Transaction Processing Systems

- Business Transaction
  - Interaction in real world
  - Usually between enterprise and person
  - Or maybe between enerprises
- Transaction Program
  - Performs function on (shared) database
- Online Transaction Processing System
  - Runs collection of transaction programs
- Our notion of "Information System"

The ACID Properties

- Atomicity
  - All (*commit*) or nothing (*abort*)
- Consistency
  - Map good states to good states
- Isolation
  - Concurrent transactions *serializable*
- Durability
  - Committed transactions are not lost

Difficult in centralized database

Especially difficult in distributed system

Atomicity

- All (commit) or nothing (abort)
- Example: transfer money between two bank accounts
- Some actions ("launch the missile") are not recoverable
- Compensating transactions?

Consistency

- Each transaction takes valid states to valid states:
  - Satisfy integrity constraints
  - Sometimes the only notion of "valid" state is "a state that could have been produced by executing a sequence of transactions

Isolation

- Each transaction behaves as if it were executed in isolation at some instant in time
- AKA *serializability*
- Consistency + Isolation implies the data remains consistent even when multiple transaction programs execute concurrently

Durability

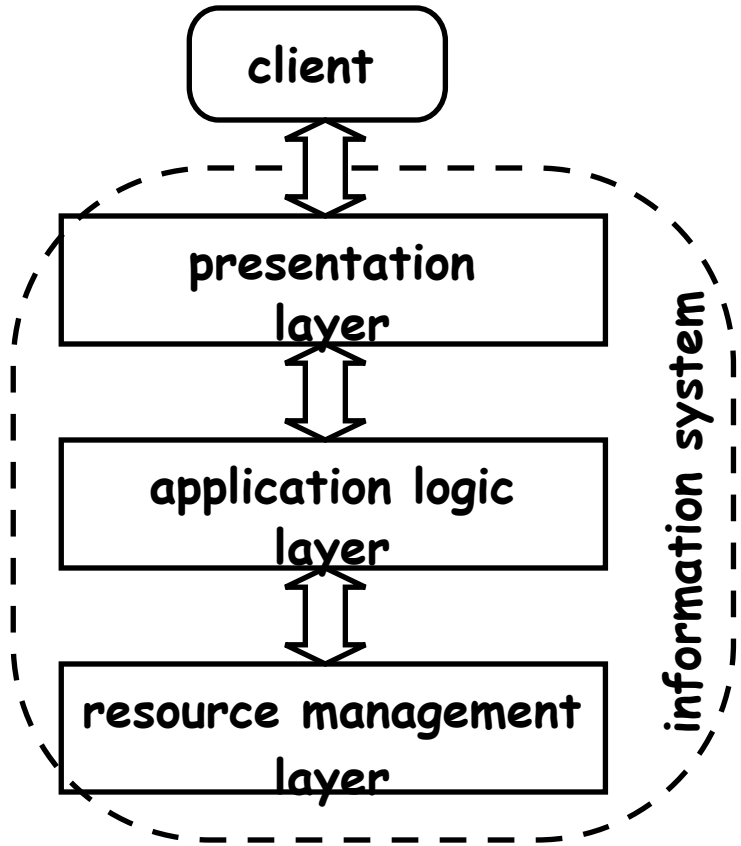
- The effect of a committed transaction will not be lost
- So data must be on stable storage before commit
- Usually done with a *log* (or *journal*) that must be forced before commit
- Crash recovery using the log

Resource Manager

- How ACID transactions are implemented
- Allocate resources to program executing a transaction
  - e.g. a locked record is a resource
- Reclaim resources in appropriate state on commit or abort

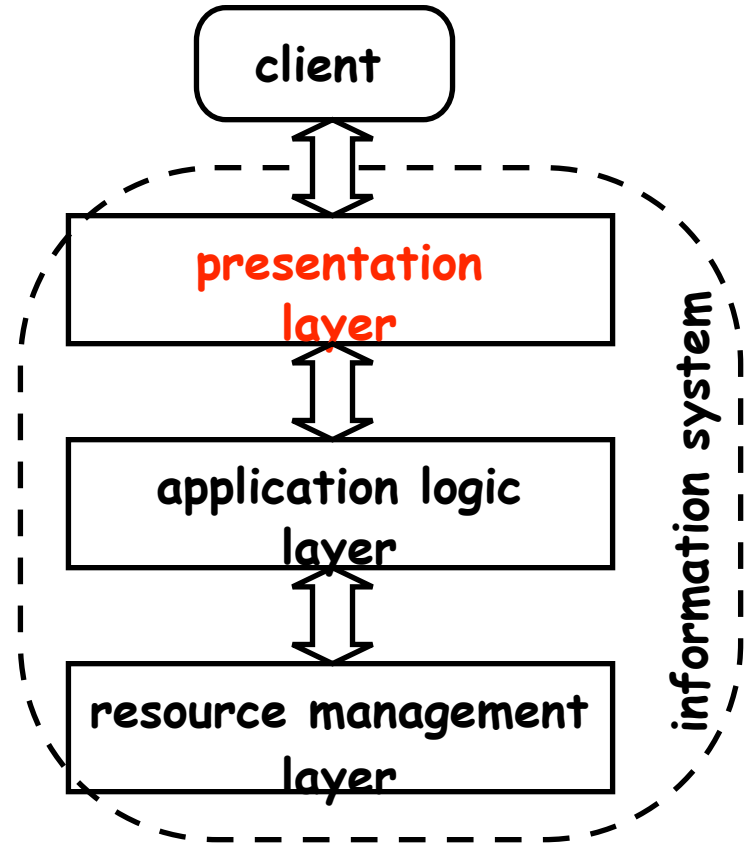
This is the meaning of "Resource Management Layer" in [ACKM04]

Three Layers of an Info System



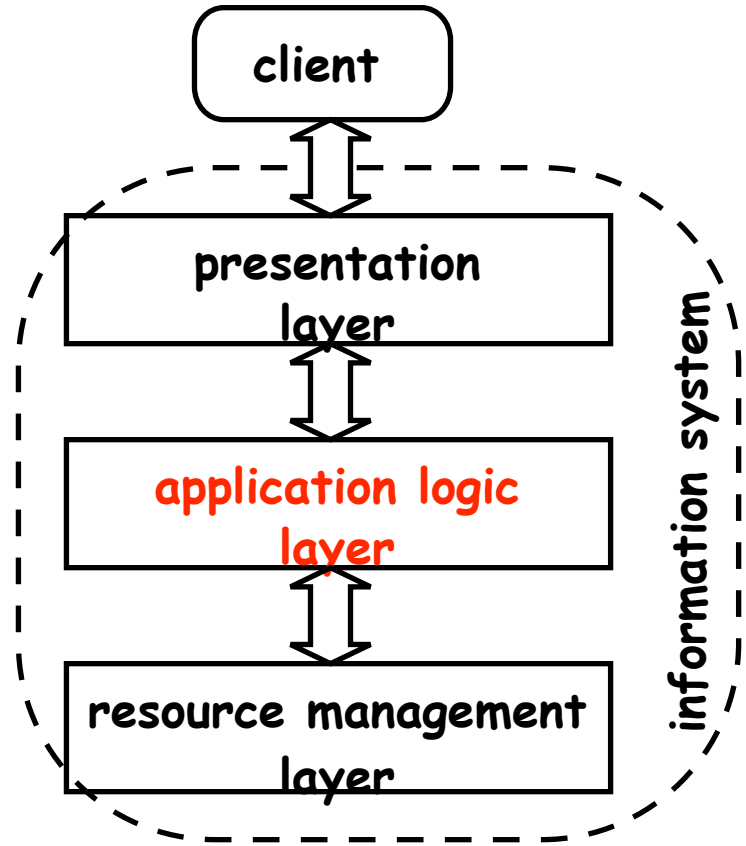
Presentation Layer

- Controls how the information system presents information to external entities and accepts it from them.
- External entities are users (UI) or other information systems (API)



