

PEPt

An Adaptable Architecture for Implementing RPC and Messaging Systems

Harold Carr

Sun Microsystems

harold.carr@sun.com

PEPt Benefits

- Shows responsibility, relationships and interactions between fundamental *remoting* (RPC and Messaging) building blocks.
- Simple but comprehensive framework to: *understand, implement, reuse, evolve, maintain* finer-grained details of *remoting* systems.
- Enables *remoting* system to adaptively change: *encodings, protocols & transports*.

Remoting: Fundamental Building Blocks

P – Presentation

data types/APIs used by programmer

E – Encoding

wire representation of data types

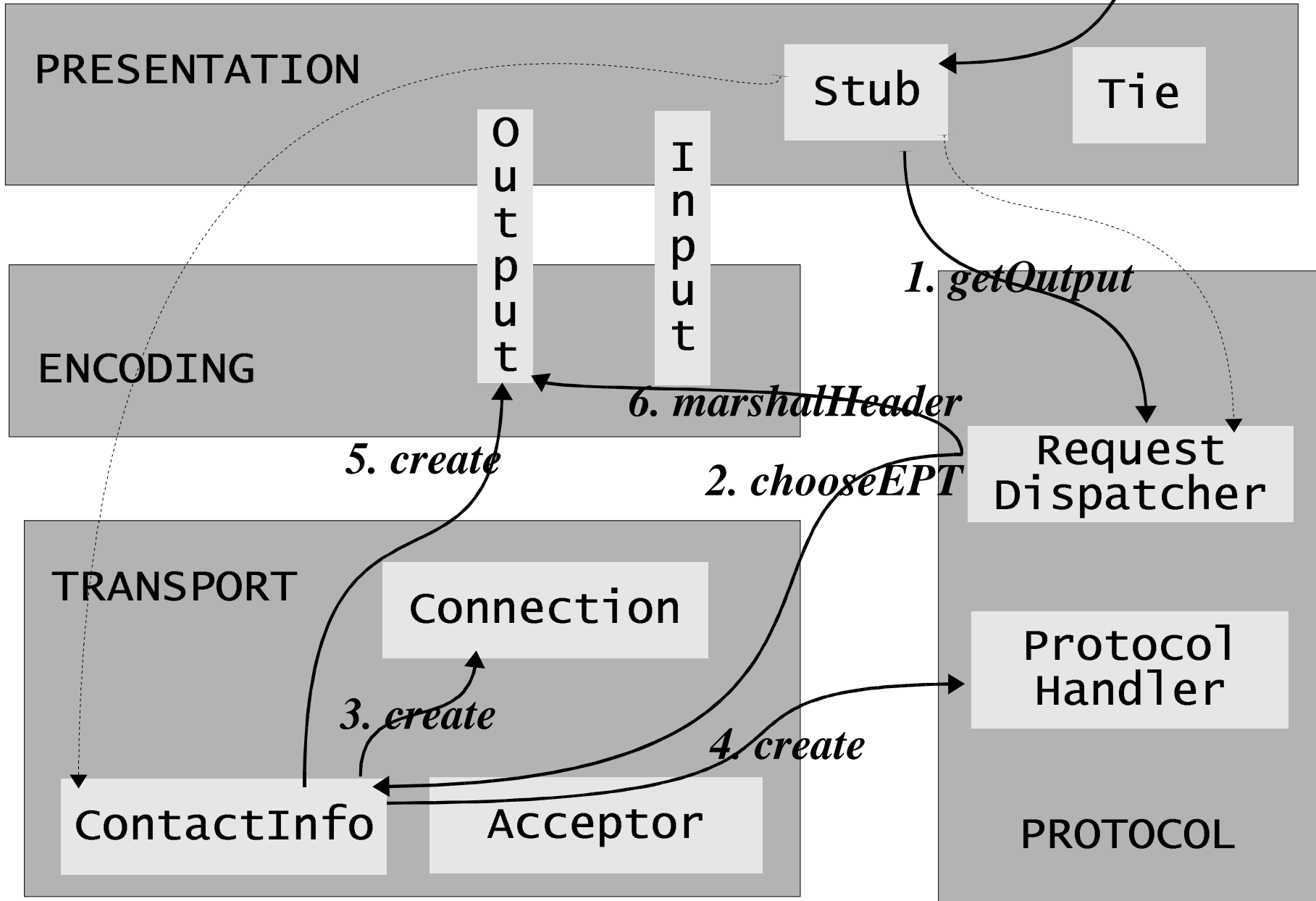
P – Protocol

meta data to frame & convey intent of message

