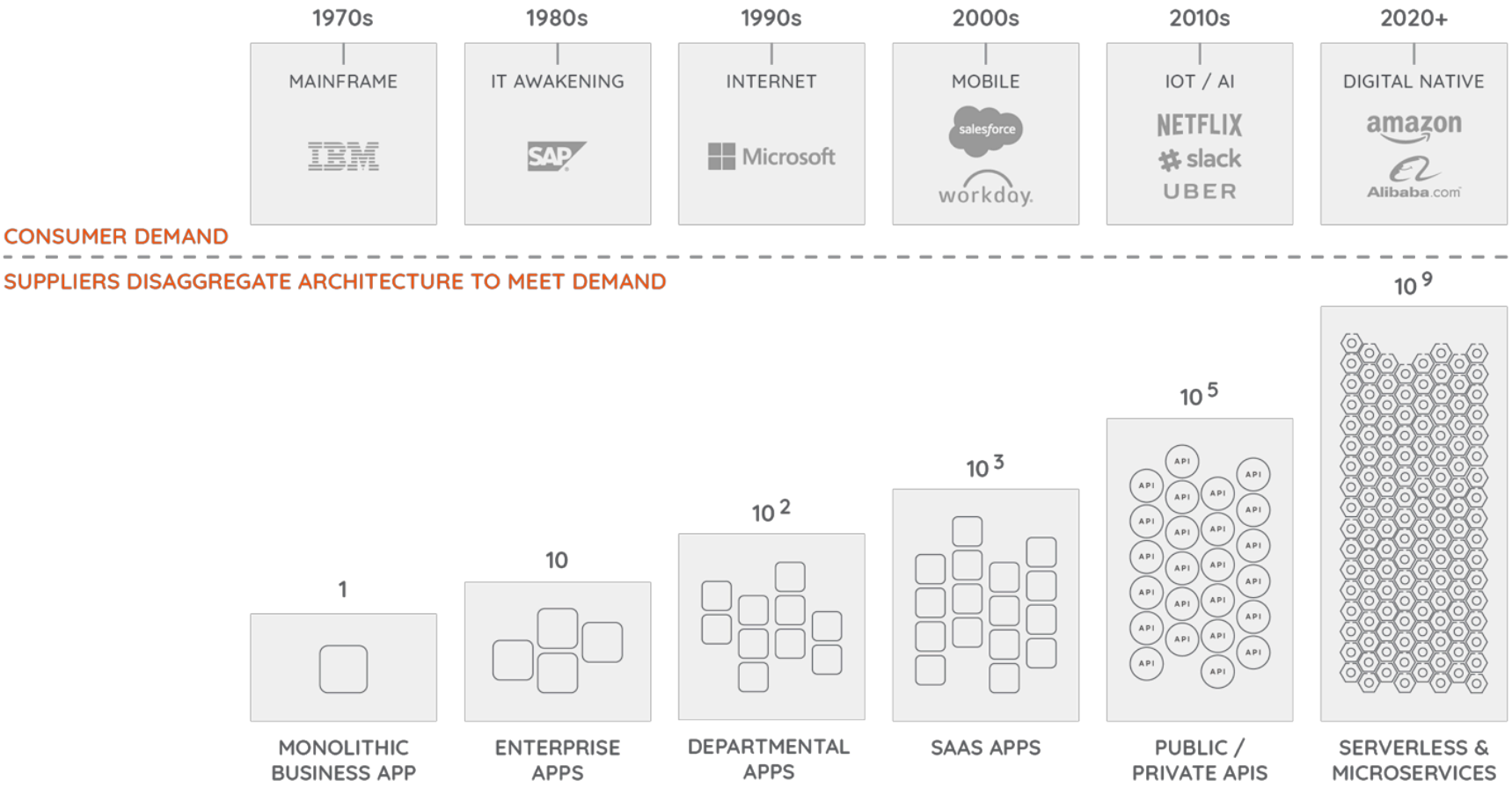
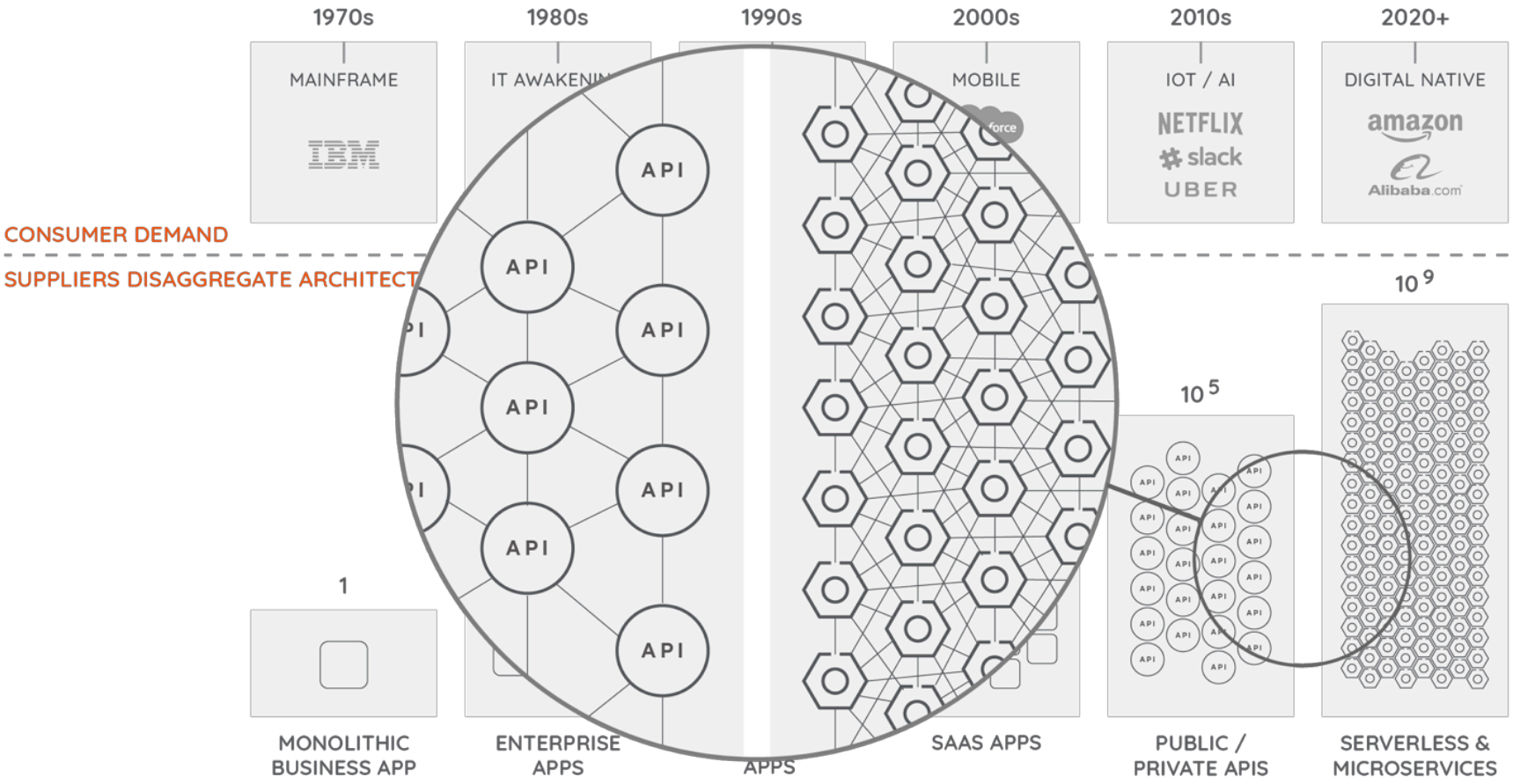
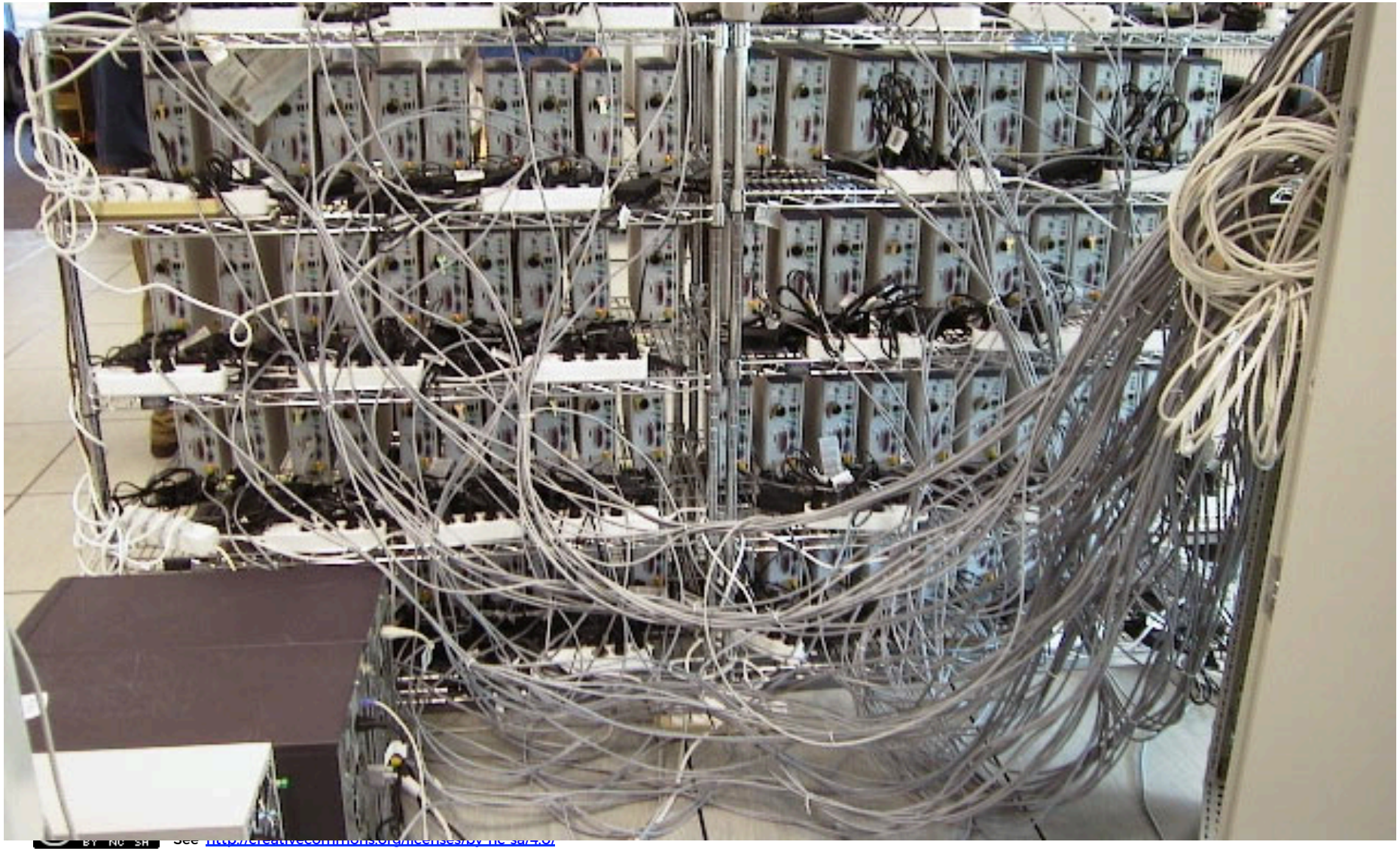
Motivation

Oxford University
Software Engineering
Programme
December 2018





Distributed Computing



Why is Distributed Computing important?

