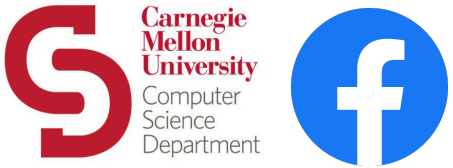
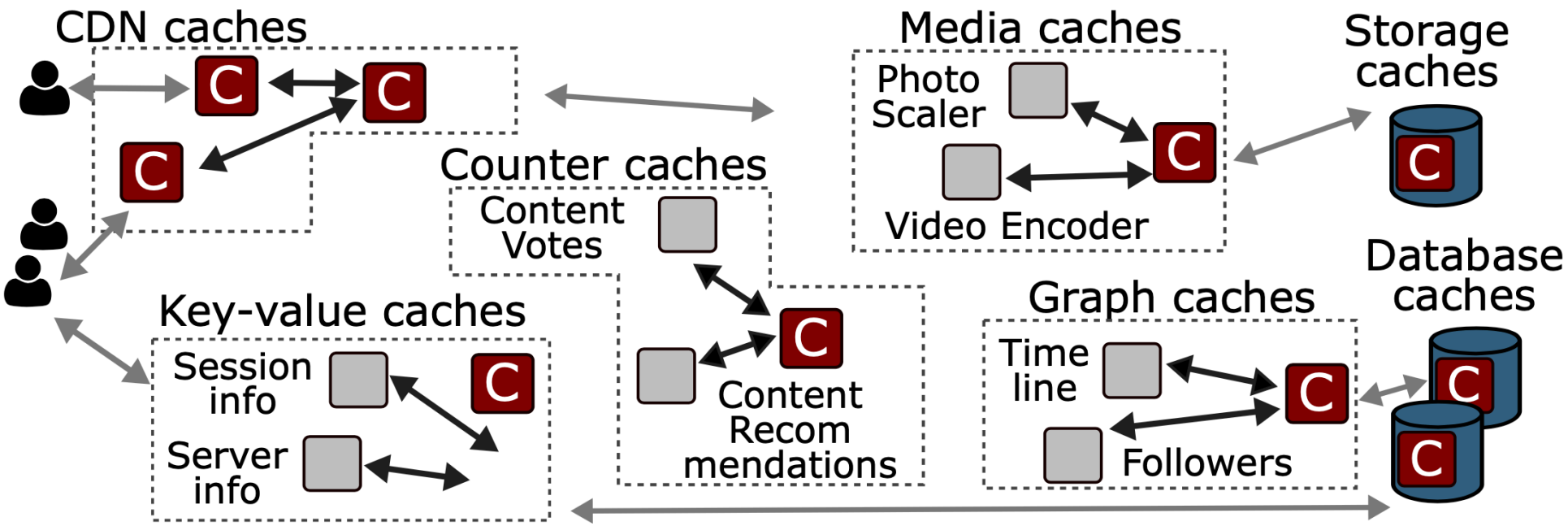
The *CacheLib* Caching Engine: Design and Experiences at Scale

Many Authors



Caching is Used in a Diverse Array of Systems

- You might be surprised at all the use cases found at Facebook

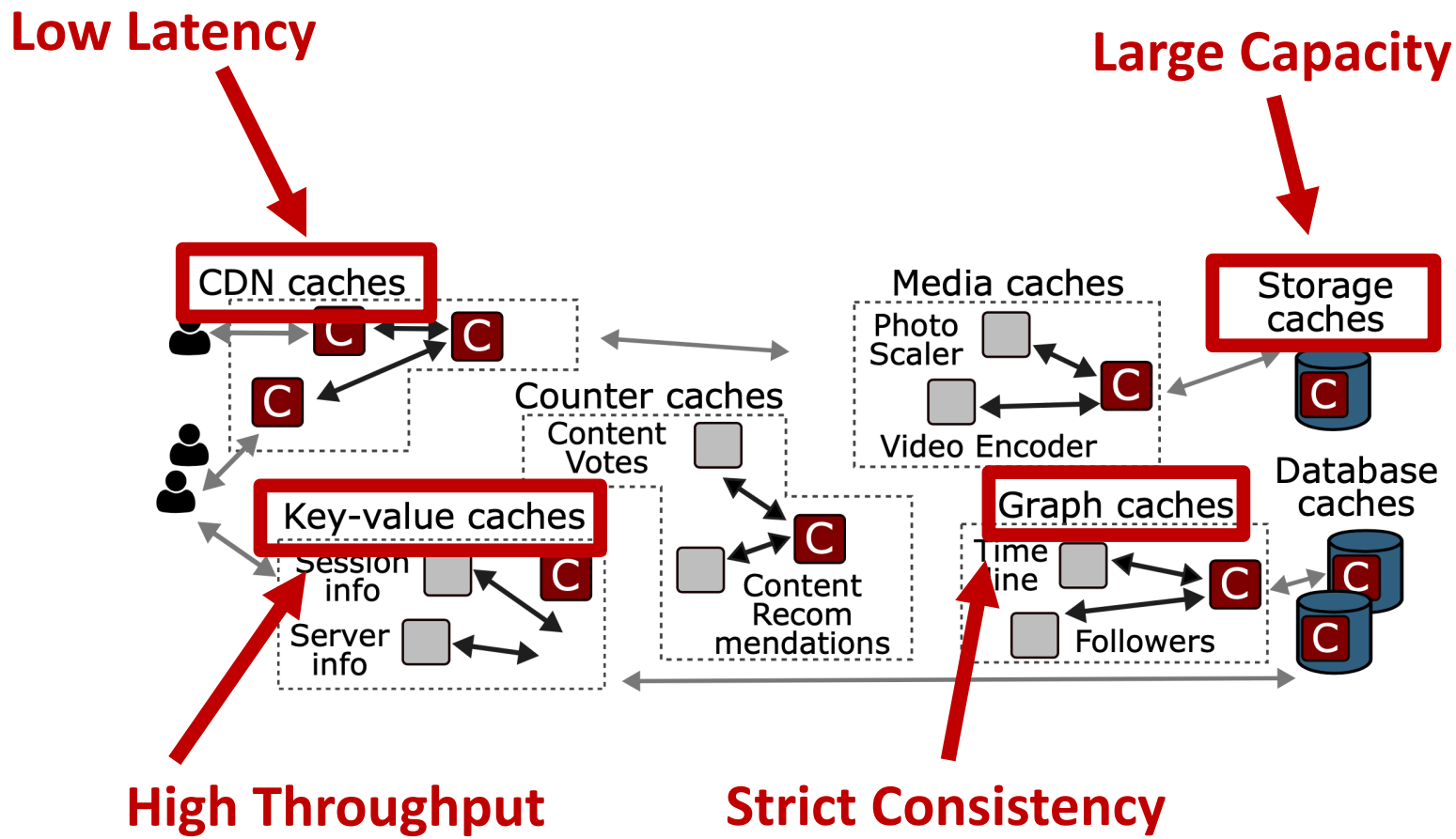


(And many more!)

Carnegie Mellon University Computer Science Department

Caching is Used in a Diverse Array of Systems

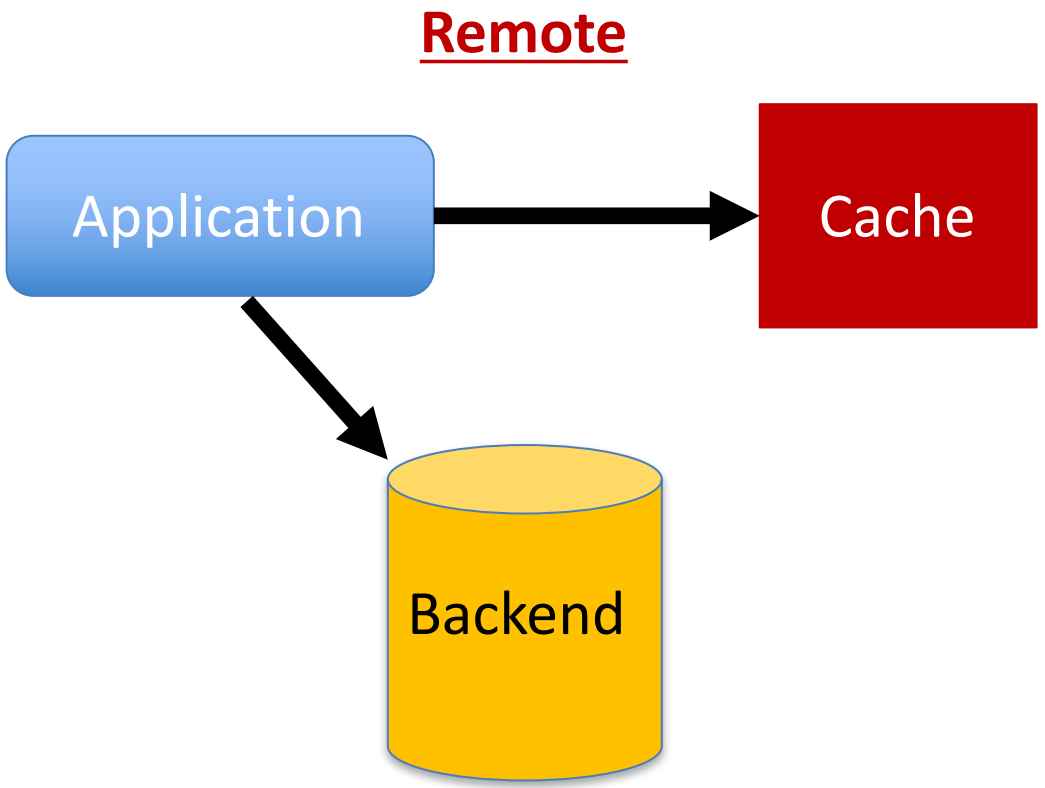
- These systems to differ along several axes:
  - Performance goals
  - System topology



