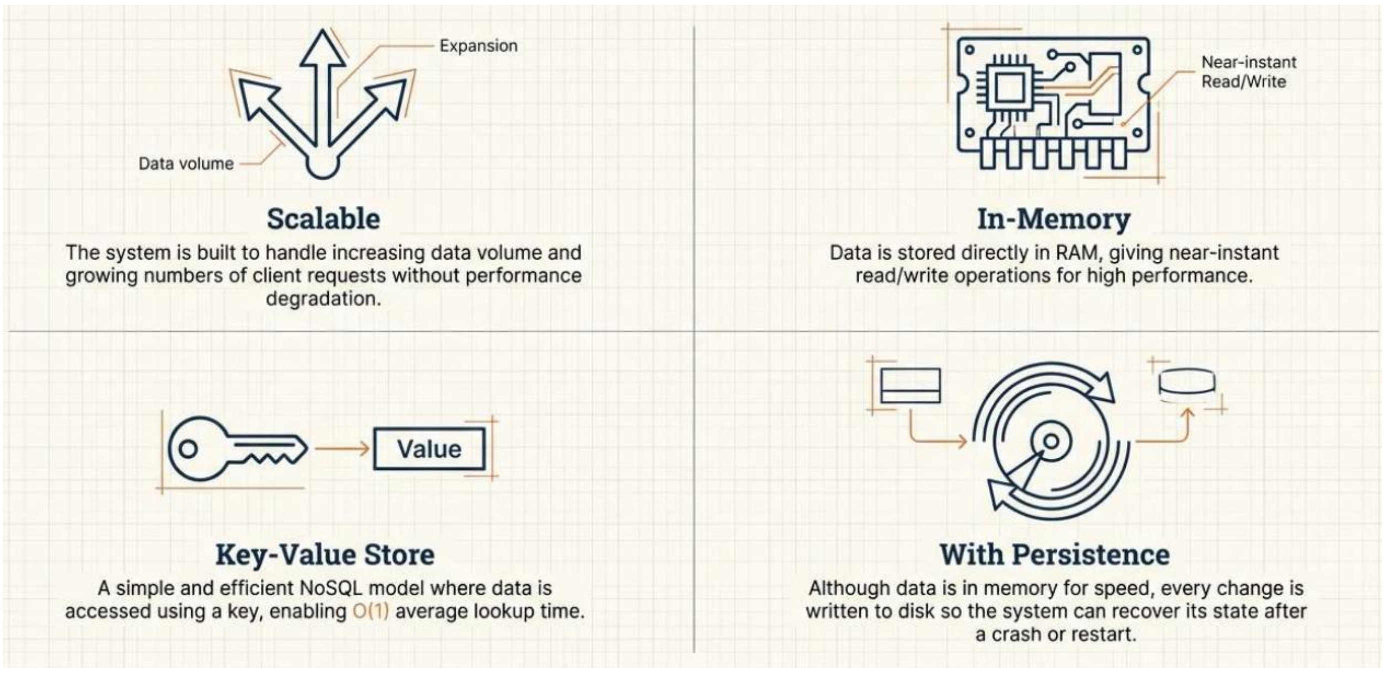
# PyKV: A Scalable In-Memory Key-Value Store with Persistence

- The title *“Scalable In-Memory Key-Value Store with Persistence”* reflects four key characteristics of the project.



**Scalable**
The system is built to handle increasing data volume and growing numbers of client requests without performance degradation.

