



WRITE ONCE.
SCALE ANYWHERE.

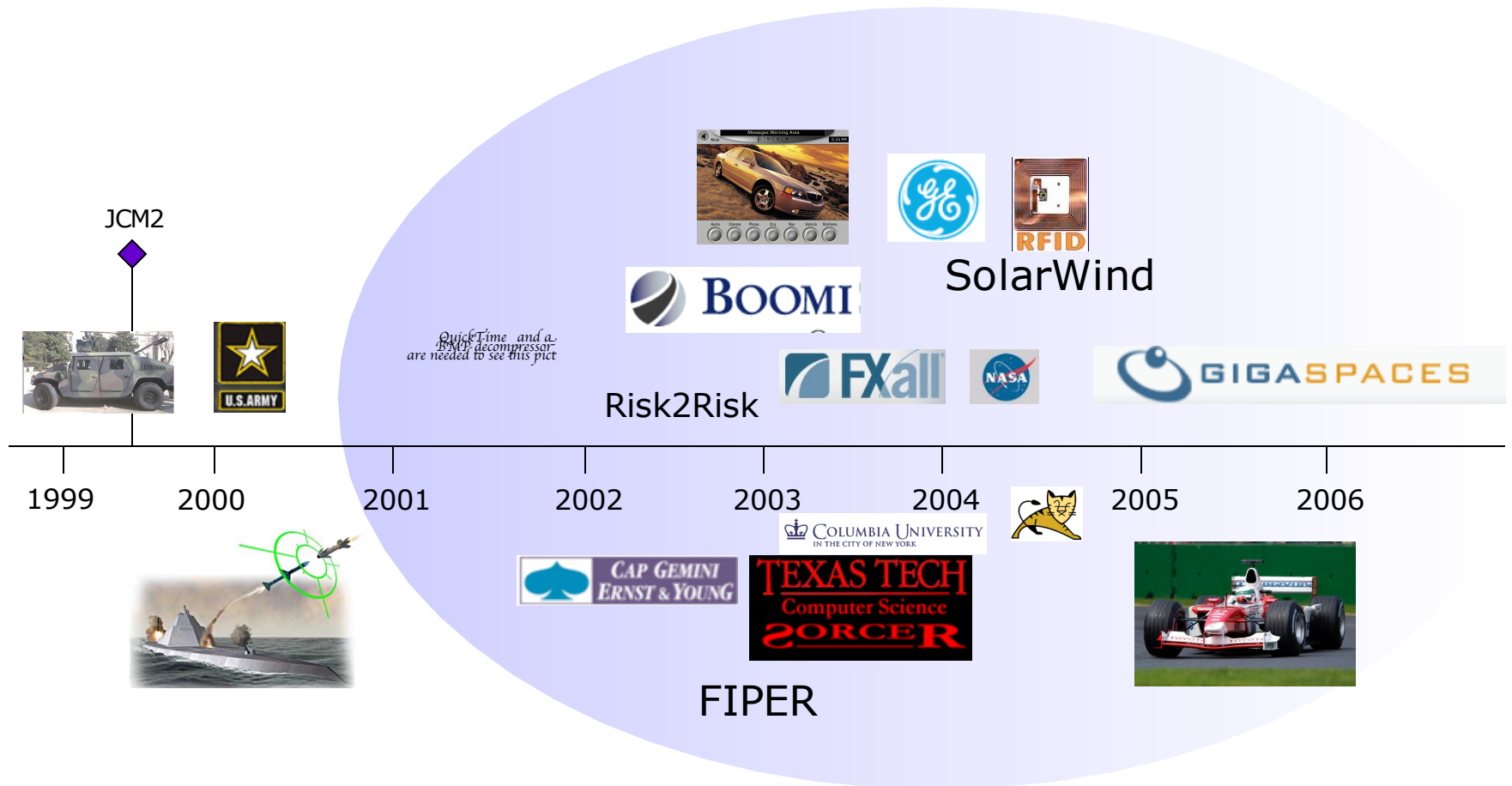
Project Rio Overview and Status



Dennis Reedy
GigaSpaces Technologies

The bottom section of the slide features a wide image. On the left, a photograph shows four people in an office setting: three are seated around a table, and one man in a pink shirt and dark trousers is standing and looking towards them. The office has large windows. On the right side of this section, there is a large, abstract graphic with blue and white geometric lines that create a sense of depth and perspective, resembling a stylized architectural structure or a data visualization.

