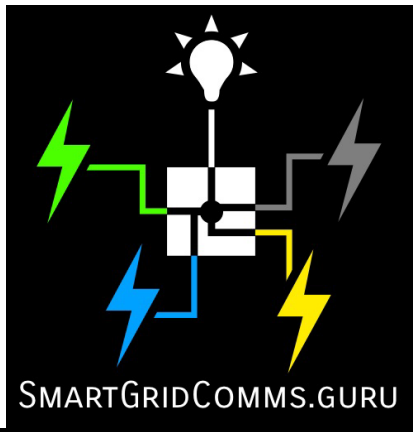
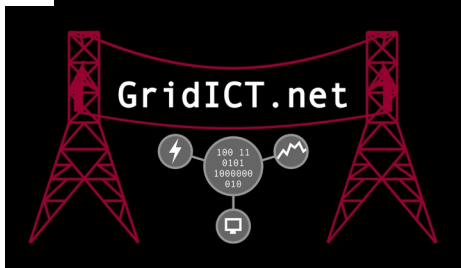
Middleware 101

Professor David E. Bakken, Ph.D.
bakken@wsu.edu
Washington State University (WSU) &
www.SmartGridComms.guru

Schweitzer Engineering Labs seminar
Pullman, Washington, USA
June 3, 2026
Updated November 4, 2026





WASHINGTON STATE UNIVERSITY

Outline

- Background
- Middleware Basics
- CORBA Remote Object Middleware
- Quality Objects: DARPA Middleware on Steroids
- Discussion

Example Local Object Invocation (in same process)

Caller:

```
// declare and init stuff
x = new int [100];
y = new util;// alloc+rtns ref
int flag, value2;
flag = y.sort(x, 100);
Value2 = SomeFunction(flag);
```

Callee:

```
// declare and init stuff
int util:sort(int [] a, int max) {
    // implementation of sort...
    return status;
}
```



- Potential assumptions:
  1. Object Invocation conventions between caller ("client") and callee
  2. In same address space (on same computer)
  3. In same programming language (virtually always for local calls)
  4. Same operating system for both caller and callee
  5. Same CPU type for both caller and callee
  6. Can transfer data and control quickly, effectively in zero time
  7. Both fail, or neither do (for the most part)
- **None** of these assumptions are always true in a distributed system!

Many Local Call Assumptions Untrue for Remote Calls

- Not a local object Invocation, so need more help
  - Need remote equivalent of local (assembler) calling conventions
  - In a longer class we study this "plumbing" much deeper
- Not in same programming language (can't assume)
- Not running same operating system for caller (client) and callee (server)
- Not same CPU type for caller and callee

Many Local Call Assumptions Untrue for Remote Calls(2)

- Not always in the same administrative domain
- Latency for transfer of control and data can be large and, worse, unpredictable
- Partial failures
- Membership of the system (the computers in its collection) can change
- Unreliable or insecure communication
- Middleware handles this all, and more

Outline

- Background
- Middleware Basics
- CORBA Remote Object Middleware
- Quality Objects: DARPA Middleware on Steroids
- Discussion

Middleware Perspective

- "Middleware is like underwear: it is absolutely essential, but it should never be seen in public." -- unknown witticist
- Background info (only first page crucial): http://www.eecs.wsu.edu/~bakken/middleware.pdf

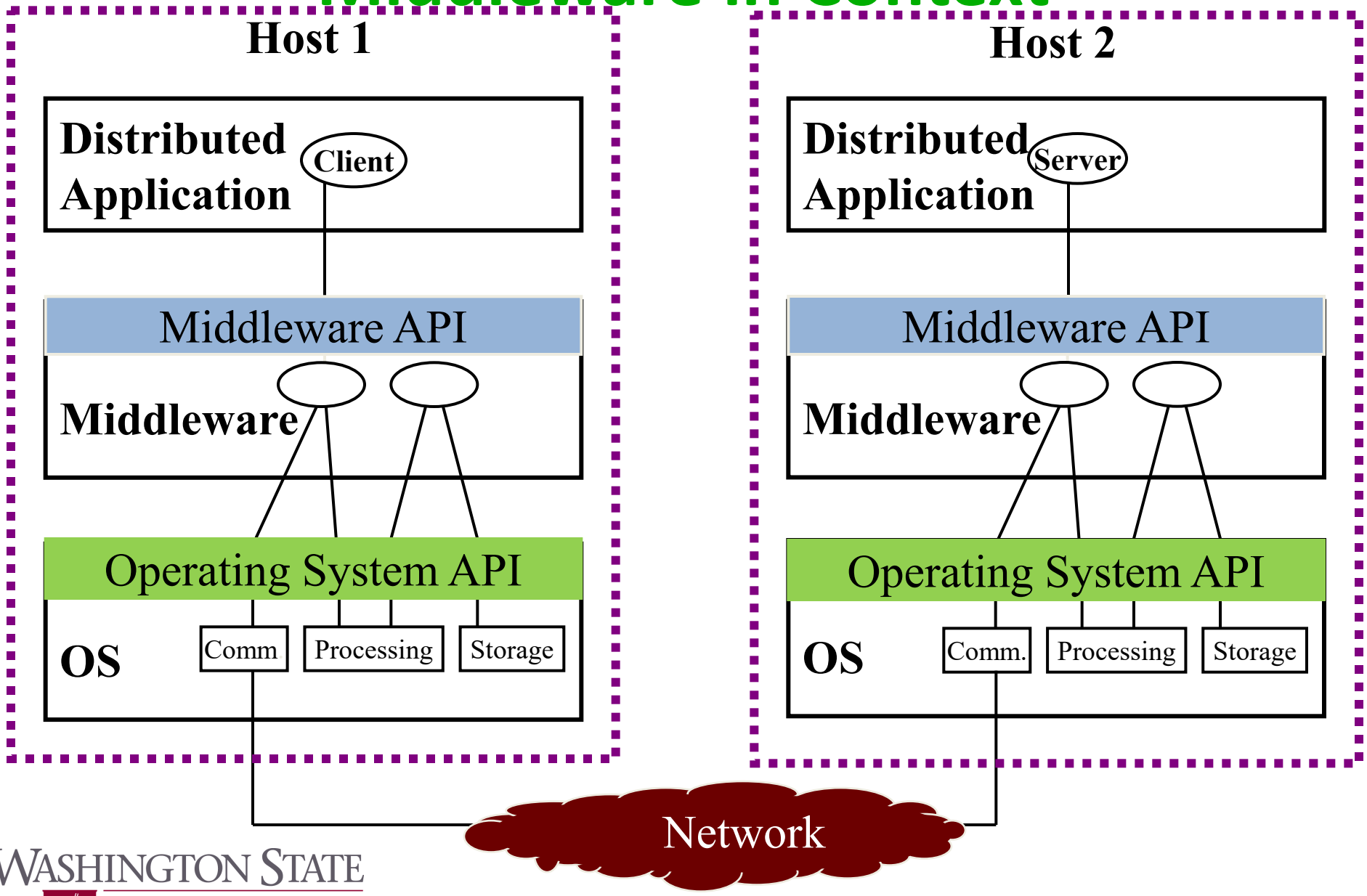
Context: (Most) Technology Marches On

- 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 proven technology for programmability of distributed computing systems

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**"
- 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
  - First started at BBN in 1979

