Revisiting Concurrency in High-Performance NoSQL Databases

Yuvraj Patel[1], Mohit Verma[2],
Andrea C. Arpaci-Dusseau[1], Remzi H. Arpaci-Dusseau[1]
University of Wisconsin-Madison[1], NVIDIA[2]



Technological changes

Faster storage devices

	DRAM	**NVM**	**SSD**	**HDD**
Read Latency	100-200 ns	500ns-2us	20-50 us	5-10ms
Write Bandwidth	20-50 GB/sec	5-10 GB/s per die	200-500 MB/s	50-100 MB/sec

More processing power available per node

Year	2006	2010	2014	2019
# of cores	2	8	32-64	128

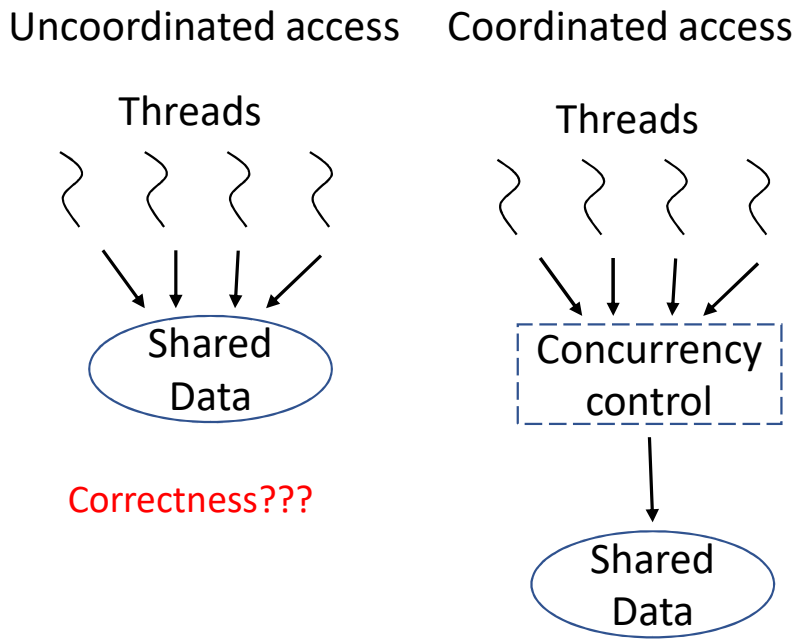
Concurrency control

Concurrency control

- Act of coordinating concurrent accesses

Key to scaling

- Decades worth of effort
  - Optimistic concurrency control
  - Adaptive concurrency control
  - Many other ways…..



Key Question

Concurrency control designed based on older technology

- Very few cores
- Hard disk

Is concurrency control a bottleneck?

- Can current techniques work well with newer technological changes?
- Is it time to revisit concurrency control?

Answer – Quantitative study

We study performance scaling of 5 popular systems

- MongoDB, Cassandra, CouchDB, ArangoDB, Oracle NoSQL DB

Measure actual and zero-contention concurrency control(ZCCC) scaling performance on single node

Observation

- No system scales well
- Presence of faster storage device does not help
- Difference between ZCCC and Actual is more than 3X

Answer – Qualitative study

Qualitative study of techniques used by these systems

- Categorize to understand weaknesses
  - Thread Architecture, Batching, Granularity, Partitioning, Scheduling, Low-level efficiency

Observations

- Batching, Thread Architecture – Used by all
- Partitioning, Scheduling not used efficiently for scaling
- Scope for improvement to optimize common cases

Our Solution

We present Xyza – an extension of MongoDB

- Concentrate on Partitioning, Scheduling & Low-level efficiency
- Per client and key-space partitioning
- Two novel scheduling techniques
  - Contention-aware scheduling & Semantic-aware scheduling
- Optimize common cases using atomics

Performance

- 2X to 3X faster than MongoDB
- Xyza single instance performance 0.8X-0.9X that of ZCCC performance

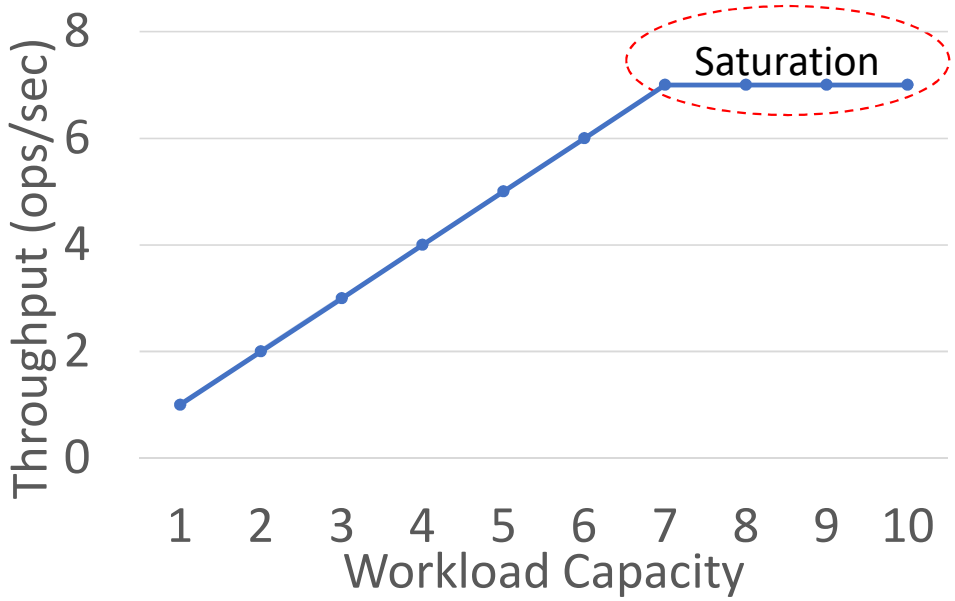
Outline

- Overview
- Concurrency Analysis
- Xyza Design
- Evaluation
- Conclusion

Zero-contention concurrency control(ZCCC)

Performance increases linearly as the workload increases

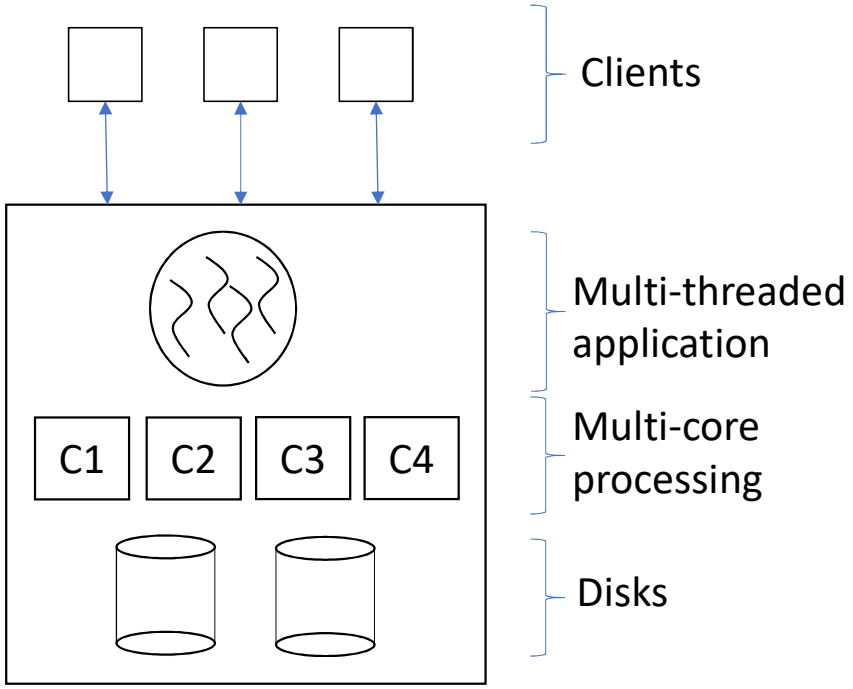
Saturates once the cpu/disk/network/memory exhausts



Concurrent application

Today's concurrent application architecture

- Clients connect to server
- Application – multithreaded
- Multi/Many processing cores
- One or more disks
- Memory



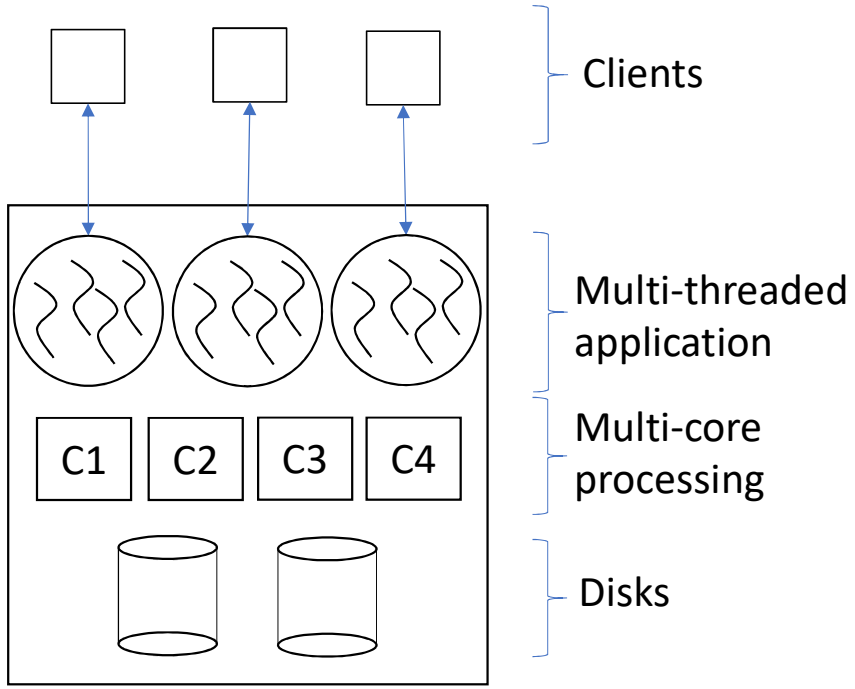
Identify zero-contention concurrency control

For ZCCC,

- Hypothesis - Less contention of resources by perfectly partitioning

How to reduce contention of resources?

- Instantiate multiple instances of the same application
- CPU/disk/memory/network still stressed



Actual vs ZCCC

Why writes?

- Modify global structures frequently
- Hard to scale

Why weakest consistency option?

