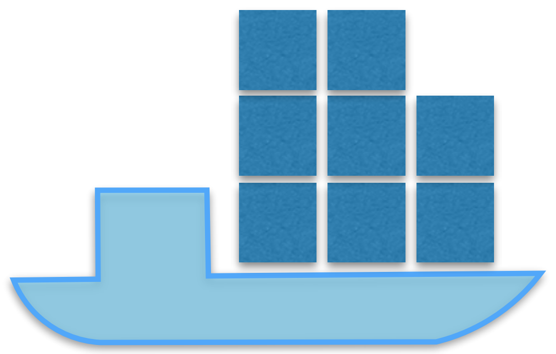
JavaScript for Extending Low-latency In-memory Key-value Stores

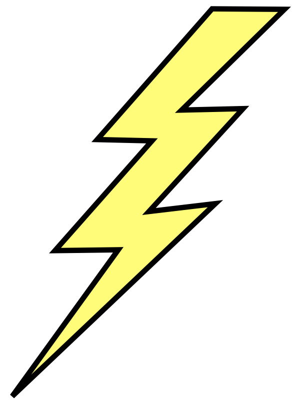
Tian Zhang Ryan Stutsman

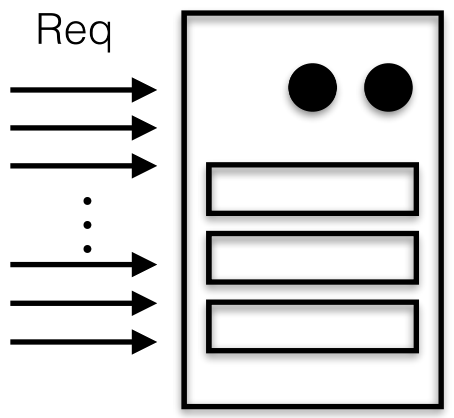
THE UNIVERSITY OF UTAH

Introduction

Today's large scale key-value stores (e.g. Ramcloud, FaRM,etc.) are able to:







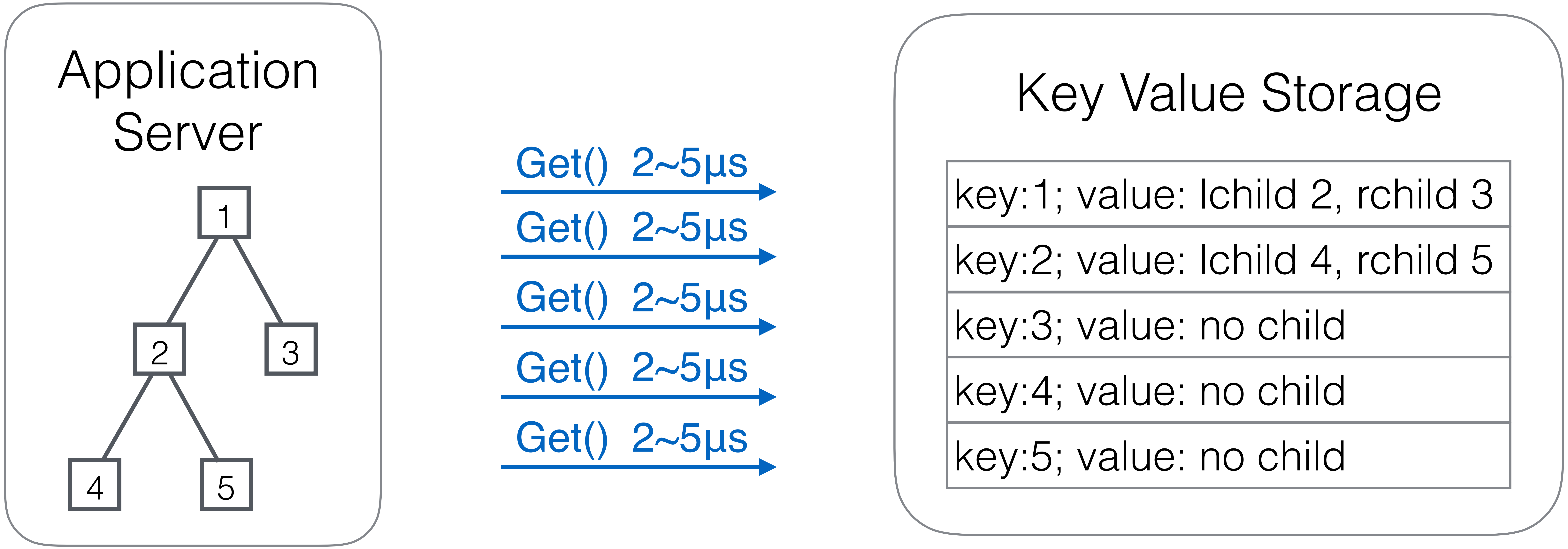
- Store TBs ~ PBs of data.
- 2~5μs end to end access time.
- Perform billions of operations per second.

Have today's applications been able to properly leverage such systems?

Not yet.

