# Using Real-time CORBA Effectively
## *Patterns & Principles*


**Angelo Corsaro**
**Carlos O'Ryan**
**Douglas Schmidt**

{corsaro, coryan, schmidt}@uci.edu

**Elec. & Comp. Eng. Dept.**
**University of California, Irvine**

**Irfan Pyarali**

irfan@cs.wustl.edu

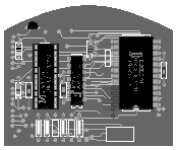
**Department of Computer Science, Washington University of St. Louis**


www.cs.wustl.edu/~schmidt/tutorials-corba.html


## Motivation for QoS-enabled Middleware

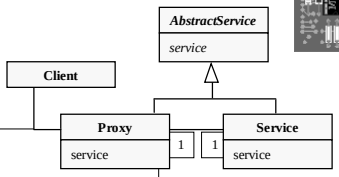
**Trends**

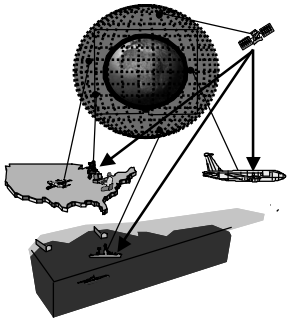
- Hardware keeps getting smaller, faster, & cheaper
- Software keeps getting larger, slower, & more expensive



**Historical Challenges**

- Building distributed systems is hard
- Building them on-time & under budget is even harder



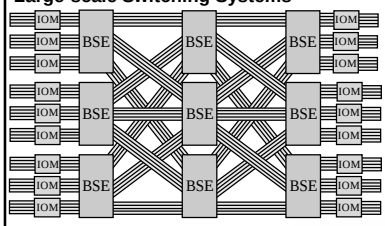
**New Challenges**

- Many mission-critical distributed applications require real-time QoS support
  - e.g., combat systems, online trading, telecom
- Building QoS-enabled applications manually is tedious, error-prone, & expensive
- Conventional middleware does not support real-time QoS requirements effectively

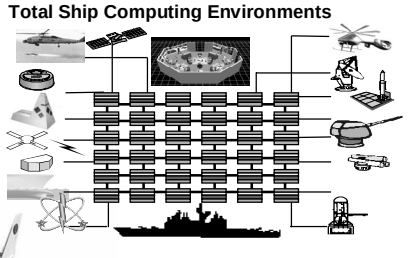
2

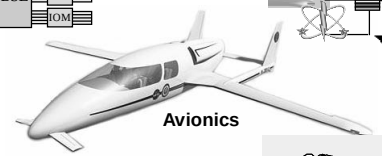
# Motivating Mission-Critical Applications

**Large-scale Switching Systems**




**Total Ship Computing Environments**

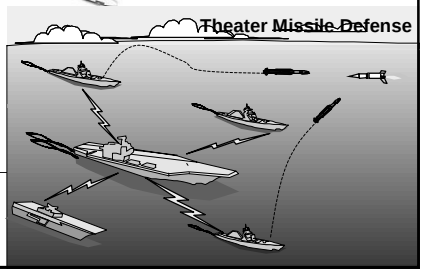




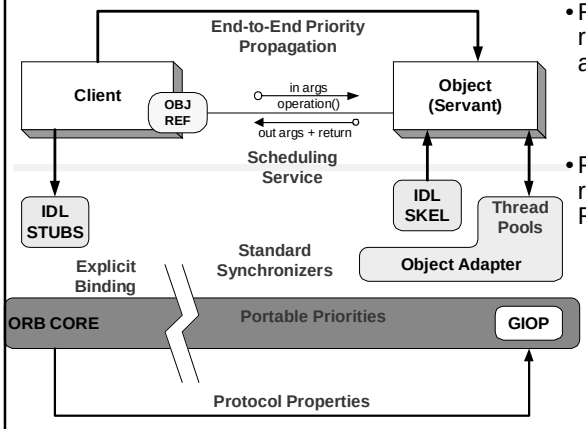
**Avionics**

**Industrial Process Control**

**Theater Missile Defense**



# Real-Time CORBA Overview




Real-time CORBA leverages the CORBA Messaging QoS Policy framework

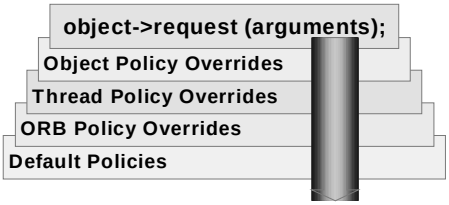
- RT CORBA adds QoS control to regular CORBA improve the application *predictability*, *e.g.,*
  - Bounding priority inversions &
  - Managing resources end-to-end
- Policies & mechanisms for resource configuration/control in RT-CORBA include:
  1. **Processor Resources**
     - Thread pools
     - Priority models
     - Portable priorities
  2. **Communication Resources**
     - Protocol policies
     - Explicit binding
  3. **Memory Resources**
     - Request buffering
- These capabilities address *some* (but by no means all) important real-time application development challenges

4

# Overview of the CORBA QoS Policy Framework

- CORBA defines a QoS framework that includes policy management for *request priority, queueing, message delivery quality, timeouts, etc*.
  - QoS is managed through interfaces derived from **`CORBA::Policy`**
    - Each QoS Policy can be queried with its **`PolicyType`**

- Client-side policies can be specified at 3 "overriding levels"
  1. ORB-level via **`PolicyManager`**
  2. Thread-level via **`PolicyCurrent`**
  3. Object-level via *overrides* in an object reference



