Elasticity Management in the Cloud

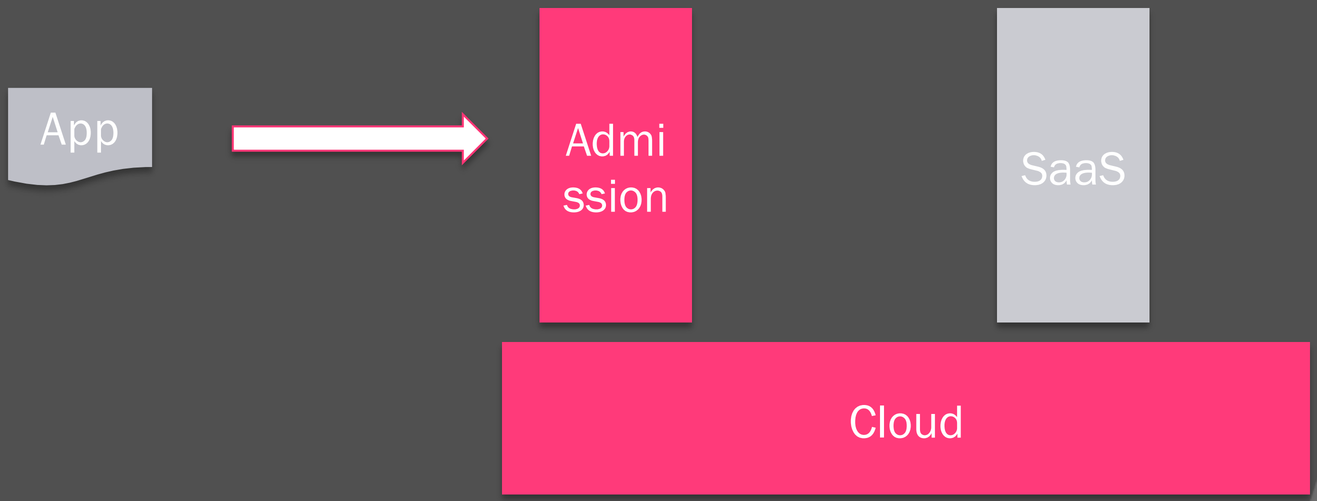
José M. Bernabéu-Aubán

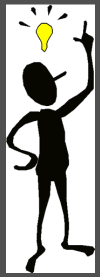
kumori systems

The Cloud and its goals

- Innovation of the Cloud
  - Utility model of computing
  - Pay as you go
  - Variabilize costs for everyone
    - But the Infrastructure providers
- Underlying theme
  - Optimize
  - Use only what you need
  - Do not overinvest

Ideal





Motivation

No developer left behind:

Bring all of them to the cloud

What is needed?

The Cloud

Problems SaaS providers need to solve

- Service guarantees
  - How to express them
- Service continuity
  - Promptly react to failures
    - Losing as little state as possible
  - Carefully perform software updates
- Maintain user's expected performance level
  - Promptly react to load changes
    - When possible, predict them

The Cloud

SaaS provider's concerns

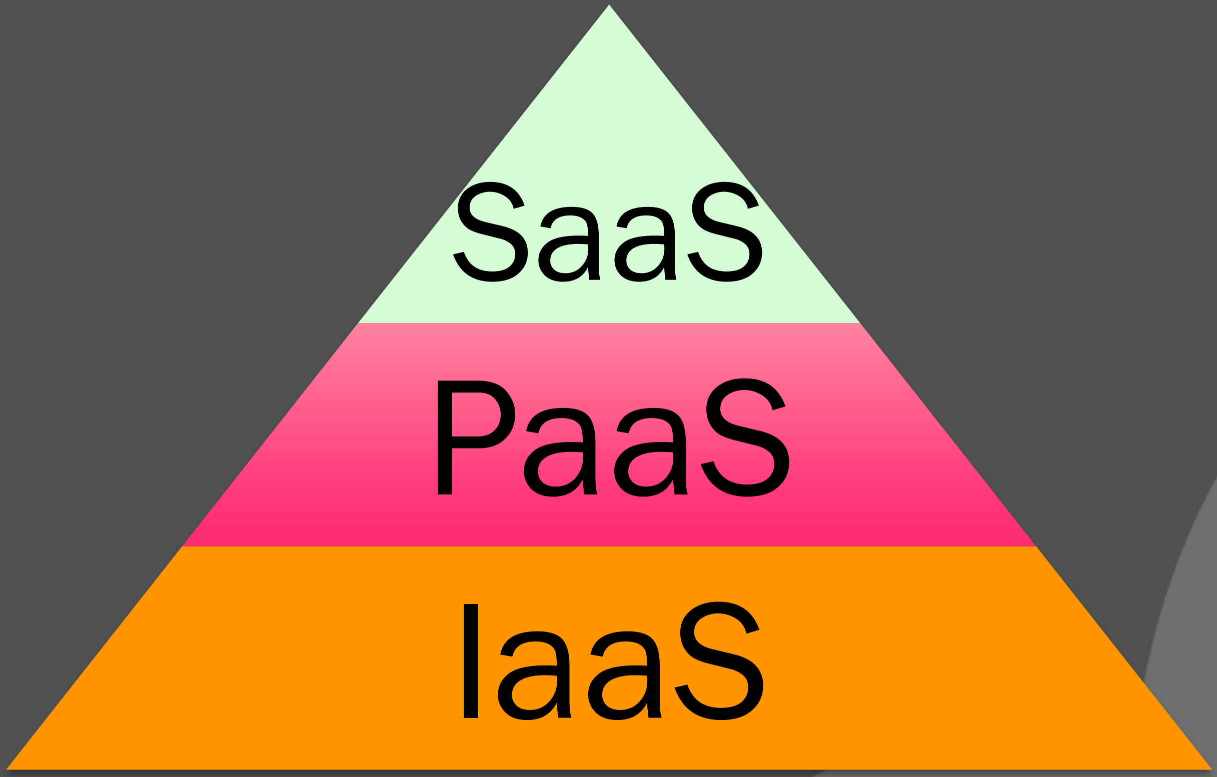
- The SLA: A contract
  - I'll do this if you do that
    - SLOs: Service Level Objectives
  - Penalties included
- Optimize benefits
  - Compute a penalty function out of an SLA: P_f
    - Factor-in costs of SLA violations
  - Avoid P_f increases
    - When failures, or updates
    - When load changes
    - While minimizing resources

