Distributed Computing

Paradigms of Distributed Computing

Distributed computing paradigms

Paradigms for Distributed Applications

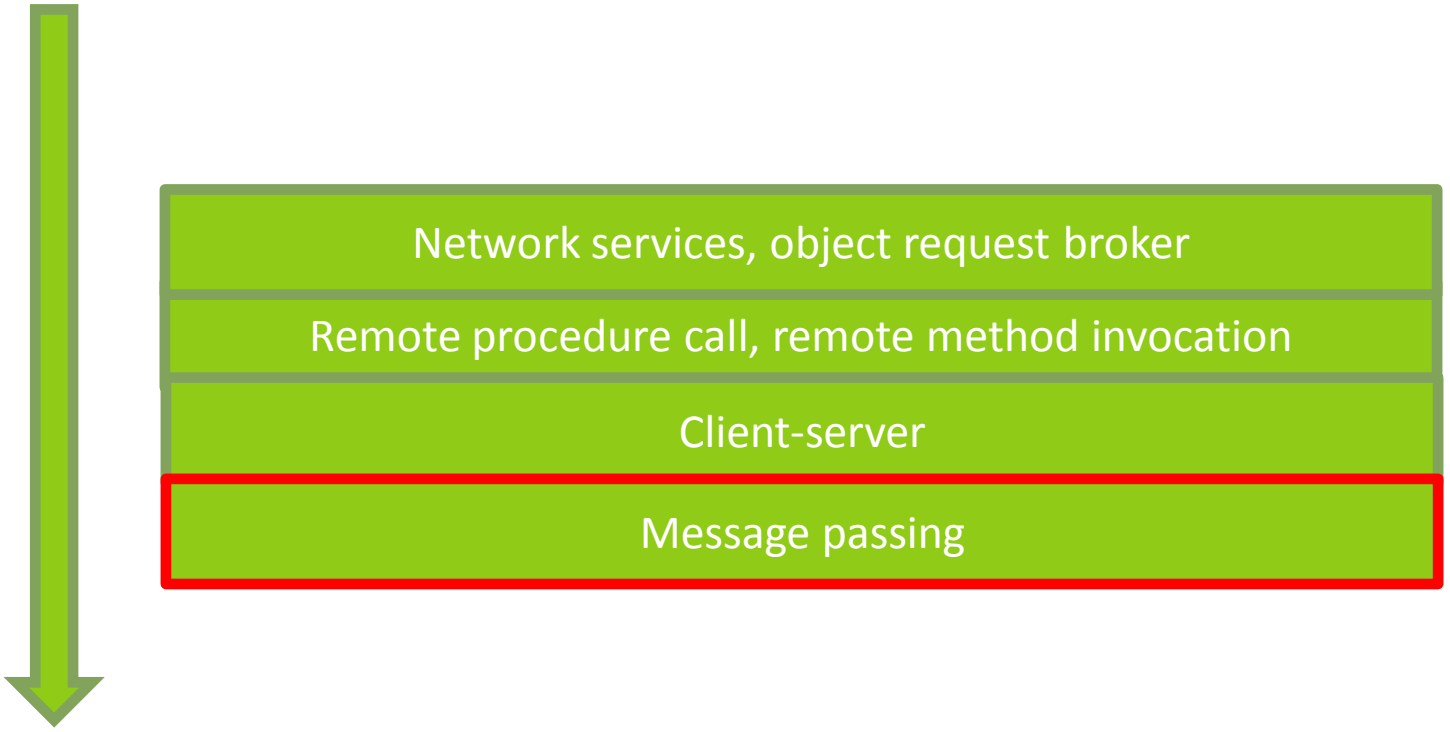
- It is useful to identify the basic patterns or models of distributed applications, and classify the detail according to these models.
- Characteristics that distinguish distributed applications from conventional applications running on a single machine are:
  - *Interprocess communication*: A distributed application require the participation of two or more independent entities (processes). To do so, the processes must have the ability to exchange data among themselves.
  - *Event synchronization*: In a distributed application, the sending and receiving of data among the participants of a distributed application must be synchronized.

