Distributed Quality of Service

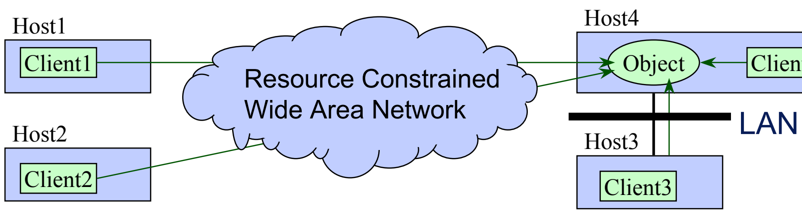
Dave Bakken
http://www.eecs.wsu.edu/~bakken/

Cpt. S 464/564 Lecture
December 6 &11, 2000

Outline of Talk

- QoS: The problem, and basic definitions
- QoS Implementation Issues
- Quality Objects (QuO)
- QuO Case Study
- Future QoS directions

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

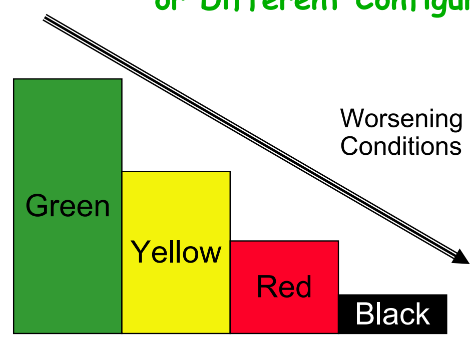
QoS Basic Definitions (cont.)

- 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)

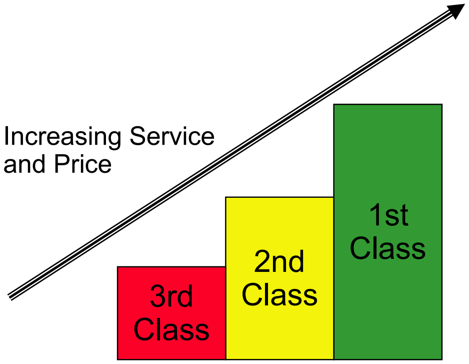
QoS for Users: Adapting to Worsening Conditions or Different Configurations



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

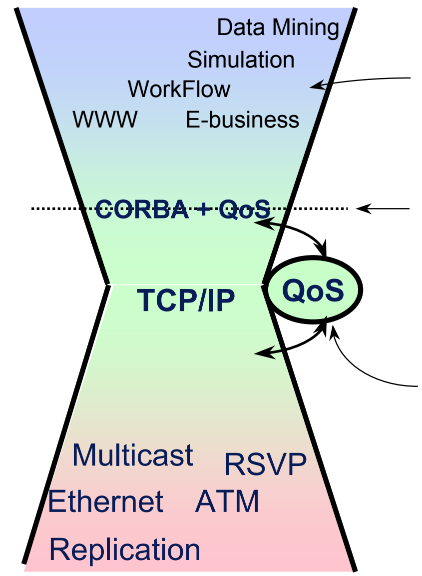
Condition	Conferencing	Participants	Info Service
Green	Full color multimedia	Key and useful participants	Quick DB queries
Yellow	B&W multimedia	Key and useful participants	Acceptable DB queries
Red	Audio	Key participants only	Acceptable DB queries
Black	None	None	None

QoS for Service Providers (and their 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

Distributed object middleware with QoS extensions is a powerful abstraction layer on which to build applications

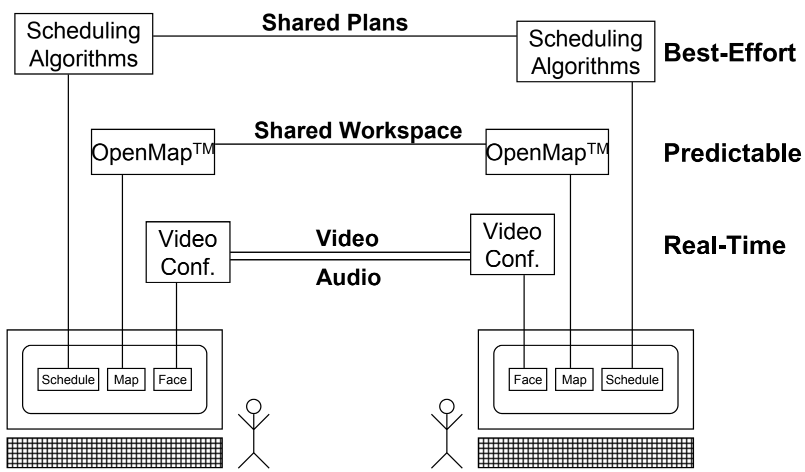


Diverse applications for geographically dispersed, heterogeneous environments – not just multimedia apps!

