CMPE 282 Cloud Services
Cloud-Native Application Design Patterns

Instructor: Kong Li

Content

- What and why
- Coffee Shop
  - Decomposition
  - Workload
  - Data/state
  - Component refinement
  - Elasticity and Resiliency
- Composite Cloud Applications

1

2

3

Design Patterns

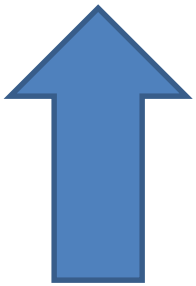
- What
  - Reusable solutions to problems
  - Domain independent
  - Not a cookbook, but a guide
  - Not a finished solution
- Why
  - Makes the intent of code easier to understand
  - Provides a common language for solutions
  - Be able to reuse code
  - Known performance profiles and limitations of solutions

Why App Design Patterns

- Simply porting existing app to the cloud does not leverage from cloud features
- Recurring patterns in cloud services/apps
- Lots of new cloud services/apps every day
- Community is reaching the right level of maturity
- Patterns allow us
  - Evaluation of cloud offerings
  - Building apps on top of cloud offerings
  - Building custom cloud offerings
  - Evaluation of app landscapes for cloud-readiness

We need to do Cloud!

- Typical resulting questions
  - Which cloud infrastructure (provider) is the right one for our enterprise?
  - Is this app suitable for the cloud?
  - Why isn't it as easy to deploy an app in our data center as it is to deploy a sample app in my favorite public cloud?
- What happens next:
  - Business Process
  - Application
  - Platform
  - **Infrastructure**



Bottom up approach

- Typical results of bottom up:
  - App can work but it becomes the bottleneck
  - App may not or cannot leverage from cloud features
  - Low cloud resource util%
  - Poor performance

We need to do Cloud – What's wrong?

- What we really want
  - Cloud native apps
  - App leverages from cloud features
  - High cloud resource util%
  - Better performance
- **Real Requirements != We need to do Cloud!**
- Example requirements:
  - Deal with dynamic load patterns w/o provisioning for peak-load

    → save money!
  - Make app deployments easier and faster

    → save time through standardization!

We need to do Cloud – The Proper Way

- Better questions (requirements driven)
  - Which of my business processes benefit from cloud properties of underlying apps?
    - Dynamic load patterns
    - High availability
    - Pay-per-use
    - Self service required?
  - Which apps drive these business processes and can they deal with:
    - Resource sharing/ pooling
    - Elasticity as a result of requirement for dynamic load patterns and pay per use?
  - What (cloud) infrastructure and platforms are needed to support these apps?
- What happens next:
  - **Business Process**
  - Application
  - Platform
  - Infrastructure