- Designed to achieve high performance

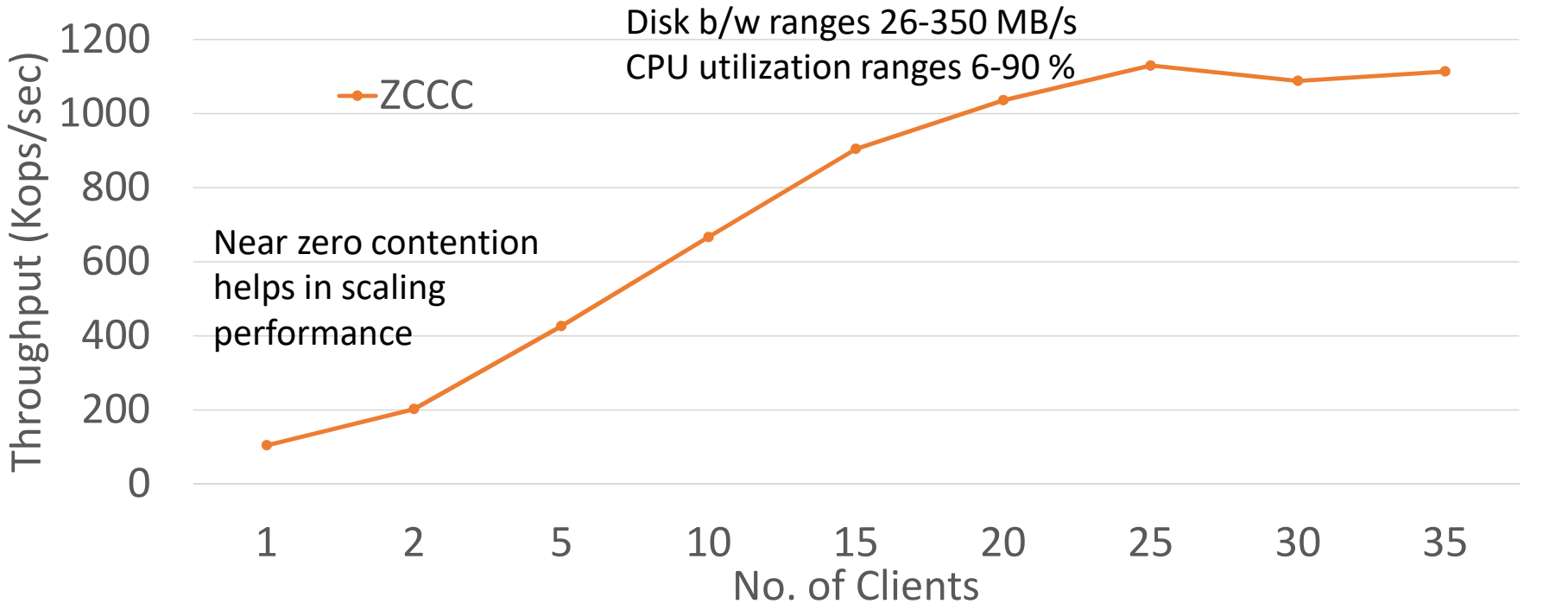
Workload

- 50M key-value pair inserts, value - 100 bytes

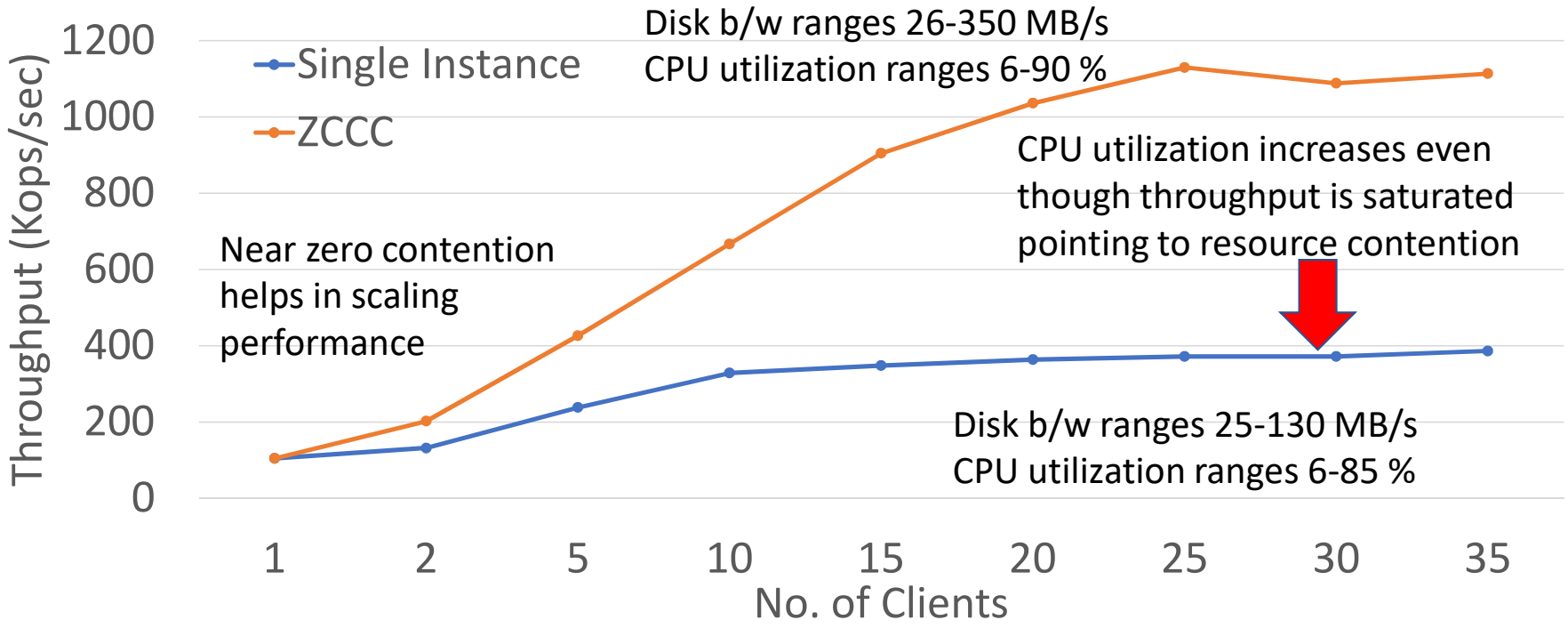
Setup

- 2 socket 8-core hyperthreaded, 1 480 GB SSD, 128 GB RAM, 10Gbps N/w

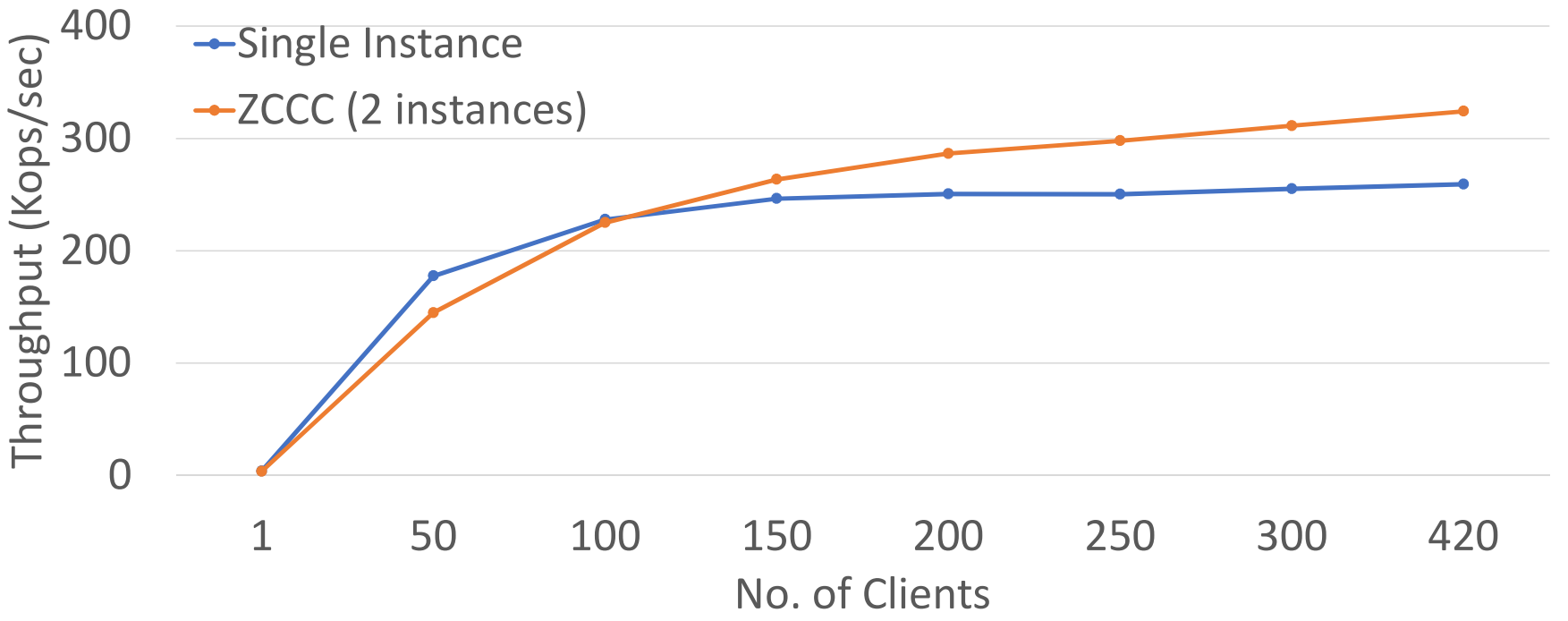
Results - MongoDB



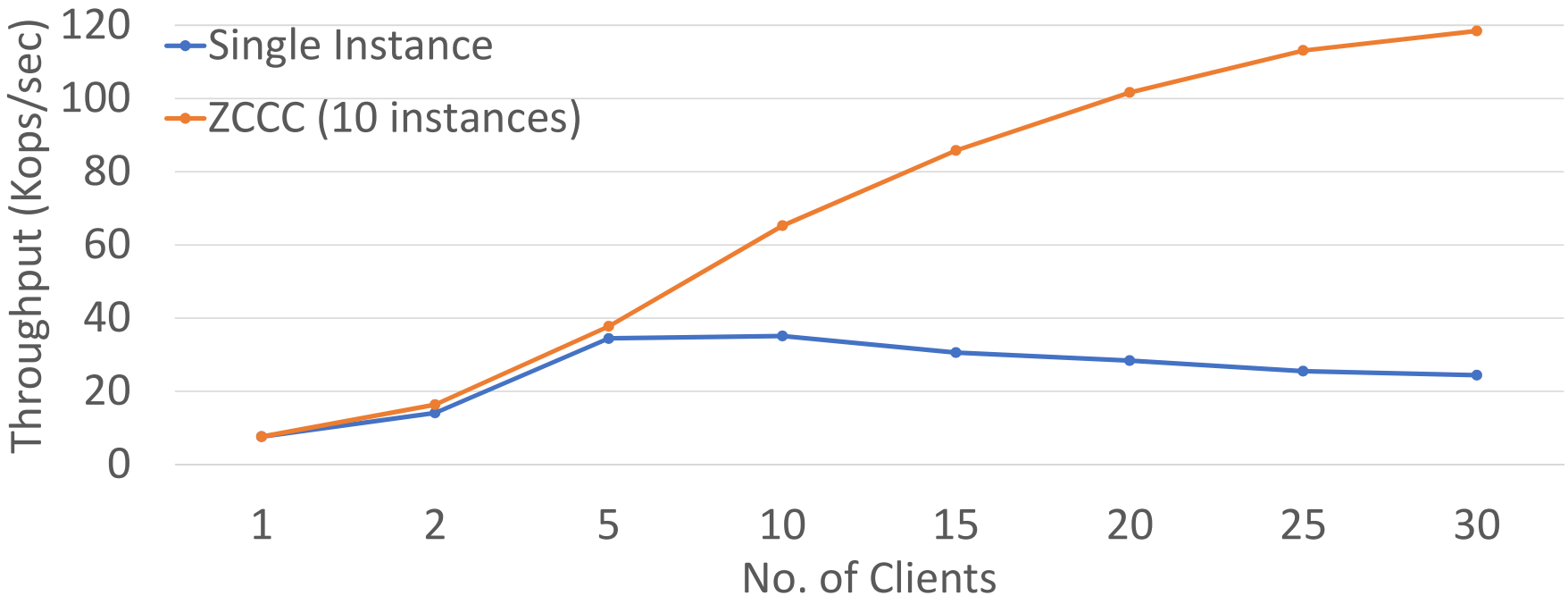
Results - MongoDB



Results – Cassandra



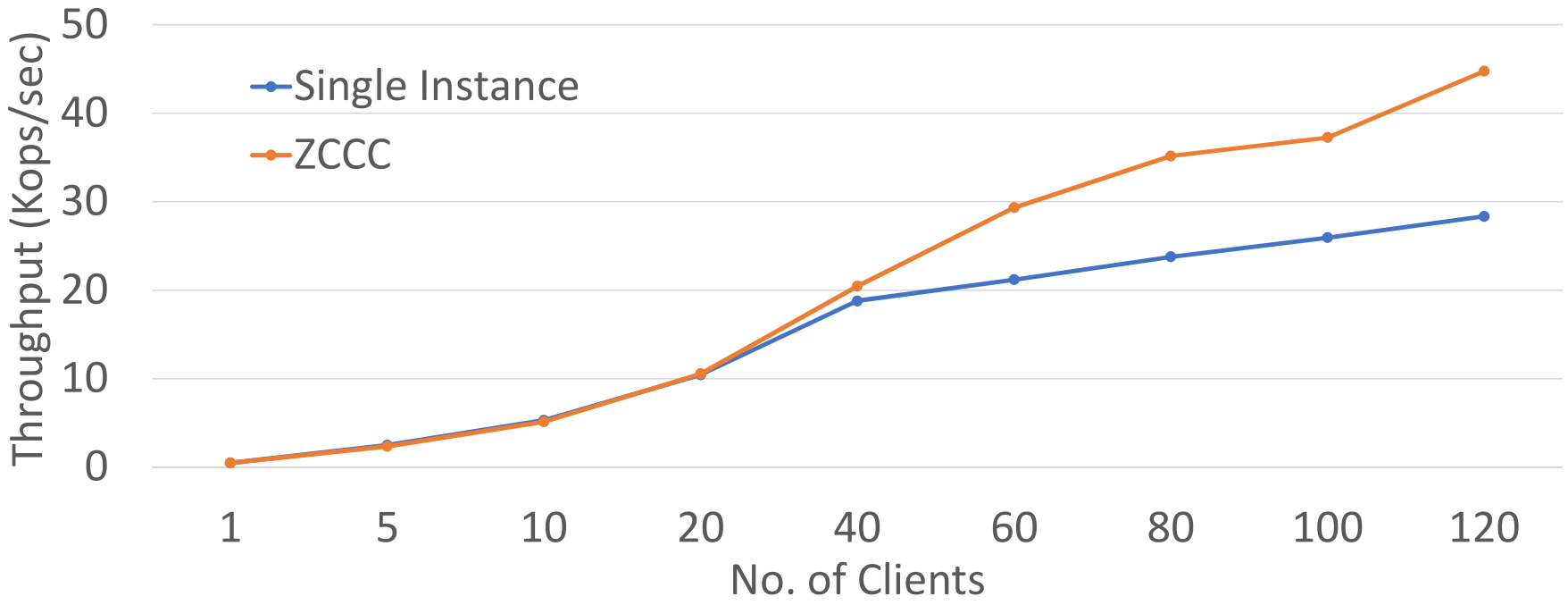
Results – Oracle NoSQL DB



Results – CouchDB



Results – ArangoDB



Categorization

Study 5 systems qualitatively

- Analyze code, design and architecture documents
- Categorize the techniques - Understand the weaknesses

6 categories

- Thread architecture, Batching, Granularity, Partitioning, Scheduling, Low-level efficiency (Read the paper)

