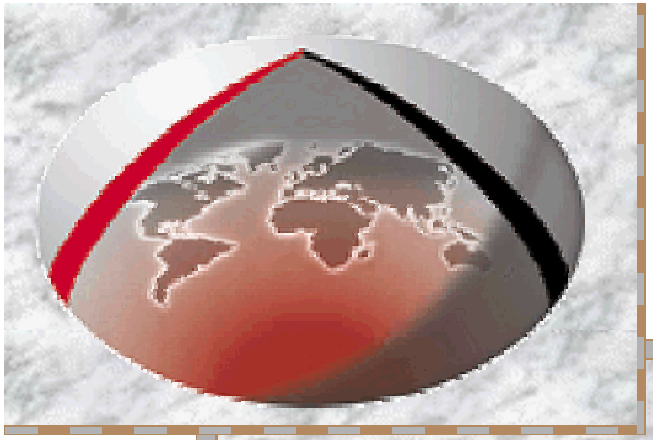


CORBA Asynchronous Messaging

Bill Binko

The Technical Resource Connection

cwbinko@trcinc.com

Presentation Preview

- CORBA Messaging Overview
- Asynchronous Method Invocation (AMI)
- Time Independent Invocation (TII)
- QoS Framework Overview
- MOM Product Integration Strategies
- Miscellaneous Issues
- Open Discussion and Q/A

Messaging Overview

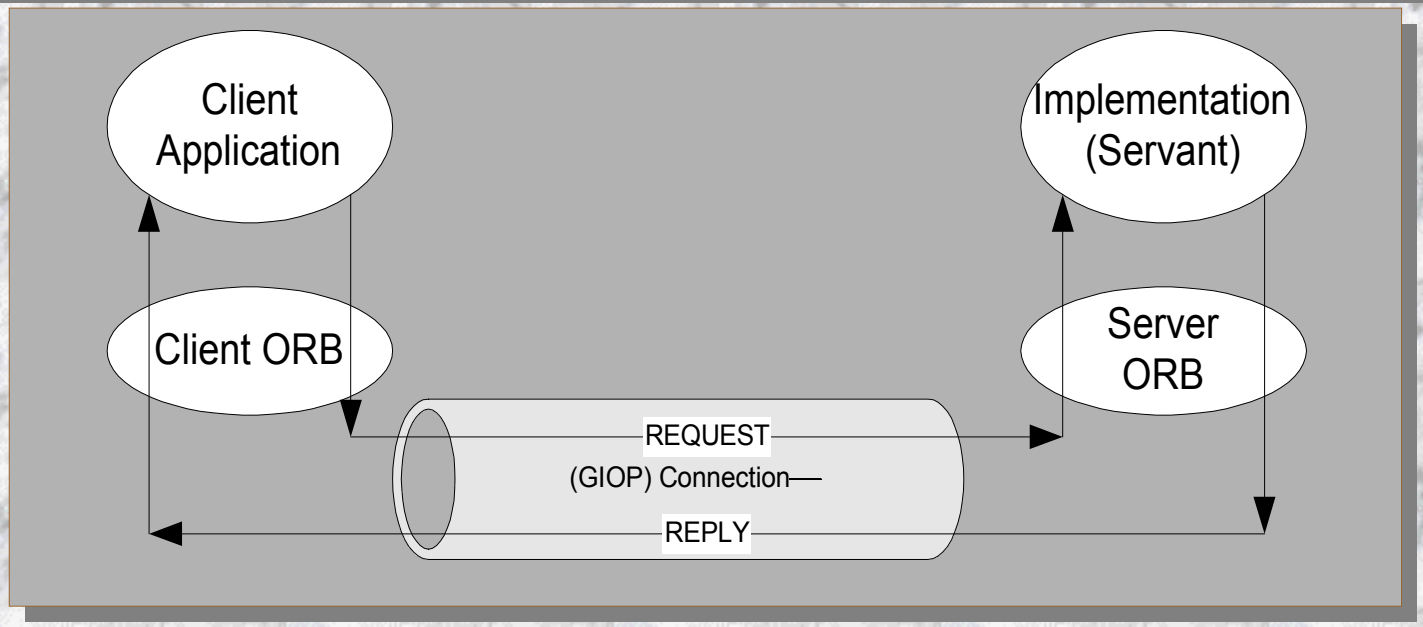
- CORBA Messaging is an integral part of version 2.4 of CORBA
- Provides ***strongly typed*** asynchronous communications
- Adds support for such QoS as store-and-forward and priority delivery
- Allows administrative routing of requests

Background – RPC Vs. MOM

- CORBA originally based on RPC systems
  - Generated Stubs/Skeletons provide networking support
  - IDL Compiler handles converting the invocation's parameters and return value into remote request/reply messages
- Remote calls "look like" local calls (Location Transparency)
- Provides Compile-Time Error Checking
  - Request and Reply Messages will always be in correct form (syntax, not semantics)

Background – RPC Vs. MOM

- CORBA's RPC Issues
  - Forces a strict request/reply sequence (Blocking)
  - Very Connection-oriented
  - Tight Coupling (At several levels)

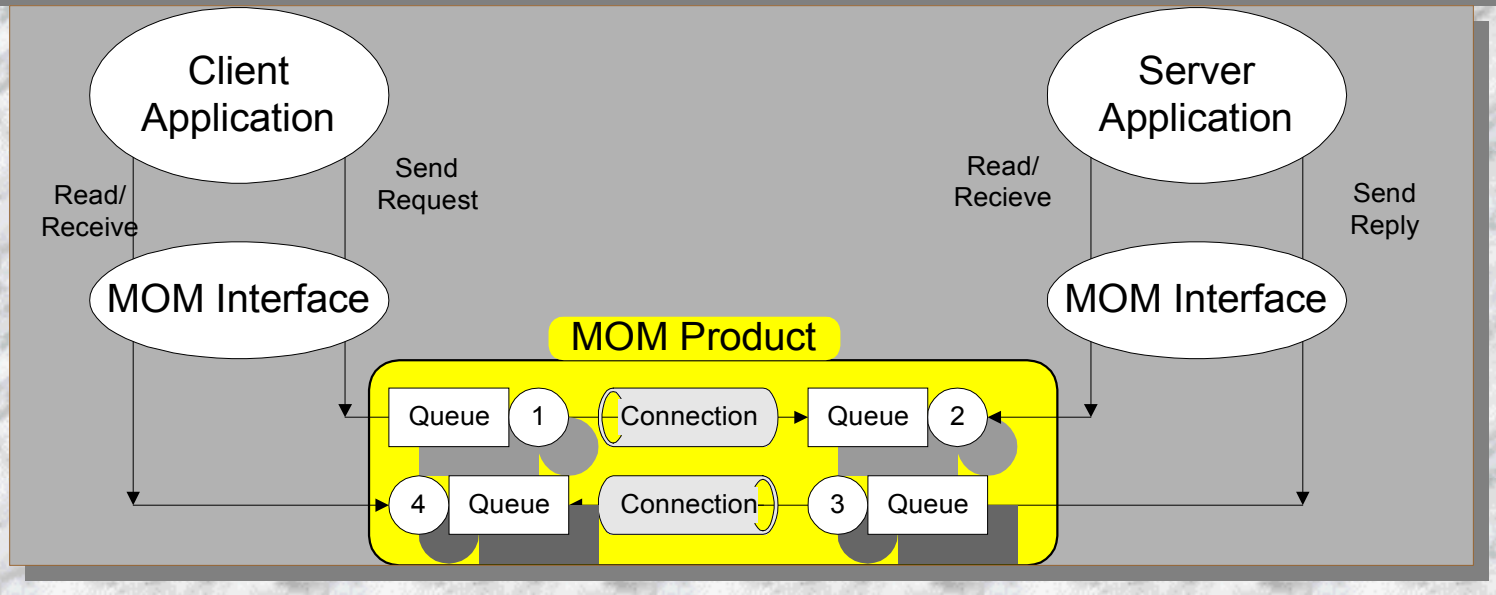


Background – RPC Vs. MOM

- Message Oriented Middleware products use messages and queues
  - Allows decoupling of sender/receiver
  - Allows administration of routing and QoS
  - Provides little/no type safety
- Request and Replies are independently targeted
- Messages are self-contained
  - Contain all information needed for deliver and execution
  - Are often meaningful without the originator's presence

Background – RPC Vs. MOM

- Does NOT look like local calls
  - Messages are built by hand and given to the MOM interface
  - XML Becoming standard format for Messages (JAXM, BizTalk, etc.)



Background – RPC Vs. MOM

- Typing discussion

Background – Messaging Goals

- Provide *some* of the benefits of MOM
  - QoS such as store/forward, priority, etc.
  - Administrative routing
  - Disconnected clients/servers
  - No P&S etc. *(Notification covers that)*
- Maintain type-safety
- Interoperate with existing CORBA servers
- Client-only changes (except for OTS Calls)

Background: Messaging QoS

- Priority: Controls the Request Priority while in Queues
- Timing: Controls various timeout aspects
  - *Request Start Time, Timeout*
  - *Reply Start Time, Timeout*
  - *Round-Trip Timeout, etc.*
- Routing: Controls Forwarding or Store/Forwarding
- Max Hops: Sets max hops before request dies
- Ordering: Sets priority/temporal/"deadline" sorting in queue