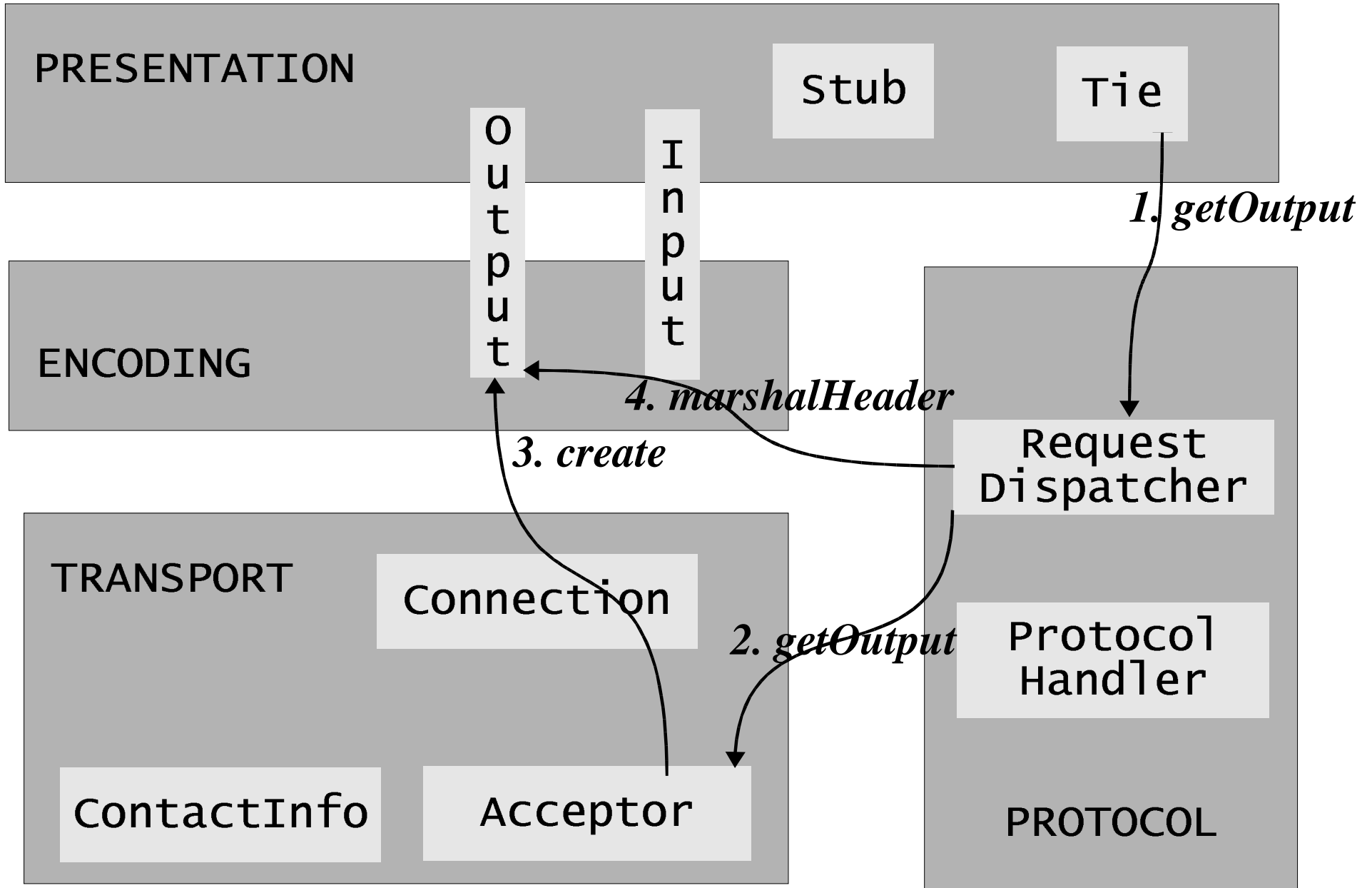
t – transport

moves data/protocol bits between locations

Client Send – Begin Request 0. (*invoke*)



Server Send – Begin Reply



Client/Server Send – Begin Request/Reply Variations

Client chooses

encoding, protocol, transport (EPT) combination for request.

Chosen EPT may use EPT-specific RequestDispatcher.

ContactInfo, Acceptor are *factories* for chosen EPT artifacts.

Connections are created (e.g., HTTP), reused (e.g., GIOP multiplexing) or non-existent (e.g., colocated).

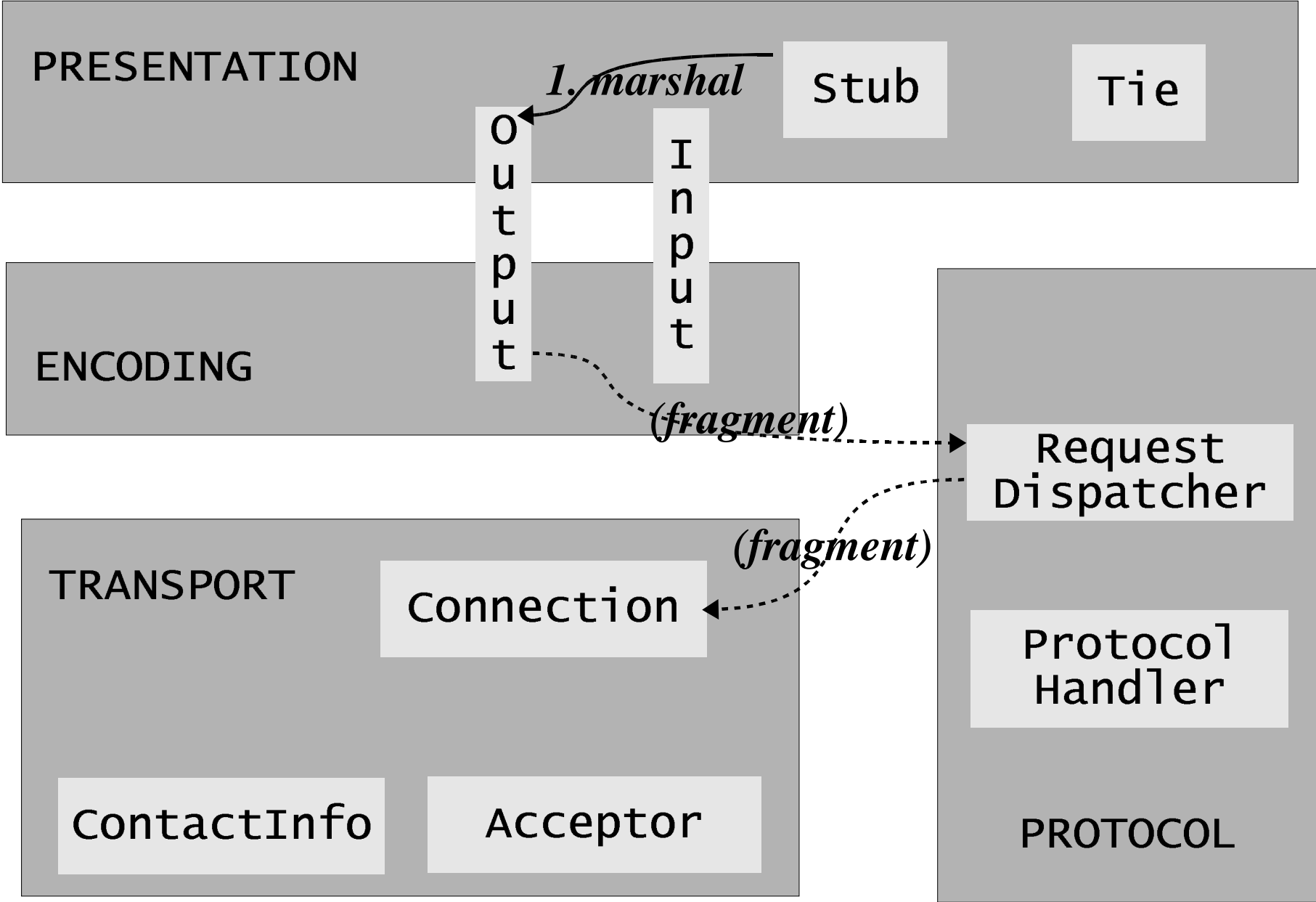
Output Objects enable presentation block data types and encapsulate *encoding* (e.g., SOAP, CDR).

Interceptors may execute before marshalHeader.

