Resource Oriented Computing (R)evolution in REST

Peter Rodgers

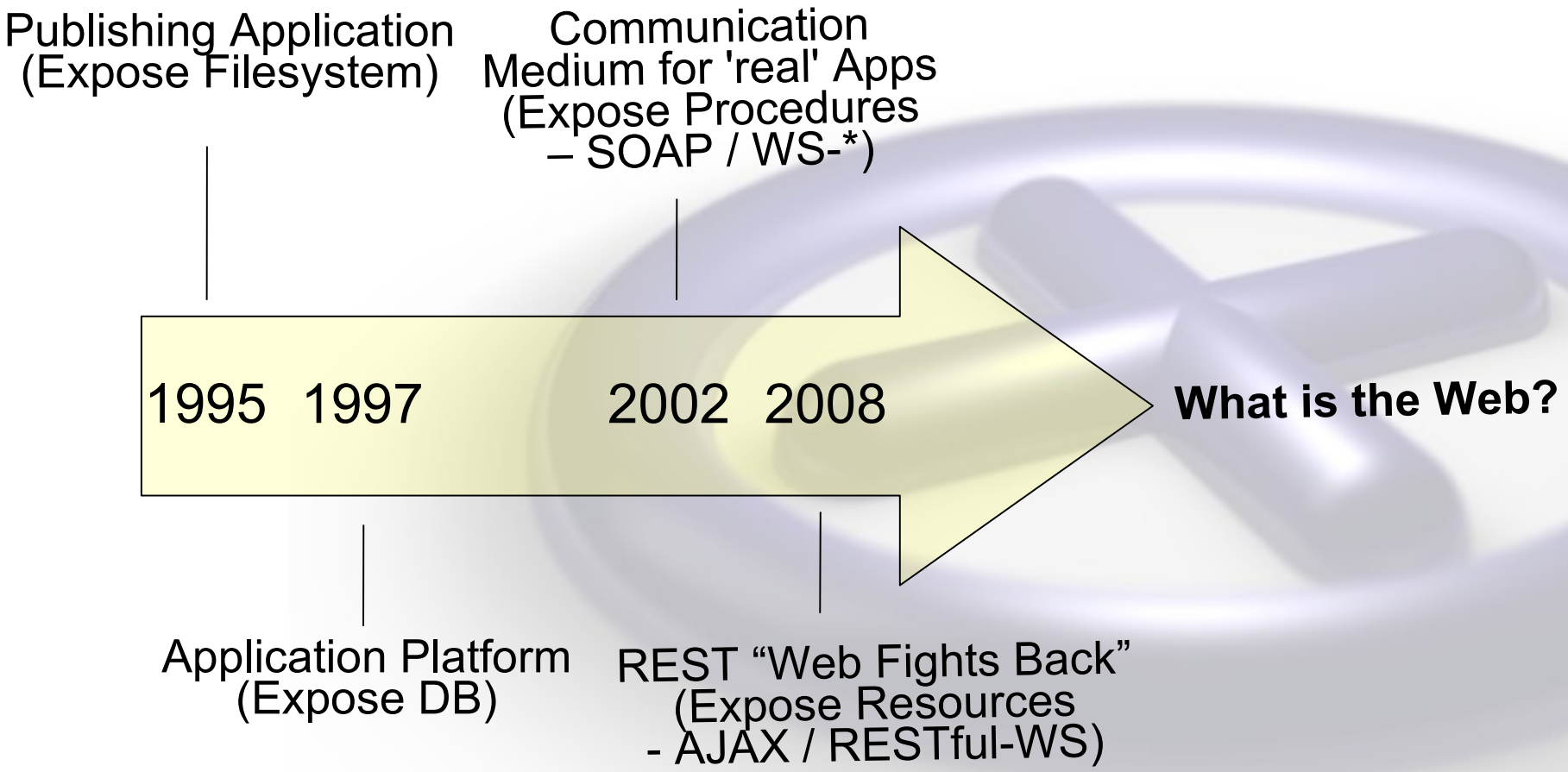
Introduction

- Physicist, Quantum Mechanics
- Hewlett-Packard Labs
- 1999 - Middleman project for Internet payment / billing / e-contract systems

- Software brittle.
- Web malleable.

