

# Search for the Holy Grail: Are SOAs the Answer?

*Rick Schlichting*

*Executive Director  
Software Systems Research  
AT&T Labs - Research*



# Acknowledgments

Matti Hiltunen

Kaustubh Joshi

Andrew Forrest

Rich Erickson

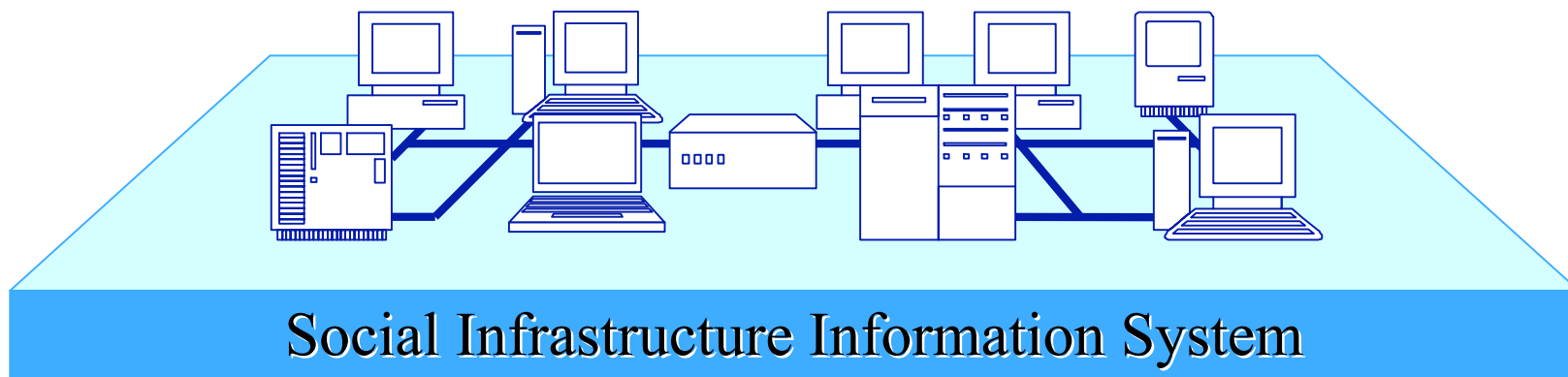
# The Quest

How can we build  
dependable applications  
and systems in a  
distributed environment?

Two themes:

- What's the reality?
- How can we structure our thinking to address the complexity?

# Why should we care?



# Characteristics

- Large number of networked machines.
- Spectrum of network types and technologies: wired, optical, wireless, ....
- Spectrum of distances: personal-area, local-area, metro-area, wide-area,....
- Spectrum of devices: from sensors to mobile units to high end machines and clusters.
- Spectrum of applications.
- Dynamic execution conditions and resource demands.
- Multiple administrative domains.

Significant technical challenges across a number of areas and at a number of levels.

# So why Dependability in Distributed Systems a Quest?

Long history shrouded in the mists of time.

- Since computers were first connected, e.g., Grapevine at Xerox PARC and its extensive use of replication.

Ancient search with a mythical (mystical?) goal.

- *Sed quis custodiet ipsos Custodes?*

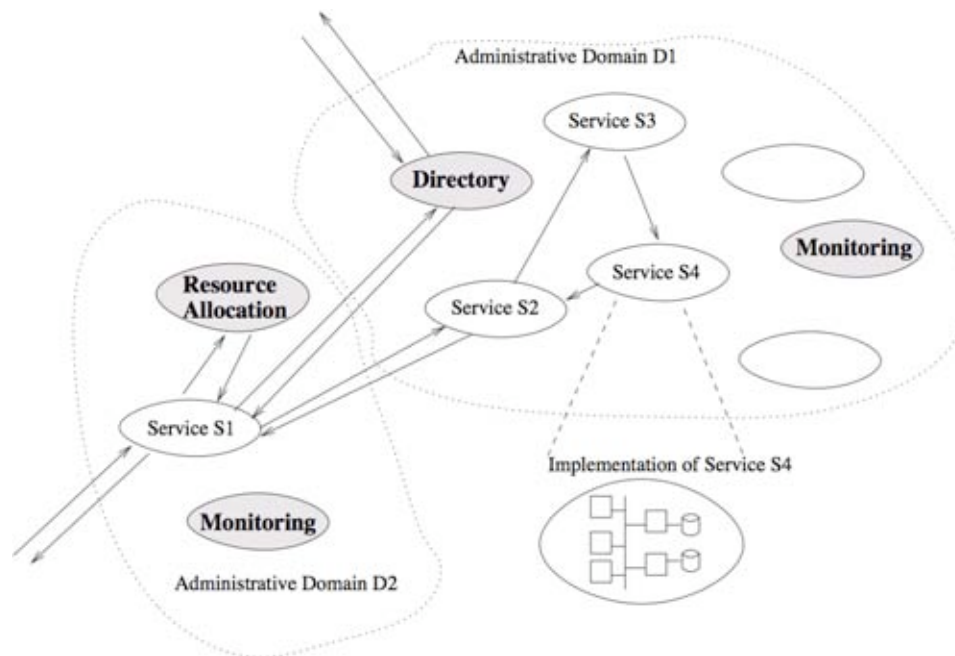
# Are SOAs Sir Galahad?

Hero of Arthurian legend. He was the son of Launcelot and Elaine, the daughter of King Pellès. Because he was the noblest and purest of the knights of Christendom, he alone, according to Sir Thomas Malory, achieved the Holy Grail.

*The Columbia Encyclopedia, 6th ed., 2001-07*

# So what is a SOA?

"Service Oriented Architectures (SOAs) structure software functionality in a distributed system as collections of interacting services. The services include both infrastructure services, such as directory services, monitoring, and resource allocation services, as well as application services that implement some application specific functions."





# Fundamental characteristics?

- Dynamic
- Distributed
- Time aware
- Composable
- Heterogeneous
- Multiple administrative domains
- Trustworthiness
- Published interface
- Evolving over time
- Migration
- Extensible

The background is a solid blue color with several curved, overlapping bands of varying shades of blue, creating a sense of motion and depth. The bands curve from the top left towards the bottom right.

**Now on to Reality**

# Reality: SOAs

Many definitions.

# AT&T Definition of SOA

*SOA is an overloaded term which requires definition and alignment.*




“**SOA**” is an approach to architecture & solution design which:

- Decomposes a domain or application into a set of abstract, highly reusable target functional interfaces (called target ‘services’).
- Brings governance to the design and selection of services as projects flow thru the development cycle, encouraging both reuse and build-out of target.

To support SOA:

- **A foundation of middleware, taxonomy and naming standards must be put in place, along with repository / management tools and governance.**
- **Target architects & lead engineers must functionally decompose their applications & enterprise domains into a set of highly reusable target services, which solution designers can reference in their designs and build out over time.**

**A common misconception is to equate SOA with Web Services or integration technologies like ESBs.**



**Service Oriented Architecture (SOA)** is a software architecture where functionality is grouped around business processes and packaged as interoperable *services*. SOA also describes IT infrastructure which allows different applications to exchange data with one another as they participate in business processes. The aim is a *loose coupling* of services with operating systems, programming languages and other technologies which underly applications. [Wikipedia]

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed. [service-architecture.com]

**SOA** Service Oriented Architecture. Depending upon the context, the meaning changes:- a technical architecture supported by standard protocols and data formats- an approach to designing information systems that exposes enterprise assets as configurable actors within a dynamic business process- a new business paradigm that synchronizes the goals of the board room (execs) and server room (IT) [soamatters.com]

**SOA** (Service-Oriented Architecture): SOA is an architecture, the aim of which is to achieve a loose connection between integrated systems. From a common public Danish perspective, the integration of IT systems across public and private organisations is part of the vision of digital administration. [capevo.com]

**Service Orientated Architecture (SOA)** - a software design that integrates business functions. Users are able to decide the information which is to be shared between the functions. SOA is therefore more flexible and more loosely coupled than ERP and generally more suitable for service rather than manufacturing companies. [bpic.co.uk]

**Service Oriented Architecture (SoA).** A Service-oriented Architecture defines how two or more entities interact in such a way as to enable one entity to perform a unit of work on behalf of another entity. The unit of work is referred to as a service, and the service interactions are defined using a well-defined description language. Each interaction is self-contained and loosely coupled, so that each interaction remains independent of any other interaction. While SOAP-enabled Web Services are the most common implementation of SoA, Web Services are not necessarily required to define a SoA. The protocol independence of SoA means that different consumers can use services by communicating with the service in different ways. Service Orientation is a method of architecting systems of autonomous services. Services are built to last, with good availability and reliability, and systems are built to change, with new service topologies evolving over time. [Dunelm]

Service-Oriented Architecture, 面向服务架构, SOA是一种 架构模型, 它可以根据需求通过网络对松散耦合的粗粒度应用组件进行分布式部署、组合和使用。服务层是SOA的基础, 可以直接被应用调用, 从而有效控制系统中与软件代理交互的人为依赖性。SOA的几个关键特性: 一种粗粒度、松耦合服务架构, 服务之间通过简单、精确定义接口进行通讯, 不涉及底层编程接口和通讯模型。

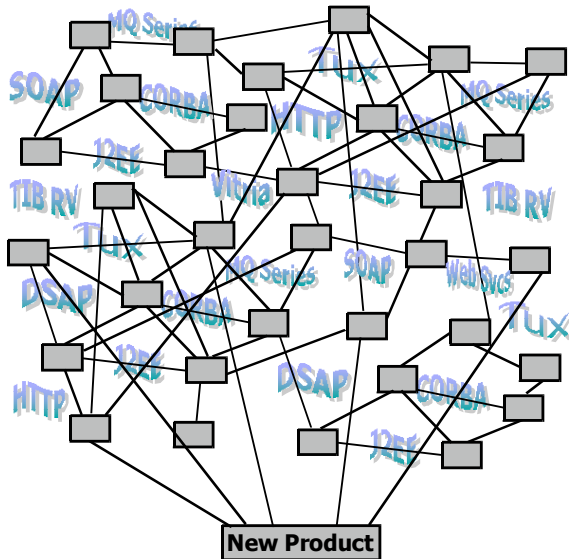


# Reality: +

## Real business value

- ➔ Interface management
- ➔ Reuse
- ➔ Standardized processes, etc.