History



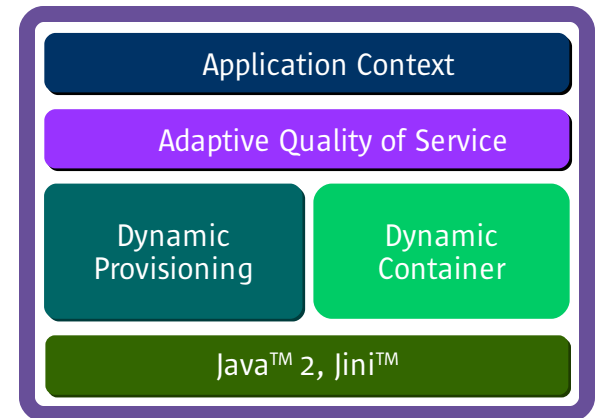
Common Threads

- Complexity
- Automation
- Self healing
- Scalable
- Dynamic
- Policy Driven
- Declarative




Rio

- Policy-Based infrastructure automating the deployment and execution of distributed applications
 - Measure Responses against Service Level Agreements
 - Dynamic execution fabric
 - Platform and application aware
- Providing ...
 - Deployment & management capabilities
 - POJO-based development approach
 - Fault detection and recovery
 - Model to inject rules & policies allowing greater automation, scalability and controlled behavior
 - Declarative service model
 - Ease of use



Status

- Owners
 - Me
 - Mark Hodapp 
- Moving to java.net
 - Classdepandjar
 - Rio
 - Rio Utilities
 - Rio Substrates
- Other related projects
 - Ipanema



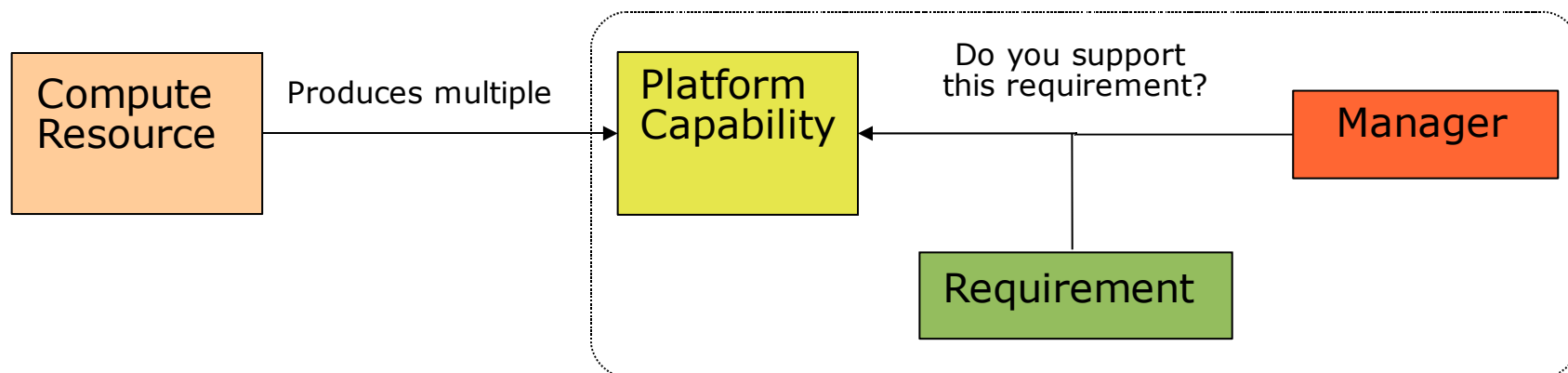
Policy Centricities

- Platform Policies
 - Whether a compute resource can support the requirements of a service
 - Installation, activation
- Behavioral Policies
 - Whether a service is operating to specified objective(s)
 - SLA Management, Service Associations, Dynamic system state
- Reachability Policies
 - Heuristics determining whether a service is available on the network

Platform Selection Policies

- Compute resources have capabilities
 - CPU, Disk, Connectivity, Operating Systems, Patch Levels, Software Components, Required Versions ...
- Software components need to run on the most appropriate compute resource based on definable criteria
- A qualitative capability indicates a specific type of mechanism or quality associated with a compute resource

Platform Capability Policy Interactions

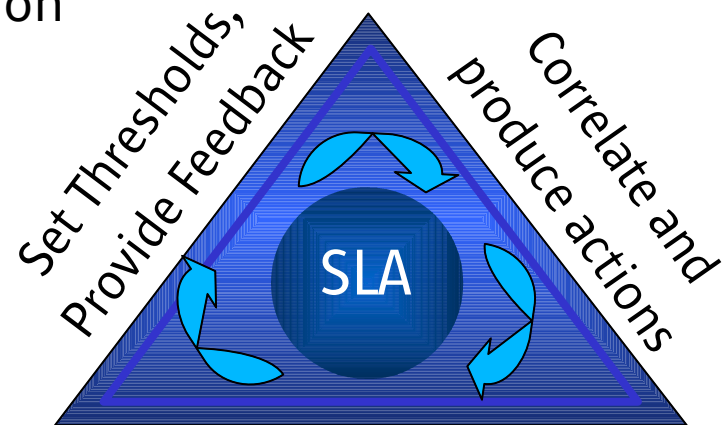


- ◆ Supportability determined by PlatformCapability objects, not by administrative configurations
- ◆ Selection policy enabled through mobile code strategy, enabling decentralized collection of platform capabilities
- ◆ Regular expression matching
- ◆ Allows for collection of compute resources with their platform capabilities to grow organically



