APM

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE • CB3 0RD UNITED KINGDOM
+44 1223 515010 • Fax: +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

ANSA Phase III

DIMMA ODP Library

Dave Otway

Abstract

Internal engineering of the DIMMA ODP library and its relation to stubs.

APM.1887.01 | **Approved** | 26th November 1996

Briefing Note

Distribution:
Supersedes:
Superseded by:

ODP Library

Dave Otway

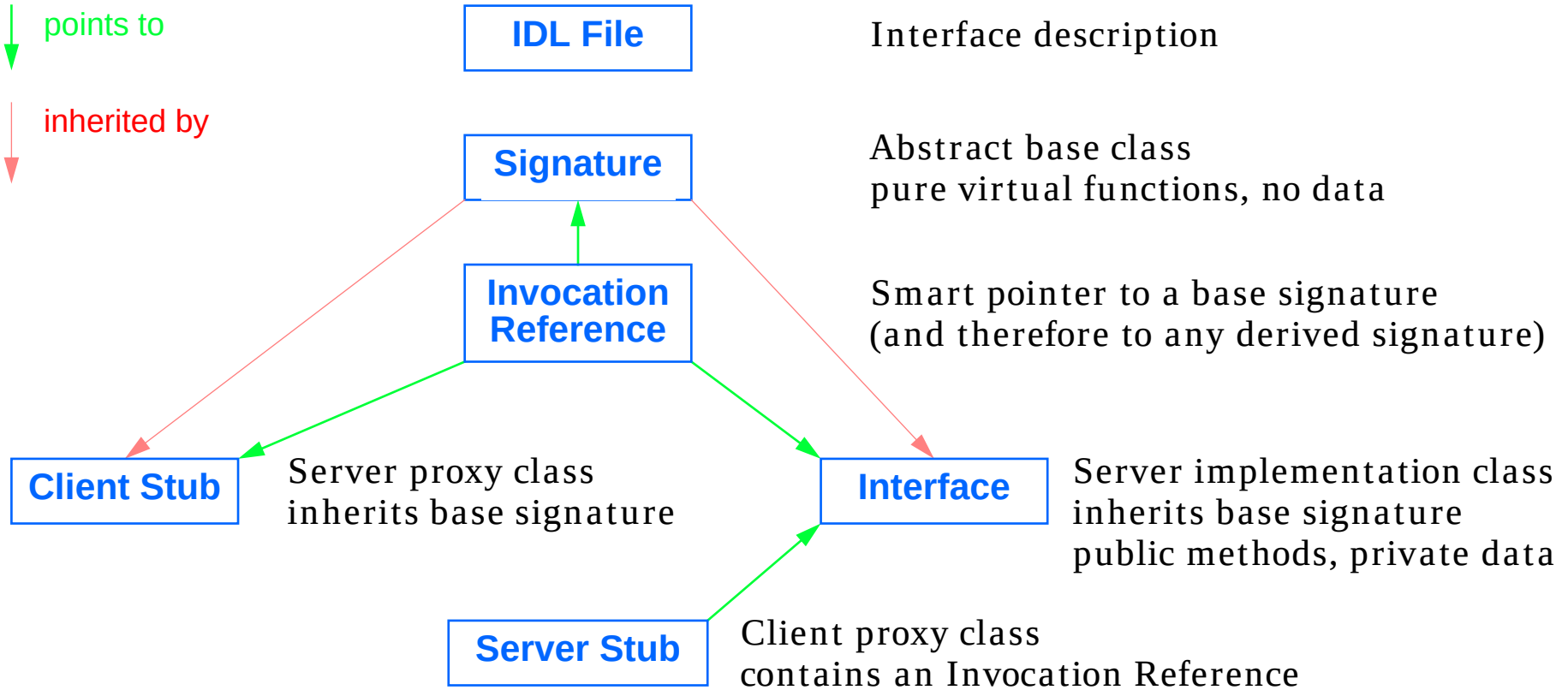
Objectives

- portable ODP conforming computational API in C++
  - common basis for a range of Distributed Programming APIs
- portable engineering API
  - computational API, protocol and platform independent
  - infrastructure for portable distribution transparencies
- portable stubs
  - computational API, protocol and platform independent
  - concurrently protocol independent