Middleware in Context



WASHINGTON STATE UNIVERSITY

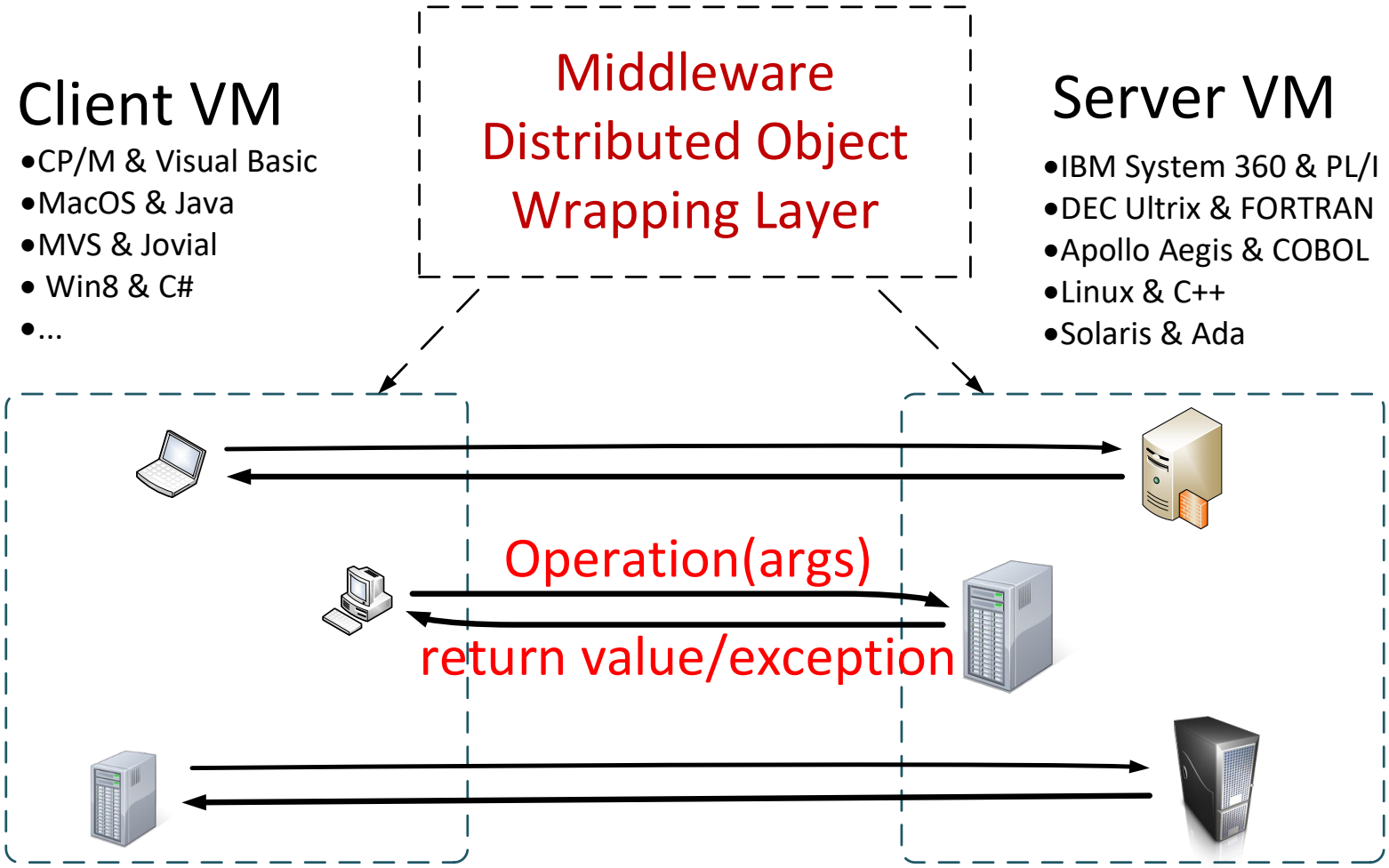
Middleware Benefit: Masking Heterogeneity

- 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 *de fiat*)
  - Almost all MW masks heterogeneity in programming language
    - Notable exception: Java RMI
  - Some MW masks heterogeneity in vendor implementations
    - Object Management Group (omg.org) best here: CORBA (object-oriented), DDS (publish-subscribe)
    - NOT OpenFMB ☹

Middleware Benefit: Transparency

- Middleware can provide useful "transparencies":
  (really opacity/opaqueness)
  - Access Transparency
  - Location transparency
  - Concurrency transparency
  - Replication transparency
  - Failure transparency
  - Mobility transparency
- Masking heterogeneity and providing transparency makes programming distributed systems much less difficult!

Middleware Wrapping Legacy Servers

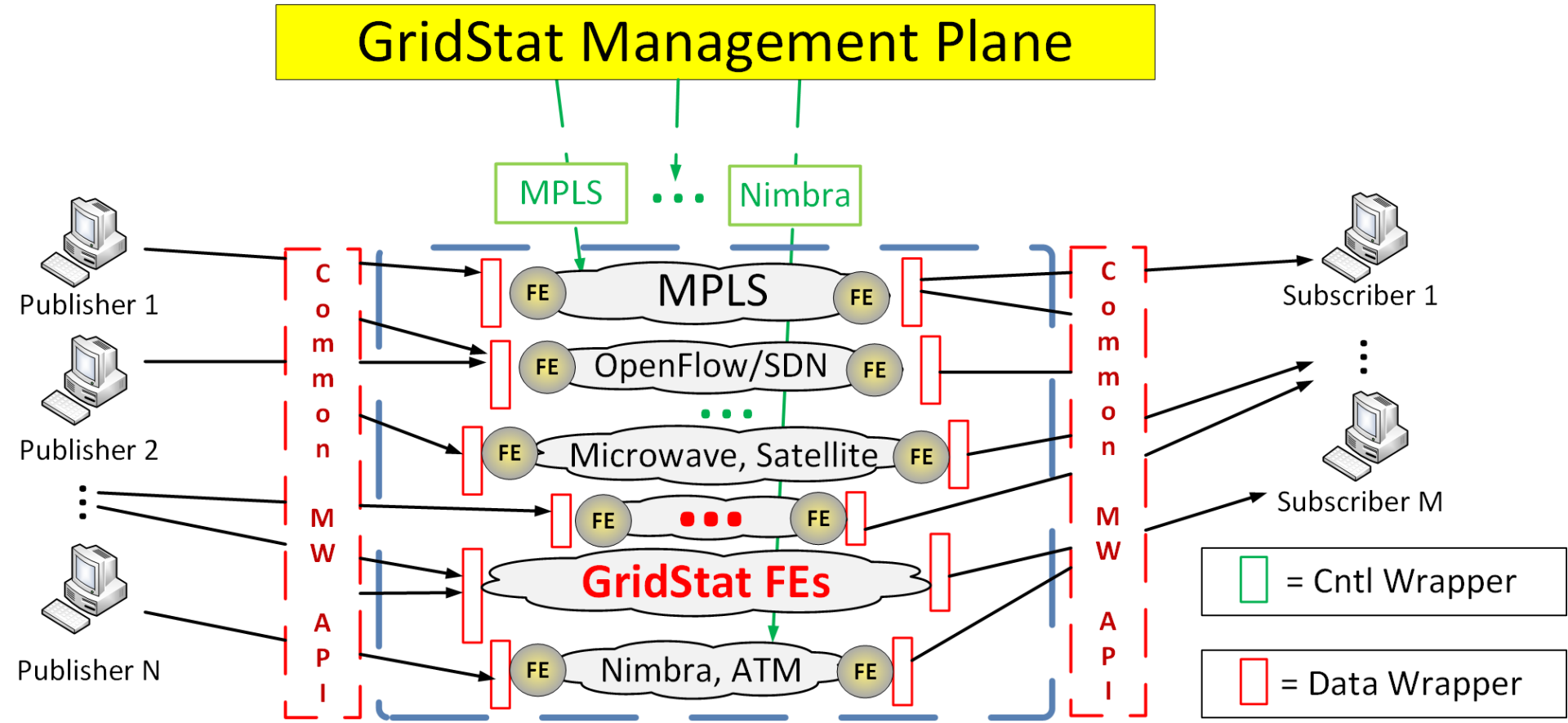


Overlay Middleware Integrating Legacy (Sub)Systems

Flow start (publisher) could also be RTU, substation router, OpenPDC, etc. i.e. not just a single sensor.

