RMI REMOTE METHOD INVOCATION



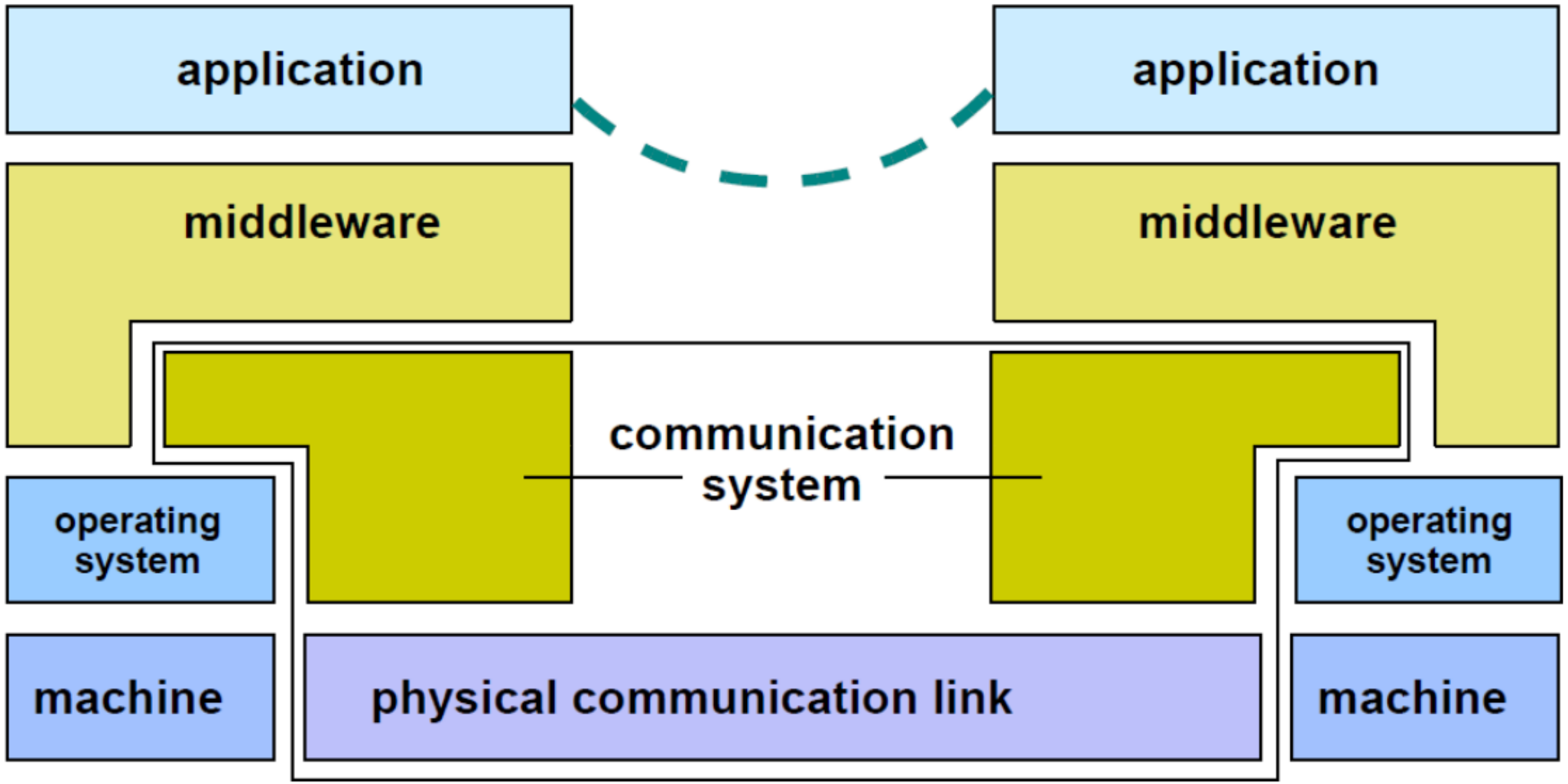
MOTIVATIONS

- Sockets are a simple and flexible technology for data communication in distributed systems
- Sockets are restricted to the transmission of data Sockets leave the semantics of this data unconsidered
- Protocols which provide the semantic interpretation of the data must be developed on the application level
- The development of such protocols is often time-consuming and error-prone

What is a middleware

Middleware is software which lies between an operating system and the applications running on it.

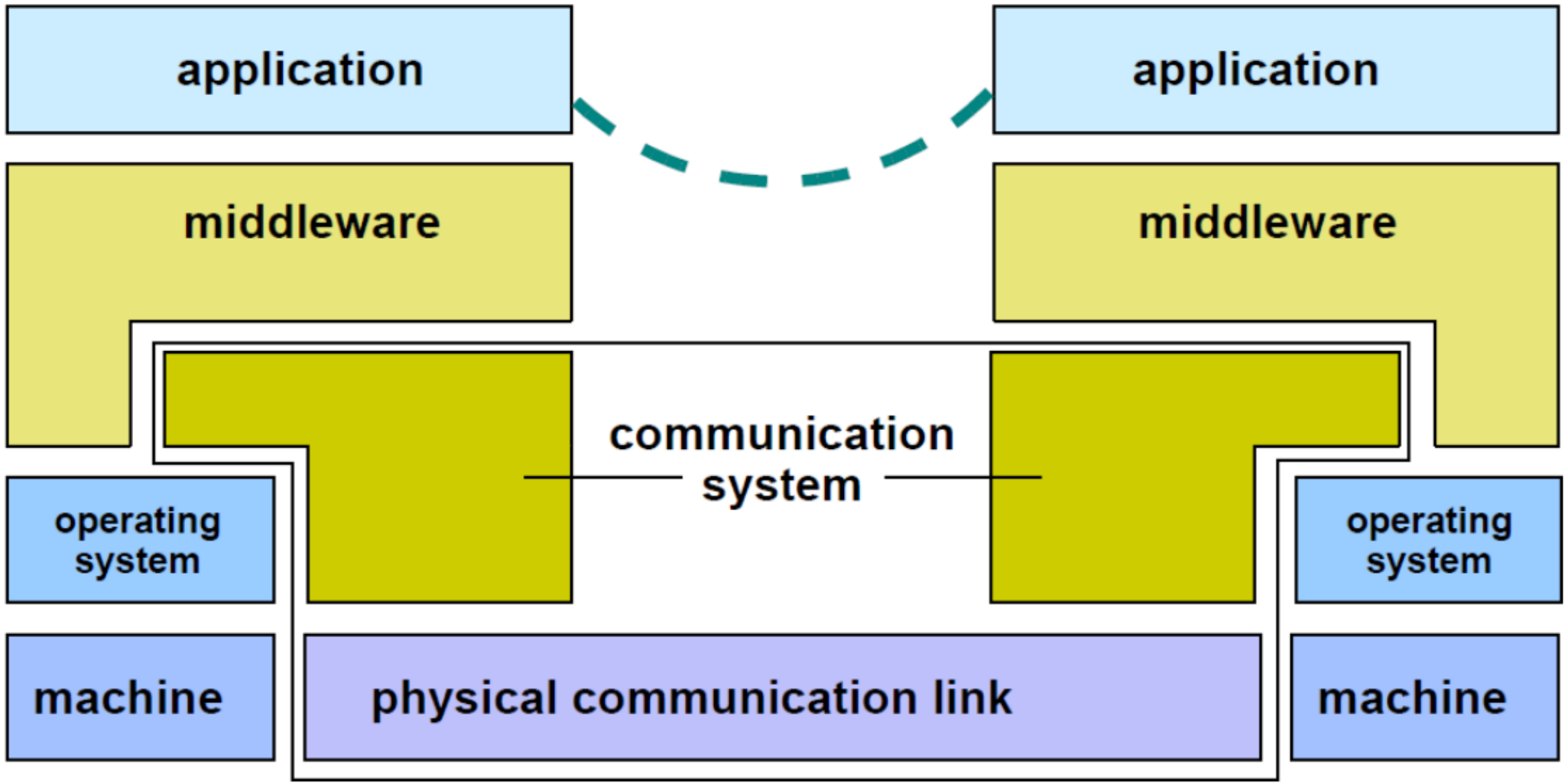
Essentially functioning as hidden translation layer, middleware enables communication and data management for distributed applications.



