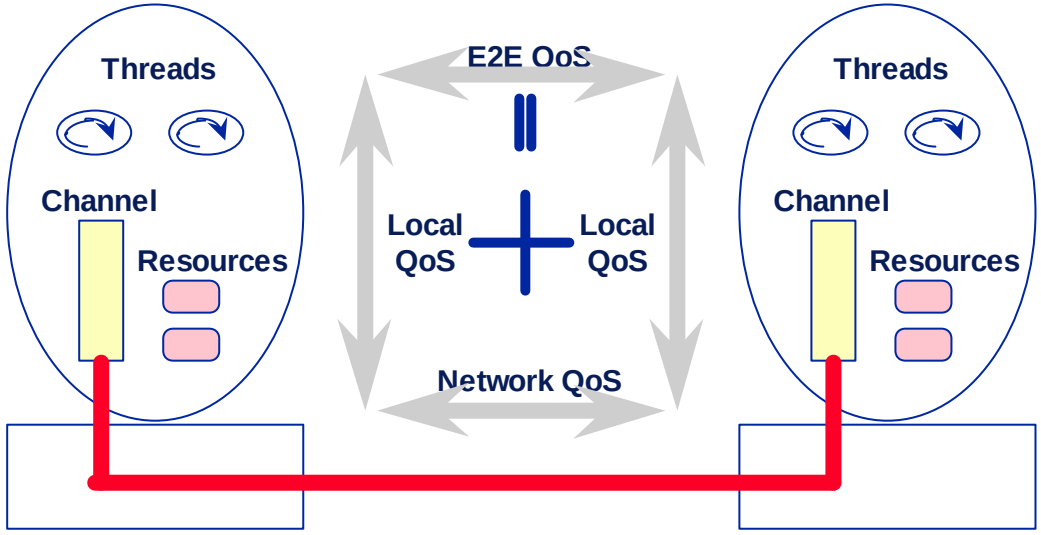
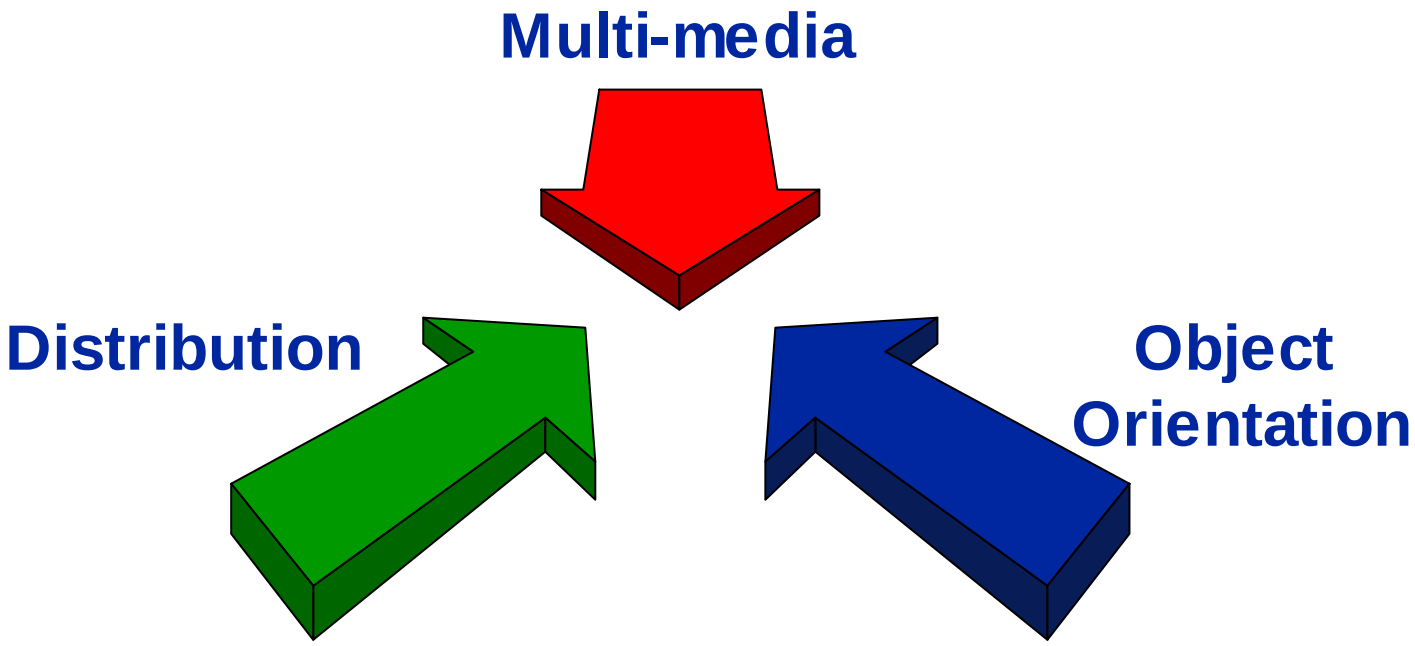
DIMMA Report



Matthew Faupel

Matthew.Faupel@ansa.co.uk

Converging Trends



Multi-Media ORB

- Object Request Brokers give distributed object orientation

BUT

- Multi-media support creates additional requirements

ORB Requirements

- Support for specifying flow interfaces
- Control over resources used
- New protocols easily added
- Minimum necessary footprint

