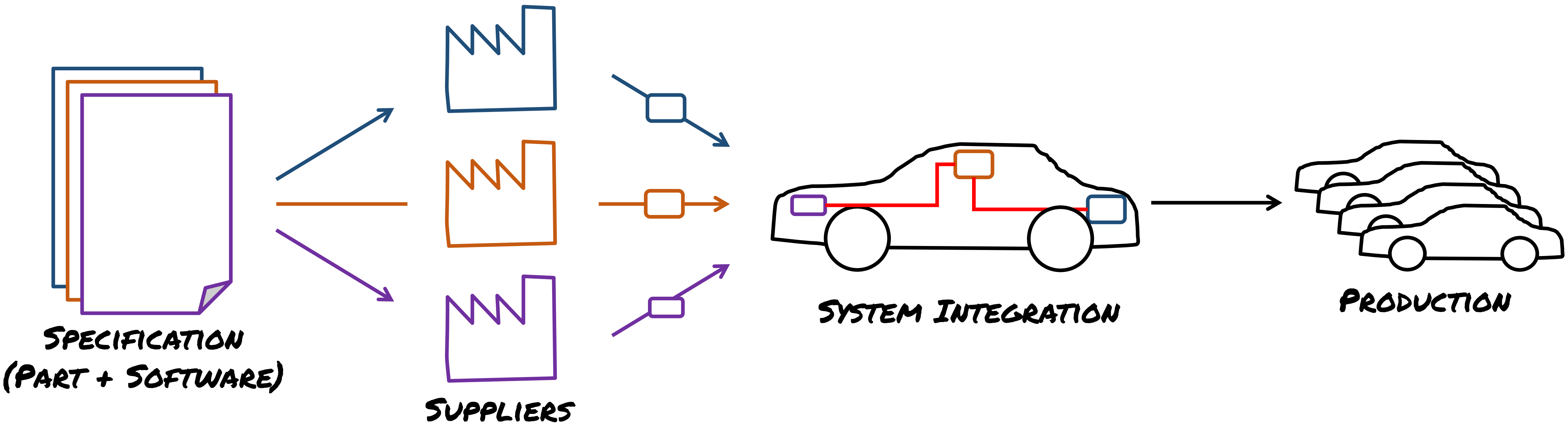
Cloud Computing is a significant enabler for **agile development**.

Soft factors **(Procurement, Licensing, Organizational Hierarchy)** pose a significant challenge!

Part 2: The Non-technical “Stuff”: Procurement, Licenses, Organizational Hierarchies

1. Impact of Cloud Computing on Procurement Processes

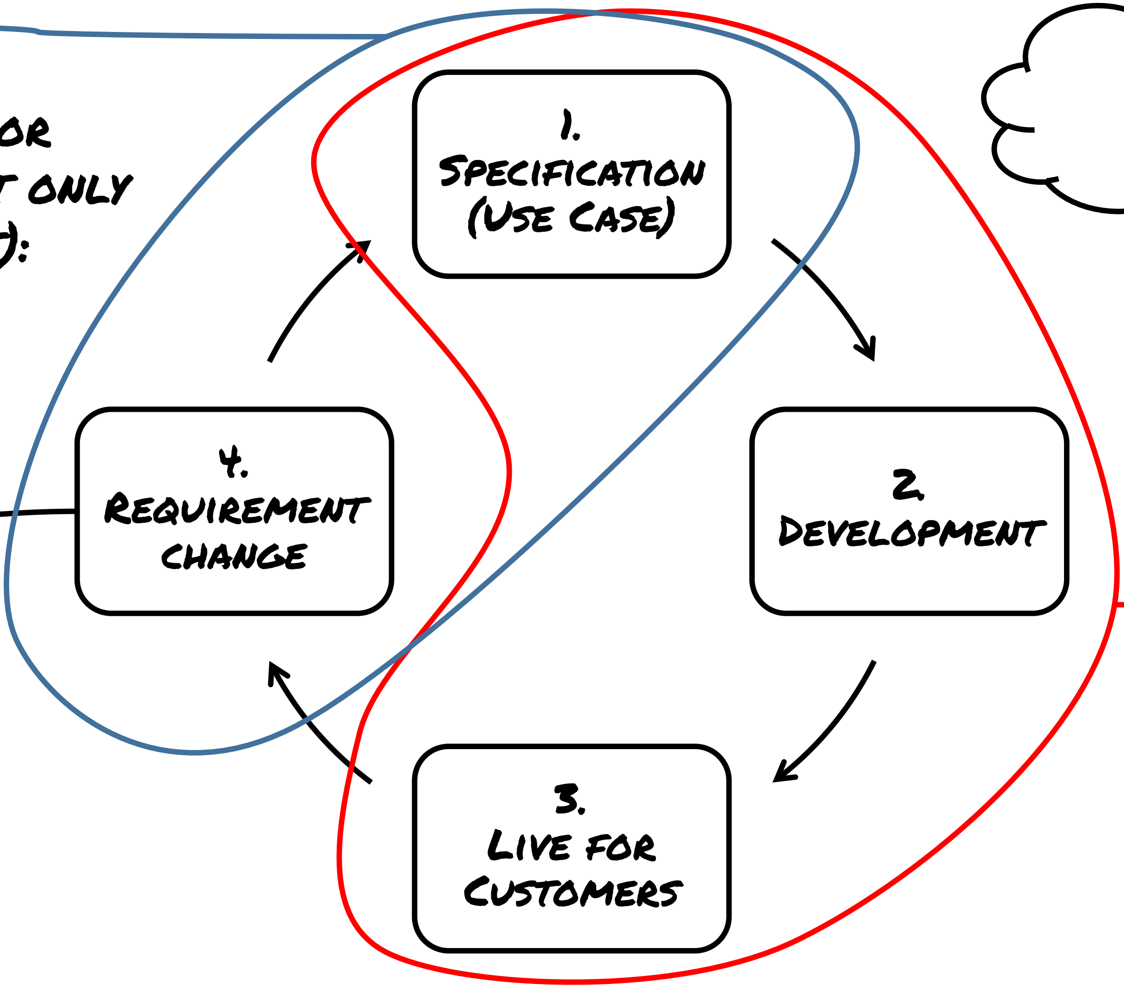
Procurement of a Car Part (simplified view)

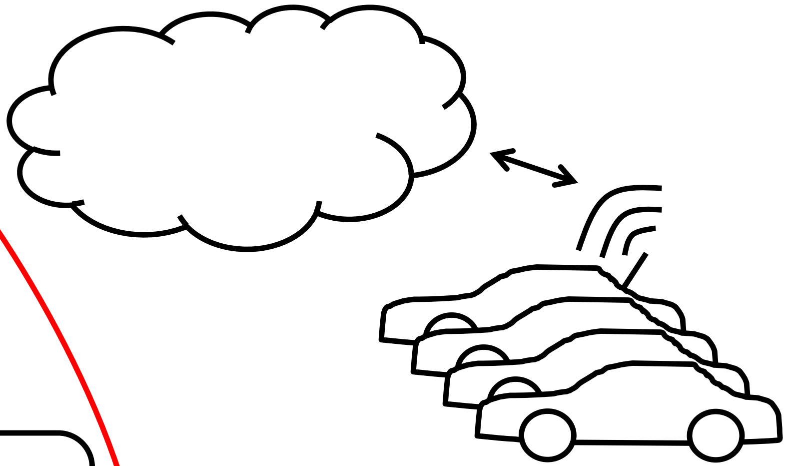


- Software scales with the number of cars as it is installed in each one of them.
- Improvements / Fixes usually involve a visit to the shop.

Procurement of a Car ~~Part~~ Backend System

Change request (€€€) or New Procurement (but only one Supplier can do it): Vendor Lock-In!



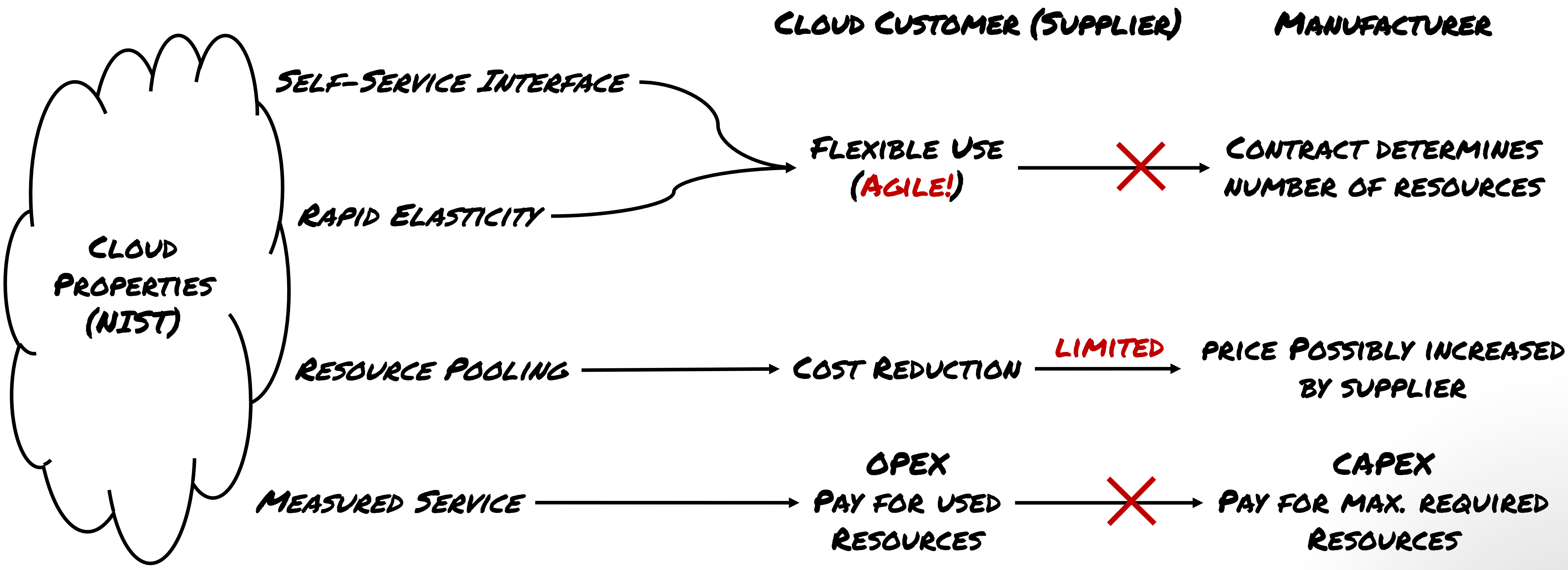


- User Behavior Change
- Cloud Interfaces
- New car models
- New functions
- Laws and regulations
- ...

Fixed price procurement

-> Procurement processes have to reflect that a backend is not a car part!