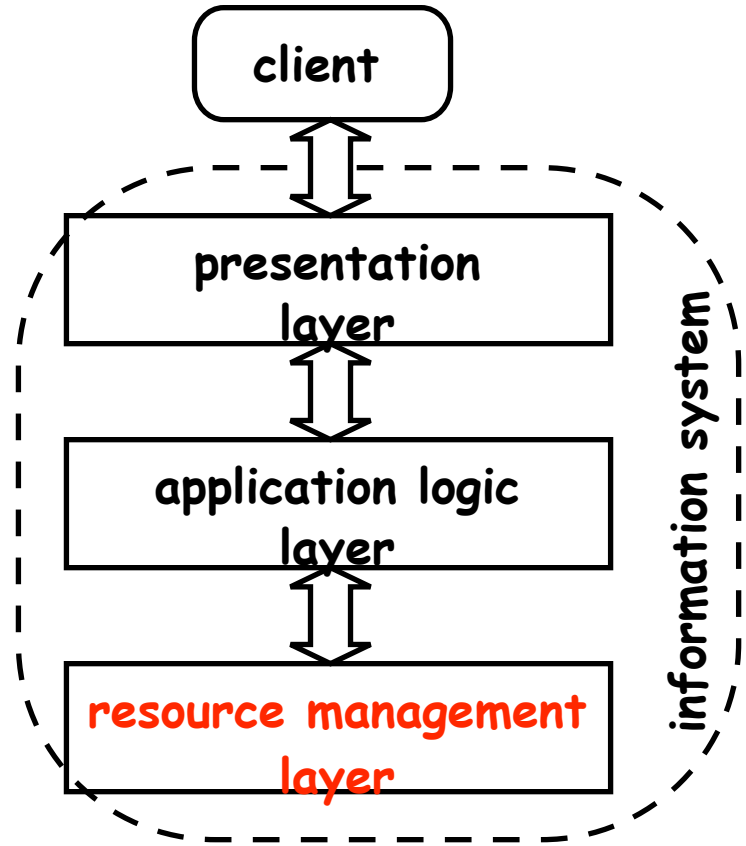
Application Logic Layer

- The program
- Business process
- Business logic
- Business rules

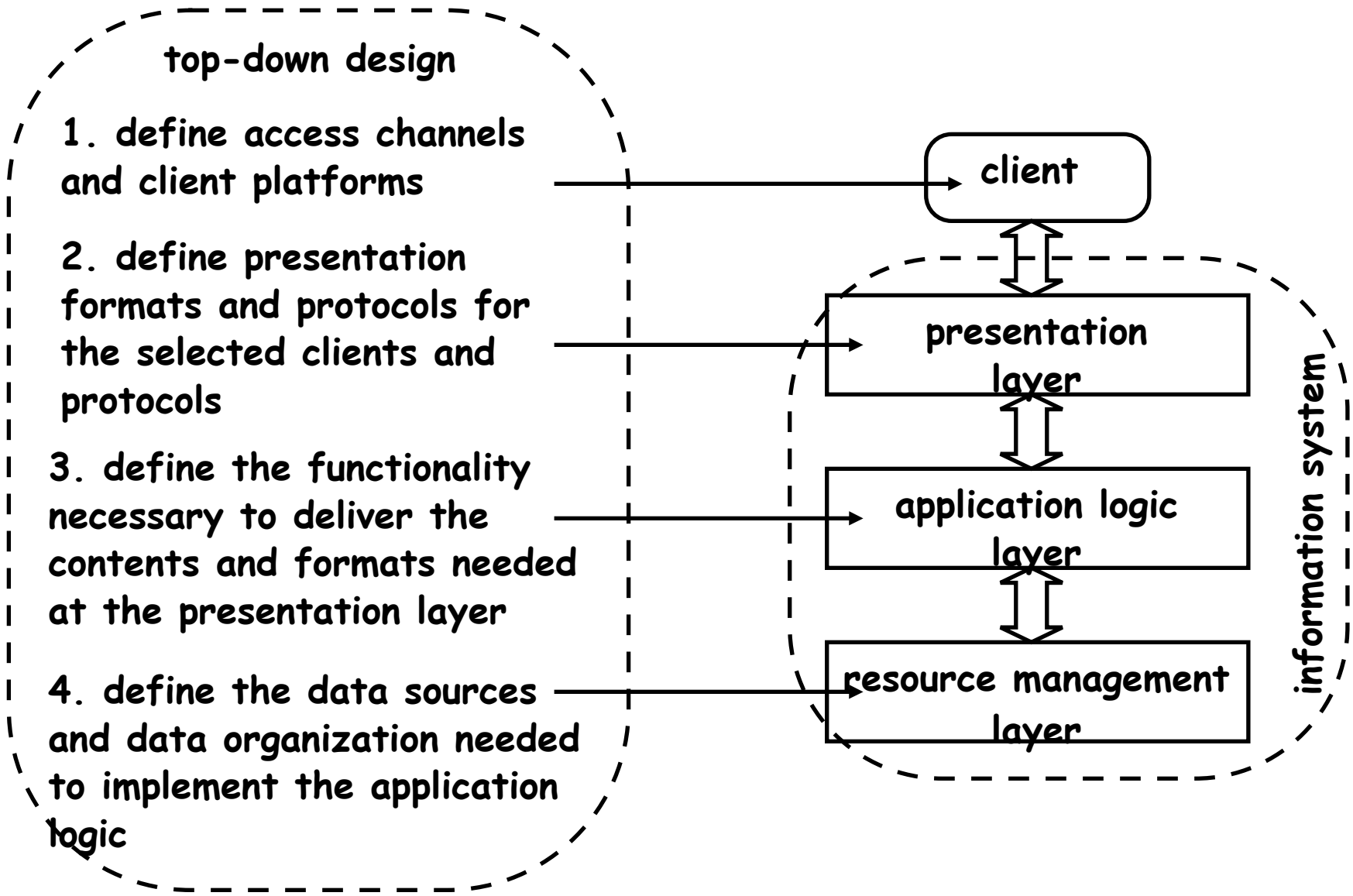


Resource Management Layer

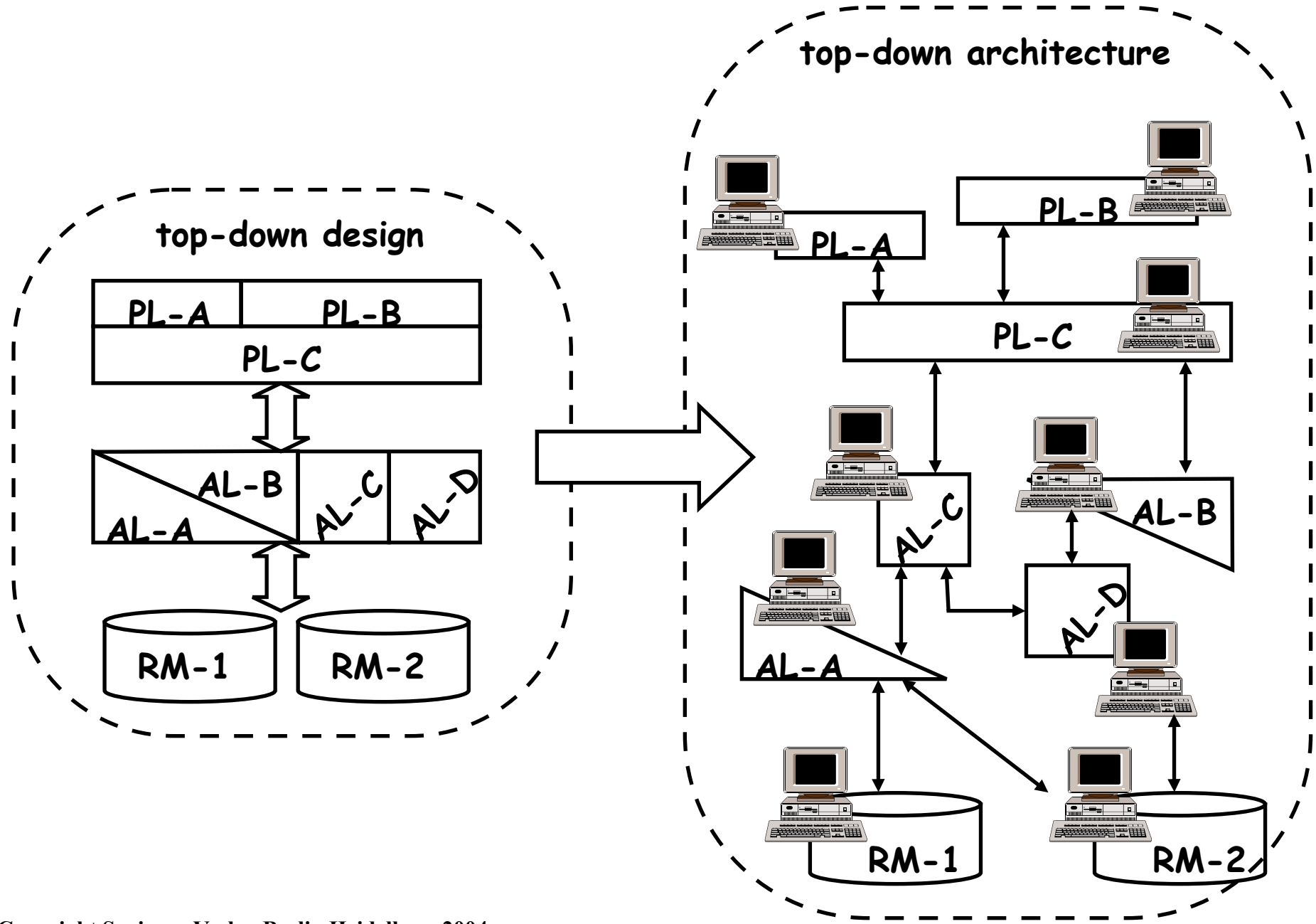
- The *data layer* as discussed above



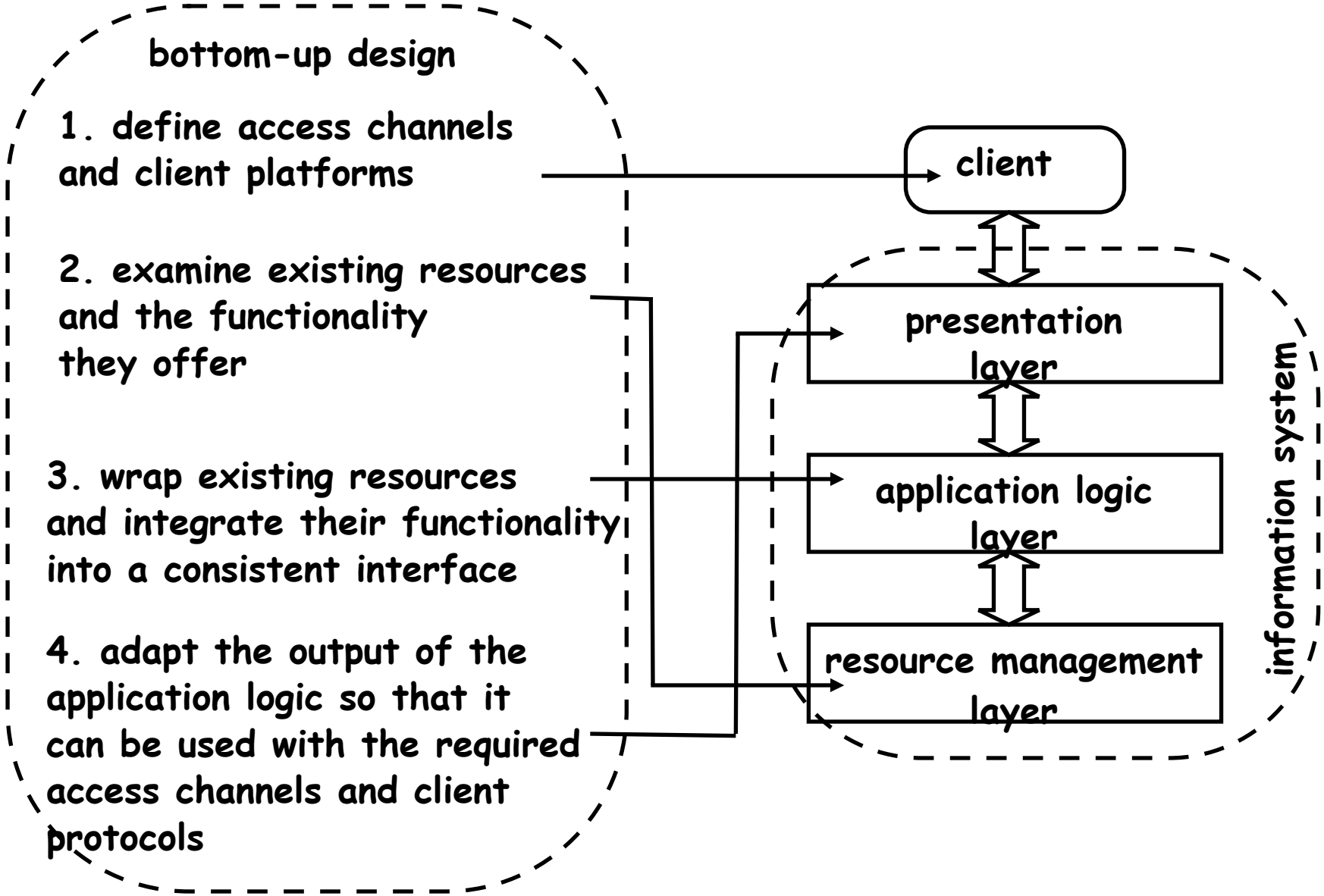
Top Down Design