The Cloud
Elasticity

- Elasticity:
  - – Degree to which a Service is managed optimally
- SaaS management should be elastic
- Difficult optimization problem
  - – Intrinsic difficulties
  - – Out of the reach for many developers
    - Especially those with, maybe, good business ideas
- Who provides Elasticity?

The Cloud

NIST layers



The Cloud

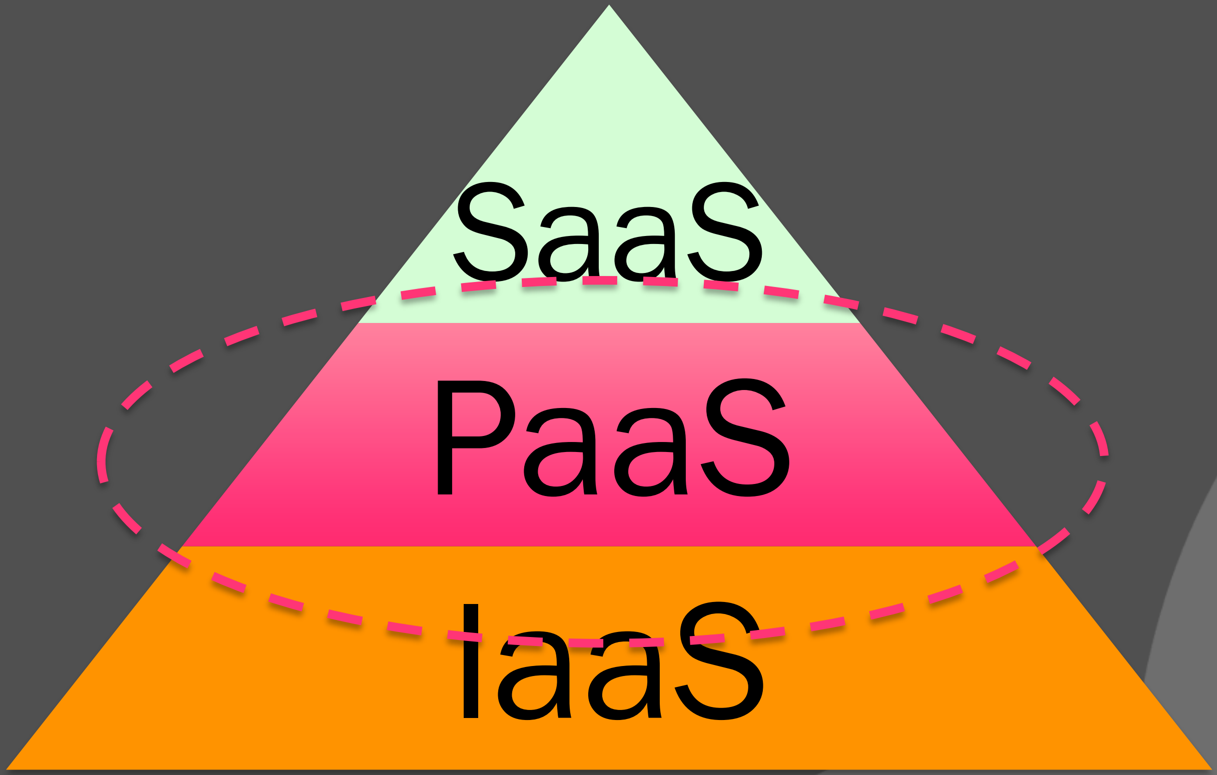
Standard Layers

- NIST definition of PaaS
- A layer where …

  the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

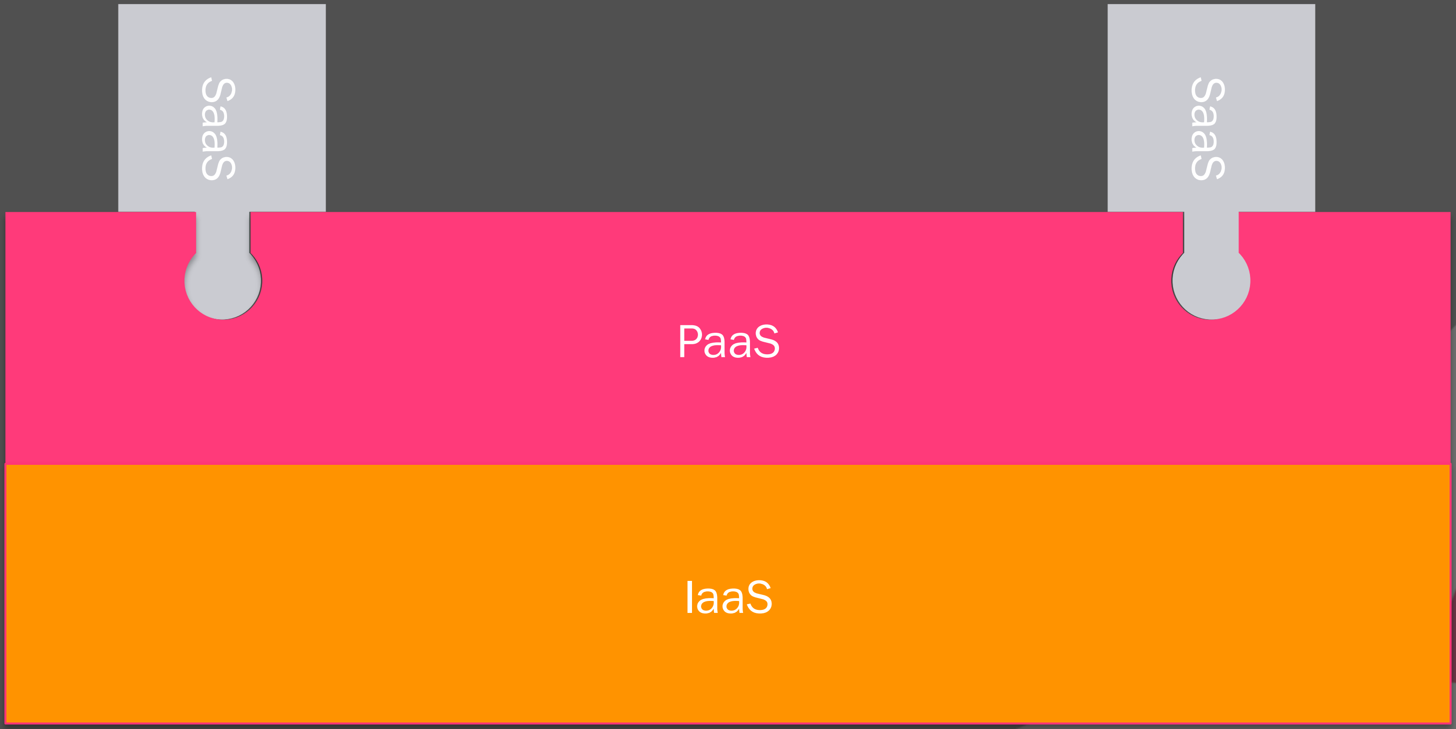
The Cloud

PaaS



The Cloud

PaaS



- Hosting Environment
  - Only HE configuration required for the SaaS
    - The SLA

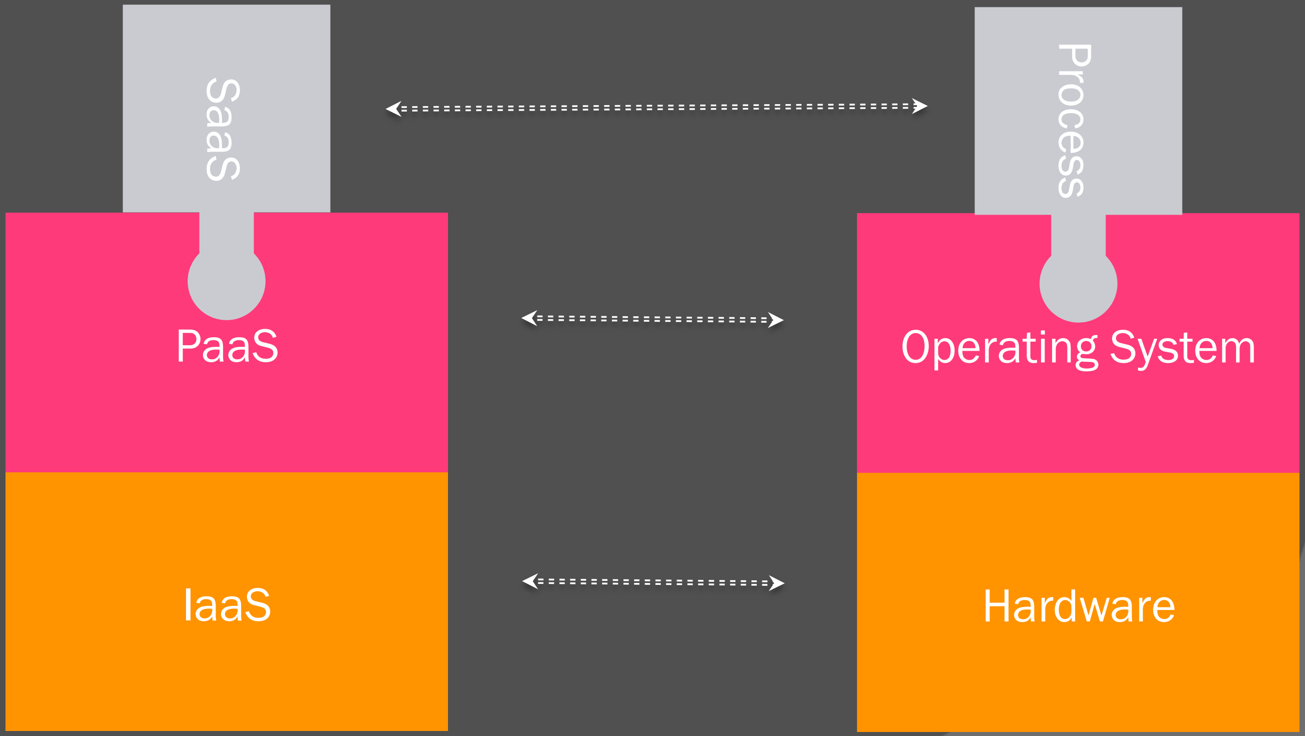
The Cloud

PaaS

- A PaaS is also a SaaS…
- This seems to be confusing
  - Many Services marketed as PaaS fail to act as a hosting environment
  - Fully managing services on top of them
  - Tend to behave more like services on which to take a dependency

The Cloud

PaaS



Elasticity

The Cloud

Generic Elasticity requirements

- Autonomy
  - *Sensors/Effectors*
  - *No human need apply*
    - *Costly and error-prone*
- *Scalability*
  - *The structure makes it possible to adapt to changing loads*
    - *Horizontal scalability as a condition*
      - *Vertical should also be considered*
- *Adaptivity*
  - *Amount of resources adapts to circumstances*
    - *Driven by SLA*

