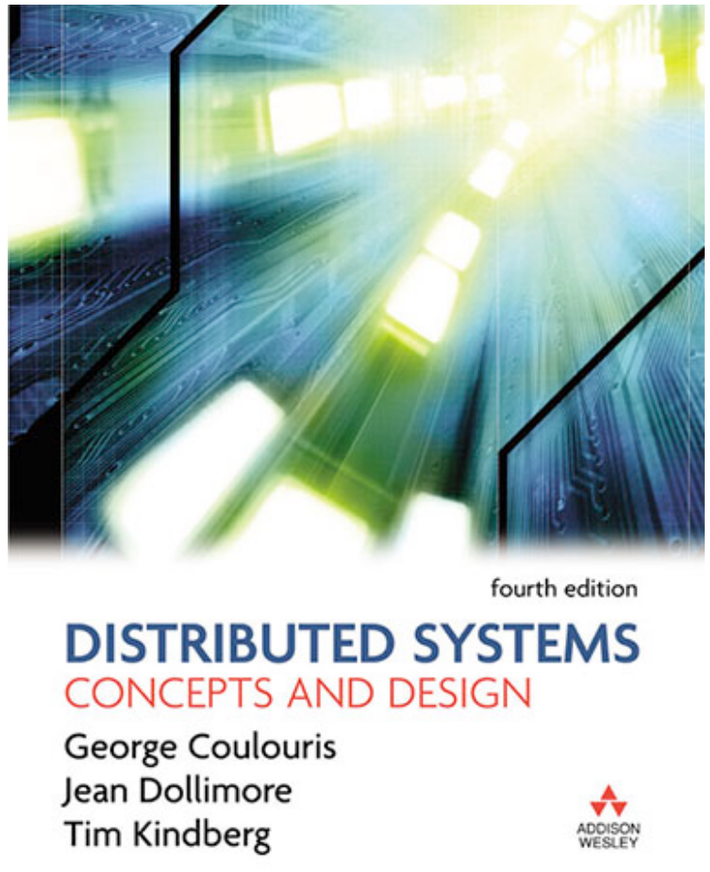
Distributed Objects and Remote Invocation

5.1 Introduction
5.2 Communication between Distributed Objects (RMI)
5.4 Events and Notifications



Paradigms for Distributed Programming

- RPC (Remote Procedure Call): the earliest and perhaps the best-known programming model for distributed programming. It *allows client programs to call procedures in server programs running in separate processes and generally in different computers from the client*.

Paradigms for Distributed Programming

- RPC (Remote Procedure Call): the earliest and perhaps the best-known programming model for distributed programming. It *allows client programs to call procedures in server programs running in separate processes and generally in different computers from the client*.

- RMI (Remote Method Invocation): extension of local method invocation of object-oriented programming. It *allows an object living in one process to invoke the methods of an object living in another process*.
  Most famous example: Java RMI (Lab 3!)

Paradigms for Distributed Programming

- RPC (Remote Procedure Call): the earliest and perhaps the best-known programming model for distributed programming. It *allows client programs to call procedures in server programs running in separate processes and generally in different computers from the client*.

- RMI (Remote Method Invocation): extension of local method invocation of object-oriented programming. It *allows an object living in one process to invoke the methods of an object living in another process*.
  Most famous example: Java RMI (Lab 3!)

- Event-Based Programming: *allows objects to receive notification of the events at other objects in which they have registered interest*.

Remote Method Invocation (RMI)

Let Us Start from Scratch: the Object Model in a Nutshell

- An object-oriented program (Java, C++, ...) consists of a collection of interacting objects, each of which consists of a set of data and a set of methods.

- An object can communicates with other objects by invoking their methods, generally passing arguments and receiving results.

