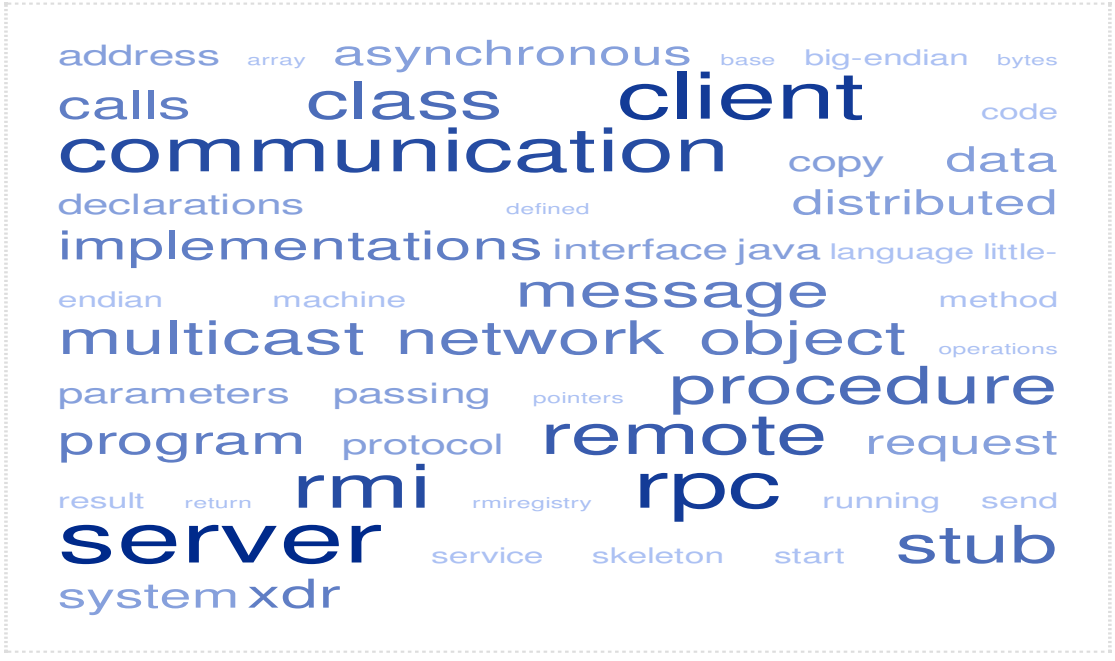
Communication



Overview

- Communication types and role of Middleware
- Remote Procedure Call (RPC)
- Message Oriented Communication
- Multicasting

Communication Types

- Transient communication. A message is stored by the communication middleware as long as the sending and receiving applications are executing.

Communication Types

- Transient communication. A message is stored by the communication middleware as long as the sending and receiving applications are executing.
- Persistent communication. A message that has been submitted for transmission is stored by the communication middleware for as long as it takes to deliver it to the receiver. E.g. Email.

Communication Types

- Transient communication. A message is stored by the communication middleware as long as the sending and receiving applications are executing.
- Persistent communication. A message that has been submitted for transmission is stored by the communication middleware for as long as it takes to deliver it to the receiver. E.g. Email.
- Synchronous communication. The sender is blocked until its request is known to be accepted. This can happen at three places.

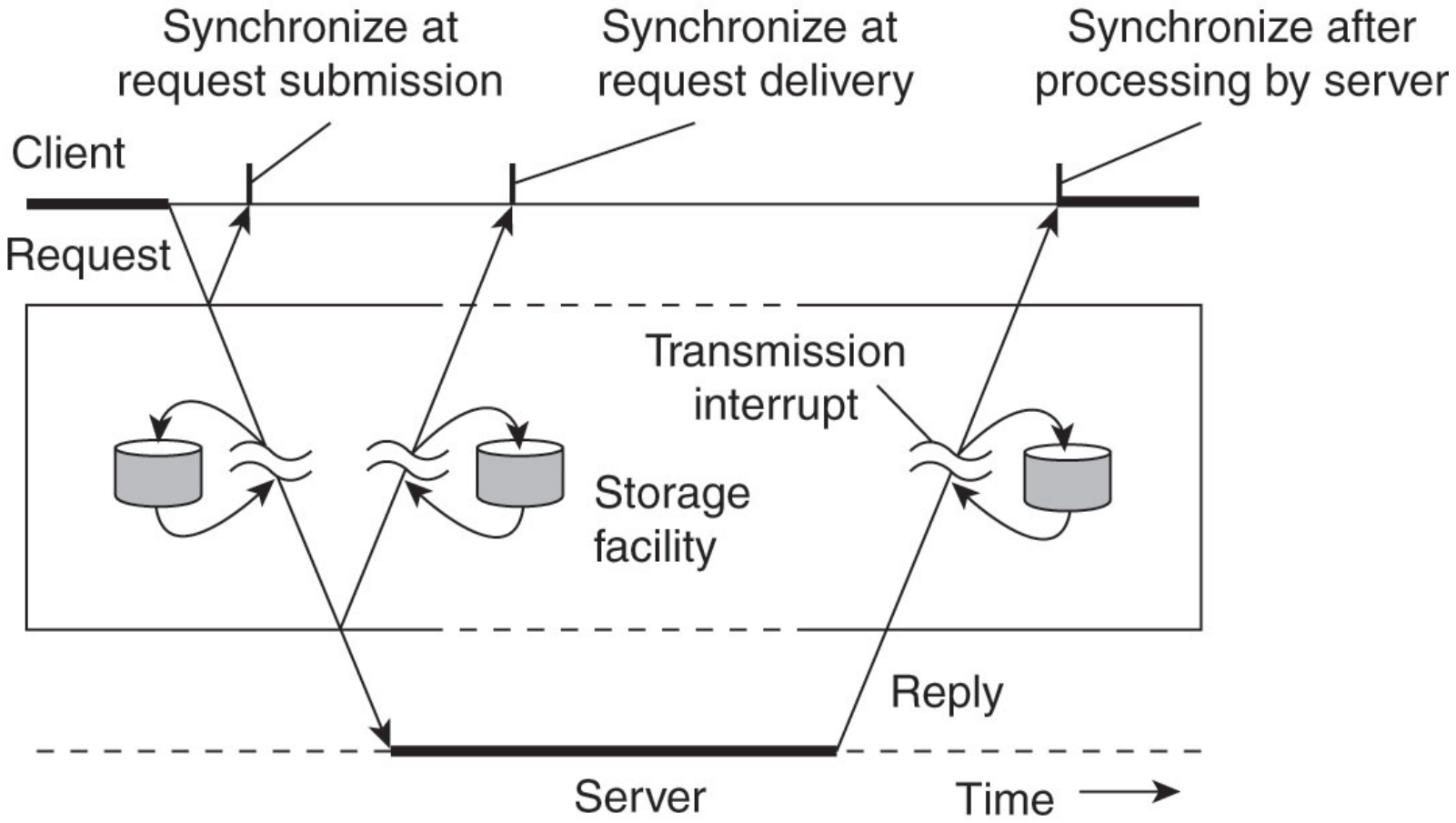
Communication Types

- Transient communication. A message is stored by the communication middleware as long as the sending and receiving applications are executing.
- Persistent communication. A message that has been submitted for transmission is stored by the communication middleware for as long as it takes to deliver it to the receiver. E.g. Email.
- Synchronous communication. The sender is blocked until its request is known to be accepted. This can happen at three places.
  - The sender may be blocked until the middleware notifies that it will take over the transmission of the message.
  - The sender may be blocked until the request has been delivered to the intended recipient.
  - The sender waits until its request has been fully processed, that is, up to the time that the recipient returns a response.

