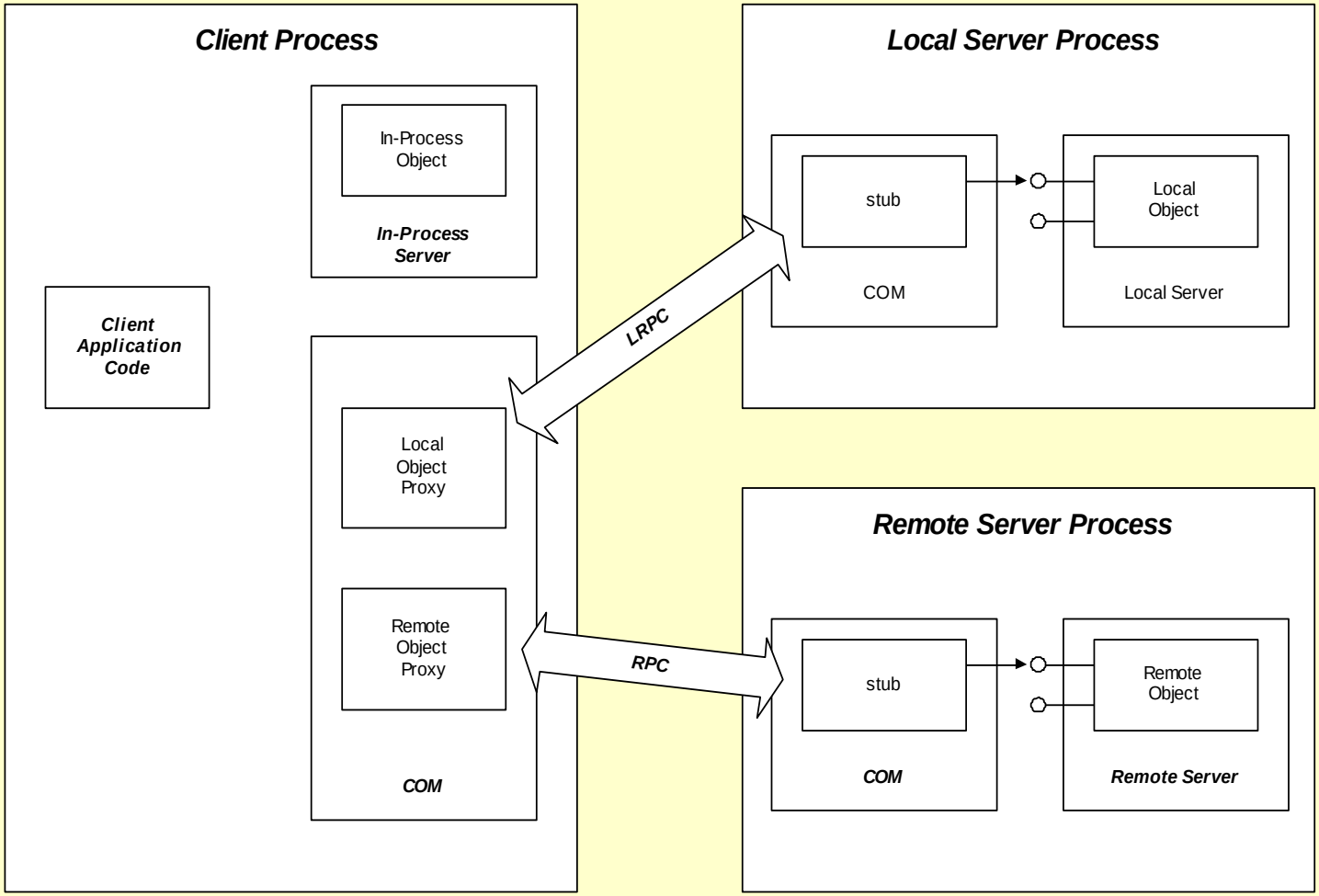
Apartments and COM Threading Models

Jim Fawcett

CSE775 - Distributed Objects

Spring 2008

COM Marshaling Architecture



Apartments

- Apartments currently come in three flavors:
  - Single Threaded Apartments (STAs)
    - COM serializes all out of apartment calls on an STA through a windows message loop. The apartment's single thread services all method invocations by taking requests off the message queue.
    - This means that a component that is not thread safe can safely operate in a Win32 multithreaded environment as long as it was created by the thread in an STA.
  - Multithreaded Apartment (MTA)
    - COM provides no serialization in an MTA. Any component created in an MTA is expected to provide its own synchronization and thus be thread safe.
  - Neutral Threaded Apartment (NTA)
    - Any thread may leave an STA or MTA to access an NTA. NTAs must be fully synchronized.

Apartment Rules

- A process may have many STAs but only one MTA.
- Every COM object belongs to exactly one apartment.
- A thread executes in exactly one apartment at a time. When a thread enters an apartment COM marks it with the apartment ID.
- Objects may be accessed directly only by threads executing in the apartment of the object.
- Objects in an STA will always be accessed by the same thread - the one that created the STA. Thus objects can never run concurrently in an STA

Creating Apartments

- An STA is created when client or EXE-based component calls:

  CoInitialize(NULL)
  or
  CoInitializeEx(NULL,COINIT_APARTMENTTHREADED)

- An MTA is created when client or EXE-based server calls:

  CoInitializeEx(NULL, COINIT_MULTITHREADED)

  for the first time. Subsequent calls by new threads in the same process result in those threads joining the MTA.

