Distributed Systems

Architectural Styles of Distributed Systems

WEEK 2: Layered Architecture, Middleware, Microservices, Publish-Subscribe, SOAP Architecture.

Online Lecture Series - 2

Felix Edesa

Addis Ababa Science and Technology University

AASTU

Contents Outline

- **Architectural Styles**
  - Layered Architectures
  - Service-Oriented Approaches (SOA, SOAP, Microservices)
  - Publish–Subscribe Model
- **Middleware in Distributed Systems**
  - Design and Organization
  - Adaptability and Modifiability
- **System Architecture Patterns**
  - Client–Server and Multitiered Models
  - Case Studies: Network File System (NFS), World Wide Web

AASTU

Architectural Styles

What is an Architectural Style?

- Defines the **logical organization** of a distributed system into software components.
- Specifies:
  - Components
  - Connections between components
  - Data exchanged
  - Overall system configuration
- Proper architecture is crucial for developing large systems successfully.

AASTU

Components in Distributed Systems

Definition

A **component** is a modular unit with well-defined **interfaces** that is **replaceable** within its environment.

- Replacement can occur while the system is running.
- Updates may involve:
  - Regular server updates
  - Switching to refreshed components after installation
  - Using replicated standbys during partial restarts
- **Interfaces must remain unchanged** to ensure smooth replacement.

Connectors in Distributed Systems

Definition

A **connector** mediates **communication, coordination, or cooperation** among components.

- Examples:
  - Remote Procedure Calls (RPC)
  - Message passing
  - Streaming data
- Enables the **flow of control and data** between components.

AASTU

Layered Architectures

Core Idea

Components are organized into layers.

- A component at layer L_j can make a **downcall** to a lower layer L_i $(i < j)$.
- Responses are typically returned upward.
- Only in special cases will an **upcall** be made to a higher layer.

AASTU

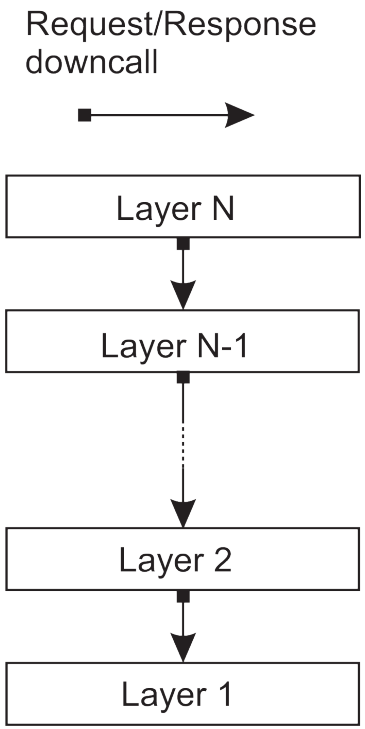
Why Layered Architectures Matter

Why It Matters

- Promotes modularity and separation of concerns.
- Simplifies maintenance and replacement of components.
- Serves as the foundation for many distributed systems.

AASTU

Pure Layered Organization



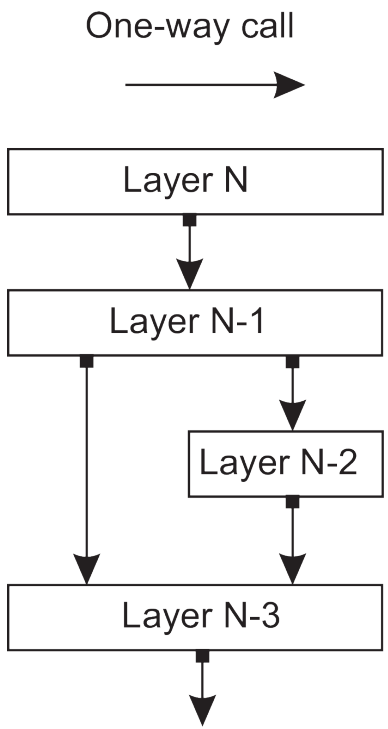
M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Description

- Components are strictly stacked in layers.
- Only **downcalls** to the next lower layer are allowed.
- Commonly used in **network communication stacks** (e.g., TCP/IP).

