



HIGH PERFORMANCE DISTRIBUTED COMPUTING

Leslie Madden & Paul Werme
NAVSEA Dahlgren Division

HiPer-D PROGRAM

DARPA GOAL:

Transition Computing
Technology to Military



HiPer-D Premise:

New Computer Program
& System Architecture
Required to Fully Exploit
COTS Technology



AEGIS GOAL:

Eliminate Capacity &
Scalability Bottlenecks



Technology & architecture




DARPA Technologies

- Advanced computers
- Operating systems
- Advanced networks
- Low latency protocols
- Quality-of-service middleware
- Resource management

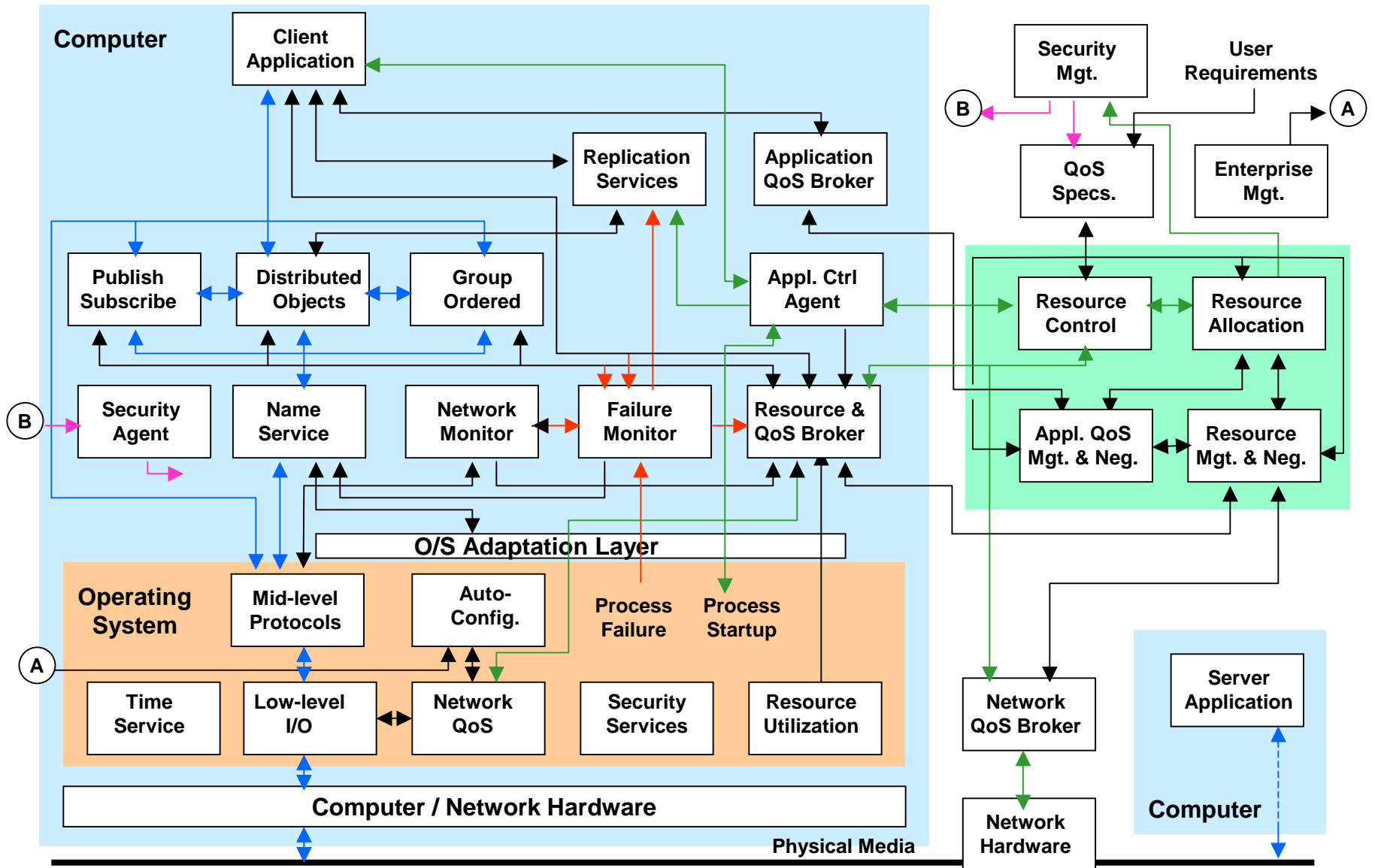
Architecture Concepts

- Distributed processing
- Open systems
- Portability
- Scalability
- Fault tolerance
- Shared resource mgt.
- Self-instrumented

