Fighting the Good Fight at the Hot Gates of Microservices

10th May 2016

Adrian Trenaman, SVP Engineering Gilt Tech / HBC
Microservices Day London
@adrian_trenaman ade@gilt.com

~300

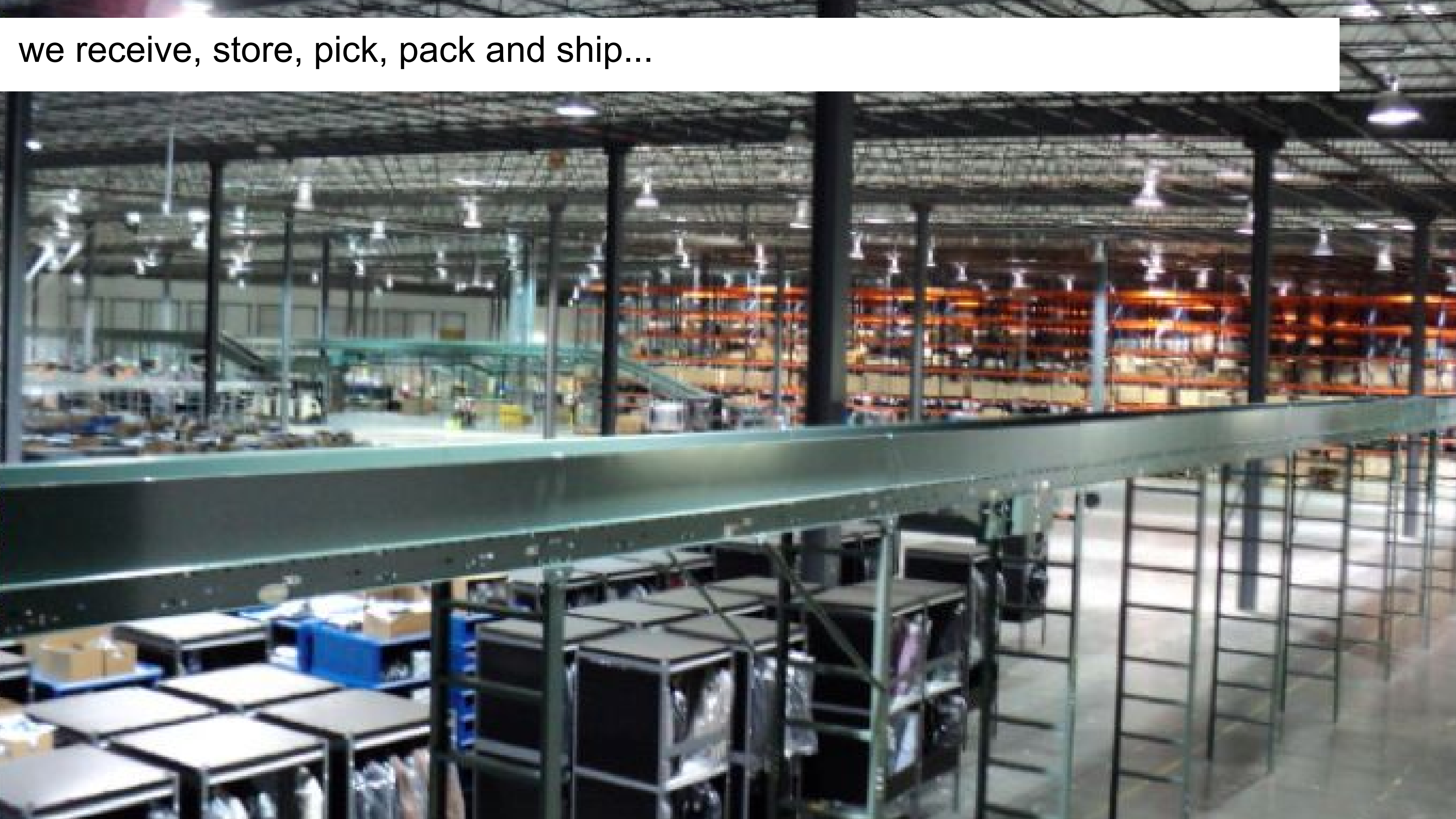
… the number of micro services running gilt.com.

But what about the Persian horde?

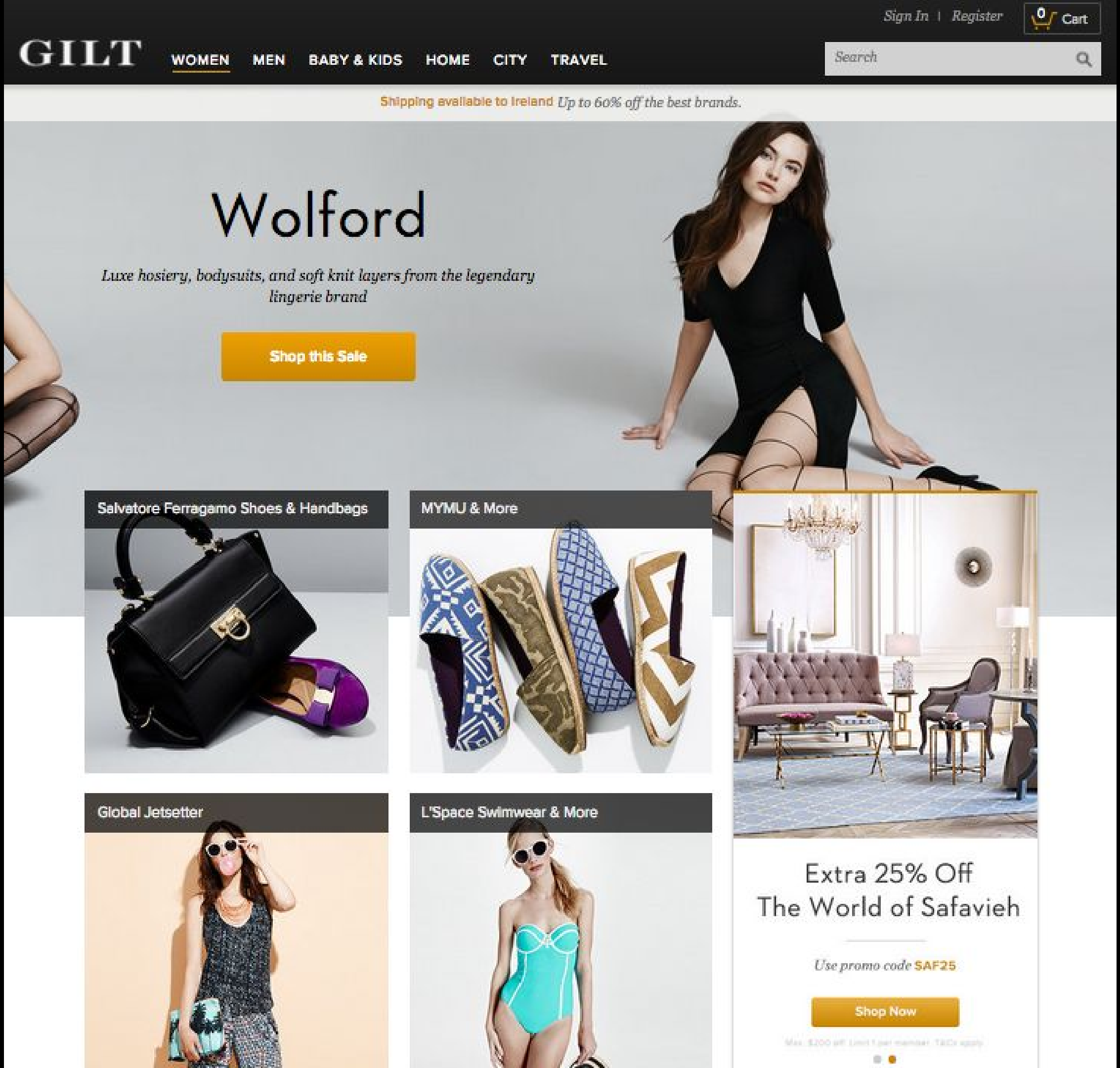
Gilt: luxury designer brands at discounted prices