Joining Apartments

- An in-proc component with no threading model announced in registry is loaded into a client's main (first) STA if one exists. Otherwise COM creates a host STA for the component.
- An in-proc component with ThreadingModel=Apartment registry entry is loaded into any client STA that instantiates component.
- An in-proc component with ThreadingModel=Free registry entry is loaded into a client's MTA if one exists. Otherwise COM creates a host MTA for the component.
- An in-proc component with ThreadingModel=Both registry entry will be loaded into any client apartment that creates it, either STA or MTA.

Access Within and Between Apartments

- All calls within an apartment are direct - no marshaling involved.
  - instances in an STA can only be directly accessed by the single thread of that STA.
  - for an in-proc component, that is the client's STA thread that created the component
  - instances in an MTA can be directly and concurrently accessed by any thread in the apartment

- All calls into components in another apartment are marshaled.
  - between processes on remote machines
  - between two process on the same machine
  - between two apartments in the same process

Comparing Threading Models for in-proc Components

Registry Entry	Client in:	Result
no entry	main STA	direct access; obj in main STA
	any STA	proxy access; obj in main STA
	MTA	proxy access; obj in main STA
Apartment	main STA	direct access; obj in main STA
	any STA	direct access; obj in calling STA
	MTA	proxy access; obj in new STA
Free	main STA	proxy access; obj in MTA
	any STA	proxy access; obj in MTA
	MTA	direct access; obj in MTA
Both	main STA	direct access; obj in main STA
	any STA	direct access; obj in calling STA
	MTA	direct access; obj in MTA

Marshaling Interface Pointers

- Interface pointers must always be marshaled between apartments.
  - Interface pointers are apartment relative. They can only be used by threads in that apartment.
  - On calls to QueryInterface COM marshals all interface pointers from server to client if they reside in different apartments.
- If a server is in-proc, residing in an STA of the client then no marshaling is required to send the interface pointer to the client on the thread of the STA.

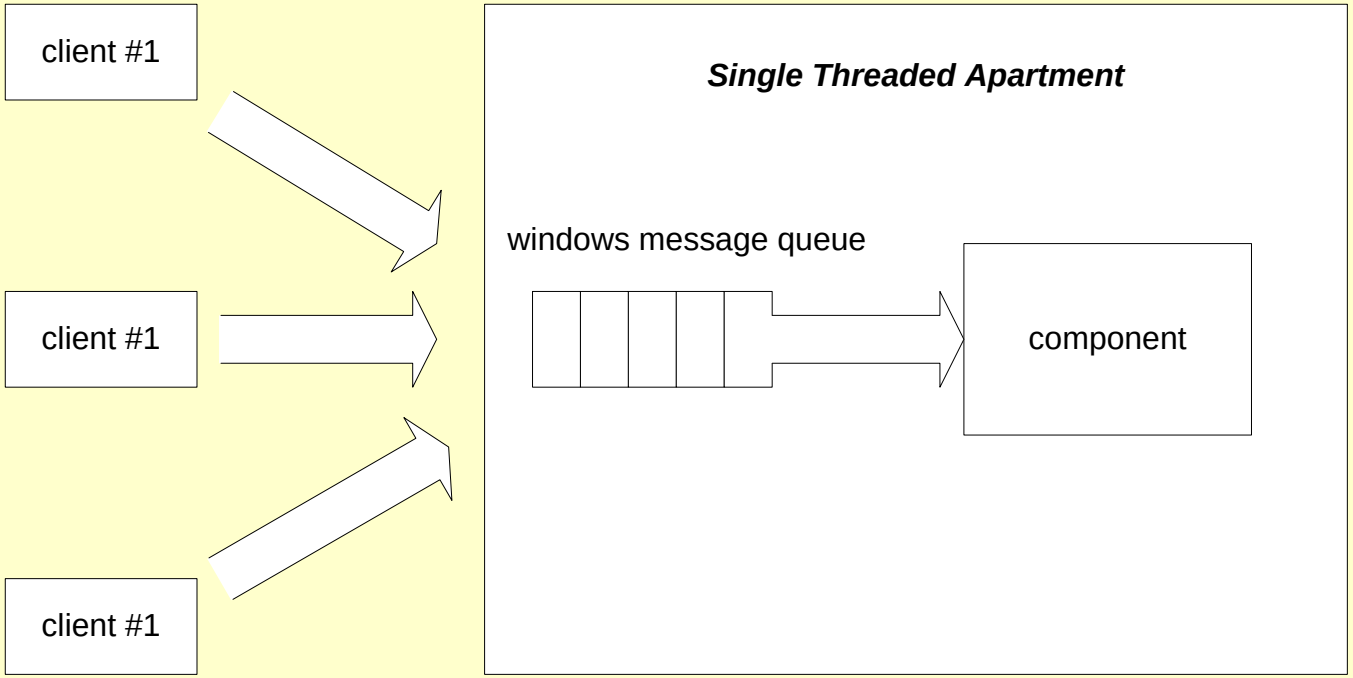
Manual Marshaling of Interface Pointers

- If you need to provide access to a component in one apartment to a thread in another apartment then an interface pointer must be marshaled to the other apartment.
- The component can marshal a pointer to the client by passing it in a call to an interface.
- If client code needs to transfer the pointer then this is done by marshaling the pointer into a stream using:
  CoMarshalInterThreadInterfaceInStream.
  The resulting IStream pointer (not the interface pointer) can then be passed to another apartment in the same process through a static member of some class. Finally the receiving thread unmarshals the pointer using:
  CoGetInterfaceAndReleaseStream
  in a form usable by that apartment.

