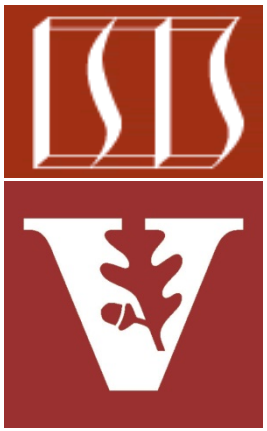
Android Services & Local IPC: The Broker Pattern (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



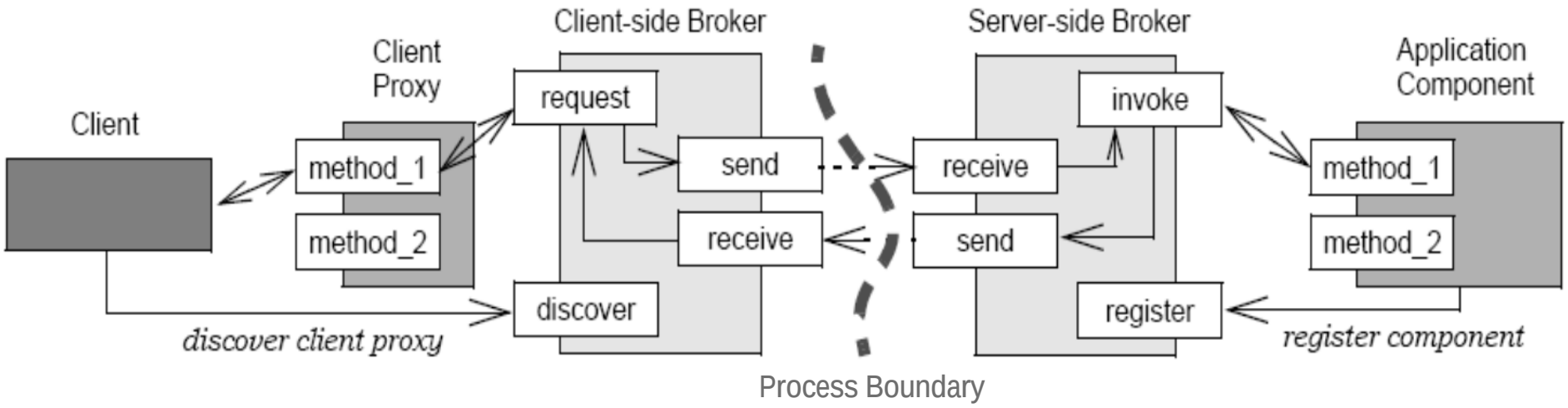
Professor of Computer Science

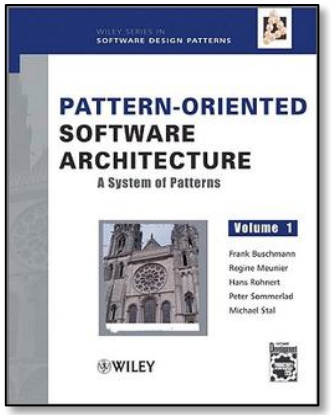
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand the *Broker* pattern



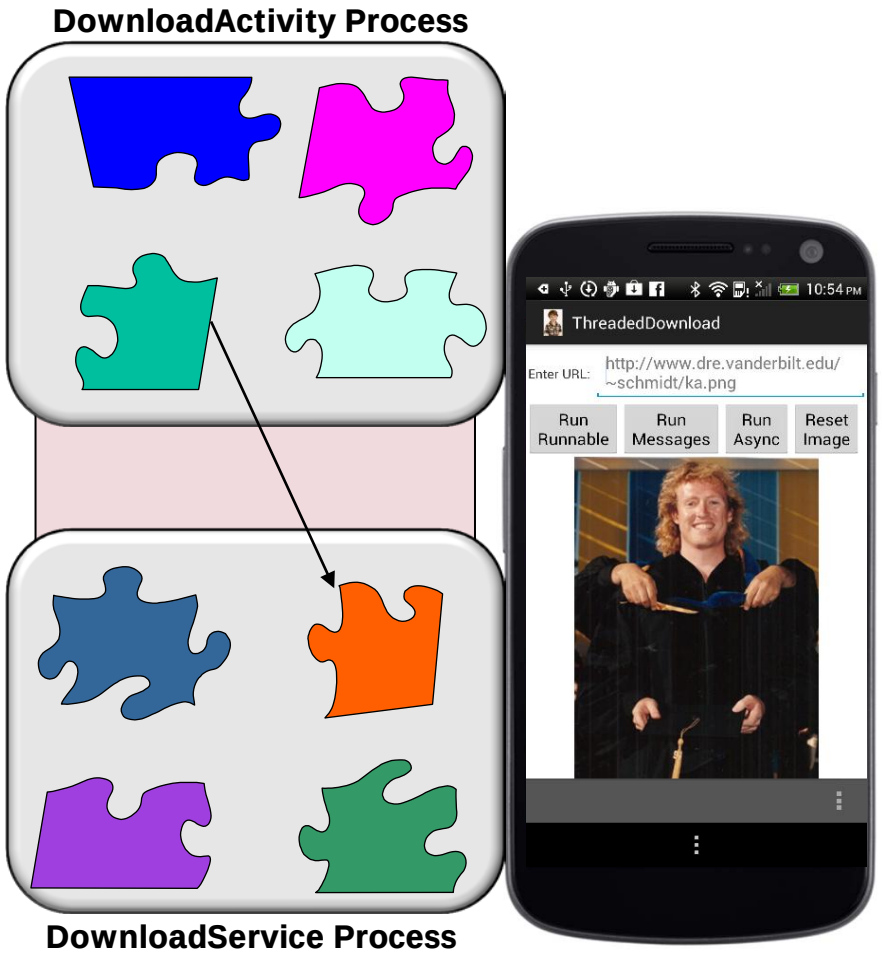


See www.kircher-schwanninger.de/michael/publications/BrokerRevisited.pdf

Challenge: Isolating Communication Concerns

Context

- A system with multiple (potentially) remote objects that interact synchronously or asynchronously

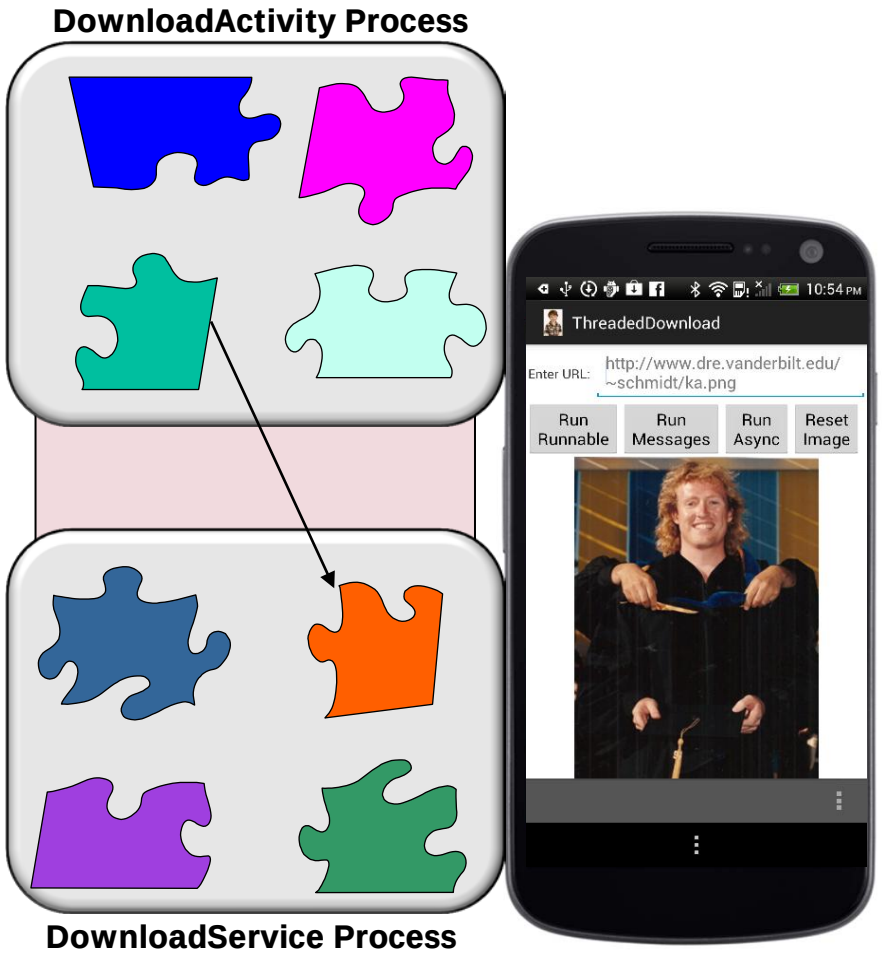


Android's Binder provides a high-performance IPC mechanism

Challenge: Isolating Communication Concerns

Problems

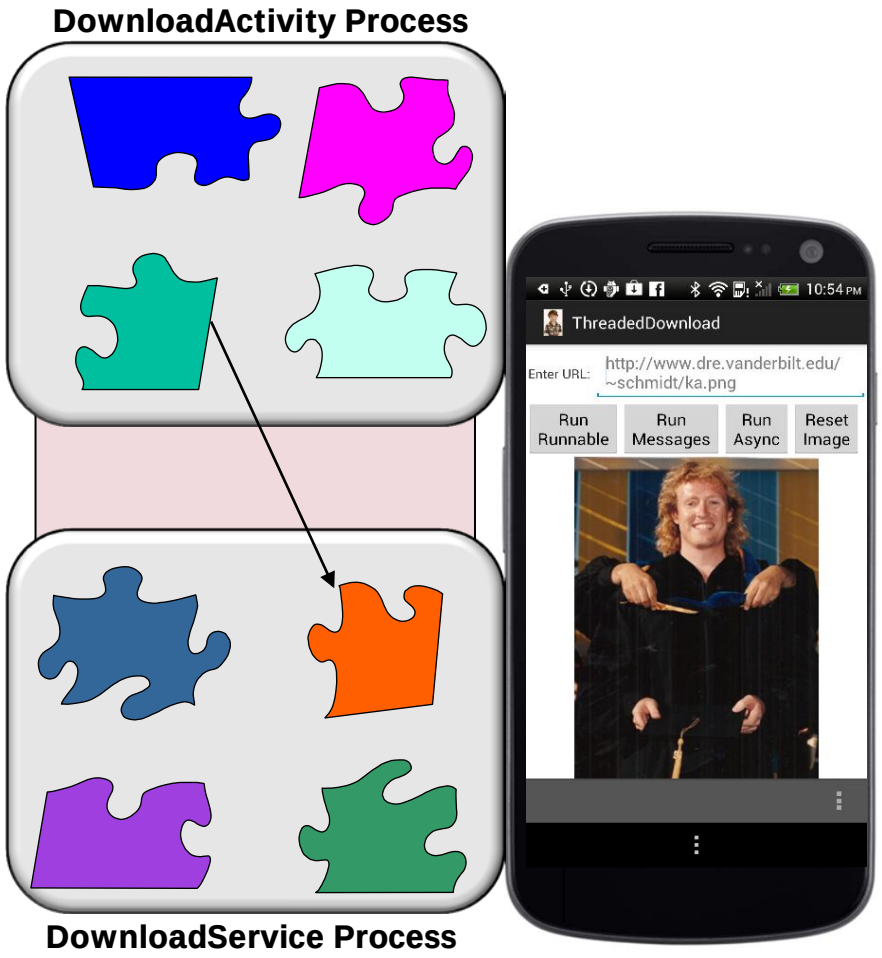
- App developers shouldn't need to handle
  - Low-level message passing, which is fraught with accidental complexity



