Big Picture: What BDA is solving

- **Why Big Data?** We moved from single servers to clusters because storage grew faster than disk speed; parallelism + replication beat single-box scale-up. Hadoop processes data where it sits (data locality) to avoid network bottlenecks.

- **Project flow (modern BDA):** clarify business goals → benefit/ROI analysis → review similar wins → pick cloud tech → implement/evaluate; the cloud killed most old infra barriers.

- **Core ecosystem (snapshot):** Storage (HDFS/HBase), Processing (MapReduce/Spark/Oozie), SQL (Hive/Impala/Trino), Visualization (Tableau/Power BI).

The "Vs" (as emphasized in your doc)

- **Three Vs (doc framing): Volume** (size), **Velocity** (ingest speed), **Variety** (formats). Expect questions that stick to these 3.

Architectures you must name-drop

- **Lambda, Kappa, HTAP** appear as key architectures to know (you'll likely be asked to list/contrast at a high level): batch+speed layer (Lambda), unified stream (Kappa), and hybrid transactional/analytical (HTAP). Keep concise definitions handy.

Scale-Up vs Scale-Out (and why scale-out wins)

- **Scale-Up (vertical):** bigger box (CPU/RAM). **Scale-Out (horizontal):** more nodes. For BDA, scale-out wins for linear scalability, fault tolerance, cost, parallelism, and elasticity. Spark's in-memory compute cuts latency.

NoSQL in one glance

- **What:** “Not Only SQL,” schema-flexible, horiz. scalable; trades strict ACID for CAP constraints.
- **Types:** Document (MongoDB), KV (Redis), Columnar (Cassandra), Graph (Neo4J). Mention **NewSQL** for ACID + scale.

Hadoop core (why it mattered)

- **HDFS replication vs RAID:** HDFS replicates **blocks** across nodes (factor ~3) vs RAID replicating disks in one server—gives cluster-level durability + parallel reads. Default HDFS block 128 MB.
- **Strengths & limits:** scalable, reliable, cost-effective; batch-oriented (minutes-level latency).
- **Evolution:** YARN (resource mgr), HBase (KV), Hive/Impala/Trino (SQL), Spark (in-memory), Solr (search).

MapReduce model & runtime behavior (exam magnets)

- **Model:** Map (emit key→value) → Shuffle (group by key) → Reduce (aggregate by key). Built-in fault tolerance and parallelism.
- **Input split:** logical chunk processed by one map task; usually aligned with HDFS block size. **Why smaller splits?** More parallelism but more overhead; sweet spot ≈ block size.
- **Partitioner:** decides which reducer gets which key (default hash partitioner).
- **Shuffle:** mapper → reducer data transfer; a major perf lever.
- **Combiner (mini-reduce):** optional local aggregation; must be idempotent; may run 0,1, or many times. Reduces network traffic.

- **Speculative execution:** duplicates stragglers to avoid tail latency.
- **Data locality:** schedule tasks where data lives to minimize network reads; remote reads happen if locality impossible.

Distributed systems concepts (timing & coordination)

- **Clock sync:** Cristian's algorithm sets client ≈ server time + half RTT; **NTP** gives internet-scale, ms-level sync.
- **Logical clocks:** Lamport (partial order) vs **Vector clocks** (detect causality **and** concurrency).
- **Mutual exclusion & leader election:** purpose is single-access to shared resource; **Bully algorithm** elects highest-ID leader.
- **Barrier synchronization:** e.g., Spark stage boundaries.
- **Reduce pattern = convergecast** (many→one aggregation).

HPC vs Hadoop (why Hadoop for data)

- **HPC (MPI + SAN):** compute-heavy, network bottlenecks for huge data.
- **Hadoop:** moves compute to data; optimized for **data-intensive** workloads and topology-aware scheduling.

Storage & performance (block sizes, devices, Optane)

- **Performance ladder:** HDD → SATA SSD → NVMe SSD → DRAM → **Persistent Memory (Optane)**; know their **throughput/IOPS/latency** ballparks (e.g., HDD ~80–160 MB/s, ~200 IOPS; NVMe 3–7 GB/s, ~1M IOPS; DRAM <100 ns). Perfect for "compare/choose best layer" questions.

- **Use the right tier:** HDD = cold lake/archival; SSD/NVMe = hot analytics; Optane = hybrid cache/persistent tier.
- **Why parallel disks matter:** single-disk reads scale badly with capacity growth; stripes of many disks + replication shrink wall-clock time and guard against failures.
- **HDFS vs RAID recap (again):** cluster vs single-server durability; block vs disk units. Expect a short table/contrast.

