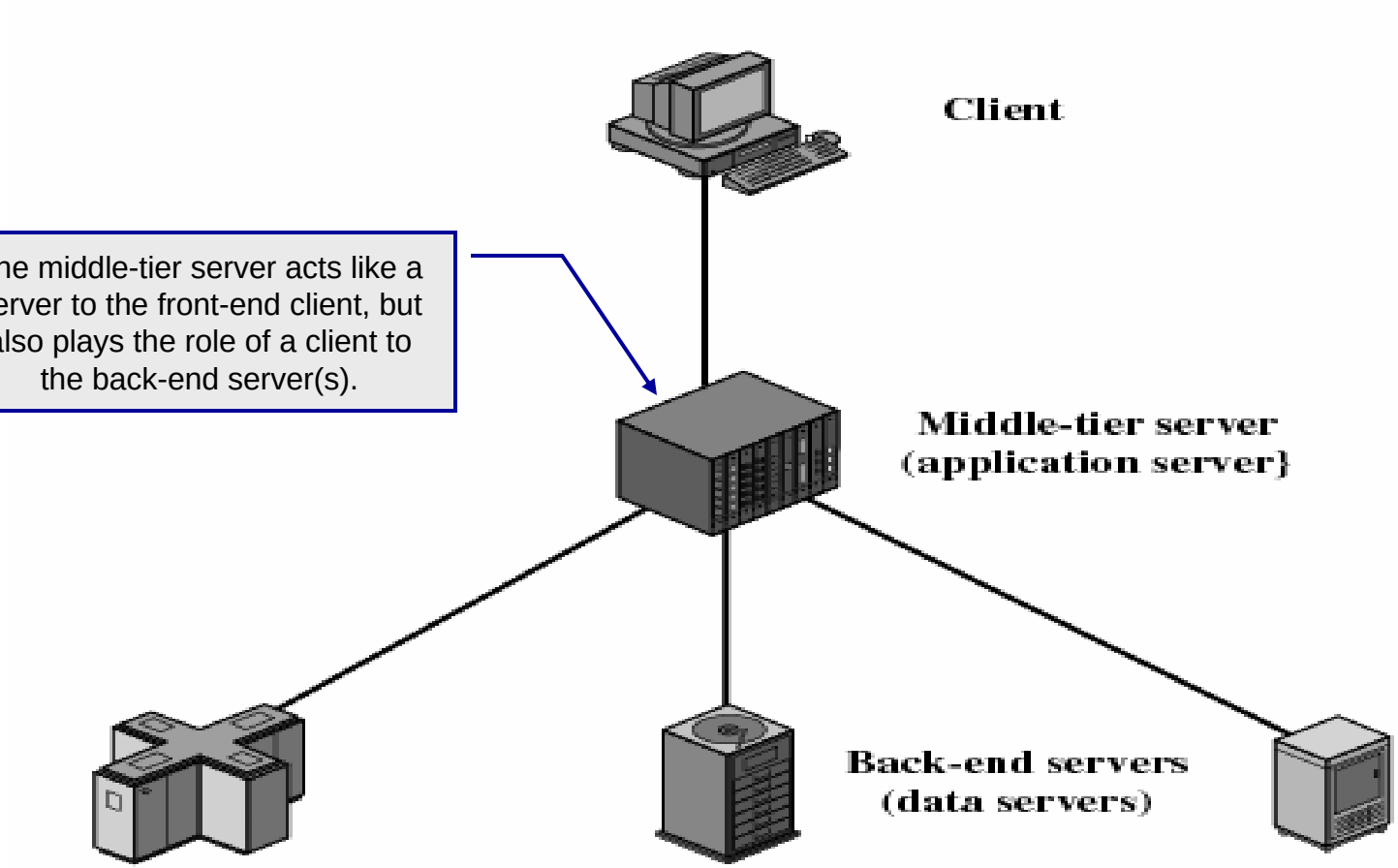
CGS 3763: Operating System Concepts Spring 2006

Client-Server Computing – Part 2

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cgs3763/spr2006

School of Electrical Engineering and Computer Science
University of Central Florida