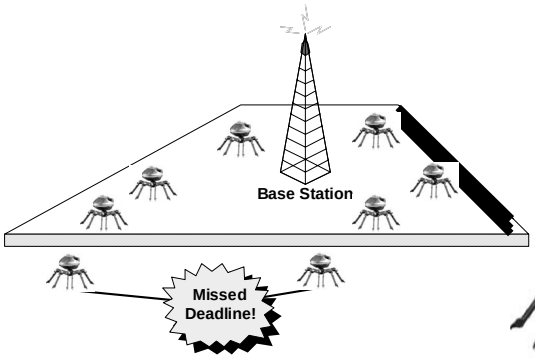

- Server-side policies can be specified at 3 overriding levels
  1. ORB-level through **`PolicyManager`**
  2. POA-level passed as arguments to **`POA::create_POA()`**
  3. Some policies can be set at the Object-level through the **`RTPOA`**
- Server-side policies can be stored in the tagged components of the IOR

| IIOP VERSION | HOST | PORT | OBJECT KEY | TAGGED COMPONENTS |
|---|---|---|---|---|

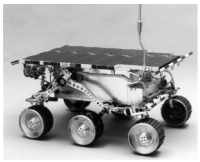
- Client-side policies are validated via **`Object::_validate_connection()`**

5

# An Example Distributed Application



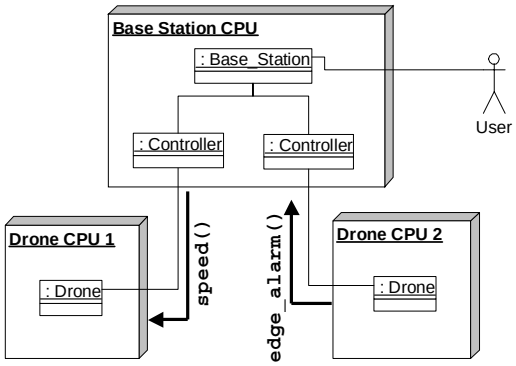

- Consider an application where cooperating *drones* explore a surface & report its properties periodically
  - *e.g.,* color, texture, etc.



- Drones aren't very "smart,"
  - *e.g.,* they can fall off the "edge" of the surface if not stopped
- Thus, a *controller* is used to coordinate their actions
  - *e.g.,* it can order them to a new position

6

# Designing the Application




- End-users talk to a **`Base_Station`** object
  - *e.g.,* they define high-level exploration goals for the drones
  - The **`Base_Station`** provides set-points for the controllers
- The **`Controller`** object controls the drones remotely using **`Drone`** objects
- **`Drone`** objects are proxies for the underlying drone vehicles
  - *e.g.,* they expose operations for controlling & monitoring individual drone behavior

- Each drone sends information obtained from its sensors back to the **`Base_Station`** via a **`Controller`** object

7

# Defining Application Interfaces with CORBA IDL

```
interface Drone {
  void turn (in float degrees);
  void speed (in short mph);
  void reset_odometer ();
  short odometer ();
  // …
};
```

```
interface Controller {
  void edge_alarm ();
  void battery_low ();
  //…
};
```

```
interface Base_Station {
  Controller new_controller
       (in string name)
    raises (Lack_Resources);
  void set_new_target
       (in float x, in float y,
        in float w, in float h);
  //……
};
exception Lack_Resources {};
```

- Each **`Drone`** talks to a **`Controller`**
  - *e.g.,* **`Drones`** send hi-priority **`edge_alarm()`** messages when they detect an edge
- The **`Controller`** should take corrective action if a **`Drone`** detects it's about to fall off an edge!
- The **`Base_Station`** interface is a **`Controller`** factory
  - **`Drones`** use this interface to create their **`Controllers`** during power up
  - End-users use this interface to set high-level mobility targets

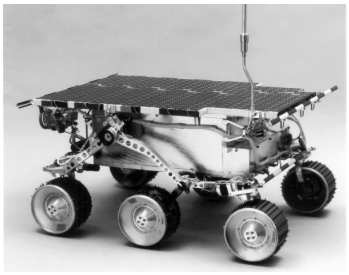
This API is a simplification of various semi-autonomous vehicle use-cases

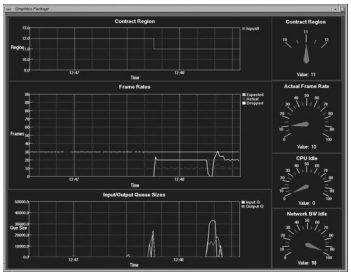
8

# QoS-related Application Design Challenges

- Our example application contains the following QoS-related design challenges

1. *Obtaining portable ORB end-system priorities*
2. *Preserving priorities end-to-end*
3. *Enforcing certain priorities at the server*
4. *Changing CORBA priorities*
5. *Supporting thread pools effectively*
6. *Buffering client requests*
7. *Synchronizing objects correctly*
8. *Configuring custom protocols*
9. *Controlling network & end-system resources to minimize priority inversion*
10. *Avoiding dynamic connections*
11. *Simplifying application scheduling*
12. *Controlling request timeouts*





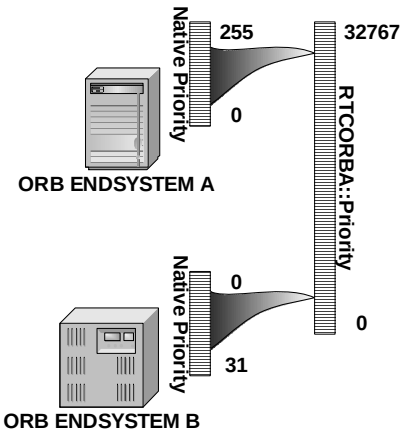

9

# Portable End-to-End Priorities

- **Problem:** How can we map global priorities onto heterogeneous native OS host thread priorities consistently end-to-end?
- **Solution:** Use Standard RT CORBA priority mapping interfaces

10

## Obtaining Portable ORB End-system Priorities




- OS-independent design supports heterogeneous real-time platforms
- CORBA priorities are “globally” unique values that range from 0 to 32767
- Users can map CORBA priorities onto native OS priorities in custom ways
- No silver bullet, but rather an ``enabling technique''
  - *i.e.,* can’t magically turn a general-purpose OS into a real-time OS!

