HashCache: Cache Storage for the Next Billion

Anirudh Badam

KyoungSoo Park   Vivek S. Pai   Larry L. Peterson

Princeton University

Next Billion Internet Users

Next Billion Internet Users

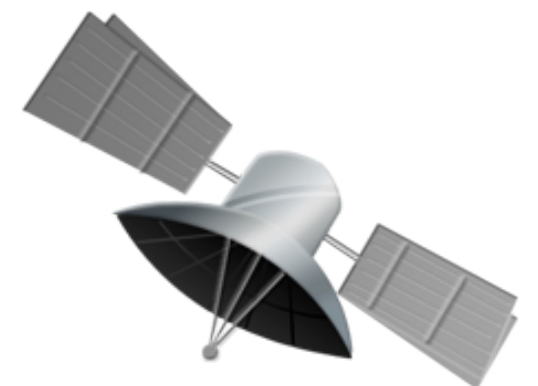
- Schools, urban middle class in developing regions

Next Billion Internet Users

- Schools, urban middle class in developing regions
- Affordable hardware
  - OLPC and Intel Classmate

Next Billion Internet Users

$200



$1500
per month

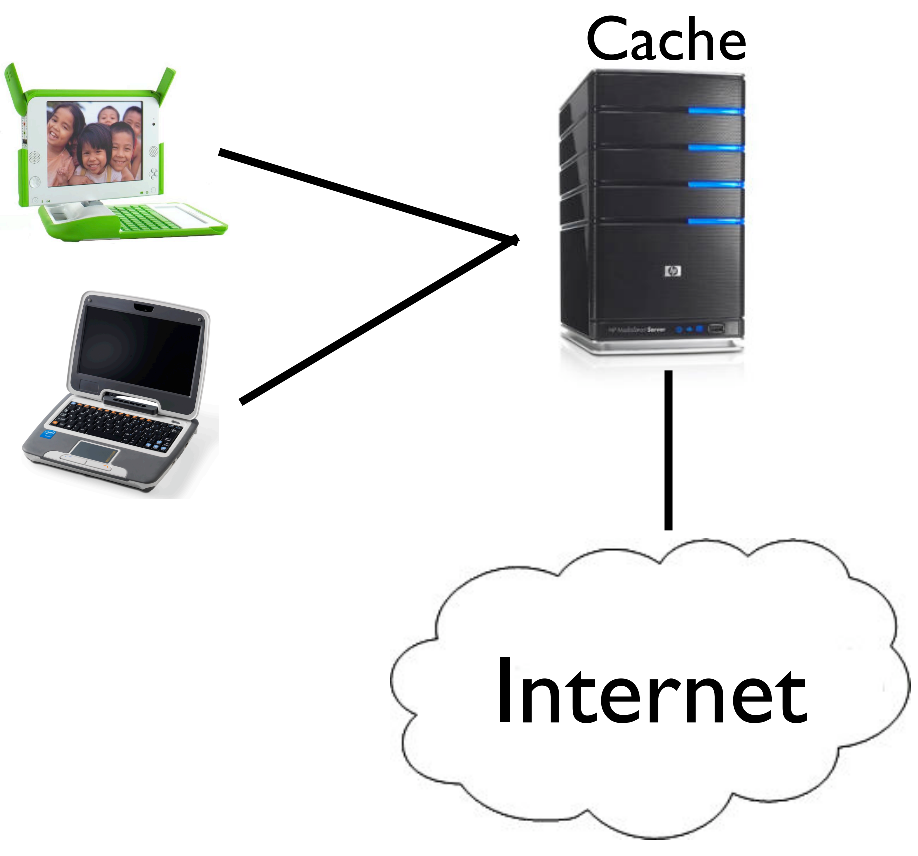
- Schools, urban middle class in developing regions
- Affordable hardware
  - OLPC and Intel Classmate
- Expensive Internet
  - $1500+ per month per Mbps
  - Unlikely to improve in the near future

Bandwidth Saving

Bandwidth Saving

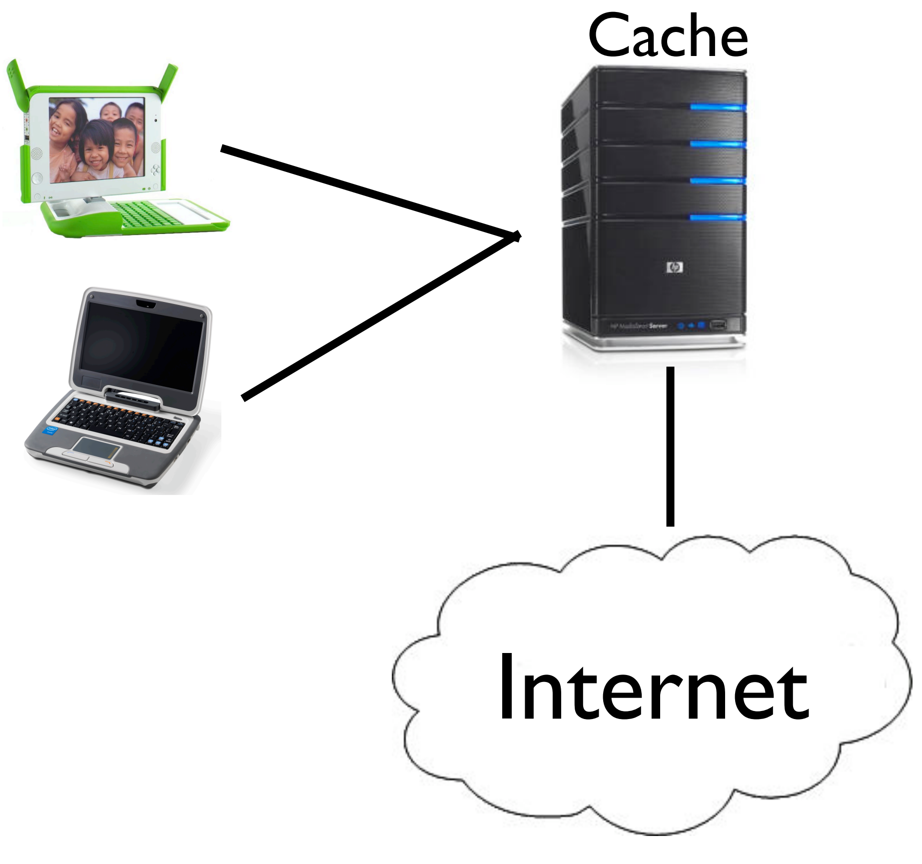
- Connectivity is a precious resource

Bandwidth Saving



- Connectivity is a precious resource
  - Saving bandwidth important
  - Disk caches reduce network bandwidth requirement

Bandwidth Saving



- Connectivity is a precious resource
  - Saving bandwidth important
  - Disk caches reduce network bandwidth requirement
- Good news - disk is very cheap
  - $100/TB

Why Large Caches?

Why Large Caches?

- Larger bandwidth savings
  - Refreshes cheaper than re-fetches
  - Overnight prefetch, content push from peers

Why Large Caches?

- Larger bandwidth savings
  - Refreshes cheaper than re-fetches
  - Overnight prefetch, content push from peers
- Good offline behavior
  - Preload websites
  - Enables local search

Why Large Caches?

- Larger bandwidth savings
  - Refreshes cheaper than re-fetches
  - Overnight prefetch, content push from peers
- Good offline behavior
  - Preload websites
  - Enables local search
- Save even on dynamic content
  - WAN Acceleration = packet caching

What is the Cost?

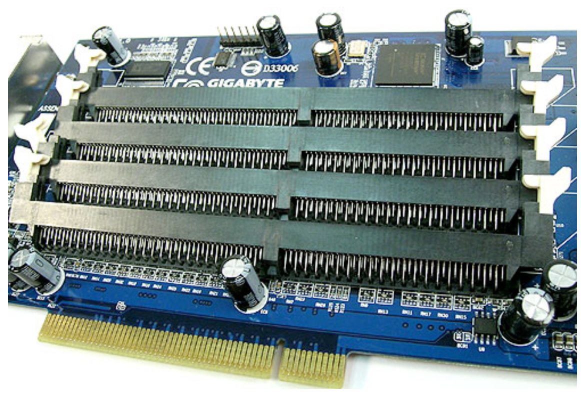
What is the Cost?

70 seeks per sec

- In-memory data structures
  - Hash table avoids seeks for misses
  - Cache replacement (LRU, etc)

What is the Cost?

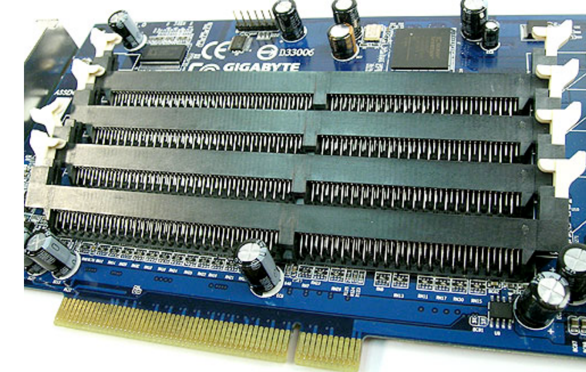
70 seeks per sec



- In-memory data structures
  - Hash table avoids seeks for misses
  - Cache replacement (LRU, etc)
- RAM index size per TB
  - Open Source (Squid) - 10 GB
  - Commercial (Tiger) - 5 GB

What is the Cost?

70 seeks per sec



x 20

- In-memory data structures
  - Hash table avoids seeks for misses
  - Cache replacement (LRU, etc)
- RAM index size per TB
  - Open Source (Squid) - 10 GB
  - Commercial (Tiger) - 5 GB
- Can not use laptops for cache
  - 2 servers, $2K each = 20 laptops

Our Solution

Our Solution

- HashCache: storage engine w/ plug-in indexing
  - 6 schemes in paper, 3 shown here
  - New Web proxy using HashCache engine

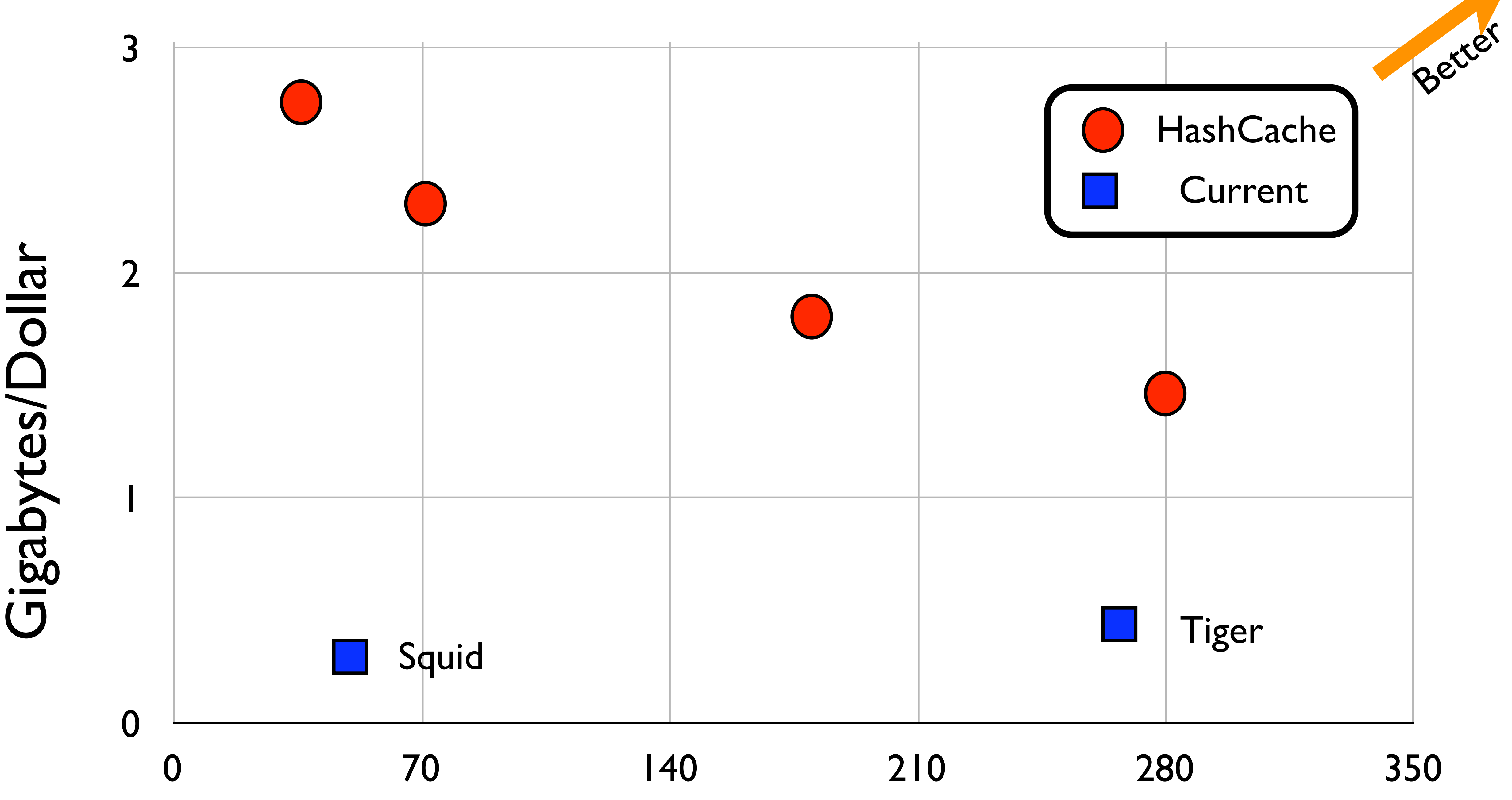
Our Solution

- HashCache: storage engine w/ plug-in indexing
  - 6 schemes in paper, 3 shown here
  - New Web proxy using HashCache engine
- Efficiency and Performance:
  - Range of policies trading speed and memory
  - 20-50x less RAM for Squid-like speed
  - 6-10x less RAM vs Tiger (Commercial)

Our Solution

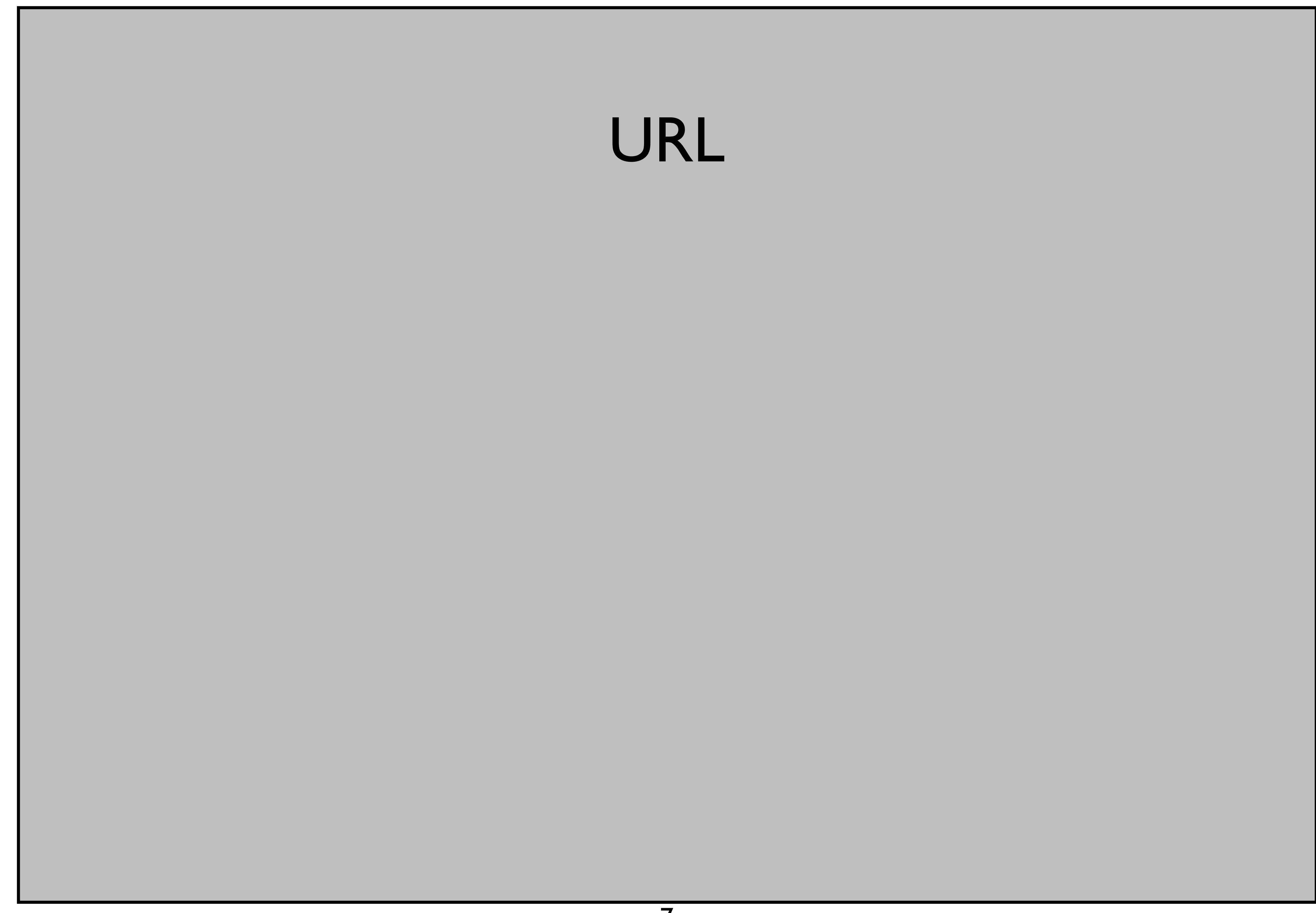
- HashCache: storage engine w/ plug-in indexing
  - 6 schemes in paper, 3 shown here
  - New Web proxy using HashCache engine
- Efficiency and Performance:
  - Range of policies trading speed and memory
  - 20-50x less RAM for Squid-like speed
  - 6-10x less RAM vs Tiger (Commercial)
- Can now use cheap laptops vs servers
  - Even for TB-sized caches