Usage Scenario: Client Disconnection

Usage Scenario: Client Disconnection

Order Entry

File View Options Window Help

NYSE | CBOE | NASD

AMZN	134
DELL	76 3/4
GE	80
IBM	83
ETC	10 1/2

Symbol: AMZN

Quantity: 1000

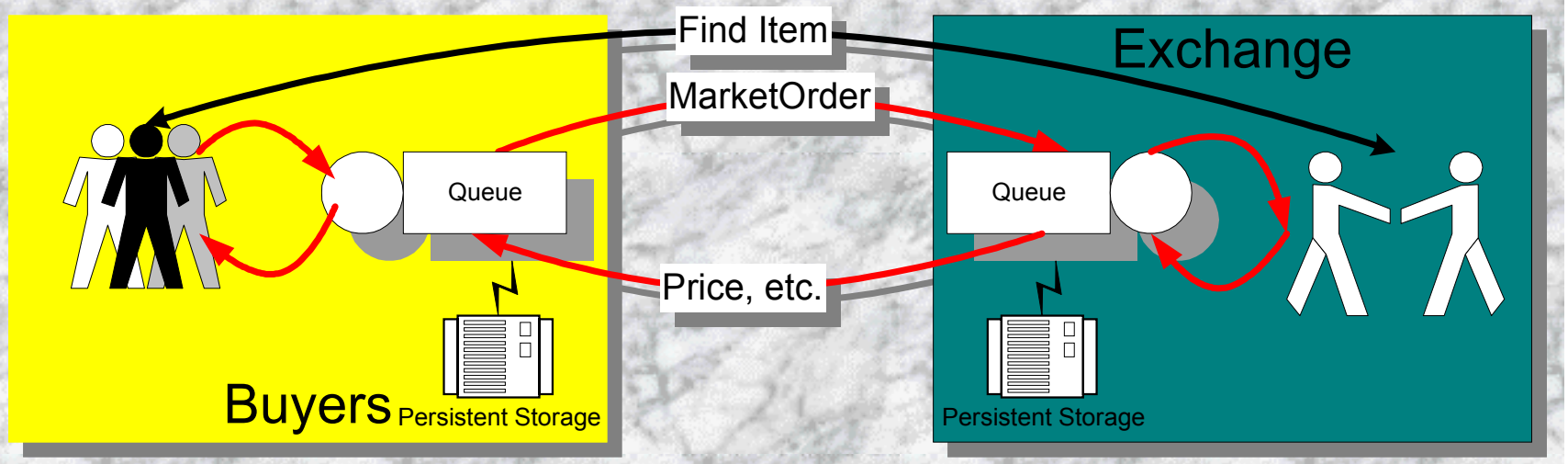
Price/Share 135

Order Type Limit Order

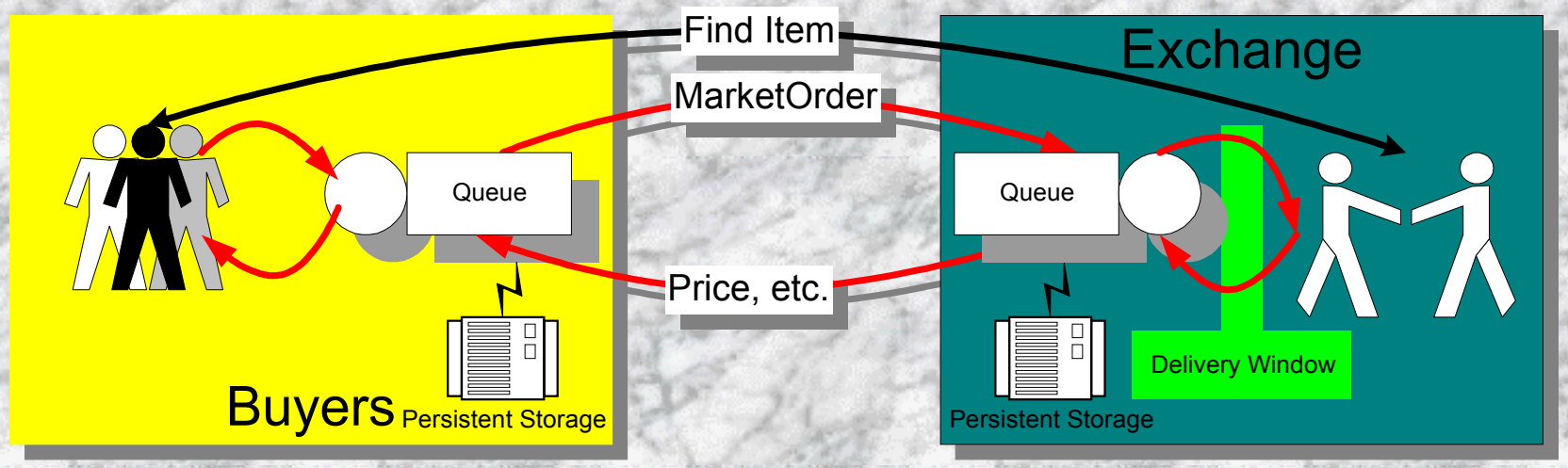
Submit

Order Id	Security	Req. Qty/$	Qty/$	Status
OID3958	ORCL	100/$5	N/A	Pending
OID35682	MSFT	500/$52	500/$52 1/2	Complete

