Table Enumeration in RAMCloud

Elliott Slaughter

Overview

- Motivation
- Consistency Goals
- RAMCloud Internals
- Enumerate Algorithm

What is enumeration?

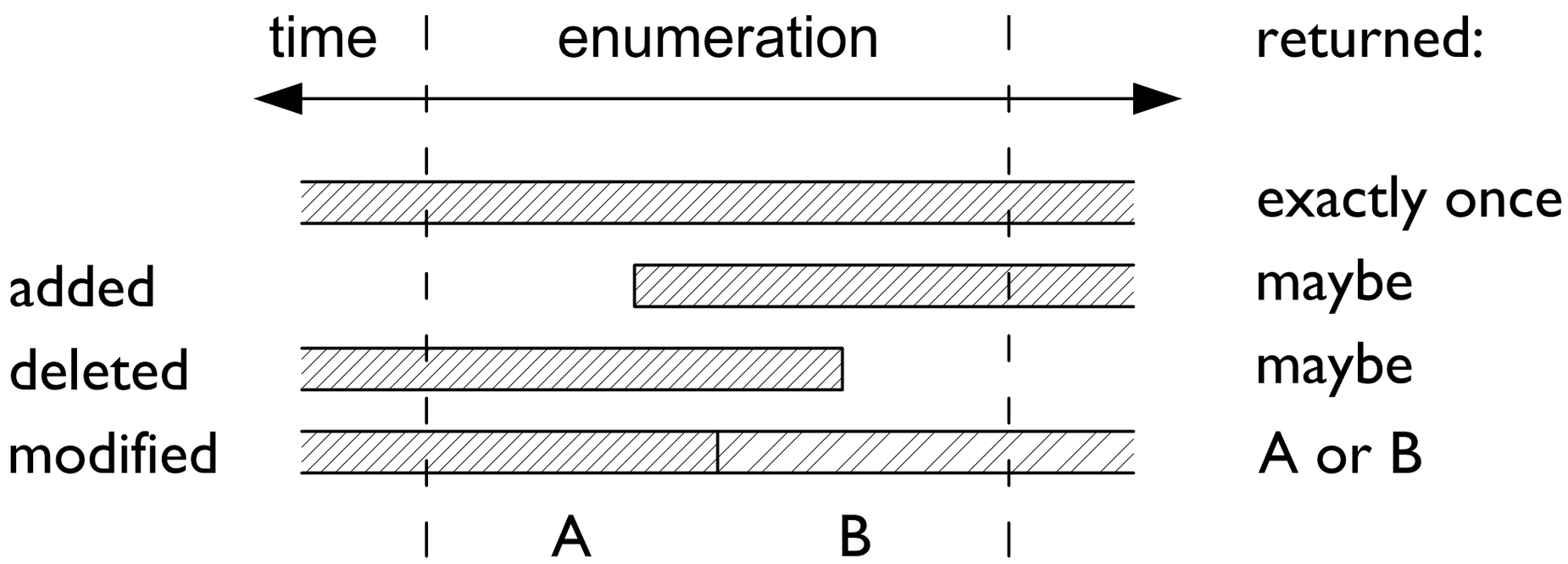
- Distributed key/value store
- Iterate over all key/value pairs in a table
- SELECT *
- Use cases:
  - “ls”
  - Offline backups

Pitfalls in enumeration

- Table can be modified during enumeration
- Table might be distributed on multiple servers
- Need multiple requests to fetch a table
- Servers can go up and down
- Table can be redistributed across the cluster
- Etc.