Our Solution

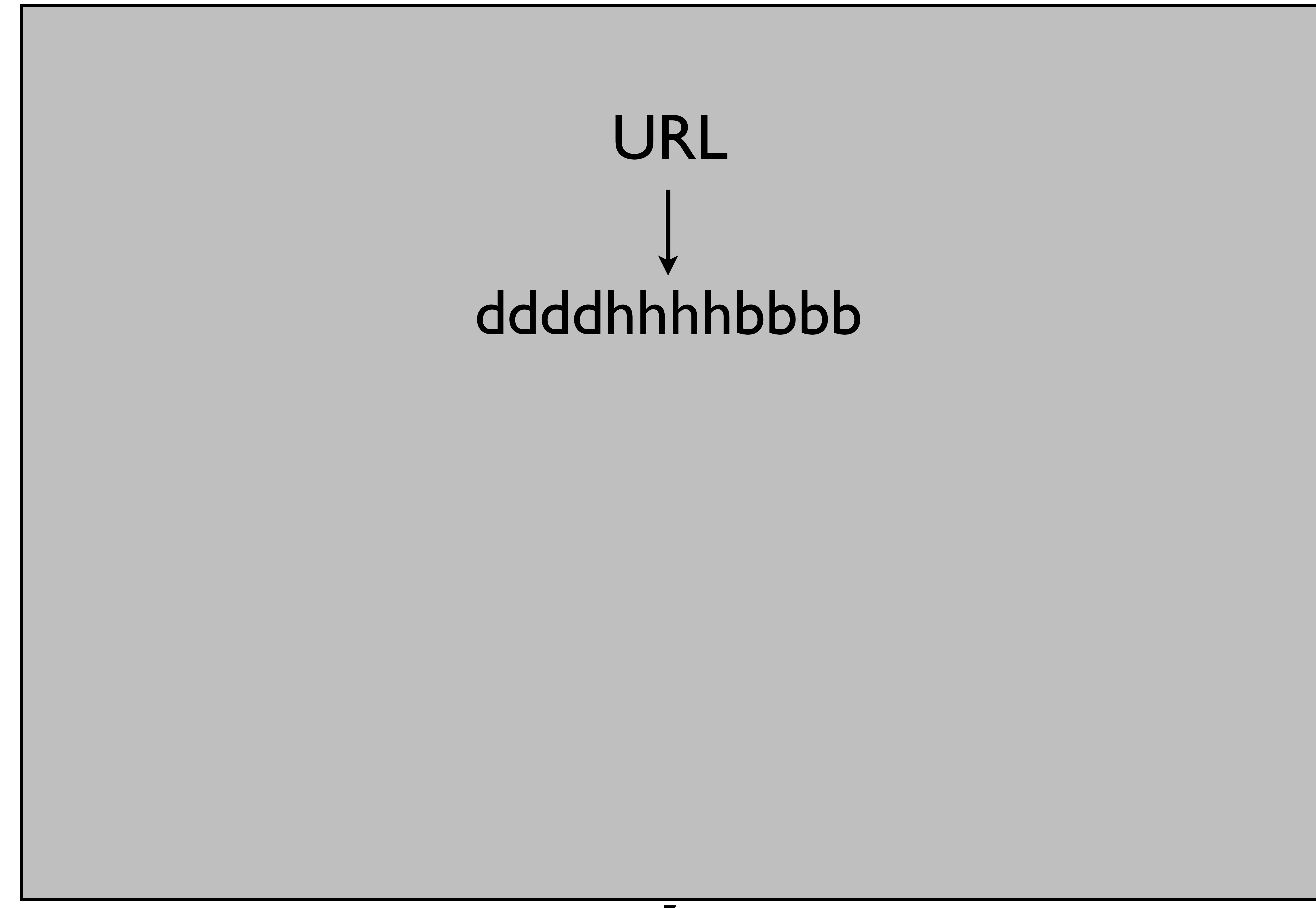


A Brief History of Cache

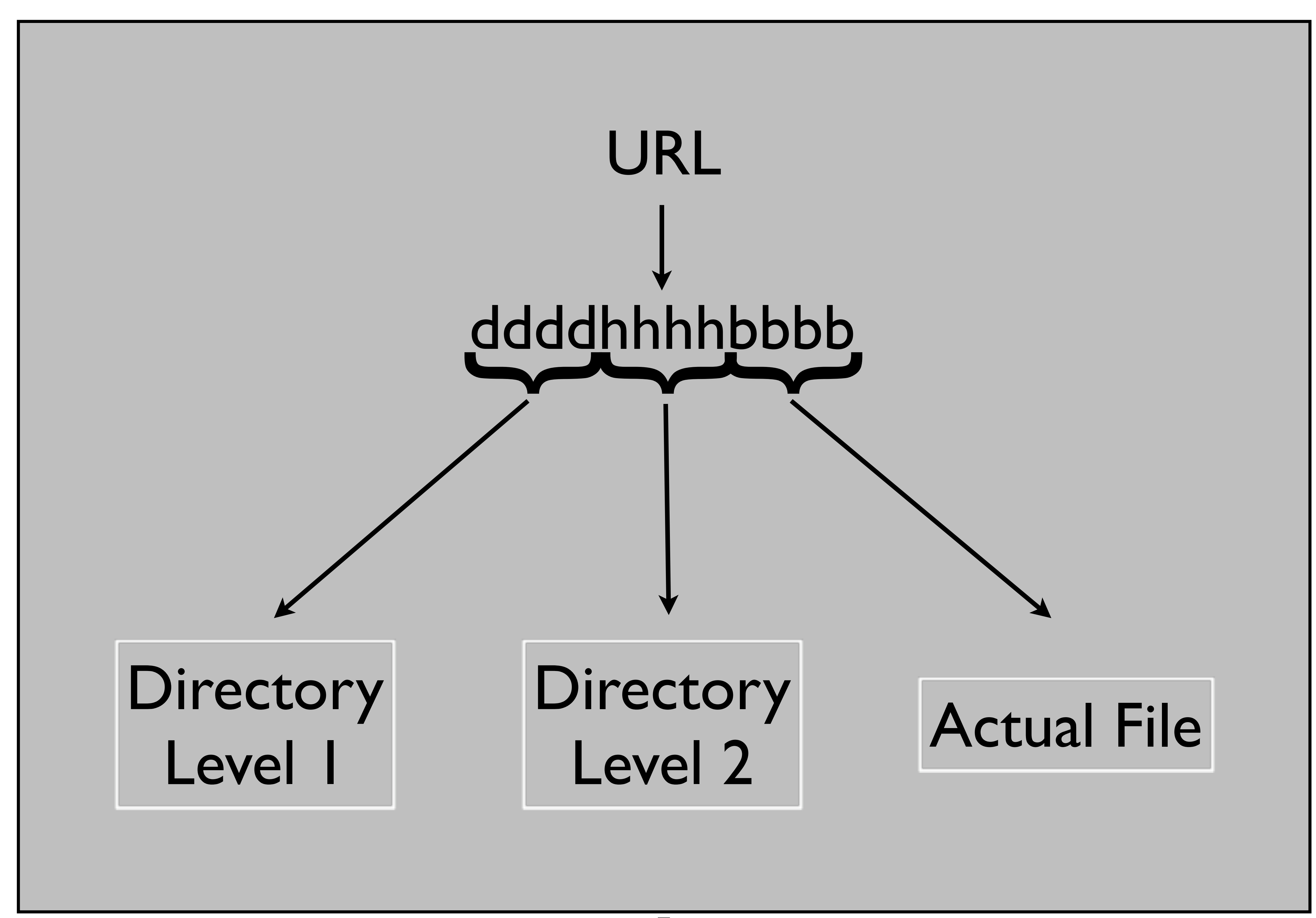
A Brief History of Cache



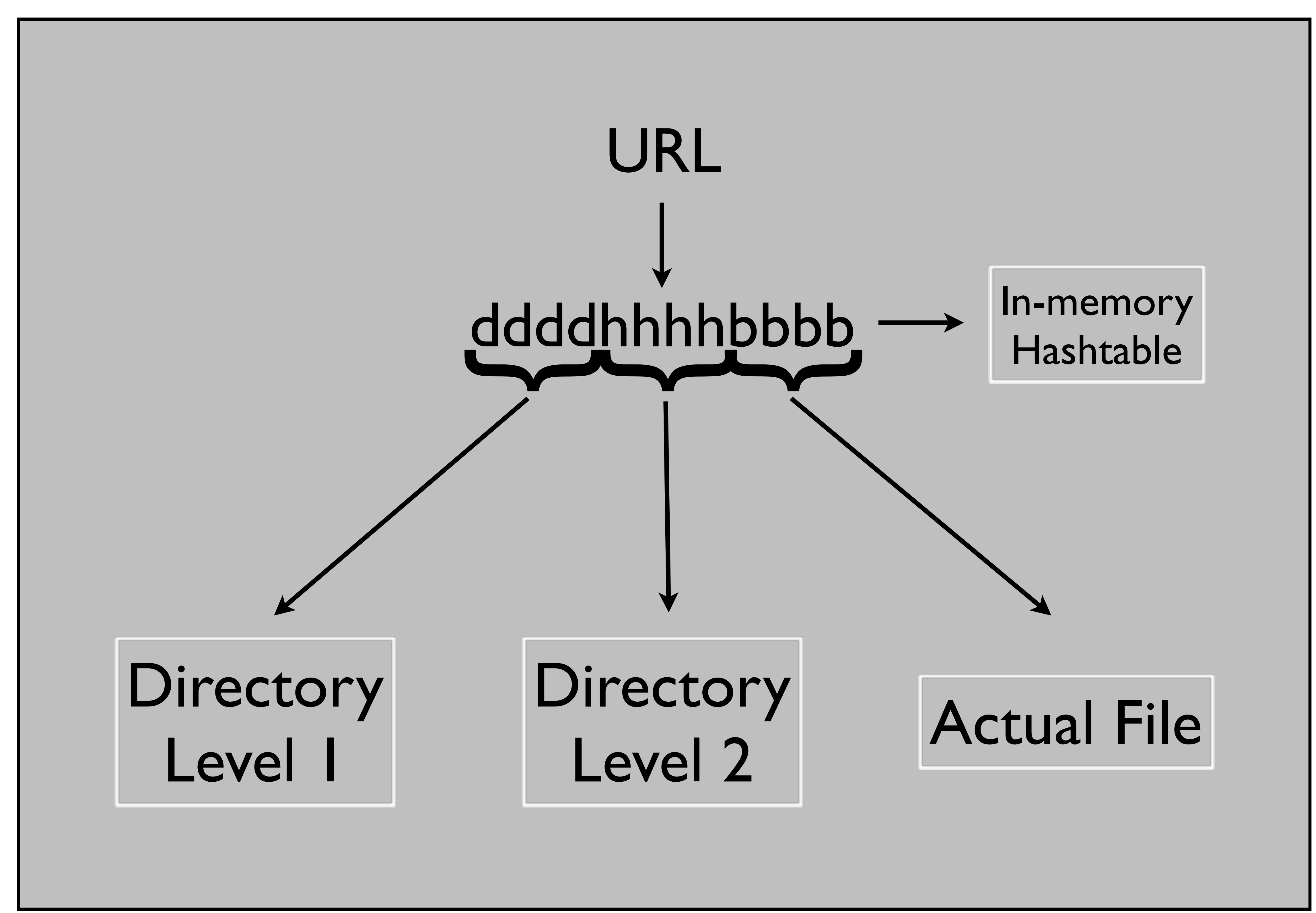
A Brief History of Cache



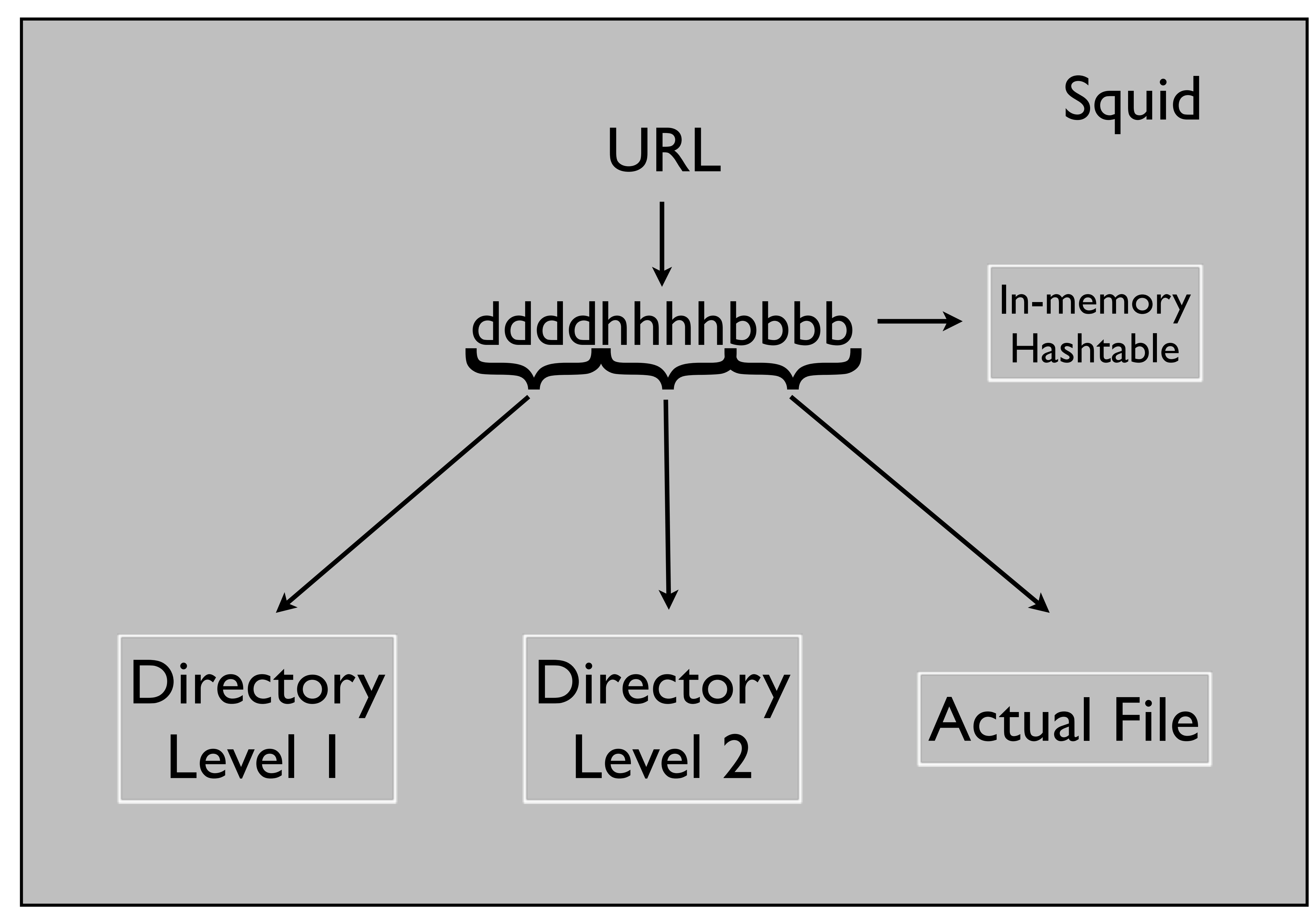
A Brief History of Cache



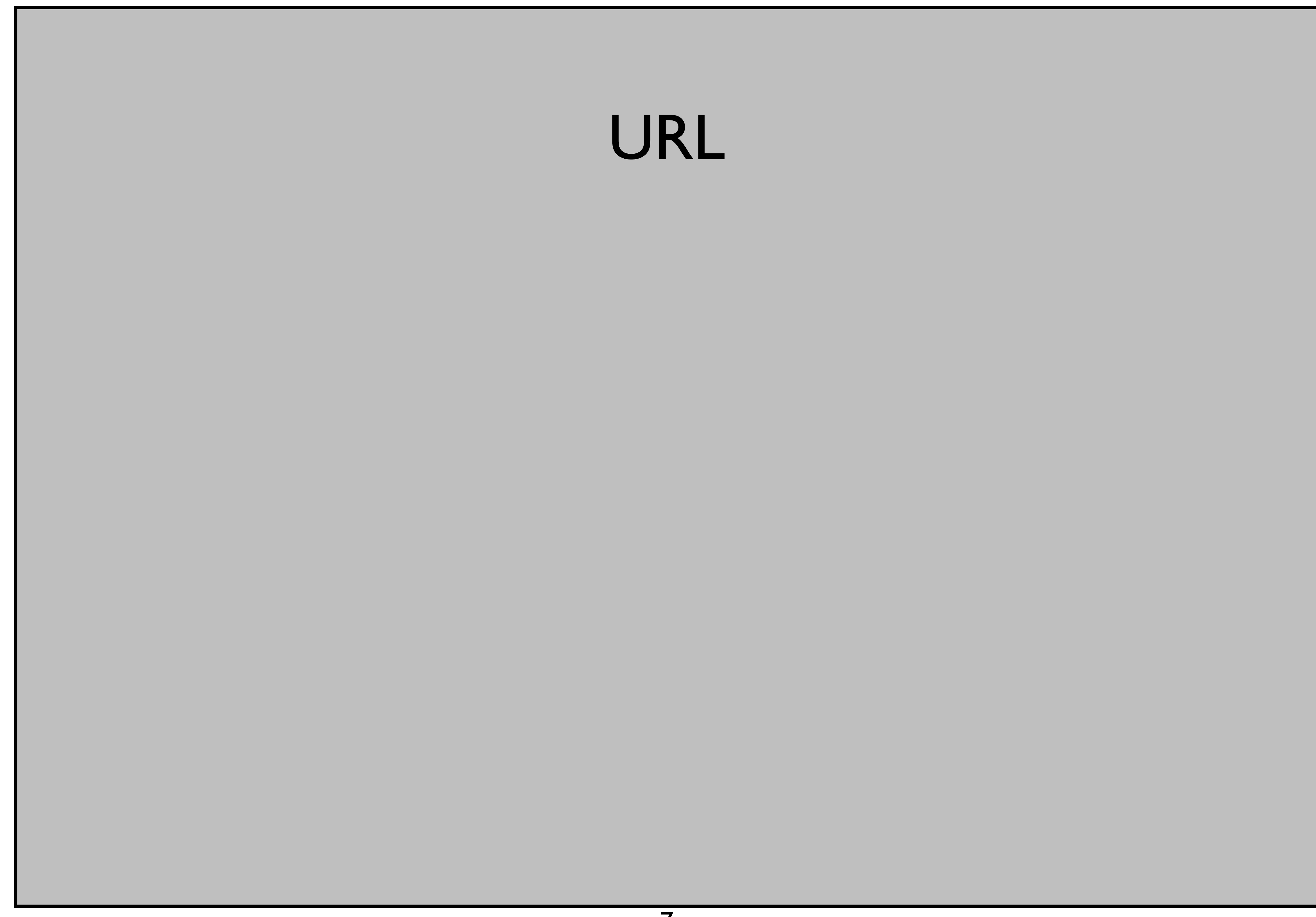
A Brief History of Cache



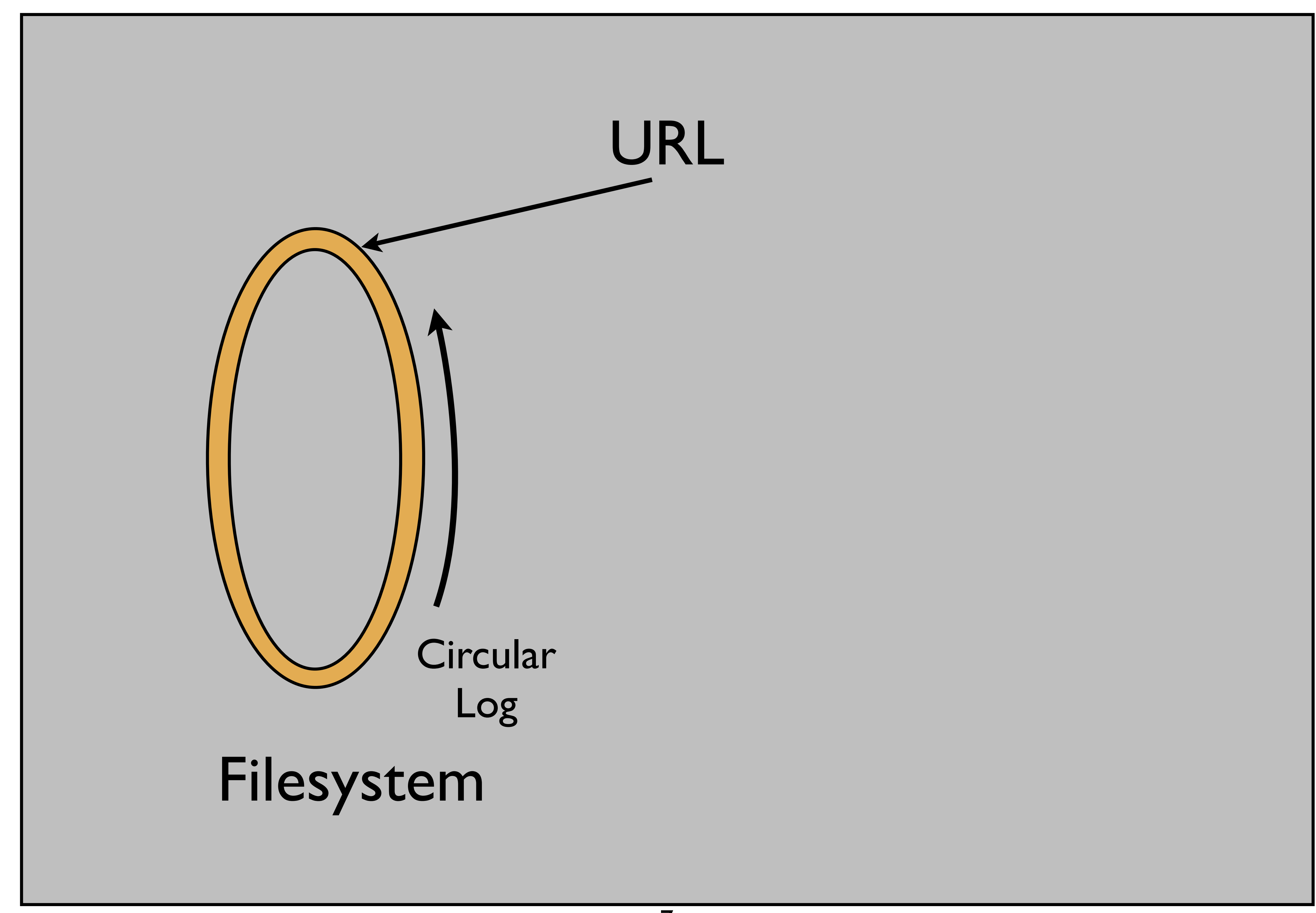
A Brief History of Cache



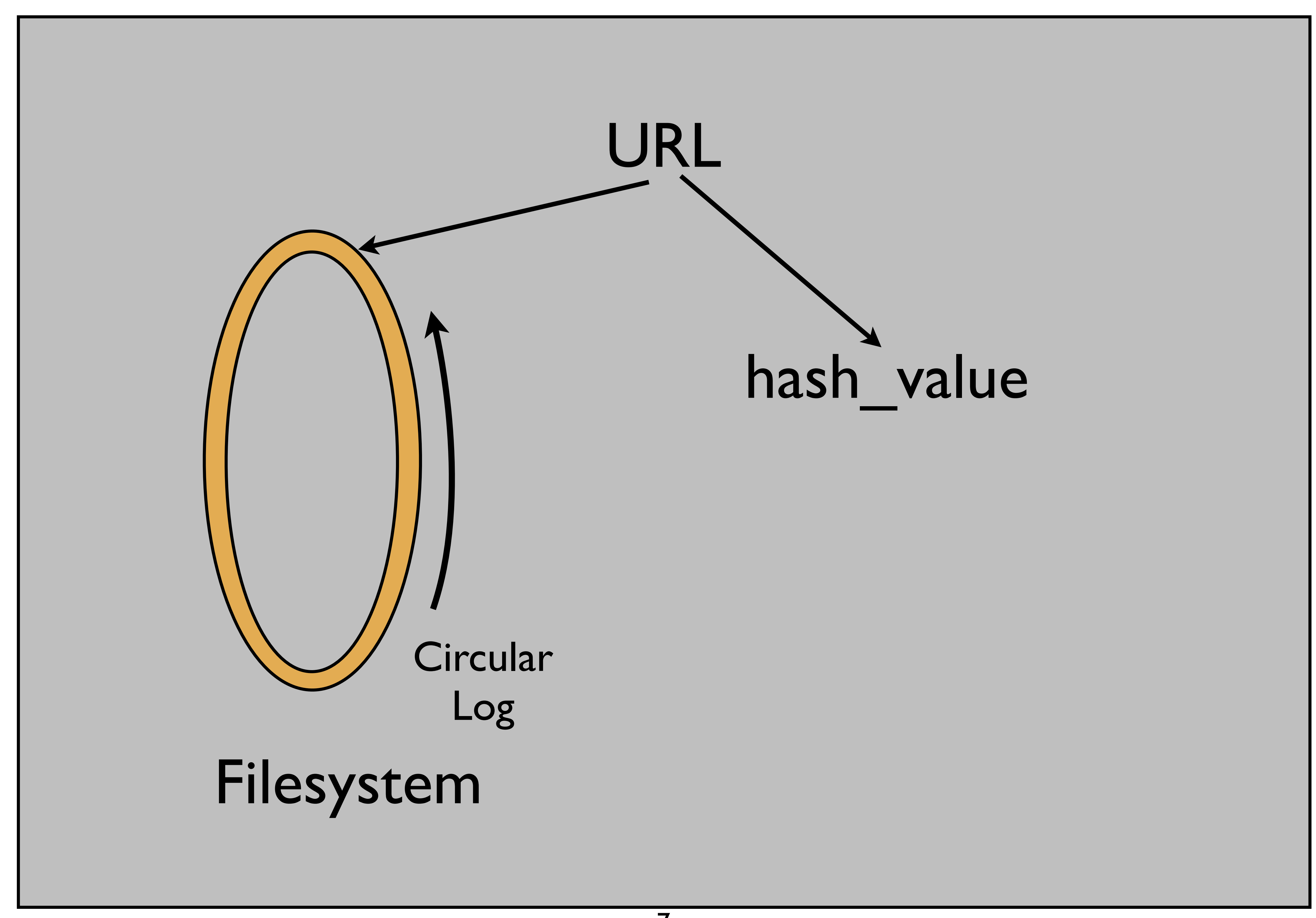
A Brief History of Cache



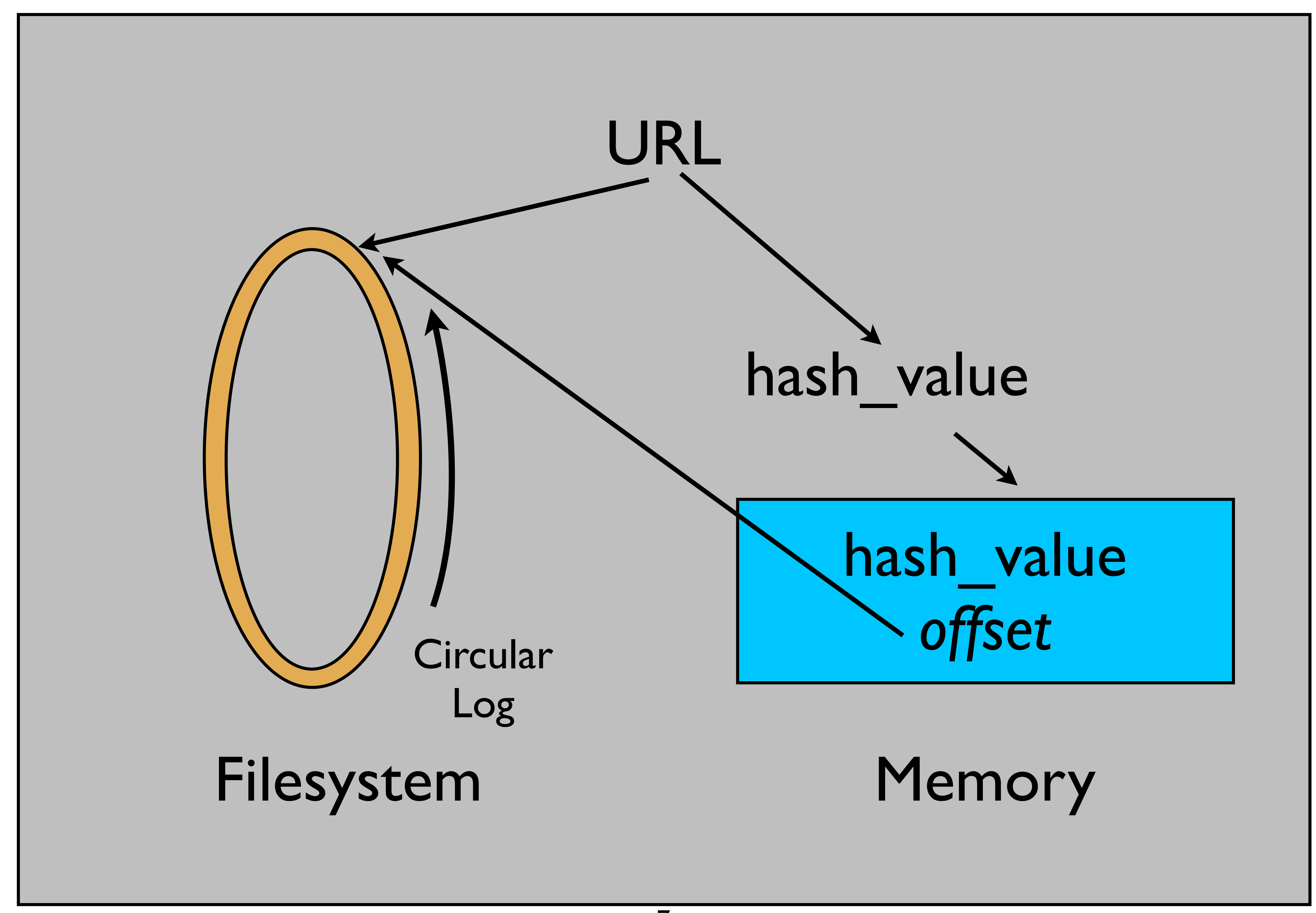
A Brief History of Cache



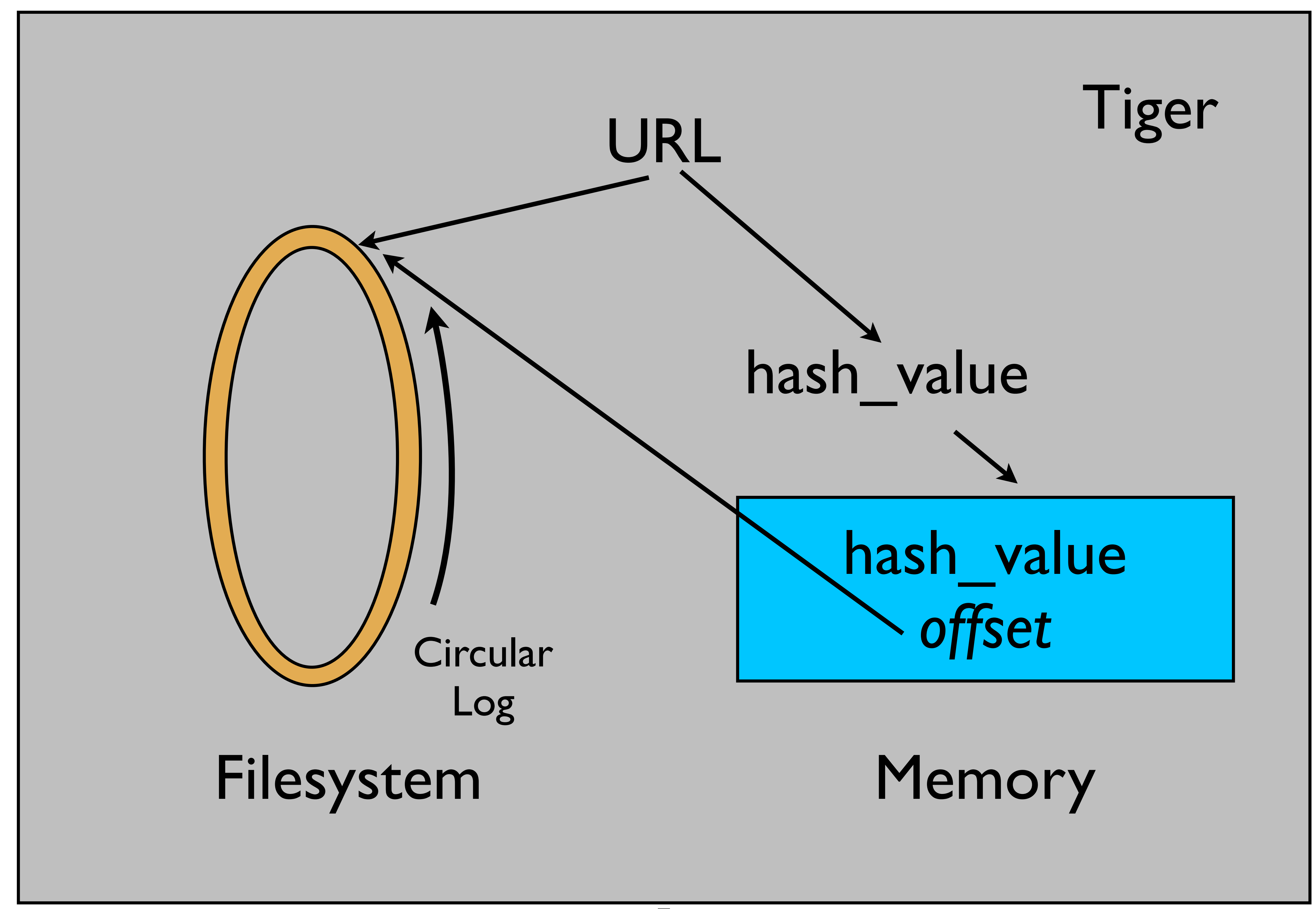
A Brief History of Cache



A Brief History of Cache



A Brief History of Cache



A Brief History of Cache

- Open Source Implementation - Squid
  - Multiple seeks for hit, miss and write
  - Dependent on default filesystems
- Commercial/High Performance - Tiger
  - One seek for hit
  - Custom file layout

Index Element Sizes

Index Element Sizes

Functionality
Existence Identification
Replacement Policy
Location Information
Other

Index Element Sizes

Functionality	Implementation Choice
Existence Identification	Hashtable Chaining Pointers
	Hash
Replacement Policy	LRU List Pointers
Location Information	Disk Offset, Version Number, etc
Other	Expiration Date, Size, HTTP header info etc
	Total

Index Element Sizes

Functionality	Implementation Choice
Existence Identification	Hashtable Chaining Pointers
	Hash
Replacement Policy	LRU List Pointers
Location Information	Disk Offset, Version Number, etc
Other	Expiration Date, Size, HTTP header info etc
	Total

Focusing mainly on reducing the size of the index

Index Element Sizes

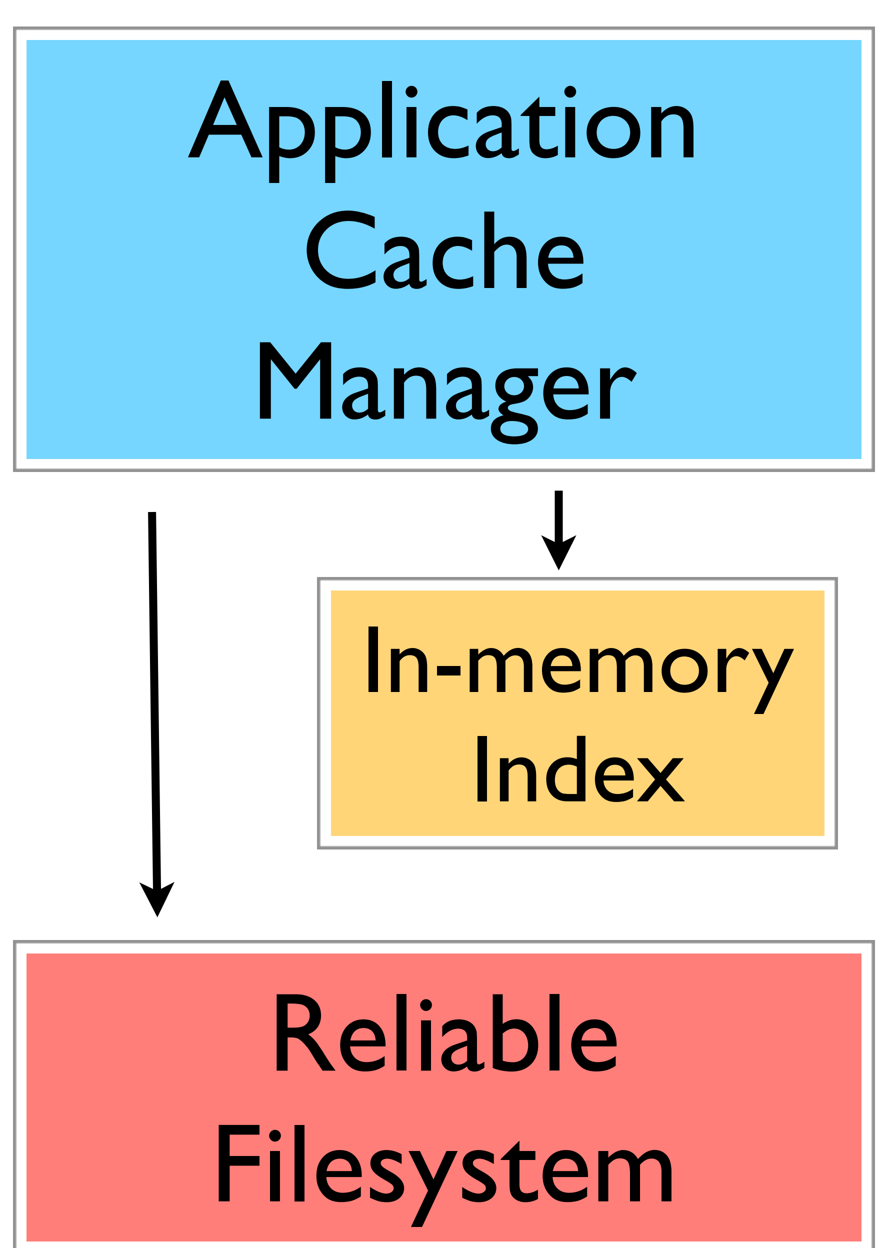
Functionality	Implementation Choice	Squid (Bits)
Existence Identification	Hashtable Chaining Pointers	96
	Hash	160
Replacement Policy	LRU List Pointers	64
Location Information	Disk Offset, Version Number, etc	0
Other	Expiration Date, Size, HTTP header info etc	240
	Total	560

Index Element Sizes

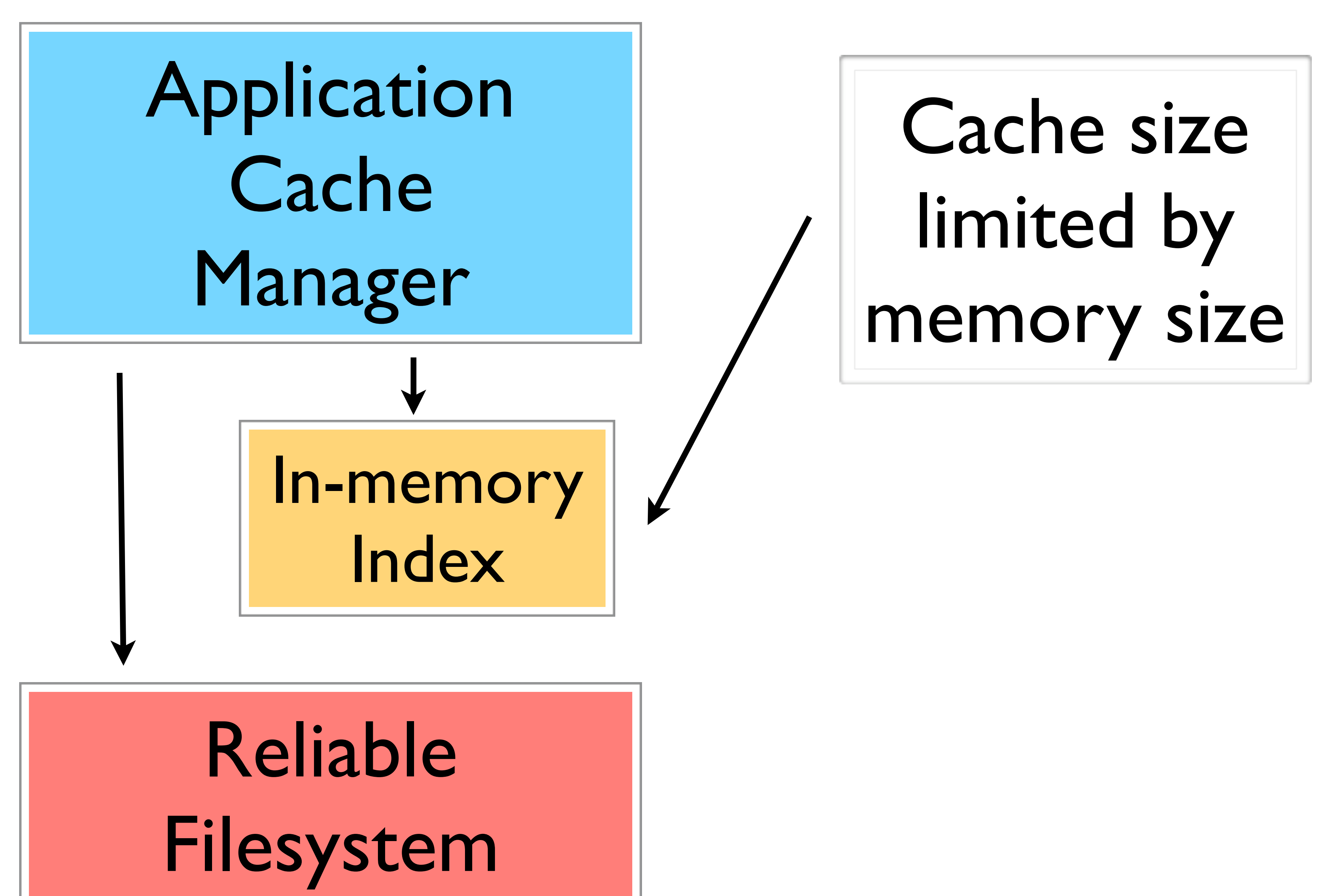
Functionality	Implementation Choice	Squid (Bits)	Tiger (Bits)
Existence Identification	Hashtable Chaining Pointers	96	96
	Hash	160	32
Replacement Policy	LRU List Pointers	64	64
Location Information	Disk Offset, Version Number, etc	0	40
Other	Expiration Date, Size, HTTP header info etc	240	0
	Total	560	232

Revisiting the Index...

