Zookeeper

Data at Scale

Zookeeper

- Hadoop’s distributed coordination server
- Design Goals
  - Simplicity
    - Distributed processes coordinate through a shared hierarchical namespace — znodes
  - Reliability
    - Uses replication

Zookeeper

- Clients communicate through a file like system
- Zookeeper implements:
  - Wait-free
  - FIFO execution of requests per client
  - Linearizability for all requests that change ZooKeeper state

Zookeeper

- Coordination between processes
  - Agreement on configuration
    - Leader election
    - Group membership
    - Locks

Zookeeper

- Other solutions:
  - Amazon simple queue service
    - Provides just queuing
  - Protocols for leader election
  - Protocols for common configurations
  - Chubby for locking with strong synchronization guarantees

Zookeeper

- Zookeeper:
    - Generic
    - Takes form of file server instead of e.g. locking

Zookeeper

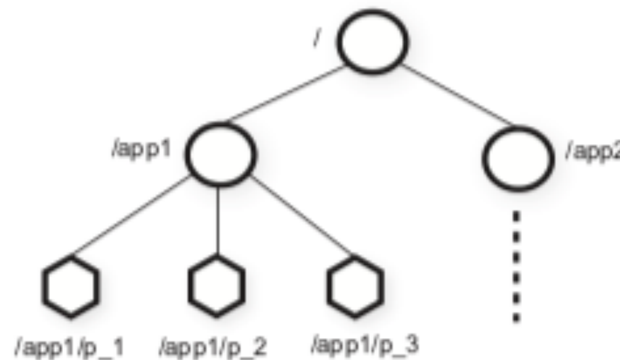
- Zookeeper:
  - Guarantees FIFO client ordering
  - Global linearizability of writes
  - Using replicated servers

Zookeeper Service

- znodes: in-memory data nodes with Zookeeper data
  - Data is organized in a data tree

Zookeeper Service

- Zookeeper provides an abstraction to clients
  - znodes are organized in a hierarchy



  - znodes can be regular
    - Created and deleted explicitly
  - znodes can be ephemeral
    - Clients create znodes, but system can remove them at end of session

Zookeeper Service

- Znodes can be sequential
  - When created, a counter is added to their name

Zookeeper Service

- Zookeeper has watches:
  - When a client issues a read operation with *watch flag* set
    - Operation returns as normal
    - But client is informed of any subsequent changes in the value

Zookeeper Service

- Data Model
  - znodes look like a file system
  - only store meta-data used for coordination among servers
    - E.g. for leader selection:
      - leader stores its name after election
      - so newly joining nodes can find the name of the leader

Zookeeper Service

- Sessions:
  - Zookeeper client connects to Zookeeper and initiates a session
  - Sessions have a timeout and clients that do not interact for a timeout are considered faulty
  - Allows clients to receive service from more than a single zookeeper server

Zookeeper Service

- Client API
  - create(path, data, flags)
  - delete(path, version)
  - exists(path, watch)
  - getData(path, watch)
  - setData(path, data, version)
  - getChildren(path, watch)
  - sync(path)
    - waits for all pending updates to propagate to servers

Zookeeper Service

- Client API
  - Synchronous API for single ZooKeeper operations
  - Asynchronous API if there are outstanding operations and other tasks are executed in parallel
    - Client then has to guarantee that callbacks are invoked in order

Zookeeper Service

- Zookeeper guarantees:
  - Linearizable writes:
    - all requests that update the state of Zookeeper are serializable and respect precedence
    - clients can have more than one request outstanding
  - FIFO client order:
    - all requests from a given client are executed in the order that they were sent by the client

Zookeeper Service

- Example
  - A system elects a leader
    - New leader changes a large number of configuration parameters
    - New leader notifies other processes when finished
  - Two Requirements
    - 1: While the leader makes changes, no other process should use configurations undergoing changes
    - 2: If the new leader dies, no process should use partial configurations

Zookeeper Service

- Example:
    - Locking can help with 1, but not with 2
  - Zookeeper:
    - Leader crates the *ready* znode
    - Other processes will only use the configuration if that znode exists
    - New leader
      1. deletes current ready znode
      2. writes configuration znodes
      3. creates ready znode
    - All changes are pipelined for fast parallel processing
    - A client that sees ready is assured that all configuration znodes have been written by current leader
    - Watches will prevent clients to confuse an old ready with a new ready znode

Zookeeper Service

- Second example:
  - Processes A and B have an outside communication channel
  - Process A makes changes and informs B of these changes
  - Process B now expect to see the changed znodes
    - But B’s znode replica can be behind A-s
  - Zookeeper solution:
    - B can issue a write to the znode
    - Guaranteed that any reads afterwards have new values
    - This is the purpose of the *sync* command

Zookeeper Service

- Implementing simple locks
  - Create a znode with a lock-file
  - Clients create znode lock file with ‘ephemeral’
  - If the creation succeeds, then client has the lock
  - Otherwise, client reads the lock with “watch” set
    - Which notifies it when current lockholder destroys the file
  - Client releases a lock if client dies are explicitly deletes the lock

Zookeeper Service

- Implementing locks without herd effect
  - Line up all clients requesting the lock and each client obtains the lock in order of request arrival

