Elevator Simulation - High Level Design

1. Problem Statement

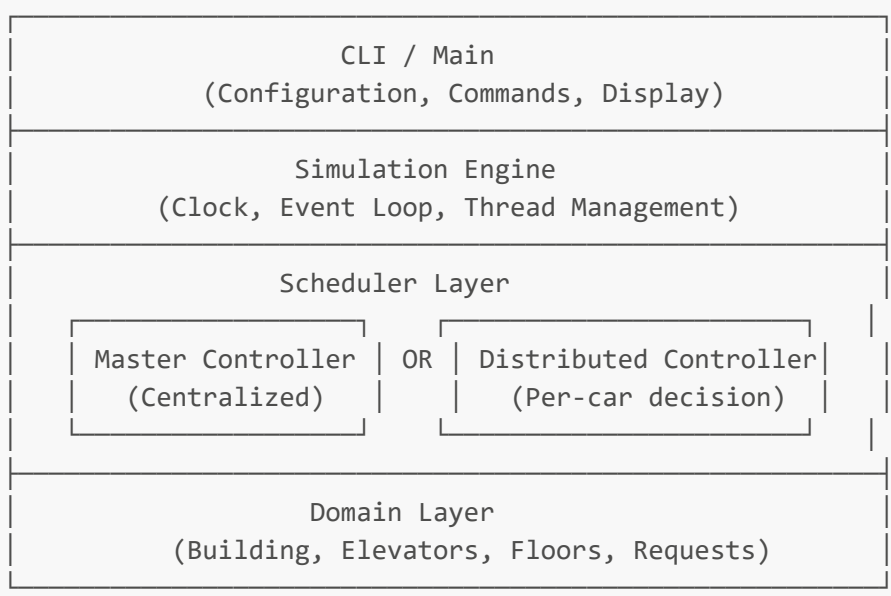
Build a realistic elevator simulation for a building with:

- **1-12 floors**
- **1-3 elevator cars**
- **Two control architectures**: Master Controller (centralized) and Distributed Controller (peer-based)

Purpose: Validate scheduling logic, concurrency correctness, and modular design.

---

2. System Architecture



---

3. Core Components

3.1 Domain Objects

Component	Responsibility
Building	Holds floors, elevators, and hall call state
Elevator	Tracks position, direction, state, car calls, passengers
Floor	Up/Down button state, waiting passengers

3.2 Controllers

Master Controller (Centralized)

- Single decision-maker for all elevators
- Receives all hall calls, assigns to optimal elevator
- Uses LOOK algorithm (reverses when no requests ahead)
- Pros: Global optimization, simpler coordination
- Cons: Single point of failure

Distributed Controller (Peer-based)

- Each elevator makes independent decisions
- Uses claim board for hall call coordination
- Nearest idle elevator claims the request
- Pros: Fault tolerant, scalable
- Cons: May have suboptimal global decisions

3.3 Simulation Engine

- Manages simulation clock (tick-based or real-time)
- Spawns and coordinates elevator threads
- Handles event dispatch and shutdown

---

4. Threading Model

```
Main Thread
    │
    ├── Simulation Engine (clock ticks)
    │
    ├── Elevator Thread 1 ──┐
    ├── Elevator Thread 2 ──┼── Each runs its own loop
    └── Elevator Thread 3 ──┘

Synchronization:
- EventQueue: mutex + condition_variable
- Shared state: protected by mutex
- No busy loops (condition_variable.wait)
```

---

5. Request Flow

Hall Call (person presses Up/Down on a floor)

```
User Input → Building registers hall call → Controller notified
    → Controller assigns elevator → Elevator moves → Doors open
    → Hall call cleared
```

Car Call (person selects floor inside elevator)

```
User Input → Elevator receives car call → Added to car's stop list
    → Elevator serves when passing → Doors open → Car call cleared
```

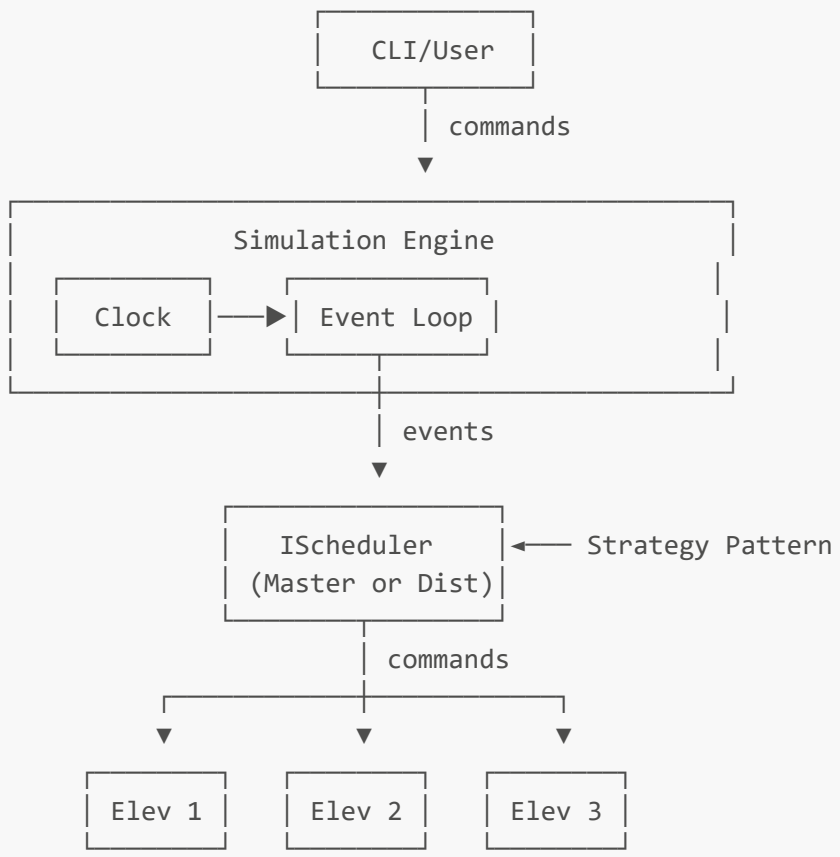
---

6. Key Design Decisions

Decision	Choice	Rationale
Controller pattern	Strategy	Swap Master/Distributed at runtime
Time model	Tick-based	Deterministic, easier testing
Inter-thread comm	Event queue	Decoupled, thread-safe
Memory management	Smart pointers	No leaks, RAII
Scheduling algo	LOOK	Efficient, fair, widely used

---

7. Data Flow Diagram



---

8. Configuration Parameters

Parameter	Range	Default

Parameter	Range	Default
Number of floors	1-12	10
Number of elevators	1-3	3
Car capacity	1-10	6
Controller mode	master/distributed	master
Simulation mode	tick/realtime	tick
Tick duration (ms)	100-1000	500

---

9. Quality Requirements

- **Thread Safety**: All shared state protected, no data races
- **No Deadlocks**: Consistent lock ordering, timeout mechanisms
- **No Memory Leaks**: Smart pointers, RAII, validated with sanitizers
- **Testability**: >60% unit test coverage
- **Build**: Clean compile with `-Wall -Wextra -Werror`

---

10. Technology Stack

Aspect	Choice
Language	C++17/C++20
Build	CMake
Threading	std::thread, mutex, condition_variable
Testing	GoogleTest
Containers	STL (vector, set, queue, map)