we shoot the product in our studios

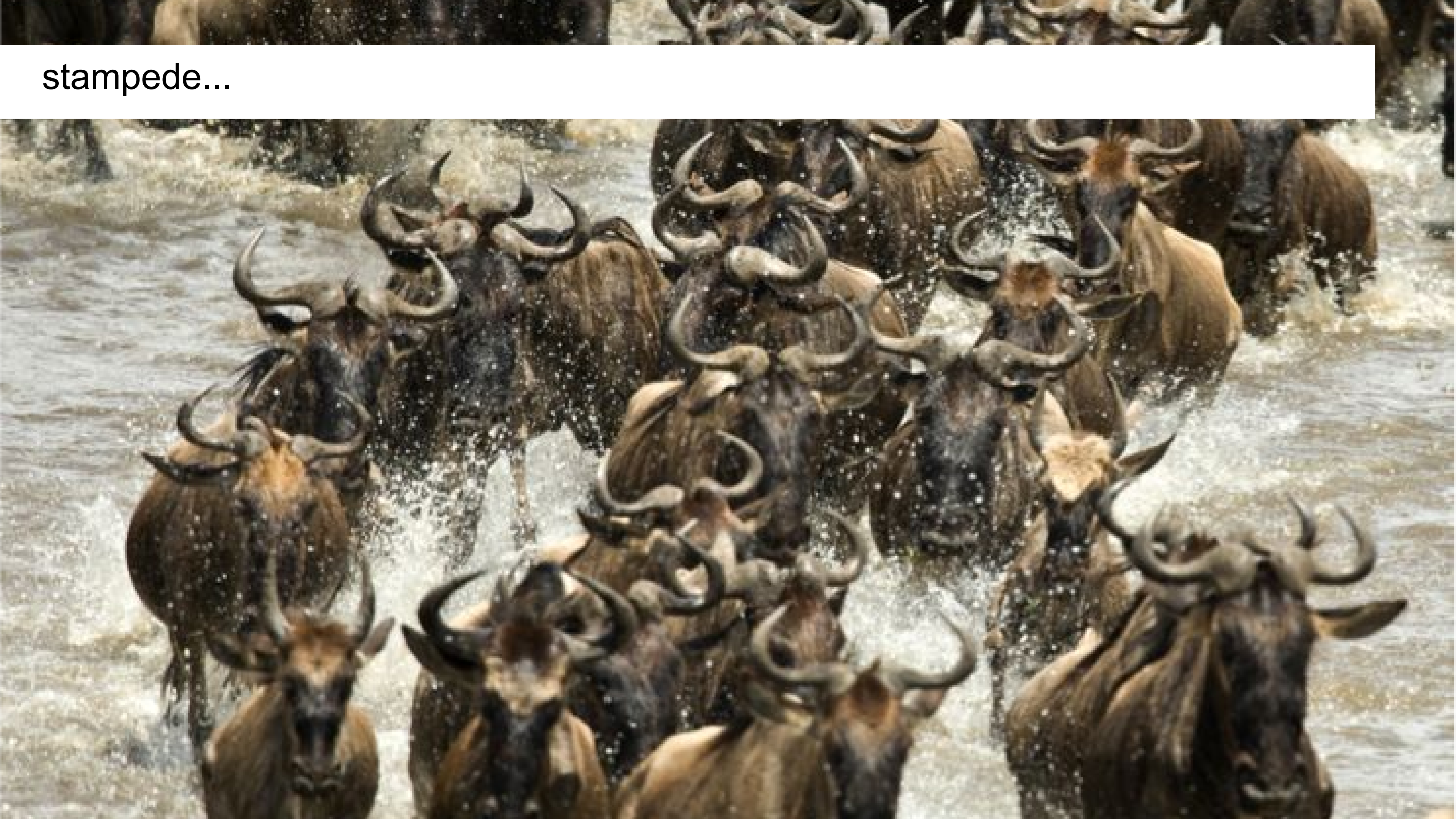
we receive, store, pick, pack and ship...

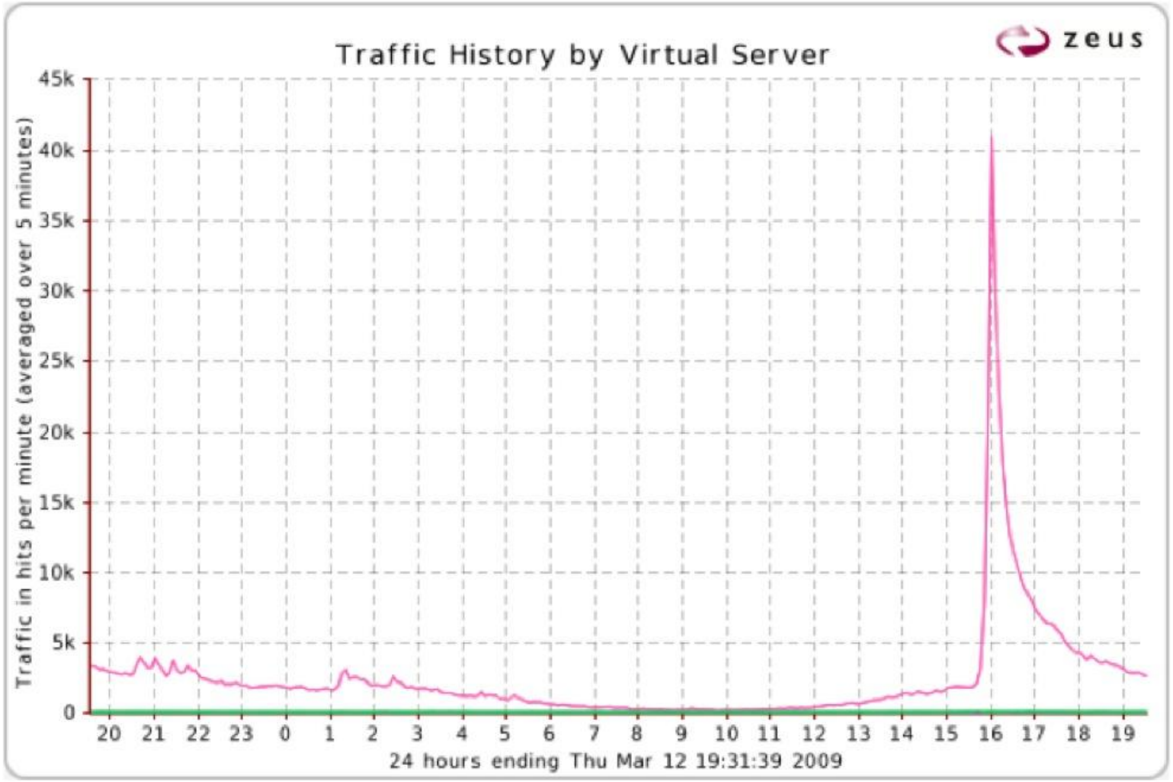


we sell every day at noon...


Sign In | Register
0 Cart
GILT
WOMEN
MEN
BABY & KIDS
HOME
CITY
TRAVEL
Search
Shipping available to Ireland Up to 60% off the best brands.
Wolford
Luxe hosiery, bodysuits, and soft knit layers from the legendary lingerie brand
Shop this Sale
Salvatore Ferragamo Shoes & Handbags
MYMU & More
Global Jetsetter
L'Space Swimwear & More
Extra 25% Off The World of Safavieh
Use promo code SAF25
Shop Now

