Back-of-the-Envelope Calculations Guide for System Design Interviews

Table of Contents

Introduction

> **"In system design interviews, being approximately right is better than being precisely wrong."**

Back-of-the-envelope calculations are rough estimates used to evaluate the feasibility of a system design. Interviewers want to see:

- Can you estimate scale and resource requirements?
- Do you understand trade-offs between different approaches?
- Can you identify potential bottlenecks?

You don't need a calculator! Simple multiplication and division with rough numbers is enough.

Core Principles

1. Round Numbers Liberally

Bad: "We have 387,654,321 daily active users..."
Good: "We have ~400 million daily active users..."

Why: Easier to calculate, equally accurate for capacity planning

2. Use Powers of 10/2

Instead of: 1,234,567 bytes
Use: ~1 MB (1 million bytes)

Instead of: 987 requests
Use: ~1K requests (1,000)

3. State Your Assumptions Clearly

Always say:

- "Assuming 100 million daily active users..."
- "Let's assume average photo size is 200 KB..."
- "I'll estimate read-to-write ratio as 100:1..."

4. Work in Standard Units

Storage: Bytes → KB → MB → GB → TB → PB
Time: Seconds → Minutes → Hours → Days → Years
Traffic: QPS (Queries Per Second)

5. Show Your Work

Don't say: "We need 10 TB of storage"
Do say: "100M photos × 200 KB = 20 TB, minus 50% compression = 10 TB"

Essential Numbers to Memorize

Powers of Two (Storage)

```
2^10 = 1,024      ≈ 1 Thousand  = 1 KB
2^20 = 1,048,576  ≈ 1 Million   = 1 MB
2^30 = 1 Billion  ≈ 1 GB
2^40 = 1 Trillion ≈ 1 TB
2^50 = 1 Quadrillion ≈ 1 PB

Pro Tip: Just remember "1024" and multiply!
- 1 KB = ~1,000 bytes
- 1 MB = ~1,000 KB = 1 million bytes
- 1 GB = ~1,000 MB = 1 billion bytes
- 1 TB = ~1,000 GB = 1 trillion bytes
```

Time Conversions

```
1 minute = 60 seconds
1 hour = 3,600 seconds (60 × 60)
1 day = 86,400 seconds (24 × 3,600)
1 month = ~2.5 million seconds (30 × 86,400)
1 year = ~31.5 million seconds (365 × 86,400)

Pro Tip: Remember "86,400 seconds per day"
```

```
- Daily to QPS: Divide by 86,400 (≈ 100K)
- Monthly to QPS: Divide by 2.5M
```

Latency Numbers (Google's Famous List)

```
L1 cache reference:                0.5 ns
Branch mispredict:                 5 ns
L2 cache reference:                7 ns
Main memory reference:             100 ns
Compress 1KB with Snappy:          10 μs
Send 1 KB over 1 Gbps network:     10 μs
Read 1 MB from memory:             250 μs
Round trip within datacenter:      500 μs
Read 1 MB from SSD:                1 ms
Disk seek:                         10 ms
Read 1 MB from network:            10 ms
Read 1 MB from disk:               30 ms
Send packet CA→Netherlands→CA:     150 ms

Takeaways:
- Memory is ~100x faster than SSD
- SSD is ~10x faster than HDD
- Network within datacenter is fast (0.5ms)
- Cross-continent is slow (150ms)
```

Common Data Sizes

```
1 character = 1 byte (ASCII)
1 character = 2-4 bytes (Unicode/UTF-8)

User ID (integer): 8 bytes
Timestamp: 8 bytes
IP address (IPv4): 4 bytes
IP address (IPv6): 16 bytes

Tweet text (280 chars): ~280 bytes
Small image (thumbnail): ~50 KB
Regular image (compressed): ~200 KB
High-res image: ~2 MB
Short video (1 min): ~10 MB
HD video (1 min): ~50 MB

User profile: ~1-2 KB
Email: ~75 KB (average)
Web page: ~2 MB (with assets)
```

Step-by-Step Framework

Step 1: Clarify Scale Requirements

Questions to Ask:

```
Users:
- How many daily active users (DAU)?
- How many monthly active users (MAU)?
- User growth rate?

Usage:
- How many posts/uploads per user per day?
- How many reads per user per day?
- Read-to-write ratio?

Data:
- Average size of user data?
- Average size of content (posts, photos, videos)?
- Data retention period?
```

Step 2: Calculate Traffic

Formula:

```
QPS = Total Operations per Day / 86,400 seconds

Example:
100M tweets per day
= 100,000,000 / 86,400
= 1,157 QPS (round to ~1.2K QPS)

Peak QPS = Average QPS × Peak Factor
Peak Factor typically: 2-5x
= 1.2K × 3 = 3.6K QPS (round to ~4K QPS)
```

Step 3: Calculate Storage

Formula:

```
Total Storage = Number of Items × Size per Item × Time Period

Example:
100M photos per month × 200 KB × 12 months × 10 years
= 100M × 200 KB × 120
= 2.4 PB
```

Step 4: Calculate Bandwidth

Formula:

```
Bandwidth = QPS × Average Response Size

Example:
Read QPS: 10K
Average response: 100 KB
Bandwidth = 10,000 × 100 KB = 1 GB/s
```

Step 5: Calculate Memory (Cache)

Formula:

```
Cache Size = % of Data to Cache × Data Size

Example (80-20 rule):
Total data: 10 TB
Cache 20% hot data: 10 TB × 0.2 = 2 TB
Distributed across 20 servers: 2 TB / 20 = 100 GB per server
```

Traffic Estimation

Calculation Method

Given: Daily Active Users (DAU) and actions per user

Formula:

```
Total Daily Operations = DAU × Operations per User
QPS = Total Daily Operations / 86,400
Peak QPS = QPS × Peak Factor (2-5x)
```

Example 1: Twitter

Given:

- 400M DAU
- Each user tweets 0.5 times per day (average)
- Each user reads 100 tweets per day (scrolling feed)

Calculate:

```
Write Traffic (Tweets):
- Tweets per day: 400M × 0.5 = 200M
- Write QPS: 200M / 86,400 ≈ 2,315 QPS
- Peak write QPS: 2,315 × 3 ≈ 7K QPS

Read Traffic (Timeline):
- Reads per day: 400M × 100 = 40 billion
- Read QPS: 40B / 86,400 ≈ 462K QPS
- Peak read QPS: 462K × 3 ≈ 1.4M QPS

Read-to-Write Ratio: 40B / 200M = 200:1
```

Interview Insight: "This is read-heavy, so we need caching and read replicas"

Example 2: Instagram

Given:

- 500M DAU
- Each user uploads 0.1 photos per day (1 photo per 10 days)
- Each user views 50 photos per day

Calculate:

```
Write Traffic (Photos):
- Photos per day: 500M × 0.1 = 50M
- Write QPS: 50M / 86,400 ≈ 578 QPS
- Peak: ~1.7K QPS

Read Traffic (Feed):
- Views per day: 500M × 50 = 25 billion
- Read QPS: 25B / 86,400 ≈ 289K QPS
- Peak: ~867K QPS

Read-to-Write Ratio: 25B / 50M = 500:1
```

Interview Insight: "Extremely read-heavy, CDN caching is critical"

Example 3: WhatsApp

Given:

- 2 billion MAU
- 50% daily active = 1B DAU
- Each user sends 40 messages per day
- Each user receives 40 messages per day

Calculate:

```
Messages per day: 1B × 40 = 40 billion

Message writes: 40B / 86,400 ≈ 463K QPS
Peak: ~1.4M QPS

With reads (checking for new messages):
- Check every 10 seconds: 1B users × 8,640 checks = 8.64 trillion checks/day
- Check QPS: 8.64T / 86,400 = 100M QPS (too high!)
- Solution: WebSocket connections, push notifications
```

Interview Insight: "Need WebSocket for efficiency, not polling"

Storage Estimation

Calculation Method

Formula:

```
Storage = Number of Objects × Size per Object × Time Period
```

Example 1: YouTube

Given:

- 500 hours of video uploaded per minute
- Average video size: 50 MB per minute
- Store for 10 years

Calculate:

```
Daily uploads:
- 500 hours/min × 60 min/hour × 24 hours = 720,000 hours per day
- 720K hours × 60 min × 50 MB = 2,160,000,000 MB per day
- = 2,160 TB per day = ~2 PB per day

Yearly: 2 PB × 365 = 730 PB per year
10 years: 730 PB × 10 = 7.3 exabytes (7,300 PB)

With multiple resolutions (360p, 720p, 1080p, 4K):
- 4 versions per video
- Total: 7,300 PB × 4 = 29 exabytes

With compression and deduplication:
- Assume 30% savings
- Effective: 29 EB × 0.7 = ~20 exabytes
```

Interview Insight: "This is why YouTube needs Google's infrastructure!"

Example 2: Instagram

Given:

- 500M DAU
- 50M photos uploaded per day
- Average photo: 200 KB
- Store for 10 years

Calculate:

```
Daily storage:
- 50M photos × 200 KB = 10,000,000,000 KB
- = 10 TB per day

Yearly: 10 TB × 365 = 3.65 PB
10 years: 3.65 PB × 10 = 36.5 PB

With thumbnails (3 sizes per photo):
- Original: 200 KB
- Large: 100 KB
- Thumbnail: 20 KB
- Total per photo: 320 KB

Adjusted: 50M × 320 KB = 16 TB per day
10 years: 16 TB × 365 × 10 = 58 PB
```

Interview Insight: "Need S3 or similar object storage, with lifecycle policies"

Example 3: Twitter

Given:

- 400M DAU
- 200M tweets per day
- Average tweet: 280 characters = ~280 bytes
- Store for 5 years
- 20% of tweets have images (~500 KB each)

Calculate:

```
Text storage:
- 200M tweets × 280 bytes = 56 GB per day

Image storage:
```

```
- 200M × 0.2 = 40M images per day
- 40M × 500 KB = 20 TB per day

Total per day: 56 GB + 20 TB ≈ 20 TB
Yearly: 20 TB × 365 = 7.3 PB
5 years: 7.3 PB × 5 = 36.5 PB

With metadata (user_id, timestamp, likes, etc.):
- Additional 100 bytes per tweet
- 200M × 100 bytes = 20 GB per day
- Negligible compared to images

Total: ~37 PB for 5 years
```

Interview Insight: "Text is cheap, media is expensive"

Bandwidth Estimation

Calculation Method

Formula:

```
Bandwidth = QPS × Average Object Size
```

Example 1: Instagram Feed

Given:

- 289K read QPS (from earlier calculation)
- Average feed response: 50 photos × 200 KB thumbnails = 10 MB

Calculate:

```
Incoming bandwidth (uploads):
- 578 QPS (writes) × 200 KB = 115 MB/s

Outgoing bandwidth (feed loads):
- 289K QPS × 10 MB = 2,890 GB/s = 2.8 TB/s

Wait, that's too high! Let's optimize:

With CDN caching (95% cache hit ratio):
- Only 5% hits origin: 2.8 TB/s × 0.05 = 140 GB/s
- CDN bandwidth: 2.8 TB/s (distributed globally)

Actual Instagram approach:
- Serve thumbnails first (150 KB instead of 200 KB)
```

```
- Lazy load full images
- Reduced to ~1 TB/s from CDN
```

Interview Insight: "CDN is essential, not optional"

Example 2: YouTube

Given:

- 1 billion video views per day
- Average video: 10 minutes at 720p = 100 MB

Calculate:

```
Views per second: 1B / 86,400 ≈ 11,574 QPS

Bandwidth: 11,574 × 100 MB = 1,157 GB/s = 1.1 TB/s

Peak (3x): 3.3 TB/s

With CDN (80% cache hit):
- Origin: 3.3 TB/s × 0.2 = 660 GB/s
- CDN: 3.3 TB/s × 0.8 = 2.6 TB/s

Optimization with adaptive bitrate:
- Serve lower quality for slow connections
- Average drops to 50 MB per view
- New bandwidth: 578 GB/s origin
```

Memory/Cache Estimation

The 80-20 Rule (Pareto Principle)

Key Insight: 20% of data generates 80% of traffic

Cache Strategy: Cache the hot 20%, serve 80% of requests

Example 1: Twitter Timeline Cache

Given:

- 400M DAU
- Each timeline: 50 tweets × 500 bytes = 25 KB
- Cache 20% of active users

Calculate:

```
Users to cache: 400M × 0.2 = 80M users
Cache size: 80M × 25 KB = 2,000 GB = 2 TB

Distributed across servers:
- 20 cache servers × 100 GB RAM each = 2 TB total
- Cost: ~$2,000/month (AWS ElastiCache)

Cache hit ratio achieved: ~80%
Database queries saved: 80% of 462K QPS = 370K QPS
```

Interview Insight: "Caching 20% of users saves 80% of database queries"

Example 2: Netflix Thumbnails

Given:

- 10,000 videos in catalog
- Each thumbnail: 50 KB
- Cache most popular 1,000 videos (10%)

Calculate:

```
Cache size: 1,000 videos × 50 KB = 50 MB

With 50% cache hit ratio:
- 11,574 QPS × 0.5 = 5,787 QPS to origin
- Saves significant bandwidth and latency

Even full catalog cache:
- 10,000 × 50 KB = 500 MB
- Tiny! Cache everything!
```

Interview Insight: "Sometimes you can cache the entire dataset"

Real-World Examples

Example 1: Design Twitter

Step 1: Clarify Requirements

```
Assumptions:
- 400M DAU
- 200M tweets per day (0.5 tweets per user)
- Each user views 100 tweets per day
- Average tweet: 280 characters = 280 bytes
```

```
- 20% tweets have images (500 KB)
- Store for 5 years
```

Step 2: Traffic Estimation

```
Writes (Tweets):
200M tweets/day ÷ 86,400 = 2,315 QPS
Peak: 2,315 × 3 = ~7K QPS

Reads (Timeline):
400M users × 100 tweets = 40B reads/day
40B ÷ 86,400 = 462,963 QPS ≈ 463K QPS
Peak: 463K × 3 = ~1.4M QPS

Read-to-Write Ratio: 463K / 2.3K = 200:1
```

Step 3: Storage Estimation

```
Daily storage:
- Text: 200M × 280 bytes = 56 GB
- Images: 200M × 0.2 × 500 KB = 20 TB
- Total: ~20 TB/day

5 years: 20 TB × 365 × 5 = 36.5 PB

With metadata and indexes (+20%):
Total: 36.5 PB × 1.2 = ~44 PB
```

Step 4: Bandwidth Estimation

```
Incoming:
- 2.3K QPS × 280 bytes = 644 KB/s (text)
- Images: 40M/day ÷ 86,400 = 463 QPS
- 463 QPS × 500 KB = 231 MB/s
- Total incoming: ~232 MB/s

Outgoing:
- 463K QPS × 280 bytes = 130 MB/s (text)
- Images: 463K × 0.2 × 200 KB (thumbnails) = 18.5 GB/s
- Total: ~19 GB/s

With CDN (95% cache hit):
- Origin: 19 GB/s × 0.05 = 950 MB/s
- CDN: 19 GB/s × 0.95 = 18 GB/s
```

Step 5: Memory Estimation

```
Cache 20% of active user timelines:
- 400M × 0.2 = 80M users
- Each timeline: 50 tweets × 280 bytes = 14 KB
- Total: 80M × 14 KB = 1.1 TB

Cache servers needed:
- 1.1 TB ÷ 100 GB per server = 11 servers
- Round up to 20 servers for redundancy
```

Summary for Interview:

```
Traffic: ~7K write, ~1.4M read (peak)
Storage: ~44 PB for 5 years
Bandwidth: ~19 GB/s (950 MB/s with CDN)
Cache: ~1.1 TB across 20 servers
Database: Cassandra (write-heavy), PostgreSQL (users), Redis (cache)
```

Example 2: Design Instagram

Step 1: Requirements

```
- 500M DAU
- 50M photos uploaded per day (0.1 per user)
- Each user views 50 photos per day
- Photo: 2 MB original, 200 KB feed, 20 KB thumbnail
- Store for 10 years
```

Step 2: Traffic

```
Uploads: 50M ÷ 86,400 = 578 QPS (peak: 1.7K QPS)
Views: 500M × 50 = 25B/day
Views QPS: 25B ÷ 86,400 = 289K QPS (peak: 867K QPS)
Ratio: 500:1 (extremely read-heavy)
```