Challenge: Isolating Communication Concerns

Problems

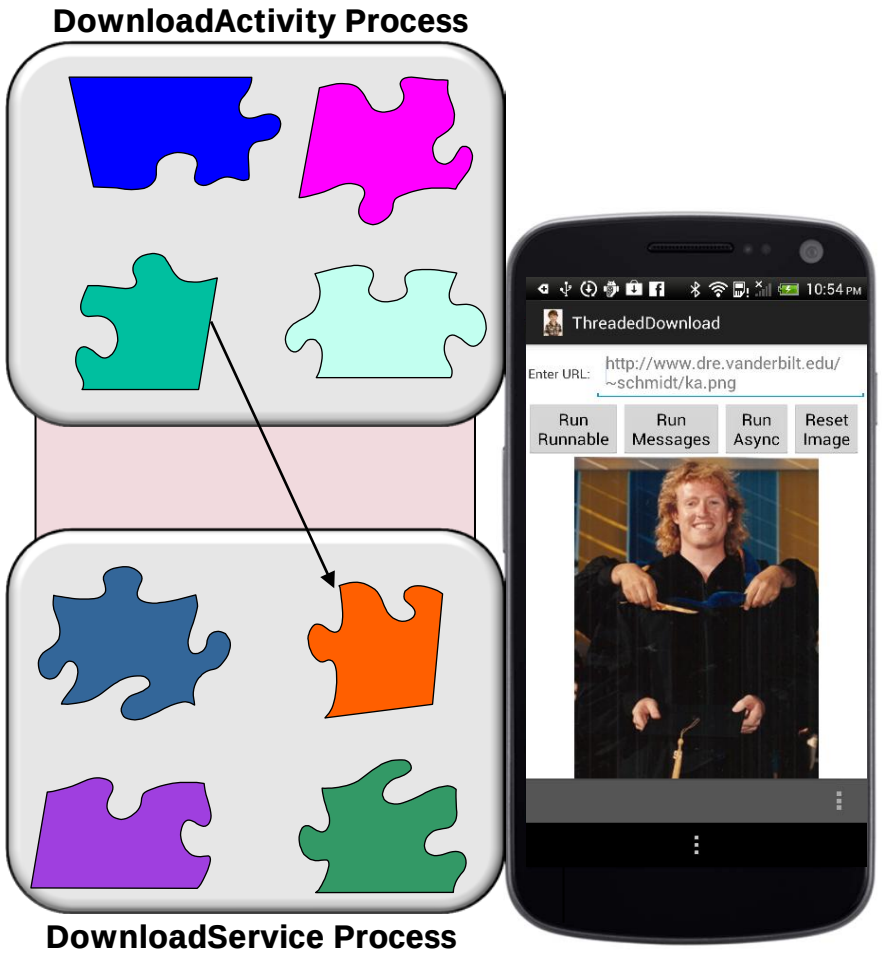
- App developers shouldn't need to handle
  - Low-level message passing, which is fraught with accidental complexity
  - Networked computing diversity
    - e.g., heterogeneous languages, operating systems, protocols, hareware, etc.



Challenge: Isolating Communication Concerns

Problems

- App developers shouldn't need to handle
  - Low-level message passing, which is fraught with accidental complexity
  - Networked computing diversity
  - Inherent complexities of communication
    - e.g., partial failures, security mechanisms, latency, etc.

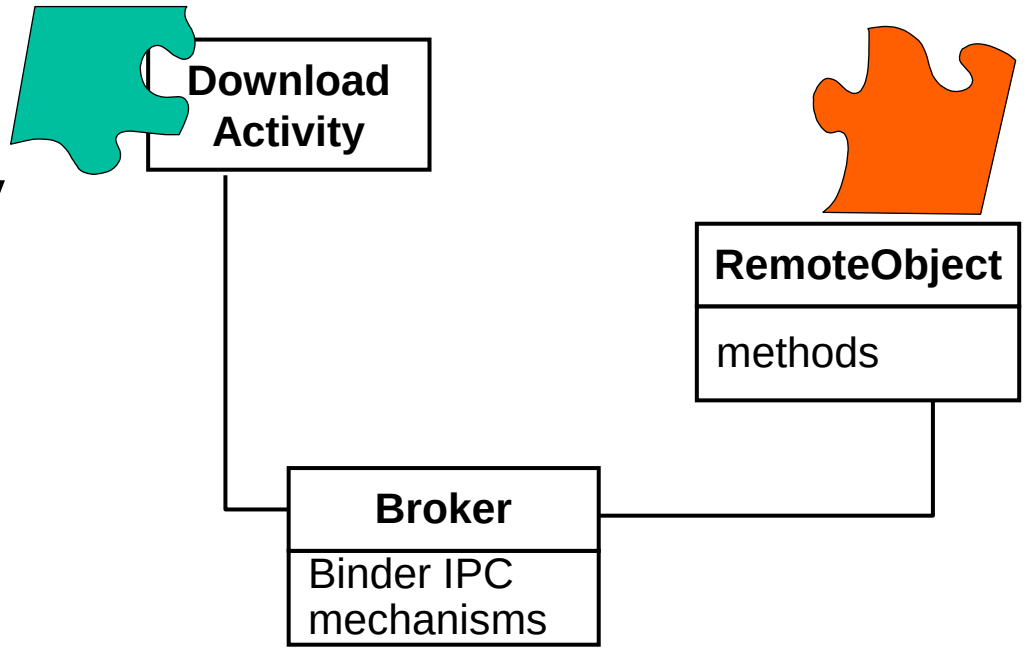


See www.dre.vanderbilt.edu/~schmidt/comm-foreword.html for more info

Use a Broker to Handle Communication Concerns

Solution

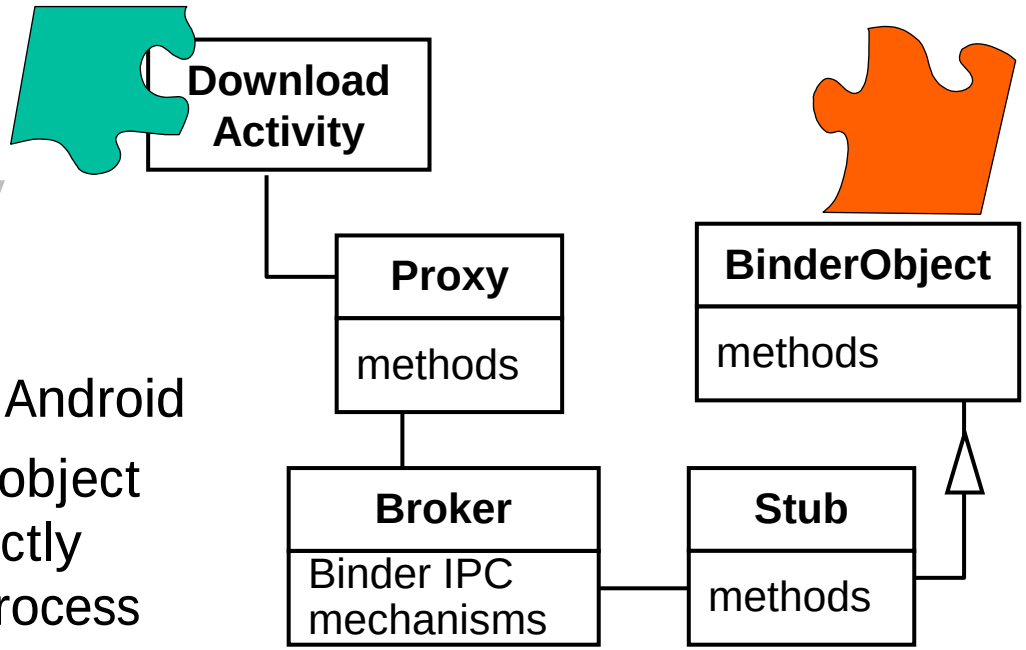
- Separate system communication functionality from app functionality by providing a *broker* that isolates communication-related concerns



Use a Broker to Handle Communication Concerns

Solution

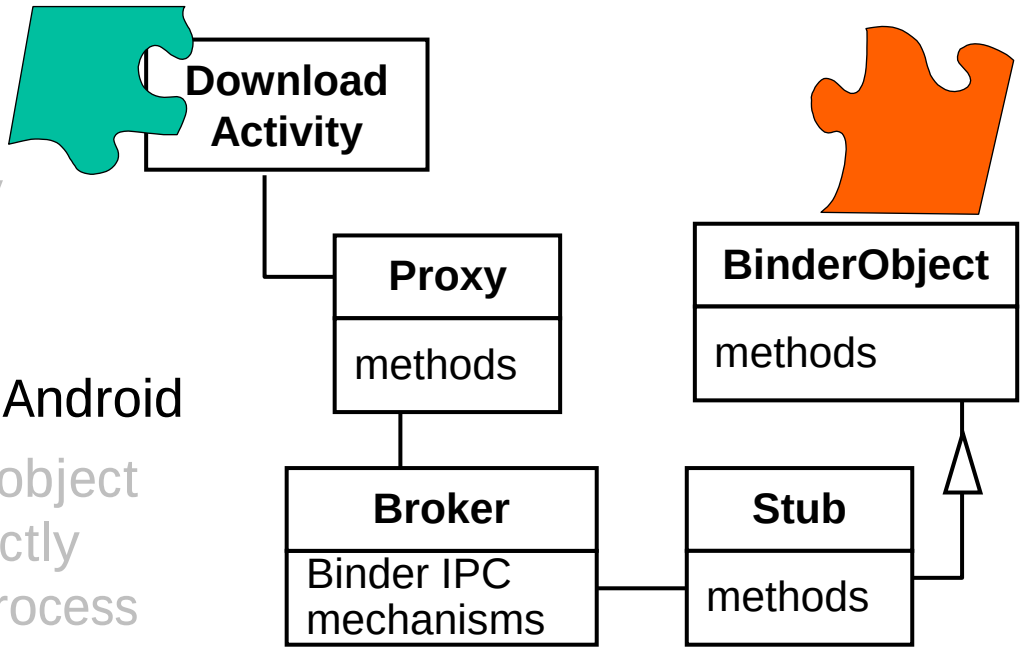
- Separate system communication functionality from app functionality by providing a *broker* that isolates communication-related concerns
- e.g., one way to implement this in Android
  - A Service implements an Binder object that a client can't accessible directly since it may reside in different process



