Software as a Service Engineering

Richard Sharp
Director of Studies for Computer Science, Robinson College

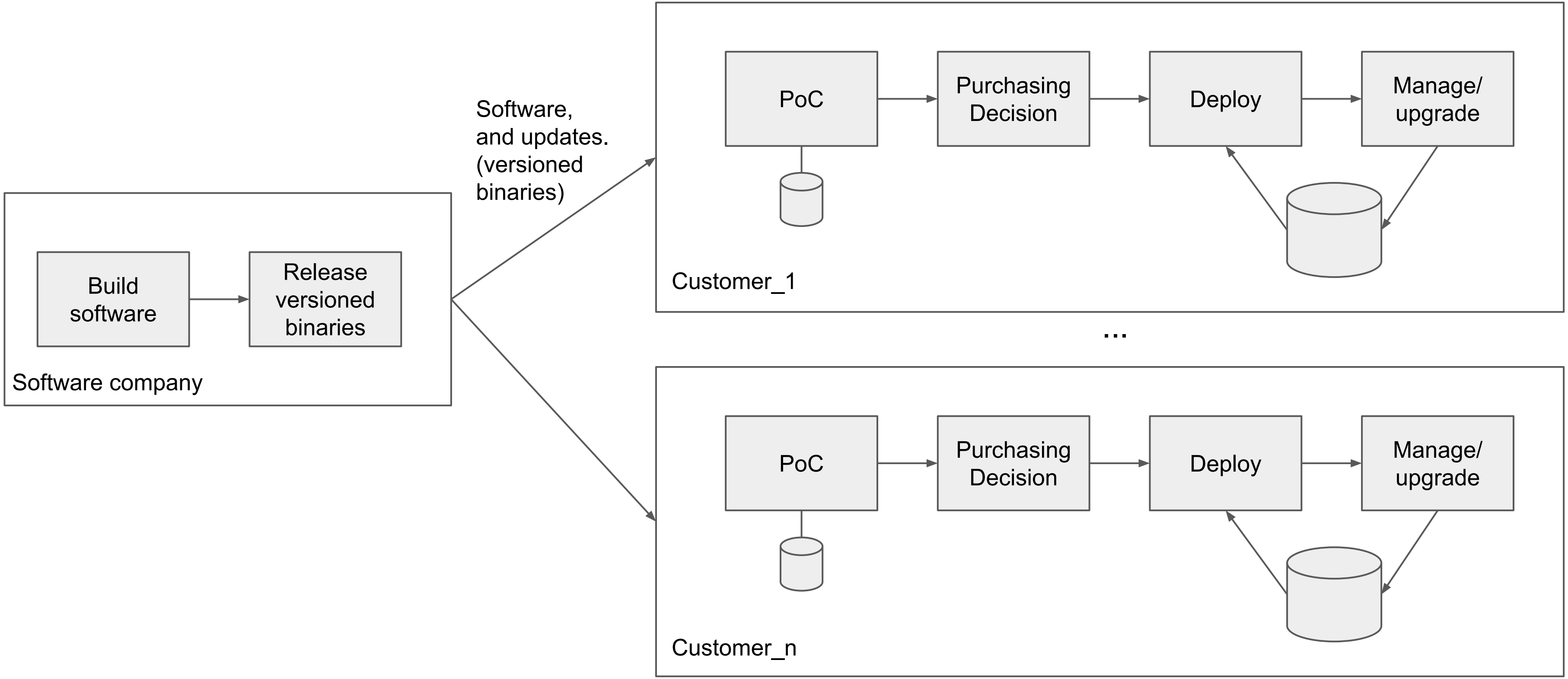
What is SaaS?

SaaS (Software as a Service) refers to software that is

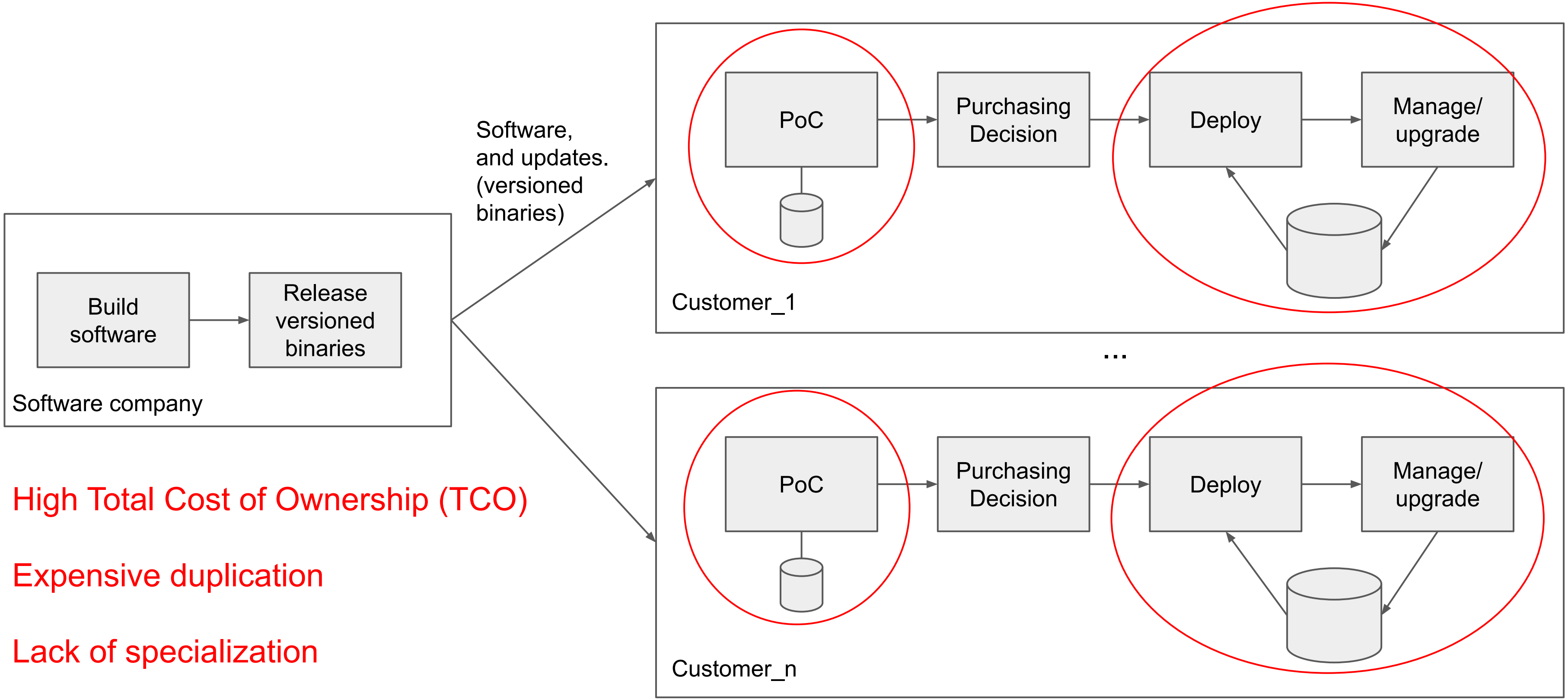
hosted centrally and *licensed to customers on a subscription basis*.

Users access SaaS software via *thin clients*, (often web browsers).

Traditional software distribution (pre SaaS)



Traditional software distribution (pre SaaS)