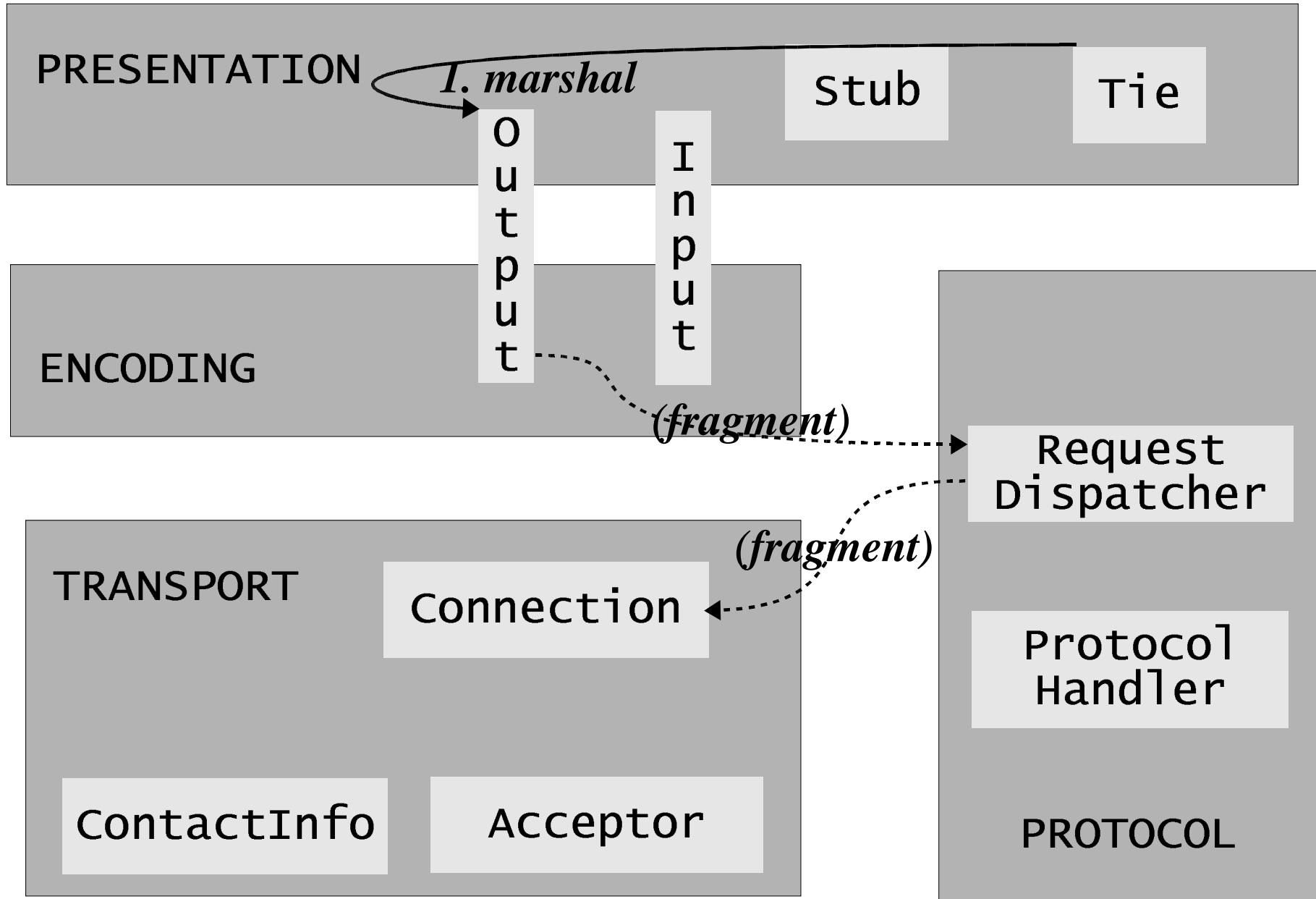
Fragmentation may occur during marshalHeader.

“Early” replies may be received when marshaling/fragmenting header.

Client Send – Marshal



Server Send – Marshal



Client/Server Send – Marshal Variations

Fragments may be sent while marshaling.

If client fragments sent must be prepared for “early” reply.

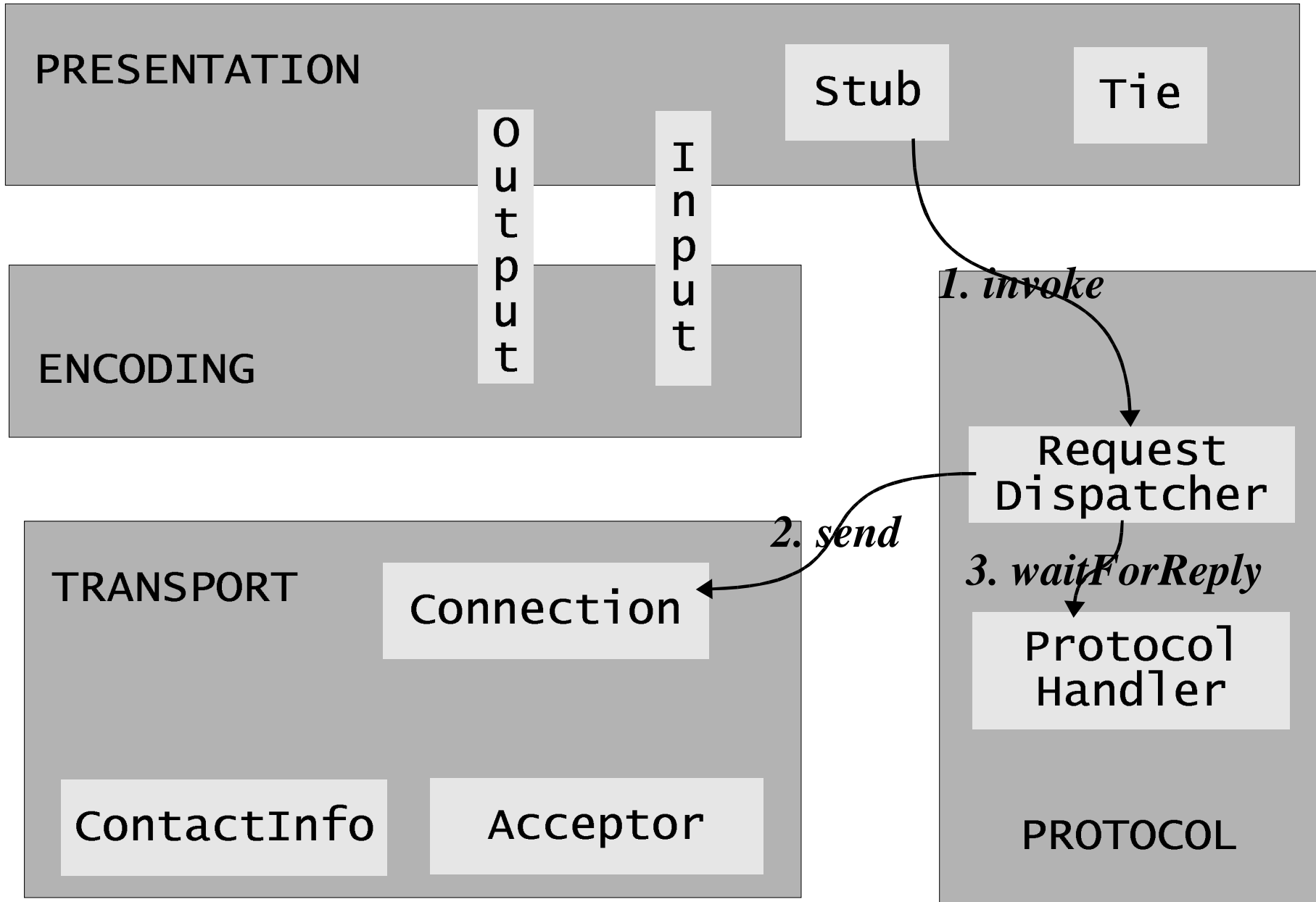
If fragments sent and marshaling error occurs then must inform other side.