Carnegie Mellon University Computer Science Department

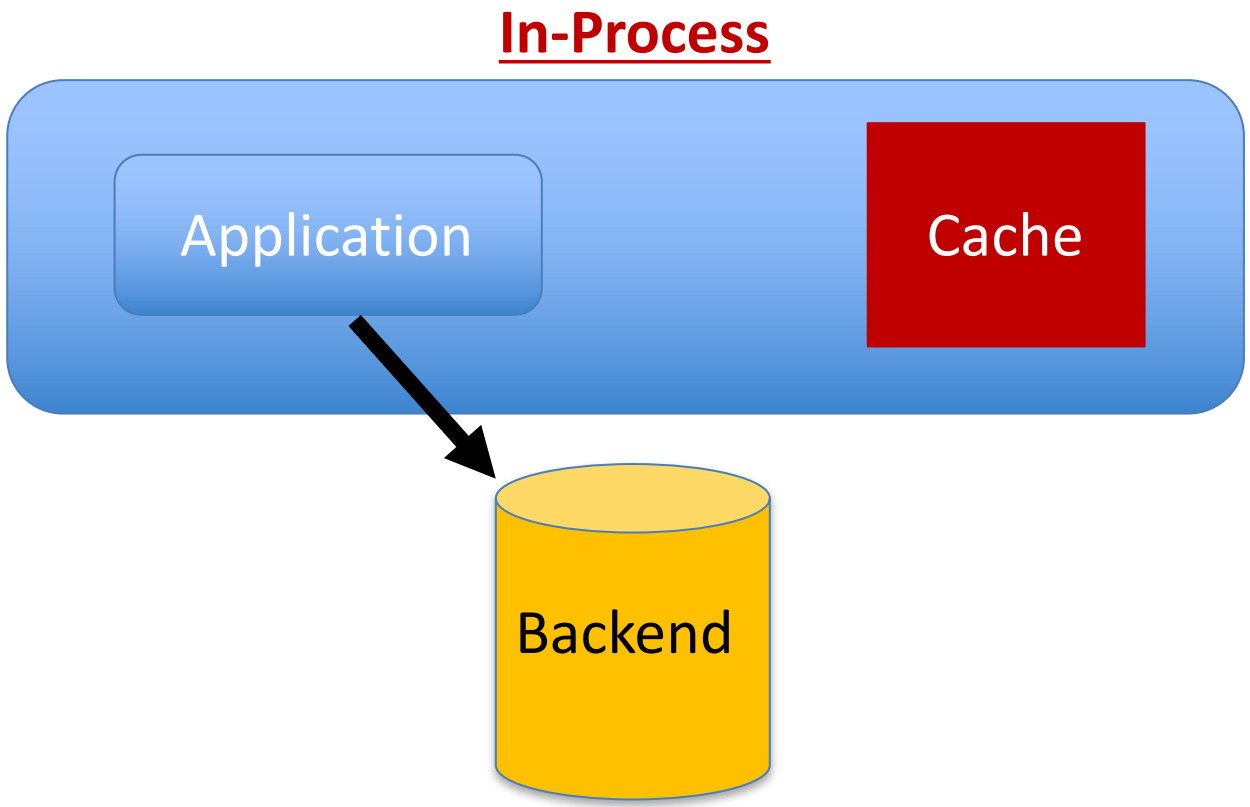
Caching is Used in a Diverse Array of Systems

- These systems to differ along several axes:
  - Performance goals
  - System topology



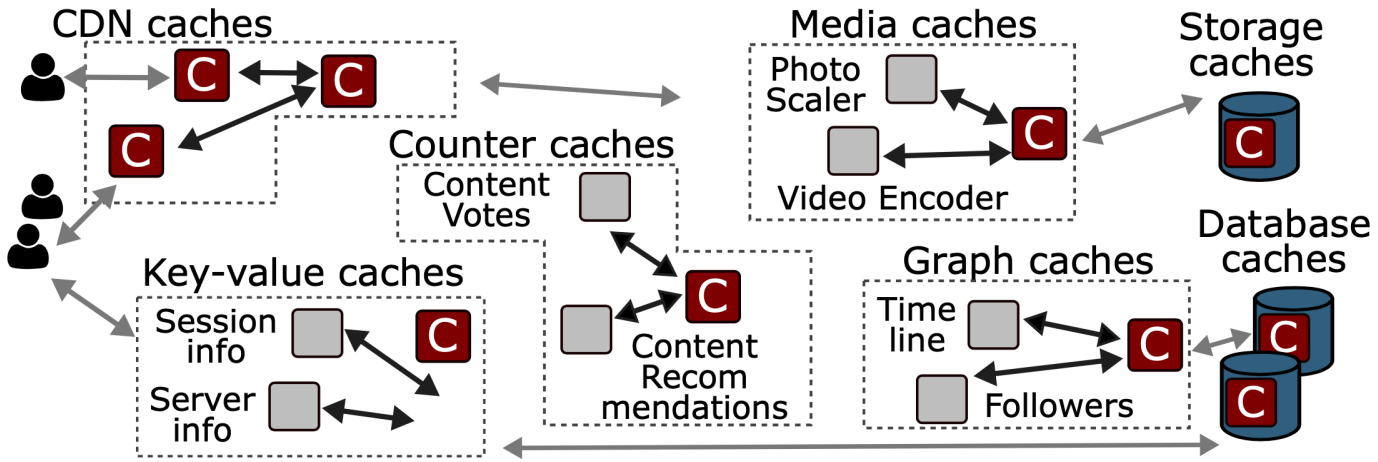
Caching is Used in a Diverse Array of Systems

- These systems to differ along several axes:
  - Performance goals
  - System topology



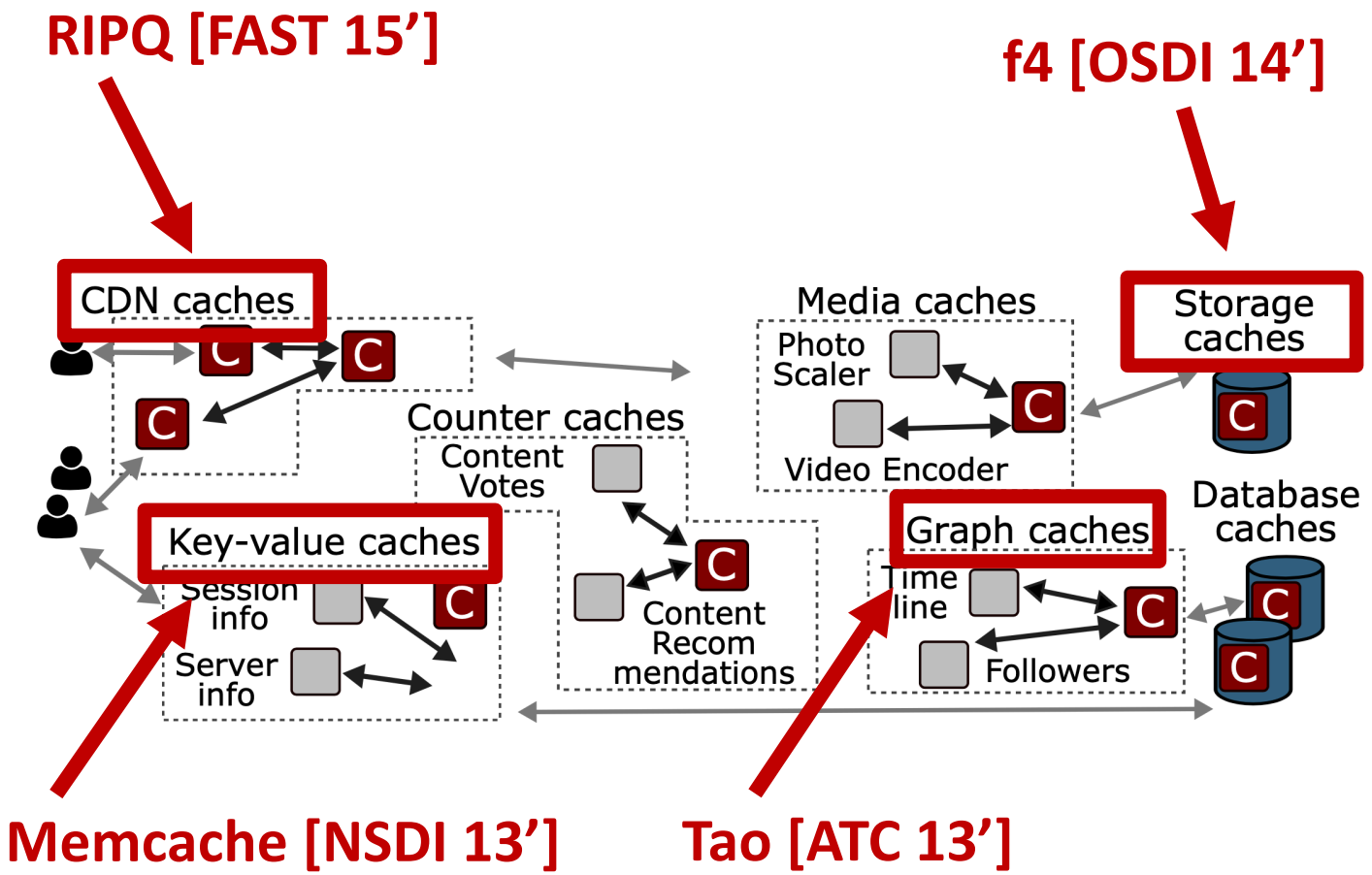
Caching is Used in a Diverse Array of Systems