AASTU

Mixed Layered Organization



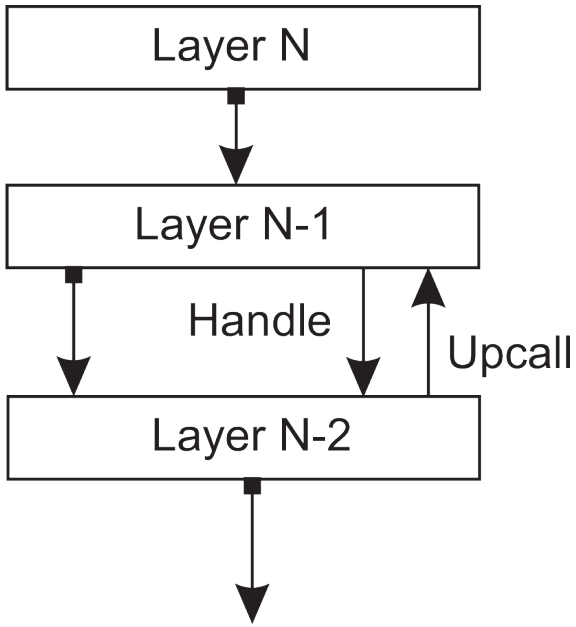
M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Description

- A higher-level component may call multiple lower layers.
- Example: an application calls both **OS libraries** and **math libraries**.
- Adds flexibility but slightly weakens strict modularity.

AASTU

Layered Organization with Upcalls



[1] M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Description

- Lower layers can make **upcalls** to higher layers.
- Used for event notification.
- Example: OS signaling an event (e.g., I/O completion) to an application.

AASTU

Interfaces

Interfaces

- **Interface:** The set of functions and constants exposed to the application.
  - `socket()`: Create a connection object
  - `accept()`: Wait for incoming connections
  - `connect()`: Connect to a specified server
  - `close()`: Close a connection
  - `send()`, `recv()`: Send and receive data
  - Constants `AF_INET`, `SOCK_STREAM` specify TCP

AASTU

Services

Services

- **Service:** What the interface provides to the application
  - Reliable, connection-oriented communication
  - Applications do not need to know TCP implementation details

AASTU

Protocols

Protocols

- **Protocol:** Rules followed to implement the service
  - TCP is the protocol used in this example
  - Ensures data integrity and ordering
  - Hidden from the application — apps just use the interface
- The combination of interface + protocol + service allows two parties to reliably communicate without knowing the internal TCP implementation.

AASTU

Application Layering

Logical Layers of Applications

- Large distributed applications often follow a layered architecture.
- Layering helps separate concerns and organize complex functionality.

AASTU

Three Logical Levels

Logical Levels

- **Application-interface level:** Interacts with users or external applications.
- **Processing level:** Core functionality of the application.
- **Data level:** Handles database or file system access.