Communication Types

- Transient communication. A message is stored by the communication middleware as long as the sending and receiving applications are executing.
- Persistent communication. A message that has been submitted for transmission is stored by the communication middleware for as long as it takes to deliver it to the receiver. E.g. Email.
- Synchronous communication. The sender is blocked until its request is known to be accepted. This can happen at three places.
  - The sender may be blocked until the middleware notifies that it will take over the transmission of the message.
  - The sender may be blocked until the request has been delivered to the intended recipient.
  - The sender waits until its request has been fully processed, that is, up to the time that the recipient returns a response.
- Asynchronous communication. The sender continues immediately after it has submitted its message for transmission. This means that the message is (temporarily) stored stored immediately by the middleware upon transmission.

Role of Middleware



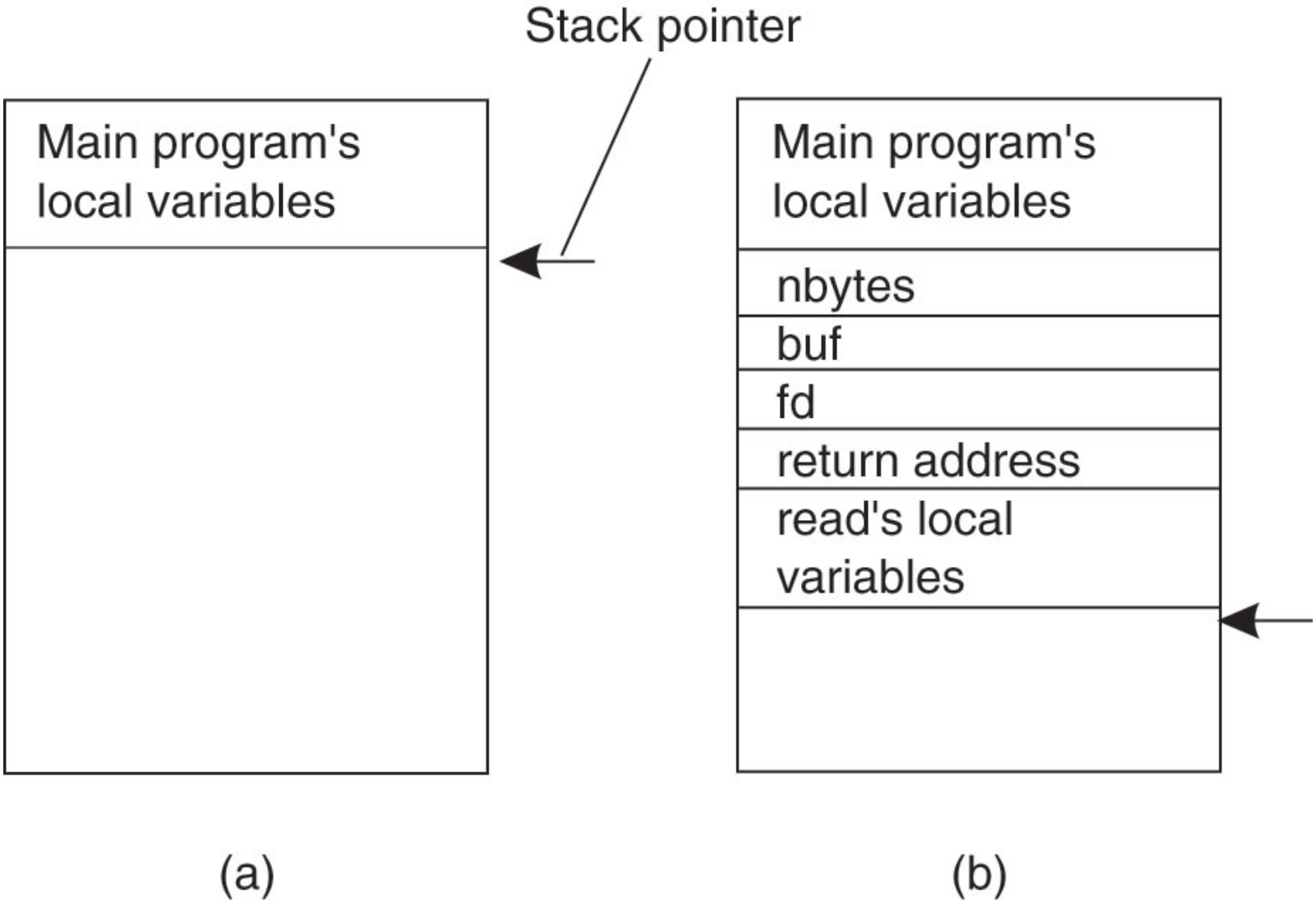
Remote Procedure Call (RPC)

- A remote procedure call (RPC) allows a program to transparently call procedures located on another machine. No message passing is visible to the programmer.

Remote Procedure Call (RPC)

- A remote procedure call (RPC) allows a program to transparently call procedures located on another machine. No message passing is visible to the programmer.
- A widely used technique that underlies many distributed systems.

Conventional Procedure Call




```
count = read(fd, buf nbytes); //in main
```

Parameter Passing

- Call-by-value: The variable is copied to the stack. The original value is left unchanged

Parameter Passing

- Call-by-value: The variable is copied to the stack. The original value is left unchanged
- Call-by-reference: The address of the variable is copied to the stack so the called procedure modifies the same copy as the procedure calling

Parameter Passing

- Call-by-value: The variable is copied to the stack. The original value is left unchanged
- Call-by-reference: The address of the variable is copied to the stack so the called procedure modifies the same copy as the procedure calling
- Call-by-copy/restore: The variable is copied to the stack by the caller and then copied back after the call, overwriting the caller's original value

Parameter Passing

- Call-by-value: The variable is copied to the stack. The original value is left unchanged
- Call-by-reference: The address of the variable is copied to the stack so the called procedure modifies the same copy as the procedure calling
- Call-by-copy/restore: The variable is copied to the stack by the caller and then copied back after the call, overwriting the caller's original value
- **In-class Exercise**. How is call-by-copy/restore different from call-by-reference?

Basic Principles of RPC

- Transparency is achieved by using client stub and server stub.

