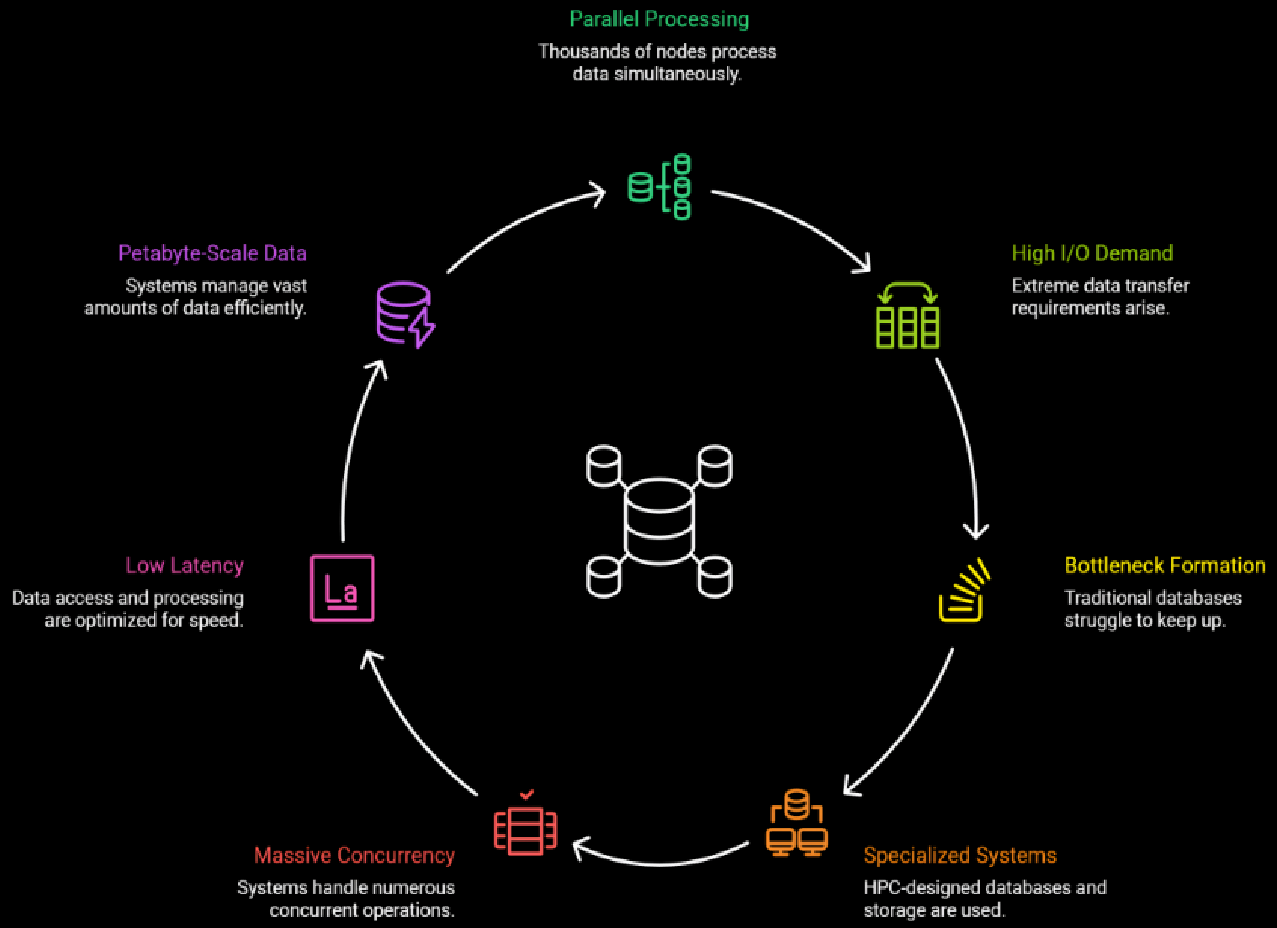
DIFFERENT TYPE OF DATABASES USED IN HIGH PERFORMANCE COMPUTING

Presented By :

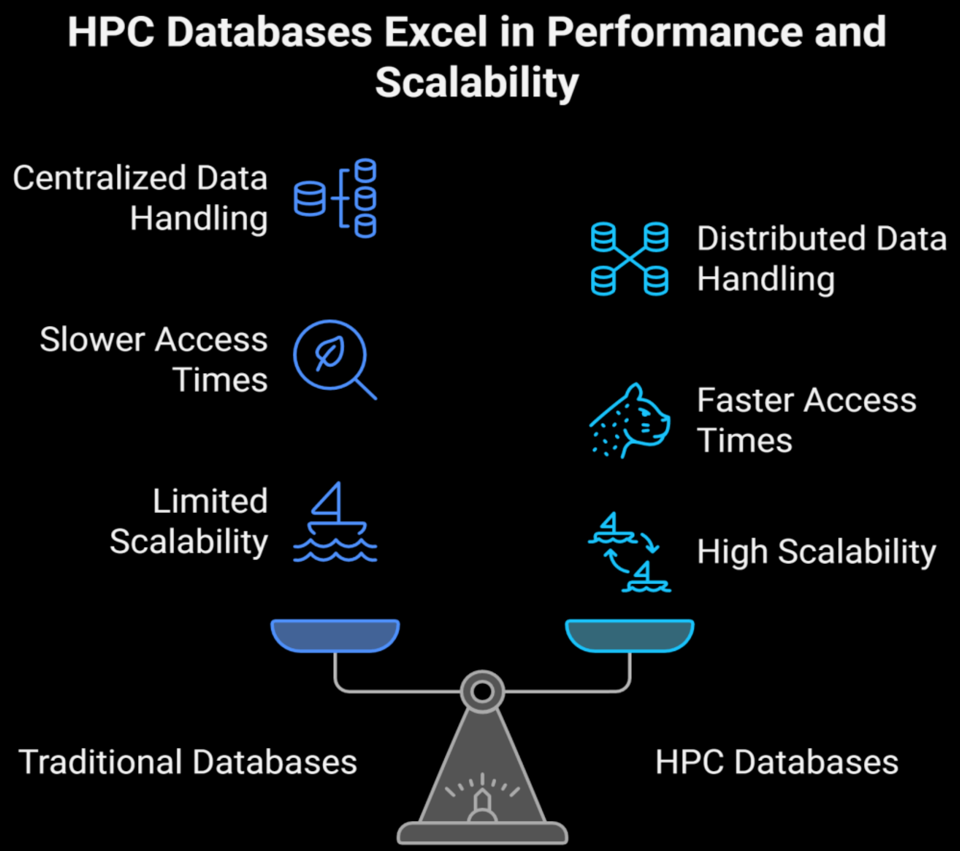
Satyam Kasar


Parallel Processing
Thousands of nodes process data simultaneously.
High I/O Demand
Extreme data transfer requirements arise.
Bottleneck Formation
Traditional databases struggle to keep up.
Specialized Systems
HPC-designed databases and storage are used.
Massive Concurrency
Systems handle numerous concurrent operations.
Low Latency
Data access and processing are optimized for speed.
La
Petabyte-Scale Data
Systems manage vast amounts of data efficiently.

INTRODUCTION

- In High Performance Computing (HPC), traditional databases often become bottlenecks because they cannot handle the extreme I/O (Input/Output) demands of thousands of compute nodes working in parallel.