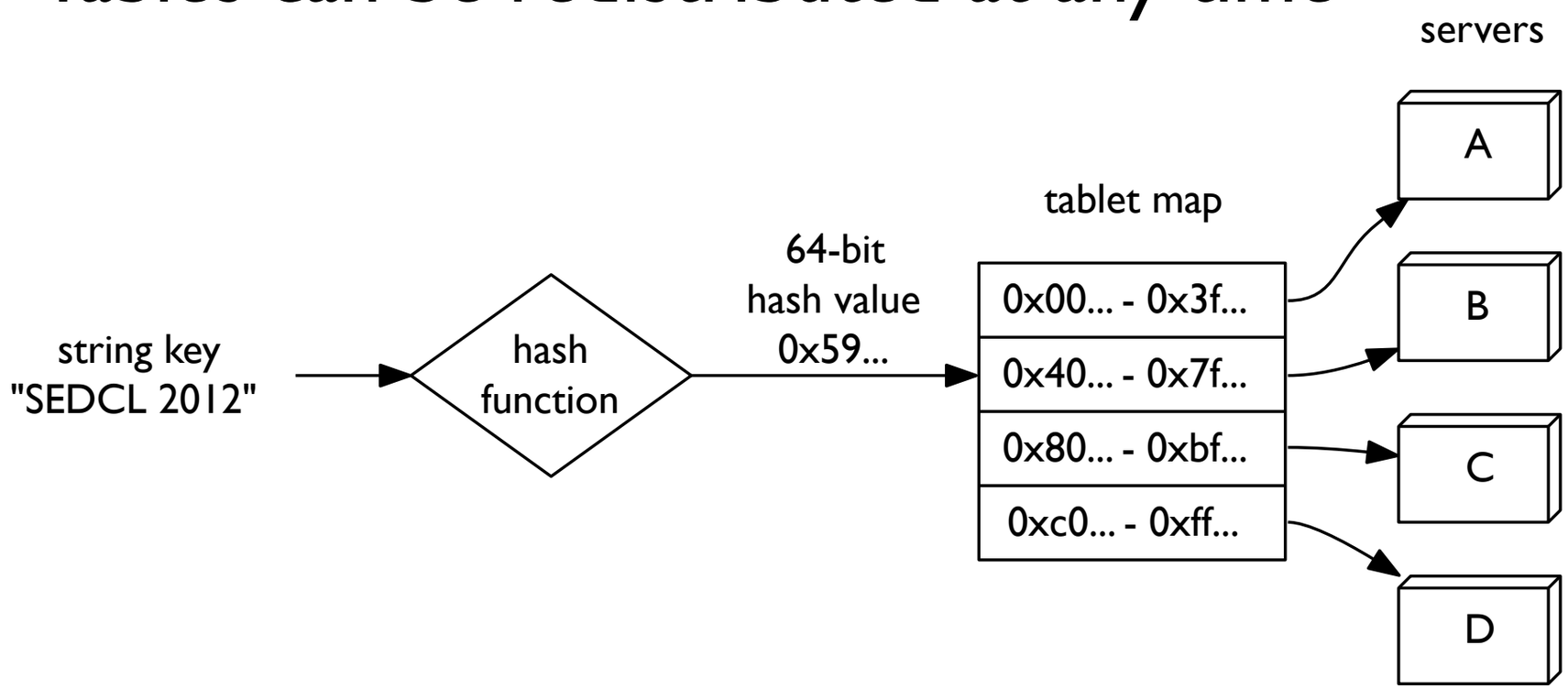
Consistency goals

- "Once-only" consistency model:
  - Any object whose lifetime spans the entire enumeration will be returned exactly once
  - No object will be returned more than once