# Complexity Reduction & Consolidation



## BEFORE – The Accidental Architecture

Over the years, many enterprises have developed ‘accidental architectures’ made up of the gradual accretion of systems and applications interconnected with diverse middleware.

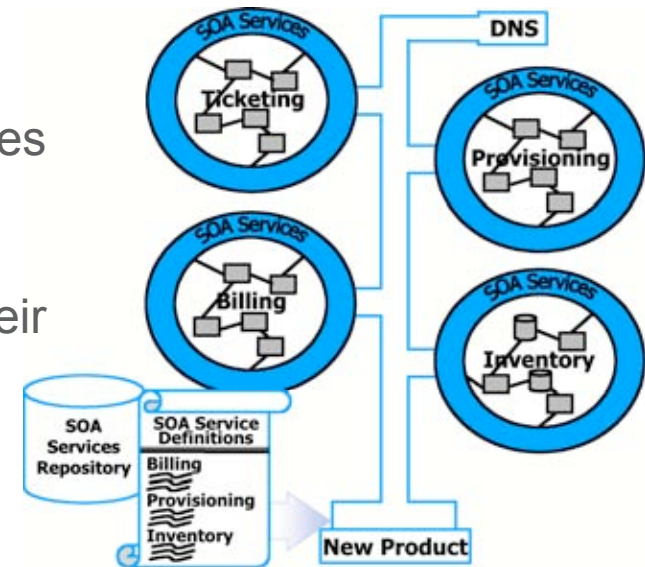
The ‘accidental architecture’ misses the primary aim of architecture, which is to break down a complicated problem into simple pieces and drive out complexity to make construction and maintenance easy.

## AFTER – Service Oriented Architecture

SOA partitions and encapsulates existing capabilities behind a well thought out set of target services.

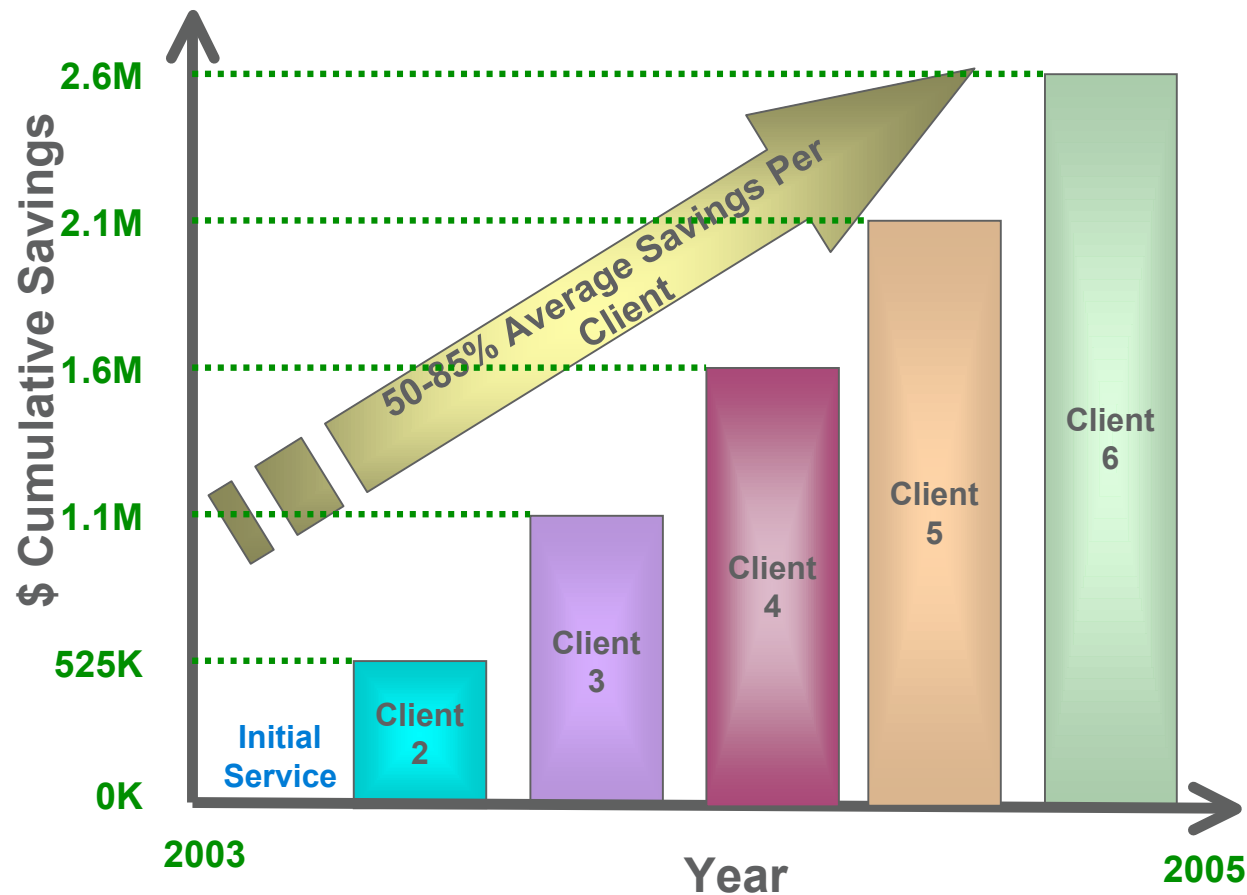
Solution teams reuse and extend this portfolio services, instead of redeveloping functionality to their specific preferences. Reuse of services cuts cost and speeds time to market.

Once encapsulated, internal infrastructure can be consolidated, enhanced and/or retired.



# SOA Case Study

By 2005 AT&T had documented over \$40 million in savings from SOA, as in this example of a system that accrued \$2.6 million in 2 years by reusing one service across 5 clients.

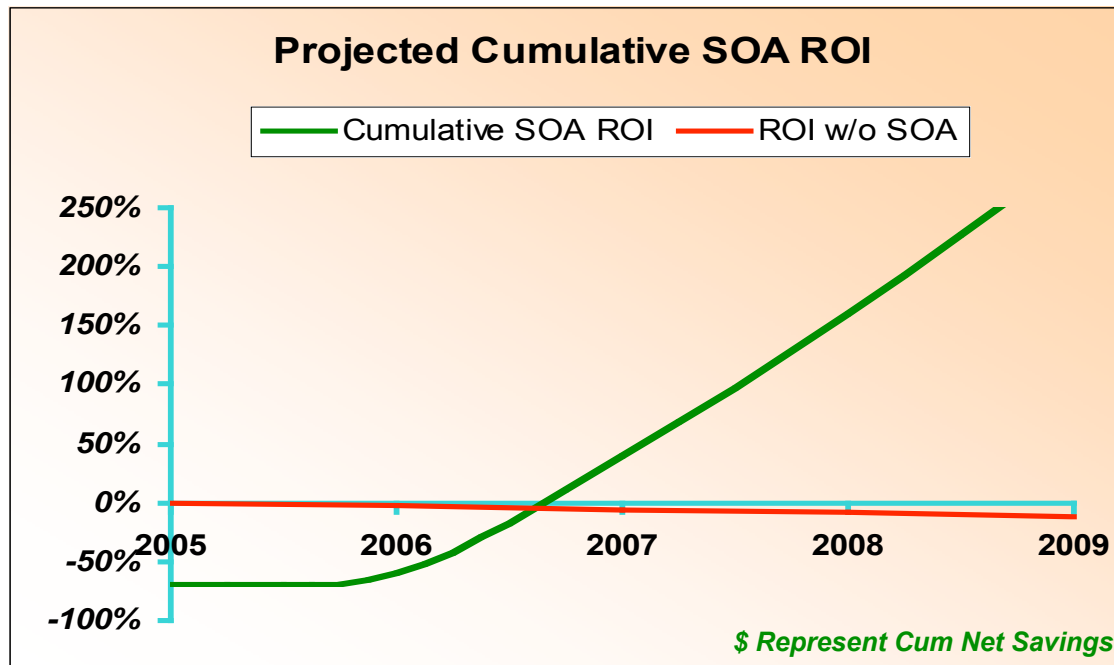


## Highlights:

- Reuse of a single service **saved 50%-85%** of the cost of building custom interfaces.
- **Savings will continue to accumulate** as more clients are added.
- **Maintenance costs will be lower** (not shown) because fewer interfaces need to be versioned and maintained.
- **Operational efficiencies will be higher** (not shown) because of increased consistency across SOA customer/client interfaces.