Top down approach

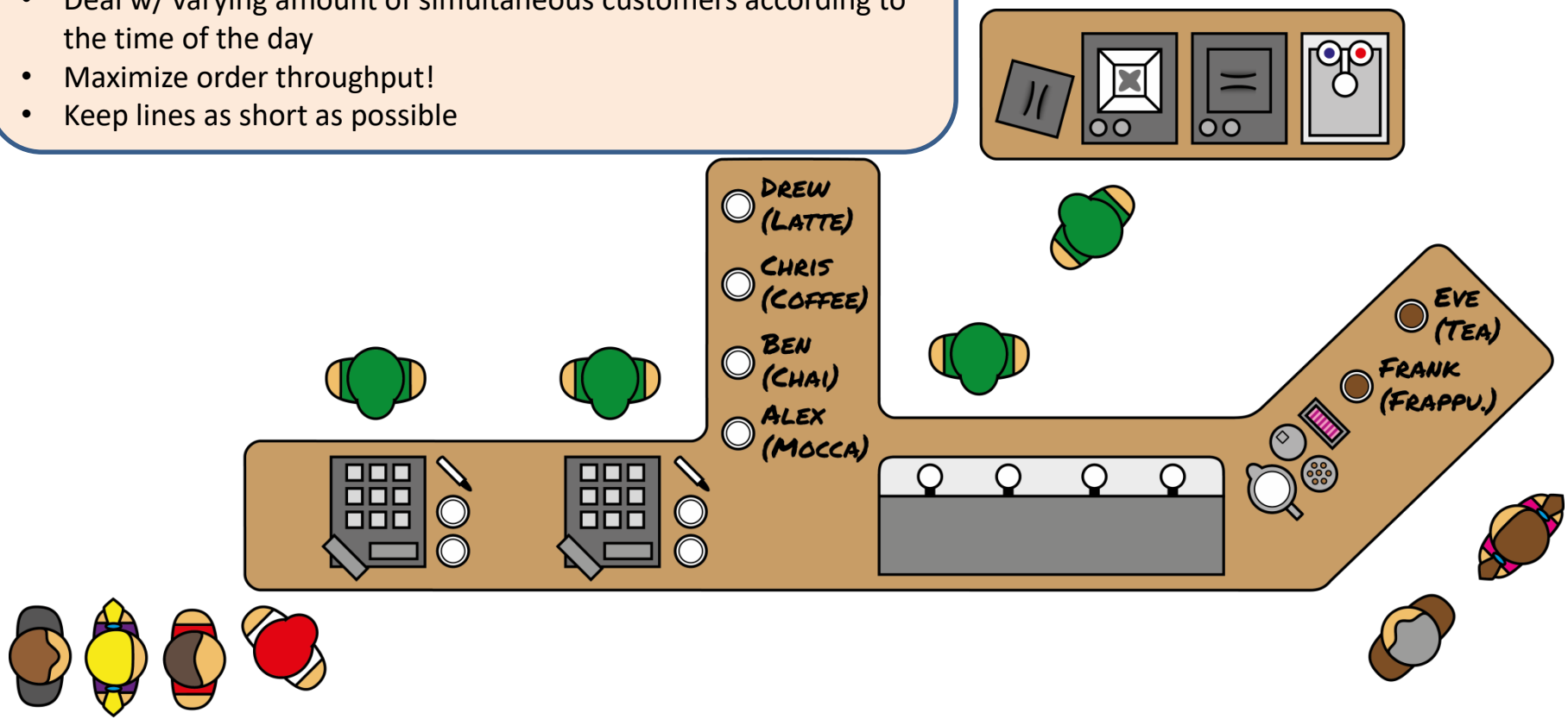
Cloud-Based Coffee Shop

Functional requirements

- Make coffee-related specialities; sell them to walk-in customers

Non-functional requirements

- Deal w/ varying amount of simultaneous customers according to the time of the day
- Maximize order throughput!
- Keep lines as short as possible



Inspired by: G. Hohpe: Your Coffee Shop Doesn't Use Two-Phase Commit, IEEE Software, 2005.

Coffee Shop App Design

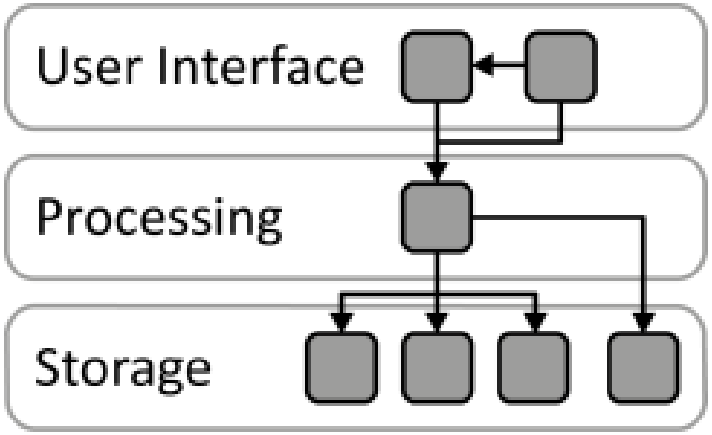
- Decomposition: How to distribute Application Functionality?
  - Distributed App
- Work load: What workload do components experience?
  - Static
  - Periodic
  - Once-in-a-lifetime
  - Unpredictable
  - Continuously Changing
- Data (State): Where does the application handle state?
  - Stateful
  - Stateless
  - Strict consistency
  - Eventual consistency
  - Data Abstractor
- Component Refinements: How are components implemented?
  - Message-oriented Middleware
  - User Interface Component
  - Processing Component
  - Batch Processing Component
  - Multi-component Image
- Elasticity and Resiliency
  - Elastic Load Balancer
  - Elastic Queue
  - Node-based Availability
  - Environment-based Availability
  - Watchdog

Coffee Shop App Design (1)

- **Decomposition: How to distribute Application Functionality?**
  - **Distributed App**
- Work load: What workload do components experience?
  - Static
  - Periodic
  - Once-in-a-lifetime
  - Unpredictable
  - Continuously Changing
- Data (State): Where does the application handle state?
  - Stateful
  - Stateless
  - Strict consistency
  - Eventual consistency
  - Data Abstractor
- Component Refinements: How are components implemented?
  - Message-oriented Middleware
  - User Interface Component
  - Processing Component
  - Batch Processing Component
  - Multi-component Image
- Elasticity and Resiliency
  - Elastic Load Balancer
  - Elastic Queue
  - Node-based Availability
  - Environment-based Availability
  - Watchdog