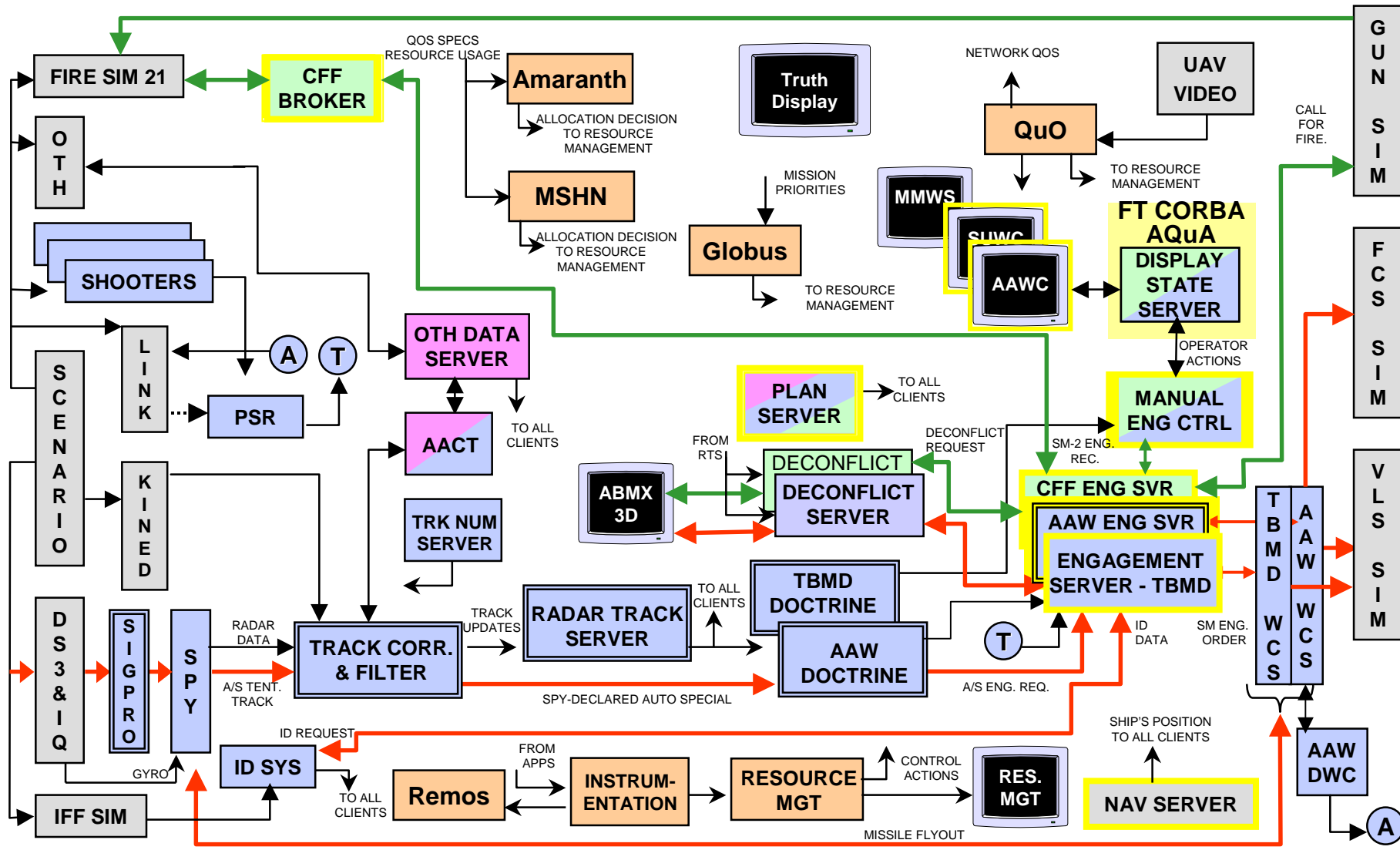
Navy Benefits

- Load-invariant tactical performance
- Information access
- Mission flexibility
- Continuous availability
- Rapid upgrades
- Low ownership cost

QoS REFERENCE ARCHITECTURE



HiPer-D DEMO BLOCK DIAGRAM



AAW, TBMD CALL FOR FIRE → Simulation AAW/TBMD Fault tolerant and/or Scalable Land Attack C2/BMC DARPA



HiPer-D/AQuA EXPERIMENT

Display State Server

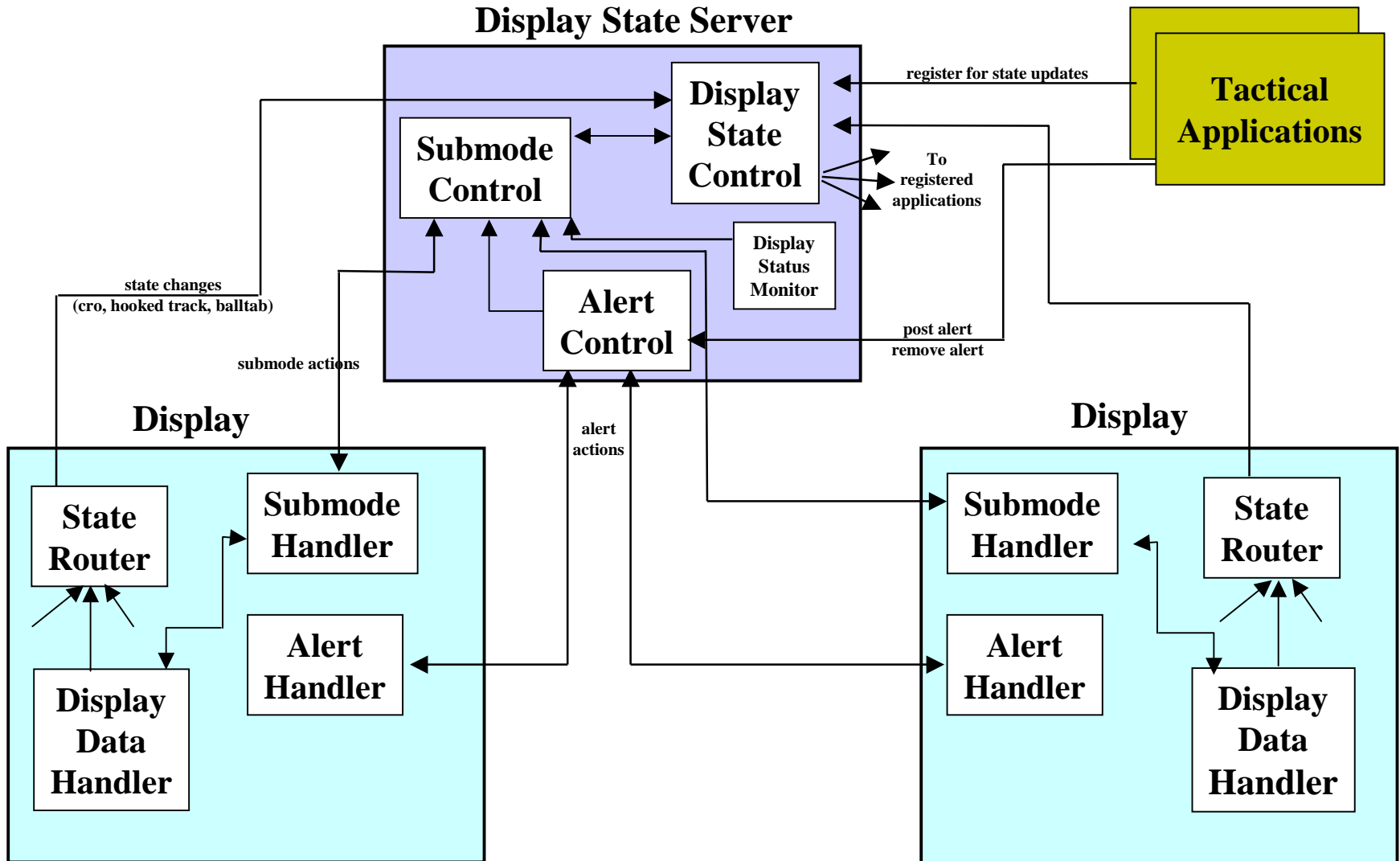
- Replace a critical part of the HiPer-D tactical display subsystem.
- Maintain critical state pertaining to operator responsibilities (submodes) and operator alerts issued by tactical processing (e.g. engagement alerts)
- Distribute display status to registered tactical applications
- State critical - unacceptable to lose track of operator responsibility assignments or to lose an unhandled alert
- Soft real-time - interfaces with displays and multiple tactical components, including some with stringent real-time requirements
- Non-deterministic behavior - multiple threads of control – order of handling of some events has the potential to affect resulting application actions and/or state

