

A Guided Tour of Real-time CORBA

Part 2b

Self Study Guide

**Worked examples using RT-CORBA 1.0 mechanisms
and use of the optional CORBA 1.0 Scheduling Service**

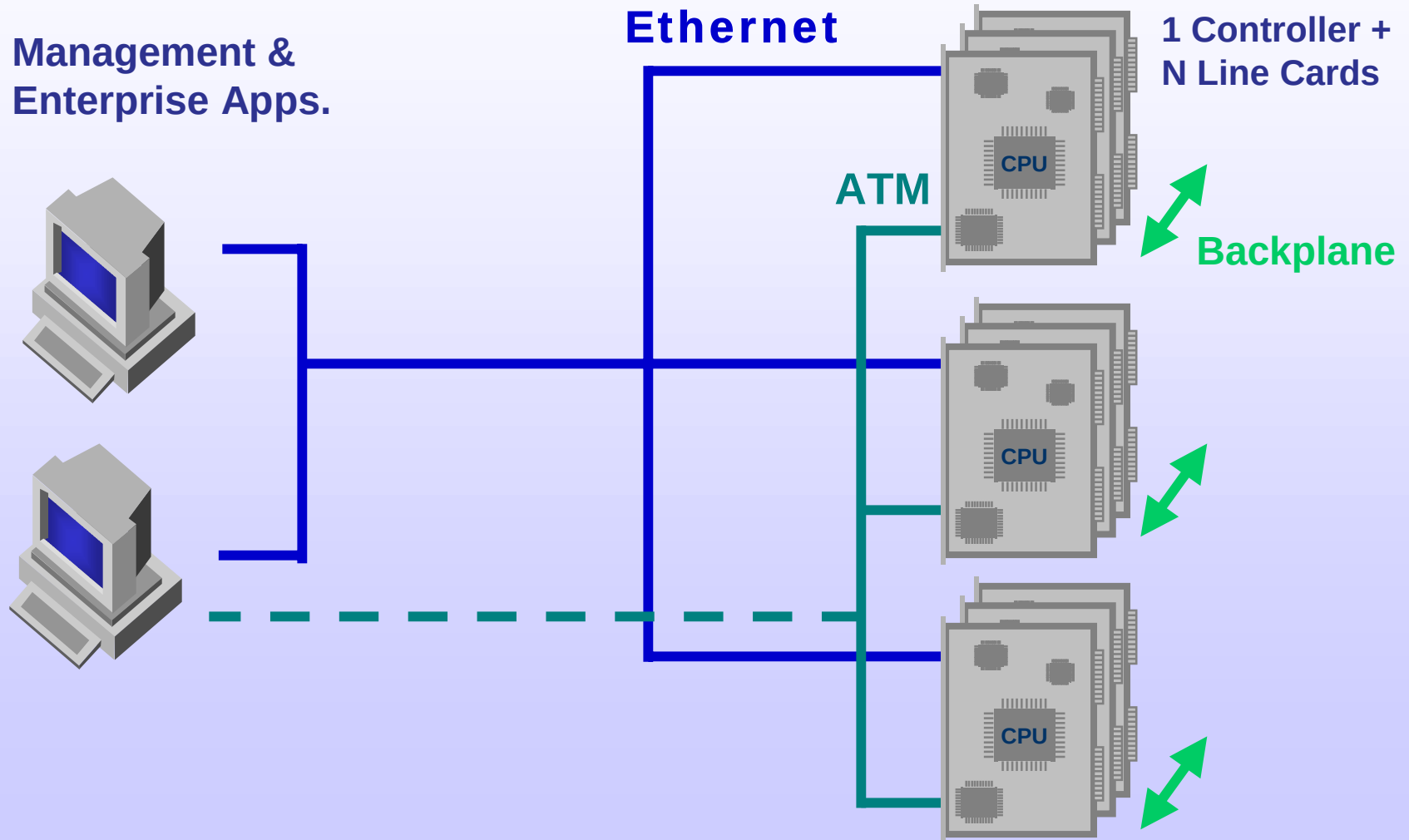
**Dr. Shahzad Aslam Mir
Chief Technology Officer
PrismTech Corporation
sam@prismtechnologies.com
www.prismtechnologies.com**

**Materials reproduced with the kind permission of Jon Currey (Highlander Engineering) &
Mr Tom Cox (Tripacific Corporation)**

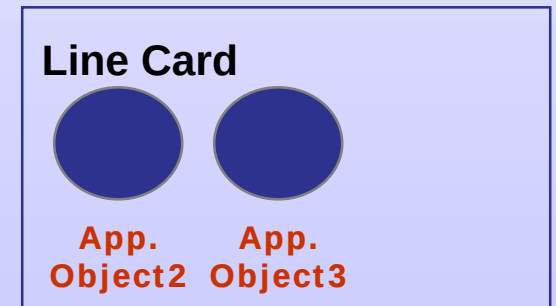
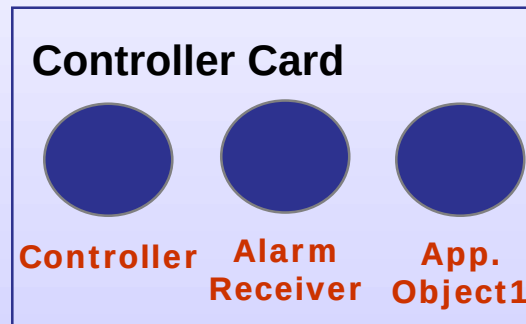
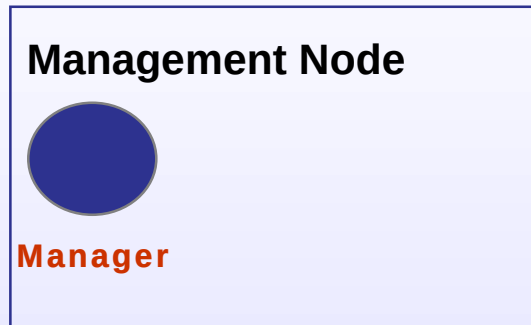
Table of Contents

- Example 1
 - Server Priority Model
 - Threadpool configuration
 - RTCORBA::Current
 - Protocol configuration
- Example 2
 - Client Priority Model
 - Laned Threadpools
 - Priority Banding
- Example 3
 - Priority Mapping
 - RTCORBA Mutex
- Example 4
 - Scheduling Service

Network Management



Simplified Object Model



Object IDL

// IDL

```
interface Manager {  
  
    void registerController (  
        in Controller c );  
    void removeController (  
        in Controller c );  
  
    StatusList reportStatus ();  
    ...  
};
```

```
interface Controller {  
  
    void registerObject (  
        in Object o );  
  
    StatusList reportStatus ();  
    ...  
};
```

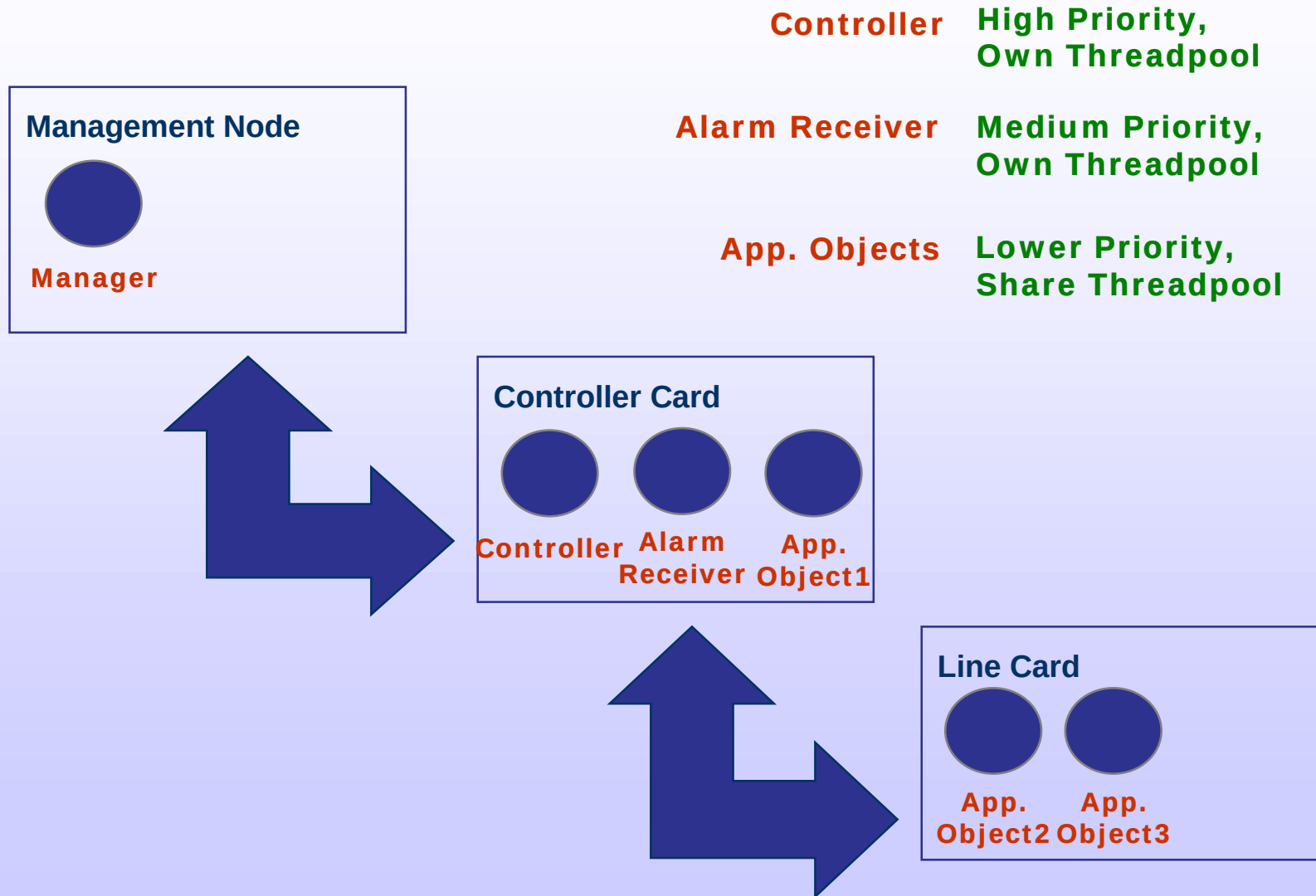
```
interface AlarmReceiver {  
    void receiveAlarm (  
        in Alarm a );  
};
```

```
interface AppObject1 {  
    ...  
};
```

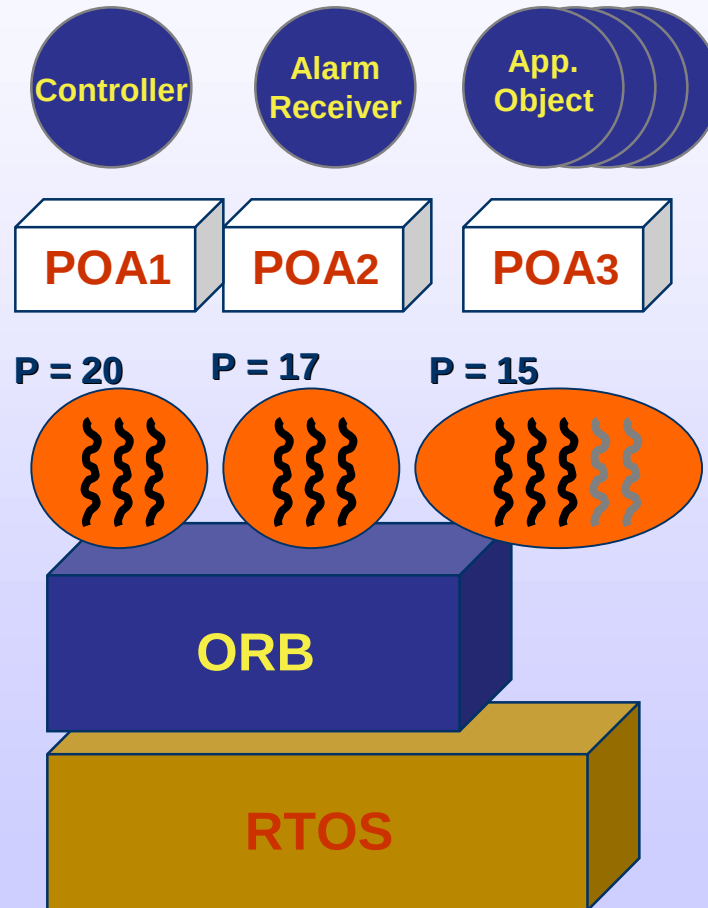
```
interface AppObject2 {  
    ...  
};
```

```
interface AppObject3  
{  
    ...  
};
```