DIMMA Goals

- Microkernel (component based) ORB for applications
- ... requiring “soft” real-time QoS and MM flows
- Control over resource sharing
  - explicit binding allowing QoS to be specified
  - ... plus supporting engineering mechanisms
- Flexibility
  - Support a wide range of QoS policies
  - Lightweight <-> full function instantiations
  - Simple to add new protocols
  - Support a variety of programming “personalities”

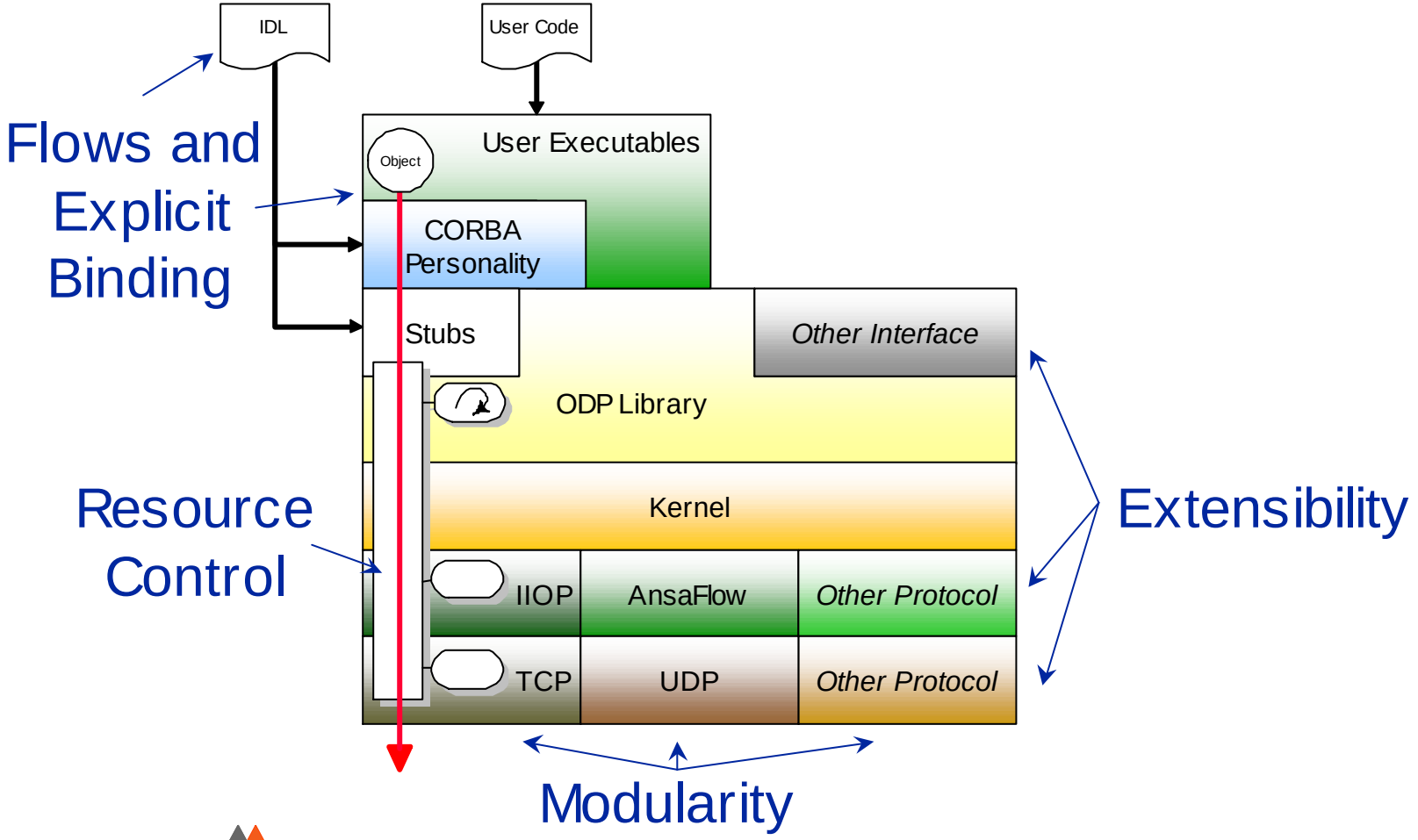
Results

- DIMMA 1.0 (Nov 96)
  - microkernel ORB with MM extensions
  - JET (CORBA) & ODP programming personalities
  - flow interfaces
  - application multi-tasking
  - IIOP & ANSA Flow protocol (multicast UDP)
- DIMMA 1.1 (Apr 97)
  - Configurable tracing
  - Improved JET <-> CORBA compliance
  - Restructured source and build system

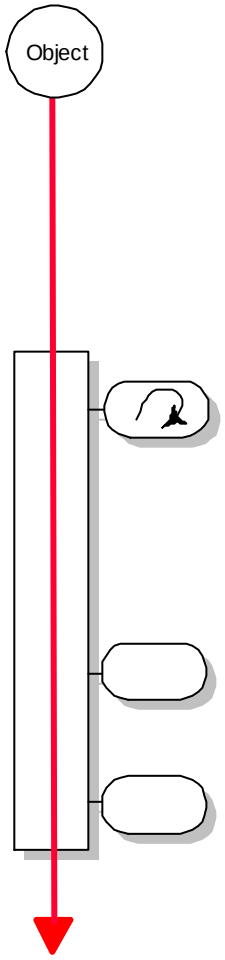
Results

- DIMMA 2.0 (May 97)
  - QoS controlled explict binding
  - populated resource control framework
  - improved protocol framework
  - added resource control to IIOP and ANSA Flow
  - dynamic protocol loading
- DIMMA 2.01 (Sep 97)
  - improved performance and robustness
  - protocol independent QoS specification
  - implemented as C++ code for Solaris 2.5

