Design and Implementation Issues in the Dynamic Scheduling Real-Time CORBA 2.0 Specification

Yamuna Krishnamurthy and Irfan Pyarali

OOMWorks, LLC

{yamuna,irfan}@oomworks.com

Christopher D. Gill
Washington University, St. Louis
cdgill@cse.wustl.edu

Victor Fay Wolfe
University of Rhode Island
wolfe@cs.uri.edu

OOMWORKS

Washington University

University of Rhode Island

Sunday, July 06, 2003

Comparison of Static and Dynamic Scheduling

Static Scheduling	Dynamic Scheduling
Tasks in the system are known	Tasks in the system may or may not be known
Execution time and QoS requirements of tasks are fixed	Execution time and QoS requirements of tasks change dynamically
A priori scheduling analysis possible	A priori scheduling analysis may not be possible

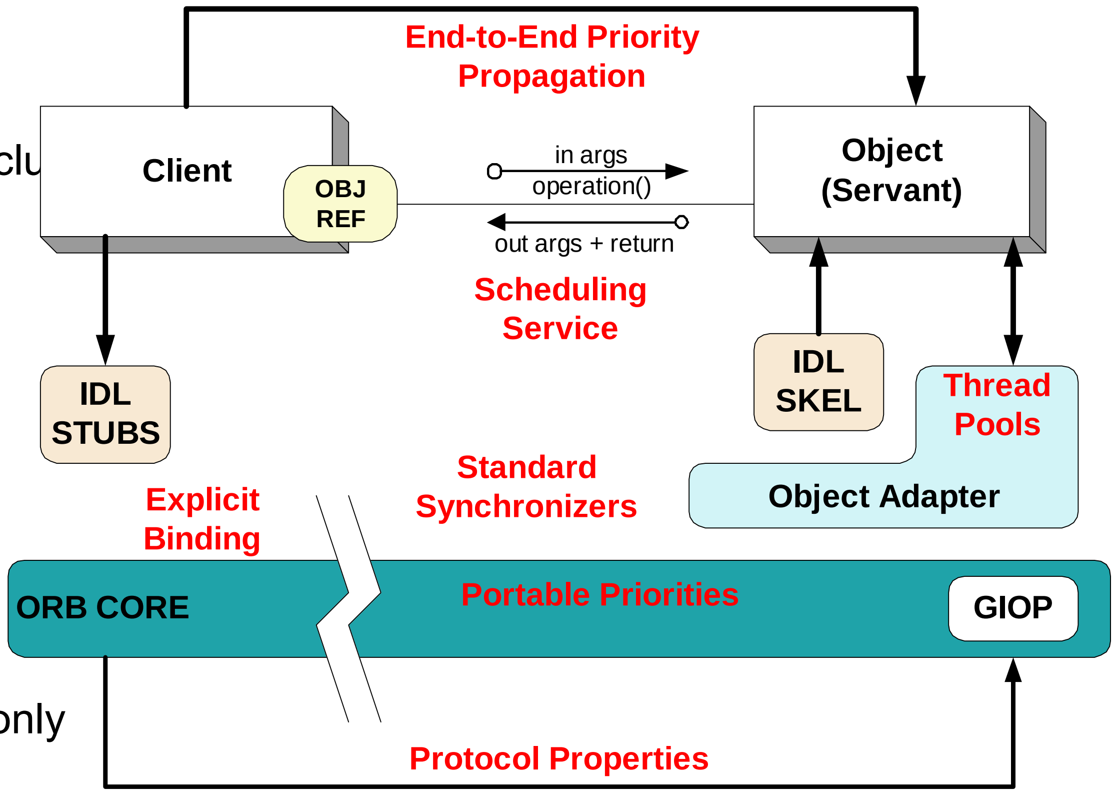
Real-Time CORBA (RTCORBA) 1.0 Overview

- RTCORBA 1.0 adds QoS control to regular CORBA to improve the application *predictability*
  - Bounding priority inversions
  - Managing resources end-to-end
- Policies & mechanisms for resource configuration/control in RTCORBA inclu