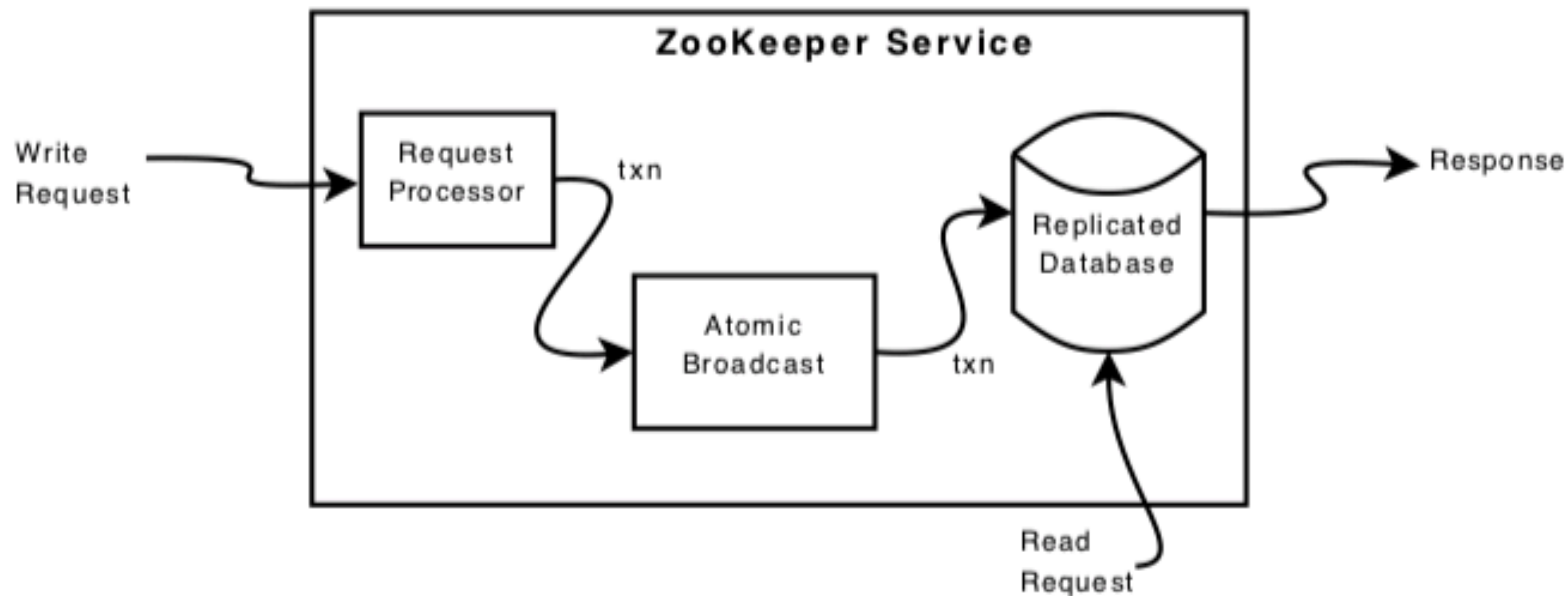
```
Lock
1 n = create(l + “/lock-”, EPHEMERAL|SEQUENTIAL)
2 C = getChildren(l, false)
3 if n is lowest znode in C, exit
4 p = znode in C ordered just before n
5 if exists(p, true) wait for watch event 6 goto 2

Unlock
1 delete(n)
```

  - “Sequential” orders the clients’ attempts to obtain lock

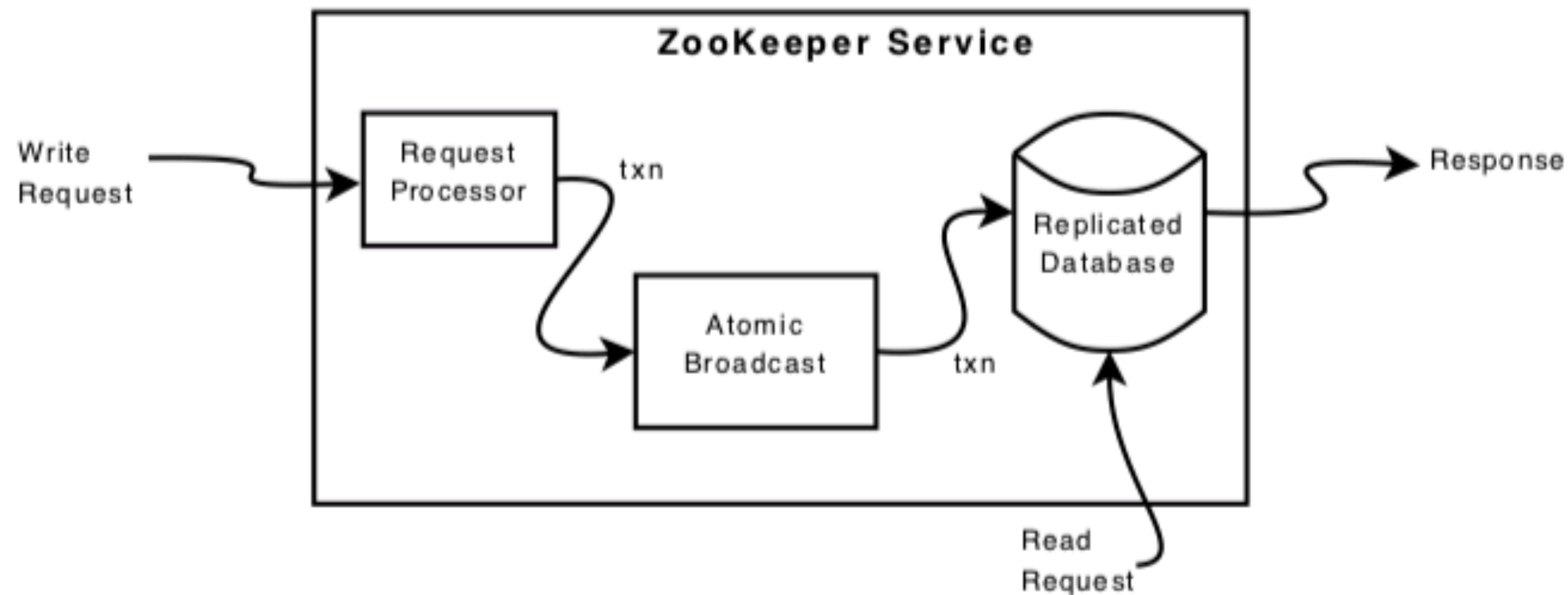
Zookeeper Implementation

- Reliability through replication
- Service components:



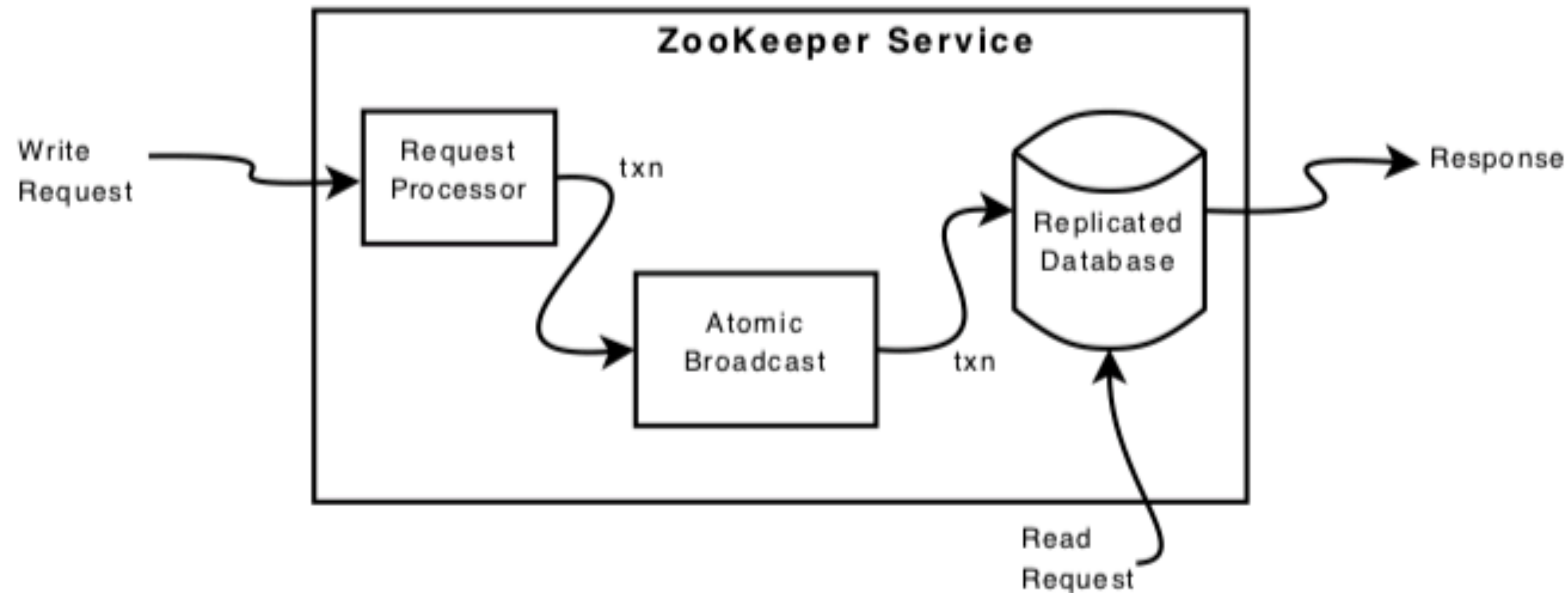
Zookeeper Implementation

- Server receives client request and prepares it for execution (request processor)



