

UNIT 2

Architecture

Distributed Systems (CACS352) — BCA III/VI

⚠ Please note

*These notes were prepared with AI assistance and **may contain errors or omissions**. Please cross-check definitions, algorithms and numerical examples against your prescribed textbook (e.g., Tanenbaum & van Steen, Distributed Systems), your class notes and your instructor before relying on them for exams.*

2.1 Architectural Styles

A **software architecture** describes how the components of a system are organised, how they are connected, the data exchanged between them, and how they are configured into a system. A **component** is a modular unit with well-defined required and provided interfaces that can be replaced; a **connector** mediates communication or coordination between components (e.g., RPC, message passing, streaming). The common **architectural styles** are:

1. Layered architecture

Components are organised in layers; a component at layer L_i may call components at the layer below (L_{i-1}) and pass results upward. Control flows top-down for requests and bottom-up for responses. The network protocol stack (OSI/TCP-IP) is the classic example. Variations allow an upcall from a lower layer to a higher one (e.g., callbacks).

2. Object-based (service-oriented) architecture

Components are **objects** that interact through **method invocations** (RPC-like); each object encapsulates state and exposes an interface. Objects may be distributed across machines and accessed through proxies (client side) and skeletons (server side). This style underlies Java RMI and CORBA and, in a coarser form, **service-oriented architectures** (SOA) and microservices, where independent services are composed.

3. Resource-centred (RESTful) architecture

The system is a collection of **resources** named by URIs and manipulated with a small, fixed set of operations (PUT, POST, GET, DELETE). Interactions are stateless: each request contains all information needed to serve it. Representations (JSON, XML) are transferred instead of objects.

4. Event-based (publish–subscribe) architecture

Components communicate by **events** that are published and delivered to interested subscribers. Processes are **loosely coupled**: they need not know each other (referentially decoupled) and need not be active at the same time (temporally decoupled) — e.g., with a message broker. A related style is **shared data-space** (tuple spaces), where processes coordinate through a persistent shared repository.

Style	Coupling	Typical example
Layered	Tight between adjacent layers	Protocol stacks, 3-tier web apps
Object-based / SOA	Interface-level (method calls)	Java RMI, CORBA, web services
Resource-centred (REST)	Uniform interface, stateless	REST APIs over HTTP
Publish–subscribe	Referentially (and often temporally) decoupled	Kafka, MQTT, RabbitMQ

2.2 Middleware Organization

Middleware is the software layer that sits between applications and the operating system to provide distribution transparency and common services. Its **organisation** is described by how it is structured internally and how it can be adapted.

Wrappers (adapters)

A **wrapper** (adapter) converts the interface of a legacy or foreign component into one that other components expect. With N applications, direct pairwise wrapping needs up to $N(N-1)$ wrappers; a **broker** with a common intermediate format needs only $2N$ — this is the idea behind message brokers in EAI.

Example 2.1 — Wrappers versus a broker

An organisation has 6 applications that must all exchange data. How many wrappers (adapters) are needed with (a) point-to-point wrapping, (b) a central broker?

Solution

(a) Every application needs a wrapper for each of the other 5, in each direction: $N(N-1) = 6 \times 5 = 30$ wrappers (15 if a wrapper works both ways).

(b) Each application only converts to/from the broker's common format: $2N = 2 \times 6 = 12$ (6 if bidirectional).

Adding a 7th application costs 12 more wrappers in (a) but only 2 in (b).

Interceptors

An **interceptor** is a software construct that breaks the usual flow of control to let other code run — for example, to intercept a call, transform it, log it or send it to a replica. Interceptors make middleware adaptable without changing application code and are the basis of replicated-object invocation, load balancing and security checks.

Modifiable middleware

- **Separation of concerns** — keep functions such as security or logging separate from business logic.
- **Component-based design** — replace individual parts at run time.
- **Reflection and adaptive software** — a system can inspect and change its own behaviour to respond to changes in its environment.

2.3 System Architecture

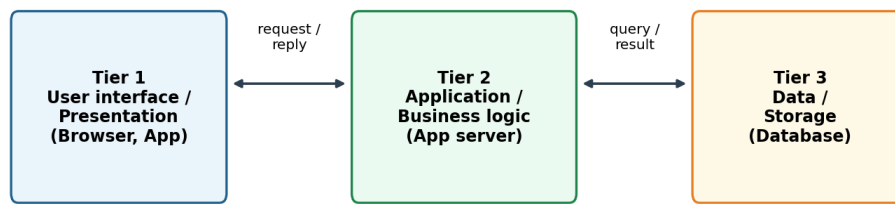
System architecture describes where the software components are actually placed on machines. It is the physical instantiation of an architectural style.