Marshaling/unmarshaling may use scatter/gather I/O.

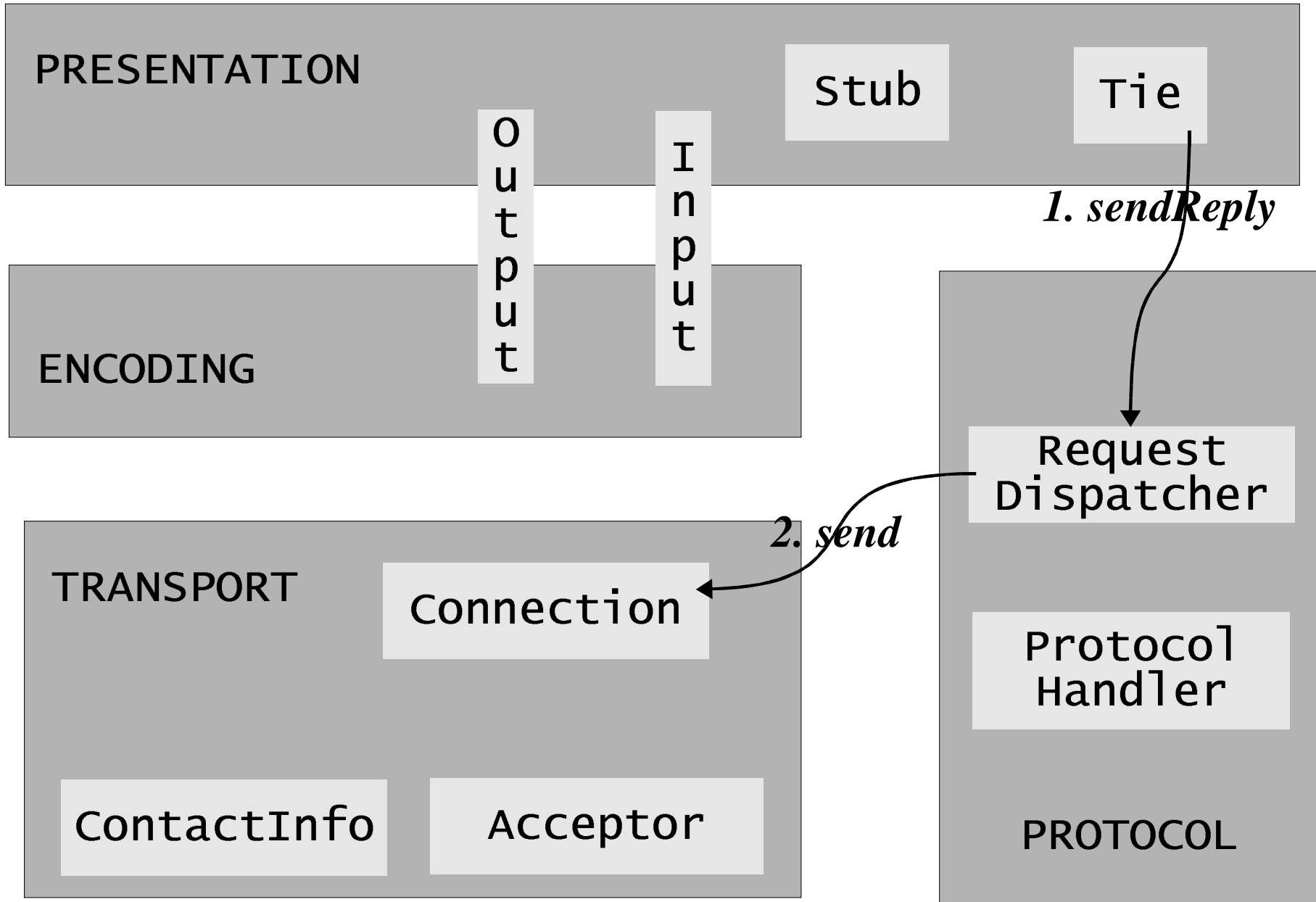
Marshaling may involve chunking, indirections, attachments.

Colocated pass-by-reference optimization (i.e., short-circuits encoding and transport).

Client Send – Marshaling Complete



Server Send – Marshaling Complete



Client/Server Send – Marshaling Complete Variations

**Send encoded message (or last fragment)
via transport connection.**

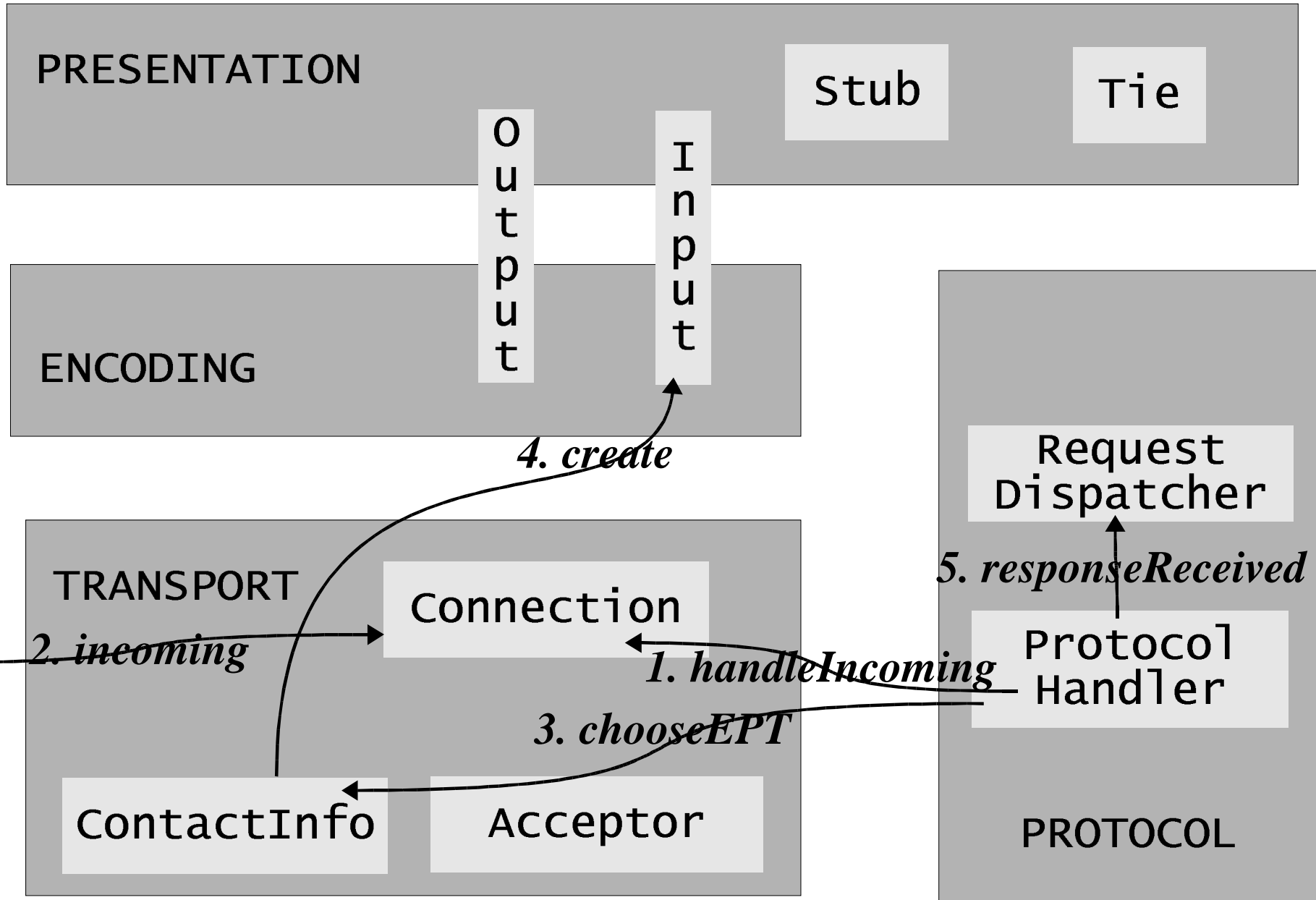
Client waitForReply:

**GIOP: suspend client thread while another thread
demultiplexes replies.**

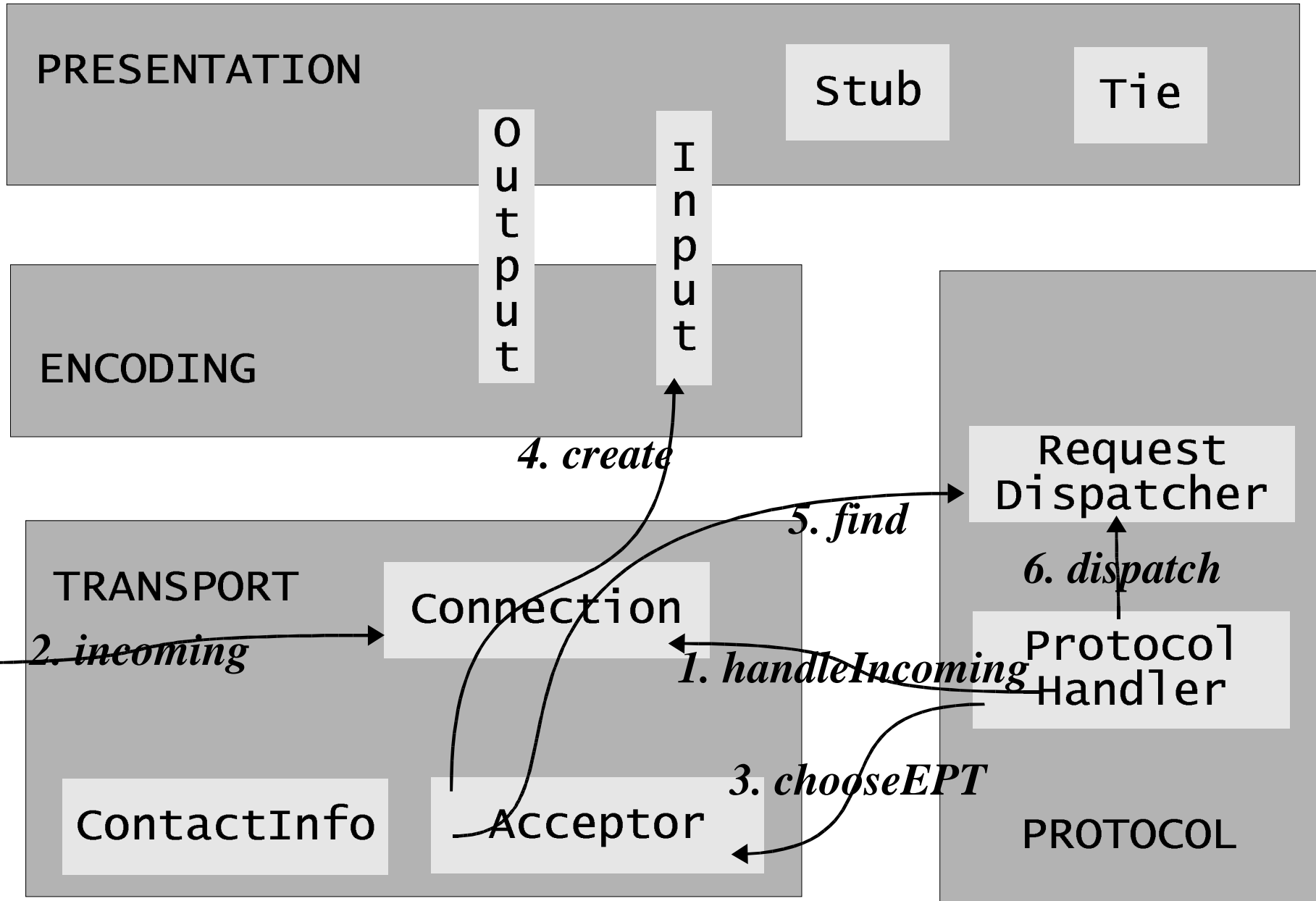
HTTP: block on read.

**Colocated/shared memory:
client thread handles server side dispatch.**

Client Receive – Handle Incoming



Server Receive – Handle Incoming



Client/Server Receive – Handle Incoming Variations

handleIncoming:

**GIOP: “Select” thread or dedicated reader thread.
Thread pools, work queues and policies.**

HTTP: Dedicated reader/worker thread.

Colocated or shared memory: handleIncoming “short-circuited”

Connection may handle multiple protocols.

