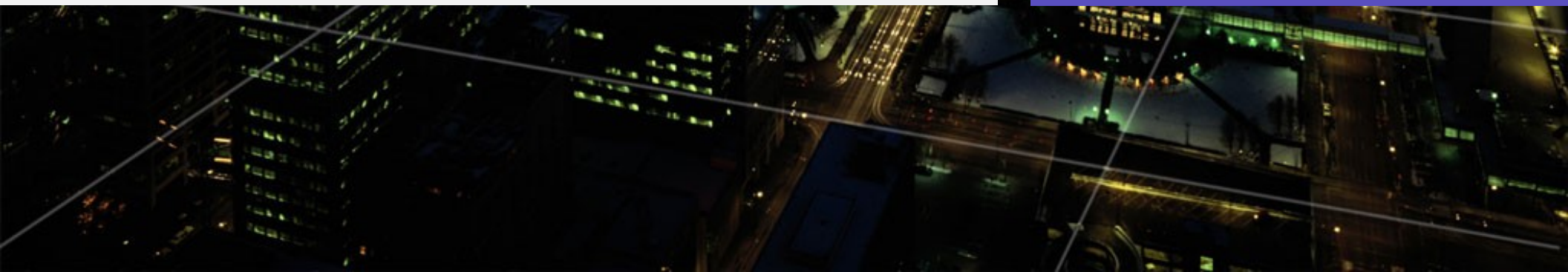
Building Grids with Jini and JavaSpaces

Carlo Nardone

Grid Program Driver – Sun Microsystems Italy

Sun microsystems



Agenda

- SOA
- Jini
- JGrid
- Jini Rio
- JavaSpaces
- (JXTA, JxGrid ...)

- thanks to Z. Juhasz, Univ. of Veszprem (Hungary) and many colleagues at Sun

Big Trends

- Grid Computing moving towards a Service Oriented Architecture (SOA)
- Draft standards “war” in Web Services space
- OGSA (Open Grid Service Architecture) is based on Web Services
- …

Big Trends

- Grid Computing moving towards a Service Oriented Architecture (SOA)
- Draft standards “war” in Web Services space
- OGSA (Open Grid Service Architecture) is based on Web Services
- … but SOA is NOT = Web Services!
- SOA architectural elements:
  - What is a service? Identity, identification …
  - How do I find a service? Discovery, registry …
  - What is the communication model?
  - What is the programming model?
  - What is the failure model?

Web Services Approach

- Assumptions:
  - WWW
  - Long-running, big services
- Service Identification : WSDL
- Service Location: UDDI
- Communication Model: SOAP
- Programming model: Document Exchange
- Failure model: HTTP failure, extended
- Optimized for
  - WWW (and firewalls)
  - Long running, different companies, etc.

Jini™

www.jini.org www.sun.com/jini

- Invented by Sun Microsystems, 1999
- Service-oriented framework for creating reliable distributed applications
- Designed with the network in mind
- Provides a spontaneous, self-healing environment
- Moves the Java platform to the network (but it is language independent!)
- Both an infrastructure and an object-oriented programming model

Jini Approach

- Assumptions:
  - Ad-hoc networking
  - Change all the time, moving objects
- Service Identification : Java types, UUIDs
- Service Location: Lookup Service/Discovery
- Communication Model: RMI/JERI
- Programming model: Java + object mobility
- Failure model: Leasing, RemoteException
- Optimized for
  - Flexibility
  - Reasonably open networks

The benefits of Jini

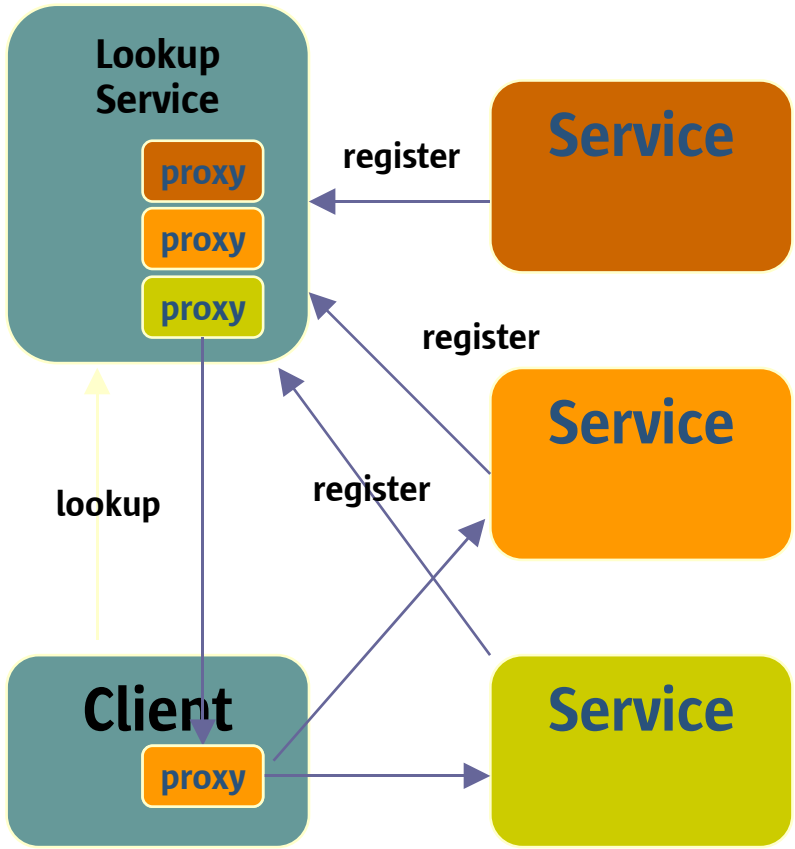
- Self-healing, fault-tolerant system
- Dynamic operation supports scaling up/down and dynamic service provisioning
- Service-oriented architecture
- Can dynamically change implementation without affecting clients
- Fast and administration free system integration

Jini Spontaneous Networking

- Jini enables clients to automatically discover services at runtime
- Associative search
  - Not by name lookup (e.g. http://some.url:port)
  - Instead: find a service that does this or that
- Loose coupling
  - Services and clients can join and leave the system (Jini federation) at any time without causing system failure

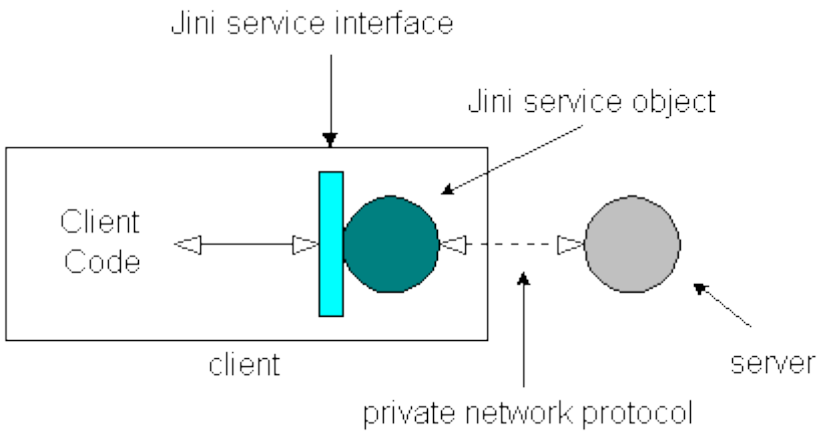
Jini Operational Overview

- Clients and service discover the lookup service
- Service register in the lookup service
- Services may join and leave the network any time
- Clients search for services in the lookup service



The Role of the Proxy

- The proxy is a Java object downloaded from the service
  - provides service or
  - transparently transfers method calls to the remote service
- Hides implementation and communication details
  - Protocol independent (TCP/IP, HTTP, SOAP, etc.)

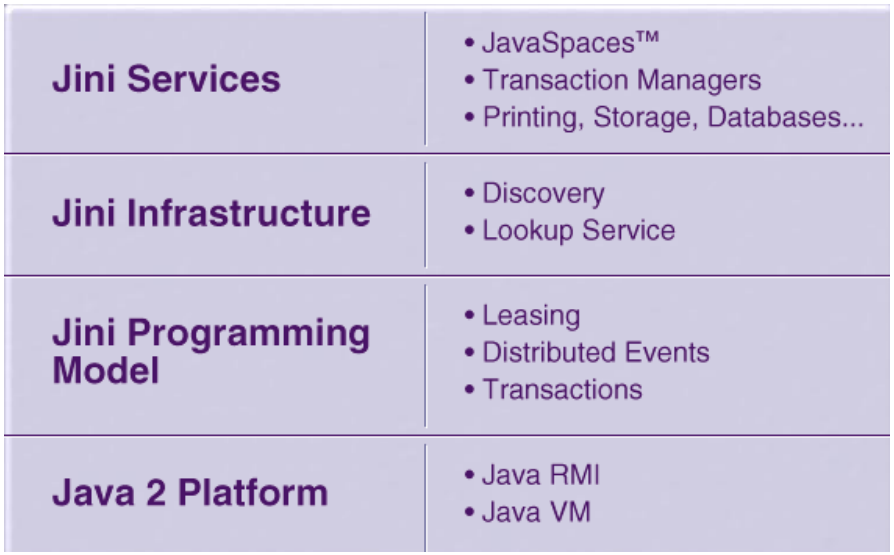


Jini Programming Model

- Jini applications lease resources
  - Provides automatic resource management and self-healing
- Can use distributed events
  - Notify about events in a publish-subscribe manner
- Can execute operations under transactions
- Can integrate non-Java implementations as well

Jini and Java

- Builds upon Java
  - Platform-neutral environment
  - Object-oriented programming model
- The Jini programming model extends Java
- The Jini infrastructure provides the basic operation mechanism: spontaneous configuration
  - discovery, join, lookup
- Jini services, client applications use the programming model and the infrastructure



JGrid

jgrid.jini.org http://pds.irt.vein.hu

- Started in 1999 at the University of Veszprem, Dept. of Information Systems (Hungary)
- Partially funded by Sun since 2003

- JGrid is a Jini-based service-oriented grid framework
- It virtualises resources and applications as Jini services
- Provides a scalable and extensible framework to create secure large-scale grid applications

Characteristics of Grid Systems

- Grid systems are dynamic
  - Accidental or planned removal or resources
  - Temporary or long-term network failure
  - Adding new services, updating existing ones
- Important requirements
  - Location transparency – no explicit server addressing (URLs don't work)
  - Loose coupling between clients and services
  - Implementation transparency

JGrid Main Features

- JGrid addresses these problems and provides:
  - Wide area service discovery
  - Platform and protocol independence based on Java and Jini
  - Advanced security architecture
  - Support for transparent sequential and parallel program execution as well as data storage

JGrid Key Services

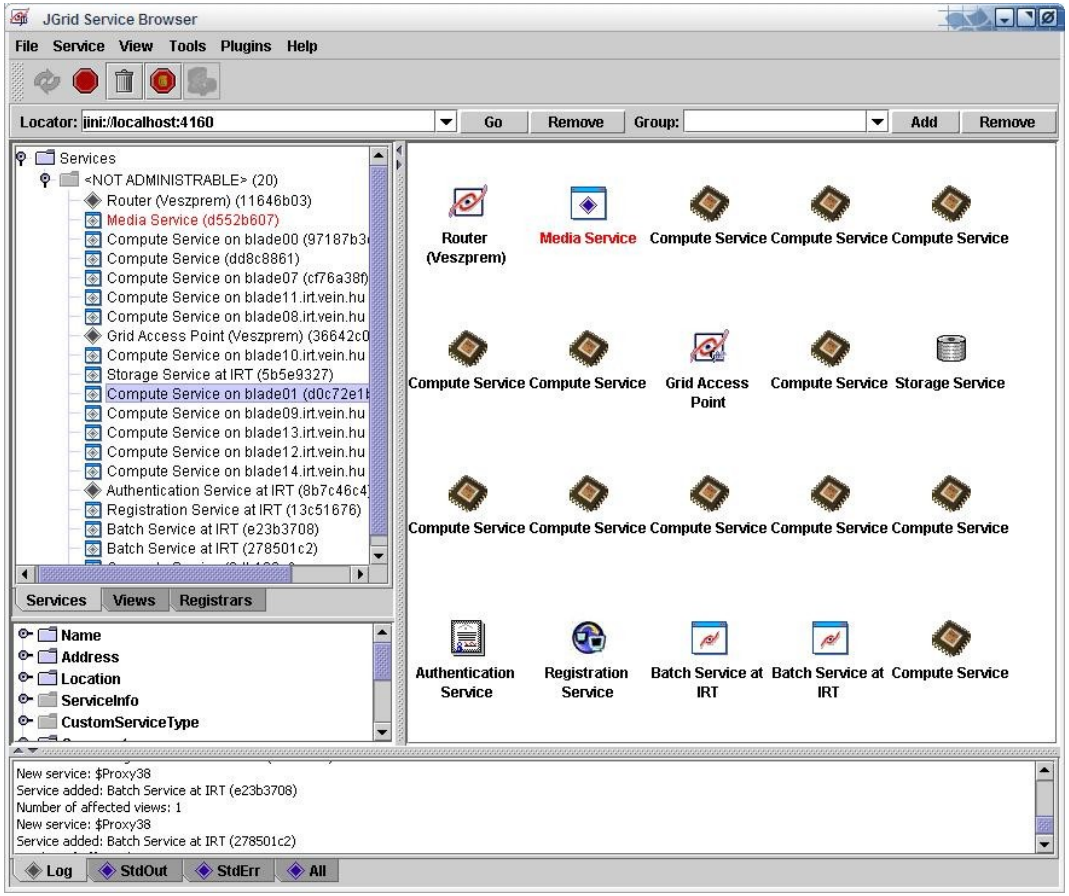
- Authentication and Registration services
  - Certificate-based access control to services and single sign-on
- Compute Service
  - Executing interactive Java programs
- Batch Service
  - Executing non-Java programs by integrating with batch environments such as Sun Grid Engine and Condor
- Storage Service
  - Providing access to user files over the network
- Broker Service
  - Helper service for locating computational services and managing program execution

