Amazon ElastiCache backed by DynamoDB

Summer Internship Presentation

Daniela Miao

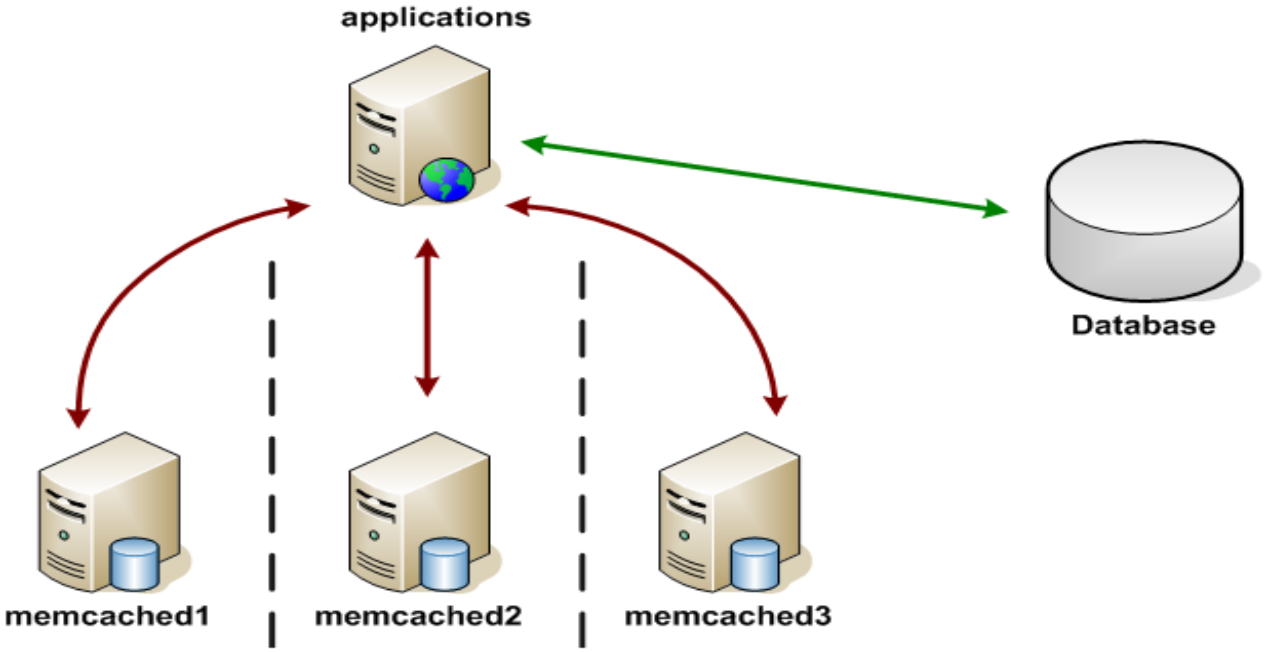
September 26th, 2013

Agenda

- Product Overview
- Intern Project Scope
- Achievements – Working Prototype
- Major Challenges
  - Design
  - Implementation
- Preliminary Performance Results
- Future Work
- Open Questions
- Q&A