DIMMA Features



Resource control

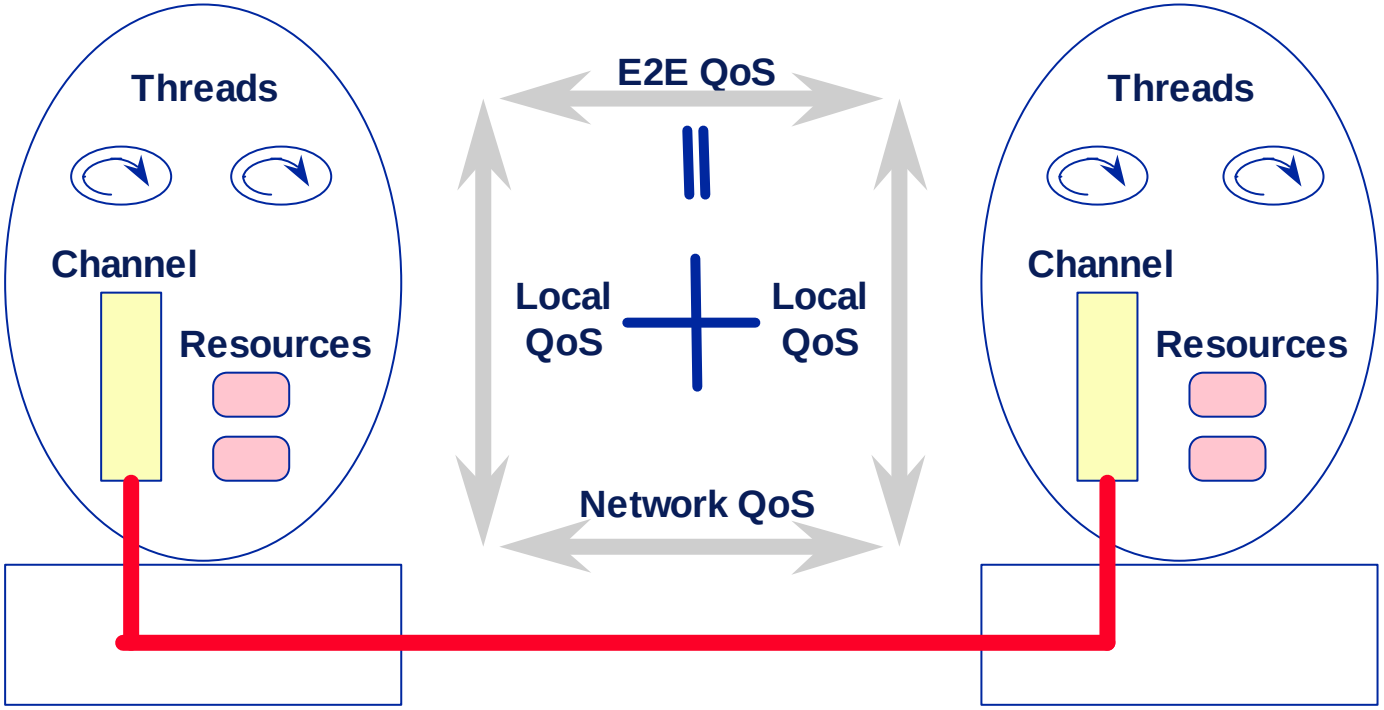


Why Resource Control?

- Real-time and multi-media need specific QoS
  - required end to end at the application
  - must be maintained during varying load
- Implies resources available when needed
- But resources often scarce and hence shared

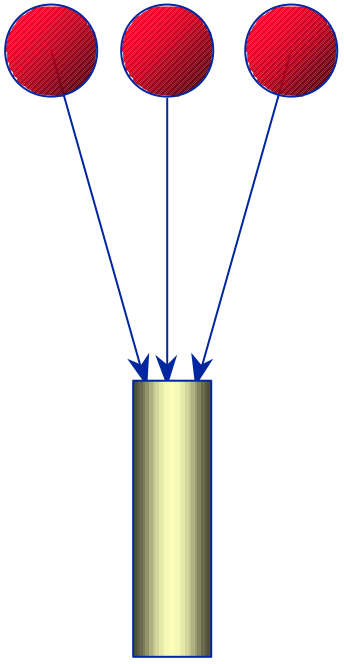
=> Sharing must be controlled

Which Resources?

