## **1. What are Performance Bottlenecks?**

* A **bottleneck** is any part of your system slowing down overall performance.
* If one component is overloaded, the entire request pipeline suffers — even if other parts are idle.
* In servers, bottlenecks usually fall into two categories:
  + **CPU-bound** — Processor is the limiting factor.
  + **I/O-bound** — Waiting on external resources (disk, DB, network).

## **2. CPU-Bound vs I/O-Bound Workloads**

### **CPU-Bound**

* **Definition:** System spends most time doing computations (math-heavy tasks, encryption, image processing, data compression).
* **Symptoms:**
  + High CPU usage (close to 100%)
  + Low memory and disk activity
  + Increasing request latency even without heavy I/O
* **Examples:**
  + Sorting large datasets in memory
  + Cryptographic hashing
  + Video encoding
* **Fixes:**
  + Optimize algorithms (lower time complexity)
  + Use parallel processing (worker threads, clusters, multiple cores)
  + Use compiled modules (C++ addons, WebAssembly) for heavy math
  + Scale horizontally with more CPU capacity

### **I/O-Bound**

* **Definition:** System spends most time waiting on Input/Output (network calls, DB queries, file reads).
* **Symptoms:**
  + CPU usage is low but latency is high
  + High disk/network wait times
  + Threads appear “idle” but requests are slow
* **Examples:**
  + API calls to external services
  + Database queries
  + File uploads/downloads
* **Fixes:**
  + Use asynchronous I/O (Node.js non-blocking, async/await)
  + Add caching (Redis, Memcached)
  + Batch I/O operations
  + Use connection pooling and faster storage
  + Reduce unnecessary network hops

## **3. How to Identify Which One You Have**

### **Step 1: Check CPU usage**

* **Linux tools:** top, htop, mpstat
* **Node.js specific:** clinic doctor, Chrome DevTools profiling
* **If CPU ~90% or higher → CPU-bound.**

### **Step 2: Check I/O wait**

* **Linux tools:** iostat, iotop, dstat
* **If CPU is low but requests are slow → I/O-bound.**

### **Step 3: Profile the code**

* Use profilers to see which functions consume time.
* For Node.js:

node --prof app.js  
node --prof-process isolate-\*.log

* If the time is spent *inside JavaScript computations* → CPU-bound.
* If the time is spent *waiting on network or DB* → I/O-bound.

## **4. Mixed Workloads**

* Many real-world apps have **both types of bottlenecks**.
* Example: An API server may be CPU-bound for image processing and I/O-bound for DB queries.
* **Fix each separately:**
  + Use worker threads for CPU tasks.
  + Use async patterns for I/O tasks.