1. Client-Server & Tiered Architectures

Assignment:

A bank wants to modernize its legacy system. Compare 2-tier and 3-tier architectures for:

1. Security
2. Scalability
3. Maintenance

Tasks:

1. Diagram both architectures.
2. Explain how the 3-tier system better handles ATM transactions vs. online banking.
3. Identify a use case where 2-tier might suffice

Deliverable:

1-page report + diagrams.

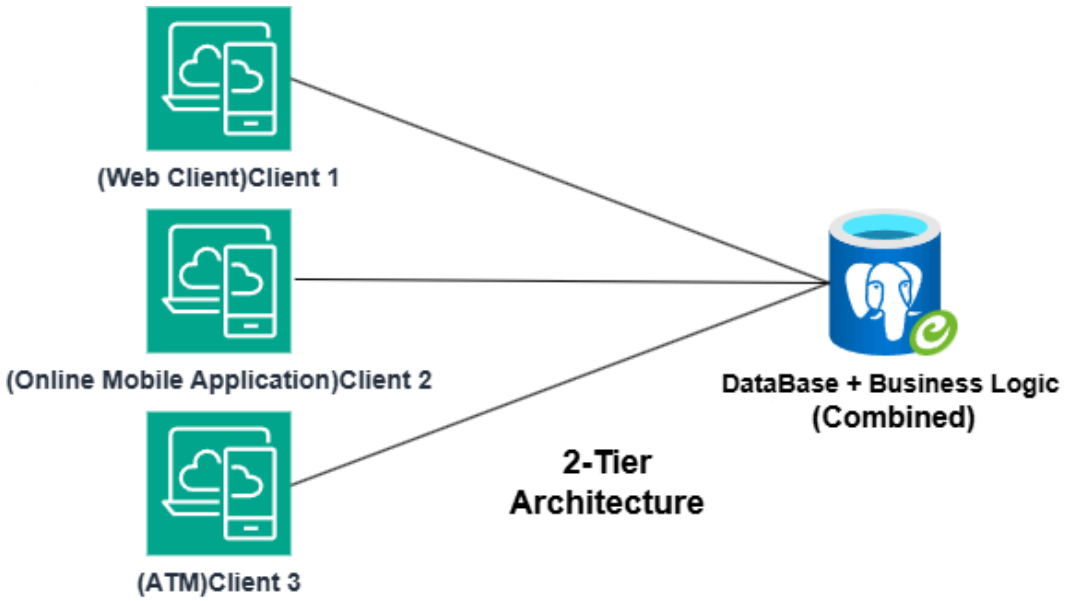
Criteria	2-Tier Architecture	3-Tier Architecture
Security	Weaker. Business logic exposed to clients.	Stronger. Business logic separated from clients; better for enforcing security policies.
Scalability	Poor. Direct DB access leads to bottlenecks as the user base grows.	Excellent. App server can handle thousands of client requests efficiently.
Maintenance	Difficult. Any logic update requires updating client-side apps.	Easier. Business logic changes only affect the middle tier, not clients.

ATM Transactions vs. Online Banking (Why 3-Tier is Better)

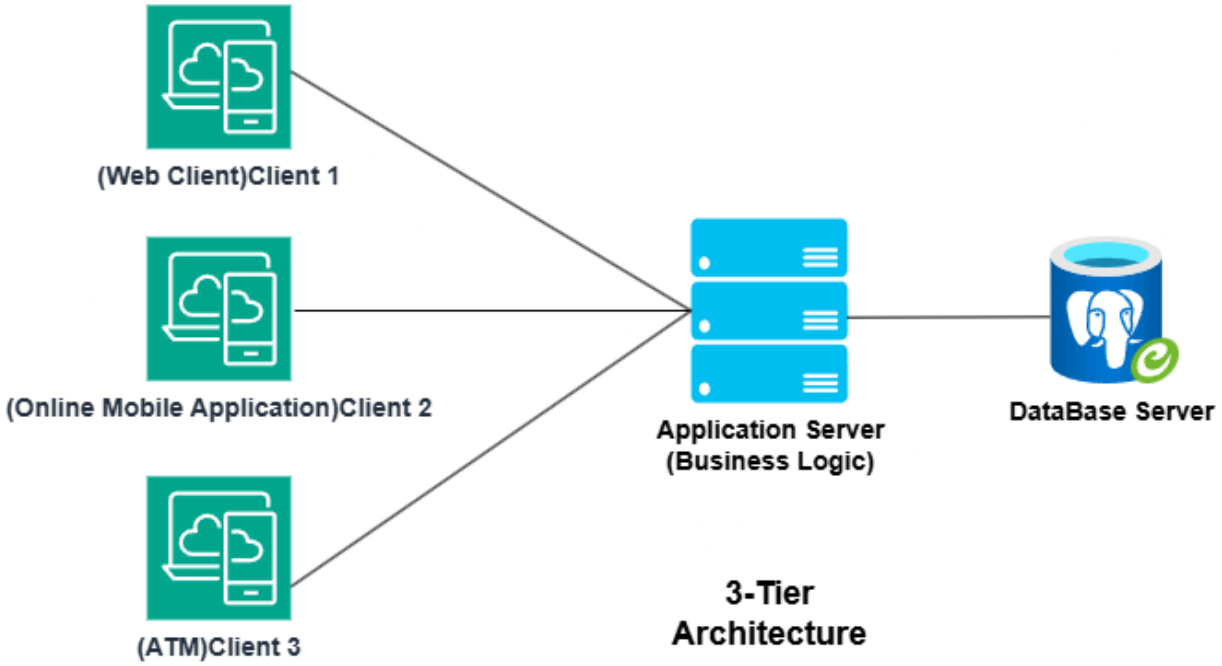
- **ATM Transactions:** Require high security (PIN, card verification) and controlled access to banking logic (balance checks, cash withdrawal rules).
- **Online Banking:** Needs web server, session management, API security, etc.
- **3-Tier System Benefits:**
    - Isolates sensitive business rules from the client.
    - Allows flexible support for multiple interfaces (mobile app, ATM, web portal).
    - Easier to enforce security and transaction consistency.

When 2-Tier Might Suffice

- **Small branch banking system** with limited users (e.g., an internal app for 2–3 tellers) where performance needs and scalability are minimal.
- Simple CRUD (Create, Read, Update, Delete) applications that don't require complex business logic separation.



2-Tier Architecture



3-Tier Architecture

2. Microservices vs. Monolith

Case Study:

An e-commerce app (user auth, inventory, payments) struggles with scaling during Diwali Festival.