Step 3: Storage

```
Per photo:
- Original: 2 MB
- Feed size: 200 KB
- Thumbnail: 20 KB
```

```
- Total: 2.22 MB per photo

Daily: 50M × 2.22 MB = 111 TB
Yearly: 111 TB × 365 = 40.5 PB
10 years: 405 PB

With compression (save 30%):
Actual: 405 PB × 0.7 = ~283 PB
```

Step 4: Bandwidth

```
Upload: 578 QPS × 2 MB = 1.2 GB/s
Download: 289K QPS × 200 KB = 57.8 GB/s

With CDN (90% cache hit):
- Origin: 57.8 GB/s × 0.1 = 5.8 GB/s
- CDN: 57.8 GB/s × 0.9 = 52 GB/s
```

Step 5: Cache

```
Cache hot photos (20% of last 30 days):
- 50M/day × 30 days = 1.5B photos
- 1.5B × 0.2 = 300M hot photos
- 300M × 200 KB = 60 TB

Distributed: 60 TB ÷ 100 servers = 600 GB each
```

Example 3: Design WhatsApp

Step 1: Requirements

```
- 2B MAU, 1B DAU (50% active daily)
- Each user sends 40 messages/day
- Each user receives 40 messages/day
- Average message: 100 bytes
- Store for 1 year
```

Step 2: Traffic

```
Messages per day: 1B × 40 = 40B
Message QPS: 40B ÷ 86,400 = 462K QPS
Peak: 462K × 3 = 1.4M QPS
```

```
This is WRITE-heavy (send/receive are both writes)
```

Step 3: Storage

```
Daily: 40B × 100 bytes = 4 TB
Yearly: 4 TB × 365 = 1.46 PB

With metadata (timestamp, read status, etc.):
- +50 bytes per message
- 40B × 150 bytes = 6 TB/day
- Yearly: 2.19 PB

With images/videos (10% of messages):
- 40B × 0.1 = 4B media messages
- Average: 1 MB per media
- 4B × 1 MB = 4 PB/day
- Yearly: 4 PB × 365 = 1,460 PB = 1.46 exabytes
```

Step 4: Bandwidth

```
Text: 462K QPS × 100 bytes = 46 MB/s
Media: 462K × 0.1 × 1 MB = 46 GB/s
Total: ~46 GB/s

Note: This is why WhatsApp compresses images aggressively!
With compression (80% reduction):
Actual: 46 GB/s × 0.2 = ~9 GB/s
```

Example 4: Design Uber

Step 1: Requirements

```
- 50M daily rides
- Each ride: 1 driver, 1 rider
- Location updates every 3 seconds during ride
- Average ride: 15 minutes
- Store location history for 1 year
```

Step 2: Traffic

```
Location Updates:
- Updates per ride: (15 min × 60 sec) / 3 sec = 300 updates
```

```
- Updates per day: 50M rides × 300 × 2 (driver + rider) = 30B updates
- QPS: 30B ÷ 86,400 = 347K QPS
- Peak (during rush hour, 3x): ~1M QPS

This is WRITE-HEAVY (real-time location tracking)
```

Step 3: Storage

```
Per location update:
- User ID: 8 bytes
- Latitude: 8 bytes
- Longitude: 8 bytes
- Timestamp: 8 bytes
- Total: 32 bytes

Daily: 30B × 32 bytes = 960 GB per day
Yearly: 960 GB × 365 = 350 TB

With trip metadata (pickup, dropoff, fare, etc.):
- Additional 1 KB per trip
- 50M × 1 KB = 50 GB per day
- Yearly: 18 TB

Total: ~368 TB per year
```

Step 4: Memory (Real-time Matching)

```
Active drivers at any time:
- Assume 5M drivers online (10% of daily drivers)
- Each driver location: 32 bytes
- Total: 5M × 32 bytes = 160 MB

Cache all active drivers in memory:
- Easily fits in single Redis instance!
- Use Redis Geo data structure (GEOADD, GEORADIUS)

Active riders:
- 5M riders waiting (matching period)
- Total: 160 MB

Combined: 320 MB (tiny!)
```

Quick Calculation Shortcuts

Shortcut 1: Daily to QPS

Formula: `Daily Requests ÷ 100K ≈ QPS`

Why: 86,400 ≈ 100,000 (close enough for estimates)

Examples:

```
10M requests/day ÷ 100K = 100 QPS
1B requests/day ÷ 100K = 10K QPS
100B requests/day ÷ 100K = 1M QPS
```

Shortcut 2: Monthly to Daily

Formula: `Monthly ÷ 30 = Daily`

Example:

```
3B requests/month ÷ 30 = 100M requests/day
100M ÷ 100K = 1K QPS
```

Shortcut 3: Storage Growth

Formula: `Daily × 365 × Years = Total`

Example:

```
10 TB/day × 365 × 5 = 18,250 TB = ~18 PB
Round to 20 PB for safety margin
```

Shortcut 4: Bandwidth from QPS

Formula: `QPS × Object Size`

Example:

```
10K QPS × 100 KB = 1,000,000 KB/s = 1 GB/s
Round to 1 GB/s for simplicity
```

Shortcut 5: 80-20 Rule for Caching

Formula: `Total Data × 0.2 = Cache Size`

Example:

```
100 TB total data
Cache: 100 TB × 0.2 = 20 TB
Achieves ~80% cache hit ratio
```

Common Estimation Patterns

Pattern 1: Social Media (Twitter, Instagram)

Characteristics:

- Read-heavy (100:1 to 1000:1 ratio)
- Small writes (text, metadata)
- Large reads (media files)

Formula:

```
Write QPS = DAU × Posts per User / 86,400
Read QPS = DAU × Feed Loads × Items per Load / 86,400
Storage = Posts per Day × Size × Retention Period
```

Pattern 2: Messaging (WhatsApp, Telegram)

Characteristics:

- Balanced read/write (1:1 ratio)
- Small messages (bytes to KB)
- High frequency (billions per day)

Formula:

```
Message QPS = DAU × Messages per User / 86,400
Storage = Messages per Day × Size × Retention
Real-time Connections = DAU (for WebSocket)
```

Pattern 3: Video Streaming (YouTube, Netflix)

Characteristics:

- Extremely read-heavy (1000:1)
- Large files (GB per video)
- CDN is mandatory

Formula:

```
Upload QPS = Uploads per Day / 86,400
View QPS = Views per Day / 86,400
Storage = Videos per Day × Size × Retention
Bandwidth = View QPS × Bitrate / 8 (bits to bytes)
```

Pattern 4: E-Commerce (Amazon, eBay)

Characteristics:

- Read-heavy (browse > buy)
- Need ACID (transactions)
- Spike during sales events

Formula:

```
Browse QPS = DAU × Page Views / 86,400
Purchase QPS = Orders per Day / 86,400
Storage = Products × Size + Orders × Size
Peak Factor = 10x (during Black Friday)
```

Interview Example Walkthroughs

Walkthrough 1: "Design a News Feed"

Interviewer: "Design a news feed like Facebook. 1 billion users."

You:

```
"Let me estimate the scale:

Assumptions:
- 1B total users, 500M DAU (50% active)
- Each user posts 0.5 times per day
- Each user refreshes feed 10 times per day
- Each feed load shows 20 posts
- Average post: 1 KB (text + metadata)
- Store for 2 years

Traffic:
Posts: 500M × 0.5 = 250M/day ÷ 100K = 2.5K write QPS
Reads: 500M × 10 × 20 = 100B reads/day ÷ 100K = 1M read QPS
Ratio: 400:1 (very read-heavy)

Storage:
Posts: 250M/day × 1 KB = 250 GB/day
2 years: 250 GB × 365 × 2 = 182 TB
```

```
Round to 200 TB with indexes

Bandwidth:
Outgoing: 1M QPS × 1 KB = 1 GB/s

Cache:
Cache 20% of 500M users' feeds: 100M users
Each feed: 20 posts × 1 KB = 20 KB
Cache size: 100M × 20 KB = 2 TB across 20 servers

Database:
- PostgreSQL/MySQL for users (ACID)
- Cassandra for posts (write-heavy, 2.5K QPS)
- Redis for feed cache (1M read QPS × 80% = 800K QPS served from cache)
"
```

Walkthrough 2: "Design a URL Shortener"

Interviewer: "Design bit.ly. 100M URLs created per month."

You:

```
"Let me calculate the scale:

Assumptions:
- 100M new URLs per month
- 10B redirects per month (100:1 read-to-write)
- Average URL: 200 bytes
- Store for 10 years

Traffic:
Writes: 100M/month ÷ 30 ÷ 100K = 33 QPS (round to 40 QPS)
Reads: 10B/month ÷ 30 ÷ 100K = 3,333 QPS (round to 4K QPS)
Peak: 40 write QPS, 12K read QPS (3x)

Storage:
URLs: 100M × 12 × 10 = 12B URLs
Size: 12B × 200 bytes = 2.4 TB
With analytics: +6 TB (assume 10 clicks per URL)
Total: ~9 TB over 10 years

Cache:
Cache hot URLs (20% of last 30 days):
- 100M/month = 3.3M/day
- 3.3M × 30 days = 100M URLs
- 100M × 0.2 = 20M hot URLs
- 20M × 200 bytes = 4 GB
This is tiny! Cache everything recent!

Bandwidth:
```

```
Incoming: 40 QPS × 200 bytes = 8 KB/s
Outgoing: 4K QPS × 200 bytes = 800 KB/s

Database:
- PostgreSQL (simple, ACID, 4K QPS easily handled)
- Redis cache (80% hit ratio, < 1ms latency)
- Cassandra for analytics (time-series clicks)
"
```

Walkthrough 3: "Design Dropbox"

Interviewer: "Design file sync service. 50M users."

You:

```
"Let me estimate:

Assumptions:
- 50M registered users, 10M DAU (20% active)
- Each user has 200 files (average)
- Average file: 1 MB
- Each user syncs 5 files per day

Traffic:
Syncs: 10M × 5 = 50M syncs/day ÷ 100K = 500 QPS
Peak: 500 × 3 = 1.5K QPS

Storage:
Total files: 50M users × 200 files = 10 billion files
Total storage: 10B × 1 MB = 10 PB

With 3 replicas (redundancy):
Total: 10 PB × 3 = 30 PB

Bandwidth:
Upload/Download: 500 QPS × 1 MB = 500 MB/s
Peak: 1.5 GB/s

Metadata cache:
- File paths, versions, permissions
- 10B files × 1 KB metadata = 10 TB
- Cache 20%: 2 TB across servers
"
```

Quick Reference Cheat Sheet

Time Conversions

```
1 second   = 1,000 ms
1 minute   = 60 seconds
1 hour     = 3,600 seconds
1 day      = 86,400 seconds (~100K for rough estimates)
1 month    = 2,592,000 seconds (~2.5M)
1 year     = 31,536,000 seconds (~30M)
```

Storage Conversions

```
1 KB  = 1,024 bytes        (~1 thousand)
1 MB  = 1,024 KB           (~1 million bytes)
1 GB  = 1,024 MB           (~1 billion bytes)
1 TB  = 1,024 GB           (~1 trillion bytes)
1 PB  = 1,024 TB           (~1 quadrillion bytes)
1 EB  = 1,024 PB           (~1 quintillion bytes)
```

QPS Estimation Shortcuts

```
Daily Operations → QPS:
1M/day     = 10 QPS
10M/day    = 100 QPS
100M/day   = 1K QPS
1B/day     = 10K QPS
10B/day    = 100K QPS
100B/day   = 1M QPS
```

Typical Ratios

```
Read-to-Write Ratios:
- Social media: 100:1 to 1000:1
- Messaging: 1:1
- E-commerce: 10:1 to 100:1
- Analytics: Write-heavy (opposite)

Cache Hit Ratios:
- Well-designed: 80-90%
- Average: 70-80%
- Poor: < 70%

Peak to Average:
- Normal: 2-3x
- Social (viral): 5-10x
- E-commerce (Black Friday): 10-20x
```

Storage Per Object

```
User profile: 1-2 KB
Tweet/Post: 280 bytes - 1 KB
Email: 50-100 KB
Small image: 50-100 KB
Medium image: 200-500 KB
Large image: 1-3 MB
Short video (1 min): 10-50 MB
HD video (1 min): 50-100 MB
```

Interview Tips & Tricks

Tip 1: Start with Round Numbers

Example:

```
Interviewer: "500 million daily active users"
You: "So roughly 500M DAU, I'll round to 500M for calculations"

Interviewer: "Each user posts 2.3 times per day"
You: "I'll approximate that as 2 posts per day"
```

Why: Makes math easier, shows you understand precision isn't critical

Tip 2: Show Units in Every Step

Bad:

```
100 × 200 = 20,000
20,000 × 365 = 7,300,000
```

Good:

```
100M photos × 200 KB = 20,000,000,000 KB = 20 TB per day
20 TB per day × 365 days = 7,300 TB = 7.3 PB per year
```

Tip 3: Sanity Check Your Numbers

After calculating, ask yourself:

- "Does this make sense?"

- "Is 1 PB reasonable for Instagram photos? Yes!"
- "Is 10 EB reasonable for Twitter text? No, that's too high!"

Common Sanity Checks:

```
Twitter text storage should be < 100 TB (tiny)
Instagram photo storage should be > 10 PB (huge)
WhatsApp messages should be > 1 PB but < 10 PB
YouTube videos should be > 1 EB (massive)
```

Tip 4: Mention Optimizations

After calculations, always mention:

```
Storage optimization:
"With compression, we can reduce this by 30-50%"

Bandwidth optimization:
"With CDN caching at 90% hit ratio, origin only serves 10%"

Memory optimization:
"Using the 80-20 rule, we only need to cache 20% of data"
```

Tip 5: Compare to Real Systems

Say things like:

```
"This is similar to Twitter's scale"
"Instagram actually uses ~100 PB of storage"
"Netflix serves petabytes per day from CDN"
"WhatsApp handles billions of messages"
```

Common Mistakes to Avoid

❌ Mistake 1: Being Too Precise

Bad: "387,654,321 requests per day"
Good: "~400 million requests per day"

Why: Precision gives false confidence, rough numbers are sufficient

❌ Mistake 2: Forgetting Peak Traffic

Bad: "10K QPS average"
Good: "10K QPS average, 30K QPS peak (3x factor)"

Why: Systems must handle peak load, not average

❌ Mistake 3: Not Showing Units

Bad: "Storage is 100"
Good: "Storage is 100 TB"

Why: Interviewer doesn't know if you mean MB, GB, or TB

❌ Mistake 4: Ignoring Media

Bad: "Twitter stores 1 TB of tweets"
Good: "Twitter stores 50 GB of text + 10 PB of images"

Why: Media dominates storage, don't ignore it

❌ Mistake 5: Forgetting Replication

Bad: "Need 10 TB storage"
Good: "Need 10 TB × 3 replicas = 30 TB storage"

Why: Replication is standard for reliability

Practice Problems

Problem 1: Design Google Photos

Given:

- 1B users, 500M DAU
- Each user uploads 5 photos per day
- Each photo: 3 MB
- Each user views 50 photos per day
- Store for lifetime (20 years)

Your turn! Calculate:

1. Write QPS?
2. Read QPS?
3. Total storage for 20 years?
4. Bandwidth required?
5. Cache size?

▶ Click for Answer

```
Traffic:
- Uploads: 500M × 5 = 2.5B/day ÷ 100K = 25K write QPS
- Views: 500M × 50 = 25B/day ÷ 100K = 250K read QPS
- Peak: 75K write, 750K read QPS

Storage:
- Daily: 2.5B × 3 MB = 7.5 PB/day
- Yearly: 7.5 PB × 365 = 2,737 PB = 2.7 EB
- 20 years: 2.7 EB × 20 = 54 EB
- With compression (50%): 27 EB

Bandwidth:
- Upload: 25K QPS × 3 MB = 75 GB/s
- Download: 250K QPS × 3 MB = 750 GB/s
- With CDN (95% cache): 37.5 GB/s origin

Cache:
- Hot photos (20% of last 90 days):
- 2.5B × 90 × 0.2 = 45B photos
- 45B × 3 MB = 135 PB (too large to cache originals!)
- Cache thumbnails instead: 45B × 200 KB = 9 PB
- Still large! Cache metadata only: 45B × 1 KB = 45 TB
```

Problem 2: Design Spotify

Given:

- 400M users, 100M DAU
- Each user streams 10 songs per day
- Each song: 5 MB (compressed)
- Catalog: 100M songs
- Store catalog forever

