yatlas

SYSTEM DESIGN PREP

(RESOURCES USED)

-youtube:

- BYTEMONK
- BYTEBYTEGO
- TECHWORLD WITH NANA

-infinitecourse.org:

-wikepedia:

computer architecture:

1 bit: smallest unit in computing: can be 0 or 1.
1 byte(8 bits): smallest unit of addressable space that a cpu can reference.

Functional Units and Latency Hierarchy

CPU

- Contains general-purpose registers (32 × 64 bit on x86-64) and the instruction pipeline.
- Access latency to a register file entry: 0.5 ns, one clock cycle at 3 GHz.

Cache subsystem

-3 levels are listed: L1, L2, L3.
• They keep copies of recently used bytes so the CPU does not leave the chip.
• A cache-line (64 Bytes) is the smallest package the cache will carry; even if you ask for one bit, it brings 64 bytes.

- L1 data cache: 32 KiB (1KiB = 1,024 bytes)per core, 8-way set-associative, 64 B line size. (latency ≈ 1 ns || 3–4 cycles)
- L2 unified cache: 256–512 KiB per core, inclusive. (latency ≈ 3–5 ns || 7–14 cycles)
- L3 shared cache: up to 32 MiB per chip, victim cache, 64 B line transfer unit. (latency : ≈ 10–20 ns || 20–40 cycles)
- Cache coherence protocol: MESIF on Intel, MOESI on AMD, maintains consistency across sockets.

RAM (random access memory) “Main Memory”

- Once the on-chip caches miss, the request crosses the motherboard to RAM.
- it contains data structures, variables and application data
- Page size is 4 KB on x86, 16 KB on arm64—those are the crates the warehouse ships. (a note is on the next page)
- NUMA note: sockets have local versus remote memory
- local is faster.
- TLB caches virtual-to-physical mappings.

note

- ARM is often preferred for cost optimization, energy efficiency, and integration into small form factors and IoT devices. x86 remains dominant in environments requiring extensive software compatibility, high processing speed, and advanced features like virtualization and hyper-threading.
- Non-uniform memory access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor. Under NUMA, a processor can access its own local memory faster than non-local memory (memory local to another processor or memory shared between processors).
- The CPU provides a mechanism called the Translation Lookaside Buffer (TLB) to ensure that recently accessed pages of memory can be identified and read faster using L1 or L2 CPU cache.

ROM (read only memory)

-Rotational storage: HDD

- 7200 RPM HDD: average rotational latency 4.17 ms, average seek 4 ms, total 8–10 ms.
- Sector size: 512 B logical, 4 KiB physical; NCQ supports 32-command reordering.

-Solid-state drive: SSD

- NVMe SSD: PCIe Gen4 ×4 link, 64 KiB deep queue, 150 µs typical read latency.
- NAND flash page: 4–16 KiB; erase block: 256 KiB–4 MiB; ECC engine on-die.

goal of system design:

scalability - maintainability - reliability - efficiency

2 cores principles: availability and reliability

- **availability** depends on SLO**(service level objectives)** and SLA**(service level agreements).**
  availability(%) = functionning / timeframe
  -functionning= uptime.
  -timeframe= uptime+downtime.
- **reliability** depends on the meantime between failures which is the **total time in service/number of failures**

ps: we should focus more on availability since it is easier to improve (example: have more backup resources)

3 key elements of good design

- **moving data:** we should optimise it for speed and security
- **storing data(more on this later):** access patterns - indexing strategies - backup solutions
- **transforming data:** turning raw data into meaningful information

speed

- **throughput:** how much data can be handeled over a period of time.

-RPS: number of requests a server can handle per sec
-QPS: number of queries a DB can handle per sec
-data throughput: number of data handeled per sec in a network(B/s)

- **latency:** how long it takes to handle a request and get a response

ps: optimising one can diminish the other

cap theorem

-The CAP theorem, formulated by Eric Brewer, states that in a distributed data store, you can only guarantee two of the following three properties at the same time:

- **Consistency:** Every read receives the most recent write or an error. All nodes in the system view the same data at the same time.
- **Availability:** Every request (read or write) receives a response, even if it's not the most recent data. The system remains operational and responsive.
- **Partition Tolerance:** The system continues to operate despite network partitions that prevent some nodes from communicating with others.