Questions:

1. How would microservices improve resilience over a monolith?
2. What's a potential downside?
3. Which GCP service would manage microservices?

Task:

Design a microservice split for the app (3-5 services).

How Microservices Improve Resilience Over Monolith

Monolithic Architecture	Microservices Architecture
All functions (User Auth, Inventory, Payments) are tightly coupled in a single codebase.	Each function is independently deployed, scaled, and managed.
A failure in Payments or Inventory could crash the entire app during Diwali traffic.	Failure in Payments service does not affect User Auth or Inventory; other parts remain operational.
Hard to scale only the "hot" services like Payments.	Can scale specific microservices (e.g., Payments) to handle festival load.

Microservices allow **independent failure handling**, load balancing, and selective scaling — leading to greater overall **system resilience**.

Potential Downside of Microservices

- **Complexity:**
  Managing multiple services increases **deployment complexity**, requires **service discovery**, API gateways, and **robust inter-service communication**.
- **Data Consistency Challenges:**
  Distributed data handling (across services) introduces risks like **eventual consistency**, complicating transactions (e.g., Payments & Inventory sync).

GCP Service Recommendation

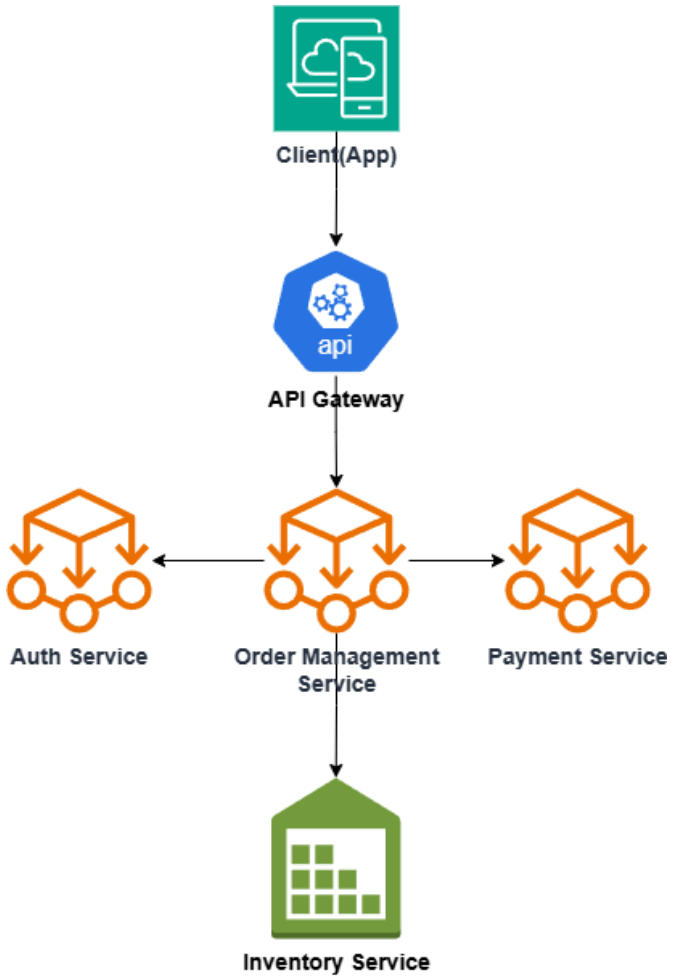
- **Google Kubernetes Engine (GKE):**
  Manages deployment, scaling, and monitoring of containers running microservices.

Other relevant services:

- **Cloud Run:** Serverless microservice hosting.
- **Anthos:** Hybrid/multi-cloud service management.
- **Cloud Pub/Sub:** Inter-service communication.

Suggested Microservice Split for E-Commerce App

Microservice	Responsibilities
1. User Auth Service	User login, registration, session/token management.
2. Product/Inventory Service	Product catalog, stock management, availability checks.
3. Payment Service	Order payment processing, wallet/UPI/credit card handling.
4. Order Management Service	Cart handling, order placement, tracking.
5. Notification Service	Send emails, SMS, app notifications on order/payment updates.



3. Batch vs. Stream Processing using GCP Dataflow

Scenario:
You are analyzing online sales data:
• Batch case: Summarize all sales data of the last month.
• Stream case: Track real-time new orders and alert on large orders instantly.
Tasks:
• Describe when you would use GCP Dataflow for batch and when for stream.
• Identify any other GCP services you would integrate.
Deliverable:
Write a short design note explaining both batch and stream pipelines with GCP services.

1. Batch Processing Design (Monthly Sales Summary)

Use Case: Summarize all online sales data from the past month for reporting and business insights.

GCP Dataflow Usage:

- Run **bounded pipelines** (input dataset is finite: last month's sales data).
- **Pipeline Steps:**
    1. **Source:** Google Cloud Storage (GCS)
    2. **Processing:** GCP Dataflow (Apache Beam) — aggregation, total sales, top products.
    3. **Sink:** BigQuery (for analysis and dashboarding).