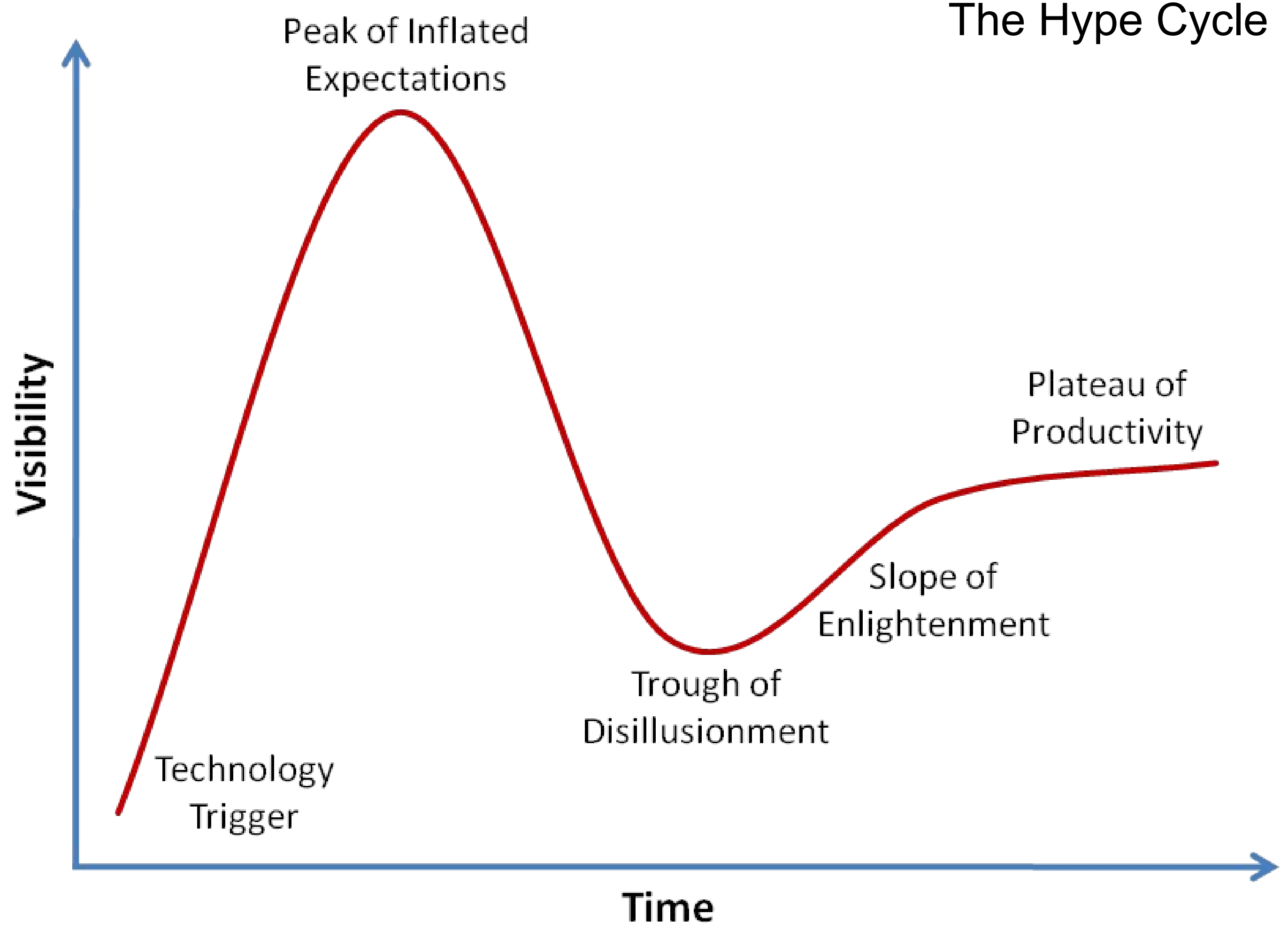
stampede...