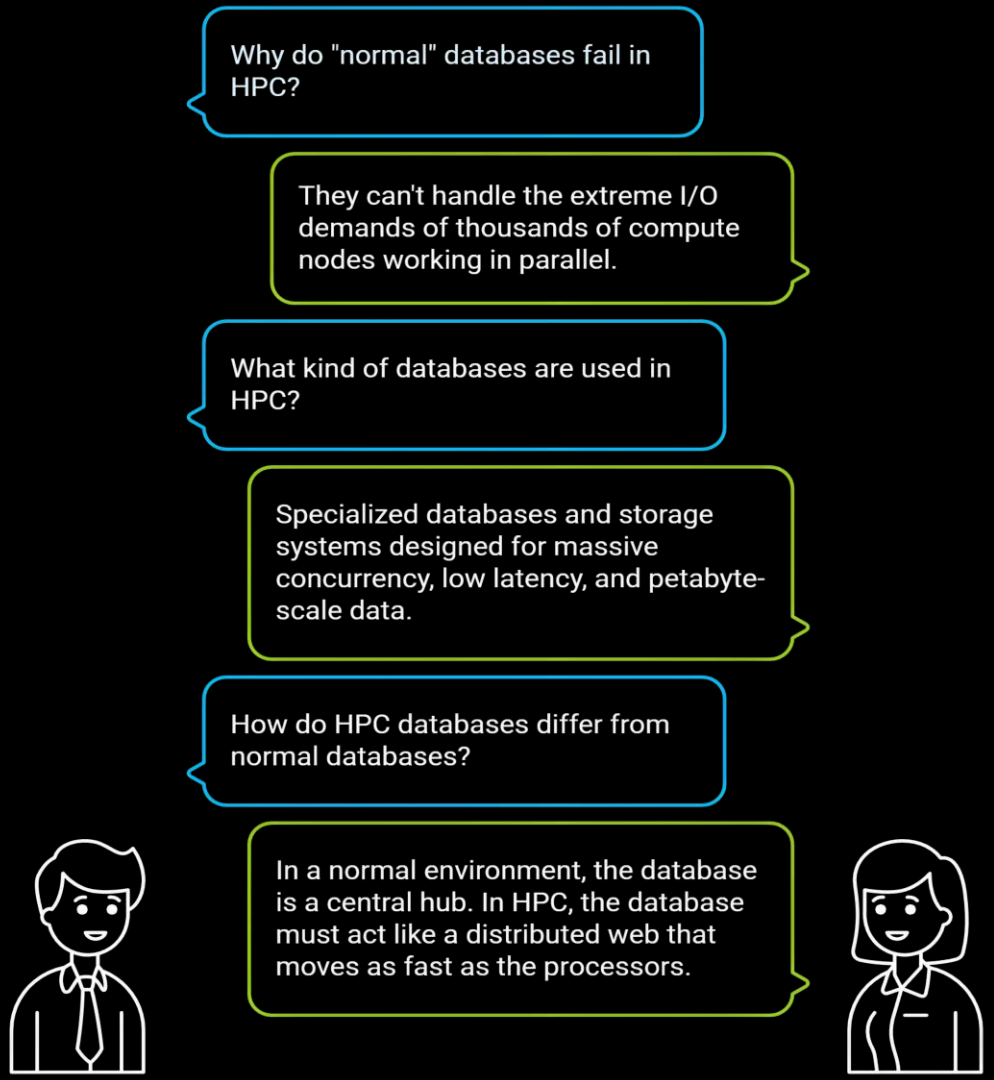
- HPC architectures use specialized databases and storage systems designed for massive concurrency, low latency, and petabyte-scale data.



INTRODUCTION

- Normal DBs: Single point of control → bottleneck under HPC-scale loads
- HPC reality: Processors run at extreme speeds (e.g., petaflops) → DB must match that velocity
- Moves as fast as the processors via parallel, decentralized data flow
- Enables seamless integration with HPC clusters (e.g., Slurm-managed nodes, InfiniBand networks)

Databases in High-Performance Computing (HPC)



Why do "normal" databases fail in HPC?

They can't handle the extreme I/O demands of thousands of compute nodes working in parallel.

What kind of databases are used in HPC?

Specialized databases and storage systems designed for massive concurrency, low latency, and petabyte-scale data.

How do HPC databases differ from normal databases?

In a normal environment, the database is a central hub. In HPC, the database must act like a distributed web that moves as fast as the processors.

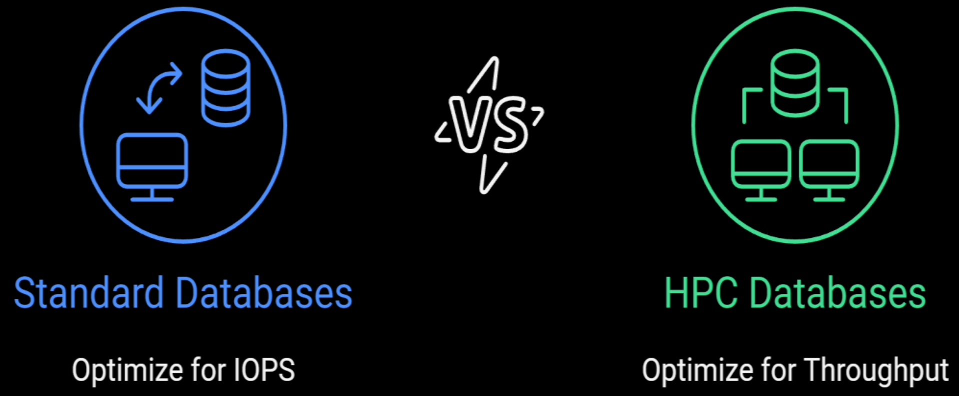
INTRODUCTION

1. The Core Philosophy: "IOPS vs. Throughput"

HPC databases are designed differently because their goals are different from standard business databases:

- **Standard Databases (OLTP):** Focus on **IOPS** (Input/Output Operations Per Second). They are built to handle millions of tiny tasks, like "Update user password" or "Check bank balance."
- **HPC Databases:** Focus on **Throughput**. They are built to move massive "chunks" of data (Gigabytes per second) so that a scientific simulation doesn't have to wait for the disk to finish spinning.

Choose the appropriate database type for specific performance needs.



INTRODUCTION

2. The "Three-Tier" HPC Data Architecture

HPC system doesn't just have one "database." It uses a hierarchy of storage layers to manage data as it moves from the experiment to the final result

Tier 1: **The Burst Buffer (Temporary/Extreme Speed)**

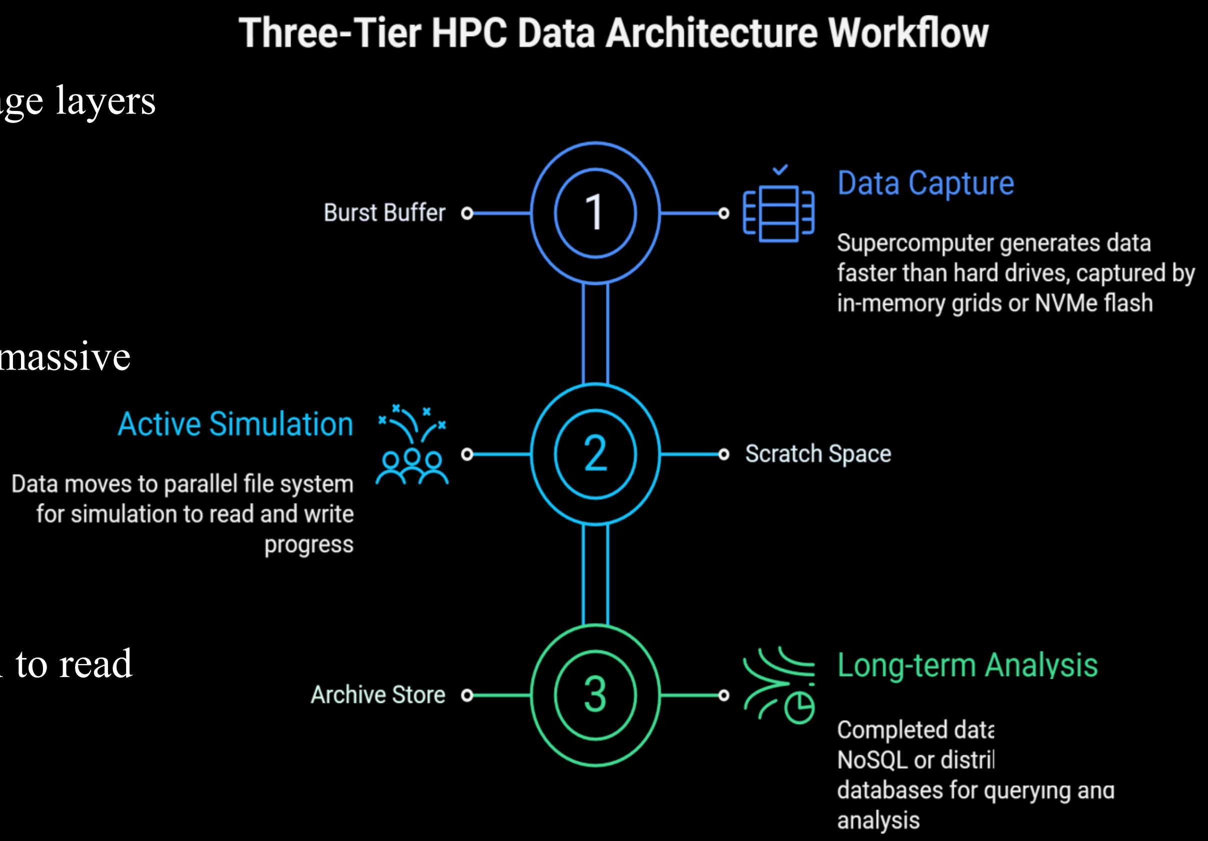
This is the "waiting room" for data. When a supercomputer is running a massive simulation, it generates data faster than any hard drive can record.

