Here are **bullet-point notes** for designing a **Rate Limiter**, a common component in distributed systems to control request flow and prevent abuse:

**🧩 1. Problem Statement**

* Limit the number of requests a user/client can make in a given time window.
* Prevent abuse, DDoS attacks, and ensure fair usage of system resources.

**📎 2. Requirements**

**✅ Functional**

* Allow up to **N requests per time window** (e.g., 100 requests per minute).
* Identify clients via IP, API key, or user ID.
* Optionally support different rules per client (e.g., premium vs free users).

**❌ Non-Functional**

* Low latency (inline with request path).
* High availability and fault-tolerance.
* Accuracy vs performance tradeoffs.
* Scalable and distributed.

**⚙️ 3. Rate Limiting Algorithms**

**✅ 1. Fixed Window**

* Count requests in fixed time windows (e.g., 60 seconds).
* Simple, but can cause bursts at window edges.

**✅ 2. Sliding Window Log**

* Keep timestamped logs of requests and prune old ones.
* Accurate, but high memory and compute cost.

**✅ 3. Sliding Window Counter**

* Approximation of sliding window using partial counters.
* Efficient and smoother than fixed window.

**✅ 4. Token Bucket**

* Tokens refill at a fixed rate; each request consumes one token.
* Allows bursts while enforcing average rate.
* Common in APIs and network traffic shaping.

**✅ 5. Leaky Bucket**

* Queue-like behavior with a fixed outflow rate.
* Smooth request processing, ideal for consistent load.

**📦 4. Data Structures**

* **Fixed Window**: Map<ClientID, Counter + Timestamp>
* **Sliding Log**: Map<ClientID, List<Timestamps>>
* **Token Bucket**: Map<ClientID, {tokens, lastRefillTime}>

**💾 5. Storage & State**

* **In-memory**: Fast (Redis, Memcached, local memory) but not durable.
* **Distributed**: Redis with TTL or Lua scripts for atomic updates.
* **Cluster-aware**: Use consistent hashing or sticky sessions to minimize cross-node communication.

**🔐 6. Identification**

* Use IP address, API key, OAuth token, or user ID as rate limit key.

**🌐 7. Distributed Considerations**

* **Centralized**: Single Redis cluster for rate tracking.
* **Local with sync**: Sync state periodically across nodes.
* **Rate Limiting Proxy**: Deploy at edge (API Gateway, NGINX with Lua).

**🧪 8. API Design**

http

CopyEdit

GET /resource

→ 429 Too Many Requests (if limit exceeded)

Headers:

X-RateLimit-Limit: 100

X-RateLimit-Remaining: 5

X-RateLimit-Reset: 1695152023

**📊 9. Monitoring & Logging**

* Log client key, usage count, and rejections.
* Alert on suspicious activity or spike patterns.

**🚧 10. Use Cases**

* API rate limiting
* Login attempt control
* DDoS protection
* Per-user/per-service throttling

**🔐 11. Security & Abuse Handling**

* Throttle based on failed login attempts/IP.
* Blacklist IPs or API keys after repeated limits exceeded.
* CAPTCHA on abusive behavior.

**💡 12. Bonus Features**

* Burst tokens (for premium clients).
* Dynamic limit configuration via admin dashboard.
* Exponential backoff or retry headers for clients.