Distributed App - Layer

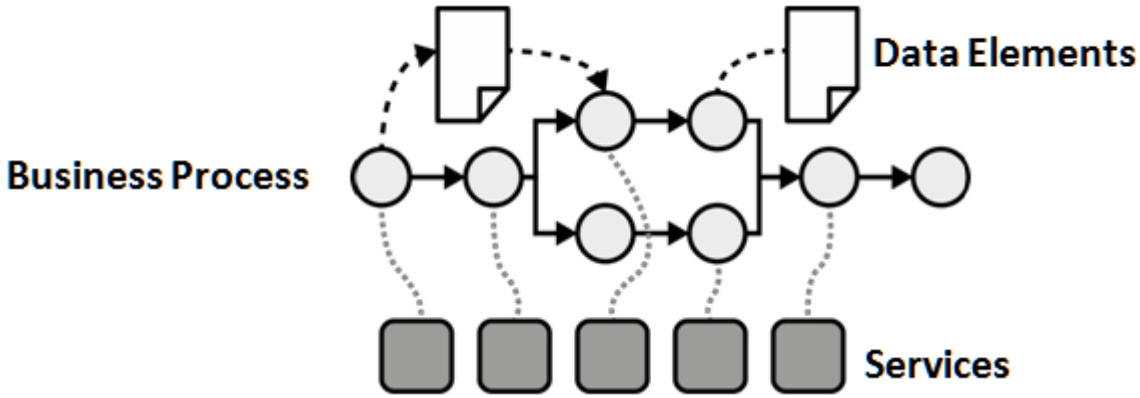
- Intent: A cloud app divides provided functionality among multiple app components that can be scaled out independently



- Solution: Layer-based decomposition - Components reside on separate functional layers
  - Often: UI, processing, storage
  - Access is only allowed to same layer and the layer below
    - Dependencies between layers and interfaces are controlled

Distributed App - Process

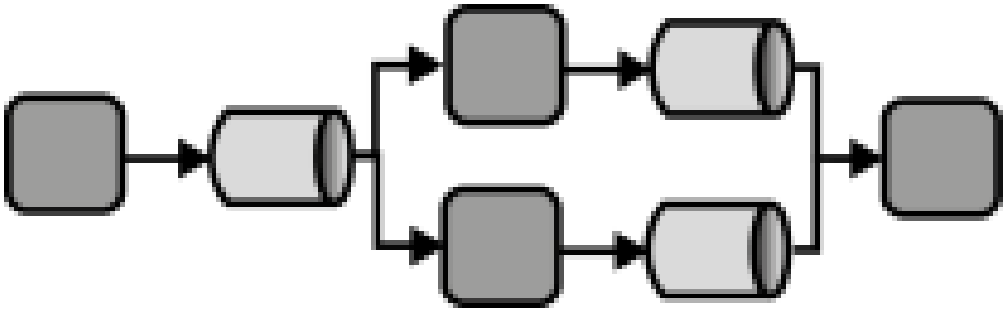
- Intent: A cloud app divides provided functionality among multiple app components that can be scaled out independently



- Solution: 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 app components (**services**) are accessed by process

Distributed App – Pipes/Filters

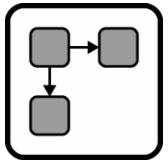
- Intent: A cloud app divides provided functionality among multiple app components that can be scaled out independently