- These systems differ along several axes:
  - Performance goals
  - System topology
  - Workload
  - Domain-specific features



Specialized Caching is the State-of-the-art

- Historically, Facebook maintained ***specialized*** caching implementations
- Long tradition of specialization in academia
  - Distcache, Kvell, Cliffhanger, many more



Specialized Caching is the State-of-the-art

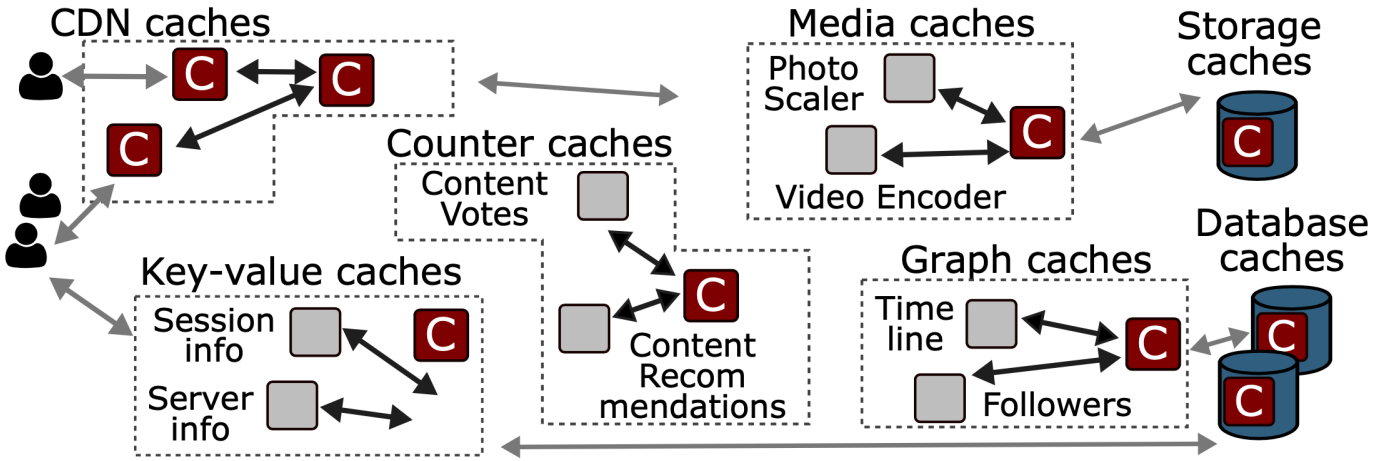
- Historically, Facebook maintained ***specialized*** caching implementations

- Long tradition of specialization in academia
  - Distcache, Kvell, Cliffhanger, many more

Problem:

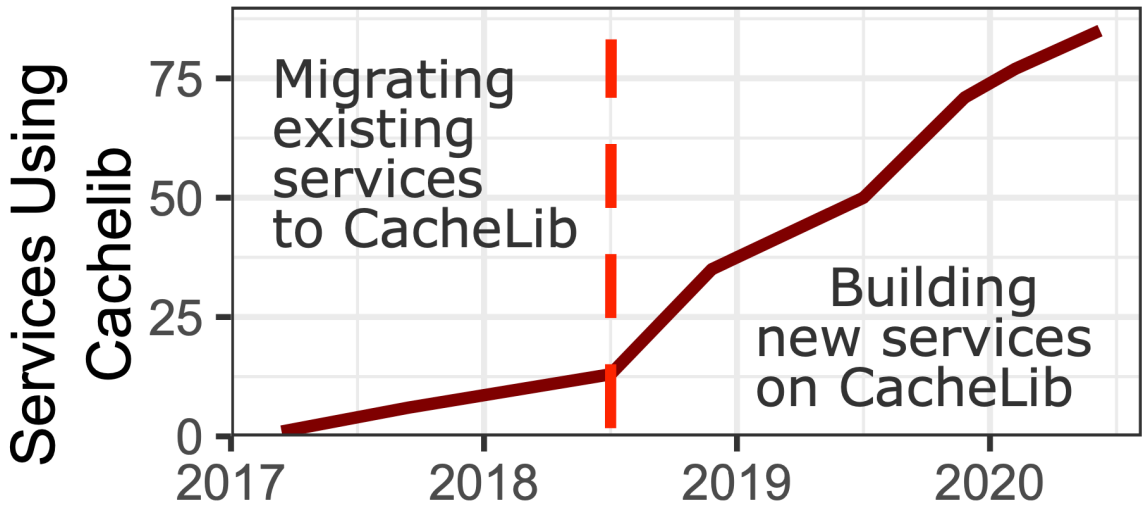
Hard to maintain an increasing number of specialized implementations

- Redundant code
- Narrow feature sets
- Barrier to implementing new ideas



Solution: CacheLib Caching Engine

- CacheLib is a widely used, **general-purpose** caching engine
  - Enables high-capacity caches
  - Provides a rich feature set
  - Aggregates optimizations
- Widely adopted at Facebook
  - replaced many specialized implementations

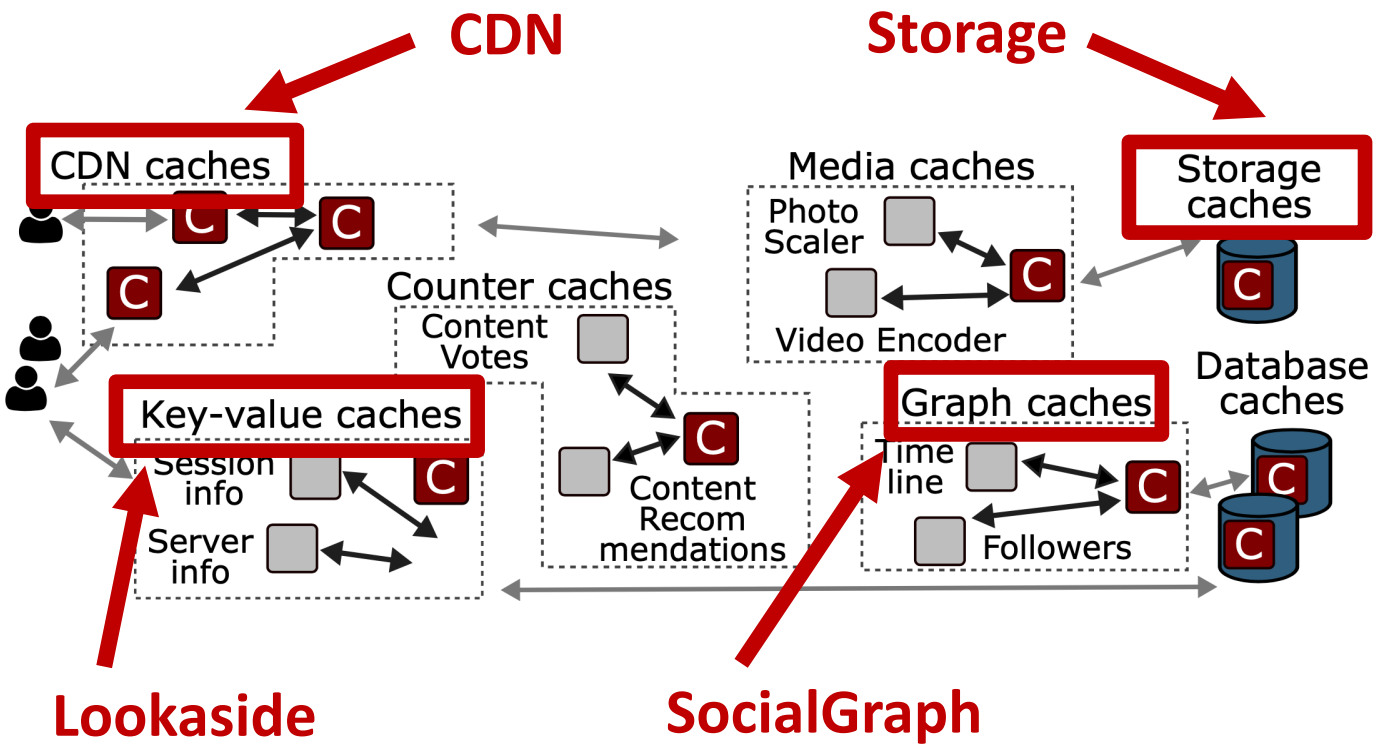


The CacheLib Caching Engine

- Common challenges/characteristics of caching systems
- Design of CacheLib
- CacheLib outperforms specialized implementations
- Lessons learned from deploying CacheLib in production