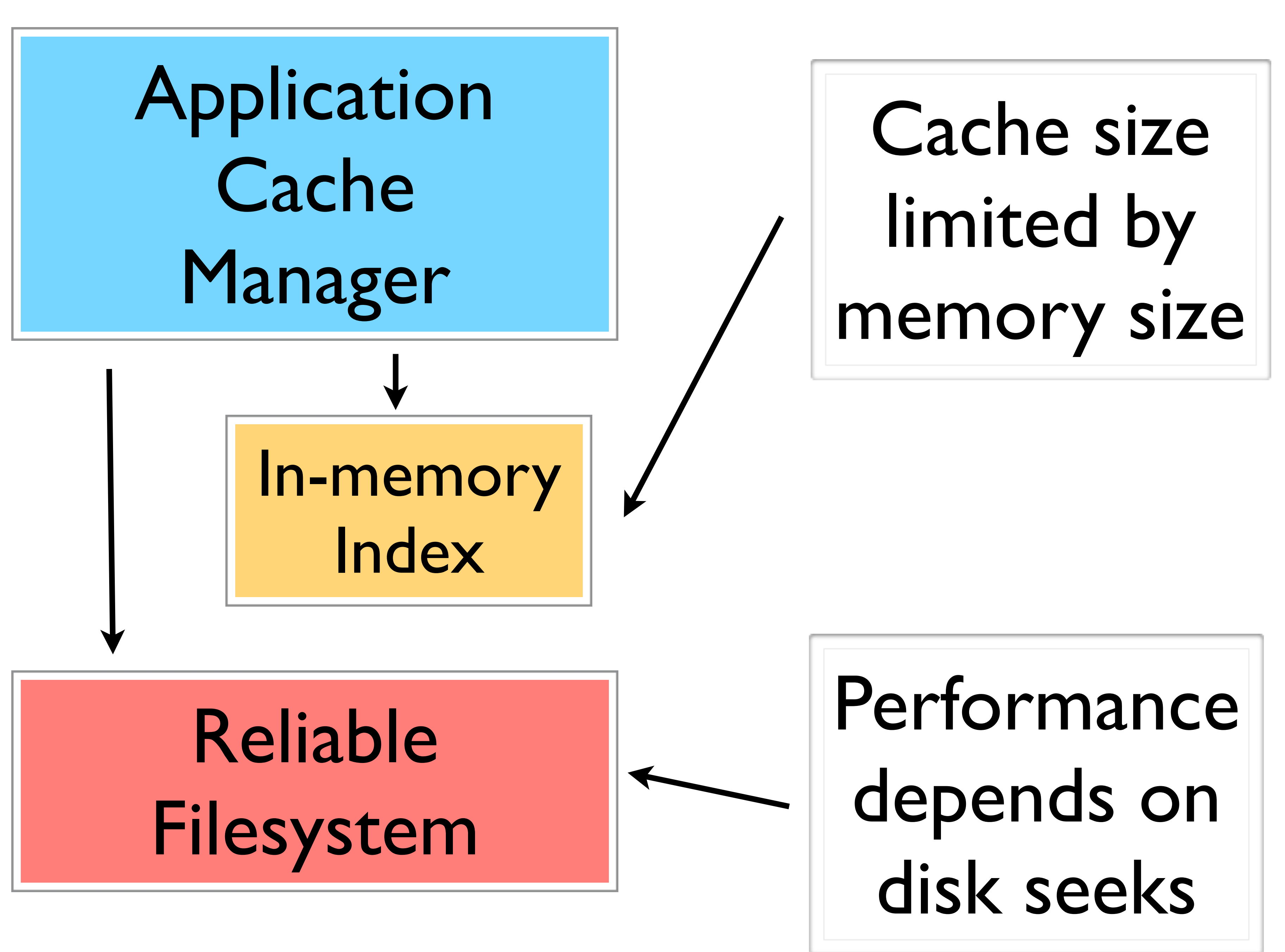
Revisiting the Index...



Revisiting the Index...



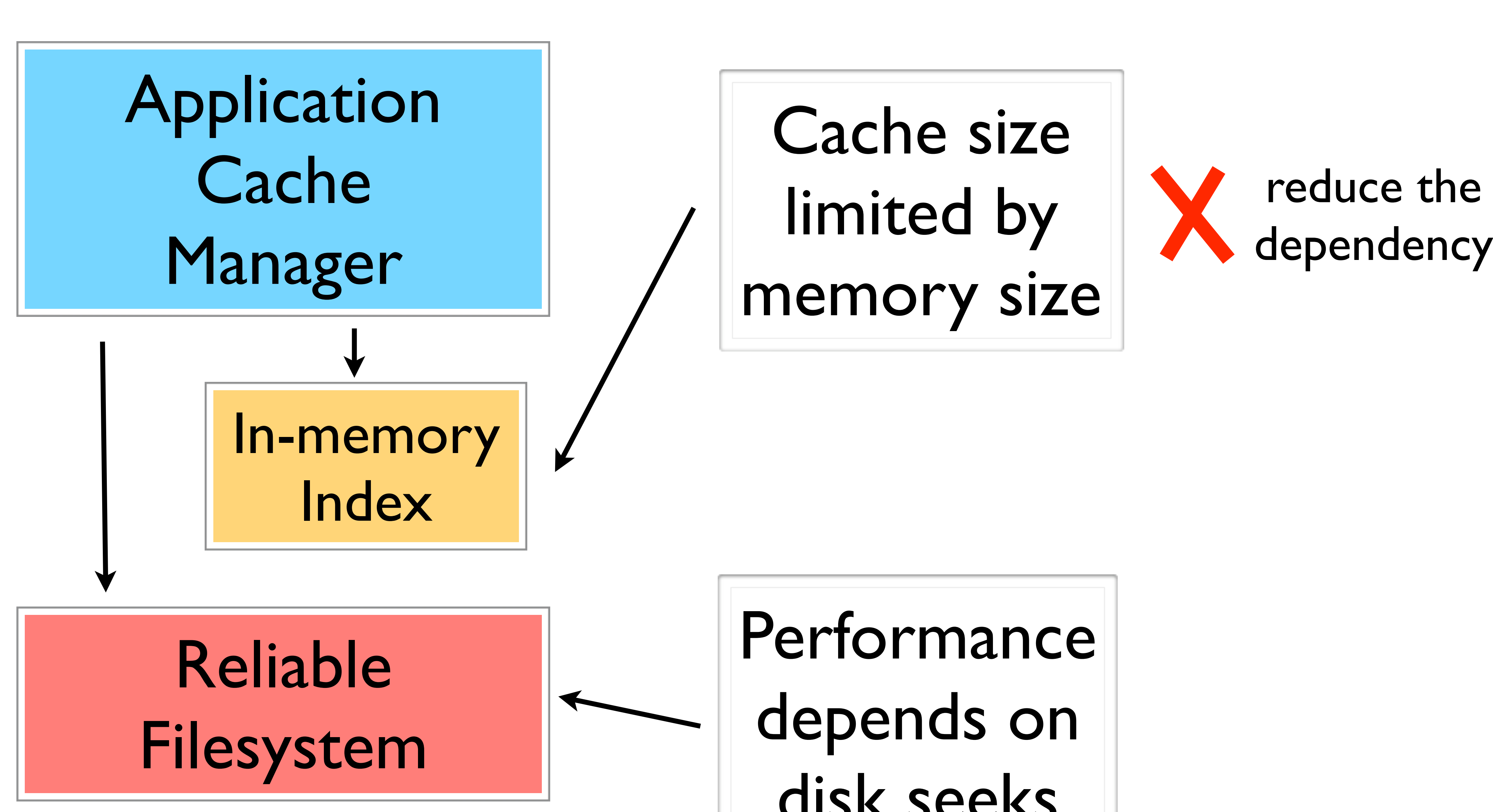
Revisiting the Index...



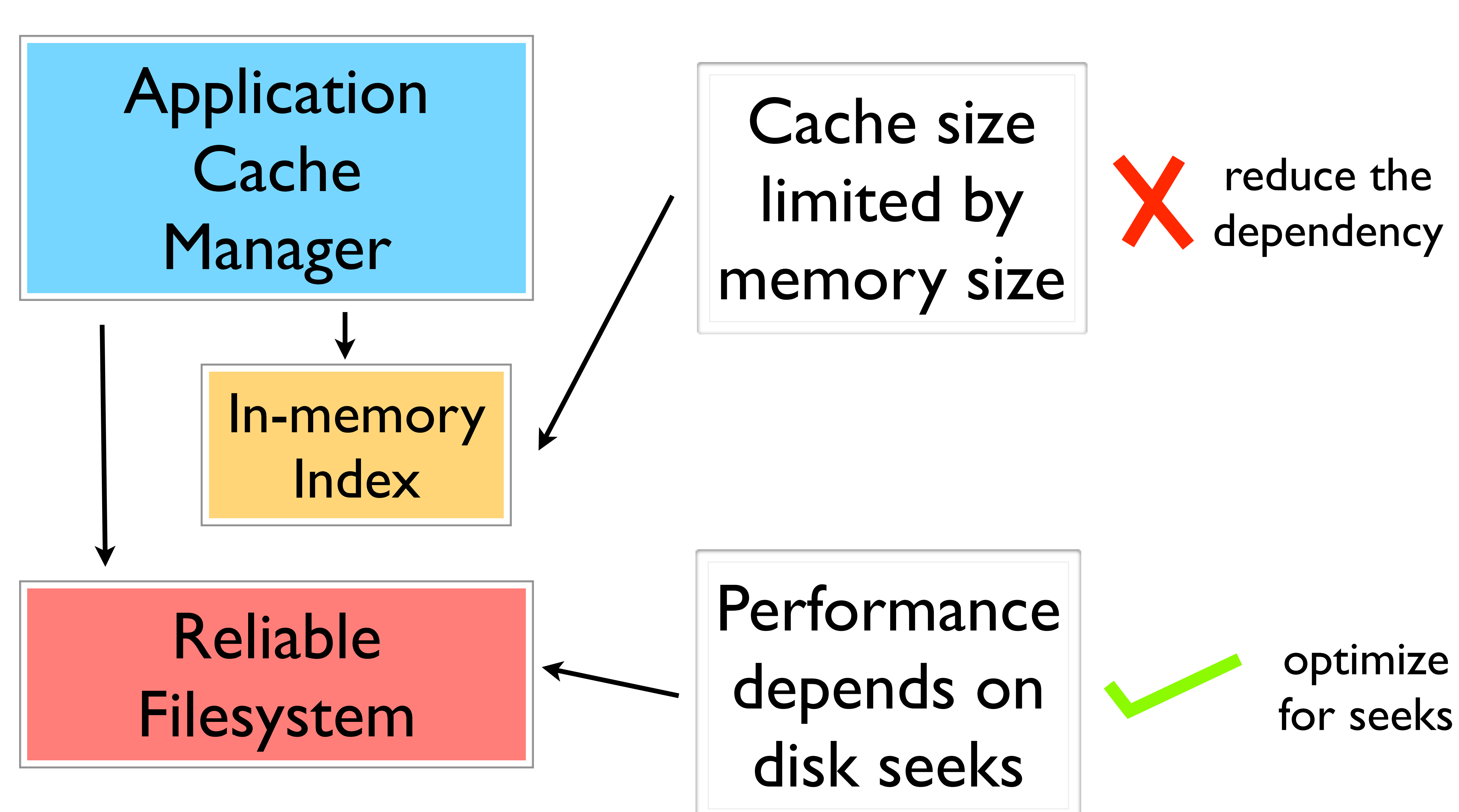
Cache size limited by memory size

Performance depends on disk seeks

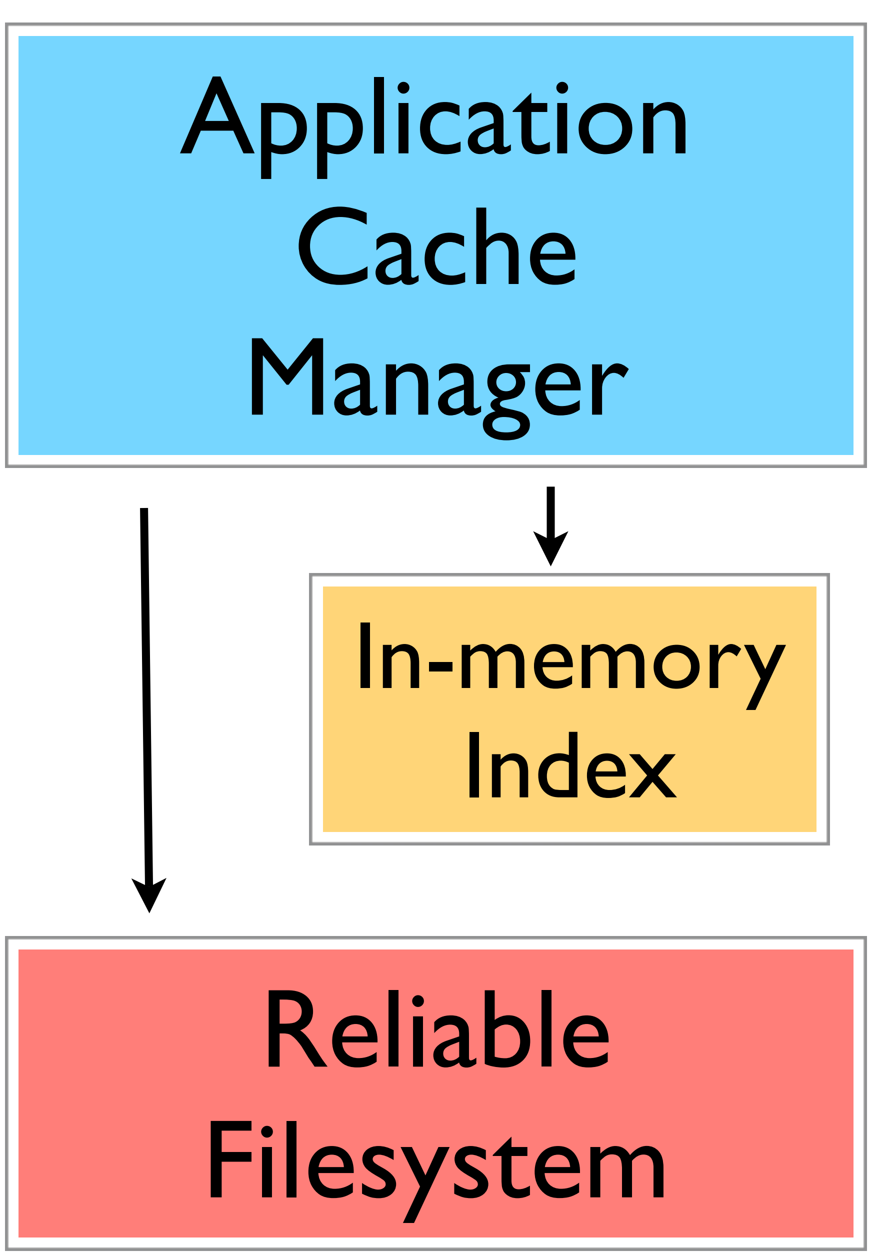
Revisiting the Index...



Revisiting the Index...

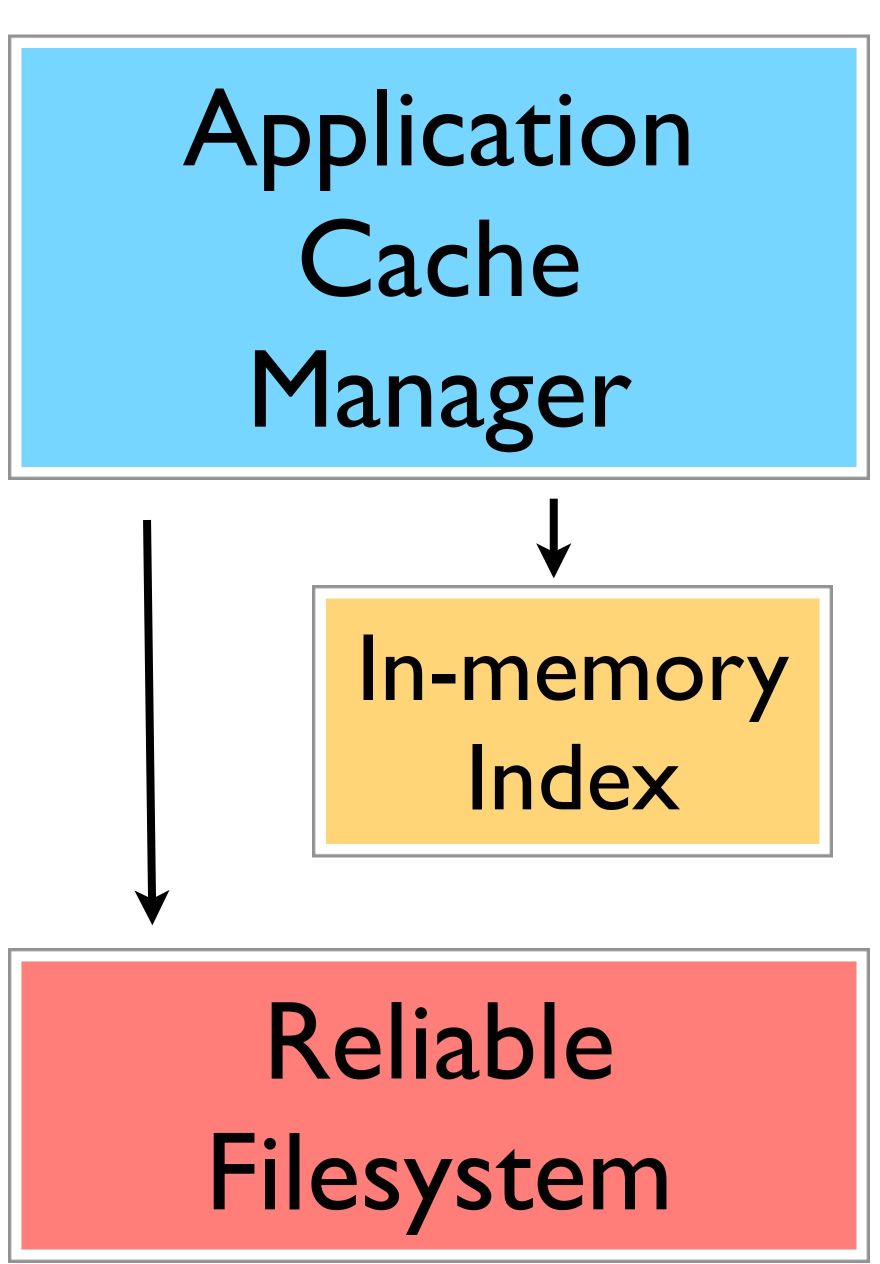


Revisiting the Index...



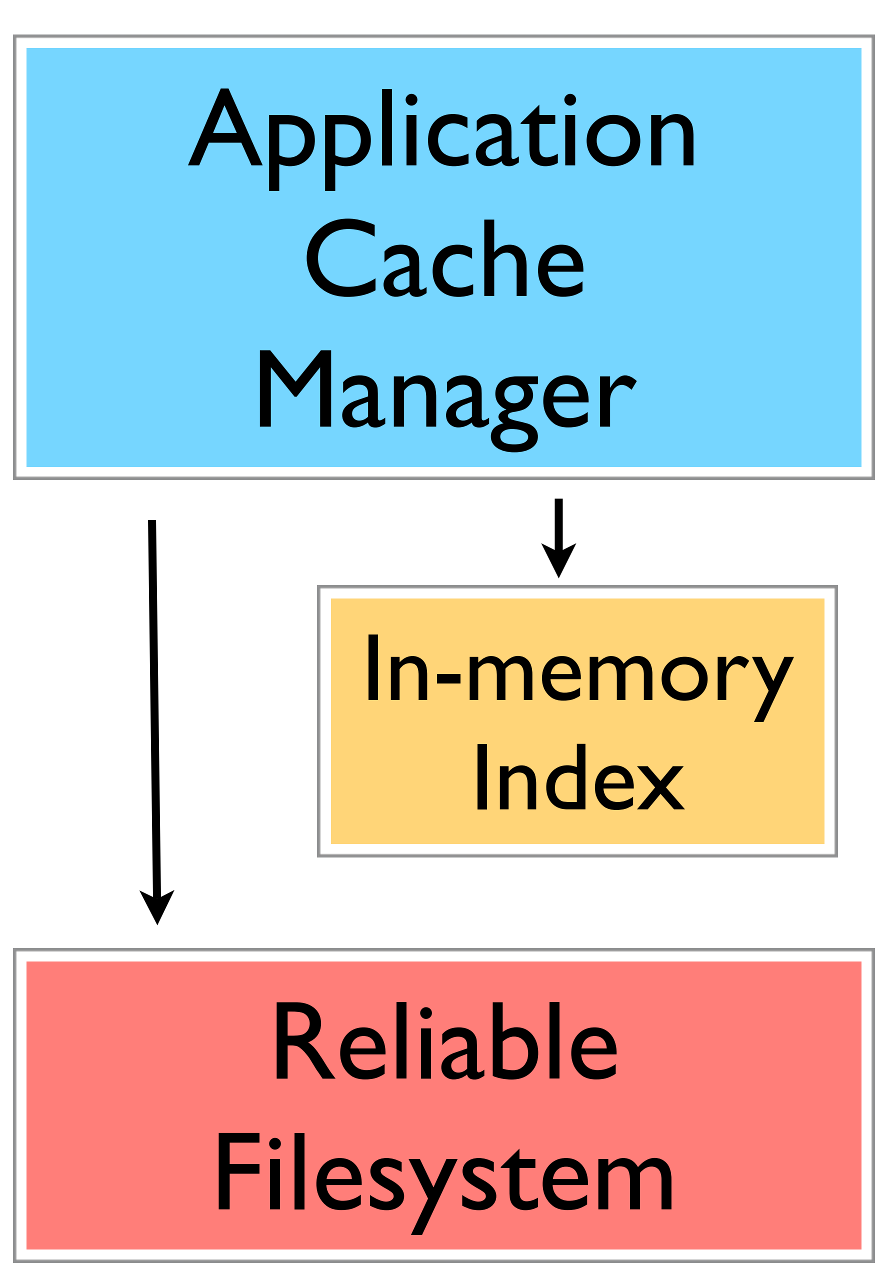
- Eliminate(?) in-memory index

Revisiting the Index...



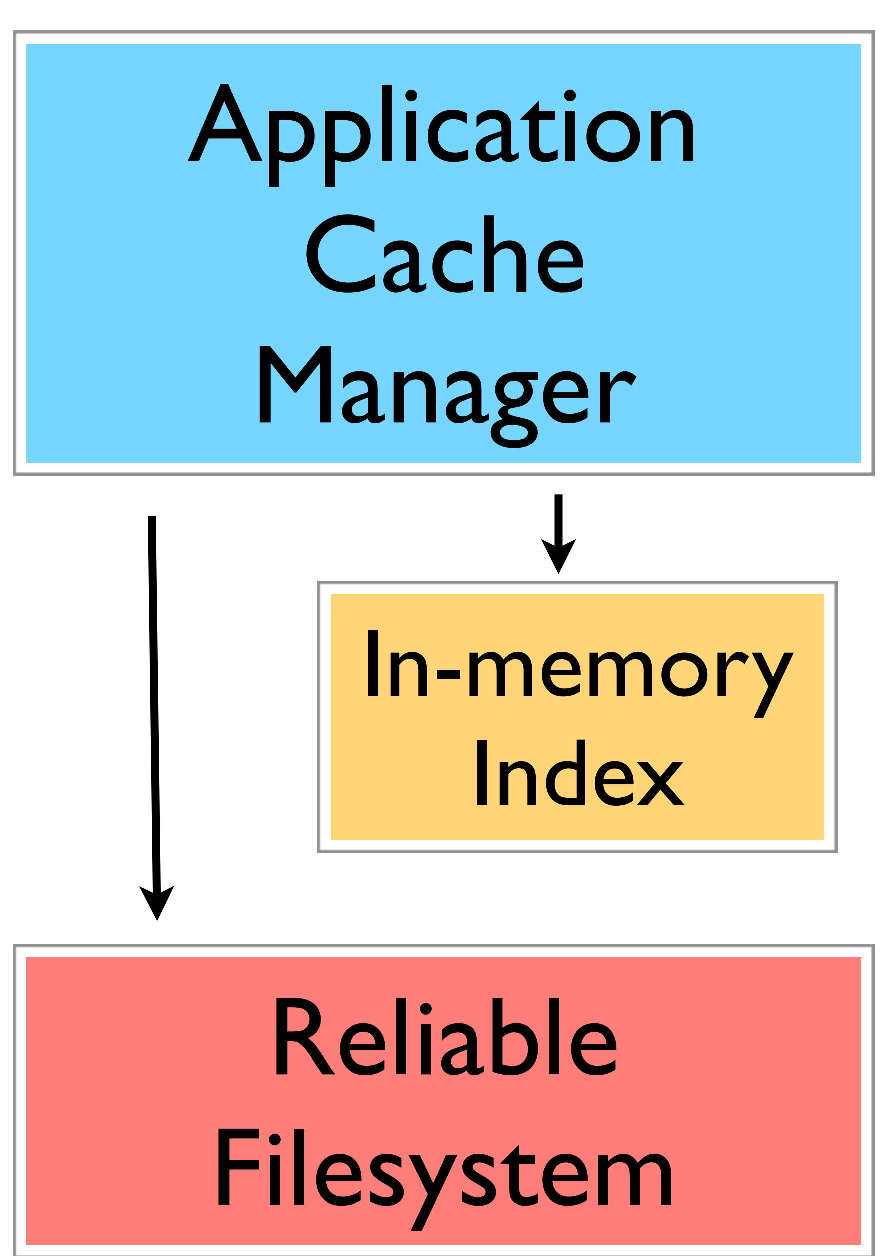
- Eliminate(?) in-memory index
- Need membership and location information

Revisiting the Index...



- Eliminate(?) in-memory index
- Need membership and location information
- Use disk as hash table

Revisiting the Index...



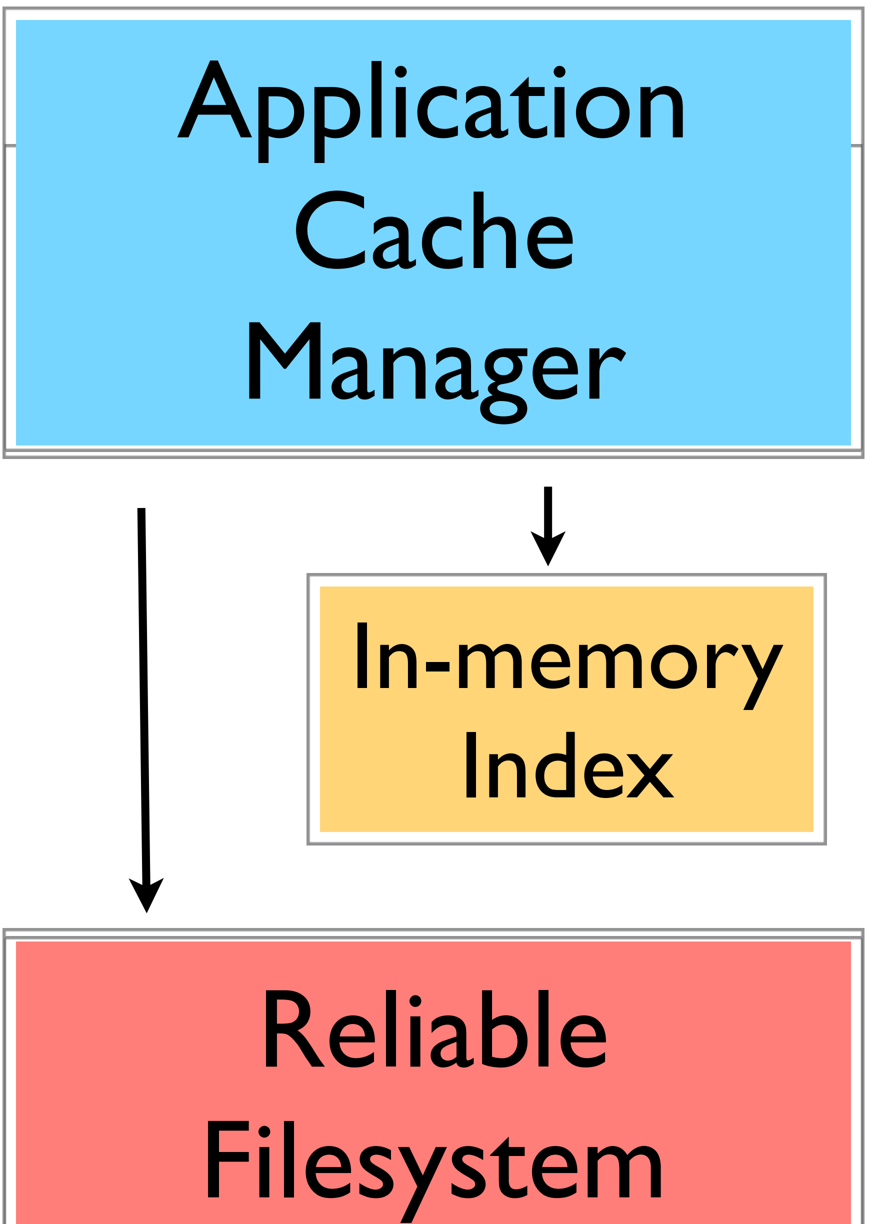
- Eliminate(?) in-memory index
- Need membership and location information
- Use disk as hash table
  - On disk data structures for key lookup
  - Store the object as values for the keys

Revisiting the Index...