Tier 2: **The Scratch Space (Active Work)**

This is where the actual "Parallel File System" (like **Lustre**) lives. It is a temporary workspace that provides the raw bandwidth for the simulation to read and write its progress.

Tier 3: **The Campaign/Archive Store (Long-term Analysis)**

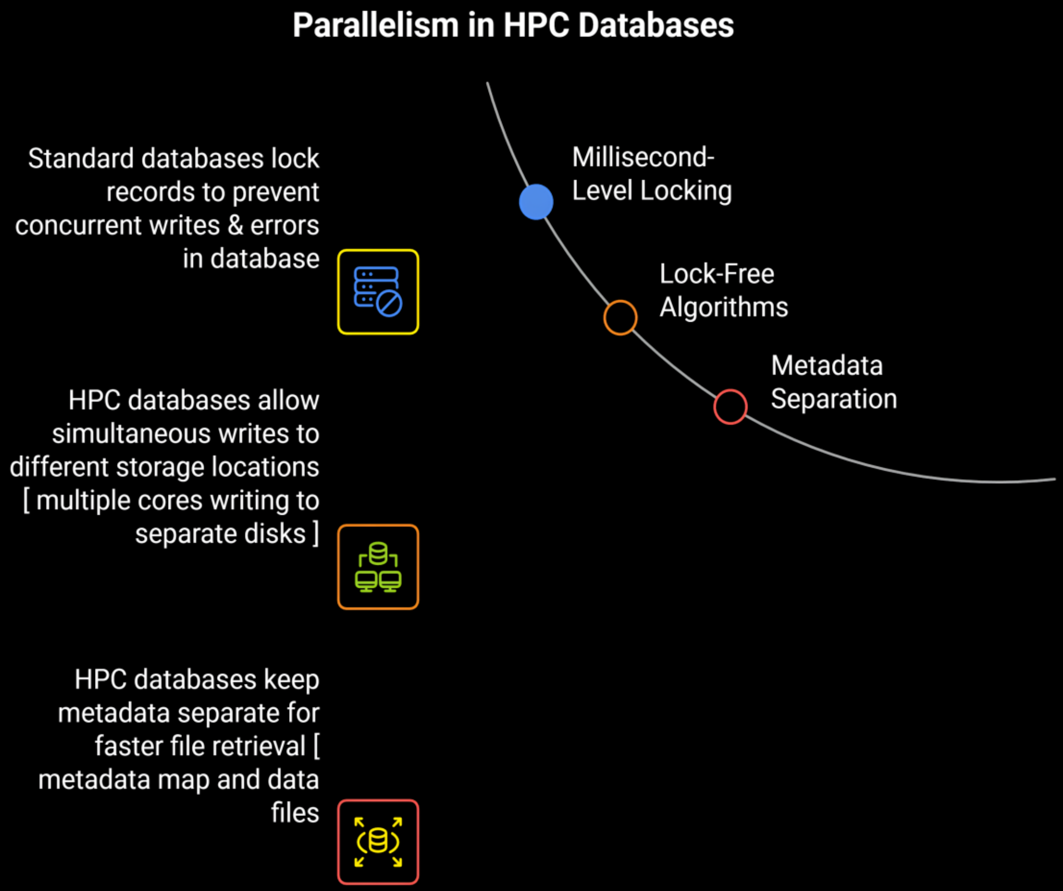
Once the simulation is done, the data is moved here for researchers to "query." This is where **NoSQL** or **Distributed SQL** databases come in to help scientists find specific results within petabytes of data.


Three-Tier HPC Data Architecture Workflow
Burst Buffer
1
Data Capture
Supercomputer generates data faster than hard drives, captured by in-memory grids or NVMe flash
Active Simulation
Data moves to parallel file system for simulation to read and write progress
2
Scratch Space
Archive Store
3
Long-term Analysis
Completed data
NoSQL or distril
databases for querying and analysis

INTRODUCTION

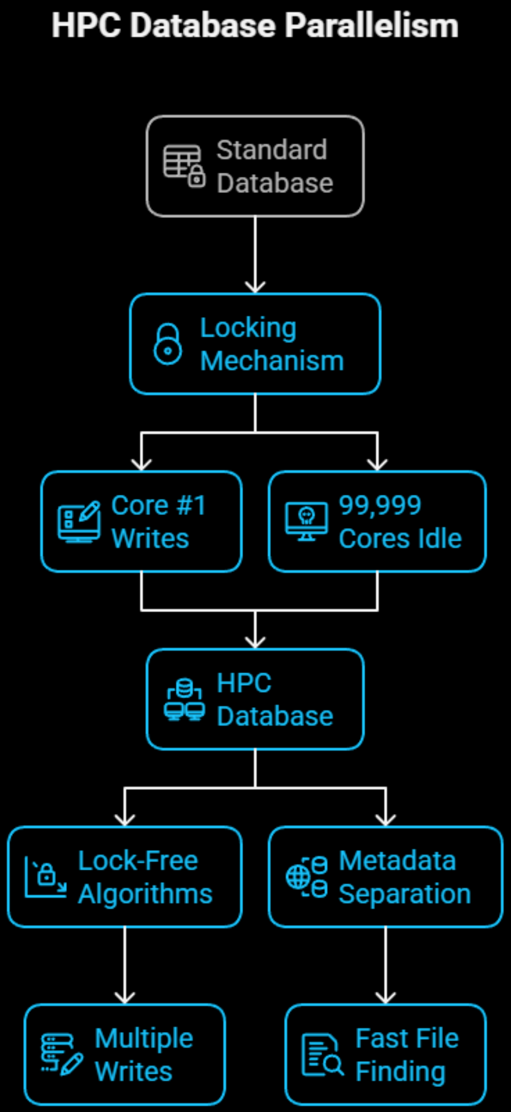
3. Why Parallelism is Mandatory

- In a standard database, if two people try to change the same piece of data at the exact same millisecond, the database "locks" the record to prevent errors.
- In HPC, **locking is the enemy.** If a supercomputer has 100,000 cores and the database "locks" a file to let Core #1 write, the other 99,999 cores sit idle.