What is a middleware

Middleware is software which lies between an operating system and the applications running on it.

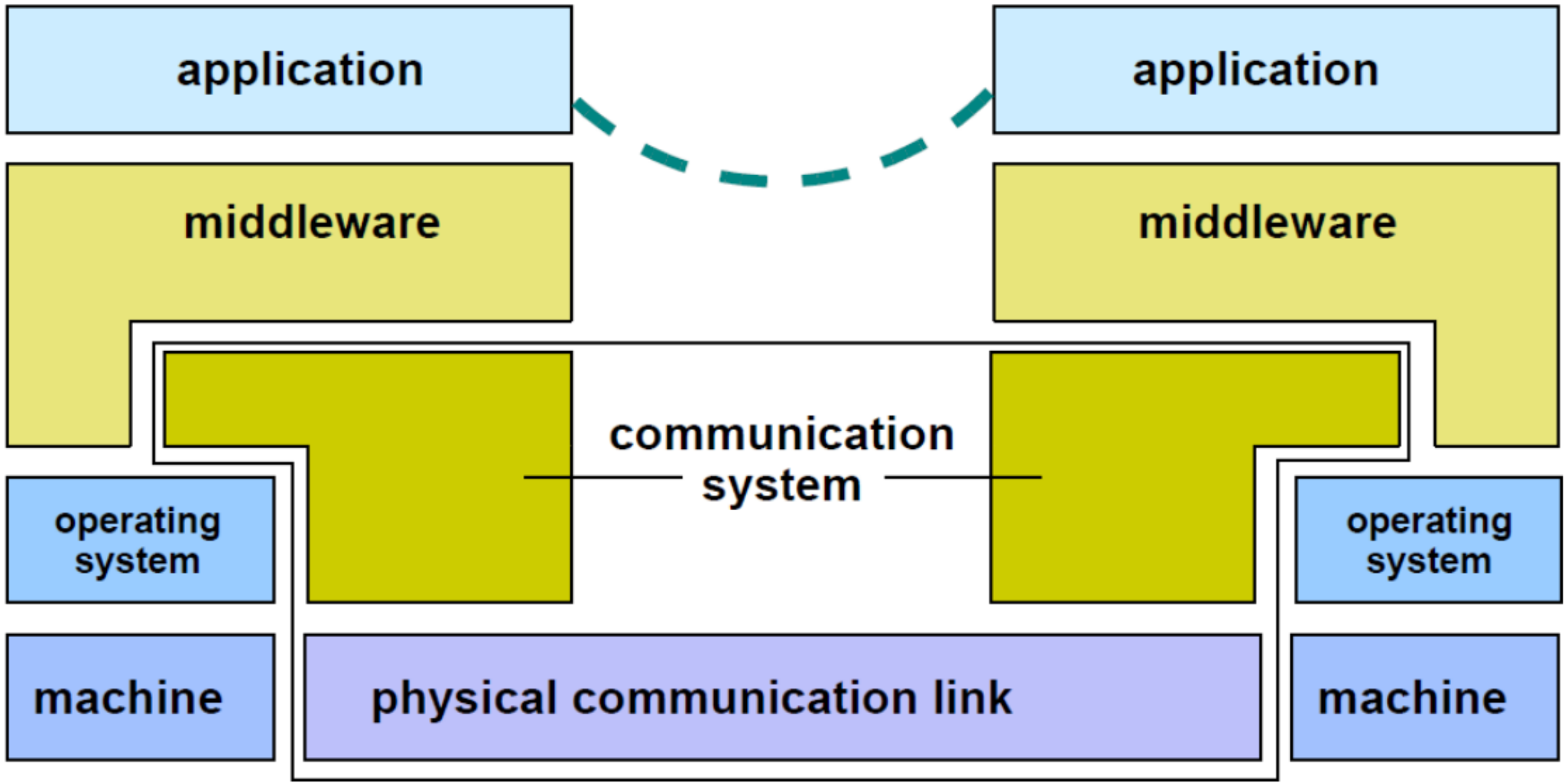
Essentially functioning as hidden translation layer, middleware enables communication and data management for distributed applications.



What is a middleware

Middleware is software which lies between an operating system and the applications running on it.

Essentially functioning as hidden translation layer, middleware enables communication and data management for distributed applications.

