August 28

KEYSPACE

Amsterdam

What’s new in Valkey 9.0

Madelyn Olson
Valkey Maintainer

Ping Xie
Valkey Maintainer

Ran Shidlansik
Valkey Maintainer

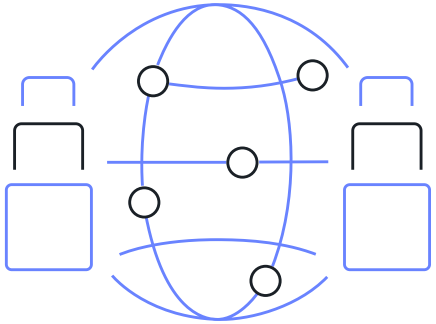
Background The Horsehead Nebula and its surroundings. The reflection nebula NGC 2023 in the bottom left corner. / Stephanh / License: CC BY 4.0

Introducing the Valkey project

Fully compatible with Redis OSS 7.2

THE LINUX FOUNDATION

Vendor Neutral BSD-3 Licensed



Built by contributors in the open source community

A year in Valkey

Valkey 8.0 Release

- 1M Requests per second (RPS)
- Dual channel replication
- Enhanced slot migration reliability

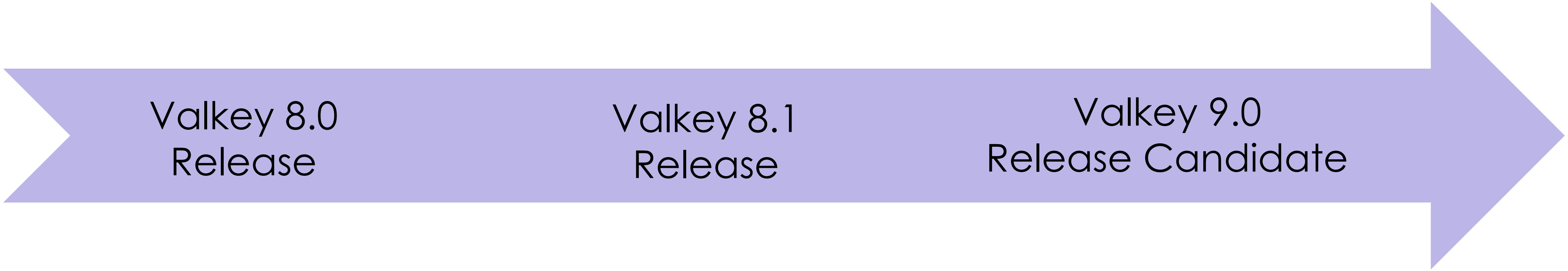
A year in Valkey

Valkey 8.0 Release

Valkey 8.1 Release

- Reduced memory overhead by 20%
- Vector similarity, Bloom, and JSON modules
- New command log and metrics

A year in Valkey



Valkey 8.0
Release

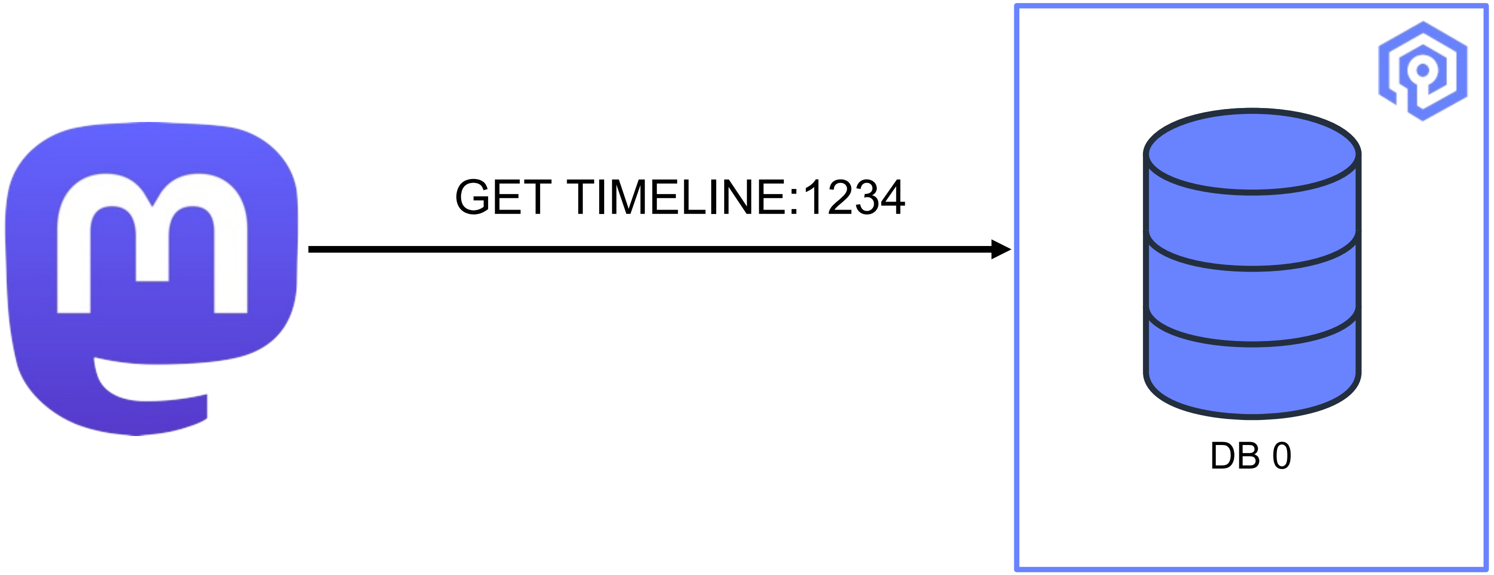
Valkey 8.1
Release

Valkey 9.0
Release Candidate

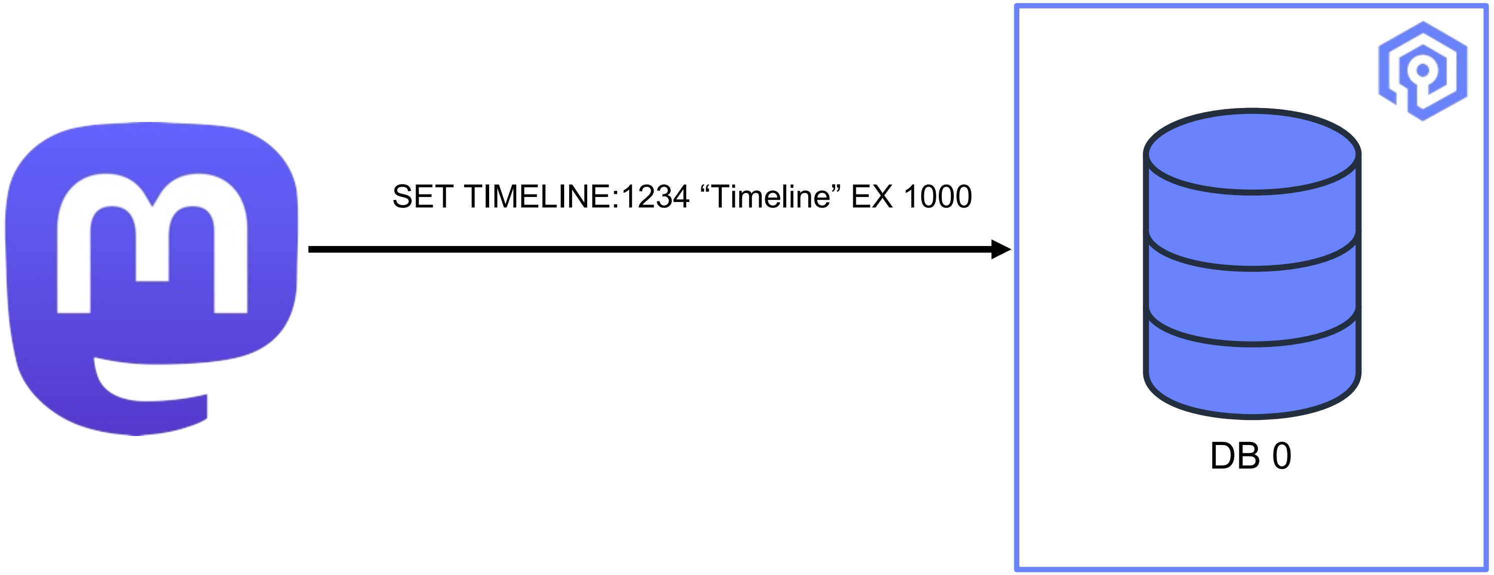
- Multiple databases in cluster mode
- Atomic slot migration
- Expiration on hash field items