Three-Tier Client/Server Architecture



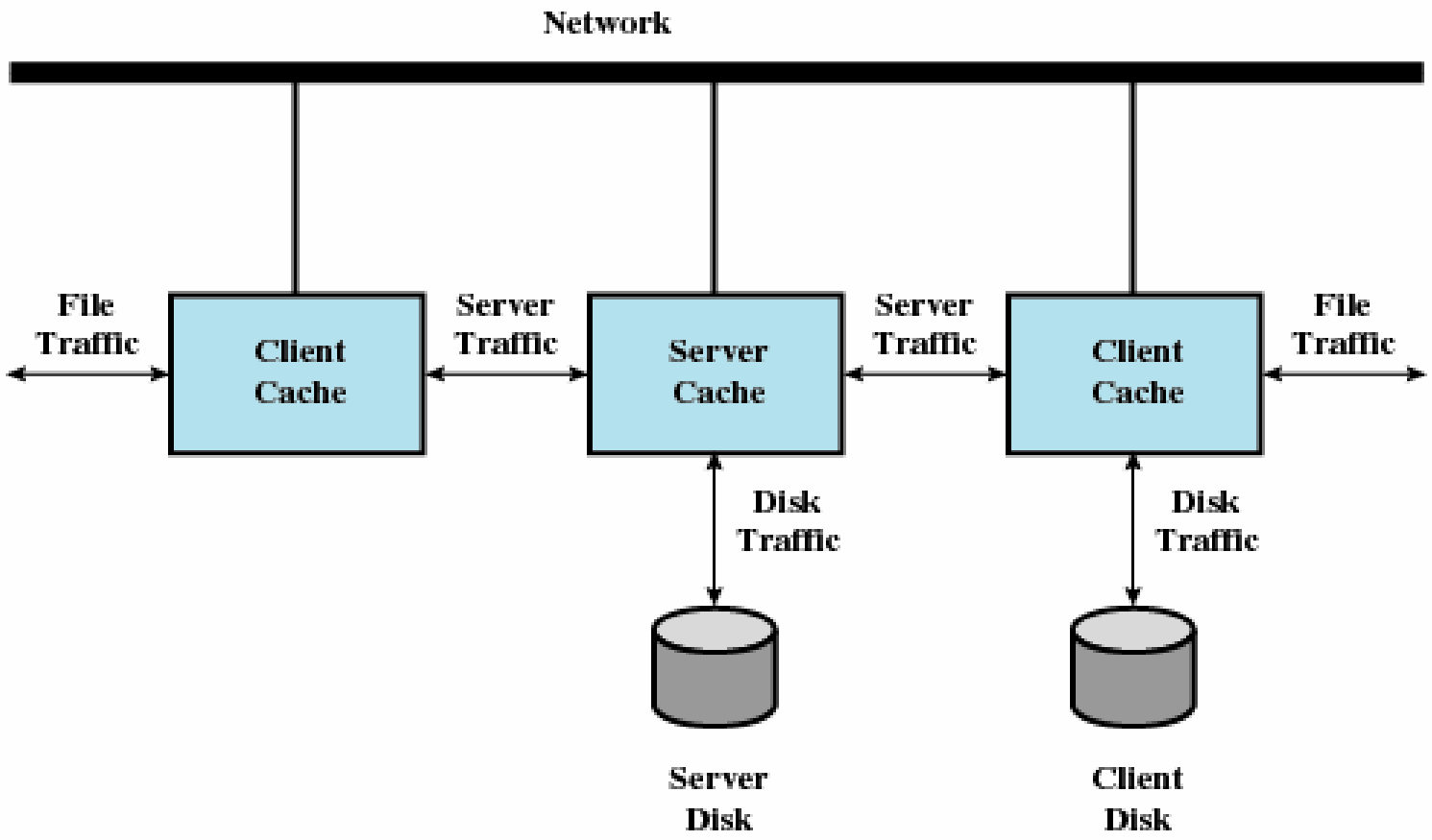
File Cache Consistency

- When a file server is used, performance of file I/O can be noticeably degraded relative to local file access because of the delays imposed by the network.
- To reduce this performance penalty, individual systems can use file caches to hold recently accessed file records.
- Because of the principle of locality (remember this from the page replacement schemes in memory management), use of a local file cache should reduce the number of remote server accesses.
- Caches are consistent when they contain exact copies for remote data.

File Cache Mechanism

- The next page illustrates a typical distributed mechanism for caching files among a networked collection of workstations.
- When a process makes a file access, the request is presented first to the cache of that process's workstation ("file traffic").
- If the request is not satisfied there (a file cache miss), the request is passed either to the local disk, if the file is stored there ("disk traffic"), or to a file server, where the file is stored ("server traffic").
- At the server, the server's cache is first interrogated and, if there is a miss, then the server's disk is accessed.
- The dual caching approach is used to reduce communications traffic (client cache) and disk I/O (server cache).

Typical Distributed File Cache Mechanism



File Cache Consistency

- When caches always contain exact copies of remote data, we say that the caches are consistent.
- It is possible for caches to become inconsistent when the remote data are changes and the corresponding obsolete local cache copies are not discarded.
  - This can happened if one client modifies a file that is also cached by other clients.
- The difficulty actually occurs at two levels:
  1. If a client adopts a policy of immediately writing any changes to a file back to the server, then any other client that has a cache copy of the relevant portion of the file will have obsolete data.
  2. The problem is even worse if the client delays writing back changes to the server. In this case, the server itself has an obsolete version of the file, and new file read requests to the server may obtain obsolete data.
- The problem of keeping local cache copies up to date to changes in remote data is known as the cache consistency problem.

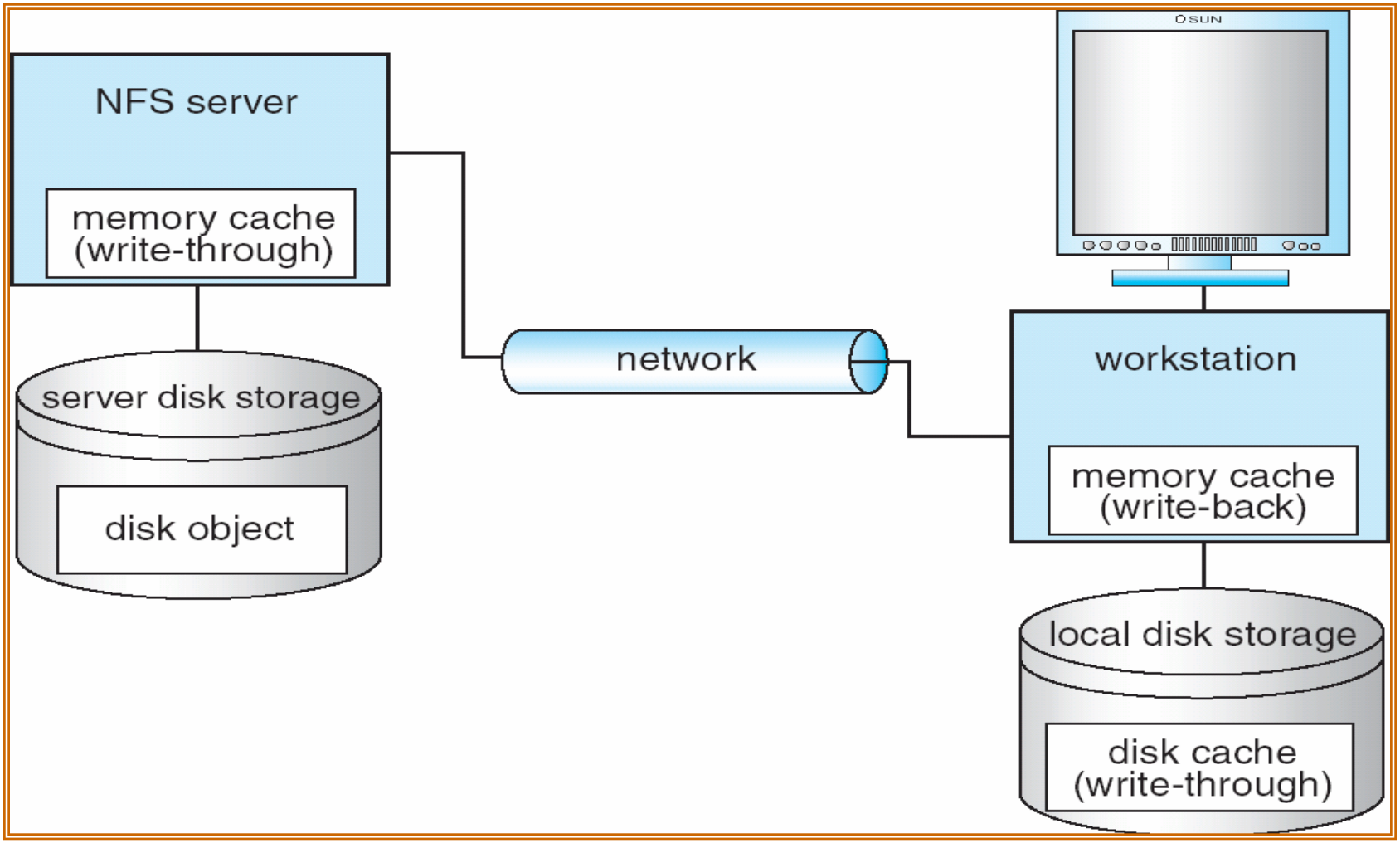
File Cache Consistency (cont.)

