PAYAP UNIVERSITY

Distributed Objects and Remote Invocation

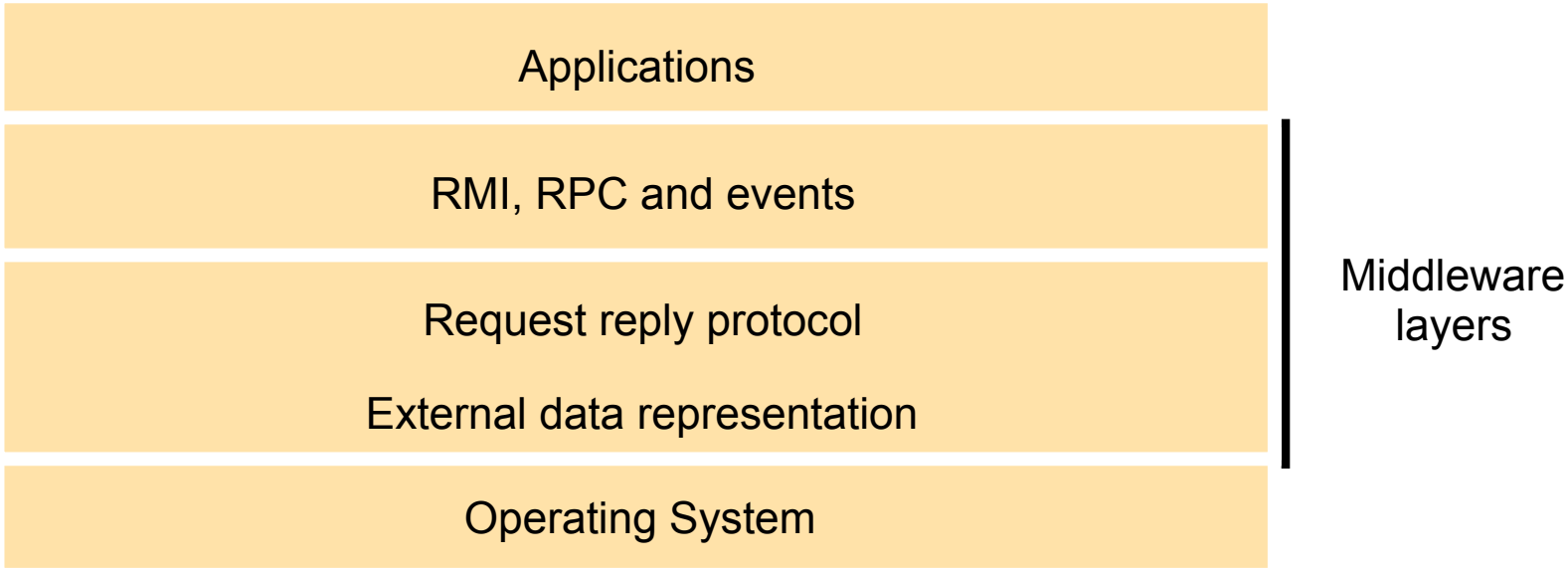
Vaidė Narváez

Computer Information Systems

June 28th, 2010

PAYAP UNIVERSITY

Middleware layer



1. Location transparency
2. Communication protocols
3. Computer hardware
4. Operating systems
5. Use of several programming languages

PAYAP UNIVERSITY

Middleware layer

- *"Middleware is the slash in the term client/server"*
- *"Middleware is software that allows elements of applications to interoperate across network links, despite differences in underlying communication protocols, system architectures, OSes, databases, and other application services"*
- *"Middleware succeeds by providing five main services: hardware independence, interchangeability of key software components (e.g., DBMSs), network independence, operational savings, and administrative savings"*
- *"Middleware is software that no one wants to pay for"*