JGrid Wide Area Discovery

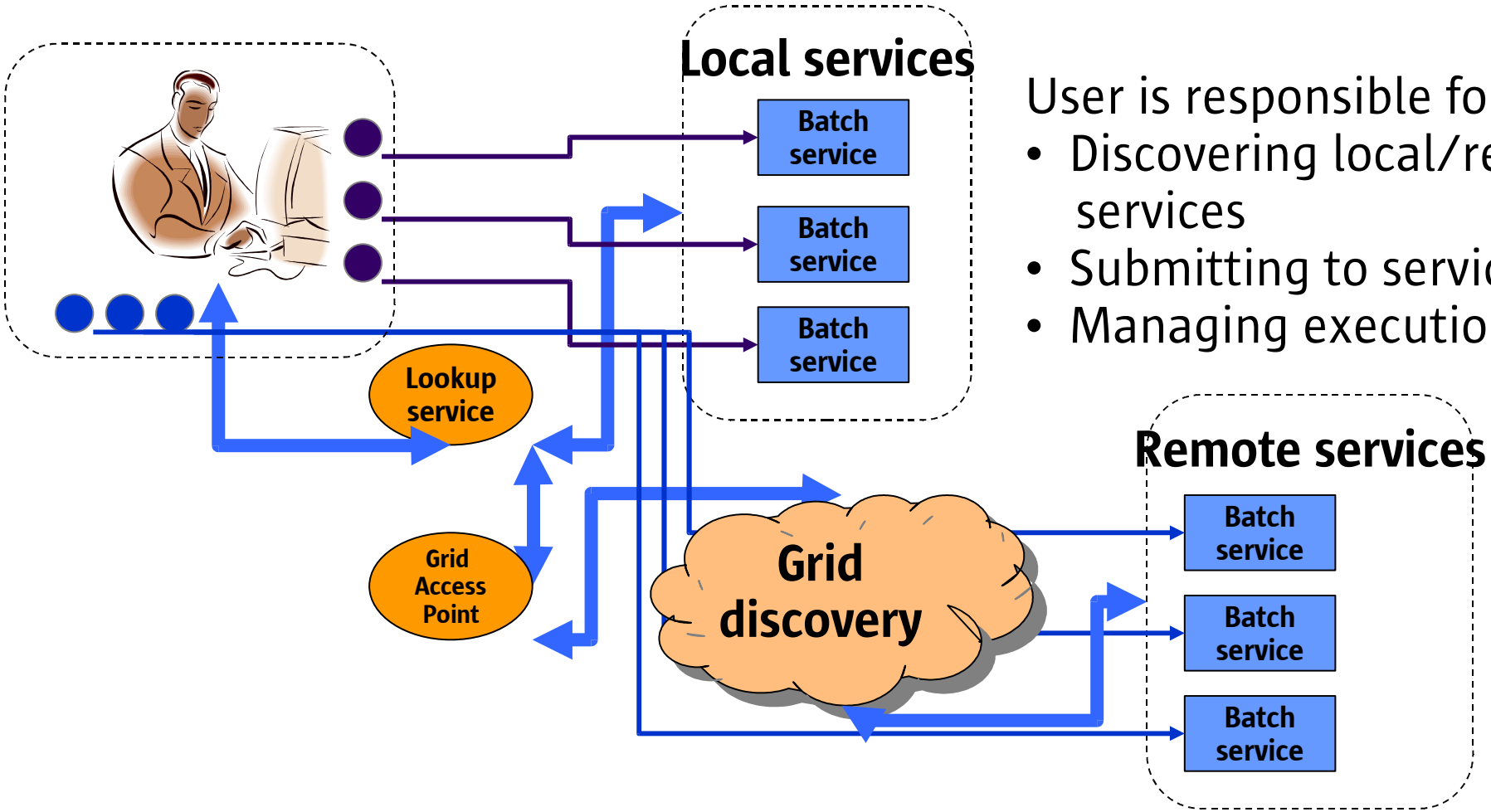
- Requires extension of standard Jini model
  - Simple lookup service federation is complicated
  - Large delays can block discovering entities
  - Some extensions use P2P and flooding – not suitable for very large systems due to unpredictable performance and network load
- JGrid approach
  - A hierarchical service overlay network
  - Lookup services provide service information input
  - Grid Access Points are the main gateways
  - Information aggregation for content-based query routing and flexible service matching

User Access – Service Browser

- The JGrid Service Browser features:
  - Jini and wide-area service discovery
  - View definitions
  - ServiceUI support
  - Security
  - Monitoring
  - Plug-in mechanism for integrating client programs with grid



Trivial JGrid Batch Execution

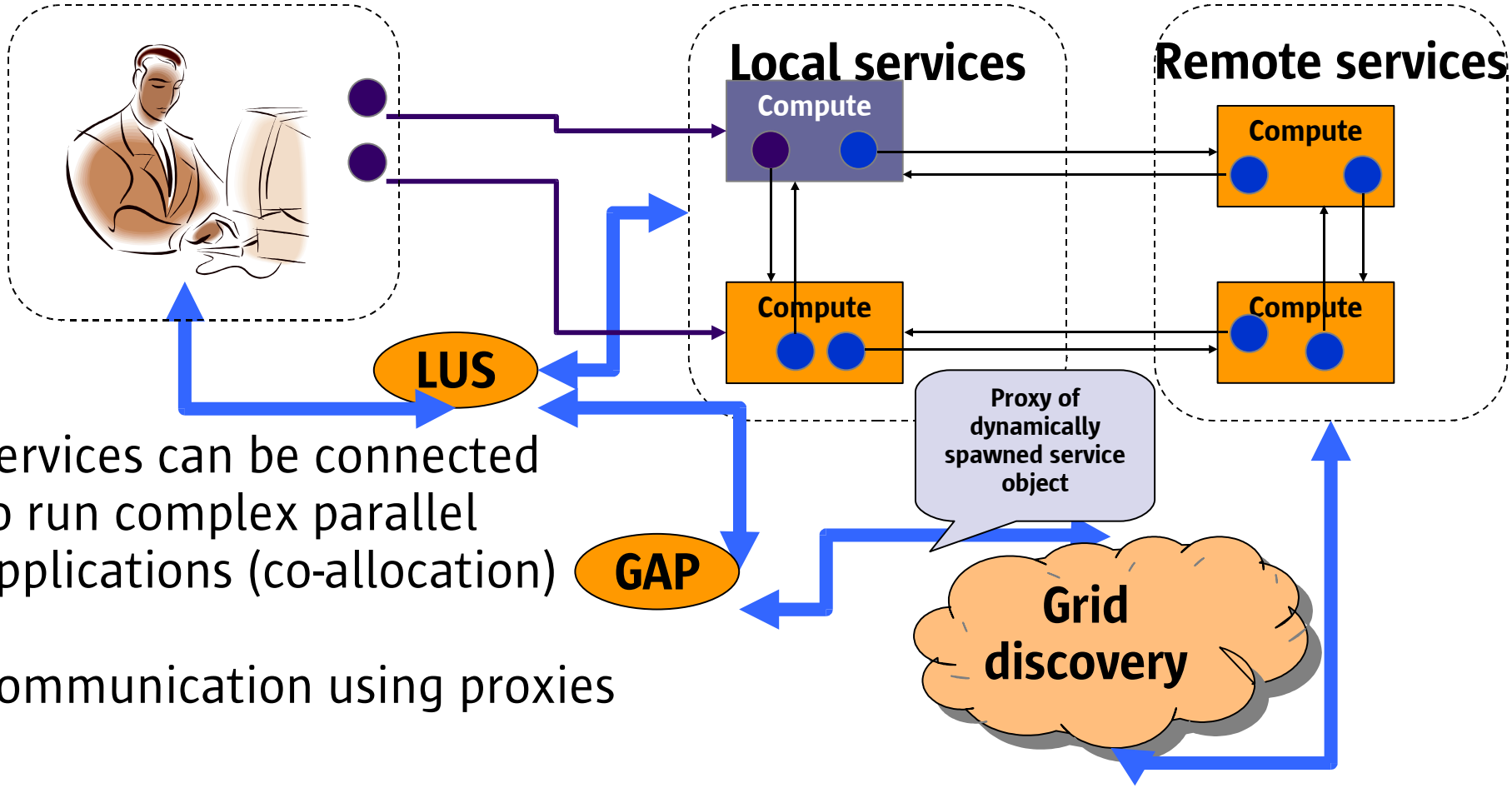


User is responsible for:

- Discovering local/remote services
- Submitting to services
- Managing execution

JGrid Interactive Execution

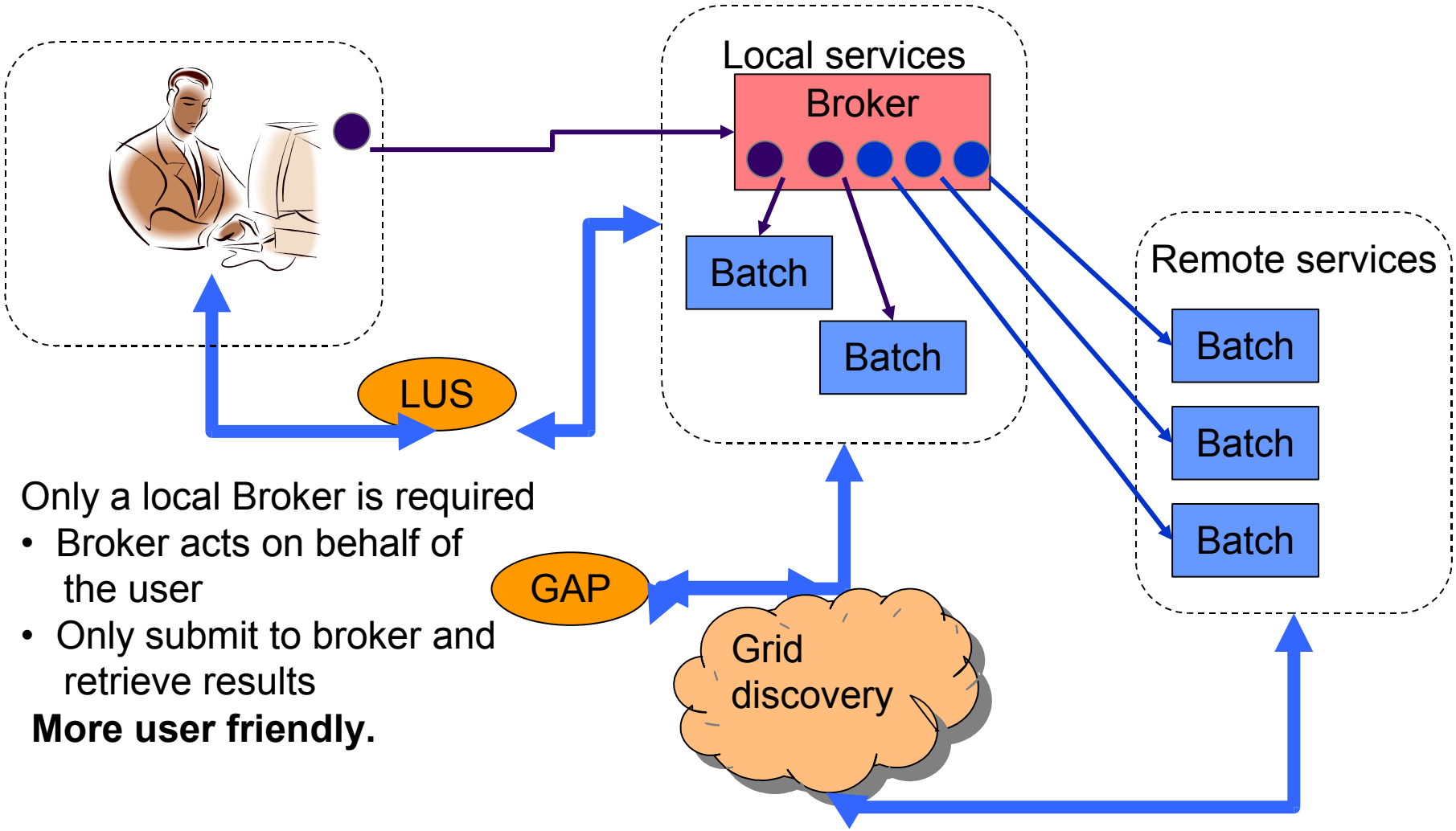
- Complex grid service or parallel program execution



Services can be connected to run complex parallel applications (co-allocation)