- Scale
- Evolution
- Integration
- Cross-boundary
- Resilience

Learning from

THE AMAZON

TECHNOLOGY

PLATFORM

Many think of Amazon as "that hugely successful online bookstore." You would expect Amazon CTO Werner Vogels to embrace this distinction, but in fact it causes him some concern. "I think it's important to realize that first and foremost Amazon is a technology company," says Vogels. And he's right. Over the past years, Vogels has helped Amazon grow from an online retailer (albeit one of the largest, with more than 55 million active customer accounts) into a platform on which more than 1 million active retail partners worldwide do business. Behind Amazon's successful evolution from retailer to technology platform is its SOA (service-oriented architecture), which broke new technological ground and proved that SOAs can deliver on their promises.

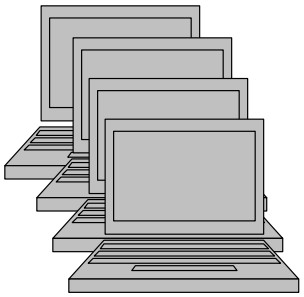
Vogels came to Amazon from Cornell University, where he was working on high-availability systems and the management of scalable enterprise systems. He maintains that research spirit at Amazon, which regularly must solve problems never before encountered. "Maybe other companies call it research. We just call it development," he points out.

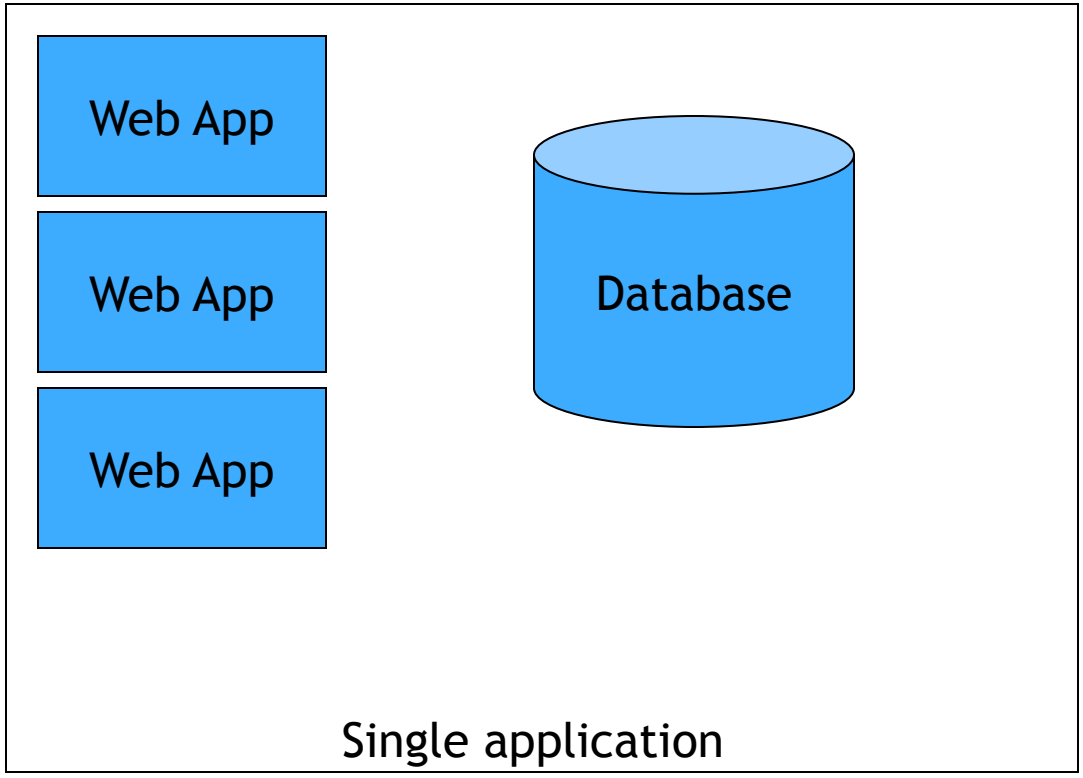
Interviewing Vogels is ACM Turing Award winner and Microsoft Technical Fellow Jim Gray.

application, running on a Web server, talking to a database on the back end. This application, dubbed Obidos, evolved to hold all the business logic, all the display logic, and all the functionality that Amazon eventually became famous for: similarities, recommendations, Listmania, reviews, etc.

Source: Interview with Werner Vogels, ACM Queue https://queue.acm.org/detail.cfm?id=1142065

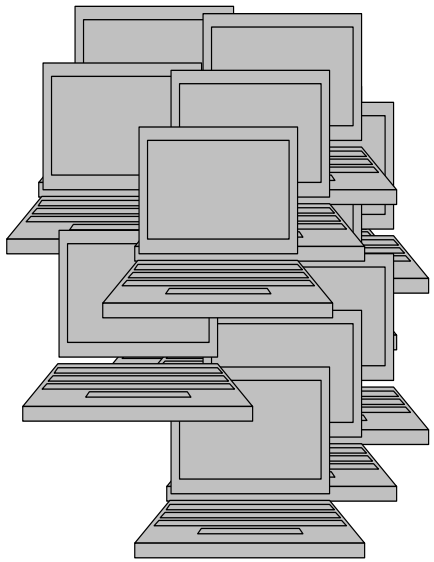
“Obidos”

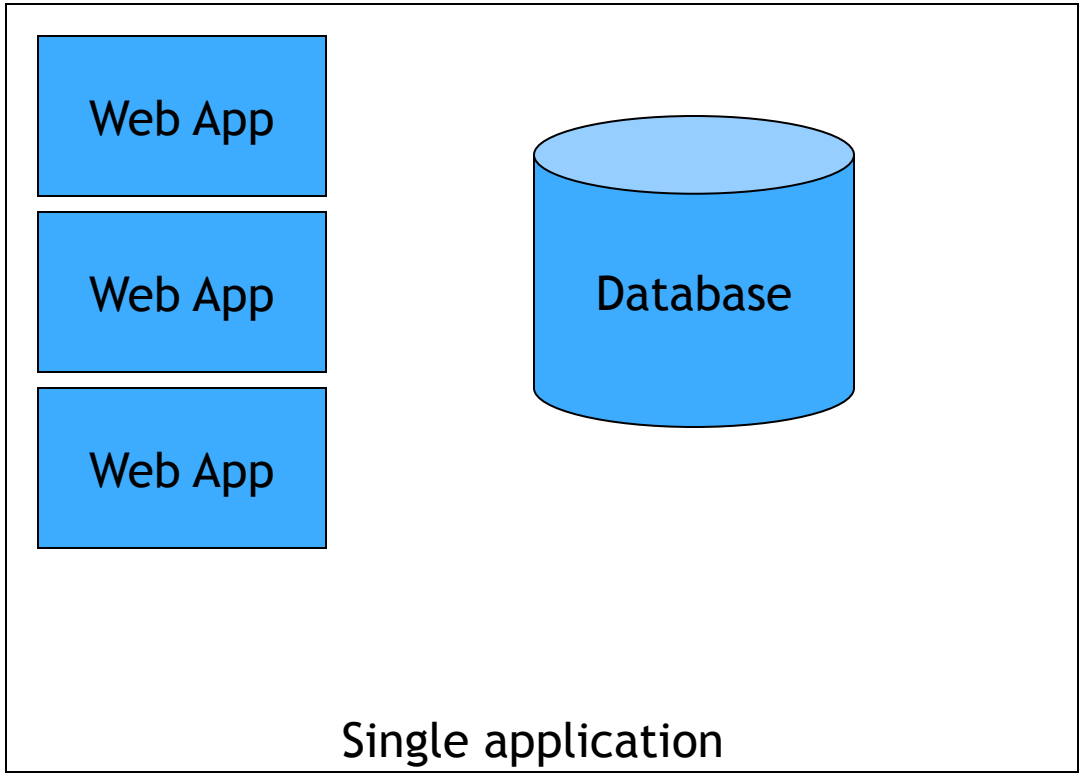




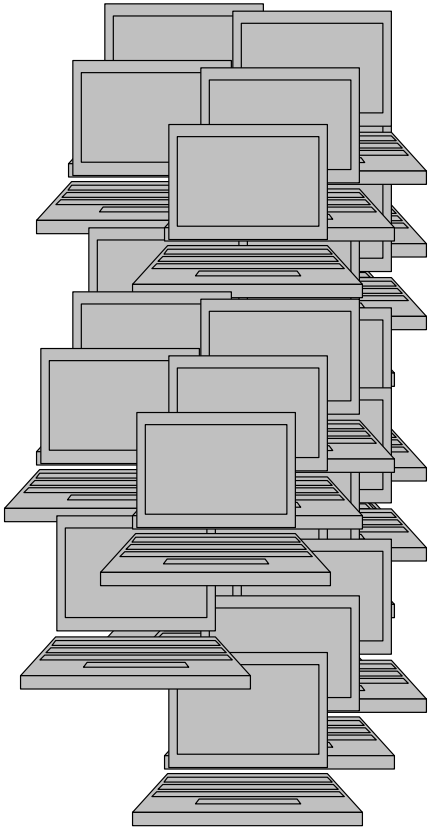
But it was Successful!

