Towards Quality of Service in Distributed Systems

Dave Bakken

School of Electrical Engineering and Computer Science

Washington State University, USA

http://www.eecs.wsu.edu/~bakken/

http://www.dist-systems.bbn.com/tech/QuO/

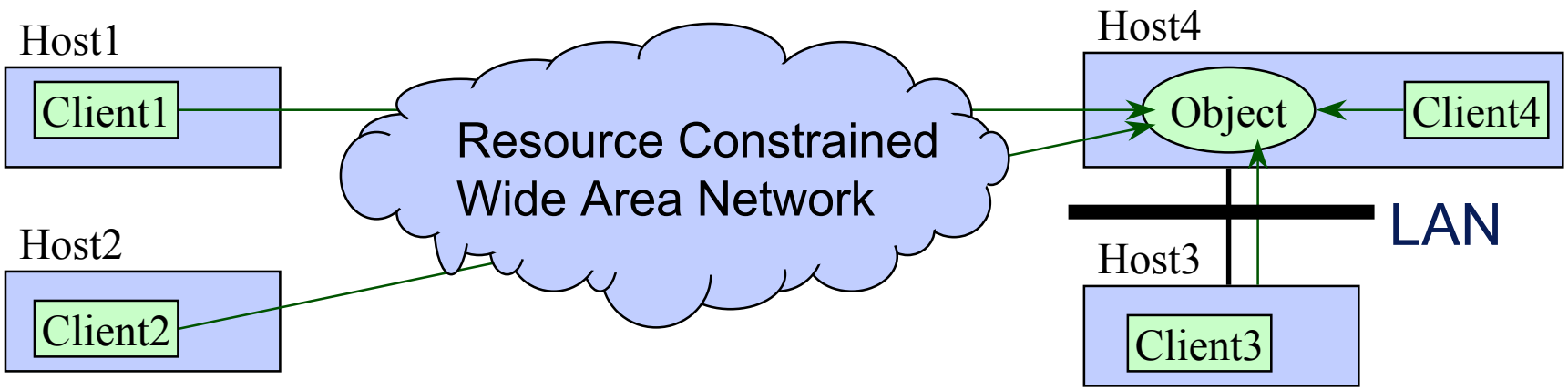
Seminar at University of Oslo

June 28, 2001

Outline of Talk

- QoS: The problem, and basic definitions
- QoS Implementation Issues
- Quality Objects (QuO) 2.0 Architecture
- QuO 2.0 Case Study
- QuO 3.0: Aspects and Reuse
- QuO 3.0 Case Study: UAV Multimedia adaptivity

The Problem: Wide-Area Distributed Applications Are Hard to Build and Maintain



- WANs are dynamic, unpredictable, and unreliable
- Hosts span a wide range of platforms
- Servers provide a variety of services and interfaces
- Changing requirements and configurations
- Complex interactions

Client just wants predictable behavior (as much as possible)!

Client programmer does not want to deal with managing the above details!

The Problem (cont.)

- Many distributed systems are too expensive to build and maintain, and
  - Cannot adapt sufficiently at runtime
  - Cannot evolve over lifetime to handle new requirements or work in new environments
- One reason: no systemic support for building distributed systems using shared resources
- Key challenge: how to create predictable distributed systems application programs which
  - Can operate acceptably when usage patterns or available resources vary over a wider spectrum and with much less predictability
  - Can be modified in a reasonable amount of time
  - Are reasonably affordable
- Needed: Middleware which makes a distributed application's hidden quality of service assumptions (usage, resources) explicit, to
  - Help make the environment more predictable to the app, and
  - Help the app. to adapt when predictability fails
  - Note: this involves both distributed systems and software engineering issues!

QoS == the "how" to do the functional (IDL-described) "what"

- IDL tells "what" can or should be done
  - `void sort(inout long a[], in long n);`
  - `long lookup(in string name);`
- Quality of Service (QoS) is the non-functional "how" to do the above "what"
  - timeliness (delay, jitter)
  - throughput (volume)
  - availability/depenability
  - security (integrity, confidentiality)
  - cost
  - precision
  - accuracy
- No standard definition(s) of QoS yet, but progress being made towards implementing multiple QoS properties (a.k.a. QoS dimensions -- the "what" items: timeliness, etc. above) in one framework

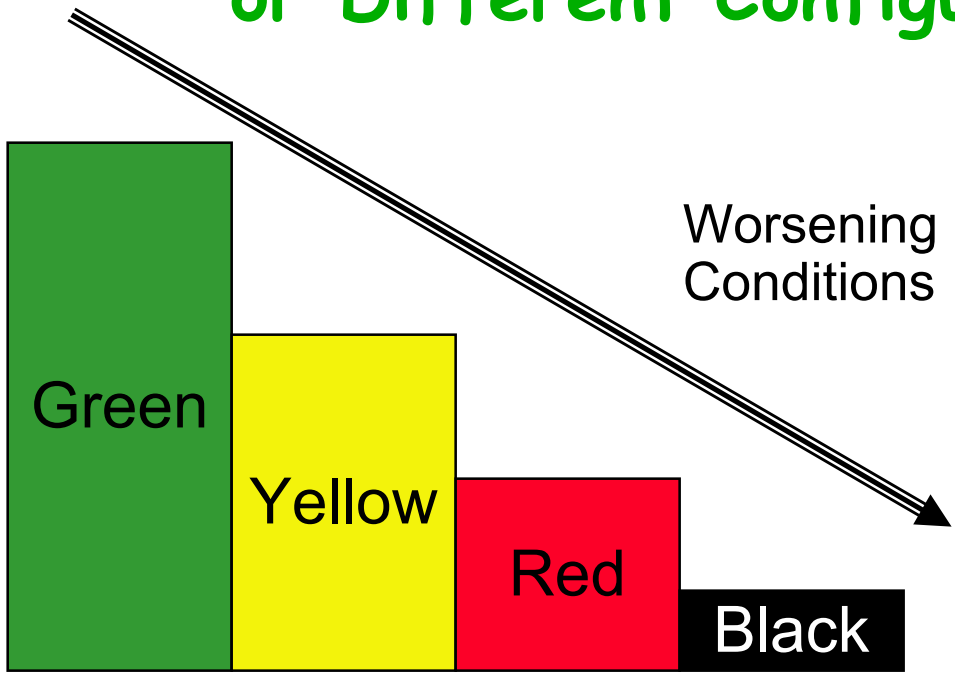
QoS Basic Definitions (cont.)

- Premise 1: Different levels of service (not “all or nothing”) are possible and desirable under different conditions and costs
- Premise 2: The level of service in one dimension must be coordinated with and often traded off against the level of service in another
- Premise 3: Keep the functional and non-functional separate if possible
  - Let them be able to change independently (reuse)
  - Let them be managed by different people (QoS specialist, domain specialist)
- Question: How aware should client applications be of QoS:
  - Unaware (totally handled by something else)
  - “Awareness without pain”
  - Immersion (has to handle large amounts of QoS details and issues etc)

Different Views of QoS

- LAN multimedia with no adaptivity
- Bill Gates: end-user satisfaction
- “World Wide Wait”
- ISPs
- Power users
- IT Managers
- Dilbert Managers
- HP and other vendors (IWQoS ‘97, WebQoS, …)
- Builders of Big and Critical Systems
  - Cannot manage the “non-functional” behavior of their systems well
  - Cannot ride the technology curve over the lifecycle!
  - Examples
    - DARPA ITO Quorum program and Navy’s DD-21 ship program
    - Boeing (Commercial, Phantom Works, other)

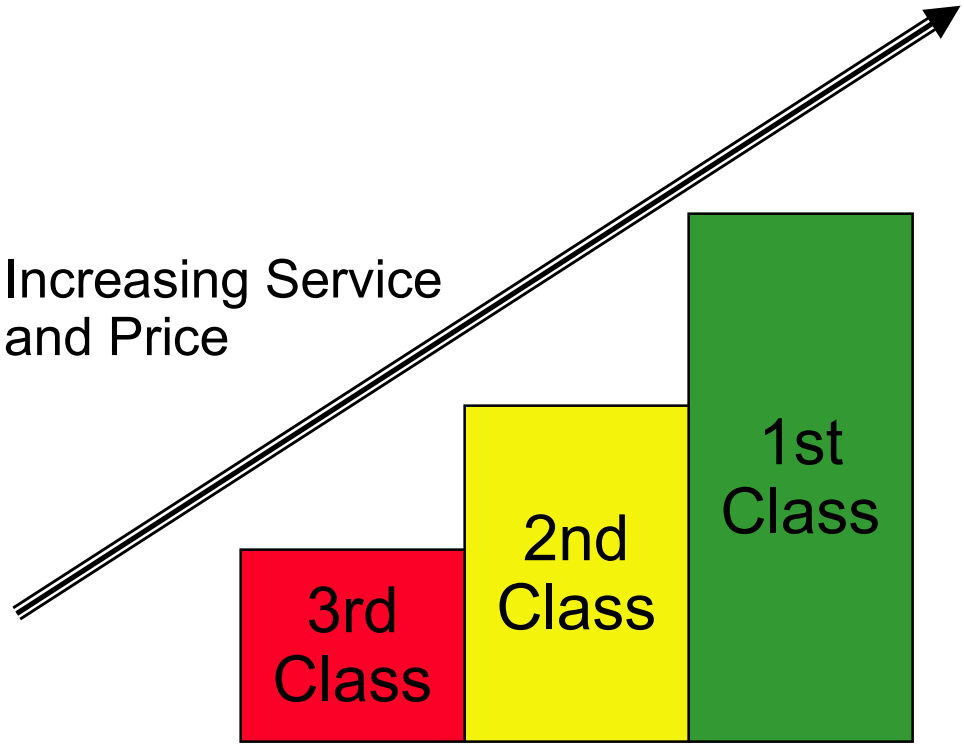
QoS for Users: Adapting to Worsening Conditions or Different Configurations



- Program can be empowered to automatically adapt to worsening conditions (balance of supply of to demand on current shared resources)

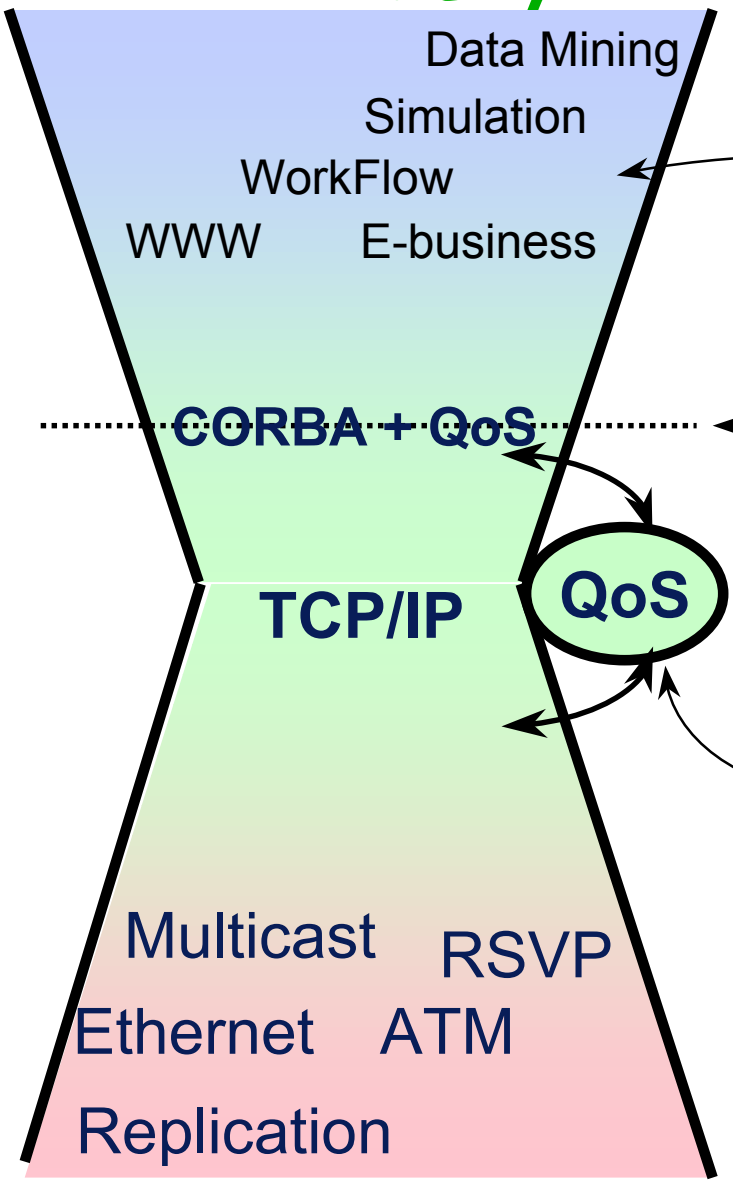
Condition	Conferencing	Participants	Info Service
Green	Full color multimedia	Key and useful participants	Quick DB queries
Yellow	B&W multimedia	Key and useful participants	Acceptable DB queries
Red	Audio	Key participants only	Acceptable DB queries
Black	None	None	None

QoS for ISPs & HW Suppliers: Multiple Levels of Service Enable Differentiated Products



- 3rd class: Best-effort
- 2nd class: Statistical performance guarantees
- 1st class: Absolute performance and availability guarantees

OO Middleware with QoS Extensions is a Powerful Abstraction Layer on which to Build Applications



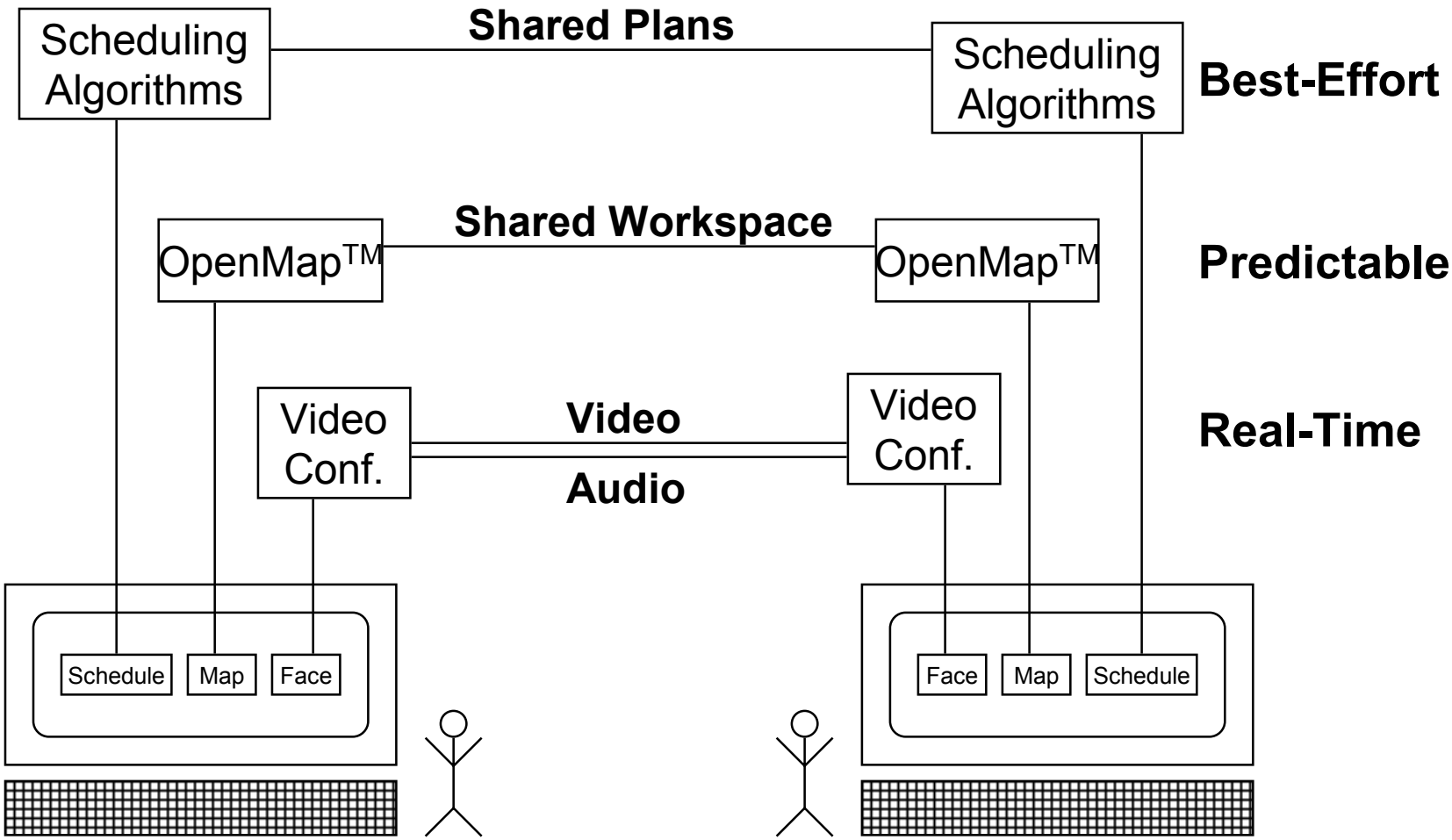
Diverse applications for geographically dispersed, heterogeneous environments – <u>not just multimedia apps</u>!

Distributed objects are the first abstraction layer that unifies CPU, storage, and communications

This interface needs to be hidden from applications

- It is too complicated
- It is changing too quickly

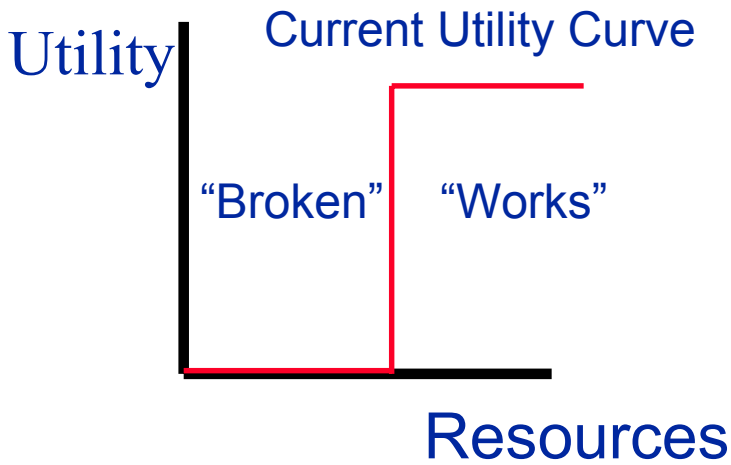
QoS is Not Just Multimedia over a LAN or MAN

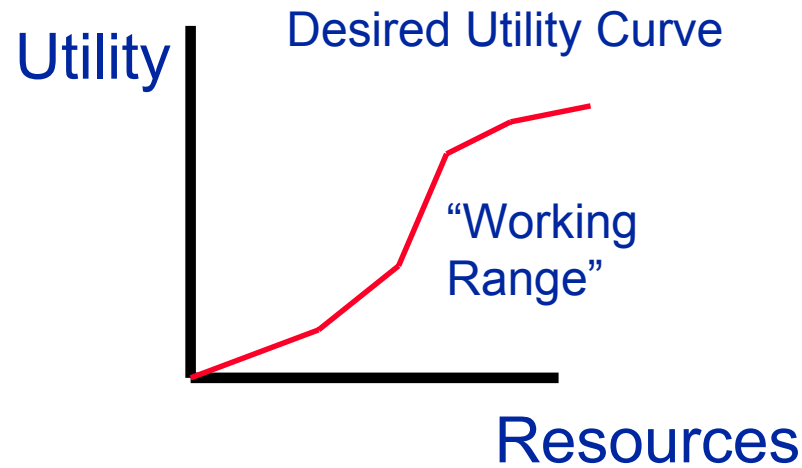


Outline of Talk

- QoS: The problem, and basic definitions
- **QoS Implementation Issues**
- Quality Objects (QuO) 2.0 Architecture
- QuO 2.0 Case Study
- QuO 3.0: Aspects and Reuse
- QuO 3.0 Case Study: UAV Multimedia adaptivity

"Awareness without Pain" I: Users' and Application Programmers' Awareness



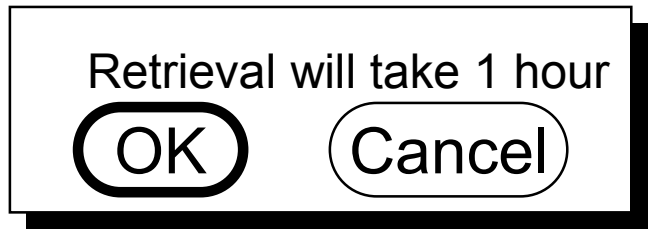


- Users and application programmers need to be **aware** of their demand for resources, and be able to **change!**
- Users/applications must understand the **utility** of their demand
  - know their usage patterns and QoS requirements
- Users/applications must be able to **change demand** based on volatility in the environment
  - need to be able to determine utility of additional resources, and ability to do without
- System infrastructure will improve its "transparency" over time, and its effectiveness of masking variability

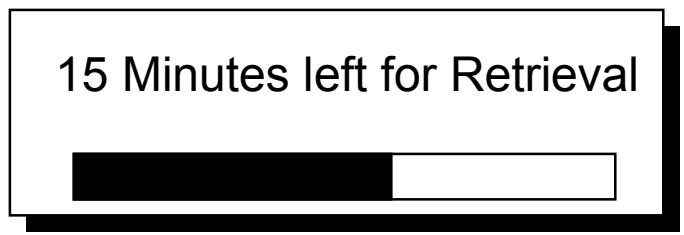
"Awareness without Pain" II: Users Should See a "Graceful Degradation" of App, not a Hard Failure

- Functions marked with cost cues

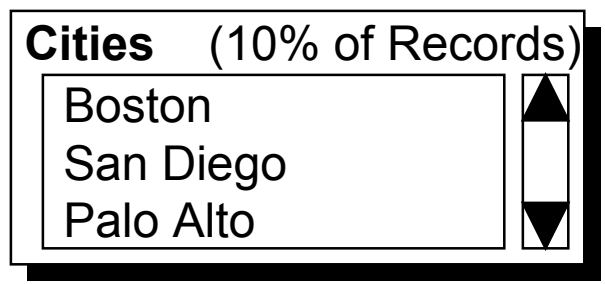
- Middleware asks for more advice



- Middleware predicts long response times

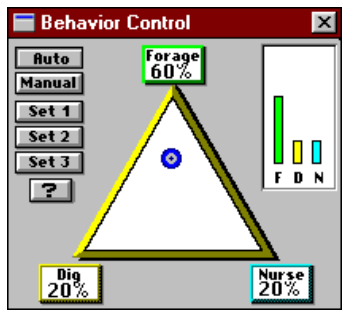


- Application tolerates aborted operations with partial results

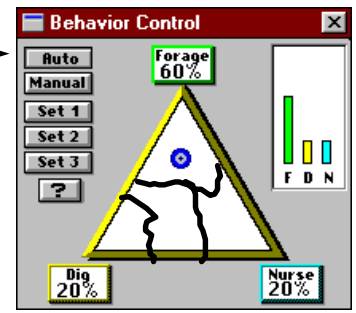


"Awareness without Pain" III: Needed--- Higher-level QoS APIs and User Interfaces

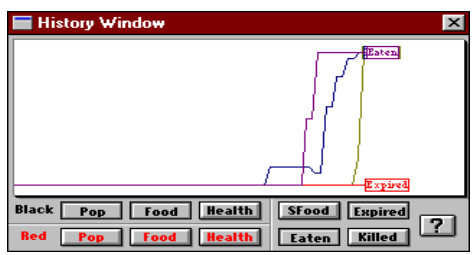
- Most programmers and users of advanced distributed applications can't deal with QoS because they
  - Are not very sophisticated in distributed systems issues (let alone QoS)
  - Have enough to do already providing/using the applications' main job without worrying about QoS
- QoS contracts can give a high-level API for programmers to use, with the help of QoS framework implementers & QoS developers
- Simple (single-) application management user interfaces can help
  - User control:



QoS developer provides multiple implementations trading off multiple properties and resources, with a high-level mapping



  - User feedback:



QoS-Aware Resource Management I: Many Mechanisms Give the Correct Functionality, But Are Appropriate for a Small Set of Conditions

Usage Pattern

Arrival Rate
Priority

Applications

know Their Usage Pattern and QoS Requirements

QoS

Performance
Availability
Security
...