Basic Principles of RPC

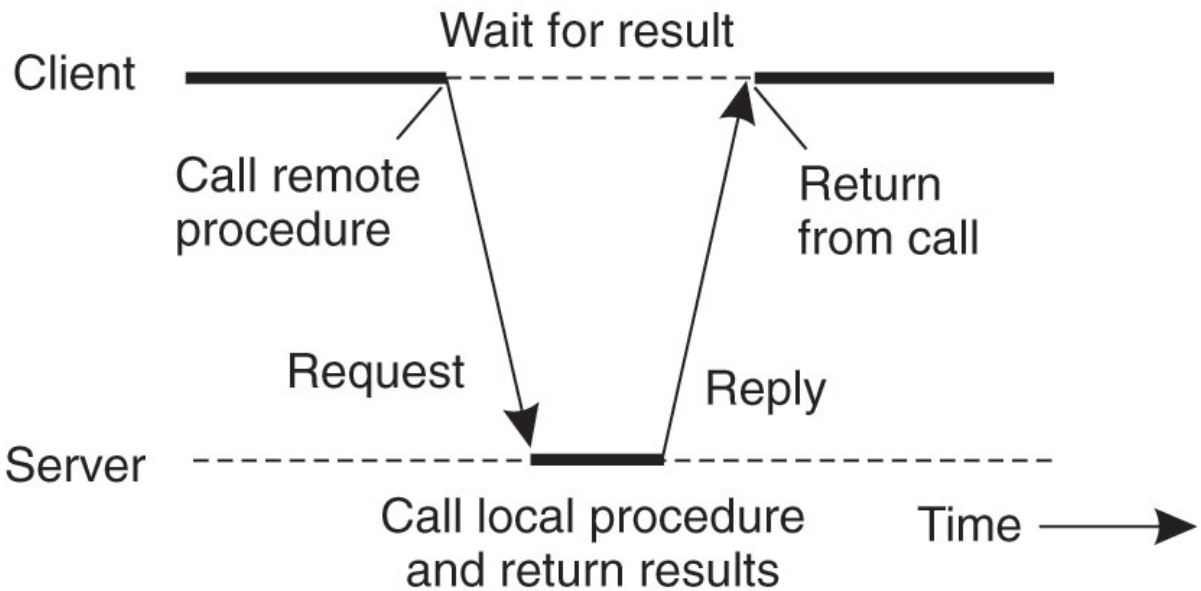
- Transparency is achieved by using client stub and server stub.
- A client stub is a local library procedure that translates the call into a request to the server and sets up the results as a return to the call from the client.

Basic Principles of RPC

- Transparency is achieved by using client stub and server stub.
- A client stub is a local library procedure that translates the call into a request to the server and sets up the results as a return to the call from the client.
- A server stub is a procedure (local to the server) that transforms requests coming over the network into local procedure calls.

Basic Principles of RPC

- Transparency is achieved by using client stub and server stub.
- A client stub is a local library procedure that translates the call into a request to the server and sets up the results as a return to the call from the client.
- A server stub is a procedure (local to the server) that transforms requests coming over the network into local procedure calls.



Remote Procedure Call: Step by Step (1)

- The client procedure calls the client stub in the normal way.

Remote Procedure Call: Step by Step (1)

- The client procedure calls the client stub in the normal way.
- The client stub builds a message and calls the local operating system.

Remote Procedure Call: Step by Step (1)

- The client procedure calls the client stub in the normal way.
- The client stub builds a message and calls the local operating system.
- The client's OS sends the message to the remote OS.

Remote Procedure Call: Step by Step (1)

- The client procedure calls the client stub in the normal way.
- The client stub builds a message and calls the local operating system.
- The client's OS sends the message to the remote OS.
- The remote OS gives the message to the server stub.

Remote Procedure Call: Step by Step (1)

- The client procedure calls the client stub in the normal way.
- The client stub builds a message and calls the local operating system.
- The client's OS sends the message to the remote OS.
- The remote OS gives the message to the server stub.
- The server stub unpacks the parameters and calls the server.

Remote Procedure Call: Step by Step (2)

- The server does the work and returns the result to the server stub.

Remote Procedure Call: Step by Step (2)

- The server does the work and returns the result to the server stub.
- The server stub packs it in a message and calls it's local OS.

Remote Procedure Call: Step by Step (2)

- The server does the work and returns the result to the server stub.
- The server stub packs it in a message and calls it's local OS.
- The server's OS sends the message to the client's OS.

Remote Procedure Call: Step by Step (2)

- The server does the work and returns the result to the server stub.
- The server stub packs it in a message and calls it's local OS.
- The server's OS sends the message to the client's OS.
- The client's OS gives the message to the client stub.

Remote Procedure Call: Step by Step (2)

- The server does the work and returns the result to the server stub.
- The server stub packs it in a message and calls it's local OS.
- The server's OS sends the message to the client's OS.
- The client's OS gives the message to the client stub.
- The stub unpacks the result and returns to the client.

Parameter Passing in RPC

Passing parameters to RPCs can be tricky since it requires packing parameters into a message (parameter marshaling) to be interpreted correctly on another system (unmarshaling). Here are some issues:

Parameter Passing in RPC

Passing parameters to RPCs can be tricky since it requires packing parameters into a message (parameter marshaling) to be interpreted correctly on another system (unmarshaling). Here are some issues:

- Machines may use different character codes.

Parameter Passing in RPC

Passing parameters to RPCs can be tricky since it requires packing parameters into a message (parameter marshaling) to be interpreted correctly on another system (unmarshaling). Here are some issues:

- Machines may use different character codes.
- Different representation of integers and floating-point numbers

Parameter Passing in RPC

Passing parameters to RPCs can be tricky since it requires packing parameters into a message (parameter marshaling) to be interpreted correctly on another system (unmarshaling). Here are some issues:

- Machines may use different character codes.
- Different representation of integers and floating-point numbers
- Different byte addressing: little-endian versus big-endian format

Parameter Passing in RPC

Passing parameters to RPCs can be tricky since it requires packing parameters into a message (parameter marshaling) to be interpreted correctly on another system (unmarshaling). Here are some issues:

- Machines may use different character codes.
- Different representation of integers and floating-point numbers
- Different byte addressing: little-endian versus big-endian format
- How to pass reference parameters (like pointers) since an address will be meaningless on another system? Can we use call-by-copy/restore?

Parameter Passing in RPC

Passing parameters to RPCs can be tricky since it requires packing parameters into a message (parameter marshaling) to be interpreted correctly on another system (unmarshaling). Here are some issues:

- Machines may use different character codes.
- Different representation of integers and floating-point numbers
- Different byte addressing: little-endian versus big-endian format
- How to pass reference parameters (like pointers) since an address will be meaningless on another system? Can we use call-by-copy/restore?
- How to pack types that take less space than a word? For example: a `short` or a `char`.