  Processor Resources
  - Thread pools
  - Priority models
  - Portable priorities

  Communication Resources
  - Protocol policies
  - Explicit binding

  Memory Resources
  - Request buffering



- These capabilities however address only static real-time systems

Challenges: Dynamic Scheduling in Distributed Systems

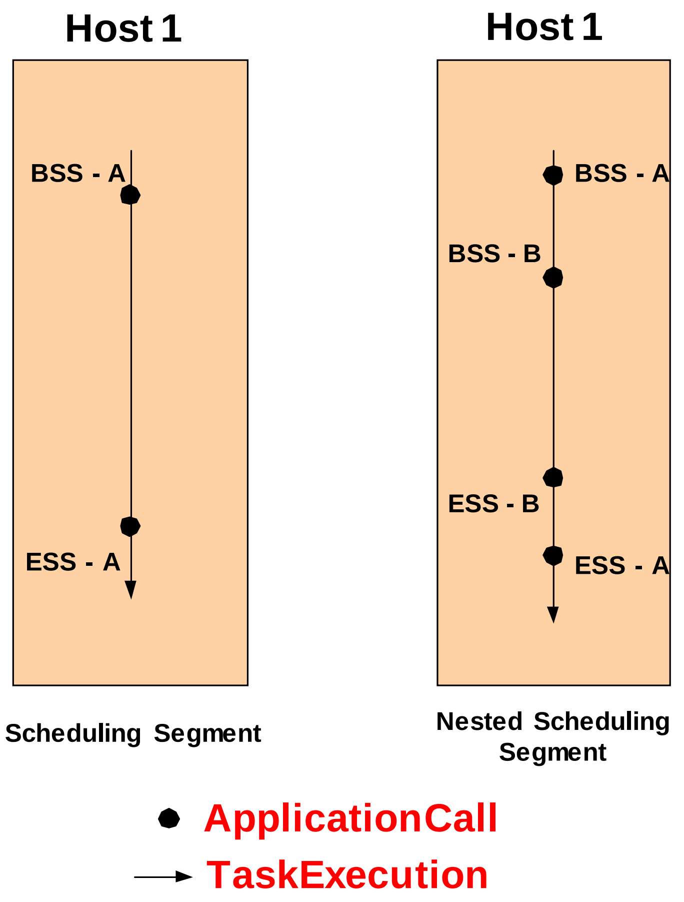
- Tasks span multiple heterogeneous hosts
- Tasks and hosts they span are not known a priori
- Propagate scheduling requirements of a task across hosts it spans
- Allow custom scheduling discipline plug-ins
- Fixed priorities for tasks insufficient for scheduling analysis
- Cancel a distributed task and reschedule accordingly on hosts it spans
- Local schedulers need to reschedule when tasks return from a remote invocation
- Collaboration of schedulers in the system to ensure global scheduling

Solution: RTCORBA 2.0

- *Scheduling Segment* is a sequence of code with specific scheduling requirements
- *Distributable Thread (DT)* a programming model abstraction for a distributed task
  - Loci of execution spans multiple nodes and scheduling segments
  - Identified by a unique system wide id GUID (Globally Unique Id)
  - DT scheduling information passed through GIOP Service Contexts across hosts it spans
- *Pluggable Scheduler* facilitates the use of custom scheduling disciplines
- *Scheduling Points* to interact with the scheduler at application and ORB level