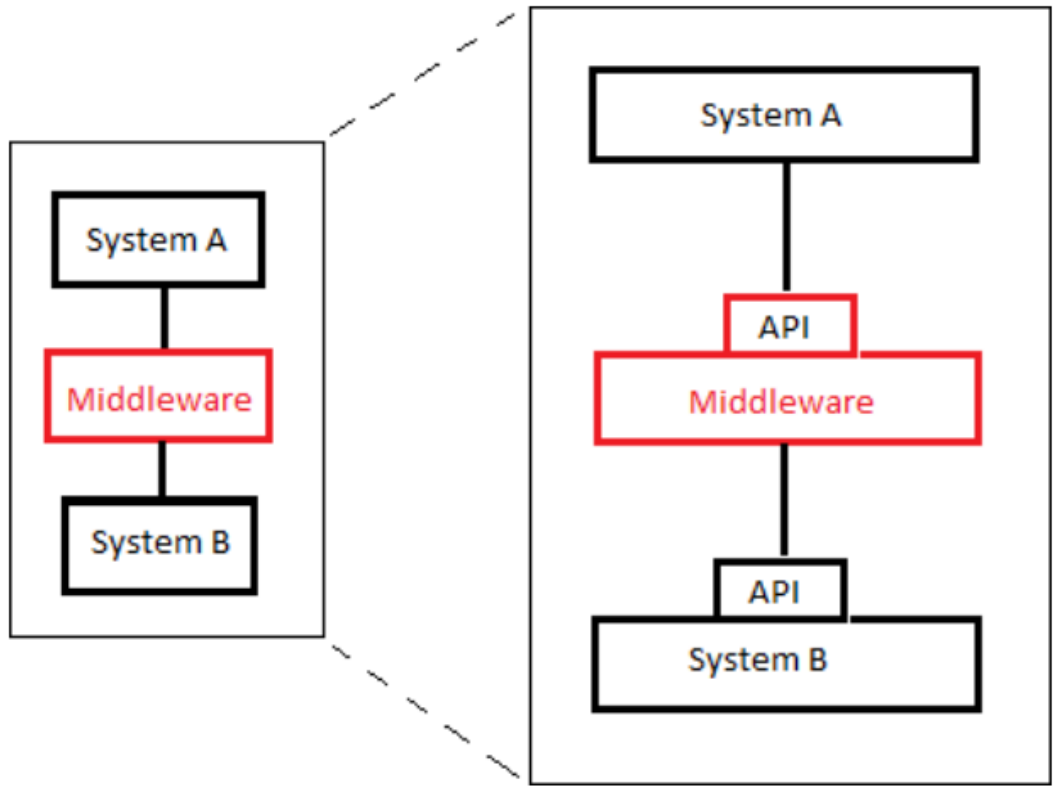


MIDDLEWARE

- Middleware is an application which connects two or more applications, How? through some interface that is called API.
- Middleware is a general term, API comes in technical details - how systems (will) communicate via or without middleware.

You can compare middleware to a translator helping people who speak different languages understand each other.

Middleware has also been referred to as "software glue", since it helps "glue" together different software so they can work together.

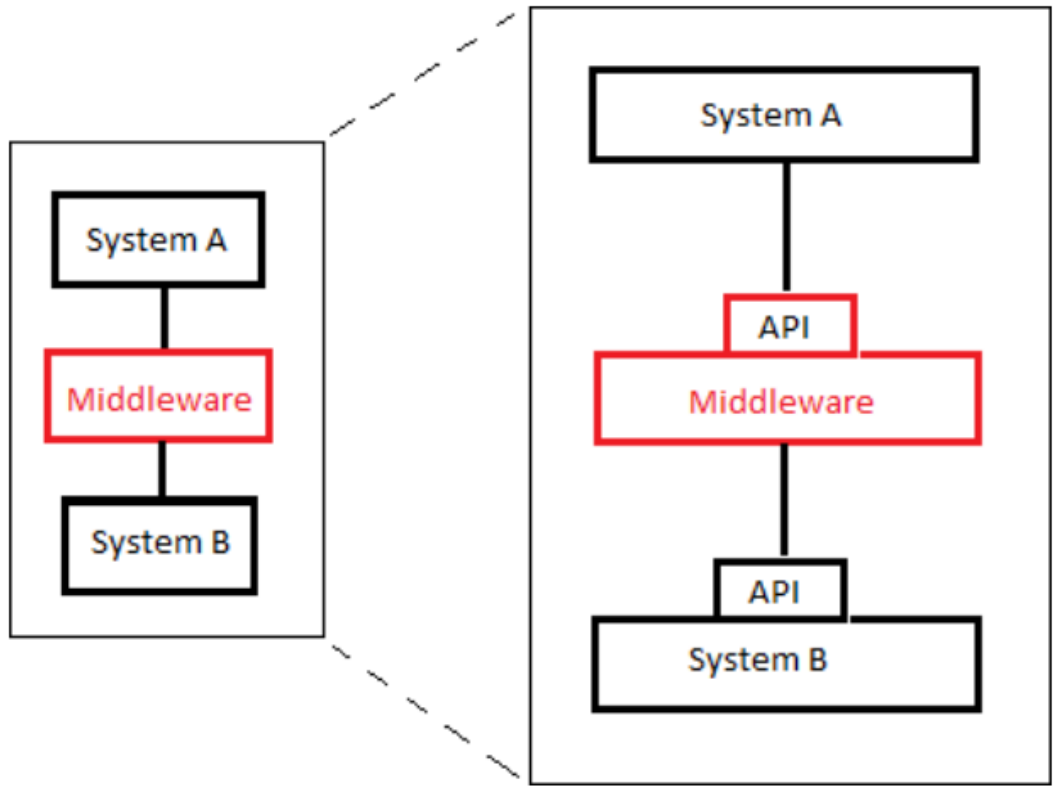


MIDDLEWARE

- Middleware is an application which connects two or more applications, How? through some interface that is called API.
- Middleware is a general term, API comes in technical details - how systems (will) communicate via or without middleware.

You can compare middleware to a translator helping people who speak different languages understand each other.

Middleware has also been referred to as "software glue", since it helps "glue" together different software so they can work together.



FUNCTIONS OF A MIDDLEWARE

- A middleware has mainly four functions :
  1. Make distribution as invisible (transparent) as possible
  2. Provide a homogeneous view of underlying heterogeneous hardware and software systems
  3. Provide services of common use for distributed systems
  4. Provide a high-level interface or API (Applications Programming Interface) for programming distributed applications

FUNCTIONS OF A MIDDLEWARE

- A middleware has mainly four functions :
  1. Make distribution as invisible (transparent) as possible
  2. Provide a homogeneous view of underlying heterogeneous hardware and software systems
  3. Provide services of common use for distributed systems
  4. Provide a high-level interface or API (Applications Programming Interface) for programming distributed applications

MIDDLEWARE FOR DISTRIBUTED SYSTEMS

- Middleware aims at simplifying programming distributed Systems
  - Implementation, evolution and reuse of applications code
  - Inter-platform portability of applications
  - Interoperability between heterogeneous applications

MIDDLEWARE FOR DISTRIBUTED SYSTEMS

- Middleware aims at simplifying programming distributed Systems
  - Implementation, evolution and reuse of applications code
  - Inter-platform portability of applications
  - Interoperability between heterogeneous applications

MIDDLEWARE FOR DISTRIBUTED SYSTEMS

- Middleware aims at simplifying programming distributed Systems
  - Implementation, evolution and reuse of applications code
  - Inter-platform portability of applications
  - Interoperability between heterogeneous applications

MIDDLEWARE FOR DISTRIBUTED SYSTEMS

- Middleware aims at simplifying programming distributed Systems
  - Implementation, evolution and reuse of applications code
  - Inter-platform portability of applications
  - Interoperability between heterogeneous applications

MIDDLEWARE FOR DISTRIBUTED SYSTEMS

- Middleware aims at simplifying programming distributed Systems
  - Implementation, evolution and reuse of applications code
  - Inter-platform portability of applications
  - Interoperability between heterogeneous applications

TYPES

- Remote Procedure Call (RPC)
- Object-Oriented Middleware (OOM)
  - Java RMI
  - CORBA
- Message-Oriented Middleware (MOM)
  - Java Message Service
  - IBM MQSeries
  - Web Services

TYPES

- Remote Procedure Call (RPC)
- Object-Oriented Middleware (OOM)
  - Java RMI
  - CORBA
- Message-Oriented Middleware (MOM)
  - Java Message Service
  - IBM MQSeries
  - Web Services

TYPES

- Remote Procedure Call (RPC)
- Object-Oriented Middleware (OOM)
  - Java RMI
  - CORBA
- Message-Oriented Middleware (MOM)
  - Java Message Service
  - IBM MQSeries
  - Web Services

TYPES

- Remote Procedure Call (RPC)
- Object-Oriented Middleware (OOM)
  - Java RMI
  - CORBA
- Message-Oriented Middleware (MOM)
  - Java Message Service
  - IBM MQSeries
  - Web Services

WHAT TO USE?

