PREMIUM · LEARN.AIWITHPRIYABRAT.COM

System Design Quick Revision Sheet

The numbers, building blocks, trade-offs and classic problems you need for a system design round — on printable pages you can revise in 15 minutes and keep coming back to.

By **Priyabrat Dalbehera** · GenAI engineer

13 pages · **11** parts · **132** reference cards · **26** classic problems · **30** interview Q&As

What's inside

1 Interview framework — p. 2
6 steps with timings, questions to ask, scoring

2 Numbers to know — p. 3
Latency, throughput, sizes, conversions, nines

3 Estimation — p. 4
Formulas + a full worked example

4 Traffic building blocks — p. 5
DNS, load balancers, gateway, CDN, rate limiting

5 Caching — p. 6
Patterns, eviction, stampedes, Redis

6 Databases & storage — p. 7
SQL vs NoSQL, replication, sharding, isolation

7 Distributed systems — p. 8
CAP, consistency, consensus, IDs, failures

8 Messaging & APIs — p. 9
Kafka vs RabbitMQ vs SQS, REST, gRPC, WebSockets

9 Scale & reliability — p. 10
Scaling steps, SLOs, monitoring, DR, security

10–11 26 classic problems — p. 11
URL shortener to payments, Docs and an LLM chatbot

12 Trade-offs & Q&A — p. 13
Core trade-offs, 30 questions, glossary

15-minute revision
Read parts 1–3 the night before; skim the problem cards an hour before the round.

Tip & Watch out
Green boxes are what interviewers want to hear. Amber boxes are traps.

Real numbers
Latency, throughput and availability tables with a fully worked estimation.

learn.aiwithpriyabrat.com

© 2026 Priyabrat Dalbehera · **Free preview · the full sheet is at learn.aiwithpriyabrat.com**

2.1 Latency numbers every engineer should know

Operation	≈ Time	Scale
L1 cache reference	1 ns	
Branch mispredict	3 ns	
L2 cache reference	4 ns	
Mutex lock/unlock	17 ns	
Main memory reference	100 ns	100× L1
Compress 1 KB (fast codec)	2 µs	
Read 1 MB sequentially from memory	3 µs	
Read 4 KB randomly from SSD	16 µs	~150× memory
Read 1 MB sequentially from SSD	50 µs	
Round trip within a datacenter	0.5 ms	
Read 1 MB sequentially from HDD	1 ms	~20× SSD
HDD seek	2–10 ms	
Round trip across regions	50–150 ms	speed of light
Packet California → Netherlands → California	150 ms	

Approximate figures for modern hardware (after the classic Dean/Norvig list). Remember the **ratios**: memory ≫ SSD ≫ network ≫ disk seek ≫ cross-region.

2.2 Service-level latencies (rough)

Call	Typical
In-memory cache GET (Redis, same AZ)	0.2–1 ms
Indexed DB read (primary key)	1–5 ms
DB write with fsync / replication	5–20 ms
Cross-AZ round trip	1–2 ms
Third-party API call	50–500 ms
LLM API call	0.5–10 s
Human perception: "instant"	< 100 ms

FREE PREVIEW · FULL SHEET AT LEARN.AIWITHPRIYABRAT.COM

2.3 Throughput rules of thumb (per node)

Component	Order of magnitude
Stateless web/API server	1k–10k+ RPS (depends on work per request)
Redis / Memcached	~100k+ ops/s
PostgreSQL / MySQL	1k–20k simple queries/s; fewer heavy writes
Kafka broker	100s of MB/s; millions of small msgs/s per cluster
1 Gbps network	≈ 125 MB/s
NVMe SSD sequential	2–7 GB/s
HDD sequential	100–250 MB/s

Watch out: These are starting points for estimates, not guarantees: say "roughly" and benchmark in real life.

2.4 Data sizes

Unit	Bytes	≈
KB	2^{10}	10^3 (thousand)
MB	2^{20}	10^6 (million)
GB	2^{30}	10^9 (billion)
TB	2^{40}	10^{12} (trillion)
PB	2^{50}	10^{15}

Object	Rough size
ID (int64) / UUID	8 B / 16 B
Short text post / chat message	100–500 B
JSON API record	0.5–2 KB
Thumbnail / photo	20–50 KB / 0.2–3 MB
1 min of video (720p–1080p)	5–50 MB

2.5 Time conversions

Per day	≈ Per second
1 day	86,400 s ≈ **10^5 s**
1 million requests/day	≈ 12 RPS
10 million/day	≈ 120 RPS
100 million/day	≈ 1,200 RPS
1 billion/day	≈ 12,000 RPS

Tip: Peak ≈ 2–3× average (up to 10× for spiky events). 1 month ≈ 2.6 M s; 1 year ≈ 31.5 M s.

2.6 Availability: the nines

Availability	Down per year	Per month	Per week
99% (two nines)	3.65 days	7.3 h	1.7 h
99.9%	8.8 h	43.8 min	10.1 min
99.95%	4.4 h	21.9 min	5 min
99.99%	52.6 min	4.4 min	1 min
99.999%	5.3 min	26 s	6 s

Components in **series** multiply: two 99.9% services → 99.8%. In **parallel** (redundant): 1 − (0.001 × 0.001) ≈ 99.9999%.

2.7 Connections & concurrency

Thing	Rough limit
Concurrent TCP / WebSocket connections per tuned server	100k–1M (memory-bound)
Threads per server (thread-per-request)	Hundreds to a few thousand
Async/event-loop server	10k+ concurrent requests
Postgres connections (practical)	Hundreds: use a pooler

2.8 Cloud cost rules of thumb

Resource	Order of magnitude
Object storage (standard tier)	≈ $0.02 per GB-month
Cold / archive storage	5–20× cheaper, slow to retrieve
Data egress to the internet	≈ $0.05–0.09 per GB, often the surprise bill
Memory vs SSD vs object storage	RAM ≫ SSD ≫ object storage per GB

Approximate list prices; they vary by provider, region and discounts. Use them to compare designs, not to quote.

