## **1. What is Latency vs Throughput?**

### **Latency**

* **Definition:** Time taken for a single request to travel through a system (from input to response).
* **Analogy:** *How long does it take for a single car to cross a bridge?*
* **Metric:** Usually measured in **milliseconds (ms)** or **microseconds (µs)**.
* **Goal:** *Lower latency = faster response per request.*

### **Throughput**

* **Definition:** Number of requests the system can handle per unit time.
* **Analogy:** *How many cars can cross the bridge per minute?*
* **Metric:** Usually measured in **requests per second (RPS)** or **transactions per second (TPS)**.
* **Goal:** *Higher throughput = more requests handled simultaneously.*

## **2. How to Measure Them**

### **Measuring Latency**

* **End-to-end measurement:**
  + Time between sending a request and receiving a response.
  + Tools: curl -w "%{time\_total}\n" -o /dev/null -s <http://server>
  + Profiling tools: Chrome DevTools (for frontend), Postman, Apache Benchmark (ab), JMeter.
* **Percentile Latency:**
  + p50 (median) → Typical latency.
  + p95, p99 → Tail latency (important in real systems).

### **Measuring Throughput**

* **Load testing:** Send many requests per second and count how many succeed.
* Tools: Apache Benchmark (ab), JMeter, Locust, k6, Artillery.
* Example:

ab -n 1000 -c 50 <http://localhost:3000/>

* + -n 1000 → total requests
  + -c 50 → concurrent requests

## **3. How to Optimize Latency and Throughput**

### **Optimizing Latency**

1. **Reduce server response time** — Optimize code logic and database queries.
2. **Use caching** — Avoid repeated expensive computations or DB calls.
3. **Use faster data formats** — JSON vs Protobuf vs MsgPack.
4. **Minimize network hops** — Reduce unnecessary API calls, batch them if possible.
5. **CDN for static assets** — Place content closer to users.

### **Optimizing Throughput**

1. **Horizontal scaling** — Add more servers behind a load balancer.
2. **Database optimizations** — Sharding, replication, connection pooling.
3. **Asynchronous processing** — Use queues for tasks that don't need immediate response.
4. **Microservices and service isolation** — Avoid bottlenecks in monolithic code.
5. **Rate limiting and backpressure control** — Prevent overload.

## **4. Trade-offs**

* Sometimes **reducing latency (fast per request)** can hurt **throughput (requests per second)**.
* Example: Adding strict consistency checks (e.g., database locks) improves correctness but slows both latency and throughput.
* Good systems **balance** both depending on requirements (e.g. banking → prioritize latency consistency, social apps → prioritize throughput).