Memcached vs Redis

How does performance of *Memcached* compare to *Redis* on a common feature set?

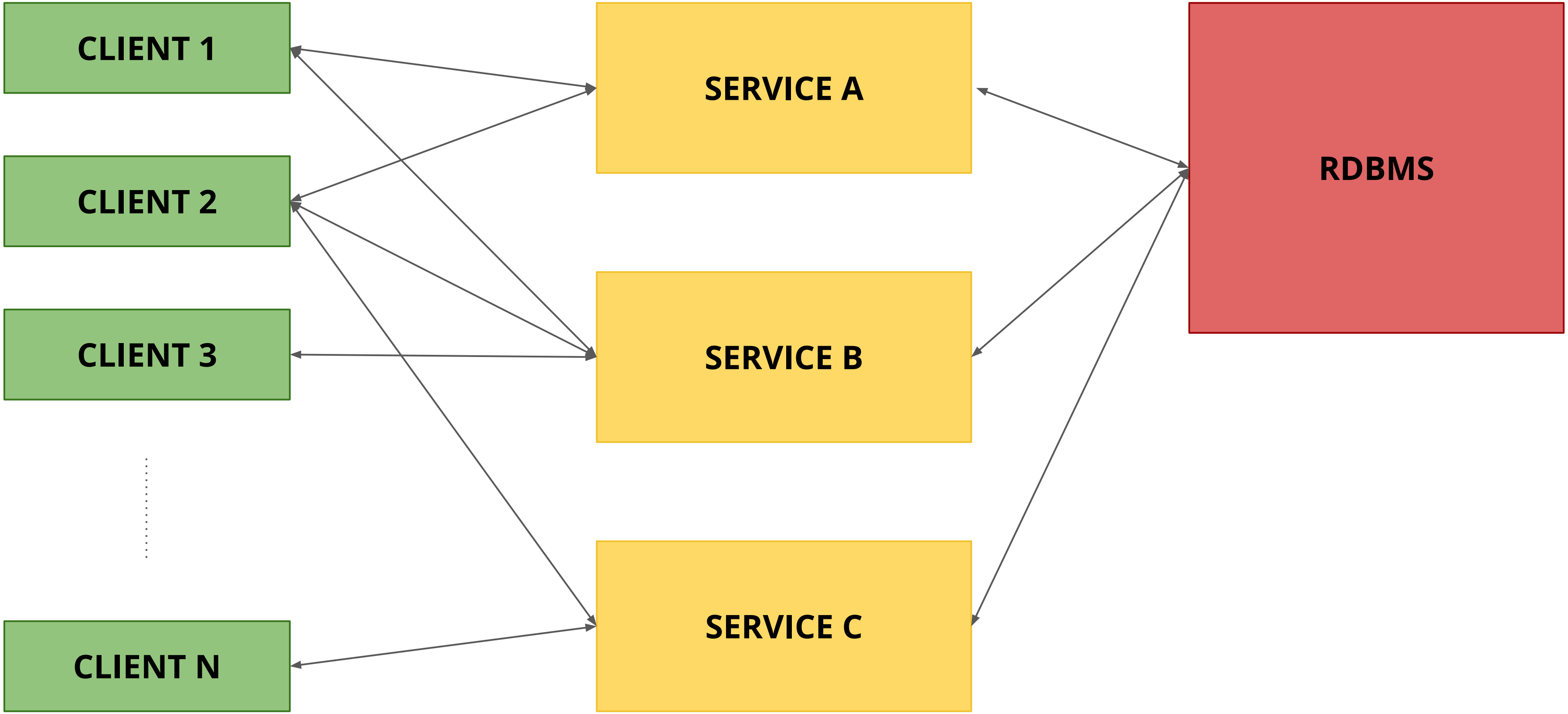
OUTLINE

- Motivation
- Object Caches
- Methodology
- Memcached & Redis features
- Benchmarks

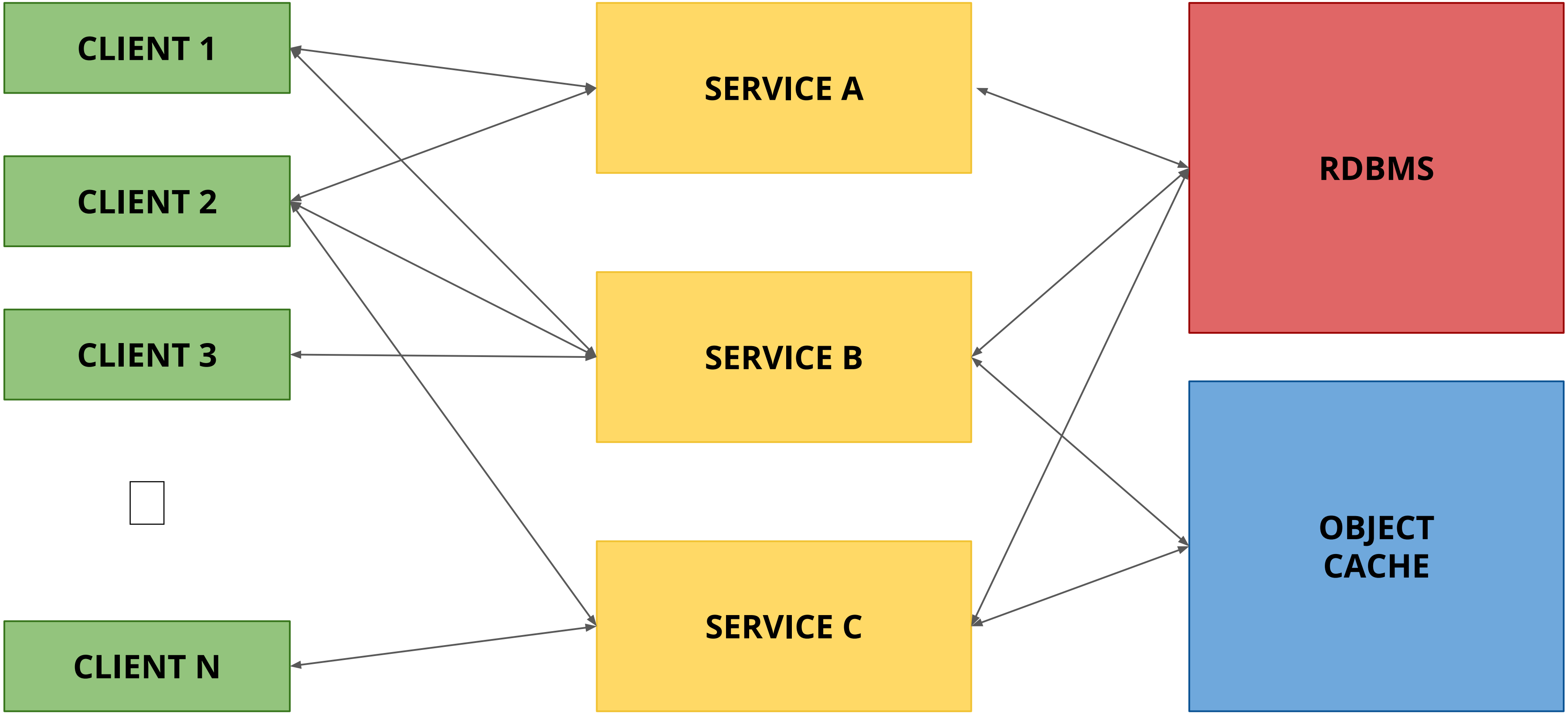
MOTIVATION

- **Caching is essential to system scalability**
- **Memcached**
  - Older
  - Extensively researched
  - Multi-threaded
  - Heavily utilized at Facebook, Twitter, Google, Amazon
- **Redis**
  - Younger
  - Single-threaded
  - Richer feature-set

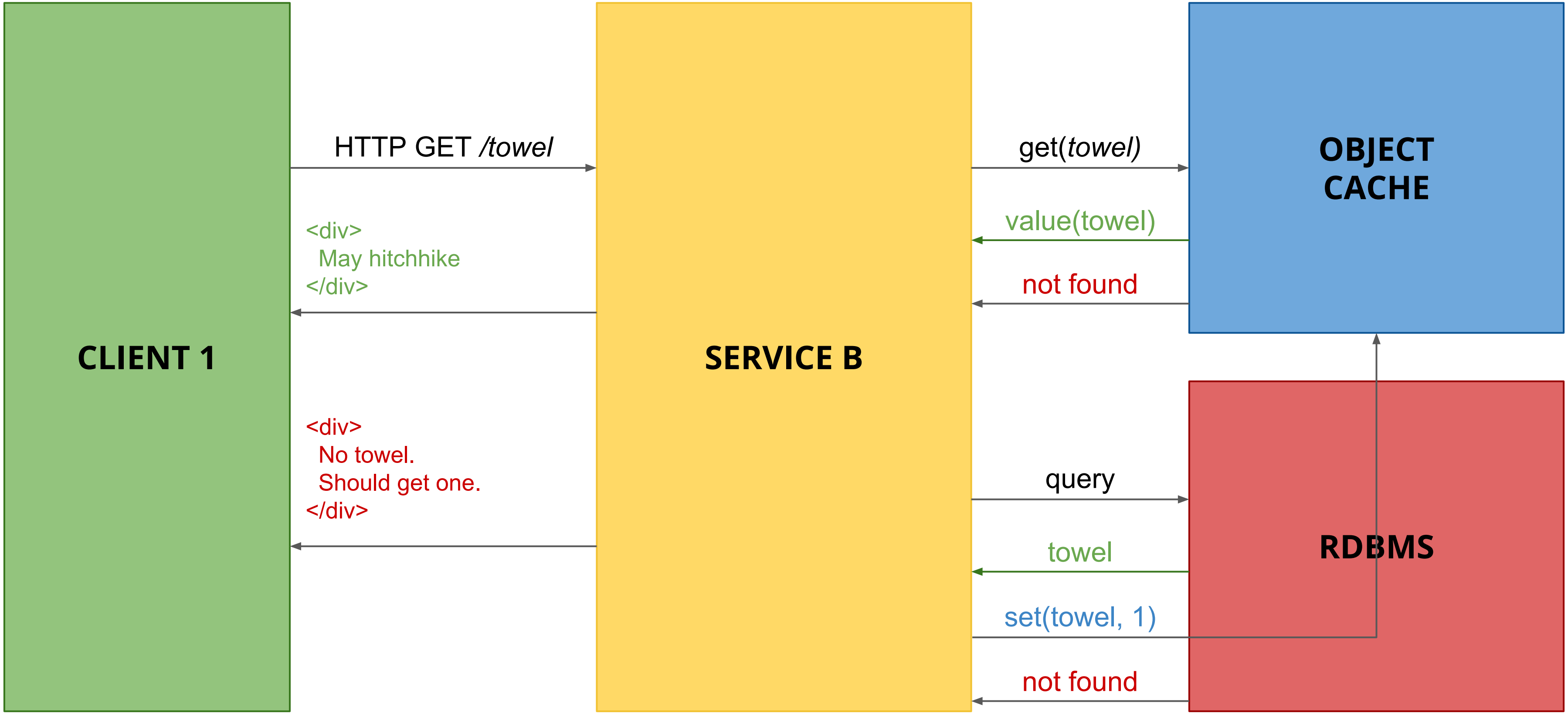
WEB SERVICES



WEB SERVICES WITH CACHES



ASK A CACHE


CLIENT 1
SERVICE B
OBJECT CACHE
RDBMS
HTTP GET /towel
<div>
May hitchhike
</div>
<div>
No towel.
Should get one.
</div>
get(towel)
value(towel)
not found
query
towel
set(towel, 1)
not found

OBJECT CACHES

- **API**
  - *get, set, delete*
- **Advantages**
  - Reduce response time
  - Avoid re-computation
  - Decrease RDBMS Load
  - Exploit temporal usage patterns
- **Applications**
  - Memcached
  - Redis

MEMCACHED & REDIS FEATURES

- **Memcached**
  - Multi-threaded (locks)
  - Multi-server
  - *get, set, delete, mget*
- **Redis**
  - Single-threaded / Multiple instances
  - Data persistence
  - *get, set, delete, mget* & *hyperloglog, lists, sets, sorted sets*