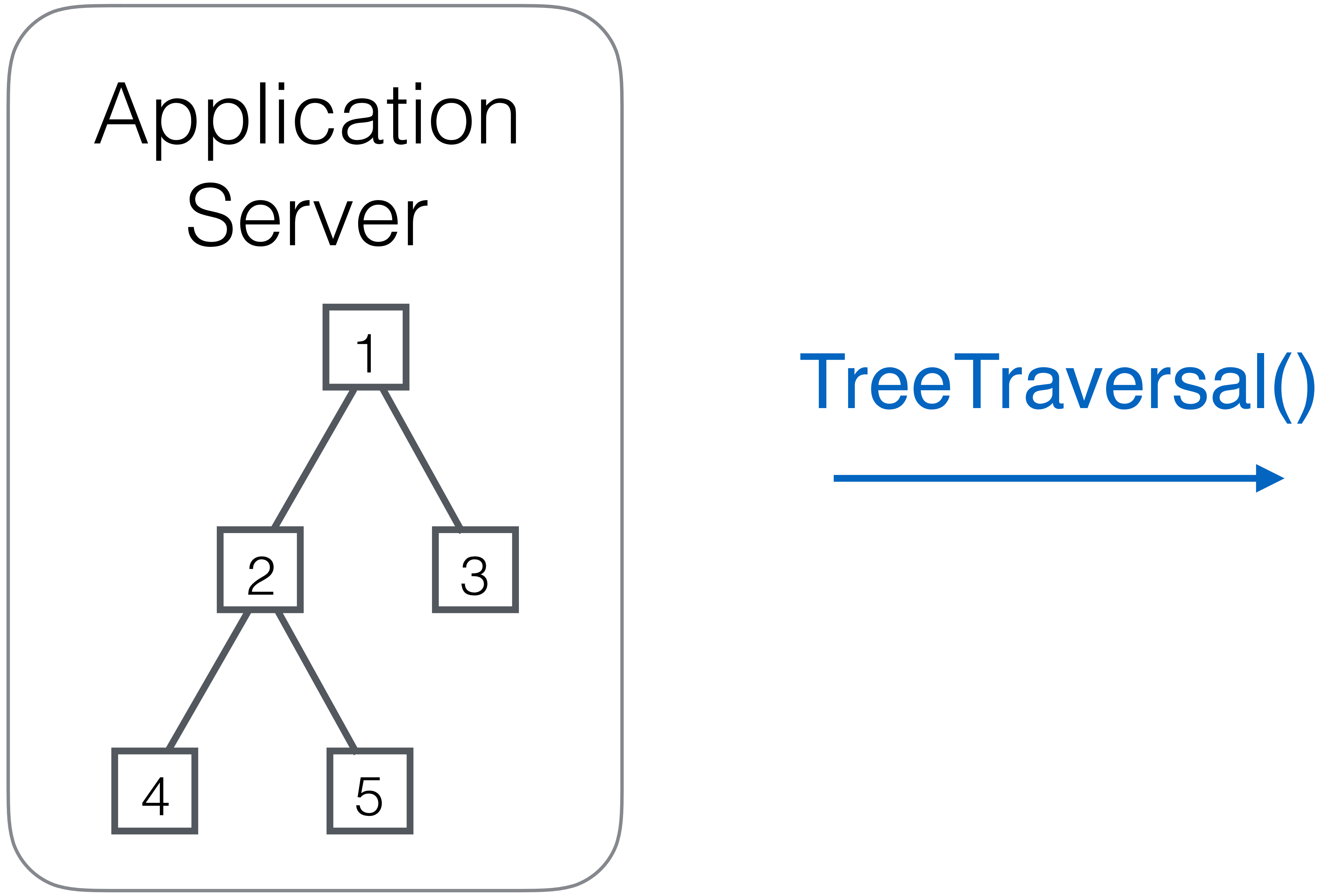
Semantic Gap

- Implementing high level semantics with KVS APIs requires many roundtrips.



Semantic Gap

- Implementing high level semantics with KVS APIs requires many roundtrips.

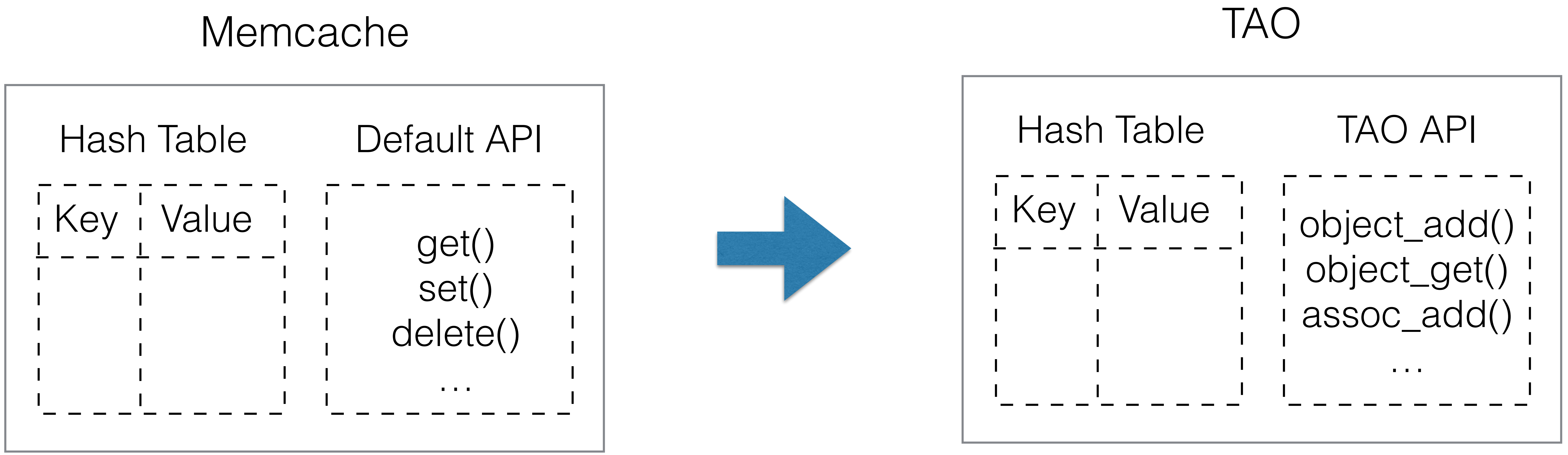


Key Value Storage

key:1; value: lchild 2, rchild 3
key:2; value: lchild 4, rchild 5
key:3; value: no child
key:4; value: no child
key:5; value: no child

Existing Solution - Customized KVS

- Facebook has implemented TAO, a social graph data model in Memcache.
- Entities (e.g. people) are modeled as objects, their connections as associations.
- TAO stores objects and association lists, and provide APIs to operate on them.