Procurement of Cloud-Based Backend Systems

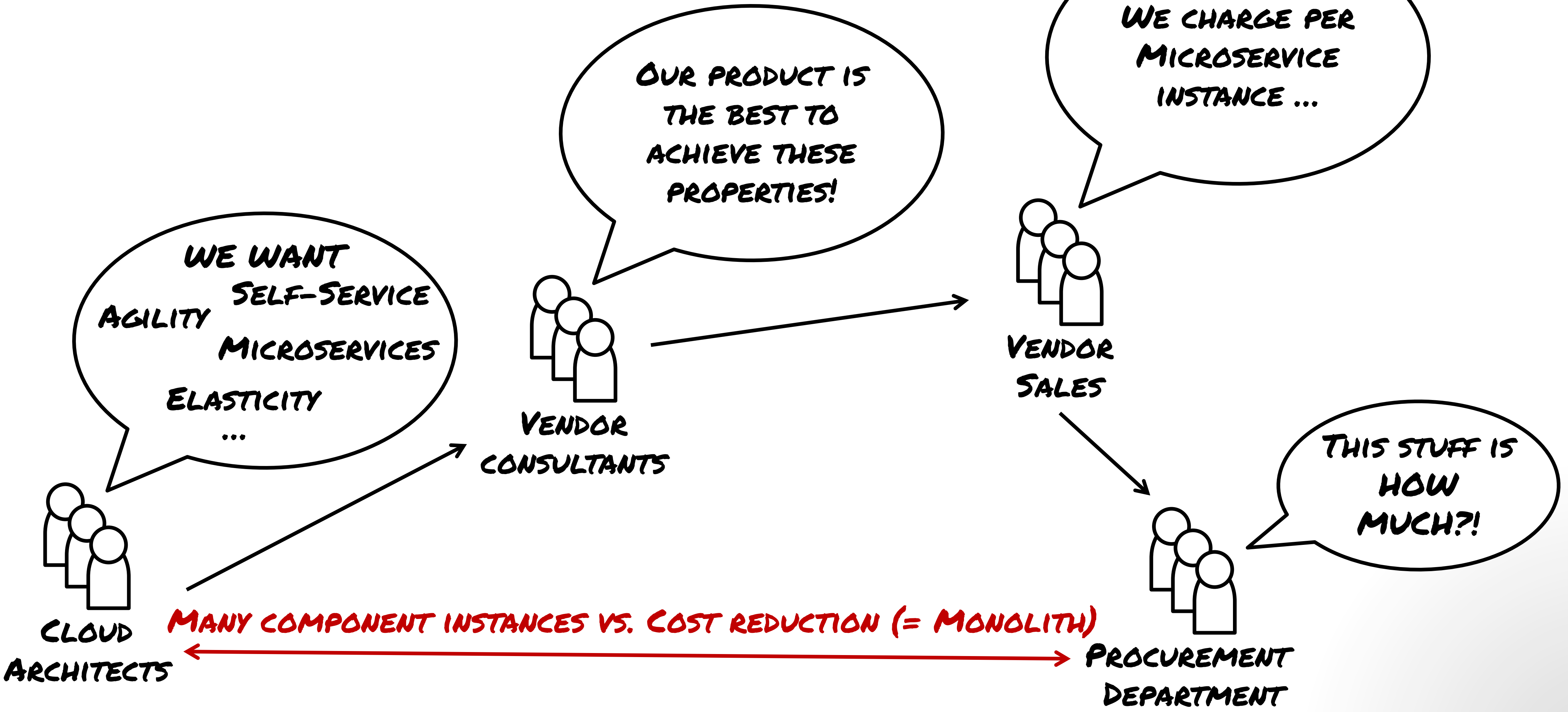


-> The Cloud Customer benefits from Cloud Properties. This should not be the supplier...

-> The Benefits of clouds are highly dependent on procurement Processes!

2. Licenses in the cloud

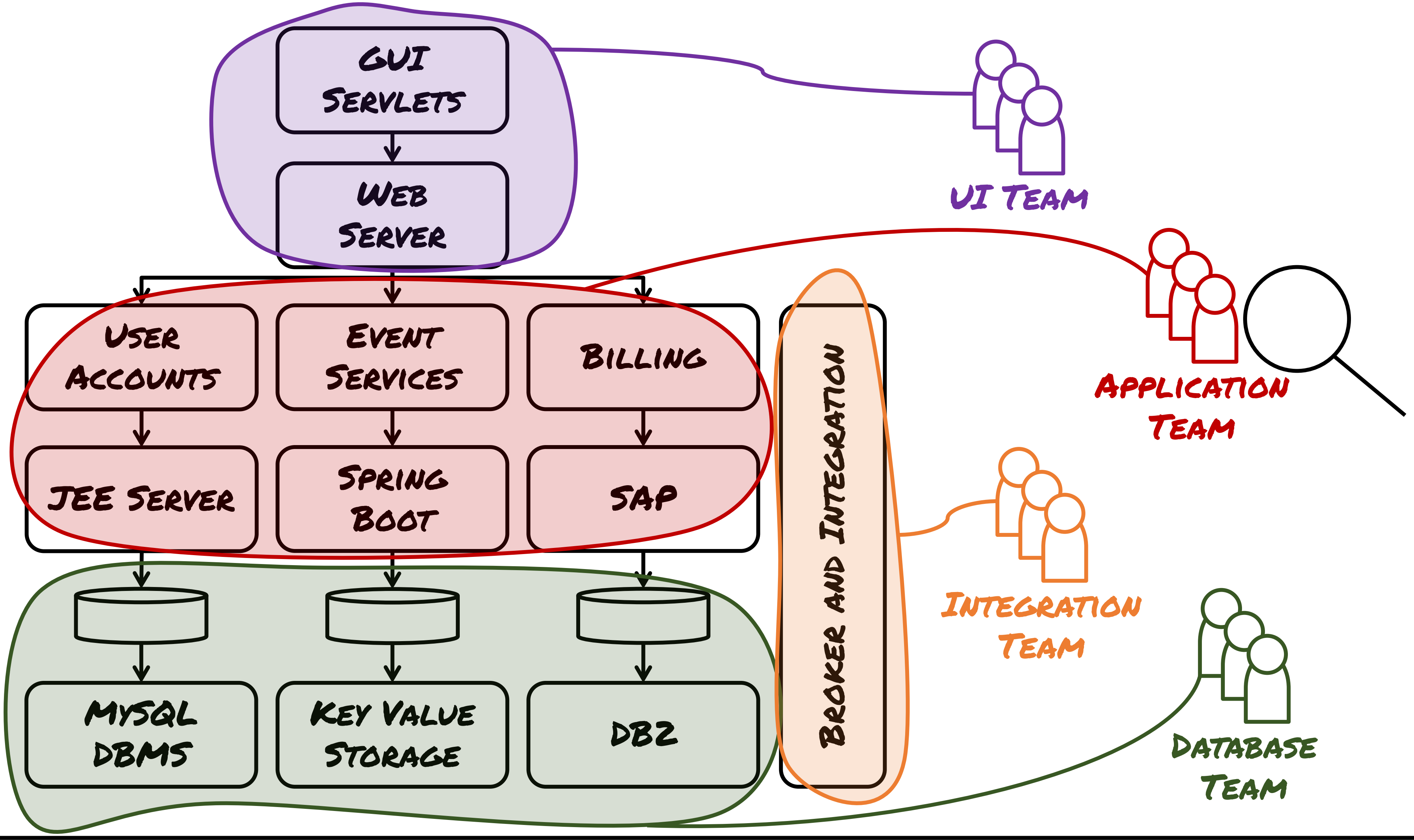
The License Conflict



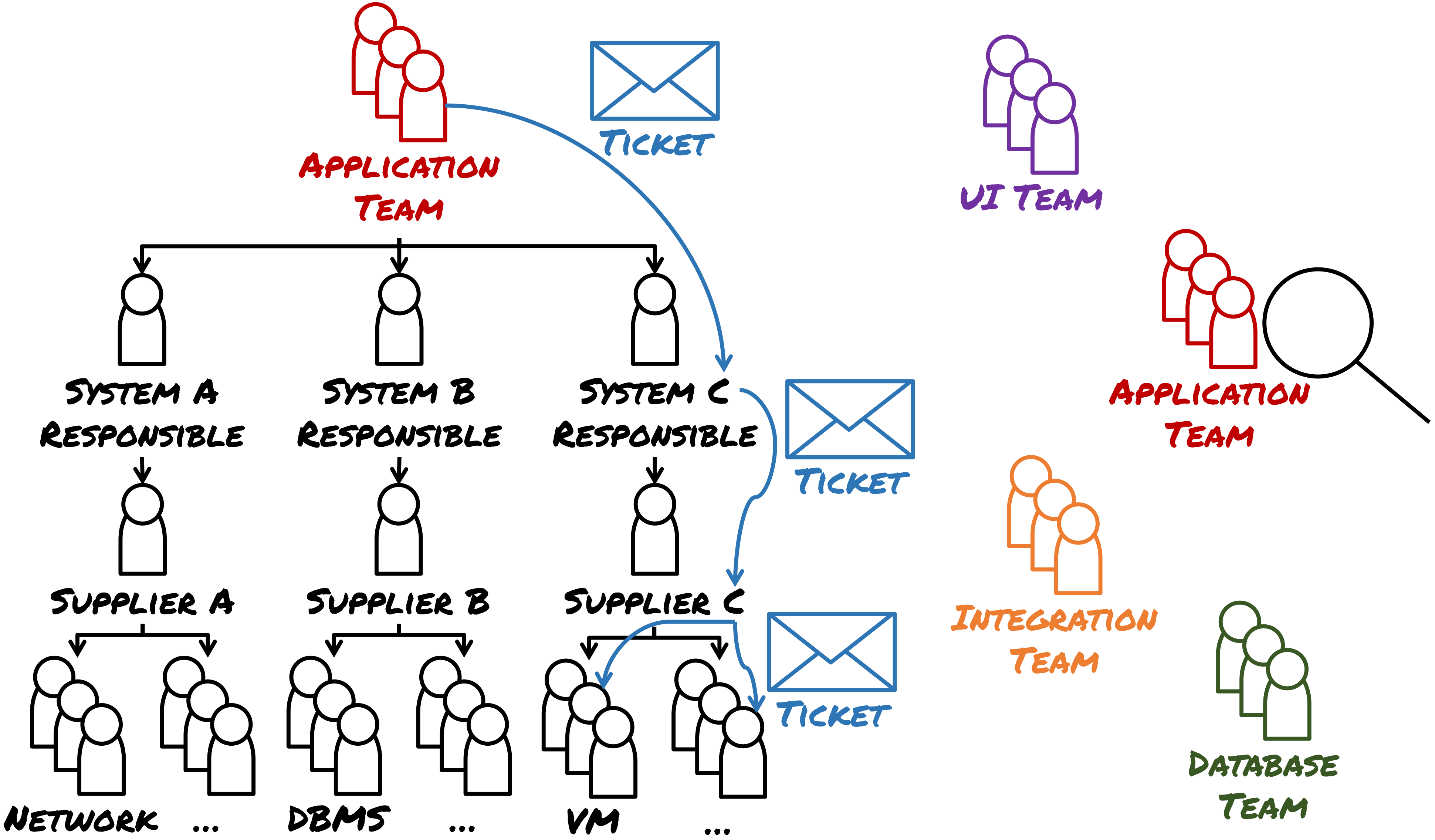
-> Licensing models of many cloud products conflict with our architectural goals!