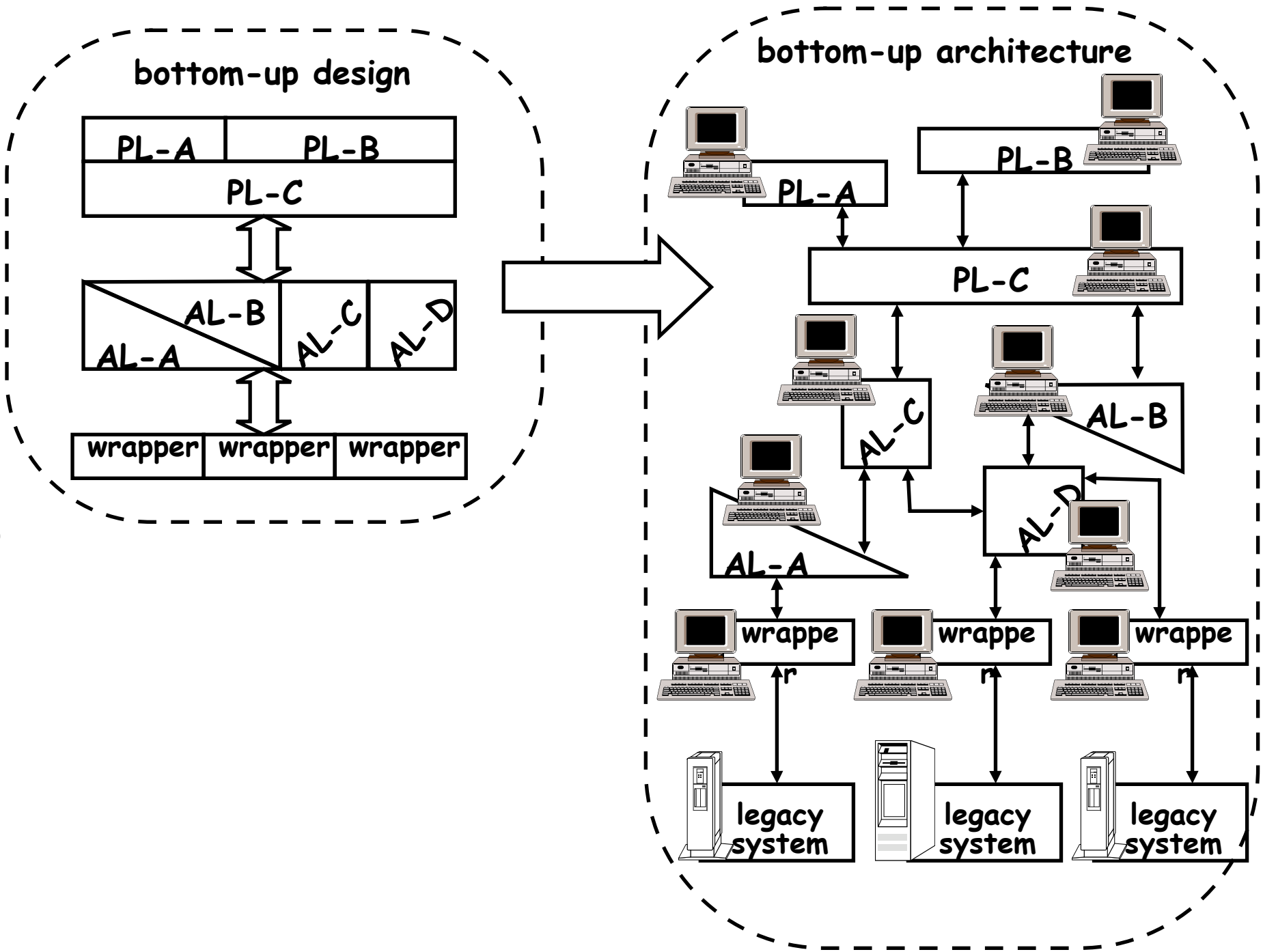
Top Down Architecture



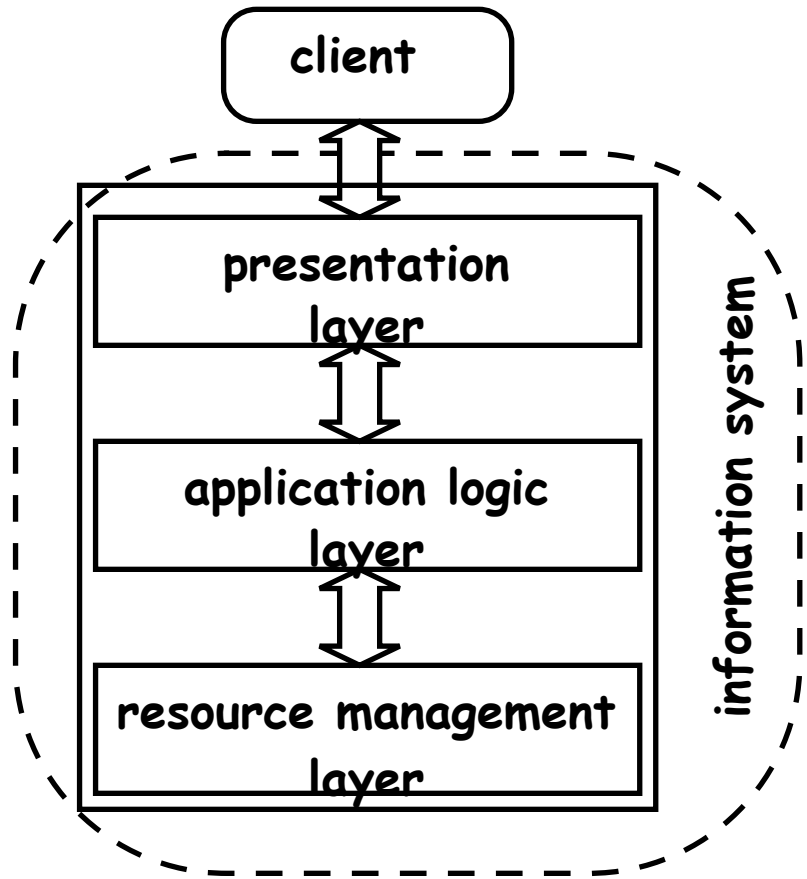
Bottom Up Design



Bottom Up Architecture



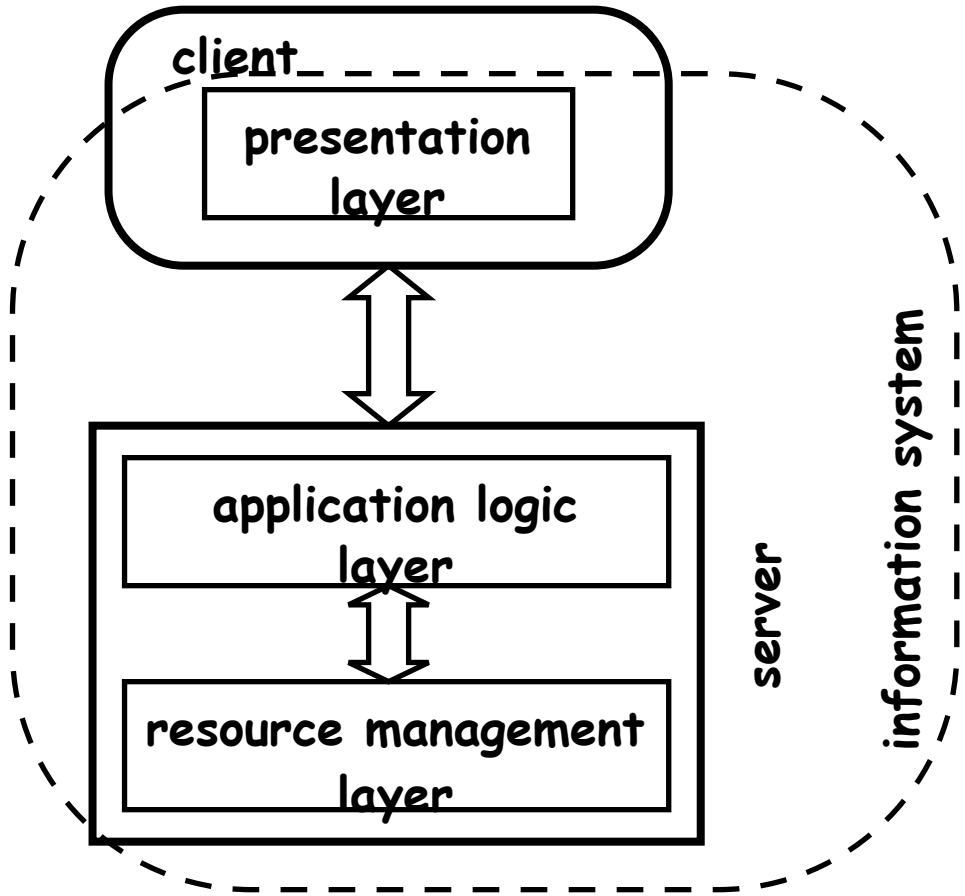
1-Tier Architecture



1-Tier - Remarks

- System is necessarily monolithic
- May be highly efficient
- No stable service interface API