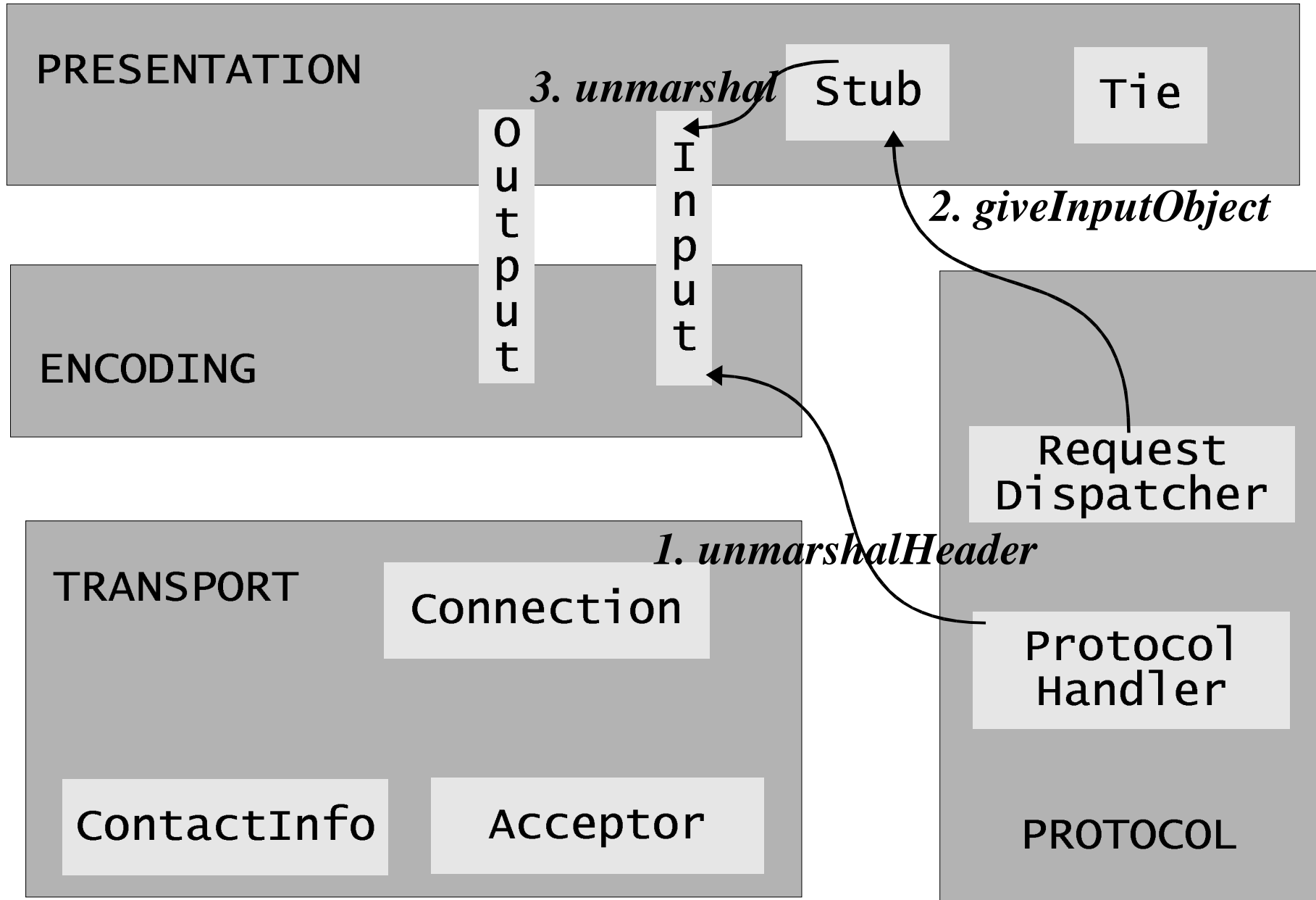
ContactInfo/Acceptor recognizes EPT then creates/finds correct artifacts.

ProtocolHandler determines message type.

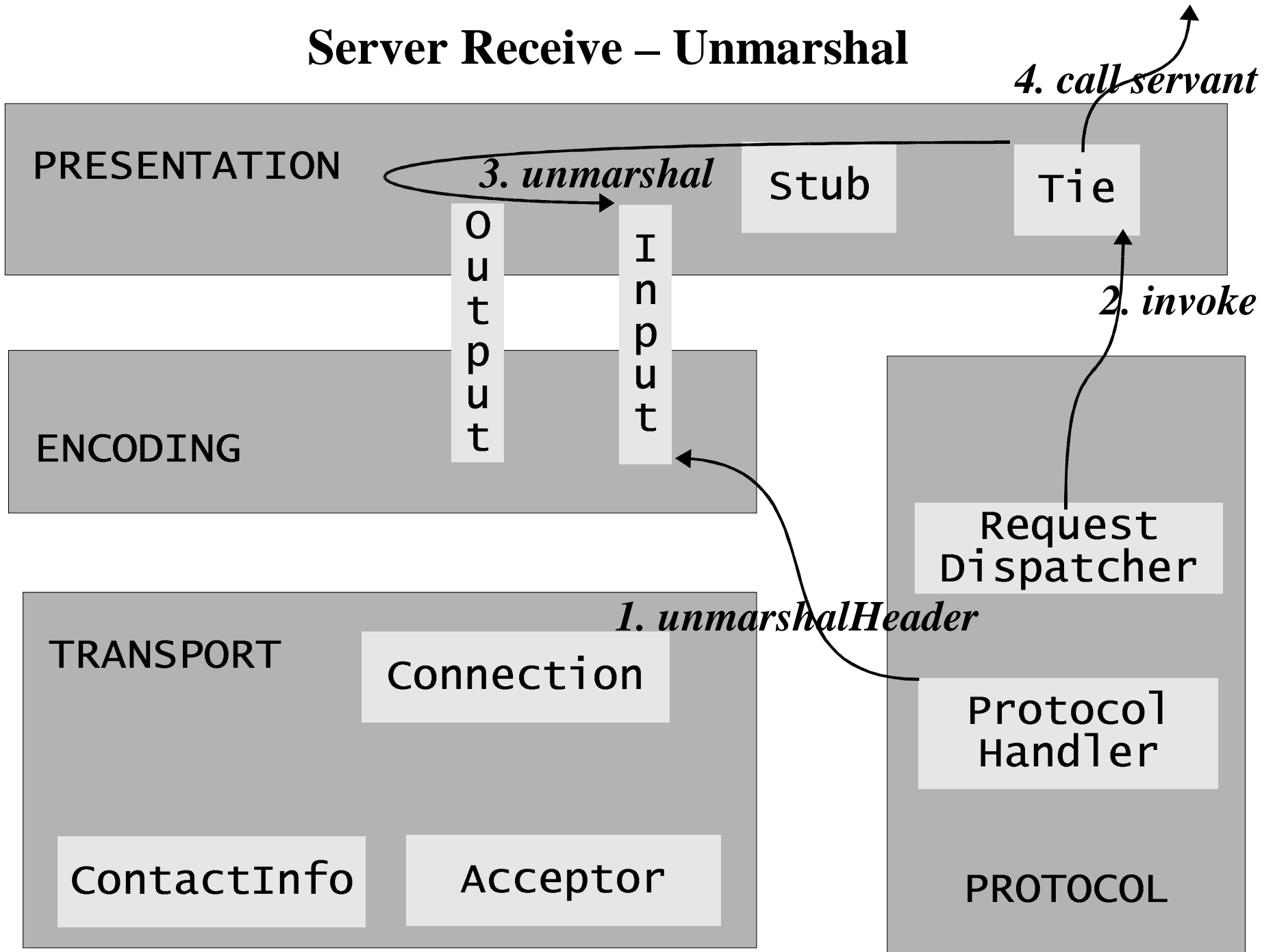
(EPT-specific) RequestDispatcher may handle request/reply.