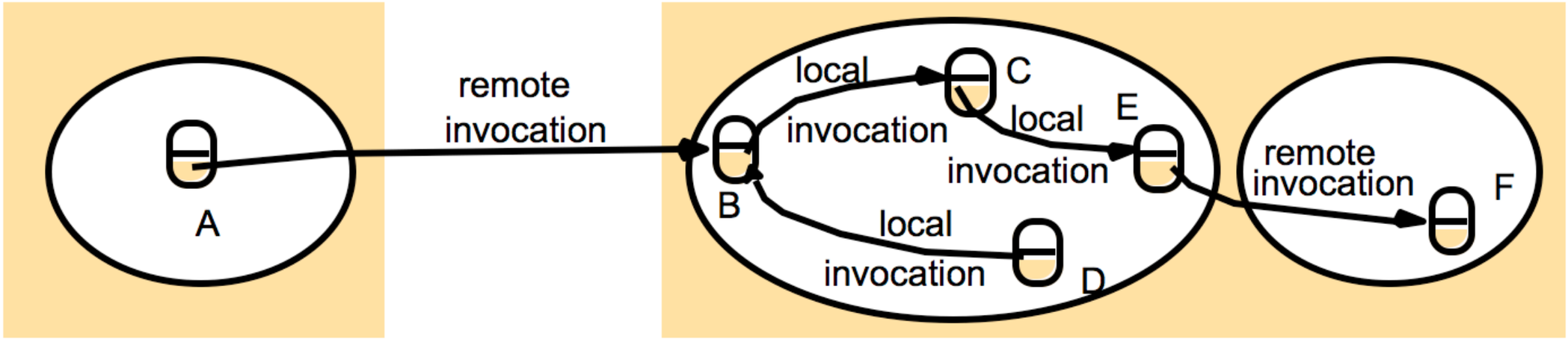
- Objects can encapsulate their data and the code of their methods.

- Some languages (JAVA, C++) allow programmers to define objects whose instance variables can be accessed directly.

- In a distributed object system, an object's data should be accessible only via its **methods (or interface)**.

The Distributed Object Model

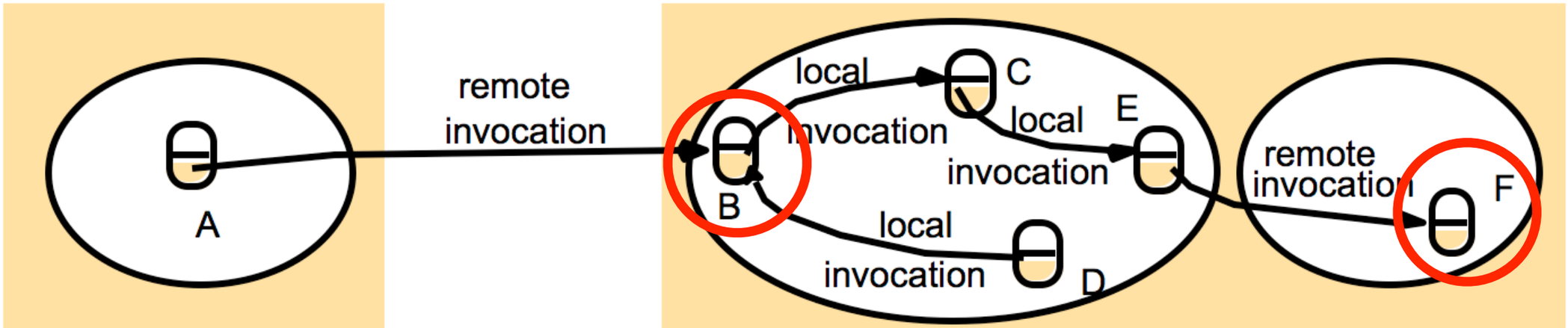
- Each process contains a collection of objects
  - some of which can receive both local and remote invocations
  - whereas the other objects can receive only local invocations.



- Method invocations between objects in different processes, whether ***in the same computer or not***, are known as *remote method invocations*.
- Method invocations between objects in the same process are *local method invocations*.

Remote Objects

- Remote objects: objects that can receive remote invocations.



- Fundamental concepts of the distributed object model:
  - Objects can invoke the methods of a remote object if they have access to its *remote object reference*.
  - Every remote object has a *remote interface* that specifies which of its methods can be invoked remotely.