- The simplest approach to cache consistency is to use file-locking techniques to prevent simultaneous access to a file by more than one client.
  - This guarantees consistency at the expense of performance and flexibility.
- A more powerful approach (as used in the Sprite Network OS) operates as follows:
  - Any number of remote processes may open a file or reading and create their own client cache.
  - When an open file request to a server requests write access and other processes have the file open for read access, the server takes two actions.
    - First, it notifies the writing process that, although it may maintain a cache, it must write back all altered blocks immediately upon update. There can be at most one such client (one writing process, many reading processes).
    - Second, the server notifies all reading processes that have the file open that the file is no longer cacheable.

File Cache Consistency (cont.)

- **Write-through** – write data through to disk as soon as they are placed on any cache.
  - Reliable, but poor performance.
- **Delayed-write** – modifications written to the cache and then written through to the server later.
  - Write accesses complete quickly; some data may be overwritten before they are written back, and so need never be written at all.
  - Poor reliability; unwritten data will be lost whenever a user machine crashes.
  - Variation – scan cache at regular intervals and flush blocks that have been modified since the last scan.
  - Variation – **write-on-close**, writes data back to the server when the file is closed.
    - Best for files that are open for long periods and frequently modified.

File Cache Consistency (cont.)



File Cache Consistency (cont.)

- Is locally cached copy of the data consistent with the master copy?

- **Client-initiated approach**
  - Client initiates a validity check.
  - Server checks whether the local data are consistent with the master copy.

- **Server-initiated approach**
  - Server records, for each client, the (parts of) files it caches.
  - When server detects a potential inconsistency, it must react.

File Cache Consistency (cont.)

- In caching, many remote accesses handled efficiently by the local cache; most remote accesses will be served as fast as local ones.
- Servers are contracted only occasionally in caching (rather than for each access).
  - Reduces server load and network traffic
  - Enhances potential for scalability
- Remote server method handles every remote access across the network; penalty in network traffic, server load, and performance.
- Total network overhead in transmitting big chunks of data (caching) is lower than a series of responses to specific requests (remote-service).

