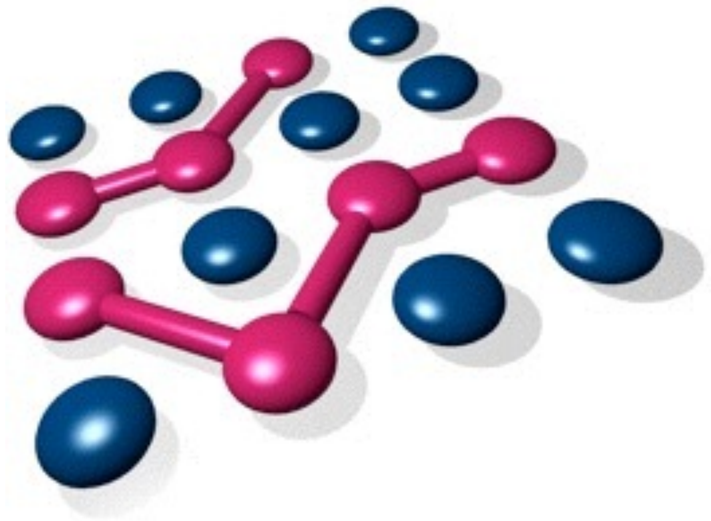
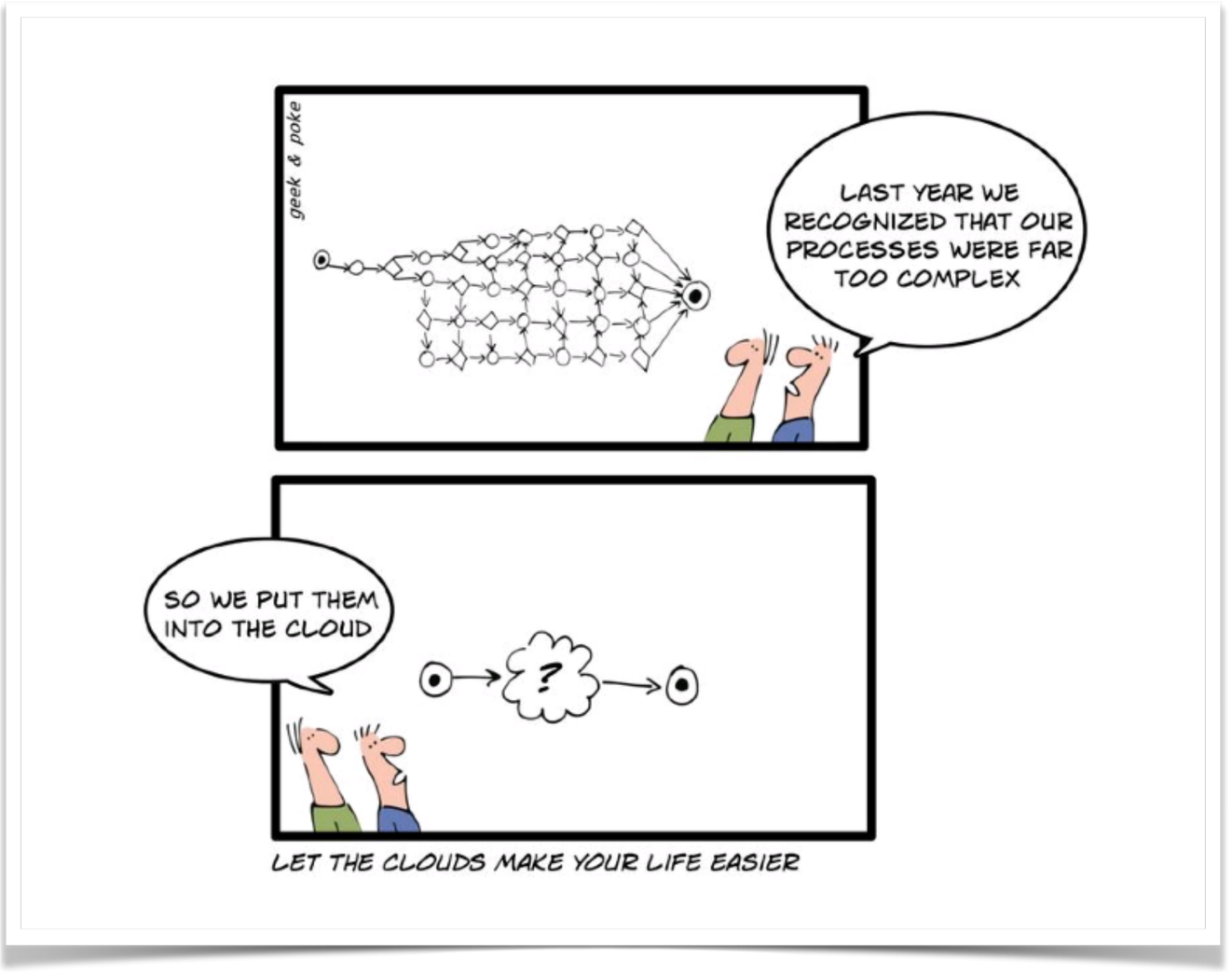
Paremus



Cloudy with a chance of bundles (and non Java components)

More a Ramble than a Forecast



CLOUD ARCHAEOLOGY

1950

1960

1970

…modular architecture was realized by architects like Fred Brooks and Gerrit Blaauw, Gordon Bell and Allen Newell, and Carver Mead and Lynn Conway in the 1960s and 1970s.

3 Modular architectures in turn enabled the computer industry to evolve to its present form, which we call a “modular cluster”.

http://www.people.hbs.edu/cbaldwin/DR2/BaldwinClarkCES.pdf

1980

1990

2000

then a company enforced SOA…

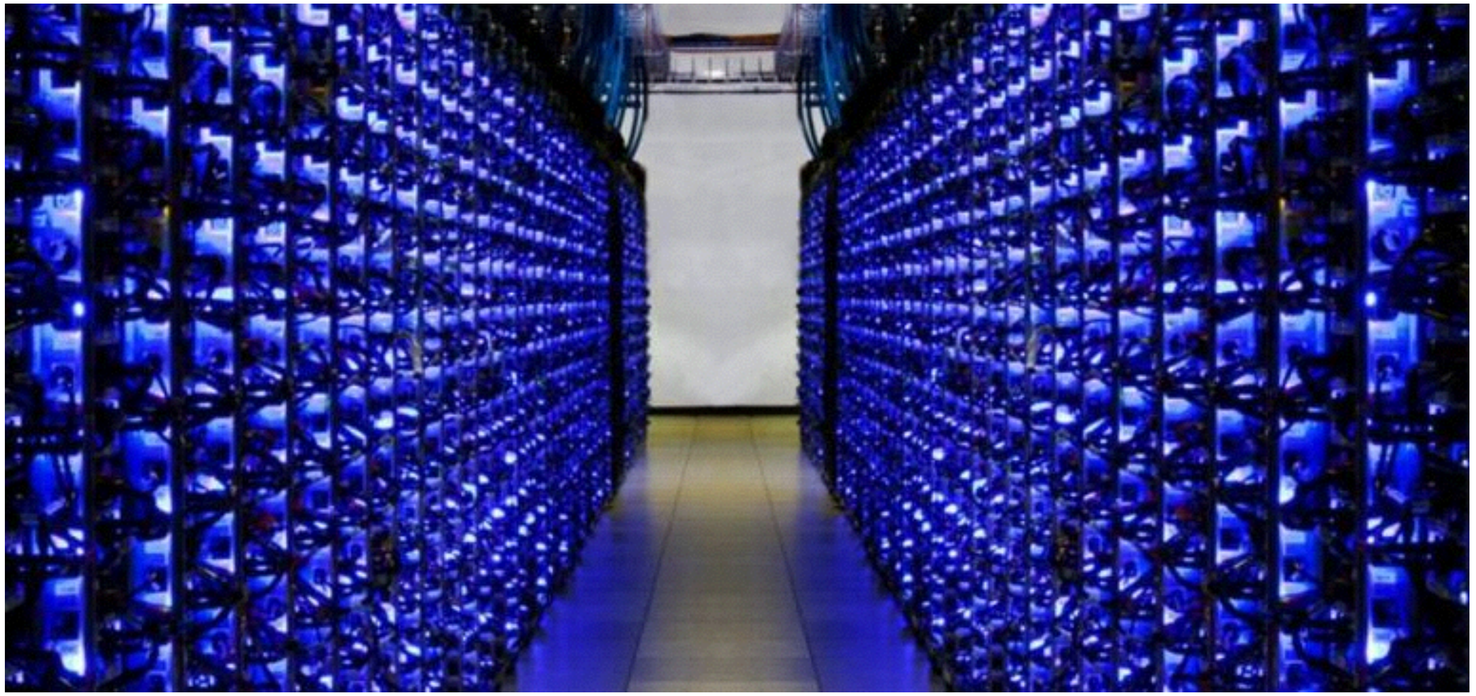
…Jeff Bezos issued a mandate, sometime back around 2002 (give or take a year):

- *All teams will henceforth expose their data and functionality through service interfaces*
- *Teams must communicate with each other through these interfaces.*
- *There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.*
- *It doesn't matter what technology they use.*
- *All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.*

The mandate closed with*: Anyone who doesn't do this will be fired. Thank you; have a nice day!*