3. Impact of Cloud Computing on Organization Hierarchies

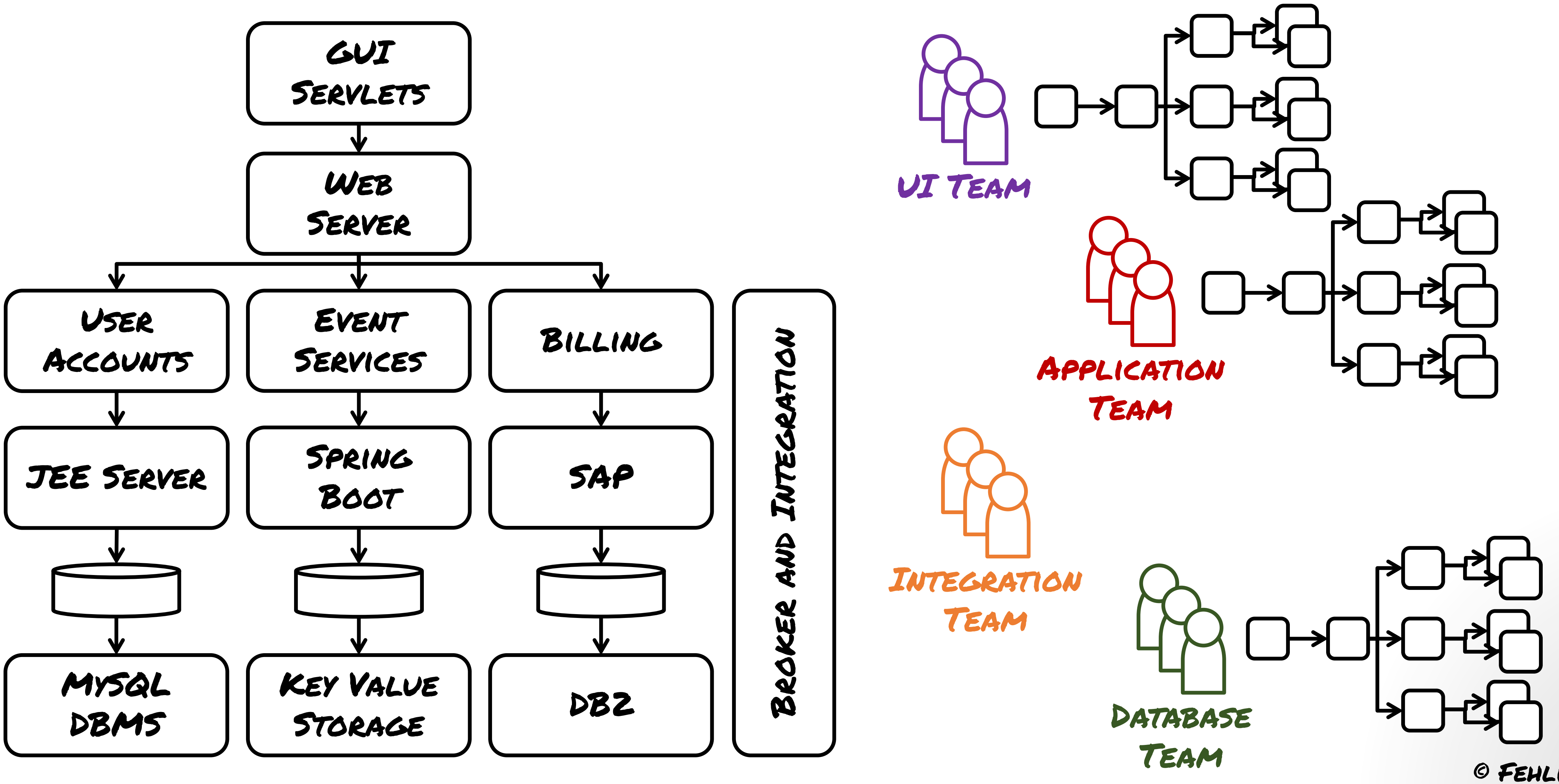
Organization Hierarchies and Clouds



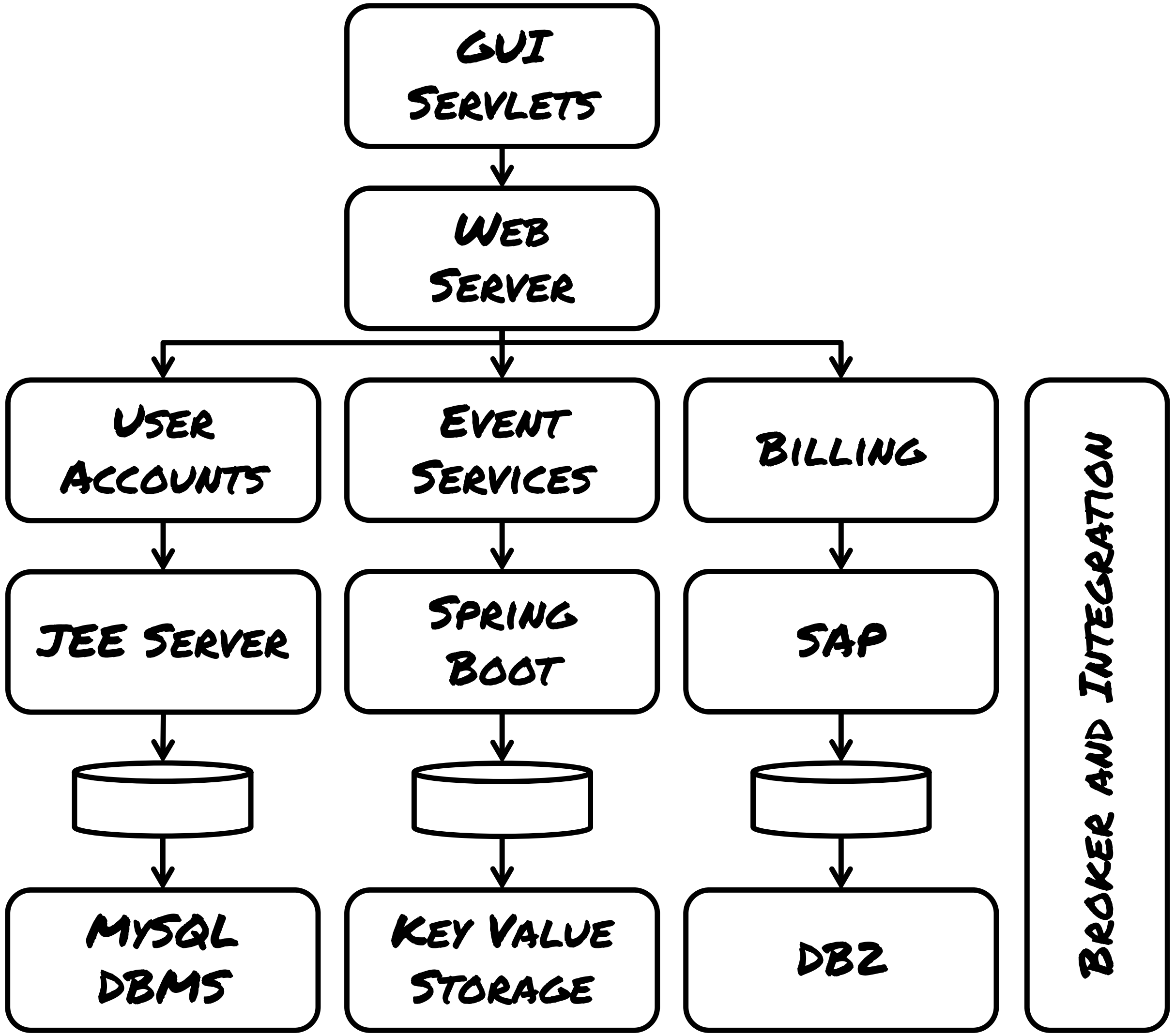
Organization Hierarchies and Clouds



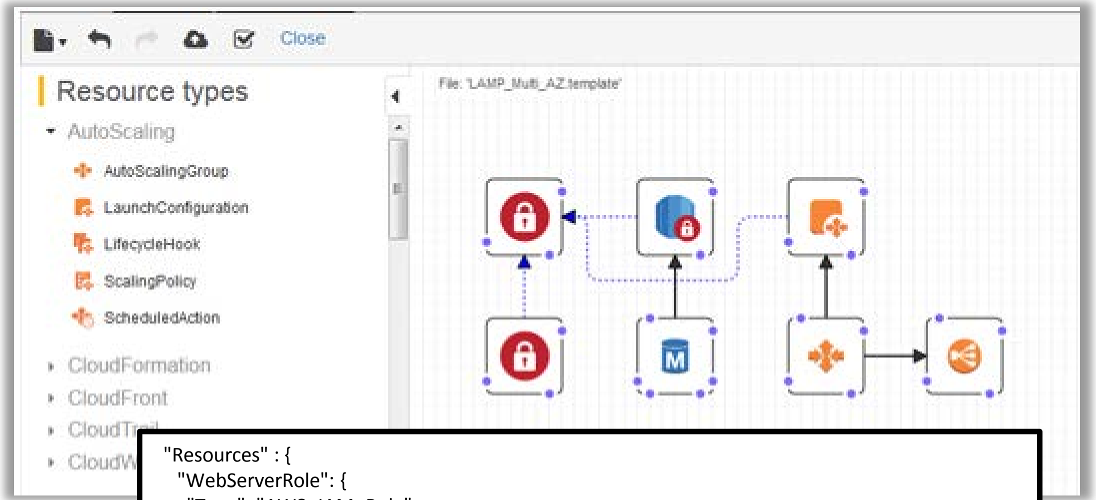
Organization Hierarchies and Clouds



Side Note: Clouds Thrive on Automation!



