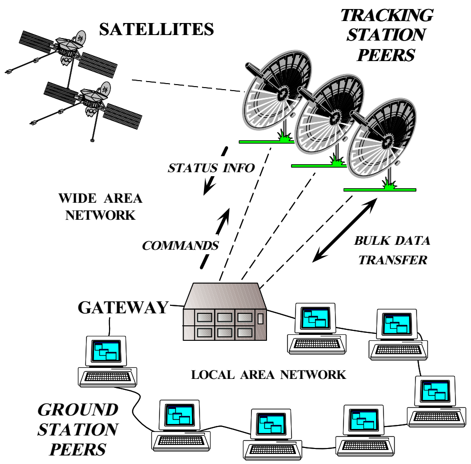
Principles and Patterns of High-performance, Real-time Object Request Brokers

Douglas C. Schmidt
schmidt@cs.wustl.edu

Washington University, St. Louis
http://www.cs.wustl.edu/∼schmidt/TAO.html

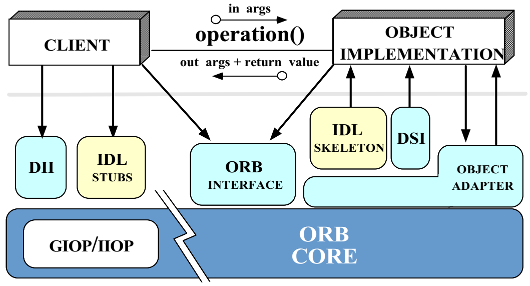
– Typeset by FoilTEX –

Motivation



- Many applications require QoS guarantees
  – *e.g.*, telecom, avionics, WWW
- Existing middleware doesn't support QoS effectively
  – *e.g.*, CORBA, DCOM, DCE
- Solutions must be *integrated*
  – *Vertically* and *horizontally*

Candidate Solution: CORBA



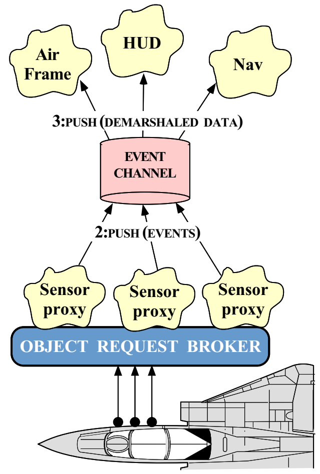
http://www.cs.wustl.edu/∼schmidt/corba.html

- **Goals of CORBA**
  – Simplify distribution by automating
    ∗ Object location and activation
    ∗ Parameter marshaling
    ∗ Demultiplexing
    ∗ Error handling
  – Provide foundation for higher-level services
- **Limitations of CORBA**
  – Poor performance
  – Lack of QoS features

Components in TAO

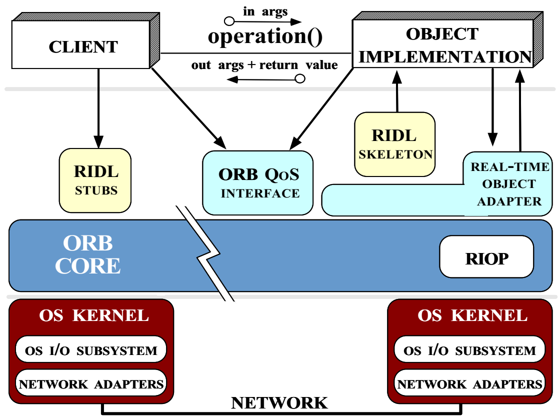
- IDL Compiler
  - Based on Sun IDL front-end + our back-end
- RIOP Protocol Engine
  - Optimized version of Sun's GIOP/IIOP protocol engine with real-time enhancements
- ACE ORB Core
  - Multi-threaded ORB run-time system based on ACE
- Real-time Object Adapter
  - Demultiplex, schedule, and dispatch client requests in real-time
- Object Services
  - Real-time Event Channels and Multimedia Streaming Service

Applying ORBs to Real-time Avionics



- **Domain Challenges**
  - Periodic hard real-time deadlines
  - COTS infrastructure
  - Open systems
- **Related work**
  - Deng, Liu, and J. Sun '96
  - Gopalakrishnan and Parulkar '96
  - Wolfe et al. '96

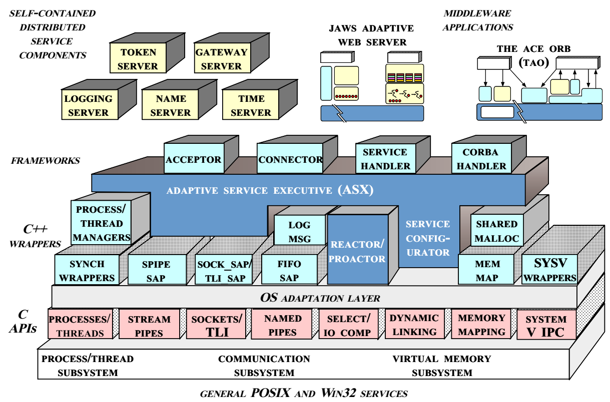
The ACE ORB (TAO)



http://www.cs.wustl.edu/∼schmidt/TAO.html

- **TAO Overview**
  - A high-performance, real-time ORB
    - ∗ Networking and avionics focus
  - Leverages the ACE framework
    - ∗ Ported to VxWorks, POSIX, and Win32
- **Related work**
  - QuO at BBN
  - ARMADA at U. Mich.

The ADAPTIVE Communication Environment (ACE)



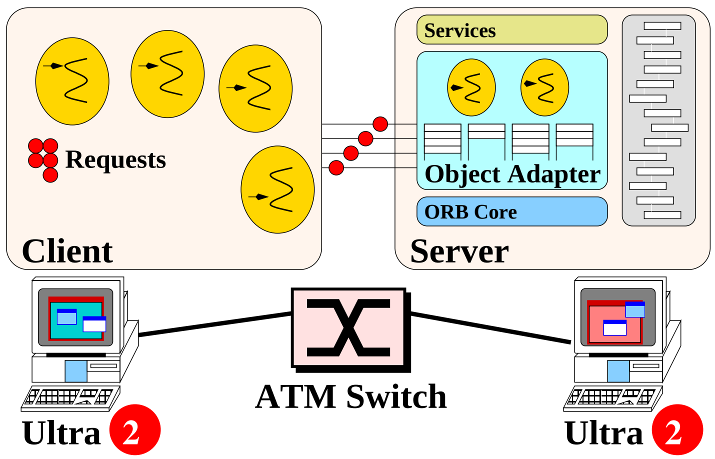
http://www.cs.wustl.edu/∼schmidt/ACE.html