AQuA FT CORBA Framework

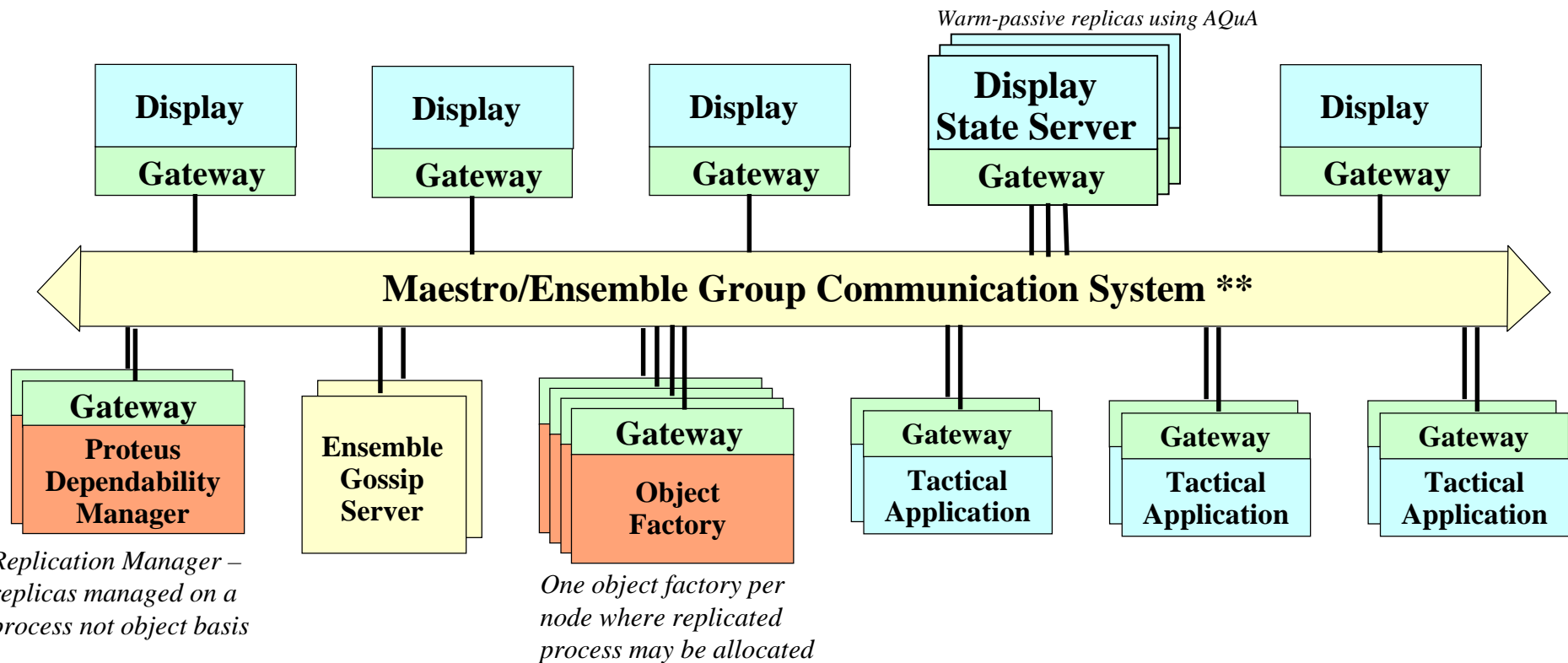
- Developed by University of Illinois & BBN
- Uses Ensemble Group Communications System and TAO ORB
- AQuA Gateway allows FT application to interact with other ORBs
- Complies with the spirit of the FT CORBA Specification:
 - Strong Replica Consistency via state transfer, Ensemble reliable ordered multicast
 - Warm passive & active replication
 - Voting mechanism for tolerance of value faults
 - Detection of & recovery from crash failures
 - Dependability manager to monitor replica status and manage replication level
 - Object factories to initiate new replicas
- Value added:
 - Support for some types of non-determinism via per object invocation state xfer
 - Replicas managed per process not per object

Constraint: *Fulfill all required functionality of the application within the context of the existing system and with minimal change to the other applications.*