2. Stream Processing Design (Real-Time Order Tracking & Alerts)

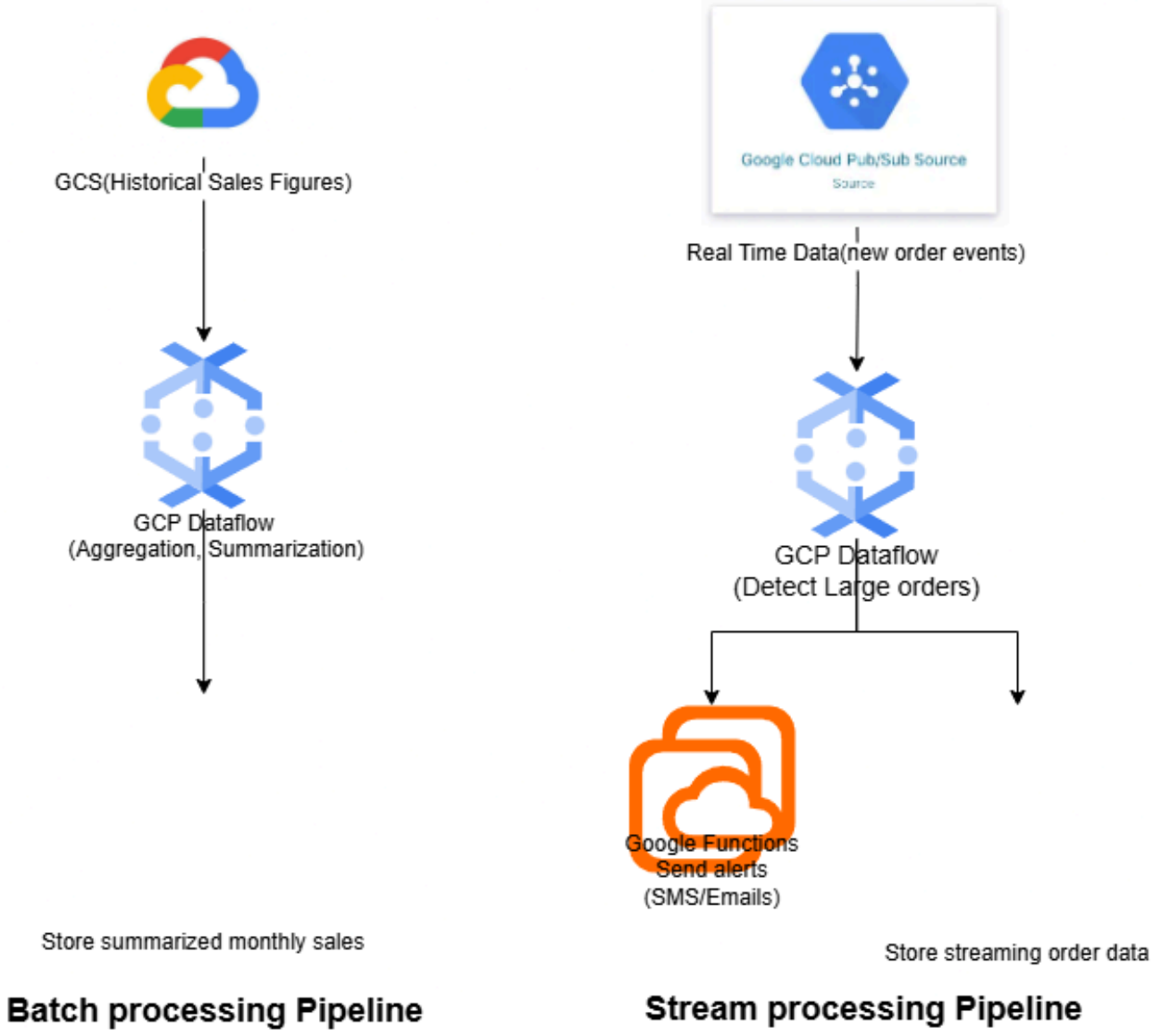
Use Case: Track live orders to detect and instantly alert on large-value orders (e.g., >₹1 lakh).

GCP Dataflow Usage:

- Run **unbounded streaming pipelines** (input is continuous).
- **Pipeline Steps:**
    1. **Source:** Cloud Pub/Sub — receives real-time order events.
    2. **Processing:** GCP Dataflow — filter large orders, trigger alerts.
    3. **Sink:**
        - **Cloud Functions / Cloud Run:** Send alerts via SMS, email.
        - **BigQuery:** Persist processed stream data for analytics.

Aspect	Batch Pipeline	Stream Pipeline
Data Type	Historical (e.g., last month's data)	Real-time (new incoming orders)

Dataflow Mode	Bounded	Unbounded
Source	Google Cloud Storage	Cloud Pub/Sub
Sink	BigQuery	BigQuery, Cloud Functions (for alerts)
Use Case	Monthly sales reports	Instant alerting on large transactions



4. CAP Theorem

Scenario:
You are designing a distributed database for an international chat application.
Tasks:
1. Which two guarantees would you choose out of Consistency, Availability, Partition Tolerance?
2. Justify your choice based on chat app requirements.
3. Suggest an example database (like Cassandra, MongoDB, or CockroachDB).
Deliverable:
Write a short justification (~1 page).

In a **globally distributed chat application**, network partitions (communication breaks between data centers) are inevitable due to factors like latency, outages, or geographical distances. As per the **CAP theorem**, we must tolerate these partitions to ensure that the app continues functioning across all regions — hence **Partition Tolerance is non-negotiable**.

Next, **Availability** is critical:

- Users must be able to **send and receive messages anytime**, regardless of the server or data center health.
- A temporary inconsistency (e.g., seeing a message a moment later on another device) is acceptable in most chat apps — users expect eventual message delivery, not strict instant global consistency.

Consistency (C) can be relaxed to eventual consistency:

- For non-critical messages (casual chats), strict consistency isn't mandatory.
- **Critical operations** (like payment confirmations or password resets) may use additional consistency layers if needed, but regular chat messages can afford slight delays across replicas.

Apache Cassandra is a suitable choice:

- Designed for **high availability and partition tolerance**.
- **Eventual consistency** model aligns with the chat application's needs.
- Used by apps like Facebook Messenger for message storage due to its write-scalability and tolerance for distributed environments.

5. Cloud Deployment & Service Models

Matching Exercise:

Match scenarios to:

• Deployment Models: Public (GCP), Private (On-prem), Hybrid, Multi-cloud

• Service Models: IaaS, PaaS, SaaS

Scenarios:

• A company runs sensitive workloads on-prem but uses GCP for analytics.

• A startup uses BigQuery without managing servers.

Task:

Provide 2 more examples for each model.

1.Deployment Models:

Scenario	Deployment Model
A company runs sensitive workloads on-prem but uses GCP for analytics.	**Hybrid Cloud**
A startup uses BigQuery without managing servers.	**Public Cloud (GCP)**

Two More Examples Each:

- **Public Cloud (e.g., GCP, AWS, Azure)**
    1. Hosting a public website on Google App Engine.
    2. Running a serverless function on AWS Lambda for image processing.
- **Private Cloud (On-Premises)**
    1. A bank running an internal risk calculation system in its own data center.
    2. A government agency hosting classified documents on an in-house OpenStack cloud.
- **Hybrid Cloud**
    1. A retail company processing customer payments on-prem but using GCP's AI tools for customer recommendations.
    2. A hospital storing patient records locally while analyzing anonymized data in Azure Cloud.
- **Multi-Cloud**
    1. A gaming platform using AWS for hosting game servers and GCP for machine learning matchmaking.
    2. A global enterprise using Microsoft Azure for internal employee tools and Oracle Cloud for database services.