Use a Broker to Handle Communication Concerns

Solution

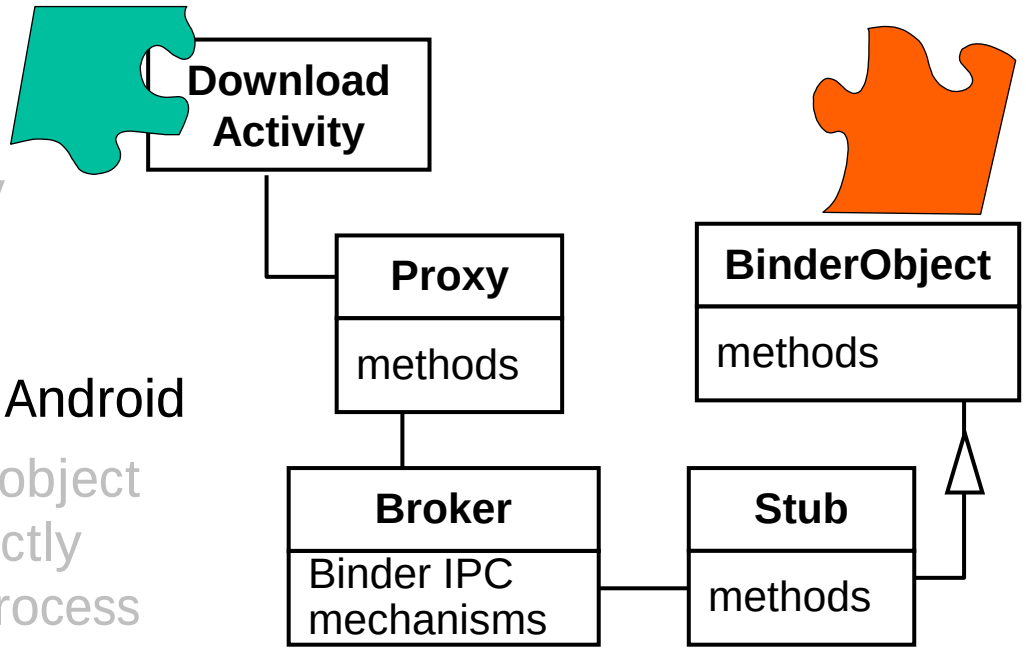
- Separate system communication functionality from app functionality by providing a *broker* that isolates communication-related concerns
- e.g., one way to implement this in Android
  - A Service implements an Binder object that a client can't accessible directly since it may reside in different process
  - Clients call a method on the proxy, which uses the Android Binder IPC mechanism (broker) to communicate with the object across process boundaries



Use a Broker to Handle Communication Concerns

Solution

- Separate system communication functionality from app functionality by providing a *broker* that isolates communication-related concerns
- e.g., one way to implement this in Android
  - A Service implements an Binder object that a client can't accessible directly since it may reside in different process
  - Clients call a method on the proxy, which uses the Android Binder IPC mechanism (broker) to communicate with the object across process boundaries
  - The Binder IPC mechanisms use a stub to upcall a method to the object

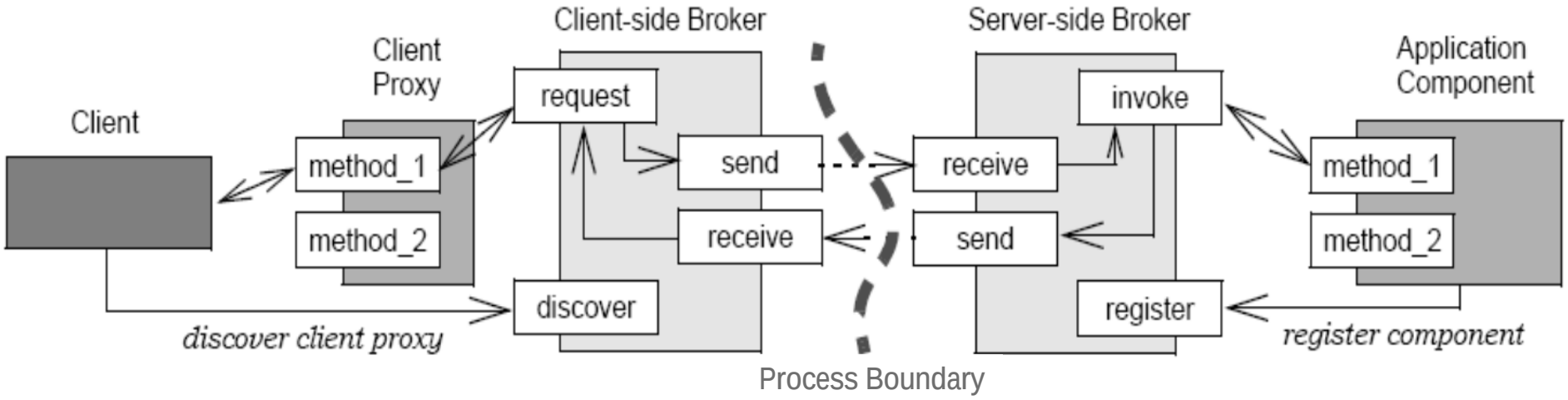


Android Binder uses *Broker* & *Proxy* to support sync & async communication

Broker POSA1 Architectural Pattern

Intent

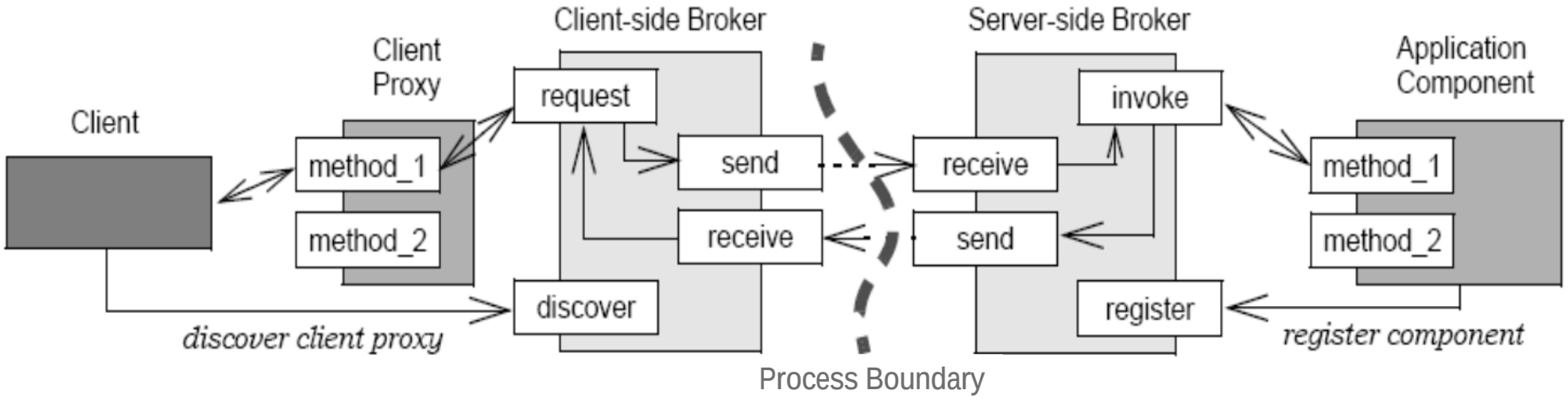
- Connect clients with remote objects by mediating invocations from clients to remote objects, while encapsulating the details of local and/or remote IPC



Broker POSA1 Architectural Pattern

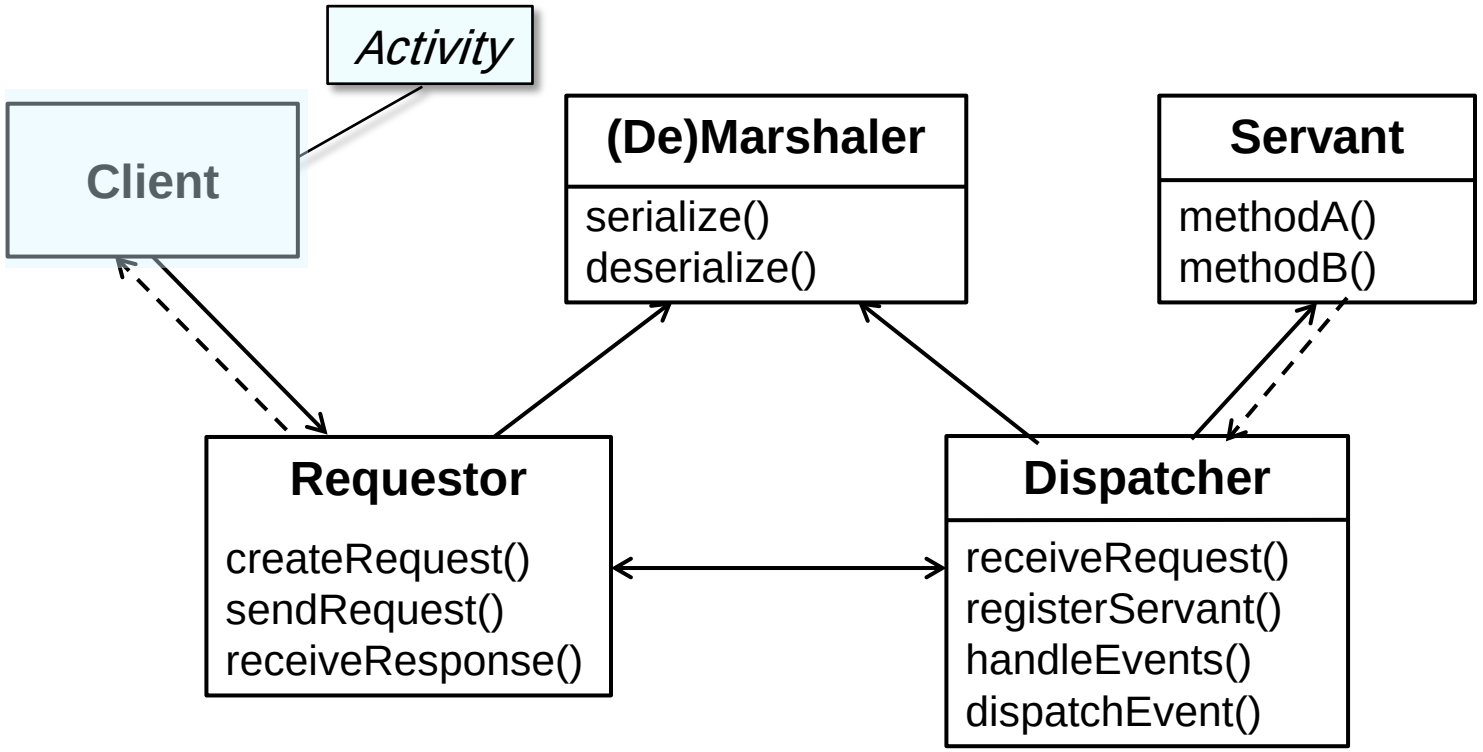
Applicability

- When apps need reusable capabilities that
  - Support (potentially) remote communication in a location transparent manner
  - Detect/handle faults & manage end-to-end QoS
  - Encapsulate low-level systems programming details
    - e.g., memory management, connection management, data transfer, concurrency, synchronization, etc.