Centralized organisations — client–server

Processes are either **servers** (offer a service) or **clients** (request a service by sending a request and waiting for a reply). Communication is usually request–reply. Simple and easy to manage, but the server is a possible bottleneck and single point of failure.

Multi-tiered architectures

The logical layers — user interface, processing (application logic) and data — are mapped onto physical **tiers**. Two-tier: client + server (thin client vs fat client). Three-tier: client, application server, database server. Distributing tiers on different machines is **vertical distribution**.



Physical distribution: each tier may run on a different machine (vertical distribution)

Figure 2.1 — A three-tier architecture (vertical distribution).

Decentralized organisations — peer-to-peer (P2P)

In **horizontal distribution** every process plays the role of both client and server (a *servant*). Peers form an **overlay network** on top of the physical network.

- **Structured P2P** — the overlay follows a deterministic topology; a **Distributed Hash Table (DHT)** maps keys to nodes, e.g., **Chord**: nodes and keys get m -bit identifiers placed on a logical ring; key k is stored at its *successor* — the first node with $\text{id} \geq k$. Lookup takes $O(\log N)$ hops using finger tables.
- **Unstructured P2P** — neighbours are chosen randomly; search uses flooding or random walks (e.g., early Gnutella). Simple, resilient to churn, but searching is expensive.
- **Hierarchical (super-peer)** — a few powerful super-peers index content for ordinary peers (e.g., Skype's original design).

Example 2.2 — Chord successor

In a Chord ring with $m = 5$ (identifier space $0 \dots 31$), nodes have ids 1, 8, 14, 21, 28. On which node is key 10 stored? On which node is key 30 stored?

Solution

Key k is stored at $\text{successor}(k)$, the first node whose $\text{id} \geq k$ (going clockwise, wrapping around).

Key 10 → the first node ≥ 10 is **14**.

Key 30 → no node ≥ 30 , so wrap around to the smallest id: **1**.

Hybrid architectures

- **Edge-server systems** — servers placed at the edge of the network (near users), e.g., CDNs, ISP-hosted servers.

- **Collaborative distributed systems** — a centralised part for start-up/coordination, then decentralised operation, e.g., BitTorrent (a tracker/DHT finds peers; downloading is peer-to-peer using chunks and tit-for-tat).

2.4 Example Architectures

Network File System (NFS)

NFS is a client–server file system. A client mounts a remote directory into its local name space through a **Virtual File System (VFS)** layer that hides whether a file is local or remote. Server-side, the NFS server exports directories. Classic NFS (v2/v3) servers were **stateless** (simple crash recovery); NFSv4 adds state (open, locking) and compound RPCs. Communication uses RPC.

The Web (Apache-style multi-tier)

A client browser sends HTTP requests to a Web server, which may hand dynamic requests to an application server (via CGI, servlets, PHP, etc.) that queries a database. Large sites add load balancers, caches and CDNs — a hybrid of client–server, replication and edge-server ideas.

BitTorrent

To download a file, a client obtains a **.torrent** (metadata), contacts a tracker (or DHT) for peers, and downloads **pieces** from many peers simultaneously while uploading pieces it already has. A tit-for-tat policy rewards peers who upload, discouraging free-riding.

Example	Style	Key idea
NFS	Client–server (layered)	VFS hides remote vs local; RPC-based
Web	Multi-tier + edge servers	Stateless HTTP, caching, CDNs
BitTorrent	Collaborative P2P	Chunked, parallel download; tit-for-tat
Chord	Structured P2P (DHT)	Ring, consistent hashing, $O(\log N)$ lookup

Key Takeaways

- ✓ Architectural styles: layered, object-based/SOA, resource-centred (REST), event-based (publish–subscribe).
- ✓ Middleware is organised with wrappers, brokers and interceptors so that it can be adapted without changing applications.

- ✓ System architectures: centralised (client–server, multi-tier), decentralised (P2P: structured, unstructured, super-peer) and hybrid (edge servers, BitTorrent).
- ✓ In Chord, key k lives at $\text{successor}(k)$; lookup costs $O(\log N)$ hops.
- ✓ Real systems (NFS, Web, BitTorrent) combine several styles.

— *End of Unit 2* —

bcalink.com