- **ACE Overview**
  - A concurrent OO networking framewor
  - Widely used in industry
  - Available in C++ an Java
  - Ported to VxWorks, POSIX, and Win32
- **Related work**
  - x-Kernel
  - System V STREAMS
  - Conduit+

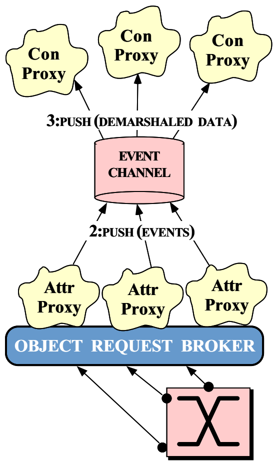
Research Objectives

- Can CORBA be used for performance-sensitive applications (*e.g.*, telecommunications, satellite surveillance, medical imaging, avionics) on high-speed networks?
  - Goal is to determine empirically
- Identify features and architectural patterns needed for real-time ORBs
  - Both hard real-time and statistical real-time
- Develop optimizations required to build high-performance ORBs
  - *e.g.*, Gigabit bandwidth and ∼10 microsecond latency
- Determine changes needed to CORBA specification
  - *e.g.*, APIs for defining end-to-end QoS requirements

Experimental Setup for CORBA/ATM Testbed

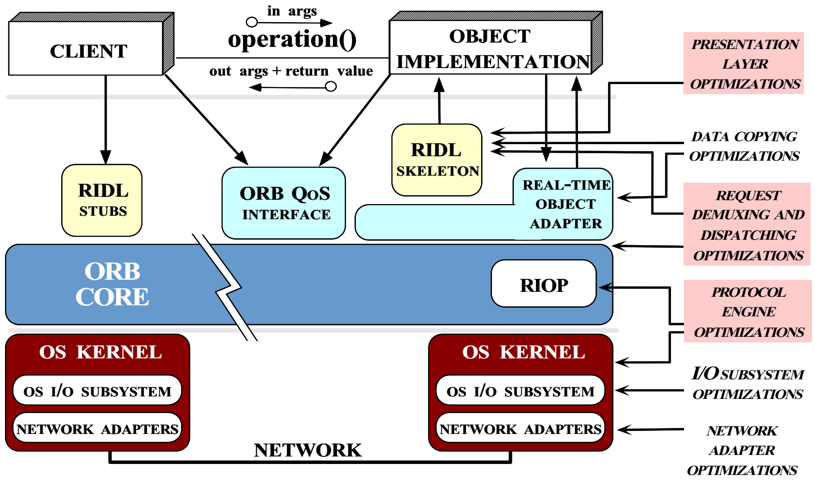


Applying ORBs to Real-time Network Management

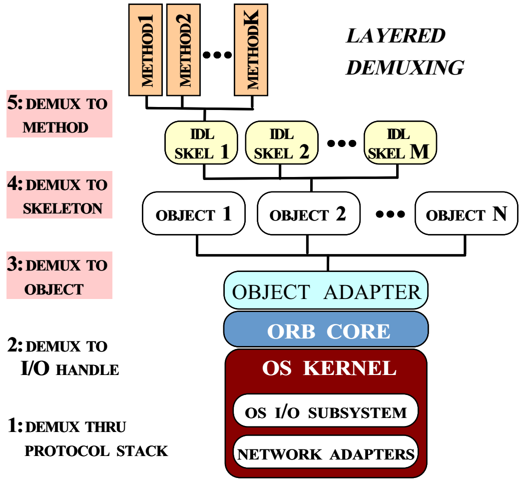


- **Domain Challenges**
  - Periodic statistical real-time deadlines
  - COTS infrastructure
  - Open systems
- **Related work**
  - Deng, Liu, and J. Sun '96
  - Gopalakrishnan and Parulkar '96
  - Wolfe et al. '96

Real-time Features and Optimizations in TAO

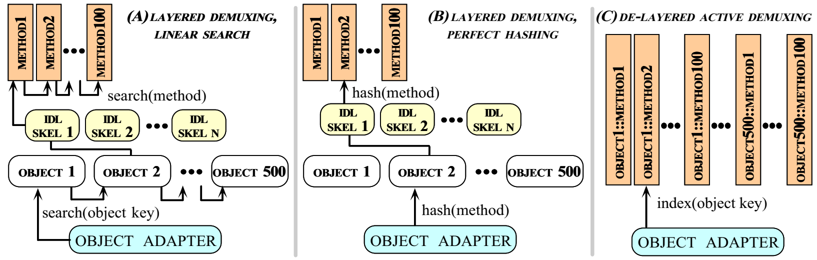


Problem: Reducing Demultiplexing Latency



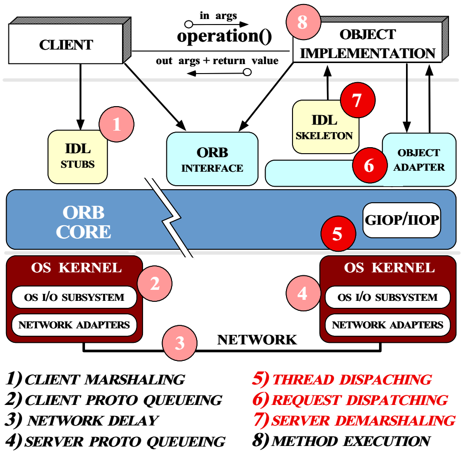
- **Design Challenges**
  - Minimize demuxing layers
  - Provide $O(1)$ operation demuxing
  - Avoid priority inversions
  - Remain CORBA-compliant

Demultiplexing Performance Experiments



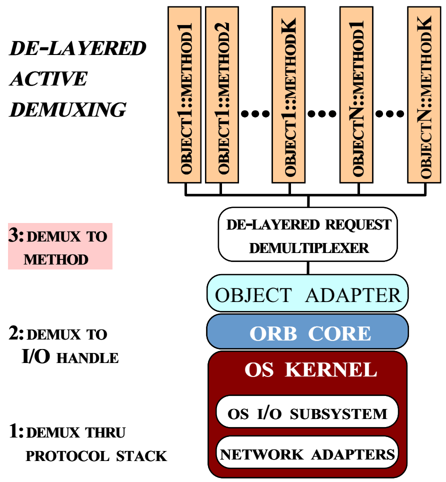
- Linear search based on `Orbix` demuxing strategy
- Perfect hashing based on GNU `gperf`
  - `http://www.cs.wustl.edu/~schmidt/gperf.ps.gz`
- Results at `http://www.cs.wustl.edu/~schmidt/GLOBECOM-97.ps.gz`