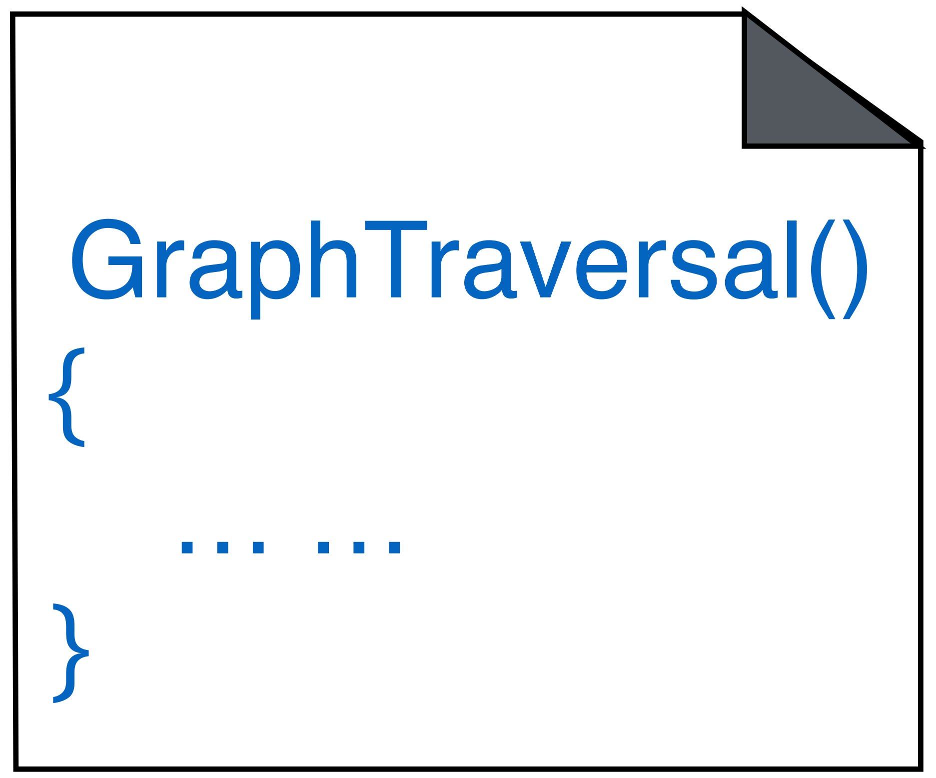
Existing Solution - Customized KVS

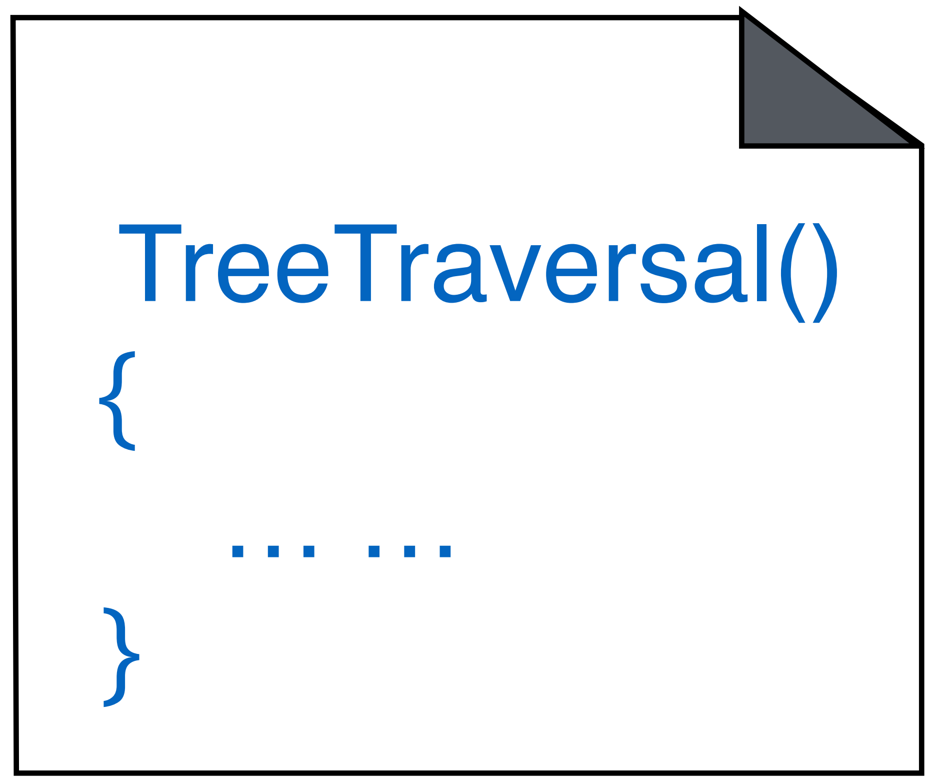
- Other customized KVS:
  - Md-hbase with multi-attribute access support.
  - Comet with application-specific actions.
  - G-store with consistent multi-key access support.