Observation

- Partitioning and Scheduling not used efficiently
- Scope to improve common cases performance

Analysis - Summary

Quantitative study shows

- Resource competition kills actual performance
- ZCCC performance scales as hypothesized
- Storage and Network not a bottleneck

Qualitative study shows

- Partitioning & Scheduling not used efficiently
- There is a need to optimize for common cases

Systems designed for slow-storage media than faster devices

Outline

- Overview
- Concurrency Analysis
- Xyza Design
- Evaluation
- Conclusion

Xyza Design

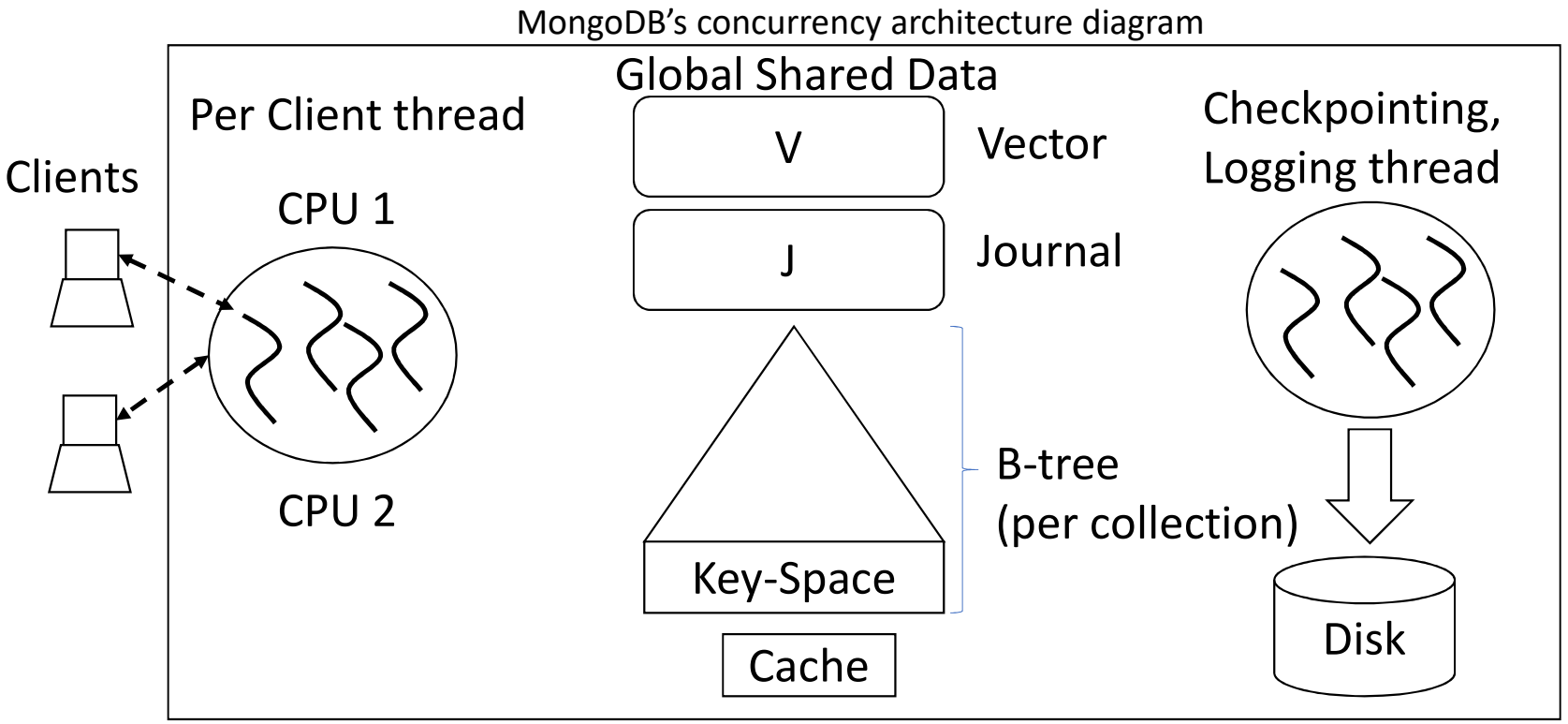
Xyza – extension of MongoDB

Why MongoDB

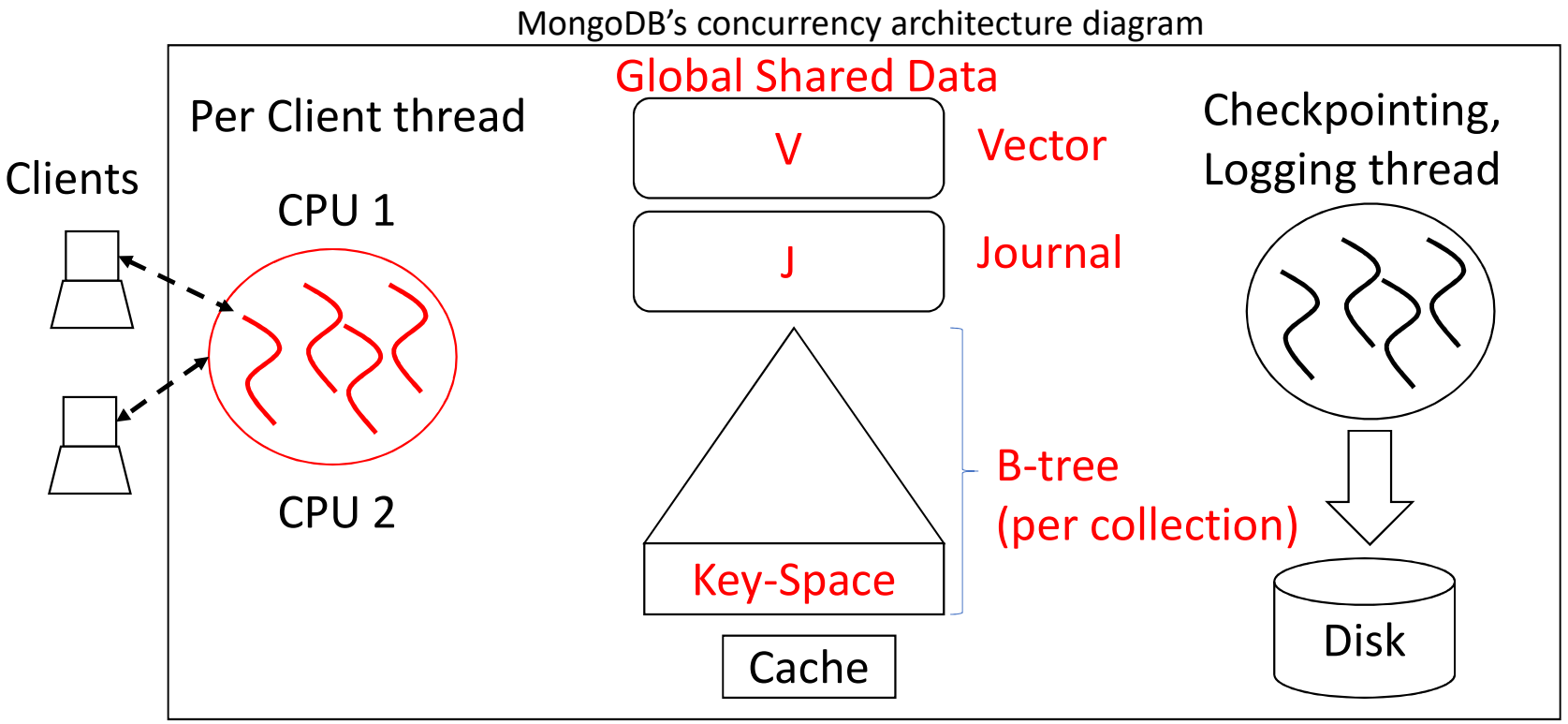
- Certain design ideas useful for Xyza
- Scope for improvements
- Popular and heavily used