http://apievangelist.com/2012/01/12/the-secret-to-amazons-success-internal-apis/

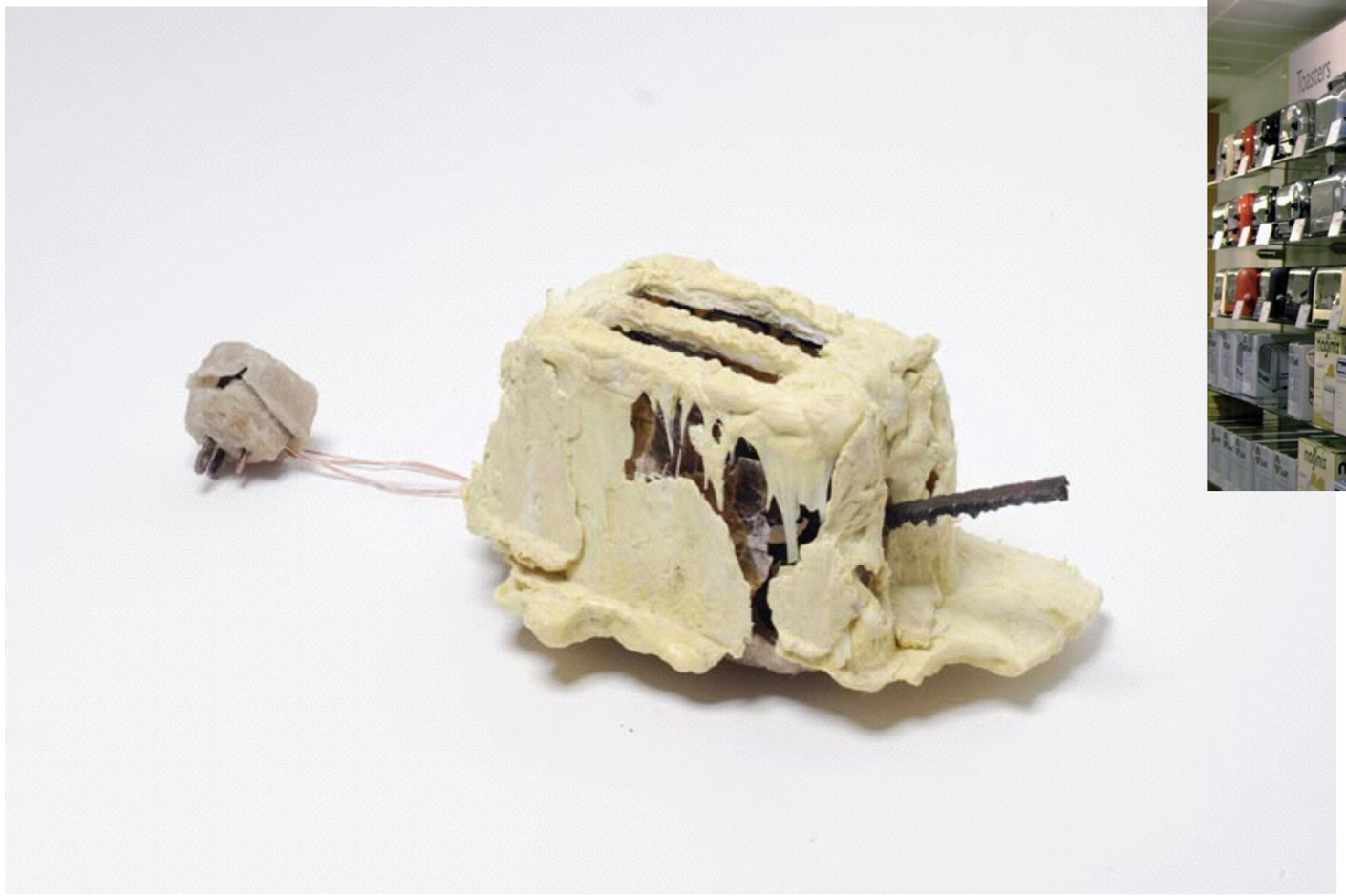
Today



Today's 'Cloud' enabled by…

- Fast / Pervasive WAN.
- High degree of physical modularity
- Coarse Grained Software Modularity - loose coupling via REST / SOA based architectures.
- Logical Resource Partitioning (VMI's) - standard deployment artefact.

Modularity - The Toaster Project…



Modularity enabled the rapid evolution of the hardware / manufacturing industries. Modularity increased product diversity while driving costs down rapidly.

http://www.sciencemuseum.org.uk/smap/collection_index/thomas_thwaites_the_toaster_project.aspx

Where Next?

role up, role up… Google Embraces Docker, the Next Big Thing in Cloud Computing
http://www.wired.com/2014/06/eric-brewer-google-docker/

A more modular approaches to Software artefacts?

- **The Application as the Module -** Dynamic install of only what is needed 'Docker' & traditional packages rather than the VMI (virtual machine image) '*Kitchen Sink*' approach!

- **Modular Applications -** The Rise of OSGi…!
  - Dynamic composition of the runtime Artifact from re-usable components?
  - Dynamic assembly in the context of / in response to the capabilities of / the runtime environment.

Where Next?

Modularity & Dynamism at the Services layer.

- Dynamic Microservices:
  - Behaviors - Async / Circuit Breaker / Back Pressure / Actor.
  - Service Wire Up - http://techblog.netflix.com/2014/06/building-netflix-playback-with-self.html

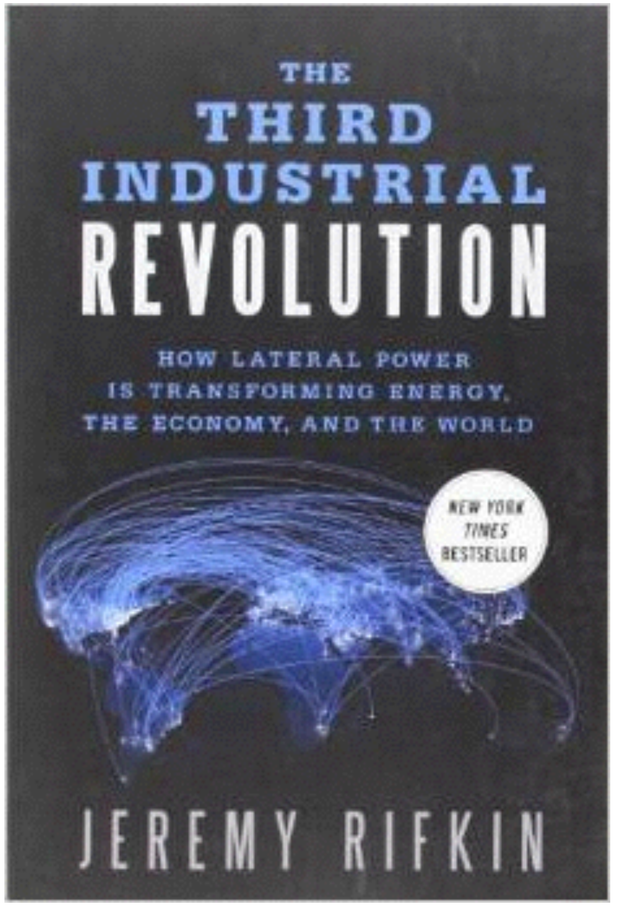
- OSGi μServices - “probably the best μServices architecture in the World” ;)

Where Next?

Evolution from Centralised….

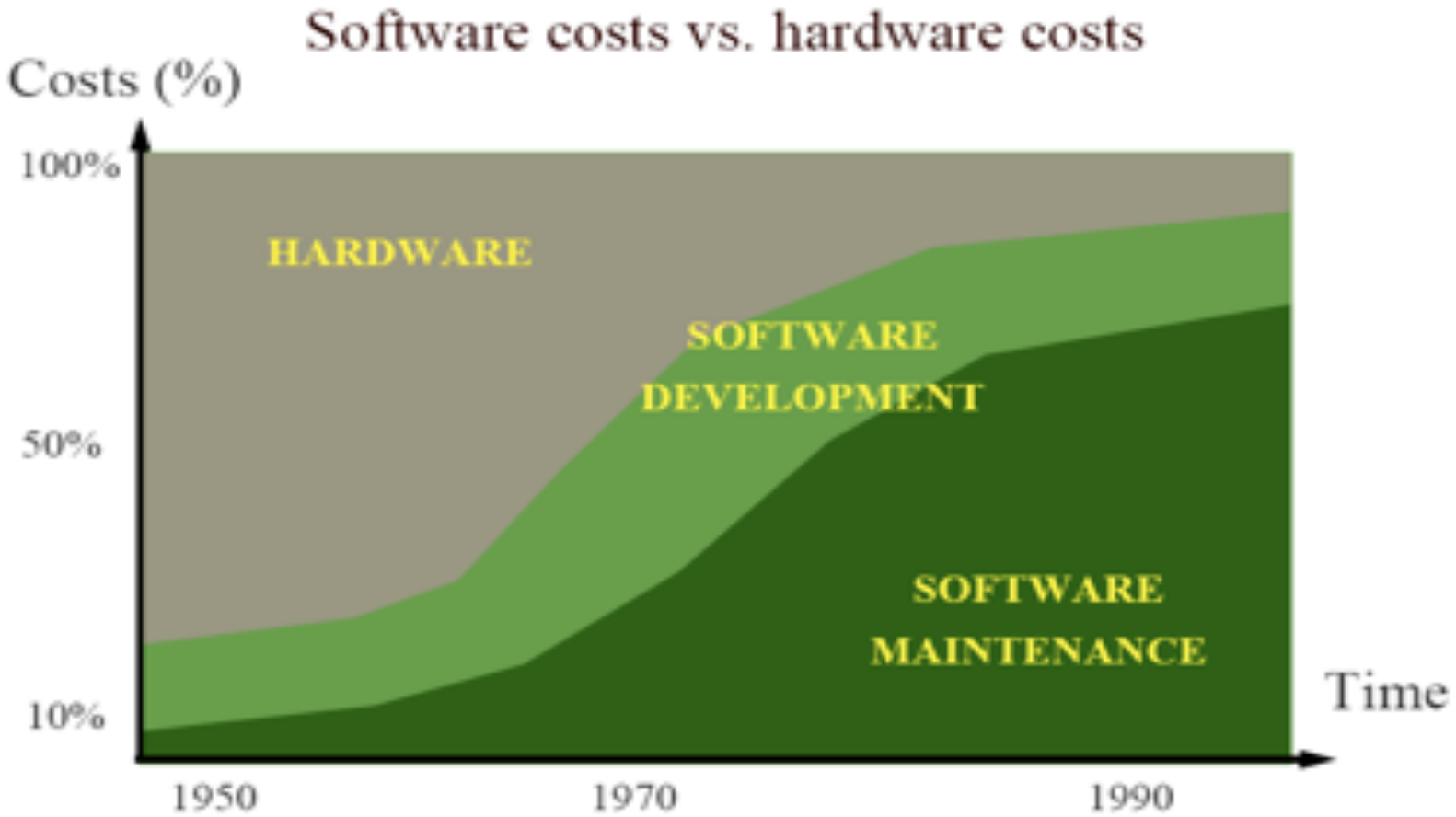
- Locality Matters!
  - Data Locality - Regulatory / Political / Social
  - Data Privacy - Regulatory / Political / Social
- Internet Robustness - remember ARPNET?
- Smart Energy / Distributed Energy Production.

to Federated Clouds?



The Symptoms (forcing factors)

Environmental Complexity / OPEX



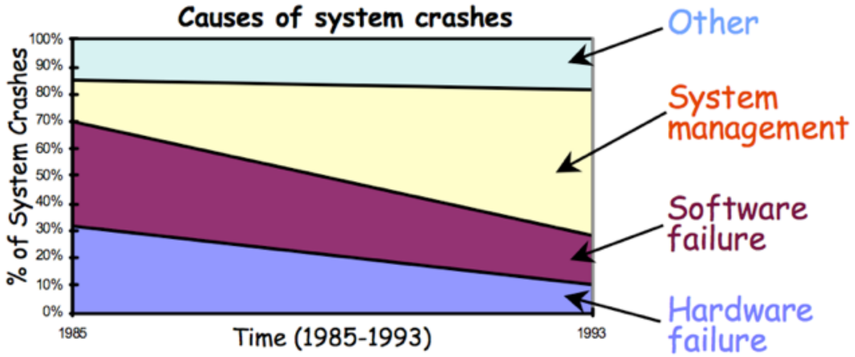
Environmental Complexity & Failure

"Digital infrastructure exceeding limits of human control, industry experts warn"

Guardian 23rd August 2013

http://www.theguardian.com/technology/2013/aug/23/nasdaq-crash-data

What causes un-availability?