- Sockets API ≡ send & recv calls ≡ I/O
- Remote Procedure Calls (RPC)
  - Goal: to provide a procedural interface for distributed (i.e., remote) services
  - To make distributed nature of service transparent to the programmer
- Remote Method Invocation (RMI)
  - RPC + Object Orientation
  - Allows objects living in one process to invoke methods of an object living in another process

WHAT TO USE?

- Sockets API ≡ send & recv calls ≡ I/O
- Remote Procedure Calls (RPC)
  - Goal: to provide a procedural interface for distributed (i.e., remote) services
  - To make distributed nature of service transparent to the programmer
- Remote Method Invocation (RMI)
  - RPC + Object Orientation
  - Allows objects living in one process to invoke methods of an object living in another process

WHAT TO USE?

- Sockets API ≡ send & recv calls ≡ I/O
- Remote Procedure Calls (RPC)
  - Goal: to provide a procedural interface for distributed (i.e., remote) services
  - To make distributed nature of service transparent to the programmer
- Remote Method Invocation (RMI)
  - RPC + Object Orientation
  - Allows objects living in one process to invoke methods of an object living in another process

WHAT TO USE?

- Sockets API ≡ send & recv calls ≡ I/O
- Remote Procedure Calls (RPC)
  - Goal: to provide a procedural interface for distributed (i.e., remote) services
  - To make distributed nature of service transparent to the programmer
- Remote Method Invocation (RMI)
  - RPC + Object Orientation
  - Allows objects living in one process to invoke methods of an object living in another process

WHAT TO USE?

- Sockets API ≡ send & recv calls ≡ I/O
- Remote Procedure Calls (RPC)
  - Goal: to provide a procedural interface for distributed (i.e., remote) services
  - To make distributed nature of service transparent to the programmer
- Remote Method Invocation (RMI)
  - RPC + Object Orientation
  - Allows objects living in one process to invoke methods of an object living in another process

WHAT TO USE?

- Sockets API ≡ send & recv calls ≡ I/O
- Remote Procedure Calls (RPC)
  - Goal: to provide a procedural interface for distributed (i.e., remote) services
  - To make distributed nature of service transparent to the programmer
- Remote Method Invocation (RMI)
  - RPC + Object Orientation
  - Allows objects living in one process to invoke methods of an object living in another process

REMOTE PROCEDURE CALL (RPC)

- RPC is a technology developed in the 80s to call procedures on remotes computers (with the procedural programming paradigm)
- RPC allows the call of procedures located in another process space on a remote computer (or on the same machine)
- Technical issues for implementing RPC
  - Different address spaces
  - Heterogeneous machines

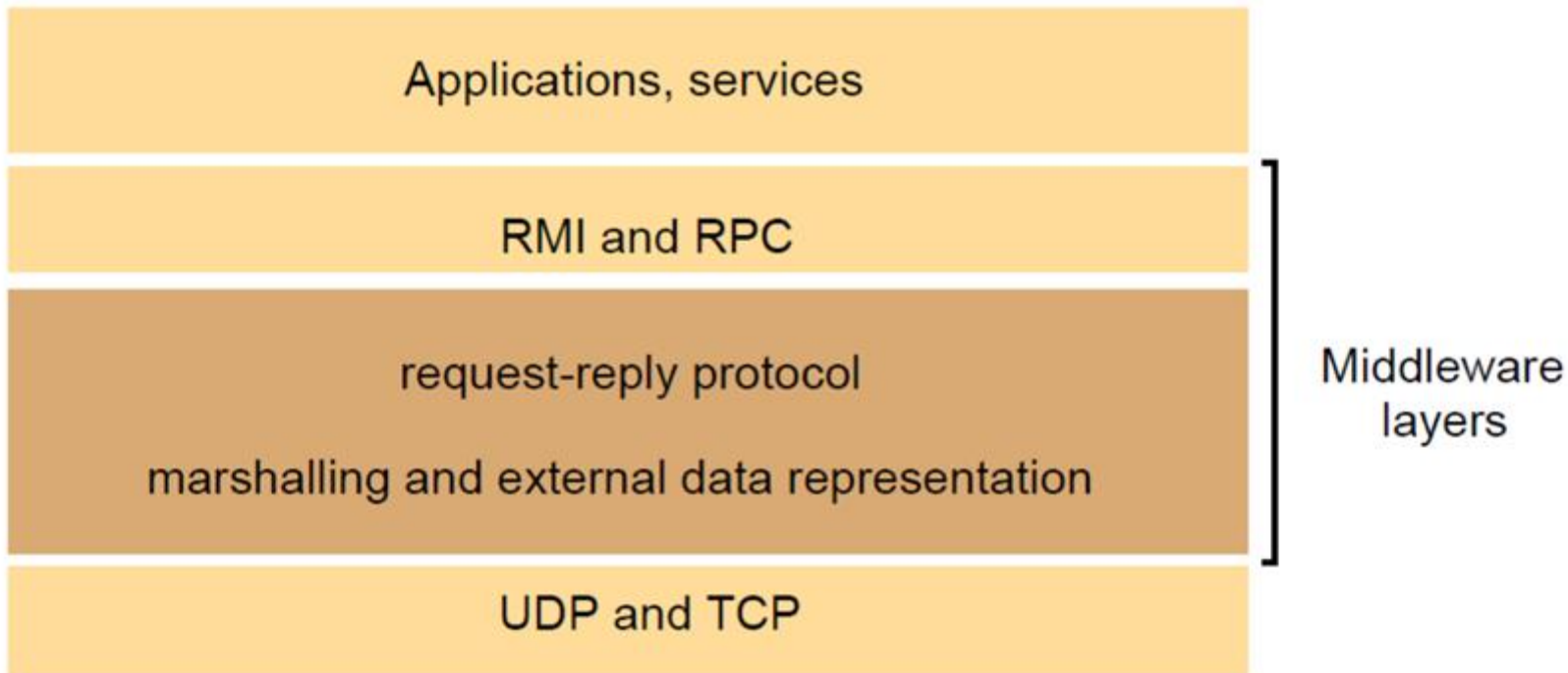
REMOTE PROCEDURE CALL (RPC)

- RPC is a technology developed in the 80s to call procedures on remotes computers (with the procedural programming paradigm)
- RPC allows the call of procedures located in another process space on a remote computer (or on the same machine)
- Technical issues for implementing RPC
  - Different address spaces
  - Heterogeneous machines

