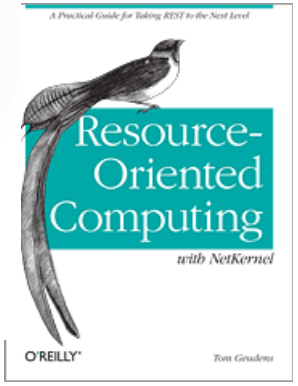
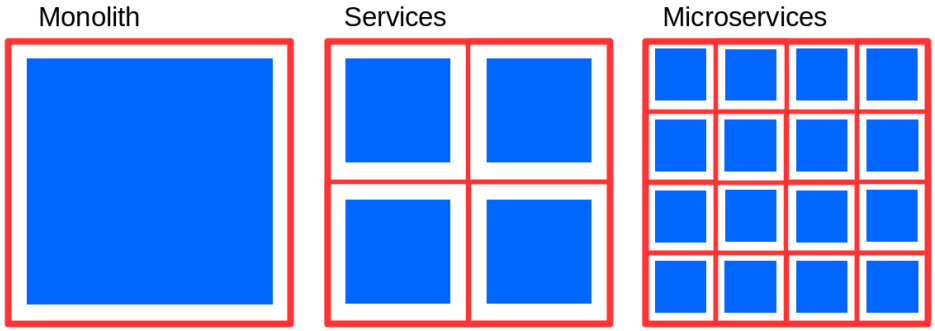
Resource Oriented Computing

goto conference; London

Peter Rodgers
September 2015

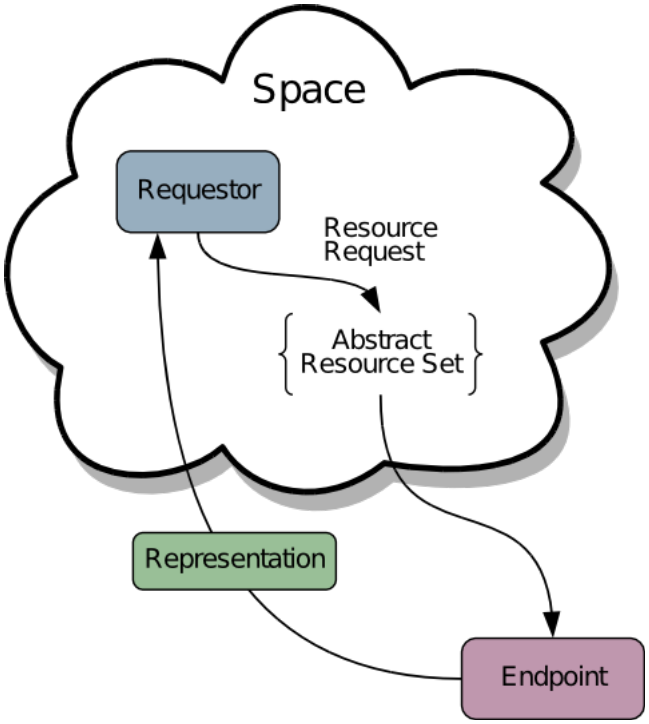


Trend to MicroServices



- Unix philosophy - make simple well formed things.
- Compose the things to create new things.
- In engineering terms: a composite's value is greater than the sum of the parts
- **But** for Microservice read "MicroResources", but what is a "Resource"...

Resource Oriented Abstraction (WWW / REST)

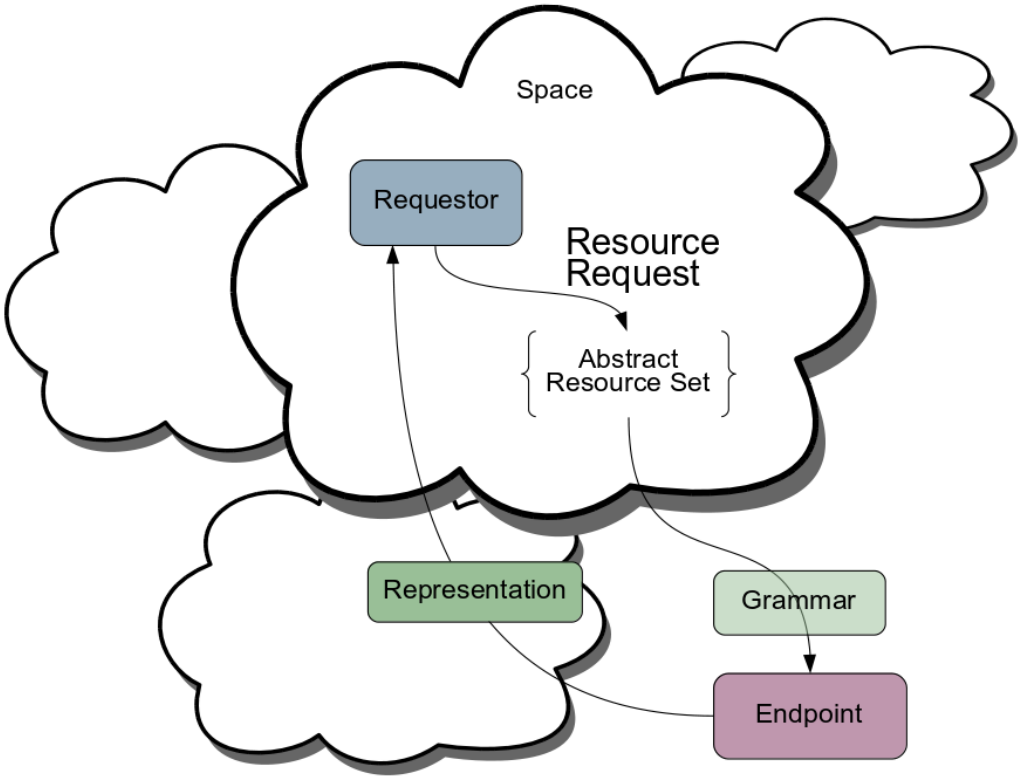


- Resources are logical abstractions.
- Objective is not to run code, objective is to obtain a representation.

We ❤ WWW but ...

- A single flat address space
  - Every endpoint is a peer.
  - Every microservice has to manage its own security, scaling, availability...
  - Stateless = good, but: stateless = no context && no context = bad.
- As we move to compositions of microservices, how do we debug them, measure them, deal with failures?
- How do we manage the state of composite resources? Scaling and caching that works for a page based model of the Web no longer works.
- HTTP is great but... (whisper it) its not actually Resource Oriented.
- What if we had a pure Resource Oriented abstraction...

Resource Oriented Abstraction (General)



~~Resource Requests~~ Do it later...

NetKernel Demo

```
active:xslt
+operator@res:/transform.xsl
+operand@res:/data.xml
```

<NetKernel powered by>

Issue Request

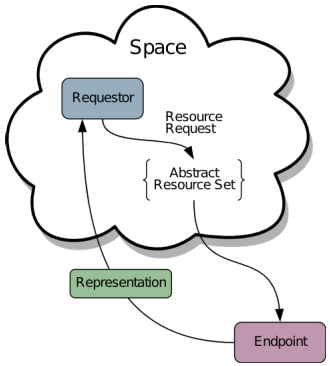
Presets - Raw Identifiers

- Hello
- http://www.google.com
- active:uppercase
- Hello Code/Languages
- Active URI
- Active URI (2)
- Functional URI
- Abstracted res:/greenbox
- Non-local Computing
- DPML RDF-Pipeline

Presets - Declarative Requests

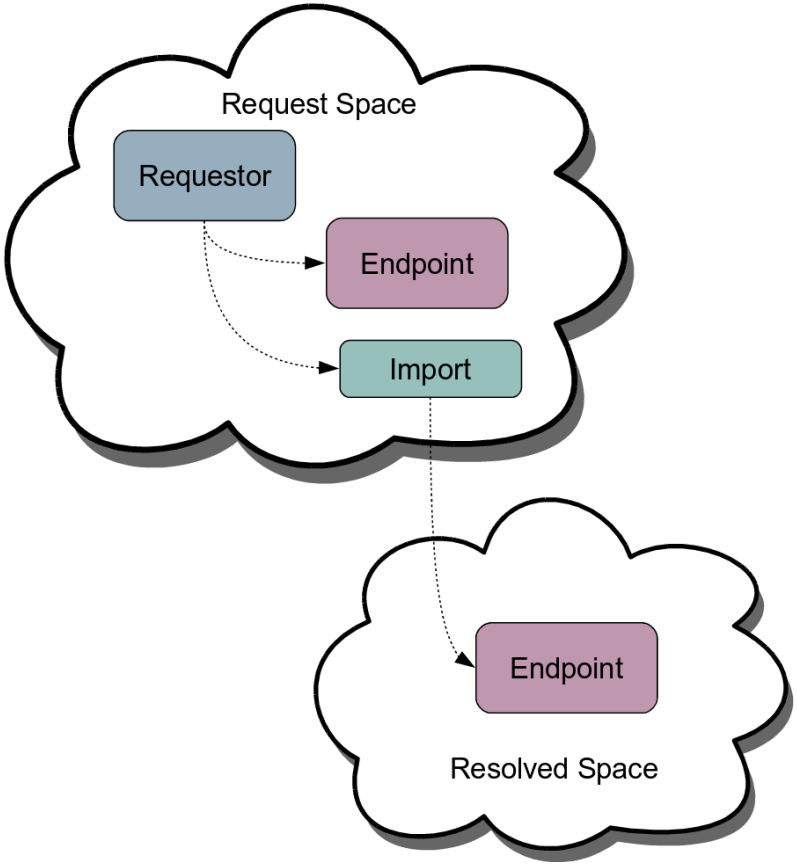
- Hello
- 2+2
- http://www.google.com
- fib(5)
- Active URI
- Literal Arguments
- Active URI (2)
- Functional Requests
- Active Groovy Literal
- DPML Literal RDF Pipeline