Real-Time Design Considerations



Controller Card Configuration



Initialize CORBA

```
// C++
```

```
CORBA::ORB_ptr orb;
```

```
PortableServer::POA_ptr root_poa;
```

```
void initController()
```

```
{
```

```
// initialize ORB
```

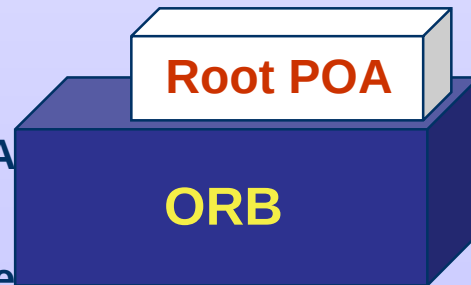
```
    orb = CORBA::ORB_init(argc, argv, orb_id);
```

```
// obtain Root Object Adapter
```

```
    CORBA::Object_var ref1 =
```

```
        orb->resolve_initial_references("RootPOA");
```

```
    root_poa = PortableServer::POA::_narrow(ref1);
```



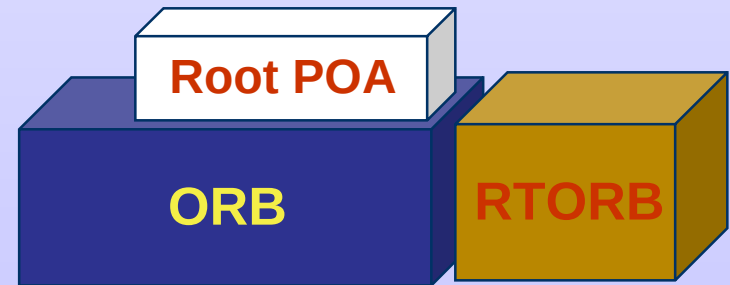
Access Real-Time CORBA API

```
RTCORBA::RTORB_ptr rt_orb;
```

```
// access Real-Time extensions to ORB
```

```
CORBA::Object_var ref2 =  
    orb->resolve_initial_references("RTORB");
```

```
rt_orb = RTCORBA::RTORB::_narrow(ref2);
```



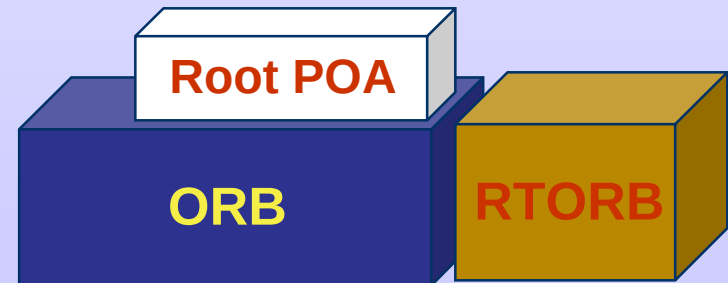
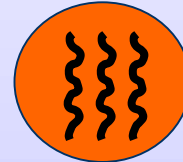
Create Threadpool

```
RTCORBA::ThreadpoolId controller_tpool;  
RTCORBA::ThreadpoolPolicy_ptr threadpool_policy;
```

```
// create Threadpool for Controller  
controller_tpool = rt_orb->create_threadpool(  
    30000, // stacksize  
    3, // num static threads  
    0, // num dynamic threads  
    20, // default thread priority  
    0, 0, 0);
```

```
// create Threadpool Policy object for that Threadpool  
threadpool_policy = rt_orb->  
    create_threadpool_policy(controller_tpool);
```

P = 20



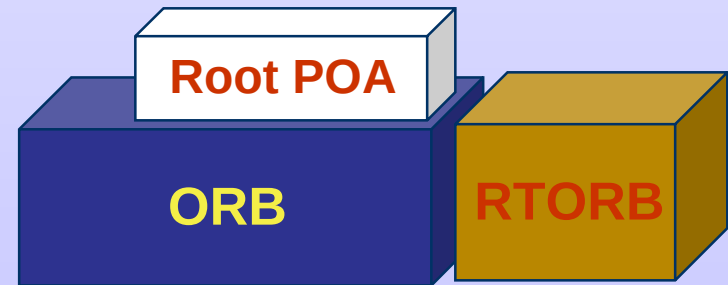
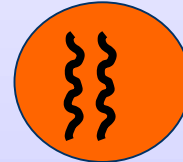
Prepare POA Configuration

```
RTCORBA::PriorityModelPolicy_var priority_model_policy;  
CORBA::PolicyList controller_policies(2);
```

```
// create Server Declared Priority Model policy object  
priority_model_policy =  
    rt_orb->create_priority_model_policy  
        (RTCORBA::SERVER_DECLARED,  
         20); // default object priority
```

```
// assemble policy list for initializing  
// Controller object adapter  
controller_policies[0] = priority_model_policy;  
controller_policies[1] = threadpool_policy;
```

P = 20



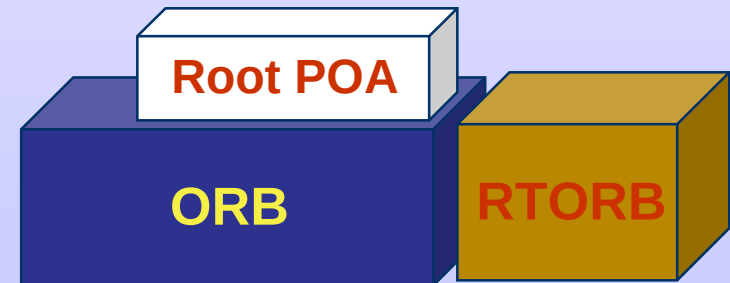
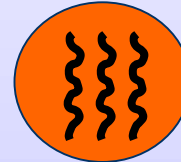
Create Controller POA

```
PortableServer::POA_ptr controller_poa;
```

```
// create Object Adapter  
controller_poa =  
    root_poa->create_POA("ControllerPOA", 0,  
        controller_policies);
```

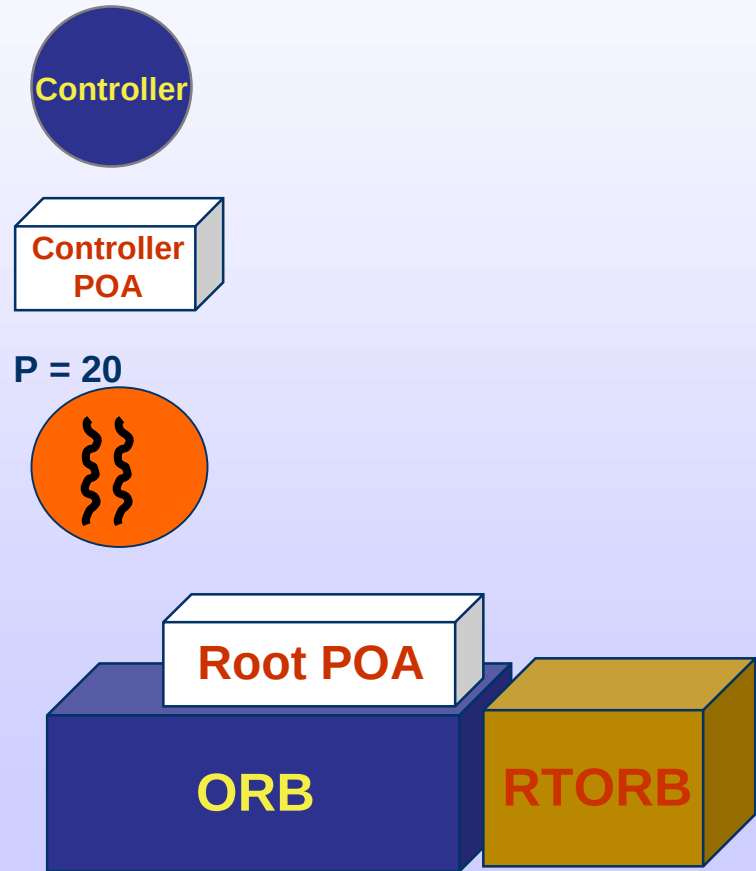


P = 20

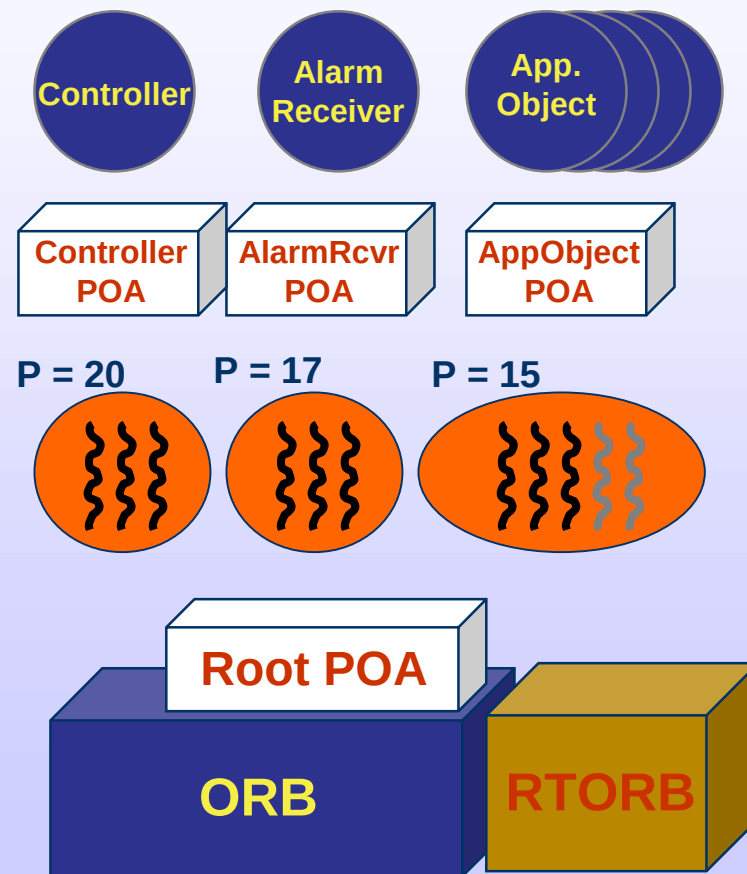


Create Controller Object

```
ControllerServant * controller_servant;  
  
// create Controller object  
controller_servant = new ControllerServant();  
  
// activate object  
controller_poa->  
    activate_object(controller_servant);
```



Complete CORBA Configuration



(Per-Object Priority Configuration)

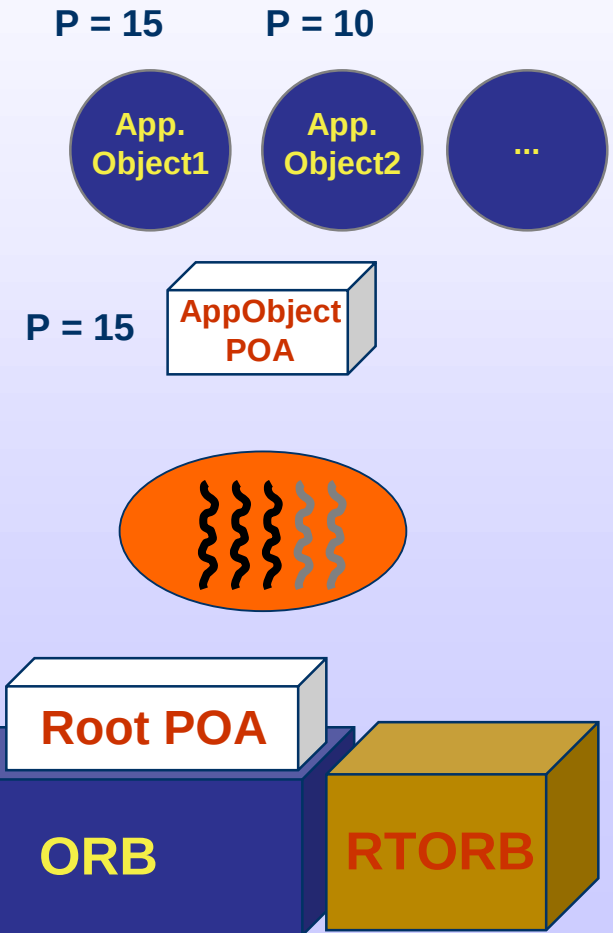
Server Declared Model only

```
AppObjectServant * app_object1, * app_object2;  
app_object1 = new AppObjectServant();  
app_object2 = new AppObjectServant();
```

```
// object activated through normal PortableServer API  
// will have the (default) priority assigned to its POA  
appobject_poa->activate_object(app_object1);
```

```
// priority can be set to a different value by activating  
// through RTPortableServer API  
RTPortableServer::POA_var appobject_poa_rt =  
    RTPortableServer::POA::_narrow(app_object_poa);
```

```
appobject_poa_rt  
    ->activate_object_with_priority(app_object2, 10);
```