Distributed Application Paradigms

Level of abstraction

high

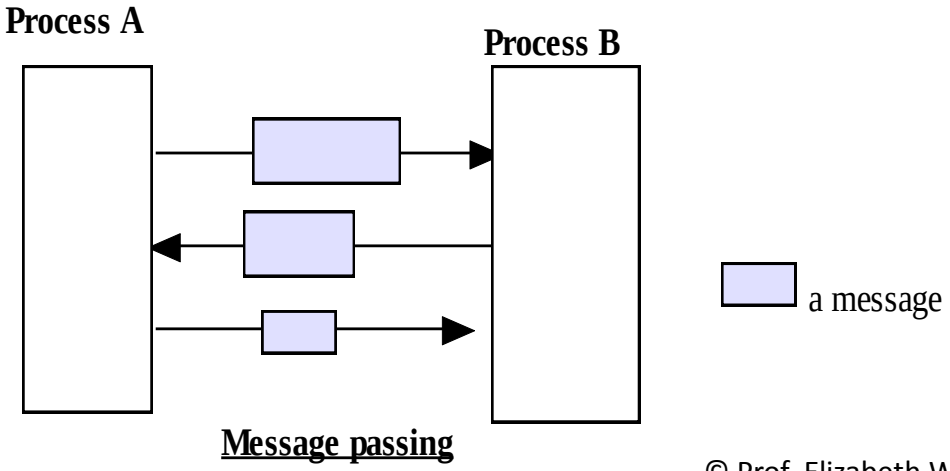
low



The Message Passing Paradigm

Message passing is the most fundamental paradigm for distributed applications.

- A process sends a message, often representing a request.
- The message is delivered to a receiver, which processes the message, and possibly sending a message in response.
- In turn, the reply may trigger a further request, which leads to a subsequent reply, and so forth.