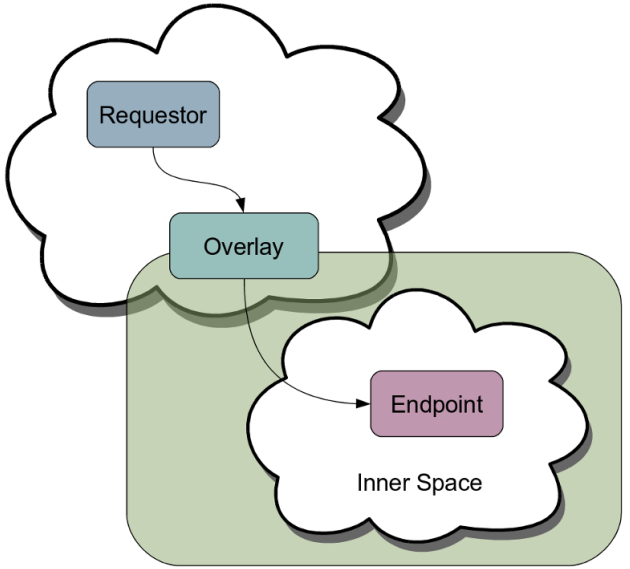
Presets - Declarative Requests Abbreviated Syntax



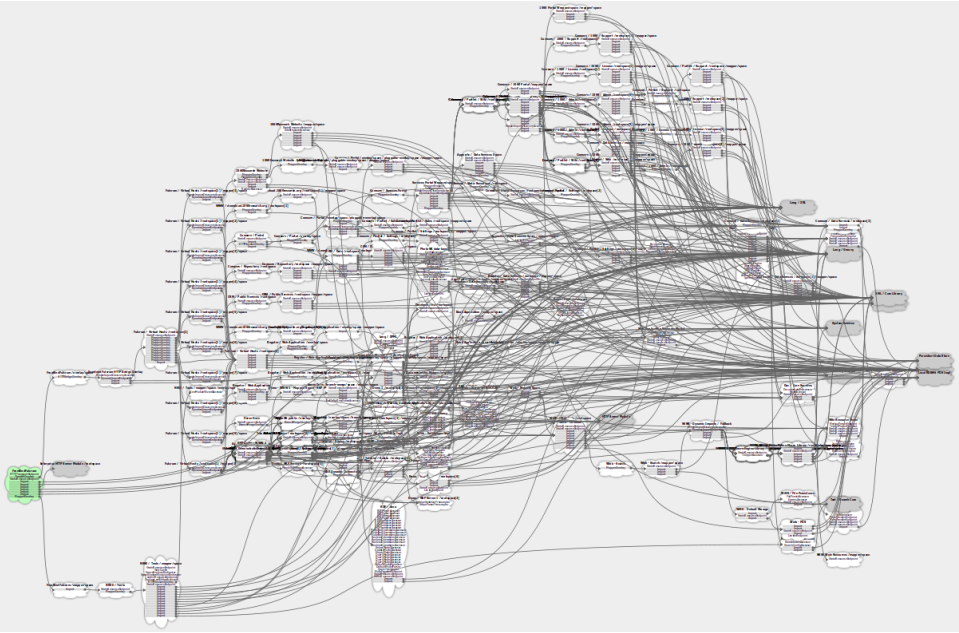
reset "In NetKernel Apposite install “demo1”"

ROC Architecture





Scale



- How many microservices have you got?
- There's a resource for that...

active:moduleStats

New Tools Needed

- In the web things aren't simply loosely coupled, they're **decoupled**.
- How can we see if its working?
- How can we fix it when its not?
- How can we measure the performance?

Visualizer

ROC Performance

- 2-phase computation
  - Resolution
  - Execution
- Performance must suffer?
- **No** performance improves!
- "Loadbalance Inside" - linear scaling on multicore
- But there's more ... What if you cache everything?
- Cache in every dimension simultaneously...
- Live - State Distribution
- +++ you can do better than time-based expiration...
  - Resource Dependency Model...

~~Composite Resources (+ Dependency Caching)~~ Nothing to see here...

[index]

Composite Resource - Hi BYU

Demo Resource 1 - Hi BYU!

12:52:28

Demo Resource 2 - Hi BYU!

12:52:28

Demo Resource 3 - Hi BYU!

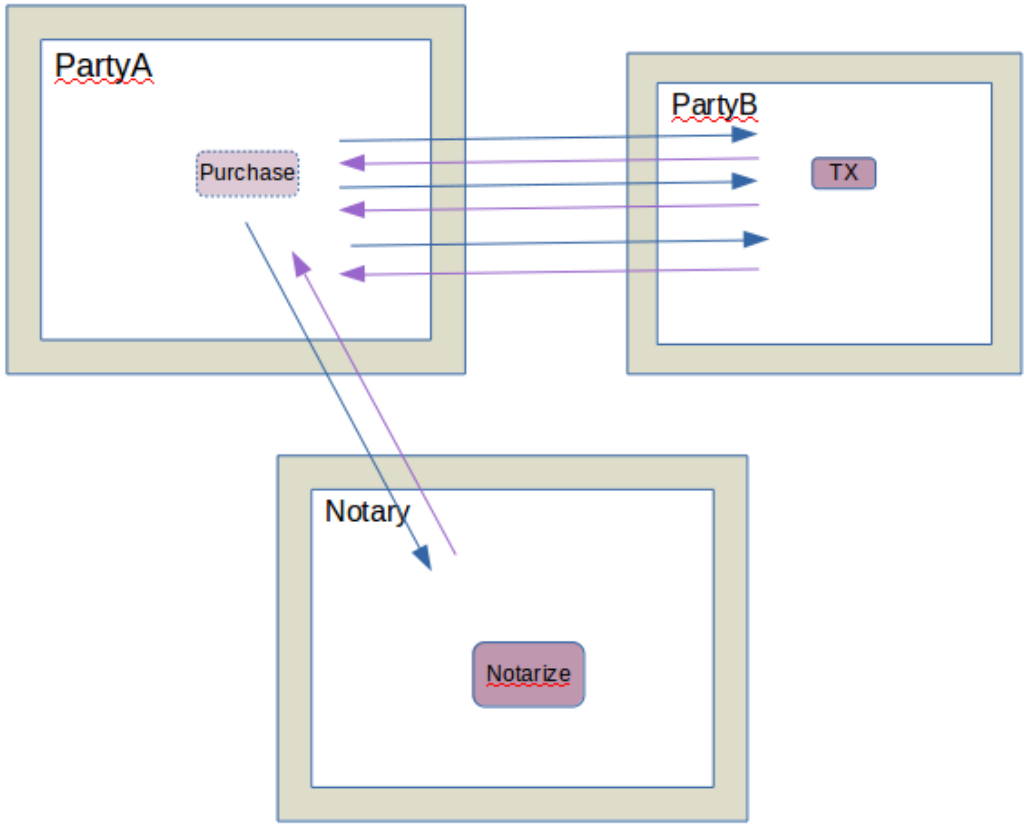
12:52:28

Demo Resource 4 - Hi BYU!

12:52:28

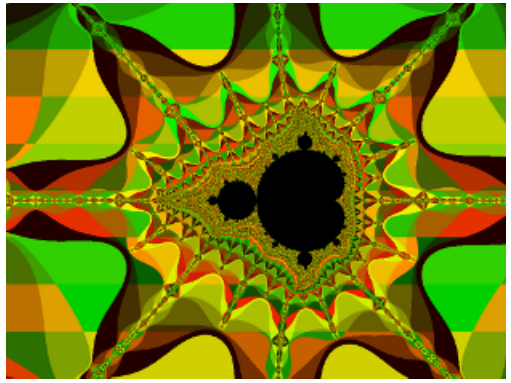
Fullscreen

~~Measurable Economic Impact: N-Party Interaction~~ Move on...



~~Distributed ROC~~ No time...

NetKernel Protocol Demo



Demos

- Local
- Remote*
- Remote Runtime*
- Spanning Cloud

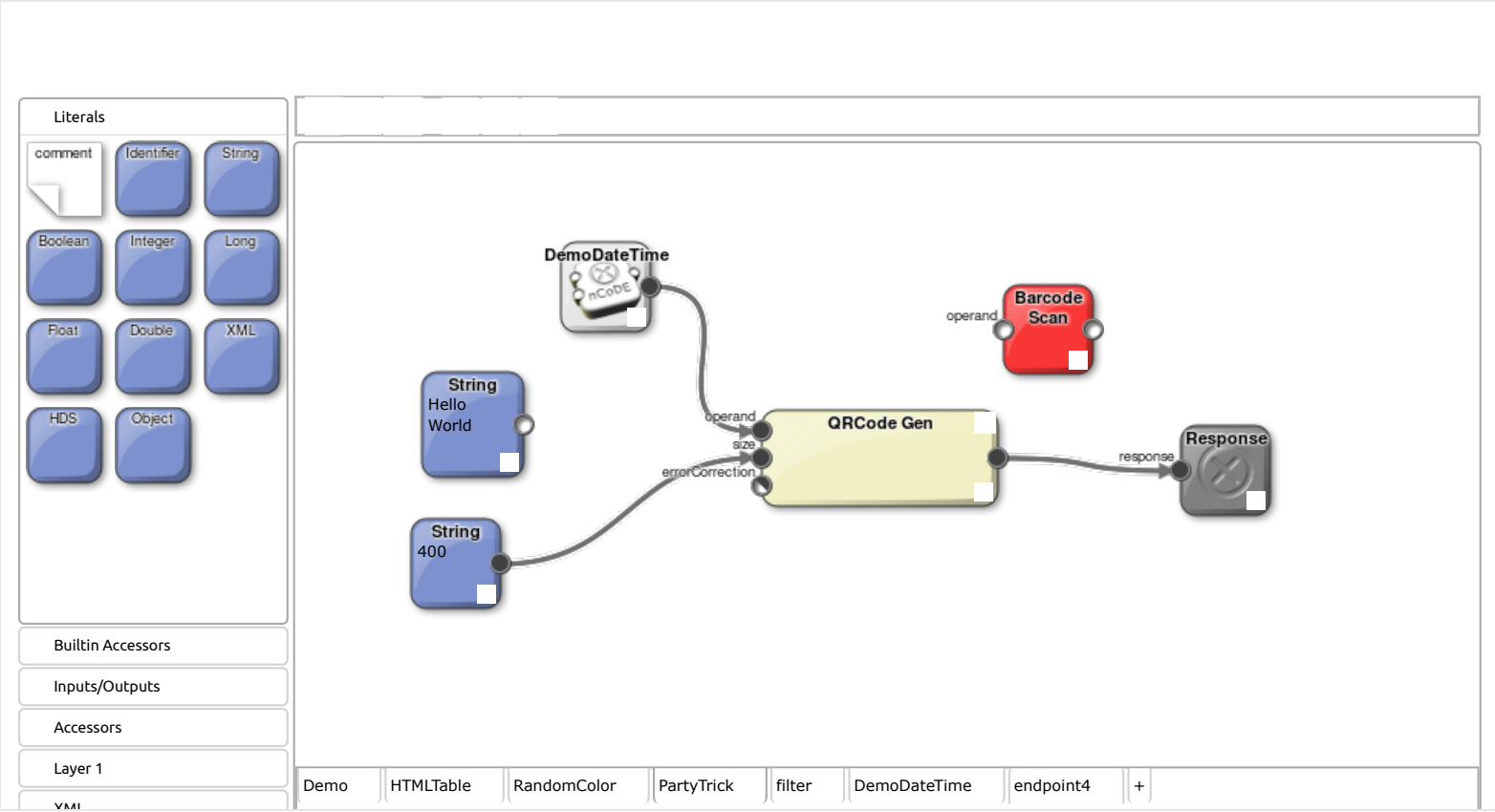
Set NKP Demo Cluster Credentials

* Requires access control credentials (see documentation for details)