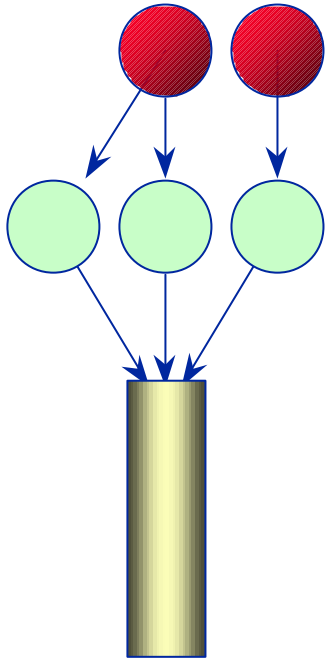


Channel Multiplexing

Objects sharing channel



Plus multiple Invocations

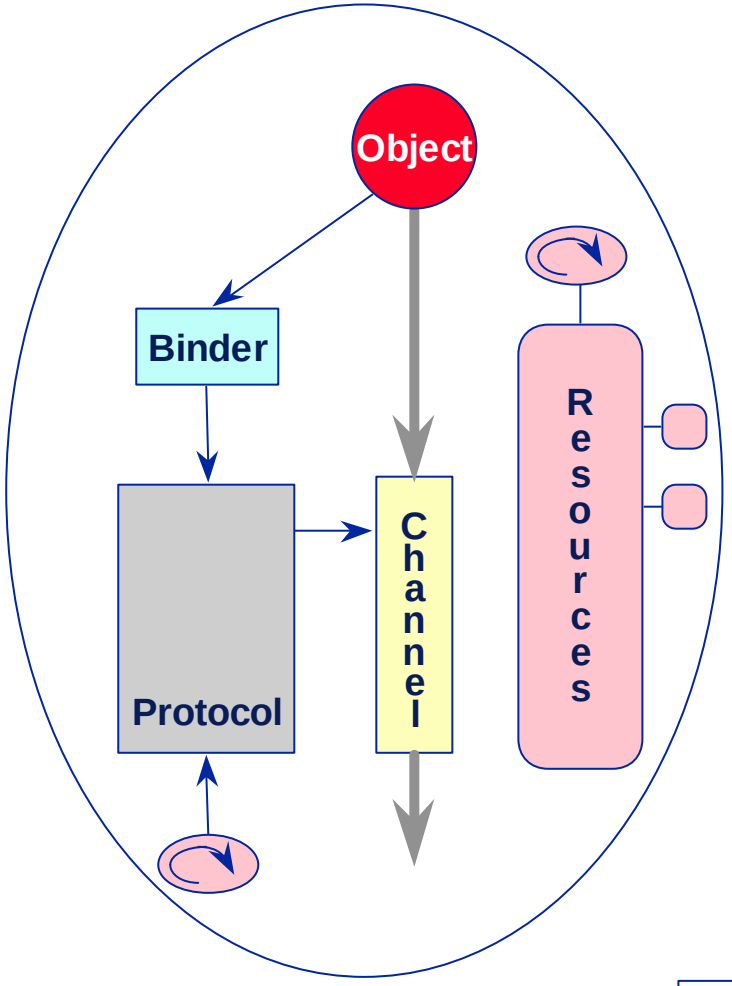


CPU Multiplexing

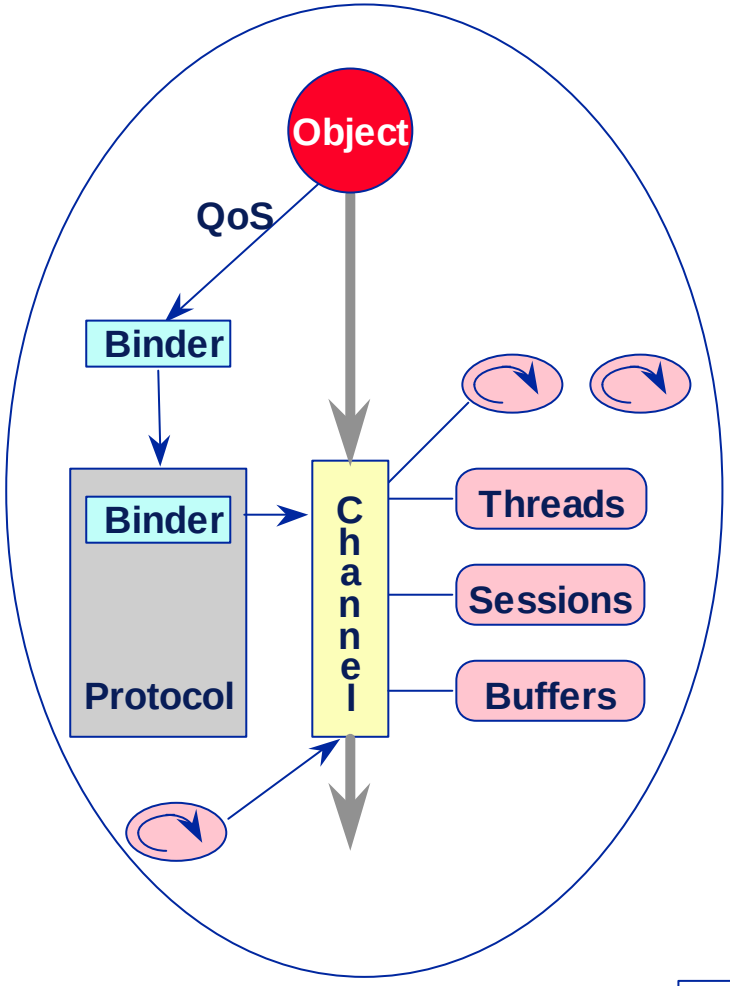
Invocation concurrency

Shared read/demux Thread

Vanilla ORB Capsule



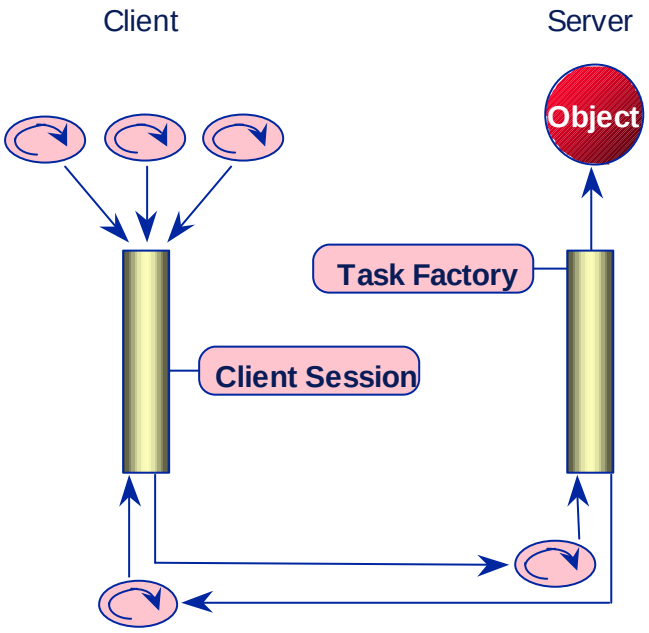
Resourcing the Channel



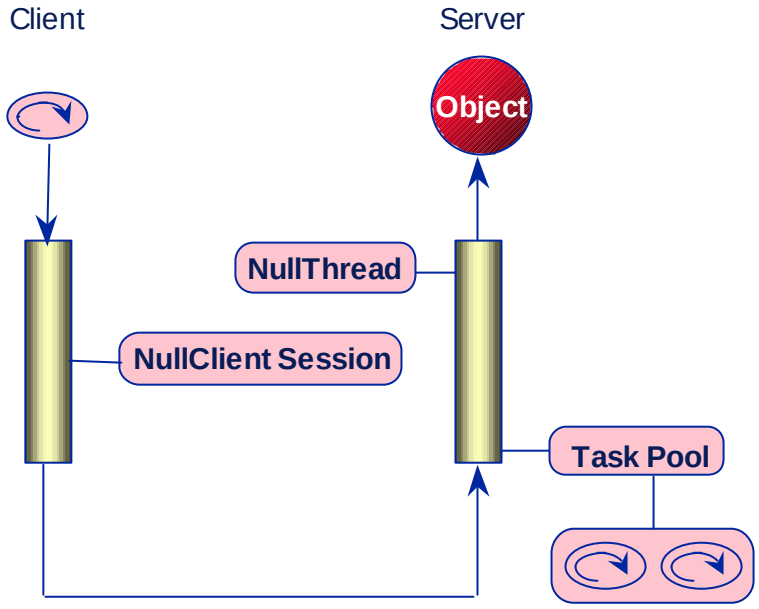
Components

- Generic resource control through allocators
  - factories
  - pools
- Threading framework
  - null thread
  - normal thread (task)
  - scheduled light-weight threads
- May be composed to form a range of policies