DSS ARCHITECTURE



DSS/AQuA ARCHITECTURE

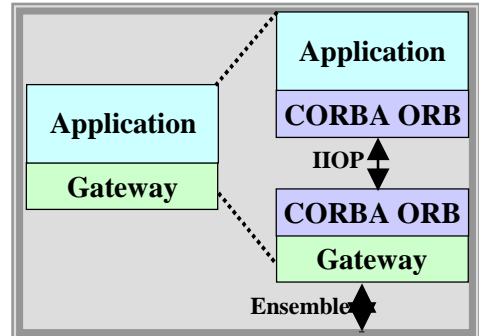


Replication Manager – replicas managed on a process not object basis

One object factory per node where replicated process may be allocated

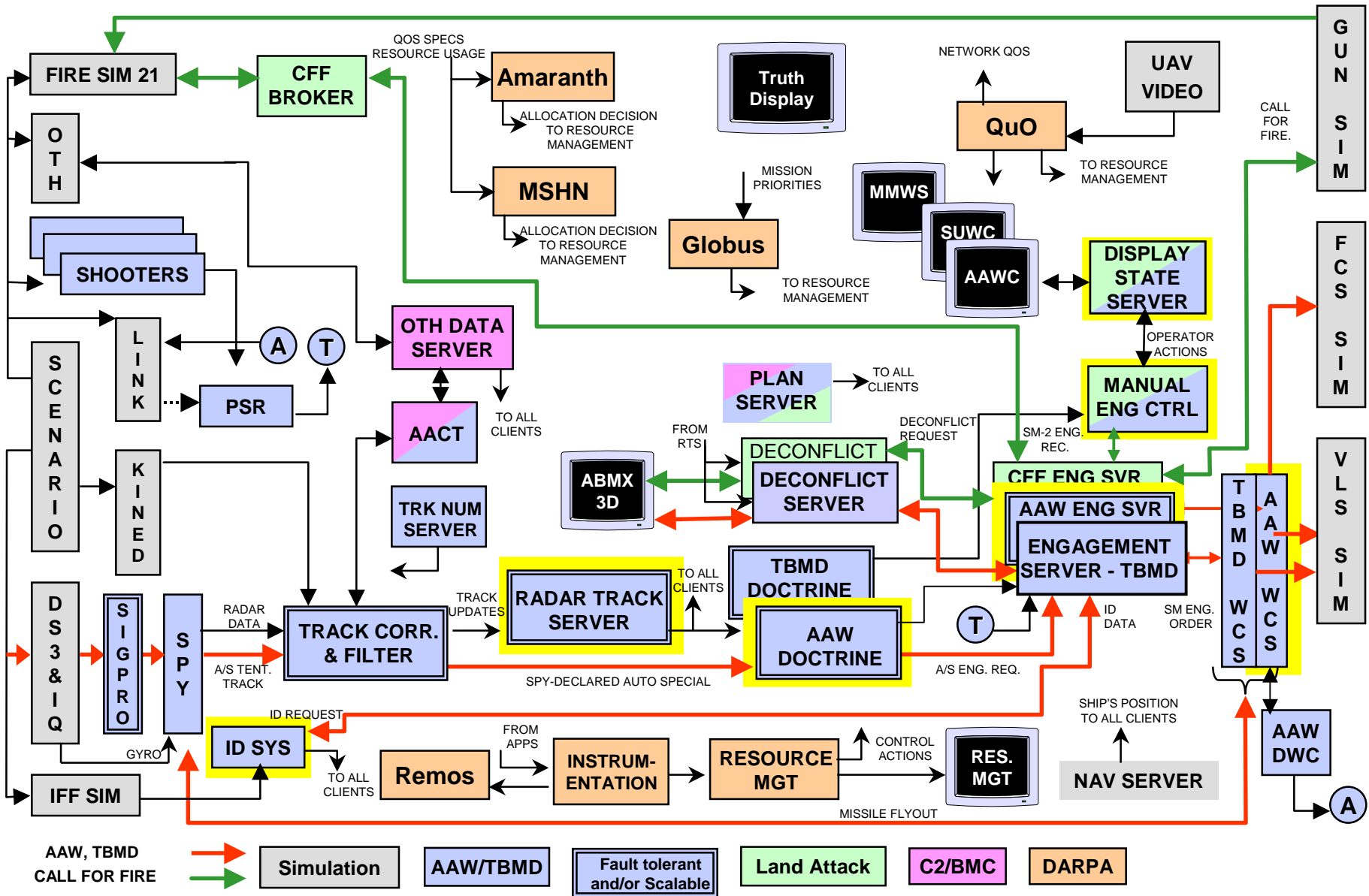
* AQuA passive replication performs state transfer after each method call on/by a replicated object.

** Ensemble GCS provides FT Group membership, heartbeats, failure detection, ordered atomic multicast.



Interceptors transform CORBA method calls into Ensemble messages.