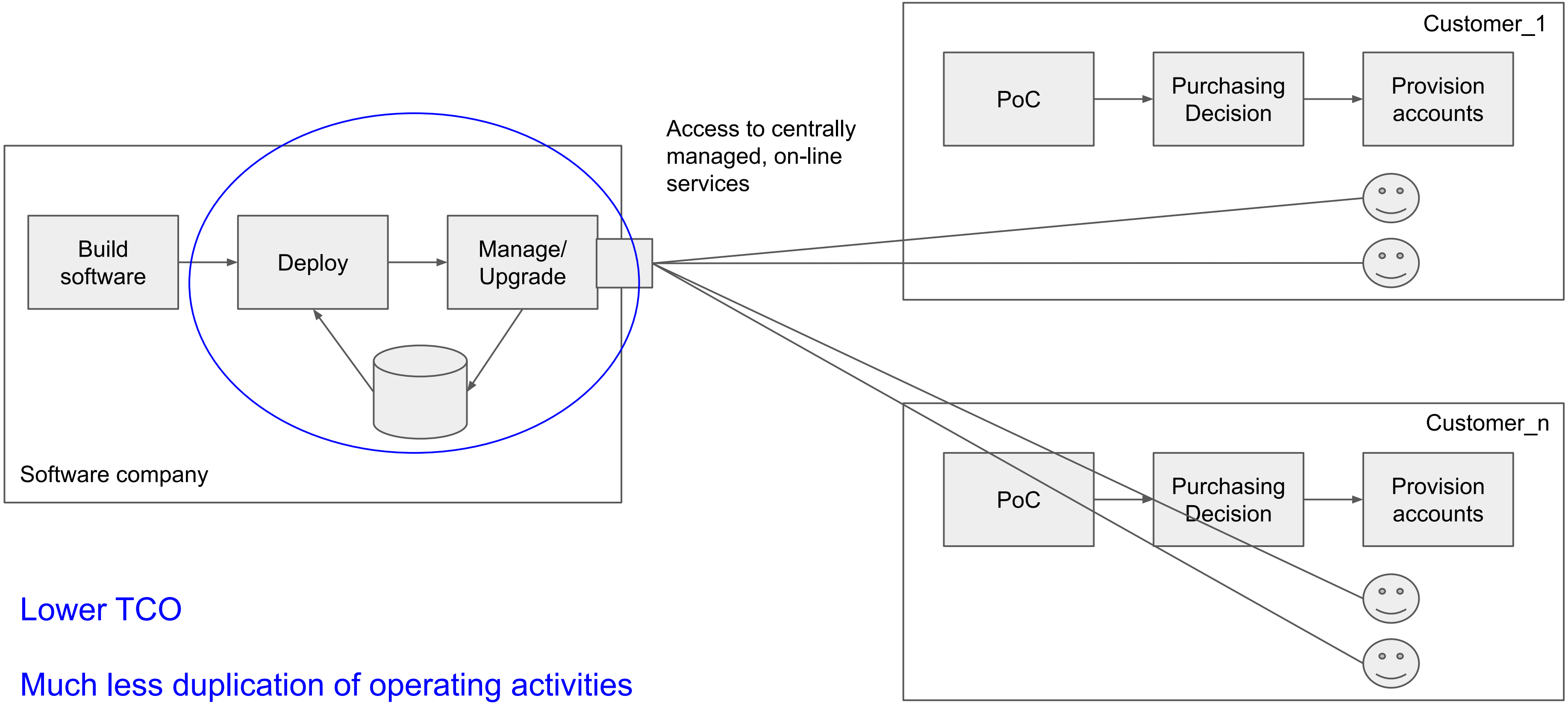


High Total Cost of Ownership (TCO)

Expensive duplication

Lack of specialization

SaaS



Lower TCO

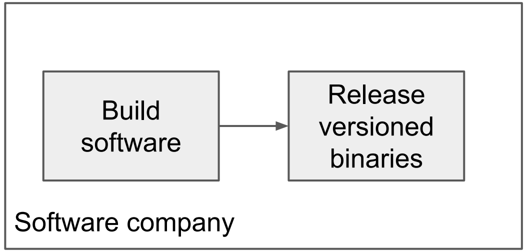
Much less duplication of operating activities

Much better specialisation in this division of labour

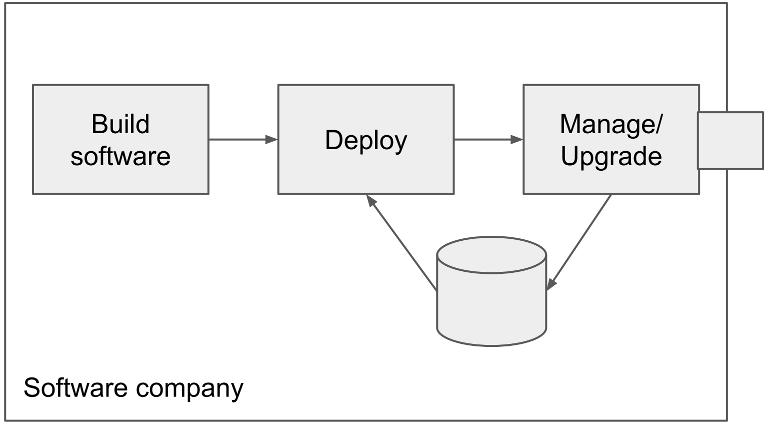
Impact of SaaS on the Software Engineering Process

Impact on the ‘software company’

Binary distribution



SaaS



Impact on the ‘software company’

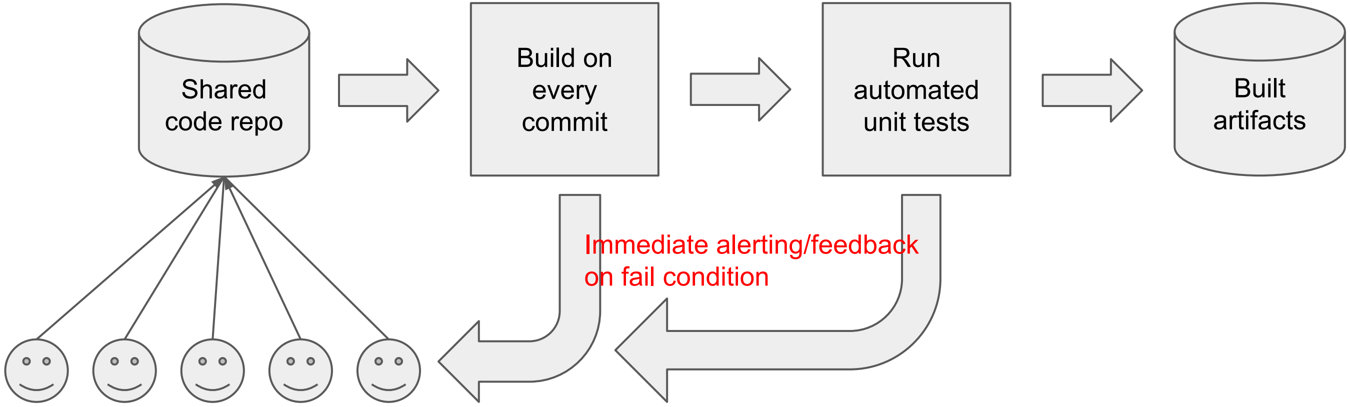
- Now have to worry about building software *and running it*
- Have to continue evolving/upgrading the software with *zero downtime*

But the good news:

- ‘Software release’ no longer an all-or-nothing discrete event
  - Provides new ways to manage quality and reduce risk
- Continuous visibility into user behavior
  - Provides user/commercial insights back into iterative software development process

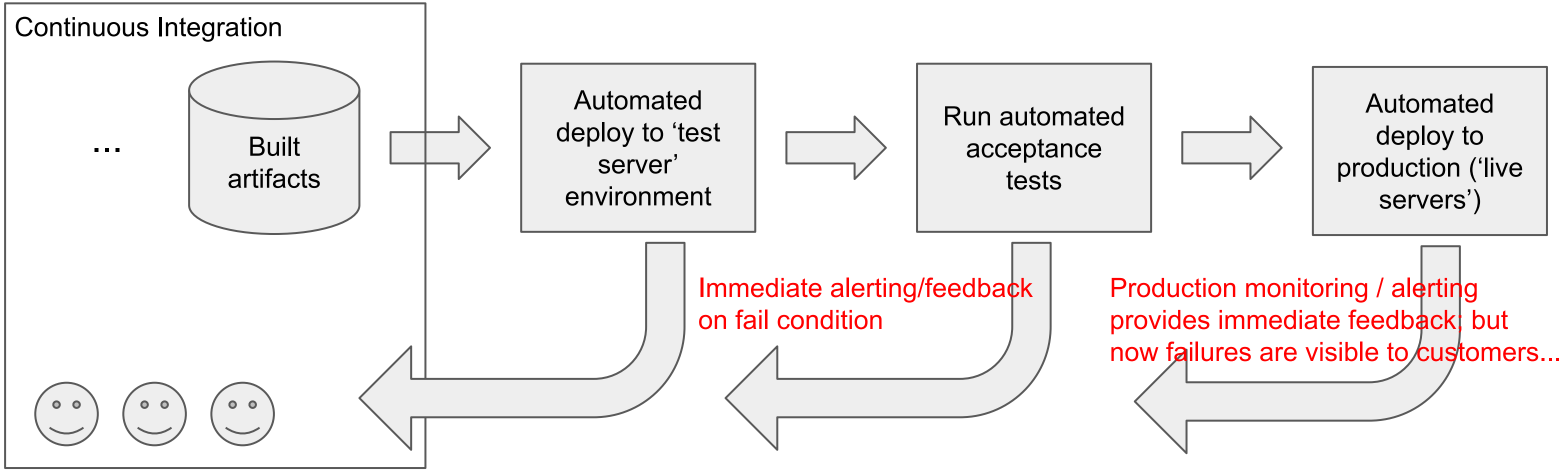
Managing Continuous Deployment Without Downtime

