

Realtime CORBA

Joint Revised Submission orbos/98-10-05

Alcatel

Lucent Technologies

Hewlett-Packard

Nortel Networks

Highlander Communications

Objective Interface

INPRISE Corporation

Object-Oriented Concepts

IONA Technologies

Sun Microsystems

Lockheed Martin Federal
Systems

Tri-Pacific Software

Bill Beckwith

Jon Currey

Peter Kortmann

Objective Interface

Highlander

Tri-Pacific

Supporting Organizations



- France Telecom
- Humboldt-University
- MITRE Corp.
- Motorola, Inc.
- University of Rhode Island
- Washington University

Presentation Schedule



- Architectural Overview
25 min. Bill Beckwith
- Realtime CORBA in Detail
1 hour Jon Currey
- Realtime CORBA Scheduling Service
20 min. Peter Kortmann
- Submission Status
- Questions

Realtime CORBA Architecture



- Goal
- Realtime System Components
- Scope
- Definitions
- Referencing Existing Standards

Goal

- Support End-to-End Predictability
 - respecting thread priorities between client and server for resolving resource contention during the processing of CORBA invocations;
 - bounding the duration of thread priority inversions during end-to-end processing;
 - bounding the latencies of operation invocations.

Realtime System Components



- 1. the scheduling mechanisms in the OS;
- 2. the Realtime ORB;
- 3. the communication transport;
- 4. the application(s).

Scope



- affect of

the Realtime ORB

upon

end-to-end predictability

Scope (cont.)

- Pluggable protocols is out of scope
 - Beyond scope of Realtime CORBA
 - Useful to all CORBA
 - Significant enough technology to warrant separate RFP

Definitions

- Thread
- Priority
- Native Thread Priority
- Base Native Thread Priority
- Derived Native Thread Priority
- CORBA Priority
- Priority Inheritance
- Resource
- Mutex
- Schedulable
- Activity

Definitions: Thread

- Thread is a “schedulable entity”
- Generally a Thread is a “single sequence of control flow with an address space”
- Next submission will revise Thread definition

Definitions: Priority

- RTOS will have some discrete representation of a thread priority
- A thread of a higher priority runs at the exclusion of lower priority threads
- Assume:
 - Priority means Thread Priority for this proposal

Definitions: Native Thread Priority

- Simple:
 - The RTOS's representation of priority
- Modeled in Realtime CORBA as a short

Definitions:

Base Native Thread Priority

- “Base” means the priority that the thread was born with or to which its priority was later set

Definitions: Derived Native Thread Priority

- “Derived” means the temporarily boosted priority of a thread caused by some form of priority inheritance or priority ceiling protocol
- The term “active priority” is often used

Definitions: CORBA Priority

- A universal, platform independent priority scheme
- Overcomes the heterogeneity of different native priority schemes
- Allows prioritized CORBA invocations in a consistent fashion between nodes

Definitions: Priority Inheritance

- The temporary boosting of a thread's priority because it holds a resource that higher priority threads may contend or are contending for
- Used to avoid or bound "*priority inversions*": certain situations can occur where a high priority thread can never run because it is waiting for a resource held by a low priority thread and the low priority thread can't run because a medium priority thread is running

Definitions: Resource

- In this proposal:
 - protected by a *RT_CORBA::Mutex*
 - thread, threadpool, transport connection
 - possibly request buffer

Definitions:

Mutex

- The mechanism for coordinating contention for system resources
- A locality constrained IDL interface:
 RT_CORBA::Mutex
- Must implement some form of priority inheritance protocol
 - e.g. simple priority inheritance or some form of priority ceiling locking protocol
 - choice of protocol is ORB implementation specific

Definitions: Schedulable

- A system that can meet all of its deadlines all of the time

Definitions: Activity

- An execution context that “travels” with invocations
 - Like a thread that travels from client-to-server and back again
- A useful concept for describing the semantics of distributed systems
- Not a normative part of the Realtime CORBA specification

Referencing Existing Standards

- Operating System
 - An OS that implements
IEEE POSIX 1003.1-1996
Real-Time Extensions
is sufficient
 - But POSIX is **not** *necessary* or *required*