  - That's what screen scrapers are for
- Problem of *Legacy Systems*

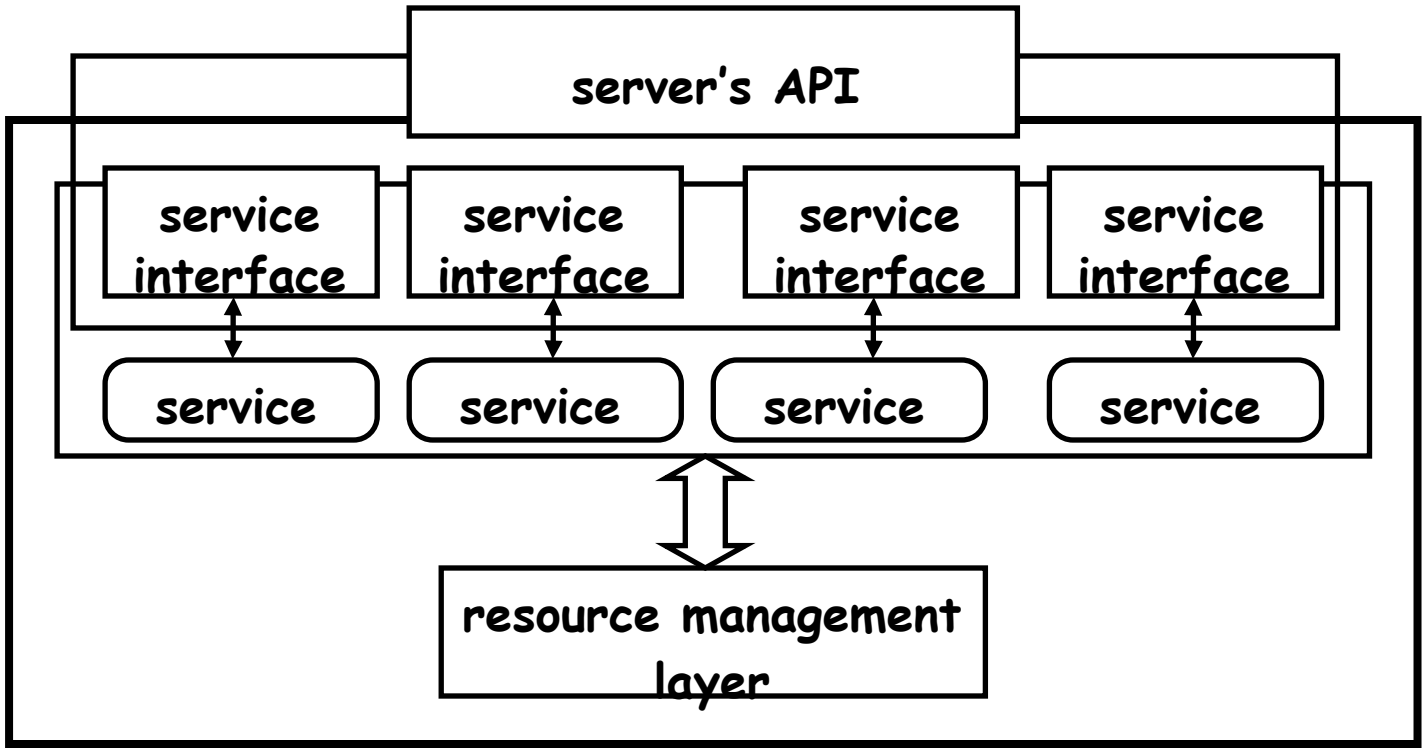
2-Tier Architecture



2-Tier - Advantages

- Added flexibility in presentation layer
  - e.g. multiple specialized presentation layers add no complexity to application
- Encouraged stable, published APIs
  - So clients could be developed