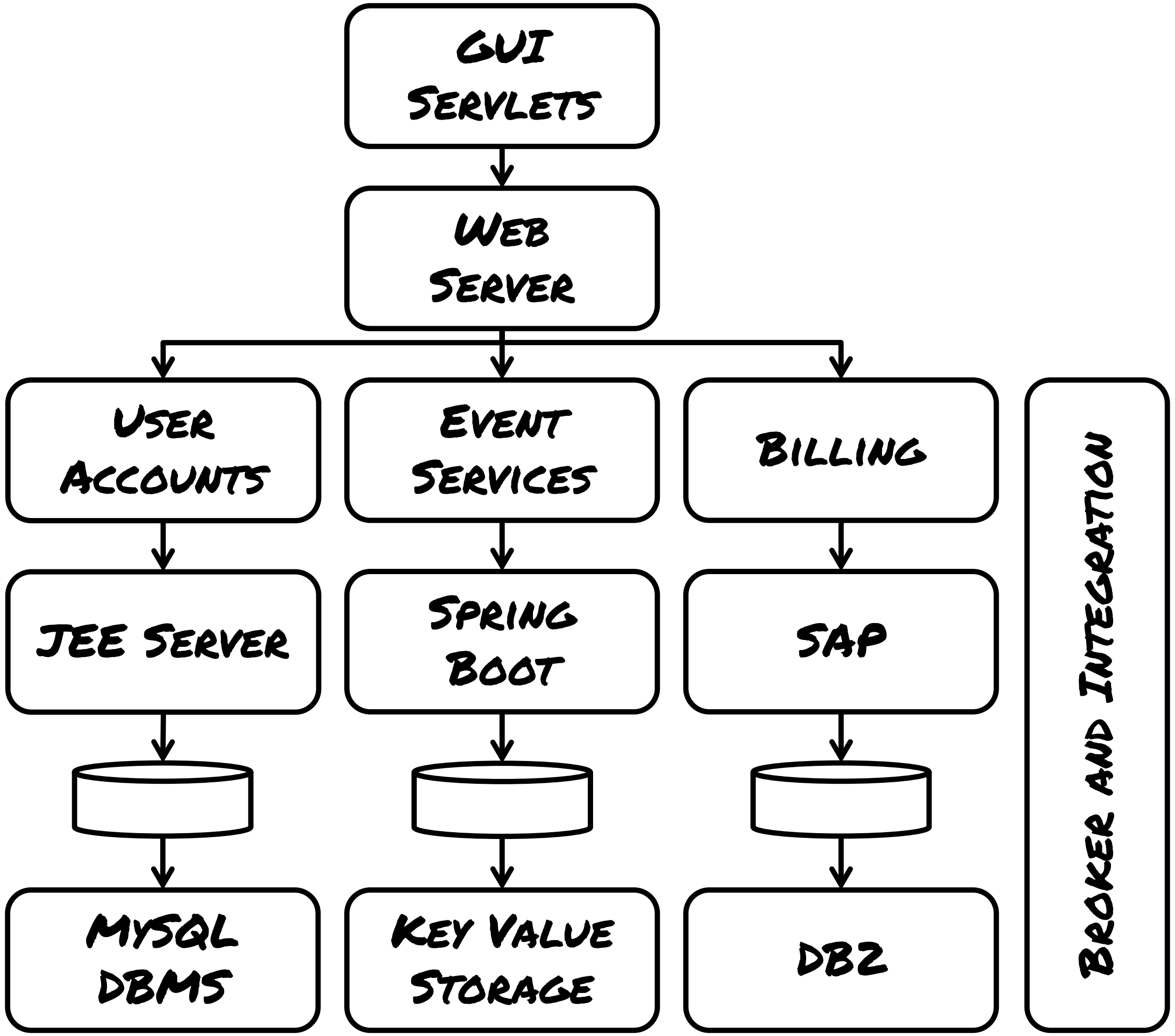
Amazon Cloud Formation




```
"Resources" : {
  "WebServerRole": {
   "Type": "AWS::IAM::Role",
   "Properties" : {
    "AssumeRolePolicyDocument" : {
     "Statement"  : [{
      "Effect"   : "Allow",
      "Principal": { "Service": [{ "Fn::FindInMap" : ["Region2Principal", {"Ref" : "AWS::Region"},
"EC2Principal"]}]},
      "Action"   : [ "sts:AssumeRole" ]
     } ]
    },
    "Path": "/"
   }
  },

  "WebServerRolePolicy": {
   "Type": "AWS::IAM::Policy",
   "Properties": {
    "PolicyName" : "WebServerRole",
```

Side Note: Clouds Thrive on Automation!



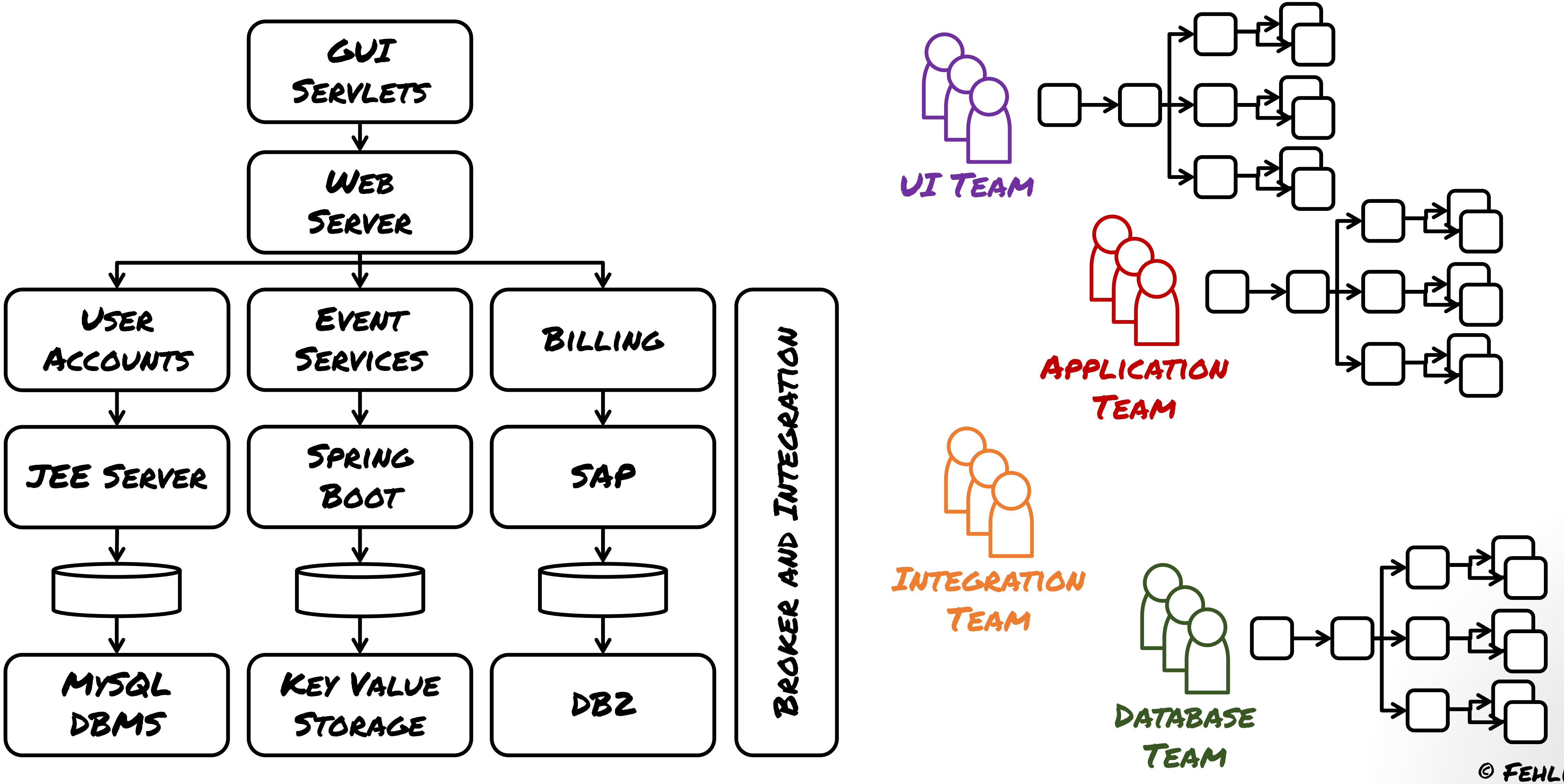
Docker Compose

CHEF™

ANSIBLE

->Infrastructure as Code is mandatory to benefit from the cloud Properties!

OK! Let's Automate our IT Stack!

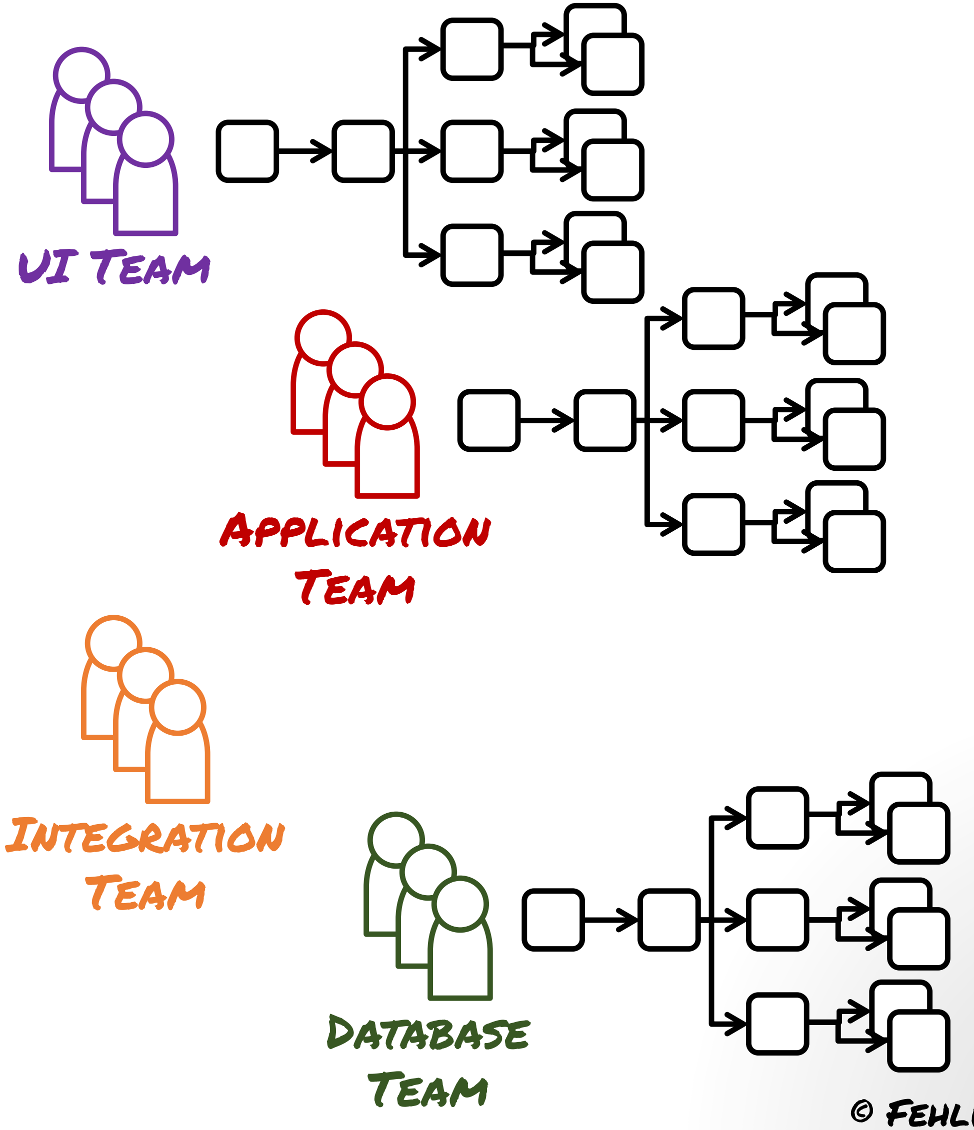


© Fehling 42

OK! Let's Automate our IT Stack!

