System Design Document: Mini1 - Memory & Concurrent Processing

Project Objective: To investigate memory utilization and concurrent processing by building a C/C++ library that ingests, parses, and searches a massive dataset (>12 GB, >2 million records). The project relies on analyzing strong/weak scaling, memory performance, and benchmarking across three distinct architectural phases.

---

1. System Architecture & Tooling

Language & Compilation:

- **Core Language:** C/C++ using GCC (v13 or newer) or Clang (v16 or newer).
- **Build System:** CMake is **required** to compile the library and testing harness.
- **Prohibited Tech:** No third-party databases or external services are allowed. Basic CSV parsing tools are permitted. IDE VMs should not be used for running/testing.

Data Pipeline:

- **Source:** NYC OpenData (e.g., TLC Taxi Data, combined to exceed 12 GB and 2M records).
- **Data Types:** Data fields must be strictly represented as primitive types (e.g., integers, floats).
- **Output / Analysis:** Python 3 will be used to generate graphs from the benchmark data.

---

2. Phase 1: Serial Object-Oriented Design (Baseline)

Goal: Build the baseline architecture focusing on object-oriented abstraction and serial processing.

- **Design Pattern (Array-of-Objects):**
  - Design a class (e.g., `DataRecord`) where each instantiated object represents a single row from the CSV using primitive types.
  - Design a container/manager class (e.g., `DatasetManager`) that holds a collection (like `std::vector`) of these objects.
- **Required APIs:** The code must be designed as a long-term library providing APIs specifically for **data reading** and **basic range searching**.
- **Execution Restriction: Do not use threads in this phase; it must be strictly serial processing**.

- **Benchmarking Harness:** Create a test harness to execute your reading and searching APIs **10+ times** to calculate an average performance baseline. You must document both successes and failures.

---

3. Phase 2: Parallelization

Goal: Introduce concurrency to improve the baseline performance.

- **Implementation:** Apply threading or OpenMP to the Phase 1 APIs (data ingestion and range searching).
- **Testing:** Use the benchmarking harness developed in Phase 1 to record the parallelized results and compare them directly against your serial baseline.

---

4. Phase 3: Vectorized Organization (Optimization)

Goal: Achieve maximum performance by altering the memory layout of your classes.

- **Design Pattern (Object-of-Arrays):** Rewrite your data classes from an "Array of Objects" to a **vectorized organization (Object-of-Arrays)**.
  - *Implementation Strategy:* Instead of storing objects that each contain a float and an int, create a single class that contains massive, contiguous arrays for each specific primitive column (e.g., one array for all floats, one array for all ints).
- **Query-Driven Design:** The assignment hints that your specific search queries should shape this design to maximize cache hits and memory efficiency.
- **Benchmarking:** Test how this structural memory modification affects performance compared to Phase 1 and Phase 2. Performance is the highest priority in this phase.

---

5. Deliverables & Artifacts Structure

When the design and implementation are complete, your team must package the following into a single `tar.gz` archive (e.g., `teamname.tar.gz`) for submission:

1. **The Code:** Must include your C/C++ source, CMake configuration, and benchmark harness. **Do not include the 12 GB test data**.
2. **The Report:** A paragraph-formatted document detailing your approach, how the team achieved the solution, individual contributions, and citations. It must include tabular data, generated graphs, observations backing up your conclusions, and documentation of failed attempts.

3. **The Presentation: Exactly one slide** (like a poster) focusing on a single, unique, important discovery your team made. It cannot be a project summary, class diagram, or tutorial.