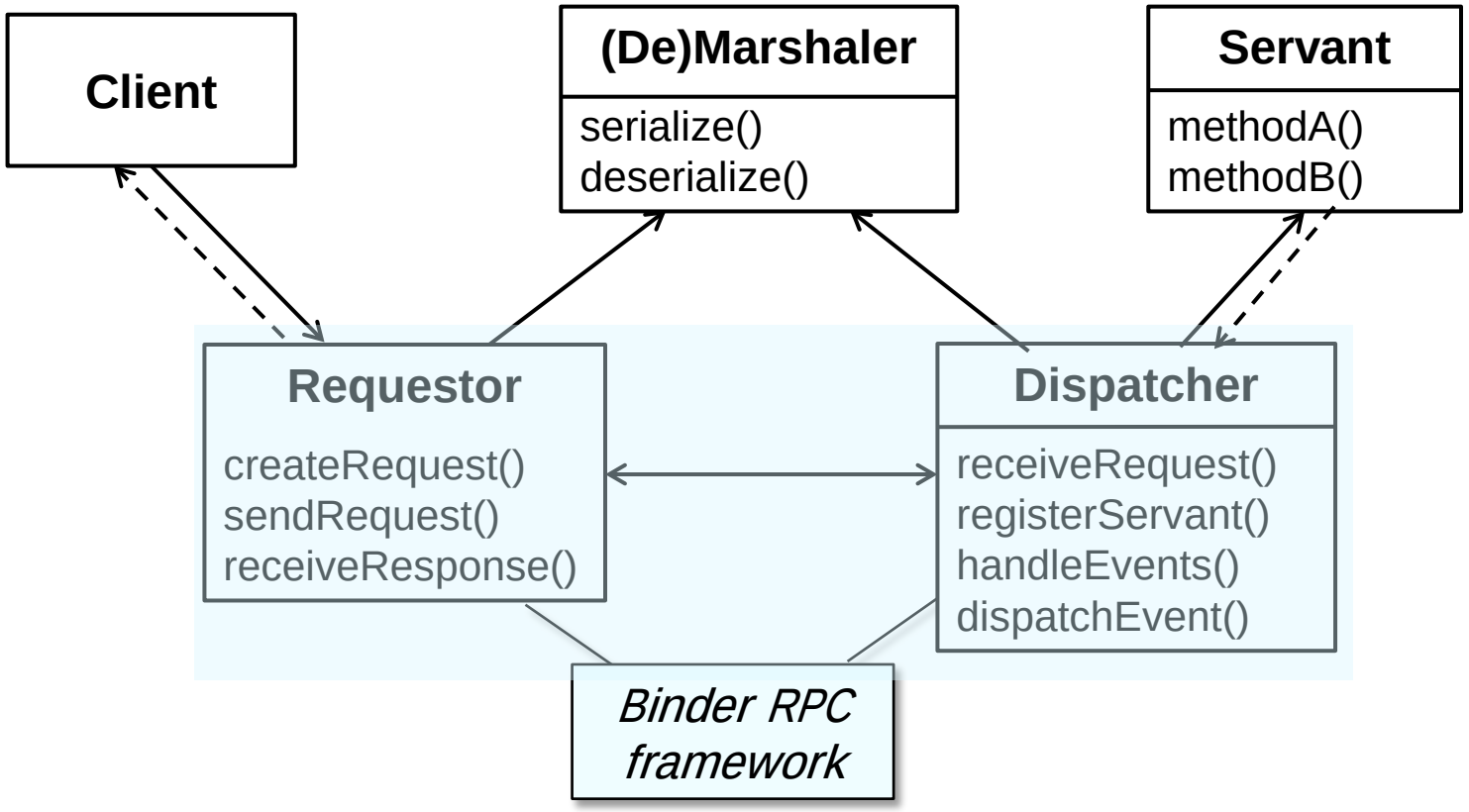
Broker POSA1 Architectural Pattern

Structure & Participants



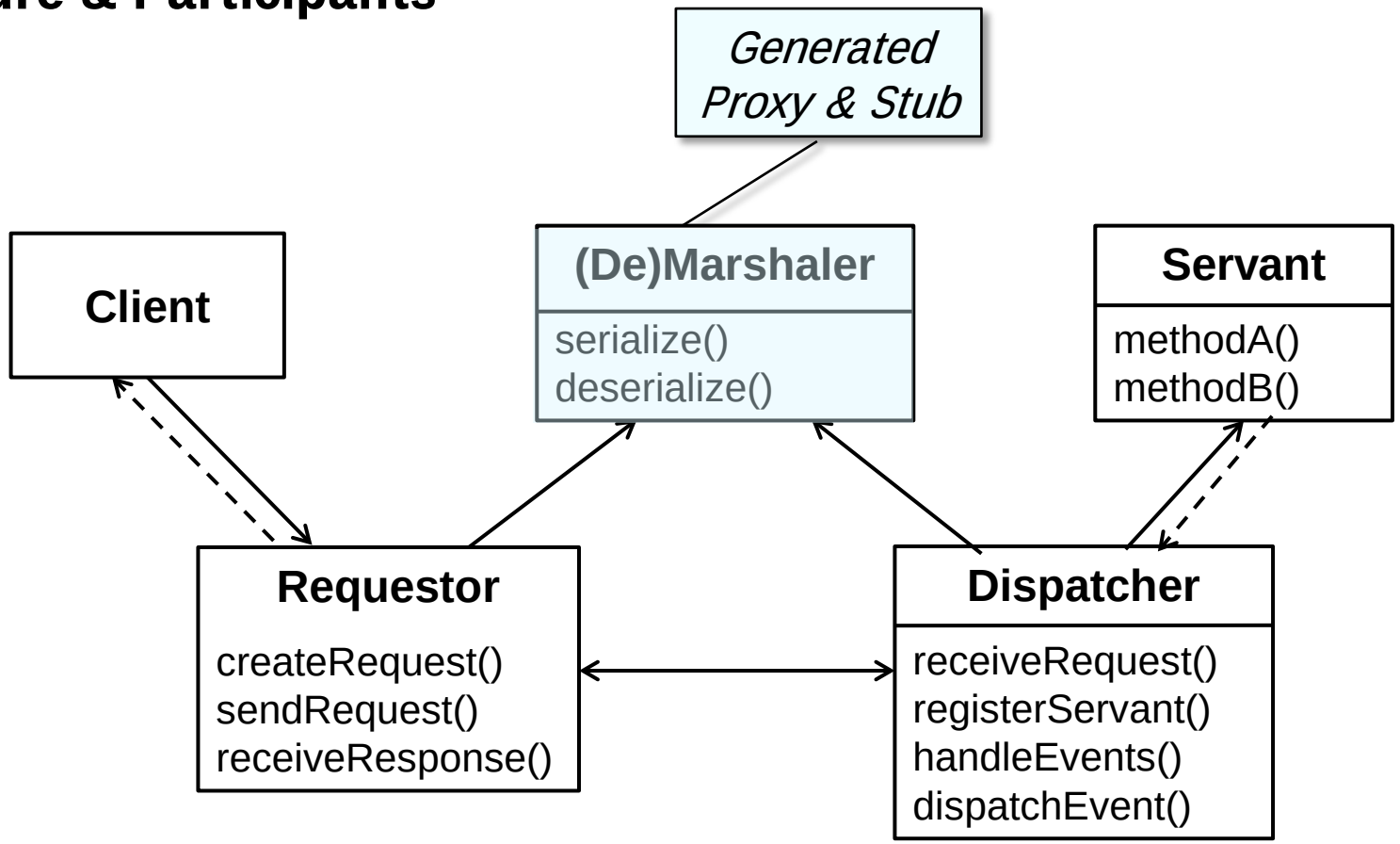
Broker POSA1 Architectural Pattern

Structure & Participants



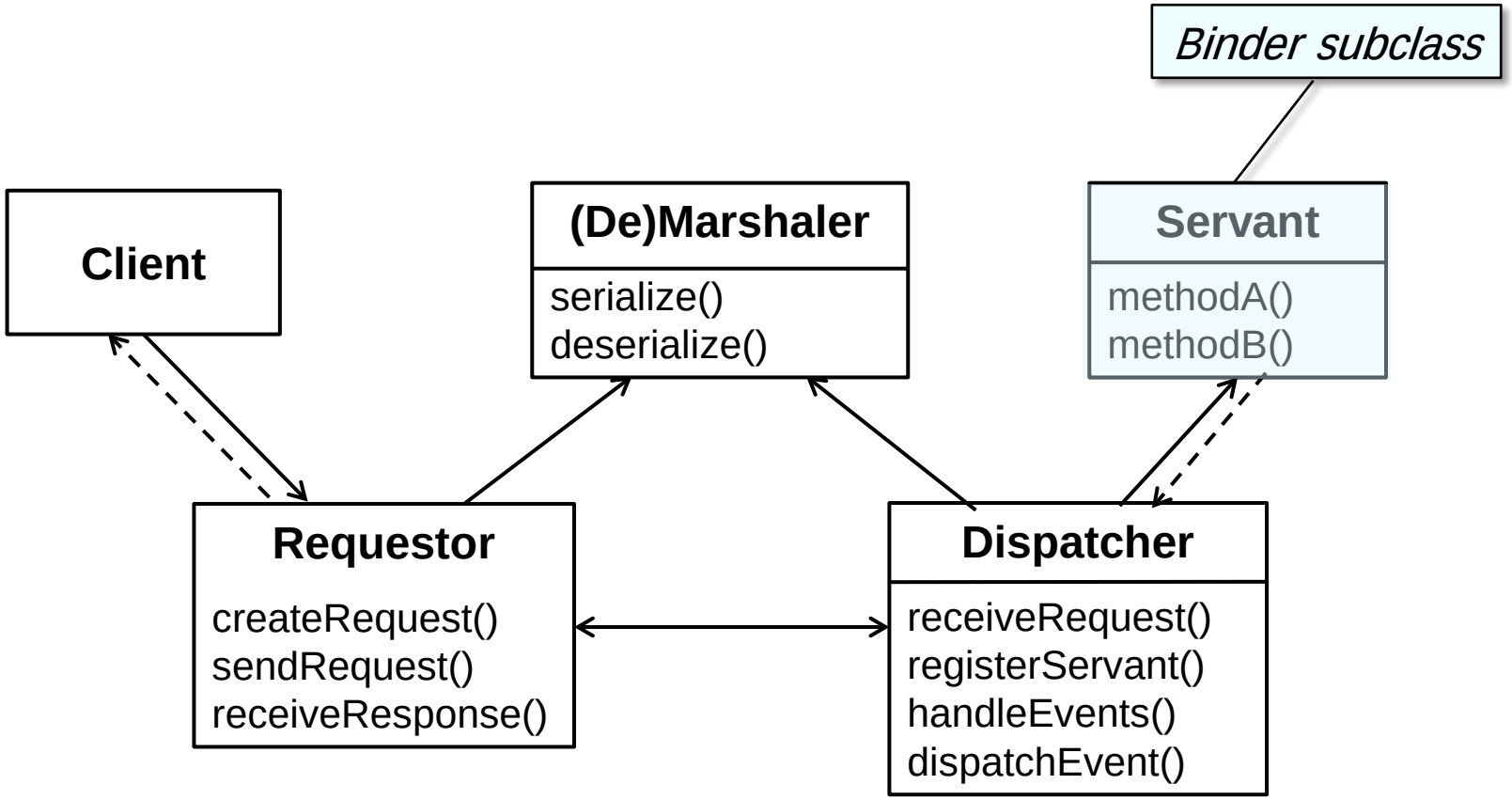
Broker POSA1 Architectural Pattern

Structure & Participants



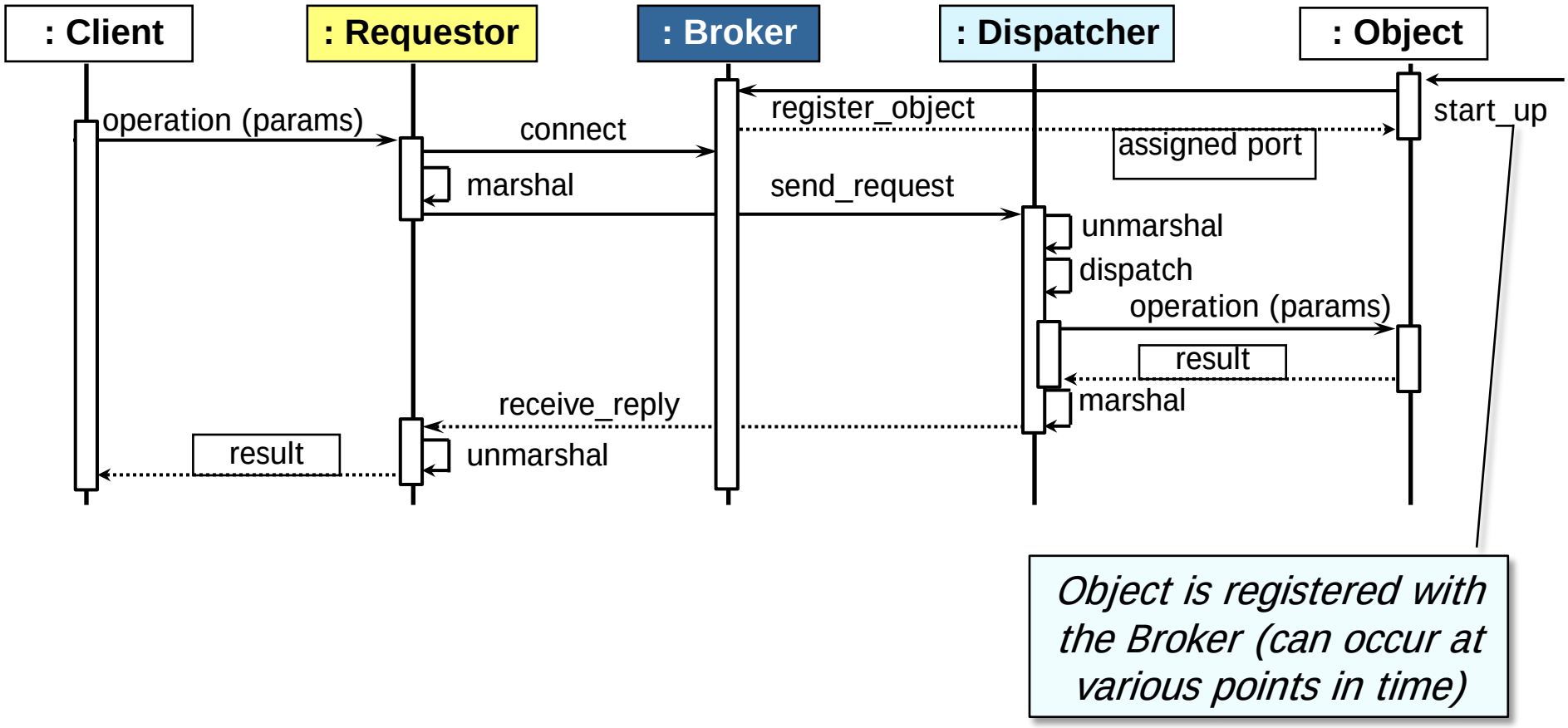
Broker POSA1 Architectural Pattern

Structure & Participants



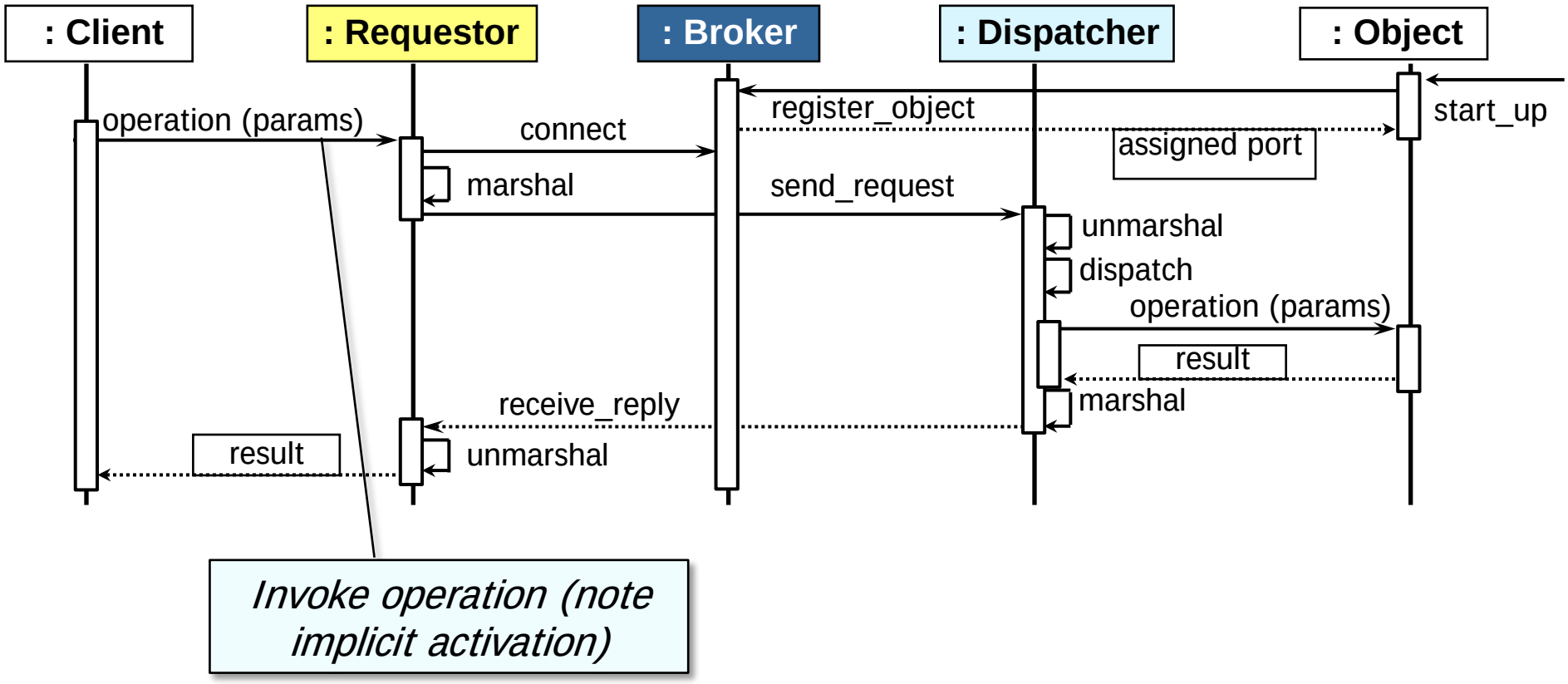
Broker POSA1 Architectural Pattern

Dynamics



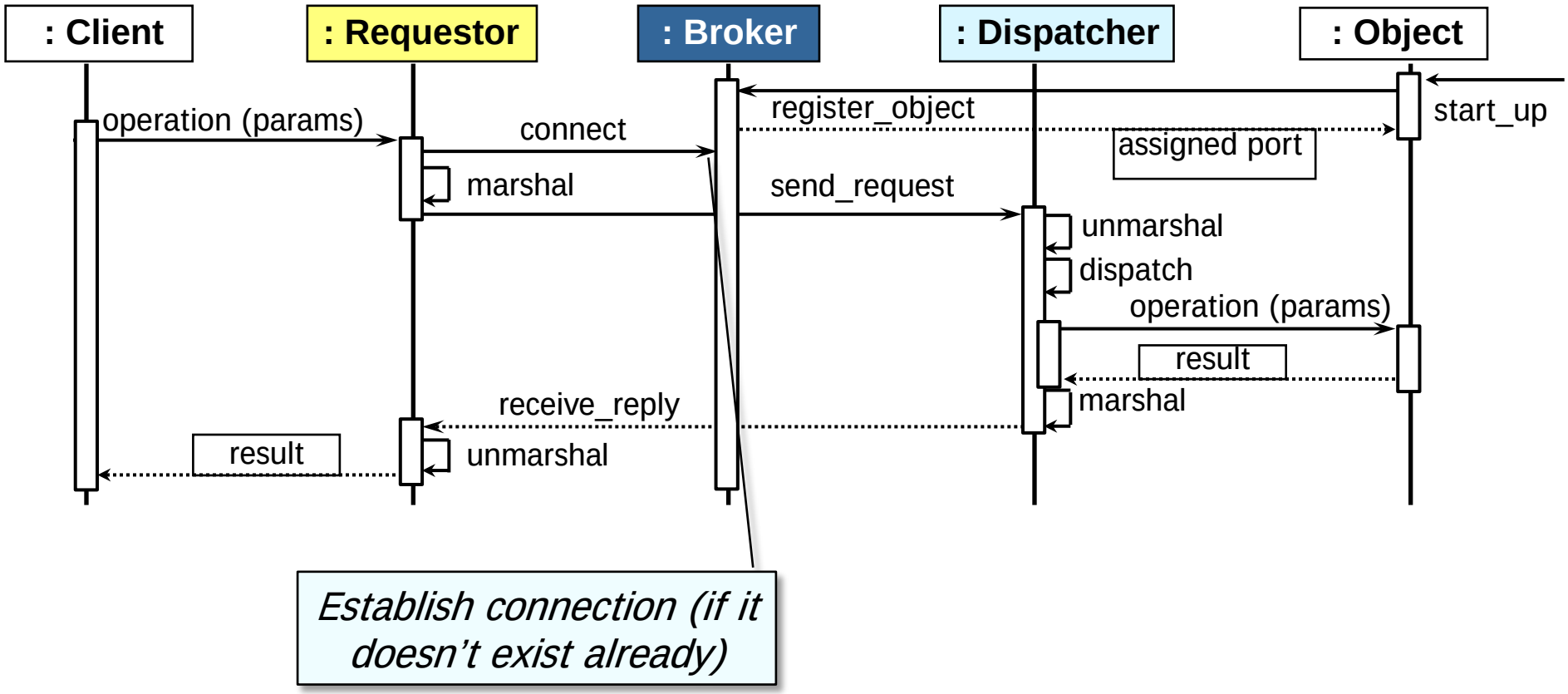
Broker POSA1 Architectural Pattern