METHODOLOGY

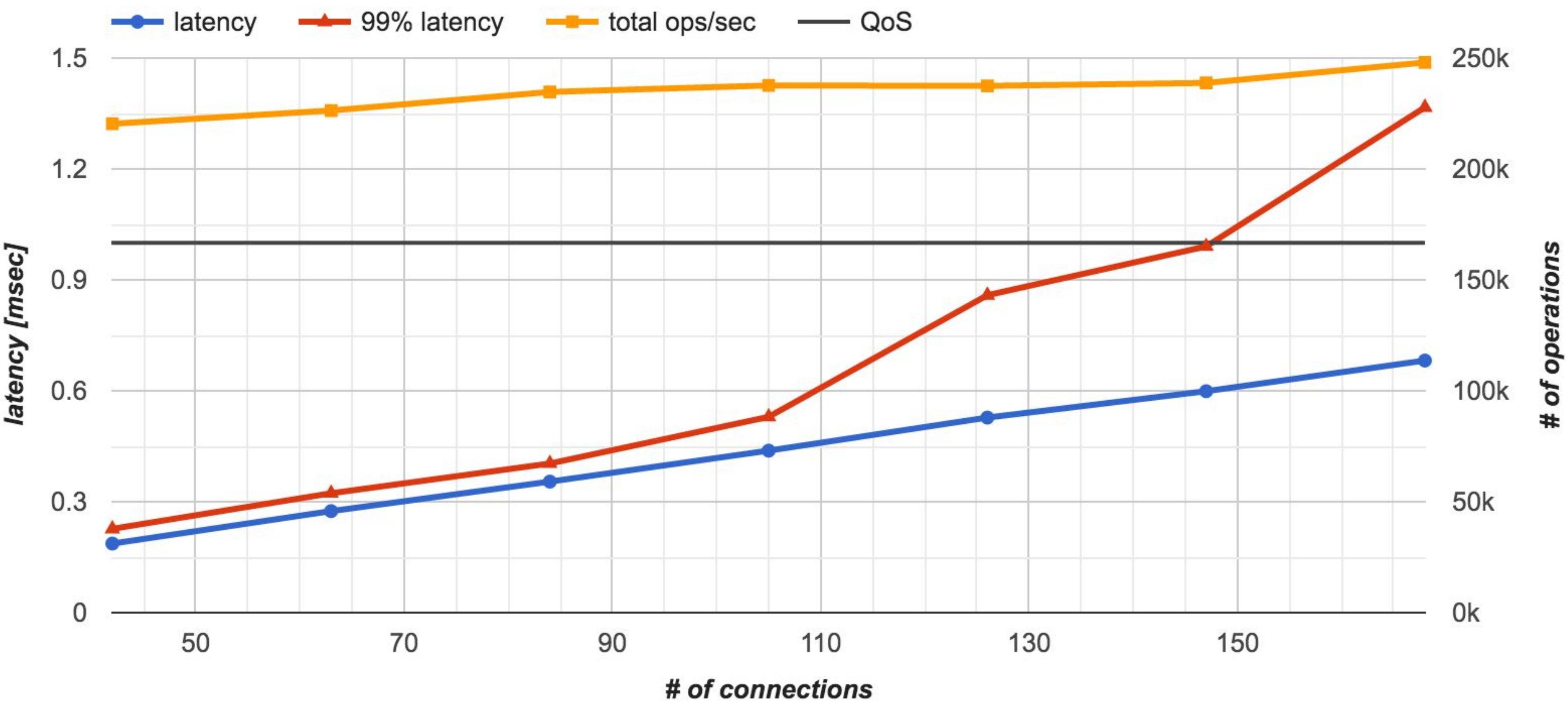
- **Metrics**
  - Latency & 99th Percentile Latency
  - CPU Utilization
  - Quality of Service (*99th* < *1ms*)
- **Benchmark**
  - 1 host, 7 clients, 1 rack
  - 6-core Intel Xeon @ 1.60GHz, 8GB RAM, 1Gbps NIC
  - Utilizes *MemtierBenchmark* by *RedisLabs*

BENCHMARKS

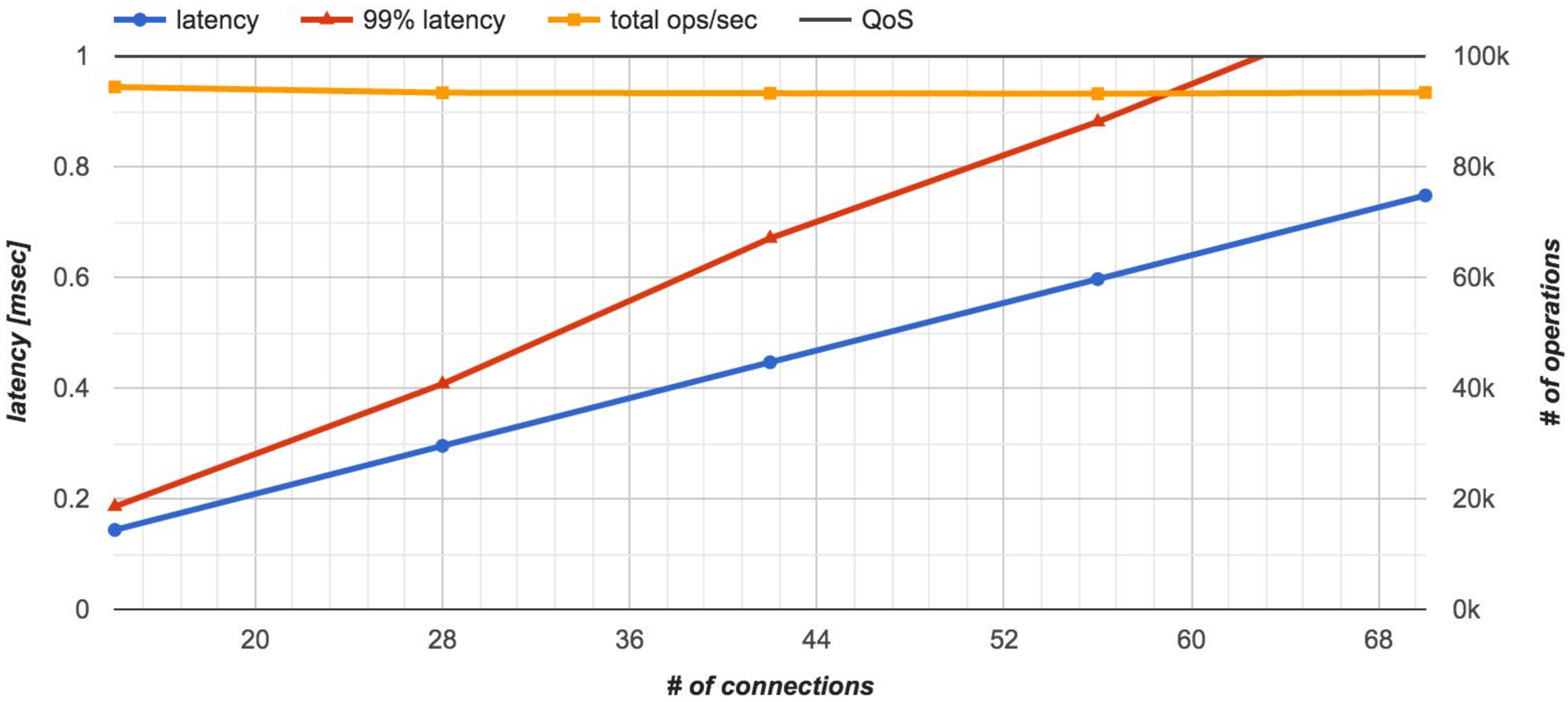
- **Out of the box performance**
- **Scaling up**
- **Object Size**
- **Key Distribution**

- Unless stated otherwise
  - object size is *64 bytes*
  - Key distribution is *uniform* with *100m keys*

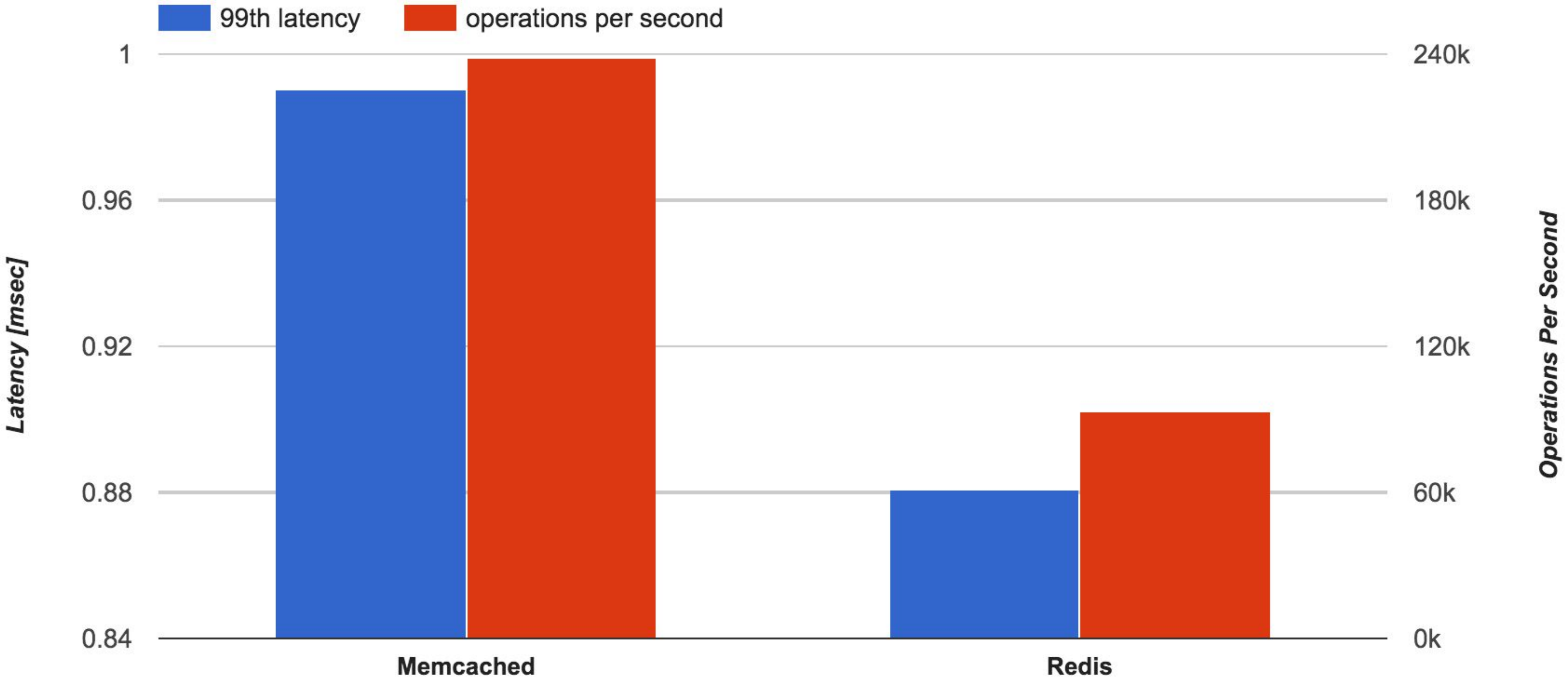
BASELINE: MEMCACHED (4 threads)



BASELINE: REDIS (1 thread, 1 instance)



BASELINE: MEMCACHED VS REDIS

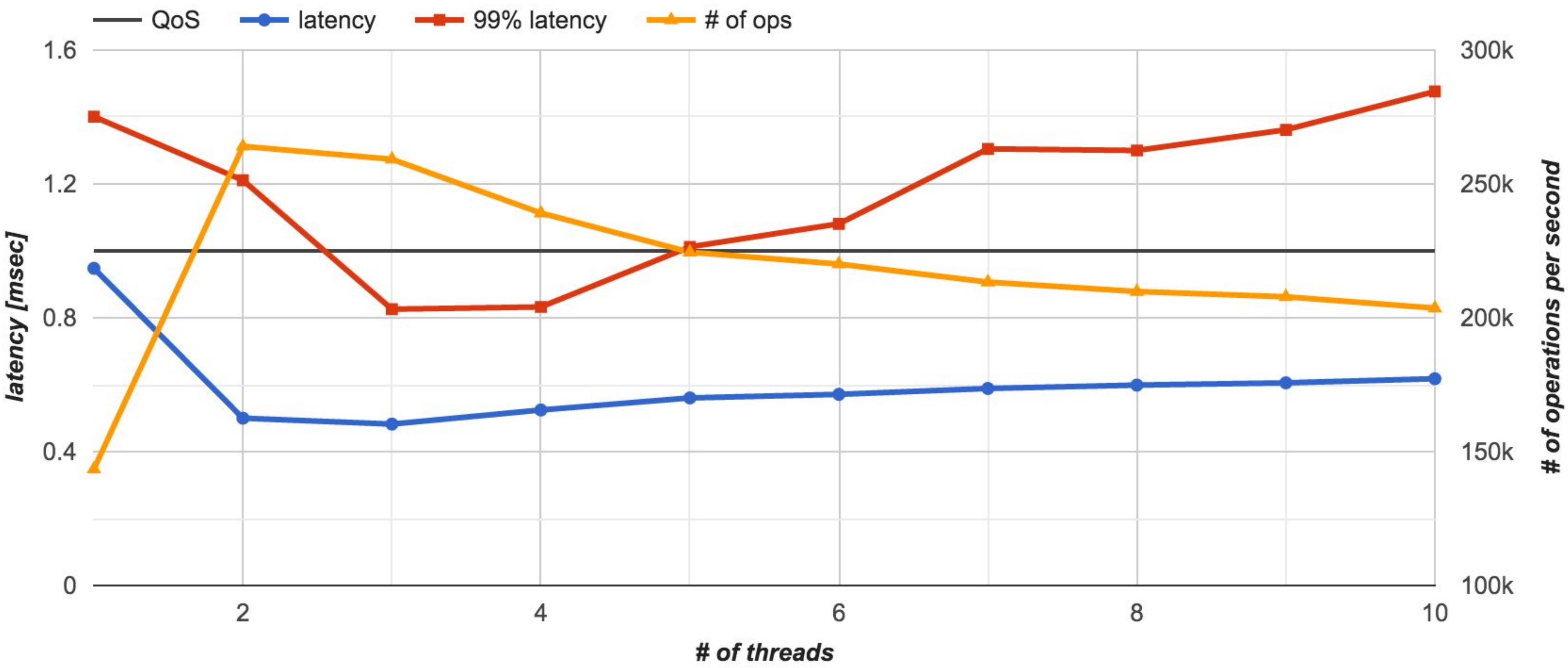


SCALE UP: How do we scale M & R vertically?