<NetKernel powered by ®>

Fullscreen

nCoDE - Visual Functional Resource Composition



Demo1 Party Trick

ROC: Reaping the Economic Dividend

- ROC Architecture is 100% decoupled (not simply loose coupling)
  - Hot-swappable
  - Legacy coexistence
  - Genuine reuse
  - Unlimited evolvability
- Hugely cheaper to develop
  - 80% of a problem is solved by composition of existing tools
  - Very easy to change/evolve - recomposition.
  - Powerful engineering levers available (throttle, one-way-trapdoor...)
  - Simplified configuration management: "Everything is a resource"
  - Logging "the crime scene" is redundant "execution state is a resource" Visualizer
- Provable Security / Trust
  - Constraints are spacial boundary conditions
  - Trust and non-repudiation
  - Validation, Semantic integrity

...and higher performance too!

ROC

- Radically increases **Attainable Scale** of Software
- Introduces **engineering qualities** to complex systems.
- **Huge performance gains** - Systemic Memoisation (Caching) and Async Linear Scaling
- Changes Economics of Software => **Eliminate Saw-Tooth, Track the Exponential**
- Brings the **Web Inside** and makes it general purpose.

NetKernel v5.2.1

- The **Uniform Resource Engine**
  - General Standalone Application Server
  - Embeddable as "ROC Engine"
- Proven with hard-core, carrier-class deployments
  - Telecoms
  - Black Friday Retail
  - Huge dot-com platforms
  - Core Web Instracture - PURLs, Dublin Core
  - Government Open Linked Data

Reference

- NetKernel Resource Oriented Computing Platform is developed by 1060 Research and is published under a dual-license open source model.
- Onsite Training and Consulting in Resource Oriented solutions is available from 1060 Research
- 1060 Research: profitable, 10 year, low-profile, hard-core infrastructure business.

Contact

- email: pjr@1060research.com
- twitter: @netkernel

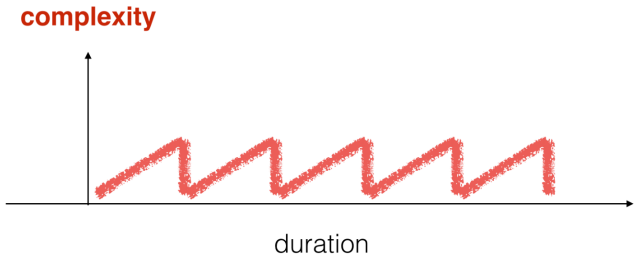
<1060® research>  <NetKernel powered by™>

The stuff we won't have time for...

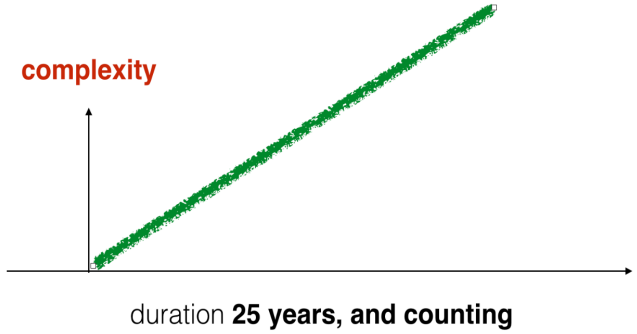
Background

- Peter Rodgers - originally a Physicist. 1995: Hewlett-Packard Laboratories
- Research Ambitious Internet Scale Systems

Why is software so brittle?



Yet the WWW keeps growing?

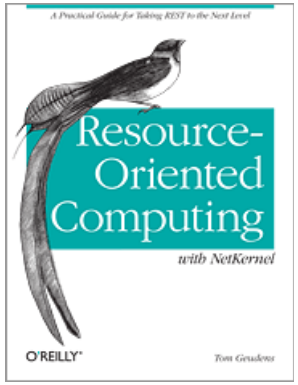


History of ROC

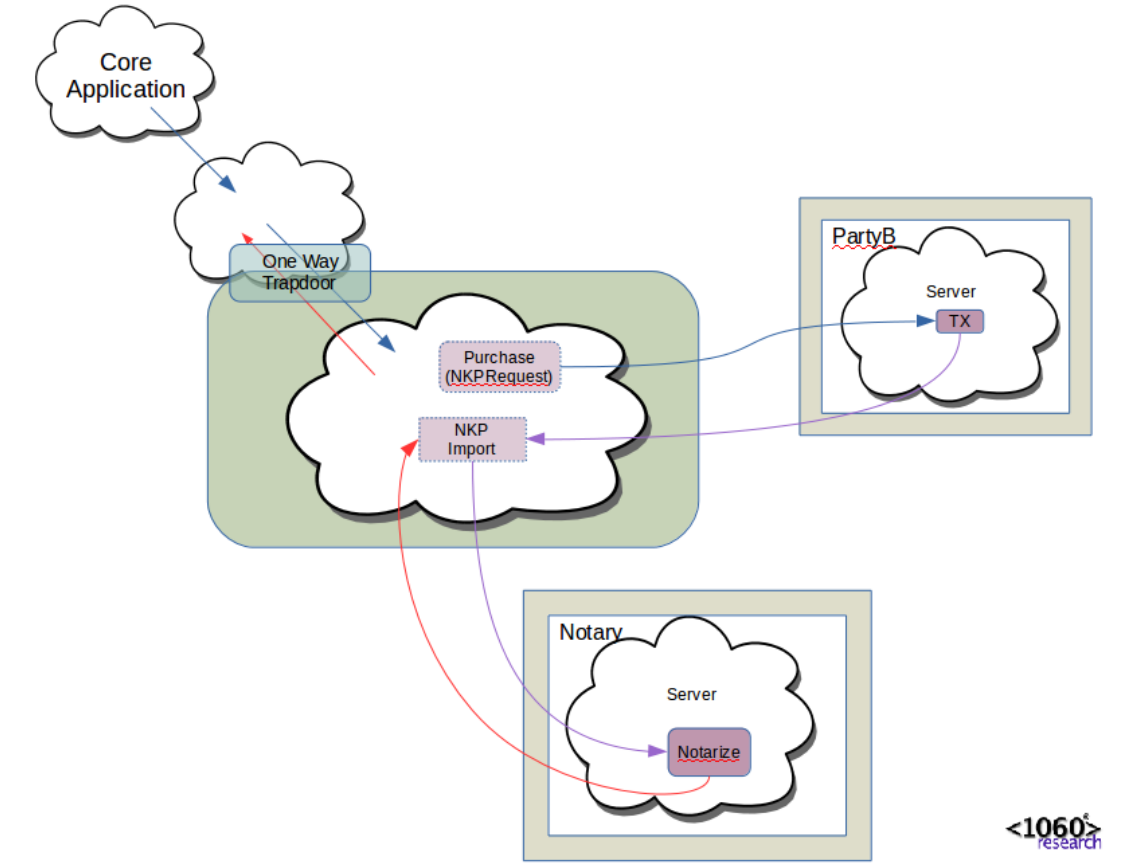
- "Build another framework" doesn't cut it. **Back to first principles...**
- What if we **really understood the Web?**
- What if we could **tap the economics in general?**
- Late 90's researched concepts of REST (before REST)...
- **Generalized to ROC. Discovered new world of possibilities.**

Timeline

- 2002: Founded 1060 Research
  - Developed ROC NetKernel
  - Matured technology in production
  - Patiently waited for market...
- 2010: Awareness of REST began to build
- 2012: Resource Oriented Computing with NetKernel O'Reilly book.
- 201x: ROC, what happens beyond REST...



Measurable Economic Impact: Security Analysis



Extrinsic Recursive Algorithms

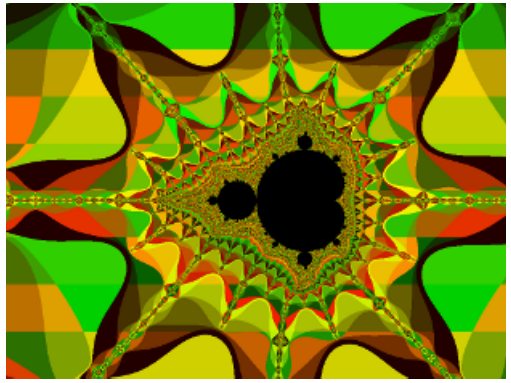
Fibonacci Demos

- Fibonacci Double Recursion
- Ackermann Function

Demo Visualizer P v NP ROCing the Cloud

~~Distributed ROC~~ No time...