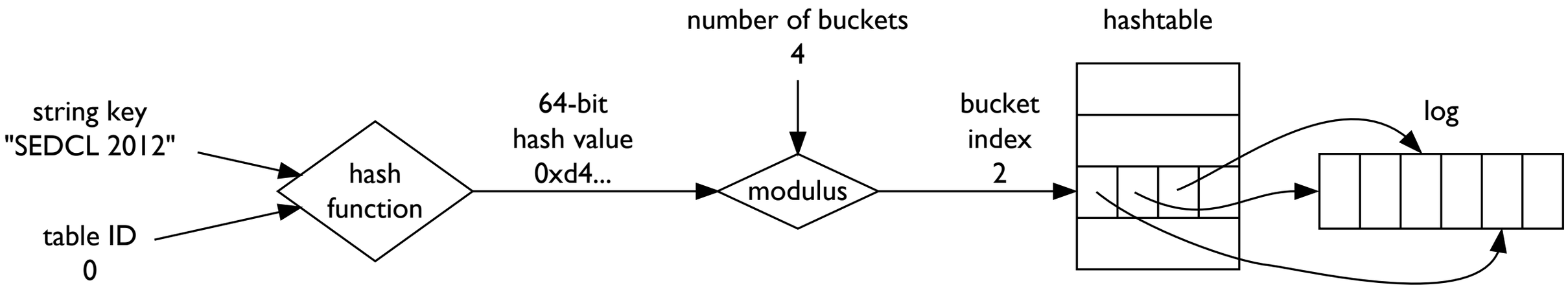
How clients locate objects on servers

- Clients locate objects by hashing keys
- Table is divided into chunks in the hash space and distributed among servers
- Tables can be redistributed at any time



How servers locate objects for clients

- Servers use a hash table to locate objects in their log
- Many different tablets may be collocated in the hash table