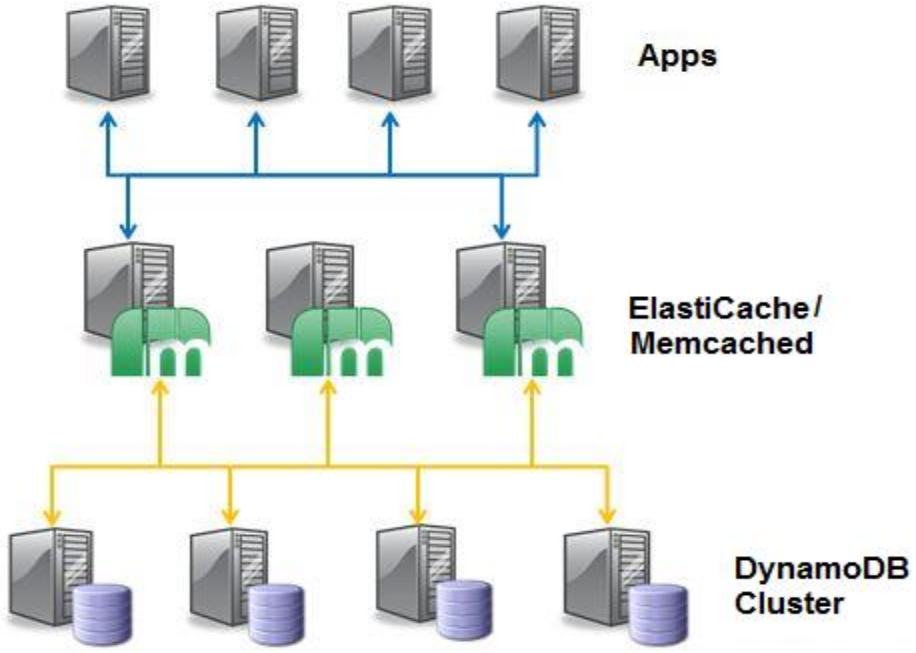
ElastiCache Product Overview

- In-memory cache in the cloud, backed up the popular Memcached engine (used by Facebook, Livejournal etc.)
- Improves the performance of web applications by allowing retrieval of information from fast cache nodes and clusters
- Existing customers: airbnb, PBS, tapjoy etc.



ElastiCache Product Overview

- Existing Problems
  - In-memory cache lacks data persistence and durability
  - Loses all data in case of power outages, node failures or inadvertent machine reboots
  - Customers are interested in getting best of both worlds: scalable performance of in-memory cache and data reliability across node reboots.



Intern Project Scope

- Explore feasibility of the product and major challenges
- Focus on “set” requests to memcached (write requests to DynamoDB)
- Prototype focused on solving the major issue of maintaining consistency across memcached engine and DynamoDB, without extensively considering error cases
- Generate initial performance results to gain a basic understanding of the impacts
- Document design progress on wiki, including a quick overview of the basic memcached architecture (included in Appendix)

Achievements – Working Prototype

- Connection between a local instance of ElastiCache (memcached engine) and DynamoDB instance launched created via AWS Console
  - AWS Console serves all AWS products/services
- Supports manual request (demo completed in Amazon)
- Supports automated requests (stress test including hundreds of concurrent requests)
  - Difficult!

Challenges – Design & Implementation

- Memcached is open-source, scalable caching engine written in C - unfortunately not documented very well
  - Libevent enables powerful and efficient connection management