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

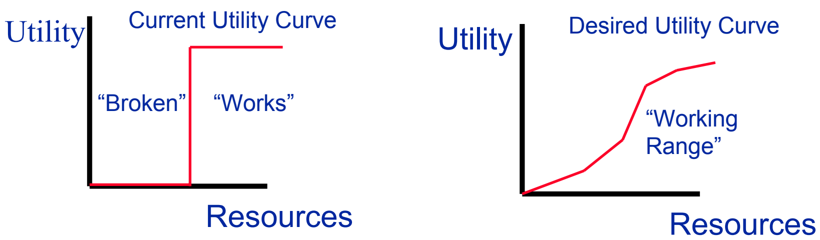


See "ARPA-Rome Planning Initiative" IEEE Expert, Feb 1995

Outline of Talk

- QoS: The problem, and basic definitions
- QoS Implementation Issues
- Quality Objects (QuO)
- QuO Case Study
- Future QoS directions

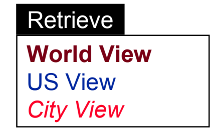
"Awareness without Pain" I: Users' and Application Programmers' Awareness



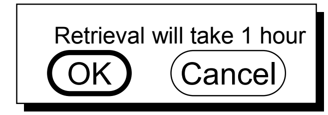
- Users and application programmers need to be **aware** of their demand for resources, and be able to **change!**
- Users/applications must understand the **utility** of their demand
  - know their usage patterns and QoS requirements
- Users/applications must be able to **change demand** based on volatility 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: The User Should See a "Graceful Degradation" of the App, not a Hard Failure

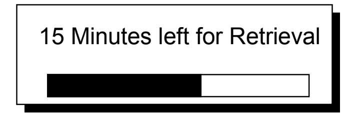
- Functions marked with cost cues



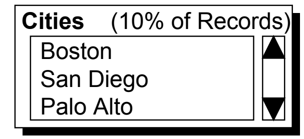
- Middleware asks for more advice



- Middleware predicts long response times

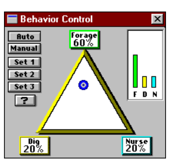


- Application tolerates aborted operations with partial results

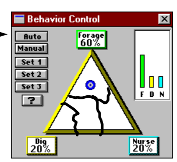


"Awareness without Pain" III: Needed: Higher-level QoS APIs and User Interfaces

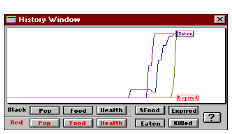
- Most programmers and users of advanced distributed applications can't deal with QoS because they
  - Are not very sophisticated in distributed systems issues (let alone QoS)
  - Have enough to do already providing/using the applications' main job without worrying about QoS
- QoS contracts can give 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:



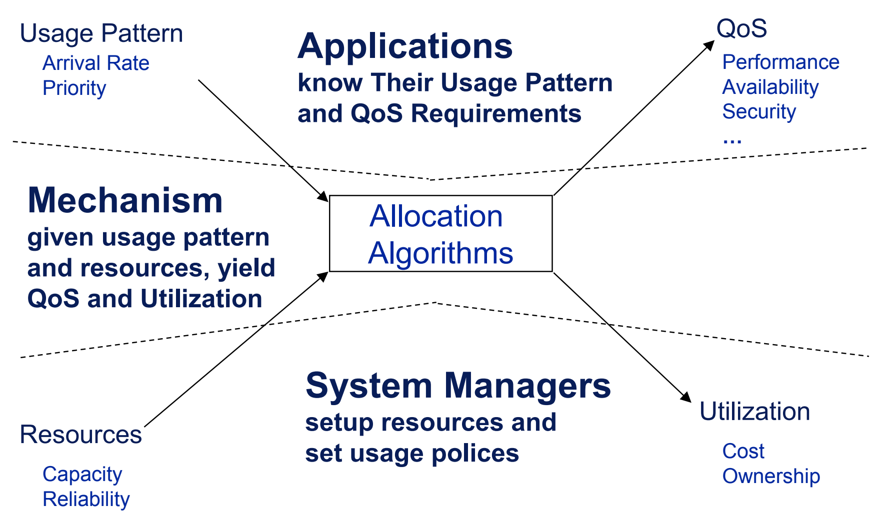
QoS developer provides multiple implementations trading off multiple properties and resources, with a high-level mapping



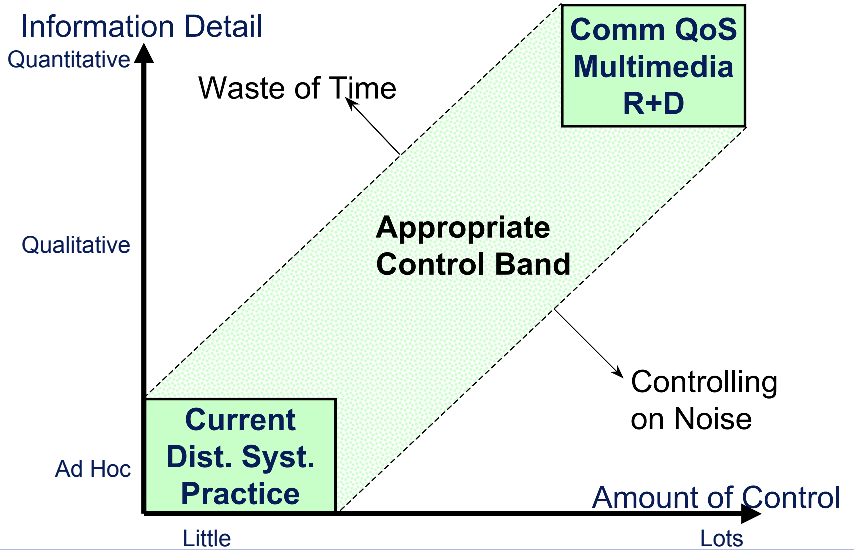
  - User feedback:



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



QoS-Aware Resource Management II: Control over Resource Allocation is Useless without Information on Usage Patterns and 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

QuO's Philosophy is to Support Monitoring of System Conditions and 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, …

QuO's Philosophy is to Support 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