Remote Object Reference

- A remote object reference is an identifier that can be used throughout a distributed system to refer to a particular *unique* remote object.

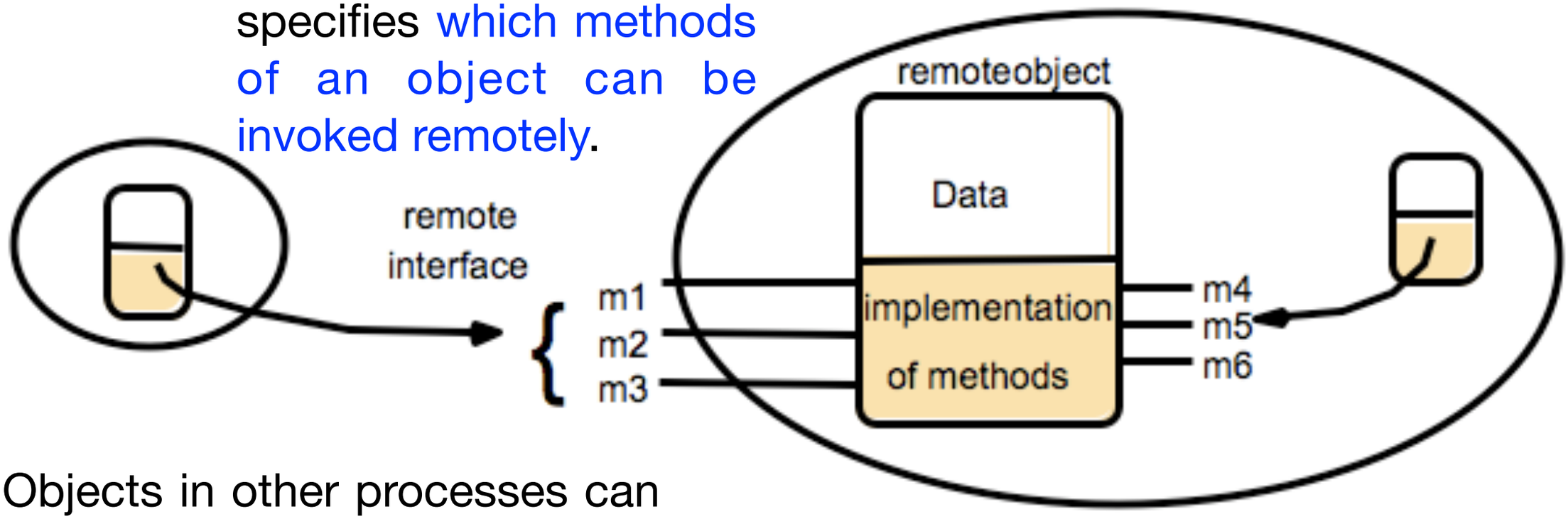
- A remote object reference is passed in the invocation message to specify which object is to be invoked.

- Remote object references are analogous to local ones in that:
  - the remote object to receive a remote method invocation is specified by the invoker as a remote object reference
  - remote object references may be passed as arguments and results of remote method invocations.

Remote Interface

The remote interface specifies which methods of an object can be invoked remotely.

The class of a remote object implements the methods of its remote interface.



Objects in other processes can invoke *only* the methods that belong to the remote interface of a remote object.

Local objects can invoke the methods in the remote interface as well as other methods implemented by a remote object.