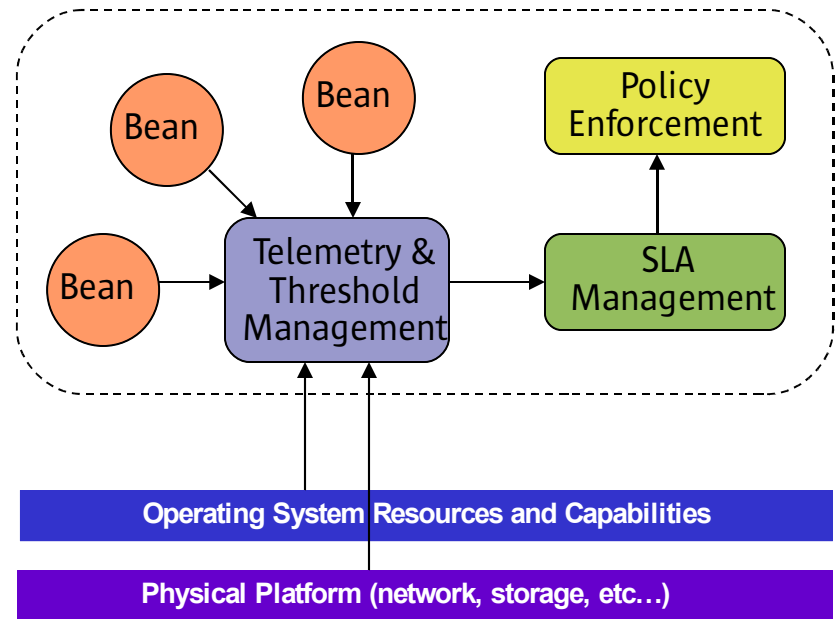
Behavioral Policies: SLA

- Compute resources have capabilities
 - CPU, Disk, Connectivity, Bandwidth, ...
- Software components need to run on most appropriate compute resource based on definable criteria
- Feedback mechanism to subscribe to changes to quantitative QoS mechanisms
- Provide a resource cost approach to measure compute resource capabilities in a heterogeneous environment



Autonomic SLA

- Sensor-effector pattern
- Data is observed from applications, OS, hardware, etc. and measured against declared thresholds
- Policy enforcement can happen locally, distributed or hierarchically
- SLA Threshold Events are fired to registered consumers
- Each SLA is Autonomic