Example: Bank Transfer

Consider a bank transfer operation where funds are transferred from Account A to Account B. This process involves three databases:

- **Database A: Holds the balance of Account A.**
- **Database B: Holds the balance of Account B.**
- **Database C: Logs the transaction.**

During a transfer, the system must ensure that:

- **Consistency:** If Account A shows a balance of $100 and Account B shows $50, after a transfer of $30, Account A should show $70 and Account B should show $80. All databases must reflect this update simultaneously.
- **Availability:** If a user initiates the transfer, the system must respond instantly, confirming that the transfer has been initiated, even if the databases are not in sync at that moment.
- **Partition Tolerance:** If a network failure occurs, and Database A cannot communicate with Database B, the system must still function.

Conclusion

According to the CAP theorem, in the event of a network partition, **a distributed system can only guarantee two of these properties**. For example, if the system prioritizes Consistency and Partition Tolerance (CP), it may refuse to process requests until the partition is resolved, sacrificing Availability. Conversely, if the system prioritizes Availability and Partition Tolerance (AP), it may allow the transfer to proceed even if it could lead to inconsistent data across the databases. Thus, the CAP theorem illustrates the trade-offs inherent in distributed systems.

networking

DEFINITIONS

- **IP Address**

32-bit (IPv4) or 128-bit (IPv6) logical identifier assigned to a host interface; used for end-to-end routing across networks.

- **MAC Address**

48-bit (EUI-48) hardware identifier burned into the NIC; unique per device; used for L2 frame delivery inside a single broadcast domain.

OSI model

Layer	Name	Function	Typical protocols/devices
7	Application	User-interface data	HTTP, HTTPS, SMTP, DNS, DHCP, FTP, SSH, Telnet, SNMP, NTP, MySQL, PostgreSQL, Redis, Docker, nginx, Kubernetes API
6	Presentation	Syntax/encryption/compression	TLS, SSL, JSON, XML, ASCII
5	Session	Dialog control & synchronisation	NetBIOS, RPC, SQL-session
4	Transport	End-to-end segments, ports, reliability	TCP, UDP, QUIC
3	Network	Routing & logical addressing	IP (v4/v6), ICMP, ARP, OSPF, routers
2	Data-Link	MAC framing, local delivery	Ethernet, Wi-Fi (802.11), PPP, switches
1	Physical	Bits on wire/fiber/wireless	1000BASE-T, 10GBASE-SR, fiber, copper, NIC LEDs

TCP/IP model

Layer	Name	Covers OSI layers	Function	Protocols
4	Application	5-6-7	Process-to-process messages	HTTP, HTTPS, SMTP, DNS, DHCP, FTP, SSH, Telnet, SNMP, NTP, MySQL, PostgreSQL, Redis, Docker, nginx, k8s API
3	Transport	4	Segments, ports, reliability	TCP, UDP, QUIC
2	Internet	3	Packets, routing, IP	IPv4/IPv6, ICMP, ARP, OSPF
1	Network Access	1/2/2025	Frames & bits, local media	Ethernet, Wi-Fi, PPP, NIC drivers

TCP vs UDP

- **TCP**

Connection-oriented: 3-way handshake, 4-way close
Reliable: sequence numbers, ACKs, retransmissions, congestion control
Ordered: byte-stream, no duplication
Use-cases: Web (HTTP/S), email (SMTP), file (FTP), DB (MySQL)

- **UDP**

Connection-less: no handshake, no state
Unreliable: no ACK, no retransmission
Message-oriented: datagram boundaries preserved
Use-cases: DNS, VoIP, video streaming, SNMP, DHCP

TCP 3 way hadshake and 4 way close

- **3-Way Handshake (connection establishment)**

SYN – Client sends segment with SYN=1, seq=x
SYN-ACK – Server replies SYN=1, ACK=1, seq=y, ack=x+1
ACK – Client sends ACK=1, seq=x+1, ack=y+1
After step 3 both sides are in ESTABLISHED state and byte-stream begins.