Multiple-databases in Cluster mode

Simple Valkey use case



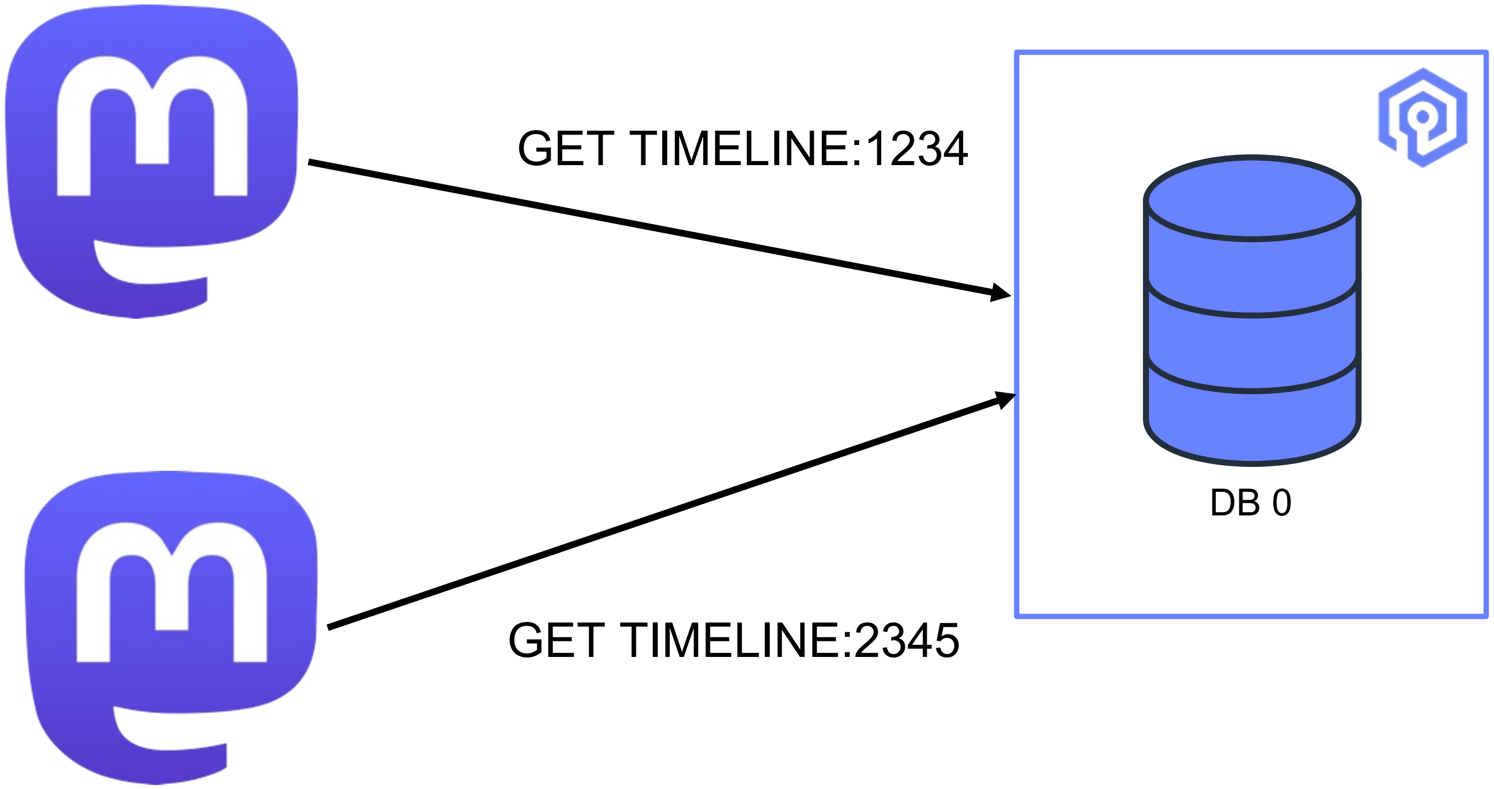
Simple Valkey use case



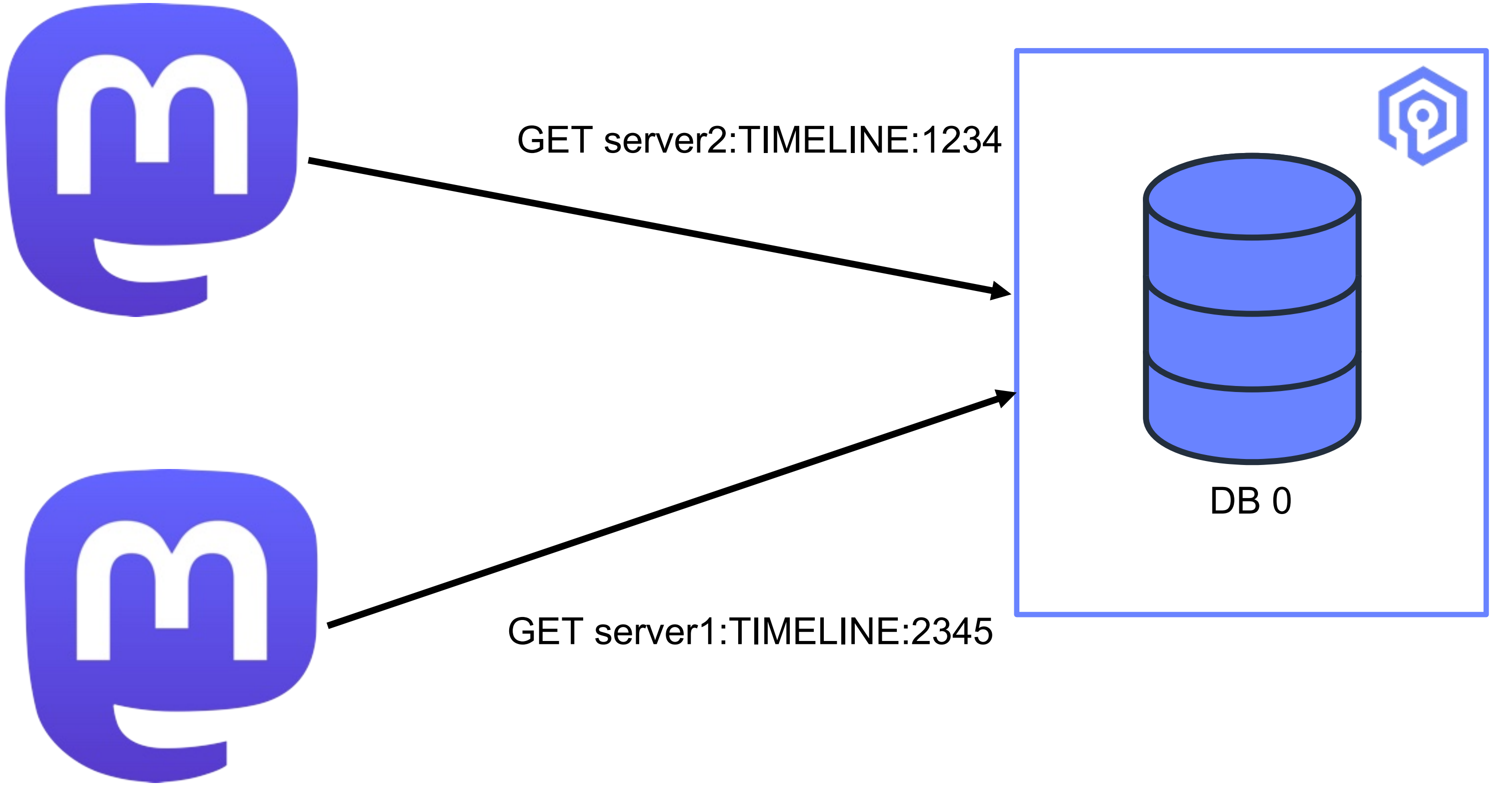
SET TIMELINE:1234 “Timeline” EX 1000

DB 0

Using Valkey Databases as namespaces



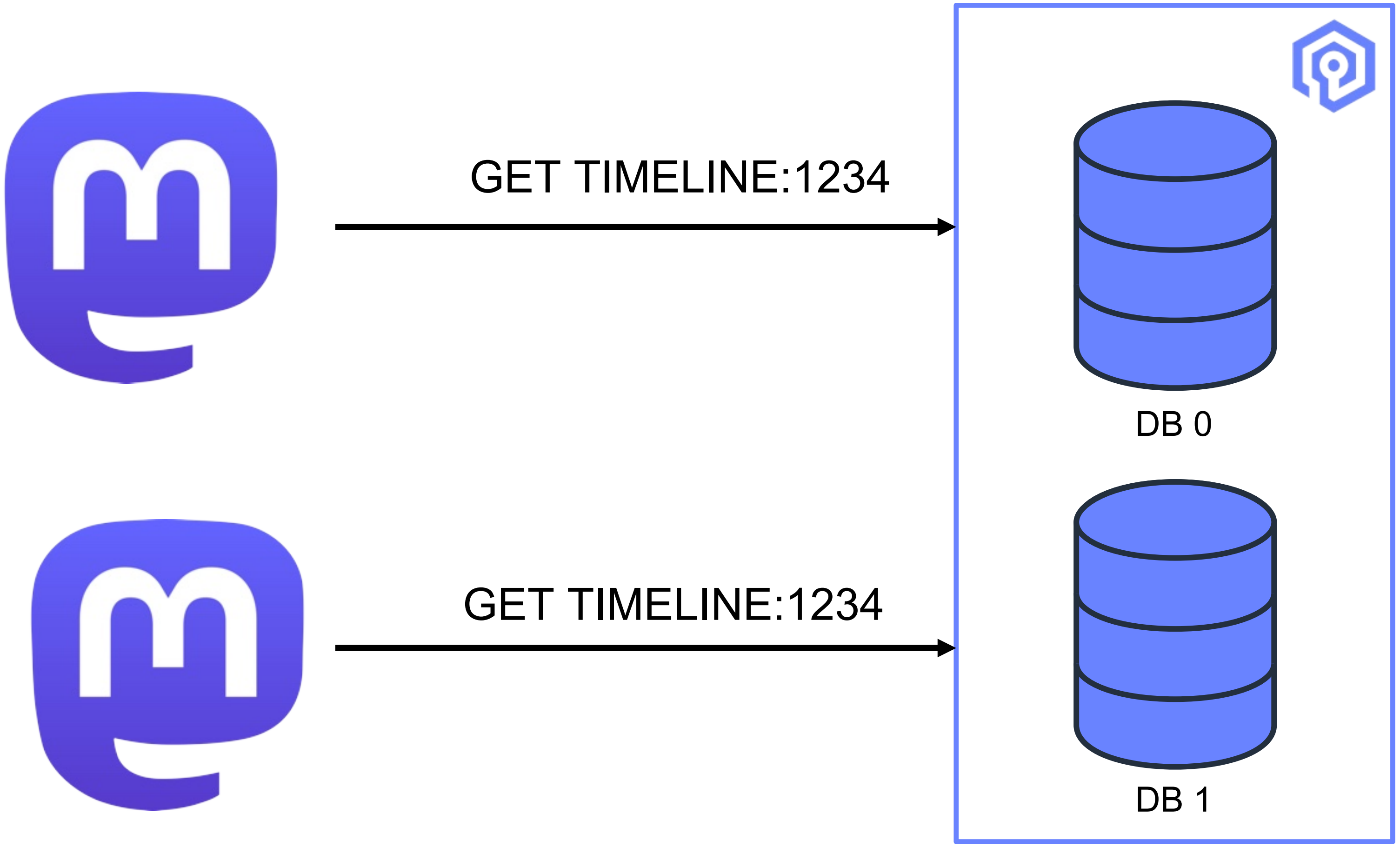
Using Valkey Databases as namespaces



GET server2:TIMELINE:1234

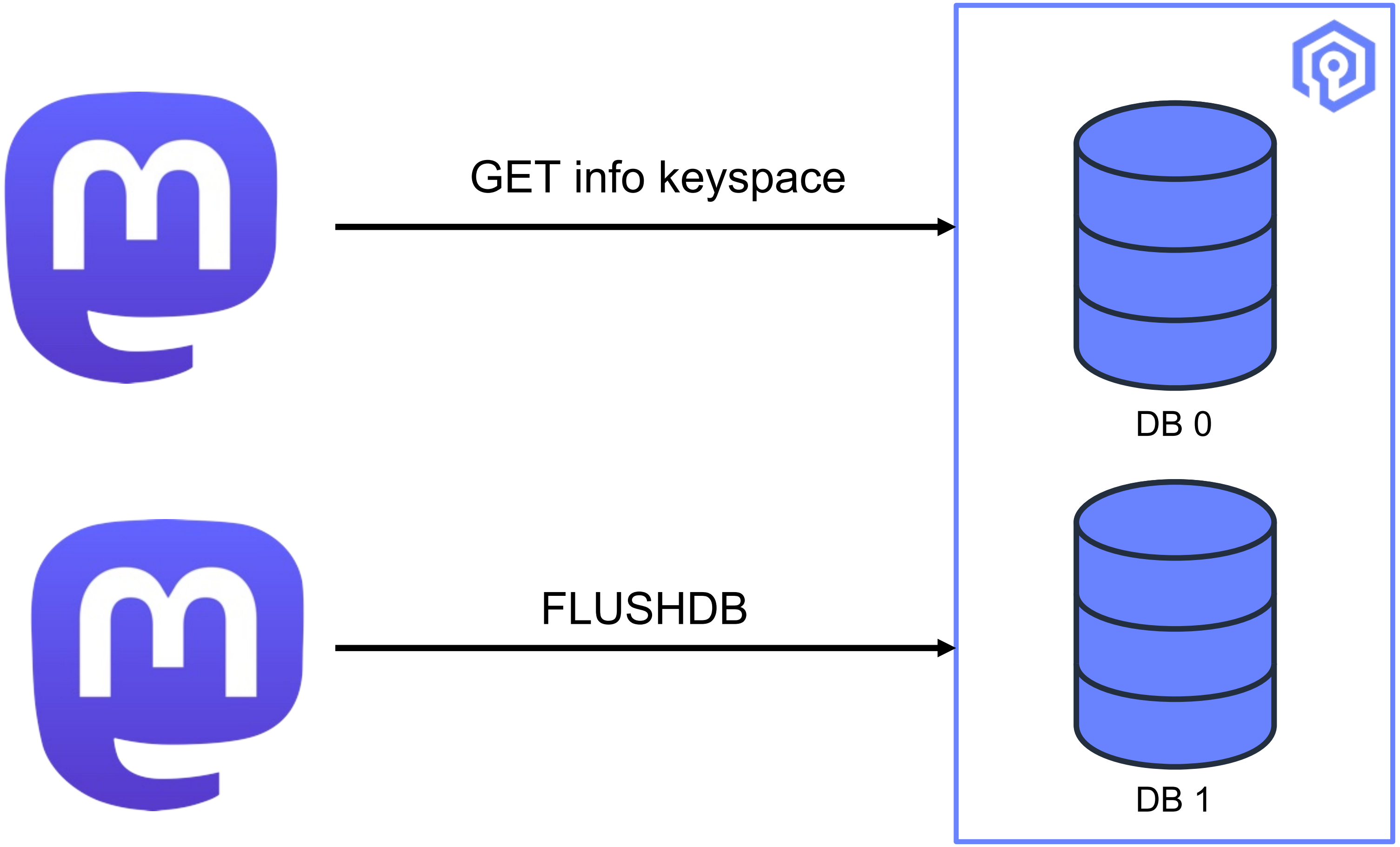
GET server1:TIMELINE:2345

Using Valkey Databases as namespaces



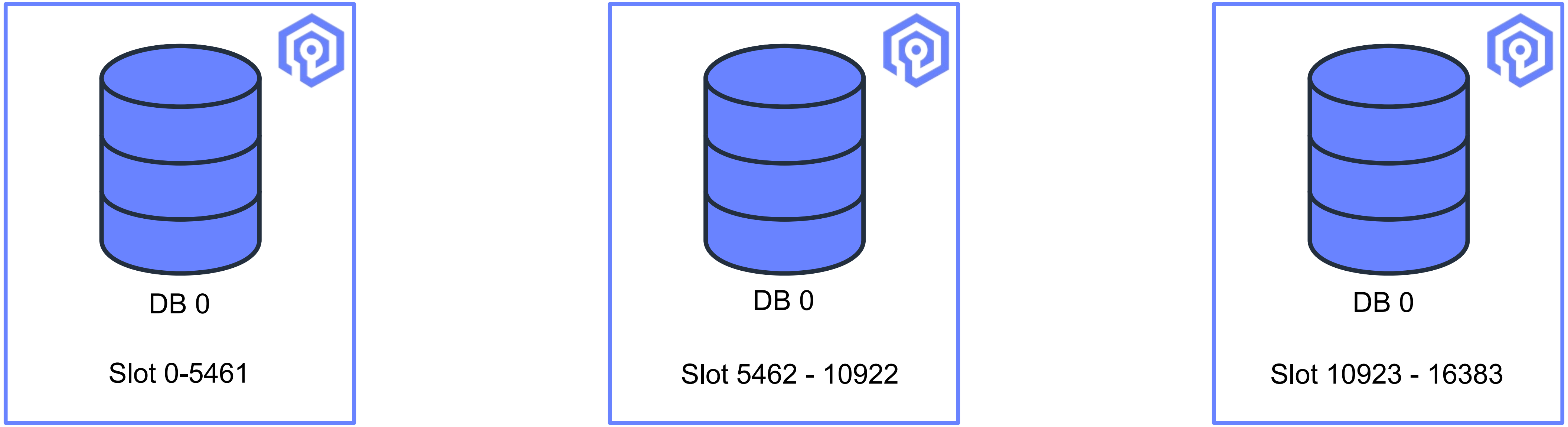
Valkey

Using Valkey Databases as namespaces



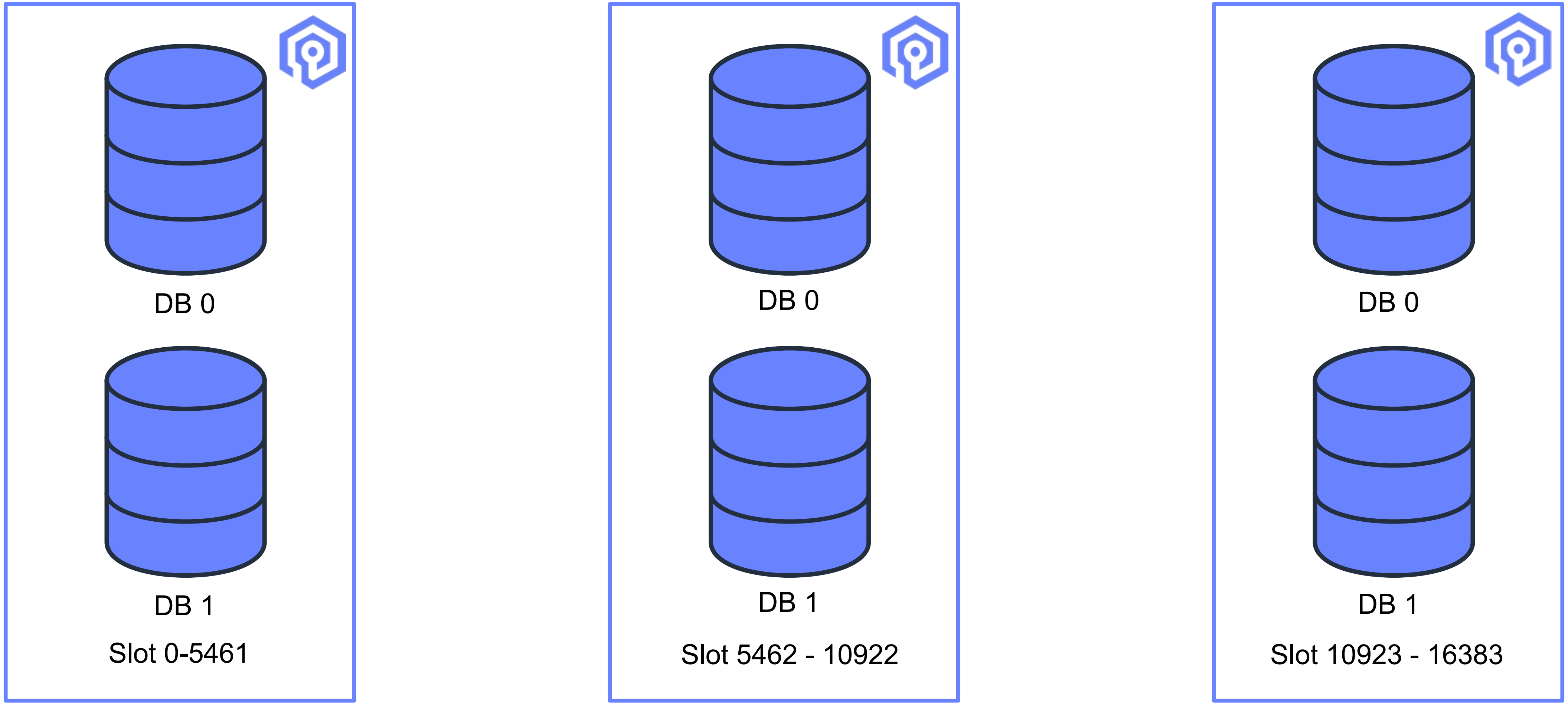
Valkey

Extending databases to cluster mode



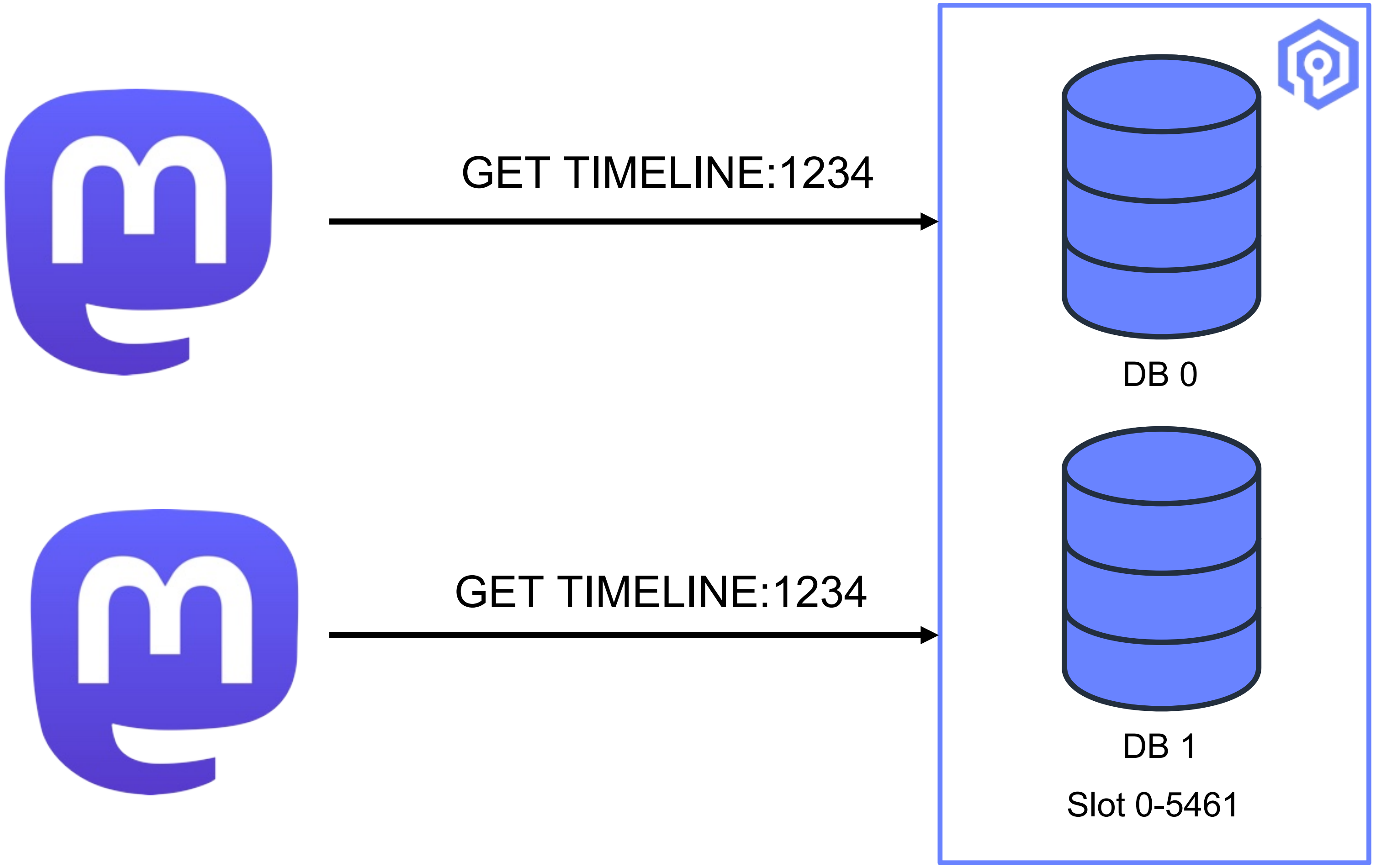
Valkey

Extending databases to cluster mode



Valkey

Extending databases to cluster mode

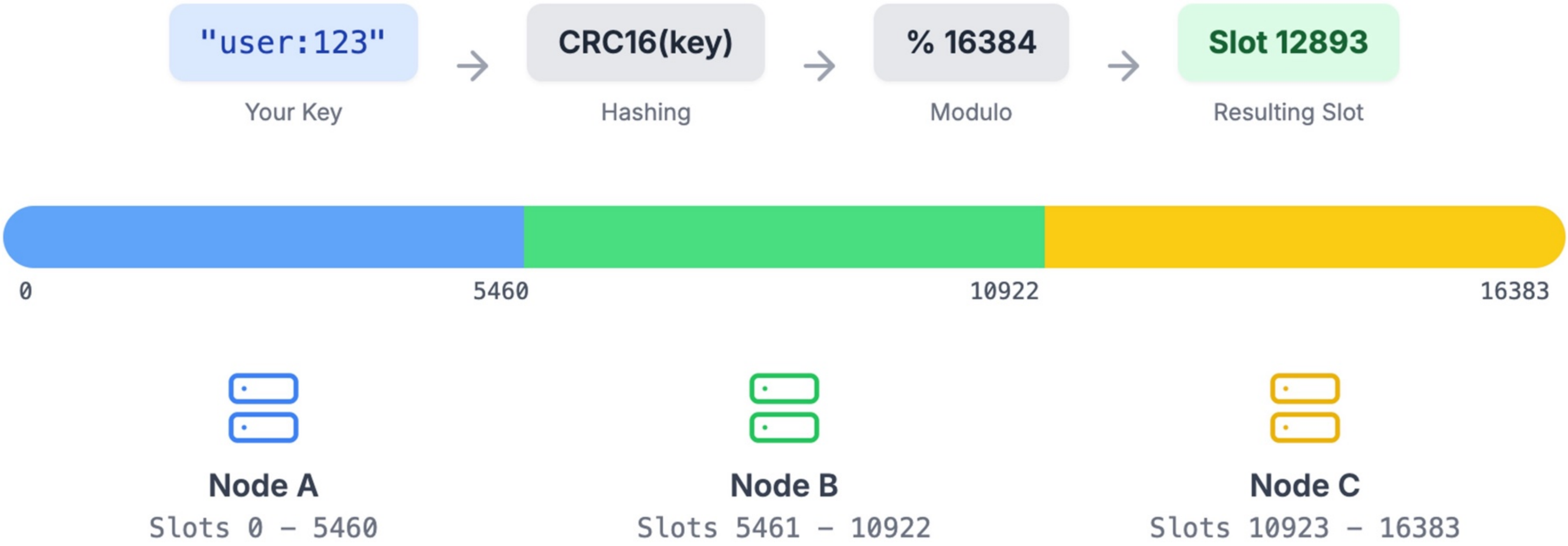


Summary of clustered databases

- Provides namespaces that can scale horizontally
- Zero-overhead when unused
- More database features coming soon!