2.Service Models:

Scenario	Service Model
A company runs sensitive workloads on-prem but uses GCP for analytics.	**IaaS/PaaS (for analytics on GCP)**
A startup uses BigQuery without managing servers.	**PaaS (BigQuery)**

Two More Examples Each:

- **IaaS (Infrastructure as a Service)**
    1. Creating virtual machines in Google Compute Engine (GCE).
    2. Using AWS EC2 instances for web hosting.
- **PaaS (Platform as a Service)**
    1. Deploying a web app using Google App Engine.
    2. Running Python applications in AWS Elastic Beanstalk.
- **SaaS (Software as a Service)**
    1. Using Google Workspace (Docs, Sheets) for collaboration.
    2. Accessing Salesforce CRM via web browser for customer management.

6. GCP Services for Cloud and Batch Processing

Scenario:
You are given data from sensors on an agricultural farm.
Tasks:
• For batch processing (e.g., seasonal yield analysis), select and justify GCP services.
• For near-real-time anomaly detection (e.g., sudden temperature drop), select and justify GCP services.
• Services to consider: Cloud Storage, Pub/Sub, Dataflow, BigQuery, AI Platform.
Deliverable:
Mapping of the architecture for both batch and real-time.

GCP Architecture Mapping: Agricultural Farm Sensor Data

1. Batch Processing Pipeline (Seasonal Yield Analysis)

Objective: Analyze **historical sensor data** (collected over a season) for trends like crop yield prediction or irrigation planning.

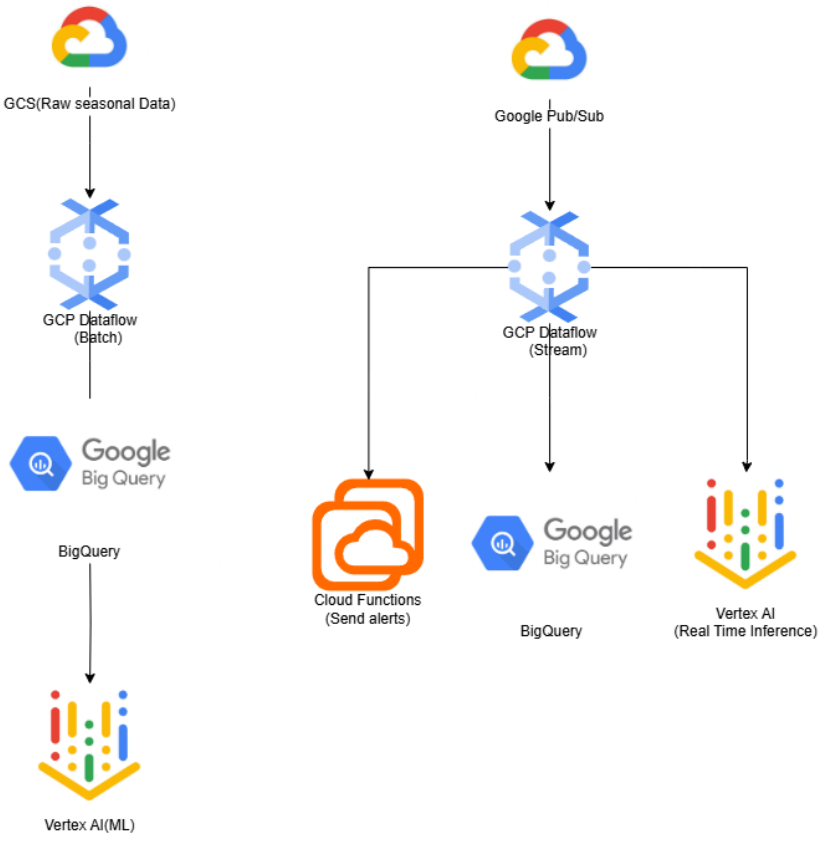
Service	Role & Reason
Cloud Storage	Store raw sensor data files (CSV, JSON) gathered over the season.
Dataflow (Batch)	Process and clean the data (filter noise, handle missing values).

BigQuery	Store processed, structured data for querying and analysis.
AI Platform (Vertex AI)	Optional: Train machine learning models for yield prediction based on historical data.

2. Near-Real-Time Processing Pipeline (Anomaly Detection)

Objective: Detect sudden temperature drops or moisture anomalies to trigger immediate alerts for farm staff.

Service	Role & Reason
Cloud Pub/Sub	Ingest real-time sensor readings (temperature, humidity, etc.).
Dataflow (Streaming)	Filter & detect anomalies (e.g., temperature drops below threshold).
AI Platform (Vertex AI)	Optional: Deploy trained models for real-time anomaly classification.
BigQuery	Log detected anomalies and all sensor data for audit/history.
Cloud Functions / Cloud Run	Send instant alerts (SMS, email) when anomalies are detected.



7. GPU & CUDA Cores

Case Study:
A data science team trains a CNN model on 100GB of images.
Questions:
• Why use GPUs over CPUs here?
• What's the role of CUDA cores?
• How would you distribute this on GCP?

GPU & CUDA Cores for CNN Training: Case Study Explanation

1. Why Use GPUs over CPUs?

- **CNN training** involves millions of **matrix multiplications and convolutions** — highly parallelizable tasks.
- **GPUs** (Graphics Processing Units):
  - Have thousands of cores (vs. CPUs with 4–64 cores).
  - Execute **parallel operations** efficiently on large data tensors (image batches).
  - Drastically reduce training time (from weeks on CPU to hours on GPU).
  - Specifically optimized for **deep learning frameworks** (TensorFlow, PyTorch, etc.).

2. What's the Role of CUDA Cores?

- **CUDA cores** are parallel processing units inside NVIDIA GPUs:
  - Execute **SIMD (Single Instruction Multiple Data)** instructions.
  - Handle deep learning tasks like **convolutions, pooling, activation functions** in parallel across all image pixels/features.
  - Enable frameworks like **TensorFlow, PyTorch** to offload matrix computations to the GPU.
  - Allow **efficient mini-batch processing** (e.g., training on 256 images simultaneously).

3. Distribution on GCP:

Recommended GCP Setup:

GCP Service	Role
AI Platform (Vertex AI)	Fully managed service for training CNN models with GPU support.
Compute Engine (VMs with GPUs)	Custom VM instances (NVIDIA A100/T4/V100 GPUs) for flexible training setups.
Cloud Storage	Store the 100GB of image data.

AI Platform Pipelines	(Optional) Automate preprocessing, training, evaluation steps.

Distributed Training Setup:

1. **Store dataset** in **Cloud Storage (GCS)**.
2. **Provision multiple GPU-enabled VMs** using Compute Engine or Vertex AI.
3. **Use TensorFlow's or PyTorch's multi-GPU/multi-node strategy**
4. **Results** (models, logs) stored back in **Cloud Storage or BigQuery**.