Disadvantage : ad-hoc solutions for specific applications, not general.

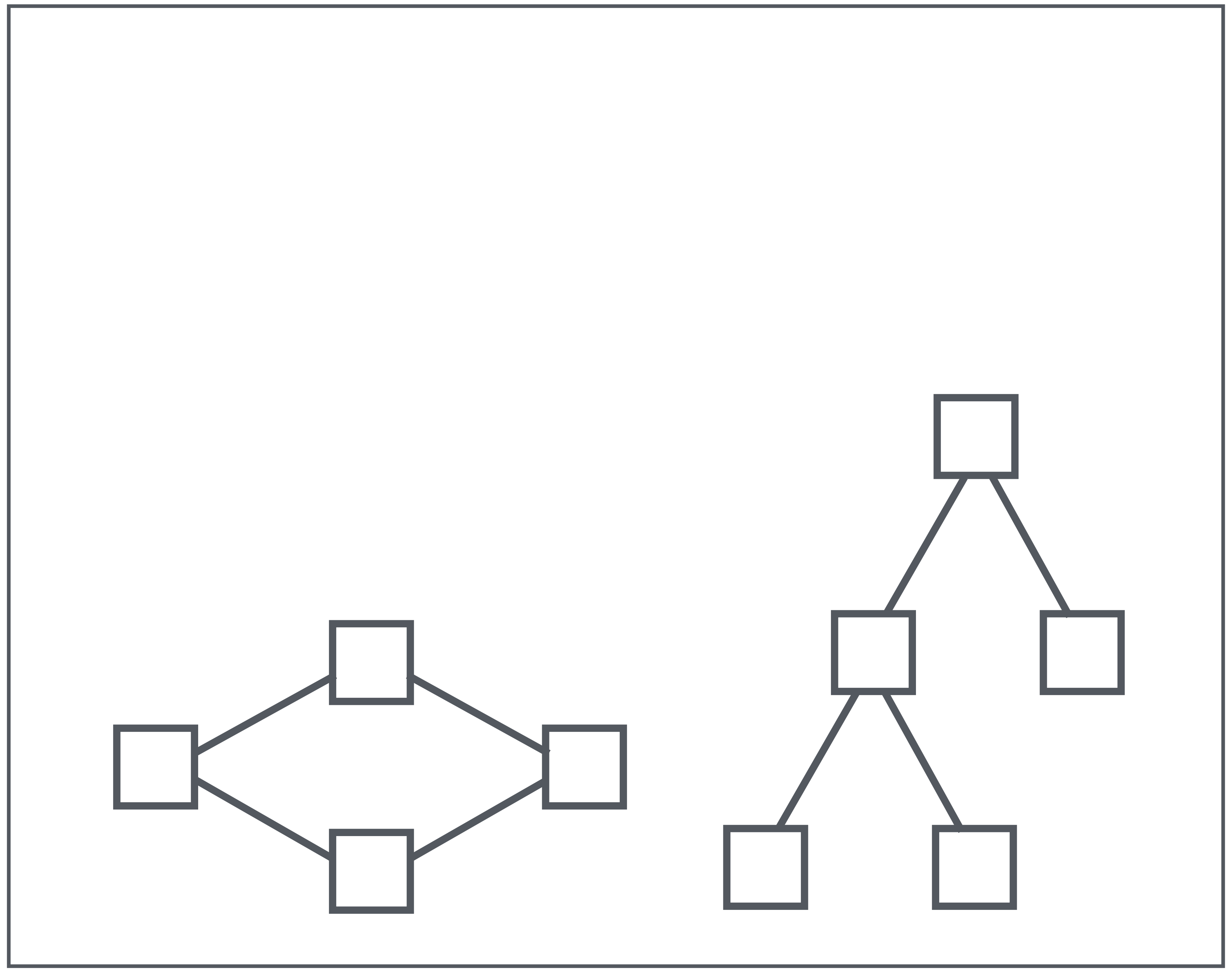
Our Solution - Runtime Extensibility

- A more general solution is to allow pushing custom logic to KVS at runtime.
- The KVS can be dynamically reconfigured to support new applications.



