Experience Implementing Quality of Service for a Fault Tolerant Real Time Event Channel

Sylvester Fernandez

Joseph Cross*

Lockheed Martin Tactical Systems

July, 2003

* Joe Cross has since taken up a new position at DARPA

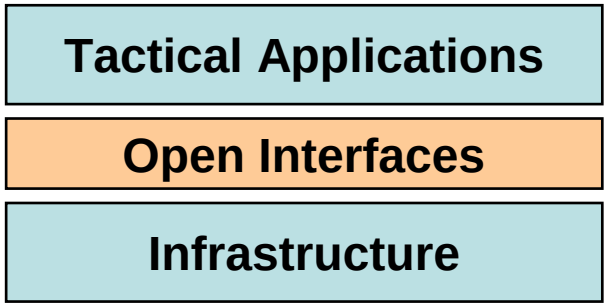
Military Use of Commercial Technology

Often cited goals for military systems

- Compatibility and interoperability across systems
- Reduced life-cycle costs
- Faster, easier system upgrades
- Reduced COTS refresh costs

How do we get there

- Avoid point-solutions
- Use open standards
- Adopt common operating environments



Essential Tensions

COTS responds to commercial market forces, which are different from the forces that operate in military environments

Military Applications

- Long Lived, typically 15 to 20 years or longer
- Needs prioritized access to resources
- Requires predictable behavior
- Evolves based on what the military needs

Standard Interfaces

Not always fully specified, leaving vendors with reasons to 'extend' the standard to provide needed capability

- This locks the system to proprietary products in spite of conformance to open standards
- Commercial technology typically evolves faster than the standards to which they conform

COTS Infrastructure

- Short Lived, typically 6 to 18 months
- Optimized for good average behavior
- Adaptable, converging over time to desired behavior
- Evolves based on what the customer will buy

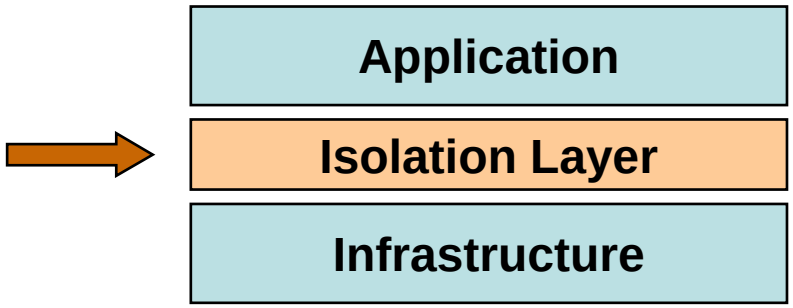
Research Focus

Problem

Standards-conforming middleware alone may not be adequate for military systems with reliability and time-critical performance requirements

- Middleware standards only codify functional behavior
- Qualities of Service (QoS) concerns such as performance, reliability, and security are not controlled by standards
- Vendors are free to provide different QoS while claiming conformance to standards

Our efforts are focused on how best to achieve true infrastructure isolation



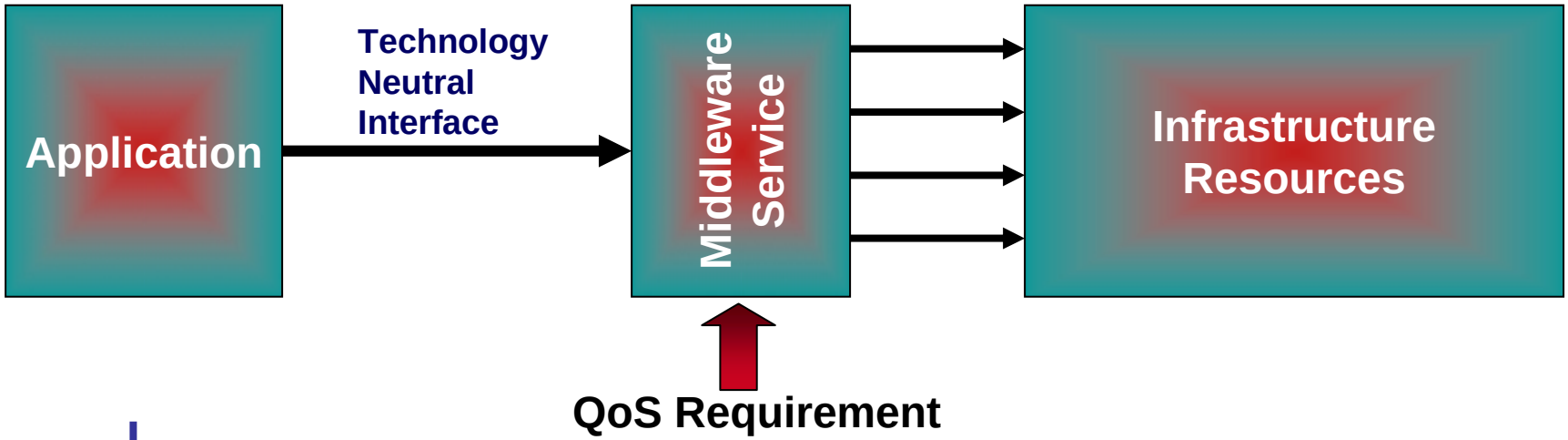
Proposed Solution – QoS Enabled Middleware

To achieve true isolation between applications and the infrastructure, you must be able to specify and obtain QoS in technology-neutral terms across standard functional interfaces

- Absent this degree of isolation, systems built on “open” COTS infrastructure will continue to exhibit the symptoms of point solutions
  - Even for systems that may be able to meet its QoS requirements using current technology, awareness of QoS requirements at the middleware interface is a necessary design consideration for infrastructure isolation and hence the smooth evolution of the system

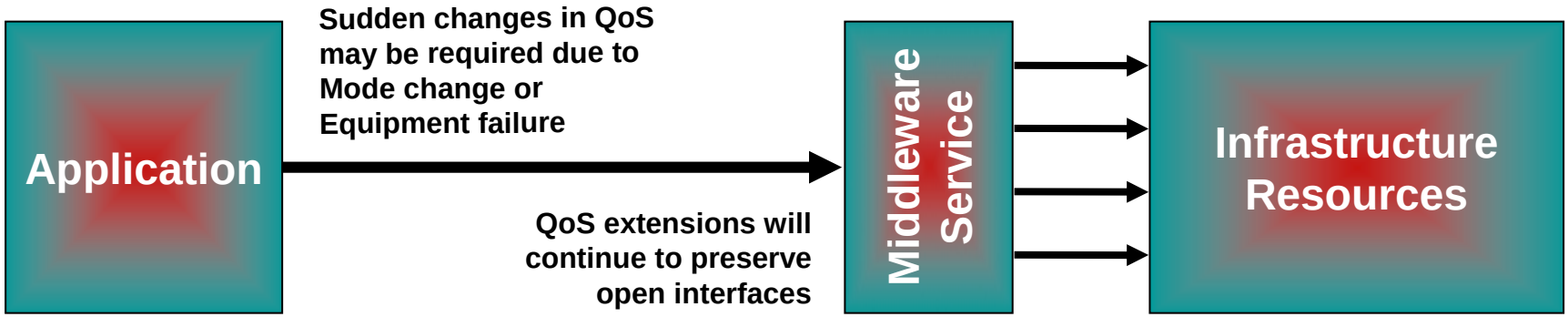
The Basic Approach

- Applications use standard interfaces to specify **what** they want done
- *In addition* they also specify **how well** it must be done, but in *technology-neutral* terms
  - This can be specified either at service access points, or as system-level policy
- Middleware maps the requested service to the appropriate resources to meet the specified QoS
  - QoS can be varied based on system state