Continuous Integration (CI): short integration cycles lead to greater throughput

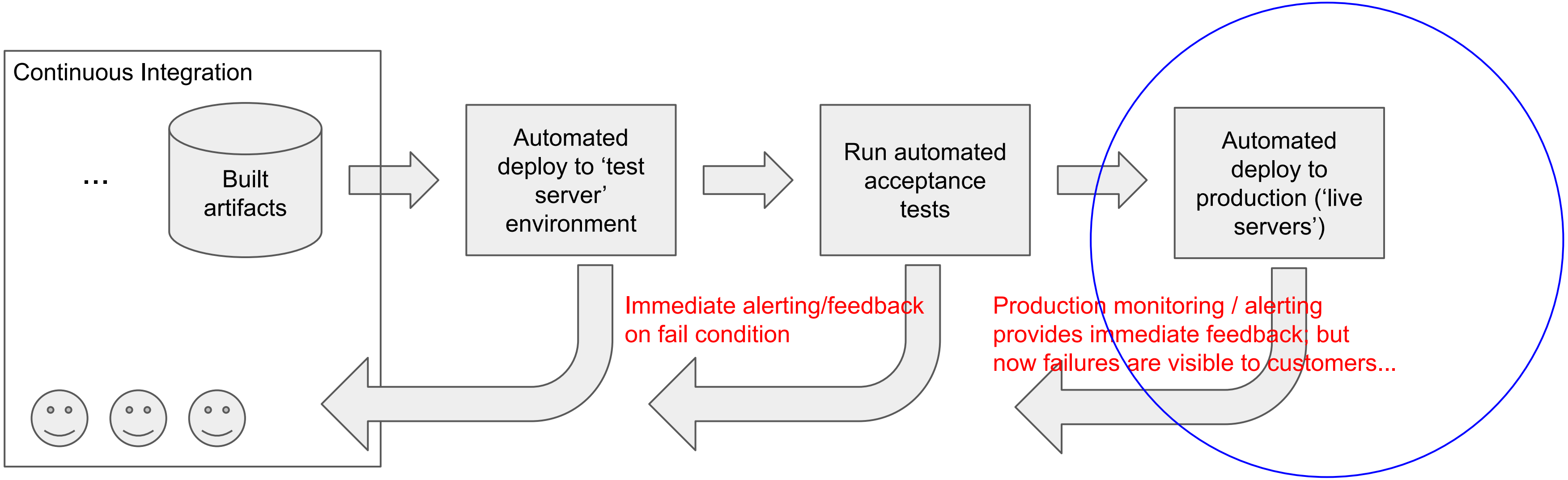


Developers commit to shared dev ‘mainline’ branch frequently (e.g. at least once a day)

Continuous Deployment (CD): bring ‘deploy’ into the ‘short cycle’

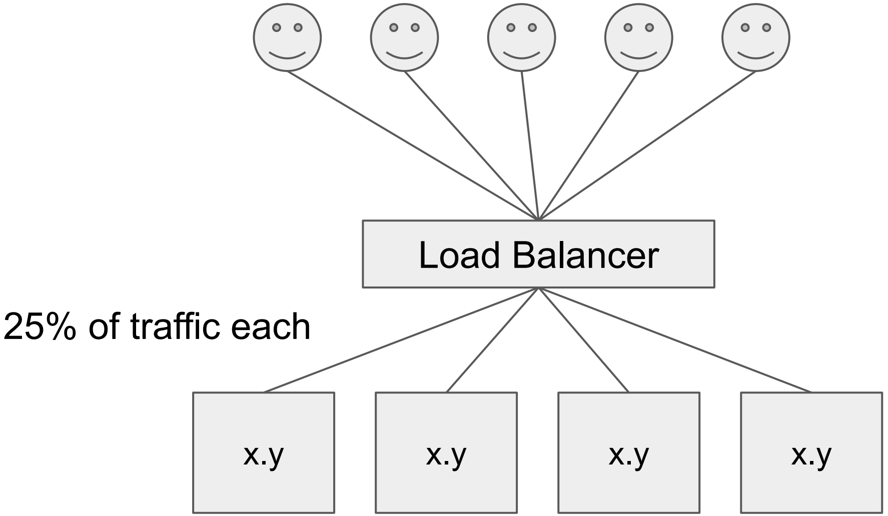


Continuous Deployment (CD):
bring ‘deploy’ into the ‘short cycle’



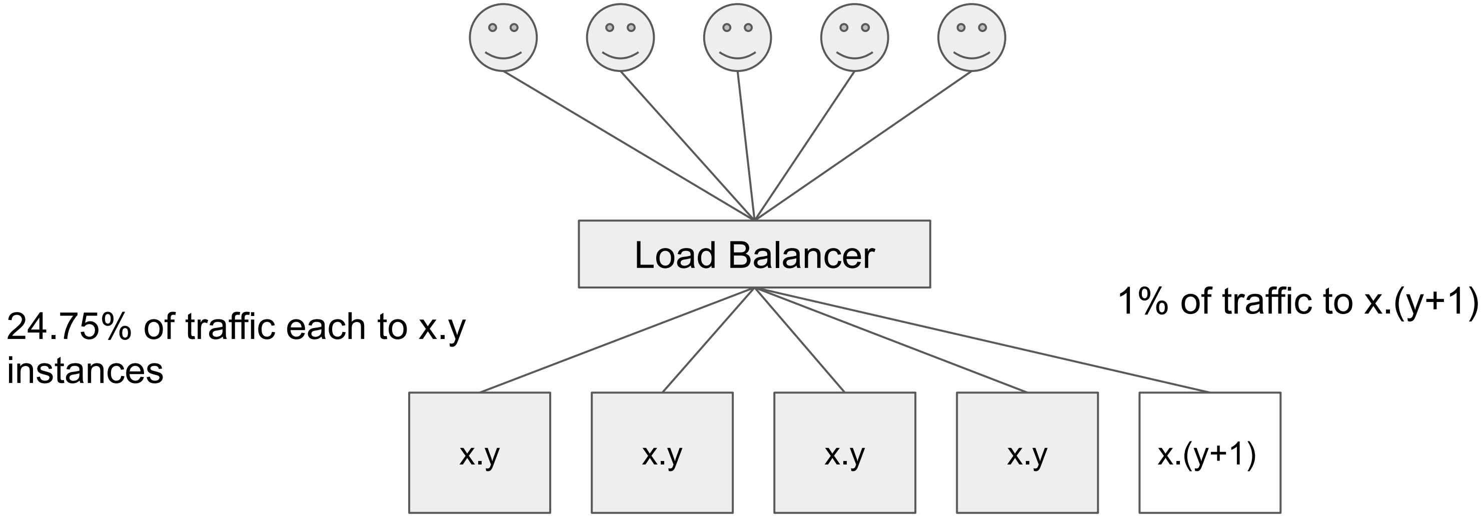
How to do this while reducing risk?
How to do this while ‘always on’?