# SOA Value to AT&T

*The SOA benefit model was recast and zeroed out in 2005. It projects additional savings in excess of \$100M by 2009.*



## SOA Benefit Model:

- Service reuse contributes an average 50% reduction in integration cost.
- Includes engineering efficiencies from use of standards, models and repositories.
- Includes development efficiencies from use of standard integration toolkits
- Without SOA costs and complexity continue to increase.

### Key Assumptions:

- Constant annual development budget spend at 2005 levels.
- Rate of re-use of existing services is approximately 3 times per service during a 10 year period.
  - Note: The system on the previous slide provided 5 instances of reuse within 2 years
- SOA adoption rate grows from 25% of projects in 2006 to 90% of projects by 2009.
- Average overhead to create SOA services for the first time is 10% over the current costs.
- Cost of a new interface is \$(att proprietary) on average.

# SOA Business Value Summary

*With the correct execution strategy, SOA will deliver significant benefits across the enterprise.*

Driver	Description	Benefits
➔ Reduced development costs	Reuse & less reinvention of functionality across projects	20% reduction in development cost; 50% savings per reuse
➔ Reduced maintenance costs	Fewer interfaces, versions and middlewares to maintain	Ongoing cost savings beyond development
➔ Reduced complexity	Encapsulates complexity behind simple service interfaces	Teams see SOA services; not legacy systems and technology
➔ Reduced effort in design & testing	Complexity reduction leads to easier design and testing	Higher quality features; reduced fallout
➔ Accelerated time to market	Reuse and complexity reduction cuts time required to deliver new features	Greater responsiveness to competitive pressures
➔ Increased solution assembly	Solutions are delivered by orchestrating a library of existing services	Process centric solutions; more time for business logic
➔ Easier systems consolidation	Breaks the direct link between users and legacy assets	Business can dictate when & where to rationalize assets
➔ More integrated & agile processes	Process tasks leverage a growing library of SOA services	Reduced re-keying & input errors

# Reality: -

More static and predefined than pure model.

Complexity of applications/services!

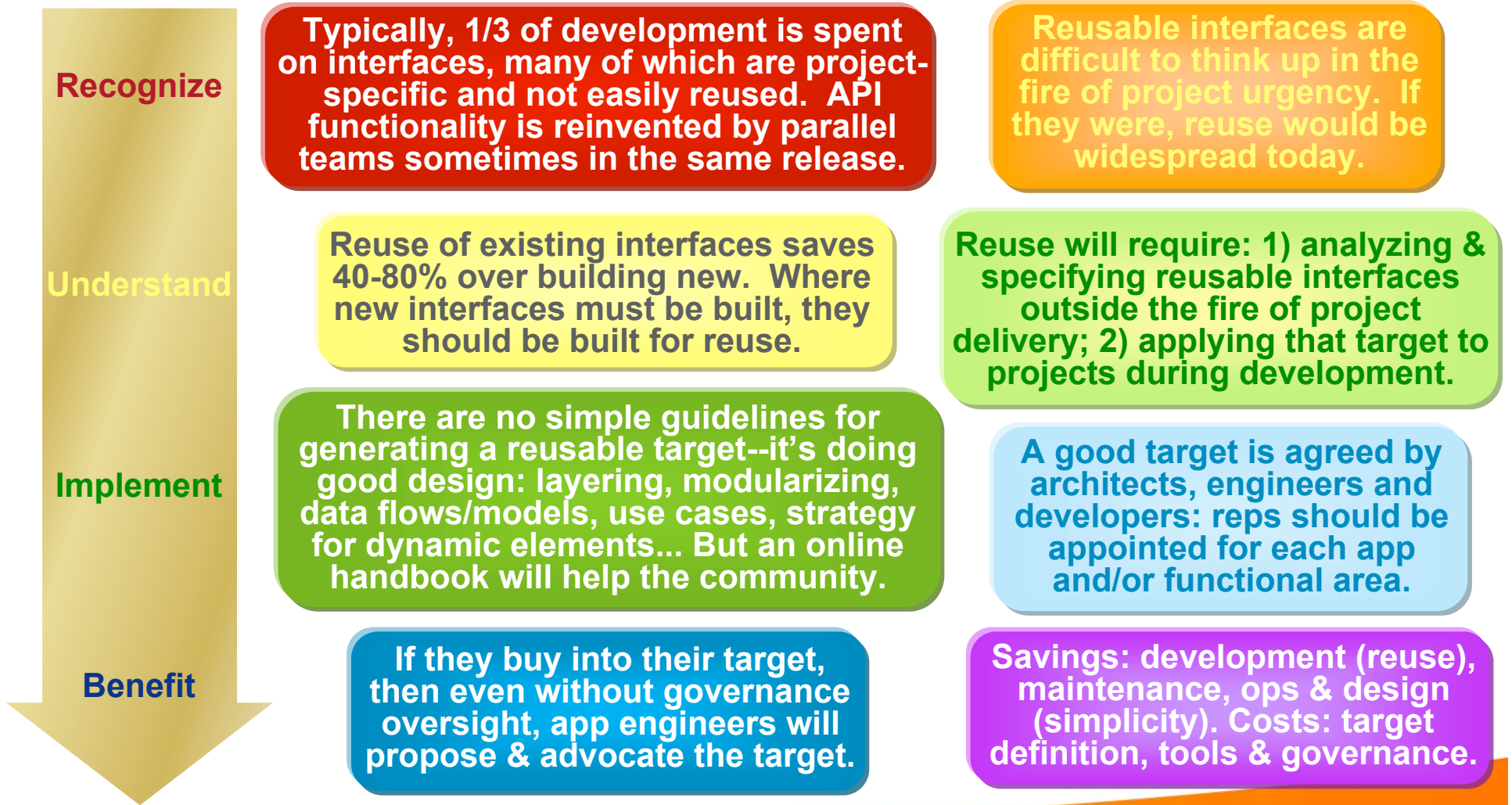
- Lots of technologies
- Scale
- Interface version management

Trust

And ...

# The SOA Reuse Challenge

*Reuse requires a repository of existing & target interfaces, plus governance or minimally buy-in from app owners.*





# Reality: Execution Infrastructure

No longer as simple as server + OS.

➔ Layers upon layers of software.

Dynamic shared environments.

➔ Cloud computing and utility computing.

➔ VMs





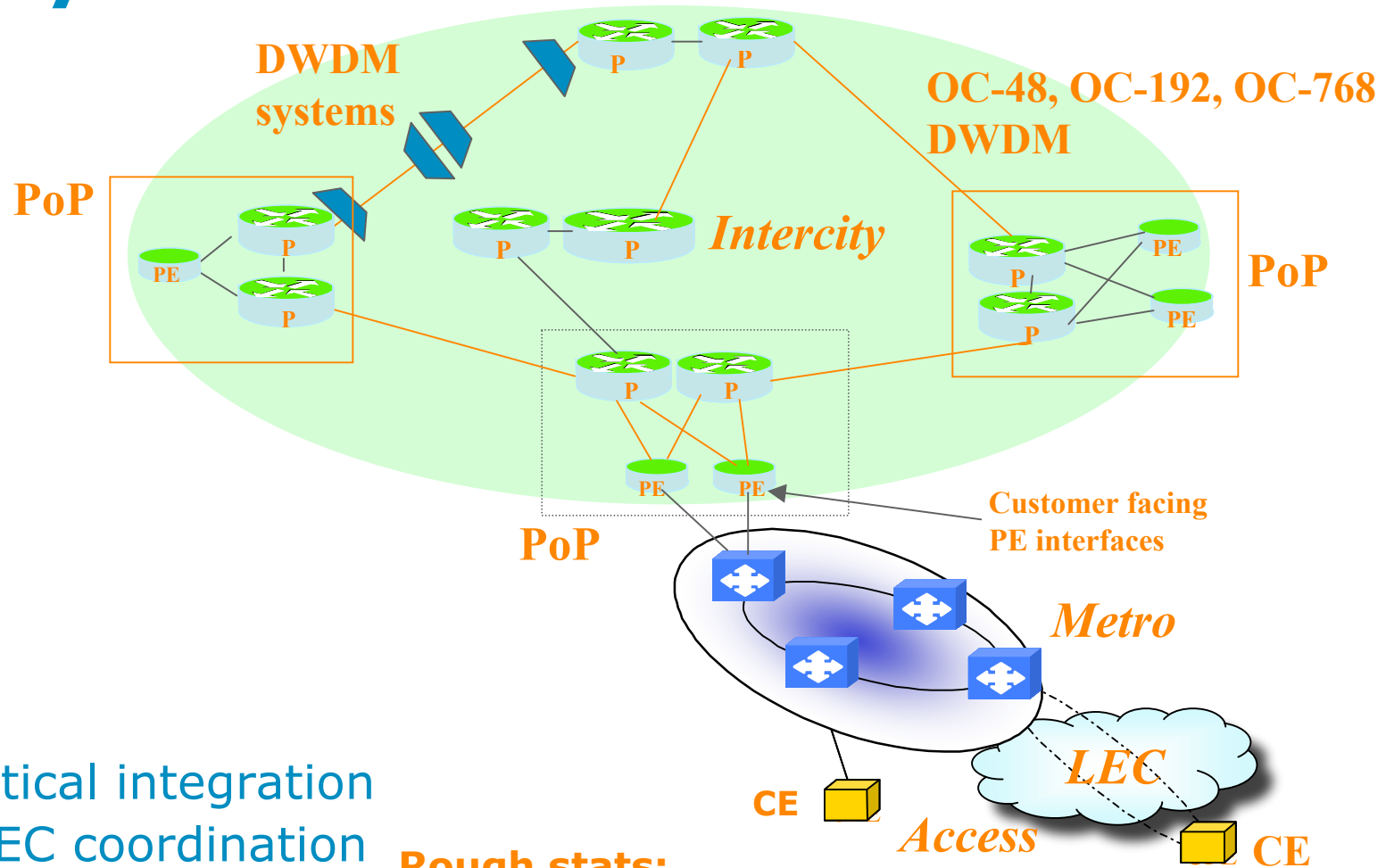
# Reality: Global Networks (AT&T)