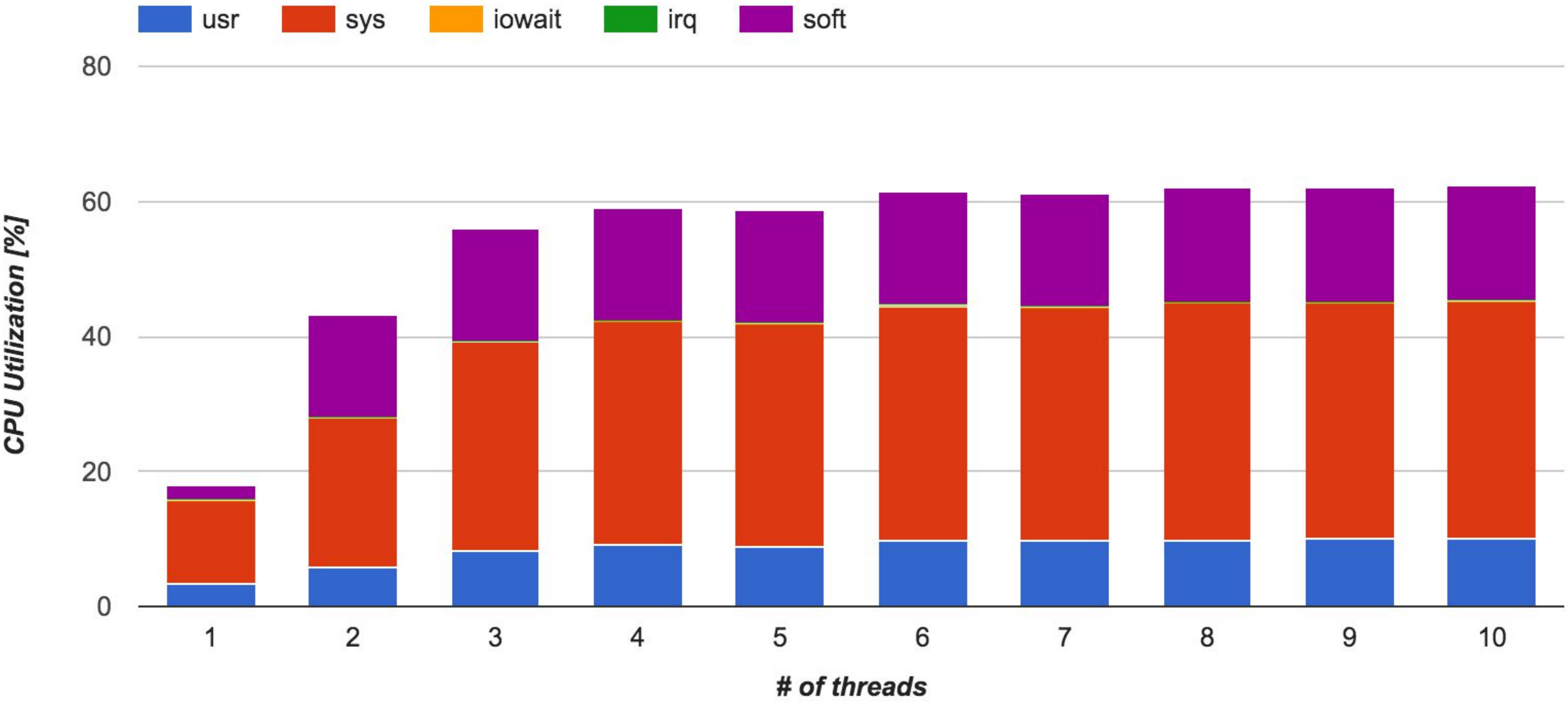
- **More hardware**
- **Faster hardware**
- **Multiple Threads**
    - Only Memcached
- **Multiple Instances**
    - (spawn multiple isolated processes of the same application)
    - Both Memcached and Redis

Note: Each server has 6 CPUs.