- **Many different factors are involved**
  - human behavior during maintenance dominates

> *"We don't yet have a design for society that can run this technology well. We haven't figured out what the right human roles should be."*

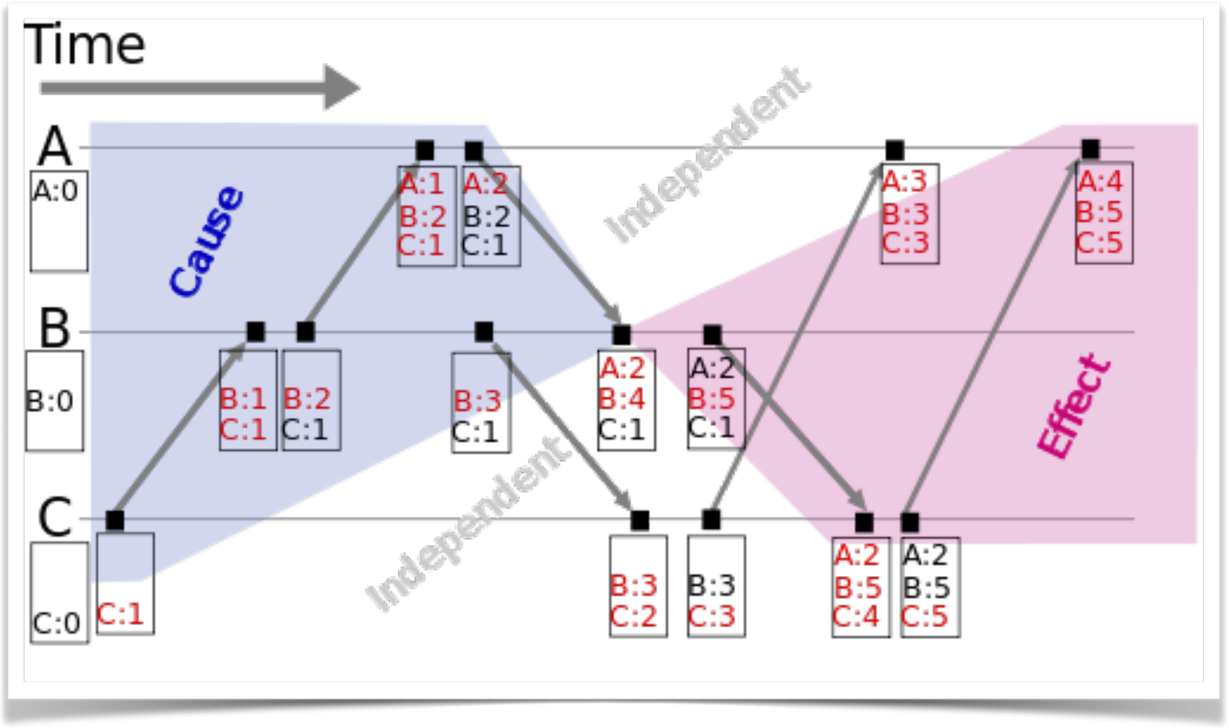
> *"These outages are absolutely going to continue," said Neil MacDonald, a fellow at technology research firm Gartner. "There has been an explosion in data across all types of enterprises. The complexity of the systems created to support big data is beyond the understanding of a single person and they also fail in ways that are beyond the comprehension of a single person."*

> *"The outage at Amazon last year was traced back to some of the processes and technologies they had put in place to make it more resilient," said MacDonald.* ***"It is almost like an auto-immune disease, where the systems they created to make it more resilient actually spread the failure more rapidly."***

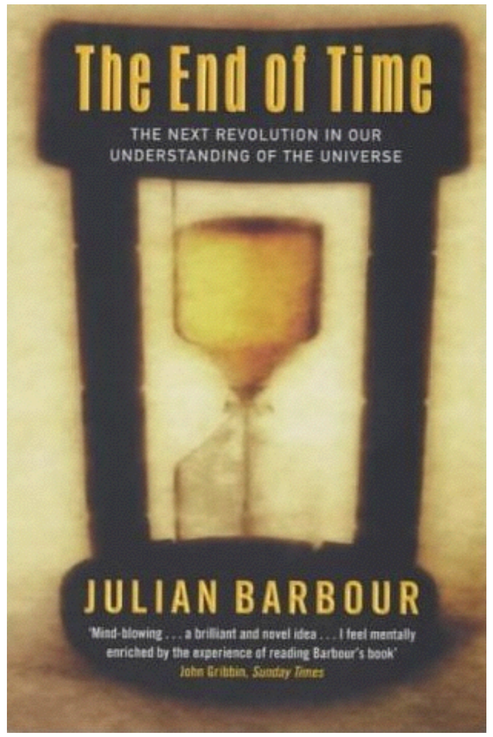
Diagnosis - The underlying problems

Change & the nature of Time (possibly)

Planned or Unplanned - Change is Fundamental, Change is Unavoidable

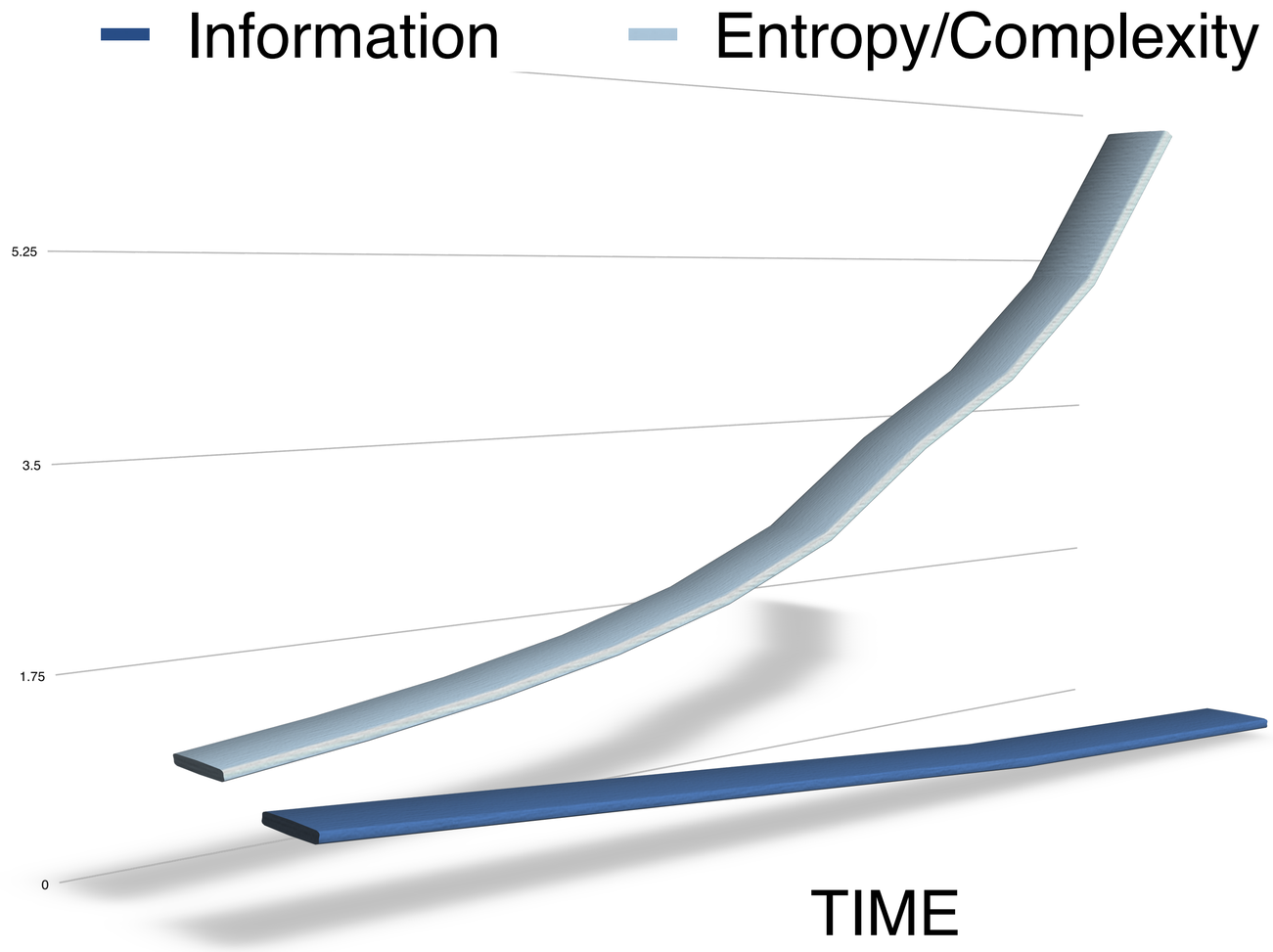


http://bit.ly/1mi8RtB



http://platonia.com/index.html

Information Loss



Within every SOA solution, within every Cloud deployment, lies a rotting codebase.

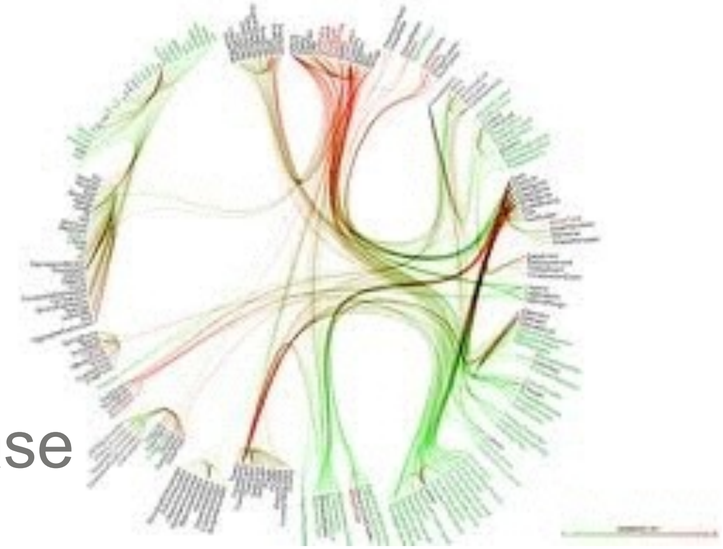
At least two contributing factors?

1. **DEPENDENCIES TEND TO INCREASE** - "All repairs tend to destroy the structure, to increase the entropy and disorder of the system. Less and less effort is spent on fixing the original design flaws; more and more is spent on fixing flaws introduced by earlier fixes. As time passes, the system becomes less and less well-ordered..…"

'**No Sliver Bullets'** F Brooks - http://www.cs.nott.ac.uk/~cah/G51ISS/Documents/NoSilverBullet.html

2. KNOWLEDGE OF THESE DEPENDENCIES TEND TO DECREASE -
The Dead Sea Effect…
http://brucefwebster.com/2008/04/11/the-wetware-crisis-the-dead-sea-effect/

Dependencies



We **know that over time** …

⟹ Dependencies tend to Increase

⟹ Knowledge concerning dependencies tend to decrease

BUT As Things Change …

⟹ Dependencies Change

Automated, Dynamic Dependency Discovery & Management is ESSENTIAL

All forms of Dependency: Bundle / Service <> Bundle / Service, Bundle / Service > Environment, Bundle / Service <> Configuration.

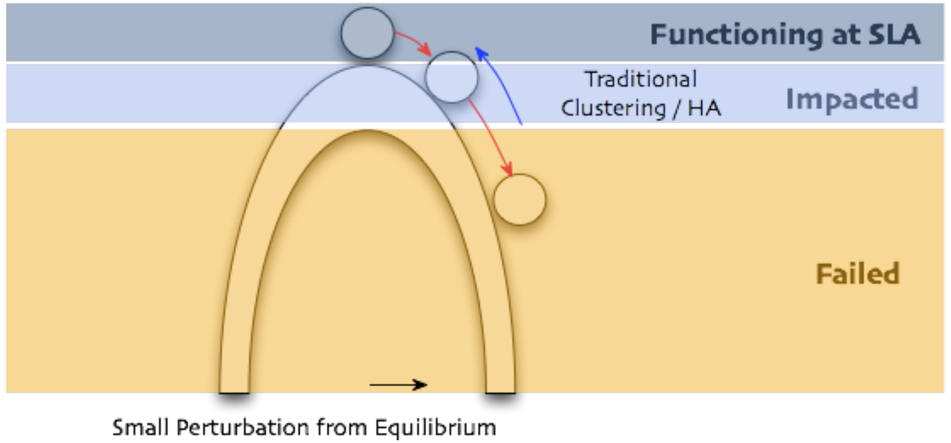
Tight Coupling & Black Swans

- Tightly Coupled systems are prone to catastrophic cascading failures (Black Swan events)
- Yet we continue to incorrectly focus on MTTF & build rigid / locked down environments?