Rolling deploy

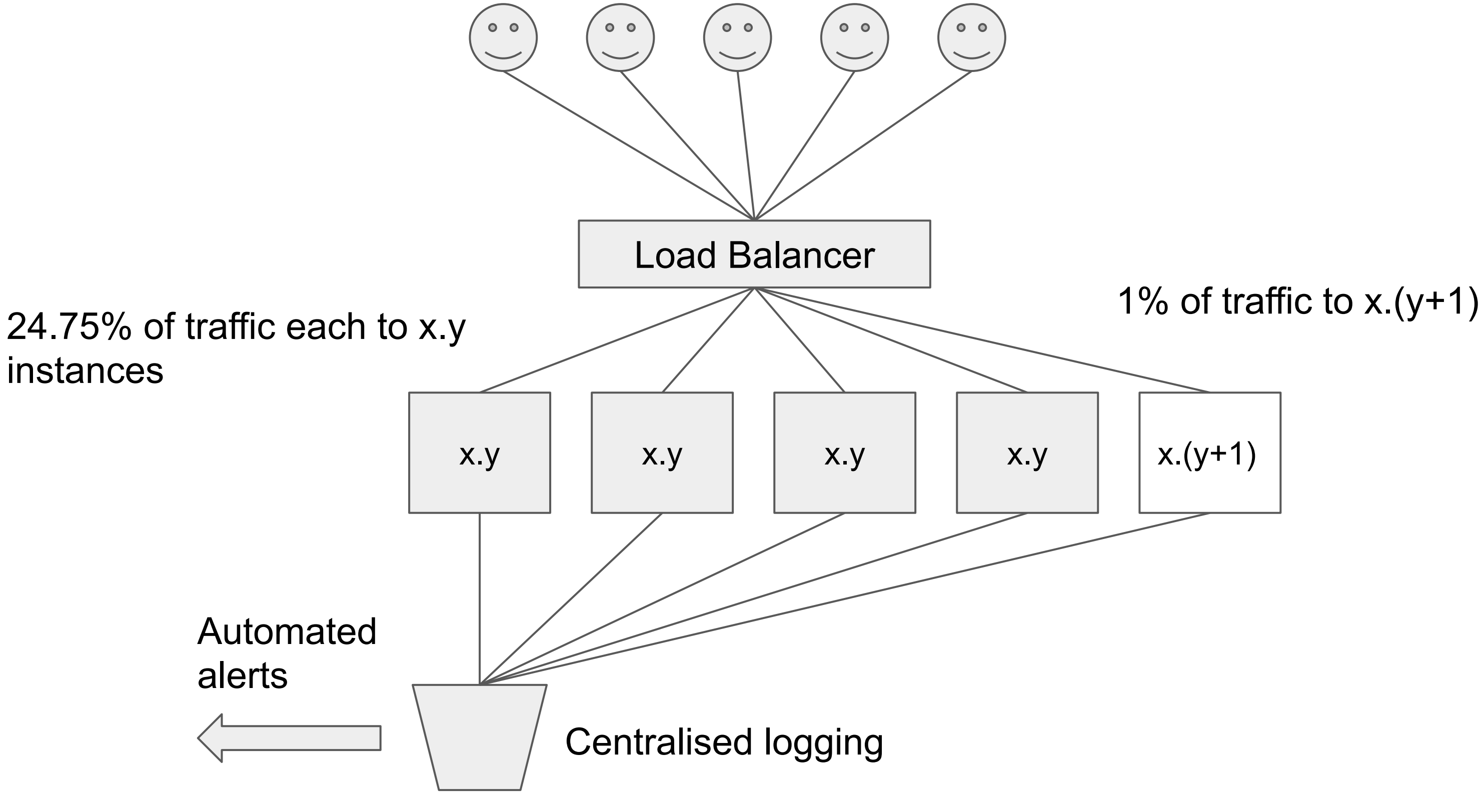


Note: these resources are usually running in a cloud platform. So virtual machines, load balancers, storage, network etc. can all be provisioned and configured through the cloud platform's APIs.

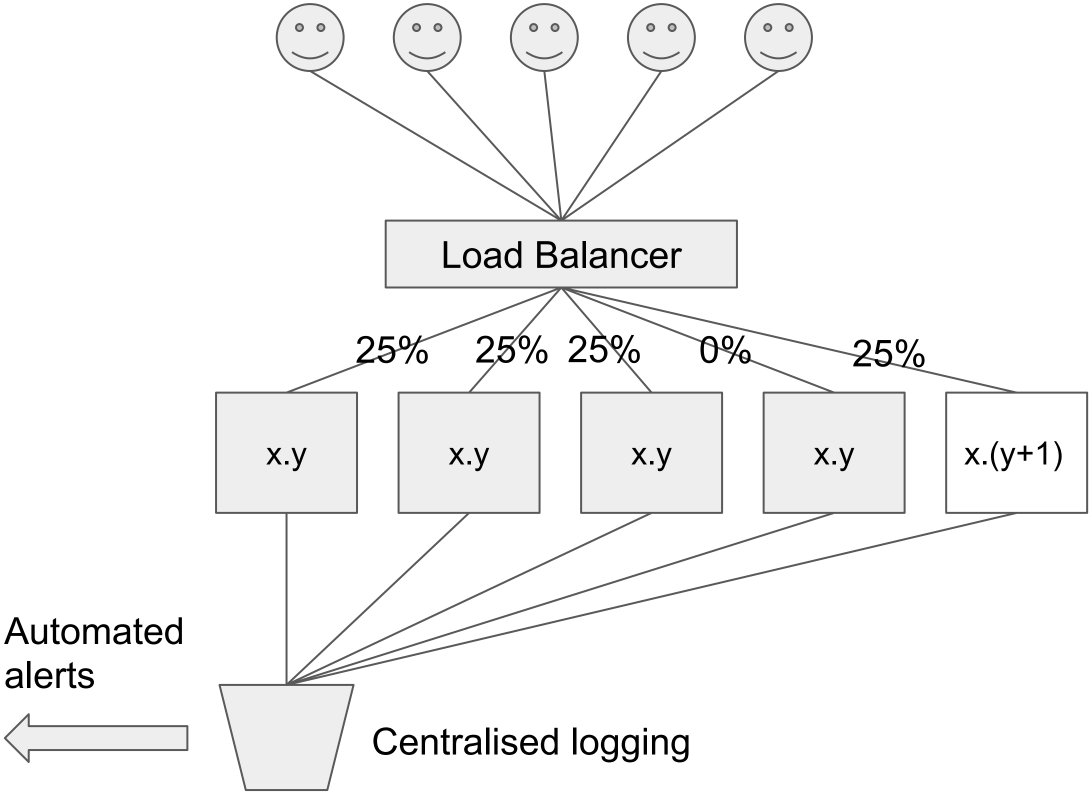
Rolling deploy: 1) Deploy ‘canary’ (limit exposure/risk)



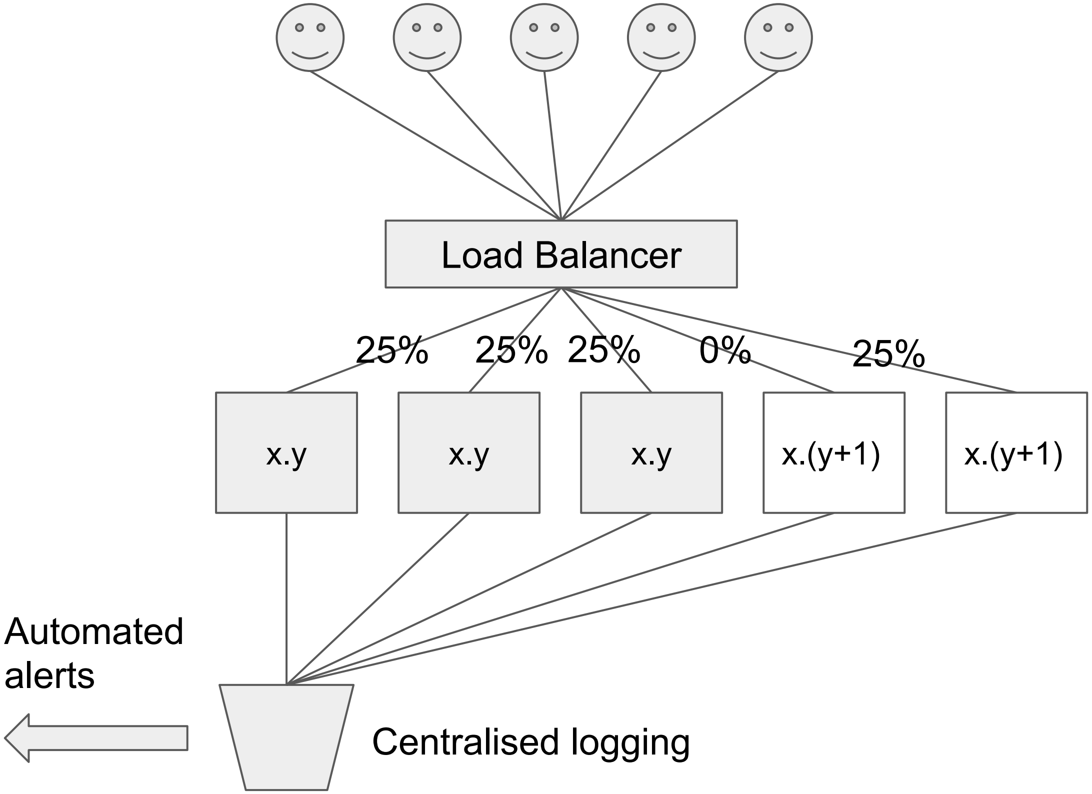
Rolling deploy: 2) Automated monitoring of error rates - OK?



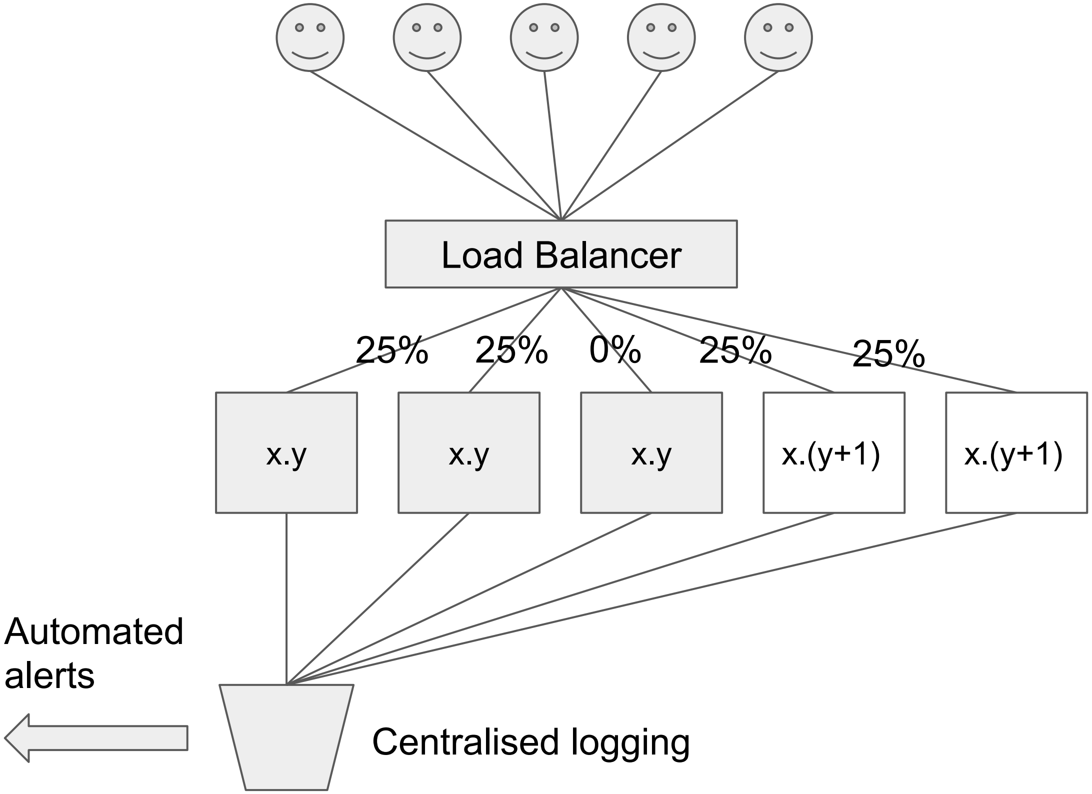
Rolling deploy: 3) Move traffic from old instance to new