Problem: Meeting End-to-End QoS Requirements



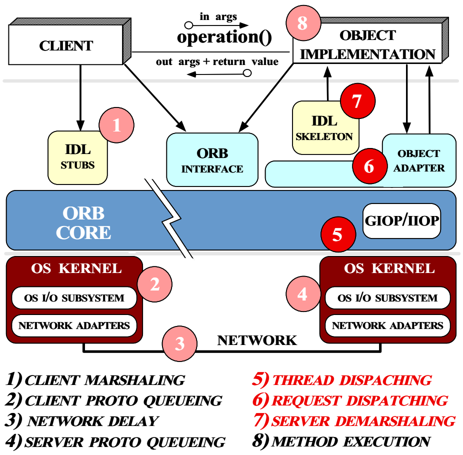
- **Design Challenges**
  - Specifying QoS requirements
  - Reducing demultiplexing latency
  - Meeting scheduling deadlines
  - Reducing presentation layer overhead

Solution: De-layered Active Demultiplexing



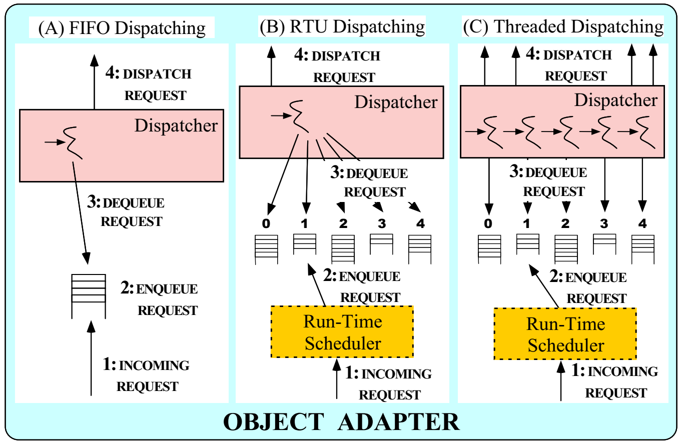
- **Solution Approach**
  - Pre-negotiate demuxing keys
  - Tunnel demuxing key with Object key
  - Use ACT pattern for validation
- **Related Work**
  - Yau and Lam '97
  - Dittia and Parulkar '97
  - Engler and Kaashoek '96

Problem: Meeting CORBA Request Deadlines



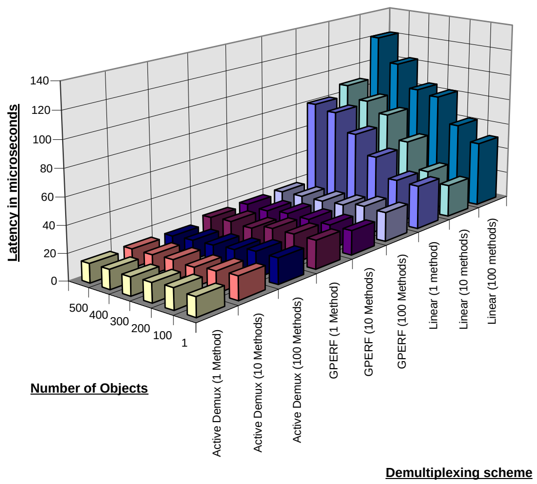
- **Design Challenges**
  - Specifying/enforcing QoS requirements
  - Focus on *Objects* and *Operations*
    - ∗ Not on threads or comm. channels
- **Assumptions**
  - Static scheduling
  - Non-distributed (initially)

Real-time Dispatching Experiments



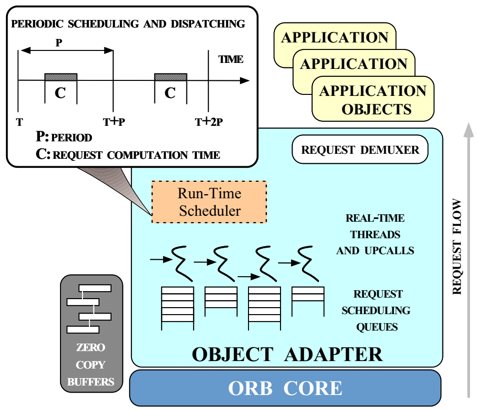
- `http://www.cs.wustl.edu/~schmidt/oopsla.html`

Demultiplexing Performance Results



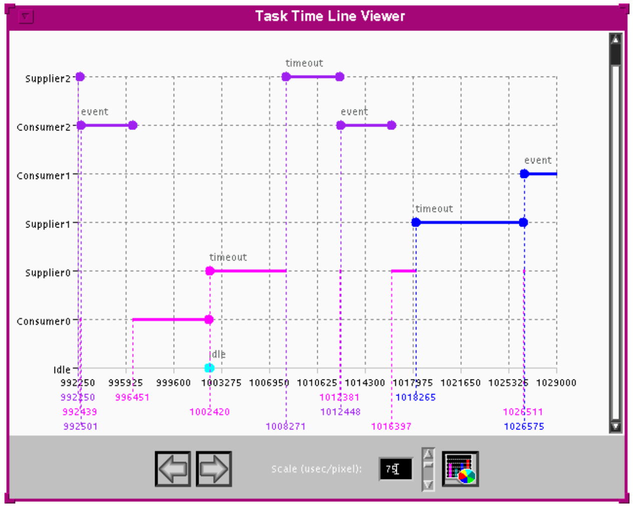
- **Synopsis**
  - gperf solution is 100% compatible, but static
  - Active demux isn't 100% compatible, but is dynamic

Solution Part 1: Real-time Object Adapter

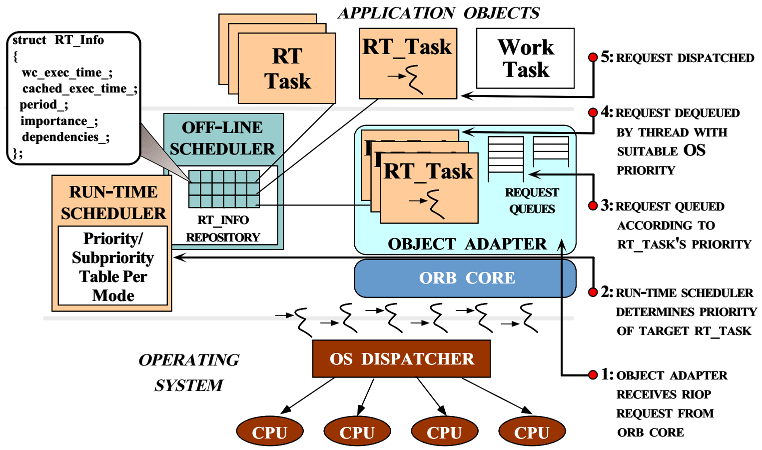


- **Solution Approach**
  - Integrate RT dispatcher into ORB
  - Support multiple request scheduling strategies
    - ∗ *e.g.*, RMS, RMS with Deferred Preemption, and EDF
- **Related work**
  - Zinky, Bakken, and Schantz, '95
  - Lee, Rajkumar, and Mercer '96