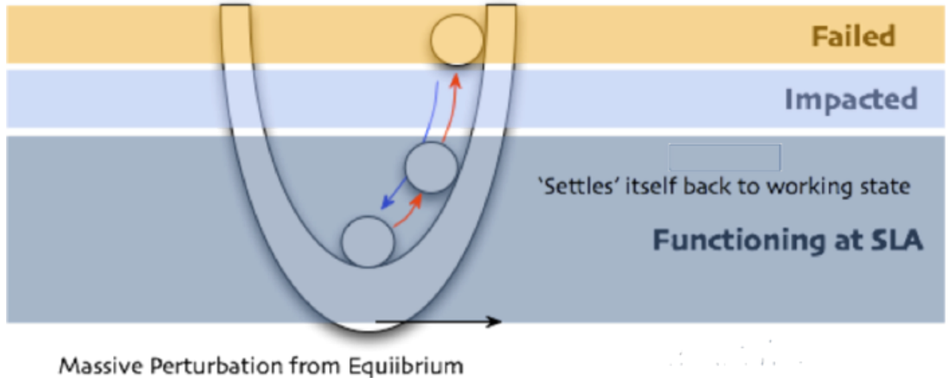
The Treatment

Failure is Inevitable…

Traditional Systems



Recovery Oriented Approach



Robust systems detect and respond to this.

Chaos Monkey. "We have found that the best defence against major unexpected failures is to fail often." http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html.

Paremus influenced by ROC (Berkley) and Crash Only (Stanford) work and general research in Complex Adaptive Systems.

Dynamism - forms of…

- **Agile** - Easy to Change

the enabler for →

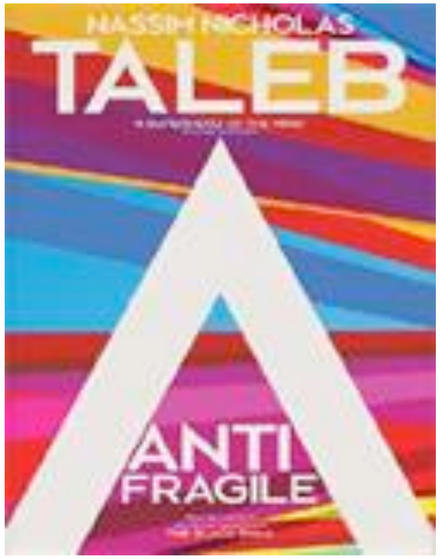
- **Adaptive** - Ability to change in response to external influences (Environmental changes, User Behaviors)

the enabler for →

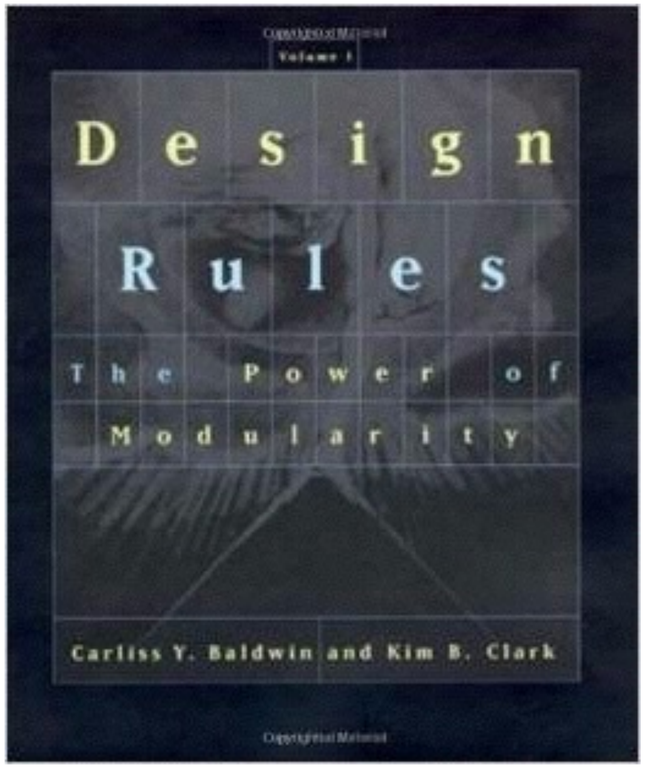
- **Robust** - Ability to change to ensure a ‘working’ state

the enabler for →

- **Anti-fragile / Evolvable** - Ability to actually improve performance courtesy of unforeseen environment change!



Modularity… the value of



1. Modularity makes **complexity manageable**;
2. Modularity enables parallel work.
3. Modularity is tolerant of uncertainty

Any of which may justify an investment:

Elements of a modular design:

- may be changed
- *after the fact* and
- *in unforeseen ways*

As long as design rules are obeyed - "*tolerance of uncertainty*".

Design Rules, Volume 1: The Power of Modularity (MIT Press, 2000) *http://www.amazon.com/Design-Rules-Vol-Power-Modularity/dp/0262024667*

Modularity & Complexity

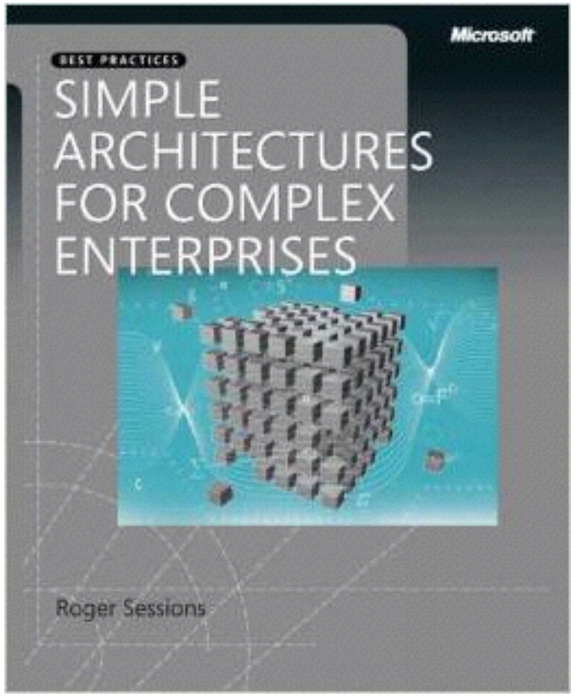
Glass's Law states that increasing the functionality of a system by 25% doubles the complexity of that system…

From which one derives…

C = complexity

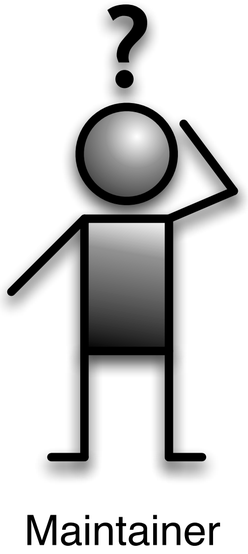
F = functionality

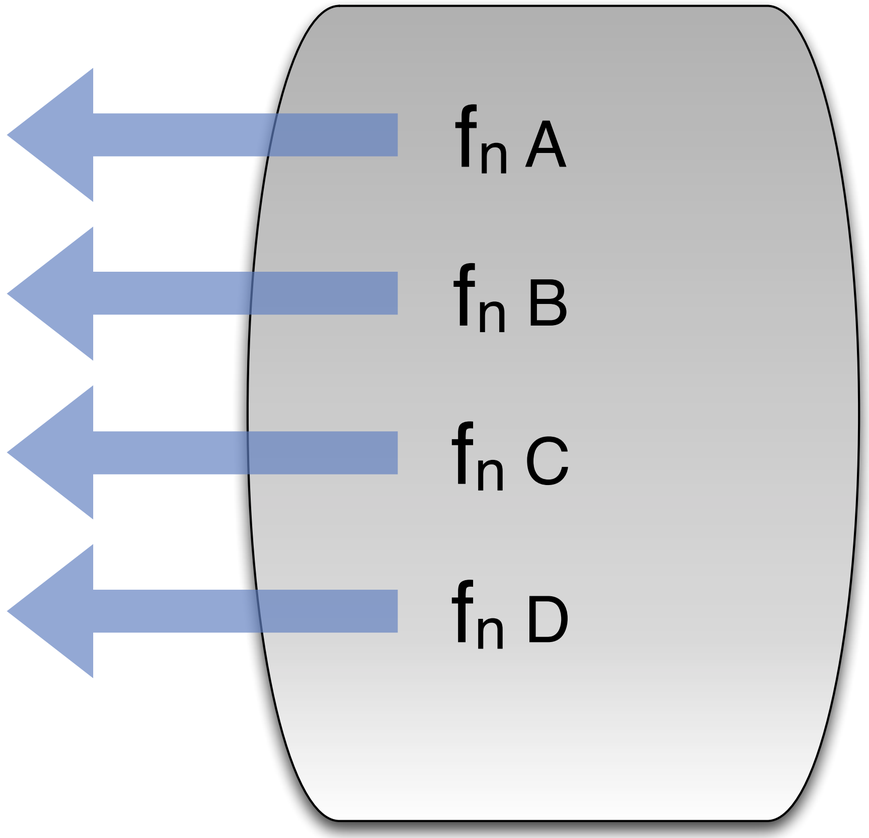
$$C = F^{3.11}$$



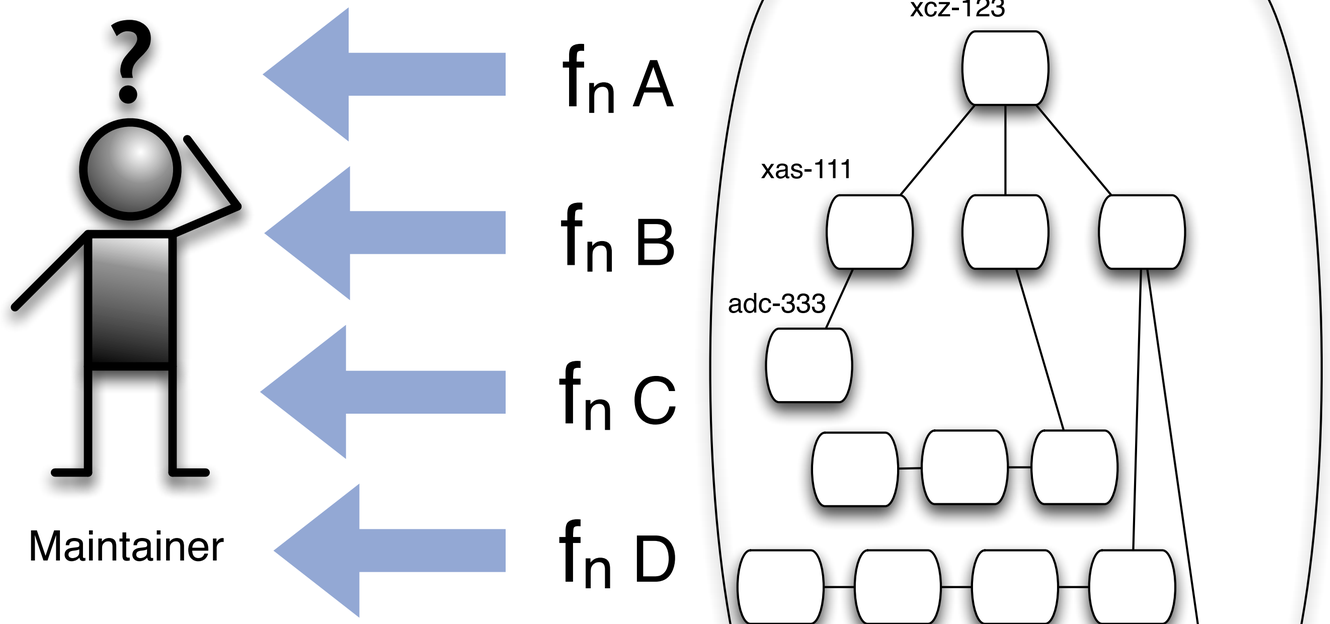
http://objectwatch.com
http://bit.ly/1x5TLv8

Modularity… Break down Monoliths





Modularity… Implicit dependencies are now Explicit

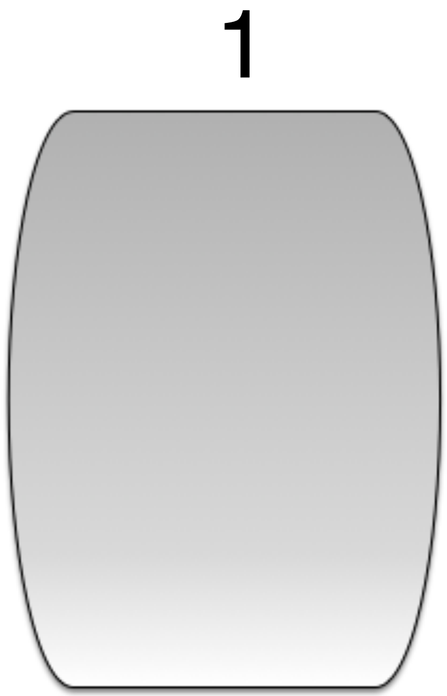


- Enforced isolation between Modules.
- Relationships explicitly defined - **not by names** - but by advertised *Requirements* and *Capabilities*.
- Impact of Change communicated by Semantic Versioning.