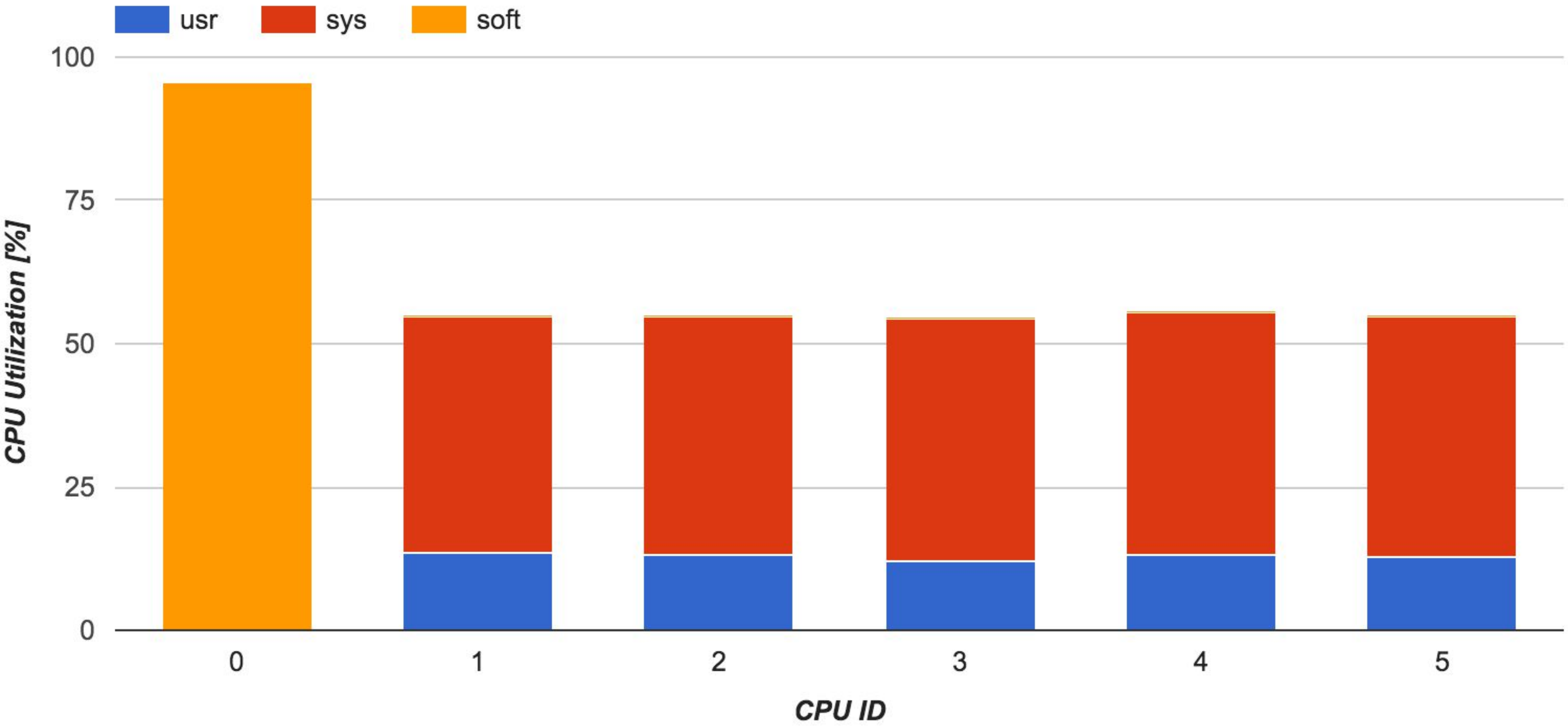
SCALE UP: MEMCACHED - Latency



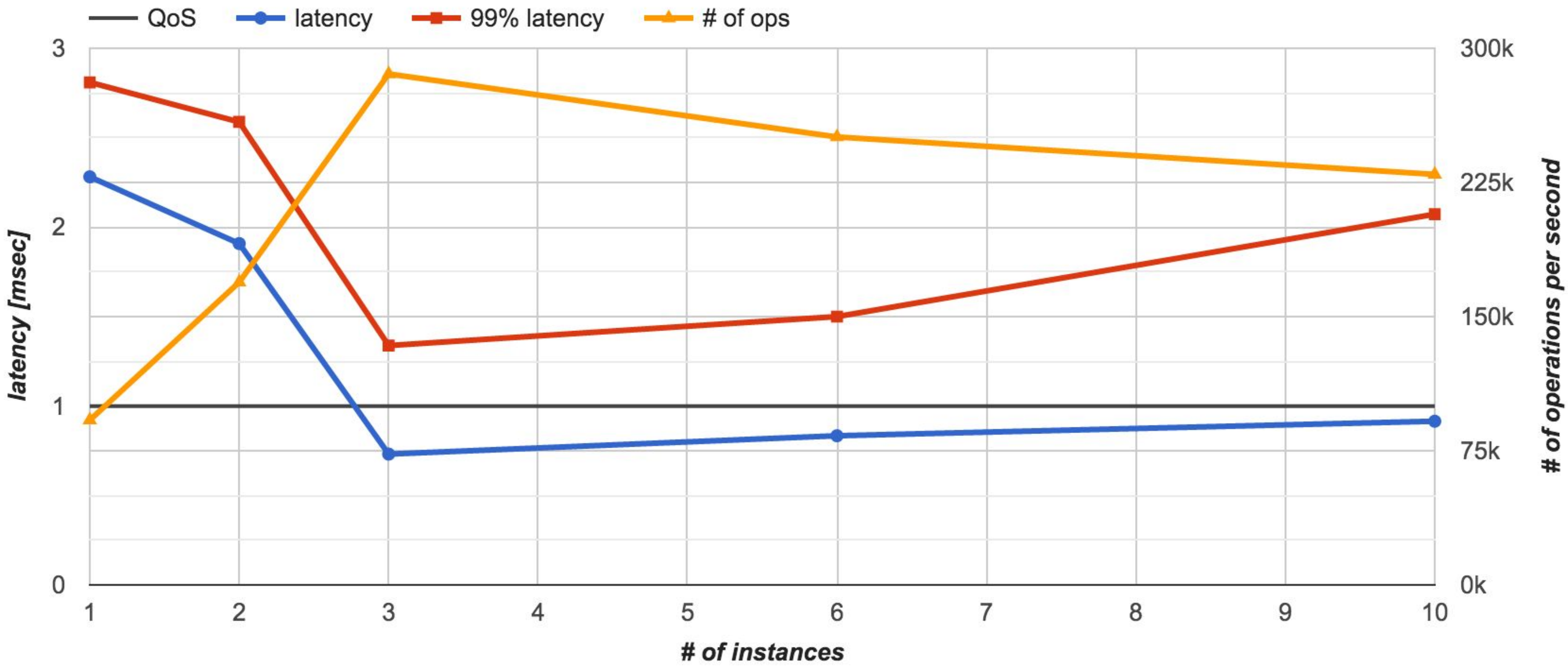
SCALE UP: MEMCACHED - CPU Utilization



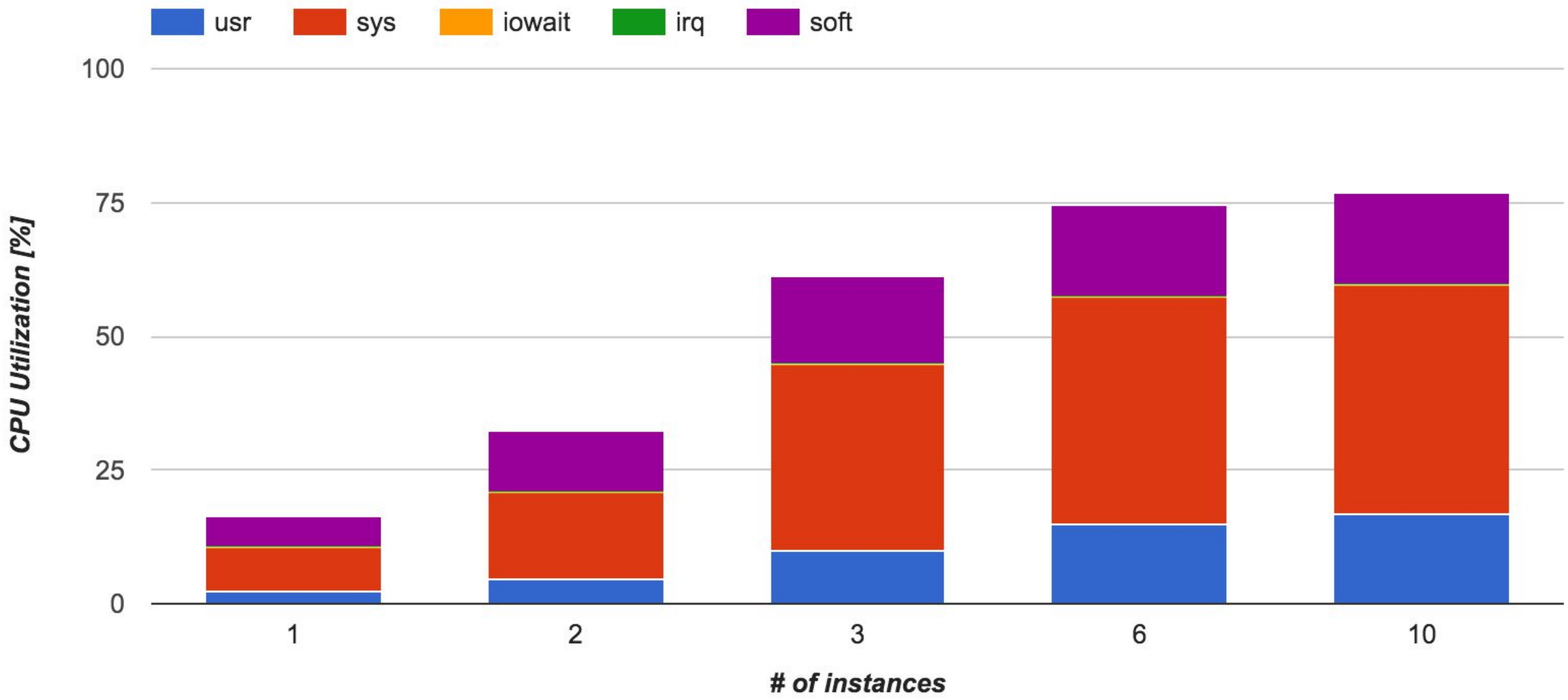
SCALE UP: MEMCACHED - Individual CPU Utilization



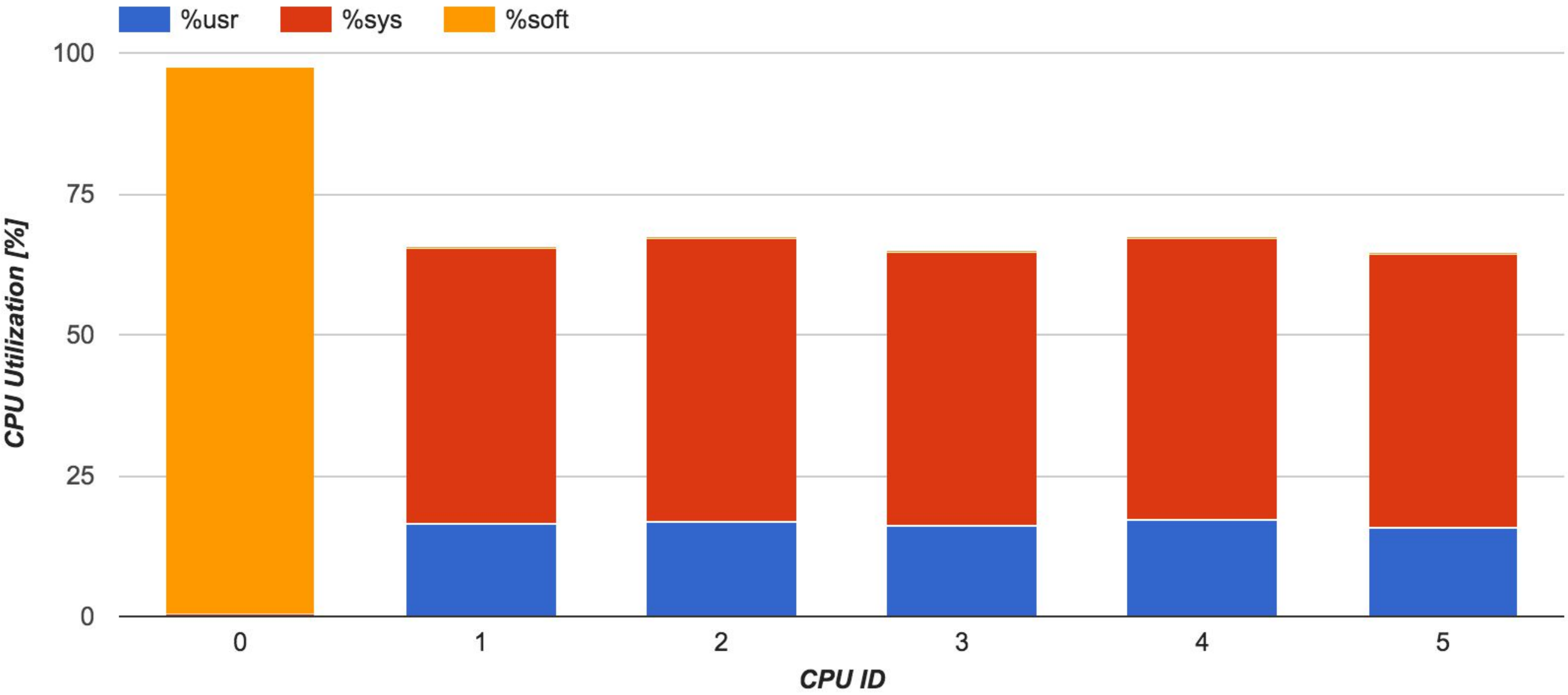
SCALE UP: REDIS - Latency



SCALE UP: REDIS - CPU Utilization



SCALE UP: REDIS - Individual CPU Utilization



SCALE UP: THE PROBLEM

Why are all software interrupts processed on a single core?

```
>> IRQ Affinity
```

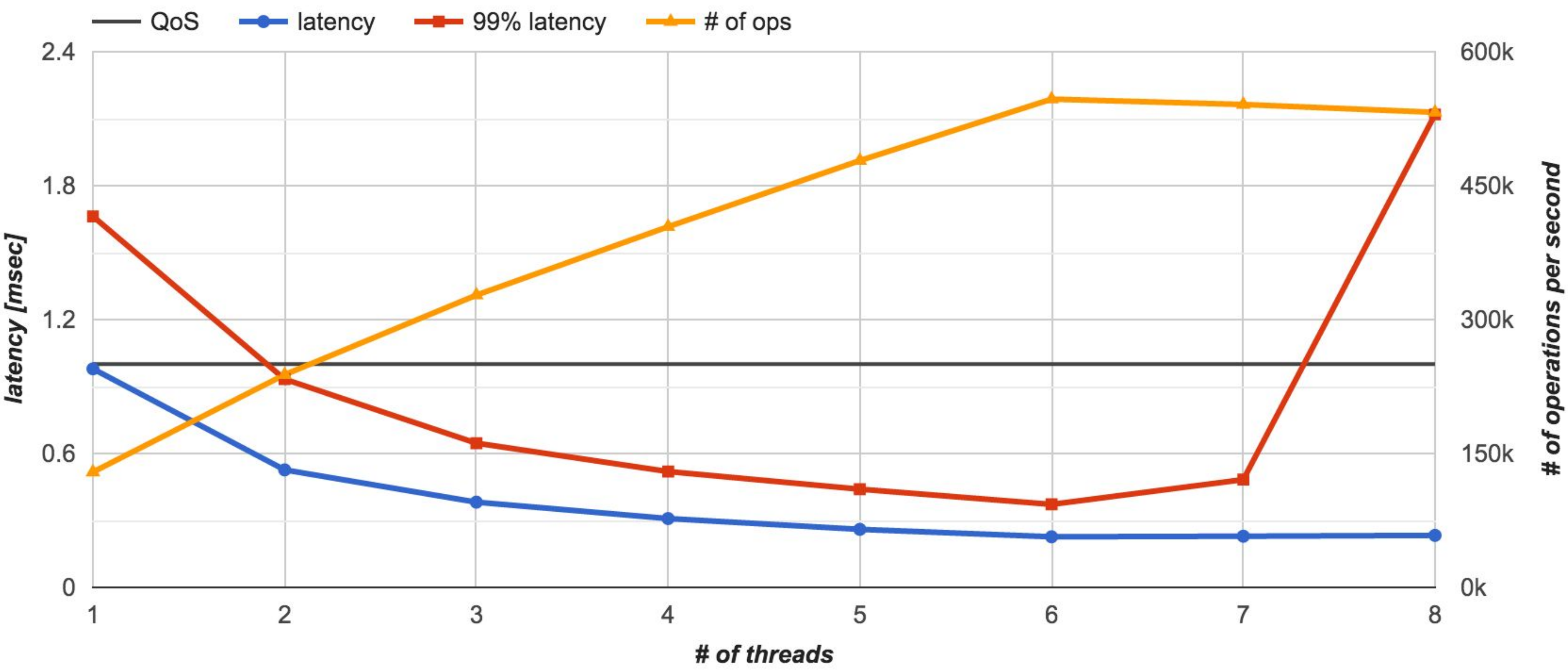
SCALE UP: THE SOLUTION

Distribute software interrupt processing across all cores.

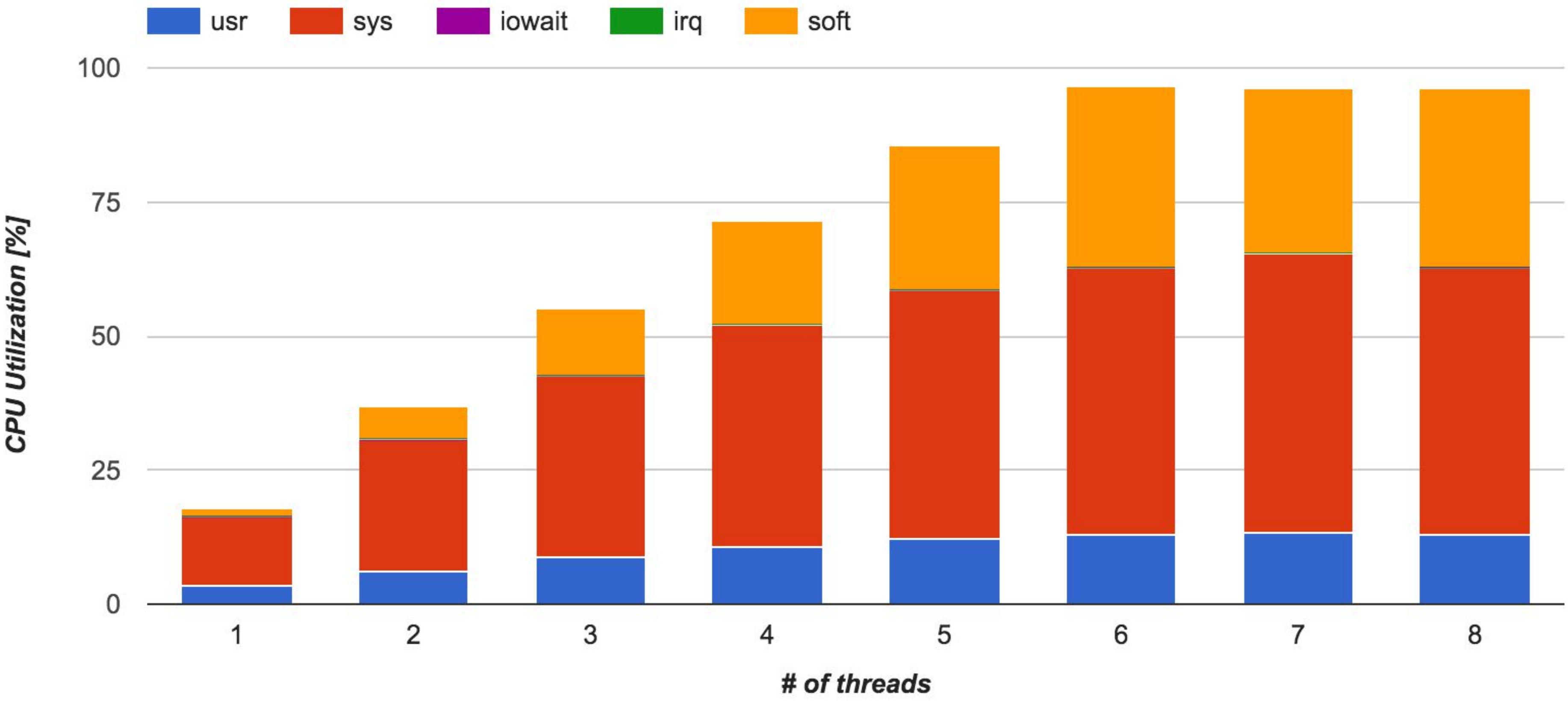
```
$ cat /proc/interrupts | grep eth0 | awk ’{ print $1 " " $9 }’

$ echo CPU_ID > /proc/irq/QUEUE_ID/smp_affinity_list
```