Parallelism in HPC Databases
Standard databases lock records to prevent concurrent writes & errors in database
HPC databases allow simultaneous writes to different storage locations [multiple cores writing to separate disks]
HPC databases keep metadata separate for faster file retrieval [metadata map and data files
Millisecond-Level Locking
Lock-Free Algorithms
Metadata Separation

INTRODUCTION

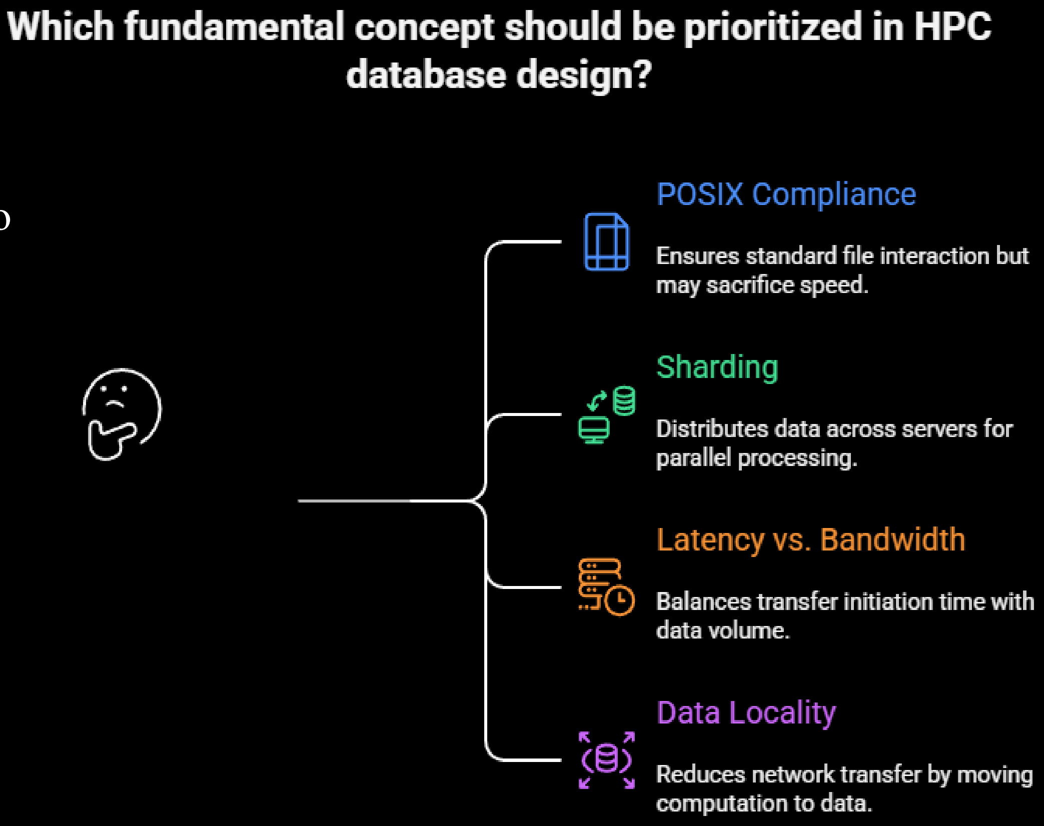
- **Lock-free algorithms:** They allow multiple "writes" to happen in different physical locations of the storage simultaneously.
- **Metadata Separation:** They keep the "map" of the data (where it is) separate from the "content" (the data itself) so that finding a file doesn't slow down reading the file.



INTRODUCTION

4. Fundamental Concepts to Know

1. **POSIX Compliance:** This refers to the "standard" way computers talk to files. Many HPC databases break these rules slightly to get more speed.
2. **Sharding:** Breaking a massive database into 1,000 small pieces so that 1,000 servers can each handle one piece.
3. **Latency vs. Bandwidth:** Latency is how long it takes to *start* a transfer; Bandwidth is how *much* data moves once it starts. HPC databases prioritize Bandwidth.
4. **Data Locality:** The idea of moving the "calculation" to where the "data" is stored, rather than dragging a 10TB file across a network.


Which fundamental concept should be prioritized in HPC database design?
POSIX Compliance
Ensures standard file interaction but may sacrifice speed.
Sharding
Distributes data across servers for parallel processing.
Latency vs. Bandwidth
Balances transfer initiation time with data volume.
Data Locality
Reduces network transfer by moving computation to data.

INTRODUCTION

Feature	Standard Database (MySQL/PostgreSQL)	HPC Database (Lustre/Cassandra)
Primary Goal	Consistency & Accuracy	Throughput & Scalability
Data Size	Gigabytes to Terabytes	Petabytes to Exabytes
User Count	Many users, small requests	Few users, massive requests
Failure Mode	Usually 1-2 central servers	Distributed (survives node failure)
Hardware	Standard SSDs/HDDs	High-speed InfiniBand & NVMe

PRIMARY TYPES OF DATABASES USED IN HPC ENVIRONMENTS:

1. Parallel File Systems (The HPC "Database")

In many HPC workloads (like weather modeling or genomics), the "database" is actually a **Parallel File System**. Unlike standard file systems (NFS), parallel file systems allow hundreds or thousands of nodes to read/write to the same file simultaneously by "striping" data across multiple storage servers.

- **Key Examples:** **Lustre**, **GPFS** (IBM Spectrum Scale), **Bee GFS**.
- **Why they are used:** They provide the raw throughput (GB/s or TB/s) needed for large-scale simulations where traditional SQL databases would crash under the connection load.

2. Distributed NoSQL Databases

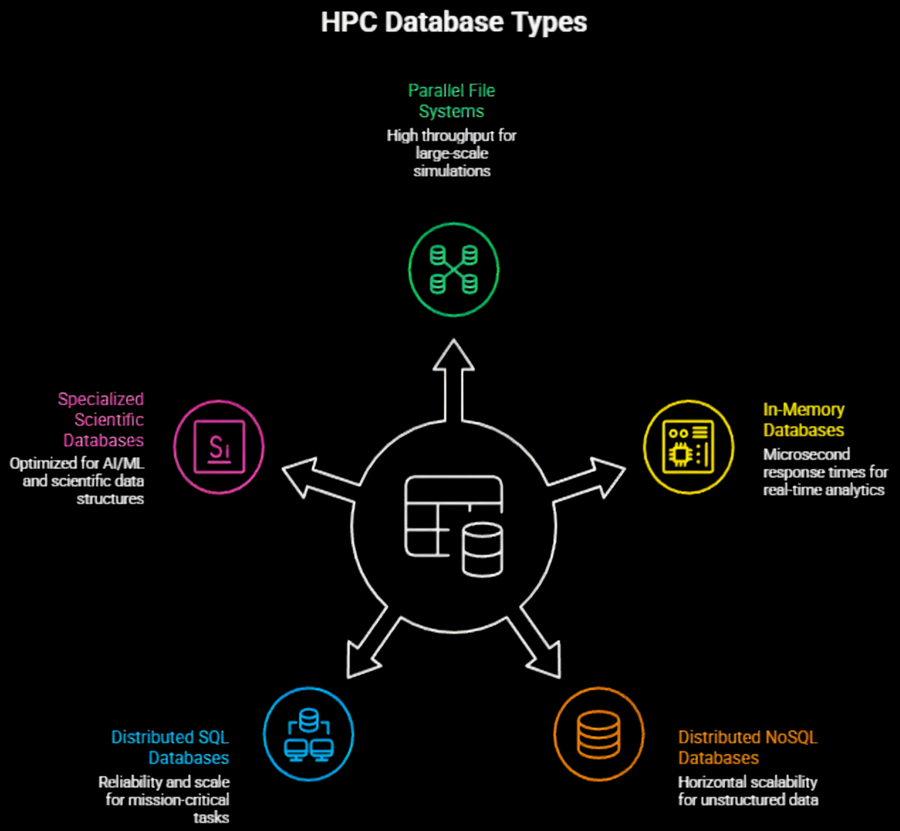
HPC often involves "unstructured" or "semi-structured" data (like sensor logs or satellite imagery).NoSQL databases scale horizontally by adding more servers (nodes) rather than making one server bigger.

- **Key Types & Examples:**
  - **Wide-Column Stores:** **Apache Cassandra**, **HBase** (ideal for time-series and write-heavy scientific data).
  - **Document Stores:** **MongoDB** (used for complex metadata storage).
  - **Key-Value Stores:** **Amazon DynamoDB**, **Riak**.
- **Why they are used:** They offer "high availability" and can handle the massive "ingest" of data coming from scientific instruments without a rigid schema.

3. In-Memory Databases (IMDB)

For real-time analytics or financial modeling, disk access is too slow. In-memory databases store the entire dataset in the system's RAM (Random Access Memory), providing microsecond response times.