11

## Setting Custom Priority Mapping

- **Problem:** How do we configure the **`PriorityMapping`** that the ORB should use?
- **Solution:** Use TAO’s **`PriorityMappingManager`**!

12

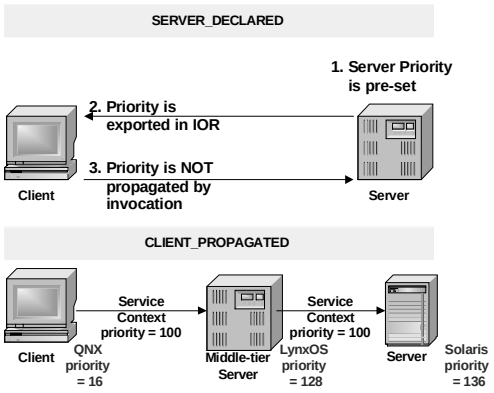
# Preserving Priorities End-to-End

- **Problem:** How can we ensure requests don't run at the wrong priority on the server?
  - *e.g.,* this can cause major problems if **`edge_alarm()`** operations are processed too late!!!
- **Solution:** Use RT CORBA priority model policies

13

# Preserving Priorities End-to-End

- RT CORBA priority model policies
  - **`SERVER_DECLARED`**
    - Server handles requests at the priority declared when object was created
  - **`CLIENT_PROPAGATED`**
    - Request is executed at the priority requested by client
    - Priority is encoded as part of client request




14

## Changing CORBA Priorities

- **Problem:** How can RT-CORBA client application change the priority of operations?
- **Solution:** Use the **`RTCurrent`** to change the priority of the current client thread explicitly

15

## Design Interlude: The RT-ORB Interface

- **Problem:** How can an ORB be extended to support RT-CORBA *without* changing the **`CORBA::ORB`** interface?
- **Solution:**Use *Extension Interface* pattern from POSA2 book <www.posa.uci.edu>
  - Use **`resolve_initial_references()`** interface to obtain the extension
  - Thus, non real-time ORBs and applications are not affected by RT CORBA enhancements!

16

## Getting the RTORB Extension Interface

```
CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);
CORBA::Object_var obj =
   orb->resolve_initial_references ("RTORB");
RTCORBA::RTORB_var rtorb =
   RTCORBA::RTORB::_narrow (obj);
// Assuming that <_narrow> succeeds we can henceforth use RT
// CORBA features
```

- The **`resolve_initial_references()`** method takes a string representing the desired *extension interface*
- It either returns an object reference to the requested extension interface or it returns nil

17

## Enforcing CORBA Priorities

- **Problem:** How to ensure that certain operations always run at a fixed priority?
  - *e.g.,* the **`Base_Station`** methods are not time-critical, so they should always run at lower priority than the **`Controller`** methods
- **Solution:** Use the RT CORBA **`SERVER_DECLARED`** priority model

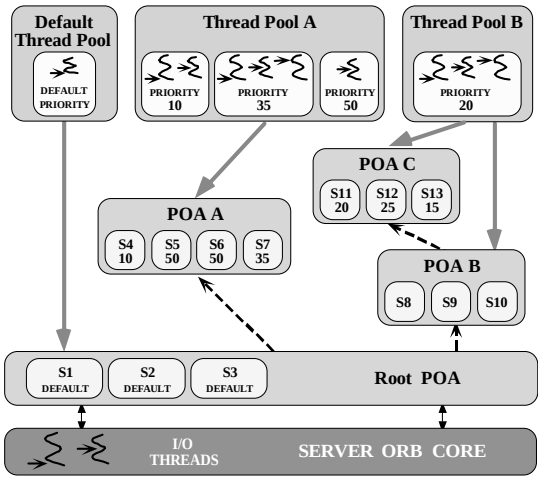
18

# Thread Pooling

- **Problem:** How can we pre-allocate threading resources on the server *portably* & *efficiently?*
  - *e.g.*, the **`Base_Station`** must have sufficient threads for all its priority levels
- **Solution:** Use RT CORBA thread pools

19

# RT-CORBA Thread Pools

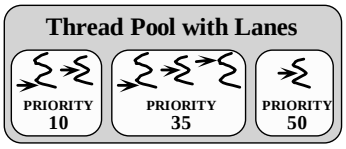



- Pre-allocation of threads
- Partitioning of threads
- Bounding of thread usage
- Buffering of additional requests

20

## Partitioning Thread Pools

- **Problem:** How can we prevent exhaustion of threads by low priority requests?
  - *e.g.,* many requests to the **`Base_Station`** methods use up all the threads in the thread pool so that no threads for high-priority **`Controller`** methods are available
- **Solution:** Partition thread pool into subsets, which are called *lanes*, where each lane has a different priority



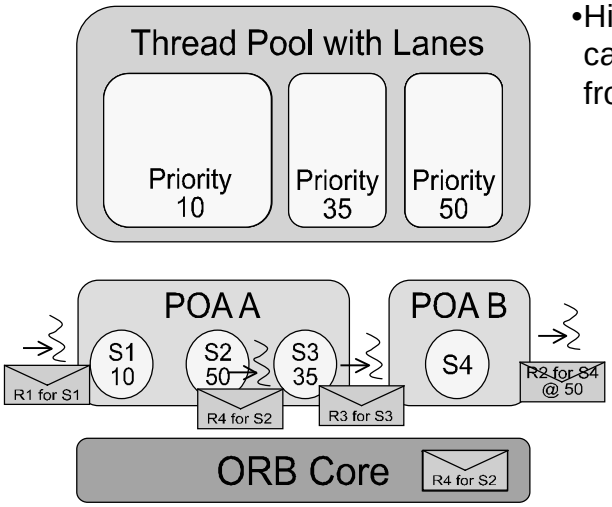

21

## When you run out of Threads...

- **Problem:** How can we prevent bursts or long-running requests from exhausting maximum number of static & dynamic threads in the lane?
- **Solution:** Use the Real-time CORBA thread pool *lane borrowing* feature