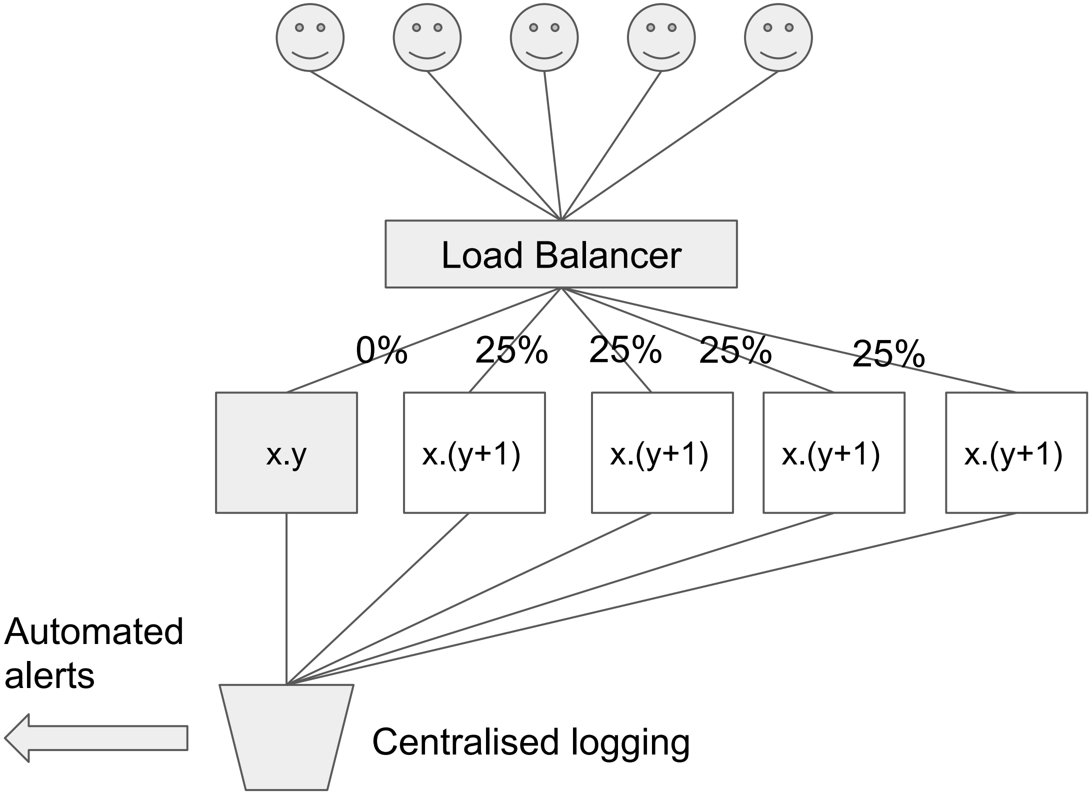
Rolling deploy: 4) Upgrade 0% instance



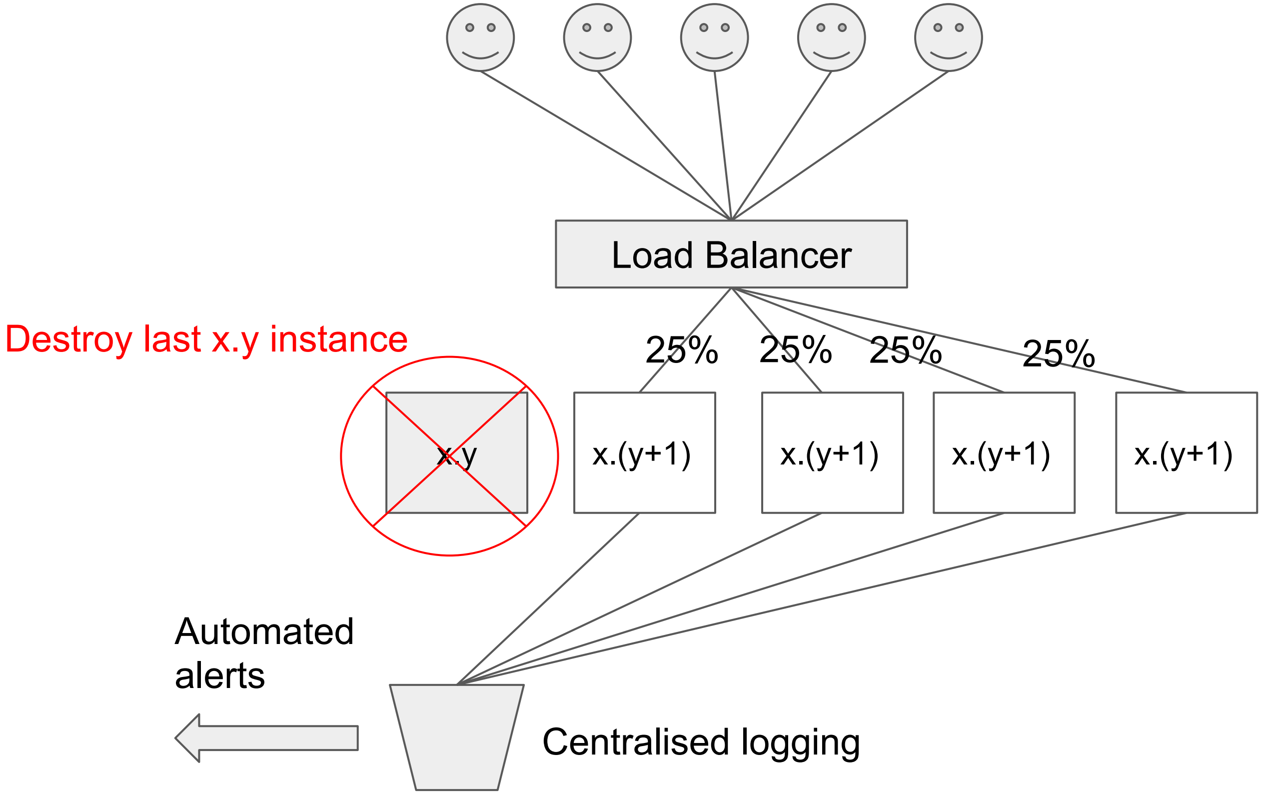
Rolling deploy: 5) Move traffic from old instance to new etc.



Rolling deploy: Repeat {move traffic old->new; upgrade old}

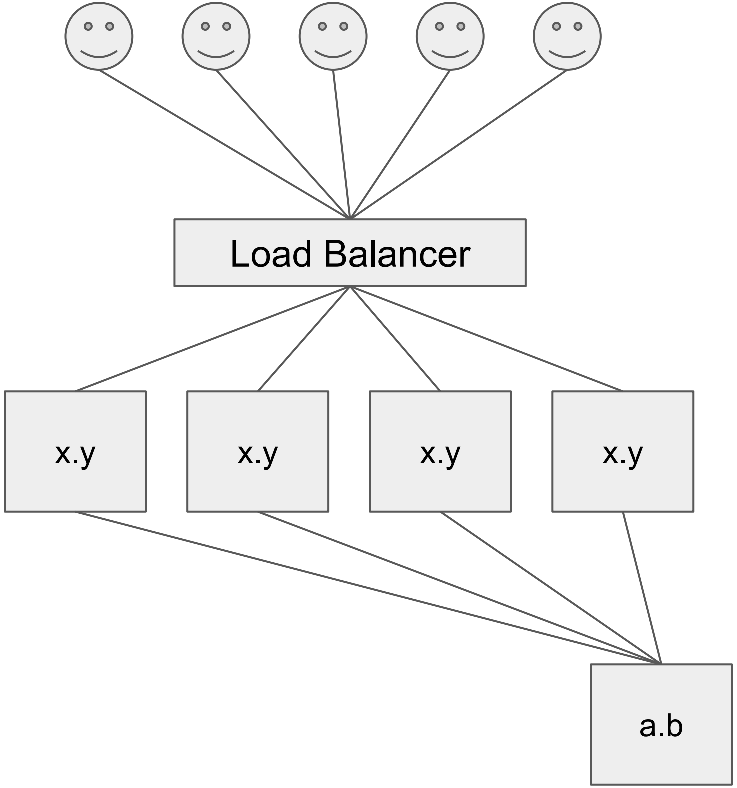


Rolling deploy: …



(If anything unexpected happens then can **pause** at any point; aim to 'roll forward' rather than 'rolling back'...)

