Middleware

What it is, and How it Enables Adapdivity and Dependability

David E. Bakken
School of Electrical Engineering and Computer Science
Washington State University
Pullman, Washington USA

43rd Meeting of IFIP Working Group 10.4
Sal, Cape Verde
January 4, 2003

Context: (Most) Technology Marches On

- Hardware technology's progress phenomenal in last few decades
  - Moore's Law
  - Metcalf's Law
  - Graphics processing power
- Software technology's progress is much more spotty
  - "Software crisis"
  - Yet SW is a large and increasing part of complex apps/systems!
- Apps and systems are rapidly becoming (more) networked
  - Oops, distributed software is much harder yet to get right…
- Middleware a promising technology for programability of distributed systems
  - Also fertile grounds for adaptivity and dependability….

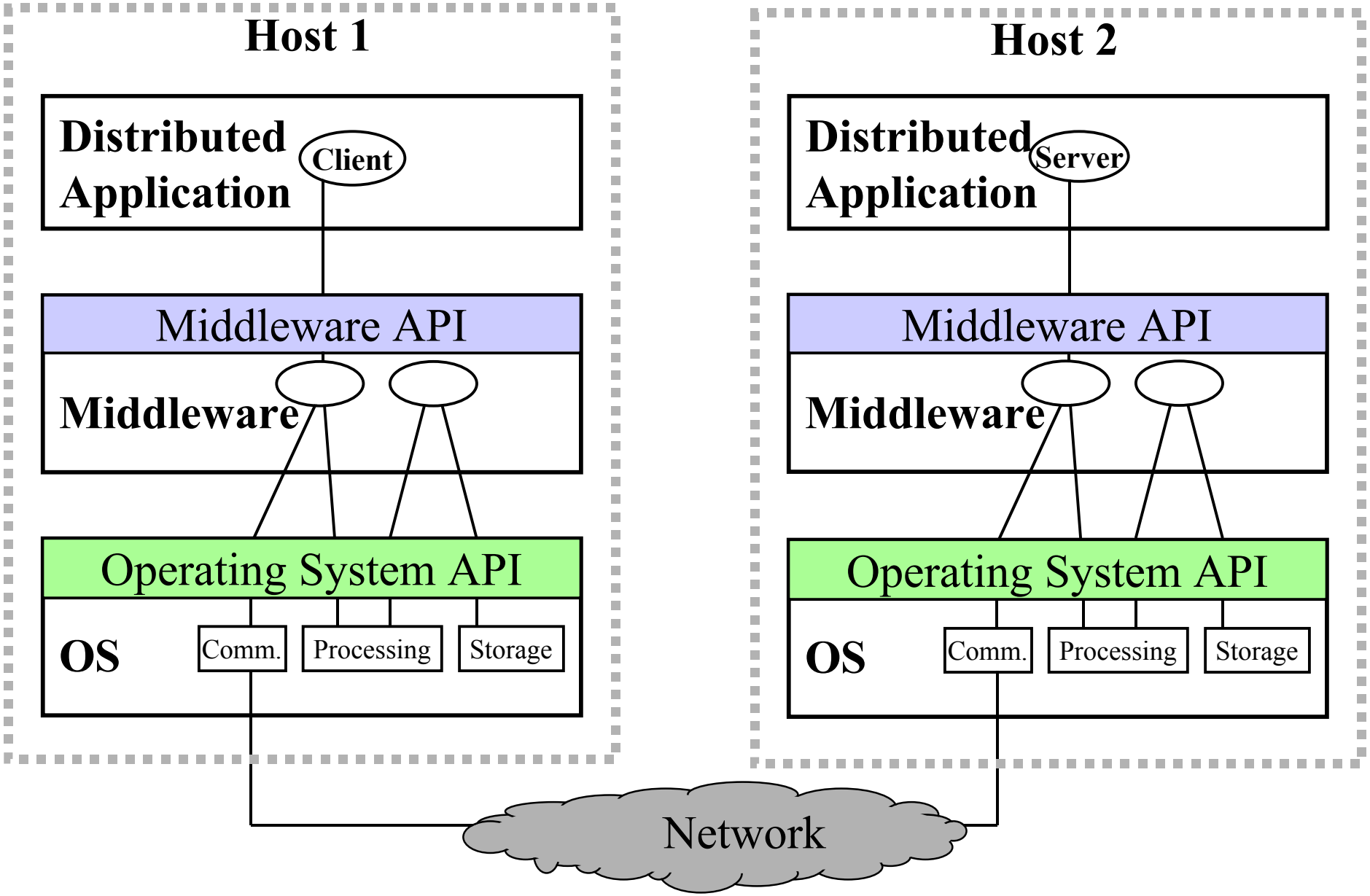
Outline

- **Middleware: Definition and Benefits**
- Examples of Middleware
- Middleware and Adaptation
- Multi-Layered Middleware
- Middleware and Dependability

Why Middleware?

- Middleware == "**A layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system**" [Bak03]
- Middleware exists to help manage the complexity and heterogeneity inherent in distributed systems
- Middleware provides higher-level building blocks ("abstractions") for programmers than the OS provides
  - Can make code much more portable
  - Can make them much more productive
  - Can make the resulting code have fewer errors
  - Analogy — MW:sockets ≈ HOL:assembler
- Middleware sometimes is informally called "plumbing"
  - Connects parts of a distributed application with "data pipes" and passes data between them

Middleware in Context



Middleware Benefit: Masking Heterogeneity

- Middleware's programming building blocks mask heterogeneity
  - Makes programmer's life much easier!!
- Kinds of heterogeneity masked by middleware (MW) frameworks
  - All MW masks heterogeneity in network technology
  - All MW masks heterogeneity in host CPU
  - Almost all MW masks heterogeneity in operating system (or family thereof)
    - Notable exception: Microsoft middleware (*de facto*; not *de jure* or *by fiat*)
  - Almost all MW masks heterogeneity in programming language
    - Noteable exception: Java RMI
  - Some MW masks heterogeneity in vendor implementations
    - CORBA best here

Middleware Benefit: Transparency

- Middleware can provide useful transparencies:
  - Distribution Transparency
  - Location transparency
  - Concurrency transparency
  - Replication transparency
  - Failure transparency
  - Mobility transparency
- Masking heterogeneity and providing transparency makes programming distributed systems much easier to do!

Programming with Middleware

- Programming with Middleware
  - Do not have to learn a new programming language! (Usually)
  - Use an existing one already familiar with: C++, Java, C#, Ada, (yuk) Visual Basic, (yuk) COBOL
- Ways to Program with Middleware
  1. Middleware system provides library of functions (Linda, others)
  2. Support directly in language from beginning (Java and JVM)
  3. External Interface Definition Language (IDL) that "maps" to the language and generates local "proxy"

Kinds of Middleware

- Distributed Tuples: (a, b, c, d, e)
  - Relational databases, SQL, relational algebra
  - Linda and tuple spaces
  - JavaSpaces (used by Java Jini)
- Remote procedure call (RPC
  - make a function call look local even if non-local
- Message-Oriented Middleware (MOM)
  - messages and message queues
- Distributed Object Middleware
  - Make an object method look local even if non-local
  - CORBA
  - DCOM/SOAP/.NET
  - Java RMI

Kinds of Middlware (cont.)

Different middleware systems encapsulate and integrate the different kinds of resources with varying degrees:

Middleware Category	Communication	Processing	Storage
Distributed Tuples	Yes	Limited	Yes
Remote Procedure Call	Yes	Yes	No
Message-Oriented MW	Yes	No	Limited
Distributed Objects	Yes	Yes	Yes

For many (non-database) applications, and supporting adaptation, distributed object middleware is better because it is more general

Middleware and Legacy Systems

- Legacy systems are a huge problem (and asset) in industry and military domains!
- Middleware often called a "glue" technology: integrated "legacy" components
  - Much distributed programming involves integrating components, not building them from scratch!
- Middleware's abstractions are general enough to allow legacy systems to be "wrapped"
  - Distributed objects are best here because more general
  - End result: a very high-level "lowest common denominator" of interoperability

Outline

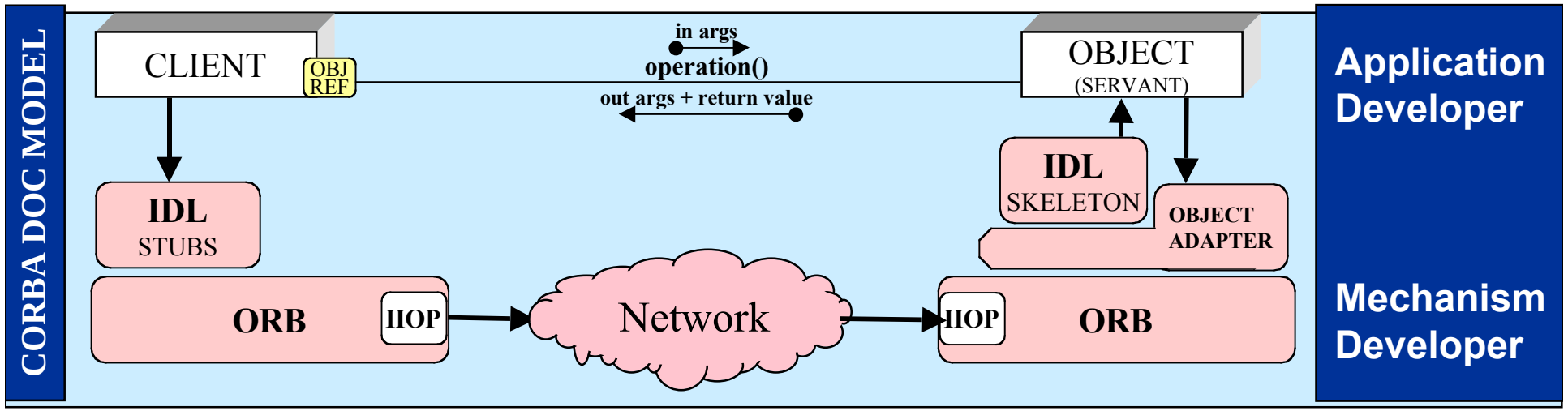
- Middleware: Definition and Benefits
- **Examples of Middleware**
  - **CORBA**
  - **.NET**
- Middleware and Adaptation
- Multi-Layered Middleware
- Middleware and Dependability

On Commercial Middleware and Research

- Creating a comprehensive, useable MW system is a huge undertaking, even with no adaptation or dependability
- Many military and industrial organizations require (or at least attempt) leveraging commercial off-the-shelf (COTS) MW for economic reasons
- To have impact, IMHO researchers thus need to interact with COTS MW at some level
  - Interoperate with it
  - Wrap it
  - Add layers above it for adaptation and/or dependability
  - Insert layers below it for managing the environment (net mgmt…)
  - Evaluate risks of using COTS MW for challenging/critical domains (space, aviation, military C2, military embedded, …)