9. Software Design Principles

Scenario: A team is building a food delivery app with 10+ microservices.
Assignment:
• Modularity: Split the app into 3 logical modules (e.g., user auth, order mgmt, payments).
• Scalability: Explain how you'd scale the "order tracking" service during peak hours.
• Maintainability: Identify 2 bad practices (e.g., tightly coupled code) and fixes.
• Reusability: Propose a shared utility module (e.g., logging, error handling).
Deliverable:
1-page design doc + diagram

Software Design Principles for Food Delivery App (Microservices Architecture)

1. Modularity: Logical Modules Split

Module	Microservices Included
User Management	User Authentication, Profile Management, Notification Service
Order Management	Restaurant Listing, Menu Management, Order Processing, Order Tracking
Payment & Billing	Payment Gateway, Billing & Invoicing, Promotions/Discounts Service

2. Scalability: "Order Tracking" Service

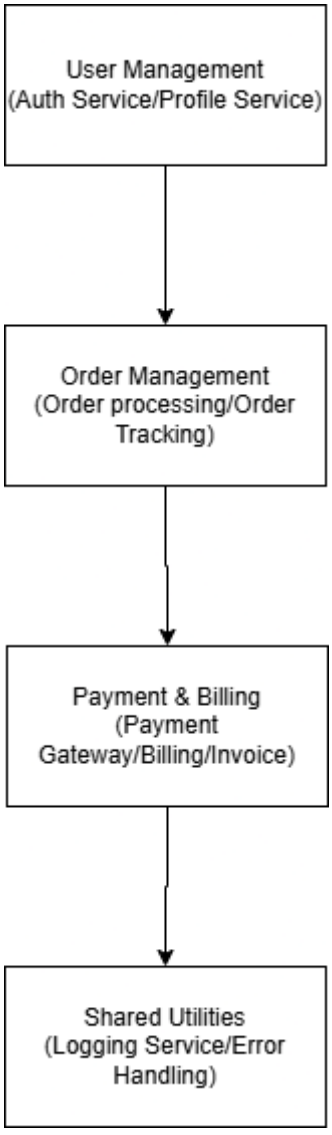
- **Problem:** Order tracking demands heavy use during **peak lunch/dinner hours**.
- **Solution:**
    1. **Auto-scaling via GCP Kubernetes Engine (GKE)** or Compute Engine Managed Instance Groups.
    2. Deploy **dedicated caching** (e.g., Memorystore/Redis) to reduce load from DB for real-time location data.
    3. Use **Pub/Sub** to handle updates from delivery agents efficiently.

3. Maintainability: 2 Bad Practices & Fixes

Bad Practice	Fix
Tightly Coupled Code (e.g., direct calls between services)	**Use REST APIs or gRPC with proper contracts to decouple services.**
Hardcoded Configuration (e.g., DB strings, limits)	**Move to Environment Variables or GCP Secret Manager for manageability.**

4. Reusability: Shared Utility Module

Proposed Module:
Centralized Logging & Error Handling Service
Used by all microservices for consistent logging formats, error tracking (e.g., via Cloud Logging & Error Reporting).



10. Agile vs. Waterfall

Case Study:

A bank is developing a mobile app. Compare SDLC models:

Questions:

• Which model fits better if:

o Requirements are fixed?

o Features evolve weekly?

• List 2 artifacts from each model

• What's the risk of using Agile for taxation/finance software?

Agile vs. Waterfall SDLC Models: Bank Mobile App Case Study

1. Which Model Fits Better?

Scenario	Recommended SDLC Model
Requirements are fixed & well-defined (e.g., core banking system, security features)	**Waterfall** – Best for projects with clear, unchanging scope.
Features evolve weekly (e.g., UI updates, customer feedback-driven features)	**Agile** – Suitable for iterative development, frequent changes.

2. Key Artifacts from Each Model

Waterfall Artifacts	Agile Artifacts
1. Software Requirements Specification (SRS)	**1. Product Backlog (list of evolving features)**
2. Design Document (high-level + low-level design specs)	**2. Sprint Backlog / Burndown Chart (work to be completed in each sprint)**

3. Risk of Using Agile for Taxation/Finance Software

- **Risk:**
  Regulatory Non-Compliance — Frequent, uncontrolled changes in Agile may violate strict regulatory, auditing, or tax calculation standards.
- **Why?**
  - Tax/Finance software requires **traceability**, **documentation**, and **approval-heavy changes** (e.g., legal compliance, banking norms).
  - Agile's flexibility may conflict with **mandatory certification, audits**, or **version-controlled reporting requirements**.

11. OOP Principles
Scenario: A game developer is coding a Character class with Warrior/Mage subclasses.
Tasks:
1. Encapsulation: Hide health property behind getters/setters.
2. Inheritance: Extend Character for Warrior with armor attribute.
3. Polymorphism: Override attack() for Mage (spells vs. swords).
4. Abstraction: Define an abstract move() method.
Deliverable:
Code snippets + explanations.

1. Encapsulation

```
CLASS Character
   PRIVATE health
   METHOD getHealth() RETURN health
   METHOD setHealth(value) IF value >= 0 THEN health = value
END CLASS
```

Explanation:

- `health` is hidden (private).
- Accessed/modified **only through get/set methods**, preventing invalid values.

2. Inheritance

```
CLASS Warrior EXTENDS Character
   ATTRIBUTE armor
END CLASS
```

Explanation:

- `Warrior` **inherits** `Character` features.
- Adds new feature: **armor**.

3. Polymorphism

```
CLASS Mage EXTENDS Character
   METHOD attack() PRINT "Mage casts spell"

CLASS Warrior EXTENDS Character
   METHOD attack() PRINT "Warrior swings sword"
```

Explanation:

- Same method `attack()` behaves differently for **Mage and Warrior**.
- Demonstrates **polymorphism (many forms)**.

4. Abstraction

```
ABSTRACT CLASS Character
   ABSTRACT METHOD move()
```

Explanation:

- `move()` is **abstract** — no code in base class.
- All subclasses **must define their own `move()`** method.

12. Compare Von Neumann and Harvard Architectures
Objective:
Understand and compare the two architectures.
Tasks:
• Define the Von Neumann architecture with a neat block diagram.
• Define the Harvard architecture with a block diagram.
• Make a comparison table based on aspects like data and instruction separation, memory organization, speed, and use cases.
• Explain with examples where each architecture is typically used (e.g., microcontrollers vs. general-purpose computers).
• Summarize why understanding memory-data separation matters for data science models.