Calculate: Traffic, Storage, Bandwidth, Cache

▶ Click for Answer

```
Traffic:
- Streams: 100M × 10 = 1B/day ÷ 100K = 10K QPS
- Peak: 30K QPS

Storage:
- Catalog: 100M songs × 5 MB = 500 TB
- With multiple qualities: 500 TB × 3 = 1.5 PB

Bandwidth:
- Download: 10K QPS × 5 MB = 50 GB/s
- With CDN (90% cache): 5 GB/s origin
```

```
Cache:
- Hot songs (20% of catalog): 100M × 0.2 = 20M songs
- 20M × 5 MB = 100 TB
- Distributed: 100 TB ÷ 100 servers = 1 TB each
```

The Universal Template

Use this template for ANY system design question:

```
1. TRAFFIC (QPS):
   Write QPS = Writes per Day ÷ 100K
   Read QPS = Reads per Day ÷ 100K
   Peak QPS = Average × 3
   Ratio = Reads / Writes

2. STORAGE:
   Daily = Items per Day × Size per Item
   Total = Daily × 365 × Years
   With Replication = Total × 3

3. BANDWIDTH:
   Incoming = Write QPS × Object Size
   Outgoing = Read QPS × Object Size
   With CDN = Outgoing × (1 - Cache Hit Ratio)

4. MEMORY (Cache):
   Cache Size = Total Data × 0.2 (80-20 rule)
   Servers = Cache Size ÷ 100 GB
   Cache Hit Ratio = ~80%

5. DATABASE:
   If Read-Heavy → SQL + Read Replicas + Cache
   If Write-Heavy → Cassandra/NoSQL
   If < 1 TB → Single SQL instance
   If > 10 TB → Sharding or NoSQL
```

Real Numbers from Production Systems

Twitter (Actual)

```
DAU: 400M
Tweets/day: 500M
QPS: ~6K write, ~300K read
Storage: ~100 PB
Bandwidth: ~1 TB/s (with CDN)
```

Instagram (Actual)

```
DAU: 500M
Photos/day: 95M
QPS: ~1K write, ~500K read
Storage: ~100 PB
Cache: Redis Cluster (50+ nodes)
```

WhatsApp (Actual)

```
DAU: 2B
Messages/day: 100B
QPS: ~1M write, ~1M read
Storage: ~10 PB
Erlang servers: 1000+
```

YouTube (Actual)

```
Videos watched/day: 5B
Upload: 500 hours/minute
Storage: ~1 EB (yes, exabyte!)
Bandwidth: ~1 PB/s globally
CDN: Critical infrastructure
```

Technology Selection Based on Scale Estimates

Database Selection Framework

PostgreSQL/MySQL - When Numbers Justify It

Use PostgreSQL when your calculations show:

```
Storage: < 10 TB
Write QPS: < 10,000
Read QPS: < 100,000 (with 5-10 read replicas)
Need: ACID transactions, complex queries, JOINs
```

Real Examples from This Guide:

TinyURL:

```
✓ Storage: 8 TB over 10 years (< 10 TB threshold)
✓ Write QPS: 40 (far below 10K limit)
✓ Read QPS: 4K (easily handled with 2-3 replicas)
✓ Needs: ACID for URL creation (prevent duplicates)

Justification:
"PostgreSQL handles our scale easily. With 8 TB storage, we're well
under
the 10-16 TB single instance limit. Our 40 write QPS is 0.4% of
PostgreSQL's
10K capacity. With 3 read replicas, we can serve 30K QPS, but we only
need
4K. PostgreSQL provides ACID guarantees we need for URL creation."

Configuration:
- 1 master (r5.2xlarge): 8 vCPU, 64 GB RAM
- 3 read replicas for HA
- Cost: ~$800/month
```

Pastebin:

```
✓ Storage: 21 TB over 5 years
✓ Write QPS: 115
✓ Read QPS: 12K
✓ Features: TTL (auto-delete expired pastes)

Justification:
"21 TB fits in sharded PostgreSQL (3 shards × 7 TB each) or single
instance
with cleanup. 115 write QPS is 1% of capacity. 12K read QPS handled by
5 replicas. PostgreSQL's TTL support (DELETE WHERE created_at < NOW() -
INTERVAL '24 hours') perfectly matches our expire requirement."

Configuration:
- Option 1: 1 master + 5 read replicas (~$1,500/month)
- Option 2: 3-shard setup (~$2,000/month) for future growth
```

Cassandra/NoSQL - When Numbers Demand Scale

Use Cassandra when your calculations show:

```
Storage: > 10 TB (need horizontal scaling)
Write QPS: > 10,000 (need distributed writes)
OR
Storage: > 100 TB (PostgreSQL sharding too complex)
Data Pattern: Time-series, append-heavy
Consistency: Eventual acceptable
```

Real Examples from This Guide:

Twitter Tweets:

```
✗ Storage: 44 PB (4,400x PostgreSQL limit!)
✓ Write QPS: 7,000 (approaches PostgreSQL limit)
✓ Pattern: Time-series (tweets sorted by timestamp)
✓ Consistency: Eventual OK (tweets appear within 1 sec)

Justification:
"44 PB storage requires distributed database. PostgreSQL would need
44,000 GB ÷ 16 TB = 2,750 shards - impossible to manage. Cassandra
scales linearly: 44 PB ÷ 2 TB per node = 22,000 nodes. Each node adds
100 write QPS, giving us 2.2M QPS capacity for our 7K need. Cassandra's
write-optimized LSM tree perfect for tweet insertion pattern."

Configuration:
- 22,000 nodes (in reality, fewer due to data lifecycle)
- Each: i3.2xlarge (8 vCPU, 61 GB RAM, 1.9 TB NVMe)
- Replication factor: 3
- Cost: ~$300K/month (at scale pricing)
```

WhatsApp Messages:

```
✗ Storage: 1.46 EB = 1,460 PB (impossible for PostgreSQL)
✓ Write QPS: 1.4 million (140x PostgreSQL limit!)
✓ Pattern: Append-only messages
✓ Consistency: Eventual OK for message delivery

Justification:
"1.4M write QPS requires massive distribution. PostgreSQL max 10K writes
= would need 140 masters! Cassandra: 1.4M ÷ 100 per node = 14,000 nodes
minimum. Storage: 1.46 EB ÷ 2 TB = 750,000 nodes. In practice, messages
deleted after 30 days, so 50K nodes at ~30 PB. Cassandra's peer-to-peer
architecture enables this scale."

Configuration:
- 50,000 nodes (with 30-day retention)
- Distributed globally (20 regions × 2,500 nodes)
- Cost: ~$700K/month
```

Uber Location Updates:

```
✗ Storage: 2 PB (requires 125 PostgreSQL shards)
✓ Write QPS: 1 million (100x PostgreSQL limit)
✓ Pattern: Constant location writes every 3 seconds
```

```
✓ Query: Time-range lookups (trip replay)

Justification:
"1M location writes/sec impossible for PostgreSQL (10K max). Cassandra:
1M ÷ 1K per node = 1,000 nodes. Storage: 2 PB ÷ 2 TB = 1,000 nodes.
Perfect match! Cassandra's write path (MemTable → SSTable) optimized
for constant writes. Time-series partition key enables efficient trip
replay queries."

Configuration:
- 1,000 nodes globally
- Partition key: (driver_id, date)
- Clustering key: timestamp
- Cost: ~$140K/month
```

MongoDB - The Middle Ground

Use MongoDB when your calculations show:

```
Storage: 1-100 TB (too big for single PostgreSQL, not big enough for
Cassandra)
Write QPS: 1,000-50,000 (moderate)
Schema: Flexible/varying structure
Need: Document queries, indexes
```

Reddit Example:

```
✓ Storage: 250 TB (would need 16 PostgreSQL shards)
✓ Write QPS: 30,000 (3x PostgreSQL limit)
✓ Schema: Posts vary (text, images, videos, polls)
✓ Queries: Complex (find by subreddit, user, time)

Justification:
"250 TB too large for single PostgreSQL (need 16 shards). But 30K write
QPS doesn't justify Cassandra's complexity. MongoDB offers middle
ground:
16 shards × 15 TB each. Flexible schema handles varying post types.
Built-in sharding simpler than manual PostgreSQL sharding."

Configuration:
- 16 shards × 3 replica sets = 48 nodes
- Each: r5.2xlarge (8 vCPU, 64 GB RAM)
- Cost: ~$9,000/month
```

Cache Technology Selection

Redis - The Default Choice

Use Redis (almost always) when:

```
Cache size: Any (MB to TB)
QPS: Up to 1M per instance
Need: Data structures, persistence, Pub/Sub, TTL
```

Redis Sizing by Numbers:

```
Cache < 1 GB:
- Single Redis instance (r5.large: 16 GB RAM)
- Handles: 100K QPS
- Cost: ~$140/month
- Examples: Rate Limiter (400 MB), Parking Lot (183 MB)

Cache 1-100 GB:
- 2-5 Redis instances (HA cluster)
- Handles: 200-500K QPS
- Cost: ~$280-700/month
- Examples: Reddit (20 GB), Autocomplete (10 GB)

Cache 100 GB - 1 TB:
- Redis Cluster (10-20 nodes)
- Handles: 1-2M QPS
- Cost: ~$1,400-2,800/month
- Examples: Twitter timelines (1.1 TB), Instagram (60 TB needs 600
nodes)

Cache > 1 TB:
- Redis Cluster (50-1000 nodes)
- Handles: 5-50M QPS
- Cost: ~$7,000-140,000/month
- Examples: Instagram hot photos (60 TB), LinkedIn (7.68 TB)
```

Real Examples with Justifications:

TinyURL Cache (2.5 GB):

```
Calculations show:
- Cache size: 20M hot URLs × 125 bytes = 2.5 GB
- QPS: 12K reads (with 95% cache hit)
- Latency requirement: < 10ms

Redis Configuration:
- Single instance: cache.r5.large (13.5 GB RAM)
- Headroom: 11 GB free (440% capacity)
```

```
- Performance: 100K QPS capacity vs 12K need (8x headroom)
- Latency: 0.5-1ms average (meets < 10ms requirement)

Cost Justification:
- Redis: $140/month
- Alternative (Memcached): $130/month (save $10)
- Benefit of Redis: Persistence, TTL, sorted sets
- Decision: Redis (minimal cost difference, more features)
```

Twitter Timeline Cache (1.1 TB):

```
Calculations show:
- Cache size: 80M users × 14 KB timeline = 1.1 TB
- QPS: 1.4M reads
- Cache hit requirement: 80%+ (save DB from 1.1M QPS)

Redis Cluster Configuration:
- 20 nodes × 64 GB RAM = 1.28 TB capacity
- Per node: cache.r5.4xlarge (52 GB usable)
- Each handles: 70K QPS
- Total capacity: 1.4M QPS ✓

Cost Analysis:
- Redis Cluster: $5,600/month
- Saves: Database capacity for 1.1M QPS
- Alternative DB scaling: $50K/month (20x more expensive!)
- ROI: Cache pays for itself 9x over

Configuration Details:
- Partition: 16,384 hash slots across 20 nodes
- Replication: Each node has 1 replica (40 nodes total)
- Eviction: LRU (Least Recently Used)
- Total cost: $11,200/month (with replicas)
```

Rate Limiter Cache (400 MB):

```
Calculations show:
- Active counters: 10M users × 20 bytes = 200 MB
- With sliding window: 200 MB × 2 = 400 MB
- QPS: 36K checks/sec
- Latency requirement: < 1ms (critical!)

Redis Configuration:
- Single instance: cache.r5.large (13.5 GB)
- Memory usage: 400 MB = 3% of capacity
- Operations: INCR, EXPIRE (atomic, fast)
- Performance: Sub-millisecond latency ✓
```

```
Why Redis over Memcached:
- Need atomic INCR (prevent race conditions)
- Need TTL per key (auto-cleanup windows)
- Memcached lacks both features
- Cost: Same ($140/month)
- Decision: Redis (required features, same cost)
```

Memcached - Rare Use Case

Only use Memcached when:

```
- Pure key-value (no data structures needed)
- Multi-threading benefit significant (>1M QPS per instance)
- No persistence needed
- No TTL per-key needed
```

Realistic scenario: < 1% of systems need Memcached over Redis

---

CDN Selection Based on Bandwidth Numbers

When CDN is Mandatory

Use CDN when calculations show:

```
Bandwidth: > 1 GB/s outgoing
Content: Static (images, videos, CSS, JS)
Users: Global distribution
Pattern: Read-heavy (can cache)
```

Cost-Benefit Analysis:

Instagram Example (57 GB/s bandwidth):

```
Without CDN (Origin Only):
- Bandwidth: 57 GB/s × $0.08/GB = $4.56/sec
- Per hour: $16,416
- Per month: $11.82 million

With CDN (90% cache hit):
- Origin: 5.7 GB/s × $0.08/GB = $0.456/sec = $328K/month
- CDN: 51.3 GB/s × $0.02/GB = $1.026/sec = $739K/month
- Total: $1.067 million/month

Savings: $11.82M - $1.067M = $10.75M/month (91% cost reduction!)
```

```
Additional Benefits:
- Latency: 200ms (origin) → 20ms (edge) = 10x faster
- Origin load: 57 GB/s → 5.7 GB/s = 90% reduction
- Global reach: 400+ edge locations vs single origin
- DDoS protection: Included

Decision: CDN absolutely mandatory, not optional
```

Netflix Example (8.3 TB/s bandwidth):

```
Without CDN:
- Cost: 8,300 GB/s × $0.08 = $664/sec = $1.9 billion/month
- Impossible! Can't serve from origin.

With Custom CDN (95% from edge):
- Origin: 415 GB/s × $0.08 = $33.2/sec = $86M/month
- CDN (Open Connect): Custom infrastructure
- Netflix deploys servers in ISP facilities (free transit)
- Estimated cost: $10M/month (vs $1.9B without CDN)

Savings: Must use CDN to be economically viable

Netflix's Approach:
- Open Connect: Custom CDN appliances
- Pre-populate hot content (new releases)
- 90% of traffic served within ISP network (no internet transit)
- Result: Streams 1+ billion hours/day profitably
```

When CDN is Optional

Small bandwidth (< 100 MB/s):

```
Rate Limiter (36 KB/s):
- Cost without CDN: $0.0000288/sec = $75/month
- Cost with CDN: $50/month (minimum) + setup complexity
- Decision: Skip CDN, not worth it

Parking Lot (1 MB/s):
- Cost without CDN: $0.08/sec = $2,100/month
- Cost with CDN: $400/month
- Savings: $1,700/month
- But: Local system (single facility), CDN adds latency
- Decision: Skip CDN for local-only service
```

Message Queue Selection

Kafka - For High Throughput

Use Kafka when calculations show:

```
Events: > 10K/sec
Need: Durability (no message loss)
Need: Replay capability
Pattern: Multiple consumers
```

Real Examples:

Twitter Fanout (7K tweets/sec):

```
Requirements:
- 7K tweet events/sec
- Each tweet → fanout to avg 500 followers
- Total: 7K × 500 = 3.5M fanout operations/sec
- Need durability (can't lose tweets)
- Need ordering per user

Kafka Configuration:
- 10 brokers (each handles 100K msg/sec)
- 3.5M ÷ 100K = 35 brokers minimum
- With replication (factor 3): 105 brokers
- Cost: ~$15,000/month

Why Kafka over alternatives:
- RabbitMQ: Max 50K/sec/node (need 70 nodes, similar cost)
- SQS: $0.40/million = 3.5M × 86,400 × 30 = $3.6M/month (240x more!)
- Redis Pub/Sub: No durability (lose messages on crash)

Decision: Kafka for durability + performance at scale
```

WhatsApp Messages (1.4M/sec):

```
Requirements:
- 1.4M message events/sec
- Must guarantee delivery
- Need message ordering per conversation
- Global distribution

Kafka Configuration:
- 100 brokers globally (20 regions × 5 brokers)
- Each region: 70K msg/sec
- Replication: 3x within region
- Cost: ~$140K/month
```

```
Alternative (AWS SQS):
- Cost: 1.4M × 86,400 × 30 × $0.40/million = $1.45M/month
- 10x more expensive!
- Kafka justified by cost savings alone
```

SQS/Cloud Queue - For Moderate Load

Use SQS when:

```
Events: < 10K/sec
Pattern: Simple queue (no replay needed)
Need: Managed service (no ops overhead)
```