Fragments: producer/consumer between ProtocolHandler/InputObject

Client Receive – Unmarshal



Server Receive – Unmarshal



Client/Server Receive – Unmarshal Variations

Handle errors during header unmarshaling.

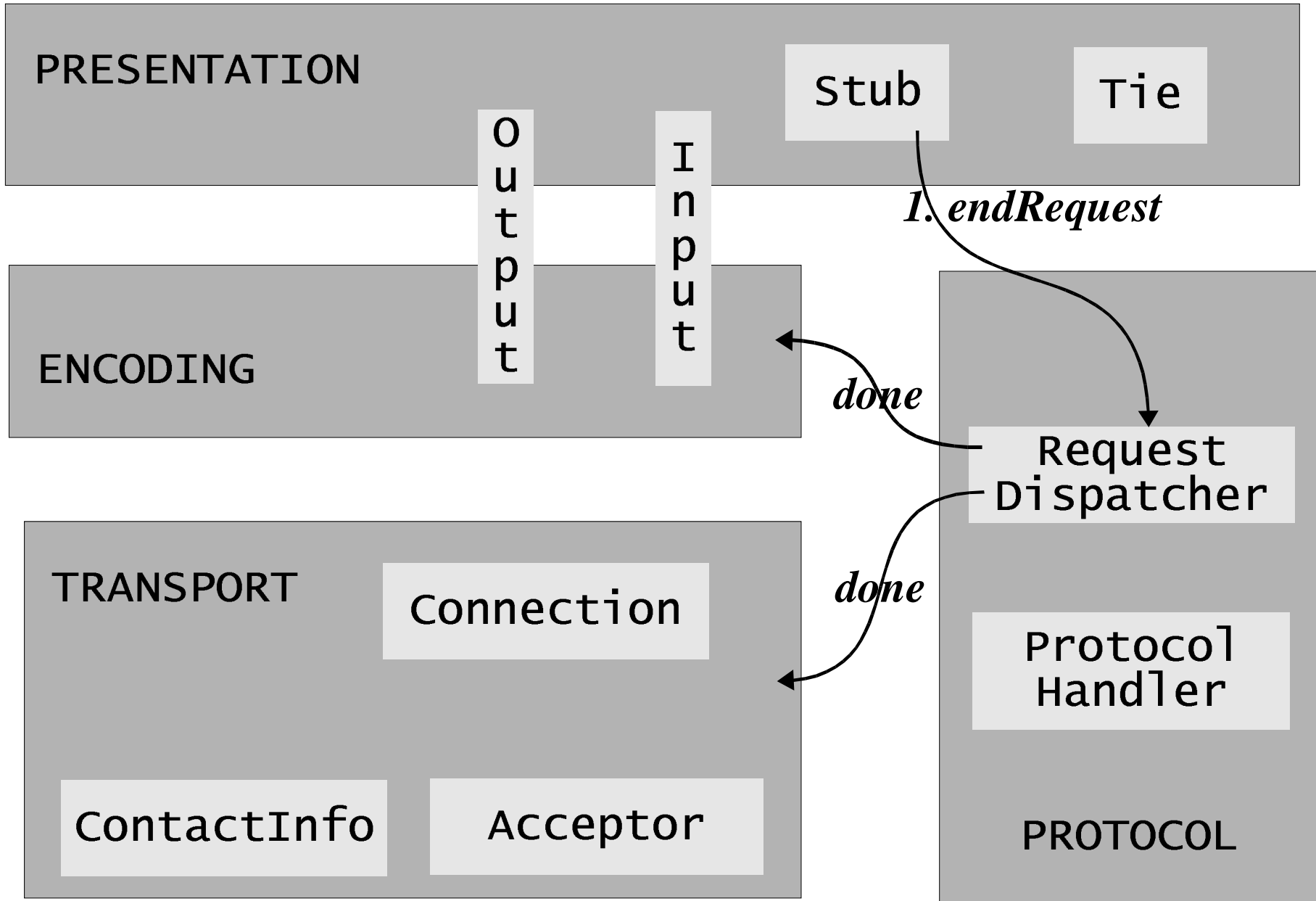
Interceptors may execute after unmarshaling header.

Server: find tie/servant:

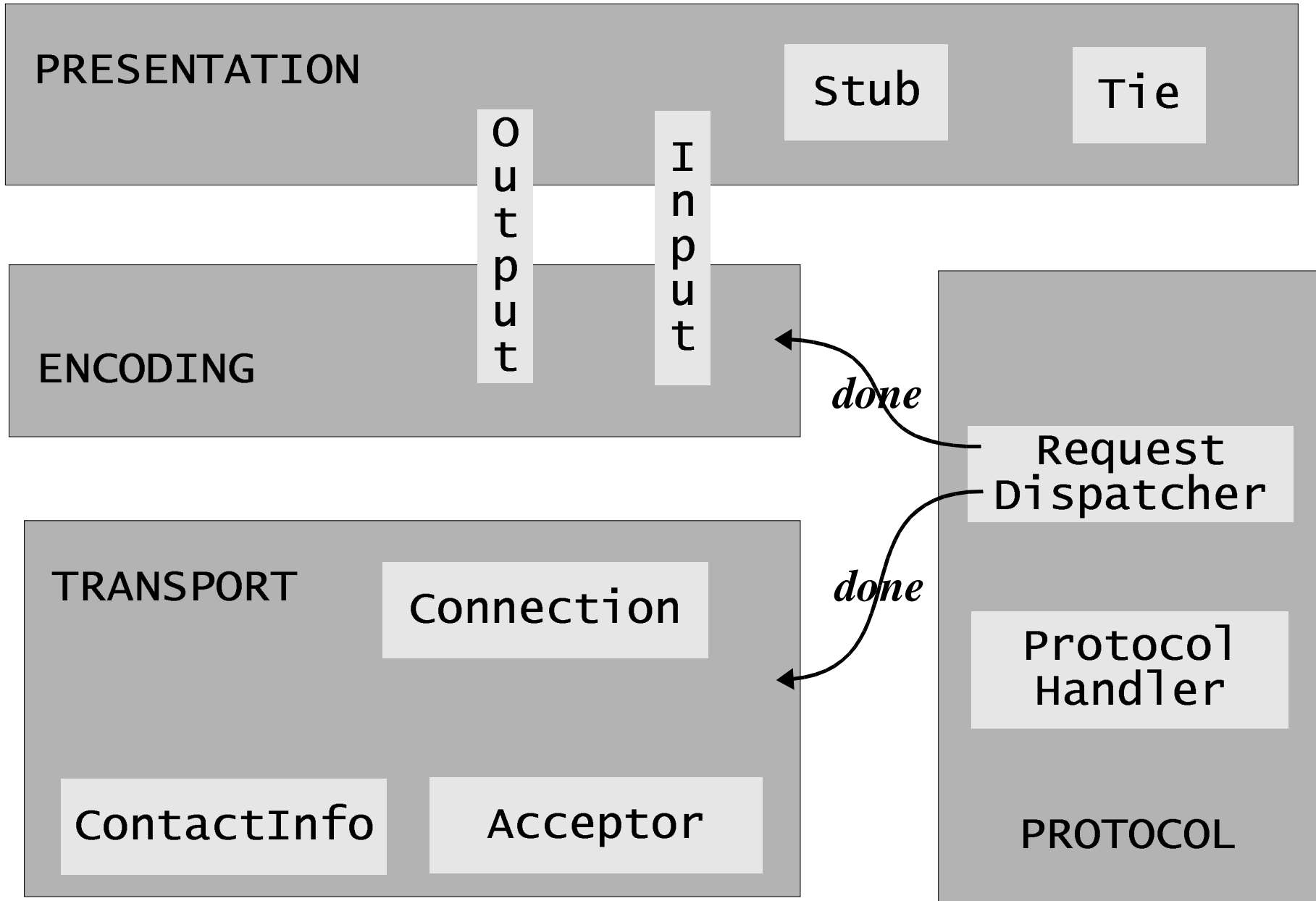
CORBA: use Portable Object Adapter (POA).