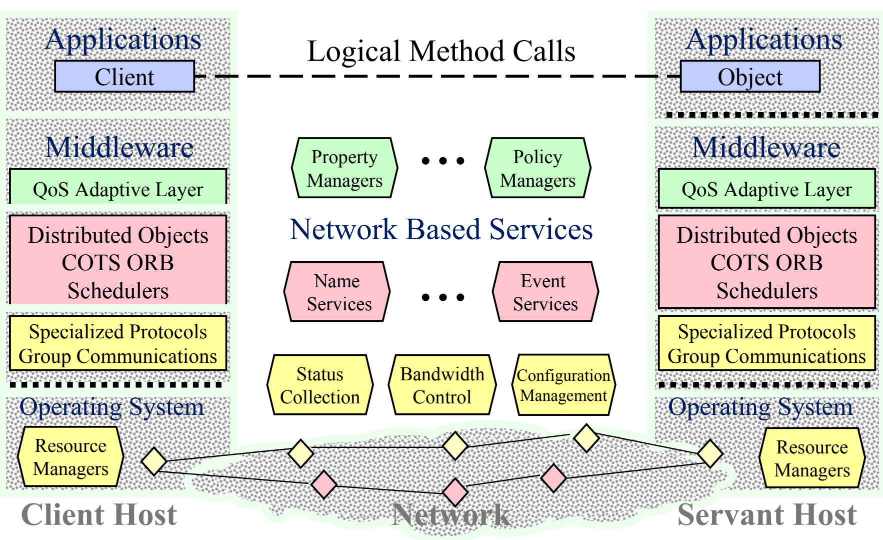
Outline of Talk

- QoS: The problem, and basic definitions
- QoS Implementation Issues
- Quality Objects (QuO)
- QuO Case Study
- Future QoS directions

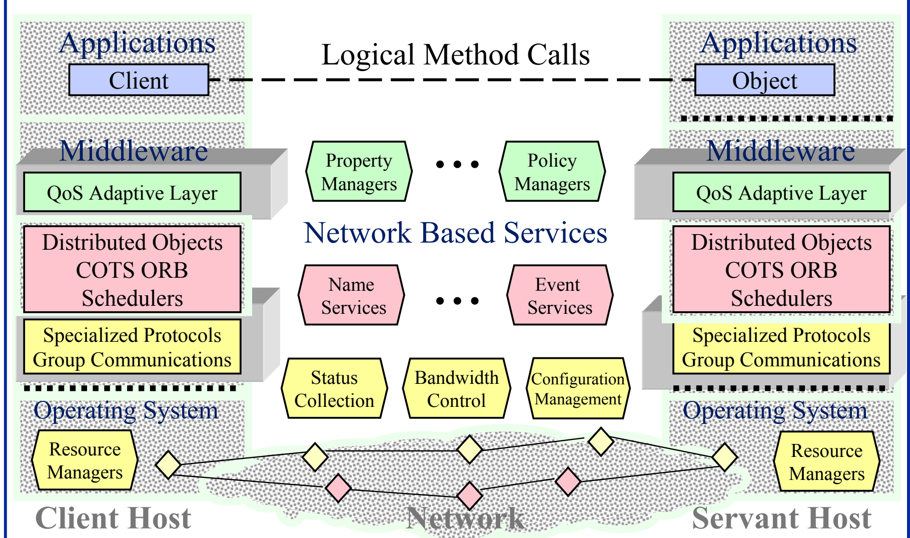
QuO History

- 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, U. Oregon, U. Illinois, Wash. U. St. Louis., Columbia U, Trusted Information Systems(TIS), Boeing,...)

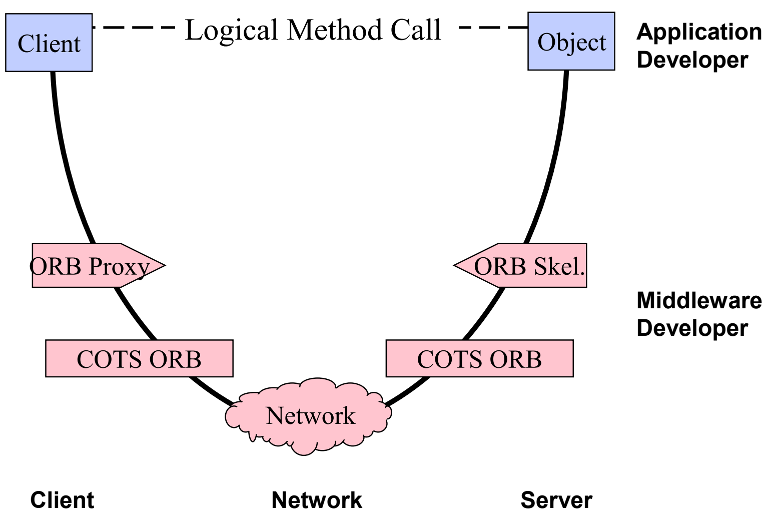
Simplified Common System Model



Adaptive QoS Interface and Control



Simplified DOC (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

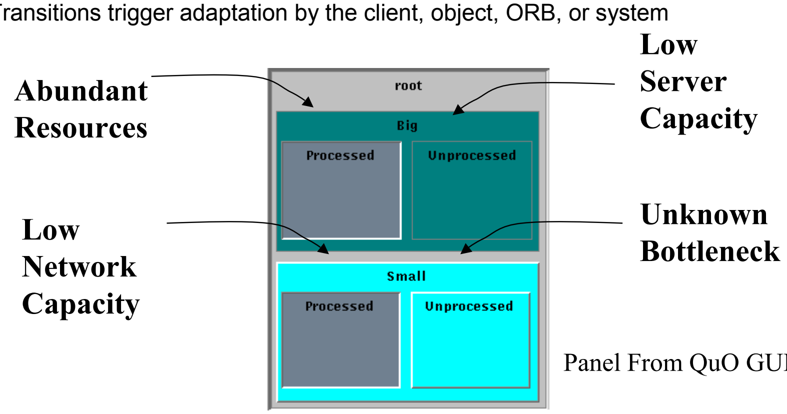
The QuO framework provides

- 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

QuO

Contracts Summarize System Conditions into Regions Each are 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