Cost Comparison:

```
Notification System (36K events/sec):

Kafka:
- 5 brokers × $140/month = $700/month
- Operational overhead: 20 hours/month × $100/hr = $2,000
- Total: $2,700/month

SQS:
- 36K × 86,400 × 30 = 93 billion requests/month
- Cost: 93B × $0.40/million = $37,200/month
- No operational overhead
- Total: $37,200/month

Decision: Kafka (13x cheaper including ops cost)
```

```
Low-volume system (100 events/sec):

Kafka:
- Minimum 3 brokers: $420/month
- Ops overhead: $2,000/month
- Total: $2,420/month

SQS:
- 100 × 86,400 × 30 = 259 million requests/month
- Cost: 259M × $0.40/million = $104/month
- Total: $104/month

Decision: SQS (23x cheaper for low volume)
```

Object Storage Selection

S3/Cloud Storage - For Media Files

Use S3 when calculations show:

```
Storage: > 100 GB of media files
Type: Images, videos, documents
Need: 11-nines durability
Access: Random (not sequential)
```

Cost Analysis with Lifecycle Policies:

Instagram Photos (283 PB):

```
Storage Tiers:
- Hot (< 30 days): 50M × 30 days = 1.5B photos
  Size: 1.5B × 200 KB = 300 TB
  Cost: 300 TB × $23/TB = $6,900/month (S3 Standard)

- Warm (30-90 days): 50M × 60 days = 3B photos
  Size: 3B × 200 KB = 600 TB
  Cost: 600 TB × $12.50/TB = $7,500/month (S3 IA)

- Cold (> 90 days): 50M × 3,560 days = 178B photos
  Size: 178B × 200 KB = 35.6 PB
  Cost: 35.6 PB × $4/TB = $142,400/month (S3 Glacier)

Total monthly cost: $156,800/month
Total storage: 283 PB

Without lifecycle policies:
- All in Standard: 283 PB × $23/TB = $6,509,000/month
- Savings: $6.35M/month (97% reduction!)

Justification:
"98% of photo views are < 90 days old. By using lifecycle policies,
we move 126 PB to warm and 35.6 PB to cold storage. This reduces
costs by 97% while maintaining instant access to hot photos."
```

Netflix Catalog (300 TB):

```
All videos in S3 Standard:
- 300 TB × $23/TB = $6,900/month
- Need instant access for streaming
- Can't use lifecycle (all content frequently accessed)
```

```
Additional copies:
- 4 resolutions × 300 TB = 1.2 PB
- Cost: 1.2 PB × $23/TB = $27,600/month

CDN pre-population:
- Popular content (20%): 240 GB at each edge
- 100 edges × 240 GB = 24 TB
- Cost: Included in CDN edge cache

Total S3 cost: $27,600/month for origin storage
Reasonable for $30B/year revenue company
```

Complete Technology Stack Examples

Small Scale (< 1M users, < 10K QPS)

Example: Parking Lot System

```
Numbers:
- DAU: 5M vehicles
- QPS: 519 peak
- Storage: 11 TB

Technology Stack:
✓ Database: PostgreSQL (single instance)
  Why: 11 TB fits, 519 QPS trivial
  Instance: db.r5.xlarge (4 vCPU, 32 GB)
  Cost: $350/month

✓ Cache: Redis (single instance)
  Why: 183 MB cache
  Instance: cache.r5.large (13.5 GB)
  Cost: $140/month

✓ Web Servers: 5 instances
  Why: 519 QPS ÷ 100 per server = 5.19
  Instance: t3.medium (2 vCPU, 4 GB) × 5
  Cost: $250/month

✗ CDN: Not needed
  Why: 1 MB/s bandwidth, local facility only

✗ Message Queue: Not needed
  Why: Synchronous flow sufficient

Total Cost: $740/month
Scaling headroom: 10x (can handle 5K QPS)
```

Medium Scale (10-100M users, 10-100K QPS)

Example: Reddit

```
Numbers:
- DAU: 50M
- Read QPS: 88K peak
- Write QPS: 30K peak
- Storage: 250 TB

Technology Stack:
✓ Database: MongoDB (sharded)
  Why: 250 TB too large for single PostgreSQL
       30K write QPS exceeds single instance
  Configuration: 16 shards × 3 replicas = 48 nodes
  Instance: r5.2xlarge per node
  Cost: $9,000/month

✓ Cache: Redis Cluster
  Why: 20 GB cache, 88K read QPS
  Configuration: 5 nodes
  Instance: cache.r5.xlarge per node
  Cost: $700/month

✓ Search: Elasticsearch
  Why: Full-text search on posts/comments
  Configuration: 10 nodes
  Cost: $2,000/month

✓ CDN: CloudFront
  Why: Image content, reduce latency
  Bandwidth: 5 GB/s × 30% CDN-worthy = 1.5 GB/s
  Cost: $1,500/month

✓ Queue: Kafka
  Why: Vote aggregation, comment fanout
  Configuration: 5 brokers
  Cost: $700/month

✓ Web Servers: 50 instances
  Why: 88K QPS ÷ 2K per server = 44 servers
  Instance: c5.2xlarge × 50
  Cost: $6,000/month

Total Cost: $19,900/month
User cost: $0.40 per user per month
Revenue needed: $0.50+ per user for profitability
```

Large Scale (100M-1B users, 100K-1M QPS)

Example: Twitter

```
Numbers:
- DAU: 400M
- Read QPS: 1.4M peak
- Write QPS: 7K peak
- Storage: 44 PB

Technology Stack:
✓ Database (Tweets): Cassandra
  Why: 44 PB storage, write-optimized
  Configuration: 22,000 nodes (in reality ~5K with lifecycle)
  Instance: i3.2xlarge per node
  Cost: $300K/month (negotiated volume pricing)

✓ Database (Users): PostgreSQL
  Why: 800 GB, need ACID
  Configuration: 1 master + 10 read replicas
  Instance: db.r5.4xlarge
  Cost: $2,500/month

✓ Cache: Redis Cluster
  Why: 1.1 TB timeline cache
  Configuration: 20 master + 20 replica = 40 nodes
  Instance: cache.r5.4xlarge per node
  Cost: $11,200/month

✓ Search: Elasticsearch
  Why: Tweet search, trending topics
  Configuration: 100 nodes
  Cost: $20,000/month

✓ CDN: CloudFront
  Why: 19 GB/s bandwidth for media
  Configuration: Global distribution
  Cost: $50,000/month (with 95% cache hit)

✓ Queue: Kafka
  Why: Tweet fanout, timeline generation
  Configuration: 50 brokers globally
  Cost: $7,000/month

✓ Web Servers: 300 instances
  Why: 1.4M QPS ÷ 5K per server = 280
  Instance: c5.4xlarge × 300
  Cost: $75,000/month

✓ Object Storage (S3)
  Why: Image/video storage (20 PB with lifecycle)
  Cost: $50,000/month

Total Cost: $515,700/month = $6.2M/year
```

```
DAU: 400M
Cost per DAU: $1.29/month
Revenue needed: $2+/user/month for profitability
Twitter actual revenue: $5+/user (ads + premium)
```

Extreme Scale (1B+ users, 1M+ QPS)

Example: WhatsApp

```
Numbers:
- DAU: 1B
- Message QPS: 1.4M peak
- Storage: 30 PB (with 30-day retention)
- Connections: 500M concurrent WebSocket

Technology Stack:
✓ Database: Cassandra
  Why: 1.4M write QPS, 30 PB storage
  Configuration: 50,000 nodes globally
  Instance: i3.2xlarge per node
  Cost: $700K/month

✓ Cache: Redis Cluster
  Why: 2.5 TB (presence, sessions, queues)
  Configuration: 200 nodes
  Cost: $28,000/month

✓ WebSocket Servers: 10,000 instances
  Why: 500M connections ÷ 50K per server
  Instance: c5.xlarge × 10,000
  Cost: $150,000/month

✓ Queue: Kafka
  Why: 1.4M message events/sec
  Configuration: 100 brokers (20 regions)
  Cost: $140,000/month

✓ Object Storage: S3
  Why: Media files (aggressive lifecycle)
  Storage: 1 PB (after compression + lifecycle)
  Cost: $23,000/month

✓ Media Processing: GPU instances
  Why: Image/video compression
  Configuration: 1,000 instances
  Cost: $200,000/month

Total Cost: $1.241 million/month = $14.9M/year
DAU: 1B
Cost per DAU: $0.01/month (!!)
```

```
Revenue: Free service (Meta subsidizes)
Actual monetization: Data for ad targeting across Meta products
```

Technology Decision Checklist

Before Recommending Technology, Verify:

Database Selection:

```
□ What is storage requirement? (< 10 TB = PostgreSQL possible)
□ What is write QPS? (< 10K = PostgreSQL possible)
□ What is read QPS? (< 100K = PostgreSQL + replicas possible)
□ Need ACID? (Yes = prefer PostgreSQL)
□ Time-series data? (Yes = consider Cassandra)
□ Storage > 100 TB? (Yes = Cassandra/DynamoDB)
```

Cache Selection:

```
□ What is cache size? (Size Redis cluster accordingly)
□ What is QPS? (1 Redis = 100K QPS)
□ Need data structures? (Yes = Redis, No = Memcached option)
□ Need persistence? (Yes = Redis, No = Memcached option)
□ Need Pub/Sub? (Yes = Redis)
□ Calculate cost: size ÷ 64 GB × $280/month
```

CDN Decision:

```
□ What is bandwidth? (< 100 MB/s = optional, > 1 GB/s = mandatory)
□ Content type? (Static = yes, Dynamic = no)
□ Global users? (Yes = CDN beneficial)
□ Calculate savings: Compare origin vs CDN cost
□ Current: X GB/s × $0.08 = $Y/month
□ With CDN (90% hit): X × 0.1 × $0.08 + X × 0.9 × $0.02 = $Z/month
```

Message Queue:

```
□ What is event rate? (< 1K/sec = SQS, > 10K/sec = Kafka)
□ Need durability? (Yes = Kafka, No = Redis Pub/Sub)
□ Need replay? (Yes = Kafka)
□ Calculate cost: Events/month × $0.40/million (SQS) vs nodes × $140
(Kafka)
```

Interview Scoring Rubric

What Interviewers Look For:

✅ **Structured Approach** (30%)

- Clear methodology
- Step-by-step calculations
- Organized presentation

✅ **Reasonable Assumptions** (20%)

- States assumptions clearly
- Numbers are realistic
- Justifies choices

✅ **Correct Math** (20%)

- Gets order of magnitude right
- Shows work
- Unit conversions correct

✅ **Identifies Bottlenecks** (15%)

- "At 1M QPS, we need caching"
- "44 PB requires distributed storage"
- "This is write-heavy, need Cassandra"

✅ **Mentions Optimizations** (15%)

- CDN for bandwidth
- Caching for reads
- Compression for storage

✅ **Technology Justification** (NEW - Critical!)

- Explains WHY specific tech chosen
- Uses numbers to justify decisions
- Compares alternatives with costs
- Shows scalability path

Example Good Answer:

```
"Based on our calculations:
- 44 PB storage requires distributed DB (PostgreSQL max 16 TB)
- 7K write QPS approaches single instance limit

---

## The Mental Math Trick
```

```
### Multiplication by Powers of 10
```

× 10: Add one zero
× 100: Add two zeros
× 1,000: Add three zeros (1K)
× 1,000,000: Add six zeros (1M)
× 1,000,000,000: Add nine zeros (1B)

```
**Example**:
```

5 MB × 1,000 = 5,000 MB = 5 GB
100 KB × 1,000,000 = 100,000,000 KB = 100 GB

```
### Division by Large Numbers
```

÷ 100: Remove two zeros
÷ 1,000: Remove three zeros
÷ 86,400: Divide by 100,000 (close enough)

```
**Example**:
```

10,000,000 ÷ 100,000 = 100
1,000,000,000 ÷ 100,000 = 10,000

```
---

## Advanced: Server Capacity Estimation

### Typical Server Specs

**Small Instance** (t3.medium):
- 2 vCPU, 4 GB RAM
- Handles: ~500 QPS
- Cost: ~$50/month

**Medium Instance** (m5.xlarge):
- 4 vCPU, 16 GB RAM
- Handles: ~2,000 QPS
```

```
- Cost: ~$150/month

**Large Instance** (m5.4xlarge):
- 16 vCPU, 64 GB RAM
- Handles: ~10,000 QPS
- Cost: ~$600/month

### Calculating Server Count

**Formula**:
```

Servers = (Peak QPS / QPS per Server) × Safety Factor

Example:
Peak: 30K QPS
Server capacity: 2K QPS
Safety factor: 1.5 (50% headroom)

Servers = (30,000 / 2,000) × 1.5 = 22.5
Round up: 25 servers

```


### Database Server Capacity

**PostgreSQL (single instance)**:
- Reads: 10K–50K QPS
- Writes: 1K–10K QPS
- Storage: Up to 16 TB

**Cassandra (per node)**:
- Reads: 10K–50K QPS
- Writes: 10K–100K QPS
- Storage: Up to 8 TB per node

**Redis (single instance)**:
- Reads: 100K–1M QPS
- Writes: 100K–1M QPS
- Storage: Up to 512 GB RAM

---

## Estimation Worksheet Template

**Copy this for every interview**:
```

SYSTEM: _______________

ASSUMPTIONS:

- DAU: ________________
- Operations per user: ________________
- Data size: ________________
- Retention: ________________

TRAFFIC CALCULATIONS:

- Daily operations: DAU × ops = ________________
- Average QPS: daily ÷ 100K = ________________
- Peak QPS: avg × 3 = ________________
- Read/Write ratio: ________________

STORAGE CALCULATIONS:

- Per item size: ________________
- Daily: items/day × size = ________________
- Yearly: daily × 365 = ________________
- Total: yearly × years = ________________
- With replication: total × 3 = ________________

BANDWIDTH CALCULATIONS:

- Incoming: write QPS × size = ________________
- Outgoing: read QPS × size = ________________
- With CDN: outgoing × (1 - hit ratio) = ________________

MEMORY/CACHE CALCULATIONS:

- Hot data (20%): total × 0.2 = ________________
- Cache servers: cache ÷ 100 GB = ________________
- Cache hit ratio: ________________

DATABASE CHOICE:

- Primary: ________________ (because: ________________)
- Cache: ________________
- Analytics: ________________

```


---

## Final Checklist

Before saying "I'm done with calculations", verify:

✅ **Traffic Estimation**
- Calculated write QPS
- Calculated read QPS
- Mentioned peak factor (2-3x)
- Identified if read-heavy or write-heavy
```

```
✅ **Storage Estimation**
- Calculated total storage
- Considered retention period
- Mentioned replication (3x)
- Separated data types (text vs media)

✅ **Bandwidth Estimation**
- Calculated incoming bandwidth
- Calculated outgoing bandwidth
- Mentioned CDN impact

✅ **Memory Estimation**
- Applied 80-20 rule
- Calculated cache size
- Determined number of servers

✅ **Identified Bottlenecks**
- Database: "At 10K write QPS, need sharding"
- Network: "At 1 TB/s, need CDN"
- Memory: "At 10 TB cache, need 100 servers"

---

## Summary: The Golden Rules

1. **Round liberally** - 387M → 400M
2. **State assumptions** - "Assuming X..."
3. **Show your work** - Write calculations down
4. **Use shortcuts** - Daily ÷ 100K = QPS
5. **Apply 80-20 rule** - Cache 20%, serve 80%
6. **Mention peak traffic** - Average × 3
7. **Don't forget media** - Images dominate storage
8. **Include replication** - × 3 for redundancy
9. **Sanity check** - Does 1 EB for Twitter make sense? No!
10. **Mention optimizations** - CDN, compression, caching

---

## References

### Books
1. **"Designing Data-Intensive Applications"** - Martin Kleppmann
2. **"System Design Interview"** - Alex Xu

### Online Resources
1. **System Design Primer** - GitHub (donnemartin)
2. **Google's Latency Numbers** - Jeff Dean
3. **AWS Calculator** - For cost estimates
4. **High Scalability Blog** - Real-world numbers

### Videos
1. **"System Design Course"** - YouTube (Gaurav Sen)
```

```
2. **"Back of Envelope Calculations"** - YouTube (SystemDesignInterview)

---

## Appendix: Pre-Calculated Common Scenarios

### Scenario: 1 Million Users
```

Assumptions: 1M DAU, 10 operations/user/day
Traffic: 10M/day ÷ 100K = 100 QPS
Storage (1 KB per op): 10M × 1 KB = 10 GB/day = 3.6 TB/year
Cache: 1M users × 10 KB = 10 GB
Servers: 5-10 app servers

```

### Scenario: 10 Million Users
```

Assumptions: 10M DAU, 10 operations/user/day
Traffic: 100M/day ÷ 100K = 1K QPS
Storage: 100M × 1 KB = 100 GB/day = 36 TB/year
Cache: 10M × 10 KB = 100 GB
Servers: 10-20 app servers, single DB with replicas

```

### Scenario: 100 Million Users
```