Example Configurations



Default QoS

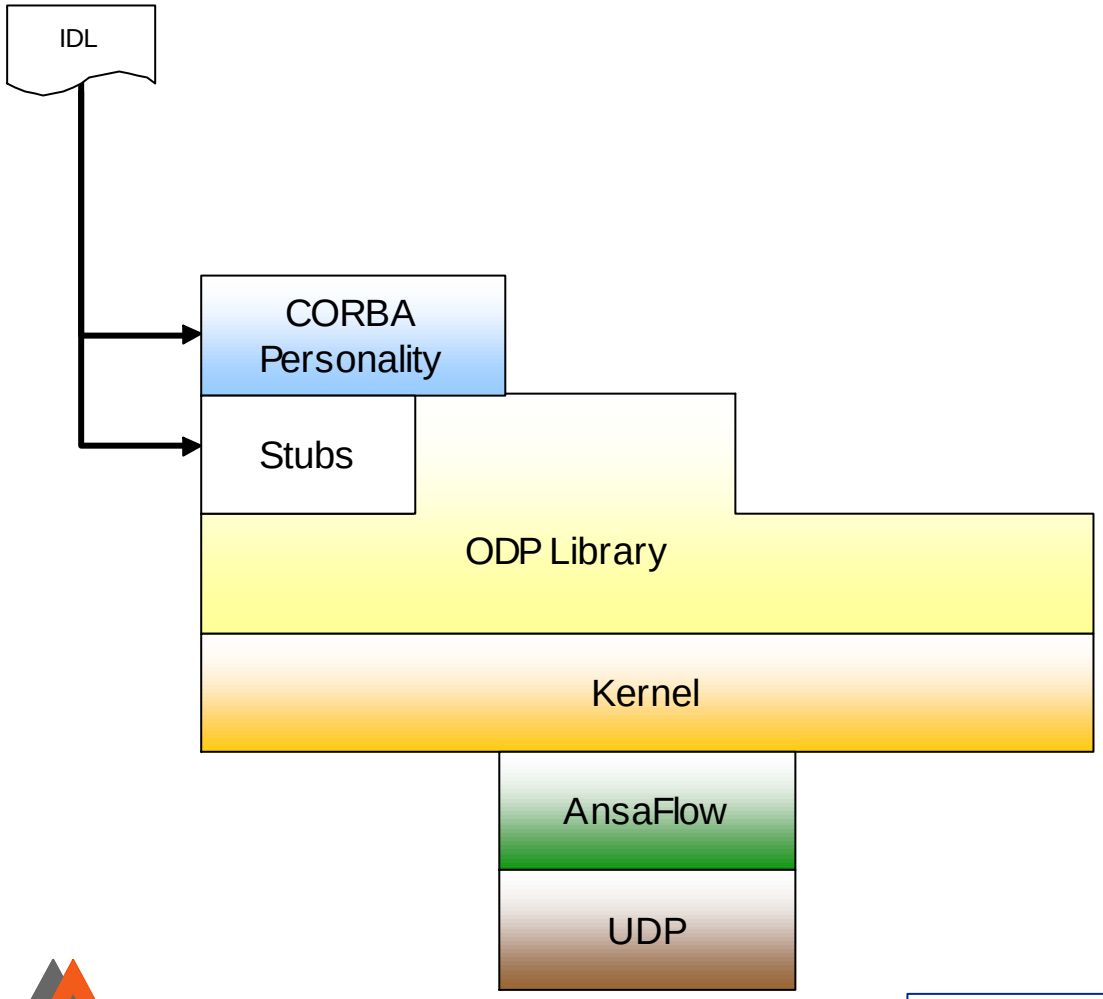


High performance

Implementation Pitfalls

- Failure to observe strict separation of concerns
  - allowing mechanism to determine policy
  - functional overlap preventing fine grain control
- Implicit assumptions
  - e.g. memory management policy
- Regarding all resources as equal
  - e.g. active tasks are not like passive buffers