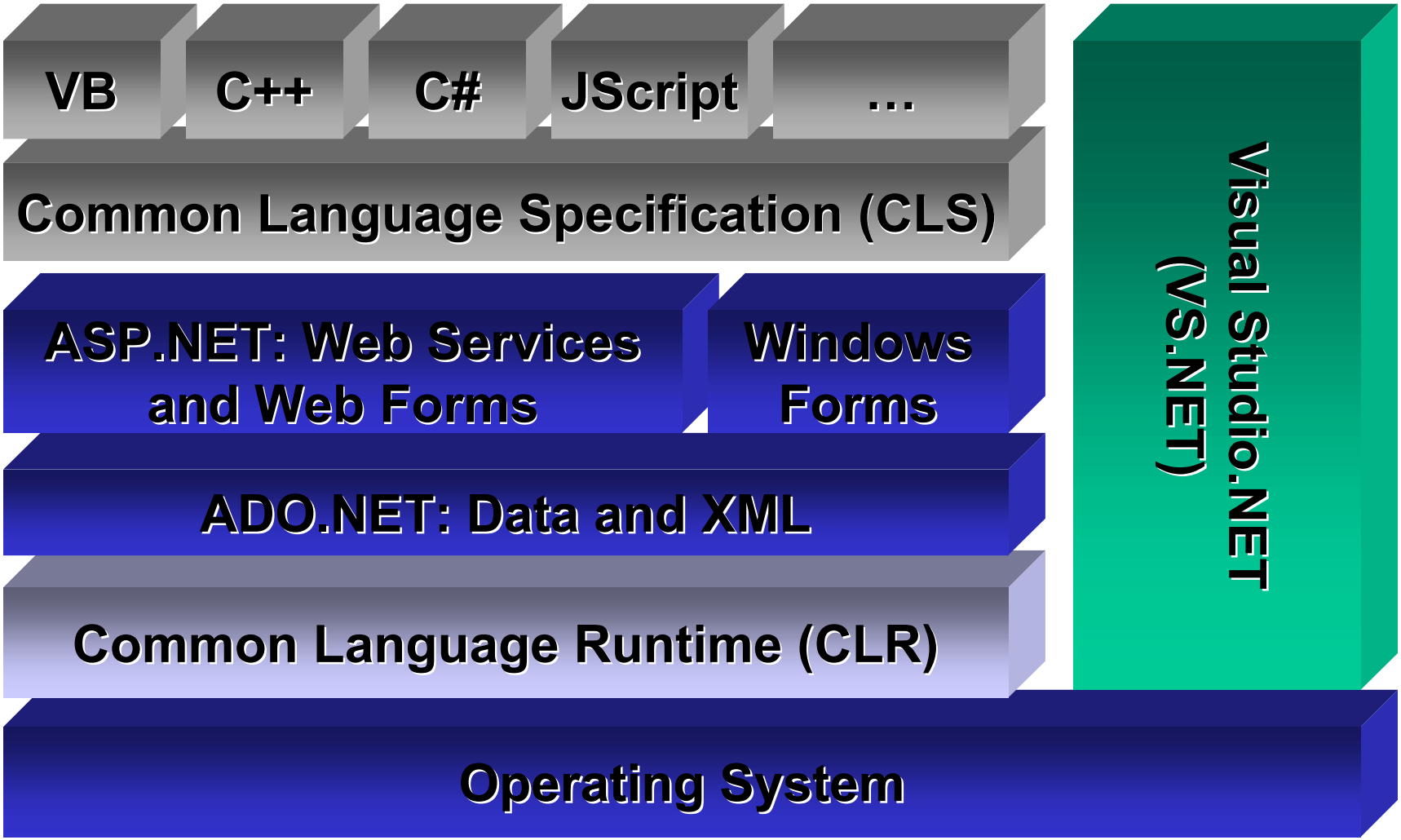
I.e., do value-added research to enhance COTS MW apps!

Simplified CORBA Runtime Components



Note: path from stub (a.k.a. proxy) to object is managed by ORB on behalf of that client-object interaction…

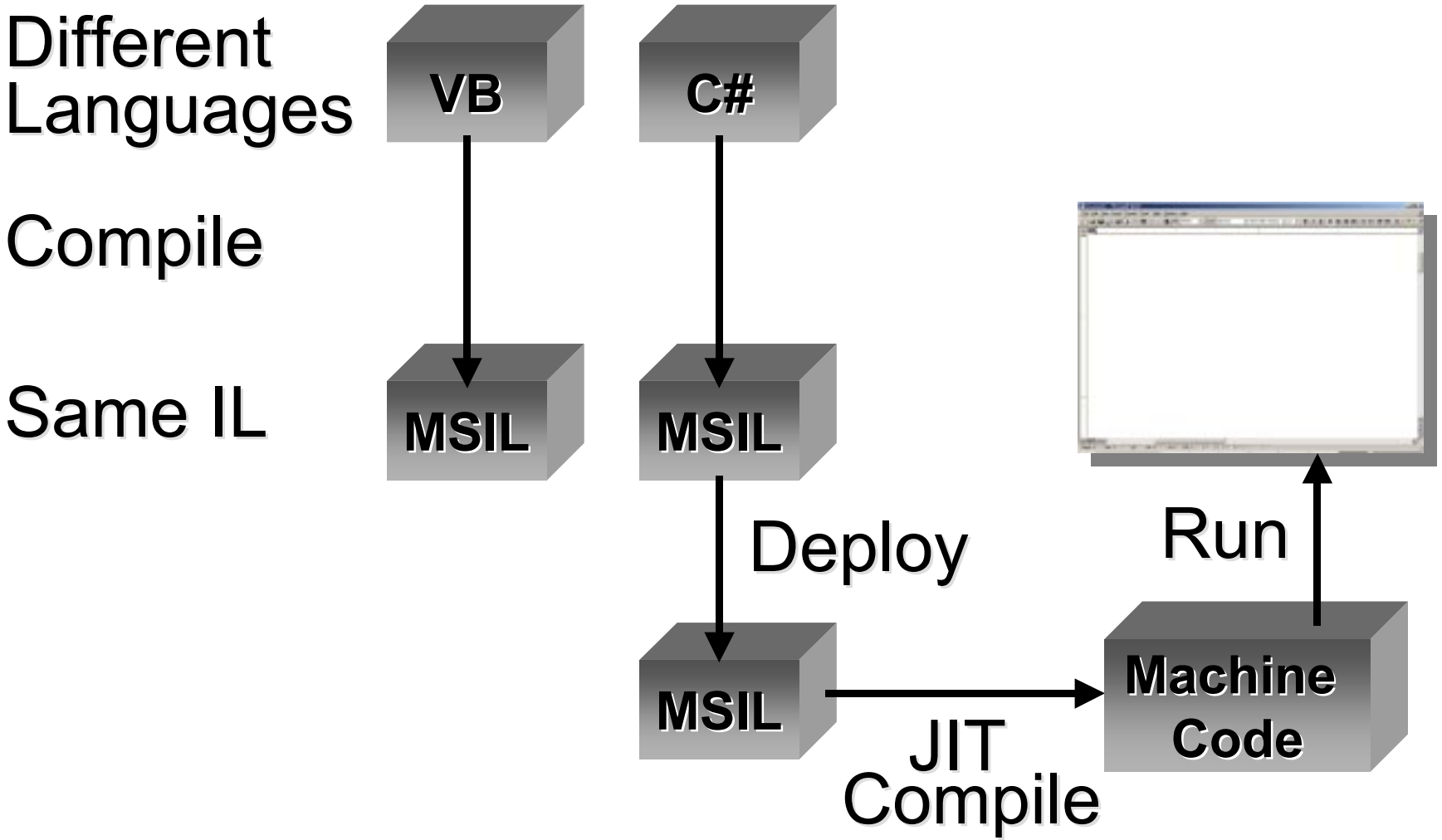
.NET Framework Overview



Notes: 1) CLR ≈ JVM **2) "Hidden" stuff via VS.NET**

(Courtesy of Microsoft)

.NET Framework Overview



Outline

- Middleware: Definition and Benefits
- Examples of Middleware
- **Middleware and Adaptation**
- Multi-Layered Middleware
- Middleware and Dependability

Application-Level Adaptation Choices

- How can distributed applications (including the middleware that supports them) 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, i.e. tradeoffs between storage and processing and communications)
  - Take longer
    - reschedule for later
    - tolerate finishing later than originally expected
  - Do less
    - lower precision
    - lower accuracy
    - smaller quantity

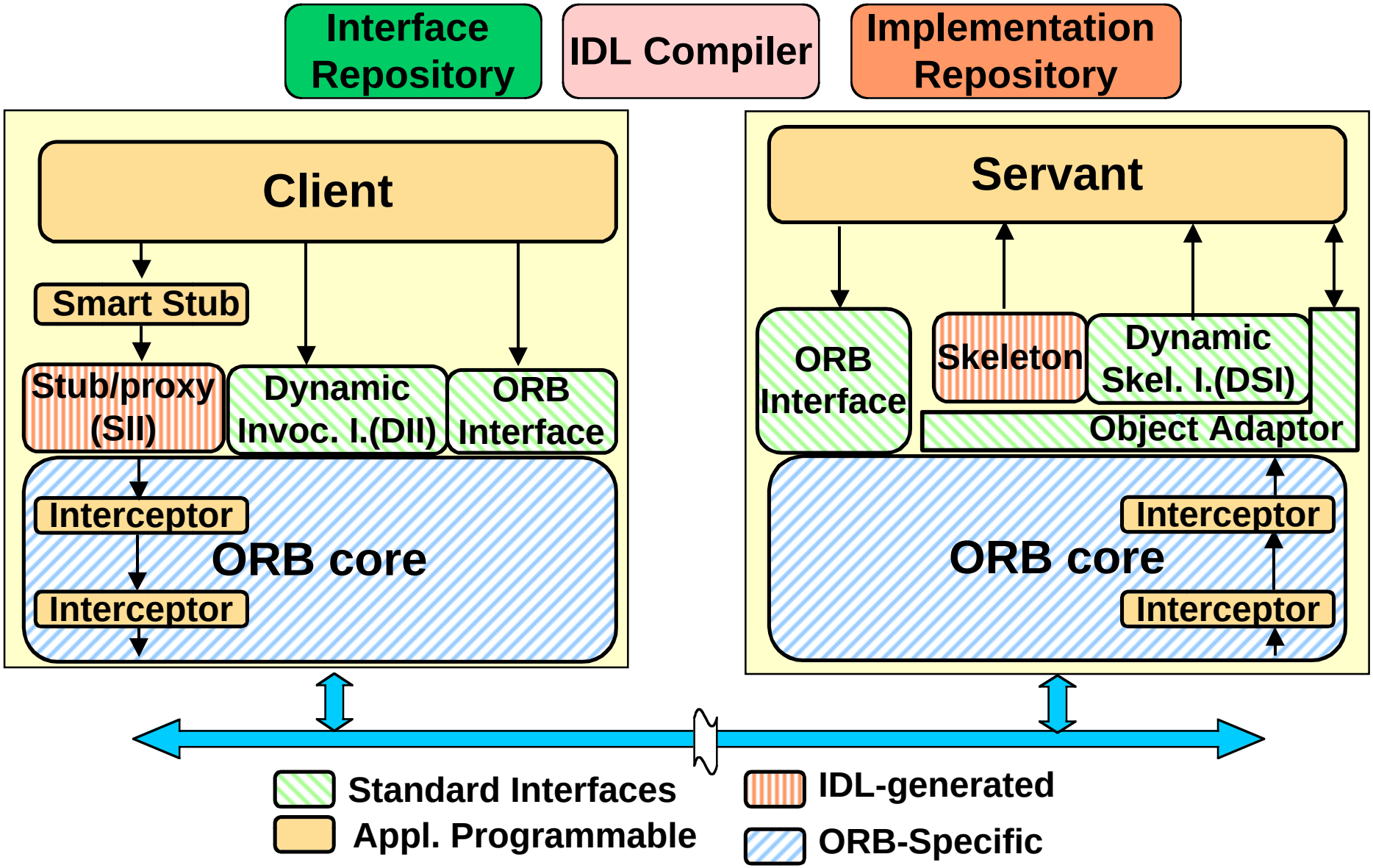
Application-Level Adaptation Choices (cont.)