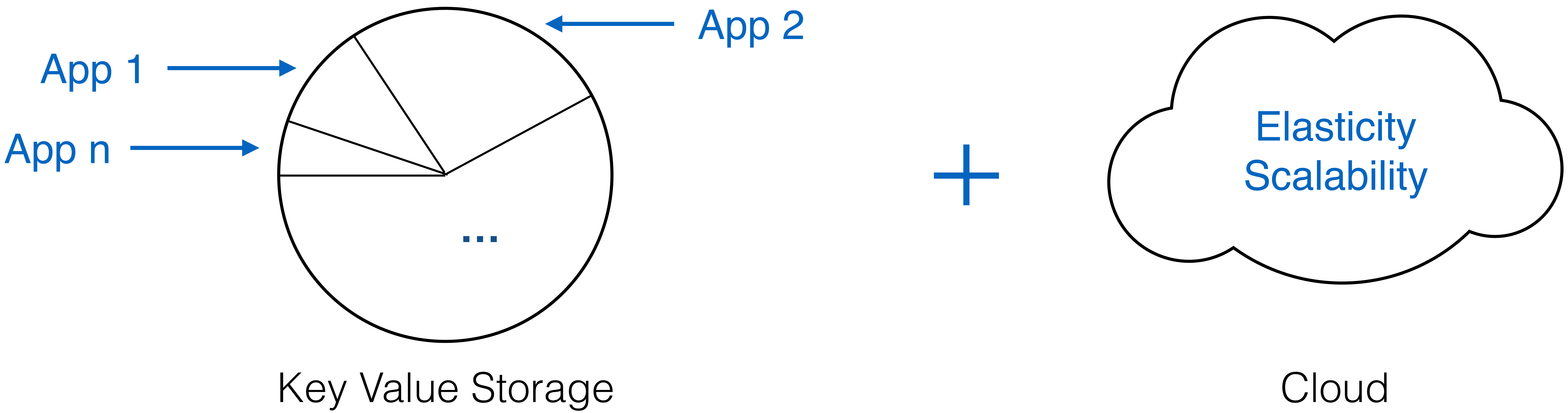


Key Value Storage



More to Consider - Cloud Service

- Combining workloads improves utilization.
- Deploying the system on cloud to leverage the elasticity and scalability.



Challenge - Isolation with Low Overhead

- KVS is fast, server processes requests in 2 µs.
    - Its performance extremely sensitive to any overhead, even cache misses.
- Security isolation incurs 3 sources of overhead:

  - The cost of safer languages.

  - Context switches between protection domains.

  - Interactions with DB across protection domain boundaries.

Approaches

SQL - Difficult to implement new operators or complex algorithms.

- Leading to ad-hoc extensions such as SimSQL, SciDB etc.

Native/C++ - Flexible. Need process isolation, interactions happen over IPC.

JavaScript - Flexible. Embedding V8 engine in DB process.

C++ vs. JavaScript

- Our expectation : JavaScript may be slower than C++:

  - JIT compiler doesn’t optimize as aggressively as C++ compiler.

  - Less static type information.

  - Garbage collection.

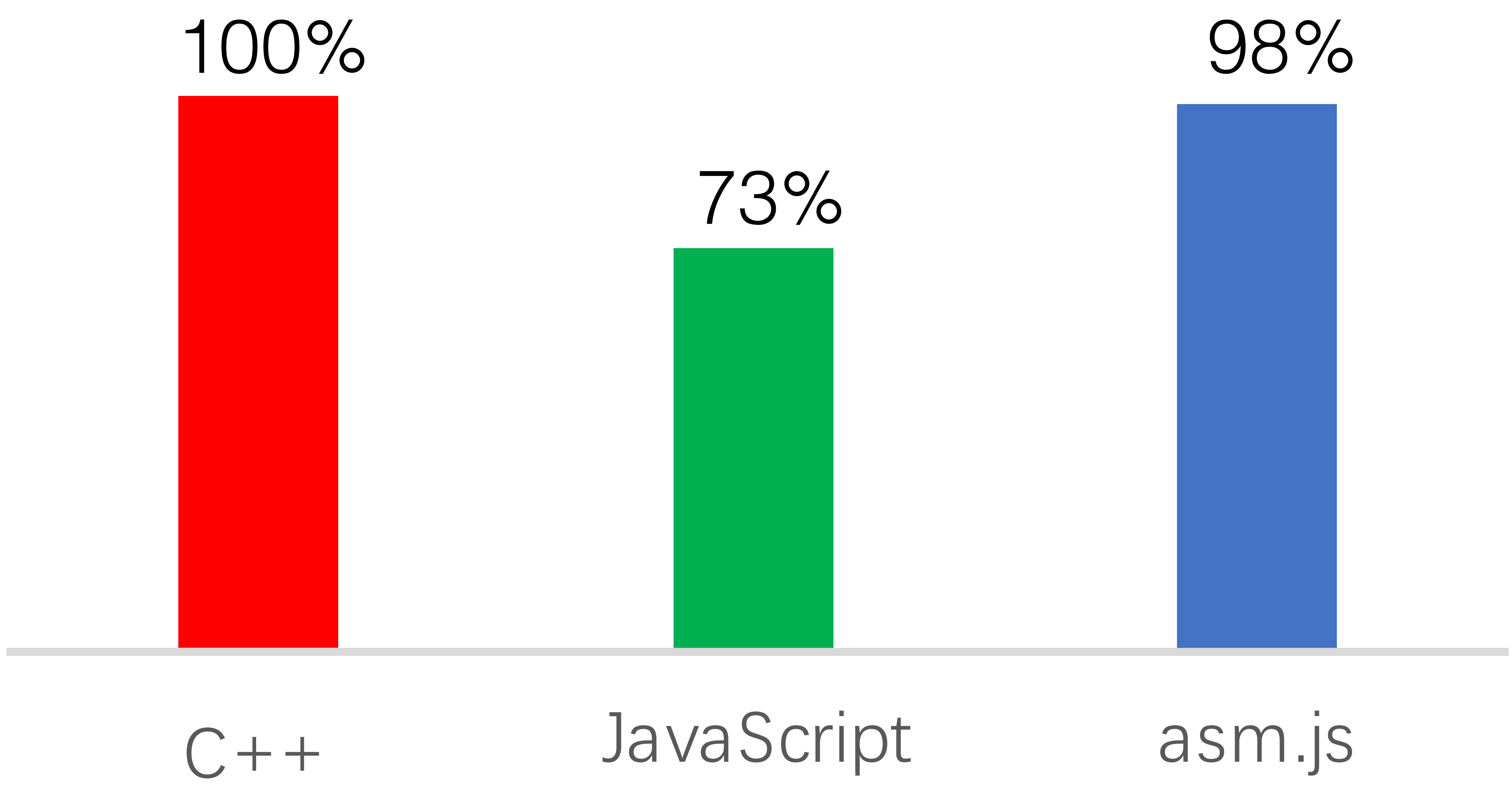
C++ vs. JavaScript

- Experiment setup:
  - Compare same query logics written in **C++** and **JavaScript**.
  - We also compare these queries written in C++ and compiled to **asm.js**.
  - Queries process 1 GB of records with varied selectivity and compute intensity.
  - We don't consider GC in this experiment, assume procedures are often short.