**In-Memory**
Data is stored directly in RAM, giving near-instant read/write operations for high performance.

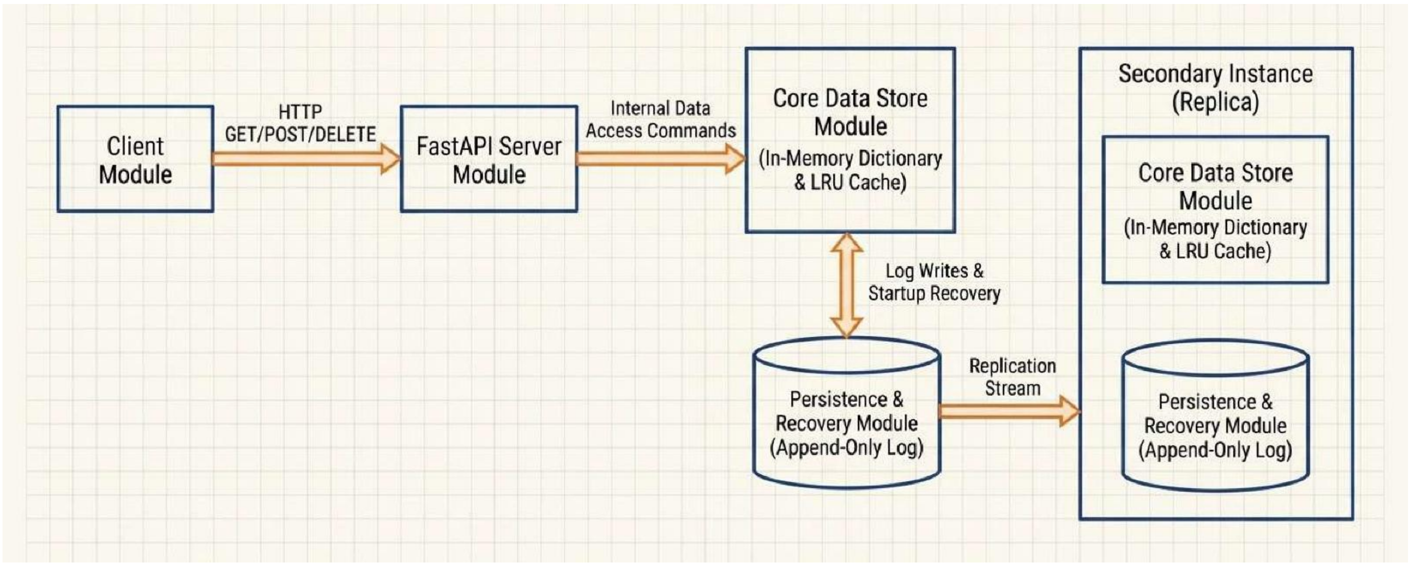
**Key-Value Store**
A simple and efficient NoSQL model where data is accessed using a key, enabling O(1) average lookup time.

**With Persistence**
Although data is in memory for speed, every change is written to disk so the system can recover its state after a crash or restart.

- Together, these elements describe a system engineered for both performance and reliability.

---

**SYSTEM ARCHITECTURE :**


Client Module
HTTP GET/POST/DELETE
FastAPI Server Module
Internal Data Access Commands
Core Data Store Module (In-Memory Dictionary & LRU Cache)
Log Writes & Startup Recovery
Persistence & Recovery Module (Append-Only Log)
Replication Stream
Secondary Instance (Replica)
Core Data Store Module (In-Memory Dictionary & LRU Cache)
Persistence & Recovery Module (Append-Only Log)

---

## Library Analogy: Step-by-Step Operation

To help understand how PyKV works, we can visualize it as a **library system**, where clients interact with a librarian and shelves.
This analogy simplifies the flow of GET, SET, DELETE and UPDATE operations.

**1. GET Operation (Borrow a Book)**

1. Patron (Client) says: “Give me book **X**.”
2. Librarian (FastAPI Server) receives the request.
3. Librarian checks if book **X** exists and asks the shelves (Core Data Store).
4. Shelves check dictionary:
    - Book exists → move to most recently used (LRU) → return book.
    - Book does not exist → return “Book not found.”
5. Librarian delivers book **X** to the patron.