- Note the multiple possible layers of adaptation:
  - Client application
  - Above the ORB core on client-side
  - Inside the ORB (if has hooks)
  - In the network (via "gateway", bandwidth management)
  - Above the ORB core on server-side
  - Server application
- Adaptation can occur
  - *in-band*: triggered synchronously to an invocation
  - *out-of-band*: triggered asynchronously, not by any invocation
- Premise: supporting all the above choices is helpful!
  - Need help from the application in choosing right tradeoffs
  - **Need lots of help from the middleware to keep the application's job simple and high-level ("Awareness without Pain")**

Middleware and Adaptivity

- Why is middleware helpful/useful for adaptation?
- High enough level to capture application's structure (nicely encapsulated objects/components and their interactions)
- Lots of ways middleware can implement its high-level abstraction
  - Reflection-based adaptation a useful way to organize this [Cou02]
  - Aspect-Oriented Programming and variants/mutants also help
  - Multi-layered middleware (later slides…)

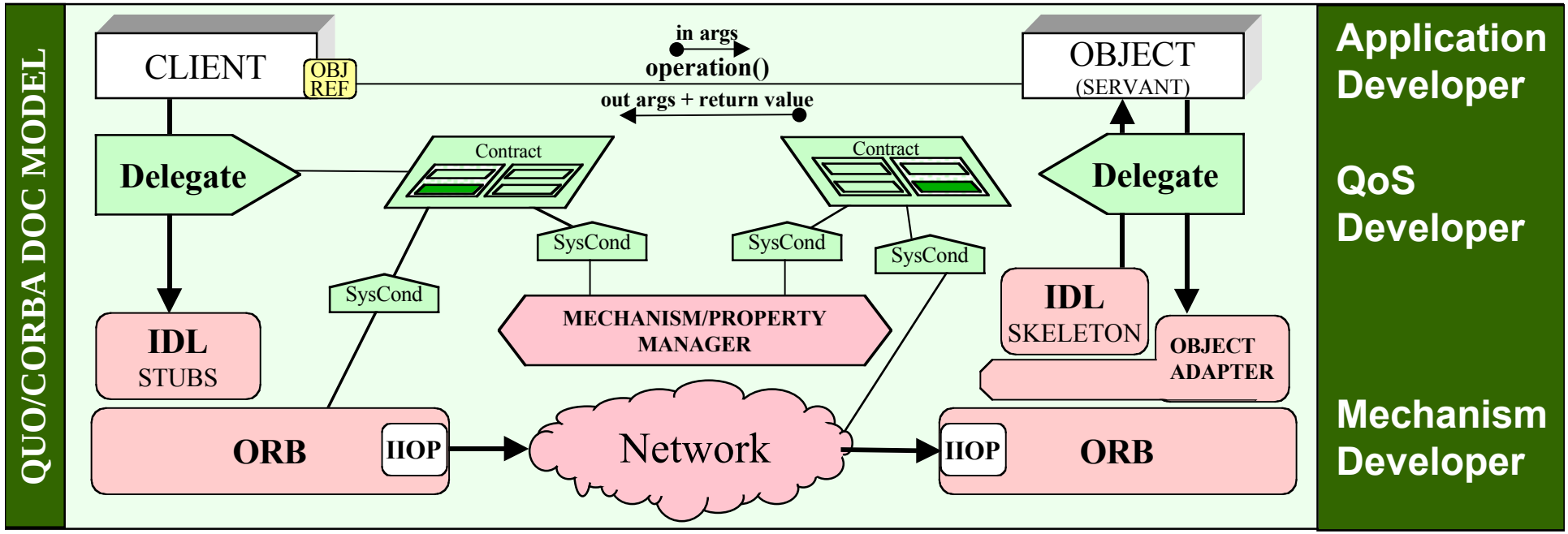
CORBA System Builders' Hooks



Note: IMHO few middleware systems give so many hooks…. for adaptation, QoS, …

(Courtesy of S. Yajnik)

Quality Objects (QuO) Adaptation Example



- QuO 2.0 in a nutshell
  - Delegate: adaptive stubs
  - Contract: specify level of QoS desired and regions of operation
  - SysCond: way to plug in status info into contract or control mechanisms from contract
  - See [Quo02] for more info (Courtesy of BBN)
- QuO and Adaptation
  - QuO supports adaptation at many layers, locations, etc.
  - Delegate adaptation: first hard-coded, later SDL added when patterns became clear
  - Contracts: first simple region language, then FSA, others later

Outline

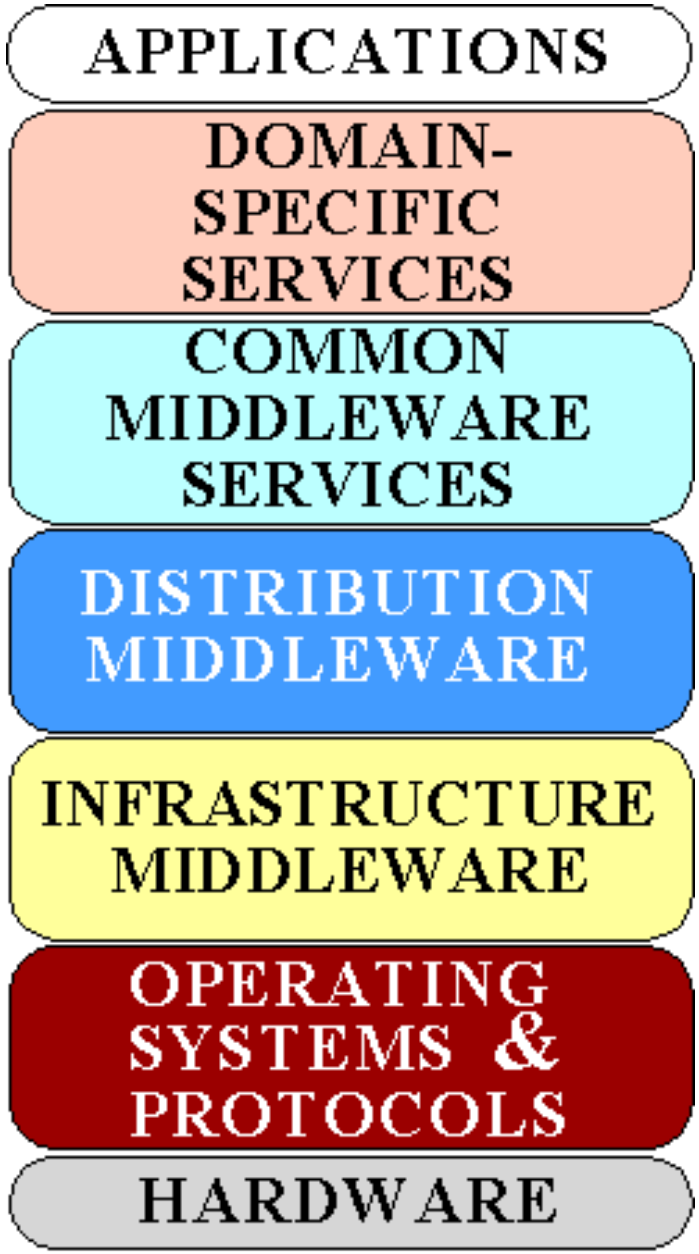
- Middleware: Definition and Benefits
- Examples of Middleware
- Middleware and Adaptation
- **Multi-Layered Middleware**
- Middleware and Dependability

On Layers and Middleware

- Discussion issue for panel: make one middleware framework do all, or layer to (re)use others' middleware???
- Grist for panel discussion:
  - "What is it about distribution that requires new abstractions? Can't we just use traditional abstractions to factor out the complexity of distributed programming and consider distribution as an implementation detail? …. This is the philosophy underlying the notino of *remote procedure call* and its successor, *remote method invocation*. The idea is to give the illusion that the application was not distributed and that remote objects look as if they were local. This is a very misleading view, and is one of the reasons why none of CORBA, DCOM, and RMI does really make sense in a real genuinely distributed setting…." [LPD02], emphasis mine

  Issue: should we reject COTS MW because it sometimes violates transparencies, or should we employ higher layered middleware to compensate?

One Middleware Layering Taxonomy (US DARPA)



- Infrastructure MW
  - Encapsulates core OS Comm. and concurrency services (sometimes enhances them too)
  - Examples: JVMs, ACE, group comm.
- Distribution MW
  - Provides rich distributed object model that supports much heterogeneity and transparency
  - Examples: CORBA, DCOM, .NET., Java RMI
- Common MW Services
  - Adds high-level, domain-independent reusable services for events, fault tolerance, security,
  - Examples: CORBAServices, Eternal
- Domain-Specific Services
  - Services and APIs tailored to (and reusable only within) certain domains (health care, telecommunications, etc)
  - Examples: CORBA Domain Interfaces, Boeing Bold Stroke architecture [Sha98]

(Courtesy of D. Schmidt; see [SKY+00,SSM+01])

Multi-Layered Adaptation Example: WSOA

- Weapon Systems Open Architecture (WSOA)
- Adaptive architecture, prototype has flown in an experimental aircraft
- Supporting F-15 fighter and command+control (C2) collaboration in-flight, to retarget and rehearse
- Very brief overview follows, see [Loy01,Cor00] for more details
- Note: multiple layers of adaptation, not same MW layers on last slide…

Fighter Information Technology Past and Future

BOEING

- Historical approach
  - Information sources primarily constrained to onboard, deterministic resources
  - Functionality limited to static algorithms, cyclic processing, worst case, static scheduling
  - Expensive to maintain and test
- Future needs
  - Offboard/onboard integration of data and functionality
  - Non-deterministic communication and functionality
  - Lower cost, rapid change, user/mission customization
  - Integration with Joint Forces Global Info Grid to exploit information superiority