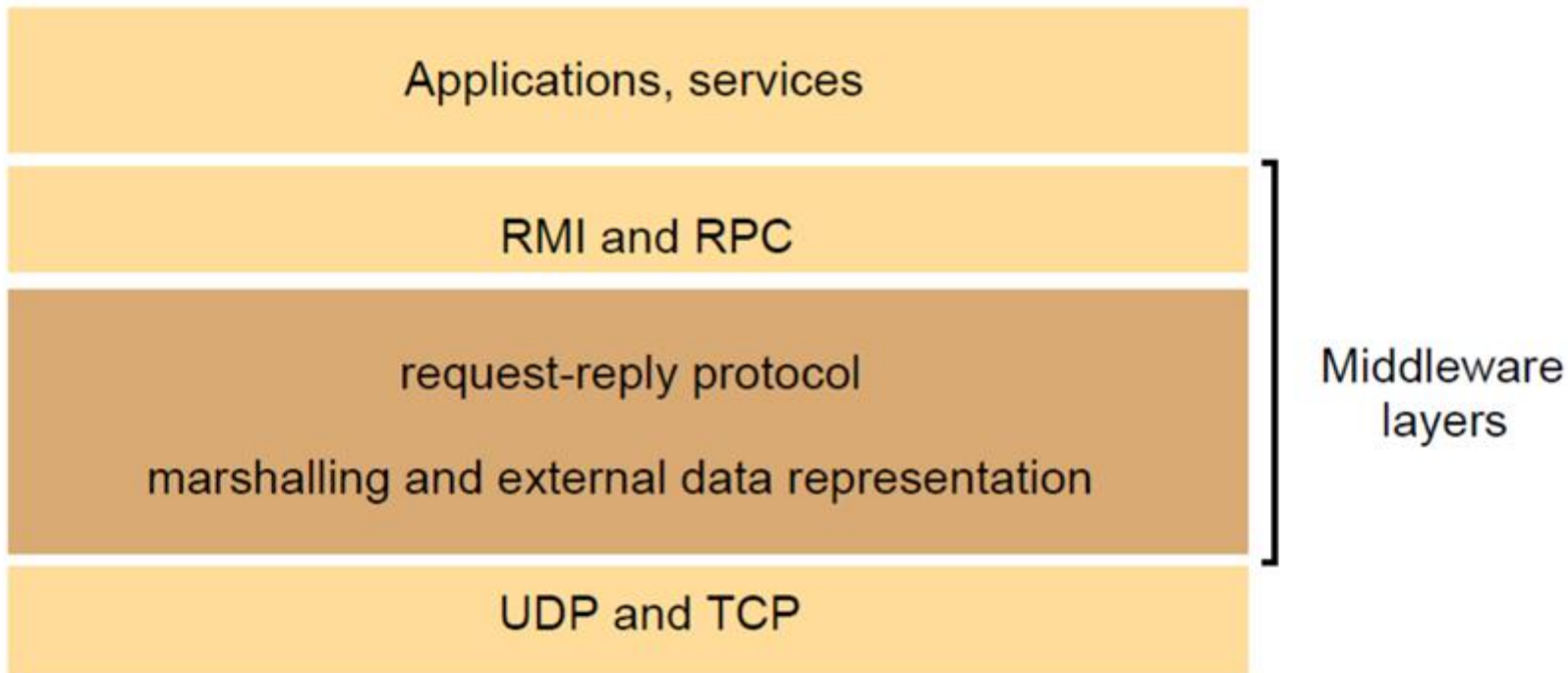
REMOTE PROCEDURE CALL (RPC)

- RPC is a technology developed in the 80s to call procedures on remotes computers (with the procedural programming paradigm)
- RPC allows the call of procedures located in another process space on a remote computer (or on the same machine)
- Technical issues for implementing RPC
  - Different address spaces
  - Heterogeneous machines