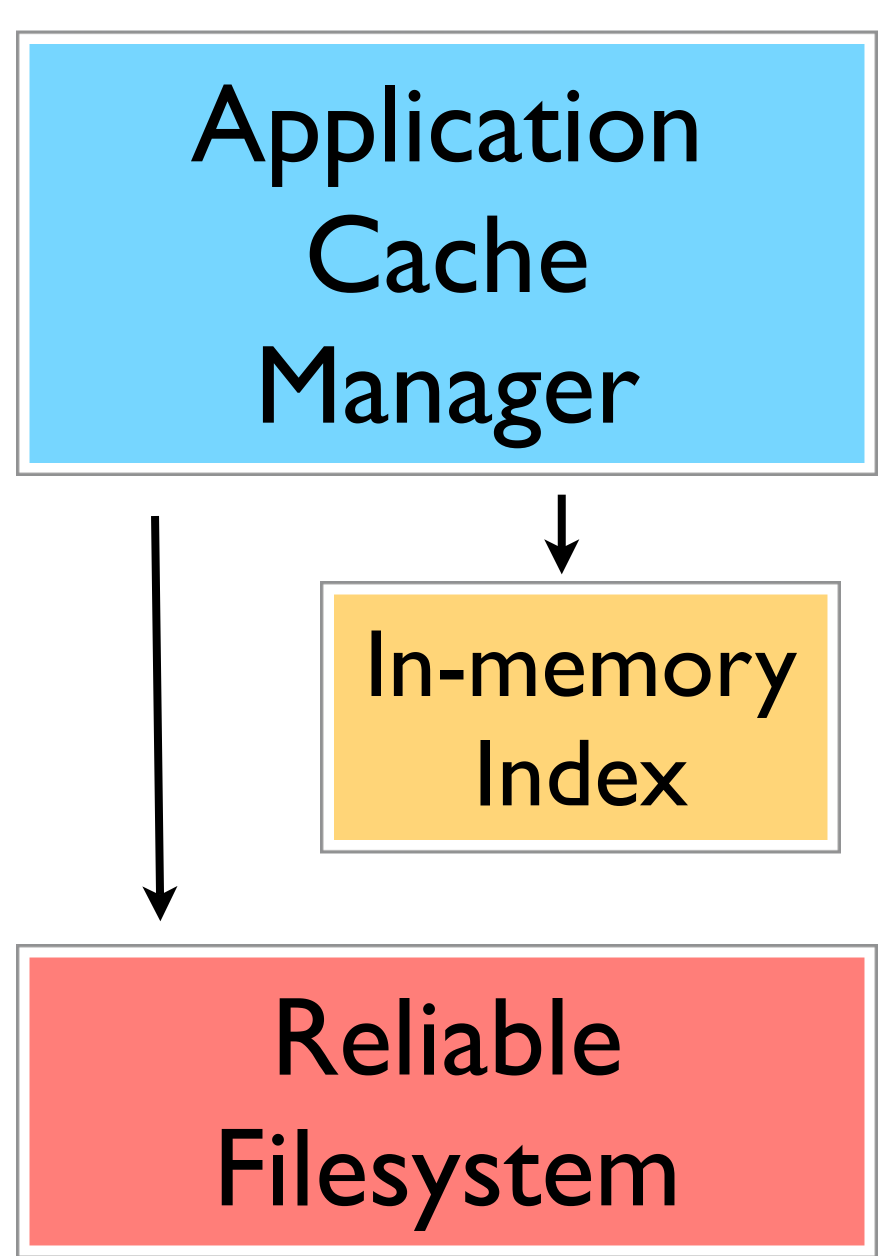
Application Cache Manager

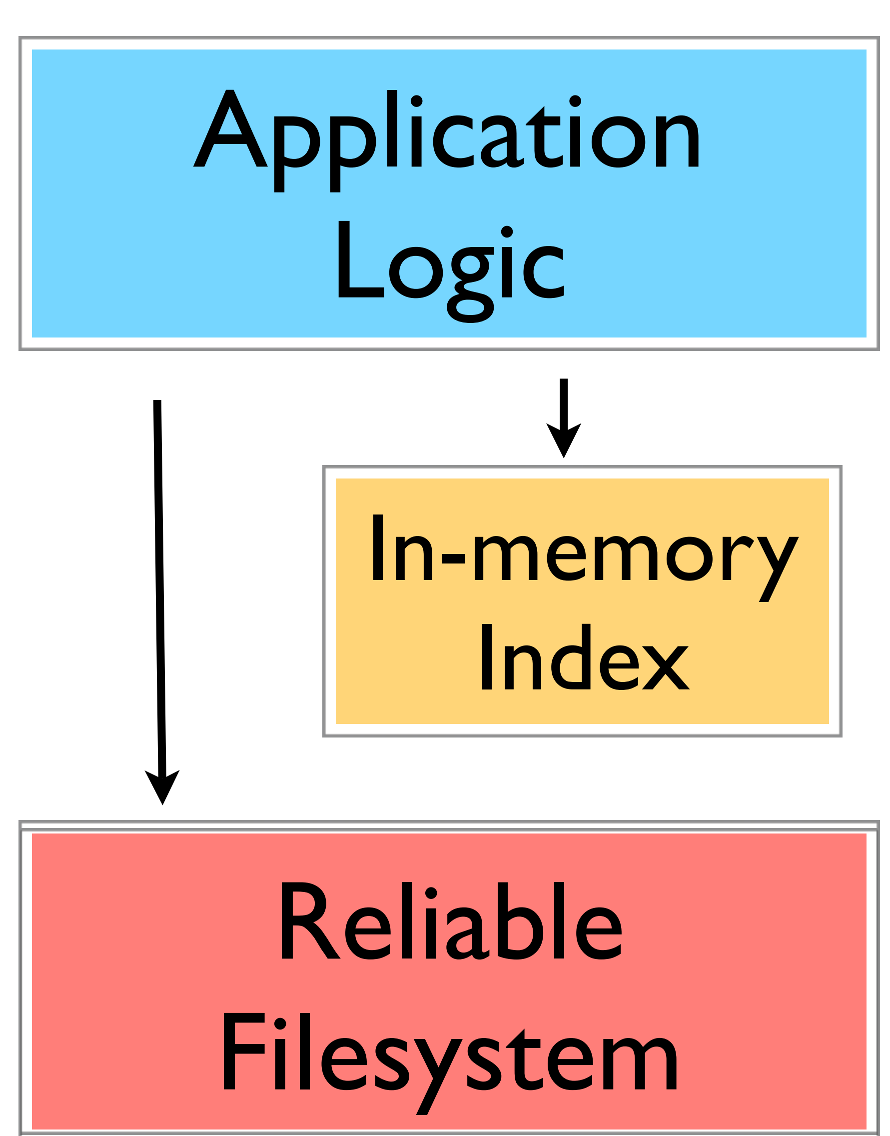
In-memory Index

Reliable Filesystem

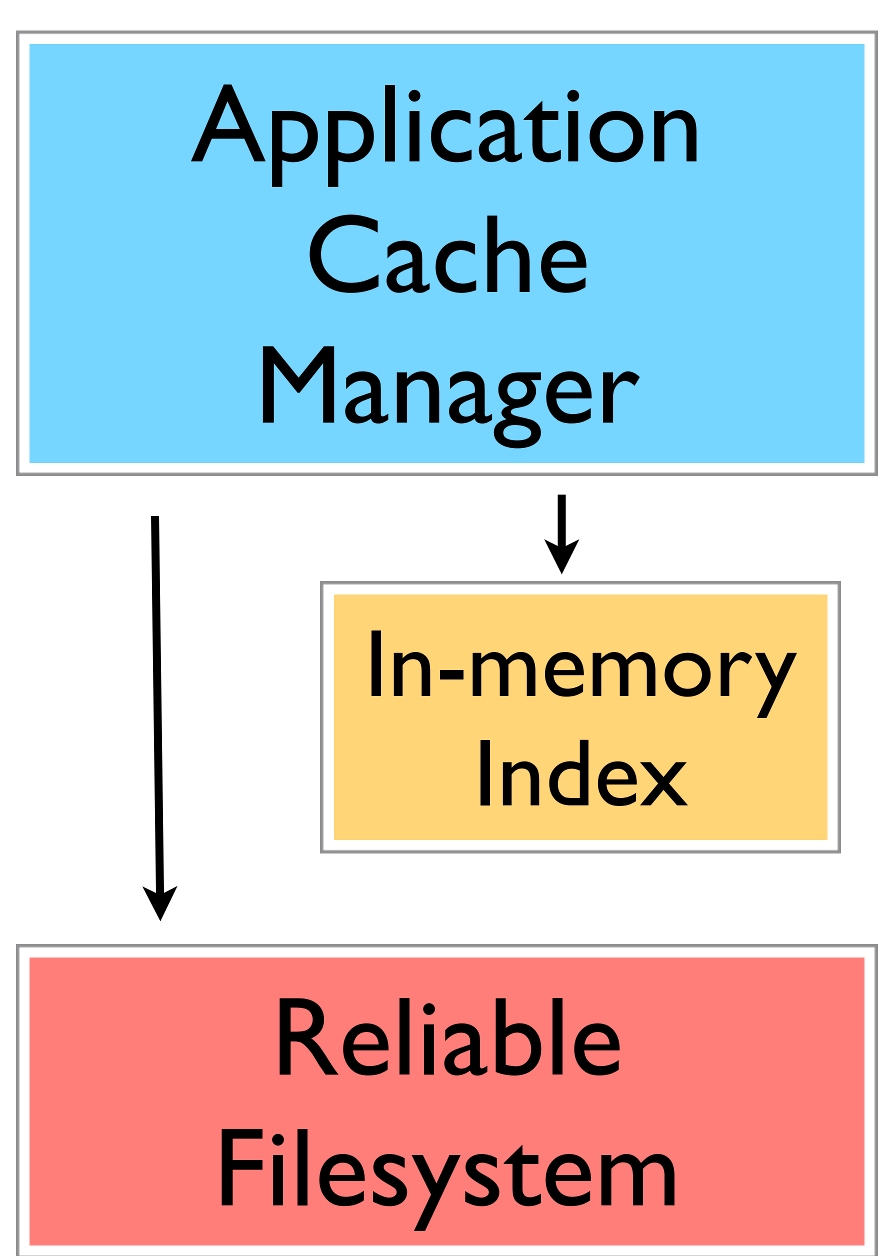


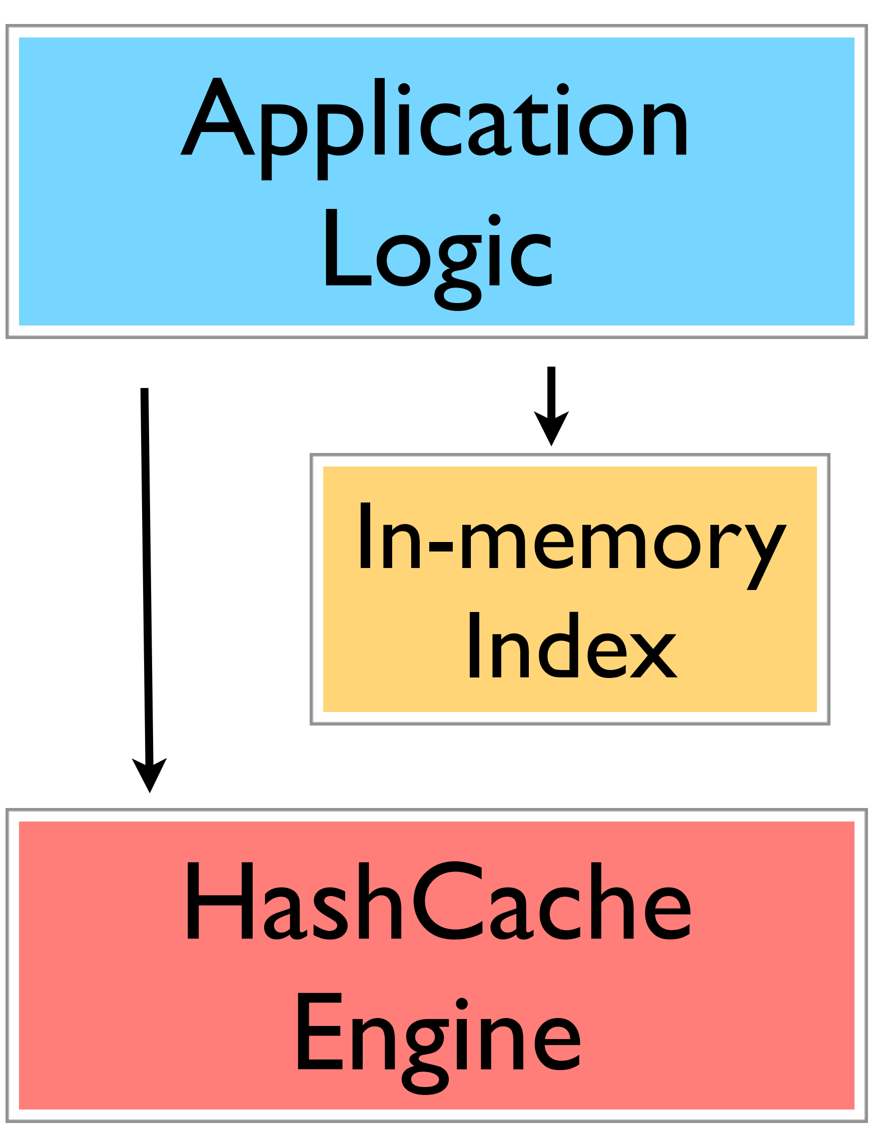
Revisiting the Index...





Revisiting the Index...



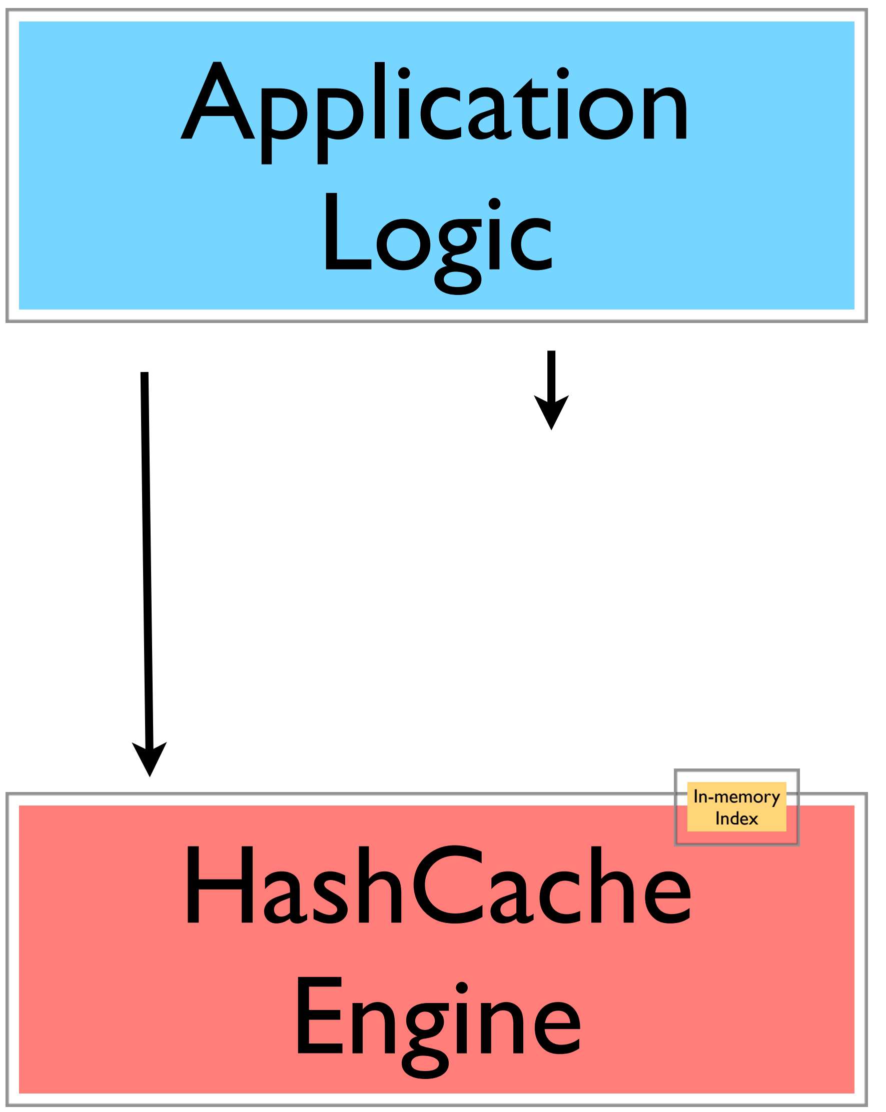


Revisiting the Index...

Application Cache Manager

In-memory Index

Reliable Filesystem

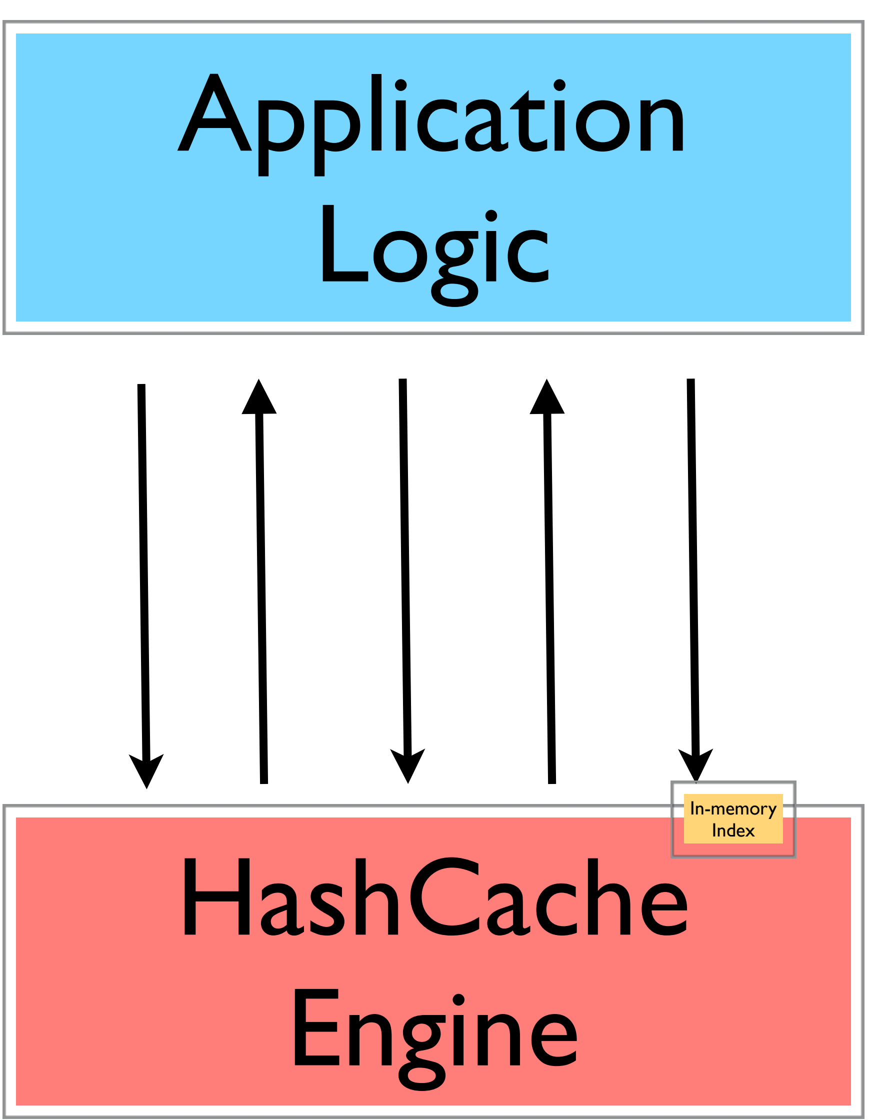


Revisiting the Index...