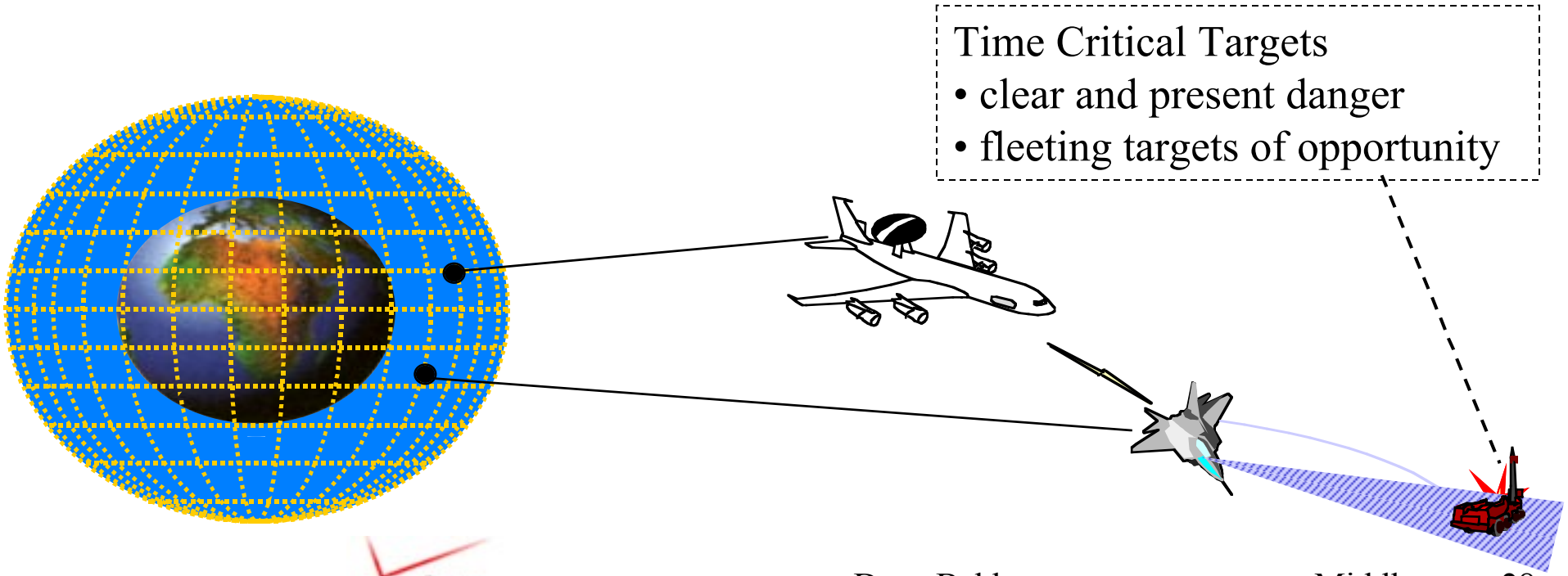
Simplified WSOA Context

Airborne C2 Node

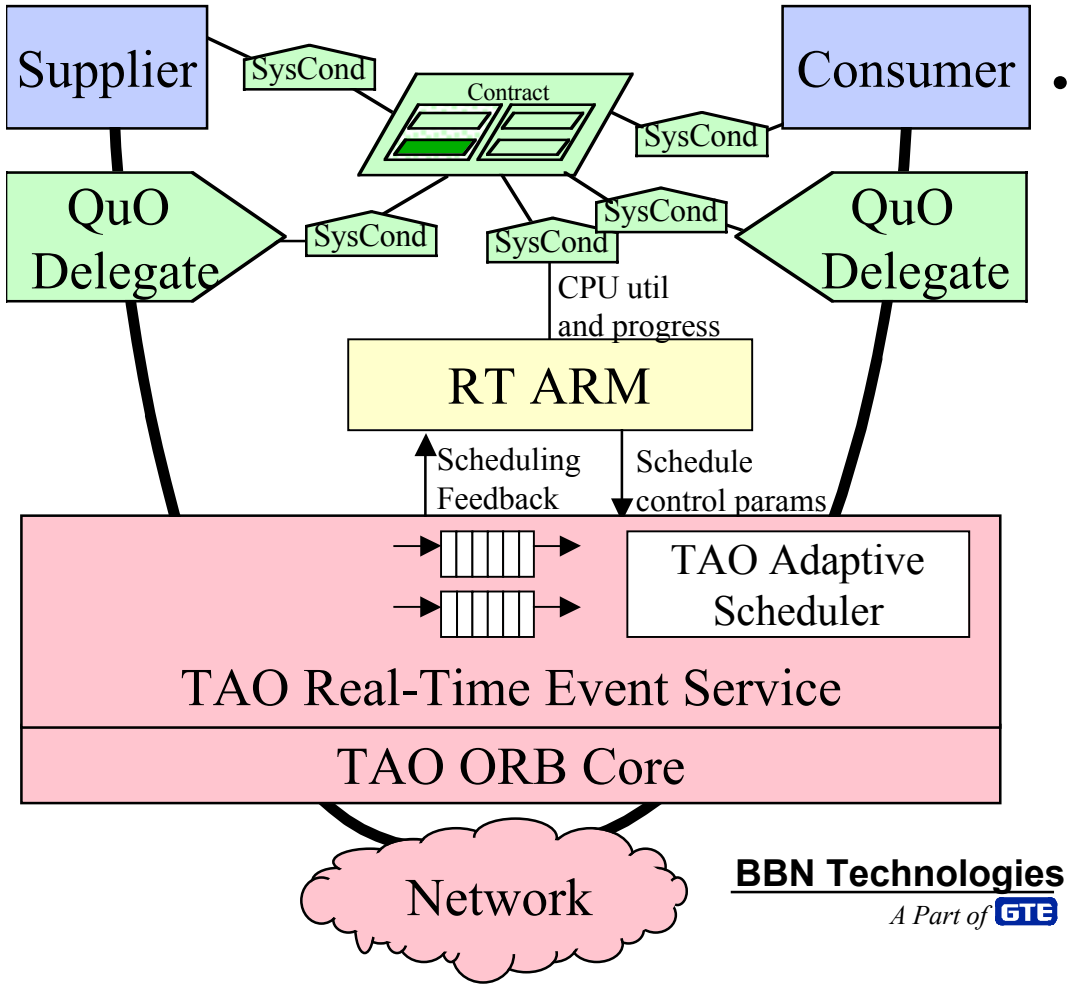
- Compiles Virtual Target Folder (VTF)
- Retasks enroute to strike
- Collaboration with F-15 to replan route
- CORBA IDL Interface

Warrior in F-15

- "Browser" requests for target and imagery data
- Collaboration with C2 node for target review and mission replan
- Previews updated mission enroute
- CORBA IDL Interface

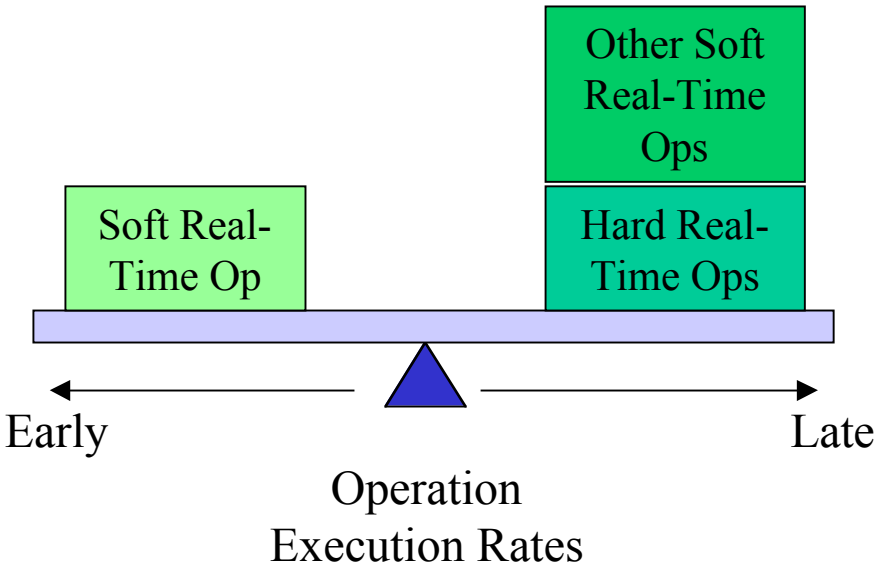


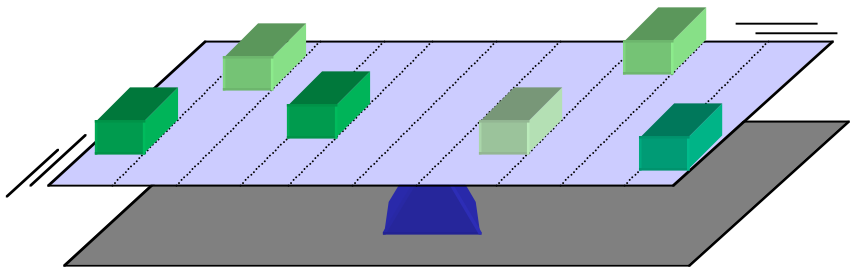
Providing Robust End-to-End QoS Management with QuO



- Challenge: responding appropriately to either transient or persistent overloads and opportunities
- Approach: providing real-time adaptive resource management on multiple scales of time and distribution
  - •Immediate local adaptation in TAO's Real-Time Event Channel service, via a hybrid static/dynamic scheduling strategy
  - •Medium time-scale resource-centric adaptation by resource managers, e.g., RT ARM (admission control adjustments and adaptation)
  - •Broader adaptive application-centric reconfiguration in the face of persistent overloads, managed by the QuO framework at the minimal appropriate time scale

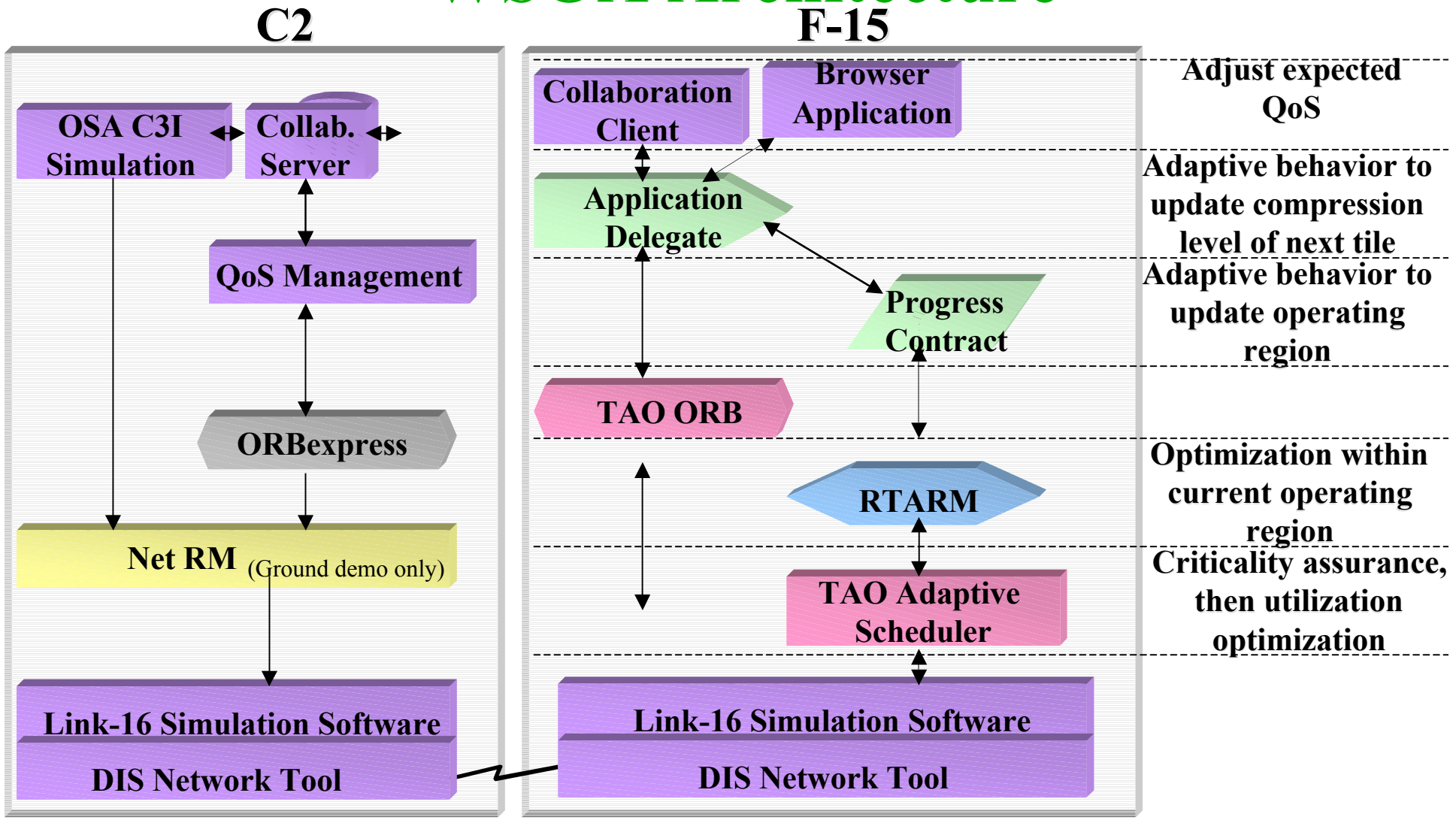
RT-ARM CPU Management





- Updated approach is a “sliding box” algorithm, which attempts to maximize the number of operations through load balancing
- Operation execution rates are adjusted according to their current scheduling progress, measured against the availability of CPU cycles
- Soft real-time ops are adaptable, while hard real-time ops have fixed maximum bounds
- Soft real-time ops are over-scheduled and rates adjusted to compensate for actual results