Additional Requirements & Constraints

- Distributed architecture – no centralized control
- No access to internals of service implementation
- No access to internals of resources
- Compatible with model-based development processes (e.g., MDA)



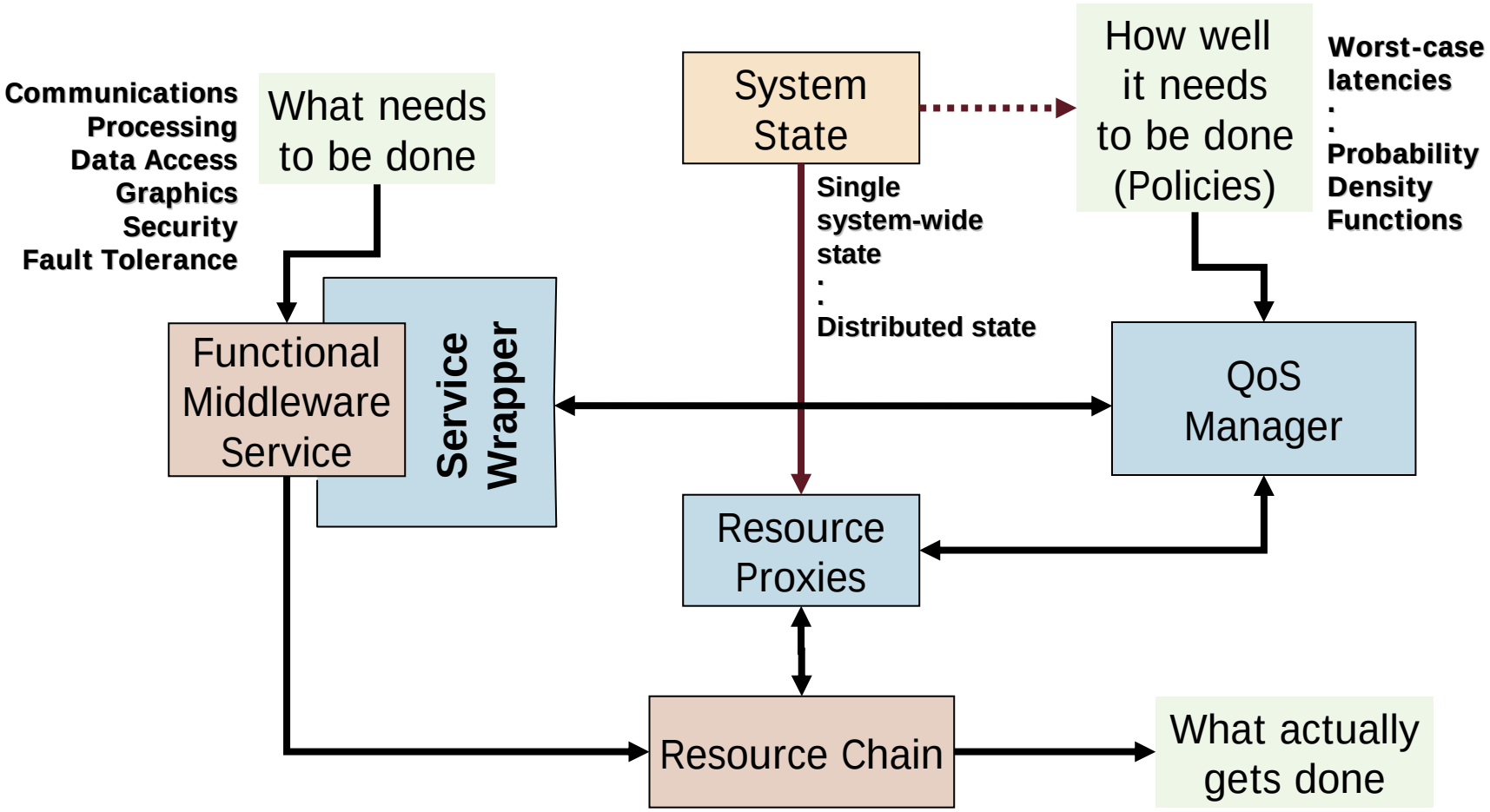
Contracts

- QoS is specified, negotiated and obtained through ‘contracts’
- Contracts are tied to system state and apply to service access points
- The default contract is ‘best effort’

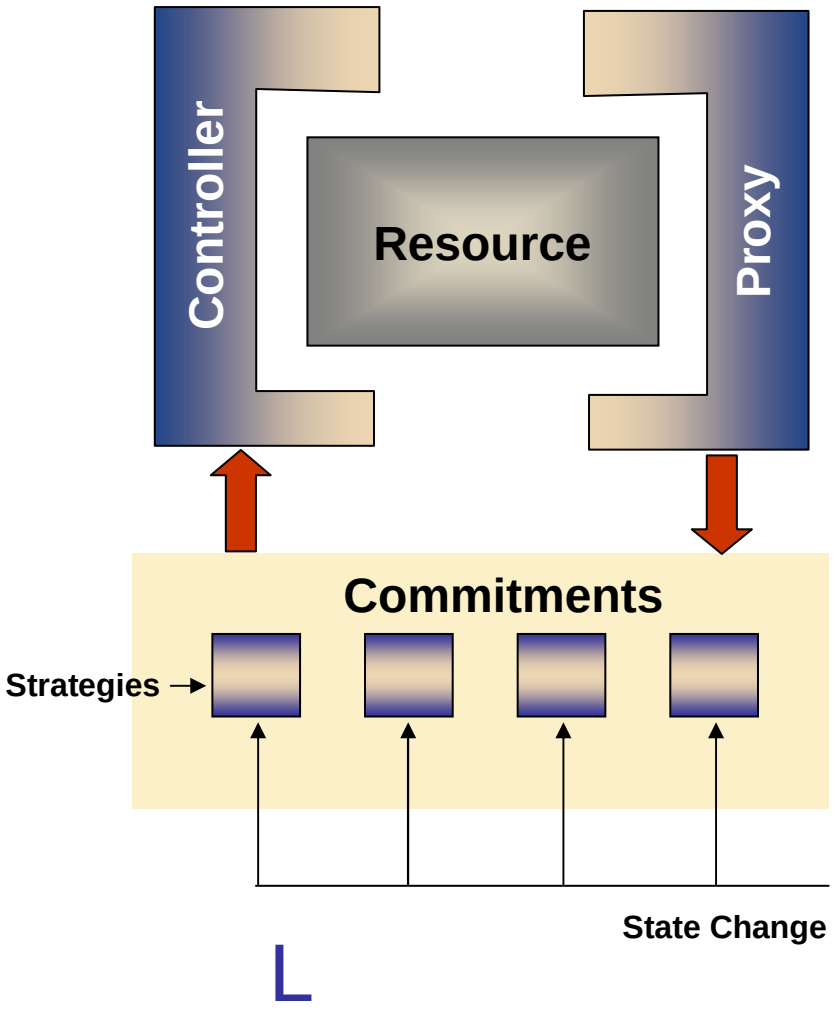
Contract negotiations can occur at any time for any condition