22

# Thread Borrowing




- Higher priority lanes can borrow threads from lower priority lanes

**Restoring threads**

- Priority is raised when thread is borrowed
- When there are no more requests, borrowed thread is returned & priority is restored

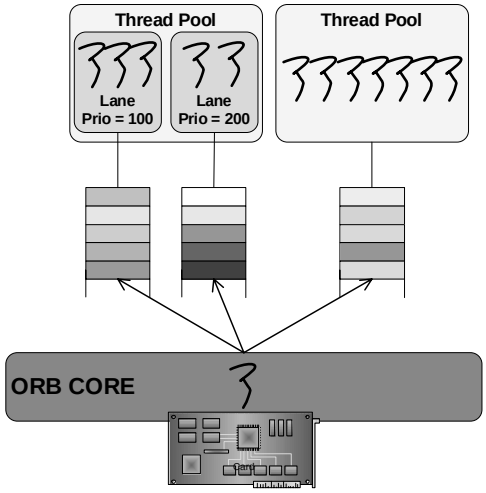
23

# Managing Bursty Requests

- **Problem:** How can we support real-time applications that need more buffering than is provided by the OS I/O subsystem
  - *e.g.,* to handle "burstly" client traffic
- **Solution:** Buffer client requests in ORB

24

# Buffering Client Requests




- RT CORBA thread pool buffer capacities can be configured according to:
  1. Maximum number of bytes and/or
  2. Maximum number of requests

25

# Thread Pools Implementation Strategies

- There are two general strategies to implement RT CORBA thread pools:
  1. Use the *Half-Sync/Half-Async* pattern to have I/O thread(s) buffer client requests in a queue & then have worker threads in the pool process the requests
  2. Use the *Leader/Followers* pattern to demultiplex I/O events into threads in the pool *without* requiring additional I/O threads

- Each strategy is appropriate for certain application domains
  - *e.g.,* certain hard-real time applications cannot incur the non-determinism & priority inversion of additional request queues
- To evaluate each approach we must understand their consequences
  - Their pattern descriptions capture this information
  - Good metrics to compare RT-CORBA implementations

26

# Evaluating Thread Pools Implementations

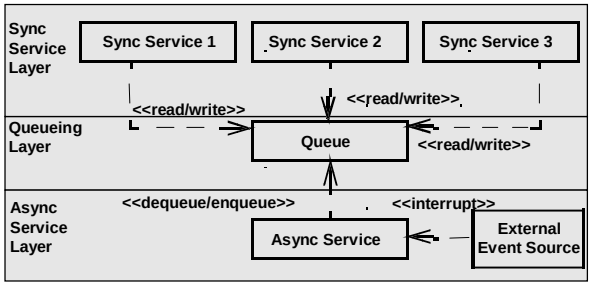
- RT-CORBA spec under-specifies many quality of implementation issues
  - *e.g.:* Thread pools, memory, & connection management
  - Maximizes freedom of RT-CORBA developers
  - Requires application developers to understand ORB implementation
  - Effects schedulability, scalability, & predictability of their application
- Examine patterns underlying common thread pool implementation strategies
- Evaluate each thread pool strategy in terms of the following capabilities

| **Capability** | Description |
|---|---|
| **Feature support** | Request buffering & thread borrowing |
| **Scalability** | Endpoints & event demultiplexers required |
| **Efficiency** | Data movement, context switches, memory allocations, & synchronizations required |
| **Optimizations** | Stack & thread specific storage memory allocations |
| **Priority inversion** | Bounded & unbounded priority inversion incurred in each implementation |

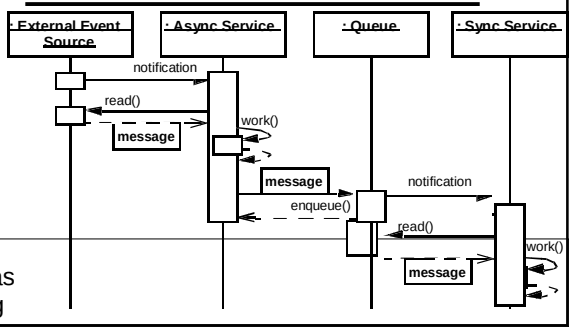
27

# The Half-Sync/Half-Async Pattern

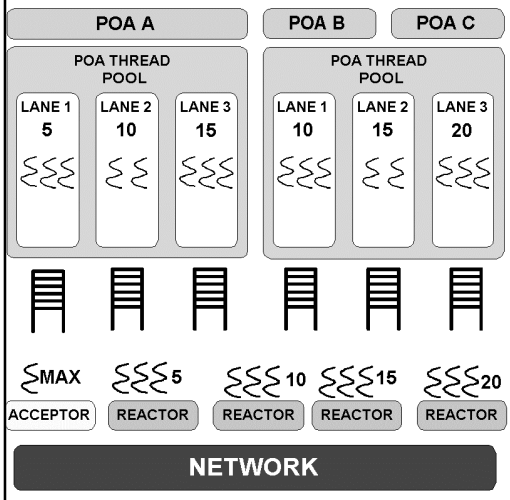
**Intent**
The *Half-Sync/Half-Async* architectural pattern decouples async & sync service processing in concurrent systems, to simplify programming without unduly reducing performance