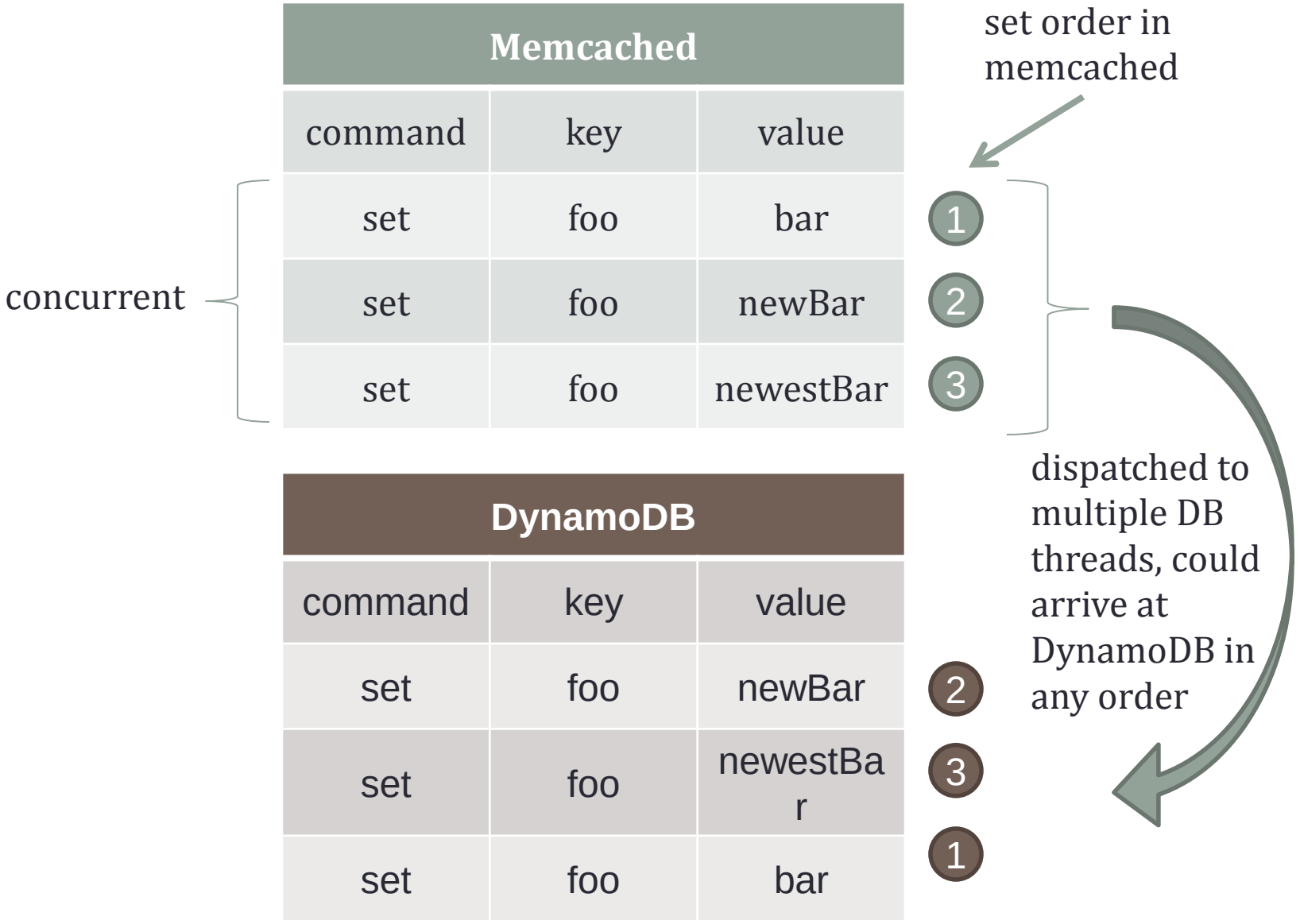
- Major Issues:
  1. How to integrate DynamoDB backend without compromising existing memcached performance (using libevent)
     - Current solution: A second thread pool for database operations
  2. No existing C Client for DynamoDB
     - Current solution: Custom C wrapper around the C++ Client (in dev)