Parameter Passing in RPC

Passing parameters to RPCs can be tricky since it requires packing parameters into a message (parameter marshaling) to be interpreted correctly on another system (unmarshaling). Here are some issues:

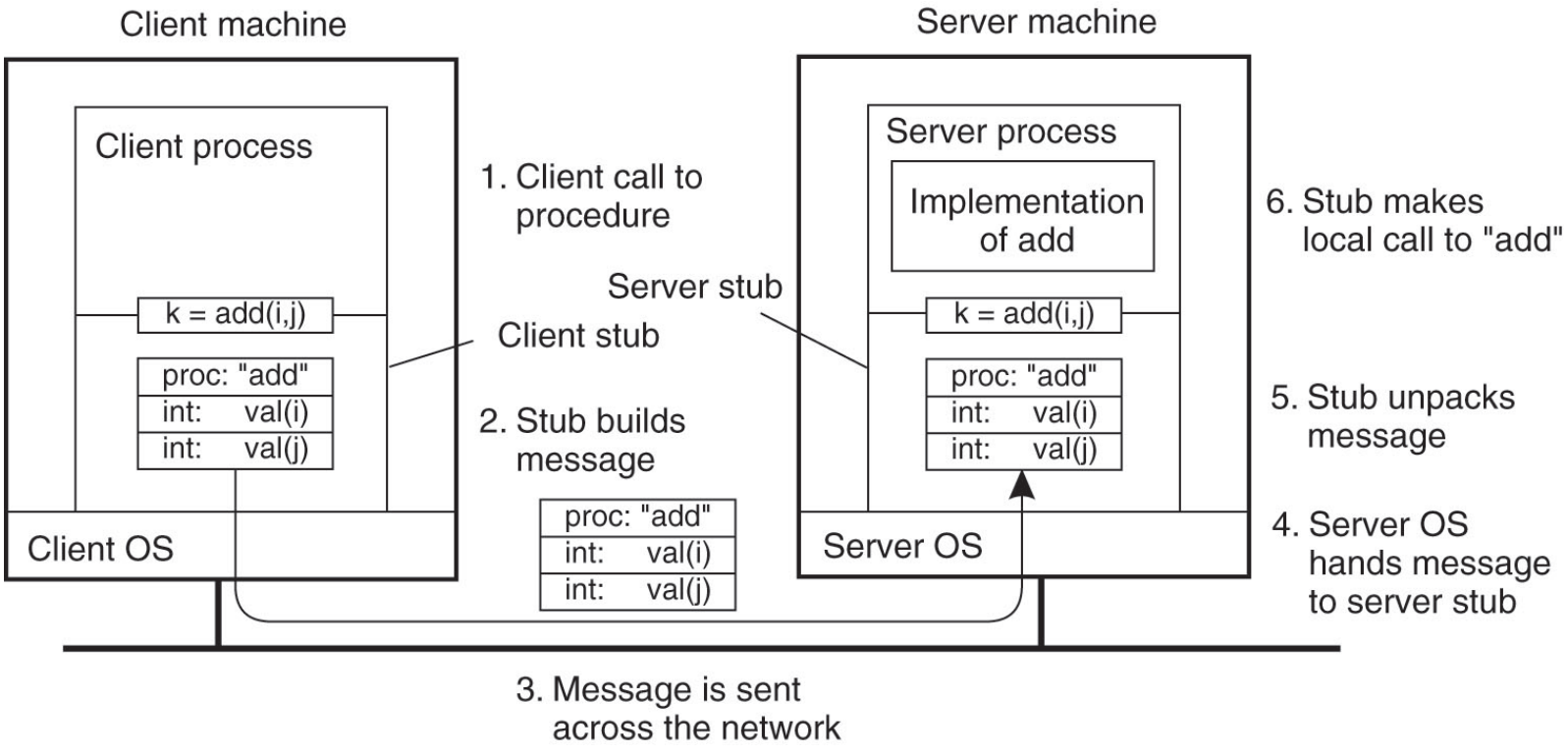
- Machines may use different character codes.
- Different representation of integers and floating-point numbers
- Different byte addressing: little-endian versus big-endian format
- How to pass reference parameters (like pointers) since an address will be meaningless on another system? Can we use call-by-copy/restore?
- How to pack types that take less space than a word? For example: a `short` or a `char`.
- How to pack an array?

Parameter Passing in RPC

Passing parameters to RPCs can be tricky since it requires packing parameters into a message (parameter marshaling) to be interpreted correctly on another system (unmarshaling). Here are some issues:

- Machines may use different character codes.
- Different representation of integers and floating-point numbers
- Different byte addressing: little-endian versus big-endian format
- How to pass reference parameters (like pointers) since an address will be meaningless on another system? Can we use call-by-copy/restore?
- How to pack types that take less space than a word? For example: a `short` or a `char`.
- How to pack an array?
- How to pack an object?

RPC: Passing Parameters



Representation: Little-Endian vs Big-Endian (1)

- Little-endian machine: the bytes in a word are numbered right to left. E.g. Intel processors.

Representation: Little-Endian vs Big-Endian (1)

- Little-endian machine: the bytes in a word are numbered right to left. E.g. Intel processors.
- Big-endian machine: the bytes in a word are numbered left to right. E.g. IBM z/Architecture.

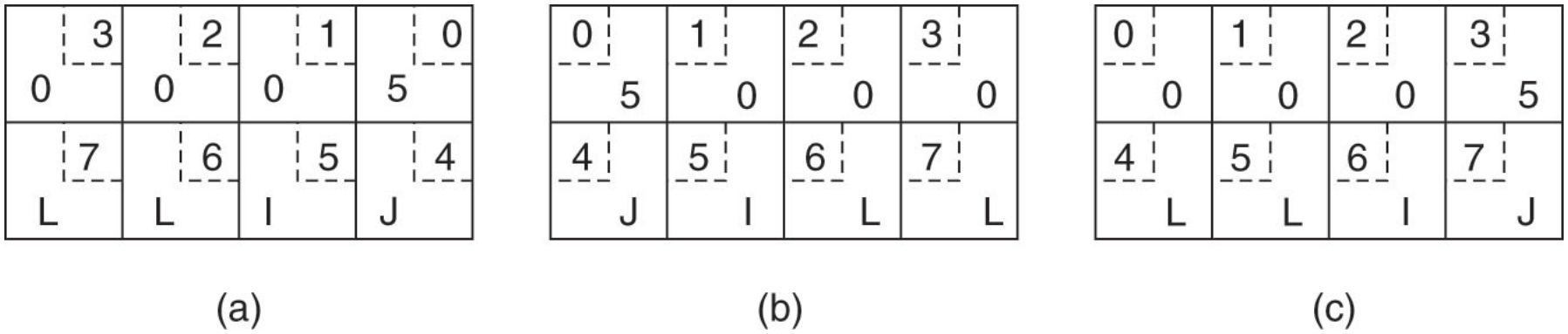
Representation: Little-Endian vs Big-Endian (1)

- Little-endian machine: the bytes in a word are numbered right to left. E.g. Intel processors.
- Big-endian machine: the bytes in a word are numbered left to right. E.g. IBM z/Architecture.
  - Protocols like IPv4, IPv6, TCP and UDP use big-endian so it is also known as the network byte order.

Representation: Little-Endian vs Big-Endian (1)

- Little-endian machine: the bytes in a word are numbered right to left. E.g. Intel processors.
- Big-endian machine: the bytes in a word are numbered left to right. E.g. IBM z/Architecture.
  - Protocols like IPv4, IPv6, TCP and UDP use big-endian so it is also known as the network byte order.
- Bi-endian machine: Choose either setting either via software or hardware. E.g. ARM processors, SPARC, POWER PC.