Xyza design concentrated on partitioning, scheduling and low-level efficiency

MongoDB architecture

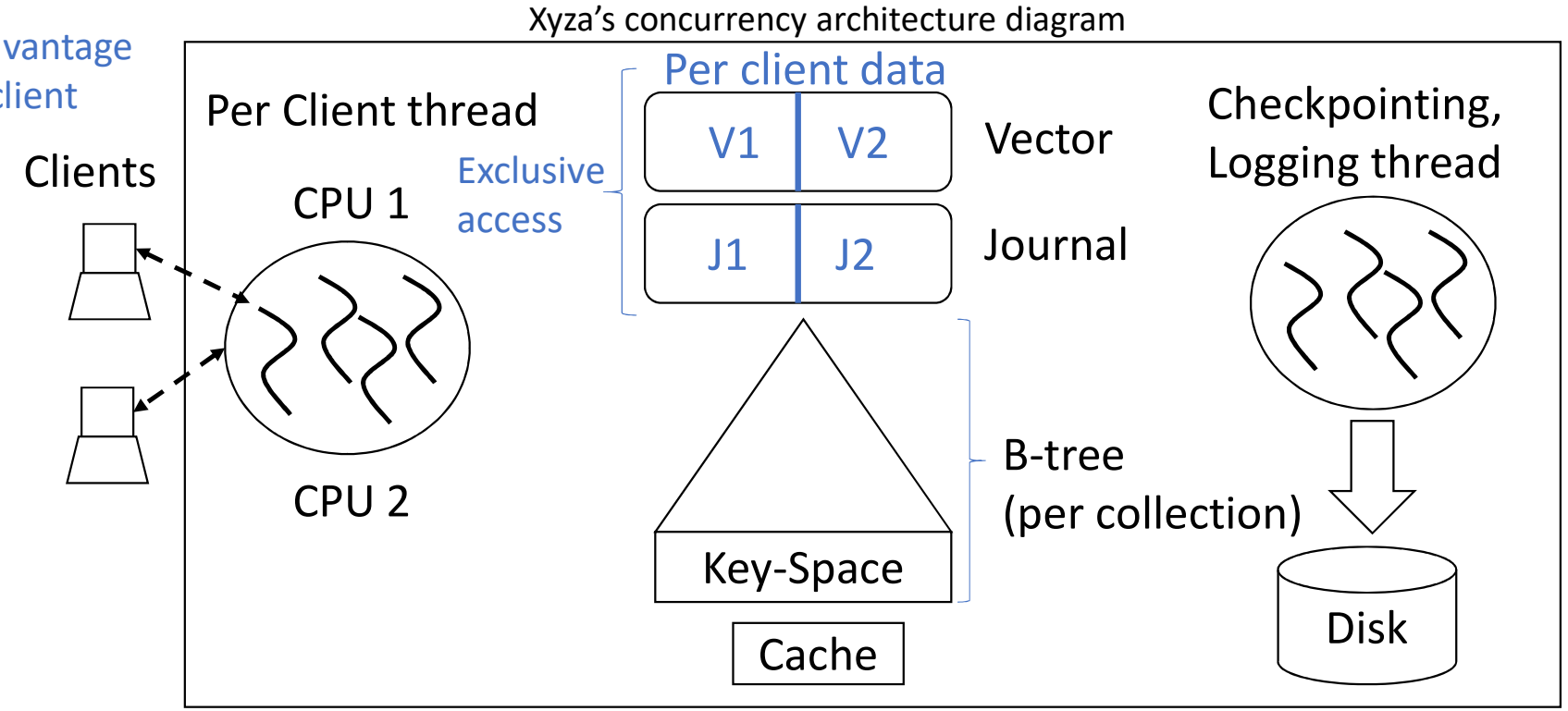


MongoDB architecture problems



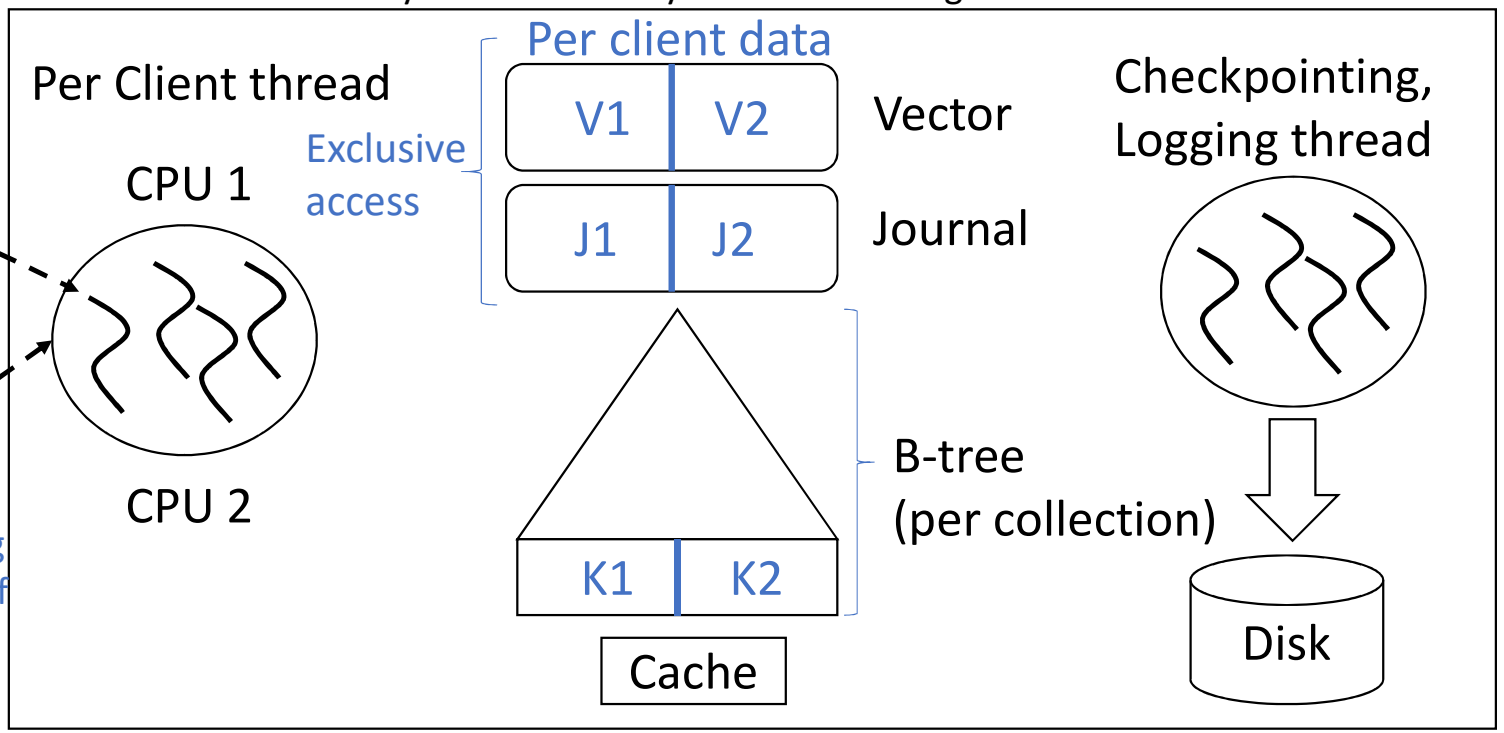
Xyza - Partitioning

- Take advantage of per client thread



Xyza - Partitioning

- Take advantage of per client thread

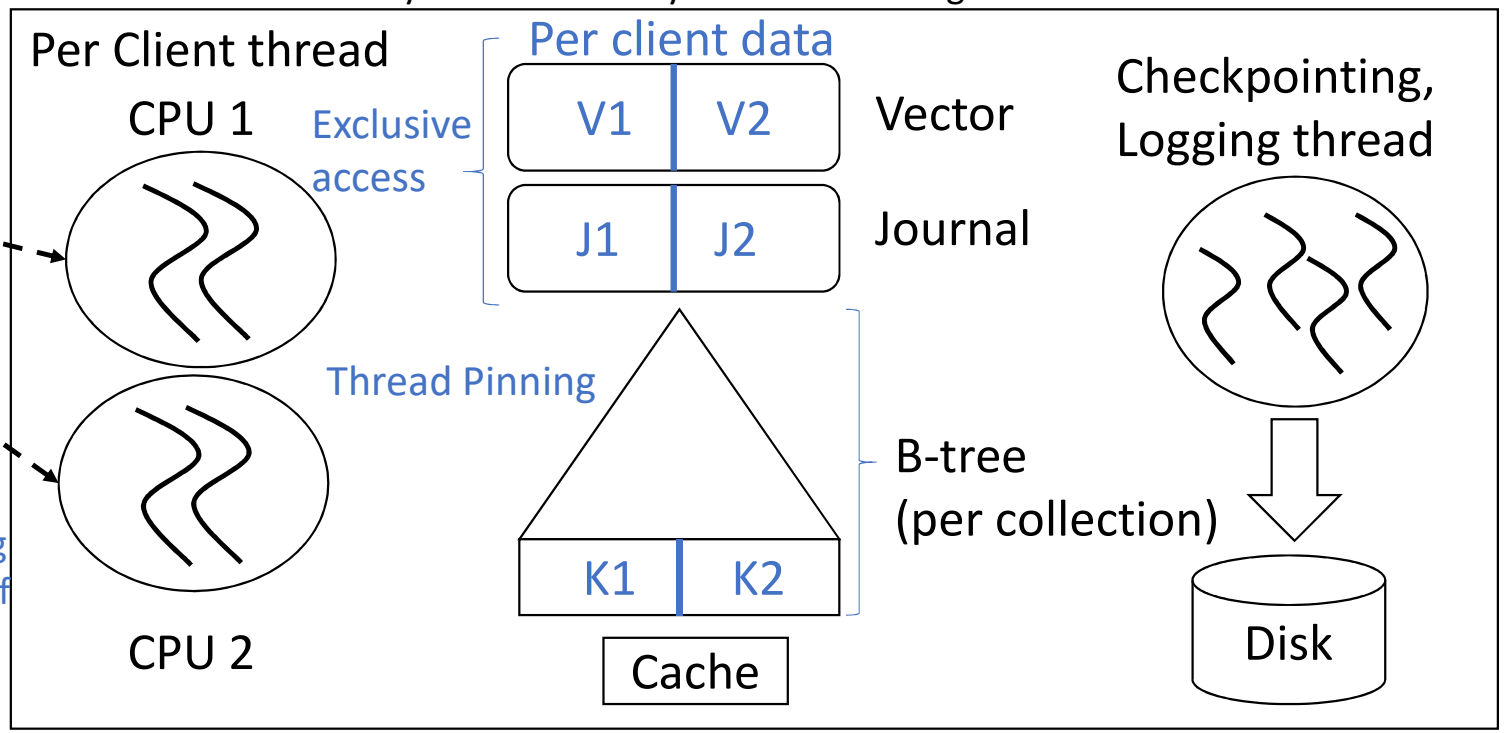


- Key-space partitioning facilitates execution of non-overlapping operations

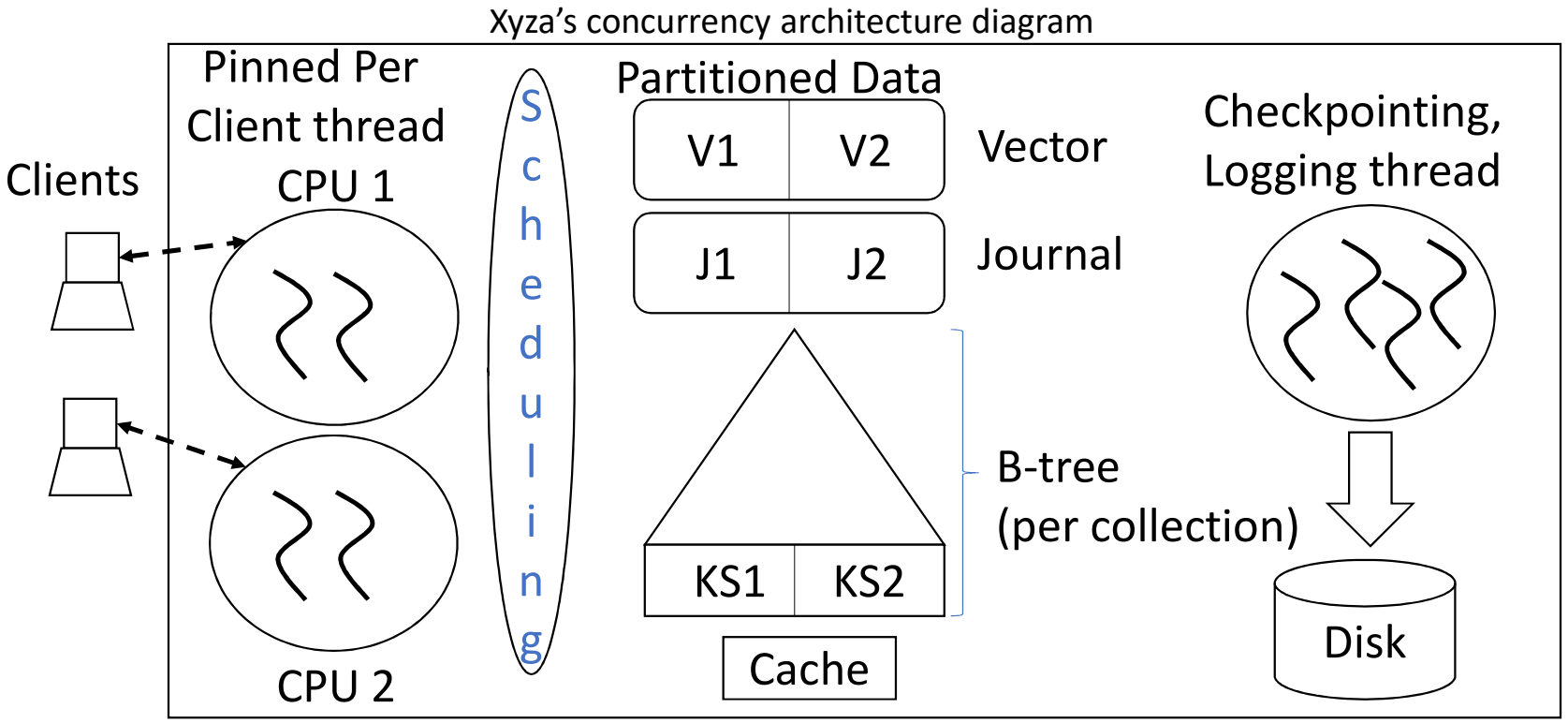