PAYAP UNIVERSITY

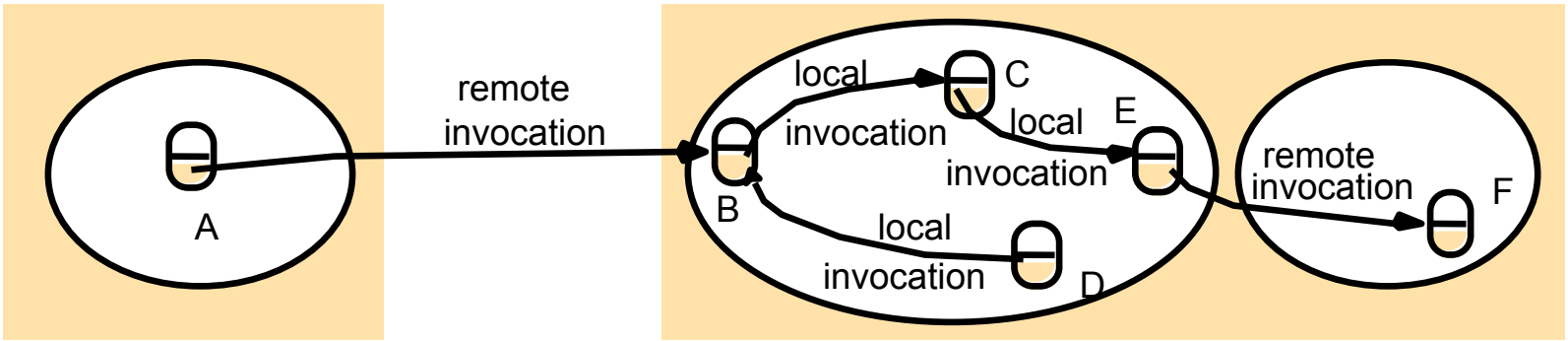
The object model

1. Object references
2. Interfaces
3. Actions
4. Exceptions
5. Garbage collection

PAYAP UNIVERSITY

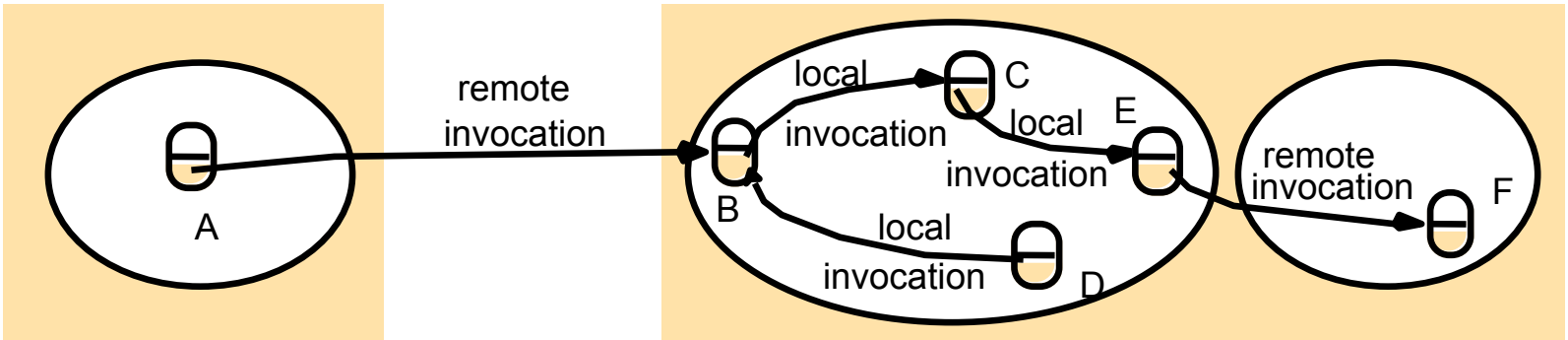
Distributed objects

- Local method invocation: within the same process
- Remote method invocation: different processes (different machines)