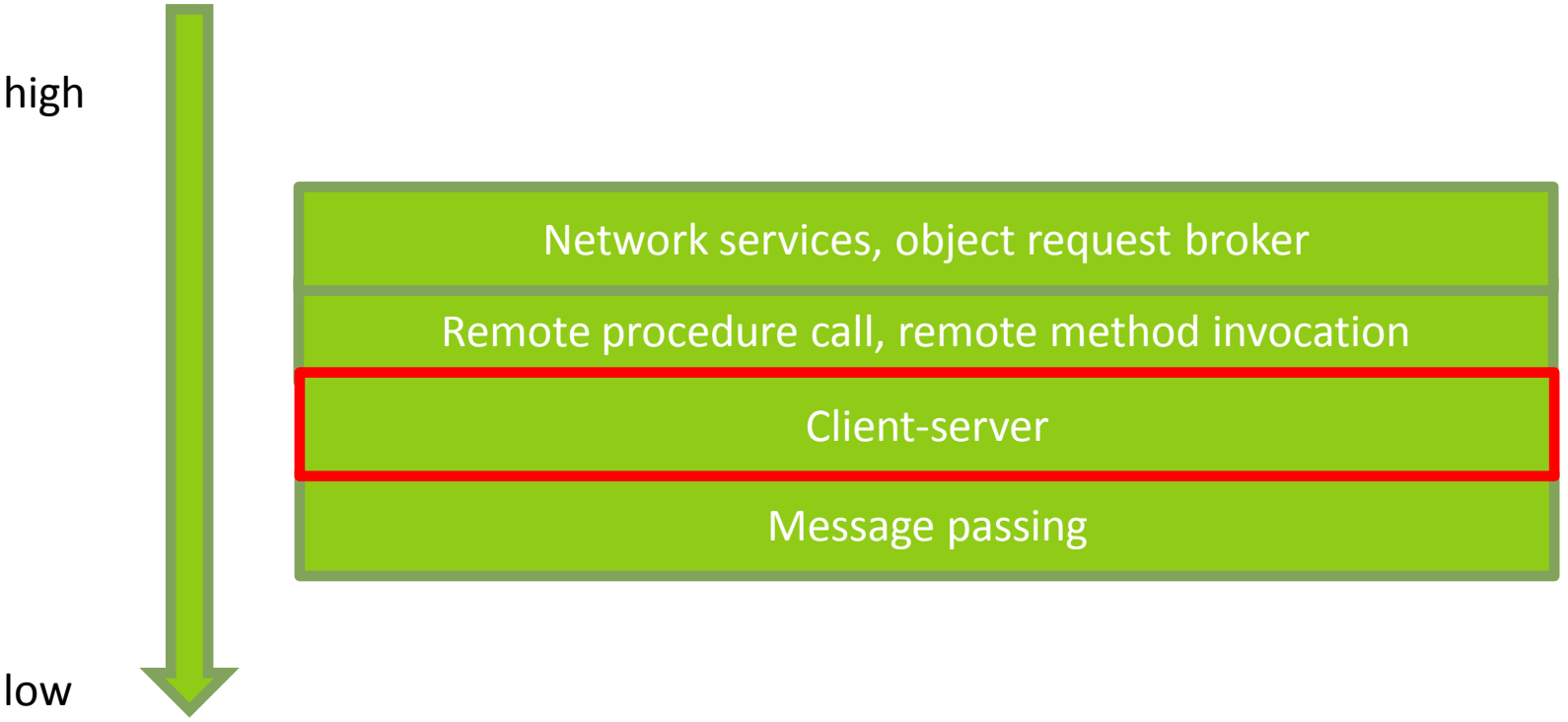


The Message Passing Paradigm - 2

- The basic operations required to support the basic message passing paradigm are *send*, and *receive*.
- For connection-oriented communication, the operations *connect* and *disconnect* are also required.
- With the abstraction provided by this model, the interconnected processes perform input and output to each other, in a manner similar to file I/O. The I/O operations encapsulate the details of network communication at the operating-system level.
- The socket application programming interface is based on this paradigm.
  - http://java.sun.com/products/jdk/1.2/docs/api/index.html
  - http://www.sockets.com/

Distributed Application Paradigms

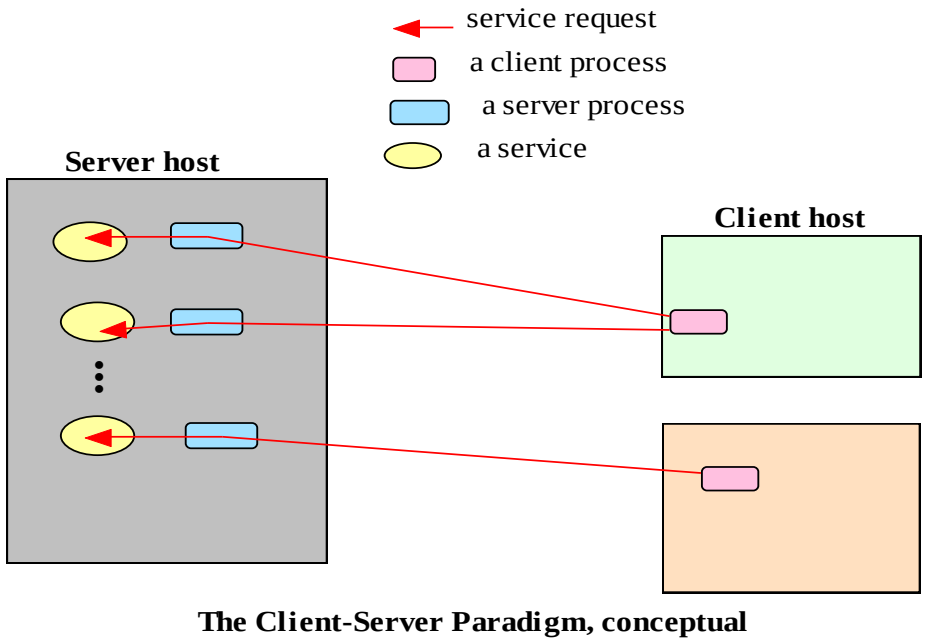
Level of abstraction



THE CLIENT-SERVER PARADIGM

Best known paradigm for network applications - the client-server model assigns asymmetric roles to two collaborating processes.

One process, the server, plays the role of a service provider which waits passively for the arrival of requests. The other, the client, issues specific requests to the server and awaits its response.