The System ‘appears’ more Complex ?

Modularity… Applying Glass's Law

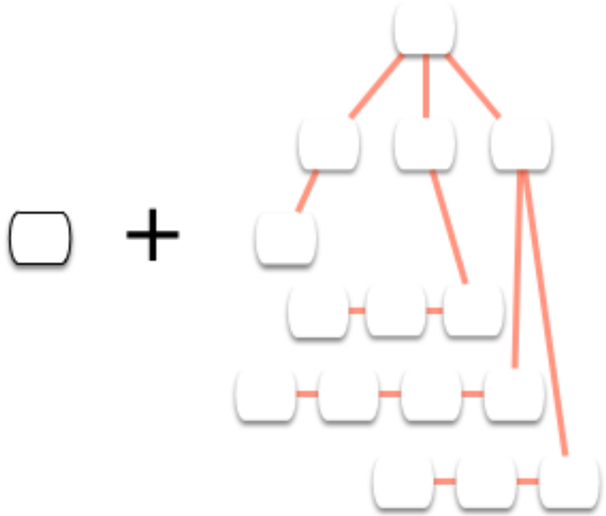
Relative Complexity Measure



Monolith

Decomposition Process

$$15\ x\ \frac{1}{4500} + \text{mod}\textit{ule wiring}$$

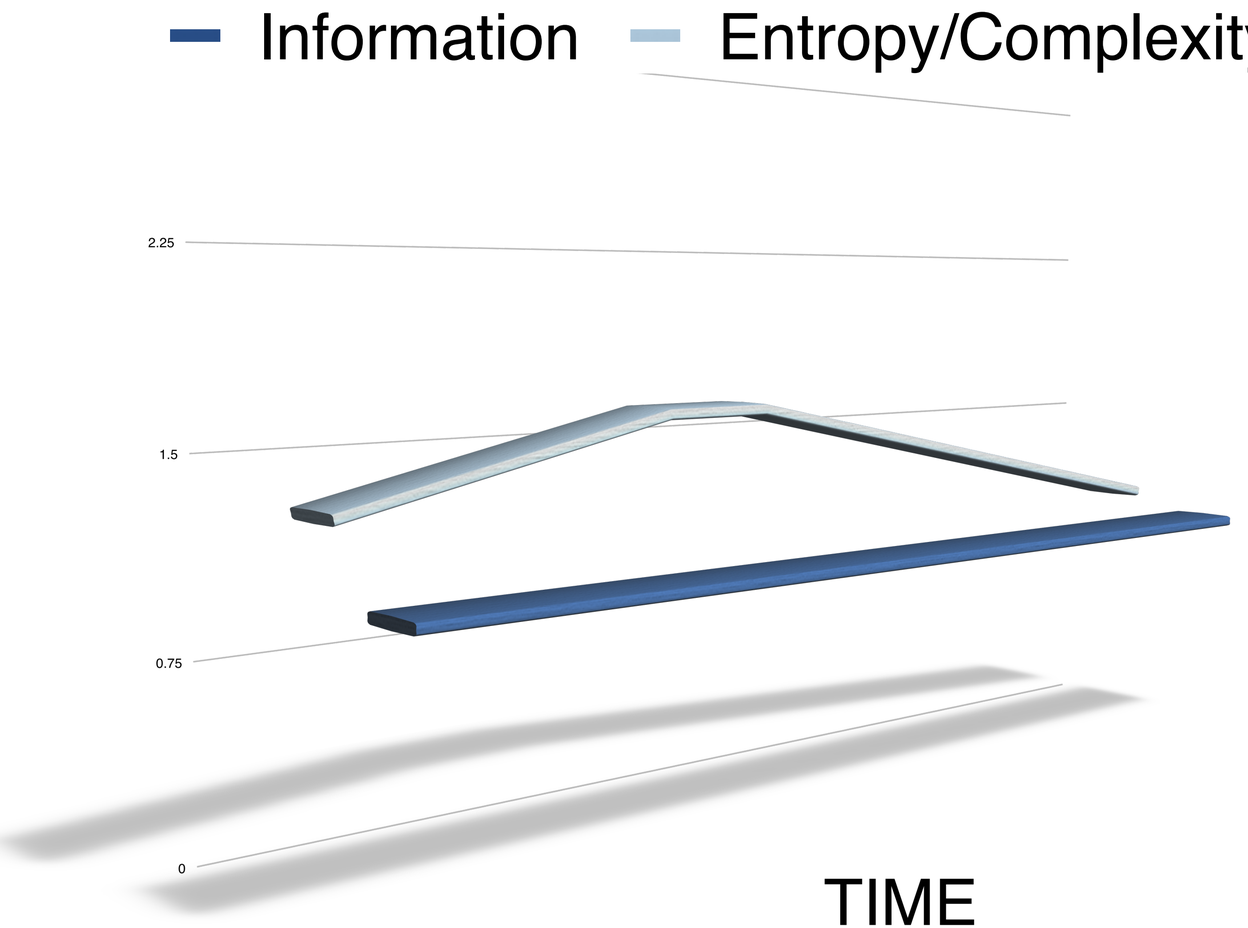


Modules

Module Wiring

Modularity - minimizes internal complexity of each Function. But we must now consider the explicit dependencies between Modules.

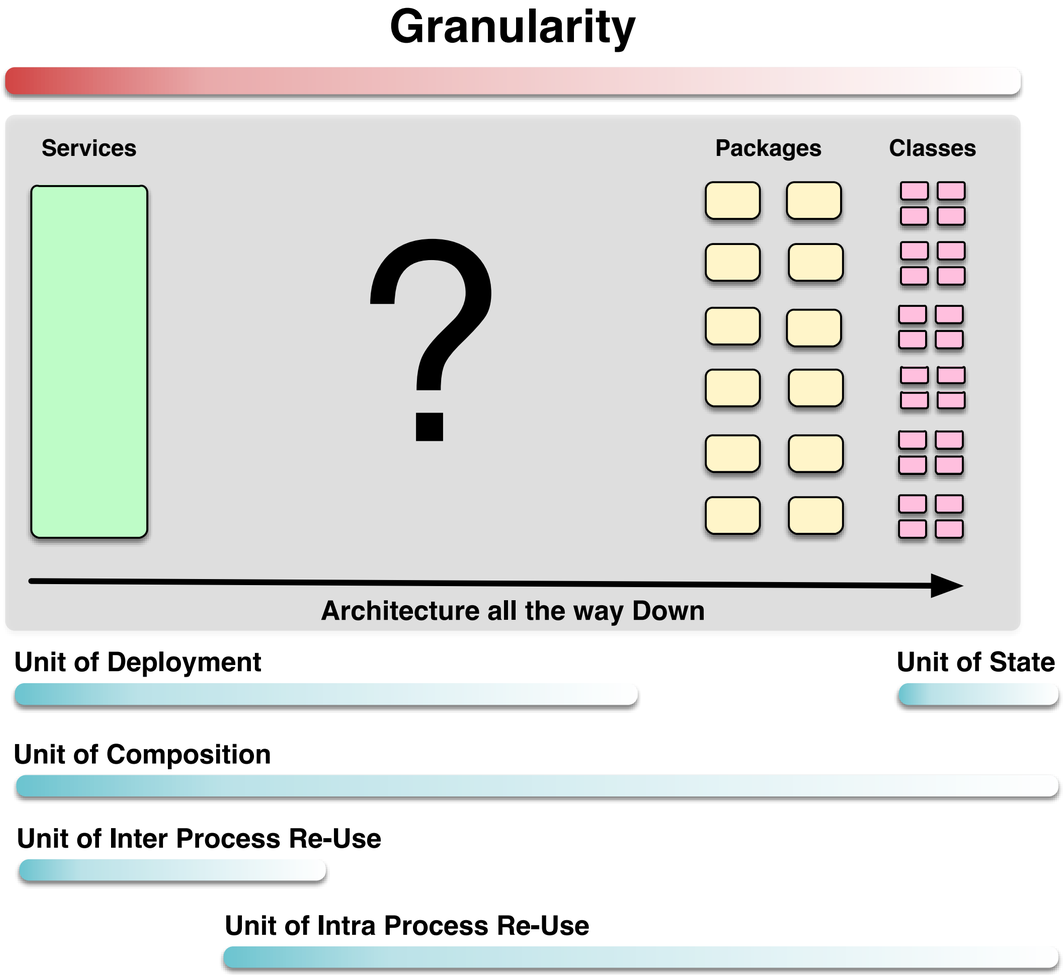
Modular Systems

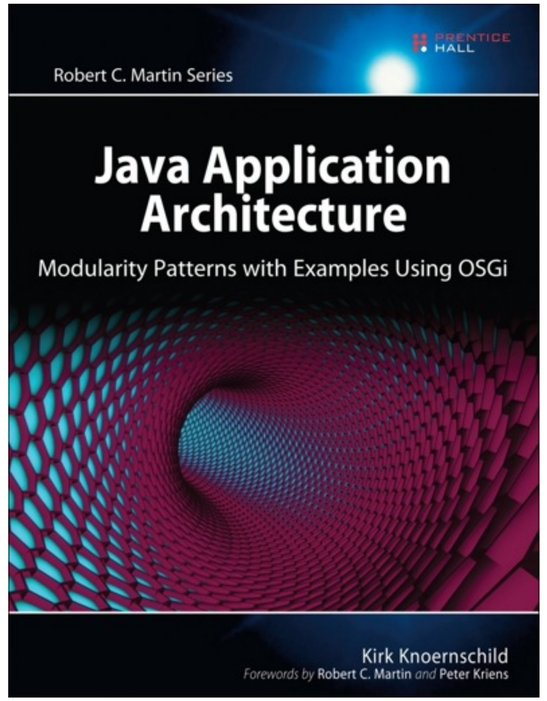


Antidote to Code Rot…

- Small components can be completely refactored in isolation.
- Self-Documenting. Structural information is actively preserved via the module's self-documenting dependencies.