NetKernel Protocol Demo



Demos

- Local
- Remote*
- Remote Runtime*
- Spanning Cloud

Set NKP Demo Cluster Credentials

* Requires access control credentials (see documentation for details)

<NetKernel powered by>

Fullscreen

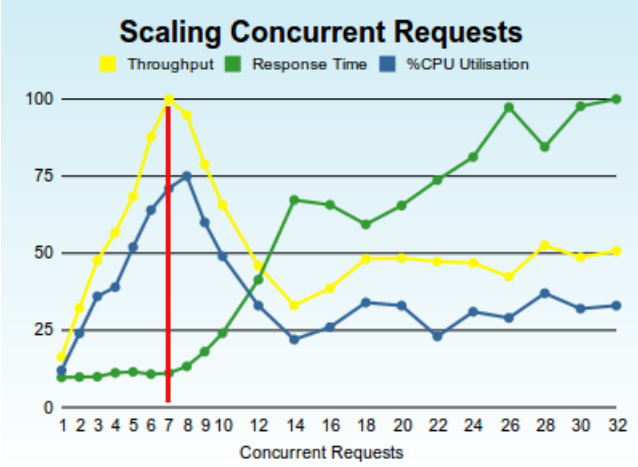
Web-Scale Capabilities of your Dreams

- Distributed Track-n-Trace
  - Sticky Headers
- Non-Repudiable Injection Attack Elimination
  - Easily shift processing to the **structural-tree-domain** away from the vulnerable **serialized-stream-domain**
  - Tree-structure is provably invulnerable to injection attacks.
- Mapper Patterns for true Mathematical Functions
  - Injections, Bijections, Surjections.
- Transrepresentation (Transreption)
  - True content negotiation
  - Linearizes the N^2 complexity type conversion problem
  - Unifies previously distinct historical CS areas Compiling, Parsing, Serializing etc etc
  - Entropy transforms
- Spacial Scope Manipulation
  - Dynamic inversion of imports
  - Contextual spacial structure
- Space Runtime
  - When everything is a resource - what happens if spaces are resources too?
  - Turtles all the way down architecture.
  - Emergent transient architecture
- Metadata-driven Architecture
  - Resources that direct resources
- Linked Data Architectures
  - The amazing conseqeuences of Push-Pull inversion
- ROC Patterns
  - Brand new patterns with no-analogue in OO, imperative or functional code.

Software Load Lines

Live System Data

Cloud Platform - Top of the Range Instance



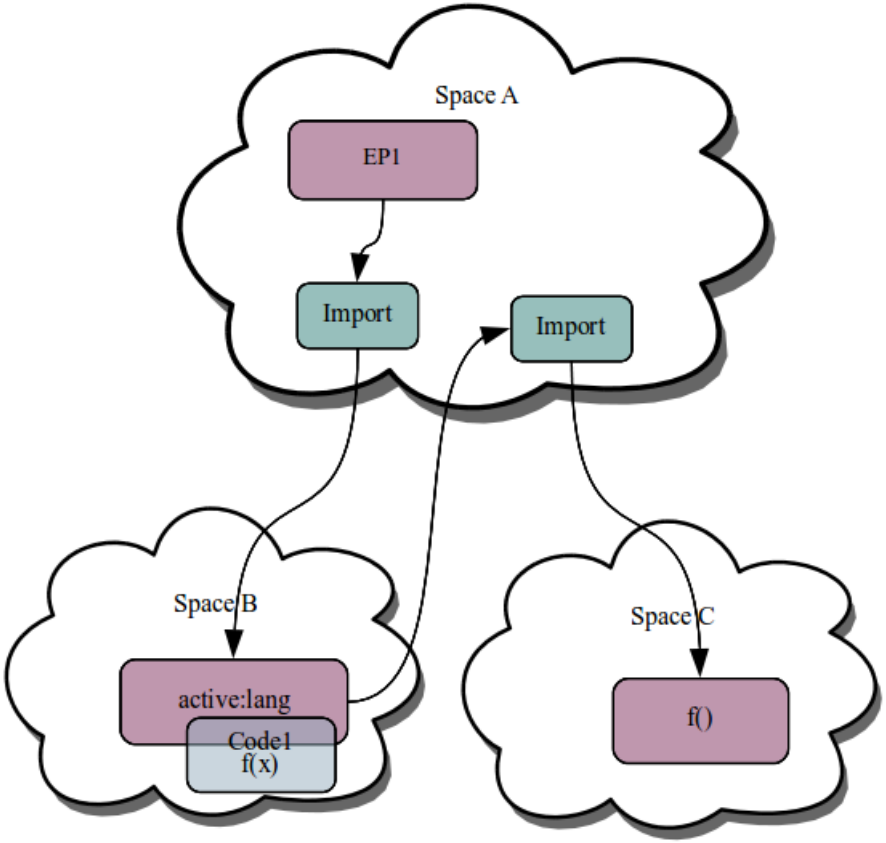
Article: ROCing the Cloud

The QRCode Clock

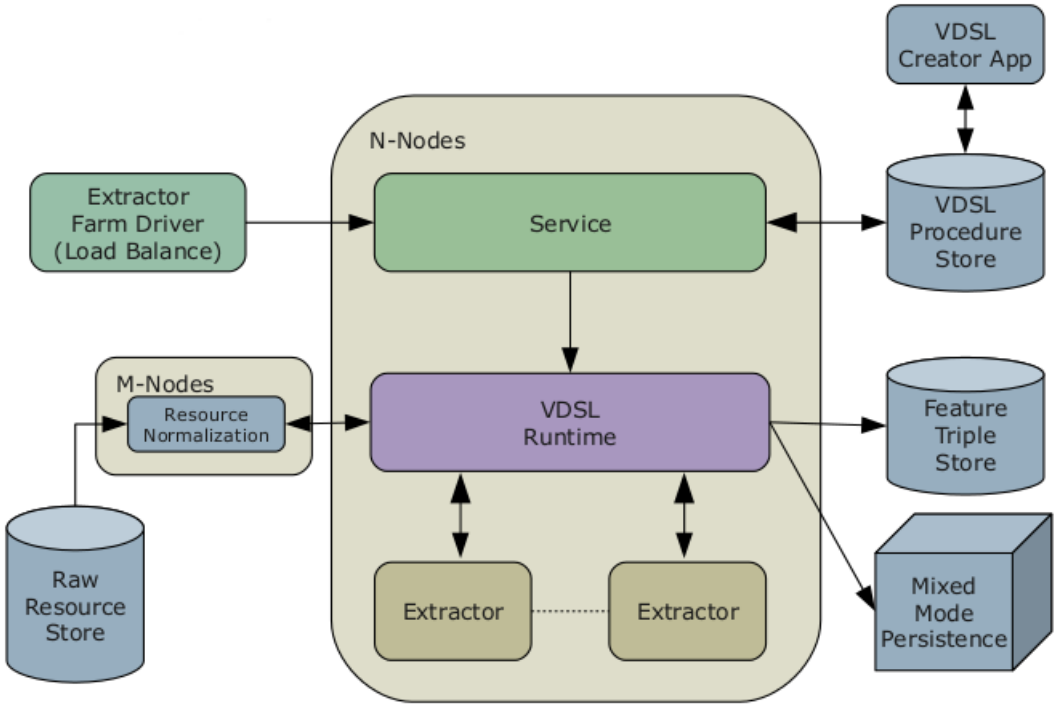
Fullscreen

Decode

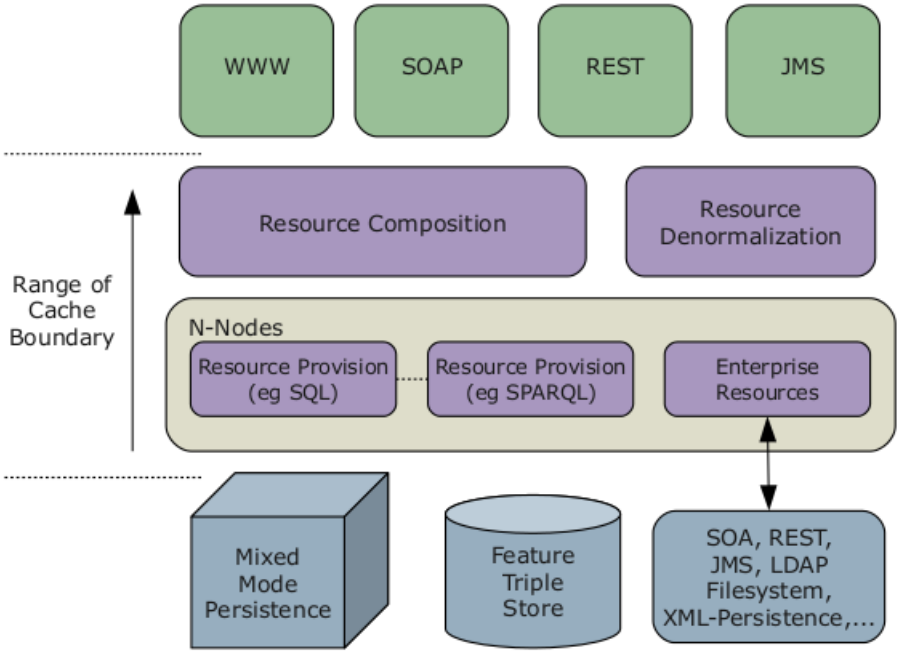
Language Runtimes



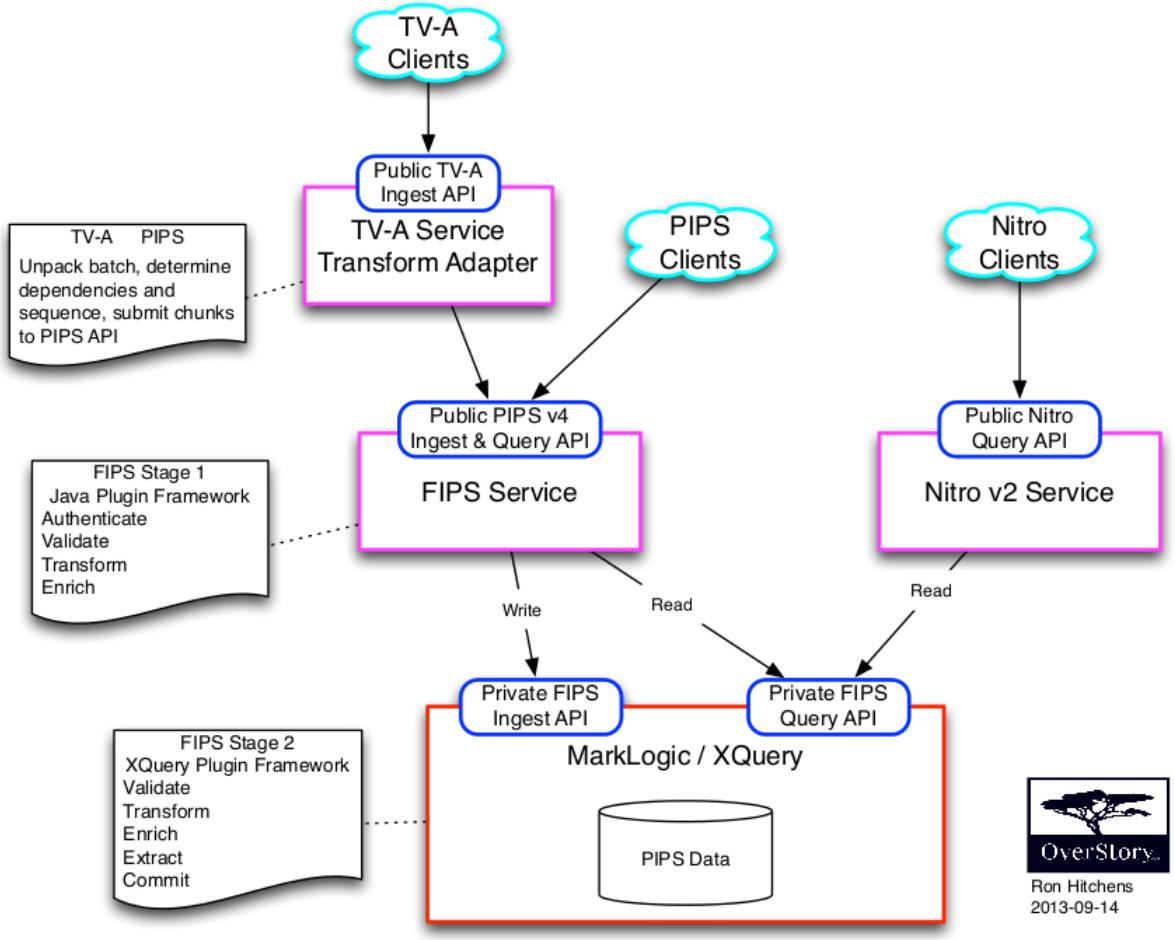
Linear Scaling, Dynamic Composable, Compute Farm