But Wait...

- Who creates (which part of) the model?
- Which System interprets and executes the model?
- Who controls the automation System?
- What about Existing Ticket Systems?!
- What about existing manual tasks?



-> Automation Requires control over the Cloud to be centralized – not delegated!
-> Empowerment of DevOps Teams is needed to use Clouds efficiently!
-> Existing Teams may be afraid to loose control.

Lessons Learned

Revise procurement processes, because...

... suppliers using a cloud for you may create a vendor lock-in!

... many benefits of the cloud properties are lost if a supplier uses a cloud for you!

Demand Cloud-compatible licenses, because...

... Costs per instance conflicts with architectural goals!

Revise your organizational hierarchies, because...

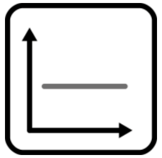
... clouds thrive on automation and require fewer delegation of manual tasks!

... self-service interfaces are more agile than ticketing systems!

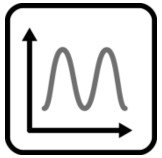
Summary

Part 1: Cloud Computing Patterns @ Mercedes Me

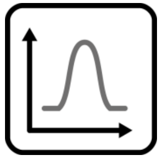
Microservice Template based on Cloud Computing Patterns and Pivotal Cloud Foundry



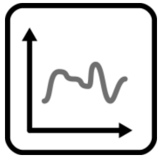
Static Workload



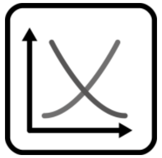
Periodic Workload



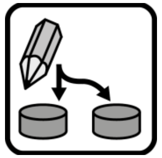
Once-in-a-lifetime Workload



Unpredictable Workload

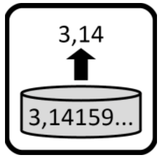


Continuously Changing Workload

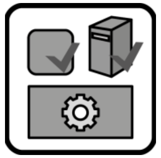


Strict Consistency

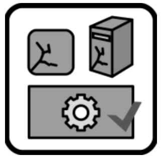
Eventual Consistency



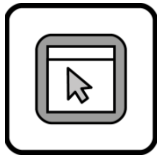
Data Abstractor



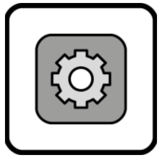
Node-based Availability



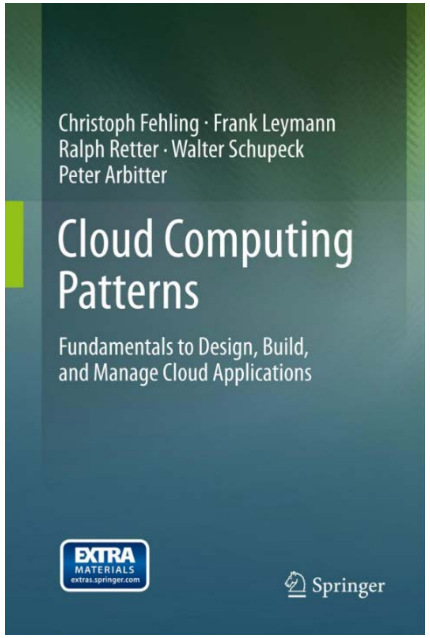
Environment-based Availability



User Interface Component



Processing Component



Part 1: Cloud Computing Patterns @ Mercedes Me

Microservice Template based on Cloud Computing Patterns and Pivotal Cloud Foundry

Part 2: The Non-technical "Stuff"…

PROCUREMENT PROCESSES HAVE TO BE ADJUSTED FOR CLOUD COMPUTING

- HOW CAN WE BUY ENVIRONMENTS FOR AGILE CLOUD DEVELOPMENT?
- HOW CAN WE BENEFIT FROM CLOUD PROPERTIES – NOT OUR SUPPLIERS?

ORGANIZATION HIERARCHIES HAVE TO BE ADJUSTED FOR CLOUD COMPUTING

- HOW CAN WE BENEFIT FROM CLOUD AUTOMATION?
- HOW CAN WE ORGANIZE WORK WITHOUT TICKETING SYSTEMS FOR MANUAL TASKS?

Part 3: Discussion during OOP ☺

I'm here all week! Contact me: fehling.c@gmail.com +49 170 58 35 456 @ccpatterns

http://www.cloudcomputingpatterns.org

BACKUP

Design Steps for Cloud Applications using Patterns

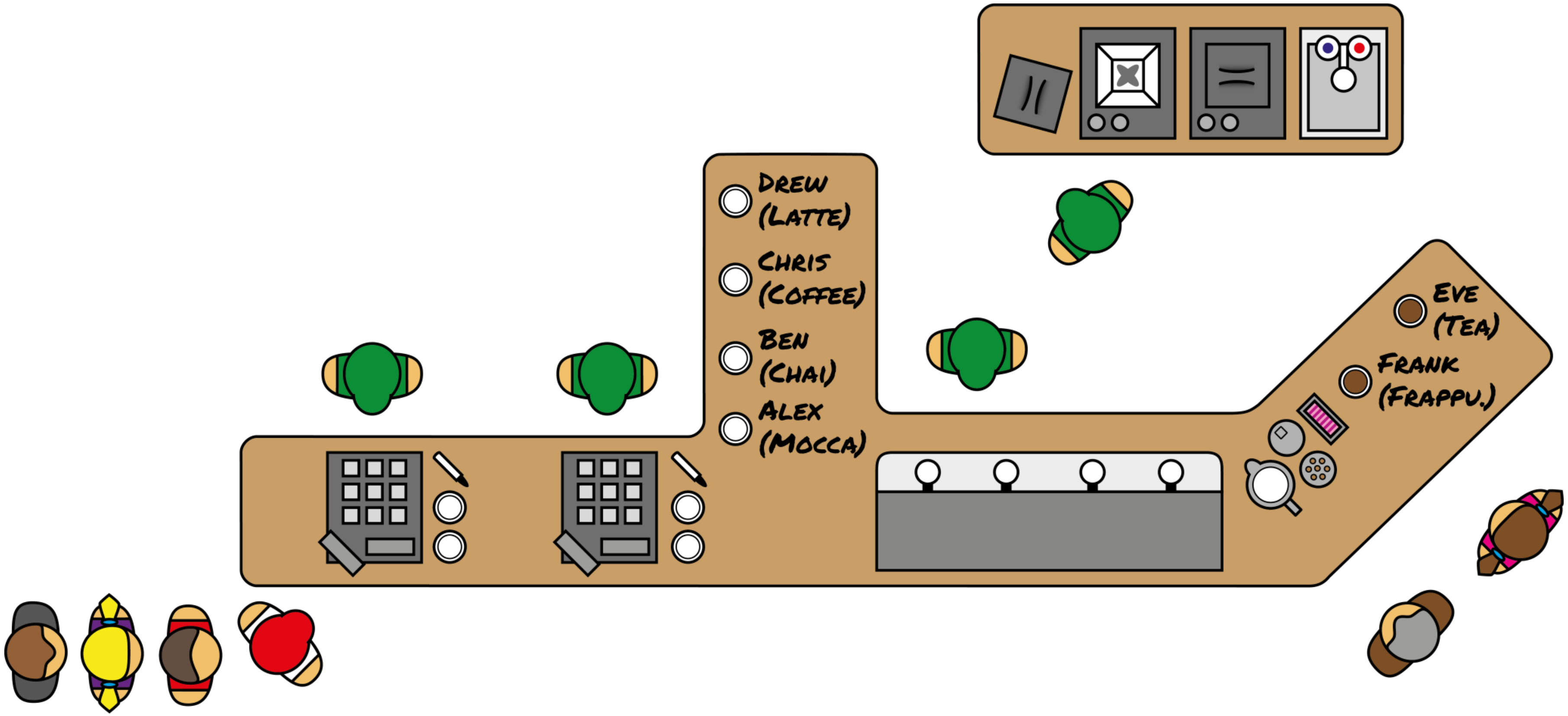
or

to see a Cloud Application Architecture you should go out and have a…

http://benedik.deviantart.com/

Design Steps for Cloud Applications

Decomposition → Workload → Data (State) → Component Refinement → Elasticity and Resiliency



IDEAL

Cloud-Native Application

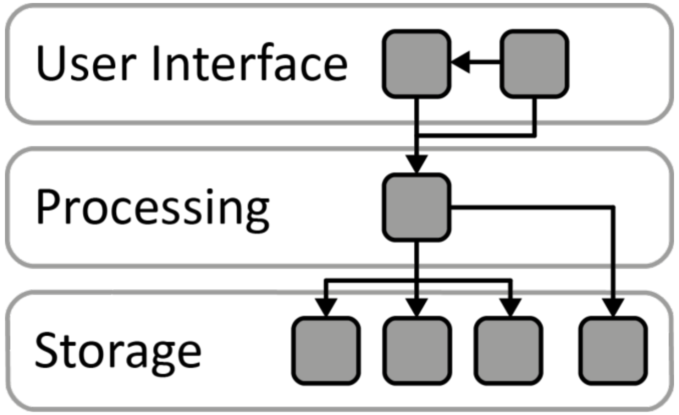
- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling

Decomposition → Workload → Data (State) → Component Refinement → Elasticity and Resiliency

How to distribute Application Functionality?

Distributed Application
A cloud application **divides provided functionality** among multiple application components that can be **scaled out independently**.