Rolling deploy with service dependencies



Dependent service

Challenge:

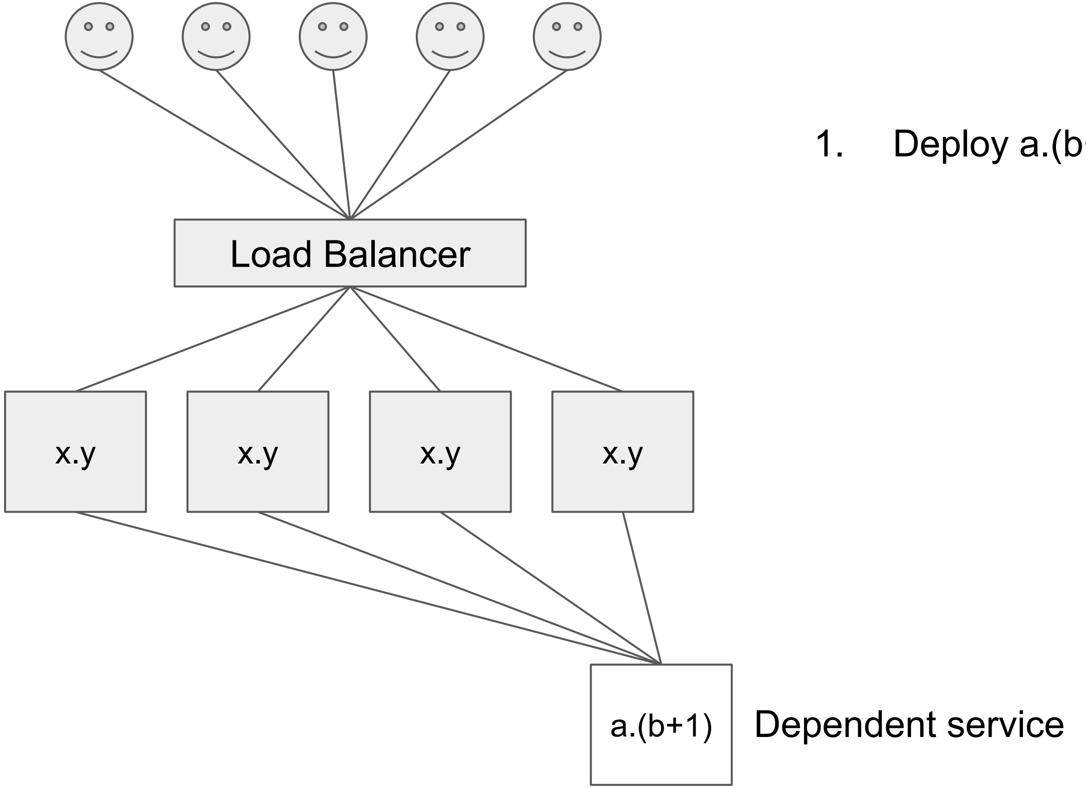
How do we upgrade the dependent service while keeping everything running?

And how do we handle this if we need to make a ‘breaking change’ to the dependent service’s API?

Rolling deploy with service dependencies

CONSTRAINTS:

a.(b+1) supports x.y
a.(b+1) supports x.(y+1)

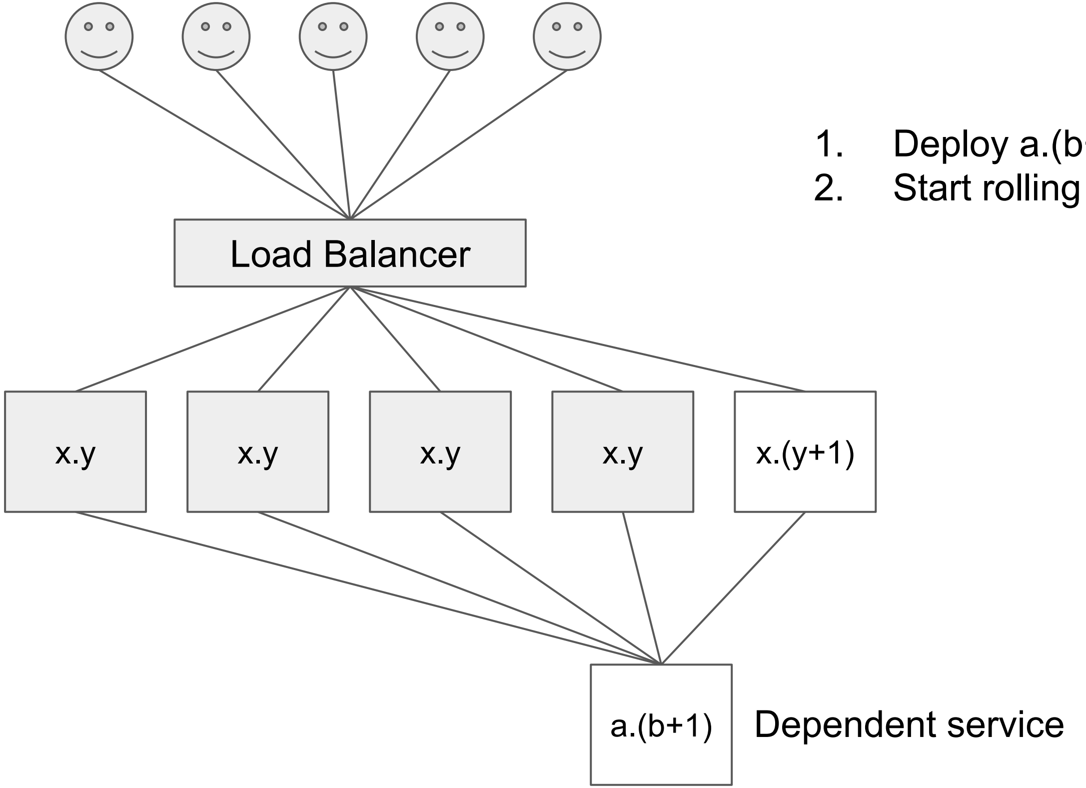


1. Deploy a.(b+1)

Rolling deploy with service dependencies

CONSTRAINTS:

a.(b+1) supports x.y
a.(b+1) supports x.(y+1)

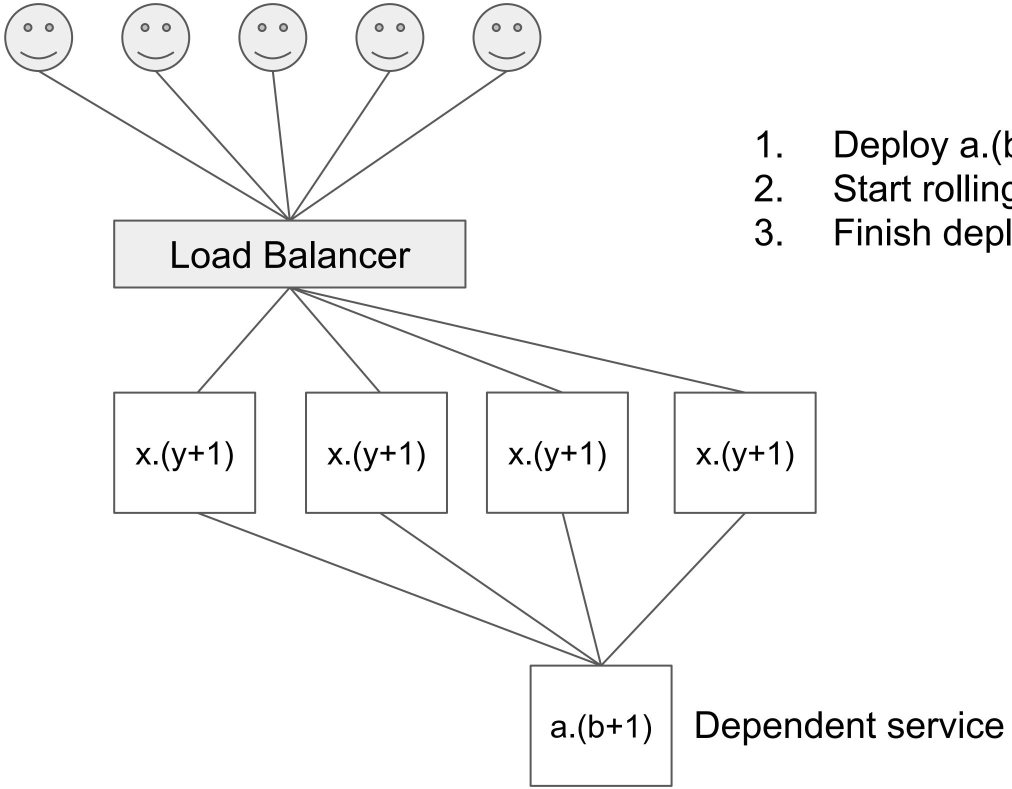


1. Deploy a.(b+1)
2. Start rolling out x.(y+1)

Rolling deploy with service dependencies

CONSTRAINTS:

a.(b+1) supports x.y
a.(b+1) supports x.(y+1)