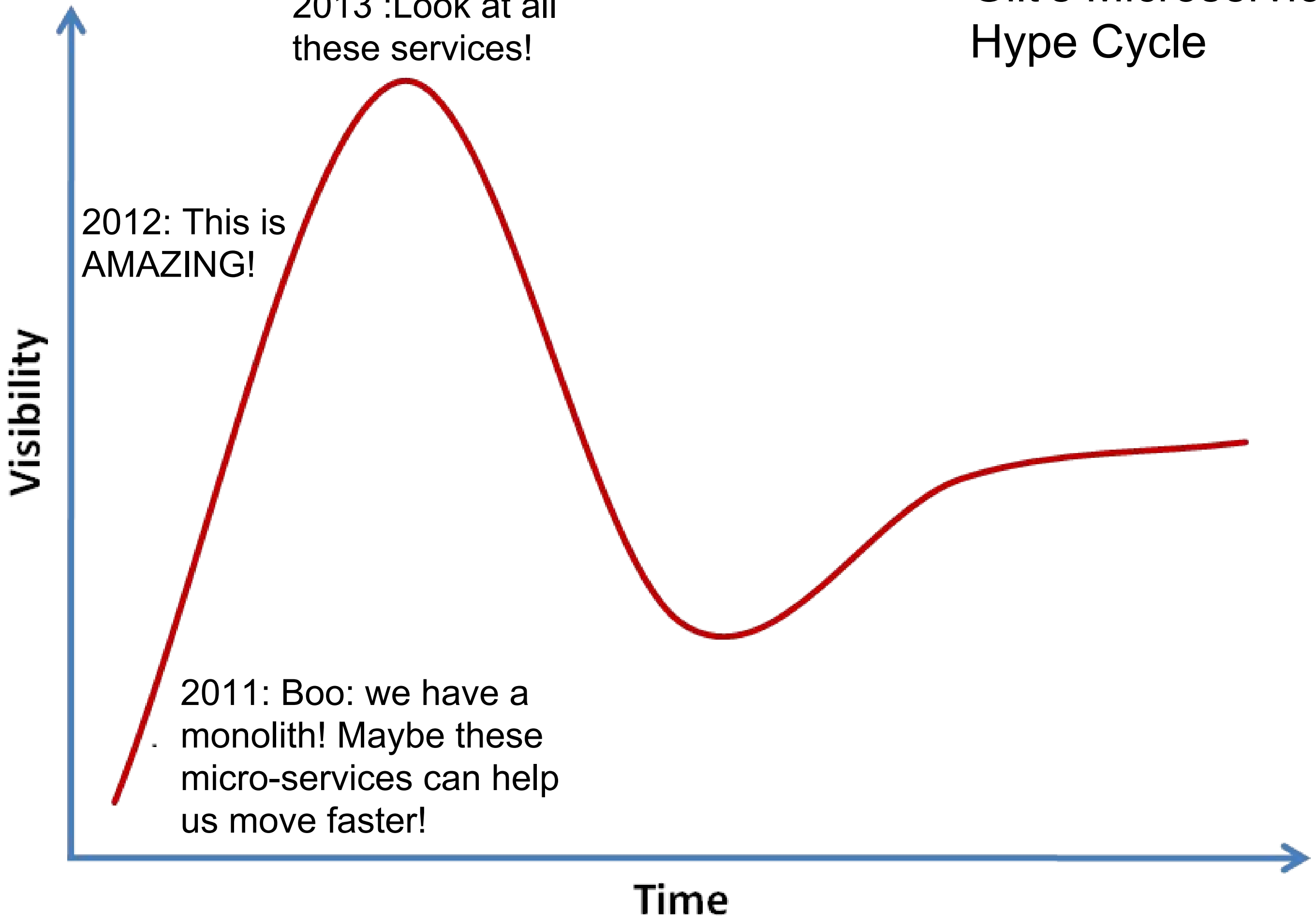


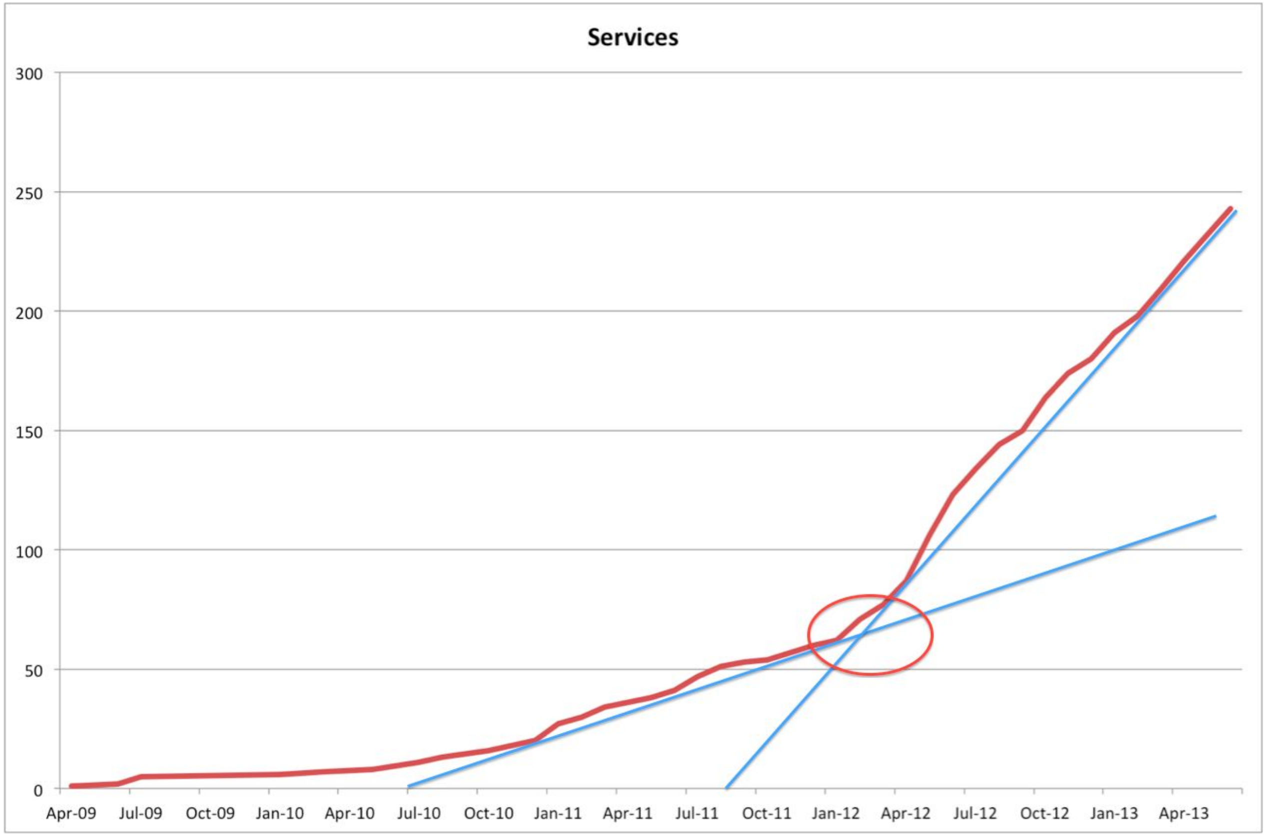
this is what the stampede really looks like...