- **Key Examples:** **Redis**, **Apache Ignite**, **SAP HANA**.
- **Why they are used:** To eliminate disk I/O bottlenecks. They are often used as a "data grid" to cache frequently accessed data between compute steps.


HPC Database Types
Parallel File Systems
High throughput for large-scale simulations
Specialized Scientific Databases
Optimized for AI/ML and scientific data structures
In-Memory Databases
Microsecond response times for real-time analytics
Distributed SQL Databases
Reliability and scale for mission-critical tasks
Distributed NoSQL Databases
Horizontal scalability for unstructured data

PRIMARY TYPES OF DATABASES USED IN HPC ENVIRONMENTS:

4. Distributed SQL (NewSQL)

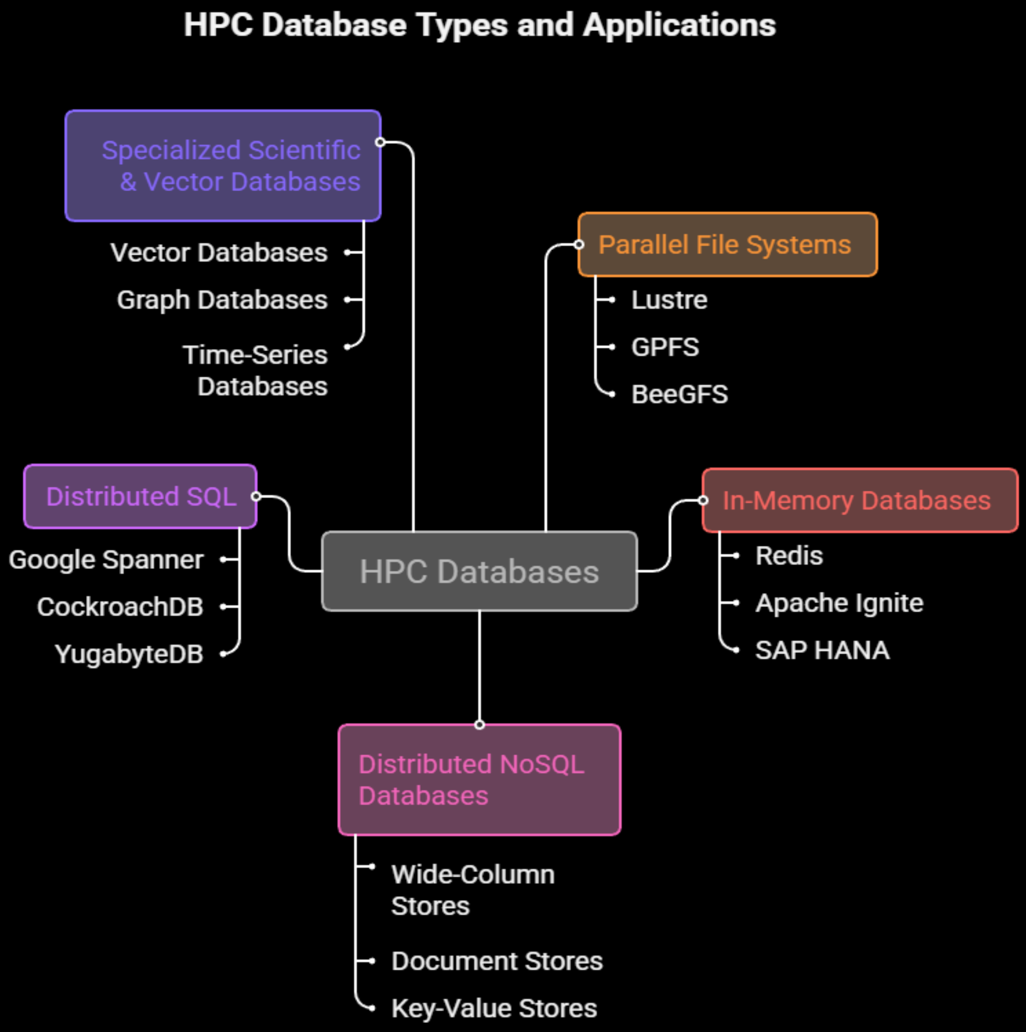
When scientific or financial data requires the absolute reliability of traditional SQL (ACID compliance) but needs the scale of HPC, "Distributed SQL" is used. These databases split data across a cluster but still look like a single relational database to the user.

- **Key Examples:** **Google Spanner**, **CockroachDB**, **YugabyteDB**.
- **Why they are used:** For mission-critical tasks where data consistency is non-negotiable, but the dataset is too large for a single MySQL or Oracle instance.

5. Specialized Scientific & Vector Databases

Modern HPC increasingly integrates AI and Machine Learning. This has led to the use of databases optimized for specific mathematical structures.

- **Vector Databases:** **Milvus**, **Pinecone** (Used to store and search "embeddings" for AI/ML models).
- **Graph Databases:** **Neo4j**, **Amazon Neptune** (Used to map complex relationships, such as molecular interactions or social networks).
- **Time-Series Databases:** **InfluxDB**, **TimescaleDB** (Optimized for tracking changes over time, common in IoT and physics simulations).



PRIMARY TYPES OF DATABASES USED IN HPC ENVIRONMENTS:

Type	Best For	Scaling Strategy	Example
Parallel File System	Large-scale raw data (Simulations)	Parallel I/O over fabric	Lustre
In-Memory	Extreme speed (Real-time)	RAM-based storage	Redis
NoSQL	Unstructured/Big Data	Horizontal (More nodes)	Cassandra
Distributed SQL	Reliable structured data	Sharded across clusters	CockroachDB
Vector/Graph	AI, ML, & Relationships	Optimized Indexing	Milvus / Neo4j

HIGH PERFORMANCE COMPUTING (HPC) DATABASES IN DETAIL:

- We must look at how they solve the **"I/O Bottleneck."** In a standard PC, the CPU waits for the disk. In HPC, thousands of CPUs are waiting for the same data simultaneously.
- In High-Performance Computing (HPC), the I/O bottleneck arises when thousands of CPUs demand data from shared storage simultaneously, far exceeding standard PC scenarios where a single CPU waits on disk access. HPC systems address this through parallel file systems like Lustre or GPFS, which distribute data across many storage servers and use high-speed interconnects to enable concurrent reads/writes from multiple nodes.