- **4-Way Close (connection termination)**

FIN – Active side sends FIN=1, seq=u
ACK – Passive side acknowledges ack=u+1 (enters CLOSE-WAIT)
FIN – Passive side later sends its own FIN=1, seq=v
ACK – Active side acknowledges ack=v+1 (enters TIME-WAIT 2×MSL)

DNS

- DNS translates human-readable names (www.example.com) into IP addresses.
- Resolver library on host first checks local cache; if miss, query is sent to recursive resolver (usually ISP or public DNS).
- Recursive server walks hierarchy: root (.) → TLD (.com) → authoritative (example.com).
- Each step is a UDP datagram on port 53; if response > 512 B, TCP 53 is used.
- Resource-record types: A (IPv4), AAAA (IPv6), CNAME (alias), MX (mail), NS (name-server), TXT (text).
- TTL field controls caching duration in seconds.

EDNS-client-subnet extension passes end-user network prefix to authoritative server, enabling geo-aware replies.

COMMON PROTOCOLS, PORTS, TRANSPORT

Protocol	Port	Transport
HTTP	80	TCP
HTTPS	443	TCP
SMTP	25	TCP
DNS	53	TCP/UDP
DHCP	67/68	UDP
FTP	21	TCP
SSH	22	TCP
Telnet	23	TCP
SNMP	161/162	UDP
NTP	123	UDP
MySQL	3306	TCP
PostgreSQL	5432	TCP
Redis	6379	TCP
Docker API	2375 (plain) 2376 (TLS)	TCP
nginx (default)	80 / 443	TCP
Kubernetes API server	6443	TCP
Kubernetes kubelet	10250	TCP

API design

REST API BASICS

- REST **(Representational State Transfer)** is an architectural style, not a standard; it uses HTTP semantics as the contract.
- Every resource is identified by a unique URL (nouns, not verbs).
- Use HTTP methods explicitly: GET = read, POST = create, PUT = full update, PATCH = partial update, DELETE = remove.
- Interaction must be stateless—each request contains all information needed to service it; server stores no client context between calls.

RESOURCE NAMING CONVENTIONS

- **GET /users –** list all users
- **GET /users/{id} –** retrieve specific user
- **POST /users –** create new user
- **PUT /users/{id} –** replace entire user
- **PATCH /users/{id} –** partial update
- **DELETE /users/{id} –** remove user

1. Avoid verbs inside the URL; the method is the verb.
2. use plural nouns for collections;
3. use path parameters for identifiers,
4. use query parameters for filters, sorting, pagination.

HTTP STATUS-CODE CATEGORIES

- 1xx Informational (rarely used)
- 2xx Success – 200 OK, 201 Created, 204 No Content
- 3xx Redirection – 301 Moved Permanently, 302 Found, 304 Not Modified
- 4xx Client Error – 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 409 Conflict
- 5xx Server Error – 500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable

API design

REQUEST & RESPONSE HEADERS

Common request headers:

- Authorization – bearer token or basic auth
- Content-Type – application/json, multipart/form-data
- Accept – what representation client wants

Common response headers:

- Content-Type – actual format returned
- Location – URI of newly created resource (201)
- Cache-Control – max-age, no-cache directives
- X-Rate-Limit-Remaining – custom meta-data (prefix with X-)

IDEMPOTENCY & SAFETY

1. GET, PUT, DELETE are idempotent – repeating the call yields the same server state.
2. POST is not idempotent – repeating it may create multiple resources.

VERSIONING STRATEGIES (3 mainstream approaches)

- URL path: /v1/users, /v2/users – simple, visible in logs
- Query param: /users?version=1 – avoids path change
- Header: Accept: application/vnd.myapi.v1+json – clean URI, harder to debug

Keep old version running for a deprecation window; return Sunset header with date.
sunset header format: "Sunset: Fri, 31 Dec 2023 23:59:59 GMT"

API design

PAGINATION, FILTERING, SORTING

pagination

Imagine the database has one million users.
If you return them all in a single JSON you will:

- **Crash the browser**
- **Kill your DB**
- **Exceed every timeout**

So you give the caller one page at a time.
Two ways to pick that page:

A. Offset paging (old-school)

URL: /users?offset=0&limit=20
Meaning: skip 0 rows, take 20 rows.
Good: easy to understand.
Bad: on big offsets the DB still counts/skipps 40 000 rows just to give you rows 40 000-40 020 → gets slower and slower.