Assumptions: 100M DAU, 10 operations/user/day
Traffic: 1B/day ÷ 100K = 10K QPS
Storage: 1B × 1 KB = 1 TB/day = 365 TB/year
Cache: 100M × 10 KB = 1 TB across 10 servers
Servers: 50-100 app servers, sharded DB or NoSQL

```

### Scenario: 1 Billion Users (Facebook Scale)
```

Assumptions: 1B DAU, 10 operations/user/day
Traffic: 10B/day ÷ 100K = 100K QPS
Storage: 10B × 1 KB = 10 TB/day = 3.6 PB/year
Cache: 200M hot users × 10 KB = 2 TB across 20 servers
Servers: 500-1000 app servers, distributed NoSQL

```
```

```
---

## Top 15 HLD Questions: Detailed Calculations

### 1. Design TinyURL / URL Shortener ⭐⭐⭐

**Requirements**:
```

- 100M new URLs per month
- 100:1 read-to-write ratio
- URLs never expire
- Short URL: 7 characters (a-zA-Z0-9) = 62^7 = 3.5 trillion combinations

```
**Traffic Calculation**:
```

Writes (URL creation):
100M URLs/month ÷ 30 days = 3.3M URLs/day
3.3M ÷ 86,400 = 38 QPS (round to 40 QPS)
Peak: 40 × 3 = 120 QPS

Reads (Redirects):
100:1 ratio → 10B redirects/month
10B ÷ 30 = 333M redirects/day
333M ÷ 86,400 = 3,854 QPS (round to 4K QPS)
Peak: 4K × 3 = 12K QPS

```
**Storage Calculation**:
```

Per URL entry:

- Original URL: 100 bytes (average)
- Short URL: 7 bytes
- Created timestamp: 8 bytes
- User ID: 8 bytes
- Total: ~125 bytes per entry

Over 10 years:

- 100M/month × 12 × 10 = 12 billion URLs
- 12B × 125 bytes = 1.5 TB
- With indexes: 1.5 TB × 1.5 = 2.25 TB

Analytics (click tracking):

- Assume 10 clicks per URL average
- 12B URLs × 10 clicks = 120B click events
- Each event: 50 bytes (URL_id, timestamp, IP, user_agent)
- 120B × 50 bytes = 6 TB
- Total with analytics: 2.25 TB + 6 TB = 8.25 TB

```
**Cache Calculation**:
```

80-20 Rule: 20% URLs get 80% traffic

Hot URLs (recent + popular):

- Recent: Last 30 days = 100M URLs
- Apply 80-20: 100M × 0.2 = 20M hot URLs
- Size: 20M × 125 bytes = 2.5 GB

Decision: Cache ALL recent URLs! Only 2.5 GB needed
Single Redis instance can handle entire cache

```
**Server Estimate**:
```

Database:

- Single PostgreSQL instance handles 4K read QPS easily
- With 5 read replicas: 20K+ QPS capacity
- Well within requirements

Cache:

- 1 Redis instance (16 GB) - plenty of headroom
- Expected cache hit rate: 95%
- Only 5% (200 QPS) hits database

Application servers:

- 4K peak QPS ÷ 500 QPS per server = 8 servers
- With redundancy: 15-20 servers

```
**Interview Summary**:
```

"TinyURL is actually a simple system at this scale:

- Traffic: 4K read QPS, 40 write QPS (manageable)
- Storage: 8 TB over 10 years (tiny!)
- Cache: 2.5 GB (fits in single Redis)
- Database: Single PostgreSQL + read replicas sufficient
- The key optimization is caching hot URLs for 95%+ hit rate"

```

---

### 2. Design Netflix / Video Streaming ⭐⭐⭐

**Requirements**:
```

- 200M subscribers, 100M DAU (50% daily active)
- Each user watches 2 hours per day
- Average video: 1 hour duration
- Multiple qualities: 360p, 720p, 1080p, 4K
- Store catalog forever

```

**Traffic Calculation**:
```

Video plays per day:
100M users × (2 hours / 1 hour per video) = 200M video starts/day
200M ÷ 86,400 = 2,314 QPS (round to 2.5K QPS)
Peak: 2.5K × 3 = 7.5K QPS

Streaming QPS (continuous):
Active streamers at any moment:

- 100M DAU × 2 hours / 24 hours = 8.3M concurrent streams
- Each stream = continuous data flow (not traditional QPS)

API calls (browse, search, metadata):

- Each user: 20 API calls per session
- 100M × 20 = 2B calls/day
- 2B ÷ 86,400 = 23K QPS
- Peak: 70K QPS

```

**Storage Calculation**:
```

Catalog size:

- Assume 10,000 movies/shows
- Average content length: 90 minutes

Per video, multiple qualities:

- 360p: 500 MB (0.5 GB)
- 720p: 1.5 GB
- 1080p: 3 GB
- 4K: 8 GB
- Total per video: 13 GB

Catalog storage:

- 10,000 videos × 13 GB = 130 TB

New content per month:

- Assume 100 new titles per month
- 100 × 13 GB = 1.3 TB per month
- Yearly: 1.3 TB × 12 = 15.6 TB

10-year catalog: 130 TB + (15.6 TB × 10) = 286 TB

User data (watch history, preferences):

- 200M users × 10 KB = 2 TB (negligible)

Total: ~300 TB for catalog

```


**Bandwidth Calculation**:
```

Concurrent streams: 8.3M

Bitrate by quality:

- 360p: 1 Mbps = 125 KB/s
- 720p: 3 Mbps = 375 KB/s
- 1080p: 5 Mbps = 625 KB/s
- 4K: 25 Mbps = 3.125 MB/s

Average (assume 40% HD, 40% SD, 20% 4K):

- (0.4 × 625 KB/s) + (0.4 × 375 KB/s) + (0.2 × 3.125 MB/s)
- = 250 + 150 + 625 = 1,025 KB/s ≈ 1 MB/s per stream

Total bandwidth:

- 8.3M streams × 1 MB/s = 8.3 TB/s

With CDN (Netflix uses 95% CDN):

- Origin: 8.3 TB/s × 0.05 = 415 GB/s
- CDN: 8.3 TB/s × 0.95 = 7.8 TB/s

CDN edge locations: 100+ globally
Per edge: 7.8 TB/s ÷ 100 = 78 GB/s

```
**Cache Calculation**:
```

Popular content (80-20 rule):

- 10,000 videos × 0.2 = 2,000 hot videos
- 2,000 × 13 GB = 26 TB

CDN cache per edge:

- Distribute 26 TB across 100 edges
- 26 TB ÷ 100 = 260 GB per edge
- Manageable!

Metadata cache (Redis):

- 10,000 videos × 10 KB = 100 MB
- Cache ALL metadata! Tiny!

```
**Interview Summary**:
```

"Netflix's key challenge is bandwidth, not storage:

- Traffic: 8.3 TB/s global streaming bandwidth
- Storage: 300 TB catalog (manageable)
- CDN: Absolutely critical - serves 95% of traffic
- Cache: Popular content (20%) at edge locations
- Hot content (new releases) pre-pushed to all edges"

```
---

### 3. Design Uber / Lyft ⭐⭐⭐

**Requirements**:
```

- 50M daily rides
- 100M registered users (50% are drivers)
- Location updates every 3 seconds while active
- Average ride: 15 minutes
- Store trip data for 5 years

```
**Traffic Calculation**:
```

Active drivers at peak:

- Assume 10% of drivers active: 50M × 0.1 = 5M drivers
- Peak hour (10% of daily rides): 50M × 0.1 = 5M rides/hour

Location updates (real-time):
Per ride: 15 min ÷ 3 sec = 300 updates per person
Daily updates: 50M rides × 300 × 2 (driver + rider) = 30B updates
Update QPS: 30B ÷ 86,400 = 347K QPS
Peak: 347K × 3 = 1M QPS

Ride matching requests:
50M rides/day ÷ 86,400 = 578 QPS
Each match query searches nearby drivers
Database queries: 578 × 10 (search iterations) = 5.7K QPS

Trip requests (ETAs, pricing):

- Users check prices multiple times
- 50M rides × 5 price checks = 250M requests/day
- 250M ÷ 86,400 = 2,893 QPS (round to 3K QPS)

```
**Storage Calculation**:
```

Location history:

- 30B updates/day × 32 bytes (lat, long, timestamp, user_id) = 960 GB/day
- Yearly: 960 GB × 365 = 350 TB
- 5 years: 350 TB × 5 = 1.75 PB

Trip data:

- 50M trips/day × 2 KB (pickup, dropoff, fare, route) = 100 GB/day
- Yearly: 100 GB × 365 = 36 TB
- 5 years: 36 TB × 5 = 180 TB

Driver/Rider profiles:

- 100M users × 5 KB = 500 GB (negligible)

Total: 1.75 PB + 180 TB = 1.93 PB ≈ 2 PB

```
**Memory Calculation (Critical for Real-time)**:
```

Active drivers (for matching):

- 5M drivers × 100 bytes (location + metadata) = 500 MB
- Fits in single Redis instance!

Active ride requests:

- During peak: 100K concurrent searches
- 100K × 200 bytes = 20 MB

Surge pricing data:

- Store per geohash (city divided into grids)
- 1M geohashes × 50 bytes = 50 MB

Route cache:

- Popular routes: 1M routes × 500 bytes = 500 MB

Total in-memory: 500 MB + 20 MB + 50 MB + 500 MB = 1.07 GB
Single Redis instance handles everything!

```
**Server Estimate**:
```

Application servers:

- 1M peak QPS ÷ 1K QPS per server = 1,000 servers
- With redundancy: 1,500 servers globally

Database:

- Cassandra for location history (write-heavy, 1M QPS)
- PostgreSQL for trips (ACID needed for payments)
- Redis for real-time matching (sub-second)

```
**Interview Summary**:
```

"Uber's main challenge is real-time location processing:

- Traffic: 1M location updates/sec peak
- Storage: 2 PB for 5 years (manageable)
- Memory: 1 GB for real-time data (surprisingly small!)
- Key: Redis Geo commands (GEOADD, GEORADIUS) for O(log N) nearby search
- Database: Cassandra for high write throughput"

```
---

### 4. Design Amazon / E-commerce ⭐⭐⭐

**Requirements**:
```

- 300M customers, 50M DAU (active shoppers)
- 1M products in catalog
- 1M orders per day
- Each user views 50 product pages per day
- Average product page: 500 KB (images, reviews)
- Store order history forever

```
**Traffic Calculation**:
```

Product views:
50M users × 50 pages = 2.5B page views/day
2.5B ÷ 86,400 = 28,935 QPS (round to 30K QPS)
Peak (Black Friday, 10x): 300K QPS

Product searches:
50M users × 10 searches = 500M searches/day
500M ÷ 86,400 = 5,787 QPS (round to 6K QPS)
Peak: 60K QPS

Orders (writes):
1M orders/day ÷ 86,400 = 11.5 QPS (round to 12 QPS)
Peak: 120 QPS (easily handled)

Checkout process (critical path):
1M checkouts/day ÷ 86,400 = 11.5 QPS
Must be ACID-compliant (payment processing)

Inventory checks (frequent):

- Every add-to-cart checks inventory
- 50M users × 5 add-to-cart = 250M checks/day
- 250M ÷ 86,400 = 2,893 QPS (round to 3K QPS)

```
**Storage Calculation**:
```

Product catalog:

- 1M products × 50 KB (details, specs) = 50 GB
- Product images: 1M × 2 MB = 2 TB
- Total catalog: 2.05 TB (small!)

Order history:

- 1M orders/day × 5 KB (order details, items, customer) = 5 GB/day
- Yearly: 5 GB × 365 = 1.825 TB
- 10 years: 18.25 TB

Product reviews:

- Assume 10% of orders leave review
- 1M × 0.1 = 100K reviews/day
- 100K × 1 KB = 100 MB/day
- 10 years: 100 MB × 365 × 10 = 365 GB

User data:

- 300M users × 2 KB = 600 GB

Total storage: 2 TB + 18 TB + 0.4 TB + 0.6 TB = 21 TB

With product images at multiple sizes:

- Original: 2 TB
- Large (product page): 1 TB
- Thumbnail: 200 GB
- Additional: 3.2 TB

Grand total: 25 TB (surprisingly small!)

```
**Cache Calculation**:
```

Hot products (80-20 rule):

- 1M products × 0.2 = 200K hot products

- 200K × 50 KB = 10 GB (metadata)
- Images served from CDN

Recent orders (last 90 days):

- 1M/day × 90 = 90M orders
- 90M × 5 KB = 450 GB

Shopping carts (active sessions):

- 5M active shoppers × 20 KB = 100 GB

Total cache: 10 GB + 450 GB + 100 GB = 560 GB
Across 10 Redis nodes: 56 GB each

```

**Bandwidth Calculation**:
```

Product page loads:
30K QPS × 500 KB = 15 GB/s

With CDN caching (90% hit rate):

- Origin: 15 GB/s × 0.1 = 1.5 GB/s
- CDN: 15 GB/s × 0.9 = 13.5 GB/s

Product images dominate bandwidth, not product data

```

**Interview Summary**:
```

"E-commerce is read-heavy with critical write path:

- Traffic: 30K product view QPS, only 12 order QPS
- Storage: 25 TB total (orders + catalog)
- Cache: 560 GB for hot products + carts
- Critical: Inventory management (3K QPS checks)
- Database: PostgreSQL for orders (ACID), Redis for cart/inventory
- Need: Strong consistency for checkout, eventual OK for catalog"

```

---

### 5. Design Reddit / News Aggregator ⭐⭐

**Requirements**:
```

- 50M DAU
- 500K posts per day (0.01 posts per user)
- Each user views 100 posts per day
- Each post: 2 KB (text, metadata)
- Comments: 5M per day (avg 10 per post)
- Store for 10 years

```
**Traffic Calculation**:
```

Post creation:
500K posts/day ÷ 86,400 = 5.7 QPS (round to 6 QPS)
Peak: 18 QPS

Comment creation:
5M comments/day ÷ 86,400 = 57 QPS
Peak: 171 QPS

Post views:
50M × 100 = 5B views/day
5B ÷ 86,400 = 57,870 QPS (round to 58K QPS)
Peak: 174K QPS

Vote operations (upvote/downvote):

- Assume each user votes 50 times/day
- 50M × 50 = 2.5B votes/day
- 2.5B ÷ 86,400 = 28,935 QPS (round to 30K QPS)
- Peak: 90K QPS

Total read QPS: 58K (views) + 30K (votes) = 88K QPS
Total write QPS: 6 (posts) + 57 (comments) + 30K (votes) = 30K QPS

```
**Storage Calculation**:
```

Posts:

- 500K/day × 2 KB = 1 GB/day
- 10 years: 1 GB × 365 × 10 = 3.65 TB

Comments:

- 5M/day × 1 KB = 5 GB/day
- 10 years: 5 GB × 365 × 10 = 18.25 TB

Votes:

- Store user_id + post_id + vote_type
- 2.5B votes/day × 25 bytes = 62.5 GB/day
- 10 years: 62.5 GB × 365 × 10 = 228 TB

User data:

- 50M × 5 KB = 250 GB

Total: 3.65 TB + 18.25 TB + 228 TB + 0.25 TB = 250 TB

```
**Cache Calculation**:
```

Front page posts (hot):

- Top 1000 posts (refreshed every 30 min)
- 1000 × 2 KB = 2 MB (tiny!)
- Cache entire front page

Hot posts (trending):

- Top 10K posts × 2 KB = 20 MB

Comment trees (popular threads):

- Top 1000 threads × 100 comments × 1 KB = 100 MB

User karma/profile:

- 10M active × 2 KB = 20 GB

Total cache: 20 GB + 100 MB + 20 MB = ~20 GB
Single Redis instance sufficient!

```
**Server Estimate**:
```

Read servers:
88K QPS ÷ 2K QPS per server = 44 servers
With redundancy: 70 servers

Write servers:
30K QPS ÷ 2K QPS per server = 15 servers
With redundancy: 25 servers

Database:

- PostgreSQL for posts/comments (small dataset)
- Redis for vote counters (high write QPS)
- Elasticsearch for search

```
**Interview Summary**:
```

"Reddit is interesting - high vote QPS but small storage:

- Traffic: 88K read QPS, 30K write QPS (balanced)
- Storage: 250 TB over 10 years (modest)
- Cache: 20 GB (front page + hot threads)
- Challenge: Vote aggregation in real-time
- Solution: Redis counters + periodic batch updates"

```
---

### 6. Design Rate Limiter ⭐⭐⭐

**Requirements**:
```

- 1B API calls per day
- Rate limit: 100 requests per minute per user
- 10M active API users
- Track per user, per API key, per IP

```
**Traffic Calculation**:
```

API calls:
1B calls/day ÷ 86,400 = 11,574 QPS (round to 12K QPS)
Peak: 12K × 3 = 36K QPS

Rate limit checks:

- Every API call needs rate limit check
- Same as API traffic: 12K QPS, peak 36K QPS
- Must add < 10ms latency (or API too slow)

```
**Storage Calculation**:
```

Counter storage (in-memory):
Per user counters:

- user_id: 8 bytes
- counter: 4 bytes
- window_timestamp: 8 bytes
- Total: 20 bytes per counter

Active users in last minute:

- 10M users × 20 bytes = 200 MB
- With multiple time windows (sliding window):
- Current + Previous window: 200 MB × 2 = 400 MB

Total memory: 400 MB (fits easily in Redis!)