2.9 Powers of two you'll actually use

Power	Value	Where it shows up
2^{10}	1,024 ≈ 1 thousand	KB
2^{16}	65,536	Ports per IP; small counters
2^{32}	≈ 4.3 billion	IPv4 addresses; int32 IDs run out here
2^{64}	≈ 1.8×10^{19}	int64 IDs: effectively unlimited
62^7	≈ 3.5 trillion	7-char base-62 short codes

2.10 Bandwidth conversions

Thing	Rough rate
1 Gbps / 10 Gbps link	125 MB/s / 1.25 GB/s
HD (1080p) video stream	~5 Mbps
4K video stream	~15–25 Mbps
Voice call / video call	~0.1 Mbps / 1–3 Mbps
Typical mobile (4G/5G)	10–100+ Mbps, high variance

5.1 Where caching happens

Layer	Caches	Example
Browser / client	Static files, API responses	HTTP cache headers
CDN / edge	Static assets, media, public pages	CloudFront, Cloudflare, Akamai
Application (in-process)	Hot config, small lookups	LRU dict, Caffeine
Distributed cache	Sessions, objects, query results	Redis, Memcached
Database	Buffer pool, query cache	Postgres shared buffers

5.2 Caching patterns

Pattern	How	Good for
Cache-aside (lazy)	App reads cache; on miss reads DB and fills cache	Default; read-heavy
Read-through	Cache itself loads from DB on miss	Simpler app code
Write-through	Write cache and DB together	Fresh cache; slower writes
Write-back (behind)	Write cache; flush to DB later	Write-heavy; risk of loss
Write-around	Write DB only; cache fills on read	Data rarely re-read

```
def get_user(uid):
    user = cache.get(f"user:{uid}")
    if user is None:                        # miss
        user = db.fetch_user(uid)
        cache.set(f"user:{uid}", user, ttl=300)
    return user

def update_user(uid, data):
    db.update_user(uid, data)
    cache.delete(f"user:{uid}")             # invalidate
```

5.3 Eviction policies

Policy	Evicts	Note
LRU	Least recently used	Good default
LFU	Least frequently used	Stable hot sets
FIFO	Oldest inserted	Simple, rarely best
TTL	Expired entries	Bounds staleness; combine with LRU

5.4 Cache problems and fixes

Problem	What happens	Fix
Stampede (thundering herd)	Hot key expires; thousands hit the DB at once	Lock/single-flight on refill; early refresh; jittered TTL
Penetration	Requests for keys that don't exist always miss	Cache negative results; Bloom filter
Avalanche	Many keys expire together, or the cache dies	Random TTLs; replicas; circuit breaker to DB
Hot key	One key overloads one node	Replicate the key; local cache; split key
Stale data	Cache disagrees with DB	Delete on write; short TTL; versioned keys

5.5 Redis vs Memcached

	Redis	Memcached
Data types	Strings, hashes, lists, sets, sorted sets, streams	Strings only
Persistence	Optional (RDB snapshots, AOF)	None
Replication / cluster	Yes	Client-side sharding
Extras	Pub/sub, Lua, TTLs, geo, rate limiting	Simple, multi-threaded

Tip: Sorted sets → leaderboards; INCR + EXPIRE → rate limits; SETNX with TTL → simple locks.

5.6 Cache sizing & hit ratio

- Start with the hot set: ~20% of data serves ~80% of reads.
- Track hit ratio (aim 80–99% for read-heavy paths), evictions and memory.
- A 90% → 99% hit ratio cuts DB load **10×**: small hit-ratio gains matter.

5.7 Cache key design

- Namespaced and versioned: `v2:user:42:profile`: bump the version to invalidate everything.
- Include every input that changes the result (user, locale, filters, page).
- Never cache per-user data under a shared key: a classic data-leak bug.
- Keep values small; compress large ones; avoid giant hot keys.

5.8 Running a distributed cache

Setup	Notes
Redis replica + Sentinel	Automatic failover for a single primary
Redis Cluster	16,384 hash slots spread over shards; scales writes
Client-side consistent hashing	Memcached style; add nodes with little reshuffling
Managed (ElastiCache, Memorystore)	Less ops; same patterns

Watch out: The cache is not the source of truth: the system must survive a cold or lost cache (protect the DB with rate limits).

5.9 What not to cache

- Rapidly changing data that must be exact (account balances, stock counts at checkout).
- Data read once (no reuse) or very large blobs better served by a CDN.

FREE PREVIEW · FULL SHEET AT LEARN.AIWITHPRIYABRAT.COM

5.10 Keeping cache and DB consistent

- Prefer **update DB, then delete the cache key** (not update the cache): next read refills fresh data.
- Race: a slow reader can re-insert stale data after the delete → short TTLs, or delete again after a delay.
- At scale, invalidate from the DB change stream (CDC) so every writer path is covered.
- Accept bounded staleness explicitly in requirements ("feed may be 30 s old").

5.11 The 30-second caching answer

```
"Reads dominate, so I'd use cache-aside with Redis
in front of the DB: keys like user:{id}, TTL 5 min,
delete on write. Hot static content goes on the CDN.
To avoid stampedes: single-flight refill and jittered
TTLs. We'd watch hit ratio and DB load; the system
must still work if the cache is cold."
```