- Research Goal: What is the web? How can we bring its economic properties to software?

The Web – A brief history



REST

- **RE**presentation **S**tate **T**ransfer
  - Client-Server
  - Stateless Application Protocol
  - Client-side State Management / Stateless Server
  - Uniform Interface
  - Caching
  - Layering
  - Code on demand
  - Resources are abstract
  - Representations physical *copy* of Resource.

Why does the Web/REST work?

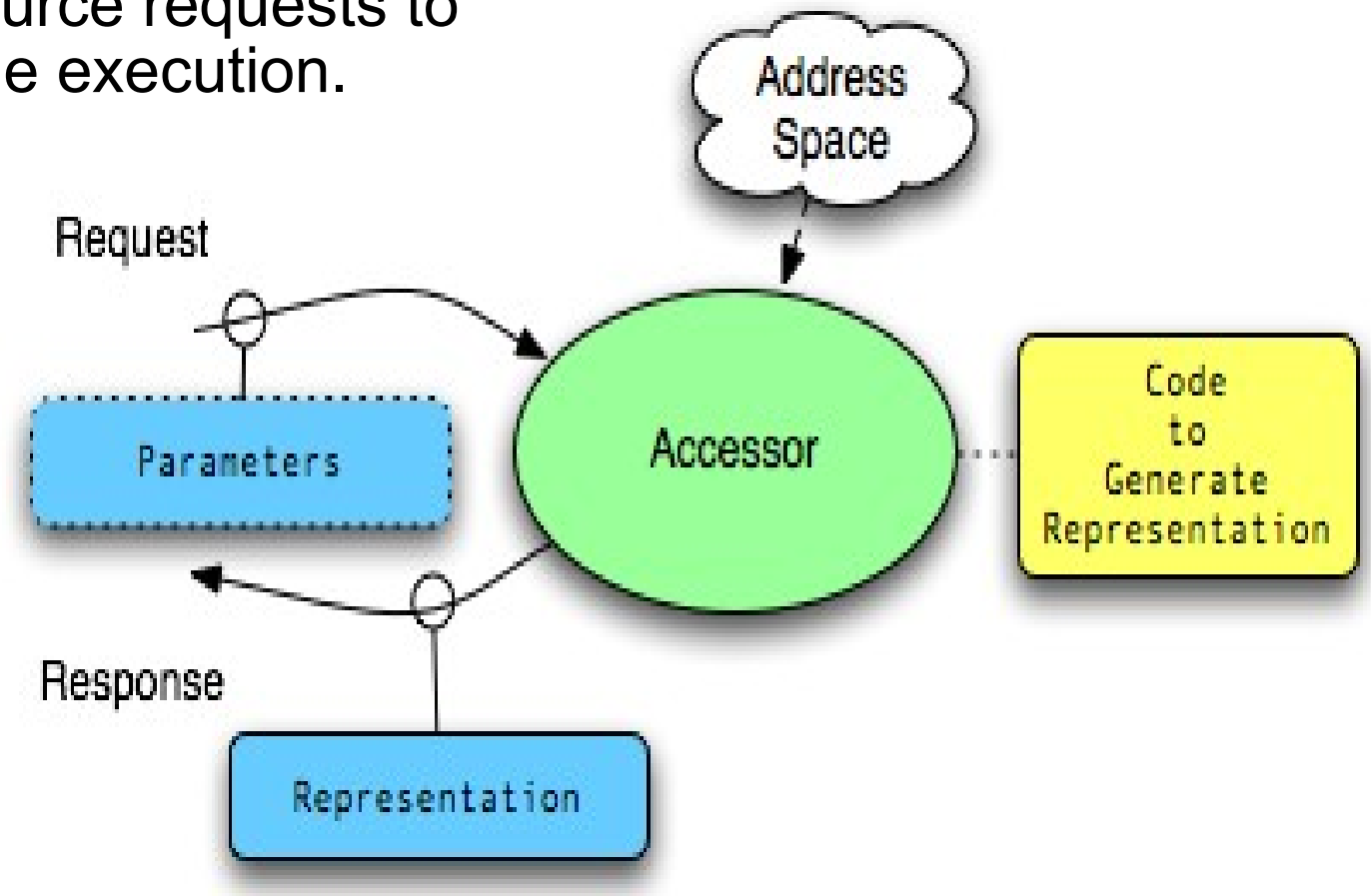
- Cost of Change << Value Added
  - Low cost of entry
  - Legacy co-exists
  - Linear scaling
  - Client/Server Independence
  - Instant deployment

  - Web is Resource Oriented: *Logical Requests for Information are isolated from Physical Implementation and Typeless*
  - Web is a Uniform Address Space

What is an Address Space?