Application Cache Manager

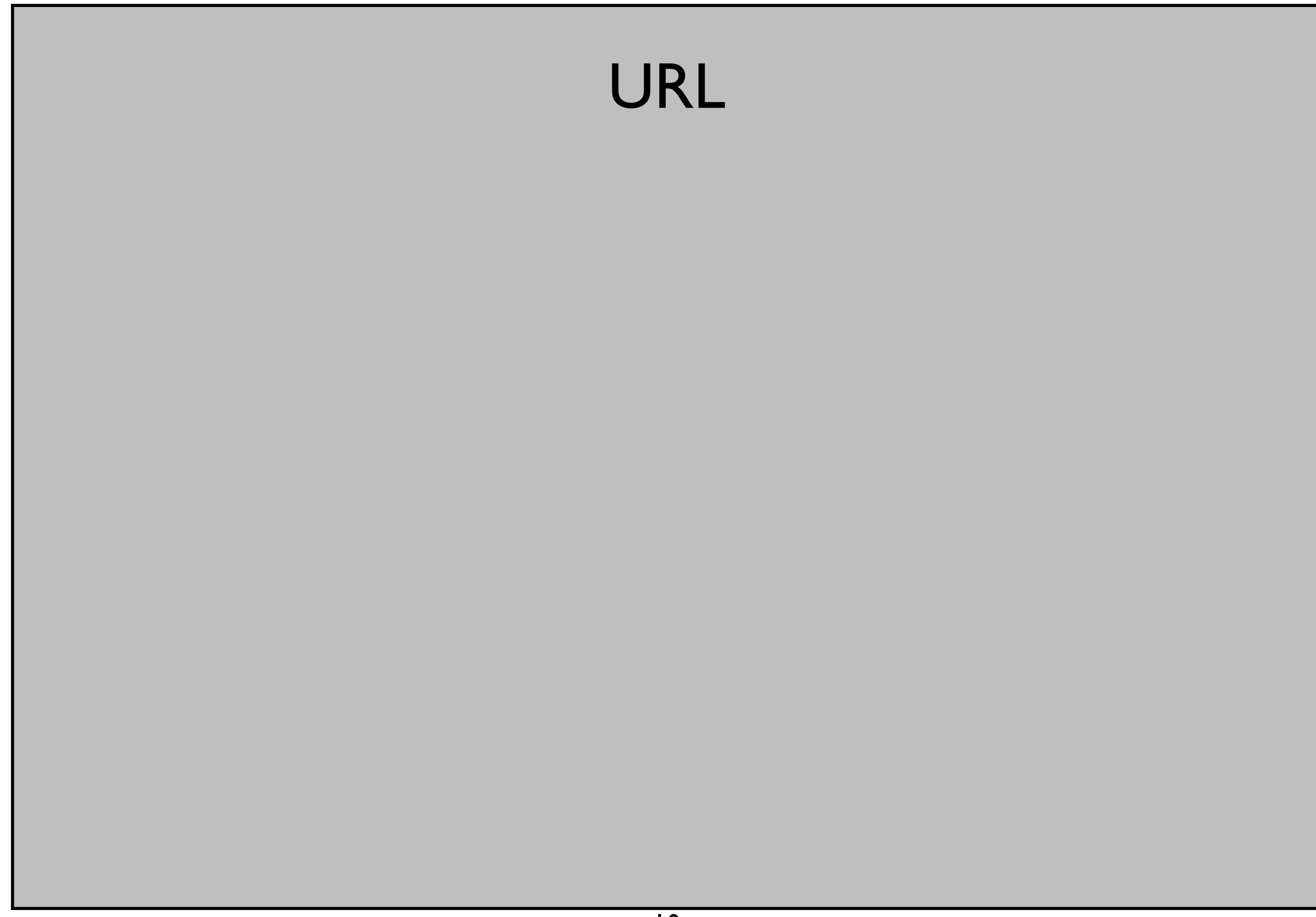
In-memory Index

Reliable Filesystem

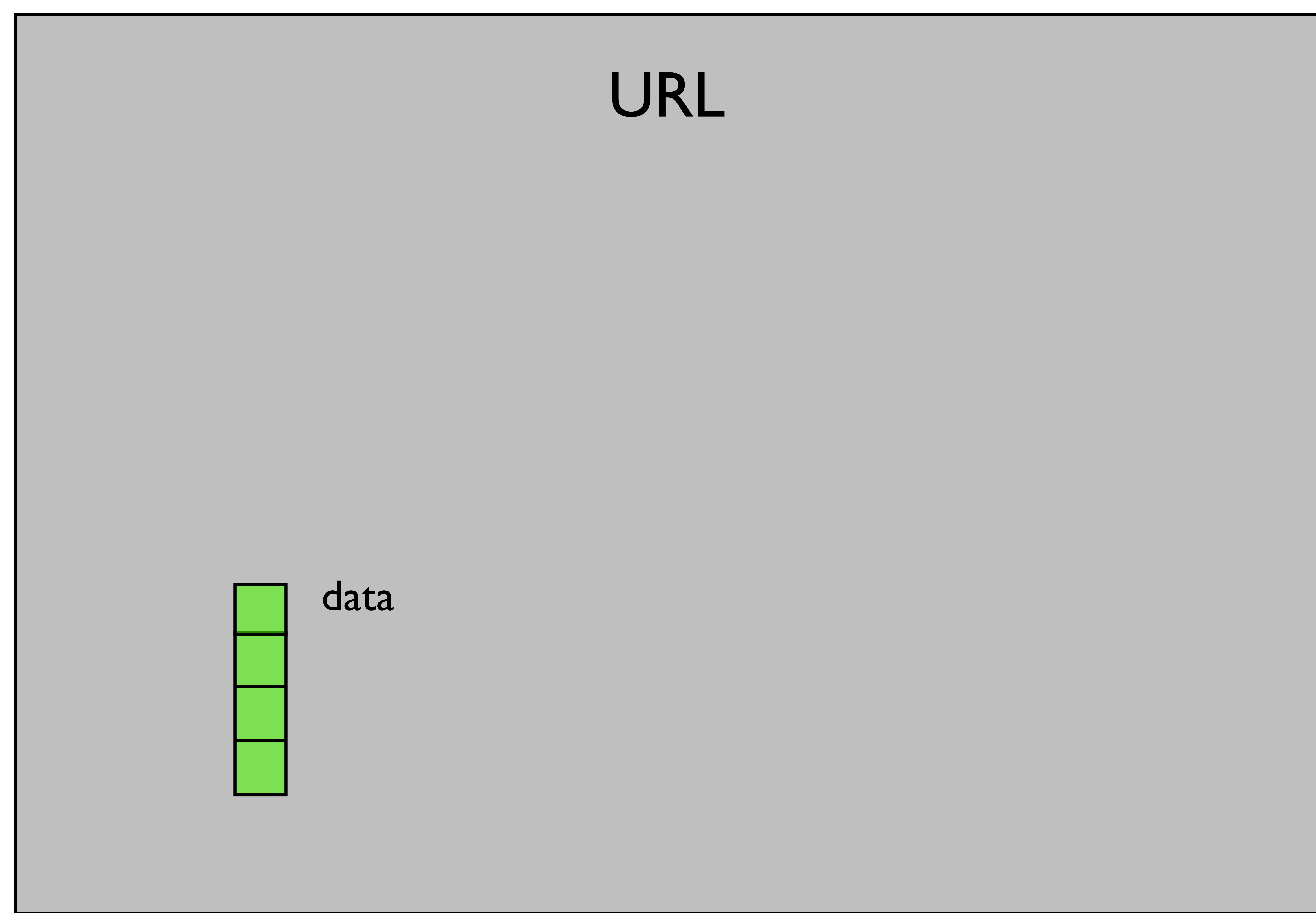


HashCache: Basic Policy

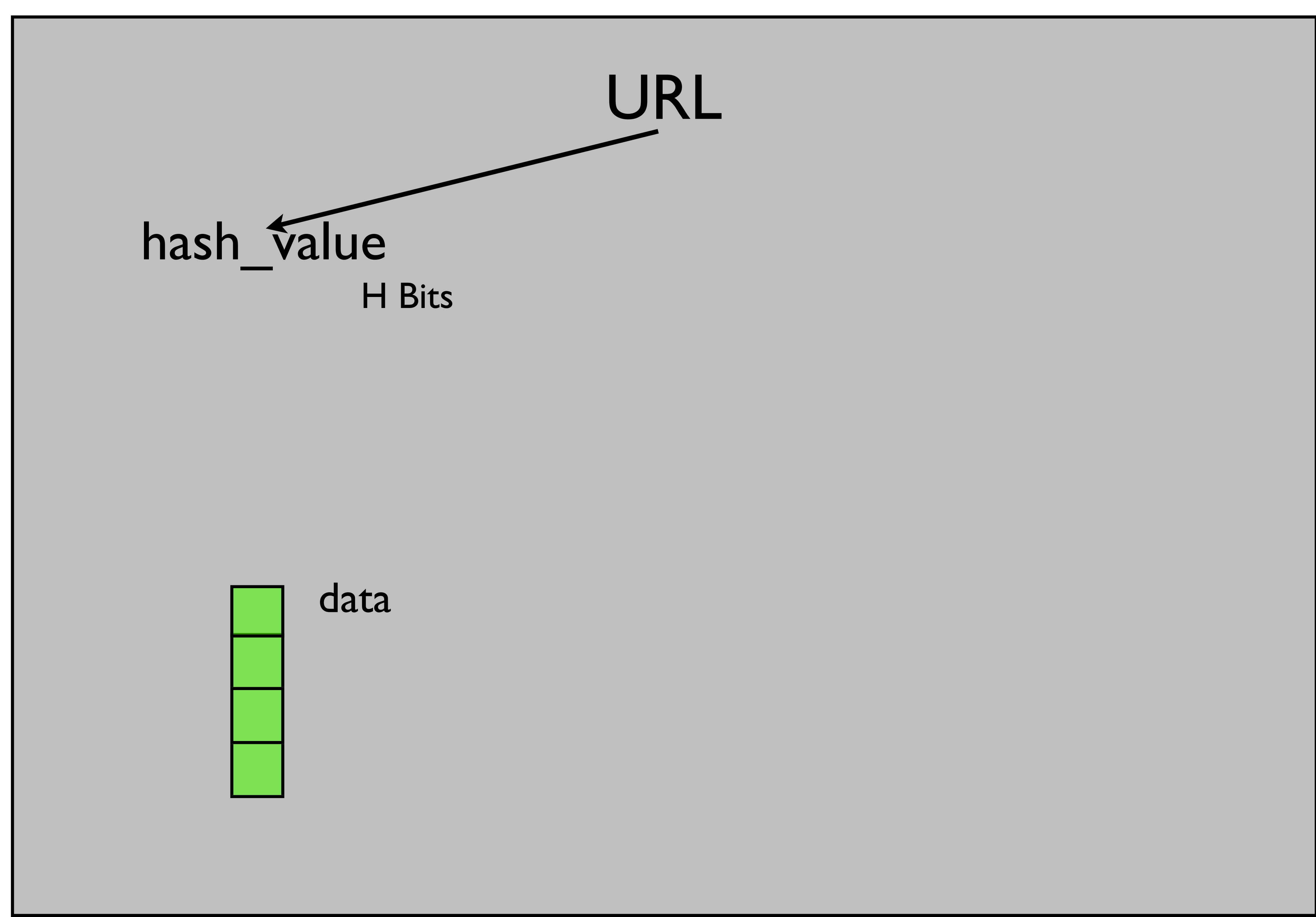
HashCache: Basic Policy



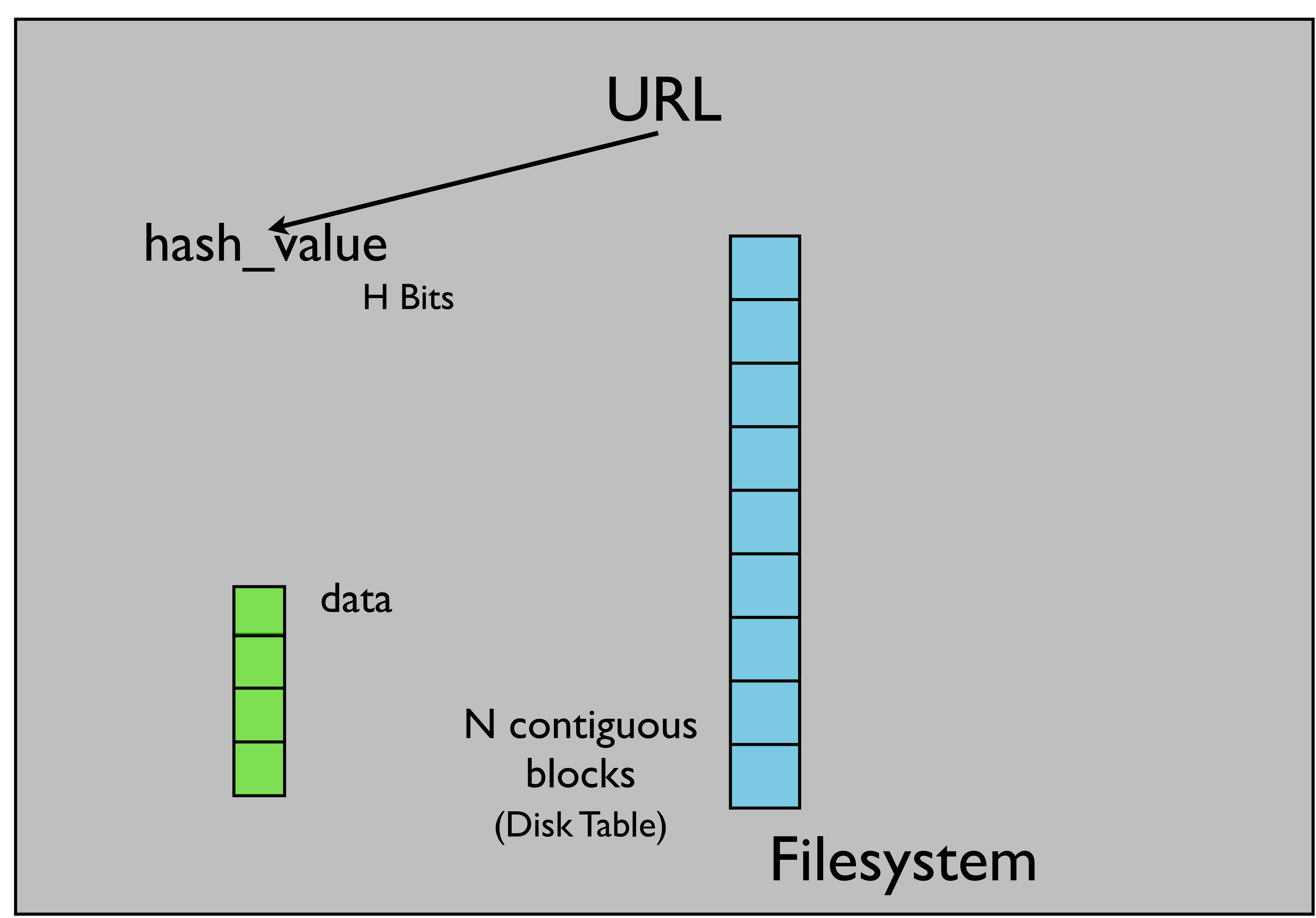
HashCache: Basic Policy



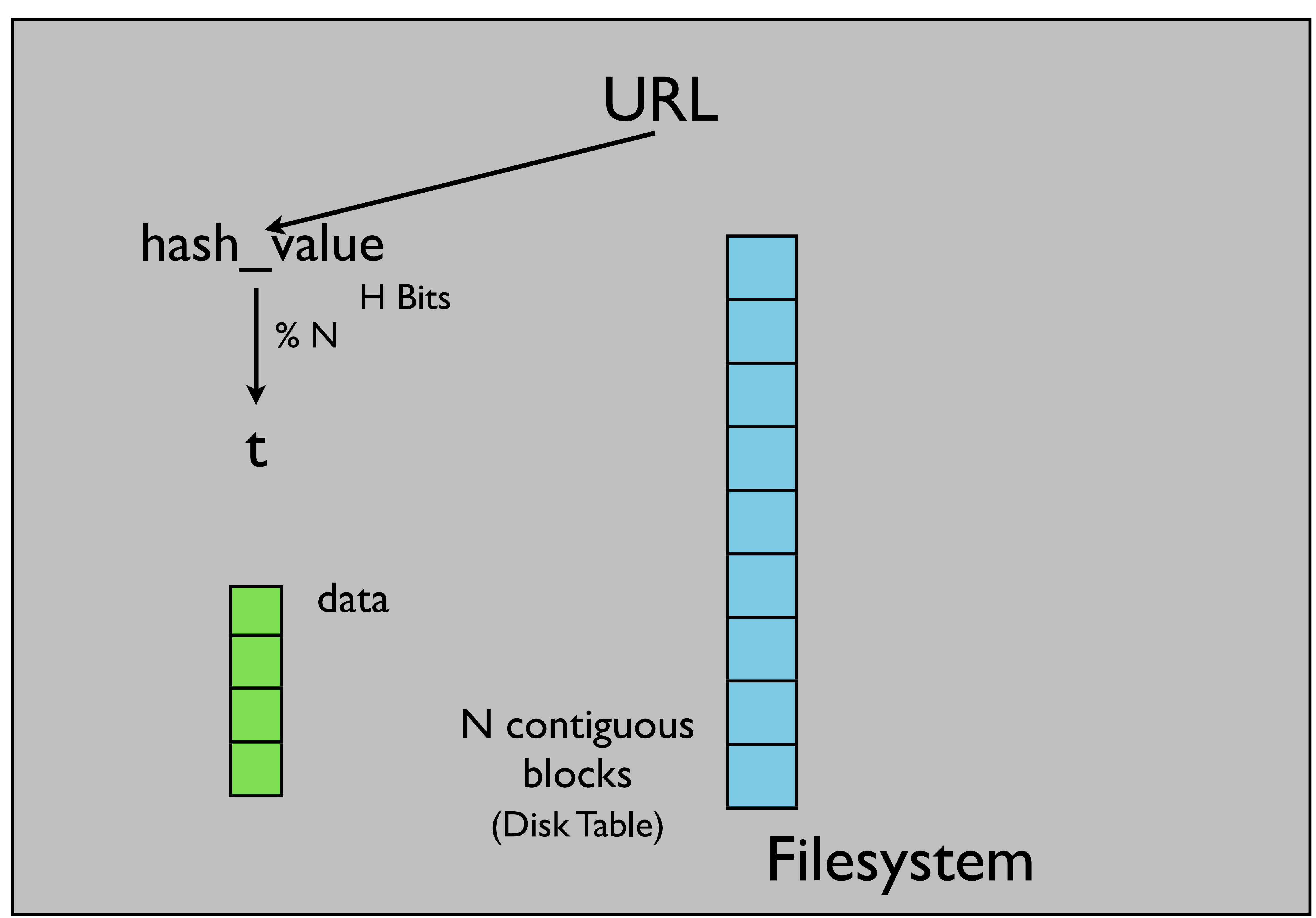
HashCache: Basic Policy



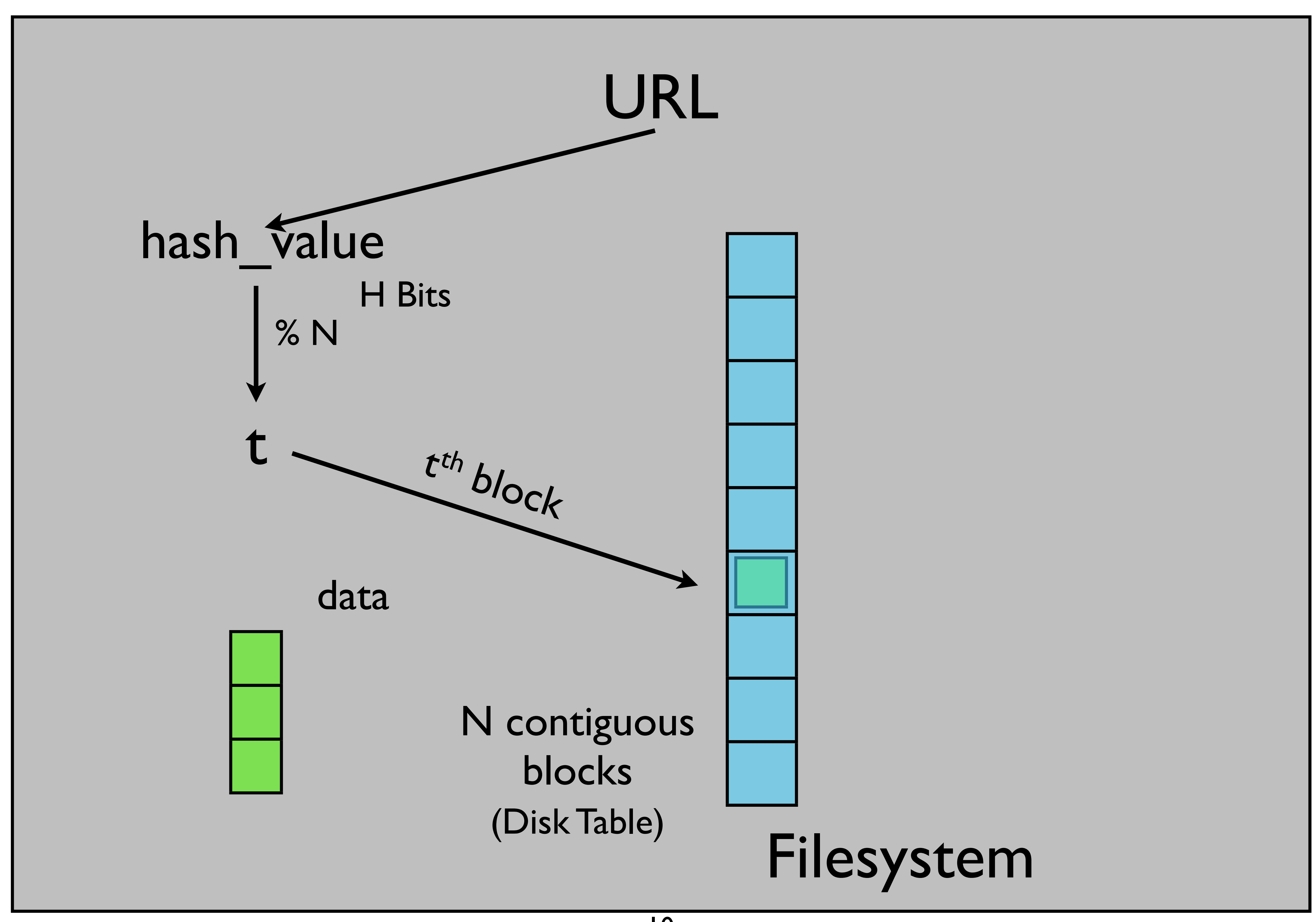
HashCache: Basic Policy



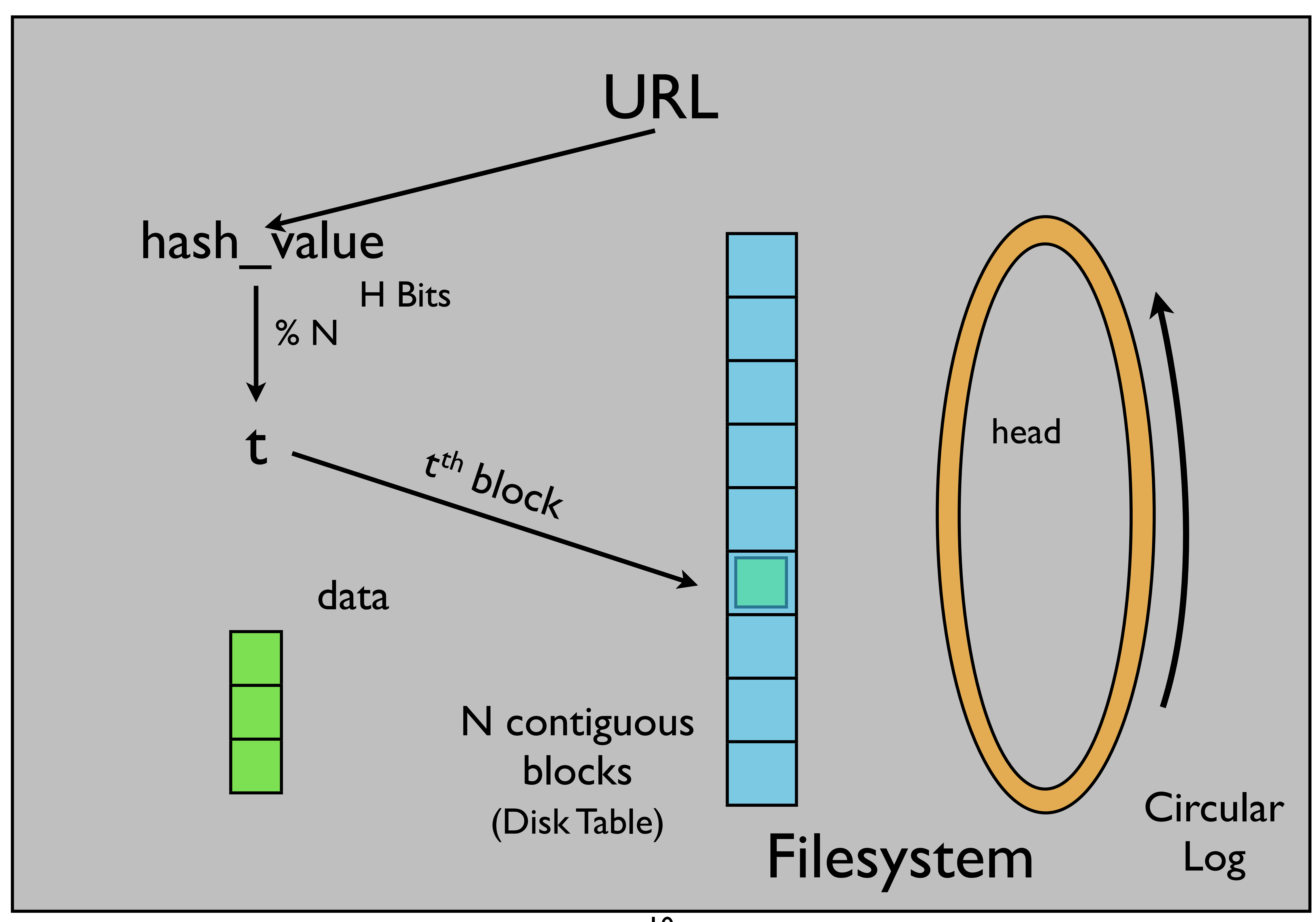
HashCache: Basic Policy



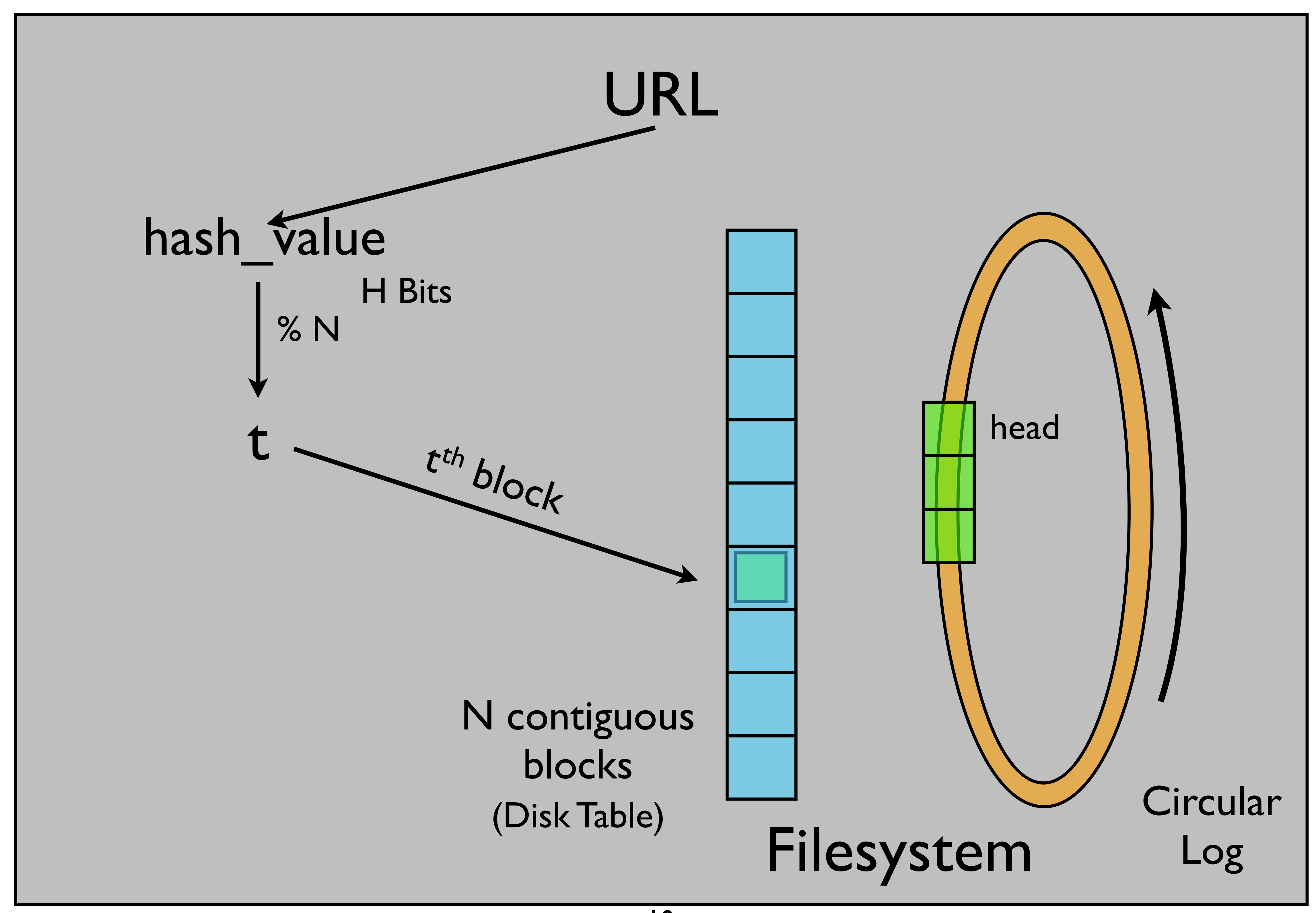
HashCache: Basic Policy



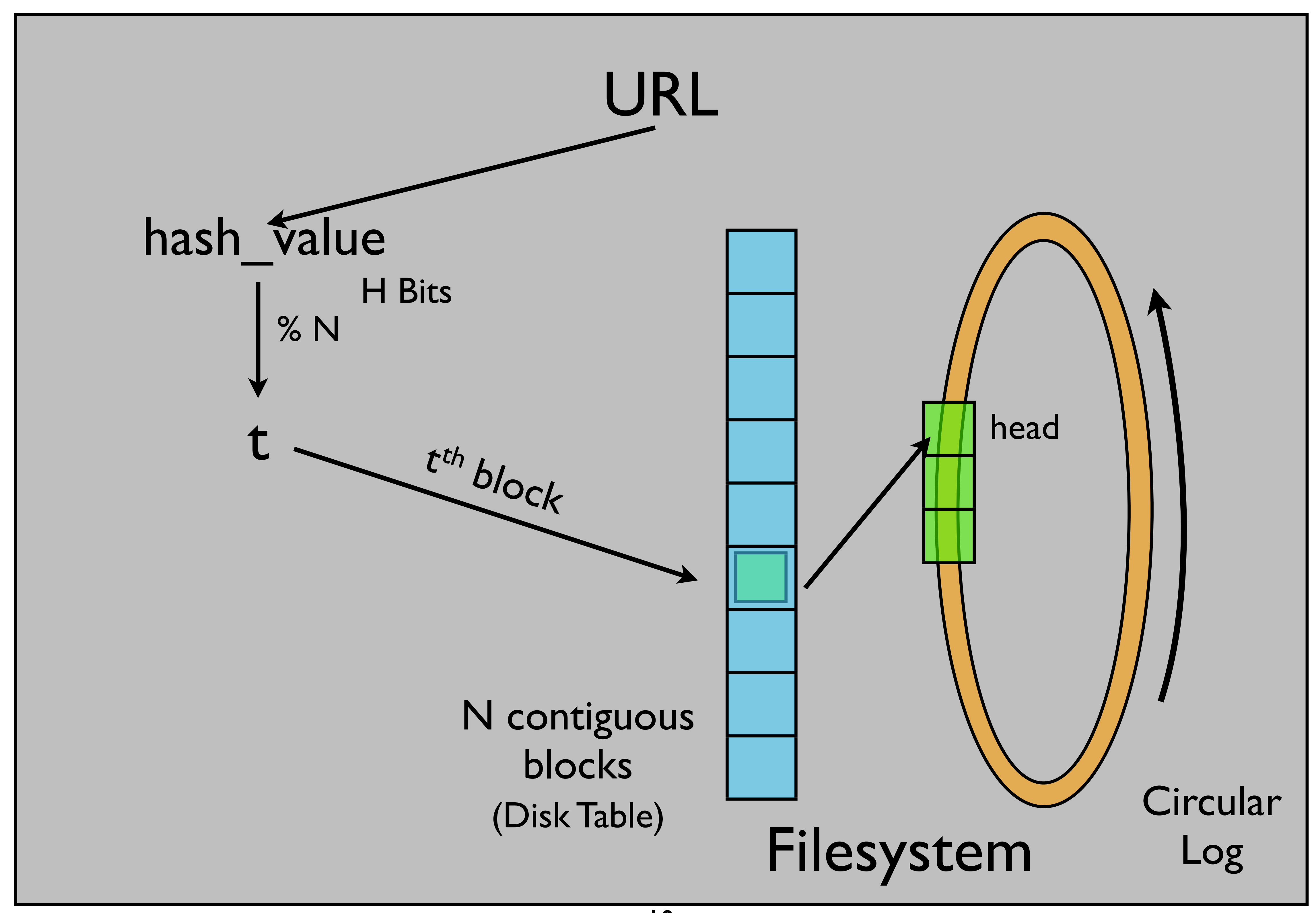
HashCache: Basic Policy



HashCache: Basic Policy



HashCache: Basic Policy



HashCache: Basic Policy

- Advantages
  - No index memory needed
  - Tuned for one seek for most objects

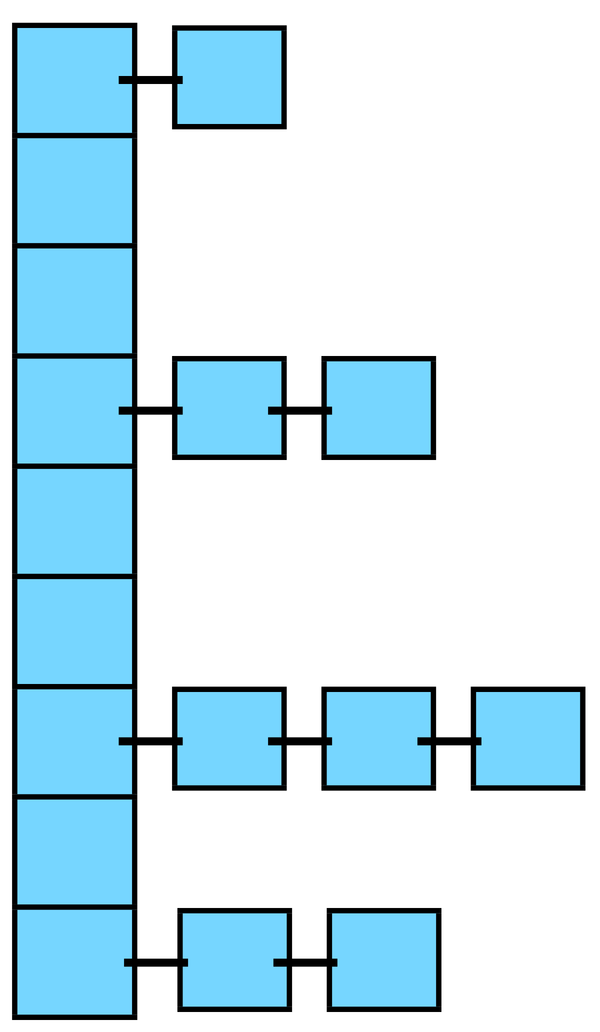
HashCache: Basic Policy

- Advantages
  - No index memory needed
  - Tuned for one seek for most objects
- Disadvantages
  - One seek per miss
  - No collison control
  - No cache replacement policy

Collision Control

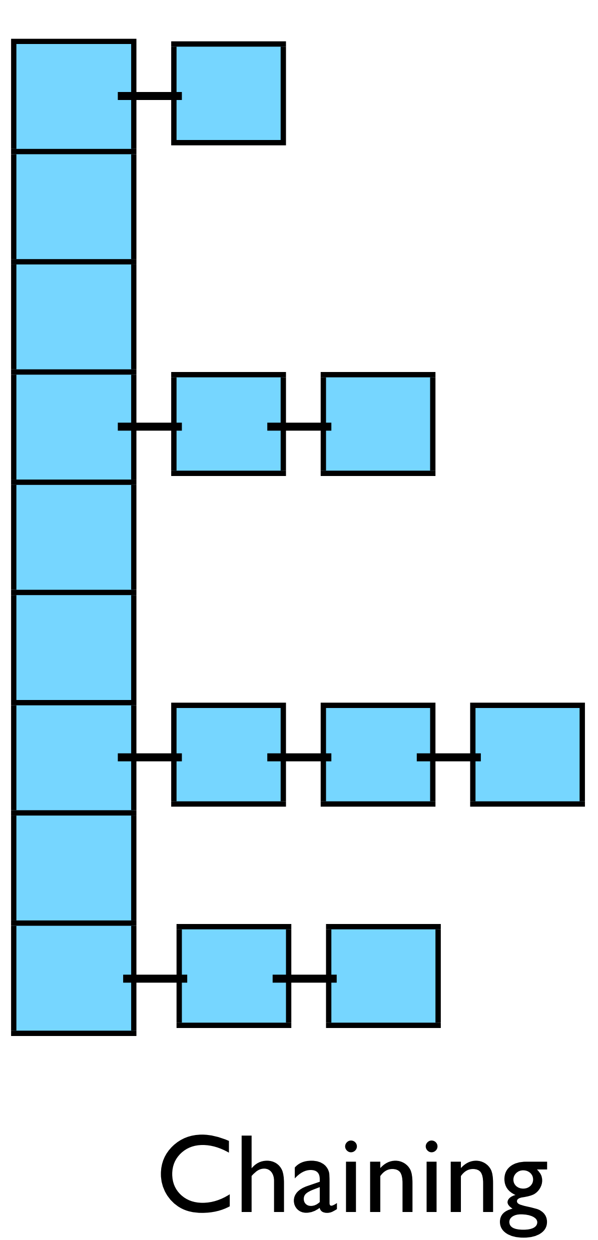
Collision Control

Collision Control



Chaining

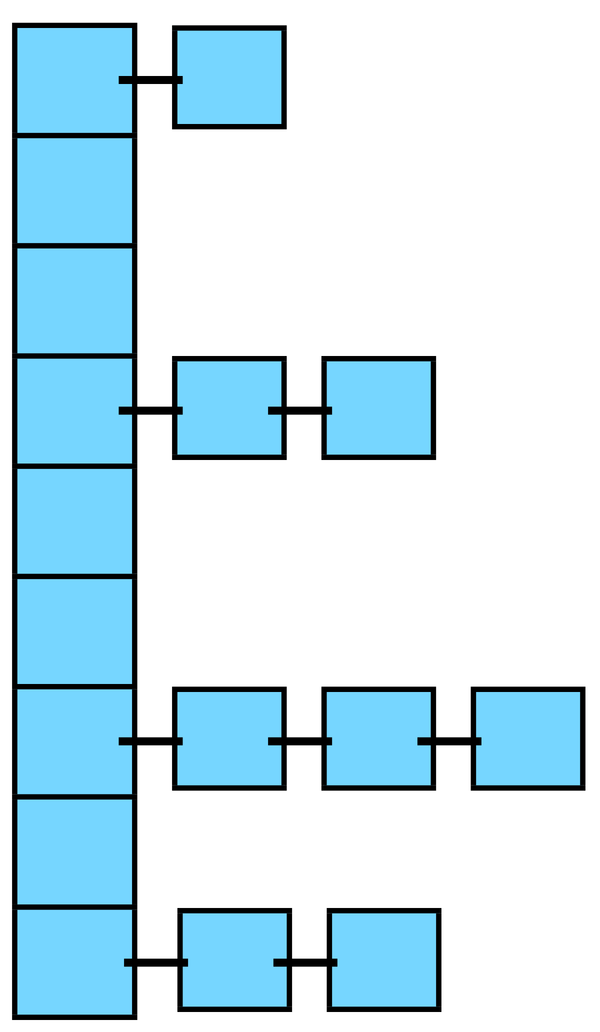
Collision Control



Chaining

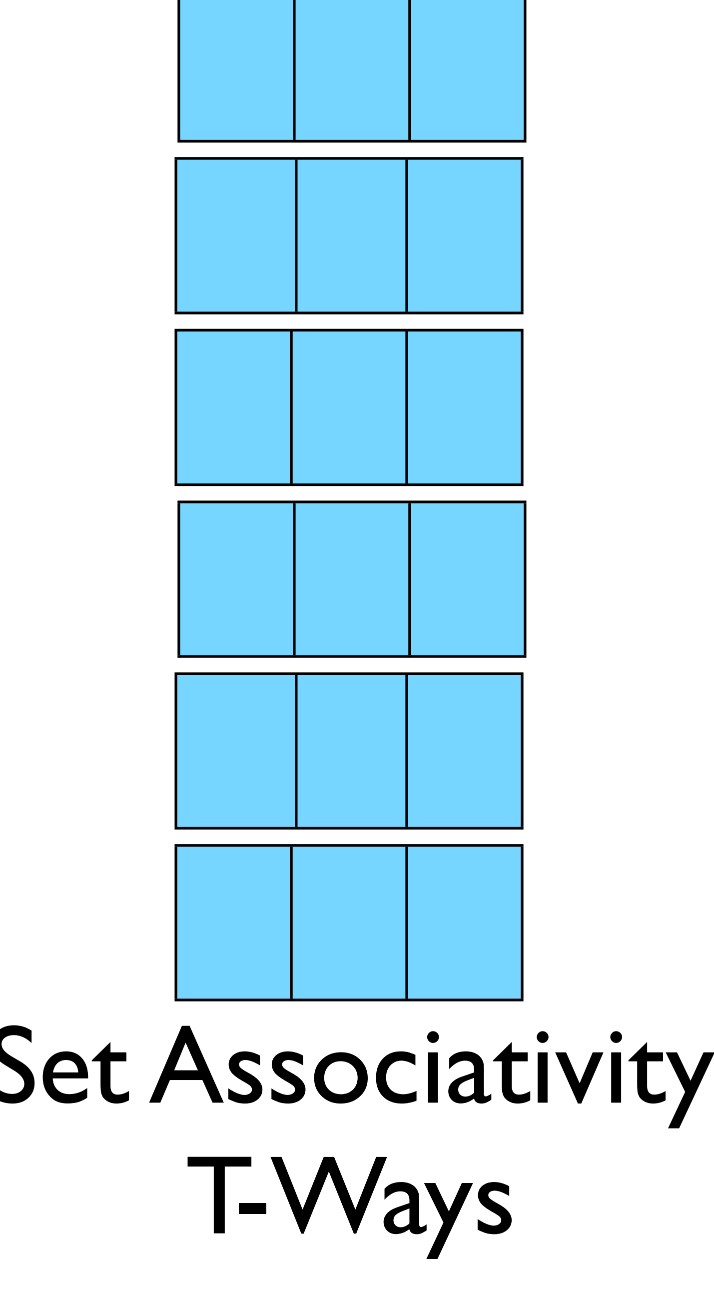
- Does not transition well to disk-based
  - Multiple seeks per operation
  - Walking hash bin list
  - Global replacement policy crosses bins