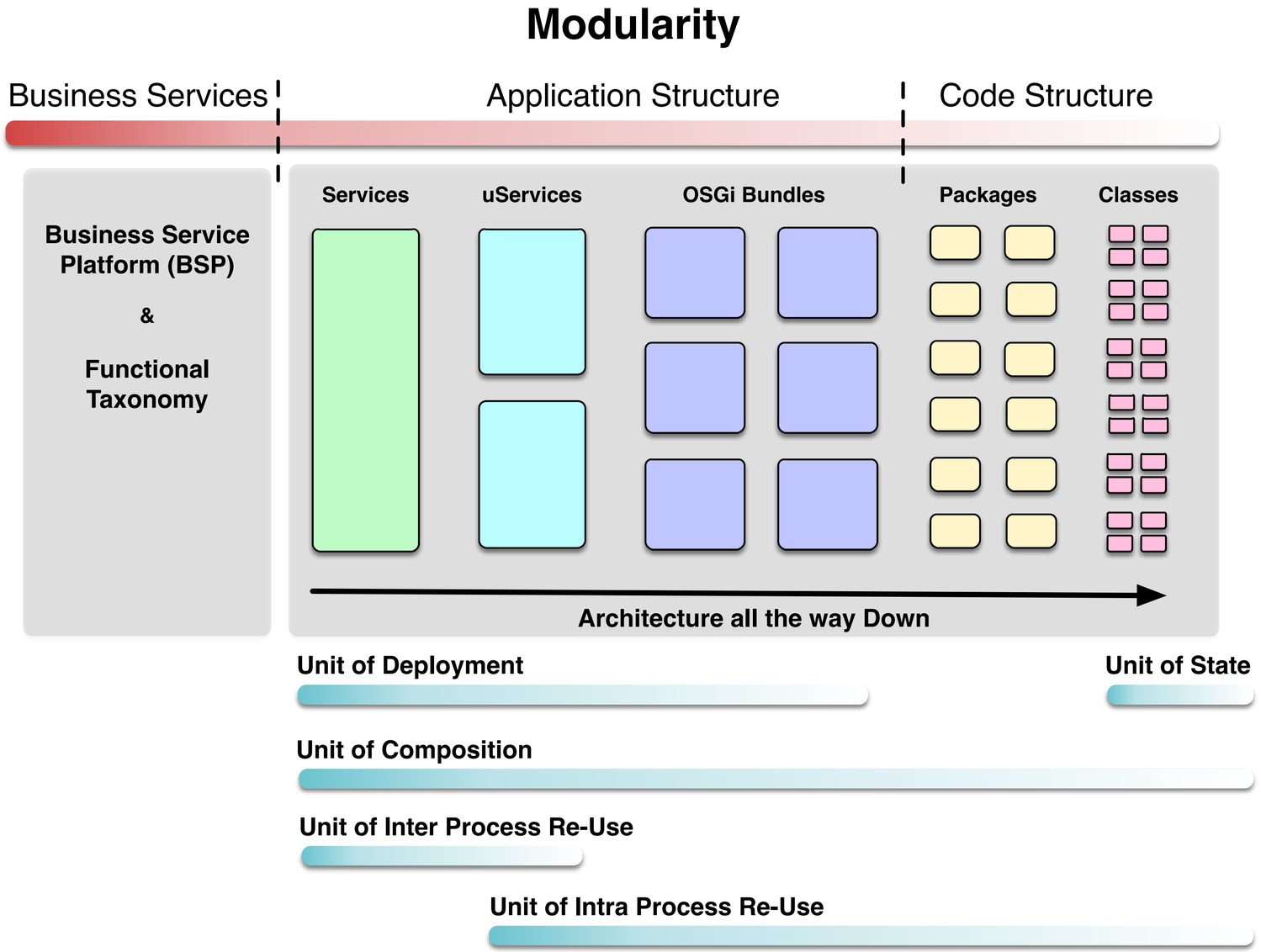
OSGi's Raison d'être





http://techdistrict.kirkk.com

OSGi - But increasingly much more….



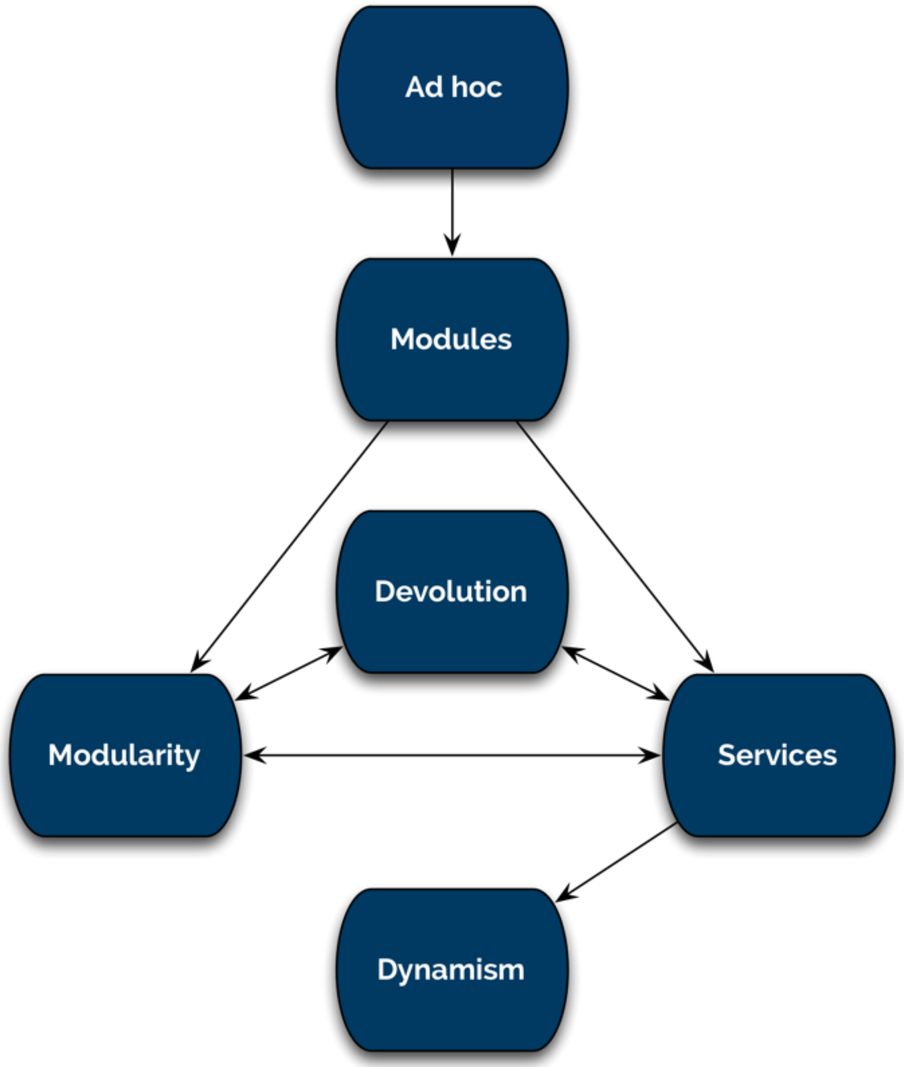
BUT NOT just Java -

- All JVM based languages.
- Non-JVM languages.

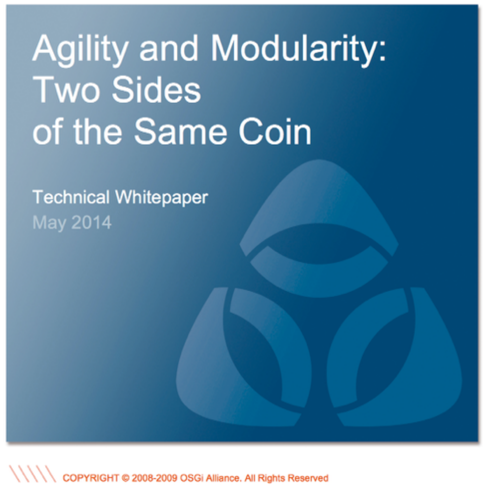
NOT JUST Code Structure -

- OSGi has a powerful μServices Architecture - e.g. Tim Ward's presentation on Async Services & Promises.
- OSGi life-cycle, metadata, deployment artefact and configuration mechanisms are actually language agnostic!

Modularity Maturity Model



Modularity also enables Agile teams and development Processes!