Dynamics



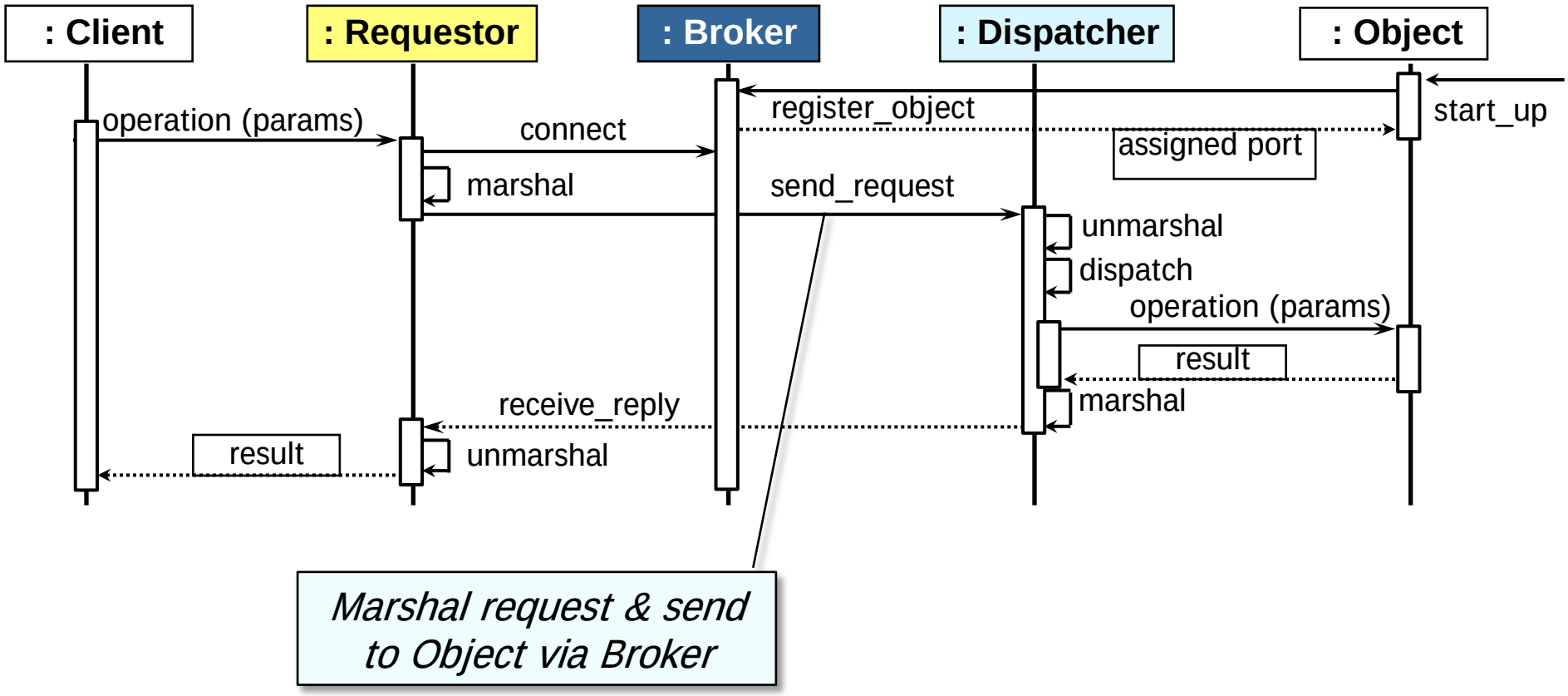
Broker POSA1 Architectural Pattern

Dynamics



Broker POSA1 Architectural Pattern

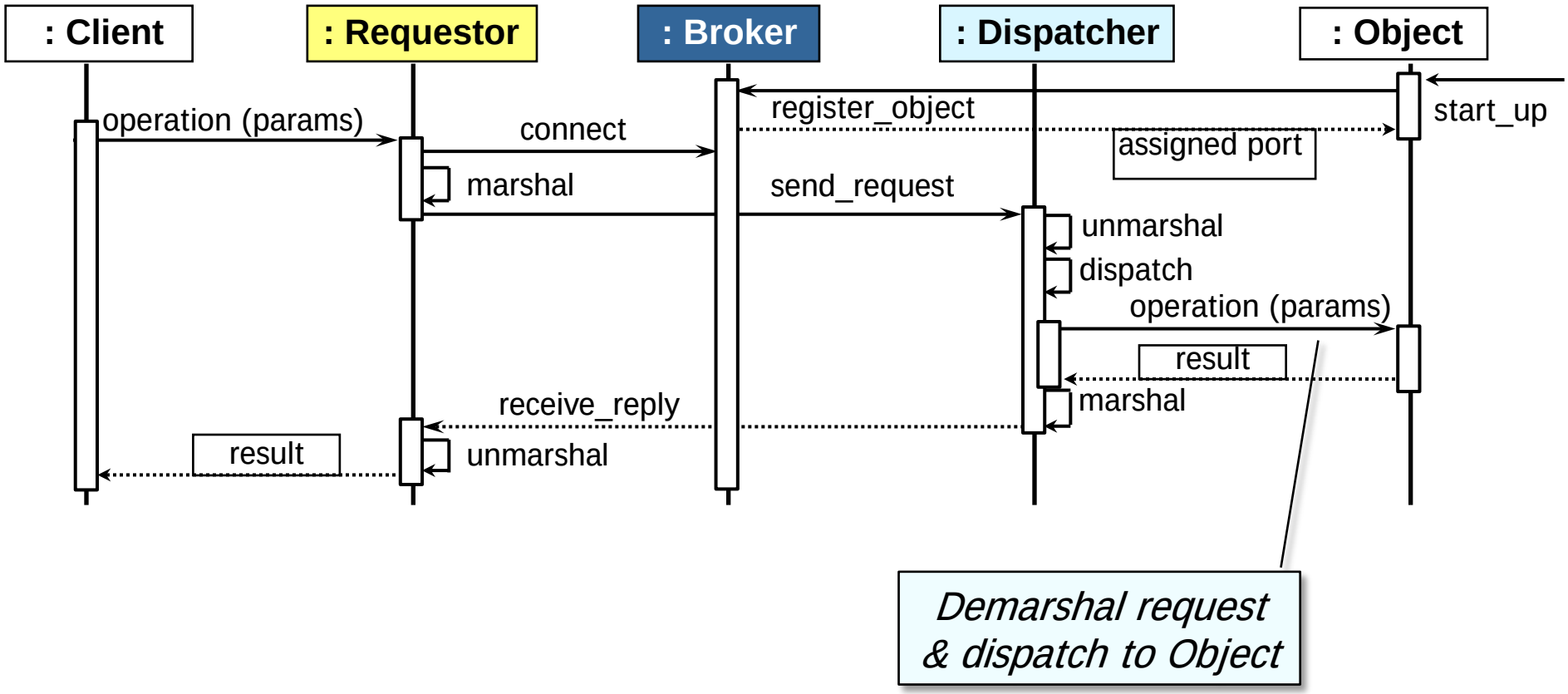
Dynamics



A Broker might or might not run in a separate process or thread

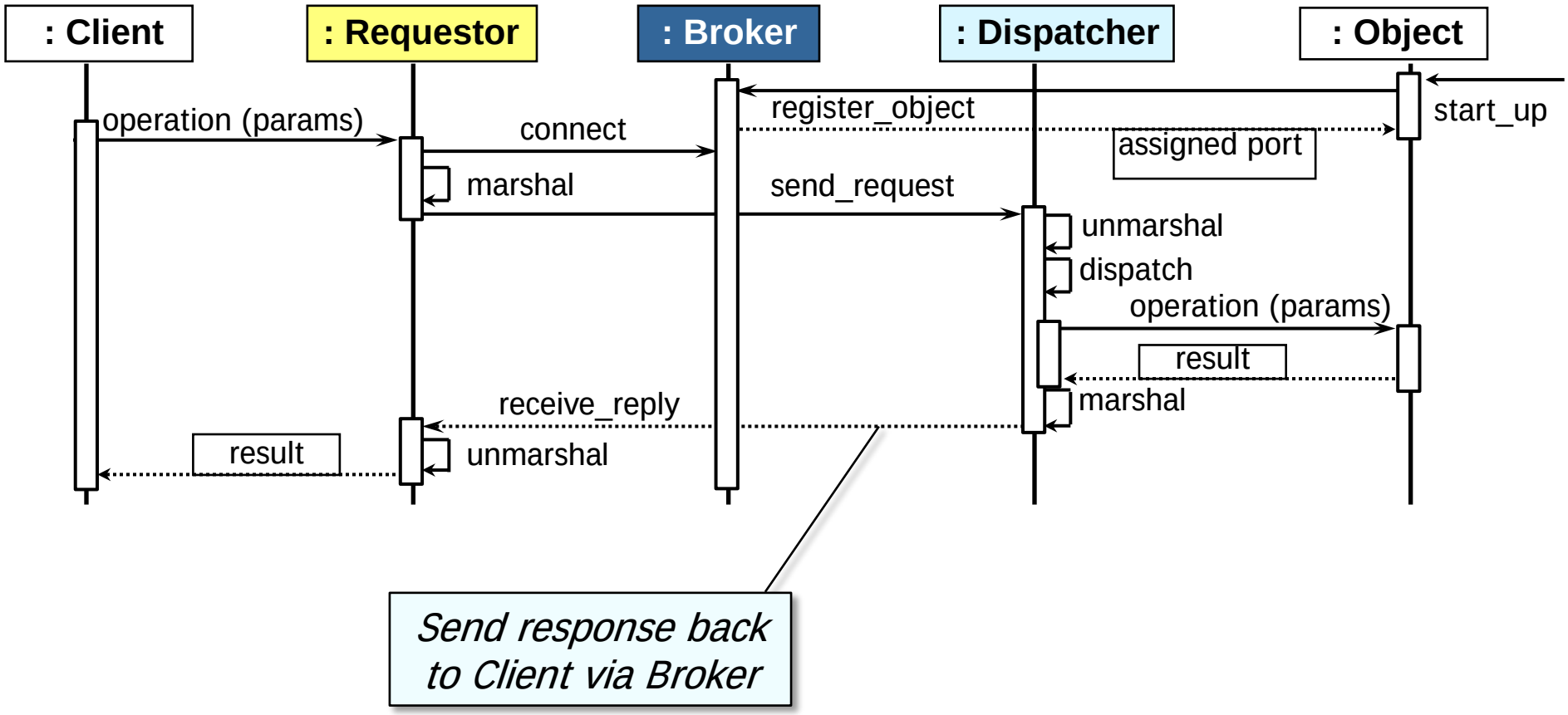
Broker POSA1 Architectural Pattern

Dynamics



Broker POSA1 Architectural Pattern

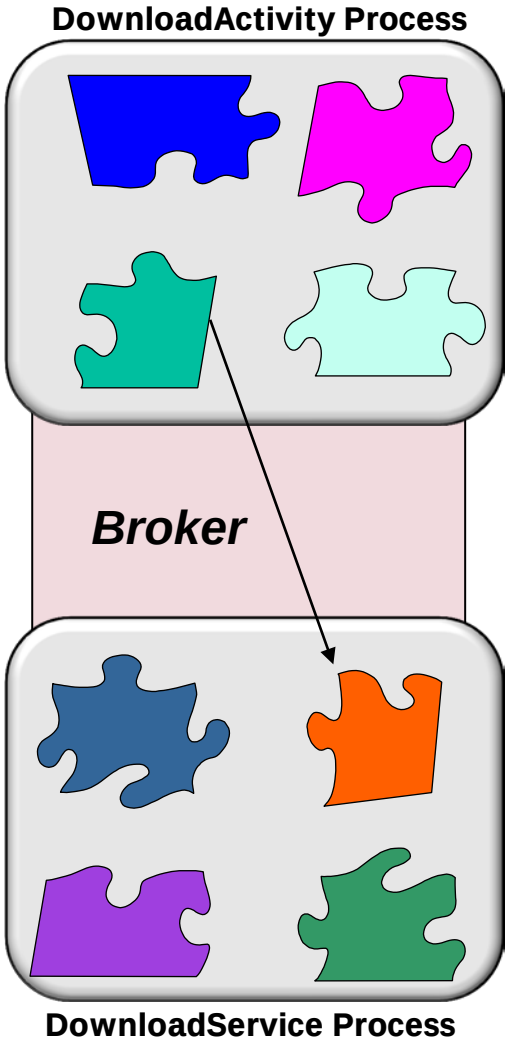
Dynamics



Broker POSA1 Architectural Pattern

Consequences

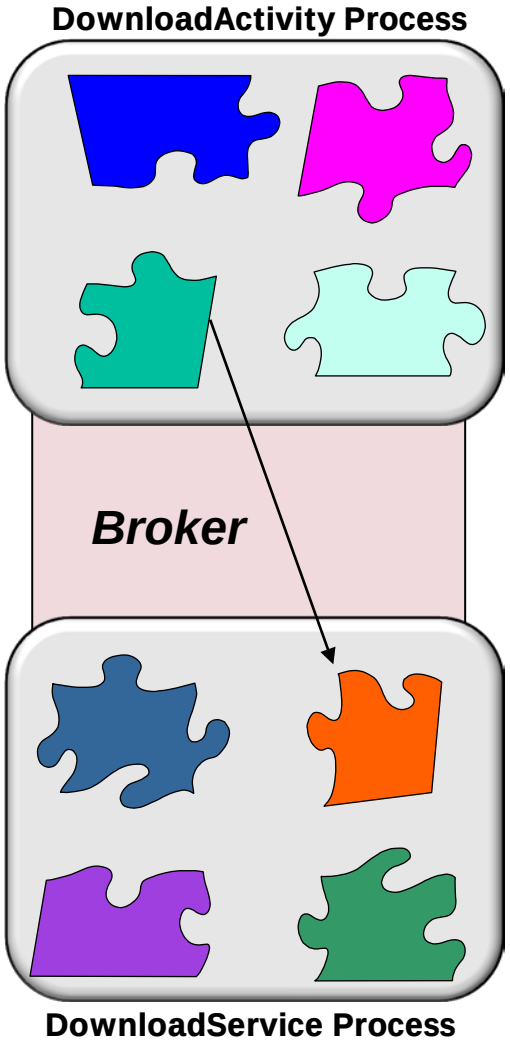
+ Location independence