Mechanism

given usage pattern and resources, yield QoS and Utilization

Allocation Algorithms

System Managers

setup resources and set usage polices

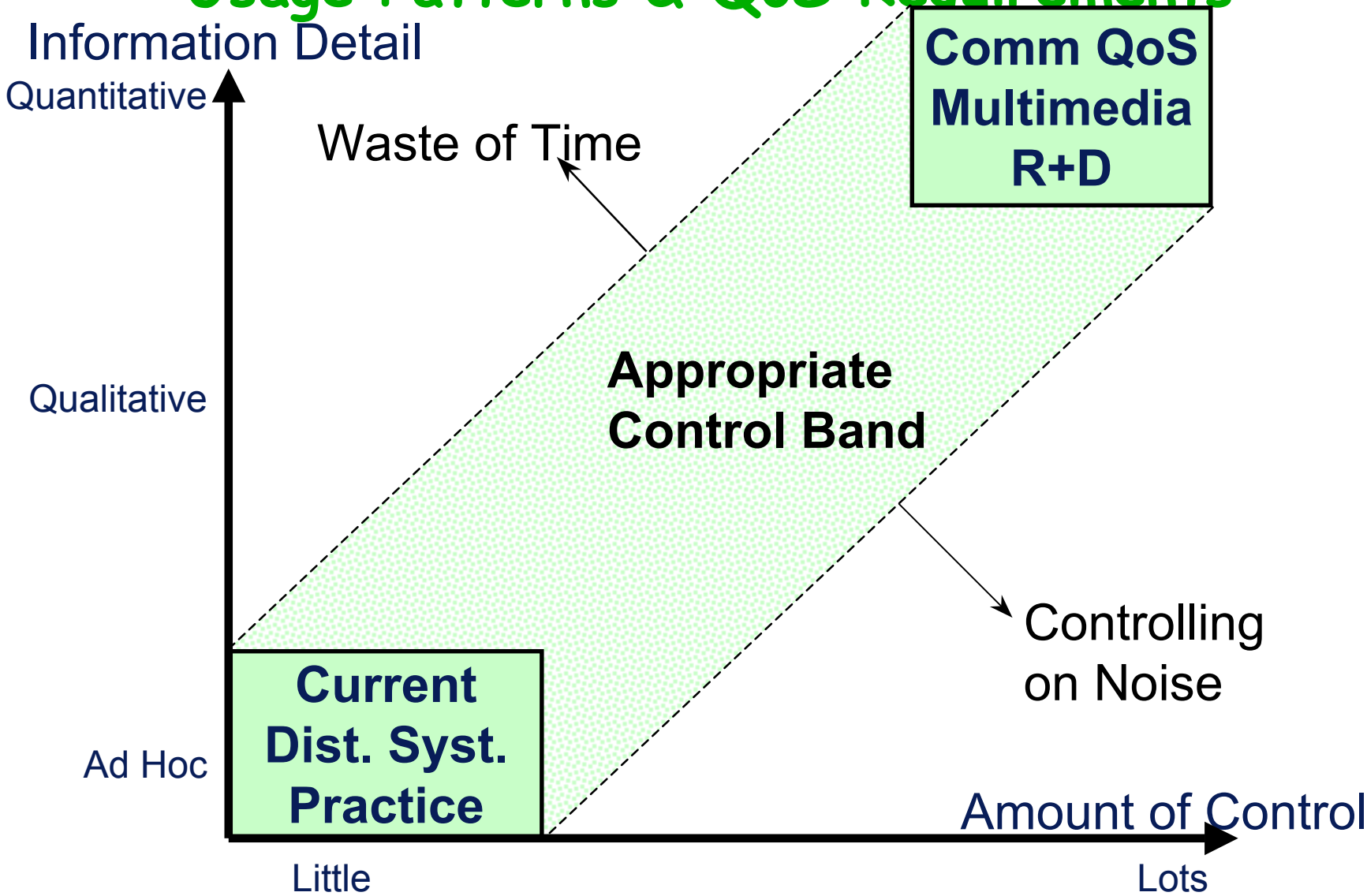
Resources

Capacity
Reliability

Utilization

Cost
Ownership

QoS-Aware Resource Management II: Control over Resource Allocation is Useless w/o Information on Usage Patterns & QoS Requirements



Application-Level Adaptation Choices

- How can distributed applications become more predictable and adapt to changing system conditions?
  - Control and Reserve Resources
  - Utilize alternate Resources (redundancy)
  - Use an alternate mechanism (with different system properties)
  - Take longer
    - reschedule for later
    - tolerate finishing later than originally expected
  - Do less
- Note the multiple possible layers of adaptation:
  - Client application
  - Above the ORB core on client-side
  - Inside the ORB
  - Above the ORB core on server-side
  - Server
- Premise: supporting all the above choices is helpful!

QuO's Philosophy: Support Monitoring of System Conditions & Adapting to Changes at All Levels

- QoS middleware needs to integrate disparate information ("QoS meta-data") over:
  - providers
    - QoS API+middleware designer
    - QoS contract designer
    - application program (client)
    - remote object
    - operations staff (configure resources)
    - network management information, …
  - locations
    - client host
    - remote object host
    - network
  - times
    - language design
    - application development
    - application initialization
    - contract setup
    - change in network conditions
    - invocation, …

Supporting Monitoring of System Conditions and Adapting to Changes at All Levels(cont.)

- Guarantees/correctness versus Advice/Improvement for Predictable Behavior
  - It is not feasible to provide absolute “guarantees” over WANs with an arbitrary mix of hosts, resources, operating systems, etc.
  - It is useful to be able to
    - Organize information about an application’s requirements and expected usage
    - Reserve as much of the end-to-end resources as possible to make the application more predictable (lower variance)
- QoS contracts are crucial for adaptivity, i.e., regions representing state of QoS expectations vs. actual conditions
- Need to provide for a new role -- QoS engineer -- to help simplify the application developer’s task

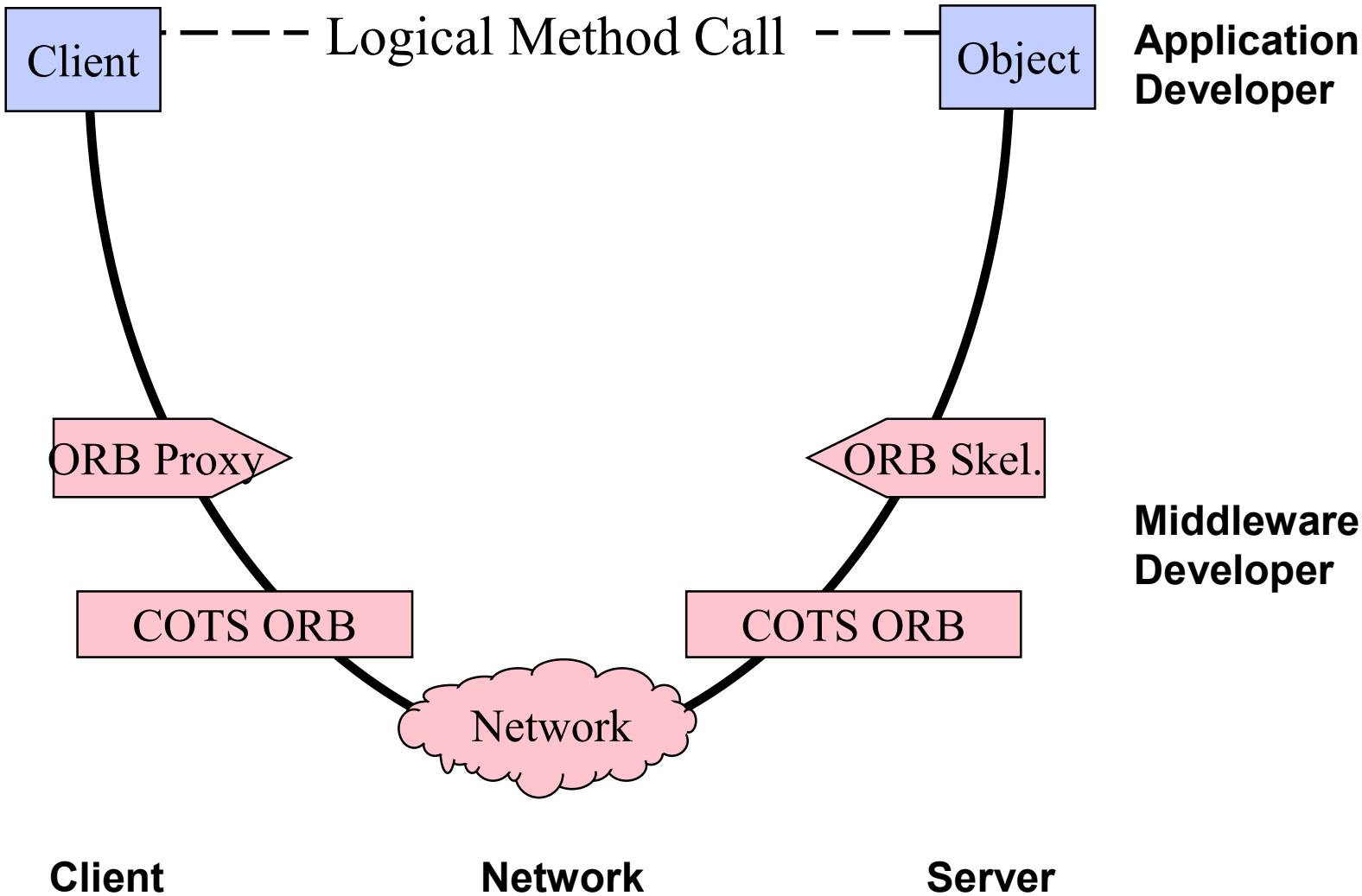
Outline of Talk

- QoS: The problem, and basic definitions
- QoS Implementation Issues
- **Quality Objects (QuO) 2.0 Architecture**
- QuO 2.0 Case Study
- QuO 3.0: Aspects and Reuse
- QuO 3.0 Case Study: UAV Multimedia adaptivity

QuO History

- BBN Distributed Systems Dept had lots of experience since late 1970s
  - Distributed Applications over WANs
  - Middleware to support above (CORBA-like Cronus/Corbus)
- New Rome Lab Contract "Distributed Computing over New Technology Networks" for a study project, started in 8/1994
  - New networking technologies coming….
  - But how can they help the application level?
  - (I was hired for this, right after PhD)
- Candidate technologies: multicast and reservations/QoS...
- QuO architecture requirements and initial design: Zinky and Bakken and Schantz (1995-6), a handful of others since
- Led to 6+ DARPA ITO and ISO QuO contracts, and still going strong!
- Used at a number of universities & companies to integrate their QoS research (CMU, GaTech, Cornell, U. Illinois, Wash. U. St. Louis., Columbia U, Trusted Information Systems(TIS), Boeing,...)

Simplified Distributed Object (CORBA) Runtime Components

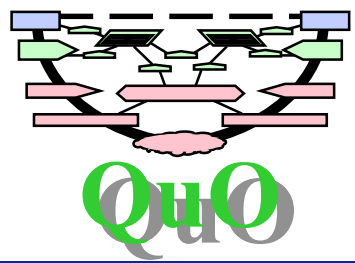


The Quality Objects (QuO) Framework Supports Development of Adaptive Distributed Applications

QuO is a reuseable middleware framework that provides a common approach to adaptable QoS suitable for applying to any number of QoS dimensions

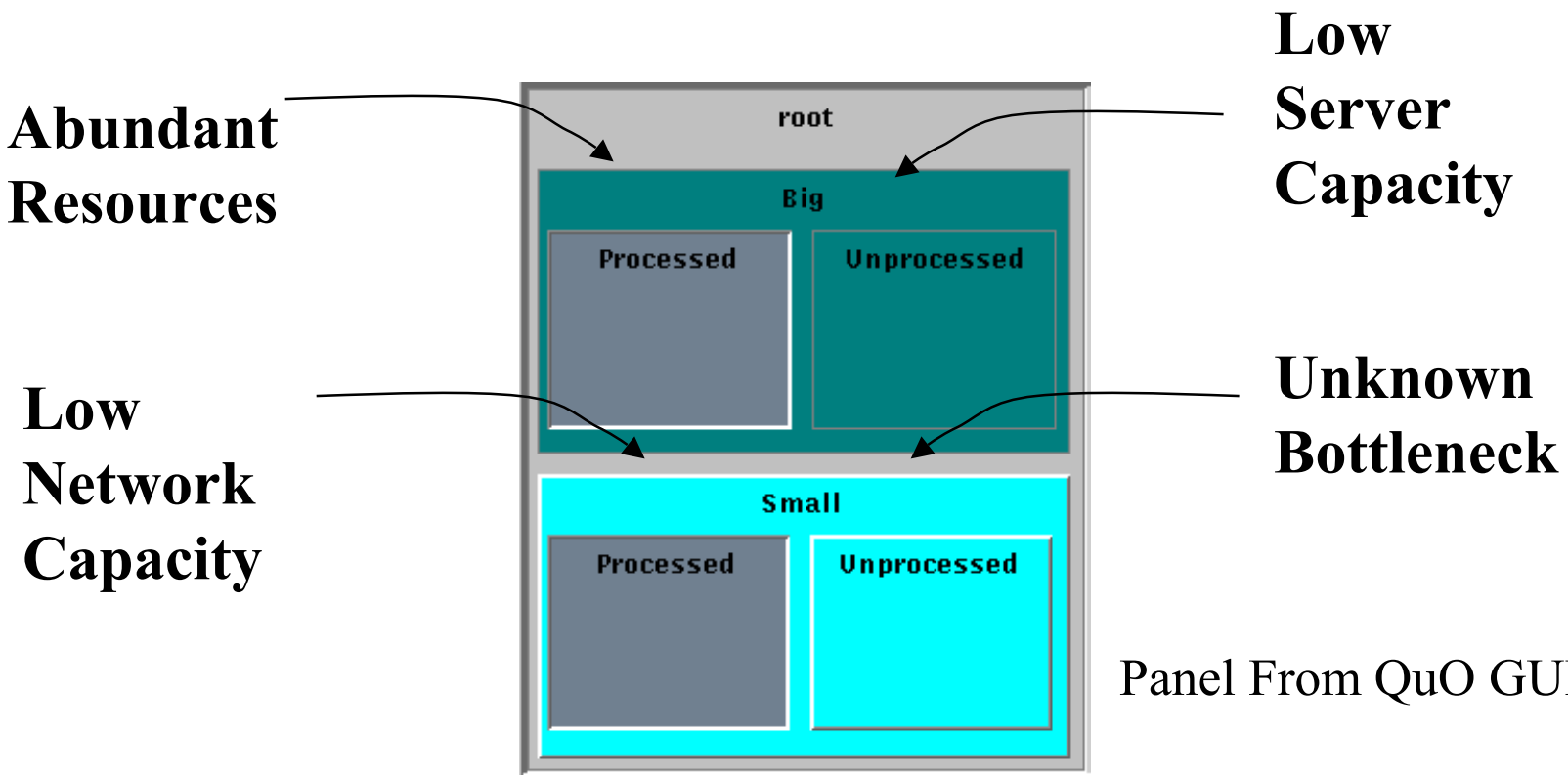
The QuO framework provides

- Separation of concerns between software functional properties and QoS needs
  - Specify QoS desires, implementation alternatives separately from the functional application
- Monitor and measure QoS in the system
  - Consistent interfaces for QoS measurement and resource management control
- Facilities to enable application- and system-level adaptation



Contracts Summarize System Conditions into Regions, Each Appropriate for Different Situations

- Contract defines nested regions of possible states based on measured conditions
- Predicates using system condition objects determine which regions are valid
- Transitions occur when a region becomes invalid and another becomes valid
- Transitions trigger adaptation by the client, object, ORB, or system

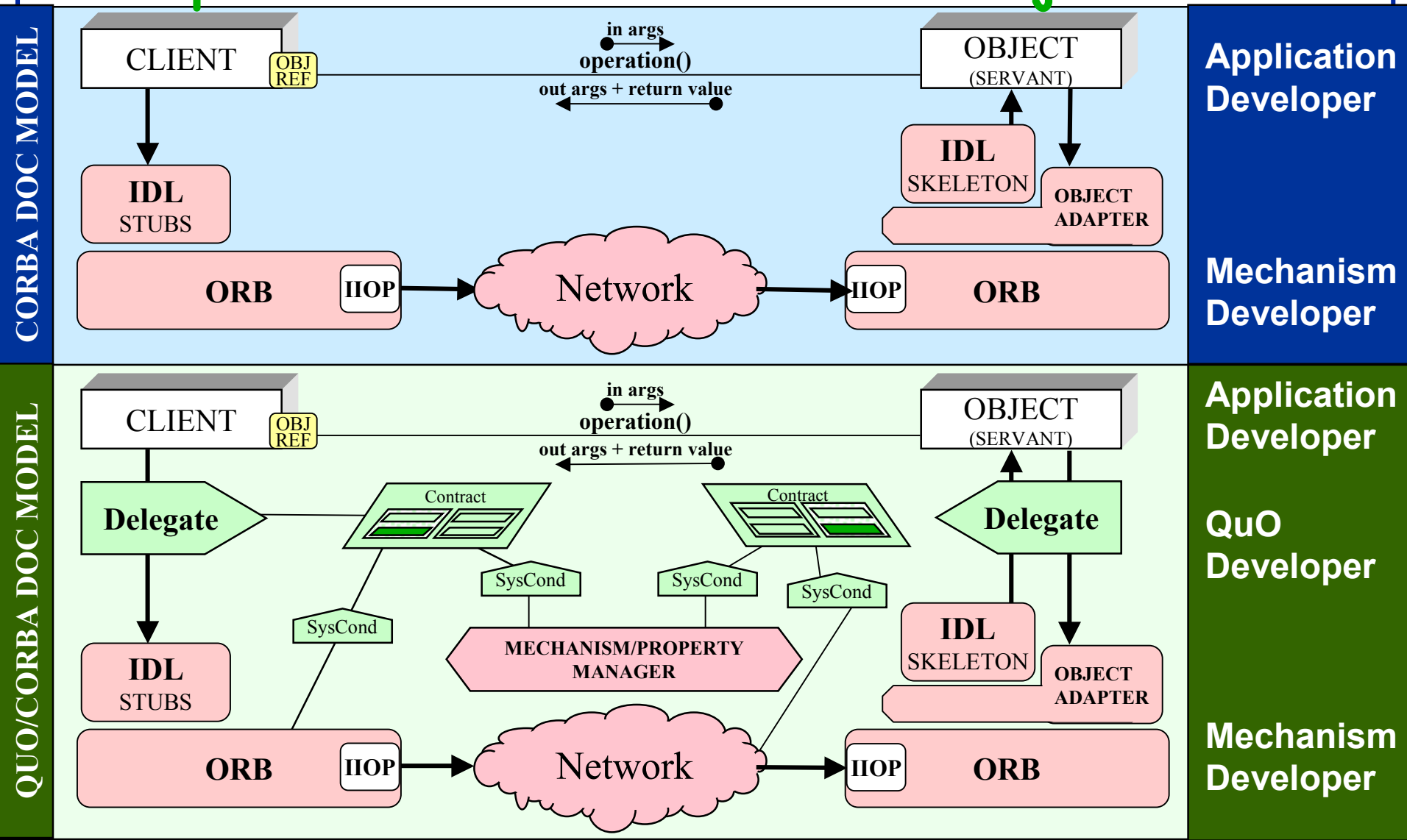


Panel From QuO GUI

A QuO application contains additional components (from traditional CORBA/DOC applications)

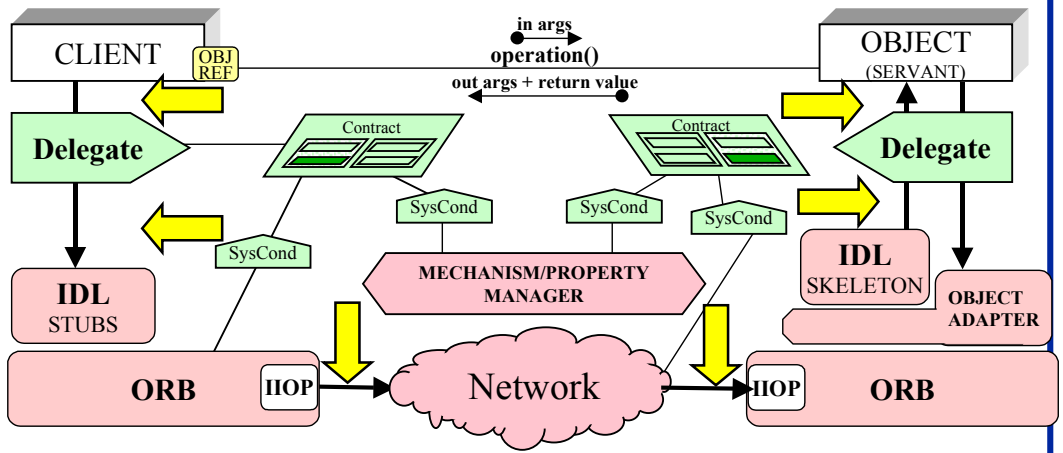
- **Contracts** summarize the possible states of QoS in the system and behavior to trigger when QoS changes
  - Regions can be nested, representing different epochs at which QoS information becomes available, e.g., *negotiated regions* represent the levels of service a client expects to receive and a server expects to provide, while *reality regions* represent observed levels of service
  - Regions are defined by *predicates* over system condition objects
  - *Transitions* specify behavior to trigger when the active regions change
- **System condition objects** are used to measure and control QoS
  - Provide interfaces to system resources, client and object expectations, mechanisms, managers, and specialized ORB functions
  - Changes in system condition objects observed by contracts can cause region transitions
  - Methods on system condition objects can be used to access QoS controls provided by resources, mechanisms, managers, and ORBs
- **Delegates** provide local QoS state for remote objects
  - Upon method call/return, delegate can check the current contract state and choose behavior based upon the current state of QoS
  - For example, delegate can choose between alternate methods, alternate remote object bindings, perform local processing of data, or simply pass the method call or return through