Scheduling Segments

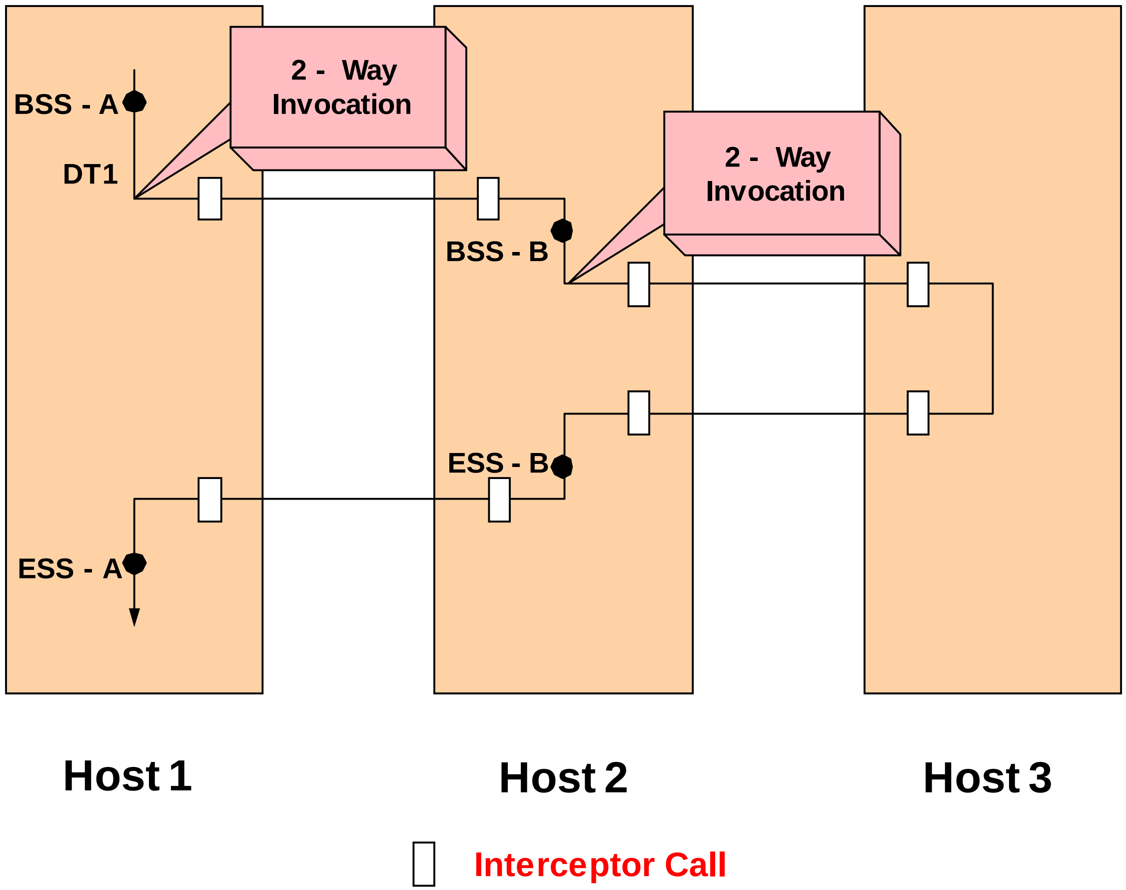
BSS -*begin_scheduling_segment*
ESS -*end_scheduling_segment*



- `RTScheduling::Current` has methods to begin, end and update scheduling segments and spawn a new DT
- Each segment's scheduling characteristics is defined by its *Scheduling Parameters*
- Nested scheduling segments allow association of different scheduling parameters
- Begin and End of a scheduling segment should be on the same host

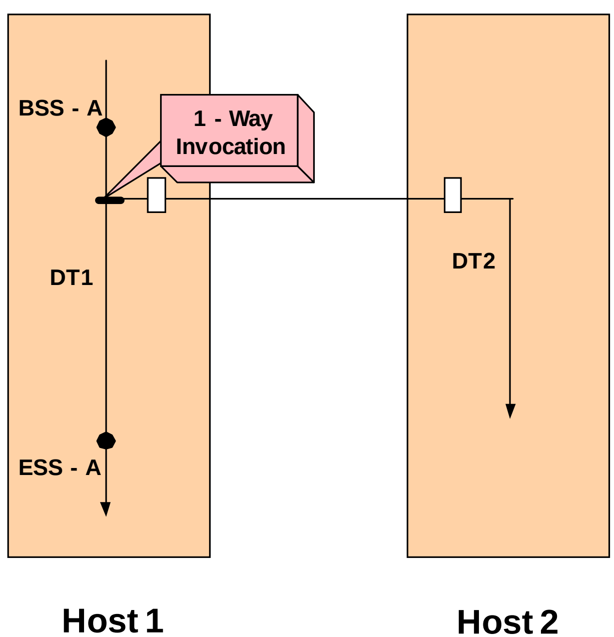
Distributable Threads May Span Endsystems

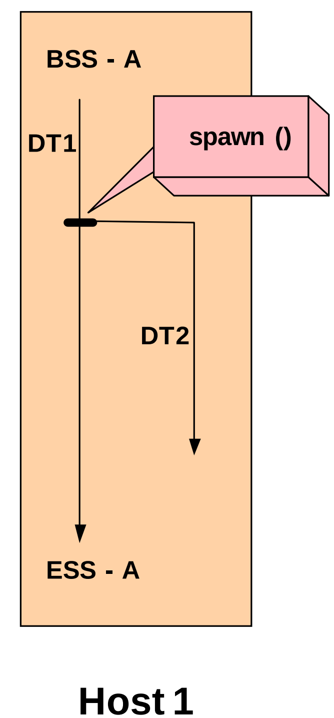
- Distributable Thread traversing multiple hosts with two-way invocations