  - A broker is responsible for locating servers, so clients need not know where servers are located



Broker POSA1 Architectural Pattern

Consequences

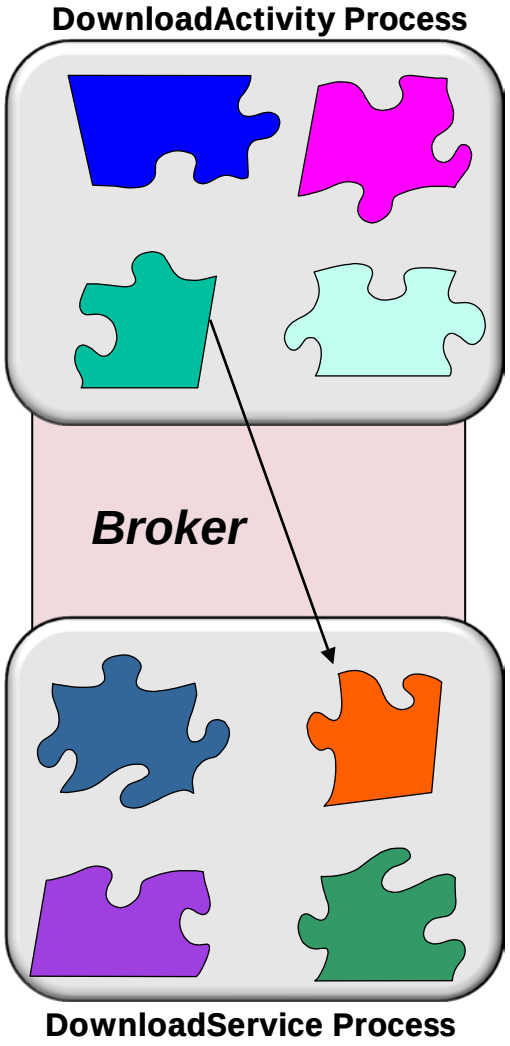
+ Location independence
+ Separation of concerns
  - If server implementations change without affecting interfaces clients should not be affected



Broker POSA1 Architectural Pattern

Consequences

+ Location independence
+ Separation of concerns
+ Portability, modularity, reusability, etc.
  - A broker hides OS & network details from clients & servers via indirection & abstraction layers