Realttime CORBA In Detail



Realtime CORBA In Detail

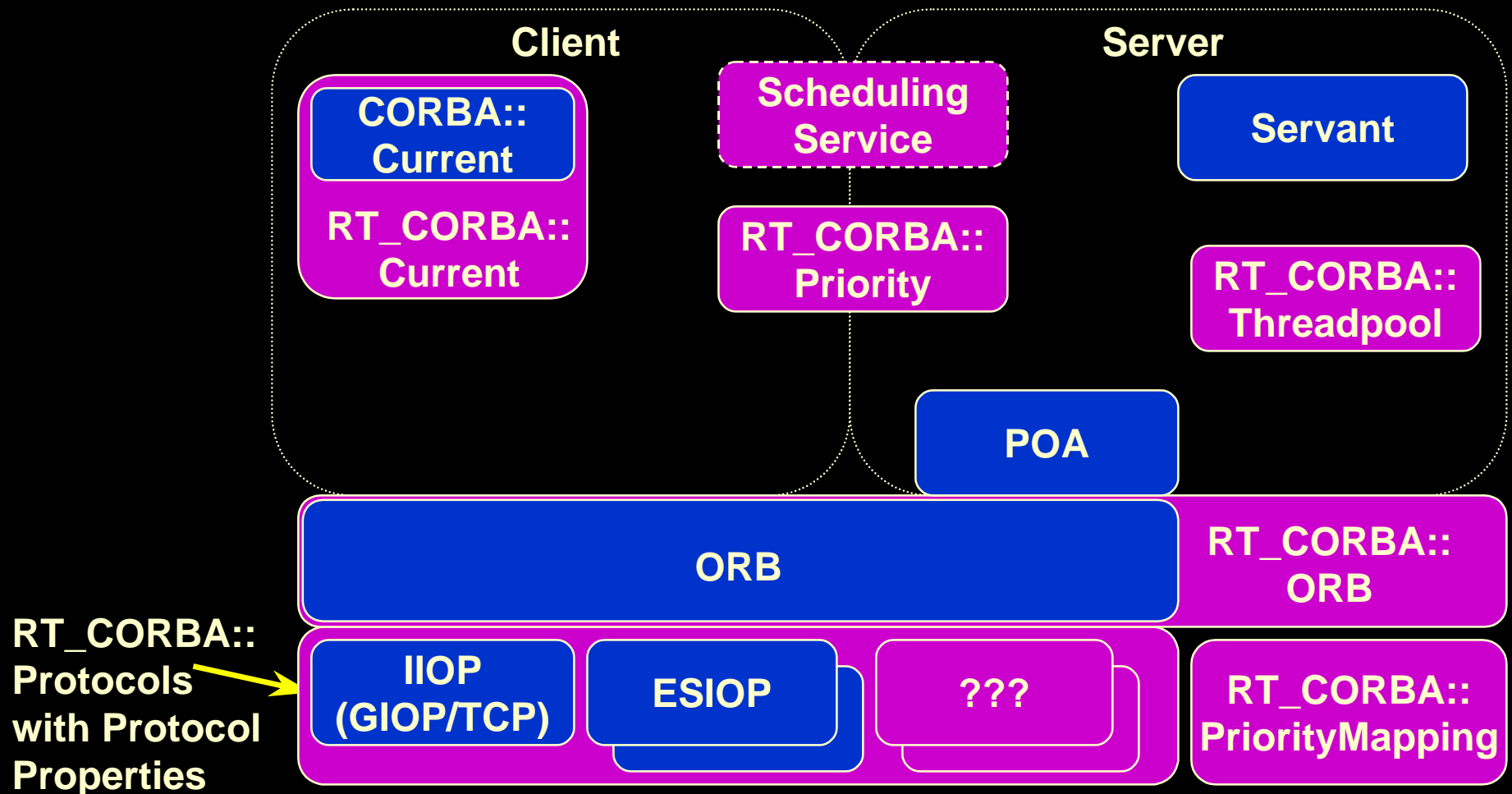


- Realtime CORBA Extensions
- Realtime CORBA Priority
- Priority Propagation
- Server-side Policies
- Client-side Policies

Realtime CORBA Extensions

- Realtime CORBA specified as extensions to CORBA
- module RT_CORBA
 - RT_CORBA::ORB, RT_CORBA::Current
 - new RT_CORBA Policies
 - new RT_CORBA entities
 - Priority, PriorityMapping, Threadpool, Protocol, ProtocolProperties

Realtime CORBA Extensions



RT_CORBA::Priority

```
// IDL
```

```
module RT_CORBA {  
    typedef short Priority;  
    const Priority minPriority = 0;  
    const Priority maxPriority = 32767;  
    ...  
}
```

- Universal, platform independent priority scheme
- Allows prioritized CORBA invocations to be made in a *consistent* fashion between nodes with *different native priority schemes*

RT_CORBA::Priority Mapping

```
typedef short NativePriority;

// Locality Constrained Interface
interface PriorityMapping {

    boolean to_native (
        in Priority corba_priority,
        out NativePriority native_priority );

    boolean to_corba (
        in NativePriority native_priority,
        out Priority corba_priority );
};
```