Around 10 Petabytes of Traffic Average Business Day



MPLS-based Services in 127 countries at 1500+ service nodes  
112,000+ MPLS customer ports  
30 data centers on 4 continents  
Over 525,000 route miles

# Reality: IP Networks

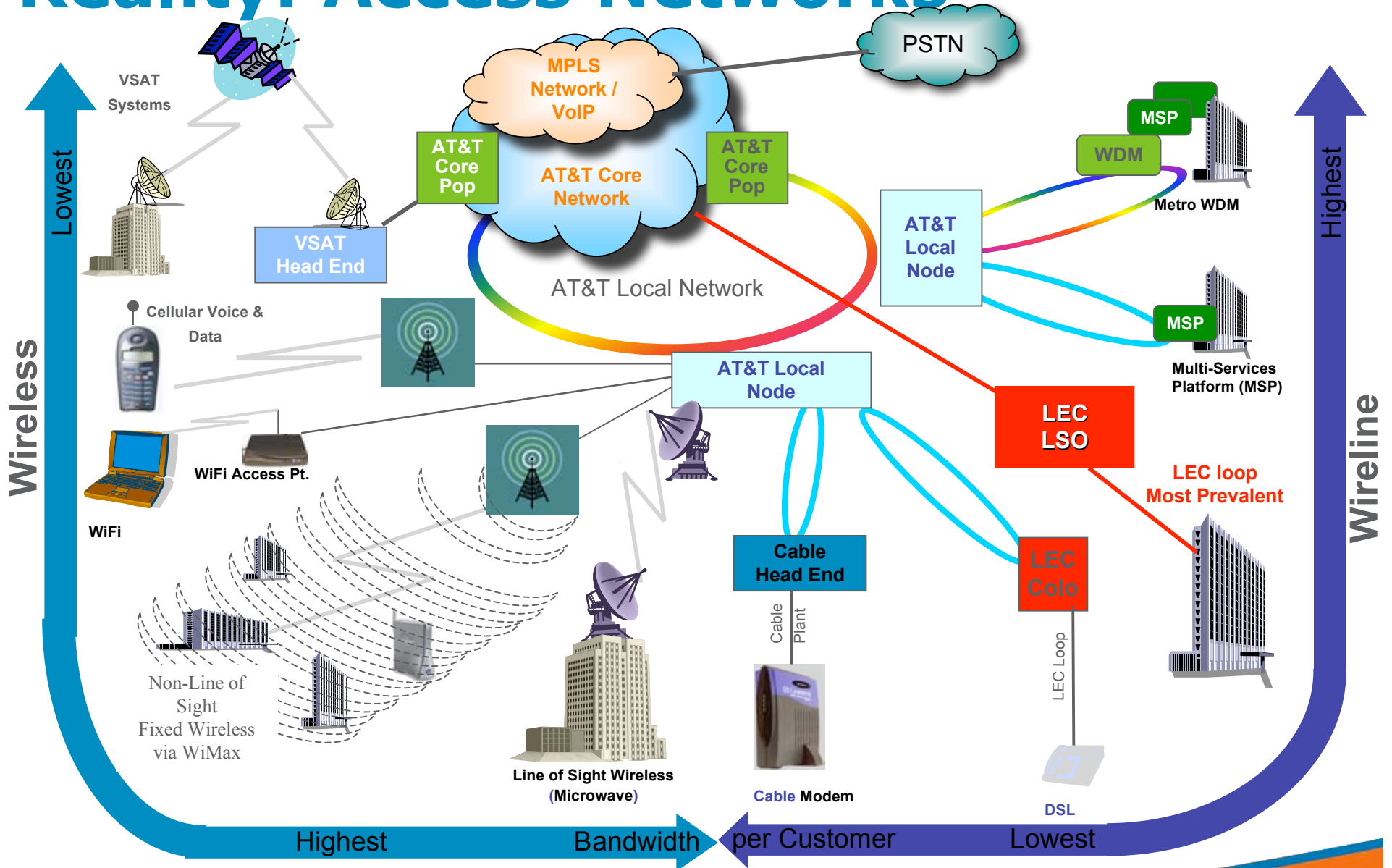


- IP/Optical integration
- ISP/LEC coordination

**PoP:** Point-of-Presence  
**P:** Backbone (core) Router  
**PE:** Provider Edge Router  
**CE:** Customer Edge Router

**Rough stats:**  
**100s of offices**  
**100s of Ps, 1000s of PEs, 10000s of CEs**  
**100,000s of transport facilities**

# Reality: Access Networks



The background is a solid blue color with several curved, overlapping bands of varying shades of blue, creating a sense of motion and depth. The text is positioned on the left side of the image.

# **Dealing with Dependability Challenges**

# Dependability

**Definition: The ability of a system to avoid service failures that are more frequent or more severe than is acceptable.** (*Avizienis, et al., Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Volume 1, Number 1, Jan-Mar 2004.*)

## Includes many properties and attributes

- Reliability
- Availability
- Safety
- Security
- Timeliness

## Non-functional or Quality of Service (QoS) attributes

- Focus is not on *what* gets done, but rather *how well*.

## **Immensely challenging to build SOAs with these attributes!**

- Failures, intrusions....
- Concurrent and non-deterministic execution
- Heterogeneous systems and networks
- Resource constraints
- Multiple administrative domains
- Scale
- Dynamic shared execution infrastructure
- Dynamic application

## **How are these different than generic distributed applications/systems?**

- Emphasis/degree: some clearly more relevant in SOAs

## **How to deal with this complexity?**

# Handle in the Application

Why?

- Traditional approach: treat infrastructure as a black box (or at least as a simple virtual platform)
- Much of the existing work has been in this space.

The value of system abstractions

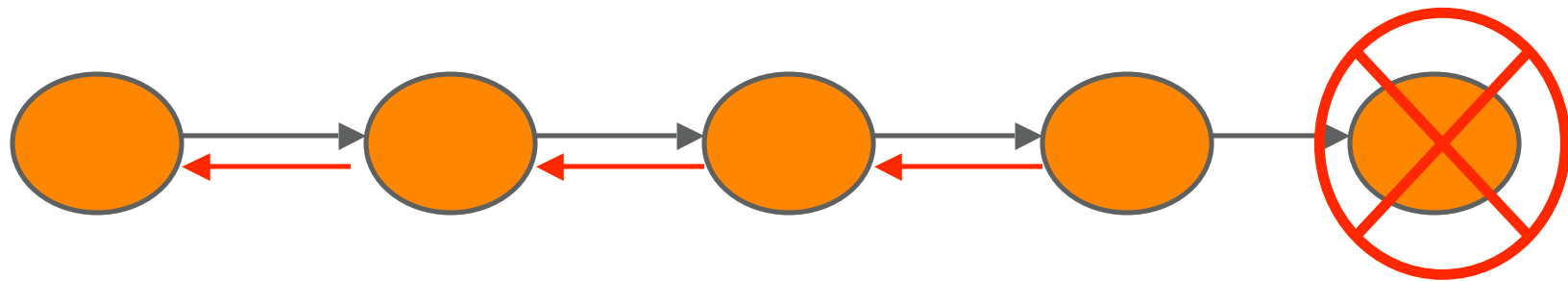
Customization and adaptability

Example: Customizable *durability* service state attribute (EDCC06).

# Customizable State Durability

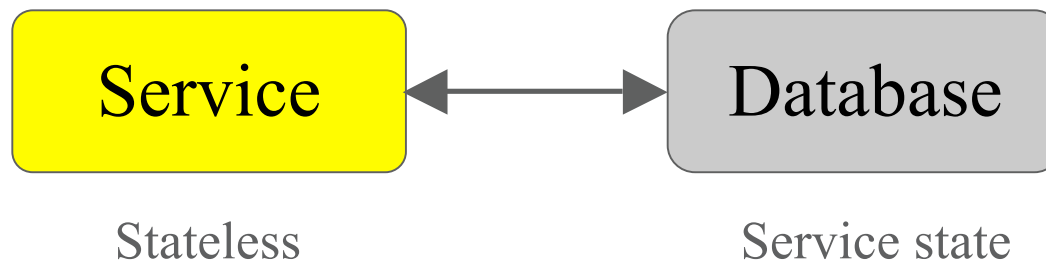
(with X. Zhang, M. Hiltunen, K. Marzullo)

In SOAs, unavailability of a service can result in the unavailability of a large number of other services.