Single Threaded Apartments (STAs)

- An STA is created when a thread calls CoInitialize(NULL) or CoInitializeEx(NULL, COINIT_APARTMENTTHREADED)
- Only one thread may reside in an STA, the thread that made the call to create the STA. That thread must ultimately call CoUninitalize(). The thread is marked with an identifier of the apartment, e.g., OXID.
- a process can have zero, one, or many STAs.
- An STA owns any COM object instantiated by its thread (unless its threading model is none). Only that thread can make method calls on the component.
- During CoInitialize(EX) creating an STA in an apartment separate from the client, COM calls CreateWindow to create a hidden window. All method calls to an STA component, from outside the apartment, are dispatched with that window's message loop. This means that an STA component must include a windows message loop in the code you write.

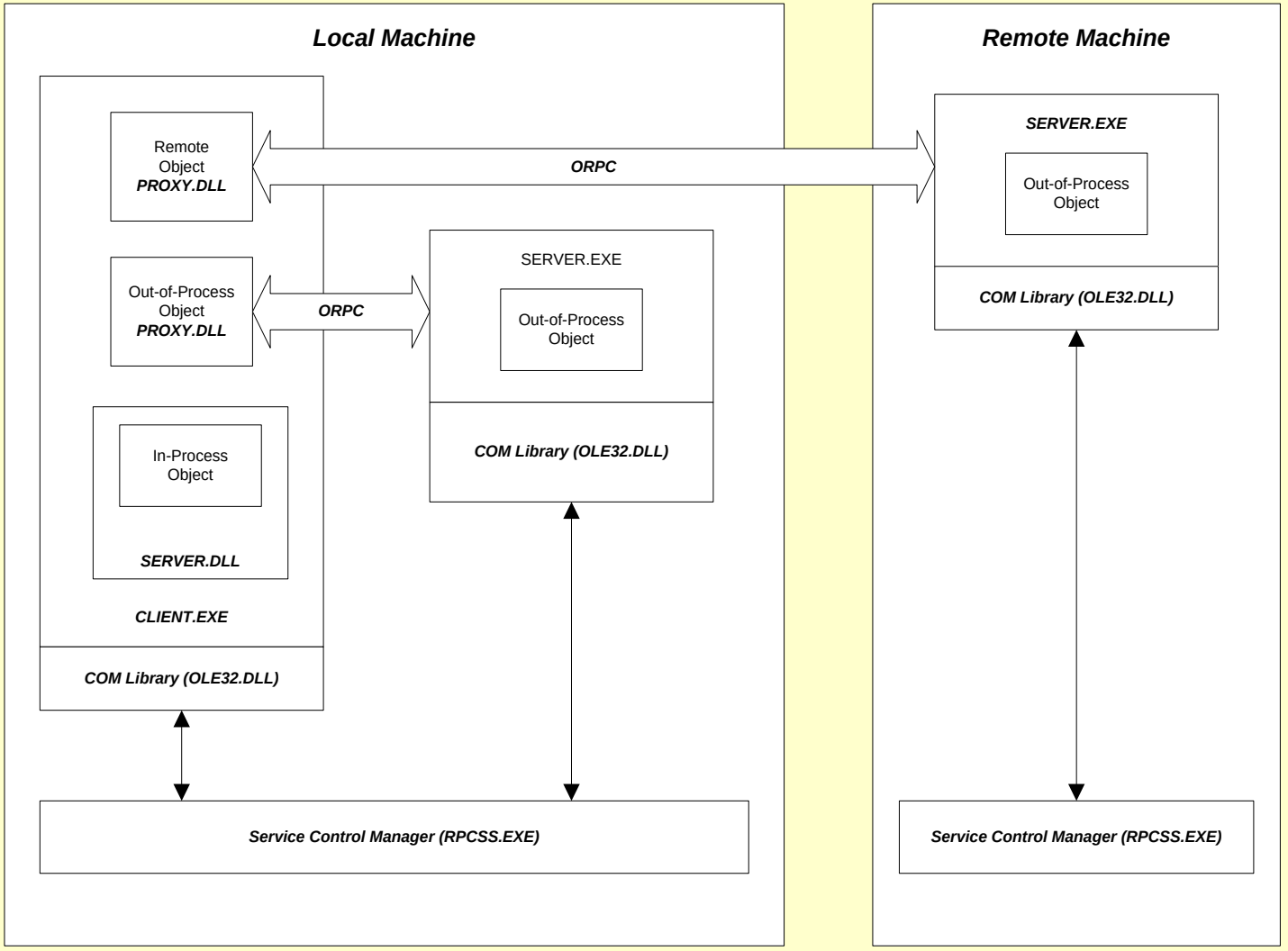
STA Synchronization



Multithreaded Apartments (MTAs)

- An MTA is created when a thread calls CoInitializeEx(NULL, COINIT_MULTITHREADED)
- More than one thread may reside in an MTA
  - the first thread that calls CoInitializeEx creates the apartment.
  - subsequent threads in the same process that call CoInitialize(NULL,COINIT_APARTMENTTHREADED) join the MTA.
  - Each thread is marked with an identifier of the apartment, e.g., OXID.
- a process can have only one MTA.
- An MTA owns any COM object instantiated by any of its threads, provided that the component is threading compatible. Any thread from the MTA can make direct calls to the object.