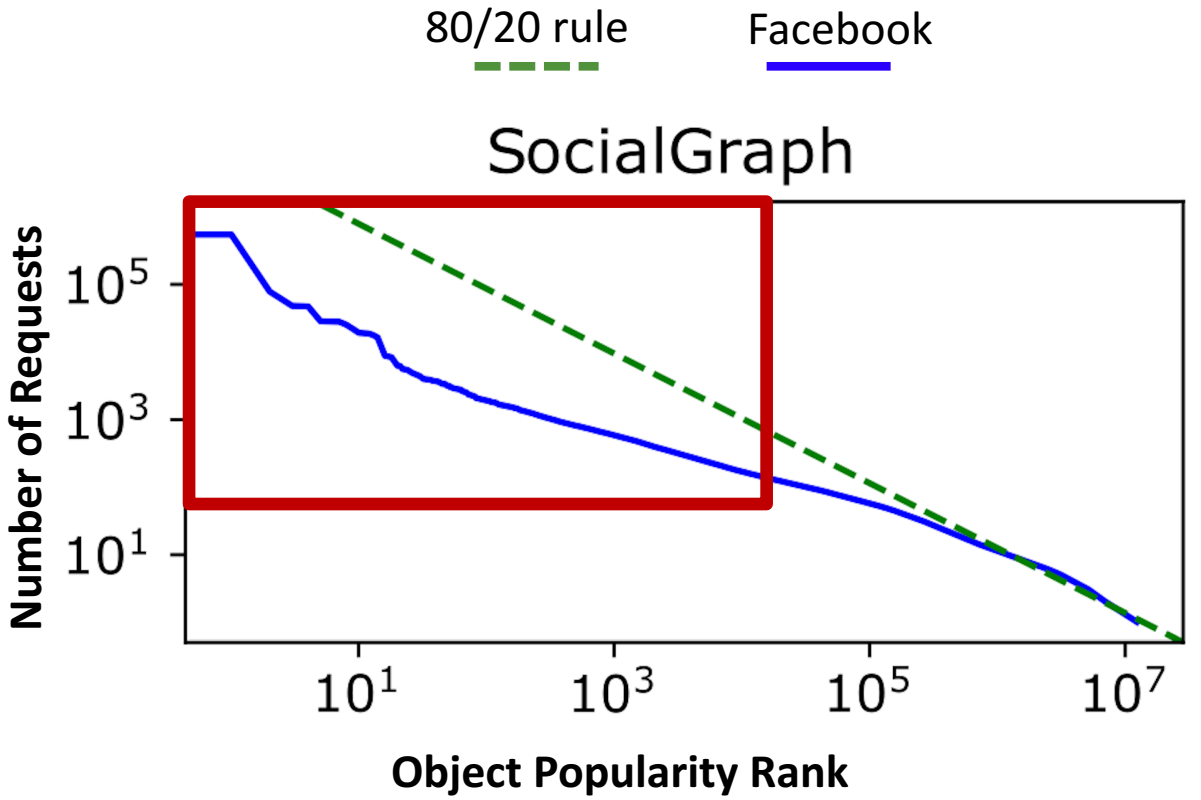
Identifying Common Challenges in Caching

Largest 4 caching systems at Facebook



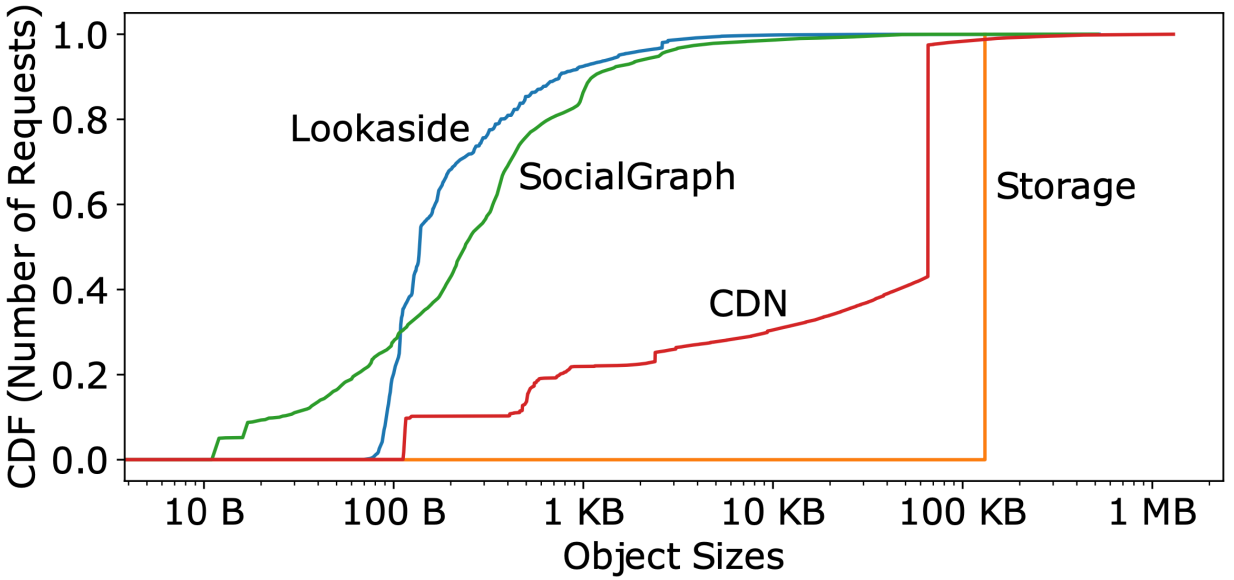
Popularity Distributions are Diffuse

- Request popularities are roughly assumed to follow the “80/20” rule
  - For SocialGraph: “50/20 rule”
  - For Storage: “40/20 rule”
  - For CDN “60/20 rule”
- Low popularity of hot objects → low hit ratio
- **Need large cache capacities**



Object Sizes are Highly Variable

- Object sizes are highly variable
- Small object sizes are common
  - Memcached: 56 B per object
  - MemC3 [NSDI 13'']: 40 B per object
  - 1 TB of 100B objects?
    - 256 GB DRAM overhead
- **Caches need low per-object overhead, ability to index billions of objects**



Common Challenges of the Production Environment

- Are there shared challenges of a “real deployed system”?

- Stability requirements for production caching systems
  - Bursty traffic
  - Frequent code updates / restarts

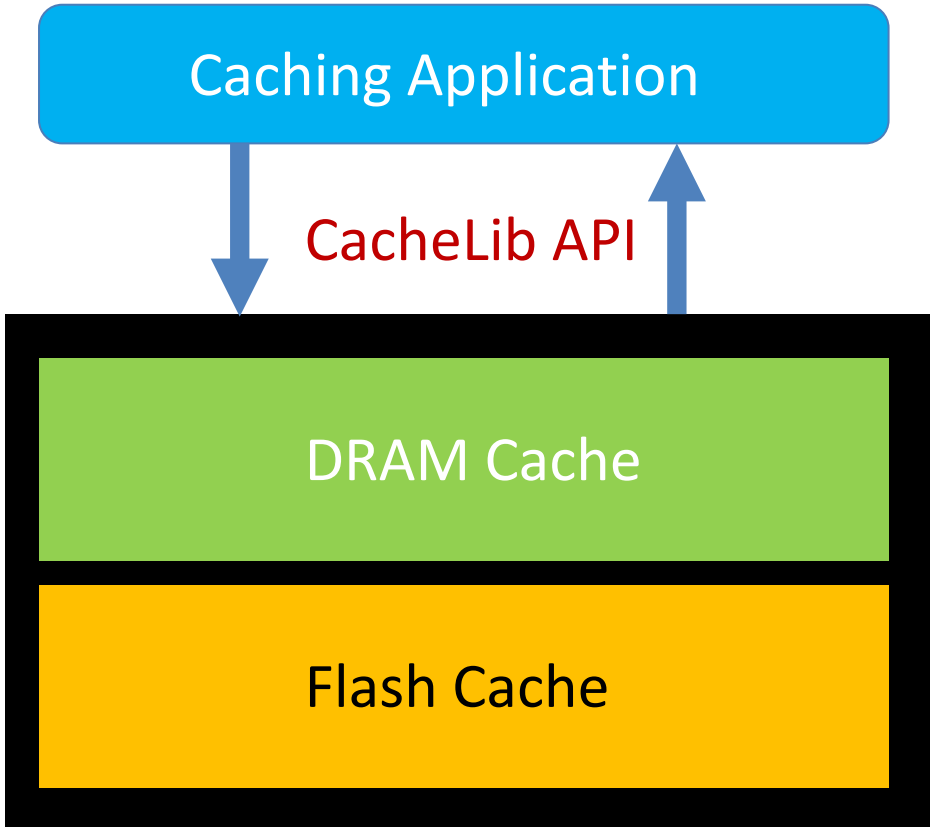
- **Solution: These challenges could be addressed once by a unified caching implementation**