- Most negotiations are expected to occur at initialization
- Negotiations may consume a large amount of computing resources

QoS-Enabled Middleware



Resource Allocation



- A contract may be satisfied by a set of resources
  - Derived from configuration topology
  - Based on ‘reachable’ nodes that match service resource requirements pattern
  - May be ordered based on optimum search path for a given service
- Proxies speak for the resource in all matters related to contract negotiation
  - Remembers commitments already made
  - Generates strategies
- Strategies specify how the resource must be managed to achieve the required QoS
  - May be merged with other strategies in same mode
  - Listens for changes in state
- Controller
  - Executes strategy
- Resources can contain other resources

Distributed State – An Example

Resource allocations can vary depending on the state of each subsystem

Local resource allocations can be based on state of remote subsystems

Resource requirements can established a priori, based on possible states

Hidden

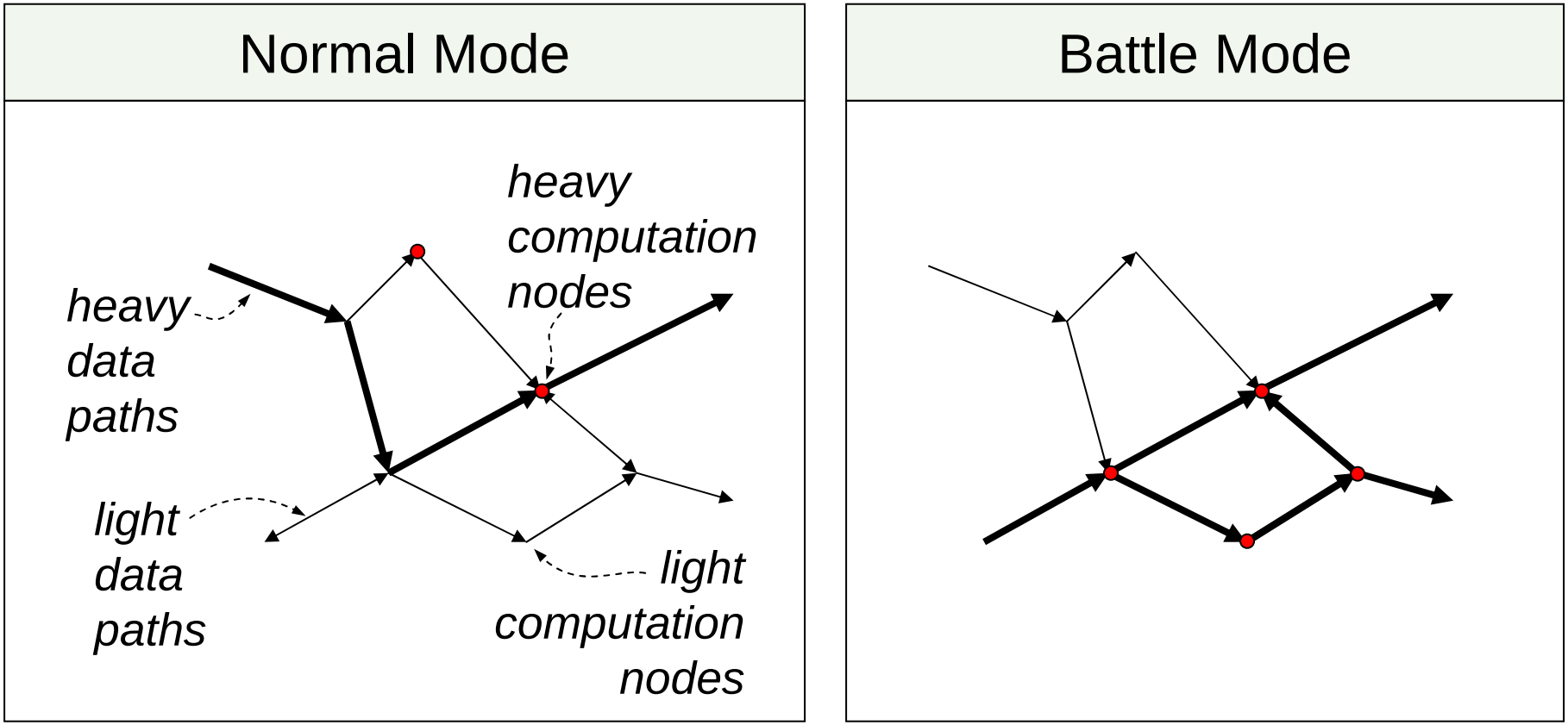
Routine Patrol

High Alert

Monitoring

Briefing

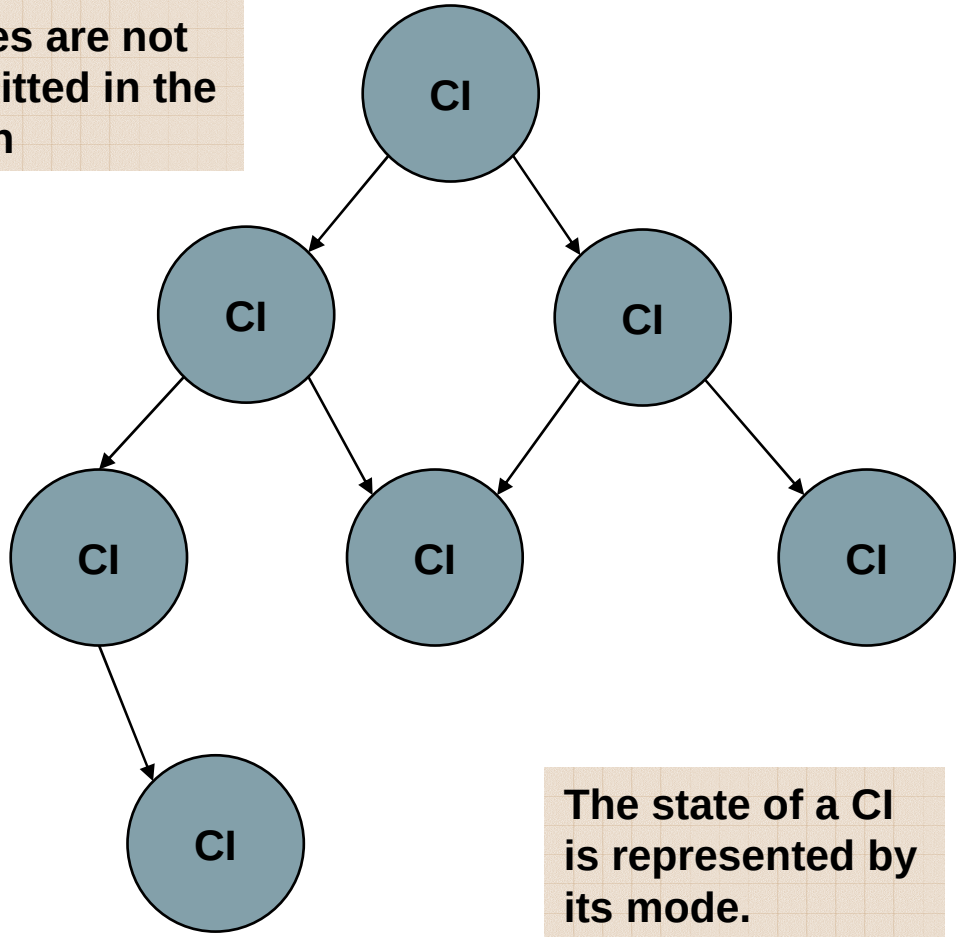
The Need to Change Allocations



And: Do It Quickly

Configuration Items

Cycles are not permitted in the graph



The state of a CI is represented by its mode.

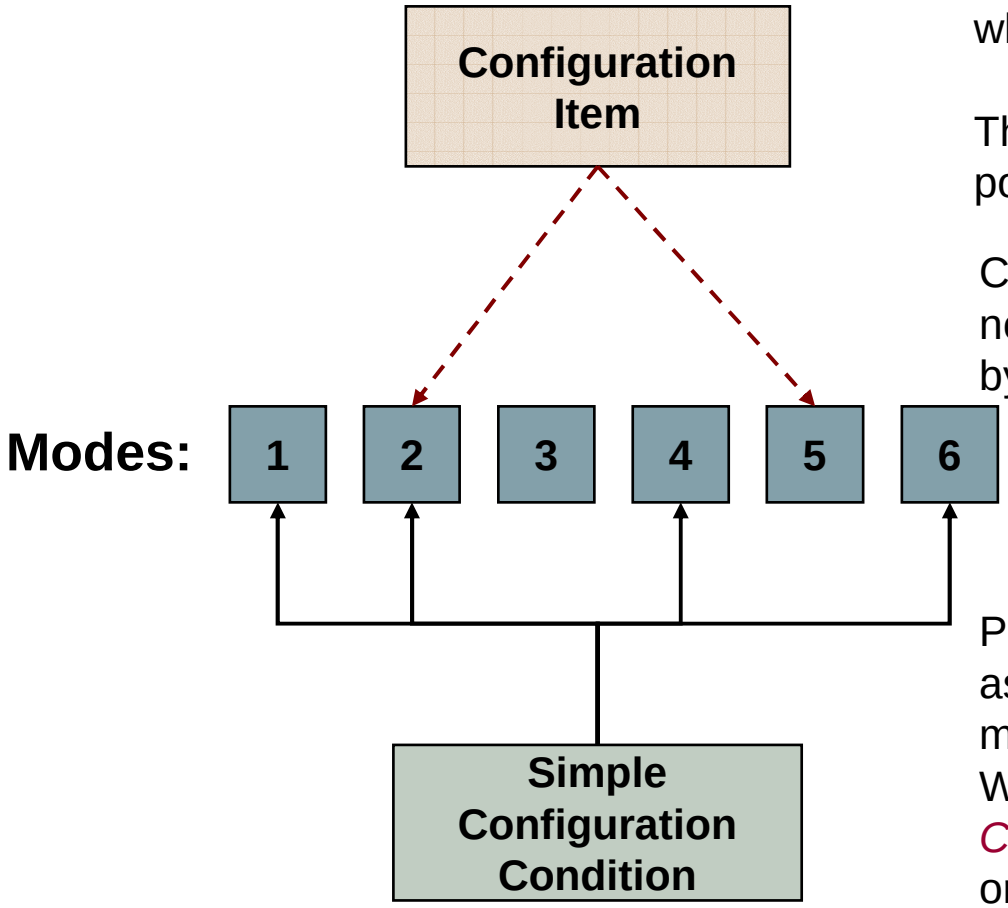
A configuration item can contain another configuration item.

A configuration item can be contained in another configuration item.

A configuration item cannot contain itself, either directly or indirectly.

Configuration Items are not limited to just static entities; they may include dynamic components