PAYAP UNIVERSITY

Distributed objects



- Remote object reference
- Remote interface

PAYAP UNIVERSITY

Distributed object model

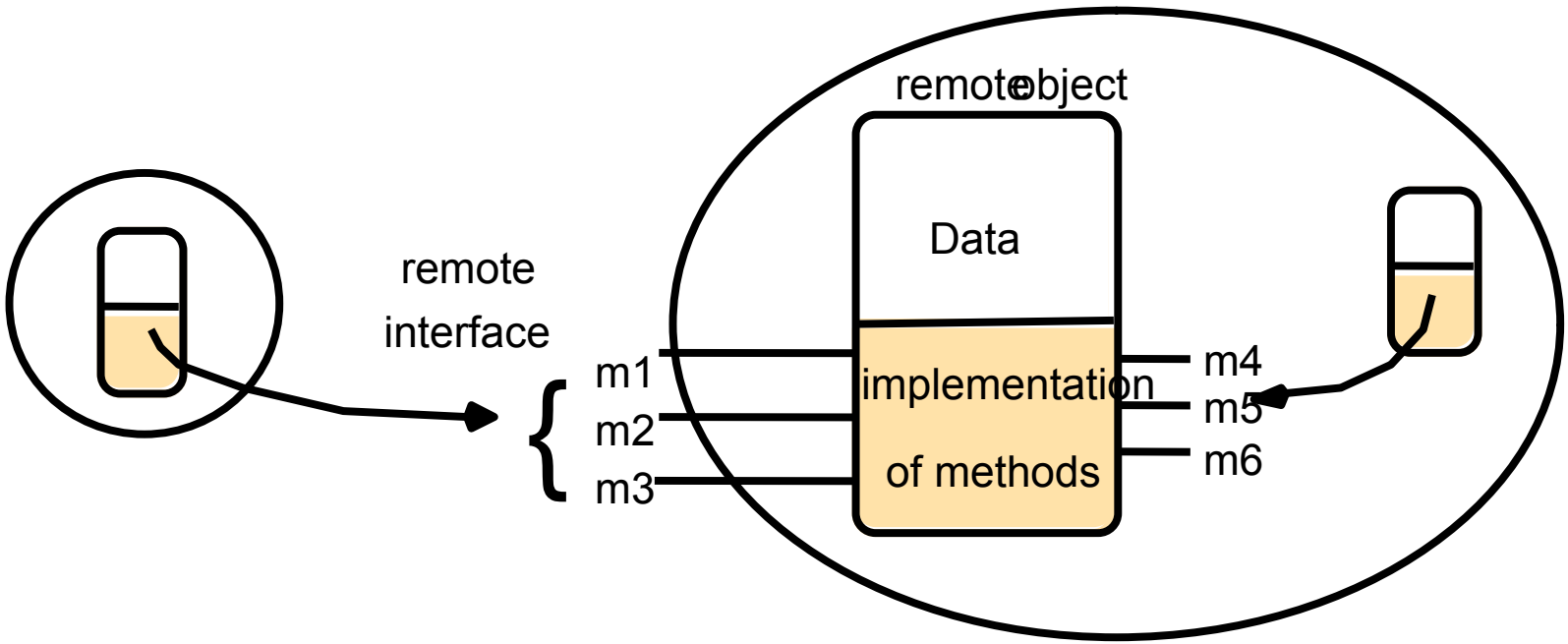
Remote Object reference

- other objects can invoke the methods of a remote object if they have access to its remote object reference
- it is an identifier throughout a distributed system
- can be passed as arguments and results of remote method invocations

Distributed object model

Remote interfaces

- every remote object has a remote interface that specifies which of its methods can be invoked remotely
- name, arguments, return type
- IDL