Distributable Thread Creation

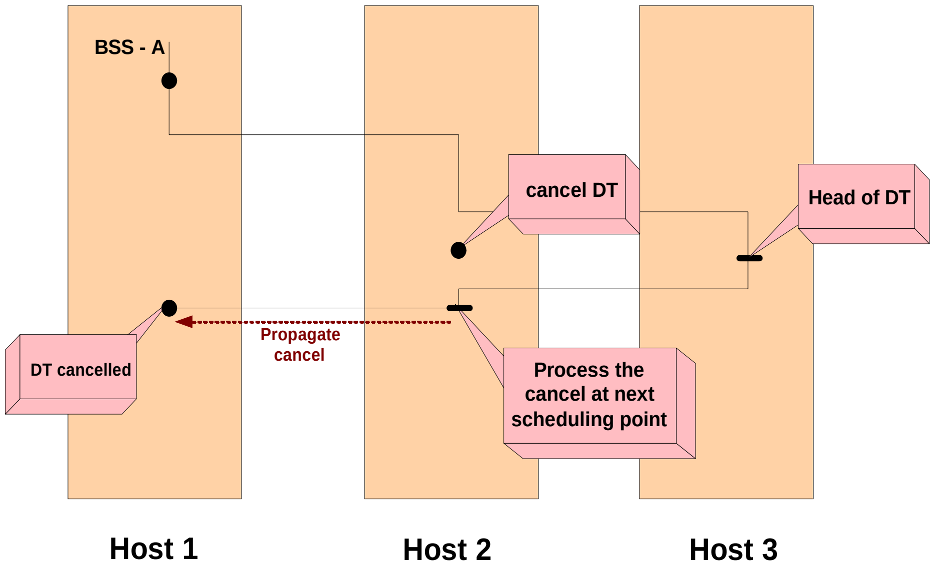
- Distributable Thread making One-Way invocation
  - New DT created implicitly
  - With implicit scheduling parameters
- Distributable Thread Spawn
  - New native thread created
  - New DT created
  - With implicit scheduling parameters



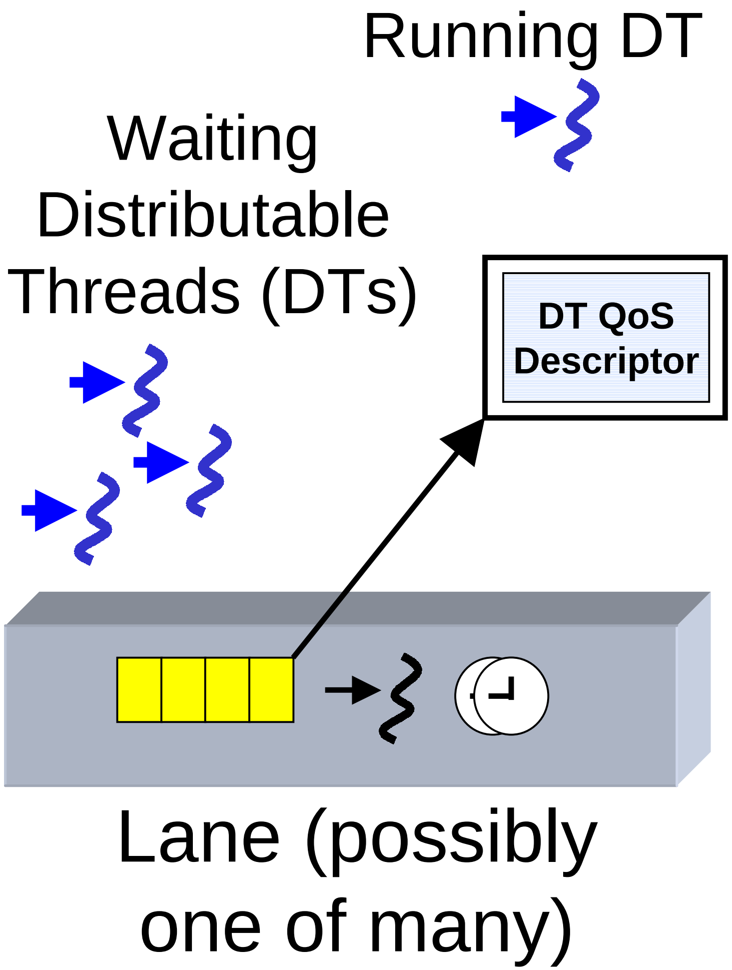


Distributable Thread Cancellation

- RTScheduling::DistributableThread::cancel () cancels the DT
- This operation raises the CORBA::Thread_Cancelled exception at the next scheduling point
- The exception is propagated to the start of the DT
- The exception is not propagated to the head of the DT
- DT can be cancelled on any host that it currently spans