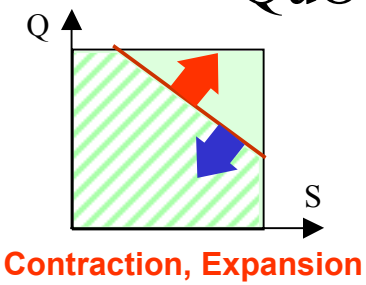
WSOA Architecture



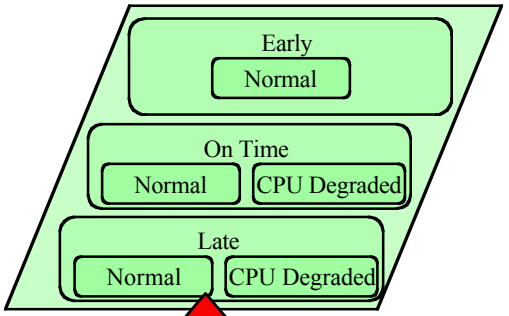
Boeing (StL)
Oregon Graduate Institute
Honeywell Tech. Center
BBN
Washington University (StL)

WSOA QoS Control Flow

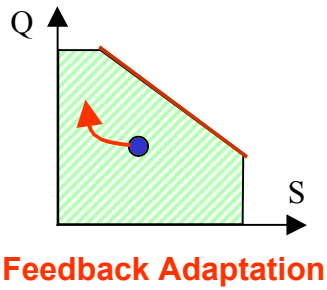
QuO



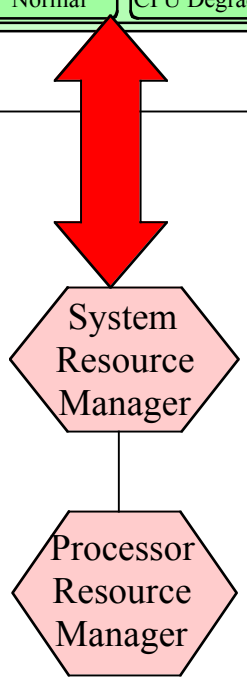
- Manages application progress
  - Early, On-Time, or Late for each operation
- Defines operating regions
  - Range of rates for each operation
- Also handles image tiling (not shown)



RT-ARM



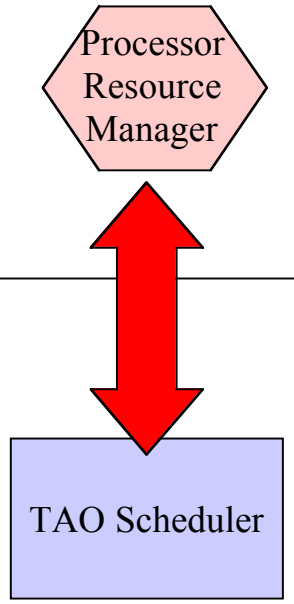
- Manages QoS parameters within the given operating regions
  - Adjust rates within defined ranges for each operation
- Reports when operating region is violated (or will be violated)



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

- Middleware: Definition and Benefits
- Examples of Middleware
- Middleware and Adaptation
- Multi-Layered Middleware
- **Middleware and Dependability**

Middleware and Dependability

- [… Too many things to say about dependable middleware, too little time….]
- Middleware's rich abstractions can be used to implement dependability in many ways….
- A few examples follow, but some (of many) others worth mentioning: MAFTIA, Eternal, Immune, FT-CORBA, DACE, DOORS, X-ability, TTP, MARS, ….. and of course Delta-4's pioneering work

QuO Gateway: Dependability Mechanism

Dependability mechanism inserted between ORBs (in the network)

- 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

Note: this implements the *mediator* pattern

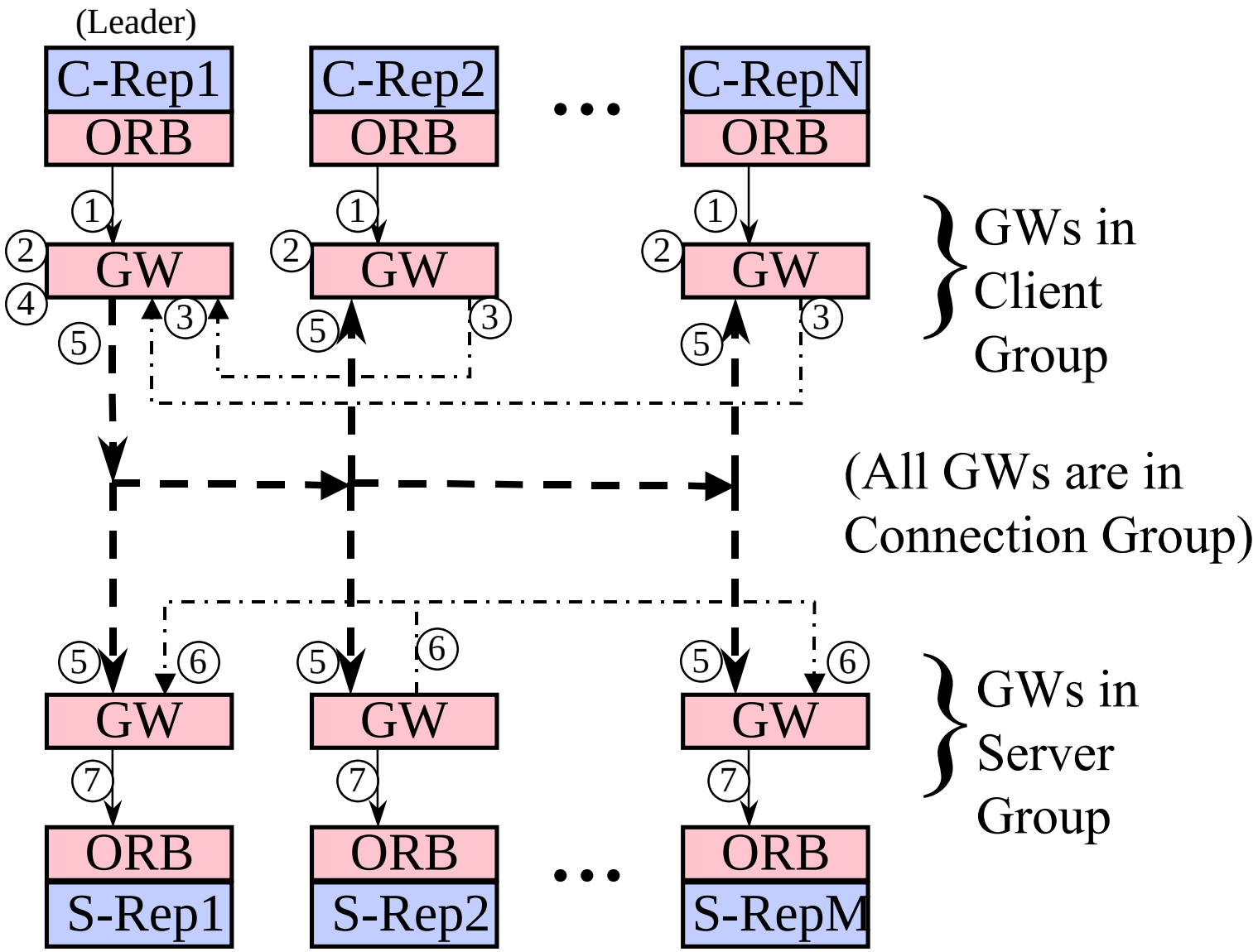
AQuA Handlers: Programming the Gateway

Design space of dependable computing has many variables! Lots of different ways to use group communication between two groups:

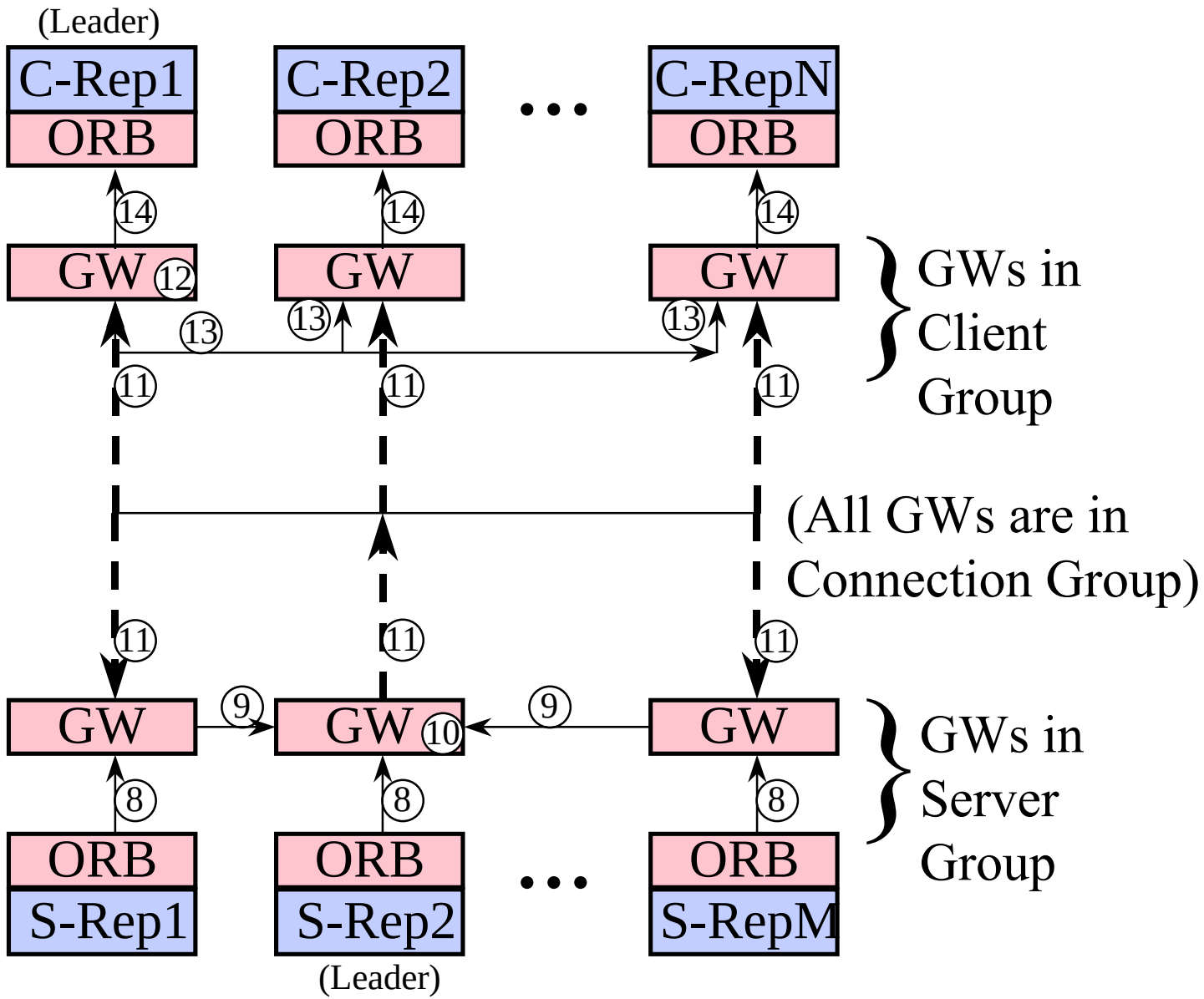
- Client group has leader or has no leader
  - how much do you trust client group?
- Server group has leader or has no leader
- Multicast strengths (total, causal, FIFO, …) used in connection group
- Which members of client and server groups are in "connection group"
- Location and algorithm for voting/collation
- 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