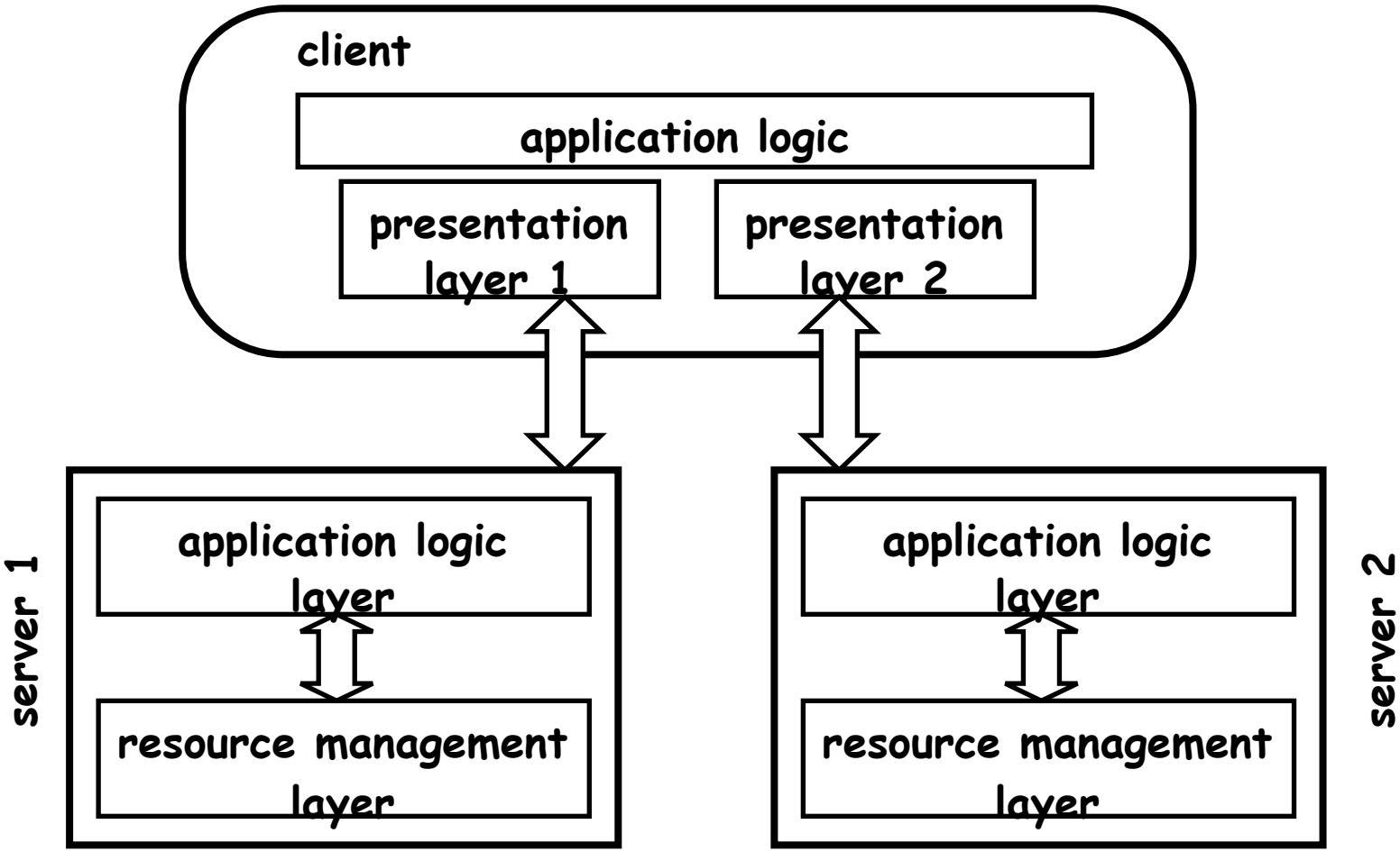
Server Organization: 2-Tier



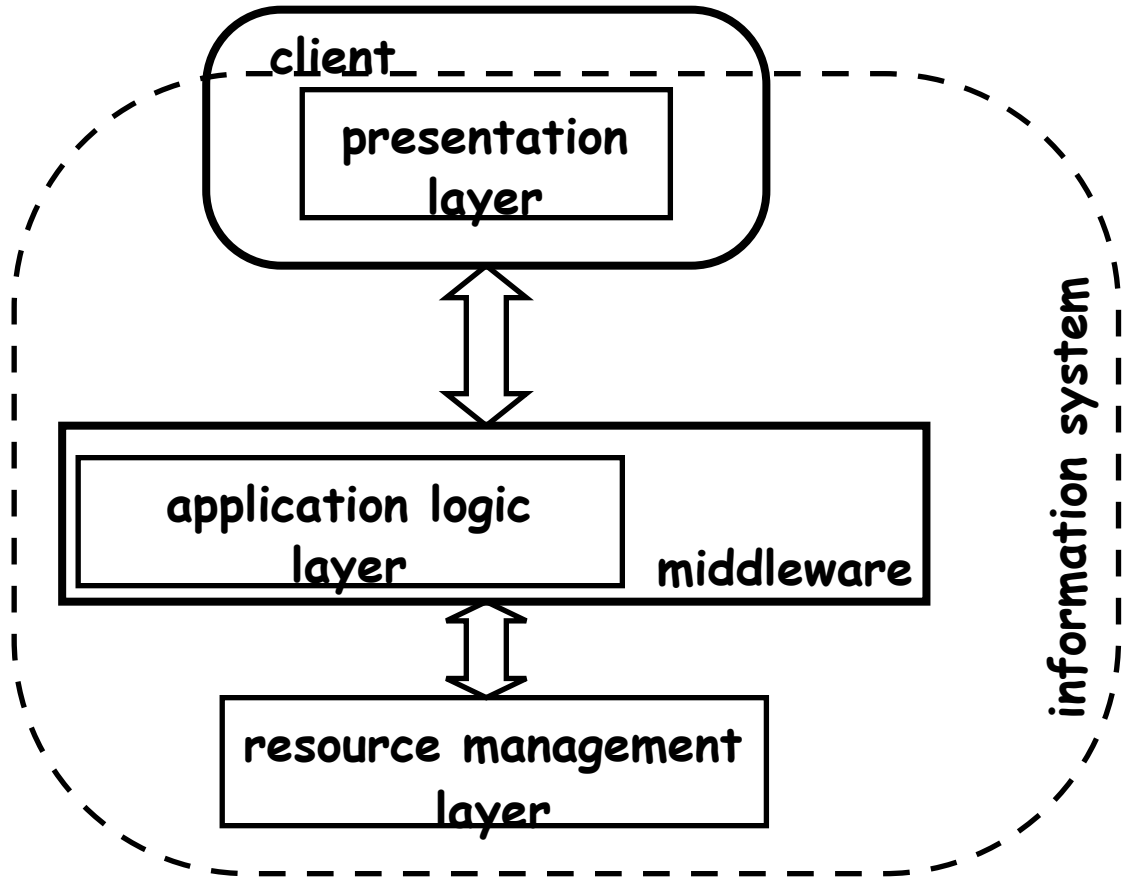
2-Tier - Disadvantages

- A single server doesn't scale
- Integration of multiple services must be done at client