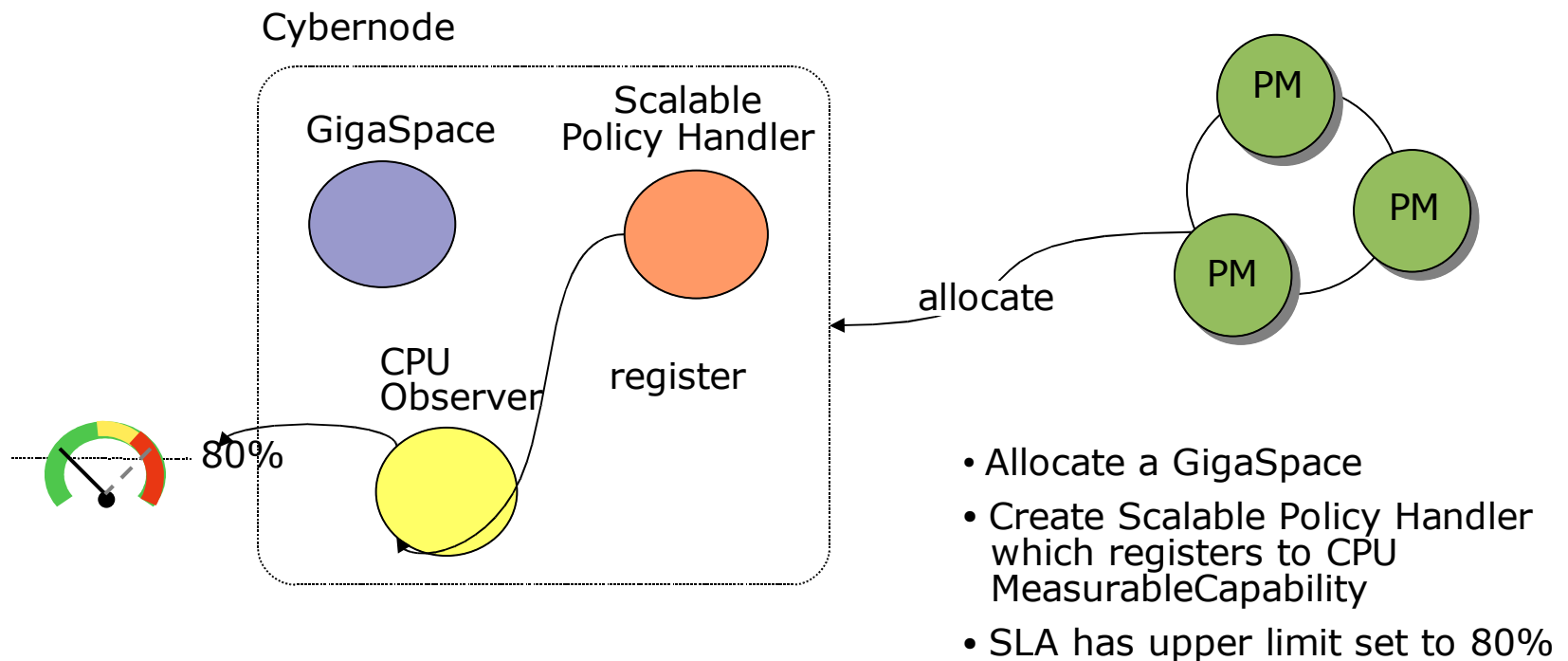


SLA Artifacts

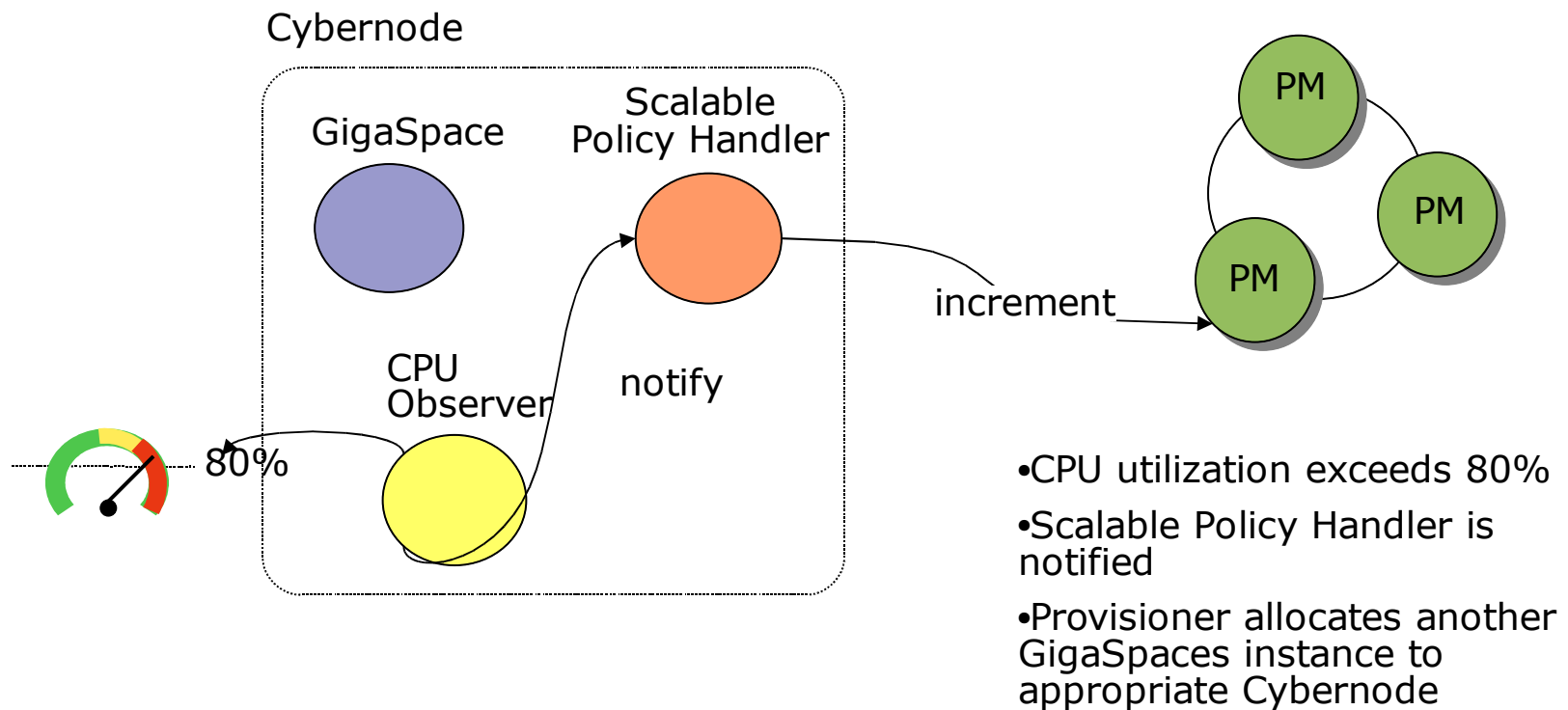
- XML Descriptor
- Can be imbedded in deployment descriptor
- Can be maintained and added at deployment time
- Can update an existing deployment
- Can be used as a basis when redeploying



