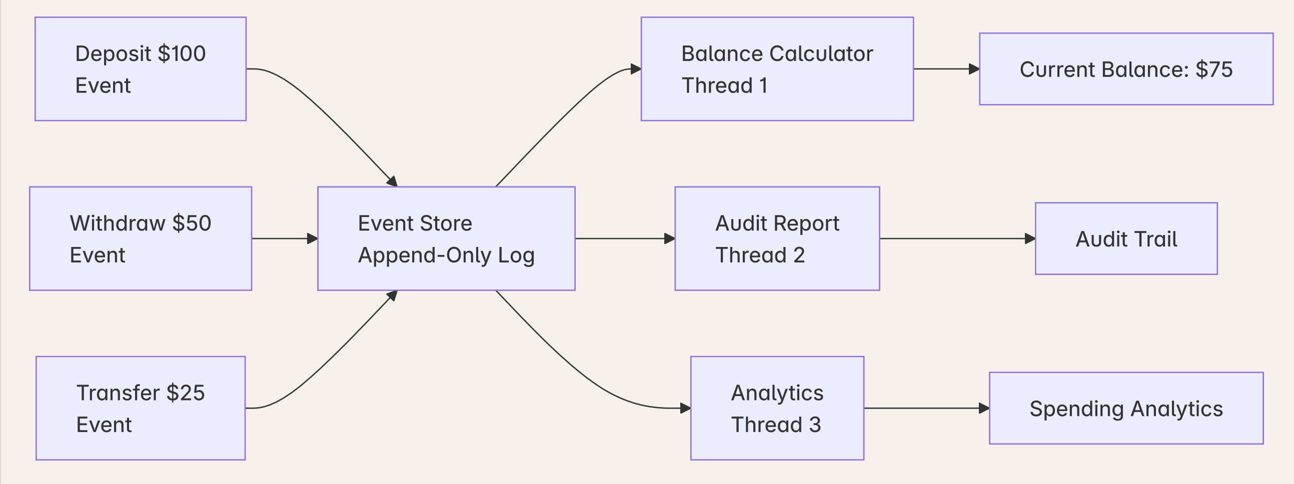
Event Sourcing Pattern

Instead of storing current state, store a sequence of events. Current state is derived by replaying events.

How it works:

- Every state change of a **system** is recorded as an event
- To get current state, replay all events from the beginning.
- Events are immutable and append-only.
- Multiple threads can read events safely.
- Writing needs to be locked.

Example



Real-world example: Banking system where instead of storing account balance, you store "deposited $100", "withdrew $50", etc. Balance is calculated by summing all events.

Benefits: Complete audit trail, easy to debug, natural concurrency for reads.

Design Considerations:

Choosing

Ask yourself this when choosing the design pattern:

- Do you need a complete audit trail of all changes?
- Is your domain naturally event-driven?
- Do you need to support temporal queries (state at any point in time)?

- Do you have complex business rules that benefit from event replay?
- Do you need multiple read models derived from the same data?
- Are you building a system where debugging requires understanding what happened?

Building Outline

Event Design:

- What granularity should events have?
- How do you version events as your system evolves?
- Are events immutable once stored?
- What's your event naming and schema strategy?
- How do you handle event correlation and causation?

Event Store:

- What's your event storage strategy? (SQL, NoSQL, specialized event store)
- How do you handle event ordering and consistency?
- What's your partitioning strategy for large event volumes?
- How do you handle event store scaling and replication?
- What's your backup and recovery strategy?

Event Processing:

- How do you replay events to rebuild state?
- What's your strategy for handling event processing failures?
- How do you ensure event processors can handle duplicate events?
- Do you need real-time vs batch event processing?

Snapshots:

- When do you create snapshots to optimize replay performance?
- What's your snapshot storage and retrieval strategy?
- How do you handle snapshot corruption or inconsistency?
- How do you balance snapshot frequency with storage costs?

Multi-threading Considerations:

- Can multiple threads safely append events?
- How do you handle concurrent event processing?
- What's your strategy for maintaining event ordering?
- How do you handle event processor scaling?