GS subscriber could be RTU, substation router, OpenPDC, …

GridStat is
- Real-Time
- Overlay (sometimes)
- Middleware



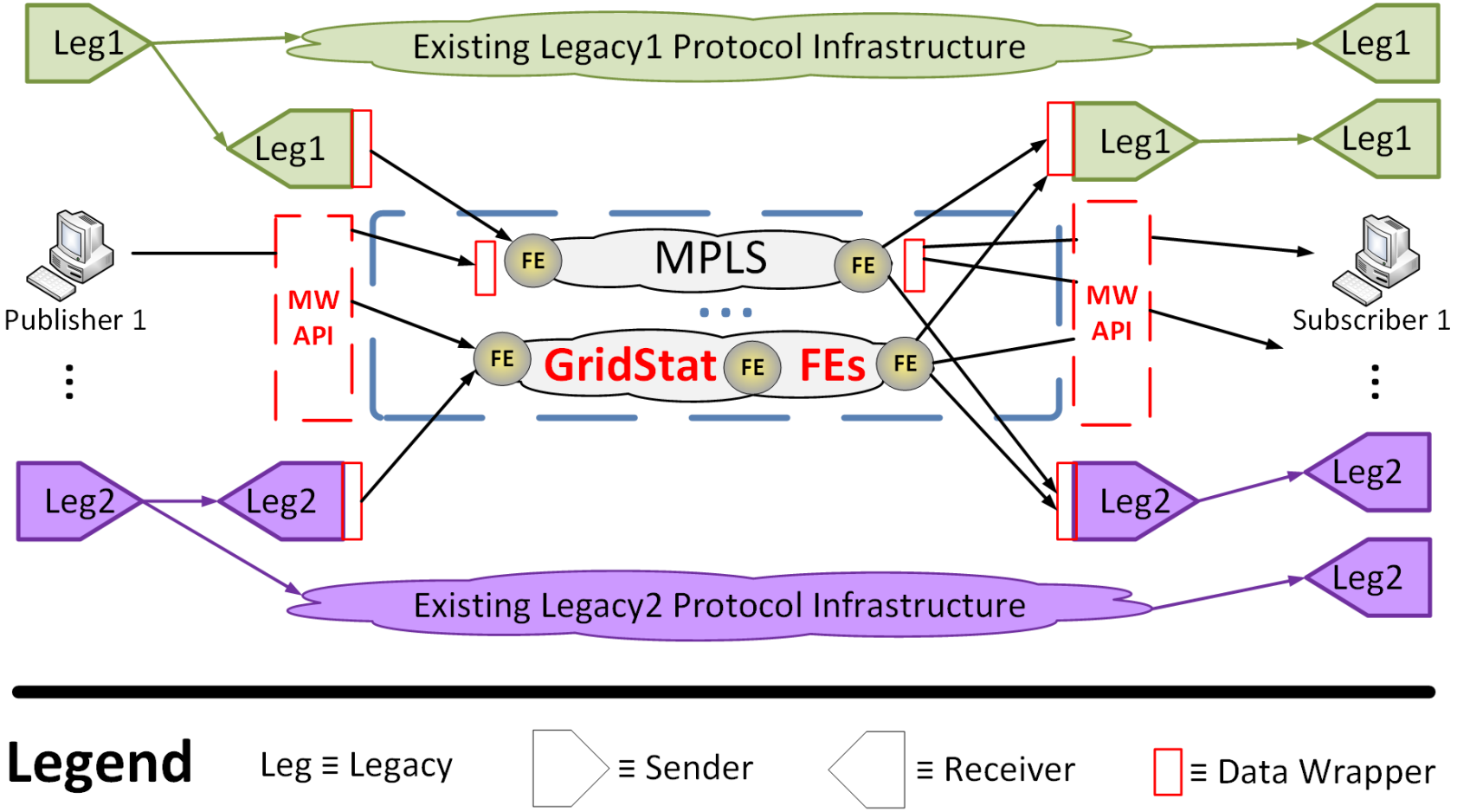
"…" could be BPL/PLC, 4G teleco, 801.1TSN, even best-effort internet, etc.

WASHINGTON STATE UNIVERSITY

© 2017 David E. Bakken

Getting There from Here

Issue: how to integrate existing legacy protocols and transition to newer ones and not break apps' data delivery?



WASHINGTON STATE UNIVERSITY

© 2018 David E. Bakken

Middleware vs. Sockets(1)

- Middleware is *much* easier to program with!
- Example interface in language-neutral CORBA (OMG) IDL
- (which is based on a C++ class declaration without local state variables and plus keywords(**in**, **out**, **inout**). Mappings exist to >10 other languages):

```
module HelloApp {
   interface Hello {
      bool MyFunction(in float a, in string b,
       in int c, in string d, in float e,
       out double ret);
       };
};
```

Middleware vs. Sockets(2)

The OMG IDL interface on the previous slide is mapped by the IDL compiler into a C++ **`class`** and a Java **`interface`**, and the proxy code is also generated. Here is how you make the local call to the C++ proxy

// above got helloImpl object reference (pointer) from a CORBA name server,
// which creates the proxy and returns the reference to it. In a one-process
// program, you would use malloc (C++) or new(Java, others) to create a
// Local-only object (not shared with any other process
// Assume all variables below are declared appropriately; omitted for brevity

```
boolean  success =
        helloImpl.MyFunction(3.3, "hello", 2345,
                             "bakken!", 67.34, &doubleBox);
```

Middleware vs. Sockets (3): Server Side

```
float a; char b[5]; int c;char d[7]; float e;
double rval; int success
//Ignore read errors. Hardcode field size, assume
//all systems are same CPU arch. and word size
read(socket, &a, sizeof(float));
read(socket, b, sizeof(char) * 5); // oops: align
read(socket, &c, sizeof(int));
read(socket, d, sizeof(char)*7);
read(socket, &e, sizeof(float));
// … continued on next slide …
```

Middleware vs. Sockets (4)

```
// … continued from previous slide …
… calculating return values etc goes here …
//send back return value
write(socket, &rval, sizeof(double));
//cant tell if it actually was recieved, or if socket
is broken
write(socket, &success, sizeof(boolean));
//again, no error checking
```

Middleware vs. Sockets (5)

- This socket code ignored all of the following:
  - Differences in CPU architecture (e.g., endianness)
  - Differences in representation of data types between languages
  - I/O errors
  - Type checking of data variables
- All of the above (and much more) are handled by middleware
- Middleware's programming building blocks (abstractions) mask heterogeneity
  - Makes programmer's life much easier!!
-

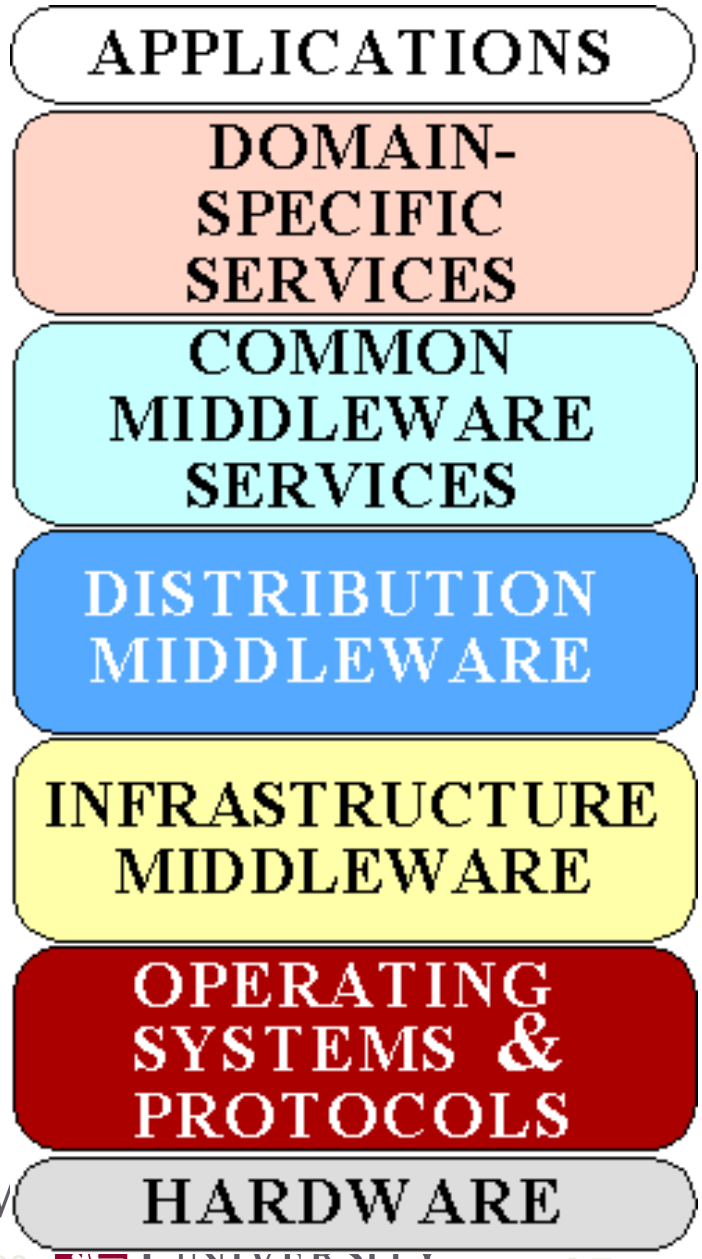
Middleware vs. Sockets (6)