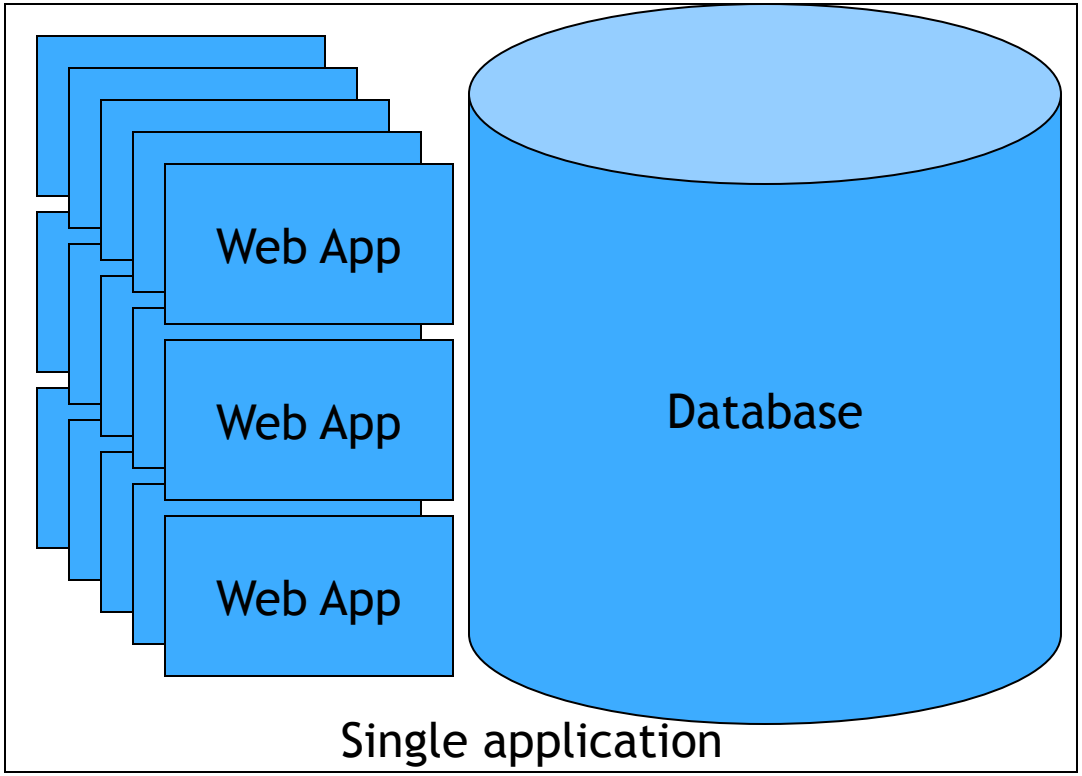
Internet Scale Up





… to bursting point





Problems

- Too many complex pieces of software in a single system
- No evolution possible
- Need to scale independently
  - Parts sharing resources with other unknown code paths
- No isolation
- No clear ***ownership***

Database scaling

- Databases a shared resource
- Hard to scale-out
- Front-end and backend shared by
  - Too many teams
  - Too many processes

A new model

- In 2001 decided on a new approach
- SOA based – even before the term was in common usage
- Encapsulating the data with the business logic that operates on the data
- Only access through a published service interface
- No direct database access is allowed from outside the service
- No data sharing among the services.

Growth

- Amazon services in the hundreds
- A typical visit to the homepage may include calls to 100 services
- Caching reduces the actual network traffic
- Fully distributed, decentralized
- The web servers are just one client into the service fabric

Matched by business growth

- Amazon is supporting many new businesses
- Books, CDs, Electronics, Toys, Tools and Hardware,...
- Plus millions of independent retailers sharing the Amazon platform

Lessons learnt

- Isolation
  - Service Orientation promotes ownership and control
- Scalability
  - By preventing direct database access, can scale the services without affecting clients
- Need a common service-access mechanism
  - Aggregation
  - Routing
  - Tracking

Organization

- "Each service has a team associated with it, and that team is completely responsible for the service—from scoping out the functionality, to architecting it, to building it, and operating it… ***You build it, you run it***" Werner Vogels, CTO, Amazon

  - Promotes Customer Focus and Innovation
  - Gives developers direct access to customers
  - And experience of how their code performs

Why did SOA evolve?

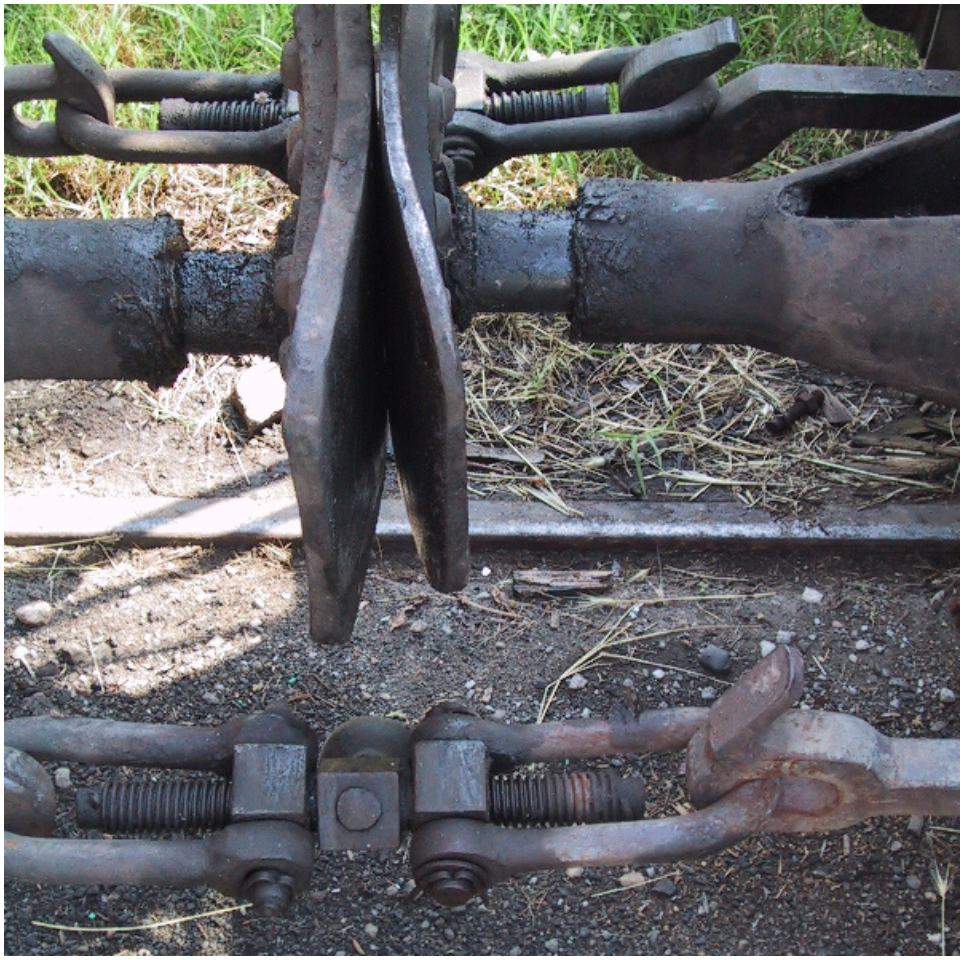
- Directly came out of XML
  - Understanding the schema and structure of messages
  - Especially within the “fabric” not just at the endpoints
- What’s different?
  - Metadata and third-party usage
  - Policies, Governance, etc
  - Inherent Security

Service

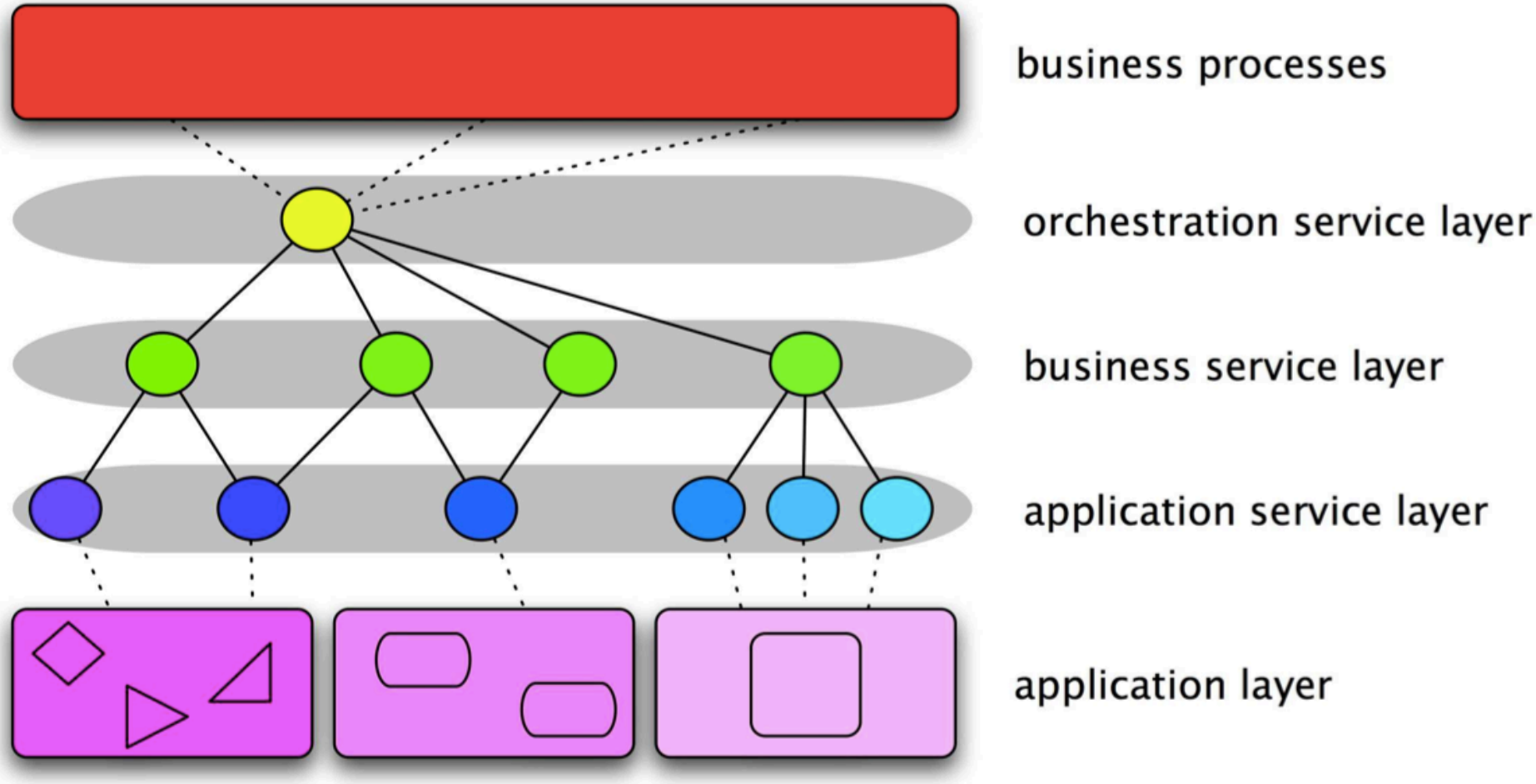
Tightly coupled

- Tightly coupled systems have significant problems:
  - Errors, delays and downtime spread through the system
  - The resilience of the whole system is based on the weakest part
  - Cost of upgrading or migrating spreads
  - Hard to evaluate the useful parts from the dead weight