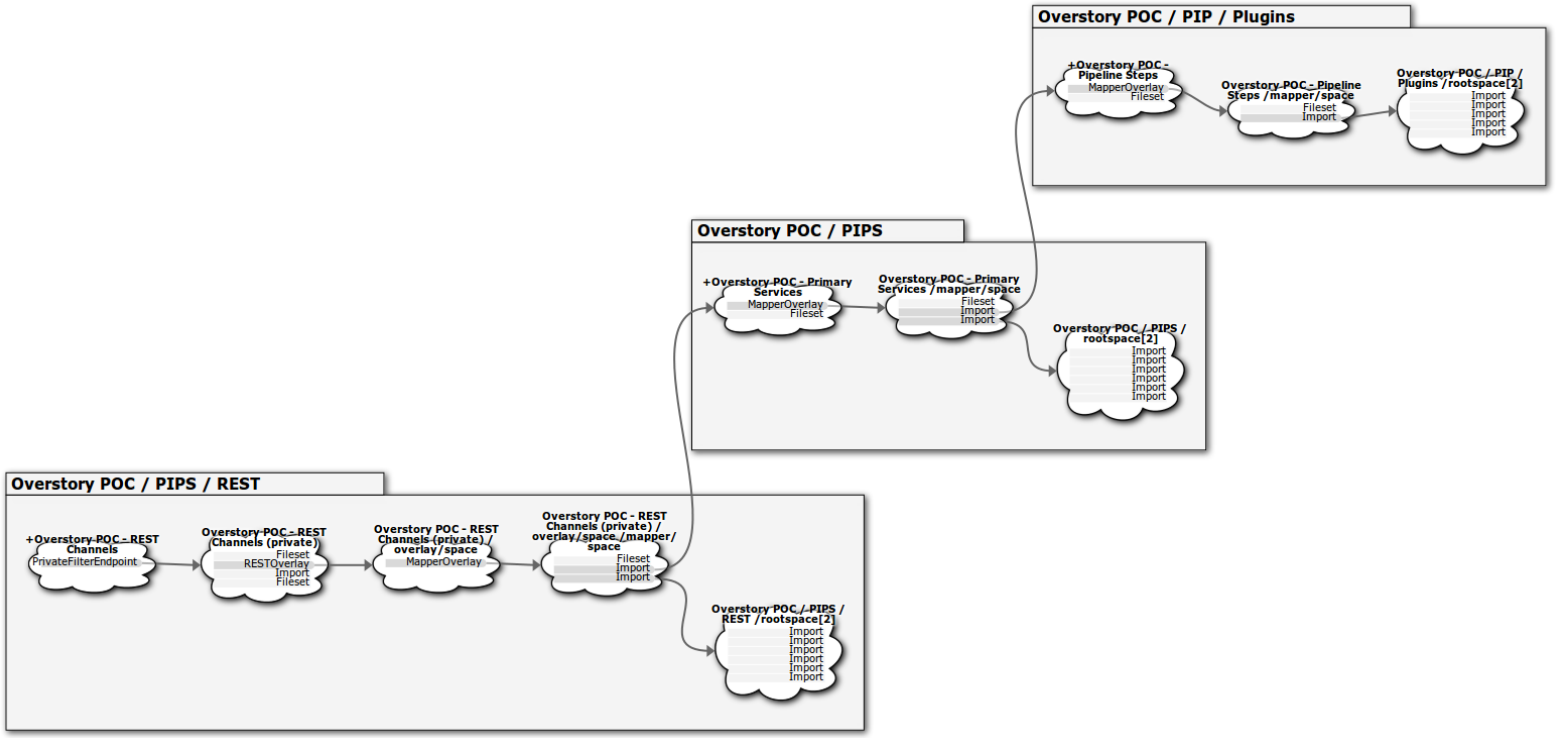
Compositing Denormalisation Platform



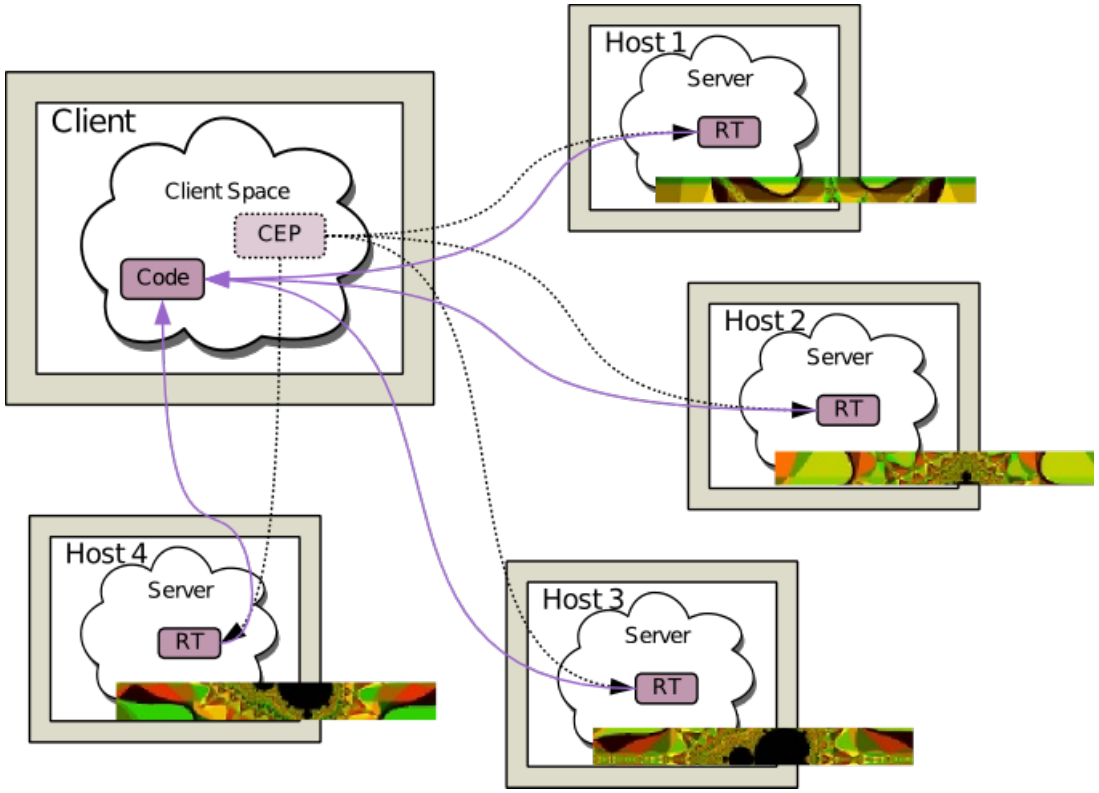
PIPs POC - BBC, Overstory



PIPs POC - BBC, Overstory, ROC

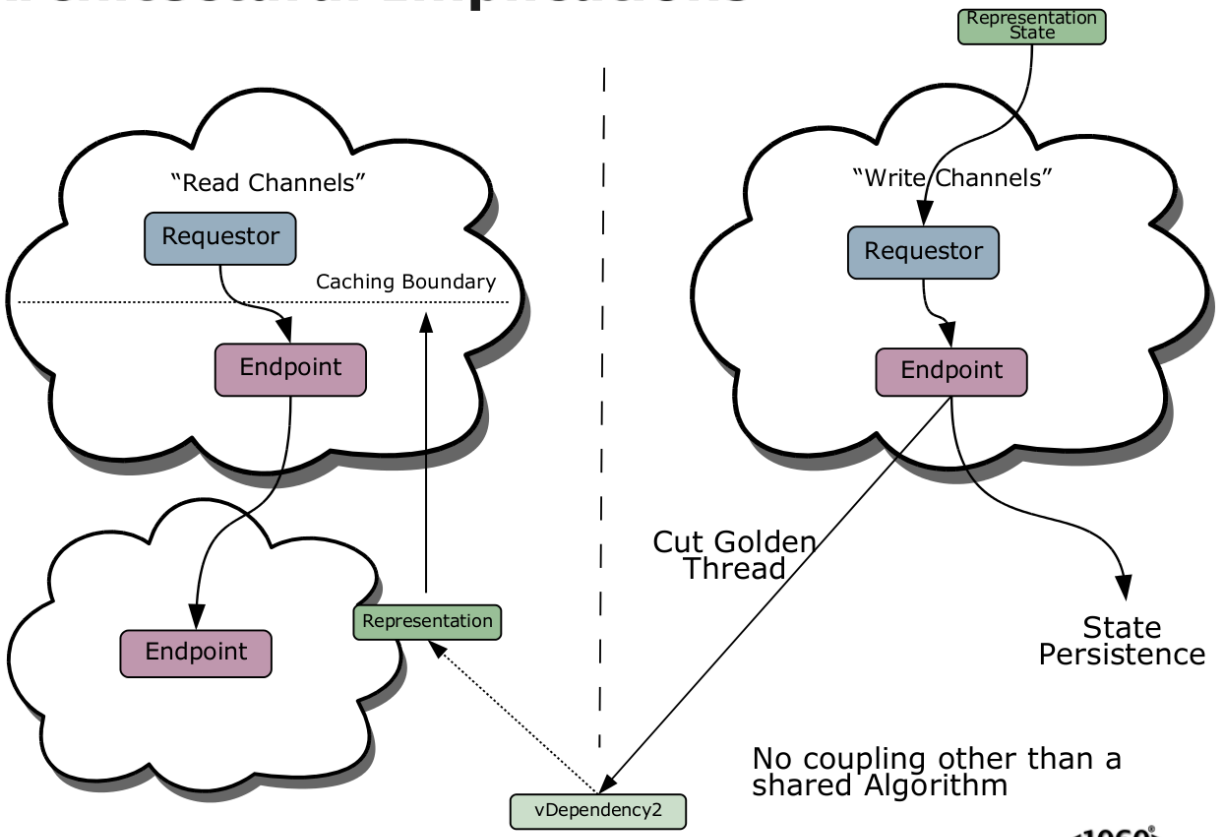


Cache Coherent Distributed Runtime Cluster



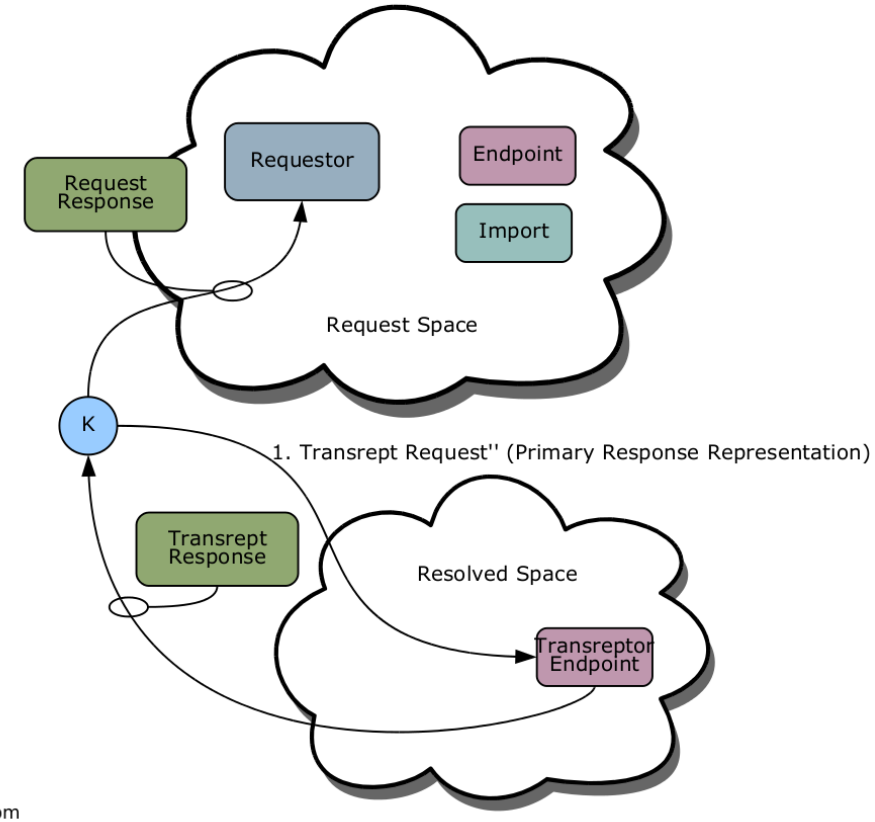
A whole world of new ROC Patterns...