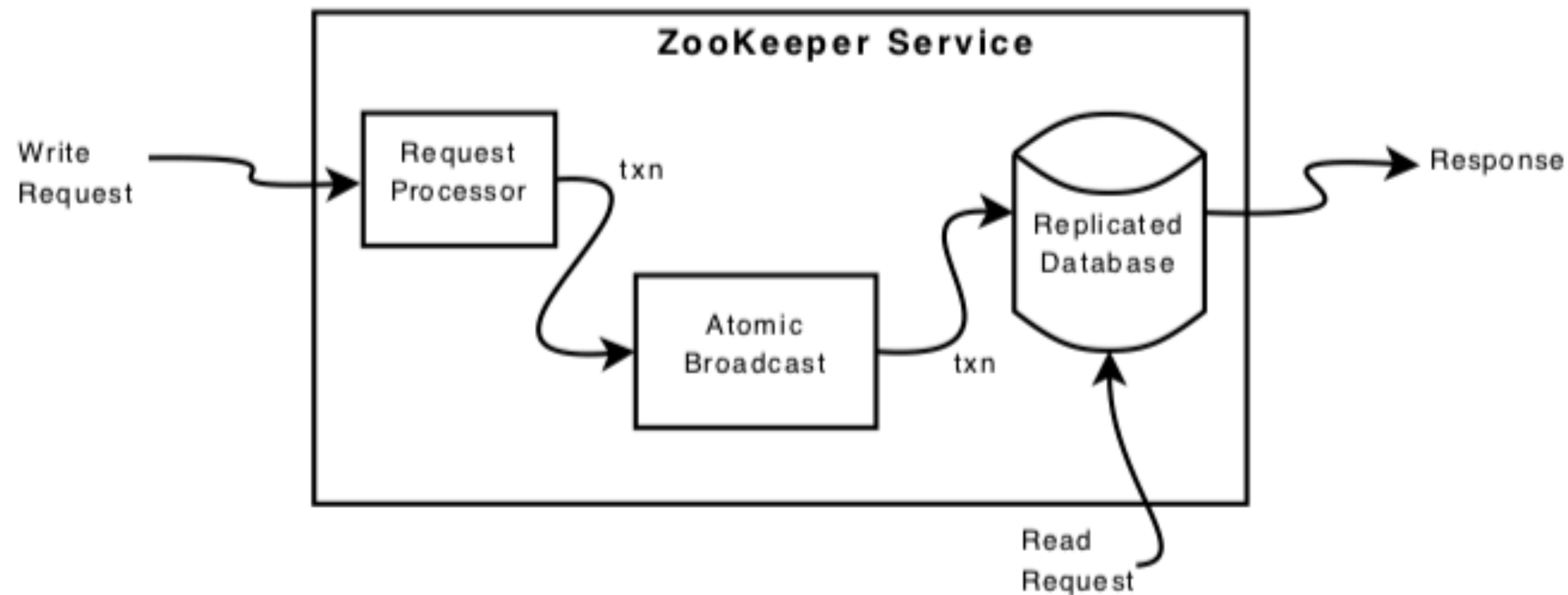
Zookeeper Implementation

- If request is a write:
  - Use agreement protocol
  - Commit across all servers in the ensemble