Xyza - Partitioning

- Take advantage of per client thread
- Key-space partitioning facilitates execution of non-overlapping operations

Xyza's concurrency architecture diagram



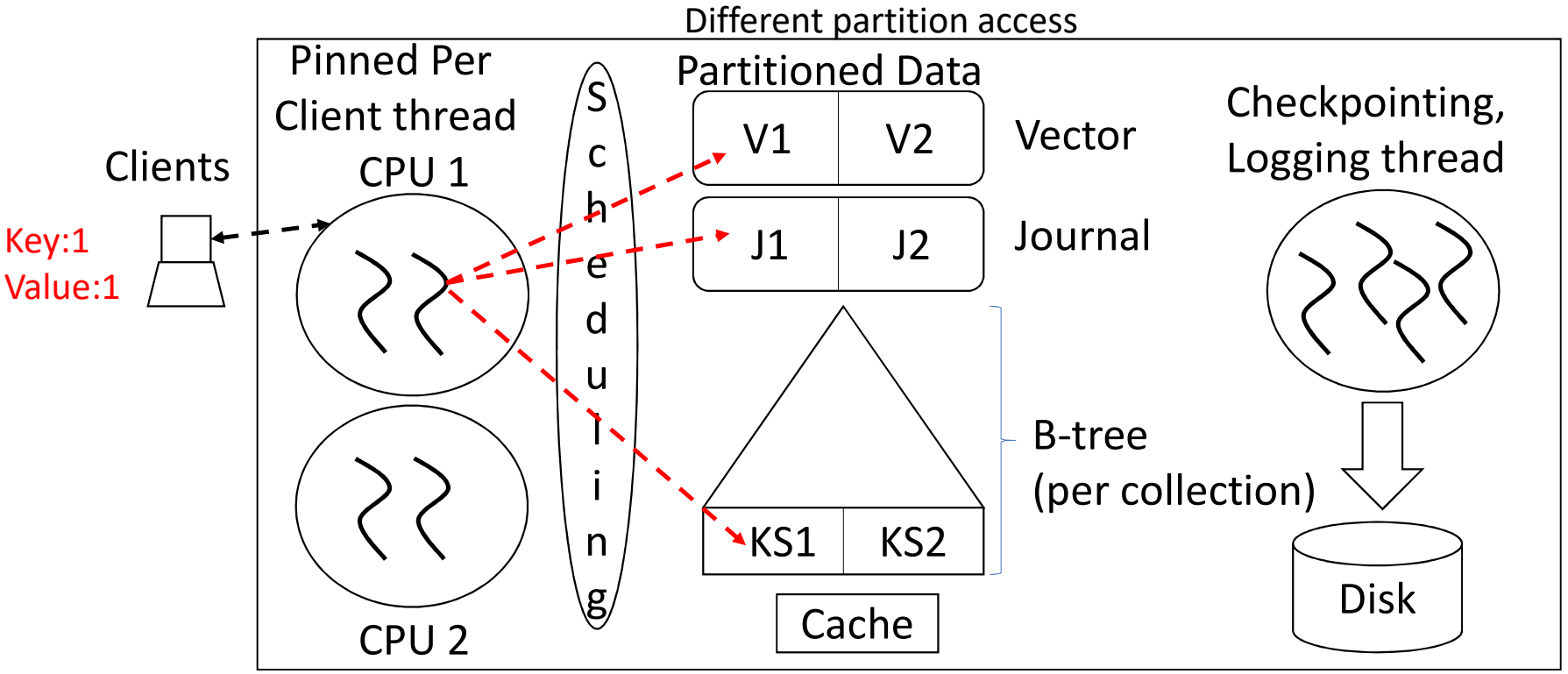
Xyza – Contention-aware scheduling



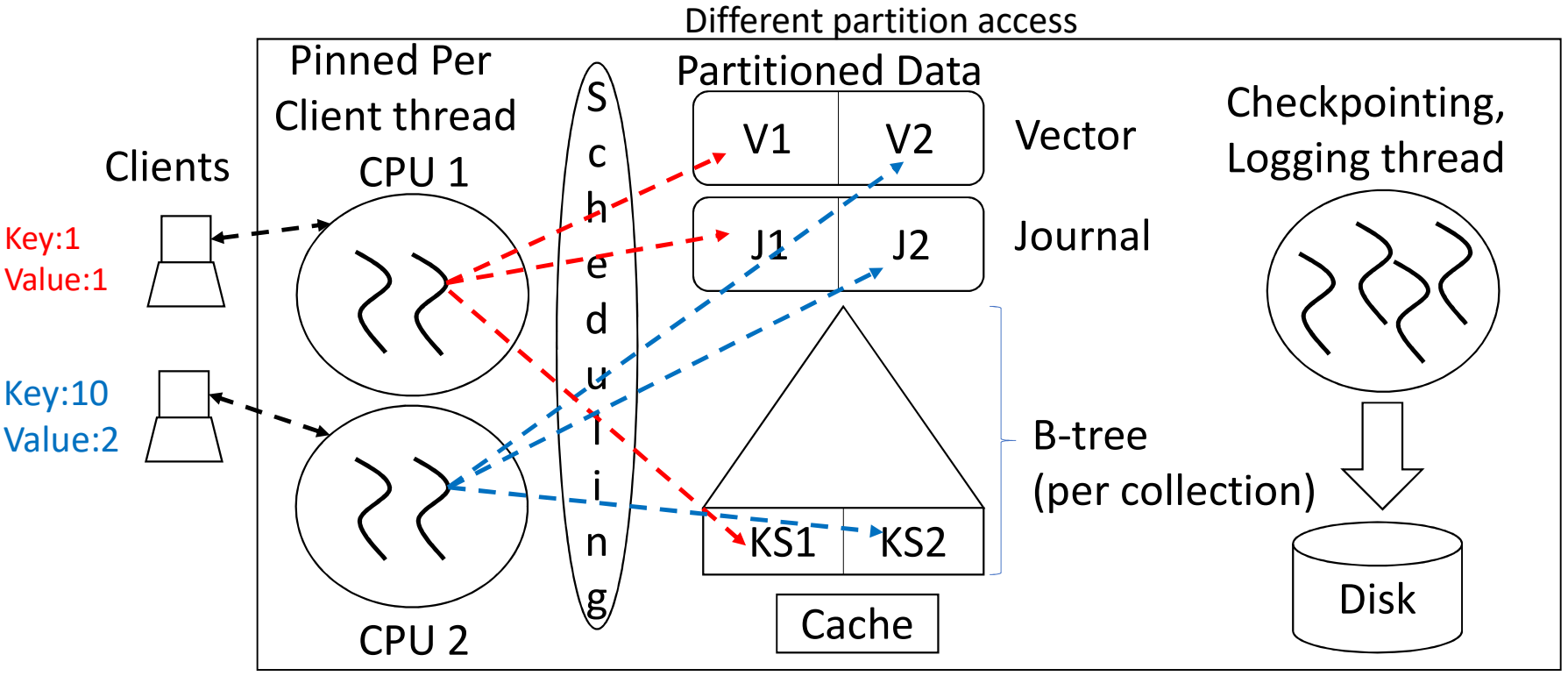
- Locks/partition as resource
- Exclusive access per key-space partition
- Execute operations that will not contend for same resources
- Overlapping operations wait