Component Activation



Component Activation

- Objects are activated by the Service Control Manager as:
  - in-process by loading component server's dll in the address space of the client, making client component interactions very efficient
  - local out-of-process by starting the server's exe in its own process on the same machine
  - remote out-of-process by notifying the remote machine's SCM to activate the server's exe on that machine
  - servers designed as in-proc dlls can also be started as local or remote components using the services of dllhost.exe, a surrogate process into which the server dll is loaded.
  - for all out-of-process components, COM loads a proxy dll into the client's address space and stub into component's address space to handle interprocess and remote communication between them.

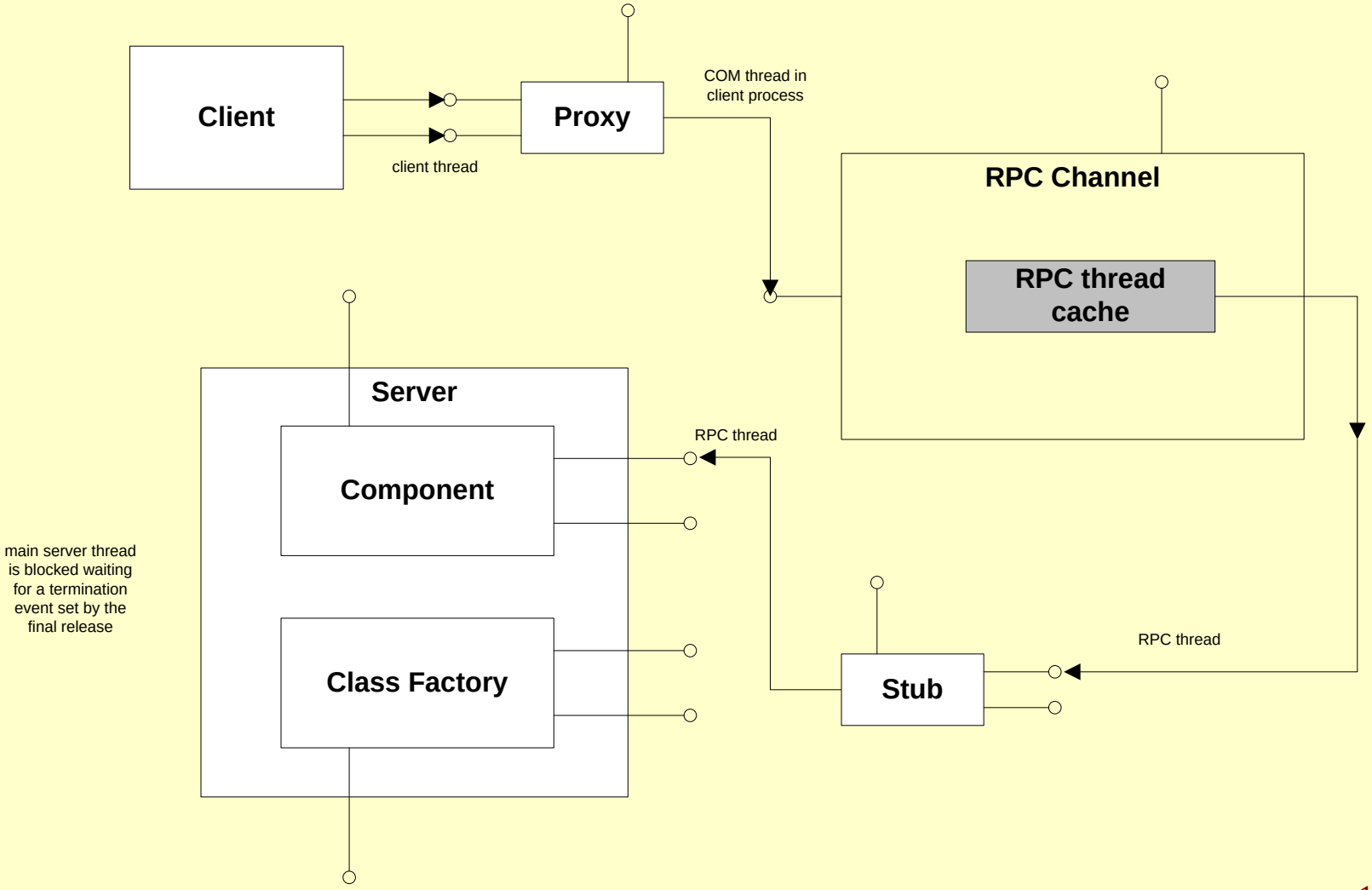
Service Control Manager

- the SCM supports three activation processes:
  - binding to class objects (class factory) using CoGetClassObject
  - binding to class instances using CoCreateInstanceEx
  - binding to persistent instances from files using CoGetInstanceFromFile
- Binding to class objects and instances brings up newly created objects with no state history except as determined by the class and class factory constructors.
- It is possible to create singleton components so that all client activations attach to the single class instance.
- If you need to preserve class state between activations, then using persistent bindings is required.