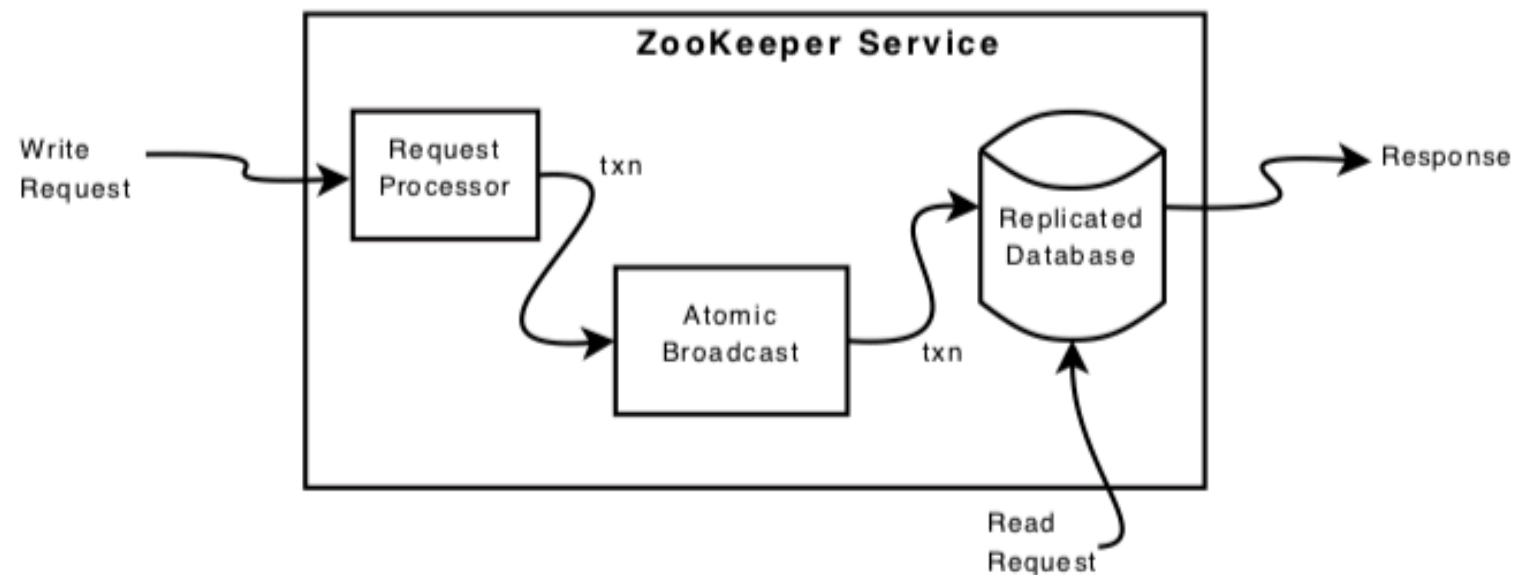
Zookeeper Implementation

- If request is a read:
  - Request processor just reads replicated database



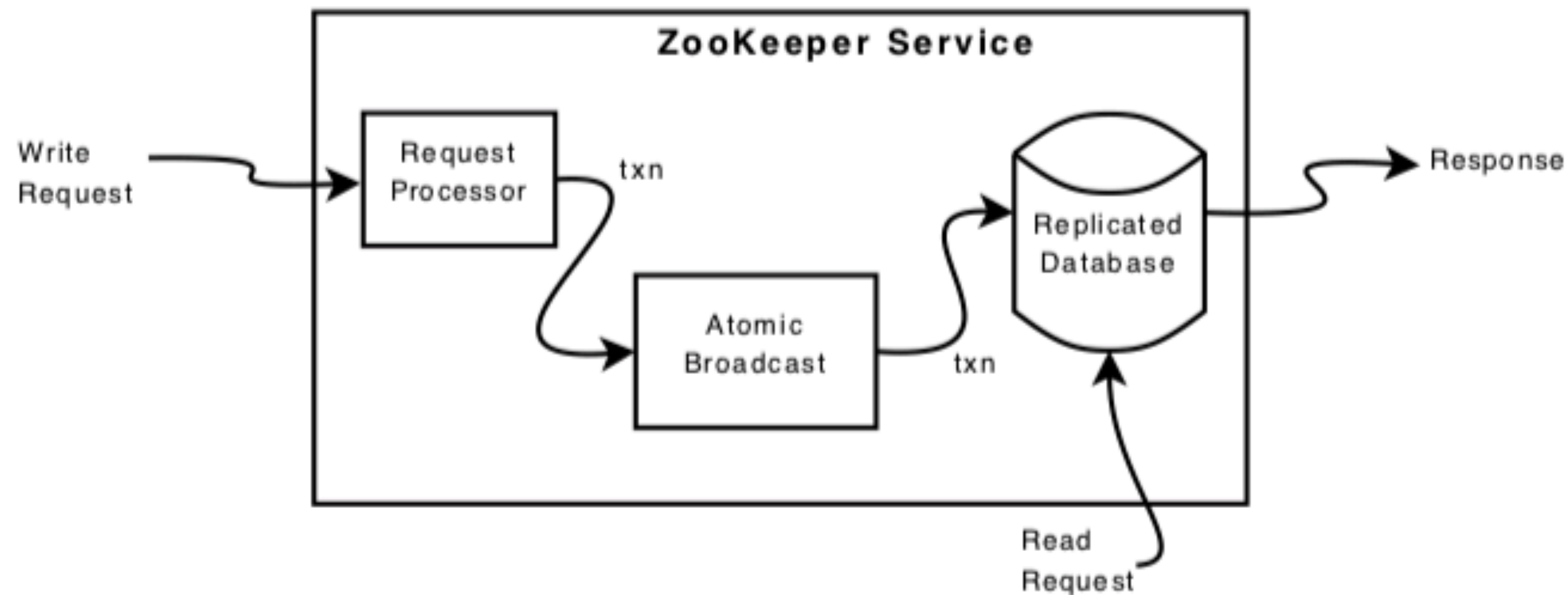
Zookeeper Implementation

- Replicated database is *in-memory*
  - Each znode stores 1MB maximum
  - Updates are logged to disk for recoverability (replay log)
  - Log writes are forced