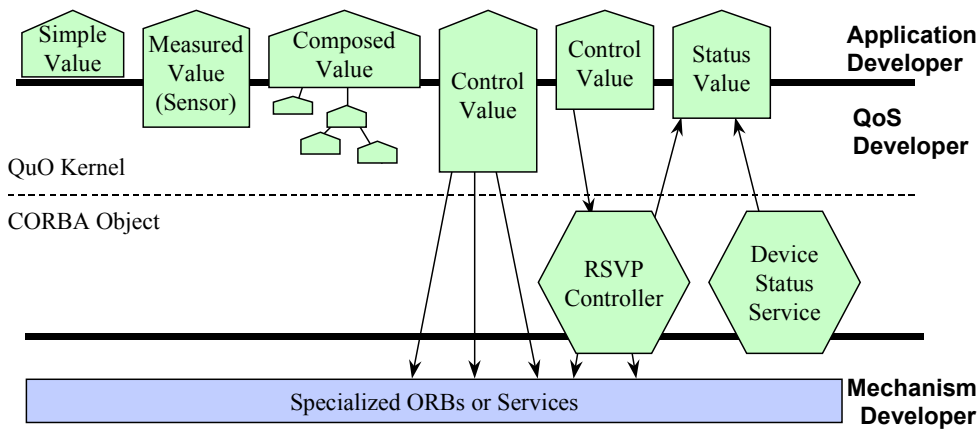
QuO Adds Specification, Measurement, and Adaptation into the Distributed Object Model



Measurement in QuO

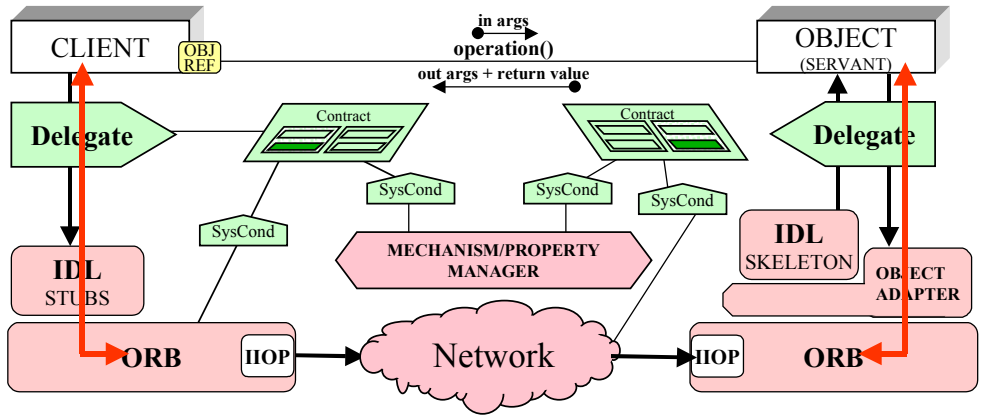
- *In-band* measurement handled by instrumentation
  - A structure is transparently passed along with the method call/return
  - Information can be inserted, read, and processed to record and evaluate method call statistics (e.g., the time spent in marshalling)
- *Out-of-band* measurement provided by system condition objects

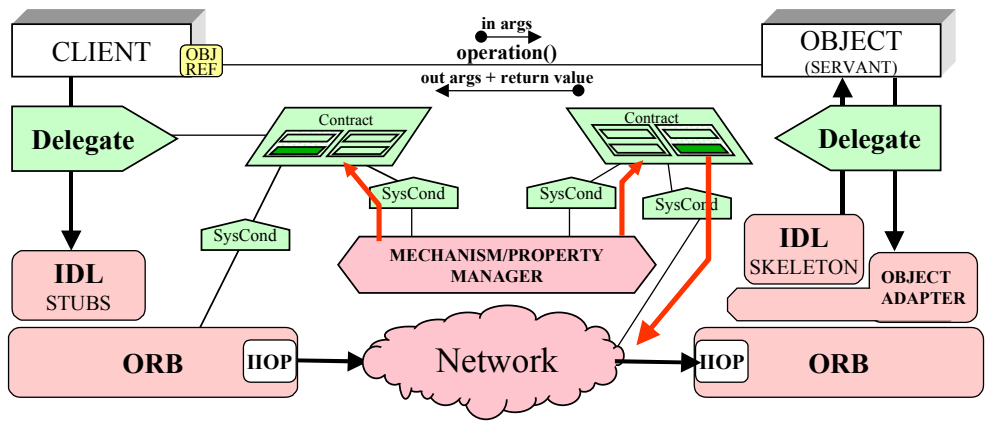




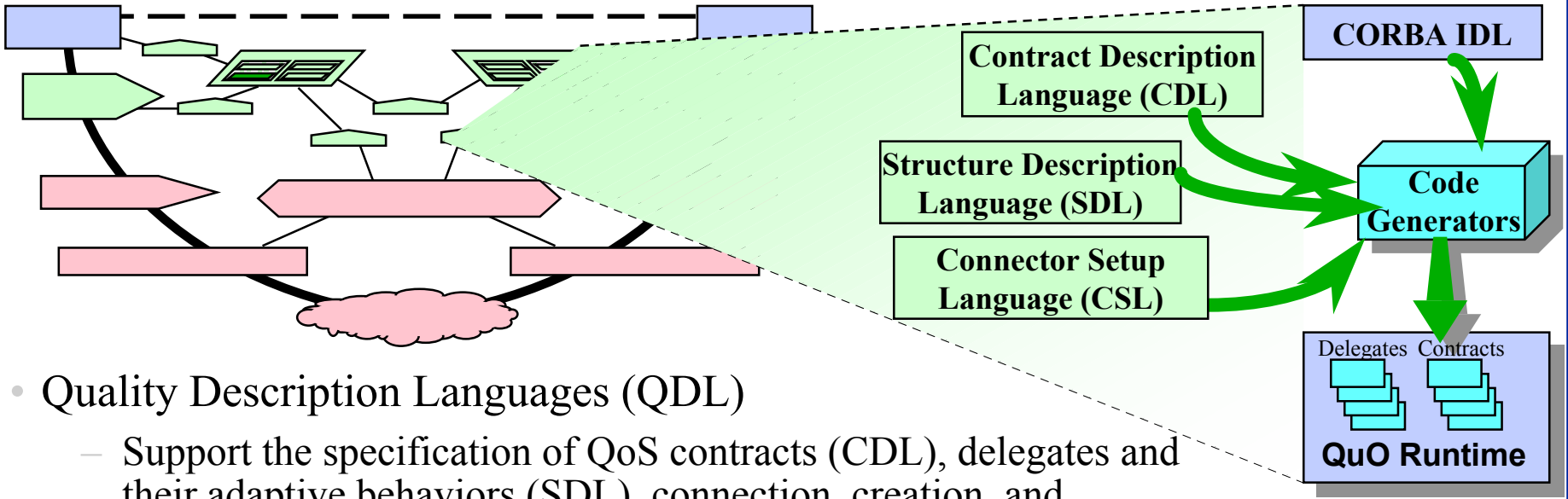
Adaptation and Control in QuO

- *In-band* adaptation provided by the delegate and gateway
  - A delegate decides what to do with a method call or return based upon the state of its contract
  - Gateway enables control and adaptation at the transport layer
- *Out-of-band* adaptation triggered by transitions in contract regions
  - Caused by changes in the system observed by system condition objects



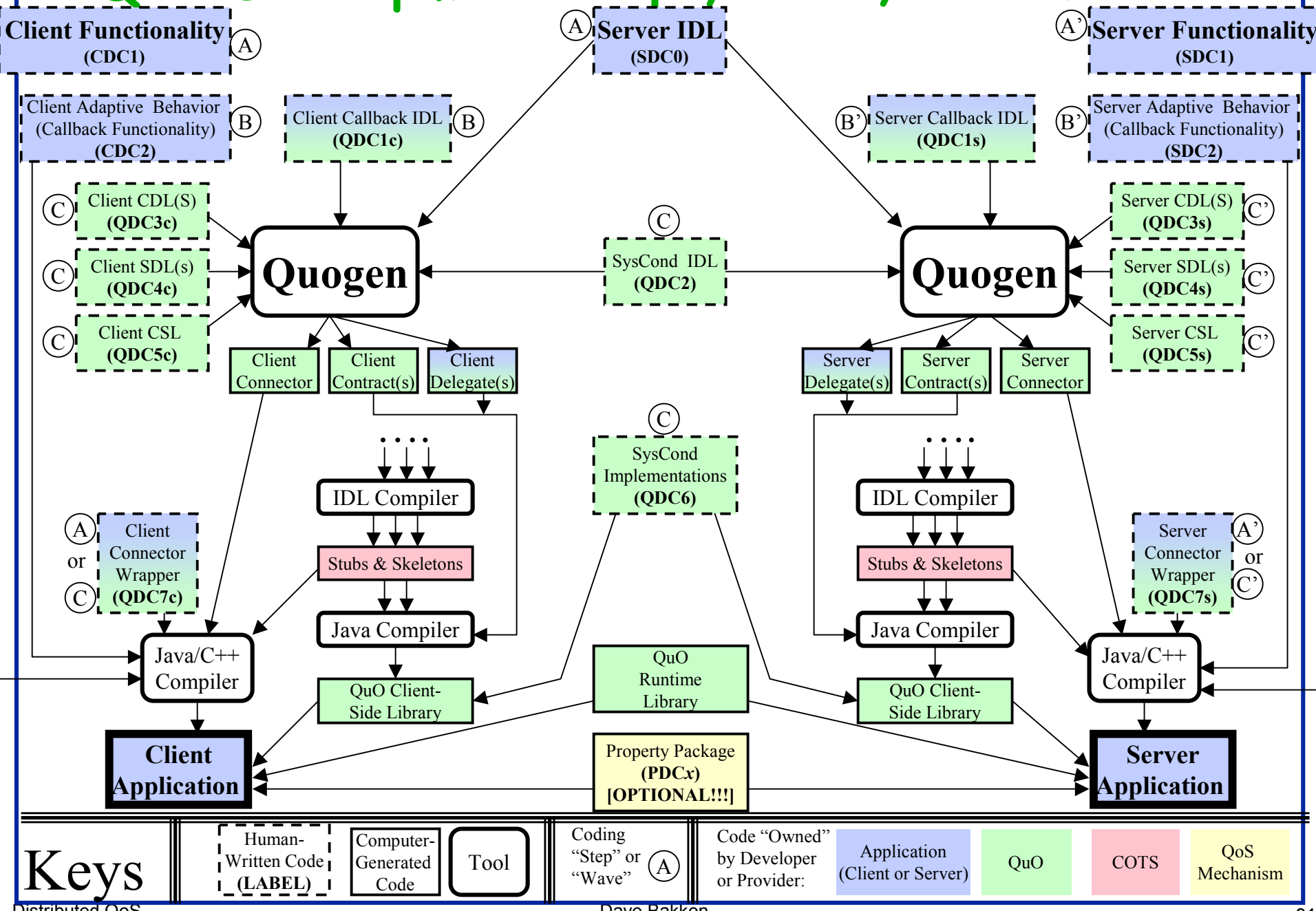


The QuO Toolkit Provides Tools for Building QuO applications



- Quality Description Languages (QDL)
  - Support the specification of QoS contracts (CDL), delegates and their adaptive behaviors (SDL), connection, creation, and initialization of QuO application components (ConnDL)
  - QuO includes code generators that parse QDL descriptions and generates Java and C++ code for contracts, delegates, creation, and initialization
- QuO Runtime Kernel
  - Contract evaluator
  - Factory object which instantiates contract and system condition objects
- System Condition Objects, implemented as CORBA objects

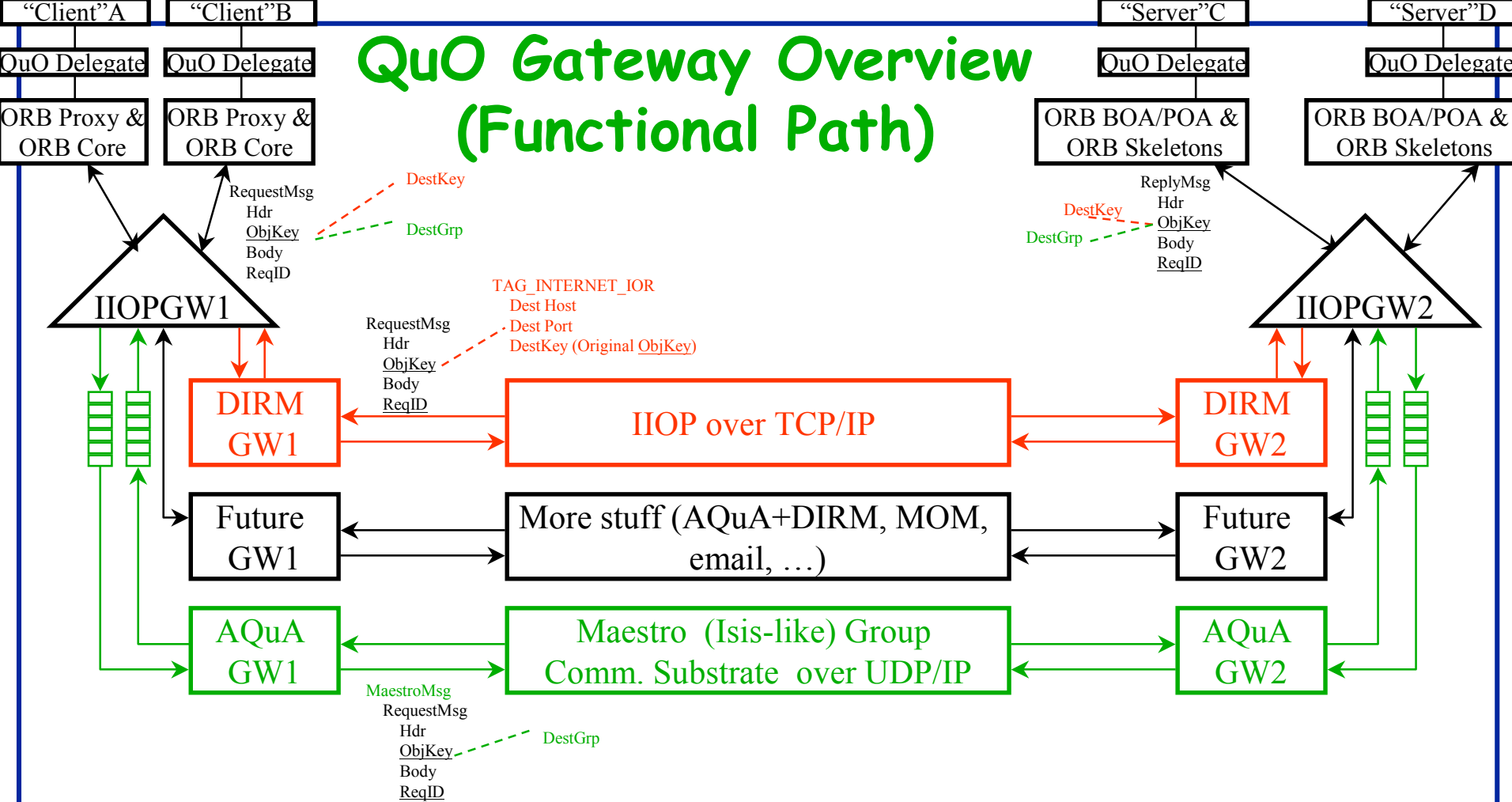
QuO Development Steps, Tools, and Modules



The QuO Gateway Manages IIOP Connections and Interfaces to Protocols which Manage QoS

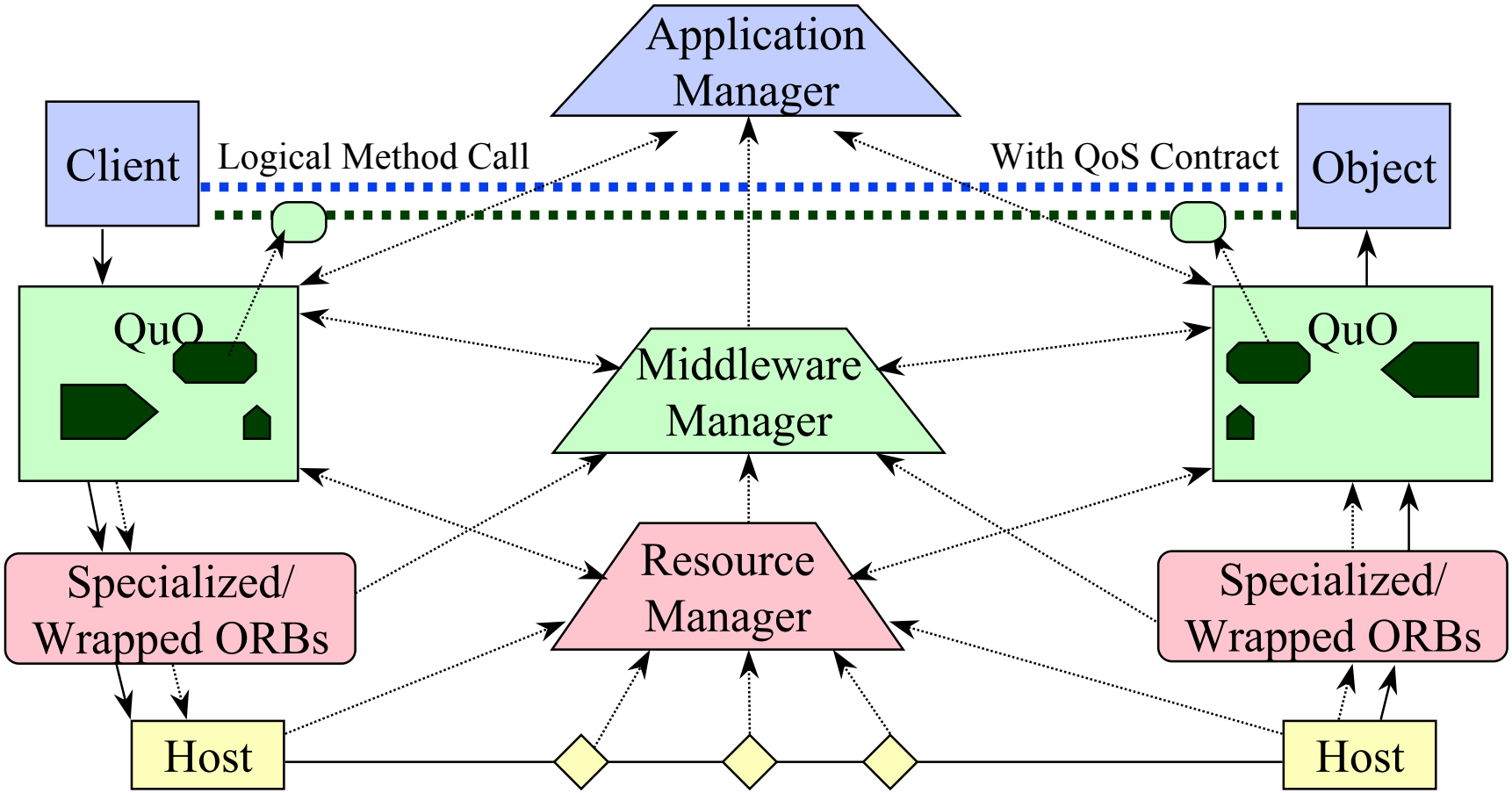
- To the “Client” ORB, the QuO Gateway looks like the object
- To the “Server” ORB, the QuO Gateway looks like a client
- The two ends of the gateway are on the same LAN as the Client/Object and may be on the same host
- CORBA Objects are used to Control QuO Gateway halves, but do not touch in-band communication
  - Different for AQuA and DIRM, later some merging will occur...