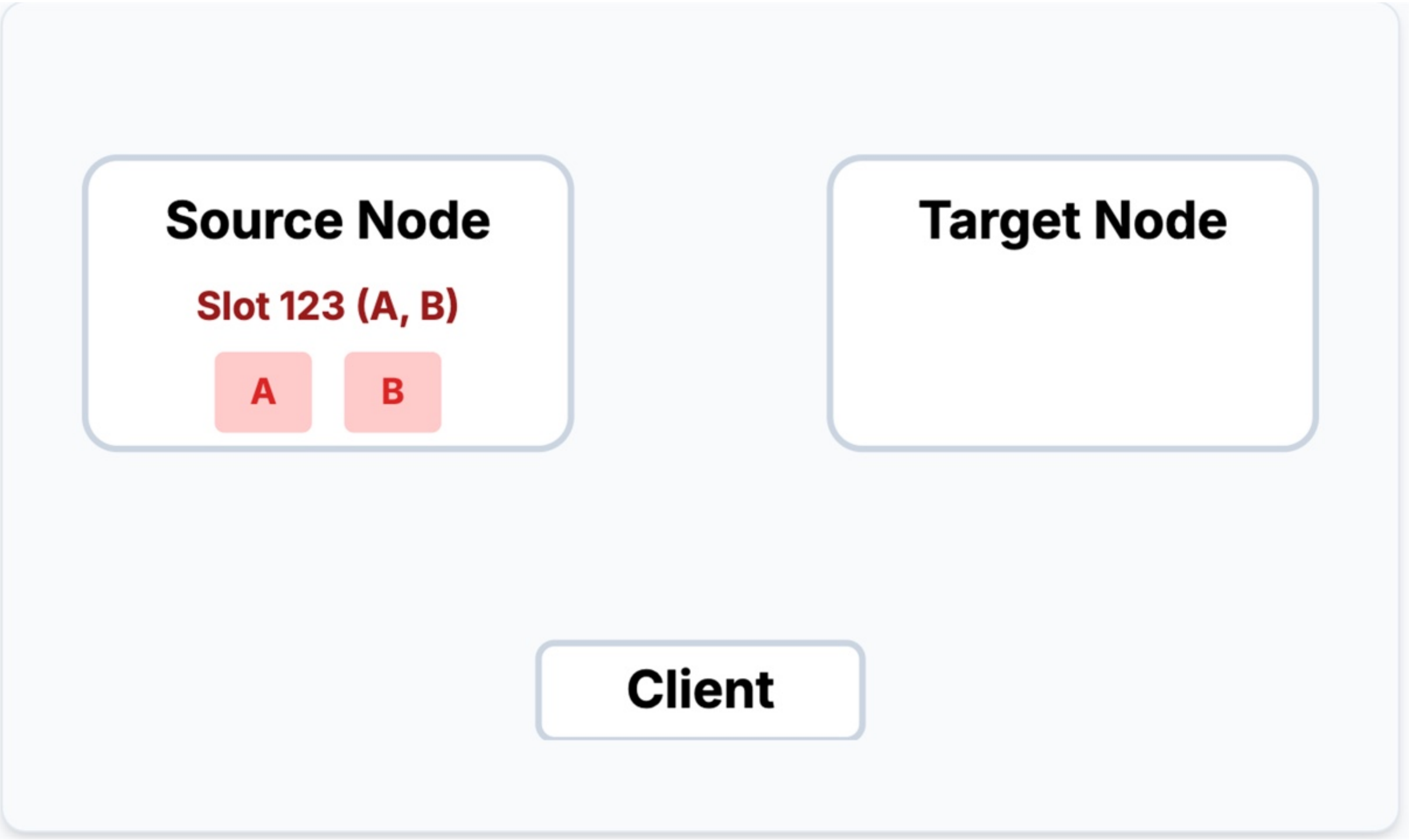
Atomic Slot Migration

How Valkey Cluster Works: The Slots

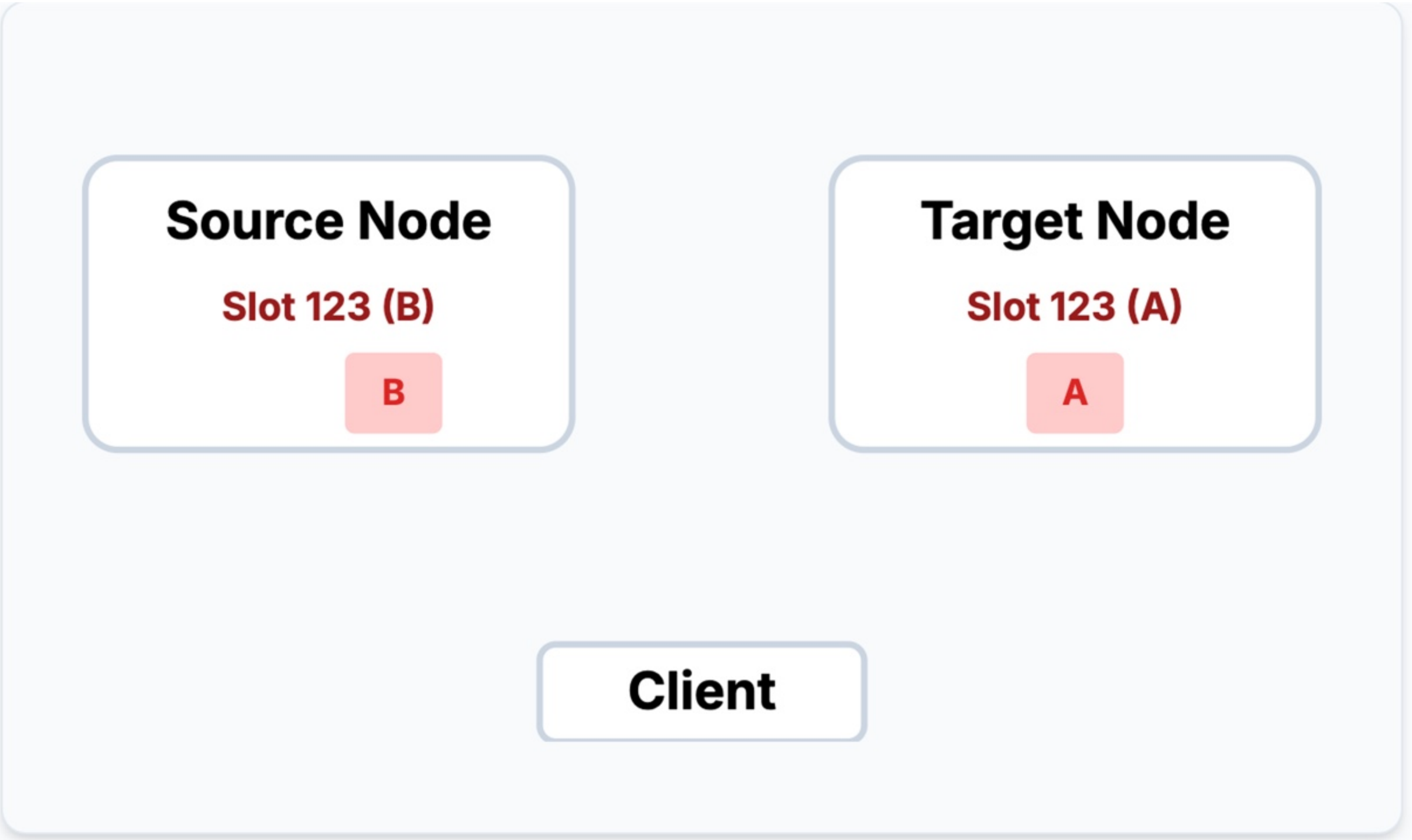


Valkey

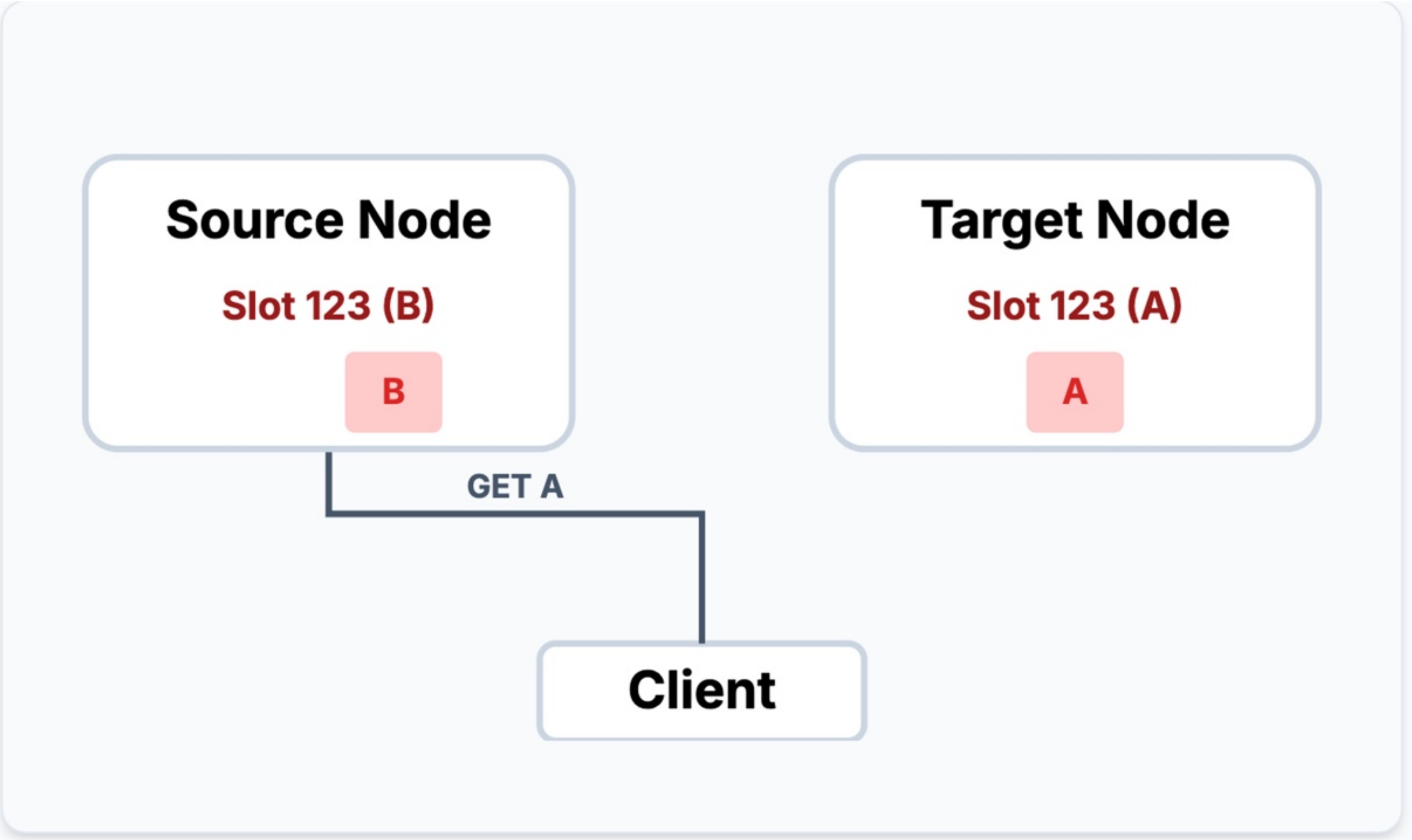
Problem 1 - Suboptimal Client Redirects



Problem 1 - Suboptimal Client Redirects

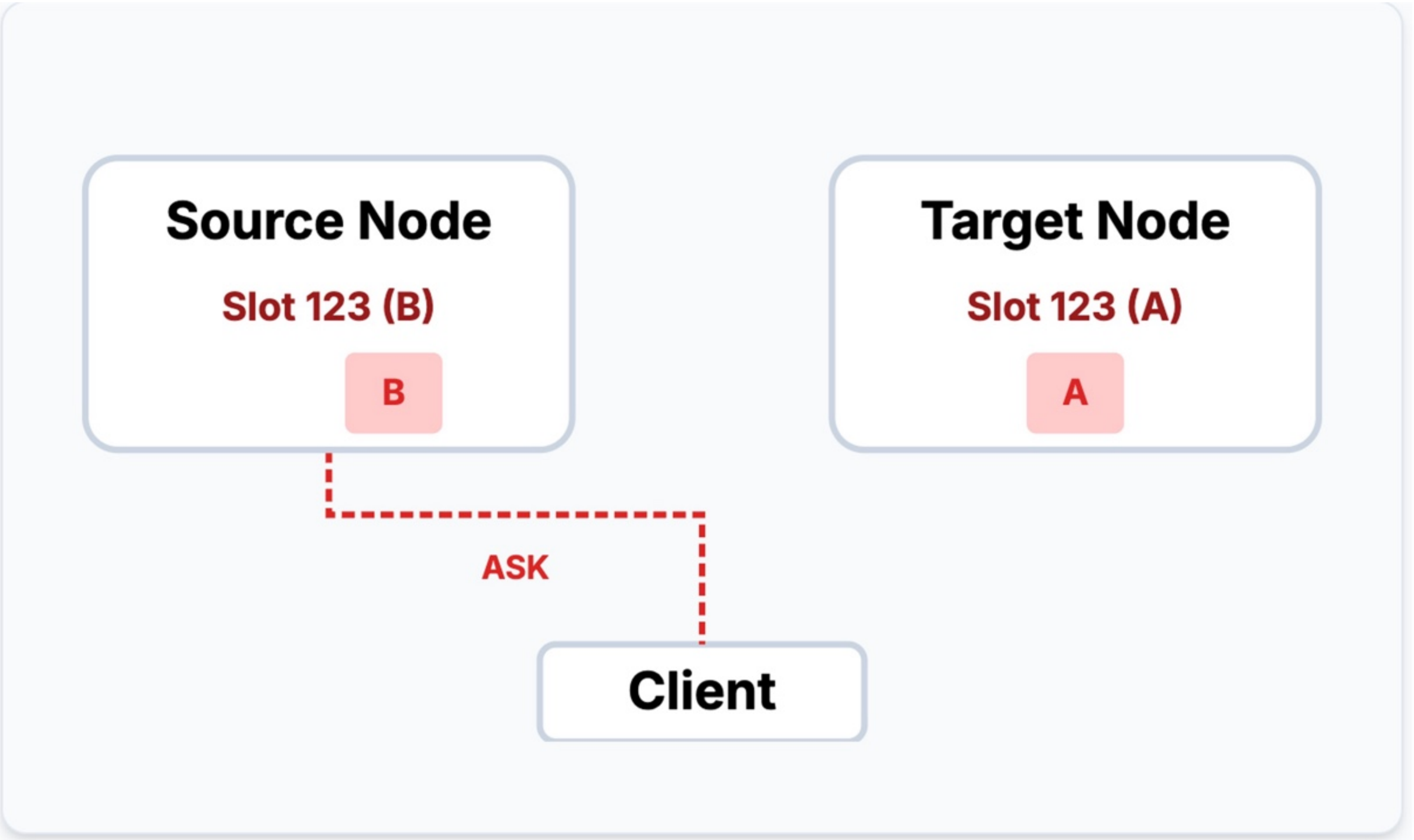


Problem 1 - Suboptimal Client Redirects



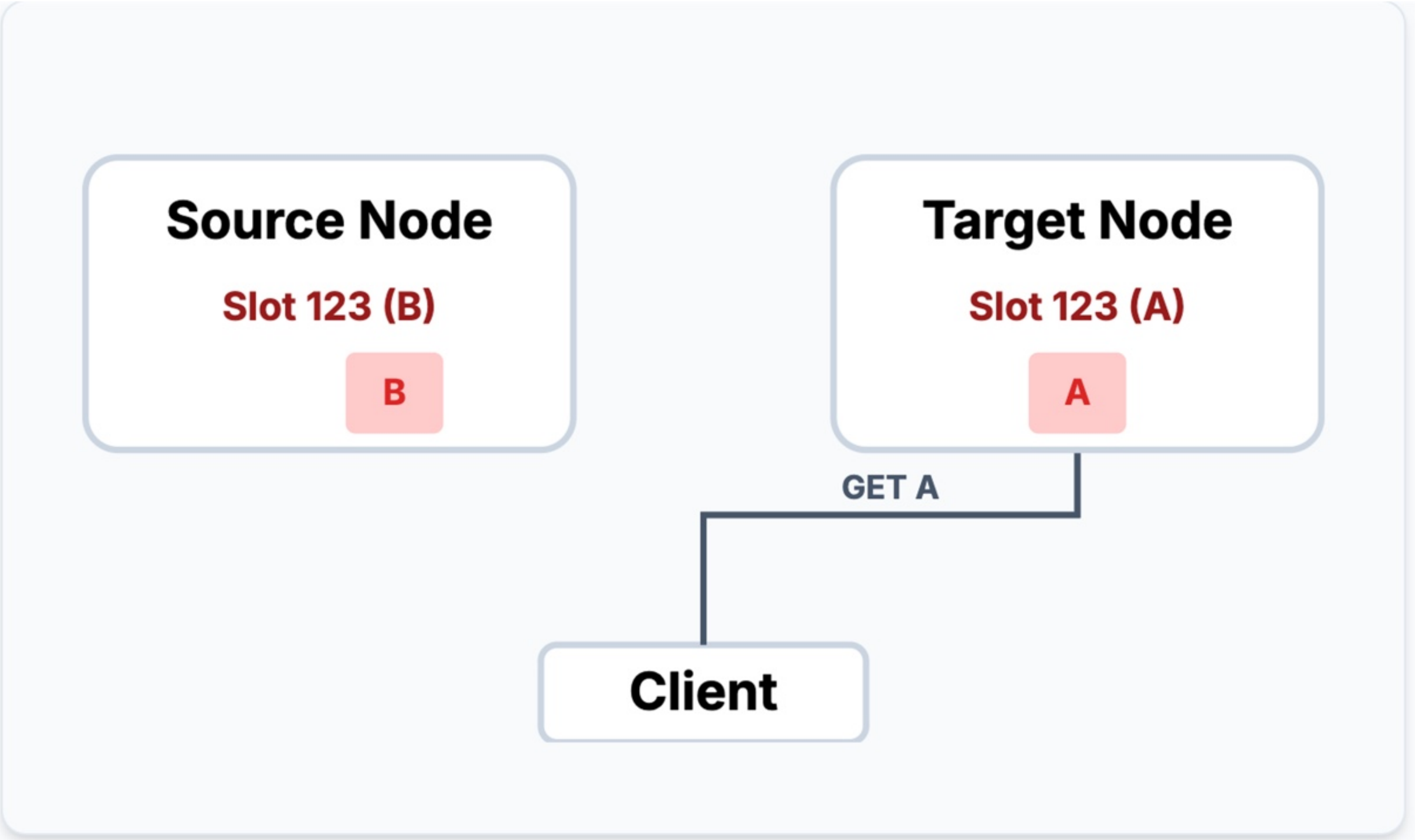
Valkey

Problem 1 - Suboptimal Client Redirects



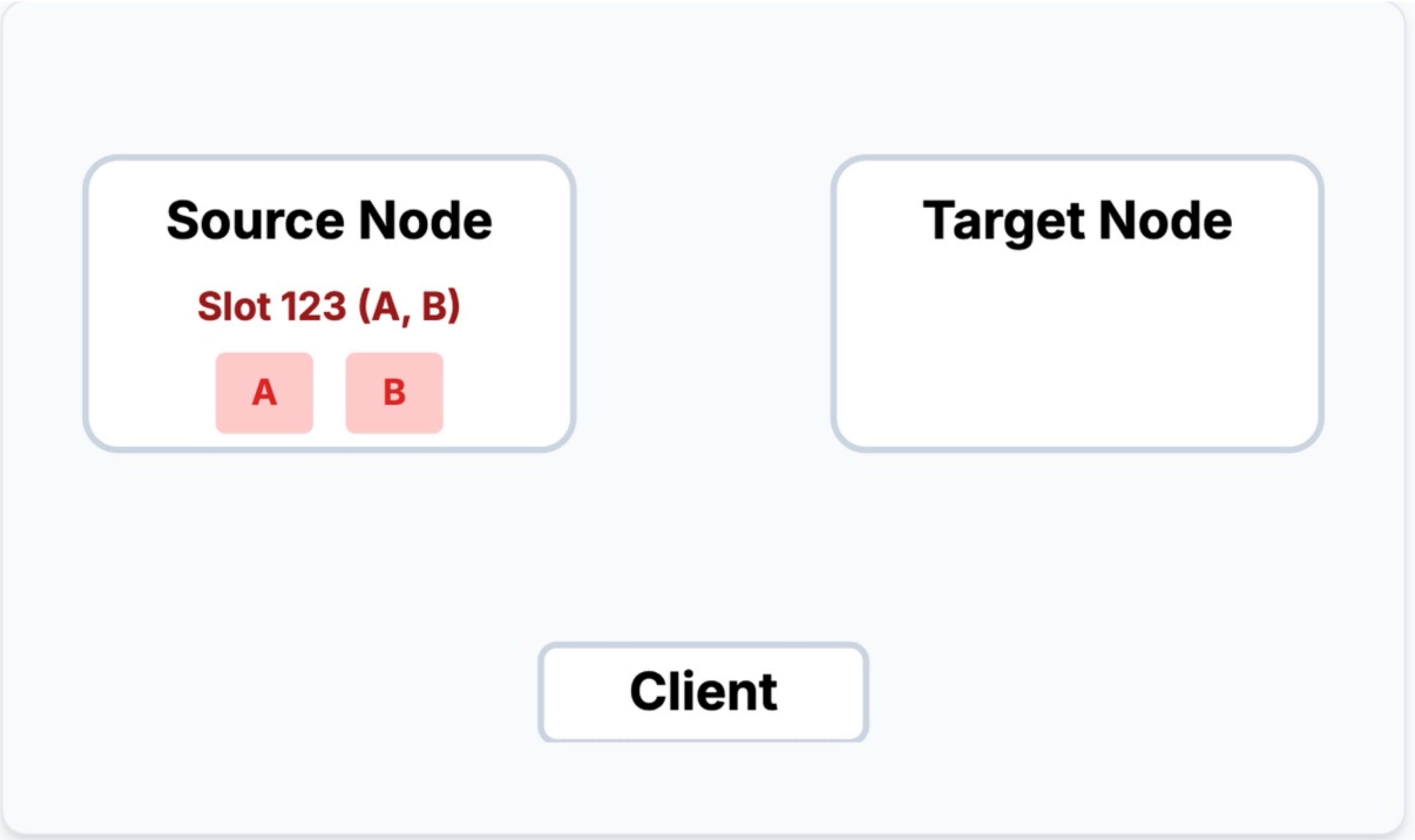
Valkey

Problem 1 - Suboptimal Client Redirects

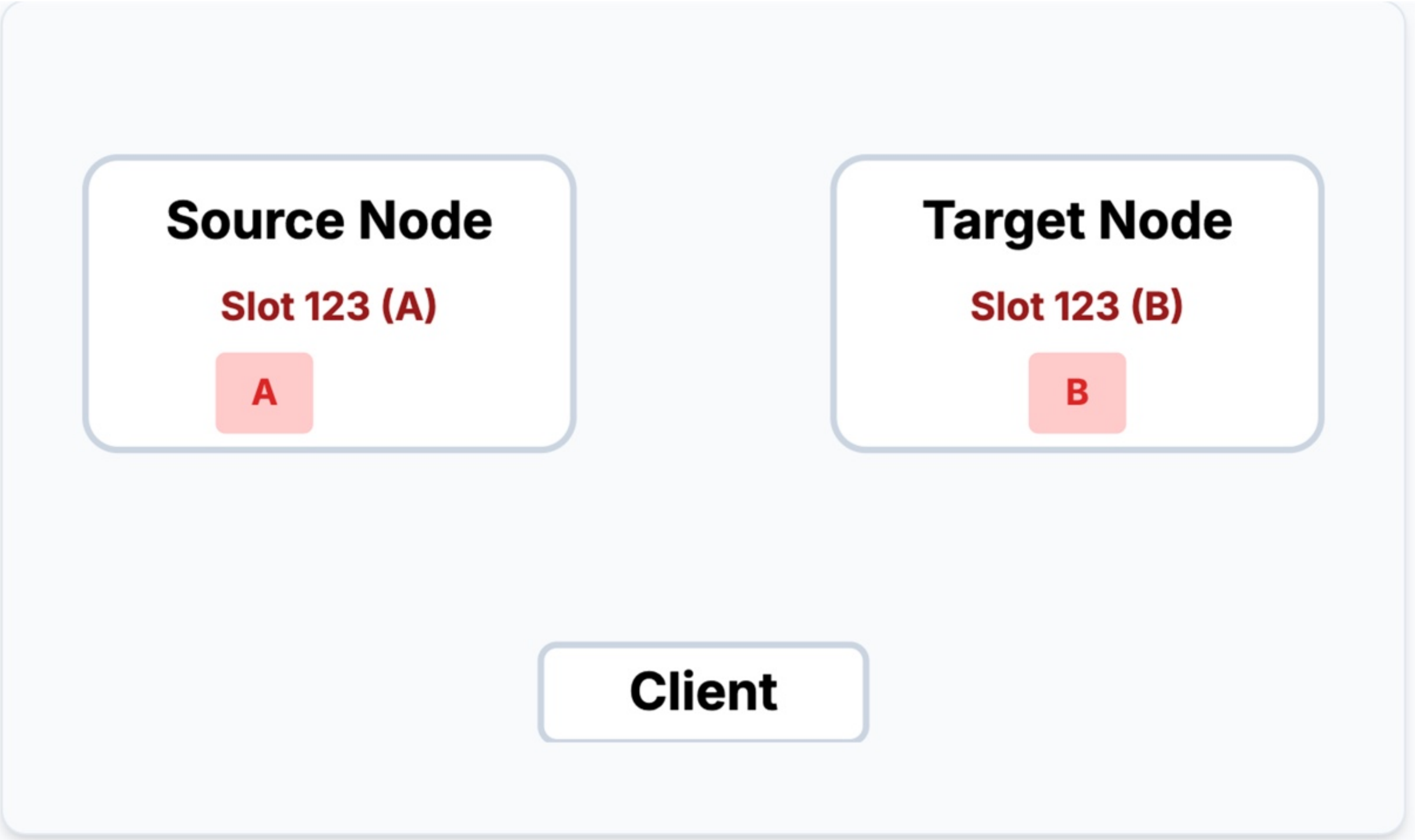


Valkey