Communication using proxies

JGrid Batch Execution using a Broker



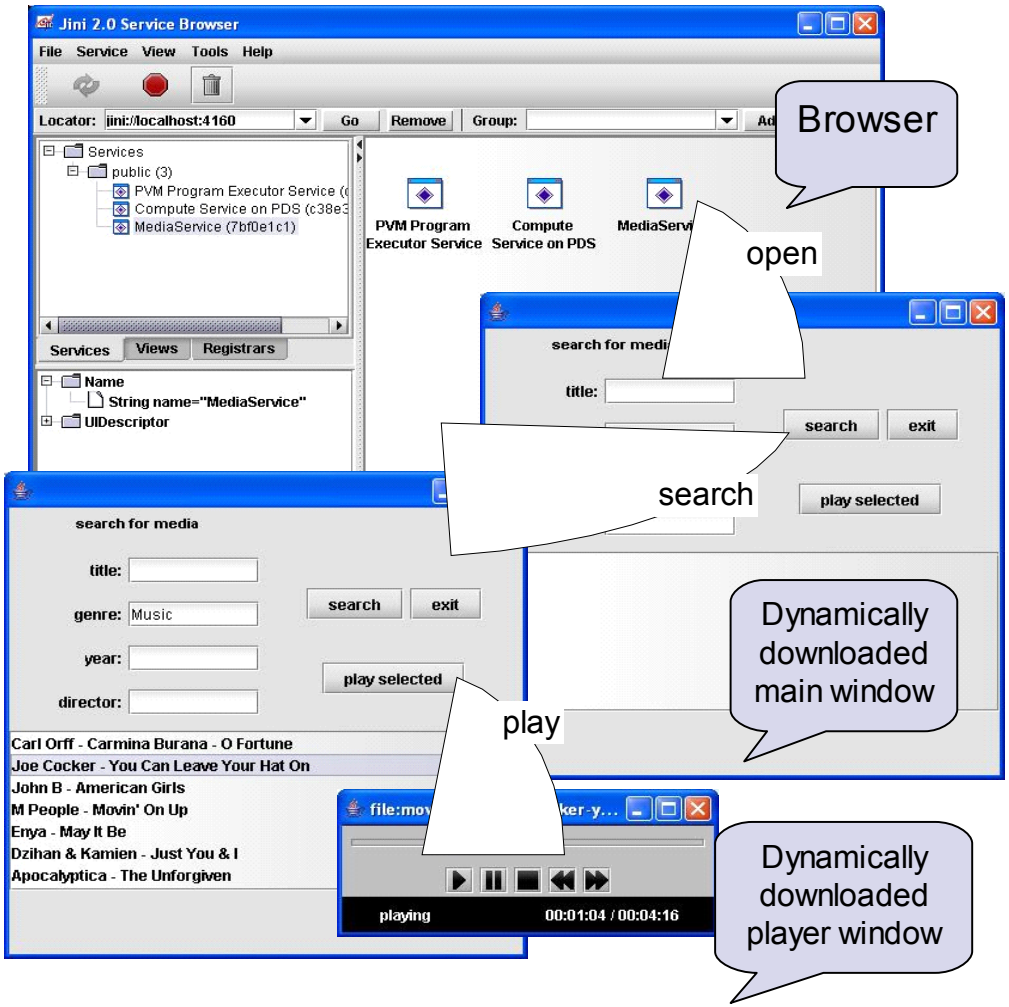
Only a local Broker is required

- Broker acts on behalf of the user
- Only submit to broker and retrieve results

More user friendly.

JGrid User Interfaces

- JGrid services use dynamic user interfaces
- User interface code arrives from service
  - No need to install clients
- Example:
  - Use of Media Service
- Jini (ServiceUI) can provide multiple, alternative user interfaces to services
  - Jini is unprecedented in this respect



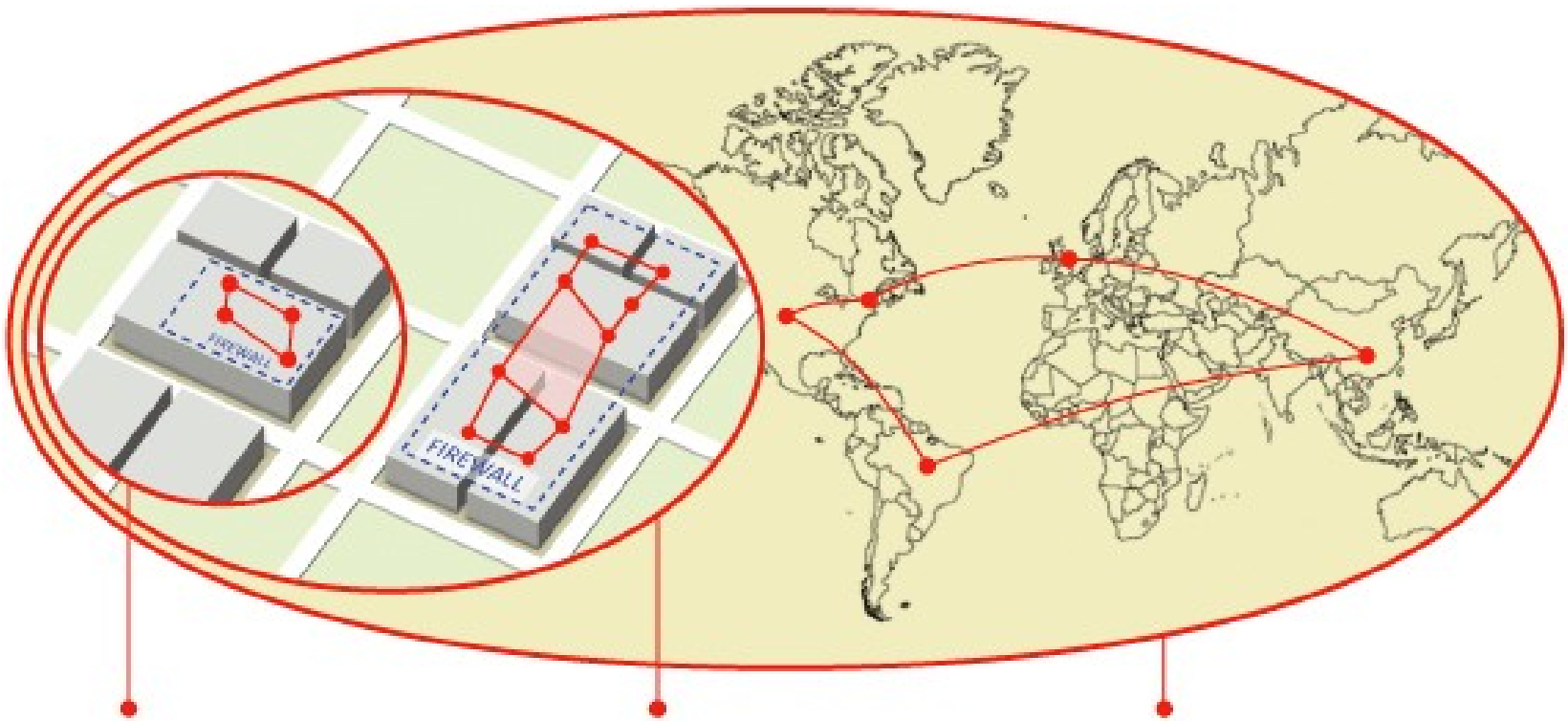
Some possible uses of JGrid

- JGrid can be used for non-computational domains as well
- Example services:
  - Streaming media delivery
  - News services
  - On-demand computing
    - Media processing and delivery, spam filtering, long-lived service applications
  - Compound services
  - Banking for more effective access for customers
  - Business-to-business applications

… but what is Grid Computing?

- Purist view vs pragmatic view
- “Don't worry about definitions – if it's distributed, connected by network, managed by middleware, it's a grid” (Wolfgang Gentzsch, D-Grid)
- Most businesses need to adopt fully distributed, virtualized architectures in their Datacenter before considering any Grand Grid Vision
- 3 phases of Grid adoption:
  - Cluster Grid
  - Enterprise Grid
  - Global Grid

Phases of Grid Computing



Cluster Grid
Departmental Computing

- Simplest Grid deployment
- Maximum utilization of departmental resources
- Resources allocated based on priorities

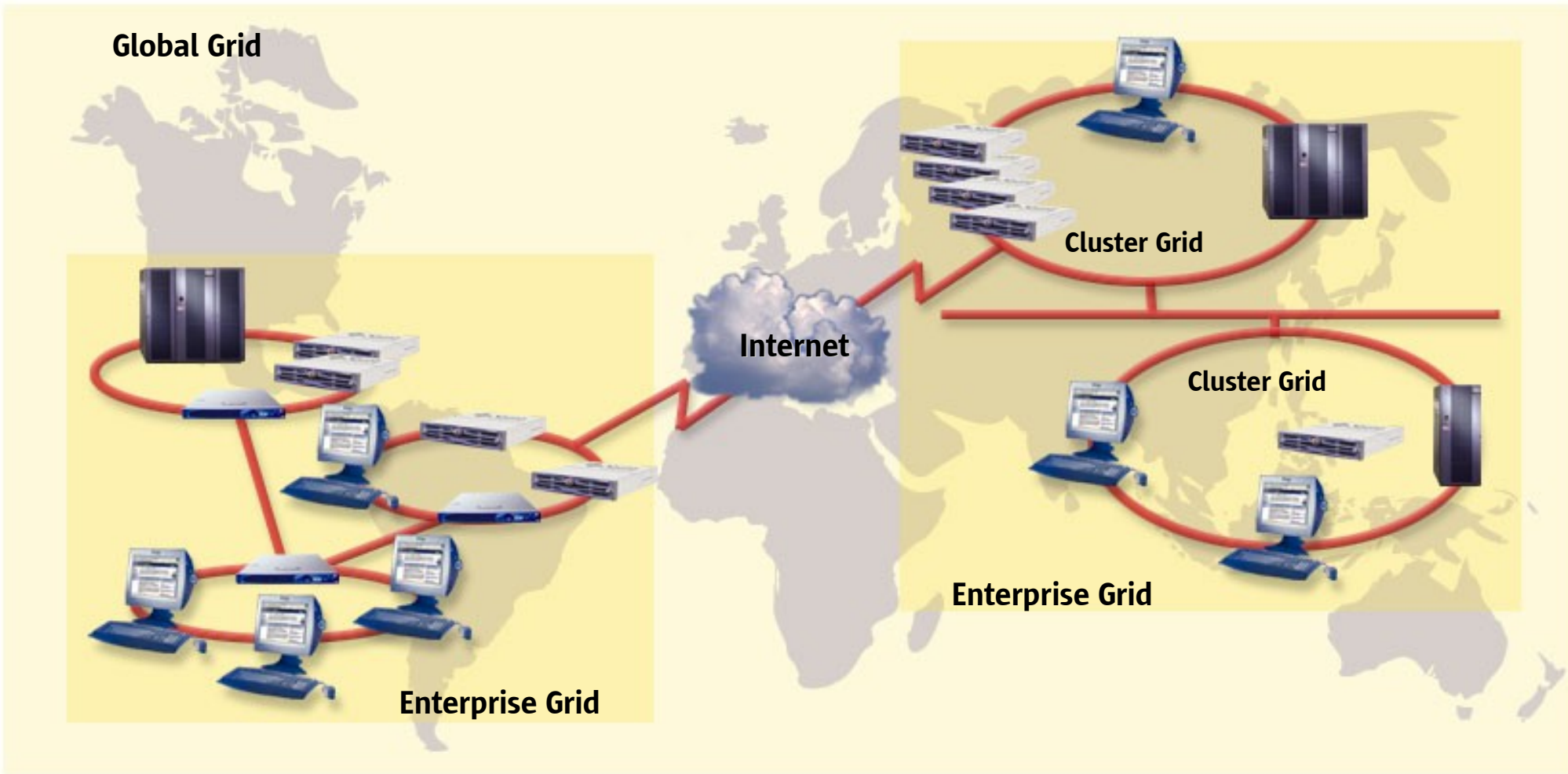
Enterprise Grid
Enterprise Computing

- Resources shared within the enterprise
- Policies ensure computing on demand
- Gives multiple groups seamless access to enterprise resources

Global Grid
Internet Computing

- Resources shared over the Internet
- Global view of distributed datasets
- Growth path for enterprise Grids

From Local to Global



Grid Adoption Trend

HPTC Grids
Tech/Tech:
Technical End User
Technical Application

End user:
- Academic/Research

Higher Priorities:
- Price/Performance
- Teraflops

Lower Priorities:
- Manageability
- HA
- SLA's
- Cost of ownership

Tech Grids
Com/Tech:
Commercial End User
Technical Application

End user:
- Manufacture
- EDA
- Oil and Gas
- Finance
- Pharma

Higher Priorities:
- Cost Acquisition
- Price/Performance
- Performance
- Manageability

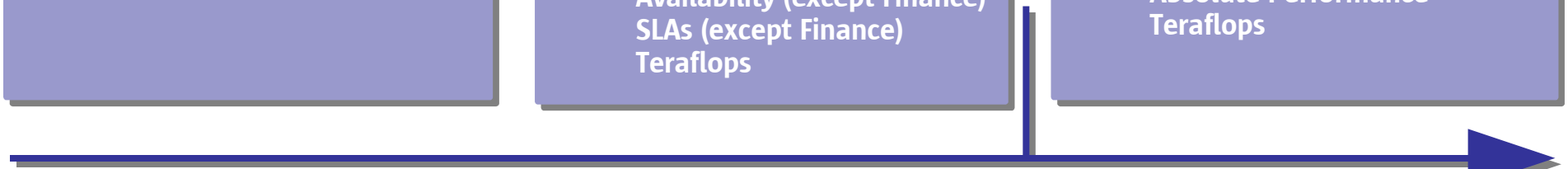
Lower Priorities:
- Availability (except Finance)
- SLAs (except Finance)
- Teraflops