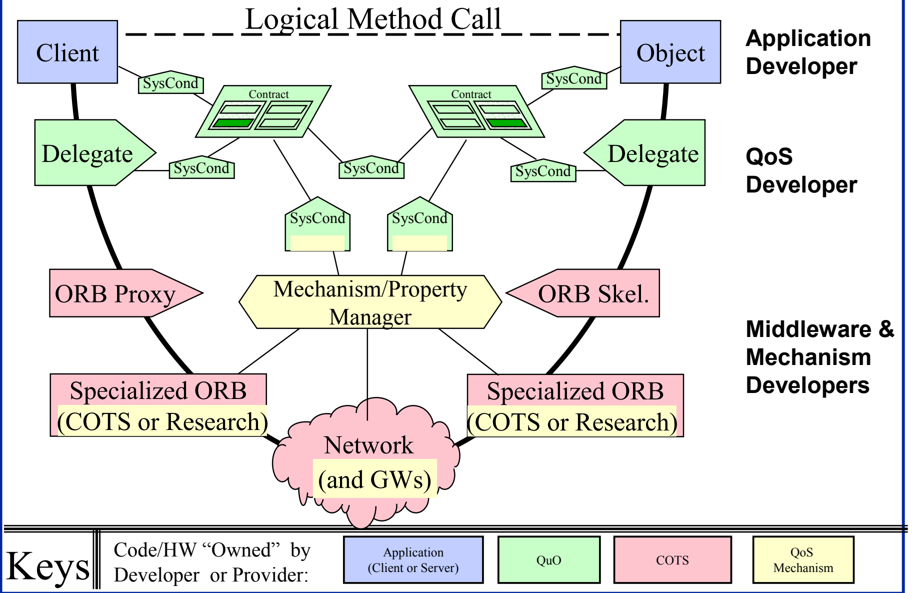


Panel From QuO GUI

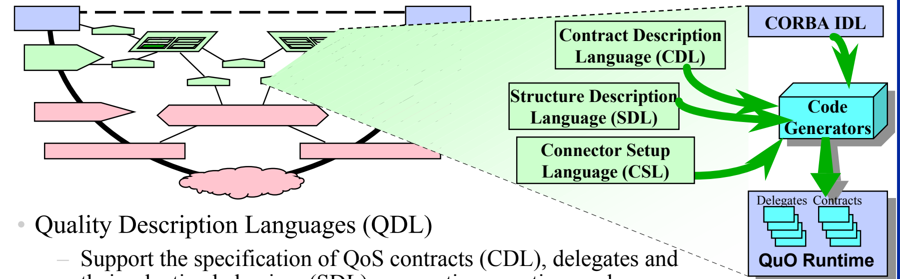
A QuO application contains additional components (from traditional CORBA/DOC applications)

- **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 QoS control and measurement into the DOC remote method call