Client Priority Configuration

```
void main()
{

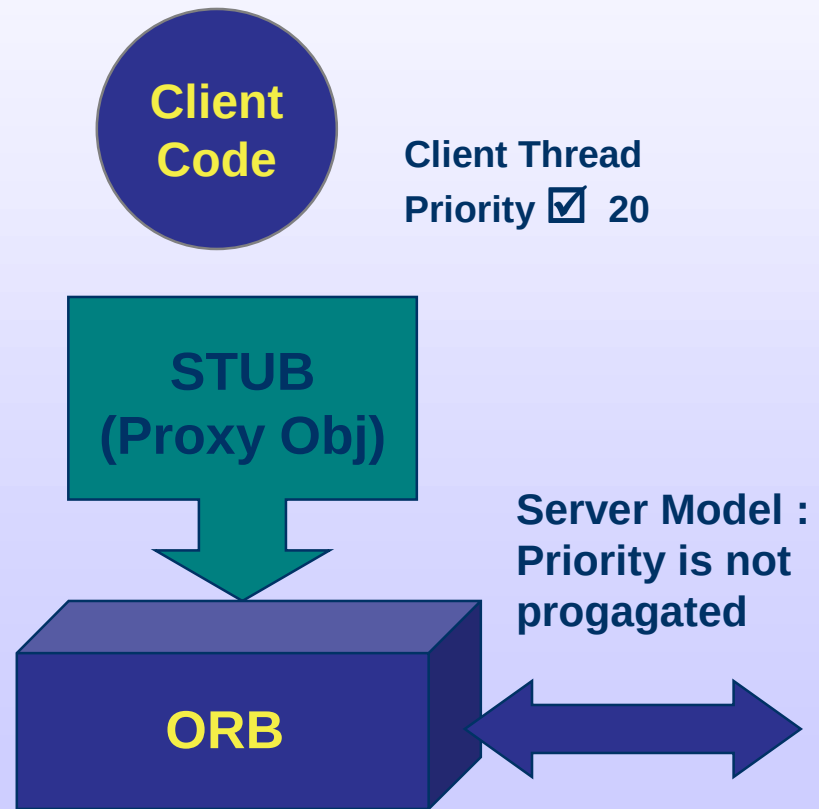
// initialize ORB
orb = CORBA::ORB_init(argc, argv, orb_id);

// obtain handle to 'RT Current'
CORBA::Object_var ref =
    orb->resolve_initial_references("RTCurrent");

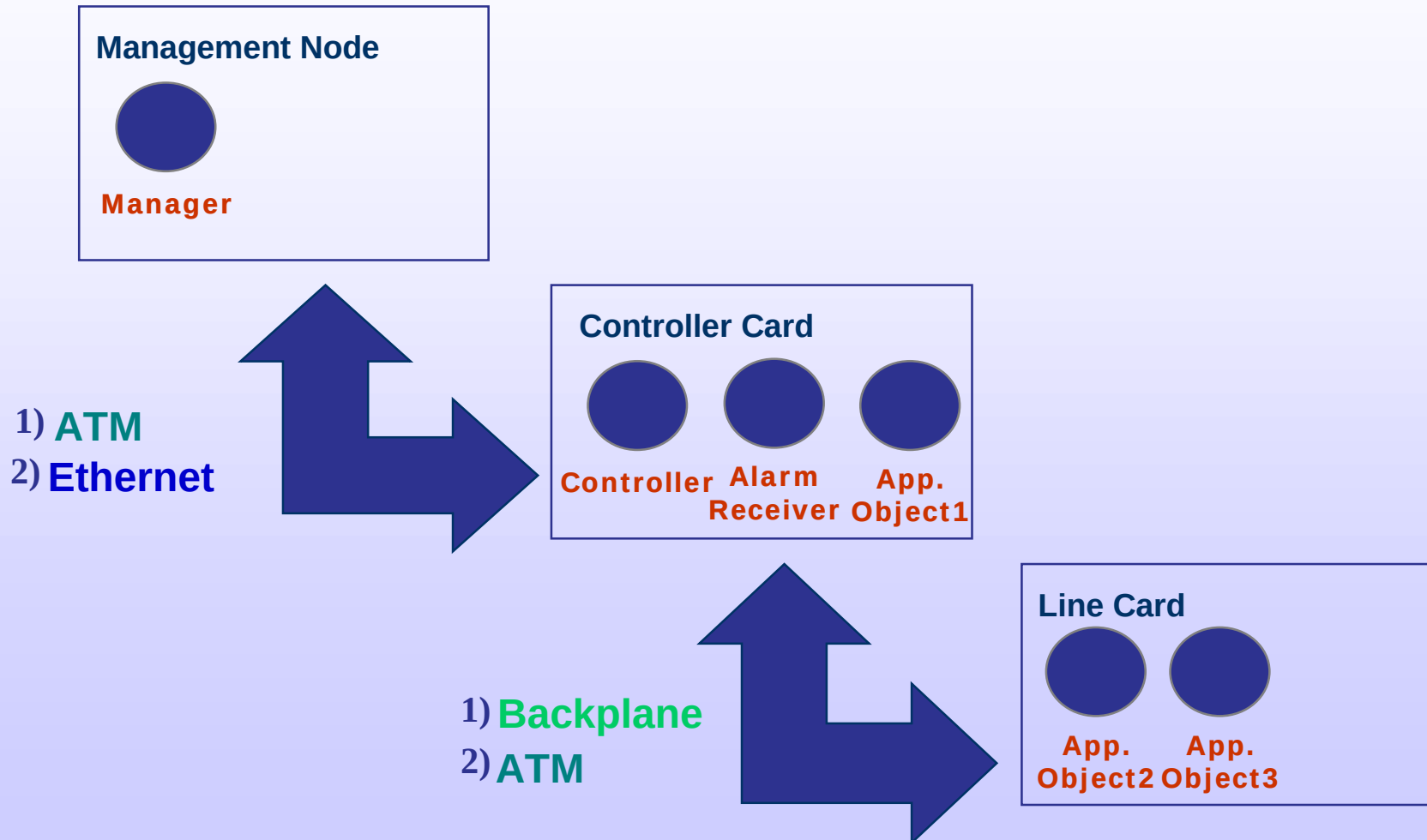
    RTCORBA::Current_var rtcurent =
        RTCORBA::Current::_narrow(ref);

// set priority of current thread
rtcurent->the_priority(20);

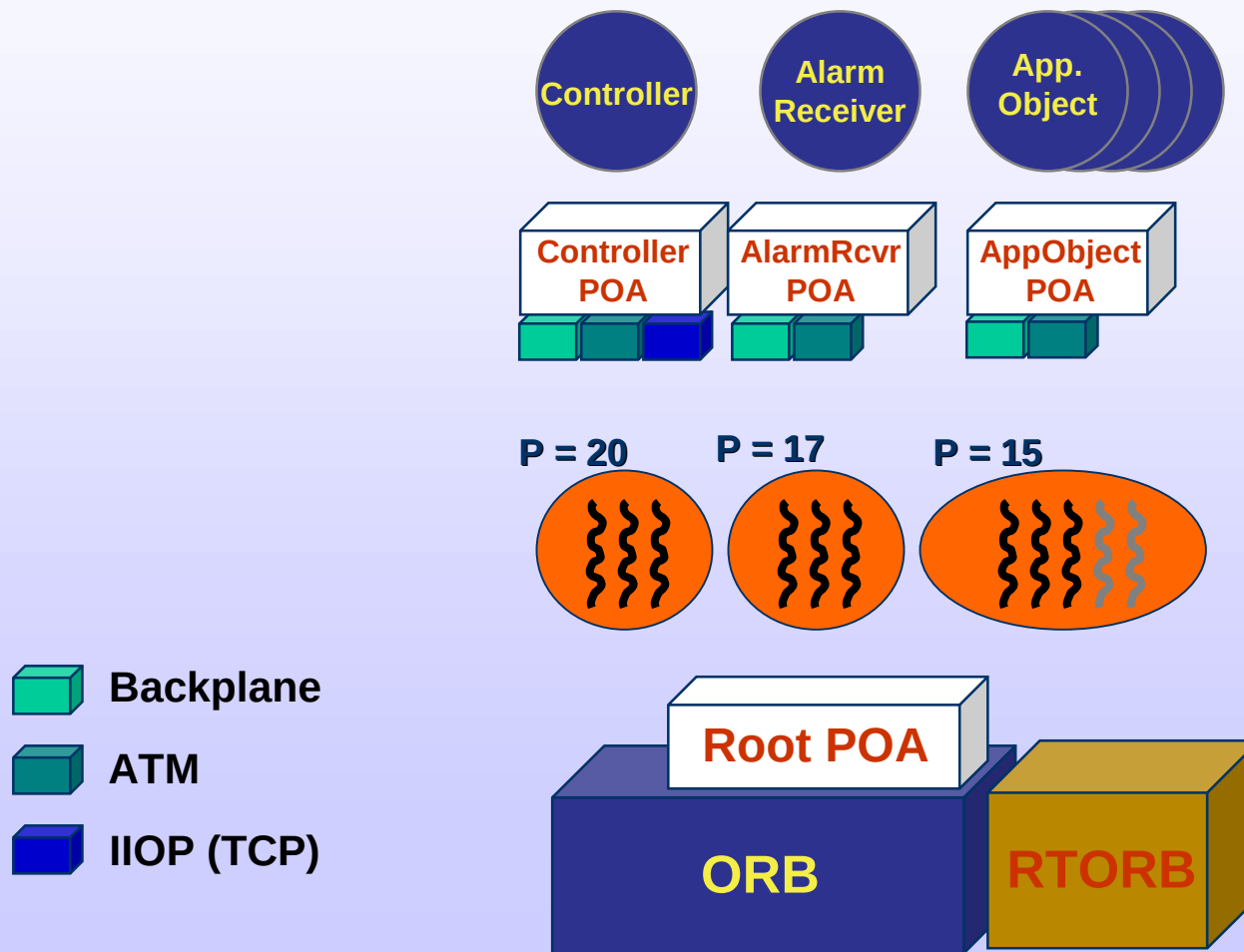
// invoke on CORBA Objects
// Priority is _not_ propagated to
// CORBA Objects that use the Server Model
alarm_handler->receive alarm(a);
```



Protocol Selection



Server-Side Protocol Selection



Server-Side Protocol Selection

```
// create ProtocolPolicy for each protocol want to use
RTCORBA::Protocol ATM_giop(GIOP_AAL5,
                           0, // default GIOP config
                           0) // default AAL5 config
RTCORBA::Protocol backplane_giop(GIOP_BP,
                                  0, // default GIOP config
                                  0) // default BP prot config

// create ProtocolList with protocols in order of preference
RTCORBA::ProtocolList alarm_receiver_protocols(2);
alarm_receiver_protocols[0] = backplane_giop;
alarm_receiver_protocols[1] = ATM_giop;

// use ProtocolList to create a Server Protocol Policy
// object. Add it to the Policy List used to configure POA
alarm_receiver_policy_list[2] = rtorb->
    create_server_protocol_policy( alarm_receiver_protocols );
```