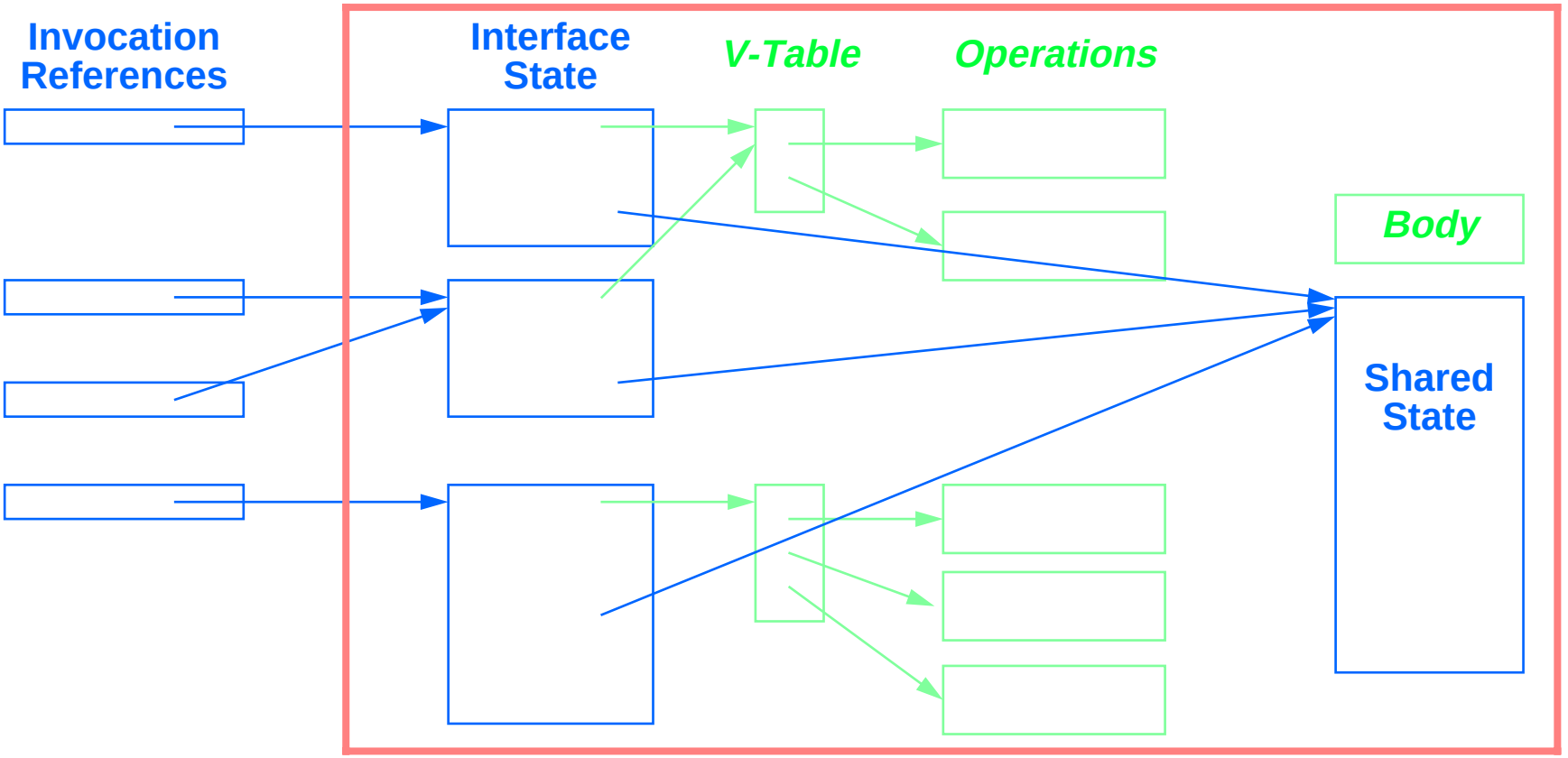
Computational API

- **Stage 1:** objects, interfaces, multiple results, basic types, the type “Any”; signatures, invocation references, named terminations, local garbage collection, (hand coded) trader client stub
- **stage 2:** structured types, threads
- **stage 3:** streams, explicit binding, QoS
- **stage 4:** preprocessor