AASTU

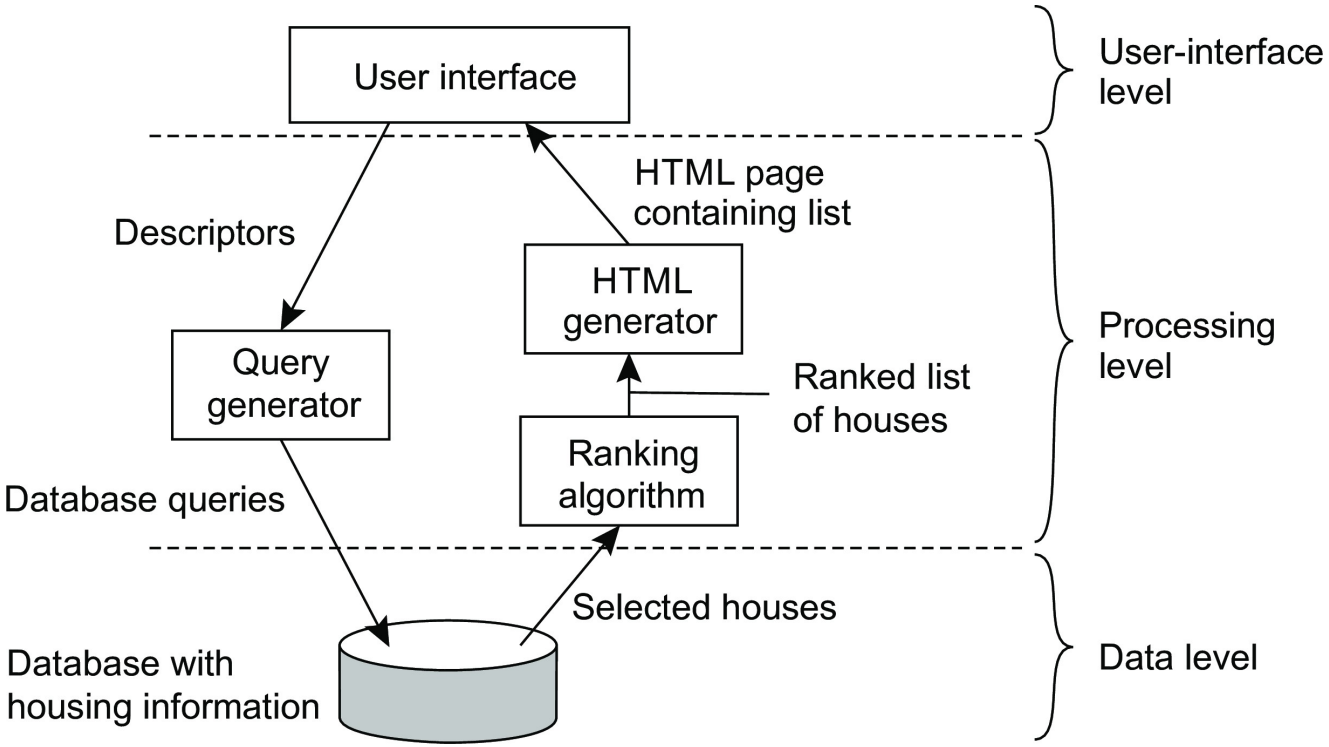
Example: Housing Search Engine

Processing and Data Layer

- **Processing layer:**
  - Converts user input into database queries
  - Retrieves and ranks results
  - Generates HTML output
- **Data layer:** Database of houses currently for sale
- Observation: Processing layer is the core for computational effort.

AASTU

Application Layering Diagram



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Service-Oriented Architectures

Motivation & Concept

- Layered architectures create strong dependencies between layers
- Example: Removing a small component can break many programs
- Service-oriented approach: Looser organization into independent entities, each encapsulating a service

AASTU

Service-Oriented Architectures

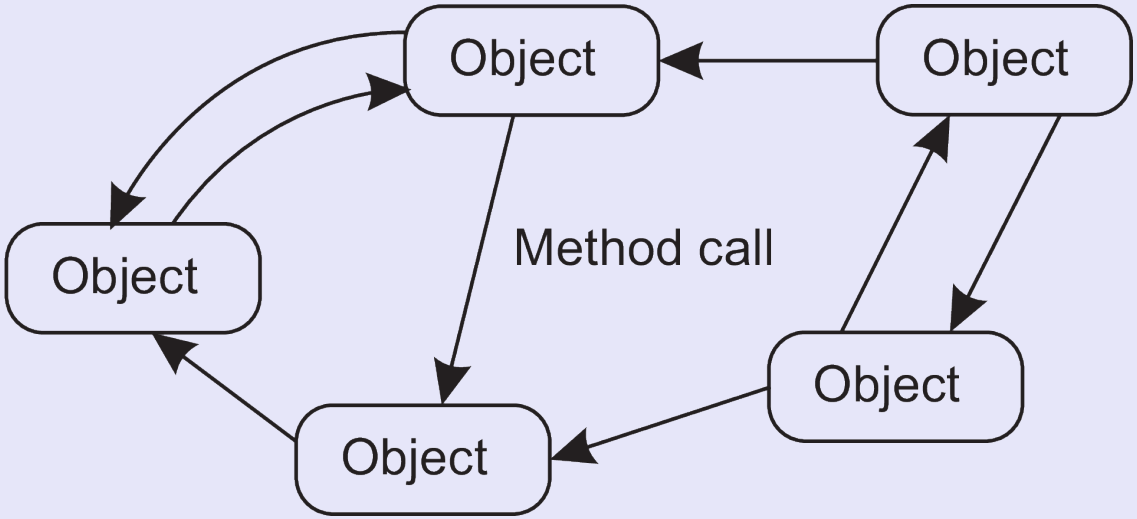
Entities & Modularity

- Each entity encapsulates a **service**
- Services, objects, or microservices run as separate processes or threads
- Running separately improves modularity, though dependencies may still exist

AASTU

Object-based architecture Style

Visual Representation



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Object-Based Architectural Style

Objects as Components

- Each object corresponds to a component
- Objects communicate via procedure calls, possibly over a network
- Caller does not need to know object location (can be local or remote)