What if Software were Resource Oriented?

Implement a dynamic address space to resolve resource requests to physical code execution.



Demonstration

- Use URI to express the computation for a resource.
- *Active URI*

  active:{base}+{name}@{uri}...

  {base} is function URI
  {name} is argument name
  {uri} is argument URI

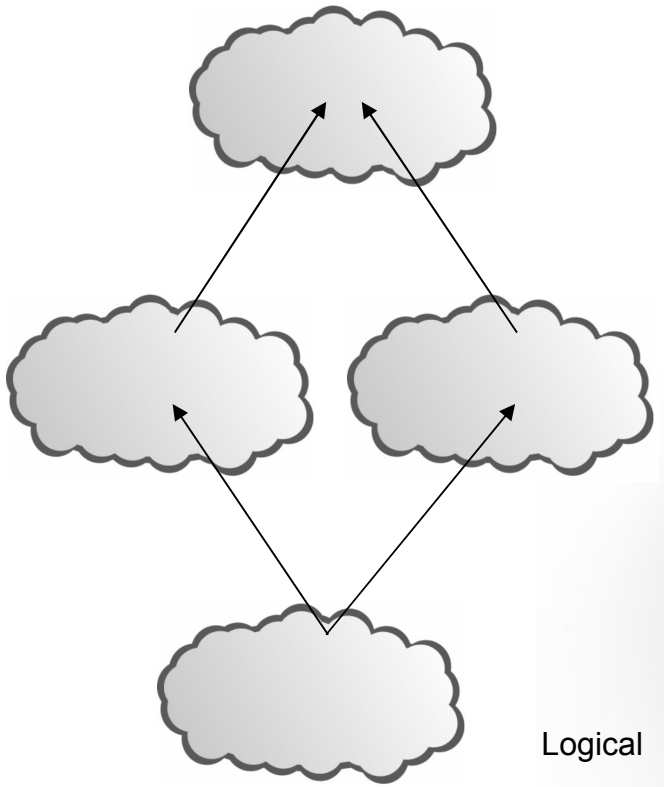
- Microkernel resolves logical URI resource requests to physical code execution.
- OS-like scheduler manages requests / threads etc...

Resource Oriented Computing

- Resource is abstract set of information
- Resource may have one or more Resource Identifiers
- Computation is the resolution of a Resource Identifier in an Address Space to a concrete Representation.
- Representation is *a* concrete form of the Resource.
- Representations are immutable.

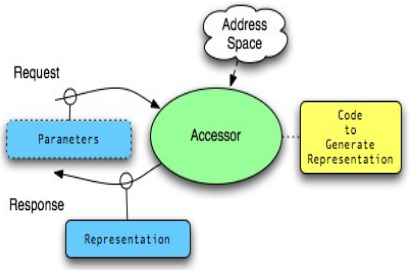
http://www.1060research.com/netkernel/whitepapers/

Dynamic Address Space



Logical

Physical



- Resource is a point in an address space
- Address space may import other address spaces
- Address space resolution is dynamic
- Dynamic modularity
- Computation: W*alk* the URI space to realize a resource representation.
- URI is program and unambiguously identifies resource => Caching

Uniform Resource Requests

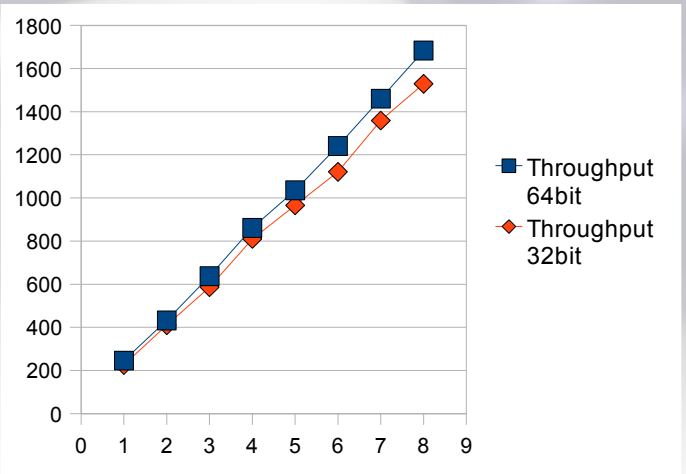
- Uniform model for information processing
  - What resource is being requested?
  - Create and issue further resource requests.
  - Add value.
  - Return resource representation.
- Client-Server symmetry