**Protocol Preference in
Object Reference**



Client-Side Protocol Selection

```
// create ProtocolPolicy for each protocol want to try using  
RTCORBA::Protocol ATM_giop(GIOP_AAL5, 0, 0);
```

```
// create ProtocolList with protocol(s) (in order of preference)  
RTCORBA::ProtocolList client_protocols(1);  
client_protocols[0] = ATM_giop;
```

```
// use ProtocolList to create a Client Protocol Policy object  
RTCORBA::ClientProtocolPolicy_var  
client_protocol_policy = rtorb->  
    create_client_protocol_policy( alarm_receiver_protocols );
```

```
// apply the Policy at one of the client-side scopes specified  
// in the CORBA QoS Policy Framework (ORB, Current or Object)
```

```
CORBA::Object_ptr ref = // Object reference for target object  
CORBA::PolicyList policy_list(1) = client_protocol_policy;  
CORBA::Object_var new_ref =  
    ref->set_policy_overrides( policy_list,  
                              CORBA::ADD_OVERRIDE);
```

```
// binding to the target CORBA Object via new_ref  
// will only occur via one of the protocols indicated
```

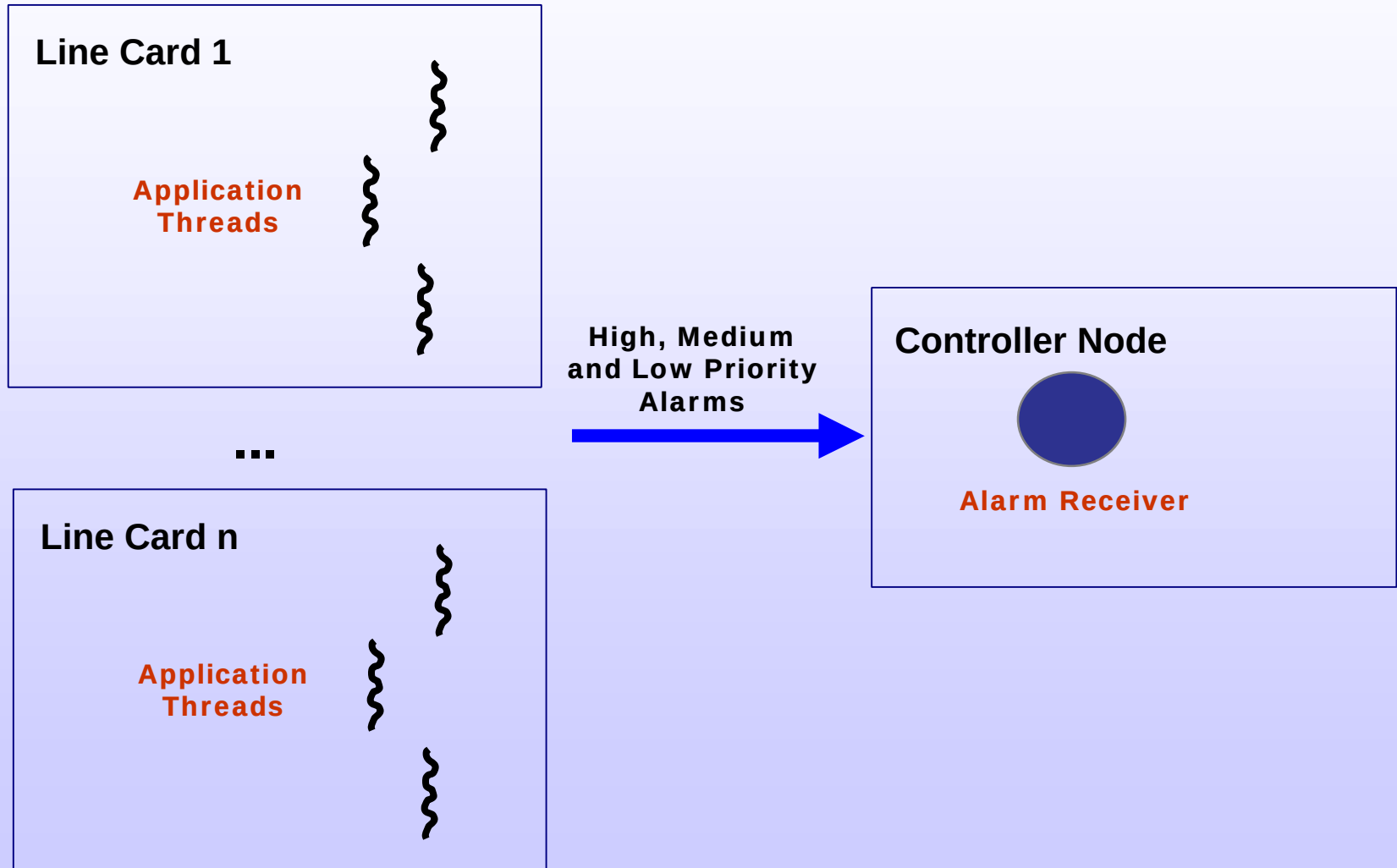
**Protocol Preference in
Object Reference**



Agenda

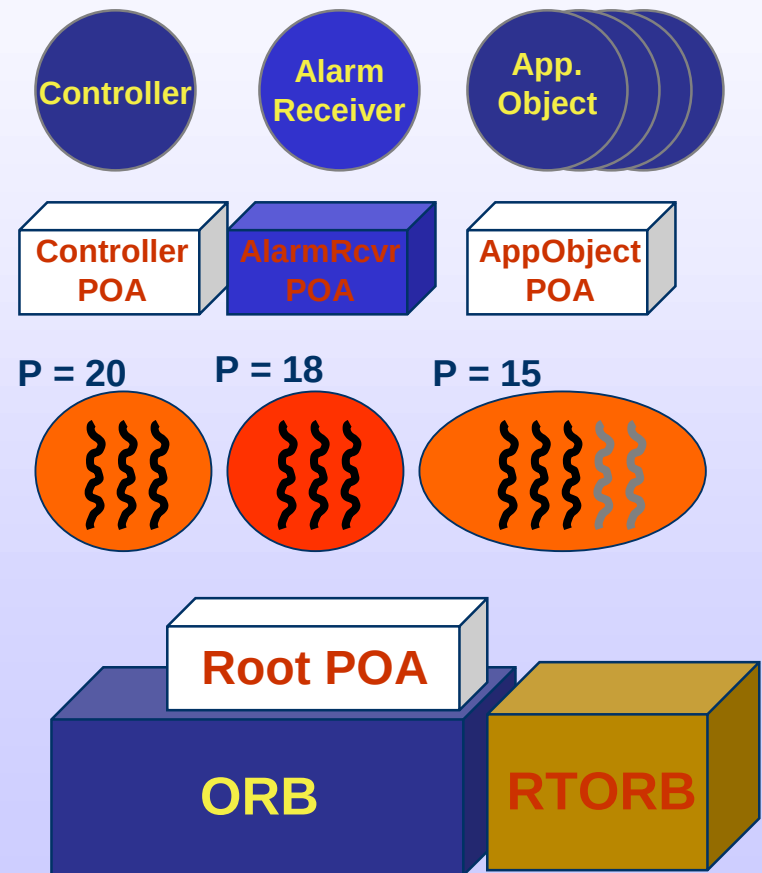
- Example 1
 - Server Priority Model
 - Threadpool configuration
 - RTCORBA::Current
 - Protocol Selection
- Example 2
 - Client Priority Model
 - Laned Threadpools
 - Priority Banding
- Example 3
 - Priority Mapping
 - RTCORBA Mutex
- Example 4
 - Scheduling Service

Prioritized Alarm Receiver



Configure Alarm Receiver Differently

- Client Declared Priority Model
- Laned Threadpool
- Priority Banded Connections



Create Laned Threadpool

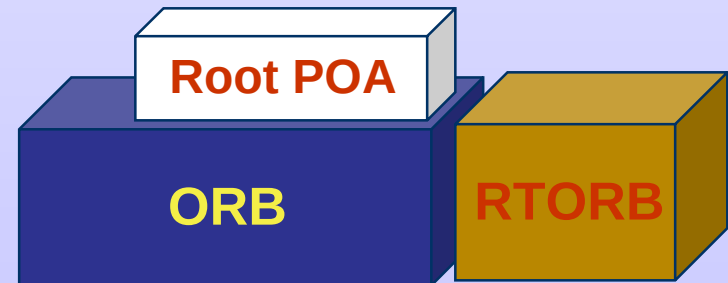
```
// define three lanes, for High, Medium and  
// Low priority alarms
```

```
RTCORBA::ThreadpoolLane high_lane =  
    { 18, // Priority  
      3, // Static threads  
      0 }; // Dynamic threads
```

```
RTCORBA::ThreadpoolLane high_lane =  
    { 17, // Priority  
      3, // Static threads  
      0 }; // Dynamic threads
```

```
RTCORBA::ThreadpoolLane high_lane =  
    { 16, // Priority  
      3, // Static threads  
      0 }; // Dynamic threads
```

```
RTCORBA::ThreadpoolLanes pool_lanes[3];  
pool_lanes[0] = high_lane;  
pool_lanes[1] = medium_lane;  
pool_lanes[2] = low_lane;
```



Create Laned Threadpool

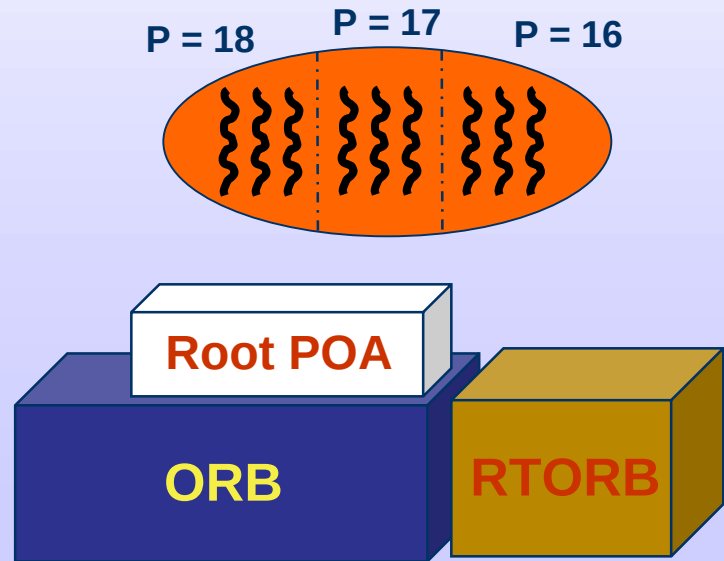
```
RTCORBA::ThreadpoolId alarm_receiver_tpool;  
RTCORBA::ThreadpoolPolicy_ptr threadpool_policy;
```

```
// create Threadpool for Alarm Receiver
```

```
alarm_receiver_tpool =  
    rt_orb->create_threadpool_with_lanes (  
        30000,      // stacksize  
        pool_lanes, // Lanes  
        FALSE,      // Don't allow thread borrowing  
        0, 0, 0 );
```

```
// create Threadpool Policy object for that Threadpool
```

```
threadpool_policy =  
    rt_orb->create_threadpool_policy (  
        alarm_receiver_tpool );
```



Specify Banded Connections

```
// specify three bands, corresponding to the  
// Threadpool lanes
```

```
RTCORBA::PriorityBand high_band = { 18, 18 };
```

```
RTCORBA::PriorityBand medium_band = { 17, 17 };
```

```
RTCORBA::PriorityBand low_band = { 16, 16 };
```

```
RTCORBA::PriorityBands priority_bands(3);
```

```
priority_bands[0] = high_band;
```

```
priority_bands[1] = medium_band;
```

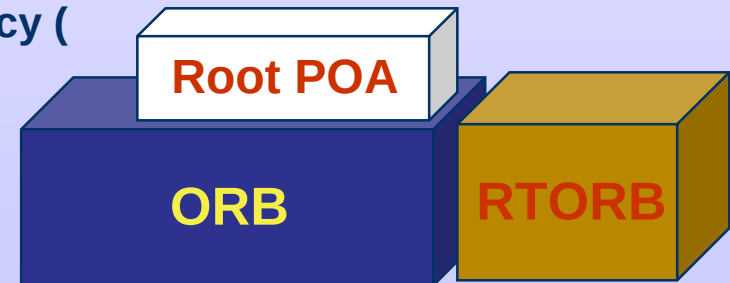
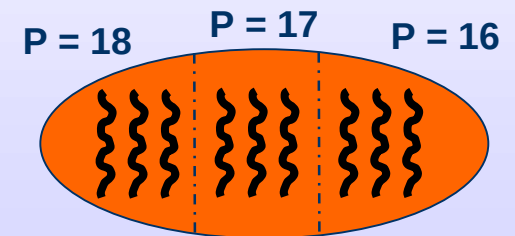
```
priority_bands[2] = low_band;
```

```
// create a Banded Connection Policy object
```

```
RTCORBA::PriorityBandedConnectionPolicy
```

```
banding_policy =
```

```
rt_orb->create_priority_banded_connection_policy (  
    priority_bands );
```