    - e.g., APIs, proxies, adapters, bridges, wrapper facades, etc.

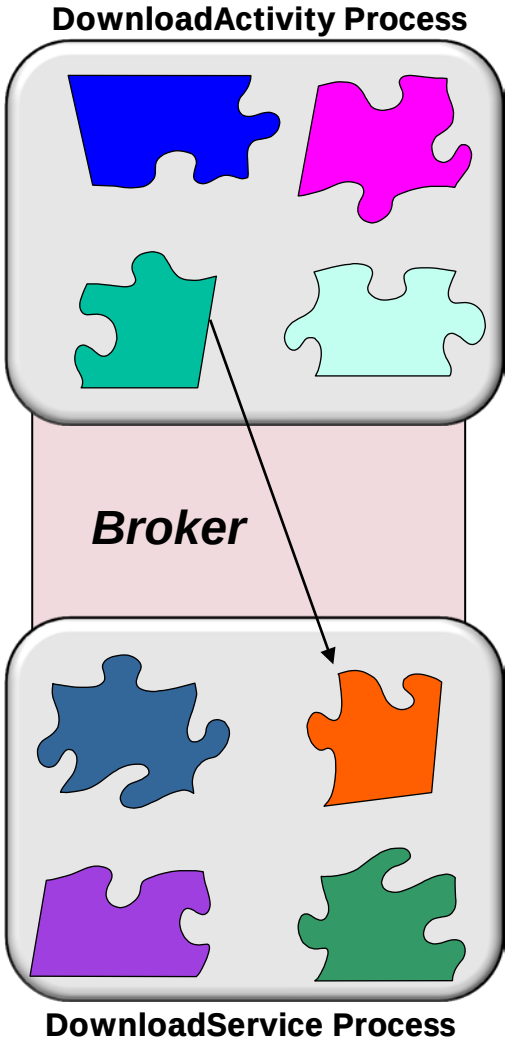


Broker POSA1 Architectural Pattern

Consequences

– Additional time & space overhead

- Applications using brokers may be slower than applications written manually



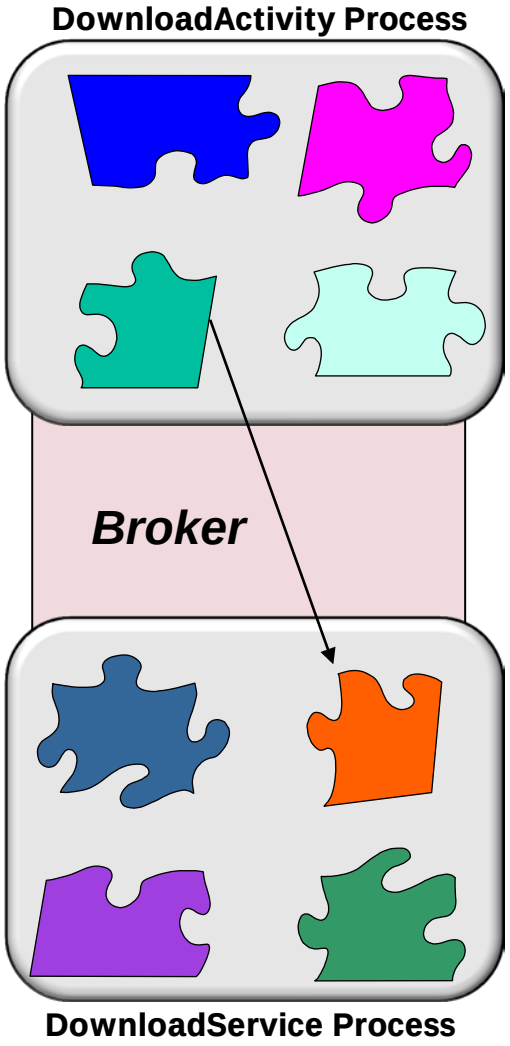
Broker POSA1 Architectural Pattern

Consequences

– Additional time & space overhead

– Potentially less reliable

- Compared with non-distributed software applications, distributed broker systems may incur lower fault tolerance