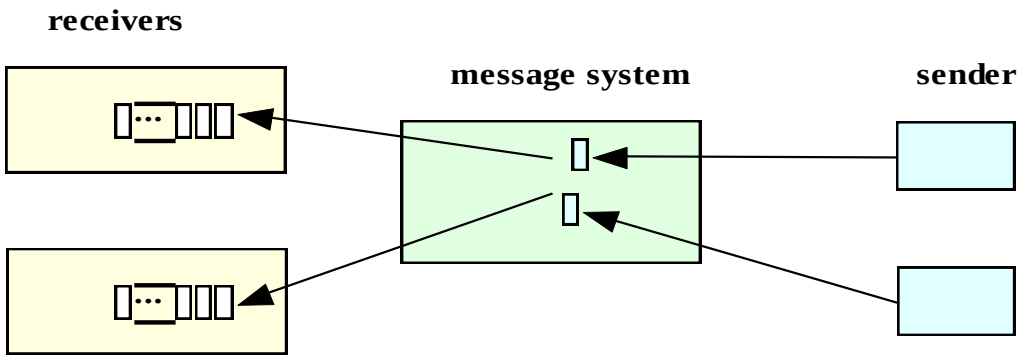
The Client-Server Paradigm, conceptual

The Client-Server Paradigm - 2

- Simple in concept, the client-server model provides an efficient abstraction for the delivery of services.
- Operations required include those for a server process to listen for and to accept requests, and for a client process to issue requests and accept responses.
- By assigning asymmetric roles to the two sides, event synchronization is simplified: the server process waits for requests, and the client in turn waits for responses.
- Many Internet services are client-server applications. These services are often known by the protocol that the application implements. Well known Internet services include HTTP, FTP, DNS, finger, gopher, etc.

The Message System Paradigm

- The Message System or Message-Oriented Middleware (MOM) paradigm is an elaboration of the basic message-passing paradigm.
- In this paradigm, a message system serves as an intermediary among separate, independent processes.
- The message system acts as a switch for messages, through which processes exchange messages asynchronously, in a decoupled manner.
- A sender deposits a message with the message system, which forwards it to a message queue associated with each receiver. Once a message is sent, the sender is free to move on to other tasks.



The Point-To-Point Message Model

- In this model, a message system forwards a message from the sender to the receiver's message queue. Unlike the basic message passing model, the middleware provides a message depository, and allows the sending and the receiving to be decoupled. Via the middleware, a sender deposits a message in the message queue of the receiving process. A receiving process extracts the messages from its message queue, and handles each one accordingly.

- Compared to the basic message-passing model, this paradigm provides the additional abstraction for asynchronous operations. To achieve the same effect with basic message-passing, a developer will have to make use of threads or child processes.