Scheduling - form of primitive synchronization

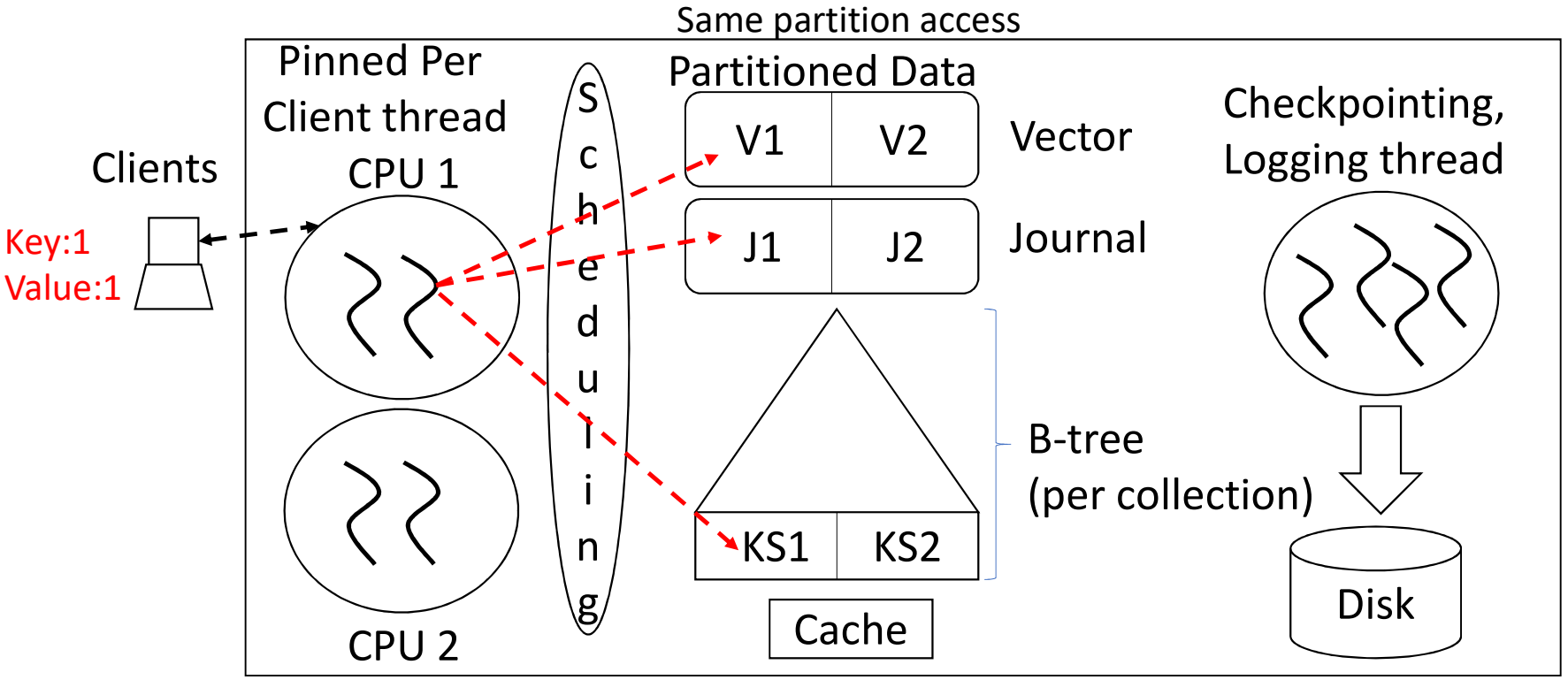
Xyza scheduling example – No conflict



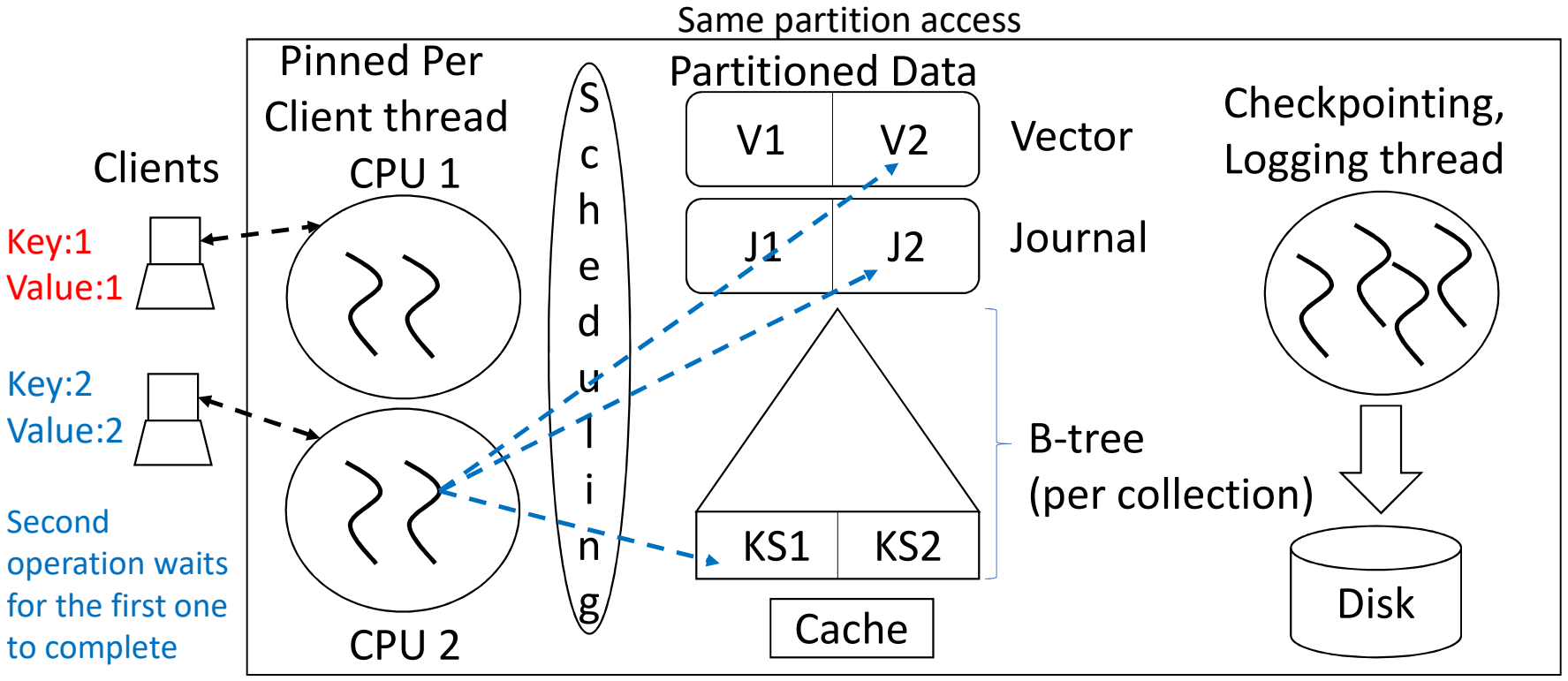
Xyza scheduling example – No conflict



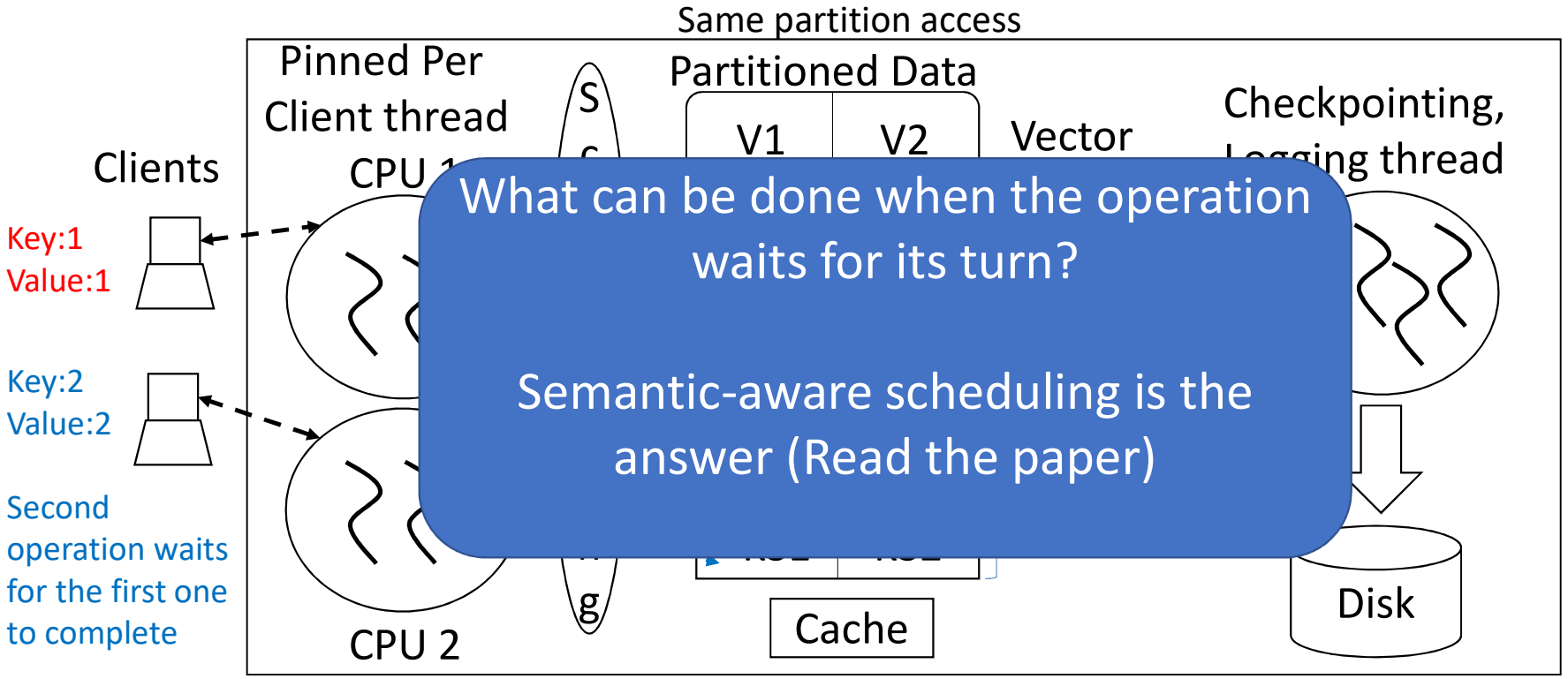
Xyza scheduling example - Conflict



Xyza scheduling example - Conflict



Xyza - Semantic-aware scheduling