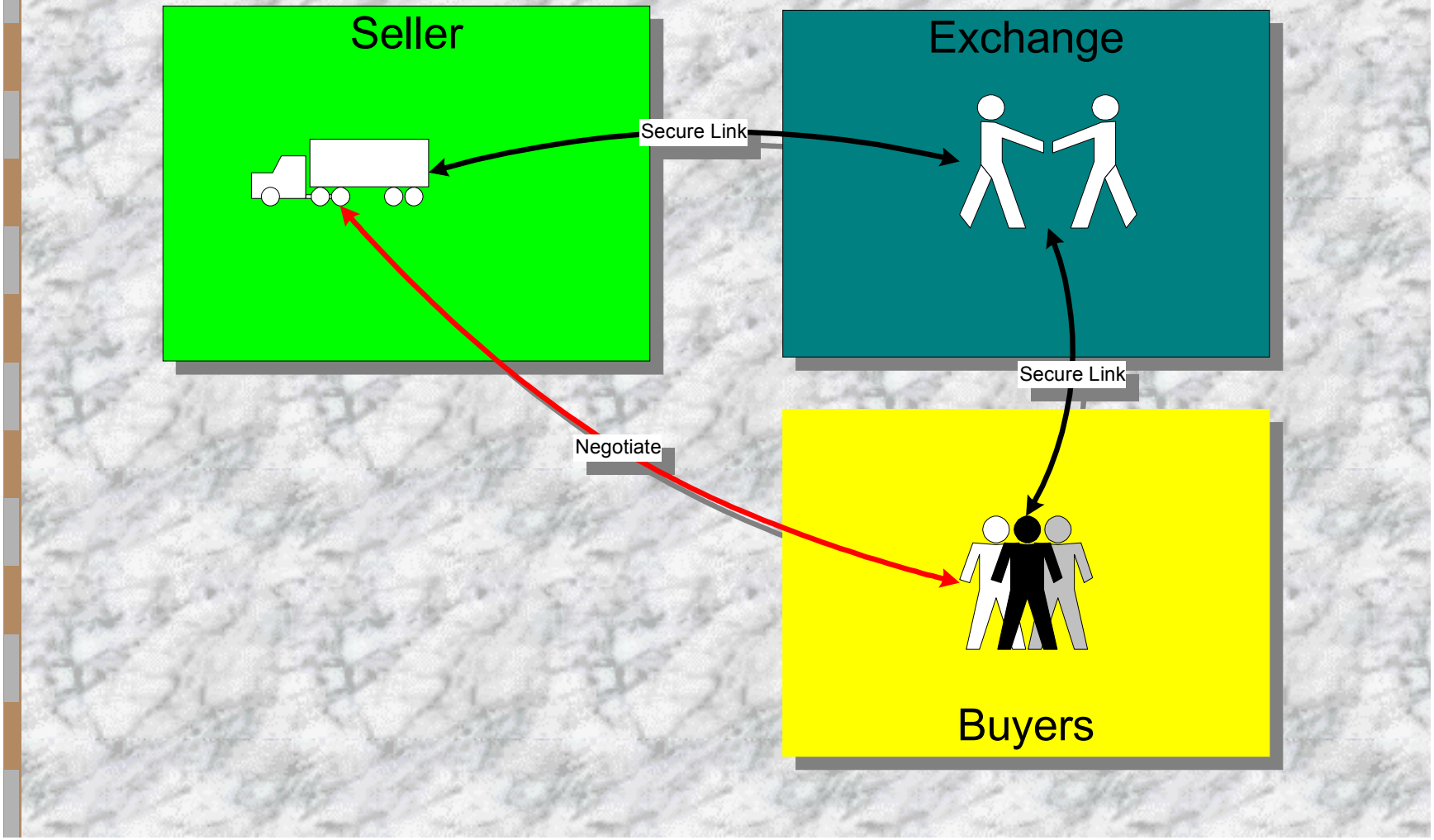
Usage Scenario: Client Disconnection



Usage Scenario: Enhanced QoS

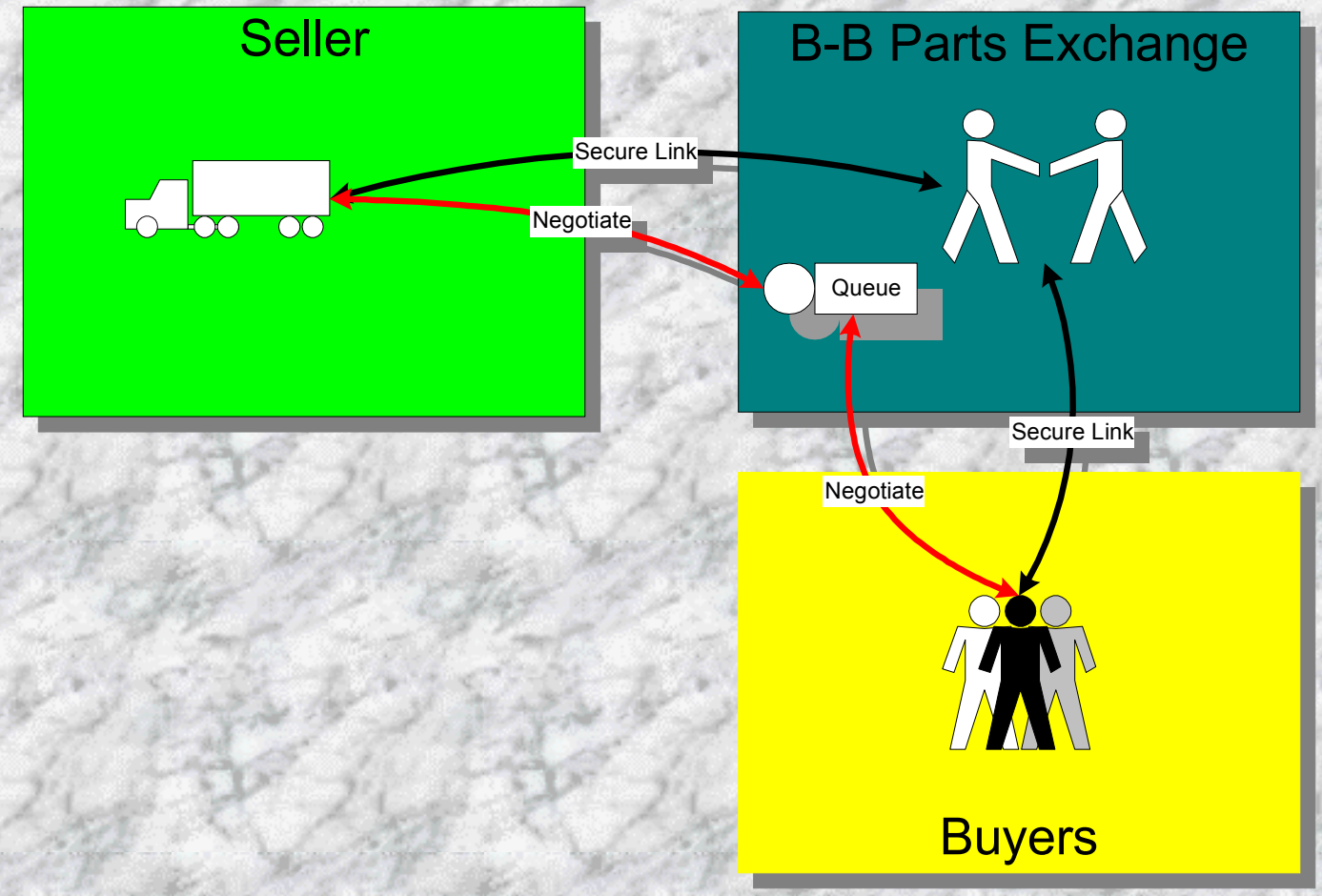


Usage Scenario: Routing Negotiations

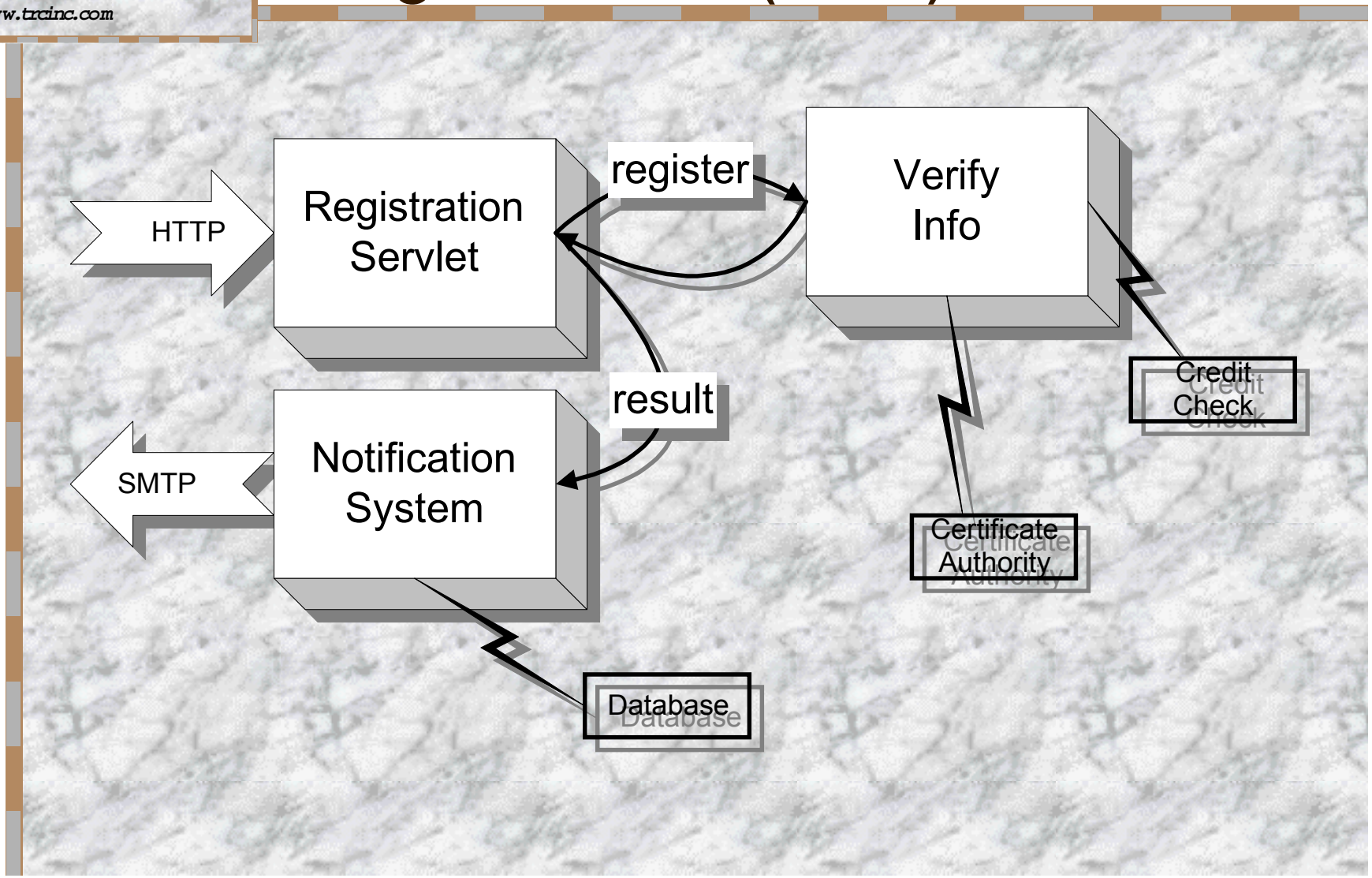


The Technical Resource Connection
www.trcinc.com

Usage Scenario: Routing Negotiations



Usage Scenario: Registration (AMI)



Background – Problems

- GIOP (CORBA's transport protocol) is very request/reply oriented
- GIOP Reply message has no destination field (cannot be routed)
- Clients have no identity in CORBA
- Users need an alternative to "normal" call semantics

Background – Problems
Blocking Calls

- IDL

```
interface A {
   long f(in float a, out float b, inout float c);
};
```

- Java

```
A ref = AHelper.narrow(someObjRef);

FloatHolder bH = new FloatHolder();
FloatHolder cH = new FloatHolder(1.2);

int result = ref.f(2.0,bH,cH);
//use result, bH, cH
```

Background – Approach

- Decouple Request/Reply at App Level
  - AMI: New invocation interface (peer to SII and DII)
  - Allows typed asynchronous calls (via Callbacks/Polling)
- Decouple Request/Reply at Wire Level
  - Specify Router Interfaces (TII)
  - Wraps GIOP Request/Reply in Routable Messages
  - Allows MOM-type queuing/routing of GIOP-level messages
- Extensible QoS Policy Framework
  - Provides control over the system

AMI: Asynchronous Method Invocation

- Allows the ORB to separate a request from its reply in a type-safe manner
- Two models: Callback and Polling
- Both rely on the IDL compiler for support (special AMI stubs)
- Like SII/DII – client-side only: server can't tell which interface is used
  - When used with TII, Transactional Servers must know about it (more on that later)

AMI Comparison

- For comparison, we will go over all of the available invocation interfaces:
  - (Normal) Synchronous Call
  - Oneway Call
  - Deferred Synchronous Call
  - Asynchronous Call (new)
- ***Disclaimer****: These are typical call-paths and are not complete. They are solely for illustration.*

AMI Comparison: SII Synchronous Call

- IDL

```
interface A {
   long f(in float a, out float b, inout float c);
};
```

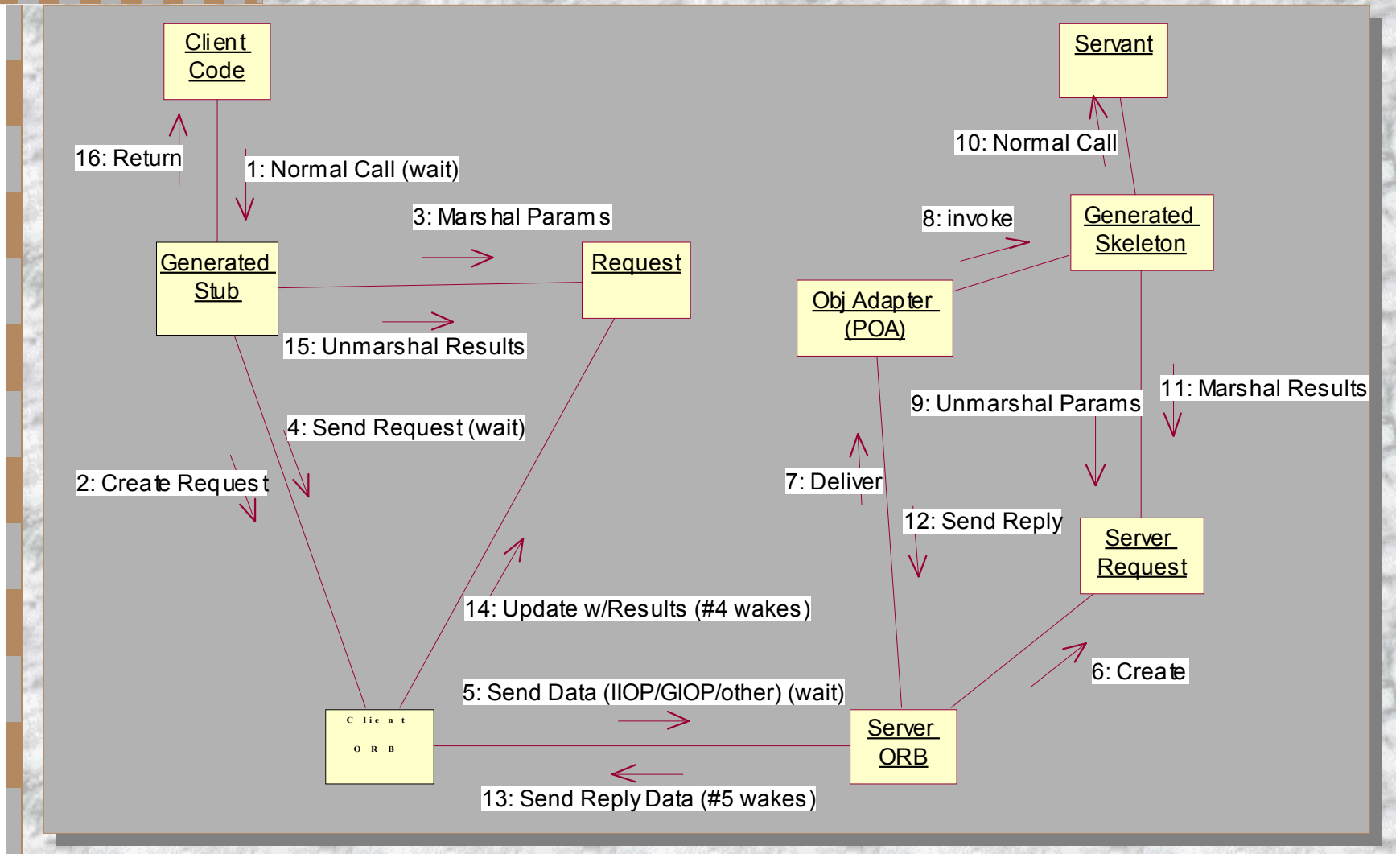
- Java *(C++, C, Smalltalk, Perl, COBOL, Others Available)*

```
A ref = AHelper.narrow(someObjRef);

FloatHolder bH = new FloatHolder();
FloatHolder cH = new FloatHolder(1.2);

int result = ref.f(2.0,bH,cH);
//use result, bH, cH
```

AMI Comparison: SII Synchronous Call



AMI Comparison: Oneway Call

- IDL

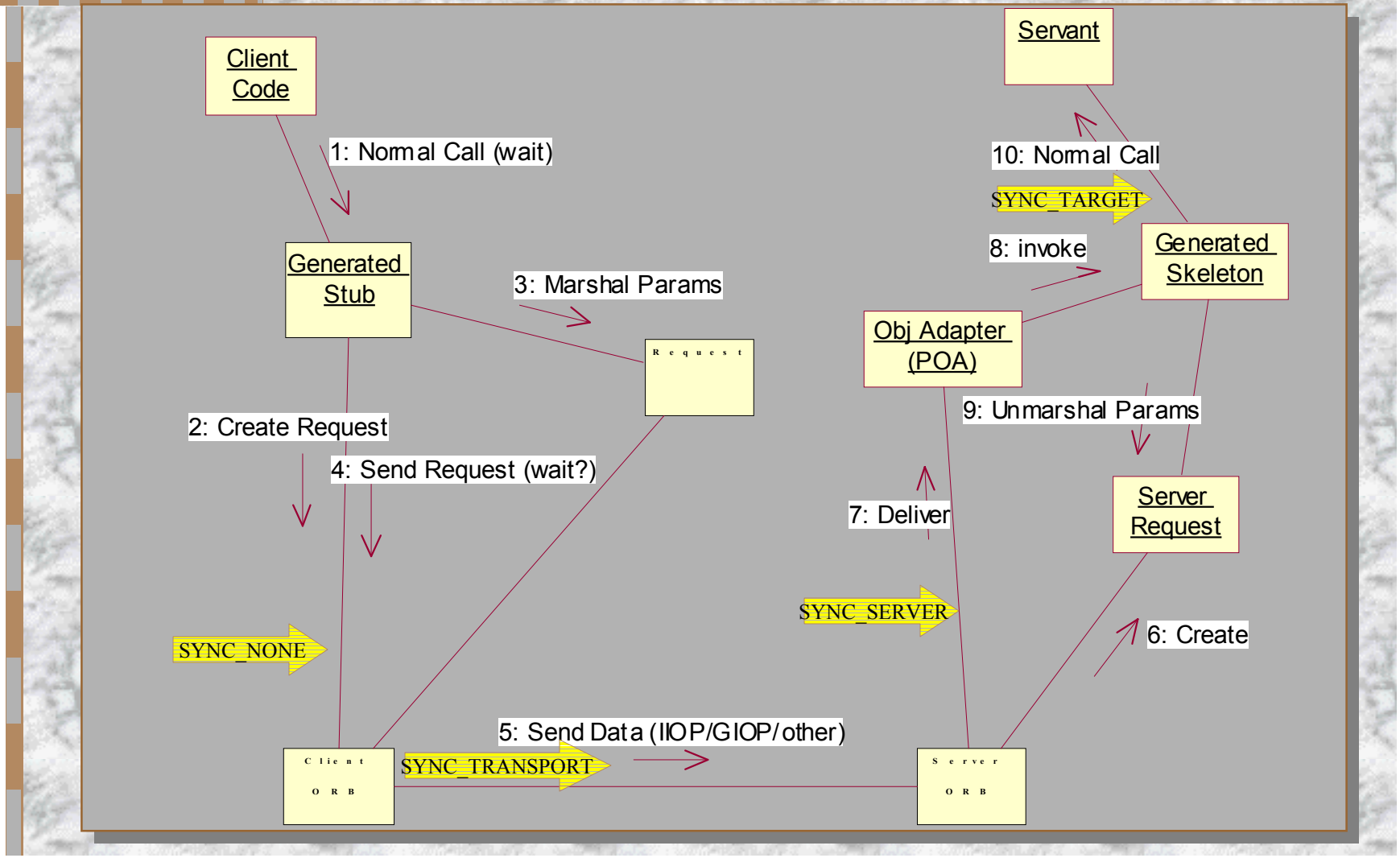
```
interface A {
   oneway void g(in float a);
};
```