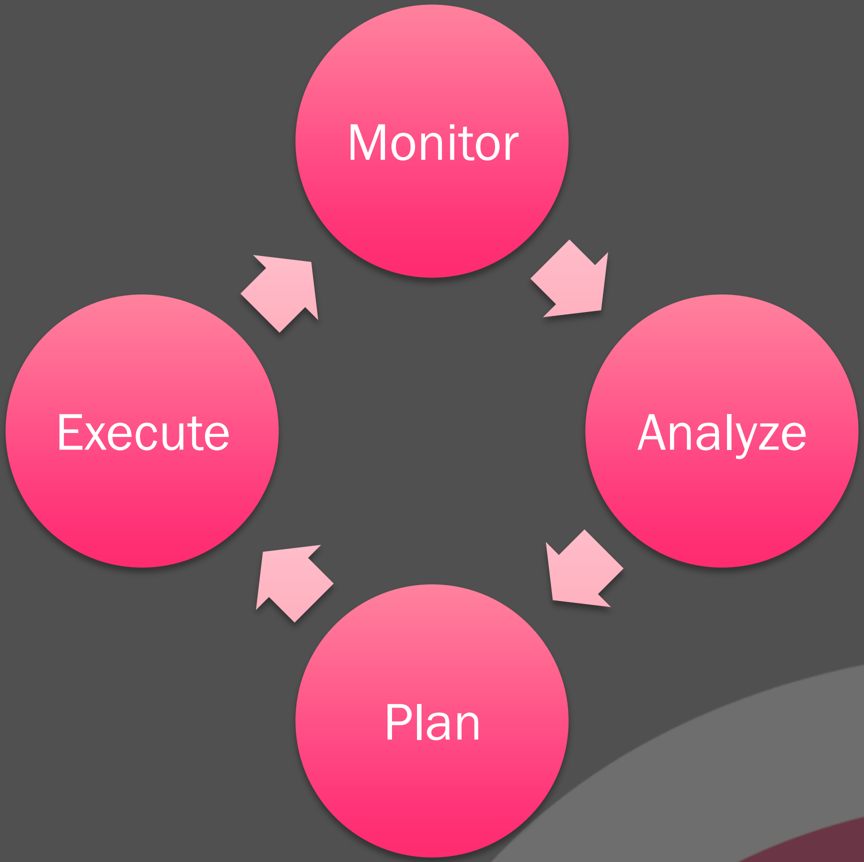
The Cloud

PaaS Elasticity requirements

- SLA-Awareness
  - *SLA made explicit to PaaS*
  - *Drives all PaaS decissions*
- *Composability*
  - *Inter-relations are important*
  - *Gives important hints about behavior*
- *Service continuity in software updates*
  - *Changes in software must keep service running*

PaaS Elasticity

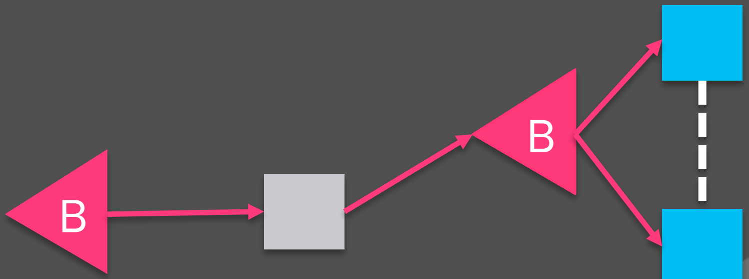
Autonomy



PaaS Elasticity

Scalability management

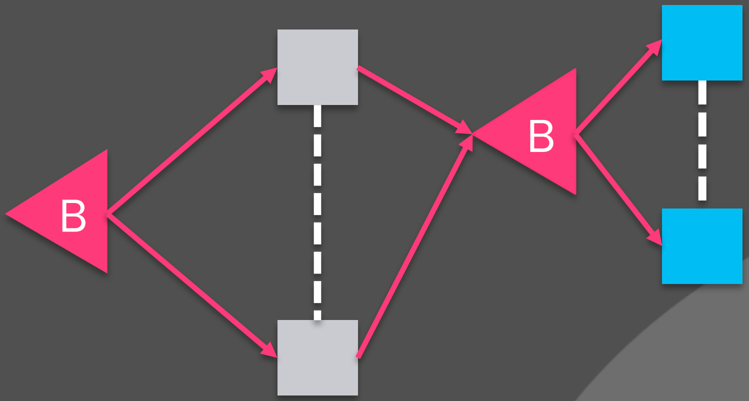
- Replication – Horizontal Scalability
  - *Vary number of instances of particular components*
    - *Complication: consistency*
      - *Use weak models when possible*
  - *Load balancing mechanism*
    - *Sticky sessions to maintain "state"*



PaaS Elasticity

Scalability management

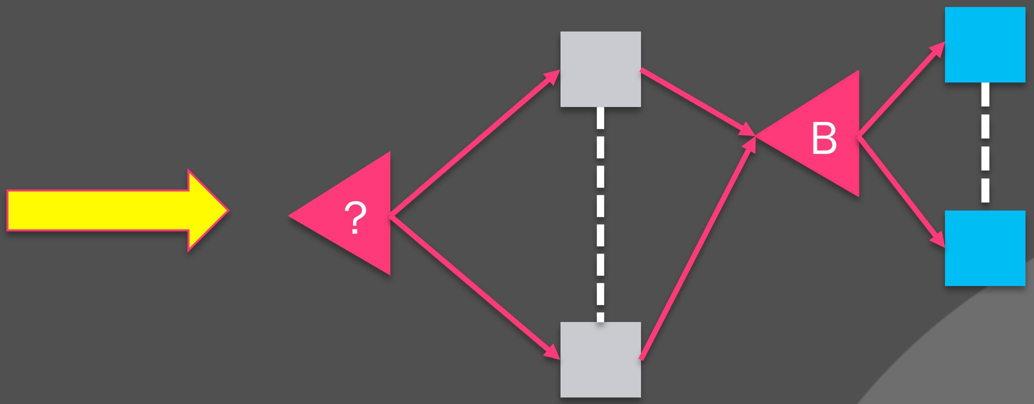
- Replication – Horizontal Scalability
  - *Vary number of instances of particular components*
    - *Complication: consistency*
      - *Use weak models when possible*
  - *Load balancing mechanism*
    - *Sticky sessions to maintain "state"*