Layer-based Decomposition

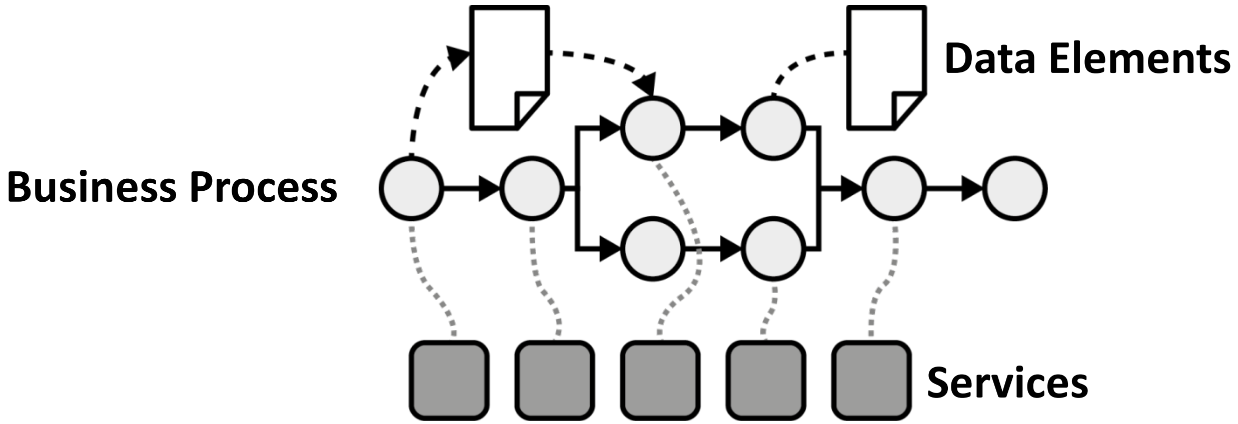


Components reside on separate functional layers
  Often: user interface, processing, storage
  Access is only allowed to **same layer and the layer below**
  →Dependencies between layers and interfaces are controlled

IDEAL
Cloud-Native Application
- [x] Isolated State
- [x] Distribution
- [ ] Elasticity
- [ ] Automated Management
- [ ] Loose Coupling

Distributed Application
A cloud application **divides provided functionality** among multiple application components that can be **scaled out independently**.

Process-based Decomposition



Business process model determines decomposition

- **Activities**: tasks executed in a specific order (**control flow**)
- **Data elements**: information handled by activities (**data flow**)
- Functional application components (**services**) are accessed by process

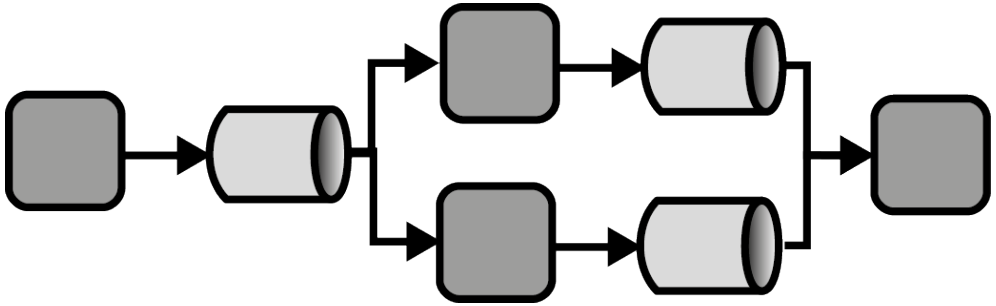
IDEAL
Cloud-Native Application

- ✔ Isolated State
- ✔ Distribution
- ○ Elasticity
- ○ Automated Management
- ○ Loose Coupling

Distributed Application

A cloud application **divides provided functionality** among multiple application components that can be **scaled out independently**.

Pipes-and-Filters-based Decomposition



Decomposition based on the data processing function

- **Filter**: application component processing data
- **Pipe**: connection between filters (commonly messaging)

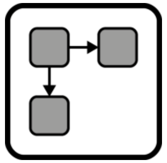
IDEAL

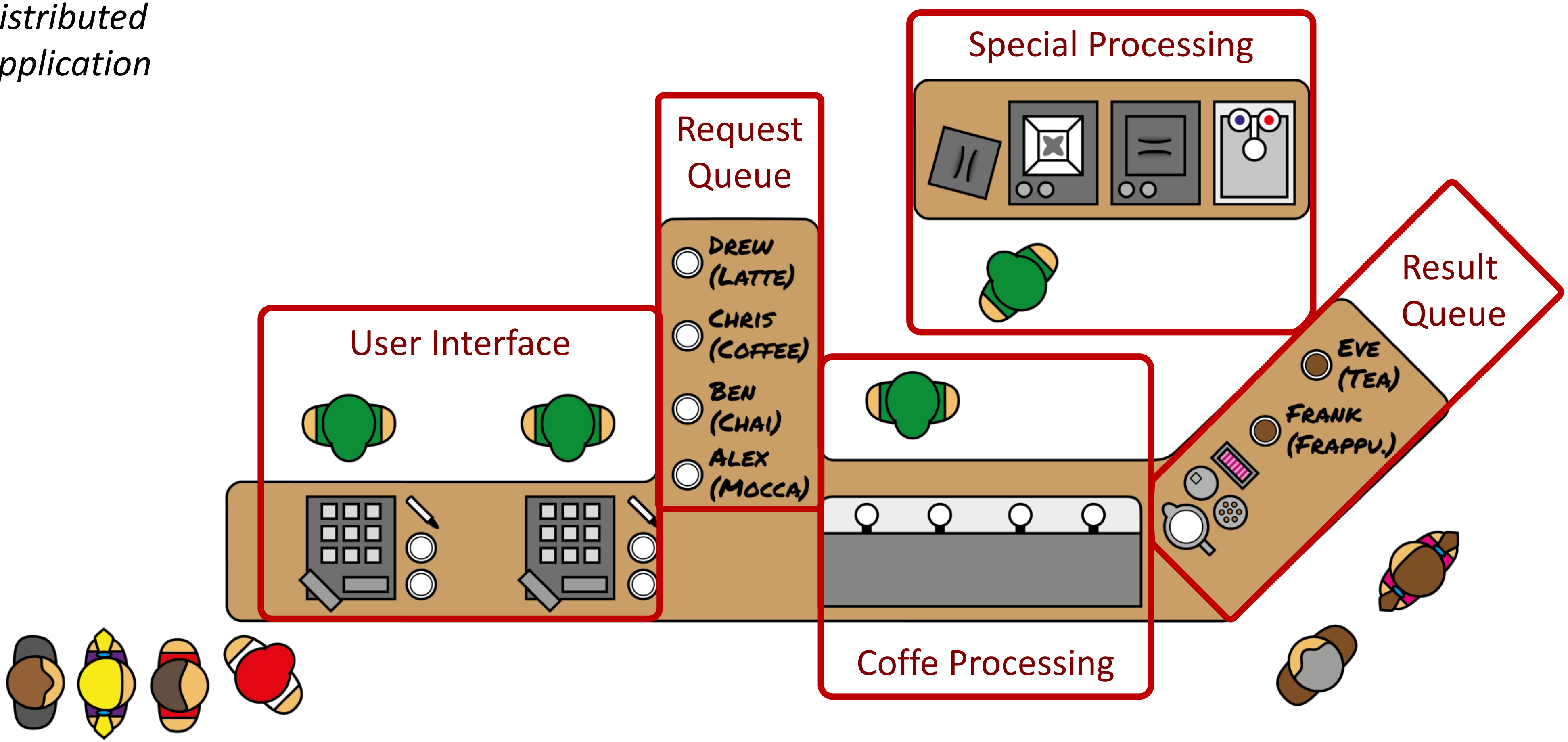
Cloud-Native Application

- ☑ Isolated State
- ☑ Distribution
- ○ Elasticity
- ○ Automated Management
- ○ Loose Coupling

Coffe Shop – Decomposition of Functions

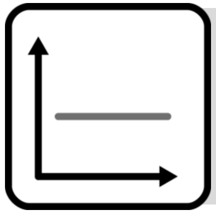
Identify functional components.


Distributed Application



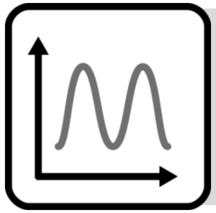
Decomposition → Workload → Data (State) → Component Refinement → Elasticity and Resiliency

What workload do components experience?



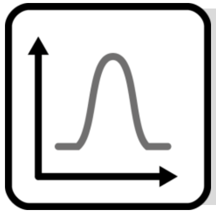
Static Workload

IT resources with an equal utilization over time experience static workload.



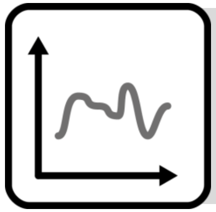
Periodic Workload

IT resources with a peaking utilization at reoccurring time intervals experience periodic workload.



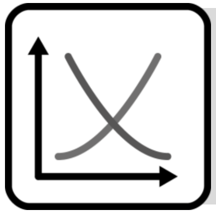
Once-in-a-Lifetime Workload

IT resources with an equal utilization over time disturbed by a strong peak occurring only once experience once-in-a-lifetime workload.



Unpredictable Workload

IT resources with a random and unforeseeable utilization over time experience unpredictable workload.



Continuously Changing Workload

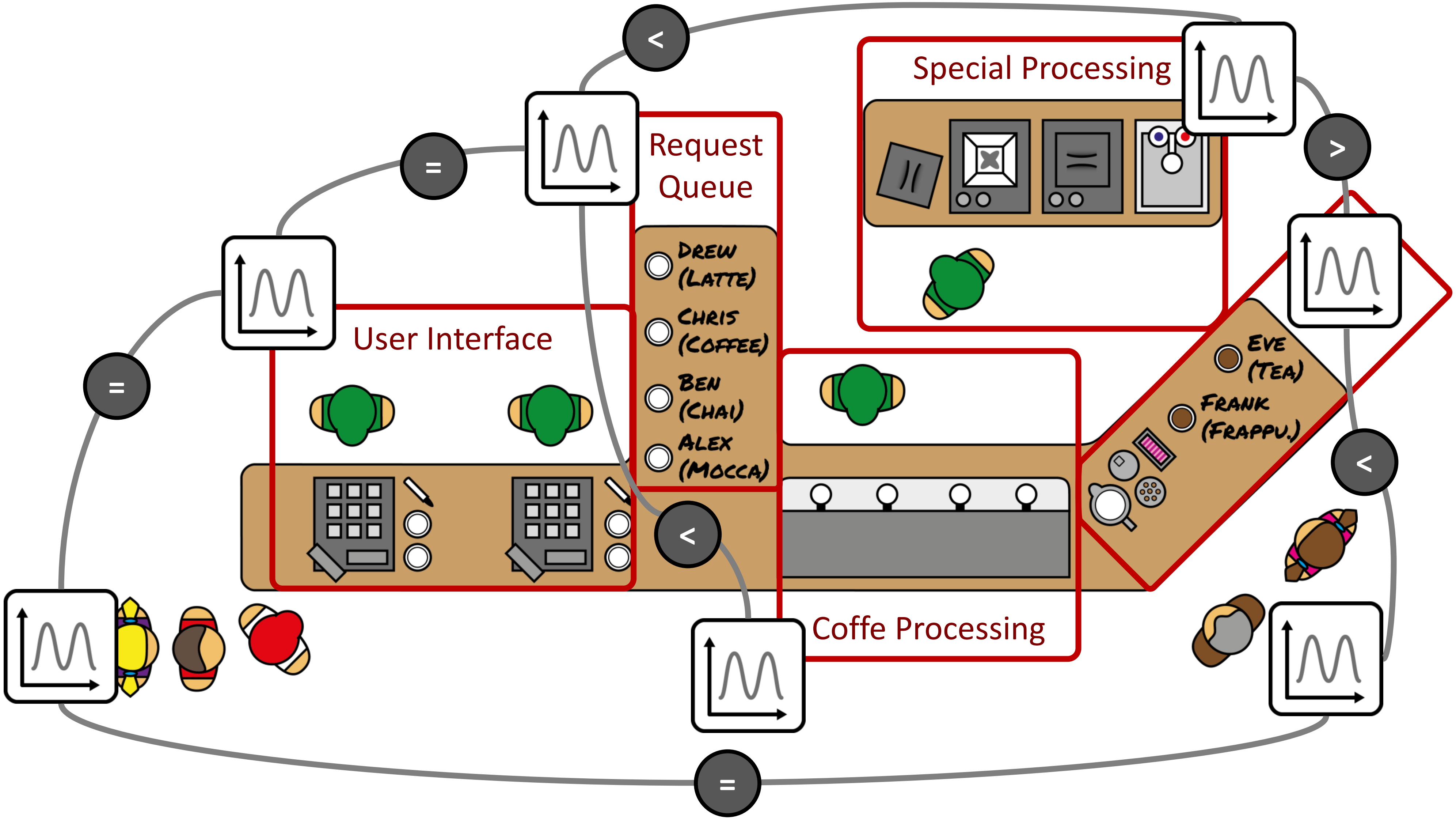
IT resources with a utilization that grows or shrinks constantly over time experience continuously changing workload.