AASTU

Object-Based Architectural Style

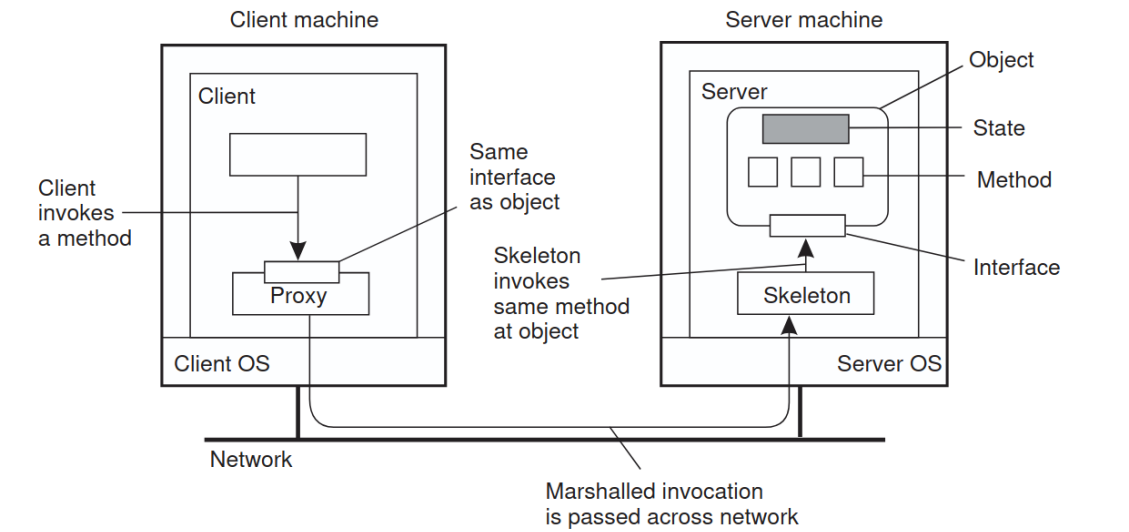
Encapsulation & Interfaces

- Objects encapsulate **state** and **methods**
- Object interfaces conceal implementation details
- Objects can be replaced if interfaces remain consistent

AASTU

Object-based architecture Style

Visual Representation



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

AASTU

Distributed Objects

Remote / Distributed Objects

- Object interface can reside on one machine, object itself on another
- Called a **distributed object** or **remote object**
- Client binds via a **proxy**
- Proxy packs method calls into messages and unpacks replies for the client

AASTU

Microservice Architectural Style

Introduction

- Microservices build on object-based architectures by encapsulating services into independent units.
- Each microservice is a self-contained entity, possibly using other services.
- Clear separation allows services to operate independently (foundation of SOA).

AASTU

Microservice Architectural Style

Characteristics

- Each microservice runs as a separate (network) process.
- Implementation can be a remote object but it's not required.
- No strict size definition, but modularization is key.
- Microservices can be deployed across edge/fog infrastructures.

AASTU

RESTful Architectures

Key Characteristics

- Resources identified through a single naming scheme.
- All services offer the same interface (max 4 operations).
- Messages are fully self-described.
- Services do not retain state after operation execution.

AASTU

RESTful Architectures

Stateless Execution

- RESTful services follow a **stateless execution** model.
- Each request is independent and contains all necessary information.
- No session state is retained on the server.

AASTU

RESTful Operations

Core Operations

- **PUT** – Modify a resource by transferring a new state
- **POST** – Create a new resource
- **GET** – Retrieve the state of a resource
- **DELETE** – Remove a resource

AASTU

RESTful Requests in S3

Examples

- **PUT**: Create bucket or upload object.
- **GET**: Retrieve list of objects in a bucket.
- Returns ordinary **HTTP responses**.
- Errors handled via standard HTTP error codes.

AASTU

RESTful: Strengths and Limitations

Comparison

- **Strength:** Simplicity, uniform interface, HTTP-based.
- **Limitation:** Difficult for complex communication (e.g., distributed transactions).
- Best fit: Simple integration schemes.

AASTU

RESTful Operations

Core Operations

Operation	Description
PUT	Modify a resource by transferring a new state
POST	Create a new resource
GET	Retrieve a resource's state
DELETE	Delete a resource

AASTU

Publish-Subscribe Architectures

Key Idea

- Loose coupling between processes
- Separation of **processing** and **coordination**
- Processes interact via **publish/subscribe** model

AASTU

Coordination in Distributed Systems

Two Dimensions of Coupling

- **Temporal coupling:** Processes must be active at same time
- **Referential coupling:** Processes explicitly reference each other