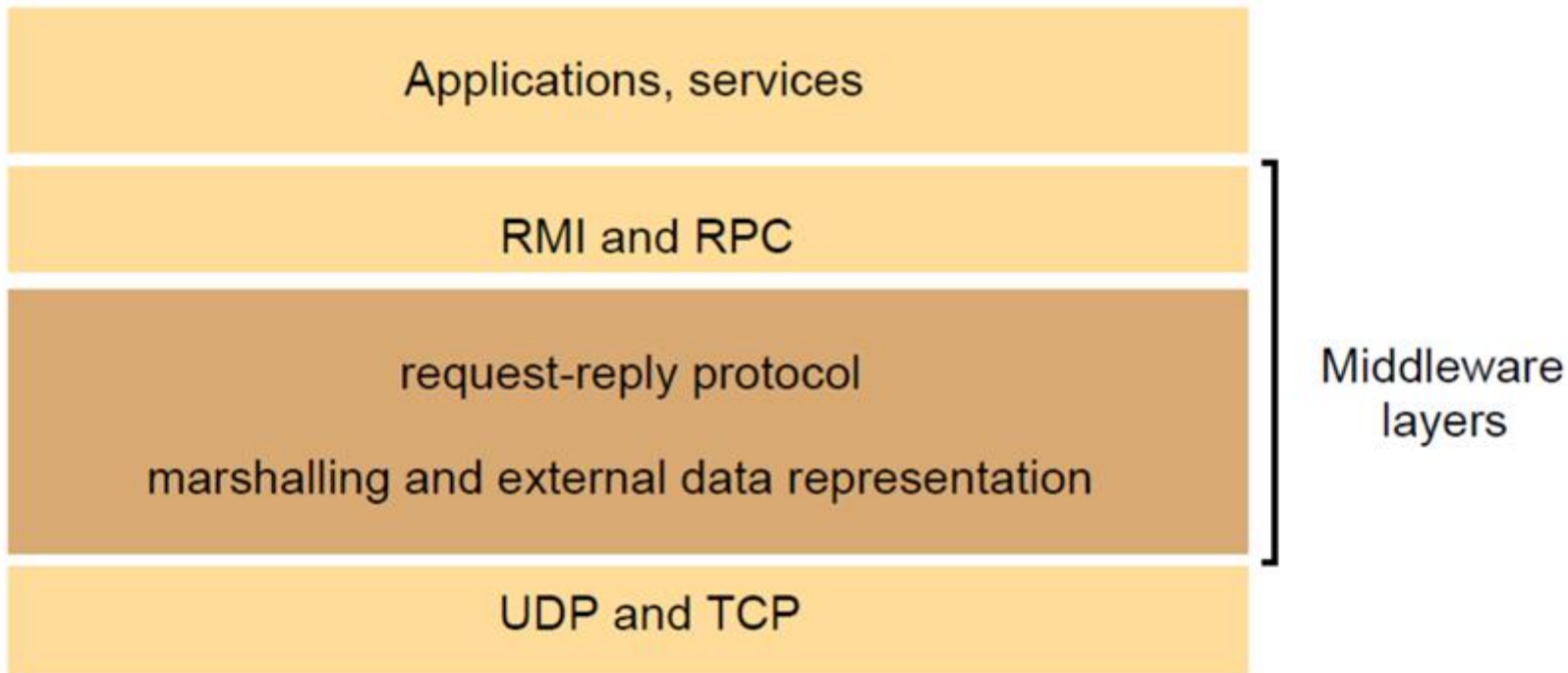
Middleware layers



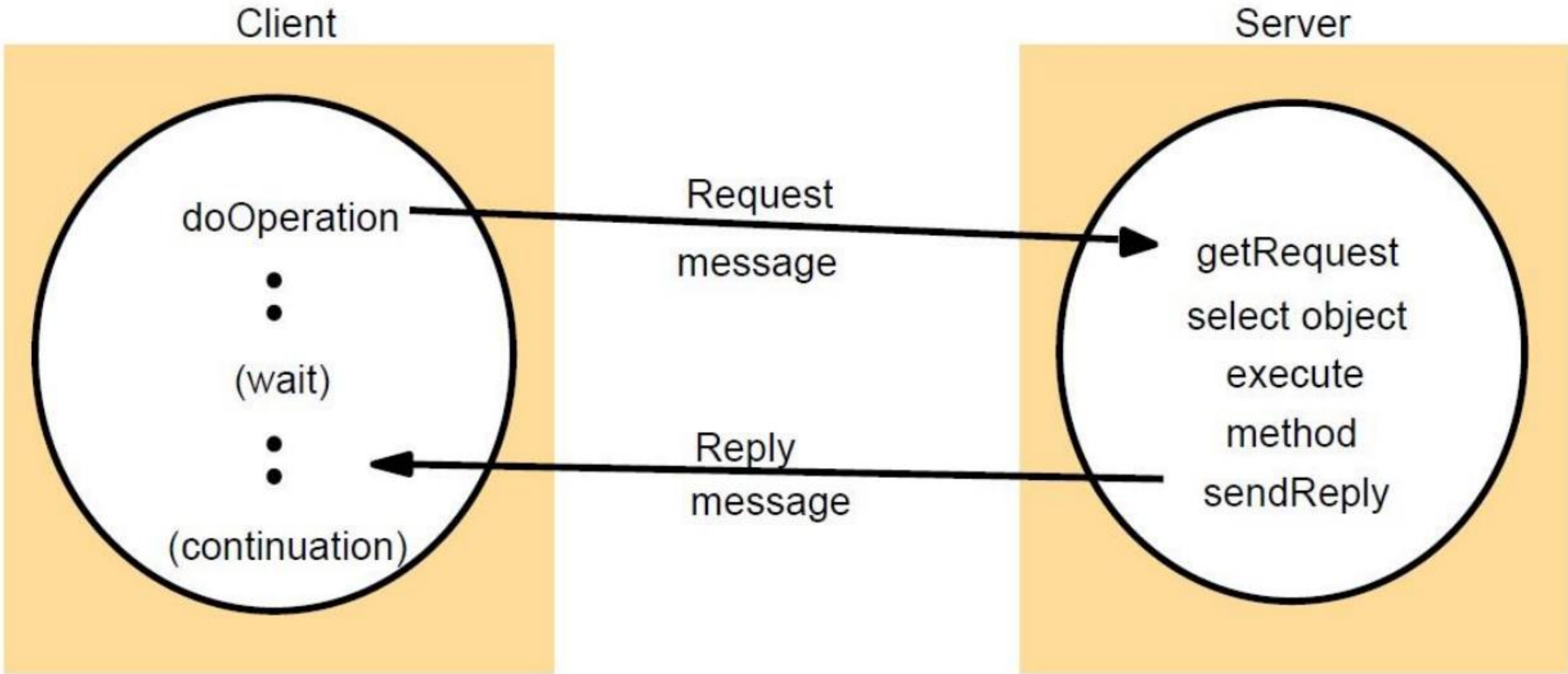
Middleware layers



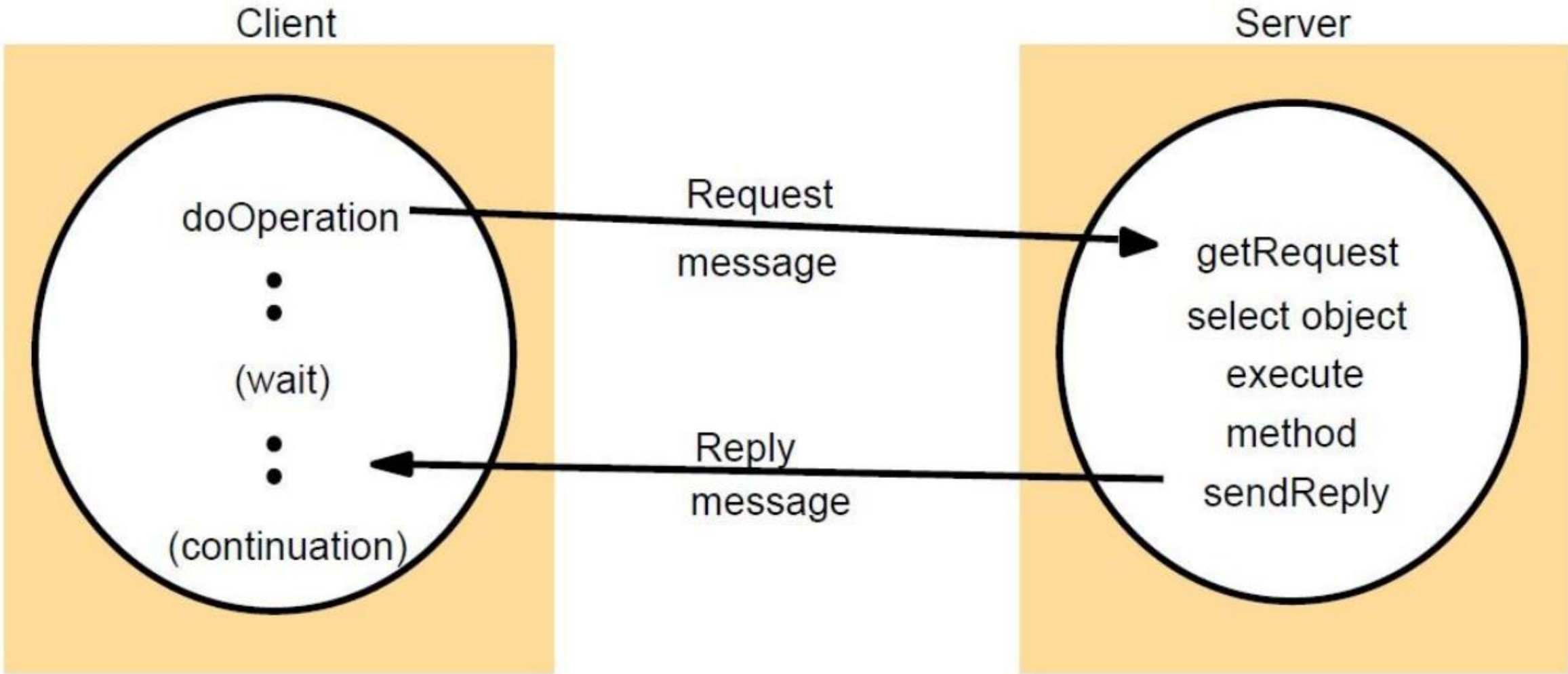
Middleware layers



Request-reply communication



Request-reply communication

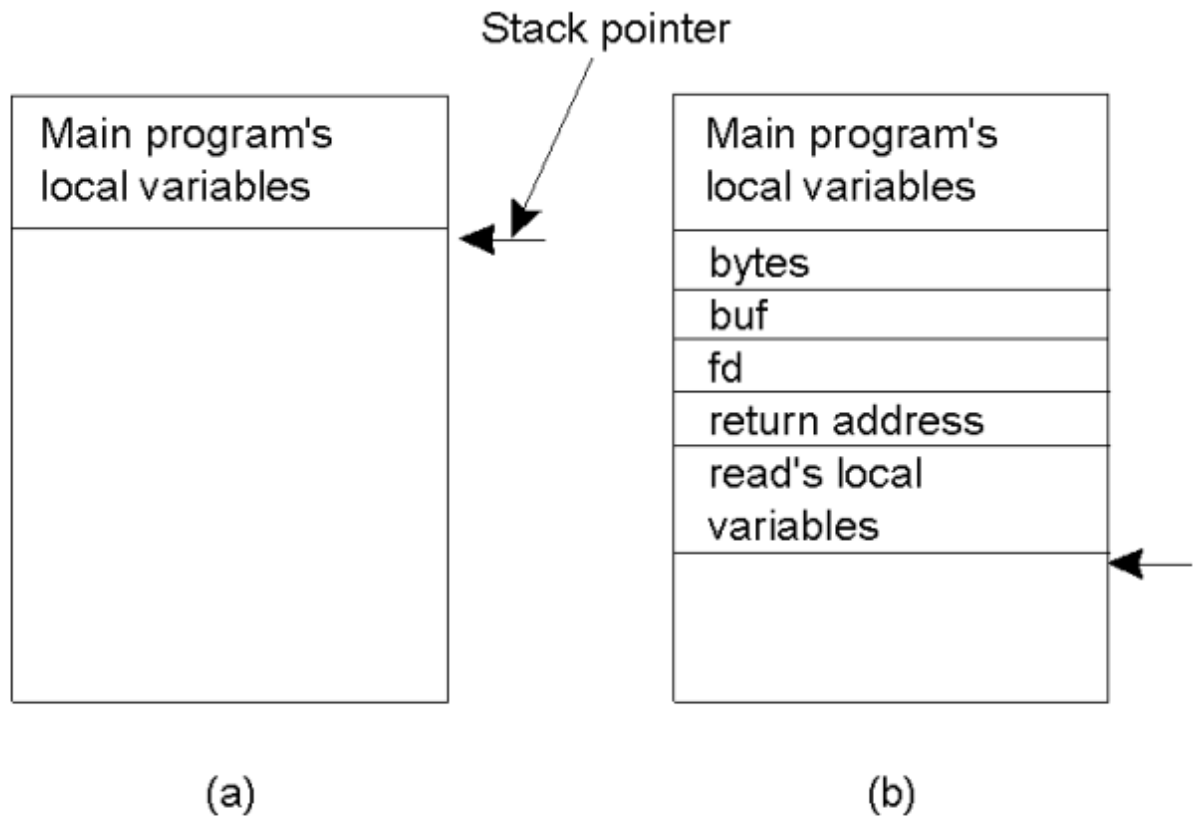


CONVENTIONAL PROCEDURE CALL

a) Parameter passing in a local procedure call: the stack before the call to read

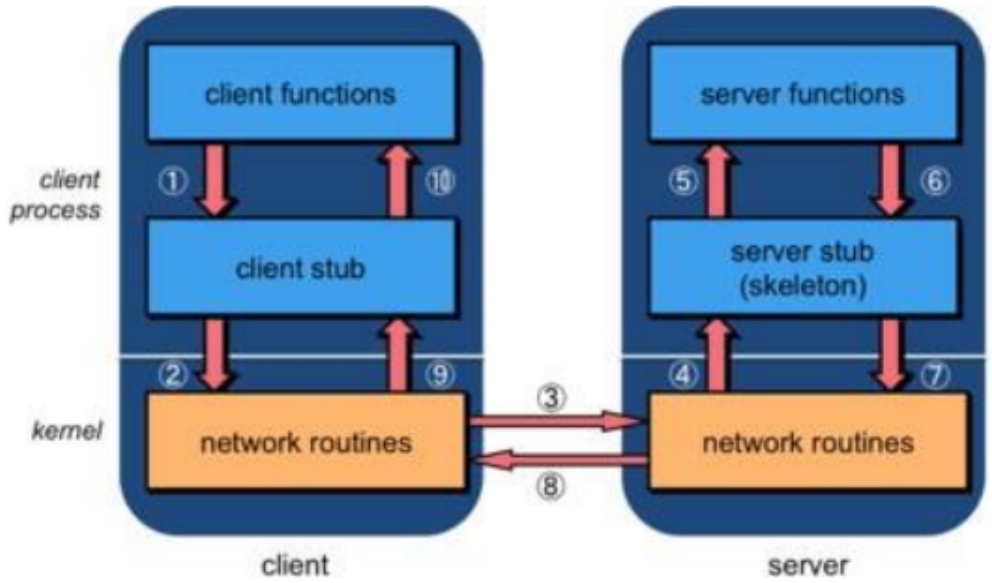
```
Count = read (fd, buf, nbytes);
```

b) The stack while the called procedure is active



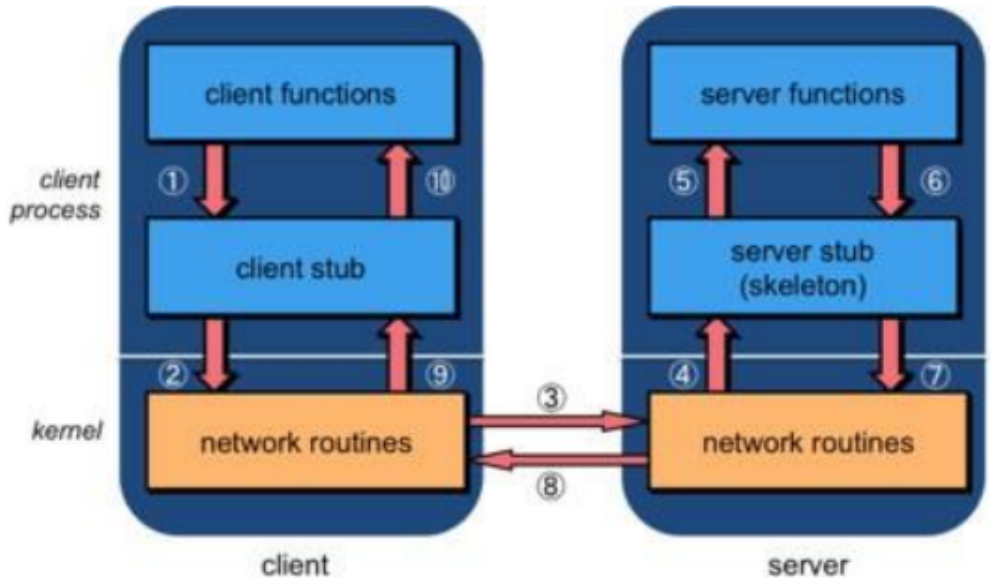
- The sequence of operations that takes place in making a remote procedure call

1. The client calls a local procedure, called the client stub. To the client process, it appears that this is the actual remote procedure, which it can call using a local procedure call (since the stub is a local procedure). The client stub packages the arguments to the remote procedure (this may involve converting them to a standard format) and builds one or more network messages. The packaging of arguments into a network message is called marshaling.
2. Network messages are sent by the client stub to the remote system (by writing to a socket via system calls to the local kernel).
3. Network messages are transferred by the kernel to the remote system via some protocol (either connectionless or connection-oriented).
4. A server stub process, sometimes called a skeleton, on the server receives the messages. It unmarshals the arguments from the messages and, if needed, converts them from a standard format into a machine-specific format.