C++ vs. JavaScript

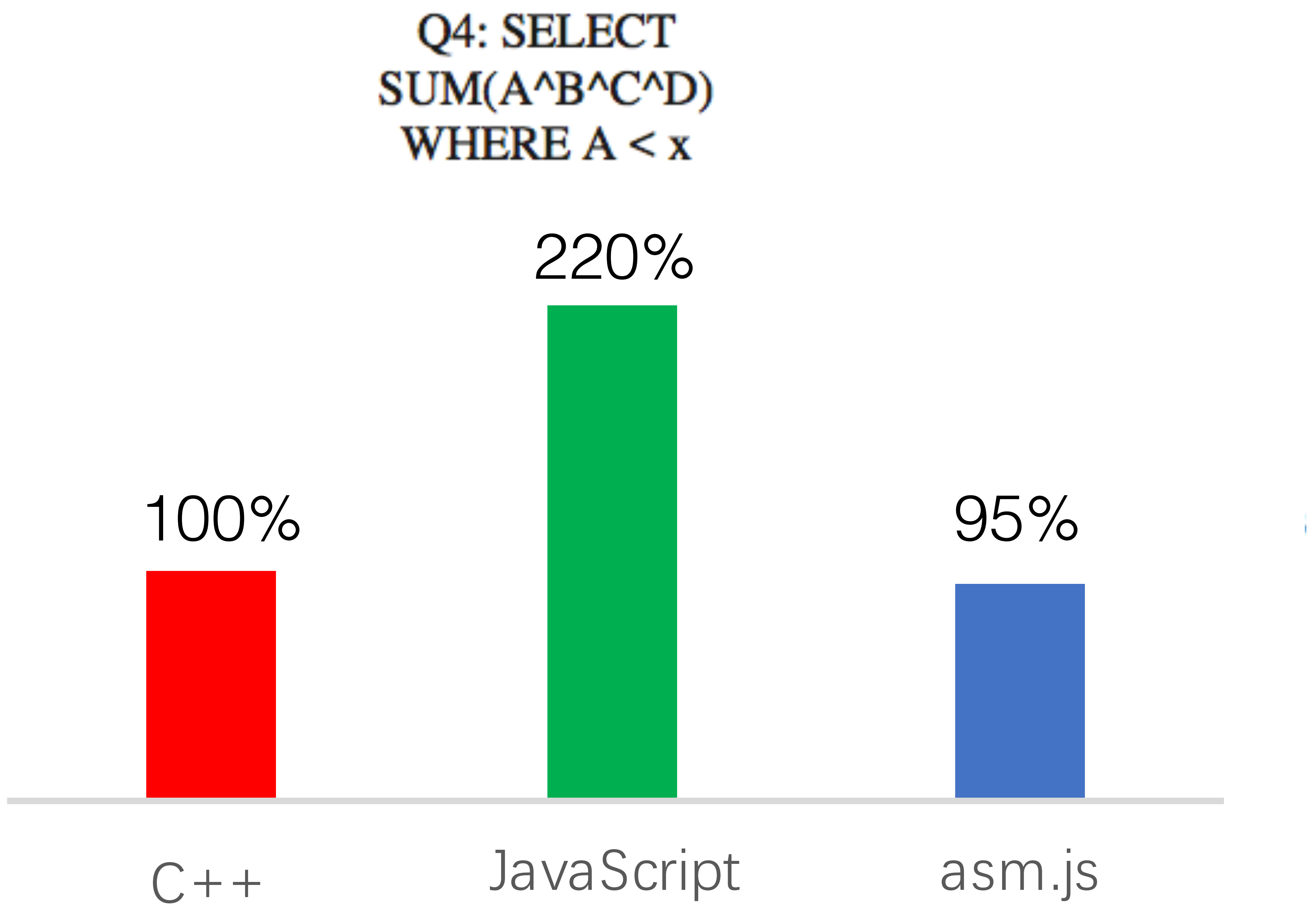
- For our memory intensive query, JavaScript is 27% slower than C++.
- Performance of asm.js is just 2% slower than C++.

**Q2: SELECT A+A
WHERE A < x**



C++ vs. JavaScript

- For our compute intensive query, JavaScript is faster than C++.
- Glibc’s pow implementation may be the cause of slower performance of C++.