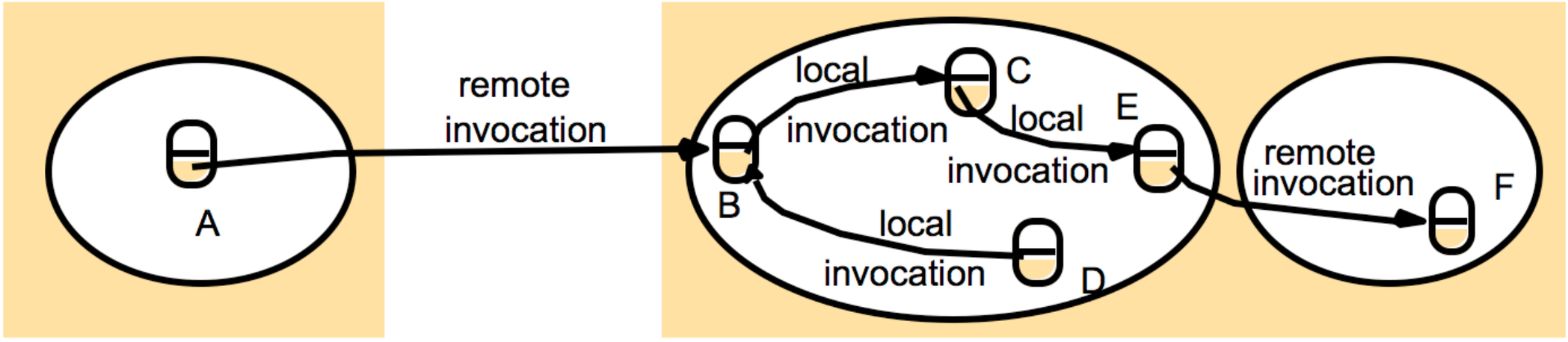
Actions in a Distributed Object System

- An action is initiated by a method invocation, which may result in further invocations on methods in other objects.

- The objects involved in a chain of related invocations may be located in different processes or different computers.

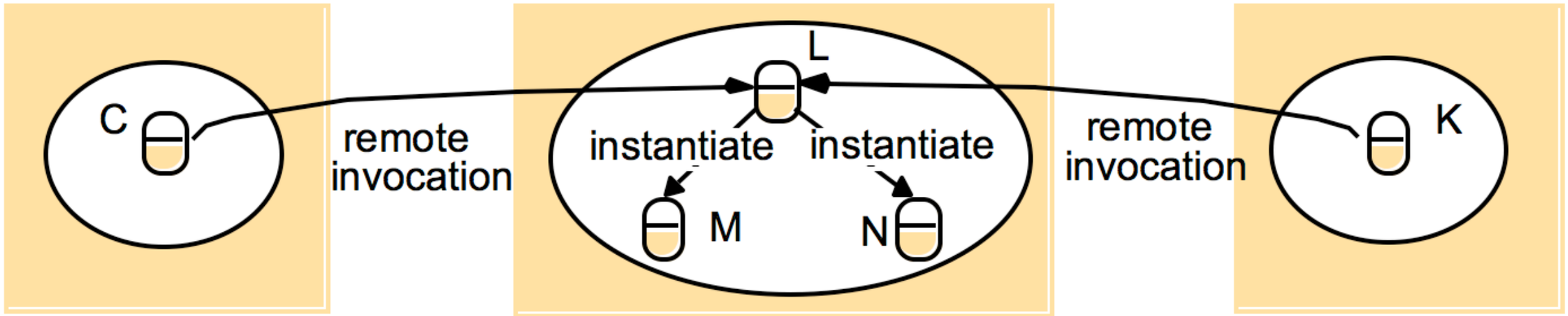
- When an invocation crosses the boundary of a process or computer, RMI is used and the remote reference of the object must be available to the invoker.

- Remote object references may be obtained as the *results of remote method invocations* (example: A might obtain a remote reference to F from B)



Creation of Remote Objects

- When an action leads to the instantiation of a new object, that object will normally live within the process where the instantiation is requested.



- If a newly instantiated object has a remote interface, it will be a *remote object* with a *remote object reference*.

Exceptions

- Any remote invocation may fail for reasons related to the invoked object being in a different process or computer from the invoker.

  Example: the process containing the remote object may have crashed or may be too busy to reply, or the invocation or result message may be lost.

Exceptions

- Any remote invocation may fail for reasons related to the invoked object being in a different process or computer from the invoker.

  Example: the process containing the remote object may have crashed or may be too busy to reply, or the invocation or result message may be lost.