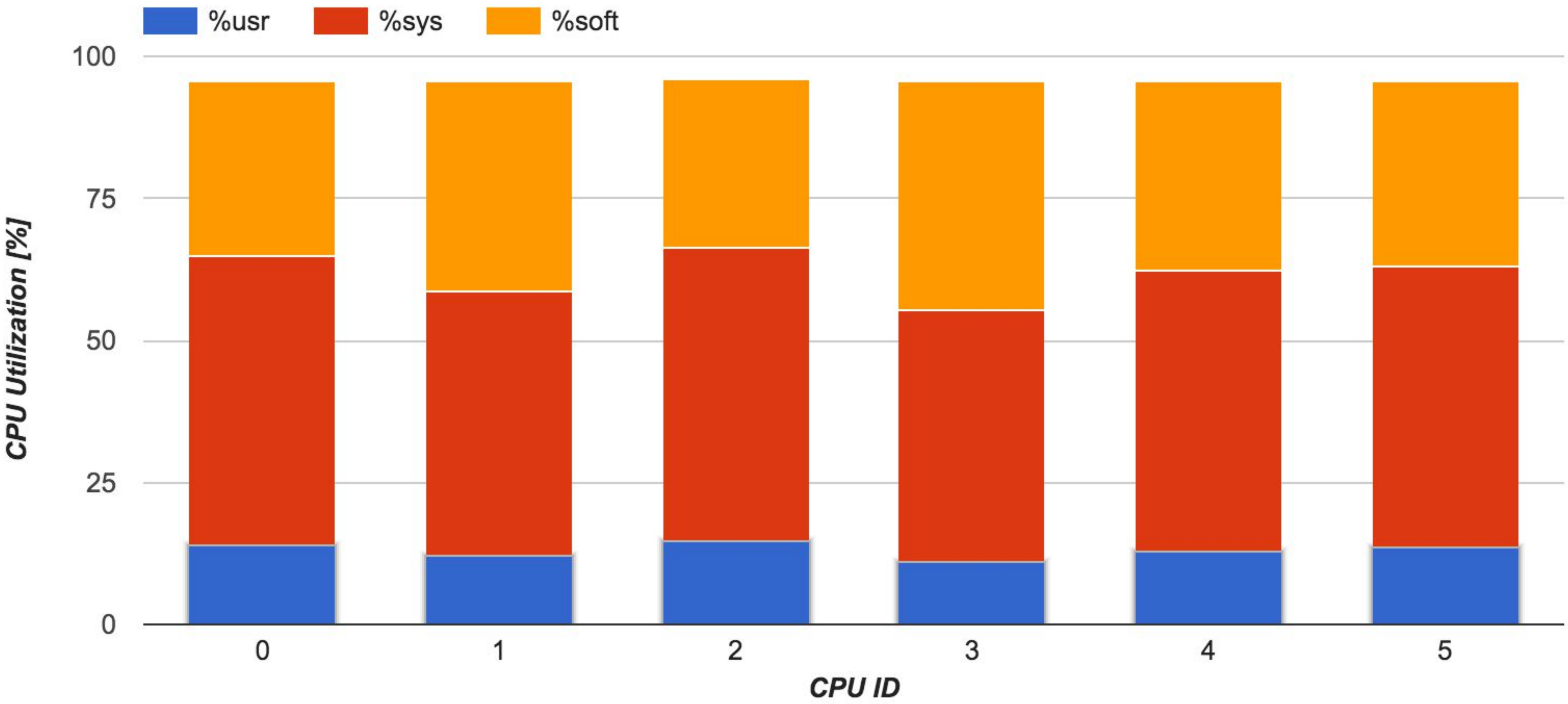
SCALE UP: MEMCACHED - Latency



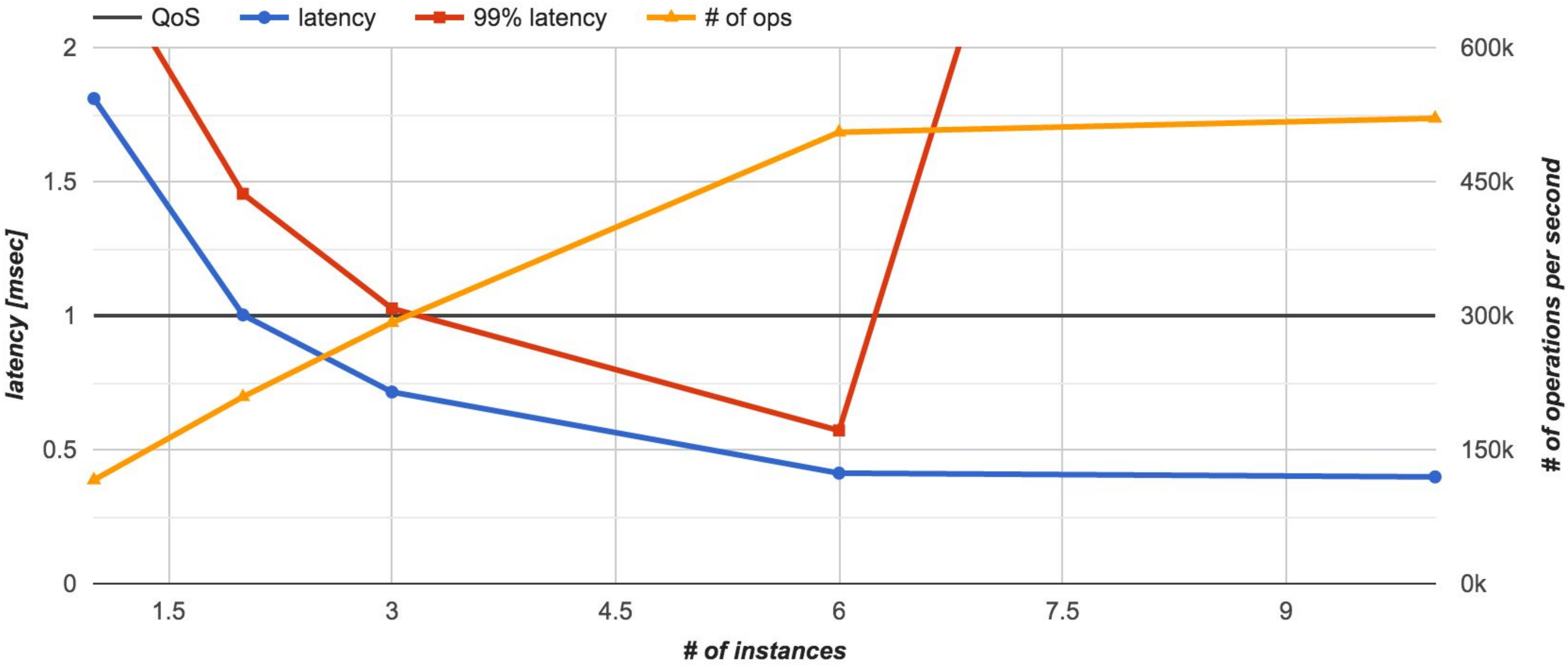
SCALE UP: MEMCACHED - CPU Utilization



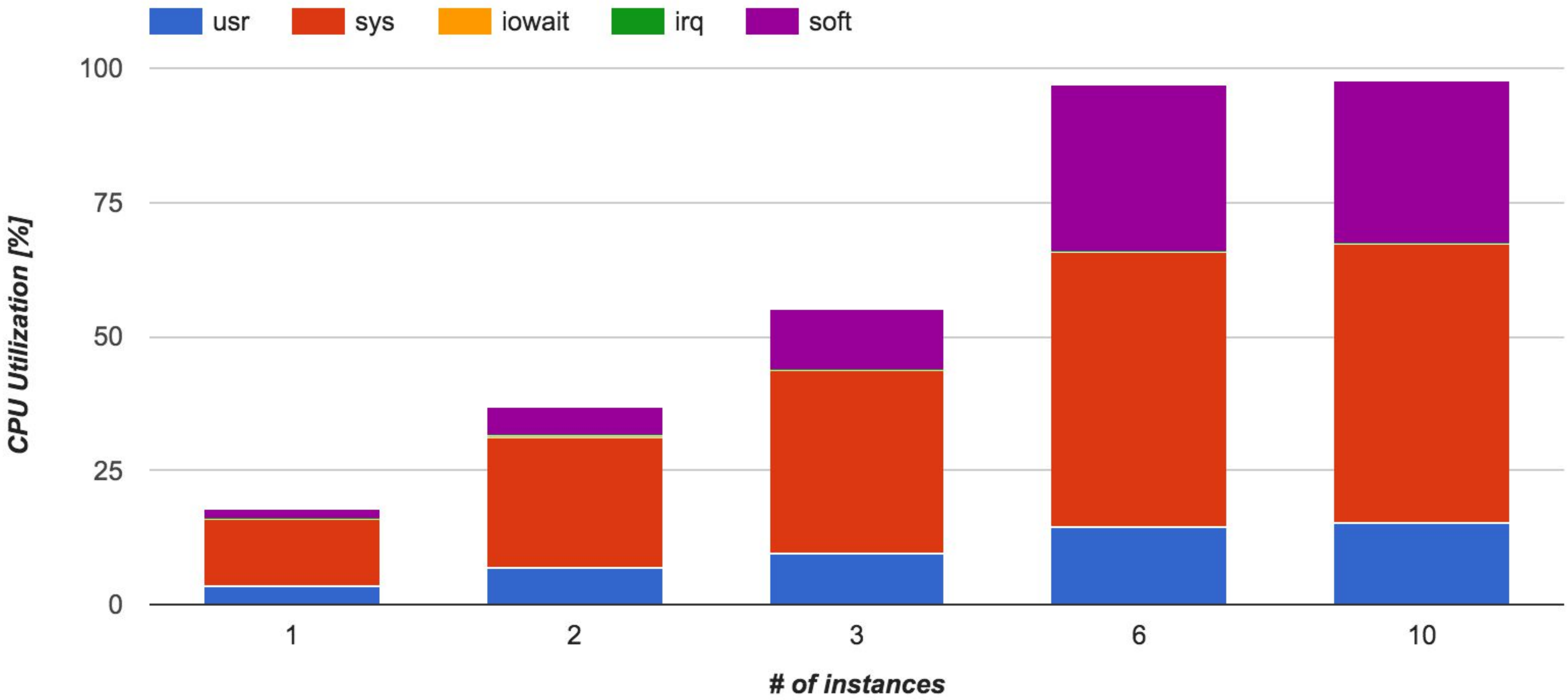
SCALE UP: MEMCACHED - Individual CPU Utilization