B. Cursor paging (modern)

URL: /users?after_id=40abc&limit=20
Meaning: give me the 20 rows that come after the row whose id = 40abc.
Good: constant speed no matter how deep you go.
Bad: user can't jump to "page 50" without walking through previous pages.

Filtering

Add WHERE-clauses through query-params:
/users?status=active&role=admin
Server ANDs those conditions.

Sorting

Tell the server which column and direction:
/users?sort=created_at:desc,name:asc
First sort by created_at newest, then by name A-Z.

ERROR RESPONSE FORMAT

Always return a consistent JSON error object:

```
{
"error": {
"code": 40001,
"message": "Invalid email format",
"field": "email"
  }
}
```

Use a global error code namespace; do not expose stack traces in production.
Set correct HTTP status code at the top level.

SECURITY MUST-DO'S

1. Use HTTPS everywhere – redirect HTTP to 443.
2. Authenticate with JWT or OAuth2 bearer tokens in Authorization header.
3. Authorize every endpoint – default deny.
4. Validate & sanitize all inputs – reject unexpected fields.
5. Rate-limit by IP or by token – return 429 Too Many Requests.
6. Enable CORS **only on necessary origins**; use strict Content-Security-Policy headers.

CASHING & CDNs

DEFINITIONS

Cache: temporary, fast storage that keeps copies of expensive data so later requests finish quicker.
CDN: geographically spread cache servers that serve static (and some dynamic) content closer to users, cutting latency and origin load.

CACHE TYPES

1. Browser cache – HTTP Cache-Control, Expires, ETag
2. CDN cache – geo-replicated POPs
3. Application cache – in-memory store (Redis, Memcached)
4. Database cache – query-result cache, buffer pool

CDN FLOW (CACHE-HIT vs CACHE-MISS)

1. User DNS resolved to closest CDN POP
2. TCP + TLS to browser
3. Browser checks local cache

- HIT: serve immediately, 0-ms origin RTT
- MISS: CDN performs origin fetch (HTTP GET), stores copy, returns to client

PULL vs PUSH MODELS

Pull (origin shield) – browser requests object only when first client asks; default mode.
Push (prefetch) – customer uploads content to CDN storage bucket; CDN replicates proactively; useful for large releases.

APPLICATION-CACHE PATTERNS

- Read-through: app checks cache; on miss fetch DB, store, return.
- Write-through: app writes to cache and DB synchronously; cache always hot.
- Write-behind: app writes to cache only; async job flushes to DB later; faster but risk of data loss on crash.

CACHE EJECTION (EVICTION) POLICIES

When the cache is full, the algorithm decides which item to eject (remove).

- **LRU – Least Recently Used**

Evicts the item that has not been accessed for the longest time.
Good for temporal locality (hot items stay).

- **LFU – Least Frequently Used**

Evicts the item with the lowest access count.
Good for skewed popularity (popular items stay).

- **FIFO – First-In, First-Out**

Evicts the oldest inserted item, regardless of usage.
Simple, but can eject hot items that arrived early.

- **TTL – Time-To-Live**

Item is evicted when its expiration timestamp is reached; independent of access pattern.
Mandatory for correctness when data can become stale.

COMMON CDN FEATURES (beyond caching)

1. Anycast routing – single IP, nearest POP auto-selected
2. Compression – gzip/Brotli on-the-fly
3. Image optimization – WebP/AVIF conversion, resize
4. Edge Workers – run JavaScript at POP (A/B routing, header injection)
5. DDoS protection – absorb > 1 Tbps floods before origin sees packet

PROXY, LOAD BALANCING, CLUSTERS

MOST USED AND KNOWN PROXY TYPES

Forward Proxy (client-side)

Sits between user and internet; hides client identity; used for access control, caching, content filtering.

Example: corporate proxy blocks social-media sites.