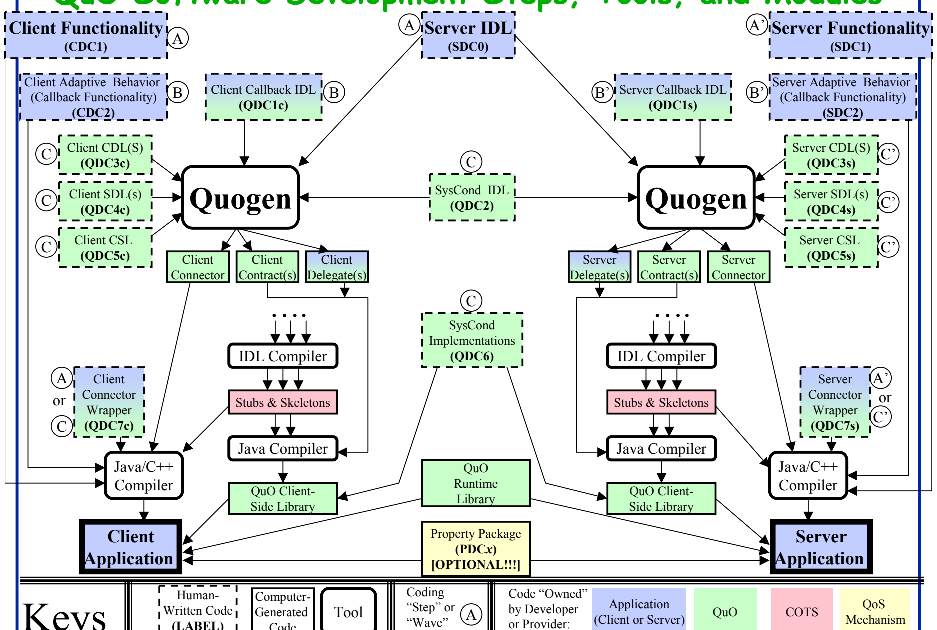


The QuO Toolkit provides tools for building QuO applications



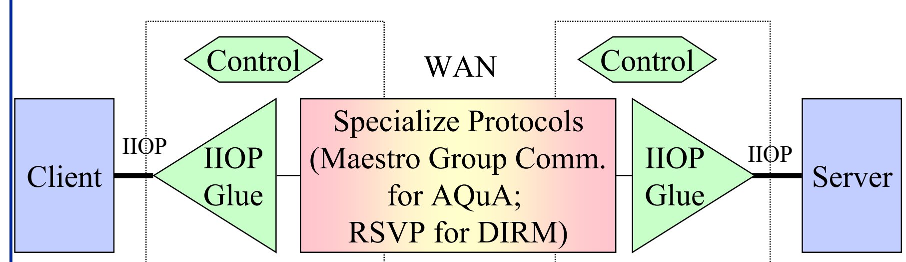
- Quality Description Languages (QDL)
  - Support the specification of QoS contracts (CDL), delegates and their adaptive behaviors (SDL), connection, creation, and initialization of QuO application components (ConnDL)
  - QuO includes code generators that parse QDL descriptions and generates Java and 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 Software Development Steps, Tools, and Modules

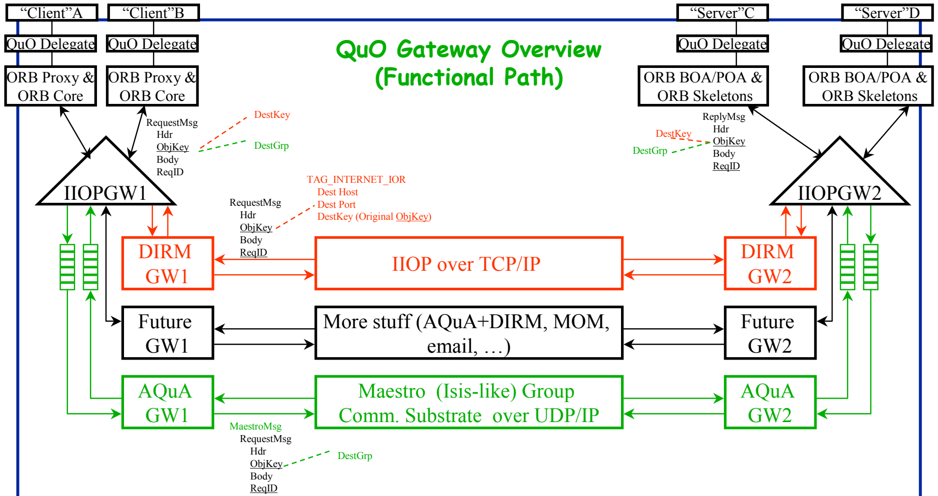


The QuO Gateway Manages IIOP Connections and Interfaces to Protocols which Manage QoS

- 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...

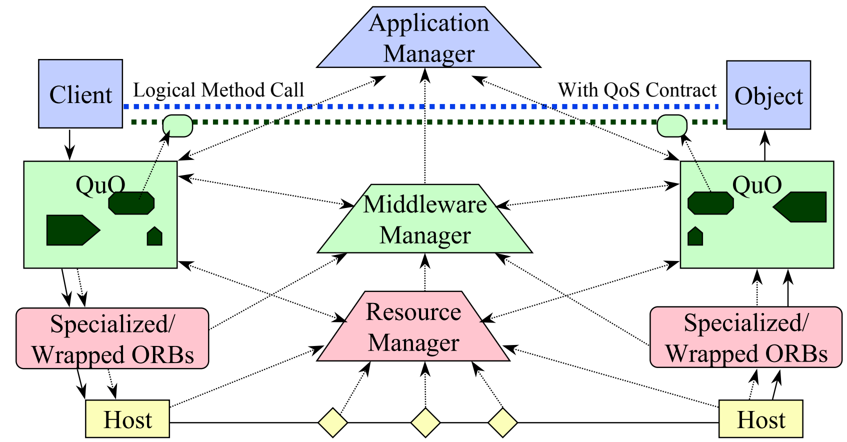


QuO Gateway Overview (Functional Path)



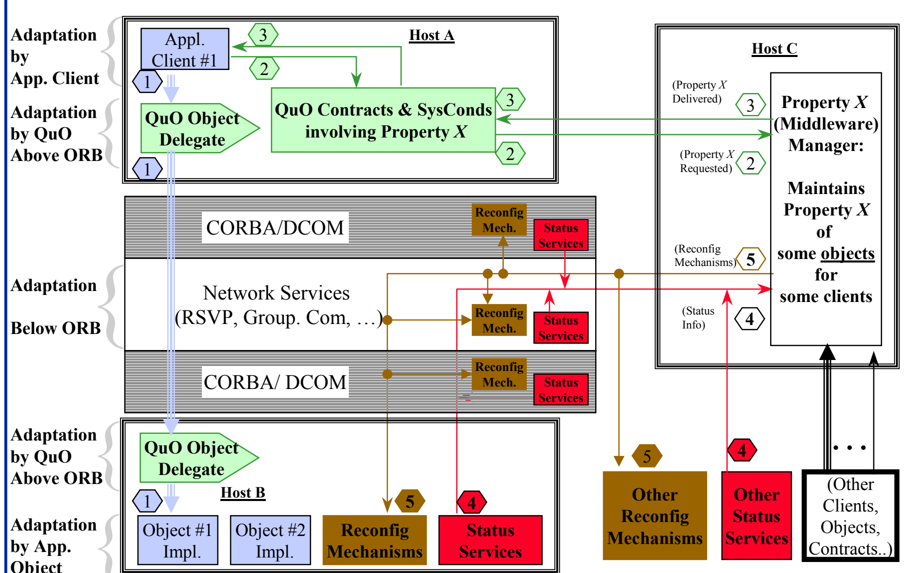
- We can (and do! and must!) rewrite ObjKey and ReqID; we just have to restore them when we pass them back to the appropriate ORB so it can use them to demux the reply, lest the poor ORB choke on it….
- Mappings between {Process,Host} and GWs is flexible (~TBD):
  - DIRM may want one per LAN/cluster 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 Reconfiguration Policies at Different Levels and from Different Sources