1. LUSTRE (PARALLEL FILE SYSTEM):

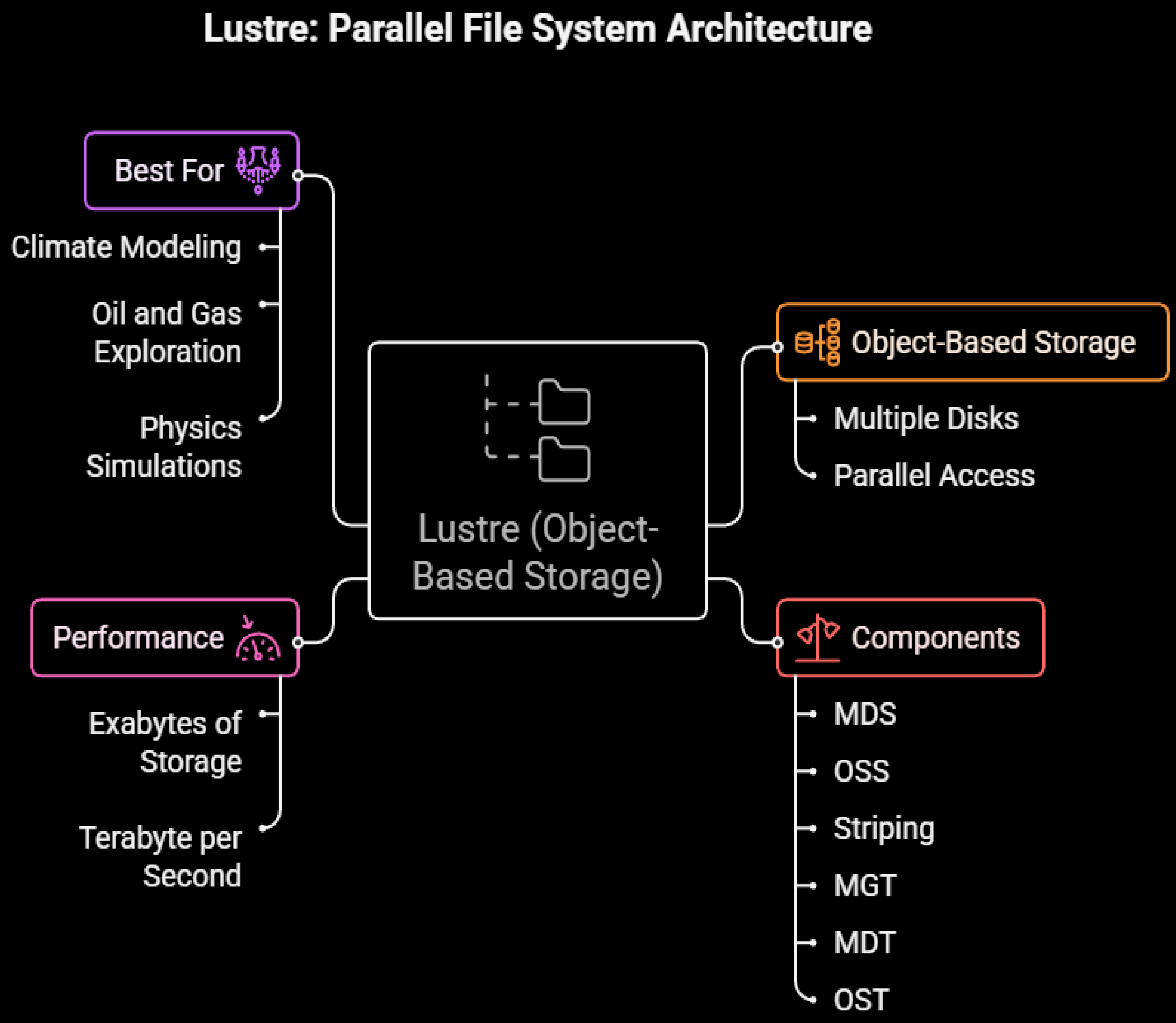
- Lustre is the most common storage system in the world's fastest supercomputers. It isn't a "database" in the SQL sense, but it acts as the primary data engine for scientific simulations.

- Lustre is an object-based file system. It is designed to allow thousands of individual "client" nodes to access the same data pool at the same time without a bottleneck.

- **Object-Based Storage:** Instead of treating a file as a single block of data, Lustre breaks it into **objects**. This allows parts of a single file to exist on multiple physical disks at once.

- **Best For:**

1. Climate modeling,
2. oil and gas exploration,
3. large-scale physics simulations (e.g., CERN).



LUSTRE (PARALLEL FILE SYSTEM):

- **Components:**

  1. **Management Server (MGT):** Stores configuration information for the whole cluster.
  2. **Metadata Target (MDT):** This is the index. It stores the "where" (filenames, directories, permissions). It usually runs on very fast SSDs because it handles millions of tiny "look-up" requests.
  3. **Object Storage Target (OST):** These are the actual "warehouses" where the data chunks live. A single Lustre setup can have thousands of OSTs.
  4. **MDS (Metadata Server):** Acts like a librarian. It knows where files are but doesn't touch the data itself.
  5. **OSS (Object Storage Server):** The muscle. These servers manage the actual data transfer.
  6. **Striping:** When a file is saved, it is chopped into chunks. Stripe 1 goes to OSS A, Stripe 2 to OSS B, and so on. This allows a single file to be written at speeds exceeding **1 Terabyte per second**.

- **Performance:** It can scale to **Exabytes** of storage. It is the "gold standard" for the world’s top 500 supercomputers.

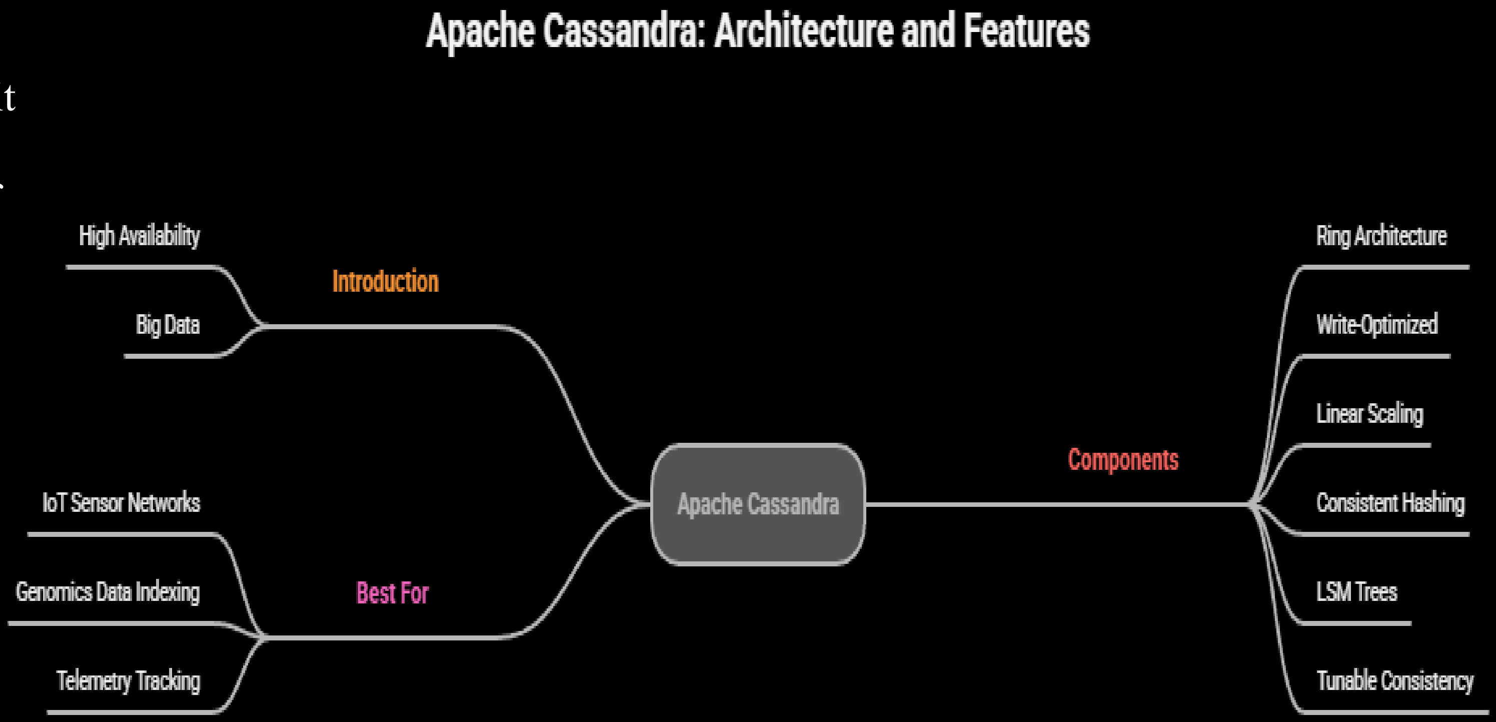
2. APACHE CASSANDRA (DISTRIBUTED NOSQL) :

- Cassandra is built for "High Availability." In HPC, it is used when you need to ingest millions of small data points (like sensor logs) without any chance of the system going down.

- Cassandra was originally open-sourced by Facebook. It is designed for "Big Data" where you need to write data 24/7 without ever stopping for maintenance.

- **Best For:**

1. IoT sensor networks,
2. genomics data indexing, and tracking millions of concurrent telemetry points,
3. Logging telemetry from a million sensors or storing DNA sequences where you need to append data constantly.



APACHE CASSANDRA (DISTRIBUTED NOSQL):

- **Components:**

1. **The "Ring" Architecture:** Every node in a Cassandra cluster is equal. There is no "master" node. If one node fails, the "Ring" simply routes around it.