Data Center Grids
Com/Com:
Commercial End User
Commercial Application

End user:
- Enterprise
- Service Providers

Higher Priorities:
- Availability
- SLAs
- Utilization
- Manageability
- Cost of ownership

Lower Priorities :
- Acquisition cost
- Price/Performance
- Absolute Performance
- Teraflops



Case Study: Financial Services App

- The Application
  - Fraud detection system used daily by millions of consumers worldwide
  - 1,000s transactions per second
  - 24x7
  - 0.3 TB of active data
  - Steady growth in throughput & data

Case Study: Financial Services App

- The Application
  - Fraud detection system used daily by millions of consumers worldwide
  - 1,000s transactions per second
  - 24x7
  - 0.3 TB of active data
  - Steady growth in throughput & data
- The Architecture
  - classic 2-tiered system
  - centralized application server, random-access data on disk
  - one giant domain on a large SMP (12 -> 32+ CPUs)
  - classic C/C++ hand-crafted code
  - single threaded, multi-process design, primitive data structures in shared memory, queues for process comm.
  - serious mathematical computations for each transaction

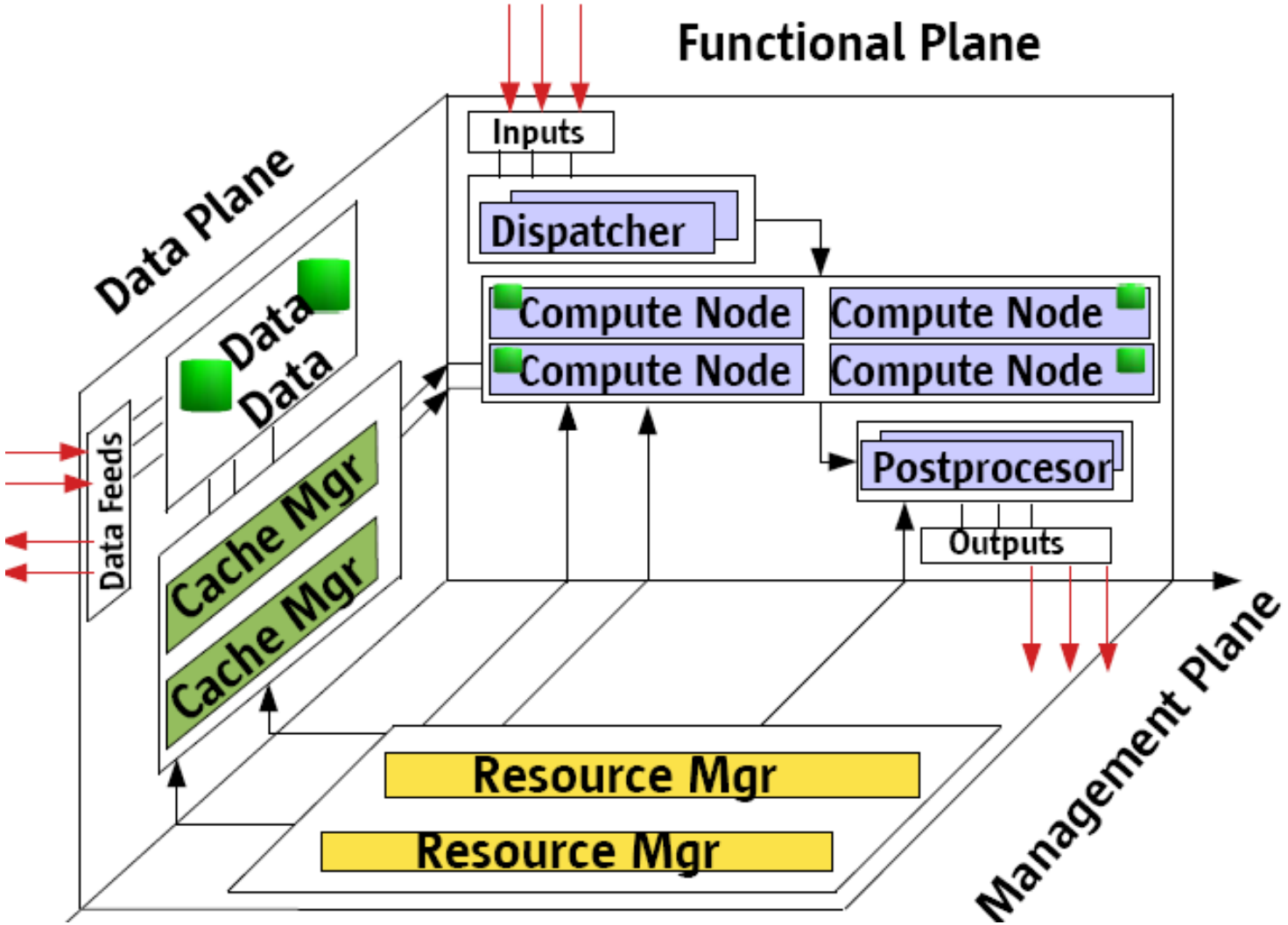
Case Study: Financial Services App

- Distributing the architecture
  - many small, cheap, fast compute nodes
    - View grid as unlimited distributed RAM
  - Divide data into "buckets"
  - Distribute, "cache" buckets into compute nodes
  - Dispatch each transaction to the "right" compute node
    - HA via N + k architecture
  - N compute nodes, k "spare nodes"
  - Jini/Rio based automatic provisioning, fault detection and recovery

Case Study: Financial Services App

- Distributing the architecture
  - many small, cheap, fast compute nodes
    - View grid as unlimited distributed RAM
  - Divide data into "buckets"
  - Distribute, "cache" buckets into compute nodes
  - Dispatch each transaction to the "right" compute node
    - HA via N + k architecture
  - N compute nodes, k "spare nodes"
  - Jini/Rio based automatic provisioning, fault detection and recovery
- Results
  - 2x better throughput, 4x better TCO/3yrs, recovery time down 9x
  - “I guess Java really works in heavy-duty environments”
  - “With such a throughput, real-time processing is possible”
  - “With this kind of resilience, scalability and cost, who needs mainframes?”

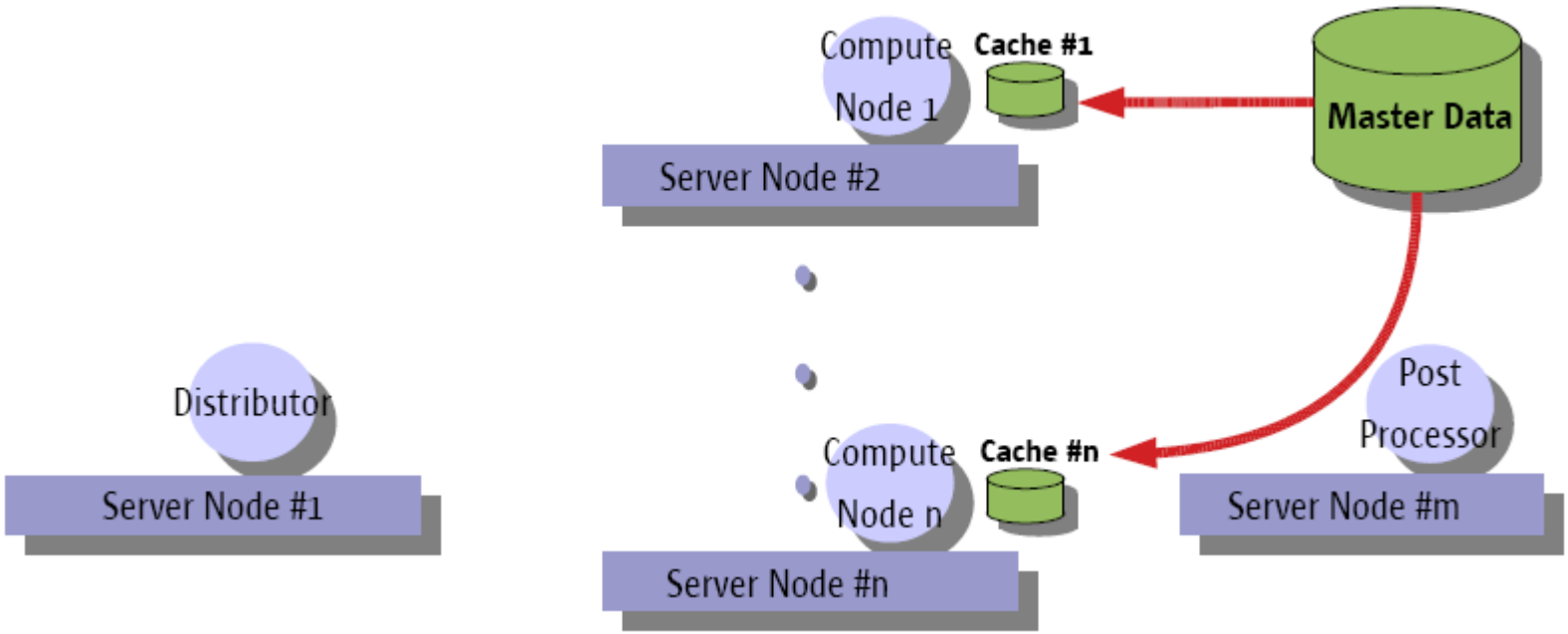
Distributed Architecture



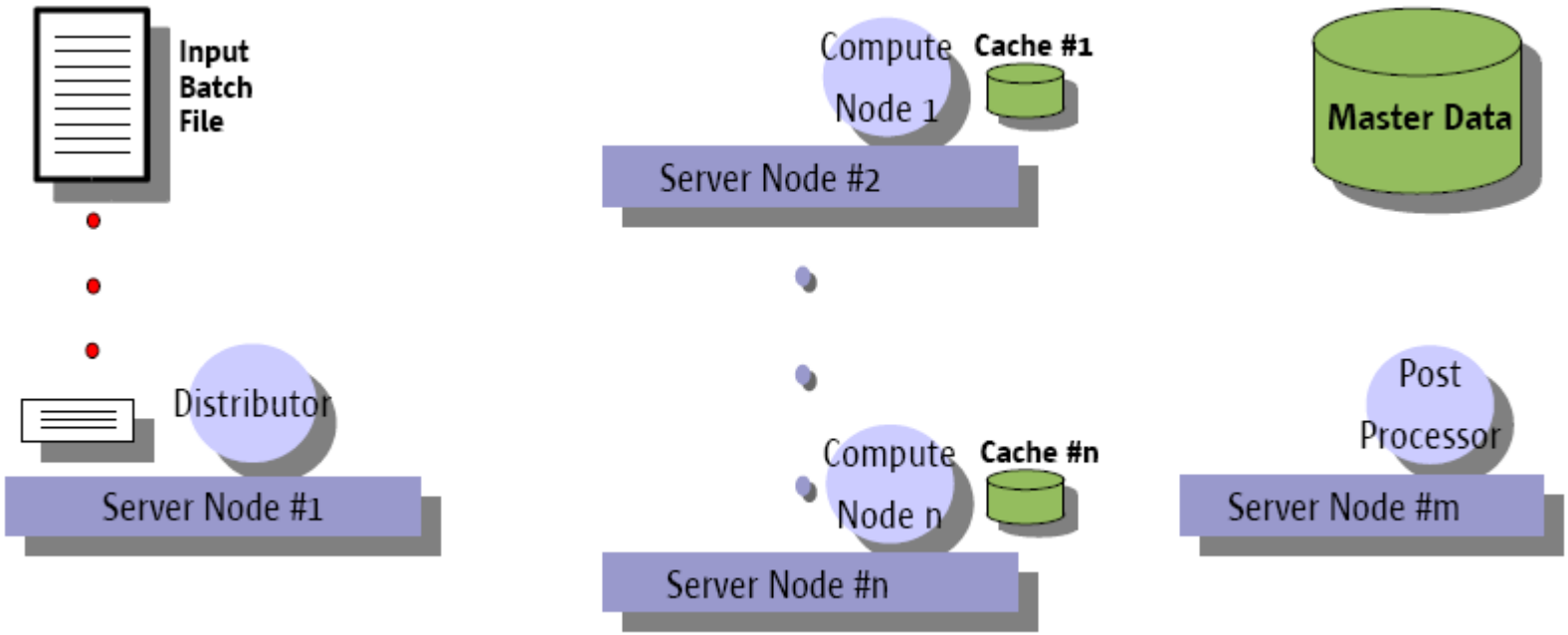
Distributed Approach

- Application fit
  - Autonomous transactions
  - Partitionable data
  - Deterministic, 1-to-1 map between transaction & partition
  - Many real-world examples: credit scoring, stock trading, indexed search, on-line banking, on-line catalog, payroll processing, readonly data marts, 90% batch systems