Von Neumann vs. Harvard Architecture

1. Von Neumann Architecture

Definition:
Single memory space is shared for both **instructions and data**. CPU fetches either instruction or data one at a time (sequentially).

```
        CPU (ALU + CU)
              ↓
Shared Memory(Instructions+Data)
```

2. Harvard Architecture

Definition:
Separate memory for **instructions and data**. The CPU can fetch instructions and data **simultaneously**, improving speed.

```
          CPU (ALU + CU)
              ↓
Instructions Memory + Data memory
```

3. Comparison Table

Aspect	Von Neumann	Harvard
Instruction & Data Separation	**No (Shared)**	**Yes (Separate)**
Memory Organization	**Single memory space**	**Separate memory for each**
Speed	**Slower (bottleneck due to single bus)**	**Faster (parallel access possible)**
Use Cases	**General-purpose computers, laptops, desktops**	**Microcontrollers, DSPs (Digital Signal Processors)**

4. Typical Examples

- **Von Neumann:**Laptops, Desktops, Servers, Smartphones (general computing).
- **Harvard:**Microcontrollers (e.g., AVR, PIC), ARM Cortex-M (embedded systems), DSP chips.

5. Why Memory-Data Separation Matters in Data Science Models

-Large ML/DL models (e.g., neural networks) perform heavy parallel matrix operations.
-GPUs and TPUs use Harvard-like designs to allow simultaneous fetch of weights and activation values, improving processing speed and efficiency.
-Poor separation can cause memory bottlenecks (like the Von Neumann bottleneck), slowing model training/inference.

13. Study Multithreading and Multicore CPUs

Objective:

Learn how modern CPUs handle parallel tasks.

Tasks:

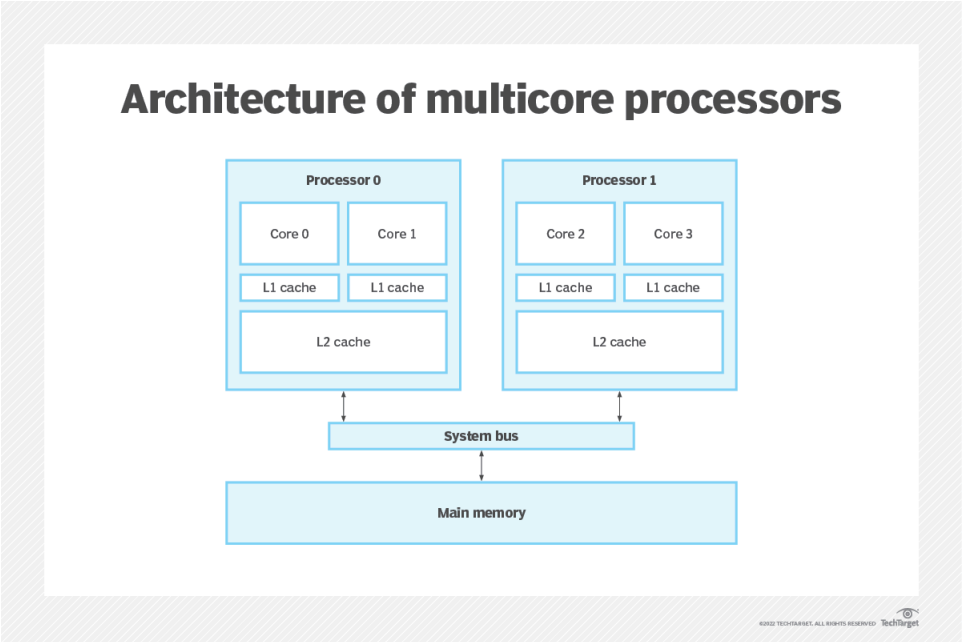
• Define multithreading and multicore concepts with diagrams.

• Explain how multithreading helps in handling data loading and preprocessing in machine learning pipelines.

• Research and list 2 CPU models with multithreading and multi core capabilities (e.g., Intel i7 vs AMD Ryzen 7).

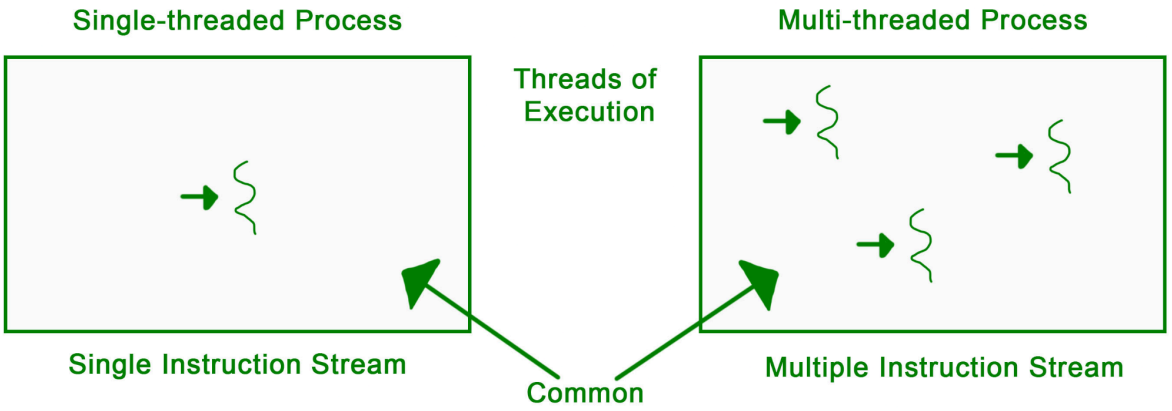
1. Definitions with Textual Diagrams

Multicore CPU:A single CPU chip with multiple processing cores, each capable of executing its own task independently.



Benefit: True parallelism — multiple tasks processed simultaneously.

Multithreading: Each core handles multiple threads (lightweight tasks) by switching between them rapidly or running them simultaneously (if hardware allows).



Single Thread and Multi Thread Process

Benefit: Better CPU utilization even if only one or two cores are busy.

2. Use of Multithreading in Machine Learning Pipelines

Data Loading & Preprocessing:

- Multithreading allows the CPU to read data, decode images, perform augmentations while other cores run model training.
- **Example:**In PyTorch/TensorFlow: `num_workers > 0` for `DataLoader` enables parallel data loading.

Prevents GPU Idle Time:

- If data preparation is slow (single thread), GPU waits idle, reducing efficiency.
- With multithreading, data is ready when the GPU needs it.

3. Real CPU Examples

CPU Model	Cores	Threads (via Hyper Threading/SMT)
Intel Core i7-12700K	**12 (8P+4E)**	**20 Threads**
AMD Ryzen 7 7800X3D	**8**	**16 Threads**

Both support multicore + multithreading to boost parallel computation — great for data science, video processing, and gaming.