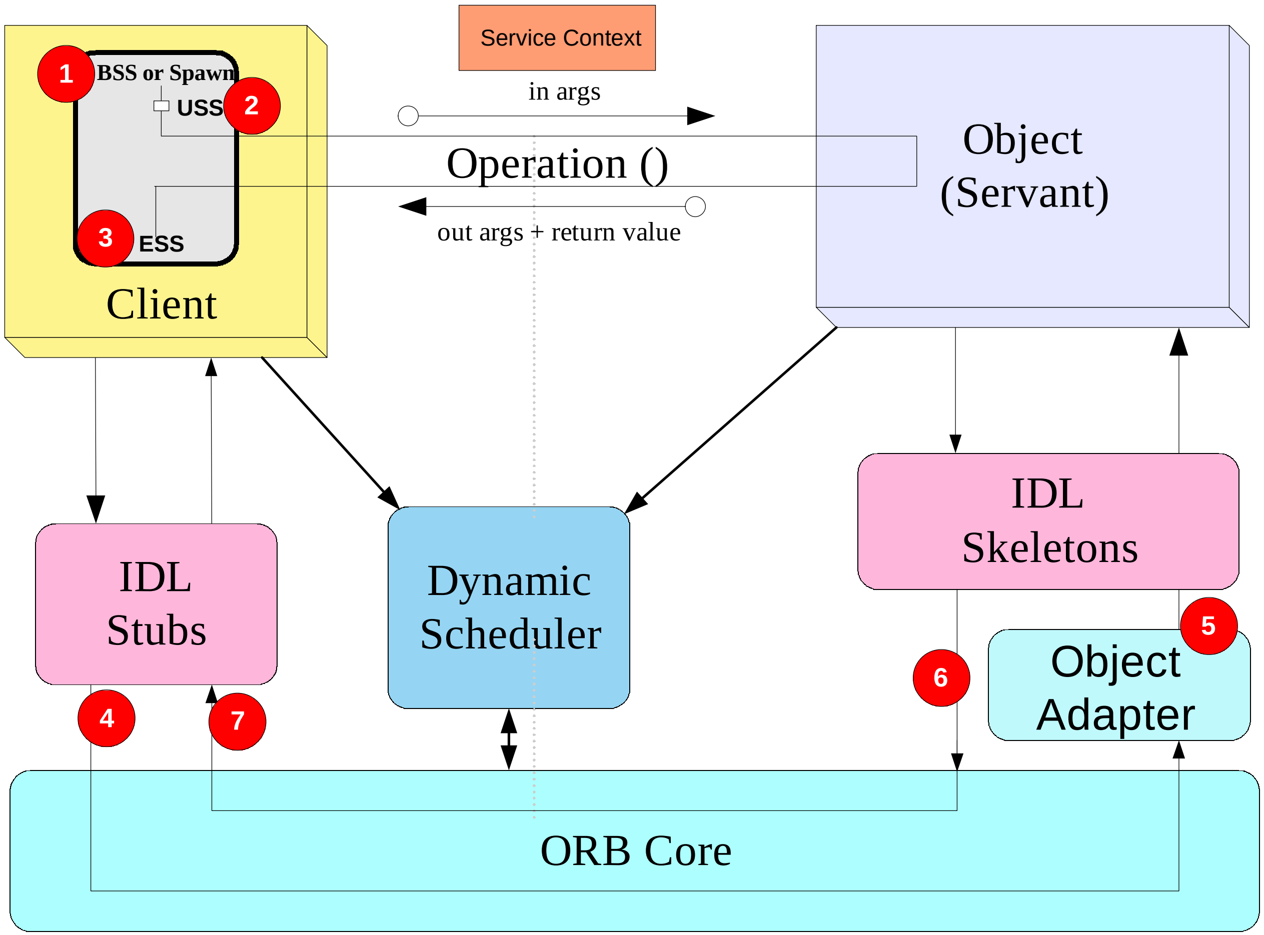


Pluggable Scheduling



- Scheduling/Dispatching
  - Enforce predictable behavior in DRE systems
- Alternative disciplines
  - RMS, MUF, EDF, LLF
  - Custom scheduling disciplines dictated by system requirements
  - Queried via resolve_initial_references
    - “RTScheduler”
- RTScheduling::Scheduler interface
  - From which implementations are derived
  - Interactions with application and ORB
- Scheduler Managers
  - Install/manage schedulers in the ORB