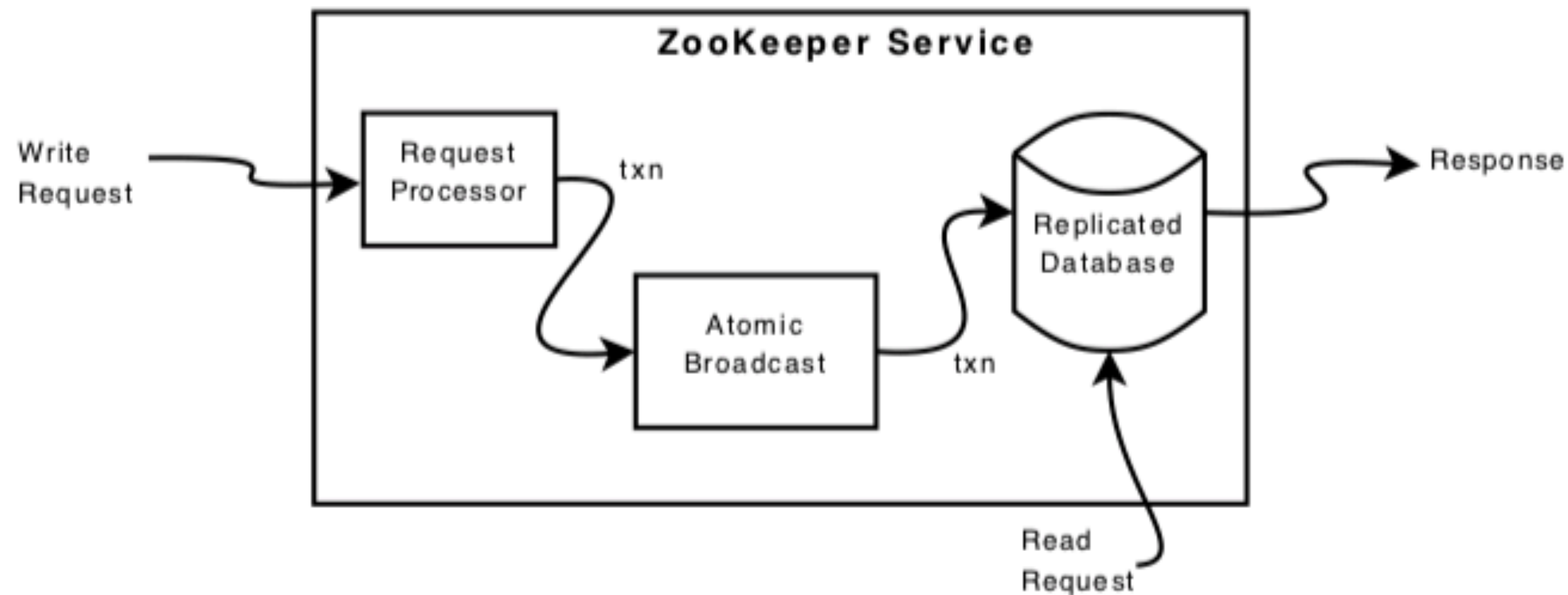
Zookeeper Implementation

- Clients connect to exactly one server



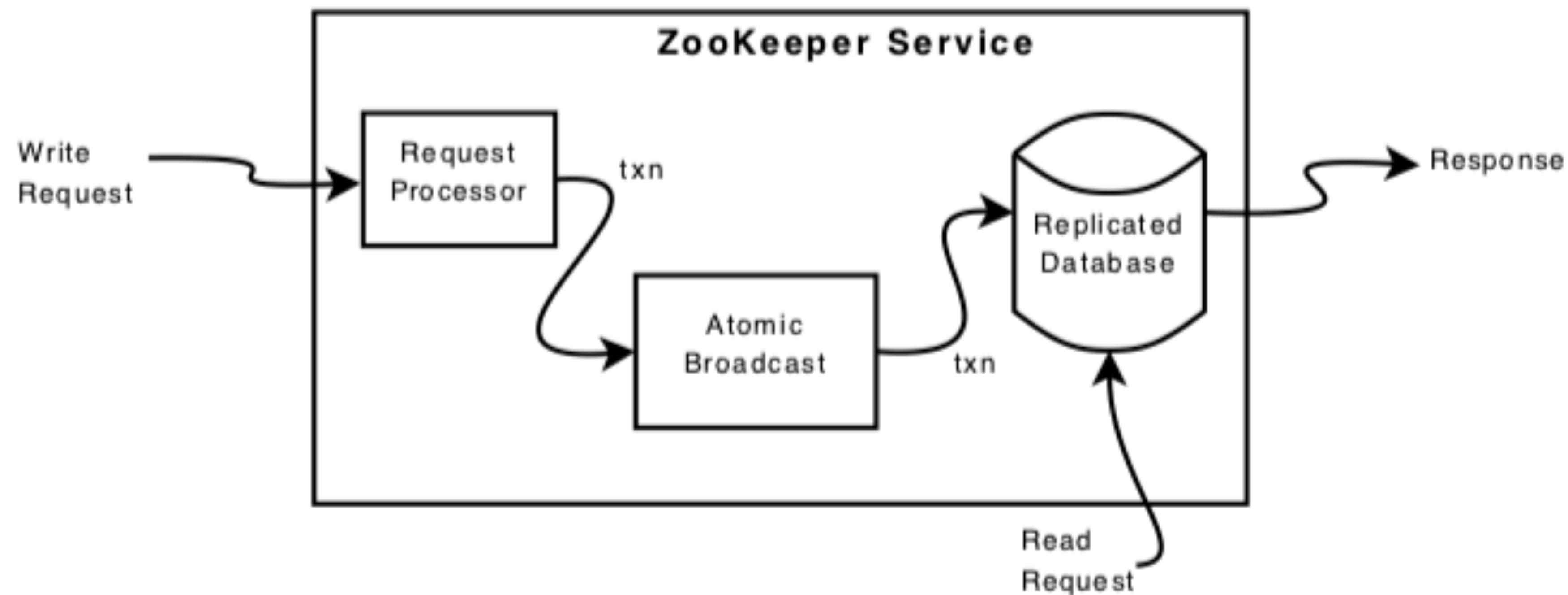
Zookeeper Implementation

- Agreement protocol:
  - write requests are forwarded to a *single* server, the *leader*
  - other zookeeper servers are *followers*



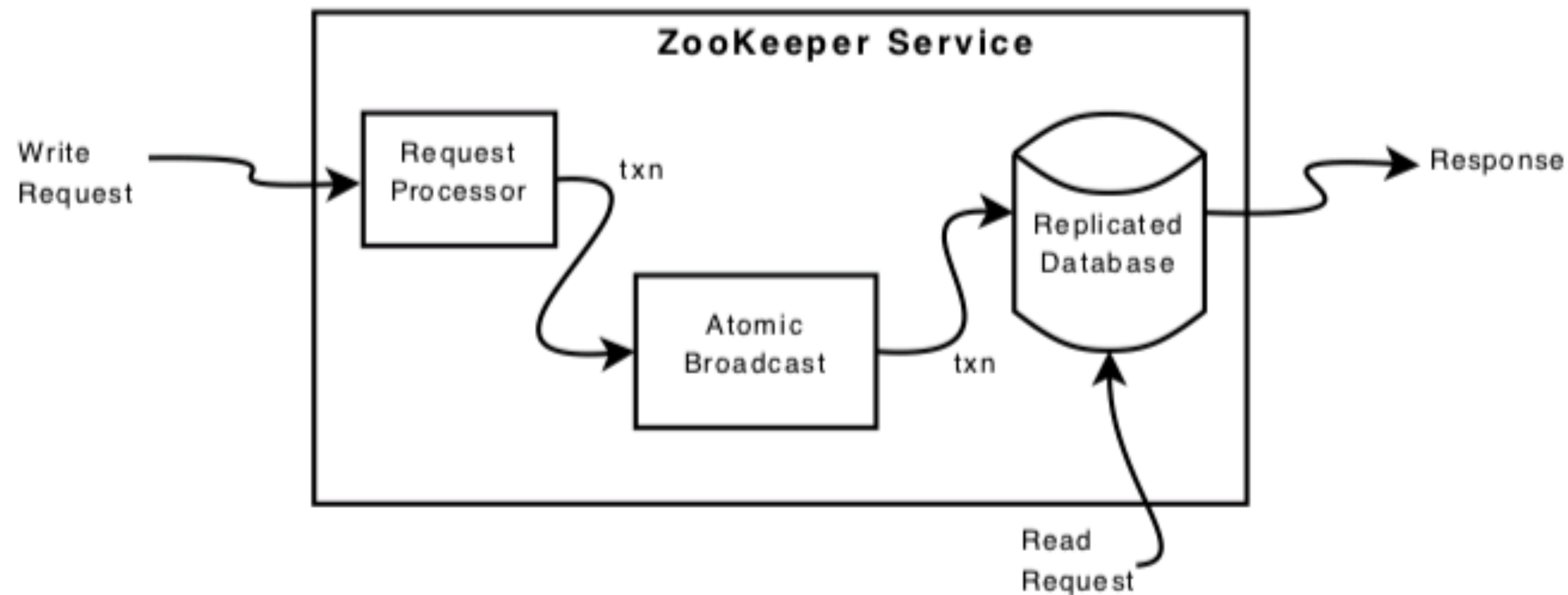
Zookeeper Implementation

- Requests generated by request processor are *idempotent*
  - Could be applied twice or more without changing effect