AASTU

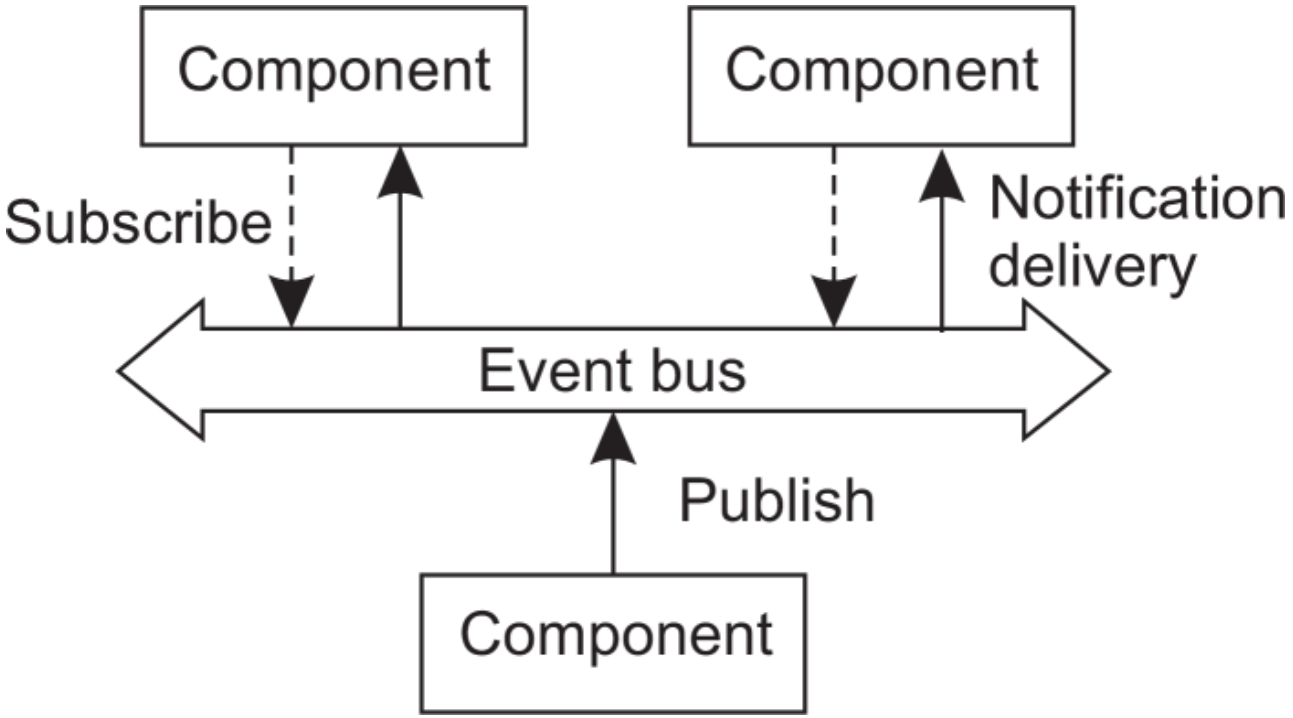
Forms of Coordination

Examples

- Direct coordination
- Mailbox coordination
- Event-based coordination
- Shared data space