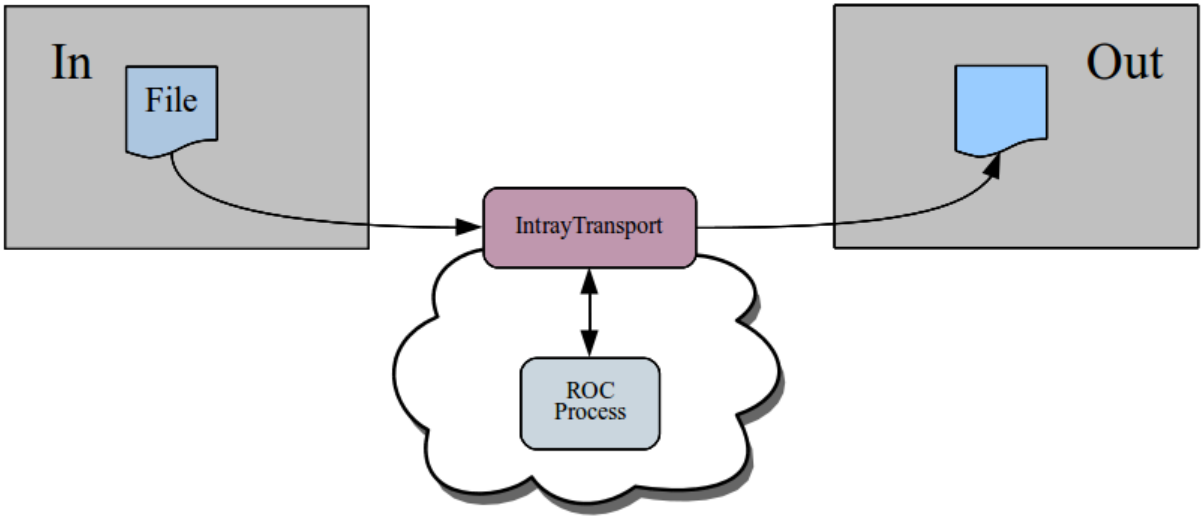
Architectural Implications



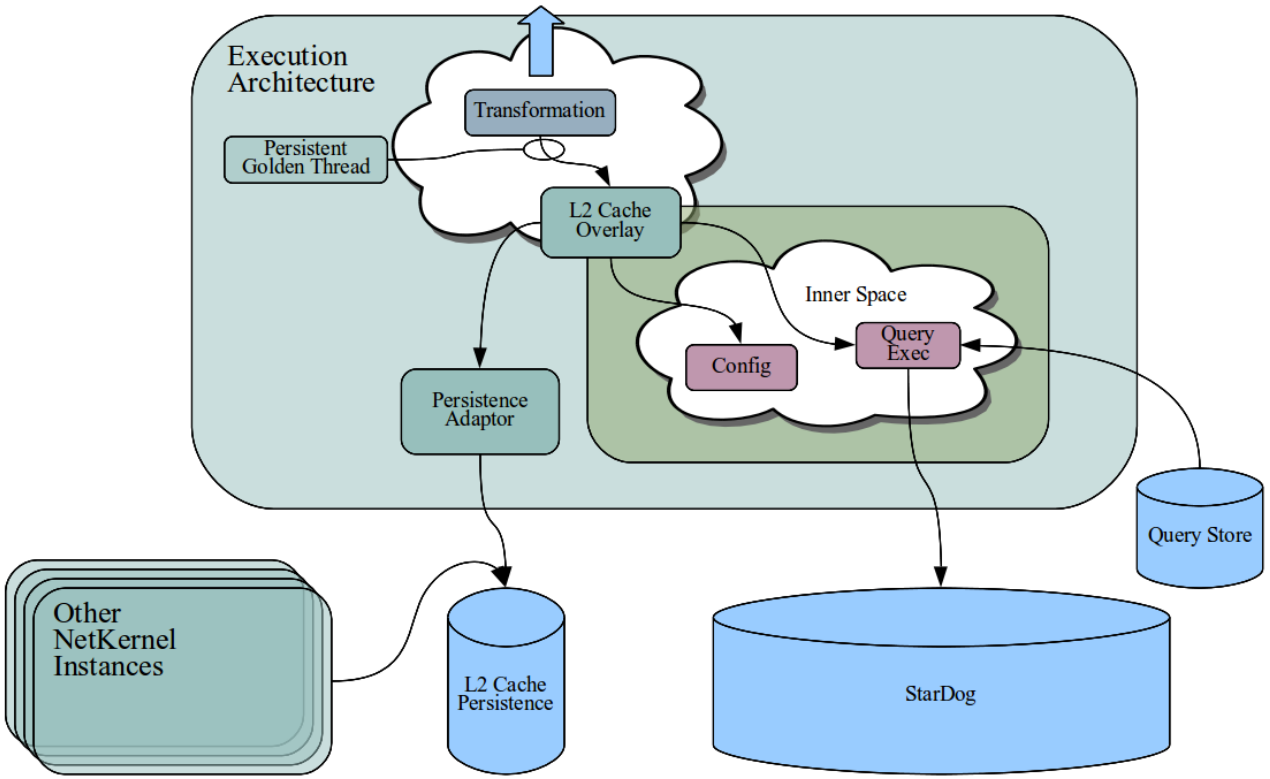
www.1060research.com

Transreption Evaluation

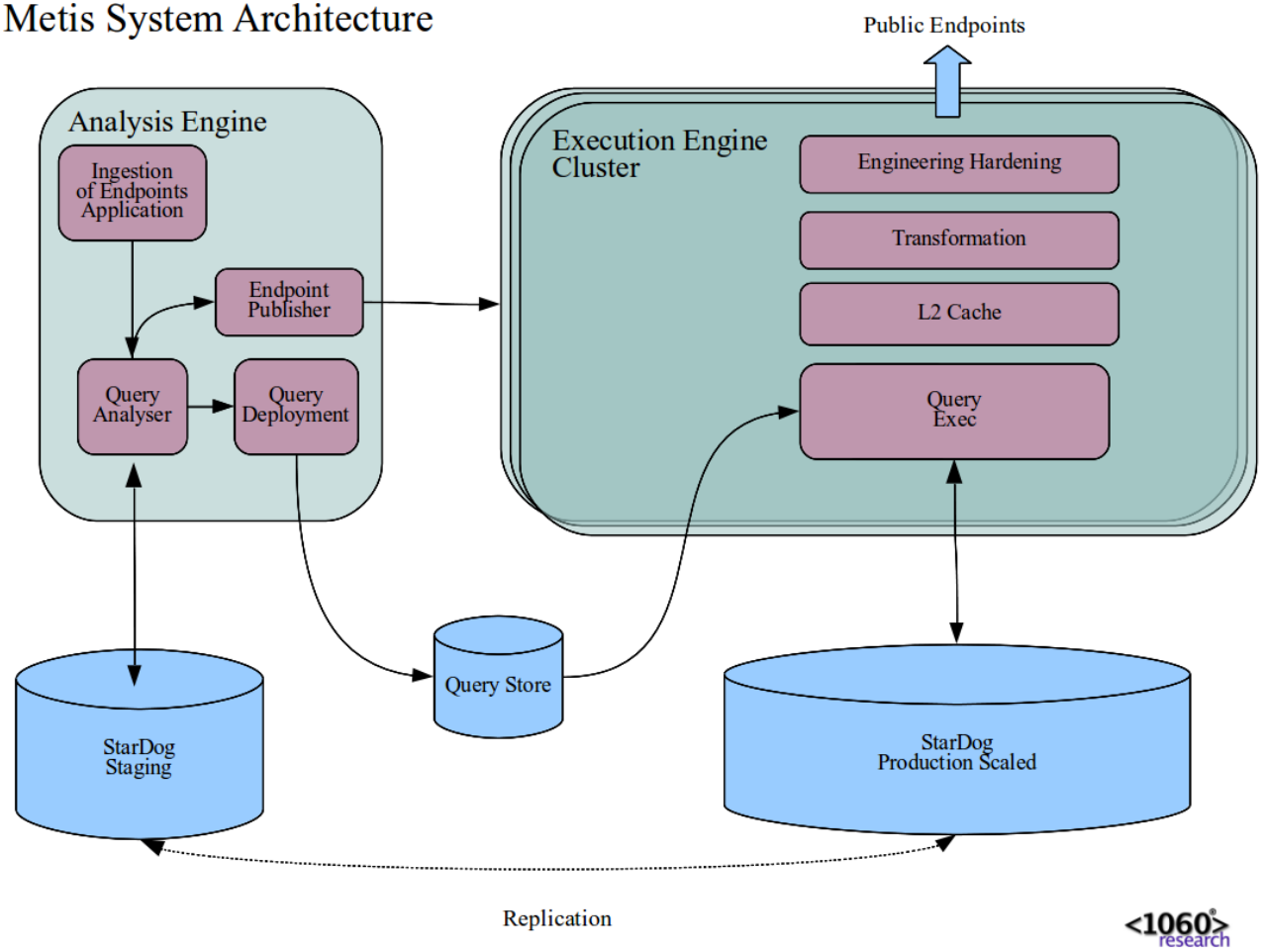


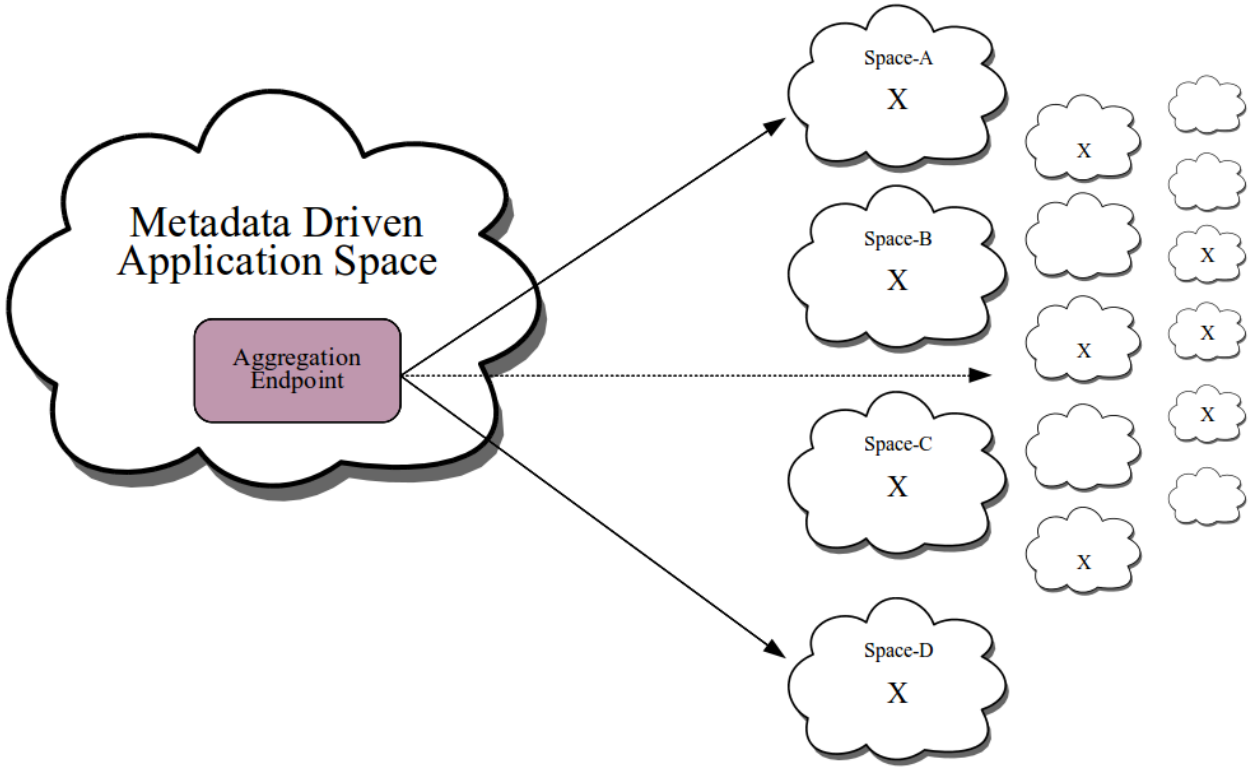

In
File
Out
IntrayTransport
ROC