Modes and Configuration Conditions



Every CI has a set of **modes** (*1 .. n*), which can be named

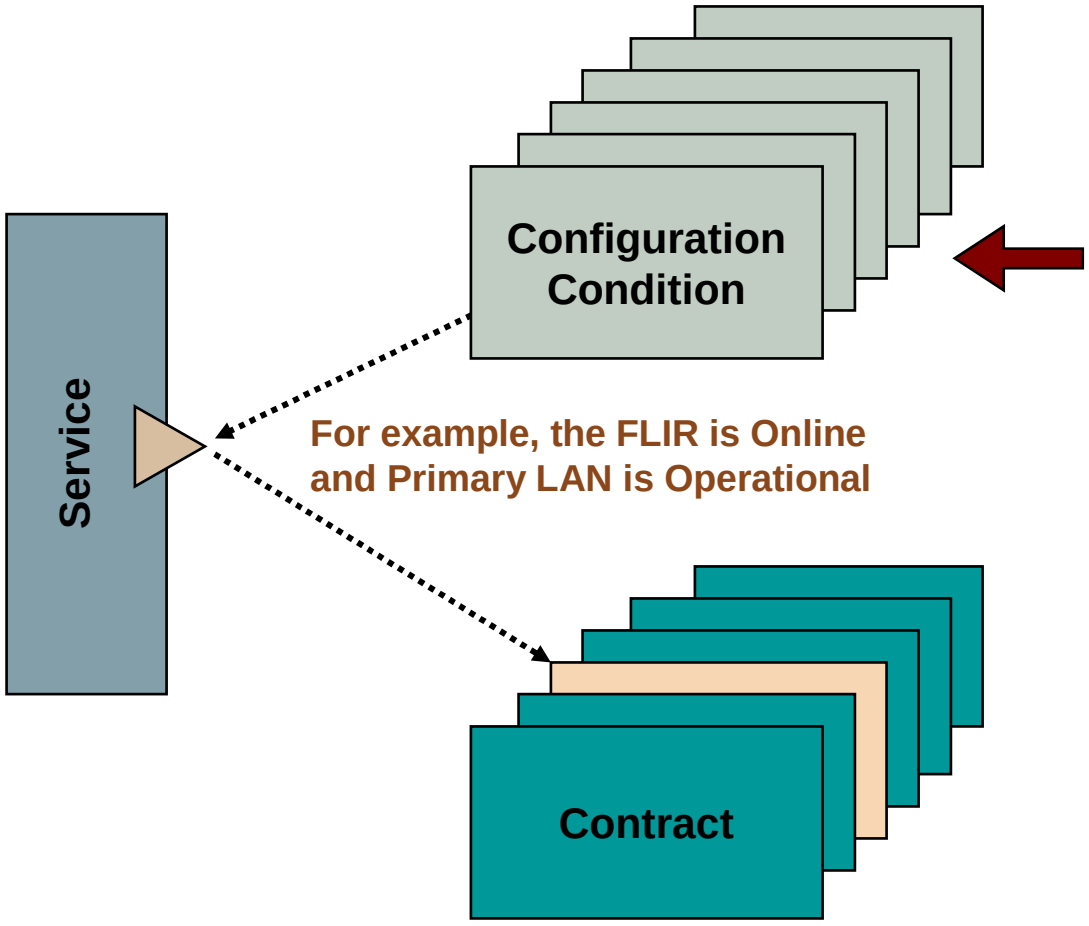
The **current mode** of a CI is one of the possible modes that it can be in

Clients can register to receive notification of mode changes by specifying:

- A pre-condition
- A post-condition
- A notification point

Pre and post-conditions are specified as boolean expressions on a CI's modes.
We refer to such conditions as simple *Configuration Conditions*, *e.g. (mode 1* or *mode 2 or mode 4 or mode 6).*

Configuration Conditions and Contracts



Boolean expressions whose terms are Configuration Conditions: (*CC*$_1$ and *CC*$_2$ or *CC*$_3$ and not *CC*$_4$).

Can be established during initialization

Evaluated when notification events occur

Determines which contract is selected

Default contract is "best effort"

Contract Specification Examples

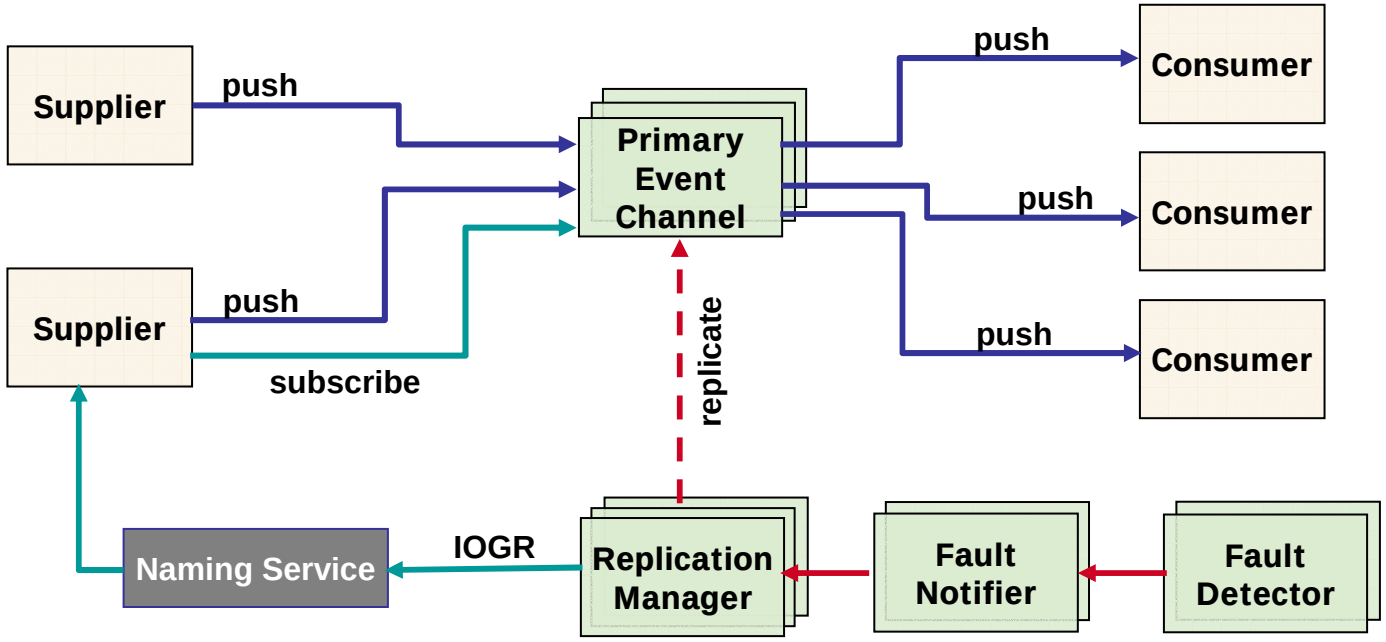
Using Worst Case Values

```
<ec_id id="Tracking_Service">
 <contract>
  <mode>battle</mode>
  <latency_ms>100</latency_ms>
  <reliability>0.99</reliability>
  <msg_size_bits>1000</msg_size_bits>
  <arrival_rate_mps>5</arrival_rate_mps>
  <clients>
    <location>
     <id>Trident</id>
     <kind>hostname</kind>
    </location>
    <client_type>Supplier</client_type>
    <location>
     <id>Viking</id>
     <kind>hostname</kind>
    </location>
    <client_type>Consumer</client_type>
    <location>
     <id>Excalibur</id>
     <kind>hostname</kind>
    </location>
   <client_type>Consumer</client_type>
  </clients>
 </contract>
</ec_id>
```

Using Density Intervals

```
<proposal>
  <mode>
    <or>
      <ci name="radioVHF" state="onLine"/>
      <ci name="radioUHF" state="onLine"/>
    </or>
  </mode>
  <QoS type="latency">
    <upperPoint secs="1.0" prob="0.99"/>
    <upperPoint secs="4.0" prob="0.9999"/>
  </QoS>
  <load type="interMessageTime">
    <upperPoint secs="1.0" prob="0.0001"/>
    <lowerPoint secs="1.0" prob="0.9999"/>
  </load>
  <load type="messageSize">
    <upperPoint secs="256" prob="1.0"/>
    <upperPoint secs="32" prob="0.5"/>
  </load>
 <load type="priority">
    <urgency val="10"/>
    <importance val="2"/>
  </load>
</proposal>
```