Historical data (analytics):

- Store violations for analysis
- 1% violation rate: 1B × 0.01 = 10M violations/day
- 10M × 100 bytes = 1 GB/day
- Yearly: 365 GB

Rule configuration:

- 10K rate limit rules × 500 bytes = 5 MB
- Cache ALL rules in memory

```


**Memory Calculation**:
```

Redis for counters:

- 400 MB active data
- Single Redis instance: 16 GB
- Plenty of headroom for growth

Rules cache (application memory):

- 5 MB per server
- Loaded at startup
- Refresh every 5 minutes

```


**Server Estimate**:
```

Rate limiter must be FAST:

- Target: < 1ms overhead
- Redis GET + INCR: ~0.5ms
- Network: ~0.5ms
- Total: ~1ms ✓

Servers needed:

- 36K peak QPS ÷ 10K QPS per server = 3.6 servers
- With redundancy: 10 servers globally
- Each server: Connection pool to Redis

Redis cluster:

- 5 nodes for redundancy
- Each handles 7.2K QPS (easy for Redis)

```
**Interview Summary**:
```

"Rate limiter is simple but critical:

- Traffic: 36K peak QPS (must be fast!)
- Storage: 400 MB in-memory (Redis)
- Latency: < 1ms overhead required
- Algorithm: Sliding window counter or Token bucket
- Key: Use Redis atomic operations (INCR)
- Scale: Single Redis instance handles millions of users"

```
---

### 7. Design Notification System ⭐⭐

**Requirements**:
```

- 1B users, 100M DAU
- Each user receives 10 notifications per day
- Notification types: Push, Email, SMS
- 80% push, 15% email, 5% SMS
- Response time: < 100ms to accept notification
- Delivery: Best effort (OK to miss some)

```
**Traffic Calculation**:
```

Total notifications:
100M users × 10 = 1B notifications/day
1B ÷ 86,400 = 11,574 QPS (round to 12K QPS)
Peak: 12K × 3 = 36K QPS

By type:

- Push: 1B × 0.8 = 800M/day = 9.2K QPS
- Email: 1B × 0.15 = 150M/day = 1.7K QPS
- SMS: 1B × 0.05 = 50M/day = 578 QPS

Fan-out scenario:

- User posts → notify 1000 followers
- 1M posts/day = 1B notifications
- Fanout QPS: Same 12K QPS

```
**Storage Calculation**:
```

Notification logs:

- 1B notifications/day × 500 bytes = 500 GB/day
- 90-day retention: 500 GB × 90 = 45 TB

User preferences:

- 1B users × 2 KB (preferences per notification type) = 2 TB

Device tokens:

- 1B users × 300 bytes (APNs/FCM tokens) = 300 GB

Templates:

- 1000 templates × 10 KB = 10 MB (negligible)

Total: 45 TB + 2 TB + 0.3 TB = 47 TB

```
**Memory Calculation**:
```

Cache user preferences:

- 100M DAU × 2 KB = 200 GB
- Cache hot users (20M) × 2 KB = 40 GB

Cache device tokens:

- 100M DAU × 300 bytes = 30 GB

Notification queue (pending):

- Buffer 5 minutes of notifications
- 12K QPS × 300 seconds = 3.6M notifications
- 3.6M × 500 bytes = 1.8 GB

Total cache: 40 GB + 30 GB + 2 GB = 72 GB
Across 5 Redis nodes: 15 GB each

```
**Server Estimate**:
```

Notification workers:

- 36K peak QPS ÷ 5K QPS per worker = 7.2 workers
- With redundancy: 15 workers

Push notification service:

- APNs connection pool: 100 connections
- FCM connection pool: 100 connections
- Each handles 1K QPS

Email service:

- Use SES/SendGrid
- 1.7K QPS (52M emails/day)
- Batched sending

SMS service:

- Use Twilio
- 578 QPS (50M SMS/day)
- Most expensive channel

```
**Interview Summary**:
```

"Notification system needs reliability and fanout:

- Traffic: 36K peak QPS (manageable)
- Storage: 47 TB (mostly logs)
- Challenge: Fan-out (1 post → 1000 notifications)
- Solution: Queue-based async processing (Kafka)
- Priority queue: High (transactions) > Medium (social) > Low (marketing)

- Deduplication: Merge similar notifications"

```
---

### 8. Design Google Search Autocomplete ⭐⭐

**Requirements**:
```

- 5B searches per day (Google scale)
- Autocomplete: Suggest after 3+ characters
- Suggest top 10 completions
- Update suggestions hourly
- Multi-language support

```
**Traffic Calculation**:
```

Autocomplete requests:

- Each search has ~5 autocomplete requests (typing)
- 5B searches × 5 = 25B autocomplete requests/day
- 25B ÷ 86,400 = 289,351 QPS (round to 300K QPS)
- Peak: 300K × 3 = 900K QPS

This is READ-ONLY and latency-critical!
Must respond in < 100ms (ideally < 50ms)

```
**Storage Calculation**:
```

Search query logs:

- 5B searches/day × 50 bytes (query text + metadata) = 250 GB/day
- 30-day retention: 250 GB × 30 = 7.5 TB

Trie/Prefix data structure:

- Store popular queries (top 10M)
- Each node: ~100 bytes
- Total nodes: ~100M (branching factor)
- 100M × 100 bytes = 10 GB

Suggestions cache:

- Pre-compute top 10 suggestions for popular prefixes
- 1M popular prefixes × 1 KB (10 suggestions) = 1 GB

Total: 7.5 TB (logs) + 10 GB (trie) + 1 GB (cache) = 7.5 TB

```
**Memory Calculation (Critical)**:
```

Must be in-memory for speed!

Trie structure: 10 GB

- Fits in single server memory
- Replicate across regions: 10 GB × 5 regions = 50 GB total

Top 10M queries cache:

- 10M × 100 bytes = 1 GB

Per-language tries:

- 20 languages × 10 GB = 200 GB
- Distribute across 20 servers: 10 GB each

Total: 200 GB distributed

```
**Server Estimate**:
```

Query servers:

- 900K peak QPS ÷ 10K QPS per server = 90 servers
- With redundancy: 150 servers globally

Per region (5 regions):

- 150 ÷ 5 = 30 servers per region

Each server:

- Loads trie into memory (10 GB)
- Serves 10K QPS
- < 10ms latency

```
**Interview Summary**:
```

"Autocomplete is latency-critical:

- Traffic: 900K peak QPS (very high!)
- Storage: 10 GB trie (must fit in memory)
- Latency: < 50ms (in-memory trie essential)
- Solution: Trie data structure in memory
- Optimization: Pre-compute popular suggestions
- Global: Replicate trie to all regions
- Update: Rebuild trie hourly from logs"

```
---

### 9. Design Web Crawler (Google Bot) ⭐⭐

**Requirements**:
```

- Crawl 1 billion web pages
- Re-crawl every 7 days (freshness)
- Average page: 500 KB
- Follow 20 links per page
- Respect robots.txt and rate limits (1 req/sec per domain)

```
**Traffic Calculation**:
```

Pages to crawl per day:
1B pages ÷ 7 days = 143M pages/day
143M ÷ 86,400 = 1,655 QPS (round to 1.7K QPS)
Peak: 5K QPS

URLs discovered per day:
143M pages × 20 links = 2.86B URLs
Need to check if already crawled: 2.86B lookups/day
2.86B ÷ 86,400 = 33K lookup QPS

DNS lookups:

- Assume 100K unique domains
- Each page needs DNS: 1.7K QPS DNS lookups
- With caching (90% hit): 170 QPS actual DNS queries

```
**Storage Calculation**:
```

Page content:

- 143M pages/day × 500 KB = 71.5 TB/day
- With 7-day rotation: 71.5 TB × 7 = 500 TB
- Only need latest snapshot: 500 TB

URL frontier (to be crawled):

- 2.86B URLs × 200 bytes = 572 GB
- Keep in Redis for fast access

Already crawled URLs (Bloom filter):

- 1B URLs tracked
- Bloom filter: ~1.2 GB (1% false positive rate)

Extracted metadata:

- 1B pages × 10 KB (title, meta, links) = 10 TB

Total: 500 TB + 0.57 TB + 10 TB = 511 TB

```


**Memory Calculation**:
```

URL frontier (priority queue):

- 100M URLs ready to crawl
- 100M × 200 bytes = 20 GB
- Redis sorted set (score = priority)

DNS cache:

- 100K domains × 100 bytes = 10 MB

Bloom filter (visited URLs):

- 1B URLs: 1.2 GB in memory

Robots.txt cache:

- 100K domains × 10 KB = 1 GB

Total: 20 GB + 0.01 GB + 1.2 GB + 1 GB = 22 GB
Fits in 3-4 servers

```


**Server Estimate**:
```

Crawler workers:

- 1.7K pages/sec to crawl
- Each worker: 10 concurrent requests = 10 pages/sec
- Workers needed: 1,700 ÷ 10 = 170 workers
- With redundancy: 250 workers

Rate limiting consideration:

- 1 req/sec per domain
- 100K domains = 100K req/sec theoretical max
- Our 1.7K QPS is well within limit

```
**Interview Summary**:
```

"Web crawler needs politeness and de-duplication:

- Traffic: 1.7K crawl QPS, 33K URL dedup QPS
- Storage: 511 TB (page snapshots rotate every 7 days)
- Memory: 22 GB (frontier + bloom filter + DNS cache)
- Challenge: Avoid re-crawling, respect rate limits
- Solution: Bloom filter for visited URLs, robots.txt caching
- Politeness: Max 1 req/sec per domain"

```
---

### 10. Design Pastebin ⭐⭐

**Requirements**:
```

- 10M DAU
- Each user creates 1 paste per day
- Each paste: 10 KB (code snippet)
- Paste expiry: 24 hours (default)
- Each paste viewed 100 times (read-heavy)
- Store active pastes (< 30 days old)

```
**Traffic Calculation**:
```

Writes (create paste):
10M creates/day ÷ 86,400 = 115 QPS

Peak: 345 QPS

Reads (view paste):
10M pastes × 100 views = 1B views/day
1B ÷ 86,400 = 11,574 QPS (round to 12K QPS)
Peak: 36K QPS

Ratio: 100:1 (read-heavy)

```
**Storage Calculation**:
```

Active pastes (30-day retention):

- 10M/day × 30 days = 300M active pastes
- 300M × 10 KB = 3 TB

With automatic expiry:

- 24-hour pastes: 10M × 10 KB = 100 GB
- 1-week pastes: 70M × 10 KB = 700 GB
- Permanent pastes (10%): 1M/day × 10 KB = 10 GB/day
  Over 5 years: 10 GB × 365 × 5 = 18 TB

Total: 3 TB (active) + 18 TB (permanent) = 21 TB

```
**Cache Calculation**:
```

Hot pastes (recent + popular):

- Last 24 hours: 10M pastes
- Apply 80-20: 10M × 0.2 = 2M hot pastes
- 2M × 10 KB = 20 GB

Cache ALL recent pastes!

- 10M × 10 KB = 100 GB
- Distributed: 10 servers × 10 GB each

Cache hit rate: 95%+ (recent pastes very popular)

```
**Bandwidth Calculation**:
```

Incoming: 115 QPS × 10 KB = 1.15 MB/s
Outgoing: 12K QPS × 10 KB = 120 MB/s

With CDN (optional for code):

- Origin: 120 MB/s × 0.1 = 12 MB/s
- CDN: 120 MB/s × 0.9 = 108 MB/s

```
**Interview Summary**:
```

"Pastebin is surprisingly simple:

- Traffic: 12K read QPS, 115 write QPS
- Storage: 21 TB over 5 years
- Cache: 100 GB (recent pastes)
- TTL: Critical feature (auto-cleanup)
- Database: PostgreSQL sufficient (small dataset)
- Optimization: Cache recent pastes (95%+ hit rate)"

```
---

### 11. Design LinkedIn / Professional Network ⭐⭐

**Requirements**:
```

- 800M users, 300M DAU
- Each user views 20 profiles per day
- Each user updates profile 0.1 times per day
- Each user sees 50 feed posts per day
- Profile size: 20 KB (resume, skills, experience)
- Connection graph: 500 connections average per user

```
**Traffic Calculation**:
```

Profile views:
300M × 20 = 6B views/day
6B ÷ 86,400 = 69,444 QPS (round to 70K QPS)
Peak: 210K QPS

Profile updates:
300M × 0.1 = 30M updates/day

30M ÷ 86,400 = 347 QPS
Peak: 1K QPS

Feed views:
300M × 50 = 15B posts viewed/day
15B ÷ 86,400 = 173,611 QPS (round to 174K QPS)
Peak: 522K QPS

Connection requests:

- Assume 2 new connections per user per day
- 300M × 2 = 600M requests/day
- 600M ÷ 86,400 = 6,944 QPS (round to 7K QPS)

```
**Storage Calculation**:
```

User profiles:

- 800M users × 20 KB = 16 TB

Connections (social graph):

- 800M users × 500 connections = 400B edges
- Each edge: 16 bytes (user_id + connection_id)
- 400B × 16 bytes = 6.4 TB

Posts/Updates:

- 30M posts/day × 2 KB = 60 GB/day
- 5 years: 60 GB × 365 × 5 = 109 TB

Messages:

- Similar to LinkedIn messaging
- Assume 100M messages/day × 500 bytes = 50 GB/day
- 2 years: 50 GB × 365 × 2 = 36 TB

Total: 16 TB + 6.4 TB + 109 TB + 36 TB = 167 TB

```
**Cache Calculation**:
```

Hot profiles (20%):

- 300M DAU × 0.2 = 60M profiles
- 60M × 20 KB = 1.2 TB

- Distributed: 20 servers × 60 GB each

Connection graph:

- Cache 60M users' connections
- 60M × 500 × 16 bytes = 480 GB

Feed cache:

- 60M users × 50 posts × 2 KB = 6 TB

Total: 1.2 TB + 0.48 TB + 6 TB = 7.68 TB
Across 80 servers: ~100 GB each

```
**Interview Summary**:
```

"LinkedIn is profile-heavy with social graph:

- Traffic: 174K feed QPS, 70K profile QPS
- Storage: 167 TB (profiles + connections + content)
- Cache: 7.68 TB (profiles + connections + feeds)
- Challenge: Connection graph queries (2nd/3rd degree)
- Database: Neo4j/graph DB for connections, PostgreSQL for profiles
- Optimization: Denormalize feed data, cache aggressively"

```
---

### 12. Design Discord / Slack ⭐⭐

**Requirements**:
```

- 150M DAU
- 10M active servers/workspaces
- Average: 1000 messages per server per day
- Each user in 5 servers average
- Real-time messaging
- Store for 90 days (free), forever (premium)

```
**Traffic Calculation**:
```

Messages per day:
10M servers × 1000 = 10B messages/day

10B ÷ 86,400 = 115,740 QPS (round to 116K QPS)
Peak: 348K QPS

User presence updates:

- 150M users × online/offline changes
- Assume 10 status changes per user per day
- 150M × 10 = 1.5B updates/day
- 1.5B ÷ 86,400 = 17,361 QPS (round to 17K QPS)

Typing indicators:

- Active: 15M users typing at any moment
- Update every 3 seconds
- 15M ÷ 3 = 5M updates/sec = 5M QPS (very high!)
- Optimization: Throttle to 1 update per 3 seconds = 17K QPS

```
**Storage Calculation**:
```

Messages:

- 10B/day × 200 bytes = 2 TB/day
- 90 days: 2 TB × 90 = 180 TB

Voice/Video data (separate calculation):

