Practical Lab
Cloud Systems Engineering
(cloud-lab)

Chair of Decentralized Systems Engineering
https://dse.in.tum.de/

Single-node KVS

Task #1:

Task #1

Your task for the next three weeks:

- Implement the server-side of a client-server KVS architecture
- Use
  - (1) RocksDB as the KV store
  - (2) kernel sockets for the networking
  - (3) google protobufs as the serialization protocol

Background

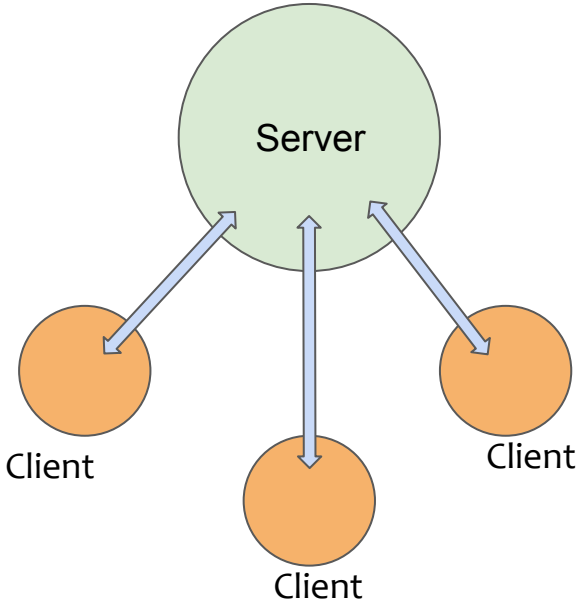
Learning goals

In this task you will learn about:

- Client-server architecture(s)
- Single node key-value stores (KVSs), e.g., RocksDB

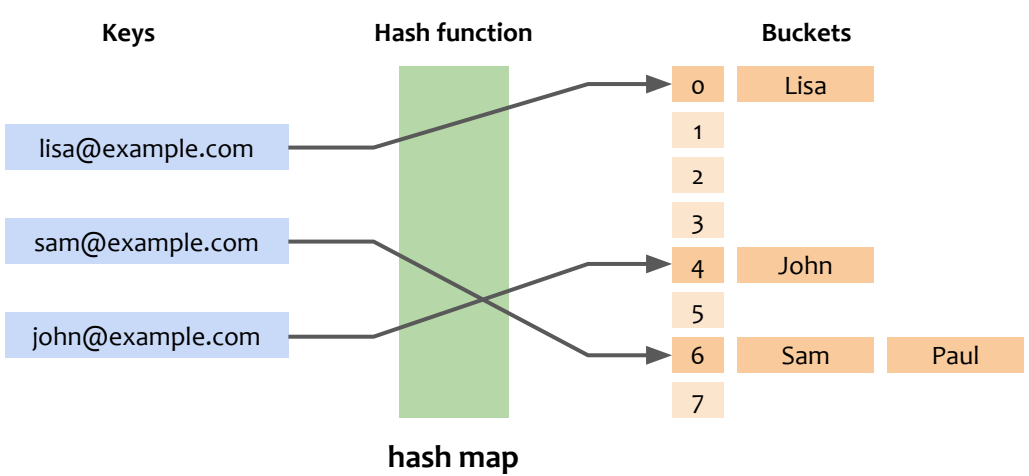
Client-server architecture

- Server
  - Usually a long running process (*daemon process*)
  - Manages some resources
  - Receives and process requests
- Client(s)
  - Sends one or more requests to the server
  - Wait for the server's reply
- Transport layer
  - Network medium
  - Transfers the data



Key-Value store (KVs)

- Data structure
  - stores, retrieves and manages data
  - e.g., dictionaries, hash-tables



Motivation

Key-value stores play an important role at tech giants:

memcached	Redis	Voldemort	Dynamo
Facebook Twitter Zynga	GitHub Digg Blizzard Interactive	LinkedIn	Amazon