HiPer-D DEMO BLOCK DIAGRAM



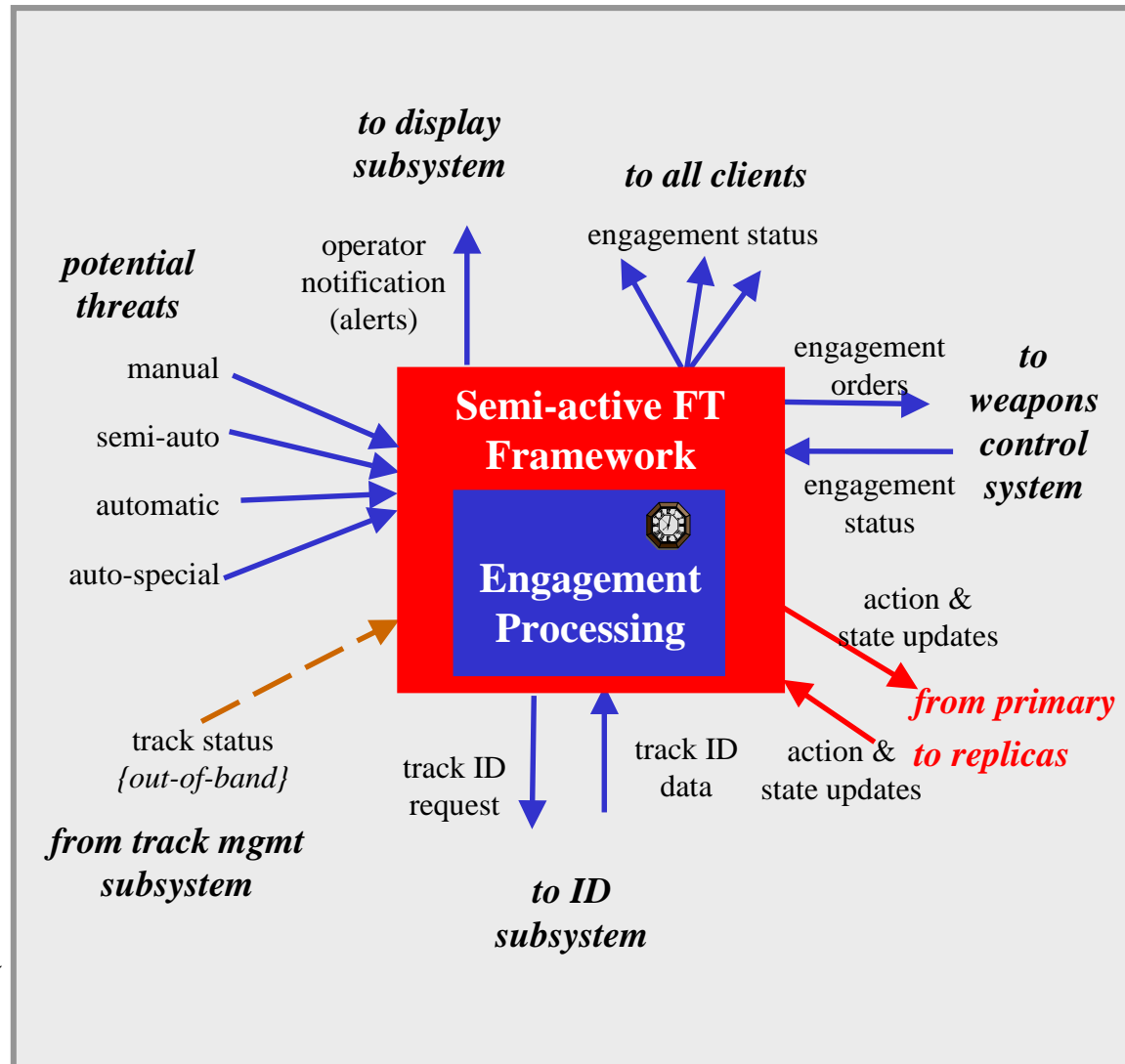
AAW ENGAGEMENT SERVER

Characteristics:

- Real-time – extremely fast recovery needed
- Event timers
- Multiple threads of control
- Message ordering via group communications (multiple groups)
- Out-of-band (pub/sub) communications
- Inputs that trigger sequences of events
- Multiple data sources & events that form a composite state

Technical Challenges:

- Affect of non-determinism introduced by thread execution order, timer expiration on composite state
- Maintaining real-time performance requirements - even during failure of a primary replica
- Replica restart – state includes both data and events





STRONG REPLICA CONSISTENCY

From CORBA 2.6 Specification, Section 25.1.3.4

... requires that the states of the members of an object group remain consistent (identical) as methods are invoked on the object group and as faults occur.

- Active Replication

... at the end of each method invocation on the object group, all of the members of the group have the same state.

- Passive Replication

... at the end of each state transfer, all of the members of the object group have the same state.

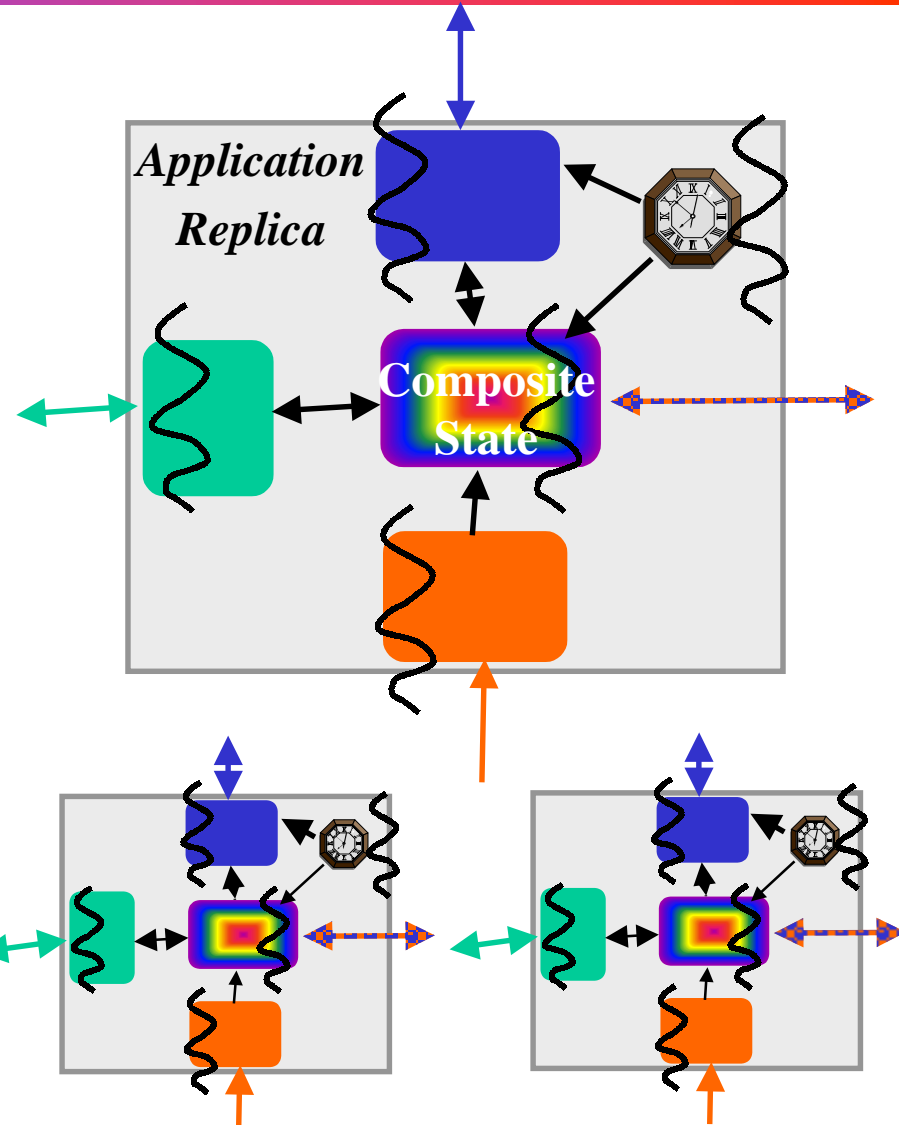
STATE CONSISTENCY

Requirement

- Maintain state data consistency among replicated components of a real-time distributed system *to the degree required to effect replica fail-over within the required time limits*