The Hype Cycle



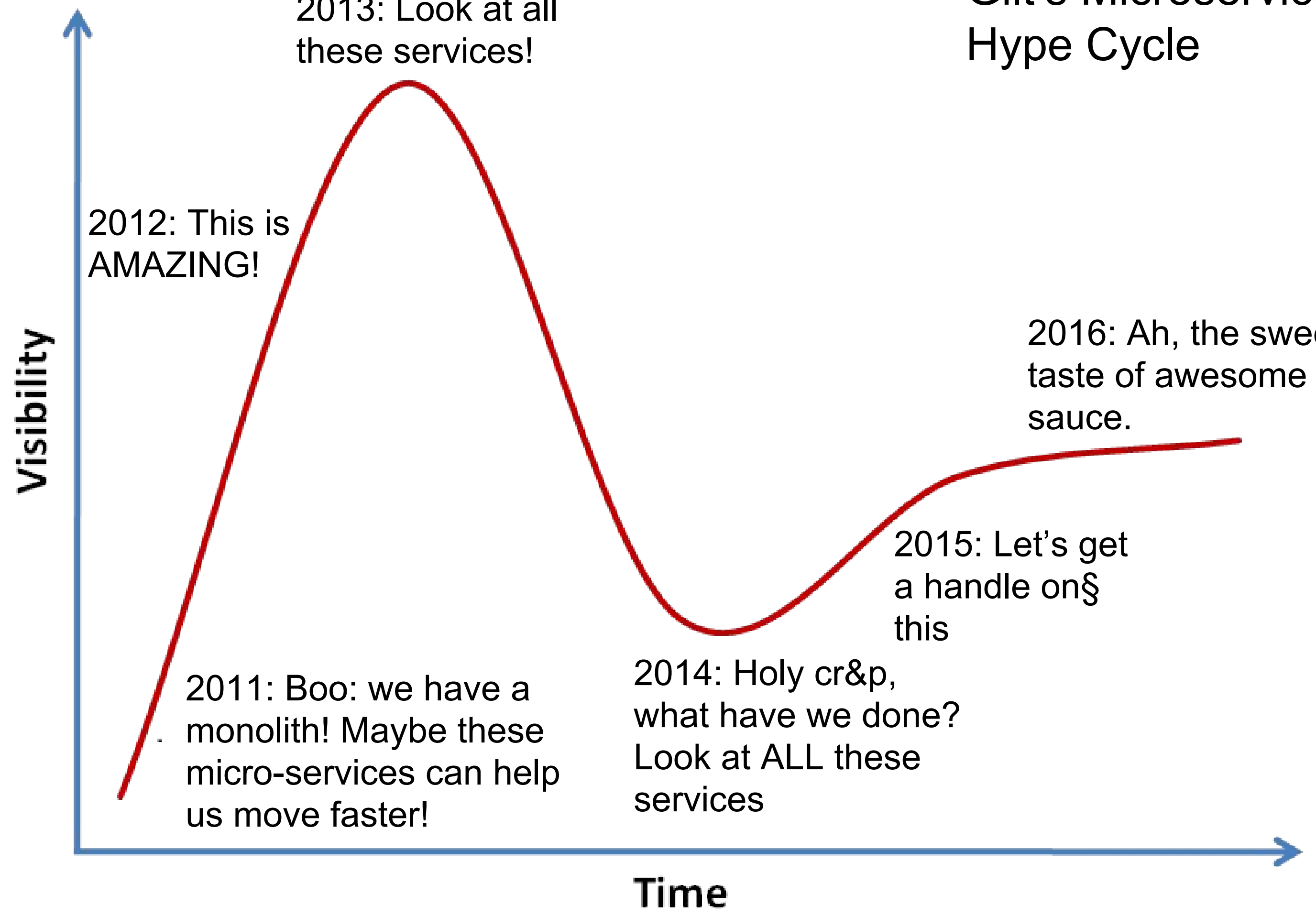
Gilt’s Microservice Hype Cycle

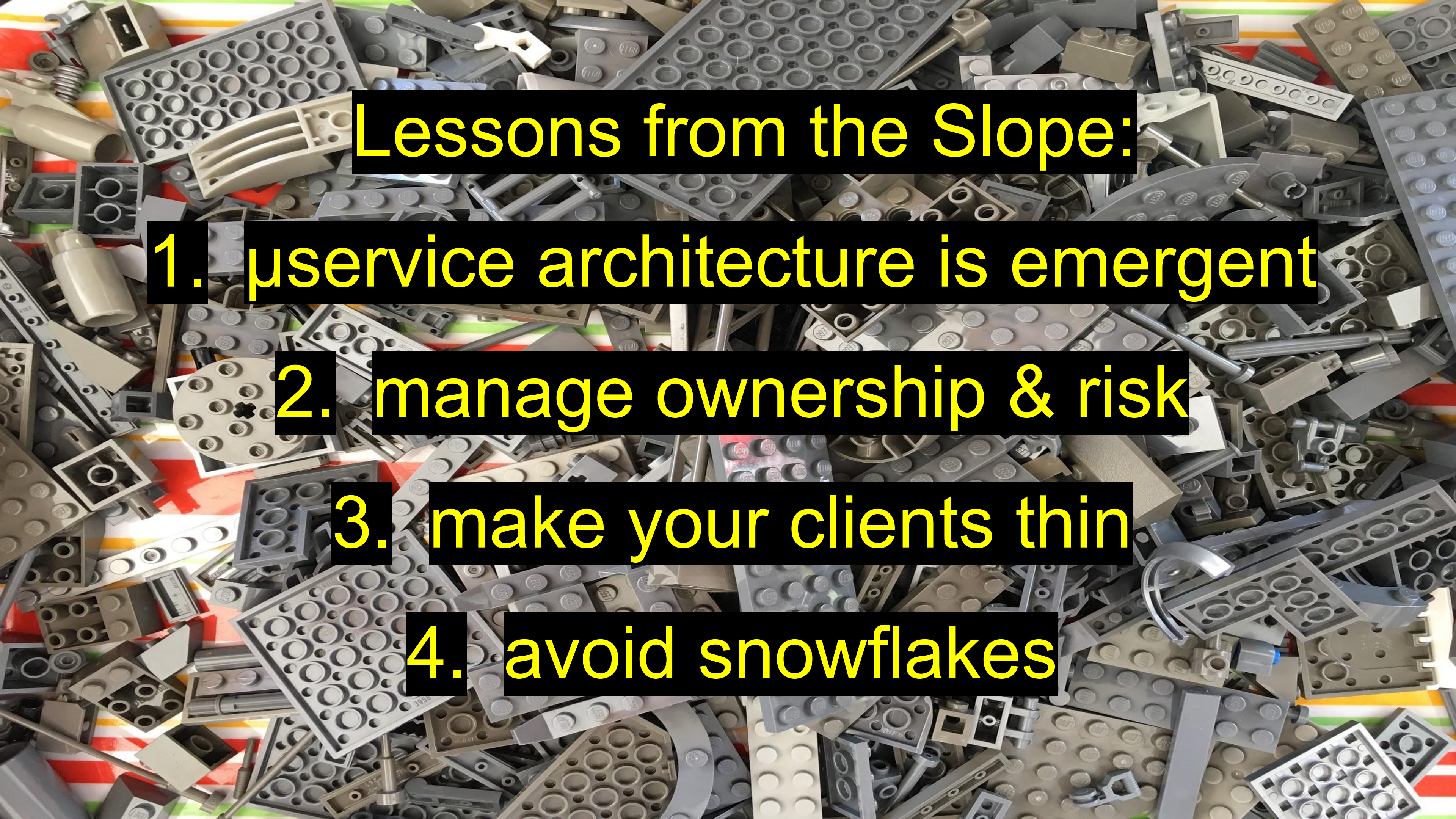




service growth over time: point of inflexion === scala.

Gilt's Microservice Hype Cycle





Lessons from the Slope:

1. µservice architecture is emergent
2. manage ownership & risk
3. make your clients thin
4. avoid snowflakes

emergent architecture

It’s *hard* to think of architecture in one dimension.

n = 265, where *n* is the number of services.

Google Sheets

… we used a “spread sheet”.
‘The Gilt Genome Project’

It’s *hard* to think of architecture in one dimension.

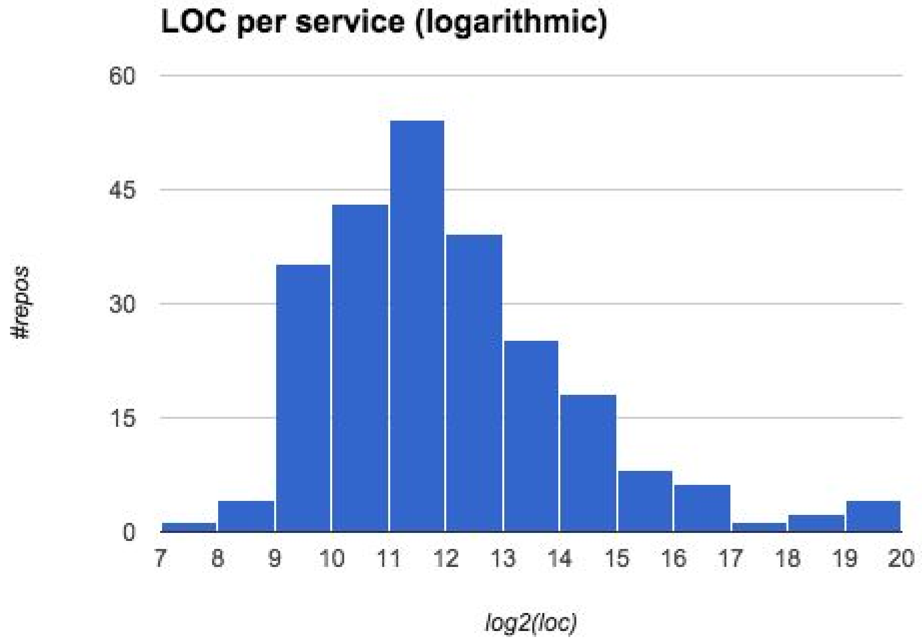
n = 265, where *n* is the number of services.

We added ‘Functional Area’, ‘System’ and ‘Subsystem’ columns to Gilt Genome; provides a strong (although subjective) taxonomy.

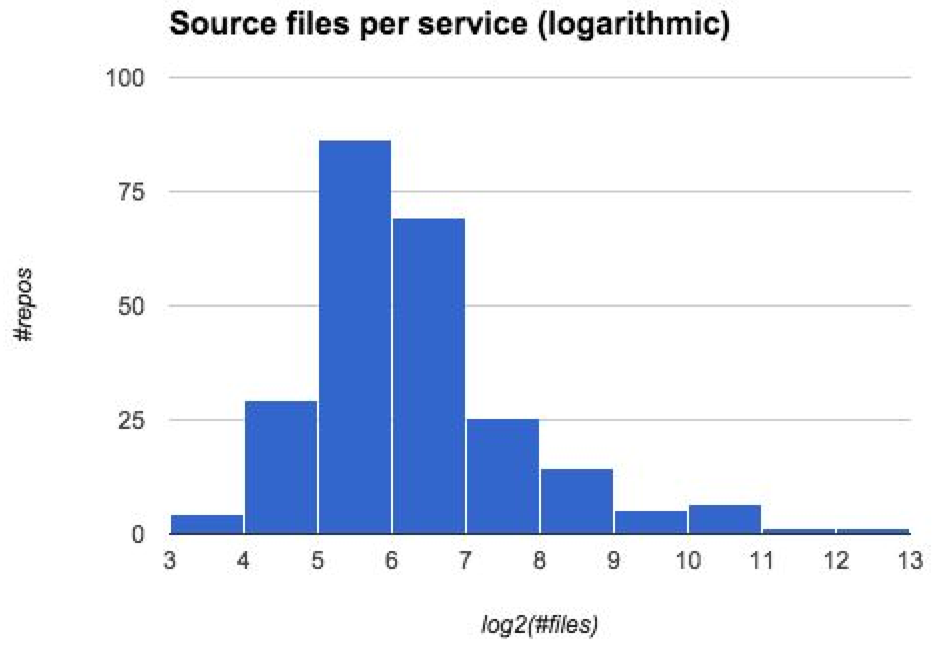
It turns out we have an elegant, *emergent architecture*.

Some services / components are *deceptively simple*.

Others are *simply deceptive*, and require knowledge of their surrounding ‘constellation’