Fault Tolerant Real Time Event Channel



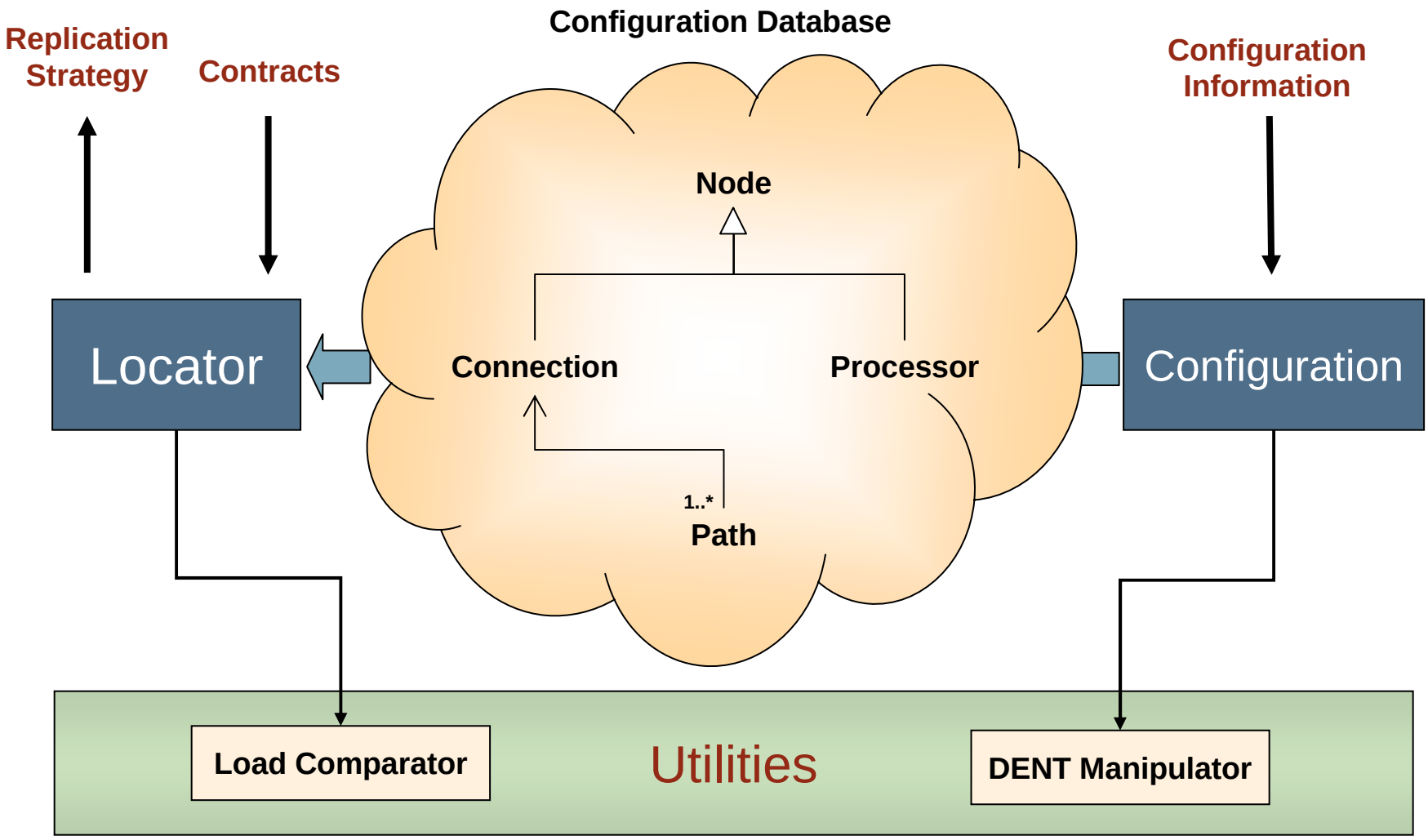
- Provides configurable robustness of event streams in the face of fail-stop faults, within real-time constraints
- Offers useful configuration knobs to Quality Connectors
  - Replicas: where and how many
  - Transactional replication depths for event subscriptions

- Tunable transaction depth used during a subscription
- Trades RT blocking time for FT assurance of replication
- One-way “soft replication” past assured transaction depth
- Event-Channel provides a messaging Façade
- Hides multiplicity of replicas and interfaces
- Reduces complexity of object references

Quality Connector (QC)

- Initial Implementation
  - Develops replication strategies that satisfy latency and reliability for
    - Event Channels
    - Replication Manager (RM)
    - Fault Notifier (FN)
    - Fault Detector (FD)
  - Assumes single fault tolerance domain, with one logical RM, FN and global FD per domain
  - One contract per event channel (contracts currently not tied to state)
  - Separate IDL interfaces for
    - QoS contracts
    - Configuration Information (processors and connections may only be removed)
    - Replication strategies (location of components, with placeholder for configuration options)
  - Contracts and configuration can also be input from XML files