The Publish/Subscribe Message Model

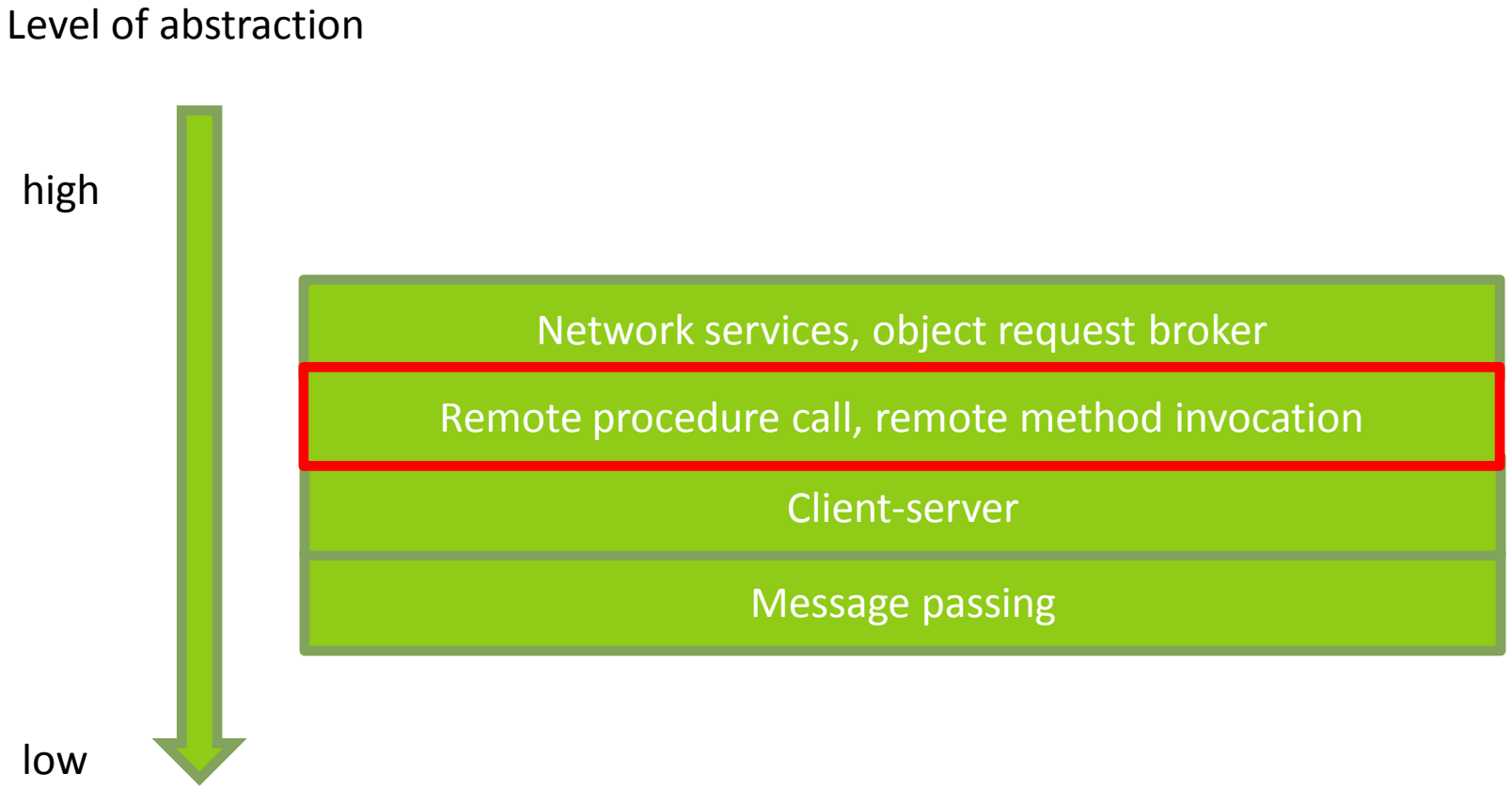
- In this model, each message is associated with a specific topic or event. Applications interested in the occurrence of a specific event may subscribe to messages for that event. When the awaited event occurs, the process publishes a message announcing the event or topic. The middleware message system distributes the message to all its subscribers.

- The publish/subscribe message model offers a powerful abstraction for multicasting or group communication. The *publish* operation allows a process to multicast to a group of processes, and the *subscribe* operation allows a process to listen for such multicast.

Toolkits based on the Message-System Paradigm

- The MOM paradigm has had a long history in distributed applications.
- Message Queue Services (MQS) have been in use since the 1980's.
- The IBM MQ*Series is an example of such a facility.
- Other existing support for this paradigm are
  - Microsoft's Message Queue (MSQ),
  - RabbitMQ (http://www.rabbitmq.com/)
    - Robust messaging for applications
    - Easy to use
    - Runs on all major operating systems
    - Supports a huge number of developer platforms
    - Open source and commercially supported