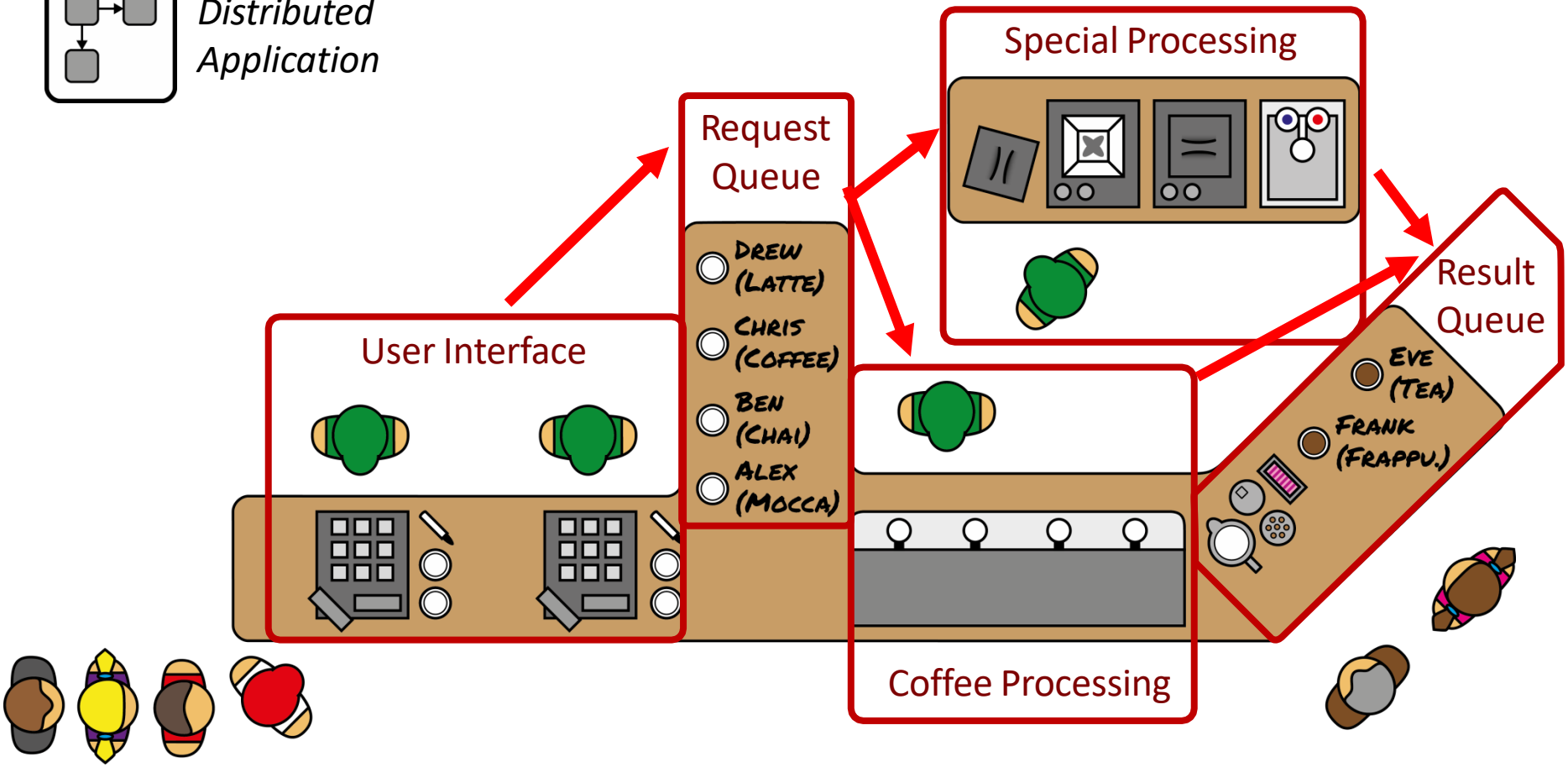
- Solution: Pipes-and-Filters-based Decomposition - Decomposition based on the data-centric processing function
  - **Filter**: application component processing data
  - **Pipe**: connection between filters (commonly messaging)