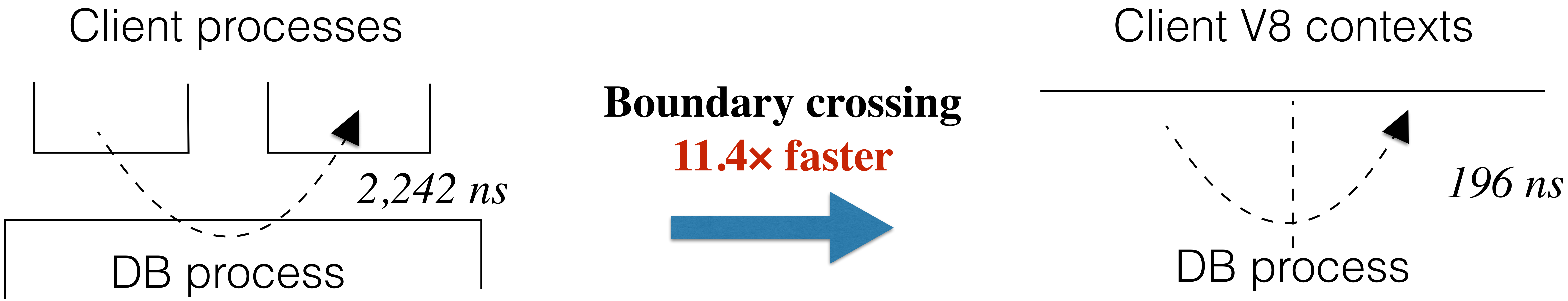


Isolation Costs, Process vs. V8

- Isolation of C++ code is done using process.
  - DB APIs invoked over IPC.
- Isolation of JavaScript code is done with V8::Context.
  - DB APIs invoked in the same process through wrappers.

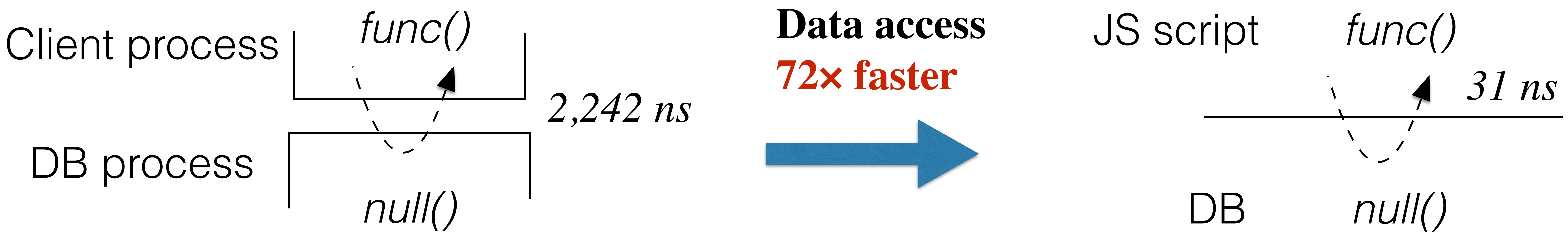
Isolation Costs, Process vs. V8

- Measured the time of process switch and V8 context switch.
- V8::Contexts switch is 11.4x faster than processes.



Isolation Costs, Process vs. V8

- Measured the time of invoking a DB API.
- Invocation from JS is 72x faster than over IPC.