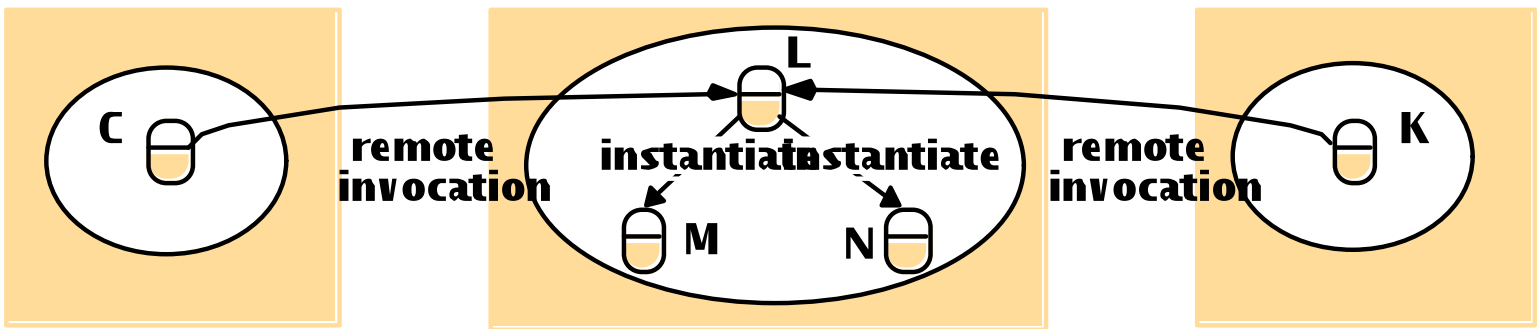
©Addison-Wesley 2005

Java RMI: A remote object must implement an interface that extends `java.rmi.Remote`

Distributed object model

Actions

- Needs remote object reference
- Calling of methods of objects in another process/host
- Remote objects might have methods for instantiation (hence remote instantiation)