RT_CORBA::Priority Mapping

- *to_native* and *to_corba* are used by the ORB as callbacks
- *NativePriority* type is defined to represent OS specific native priority scheme
- One *PriorityMapping* installed at any one time per ORB instance
- A particular *PriorityMapping* may choose only to map a sub-range of native or CORBA Priorities

RT_CORBA::Current

```
interface Current : CORBA::Current {  
    attribute RT_CORBA::Priority priority  
};
```

- Allows a *RT_CORBA::Priority* to be assigned to the current thread of execution
- Mapped to a change in underlying native thread priority via *to_native* operation of active *PriorityMapping*

CORBA Priority Propagation

- The *RT_CORBA::Priority* assigned to *RT_CORBA::Current* is passed with any invocation requests, in a new *CorbaPriority* service context, to be added to IOP

```
module IOP {  
    const ServiceId CorbaPriority = ??;  
    // value assigned by OMG  
};
```

CORBA Priority Propagation

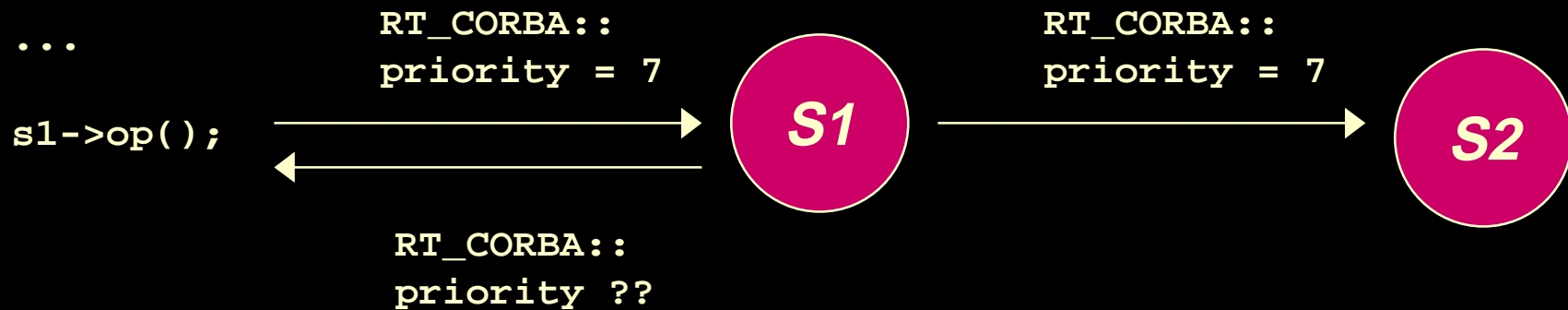
- Considering whether priority should be allowed/required to be returned in reply message
- Whether client priority is used on server-side of invocation depends on the *Server Priority Model* of the POA of the target object
 - set via the *ServerPriorityModelPolicy*
 - considering setting priority at object reference scope
 - could set via a *create_reference_with_priority* operation

ServerPriorityModelPolicy

```
module RT_CORBA {  
  
    enum ServerPriorityModel {  
        CLIENT_PRIORITY_PROPAGATION,  
        SERVER_SET_PRIORITY };  
  
    interface  
        ServerPriorityModelPolicy : CORBA::Policy {  
        readonly attribute ServerPriorityModel  
                                server_priority_model;  
        readonly attribute Priority server_priority;  
    };  
};
```

Client Priority Propagation Model

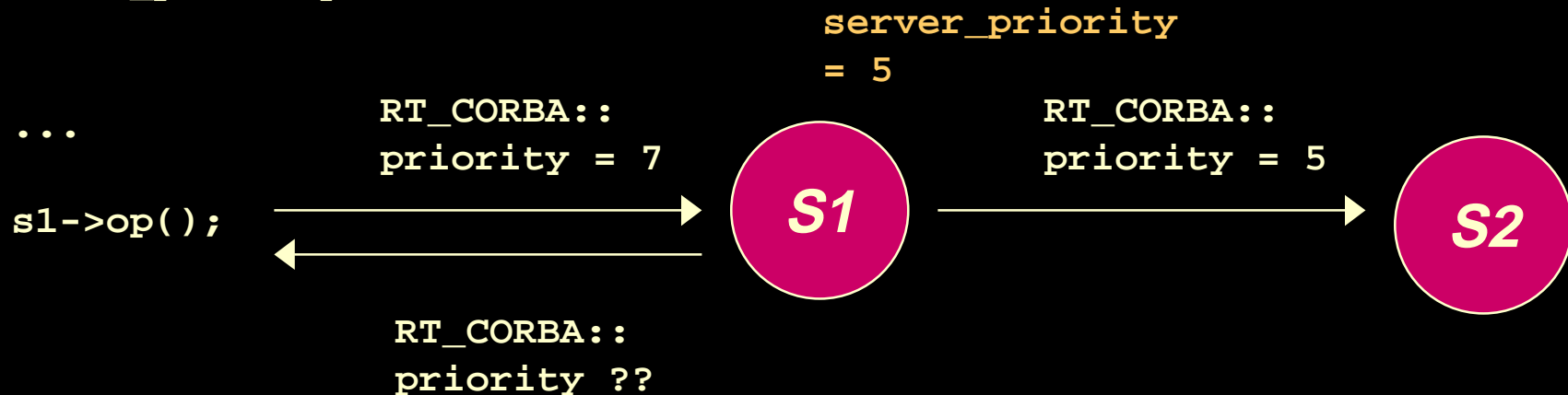
```
rt_current->  
  set_priority(7);
```