AASTU

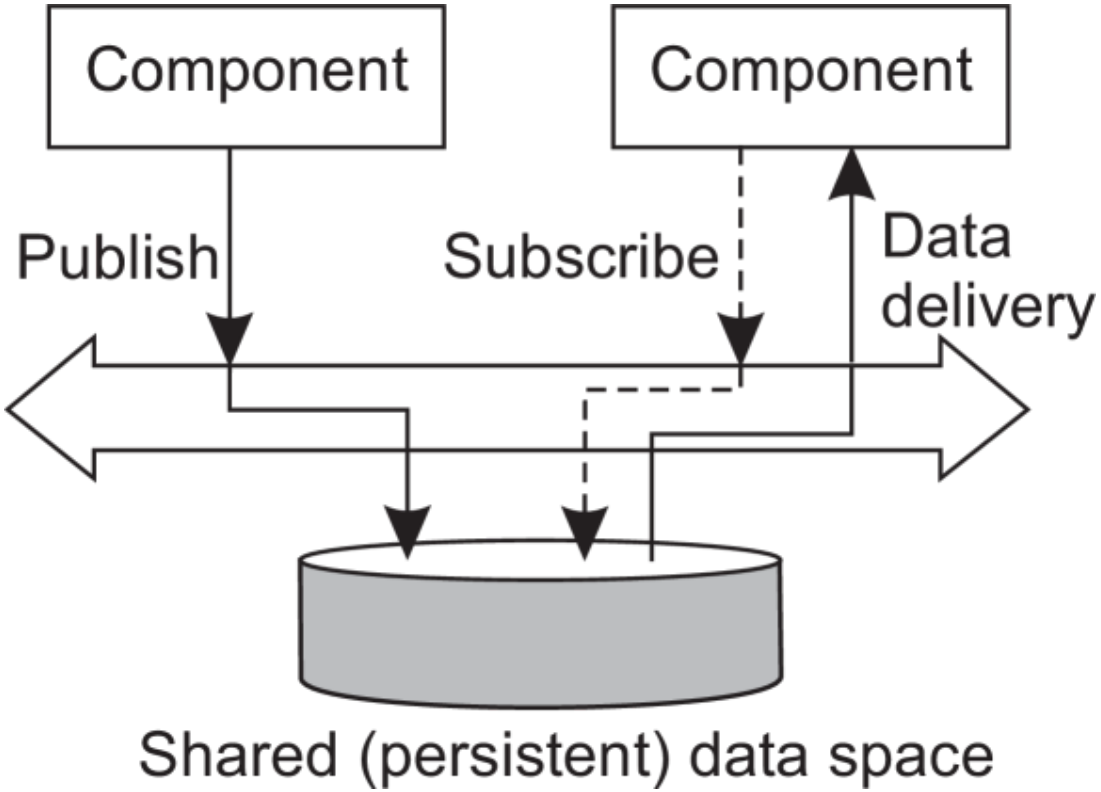
Event-Based Coordination



M. van Steen and A. S. Tanenbaum, *Distributed Systems*,

4th ed., Version 4.02, Feb. 2024.

Shared Data Space



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Publish-Subscribe Mechanism

Event Bus

- Publishers send events to a central **bus**
- Subscribers receive only relevant events
- Processes remain decoupled — no direct references

AASTU

Linda Tuple Spaces

What is a Tuple Space?

- Linda is a programming model developed in the 1980s [Carriero Gelernter, 1989]
- Shared data space is called a **tuple space**
- Tuple space supports concurrent access by multiple processes
- Operations: **out(t)**, **in(t)**, **rd(t)**

AASTU

Tuple Space Operations: Core Operations

Core Operations

- **out(t)**: add tuple `t` to the space
- **in(t)**: remove tuple matching `t` (blocking)
- **rd(t)**: read a copy of tuple matching `t` (blocking)
- Tuple space is modeled as a multiset; multiple copies of the same tuple can exist

AASTU

Tuple Space Operations: Blocking Behavior

Blocking Behavior

- **in** and **rd** block until a matching tuple is found
- Useful for coordination without explicit process communication

AASTU

Tuple Space Operations: Blocking Behavior

```
3 blog._out(("bob","distsys","I am studying chap 2"))
4 blog._out(("bob","distsys","The linda example's pretty simple"))
5 blog._out(("bob","gtcn","Cool book!"))
```

(a) Bob's code for creating a microblog and posting three messages.

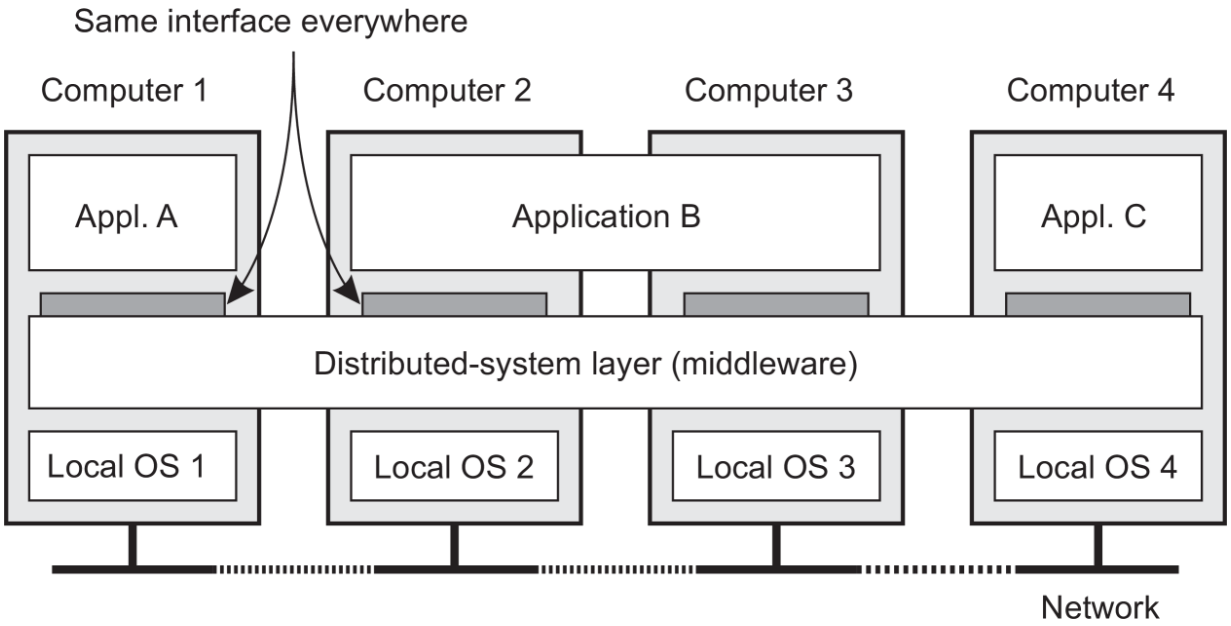
```
1 blog = linda.universe._rd(("MicroBlog",linda.TupleSpace))[1]
2
3 blog._out(("alice","gtcn","This graph theory stuff is not easy"))
4 blog._out(("alice","distsys","I like systems more than graphs"))
```