Multi-Threaded Apartment Component Member Invocations

- When CoInitialized, COM starts RPC service making component an RPC server.
  - as objects are accessed by an off-host client, network protocols are registered with the server and an RPC thread cache is started
  - the first thread from the cache listens for incoming connection requests and dispatches threads to service each request
  - the dispatched thread finds the stub manager and interface stub
  - the thread then enters the component's apartment and calls IRpcStubBuffer::Invoke method on the interface stub and enters component's method
  - in an MTA subsequent threads may access the object concurrently so synchronization of global and local static data is essential

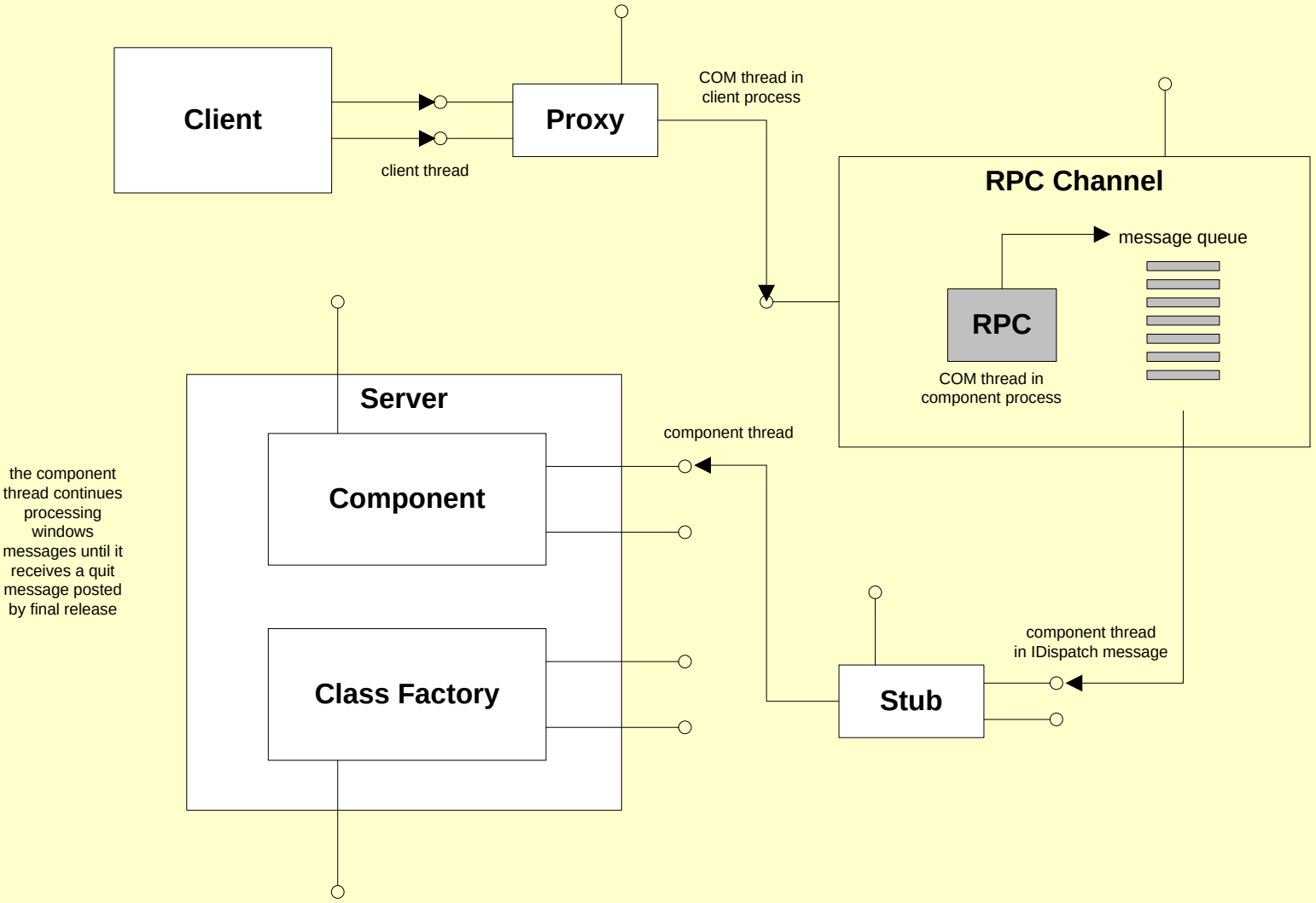
MTA Remoting Architecture