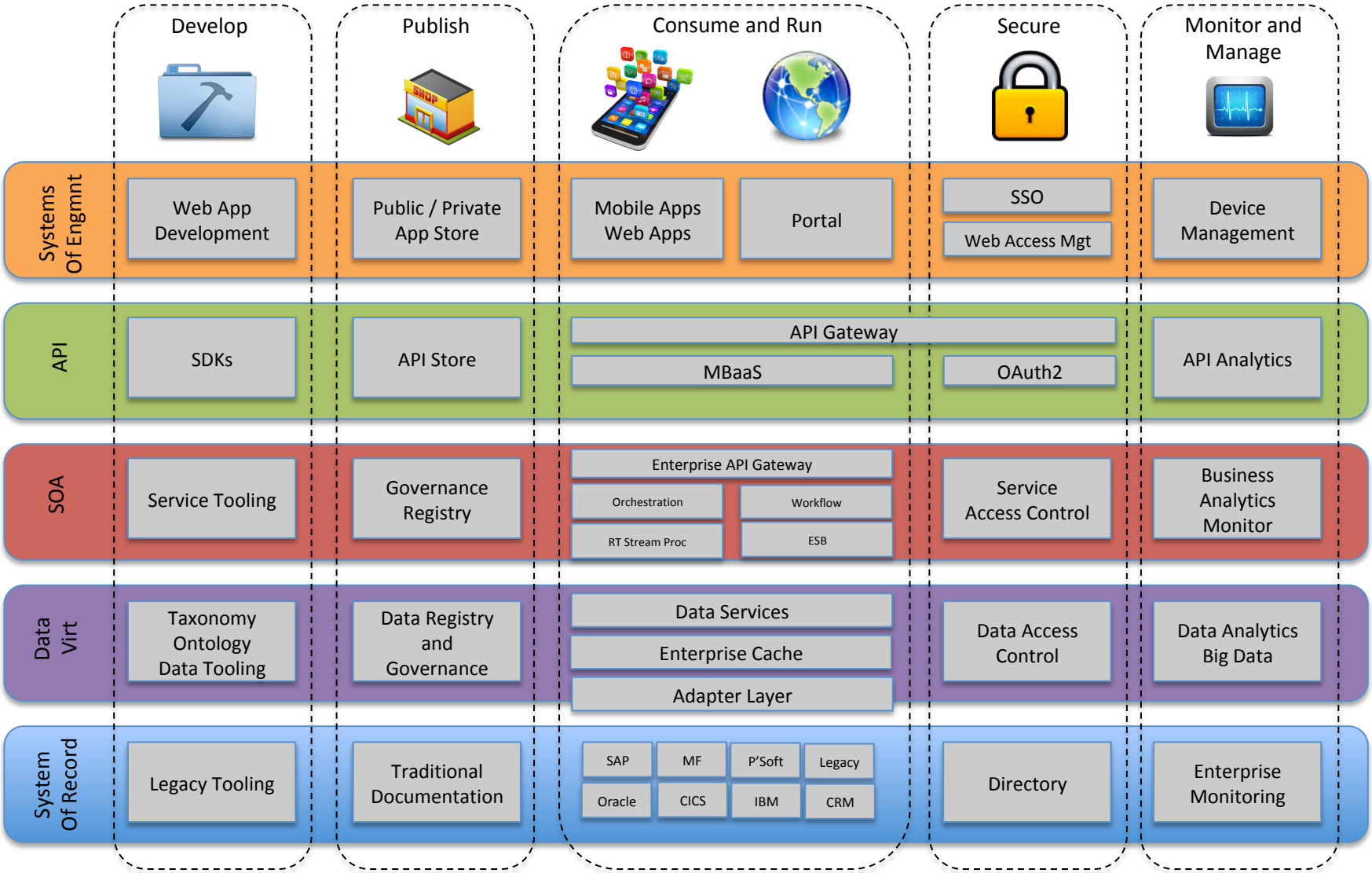
Loose coupling



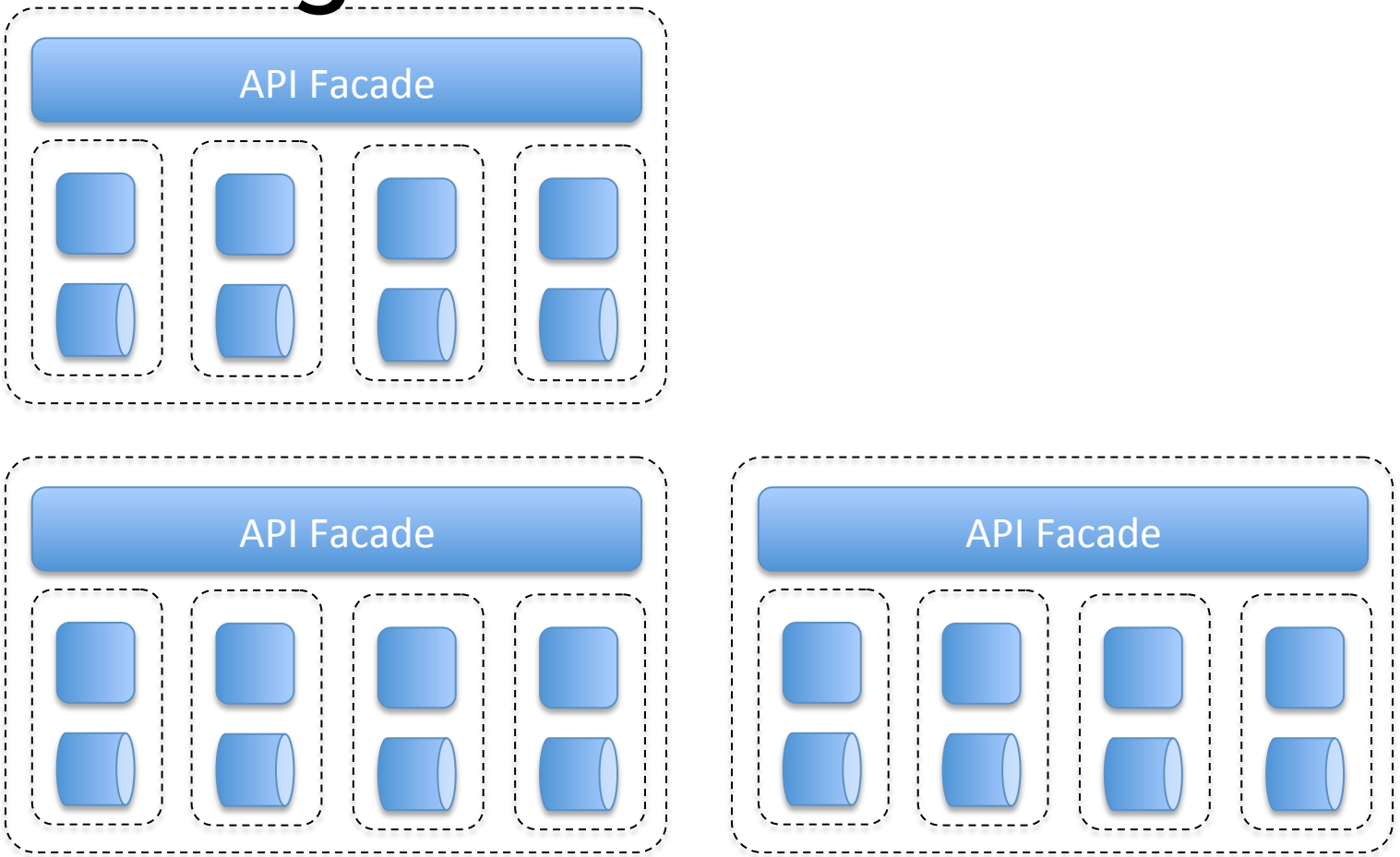
Traditional SOA = Layered Architecture

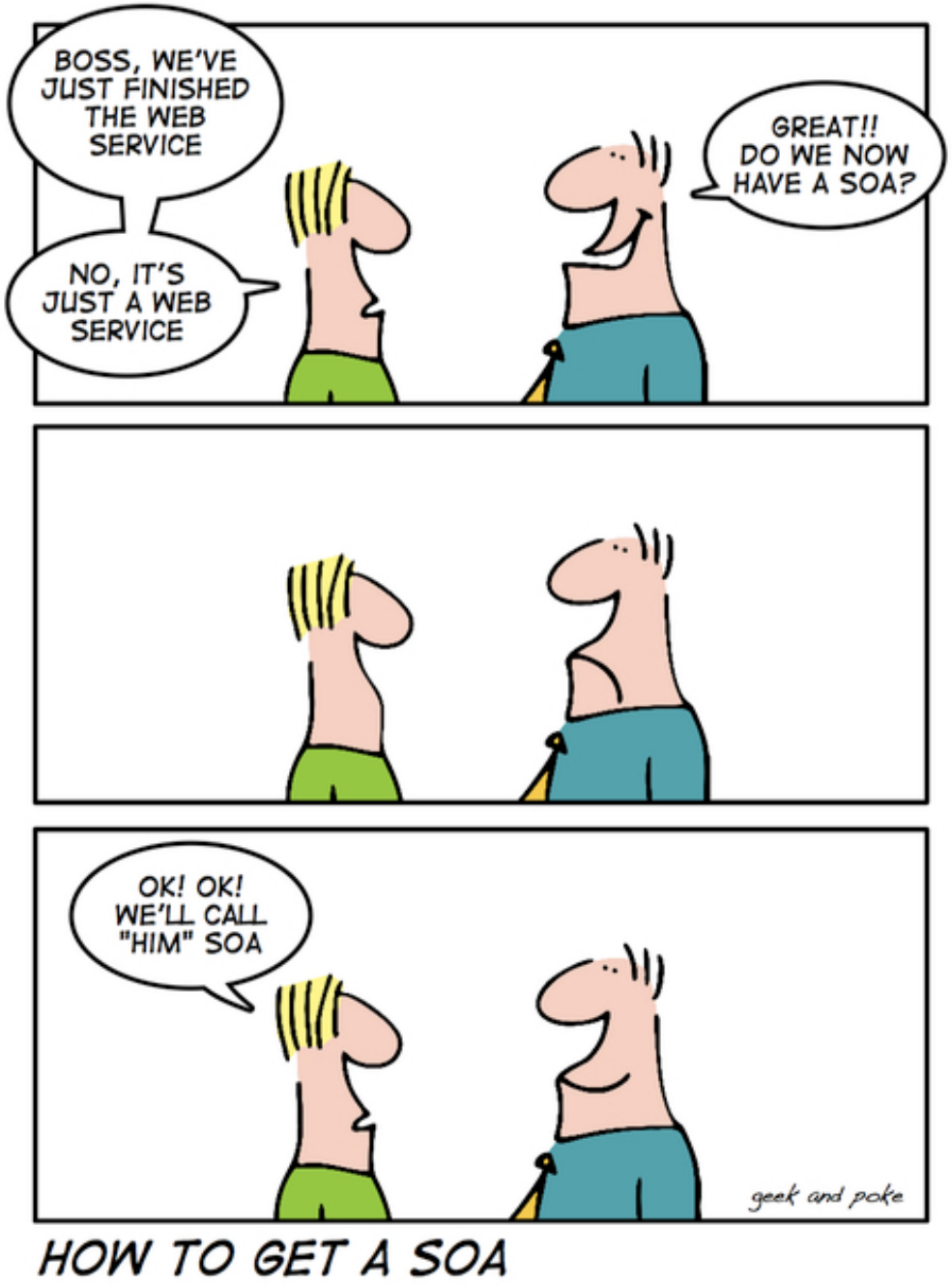


End-to-End Architecture and Functional Requirements for a Connected Business



Micro and Macro Services Segmented Architecture





HOW TO GET A SOA

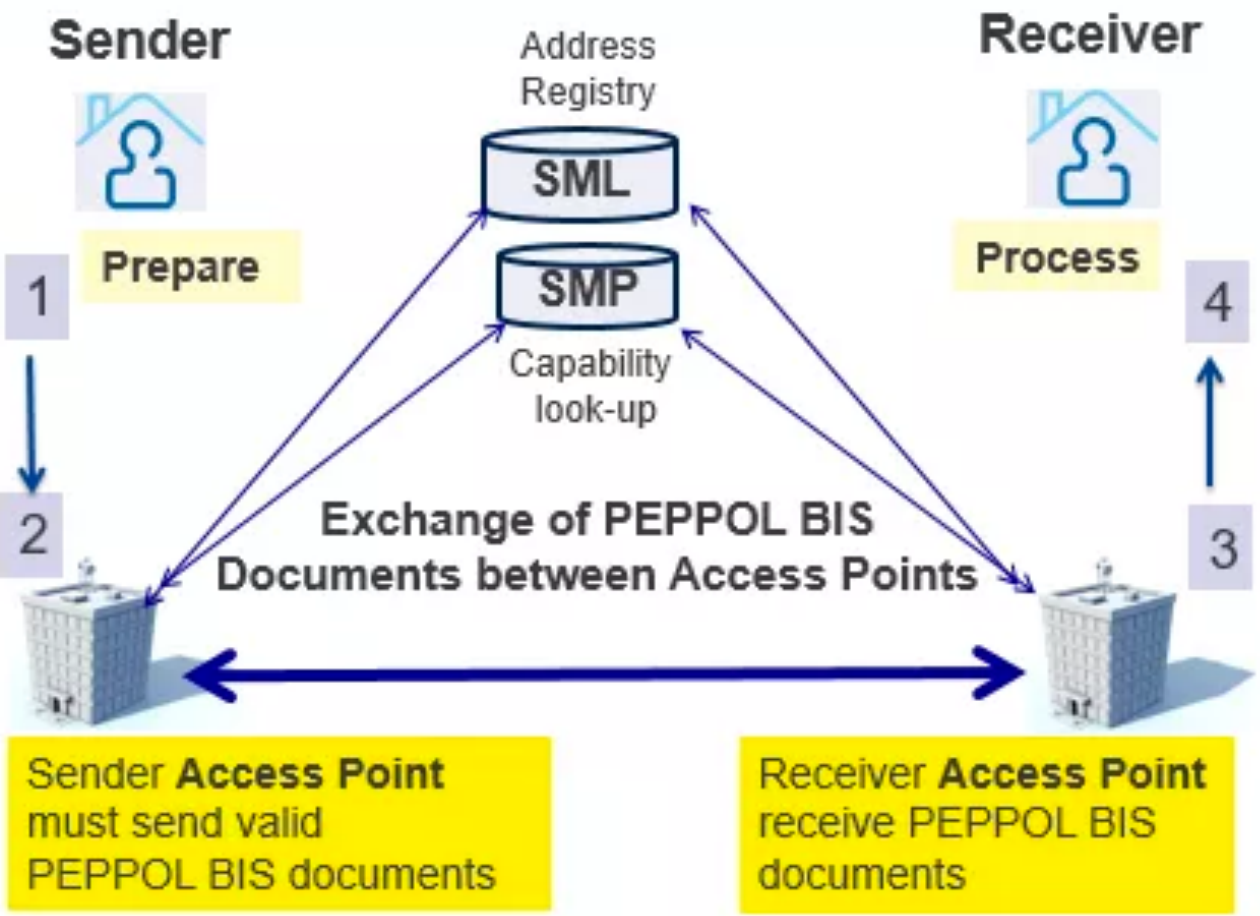
PEPPOL

Free to choose ANY Access Point in ANY country

Sign ONE Agreement; Connect to ALL

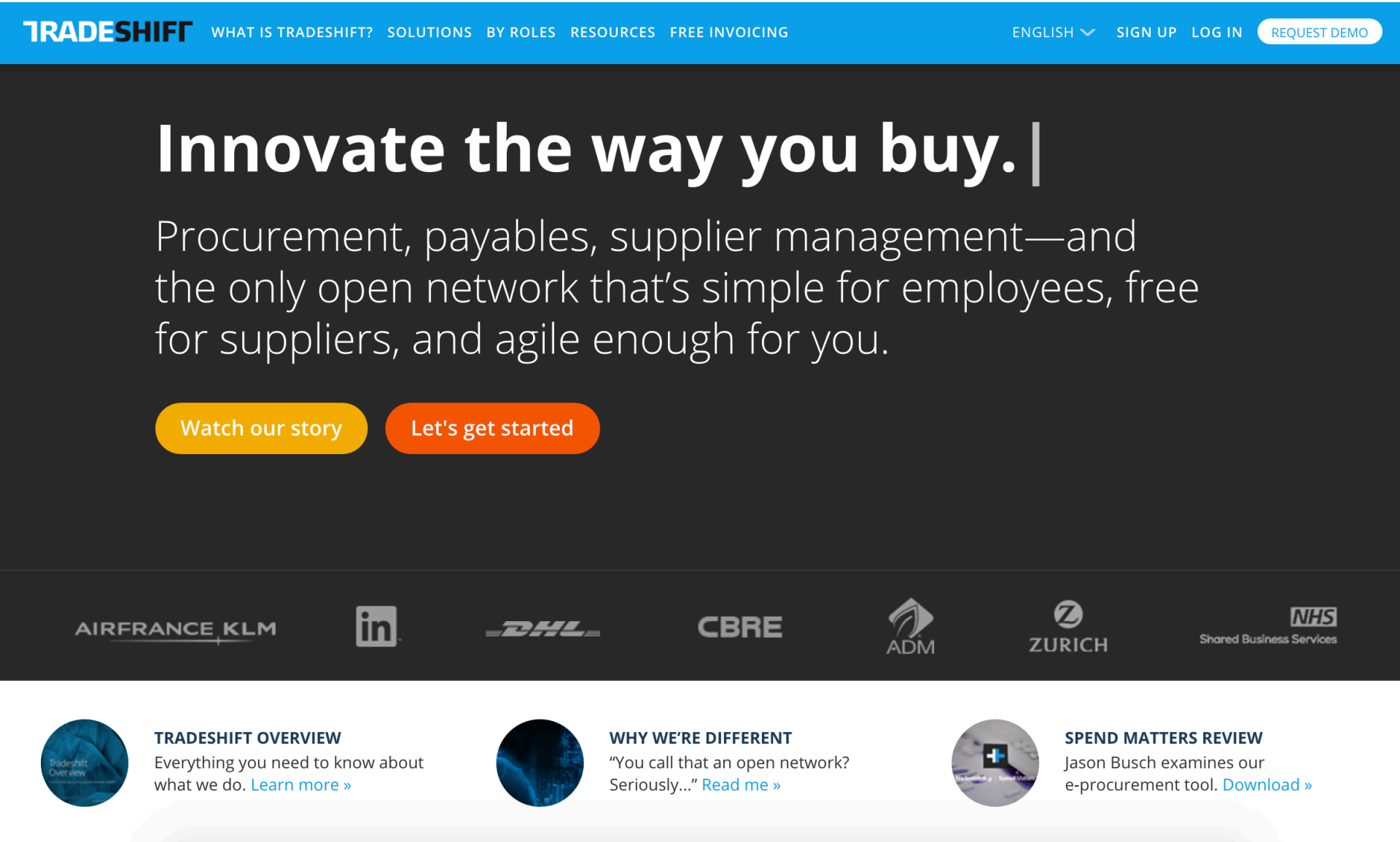
Open FOUR-corner model

NO ROAMING fees between PEPPOL Access Points



~~June 2016: Tradeshift raise $75m~~

May 2018: Tradeshift raise $250m



E-invoicing

Want to take steps to get paid quicker and submit your invoices in a way that doesn't involve paper, postage or manual handling?

e-Invoicing through Tradeshift is an exciting new business platform that is designed to make life simpler for organisations and we are working in partnership with them to deliver e-Invoicing for suppliers to our NHS clients.

Registering with Tradeshift is quick and simple and the benefits are immediate. If you are an SME or low volume supplier then the web-based portal at http://www.tradeshift.com/supplier/nhs-sbs/ is likely to be the best solution, offering immediate access to submit and review invoices wherever you are.

If you submit a higher volume of invoices, you may wish to consider integrating your existing invoicing software with the Tradeshift system. This is a simple process, with support available, to help you see just how beneficial e-Invoicing could be for your business.

Why e-invoicing?