(b) Alice's code for creating a microblog and posting two messages.

```
1 blog = linda.universe._rd(("MicroBlog",linda.TupleSpace))[1]
2
3 t1 = blog._rd(("bob","distsys",str))
4 t2 = blog._rd(("alice","gtcn",str))
5 t3 = blog._rd(("bob","gtcn",str))
```

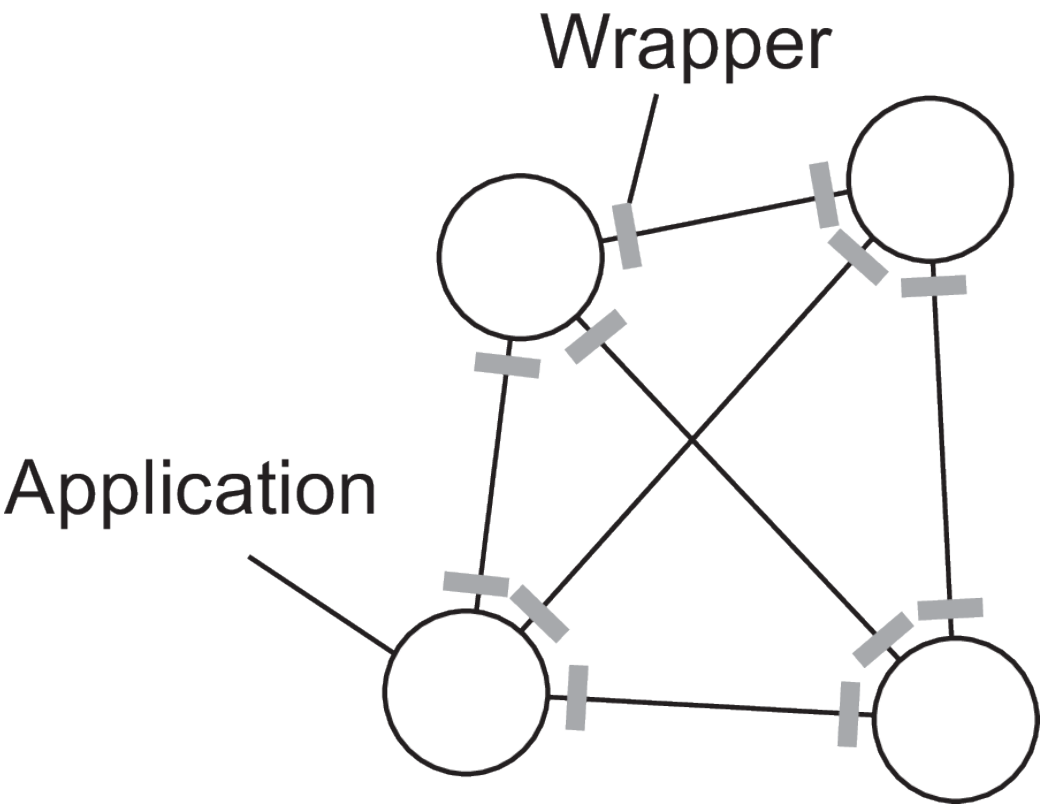
(c) Chuck reading a message from Bob's and Alice's microblog.