- Assume 10M minutes of voice per day
- 10M min × 1 MB/min = 10 TB/day
- 90 days: 10 TB × 90 = 900 TB

File uploads:

- Assume 100M files/day × 1 MB avg = 100 TB/day
- 90 days: 100 TB × 90 = 9 PB

User data:

- 150M users × 5 KB = 750 GB

Server metadata:

- 10M servers × 10 KB = 100 GB

Total: 180 TB + 900 TB + 9 PB + 0.75 TB = 10 PB

```
**Memory Calculation**:
```

Online users (presence):

- 150M concurrent × 100 bytes = 15 GB

Active servers (hot):

- 1M active servers × 1 MB (recent messages) = 1 TB

Typing indicators (ephemeral):

- 15M users × 50 bytes = 750 MB

Message queue (undelivered):

- Buffer 10 minutes: 116K QPS × 600 sec = 70M messages
- 70M × 200 bytes = 14 GB

Total: 15 GB + 1 TB + 0.75 GB + 14 GB = 1.03 TB
Across 20 Redis nodes: 51 GB each

```
**Interview Summary**:
```

"Discord/Slack is real-time group messaging at scale:

- Traffic: 348K message QPS peak
- Storage: 10 PB (90-day retention)
- Memory: 1 TB (presence + messages + queue)
- Challenge: Real-time delivery to 1000s in server
- Solution: WebSocket + fanout via Redis Pub/Sub
- Database: Cassandra for messages, PostgreSQL for users/servers"

```
---

### 13. Design Zoom / Video Conferencing ⭐⭐⭐

**Requirements**:
```

- 300M DAU
- 10M concurrent meetings at peak
- Average meeting: 4 participants, 30 minutes
- Video: 720p (1.5 Mbps), Audio: 64 Kbps
- Store recordings: 10% of meetings, for 1 year

```
```

```
**Traffic Calculation**:
```

Concurrent connections:

- 10M meetings × 4 participants = 40M concurrent connections
- Real-time streams (not traditional QPS)

Signaling QPS (WebRTC setup):

- 10M meetings/day ÷ 86,400 = 115 meetings/sec
- Each meeting: 4 participants × 10 signaling messages = 40 messages
- 115 × 40 = 4,600 signaling QPS

Media relay QPS:

- Not all peer-to-peer, some need relay
- Assume 30% need relay: 40M × 0.3 = 12M relayed connections
- Each sending/receiving: 24M streams total

```
**Bandwidth Calculation** (Critical):
```

Per participant bandwidth:

- Video upload: 1.5 Mbps = 187.5 KB/s
- Audio upload: 64 Kbps = 8 KB/s
- Total upload: ~195 KB/s
- Video downloads: 1.5 Mbps × 3 (other participants) = 4.5 Mbps = 562.5 KB/s
- Audio downloads: 64 Kbps × 3 = 192 Kbps = 24 KB/s
- Total download: ~586 KB/s

For 40M concurrent participants:

- Upload bandwidth: 40M × 195 KB/s = 7.8 TB/s
- Download bandwidth: 40M × 586 KB/s = 23.4 TB/s

This is HUGE! Need optimization:

With P2P (peer-to-peer):

- 70% connections don't hit servers
- Server bandwidth: 7.8 TB/s × 0.3 = 2.3 TB/s

With SFU (Selective Forwarding Unit) optimization:

- Server receives once, forwards to multiple
- Reduction: ~50%
- Server bandwidth: ~1.2 TB/s

```
**Storage Calculation**:
```

Meeting recordings:

- 10% of 10M meetings/day = 1M recordings/day
- Each: 30 min × 4 participants × 1.5 Mbps (video) + 64 Kbps (audio)
- Per recording: 30 min × 4 × (1.5 Mbps × 60 sec / 8) = 30 × 4 × 11.25 MB = 1.35 GB
- Daily: 1M × 1.35 GB = 1.35 PB/day
- Yearly: 1.35 PB × 365 = 492 PB

With compression (30% reduction):

- Actual: 492 PB × 0.7 = 344 PB/year

Metadata:

- 10M meetings/day × 2 KB = 20 GB/day
- Yearly: 7.3 TB (negligible)

Total: 344 PB per year

```
**Interview Summary**:
```

"Zoom's challenge is real-time bandwidth:

- Connections: 40M concurrent streams
- Bandwidth: 1.2 TB/s with SFU optimization
- Storage: 344 PB/year for recordings
- Critical: Low latency (< 200ms) for good UX
- Solution: Geographic distribution, P2P when possible, SFU for groups
- Media servers: Distributed globally (50+ regions)"

```
---

### 14. Design Ticketmaster / Event Booking ⭐⭐⭐

**Requirements**:
```

- 100M users, 10M DAU
- 10K events per day
- Average event: 10K tickets
- Flash sales: 100K users compete for 10K tickets in 1 minute
- Booking flow must prevent overselling

```
**Traffic Calculation** (Critical - Spike Pattern):
```

Normal browsing:

- 10M users × 10 page views = 100M views/day
- 100M ÷ 86,400 = 1,157 QPS (round to 1.2K QPS)

Flash sale (worst case):

- 100K users trying to book in 1 minute
- Each user: 10 attempts = 1M booking requests in 60 seconds
- Spike QPS: 1M ÷ 60 = 16,666 QPS (round to 17K QPS)
- This is 14x normal load!

Ticket inventory checks:

- Before every booking: Check seat available
- 17K booking attempts/sec = 17K inventory checks/sec
- Must be ACID-compliant (no double booking)

```
**Storage Calculation**:
```

Events:

- 10K events/day × 10 KB = 100 MB/day
- 2 years: 100 MB × 365 × 2 = 73 GB

Tickets:

- 10K events/day × 10K tickets = 100M tickets/day
- Each ticket: 500 bytes (seat, price, status)
- 100M × 500 bytes = 50 GB/day
- 2 years: 50 GB × 365 × 2 = 36 TB

Bookings:

- Assume 80% tickets sold
- 100M × 0.8 = 80M bookings/day

- Each booking: 2 KB (user, ticket, payment)
- 80M × 2 KB = 160 GB/day
- 5 years: 160 GB × 365 × 5 = 292 TB

Total: 0.07 TB + 36 TB + 292 TB = 328 TB

```
**Memory Calculation** (Critical for Flash Sales):
```

Ticket inventory (must be in-memory for speed):

- Current active events: 10K events
- 10K × 10K tickets = 100M tickets
- Each: 100 bytes (seat_id, status, price)
- 100M × 100 bytes = 10 GB

Booking locks (prevent double-booking):

- During flash sale: 17K concurrent booking attempts
- Each lock: 50 bytes
- 17K × 50 bytes = 850 KB (tiny!)

User sessions:

- 100K active users × 10 KB = 1 GB

Total: 10 GB + 0.85 MB + 1 GB = 11 GB
Single Redis instance handles it!

```
**Interview Summary**:
```

"Ticketmaster needs to handle extreme spikes:

- Traffic: 17K QPS during flash sales (vs 1.2K normal)
- Storage: 328 TB (manageable)
- Memory: 11 GB in-memory inventory (critical!)
- Challenge: Race condition (overselling)
- Solution: Redis distributed locks + optimistic locking
- Database: PostgreSQL with row-level locks for ACID
- Queue: Virtual waiting room to smooth spike"

```
---

### 15. Design Parking Lot System ⭐
```

```
**Requirements**:
```

- 10K parking spots per location
- 100 locations (malls, airports)
- 5M vehicles per day
- Average stay: 2 hours
- Real-time availability

```

**Traffic Calculation**:
```

Entry/exit events:

- 5M vehicles × 2 events (entry + exit) = 10M events/day
- 10M ÷ 86,400 = 115 QPS
- Peak (rush hour, 3x): 345 QPS

Availability checks:

- Drivers check before arriving
- 5M drivers × 3 checks = 15M checks/day
- 15M ÷ 86,400 = 173 QPS
- Peak: 519 QPS

Display updates:

- 100 locations × 10 displays = 1000 displays
- Poll every 5 seconds = 1000 ÷ 5 = 200 QPS
- Always-on, no peak variation

```

**Storage Calculation**:
```

Parking spots:

- 100 locations × 10K spots = 1M spots total
- Each spot: 200 bytes (spot_id, location, type, floor)
- 1M × 200 bytes = 200 MB (tiny!)

Parking events:

- 10M events/day × 500 bytes = 5 GB/day
- 1 year retention: 5 GB × 365 = 1.825 TB

Payment records:

- 5M transactions/day × 1 KB = 5 GB/day
- 5 years: 5 GB × 365 × 5 = 9 TB

Vehicle data:

- 5M unique vehicles × 500 bytes = 2.5 GB

Total: 0.2 GB + 1.8 TB + 9 TB + 2.5 GB = 11 TB

```
**Memory Calculation**:
```

Real-time availability:

- 1M spots × 50 bytes (spot_id, status, occupied_since) = 50 MB
- Cache ALL spots! Tiny!

Active sessions:

- Peak concurrent vehicles: 5M × (2 hours / 24 hours) = 416K
- 416K × 200 bytes = 83 MB

Recent transactions (for display):

- Last 1000 per location × 100 locations = 100K transactions
- 100K × 500 bytes = 50 MB

Total: 50 MB + 83 MB + 50 MB = 183 MB
Single Redis instance more than sufficient!

```
**Interview Summary**:
```

"Parking lot is small scale but real-time critical:

- Traffic: 519 QPS peak (very manageable)
- Storage: 11 TB over 5 years
- Memory: 183 MB (everything fits in cache!)
- Challenge: Prevent double-booking of spots
- Solution: Redis atomic operations (SETNX for locks)
- Display: WebSocket for real-time updates to screens
- Database: PostgreSQL sufficient (small dataset, ACID needed)"

```
---
```

```
## Top HLD Questions Summary Table
```

System	DAU	QPS	Storage	Cache	Pattern
TinyURL	N/A	4K R	8 TB	2.5 GB	Read-heavy
Twitter	400M	1.4M R	44 PB	1.1 TB	Read-heavy
Instagram	500M	867K R	58 PB	60 TB	Read-heavy
WhatsApp	1B	1.4M RW	1.46 EB	2.5 TB	Balanced
Uber	50M	1M W	2 PB	1 GB	Write-heavy
Netflix	100M	70K API	300 TB	26 TB	Streaming
Amazon	50M	300K R	25 TB	560 GB	Read-heavy
Reddit	50M	88K R	250 TB	20 GB	Read-heavy
Rate Limiter	N/A	36K	400 MB	400 MB	Fast
Notifications	100M	36K	47 TB	72 GB	Fanout
Autocomplete	N/A	900K	10 GB	10 GB	Latency
Web Crawler	N/A	1.7K W	511 TB	22 GB	Write-heavy
Pastebin	10M	12K R	21 TB	100 GB	Read-heavy
LinkedIn	300M	522K R	167 TB	7.68 TB	Read-heavy
Discord/Slack	150M	348K	10 PB	1 TB	Real-time
Zoom	300M	40M Con	344 PB/y	N/A	Streaming
Ticketmaster	10M	17K	328 TB	11 GB	Spike
Parking Lot	5M	519	11 TB	183 MB	Real-time

Legend: R = Read, W = Write, RW = Balanced, Con = Connections

```
---

## Technology Selection Based on Numbers

### Decision Framework: Choose Database

#### When to Use PostgreSQL/MySQL (Relational)

**Use when numbers show:**
```

✓ Storage < 10 TB (fits in single instance)
✓ Write QPS < 10K (single master handles it)
✓ Read QPS < 100K (with 5-10 read replicas)

✓ Need ACID transactions
✓ Complex queries with JOINs

Examples from above:

- TinyURL: 8 TB storage, 40 write QPS → PostgreSQL ✓
- Pastebin: 21 TB storage, 115 write QPS → PostgreSQL ✓
- Amazon orders: ACID needed for payments → PostgreSQL ✓
- Parking Lot: 11 TB, 345 write QPS → PostgreSQL ✓

```
**Justification Template:**
```

"Our calculations show:

- Storage: X TB (< 10 TB threshold)
- Write QPS: Y (< 10K single master limit)
- Read QPS: Z (< 100K with replicas)
- Need: ACID for transactions
  Therefore, PostgreSQL is sufficient and simpler than distributed DB"

```
#### When to Use Cassandra (NoSQL, Write-Heavy)

**Use when numbers show:**
```

✓ Write QPS > 10K (need distributed writes)
✓ Storage > 10 TB (need horizontal scaling)
✓ Time-series data (sorted by timestamp)
✓ Can accept eventual consistency
✓ Need linear scalability

Examples from above:

- Twitter tweets: 44 PB storage, 7K write QPS → Cassandra ✓
- WhatsApp messages: 1.46 EB, 1.4M write QPS → Cassandra ✓
- Uber locations: 2 PB, 1M write QPS → Cassandra ✓
- Discord messages: 10 PB, 348K write QPS → Cassandra ✓

```
**Justification Template:**
```

"Our calculations show:

- Write QPS: 1M/sec (far exceeds single DB limit of 10K)
- Storage: 2 PB (requires distributed storage)
- Data type: Time-series (naturally sorted)
  Therefore, need Cassandra for:
- Linear write scalability (each node adds 10K write QPS)
- Horizontal storage scaling (each node adds 2 TB)
- No single point of failure"

```
#### When to Use MongoDB (NoSQL, Flexible)

**Use when numbers show:**
```

✓ Write QPS: 1K-50K (moderate)
✓ Storage: 1-100 TB
✓ Schema flexibility needed
✓ Read QPS < 100K
✓ Document-oriented data

Examples from above:

- Reddit posts: 250 TB, 6 write QPS → Could use MongoDB
- LinkedIn profiles: 167 TB, moderate complexity → MongoDB option

```
**Justification Template:**
```

"With 10K write QPS and 50 TB storage:

- Too large for single PostgreSQL (> 10 TB)
- Not enough write load for Cassandra
- Schema varies (profiles have different fields)
  Therefore, MongoDB provides:
- Horizontal scaling (sharding)
- Schema flexibility
- Simpler than Cassandra for this scale"

```
### Decision Framework: Choose Cache

#### When to Use Redis

**Use when numbers show:**
```

✓ Cache size: Any (MB to TB)
✓ Need data structures (Lists, Sets, Sorted Sets)
✓ Need persistence
✓ Need Pub/Sub
✓ Operations QPS: Up to 1M per instance

Use Redis for (from examples):

- TinyURL: 2.5 GB cache → Single Redis ✓
- Twitter timelines: 1.1 TB → Redis Cluster (20 nodes) ✓
- Rate limiter: 400 MB counters → Redis (atomic INCR) ✓
- Uber: 1 GB geolocation → Redis (GEO commands) ✓
- All systems: Redis is the default choice

```
**Justification by Numbers:**
```

Cache < 1 GB:
"Cache is 500 MB, single Redis instance (16 GB) handles it easily
with 15GB headroom for growth"

Cache 1-100 GB:
"Cache is 20 GB, use 2-3 Redis instances for redundancy
Each handles 100K+ QPS"

Cache 100 GB - 10 TB:
"Cache is 1.1 TB, use Redis Cluster with 20 nodes
Each node: 55 GB, handles 50K QPS
Total capacity: 1M QPS"

Cache > 10 TB:
"Cache is 60 TB, use Redis Cluster with 100 nodes
Each node: 600 GB, distribute evenly
Total capacity: 5M QPS"

```
#### When to Use Memcached

**Use when numbers show:**
```

✓ Pure key-value (no complex data structures)
✓ No persistence needed
✓ Multi-threaded benefit needed
✓ Extremely high QPS (> 1M per instance)

Rare in interviews - Redis almost always better choice

Only use Memcached:
"If calculations show > 1M QPS per cache instance and pure key-value storage, Memcached's multi-threading gives 10-20% performance edge. Otherwise, use Redis."

```
### Decision Framework: When to Use CDN

**Use CDN when numbers show:**
```

✓ Bandwidth > 1 GB/s (CDN essential)
✓ Static content (images, videos, CSS, JS)
✓ Global users (serve from edge)
✓ Can accept caching (TTL-based)

Cost/Benefit Analysis:

Without CDN:

- Bandwidth: 50 GB/s × $0.08/GB = $4,000/hour = $2.9M/month

With CDN (90% cache hit):

- Origin: 5 GB/s × $0.08/GB = $400/hour = $288K/month
- CDN: 45 GB/s × $0.02/GB = $900/hour = $648K/month
- Total: $936K/month (saves $2M/month = 68% savings!)