- **Easy setup** - it's quick and simple to get started and just as simple to use
- **Free to use** - no setup fee, transaction fees or service charges
- **Improved communication** - track the status of your invoices and run key reports
- **Instant validation** - take advantage of 15 pre-submission checks to ensure your invoice is right first time

Contact Us
T: 0303 123 1177
E: SBS-W.Payables@nhs.net

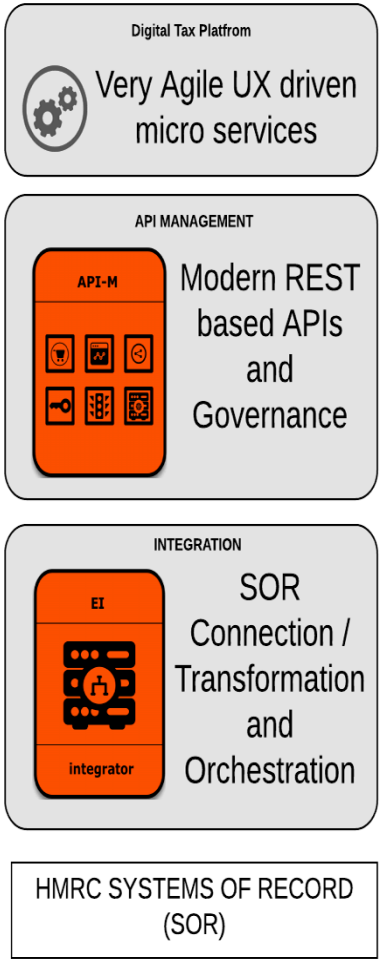
Netflix

- A REST and Cloud based SOA approach
- Continuous Delivery
- 100% Based in the cloud
- See excellent presentations from Adrian Cockcroft
  - e.g. http://www.slideshare.net/adrianco/global-netflix-platform

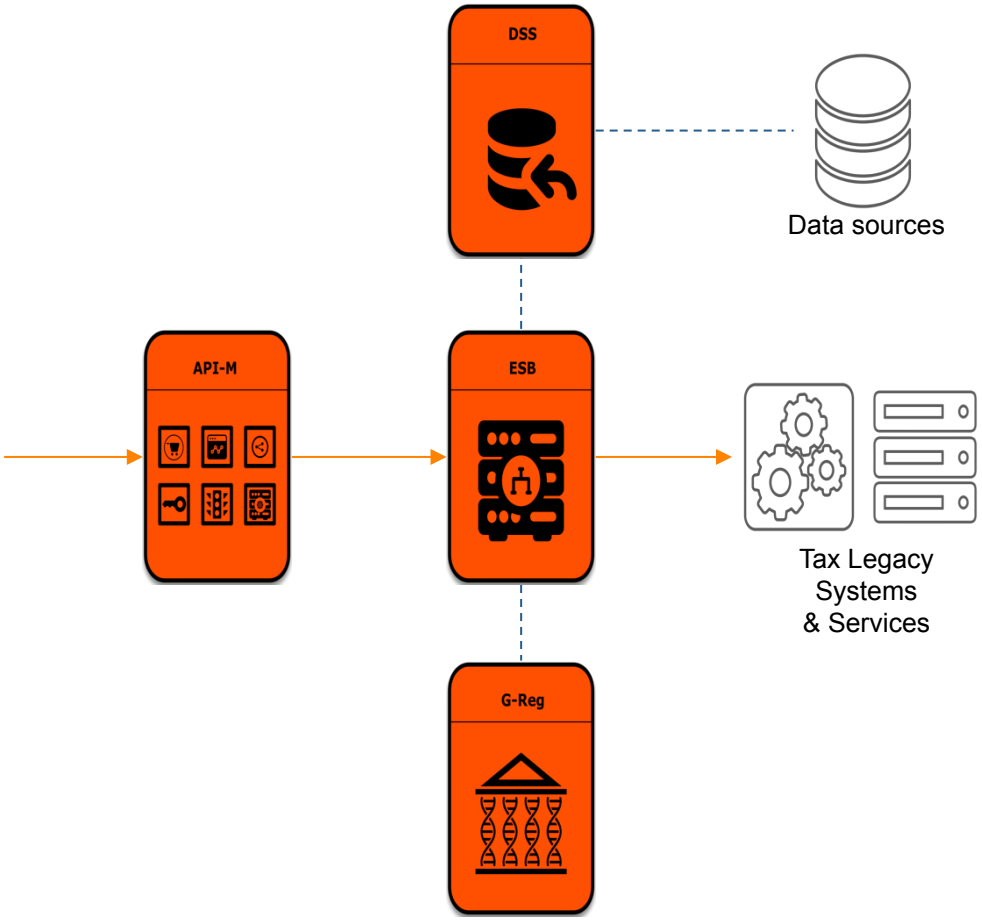
Use Case

- Businesses and individuals make tax returns online without having to use older technology and paper-based alternatives.
- Via Managed APIs, online Tax Applications get access to functionality of HMRC land 3rd Party Services on the Tax Platform
- Tax Legacy Systems that maintain all tax related information of taxpayers & perform tax processing

New Digital GOV.UK
Personal and Business
Tax Account management



Solution Architecture

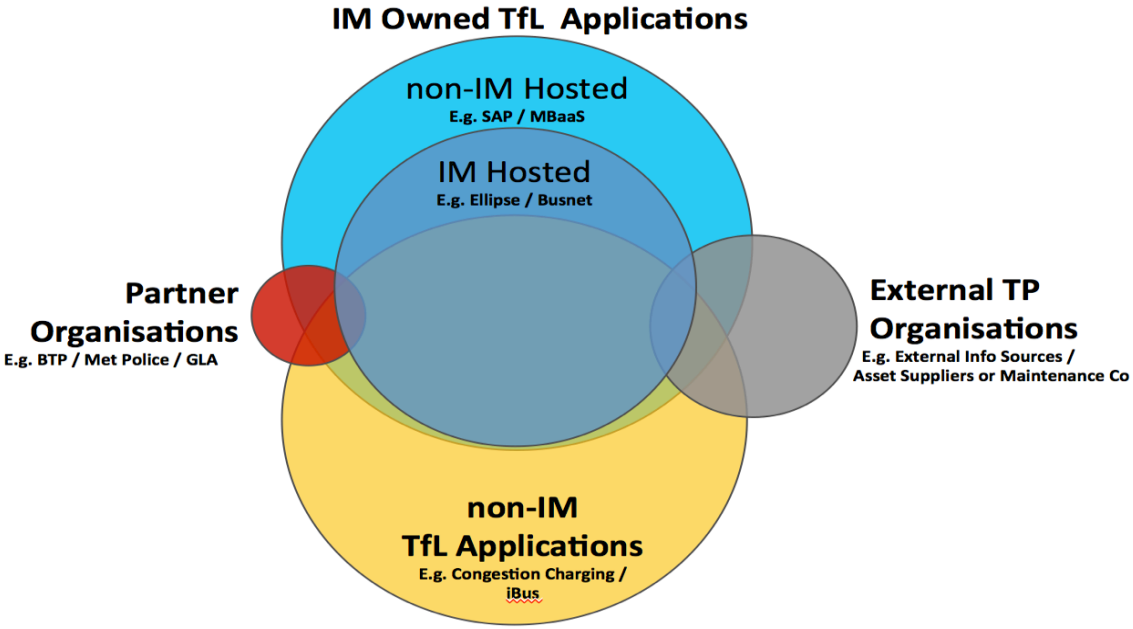


Multi Layer Approach

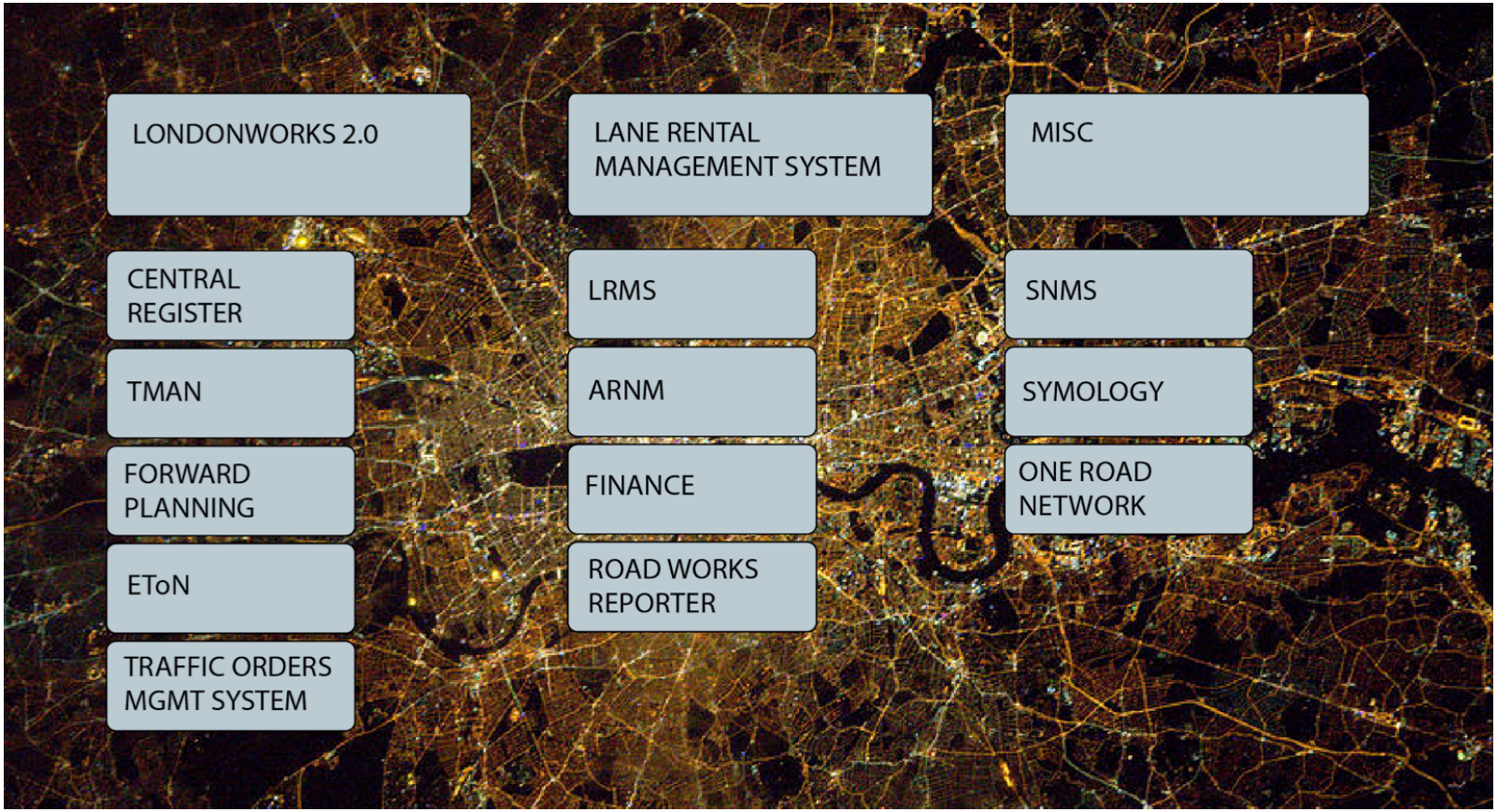
- API Management deployed in Public Cloud for external use, mainly for Third Party Integration (Large Companies, Accountants)
- Private Cloud for API Management for private APIs for internal and external use, Integration and orchestration, service integration and orchestration
- Access to Legacy System (ESB)