14. Analyze Memory Access Latency

Objective:

Understand the access times of different memory types.

Tasks:

• Research access times for Registers, Cache, RAM, SSD, HDD (list in a table).

• Explain with examples how slow memory access can bottleneck data science workflows like model training or feature engineering.

Memory Access Latency Analysis

1. Memory Access Time Table

Memory Type	Access Time (Approximate)	Description
Registers	0.3 – 1 ns	Fastest, inside CPU cores.
Cache (L1)	1 – 5 ns	Very fast, small size, near CPU.
Cache (L2)	5 – 20 ns	Slower than L1, but bigger.
Cache (L3)	20 – 50 ns	Shared by cores, higher latency.
RAM (DRAM)	50 – 100 ns	Main system memory.
SSD (NVMe)	50 – 100 µs (microseconds)	Fast storage but much slower than RAM.
HDD	5 – 10 ms (milliseconds)	Slowest; mechanical seek latency.

2. Impact on Data Science Workflows

Example 1: Model Training (TensorFlow/PyTorch)

- **Registers & Cache (L1/L2):**
  Used for **intermediate calculations (matrix ops, dot products)**.
  Fast register/cache access keeps GPU/CPU utilization high.
- **RAM:**
  Stores **large tensors & mini-batches** for training.
  If data does not fit in RAM — **paging to SSD/HDD happens**, causing significant slowdown.
- **SSD/HDD:**
  Training data often fetched from **disk at epoch start**.
  Using HDD causes **slow data loading** (5–10 ms per seek) vs SSD (100 µs), increasing **training time**.

Example 2: Feature Engineering (Pandas/Spark)

- If datasets cannot fit in **RAM**, Spark spills to **disk (SSD/HDD)**:
    - Causes massive **I/O latency** — bottlenecking joins, filters.
    - Example: Shuffling in Spark depends heavily on disk speed — **slow disks slow down pipeline execution**.

3. Summary: Why Understanding Latency Matters

Optimized model training pipelines require understanding of memory hierarchy to:

- **Prefetch data into RAM/cache**.
- Use **fast SSDs** instead of HDD for storage.
- Avoid swapping to disk from RAM.

Feature engineering in big data systems (Spark, Dask) suffers without enough memory:

- Slow disk I/O = severe performance loss.

15. Understand Throughput, Latency, and IOPS

Objective:
Clarify the key performance metrics for storage systems.
Tasks:
• Define Throughput (MBps), Latency (ms), and IOPS.
• Give real-world examples:
o When is Throughput more important? (e.g., copying large files)
o When is IOPS more important? (e.g., random database queries)
• Create two sample scenarios:
1. Large machine learning datasets (files >1 GB each).
2. High-frequency financial transaction database.
• For each, specify which storage metric matters most and why.

Throughput, Latency, and IOPS Explained

1. Key Definitions:

Metric	Definition	Units
Throughput	Amount of data transferred per second. Measures **bulk data movement** capability.	MBps (Megabytes/sec)
Latency	Time to complete a single request (from initiation to completion). Indicates **response time**.	ms (milliseconds)
IOPS	Input/Output Operations Per Second. Measures **how many read/write ops can happen per second**.	Operations/sec

2. Real-World Examples:

When Throughput Matters	When IOPS Matters
Copying/moving large files (e.g., ML datasets)	**Performing lots of small, random reads/writes (databases, OLTP systems)**
Video editing workloads	**Serving API calls on NoSQL DB (e.g., MongoDB, Cassandra)**
Streaming large data (e.g., backups)	**Handling financial transactions, stock trading systems**

3. Sample Scenarios:

Scenario 1: Large Machine Learning Datasets

- **Description:**
  Processing and loading big CSV/TFRecord files (>1 GB each) into memory for model training.
- **Most Important Metric:**
  Throughput (MBps) — Fast movement of large files from SSD/Cloud Storage to RAM speeds up training pipelines.
- **Why?**
  High throughput reduces wait time before training begins.

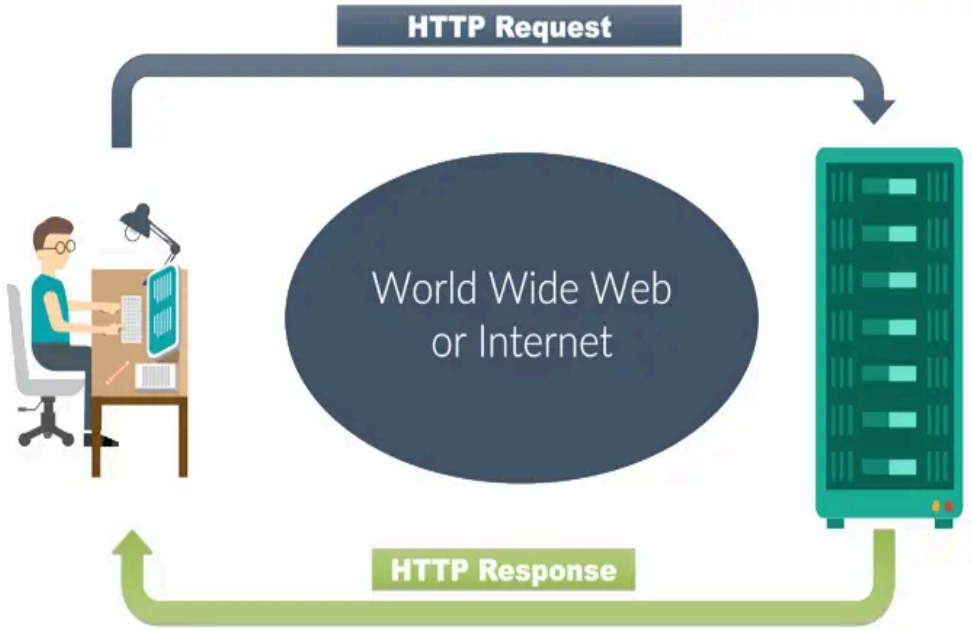
Scenario 2: High-Frequency Financial Transaction Database

- **Description:** Banking app handling millions of small, fast transaction updates/queries.
- **Most Important Metric:**
  IOPS (Operations/sec) — Critical for handling many tiny, random database reads/writes efficiently.
- **Why?**
  High IOPS ensures low transaction delay and prevents system bottlenecks under heavy load.