- Server-side runs with RT_CORBA Priority propagated to it with invocation (7 in example)
 - hence client's priority is default for onward invocations

Server-Set Priority Model

```
rt_current->  
  set_priority(7);
```



- Server-side runs with RT_CORBA Priority assigned in ServerSetPriorityPolicy (5 in example)
 - server-set priority is default for onward invocations

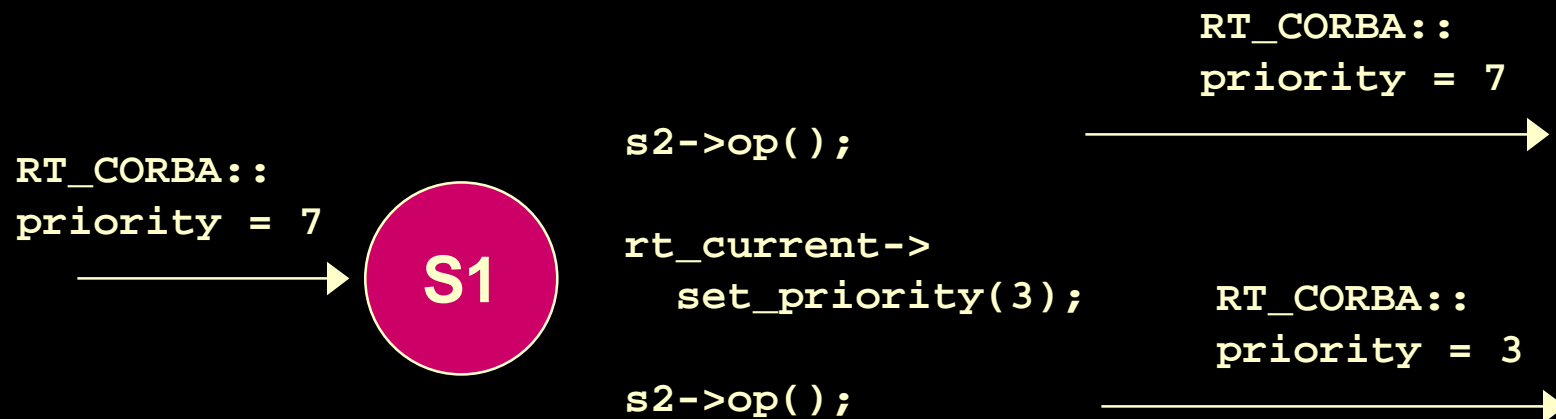
Server-Set Priority Model



- Server-Set Model provided for partitioning systems to schedule on a “node by node” basis
 - not intended for globally schedulable systems
- Considering whether the two models specified are sufficient
 - e.g. a model to support distributed priority ceiling protocol (DPCP)