- Java

```
A ref = AHelper.narrow(someObjRef);
// maybe set SYNC_SCOPE on ref

ref.g(2.0);
```

AMI Comparison: SII Oneway Call



AMI Comparison: DII Deferred Sync Call

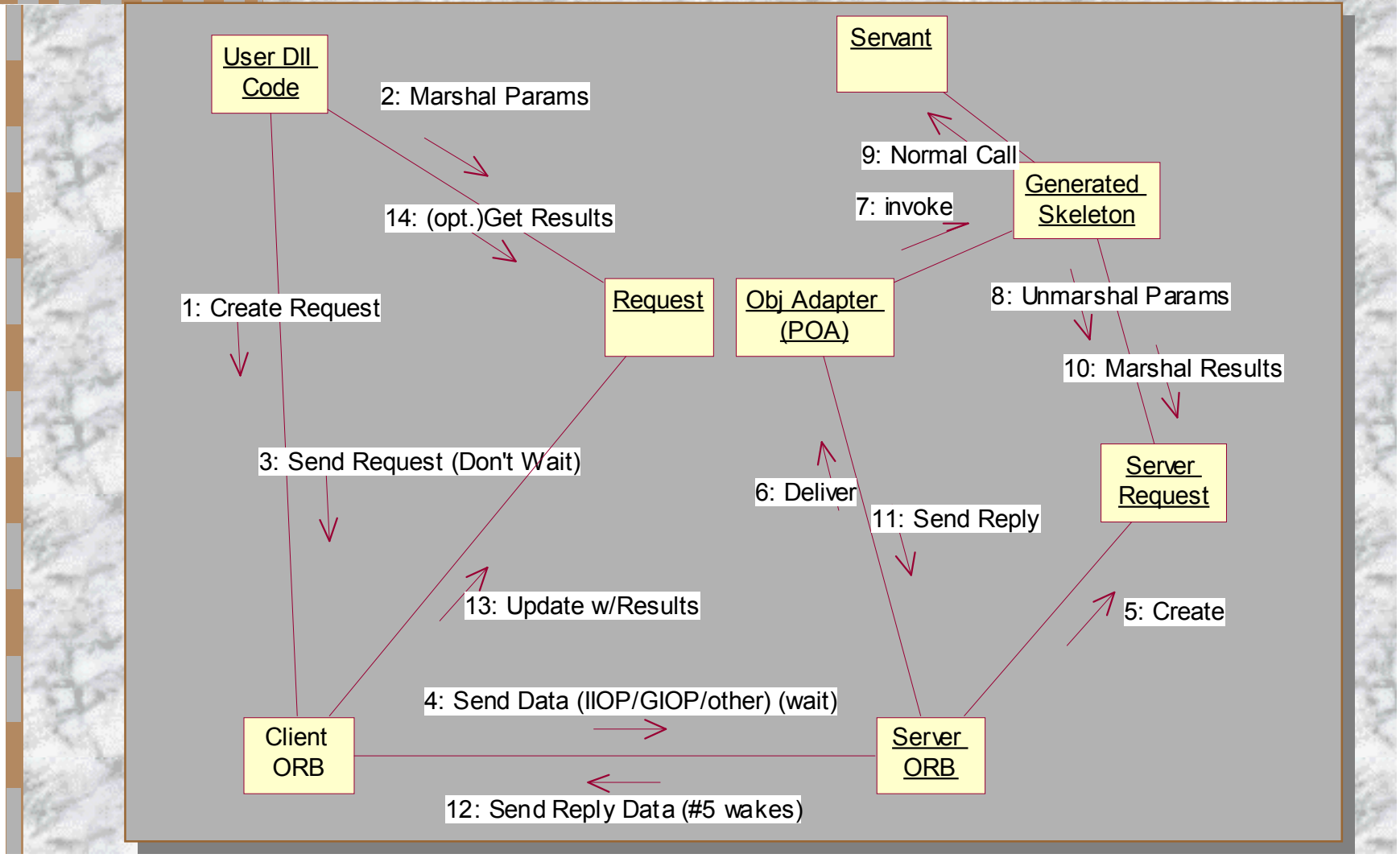
- IDL

```
interface A {
   long h(in float a);
};
```

- Java

```
org.omg.CORBA.Request r = someObjRef._request("h");
r.set_return_type(
    _orb().get_primitive_tc(TCKind.tk_long));
Any s = r.add_in_arg();
s.insert_float(2.0);
r.send_deferred(); //Can do other stuff here
r.get_response();  //will wait for result
int result = r.return_value().extract_long();
//use result
```

AMI Comparison: DII Deferred Sync Call



AMI Comparison: Async Polling Model (new)

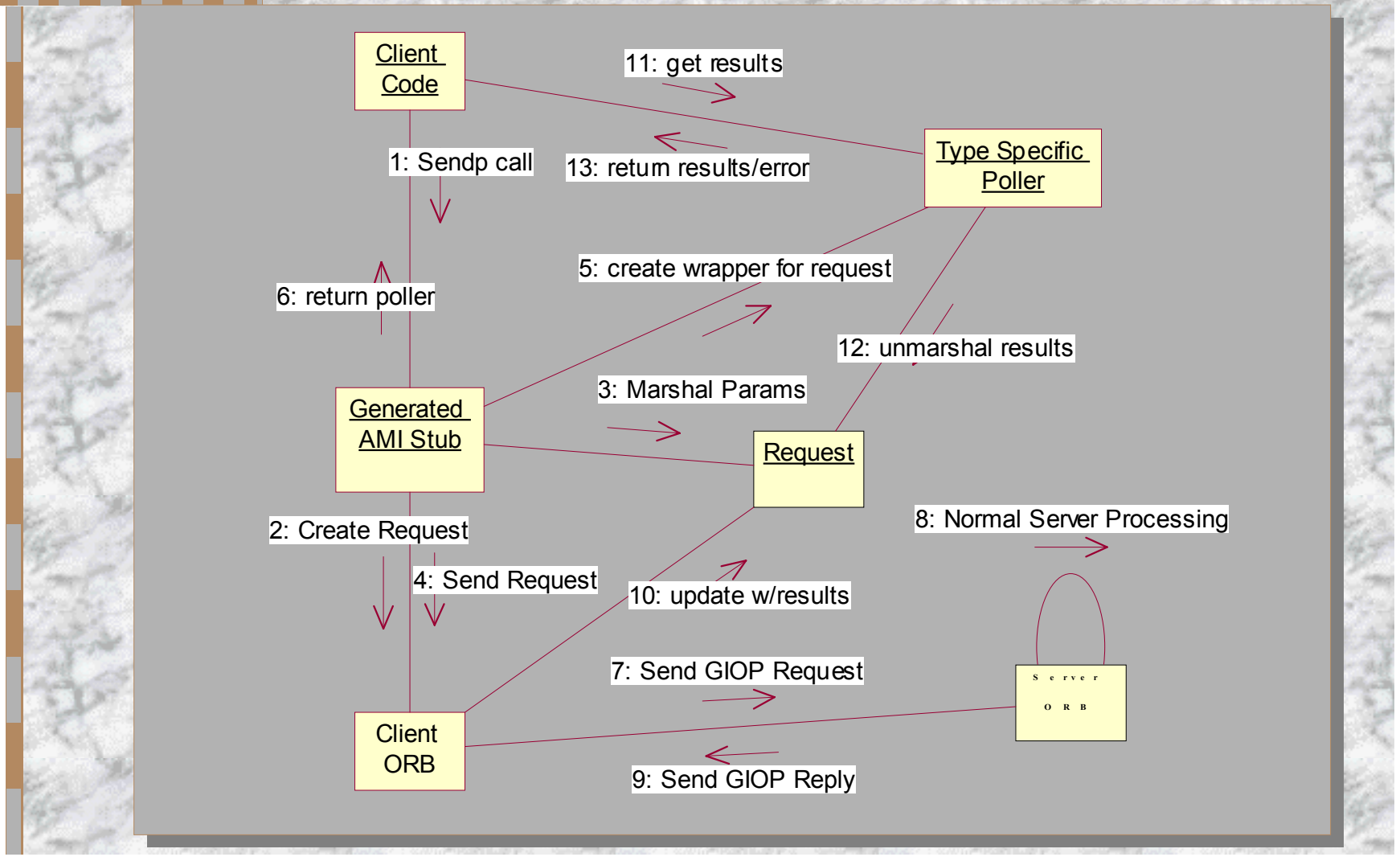
- IDL

```
interface A {
   long f(in float a, out float b, inout float c);
};
```

- Java

```
A ref = AHelper.narrow(someObjRef);
//ref is actually a stub
//only pass in/inout params
AMI_APoller poller = ref.sendp_f(2.0,1.2);
//Can do other stuff
FloatHolder bh = new FloatHolder();
FloatHolder ch = new FloatHolder();
int result = poller.f(/*timeout*/-1,bh,ch);
//Can use result, b, c
```

AMI Comparison: Async Polling Model (new)



AMI Comparison: Async Callback (new)

- IDL

```
interface A {
   long f(in float a, out float b, inout float c);
};
```

- Java

```
A ref = AHelper.narrow(someObjRef);
//ref is actually a stub
//create handler (implicit act. On root POA)
AHandler_impl handler = new AHandler_Impl();
//only pass in/inout params
ref.sendc_f(handler._this(), 2.0,1.2);
//handler's f() method will be called
//back w/ inout/out/retval
```

AMI Comparison: AMI Callback Implied IDL

- IDL

```
interface A {
   long f(in float a, out float b, inout float c);
};
```