PaaS Elasticity

Scalability management

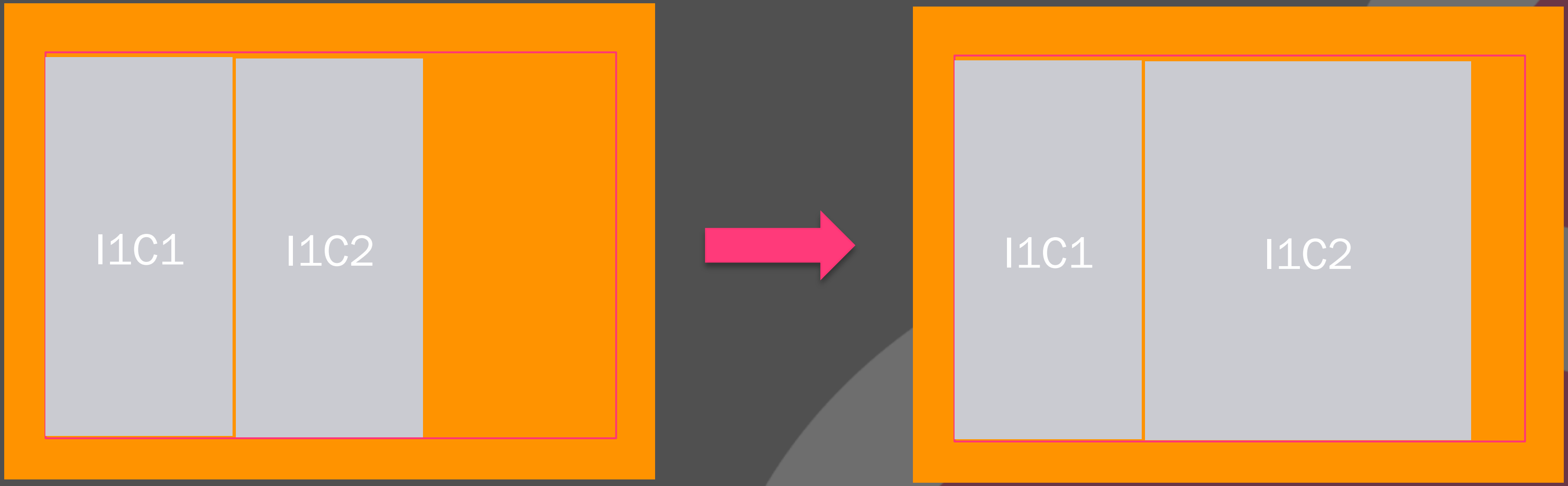
- Replication – Horizontal Scalability
  - *Vary number of instances of particular components*
    - *Complication: consistency*
      - *Use weak models when possible*
  - *Load balancing mechanism*
    - *Sticky sessions to maintain "state"*



PaaS Elasticity

Scalability management

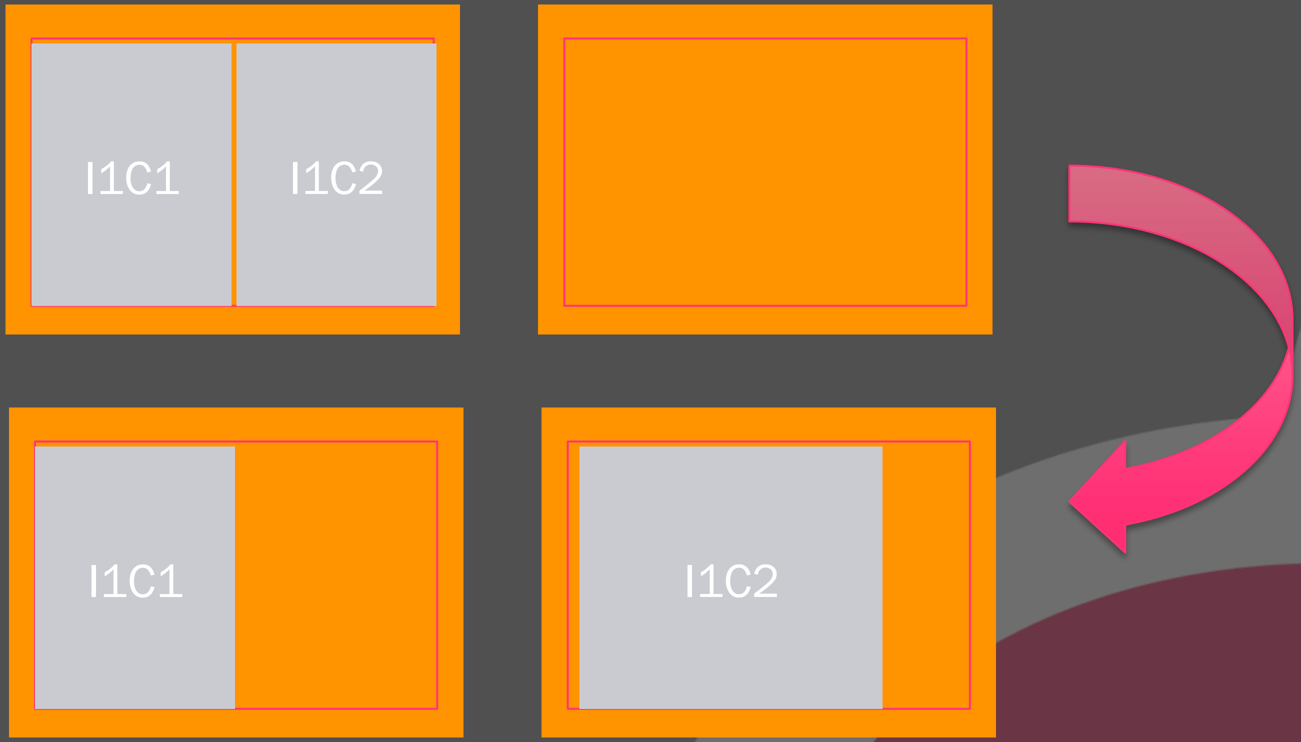
- *Re-dimensioning*
  - *Change the capacity of the environment*
    - *Needs spare capacity*
    - *Reduce complexity by having discrete choices*
  - *Lack of generalized support for run-time changes*
    - *Downtime possible*