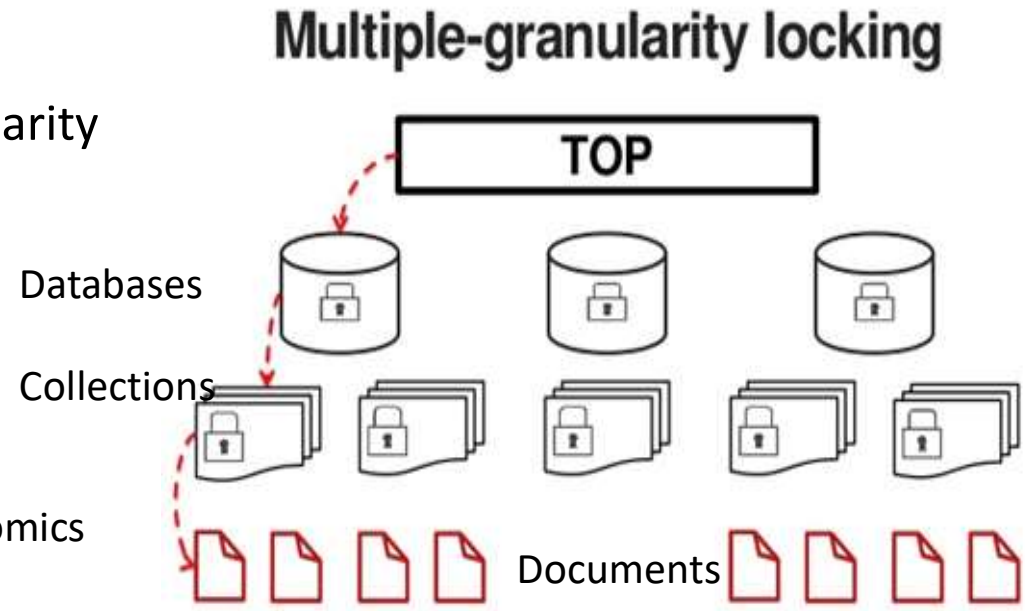
Xyza – Low-level efficiency

MongoDB

- Use atomics internally
- Lock manager for multiple-granularity locking

Xyza

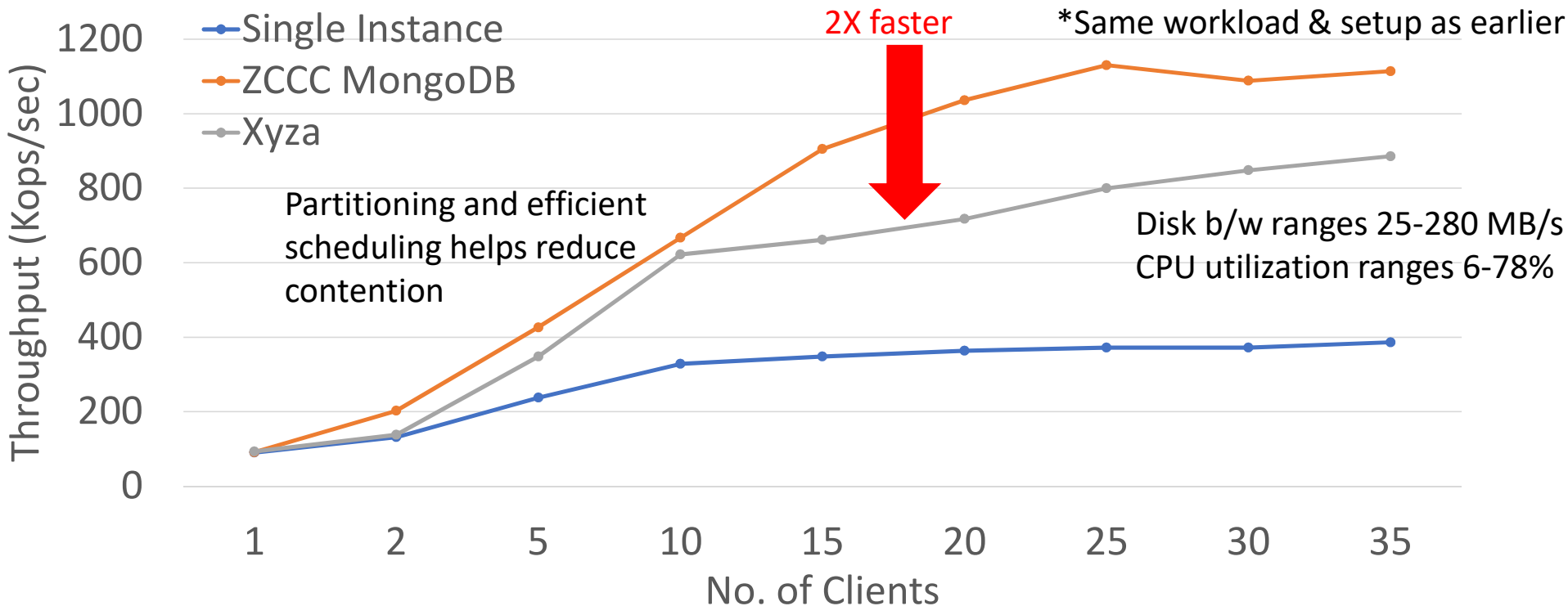
- Lock manager replaced
  - Simple wait-signal mechanism
- Optimize common case
  - Mutex locks heavy compared to atomics
  - Bypass mutex - Instead use atomics