- So how difficult is it to add a parameter or extra field in a data structure, between existing parameter/field declarations?
- Middleware: just recompile. Its all checked with strong data typing
- Sockets: before recompiling, first find EVERYWHERE that communicates with that parameter list or data structure
  - Change the code to add the parameter/field
  - Recompile

Multi-Layered Middleware



One Middleware Layering Taxonomy



- 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
- Common MW Services
  - Adds high-level, domain-independent reusable services for events, fault tolerance, security,
  - Examples: CORBAServices, Eternal, QuO
- Distribution MW
  - Provides rich distributed object model that supports much heterogeneity and transparency
  - Examples: CORBA, .NET., Java RMI
- Infrastructure MW
  - Encapsulates core OS Comm. and concurrency services (sometimes enhances them too)
  - Examples: JVM (and other VMs), ACE, group comm.

(Figure courtesy of D. Schmidt)

taxonomy courtesy of R.

Outline

- Background
- Middleware Basics
- CORBA Remote Object Middleware
- Quality Objects: DARPA Middleware on Steroids
- Discussion

CORBA Features

- "Transparencies" (really Opaqueness)
  - Programming language
  - CORBA vendor
  - Operating Systems
  - Location
  - Network HW/SW
  - Access
- Dynamic Binding
- Dynamic Typing
- Object Orientation
  - Encapsulation
  - Polymorphism
  - Inheritance
- Instantiation
- Extended Services
  - Naming/trader
  - Events/notification
  - Transactions
  - Security, domains, ftol
  - ……

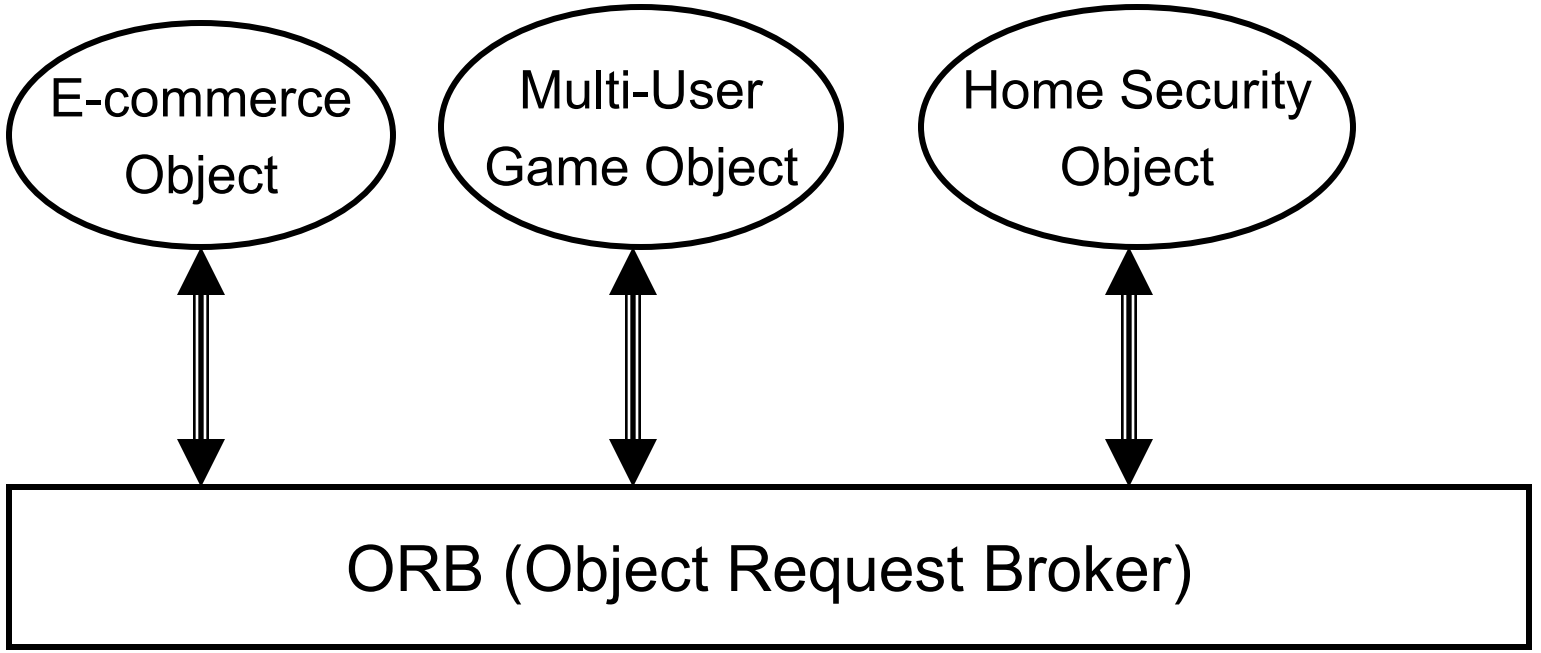
In an open specification with multivendor support

See Object Management Group (OMG) site, www.omg.org

Major CORBA Design Principles

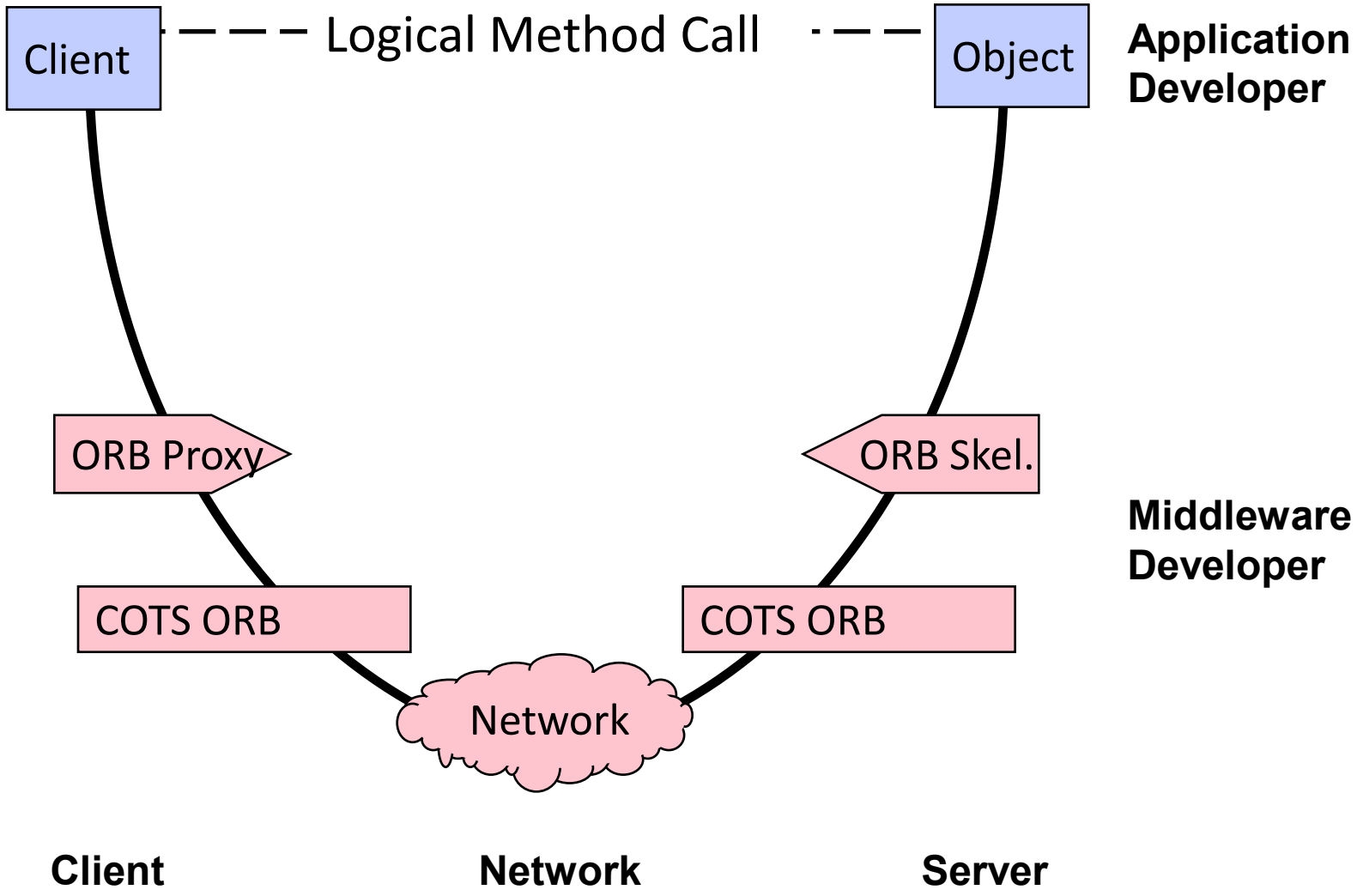
- Separation of interface and implementation
  - Clients depend on interfaces, not implementations
- Location transparency
  - Service use is orthogonal to service location
- Access transparency
  - Invoke CORBA objects just like local ones
- Typed interfaces
  - Object references are typed by interfaces
- Support of multiple inheritance of interfaces
  - Inheritance extends, evolves, and specialized behavior
  - Note: not implementation of multiple implementations!
- Support of multiple interaction styles
  - Client/server
    - Some support for mobile code, too, with Objects by Value (OBV)
  - Peer processes
  - Publish/Subscribe (aka “push”) but klunky event services, pre-DDS