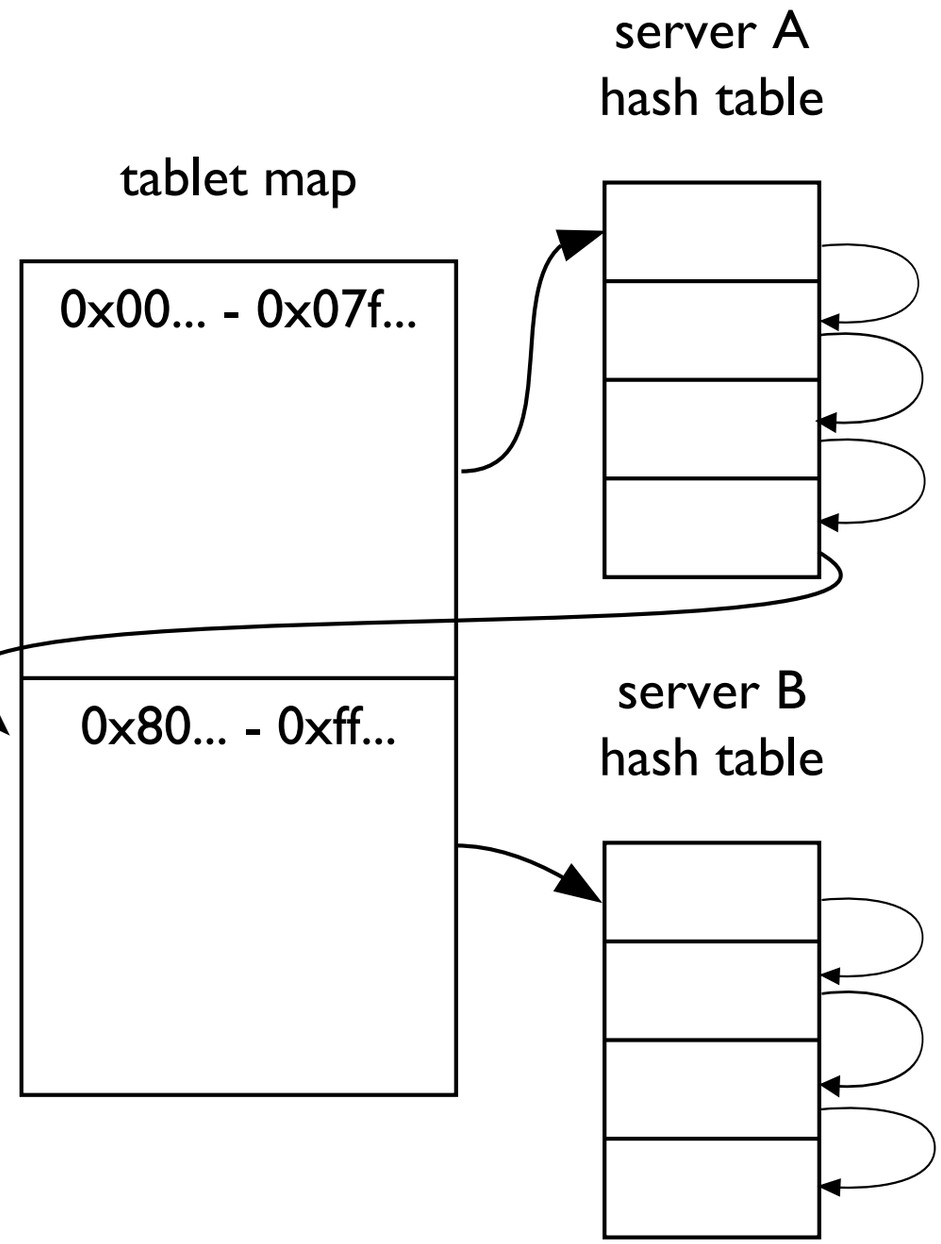
The client algorithm

- Suppose we can encapsulate the state of iteration in an opaque blob
- Enumerate(tabletStartHash, iterator) => nextTabletStartHash, nextIterator, objects
- Just call this RPC repeatedly until we receive no objects back