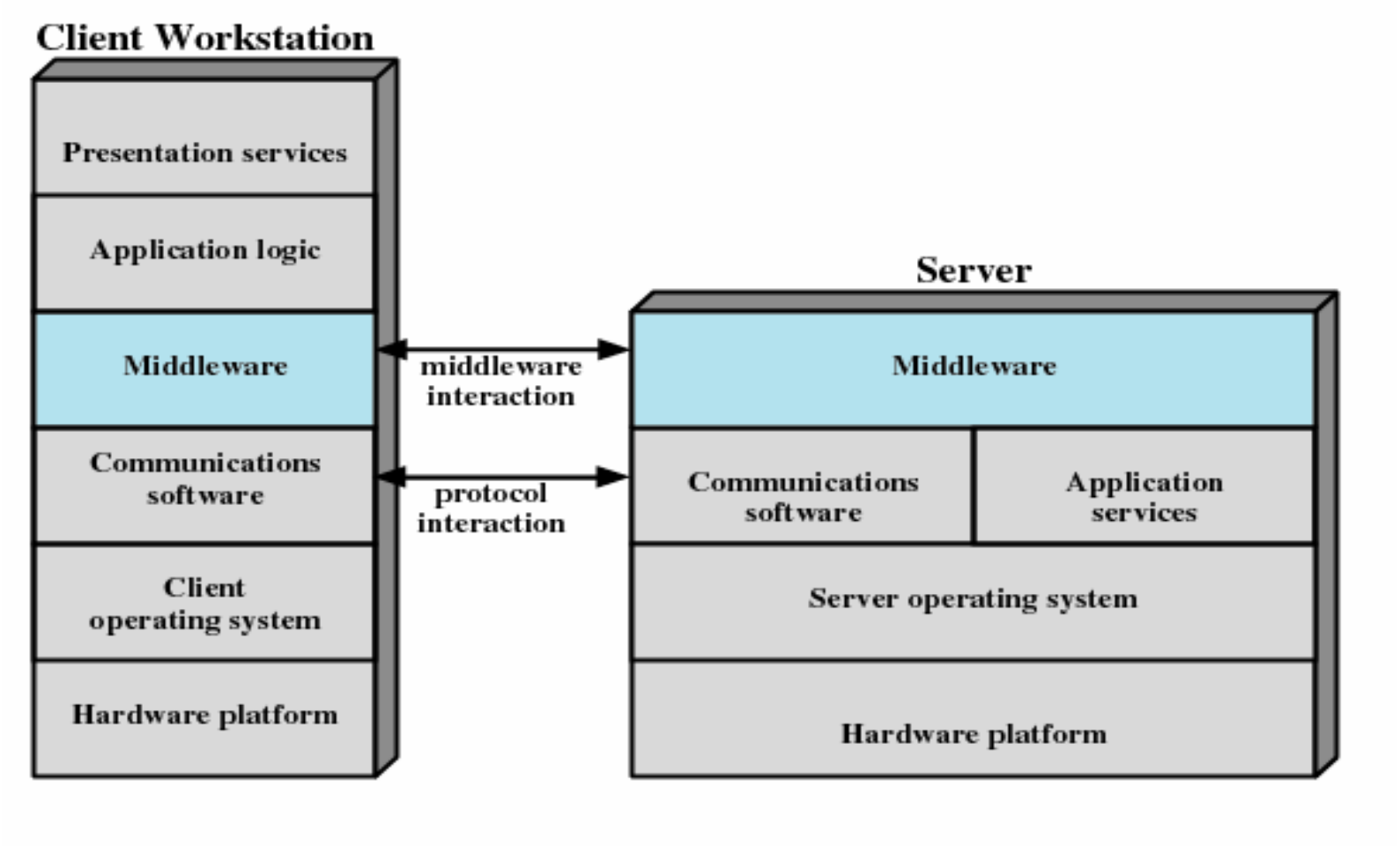
Middleware

- The development and deployment of client-server products has far outpaced efforts to standardize all aspects of distributed computing, from the physical layer up to the application layer.
- The lack of standards makes it difficult to implement an integrated, multi-vendor, enterprise-wide client-server configuration.
- Since much of the benefit of the client-server approach is based in its modularity and the ability to mix and match platforms and applications to provide business solutions, this interoperability problem must be solved.
- To achieve the true benefits of the client-server approach, developers must have a set of tools that provide a uniform means and style of access to system resources across all platforms
- These tools enable programmers to build applications that look and feel the same on various PCs and workstations, but they use the same method to access data regardless of the location of that data.

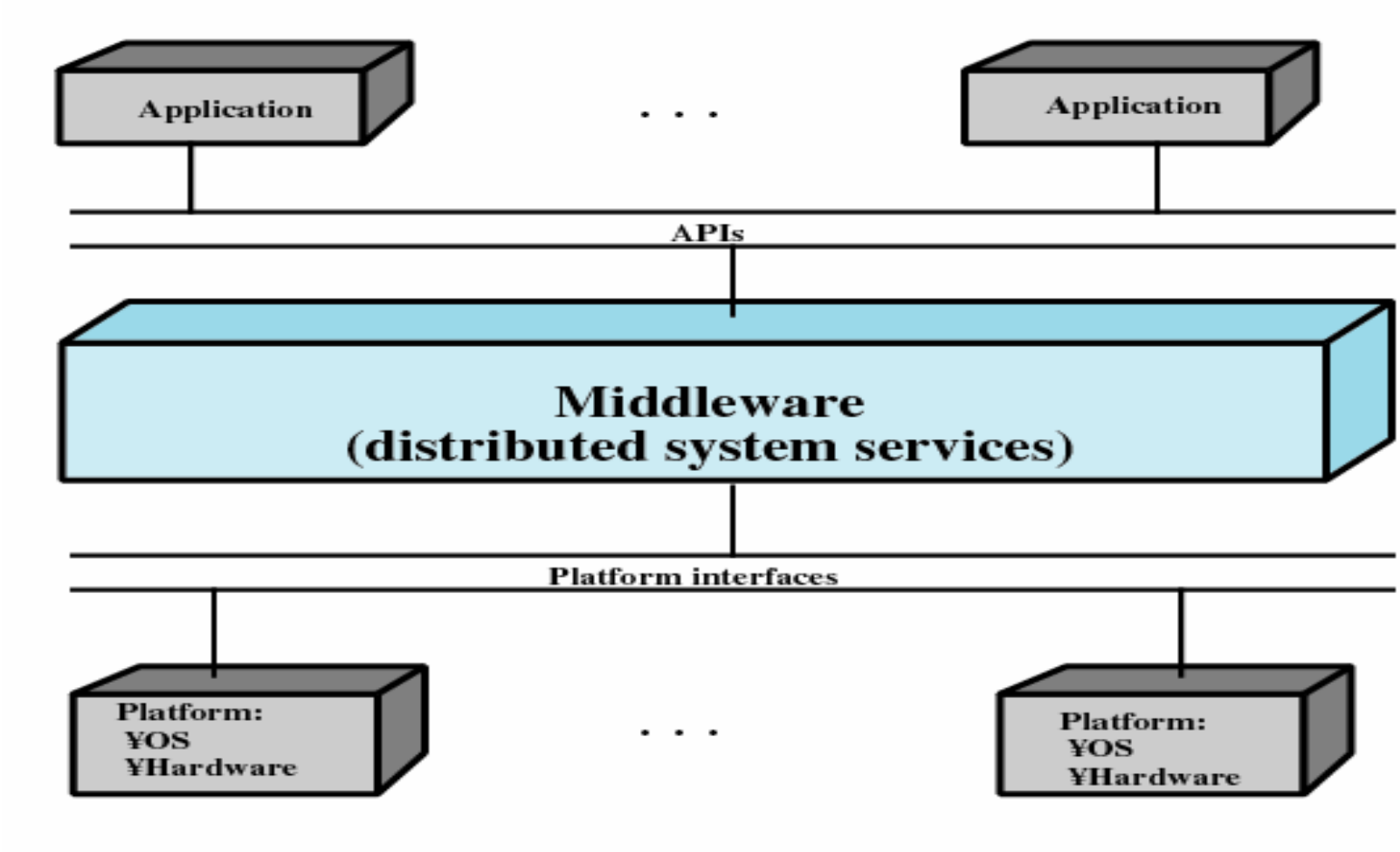
Middleware (cont.)