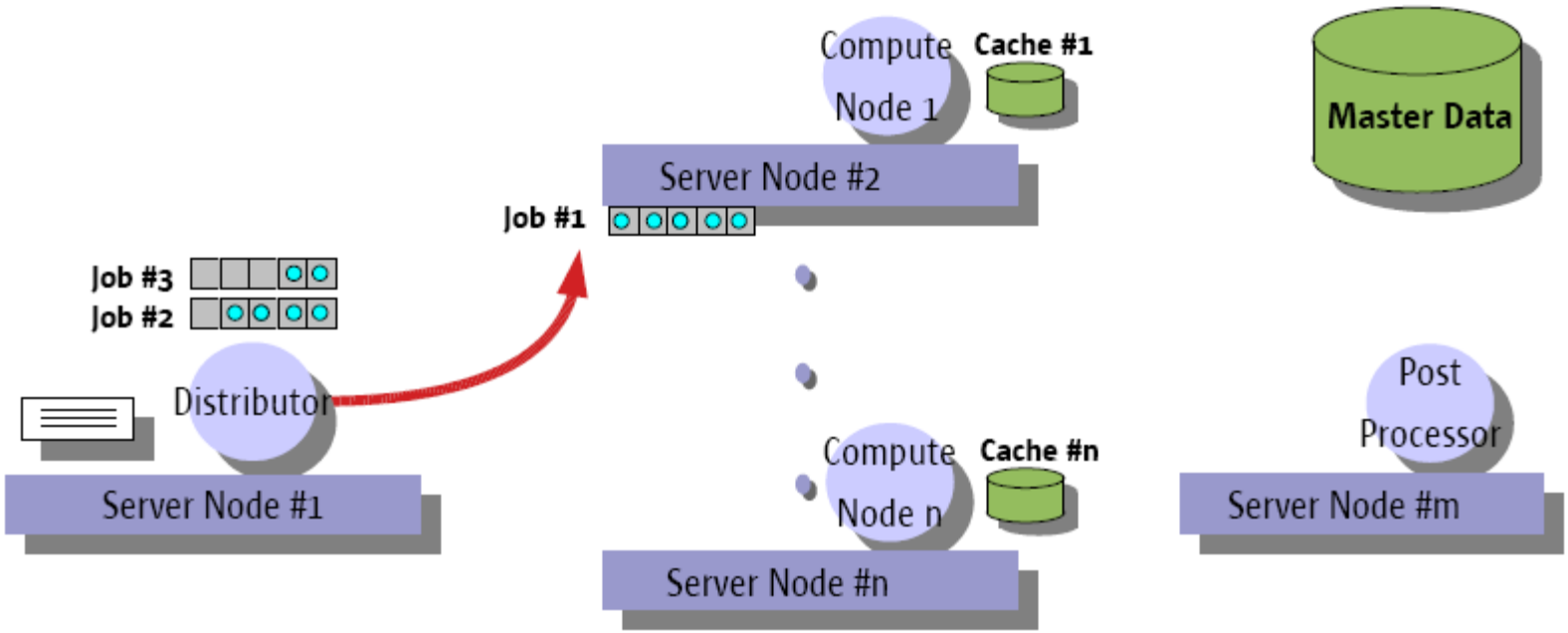
Job Scheduling



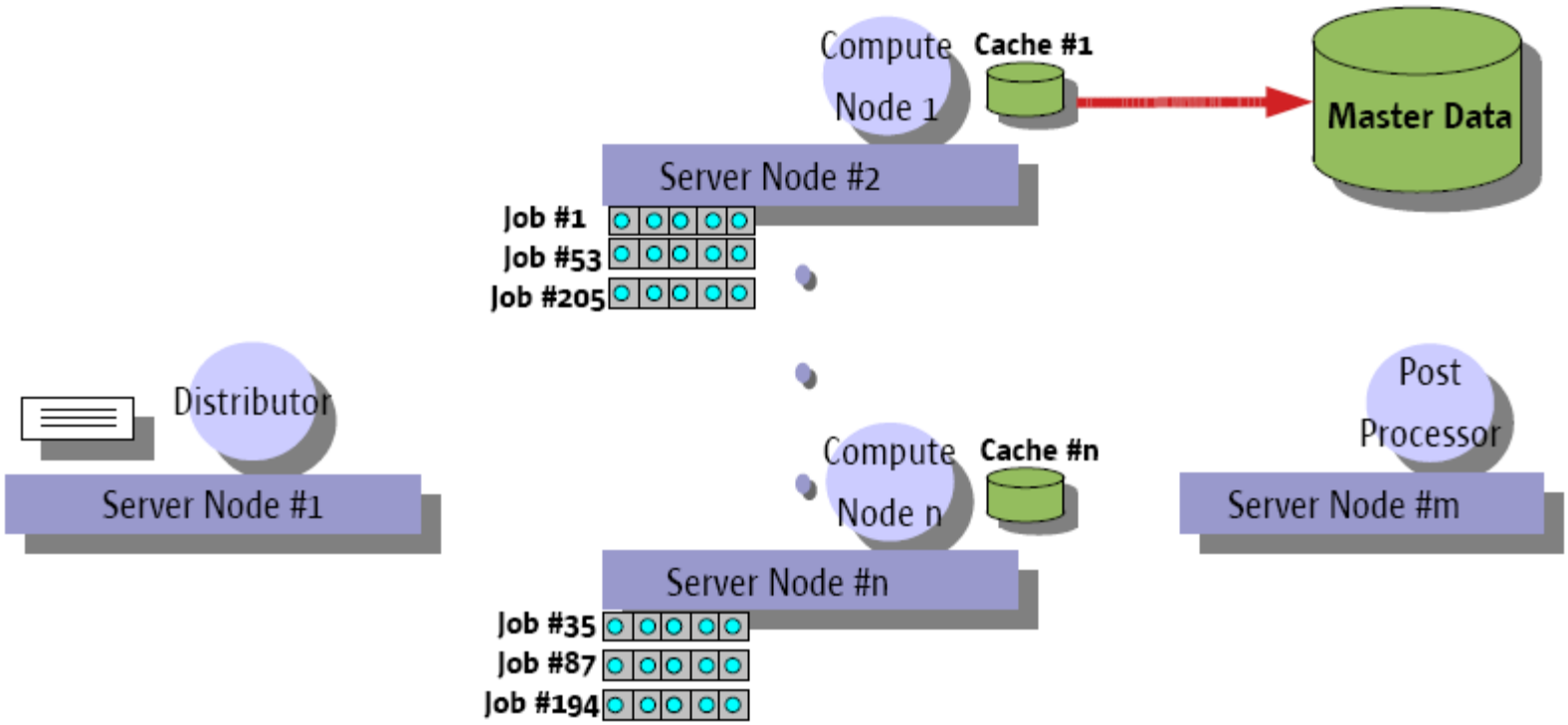
Job Scheduling



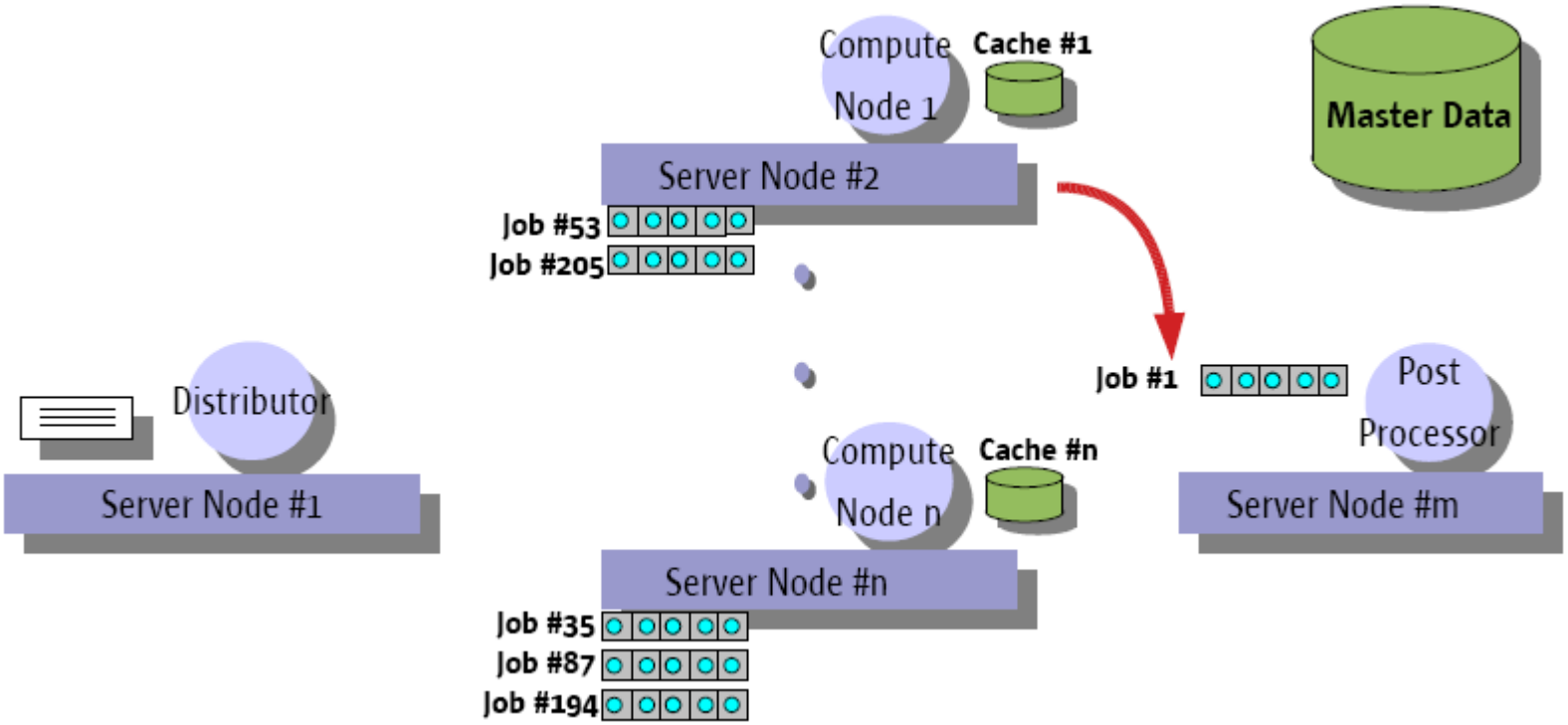
Job Scheduling



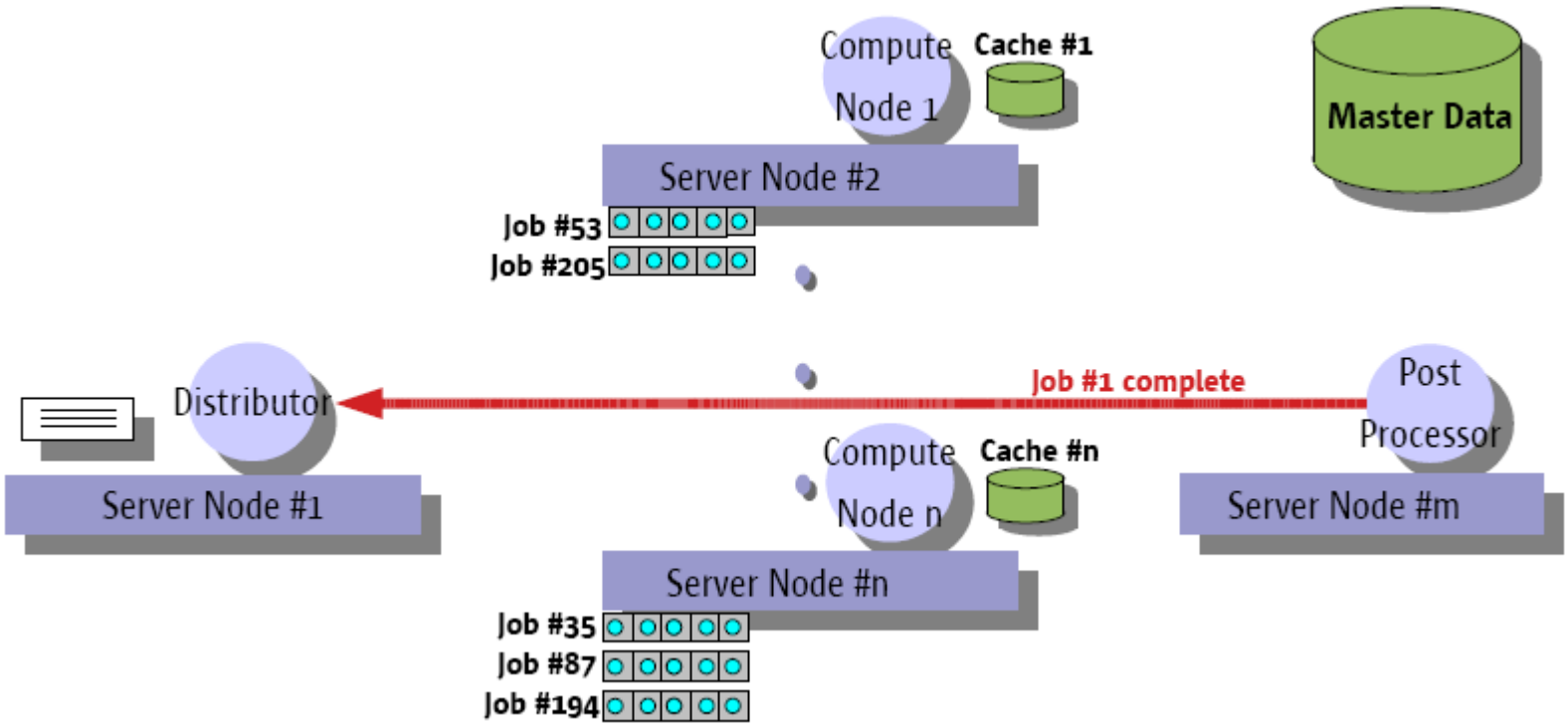
Job Scheduling



Job Scheduling



Job Scheduling



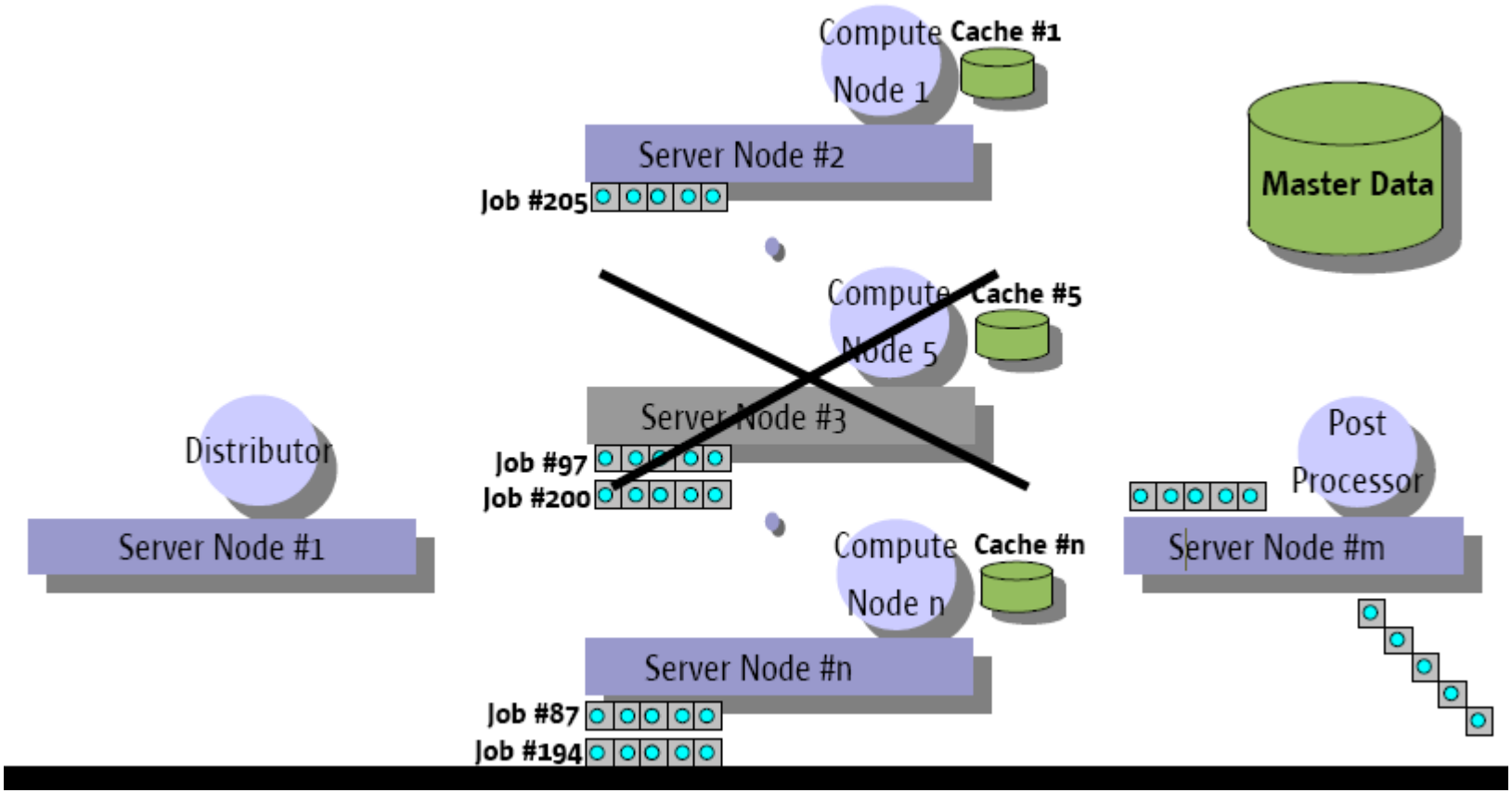
Distributed Approach

- Application fit
  - Autonomous transactions
  - Partitionable data
  - Deterministic, 1-to-1 map between transaction & partition
  - Many real-world examples: credit scoring, stock trading, indexed search, on-line banking, on-line catalog, payroll processing, readonly data marts, 90% batch systems
- Resource Management
  - Basically, what RAID is to storage, grids are to compute power
  - but ... management is hard!