- Functional Info (solid line) and "QoS meta-data" (dashed line)
- Translation between Manager Layers
- Centralized view vs. edge view

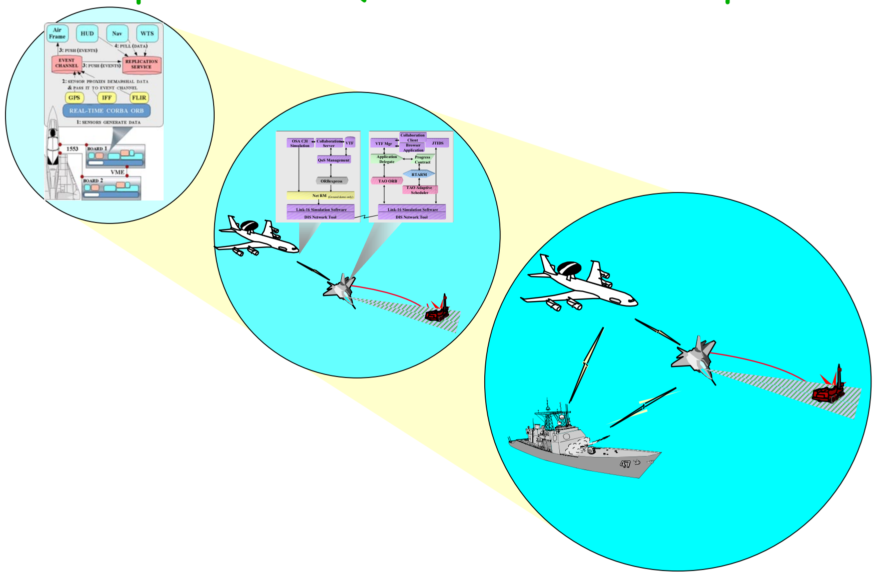
Canonical QuO Arch. for Generic Property Package *X*



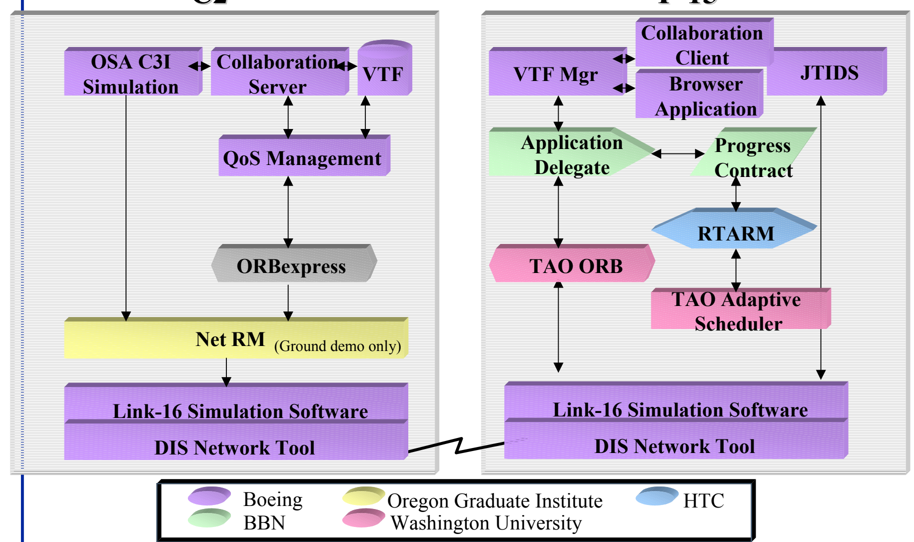
Outline of Talk

- QoS: The problem, and basic definitions
- QoS Implementation Issues
- Quality Objects (QuO)
- QuO Case Study
- Future QoS directions

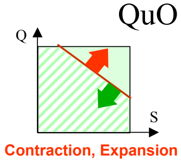
Multiple Levels of QuO Coordination are Required!