PAYAP UNIVERSITY

Distributed object model

Garbage collection

- local garbage collector
- additional module to coordinate

Exceptions

- unexpected events or errors
- more and different exceptions from local methods

Java RMI: All methods of remote objects must throw
`java.rmi.RemoteException`

PAYAP UNIVERSITY

RMI design issues

Error handling

- Retry request message
- Duplicate filtering
- Retransmission of results

Invocation semantics

- Maybe
- At-least-once
- At-most-once

PAYAP UNIVERSITY

RMI: Invocation semantics

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*

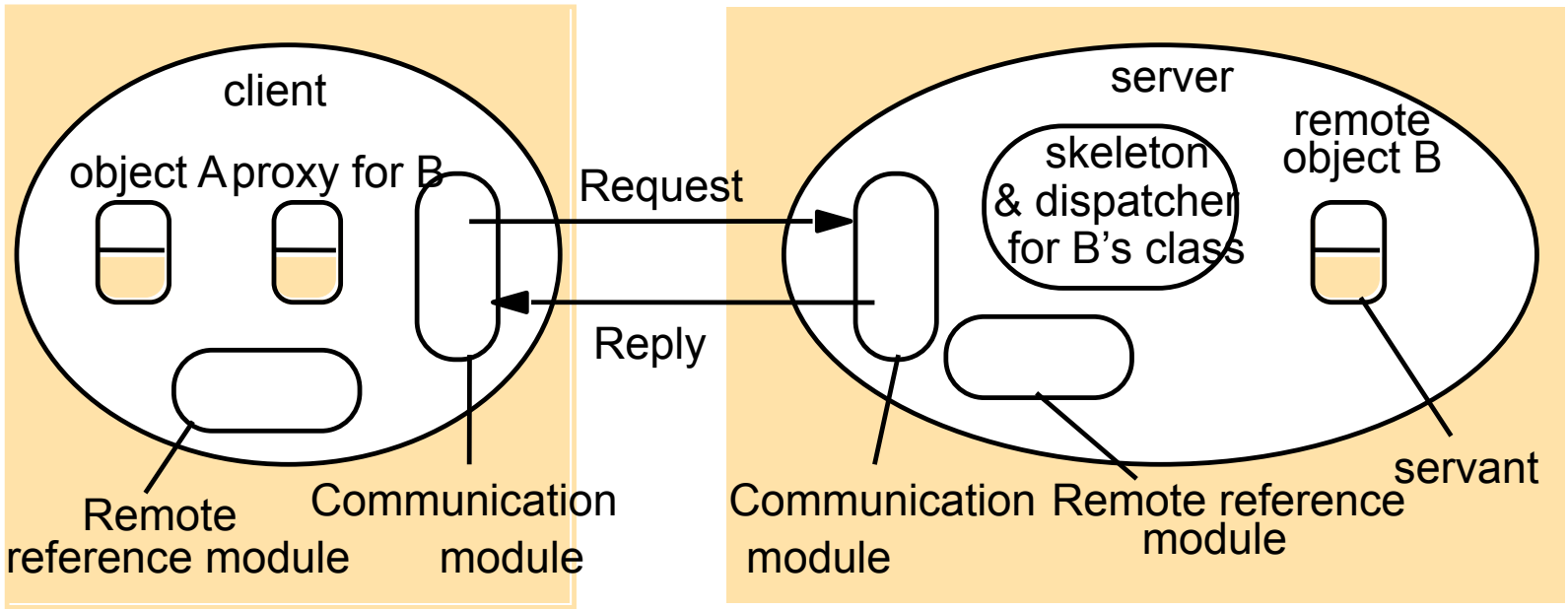
PAYAP UNIVERSITY

RMI: Transparency

- Like a local call:
  - marshalling/unmarshalling
  - locating remote objects
  - access/syntax
- Latency
- Vulnerability to failures

PAYAP UNIVERSITY

RMI Implementation



PAYAP UNIVERSITY

RMI Implementation

Communication module

- request-reply
- Client side: message type, requestID, remote object reference
- implements specific invocation semantics
- Server side: selects the dispatcher, passes on local reference from remote reference module

PAYAP UNIVERSITY

RMI Implementation

Remote reference module

- translate between local and remote object references
- create remote object references
- remote object table
  - remote objects held by the process (B on server)
  - local proxy (B on client)
- remote object (first time): add to the table, create proxy

PAYAP UNIVERSITY

RMI Implementation

RMI software

- Proxy: behaves like a local object, but represents the remote object
- Dispatcher: look at the methodID and call the corresponding method in the skeleton
- Skeleton: implements the method

Generation of proxies, dispatchers and skeletons

- IDL (RMI) compiler
  Java RMI: `rmic`

Dynamic invocation

- Proxies are static - interface compiled into client code
- Dynamic - interface available during runtime
  -Dynamic downloading of classes (Java RMI)

PAYAP UNIVERSITY

RMI Implementation

Server initialization

- Server creates the first object for remote access
- Usually clients are not allowed to create servers

Binder: locating service/object by name

- Table mapping for names and remote object references

Java RMI: `RMIregistry`

Server threads

- concurrency

PAYAP UNIVERSITY

RMI Implementation

Activation of remote objects

- many server objects, all running?
- active and passive status
  - active: available for invocation in a running process
  - passive: not running, state is stored on disk
- Activation
  - create an active object from a passive object
  - register the new active object
- Java RMI: objects can be activatable

PAYAP UNIVERSITY

RMI Implementation

Persistent object stores

- stored in marshalled from on disk for retrieval
- saved those that were modified
- persistent or not:
  - persistent root: any descendent objects (reachable from the root) are persistent
  -certain classes are declared persistent

PAYAP UNIVERSITY

RMI Implementation

Object location

- ip address, port #, ...

- location service for migratable objects
  - map remote object references to their probable current locations
  - Cache/broadcast scheme
    - Cache locations
    - If not in cache, broadcast to find it
  - Improvement: forwarding

PAYAP UNIVERSITY

RMI Implementation

Distributed garbage collection

- Reference count
- Java's approach
  - the server of an object (B) keeps track of proxies
  - addRef(B) is called when a proxy is created for a remote object
  - addRef(B) tells the server to add an entry
  - when the local host's garbage collector removes the proxy
  - removeRef(B) tells the server to remove the entry
  - when no entries for object B, the object on server is deallocated
- Race condition
  - removeRef(B) from client X
  - addRef(B) from client Y
- Communication failures
  - addRef() didn't return successfully
  - removeRef() will be issued
- Client process failures
  - leases from server
  - renew before expiration
  - entry removed if not renewed before expiration