Scaling

- Logical URI requests are independent of threads.
- Scheduler assigns threads to requests beneath the logical level. Everything runs asynchronous.

  (1) Each function is a stateless service.
  (2) Each computed representation is immutable.

  (1) + (2) means that system scales-out across CPU cores in the same way that Web load balancing scales-out across servers.
- On multi-core we get 100% utilization and linear scaling.
  - Thread safety is guaranteed "safety interlock"
  - Threads never block
  - Async development challenge is removed from developer



Caching

- In ROC Resource, Representations have
  - Identifiers (URI)
  - Measurable value: computation cost.
  - Dependency hierarchy.
    - Invalidation of dependent resources is propagated
- Every representation can be cached.
  - ie. the result of every computation is cached.
- “Survival of the fittest” cache management.
- System self-tunes to the instantaneous most computationally efficient set of resources.
- NetKernel trys the cache before lazy evaluation of functional URIs => known resources are not recomputed.
- System-wide computational energy is a local minimum.

Demonstration

- Extrinsic Recursion Demo...

NetKernel – ROC Platform

- 1060 NetKernel v3.3
  - Resource-oriented application server
  - Symmetric client/server
  - Large collection of resource models, service libraries and dynamic languages.
  - Linear scaling with CPU cores + micro-caching
  - Modular hot-deployment.
  - Mature telecoms-class infrastructure (5-years market quality assurance).
  - Dual licensed
  - Requires: Java Standard Edition 1.4+

Resource Oriented Computing Summary: Value of the Web, *Inside*

- Cost of Change << Value Added
  - Low cost of entry
  - Legacy co-exists
  - Linear scaling
  - Client/Server Independence
  - Instant deployment

  - Resource Oriented: *Logical Requests for Information are isolated from Physical Implementation and Typeless*
  - **Software becomes a dynamically composable Address Space**

Reference

- For whitepapers about ROC:

  www.1060research.com

- For NetKernel downloads, community:

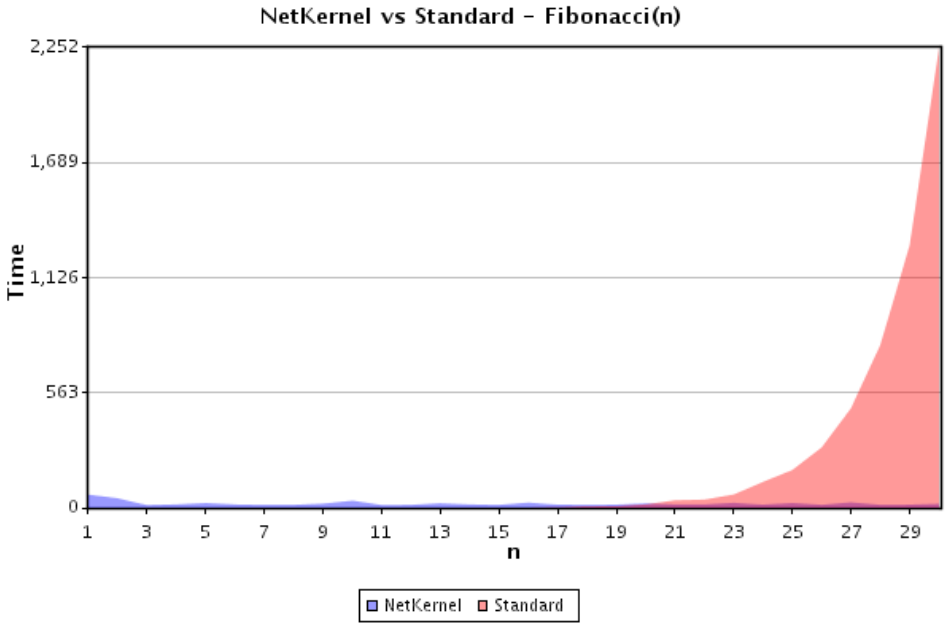
  www.1060.org

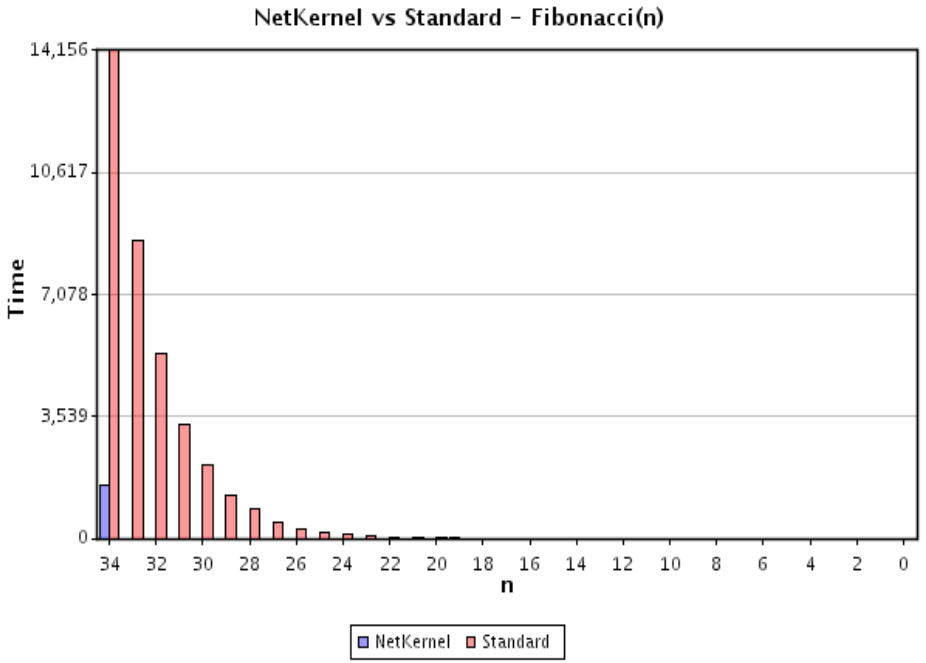
- Contact: peter.rodgers@1060research.com

- Certified training available from www.skillsmatter.com

Extra Material

Fibonacci Double Recursion





Transrepresentation

- Resource-oriented solution is focused on information, not types.
- Example, XML: File, Binary Stream, DOM, SAX, Stax, JDOM all representations of the same XML infoset information.
- Requestor can express representation preference.
- Software Function can express representation preference.
- Kernel can intermediate.
- Transreption: Isomorphic transformation of information from one representational form to another.

Information Thermodynamics

- Transreption is a dynamic generalization of: Parsing, Compilation, Configuration state, etc.
- Transreption is an abstraction by which Information Entropy can be minimized.
- Caching means energy outlay is one-time cost.
- Systemically: information is always maintained in its most computationally efficient form.
- To an application, parsing/compilation is transparent.
- Requestor-Function relationship is very malleable.
- As a generalization, whole new patterns are possible.

Principles of Unix

- Everything is a file resource
- File system is logically abstracted as a tree
- Memory is virtual
- Small, task focused tools
- Composition using scripting
- Kernel coordinates everything
- Processes have environmental context

Principles of SOA

- Asynchronous Communications
- Loosely Coupled
- Orchestration via Composition Languages
- Service Reusability
- Autonomy – service life-cycle is managed
- Discovery
- Contract