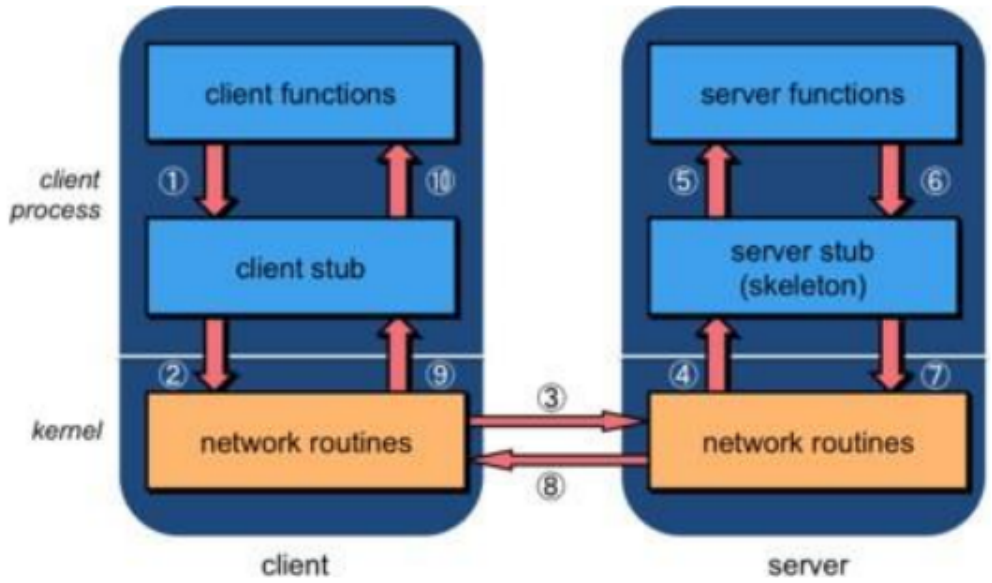
- The sequence of operations that takes place in making a remote procedure call

1. The client calls a local procedure, called the client stub. To the client process, it appears that this is the actual remote procedure, which it can call using a local procedure call (since the stub is a local procedure). The client stub packages the arguments to the remote procedure (this may involve converting them to a standard format) and builds one or more network messages. The packaging of arguments into a network message is called marshaling.
2. Network messages are sent by the client stub to the remote system (by writing to a socket via system calls to the local kernel).
3. Network messages are transferred by the kernel to the remote system via some protocol (either connectionless or connection-oriented).
4. A server stub process, sometimes called a skeleton, on the server receives the messages. It unmarshals the arguments from the messages and, if needed, converts them from a standard format into a machine-specific format.



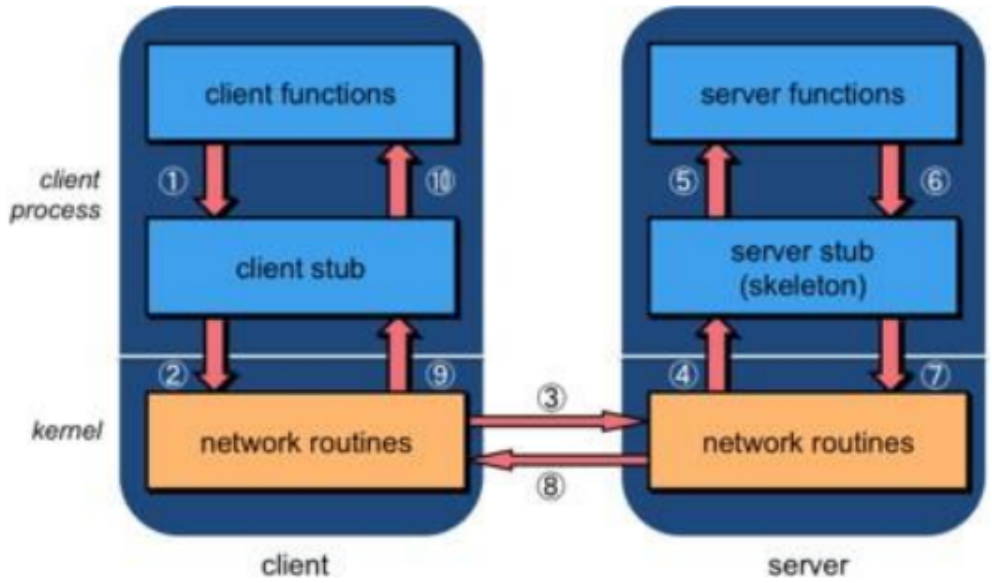
- The sequence of operations that takes place in making a remote procedure call

1. The client calls a local procedure, called the client stub. To the client process, it appears that this is the actual remote procedure, which it can call using a local procedure call (since the stub is a local procedure). The client stub packages the arguments to the remote procedure (this may involve converting them to a standard format) and builds one or more network messages. The packaging of arguments into a network message is called marshaling.
2. Network messages are sent by the client stub to the remote system (by writing to a socket via system calls to the local kernel).
3. Network messages are transferred by the kernel to the remote system via some protocol (either connectionless or connection-oriented).
4. A server stub process, sometimes called a skeleton, on the server receives the messages. It unmarshals the arguments from the messages and, if needed, converts them from a standard format into a machine-specific format.



- The sequence of operations that takes place in making a remote procedure call

1. The client calls a local procedure, called the client stub. To the client process, it appears that this is the actual remote procedure, which it can call using a local procedure call (since the stub is a local procedure). The client stub packages the arguments to the remote procedure (this may involve converting them to a standard format) and builds one or more network messages. The packaging of arguments into a network message is called marshaling.
2. Network messages are sent by the client stub to the remote system (by writing to a socket via system calls to the local kernel).
3. Network messages are transferred by the kernel to the remote system via some protocol (either connectionless or connection-oriented).
4. A server stub process, sometimes called a skeleton, on the server receives the messages. It unmarshals the arguments from the messages and, if needed, converts them from a standard format into a machine-specific format.



5. The server stub calls a local procedure call to the actual server function, passing it the arguments that it received from the client. The server function gets the illustion that it was called locally by the client since the server stub calls it locally with the same parameters that the client transmitted.
6. When the server is finished, it returns to the server stub with its return values.
7. The server stub converts the return values to a standard format (if necessary) and marshals them into one or more network messages to send to the client stub.
8. Messages get sent back across the network to the client stub.
9. The client stub reads the messages from the local kernel.
10. It then returns the results to the client function (possibly converting them first). The client feels that it just received a return value from the remote function, unaware that all the network messaging took place.