**JAX-RPC: use deployment descriptor
(maps URLs to ties/servants).**

Client Receive – End Request



Server Send – End Reply



Client/Server Receive – End Request/Reply Variations

If marshal error and fragments sent notify other side.

Clean up:

All: release input/output objects.

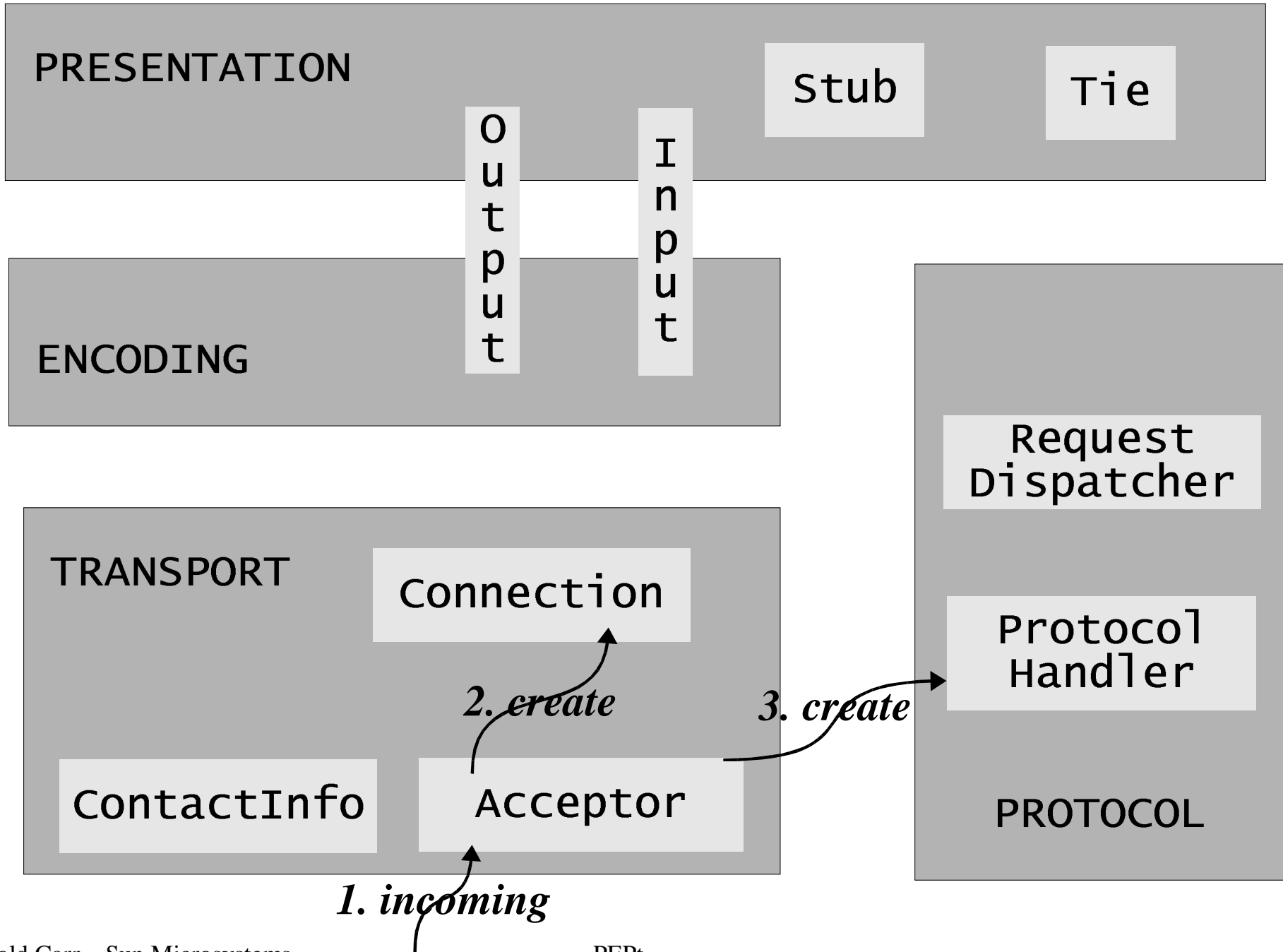
All: release thread-local data.

HTTP: close connection.

**GIOP: decrement connection active count,
perhaps close.**

Fragments: release fragment map entries.

Server Receive – Accept



Related Work

Object-Oriented Network Protocols: Boecking

Cactus: Composable Communication Protocols

BEEP: Application Protocol Framework

SASL: Security Framework for Application Protocols

**RM-ODP: Engineering Interface Channel Model:
Stub, Binder, Protocol, Interceptor**

ACE: Reactor, Task, Acceptor-Connector, Proactor, Streams

Conclusions

- **Adaptable Architecture for building Remoting Systems.**
- *Enables remoting system to adaptively change encodings, protocols and transports.*
- **Handles addressing, colocation, retries, errors, connection multiplexing, fragmentation, transactions, security, ...**
- **Used in ORB in Sun's Java 1.5 and SunONE App Server 8.0**

Future Work

- **Apply the PEPT Architecture to messaging systems.**