IDEAL
Cloud-Native Application

- [ ] Isolated State
- [ ] Distribution
- [x] Elasticity
- [x] Automated Management
- [ ] Loose Coupling

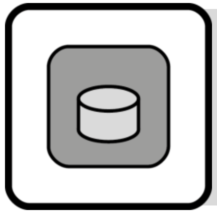
Coffe Shop – Workloads

Identify and compare workload generated by user groups at different components.

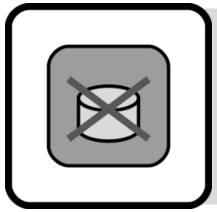


Decomposition | Workload | Data (State) | Component Refinement | Elasticity and Resiliency

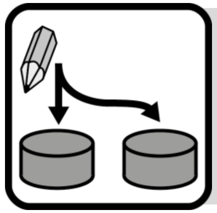
Where does the application handle state?



Stateful Component
Multiple instances of a scaled-out application component synchronize their internal state to provide a unified behavior.

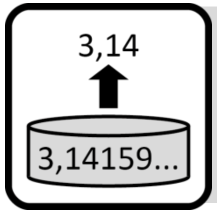


Stateless Component
State is handled external of application components to ease their scaling-out and to make the application more tolerant to component failures.



Strict Consistency
Data is stored at different locations to improve response time and to avoid data loss in case of failures while consistency of replicas is ensured at all times.

Eventual Consistency
Performance and availability of data in case of network partitioning are enabled by ensuring data consistency eventually and not at all times.



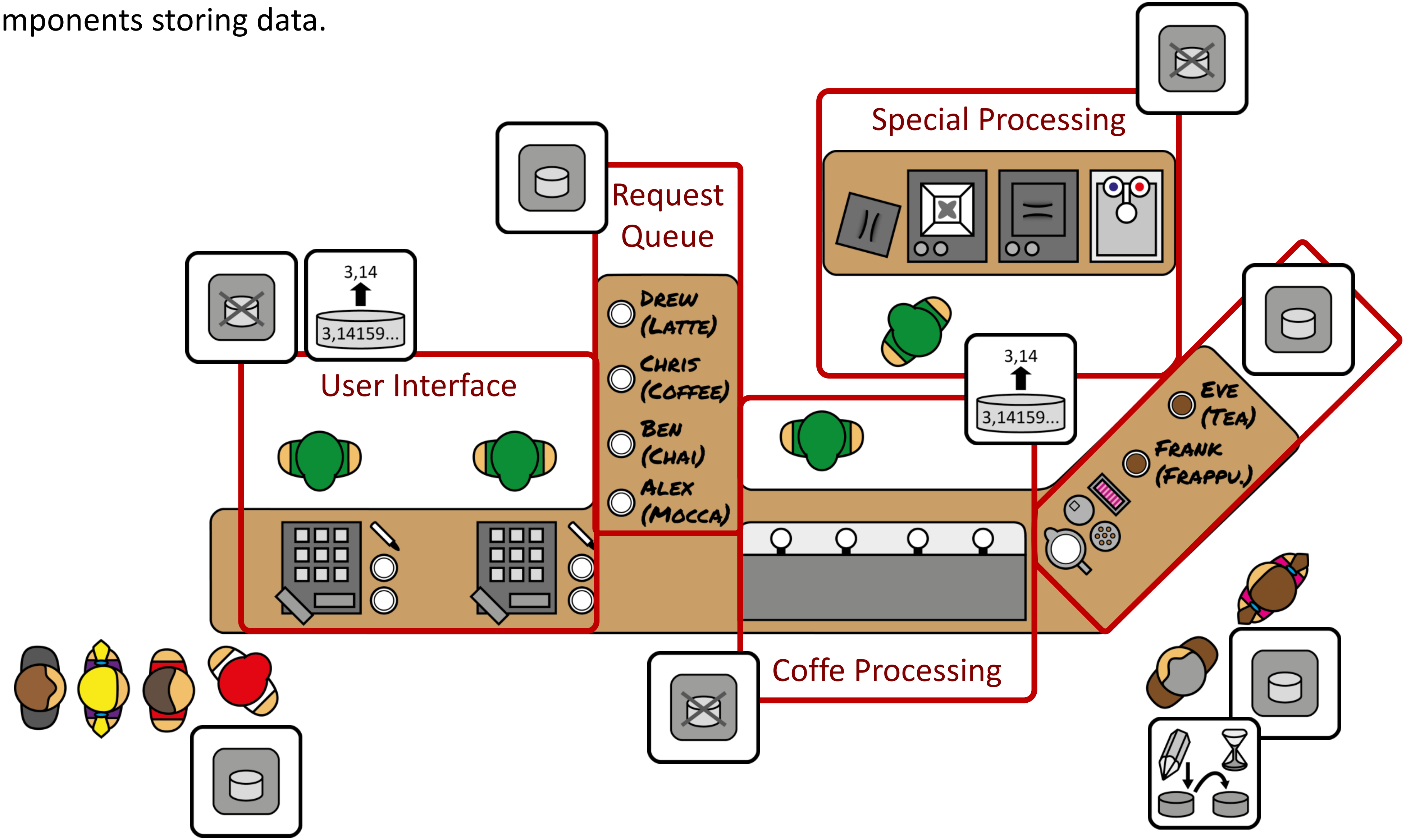
Data Abstractor
Data is abstracted to inherently support eventually consistent data storage through the use of abstractions and approximations.

IDEAL
Cloud-Native Application

- ✔ Isolated State
- Distribution
- Elasticity
- ✔ Automated Management
- Loose Coupling

Coffee Shop – Data

Identify components storing data.



Decomposition → Workload → Data (State) → Component Refinement → Elasticity and Resiliency

How are components implemented?

Message-oriented Middleware

Asynchronous communication is provided while hiding complexity of addressing, routing, or data formats to make interaction robust and flexible.

User Interface Component

Synchronous user interfaces are accessed by humans, while application-internal interaction is realized asynchronously to ensure loose coupling.

Processing Component

Processing functionality is handled by elastically scaled components.

Batch Processing Component

Requests are delayed until environmental conditions make their processing feasible.

Multi-component Image

Virtual servers host multiple application components that may not be active at all times to reduce provisioning and decommissioning operations.

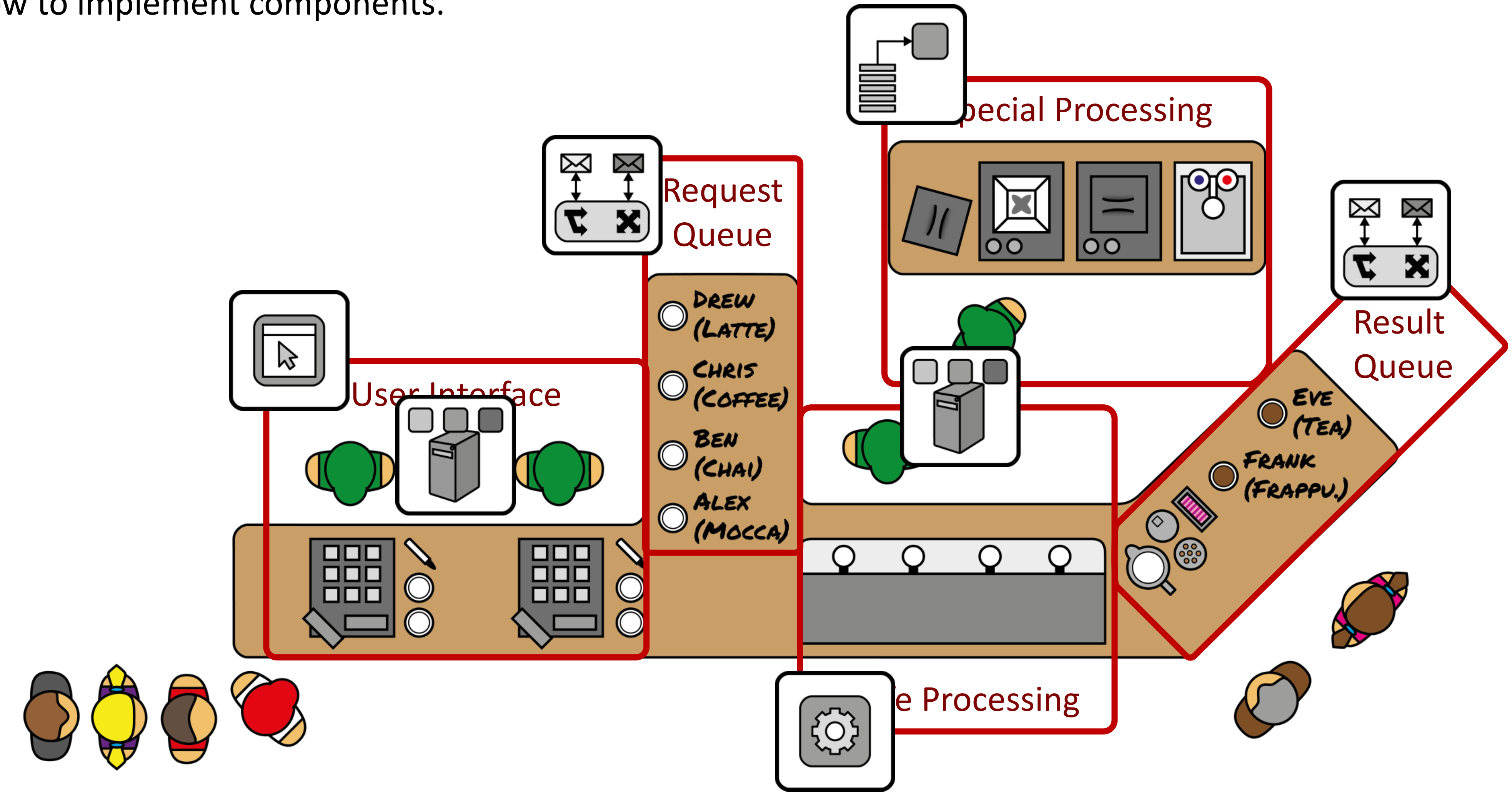
IDEAL

Cloud-Native Application

- [x] Isolated State
- [ ] Distribution
- [ ] Elasticity
- [ ] Automated Management
- [x] Loose Coupling

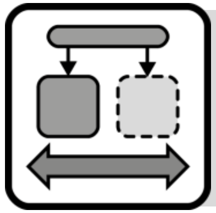
Coffee Shop – Refinement of Components

Decide how to implement components.