PaaS Elasticity

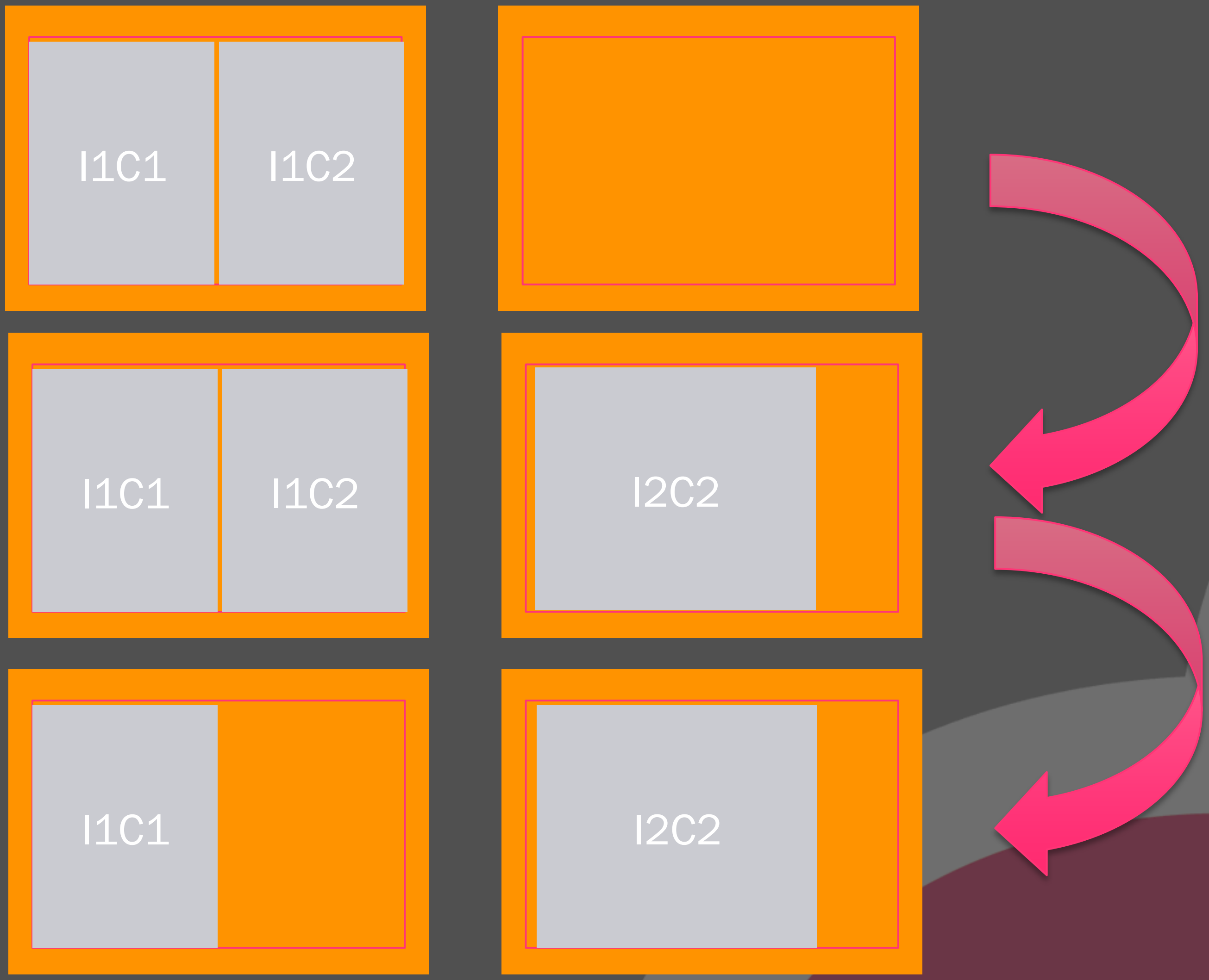
Scalability management

- *Migration*
  - *Challenge: state*
    - *Including network connections*
    - *Bandwidth concerns*



PaaS Elasticity

Scalability management



PaaS Elasticity

Scalability management

- Sharma (Kingfisher):
  - Efficiency in resource Provisioning
  - Then Minimize latency
  - *Mixture of several approaches*
  - *Federates Private/Public IaaS*
    - *Re-dimensioning in private*
- *Knauth-Fetzer*
  - *Live migration*
    - *Lengthy transfer times/Potential disruption in service*
  - *Stop/Restart*
    - *Restart on-demand: faster/lower latency*

PaaS Elasticity

Being Adaptive

- Two different dimensions:
  - Adaptability
    - *Potential of adapting a system*
      - *<u>Designed</u> with this in mind*
    - *Means needed to be scalable & meet QoS*
      - *More general concept than just scale*
  - *Adaptivity*
    - *Adaptability & finds out when it is time to adapt*
    - *How to adapt?*
      - *Autonomy must be part of the equation*