Fault Tolerant Quality Connector



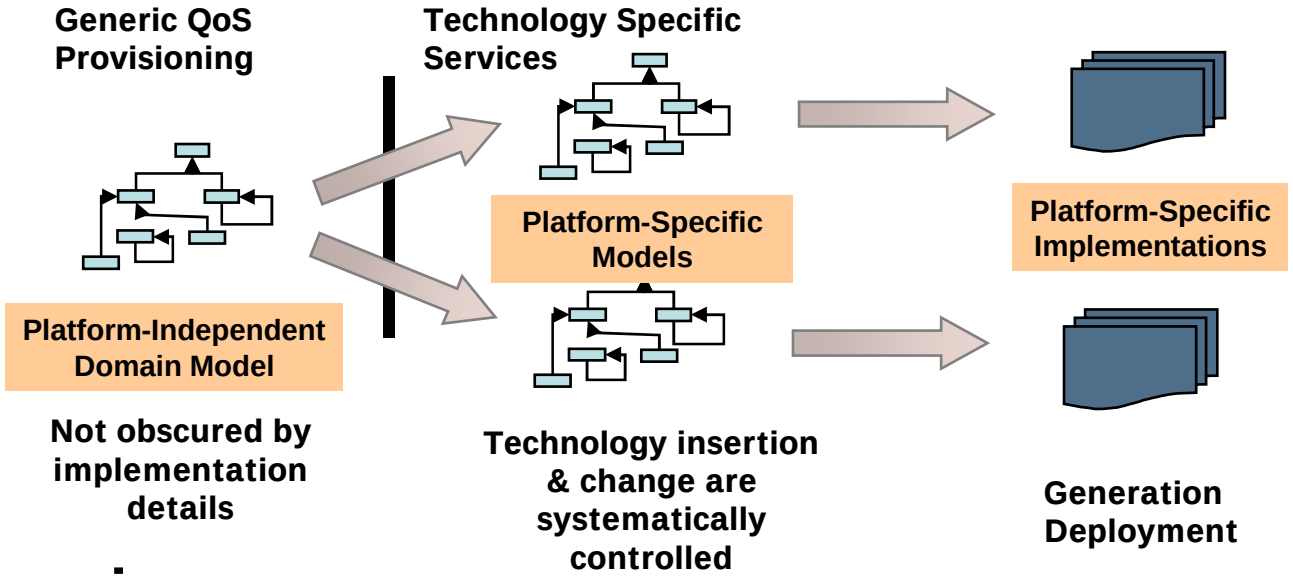
L

QC Implementation Details

- Contracts contain
  - Requested worst case latency from any supplier to all consumers on an EC
    - Measured from the time the event is pushed to the time the last consumer's push() is invoked
  - Requested worst case reliability, specified as the probability that a message will arrive at all its destinations within its latency bound
  - Message size (initially fixed)
  - Arrival rate (initially periodic)
  - Client mapping showing location of each consumer and supplier on the EC
- Configuration information
  - Consists of topological configuration of the system in terms of processing nodes and connections between them
  - Each processor and connection has the following attributes:
    - Incurred latency specified as a Density Interval (DENT)
    - Availability, specified as the probability that a processor or connection is available for use (note that messages that arrive late are considered failures for purposes of calculating reliability)
    - Capacity, specified as the maximum load that the processor or connection can handle. (The initial version assumes that connections are FIFO, and that all available capacity can be used by the EC)

MDA Investigations

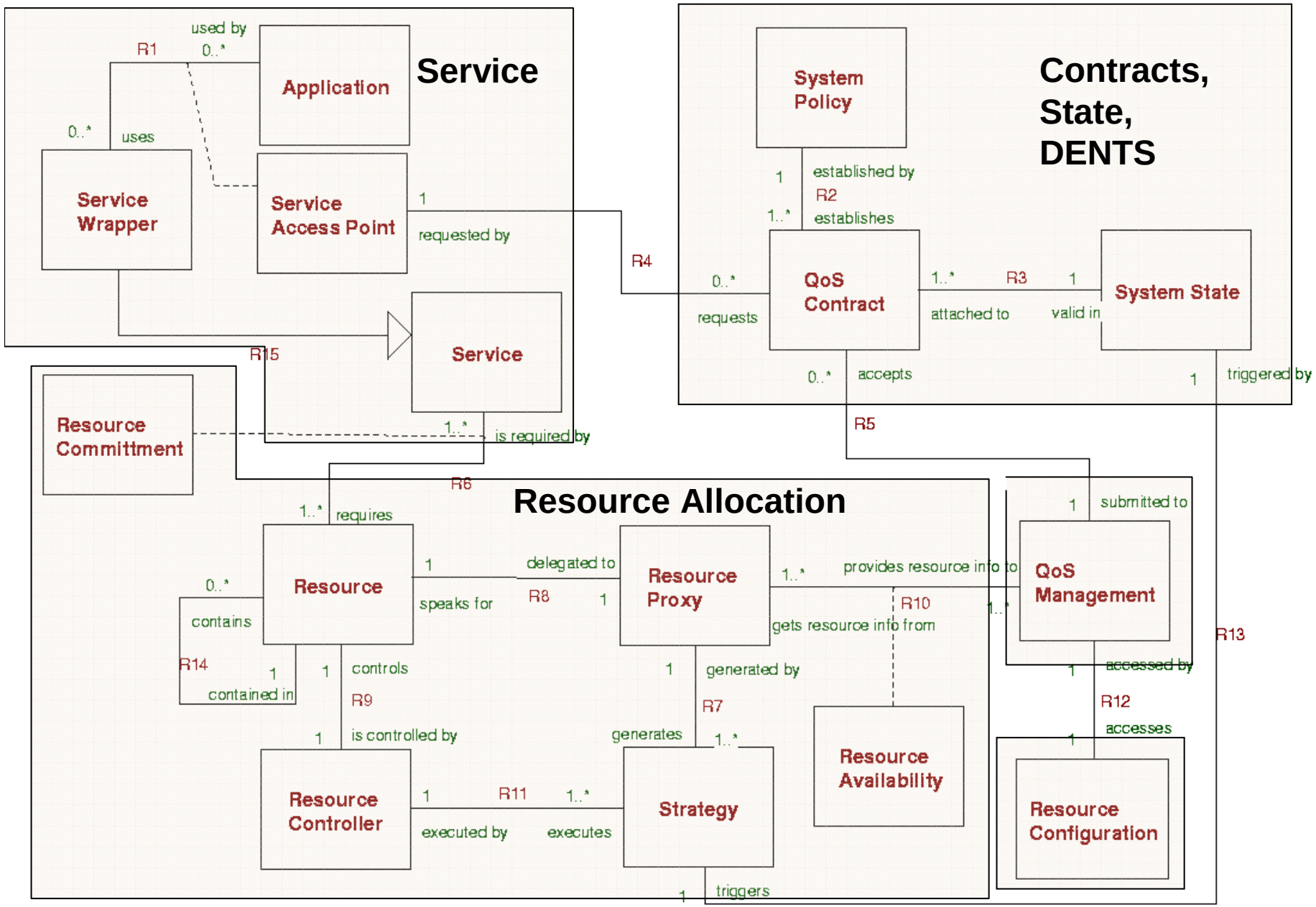
- **Challenge problem – Quality Enabled Services**
  - **Develop a PIM to capture generic pattern of QoS-enabled middleware services**
  - **Derive model for QoS aware asynchronous messaging service (e.g., publish/subscribe)**
  - **Demonstrate translation to multiple target implementations (e.g., CORBA Event Channel, reliable multicast)**



Open issues

- **Use of UML for PIM**
- **Suitability of action specification languages for time-critical applications**

QoS Enabled Middleware v1



Backup

Tactical Computing Trends

- Historically, tactical computing solutions have tended to point solutions
  - Applications have traditionally relied on particular features of the hardware, OS software and interconnect technologies
  - It has been the only way to guarantee performance and reliability
- Such systems are expensive to build and hard to maintain
  - Pervasive dependencies on proprietary hardware and software means changes are difficult to make and to verify
  - Systems built this way tended to be fragile and error-prone
- Meanwhile, rapid and dramatic evolution of commercial technology suggested that migrating to COTS might provide significant cost and performance improvements for military systems
- COTS Technology shows up primarily in the computing infrastructure
  - Processors, operating systems, network components, communication software, etc.
  - Domain specific components (sensors, weapons, etc.) continue to be custom built
- Competition in the open market caused proliferation of infrastructure options
  - Interoperability and reuse were the first casualties (consider use of i960s on F-22)
- We have since come to rely on open standards to promote interoperability and reuse across infrastructures