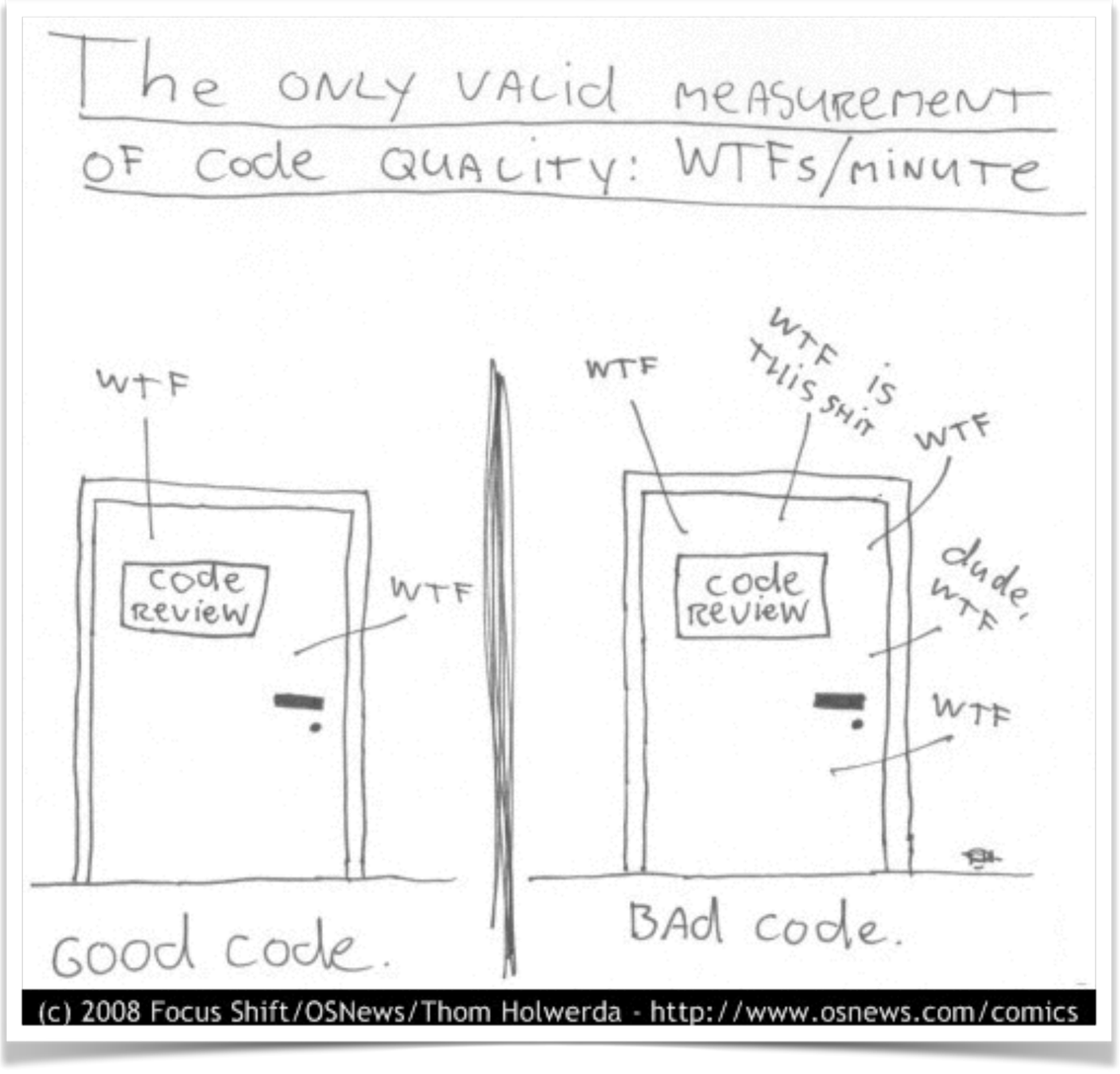


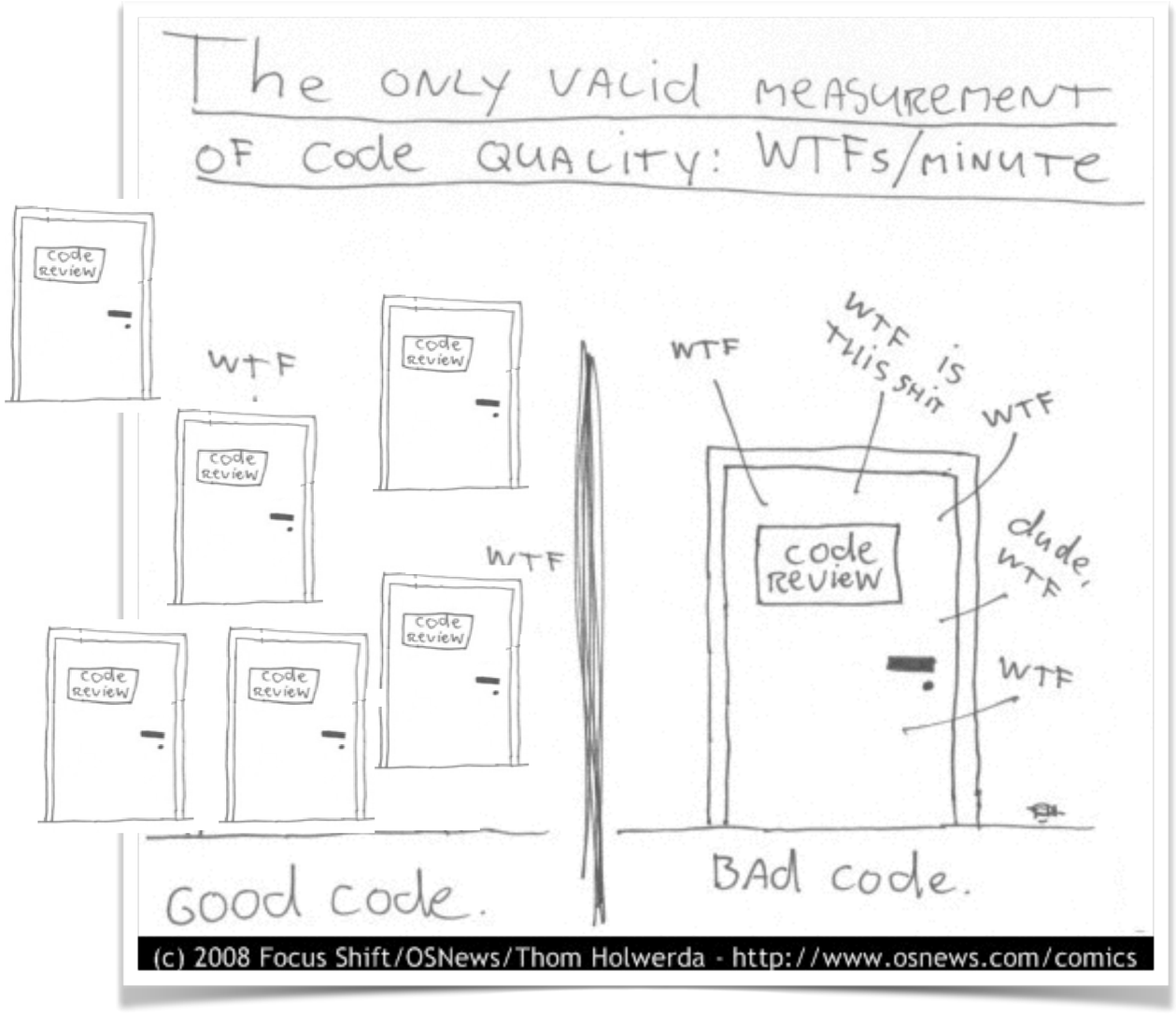
http://bit.ly/1m5sCzC

New OSGi Specifications

- Last week (week of June 2nd) the **OSGi Core Release 6 Specification** received its final approval and will be available this week to the public for downloading. http://www.osgi.org/Specifications/HomePage

- Also happening last week, the OSGi Board of Directors approved the publication of an **Early Draft Specification of OSGi Enterprise Release 6** for downloading. http://www.osgi.org/Specifications/Drafts





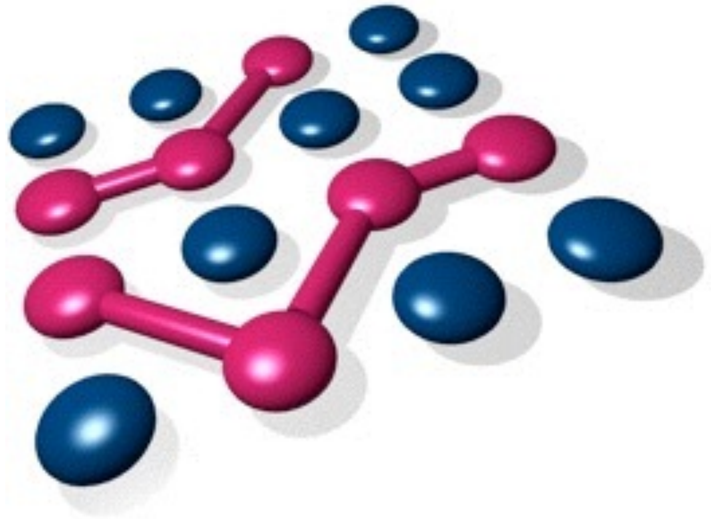
MODULAR **MONOLITHIC**

If we ignore it -
the consequences are
usually painful!

Modularity.

It's not just a good idea. It's the Law.

If we ignore it -
the consequences are
usually painful!



The Service Fabric

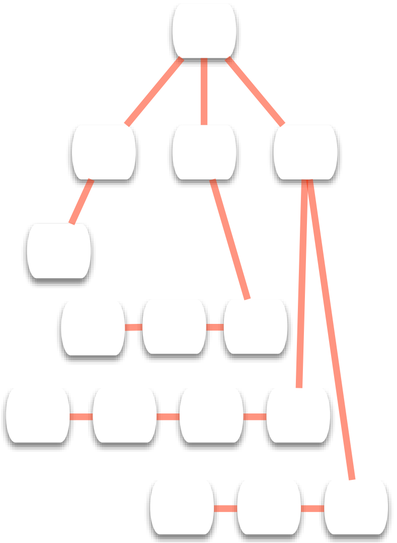
What is the Service Fabric?
Artefact and Service Modularity

- A lightweight, dynamic, distributed, modular platform
  - hosts, configures and maintains traditional applications.
  - dynamically assembles & configures highly modular applications.
- A Service centric runtime
  - '**microServices**' - deployment, configuration and advertisement / discovery of. Here the remote services implementation is fused into artefact.
  - OSGi μ**Services**! Here the remote-ing Synchronous / Async, LB-behaviour, and serialisation are configurable and provided by the Fabric!

What is the Service Fabric?
Dependency Management and Runtime Modularity

- The Fabric automates and dynamically manages **Dependencies** - Services, Bundles, Configurations and Resources
- A ‘Fabric’ is a population of ‘Fibres’:
  - A ‘Fibre’ is a ‘smart’ OSGi framework running on a JVM
  - JVM may be running in a VM or a physical resource
  - The number of participating ‘Fibres’ in a ‘Fabric’ may change over time
  - All Service Fabric management behaviors are hosted by the population of participating ‘Fibres’
- ‘Paremus ***Packager***’ extends OSGi life-cycle, metadata and configuration capabilities to any arbitrary software artifact - enabling traditional applications to be supported.

Paremus Service Fabric



The Service Fabric & OSGi

- The Service Fabric is an OSGi based Cloud Platform - introduced in 2006 - http://www.hoise.com/primeur/06/articles/monthly/AE-PR-01-06-48.html.
- The Fabric has be co-evolving with OSGi specifications since that point!
- Fabric has driven Alliance R5 resolver and more recently Asynchronous / Promises (R6) and RSA 1.1 specifications.
- In return, the Fabric has benefited from many other Alliance standards including Config Admin, DTO's and more recently the enRoute initiative.

Paremus Packager

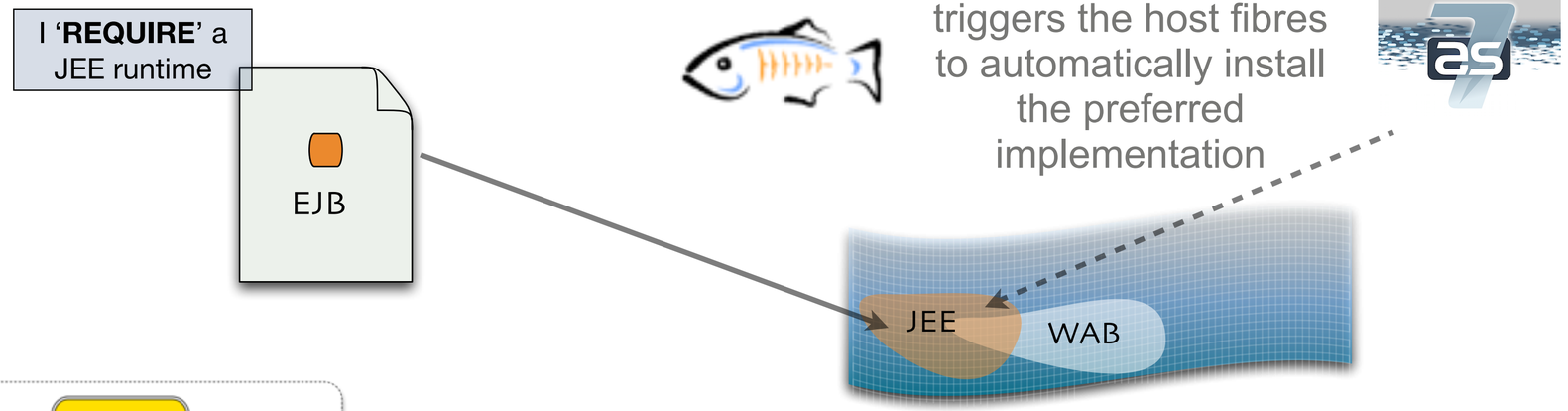
- Wrap Native Artifacts in OSGi Bundles
- Requirements / Capabilities & Semantic Versioning
- Link the Artifact Lifecycle to OSGi
  - Bundle Install/Resolve/Start => Artifact “install”
  - Service registration/unregistration => Artifact “run/stop”
  - Bundle Uninstall => Artifact “uninstall”
- Link to standard OSGi Services: Configuration Admin, Metatype, Log Service...