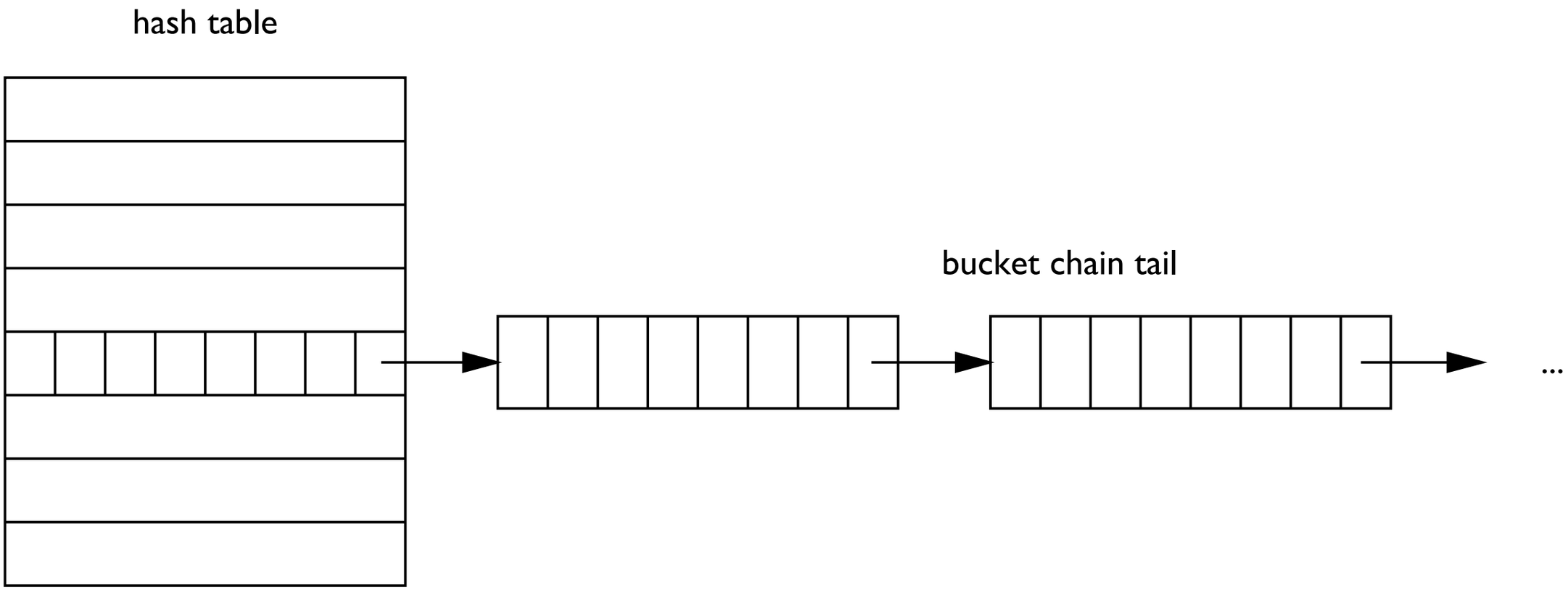
The server algorithm: Naive version

- Client iterates tablets front to back
- Server walks hash table in bucket order and collects objects