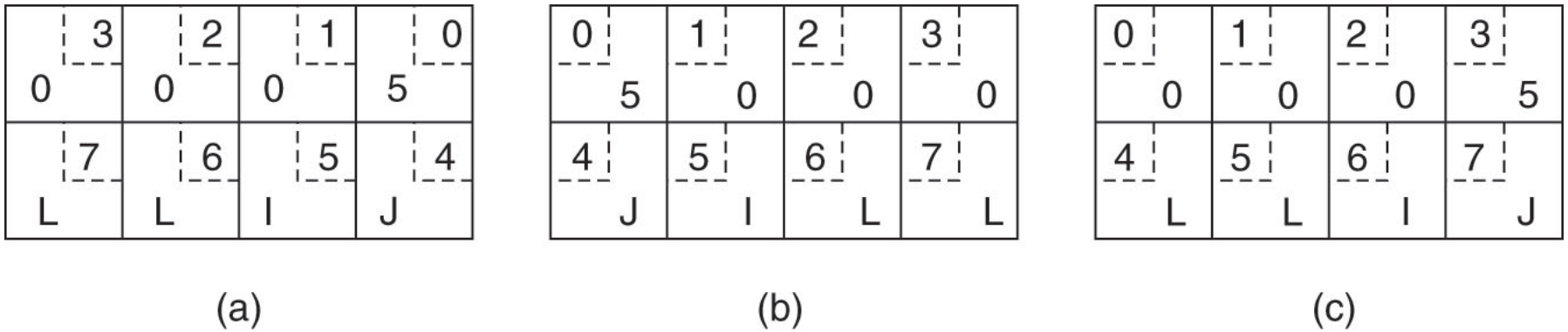
Representation: Little-Endian vs Big-Endian (2)



(a) (b) (c)

a. Little endian (data before sending)

Representation: Little-Endian vs Big-Endian (2)

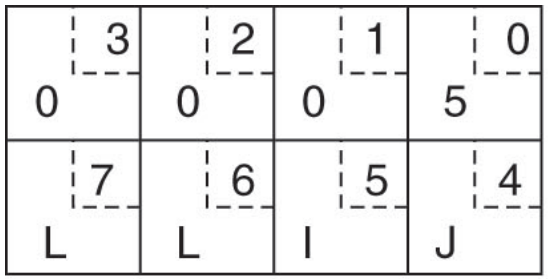


(a) (b) (c)

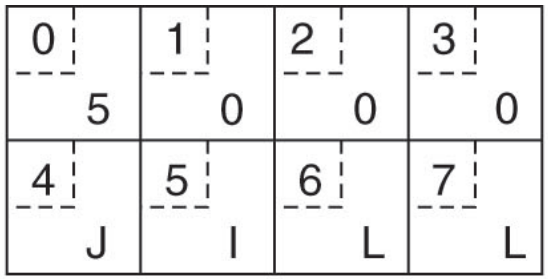
a. Little endian (data before sending)

b. Big endian (data after receiving)

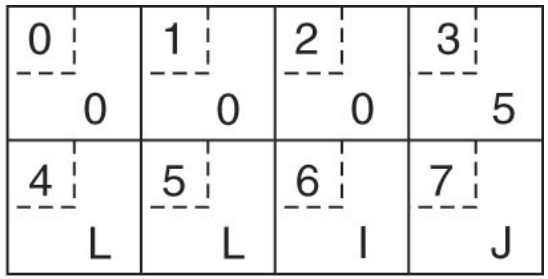
Representation: Little-Endian vs Big-Endian (2)



(a)



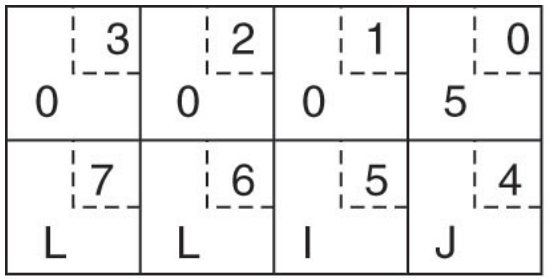
(b)



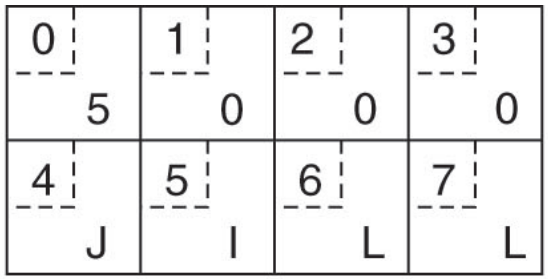
(c)

a. Little endian (data before sending)
b. Big endian (data after receiving) What is the integer 5 read as on the big-endian machine?

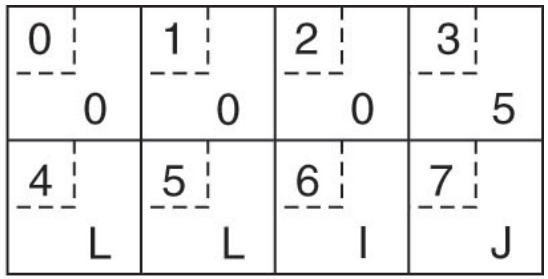
Representation: Little-Endian vs Big-Endian (2)



(a)



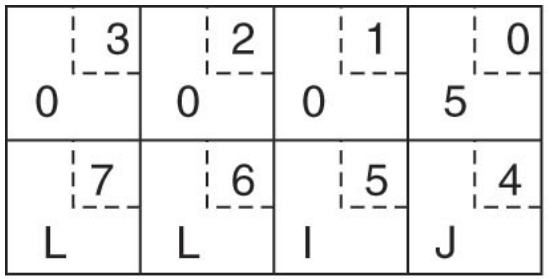
(b)



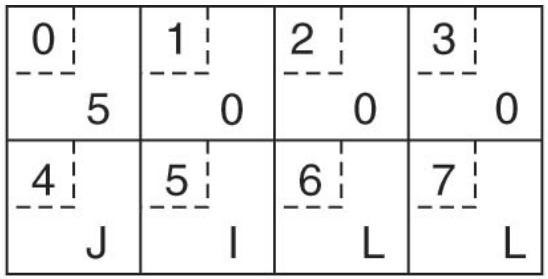
(c)

a. Little endian (data before sending)
b. Big endian (data after receiving) What is the integer 5 read as on the big-endian machine?
c. Big endian (after reversing bytes on received data)

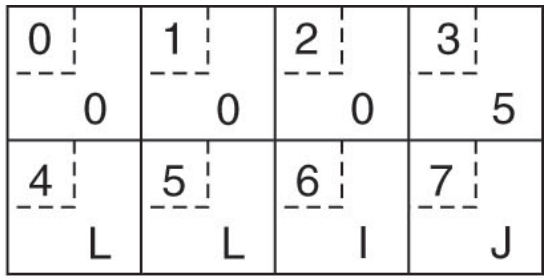
Representation: Little-Endian vs Big-Endian (2)



(a)



(b)



(c)

a. Little endian (data before sending)
b. Big endian (data after receiving) What is the integer 5 read as on the big-endian machine?
c. Big endian (after reversing bytes on received data)

Does reversing the bytes fix the problem for all data types?

Representation: Little-Endian vs Big-Endian (3)

- **In-class Exercise 1**. How would you write a C program to determine the *endianness* of the underlying system?

Representation: Little-Endian vs Big-Endian (3)

- **In-class Exercise 1**. How would you write a C program to determine the *endianness* of the underlying system? See example byteorder.c in examples/sockets-C/misc folder.