Client Propagated Priority Model

```
RTCORBA::PriorityModelPolicy_var priority_model_policy;
```

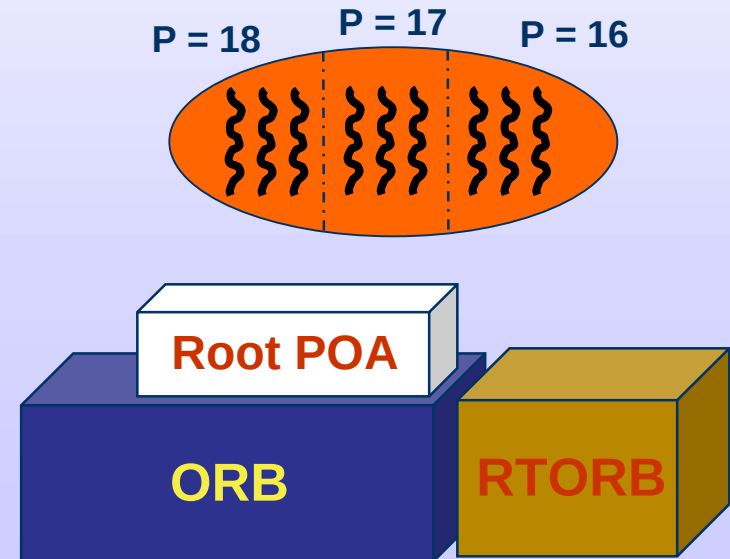
```
// create Client Propagated Priority Model policy object
```

```
priority_model_policy =
```

```
rt_orb->create_priority_model_policy (
```

```
    RTCORBA::CLIENT_PROPAGATED,
```

```
    16); // priority handle non-RT clients at
```



Create Alarm Receiver POA

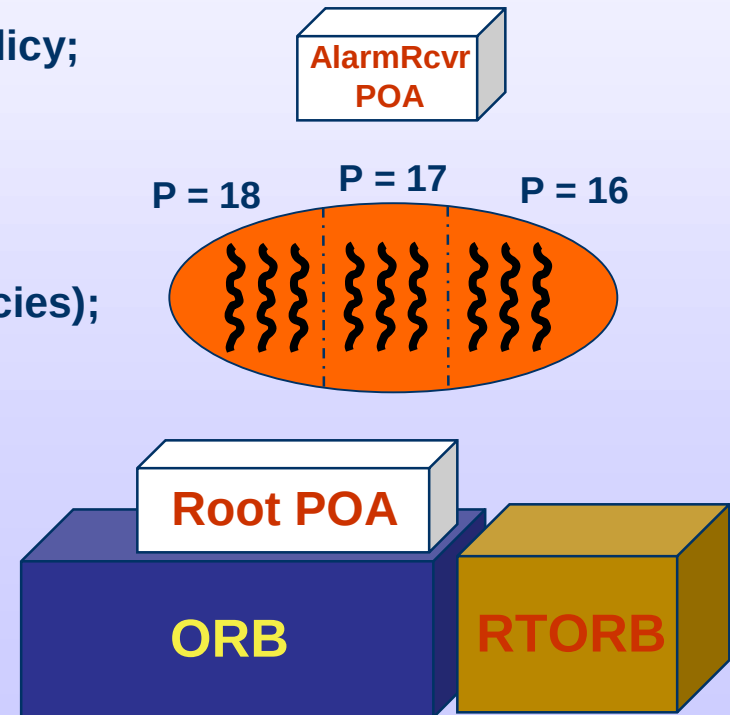
```
PortableServer::POA_ptr alarm_receiver_poa;
```

```
// prepare Policy List for Alarm Receiver POA
```

```
CORBA::PolicyList alarm_receiver_policies[3];  
alarm_receiver_policies[0] = threadpool_policy;  
alarm_receiver_policies[1] = banding_policy;  
alarm_receiver_policies[2] = priority_model_policy;
```

```
// create Object Adapter
```

```
alarm_receiver_poa =  
    root_poa->create_POA("AlarmRcvrPOA", 0,  
        alarm_receiver_policies);
```



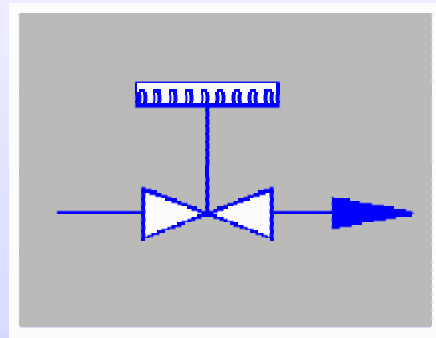
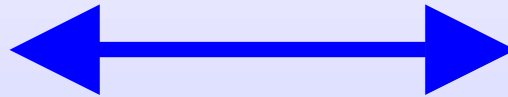
Agenda

- Example 1
 - Server Priority Model
 - Threadpool configuration
 - RTCORBA::Current
 - Protocol configuration
- Example 2
 - Client Priority Model
 - Laned Threadpools
 - Priority Banding
- Example 3
 - Priority Mapping
 - RTCORBA Mutex
- Example 4
 - Scheduling Service

Electronic Control Valve System

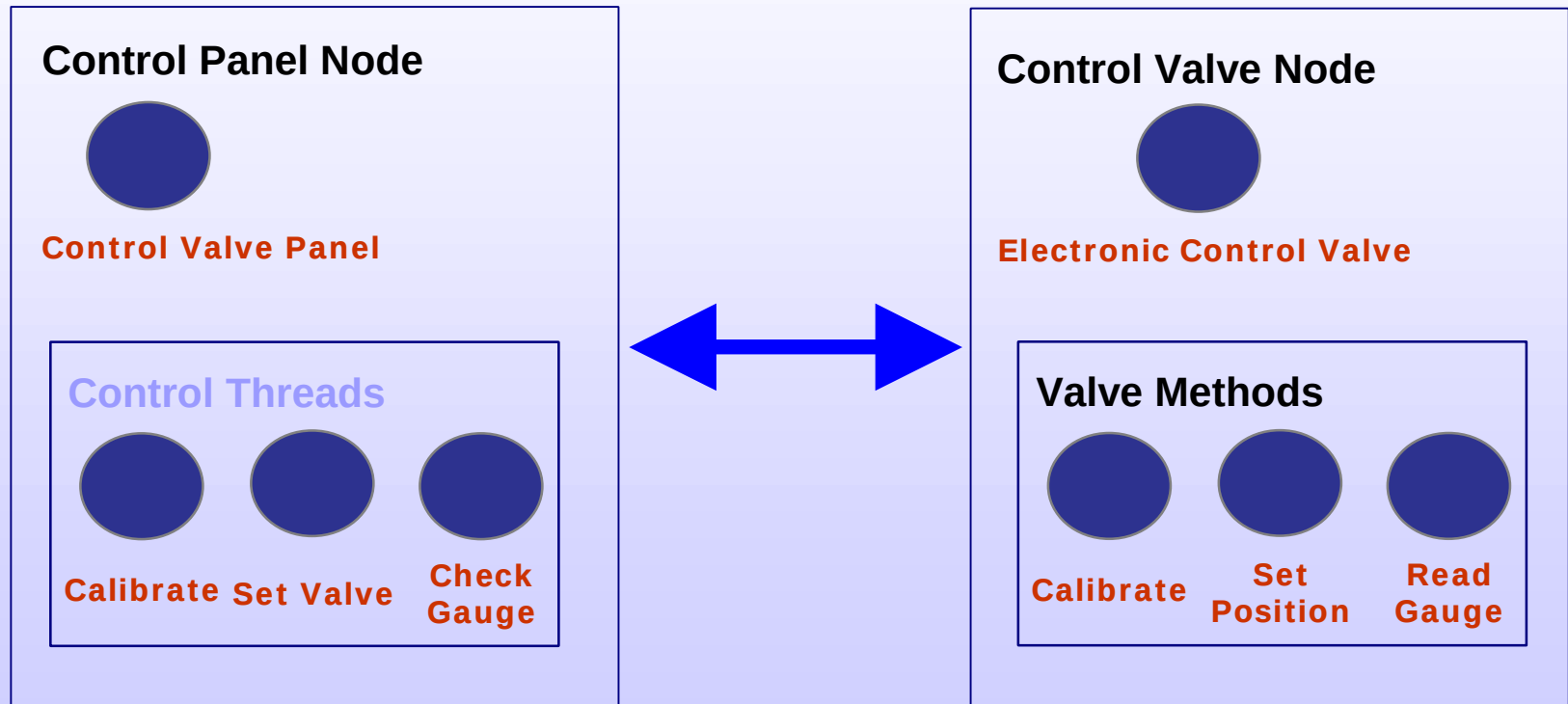


Control Panel



Control Valve

Object Model



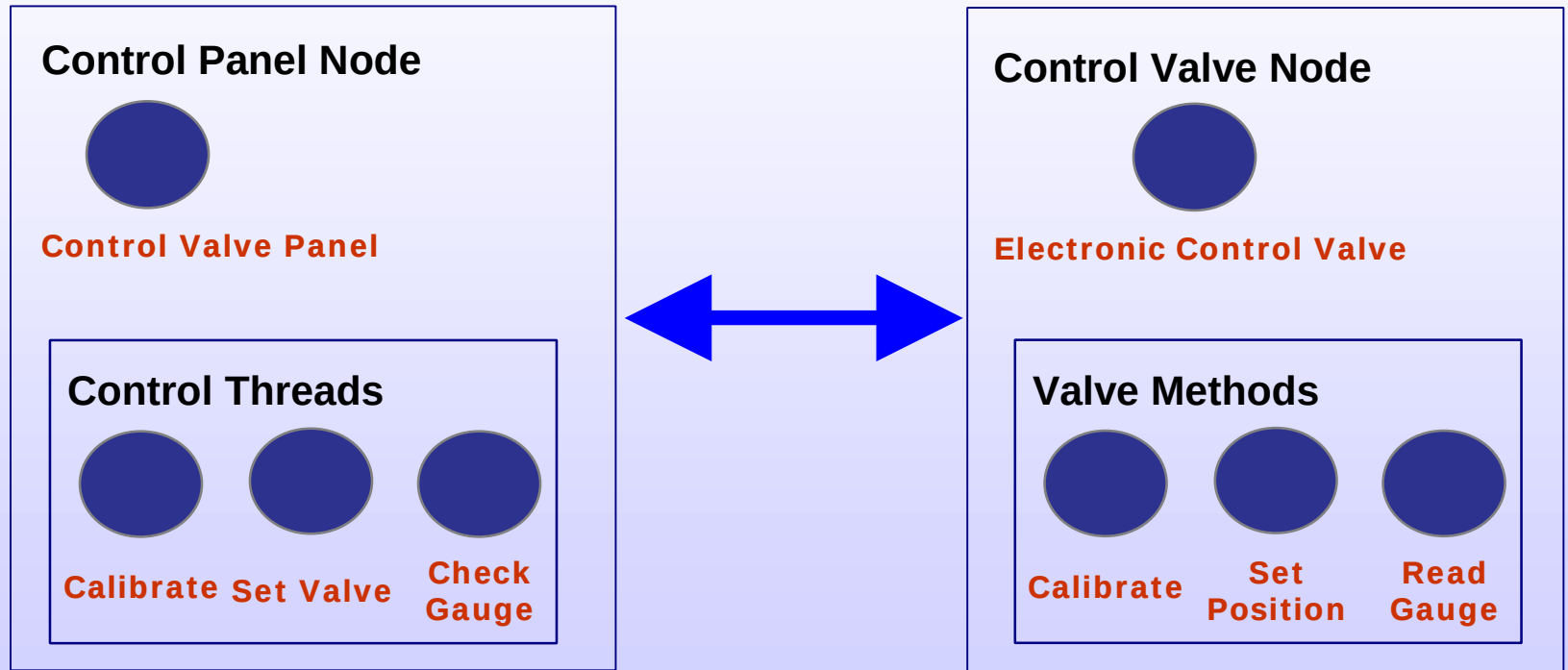
Object IDL

```
interface ControlValve  
{  
  
    boolean Calibrate ( in long Value );  
  
    short SetValvePosition ( in short Degrees );  
  
    long ReadGuage ( );  
  
};
```