  3. Maintaining consistency across memcached engine and DynamoDB table (behavior with concurrent sets on same key)
     - Current solution: Additional counter hash table to keep track of item updates

Challenges – Design & Implementation

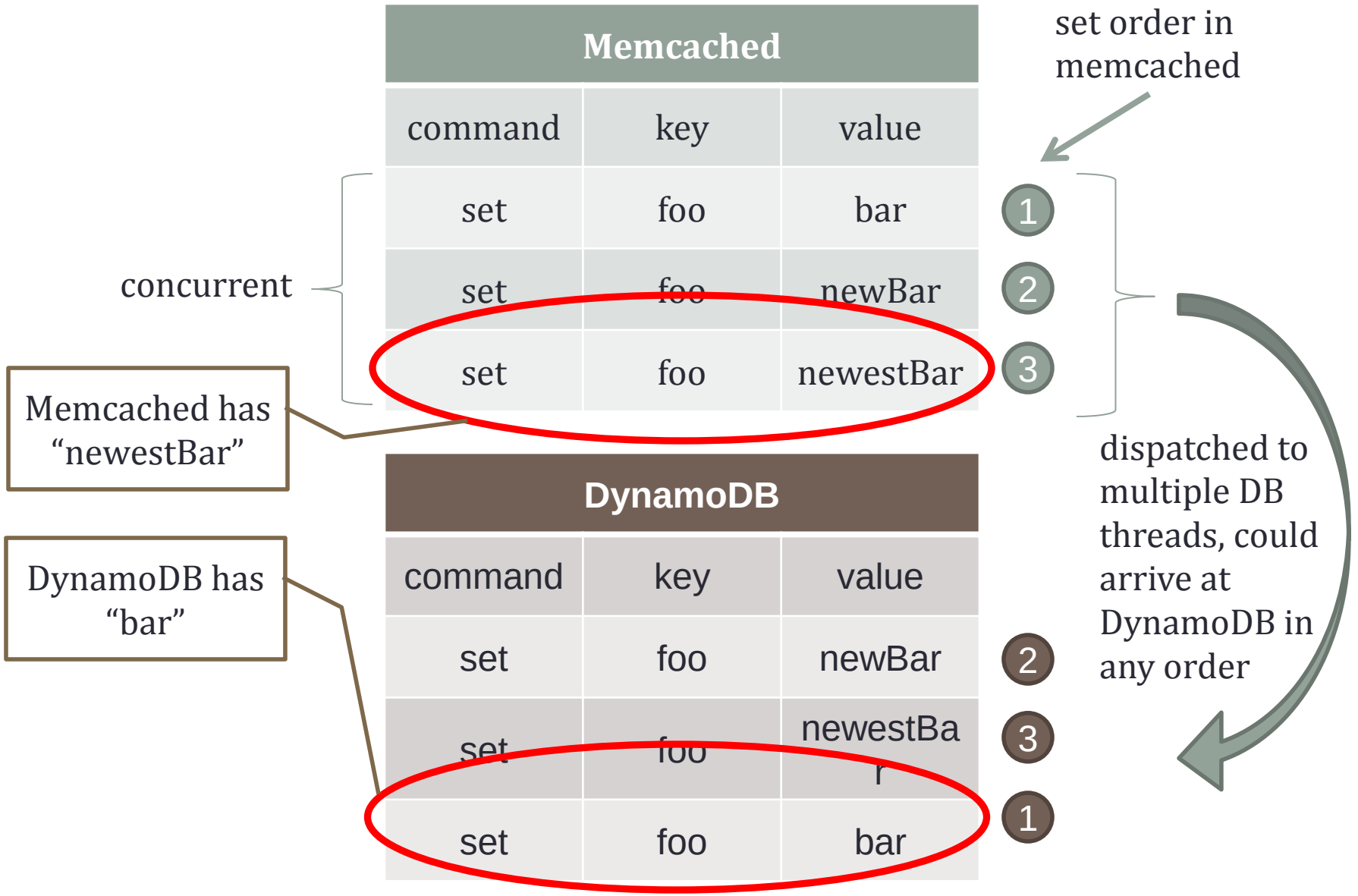
- Memcached is open-source, scalable caching engine written in C - unfortunately not documented very well
  - Libevent enables powerful and efficient connection management