- Implied Callback IDL

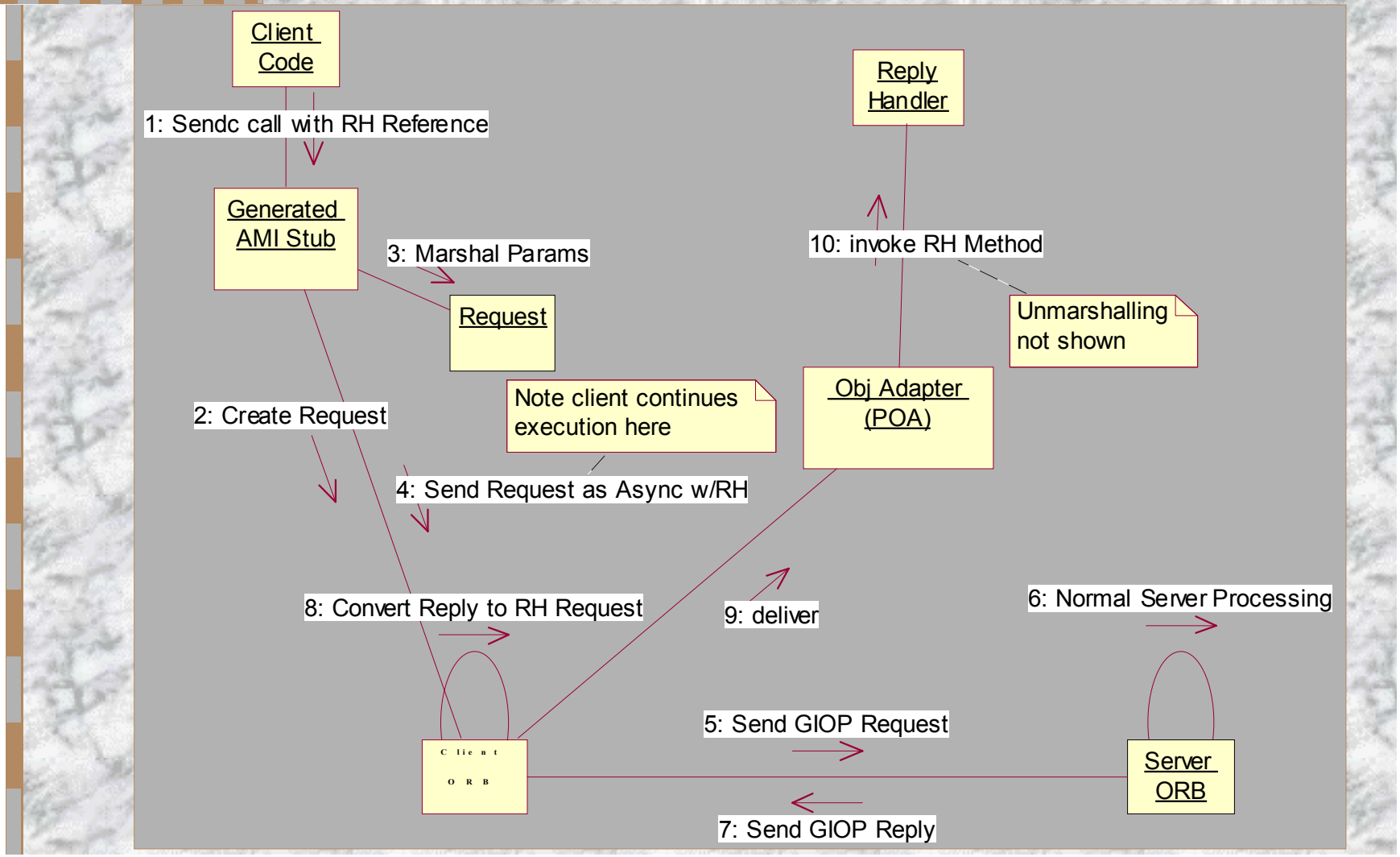
```
interface AMI_AHandler {
   f(in int retVal, in float b, in float c);
   f_excep(in AMI_AExceptionHolder holder);
};
```

AMI Comparison: AMI Callback Implementation

- Java Callback Implementation

```
public class AHandler_Impl extends
   POA_AMI_AReplyHandler {
    public void f(int retVal, float b, float c){
      //use values
    };
    public void f_excep(AMI_AExceptionHolder holder)
    {
       //handle exception on f() call
       System.err.println("It Failed: " + holder);
    };
};
```

AMI Comparison: Async Callback (new)

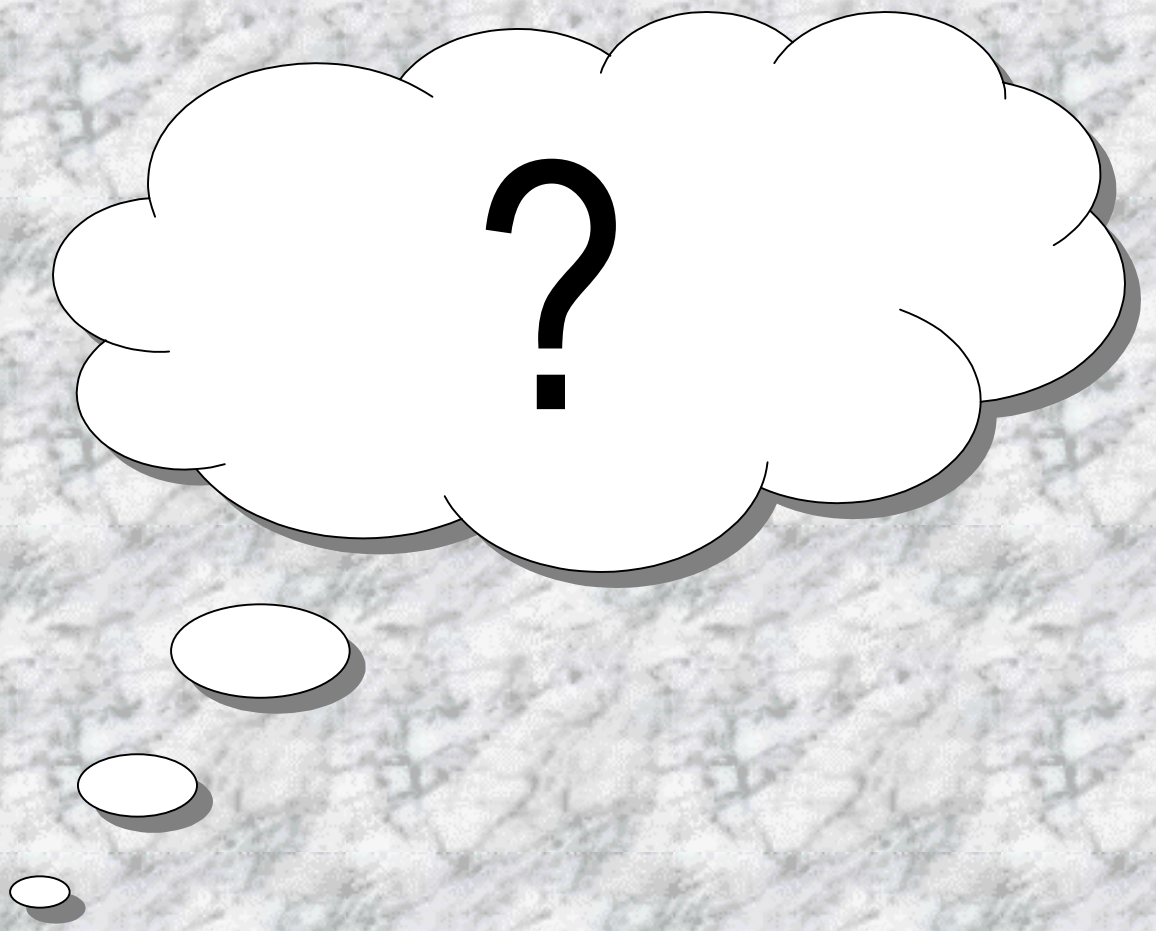


Results of AMI?

- Part One Complete: Application Code now uses decoupled requests and replies
- Users of Deferred Sync now have a typed interface (actually two – polling/callback)
- Callbacks can be used on existing servers (a big plus!) without IDL change
- Ready to decouple the wire protocol via Routers

The Technical Resource Connection
www.trcinc.com

AMI Questions?



AMI Review

- Part One Complete: Application Code now uses decoupled requests and replies
- Users of Deferred Sync now have a typed interface (actually two – polling/callback)
- Callbacks can be used on existing servers (a big plus!) without IDL change
- Ready to decouple the wire protocol via Routers

TII & Routers – Overview

- TII = Time Independent Invocation
  - Second (Optional) section of Messaging
  - Consists of Interoperable Routing Protocol
  - Defines CORBA Messaging Router Interfaces for MOM Products (or native MOM implementations)
- Even with AMI, connections between Clients and Servers must be maintained due to the RPC nature of CORBA's transport (GIOP)

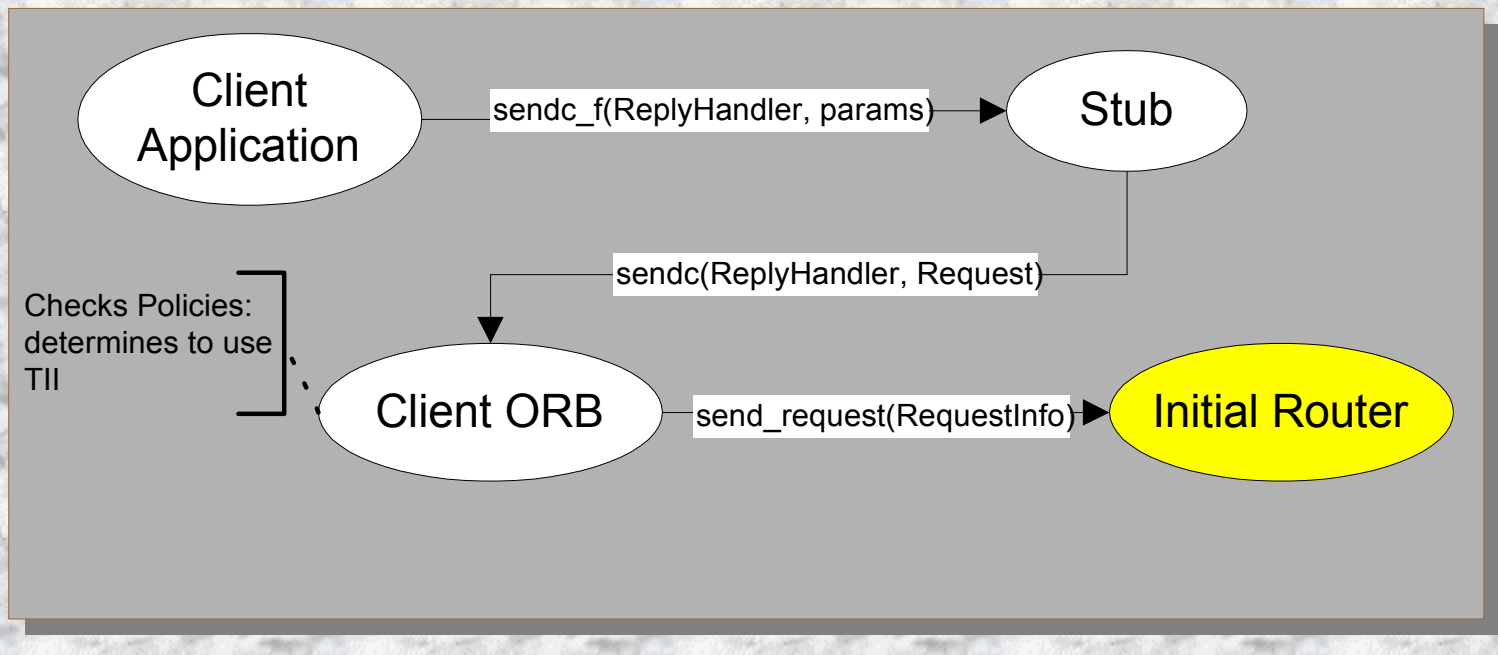
TII – Overview

- Messaging uses Routers to move Requests/Replies as first-class messages
  - Client ORB wraps Request in a Message and sends it to the Initial Router
  - Target Router makes GIOP call on server
  - Reply is routed as a wrapped Request

TII - The Initial Router

(Callback Interface)