CORBA: A "Software Bus"



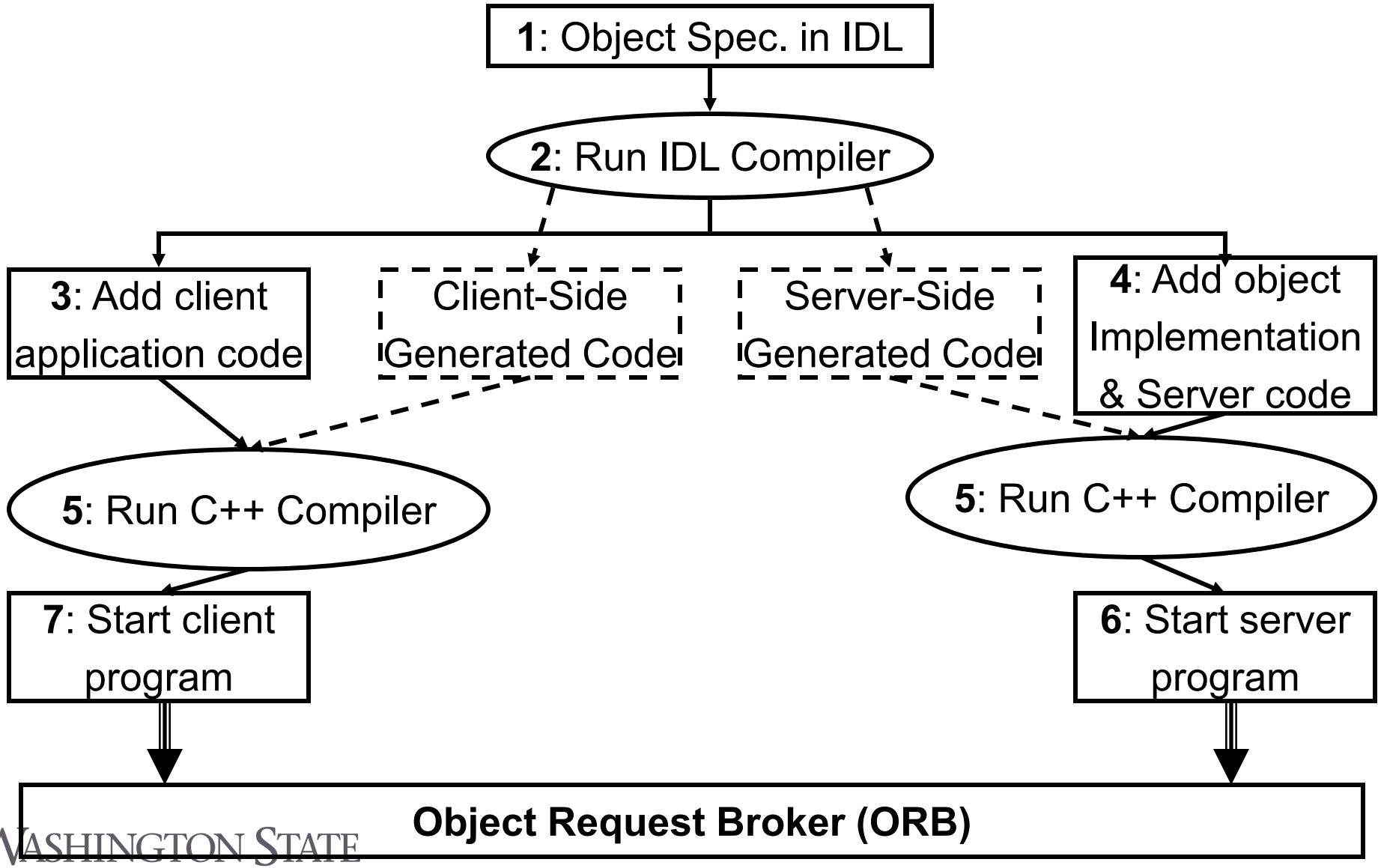
- Hardware bus: lets chips "plug and play" in a standard way
- Software bus: same idea but for software objects

Simplified Distributed Object (CORBA) Runtime Components



WASHINGTON STATE UNIVERSITY

CORBA C++ App. Development Steps



WASHINGTON STATE UNIVERSITY

CORBA & Exceptions

- Exception: out-of-band error signaling, keeping the error code separate from business logic (BBN X.500 example)

```
try {
      // normal "business logic" code", calls
      // procedures/functions
} catch {
     // catch exceptions you need to
}
```

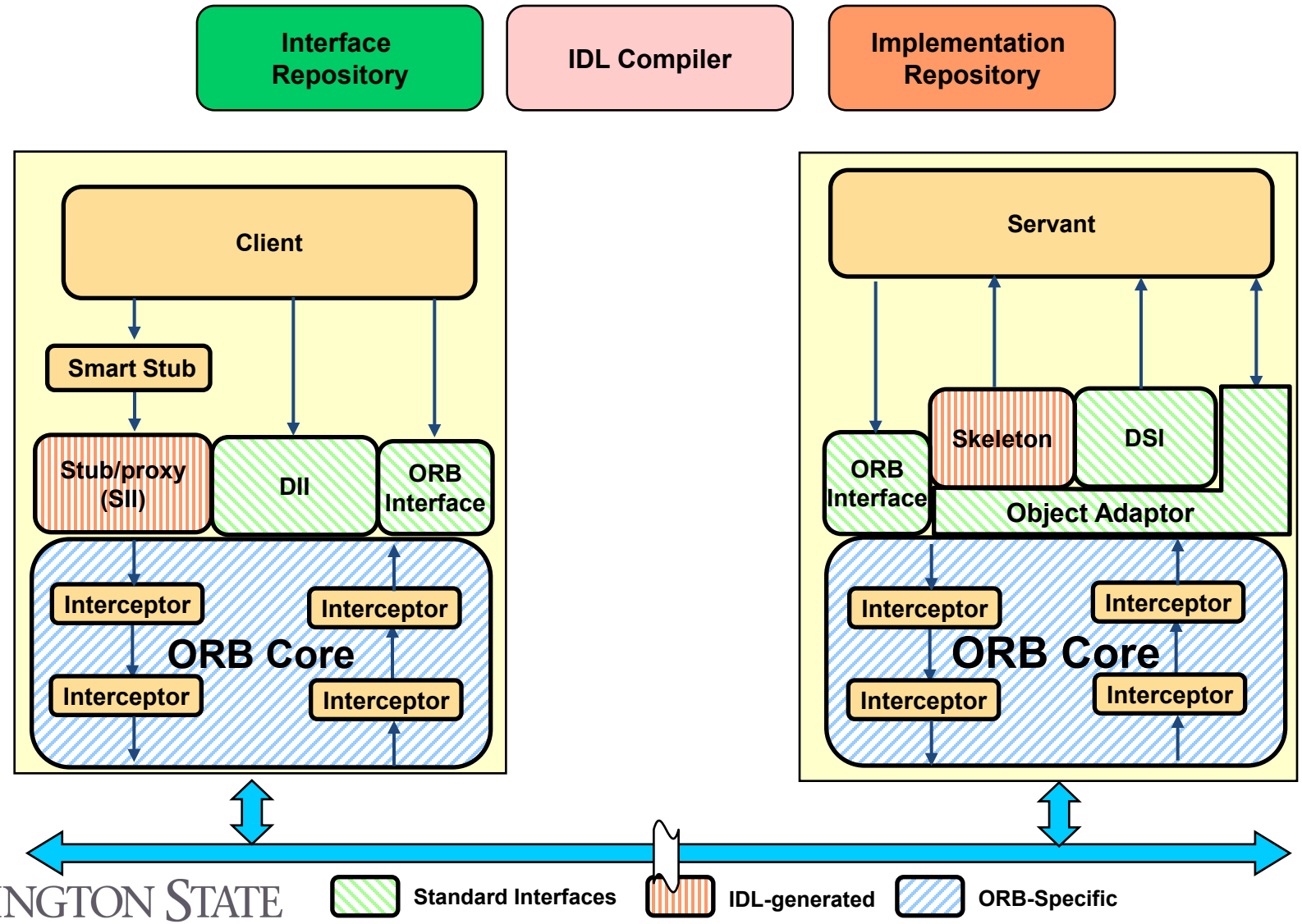
In CORBA, exceptions are communication errors and application faults that cross network boundaries. They are divided into two main categories: **System Exceptions** (21 predefined ORB errors like network failures or memory issues) and **User Exceptions** (custom, application-specific errors defined in your IDL).