Middleware and Distributed Systems



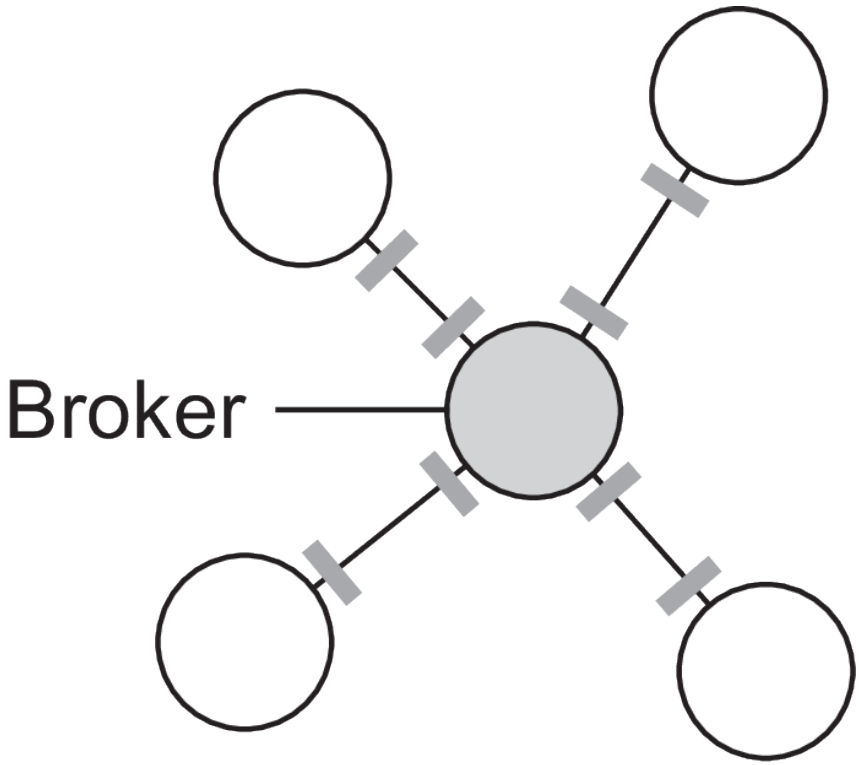
M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Wrappers



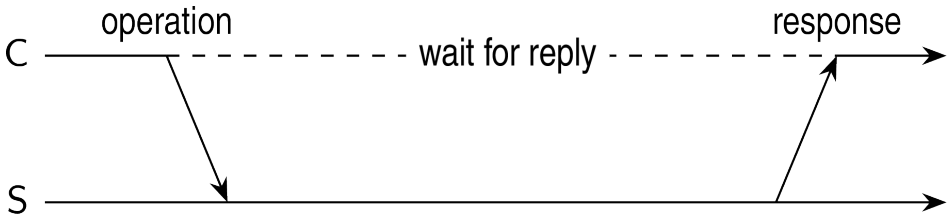
M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Interceptors



M. van Steen and A. S. Tanenbaum,
Distributed Systems, 4th ed., 2024.

Centralized System Architectures



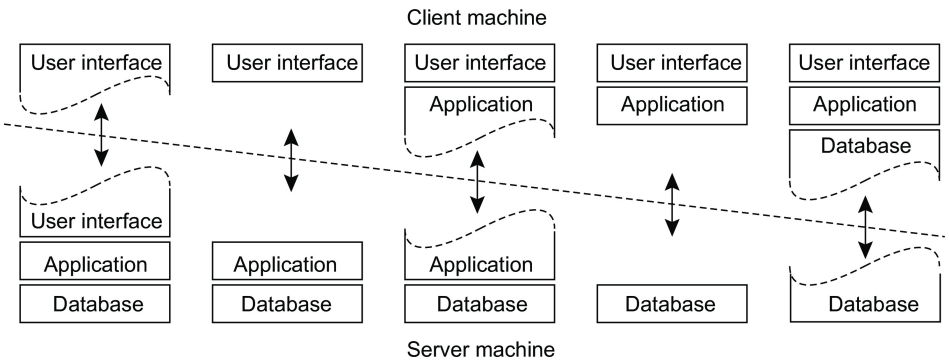
M. van Steen and A. S. Tanenbaum,
Distributed Systems, 4th ed., 2024.

Client–Server Basics

- **Servers:** Provide services
- **Clients:** Request services
- Run on separate machines
- Use request–reply communication

AASTU

Multi-tiered Centralized Architectures



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Architecture Types

- **Single-tier:** Dumb terminal + mainframe
- **Two-tier:** Client + single server
- **Three-tier:** Separate layers on different machines
- Used in many traditional organizations

AASTU

Summary

Key Takeaways

- Distributed systems rely on diverse **architectural styles** (layered, service-oriented, publish–subscribe, microservices).
- **Middleware** ensures integration, flexibility, and communication across components.
- Core **patterns** include client–server and multitiered models.
- **Case studies** such as NFS and the Web highlight real-world applications.

AASTU

Questions?

Thank you for your attention!

AASTU

References

Suggested References

1. M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., Version 4.02, February 2024.
2. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed., Addison-Wesley, 2012.
3. N. Lynch, *Distributed Algorithms*, 2nd ed., Morgan Kaufmann, 1996.
4. R. E. Bryant and D. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 4th ed., Pearson, 2024.

AASTU