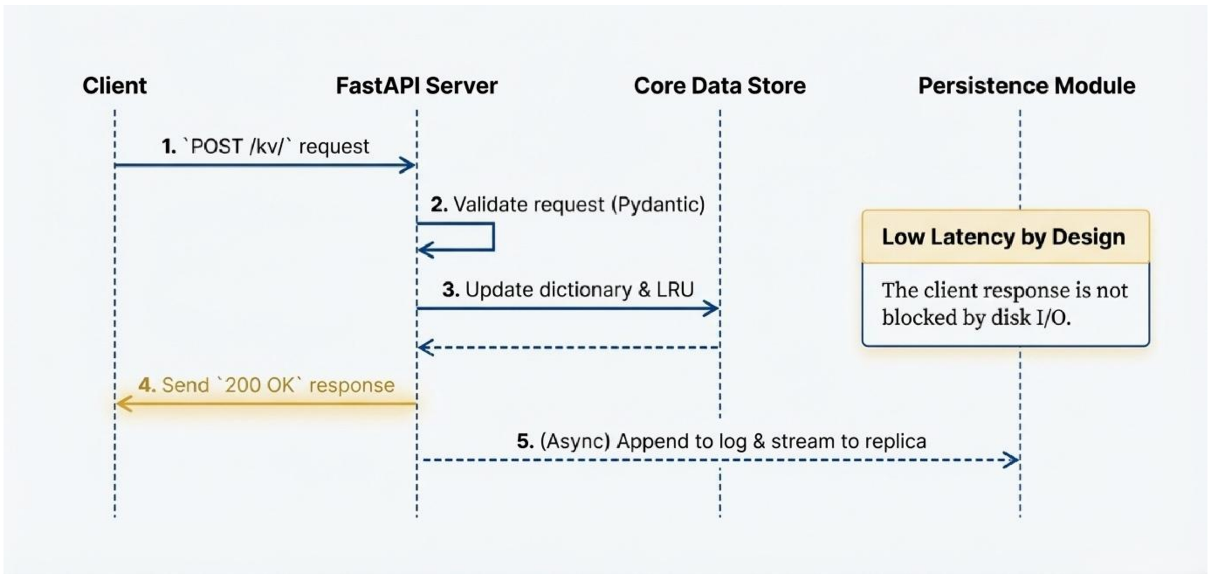
**2. SET / PUT Operation (Add a New Book)**

1. Patron says: “Add book **X** with content **Y**.”
2. Librarian receives request and validates it.
3. Shelves insert or update book **X**.
4. Move book **X** to most recently used (LRU).
5. Logbook (Persistence Module) records this action.
6. Branch library (Replication Module) receives log asynchronously.
7. Librarian confirms to patron: “Book added/updated successfully.”

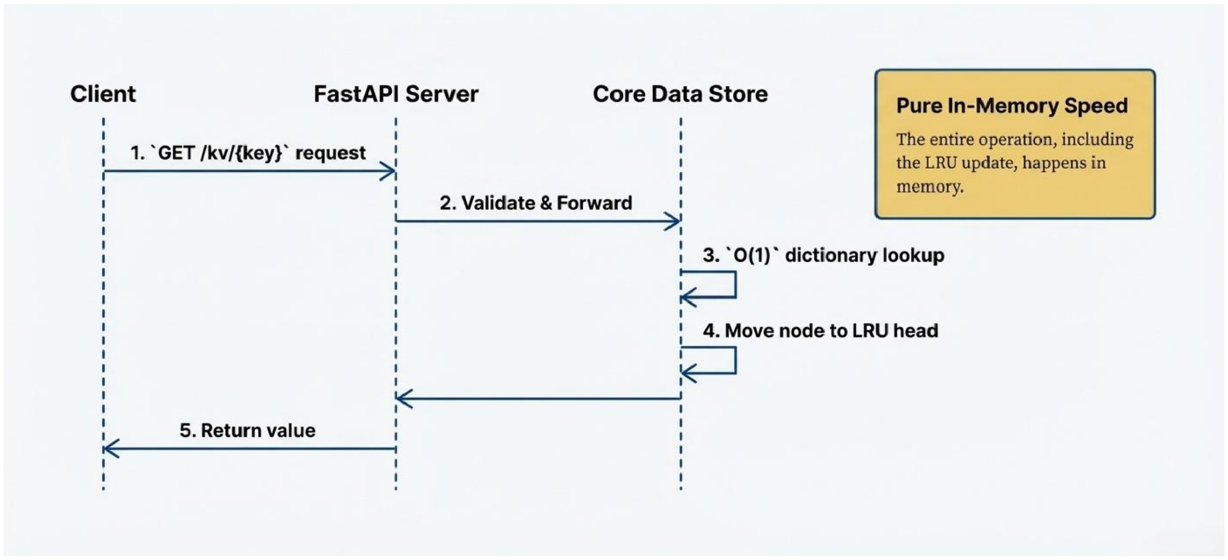
**3. DELETE Operation (Remove a Book)**

1. Patron says: “Remove book **X**.”
2. Librarian validates request.
3. Shelves remove book **X** from dictionary and LRU.
4. Logbook records DELETE operation.
5. Branch library receives replication log.
6. Librarian confirms to patron: “Book deleted successfully.”