Scheduling Points

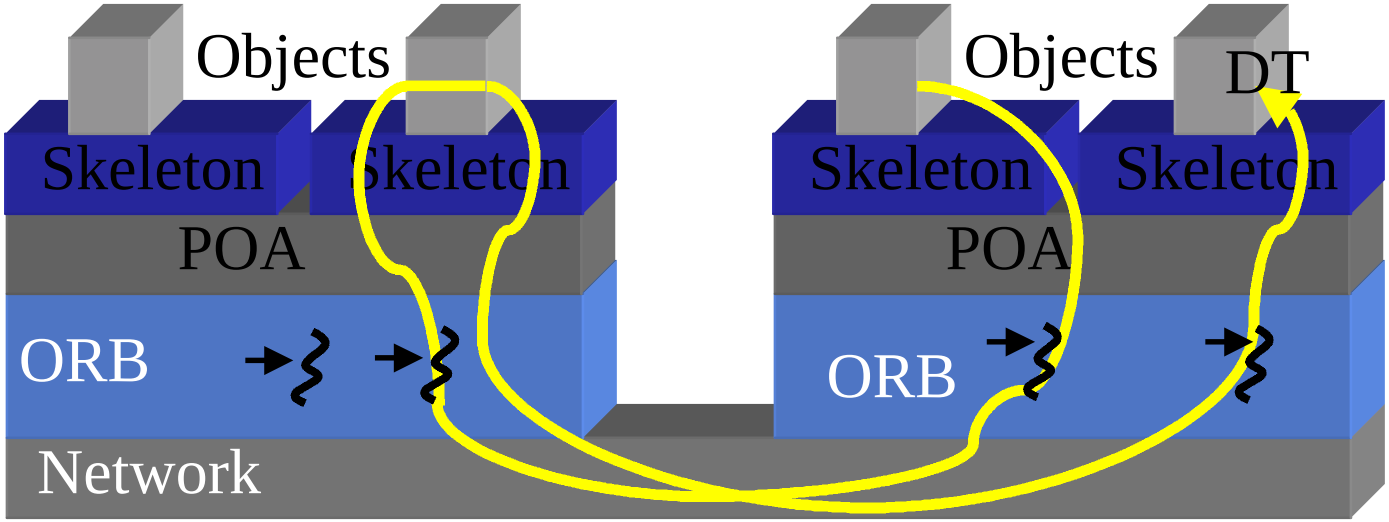


1. Begin scheduling segment or spawn
2. Update scheduling segment
3. End scheduling segment
4. Send request
5. Receive request
6. Send reply
7. Receive reply