- The most common way to meet this requirement is by the use of standard programming interfaces and protocols that sit between the application above and the communications software and OS below.
- These standardized interfaces and protocols have become to be referred to as middleware.
- With standard interfaces, it is easy to implement the same application on a variety of server types and workstation types. While this obviously benefits the customers, it also motivates the vendors to provide such interfaces. The reason for this is that customers but applications, not servers; customers will only choose among those server products that run the applications they want.
- The standardized protocols are needed to link these various server interfaces back to the clients that need to access them.
- There are a variety of middleware packages ranging from the very simple to very complex. However, they all share the capability to hide the complexities and disparities of different network protocols and operating systems.

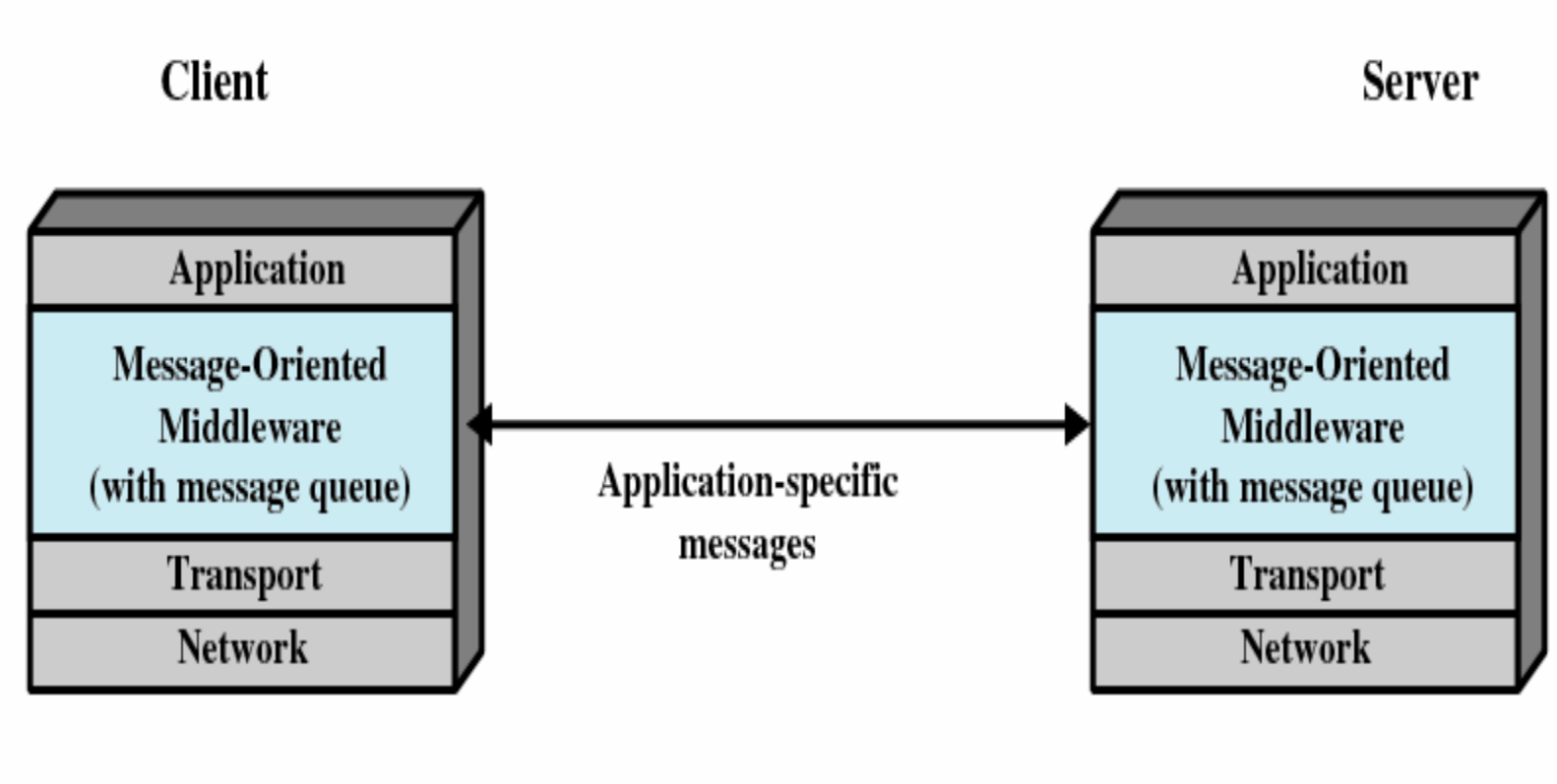
The Role of Middleware in Client-Server Architecture