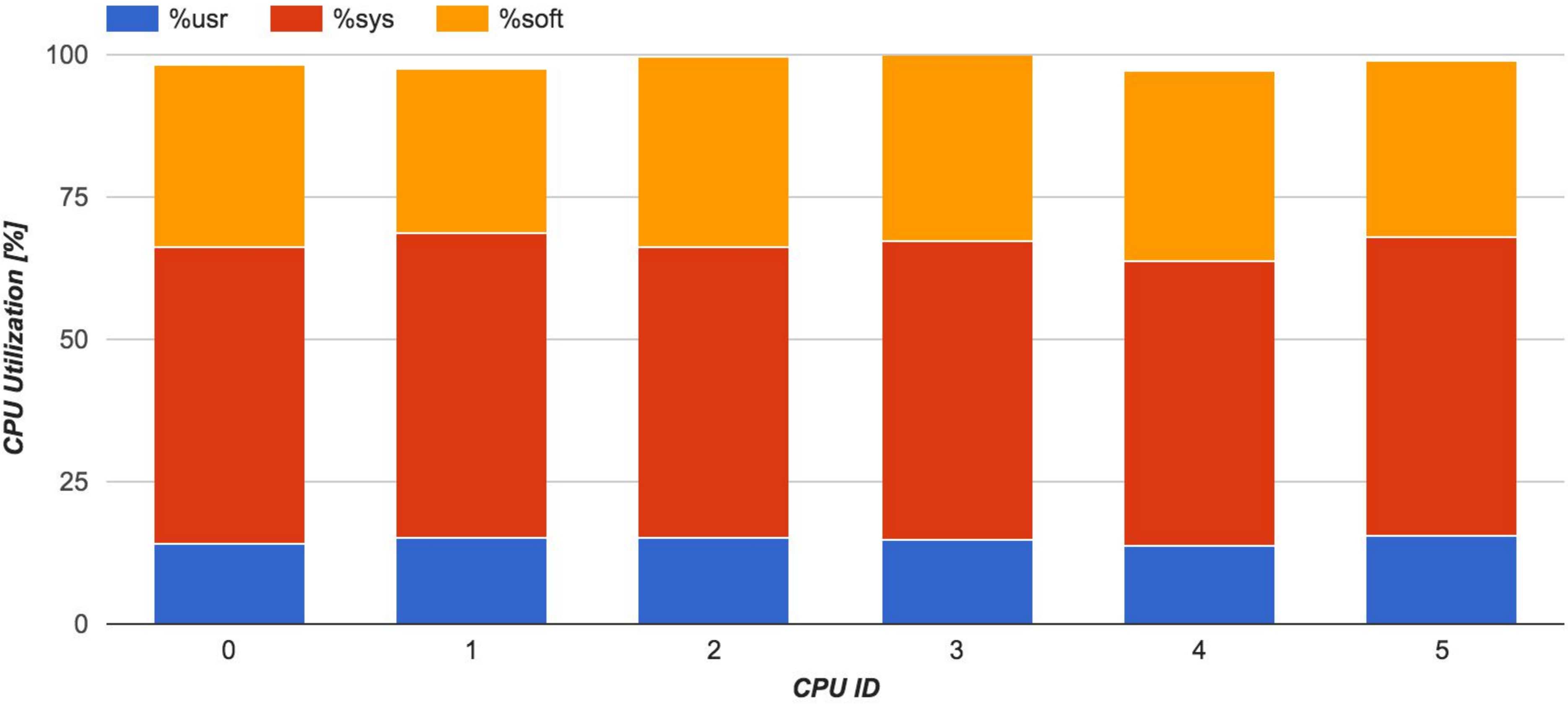
SCALE UP: REDIS - Latency



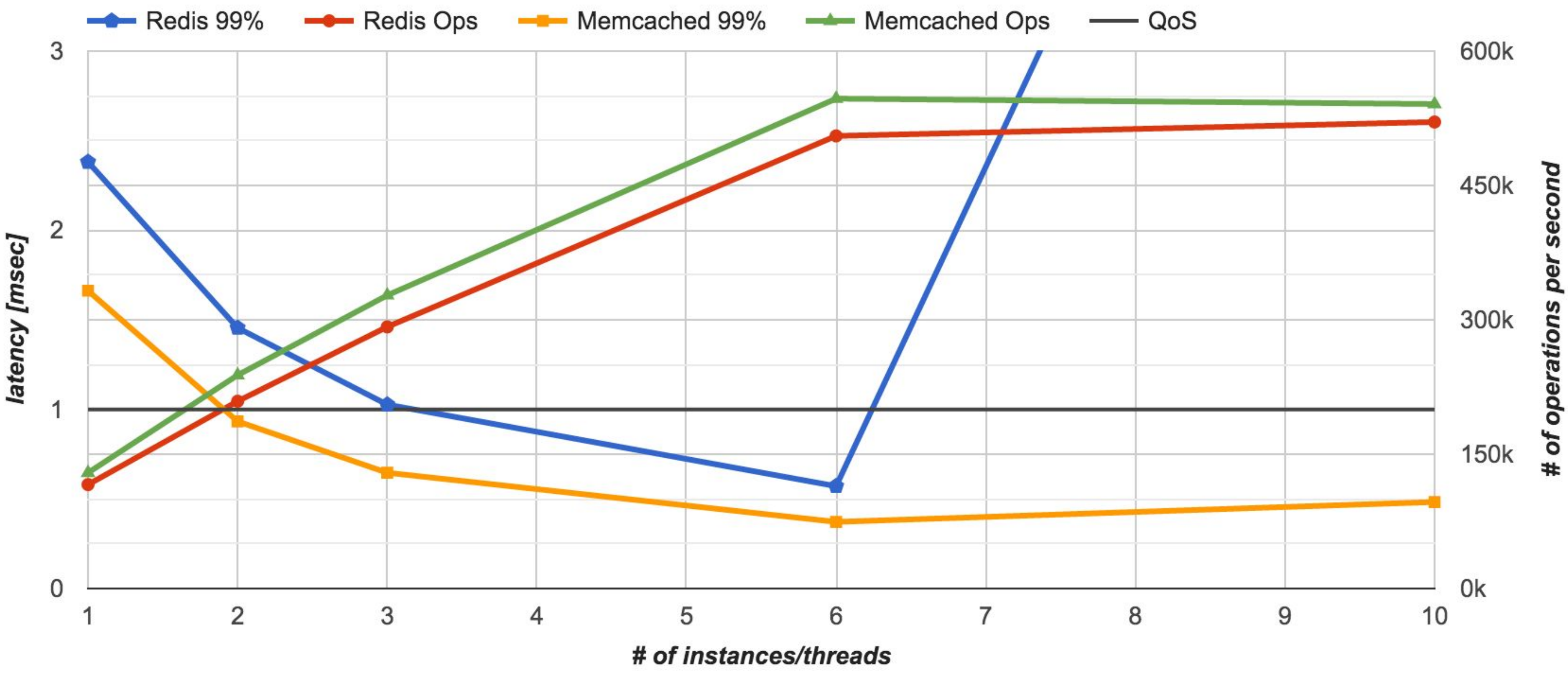
SCALE UP: REDIS - CPU Utilization



SCALE UP: REDIS - CPU Utilization



SCALE UP: REDIS vs MEMCACHED



SCALE UP: Conclusion

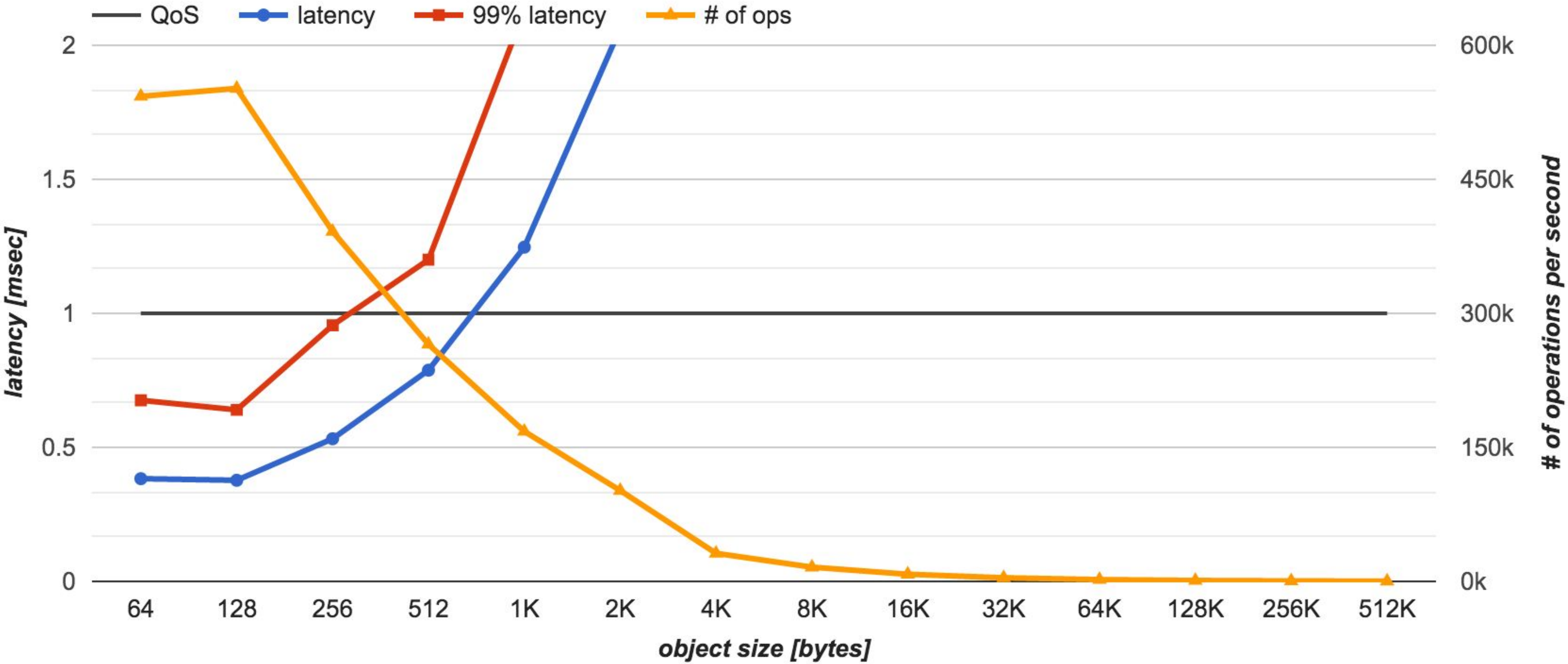
- **Software interrupt processing is a bottleneck**
  - CPU Utilization suboptimal
  - Let all CPU cores handle interrupts
- **Best performance is with as many threads/instances as CPU cores**
- **Stats:**
  - Memcached: 550k requests/second, 0.45ms
  - Redis: 500k requests/second, 0.52ms

OBJECT SIZE

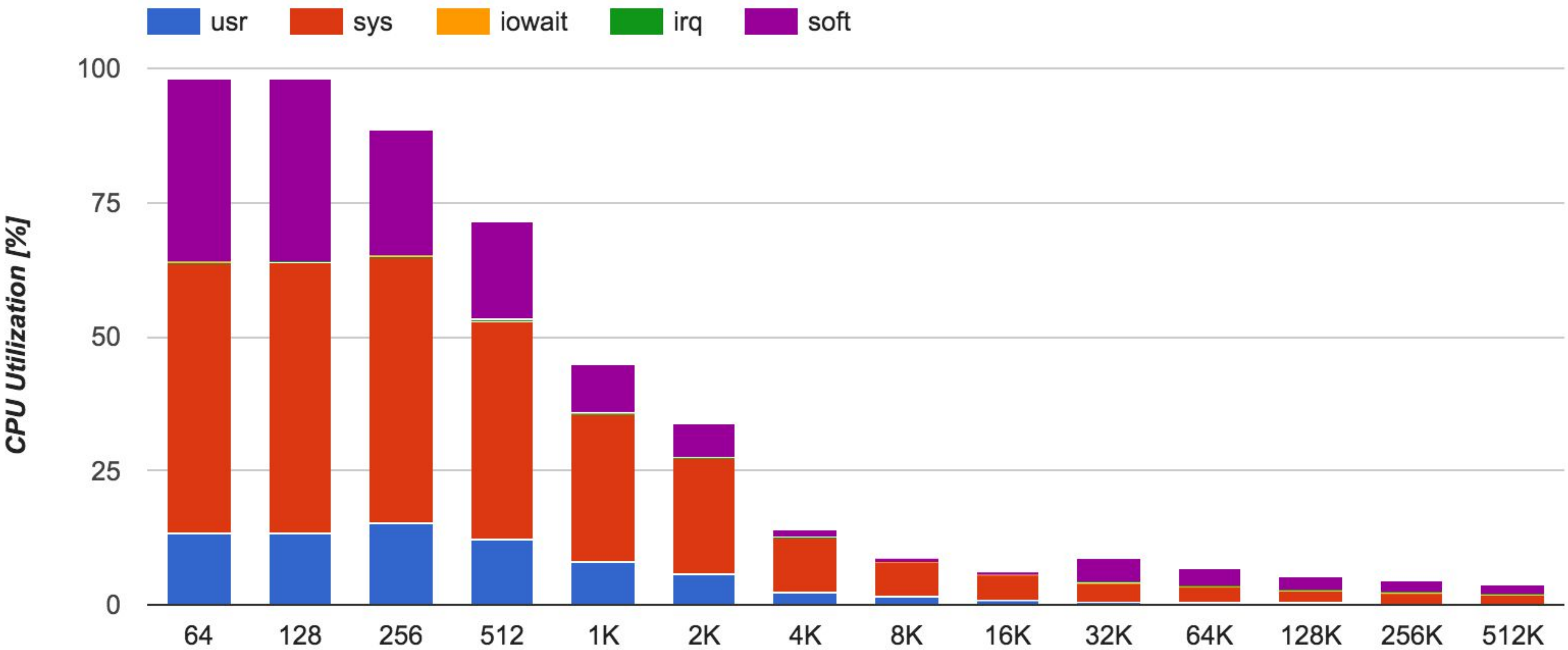
- **Vary the object size with best performing configuration**
  - 6 threads/instances
  - IRQ Affinity set