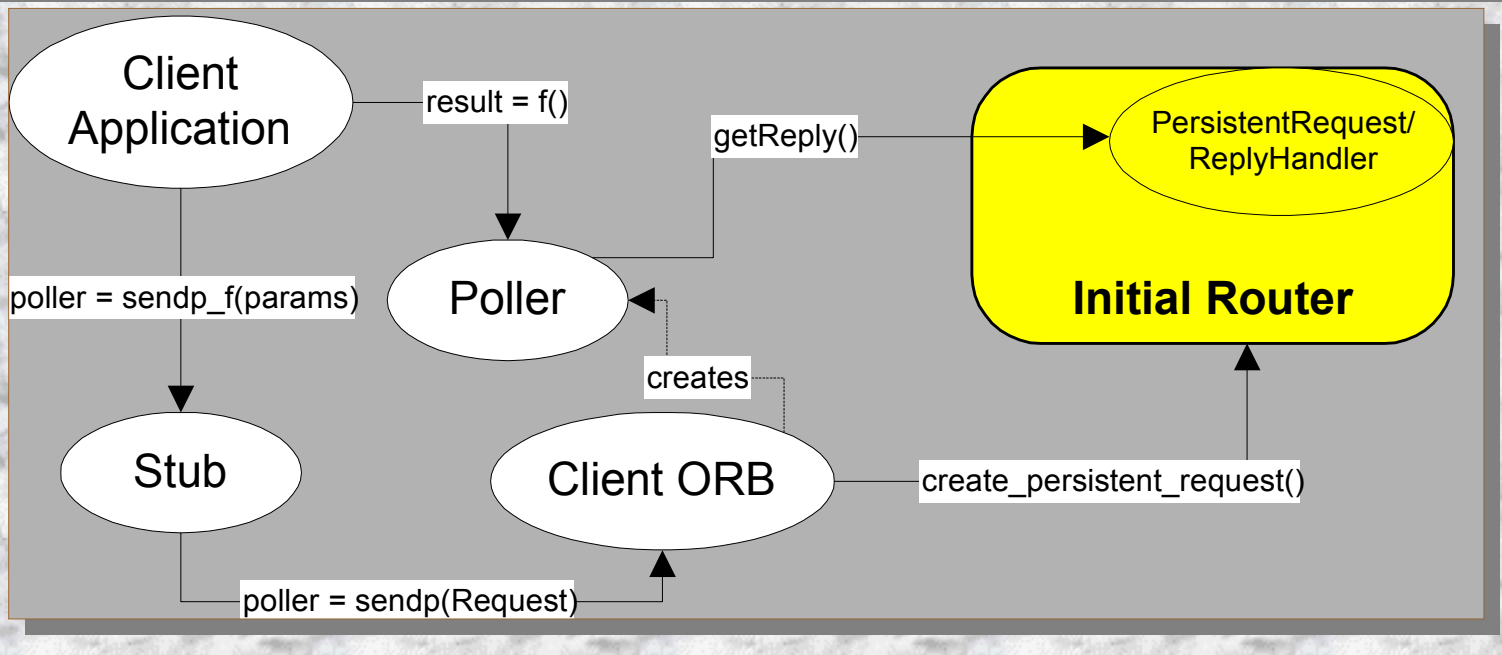
- Initial Routers Accept wrapped Requests
  - Client ORB determines Init. Router and whether to use TII from IOR and Policies
  - Client ORB passes ReplyHandler for return call



TII - The Initial Router

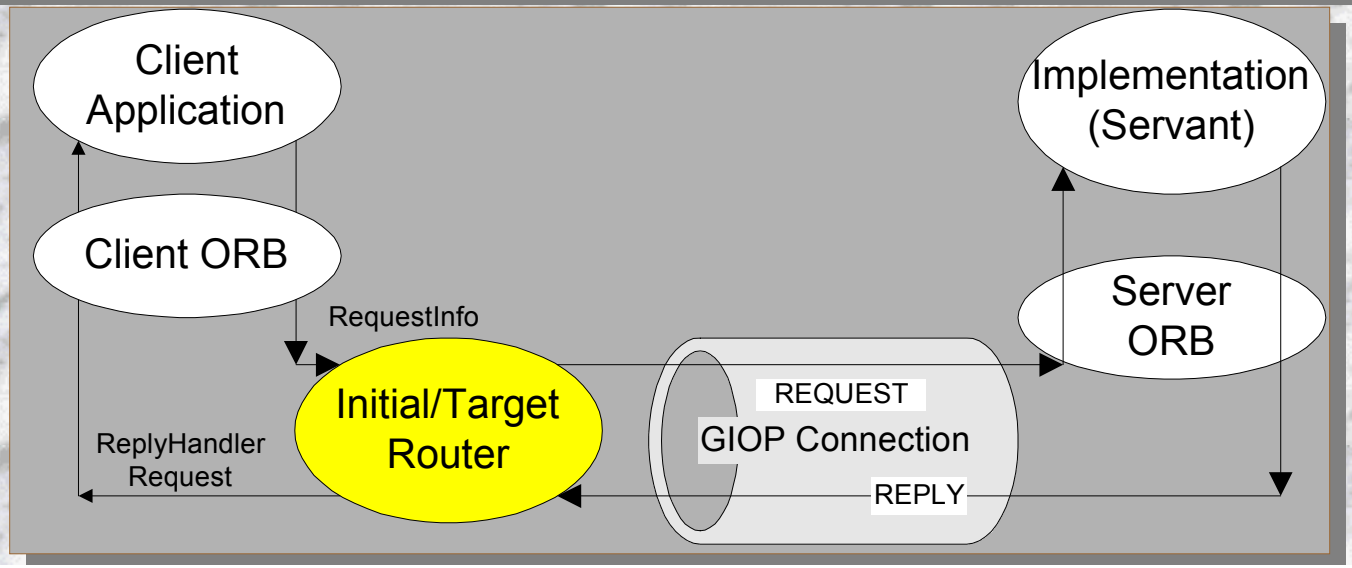
(Polling Interface)

- Initial Routers Hold ReplyHandlers For Pollers
  - Client ORB creates Pollers that query PersistentRequest interface for results
  - Client can disconnect since ReplyHander is in Router



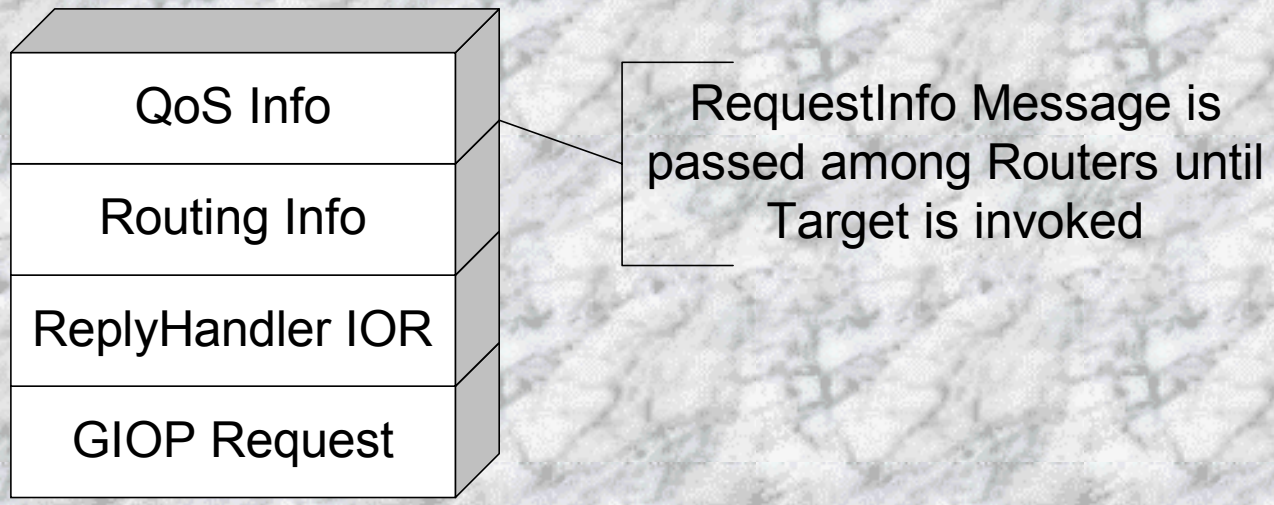
TII: Simplest Case

- One Router is Initial and Target Router
  - Accepts Wrapped Requests
  - Invokes Server
  - Converts Reply to Request
  - Invokes ReplyHandler



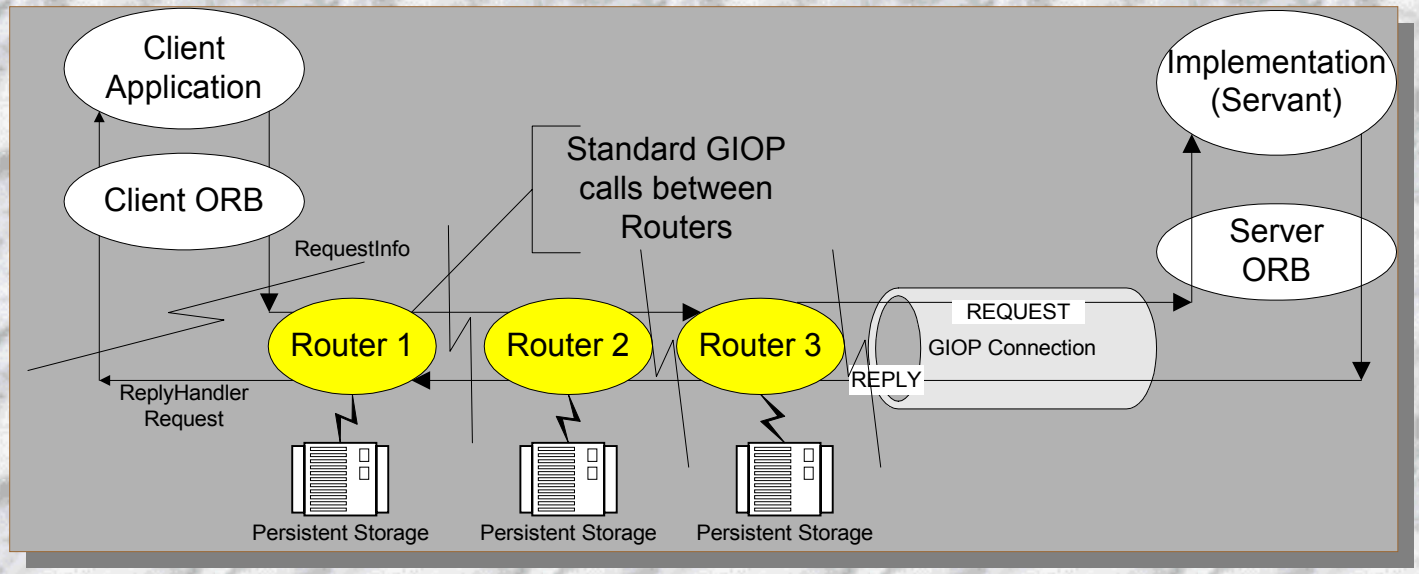
TII – RequestInfo Structs

- RequestInfo structs holds:
  - Original Request Info (needed to invoke the request)
  - QoS info
  - ReplyHandler