    - Deployment, Recovery, Monitoring ...
  - Jini Rio to the rescue!
    - Dynamic Service Provisioning
    - Automatic failover detection & recovery management
    - Service Monitoring & Management

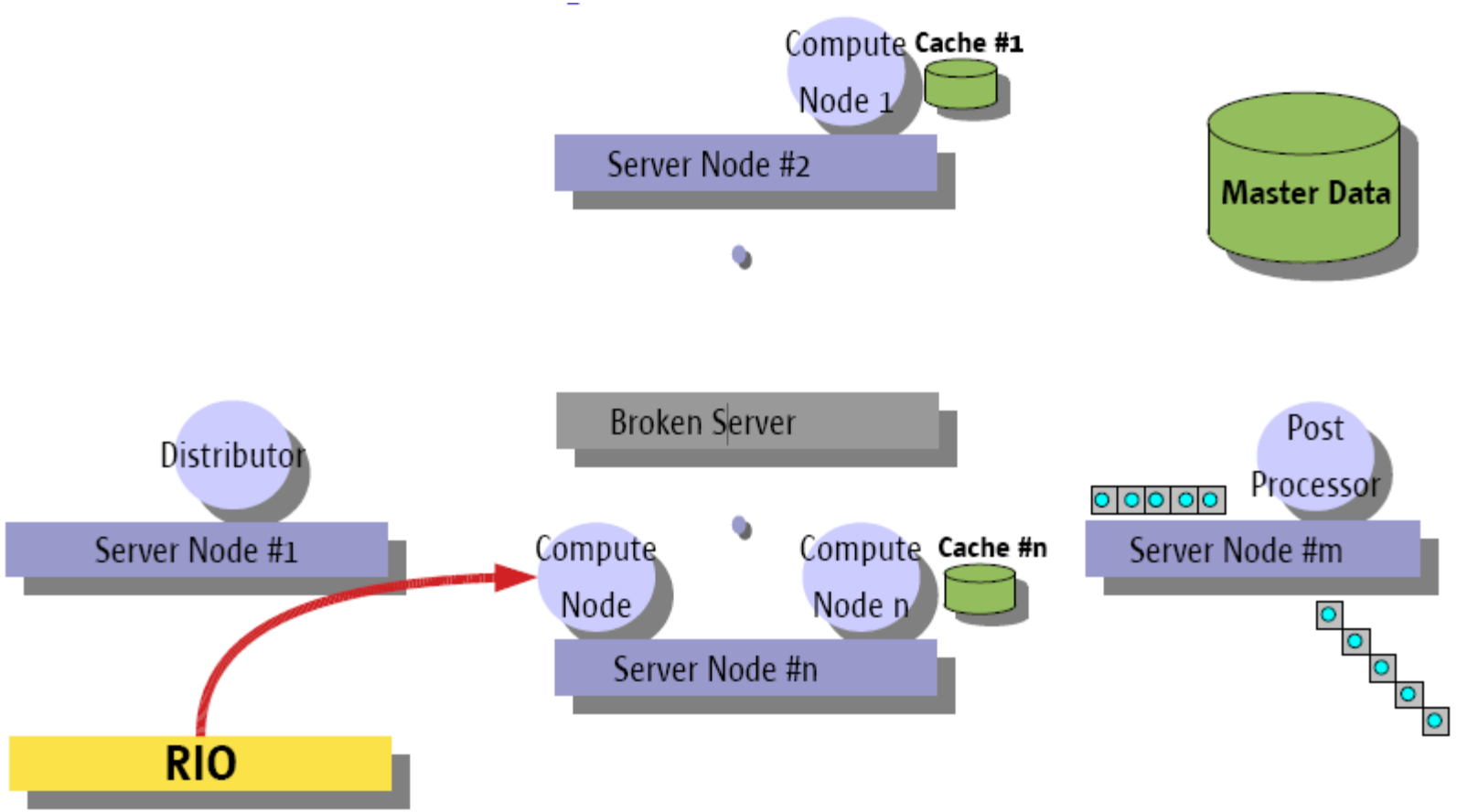
The Dynamic Adaptive Grid

- Jini Rio Overview
  - Open source Jini project
  - Dynamic service provisioning
  - Handles service fail over
  - Manages Service Level Agreements (SLAs)
  - Jini Service Beans (JSBs)
    - Simple component model
- Rio Components
  - Provision Manager
    - Handles deployment, recovery, and enforcement of SLAs
  - Cybernodes
    - Light weight container that handles service lifecycle and monitors SLAs
  - Applications may use Rio API to provide application-specific fail-over logic