Problem 2 - Broken Multi-Key Operations

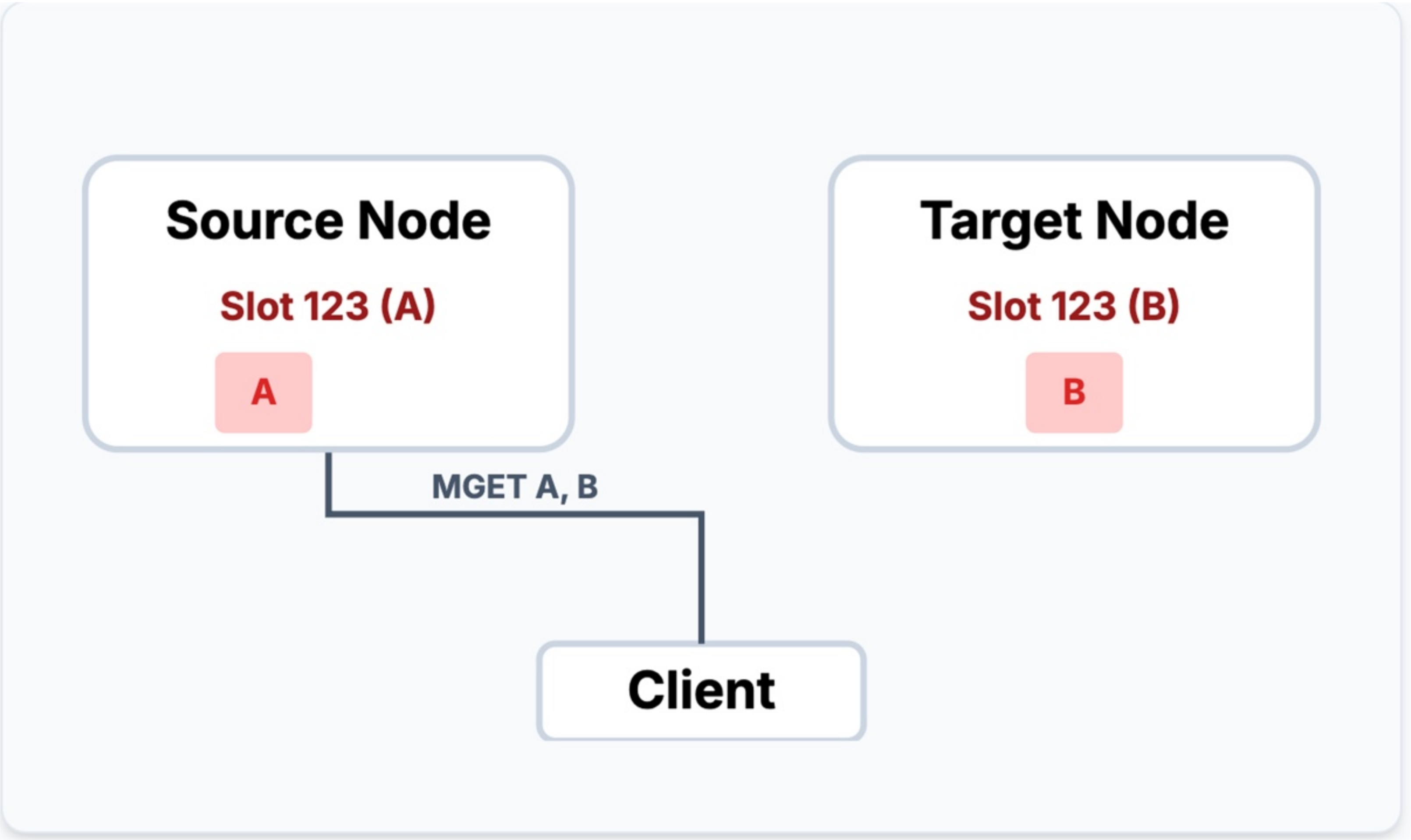


Problem 2 - Broken Multi-Key Operations



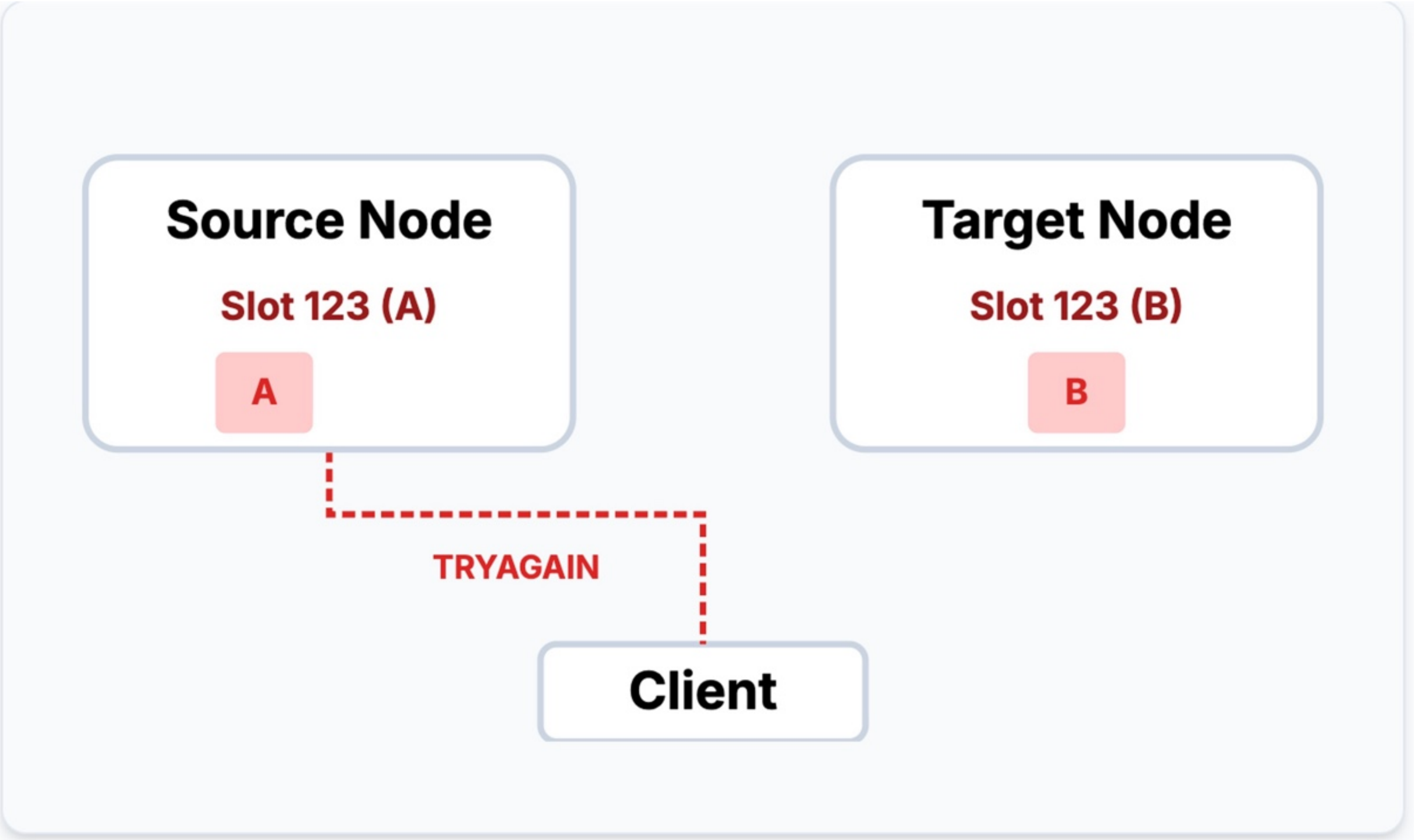
Valkey

Problem 2 - Broken Multi-Key Operations

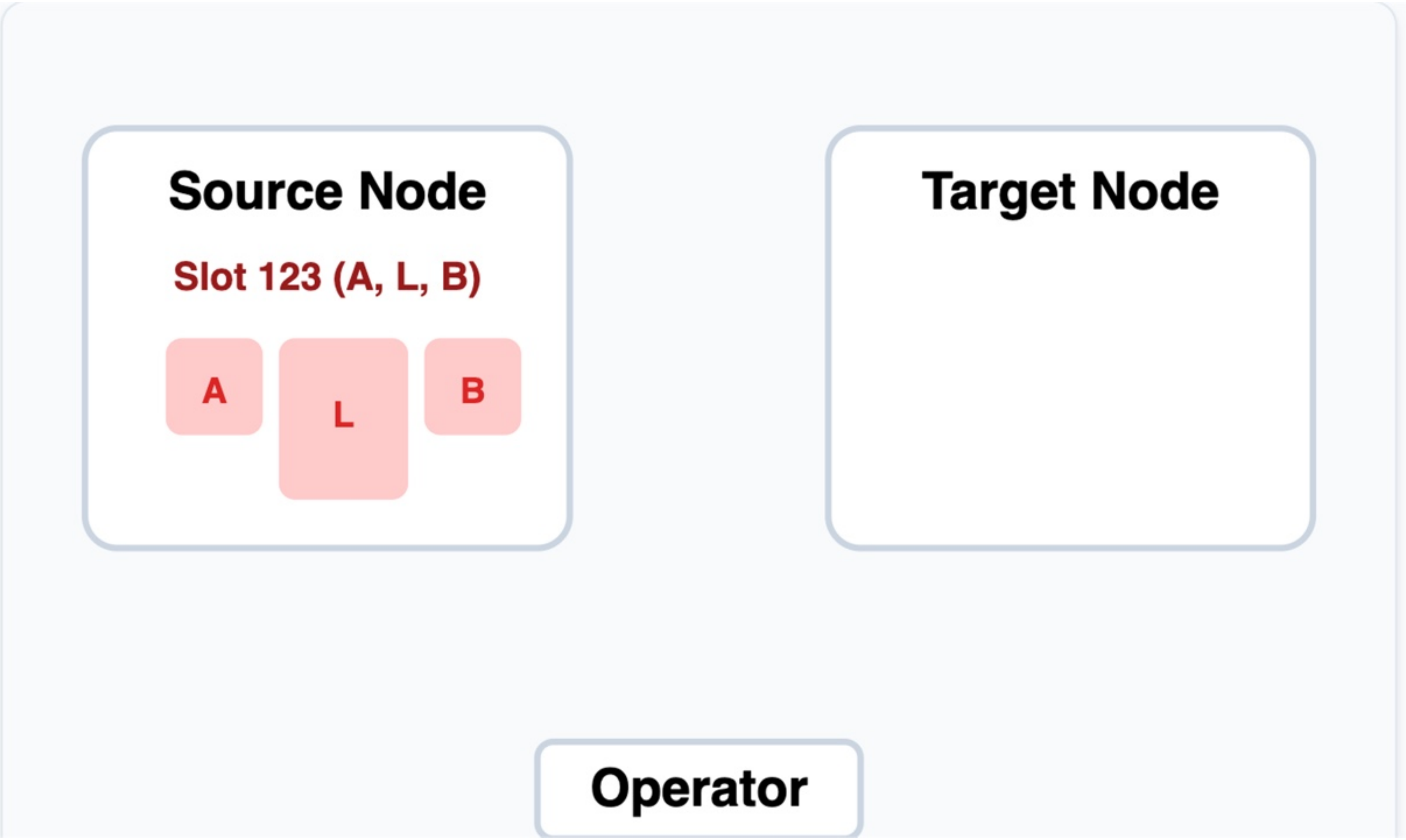


Valkey

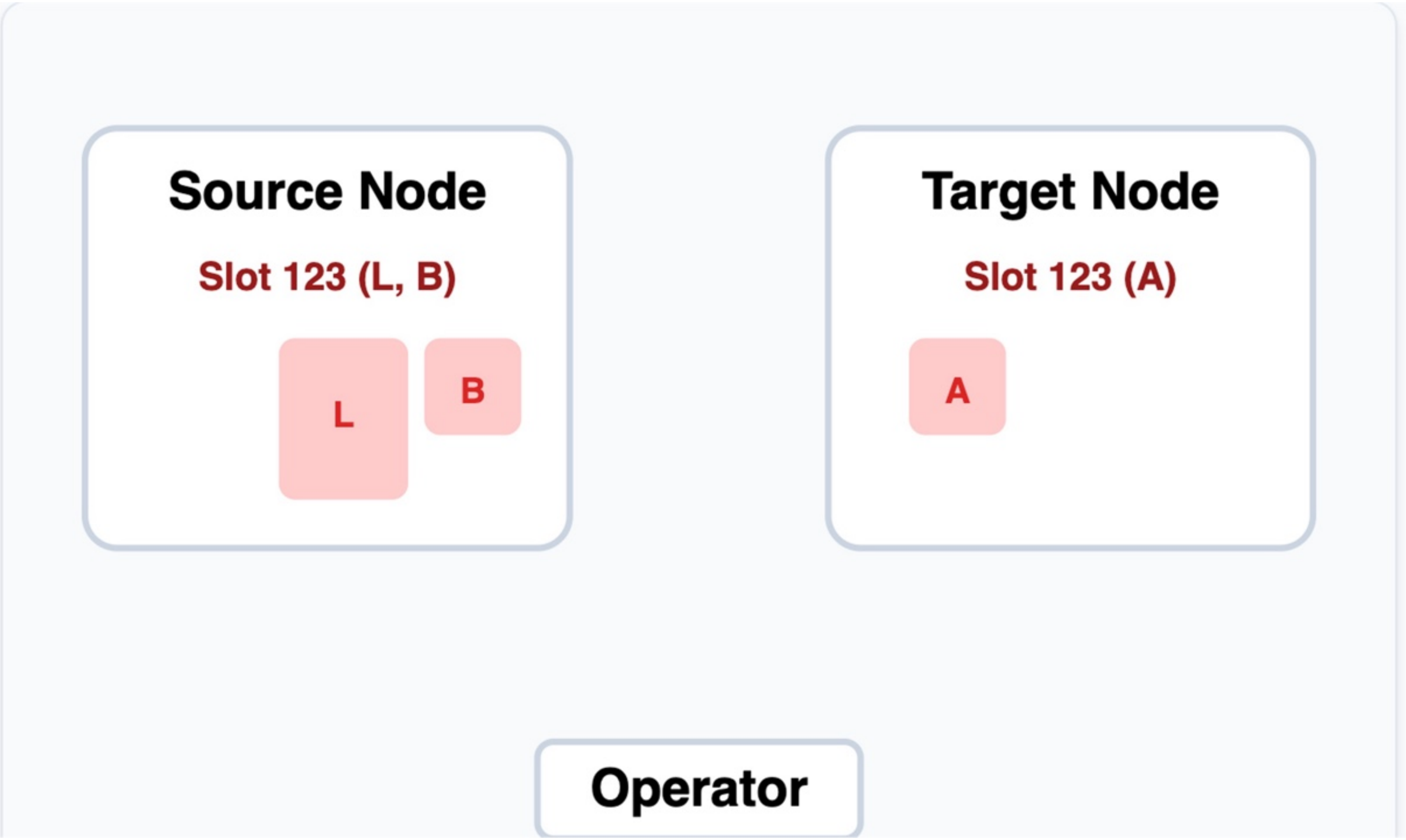
Problem 2 - Broken Multi-Key Operations



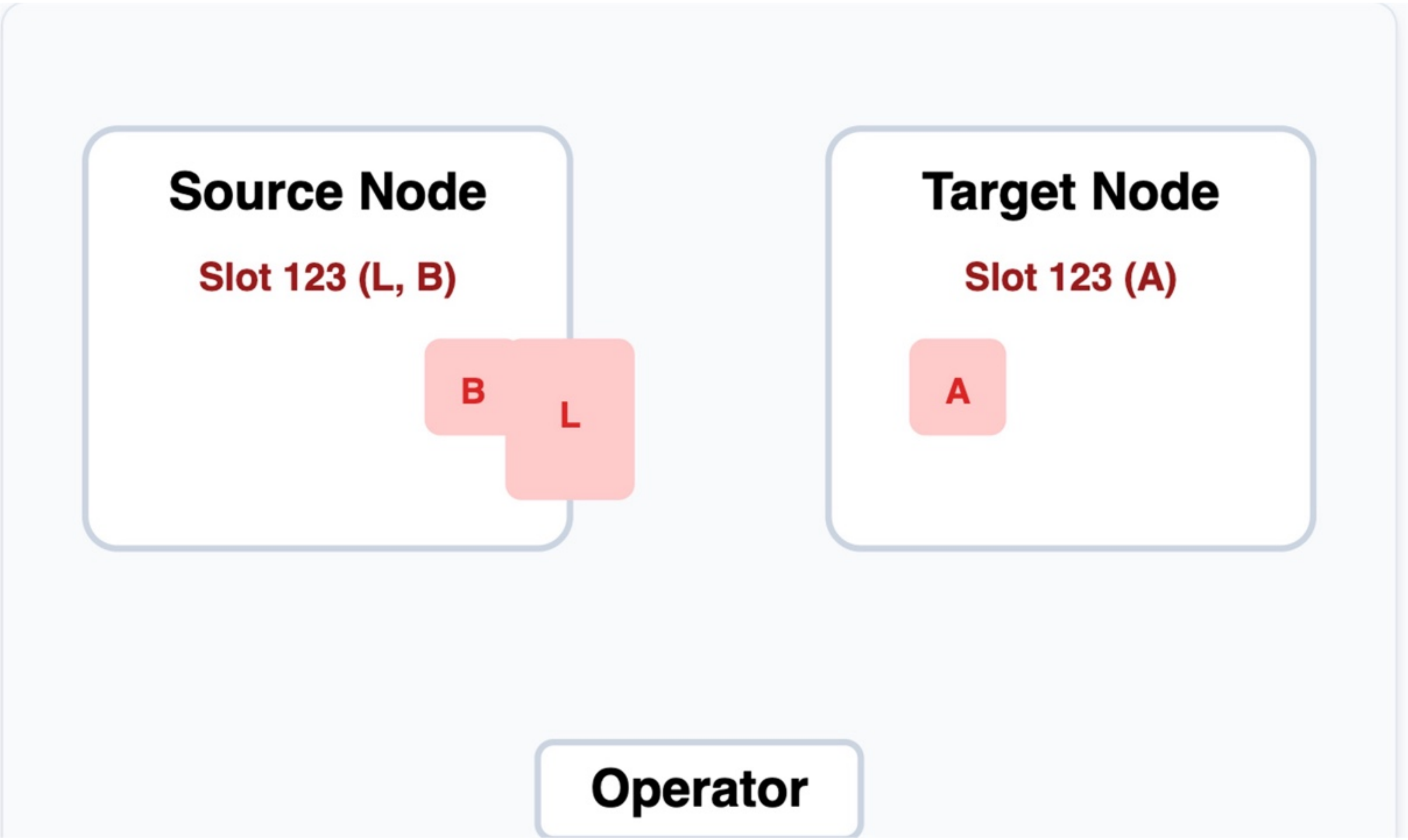
Problem 3 - Large Key Handling



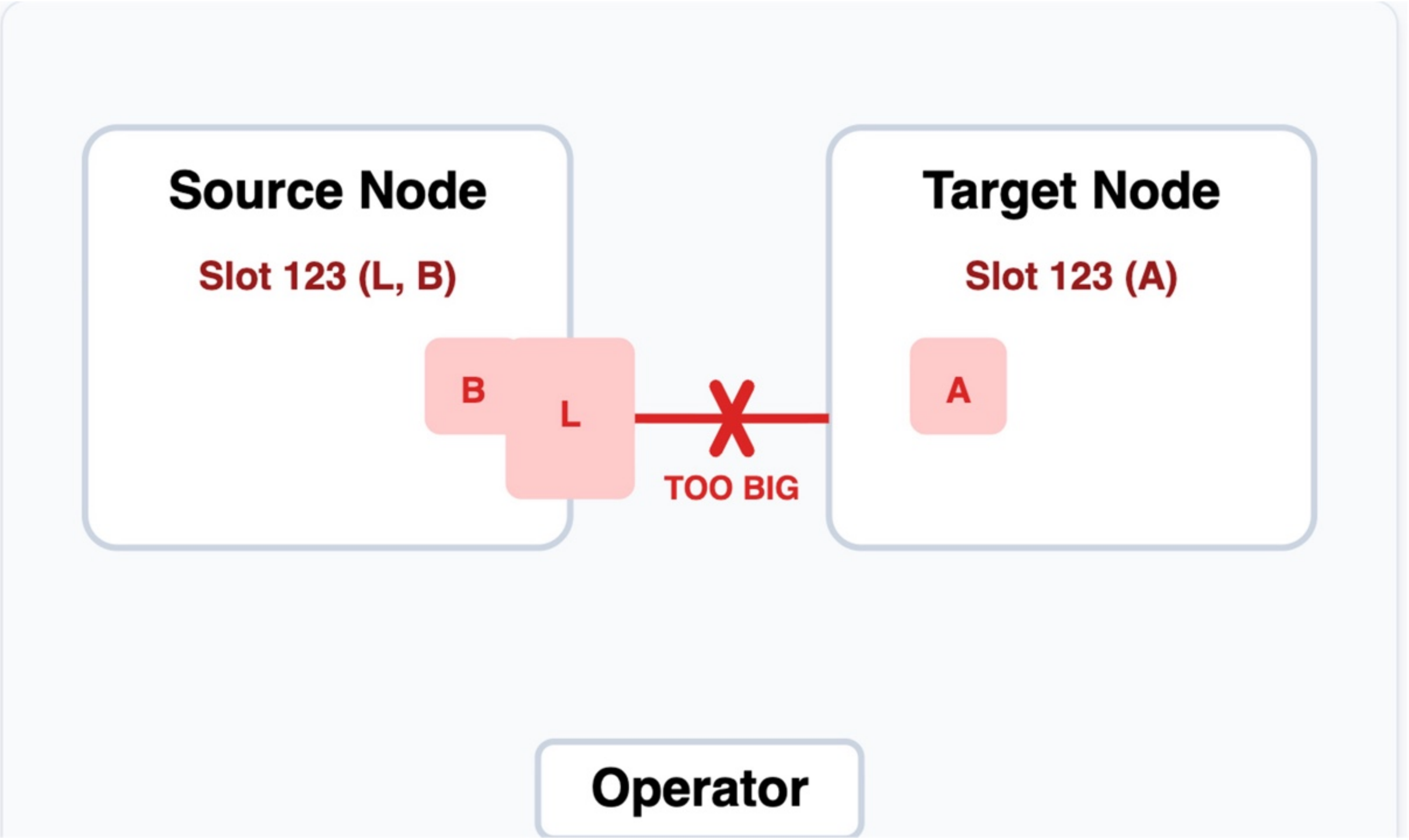
Problem 3 - Large Key Handling



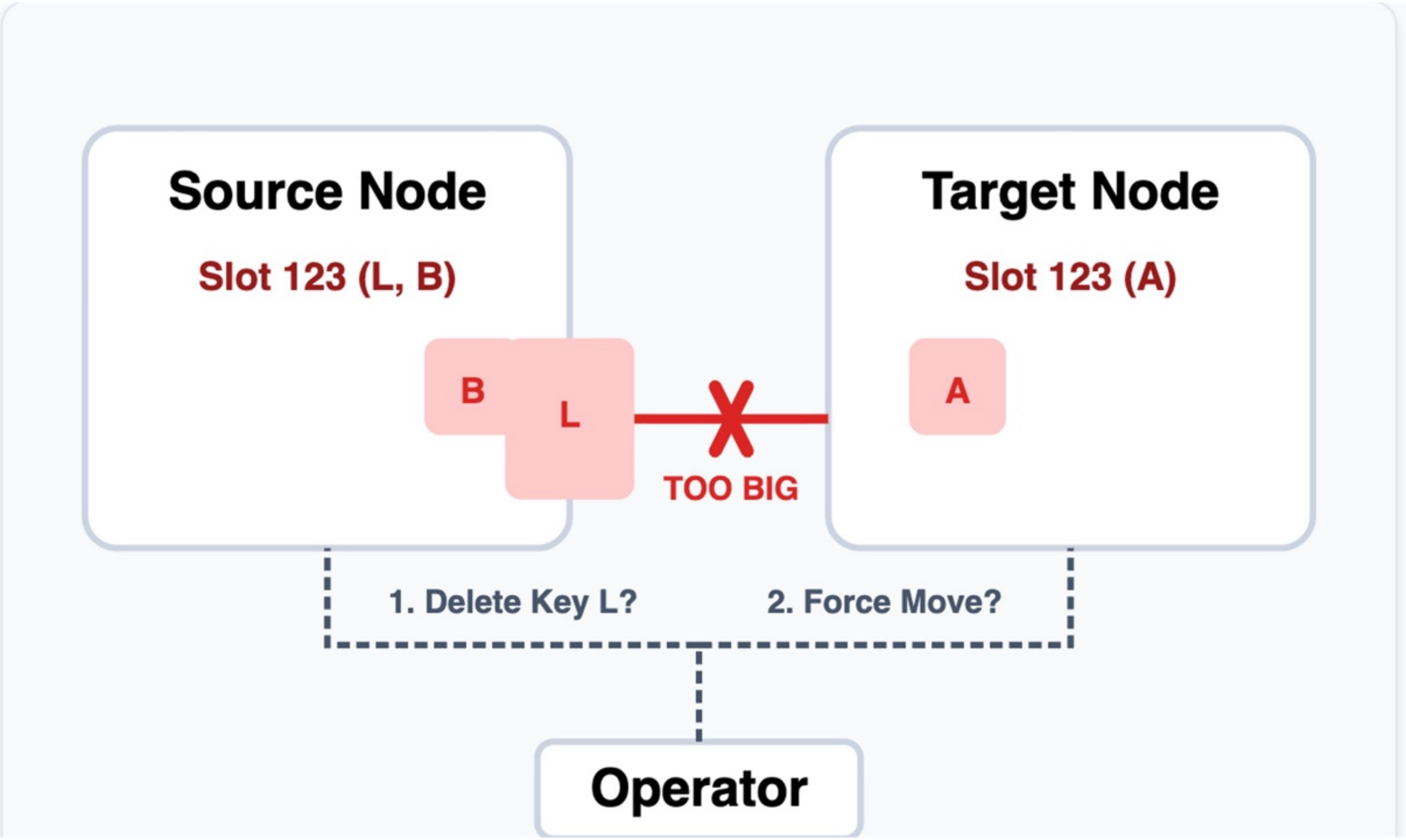
Problem 3 - Large Key Handling



Problem 3 - Large Key Handling