Coffee Shop – Decomposition of Functions

Identify functional components



Distributed Application

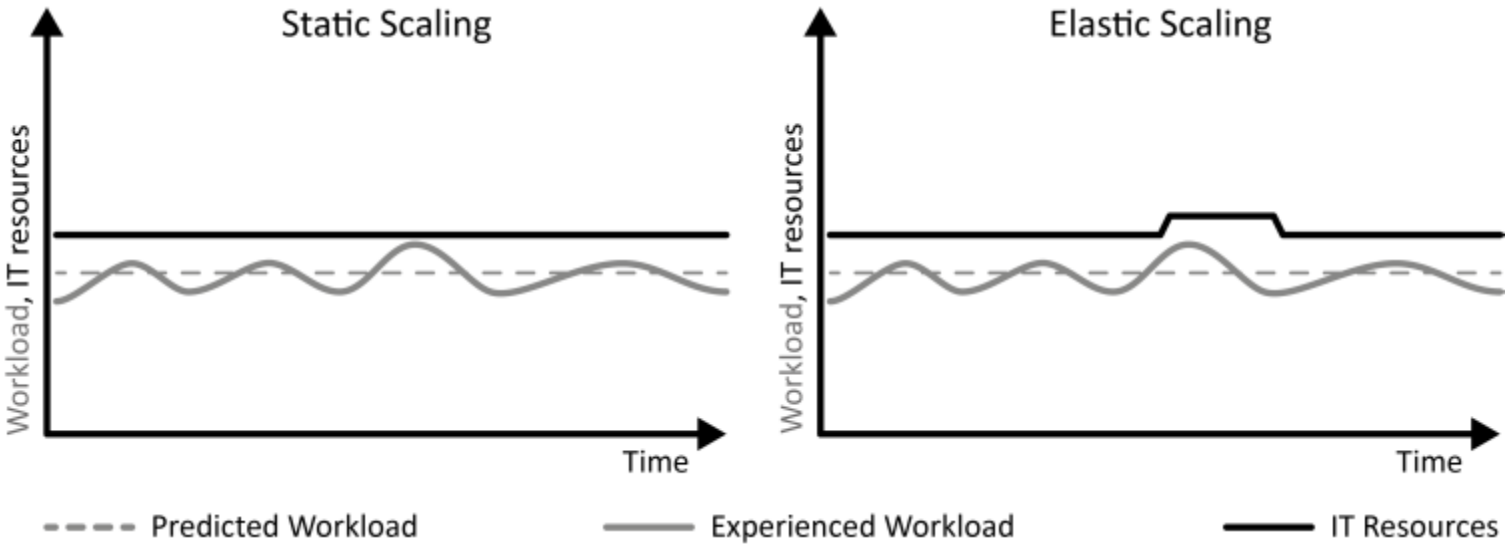


Coffee Shop App Design (2)

- Decomposition: How to distribute Application Functionality?
  - Distributed App
- **Work load: What workload do components experience?**
  - **Static**
  - **Periodic**
  - **Once-in-a-lifetime**
  - **Unpredictable**
  - **Continuously Changing**
- Data (State): Where does the application handle state?
  - Stateful
  - Stateless
  - Strict consistency
  - Eventual consistency
  - Data Abstractor
- Component Refinements: How are components implemented?
  - Message-oriented Middleware
  - User Interface Component
  - Processing Component
  - Batch Processing Component
  - Multi-component Image
- Elasticity and Resiliency
  - Elastic Load Balancer
  - Elastic Queue
  - Node-based Availability
  - Environment-based Availability
  - Watchdog

Static Workload

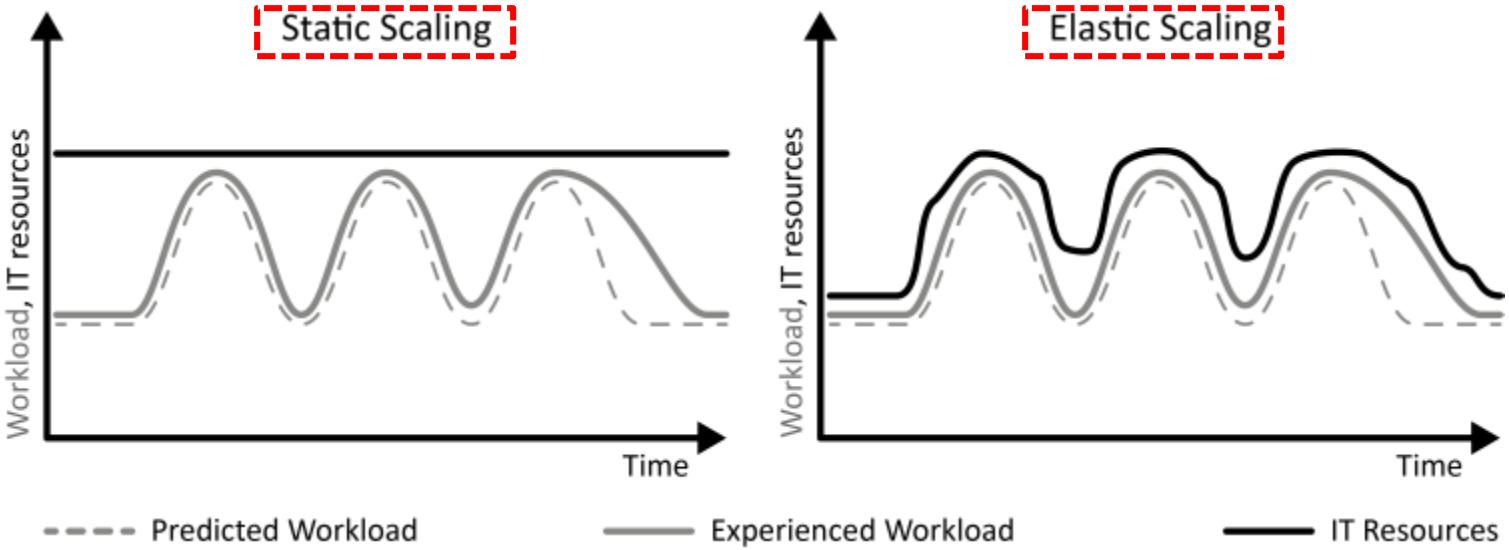
- Intent: IT resources with an equal utilization over time experience static workload



- Solution: less likely to benefit from an elastic cloud that offers a pay-per-use billing, because the number of required resources is constant
  - Resource pooling
  - Less benefits