- Key space decreases from 100m proportionately to object size
  - (large values, need less keys to keep space constant)
- Benchmarks run on a 1Gbps link

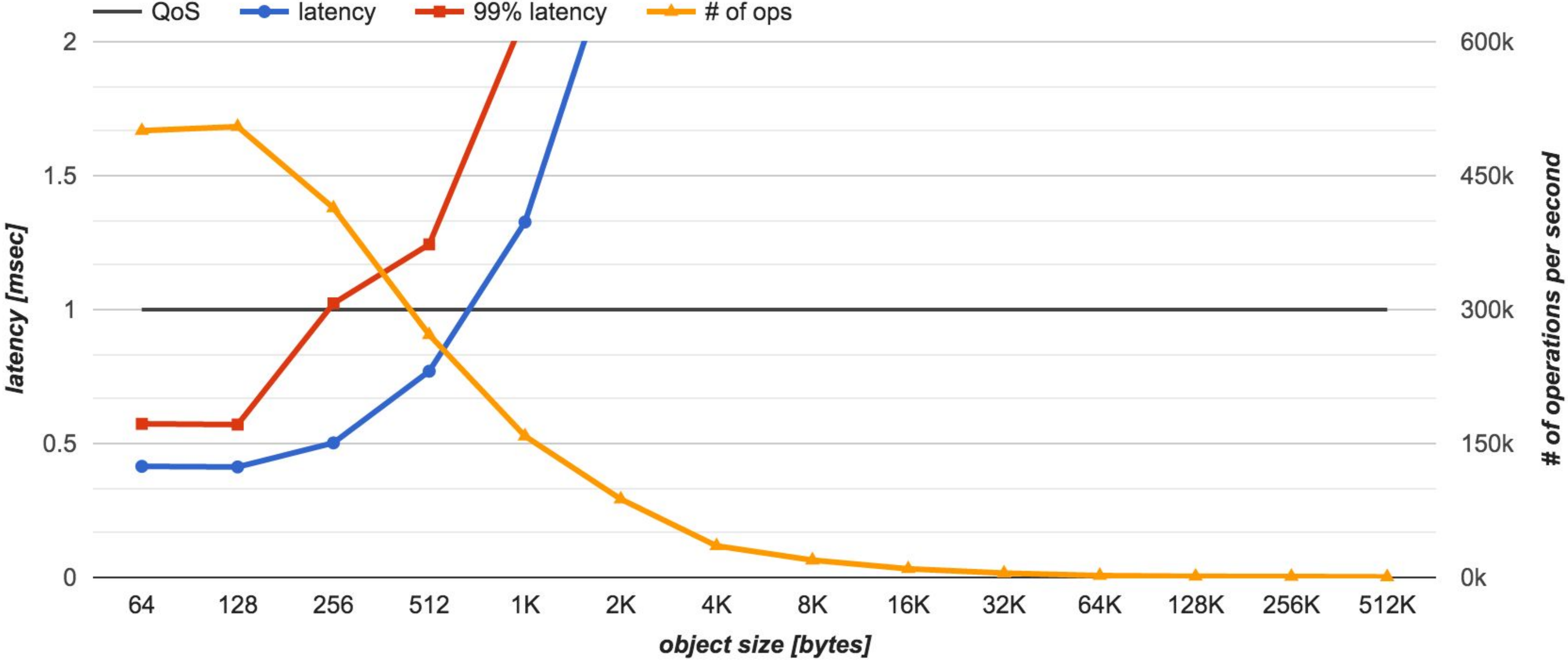
OBJECT SIZE: MEMCACHED - Latency



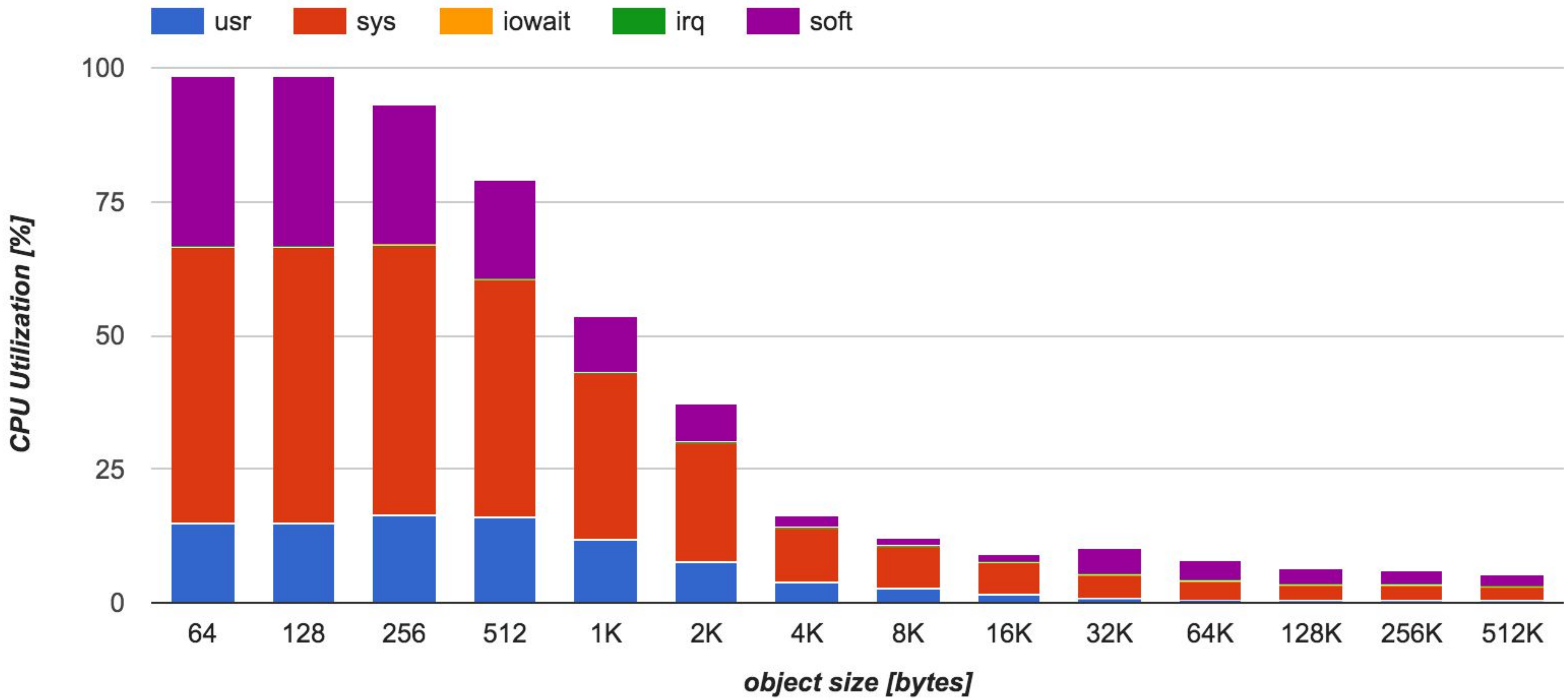
OBJECT SIZE: MEMCACHED - CPU Utilization



OBJECT SIZE: REDIS - Latency



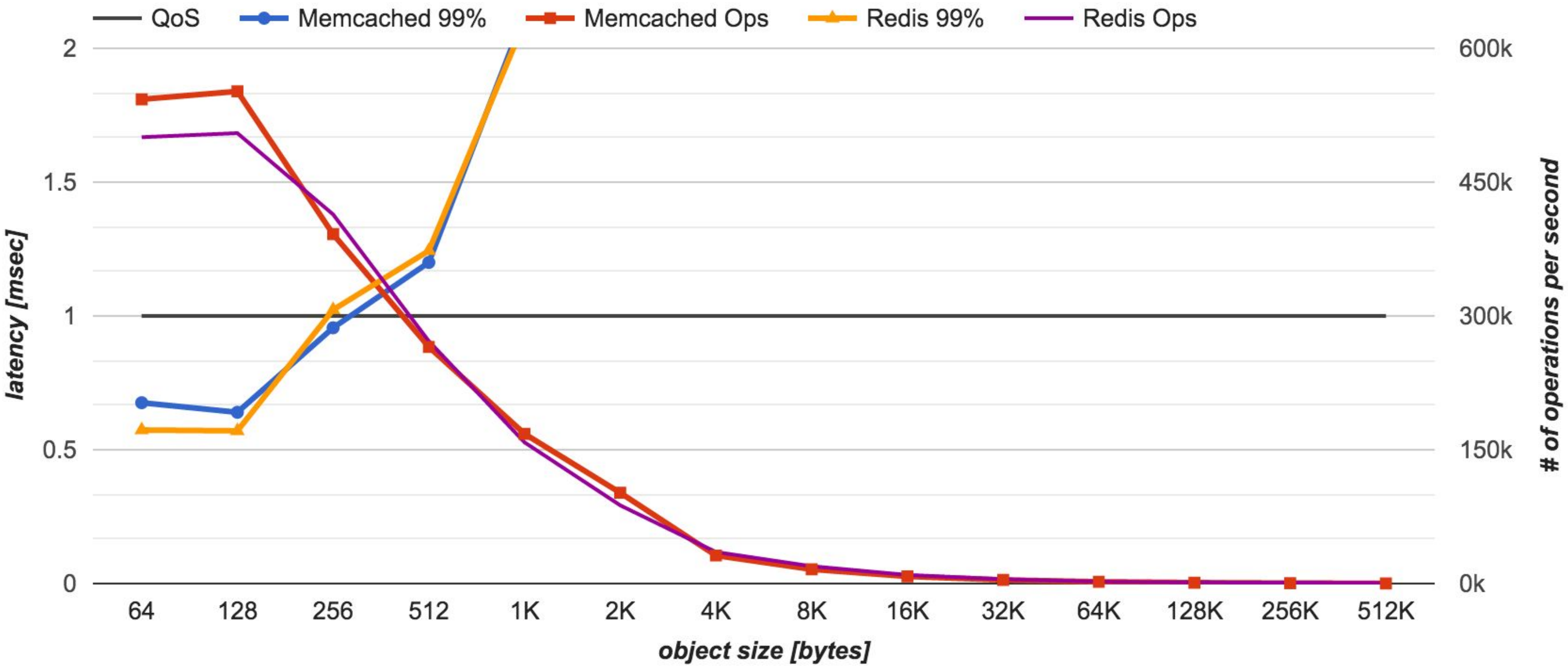
OBJECT SIZE: REDIS - CPU Utilization



OBJECT SIZE Evaluation

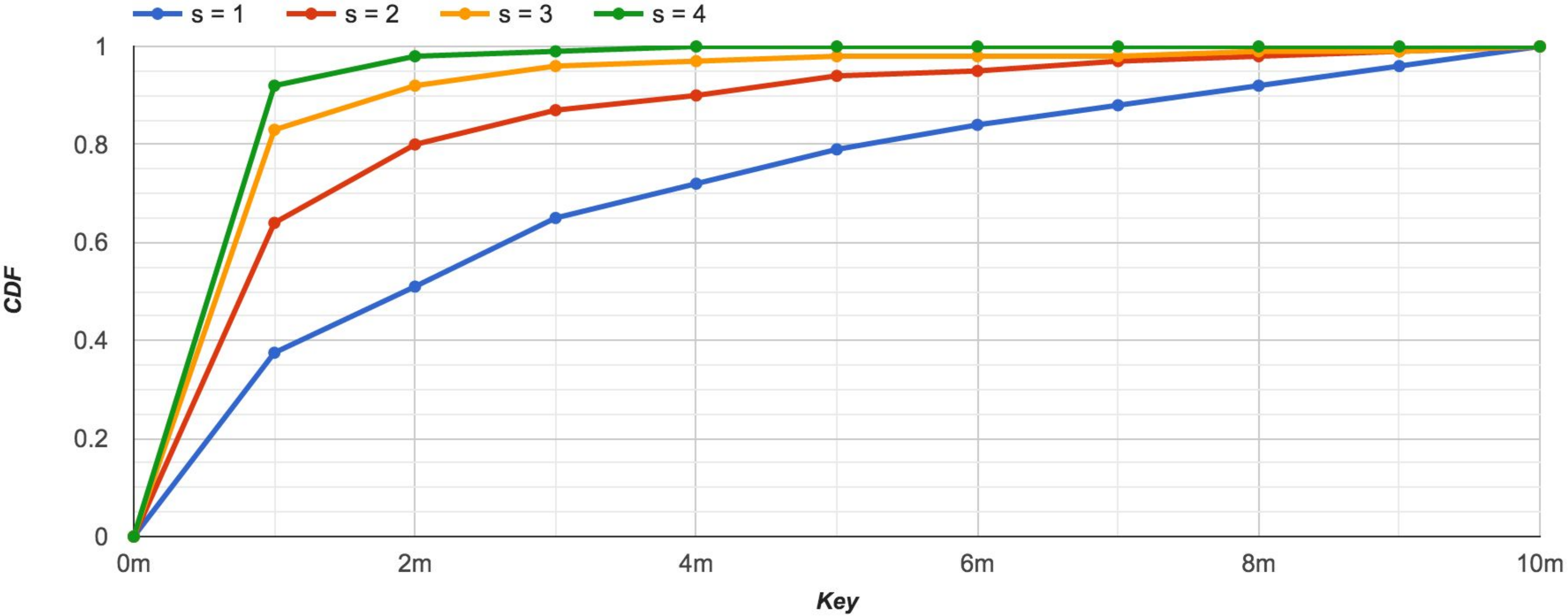
- **Network dominates large objects**
- **QoS**
  - Memcached: 256 KB
  - Redis: 128 KB
  - 99th < 1ms QoS may be too strict for large objects
- **Memcached performs better**
- **Neither optimized for large objects**