All above can be handled in MW; most can be ~transparent to apps

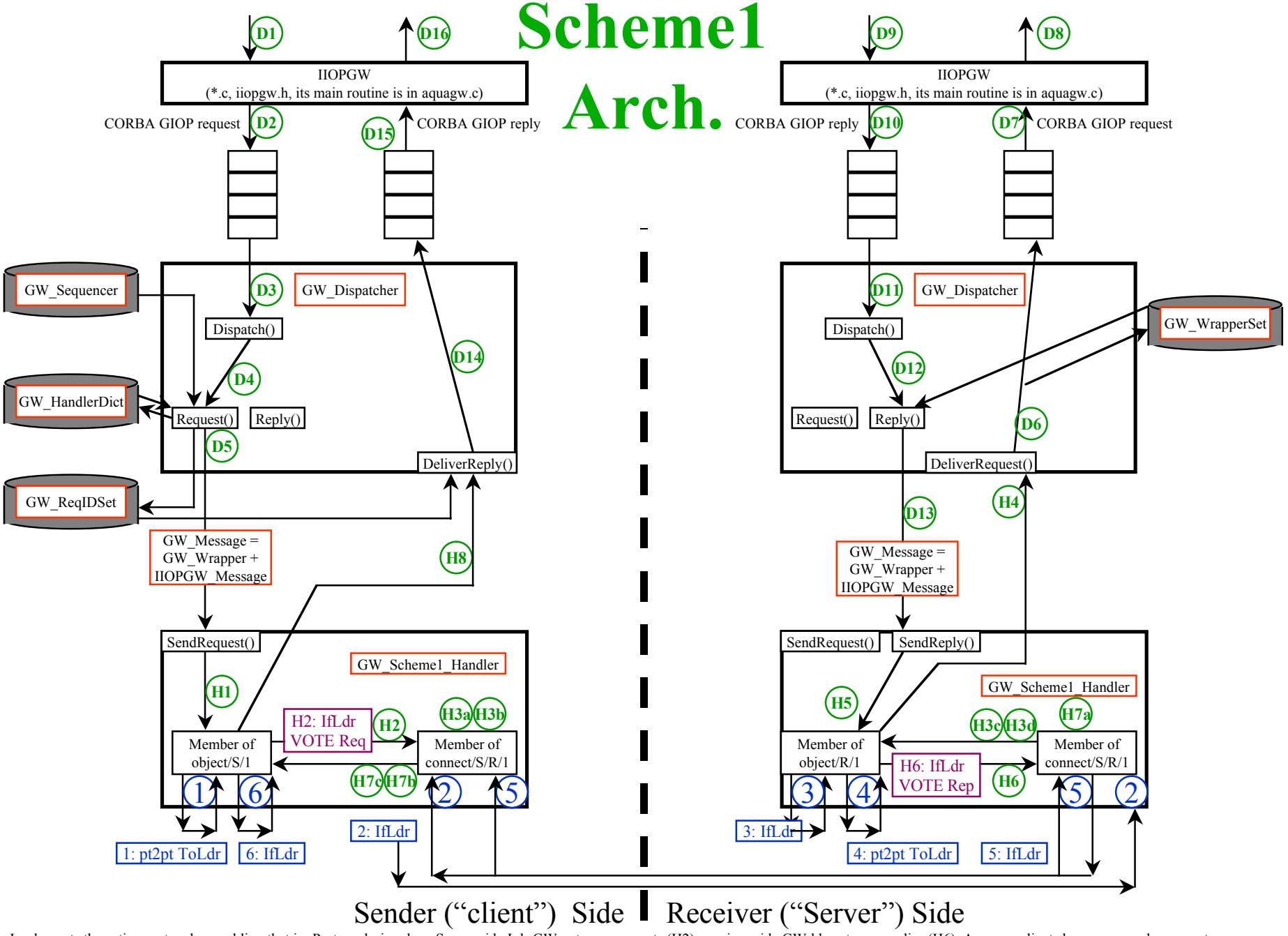
AQuA Scheme1 Request Steps



AQuA Scheme1 Reply Steps



Scheme1 Arch.



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

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

References (Not comprehensive!!!!!)

- **[Bak03]** D. Bakken, "Middleware", 5-page article by Dave Bakken in *Encyclopedia of Distributed Computing*, Kluwer, to appear. www.eecs.wsu.edu/~bakken/middleware.pdf
- **[Cor00]** D. Corman, "Weapon Systems Open Architecture", http://quite.teknowledge.com/aboutQUITE/meetings/121200_PI_Meeting/WSOA.pdf
- **[Cou02]** G. Coulson, "What is Reflective Middlware?", IEEE Distributed Systems Online, October 2002, http://dsonline.computer.org/middleware/RMarticle1.htm
- **[LPD02]** EPFL Distributed Programming Laboratory (LPD) , "DACE: Distributed Asynchronous Computing Environment", available via http://lpdwww.epfl.ch/research
- **[LSZ+01]** J. Loyall *et al*, "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications", ICDCS'01.
- **[Quo02]** http://quo.bbn.com
- **[Sha98]** D. Sharp, "Reducing Avionics Software Cost Through Component Based Product Line Development", Software Technology Conf., Salt Lake City, Apr.1998.
- **[SKY+00]** D. Schmidt *et al*, "Developing Next-Generation Distributed Applications with QoS-Enabled DPE Middleware", *IEEE Communications Magazine*, Oct. 2000.
- **[SSM+01]** D. Schmidt *et al*, "Toward Adaptive and Reflective Middleware for Network-Centric Combat Systems", *Crosstalk: The Journal of Defense Software Engineering*, November 2001.

Acknowledgements

- Doug Schmidt of DARPA and Vanderbilt provided the multi-layered middleware slide
- The CORBA hooks slide was adaptated from a FTCS-29 tutorial by Shalini Yajnik of Lucent
- The WSOA slides were provided by Boeing Phantom Works in St. Louis
- The .NET slides were provided by Microsoft

All the above were used with permission

Conclusions

- MW == "**A layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system**" [Bak03]
- MW has many practical benefits, including greatly simplifying programming distributed systems, by (largely) providing some transparencies while masking heterogeneity
- MW (especially OO-MW) is a rich and high-level enough of an abstraction to support a wide variety of adaptation and dependability schemes; many provided (largely) transparently to the application program
- Multiple layers of MW are often a good way to
  - subdivide a complex MW framework
  - Levereage COTS MW
  - Integrate other researcher's mechanisms

Outline

- Middleware: Definition and Benefits
- Examples of Middleware
- Middleware and Adaptation
- Multi-Layered Middleware
- Middleware and Dependability
- **Backup Slides**

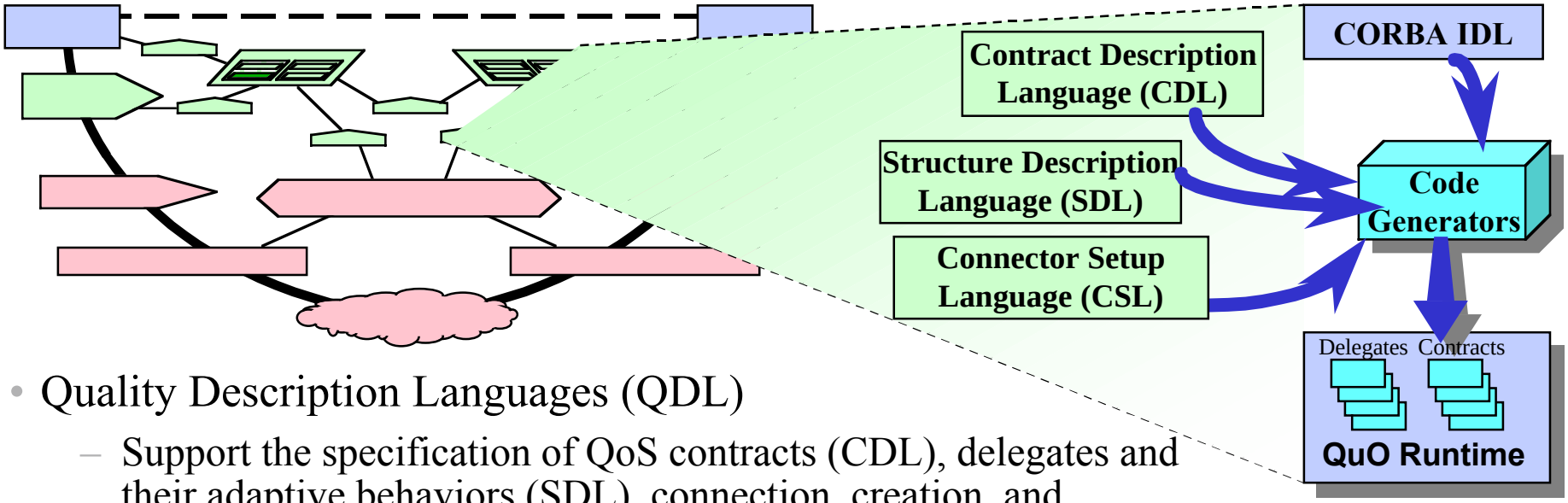
MicroQoSCORBA

- A QoS-Enabled, Reflective, and Configurable Middleware Framework for Embedded Systems
  - PI: David Bakken, Washington State University
- Issues: how to create middleware let developer choose what to strip out and constrain to get middleware small
- Novelty:
  - Only middleware that is tailored to both the application's and the environment's constraints
  - Very fine granuarity of configuration choice
- QoS so far
  - Fault Tolerance: Done (Kevin Dorow)
  - Security: Work in Progress (A. David McKinnon)
  - RealTime: Work in Progress (Dr. Wes Lawrence, others TBD)
- Funding: Cisco, NSF, likely Boeing soon