TII – Benefits

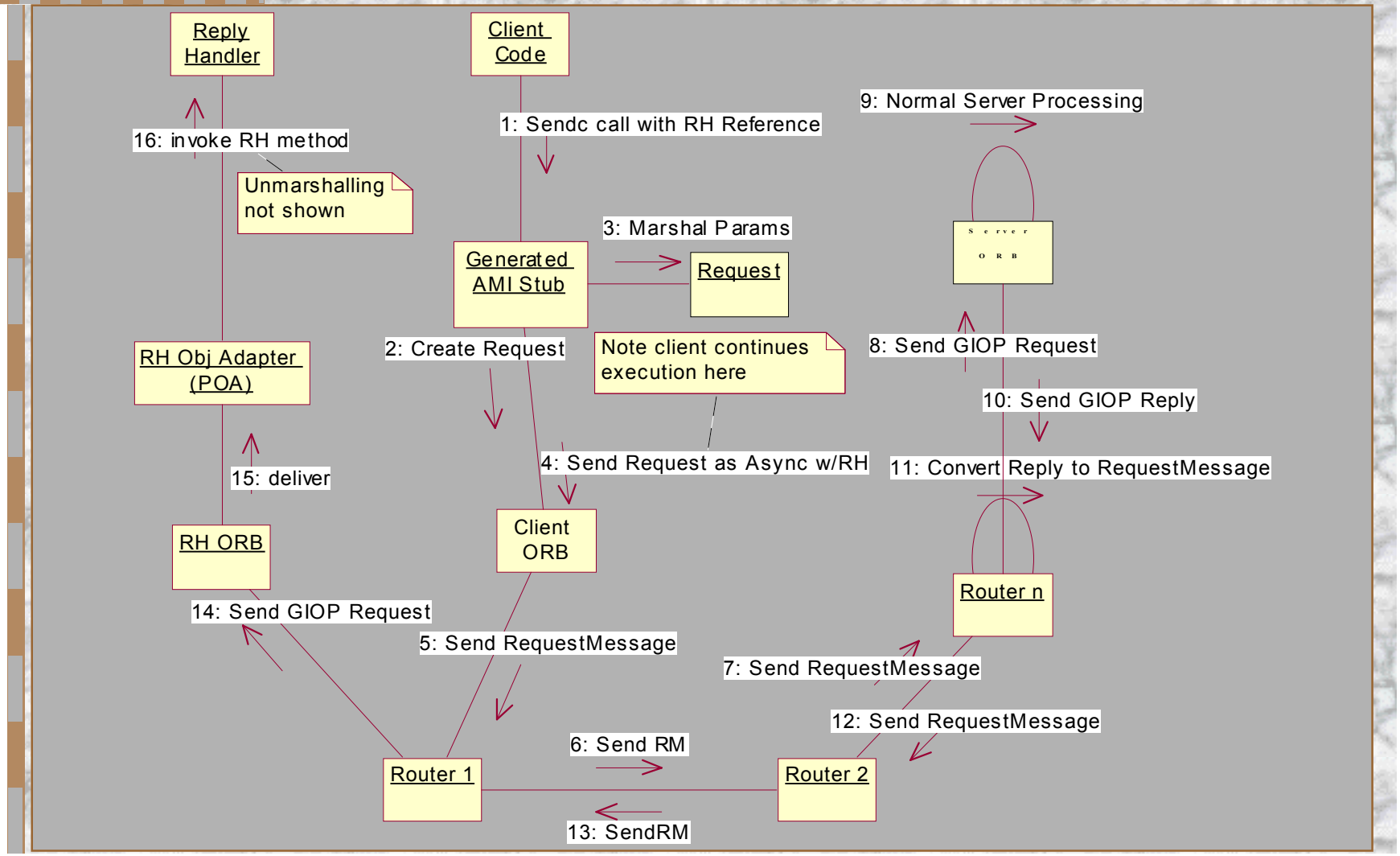
- With Request/Reply as First Class Messages, most MOM QoS can work.
- Administrators can modify routing, etc.



AMI ⇒TII: Changes Needed

- Almost none!
- Only need to set the Routing Policy from ROUTE_NONE to ROUTE_FORWARD or ROUTE_STORE_FORWARD
- Configure Client ORB to use Init. Router
- Optionally set RoutingList hints on server
- No other changes to code are necessary.

TII: TII Call Processing



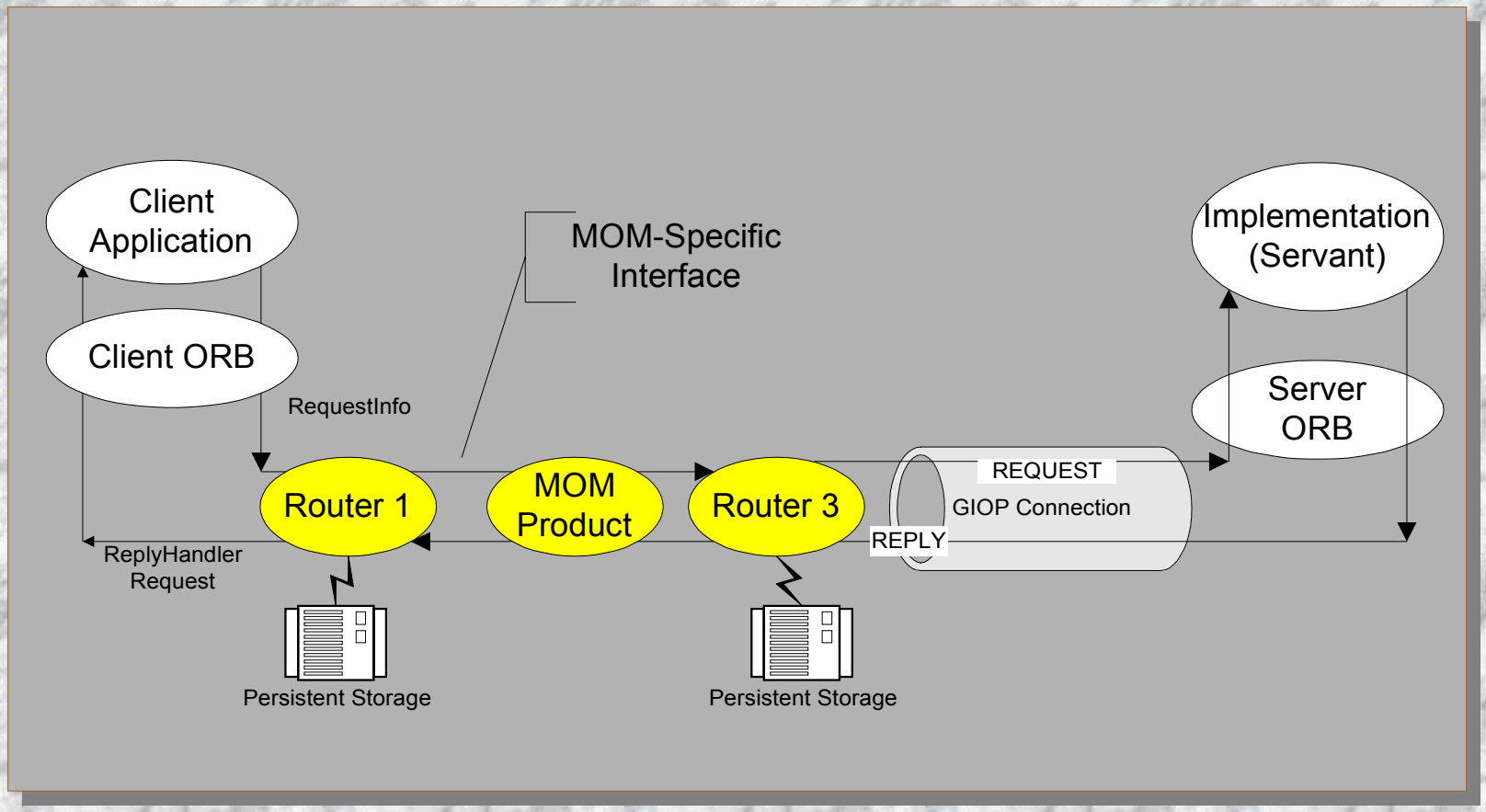
QoS Framework: Overview

- Provides mechanism for application to control the Messaging System
- Uses Policy interface already in place
- Is set at various points:
  - Default – at POA (Server Side)
  - Client ORB (Client Side)
  - Current Thread (Client Side)
  - Object Ref (Client Side)

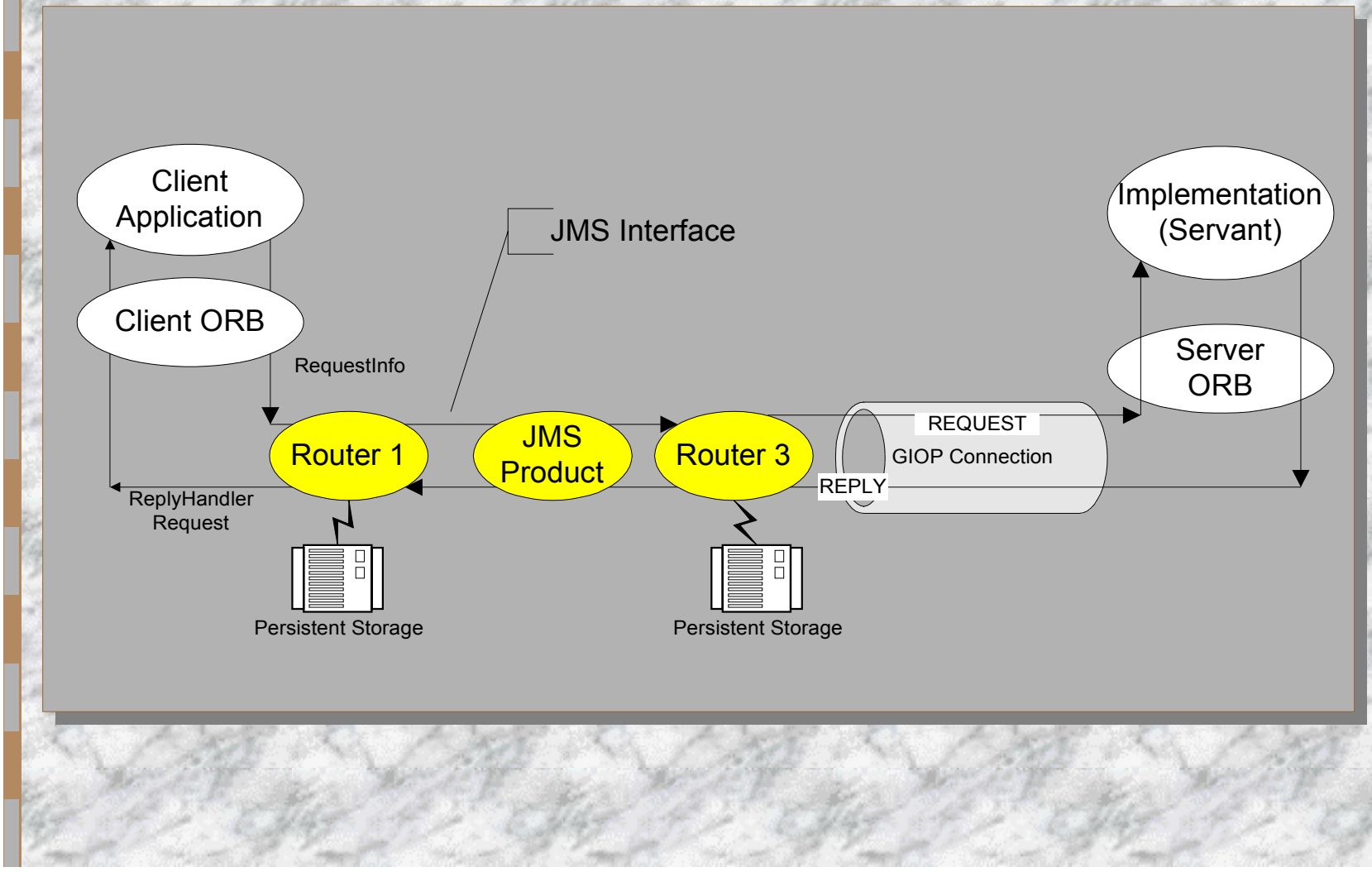
QoS Framework: Messaging QoS Menu

- Priority: Controls the Request Priority while in Queues
- Timing: Controls various timeout aspects
  - *Request Start Time, Timeout*
  - *Reply Start Time, Timeout*
  - *Round-Trip Timeout, etc.*
- Routing: Controls Forwarding or Store/Forwarding
- Max Hops: Sets max hops before request dies
- Ordering: Sets priority/temporal/"deadline" sorting in queue