Periodic Workload

- Intent: IT resources with a peaking utilization at reoccurring time intervals experience periodic workload

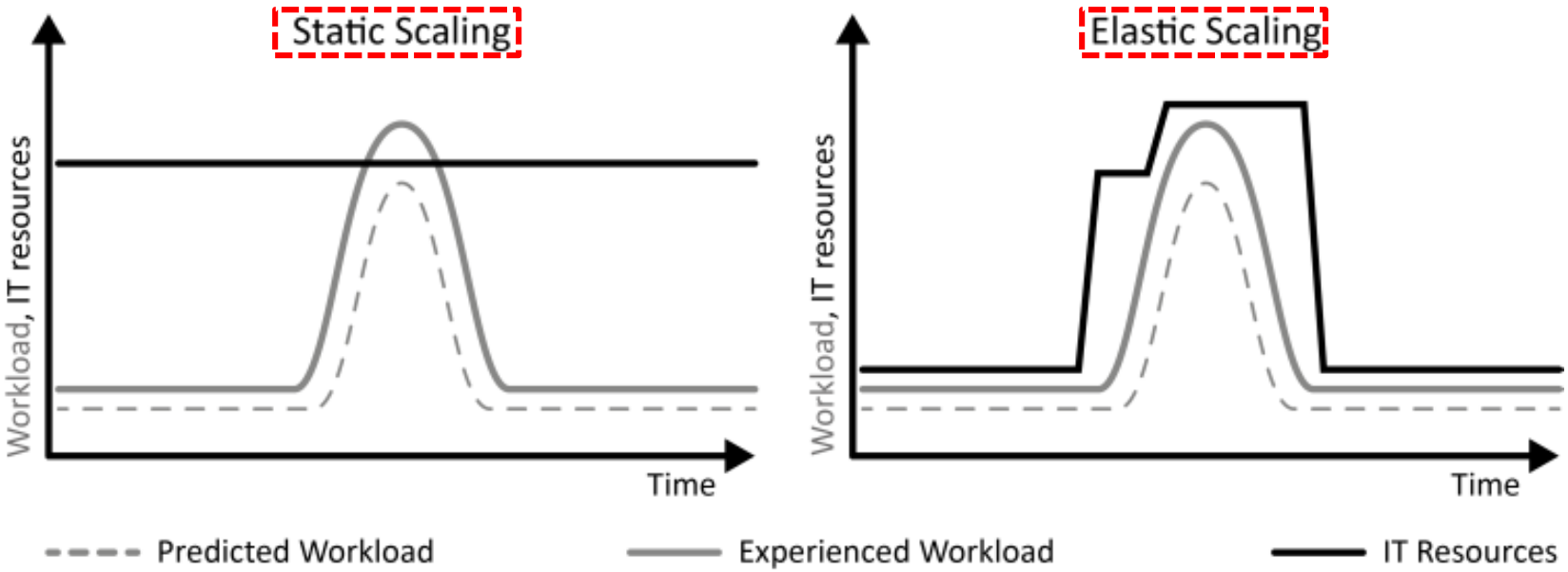


- Solution: From a customer & cost-saving perspective, use a provider with a pay-per-use pricing model
  - Only pay what you have used
  - Shift from CAPEX to OPEX