Voting and Data Fusion in Middleware

- Voting and data fusion in middleware (in the presence of heterogeneity)
  - PI: David Bakken, Washington State University
- Supporting a wide variety of algorithms via programmable primitives in a middleware-level VM
- Funding: DARPA OASIS, Air Force (new security work too)
- See DSN-2001 paper, DSN-2002 demo paper for more details

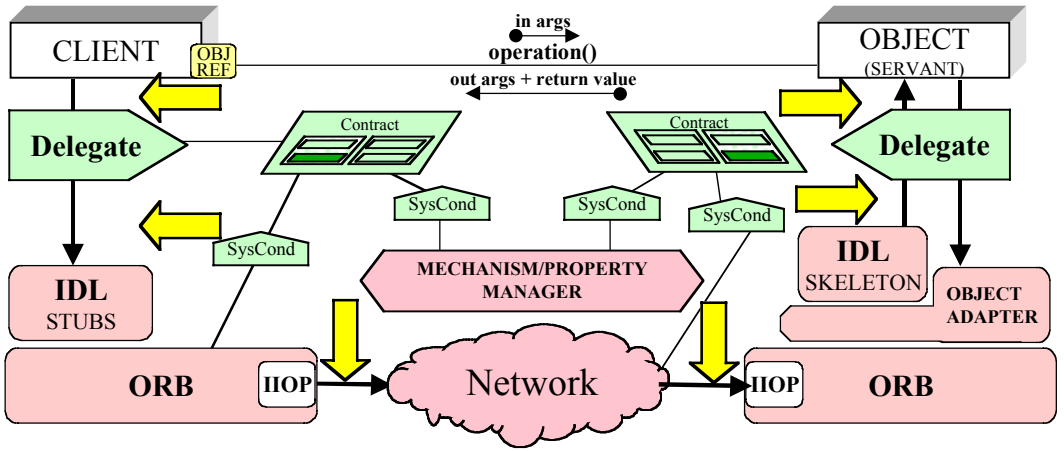
The QuO Toolkit Provides Tools for Building QuO applications

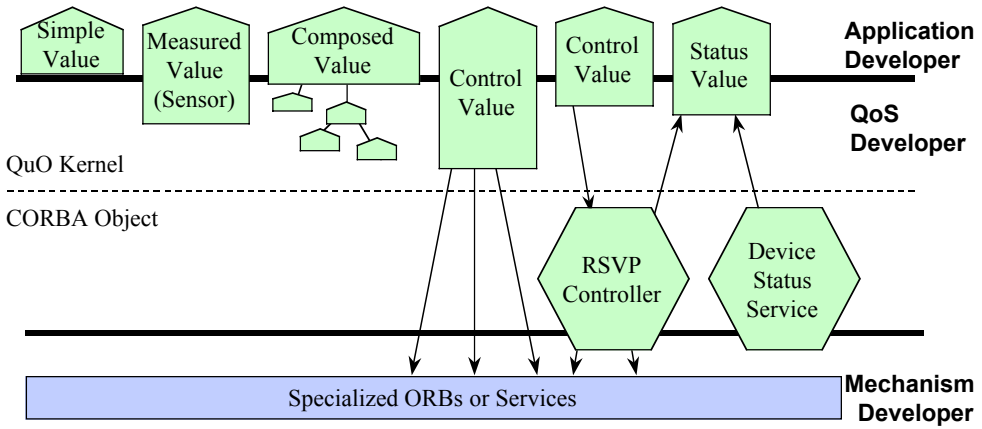


- Quality Description Languages (QDL)
  - Support the specification of QoS contracts (CDL), delegates and their adaptive behaviors (SDL), connection, creation, and initialization of QuO application components (ConnDL)
  - QuO includes code generators that parse QDL descriptions and generates Java and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

Measurement in QuO

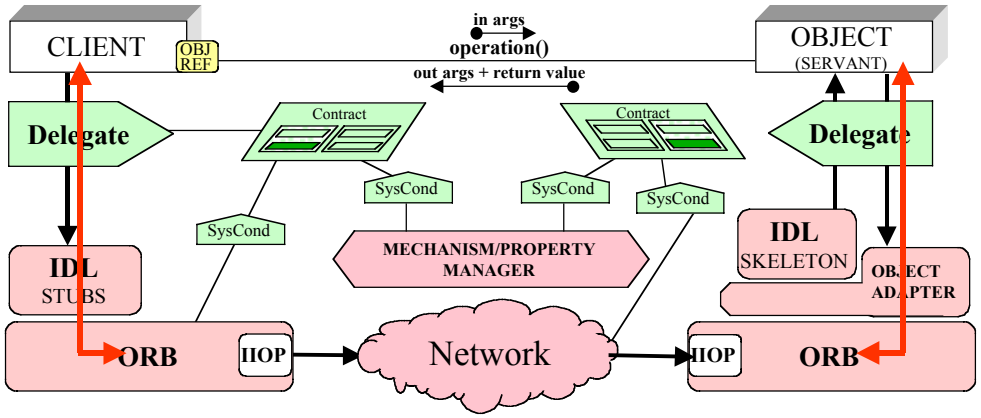
- *In-band* measurement handled by instrumentation
  - A structure is transparently passed along with the method call/return
  - Information can be inserted, read, and processed to record and evaluate method call statistics (e.g., the time spent in marshalling)
- *Out-of-band* measurement provided by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