Real-Time Design Considerations

Rate Monotonic Analysis
Priority Ceiling Protocol

Shared Resources
Limited OS Priorities



Calibrate

Lower Priority

Set Position

Medium Priority

Read Gauge

High Priority

Priority Mapping

Support for 32,767 priorities

Two priority mappings: `to_native()` and `to_CORBA()`

Implementation defined

Unique mapping for each node

Priority Mapping Example

Control Panel Priorities

	RTCORBA Priority	OS Priority
Calibrate	1	53
Set Valve	2	54
Check Gauge	3	54

Control Valve Priorities

	RTCORBA Priority Ceiling	OS Priority
Valve Gauge	3	54
Valve Position	2	54

Priority Mapping Example

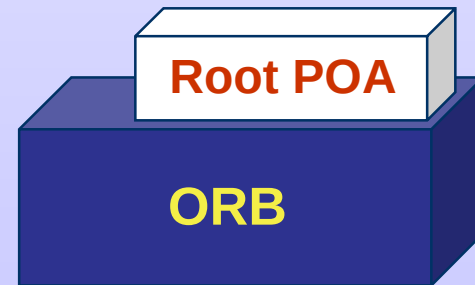
```
RTCORBA::NativePriority MappingTable[4] = { 0,  
                                           53,  
                                           54,  
                                           54 }; // Index 0 is unused.
```

```
class MappingClass : public RTCORBA::PriorityMapping {  
public:  
    CORBA::Boolean to_native(RTCORBA::Priority corba_priority,  
                             RTCORBA::NativePriority &native_priority)  
    {  
        if (corba_priority < 1 || corba_priority > (sizeof(MappingTable) / sizeof(int)))  
            return FALSE;  
        native_priority = MappingTable[corba_priority];  
        return TRUE;  
    }  
  
    CORBA::Boolean to_CORBA(RTCORBA::Priority native_priority,  
                             RTCORBA::NativePriority &corba_priority)  
    {  
        return FALSE;        // not used  
    }  
};
```

Valve Node Initialization

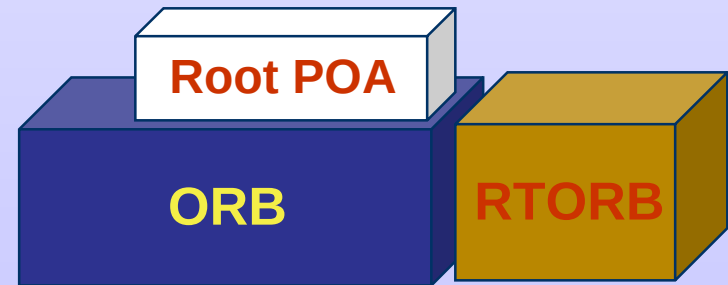
Begin with usual CORBA initialization

```
// C++  
CORBA::ORB_ptr orb;  
PortableServer::POA_ptr root_poa;  
  
void initValve()  
{  
  
    // initialize ORB  
    orb = CORBA::ORB_init(argc, argv, orb_id);  
  
    // obtain Root Object Adapter  
    CORBA::Object_var ref1 =  
        orb->resolve_initial_references("RootPOA");  
  
    root_poa = PortableServer::POA::_narrow(ref1);  
}
```



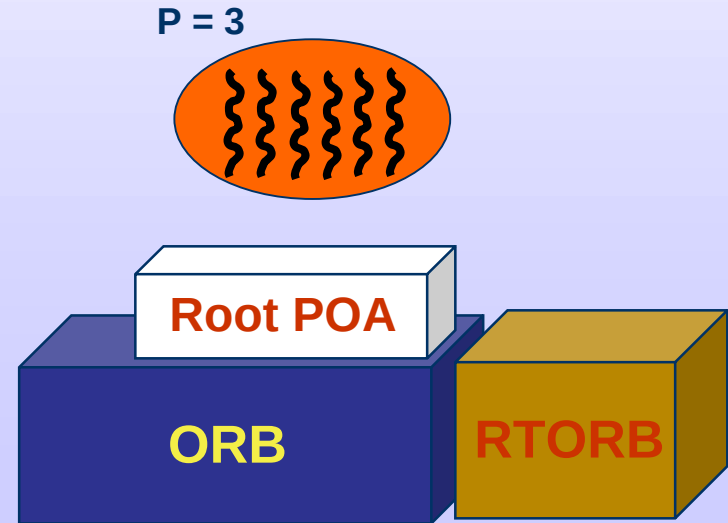
Access Real-Time CORBA API

```
RTCORBA::RTORB_ptr rt_orb;  
  
// access Real-Time extensions to ORB  
CORBA::Object_var ref2 =  
    orb->resolve_initial_references("RTORB");  
  
rt_orb = RTCORBA::RTORB::_narrow(ref2);
```



Create Threadpool

```
RTCORBA::ThreadpoolId valve_tpool;  
RTCORBA::ThreadpoolPolicy_var threadpool_policy;  
  
// create Threadpool for Valve  
valve_tpool = rt_orb->create_threadpool (  
    30000, // stacksize  
    6, // num static threads  
    0, // num dynamic threads  
    3, // default thread priority  
    FALSE, 0, 0 );  
  
// create Policy object for this Threadpool  
threadpool_policy =  
    rt_orb->create_threadpool_policy(valve_tpool);
```

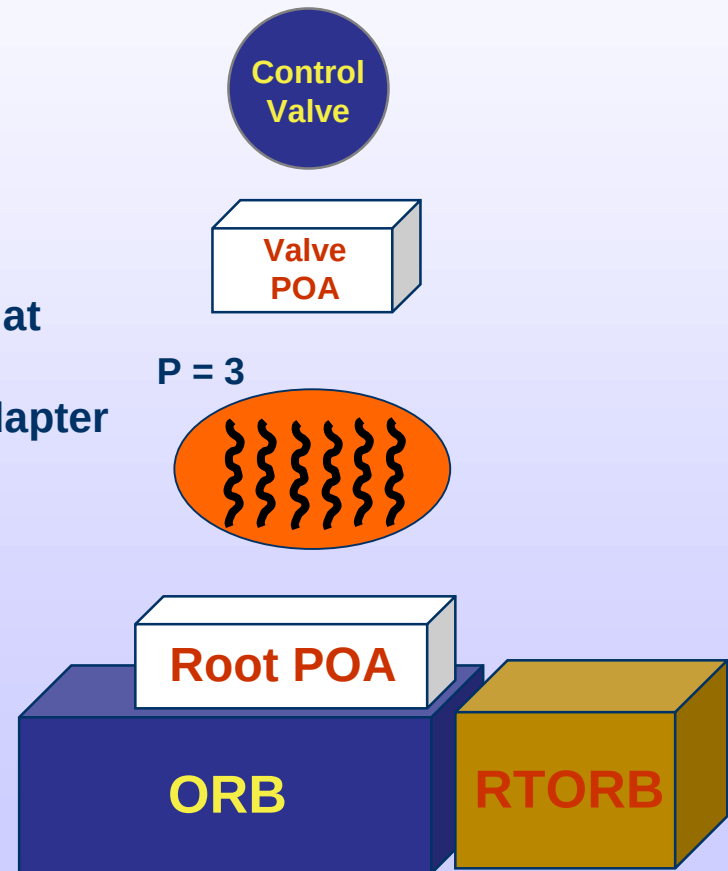


Prepare POA Configuration

```
RTCORBA::PriorityModelPolicy_var priority_model_policy;  
CORBA::PolicyList valve_policies(2);
```

```
// create Client Propagated Priority Model policy  
priority_model_policy =  
    rt_orb->create_priority_model_policy (  
        RTCORBA::CLIENT_PROPAGATED,  
        0 ); // priority handle non-RT clients at
```

```
// assemble policy list for initialising Valve object adapter  
valve_policies[0] = priority_model_policy;  
valve_policies[1] = threadpool_policy;
```



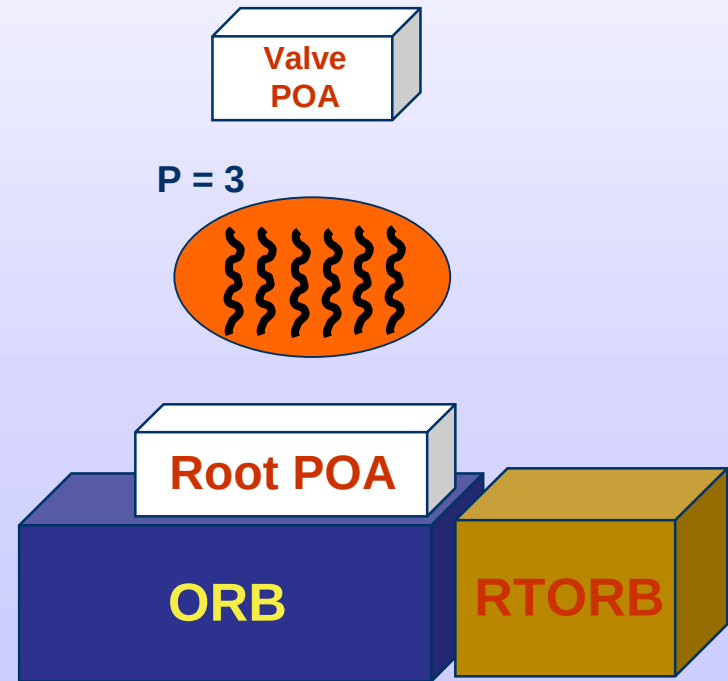
Create Valve POA

```
PortableServer::POA_ptr valve_poa;
```

```
// create Object Adapter
```

```
valve_poa =
```

```
    root_poa->create_POA("ValvePOA", 0,  
                        valve_policies);
```

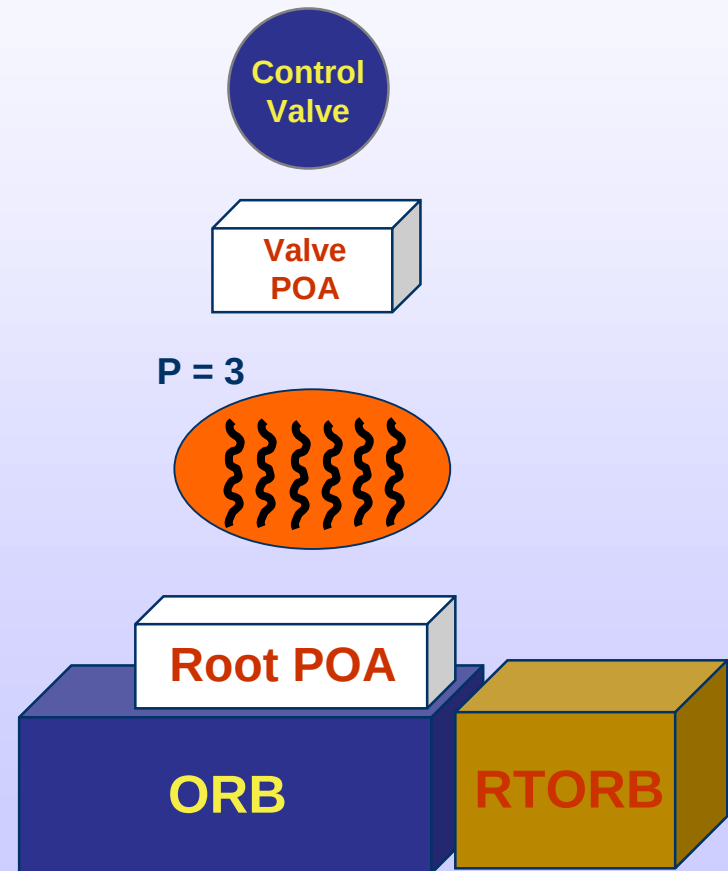


Create Valve Object

```
ValveServant * valve_servant;
```

```
// create Valve object  
valve_servant = new ValveServant();
```

```
// activate object  
valve_poa->activate_object(valve_servant);
```



Create RT Mutexes

```
RTCORBA::Mutex_ptr valve_mutex;  
RTCORBA::Mutex_ptr gauge_mutex;
```

```
ValveServant::ValveServant()
```

```
{
```

```
    // Create RTCORBA mutexes
```

```
    valve_mutex = rt_orb->create_mutex( );
```

```
    gauge_mutex = rt_orb->create_mutex( );
```

```
    // configure priority ceilings
```