Example: Service Scalability



Example: Service Scalability

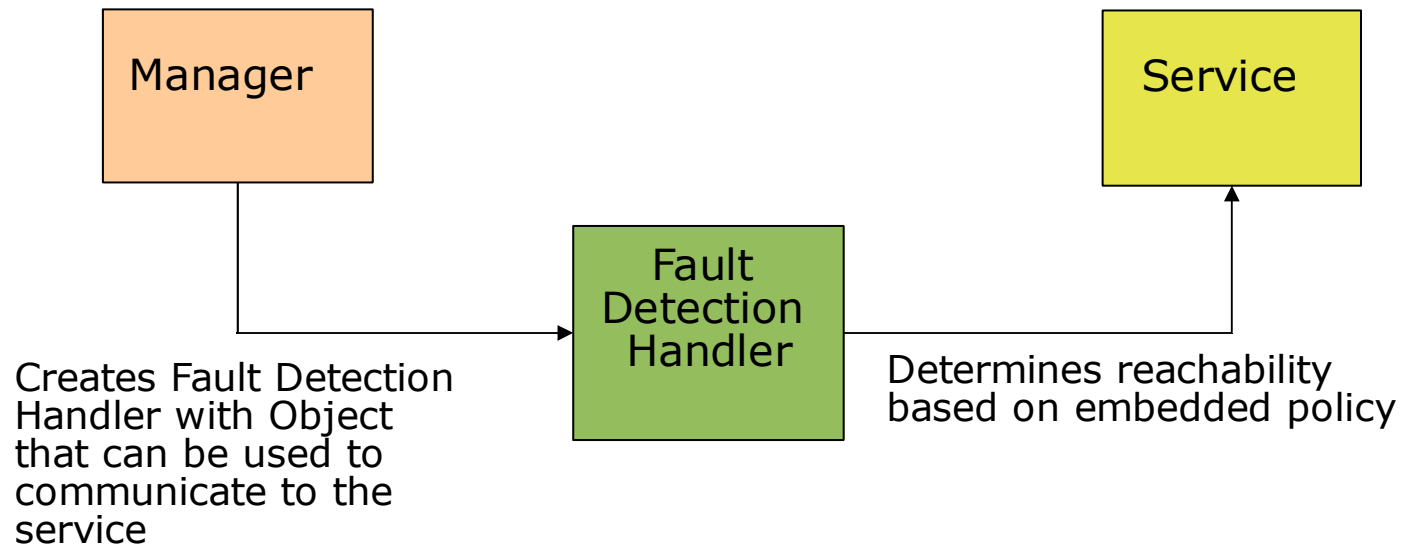


Reachability Policies : Fault Detection Handlers

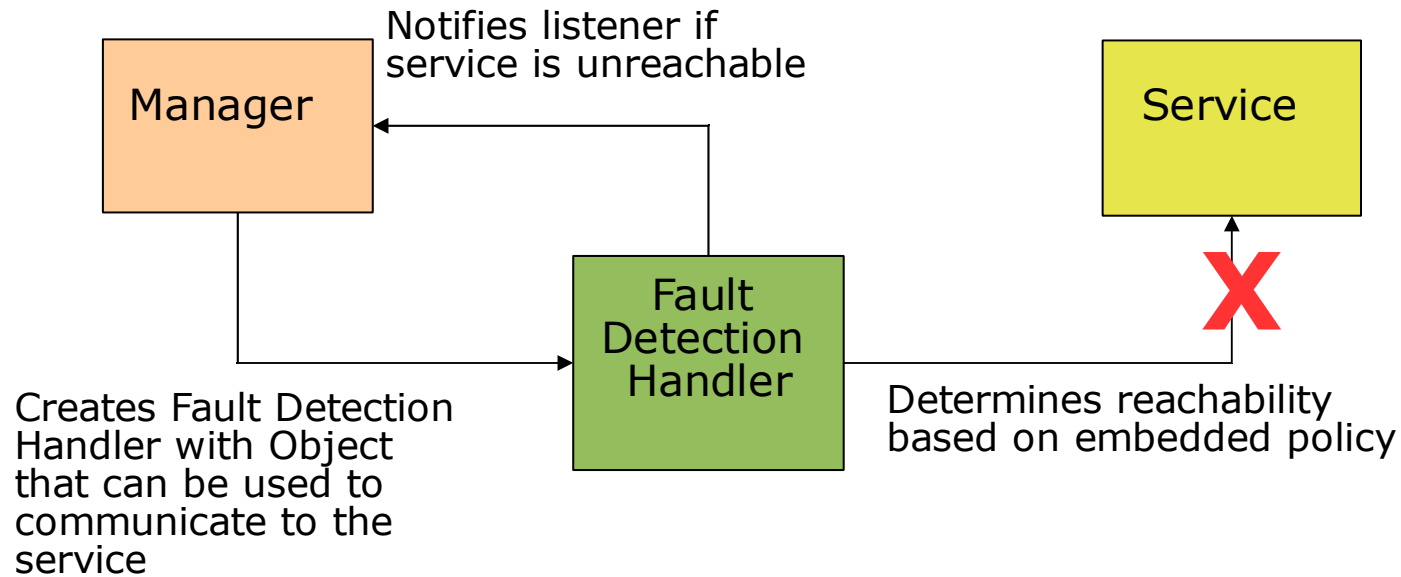
- A Fault Detection Handler is loaded by the client entities when a service is provisioned, has the service's proxy set, and periodically tests to make sure the service is reachable
- Can implement custom fault detection algorithms and protocols to determine service reachability
- Different heuristics for failure detection and recovery for different failure types
- Specify as part of the Deployment Descriptor on what heuristics to use (based on application objectives, system policy, ...)