OBJECT SIZE: MEMCACHED vs REDIS



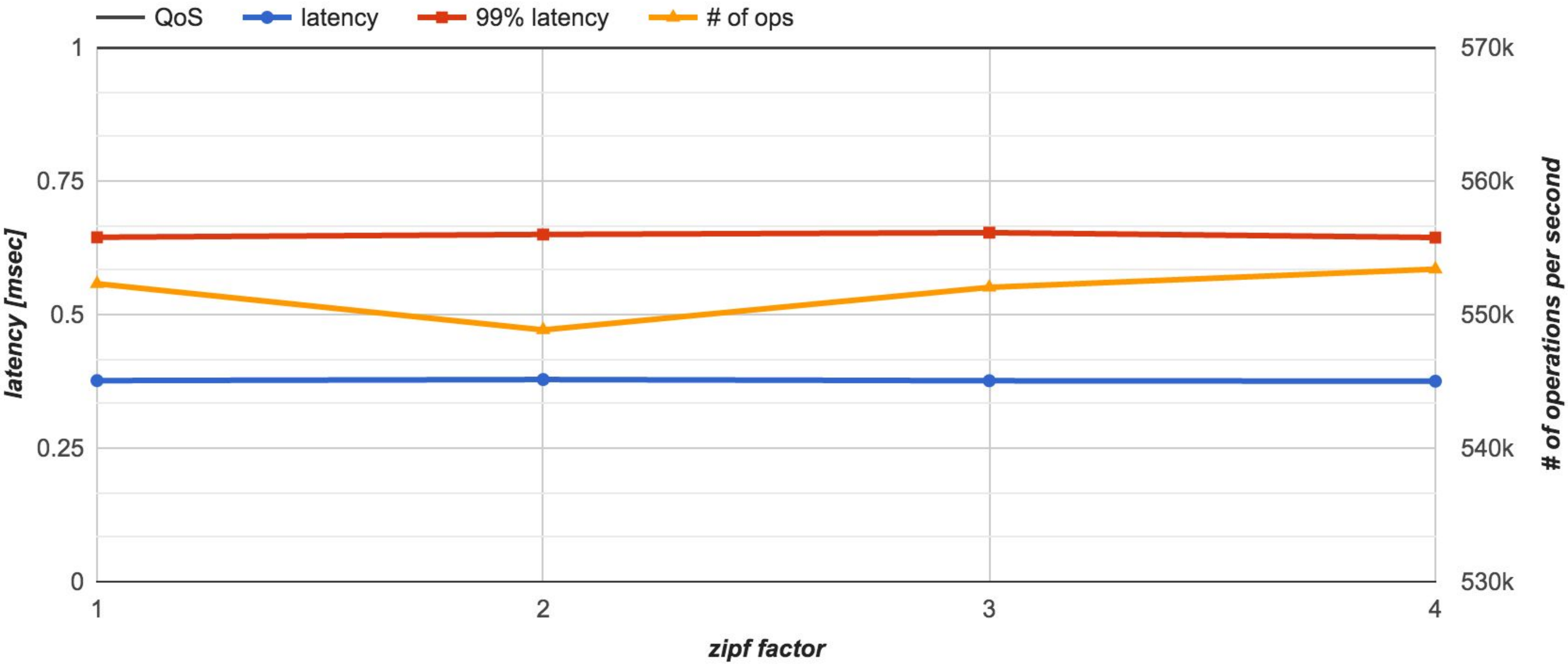
KEY DISTRIBUTION

- **Idea: Some keys appear more often than others**
  - Zipf



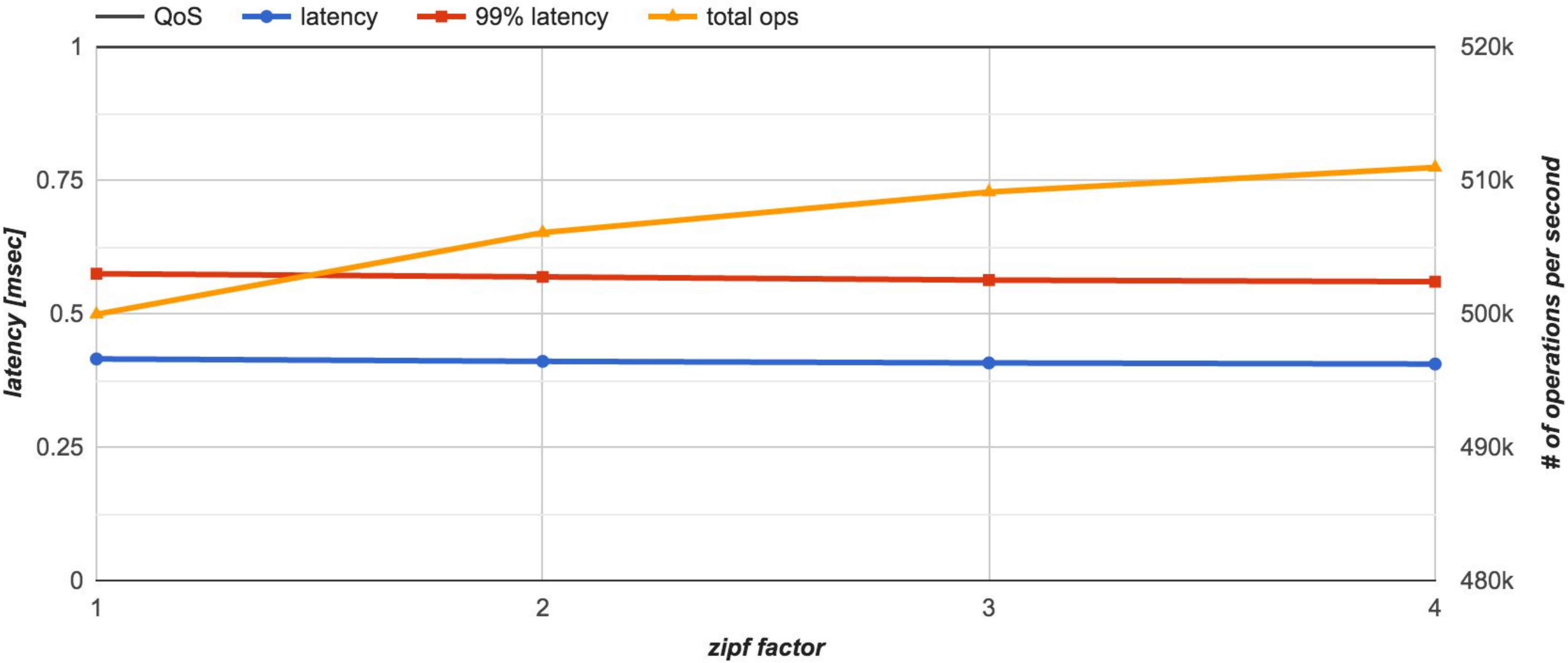
KEY DISTRIBUTION: MEMCACHED

Higher zipf factor = higher skew



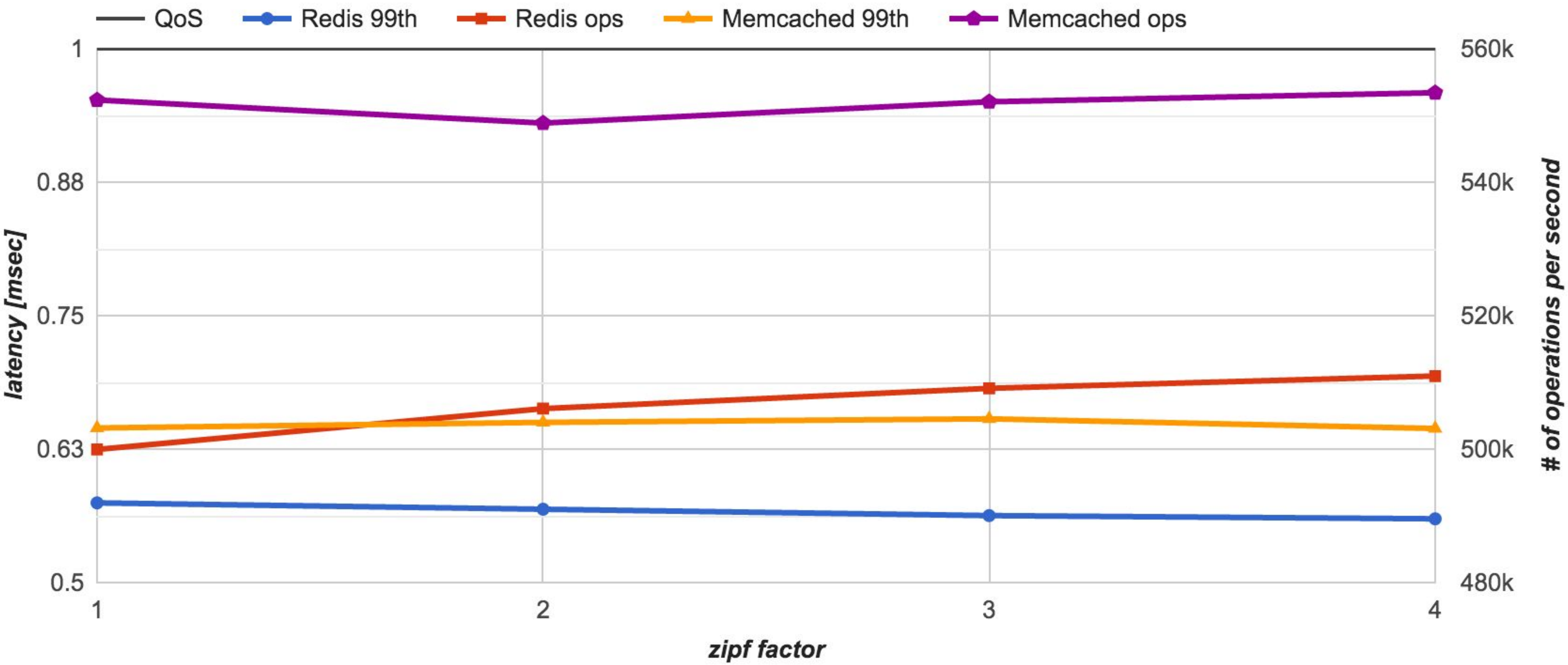
KEY DISTRIBUTION: REDIS

Higher zipf factor = higher skew



KEY DISTRIBUTION: MEMCACHED vs REDIS

Higher zipf factor = higher skew



KEY DISTRIBUTION Evaluation

- **99th percentile latency unaffected**
- **Redis operations per second improve with higher skew**
- **Memcached remains stable**

CONCLUSION

- **Memcached outperforms Redis on common feature set**
- **Redis scaled up performs nearly as good as Memcached**
    - Multi-instance single threaded application can perform nearly as good as multi-threaded
- **Both Memcached & Redis perform better with smaller objects**
    - Client side object splitting/joining
- **Redis performance improves with skewed key distributions**

FUTURE WORK

- Multiple server configurations
- More hardware
- Faster hardware
- Memcached Cluster vs (new) Redis Cluster

THANKS

Q & A