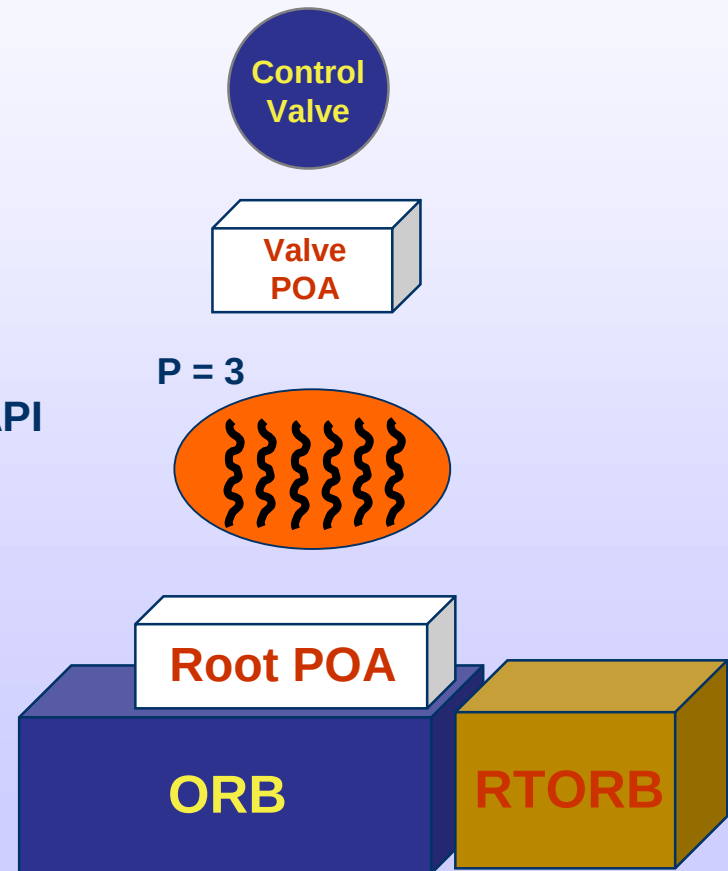
```
    // Note: Available priority protocols and API
```

```
    // are ORB implementation specific
```

```
    valve_mutex->set_ceiling (2);
```

```
    gauge_mutex->set_ceiling (3);
```

```
}
```



Object Implementation

CORBA::Boolean ValveServant::Calibrate(CORBA::Long Value)

```
{  
    gauge_mutex->lock();  
    ...  
    gauge_mutex->unlock();  
}
```

CORBA::Long ValveServant::SetValvePosition(CORBA::Short Degrees)

```
{  
    valve_mutex->lock();  
    ...  
    valve_mutex->unlock();  
}
```

CORBA::Long ValveServant::ReadGauge(

```
)  
{  
    gauge_mutex->lock();  
    ...  
    gauge_mutex->unlock();  
}
```

Control Node Initialization

Client node has no CORBA Objects, so it does not need to access the RootPOA

```
// C++
```

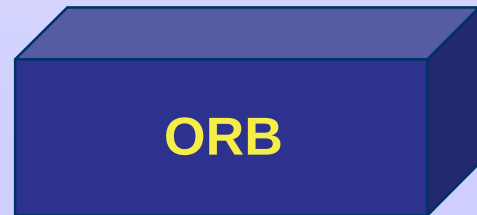
```
... // initialize mapping table
```

```
CORBA::ORB_ptr orb;
```

```
void initControlPanel()  
{
```

```
    // initialize ORB
```

```
    orb = CORBA::ORB_init(argc, argv, orb_id);
```



Obtain RTCORBA::Current

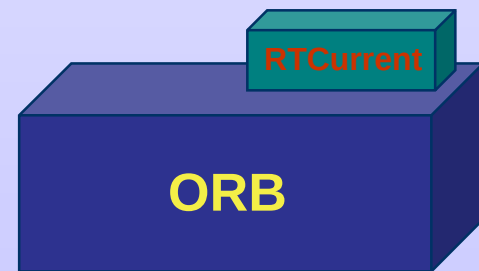
```
RTCORBA::Current_ptr rt_current;
```

```
// obtain the RTCurrent object
```

```
// Can obtain one instance and use it from all threads
```

```
CORBA::Object_var ref3 =  
    orb->resolve_initial_references("RTCurrent");
```

```
rt_current = RTCORBA::Current_narrow(ref3);
```



Create Client Threads

Client threads are implemented using native operating system methods

```
// Create client threads
```

```
thread_create(&thread_id, Calibrate);
```

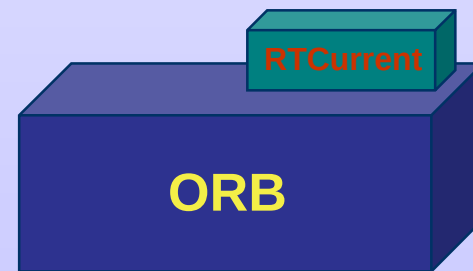
```
thread_create(&thread_id, SetValve);
```

```
thread_create(&thread_id, CheckGauge);
```

CheckGauge
Thread

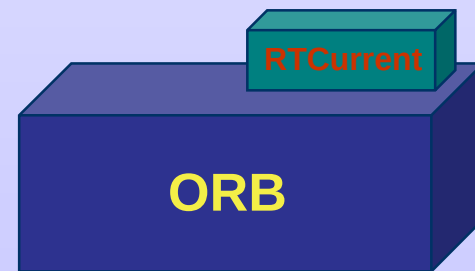
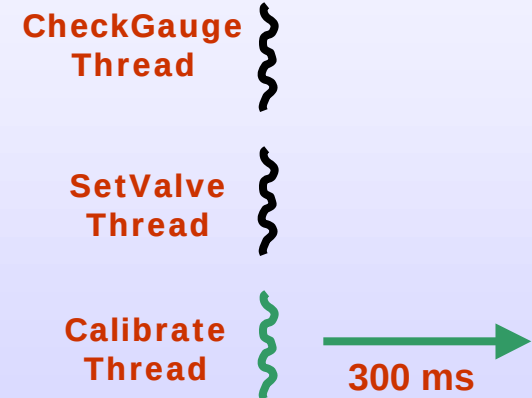
SetValve
Thread

Calibrate
Thread



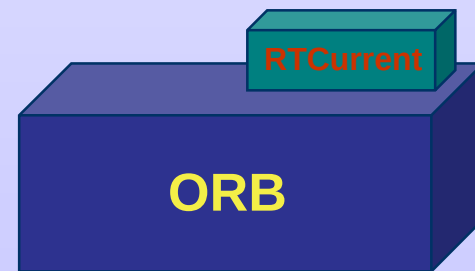
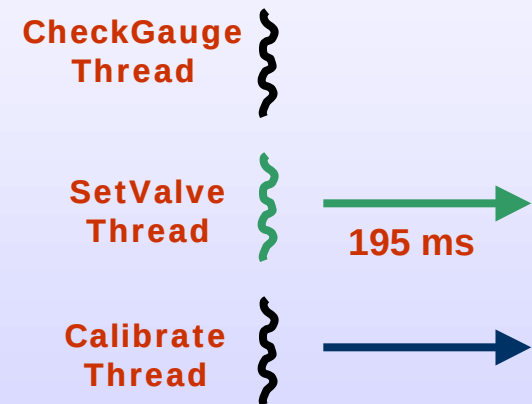
Calibrate Thread

```
void Calibrate( ) {  
  
    CORBA::Long Value;  
  
    // Get and bind object  
    ControlValve_var valve = // Obtain Valve reference  
  
    // Set the thread priority.  
    rt_current->the_priority = 1;  
  
    // Execute loop every 300 ms.  
    while (1) {  
  
        ... // Local code  
  
        valve->Calibrate(Value); // Call Valve object  
  
        ... // Local code  
  
        ... // Wait for period timer.  
    }  
}
```



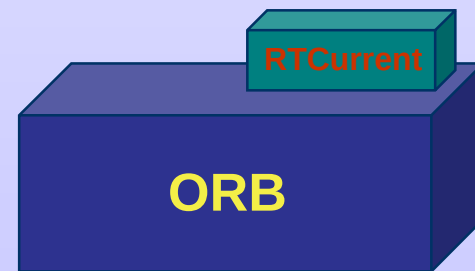
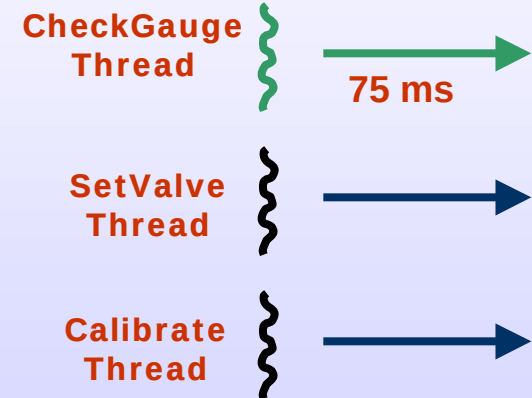
Set Valve Thread

```
void SetValve( ) {  
  
    CORBA::Short Degrees;  
  
    // Get and bind object  
    ControlValve_var valve = // Obtain Valve reference  
  
    // Set the thread priority.  
    rt_current->the_priority = 2;  
  
    // Execute loop every 195 ms.  
    while (1) {  
  
        ... // Local code  
  
        // Call Valve object  
        valve->SetValvePosition(Degrees);  
  
        ... // Local code  
  
        ... // Wait for period timer.  
    }  
}
```



Check Gauge Thread

```
void CheckGauge( ) {  
  
    CORBA::Long Value;  
  
    // Get and bind object  
    ControlValve_var obj1 = // Obtain Valve reference  
  
    // Set the thread priority.  
    rt_current->the_priority = 3;  
  
    // Execute loop every 75 ms.  
    while (1) {  
  
        ... // Local code  
  
        // Call Valve object  
        Value = valve->ReadGauge(Value);  
  
        ... // Local code  
  
        ... // Wait for period timer.  
    }  
}
```

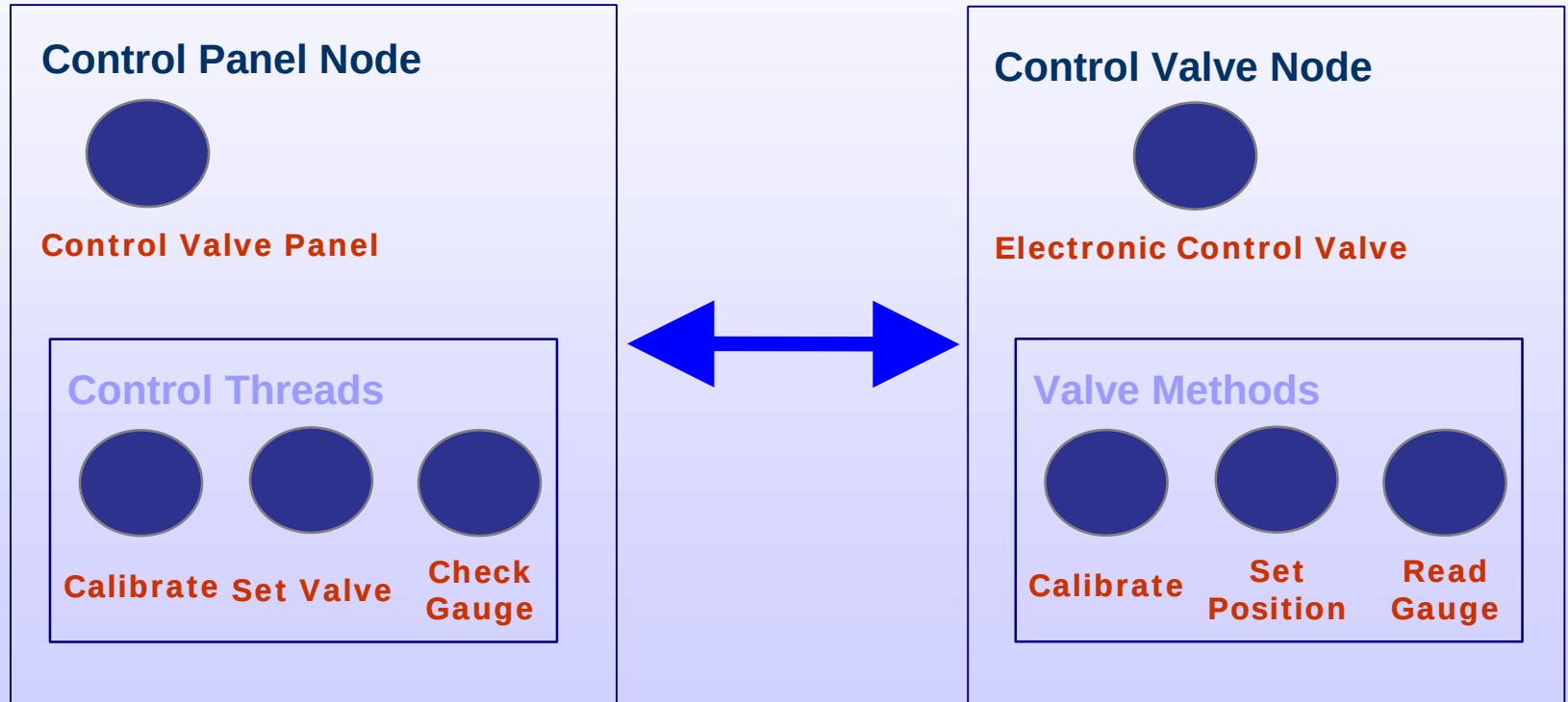


Agenda

- Example 1
 - Server Priority Model
 - Threadpool configuration
 - RTCORBA::Current
 - Protocol configuration
- Example 2
 - Client Priority Model
 - Laned Threadpools
 - Priority Banding
- Example 3
 - Priority Mapping
 - RTCORBA Mutex
- Example 4
 - Scheduling Service

