# Async processing

**Async Processing (অ্যাসিনক্রোনাস প্রসেসিং)** হলো এমন একটি পদ্ধতি যেখানে একটি কাজ শুরু করে সেটির শেষ হওয়ার জন্য বসে না থেকে অন্য কাজ চালিয়ে যাওয়া হয়।

একটি বাস্তব উদাহরণ দিলে বিষয়টি একদম পরিষ্কার হয়ে যাবে:

- **Synchronous (সিংক্রোনাস):** আপনি একটি রেস্টুরেন্টে গিয়ে কাউন্টারে অর্ডার দিলেন। যতক্ষণ না আপনার খাবার রান্না শেষ হচ্ছে, আপনি কাউন্টারে দাঁড়িয়ে থাকলেন এবং আপনার পেছনে থাকা অন্য কেউ অর্ডার দিতে পারল না।
- **Asynchronous (অ্যাসিনক্রোনাস):** আপনি অর্ডার দিলেন, আপনাকে একটি টোকেন দেওয়া হলো এবং আপনি গিয়ে টেবিলে বসলেন। বাবুর্চি খাবার রান্না করছে, আর এর মধ্যে কাউন্টারে অন্যরাও অর্ডার দিতে পারছে। খাবার তৈরি হলে আপনাকে ডেকে নেওয়া হবে।

### ১. Async Processing কেন প্রয়োজন?

আধুনিক ওয়েব অ্যাপ্লিকেশনগুলোতে কিছু কাজ থাকে যা অনেক সময় নেয়, যেমন:

- বড় কোনো ফাইল আপলোড করা।
- হাজার হাজার মানুষকে ইমেইল পাঠানো।
- অন্য কোনো ওয়েবসাইট বা API থেকে ডাটা নিয়ে আসা।
- ভিডিও প্রসেসিং বা ইমেজ কম্প্রেস করা।

যদি এই কাজগুলো **Synchronous** পদ্ধতিতে করা হয়, তবে ইউজার যতক্ষণ কাজ শেষ না হবে ততক্ষণ স্ক্রিন লোডিং দেখবে এবং অ্যাপটি ব্যবহার করতে পারবে না। **Async** পদ্ধতিতে ইউজার সাথে সাথে রেসপন্স পায় এবং ব্যাকগ্রাউন্ডে কাজগুলো হতে থাকে।

### ২. এটি কীভাবে কাজ করে? (Workflow)

Async প্রসেসিং সাধারণত ৩টি ধাপে কাজ করে:

1. **Request:** ইউজার একটি কাজ করার অনুরোধ পাঠায়।
2. **Acknowledgment:** সার্ভার সাথে সাথে বলে, "আমি তোমার অনুরোধ পেয়েছি, কাজ শুরু করছি (যেমন: HTTP 202 Accepted status)"। ইউজার তার কাজ চালিয়ে যেতে পারে।
3. **Completion:** ব্যাকগ্রাউন্ডে কাজ শেষ হলে সিস্টেম কোনো নোটিফিকেশন বা **Webhook**-এর মাধ্যমে ইউজারকে জানিয়ে দেয়।

## ৩. Async প্রসেসিংয়ের মূল টুলস

Async কাজগুলো করার জন্য সাধারণত মাঝখানে একটি **Message Queue** ব্যবহার করা হয় যা আমরা একটু আগে আলোচনা করেছি:

- **Task Queues:** RabbitMQ বা Redis (BullMQ/Sidekiq) ব্যবহার করে কাজগুলোকে লাইনে সাজানো হয়।
- **Workers:** আলাদা সার্ভার বা প্রসেস (Worker) যারা ওই লাইন থেকে কাজগুলো নিয়ে একটি একটি করে শেষ করে।
- **Promises/Callbacks:** প্রোগ্রামিং ল্যাঙ্গুয়েজের ভেতরে (যেমন JavaScript-এর async/await) কোডকে ব্লক না করে চালানোর পদ্ধতি।

## ৪. Async Processing-এর সুবিধা ও অসুবিধা

| **সুবিধা** | **অসুবিধা** |
|---|---|
| **User Experience:** অ্যাপ অনেক ফাস্ট মনে হয় কারণ ইউজারকে অপেক্ষা করতে হয় না। | **Complexity:** কোড লেখা এবং ডিবাগ করা কিছুটা কঠিন হয়ে পড়ে। |
| **Scalability:** সার্ভারের ওপর হঠাৎ চাপ পড়লে সব কাজ কিউতে জমা থাকে, সার্ভার ক্রাশ করে না। | **Data Consistency:** কাজ কখন শেষ হবে তার কোনো গ্যারান্টি নেই, তাই ডাটা আপডেট হতে দেরি হতে পারে। |

| **সুবিধা** | **অসুবিধা** |
|---|---|
| **Efficiency:** সার্ভারের রিসোর্স অপটিমাইজড ভাবে ব্যবহার করা যায়। | **Monitoring:** ব্যাকগ্রাউন্ডে কোন কাজ ফেইল করল তা ট্র্যাক করার জন্য আলাদা সিস্টেম লাগে। |