## Anatomy of a Write Operation: ‘SET/PUT’ Request


Client
FastAPI Server
Core Data Store
Persistence Module
1. `POST /kv/` request
2. Validate request (Pydantic)
3. Update dictionary & LRU
4. Send `200 OK` response
5. (Async) Append to log & stream to replica
Low Latency by Design
The client response is not blocked by disk I/O.

## Anatomy of a Read Operation: ‘GET’ Request


Client
FastAPI Server
Core Data Store
Pure In-Memory Speed
The entire operation, including the LRU update, happens in memory.
1. `GET /kv/{key}` request
2. Validate & Forward
3. `O(1)` dictionary lookup
4. Move node to LRU head
5. Return value

## CORE DATA STORE MODULE

- The **central component of PyKV** – stores all key-value pairs in memory.

**In-Memory**

- The primary storage → **Python dictionary**

**LRU (Least Recently Used) Caching Policy**

- When **MAX_CAPACITY** , the **least recently accessed key-value pair** is removed
- LRU is implemented using two structures:
    1. **Dictionary** → stores key and node
    2. **Doubly Linked List** → tracks usage order;

**Logic:**

- **GET** → Accessed key is moved to **head** (MRU)
- **SET** →
    - If new key → insert at head.
    - If capacity exceeded → remove **tail** node (LRU)
- **UPDATE** → Existing key's value is overwritten and key is moved to **head** (MRU).
- **DELETE** → Remove key

---

## FASTAPI SERVER MODULE & API SPECIFICATION

- Handles incoming requests → GET, SET, UPDATE, DELETE
- validates input → Pydantic Modules
- and ensures that multiple clients can interact with the store simultaneously without blocking.
- **Asyncio**

**API Endpoints**

| **HTTP Method** | **Endpoint** | **Description** | **Request Body / Parameters** |
|---|---|---|---|
| POST | /kv/ | Add a new key-value pair | JSON: { "key": "<key>", "value": "<value>" } |
| GET | /kv/{key} | Retrieve the value for a given key | Path: key |
| PUT | /kv/{key} | Update the value for an existing key | JSON: { "value": "<new_value>" } |
| DELETE | /kv/{key} | Delete a key-value pair | Path: key |
| GET | /kv/ | List all keys with optional filters | Query: prefix (optional) |

## PERSISTENCE & RECOVERY MODULE

- ensures **data durability** and **crash recovery** for PyKV.
- Memory is volatile, meaning data will be lost if the server crashes or shuts down.
- This module guarantees that all changes to the key-value store are **safely recorded on disk** and can be reconstructed during startup.

**Persistence Mechanism: Append-Only Log**

- All **state-changing operations** (SET, DELETE, UPDATE) are written sequentially to an **append-only log file**.
- **Log Entry Format:** JSON or lightweight binary record

```
{ operation: "SET/DELETE/UPDATE",
  key: "<key>",
  value: "<value>",
  timestamp: <time>
}
```

- Maintains **ordered history** of operations for exact reconstruction.

---

## REPLICATION & HIGH AVAILABILITY

- **primary-secondary replication model**, where the secondary node maintains a copy of the primary's data and can take over if the primary fails.

**Replication Logic**

- **Asynchronous Streaming**
- **Eventual Consistency**.
- **Failover**

---

## CLIENT MODULE & COMMUNICATION PROTOCOL (CLI + GUI)

- **both a Command-Line Interface (CLI)** and a **GUI using Streamlit** for a more user-friendly experience.