Flows and Explicit Binding

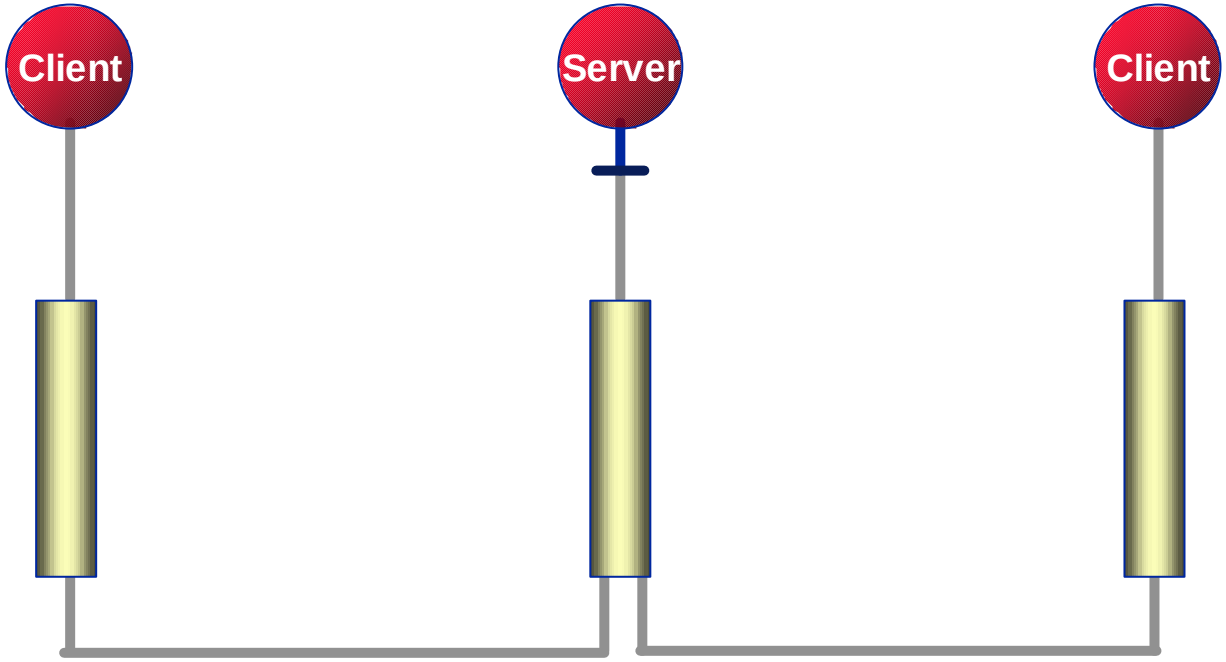


Flow Support

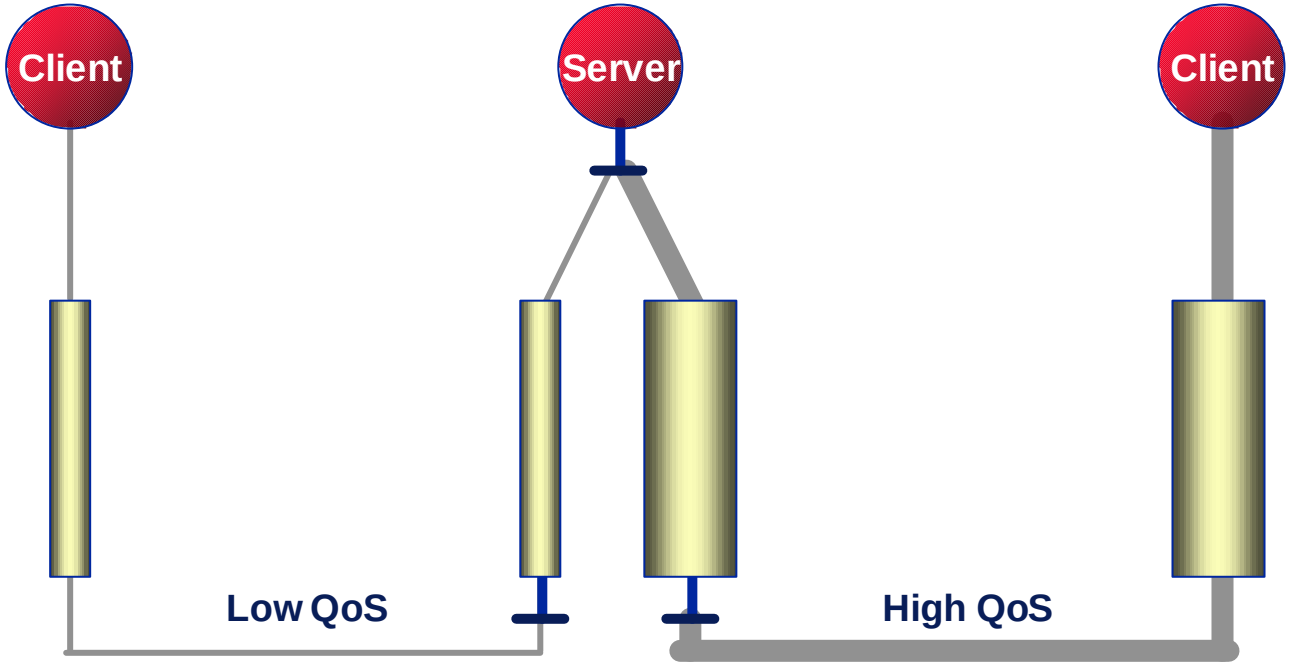
```
flow Video {
    void NewFrame (in long frame_no,
        in string image);
};

interface Video_Binder {
    exception AlreadyBound {};
    Video bind (in Video_Binder peer)
        raises (AlreadyBound);
};

interface Video_bindManager {
    Video_Binder binder ();
};
```