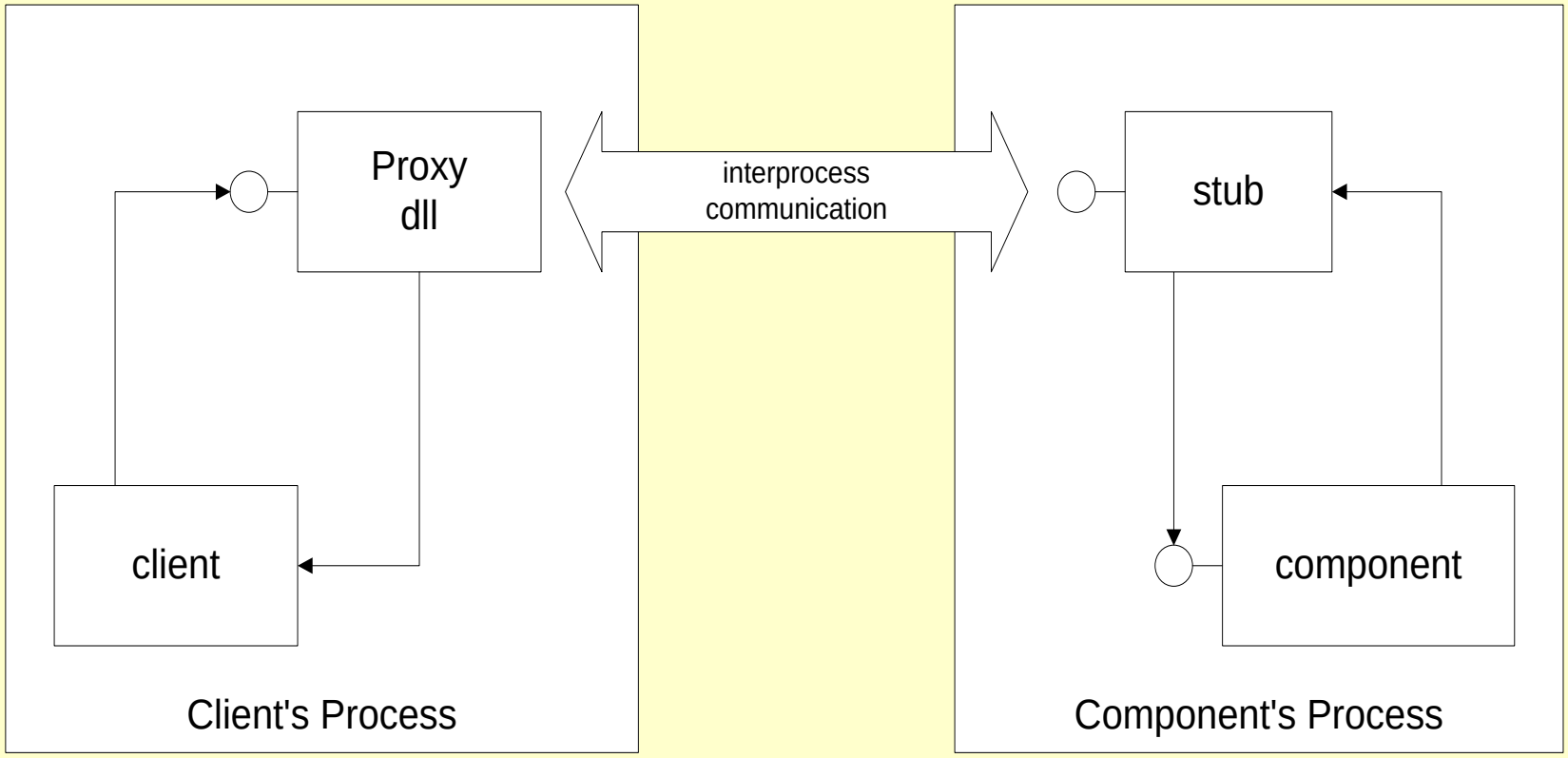
Single Threaded Apartment Component Member Invocations

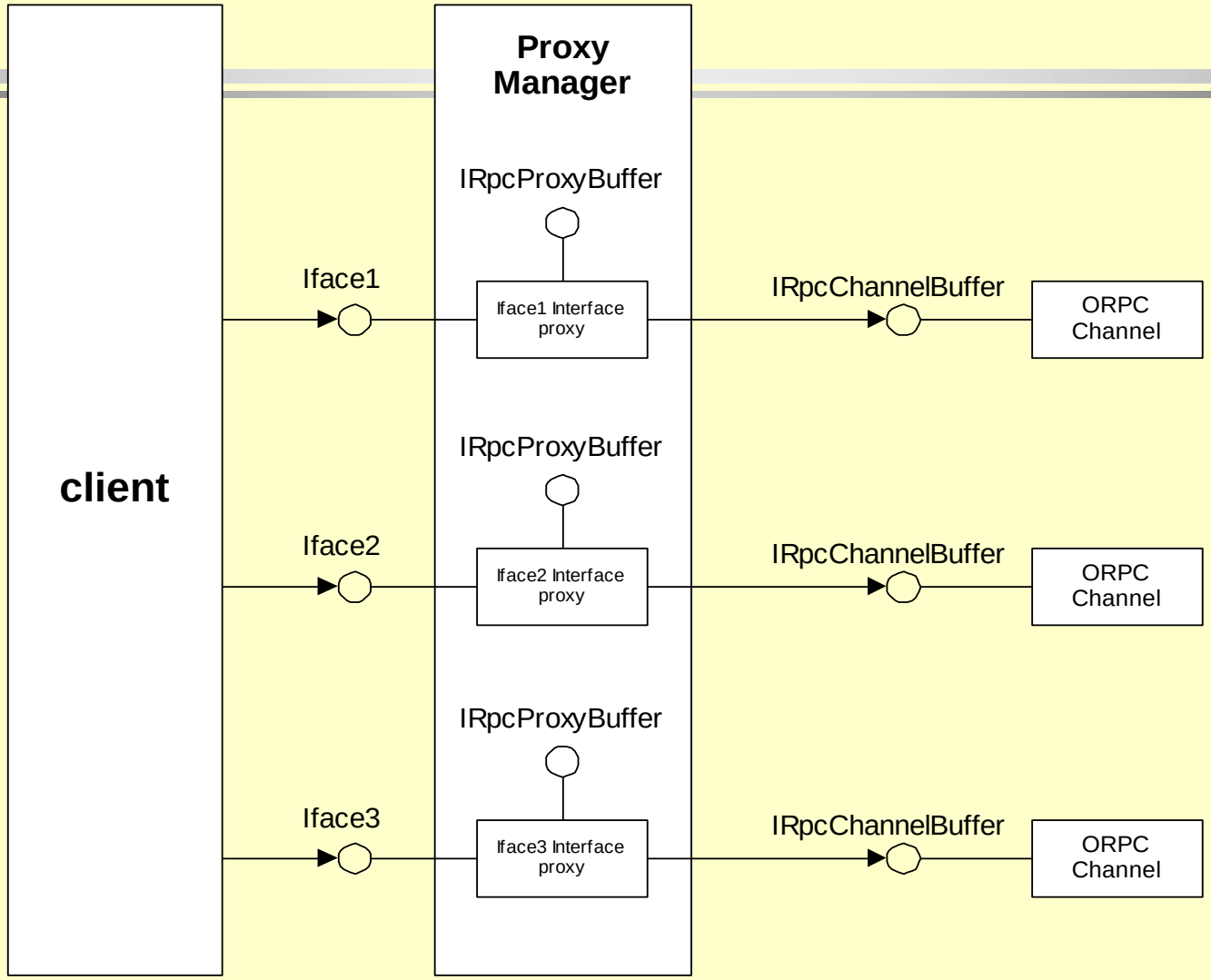
- When CoInitialized, COM starts RPC service making component an RPC server.
  - as objects are accessed by an off-host client, network protocols are registered with the server and an RPC thread cache is started.
  - when an incoming connection request arrives an RPC thread is dispatched.
  - no threads can enter an STA other than the first thread which called CoInitialize, so the RPC thread posts a message to the STA thread's message queue (a hidden window create by COM for the STA thread).
  - STA thread services queue with GetMessage, DispatchMessage.
  - Since all invocations are on the STA thread, all calls are serialized through the windows message queue.