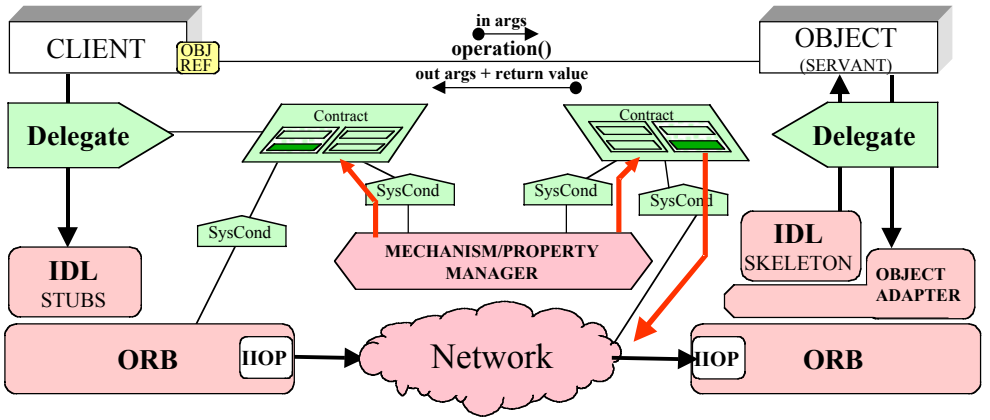




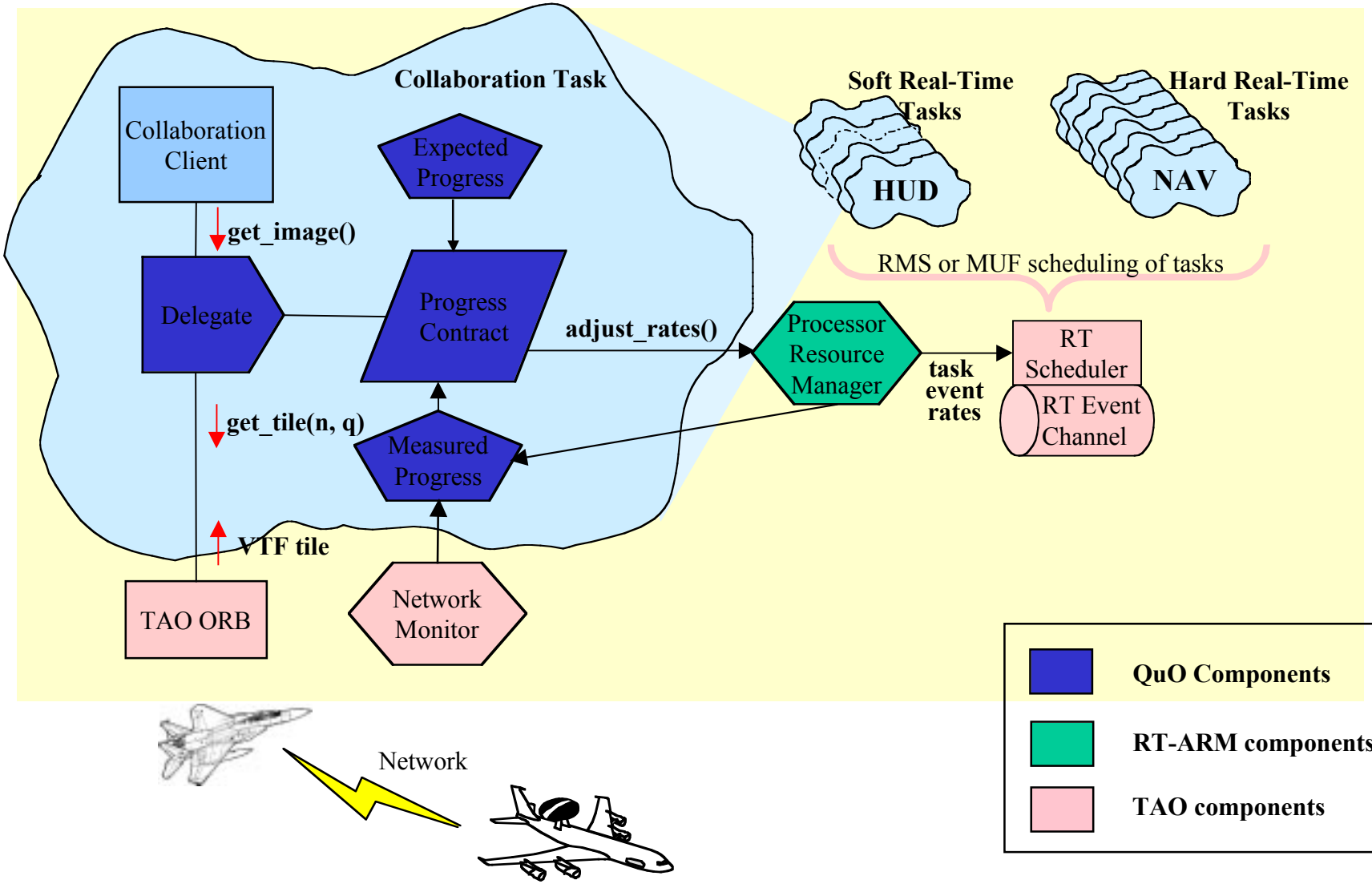
Adaptation and Control in QuO

- *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

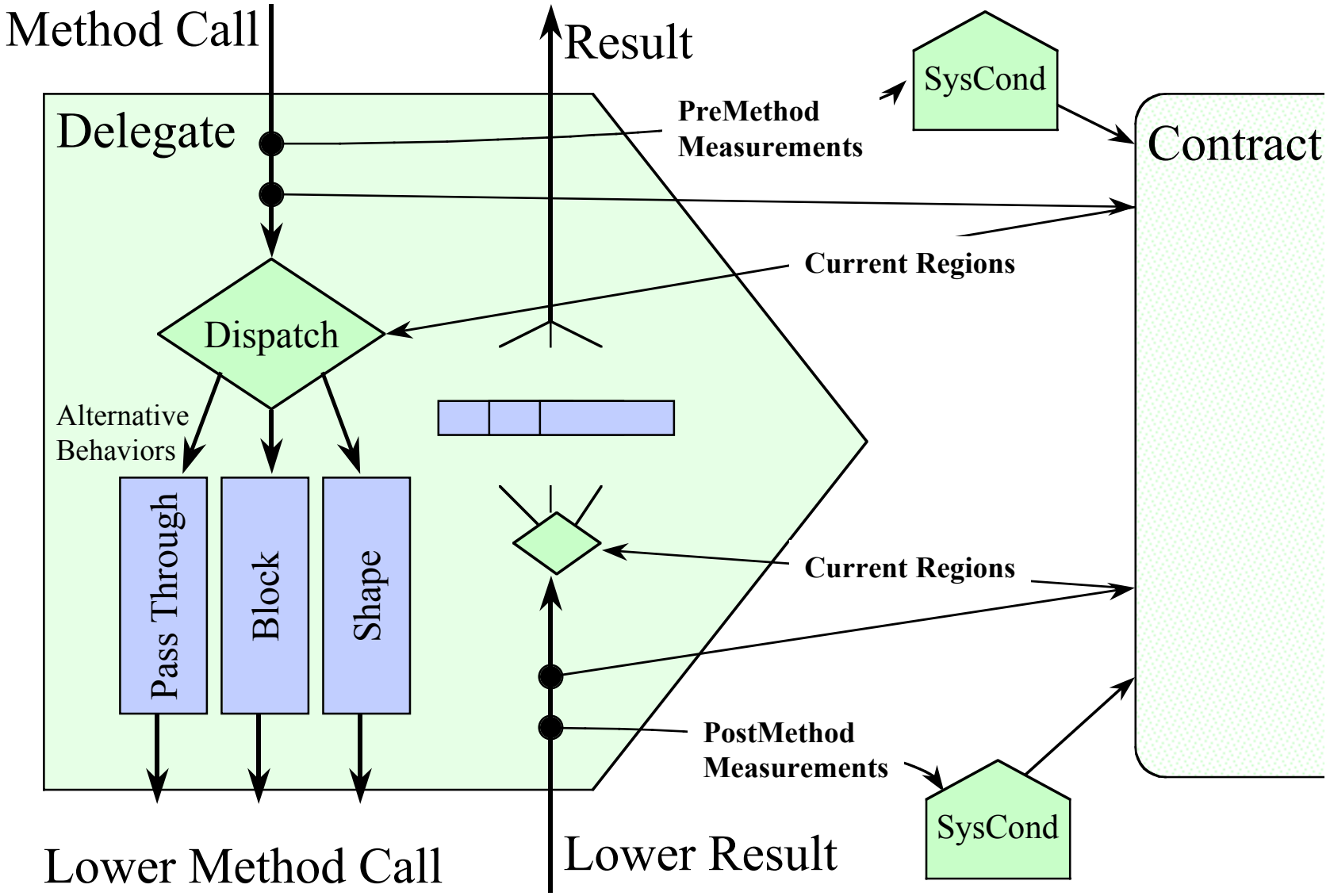




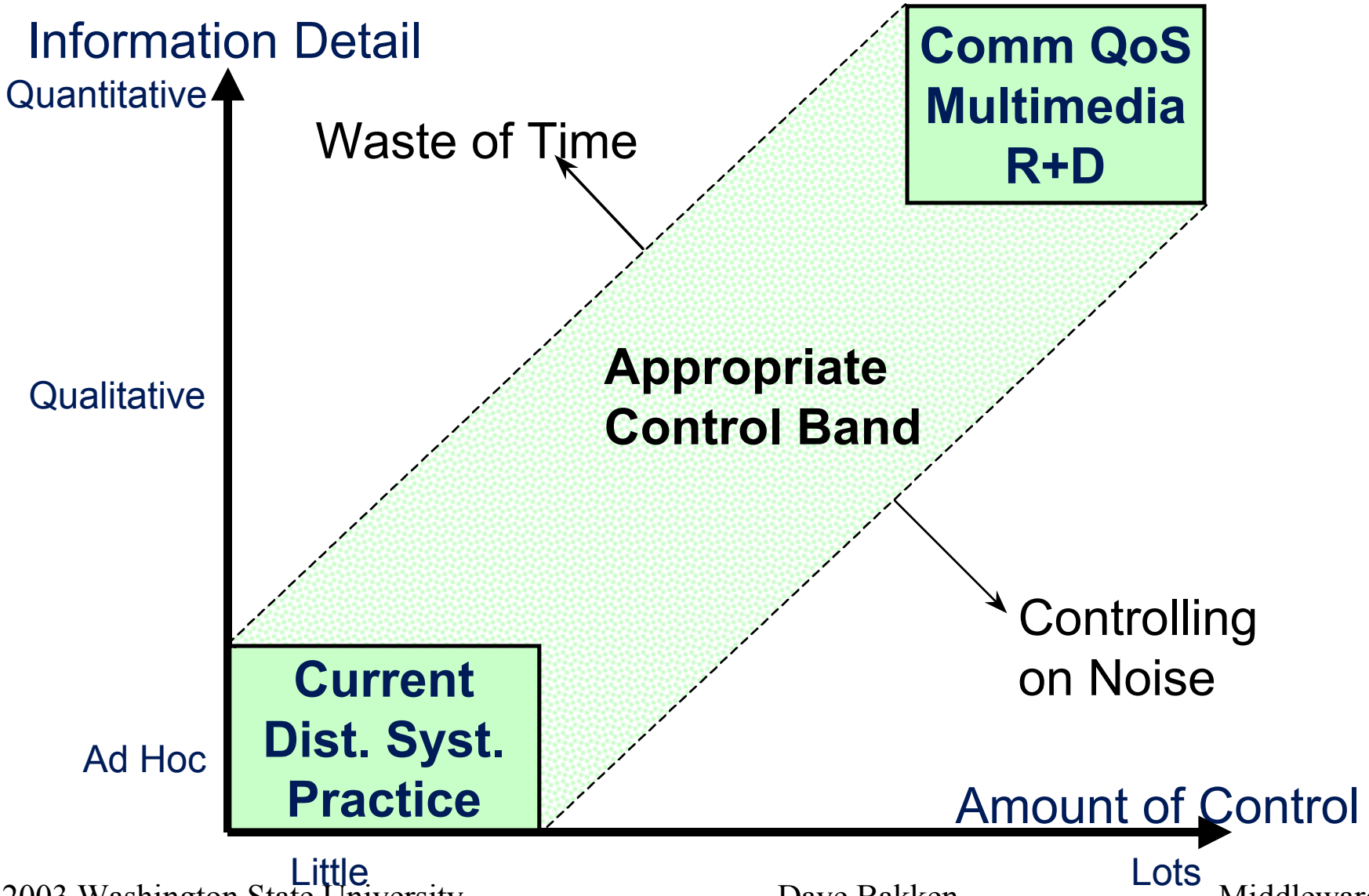
Adaptive Behavior Integrated with Advanced Resource Management in WSOA



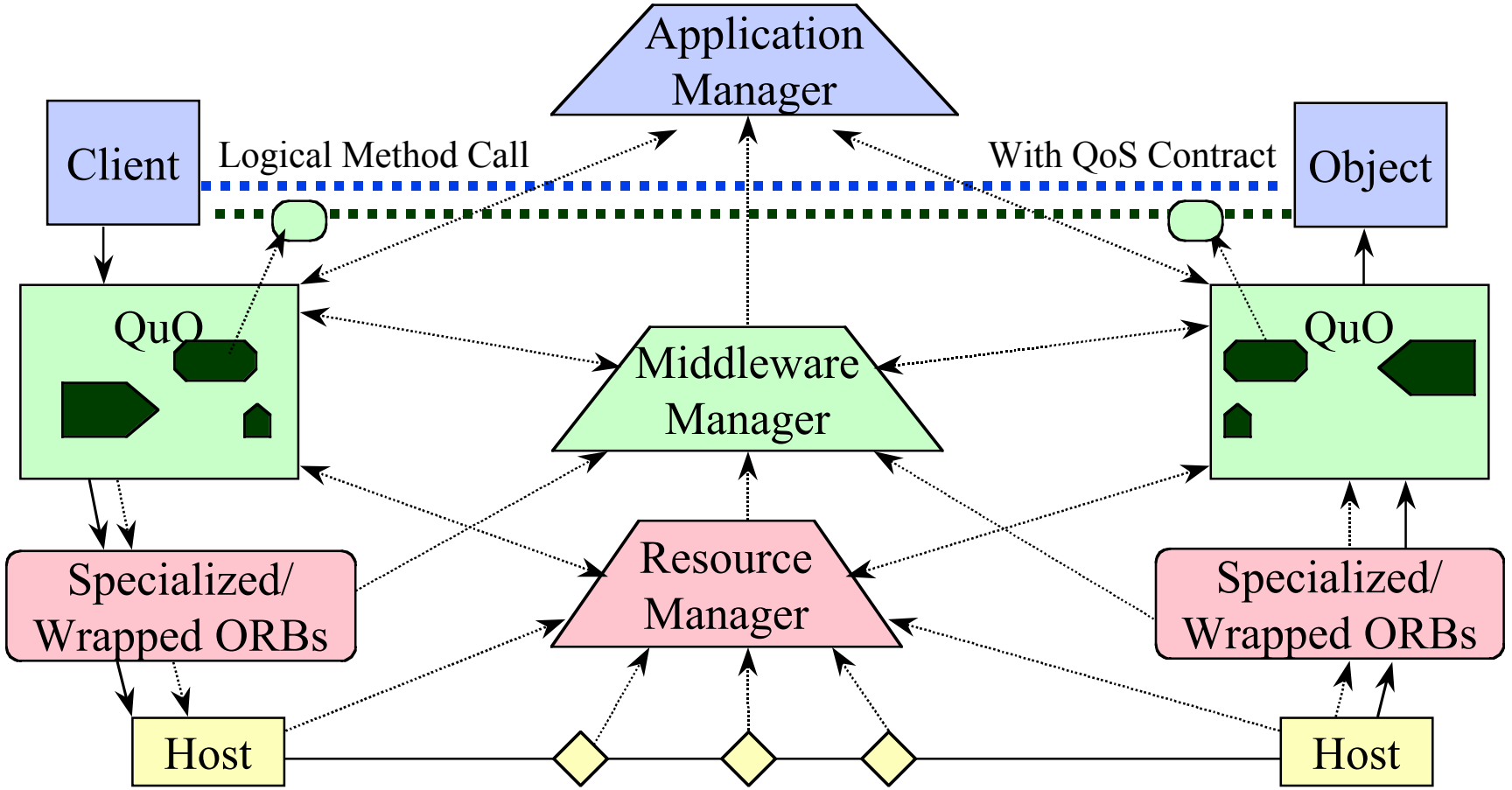
Delegates Change Their Behavior Based on Their Contract's Current Regions



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



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*