Fault Detection Handlers



Fault Detection Handlers



Service Associations

- Provide the capability to declare a service usage model
- Associations can be either “uses” or “requires”
- IoC based
 - Ease of use
 - Infrastructure does the leg work, developers don’t need to

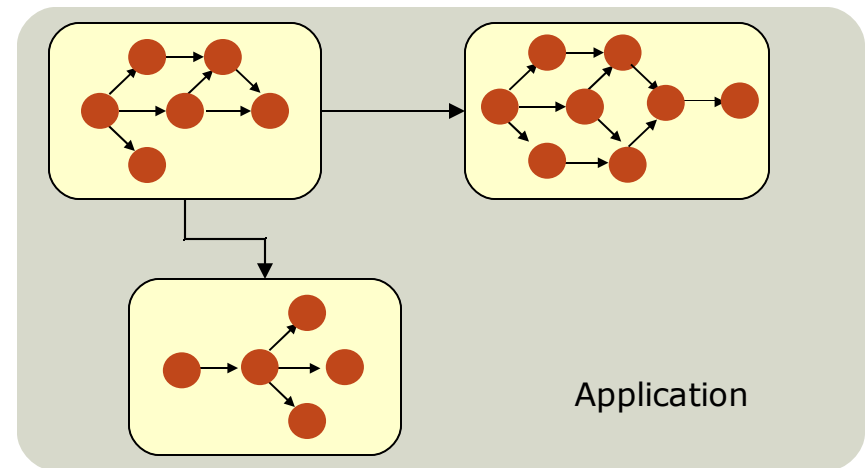
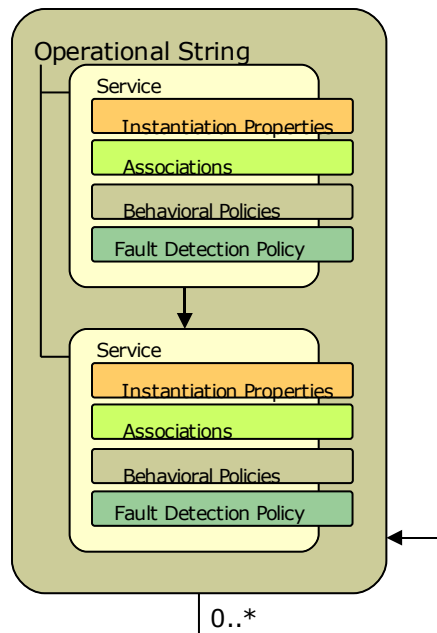
```
<association type="uses" name="The Space" property="theSpace" refid="some.ref"/>
```

```
<association type="uses" type="net.jini.space.JavaSpace"  
            name="The Space" property="theSpace"/>
```



Operational String

- An object graph composed of definitions that provide context on how to provision, manage, monitor and instantiate services
- Reflexive, may include other Operational Strings
- Created from an XML document, although there can be other source formats