Interface component relationships



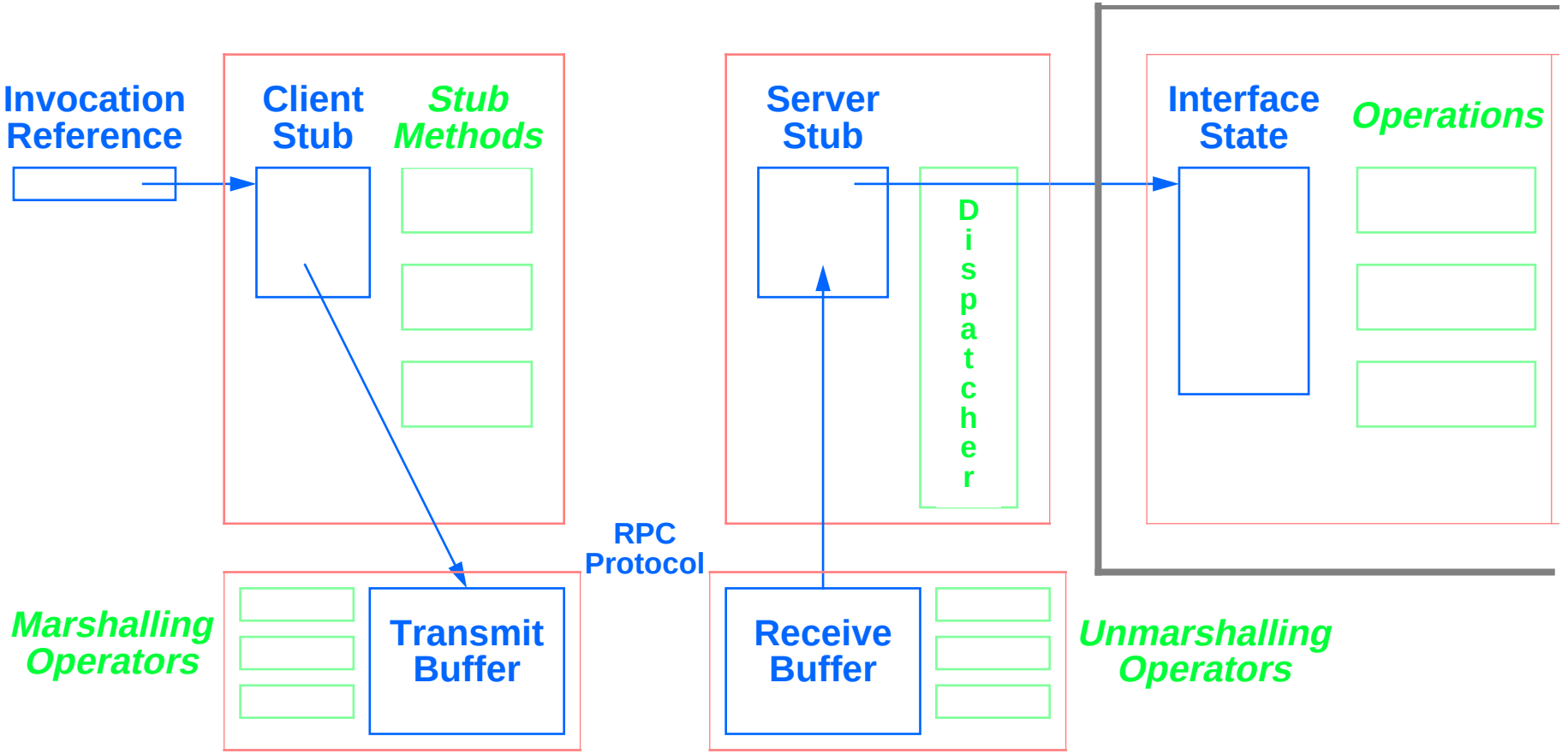
Implementing an ODP Object in C++



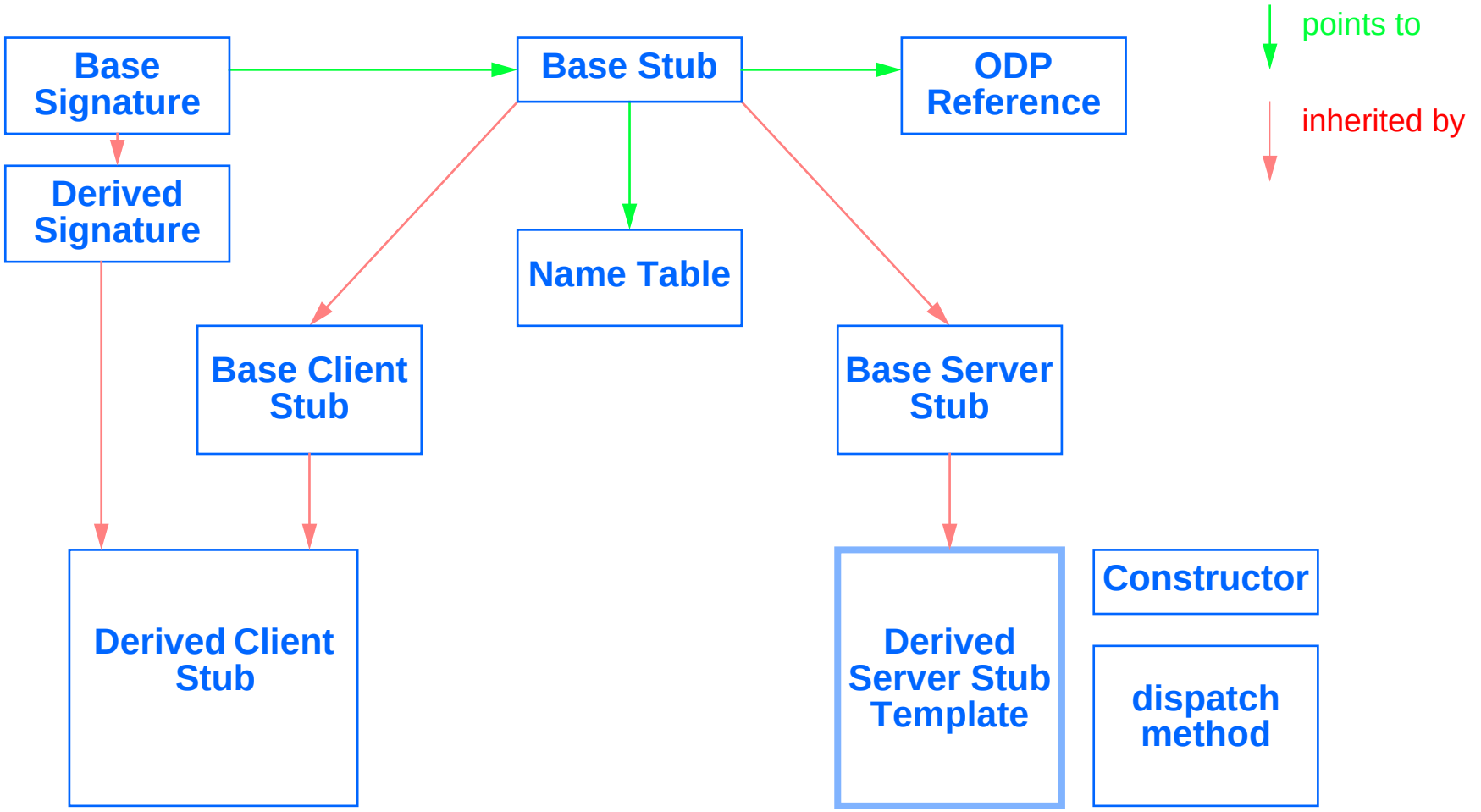
Engineering API

- supports protocol and platform independent stubs with:
- client stub and server stub base classes
  - derived client and server stubs produced by the stub generator
- transmitter and receiver base classes with virtual marshalling operators for the basic types
  - derived buffers have to be implemented for each RPC protocol
  - binder selects appropriate buffer for the client
  - protocol selects appropriate buffer for the server
- templates for generating sequence and array marshalling operators
  - structure and union marshallers produced by the stub generator

A Remote Invocation



Stub component relationships



Client Stubs

- one stub method per server method
- request function binds (if needed)
  and generates a (protocol dependent) transmitter buffer
- arguments marshalled into the buffer using << operator
- invoke function does RPC and returns a receiver buffer
- switch statement decodes response
  - 0: normal return, 1: SystemException, others: application exceptions
- results unmarshalled from the buffer using >> operator
- exceptions are thrown from the client stub
- buffer released by a smart pointer

Server stubs

- single dispatch method
  - called with a (protocol dependent) receiver buffer
- switch statement decodes operation
  - one branch per server method
- each branch:
  - unmarshalls arguments using >> operator
  - calls the corresponding server method
  - catches any exceptions
  - gets a transmitter buffer using the response function (and optionally signals an exception)
  - marshalls results using << operator
  - returns the transmitter buffer