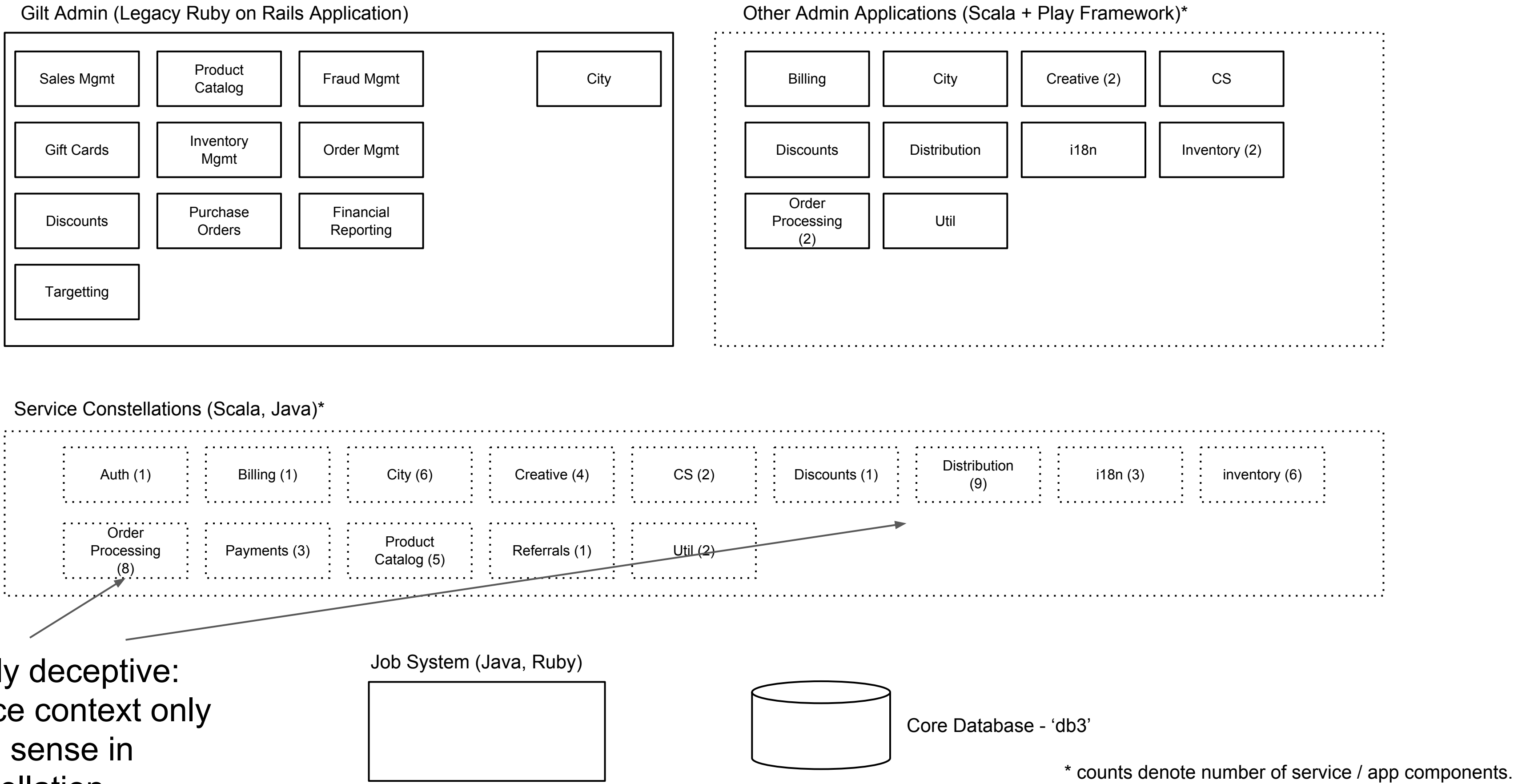


Deceptively Simple - many services are small; < 2048 loc



Deceptively Simple - many services are small, < 32 files.

Gilt Logical Architecture - Back Office Systems



Simply deceptive: service context only make sense in constellation.

Emergent Architecture:

Using the three-level taxonomy approach, we’ve been able to get a better understanding of an emergent architecture, at a *department* level, and where the complexity lies.

We’ve also concluded that the *department* is the right level of granularity for consensus on technical decisions (language, framework, …)

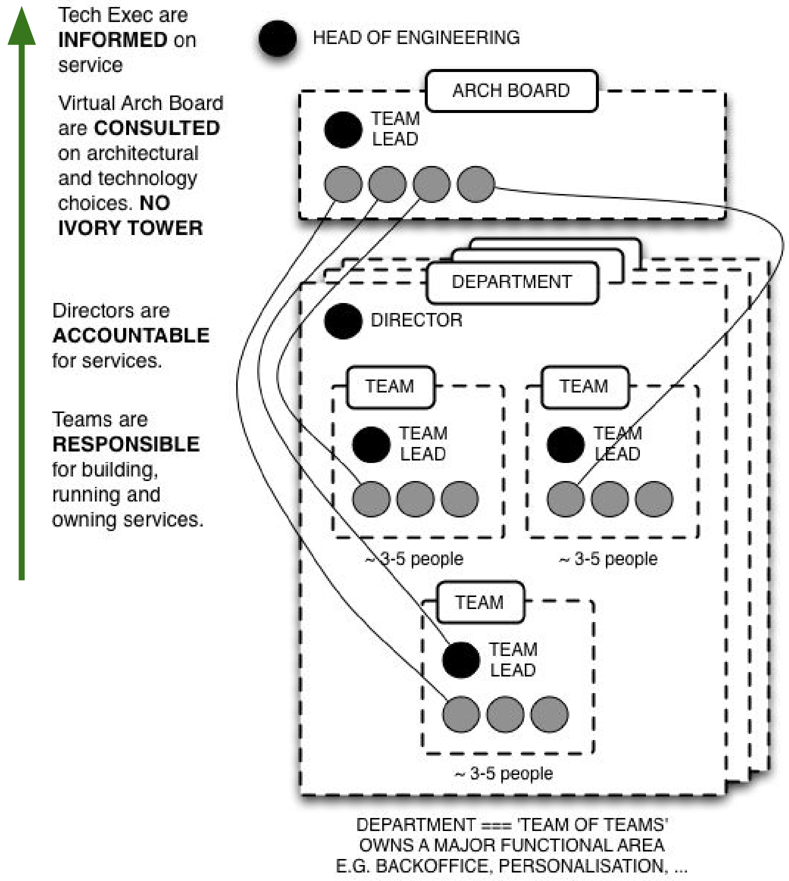
Gilt’s Architecture Board set’s the overall *standards* that teams must follow when interacting across departmental boundaries. HTTP. REST. DNS. AWS.

ownership

1. Software is owned by departments, tracked in ‘genome project’. Directors assign services to teams.

2. Teams are responsible for building & running their services; directors are accountable for their overall estate.

bottom-up ownership, RACI-style



Tech Exec are **INFORMED** on service

Virtual Arch Board are **CONSULTED** on architectural and technology choices. **NO IVORY TOWER**

Directors are **ACCOUNTABLE** for services.

Teams are **RESPONSIBLE** for building, running and owning services.

DEPARTMENT === 'TEAM OF TEAMS' OWNS A MAJOR FUNCTIONAL AREA E.G. BACKOFFICE, PERSONALISATION, ...

Notes:

Zero Power, High Influence: The Architecture Board https://github.com/gilt/arch-board

Gilt Standards and Recommendations: https://github.com/gilt/standards

5 ± 2

The perfect size for a team

20 ± 4

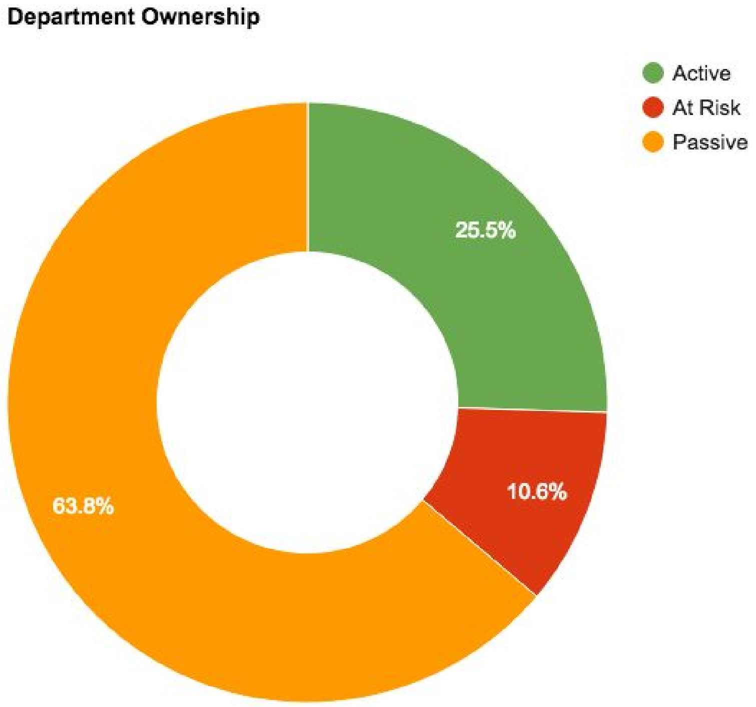
The perfect size for a ‘department’ (team of teams)

30%

Amount of time a department should spend on operations / maintenance / red-hot.
We build the notion of SRE (Site Reliability Engineering) *into* the team.

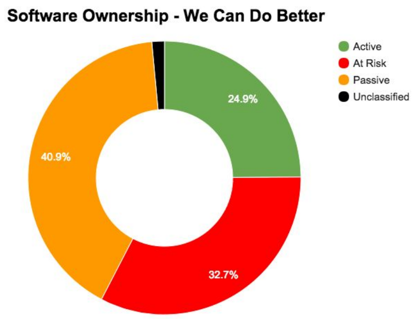
We classify ownership as: active, passive, at-risk.