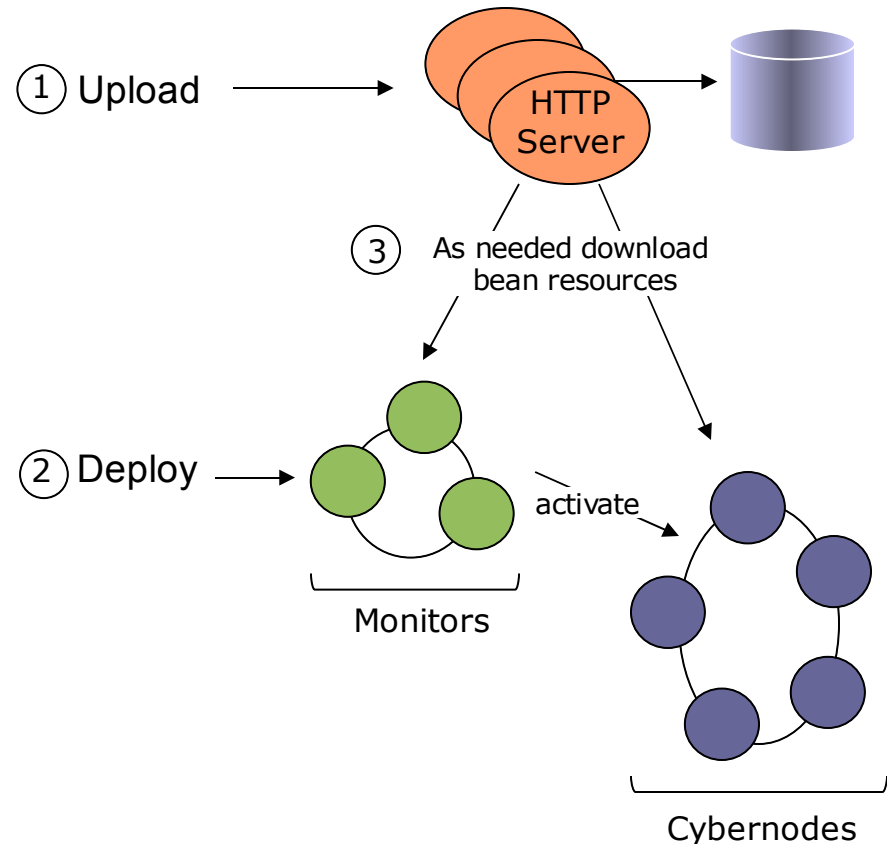
Application Deployment Semantics

- Deploy, Redeploy, Update, Undeploy
 - CLI & UI based
 - Public API specification
- Can be scheduled
 - Declare when to deploy, how long the deployment should last, how many times to repeat the deployment & the time to wait between deployments
- Additional semantics:
 - Relocate
 - Increment & Decrement

Deployment Models

- **Centralized**

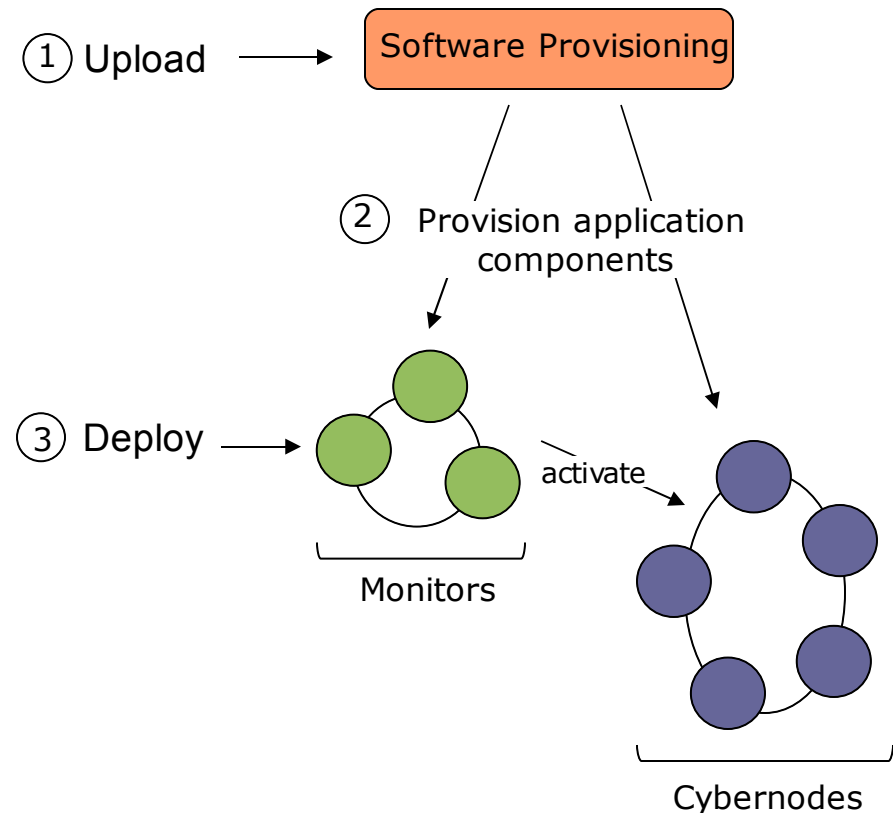
- Push to managed HTTP server (Code server) instances
- All code is dynamically served and instantiated
- Advantages
 - Manage one code repository
- Disadvantages
 - Application code availability if HTTP server instances fail



Deployment Models

• Distributed

- Provision code to all compute resources
- All code is loaded from the compute resource it is instantiated on
- Opportunity to leverage platform provisioning tools
- Advantages
 - Not rely on Code server availability
- Disadvantages
 - Version management
 - Integration