WASHINGTON STATE UNIVERSITY

Messenger Throwing User-Level Exceptions

```
CORBA::Boolean
Messenger_impl::send_message(const char* user_name,
                             const char* subject, char*& message)
   throw(Messenger::ImNotHere,
         CORBA::SystemException)
{
 if (strcmp(user_name, "The_Professor") == 0) {
     // …
} else if (strcmp(user_name, "Joe_Freshman") == 0) {
     // don't want to talk to a freshman--send him the ImNotHere
    exception
     // and tell him to talk to someone else
     cerr << "Throw ImNotHere" << endl;
     cerr << endl;
     CORBA::String_var reason =
                CORBA::string_dup("Go talk to someone else");
     throw Messenger::ImNotHere(reason);
} else {
     // …
```

CORBA Modules & System Builders' Hooks

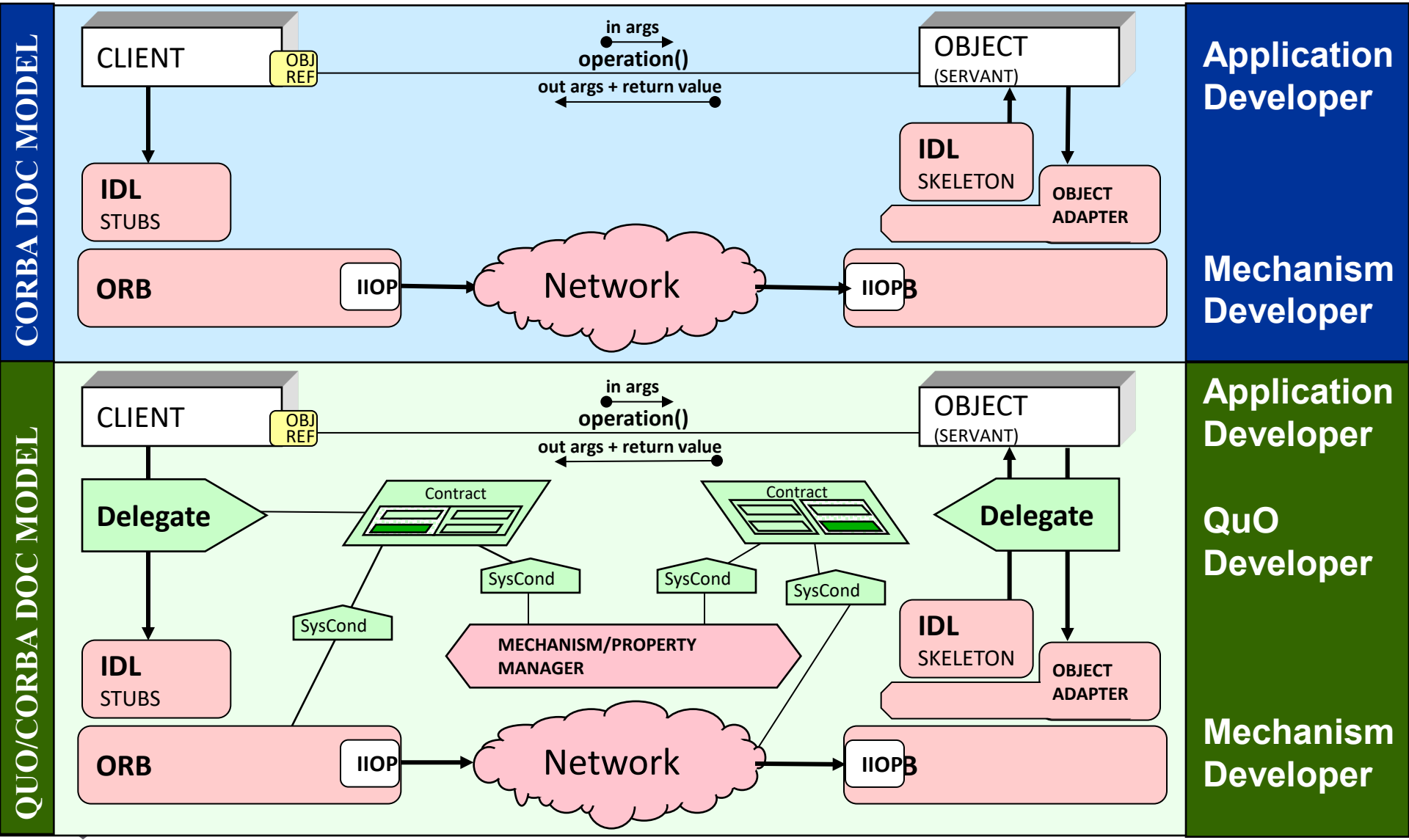


WASHINGTON STATE UNIVERSITY

Outline

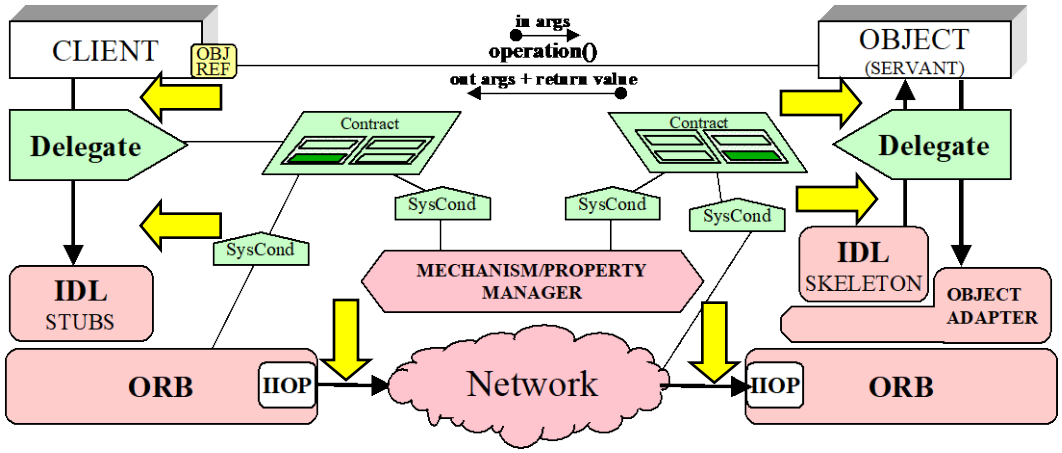
- Background
- Middleware Basics
- CORBA Remote Object Middleware
- Quality Objects: DARPA Middleware on Steroids
- Discussion