The CacheLib Caching Engine

- Common challenges/characteristics of caching systems
  - Large cache capacity, low overhead, production features
- Design of CacheLib
- CacheLib outperforms specialized implementations
- Lessons learned from deploying CacheLib in production

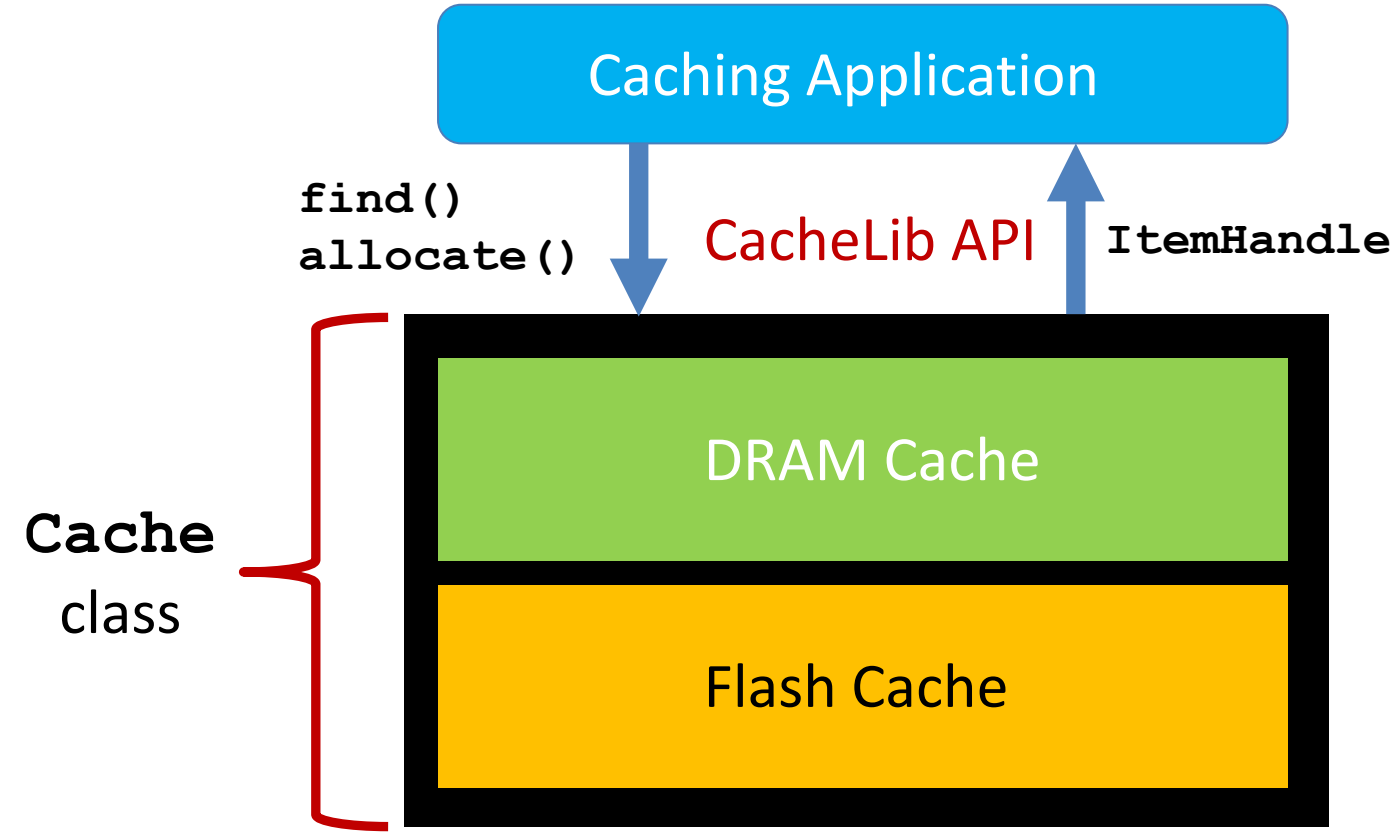
Caching Engine Requirements

- Want a library of customizable cache components
  - Easy for programmers (simple, expressive API)
- To accommodate workloads:
  - Transparent **hybrid DRAM-flash** caches for large capacity
  - Approximate indexes over billions of small objects
- For production deployment:
  - Sufficient single-machine throughput
  - Broad feature set



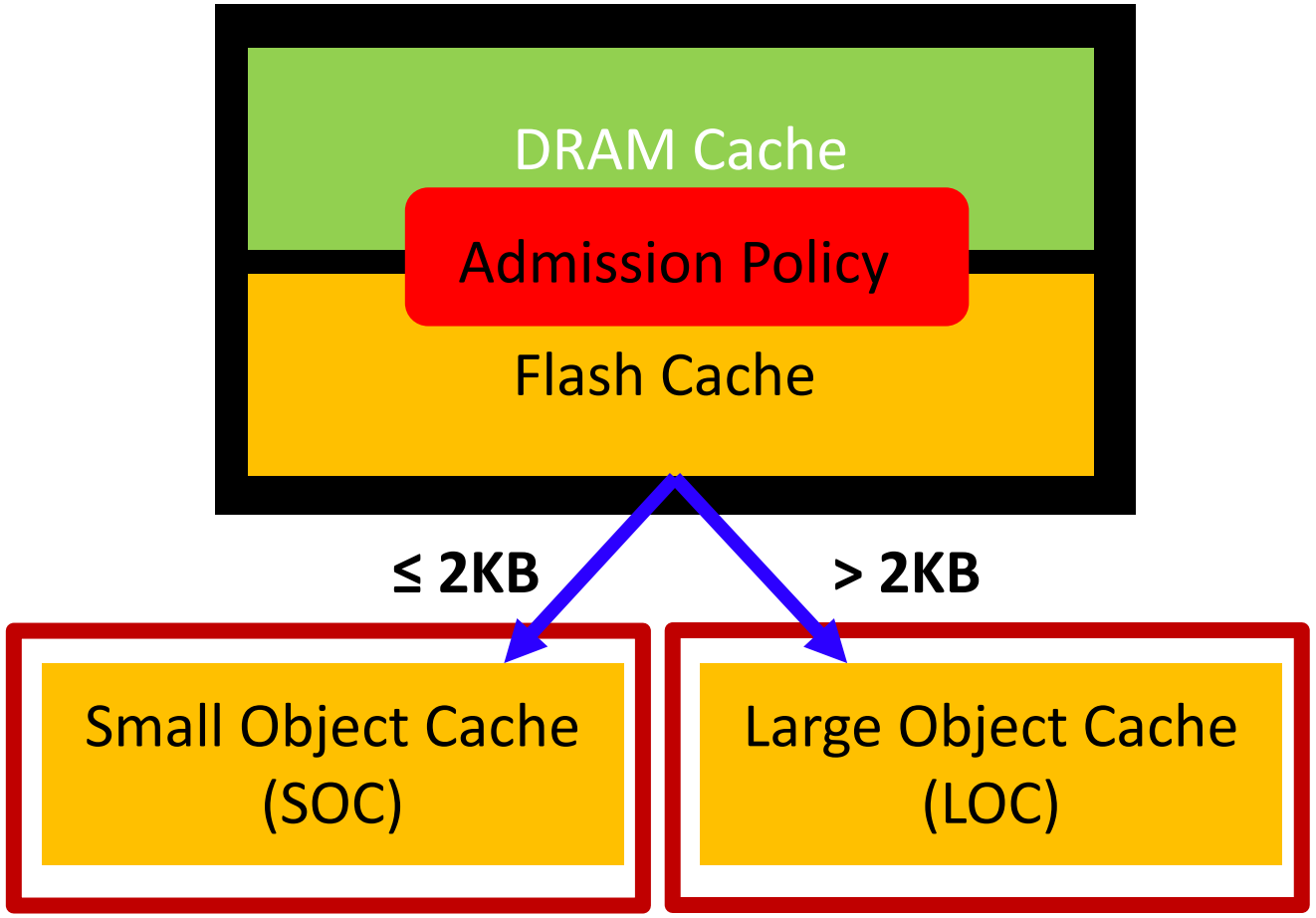
The CacheLib API

- Uniform, thread-safe API
  - Decoupled from cache configuration
  - Easy to build highly-concurrent, high throughput caches
  - Applications not tightly coupled to storage medium (DRAM, flash)