Issue: maintaining service state across failures.

One common approach – use a back-end database.





Abstraction is *stable storage* – a database in *one* implementation with certain tradeoffs.

Goal: Provide an integrated and transparent way to use different techniques for protecting service state in the context of SOAs.

Solution: *State durability* as an explicit service state attribute  
↳ The likelihood that the state can survive failures.

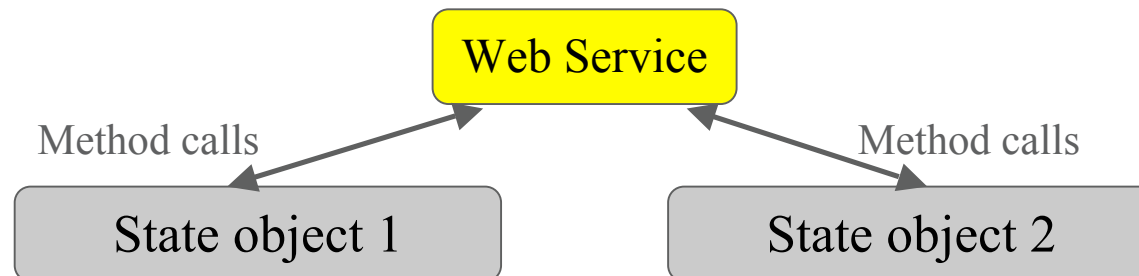
Use different techniques with different tradeoffs:

- Local disk
- Database
- State replication across a set of servers
- State reconstruction after a failure
- .....

# Transparent Customizable Durability

## Architecture:

- Service state is stored in one or more state objects (*resources*).
- Different resources of the same service may have different durability.
- Web services implemented using Java.



## Assumptions:

- Machine and web service container crashes
- Failure detection and service/resource restart handled at next higher level

Goal: Web service + state objects with customized tradeoffs based on desired durability.

## Challenges:

- Internal transparency: no manual modification of the web service code.
- External transparency: no change to external behavior of the web service.
- State update and restoration: handling different techniques for state update coherently.
- Atomicity wrt failures: ensuring all or nothing semantics in the event of failure (including ensuring correct semantics when part of a WS-Transaction).

# Approach – Key Concepts

## Durability proxies

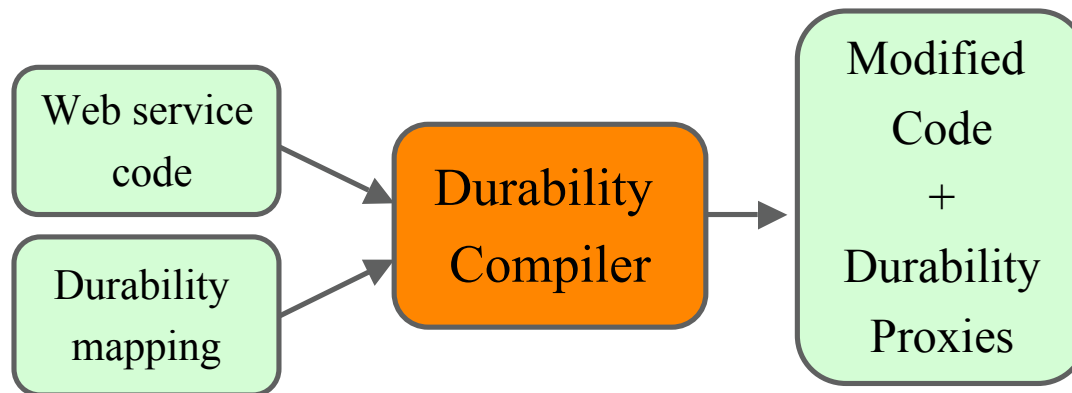
- Implement a durability mechanism in a generic, service independent, manner.

## Durability mapping

- Specify which durability mechanism to use and object-specific information.

## Durability compiler

- Generate proxies, modified web service, and resources based on mapping and original code.



# Ideas

Focus on abstraction (stable storage) rather than implementation

Durability as an explicit state attribute ➡ control over inherent tradeoffs

Many challenges ➡ focus here on transparent incorporation of different techniques.

# Handle in the Infrastructure

Idea: Provide as much dependability functionality in the infrastructure

Why?

- Simplify applications.
- Hierarchy of dependability abstractions.
- Good if you don't have access to applications.

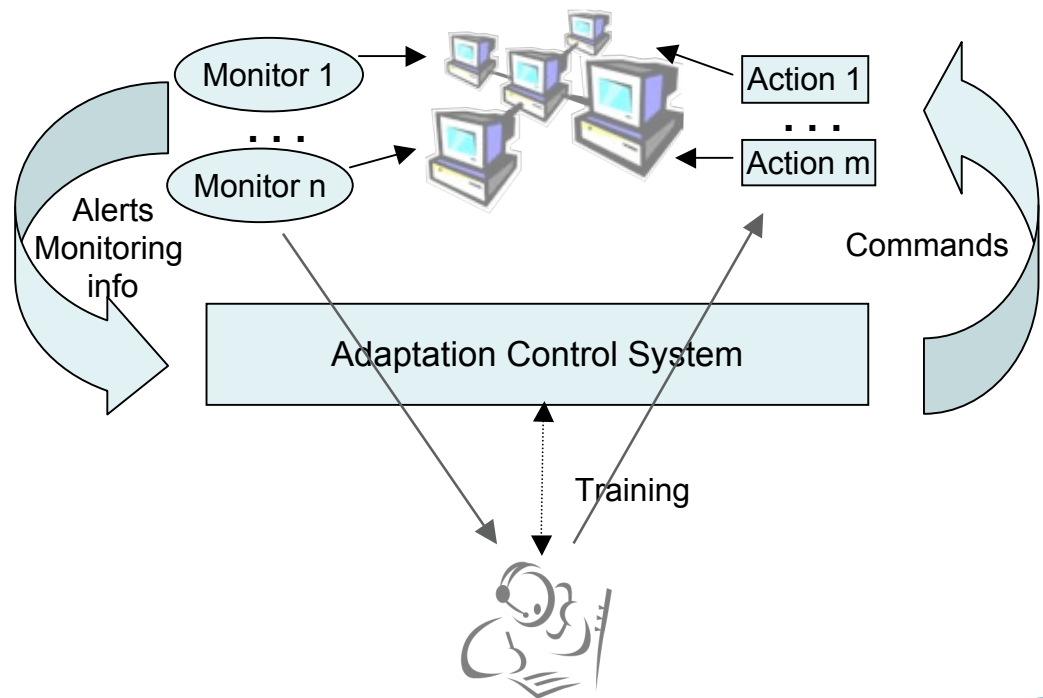
Example: Automatically generating adaptation policies (ICAC08).

# Adaptive Systems

*Dynamically changing system behavior.*

## Examples:

- Networking: Changing video frame rate in response to congestion.
- Mobile systems: Implementing location-specific services.
- Fault tolerance: Reconfiguring software to deal with a host failure.
- Survivability: To impose additional barriers to counteract an intruder.



# Challenges

Fundamental issue  $\Rightarrow$  each phase of the feedback control loop is complex in large networked systems.