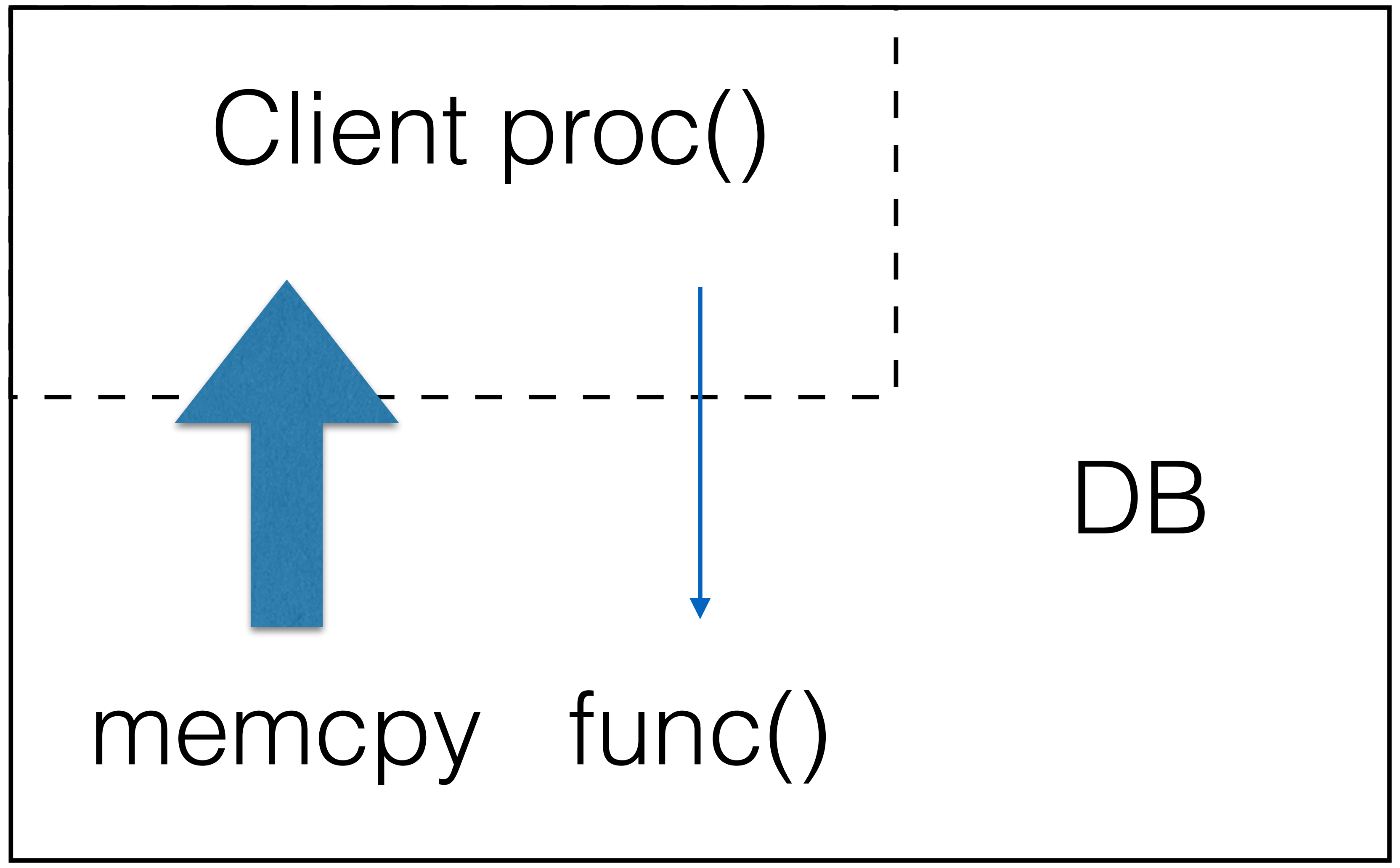


Evaluation Summary

- We compare SQL, C++ & JavaScript for their suitability of implementing our idea.
- SQL is ruled out for its limited generality.
- C++ is ruled out for high isolation overhead.
- JavaScript is promising with generality, performance and low isolation overhead.

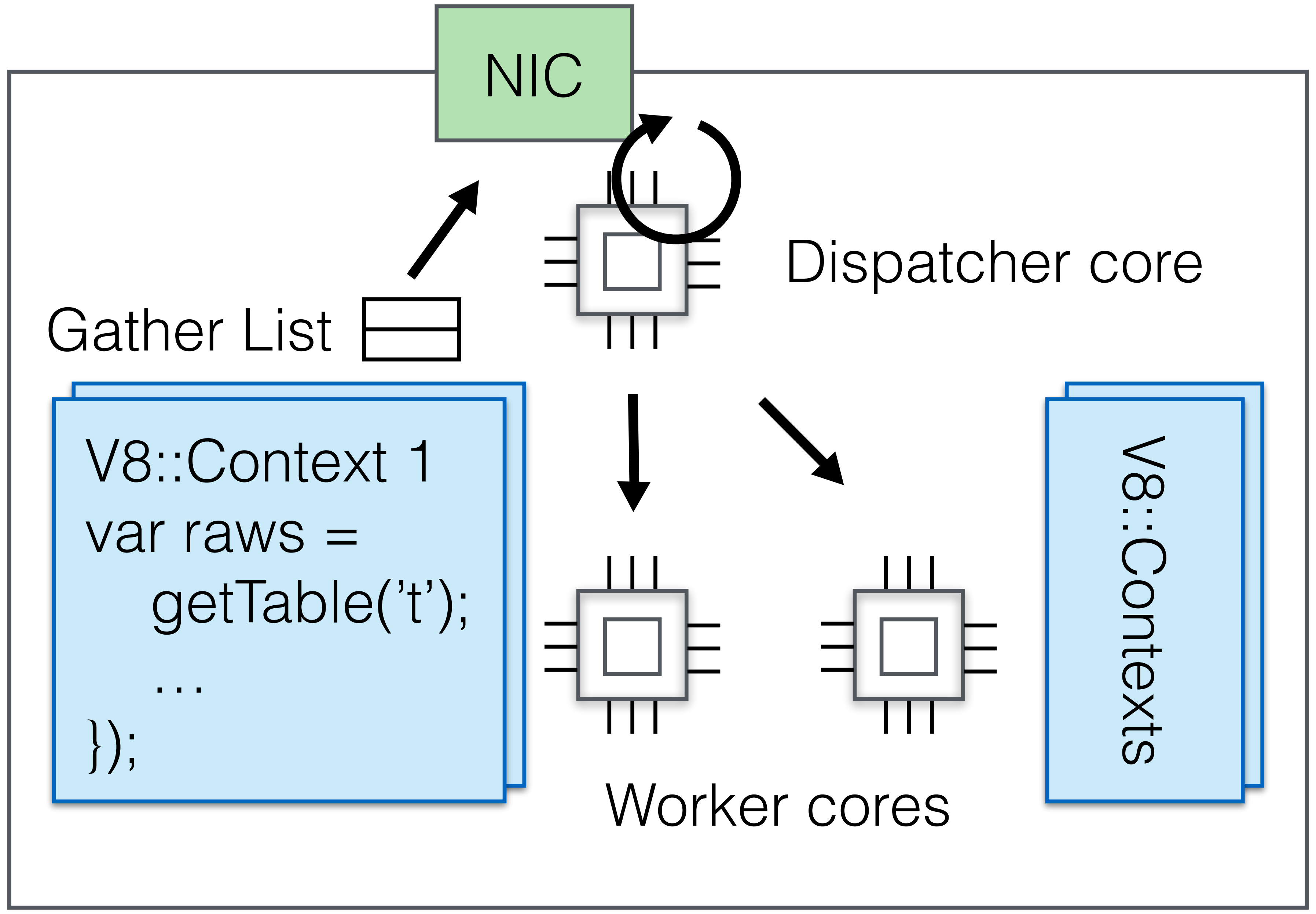
Why Not Software Fault Isolation

- For SFI, interactions require copying data between client procedure and DB.
- For data intensive procedures, that means huge overhead.



Interactions in SFI

Design



- Leverage scatter-gather list & zero copy DMA
- Leverage kernel bypassing networking (DPDK)
- Eliminating garbage collection
- Expose low level database abstractions

Conclusion & Research Questions

- **Conclusion:**

- We propose JavaScript for extending low latency in-memory KV store.
- The challenge is to keep overhead under a small fraction of 2 μs.
- Evaluation shows JavaScript as a promising choice with low isolation cost and good performance.

- **Call for feedbacks:**

- What interesting APIs can be built?
- Is there other potentially better approaches that we overlooked?