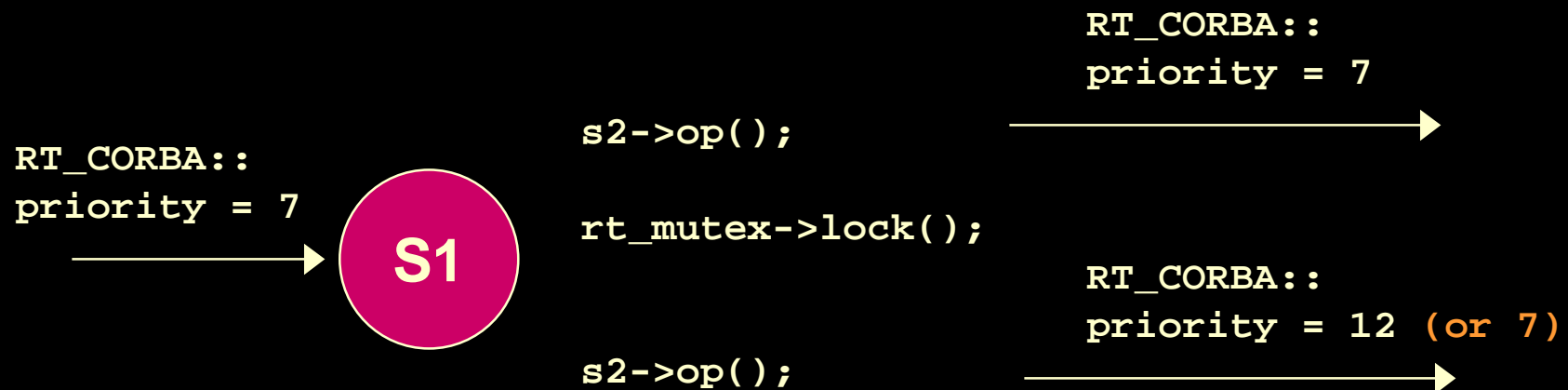
Changing Server-Side Priority

- In both Models, servant (application) code may explicitly change Priority of `RT_CORBA::Current`
 - same effect as on any other application thread
 - system may no longer be schedulable



Changing Server-Side Priority

- Priority may also change automatically, due to priority inheritance
 - application may choose whether derived or base priority is used for onward invocations, via *PriorityDerivationPolicy*



PriorityDerivationPolicy

```
enum PriorityDerivationPolicy {  
    USE_BASE_PRIORITY, USE_DERIVED_PRIORITY };  
  
interface PriorityDerivationPolicy : CORBA::Policy {  
    readonly attribute PriorityDerivationPolicy  
        derivation_policy;  
};
```

- Calculating derived CORBA Priority may or may not involve call to *to_corba* priority mapping operation

RT_CORBA::Mutex

```
interface Mutex {  
    void lock();  
    void unlock();  
    boolean try_lock(  
        in TimeBase::TimeT max_wait);  
};
```

```
interface ORB : CORBA::ORB {  
    Mutex create_mutex();  
};
```

Server-Side Policies



ServerPriorityModelPolicy

PriorityDerivationPolicy

ProtocolPolicy (also used on client)

ThreadpoolPolicy

- Set at POA level
- Defaults can be set at ORB level

ProtocolPolicy

- Enables selection and configuration of communication protocols on a per-POA basis
- Protocols are represented by *RT_CORBA::Protocol* type
 - Protocols defined as ORB/Transport level protocol pairs
- *RT_CORBA::ProtocolList* allows multiple protocols to be supported by one POA
 - Order of protocols in list indicates order of preference

ProtocolPolicy

```
struct Protocol {
    IOP::ProfileId protocol_type;
    ProtocolProperties orb_protocol_props;
    ProtocolProperties trans_protocol_props;
};

typedef sequence <Protocol> ProtocolList;

interface ProtocolPolicy : CORBA::Policy {
    readonly attribute ProtocolList protocols;
};
```

ProtocolProperties

- A *ProtocolProperties* interface to be provided for each configurable protocol supported
 - allows support for proprietary and future standardized protocols
- Interfaces are derived from a base interface type
`interface ProtocolProperties {};`
- Realtime CORBA only specifies a ProtocolProperties for TCP

TCPProtocolProperties

```
interface TCPProtocolProperties :
    ProtocolProperties {
    attribute long send_buffer_size;
    attribute long recv_buffer_size;
    attribute boolean keep_alive;
    attribute boolean dont_route;
    attribute boolean no_delay;
};
```

Threadpools

- Threadpool abstraction is used to manage threads on server-side of Realtime CORBA ORB
 - pre-allocation, partitioning, bounding usage: predictability
- Threadpool parameters
 - number of static threads
 - dynamic thread limit
 - 0 = no limit. same value as static = no dynamic threads
 - thread stacksize
 - default thread priority
 - thread priority will change as required

Threadpool IDL

```
typedef unsigned long ThreadpoolId;

interface ORB : CORBA::ORB {
    ThreadpoolId create_threadpool (
        in unsigned long stacksize,
        in unsigned long static_threads,
        in unsigned long max_threads,
        in Priority default_priority );
    void destroy_threadpool (
        in ThreadpoolId threadpool )
};
```

ThreadpoolPolicy



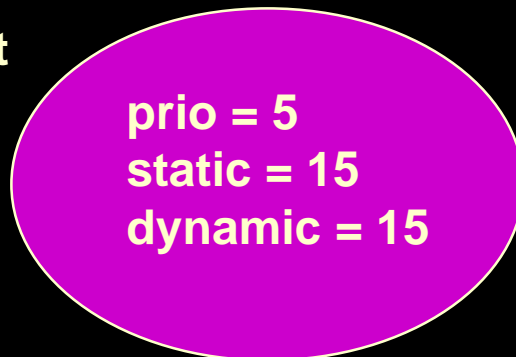
```
interface ThreadpoolPolicy : CORBA::Policy {  
    readonly attribute ThreadpoolId threadpool;  
};
```