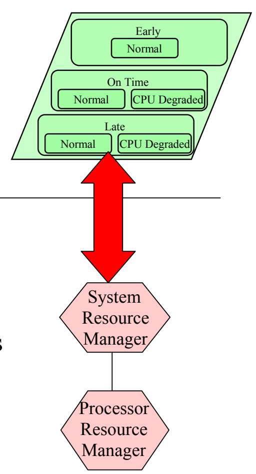
Weapon Systems Open Architecture (WSOA) Boeing Phantom Works, St. Louis



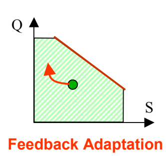
WSOA QoS Control Flow



QuO
- 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)



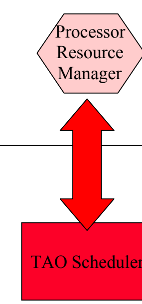
RT-ARM



- 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)
- Future QoS directions

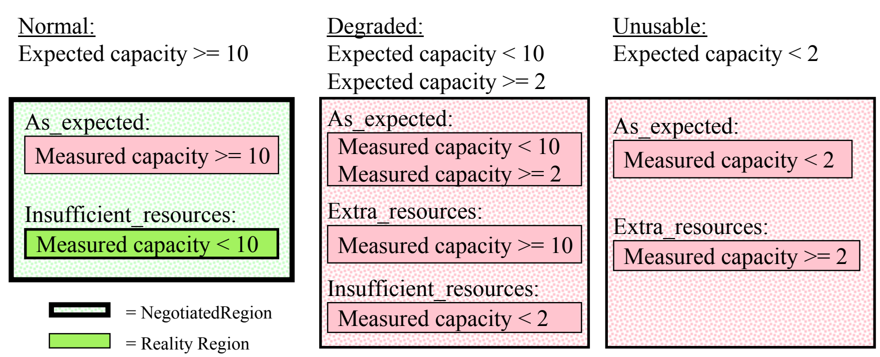
Future QoS Directions

- 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

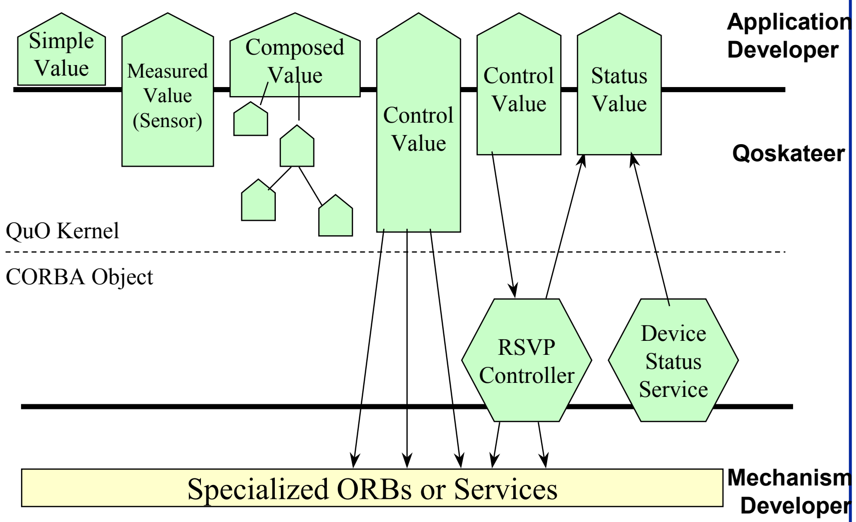
BACKUP SLIDES FOLLOW

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

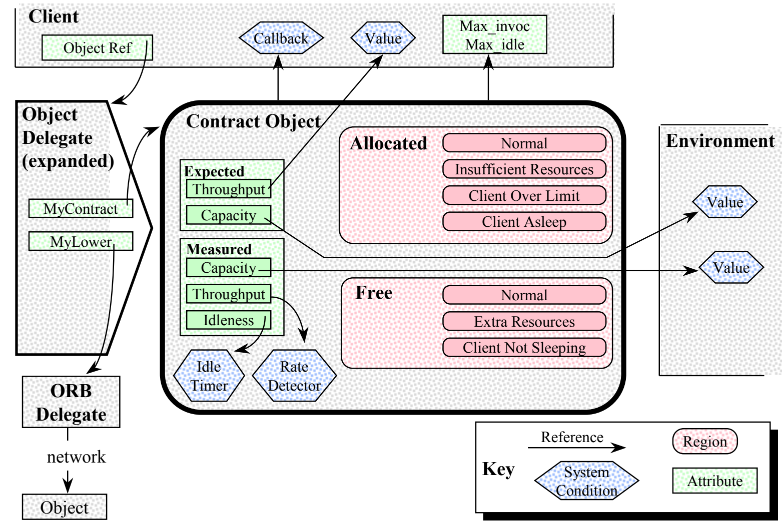
- *Negotiated* regions represent the expected behavior of client and server objects, and *reality* regions represent observed system behaviors
- Predicates using system condition objects determine which regions are valid
- Transitions occur when a region becomes invalid and another becomes valid
- Transitions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



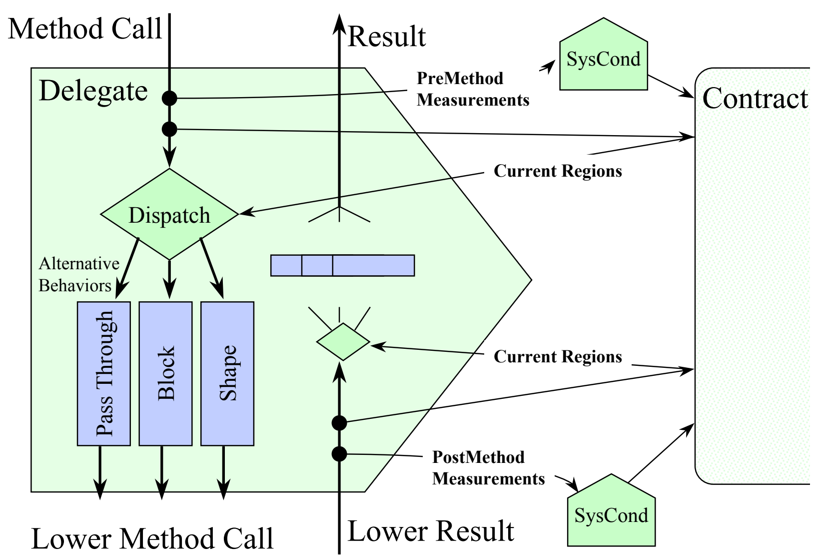
System Conditions Project a Value to the Application, But also Must Maintain the Value