Technical Issues

- Multiple objects within a single application process may interact to create a *composite state*
- *Multiple threads* may be required with a single process – possibly multiple threads per object
- *Event timers* may be used to initiate periodic processing or monitor for time-out conditions
- *Out-of-band communications*, sometimes via unreliable channels (sensor interfaces, legacy interfaces, interfaces to persistent media)
- *Time-based computations* (e.g. an algorithm that extrapolates data forward in time)
- Sequences of actions resulting from single initiating event
- *State shared* across objects in disparate, distributed processes
- High throughput/low-latency interactions, with *only a small subset of communications affecting the state* that must be maintained across replicas
- State may include both *application/object state* and *infrastructure/ORB state*



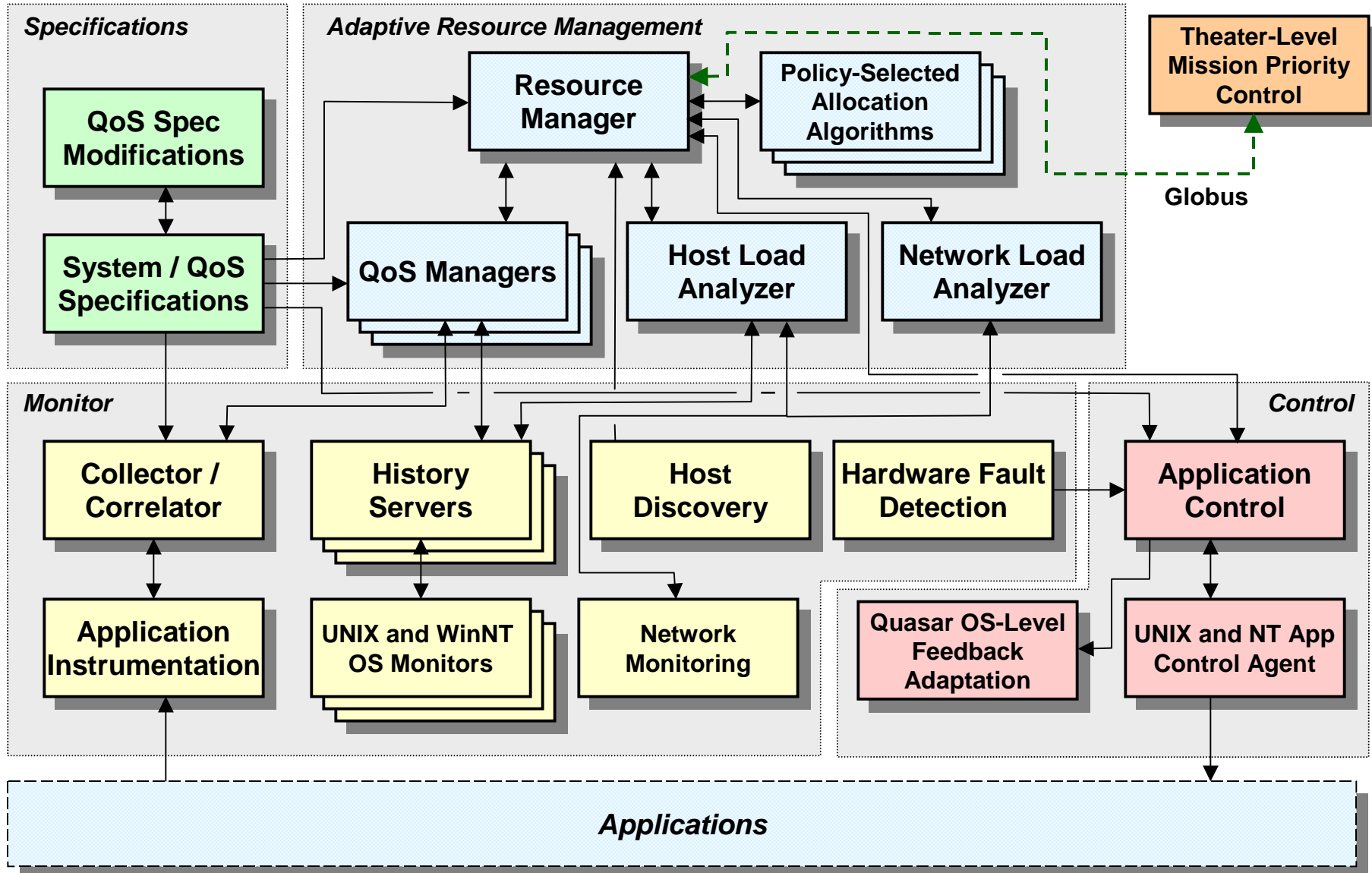


HiPer-D DYNAMIC RESOURCE MANAGEMENT

- Distributed Monitoring and Control Infrastructure
- Management of Heterogeneous Pool of Network and Host Resources
- Dynamic Allocation and Reallocation of Applications within the Computing Pool
 - to maintain user-specified system performance goals.
 - to respond to faults, tactical load changes, and mission changes
- Integrated Infrastructure Components:
 - Monitoring
 - Decision-Making
 - Control
 - Specifications

HiPer-D DEMO 01

DRM ARCHITECTURE





DEFINITION OF DRM OPEN INTERFACES

- **Ongoing effort to define and implement open DRM interfaces**
 - Use standards-based middleware where applicable
 - Allow for incremental enhancement of DRM capabilities by research community
 - Allow alternate components and algorithms to be easily integrated and evaluated
- **Initial Open Interface Candidates:**
 - Accessing and updating system specification information
 - Monitoring and updating host, network, and application status information
 - Accessing application-level instrumentation information
 - Providing alternate allocation/reallocation algorithms