- This pattern defines two service processing layers—one async and one sync—along with a queueing layer that allows services to exchange messages between the two layers
- The pattern allows sync services, such as servant processing, to run concurrently, relative both to each other and to async services, such as I/O handling & event demultiplexing



# *Queue-per-Lane* Thread Pool Design




**Design Overview**
- Single acceptor endpoint
- One reactor for each priority level
- Each lane has a queue
- I/O & application-level request processing are in different threads

**Pros**
- Better feature support, *e.g.,*
  - Request buffering
  - Thread borrowing
- Better scalability, *e.g.,*
  - Single acceptor
  - Fewer reactors
  - Smaller IORs
- Easier piece-by-piece integration into the ORB

**Cons**
- Less efficient because of queueing
- Predictability reduced without _bind_priority_band() implicit operation

29

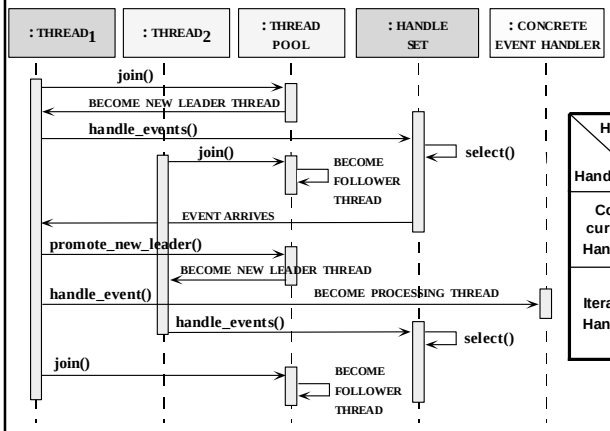
# Evaluation of Half-Sync/Half-Async Thread Pools

| Criteria | Evaluation |
|---|---|
| Feature Support | ***Good:*** supports request buffering & thread borrowing |
| Scalibility | ***Good:*** I/O layer resources shared |
| Efficiency | ***Poor:*** high overhead for data movement, context switches, memory allocations, & synchronizations |
| Optimizations | ***Poor:*** stack & TSS memory not supported |
| Priority Inversion | ***Poor:*** some unbounded, many bounded |

30

# The Leader/Followers Pattern

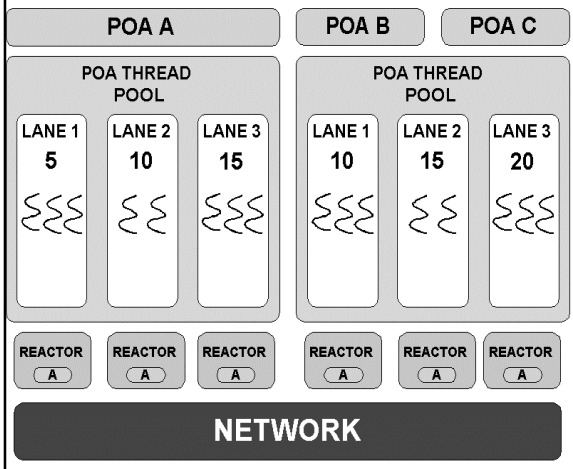
**Intent:** The Leader/Followers architectural pattern provides an efficient concurrency model where multiple threads take turns sharing event sources to detect, demux, dispatch, & process service requests that occur on the event sources




| Handles \ Handle Sets | Concurrent Handle Sets | Iterative Handle Sets |
|---|---|---|
| Concurrent Handles | UDP Sockets + `WaitForMultipleObjects()` | UDP Sockets + `select()`/`poll()` |
| Iterative Handles | TCP Sockets + `WaitForMultipleObjects()` | TCP Sockets + `select()`/`poll()` |

31

# *Reactor-per-Lane* Thread Pool Design




**Design Overview**
- Each lane has its own set of resources
  - *i.e.,* reactor, acceptor endpoint, etc.
- I/O & application-level request processing are done in the same thread

**Pros**
- Better performance
  - No extra context switches
  - Stack & TSS optimizations
- No priority inversions during connection establishment
- Control over ***all*** threads with standard thread pool API

**Cons**
- Harder ORB implementation
- Many endpoints = longer IORs

32

## Evaluation of Leader/Followers Thread-Pools

| Criteria | Evaluation |
|---|---|
| Feature Support | ***Poor:*** not easy to support request buffering or thread borrowing |
| Scalibility | ***Poor:*** I/O layer resources not shared |
| Efficiency | ***Good:*** little or no overhead for data movement, memory allocations, or synchronizations |
| Optimizations | ***Good:*** stack & TSS memory supported |
| Priority Inversion | ***Good:*** little or no priority inversion |

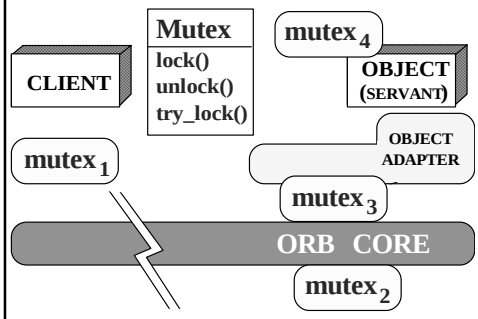
33

## Consistent Synchronizers

- **Problem:** An ORB & application may need to use the same *type* of mutex to avoid priority inversions
  - *e.g.,* using priority ceiling or priority inheritance protocols
- **Solution:** Use the **`RTCORBA::Mutex`** synchronizer

34

# Synchronizing Objects Consistently




- The **`RTCORBA::Mutex`** interface ensure consistent mutex semantics, across ORB & application domains

```
RTCORBA::Mutex_var mutex = rtorb->create_mutex ();
...
mutex->lock ();
// Critical section here…
mutex->unlock ();
...
rtorb->destroy_mutex (mutex);
```