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

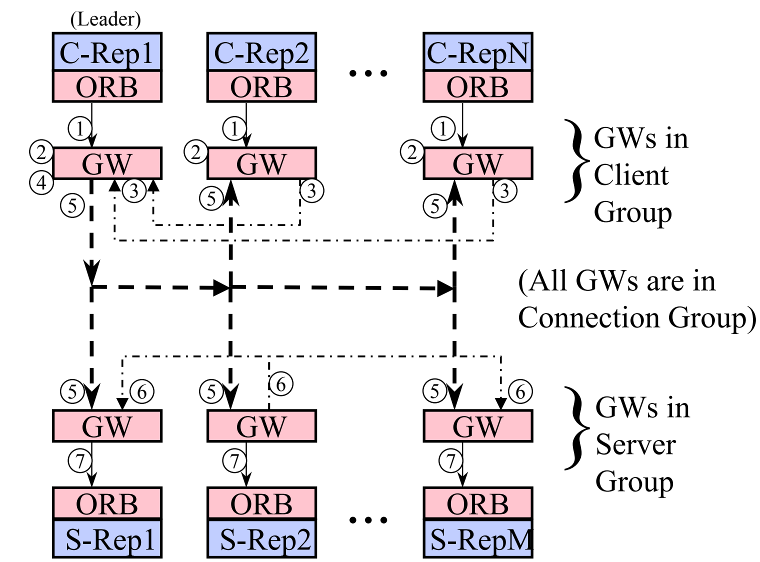
```
delegate behavior for Targeting and Replication is
  call calculate_distance_to_target :
    region Available.Normal :
      pass to calculate_distance_to_target_multicast;
    region Low_Cost.Normal :
      pass to calculate_distance_to_target_multicast;
    region Available.Low :
      java_code { 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.

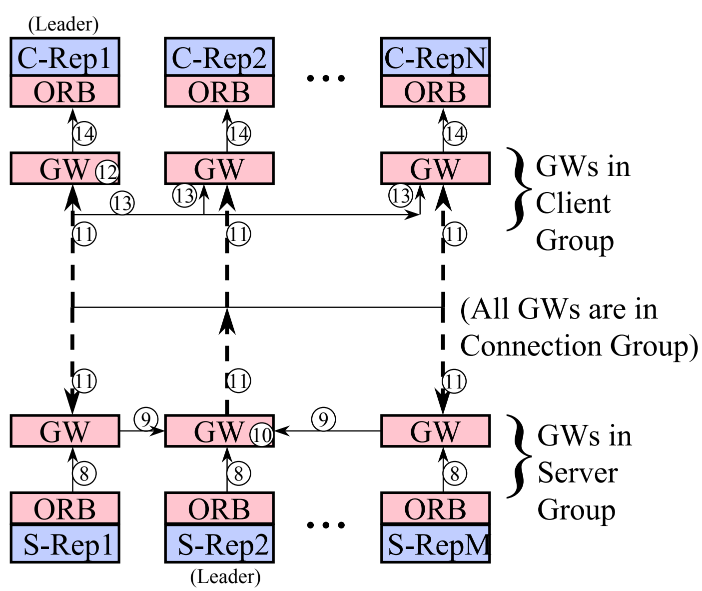
AQuA Handlers: Design Space has Many Variables!

- 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

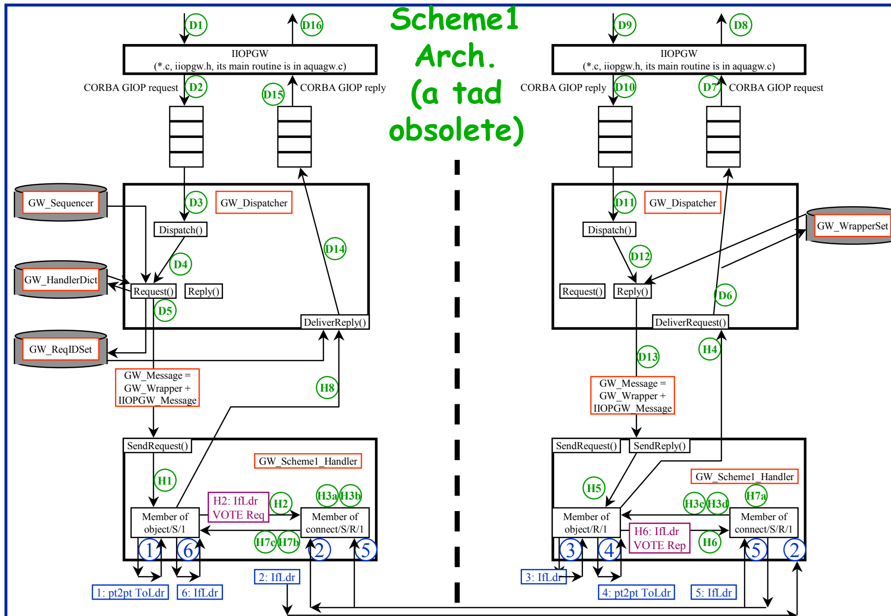
AQuA Scheme1 Request Steps



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