- Major Issues:
  1. How to integrate DynamoDB backend without compromising existing memcached performance (using libevent)
     - Current solution: A second thread pool for database operations
  2. No existing C Client for DynamoDB
     - Current solution: Custom C wrapper around the C++ Client (in dev)
  3. Maintaining consistency across memcached engine and DynamoDB table (behavior with concurrent sets on same key)
     - Current solution: Additional counter hash table to keep track of item updates

Consistency Problem

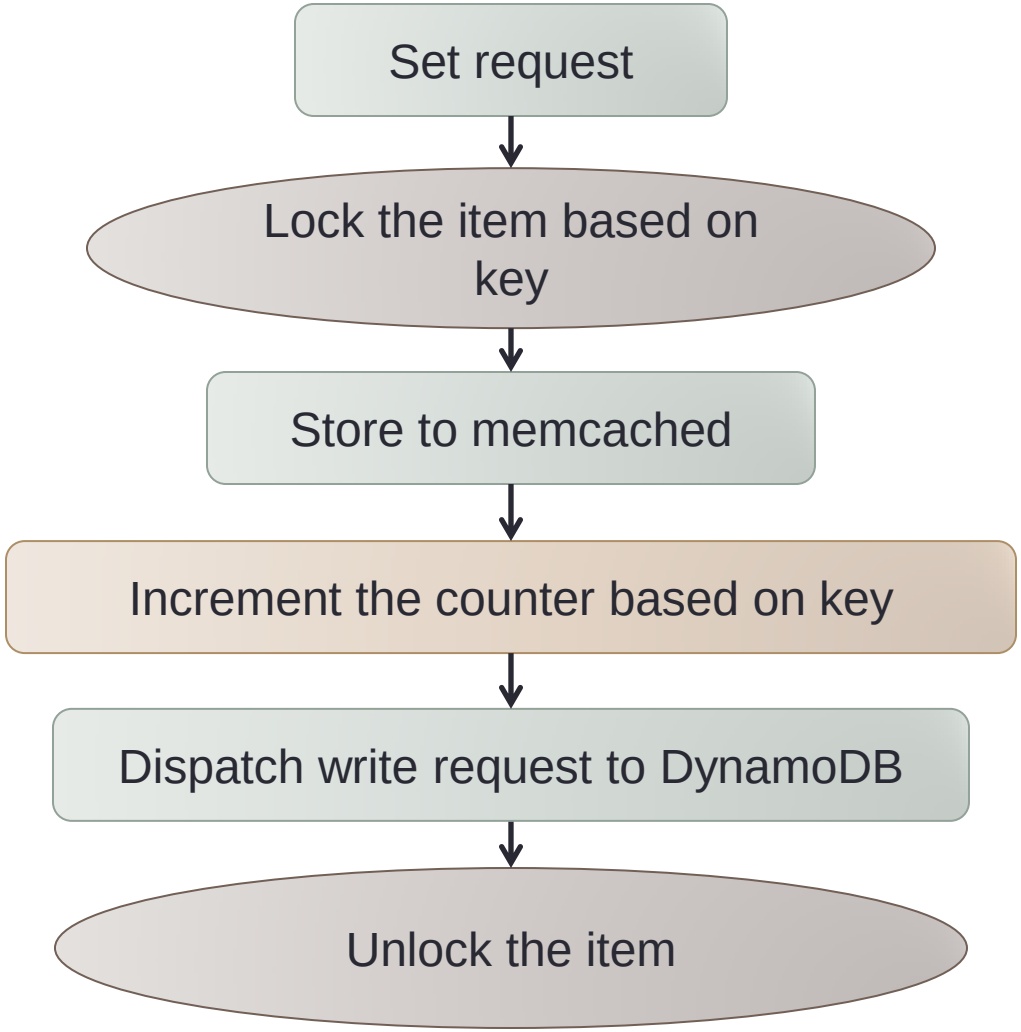


Consistency Problem

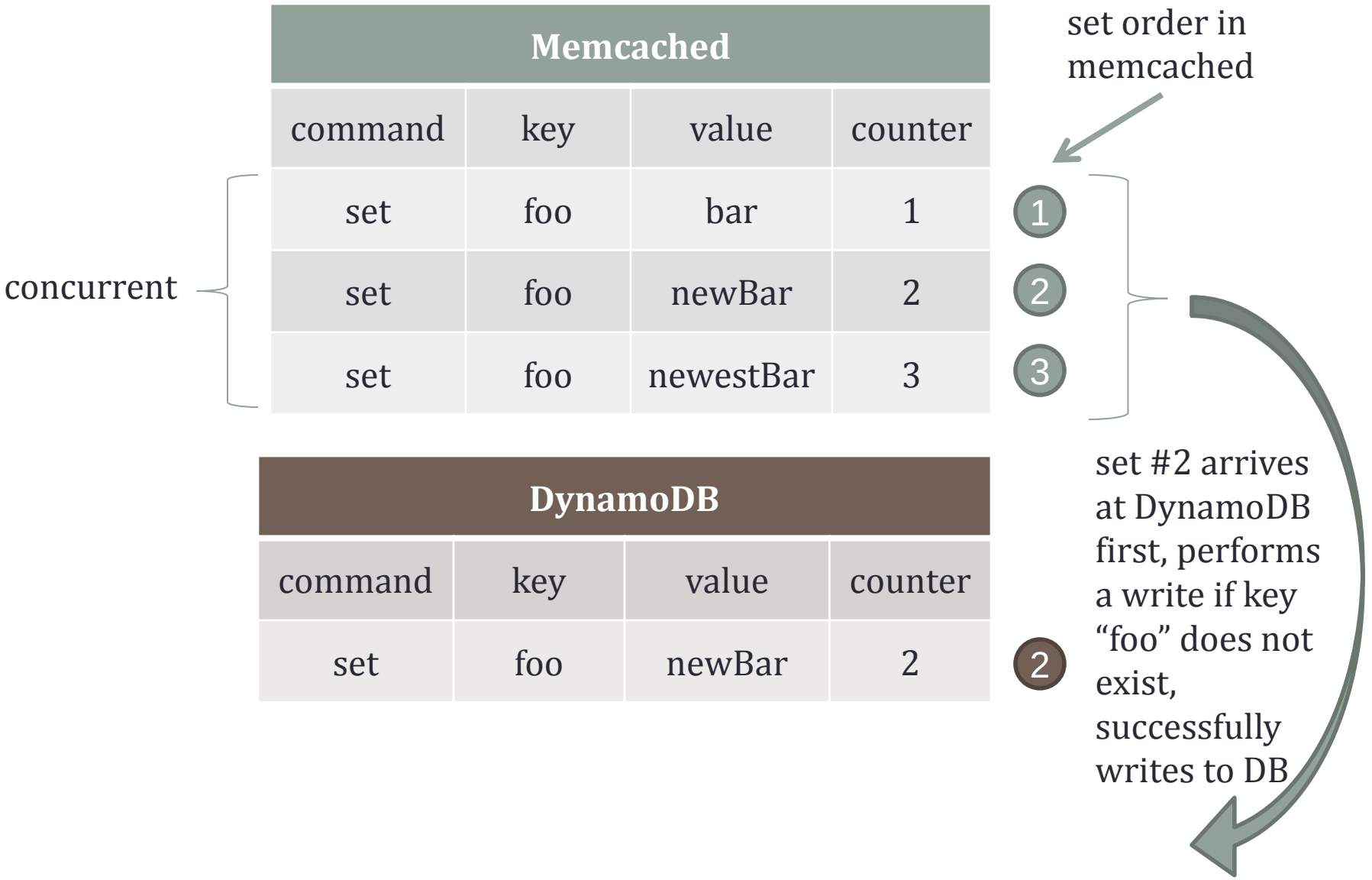


Consistency Solution (Naïve)

- Proposal: keep a counter value per key in a global table



Consistency Problem Revisited



Consistency Problem Revisited

Memcached			
command	key	value	counter
set	foo	bar	1
set	foo	newBar	2
set	foo	newestBar	3

concurrent

set order in memcached

1

2

3

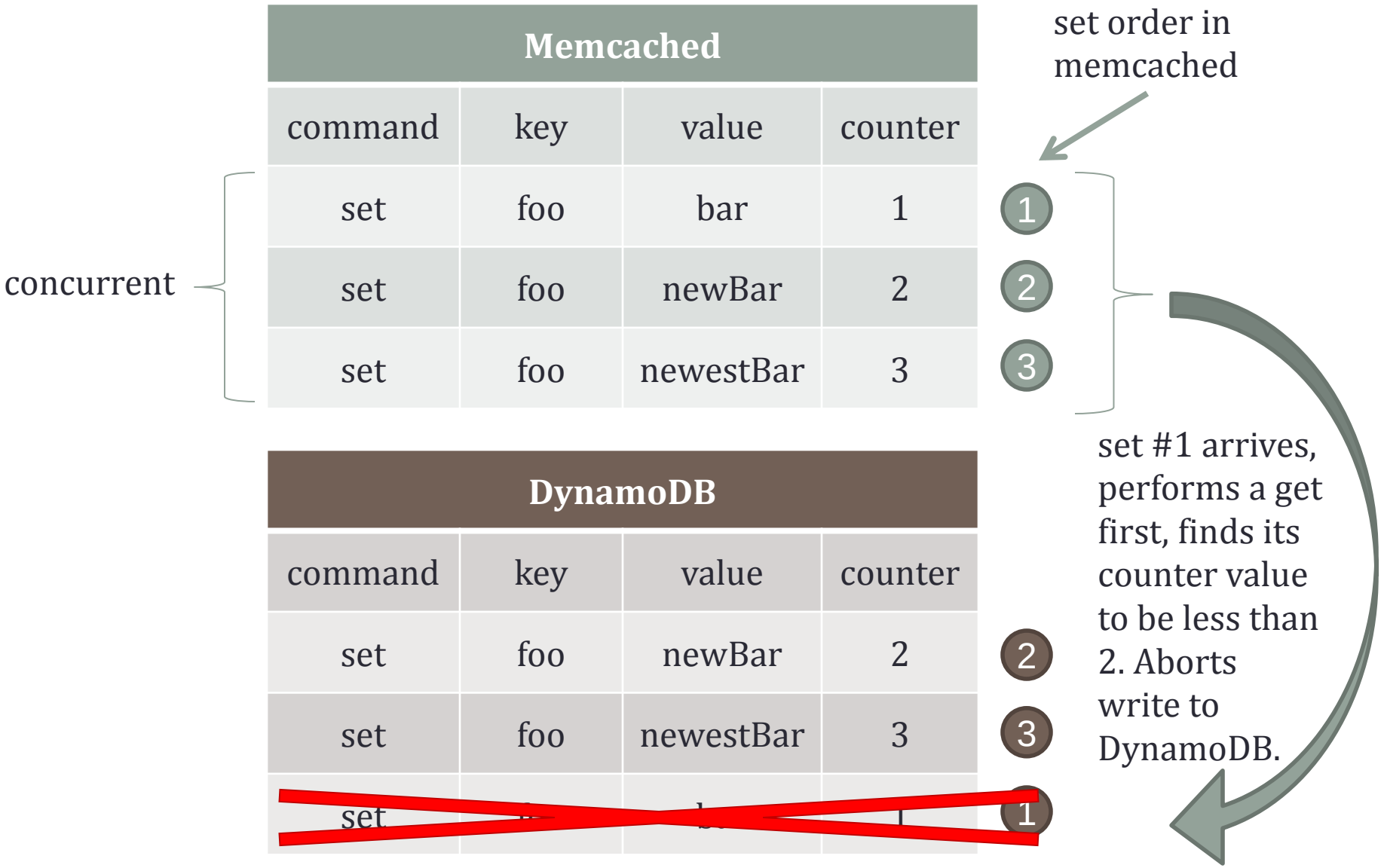
DynamoDB			
command	key	value	counter
set	foo	newBar	2
set	foo	newestBar	3