QuO Gateway Overview (Functional Path)



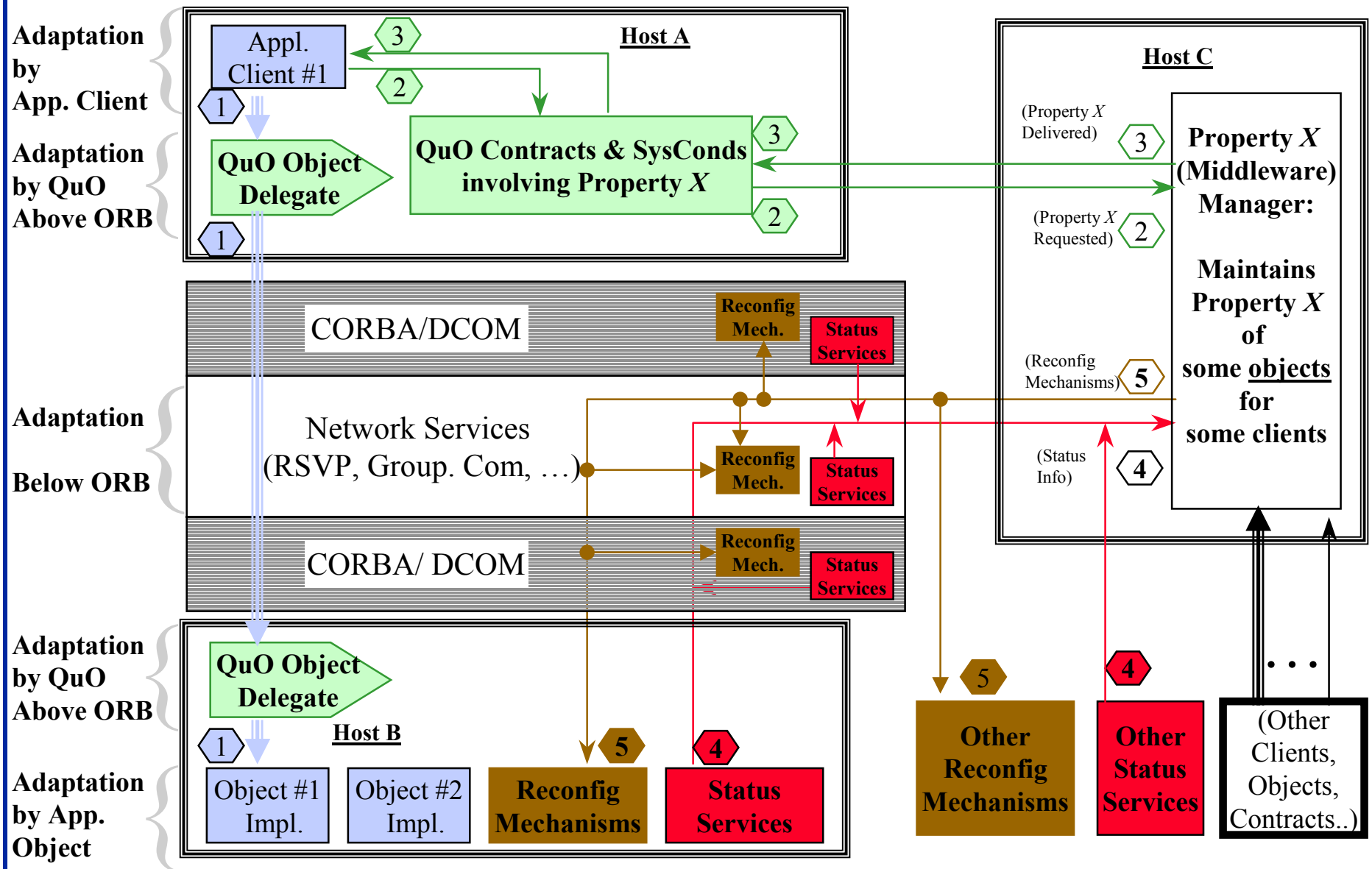
- We can (and do! and must!) rewrite ObjKey and ReqID; we just have to restore them when we pass them back to the appropriate ORB so it can use them to demux the reply, lest the poor ORB choke on it….
- Mappings between {Process,Host} and GWs is flexible (~TBD):
  - DIRM may want one per LAN/cluster to aggregate bandwidth
  - AQuA may want one per client (replica) process or even every delegate/contract inside it
  - OIT/Survivability/Security will have other constraints/preferences no doubt...
- Some Naming issues to be resolved to describe the exact flexibility; mainly engineering issues with no show stoppers
- Many research issues regarding the implications of different GW mappings on availability and performance/scalability

Layers of Managers Integrate Adaptation Policies at Different Levels & from Different Sources



- Functional Info (solid line) and "QoS meta-data" (dashed line)
- Translation between Manager Layers
- Centralized view vs. edge view
- Note: above is logical view, sometimes layers are merged…

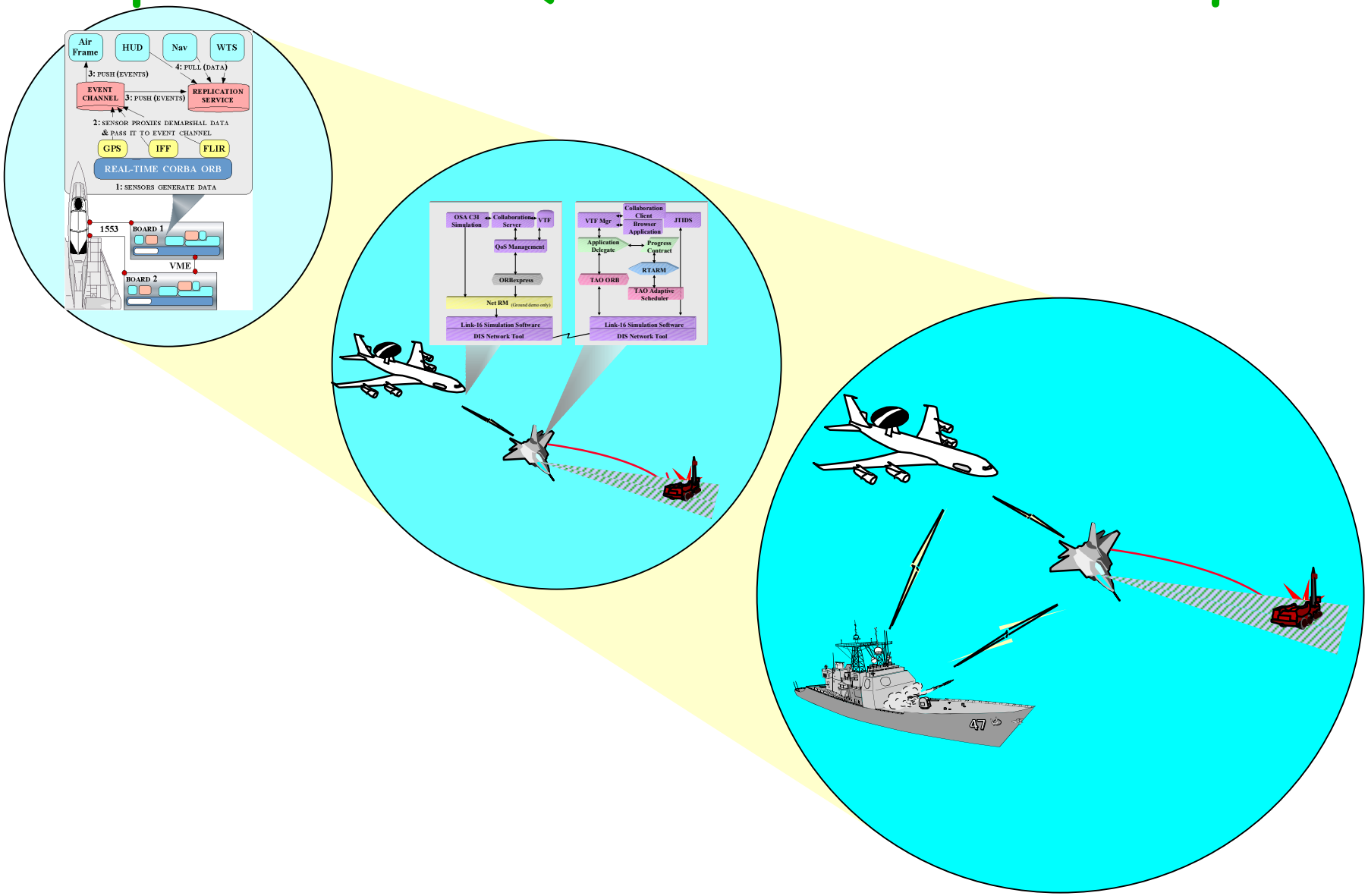
Canonical QuO Architecture for Generic Property Package *X*



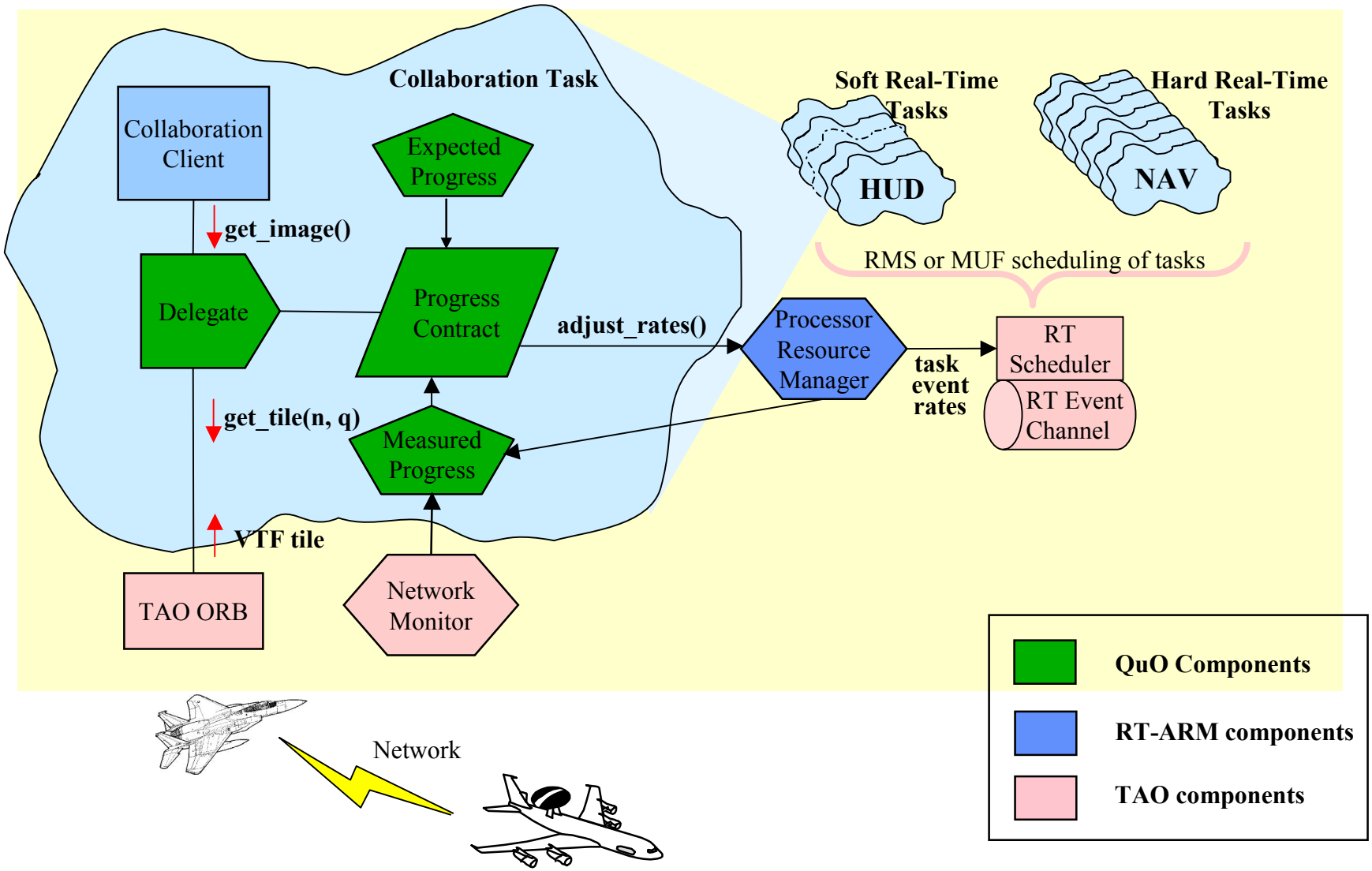
Outline of Talk

- QoS: The problem, and basic definitions
- QoS Implementation Issues
- Quality Objects (QuO) 2.0 Architecture
- **QuO 2.0 Case Study: WSOA**
- QuO 3.0: Aspects and Reuse
- QuO 3.0 Case Study: UAV Multimedia adaptivity

Multiple Levels of QuO Coordination are Required!

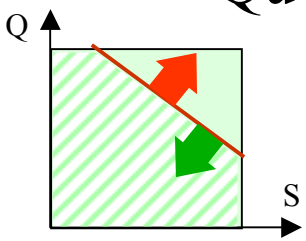


Adaptive Behavior Integrated with Advanced Resource Management in WSOA



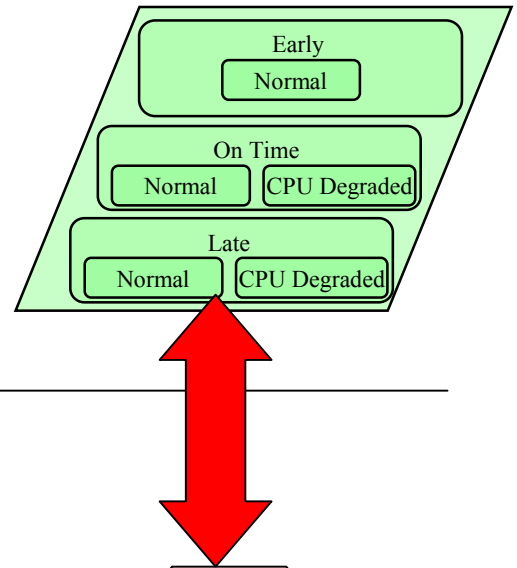
WSOA QoS Control Flow

QuO

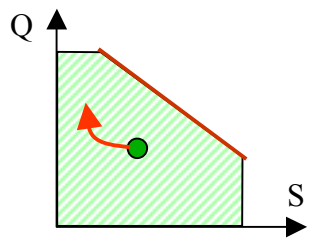


Contraction, Expansion

- Manages application progress
  - Early, On-Time, or Late for each operation
- Defines operating regions
  - Range of rates for each operation
- Also handles image tiling (not shown)

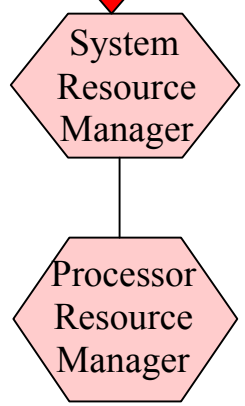


RT-ARM



Feedback Adaptation

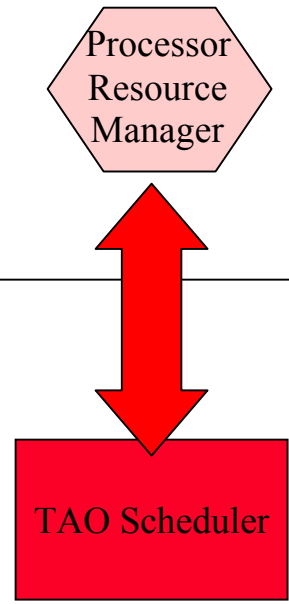
- Manages QoS parameters within the given operating regions
  - Adjust rates within defined ranges for each operation
- Reports when operating region is violated (or will be violated)



WSOA QoS Control Flow (cont'd)

RT-ARM

- Adjusts current available dispatch rate ranges for each operation
- Provides admission control policy
- Queries TAO Scheduler for monitored execution time results



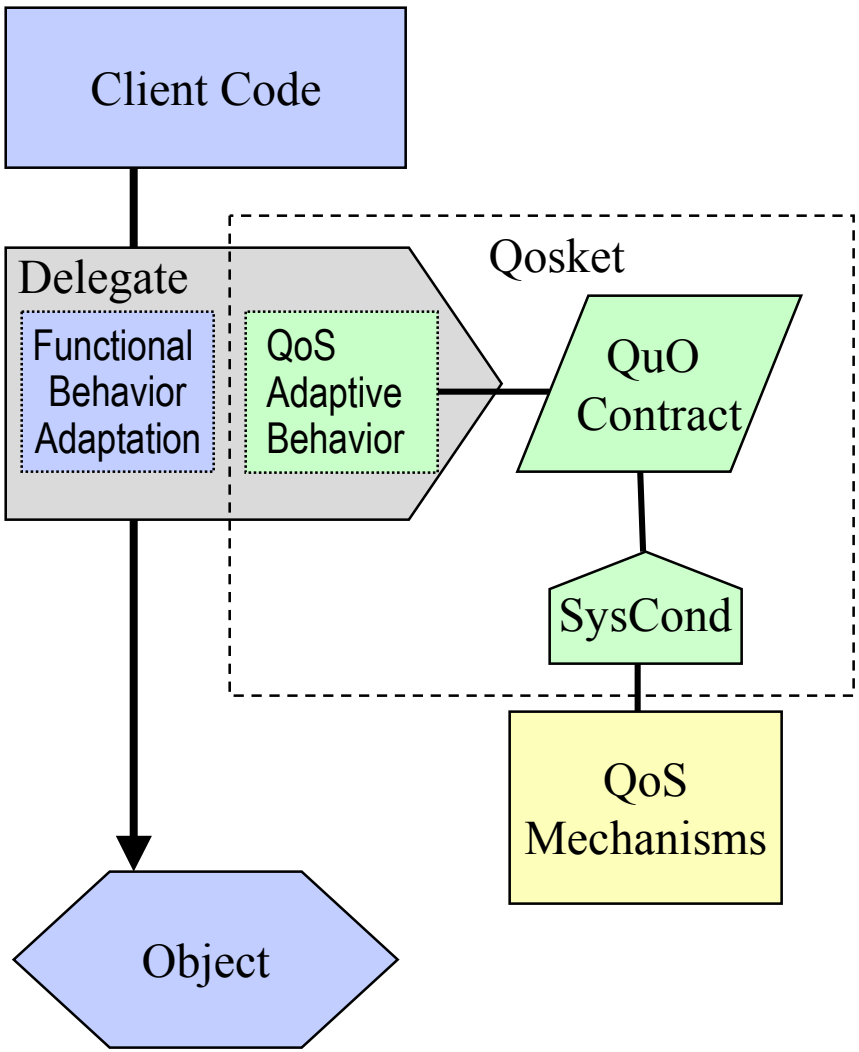
TAO Scheduler

- Binds specific rate according to RT-ARM supplied admission control policy
- Queues operations and enforces hybrid static/dynamic scheduling policy
- Makes available to RT-ARM the actual execution times of each scheduled operation

Outline of Talk

- QoS: The problem, and basic definitions
- QoS Implementation Issues
- Quality Objects (QuO) 2.0 Architecture
- QuO 2.0 Case Study
- **QuO 3.0: Aspects, Reuse, and Status Info**
  - **Provided courtesy of BBN (after I left for WSU)**
- QuO 3.0 Case Study: UAV Multimedia adaptivity

QuO 3.0 Components Are Packaged into Reusable Bundles of "Systemic Behavior" Called Qoskets

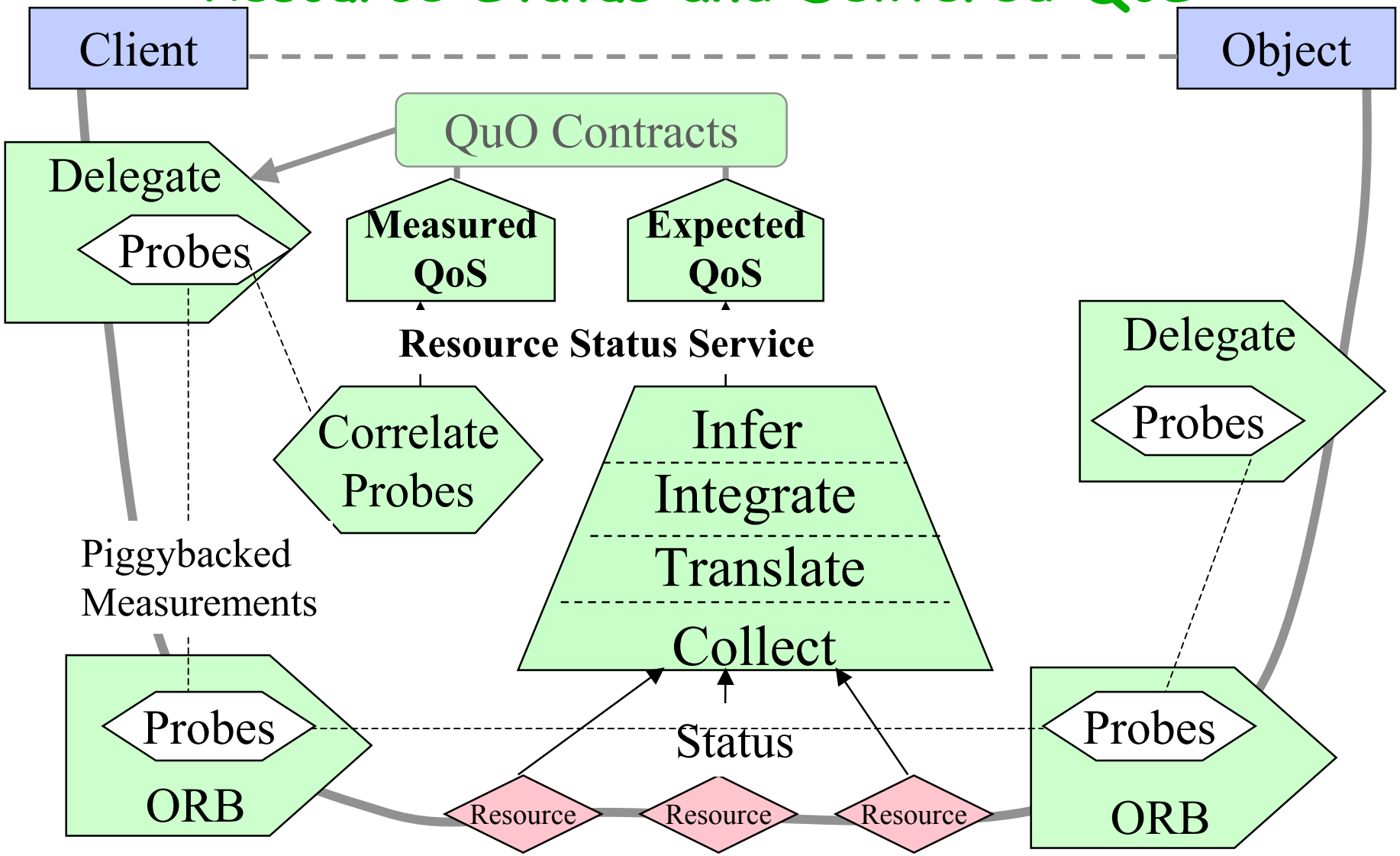


- The Qosket encapsulates a set of contracts (CDL), system condition objects (IDL), and QoS adaptive behavior (ASL)
- The Qosket exposes interfaces to access QuO controls and information (specified in IDL)
- The Qosket separates the functional adaptive behavior (business logic) from the QoS adaptive behavior and the middleware controls from the QoS mechanisms
- Greatly augmented and strengthened QuO 2.0 CSL

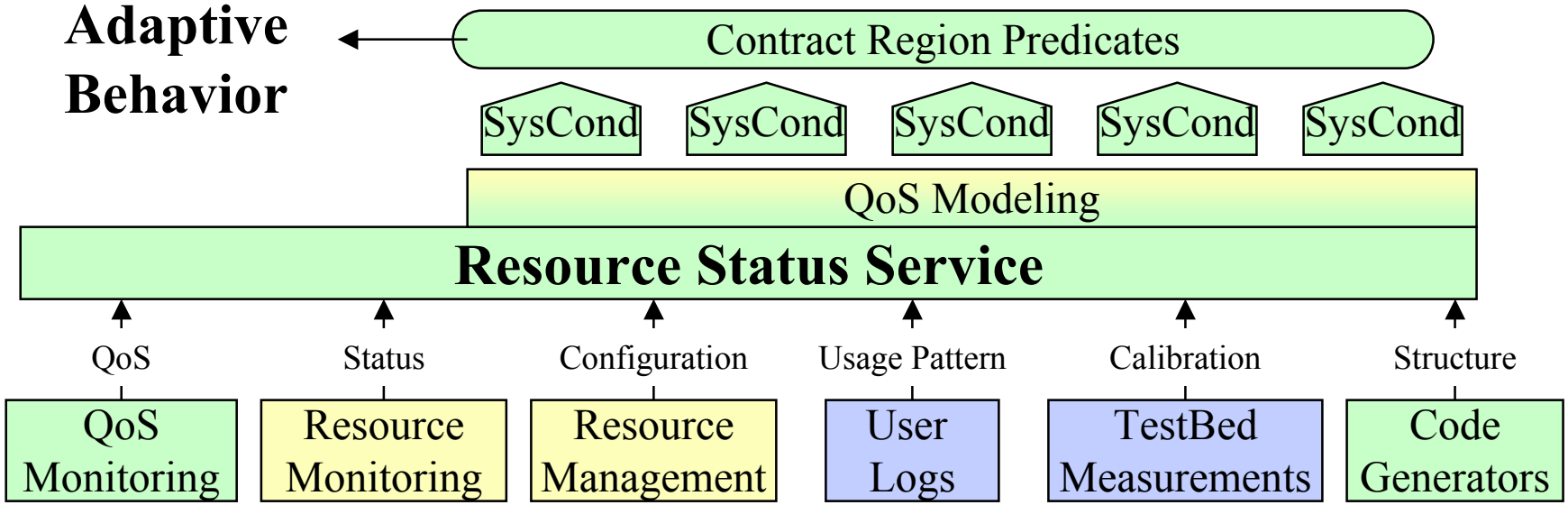
QuO 3.0 Aspects

- QuO provides hooks for different aspects specialized for application-level adaptivity
- Well-defined code join points in delegates
  - Pre-call to method
  - Post-call to method
  - After return from method
  - Extended and simplified QuO 2.0 SDL, made more regular, …