Linux essentials you're expected to know

- **Permissions (rwx → 421):** Examples: 777 (all), 755 (owner full, others r+x), 644 (owner rw, others r). **SUID/SGID/Sticky bit** (passwd, group inheritance, /tmp protection).
- **Shells:** sh, **bash**, csh, ksh, zsh; shell roles include exec, redirection, pipes, variables, scripting.
- **File systems:** ext2/3/4, XFS, Btrfs; key ideas: **inodes** and **mounting**.
- **Kernel vs user mode; devices as files under /dev** — keep the philosophy line handy for quick marks.

Bash scripting (syntax you'll write under pressure)

- **Shebang + chmod:** `#!/bin/bash` + `chmod +x` to run directly.
- **Variables/params, if/else, for/while, functions, read, I/O redirection (`>`, `>>`, `<`)** — practice micro-snippets; you'll likely be asked to write or read one.
- **Mini menu loop pattern** (great to memorize).
- **Alpine inside Docker quickstart** (apk update/add bash/nano; make & run script) — perfect for procedural marks.

Containers & Docker (virtualization emphasis)

- **Virtualization vs containerization:** VMs emulate hardware with a guest OS (Type-1 bare-metal vs Type-2 hosted), containers share host kernel → lighter/faster. Snapshots + storage virtualization basics also appear.
- **Container lineage:** chroot (1979) → FreeBSD Jails (2000) → Solaris Zones (2004) → **Namespaces** & **cgroups** (Linux) → **LXC (2008)** → **Docker (2013)** → **Kubernetes (2015)**. Keep this order.
- **Namespaces (PID/NET/MNT/UTS/IPC/USER) & cgroups:** what they isolate and why they matter (resource control).
- **Docker architecture:** CLI → dockerd → containerd → runc → Linux kernel isolation.
- **Dockerfile directives to remember:** `FROM`, `RUN`, `COPY/ADD`, `WORKDIR`, `CMD/ENTRYPOINT`, `EXPOSE`, `VOLUME` (+ the "image vs container" one-liner).
- **Volumes (you were told to review CLI meanings, not to write Dockerfiles):** purpose, persistence, sharing, backup/recovery.
- **Orchestration:** what K8s does (pods, HA, scaling), plus rkt/containerd mentions.

Data-locality & scheduling details (often short answers)

- **Why map output is local (not HDFS):** it's temporary; avoid replication overhead; speeds reduce phase.
- **Load balancing:** distribute tasks evenly; dynamic rebalancing is good form.

Classic "compare/contrast" bullets (write these fast in exams)

- **HDFS vs RAID:** cluster-wide block replication vs single-server disk redundancy; networked parallel reads vs local redundancy.
- **RDBMS vs MapReduce:** B-trees for small updates vs sort/merge efficiency for bulk rebuilds; MR beats RDBMS for **large transforms**, not OLTP.
- **MPI/HPC vs Hadoop:** explicit message passing + SAN vs KV abstraction + HDFS; manual vs automatic fault handling.
- **Container vs VM:** shared kernel (light) vs full guest OS (heavy).
- **HDD vs SSD/NVMe vs Optane/DRAM:** capacity & cost vs latency & IOPS; match tier to workload.

Likely scripting prompts (what to memorize)

- **If/elif/else & `fi`** closure; arithmetic `$(( ))`; `for`/`while`; functions; `read -p`; redirection `>`, `>>`, `<`. Keep one-liners etched.
- **Minimal Alpine flow inside Docker:** `docker run -it alpine` → `apk update` → `apk add bash nano` → write script → `chmod +x` → run.

Short factual drops (one-liners you can fire)

- "Devices are files under **/dev**" (Linux philosophy).
- "Default HDFS block often **128 MB**" (align splits to blocks).
- "Combiner must be **idempotent** (can run any number of times)."
- "Reduce = **convergecast** pattern."

Your rapid-fire revision flow (how to cram in 60–90 mins)

1. **Skim architectures & ecosystem (10–15 min):** Lambda/Kappa/HTAP, Hadoop pieces, Spark in-memory rationale.

2. **MapReduce execution path (15–20 min):** splits → map → partition → shuffle → combine → reduce → speculative exec → data locality trade-offs.

3. **Linux + Bash (15–20 min):** perms/SUID/SGID/sticky, shebang + chmod, loops & redirection mini-snips, Alpine-inside-Docker steps.

4. **Containers (15–20 min):** lineage → namespaces/cgroups → Dockerfile verbs → volumes → what K8s gives you.

5. **Storage tiers & IOPS (10–15 min):** ladder table + use-cases; HDFS vs RAID table.

6. **Distributed systems timing (10 min):** Cristian, NTP, Lamport vs Vector, Bully, barriers.