CAPEX: Capital Expenditures
OPEX: Operational Expenditure

Once-in-a-lifetime Workload

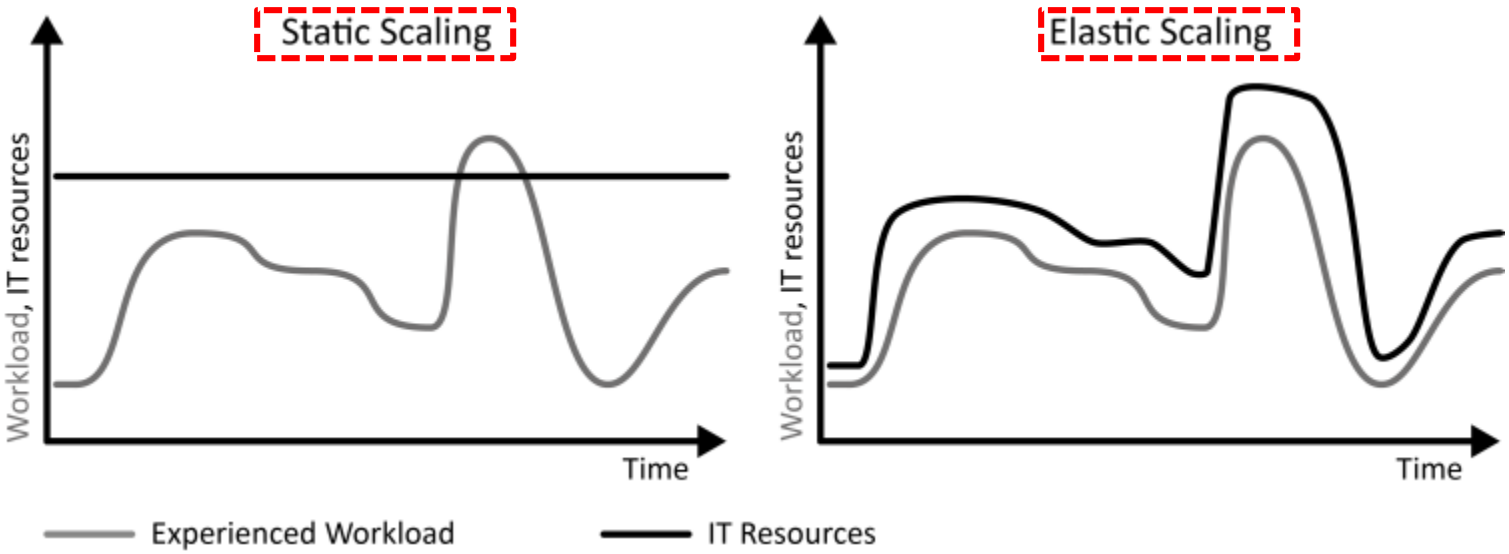
- Intent: IT resources with an equal utilization over time disturbed by a strong peak occurring only once experience once-in-a-lifetime workload



- Solution: manual task (at a known point in time) → elastic scaling of a cloud

Unpredictable Workload

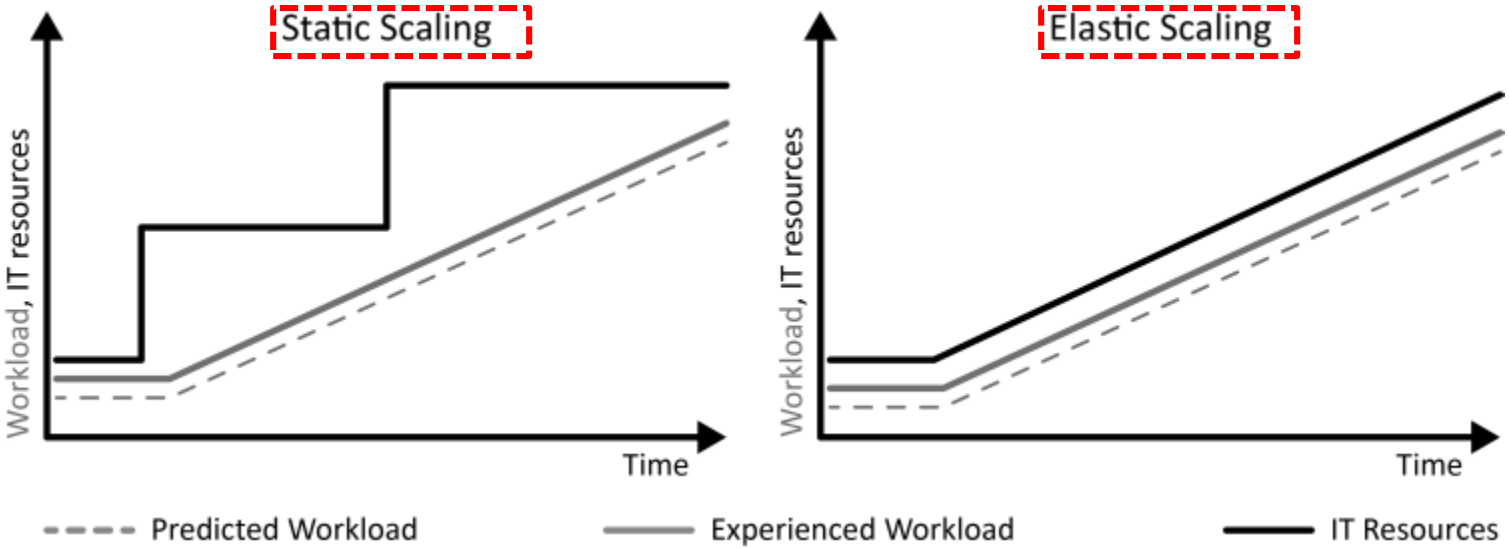
- Intent: IT resources with a random and unforeseeable utilization over time experience unpredictable workload



- Solution: automate unplanned provisioning and decommissioning of IT resources to align the resource numbers to changing workload