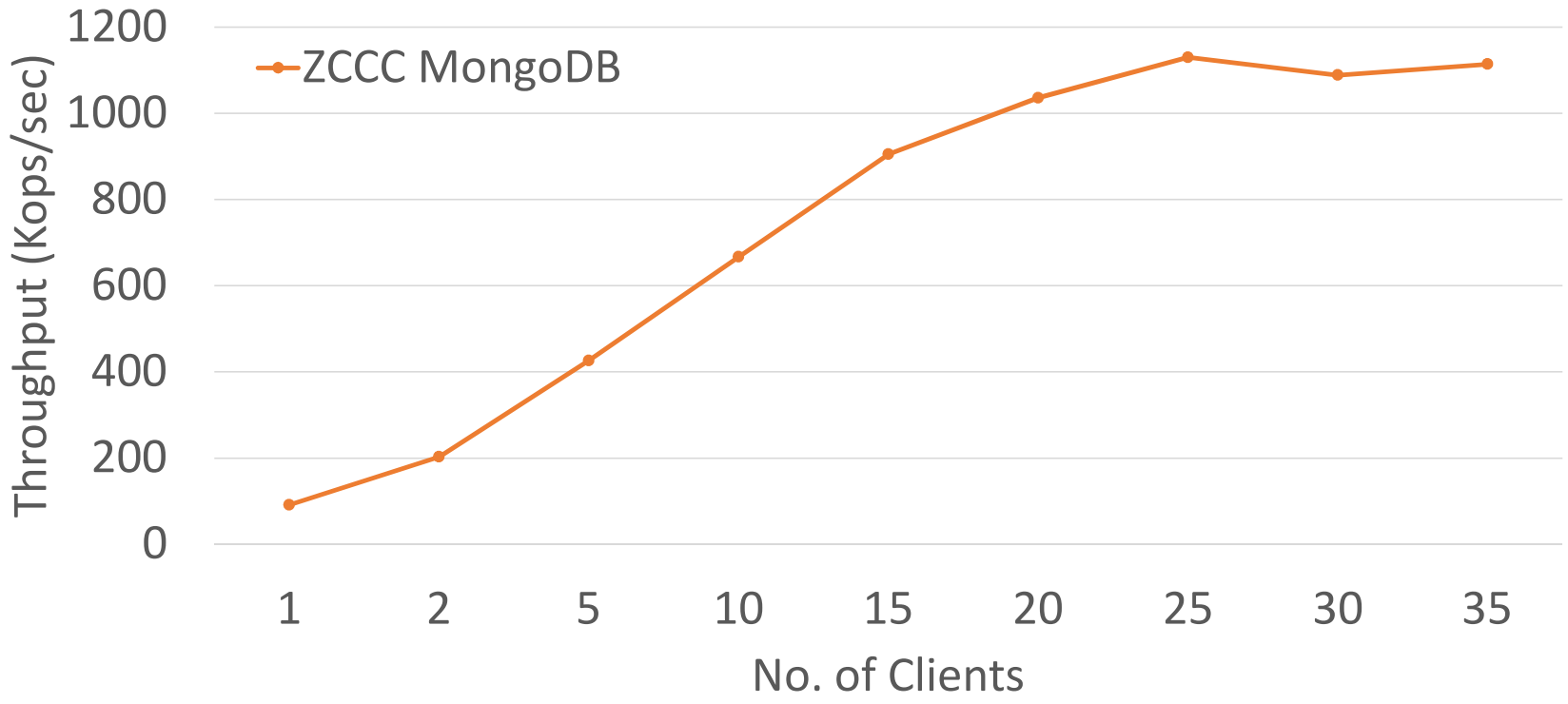
Outline

- Overview
- Concurrency Analysis
- Xyza Design
- Evaluation
- Conclusion

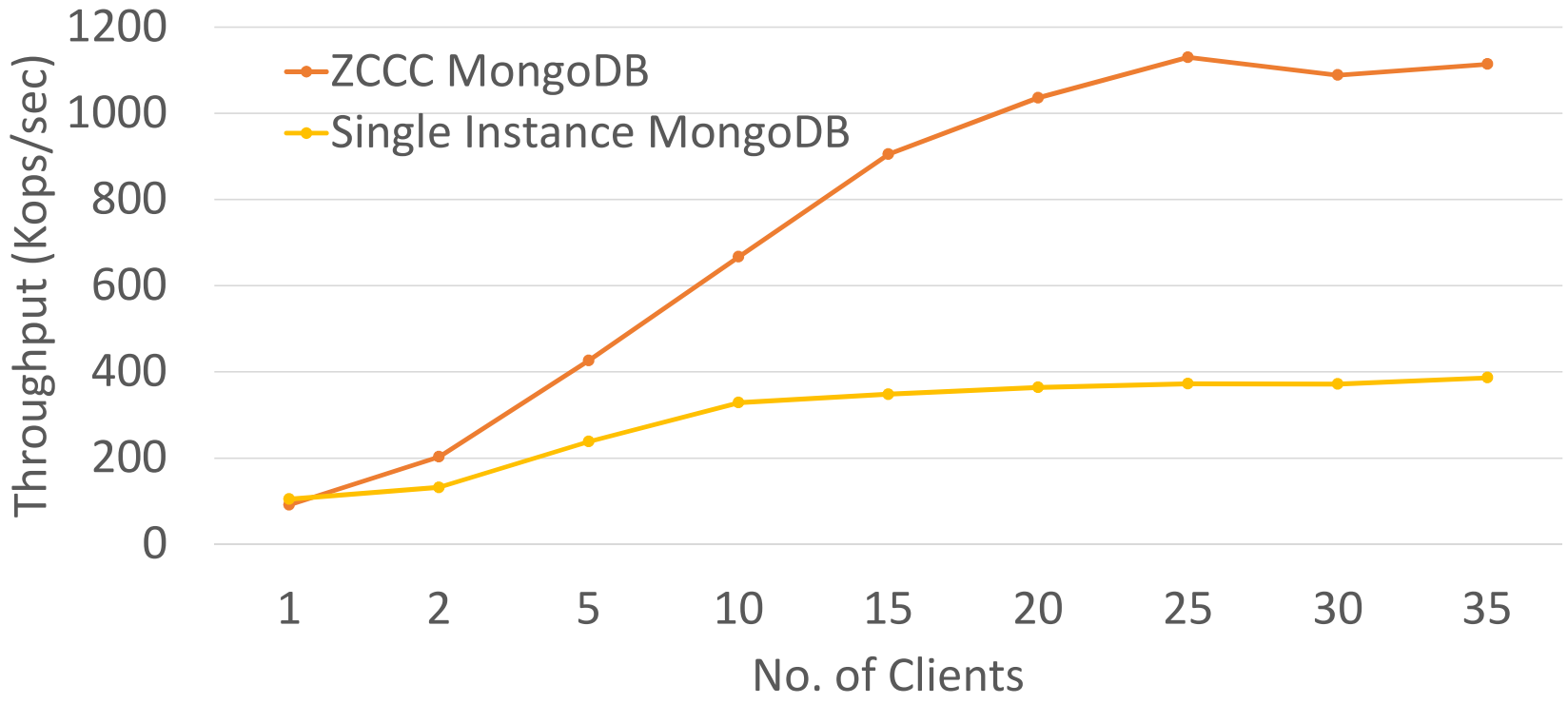
Evaluation


Single Instance
ZCCC MongoDB
Xyza
2X faster
*Same workload & setup as earlier
Partitioning and efficient scheduling helps reduce contention
Disk b/w ranges 25-280 MB/s
CPU utilization ranges 6-78%
Throughput (Kops/sec)
1200
1000
800
600
400
200
0
1
2
5
10
15
20
25
30
35
No. of Clients

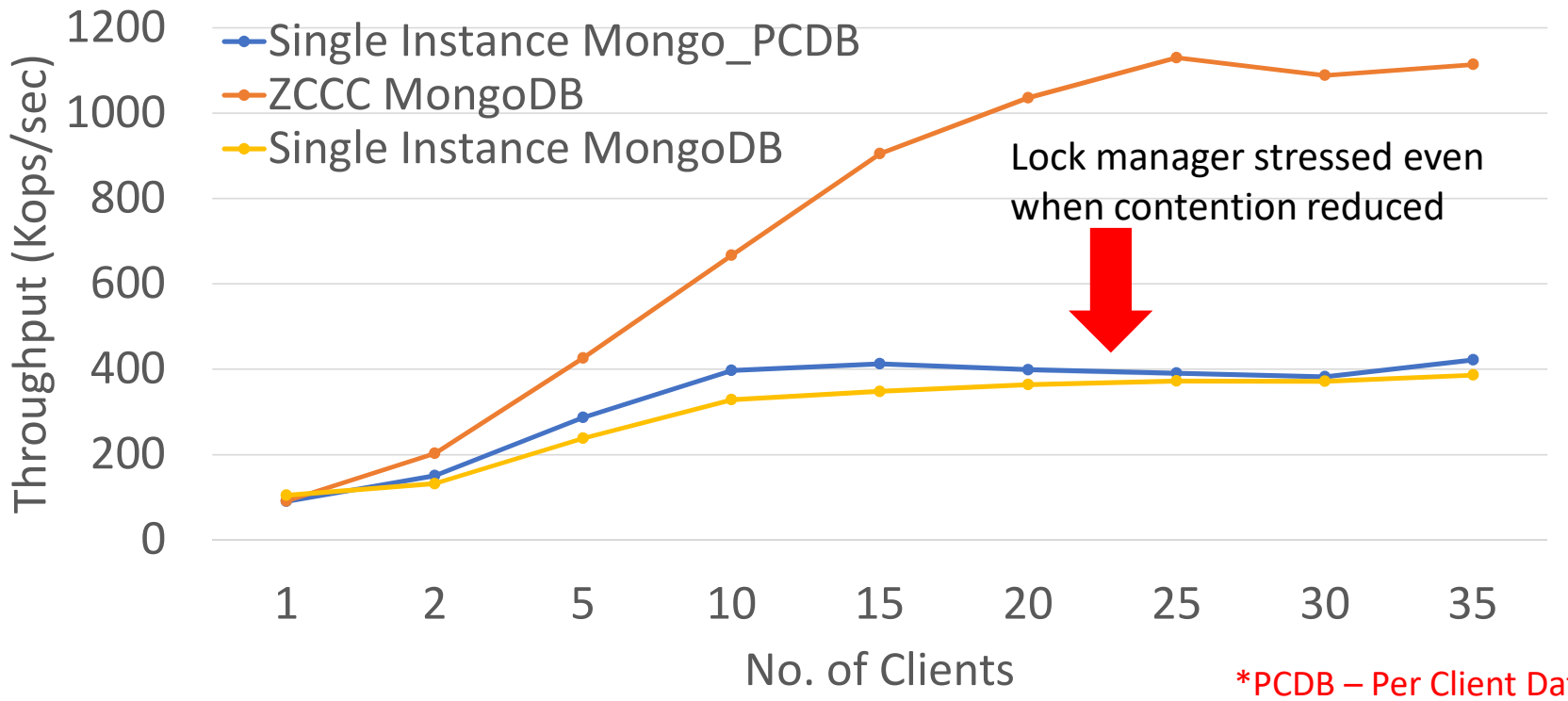
Evaluation – contention reduction



Evaluation – contention reduction

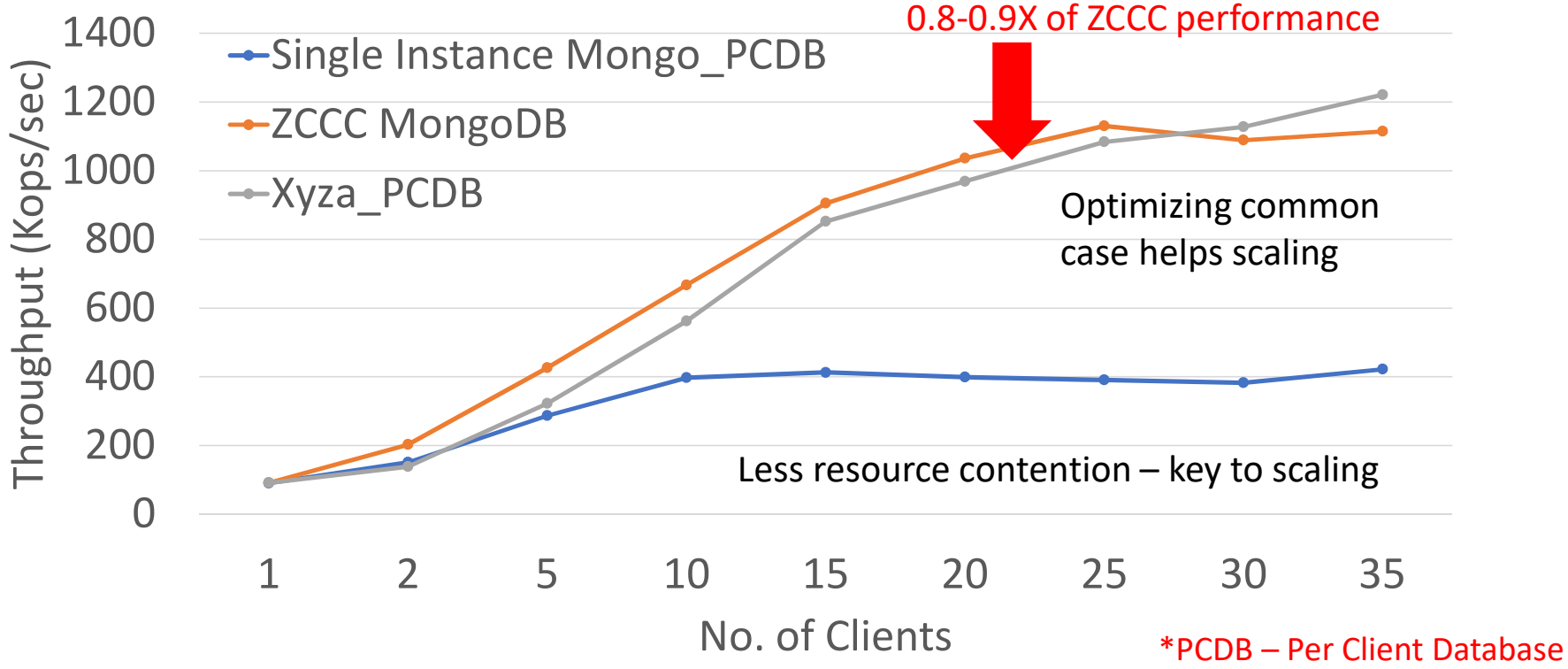


Evaluation – contention reduction



*PCDB – Per Client Database

Evaluation – contention reduction



Outline

- Overview
- Concurrency Analysis
- Xyza Design
- Evaluation
- Conclusion

Conclusion

Today's NoSQL DB's do not scale well despite the presence of faster storage medium and high processing capacity

We present Xyza that scales well and comes close to ZCCC performance

Time to revisit concurrency and consider aggressive concurrency mechanisms

The proposed techniques can be applied in many other systems

....WXYZAB....

Thank you