PaaS Elasticity

Adaptivity approaches

Proactive

Reactive

PaaS Elasticity: Adaptivity

Proactive

- Come up with two models
  - Load Prediction
  - Performance prediction
  - How?
- Fit current facts
- Predict the future
- Reconfigure if predictions warrant it
  - According to a performance model…

PaaS Elasticity: Adaptivity

Reactive

- Decide what to monitor
  - *Must be an indication of performance*
  - *Must be easy to measure*
  - *Examples: Resource usage*
    - *CPU usage*
    - *Memory Usage*
    - *….*
- *Reconfigure if*
  - *Performance endangered*
  - *Resources idle*

Proactive Adaptivity

Software Performance Prediction (SPP)

- Performance by design
  - Reliable software must be “performant”
- If architecture is wrong,
  - Reachable performance will be poor
  - Care with components and inter-dependencies
  - Care with software life cycle: changes happen
- But… also
  - Build a performance model
    - With enough detail to predict performance from inputs

Proactive Adaptivity

SPP Techniques

- At design time
  - Based on software architecture
- Requires a behavioral (code) model
- Somebody (Soft. Architect) maps both
- Difficulties
  - Gap between performance/behavioral models
  - Behavioral models may change often

Proactive Adaptivity

SPP Techniques

- Many approaches use UML diagrams for behavioral models
  - – Automate diagrams,...
- Generating variants of queuing networks for performance models
  - – Good thing: expressive power for composability
    - Help identify problematic components
      - – Qualitative at high levels
- Other:
  - – Process algebras, Petri nets,...

Proactive Adaptivity

SPP Techniques

- A PaaS would be in a good position
  - Provides tools for modeling
  - Result is directly understood by PaaS hosting environment
    - Drives decisions
- However …

Proactive Adaptivity

SPP Techniques

- Problem:
  - Software is not built this way in the real world
  - In a complex product, many different actors intervene
    - With different competence levels
    - At different times
    - From different organizations
    - Even if all are competent
      - The complexity of the composition makes it difficult to validate models
  - The real model is the code!
    - As it mutates too rapidly

Proactive Adaptivity

SPP Techniques

- A PaaS would be in a good position
  - Provides tools for modeling
  - Result is directly understood by PaaS hosting environment
    - Drives decisions
- However …
  - We can only hope for a relatively high level of description from the designers/architects
    - And drive other approaches…

Proactive Adaptivity

SPP Techniques: Mining the model

- Produce a high level description of a service
  - Components
  - Intra-dependencies
- Obtain a high-level performance model
  - Parameterized
- Deploy with SLO targets
  - Benchmark
  - Fit parameters

Proactive Adaptivity

Workload prediction

- Akin to data-mining
  - Statistical analysis/learning of load
  - Some approaches combine several predictive methods
    - Adjust weights as they are contrasted
- On occasions, we have extra information
  - Some SaaS are subject to periodic/point-in-time peaks which are known
    - Burst-type loads fall in this category
    - Two-stage services

Reactive Adaptivity

Feedback-driven reconfiguration

- Set of rules based on metric thresholds
  - Rules generally static
  - Thresholds may vary over time
  - Implicitly or explicitly based on some performance model
    - Metrics observations predict performance.
- Inputs for decisions based on current measurements by monitoring system

Reactive Adaptivity

Feedback-driven reconfiguration

- Questions
  - How often to monitor/measure
  - How many thresholds per metric
  - How many metrics to observe
  - And…
  - What works better
    - Prediction
    - Reaction

Reactive Adaptivity

Feedback-driven reconfiguration

- How often to measure/act
- How many thresholds
- Predictive vs Reactive
- At least one work on the subject
  - Metric: CPU utilization
  - One vs two thresholds
  - 1m vs 5m measurement period
- Predictor
  - Simple statistical adjustment over latest history

Reactive Adaptivity

Feedback-driven reconfiguration

- For 1 m intervals
  - Predictive bettered reactive
- For 5m intervals…
  - Single-threshold policy betters predictive
    - Even when predictions taken every 1m
- However
  - Not clear the comparison clarifies much
    - What about other predictors?
  - But at least raises some doubts on the value of "predicting"

PaaS Elasticity

What to do with SLA

- Guarantees a service provider assures
  - – Under conditions that must be met by a service user
- IaaS focus on availability
  - – Also on reserve capacity available
- SaaS focus on…
  - – Availability but at given performance levels
    - SLOs

PaaS Elasticity

SLA of PaaS

- What should PaaS focus on?
  - Guaranteeing SaaS SLOs
    - At a price ☺
- Challenge:
  - Managing SLOs from different SaaS
  - Efficiently for PaaS providers
  - Predicting the cost for its SaaS
  - Better than SaaS provider can do themselves
  - Many more…
  - Must impose SaaS-Expression to take on this job

Compound Services

Why

- Most studies use single-component services
- More flexibility
  - Higher granularity of decisions
  - Potentially, partial functionality
    - Under failure conditions
  - Scaling may affect a simple component
    - Not a large one
      - Savings in resources being used
  - Prediction beyond a component in trouble
    - May indicate need to also act on dependencies
    - Better predictability
- It is part of the behavioral model
- Of course,..., reusability

Software Upgrades

What is the impact?