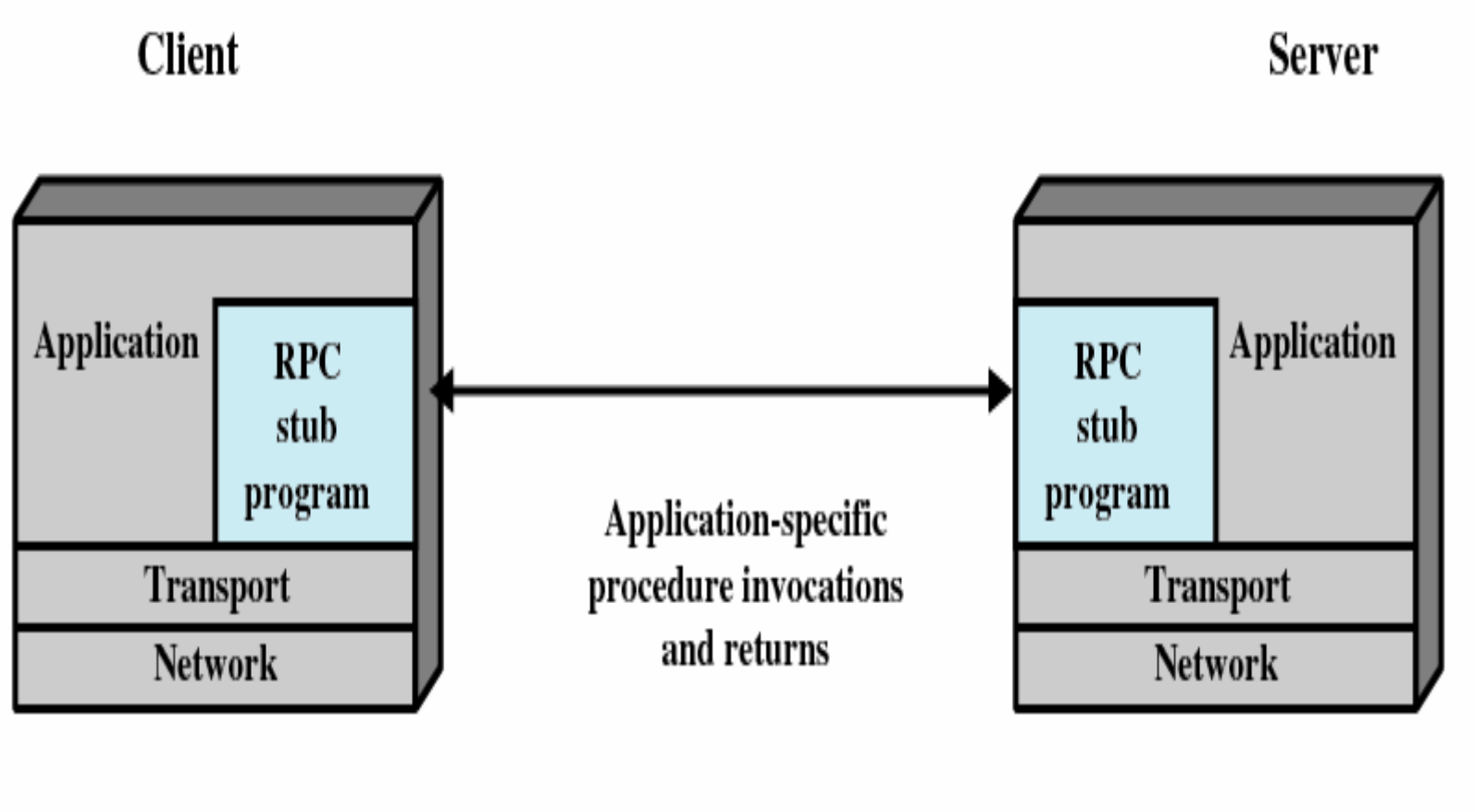
Logical View of Middleware



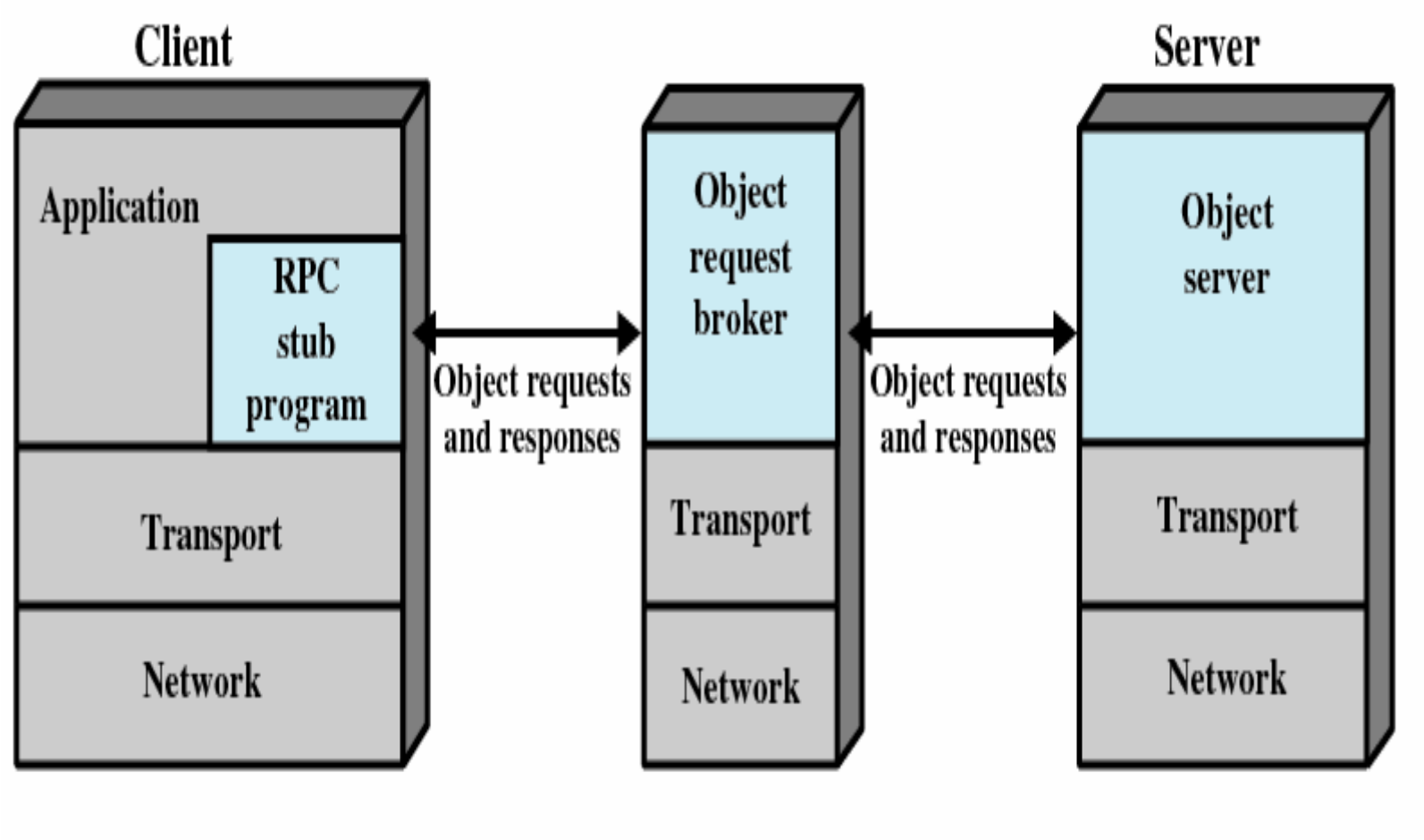
Middleware Mechanisms: Message-oriented



Middleware Mechanisms: Remote Procedure Calls



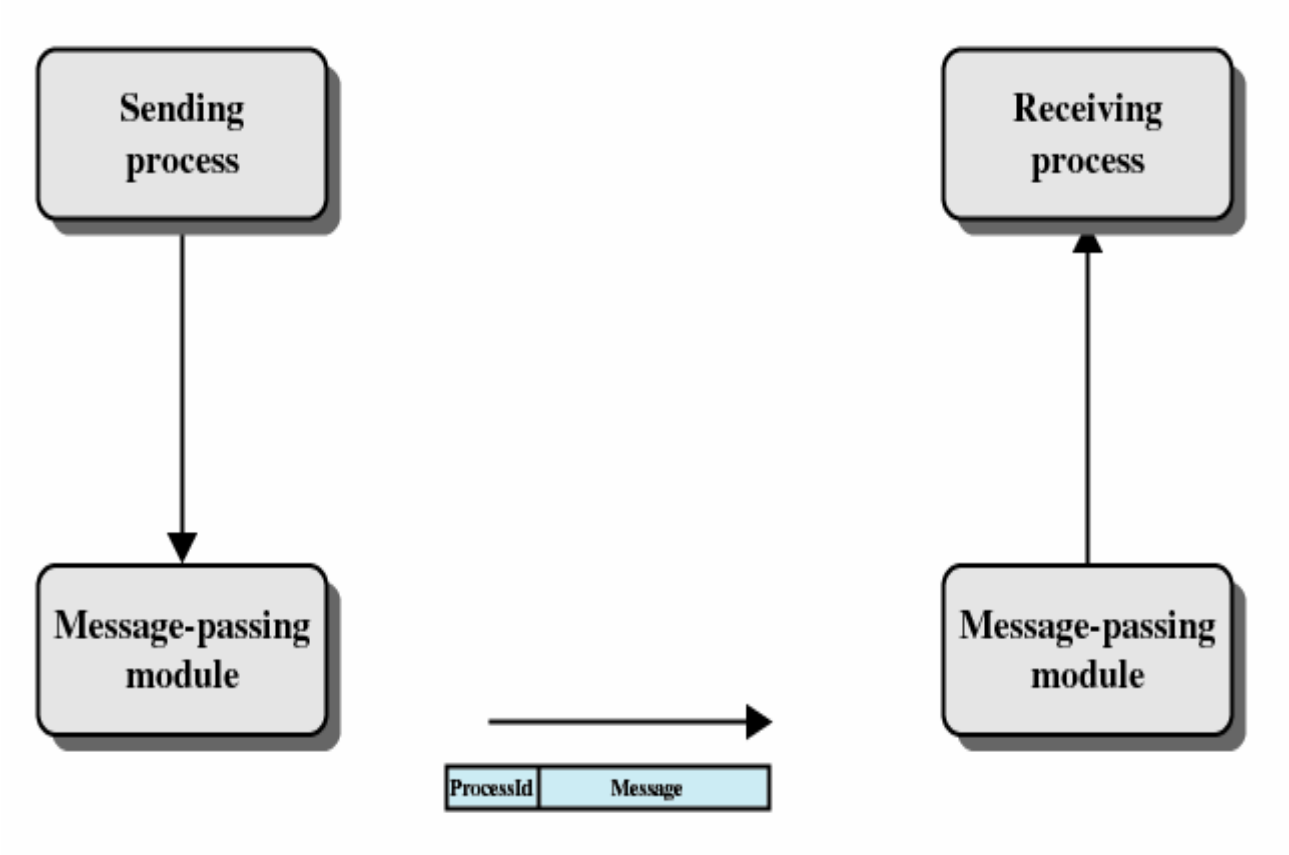
Middleware Mechanisms: Object Request Broker



Distributed Message Passing

- It is typically the case in distributed system that the computers do not share main memory; each is an isolated computer system. Thus, interprocessor communication techniques that rely on shared memory, such as semaphores, cannot be used.
- Instead, techniques that rely on message passing are used.
  - Basic message passing in a single system.
  - Remote procedure calls (built on top of basic message passing).
- The figure on the next page illustrates the basic concept of message passing.