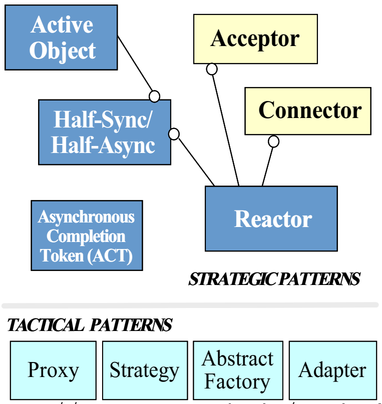
Timeline for Multi-threaded Object Adapter



Solution Part 2: Real-time Scheduling Service



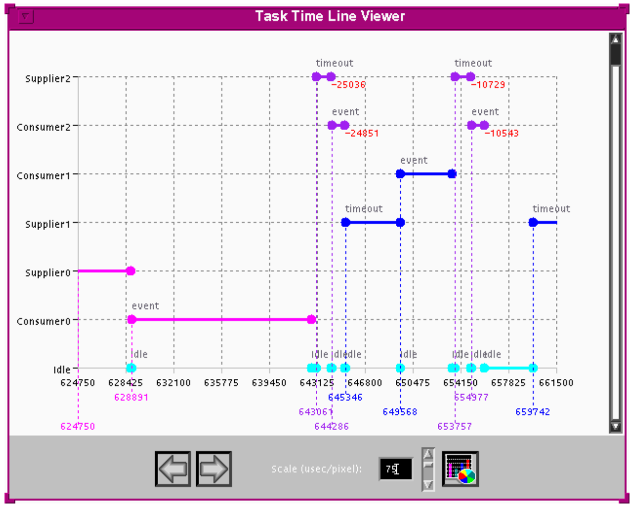
Key Patterns in TAO



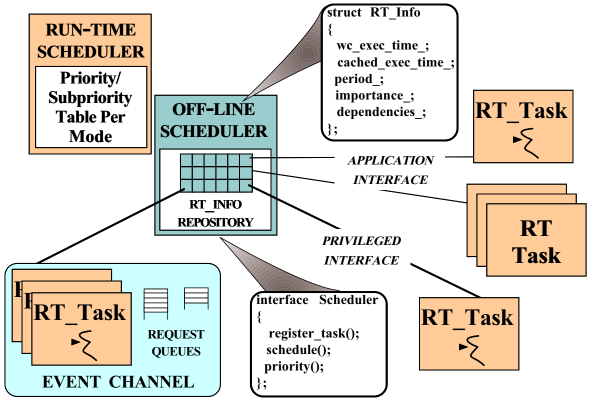
- **Definition**
  - "A recurring solution to a design problem in a particular context"
- **Benefits of Patterns**
  - Facilitate design reuse
  - Preserve crucial design information
  - Guide design choices
  - Document common traps and pitfalls

http://www.cs.wustl.edu/~schmidt/patterns.html

Timeline for FIFO Object Adapter

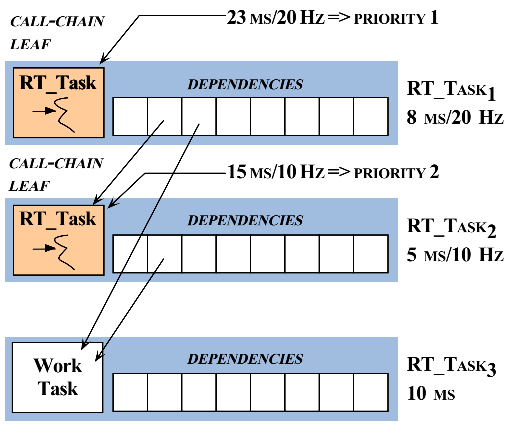


Scheduling Service Interfaces



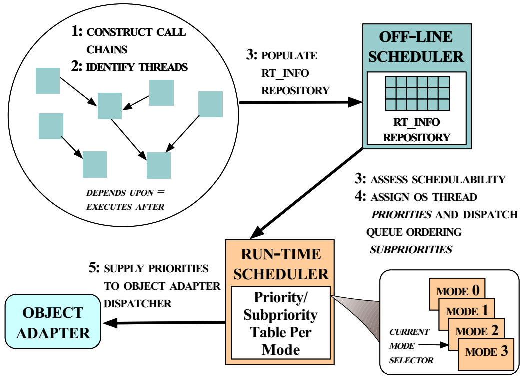
- **Components**
  - Application interface
    - Use RT_Infos
  - Privileged interface
    - Used by system tasks and services

Scheduling Service Internal Repository



- **Components**
  - RT_Info references
  - Vector of RT_Tasks called by each RT_Task
    - Vector records dependencies
  - Called-task chains are traversed to compute total CPU time and minimum period

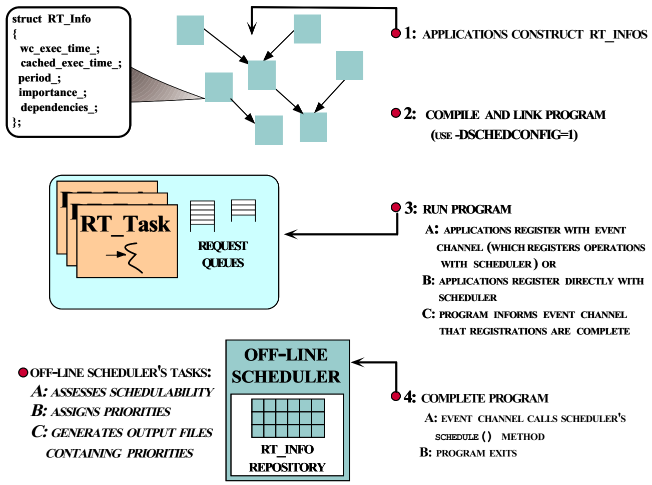
Scheduling Service Roles



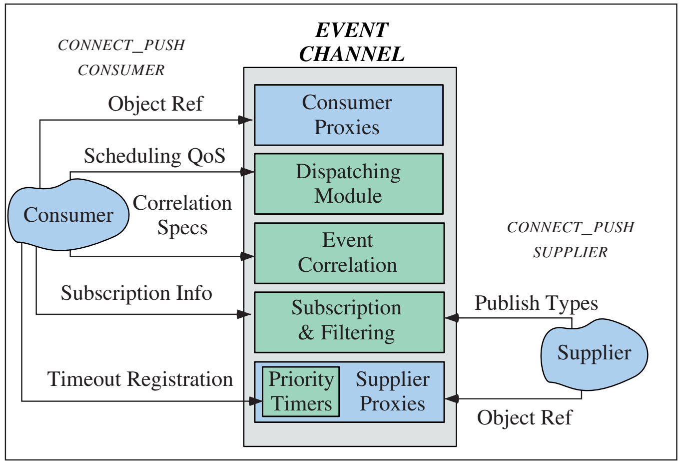
- **Components**
  - Offline
    - Assess schedule feasibility
    - Assign thread and queue priorities
  - Online
    - Supply priorities to Object Adapter's dispatcher