- Software is rarely static
  - Unless dead
- Most of it is developed in cycles that produce incomplete/imperfect/unfulfilled functionality
- Upgrades MUST be contemplated
  - This affects IaaS and PaaS providers too…
  - Ad-hoc methods for them
    - PaaS-supported methods for SaaS

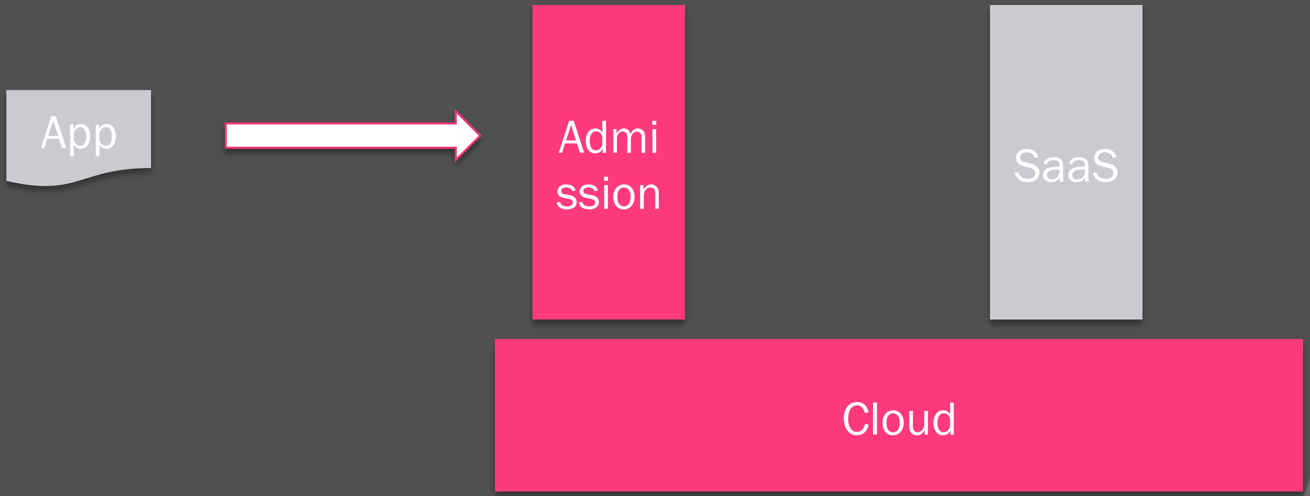
Software Upgrades

Factors

- Global Consistency
- Service Availability
  - Quiescence
- Coexistence => Service continuity
  - Dynamic versioning tagging requests
- State transfer
  - Problematic without replication
- Minimize overhead
  - Many tasks to be performed

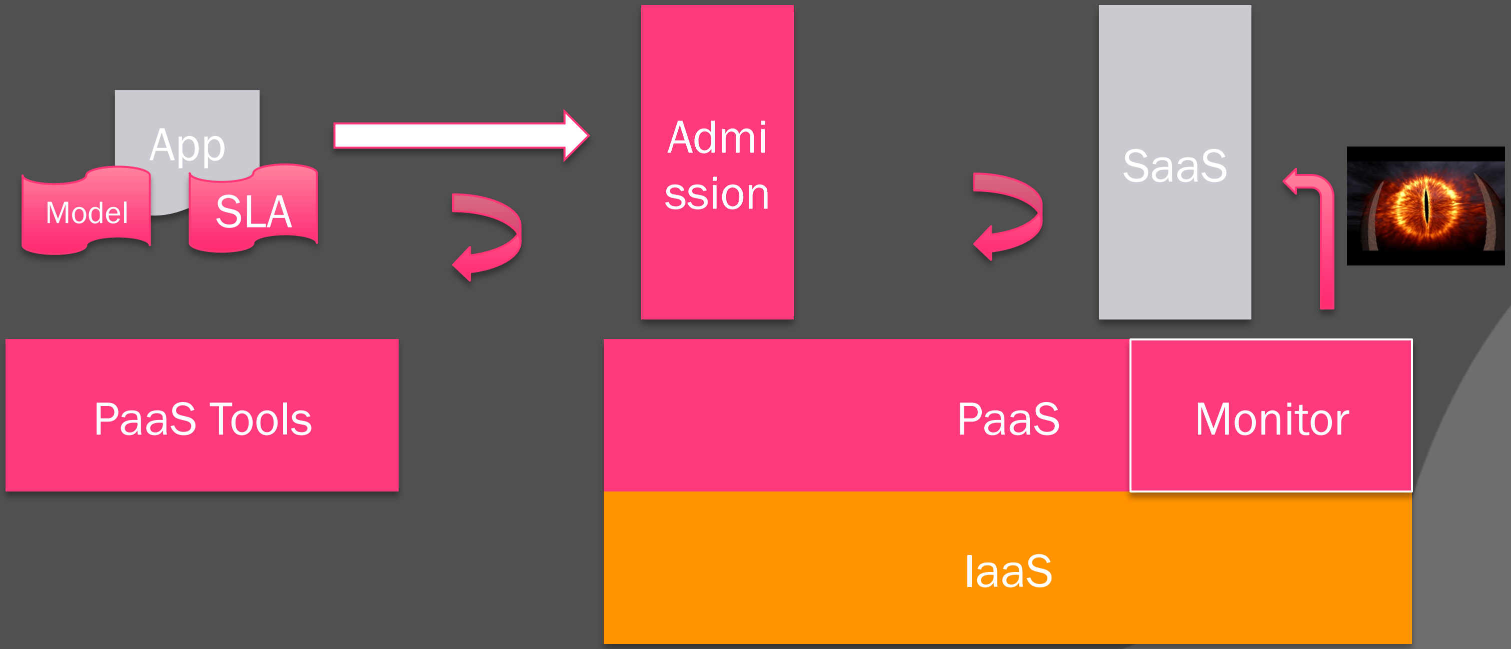
Ideal

How close are we?



Ideal

How close are we?



Not close enough: No system in this space

Open Questions

- Everything SLA
  - Difficult to capture the SLO aspects
    - What is the service, in sufficient detail?
  - How to capture reciprocals/obligations
    - To derive penalties, when due
  - How to express PaaS SLA given SaaS SLA
  - Predict/React/Both?
    - No clear approach (yet) to avoid complex performance models
    - Evaluation periods
  - Is it worth it?
    - Are simple reactive strategies enough?
- Fool-proof upgrades
- Dealing with dependencies
- Dealing with failures