Process

Metis System Architecture Execution Detail

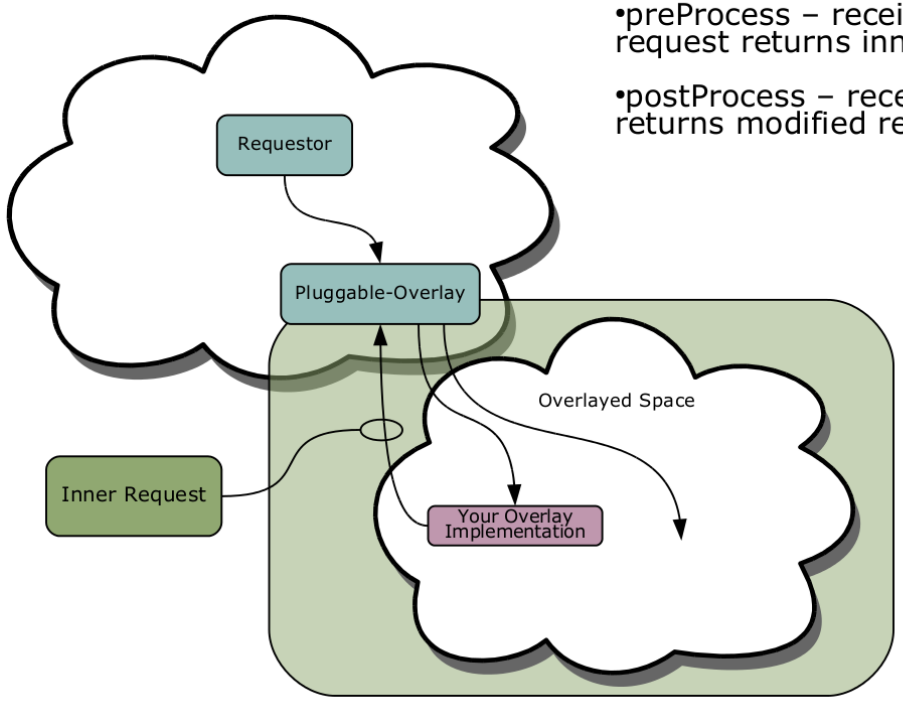


Metis System Architecture



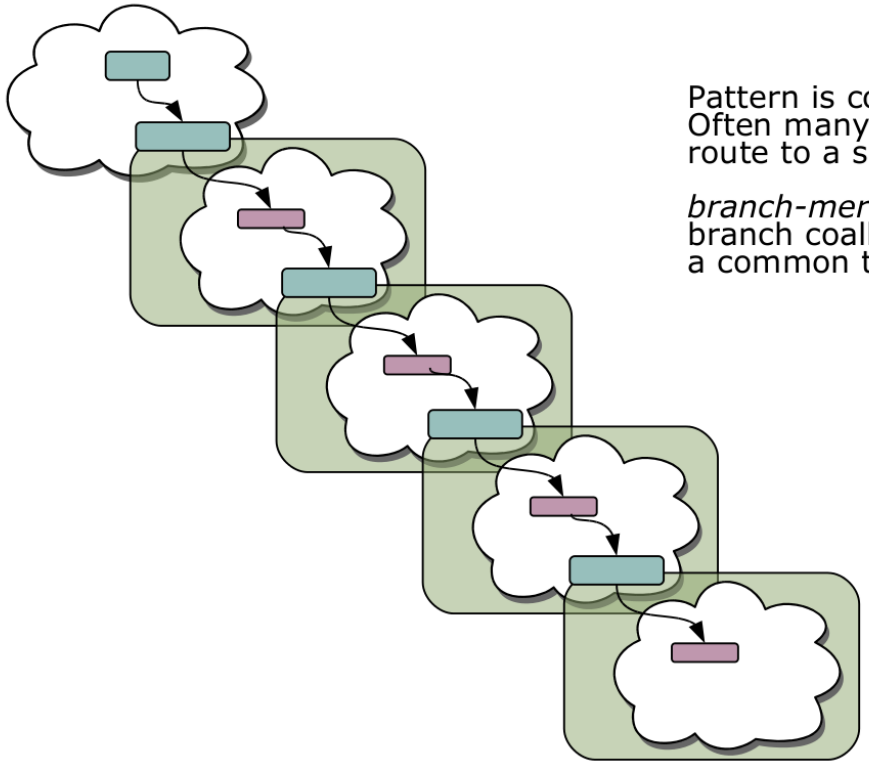

Metadata Driven
Application Space
Aggregation
Endpoint
Space-A
X
Space-B
X
Space-C
X
Space-D
X
X
X
X
X
X
X
X