Reverse Proxy (server-side)

Sits between internet and origin servers; hides server identity; provides SSL termination, caching, compression, request routing.

Example: nginx in front of Node.js fleet.

REVERSE-PROXY BENEFITS

1. SSL termination – decrypts HTTPS, sends HTTP to origins
2. Compression – gzip/Brotli on the fly
3. Caching – serves static assets without hitting origin
4. Request routing – route /api/v1 → service A, /web → service B
5. IP allow-list / deny-list – edge security
6. Hide topology – origins never exposed to internet

LOAD BALANCER = REVERSE-PROXY + DISTRIBUTION

A load balancer is a reverse proxy that also spreads traffic across multiple upstream servers to increase capacity and availability.

Health-checks remove failed nodes; algorithms decide which node gets the next request.

LB ALGORITHMS

- Round-robin – sequential list, equal weight.
- Least-connections – send to node with fewest active TCP sessions.
- IP-hash – hash(client IP) modulo server count; same client → same node (session affinity).
- Weighted-round-robin – assign larger share to bigger boxes.
- Least-time – node with lowest response time.
- Random – pick any alive node; simple, fair under high load.

HEALTH-CHECK MECHANICS

Default probe: TCP connect on service port every 5 s.
HTTP probe: GET /healthz expecting 200 OK within 2 s.
Failure threshold: 3 consecutive fails → node marked DOWN; traffic stops, probe continues.
Recovery: 3 consecutive passes → node marked UP; traffic resumes.

SESSION STICKINESS (AFFINITY)

When state is stored on a specific server, subsequent requests must return to that server.
Methods:

- IP-hash algorithm (stateless)
- Cookie insert – LB sets "SERVERID=node3" cookie, reads on return

SSL TERMINATION AT LB

LB decrypts client TLS, forwards plain HTTP to origins → saves CPU on app servers.
Can re-encrypt (SSL-bridge) or leave clear (SSL-offload).
Certificate lives on LB; renewal handled there.

HIGH-AVAILABILITY SET-UP

- **Active-Passive** – one LB handles traffic, standby takes over via VRRP/keepalived when primary fails (virtual IP floats).
- **Active-Active** – multiple LBs share traffic using ECMP or DNS round-robin; requires shared state table or stateless algorithms.

note:

- **Equal-cost multi-path routing (ECMP) is a routing strategy where packet forwarding to a single destination can occur over multiple best paths with equal routing priority.**
- **The Virtual Router Redundancy Protocol (VRRP) is a computer networking protocol that provides for automatic assignment of available Internet Protocol (IP) routers to participating hosts.**

CLUSTERS BASICS

A cluster is a group of machines (nodes) that work together and appear to the outside world as a single system.
Purpose: horizontal scaling, high availability, fault tolerance.
Shared-nothing architecture: each node owns its local CPU, RAM, disk; nodes communicate over the network.

CLUSTER TYPES

1. Compute cluster – distribute CPU-bound jobs (e.g., SLURM, Hadoop YARN)
2. Storage cluster – replicate or shard data (e.g., Ceph, GlusterFS)
3. Load-balancer cluster – several LB nodes fronted by ECMP or VRRP (active-active or active-passive)

CLUSTER VS SINGLE NODE AVAILABILITY

Single node = single point of failure (SPOF).
Cluster removes SPOF by redundancy: if one node dies, remaining nodes continue service.
Trade-off: added complexity (network partitions, split-brain, config sync).

CLUSTER MEMBERSHIP & COORDINATION

- Health-check: each node pings neighbours; failed node is removed from pool.
- Leader election: active-passive clusters use Raft, Paxos, or VRRP to pick primary node.
- Shared state: use external quorum (etcd, Consul, ZooKeeper) to avoid split-brain.

DATABASES

ACID VS BASE

acid:

A – Atomicity: if any part of the transaction fails, the whole thing is rolled back; no partial changes survive.
C – Consistency: every commit moves the database from one valid state to another, preserving all declared rules (keys, constraints, triggers).
I – Isolation: concurrent transactions execute as if they were serial; intermediate rows are invisible to others until commit.
D – Durability: once commit returns success, the data is safe on disk even if the server crashes immediately after.

base: acid without the 'c'

B – Basically Available: the system guarantees availability of data even if parts of it are failing; users can always read and write.
A – Soft state: the value of a piece of data may change over time due to eventual background sync; it is not fixed until convergence.
S – Eventually consistent: given enough time and no new updates, all replicas will converge to the same value; reads may temporarily return stale data.