QuO 3.0 Resource Status Service (RSS) is an Integration Base for Observing Resource Status and Delivered QoS

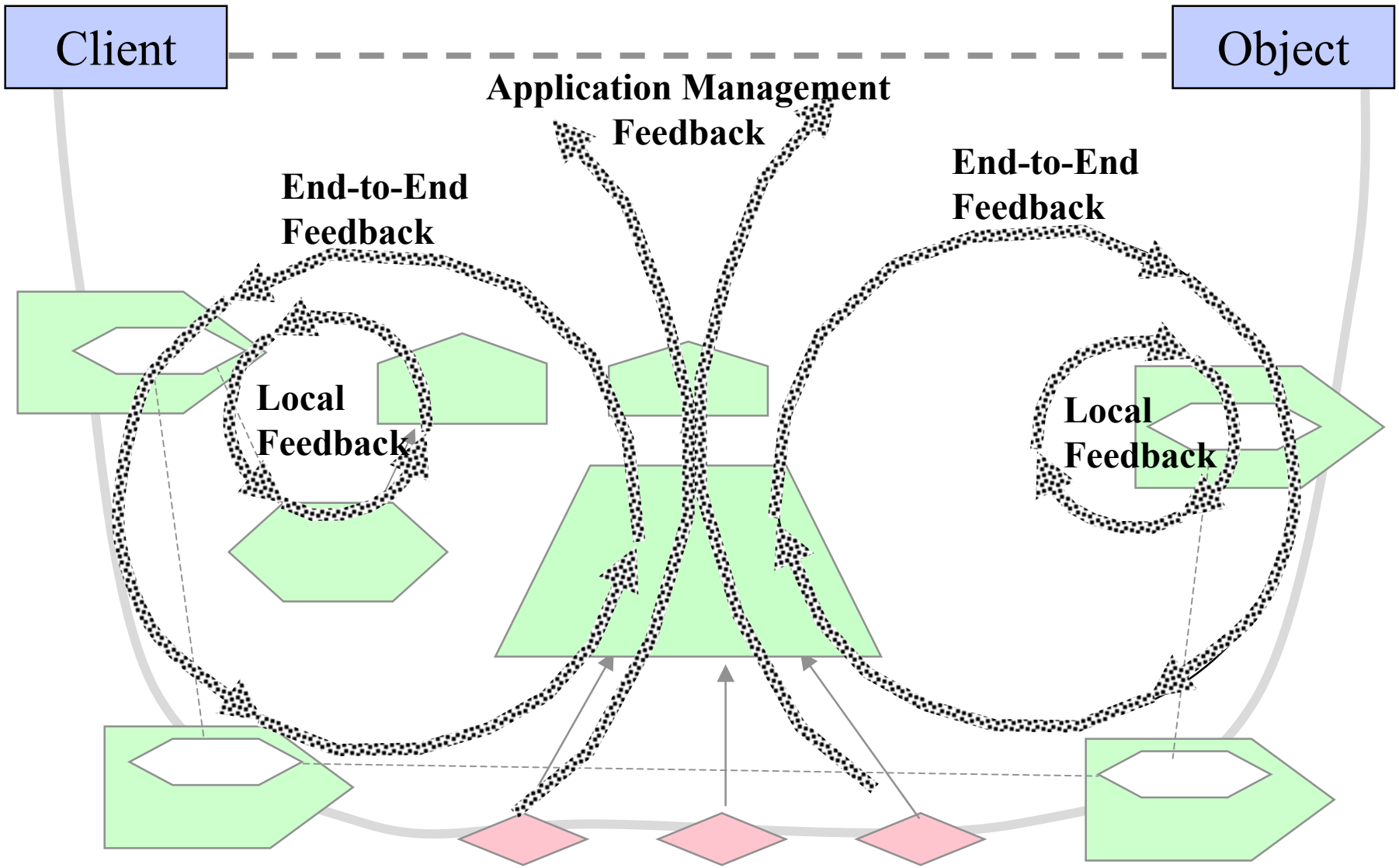


RSS Merges Off-line Analyses, Resource Monitors, and Application Measurements

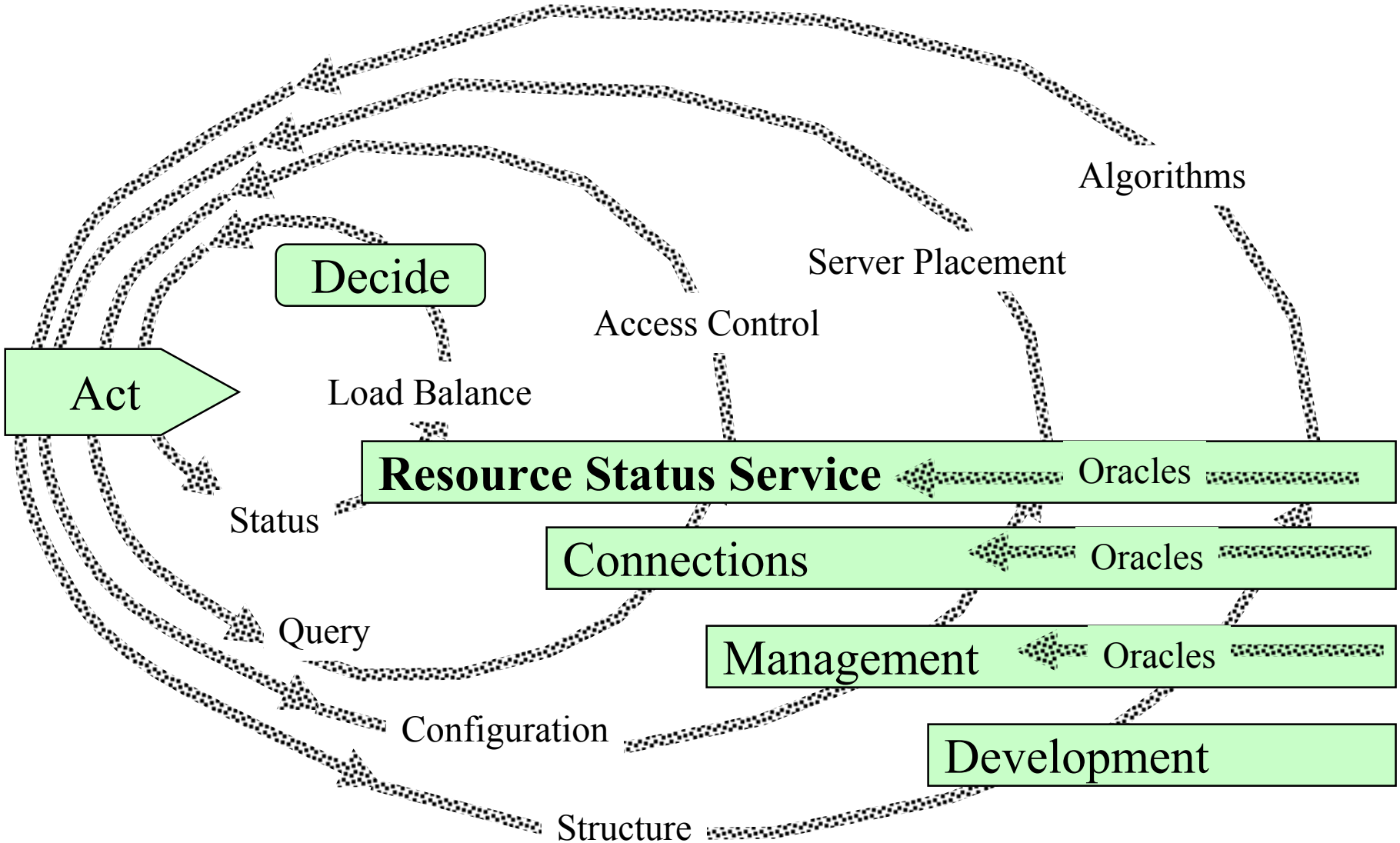


- Data is from many **sources**
- Data is from many **time horizons**
- **Common representations** are used to simplify the QoS Modeling
- **Collection and Integration** Details are hidden from QoS Modeling

RSS Helps Integrate Feedback Information from Different Locations



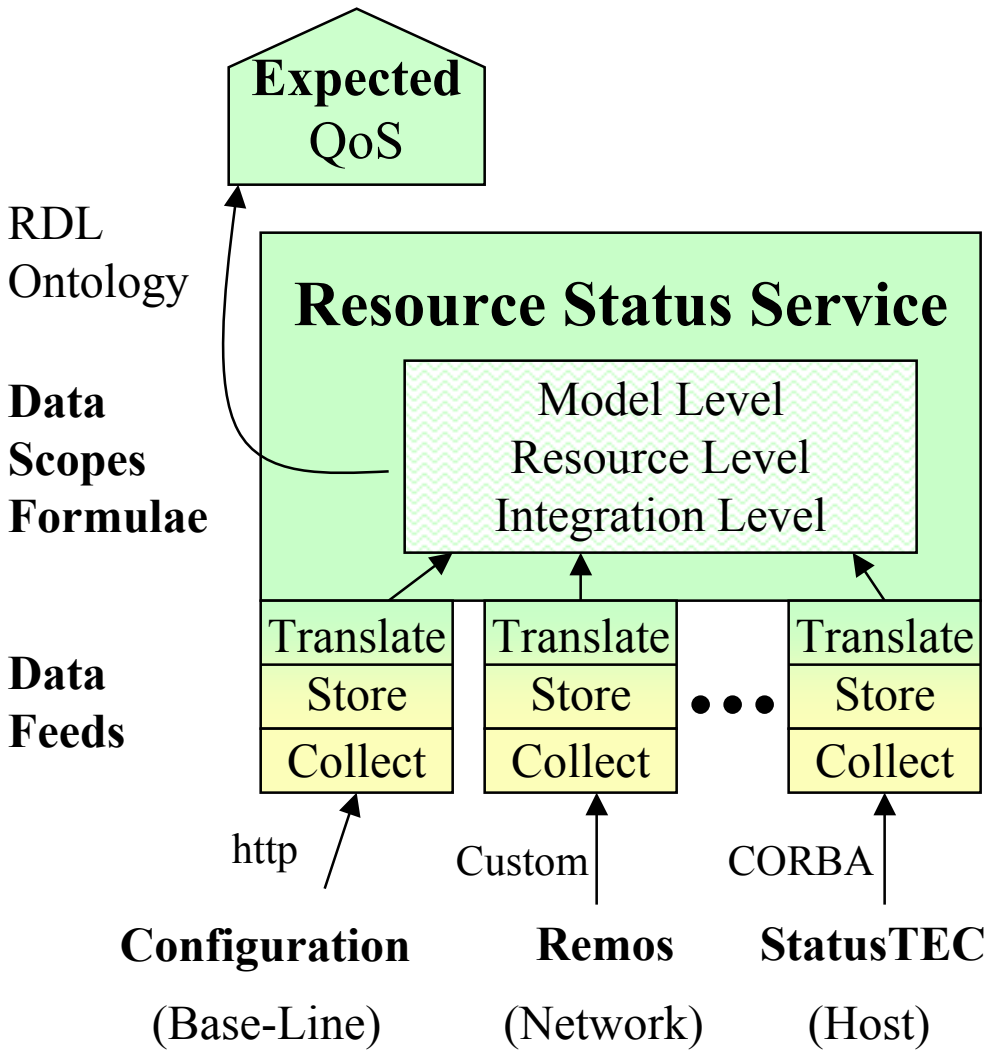
RSS Helps Integrate Feedback Information from Different Time Scales



Faster-Precise-Smaller

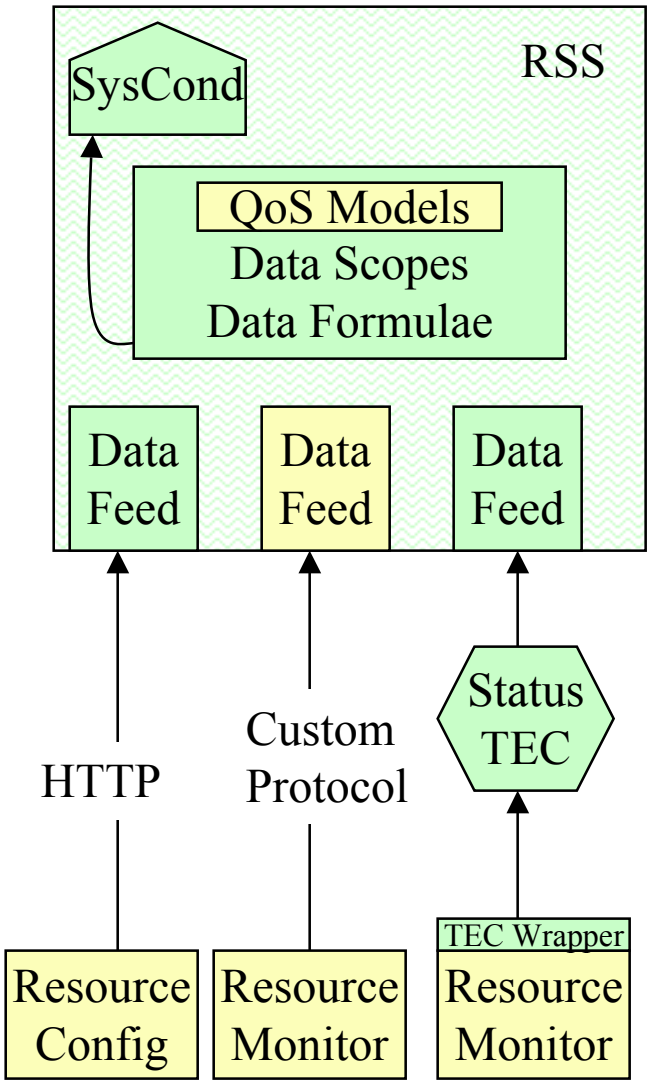
Slower-Broad-Bigger

QuO 3.0 has a Resource Status Service (RSS) Built into the QuO Java Kernel



- **Isolation:** The Quosketeer will work with a high level description of available resources
- **Integration:** Conflicting measurements will be resolved to always give the best guess.
- **Translation:** different standards for Resources MIBs will be translated into a QuO Resource Ontology
- **Collection:** interfacing details will be handled by Data Feed

New Integration Points were Added to QUO V3.0 to Manage Resource Information

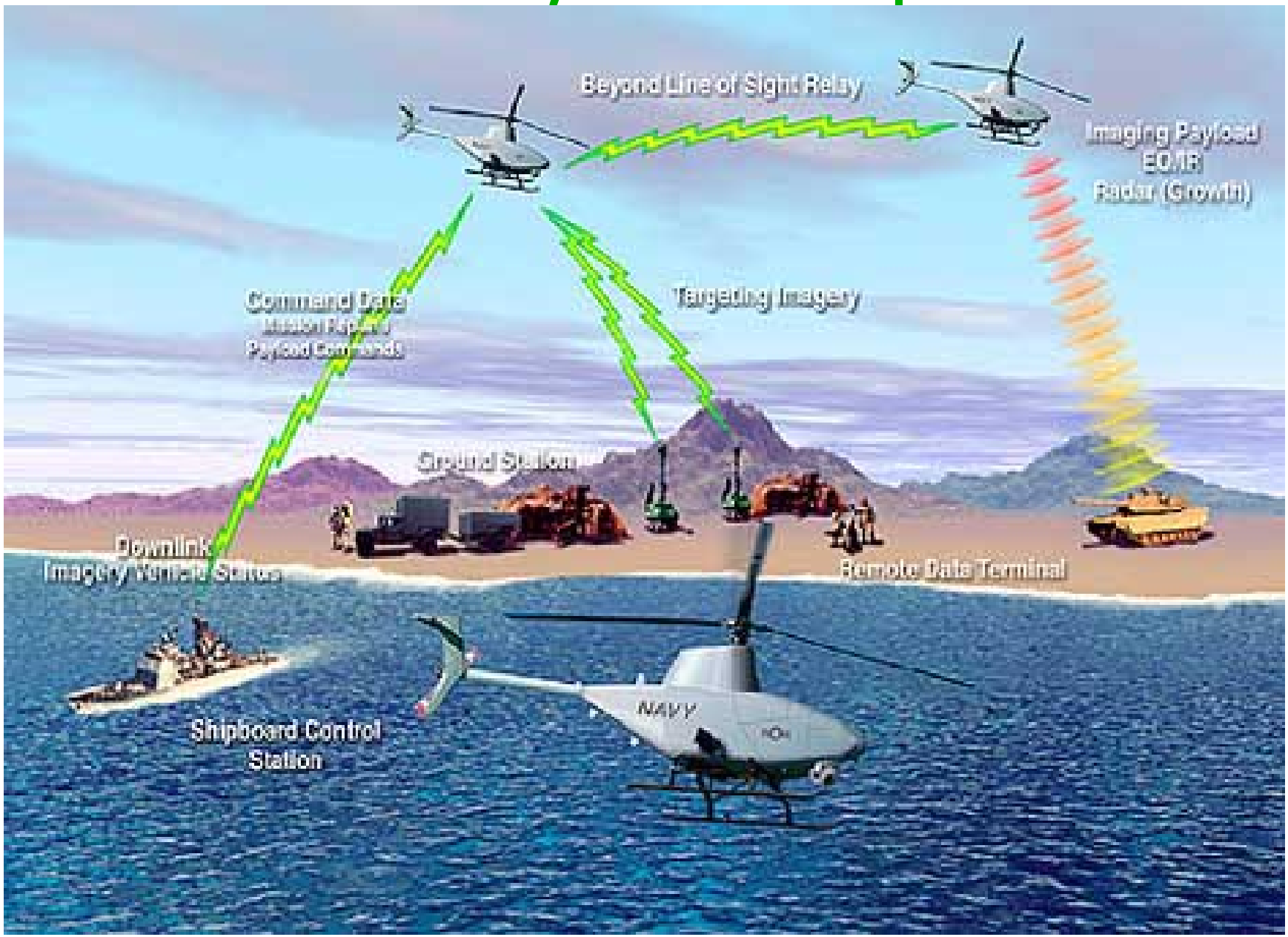


- **QoS Model** can be built using a high-level representation of the underlying Resources and their structure
- **Data Feeds** interface to the custom protocols used by distributed resource managers.
- **Status TEC** uses Push technology to publish resource status information without knowledge of the consumers or distribution channel
- **Static Information** can be published via web pages, but in a QuO specific format.

Outline of Talk

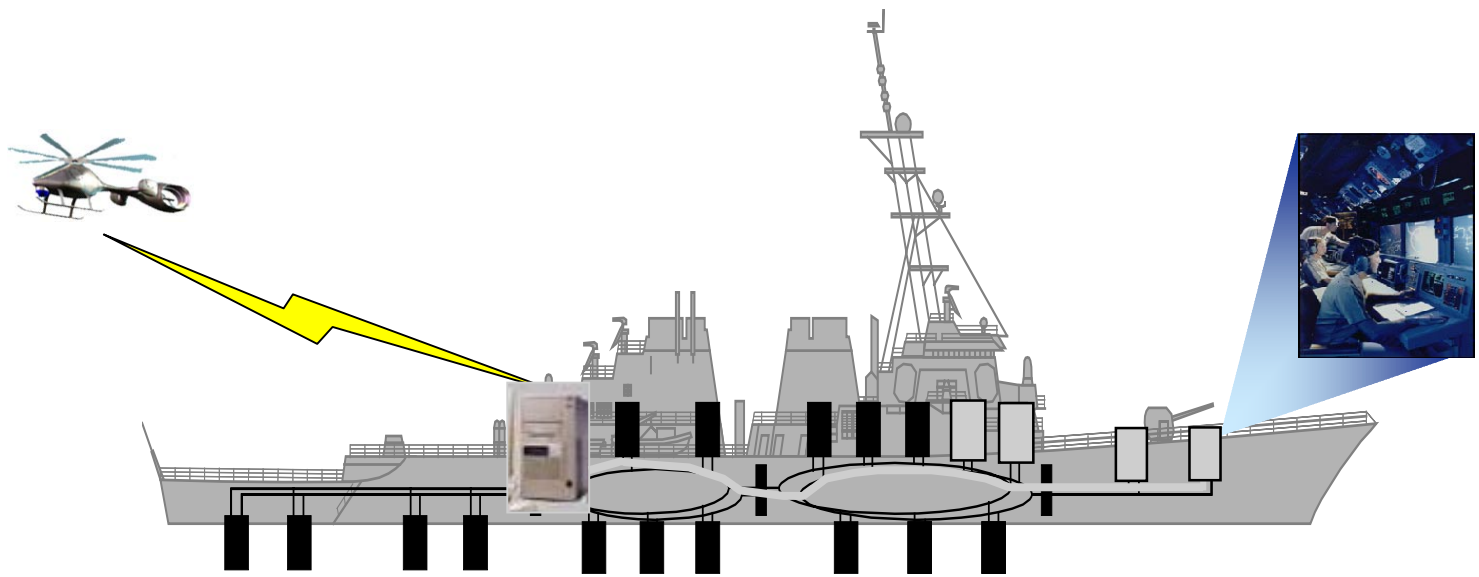
- QoS: The problem, and basic definitions
- QoS Implementation Issues
- Quality Objects (QuO) 2.0 Architecture
- QuO 2.0 Case Study
- QuO 3.0: Aspects and Reuse
- **QuO 3.0 Case Study: UAV Multimedia Adaptivity**
  - **Provided courtesty of BBN (after I left for WSU)**

US Navy UAV Concept



*From the World Wide Web site for official information about Unmanned Aerial Vehicles, http://uav.navair.navy.mil/home.htm

UAV Basic Scenario



- Video feed from off-board source (unmanned aerial vehicle)
- Video Distributor sends video to hosts on ship's network
- Users' hosts receive video and display it
- Users interact with UAV in real time