- *ThreadpoolId* allows same pool to be shared by multiple POAs

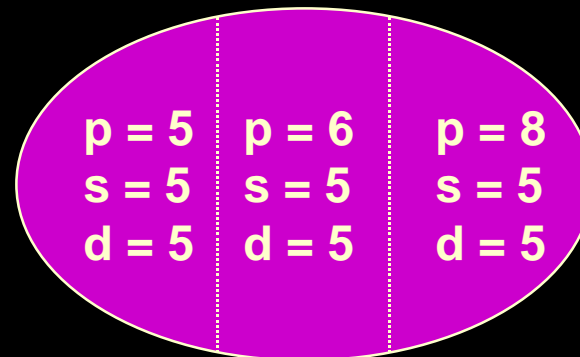
Laned Threadpools

- Alternate way of configuring a Threadpool
 - for applications with detailed knowledge of priority utilization
 - preconfigure ‘lanes’ of threads with different priorities
 - ‘borrowing’ from lower priority lanes can be permitted

**without
lanes**



**with
lanes**



Laned Threadpools

```
struct ThreadpoolLane {
    Priority lane_priority;
    unsigned long static_threads;
    unsigned long max_threads;
};

typedef sequence <ThreadpoolLane> ThreadpoolLanes;

interface ORB : CORBA::ORB {
    ThreadpoolId create_threadpool_with_lanes (
        in unsigned long stacksize,
        in ThreadpoolLanes lanes,
        in boolean allow_borrowing );
};
```

Client-side Policies



ProtocolPolicy

PriorityBandedConnectionPolicy

PrivateConnectionPolicy

- Applied at the level of an individual client binding to an object, using *explicit_bind*
- Defaults may be set at the ORB level

explicit_bind

```
interface ORB : CORBA::ORB {  
  
    exception WrongPolicy {};  
  
    Object explicit_bind (  
        in Object o,  
        in CORBA::PolicyList policies)  
        raises (WrongPolicy);  
  
};
```

explicit_bind



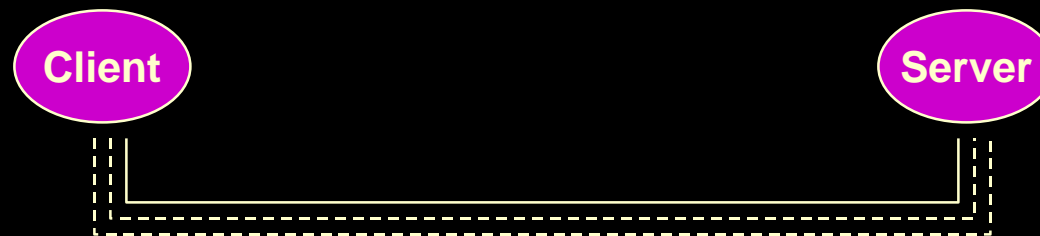
- Applies Realtime CORBA client-side Policies
- Sets up connection prior to first invocation
- Considering overlap with *set_policy_overrides* and *validate_connection* from Messaging

ProtocolPolicy (Client-side)

- Same syntax as server-side
 - *RT_CORBA::Protocol, ProtocolProperties, ProtocolList*
- On client, *ProtocolList* specifies protocols that may be used to make a connection
 - order indicates order of preference
- If *ProtocolPolicy* not set, order of protocols in target object's IOR used as order of preference

PriorityBandedConnectionPolicy

- Multiple connections, to reduce priority inversion
 - each connection handling different priority invocations



- Banding
 - each connection may represent a range of priorities, to allow resources to be traded off against limited inversion
 - may have different ranges in each band, including range of 1

PriorityBandedConnectionPolicy

```
struct PriorityBand {  
    Priority low;  
    Priority high;  
}
```

```
typedef sequence <PriorityBand> PriorityBands;
```

```
interface PriorityBandedConnectionPolicy :
```

```
                                CORBA::Policy {
```

```
    readonly attribute PriorityBands priority_bands;
```

```
};
```

PriorityBandedConnectionPolicy



- Mechanism for propagating band selection from client to server not specified
 - Considering specifying an implicit operation to propagate band configuration from client to server

PrivateConnectionPolicy



- Allows a client to demand a private transport connection to the target object
 - no multiplexing with requests for other target objects within protocol resources controllable by ORB

Realtime CORBA Scheduling Service



Objectives of Realtime CORBA 1.0



- Support for end-to-end predictability
- Enforce fixed-priority realtime scheduling policies across the Realtime CORBA system.
- Respect priorities across nodes
- Bound priority inversions
- Bound latencies

Why a Realtime CORBA Scheduling Service?