`http://www.cs.wustl.edu/~schmidt/TAO.ps.gz`

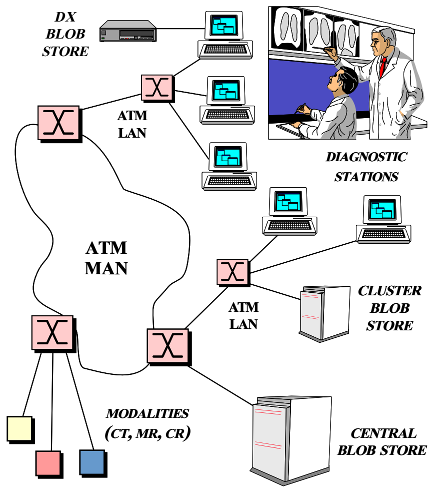
Scheduling Steps During Configuration Run



Collaboration in the RT Event Channel

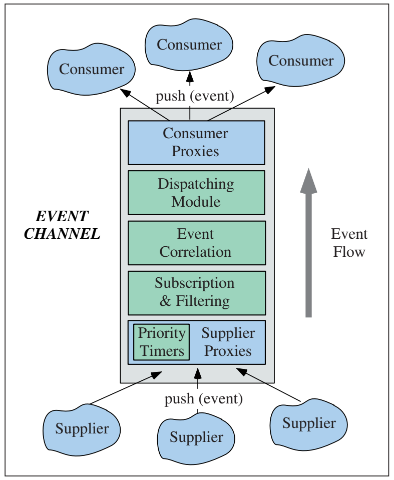


Applying CORBA to Medical Imaging



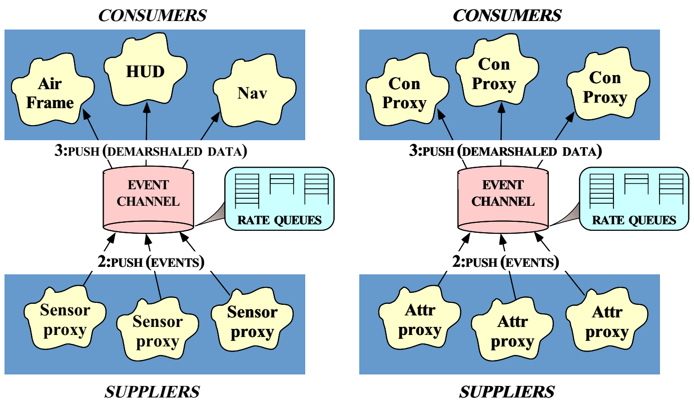
- **Domain Challenges**
  - Large volume of "Blob" data
    - *e.g.*, 10 to 40 Mbps
  - "Lossy compression" isn't viable
  - Prioritization of requests

Real-time Application Example



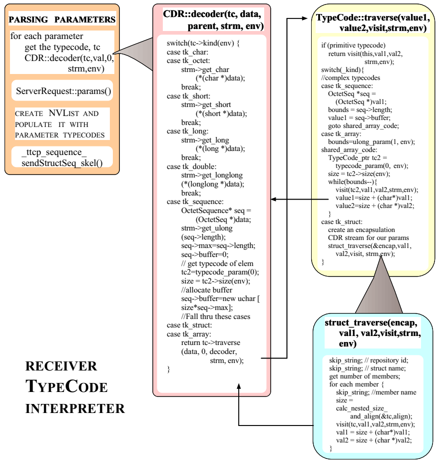
- **Real-time Event Channel Overview**
  - Used for event-driven real-time applications
  - QoS enhancements to COS Event Channel
  - Modular design addresses multiple use-cases
- **Real-time Event Channel Features**
  - Scheduling
  - Correlation dependencies
  - Filtering

RT Event Channel Use-cases



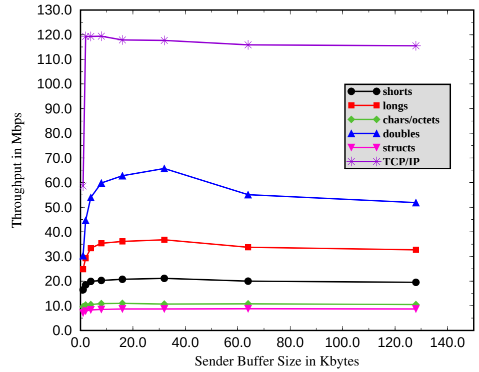
Avionics Network management

Solution: TypeCode Interpreter Optimizations



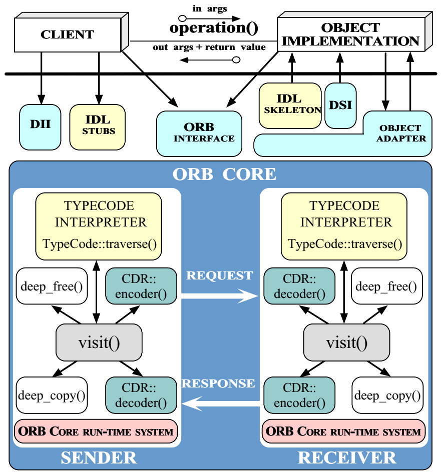
- **Solution Approach**
  - Optimized Typecode Interpreter
  - Based on SunSoft IIOP engine
- **Related work**
  - Hoschka '97
  - O'Malley, Proebsting, and Montz '94