Pluggable Overlay



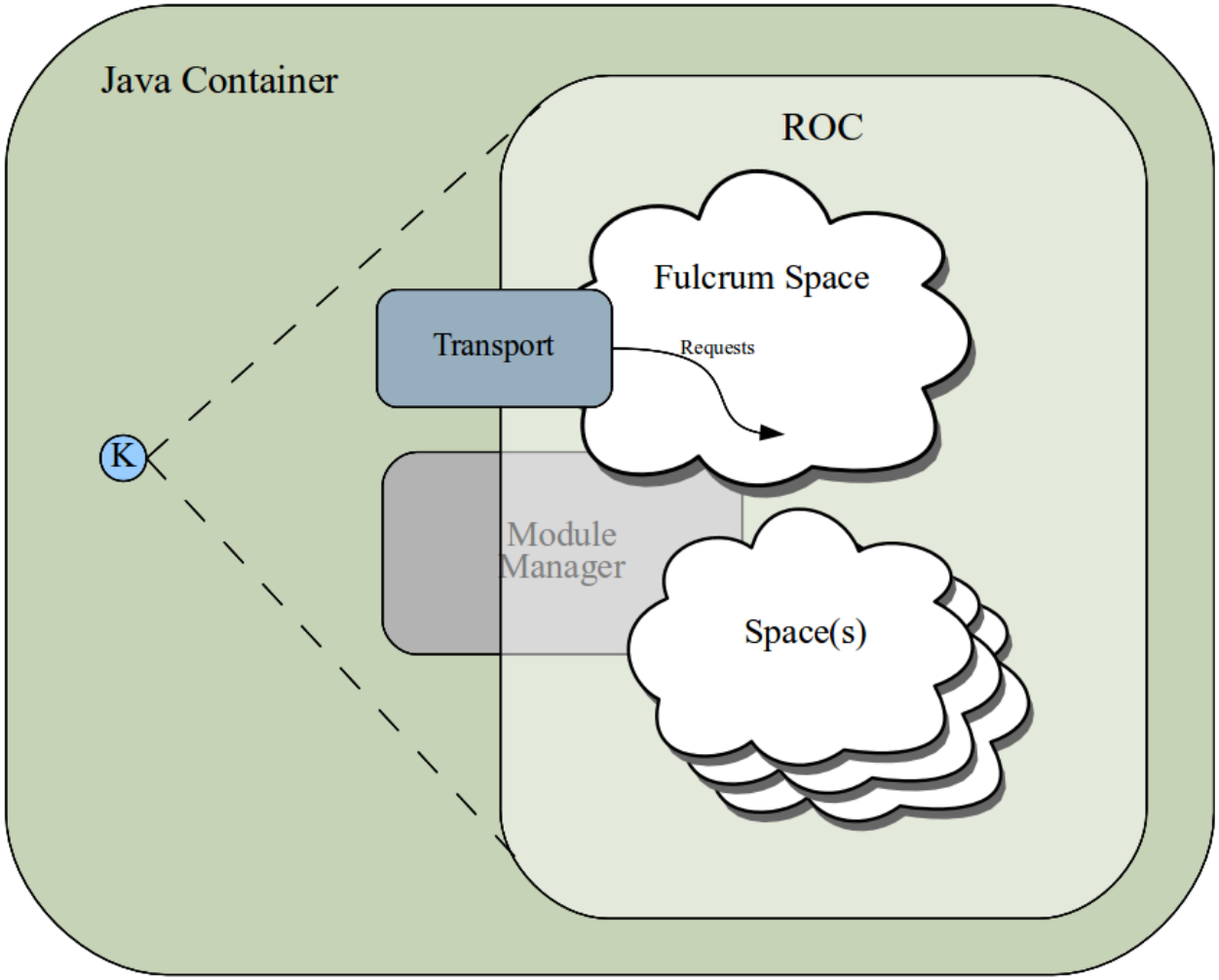
- preProcess – receives outer request returns inner request
- postProcess – receives response returns modified response

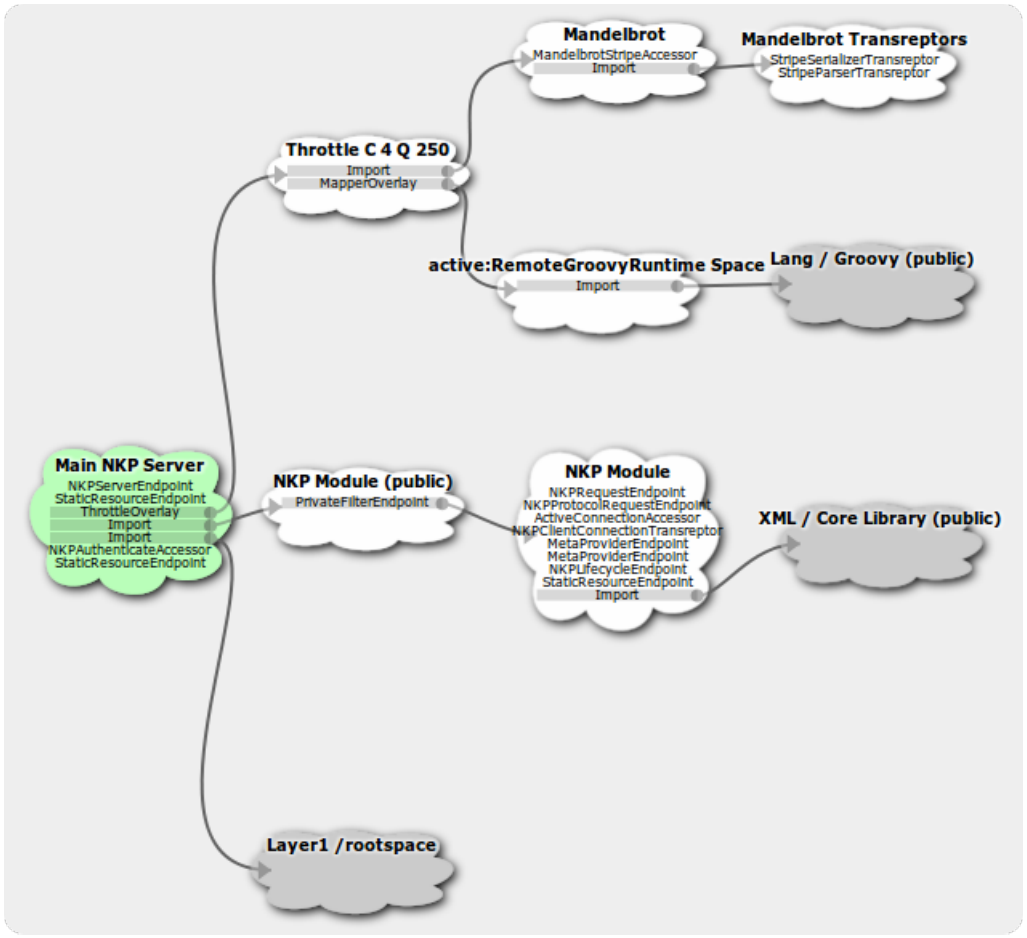
Overlay Nesting



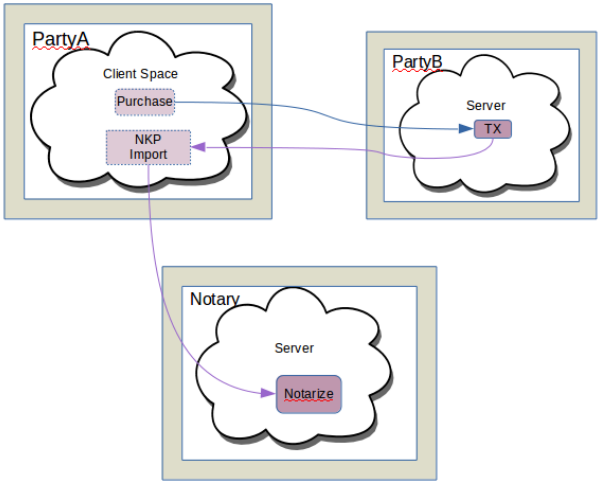
Pattern is common.
Often many inbound channels route to a single space.

branch-merge offers general branch coallescing to a common trunk space


Java Container
ROC
Fulcrum Space
Transport
Requests
K
Module
Manager
Space(s)


Mandelbrot
MandelbrotStripeAccessor
Import
Mandelbrot Transreptors
StripeSerializerTransreptor
StripeParserTransreptor
Throttle C 4 Q 250
Import
MapperOverlay
active:RemoteGroovyRuntime Space
Import
Lang / Groovy (public)
Main NKP Server
NKPServerEndpoint
StaticResourceEndpoint
ThrottleOverlay
Import
Import
NKPAuthenticateAccessor
StaticResourceEndpoint
NKP Module (public)
PrivateFilterEndpoint
NKP Module
NKPRequestEndpoint
NKPProtocolRequestEndpoint
ActiveConnectionAccessor
NKPClientConnectionTransreptor
MetaProviderEndpoint
MetaProviderEndpoint
NKPLifecycleEndpoint
StaticResourceEndpoint
Import
XML / Core Library (public)
Layer1 /rootspace

Measurable Economic Impact: ROC



demo

- 8 interactions → 2
- (27+c)^2 complexity → 27 complexity: >>27x simpler
- 6000:1 (t_old : t_new)
- Total time for round-trip 20ms (12ms PKI sign alone!)
- Surface area of attack is 1 single constrained point (minimized – cannot be smaller)
- Trust delegated from B to Notary within unique one-time “envelope of trust” (provable and measurable)
- Non-blocking logical architecture – tear down/bring up and it carries on

Overview

- Microservices are important because finer grain.
- We can build useful stuff more easily by composing pieces.
- There's nothing new here. The Unix model of specialized tools, combined into assemblies ("pipes and filters") is all about transfer of state to obtain a representation. We know that the composite is greater than the sum of the parts.
- What are we supposed to do? Have thousands, milllions of Docker containers to host each microservice?
- This doesn't work - but worse, the Web forces us into a flat monolithic address space. So all microservices are peers.
- This causes problems in security, but also in mangement and scaling and evolution.
- We need a way to partition the services into useful modular subsets.
- Here's how we do it...
- Multiple spaces, nano-services... Move away from HTTP since this makes us use the flat addressing of the internet.
- Scale invariance...
- Architecture that is decoupled and emergent...
- Introduces scope, as a concept outside language. Introduce context to our services.
- What are the practical tools that we need...
- Space explorer - we need to allow the metadata of the services to allow us to

Composite Resources (+ Dependency Caching)

Golden Thread Demo

Resources

- Composite Resource
- Resource 1
- Resource 2
- Resource 3
- Resource 4

Golden Threads

- Cut Golden Thread 1
- Cut Golden Thread 2
- Cut Golden Thread 3
- Cut Golden Thread 4

- Cut Golden Thread Odd
- Cut Golden Thread Even

- Cut Golden Thread All

Fullscreen