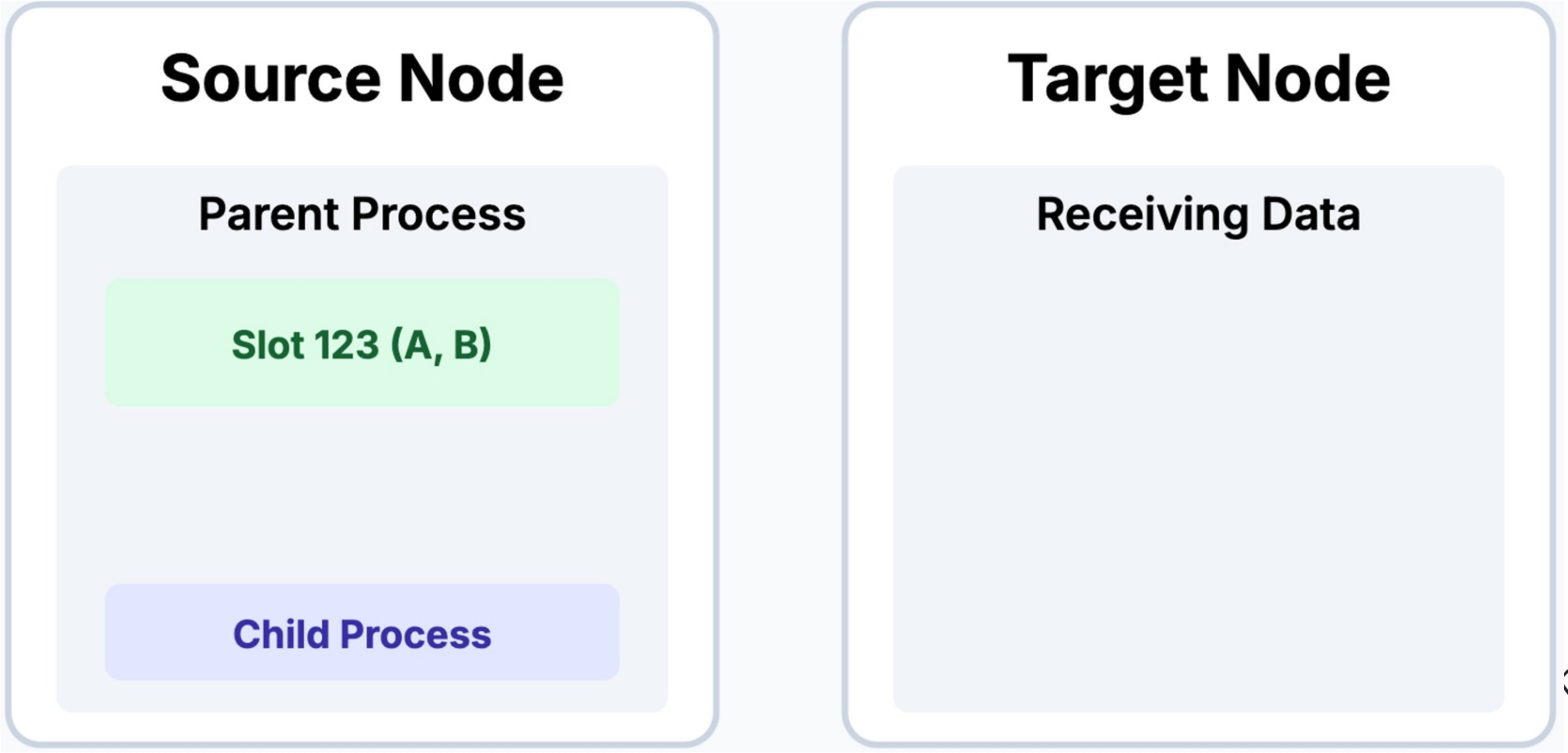


Problem 3 - Large Key Handling

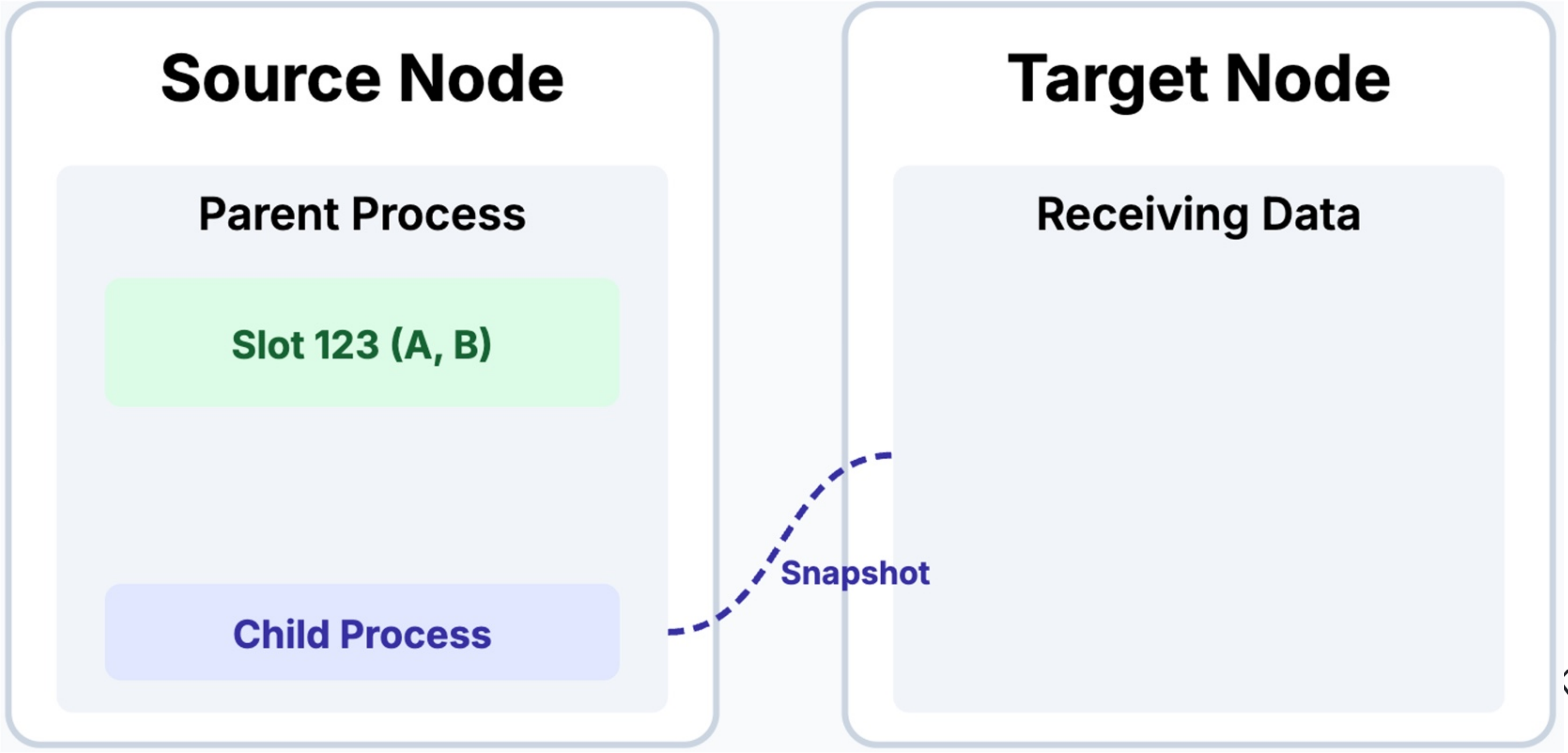


Valkey

1. A child process is forked to create a snapshot



2. The snapshot begins exporting to the target node



Source Node

Parent Process

Slot 123 (A, B)

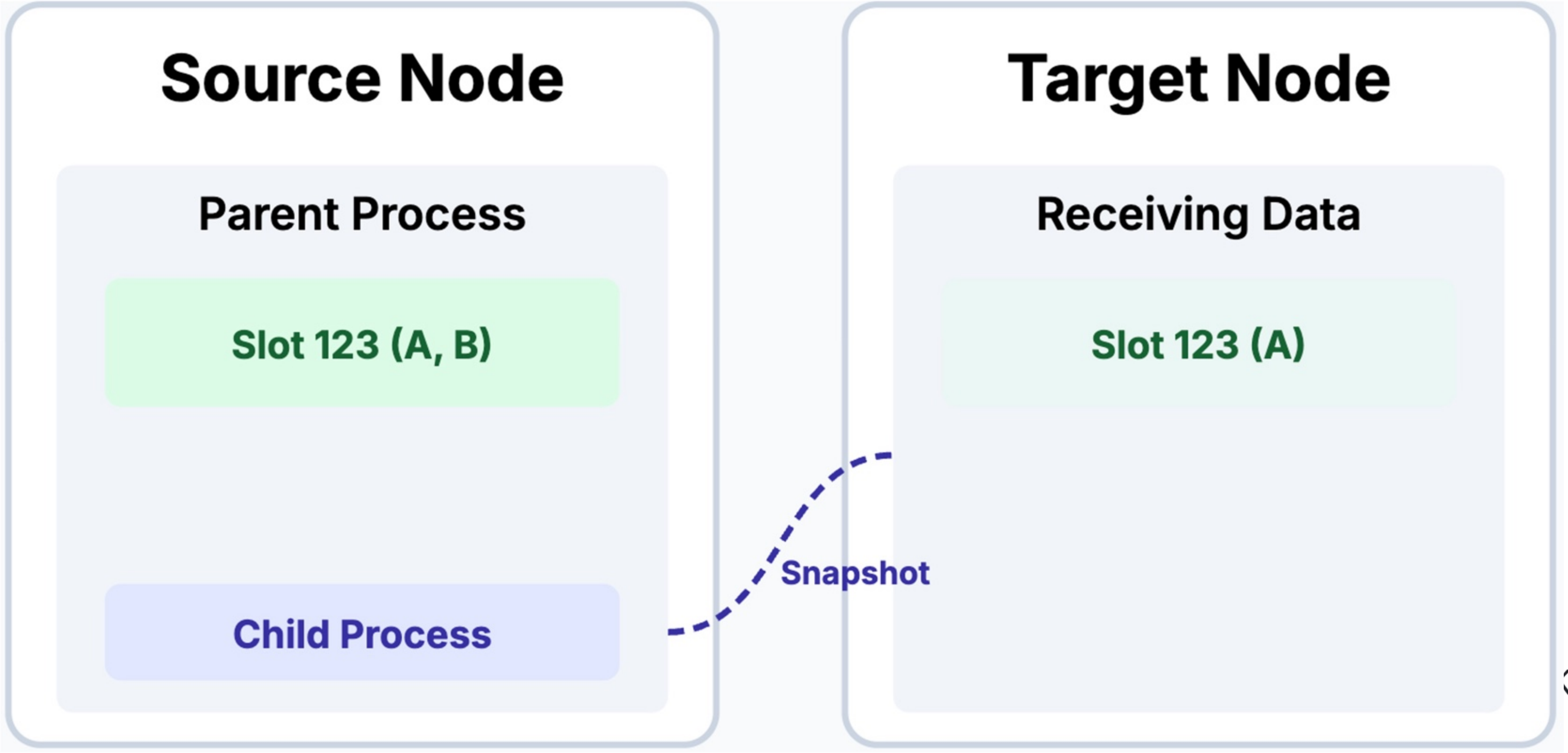
Child Process

Snapshot

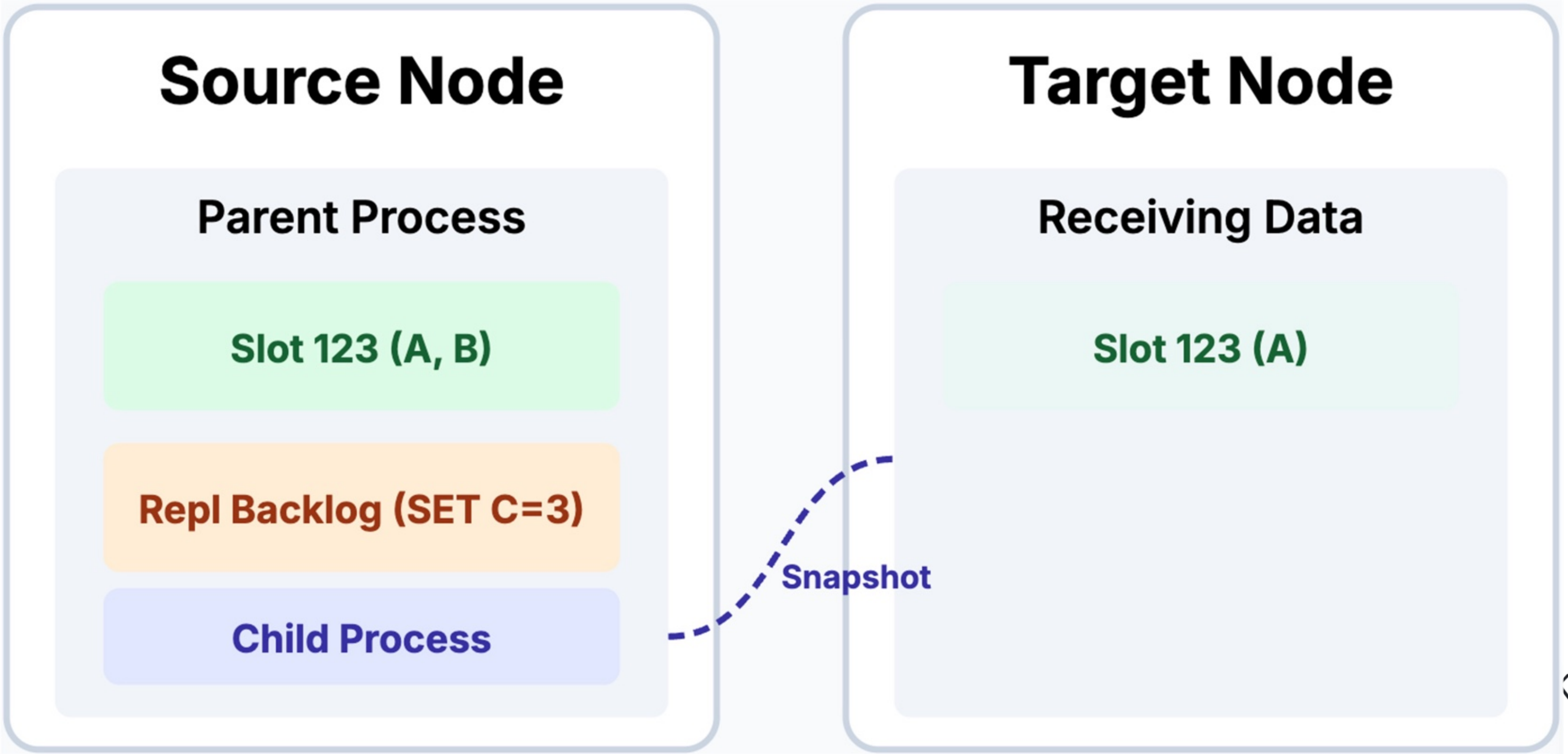
Target Node

Receiving Data

3. The target node receives the first part of the snapshot



4. A new write arrives and is buffered by parent process



Source Node

Parent Process

Slot 123 (A, B)