‘done’ === 0% ‘at risk’

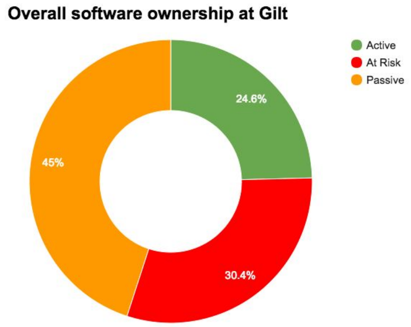


‘ownership donut’ informs tech strategy

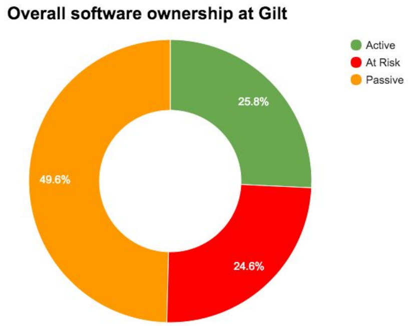
Jul 2015



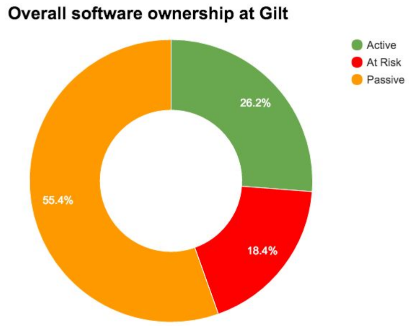
Sep 2015



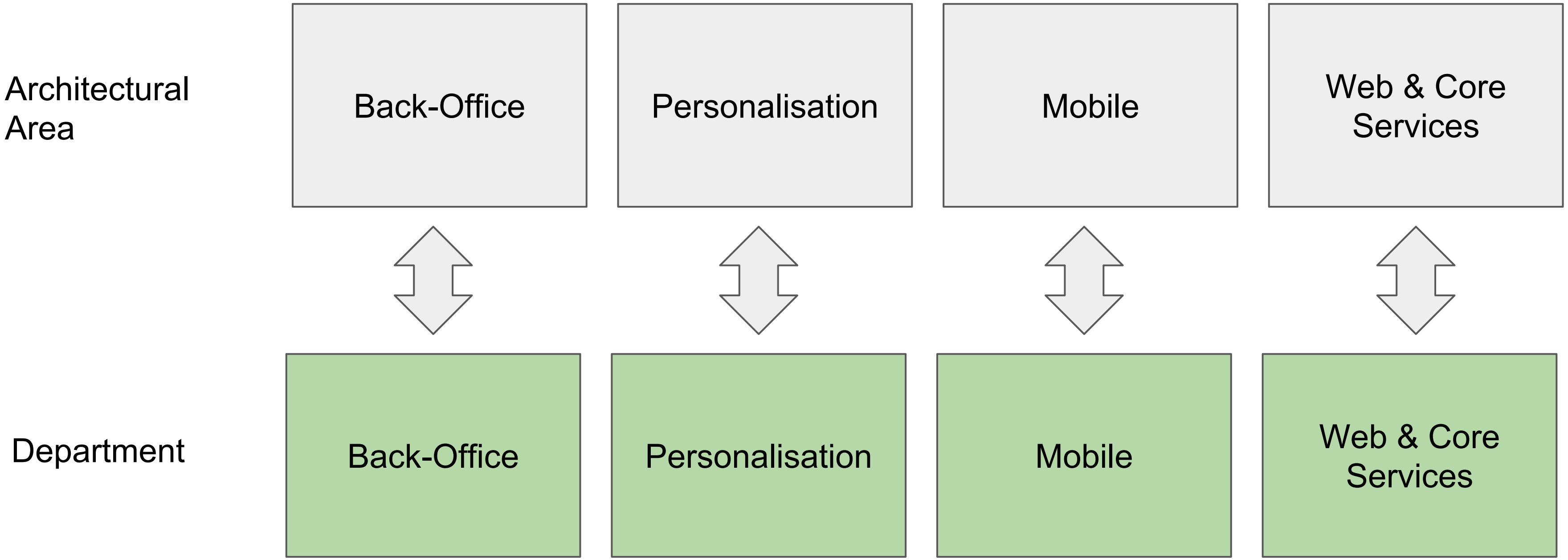
Oct 2015



Feb 2015

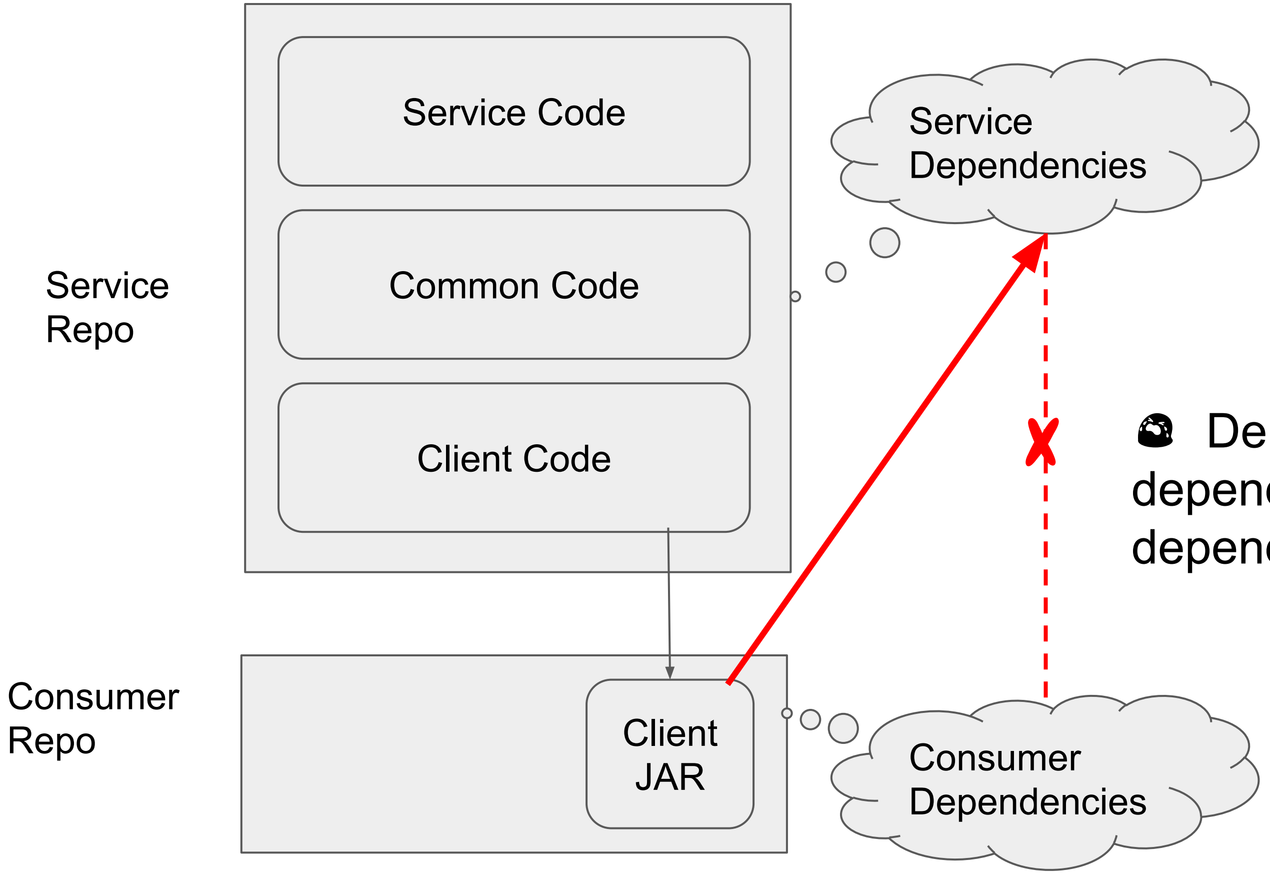


Getting a handle on ownership...