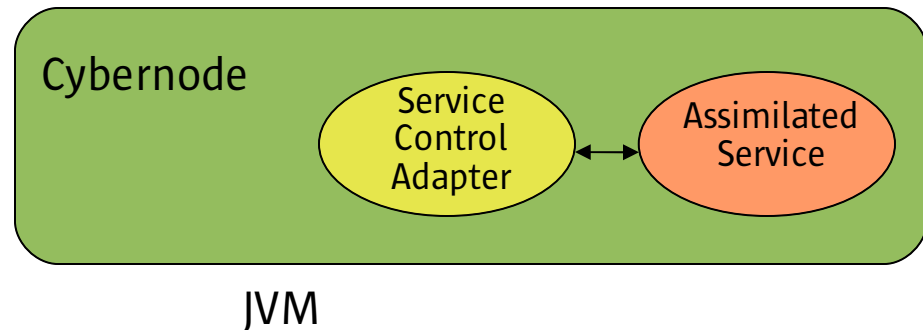
Service Development

- Dynamic services
 - POJOs
 - Spring supported deployment
 - Service Bean development
 - Simple component model defining lifecycle semantics of a dynamic service (start, init, advertise, unadvertise, stop, destroy)
 - Infrastructure provide an easy to use programming model whole maintaining access to lower level APIs
- Runtime provides
 - Fault Detection
 - Declarative SLAs
 - Association based Dependency injection

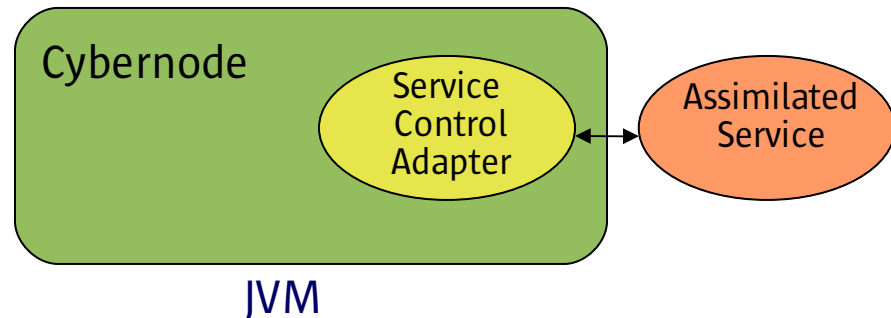
External Services

- What about external things?
- Service Control Adapter objects provide assimilation

Assimilated Service instantiated within Cybernode VM process address space

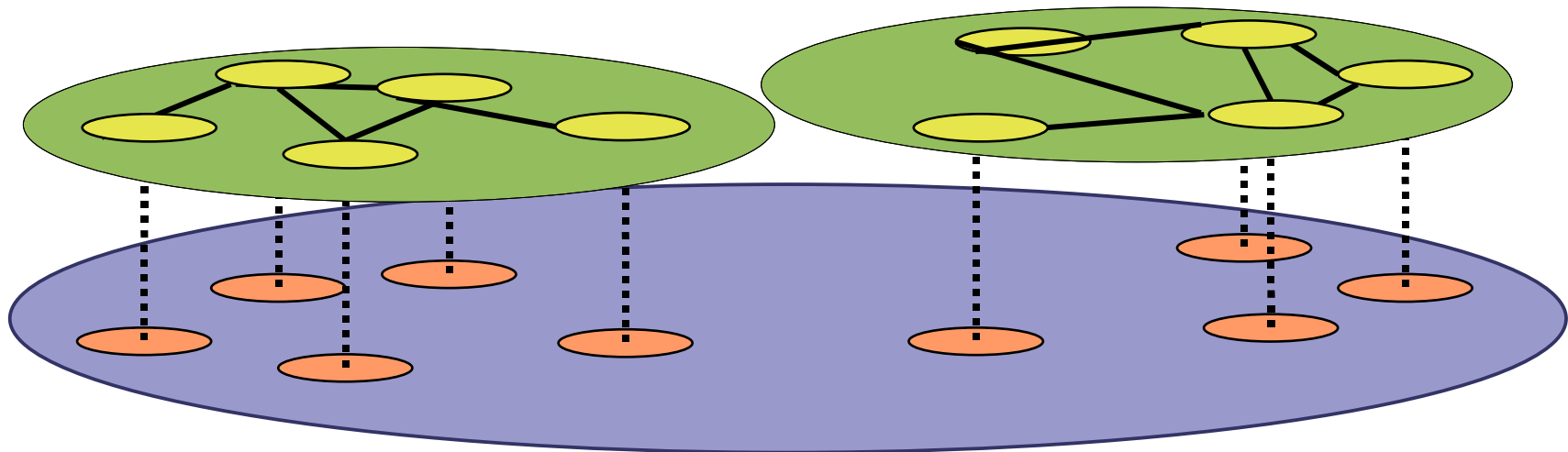


Assimilated Service instantiated in it's own process address space



External Services

- Encapsulate the control and monitoring of external services running within the context of Rio
- Service Control Adapters represent applications/services, adding network-wide visibility & control
- Attach monitoring, metering and SLA control to existing applications



That's Nice, Anything New?

- POJOs not just Service Beans
- Declarative Watches
- JMX

Whats Next?

- Release 3.3 4Q
- ... ?



*QuickTime and a
BMP decompressor
are needed to see this pictu*



Q&A