CacheLib's Caching Implementation

- DRAM uses chained hash table
  - 31B per object
- Flash cache partitioned by size
  - $< 0.2\%$ overhead in practice
- Flash has limited write endurance
  - **Admission** policies
  - Reduce **write amplification**



Carnegie Mellon University Computer Science Department

CacheLib's Broad Feature Set

	Hybrid Cache	Warm Restarts	Negative Caching	Natively Structed Items
		Persistent cache across restarts	Cache empty results with no overhead	Optimized caching of data structures
CacheLib	✓	✓	✓	Arrays/Maps

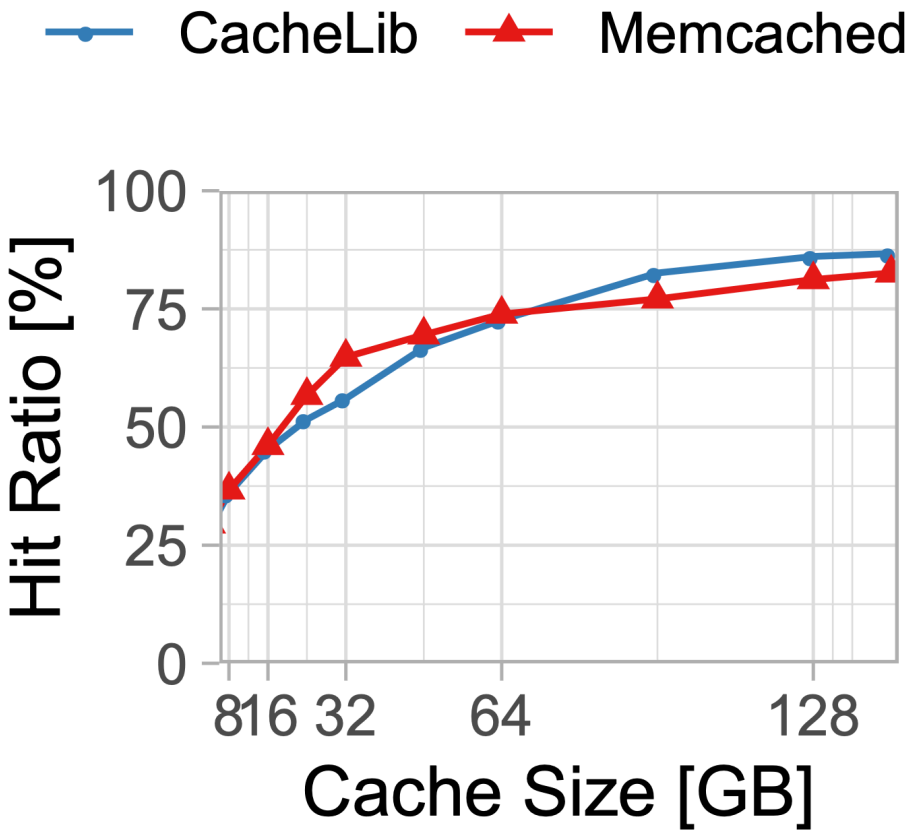
The CacheLib Caching Engine

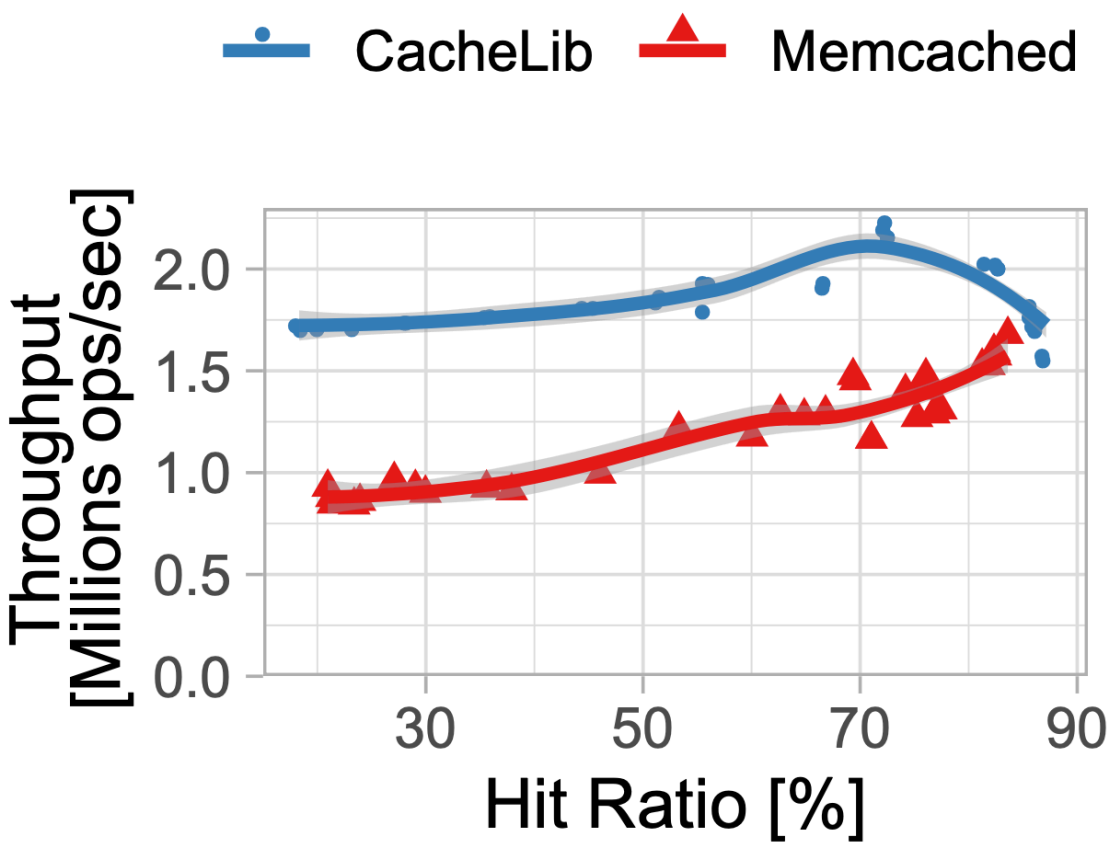
- Common challenges/characteristics of caching systems
- Design of CacheLib
- CacheLib outperforms specialized implementations
- Lessons learned from deploying CacheLib in production

Existing Systems Do Not Replicate CacheLib

	Hybrid Cache	Warm Restarts	Negative Caching	Natively Structed Items
CacheLib	✓	✓	✓	Arrays/Maps
Memcached	✓	X	X	X
Redis	X	X	X	Many
MemC3	X	X	X	X
Flashield	✓	X	X	X
Apache Traffic Server	✓	X	✓	X
Varnish	X	X	X	X
FlashCache	✓	✓	X	X
Flashtier	✓	✓	X	X

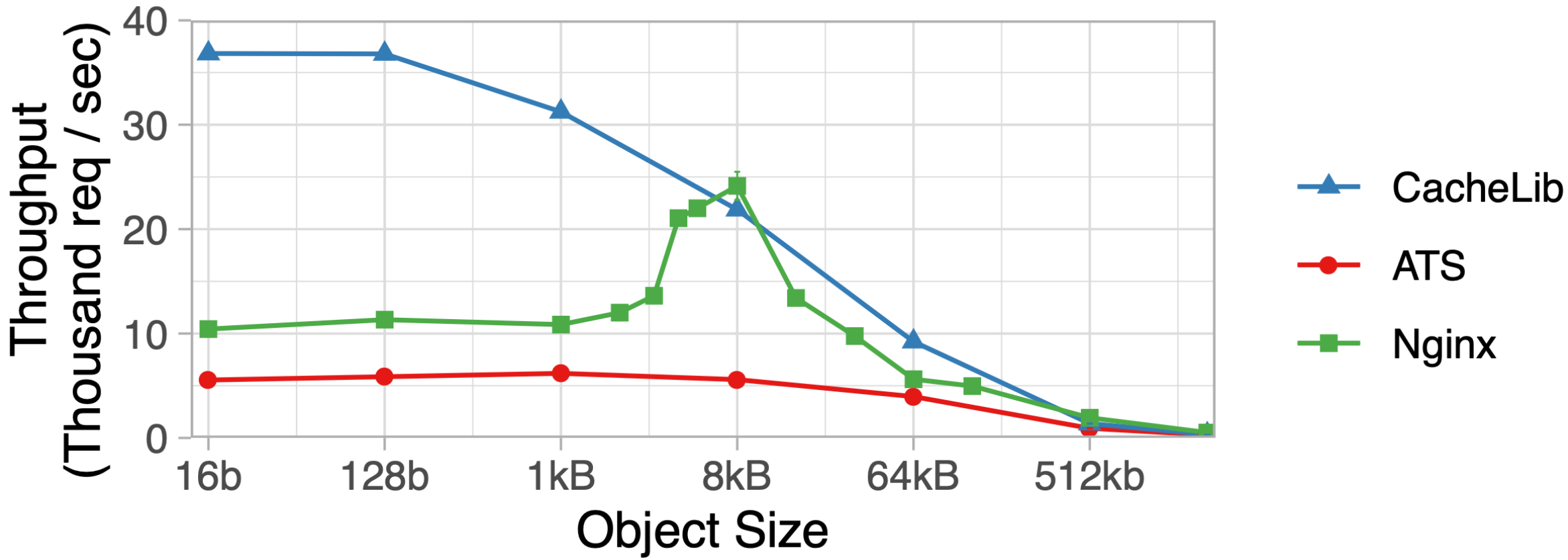
Lookaside Caching: CacheLib Outperforms Memcached