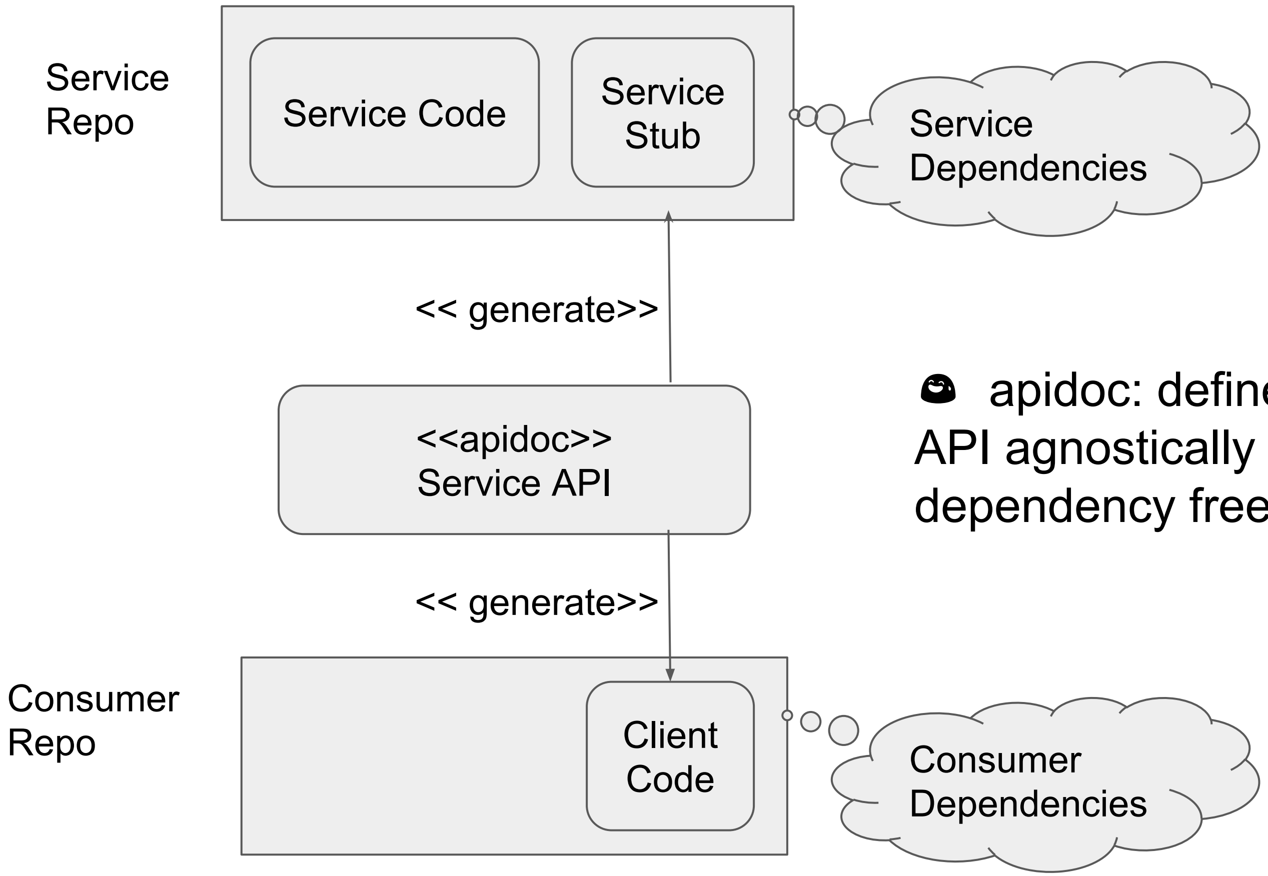
Emergent Architecture + Ownership Oriented Org:
"You just pulled an inverse Conway manoeuvre"

thin clients



😰 Dependency hell as client JAR dependencies conflicts with service dependencies.

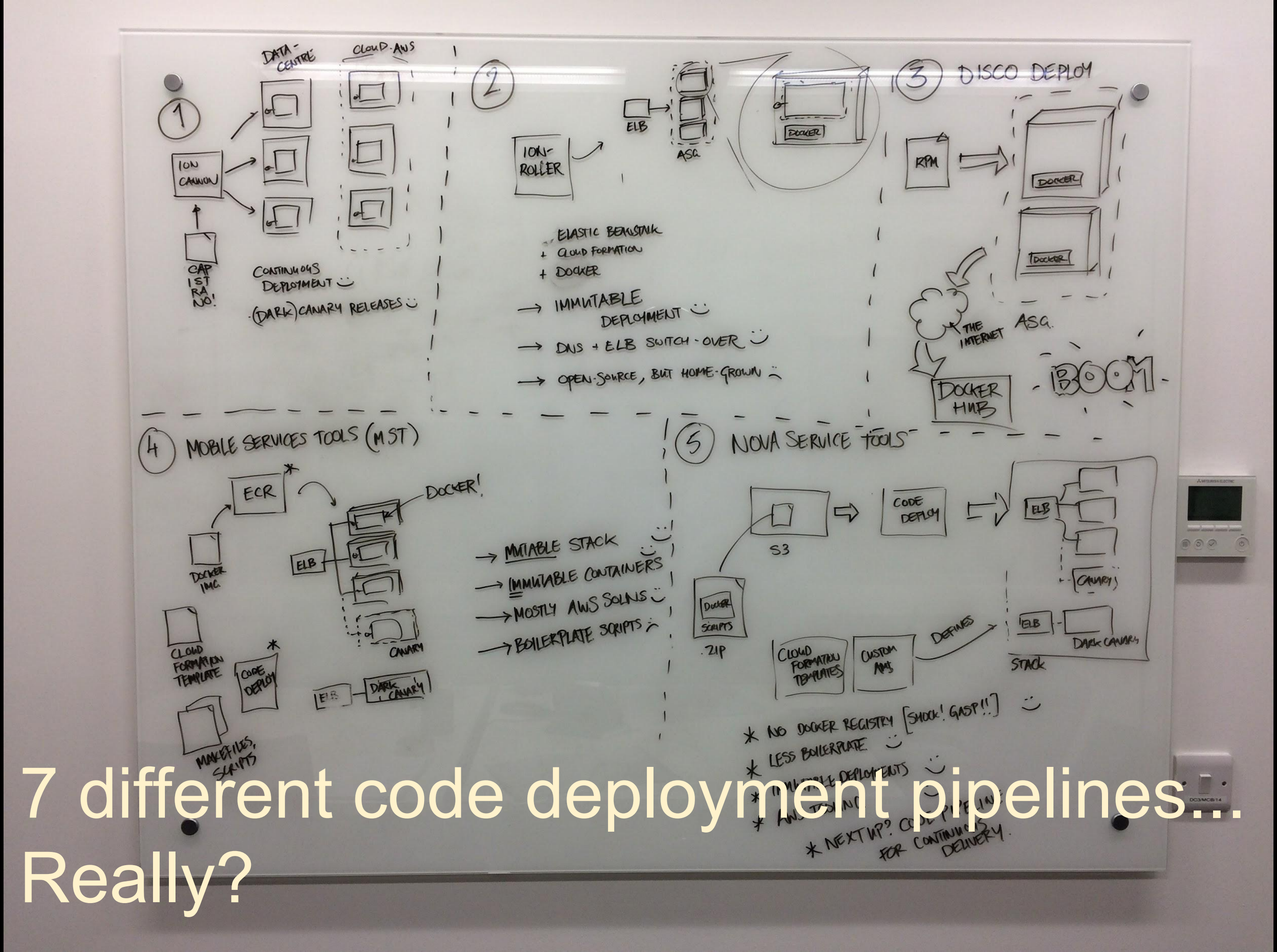
Take as *few* code dependencies as possible. This stuff HURTS when n ~= 300.



apidoc: define RESTful service API agnostically and generate dependency free, thin clients.

This is way easier. http://apidoc.me

stop building snowflakes



7 different code deployment pipelines... Really?

6

Andrey’s Rule of Six:

“We could solve this now, or, just wait six months, and Amazon will provide a solution”

Andrey Kartashov, Distinguished Engineer, Gilt.

Current thinking on deployment:

(1) Re-use as much AWS tooling as possible: Code Pipeline, Code Deploy, Cloud Formation.
(2) Very lightweight tool chain to support dark canaries, canary releases, phased roll-out and roll-back: NOVA

https://github.com/gilt/nova

Summary: pragmatically managing µservices

Make a list of your services.

Classify services: three-level taxonomy

Look for complexity, remove it or document it if it’s inherent.

Classify risk: incorporate risk reduction into tech strategy.

Organise your teams around ownership, but be flexible.

Keep your clients thin and dependency free

Encourage diversity, but prefer same in your technology choices: looks for consistency at granularity of 20±4 people.

Question the code that doesn’t directly grow your business or cut your costs.

Thank you!

ade@gilt.com

@adrian_trenaman
@gilttech