Throughput of the SunSoft IIOP Implementation



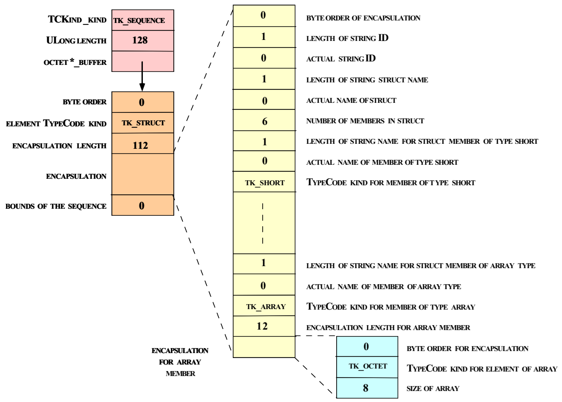
- Experimental design
  - Transfer 64 Mbytes of "oneway" data
  - Various types of data
    - ∗ *e.g.*, char, short, long, double, struct, etc.
    - ∗ Sender buffer sizes ranged from 1K to 1024K
    - ∗ Socket queues were 64k (maximum)
    - ∗ Network was 155 Mbps ATM

Problem: Reducing Protocol Engine Overhead



- **Design Challenges**
  - Small memory footprint
  - Predictable performance
  - Minimize the typecode interpreter overhead

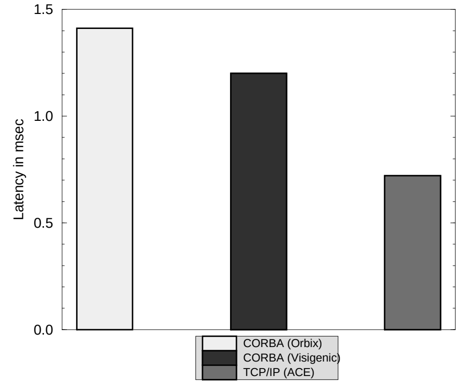
TypeCode Layout for Sequence of BinStructs



- TypeCode Description in CDR format

```
// 32 bytes
struct BinStruct{
  short s; char c; long l;
  octet o; double d;
  octet pad[8];
};
typedef sequence<BinStruct>
        StructSeq;
```

Latency Measurements

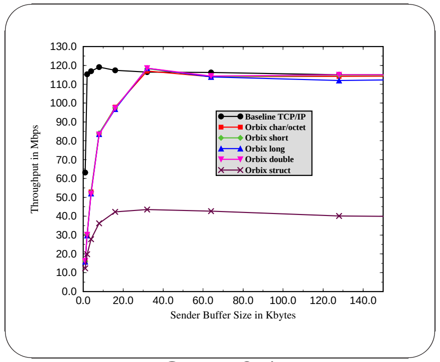


http://www.cs.wustl.edu/~schmidt/ICDCS-97.ps.gz

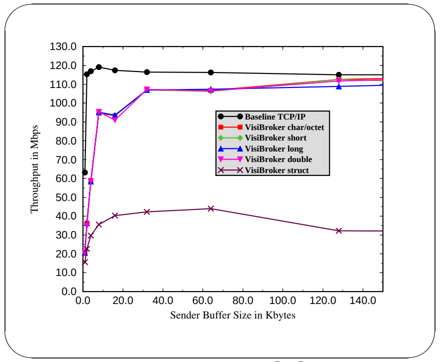
Challenges of Optimizing Complex Softare

- Problem
  - Optimizing complex software is hard
  - Small "mistakes" are costly over high-speed networks
- Solution Approach (Iterative)
  - Pinpoint sources of overhead via *white-box* metrics
    - *e.g.*, `Quantify`, `TNF`, etc.
  - Apply optimization principles
  - Validate via white-box and black-box metrics

Throughput Results using SII



Orbix 2.1

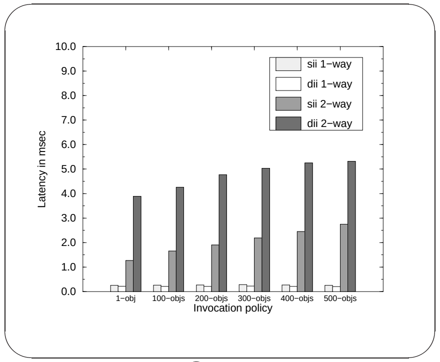


VisiBroker 2.0

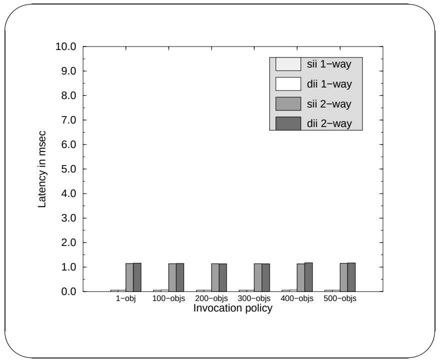
SII results – http://www.cs.wustl.edu/~schmidt/SIGCOMM-96.ps.gz

DII results – http://www.cs.wustl.edu/~schmidt/GLOBECOM-96.ps.gz

Scalability Measurements



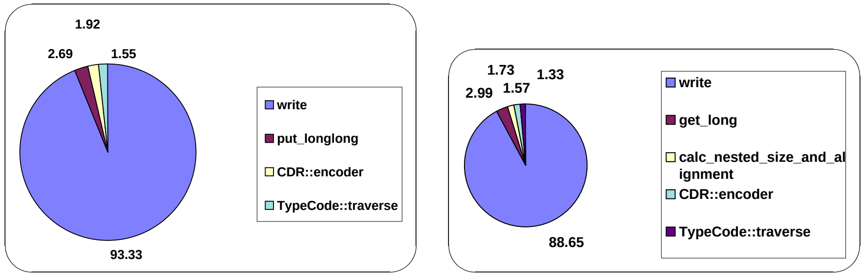
Orbix



VisiBroker

http://www.cs.wustl.edu/~schmidt/ICDCS-97.ps.gz

Sender-side Analysis of SunSoft IIOP Implementation



double **struct**

Percent Execution Time for doubles and structs

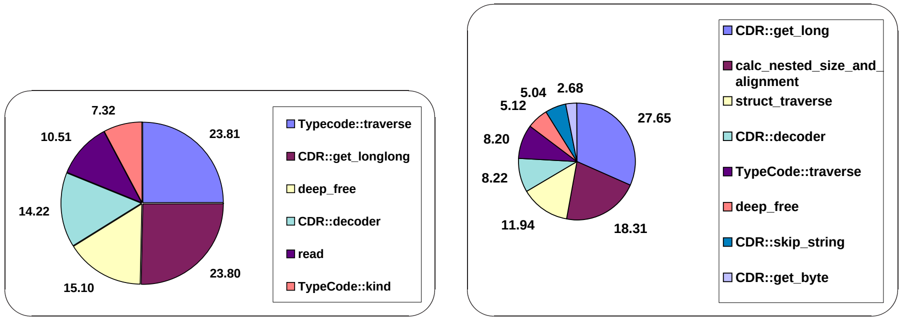
Problems and Solutions

- Problems
  - Invocation overhead for small, frequently called methods
- Solution
  - Inline method calls
- Principle
  - Optimize for the common case