Analyze and decide  $\Rightarrow$  policies

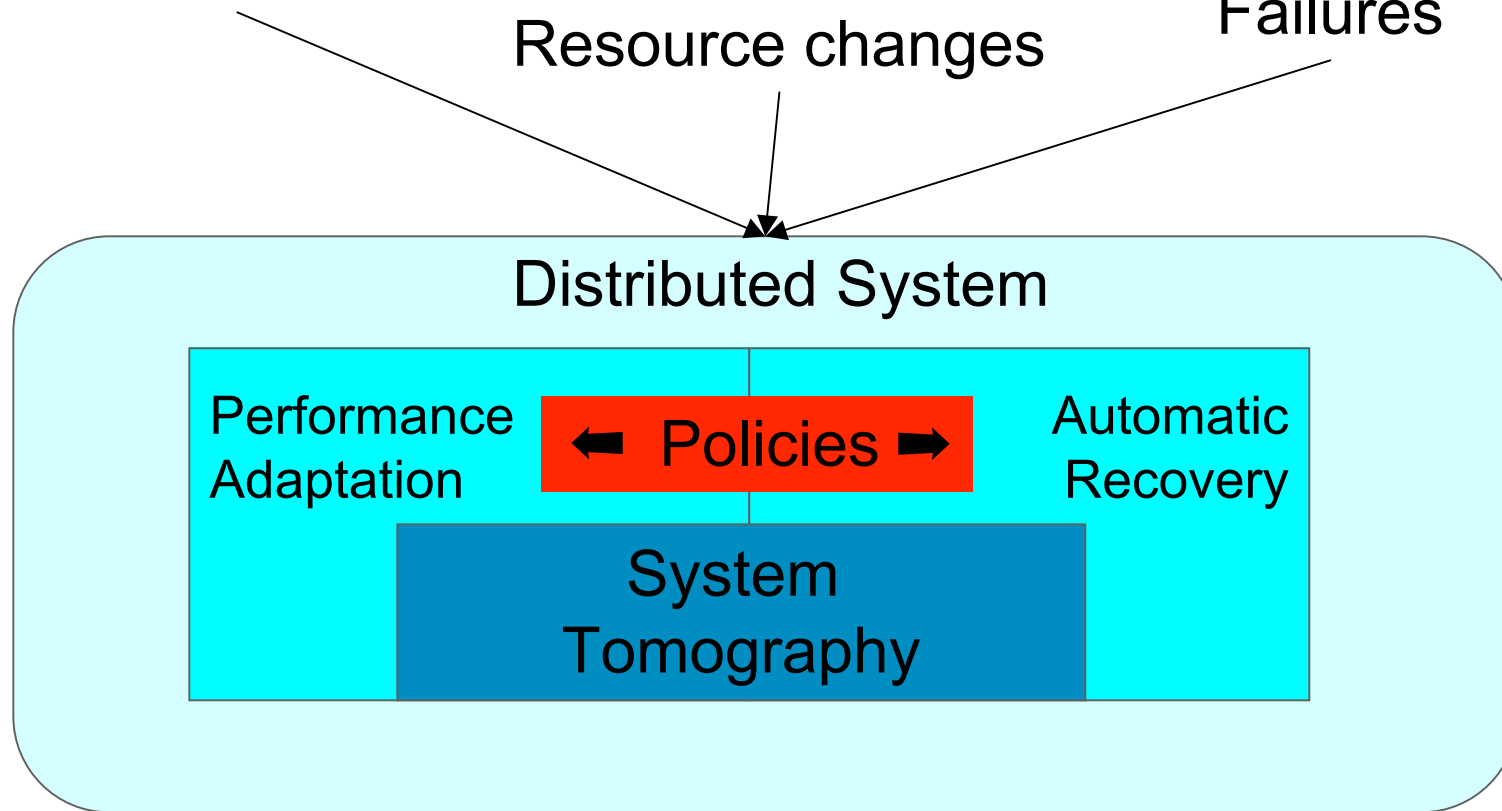
- Determining actual system state from monitoring results.
- Developing policies  $\Rightarrow$  *automatic generation*
  - Predicting impact of changes  $\Rightarrow$  *system tomography*



Workload changes

Resource changes

Failures



Cholla Adaptation Architecture

# Generating Adaptation Policies

(with G. Jung, , K. Joshi, M. Hiltunen, C. Pu)

Problem: Deciding how to continuously configure systems to adapt to changing conditions.

Technologies: Stochastic models, reinforcement learning, control theory

Typical approach:

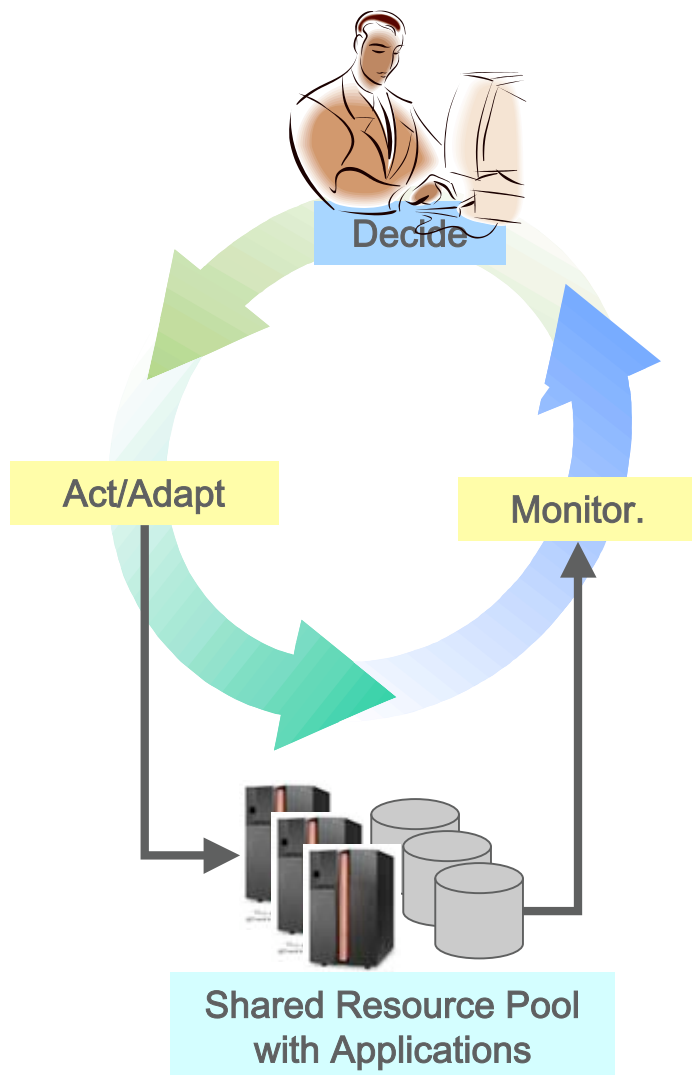
- Construct a parametric model of the target system
- Fix some parameters through experiments or learning
- Devise strategy for optimizing rest of parameters using runtime state as input
- Implement strategy as an *online controller*
- Use output of controller to configure system

Disadvantages: lack transparency and predictability, performance can be an issue, etc.

➔ Have developed a *hybrid* approach:

- *Offline* optimization and model solution to generate optimal configurations.
- Use generated rule sets at runtime (*policies*).
- Context: Server consolidation, with focus on *multi-tier enterprise* applications

# Runtime Resource Management



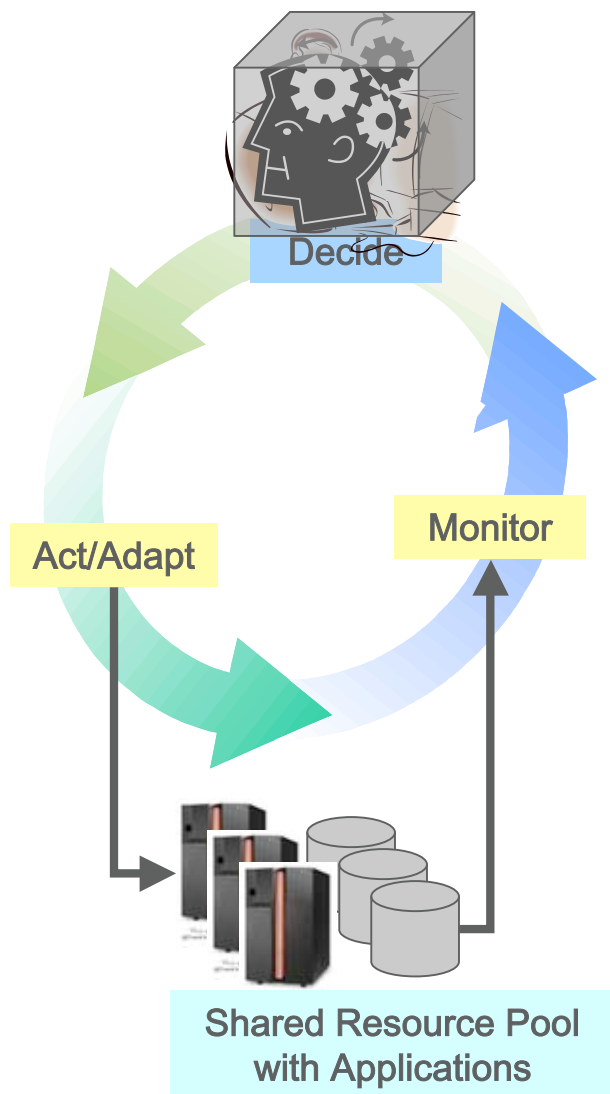
## Monitor:

- resource utilization,
- response times,
- failure alarms.

## Actions:

- Start/stop processes (e.g., adjust replication degree of a component).
- Migrate processes
- Adjust CPU allocation (e.g., virtual machine technology).

# Current approach



Significant and obvious limitations of manual approach

- Slow reaction time (10s of minutes).
- Difficult to consider all factors in a complex system.
- Human errors.
- Cost of 24/7 operations.