Representation: Little-Endian vs Big-Endian (3)

- **In-class Exercise 1**. How would you write a C program to determine the *endianness* of the underlying system? See example byteorder.c in examples/sockets-C/misc folder.
- **In-class Exercise 2**. Do we need to worry about *endianness* for server and clients written in Java?

Passing Reference Parameters

- *Passing an array or simple structure by reference*: Copy the array into the message. The server stub then calls the server with a pointer to this copy.Changes to the array happen in the message buffer in the server stub that then send back to the client stub, which then copies it back to the client.

Passing Reference Parameters

- *Passing an array or simple structure by reference*: Copy the array into the message. The server stub then calls the server with a pointer to this copy.Changes to the array happen in the message buffer in the server stub that then send back to the client stub, which then copies it back to the client.
- Replaces call-by-reference with call-by-copy/restore. Works in most cases.

Passing Reference Parameters

- *Passing an array or simple structure by reference*: Copy the array into the message. The server stub then calls the server with a pointer to this copy.Changes to the array happen in the message buffer in the server stub that then send back to the client stub, which then copies it back to the client.
- Replaces call-by-reference with call-by-copy/restore. Works in most cases.
- How about arbitrary data structures with pointers? 

RPC Protocol and Stub Generation

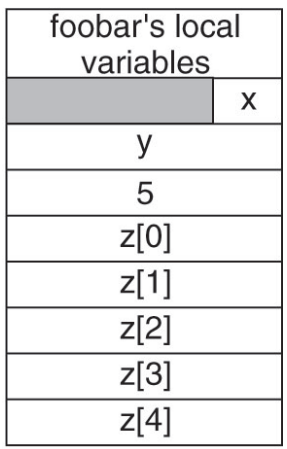
- The RPC protocol would have to define the format of the message, the representation of primitive types and arrays and other data structures. Are integers stored in 2's complement, characters in 16-bit Unicode, floats/doubles in IEEE standard #754 and if everything is big-endian or little-endian?

RPC Protocol and Stub Generation

- The RPC protocol would have to define the format of the message, the representation of primitive types and arrays and other data structures. Are integers stored in 2's complement, characters in 16-bit Unicode, floats/doubles in IEEE standard #754 and if everything is big-endian or little-endian?
- An example:

```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)



(b)

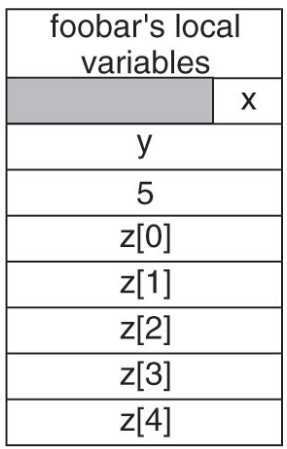
:

RPC Protocol and Stub Generation

- The RPC protocol would have to define the format of the message, the representation of primitive types and arrays and other data structures. Are integers stored in 2's complement, characters in 16-bit Unicode, floats/doubles in IEEE standard #754 and if everything is big-endian or little-endian?
- An example:

```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)



(b)

:

- Interface Definition Language (IDL) is used to define interfaces for RPC. IDL is then compiled into client and server stub along with the appropriate compile/run-time interfaces.

Asynchronous RPC (1)

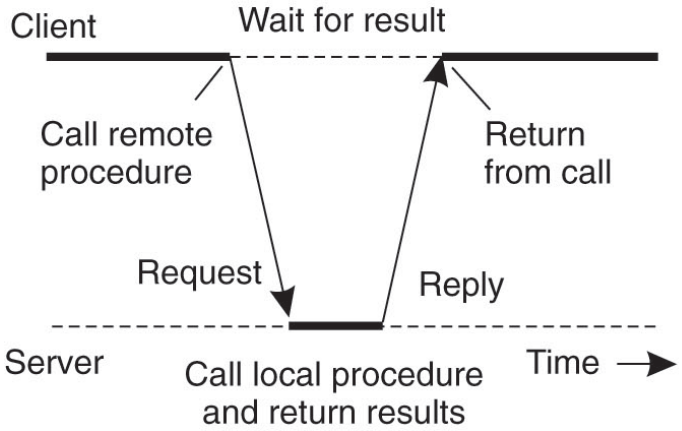
- Asynchronous RPC: A client immediately continues once the server accepts the RPC request. The server executes the RPC request after the acknowledgment.

Asynchronous RPC (1)

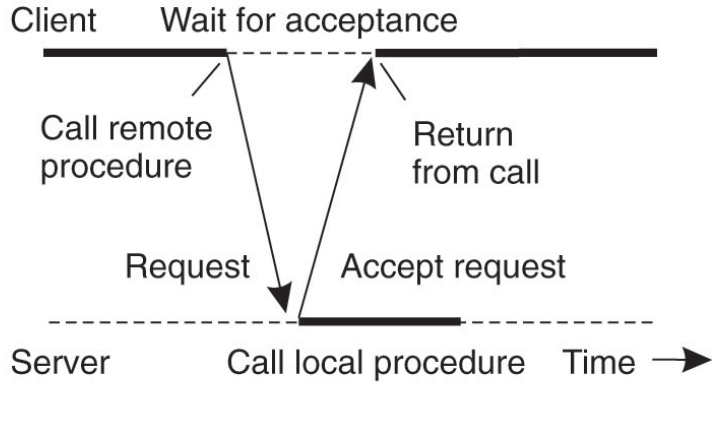
- Asynchronous RPC: A client immediately continues once the server accepts the RPC request. The server executes the RPC request after the acknowledgment.
- Deferred asynchronous RPC: The client calls the server with a RPC request and the server immediately acknowledges it. Later the server does a callback to the client with the result.

Asynchronous RPC (2)

- Synchronous (default choice) versus Asynchronous RPC



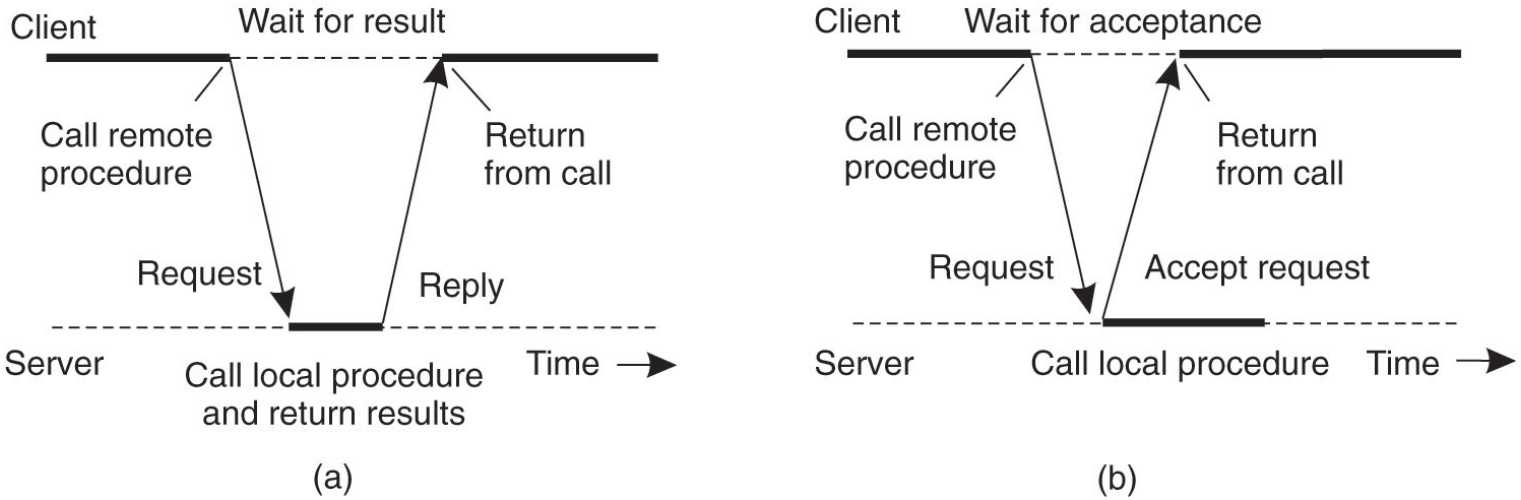
(a)