Collision Control



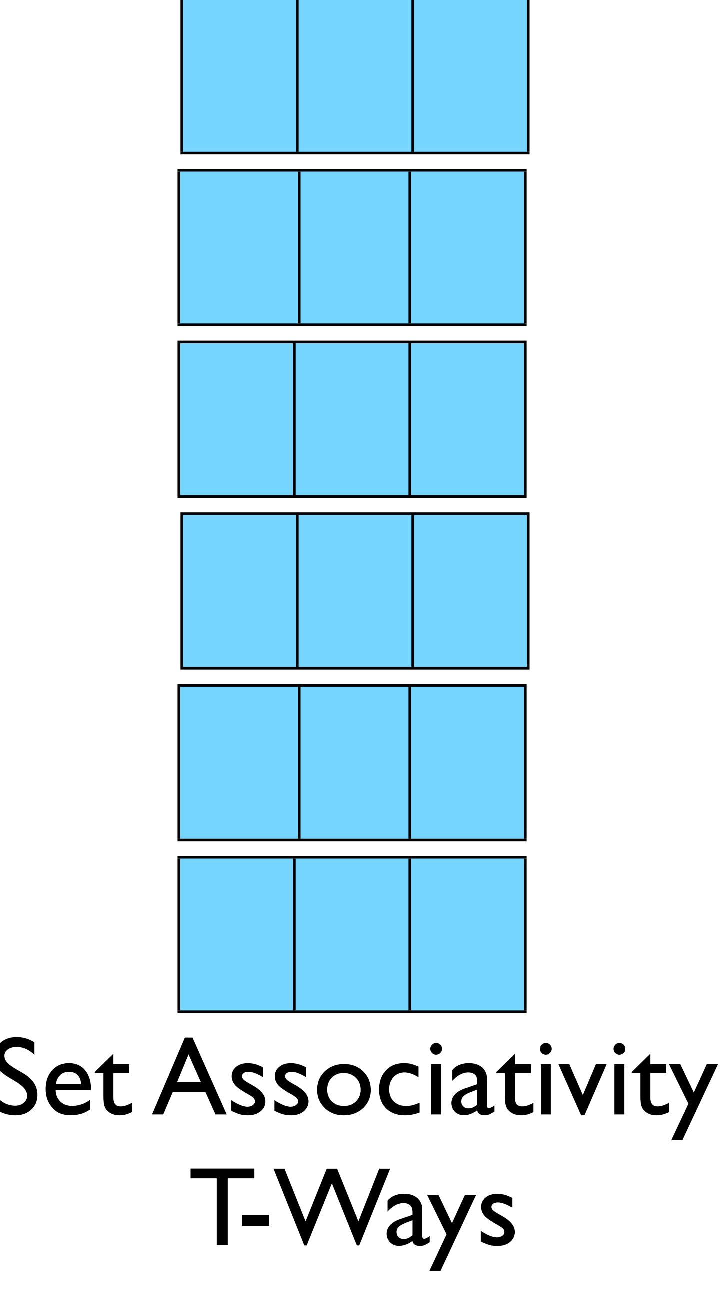
Chaining

Collision Control



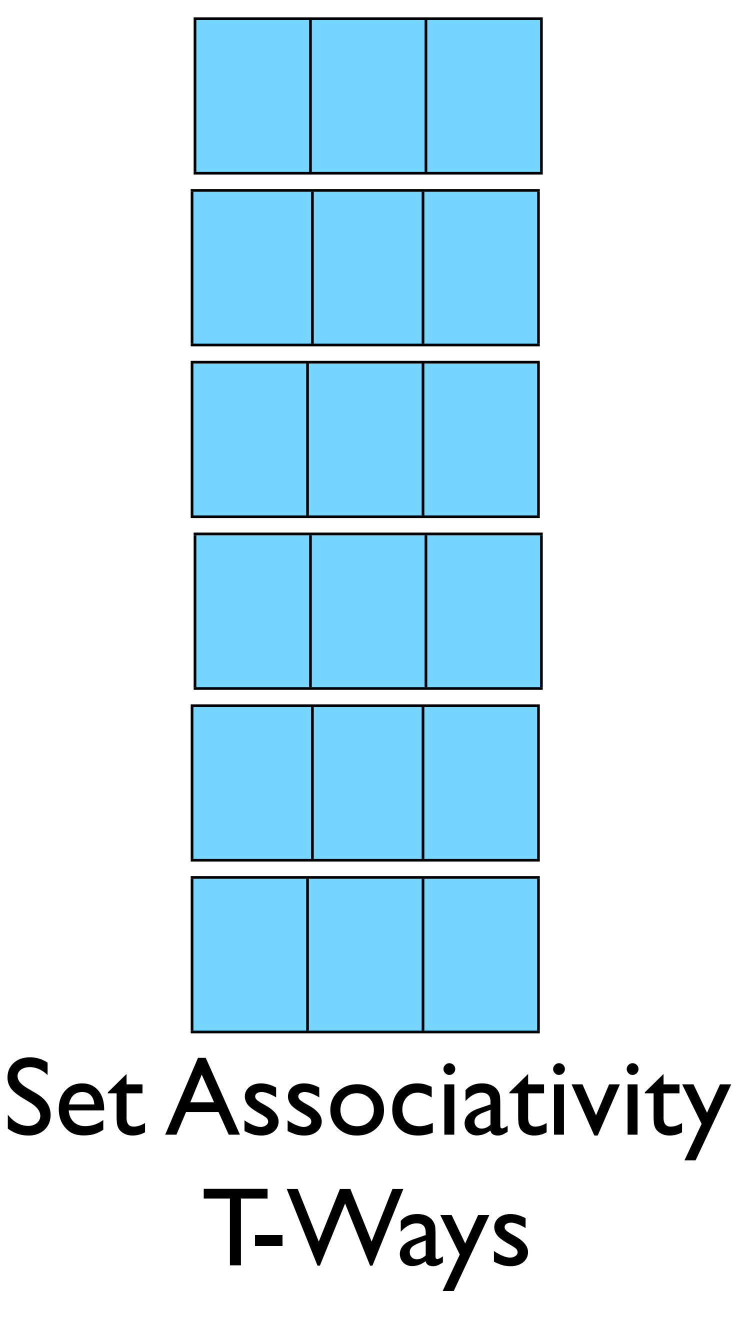
- Fixed locations where each object can be found
  - Allocated contiguously, read together

Collision Control



- Fixed locations where each object can be found
  - Allocated contiguously, read together
- Seek time dominates short read

Collision Control



- Fixed locations where each object can be found
  - Allocated contiguously, read together
- Seek time dominates short read
- Eliminate global cache replacement policies

Reducing Seeks

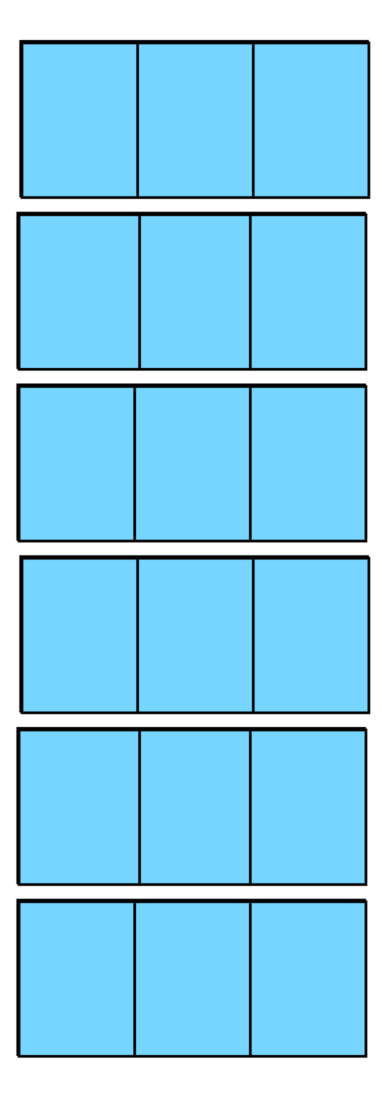
Reducing Seeks

Bin Pointers	32
Chaining Pointers	64
Hash	32
Total (bits)	128

In-memory Hash Table

- In-memory hash table
  - Too much memory for pointers

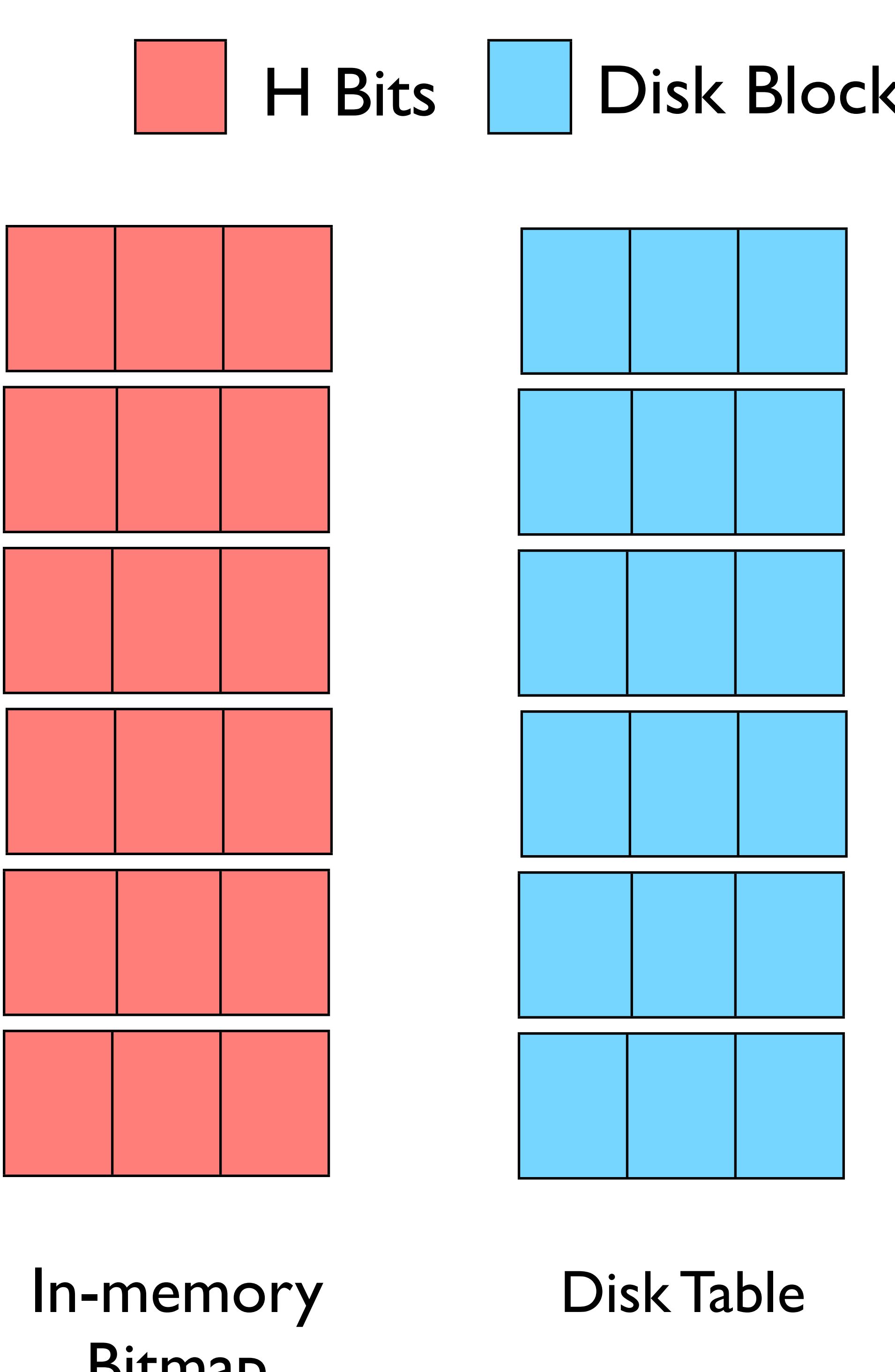
Reducing Seeks



Disk Table

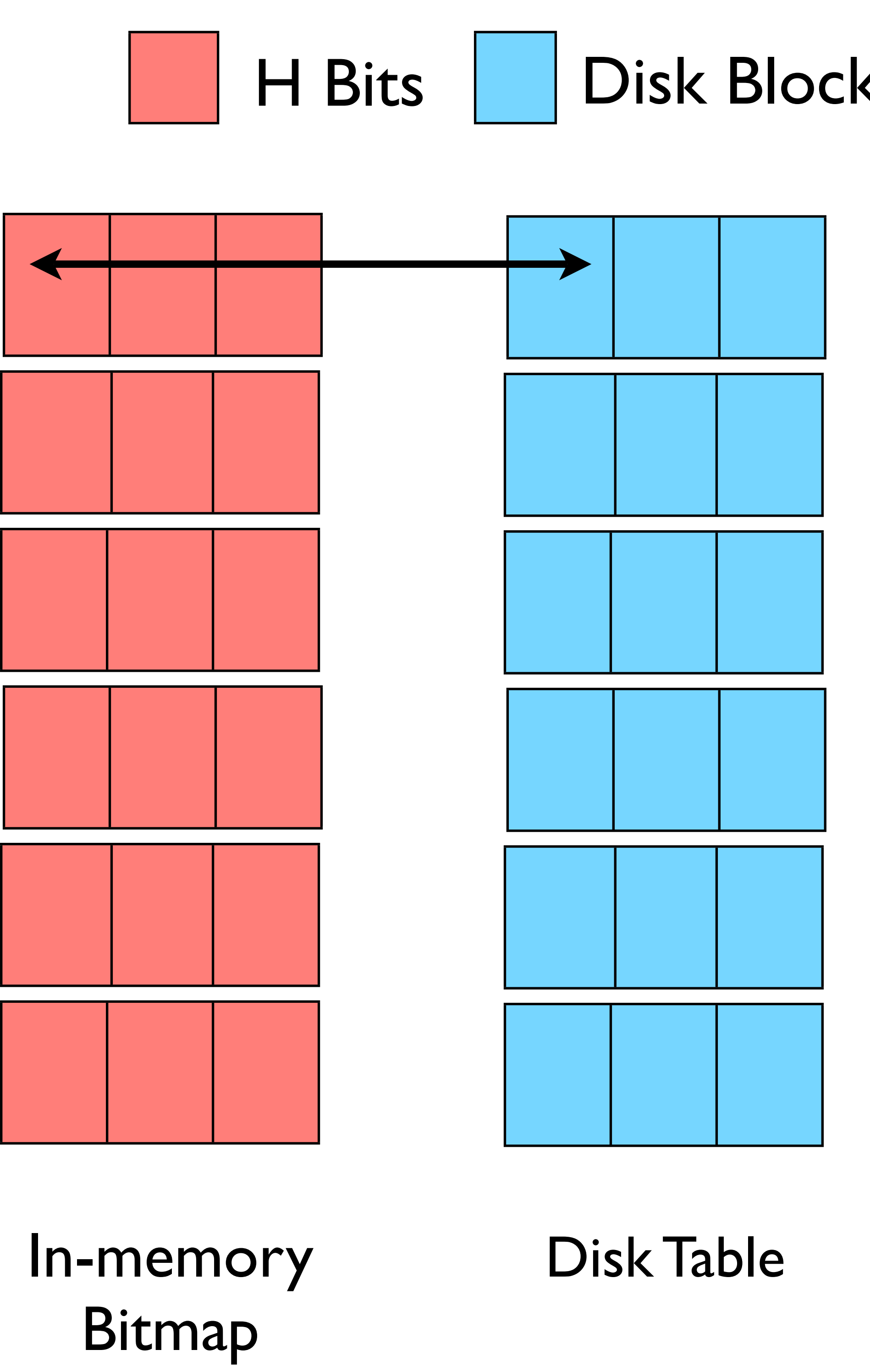
- In-memory hash table
  - Too much memory for pointers
- Disk is already a hash table
  - Pointers not needed
  - Large bitmap with the same layout as the disk

Reducing Seeks



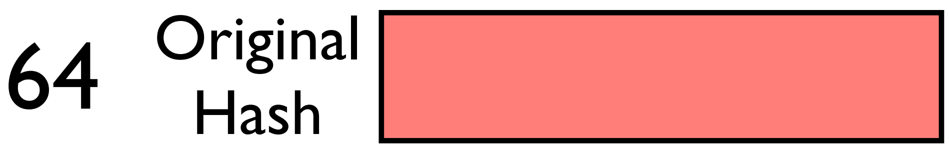
- In-memory hash table
  - Too much memory for pointers
- Disk is already a hash table
  - Pointers not needed
  - Large bitmap with the same layout as the disk
  - Just store hash per URL

Reducing Seeks



- In-memory hash table
  - Too much memory for pointers
- Disk is already a hash table
  - Pointers not needed
  - Large bitmap with the same layout as the disk
  - Just store hash per URL

Reducing Seeks



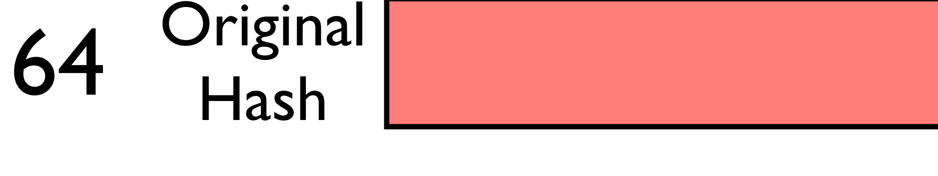
- Original hash of the URL: 64 bits

Reducing Seeks

64	Original Hash	
39	64 - log(S)	

- Original hash of the URL: 64 bits
- Eliminate bits for (same) bin # (2^{28} objs, 8-way, #bins=2^{25} (S))

Reducing Seeks

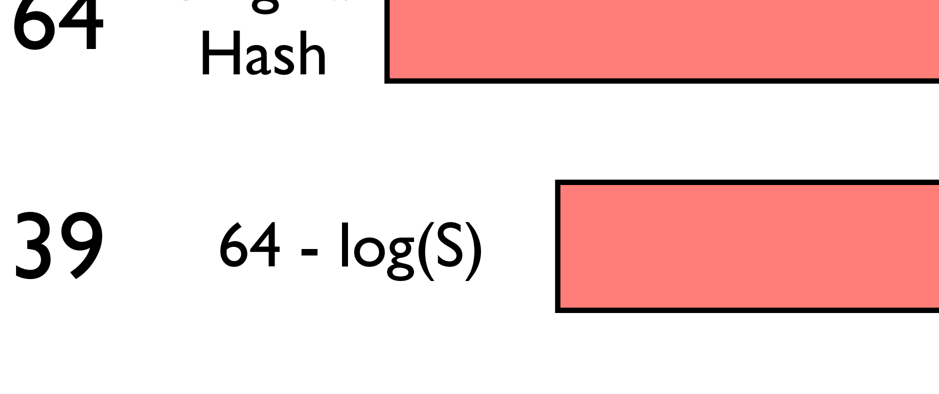


39 64 - log(S)

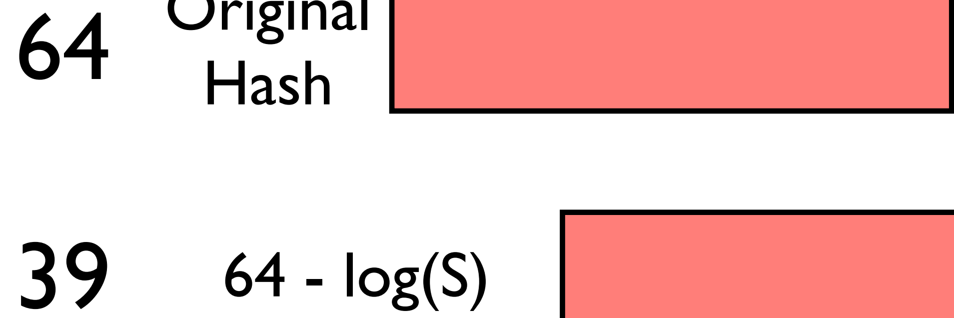
8 low FP hash

- Original hash of the URL: 64 bits
- Eliminate bits for (same) bin # (2^{28} objs, 8-way, #bins=2^{25} (S))
- Shrink hash size: Just to eliminate most false positives (8 bits)

Cache Replacement



Cache Replacement



- Large disks: 10-100+ million objects
    - Global caching relevant when disk size ≈ working set
    - When disk >> working set, local policies ≈ global policies

Cache Replacement

- Large disks: 10-100+ million objects
  - Global caching relevant when disk size ≈ working set
  - When disk >> working set, local policies ≈ global policies
- Local replacement benefits
  - 3 bits per URL
  - Performed on contiguous objects
  - False positives limited by set size

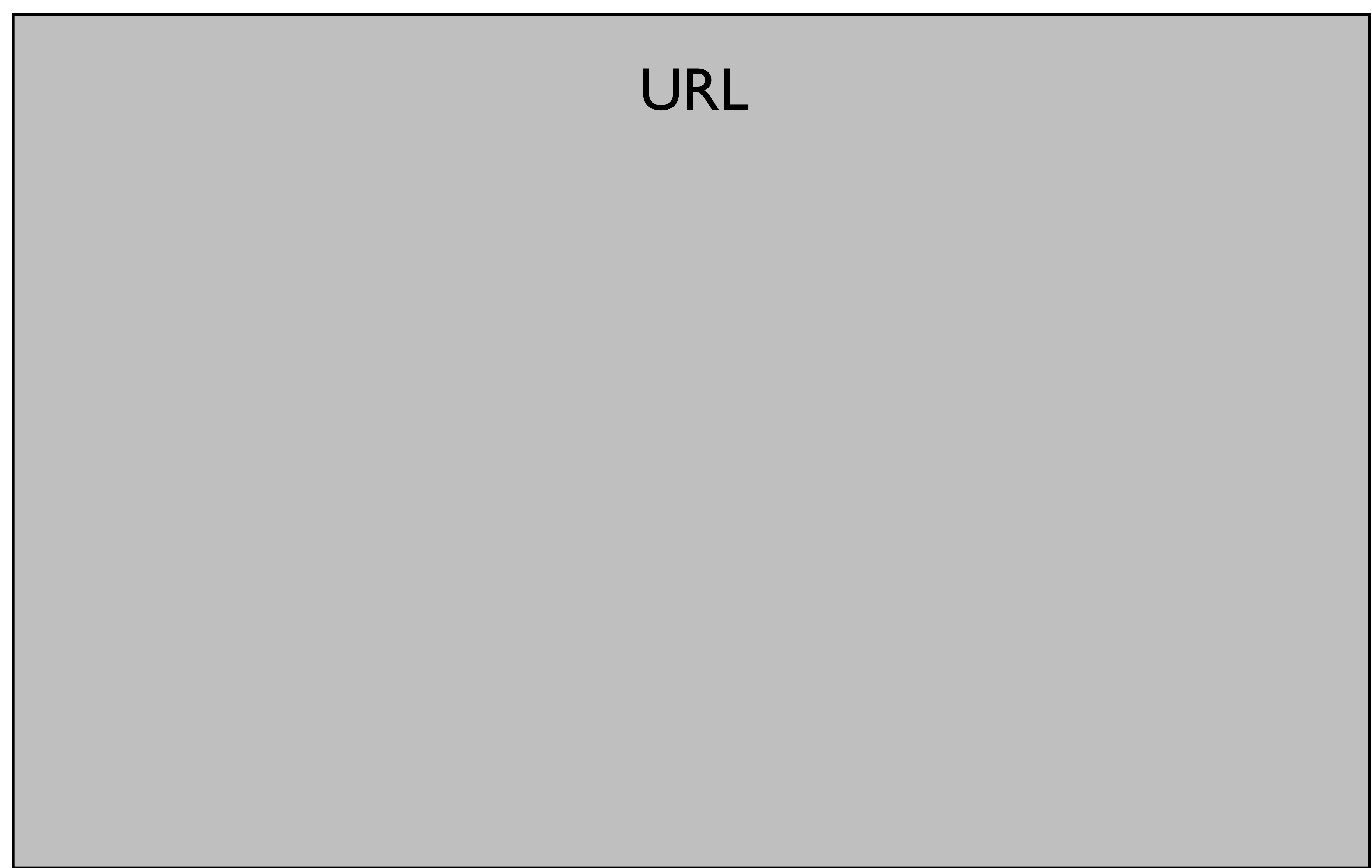
39 64 - log(S)