DATABASE TYPES

SQL (relational)

- Table-based, ACID compliance, strong consistency
- Vertical scaling preferred (bigger box)
- Examples: MySQL, PostgreSQL, Oracle

NoSQL (non-relational)

- Document, key-value, column-family, graph
- BASE property (Basically Available, Soft state, Eventual consistency)
- Horizontal scaling, flexible schema
- Examples: MongoDB (document), Redis (key-value), Cassandra (column), Neo4j (graph)

SCALING STRATEGIES

- **Vertical scaling** – add CPU/RAM/SSD to one box; limit = hardware ceiling.
- **Horizontal scaling** – add more machines; need application-level sharding or replication.
- **Read replicas –**

1. One primary node accepts all writes; its transaction log is streamed to one or more secondary servers.
2. Secondaries replay the log, staying a few seconds behind (async) or in sync (semi-sync).
3. Client applications route SELECT queries to replicas, spreading read load; writes still go to the primary.
4. If the primary fails, a replica can be promoted to primary (manual or automatic failover).

- **Sharding –**

split one logical database across multiple independent servers (shards). Each server owns a slice of the rows, acts as the primary for that slice, and handles both reads and writes for its slice.

REPLICATION TOPOLOGIES

- Master-slave – one primary accepts writes, slaves replay log; simple, but write bottleneck.
- Master-master (multi-primary) – both nodes write; conflict resolution needed (last-write-wins, timestamps).

NORMALISATION vs DENORMALISATION

NORMALISATION

- You keep one column for “Department” and put all employee rows underneath it.
- No cell is ever duplicated; if “HR” changes its name you edit one cell.
- Trade-off: to get the department’s phone you must do a second lookup (join).

DENORMALISATION

- You copy the department name and phone into every employee row.
- One SELECT gives you everything — no join.
- Trade-off: if the phone changes you must update many rows (risk of inconsistency).

DATABASE PERFORMANCE

- **Caching**

1. Keep hot data in RAM so the disk is never touched.
2. Buffer pool (InnoDB, Oracle) – pages cached automatically; aim > 99 % hit ratio.
3. Query-cache (deprecated in MySQL 8) – exact SQL match, TTL-based.
4. Application-side – Redis/Memcached in front of DB; store primary-key lookups, counts, session data.
5. CDN/edge – cache HTTP responses that come from DB-driven pages.

- **Indexing**

1. Create data structures that let the engine jump to rows instead of scanning every page.
2. Primary index – clustered, determines physical order.
3. Secondary indexes – B-tree or hash; cover WHERE, JOIN, ORDER BY.
4. Composite – order matters (left-prefix rule); put highest-selectivity column first.
5. Covering – index contains all columns needed → no table look-up.
6. Over-indexing hurts writes – each extra index is another disk write.

- **Query Optimisation**

1. Write SQL that asks for the smallest data set in the cheapest way.
2. SELECT only needed columns; avoid SELECT *.
3. Push filters into WHERE (don't fetch then filter in app).
4. Use LIMIT / pagination instead of returning millions of rows.
5. Replace N+1 loops with single JOIN or bulk IN (…) statement.
6. Let the engine sort: ORDER BY with covering index avoids filesort.
7. Parameterise queries – prevents re-parse overhead and injection.
8. Update statistics & ANALYZE so optimiser chooses correct indexes.
9. Partition large tables (range or hash) to prune data early.

SQL QUERY EXECUTION ORDER

1. FROM/JOIN – decide which tables and how they are joined
2. WHERE – filter rows
3. GROUP BY – form groups
4. HAVING – filter groups
5. SELECT – choose columns/expressions
6. DISTINCT – remove duplicate rows
7. ORDER BY – sort the result
8. LIMIT/OFFSET – return only the requested window