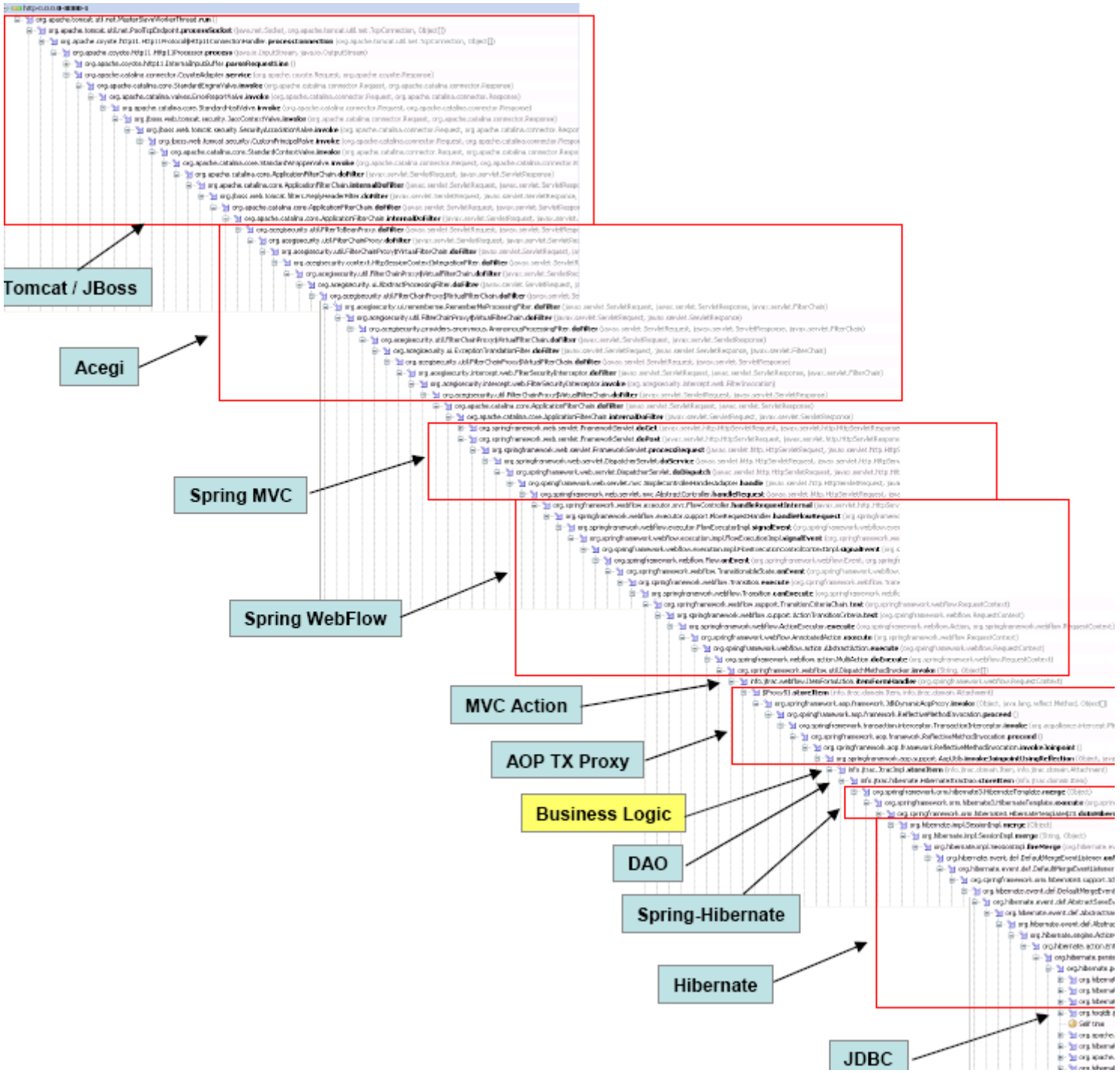
Malleability

- Modularity
  - Software linking is dynamic and re-evaluated for each request.
  - Software relationships can be dynamically reconfigured.
  - Hot Deployment, Version Management, Rollback
- Typeless Interfaces (Dynamic Type Matching)
- Logical-Logical Mappings
- Address-Space Relations – whole new world of design patterns (beyond OO).