8 low FP hash

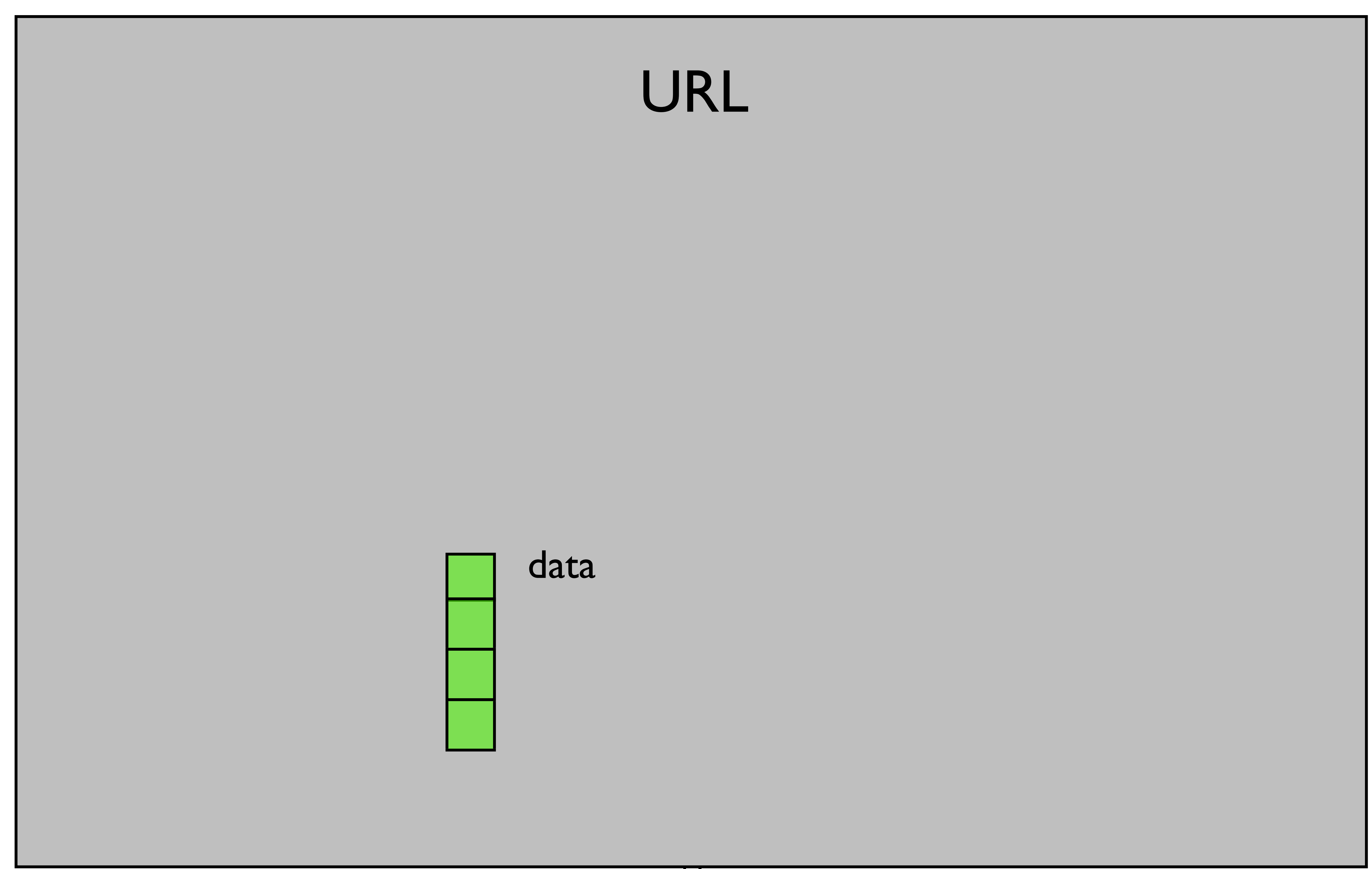
11 hash + rank

HashCache: SetMem Policy

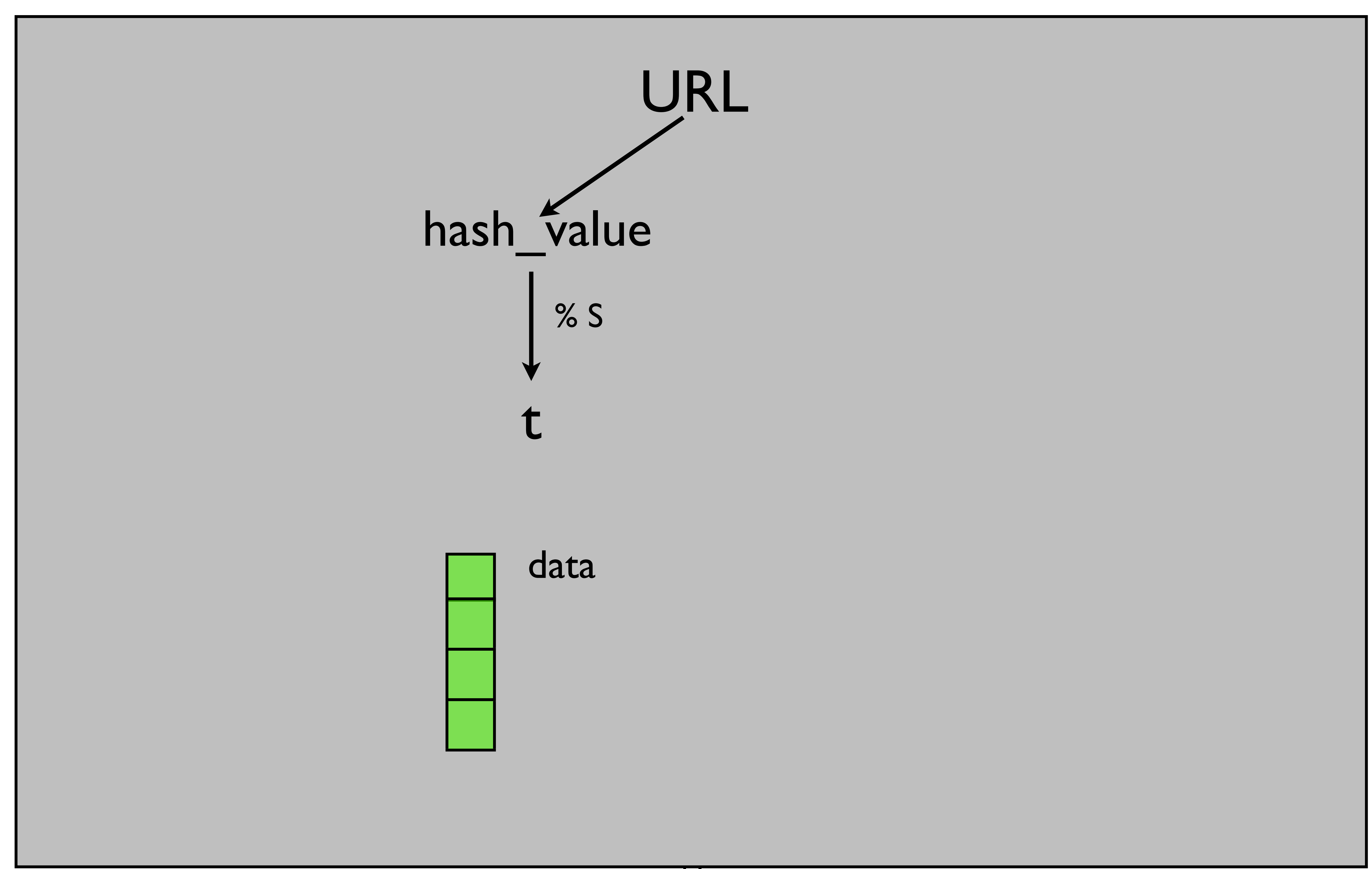
HashCache: SetMem Policy



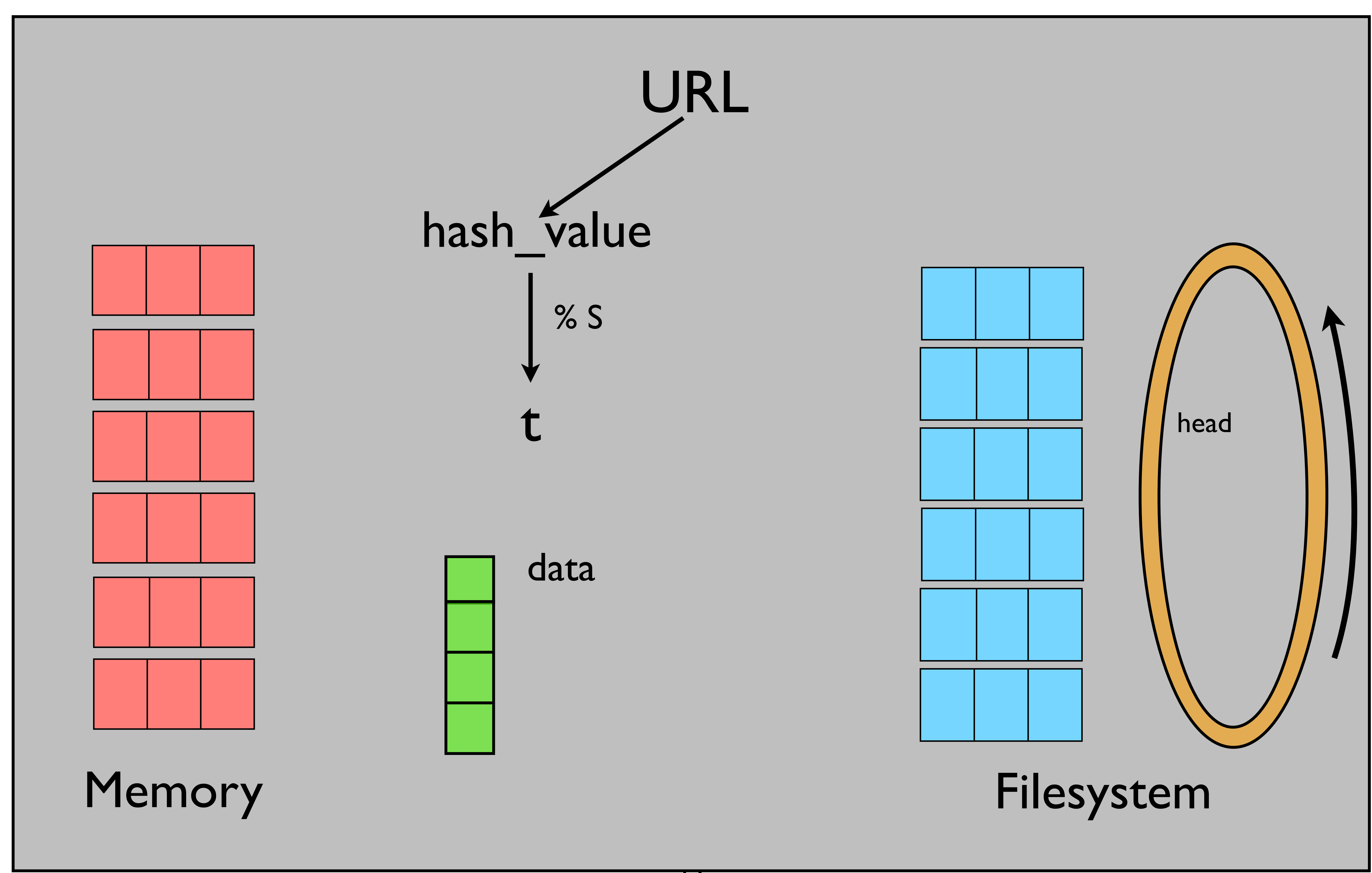
HashCache: SetMem Policy



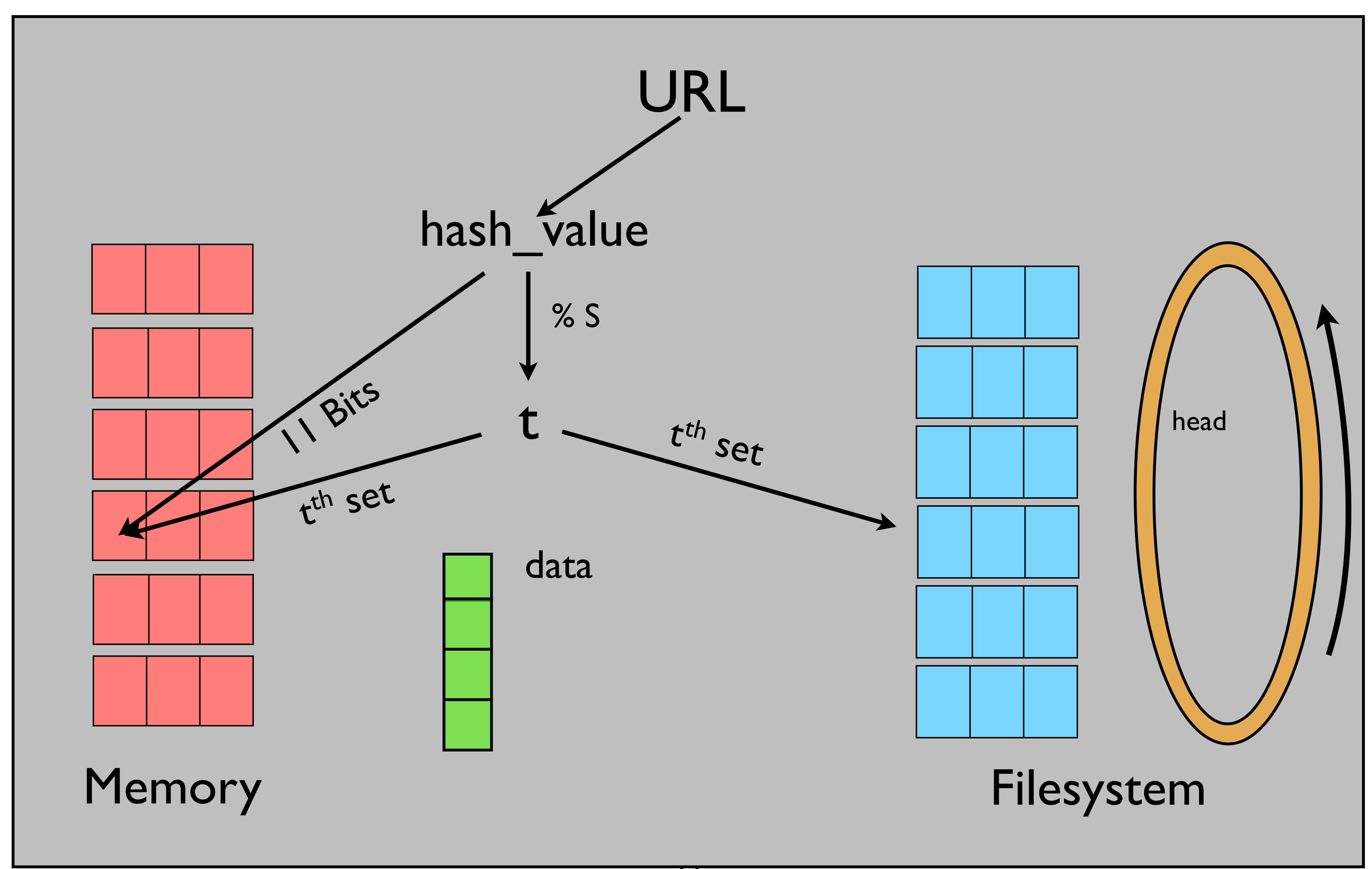
HashCache: SetMem Policy



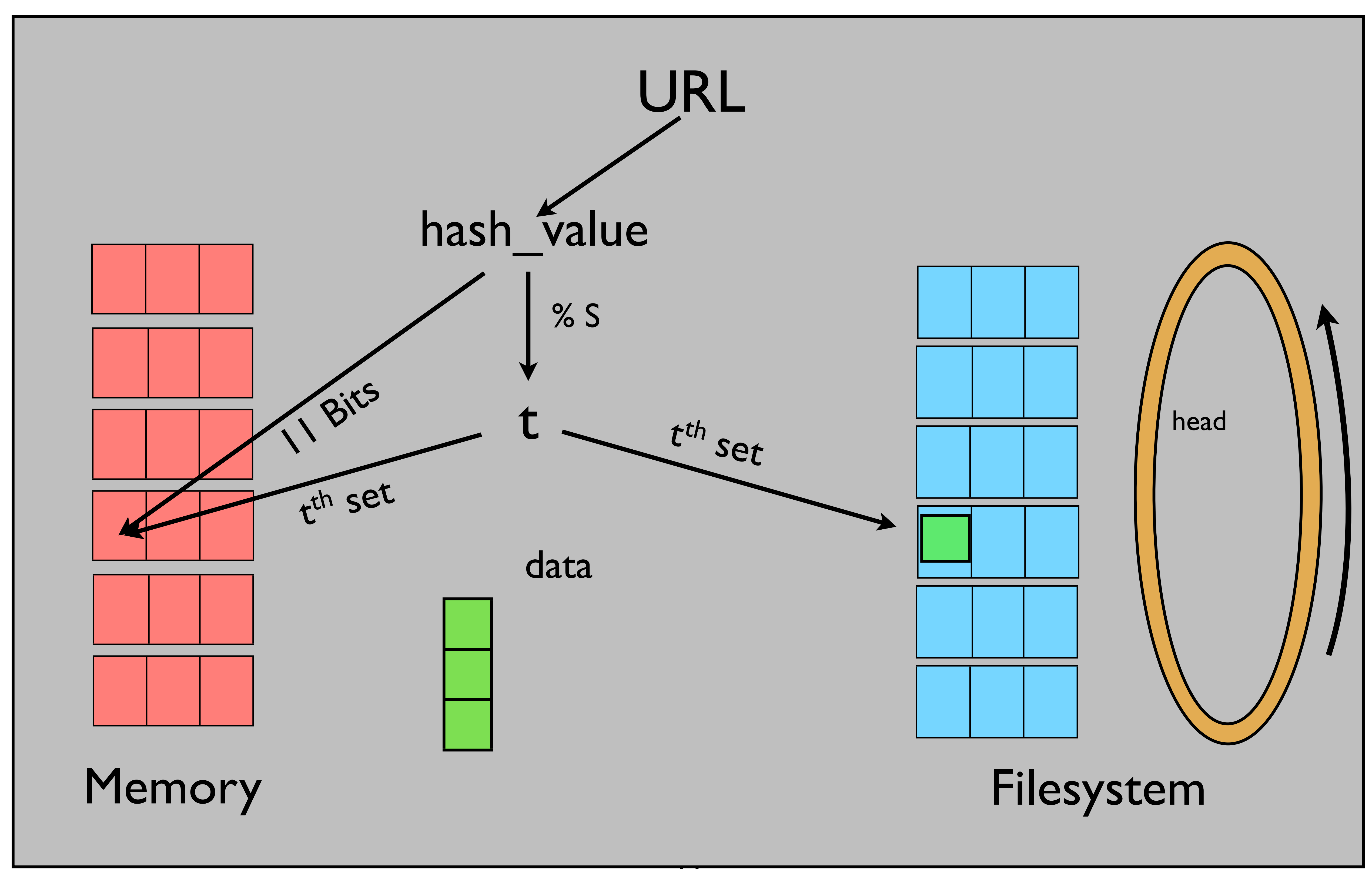
HashCache: SetMem Policy



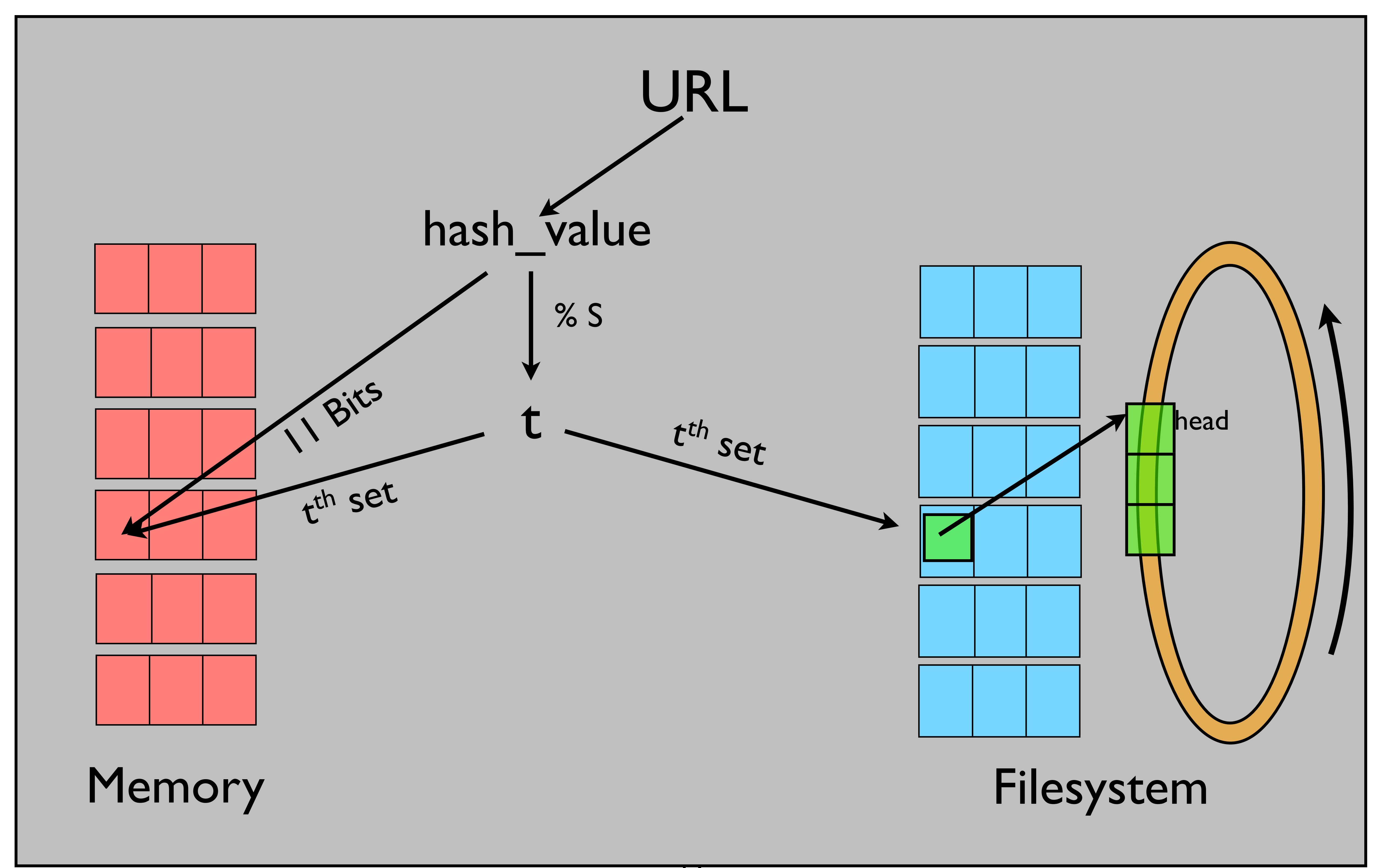
HashCache: SetMem Policy



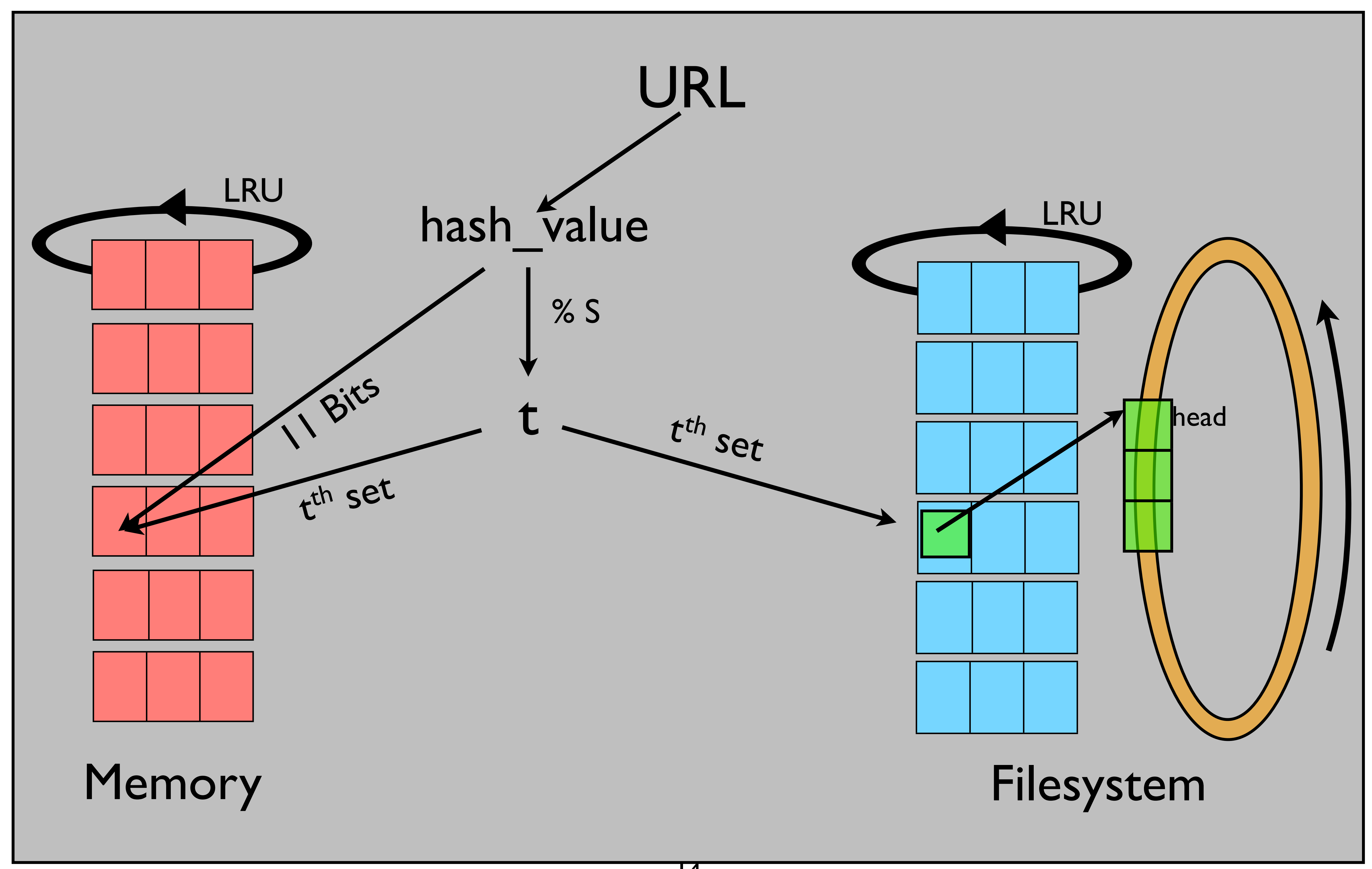
HashCache: SetMem Policy



HashCache: SetMem Policy



HashCache: SetMem Policy



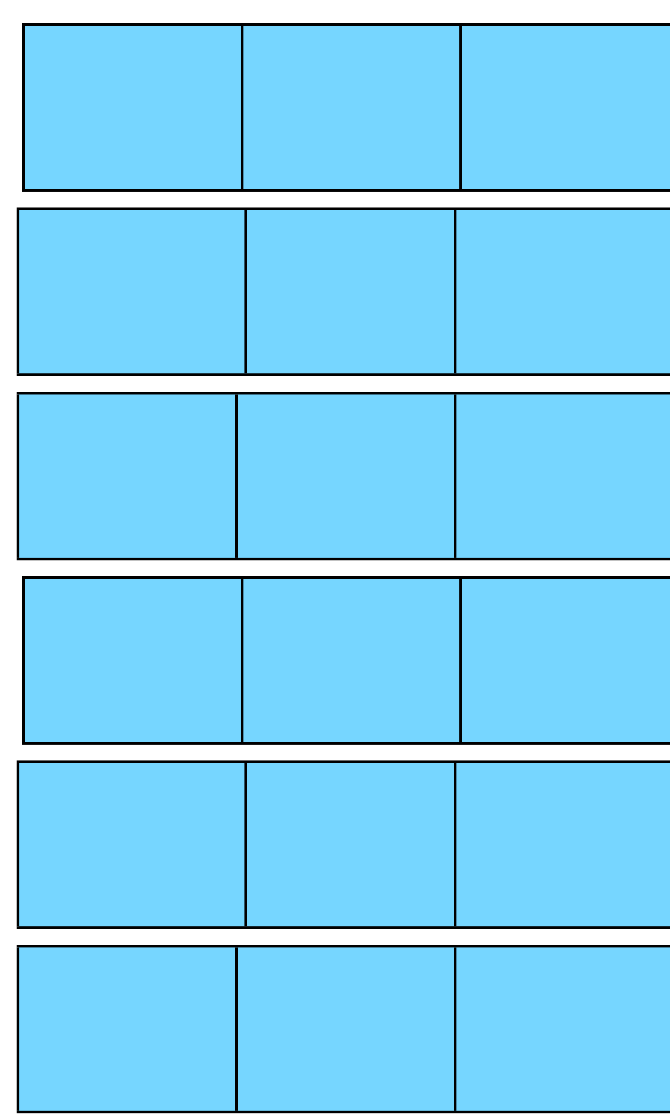
HashCache: SetMem Policy

- Advantages
  - No seeks for most misses
  - 1 seek per read, 1 seek per write
  - Good hash + replacement in 11 bits

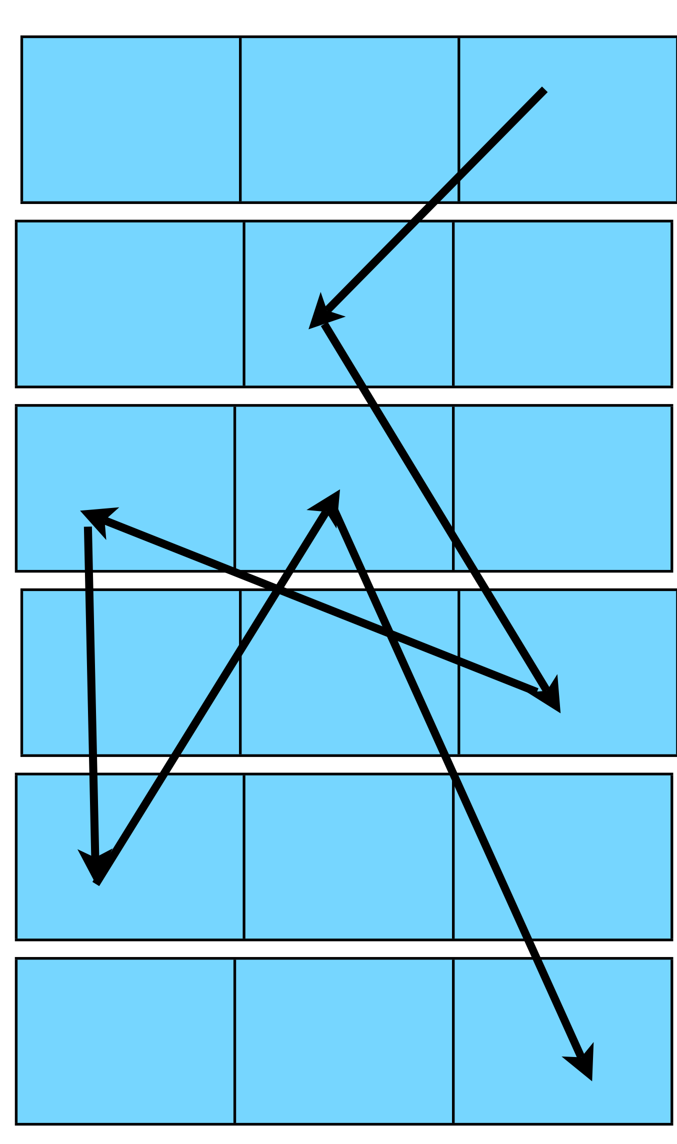
HashCache: SetMem Policy

- Advantages
  - No seeks for most misses
  - 1 seek per read, 1 seek per write
  - Good hash + replacement in 11 bits
- Disadvantages
  - Writes still need seeks

Further Reducing Seeks

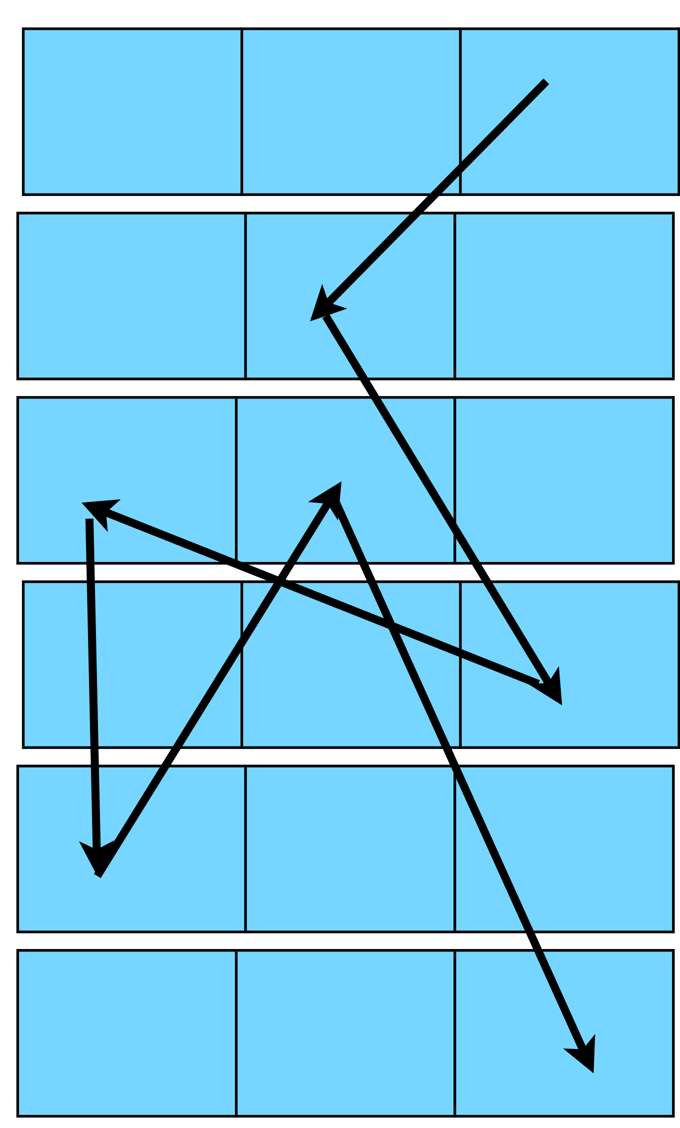


Further Reducing Seeks



- Storing objects by hash can produce random reads & writes

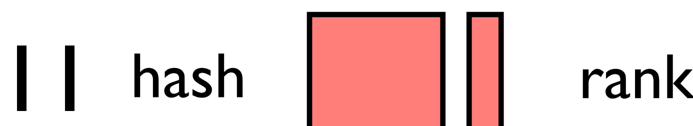
Further Reducing Seeks



Disk Table

- Storing objects by hash can produce random reads & writes
- Restructure on-disk table