Transport for London (UK)

LondonWorks

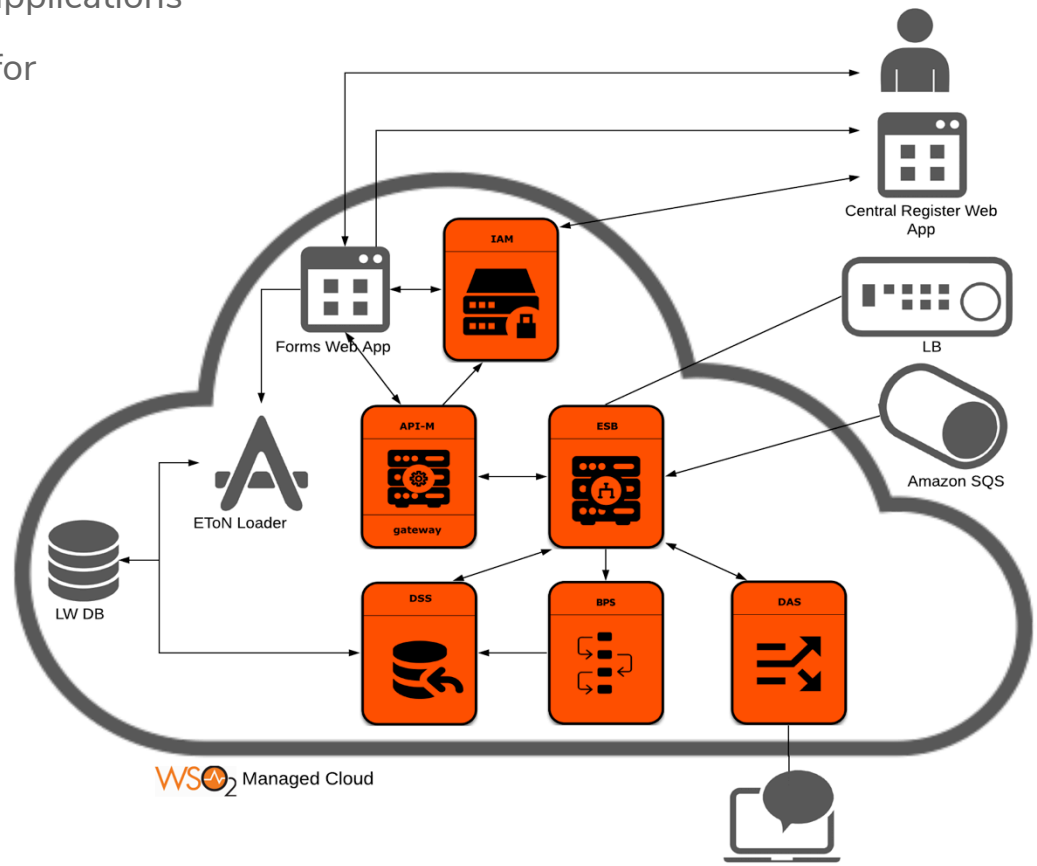


- **TfL (Transport for London)** is a Transportation Service Provider over 150 years.
- Around 31 million journeys are made across the TfL network everyday.
- TfL Integration Service is a service provided by **IM (Information Management)** department of TfL, which connects both IM and non-IM assets, and external organisations to provide/consume services to/from TfL.
- **LondonWorks** is a collection of applications used to manage the planning, programming, and delivery of street works on London's road network. It is the definitive source of road and street works information throughout TfL.

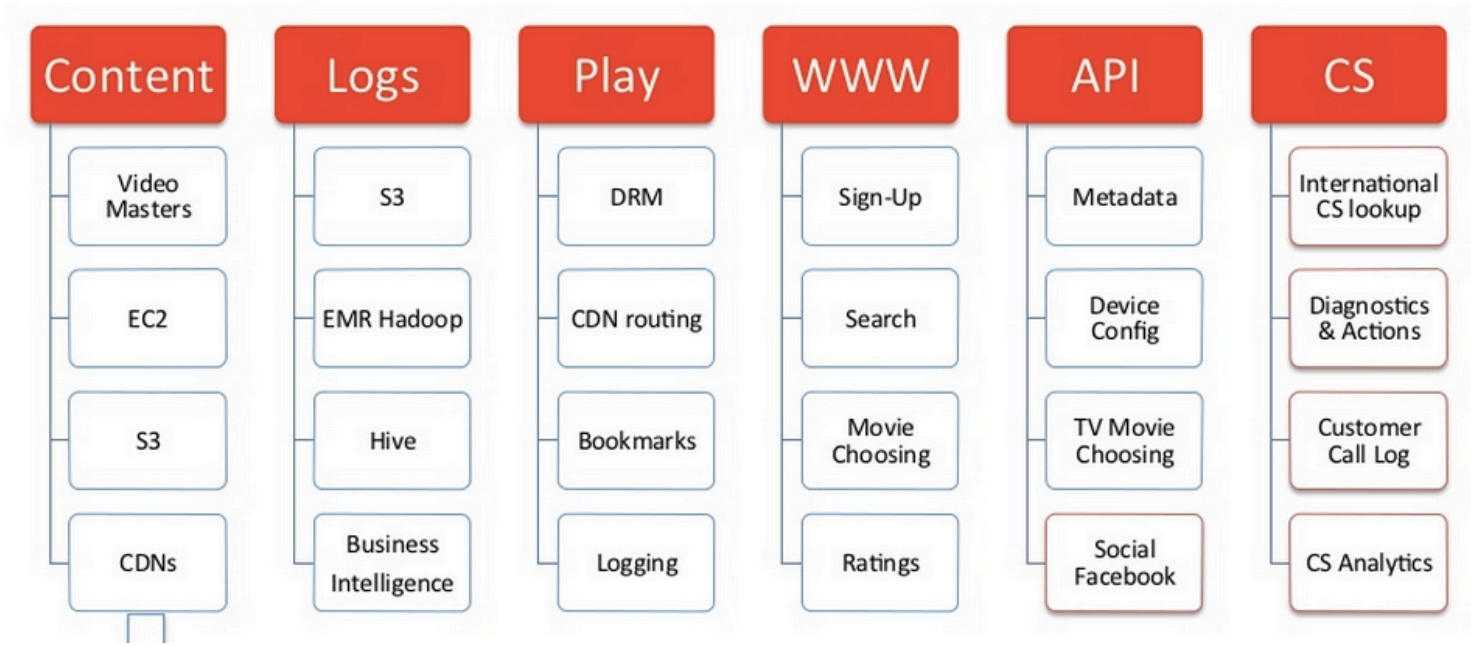


LondonWorks Architecture

- **WSO2 IS** - Unified identity across TfL network
- **WSO2 ESB, DSS** - TfL Integration Service between databases, and various other applications
- **WSO2 API-M GW** - API management for externally exposed APIs
- **WSO2 DAS** - Event filters and alerts management
- **WSO2 BPS - Business Process** management on TfL Integration Service



Netflix Deployed on AWS



Platform Services

- Discovery – service registry for “applications”
- Introspection – Entrypoints
- Cryptex – Dynamic security key management
- Geo – Geographic IP lookup
- Platformservice – Dynamic property configuration
- Localization – manage and lookup local translations
- Evcache – eccentric volatile (mem)cached
- Cassandra – Persistence
- Zookeeper - Coordination
- Various proxies – access to old datacenter stuff

The (in)famous Chaos Monkey

- Randomly kills machines
- Yes, production systems
- Proves that the system is resilient

Twitter Architecture

- Open Sourced their technology:
  - Finagle
  - http://twitter.github.io/finagle/
  - Called an RPC system, but completely asynchronous
  - Based on “Services”