- IDL extensions
- Flow Protocol (AnsaFlow)

Server Endpoints



Server Endpoints



Specifying QoS

- QoS is required between application endpoints
- ... on a per connection basis
  - e.g. different client instances using the same server interface may desire different QoS.
- QoS is determined at bind time
- ... requiring additional binding apparatus
  - ... taking QoS attributes

Endpoint Implementation

- Endpoints computationally visible as Invocation Refs
- Avoids introducing another computational type
- Allows EP to be treated just like any other InvRef
  - exported to Trader
  - passed out of the capsule as an operation parameter
- Any Invocation Ref may be explicitly bound

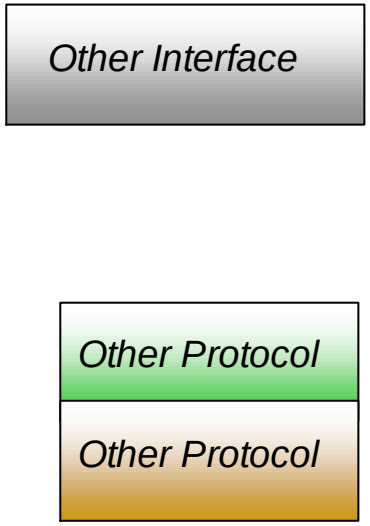
Observations

- Complete QoS control requires:
  - OS and network protocol support
- Engineering transparency trade off at all levels
  - QoS enabled binders become protocol specific
  - applications may need to be aware of engineering mechanisms
  - implications for dynamic loading of protocols
  - ... unless hidden by QoS mapping

Engineering QoS

- Engineering QoS can be supported by protocols
  - protocol converts “generic” QoS parameters to their own specific protocol parameters
  - allows applications to specify QoS requirements in protocol independent fashion
  - defers protocol choice to runtime

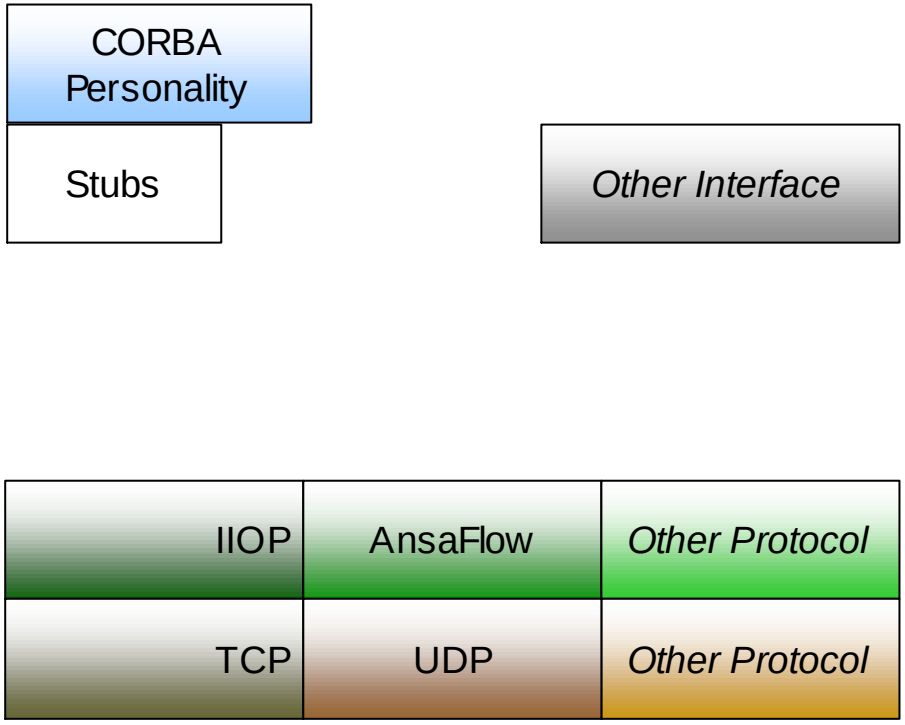
Extensibility



Extensibility

- Front-end separated from kernel so new “personalities” can be added
- Protocol binders have well defined interface
- Support for adding new protocols at runtime