Variations & Adaptations in UAV Scenario

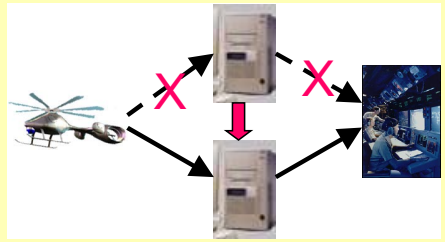
Mission requirements of UAV scenario

Timeliness
- Maintain an out-of-the-window view of UAV imagery

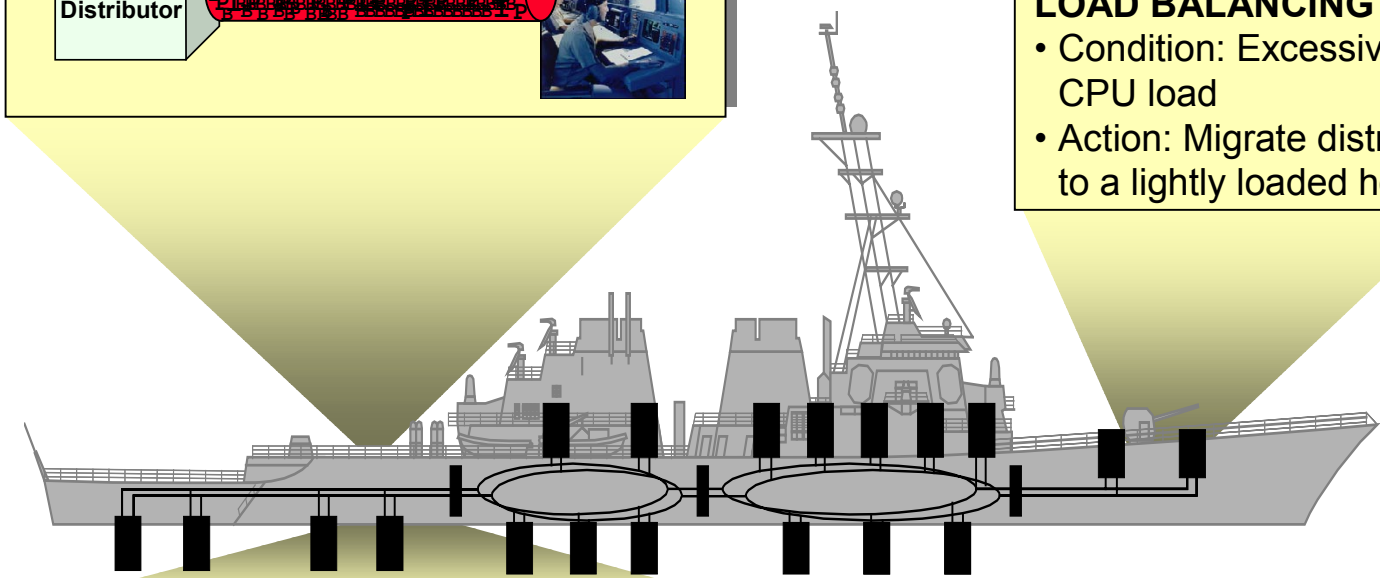
Importance
- Frames must be dropped in reverse order of importance (B, then P)

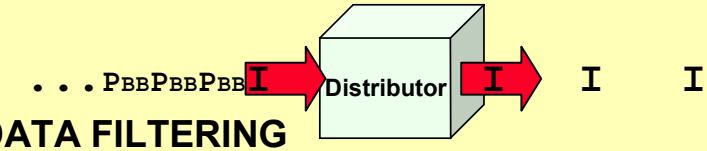
Fidelity
- Highest fidelity frames (i.e., I frames) must be delivered

NETWORK RESERVATION
- Condition: excessive Network load
- Action: Use IntServ to reserve bandwidth



LOAD BALANCING
- Condition: Excessive CPU load
- Action: Migrate distributor to a lightly loaded host

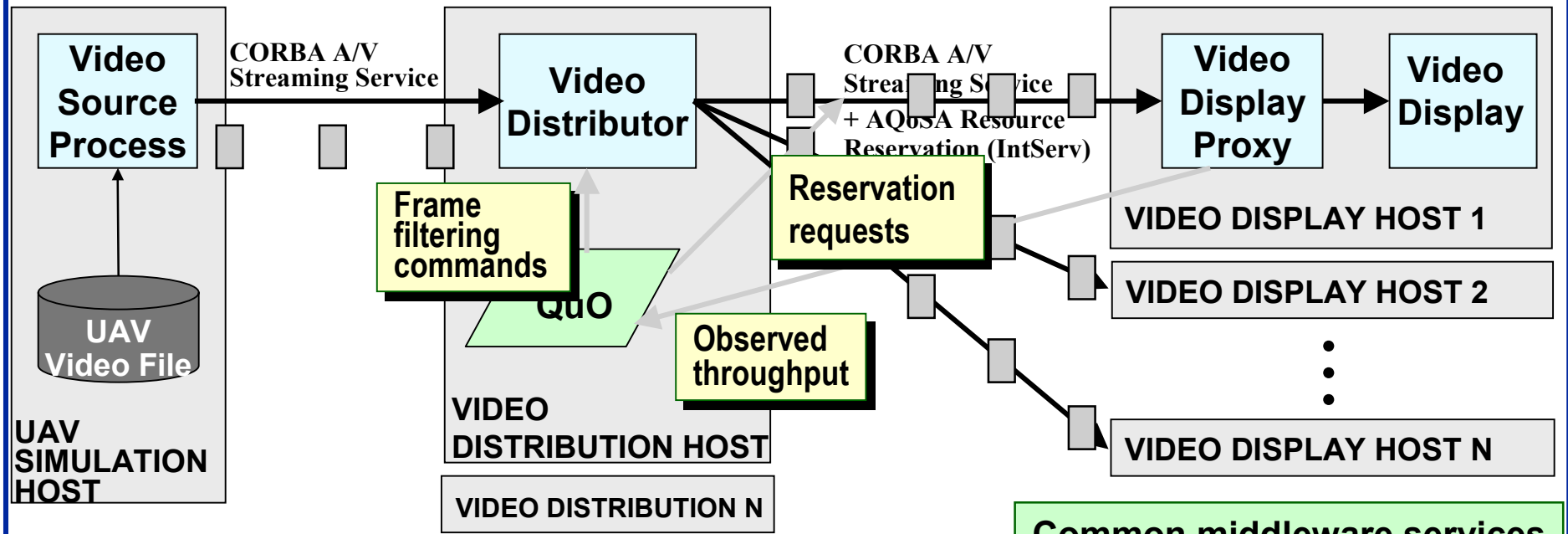




DATA FILTERING
- Condition: When excessive network or CPU load
- Action: Drop selective frames

Dynamic Variations in Operating Conditions

UAV Architecture



- Features

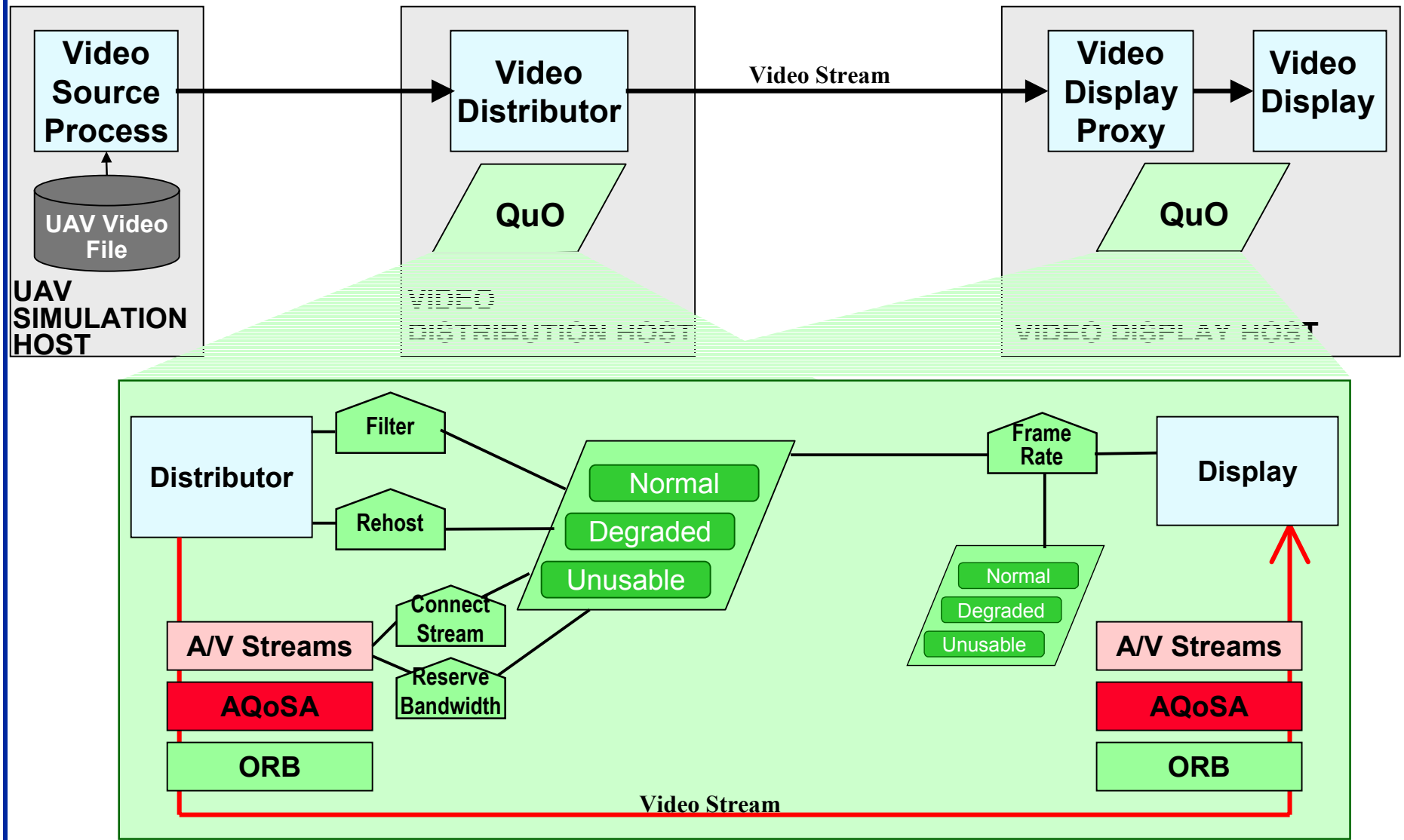
Reused Off the Shelf Software
- DVDView video player

Common middleware services
- QuO adaptive middleware
- Real-time CORBA ORB (TAO)
  - Naming Service
  - A/V Streaming Service
  - AQoSA

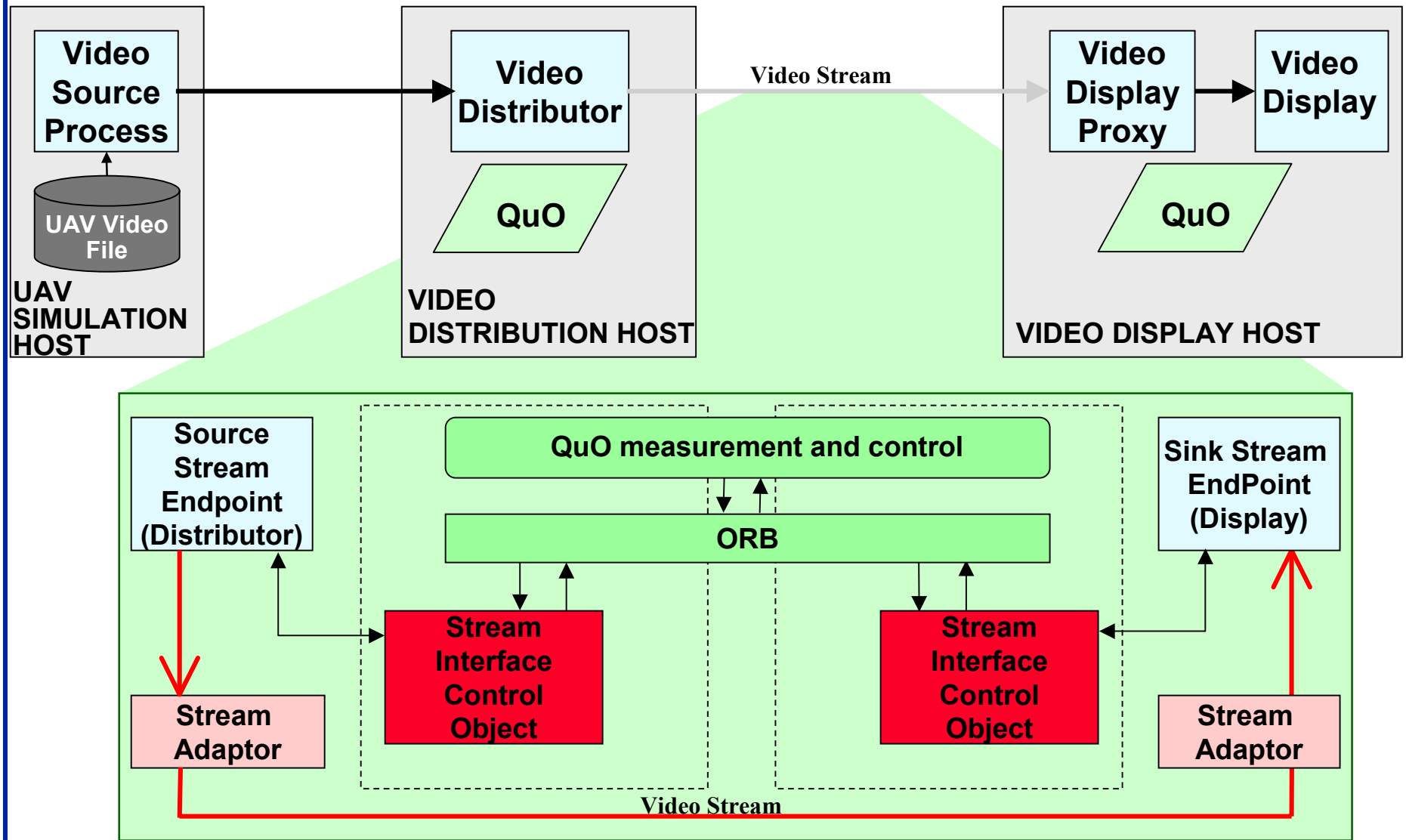
Hand coded functionality
- video forwarding
- frame processing, filtering
- timestamping and sequencing
- connection, video transport

Many of these are intertwined aspects

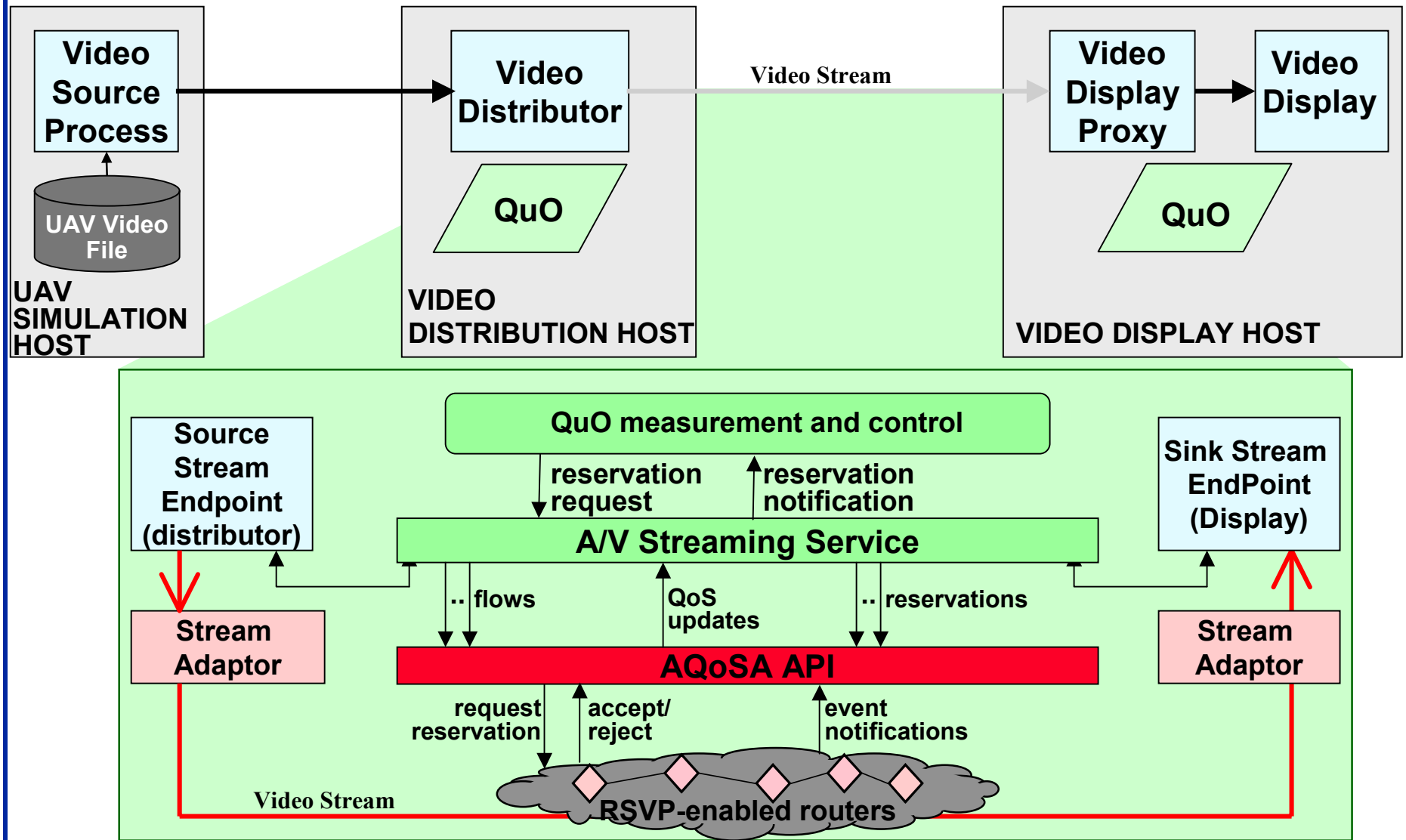
Management and adaptation in UAV using the QuO adaptive middleware



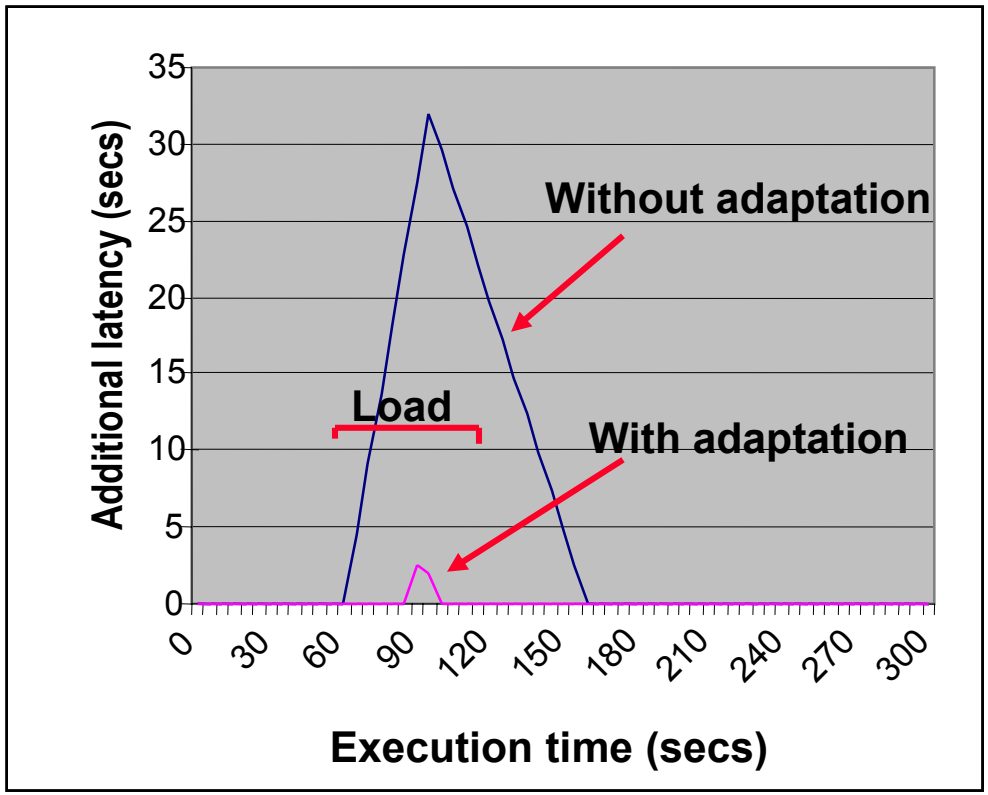
Connecting and managing UAV video streams using the CORBA A/V Streaming Service



Reserving network resources using the AQoSA API



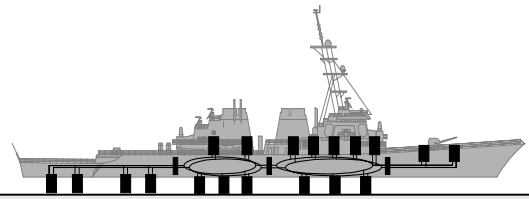
Ability to Keep UAV Video Current using Middleware-based Adaptation



- UAV running on 3 200 MHz PCs (Linux), 128 MB memory, TCP/IP
- Additional 60% CPU load introduced on second stage (3 processes requesting 20% load each) starting at approx. 60 secs, removed at approx. 120 secs.