Carnegie Mellon University
Computer Science Department

HTTP Server Caching: CacheLib Outperforms NGINX/ATS

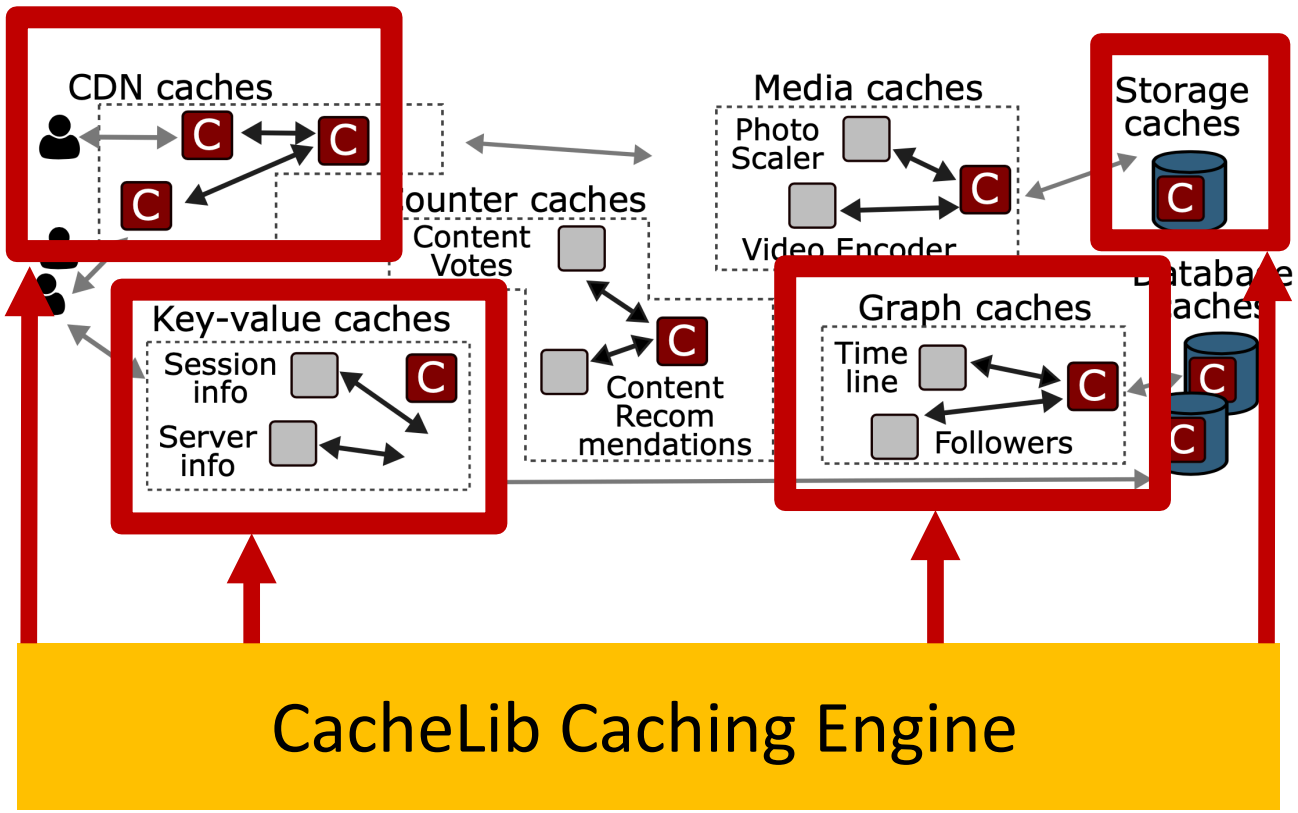


The CacheLib Caching Engine

- Common challenges/characteristics of caching systems
- Design of CacheLib
- CacheLib outperforms specialized implementations
- Lessons learned from deploying CacheLib in production

CacheLib is an Aggregation Point for Optimizations

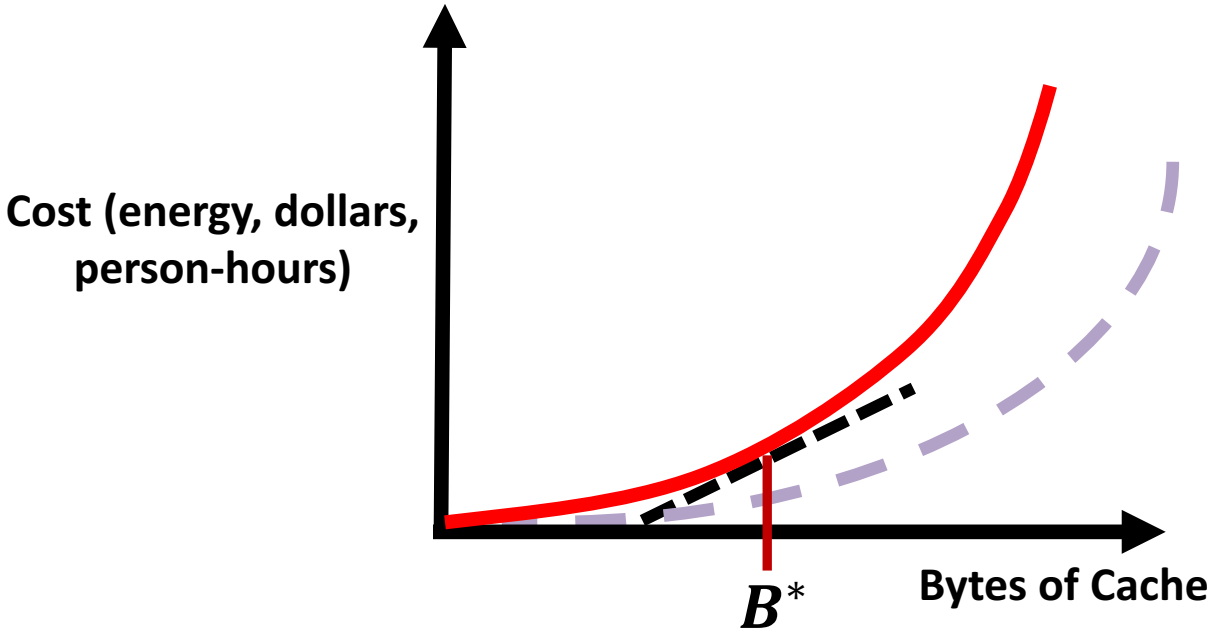
- Specialized implementations enable localized improvements
- CacheLib exports optimizations to all use cases
  - Example: Optimizing the LOC for CDN
  - Hybrid cache performance improved **everywhere**