Integration by Client



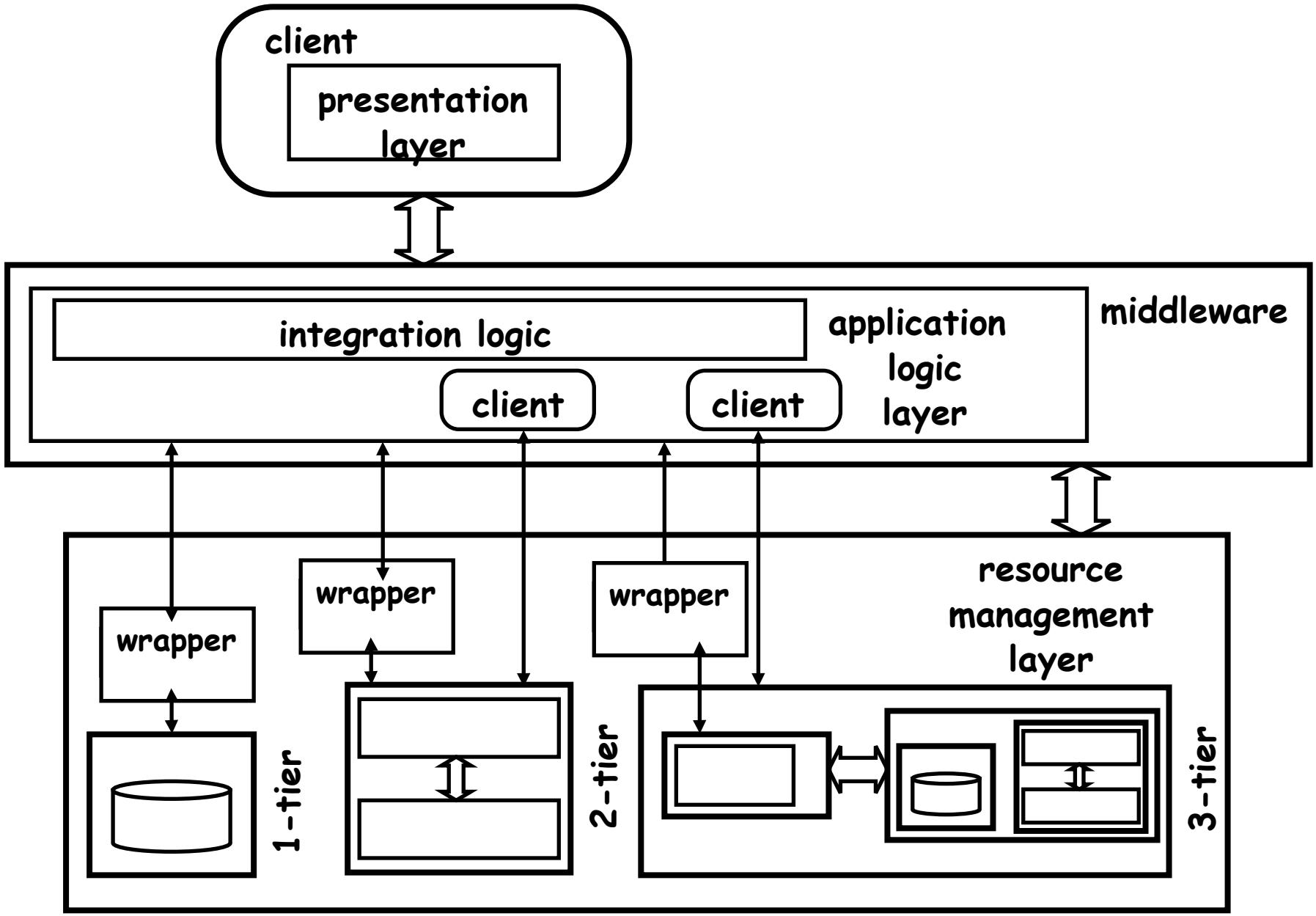
3-Tier Architecture



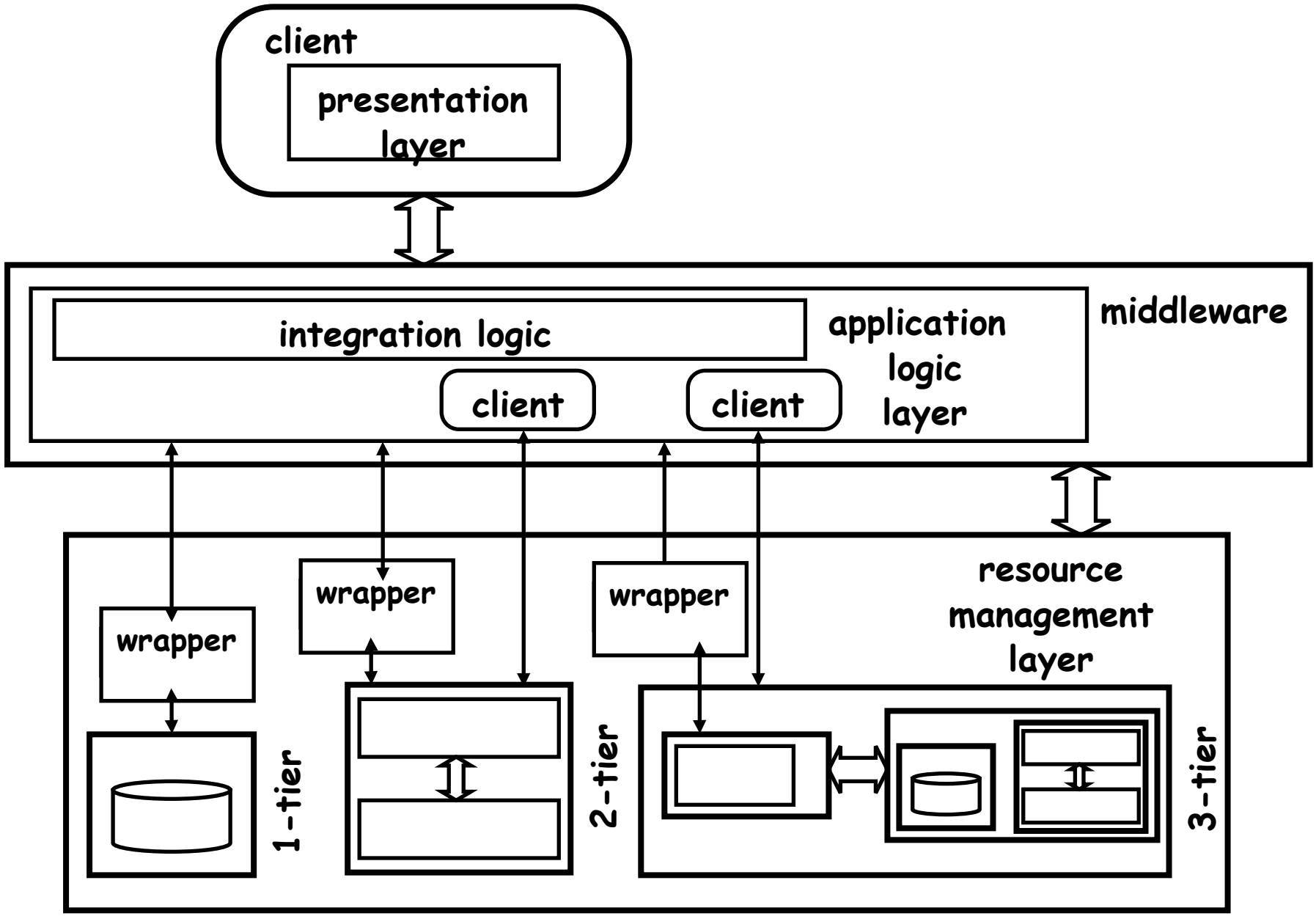
3-Tier - Advantages

- Scalability at application layer
  - Multiple application servers
- Application Integration
  - Do it in the middle tier
- Encourage stable, published APIs for resource management layer