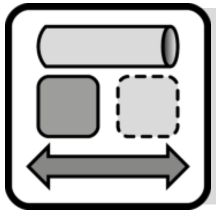


Decomposition | Workload | Data (State) | Component Refinement | Elasticity and Resiliency

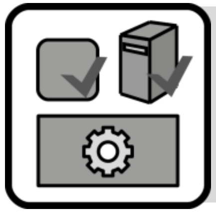
Elasticity and Resiliency



Elastic Load Balancer
The number of synchronous accesses to an elastically scaled-out application is used to determine the number of required application component instances.



Elastic Queue
The number of accesses via messaging is used to adjust the number of required application component instances.



Node-based Availability
A cloud provider guarantees the availability of nodes, such as individual virtual servers, middleware components or hosted application components.

Environment-based Availability
A cloud provider guarantees the availability of the environment hosting individual nodes, such as virtual servers or hosted application components.

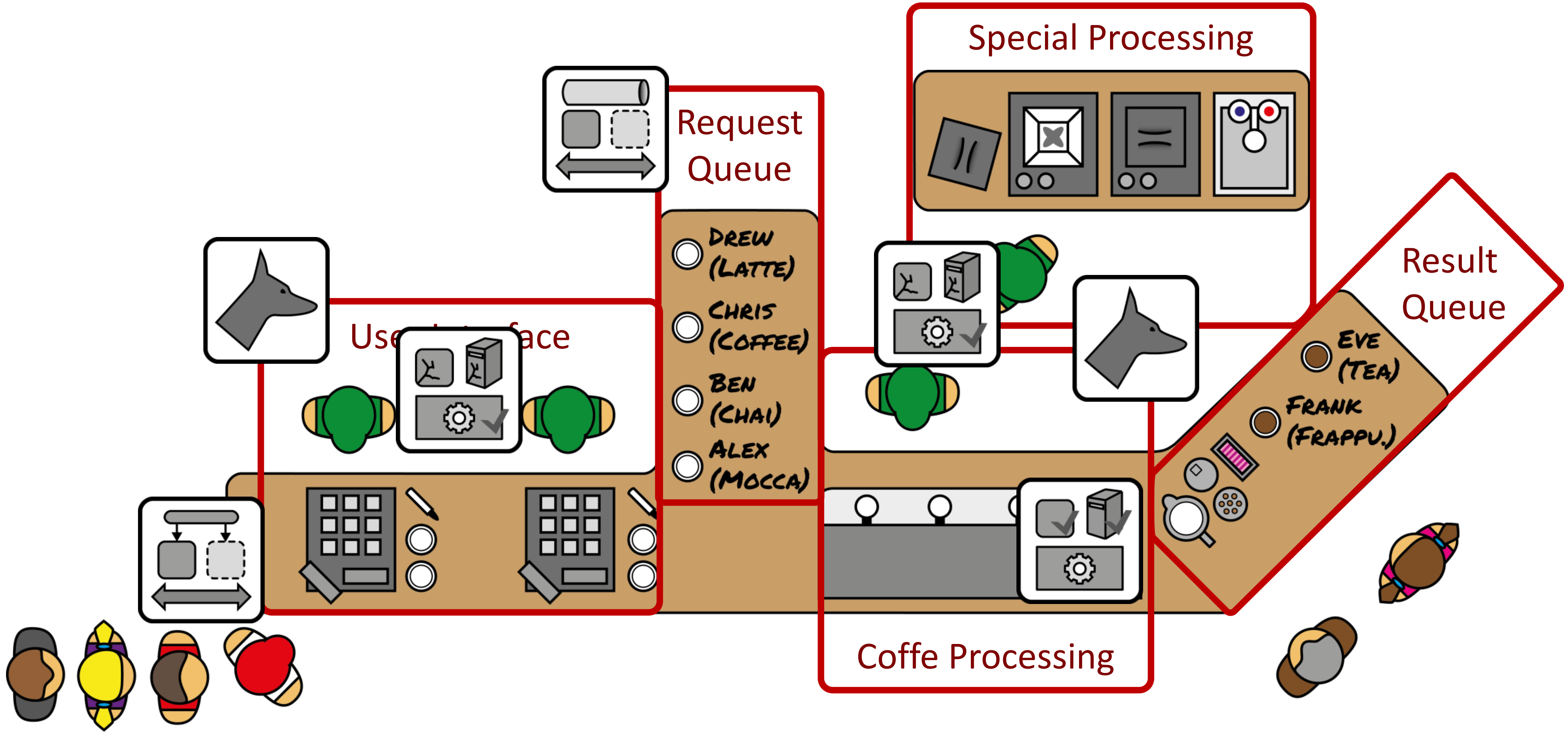
Watchdog
Applications cope with failures by monitoring and replacing application component instances if the provider-assured availability is insufficient.

IDEAL
Cloud-Native Application

- ☐ Isolated State
- ☐ Distribution
- ☑ Elasticity
- ☑ Automated Management
- ☐ Loose Coupling

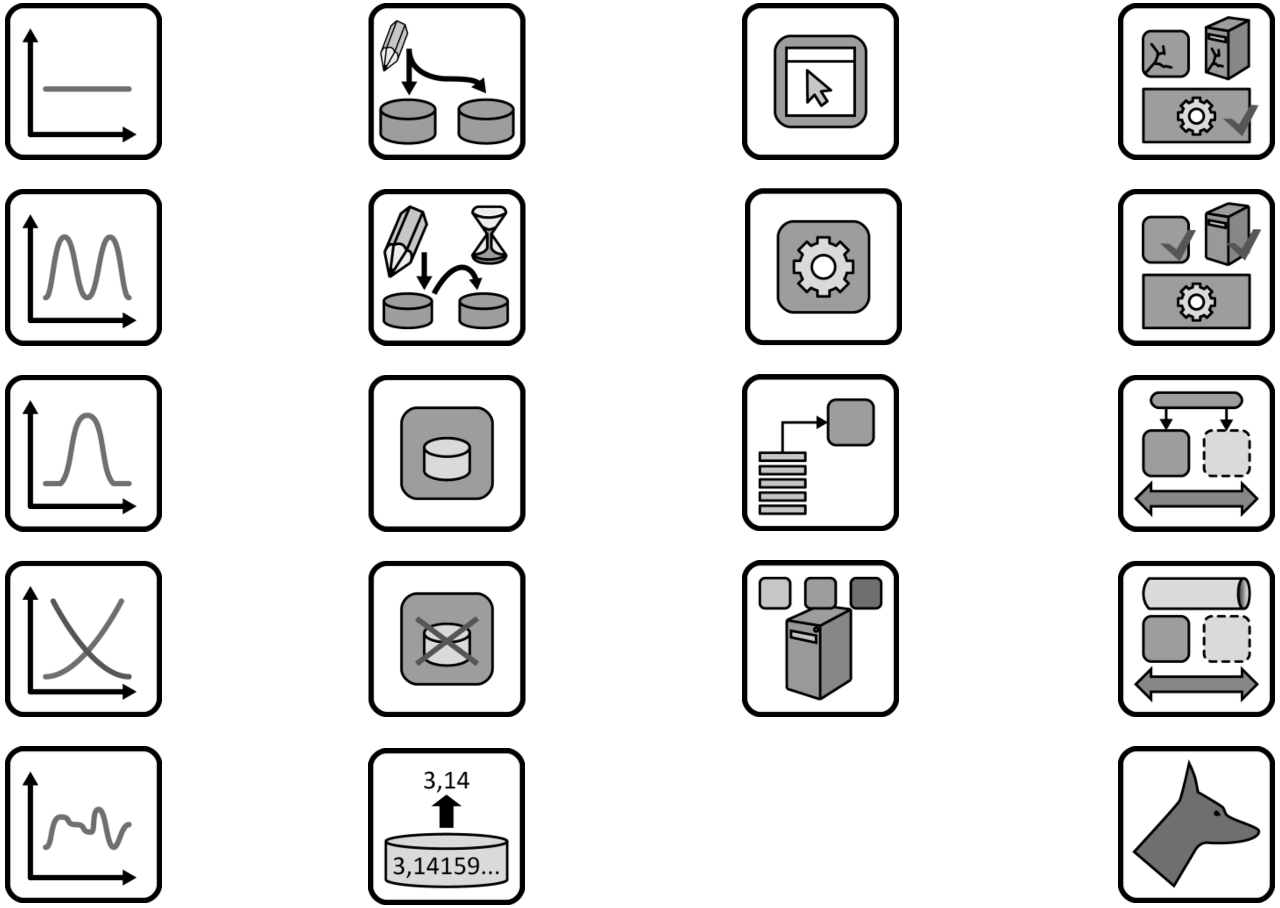
Elasticity and Resiliency

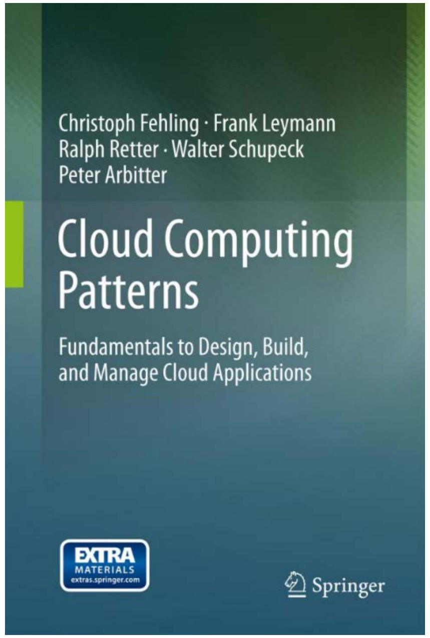
What shall happen if workload changes or something fails?



Design Steps for Cloud Applications using Patterns

Decomposition → Workload → Data (State) → Component Refinement → Elasticity and Resiliency





IDEAL

Cloud-Native Application

- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling

The End