Under load	Mean lateness	Max lateness
No adaptation	5.400 sec	32.696 sec
Adaptation	0.067 sec	1.930 sec

Functionality and aspects in the UAV application



Video Source Process

Base Functionality
- Read bytes from a file
- Convert into frames
- Send out pipe

Timing: periodic output of video frames (A)

Video Distributor Process

Base Functionality
- Receive bytes
- Convert into frames
- Send frames to registered receivers

Adding a timestamp (A)

Adding a sequence number (A)

Measuring throughput (A)

Measuring resource usage (CPU,NW) (A)

Adapting to changes (e.g., filtering frames, load balancing) (A)

Video Display Process

Base Functionality
- Receive bytes
- Convert into frames
- Display frames on the screen

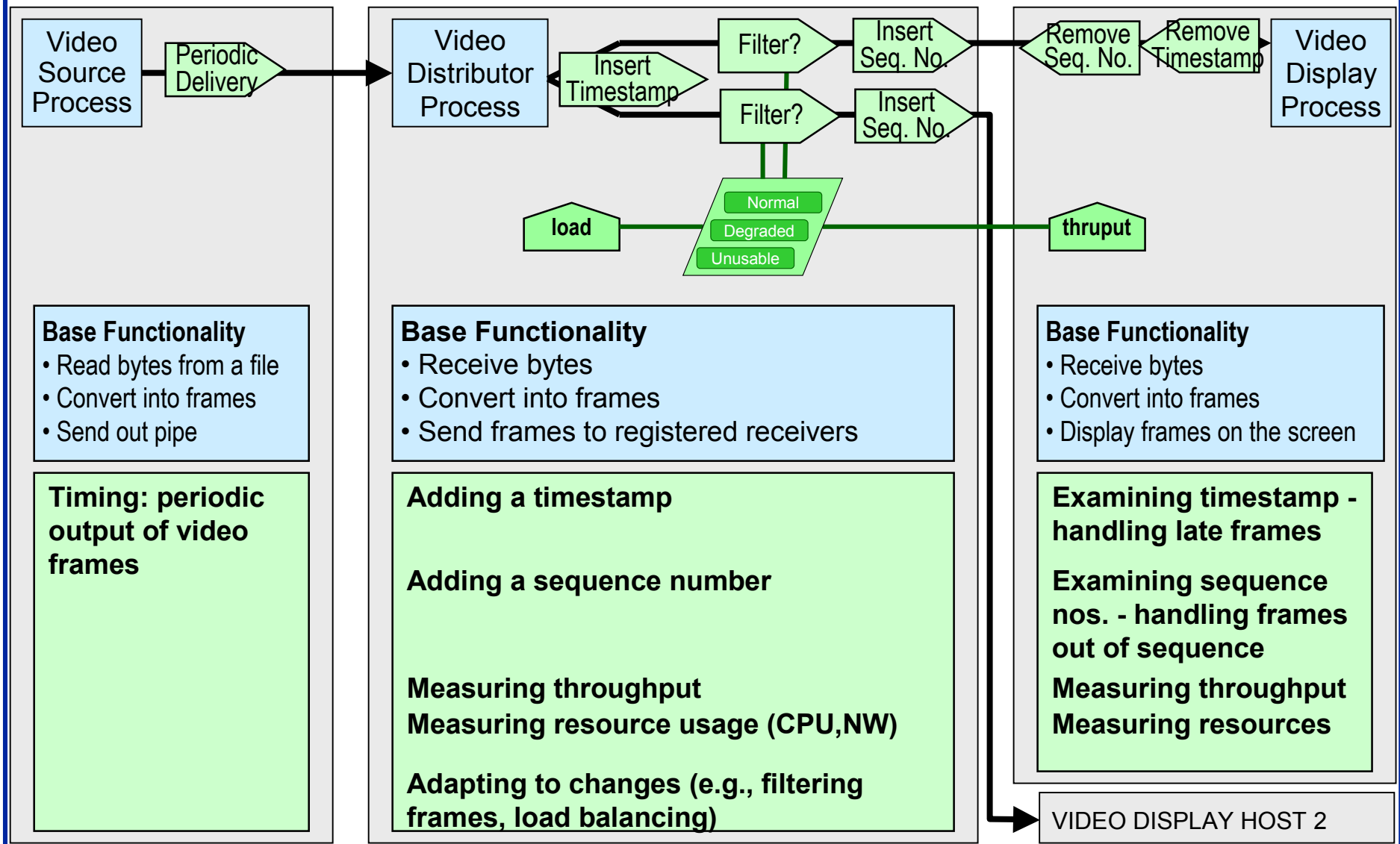
Examining timestamp - handling late frames

Examining sequence nos. - handling frames out of sequence

Measuring throughput

Measuring resources

Separating out Intertwined Aspects in the UAV Architecture



Specifying and Generating Code for QoS Aspects using QuO

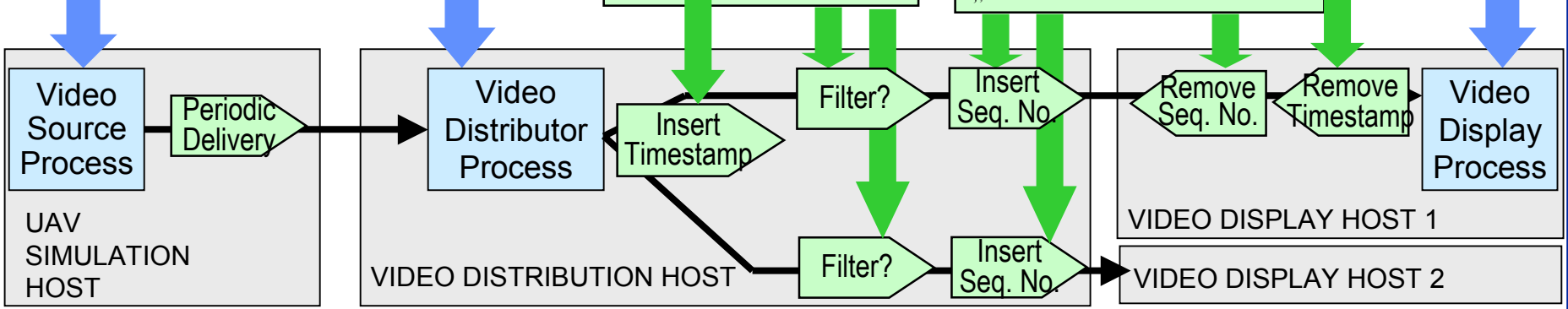
C++ Application Code

```
class Sender
{
public:
  /// Constructor
  Sender (void);
  /// Method to initialize the various data component
  int init (int argc,  char **argv, CORBA::Environm
  /// Method to pace and send data from a file.
  int pace_data (CORBA::Environment &);
  /// Accessor to the connection manager.
  Connection_Manager &connection_manager (voi
  /// Amount of debugging info to print out: 0 = non
  int debug_level_;
private:
  /// Method to parse the command line arguments.
  ...
}; ...
class Distributor
{
public:
  /// Constructor
  Distributor (void);
  /// Destructor.
  ~Distributor (void);
  /// Initialize data components.
  int init (int argc,  char **argv, CORBA::Environm
  /// Parse args.
  int parse_args (int argc,  char **argv);
  // Flag to know when we are done.
  int done (void) const;
  void done (int);
  /// Accessor to connection manager.
  Connection_Manager &connection_manager (voi
  /// Amount of debugging output to print out; 0 = n
  int debug_level_;
protected:
  /// Connection manager.
  ...
}; ...
```

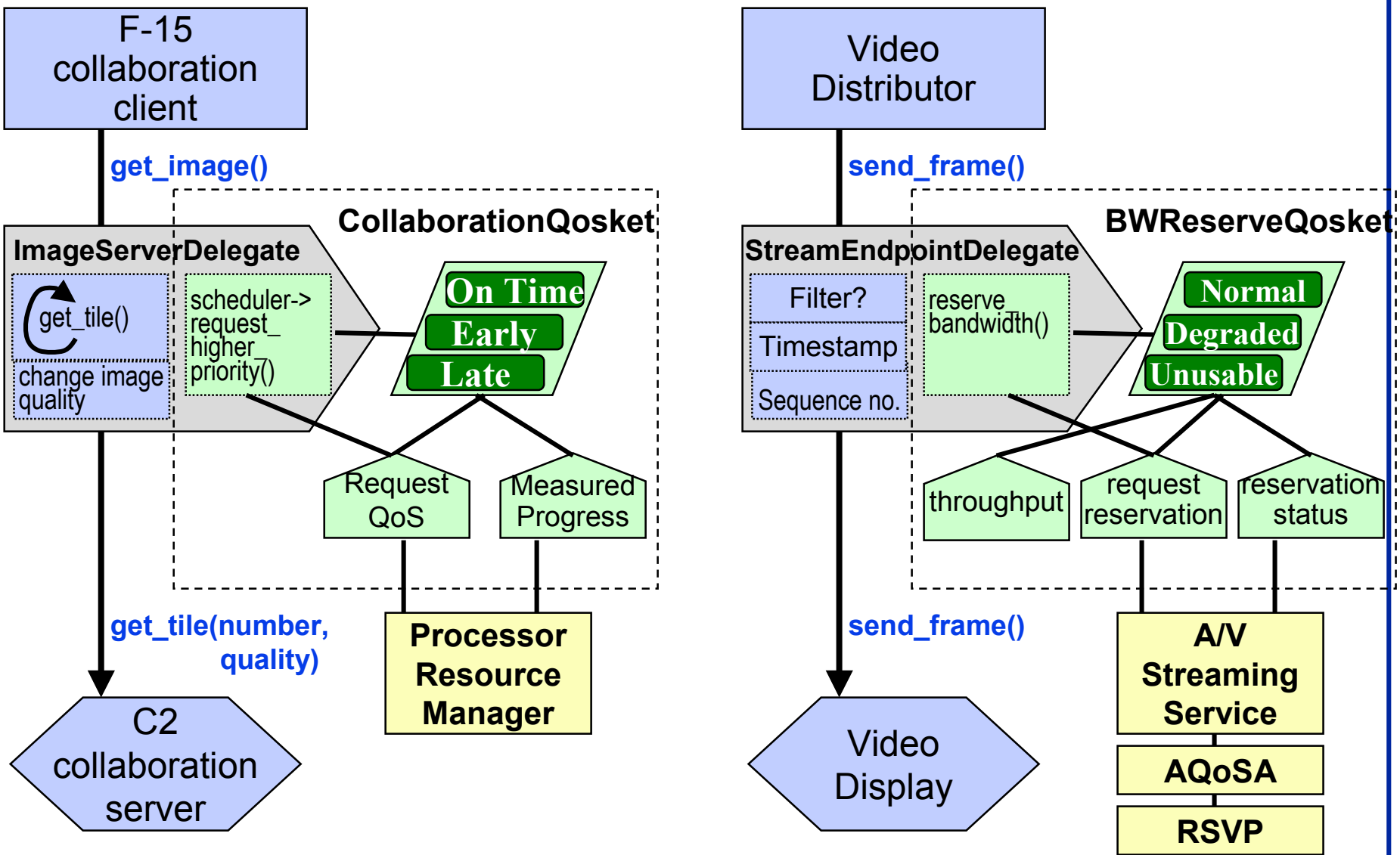
```
class Sender
{
public:
  /// Constructor
  Sender (void);
  /// Method to initialize the various data components.
  int init (int argc,  char **argv, CORBA::Environment&);
  /// Method to pace and send data from a file.
  int pace_data (CORBA::Environment &);
  /// Accessor to the connection manager.
  Connection_Manager &connection_manager (void);
  /// Amount of debugging info to print out: 0 = none, 10 = lots
  int debug_level_;
private:
  /// Method to parse the command line arguments.
  ...
}; ...
class Distributor
{
public:
  /// Constructor
  Distributor (void);
  /// Destructor.
  ~Distributor (void);
  /// Initialize data components.
  int init (int argc,  char **argv, CORBA::Environment &);
  /// Parse args.
  int parse_args (int argc,  char **argv);
  // Flag to know when we are done.
  int done (void) const;
  void done (int);
  /// Accessor to connection manager.
  Connection_Manager &connection_manager (void);
  /// Amount of debugging output to print out; 0 = none, 10 = lots
  int debug_level_;
protected:
  /// Connection manager.
  ...
}; ...
```

QuO Aspect Code

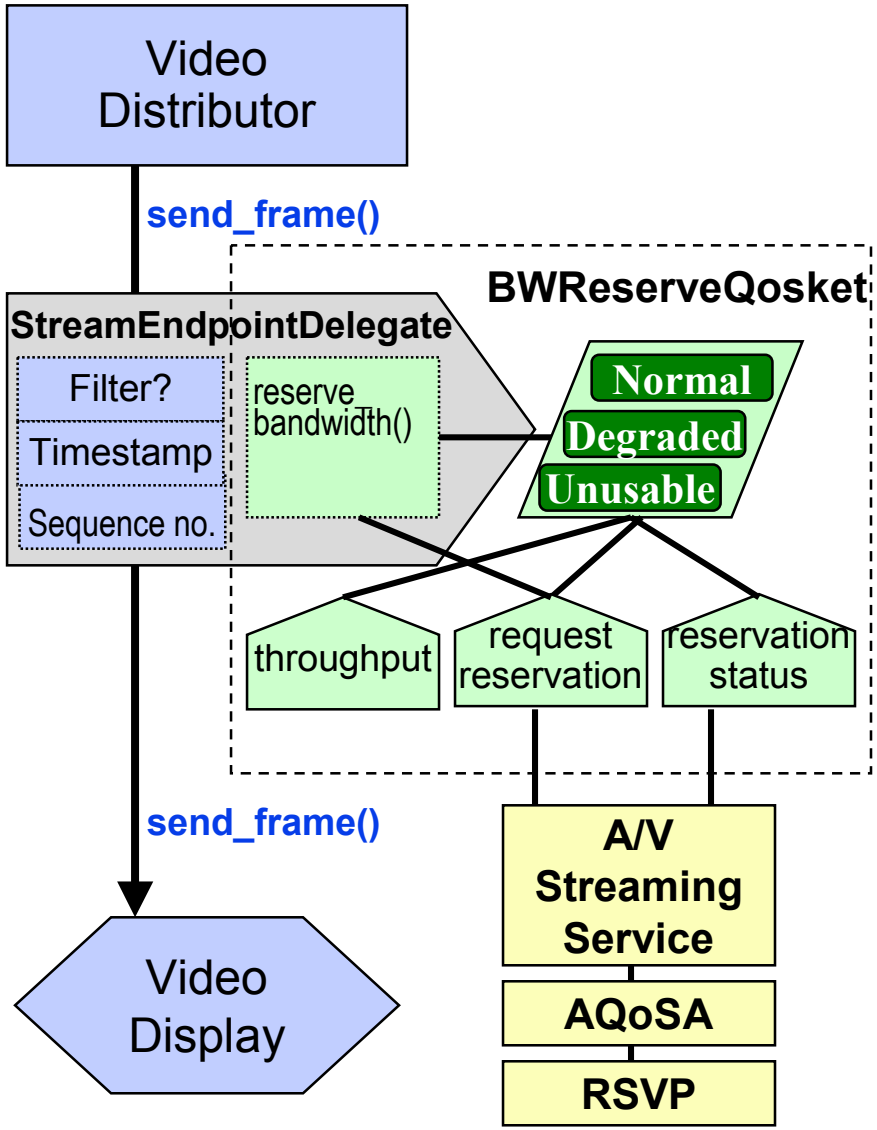
```
behavior Timestamp ()
{
  void distributor::send_frame(ACE_Message_Block *frame)
   {  inplaceof METHODCALL {
        ACE_Message_Block *timed_msg = add_timestamp(frame);
         remoteObj->send_frame(timed_msg);  }
   }
  void displayproxy::send_frame(ACE_Message_Block *frame)
   {extern long last_time_processed;
    inplaceof METHODCALL {
        ACE_Message_Block *timed_msg = remove_timestamp(frame, tsp);
        if (tsp >= last_time_processed) { // is this a late frame
         remoteObj->send_frame(timed_msg);
         last_time_processed = tsp;
        }
        // else drop frame, a more recent frame has already been delivered
};
```



Example Qoskets for the WSOA and UAV



UAV Qosket


Video Distributor
send_frame()
BWReserveQosket
StreamEndpointDelegate
Filter?
Timestamp
Sequence no.
reserve_bandwidth()
Normal
Degraded
Unusable
throughput
request reservation
reservation status
send_frame()
Video Display
A/V Streaming Service
AQoSA
RSVP

Wrapping Up: Future QoS Directions

- Moving up towards application's programming level
  - Design patterns and libraries (of contracts etc.) can help…
- More "multi-dimentional QoS" supported
  - Bandwidth "reservation": performance
  - replication+caching : availability
  - Security
  - Mobile/wireless: minimize power consumption and memory footprint
- Broadening from just the classical multimedia & http apps
  - VPNs
  - Collaboration
  - Virtual Reality
  - Application managers with QoS
- More OS-level substrates to choose from
  - Intserv & Diffserv combined, eventually across domains / ISPs
  - MS QoS (W2K has hooks for it…)
- Industry-Academic partnerships
  - Industry does not have time/labor to experiment/evaluate research substrates
  - Academics don't have time to learn industry products in depth

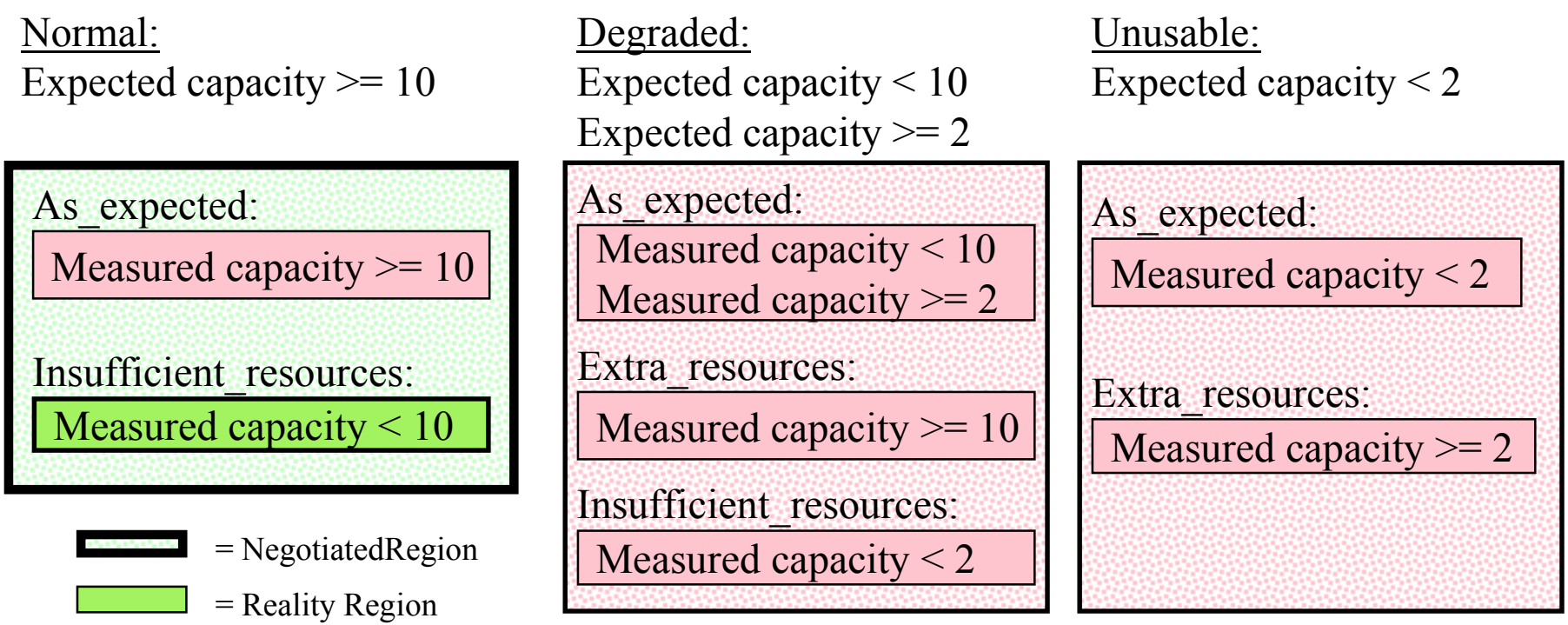
Conclusions

- Distributed QoS is a very broad area of research
  - Hot area, with lots of funding and cool problems!
  - Lots of issues that need to be dealt with systematically
- QuO provides end-to-end middleware support
  - Organizing end-to-end requirements
  - Collecting status inputs for adaptation
  - Providing adaptation at well-defined end-to-end locations
  - Reserving as many resources or as much QoS as possilble
  - Adapting when you don't get what you want
  - Reuse of all of the above application and system code
- QuO was designed from the start to integrate
  - Other researchers' mechanisms to provide QoS
  - Other researchers' status information sources
  - Multiple commercial and open-source middleware platforms
  - Has been very successful at this! (IMO; YMMV)

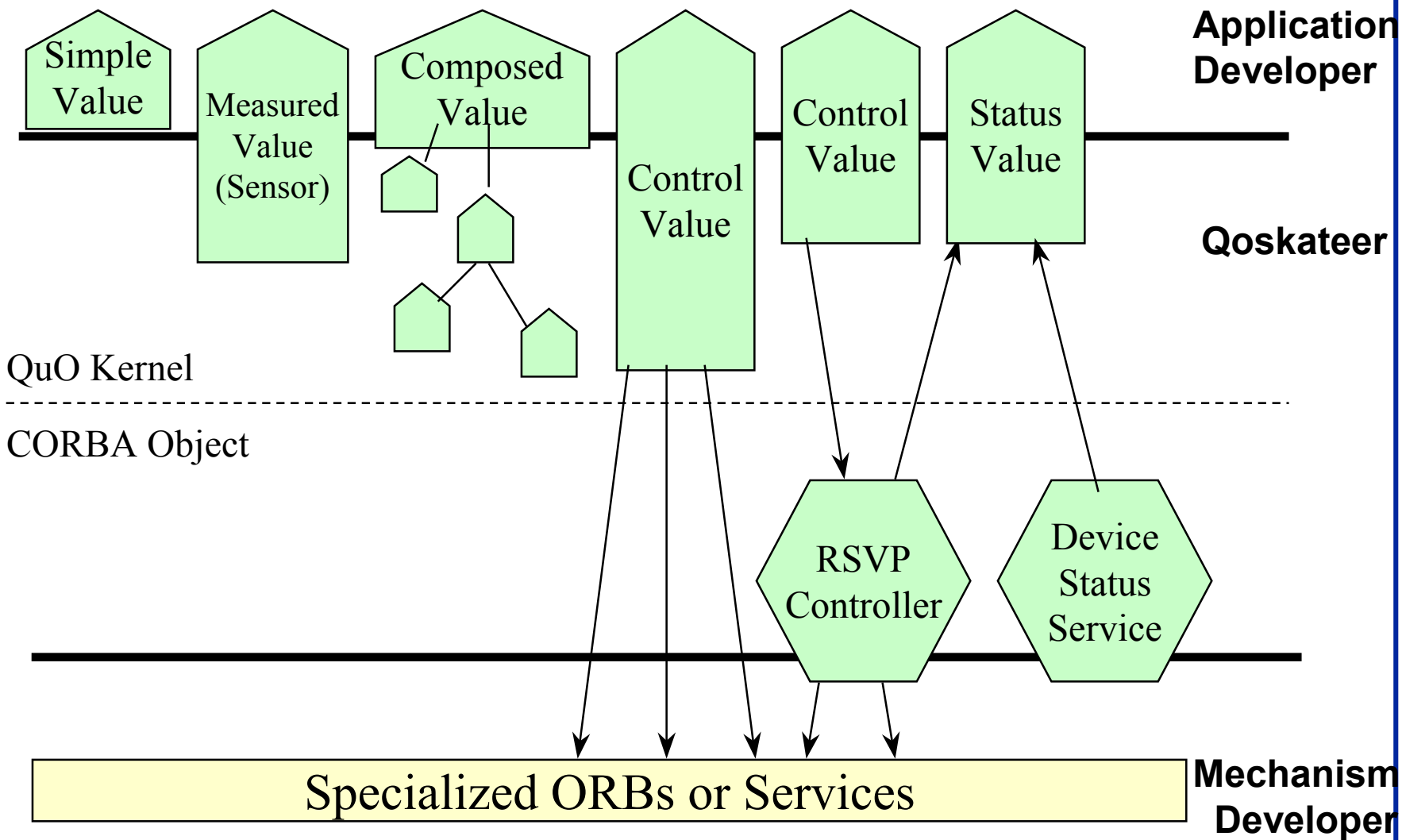
BACKUP SLIDES FOLLOW

Contracts summarize system conditions into negotiated and reality regions and define transitions between them