App Integration in Middle Tier

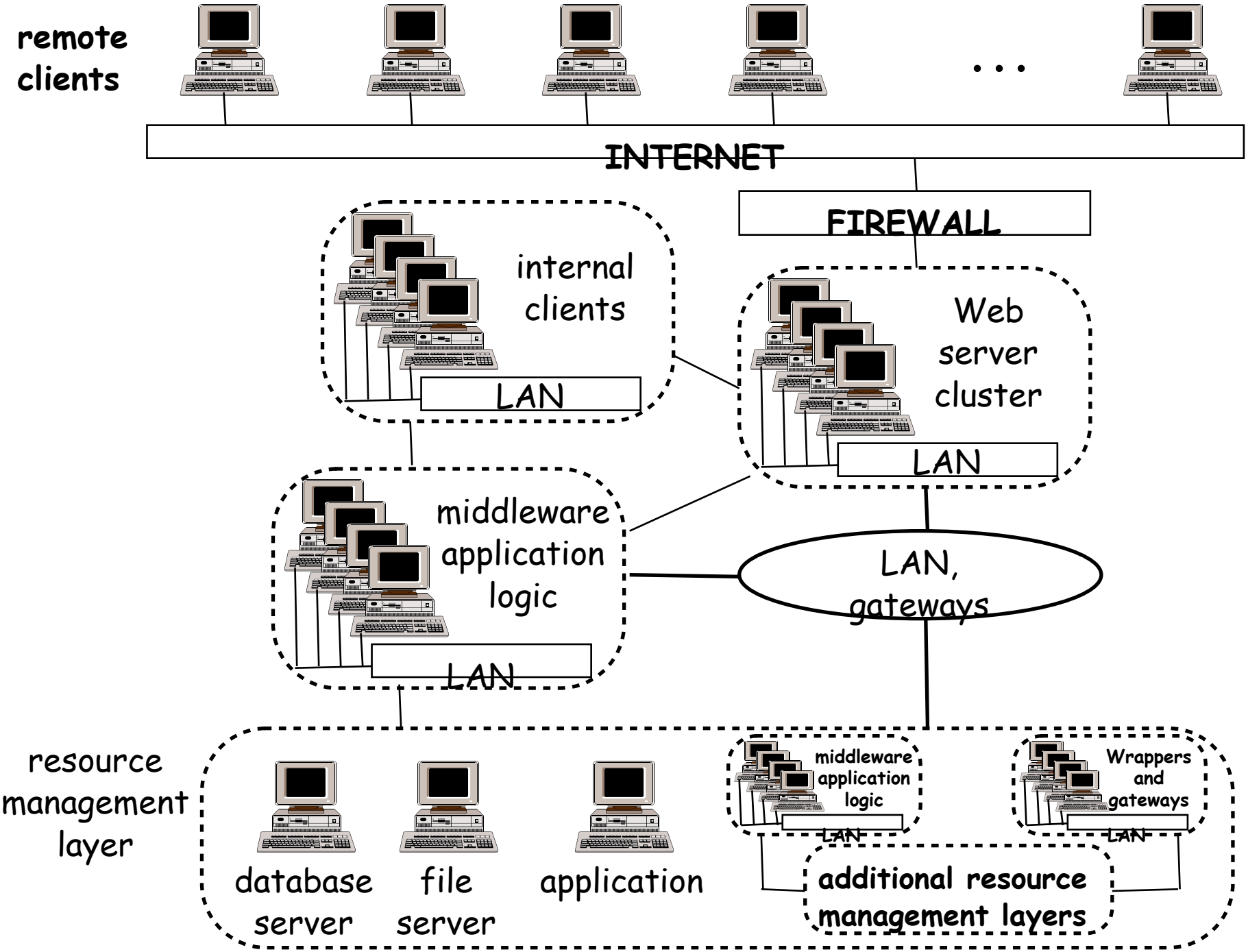


Inductively, N-Tier Architecture



3-Tier 4-Tier 5-Tier

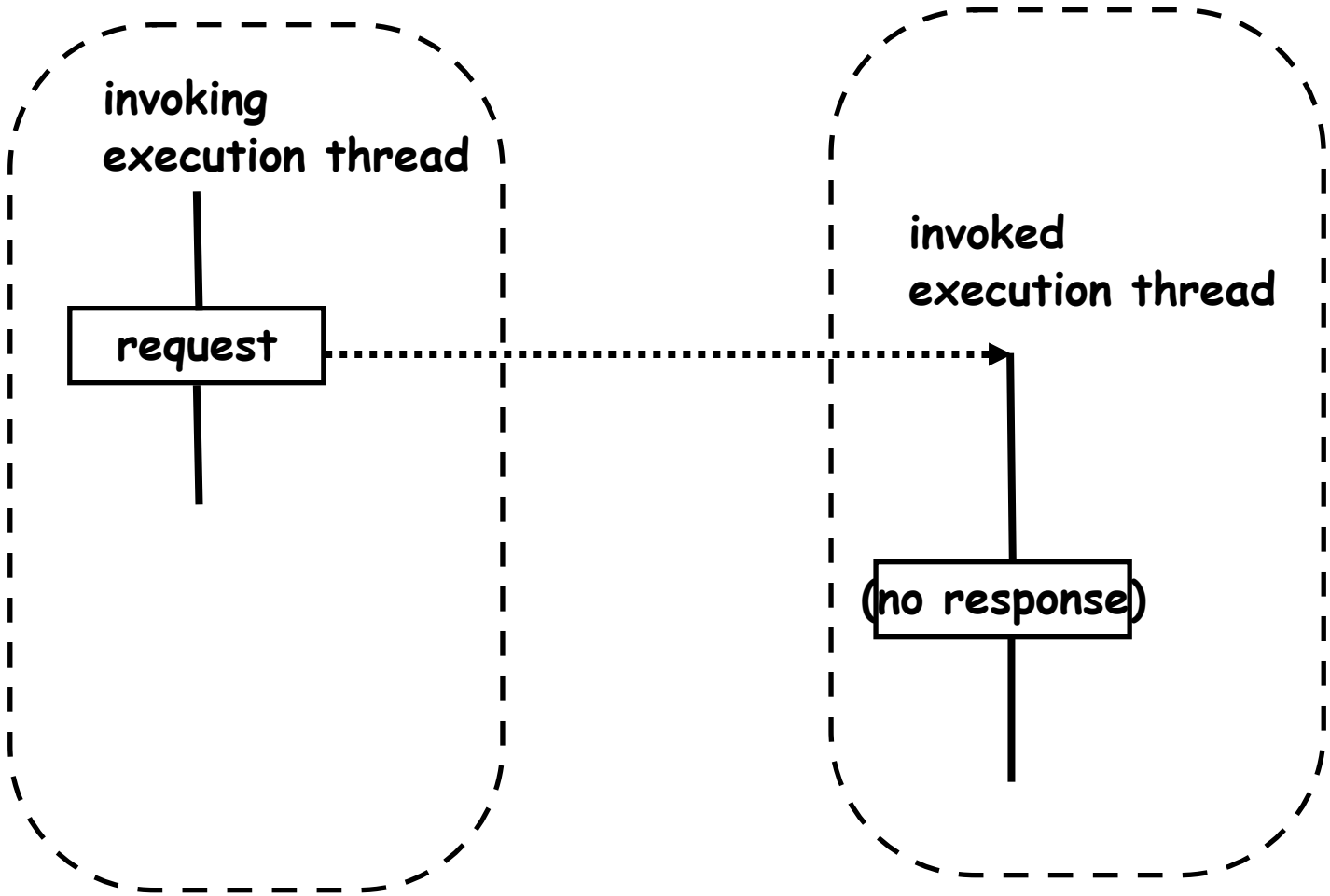
N-Tier in the Enterprise



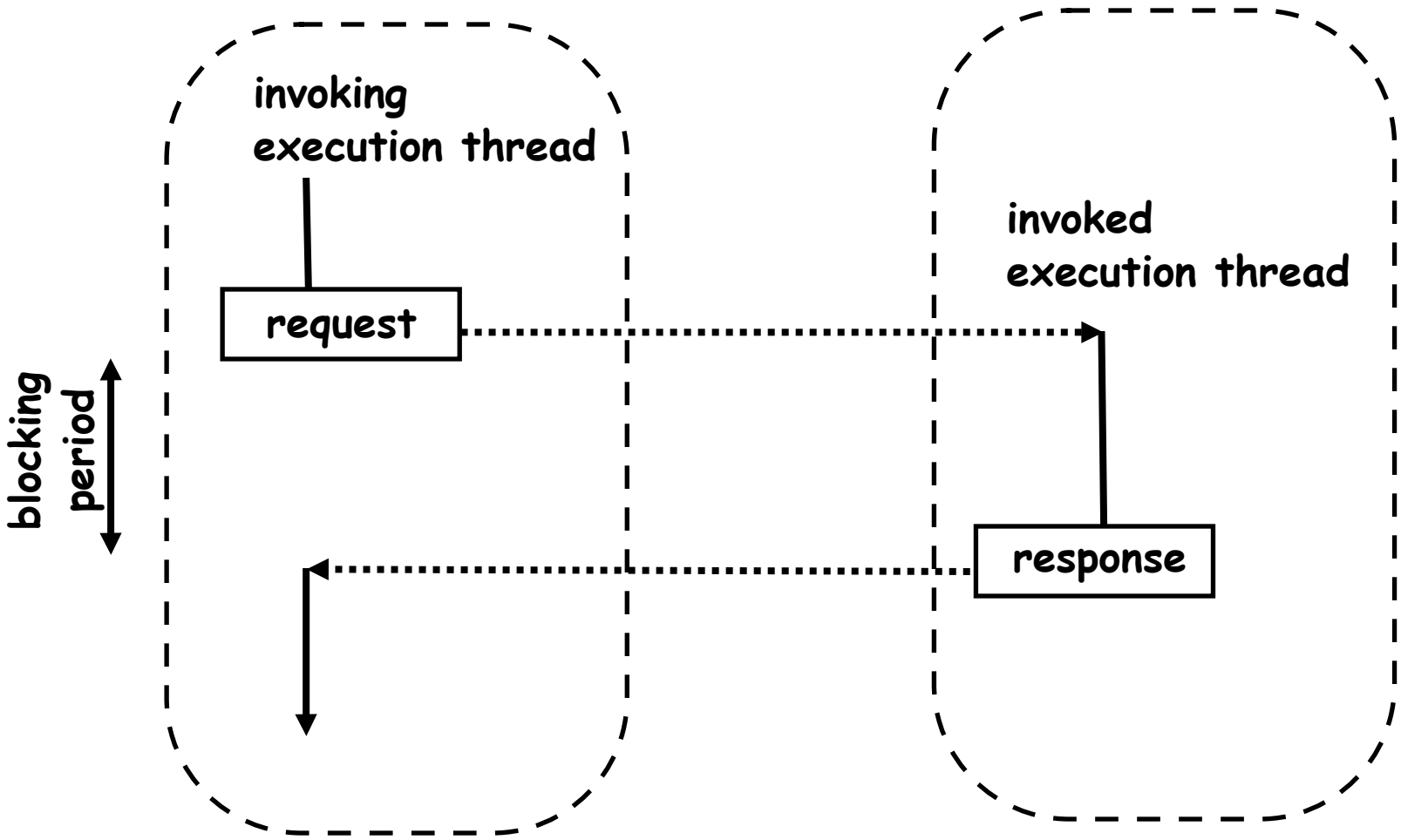
Communication

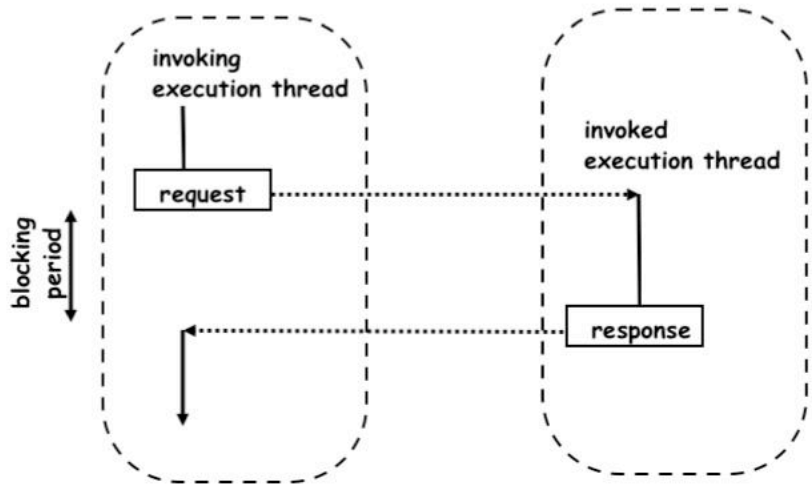
- 1-way messaging
- synchronous RPC
- asynchronous RPC