- Remote method invocation should be able to raise exceptions!

  - Timeouts that are due to distribution

  - Exceptions raised during the execution of the method invoked:

    - attempt to read beyond the end of a file
    - attempt to access a file without the correct permissions
    - ...

RMI Invocation Semantics

- Local method invocations are executed exactly **once** (exactly once invocation semantics = every method is executed exactly once).

- *This cannot always be the case for remote method invocation!*

- Request-reply protocols, such as RMI, can be implemented in different ways to provide different delivery guarantees.

- These choices lead to a variety of possible semantics for the reliability of remote invocations as seen by the invoker.

Main Choices for Implementing RMI

- Retry request message: whether to retransmit the request message until either a reply is received or the server is assumed to have failed.

- Duplicate filtering: when retransmissions are used, whether to filter out duplicate requests at the server.

- Retransmission of results: whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations at the server

- Combination of these choices lead to a variety of possible semantics for the reliability of remote invocations: Maybe, At-least-once, At-most-once.

RMI Invocation Semantics: Maybe

- The remote method may be executed once or not at all.
- Maybe semantics arises when no fault tolerance measures are applied.
- Useful only for applications in which *occasional* failed invocations are acceptable.
- This model can suffer from the following types of failure:
  - omission failures if the invocation or result message is lost
  - crash failures when the server containing the remote object fails.

RMI Invocation Semantics: At-Least-Once

- The invoker receives either
  - a result, in which case the invoker knows that the method was executed at least once, or
  - an exception informing it that no result was received.

- Can be achieved by the retrasmission of request messages, masking the omission failures of the invocation or result message.

- This model can suffer from the following types of failure:
  - crash failures when the server containing the remote object fails
  - arbitrary failures, in cases when the invocation message is retransmitted, the remote object may receive it and execute the method more thn once, possibly causing wrong values to be stored or returned.

RMI Invocation Semantics: At-Most-Once

- The invoker receives either
  - a result, in which case the invoker knows that the method was executed exactly once, or
  - an exception informing it that no result was received, in which case the method will have been executed either once or not at all.

- Can be achieved by using a combination of fault tolerance measures (retransmission, duplicate filtering (filtering out duplicate msgs at the server)).
  - The use of retries masks any omission failures of the invocation or result messages.
  - Arbitrary failures are prevented by ensuring that for each RMI a method is never executed more than once.