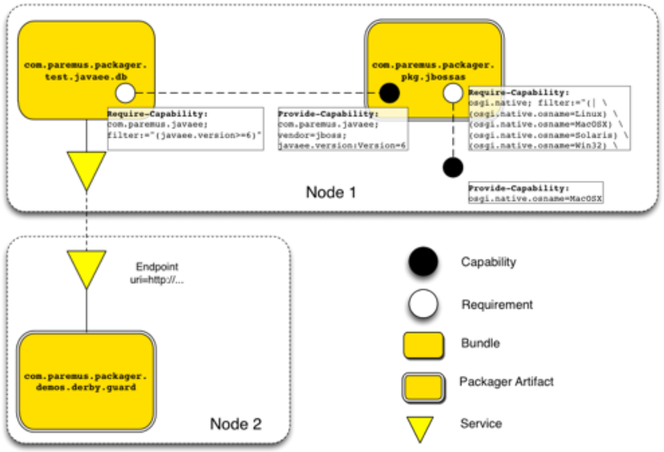
Paremus Packager

- Package programs as self-describing OSGi bundles
  - Allows Dynamic Resolution and Assembly of runtimes
  - Can be semantically versioned
- Leverages the existing bundle/service lifecycle and API
  - Allow processes to be installed and uninstalled in a running system
  - Allow processes to be dynamically started and stopped
- Allows integration with other OSGi specifications
  - Dynamic Configuration using Config Admin,
  - Use and provide Local and Remote Services
- Allow existing code to be packaged without change

An Adaptive Platform

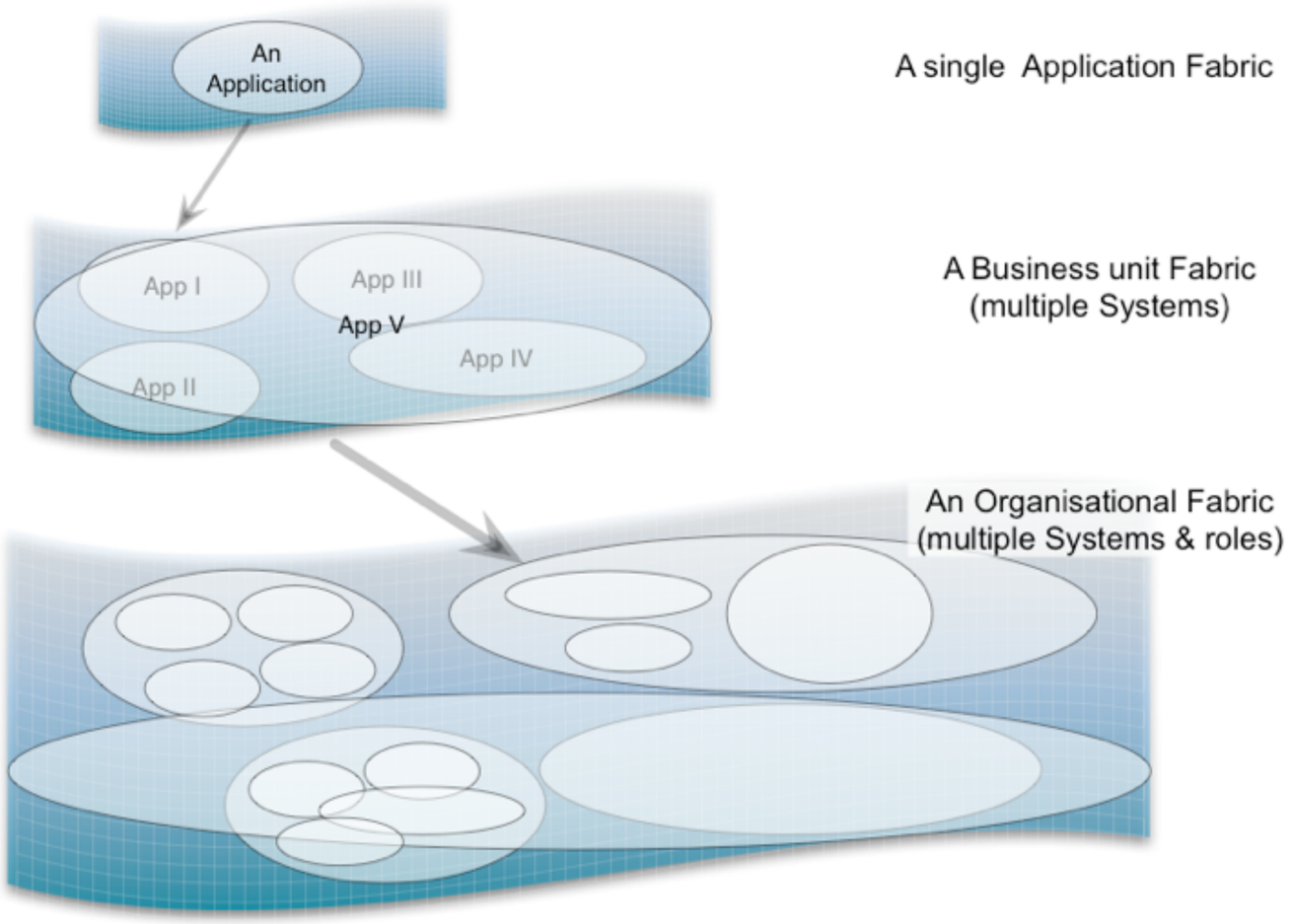
No Middleware. No Cloud Platform ‘lock-in! Really?





See - http://docs.paremus.com/x/A4EY

Runtime Modularity



Today

Fabric's may be simply and rapidly created per application; per business unit or functional area of the organisation.

Multiple Systems may run concurrently on the same Fabric

Multiple versions of the same System may run, enabling white labelling or partitioning of user population.

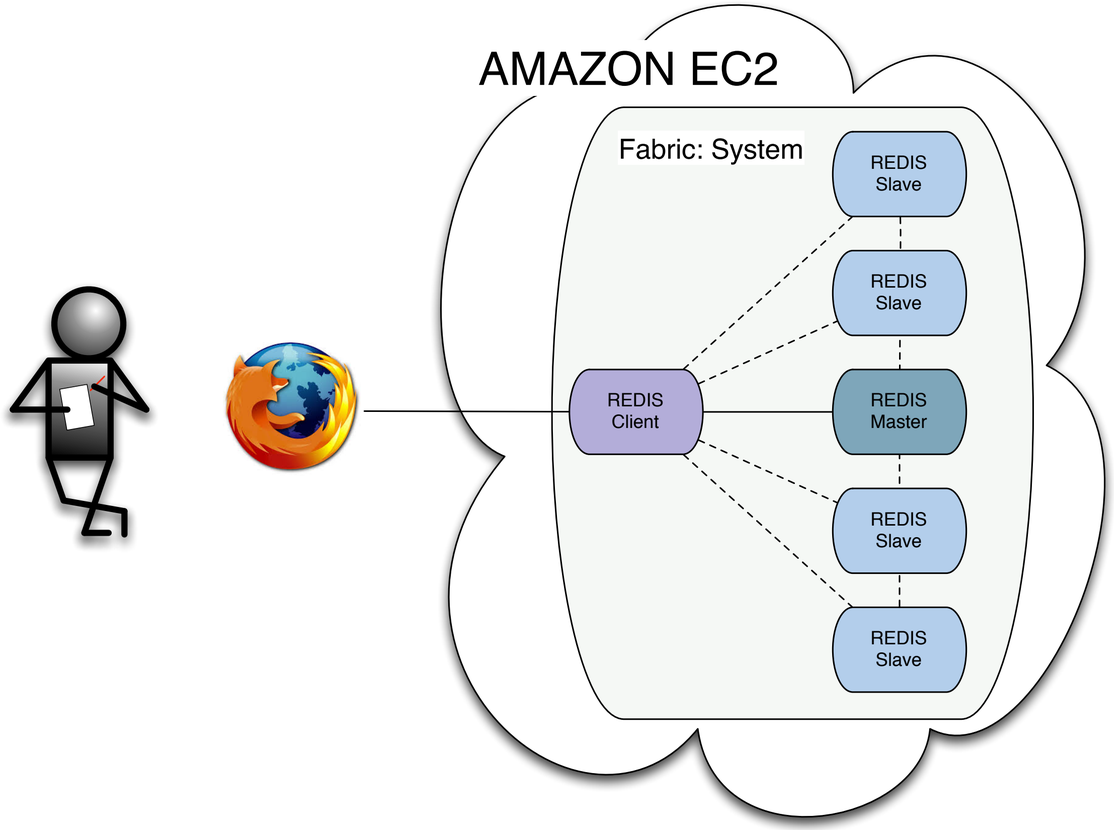
Resource contracts embedded in each System control Affinity or Aversion; i.e. whether Systems are co-located or isolated from each other in the runtime.

Tomorrow

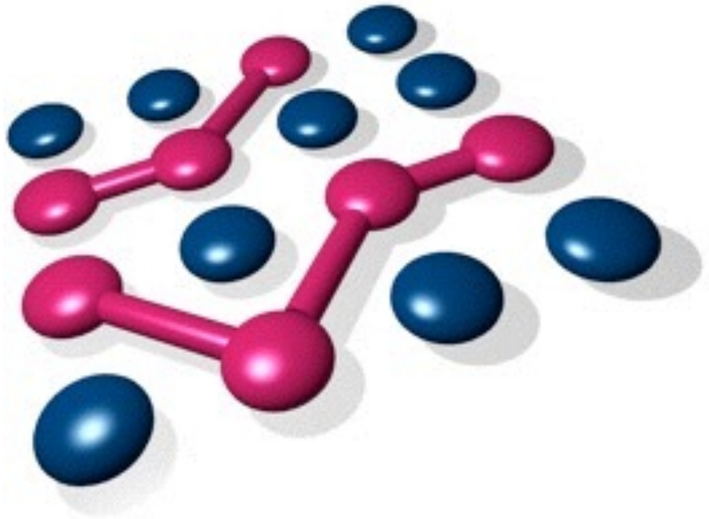
In a '*multi-System*' Fabric - roles may be used to control which Administration staff may manage which Systems *[import. deploy, configure].*

The Redis Example

Redis is an open source key-value store (http://redis.io)



See http://docs.paremus.com/display/SF110/Redis



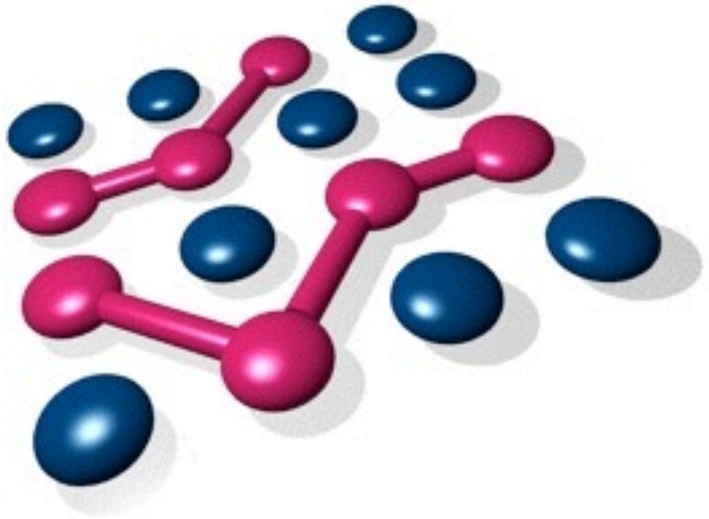
Demo Time

The Road Ahead….

- The era of the virtual machine centric Cloud is slowly drawing to a close.
- Dynamic Assembly and Adaption of runtimes - leveraging OSGi's Requirements / Capabilities - has just begun.
- The new Async / Promises specification - see Tim Ward's talk - "*Asynchronous OSGi - Promises for the Masses".*

  The start of a powerful unfolding story for μ**Services** based architectures.

Paremus



Thank You

www.paremus.com @Paremus info@paremus.com