Optimization Principles

Number	Principle
1	Optimize for the common case
2	Eliminate gratuitous waste
3	Replace inefficient general-purpose methods with efficient special-purpose ones
4	Precompute values, when possible
5	Store redundant state to speed up expensive operations
6	Pass information between layers

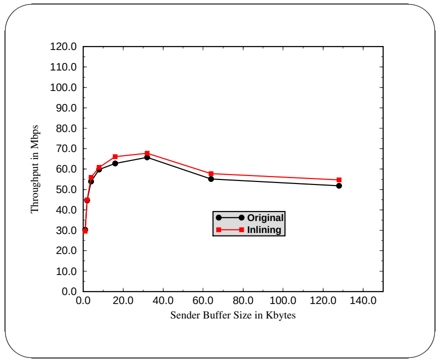
Receiver-side Analysis of SunSoft IIOP Implementation



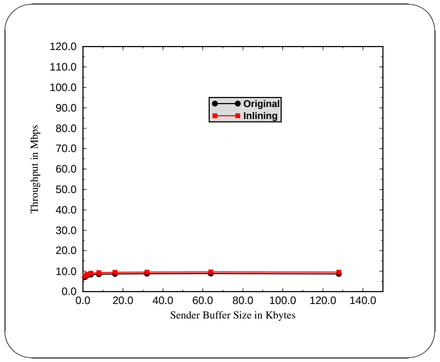
double **struct**

Percent Execution Time for doubles and structs

Throughput After 1st Optimization



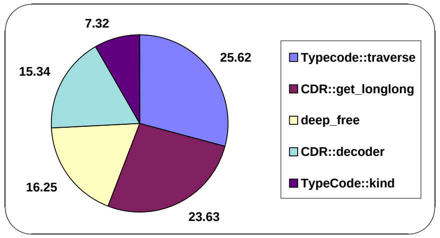
double



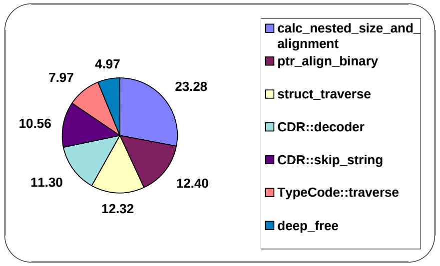
struct

Throughput for doubles and structs

Receiver-side Analysis of IIOP Implementation (1st Opt)



double



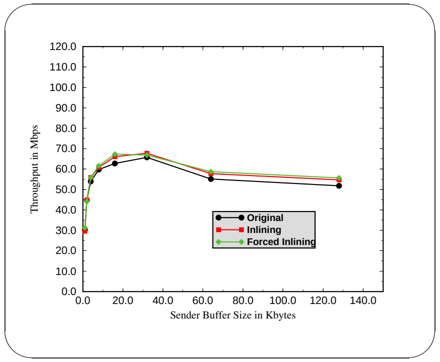
struct

Throughput for doubles and structs

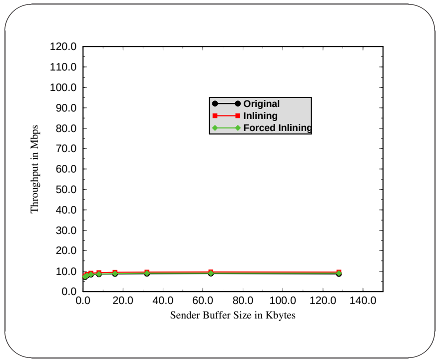
Problems and Solutions

- Problems
  - Lack of C++ compiler support for aggressive inlining
- Solution
  - Replace inline methods with preprocessor macros
- Principle
  - Optimize for the common case

Throughput After 2nd Optimization



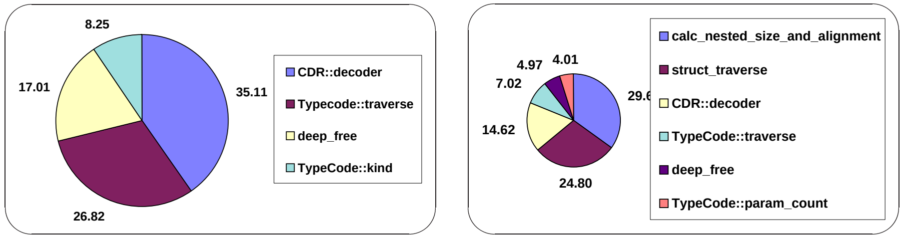
double



struct

Throughput for doubles and structs

Receiver-side Analysis of IIOP Implementation (2nd Opt)



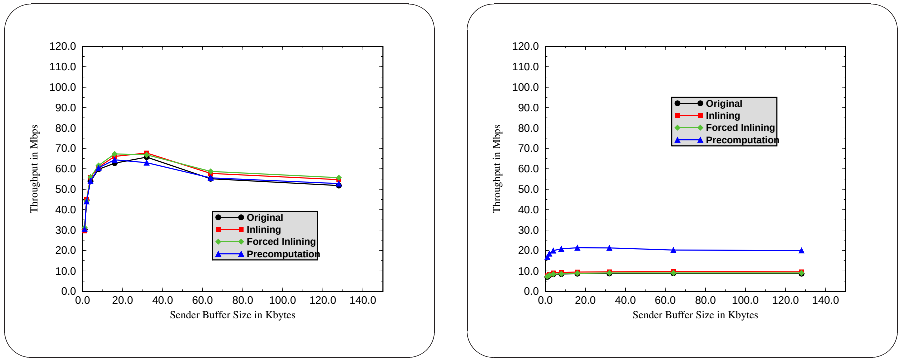
double struct

Percent Execution Time for doubles and structs

Problems and Solutions

- Problems
  - Too many method calls
  - Computing the same quantity repeatedly
- Principles
  - Precompute
  - Add extra state
  - Pass information through layers
  - Convert generic methods to special-purpose ones

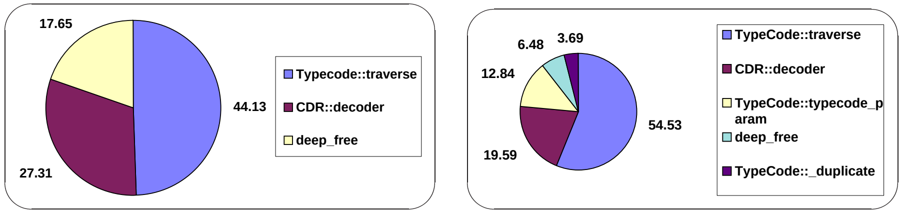
Throughput After 3rd Optimization



double struct

Throughput for doubles and structs

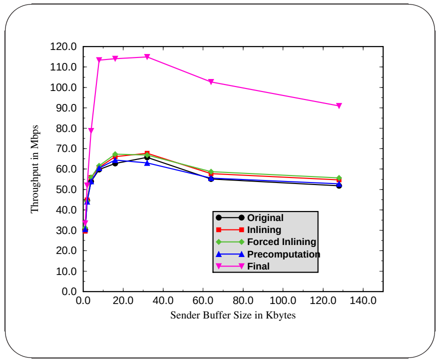
Receiver-side Analysis of IIOP Implementation (3rd Opt)