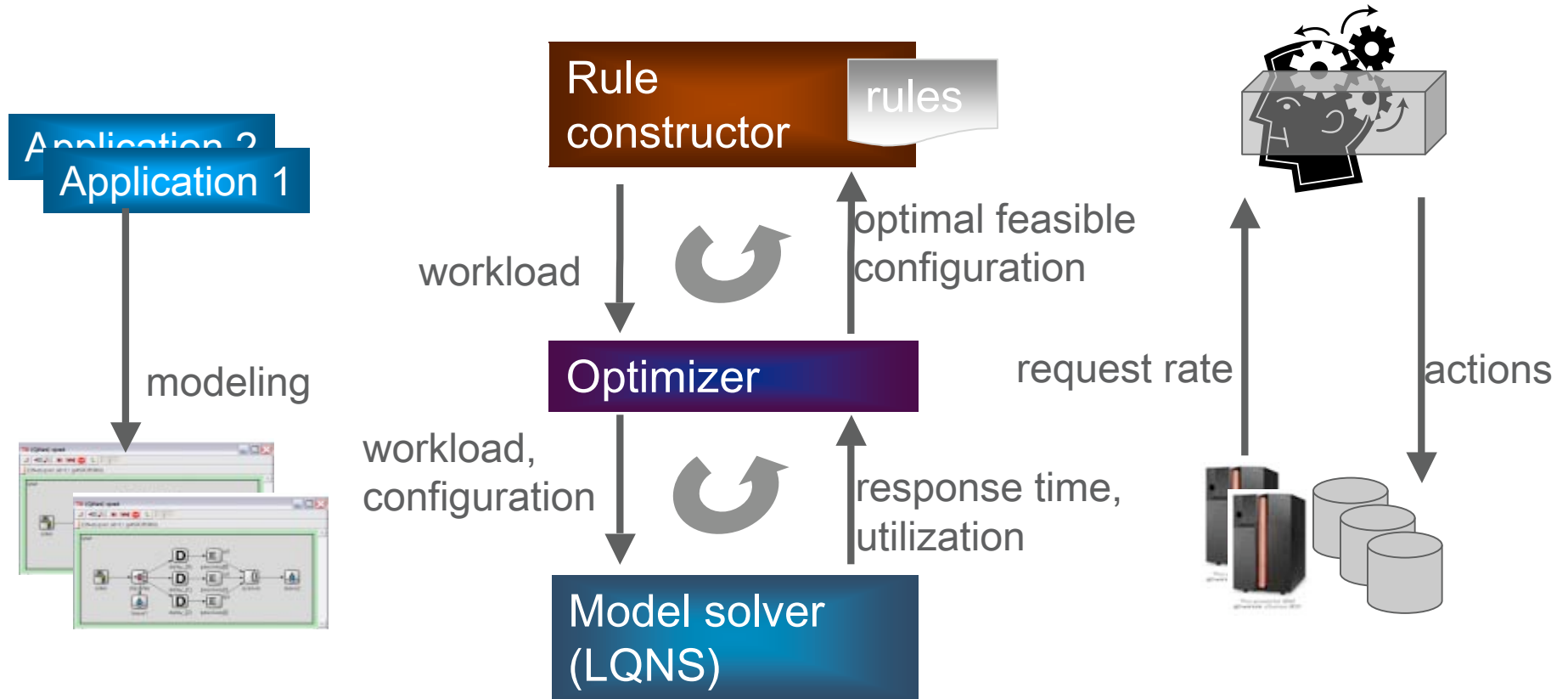
Solution: Replace operator with a rule-based management system.

Challenge: Developing rules.

➔ Use stochastic models as the basic technology.

➔ Inline vs offline

# Our Approach



# Specifics

Given:

- A set of computing resources  $R$
- A set of applications  $A$ :
  - each consists of a set of components/tiers
  - each component has a set of possible replication degrees
  - may support multiple transaction types.
- For each transaction type,
  - a transaction graph describes how the transaction uses application components
  - each component's service time
- For each application, an SLA that, for each transaction type, specifies the desired (mean) response time and the reward/penalty for meeting/missing this time

## Measured at runtime:

- Each application's workload for each transaction type

## Goal:

- Configure the set of applications  $A$  on the resources  $R$  so that the reward with current workload is maximized
- Configuration:
  1. Degree of replication for each component (of each app)
  2. Virtual machine parameters for a VM running the component (CPU fraction)
  3. Placement of VMs on the physical machines  $R$

# Optimization

For a given workload, find the configuration with the maximum utility.

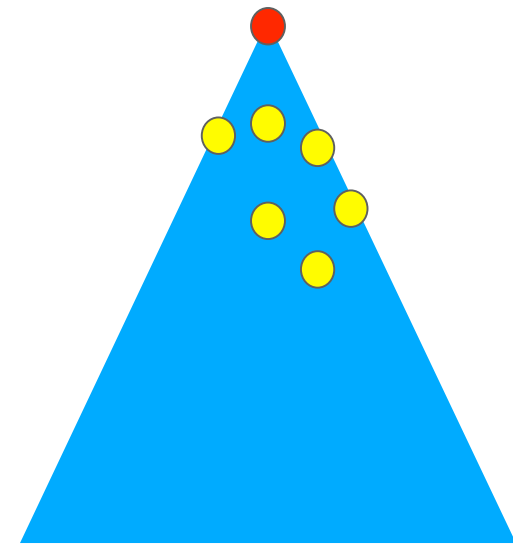
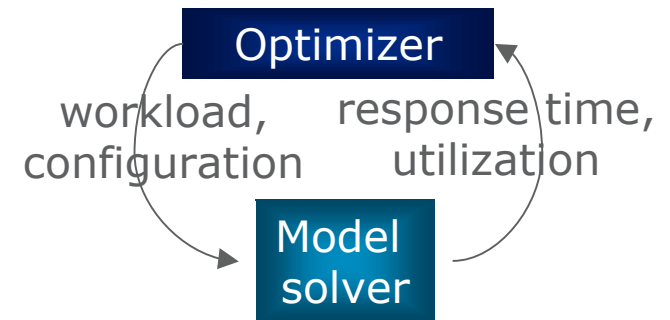
Huge parameter space to explore, NP-Complete problem.

Start from optimal configuration, search paths that reduce resource utilization while minimally reducing utility.

## Observations:

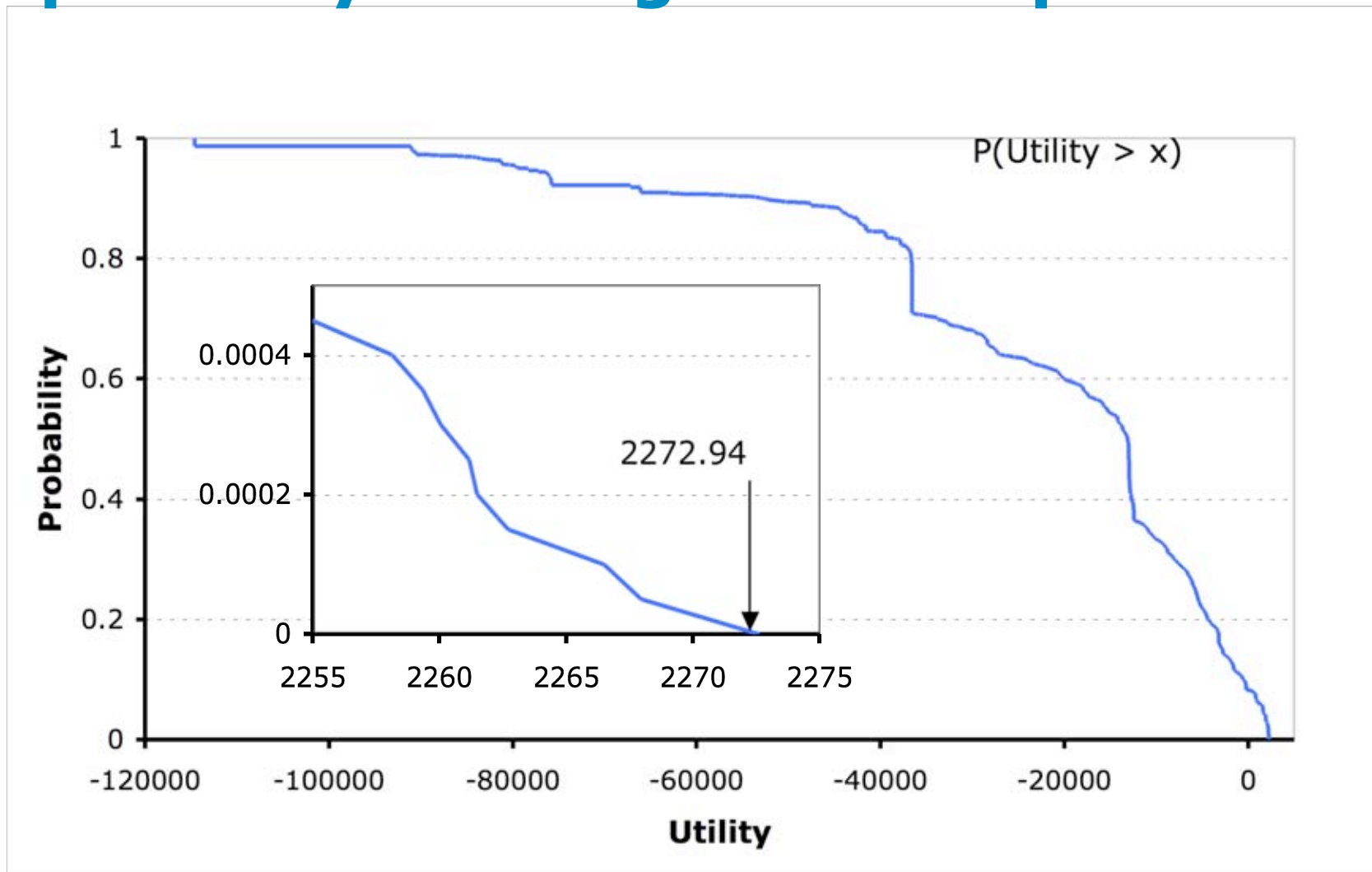
- Utility decreases when response time increases.
- Response time increases when number of replicas is reduced.
- Response time increases when CPU fraction is reduced.