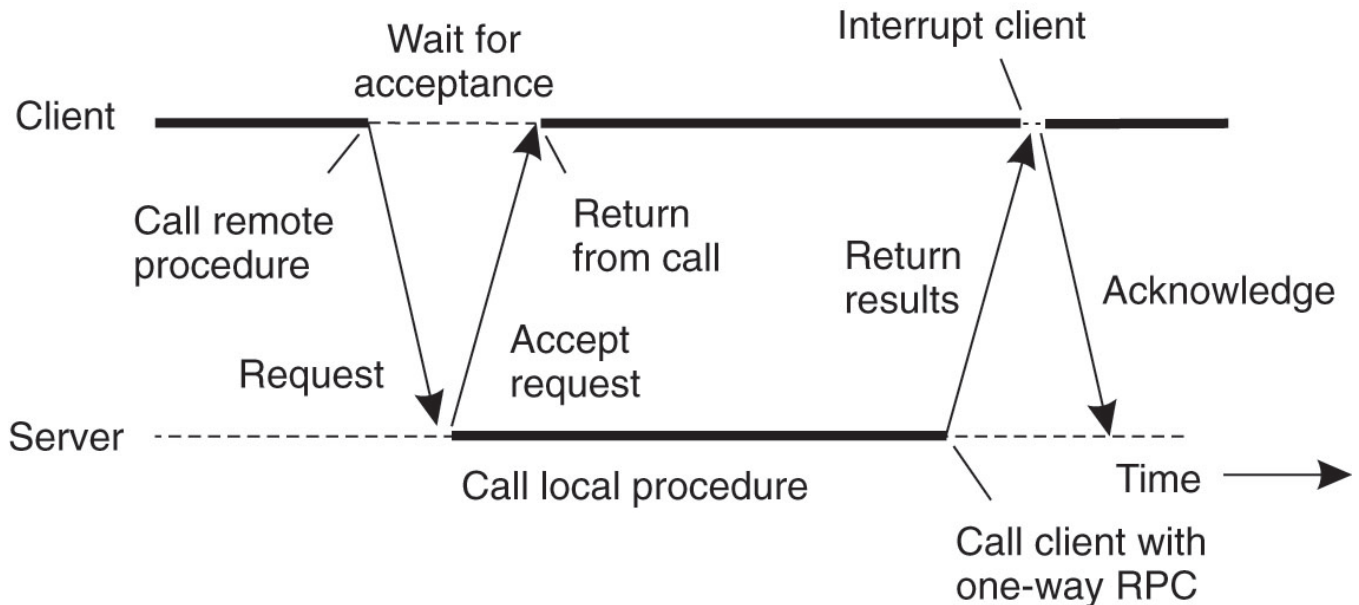
(b)

Asynchronous RPC (2)

- Synchronous (default choice) versus Asynchronous RPC



- Deferred Asynchronous RPC



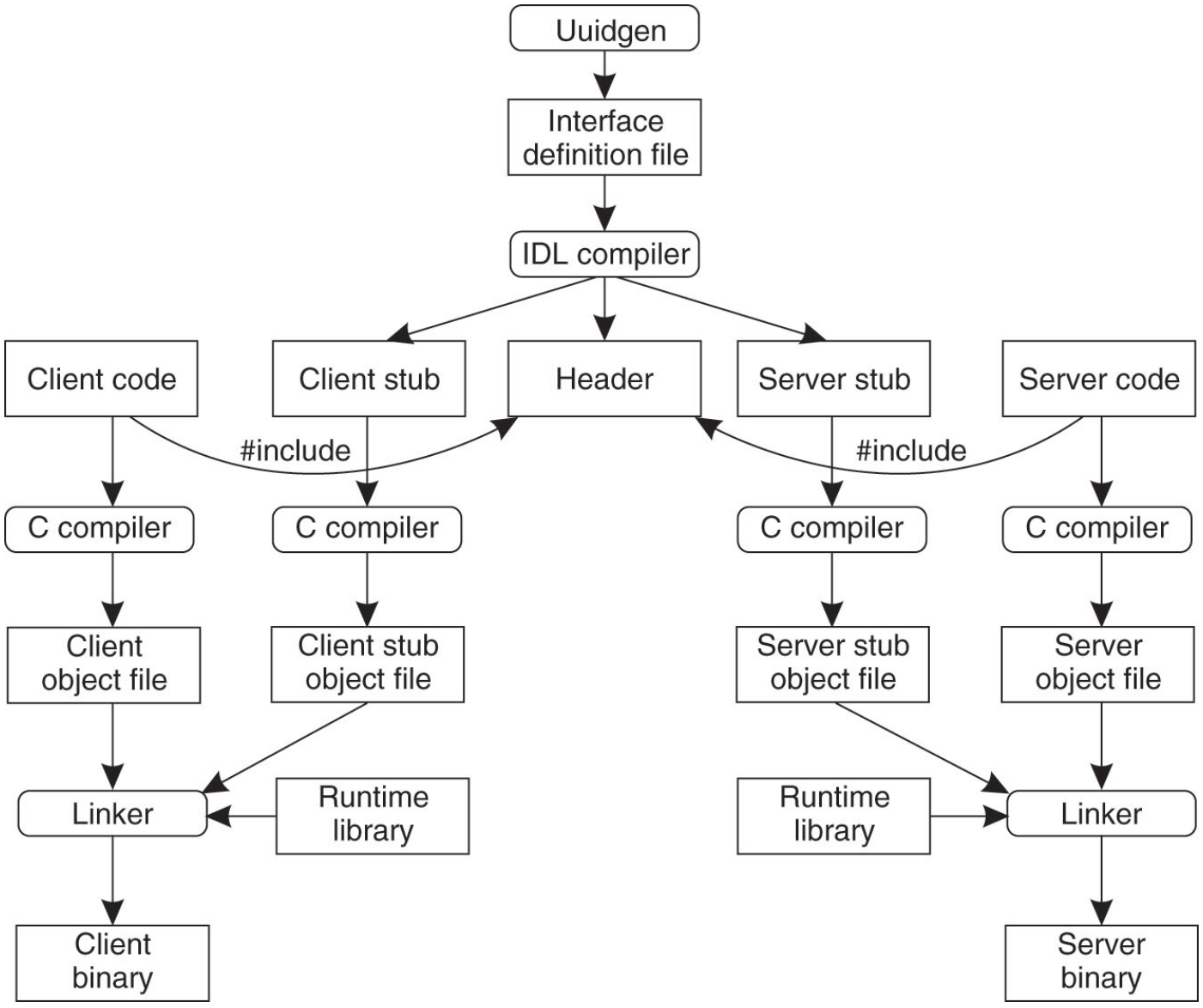
Classic RPC Implementations

- Distributed Computing Environment/Remote Procedure Calls (DCE/RPC) was commissioned by the Open Software Foundation, an industry consortium now renamed as the Open Group. The DCE/RPC specifications were adopted in Microsoft's base for distributed computing, DCOM.