- Uses the primitives of the Realtime ORB
- Helps facilitate enforcing various fixed-priority realtime scheduling policies across the Realtime CORBA system
- Abstracts away from the application some of the low-level realtime constructs

Realtime CORBA Scheduling Service



- No new requirements on Realtime or non-Realtime ORBs beyond what appears in the RT CORBA specification or CORBA specification respectively
- Primitives to create a Realtime ORB are sufficient to achieve realtime scheduling
- But effective realtime scheduling is complicated

Realtime CORBA Scheduling Service



- To ensure a uniform scheduling policy, such as global Rate Monotonic Scheduling, requires:
 - the RT ORB primitives be used properly, and
 - their parameters be set properly in all parts of the CORBA system

Realtime CORBA Scheduling Service



- Issues:
 - Determining the proper use and correct parameters can be difficult
 - once it is done, the application code may become more complex, and
 - more complex applications can make analysis and modification more difficult.

Realtime CORBA Scheduling Service

- The Scheduling Service specified meets these needs in the following way:
 - embodies a uniform scheduling policy,
 - the interface abstracts away much of the complexity from application code,
 - an application that uses the Scheduling Service is assured of having a uniform realtime scheduling policy enforced in the entire system

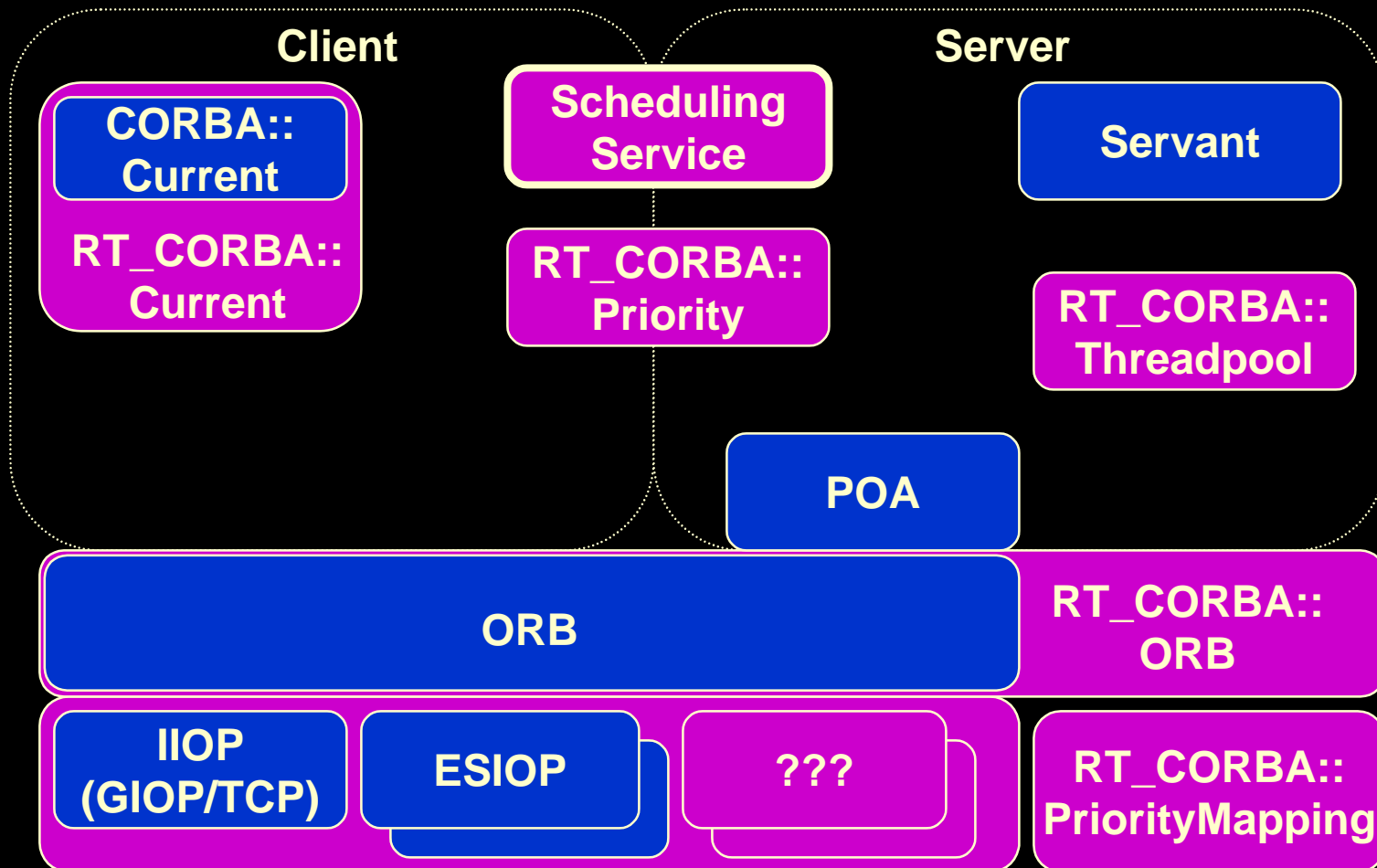
Realtime CORBA Scheduling Service



- A Scheduling Service implementation will choose:
 - CORBA Priorities,
 - POA policies, and
 - priority mappings

in such a way to realize a uniform realtime scheduling policy
- Different implementations can provide different realtime scheduling policies

Realtime CORBA Scheduling Service



Scheduling Service

- Abstraction of scheduling parameters (such as CORBA Priorities) is through the use of "names".
- The application code uses names (strings) to specify CORBA Activities and CORBA objects.
- Names are internally associated with scheduling parameters and policies for the named Activity or the named CORBA object.

Scheduling Service

- This abstraction improves:
 - portability with regard to realtime features,
 - eases uses of the realtime features, and
 - reduces the chance for errors.
- Each name used by the Scheduling Service method invocations must be unique.
- Designed to work in a "closed" CORBA system where fixed priorities are needed for a static set of clients and servers.

Scheduling Service

- The system designer identifies:
 - a static set of CORBA Activities,
 - CORBA objects that the Activities use,
 - scheduling parameters, such as CORBA Priorities, for those Activities and objects,
 - names that are uniquely assigned to those Activities and Objects and
 - the names are associated to scheduling parameters.

Scheduling Service



- This association of names to scheduling parameters is then used to configure the Scheduling Service.
- The Scheduling Service capabilities are not orthogonal to the primitives provided by the Realtime ORB.