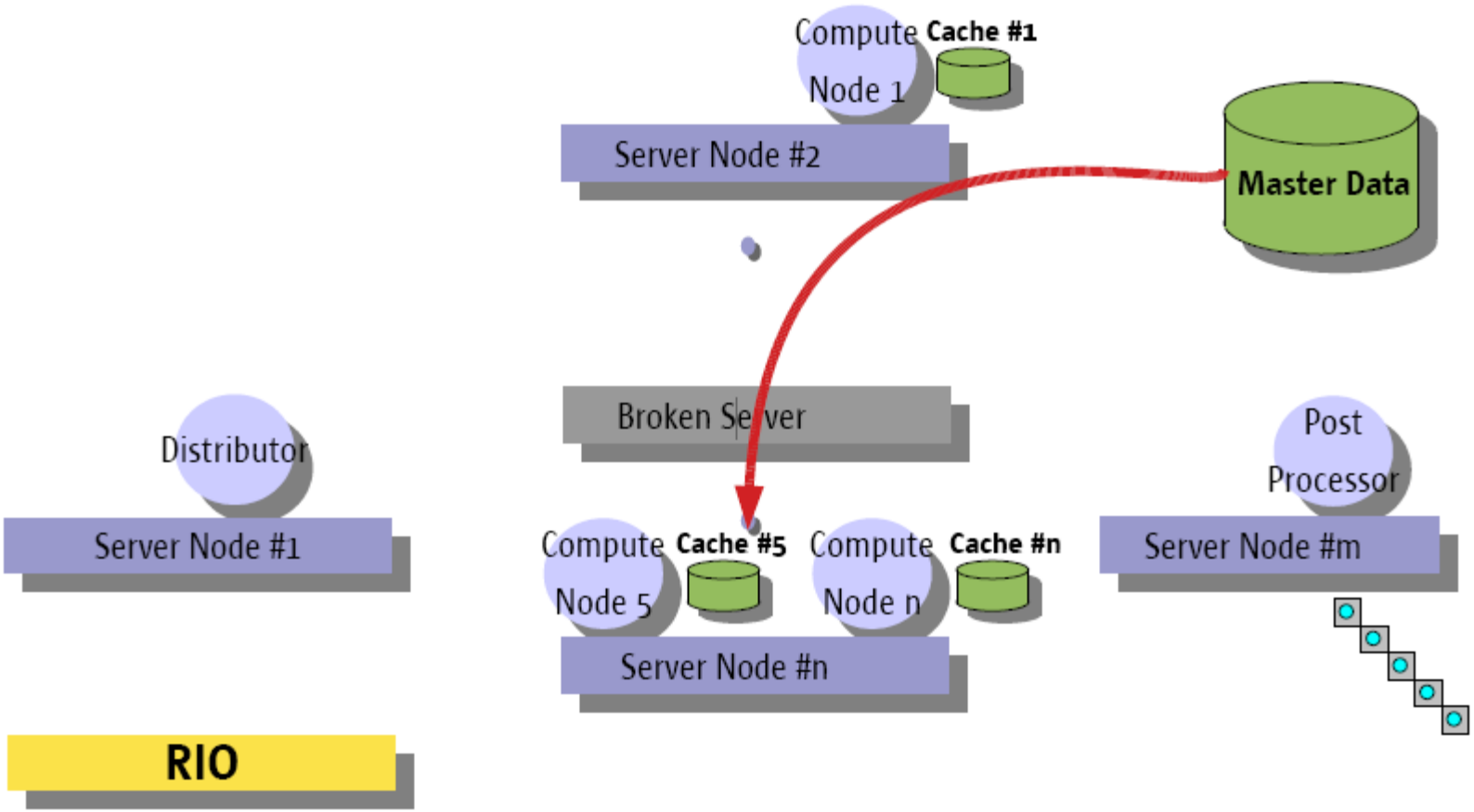
Dynamic Failover



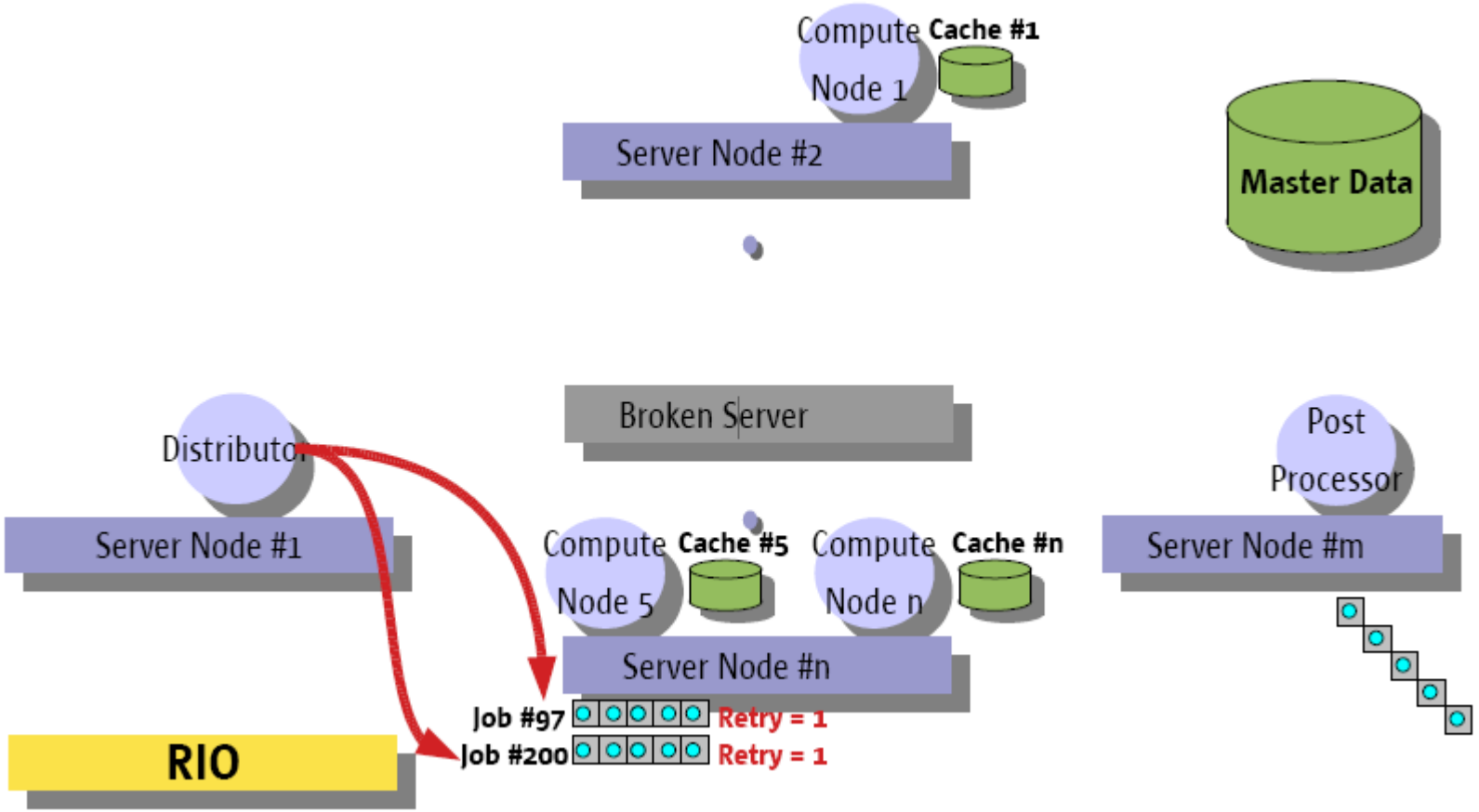
Dynamic Failover



Dynamic Failover



Dynamic Failover



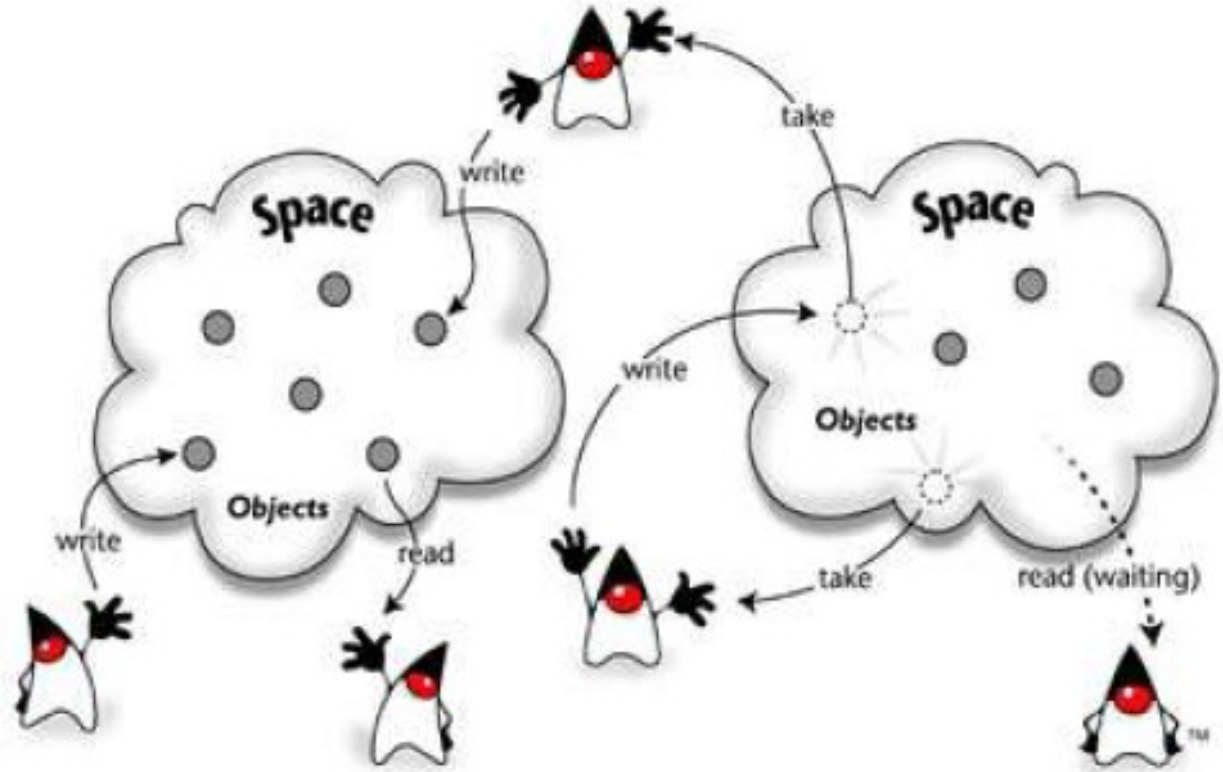
Yet Another Approach ...

- Distributed software architecture is complex
  - Remember the 8 fallacies of Network Computing by Peter Deutsch?
  - Latency, memory access, partial failure, concurrency
- Simplicity is Key
- A Complete Distributed Framework in Only 4 Basic Calls:
  - Write
  - Read/ReadIfExists
  - Take/TakeIfExists
  - Notify

JavaSpaces™

- A model for building loosely coupled systems
- An associative shared memory abstraction that clients on the network can use to share and exchange objects
  - Remember Linda?
  - No “passing messages”, “invoke remote object”
- Benefits
  - Anonymity between applications
  - Uncoupled communication
  - Programs can communicate through time or space
  - Vast savings in design and development time

What is a Space?

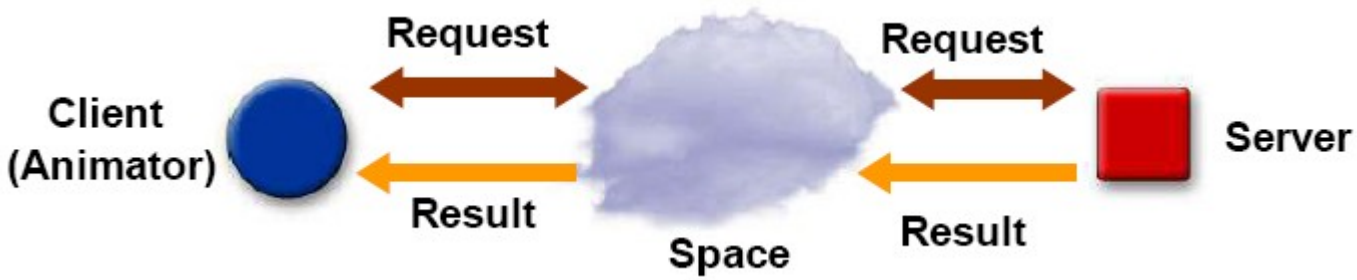


- A place on the network to share and store objects
- Associative shared memory for the network
- Unifies storage and communications
- Simple design -> only four basic operations

JavaSpaces and Jini

- An example of a Jini enabled service
- Extensive use of Jini technology programming model
  - Transactions (distributed consensus)
  - Leases (resource reclamation)
  - Remote Events (asynchronous notification)
  - Same matching rules as service attributes
- Code downloading
  - Interface defined as Java language interface
- Created by same people!

A Dynamic Server Farm



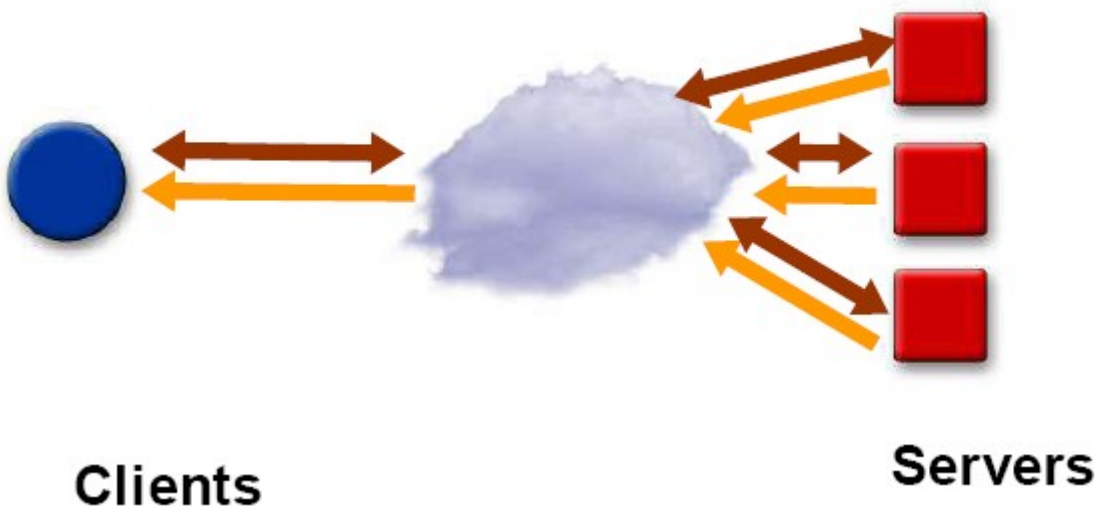
- Animator needs to render movie frames
  - Writes "request for rendering" entries
  - Takes render results written back
- Server processes takes
  - Takes "request for rendering" entries
  - Executes each request, writing back results