### ৫. বাস্তব জীবনের উদাহরণ

ধরুন আপনি Facebook-এ একটি বড় ভিডিও আপলোড দিলেন।

- ভিডিওটি আপলোড হওয়ার সাথে সাথে ফেসবুক আপনাকে বলে না যে "দাঁড়ান, আগে প্রসেসিং শেষ হোক তারপর আপনি অন্য কাজ করবেন"।
- ফেসবুক ভিডিওটি নিয়ে নেয় এবং ব্যাকগ্রাউন্ডে প্রসেস করতে থাকে। আপনি তখন অন্য সবার পোস্ট দেখতে পারেন।
- প্রসেসিং শেষ হলে আপনাকে একটি নোটিফিকেশন দেয়: "Your video is ready to watch." — এটিই হলো **Async Processing**।

সারাংশ:

আপনার অ্যাপ্লিকেশনের পারফরম্যান্স এবং ইউজার এক্সপেরিয়েন্স ভালো করতে হলে যে কাজগুলো ১ সেকেন্ডের বেশি সময় নেয়, সেগুলোকে Async Processing-এ পাঠিয়ে দেওয়া উচিত।

### ⏳ Async Processing কী?

**Asynchronous Processing** মানে—

কোনো কাজ **immediately response দিয়ে**,
heavy কাজটা **background-এ পরে complete করা**।

User → Request

Server → Response (FAST) ✅

↓

Background Task (SLOW)

👉 User অপেক্ষা করবে না ❌
👉 System responsive থাকবে ✅

## ❌ Synchronous Processing (Problem)

User → Request

Server → Do heavy task (5 sec)

Server → Response

**সমস্যা:**

- Slow response
- Timeout
- Server block
- Bad user experience 😵

## ✅ Async Processing (Solution)

User → Request

Server → "Accepted" response

↓

Queue

↓

Worker

## 🧠 Async কেন দরকার?

✓ Scalability
✓ Performance
✓ Fault tolerance
✓ Better UX

## 🔥 Real-World Examples

### 📦 Order System

Place Order

→ Response: Order placed ✅

→ Payment processing (async)

→ Inventory update (async)

→ Email/SMS (async)

### 📸 Image Upload

Upload image

→ Response instantly

→ Resize / compress (async)

### 📩 Email / SMS

Signup → Response OK

→ Verification email (async)

## 🧱 Async Architecture (Typical)

Client

↓

API Server

↓

Message Queue (Kafka / RabbitMQ / SQS)

↓

Worker / Consumer

↓

Database / External API

## 🧰 Async Processing Tools

### Message Queue

- **Kafka**
- **RabbitMQ**
- **AWS SQS**
- **Redis Queue**

### Worker

- Celery (Python)
- BullMQ (Node.js)
- Sidekiq (Ruby)

## 📨 Queue কী করে?

- Task store করে
- FIFO / partitioned
- Retry support

- Load smooth করে

## 🧠 Key Concepts (Interview MUST)

### 1. Producer

👉 Task তৈরি করে (API Server)

### 2. Consumer

👉 Task process করে (Worker)

### 3. Acknowledgement

👉 Task successful হলে ACK

### 4. Retry Mechanism

- Temporary failure হলে retry
- Exponential backoff

### 5. Dead Letter Queue (DLQ)

- বারবার fail হলে
- আলাদা queue তে পাঠানো

## ⚖️ Async vs Sync

| **Topic** | **Sync** | **Async** |
|---|---|---|
| Response time | Slow | Fast |
| Scalability | Low | High |

| | | |
|---|---|---|
| Failure impact | High | Low |
| Complexity | Simple | Medium |
| UX | Bad | Good |

### 🧠 Async + CAP theorem

Async processing সাধারণত:

- **AP systems**
- Eventual consistency

### 🛑 Challenges

❌ Debugging harder
❌ Data consistency
❌ Order guarantee
❌ Duplicate messages

### 🛡️ Solutions

✓ Idempotency key
✓ Exactly-once / At-least-once
✓ Monitoring
✓ Logging & tracing

### 🧠 Interview Golden Lines ⭐

“Async processing improves system responsiveness by decoupling request handling from heavy operations.”

“Message queues help in smoothing traffic spikes and ensuring reliability.”

### 📝 Simple Memory Trick

User waits = BAD

Queue waits = GOOD