2

3

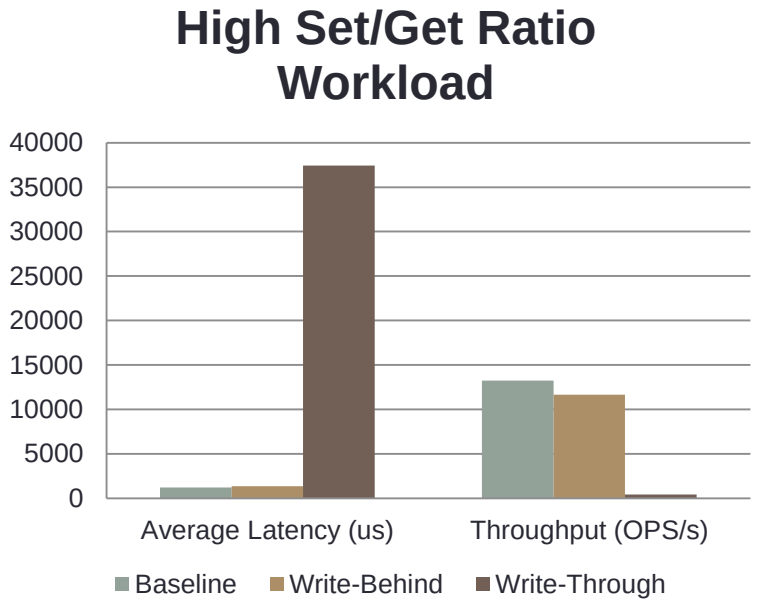
set #3 arrives, performs a get first, checks to see its own counter value is greater than 2. Performs a write if current DB value is still 2

Consistency Problem Revisited



Preliminary Performance Results

- Local testing using memaslap (memcached engine running on Developer Desktop, DynamoDB in Oregon)
- Different workloads representing different set/get ratios
  - In write-behind, all cases hit the DynamoDB write request limit (causing many failed sets), except for the low set/get ratio case



Future Work

Some critical items to turn the prototype into production:

- Extending the consistency solution (across memcached and DynamoDB table) to write-through scenario as well
- Optimizing the DynamoDB operations in memcached to reduce latency and increase throughput
- Design a highly concurrent C Client for DynamoDB
- Setting up proper credential management for memcached engine to access DynamoDB tables

Open Questions

- Intern project invokes product definition questions:
  - Using DynamoDB as primary data storage, versus just as a backup
  - Cache misses could trigger “get” requests to DynamoDB
  - At startup, ElastiCache node could be warmed up by existing DynamoDB table(s)
- Backend configuration (write-behind versus write-through)
  - Write-Behind: asynchronous data reads and writes to DynamoDB
  - Write-Through: synchronous reads and writes – more persistence!
    - Key/value not stored in memcached until write to DynamoDB succeeds
- Default behavior when an asynchronous DynamoDB requests fail (remove from memcached as well?)
- Is DynamoDB the correct backend support for ElastiCache? Currently there is a 64KB data limit, memcached’s limit is 1MB.