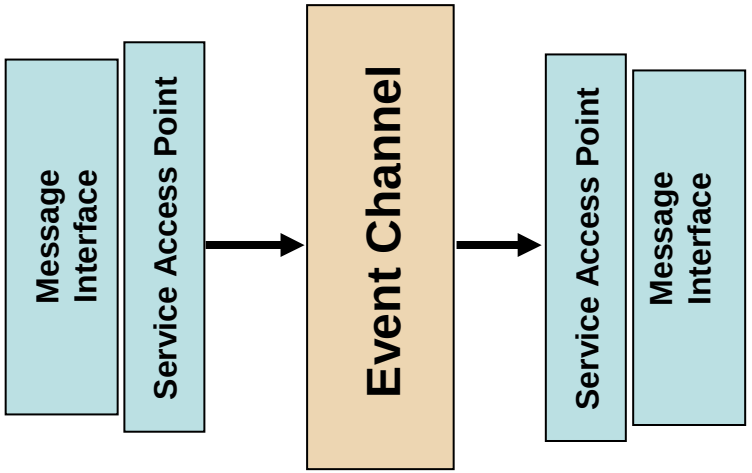
Benefits of QoS-Enabled Middleware

- Application and infrastructure can now cycle at different rates
  - Interfaces between the two are technology neutral
  - Resource allocations are managed by the service provider
    - Service provider now has better control over total life-cycle costs
  - Obsolescence problems become more manageable
    - Systems can stay current with commercial technology at much lower technical and cost risk
- Systems no longer constrained to point solutions. We can more easily support
  - Dynamic configurations
  - Distributed architectures
  - Better sharing of resources, higher utilization
- When combined with program generation technology, the middleware becomes easier to configure and use
  - More fluid configurations allow for rapid changes in mission, information flow patterns, sensor and weapons configurations – an essential enabler for network-centric architectures
  - Full life-cycle maintenance can be carried out on just the domain model
    - Technology refresh is confined to the middleware and infrastructure layers
    - Recertification can be done at the component level, rather than the system level – cheaper, faster, less error-prone.

Configuration Database

- Topologies: maintains ‘reachability’ information on resource nodes
- Resource patterns: resource types needed to support a service
- Search patterns: optimal search order through a set of resources in order to meet needs of a specific service
- Location: mapping of resources to other resources
- Paths: sequence of nodes that represent dependency graphs
- Sample node attributes
  - Capacity
  - Availability
  - Delay

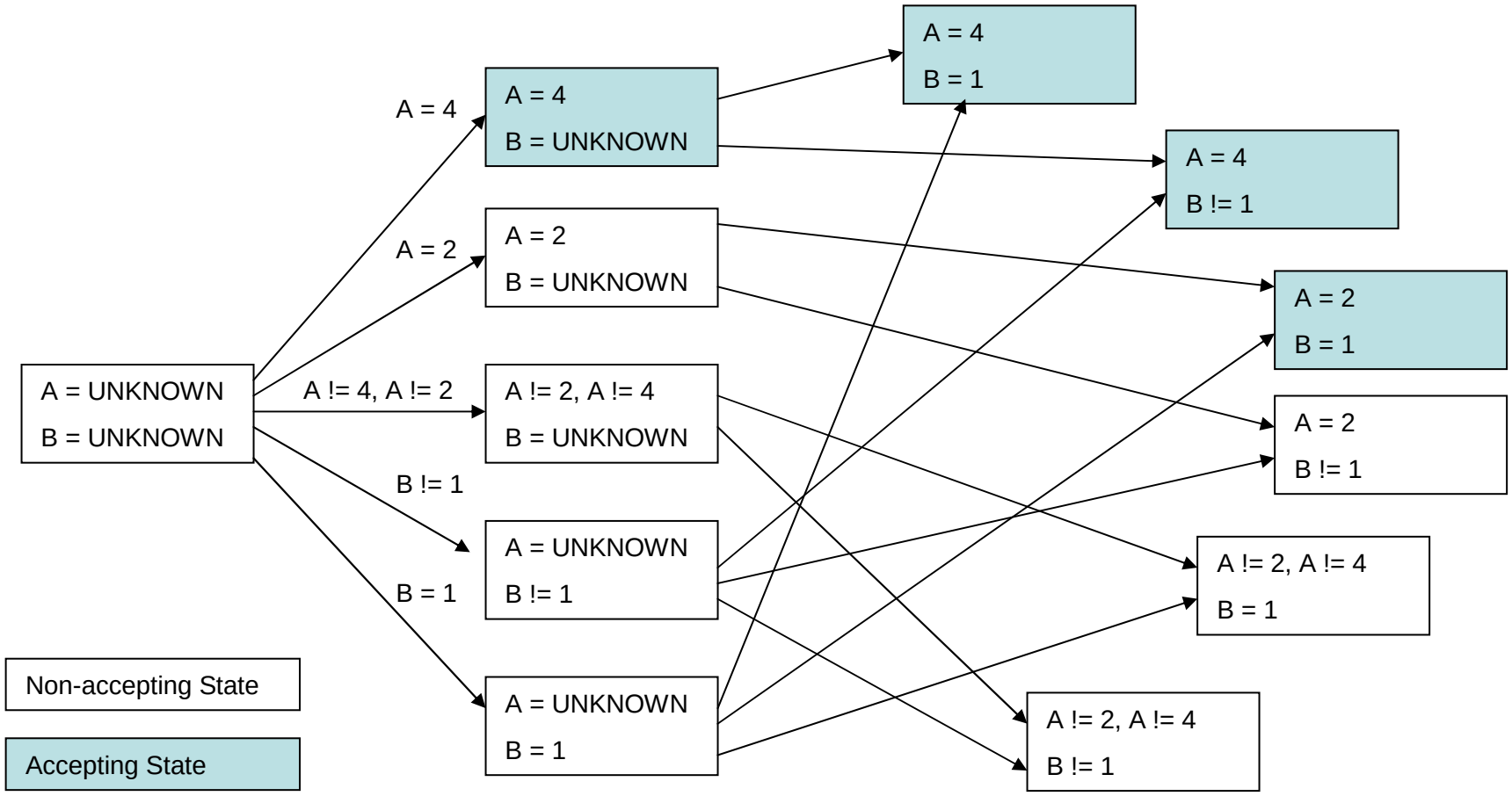
QoS on Event Channel



- Message Interface
  - Used by application to send or receive messages of a given type
  - Service is Publish/Subscribe
- Service Access Point (SAP)
  - Hides implementation of service using CORBA Event Channel
  - SAP is treated as a resource to which QoS can be applied
- RT Event Channel
  - Event Channel is also treated as a resource to which QoS can be applied

Finite State Machine Example

A EQ 4 OR (B EQ 1 AND A EQ 2)



L