Broker POSA1 Architectural Pattern

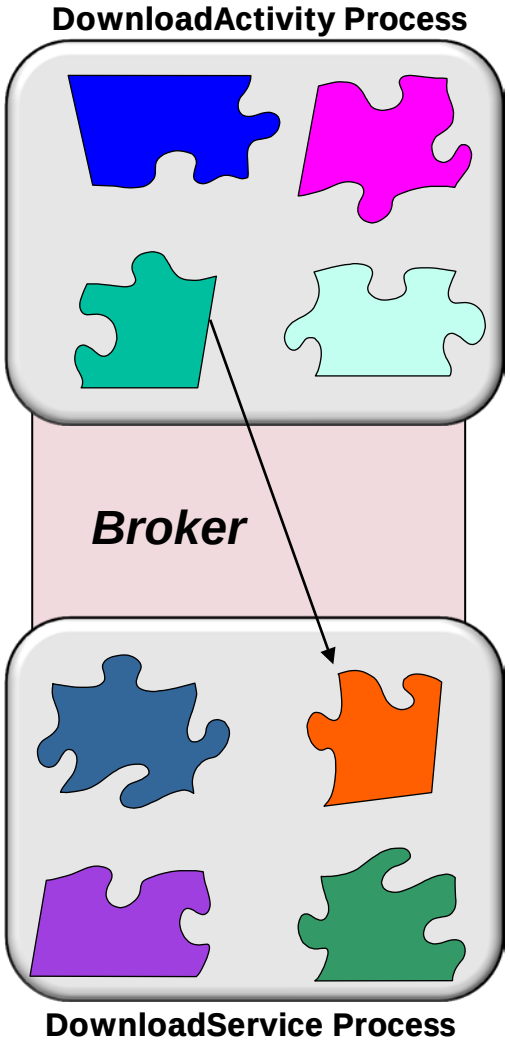
Consequences

– Additional time & space overhead

– Potentially less reliable

– May complicate debugging & testing

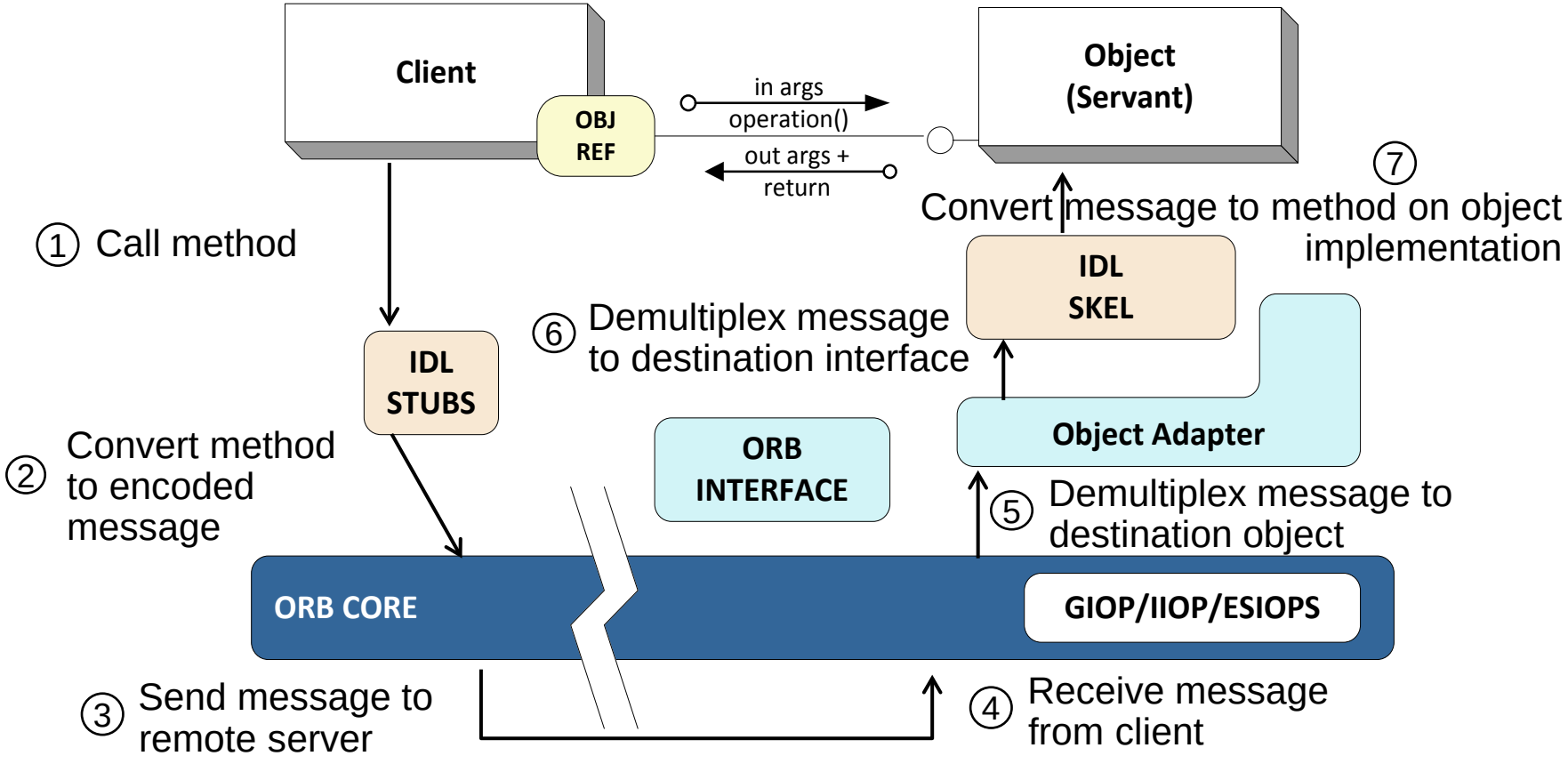
- Testing & debugging of distributed systems is tedious because of all the components involved



Broker POSA1 Architectural Pattern

Known Uses

- Distributed object computing middleware
  - e.g., OMG Common Object Request Broker Architecture (CORBA)

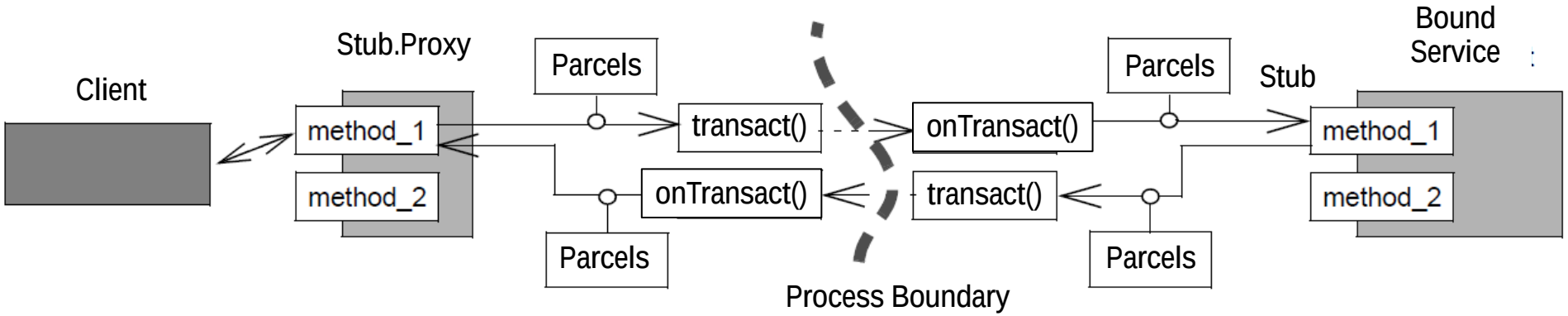


CORBA shields apps from diverse languages, operating systems, networks, etc.

Broker POSA1 Architectural Pattern

Known Uses

- Distributed object computing middleware
- Local RPC middleware on smartphones
  - e.g., Android Binder



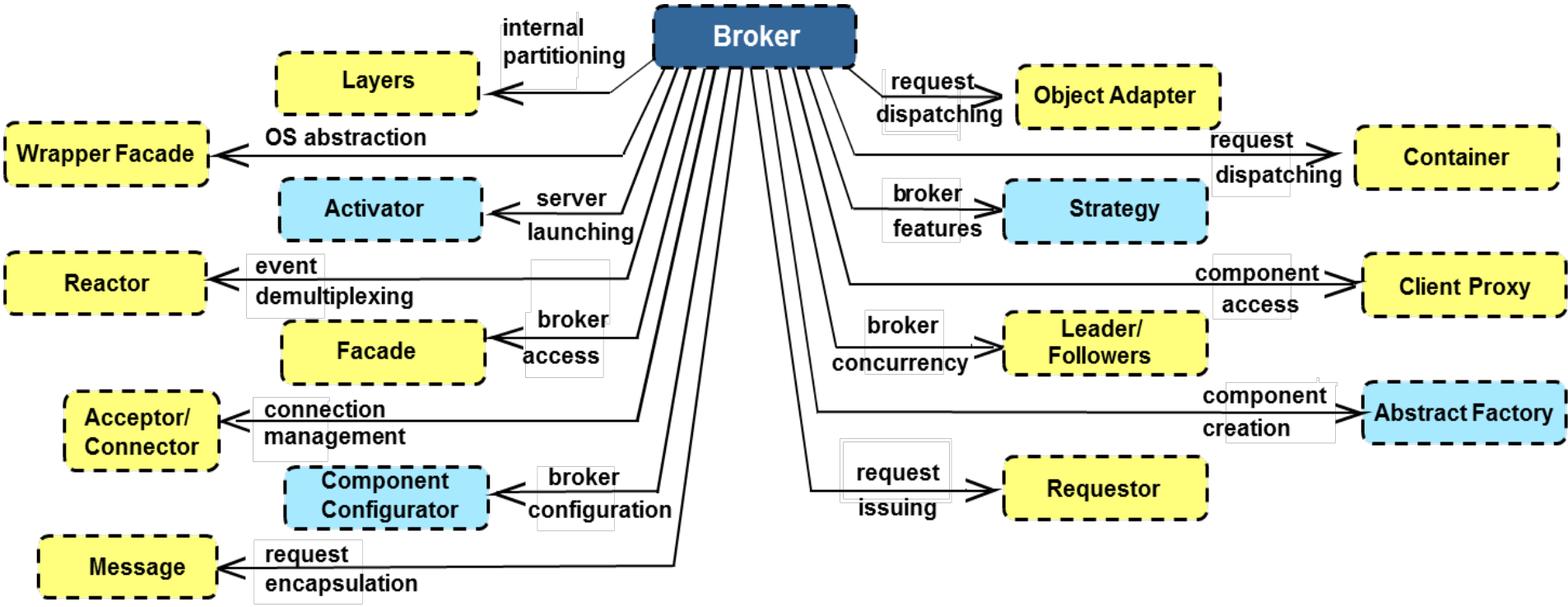
en.wikipedia.org/wiki/OpenBinder

Summary

- *Broker* is widely used in concurrent & networked software
  - Many open-source brokers available
    - e.g., TAO in C++ (www.dre.vanderbit.edu/TAO) & JacORB in Java (www.jacorb.org)

Summary

- *Broker* is widely used in concurrent & networked software
- *Broker* is more than an architecture pattern—it's actually a pattern language that is composed of over a dozen patterns



www.voelter.de/publications/books-rem.html has pattern language for *Broker*