1-Way Messaging



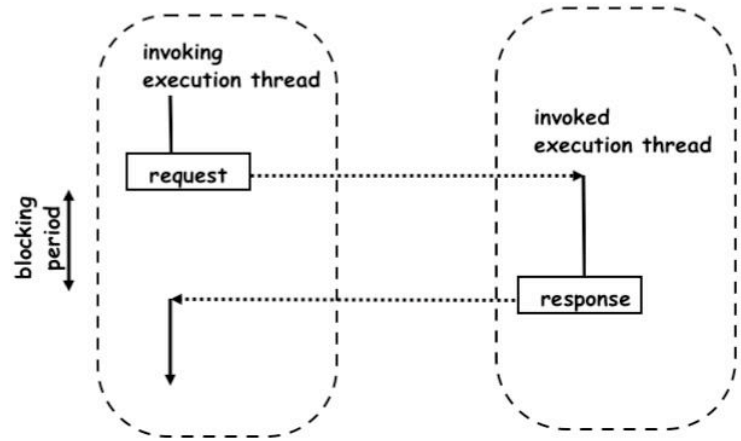
Synchronous RPC



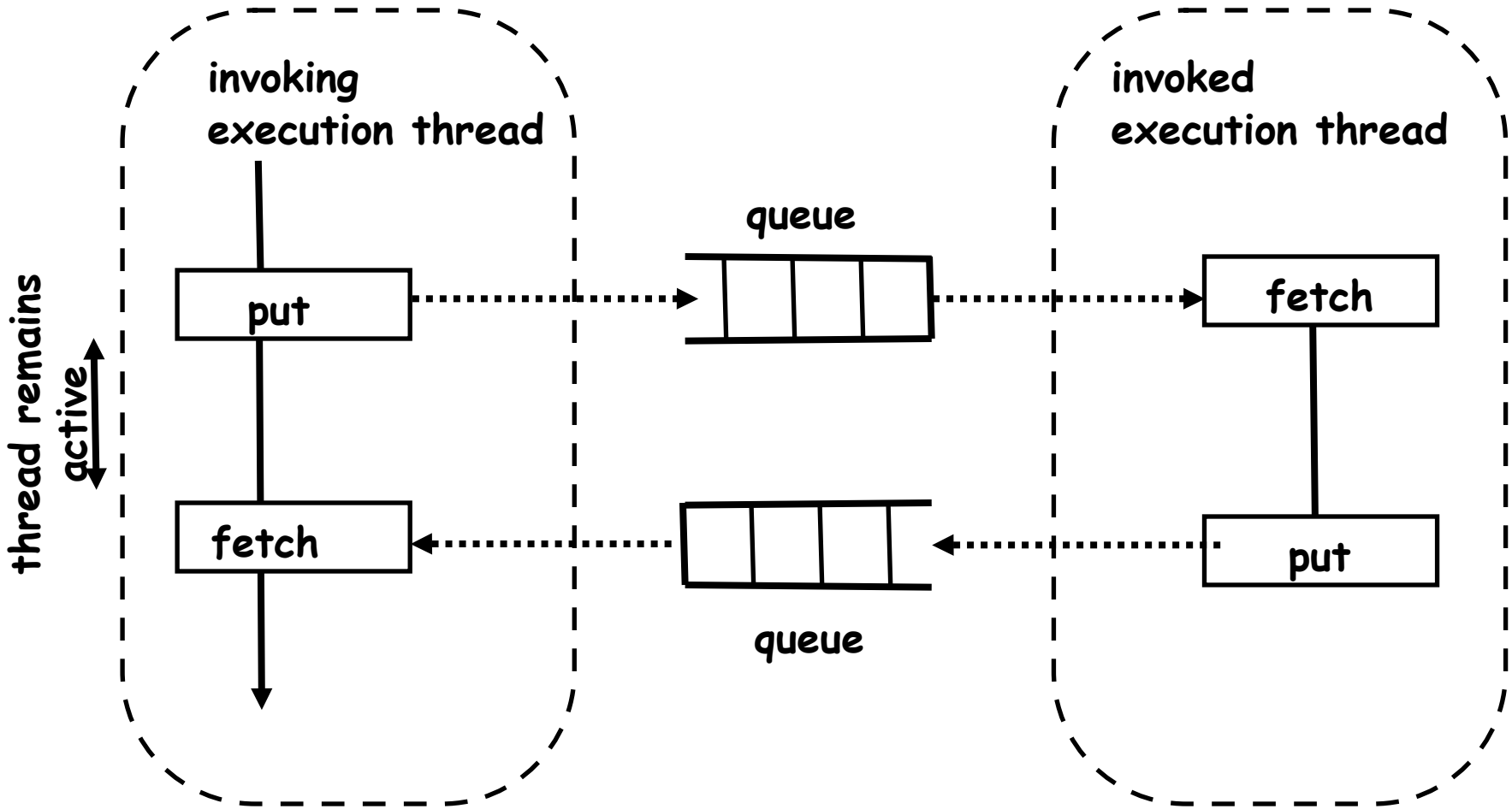

invoking
execution thread
request
blocking
period
invoked
execution thread
response

Synchronous RPC Issues

- Forcing caller to wait may reduce parallelism and waste resources
- Connection management issues
- Round-trip time issues

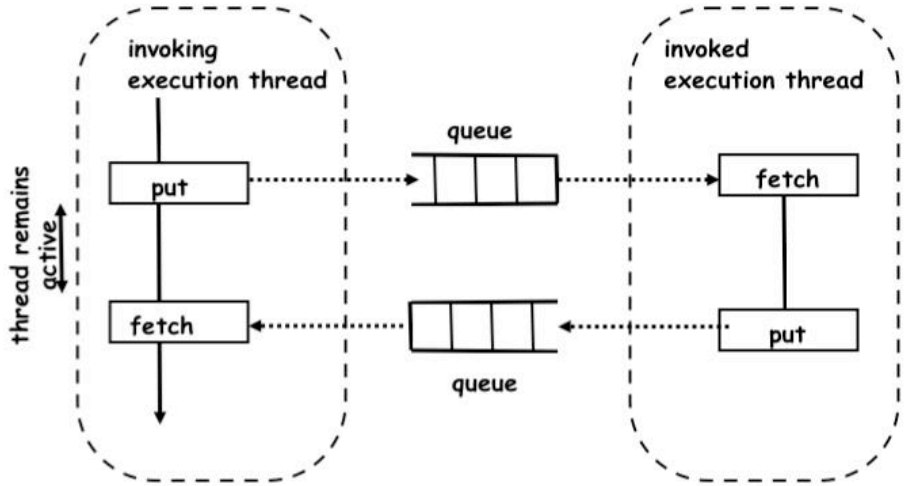


Asynchronous RPC



Asynchronous RPC

- Tolerates application (but not queue) failures
- Advantages for application integration
- Advantages for connection management
- Message-Oriented Middleware (MOM) and Message Brokers



Types of Middleware

- RPC-based systems
- TP Monitors
- Object Brokers
- Object Monitors
- Message-Oriented Middleware (MOM)
- Message Brokers

RPC Systems

- The foundation for most of the others
- We'll cover in some depth
- Requires support infrastructure
  - Interace Definition Language (IDL) compilers, stub generators
  - directory services for binding
  - etc.

TP Monitors

- Most established form of middleware
- TP-Lite
  - A 2-Tier architecture
  - Allow application logic as stored procedures in a database
- TP-Heavy
  - 3-Tier architecture
  - Implements *transactional RPC*
  - Coordinator for distributed transactions

Objects

- Object Broker (OMG CORBA)
  - Like an object-oriented RPC system
- Object Monitor
  - Like an object-oriented TP Monitor

Queues

- Message-Oriented Middleware
  - Queues (and transactional queues) to support asynchronous messaging
- Message Broker
  - All of the above
  - Ability to run application logic for message routing