Scheduling Points, Continued

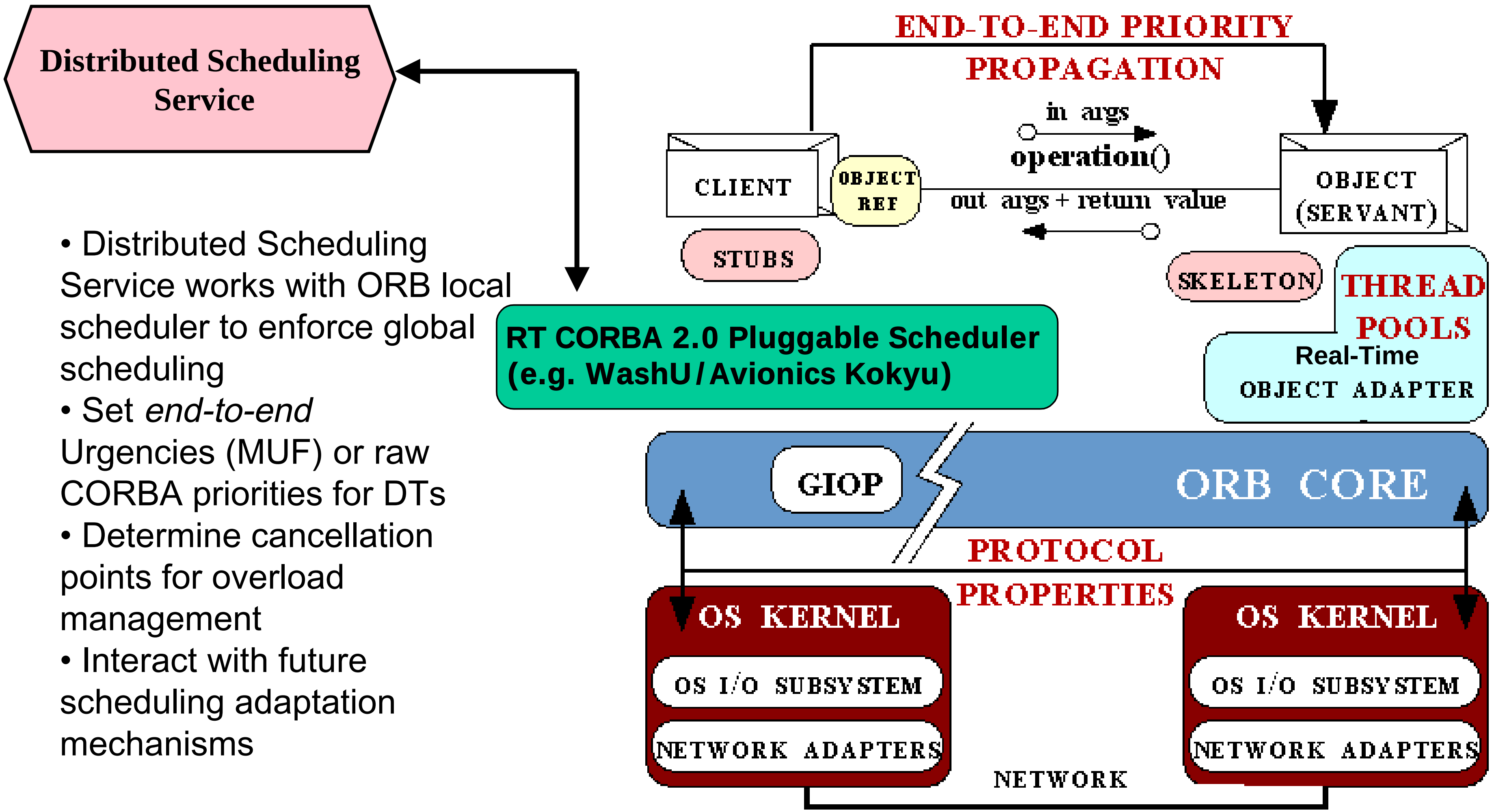
User / ORB	*Scheduler Upcall*
`Current::spawn()`	`begin_new_scheduling_segment()`
`Current::begin_scheduling_segment()` [*when creating DTs*]	`begin_new_scheduling_segment()`
`Current::begin_scheduling_segment()` [*when creating nested segments*]	`begin_nested_scheduling_segment()`
`Current::update_scheduling_segment()`	`update_scheduling_segment()`
`Current::end_scheduling_segment()` [*when ending scheduling nested segments*]	`end_nested_scheduling_segment()`
`Current::end_scheduling_segment()` [*when destroying DTs*]	`end_scheduling_segment()`
`DistributableThread::cancel()`	`cancel()`
`Outgoing request`	`send_request()`
`Incoming request`	`receive_request()`
`Outgoing reply`	`send_reply()`
`Incoming reply`	`receive_reply()`

Mapping Distributable Threads to OS Threads



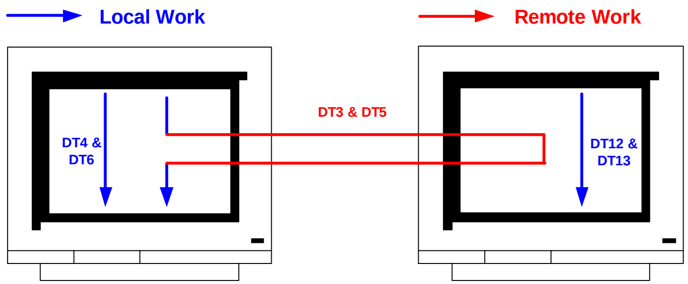
- DT identity (GUID) vs. OS thread identity (tid)
  - Mapping may be many-to-many and may evolve over time
  - Need GUID aware concurrency and synchronization mechanisms
    - ORB-level schedulers, but also lower-level mutexes, TSS, managers
  - DT cancellation semantics depends on ORB-level concurrency
    - *I.e.*, reactive vs. cooperative vs. preemptive