Repl Backlog (SET C=3)

Child Process

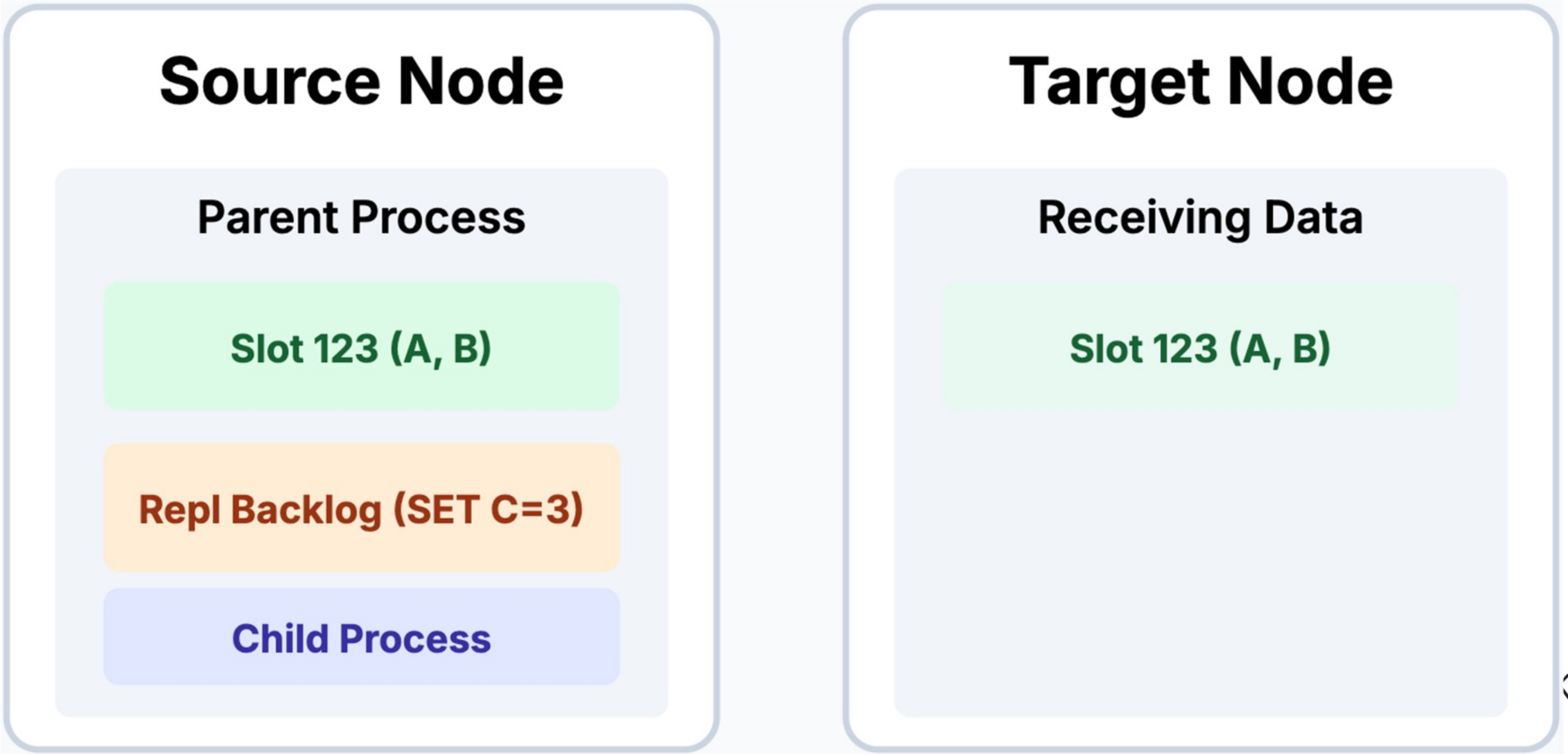
Snapshot

Target Node

Receiving Data

Slot 123 (A)

5. The target node receives the original full snapshot



Source Node

Parent Process

Slot 123 (A, B)

Repl Backlog (SET C=3)

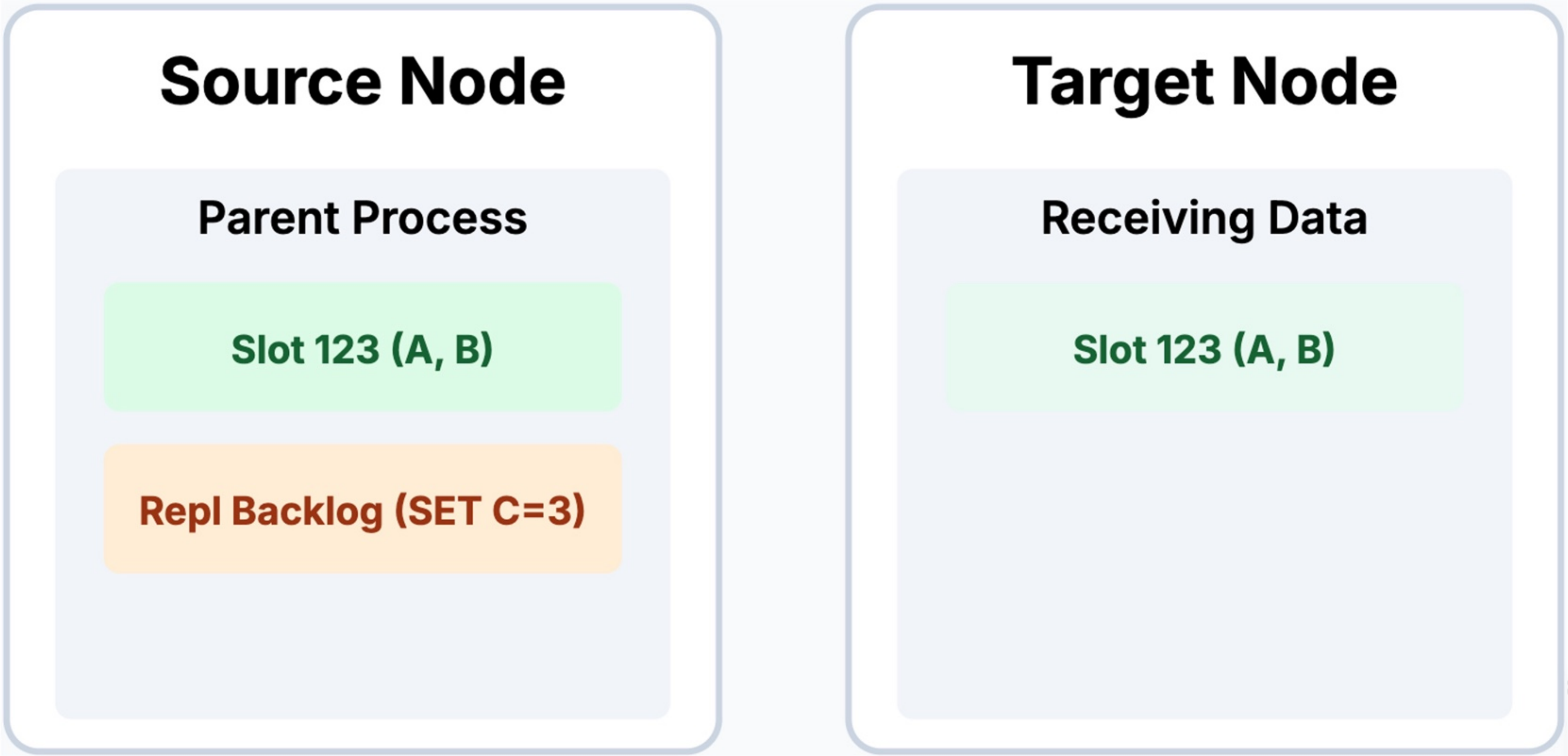
Child Process

Target Node

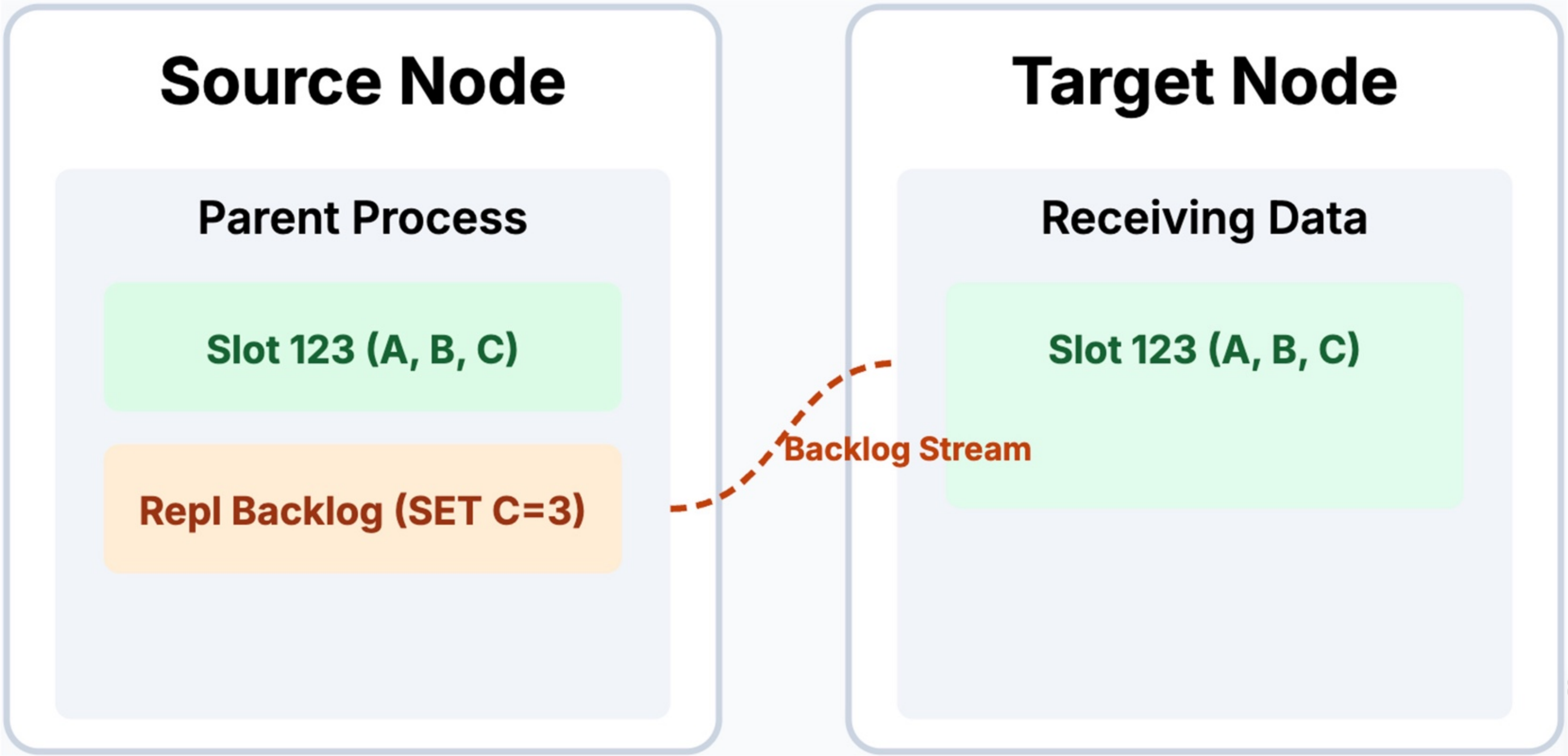
Receiving Data

Slot 123 (A, B)

6. Snapshot completes, child process exits



7. The parent process begins draining



Source Node

Parent Process

Slot 123 (A, B, C)

Repl Backlog (SET C=3)

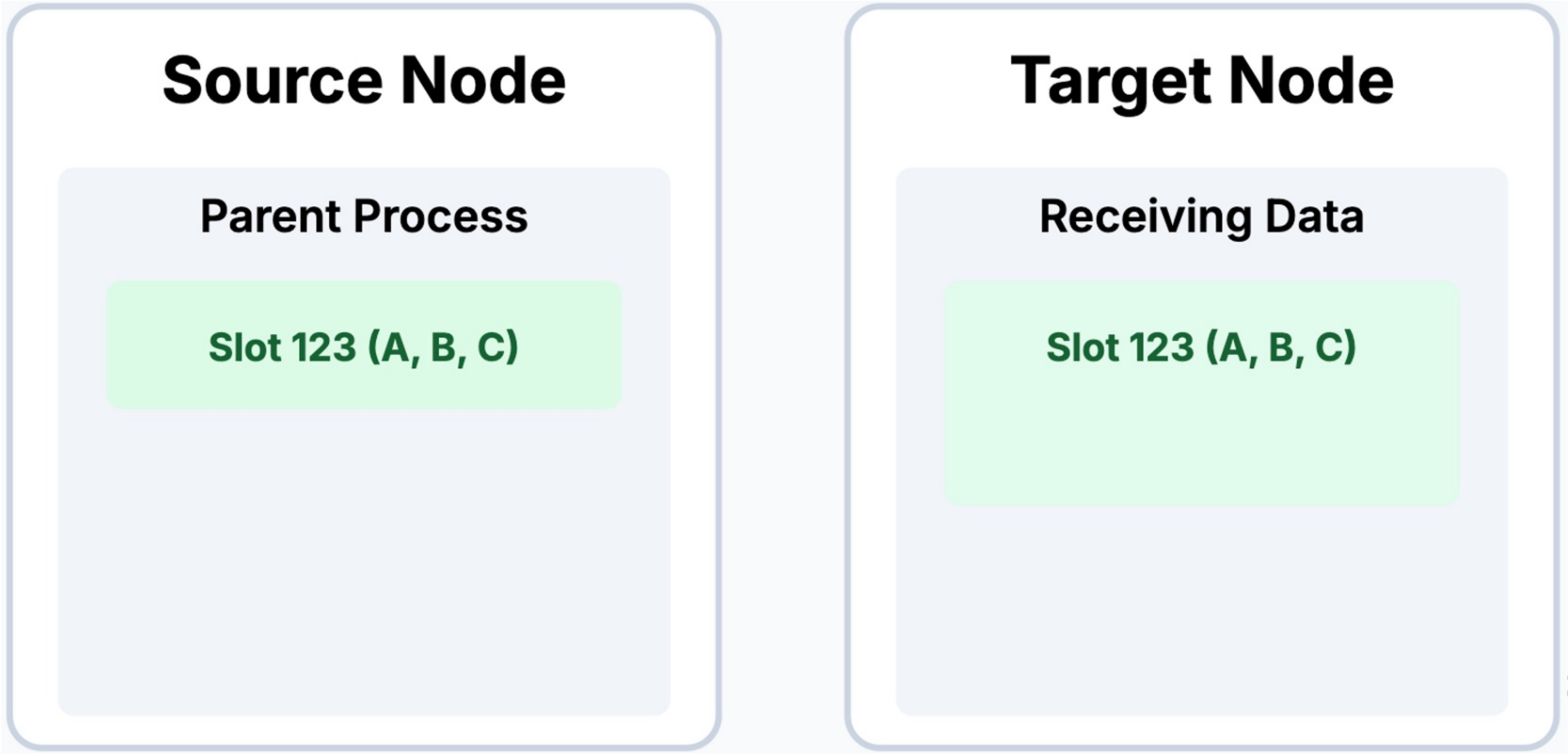
Backlog Stream

Target Node

Receiving Data

Slot 123 (A, B, C)

8. Draining completes and the target is fully synced



Source Node

Parent Process

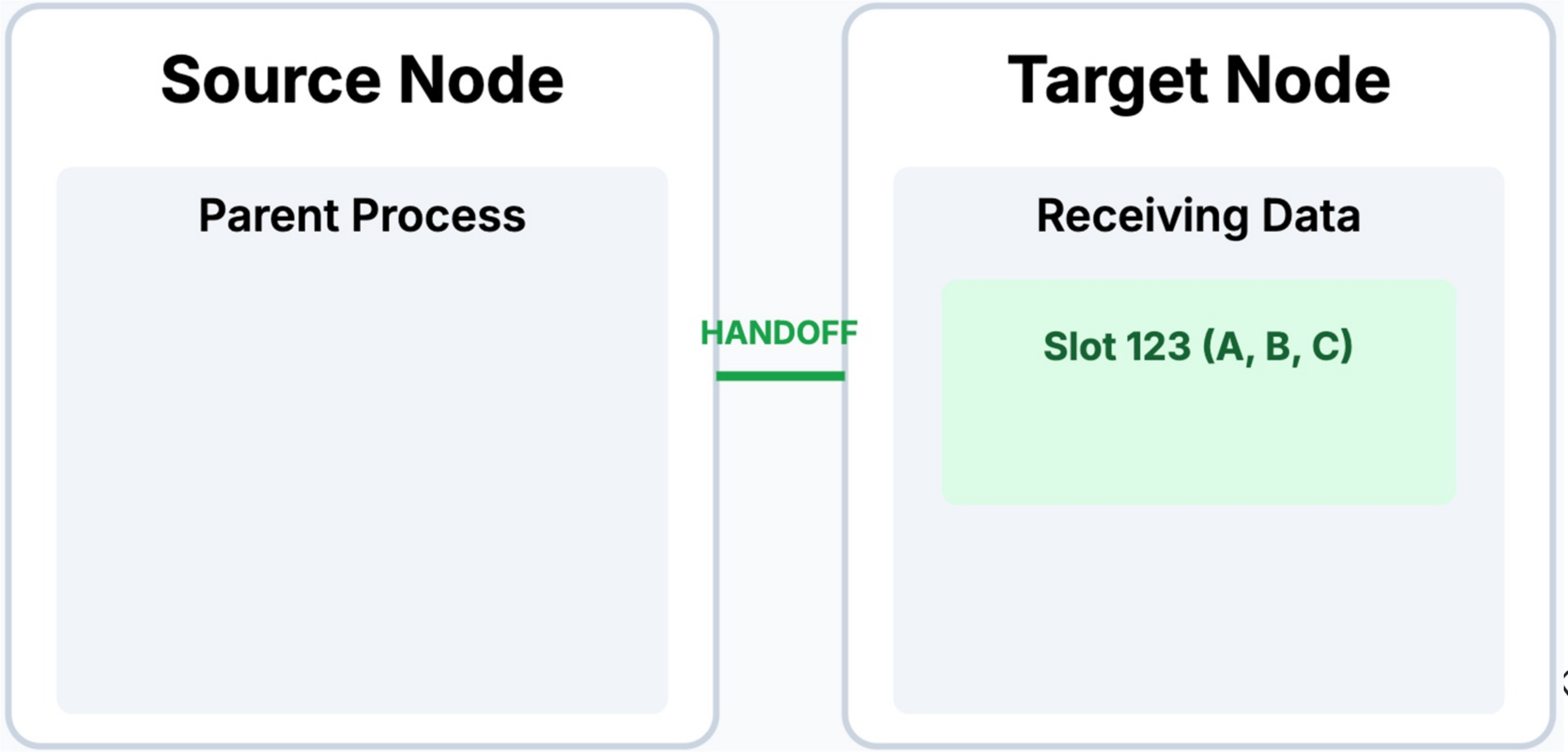
Slot 123 (A, B, C)