http://monkey.org/~marius/talks/twittersystems/#4

Late 2012 architecture

Many **open source** components

- Memcache, redis, MySQL, etc.
- Necessarily heterogeneous

Organized around **services**

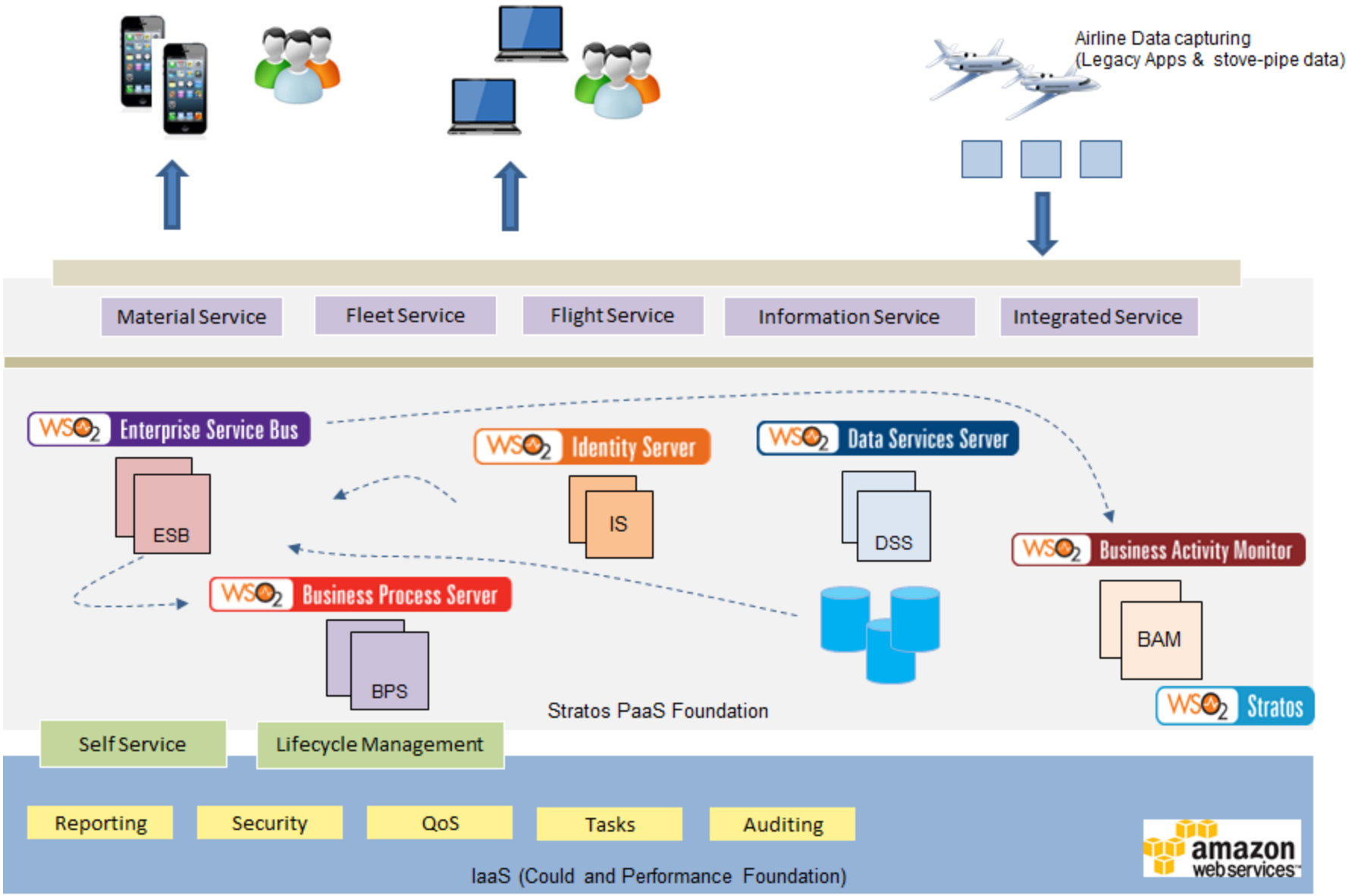
- Distinct responsibilities
- Isolated from each other
- Distributed computation and data
- RPC between systems

Multiplexing HTTP frontend

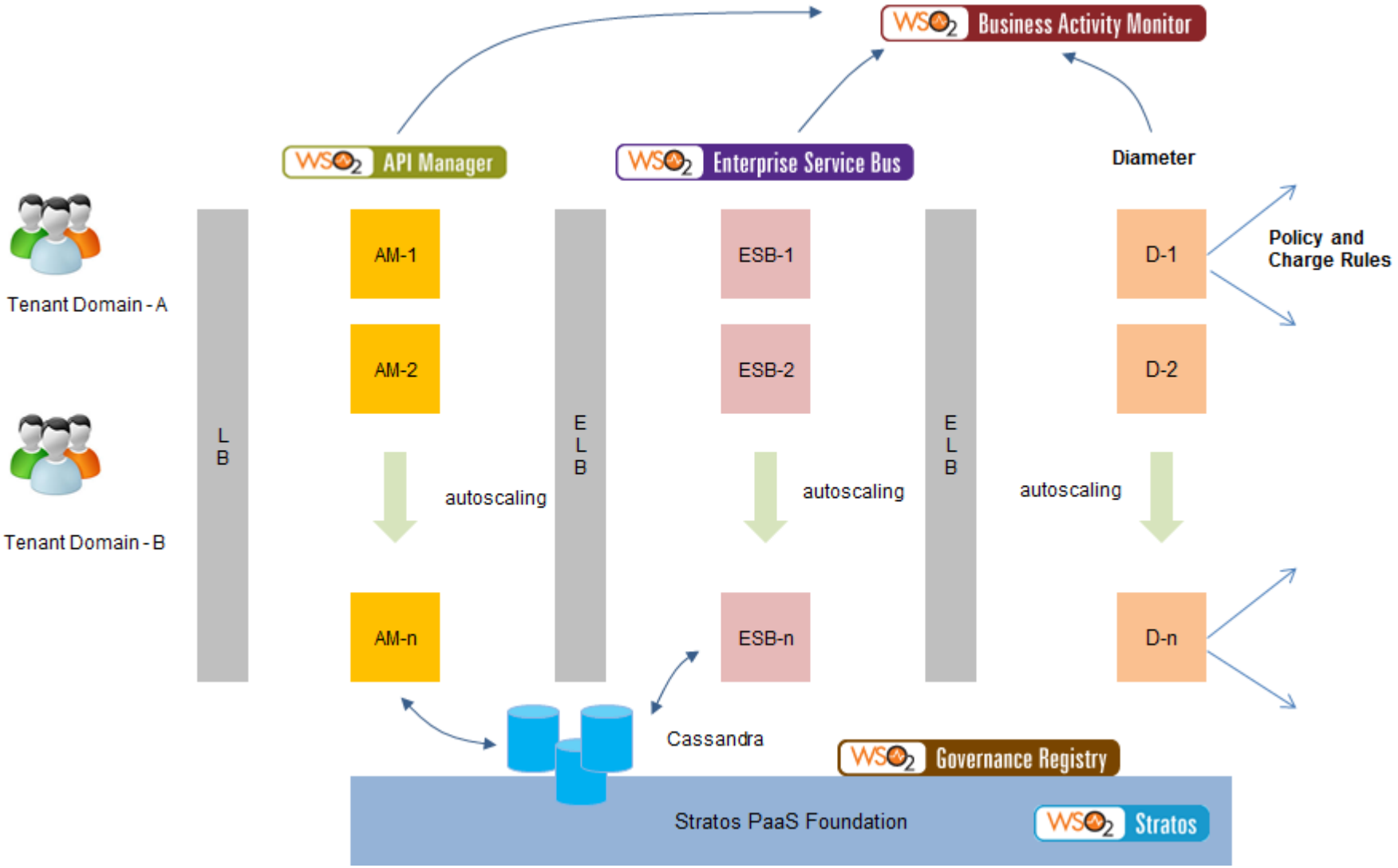
- Crucial for modularity, load balancing

Boeing Digital Airline

Case Study : Boeing - A PaaS based Integration and API ecosystem



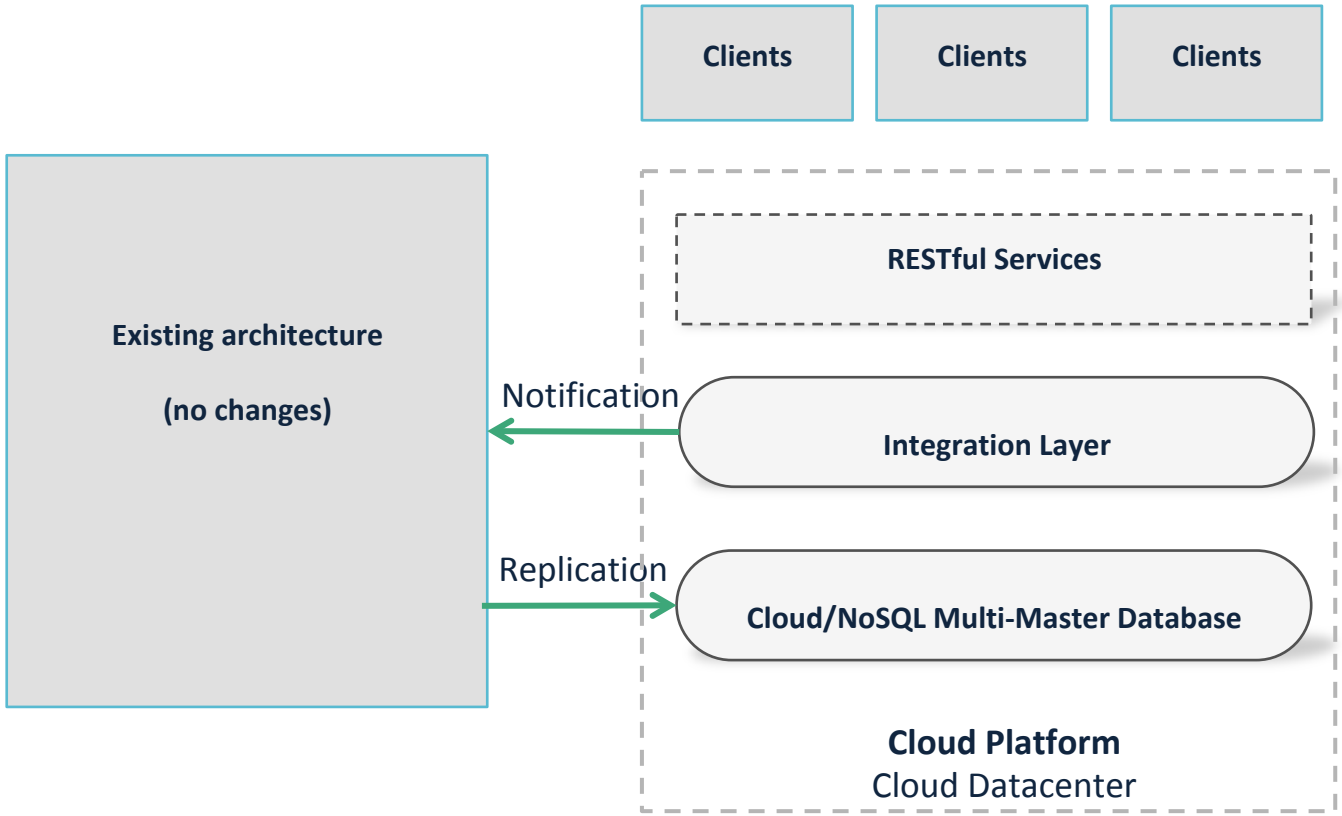
Case Study : Multi-tenanted Mobile Orchestration Gateway Platform



Pay TV company

- Needed to scale up to provide instant pay-as-you-go on mobile devices
- Support Disaster Recovery (DR)
- Elastic Scale e.g. during an important football match

Architecture



Anti-patterns

- Use a full waterfall model
- Don't budget time for integration test
  - Assume that standard coding unit test->integration test will work
- Build unit tests that don't test interoperability
  - E.g. Simulate XML request/response inside the calling system rather than calling a remote system
- Wait until all the systems are ready before starting any integration test
  - A delay to one system will hold up testing all the others
- Don't bother with continuous build and test
  - Even better build by hand
  - **Even better** test by hand too
- Have a nice complex process to hand over from development to test
  - That way each defect will take a long time
- Wait until the project is failing to find out your team doesn't have the skills

Questions?