Continuously Changing Workload

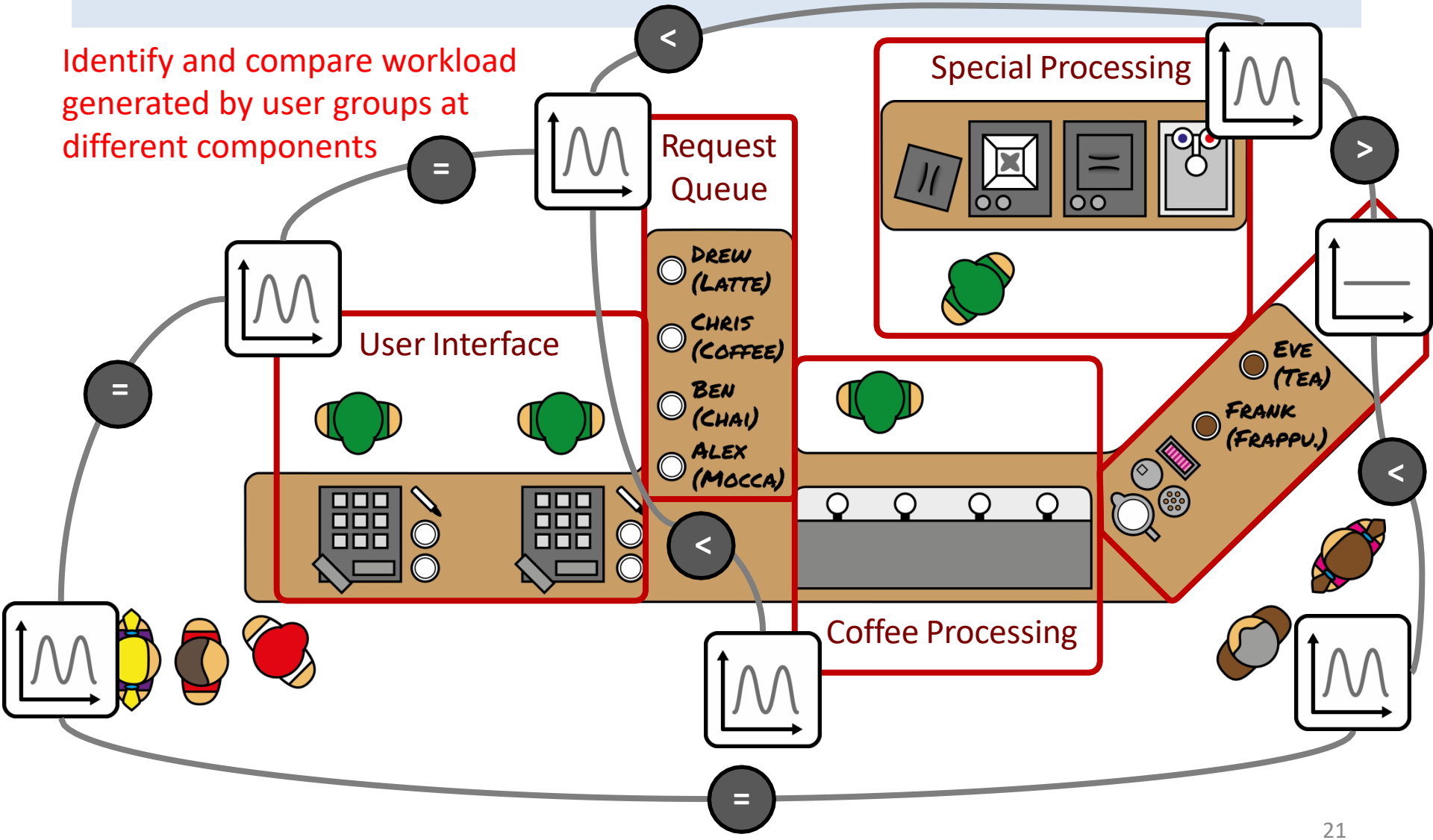
- Intent: IT resources with a utilization that grows or shrinks constantly over time experience continuously changing workload



- Solution: Elasticity of clouds

Coffee Shop – Workloads

Identify and compare workload generated by user groups at different components



Lessons - Workload

- Workload can differ **significantly** - Scaling them as a holistic unit can be very inefficient

References

- http://www.cloudcomputingpatterns.org/
- Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer; 2014
  - http://www.springer.com/978-3-7091-1567-1
- The coffee shop example is adapted from
  - https://indico.scc.kit.edu/indico/event/26/session/1/contribution/12/material/slides/0.pdf
  - http://www.sei.cmu.edu/library/assets/presentations/retter-saturn2013.pdf