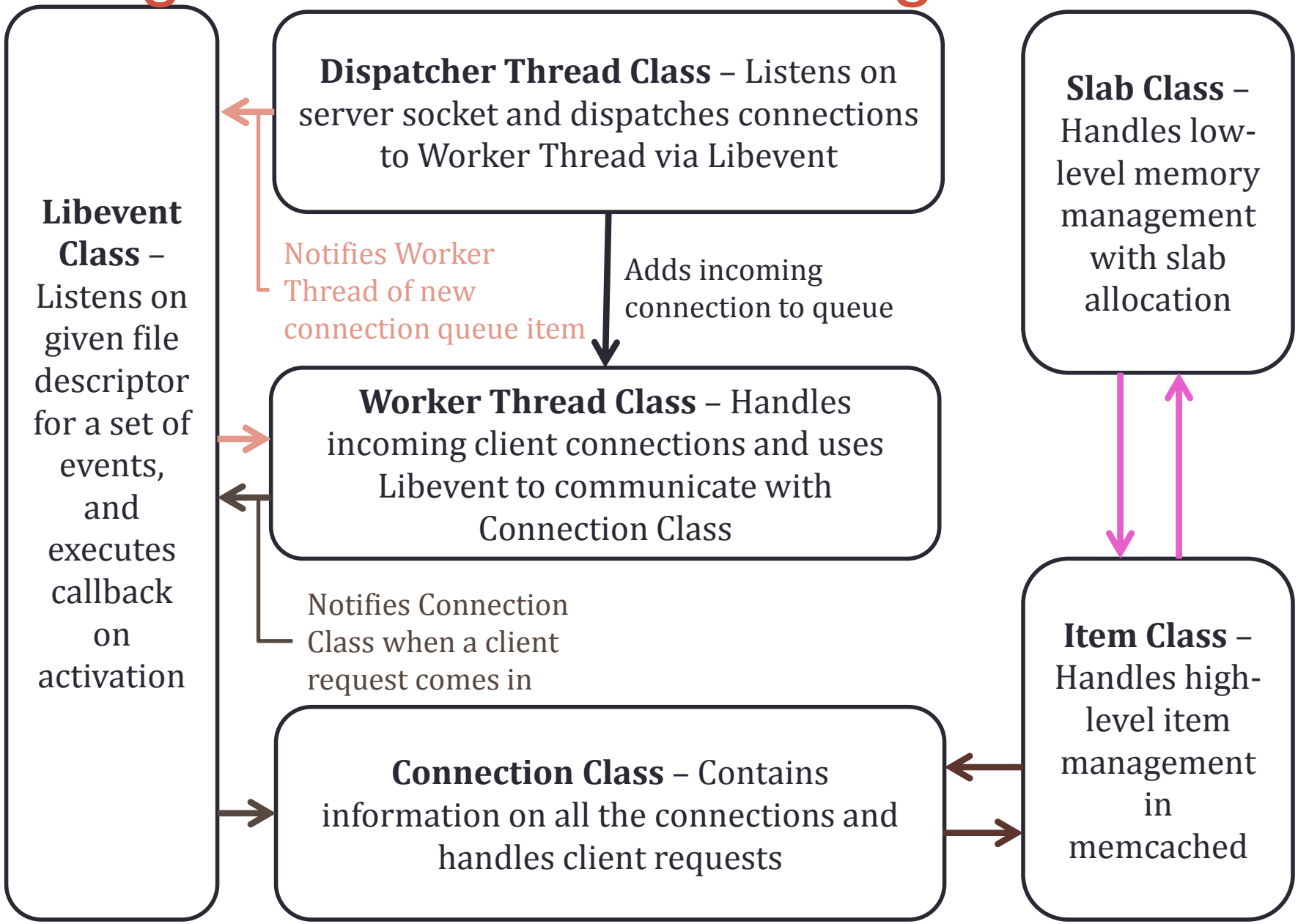
Appendix

Preliminary Performance Numbers

Low Set/Get Ratio Scenario			
	Average Latency (us)	Standard Deviation (us)	Throughput (OPS/s)
Baseline	929	2081.81	17191
Write-Behind	1081	11612.8	14782
Write-Through	4239	18704.88	3773
High Set/Get Ratio Workload			
	Average Latency (us)	Standard Deviation (us)	Throughput (OPS/s)
Baseline	1206	1156.5	13244
Write-Behind	1371*	16804.64*	11654*
Write-Through	37424	40095.22	428

*Error rate is very high (~88%) (either because DynamoDB task queues are too full, or memcached is out of memory slabs

Original Memcached Design



Extended Memcached Design