Iterator contains:
`bucketIndex`

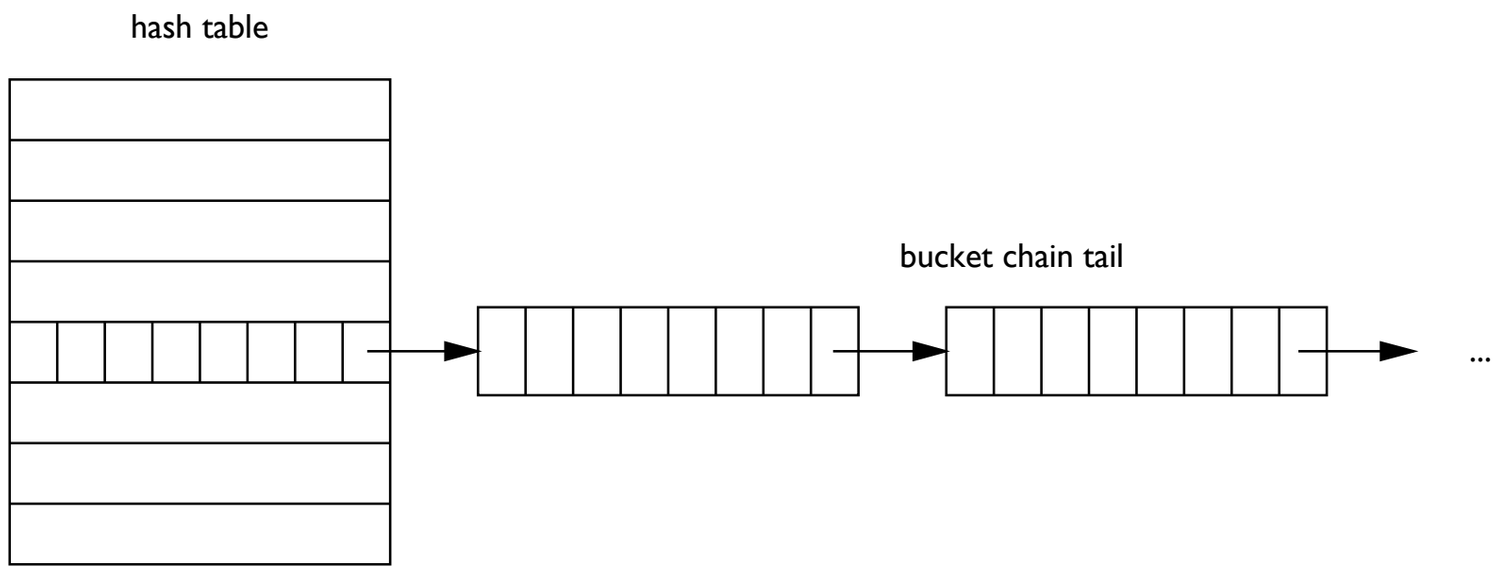


Problem: Large buckets



- No guarantees on ordering of elements in bucket

Solution: Large buckets

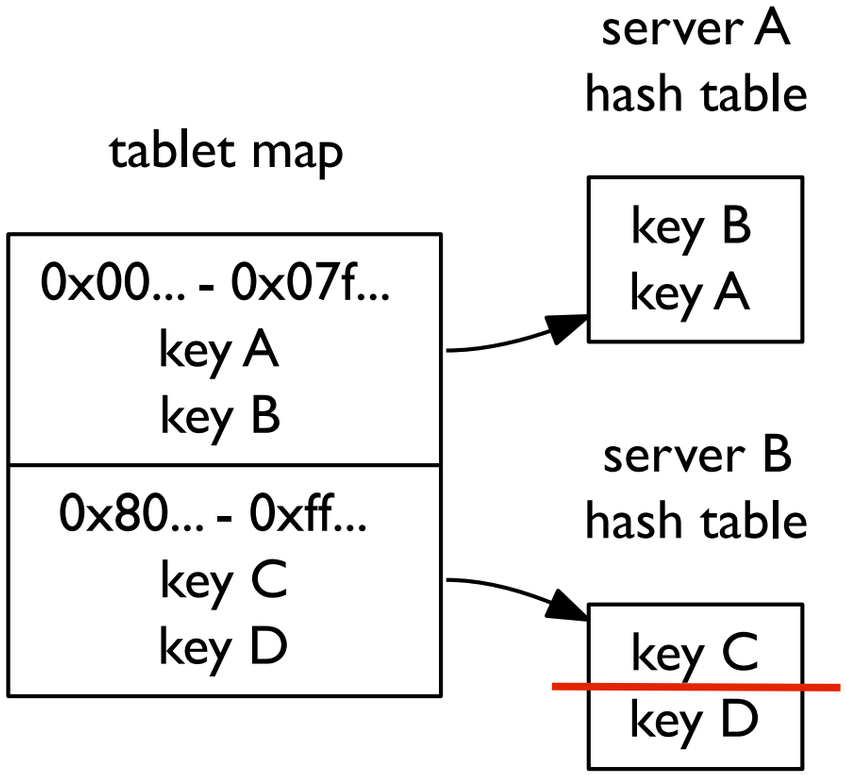


- Sort large buckets by hash value, and keep track of the next bucket hash in the iterator

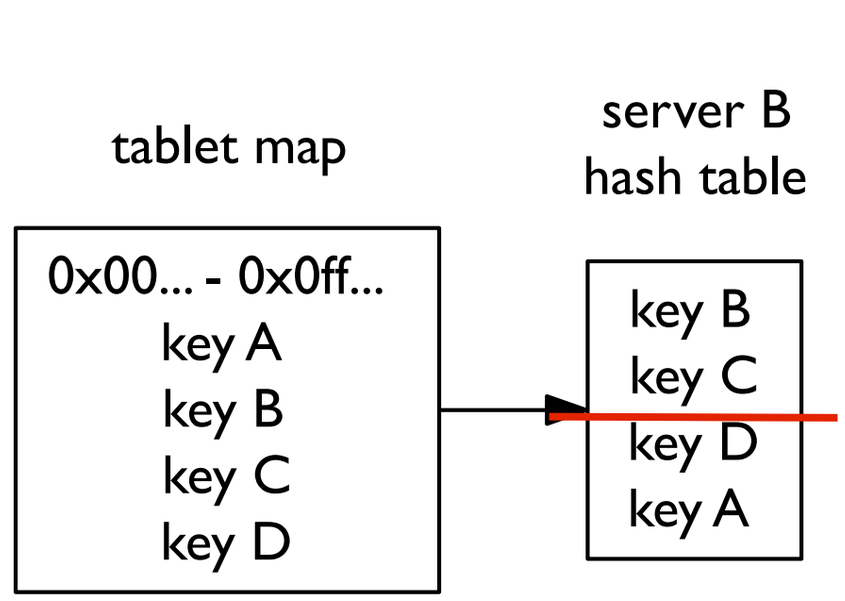
Iterator contains:
`bucketIndex`
`nextBucketHash`