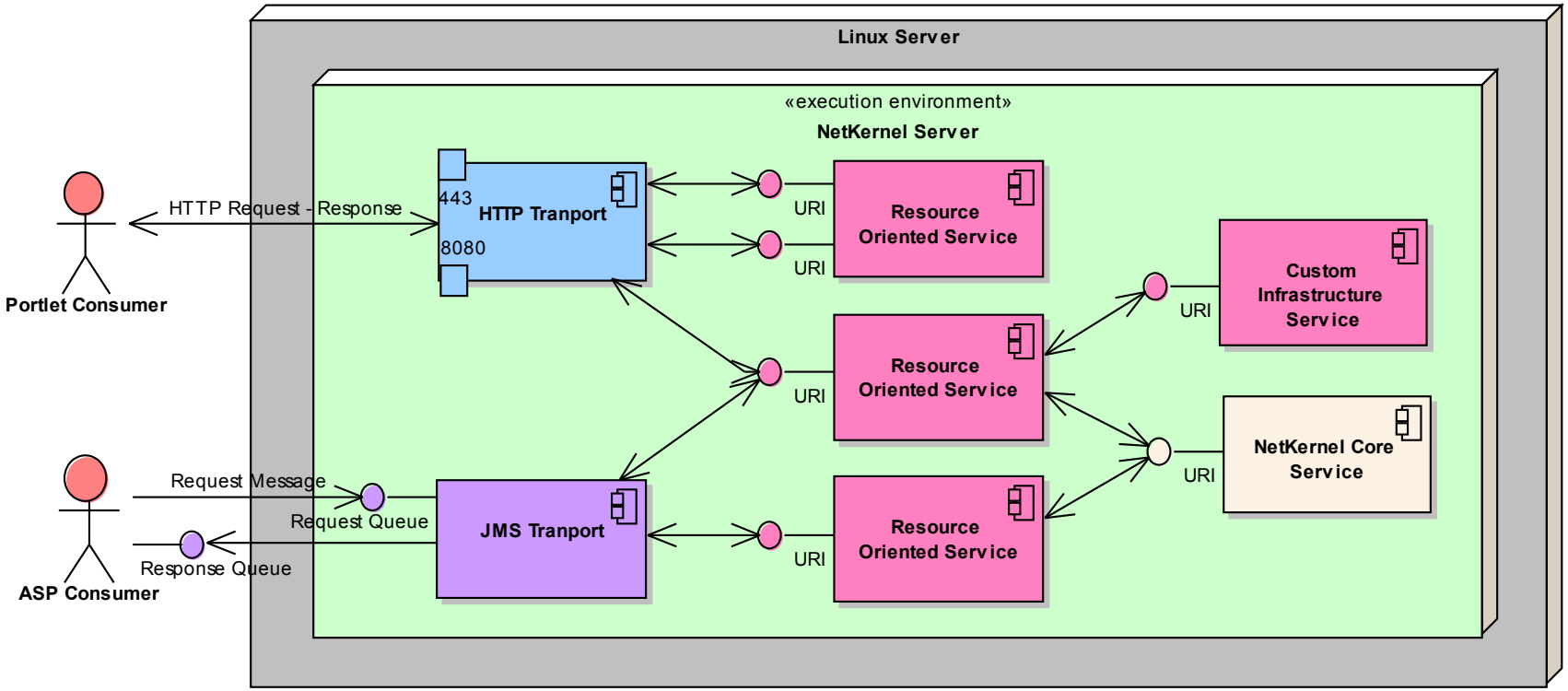
3Cs - Construct, Compose, Constrain

- Constrain
  - Transparently layer over constraints
  - Structural, Semantic, Access, Policy
- Compose
  - Develop address spaces and relationships.
  - Compose solutions by scripting (*cf* Unix)
- Construct
  - develop resource object models, accessors, transports
  - Construct only needed if off-the-peg option not available.

J2EE CallStack



Show me something real...



- RESTful ESB implemented using NetKernel – *Jeremy Deane, Collaborative Consulting for very large US University. Case Study on infoq.com . http://www.infoq.com/articles/netkernel-casestudy*
  -
  -

Real World Cases

- 1060 Research post-start-up, profitable
  - Customers: Telecoms, Insurance, US Govt (Intelligence), Financial Services, many OEM ISVs,
  - Purl.org powered by NetKernel
- Initial adopters were smart architects. Displacing J2EE.
  - Integration-server, Application-server, ESB, Multi-transport Internet peer.
- ROC Empirical Evidence (Vendor Hype!):
  - Typical solution is at least 3-4x faster than J2EE, Linear scaling with cores
  - Code size is 10-100x smaller, development time is 10x faster
  - Prototype is Application
  - Build more complex systems by composition of layered resources.