16. HTTP vs HTTPS Communication Flow

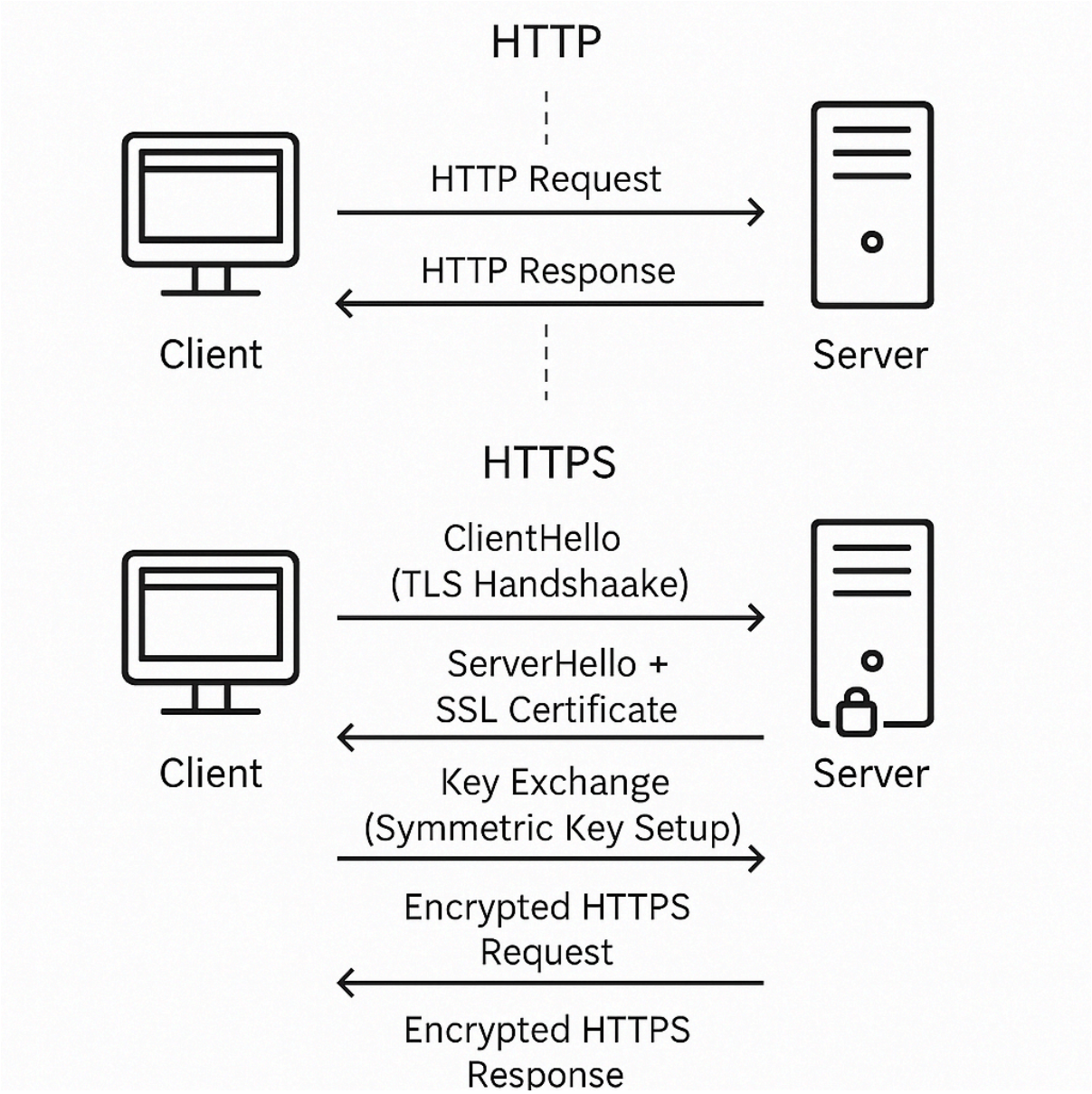
Create a diagram (or describe in words) for the typical steps in:
• HTTP request-response cycle
• HTTPS request-response cycle (emphasizing SSL/TLS encryption)
Answer in your document:
• How HTTPS ensures security
• What happens if SSL Certificate is missing or invalid

1. HTTP Request-Response Cycle (Insecure)



Data is sent as plain text — vulnerable to eavesdropping, tampering, MITM attacks.

2. HTTPS Request-Response Cycle (Secure via SSL/TLS)



Data encrypted using SSL/TLS — protects against interception, MITM, and tampering.

Detailed Steps in HTTPS Communication:

1. ClientHello (TLS Handshake Initiation):

- **Browser (client) sends a ClientHello message to the server.**
- **This includes:**
  - TLS version supported.
  - List of encryption algorithms (cipher suites) it can use.
  - Random number for key generation.

2. ServerHello + SSL/TLS Certificate:

- The server responds with a ServerHello message.
- **Sends:**
  - Chosen cipher suite.
  - Random number.
  - Server's SSL/TLS certificate (includes public key).

3. Certificate Validation:

- The client checks the server's certificate:
  - Is it issued by a trusted Certificate Authority (CA)?
  - Is the certificate valid (not expired/revoked)?
  - Is the domain in the certificate matching the server’s domain?
- If any check fails — browser shows security warnings.

4. Key Exchange & Symmetric Key Setup:

- The client and server exchange keys securely (using asymmetric cryptography).
- They agree on a shared symmetric key (used to encrypt the session).
- Symmetric encryption is faster and is used for the rest of the communication.

5. Encrypted Communication:

- All subsequent HTTP requests/responses (GET, POST, etc.) are encrypted with the symmetric key.
- Example: Login details, form data, cookies — all hidden from eavesdroppers.

6. Session Integrity Check:

- Data integrity is ensured using Message Authentication Codes (MAC).
- Detects tampering during transmission.

3. How HTTPS Ensures Security:

Encryption:
All data transferred is **encrypted** using **TLS (Transport Layer Security)** — prevents anyone from reading the data in transit.

Authentication:
Browser verifies server identity using **SSL Certificate** — ensures you're talking to the real website, not an attacker.

Integrity:
TLS ensures that data is **not altered** during transit — via hashing and cryptographic checks.

4. What Happens If an SSL Certificate is Missing/Invalid?

Condition	Browser/User Behavior
SSL Certificate Missing	Browser blocks access or shows "Not Secure" warning.
SSL Certificate Expired/Invalid	Browser shows security warning ("Your connection is not private") — prevents data exchange unless the user ignores the risk.

Impact: Users are discouraged from visiting the site; APIs might reject connections.