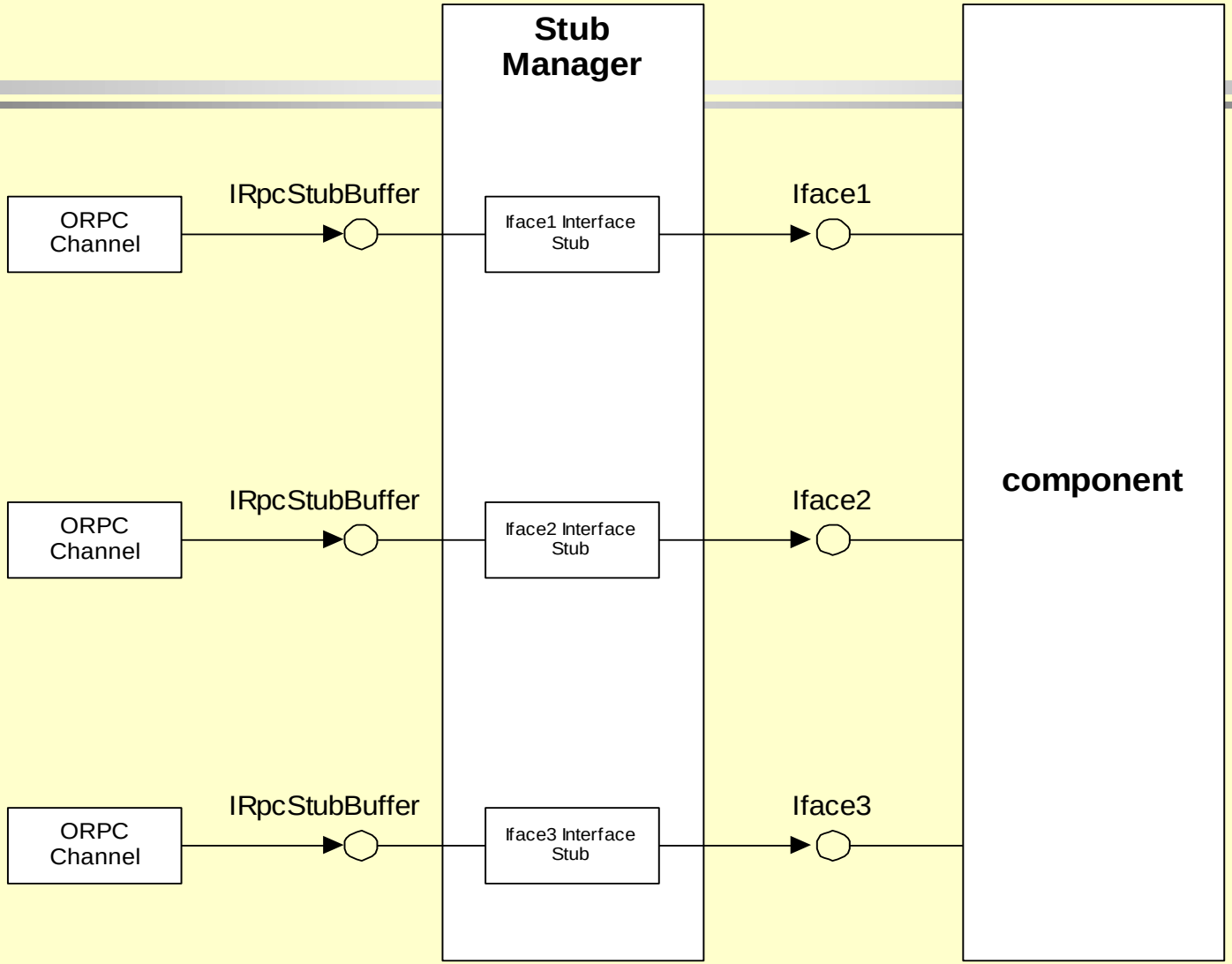
STA Remoting Architecture

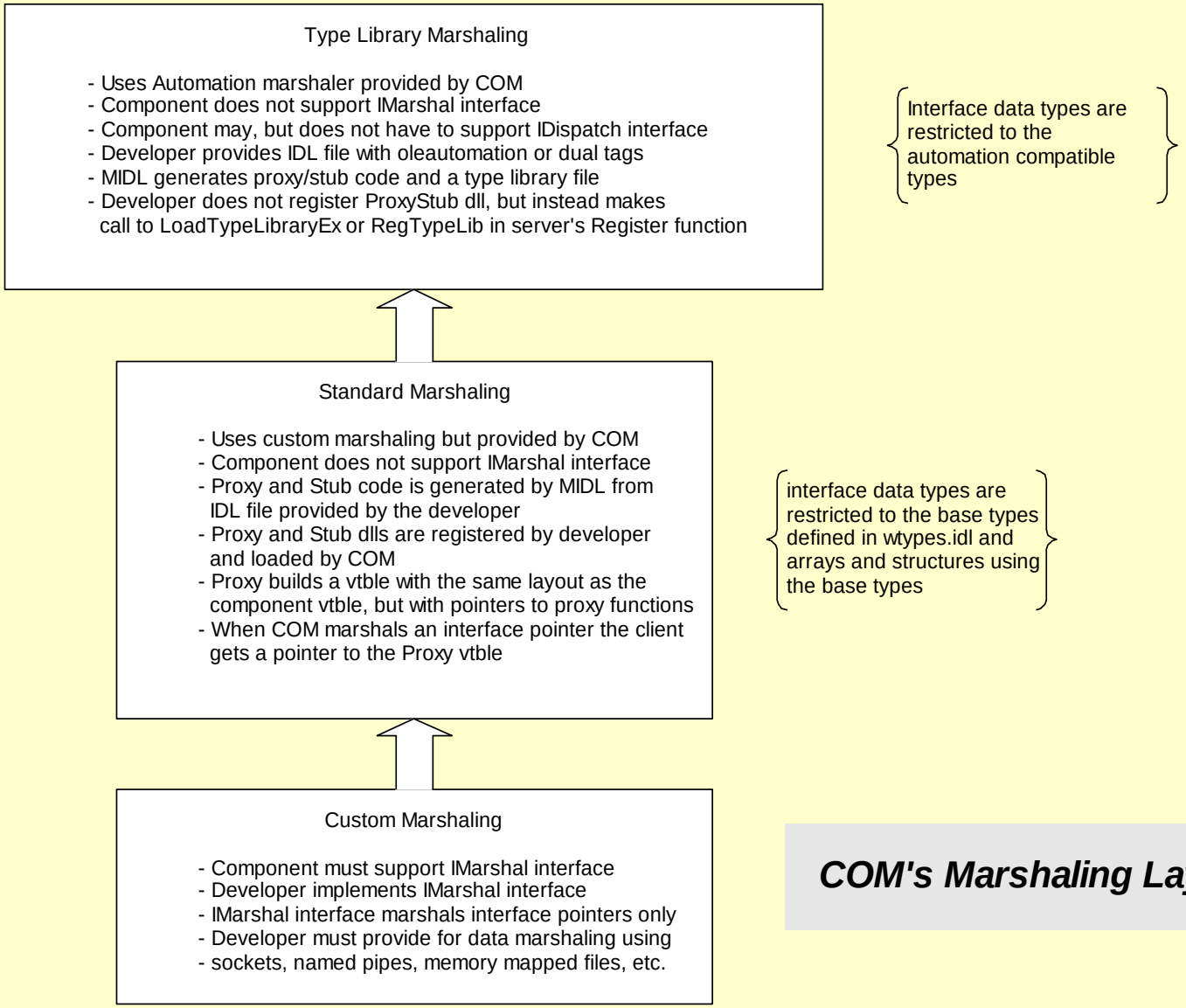


Marshaling Architecture




client
Proxy Manager
IRpcProxyBuffer
Iface1
Iface1 Interface proxy
IRpcChannelBuffer
ORPC Channel
IRpcProxyBuffer
Iface2
Iface2 Interface proxy
IRpcChannelBuffer
ORPC Channel
IRpcProxyBuffer
Iface3
Iface3 Interface proxy
IRpcChannelBuffer
ORPC Channel


Stub
Manager
ORPC
Channel
IRpcStubBuffer
Iface1 Interface
Stub
Iface1
ORPC
Channel
IRpcStubBuffer
Iface2 Interface
Stub
Iface2
component
ORPC
Channel
IRpcStubBuffer
Iface3 Interface
Stub
Iface3



COM's Marshaling Layers