Use CDN for:

- Instagram: 57 GB/s → CDN mandatory
- Netflix: 8.3 TB/s → CDN absolutely critical
- Twitter: 19 GB/s → CDN essential
- Amazon: 15 GB/s → CDN required

Don't need CDN:

- Rate Limiter: 36 KB/s → Too small
- Parking Lot: 1 MB/s → Not worth it

```
**Justification Template:**
```

"Our bandwidth calculation shows X GB/s outgoing.

At this scale:

- Without CDN: $Y/month in bandwidth costs

- With CDN (90% cache hit): $Z/month
- Savings: $(Y-Z)/month

Additionally:

- Latency improvement: 200ms → 20ms (edge vs origin)
- Origin load reduction: 90%
- Global reach: 400+ edge locations

Therefore, CDN is essential, not optional."

```


### Decision Framework: Message Queue

**Use Kafka when numbers show:**
```

✓ Event throughput > 10K events/sec
✓ Need durability (no message loss)
✓ Need replay capability
✓ Multiple consumers for same events

Examples:

- Twitter: 7K tweet events/sec → Kafka ✓
- WhatsApp: 1.4M message events/sec → Kafka ✓
- Discord: 348K message events/sec → Kafka ✓
- Notifications: 36K events/sec → Kafka ✓

```


**Justification Template:**
```

"With 100K events/second:

- Kafka throughput: 1M+ events/sec per broker
- Need: 100K ÷ 100K = 1 broker minimum
- With redundancy: 3 brokers (replication factor 3)
- Cost: ~$500/month

Compared to alternatives:

- RabbitMQ: Max ~50K/sec per node (need 2+ nodes)
- SQS: $0.40 per million = $3.5K/month (more expensive)
- Redis Pub/Sub: No persistence, lose messages on failure

Therefore, Kafka for durability + throughput at scale"

```
```

```
### Decision Framework: Object Storage

**Use S3/Cloud Storage when:**
```

✓ Media files (images, videos, documents)
✓ Storage > 1 TB
✓ Unstructured data
✓ Need durability (11 nines)

Cost Analysis:

S3 Standard:

- $0.023/GB/month
- 100 TB = $2,300/month
- 1 PB = $23,000/month
- 1 EB = $23M/month

S3 with lifecycle:

- Hot (< 30 days): Standard
- Warm (30-90 days): Infrequent Access ($0.0125/GB)
- Cold (> 90 days): Glacier ($0.004/GB)
- Savings: ~60%

Examples:

- Instagram: 58 PB → S3 with lifecycle
- Netflix: 300 TB catalog → S3 Standard
- WhatsApp: 1.46 EB → S3 with aggressive lifecycle

```
---

## Technology Decision Matrix

### Database Selection Matrix
```

Criteria	Recommendation
Write QPS < 1K	Single PostgreSQL
Write QPS 1K-10K	PostgreSQL + Write Sharding

Write QPS > 10K	Cassandra / DynamoDB
Read QPS < 10K	Single PostgreSQL
Read QPS 10K-100K	PostgreSQL + 5-10 Read Replicas
Read QPS > 100K	Add Redis Cache (80%+ hit rate)
Storage < 1 TB	Single PostgreSQL Instance
Storage 1-10 TB	PostgreSQL with partitioning
Storage 10-100 TB	Sharded PostgreSQL OR Cassandra
Storage > 100 TB	Cassandra / DynamoDB
Need ACID	PostgreSQL (always)
Need Transactions	PostgreSQL (always)
Eventual Consistency OK	Cassandra (better performance)
Time-series data	Cassandra / TimescaleDB
Social graph	Neo4j / Cassandra wide columns
Document store	MongoDB / Elasticsearch
Key-value	Redis / DynamoDB
└─────────────────┴──────────────────────────────
┘

```
### Cache Selection Matrix
```

┌─────────────────┬──────────────────────────────
┐
| Cache Size | Configuration |
├─────────────────┼──────────────────────────────
┤
< 1 GB	Single Redis (16 GB instance)
1-100 GB	2-5 Redis instances (HA)
100 GB - 1 TB	Redis Cluster (10-20 nodes)
1-10 TB	Redis Cluster (50-100 nodes)
> 10 TB	Redis Cluster (100+ nodes)
QPS < 10K	Single Redis
QPS 10K-100K	Redis with 2-3 replicas
QPS 100K-1M	Redis Cluster (10+ nodes)
QPS > 1M	Redis Cluster (50+ nodes)
TTL required	Redis (native support)
Complex data structures	Redis (Lists, Sets, Sorted Sets)

Pub/Sub needed	Redis (built-in)
Persistence needed	Redis (RDB + AOF)
Pure key-value only	Memcached (slightly faster)
└───────────────┴──────────────────────────

┘

```
### Complete Technology Recommendations by System

#### TinyURL
```

Numbers: 4K read QPS, 40 write QPS, 8 TB storage

Tech Stack:
✓ Database: PostgreSQL
Why: 8 TB fits in single instance, 4K QPS easily handled

✓ Cache: Single Redis (16 GB)
Why: 2.5 GB cache fits easily, 4K QPS trivial for Redis

✓ Web Servers: 10-15 servers
Why: 12K peak QPS ÷ 1K per server = 12 servers

✕ CDN: Not needed
Why: 800 KB/s bandwidth is tiny

✕ Message Queue: Not needed
Why: Synchronous flow sufficient at this scale

```
#### Twitter
```

Numbers: 1.4M read QPS, 7K write QPS, 44 PB storage

Tech Stack:
✓ Database: Cassandra for tweets
Why: 44 PB requires distributed storage, 7K write QPS
Each node: 2 TB storage, 100 write QPS
Nodes needed: 44 PB ÷ 2 TB = 22,000 nodes
Write capacity: 22K nodes × 100 = 2.2M QPS ✓

✓ Database: PostgreSQL for users
Why: 400M users × 2 KB = 800 GB (fits in single instance)
Need ACID for user operations

✓ Cache: Redis Cluster (20 nodes)
Why: 1.1 TB cache ÷ 20 = 55 GB per node
Handles 1M read QPS (50K per node)

✓ CDN: CloudFront
Why: 19 GB/s bandwidth, reduces to 950 MB/s origin (95% savings)

✓ Message Queue: Kafka
Why: 7K tweet events/sec, need fanout to millions

✓ Search: Elasticsearch
Why: Full-text search on 44 PB of tweets

Web Servers: 200-300 servers (1.4M QPS ÷ 5K per server)

```
#### WhatsApp
```

Numbers: 1.4M message QPS, 1.46 EB storage

Tech Stack:
✓ Database: Cassandra for messages
Why: 1.4M write QPS requires massive distribution
1.46 EB requires huge horizontal scaling
Nodes needed: 1.46 EB ÷ 2 TB = 750,000 nodes!
(In reality: Messages deleted after 30 days, much less)

✓ Database: PostgreSQL for users
Why: 2B users × 1 KB = 2 TB (manageable)
Need ACID for authentication

✓ Cache: Redis Cluster (200 nodes)
Why: 2.5 TB ÷ 200 = 12.5 GB per node
Handles presence, sessions, message queues

✓ WebSocket Servers: 8,000-16,000
Why: 500M connections ÷ 65K per server = 7,692 servers

✓ Message Queue: Kafka
Why: 1.4M events/sec, need reliable delivery
Brokers needed: 1.4M ÷ 100K per broker = 14 brokers

✓ Object Storage: S3
Why: 1.46 EB of media files
Cost: ~$300K/month with lifecycle policies

✕ CDN: Limited use
Why: Messages encrypted, can't cache

Only for profile pictures, minimal benefit

```
#### Netflix
```

Numbers: 8.3 TB/s bandwidth, 300 TB catalog

Tech Stack:
✓ Database: PostgreSQL for metadata
Why: 10K videos × 10 KB = 100 MB (tiny!)
70K API QPS handled with 10 read replicas

✓ Object Storage: S3 for videos
Why: 300 TB catalog, need 11-nines durability
Cost: 300 TB × $0.023 = $7K/month (cheap!)

✓ CDN: Custom CDN (Open Connect)
Why: 8.3 TB/s bandwidth is massive!
95% from CDN = 7.8 TB/s from edge
Origin only: 415 GB/s
CDN absolutely critical - without it, impossible

✓ Cache: Redis for metadata
Why: 100 MB metadata, cache everything
Single Redis instance sufficient

✓ Cache: 26 TB at CDN edges
Why: Hot content (20% of catalog)
Distributed: 260 GB per edge × 100 edges

Web Servers: 100-150 (70K API QPS ÷ 500 per server)

```
#### Uber
```

Numbers: 1M location update QPS, 2 PB storage, 1 GB cache

Tech Stack:
✓ Database: Cassandra for locations
Why: 1M write QPS (extreme!)
2 PB storage requires distribution
Nodes: 2 PB ÷ 2 TB = 1,000 nodes
Write capacity: 1,000 × 1K = 1M QPS ✓

✓ Database: PostgreSQL for trips/payments
Why: 12 QPS trips, need ACID for payments

180 TB over 5 years (manageable with partitioning)

✓ Cache: Single Redis (16 GB)
Why: Only 1 GB needed for real-time matching
Redis GEO commands for nearby search
Handles 100K QPS easily

✓ Message Queue: Kafka
Why: 1M location events/sec
Brokers: 1M ÷ 100K = 10 brokers minimum

Web Servers: 1,500 servers (1M QPS ÷ 700 per server)

```
#### Rate Limiter
```

Numbers: 36K QPS, 400 MB storage

Tech Stack:
✓ Cache: Redis (primary storage!)
Why: 400 MB fits in single instance (16 GB)
Need atomic operations (INCR)
Need TTL support (auto-cleanup)
Sub-millisecond latency required
Redis handles 100K-1M QPS easily

✓ Database: PostgreSQL for rules
Why: 10K rules × 500 bytes = 5 MB (tiny!)
Infrequent updates
Need ACID for rule changes

✕ Message Queue: Not needed
Why: Synchronous flow, no async processing

Application Servers: 5-10 (36K QPS ÷ 5K per server)

Critical: Rate limiter adds < 10ms latency
Redis achieves < 1ms, perfect!

```
---

## Scaling Thresholds & Technology Transitions

### When to Shard Database
```

Threshold Indicators:

Single PostgreSQL limits:

- Write QPS: 10K
- Read QPS: 100K (with replicas)
- Storage: 10-16 TB

When to shard:
✓ Write QPS > 10K
✓ Storage > 10 TB
✓ Single table > 100M rows

Sharding Examples:

Instagram (500M users):

- User table: 500M rows
- Shard by user_id (hash-based)
- 10 shards × 50M users each
- Each shard: 50M × 2 KB = 100 GB ✓

Twitter (400M users):

- Users: 400M rows → Shard by user_id (5 shards)
- Tweets: 200B rows → Use Cassandra (can't shard PostgreSQL to this scale)

```
### When to Add Read Replicas
```

Threshold: Read QPS > 10K

Rule of thumb:

- Master: 10K write QPS
- Each replica: +10K read QPS

Examples:

TinyURL (4K read QPS):
Replicas: 4K ÷ 10K = 0.4 → 1 master + 1 replica sufficient

Amazon (300K read QPS):
Replicas: 300K ÷ 10K = 30 replicas
But with 90% cache hit rate:
Actual DB QPS: 300K × 0.1 = 30K
Replicas needed: 30K ÷ 10K = 3 replicas
Configuration: 1 master + 5 read replicas (with headroom)

```
### When to Move to NoSQL
```

Decision Tree:

Storage > 100 TB AND Write QPS > 10K?
→ Cassandra (distributed, write-optimized)

Storage < 100 TB BUT Write QPS > 50K?
→ Cassandra (write throughput critical)

Storage > 10 TB AND Schema flexible?
→ MongoDB (easier than sharded PostgreSQL)

Need ACID no matter what?
→ Stay with PostgreSQL, accept complexity
→ Use sharding if needed
→ Examples: Banking, E-commerce orders

Time-series data > 1 TB?
→ Cassandra or TimescaleDB
→ Examples: Metrics, logs, sensor data

```
---

## Cost-Based Technology Decisions

### Database Cost Comparison
```

Scenario: 100 TB storage, 50K QPS

PostgreSQL (Sharded):

- 10 shards × r5.4xlarge = $2,500/month
- Complex sharding logic
- Total: ~$3,000/month

Cassandra (Managed):

- 50 nodes × $100/month = $5,000/month
- Simpler operations
- Better scalability
- Total: ~$5,000/month

MongoDB Atlas:

- Cluster: ~$4,000/month
- Middle ground

Decision: If budget tight and can manage sharding → PostgreSQL
If need simplicity and scale → Cassandra

```
### Cache Cost Comparison
```

Scenario: 1 TB cache needed

Redis Cluster (Self-managed):

- 20 nodes × r5.large ($50/month) = $1,000/month
- Operational overhead
- Full control

ElastiCache (Managed Redis):

- 20 × cache.r5.large = $2,000/month
- No operational overhead
- Auto-failover

Memcached:

- Slightly cheaper (~$1,800/month)
- Fewer features

Decision: Use managed Redis unless budget critical

```
---

## Interview Justification Examples

### Example 1: Justify Cassandra for Twitter
```

"Let me justify why Cassandra for tweets:

Our calculations show:

1. Write QPS: 7K (peak)
    - Single PostgreSQL limit: 10K

- Close to limit, but manageable

2. Storage: 44 PB

   - Single PostgreSQL limit: 16 TB
   - Would need: 44 PB ÷ 16 TB = 2,750 shards!
   - Cassandra: 44 PB ÷ 2 TB = 22K nodes (manageable)

3. Data pattern: Time-series

   - Tweets naturally sorted by time
   - Cassandra optimized for time-series
   - PostgreSQL requires complex partitioning

4. Consistency: Eventual OK

   - Tweet appears in timeline within 1 second acceptable
   - Don't need immediate consistency
   - Cassandra's eventual consistency works

5. Availability: Must be 99.99%

   - Cassandra: No single point of failure
   - PostgreSQL: Master failure = downtime

Therefore, Cassandra is the right choice for Twitter's tweet storage."

```


---

## Summary: The Golden Rules

1. **Round liberally** - 387M → 400M
2. **State assumptions** - "Assuming X..."
3. **Show your work** - Write calculations down
4. **Use shortcuts** - Daily ÷ 100K = QPS
5. **Apply 80-20 rule** - Cache 20%, serve 80%
6. **Mention peak traffic** - Average × 3
7. **Don't forget media** - Images dominate storage
8. **Include replication** - × 3 for redundancy
9. **Sanity check** - Does 1 EB for Twitter make sense? No!
10. **Mention optimizations** - CDN, compression, caching
11. **Justify technology choices** - Use numbers to back decisions
12. **Compare alternatives** - Show cost/performance tradeoffs
13. **Scale path** - Explain how system grows

---

## Final Interview Checklist

**Before presenting your design, verify:**
```

```
✅ **Calculations Complete**
- [ ] Traffic (QPS) calculated for reads and writes
- [ ] Storage estimated with retention period
- [ ] Bandwidth computed with CDN impact
- [ ] Cache sized using 80-20 rule
- [ ] Peak traffic considered (2-3x)

✅ **Technology Justified**
- [ ] Database choice explained with numbers
- [ ] Cache strategy supported by calculations
- [ ] CDN decision backed by cost analysis
- [ ] Message queue justified by throughput
- [ ] Each technology linked to specific requirements

✅ **Presentation Quality**
- [ ] Used round numbers throughout
- [ ] Showed all units in calculations
- [ ] Stated assumptions clearly
- [ ] Sanity-checked results
- [ ] Compared to real-world systems

---

## References

### Books
1. **"Designing Data-Intensive Applications"** - Martin Kleppmann
2. **"System Design Interview"** - Alex Xu (Volume 1 & 2)
3. **"Database Internals"** - Alex Petrov

### Online Resources
1. **System Design Primer** - GitHub (donnemartin)
2. **Google's Latency Numbers** - Jeff Dean
3. **AWS Calculator** - For cost estimates
4. **High Scalability Blog** - Real-world numbers

### Videos
1. **"System Design Course"** - YouTube (Gaurav Sen)
2. **"Back of Envelope Calculations"** - YouTube (SystemDesignInterview)
3. **"Scaling to Millions"** - YouTube (Engineering with Utsav)

---

**END OF GUIDE**

This guide provides a comprehensive framework for back-of-the-envelope
calculations and technology selection in system design interviews. Use
it as a reference, practice the examples, and adapt the templates to
your specific interview questions.

**Good luck with your system design interviews!**
```