Dynamic Distributed Scheduling Service



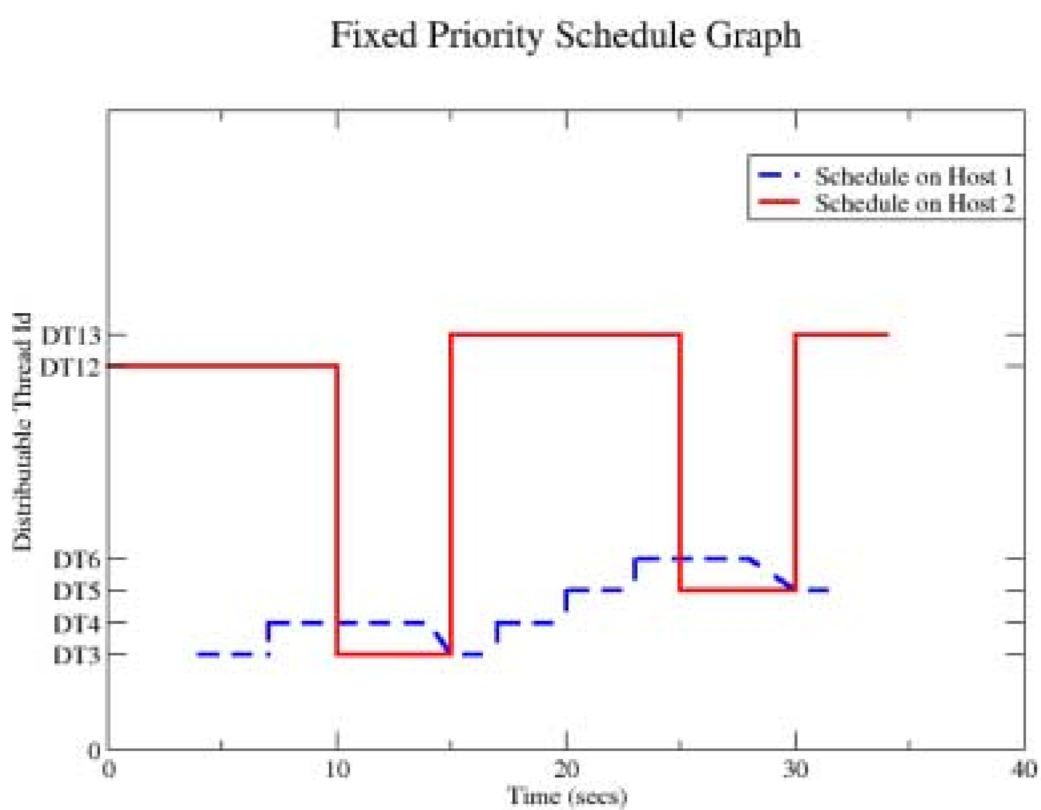
- Distributed Scheduling Service works with ORB local scheduler to enforce global scheduling
- Set *end-to-end* Urgencies (MUF) or raw CORBA priorities for DTs
- Determine cancellation points for overload management
- Interact with future scheduling adaptation mechanisms

Fixed Priority Scheduler



DT GUID on Host2	Start Time T (secs)	Importance	Execution Time (secs)	
			Local	Dist
12	0	15	10	-
13	5	2	15	-

DT GUID on Host1	Start Time T (secs)	Importance	Execution Time (secs)	
			Local	Dist
3	4	10	5	5
4	7	8	10	-
5	4	5	5	5
6	7	4	5	-



Shortcomings of the RTCORBA 2.0 specification

- Redundant operations for querying DT id
  - `Current::get_current_id()` is redundant
  - Id accessed with readonly attribute `Id`
- Insufficient operation parameters for `Current::spawn()`
  - Name, scheduling parameters
  - `CORBA::VoidData data` required by `ThreadAction::do()` method called by spawn
- Insufficient operations in `RTScheduling::Scheduler` interface for user and ORB interaction with scheduler

User / ORB	*Scheduler*
Current::spawn()	begin_new_scheduling_segment()
Current::begin_scheduling_segment()	begin_new_scheduling_segment()
Current::begin_scheduling_segment()	begin_nested_scheduling_segment()
Current::update_scheduling_segment()	update_scheduling_segment()
Current::end_scheduling_segment()	end_nested_scheduling_segment()
Current::end_scheduling_segment()	end_scheduling_segment()
DistributableThread::cancel()	cancel()
Outgoing request	send_request()
Incoming request	receive_request()
Outgoing reply	send_reply()
Incoming reply	receive_reply()

Future Work

- Global Scheduling
  - System wide scheduling algorithm
  - Interacting schedulers
- Multi-Level scheduling
  - Meta scheduling
  - Global optimality

Conclusions

- RTCORBA 2.0 provides mechanisms for dynamic scheduling in DRE systems
- RTCORBA 2.0 provides interfaces for:
  - User interaction with scheduler to schedule the task
  - ORB interaction with scheduler when sending/receiving requests
  - Scheduler developer's implementation and plug-in of specific scheduling disciplines
- Implementation experience has shown some shortcomings in the RTCORBA 2.0 standard, particularly for truly pluggable schedulers.
- RTCORBA 2.0 implementation in TAO
  - http://deuce.doc.wustl.edu/Download.html
- Contacts
  - Yamuna Krishnamurthy <yamuna@oomworks.com>
  - Chris Gill <cdgill@cse.wustl.edu>
  - Vic Fay-Wolfe <wolfe@cs.uri.edu>