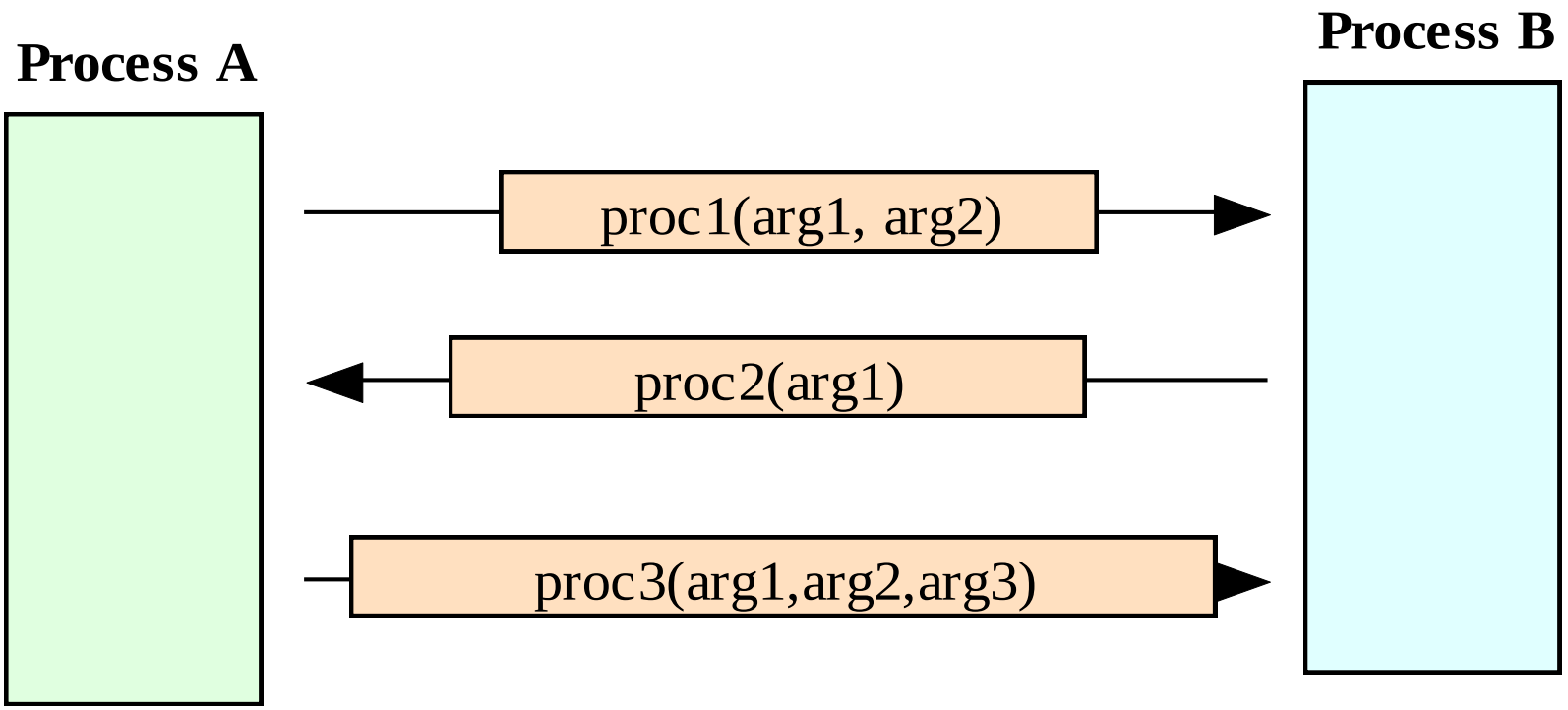
Distributed Application Paradigms



Remote Procedure Call

- As applications grew increasingly complex, it became desirable to have a paradigm which allows distributed software to be programmed in a manner similar to conventional applications which run on a single processor.
- The Remote Procedure Call (RPC) model provides such an abstraction. Using this model, interprocess communications proceed as procedure, or function, calls, which are familiar to application programmers.
- A remote procedure call involves two independent processes, which may reside on separate machines. A process, *A*, wishing to make a request to another process, *B*, issues a procedure call to *B*, passing with the call a list of argument values. As in the case of local procedure calls, a remote procedure call triggers a predefined action in a procedure provided by process *B*. At the completion of the procedure, process *B* returns a value to process *A*.