Scheduling Service Example

Using same object model as previous example



Scheduling Service

Based on analytical tools

Performs all priority mapping

Handles all 32,767 priorities

Sets thread priorities

Enforces priority algorithm

Uses names instead of numbers

Change your scheduling, not your code

Priority Mapping

Control Panel Node Mapping

	RTCORBA Priority	OS Priority
Calibrate	1	53
Set Valve	2	54
Check Gauge	3	54

Control Valve Node Mapping

	RTCORBA Priority Ceiling	OS Priority
Valve Gauge	3	54
Valve Position	2	54

Same example mapping as before, but used as input to the Scheduling Service

However now there will be no mapping table in application code

Valve Node Initialisation

```
// No Scheduling Service
// C++
CORBA::ORB_ptr orb;
PortableServer::POA_ptr root_poa;
RTCORBA::RTORB_ptr rt_orb;

void initValve()
{

// initialize ORB
    orb = CORBA::ORB_init(argc, argv, orb_id);

// obtain Root Object Adapter
    CORBA::Object_var ref1 =
        orb->resolve_initial_references("RootPOA");

    root_poa = PortableServer::POA::_narrow(ref1);

// access Real-Time extensions to ORB
    CORBA::Object_var ref2 =
        orb->resolve_initial_references("RTORB");

    rt_orb = RTCORBA::RTORB::_narrow(ref2);
```



```
// Scheduling Service
// C++
CORBA::ORB_ptr orb;
PortableServer::POA_ptr root_poa;
RTCORBA::RTORB_ptr rt_orb;

void initValve()
{

// initialize ORB
    orb = CORBA::ORB_init(argc, argv, orb_id);

// obtain Root Object Adapter
    CORBA::Object_var ref1 =
        orb->resolve_initial_references("RootPOA");

    root_poa = PortableServer::POA::_narrow(ref1);

// access Real-Time extensions to ORB
    CORBA::Object_var ref2 =
        orb->resolve_initial_references("RTORB");

    rt_orb = RTCORBA::RTORB::_narrow(ref2);
```

Valve Node Initialisation

```
// No Scheduling Service
RTCORBA::ThreadpoolId valve_tpool;
RTCORBA::ThreadpoolPolicy_ptr threadpool_policy;

// create Threadpool for Valve
valve_tpool = rt_orb->create_threadpool(
    30000, // stacksize
    6, // num static threads
    0, // num dynamic threads
    3, // default thread priority
    FALSE, 0, 0);

threadpool_policy = rt_orb->
    create_threadpool_policy(valve_tpool);
```

// Scheduling Service

**// Depending on the implementation, threadpools
// may be may be configured by the Scheduling
// Service.**



Valve Node Initialisation

// No Scheduling Service

```
RTCORBA::PriorityModelPolicy_var  
    priority_model_policy;
```

```
CORBA::PolicyList valve_policies(2);
```

// create Server Priority Model policy

```
server_policy =  
    rt_orb->create_priority_model_policy  
        (RTCORBA::CLIENT_PROPAGATED,  
         0); // priority handle non-RT clients at
```

// assemble policy lists for initializing

// Valve object adapter

```
valve_policies[0] = priority_model_policy;  
valve_policies[1] = threadpool_policy;
```

// Scheduling Service

// server scheduler selects priority model

```
RTCosScheduling::ServerScheduler * sched;
```

```
sched = new RTCosScheduling::ServerScheduler;
```

Valve Node Initialisation

// No Scheduling Service

```
PortableServer::POA_ptr valve_poa;
```

// create Object Adapter

```
valve_poa =  
    root_poa->create_POA("ValvePOA", 0,  
                        valve_policies);
```

```
ValveServant_ptr valve_servant;
```

// create Valve object

```
valve_servant = new ValveServant();
```

// activate object

```
valve_poa->activate_object(valve_servant);
```

// Scheduling Service

```
PortableServer::POA_ptr valve_poa;
```

// create Object Adapter

```
valve_poa =  
    sched->create_POA(root_poa, "ValvePOA",  
                    nil, nil);
```

```
ValveServant_ptr valve_servant;
```

// create Valve object

```
valve_servant = new ValveServant();
```

// activate object

```
valve_poa->activate_object(valve_servant);
```

Valve Node Initialisation

// No Scheduling Service

```
RTCORBA::Mutex_ptr valve_mutex;  
RTCORBA::Mutex_ptr gauge_mutex;
```

ValveServant::ValveServant()

```
{  
    // Configure ORB for PCP type mutex is  
    // defined by the ORB implementation  
    ...  

```

```
    // Create RTCORBA mutexes  
    valve_mutex = rt_orb->create_mutex();  
    gauge_mutex = rt_orb->create_mutex();
```

```
    // Set PCP ceilings is defined  
    // by the ORB implementation  
    valve_mutex->set_ceiling(2);  
    gauge_mutex->set_ceiling(3);
```

```
}
```

// Scheduling Service

// RTCORBA::Mutex handled by Scheduling Service

ValveServant::ValveServant()

```
{
```

```
    // schedule_object requires Object_ptr or  
    // or Object_var. Note the reuse of the object  
    // pointer. This is due to using the DASPCP model.  
    sched->schedule_object ( ::_this,  
                             "Valve::Gauge");  
    sched->schedule_object ( ::_this,  
                             "Valve::Position");
```

```
}
```

Object Implementation

// No Scheduling Service

```
CORBA::Boolean
ValveServer::Calibrate(CORBA::Long Value) {
    gauge_mutex->lock();
    ...
    gauge_mutex->unlock();
}
```

```
CORBA::Long
ValveServant::SetValvePosition(CORBA::Short
Degrees) {
    valve_mutex->lock();
    ...
    valve_mutex->unlock();
}
```

```
CORBA::Long ValveServant::ReadGauge() {
    gauge_mutex->lock();
    ...
    gauge_mutex->unlock();
}
```

// Scheduling Service

```
CORBA::Boolean
ValveServer::Calibrate(CORBA::Long Value) {
    ...
}
```

```
CORBA::Long
ValveServant::SetValvePosition(CORBA::Short
Degrees) {
    ...
}
```

```
CORBA::Long ValveServant::ReadGauge() {
    ...
}
```



Control Node Initialization

```
// No Scheduling Service
// C++

... // initialize mapping table
```

```
CORBA::ORB_ptr orb;
```

```
void initControlPanel()
{
```

```
  // initialize ORB
```

```
    orb = CORBA::ORB_init(argc, argv, orb_id);
```

```
// Scheduling Service
// C++
```

```
// no mapping table needed
```

```
CORBA::ORB_ptr orb;
```

```
void initControlPanel()
{
```

```
  // initialize ORB
```

```
    orb = CORBA::ORB_init(argc, argv, orb_id);
```

Control Node Initialization

// No Scheduling Service

```
RTCORBA::RTORB_ptr rt_orb;
```

```
RTCORBA::Current_ptr rt_current;
```

// access Real-Time extensions to ORB

```
CORBA::Object_ptr ref2 =
```

```
    orb->resolve_initial_references("RTORB");
```

```
rt_orb = RTCORBA::RTORB::_narrow(ref2);
```

// obtain RTCurrent

```
CORBA::Object_ptr ref3 =
```

```
    orb->resolve_initial_references("RTCurrent");
```

```
rt_current = RTCORBA::Current::_narrow(ref3);
```

// Create client threads

```
thread_create(&thread_id, Calibrate);
```

```
thread_create(&thread_id, SetValve);
```

```
thread_create(&thread_id, CheckGauge);
```

// Scheduling Service

// no access to RTCurrent needed

// Create client threads

```
thread_create(&thread_id, Calibrate);
```

```
thread_create(&thread_id, SetValve);
```

```
thread_create(&thread_id, CheckGauge);
```



Calibrate Thread

// No Scheduling Service

```
void Calibrate() {
```

```
    CORBA::Long Value;
```

```
    // Get and bind object
```

```
    ControlValve_var valve = // obtain Valve reference
```

```
    // Set the thread priority.
```

```
    rt_current->the_priority = 1;
```

```
    // Execute loop every 300 ms.
```

```
    while (1) {
```

```
        ... // Local code
```

```
        valve->Calibrate(Value); // Call Valve
```

```
        ... // Local code
```

```
        ... // Wait for period timer.
```

```
    }
```

```
}
```

// Scheduling Service

```
void Calibrate() {
```

```
    CORBA::Long Value;
```

```
    // Get and bind object
```

```
    ControlValve_var valve = // obtain Valve reference
```

```
    // Set the thread priority.
```

```
    sched->schedule_activity("Calibrate");
```

```
    // Execute loop every 300 ms.
```

```
    while (1) {
```

```
        ... // Local code
```

```
        valve->Calibrate(Value); // Call Valve
```

```
        ... // Local code
```

```
        ... // Wait for period timer.
```

```
    }
```

```
}
```

Set Valve Thread

```
// No Scheduling Service
void SetValve() {
    CORBA::Short Degrees;
```

```
// Get and bind object
ControlValve_var obj1 = // obtain Valve reference
```

```
// Set the thread priority.
rt_current->the_priority = 2;
```

```
// Execute loop every 195 ms.
while (1) {
```

```
    ... // Local code
```

```
    valve->SetValvePosition(Degrees);
```

```
    ... // Local code
```

```
    ... // Wait for period timer.
```

```
    }
}
```

```
// Scheduling Service
void SetValve() {
    CORBA::Short Degrees;
```

```
// Get and bind object
ControlValve_var valve = // obtain Valve reference
```

```
// Set the thread priority.
sched->schedule_activity("SetValve");
```

```
// Execute loop every 195 ms.
while (1) {
```

```
    ... // Local code
```

```
    valve->SetValvePosition(Degrees);
```

```
    ... // Local code
```

```
    ... // Wait for period timer.
```

```
    }
}
```


Check Gauge Thread

// No Scheduling Service

```
void CheckGauge() {
```

```
    CORBA::Long Value;
```

// Get and bind object

```
    ControlValve_var valve = // Something
```

```
// Set the thread priority.
```

```
rt_current->the_priority = 3;
```

// Execute loop every 75 ms.

```
while (1) {
```

```
    ... // Local code
```

```
    Value = valve->ReadGauge(Value);
```

```
    ... // Local code
```

```
    ... // Wait for period timer.
```

```
}
```

```
}
```

// Scheduling Service

```
void CheckGauge() {
```

```
    CORBA::Long Value;
```

// Get and bind object

```
    ControlValve_var valve = // obtain Valve reference
```

```
// Set the thread priority.
```

```
sched->schedule_activity("CheckGauge");
```

// Execute loop every 75 ms.

```
while (1) {
```

```
    ... // Local code
```

```
    Value = valve->ReadGauge(Value);
```

```
    ... // Local code
```

```
    ... // Wait for period timer.
```

```
}
```

```
}
```