2. **Write-Optimized:** It uses a **Commit Log** and **Memtables**. Data is written to memory first, making the "Save" operation nearly instantaneous for the compute nodes.

3. **Linear Scaling:** If your experiment grows, you just add more servers. 10 servers handle X data; 100 servers handle $10X$ data—a rare trait in the database world.

4. **Consistent Hashing:** Cassandra uses a "Ring" topology. When data comes in, it is assigned a "Partition Key." A mathematical formula determines exactly which node in the ring owns that key.

5. **LSM Trees (Log-Structured Merge-Trees):** This is why it is so fast. Instead of looking for a spot on the disk to overwrite old data, Cassandra always **appends** new data to the end of a file. It later "compacts" these files in the background.

6. **Tunable Consistency:** You can decide how reliable you want a "write" to be. You can tell Cassandra: "Just save it to one node and tell me it's done" (Fast) or "Wait until 3 nodes have saved it" (Safe)

3. APACHE IGNITE (IN-MEMORY DATA GRID):

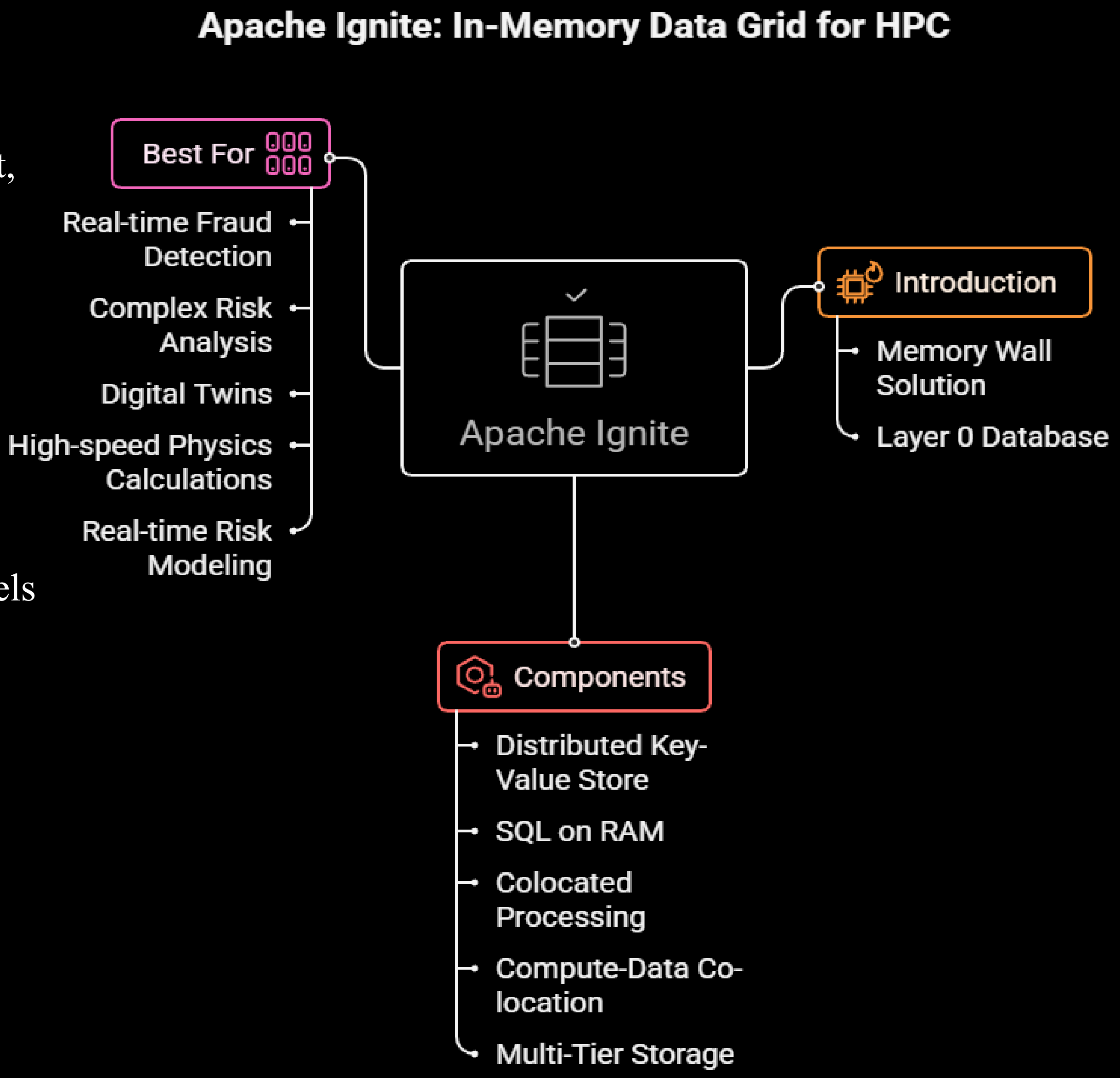
- In HPC, the "Memory Wall" is the biggest hurdle. Apache Ignite solves this by treating the RAM across a whole cluster as one giant, searchable pool.

- Ignite is often used as a "Layer 0" database that sits directly on top of the compute nodes.

- **Best For:**

1. Real-time fraud detection
2. complex risk analysis, and "Digital Twins" (real-time virtual models of physical systems).
3. High-speed physics calculations and real-time risk modeling.

•



APACHE IGNITE (IN-MEMORY DATA GRID):

- **Components:**

  1. **Distributed Key-Value Store:** It spreads data across the RAM of the entire cluster. To the user, it looks like one giant 100TB RAM stick.

  2. **SQL on RAM:** Unlike Redis (which is mostly simple keys), Ignite allows you to run complex SQL queries (JOINs, GROUP BYs) directly on the data stored in memory.

  3. **Collocated Processing:** This is the most important HPC feature. If you want to multiply all values by 2, you don't download the data to your laptop. You send the "Multiply by 2" command to the servers, and they do the work locally on their own RAM.

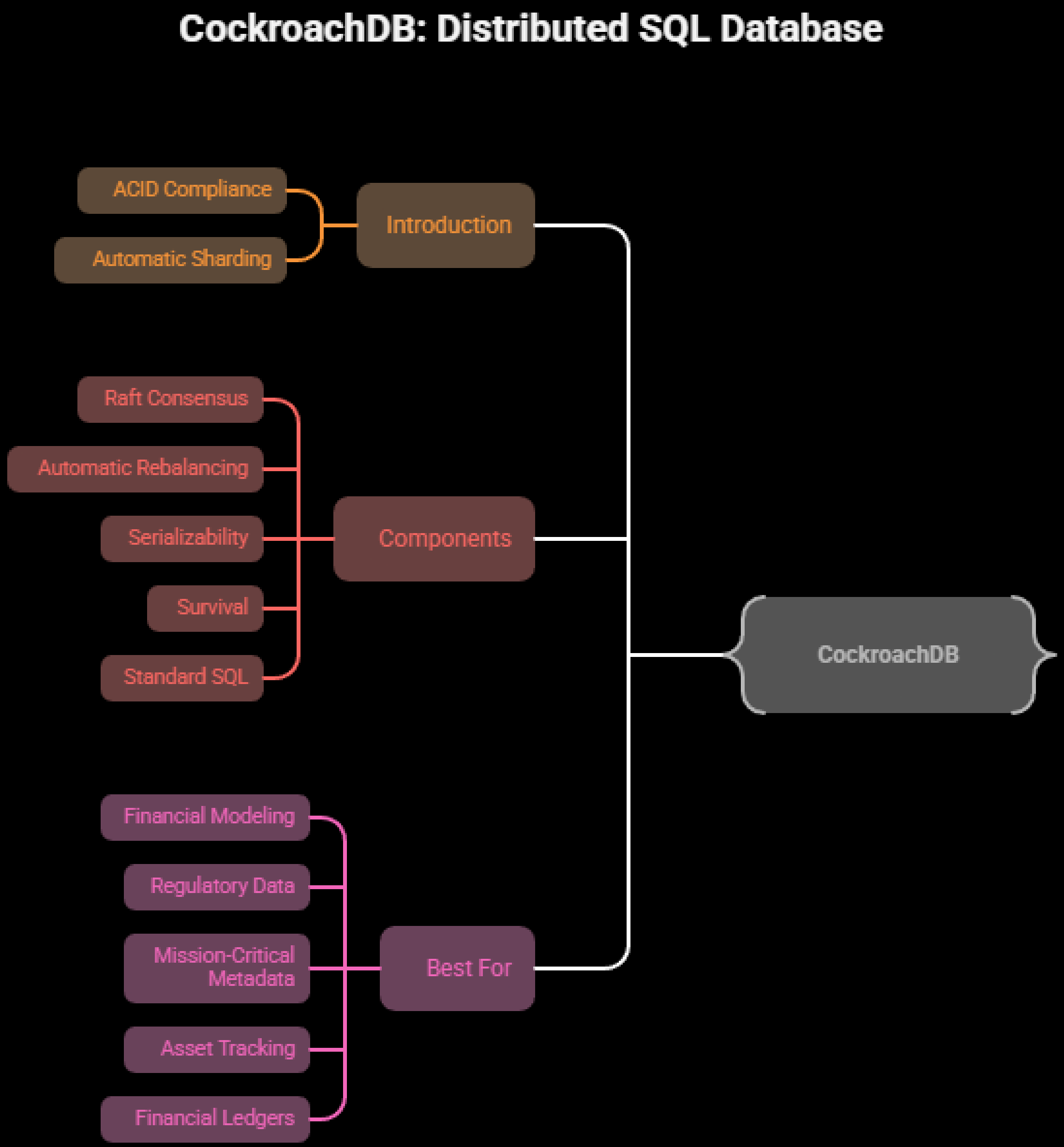
  4. **Compute-Data Co-location:** This is the "magic" of Ignite. Instead of moving 1TB of data to your code, Ignite sends your **code** to the server that already has the data in its RAM. This eliminates network latency.