Remote Procedure Call - 2



Remote Procedure Call - 3

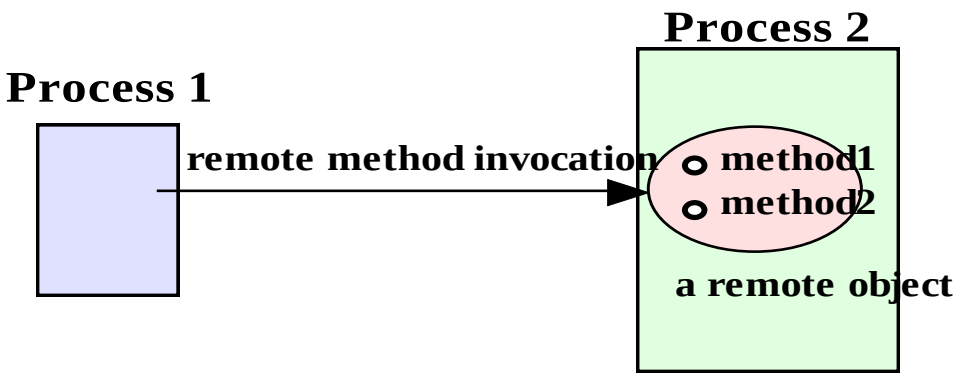
- RPC allows programmers to build network applications using a programming construct similar to the local procedure call, providing a convenient abstraction for both interprocess communication and event synchronization.
- Since its introduction in the early 1980s, the Remote Procedure Call model has been widely in use in network applications.
- There are two prevalent APIs for Remote Procedure Calls.
  - The *Open Network Computing Remote Procedure Call*, evolved from the RPC API originated from Sun Microsystems in the early 1980s.
  - The *Open Group Distributed Computing Environment* (DCE) RPC.
- Both APIs provide a tool, *rpcgen*, for transforming remote procedure calls to local procedure calls to the stub.

The Distributed Objects Paradigms

- The idea of applying object orientation to distributed applications is a natural extension of object-oriented software development.
- Applications access objects distributed over a network.
- Objects provide methods, through the invocation of which an application obtains access to services.
- Object-oriented paradigms include:
  - Remote method invocation (RMI)
  - Network services
  - Object request broker
  - Object spaces

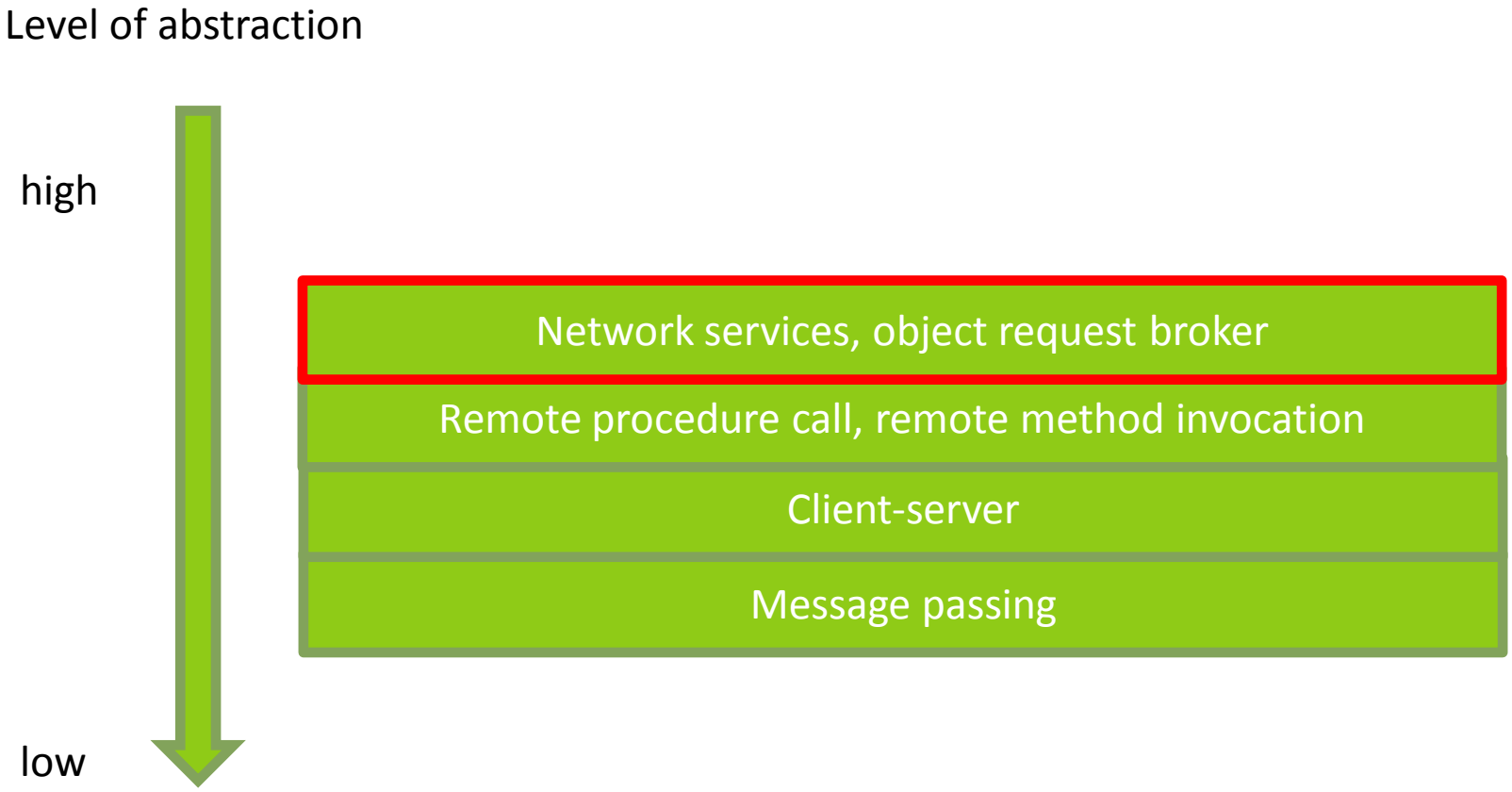
Remote Method Invocation (RMI)