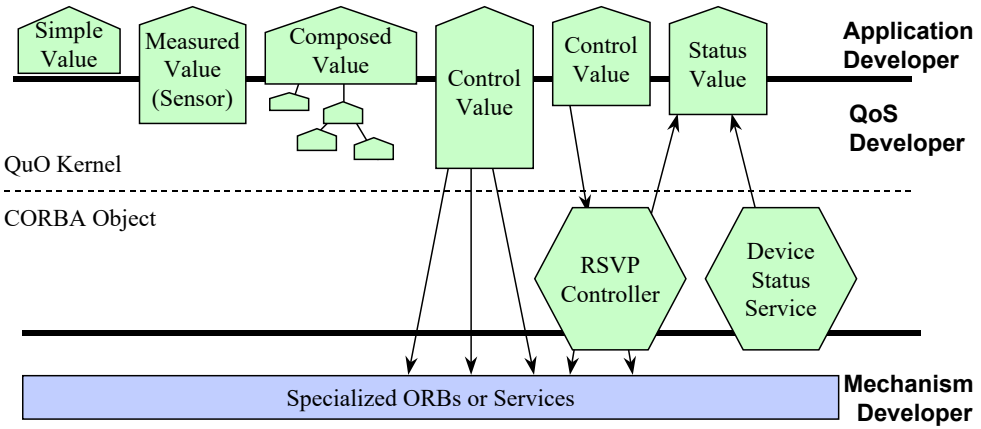
QuO Adds Specification, Measurement, and Adaptation to DOs



Measurement in QuO

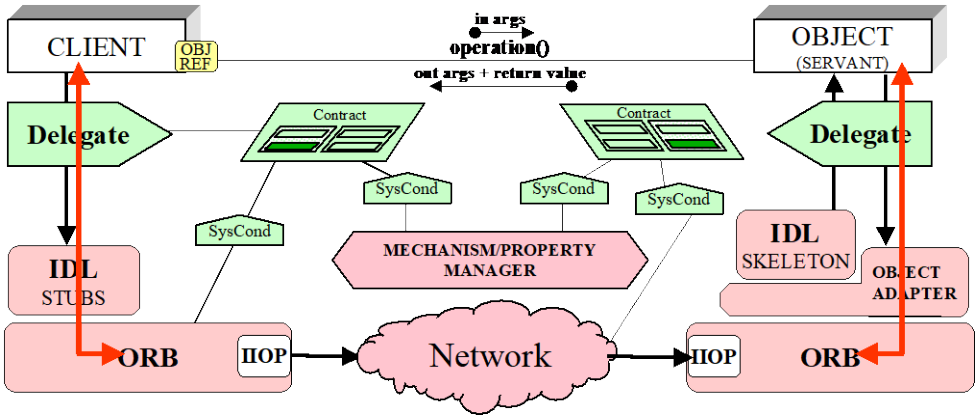
- *In-band* measurement handled by instrumentation
  - A structure is transparently passed along with the method call/return
  - Information can be inserted, read, and processed to record and evaluate method call statistics (e.g., the time spent in marshalling)
- *Out-of-band* measurement provided by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