DEFINITION OF DRM OPEN INTERFACES

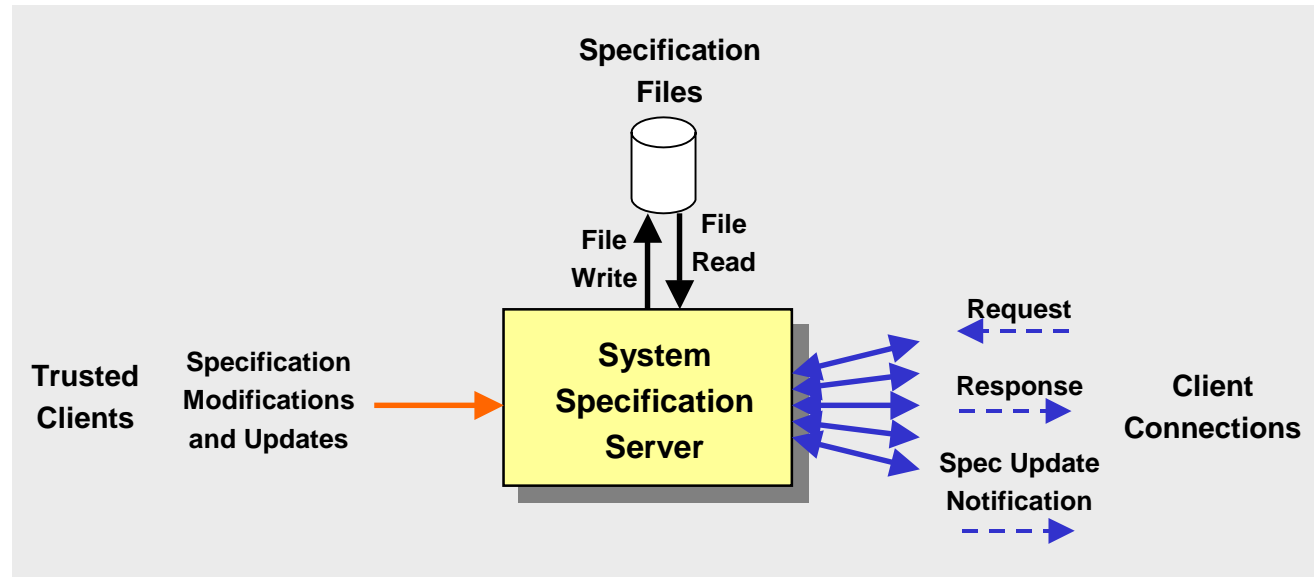
- **Approach:**

- Define and document interface requirements:
 - data requirements (needed information, data formats, etc...)
 - control flow requirements
 - QoS requirements (timing, scalability, survivability, security, etc...)
- Identify and evaluate relevant CORBA services
 - determine services that support most or all interface requirements
 - identify and document shortfalls
- Define detailed interfaces and API's based on selected services
- Implement defined interfaces
- Evaluate performance of defined interfaces
- Document lessons-learned and iterate

DRM OPEN INTERFACE EXAMPLE

Example: Distribution and Updating of System Specification Information

- Spec Info Needed Throughout RM Infrastructure
- Different RM Components Require Different Spec Info
- 10's to 100's of Clients
- Clients (and possibly Servers) Brought Up, Down, and Moved at Run-Time



Interface Requirements:

- Client Specification Requests
 - Reliable Soft Real-Time (Bounded) Response Time Requirements
 - “Timely” Client Notification when Server(s) Go Down
- Specification Updates and Distribution of Updates:
 - Reliable Near Real-Time Pub/Sub Model Needed for Notification of Updates
 - Soft RT requirements for Maintaining State Consistency Throughout System (low secs)
 - Optional Security / Authentication needed for Spec Updates
- Derived Requirements:
 - Fault Tolerance
 - State Consistency
 - Scalability



NETWORK QoS CONTROL

- Defining Network QoS Monitoring and Control Framework
 - defining requirements, responsibilities, and interactions between:
 - applications
 - middleware
 - resource management
 - network QoS policy
 - network QoS monitoring and control mechanisms
- Issues:
 - At what level(s) should network QoS control mechanisms be accessible?
 - Do Applications and Middleware require knowledge of and/or access to network QoS control mechanisms?
 - How should Network QoS Control Policy be incorporated into the framework?
 - How to monitor Application and Middleware-level network information?
 - How to control Application and Middleware-level network interfaces?

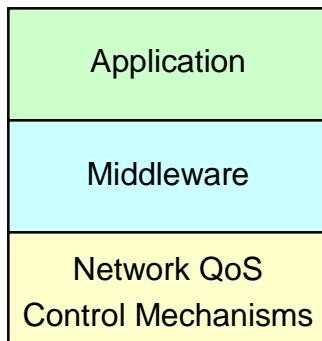
RM NETWORK QoS CONTROL EXPERIMENT

Local vs. Global View of Network QoS Control Actions:

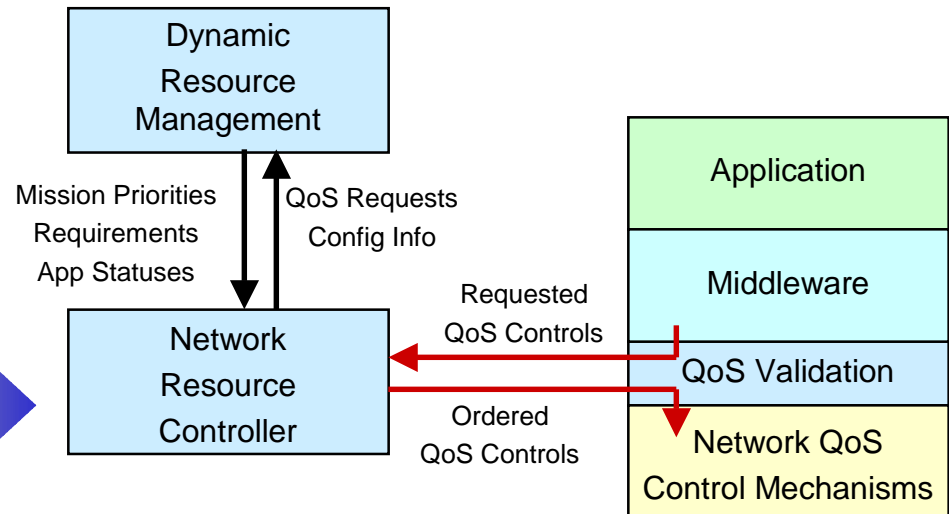
- **Application-level:**
 - local perspective only
 - knowledge of impact on application performance only
- **Resource Management-level:**
 - system-wide perspective
 - system-level view of impact across all applications