Target Node

Receiving Data

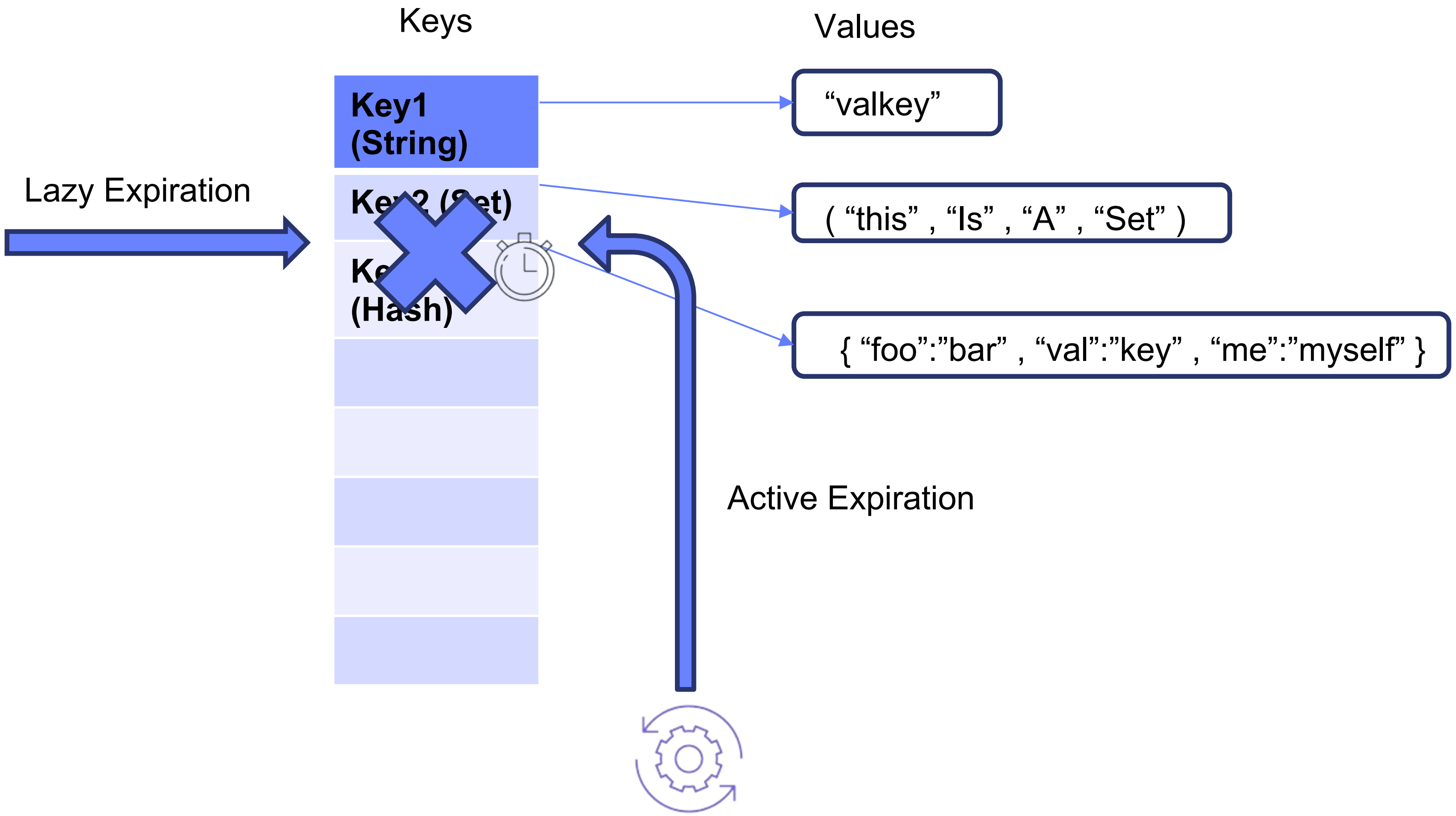
Slot 123 (A, B, C)

9. A final handoff atomically transfers ownership



Hash field expiration

Valkey Hash Objects



Per-field TTL, what is it good for ?

Hot/Cold data management
Remove long un-accessed hash entries

IoT / telemetry
Different sensors expire at different times

Log Management
Periodically Expire old logs

Session management
Store multiple sessions in one hash with separate expiries.

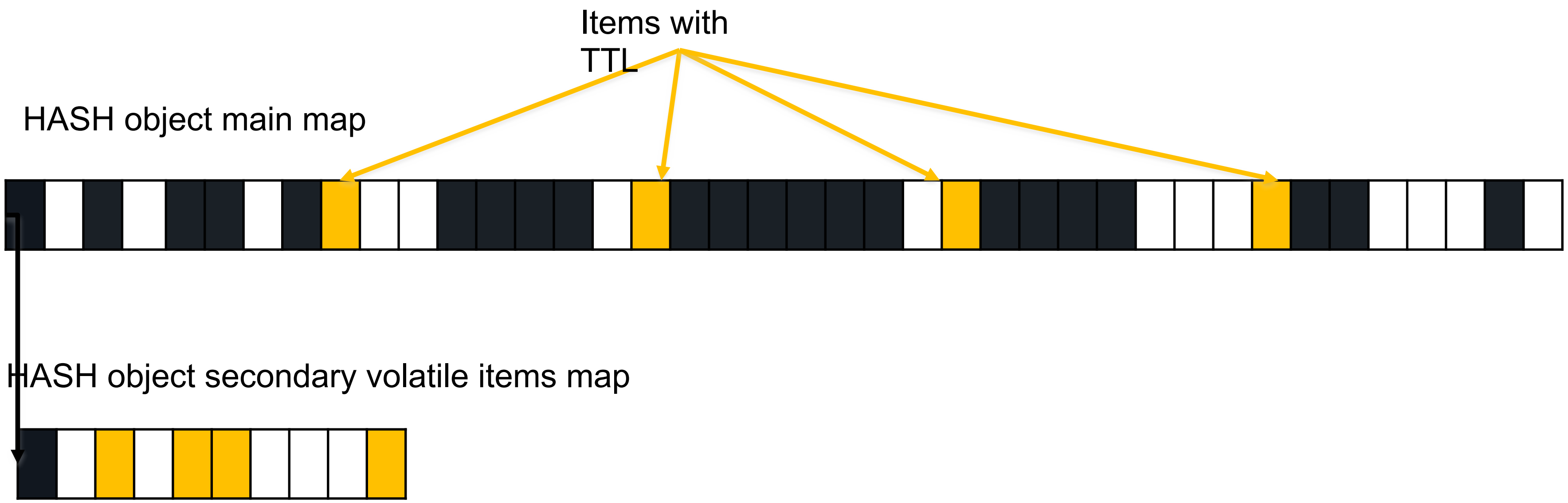
Feature flags / tokens
Expire specific configs without touching others

The Core Challenges

- Need to track and expire individual fields inside a hash (why?)
  - Expiration cycle efficiency
  - Bounded memory growth
- Cannot impact the complexity of existing Hash objects
  - Most Hash operation are O(1) complex
  - This implies we cannot simply apply sorting on volatile items
- Memory overhead
  - TTL overhead is up to 8 bytes, but tracking will require extra metadata.
- Performance
  - Support similar throughput for workloads adjusting to use hash fields expiration.

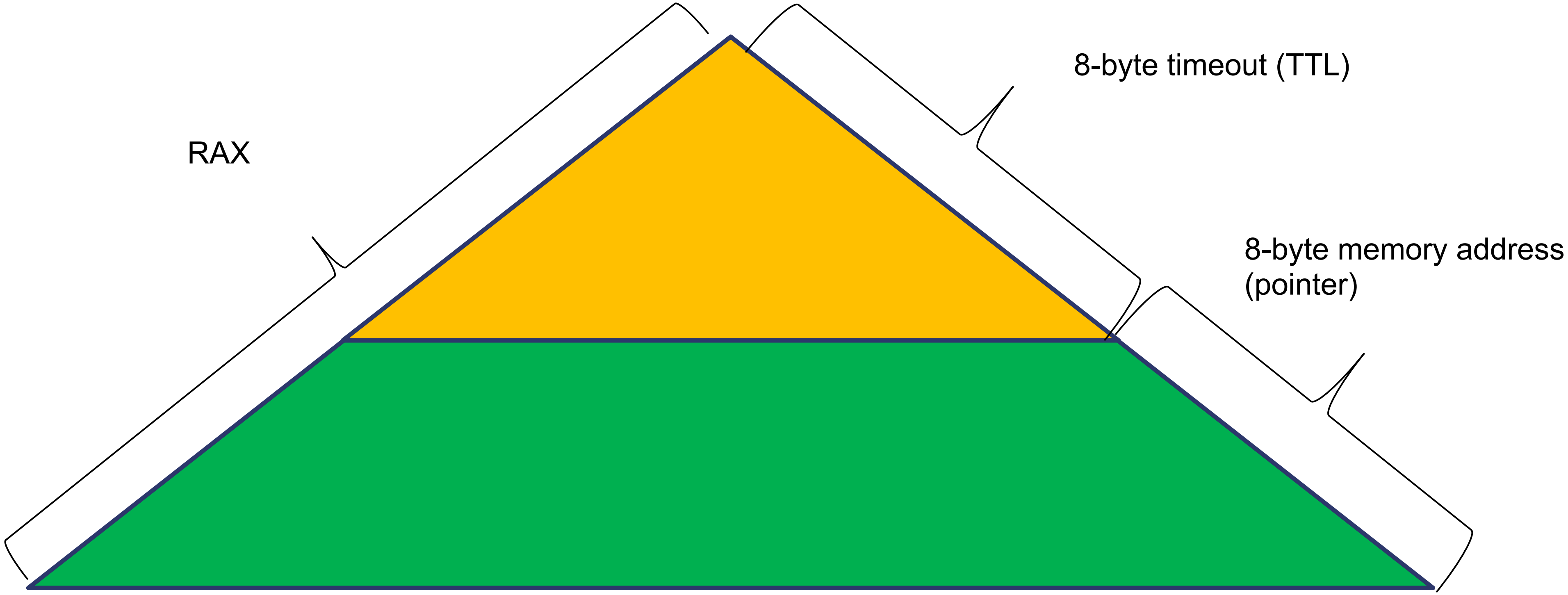
Naïve Solution 1: Separate Hashtable

Idea: Maintain parallel hash mapping field → expiry
Pros: Simple to implement
Cons: Wasted active expiration CPU cycles and high expiration staleness



Naïve Solution 2: Trie (Radix Tree)

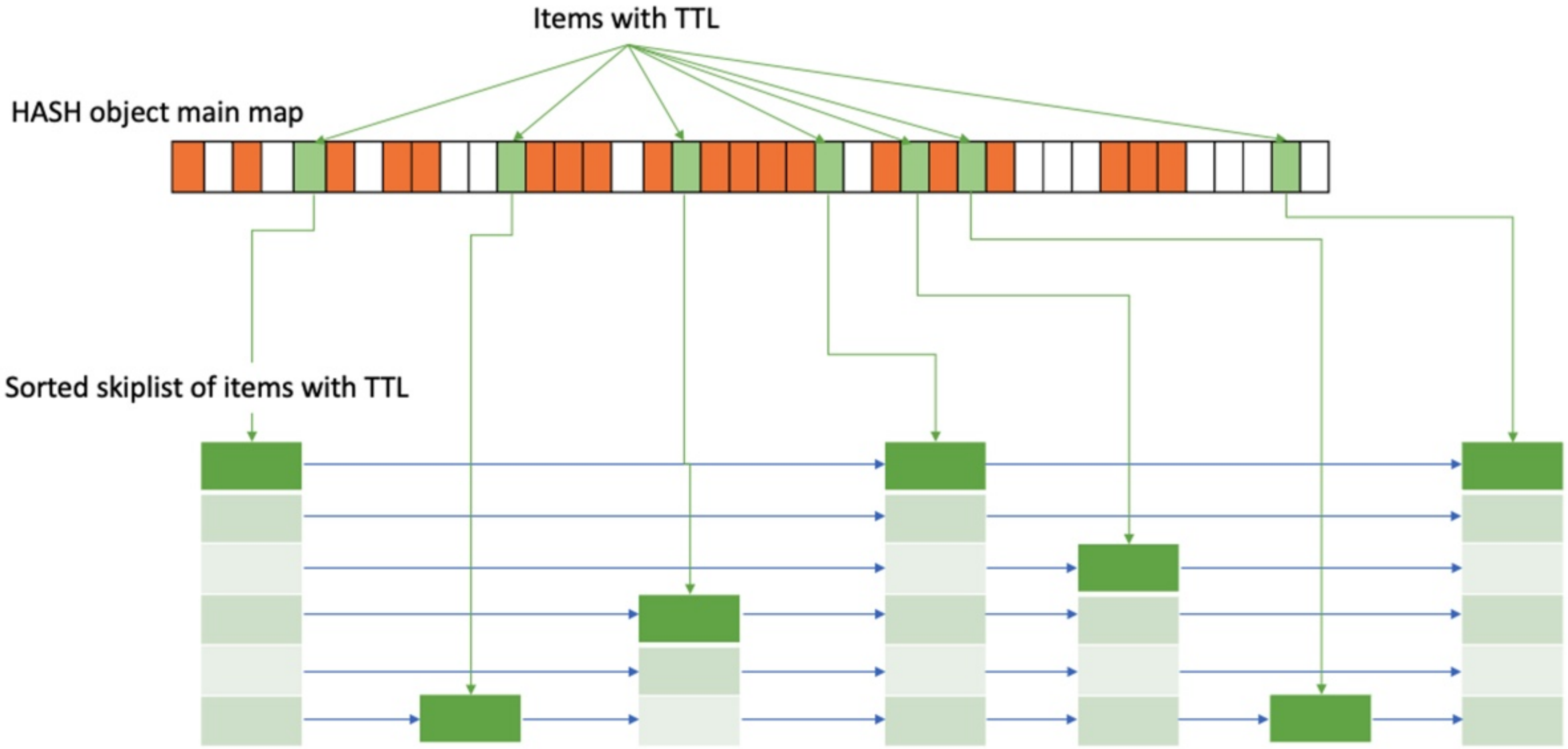
Idea: Use a radix tree keyed by (hash_key, field) for expiries
Pros: Follows an existing solution. Constant time lookups and modifications.
Cons: High memory overhead (over 54 bytes per volatile hash entry)



0x00	0x00	0x01	0x8c	0x77	0xb5	0xa4	0x03	0x7f	0xfd	0xfa	0xd4	0xc9	0x20	0x00	0x00

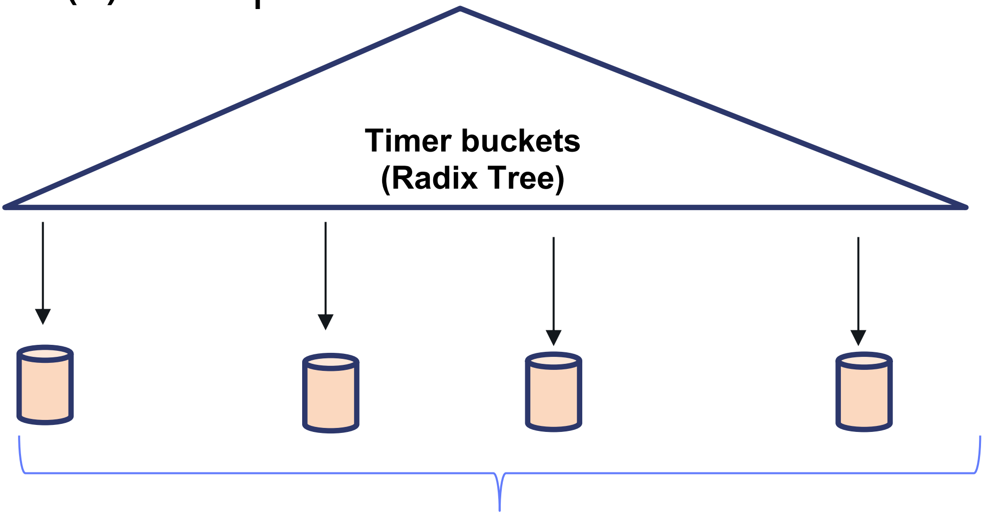
Naïve Solution 3: Sorted Structure

Idea: Maintain sorted (expiry, field) list/tree per hash
Pros: Efficient sorted iteration over volatile elements
Cons: O(log n) inserts/deletes, higher CPU cost with frequent updates



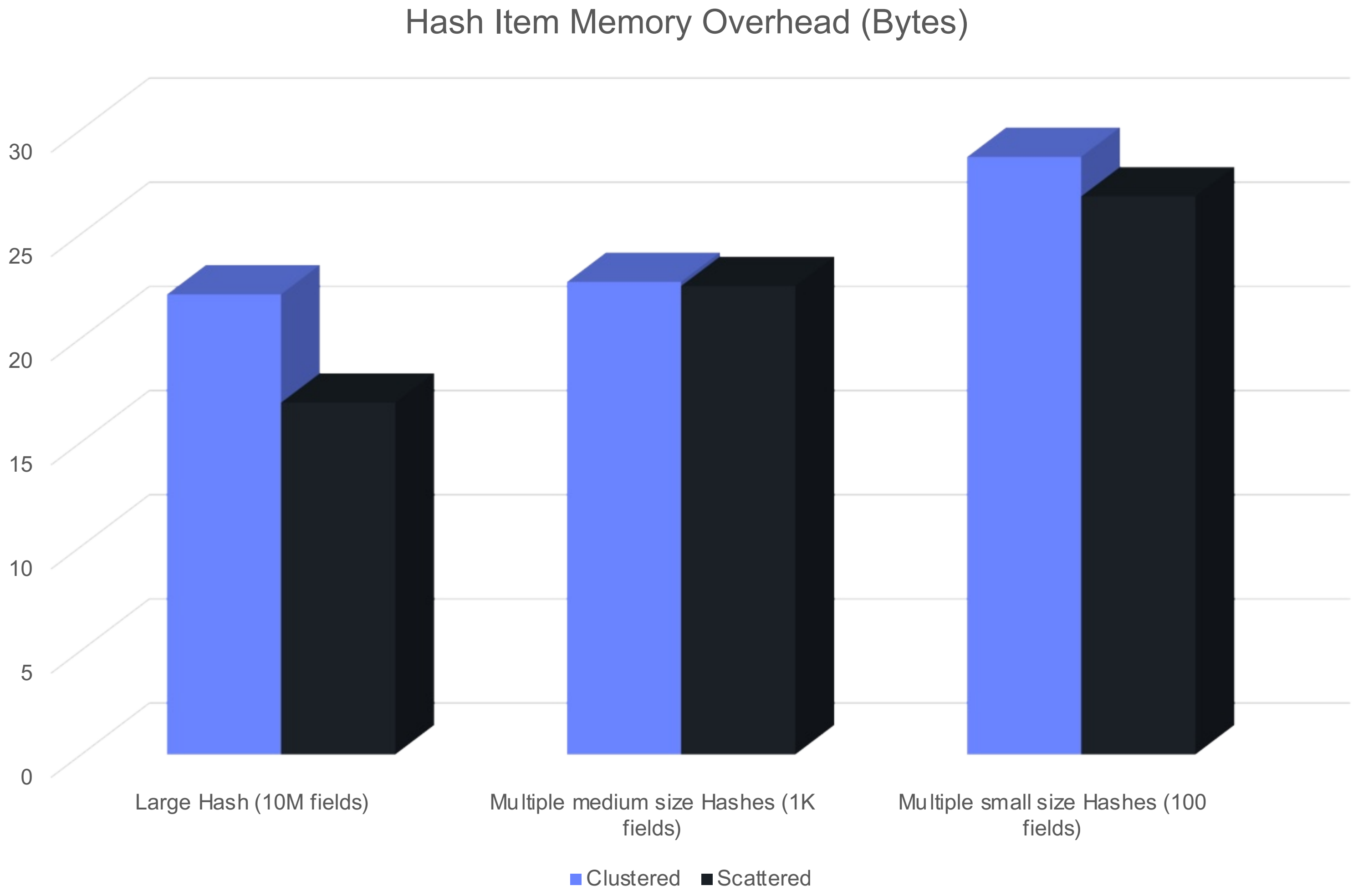
Chosen Approach: Coarse Buckets

- Idea: Semi-Sorted data structure. Group expirations into fixed time buckets
  - Buckets are sorted and maintained by a radix tree
  - Bucket has multiple encodings to support fast access/mutations and memory efficiency
  - Dynamic buckets interval resolution is adjusted as it grows (to reduce expiration staleness)
- Expire fields by processing buckets
- Benefits: O(1) lookups and modifications. minimal memory overhead, batch expiry