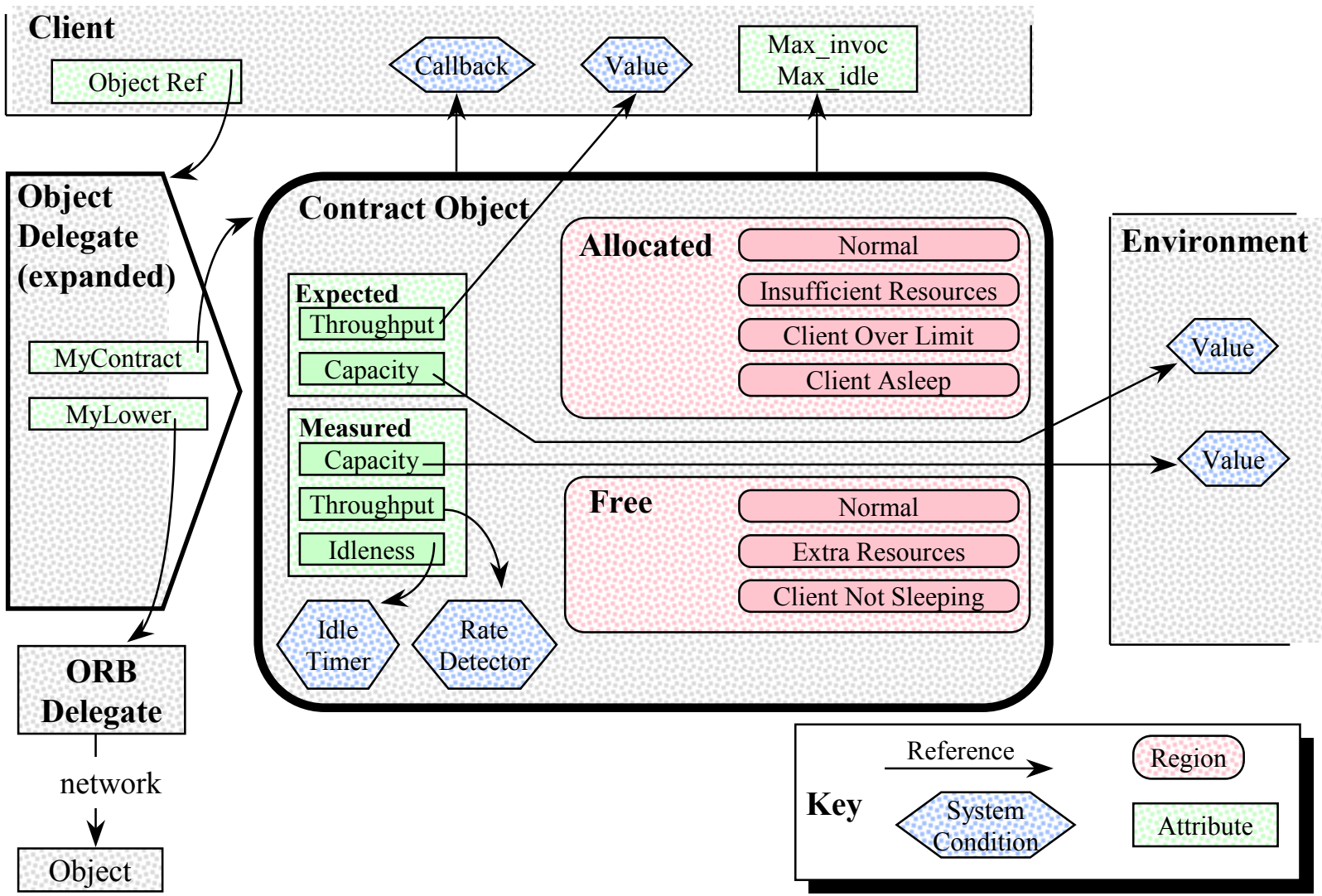
- *Negotiated* regions represent the expected behavior of client and server objects, and *reality* regions represent observed system behaviors
- Predicates using system condition objects determine which regions are valid
- Transitions occur when a region becomes invalid and another becomes valid
- Transitions might trigger adaptation by the client, object, ORB, or system



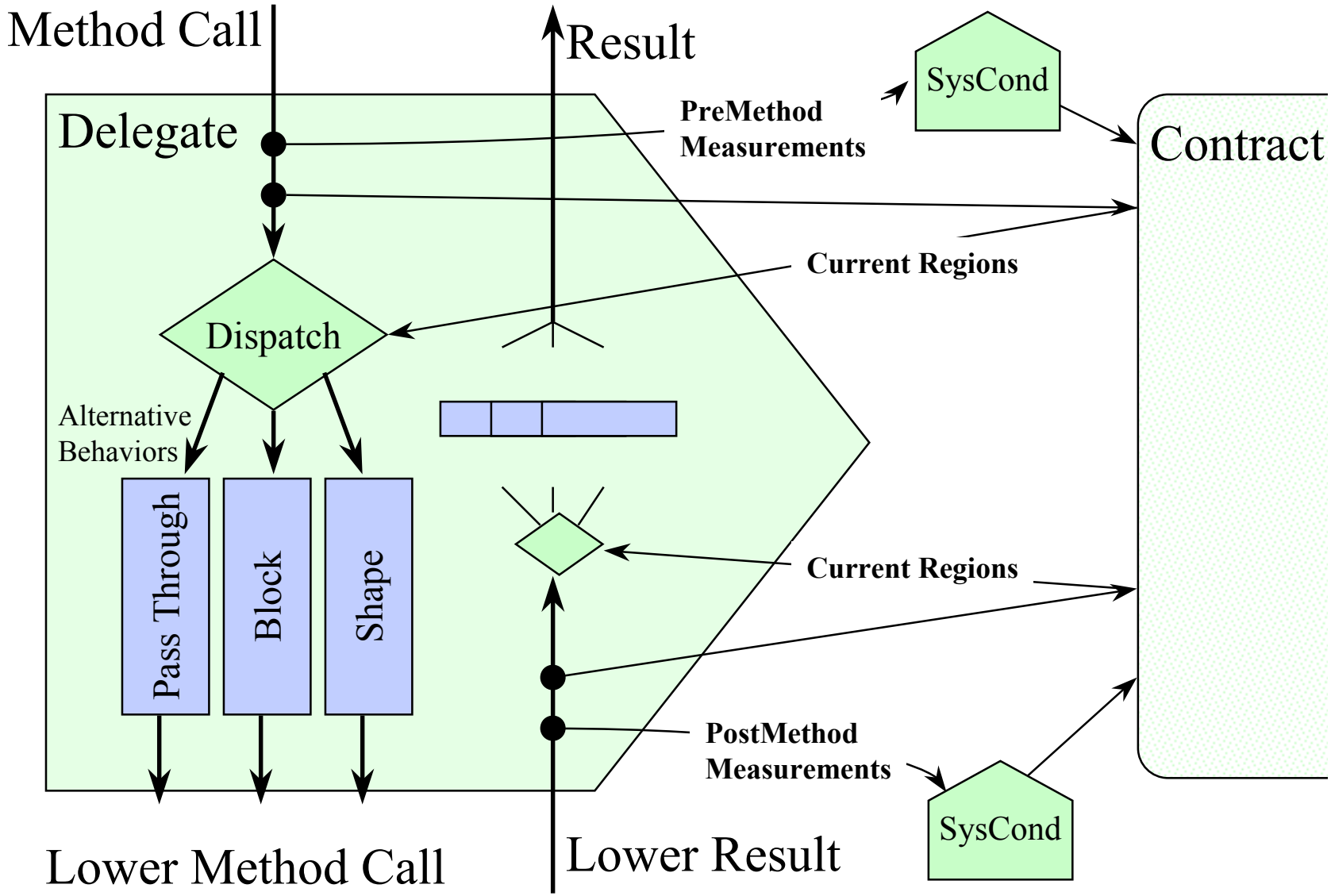
System Conditions Project a Value to the Application, But also Must Maintain the Value



Contracts Need to integrate with System Condition Probes and Object Delegates



Delegates Change Their Behavior Based on Their Contract's Current Regions



SDL code that supports choosing between replicated and non-replicated server objects

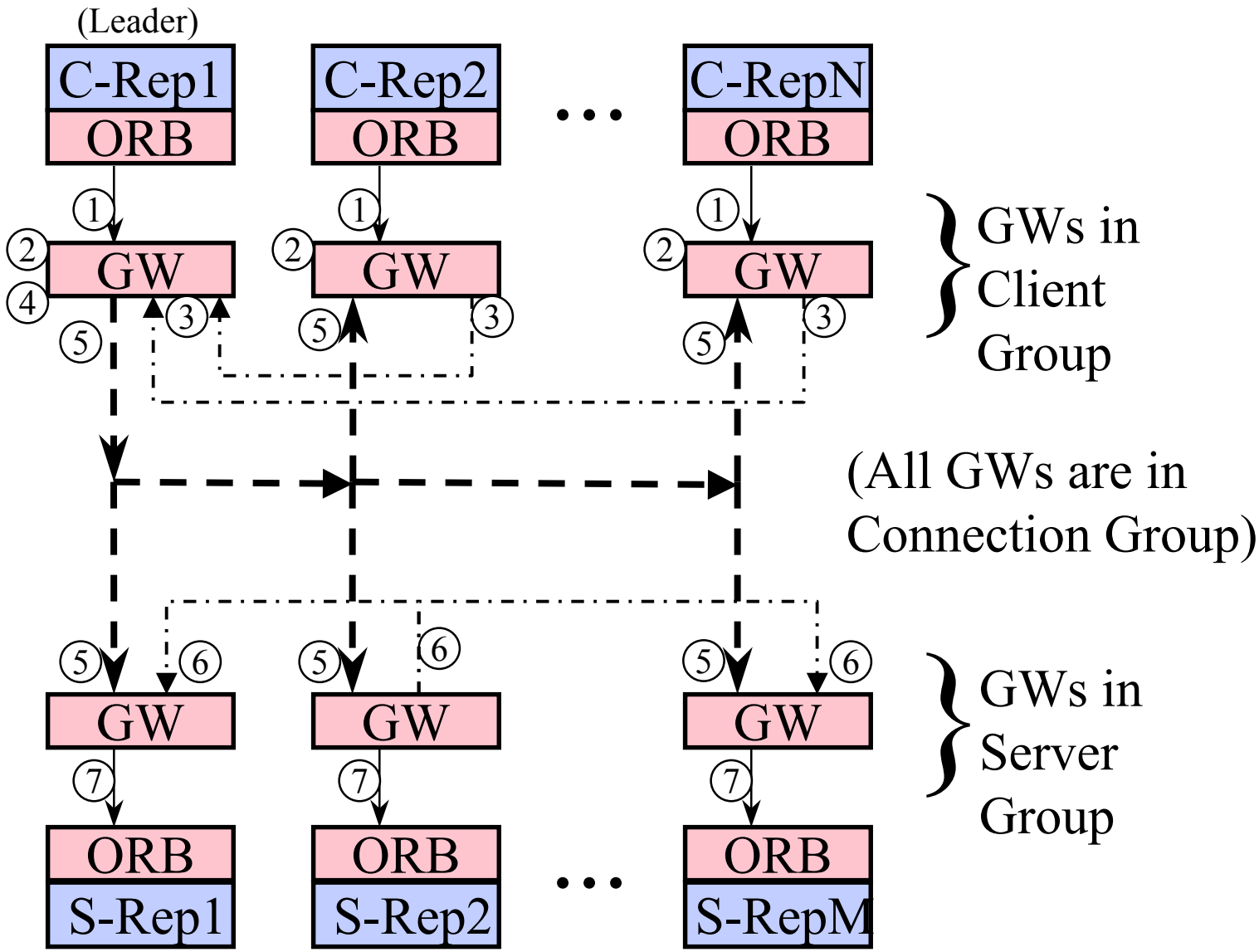
```
delegate behavior for Targeting and Replication is
  call calculate_distance_to_target :
    region Available.Normal :
      pass to calculate_distance_to_target_multicast;
    region Low_Cost.Normal :
      pass to calculate_distance_to_target_multicast;
    region Available.Low :
      java_code { System.out.println("Remote call would fail");
                  retval = -1; };
      cplusplus_code { cerr << "Remote call would fail");
                       retval = -1; };
  return calculate_distance_to_target :
    pass_through;
  default : pass_through
end delegate behavior;
```

- SDL supports choosing between methods, run-time binding, and embedded Java or C++ code.

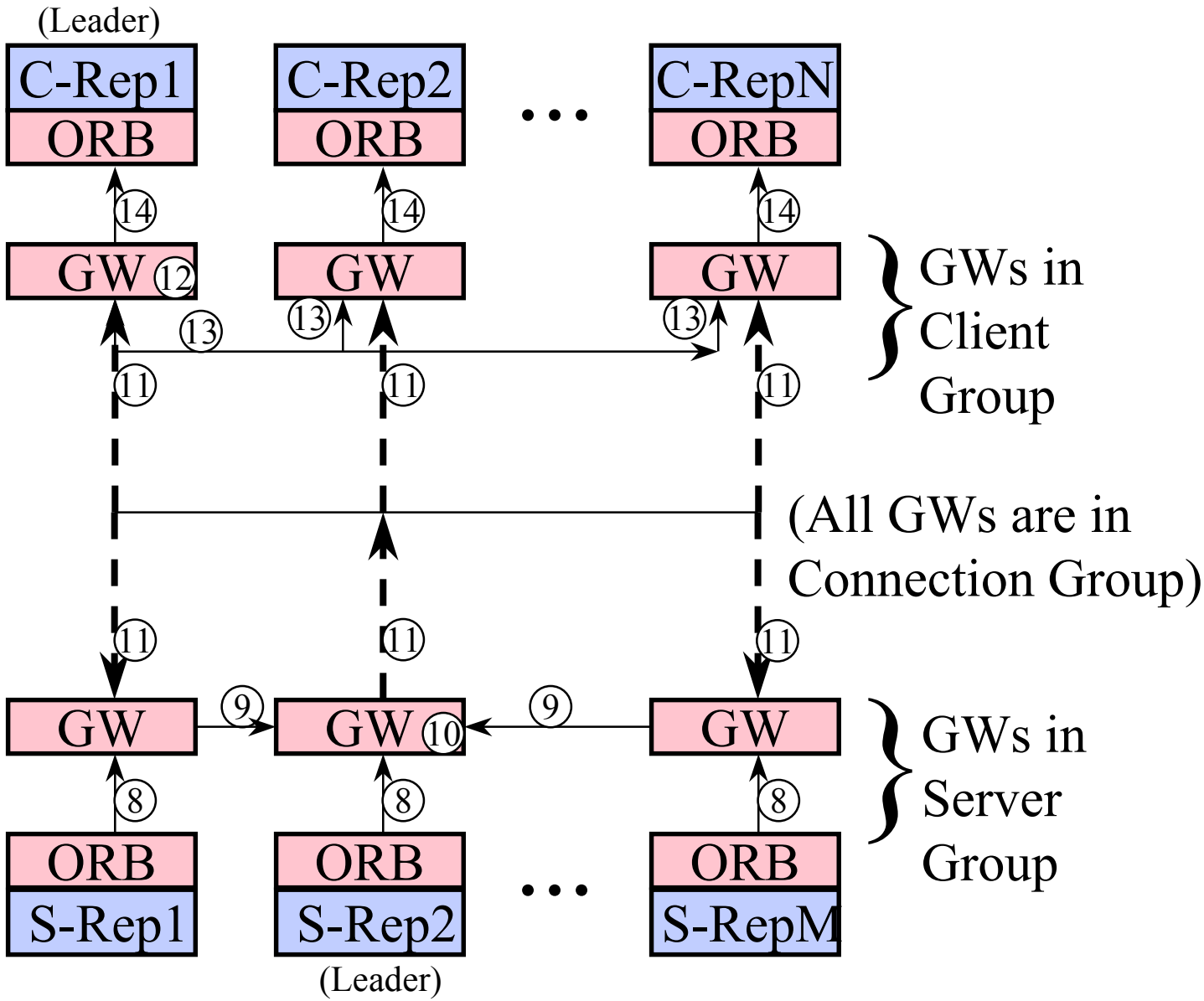
AQuA Handlers: Design Space has Many Variables!

- Client group has leader or has no leader
  - how much do you trust client group?
- Server group has leader or has no leader
- Multicast strengths (total, causal, FIFO, …) used in connection group
- Which members of client and server groups are in connection group
- Location and algorithm for voting
- How many rounds of multicasts (e.g., for byzantine)
- Location of buffering of requests/replies
  - Caveat: not shown in following diagrams
- Also: interaction with handler "upstream" or "downstream" in a nested call
  - A → B → C: handlers A → B and B→ C need to be managed together, for reasons of performance and possibly correctness

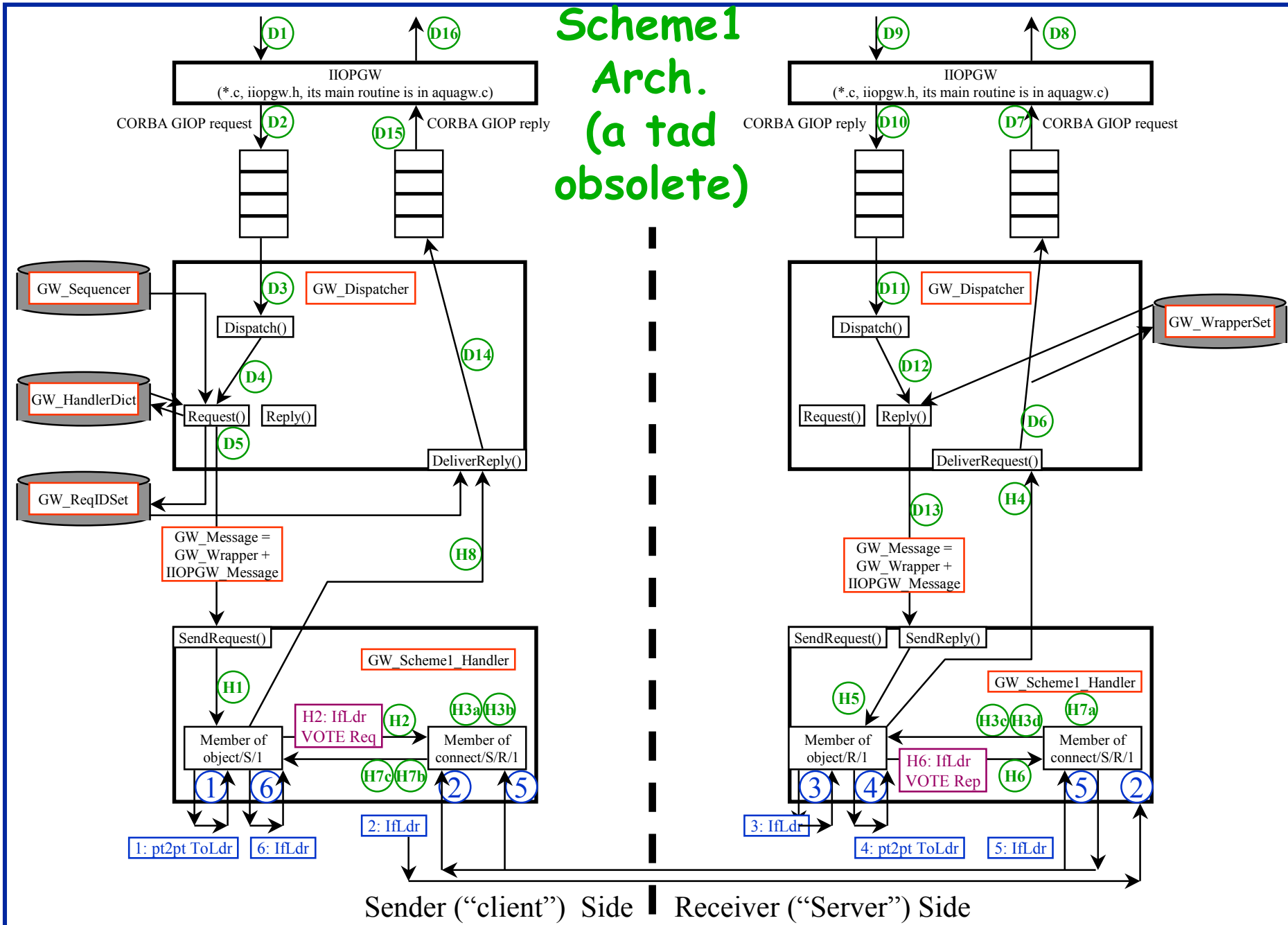
AQuA Scheme1 Request Steps



AQuA Scheme1 Reply Steps



Scheme1 Arch. (a tad obsolete)



Implements the active protocol resembling that in Proteus design doc. Server-side Ldr GW votes on requests (H2), receiver-side GW ldr votes on replies (H6). Assumes clients have no asynch. requests outstanding, so a gap in a reply sequence in H6 means a one-way request occurred (need trickier data structures to handle asynch replies: B,<n1,n2…,nk>.) Void where prohibited by law. YMMV.

Scheme1 Steps (a tad obsolete)

D1. Sender ("client") ORB delivers IIOP msg.
D2. S-IIOPGW enqueues msg
D3. Dispatcher dequeues message
D4. Dispatcher looks up next sequence and calls Request()
D5. Dispatch handler looked up and dispatched to; stores local ReqID

H1. GW_Scheme1_Handler::SendRequest() does
- **a**. **S-GWs** send pt2pt **msg #1** to **Ldr S-GW**
- **b**. **NonLdr S-GWs** buffer **msg #1** (to be deleted in **H3b**).

H2. When recv **msg #1**, **Ldr S-GW** **votes on requests**, (in this case sends just the first one), and sends chosen request in **msg #2** to connection group unordered
H3. When receive **msg #2**
- **a**. All **NonLdr R-GWs** store **msg #2** in buffer (to be deleted in **H4b**)
- **b**. **NonLdr S-GW** delete **msg #1** from buffer (stored in **H1b**)
- **c**. **Ldr R-GW** sends totally-ordered **msg #3** to **R-GWs** to order across all client groups

H4. When receive **msg #3**,
- **a**. **R-GWs** call Dispatcher->DeliverRequest()
- **b**. **NonLdr R-GW** deletes **msg #2** from buffer (stored in **H3c**)

D6. Dispatcher places invocation msg in queue for IIOPGW
D7. IIOPGW removes msg from queue
D8. IIOPGW delivers msg to Receiver ("server") ORB
D9. "server" ORB sends back IIOP reply msg to R-IIOPGW
D10. R-IIOPGW queues reply message for R-GW
D11. R-GW dequeues reply msg
D12. R-W calls dispatch->Reply()
D13. R-GW Dispatcher->Reply() notes handler# from Msg, looks up wrapper, and calls Handler1->SendReply()

H5. GW_Scheme1_Handler::SendReply() does
- **a**. **R-GWs** send reply **msg #4** pt2pt to **Ldr R-GW**
- **b**. **NonLdr R-GW** buffers **msg #4** (to be deleted in **H7a**)

H6. When **msg #4** arrives **Ldr R-GW** **votes on replies** and sends chosen reply (in this case the first **msg #4** with this seq#) in **msg #5** unorderd to connection grp. Discards the rest of the replies with same seq#. Gaps in seq# may occur here, but if so this is due to a one-way request, since for now we assume no asynch client requests.
H7. When **msg #5** received
- **a**. **NonLdr R-GW** can delete buffered reply **msg #4** (stored in **H5b**) (note **Ldr R-GW** does not receive it because unorderd; else it would just discard it)
- **c**. **Ldr S-GW** sends reply **msg #6** ordered multicast to all **S-GWs**
- **c**. **NonLdr S-GW** stores reply **msg #6** in buffer (deleated in **H8b**)

H8. When **msg #6** arrives,
- **a**. **S-GWs** call dispatcher->DeliverReply() with this reply message.
- **b**. **NonLdr S-GWs** delete **msg #5** from buffer (stored in **H7c**).

D14. **S-GWs** DeliverReply() queues msg for IIOPGW
D15. IIOPGW dequeues message
D16. IIOPGW sends IIOP message to sender "client" ORB