Basic Message-Passing



Reliability Versus Unreliability

- Reliable message-passing guarantees delivery if possible
  - Not necessary to let the sending process know that the message was delivered
- Send the message out into the communication network without reporting success or failure
  - Reduces complexity and overhead

Blocking Versus Nonblocking

- Nonblocking
  - Process is not suspended as a result of issuing a Send or Receive
  - Efficient and flexible
  - Difficult to debug
- Blocking
  - Send does not return control to the sending process until the message has been transmitted
  - OR does not return control until an acknowledgment is received
  - Receive does not return until a message has been placed in the allocated buffer

Remote Procedure Calls

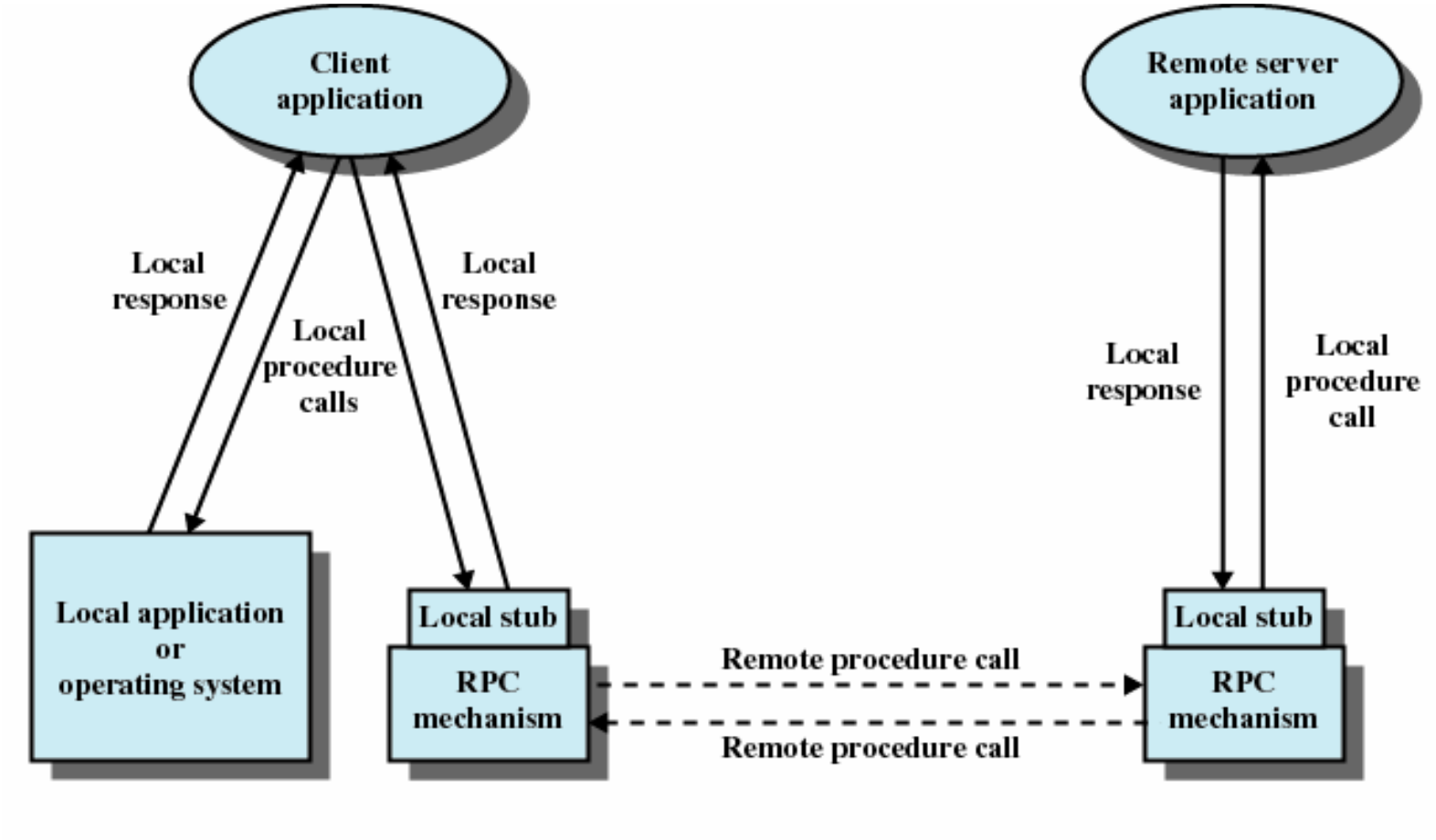
- Remote procedure calls (RPC) are a variation on basic message passing.
- RPC is a widely accepted and common method for encapsulating communication in a distributed system.
- The essence of the technique is to allow programs on different machines to interact using simple procedure call/return semantics, just as if the two programs were on the same machine.
  - That is, the procedure call is used for access to remote services.

Remote Procedure Calls (cont.)

- The popularity of remote procedure calls is based on the following advantages over basic message passing:

1. The procedure call is a widely accepted, used, and understood abstraction.
2. The use of remote procedure calls enables remote interfaces to be specified as a set of named operations with designated types. Thus, the interface can be clearly documented and distributed programs can be statically checked for type errors.
3. Since a standardized and precisely defined interface is specified, the communication code for an application can be generated automatically.
4. Because a standardized and precisely defined interface is specified, developers can write client and server modules that can be moved among computers and operating systems with little modification and recoding.

Remote Procedure Call Mechanism



Client/Server Binding

- Binding specifies the relationship between the remote procedure and the calling program.
- Non-persistent binding
  - A logical connection is established at the time of the remote procedure call. As soon as the values are returned, the connection is terminated. (Conserves resources.)
- Persistent binding
  - The connection is sustained after the procedure returns. The same connection can then be used for subsequent remote procedure calls.

Synchronous versus Asynchronous RPCs

- Synchronous versus asynchronous RPCs is analogous to blocking versus nonblocking message passing.
- Synchronous RPC
  - This is the traditional RPC.
  - Behaves much like a subroutine call.
  - Easy to understand and program, but fails to fully exploit the parallelism inherent in distributed systems, thus lowering performance.
- Asynchronous RPC
  - Does not block the caller. Replies are received as and when they are needed, allowing client execution to proceed locally in parallel with server invocation remotely.
  - A typical asynchronous RPC use is to enable a client to invoke a server repeatedly so that the client has a number of requests in the pipeline at one time, each with its own set of data.
  - Synchronization between the client and server is achieved either by a higher-layer application in the client and server which initiates the exchange and then checks that all the requested actions have been performed, or, the client issues a string of asynchronous RPCs followed by a final synchronized RPC. The server respond only to the synchronized RPC after completing all the work requested in the preceding asynchronous RPCs.