- The Scheduling Service is an optional compliance point in this specification

Scheduling Service IDL

```
module RTScheduling {  
    exception UnknownName {};  
  
    // locality constrained interface  
    interface ClientScheduler {  
        void schedule_activity(in string name)  
            raises(UnknownName);  
    };  
};
```

Scheduling Service IDL (cont)

```
// locality constrained interface
interface Serverscheduler {
    PortableServer::POA create_POA (
        in PortableServer::POA parent,
        in string adapter_name,
        in PortableServer::POAManager
                                     a_POAManager,
        in CORBA::PolicyList policies)
}
```

Scheduling Service IDL (cont)

```
raises (
PortableServer::POA::AdapterAlreadyExists,
PortableServer::POA::InvalidPolicy );

void schedule_object( in Object obj,
                     in string name)
raises (UnknownName);
};
};
```

Scheduling Service Semantics



- A CORBA client obtains a local reference to a *ClientScheduler* object
- Whenever the client begins a region of code with a new deadline or priority (indicating a new CORBA Activity), it invokes "schedule_activity" with the name of the new activity

Scheduling Service Semantics



- The Scheduling Service associates a CORBA priority with this name and it invokes appropriate RT ORB and RTOS primitives to schedule this activity
 - if the name is invalid an exception is thrown

Scheduling Service Semantics



- The "create_POA" method accepts parameters allowing it to create a POA
- This POA will enforce all of the non-realtime policies in the Policy List input parameter
- All realtime policies for the returned POA will be set internally by this scheduling service method

Scheduling Service Semantics



- This ensures a selection of realtime policies that is consistent with the scheduling policy being enforced by that implementation
- The implementation should clearly document what Realtime POA policies it will use under various conditions

Scheduling Service Semantics

- "schedule_object" is provided to allow the Scheduling Service to achieve object-level control over scheduling of the object
- RT POA policies in the RT ORB allow some control over the scheduling of object invocations, but must do so for all objects managed by each POA
- Some realtime scheduling, such as priority ceiling concurrency control, requires object-level scheduling

Submission Status



Submission Status



- Resolved many issues
 - Scheduling Semantics
 - Resource Protection
 - Priority Propagation
 - QoS Mapping
- Good foundation for Final Submission

Outstanding Issues

- Relationship to Messaging specification
 - *explicit_bind* and *validate_connection*
- Implicit Bind
 - currently don't specify way to apply policies on implicit bindings
 - also related to resolution of relationship to Messaging

Outstanding Issues (cont.)



- Server Set Priority Model
- Pass Priority on reply messages
- Request Buffers
 - To maintain predictability in overload situations