`create_mutex()` is a factory method

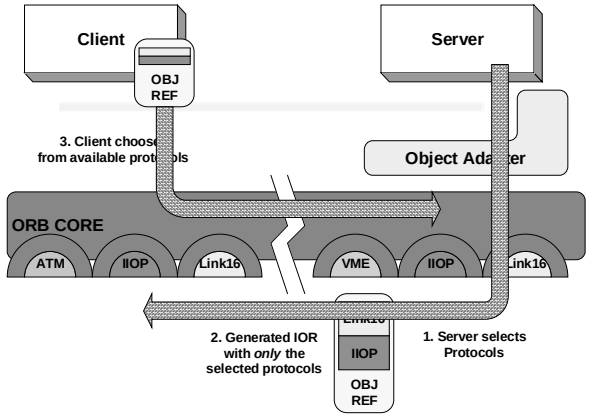
35

# Custom Protocol Configuration

- **Problem:** Selecting communication protocol(s) is crucial to obtaining QoS
  - TCP/IP is inadequate to provide end-to-end *real-time* response
  - Thus, communication between **`Base_Station`**, **`Controllers`**, & **`Drones`** must use a different protocol
  - Moreover, some messages between **`Drone`** & **`Controller`** cannot be delayed
- **Solution:** Use RT-CORBA Protocol Policies to select and/or configure communication protocols

36

# Configuring Custom Protocols




- Both server-side & client-side policies are supported
- Some policies control protocol selection, others configuration
- Order of protocols indicates protocol preference

**Ironically, RT-CORBA specifies only protocol properties for TCP!**

37

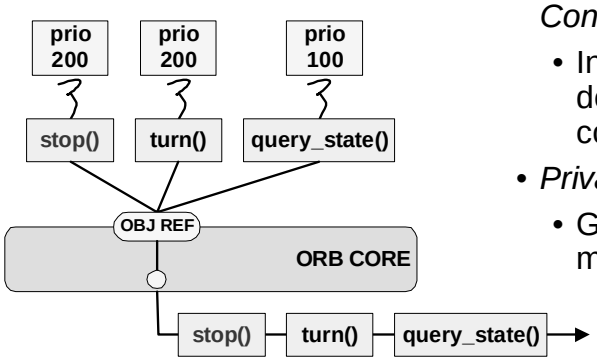
# Network Resource Issues

- **Problem:** How can we achieve the following?
  - Control jitter due to connection setup
  - Minimize thread-level priority inversions
  - Avoid request-level ("head-of-line") priority inversions
- **Solution:** Use RT CORBA *explicit binding* mechanisms

38

# Controlling Network Resources

Note the priority inversion below since the **`stop()`**, **`turn()`**, and **`query_state()`** requests all share the same connection




- *Connection pre-allocation*
  - Eliminates a common source of operation jitter
- *Priority Banded Connection Policy*
  - Invocation priority determines which connection is used
- *Private Connection Policy*
  - Guarantees non-multiplexed connections

39

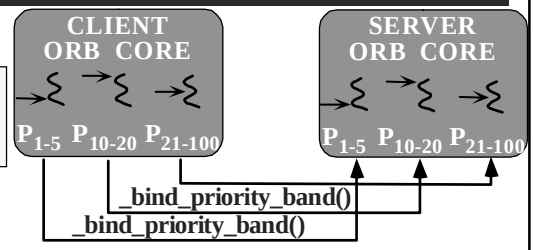
# Connection Establishment

- **Problem:** How can we prevent connection establishment between the base station and the drones from resulting in unacceptable jitter?
  - Jitter is detrimental to time-critical applications
- **Solution:** Pre-allocate one or more connections using the **`Object::_validate_connection()`** operation

40

# Pre-allocating Network Connections

The `_validate_connection()` operation must be invoked before making any other operation calls




```
// Drone reference
Drone_var drone = ...;

// Pre-establish connections
// using current policies
CORBA::PolicyList_var invalid_policies;

// The following operation causes a _bind_priority_band()
// "implicit" request to be sent to the server
CORBA::Boolean success =
  drone->_validate_connection (invalid_policies);
```

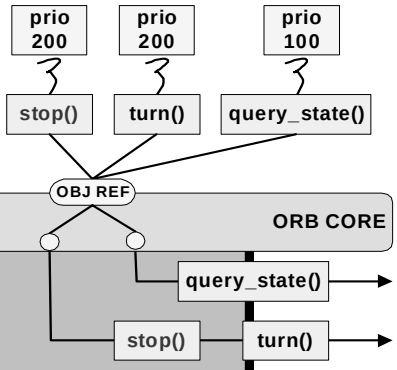
# Connection Banding

- **Problem:** How can we minimize priority inversions, so that high-priority operations are not queued behind low-priority operations?
- **Solution:** Use different connections for different priority ranges via the RT CORBA **`PriorityBandedConnectionPolicy`**

42

# Priority Banded Connection Policy

Note how the **`stop()`** and **`turn()`** requests no longer share the same connection as **`query_state()`** requests




```
// Create the priority bands
RTCORBA::PriorityBands bands (2);
bands.length (2);
// We can have bands with a range
// of priorities...
bands[0].low  = 0;
bands[0].high = 150;
// ... or just a "range" of 1!
bands[1].low  = 200;
bands[1].high = 200;

CORBA::Policy_var policy = rtorb->
  create_priority_banded_connection_policy (bands);
```