Motivation

Key-value stores play an important role in the scientific community:

- FASTER: A Concurrent Key-Value Store with In-Place Updates
  [SIGMOD ‘18]
- KVell: Design and Implementation of a Fast Persistent Key-value Store
  [SOSP ‘19]
- Nova-LSM: A Distributed, Component-based LSM-tree Key-value Store
  [SIGMOD ‘21]

KVs operations

Key-value stores implement at least two operations:

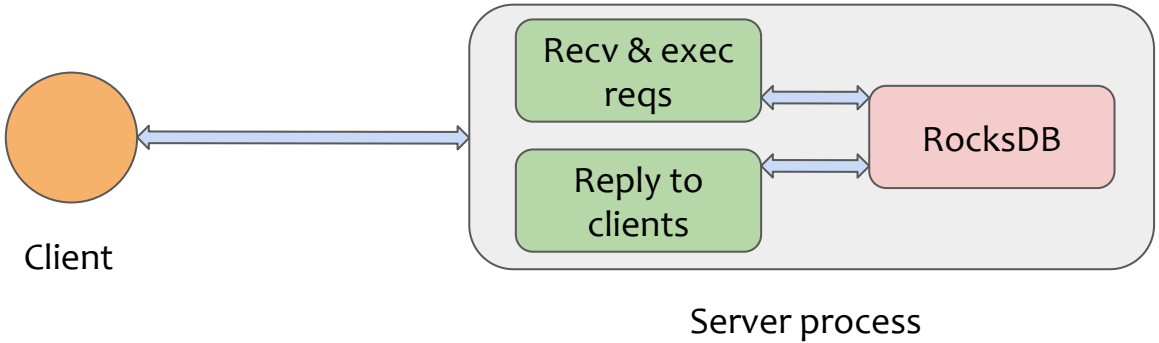
- GET
  - Retrieve a value by key
- PUT
  - Insert or update a key-value pair

Challenges - design goals

- Performance
  - lock contention, significant write-traffic, complex memory management
  - low-latency operations and high throughput (I/O, batching)
  - parallelism (e.g., keys hashing)

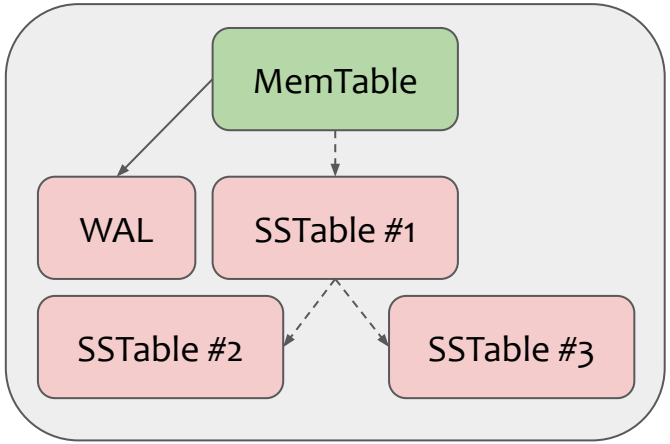
- Data properties
  - Persistency, e.g., persistent KVs or in-memory KVs
  - Consistency, e.g., linearizability or sequential consistency
  - Durability or crash consistency (for persistent KVs)

Task #1: Client/server arch + RocksDB



RocksDB architecture

- LSM-data structure
  - In-memory skiplist (MemTable)
  - SSTable files (persistent) organized on levels with (sorted) KV pairs
  - Compaction (background, multithreaded)
- Data properties
  - Linearizable reads, a read always "sees" the latest write
  - Durability, SSTables are persistent
  - Crash-consistency through Write-Ahead-Log (WAL)
- API
  - supports PUT, GET, DELETE queries



RocksDB

References

- Rocksdb: https://github.com/facebook/rocksdb
- Protobufs: https://developers.google.com/protocol-buffers

Thank you for listening!

See you in the Q&A session