1. Deploy a.(b+1)
2. Start rolling out x.(y+1)
3. Finish deploy of x.(y+1)

Rolling deploy with service dependencies

CONSTRAINTS:

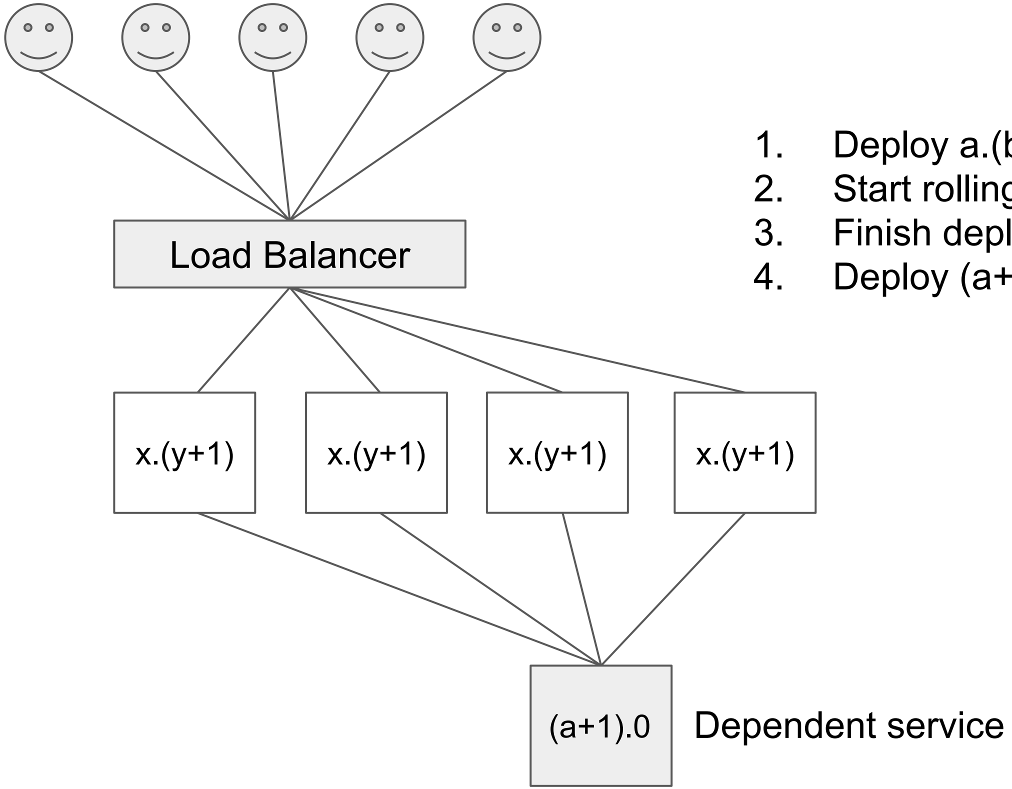
a.(b+1) supports x.y
a.(b+1) supports x.(y+1)

(a+1).0 supports x.(y+1)
(a+1).0 **doesn't** support x.y

We say:

a.(b+1)'s API is **backwards compatible** (wrt a.b)

(a+1).0's API introduces a **breaking change**



1. Deploy a.(b+1)
2. Start rolling out x.(y+1)
3. Finish deploy of x.(y+1)
4. Deploy (a+1).0

On Automation: Infrastructure-as-Code

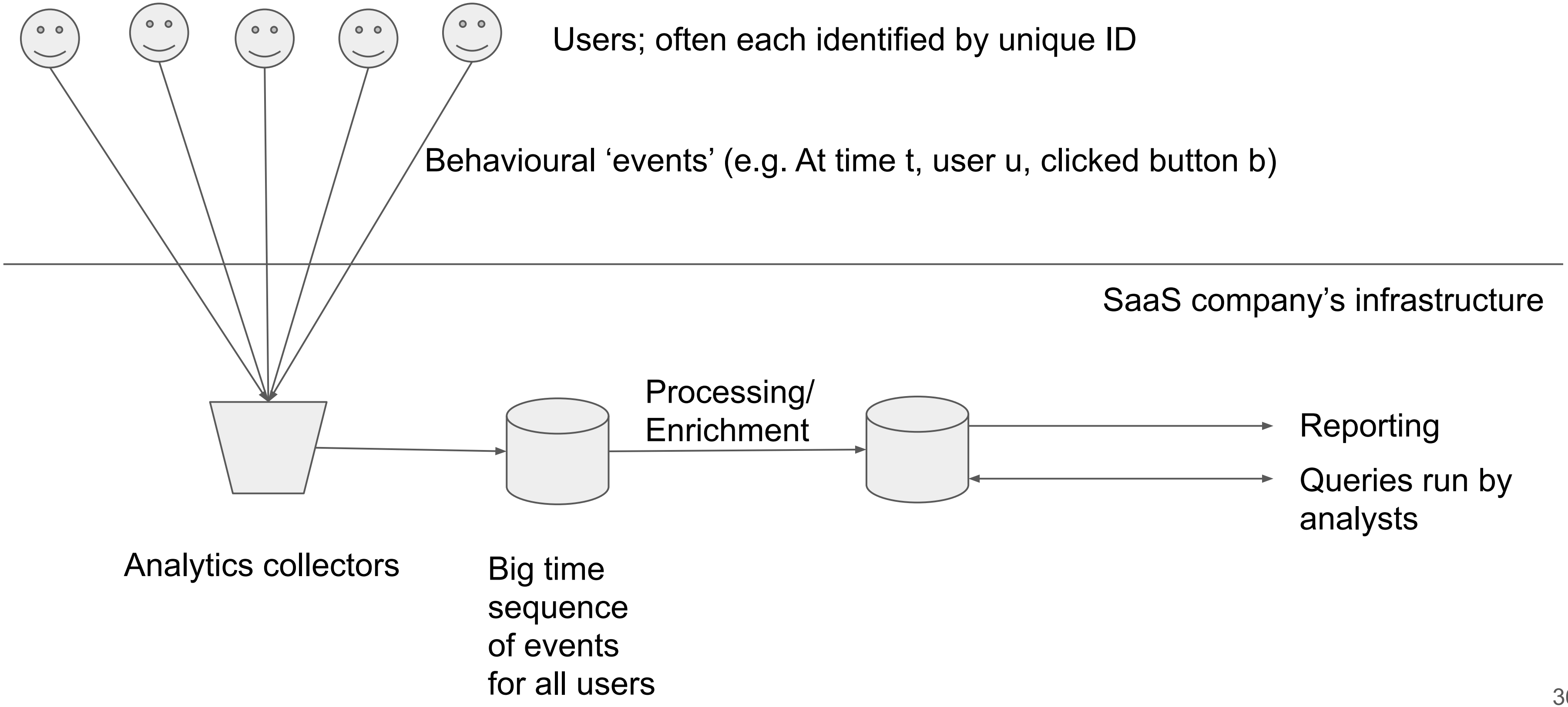
- Problem:
  - Manual deployments are time-consuming and error-prone. Subtle environmental differences cause bugs.
- Solution:
  - Write code to automate deployments, using Cloud APIs etc.
  - Put deployment code under version control, just like all other code
  - Have development teams write:
    - Application code
    - Code to test the application
    - **Code to deploy the application and its associated cloud infrastructure**
    - **Code to monitor the application and generate alerts**
- Frameworks like Terraform and CloudFormation help with this

Review

- Rolling deploy:
  - Technique for upgrading and developing SaaS software with zero downtime
  - Enables new ways of managing quality/risk, which changes the economics of testing
- Infrastructure-as-code:
  - Foundational technology for managing cloud-based SaaS services
  - Developers write code that enables applications to deploy and monitor themselves

Behavioural analytics and experiments

A simple behavioural analytics pipeline



What can we learn from the event logs?