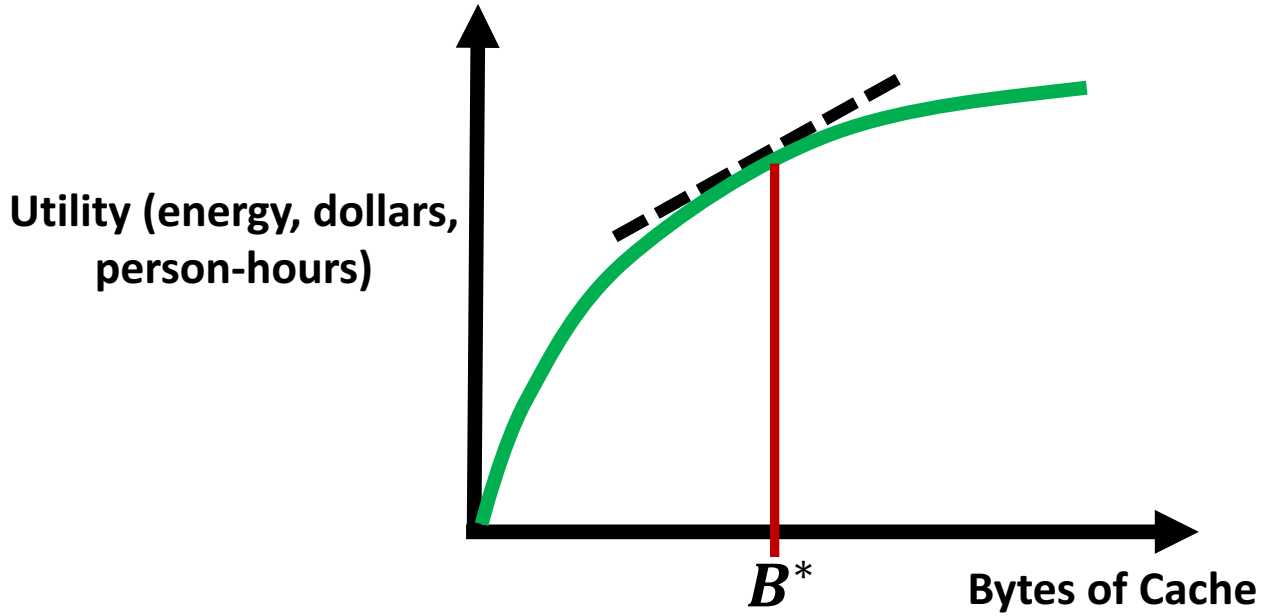


Carnegie Mellon University Computer Science Department

CacheLib Reduces the "Cost" of Caching

- Typical calculation in provisioning a cache:

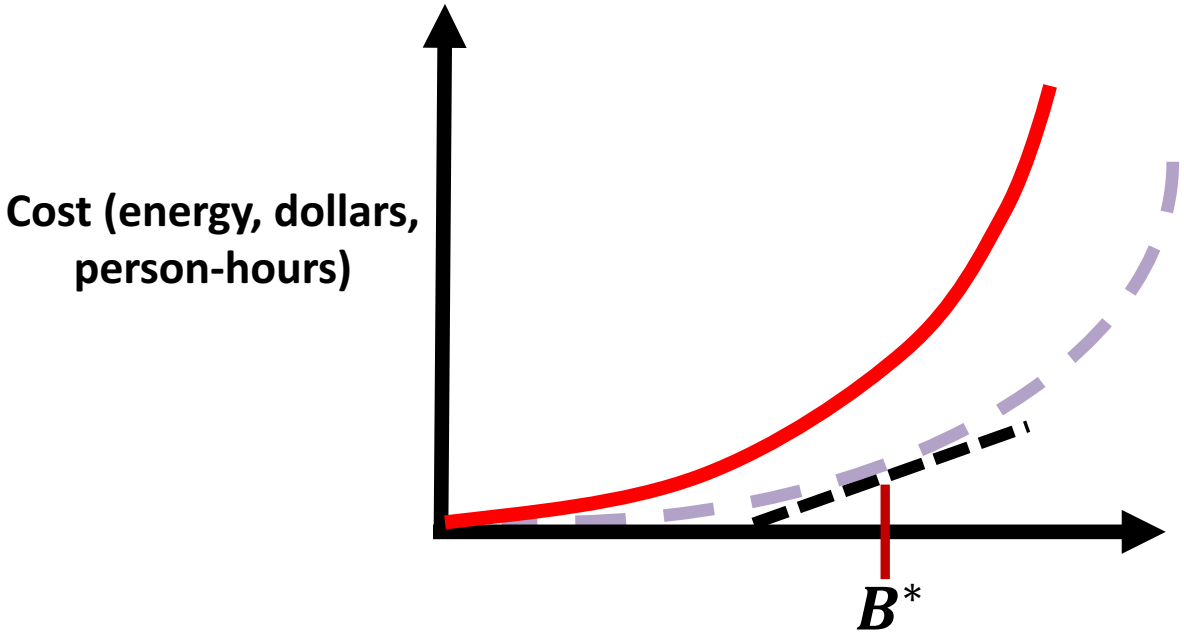


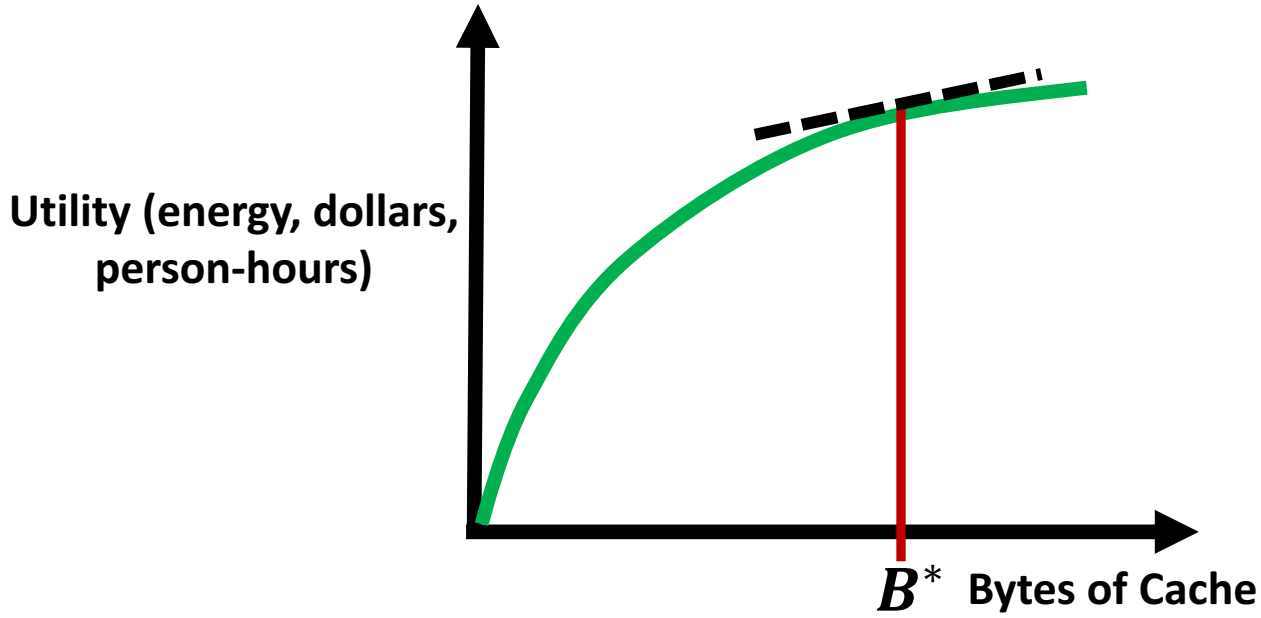


Set marginal cost equal to marginal benefit

CacheLib Reduces the "Cost" of Caching

- Typical calculation in provisioning a cache:





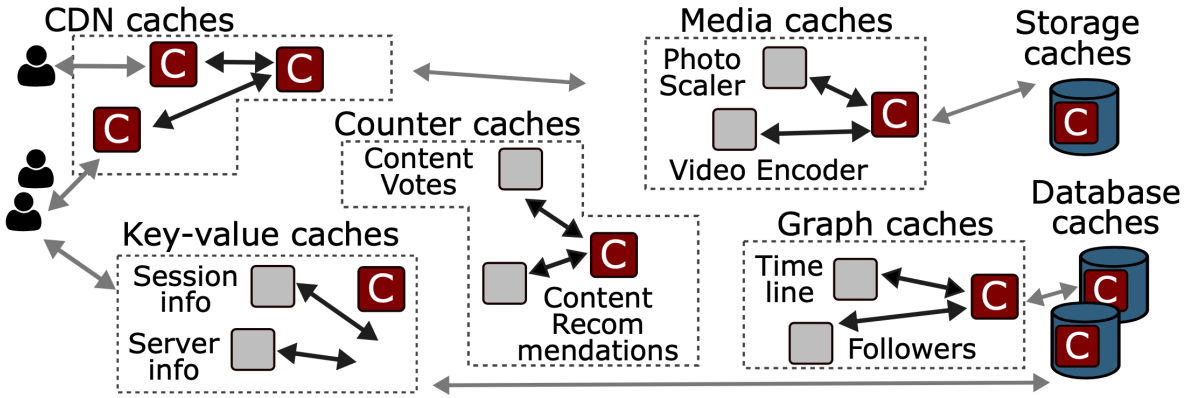
Set marginal cost equal to marginal benefit

Impact

- Cache capacities are growing
- Number of caches is growing

Conclusion

Historically cache implementations were **specialized**



Problem:

Hard to maintain an increasing number of specialized implementations

- Redundant code
- Narrow feature sets
- Barrier to implementing new ideas

Solution: CacheLib, a widely used **general-purpose** caching engine

- Extracts common caching functionality
- Aggregates optimizations
- Reduces the "cost" of caching
- Widely used at Facebook

Carnegie Mellon University Computer Science Department

Thank you!

- Contact the authors:
  - Benjamin Berg: bsberg@cs.cmu.edu

- See www.cachelib.org for more information