Modularity



Modularity

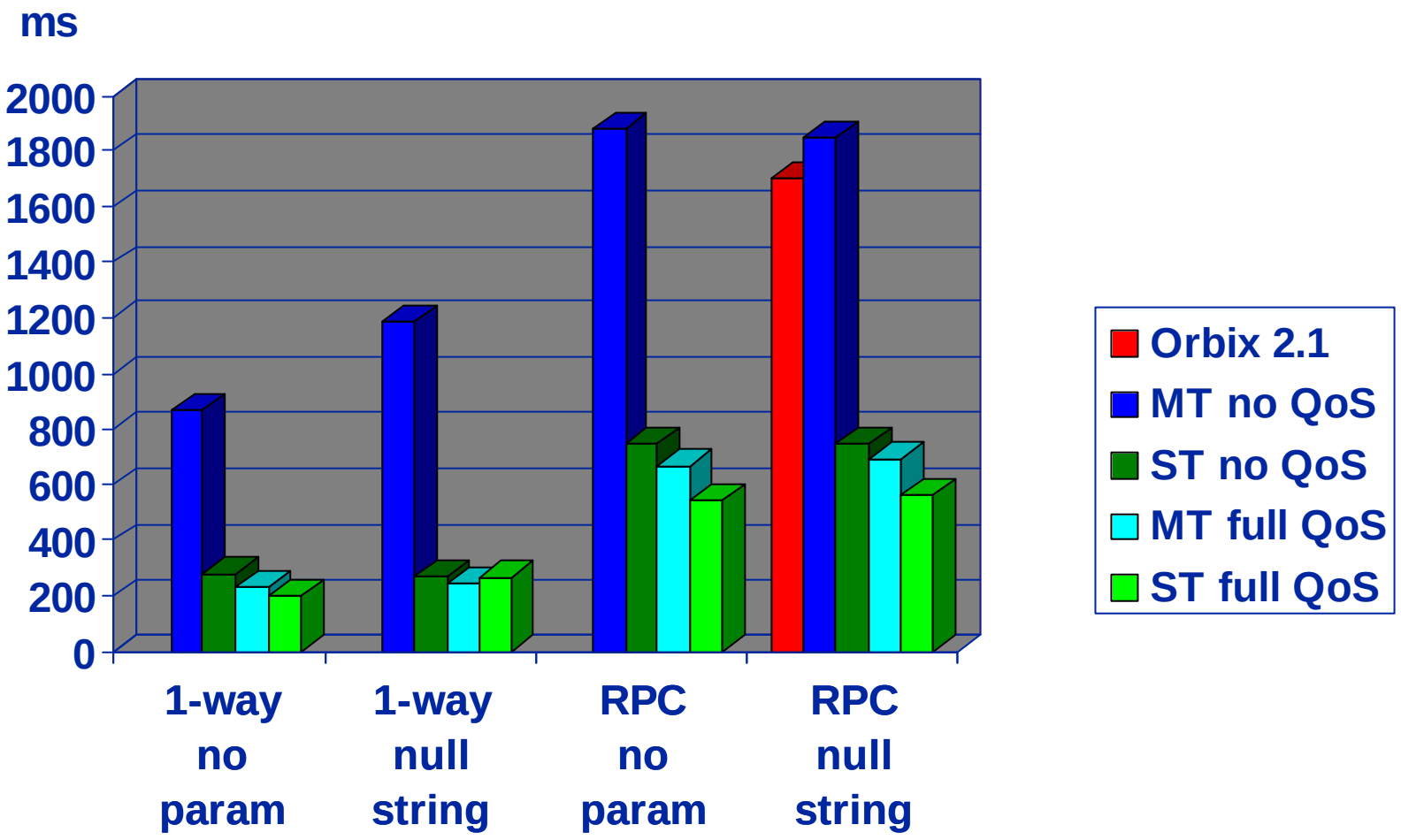
- Small kernel
- Only desired front-end and protocol components need be linked in
- Protocols layers can be reused (e.g. TCP and UDP layers).

Performance

Performance Improvements

- Replace iostream marshalling with custom
  - eliminate low level mutex locking
  - compile time decisions (use of templates)
- IIOP optimisation
  - replace marshall/unmarshall of IIOP headers at different protocol layers
  - no context switch for high performance QoS configurations of IIOP client sessions
- TCP read-ahead
  - minimise ‘recv’ system call overhead

Performance



DIMMA Summary

- Unique, flexible ORB supporting RT and MM
- Implementation framework for
  - threading
  - resource control mechanisms
  - protocol composition
- Layered architecture
  - clean interfaces allow multiple “personalities”
  - reuse encouraged
- API providing transparency between RPC and flows

DIMMA Summary

- Explicit binding with specified QoS
  - abstract Engineering QoS binder
  - protocol specific QoS binders
- Wide range of possible QoS
  - protocol read/multiplex task policy
  - session dispatch task policy
  - channel multiplex policy
  - buffer pools and specific buffer sizes
- Dynamic protocol loading

Documentation

- Overview - APM.1995
- Tracing - APM.1980
- Build and Installation - APM.2036
- Writing an application - APM.2037
- Design and Implementation - APM.2063
- Performance Analysis - APM.2046

- ...plus other workshop presentations available