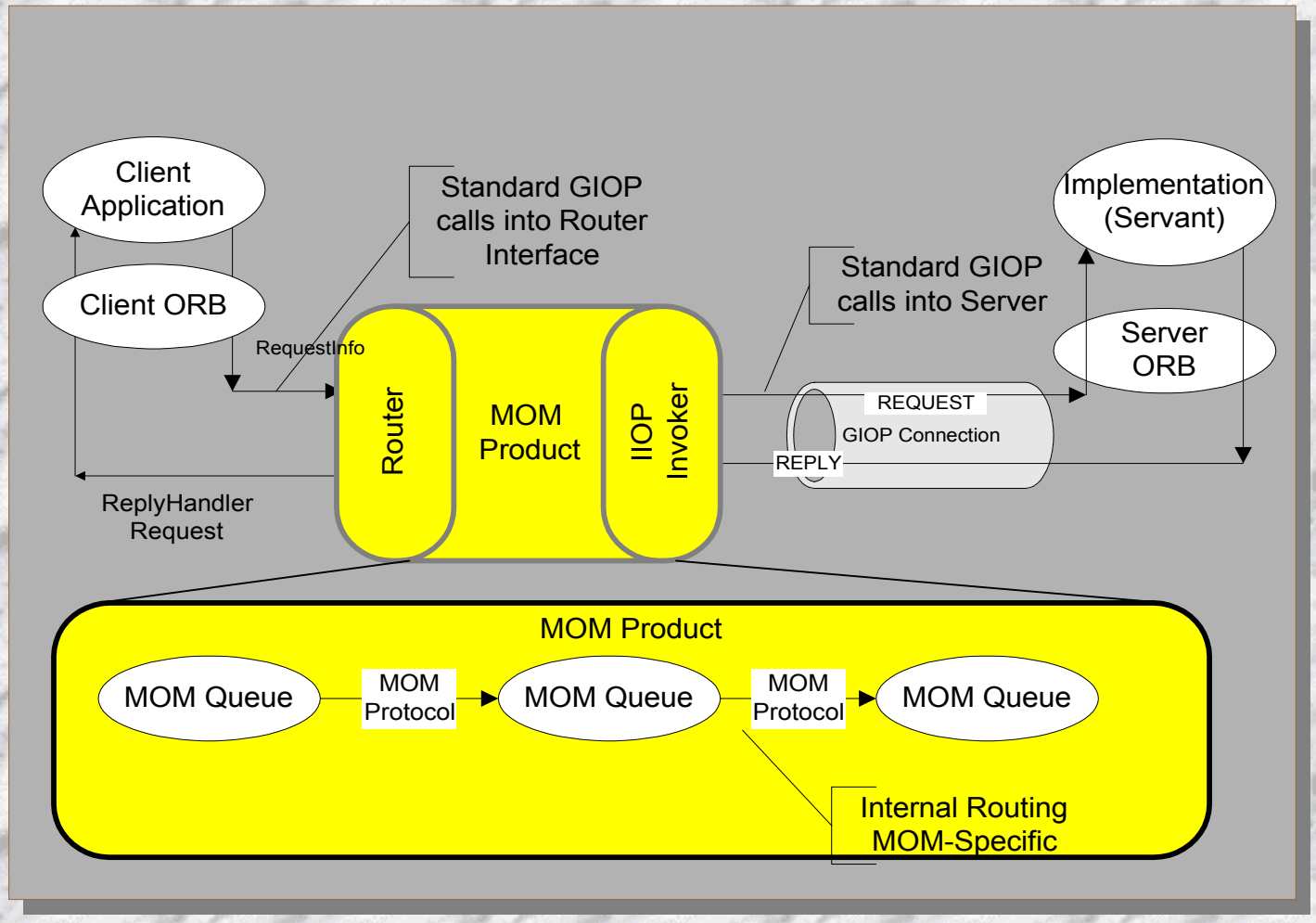
MOM Integration: Wrapping MOM Product



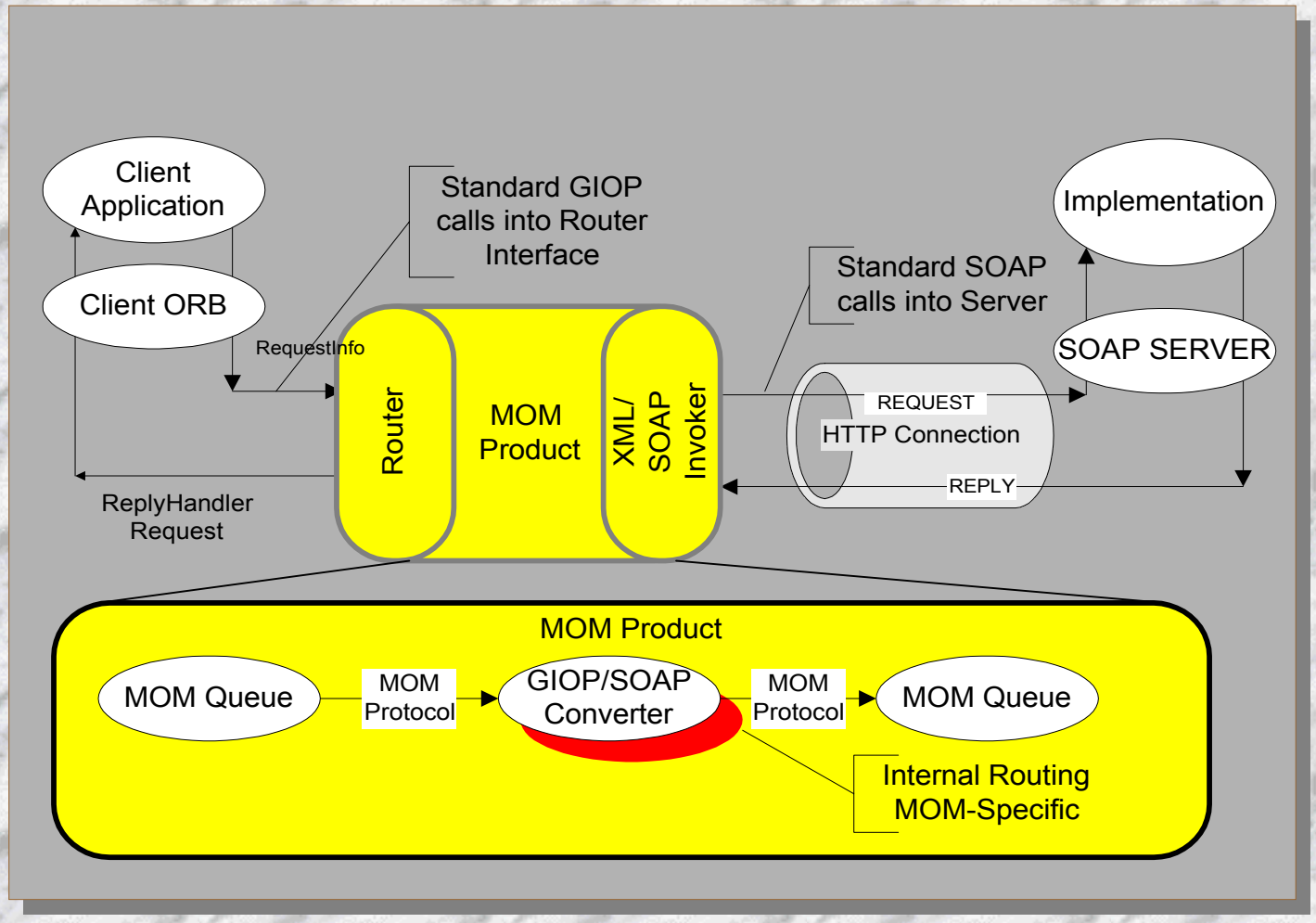
MOM Integration: Wrapping JMS Product



MOM Integration: MOM Supports TII



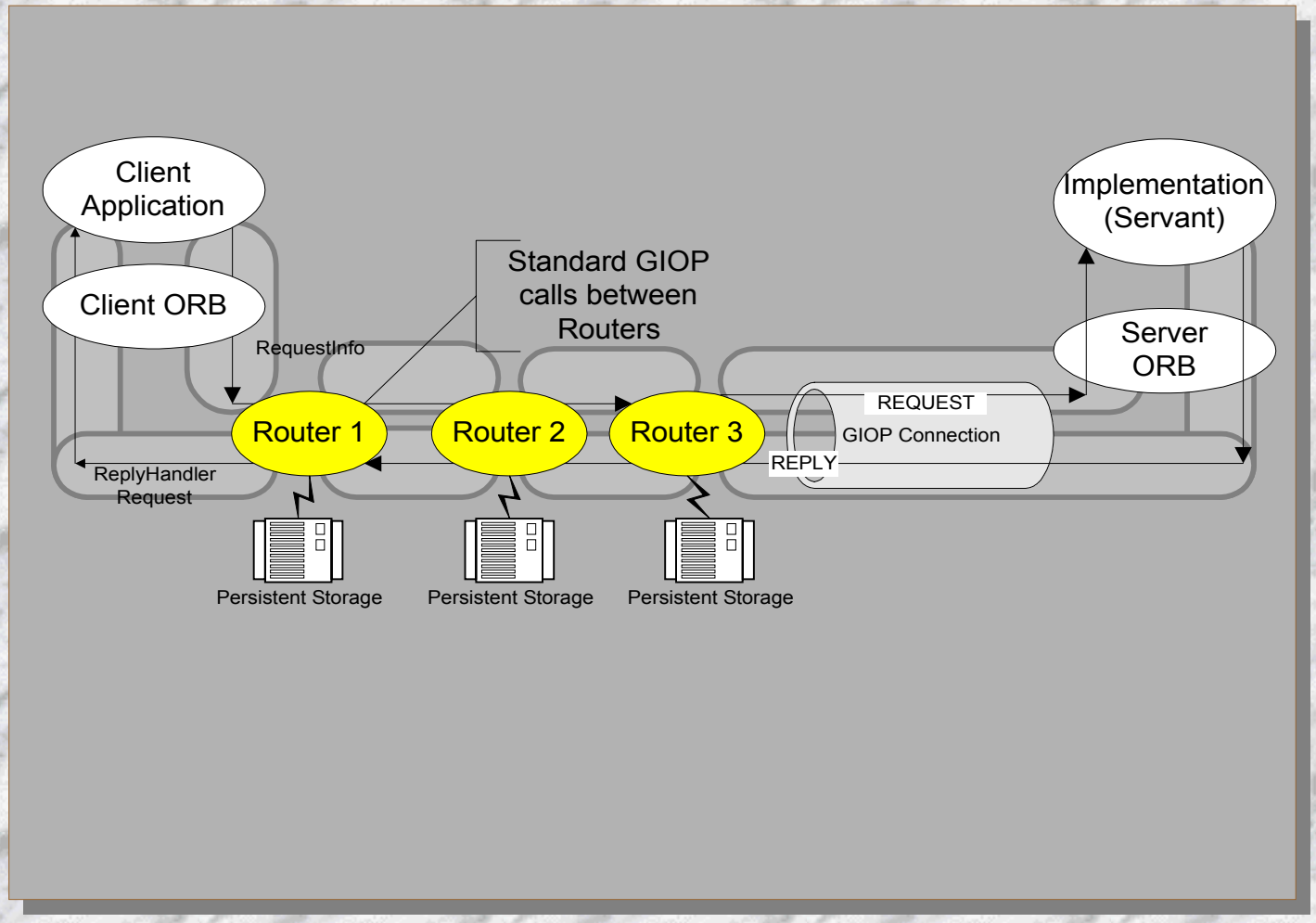
MOM Integration: MOM Bridges TII



Misc Issues

- AMI does not effect OTS: It works fine
- TII inherently breaks OTS
- Disconnected client cannot participate in 2PC
- New Transaction Mode: “Unshared”
  - New Transaction “Each Hop”
  - Used for STORE_AND_FORWARD and existing servers that “require” a transaction context
  - OTS-aware Servers can say whether they will participate in Unshared Transaction

Unshared Transaction Example



Availability

- Part of CORBA 2.4
- Currently in development by major vendors and third parties
- RTF complete: issues rolled into ORB Revision
  - Specification Doc: orbos/98-05-06
  - RTF Output is available