Goal: RM Monitoring and Control of Network QoS Via Interception and Remapping of Application / Middleware-Initiated QoS Calls

- Transparent RM Control of Network QoS
- Ability to Monitor and Control Middleware Network QoS Activity



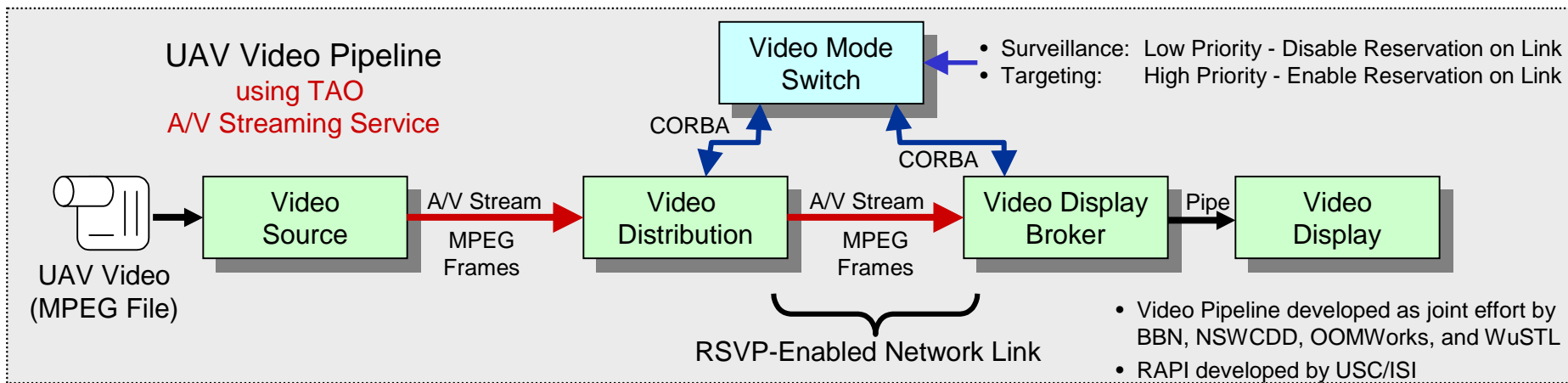
Notional Layering for Application-Level Control of Network QoS (via Middleware)



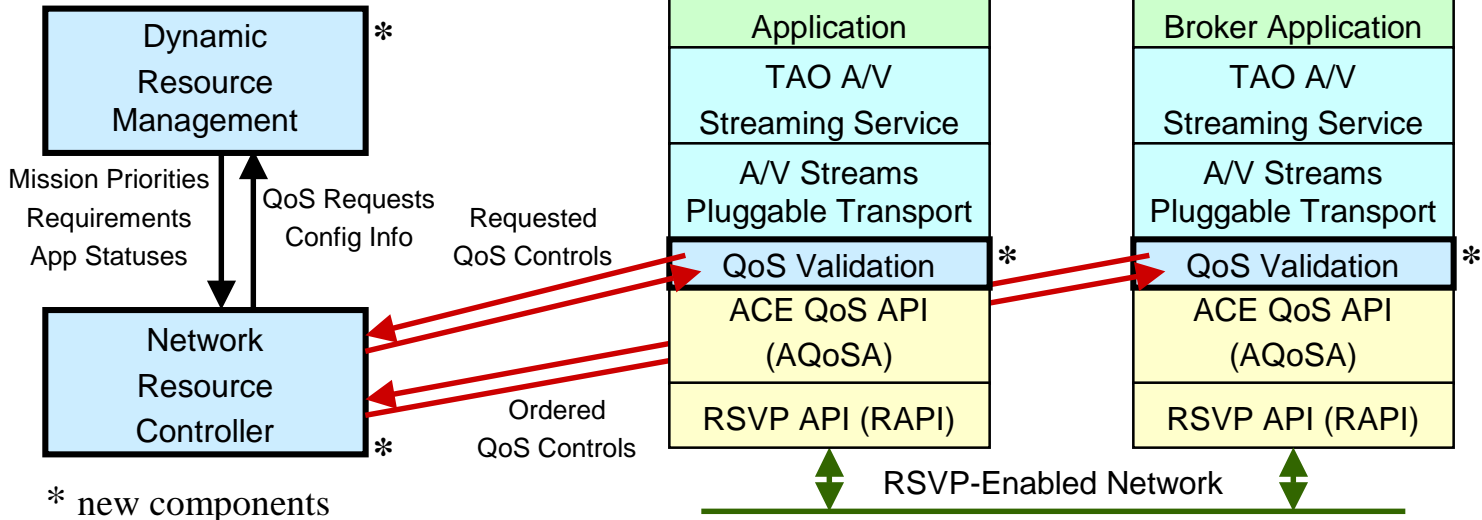
Notional Layering and Interfaces with Resource Management for Network QoS Control

RM NETWORK QoS CONTROL EXPERIMENT

Scenario: Unmanned Aerial Vehicle (UAV) Video Pipeline



Experiment Architecture:



3 Control Cases:

- Application QoS requests will be accepted, rejected, or modified based on mission priorities and requirements
- Asynchronous changes to network QoS may be ordered based on mission priority changes
- QoS change indications originating from network devices will be handled

* new components