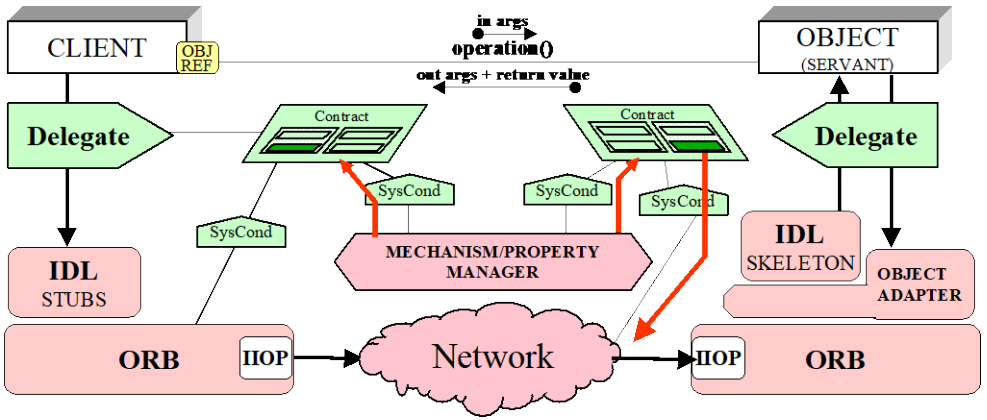




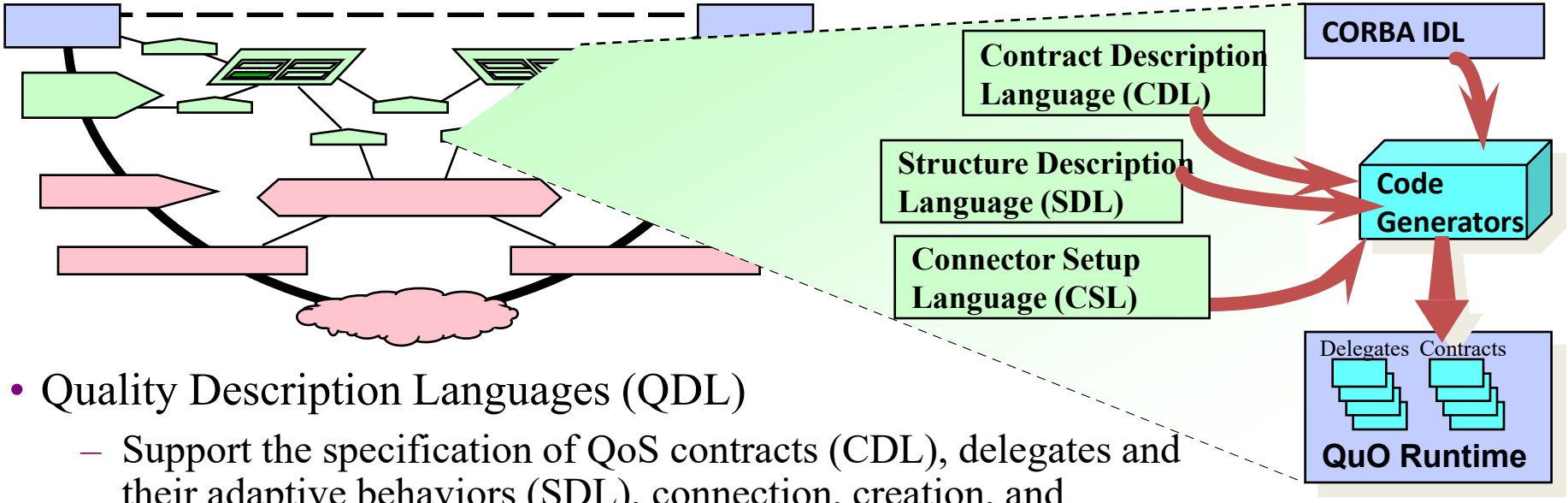
Adaptation and Control in QuO

- *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



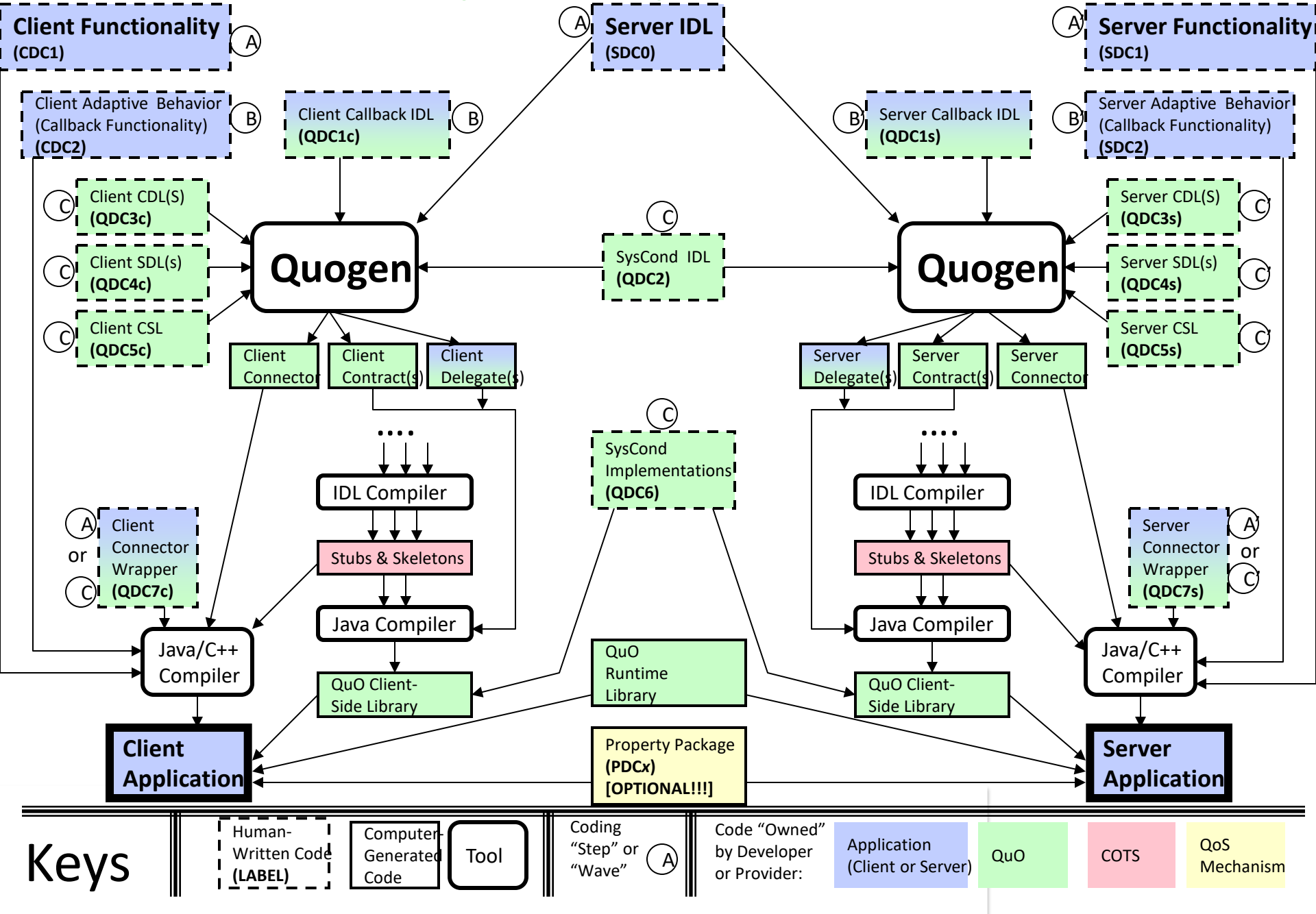


The QuO Toolkit Provides Tools for Building QuO applications (Note: QuO 3.0 different…)



- 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



Outline

- Background
- Middleware Basics
- CORBA Remote Object Middleware
- Quality Objects: DARPA Middleware on Steroids
- Discussion

PARKING LOT

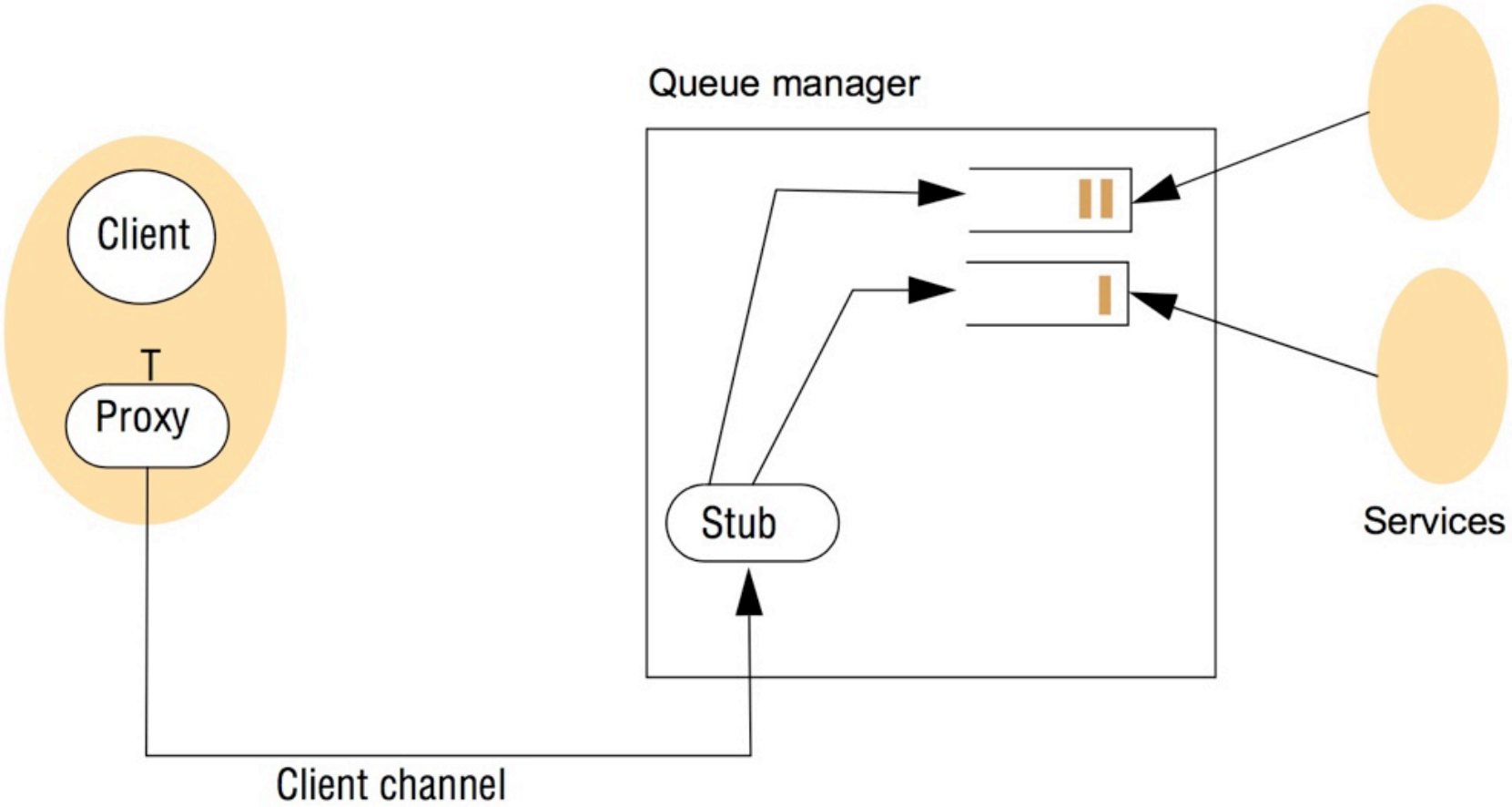
- David E. Bakken, Richard E. Schantz, and Richard D. Tucker. "Smart Grid Communications: QoS Stovepipes or QoS Interoperability", in *Proceedings of Grid-Interop 2009*, GridWise Architecture Council, Denver, Colorado, November 17-19, 2009.
- · **Best Paper Award for "Connectivity" track**.
- Argued that middleware is necessary for smart grids

Implementation issues [6.3.2]

- Key choice: centralized vs. distributed implementation
  - Tradeoffs?
- Case study: IBM Websphere MQ
  - **Queue managers** host and manage queues, enable apps to access via ***Message Queue Interface*** (MQI)
    - Connect or disconnect to/from a queue
    - Send/receive messages to/from a queue (via a RPC call)
    - Clients not on same host (usual case) vi a **client channel** (w/proxy+stub)

WASHINGTON STATE UNIVERSITY

A simple networked topology in WebSphere MQ



© Pearson Education 2005

IBM WebSphere (cont.)

- Queues usually linked into a federated structure
  - Resembles pub-sub, but choose right topology for app
  - Queues linked with **message channel**(MC)
  - **Message channel agent** (MCA) manages each end of MC
  - Queue managers have routing tables
  - Lots of tools to create different topologies, manage components, etc
- Hub-and-spoke topology (common)
  - Hub has lots of services (and resources to support)
  - Spoke queues are distant, place close(r) to clients
  - Clients interface with spoke queues