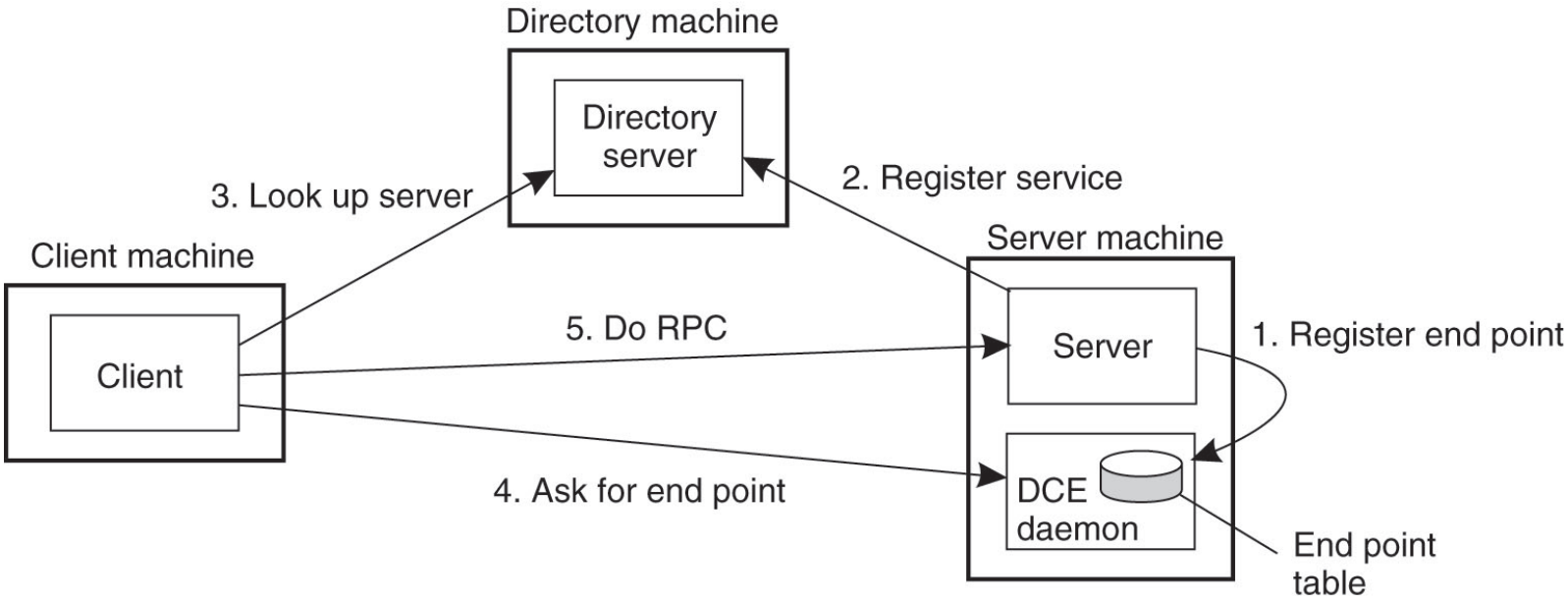
Classic RPC Implementations

- Distributed Computing Environment/Remote Procedure Calls (DCE/RPC) was commissioned by the Open Software Foundation, an industry consortium now renamed as the Open Group. The DCE/RPC specifications were adopted in Microsoft's base for distributed computing, DCOM.
- Open Network Computing Remote Procedure Call (ONC RPC) is a widely deployed remote procedure call system. ONC was originally developed by Sun Microsystems as part of their Network File System project, and is sometimes referred to as Sun ONC or Sun RPC.

Building a RPC Server and Client



Binding a Client to a RPC Server



Note: ONC RPC uses a `portmapper` in place of the DCE daemon. It doesn't use a directory server.

Issues with RPCs

How to deal with network and system errors?

Issues with RPCs

How to deal with network and system errors?

- At-most-once operation: No call is carried out more than once even if the system crashes.

Issues with RPCs

How to deal with network and system errors?

- At-most-once operation: No call is carried out more than once even if the system crashes.
- At-least-once operation: The call is tried multiple time until it succeeds. It is possible for the call to happen multiple times.

Issues with RPCs

How to deal with network and system errors?

- At-most-once operation: No call is carried out more than once even if the system crashes.
- At-least-once operation: The call is tried multiple time until it succeeds. It is possible for the call to happen multiple times.
  - This only works for idempotent operations: something that can be done multiple times without harm.

More RPC Implementations

- Java RMI (Remote Method Invocation).

More RPC Implementations

- Java RMI (Remote Method Invocation).
- XML-RPC and its successor SOAP: An RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism.

More RPC Implementations

- Java RMI (Remote Method Invocation).
- XML-RPC and its successor SOAP: An RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism.
- Microsoft .NET Remoting offers RPC/RMI facilities for distributed systems implemented on the Windows platform.

More RPC Implementations

- Java RMI (Remote Method Invocation).
- XML-RPC and its successor SOAP: An RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism.
- Microsoft .NET Remoting offers RPC/RMI facilities for distributed systems implemented on the Windows platform.
- CORBA provides remote procedure invocation through an intermediate layer called the object request broker.

More RPC Implementations

- Java RMI (Remote Method Invocation).
- XML-RPC and its successor SOAP: An RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism.
- Microsoft .NET Remoting offers RPC/RMI facilities for distributed systems implemented on the Windows platform.
- CORBA provides remote procedure invocation through an intermediate layer called the object request broker.
- Facebook's Thrift protocol and framework.

More RPC Implementations

- Java RMI (Remote Method Invocation).
- XML-RPC and its successor SOAP: An RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism.
- Microsoft .NET Remoting offers RPC/RMI facilities for distributed systems implemented on the Windows platform.
- CORBA provides remote procedure invocation through an intermediate layer called the object request broker.
- Facebook's Thrift protocol and framework.
- Google Protocol Buffers combined with gRPC.

More RPC Implementations

- Java RMI (Remote Method Invocation).
- XML-RPC and its successor SOAP: An RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism.
- Microsoft .NET Remoting offers RPC/RMI facilities for distributed systems implemented on the Windows platform.
- CORBA provides remote procedure invocation through an intermediate layer called the object request broker.
- Facebook's Thrift protocol and framework.
- Google Protocol Buffers combined with gRPC.
- Apache Avro is a RPC and data serialization framework developed for the Apcahe Hadoop project.

References

- Wikipedia article on Endianness
- Linus Torvalds on Endianness
-
-