A Dynamic Server Farm

Clients

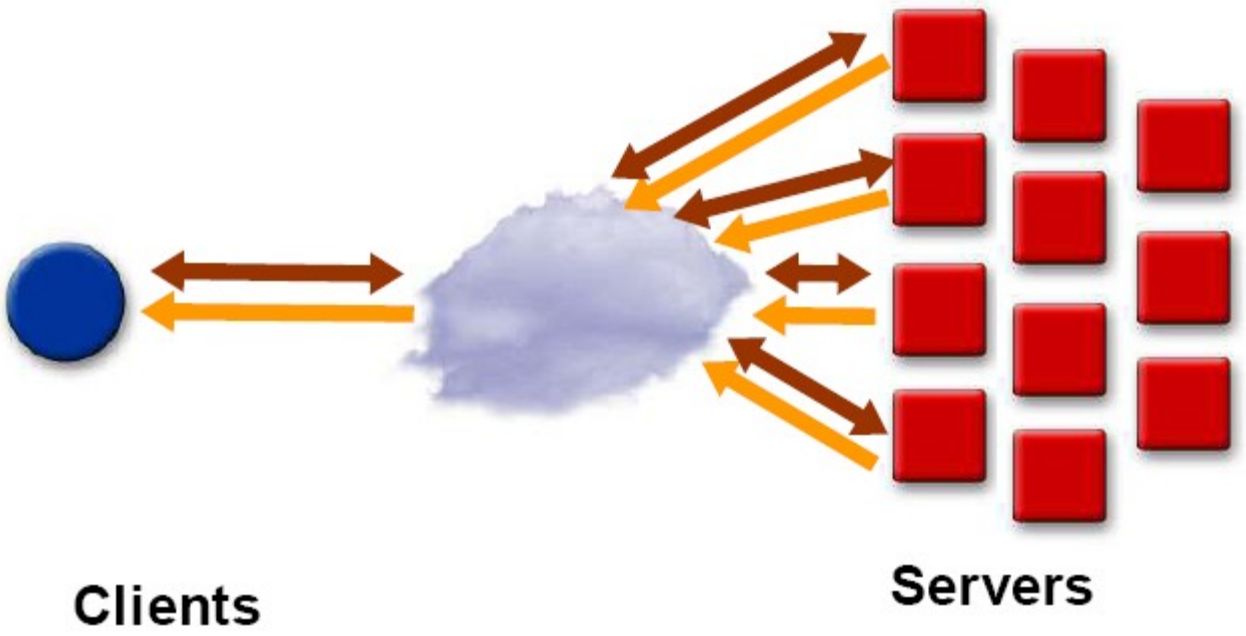
Servers

A Dynamic Server Farm



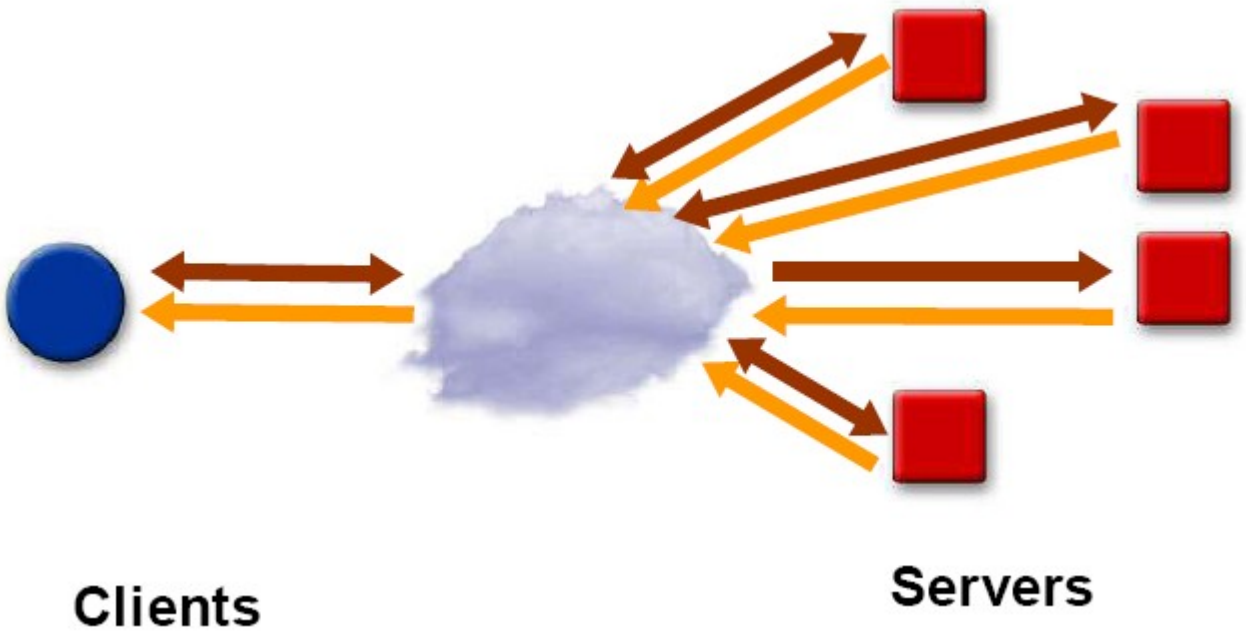
- Add more servers
  - Don’t need to tell client about new servers

A Dynamic Server Farm



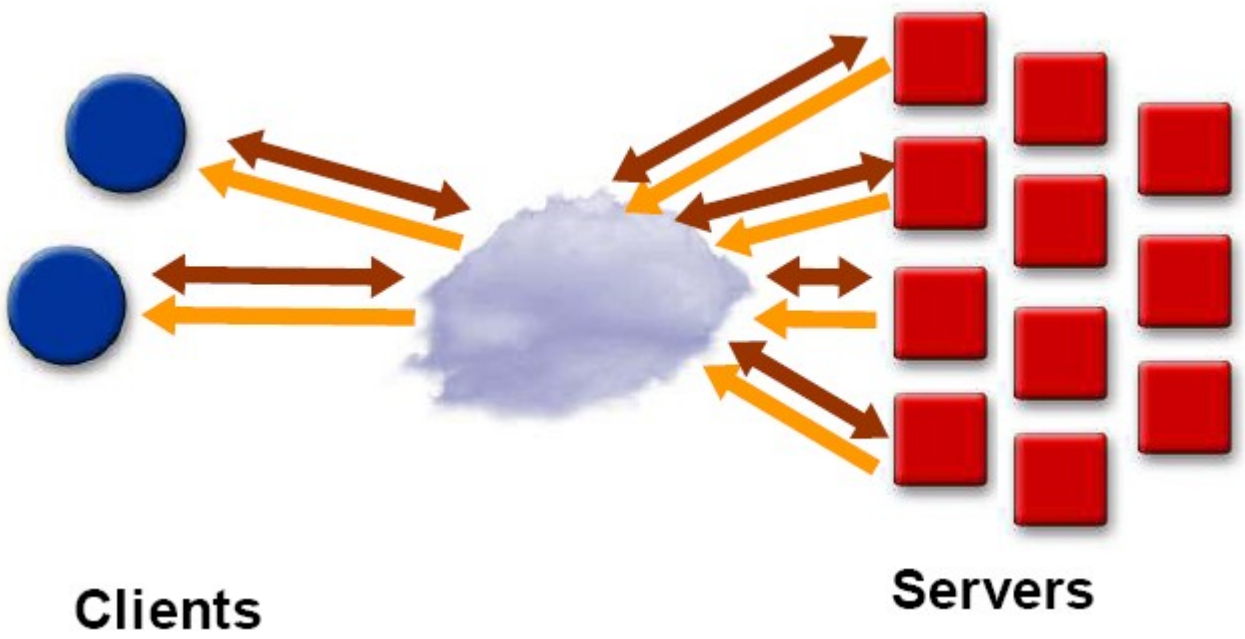
- Add more servers
  - Don’t need to tell client about new servers

A Dynamic Server Farm



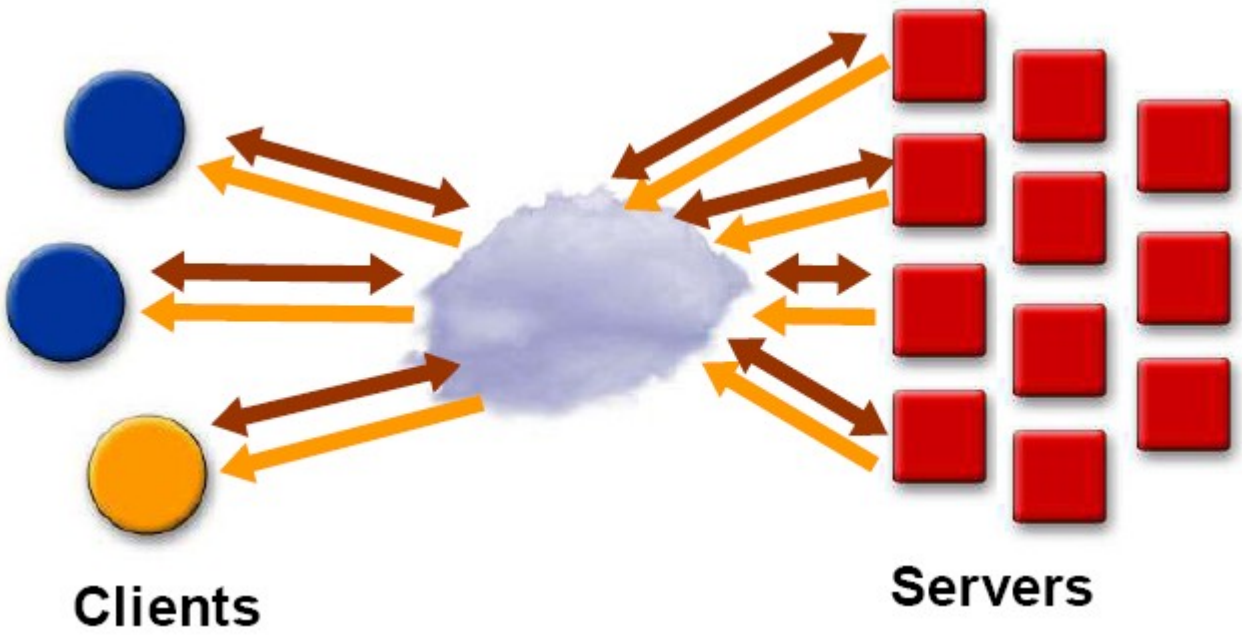
- Sometimes servers crash
  - No need to tell client about missing servers

A Dynamic Server Farm



- Add more animators
  - No need to tell servers about new clients

A Dynamic Server Farm



- We run other jobs
  - We can add new types of jobs without touching servers

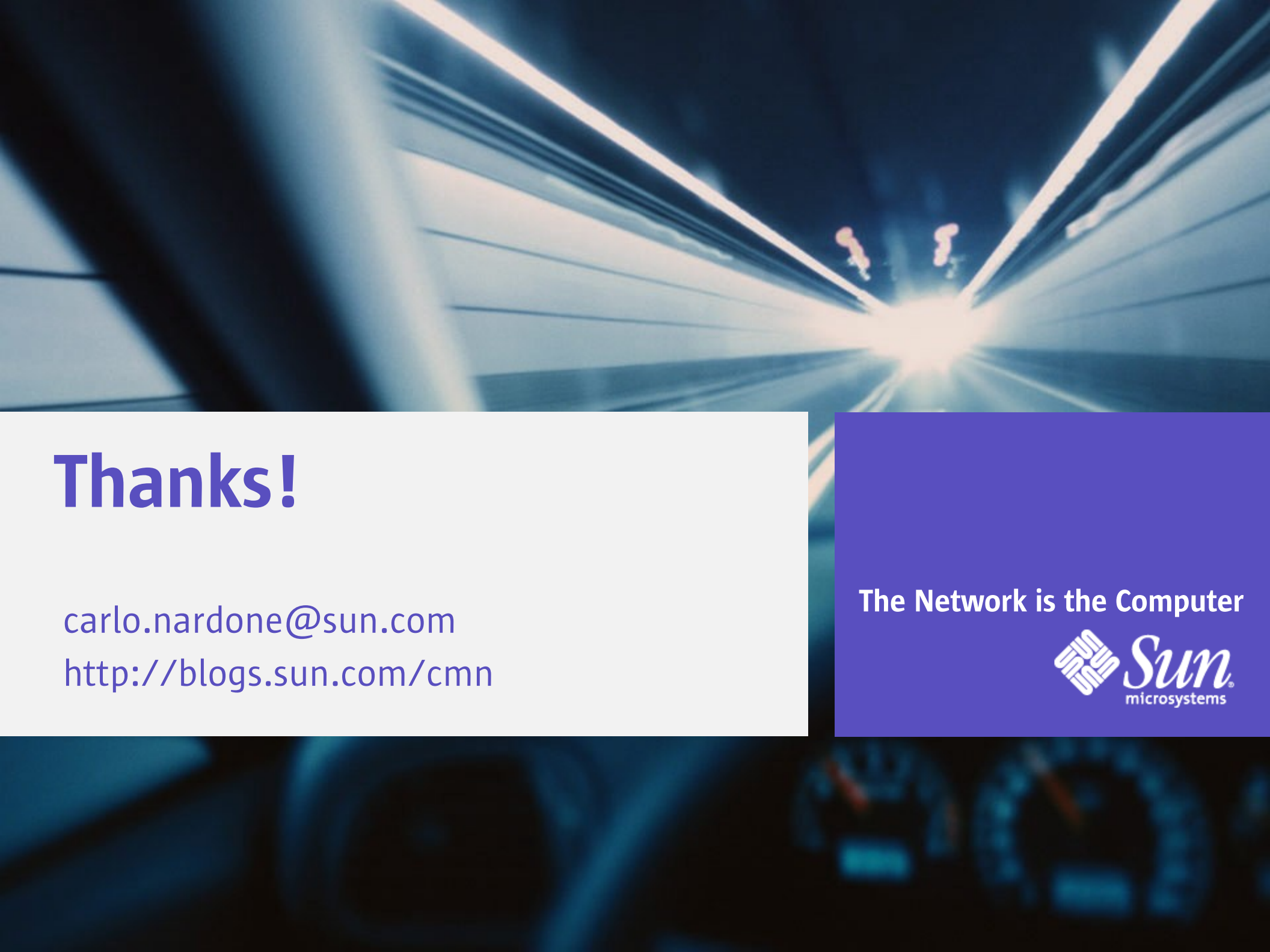
JavaSpaces Implementations

- Outrigger
  - Sun Microsystems' contributed implementation
  - Part of the starter kit
    - Includes source under SCSL
  - 10,000 to 100,000 entries
  - www.sun.com/jini
- GigaSpaces
  - Enterprise implementation
    - Clustering
    - Scalability
    - High Availability
    - Performance enhancements
    - Integration with web services, SOAP, WSDL, UDDI, JDBC
  - www.gigaspaces.com

JavaSpaces Real World Projects

- TeamVest
  - Developed online 401(K) investment site for Intuit
  - Uses spaces to run Monte Carlo simulations
    - Compute server
  - www.teamvest.com
- Cisco
  - Scalable Infrastructure (SI) communication framework
    - High availability
    - Agents
  - http://developer.jini.org:80/exchange/projects/si/

The Network Is the Computer



Thanks!

carlo.nardone@sun.com
http://blogs.sun.com/cmn

The Network is the Computer

Sun microsystems