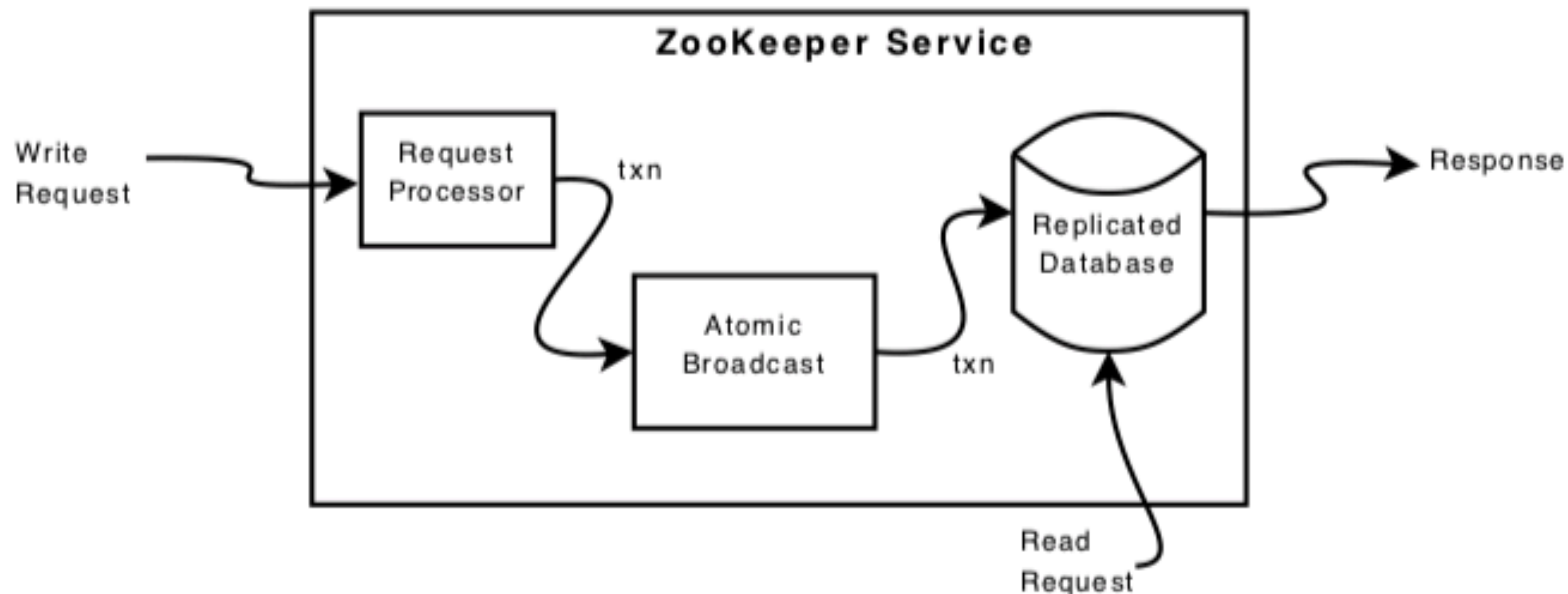
Zookeeper Implementation

- All requests are broadcast (via ZAB)
    - ZAB uses a simple majority quorum to decide on a proposal