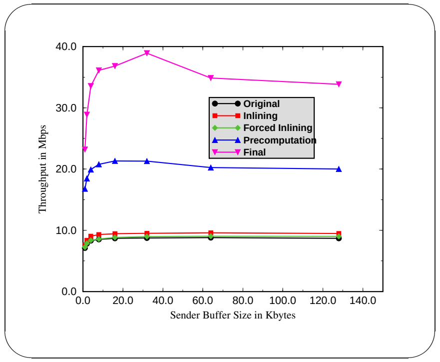
double struct

Percent Execution Time for doubles and structs

Throughput After Optimizations



double



struct

Throughput for doubles and structs

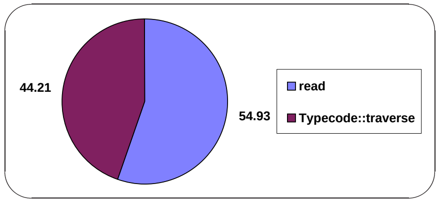
Problems and Solutions

- Problems
  - Code for the TypeCode interpreter “traverse” is monolithic and large
  - Bad for cache
- Principles
  - Break the monolithic “traverse” routine into specialized methods
  - Perform outlining (Peterson, Mosberger et.al. Sigcomm 96).

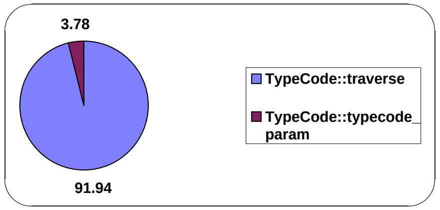
Problems and Solutions

- Problems
  - Expensive no-ops for memory deallocation
- Principles
  - Eliminate gratuitous waste
  - Specialize generic methods

Receiver-side Analysis of IIOP Implementation after Optimizations



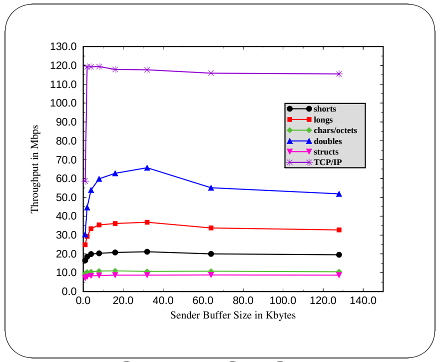
double

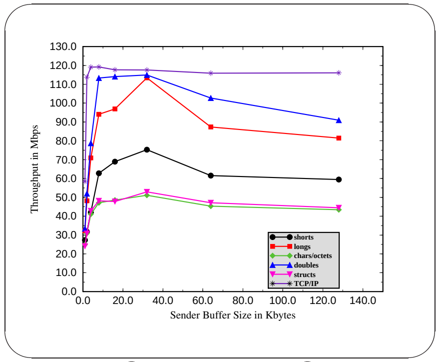


struct

Percent Execution Time for doubles and structs

Throughput Comparisons

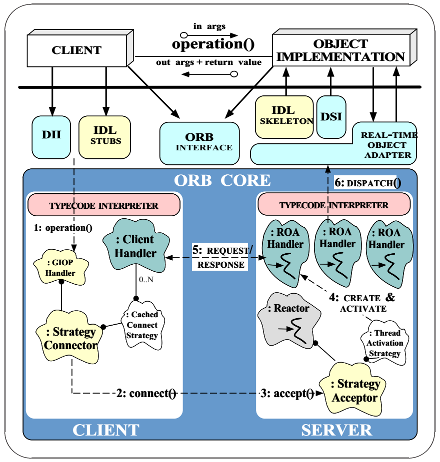


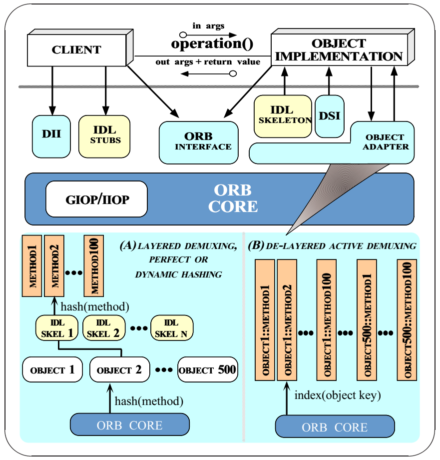


Original SunSoft **Optimized TAO**

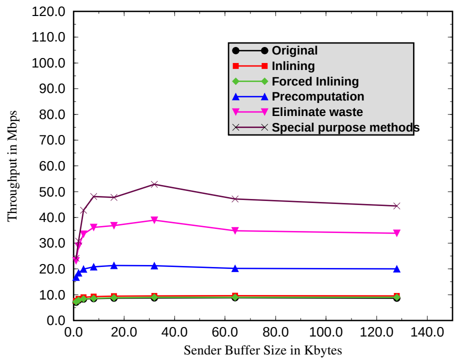
Throughput for SunSoft and TAO Versions of IIOP

Current Status of TAO





Throughput After Optimizations for Cache



Performance for sending structs

Results for Typecode Interpreter Optimizations

- Our measurement-driven, principle-based optimization process improved TAO's IIOP protocol engine performance as follows
  - 1.8 times for doubles
  - 3.3 times for longs
  - 3.75 times for shorts
  - 5 times for chars/octets
  - 6.2 times for structs
- Results available at `http://www.cs.wustl.edu/~schmidt/IIOP.ps.gz`

Concluding Remarks

- Current Focus: High-performance, Real-time ORBs
  - Reducing latency via *de-layered active demuxing*
  - Applying optimization principles to TypeCode interpreter
  - Enforcing periodic deadlines via Real-time Object Adapter
    - *i.e.*, support static request scheduling
  - Applying optimization principles to presentation layer
- Future Work
  - Pinpoint non-determinism and priority inversions in ORBs
  - Dynamic scheduling of requests
  - Distributed QoS and integration with RT I/O Subsystem
  - IDL compiler and optimized stub generation
  - TypeCode compiler optimizations