  5. **Multi-Tier Storage:** It can use RAM for speed but "overflow" to SSDs if the dataset gets too big, ensuring the system never crashes due to "Out of Memory" errors.

4. COCKROACHDB (DISTRIBUTED SQL)

- HPC often requires **ACID compliance** (guaranteeing that data is 100% correct, never partial). Traditional SQL can't scale, but CockroachDB "shards" (splits) SQL tables automatically.
- CockroachDB gives you the power of a standard database (like Postgres) but stretches it across an entire cluster.
- **Best For:**

1. Financial modeling (High-Frequency Trading),
2. regulatory data,
3. mission-critical metadata,
4. Tracking expensive assets, financial ledgers, or any scenario where a "math error" due to data collision would be catastrophic.



COCKROACHDB (DISTRIBUTED SQL)

- **Components:**

  1. **Raft Consensus:** It uses the Raft protocol to ensure that at least a majority of nodes agree on a piece of data before it's "committed."

  2. **The Raft Protocol:** This is the brain. For every piece of data, there are at least three copies. Before a change is finalized, a "Leader" node must get a "Vote" from the other nodes. This ensures that even if a server explodes mid-transaction, the data is never corrupted.

  3. **Automatic Rebalancing:** If you add 10 new servers to your HPC cluster, CockroachDB notices the extra space and automatically moves "ranges" of data to the new servers to balance the load.

  4. **Serializability:** It provides the highest level of isolation. This means that even though 1,000 nodes are working at once, the result is the same as if one single node did all the work perfectly in order.

  5. **Survival:** True to its name, it can survive the "death" of an entire data center. If a rack of servers in an HPC cluster loses power, the database keeps running.

  6. **Standard SQL:** Researchers can use familiar SQL queries while the database handles the complex work of distributing that query across 500 servers.

5. MILVUS (VECTOR DATABASE)

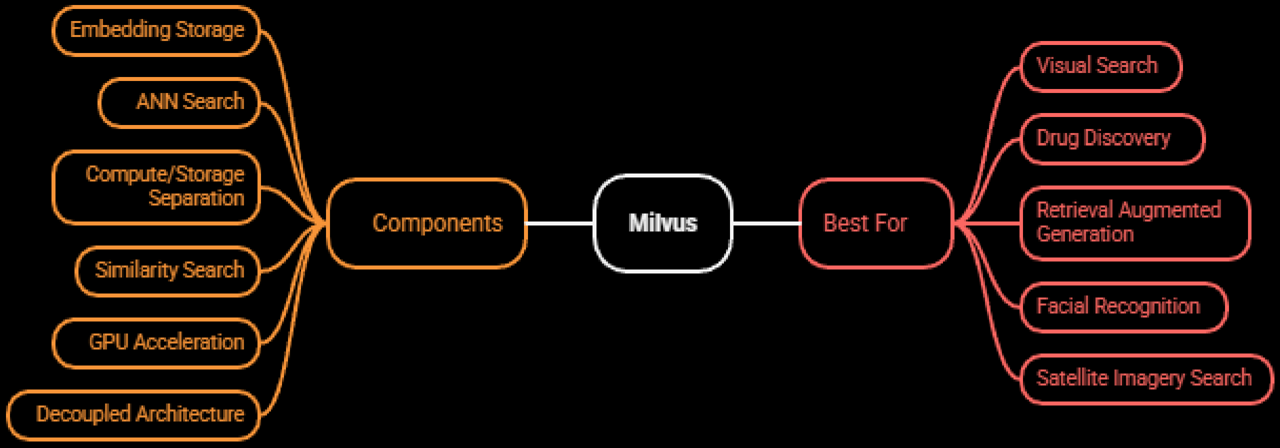
- As HPC moves toward **AI and Machine Learning**, we need to store "Vectors" (mathematical representations of data) rather than just numbers or text.

- Milvus is a specialized database built for the age of AI. It doesn't store words or numbers; it stores **Vectors** (long lists of coordinates like [0.12, -0.5, 0.88...]).

- **Best For:**

1. Visual search,
2. drug discovery (finding molecules similar to a target),
3. Retrieval Augmented Generation (RAG) for AI.
4. facial recognition and searching through massive libraries of satellite imagery.

Milvus: Vector Database for AI and Machine Learning



MILVUS (VECTOR DATABASE)

- **Components:**

1. **Embedding Storage:** It takes unstructured data (like an image of a galaxy or a protein structure) and stores its "mathematical fingerprint."
2. **ANN Search (Approximate Nearest Neighbor):** In HPC, finding an "exact" match in a billion vectors is too slow. Milvus uses algorithms like **HNSW** (Hierarchical Navigable Small World) to find the "nearest" results in milliseconds.
3. **Compute/Storage Separation:** You can scale the "Query Nodes" (which do the heavy math to find neighbors) separately from the "Data Nodes" (which just hold the vectors).
4. **Similarity Search:** Unlike SQL (which looks for exact matches), Milvus finds things that are "similar" using math (like Cosine Similarity).
5. **GPU Acceleration:** Milvus can use NVIDIA GPUs to speed up the search process by 10x-50x compared to standard CPUs.
6. **Decoupled Architecture:** It separates "Proxy" nodes (handling requests) from "Worker" nodes (doing the math), allowing you to scale your search power independently of your storage.

USE AS PER YOUR NEED

If you need...	Use...
Raw I/O Speed (Large Files)	Lustre
No-Downtime Write Scaling	Cassandra
Exact, Reliable SQL Records	CockroachDB
Extreme Low Latency (RAM)	Apache Ignite
AI/ML Pattern Matching	Milvus

FEATURES TABLE

Feature	Lustre	Cassandra	CockroachDB	Apache Ignite	Milvus
Primary Unit	Files / Objects	Rows / Columns	Relational Tables	Key-Value / Objects	Vectors (Tensors)
Storage Medium	Parallel Disk/SSD	Disk (Append-only)	Distributed Disk	RAM (Primary)	Disk/RAM + GPU
Scalability	Massive (Exabytes)	Linear (Unlimited)	High (Multi-region)	High (RAM limited)	High (AI-focused)
Consistency	POSIX	Eventual/Tunable	Strong (ACID)	Strong/Atomic	Eventual