- User/growth metrics:
  - Monthly Active Unique Users (MAU); Daily Active Unique Users (DAU)
- Engagement:
  - Time spent using the service
- Feature usage/growth/engagement metrics:
  - X% of users tried feature F at least once in the last month
  - Y% of users used feature F2 for at least 5 minutes last week
  - Feature F3 usage growing at Z% year-on-year
- Insights based on user segmentation:
  - Users who signed up in January 2018 exhibit an average 2% monthly churn
  - Female users aged between 20-25 are X% more likely to use feature F at least once

What else can we learn from the event logs?

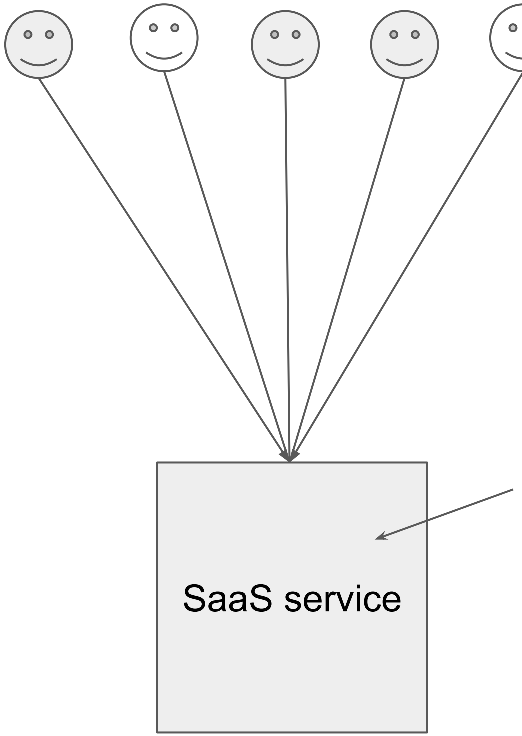
- Correlations
  - Usage of feature F2 is correlated with usage of feature F1
  - Daily time spent on the platform is correlated with the number of days since sign-up
- But NOT cause and effect… At least not without an experiment framework.

How can we move from correlations to cause/effect?

- Run controlled experiments:
    - Determine hypothesis to test
    - Determine level of exposure, E, (% of users that will go into experiment group)
    - Bucket users into either experiment group (E%) or control group (100-E)%
    - Release a change to the experiment group only
    - Measure relevant metric(s) in both control group and experiment group and determine whether the observed **difference** is statistically significant
- By measuring difference between control and experiment groups we can have some confidence that the difference is due to our ‘change under test’
- Often pick low E and ramp up (e.g. 1%, 10%, 25%, 50%)
    - Similar to phased deploy alerting, but measures ‘do users like it’ rather than ‘are there errors’
- Experiment throughput can quickly become limited by traffic volume

A/B test architecture

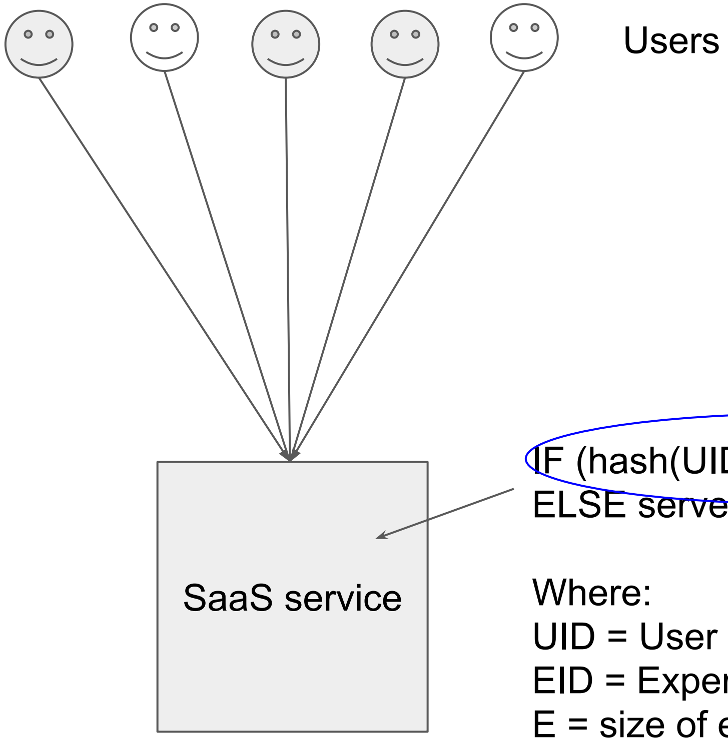
Users



IF (hash(UID.EID) mod 100) < E THEN serve experiment variant
ELSE serve control variant

Where:
UID = User ID
EID = Experiment ID (one per experiment)
E = size of experiment group for experiment EID

A/B test architecture

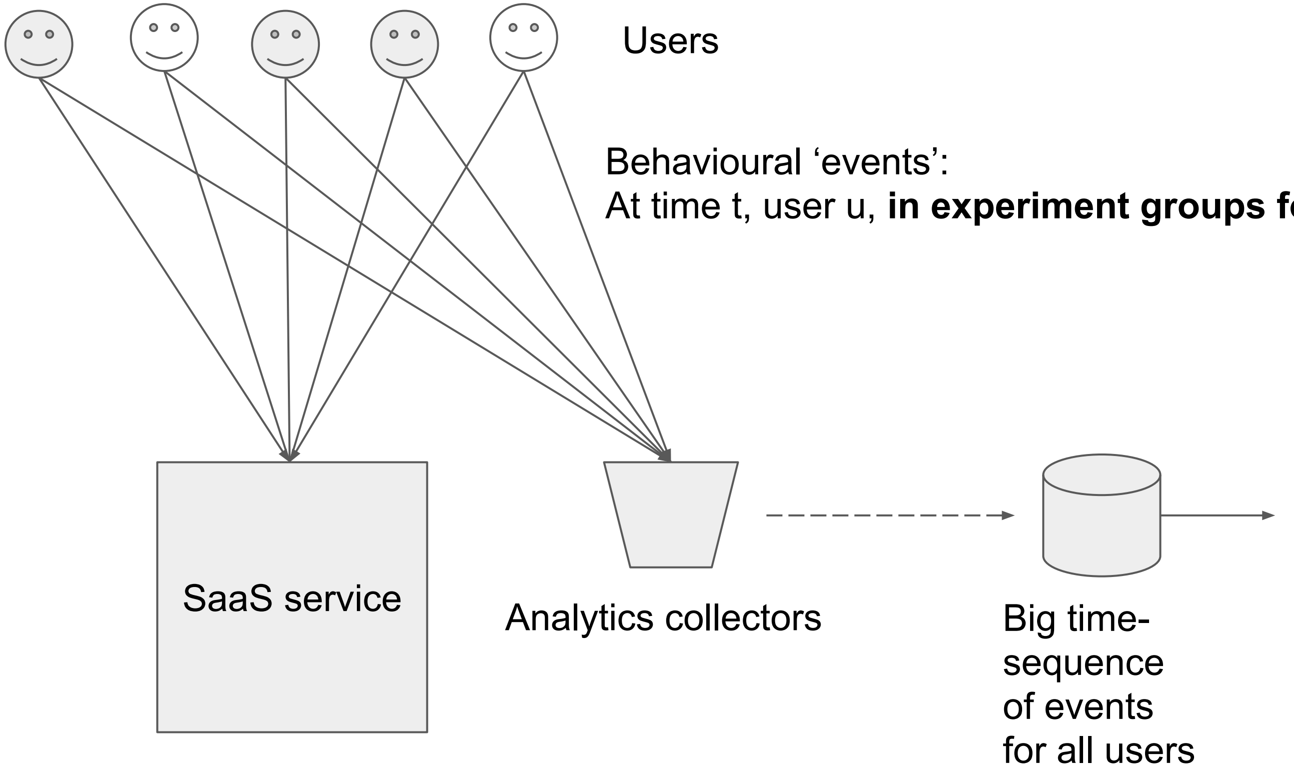


- Users *persistently* in a control or experiment group; don't 'flap'
- Users in existing experiment group remain in experiment group as E increased
- Works for multiple concurrent experiments (but be careful of independence assumptions)

IF (hash(UID.EID) mod 100) < E THEN serve experiment variant
ELSE serve control variant

Where:
UID = User ID
EID = Experiment ID (one per experiment)
E = size of experiment group for experiment EID

A/B test architecture



For each experiment, e, generate reports for metrics of interest segmented by (i) 'in EID_e'; and (ii) 'not in EID_e'. Compare these results for each metric and test statistical significance.

Summary

Summary

- Putting the manage/deploy/upgrade cycle into the software company is a profound change with far-reaching consequences:
    - Economically:
        - Reduces customer TCO and barriers to purchasing
        - Leads to better specialisation, and less duplication; creates new business models
    - Operationally:
        - Enables new ways of doing QA, which changes the economics of testing
        - Phased releases (which can take place over days if required, with flexibility to pause and fix at any time); live monitoring/alerting
    - Enables building of higher quality software through increased visibility of user behavior. (N.B. with great power comes great responsibility!)
        - Behavioural analytics
        - Experiments