**Command-Line Interface (CLI)**

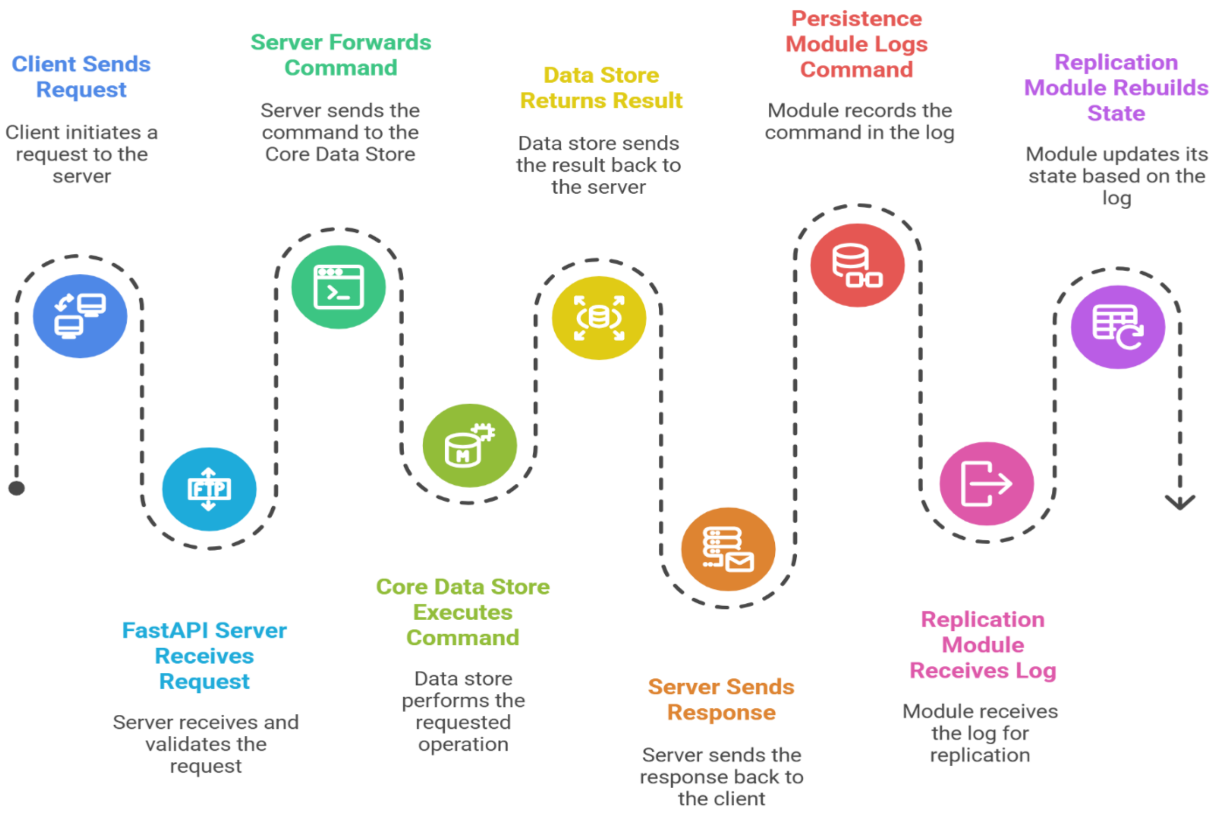
- **Operations Supported:**
  - SET <key> <value> → Add or update a key-value pair.
  - GET <key> → Retrieve a value.
  - DEL <key> → Delete a key.
- Sends **HTTP requests** to the FastAPI server and receives **real-time responses**.

**Streamlit GUI**

- Provide a **visual web-based interface** to perform key-value operations without CLI commands.
- Display server responses and optionally performance metrics.
- **Input Fields**
- **Buttons**
- **Output Area**
- **Dashboard**

---

## FINAL PROJECT OUTPUT & USER EXPERIENCE


Client Sends Request
Client initiates a request to the server
FastAPI Server Receives Request
Server receives and validates the request
Server Forwards Command
Server sends the command to the Core Data Store
Core Data Store Executes Command
Data store performs the requested operation
Data Store Returns Result
Data store sends the result back to the server
Server Sends Response
Server sends the response back to the client
Persistence Module Logs Command
Module records the command in the log
Replication Module Receives Log
Module receives the log for replication
Replication Module Rebuilds State
Module updates its state based on the log