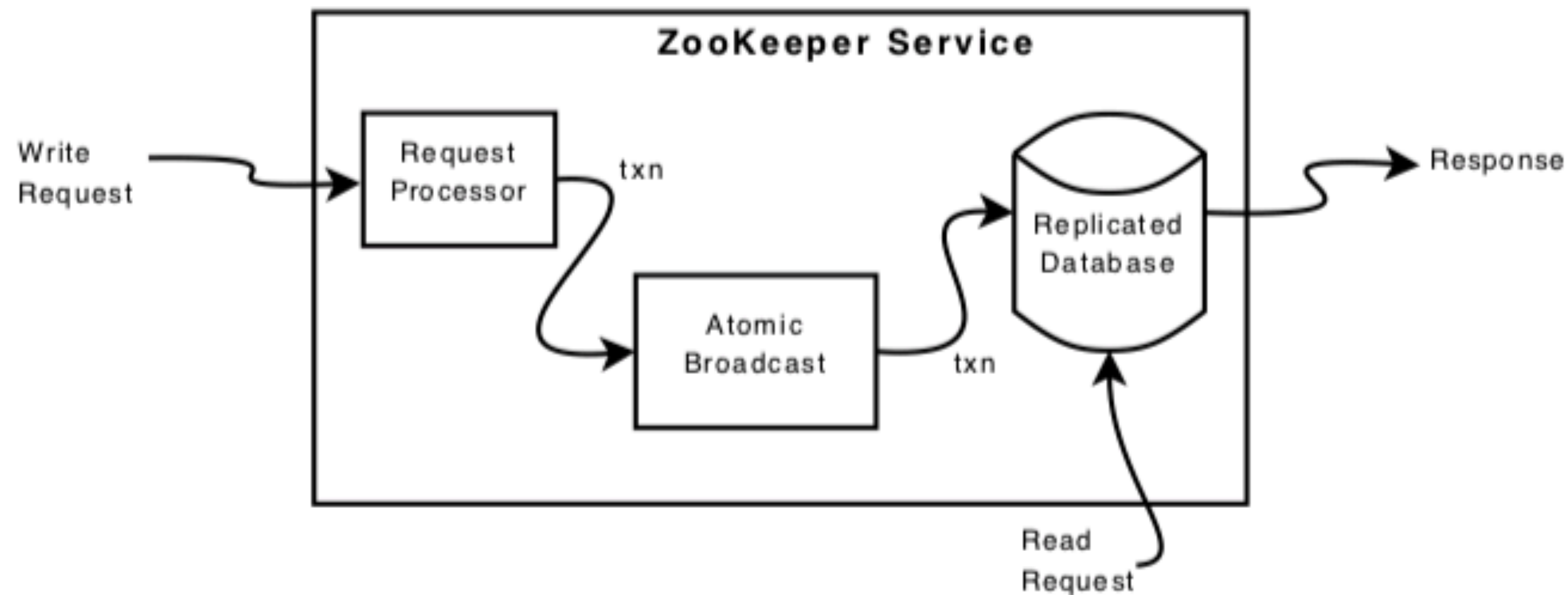
Zookeeper Implementation

- Each replica of DB has a copy in memory of Zookeeper state
- To recover state, use *fuzzy snapshots* (without locking)
    - Possible because of idempotency



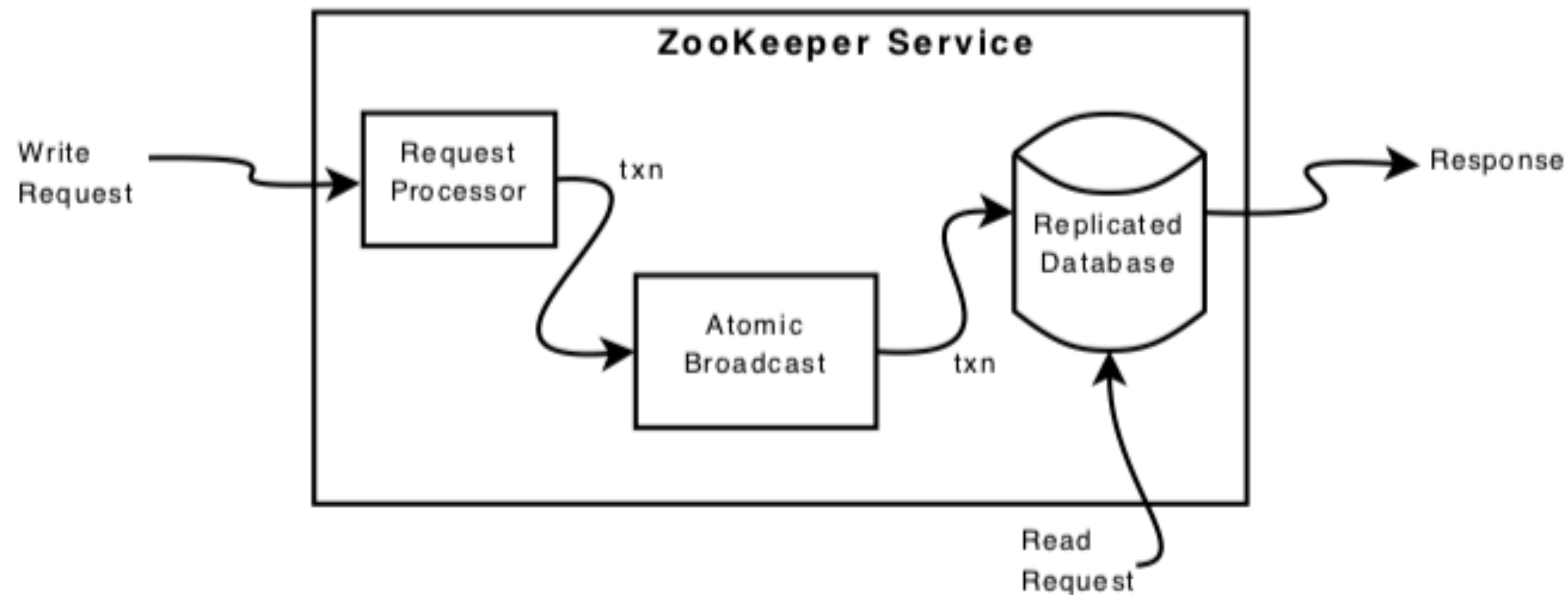
Zookeeper Implementation

- If a server processes a write request:
  - Sends out notification to any watches



Zookeeper Implementation

- Fast reads:
  - Reads are not coordinated
    - No guarantee for precedence



Zookeeper Applications

- Fetching Service at Yahoo!
  - crawls billions of web documents
  - Has master processes that command page-fetching processes
  - Masters provide fetchers with configuration
- Main advantage of using ZooKeeper
  - Recovery from failure of masters

Zookeeper Applications

- Yahoo! Message Broker
  - Manages thousands of topics
    - Clients can publish to topics and receive updates
  - Each topic is replicated to two machines
  - ZooKeeper
    - manages distribution of topics
    - deals with failure of machines
    - operates system control

Zookeeper Applications