- Remote method invocation is the object-oriented equivalent of remote method calls.
- In this model, a process invokes the methods in an object, which may reside in a remote host.
- As with RPC, arguments may be passed with the invocation.



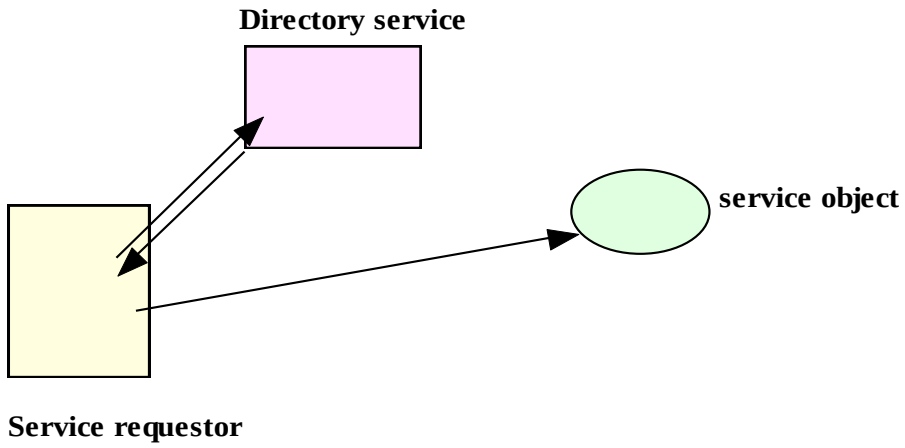
The Remote Method Call Paradigm

Distributed Application Paradigms



The Network Services Paradigm

- In this paradigm, service providers register themselves with directory servers on a network. A process desiring a particular service contacts the directory server at run time, and, if the service is available, will be provided a reference to the service. Using the reference, the process interacts with the service.
- This paradigm is essentially an extension of the remote method call paradigm. The difference is that service objects are registered with a global directory service, allowing them to be look up and accessed by service requestors on a federated network.
- XML Web Services technology is based on this paradigm.



The Object Request broker Paradigm

- In the object broker paradigm , an application issues requests to an *object request broker* (ORB), which directs the request to an appropriate object that provides the desired service.
- The paradigm closely resembles the remote method invocation model in its support for remote object access. The difference is that the object request broker in this paradigm functions as a middleware which allows an application, as an object requestor, to potentially access multiple remote (or local) objects.
- The request broker may also function as an mediator for heterogeneous objects, allowing interactions among objects implemented using different APIs and /or running on different platforms.