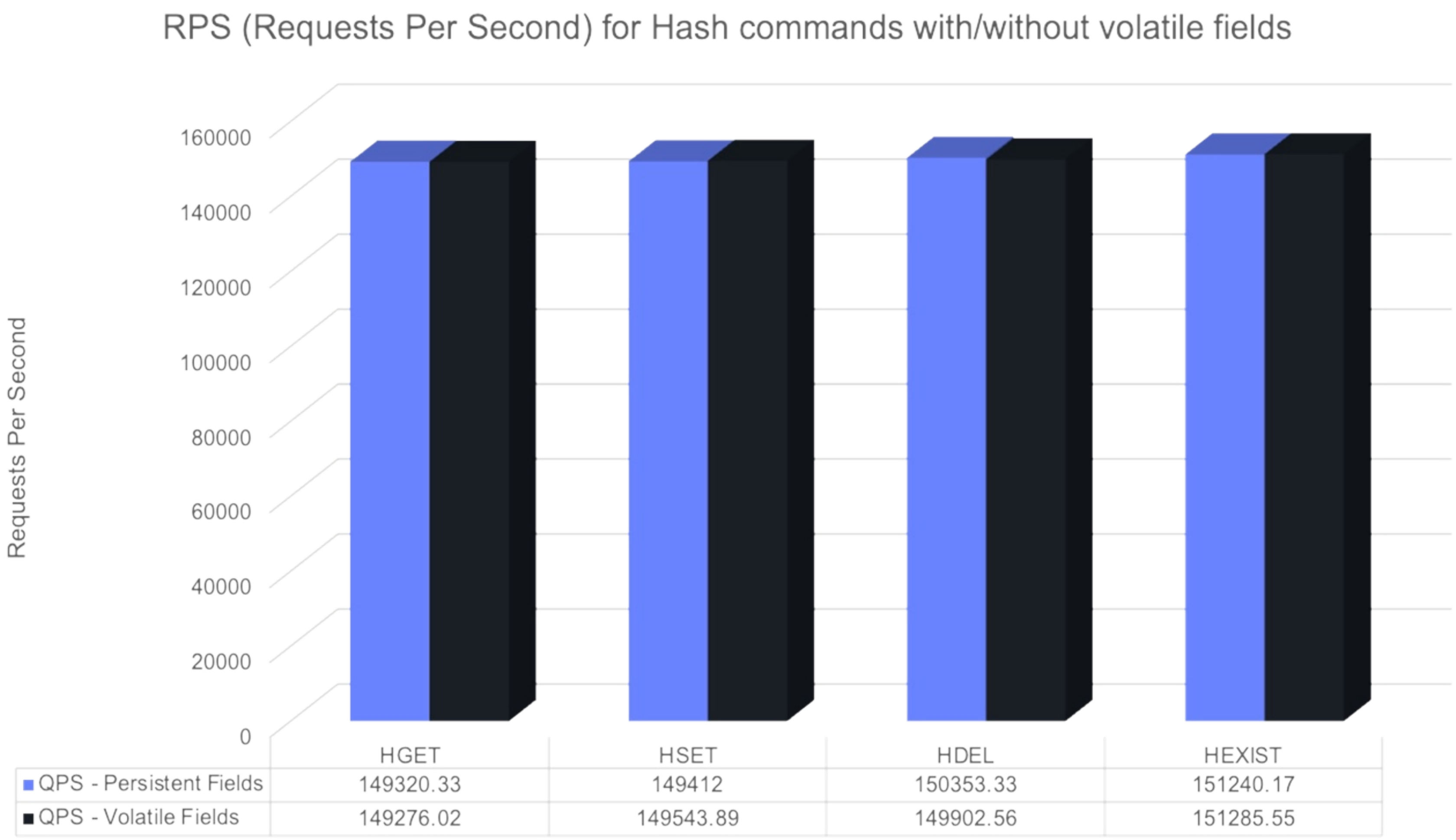
Ranged resolution Intervals (eg 16 milli – 10 seconds)

Benchmarking - Memory



Valkey

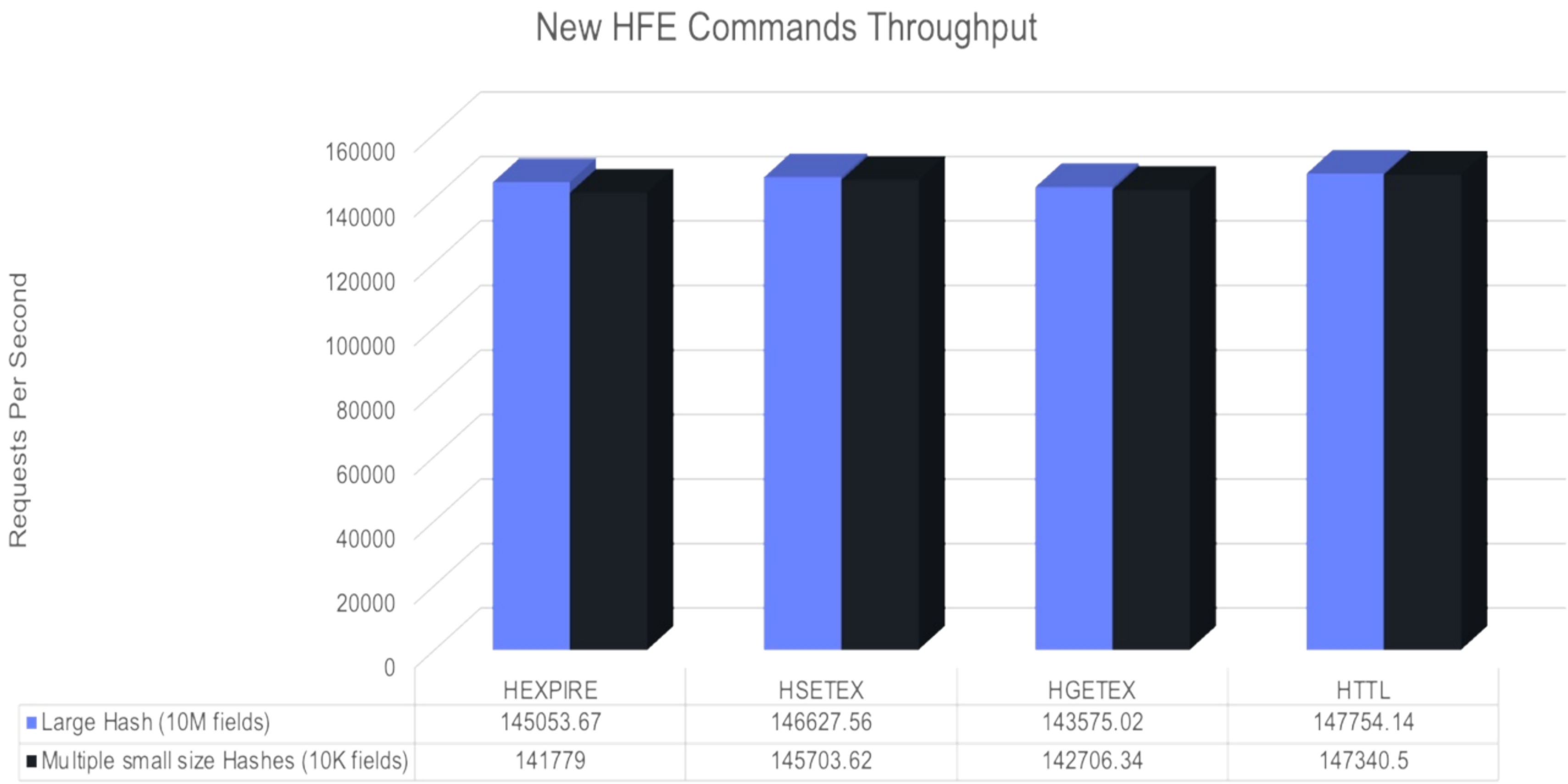
Benchmarking


RPS (Requests Per Second) for Hash commands with/without volatile fields

	HGET	HSET	HDEL	HEXIST
🟦 QPS - Persistent Fields	149320.33	149412	150353.33	151240.17
■ QPS - Volatile Fields	149276.02	149543.89	149902.56	151285.55

Benchmarking

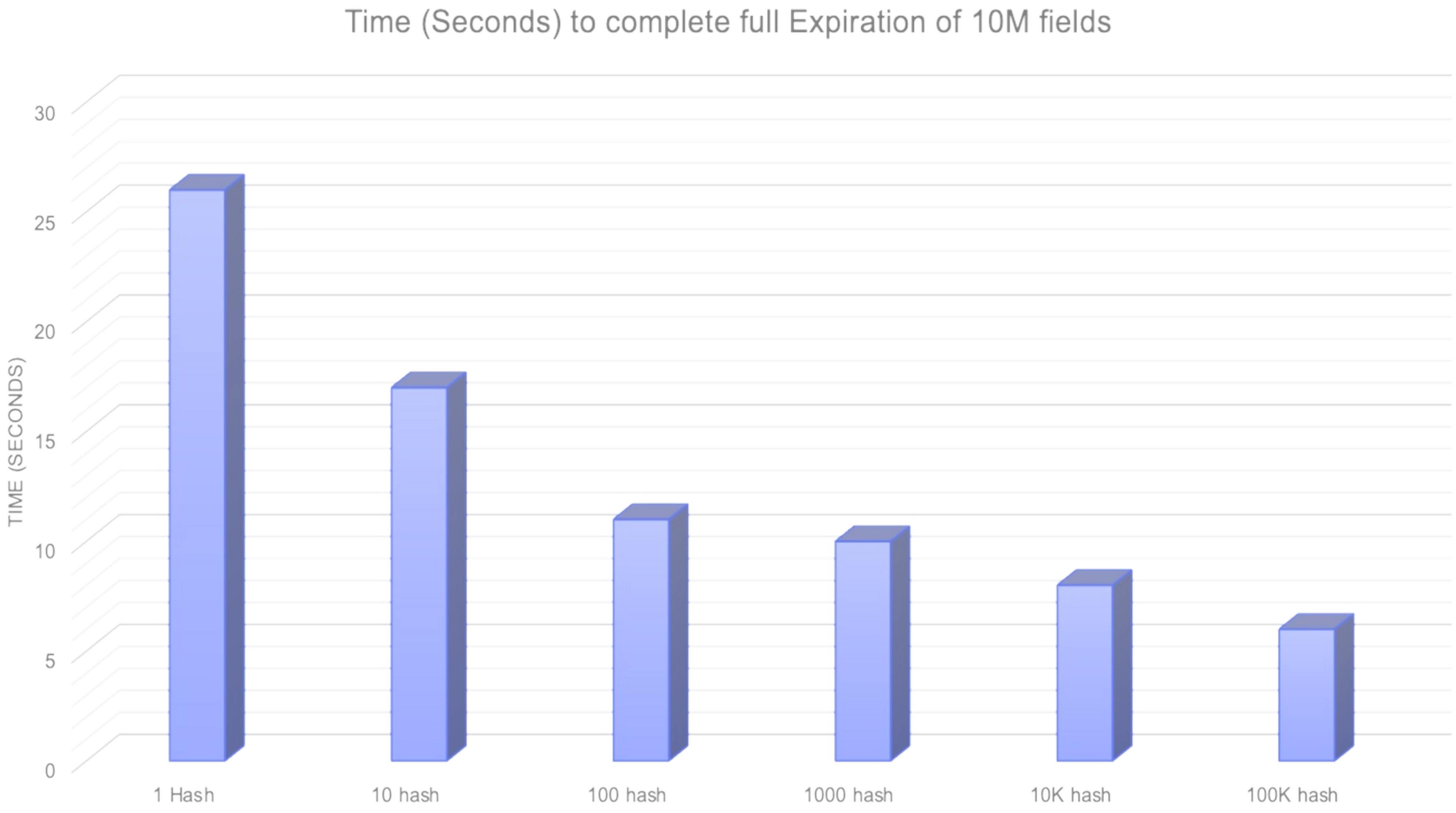
New HFE Commands Throughput



	HEXPIRE	HSETEX	HGETEX	HTTL
🟦 Large Hash (10M fields)	145053.67	146627.56	143575.02	147754.14
⬛ Multiple small size Hashes (10K fields)	141779	145703.62	142706.34	147340.5

Valkey

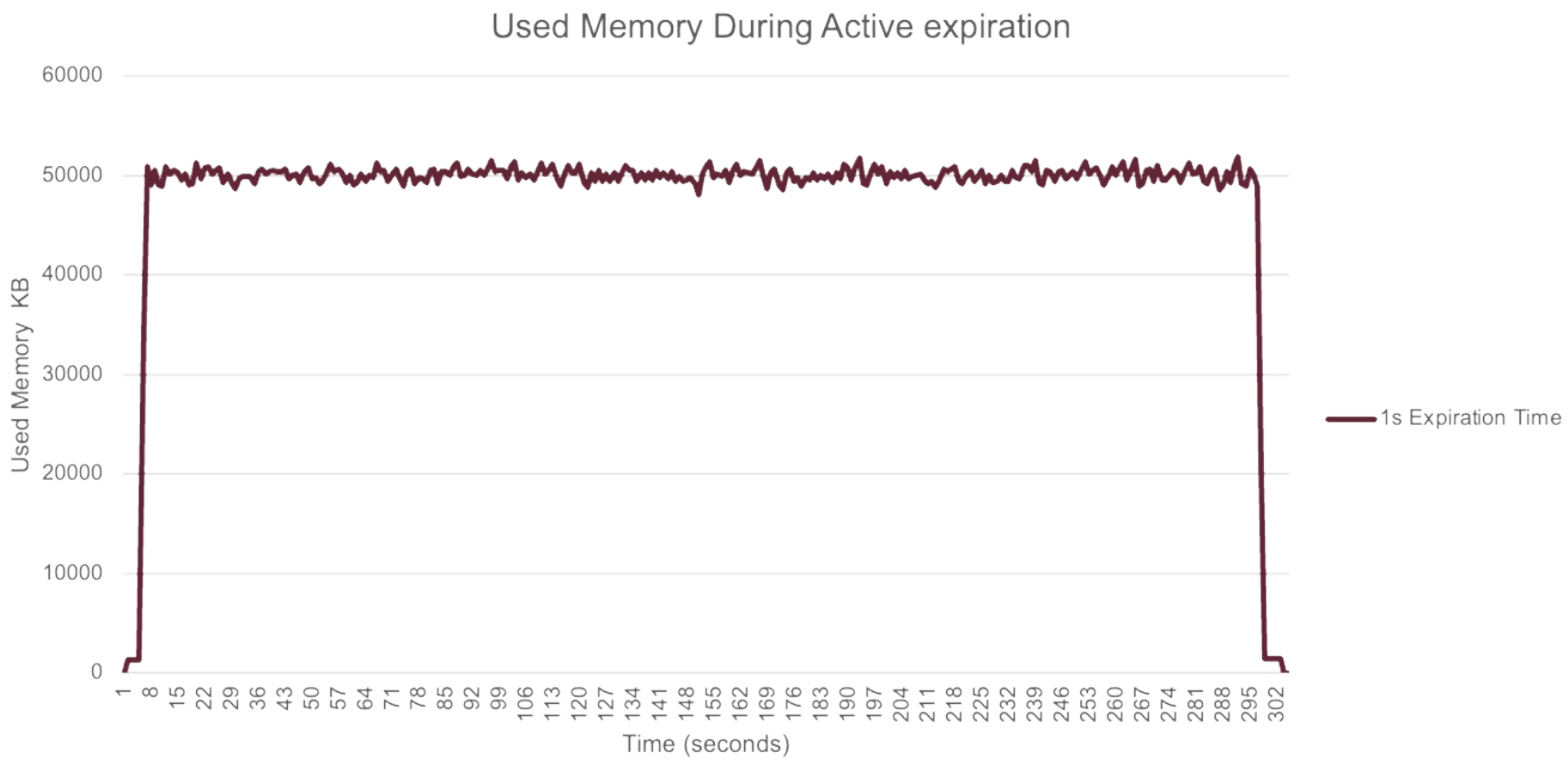
Benchmarking – Expiration



Valkey

Active expiration keeps bounded memory footprint

Memory = (Injection Throughput) x (AVG TTL) x (AVG Item memory)



Next Steps

Improved memory efficiency

- Support “packed” small hashes for better memory efficiency
- Use overloaded hashtables to reduce the memory consumption of large buckets

Improved performance

- Better CPU utilization with use of prefetching and SIMD techniques

Extended functionality

- Allow placing TTL on SET object fields.

Just the beginning

What else will be new in Valkey 9?

- Zero-copy responses for large requests (Up to 20% higher throughput)
- Support for Multipath TCP
- Memory prefetching for pipelining commands (Up to 40% higher throughput)
- Stability improvements for large (1000+ node) clusters
- SIMD optimizations for BITCOUNT and hyperloglog commands (up to 200% higher throughput)
- New filtering options for CLIENT LIST command
- New DELIFEQ command to conditionally delete
- By-polygon support for Geospatial indexes
- … and so much more …

This could be you!

- Ran Shidlansik **@ranshid**
- Binbin **@enjoy-binbin**
- Jacob Murphy **@murphyjacob4**
- Madelyn Olson **@madolson**
- YueTang-Vanessa **@YueTang-Vanessa**
- cxljs **@cxljs**
- Sarthak Aggarwal **@sarthakaggarwal97**
- amanosme **@amanosme**
- Hanxi Zhang **@hanxizh9910**
- Seungmin Lee **@sungming2**
- uriyage **@uriyage**
- Katie Holly **@Fusl**
- Nicky-2000 **@Nicky-2000**
- Allen Samuels **@allenss-amazon**
- yzc-yzc **@yzc-yzc**
- zhaozhao.zz **@soloestoy**
- asagegeLiu **@asagege**
- nitaicaro **@nitaicaro**
- Matthew **@utdrmac**
- Omkar Mestry **@omanges**
- Viktor Söderqvist **@zuiderkwast**
- kukey **@kukey**
- Harkrishn Patro **@hpatro**
- Avi Fenesh **@avifenesh**
- Amit Nagler **@naglera**
- Josh Soref **@jsoref**
- youngmore1024 **@youngmore1024**
- Rain Valentine **@SoftlyRaining**
- skyfirelee **@artikell**
- Wen Hui **@hwware**
- yulazariy **@yulazariy**
- Yakov Gusakov **@gusakovy**
- charsyam **@charsyam**
- Simon Baatz **@gmbnomis**
- Thalia Archibald **@thaliaarchi**
- chzhoo **@chzhoo**
- xbasel **@xbasel**
- Stav Ben-Tov **@stav-bentov**
- wuranxx **@wuranxx**
- Ayush Sharma **@ayush933**
- chx9 **@chx9**
- KarthikSubbarao **@KarthikSubbarao**
- Hüseyin Açacak **@huseyinacacak-janea**
- アンドリー・アンドリ @odaysec
- Ping Xie **@PingXie**
- Lipeng Zhu **@zhulipeng**
- Linus Unnebäck **@LinusU**
- Vitah Lin **@vitahlin**
- kronwerk **@kronwerk**
- Vadym Khoptynets **@poiuj**
- muelstefamzn **@muelstefamzn**
- zhenwei pi **@pizhenwei**
- George Padron **@DoozkuV**
- Björn Svensson **@bjosv**
- aradz44 **@aradz44**
- Hiranmoy Das Chowdhury **@HiranmoyChowdhury**
- Yair Gottdenker **@yairgott**
- Roshan Khatri **@roshkhatri**
- nesty92 **@nesty92**
- carlosfu **@carlosfu**
- Arthur Lee **@arthurkiller**
- Shai Zarka **@zarkash-aws**
- Sergey Kolosov **@skolosov-snap**
- Nathan Scott **@natoscott**
- lucasyonge **@lucasyonge**
- WelongZuo **@WelongZuo**
- Jim Brunner **@JimB123**
- jeon1226 **@jeon1226**
- Benson-li **@li-benson**
- Meinhard Zhou **@MeinhardZhou**
- Nikhil Manglore **@Nikhil-Manglore**
- Bogdan Petre **@bogdanp05**
- eifrah-aws **@eifrah-aws**
- Ricardo Dias **@rjd15372**
- secwall **@secwall**
- Anastasia Alexandrova **@nastena1606**
- Marek Zoremba **@zori-janea**
- VoletiRam **@VoletiRam**

Valkey

Thank you!