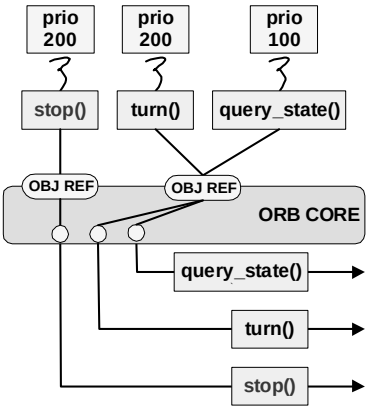
43

# Controlling Connection Multiplexing

- **Problem:** How can we minimize priority inversions by ensuring applications don't share a connection between multiple objects running at different priorities?
  - *e.g.*, sending a **`stop()`** request should use exclusive, pre-allocated resources
- **Solution:** Use the RT CORBA **`PrivateConnectionPolicy`** to guarantee non-multiplexed connections

44

# Private Connection Policy




Note how the **`stop()`** and **`turn()`** requests no longer share the same connection from client to server

```
policies[0] =
   rtorb->create_private_connection_policy ();
CORBA::Object_var object =
   drone->_set_policy_overrides (policies,
                                 CORBA::ADD_OVERRIDES);
```

45

# Scheduling Activities

- **Problem:** How can RT-CORBA give developers control over system resources while avoiding the following two deficiencies:
  - It can be tedious to configure all the CORBA client/server policies
  - Application developers must select the right priority values
- **Solution:** Apply the RT-CORBA Scheduling Service to simplify application scheduling
  - Developers just declare the current *activity*
    - *i.e.*, a named chain of requests scheduled by the infrastructure
  - Properties of an activity are specified using an (unspecified) external tool

46

# Client-side Scheduling

- **The client-side programming model is simple**

```
// Find the scheduling service
RTCosScheduling::ClientScheduler_var scheduler = ...;

// Schedule the 'edge_alarm' activity
scheduler->schedule_activity ("edge_alarm");

controller->edge_alarm ();
```

- Note the Scheduling Service is an optional part of RT-CORBA 1.0

47

# Server-side Scheduling

- Servers can also be configured using the Scheduling Service

```
// Obtain a reference to the scheduling service
RTCosScheduling::ServerScheduler_var scheduler = ...;

CORBA::PolicyList policies; // Set POA policies

// The scheduling service configures the RT policies
PortableServer::POA_var rt_poa = scheduler->create_POA
  ("ControllerPOA",
   PortableServer::POAManager::_nil (),
   policies);

// Activate the servant, and obtain a reference to it.
rt_poa->activate_servant (my_controller);
CORBA::Object_var controller =
   rt_poa->servant_to_reference (my_controller);

// Configure the resources required for this object
// e.g., setup interceptors to control priorities
scheduler->schedule_object (controller, "CTRL_000");
```

## Other Relevant CORBA Features

- RT CORBA leverages other advanced CORBA features to provide a more comprehensive QoS-enabled ORB middleware solution, *e.g.:*
  - ***Timeouts***: CORBA Messaging provides policies to control roundtrip timeouts
  - ***Reliable oneways***: which are also part of CORBA Messaging
  - ***Asynchronous invocations***: CORBA Messaging includes support for type-safe asynchronous method invocation (AMI)
  - ***Real-time analysis & scheduling***: The RT CORBA 1.0 Scheduling Service is an optional compliance point for this purpose
    - However, most of the problem is left for an external tool
  - ***Enhanced views of time*****:** Defines interfaces to control & query "clocks" (orbos/1999-10-02)
  - ***RT Notification Service*****:** Currently in progress in the OMG (orbos/00-06-10), looks for RT-enhanced Notification Service
  - ***Dynamic Scheduling*****:** The Joint Submission (orbos/01-06-09) has be accepted

49

## Controlling Request Timeouts

- **Problem:** How can we keep our **`Controller`** objects from blocking indefinitely when trying to stop a drone that's about to fall off an edge?!
- **Solution:** Override the timeout policy in the **`Drone`** object reference

50

## Applying Request Timeouts

```
// 10 milliseconds (base units are 100 nanosecs)
CORBA::Any val; val <<= TimeBase::TimeT (100000UL);

// Create the timeout policy
CORBA::PolicyList policies (1); policies.length (1);
policies[0] = orb->create_policy
   (Messaging::RELATIVE_RT_TIMEOUT_POLICY_TYPE, val);

// Override the policy in the drone
CORBA::Object_var obj = drone->_set_policy_overrides
   (policies, CORBA::ADD_OVERRIDE);

Drone_var drone_with_timeout = Drone::_narrow (obj);
try {
      drone_with_timeout->speed (0);
}
catch (const CORBA::TIMEOUT &e) { /* Handle exception. */ }
```

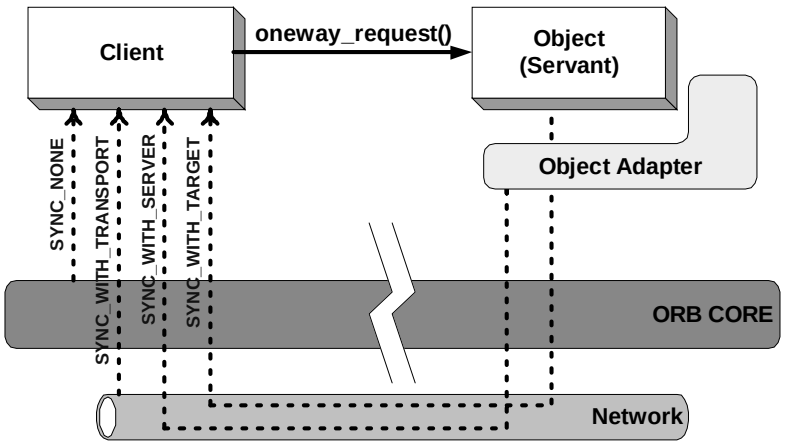
51

## Oneway Calls

- **Problem:** How can we handle the fact that CORBA one-way operation semantics aren't precise enough for real-time applications?
- **Solution:** Use the **`SyncScope`** policy to control one-way semantics