Problem: Tablet reconfiguration

Before merge



After merge

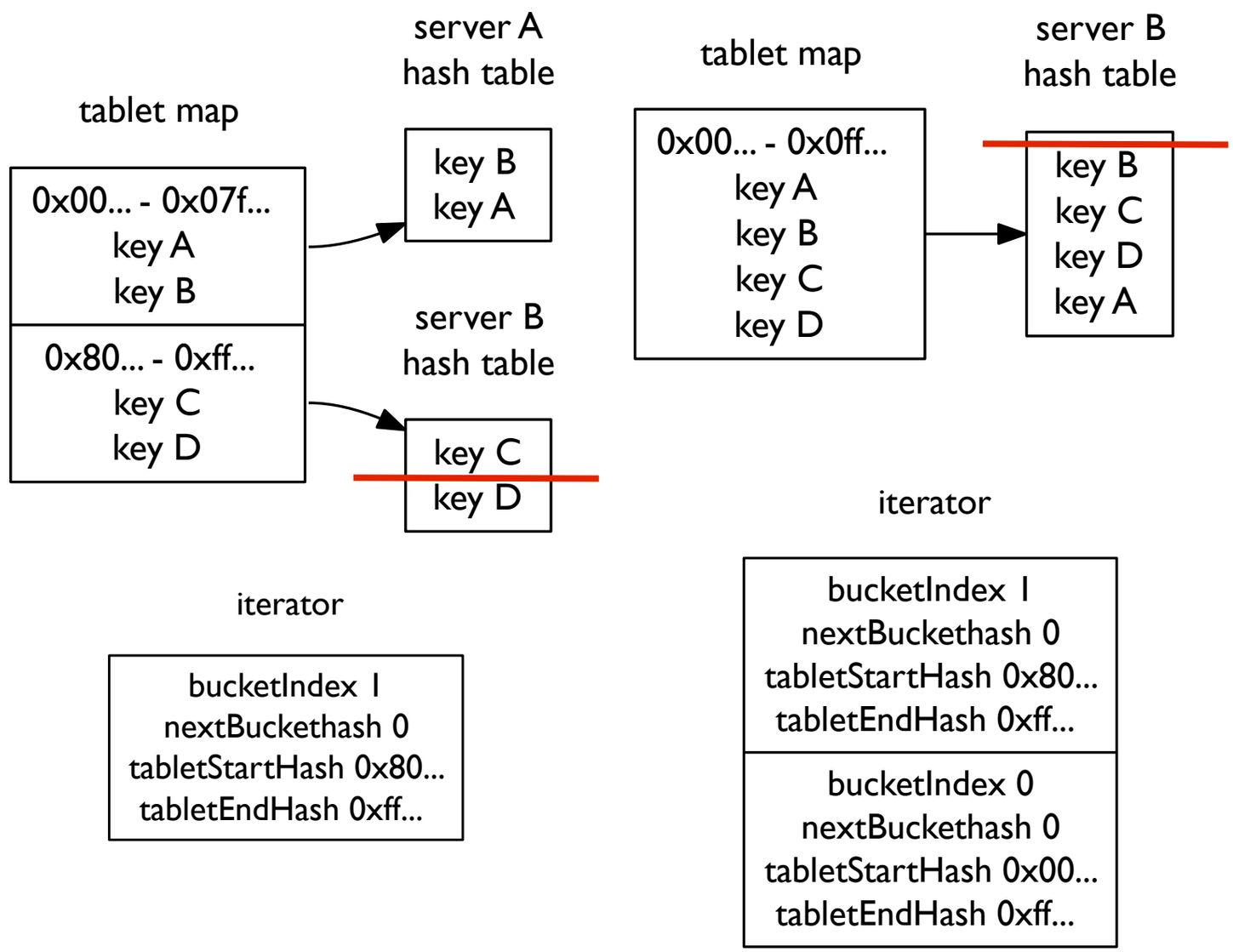


Solution: Tablet reconfiguration

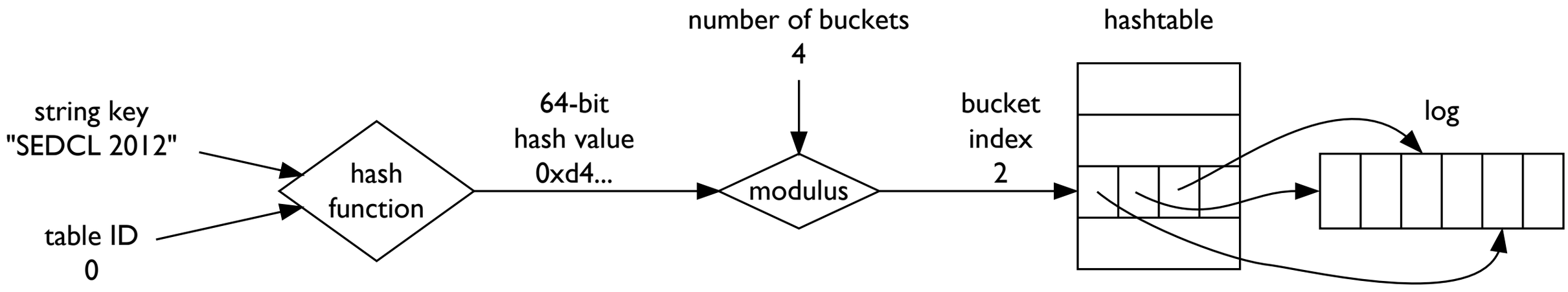
- Iterator is now a stack
- Keep track of tablet boundaries

Iterator contains:
`bucketIndex`
`nextBucketHash`
`tabletStartHash`
`tabletEndHash`

*

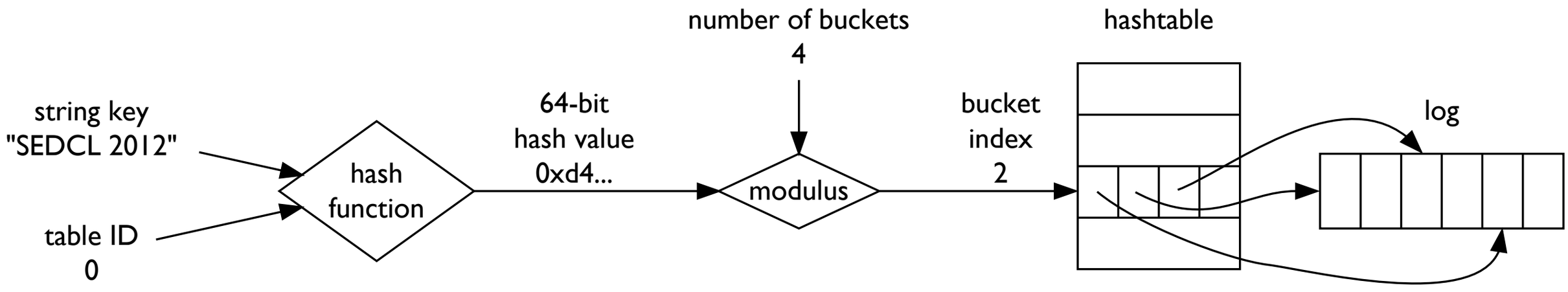


Problem: Rehashing



- Mapping to buckets depends on size of hash table
- Different servers might have different sizes of hash tables

Solution: Rehashing



- Track the size of the hash table to handle resizing the table

Iterator contains:

```
bucketIndex
nextBucketHash
tabletStartHash  *
tabletEndHash
numBuckets
```

Conclusions

- Pros:
  - Iterator contains entire state (server is stateless)
  - No extra data structures
- Cons:
  - Efficiency?
  - Consistency?

Iterator contains:
`bucketIndex`
`nextBucketHash`
`tabletStartHash` *
`tabletEndHash`
`numBuckets`

P.S. I know of at least two more problems. Can you find them?