RMI Invocation Semantics Summary

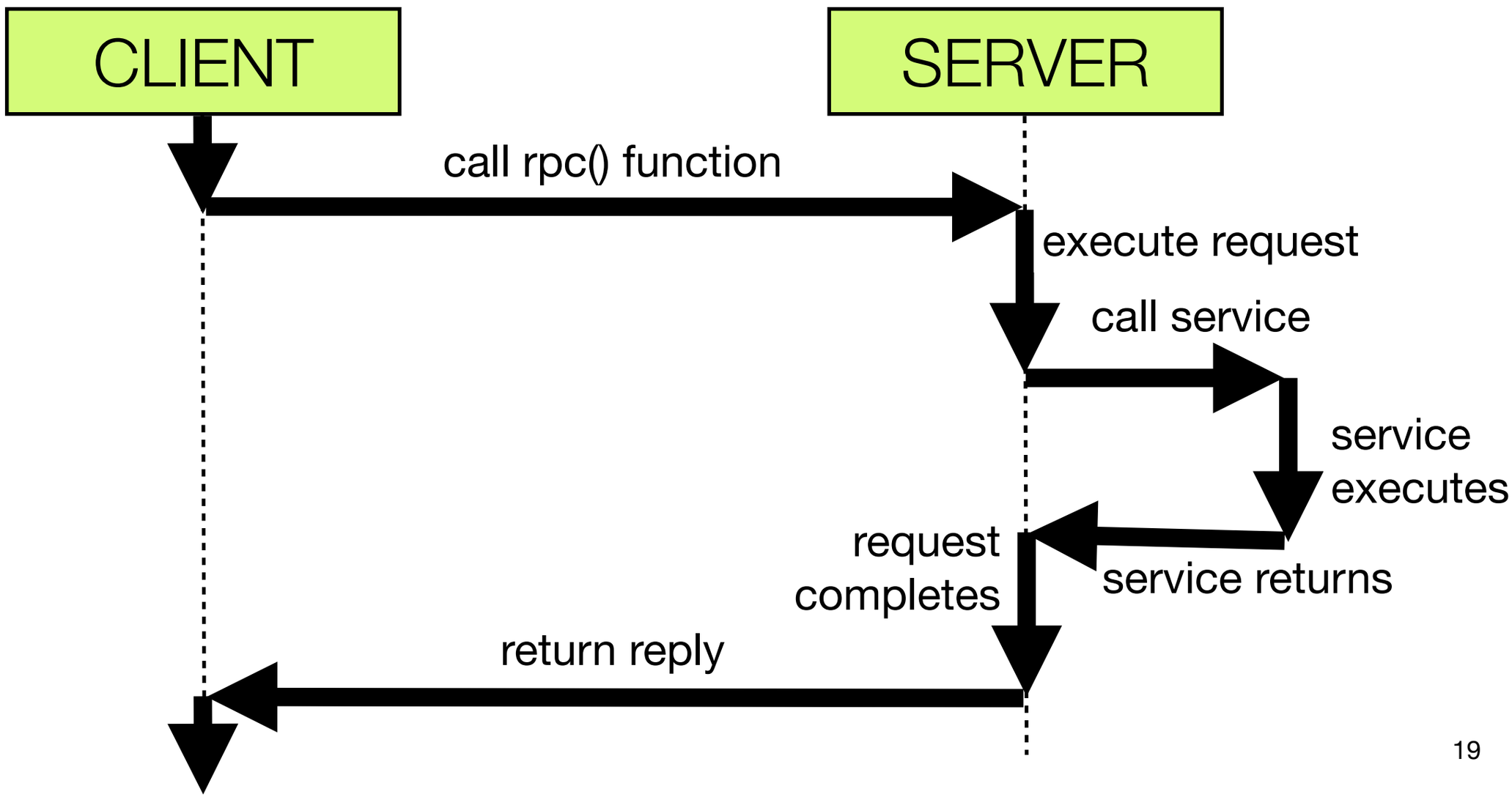
Fault tolerance measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

- In Java RMI the invocation semantics is *at-most-once*.
- In CORBA is *at-most-once* but *maybe* semantics can be requested for methods that do not return results.

Remote Procedure Call (RPC)

Paradigms for Distributed Programming

- RPC (Remote Procedure Call): the earliest and perhaps the best-known programming model for distributed programming. It *allows client programs to call procedures in server programs running in separate processes and generally in different computers from the client*.



RPC and RMI?

- A remote procedure call is very similar to a RMI in that a client program calls a procedure in another program running in a server process.

- Server may be clients of other servers to allow chains of RPCs.

- A server process must define in its service interface the procedures that are available for calling remotely.

- RPC, like RMI, may be implemented to have one of the choices of invocation semantics previously discussed (maybe, at-least-one, at-most-one).

Events and Notifications

Idea Behind the Use of Events

- One object can *react* to a change occurring in another object.

- Notifications of events are essentially asynchronous and determined by their receivers.

- Example: interactive applications
  - the actions that a user performs on objects are seen as *events that causes changes in the objects that maintain the state of the application*
    - manipulating a button with the mouse
    - entering text in a text box via the keyboard
  - the objects that are responsible for displaying a view of the current state are notified whenever the state changes.

Distributed Event-Based Systems

- Distributed event-based-systems allow multiple objects at different locations to be notified of events (actions on objects) taking place at an object.

- Publish-subscribe paradigm:
  - an object that generates events publishes the type of events that it will make available for observation by other objects
  - objects that want to receive notifications from an object that has published its events subscribe to the types of events that are of interest to them.

- Different event types may, for example, refer to the different methods executed by the object of interest.