Use bin-packing algorithm.





# Optimality of the generated policies



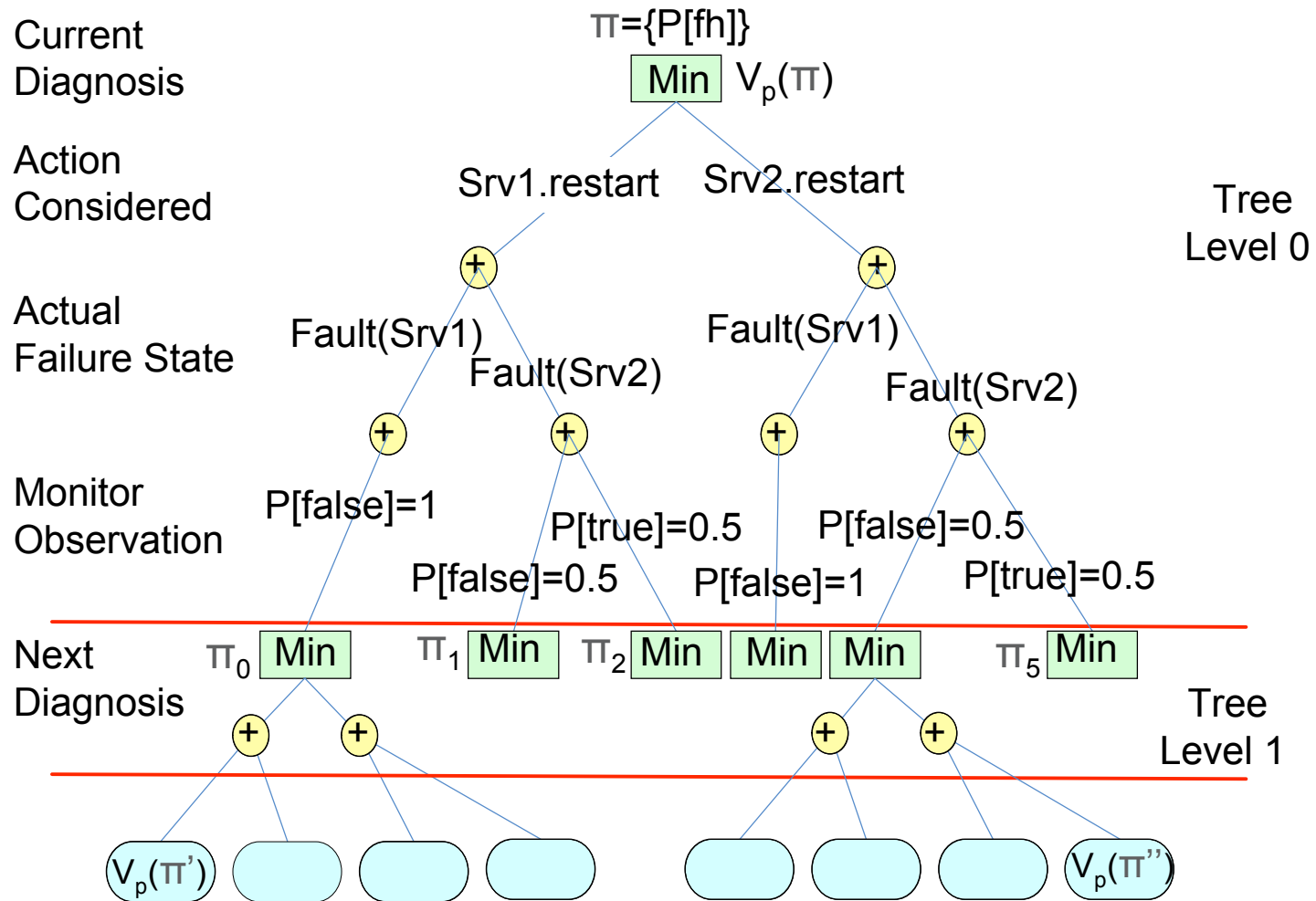
# Ideas

## Summary:

- Dynamic resource management crucial for server consolidation
- Development of adaptation policy rules a challenging problem
- Propose a hybrid approach based on offline modeling for rule generation

What about applying same approach to generate policies for automatic recovery?

# Multi-step Look-ahead Exploration



Action that minimizes “subsequent recovery cost” is chosen at each step

# Application+ Infrastructure: The One True Grail?

Vertical integration, with (controlled) sharing of information and control.

Translucent abstractions (services):

- Explicitly exposes useful information about internal operation.
- Useful, for e.g., for TCP operation over wireless links.

Example: *Accrual failure detectors*

- Provide an estimate of the probability that a host has failed rather than just a binary indication [Hayashibara, Défago, Katayama]

# Larger Scale

e-science and Lambda networking

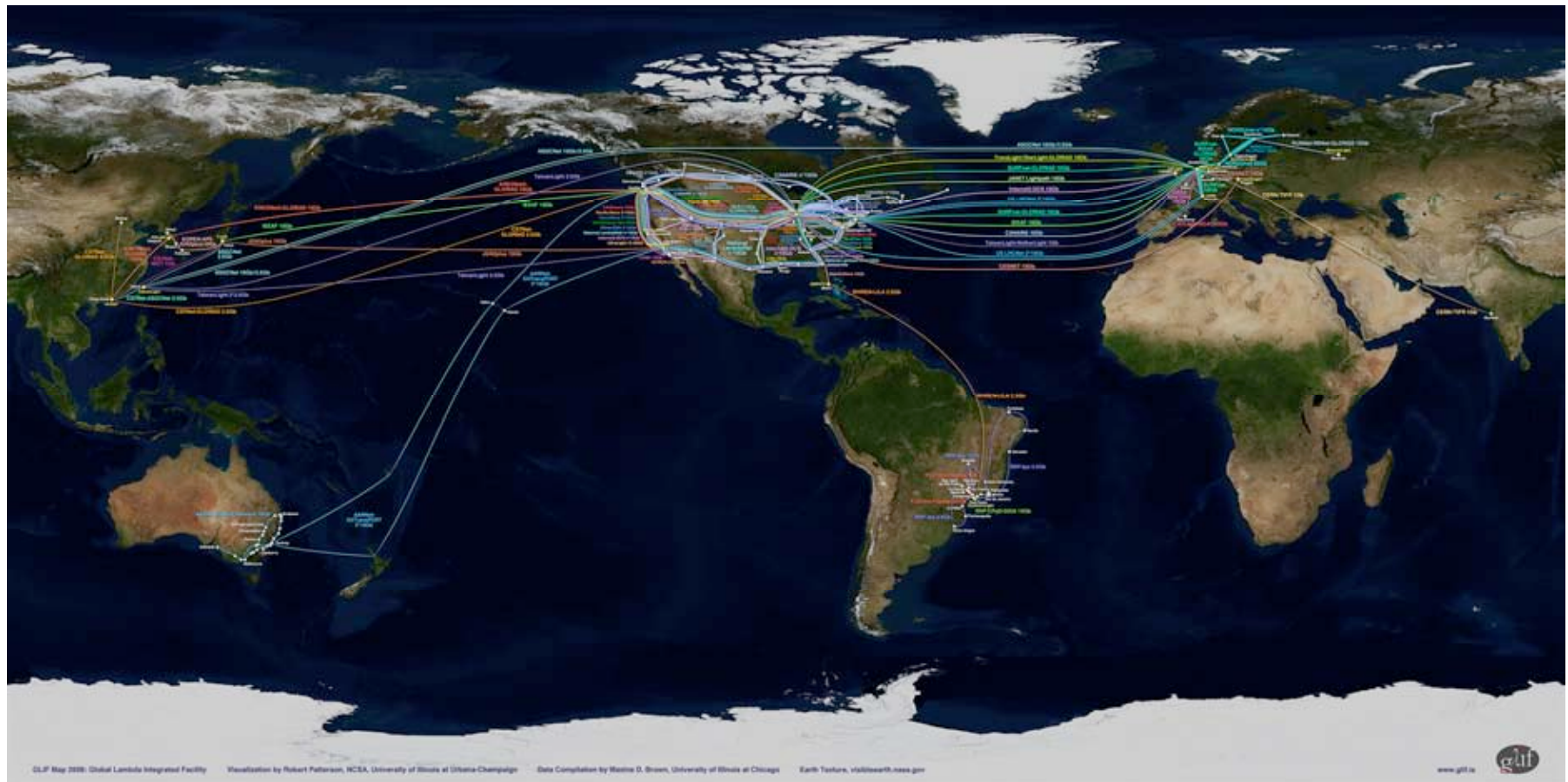
- Vertical integration of application and control plane of optical networks.

GLIF ([www.glif.is](http://www.glif.is)): Global Lambda Integrated Facility

- "GLIF is interested in developing "application-empowered" networks, in which the networks themselves are schedulable Grid resources. These application-empowered deterministic networks, or "LambdaGrids", complement the conventional networks, which provide a general infrastructure...."



# Current GLIF Testbed



# What about SOAs?

The background is a solid blue color with several white, curved, wavy lines that sweep across the frame from the top left towards the bottom right. The lines vary in thickness and opacity, creating a sense of movement and depth.

**Thank you!**