52

# Reliable Oneways



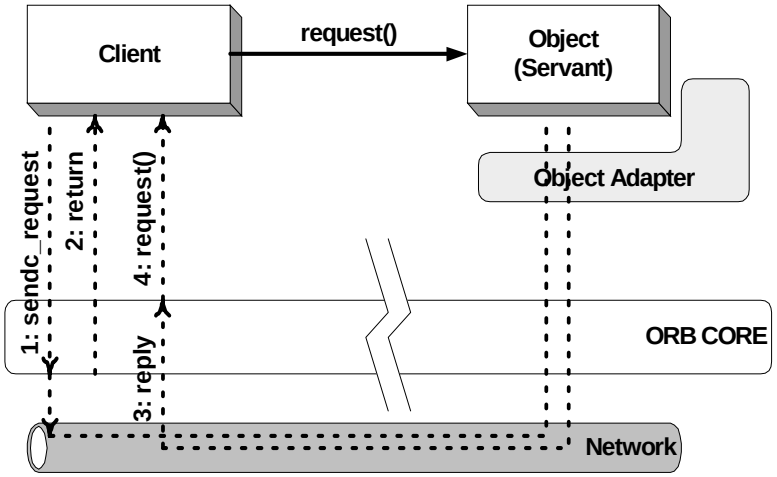

53

# Asynchronous Method Invocation

- **Problem:** How can we simultaneously
  1. Prevent clients from blocking while long-duration requests complete &
  2. Allow many requests to be issued concurrently
- **Solution:** Use the CORBA Asynchronous Method Invocation (AMI) interfaces to separate (in time & space) the thread issuing the request from the thread processing the reply

54

# Asynchronous Method Invocation (AMI)




55

# Open Issues with the Real-Time CORBA Specification

1. No standard APIs for setting & getting priority mappings & priority transforms

2. Few compelling use-cases for server-set client protocol policies

3. Semantic ambiguities
   - Valid policy configurations & their semantics
     - *e.g.,* should a protocol property affect all endpoints or just some?
   - Resource definition & allocation
   - Mapping of threads to connection endpoints on the server

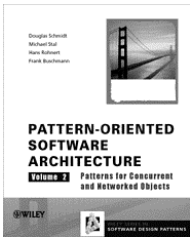
4. The bounds on priority inversions is a quality of implementation
   - No requirement for I/O threads to run at the same priority as request processing threads

**Bottom-line: RT CORBA applications remain overly dependent on implementation details**

56

# Additional Information

- **CORBA 2.6 specification (includes RT CORBA)**
  - `www.omg.org/technology/documents/formal/corbaiiop.htm`
- **Patterns for concurrent & networked OO middleware**
  - `www.posa.uci.edu`
- **Real-time & Embedded CORBA ORBs**
  - `e*ORB <www.vertel.com>`
  - `HighComm <www.highcomm.com>`
  - `ORBacus/E <www.ooc.com>`
  - `ORBexpress <www.ois.com>`
  - `TAO <www.theaceorb.com>`

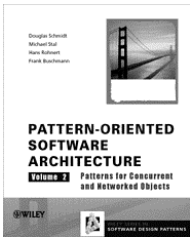



- **CORBA research papers**
  - `www.cs.wustl.edu/~schmidt/corba-research.html`
- **CORBA tutorials**
  - `www.cs.wustl.edu/~schmidt/tutorials-corba.html`
- **CORBA columns (with Steve Vinoski)**
  - `www.cs.wustl.edu/~schmidt/report-doc.html`

57

# Additional Information

- **CORBA 2.4 specification (includes RT-CORBA)**
  - `www.omg.org/technology/documents/formal/corbaiiop.htm`
- **Patterns for concurrent & networked objects**
  - `www.posa.uci.edu`
- **ACE & TAO open-source middleware**
  - `www.cs.wustl.edu/~schmidt/ACE.html`
  - `www.cs.wustl.edu/~schmidt/TAO.html`

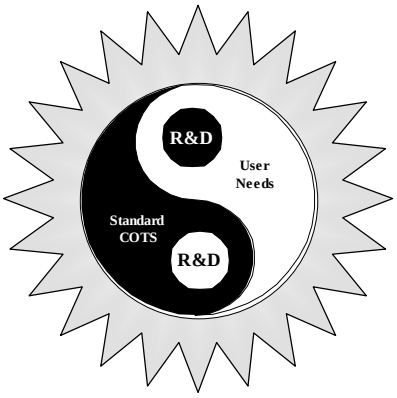



- **CORBA research papers**
  - `www.cs.wustl.edu/~schmidt/corba-research.html`
- **CORBA tutorials**
  - `www.cs.wustl.edu/~schmidt/tutorials-corba.html`
- **CORBA columns**
  - `www.cs.wustl.edu/~schmidt/report-doc.html`

58

# Concluding Remarks

- RT CORBA 1.0 is a major step forward for QoS-enabled middleware
  - *e.g.,* it introduces important capabilities to manage key ORB end-system/network resources
- We expect that these new capabilities will increase interest in--and applicability of--CORBA for distributed real-time & embedded systems
- RT CORBA 1.0 doesn't solve *all* real-time development problems, however
  - It lacks important features:
    - Standard priority mapping manager
    - Dynamic scheduling
      - Addressed in RT CORBA 2.0
  - Portions of spec are under-specified
    - Thus, developers must be familiar with the implementation decisions made by their RT ORB
- Our work on TAO has helped advance middleware for distributed real-time & embedded systems by implementing RT CORBA in an open-source ORB & providing feedback to users & OMG




59