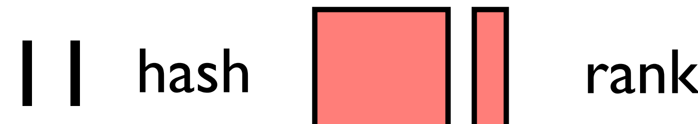
Further Reducing Seeks

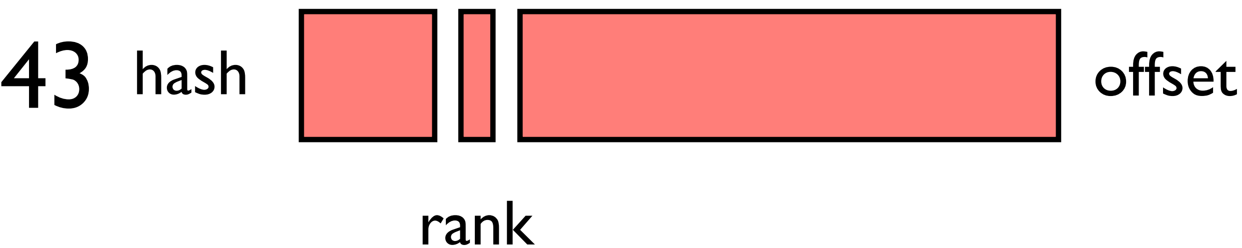


- Storing objects by hash can produce random reads & writes
- Restructure on-disk table
  - Store only hash, rank, offset

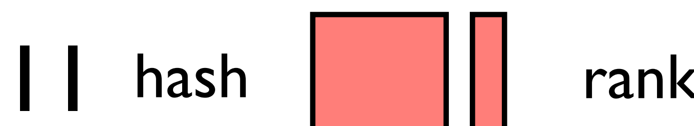
Further Reducing Seeks

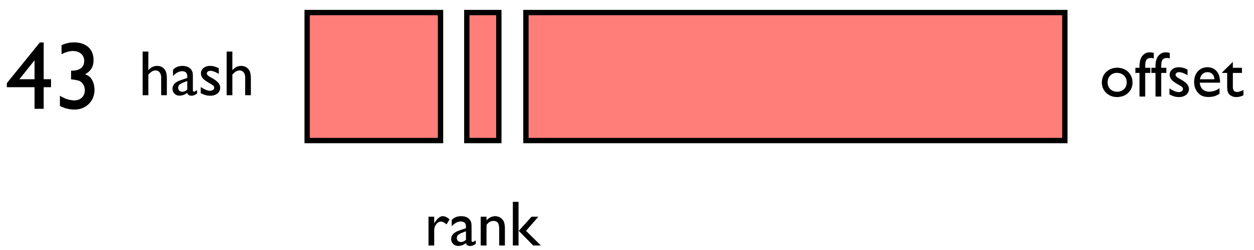
- Storing objects by hash can produce random reads & writes
- Restructure on-disk table
  - Store only hash, rank, offset
  - Move all data to log





Further Reducing Seeks





- Storing objects by hash can produce random reads & writes
- Restructure on-disk table
  - Store only hash, rank, offset
  - Move all data to log
- Benefits
  - Group writes amortize seeks
  - Scheduling related writes enables read prefetch
  - Both reads & writes < 1 seek

Further Reducing Seeks

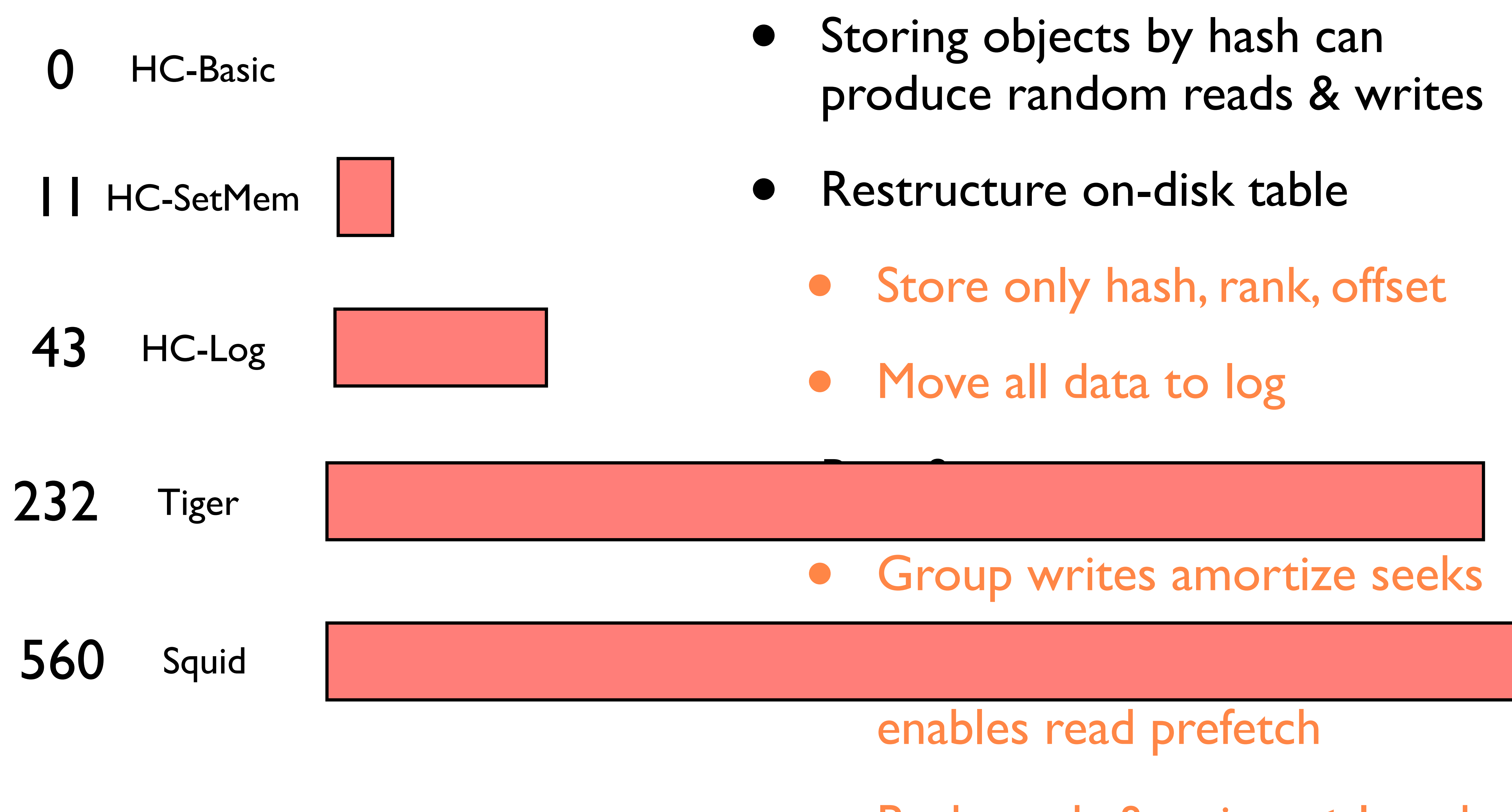
0 HC-Basic

11 HC-SetMem

43 HC-Log

- Storing objects by hash can produce random reads & writes
- Restructure on-disk table
  - Store only hash, rank, offset
  - Move all data to log
- Benefits
  - Group writes amortize seeks
  - Scheduling related writes enables read prefetch
  - Both reads & writes < 1 seek

Further Reducing Seeks

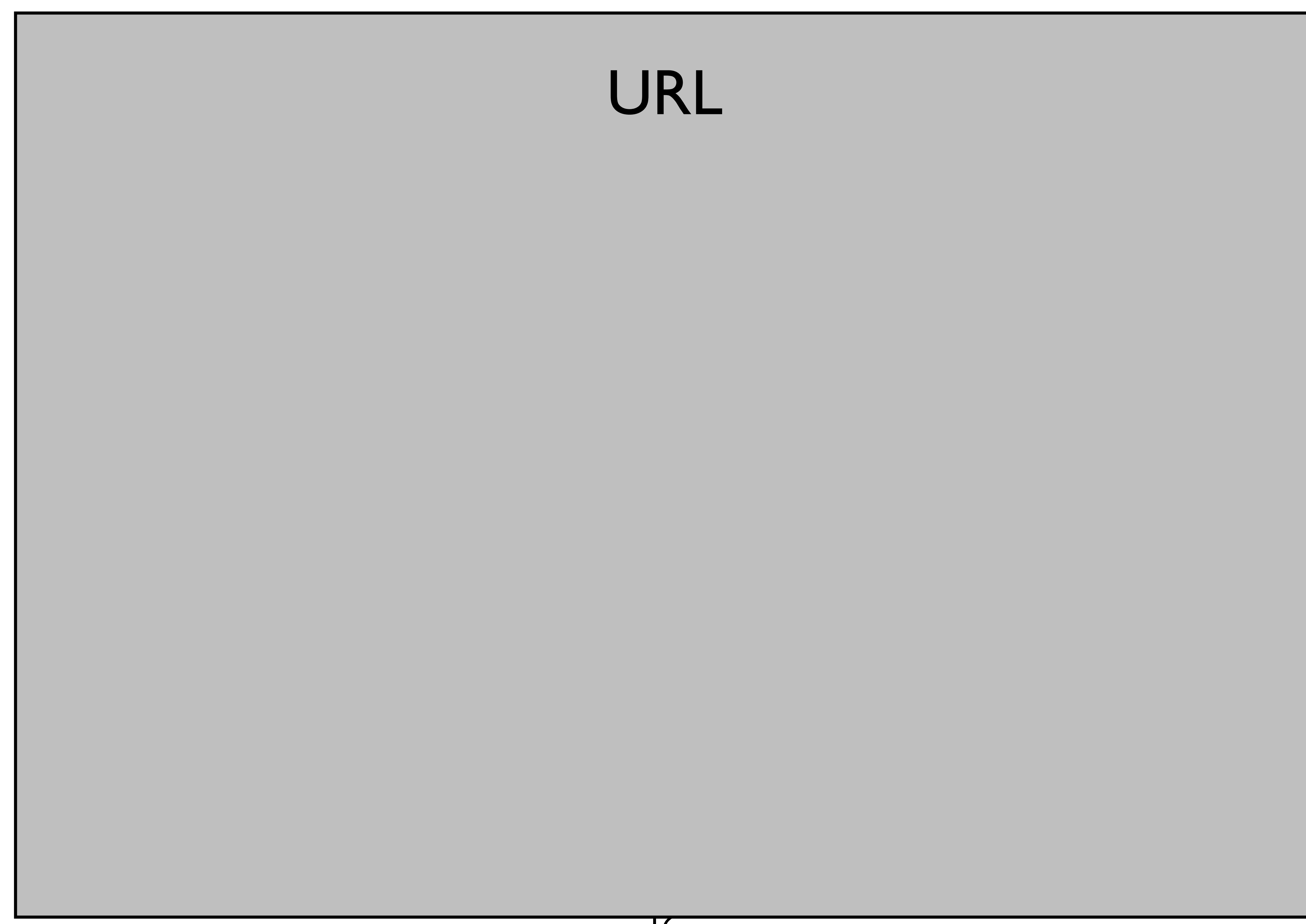


- Storing objects by hash can produce random reads & writes
- Restructure on-disk table
  - Store only hash, rank, offset
  - Move all data to log
  - Group writes amortize seeks

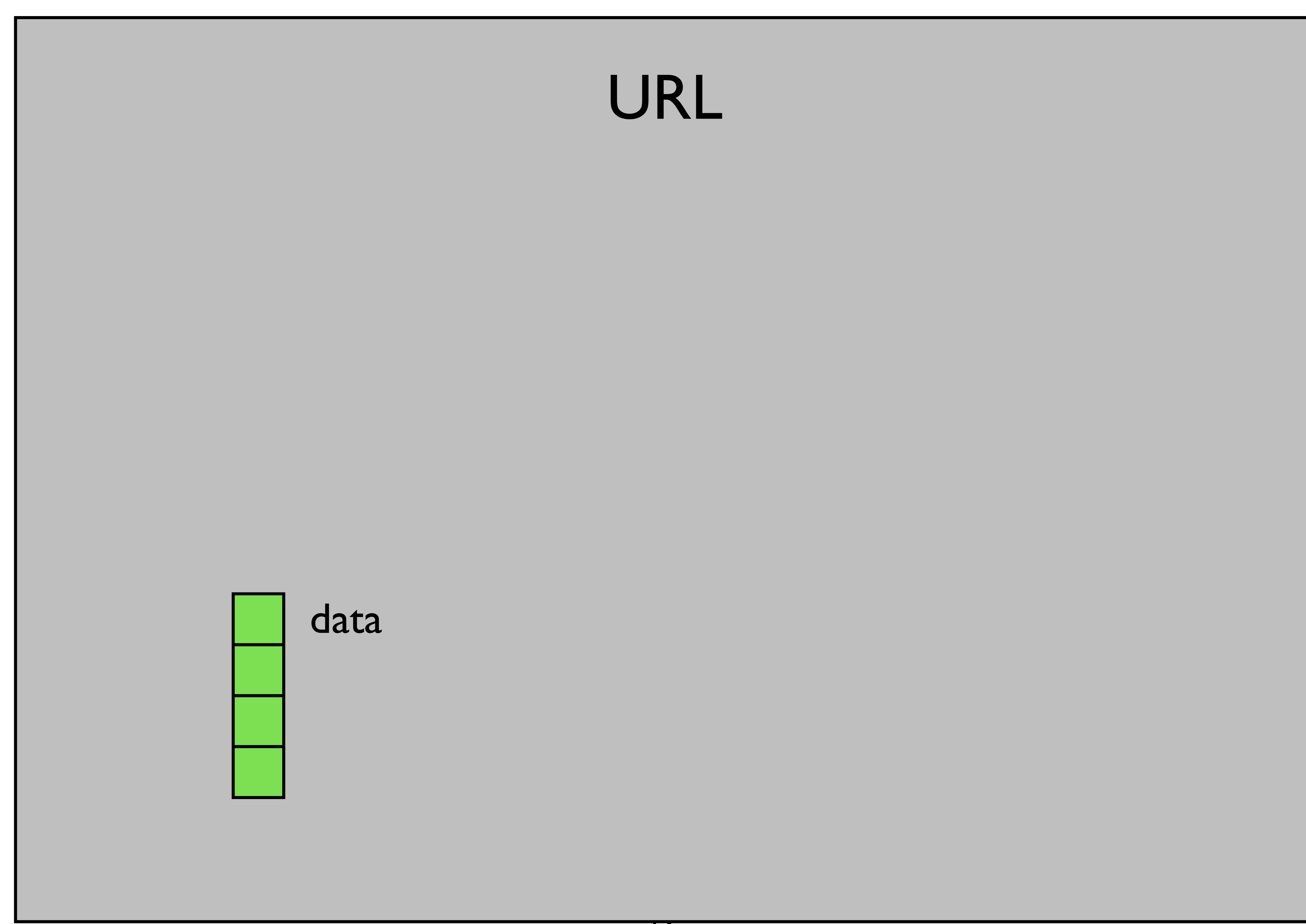
    enables read prefetch
  - Both reads & writes < 1 seek

HashCache: Log Policy

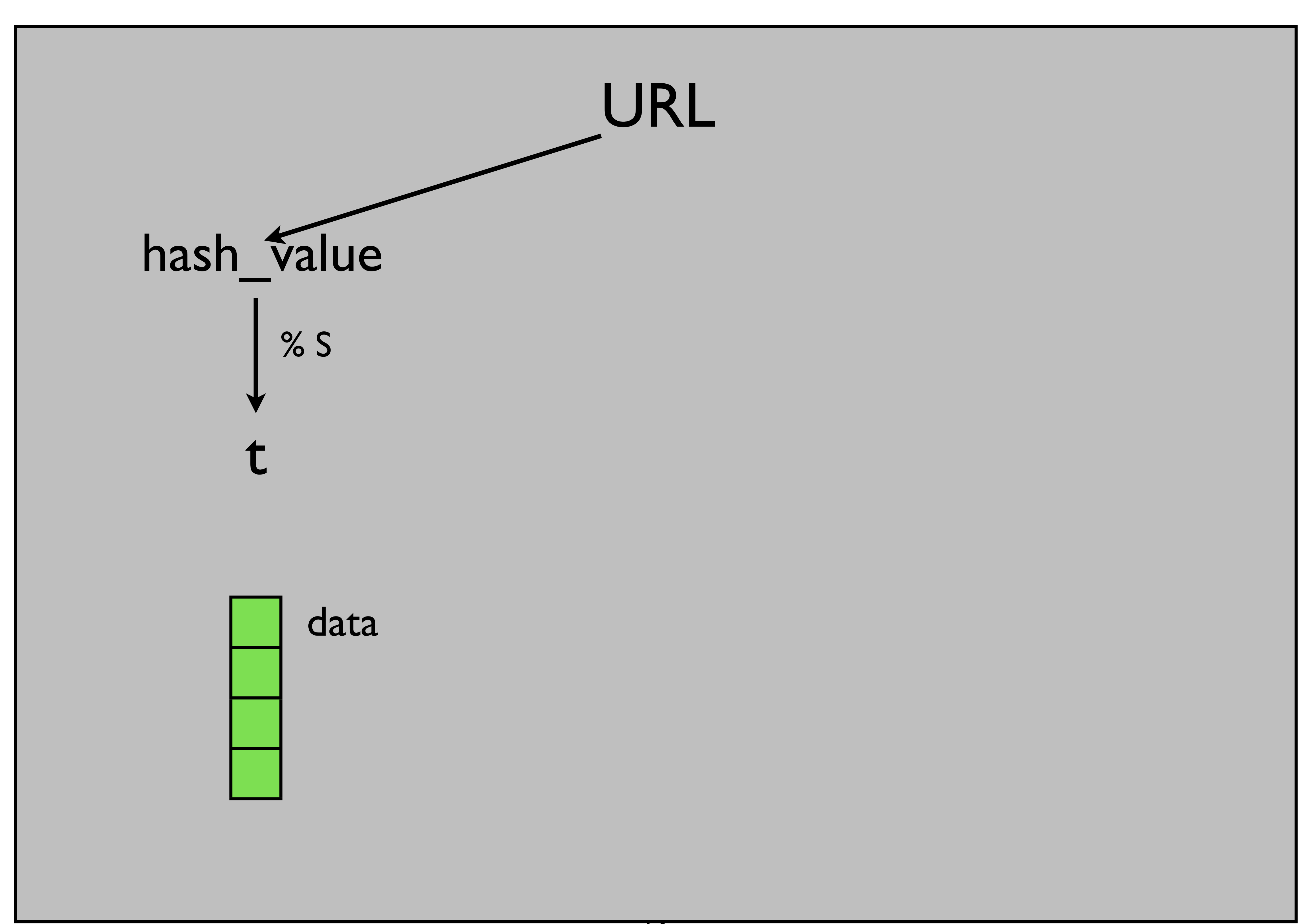
HashCache: Log Policy



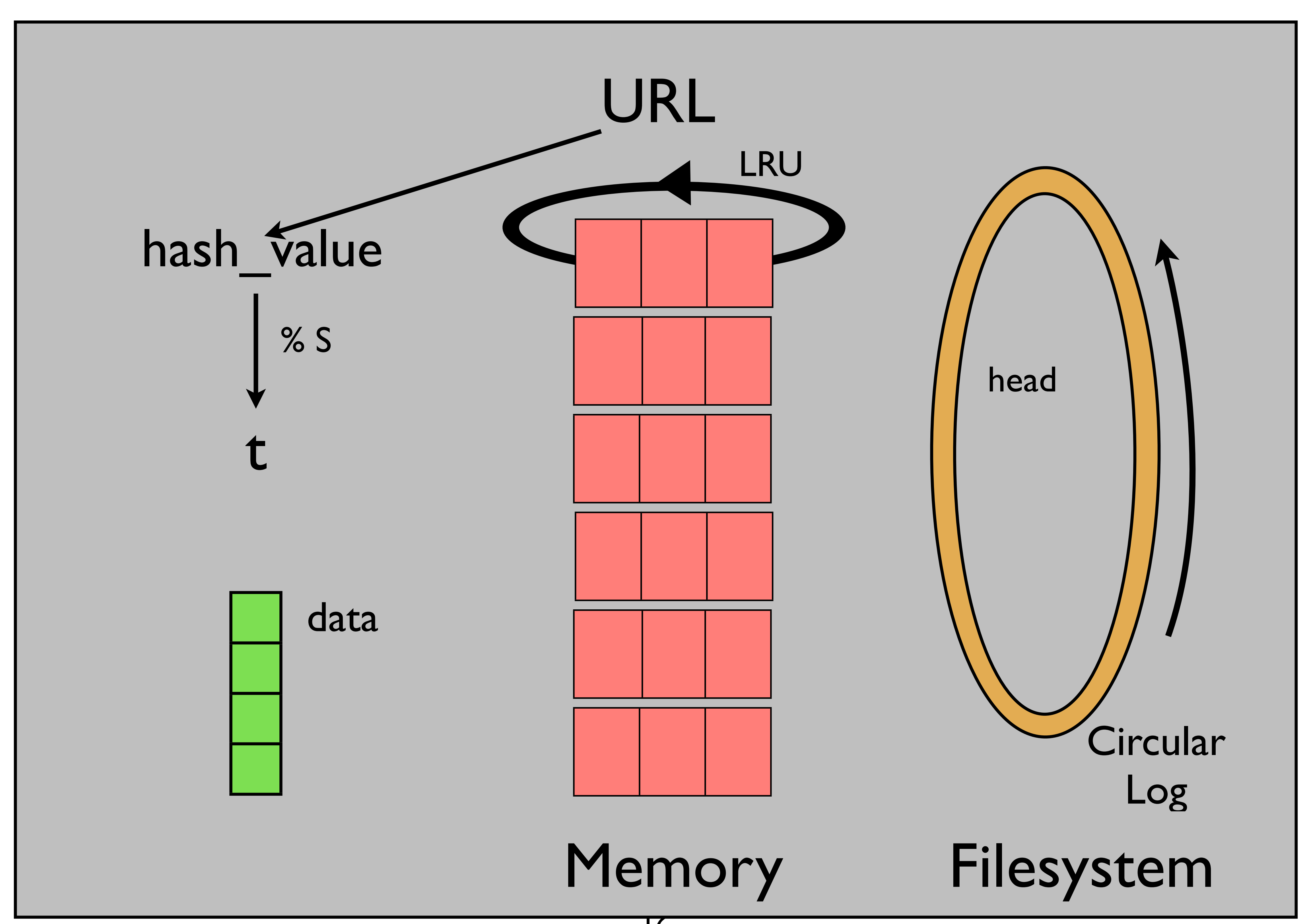
HashCache: Log Policy



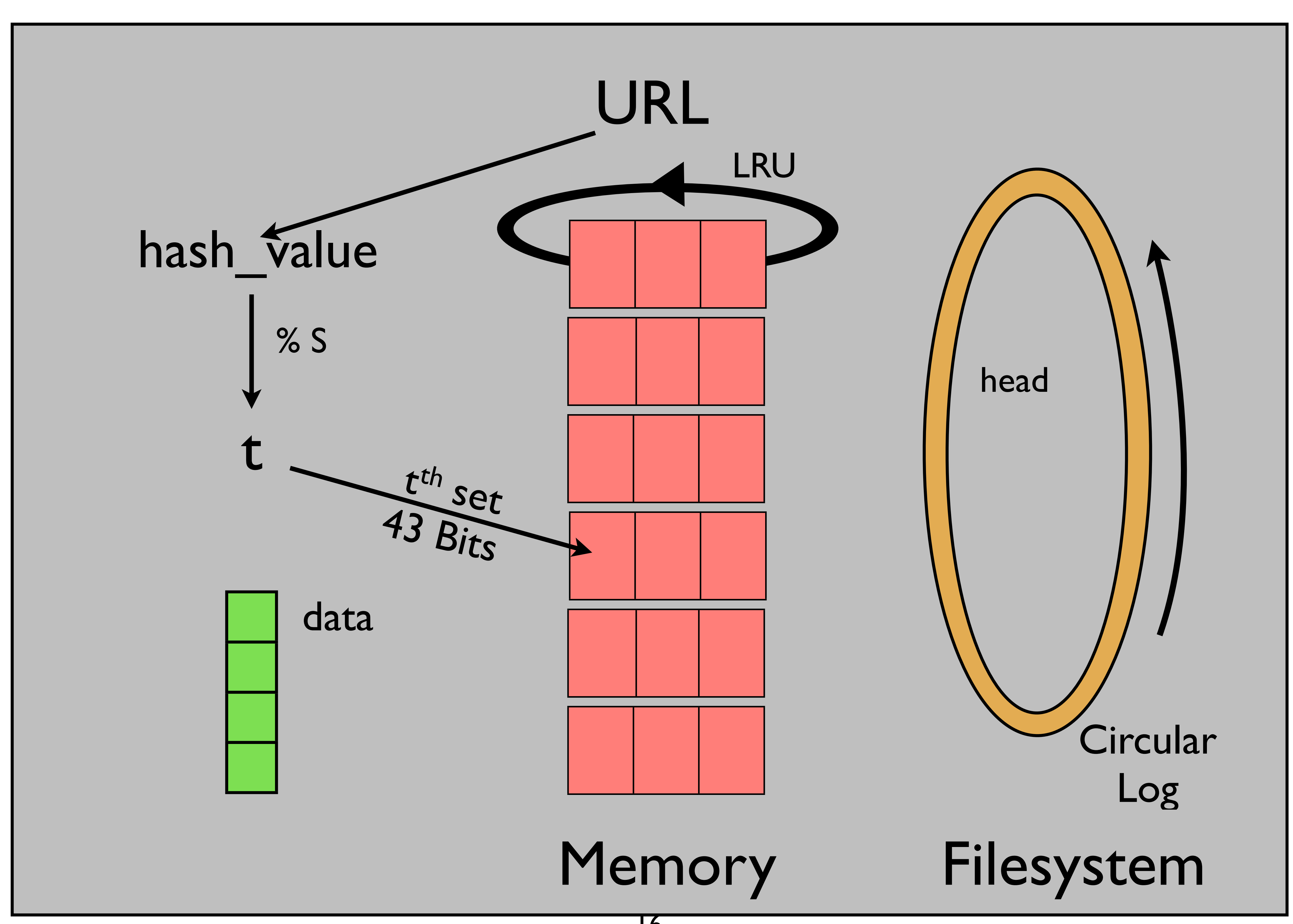
HashCache: Log Policy



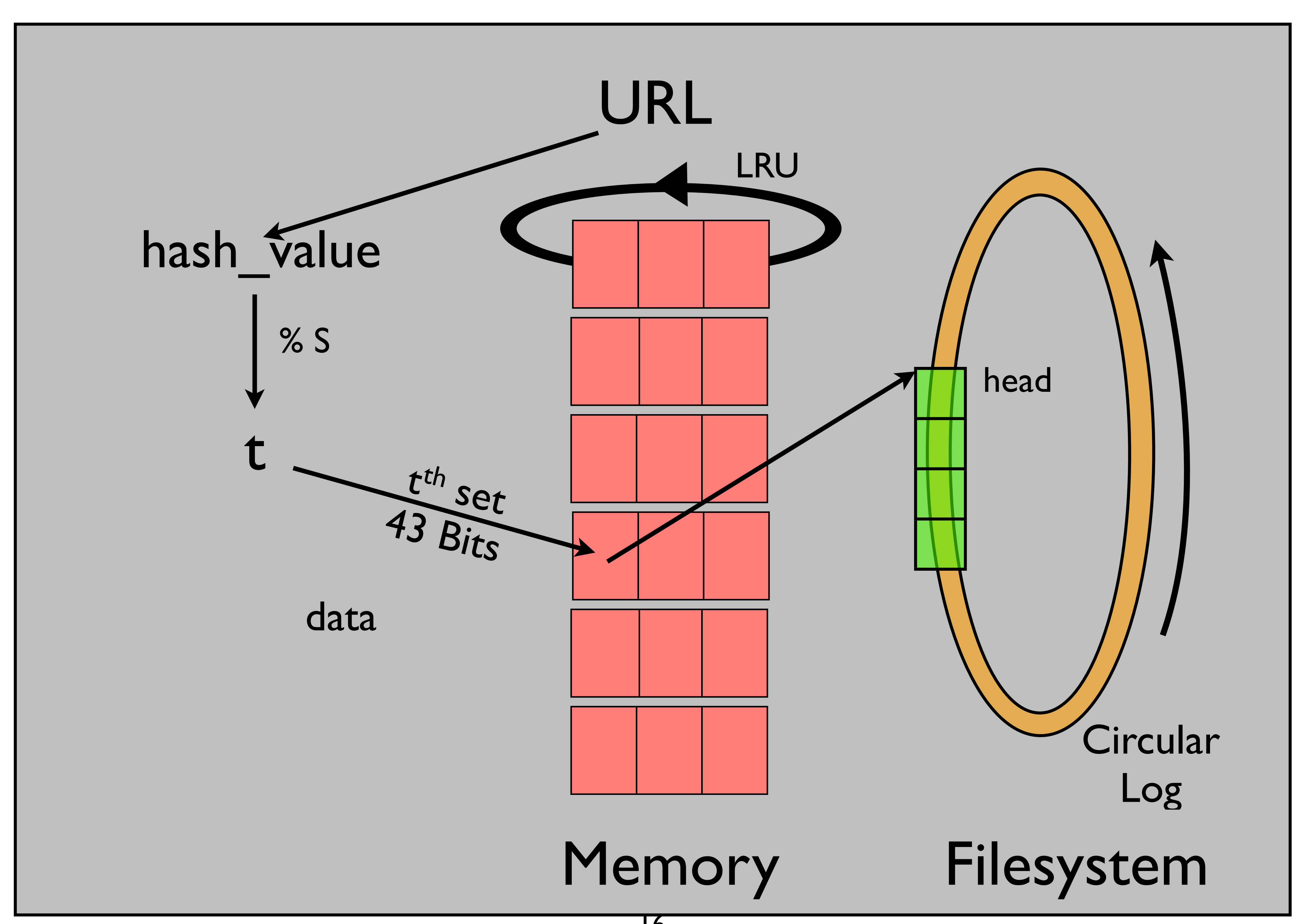
HashCache: Log Policy



HashCache: Log Policy



HashCache: Log Policy



Implementation

Implementation

- HashCache Storage Engine with plug-in policies

Implementation

- HashCache Storage Engine with plug-in policies
- HashCache Web proxy using storage engine

Implementation

- HashCache Storage Engine with plug-in policies
- HashCache Web proxy using storage engine
- Multiple apps on same box, sharing memory

Implementation

- HashCache Storage Engine with plug-in policies
- HashCache Web proxy using storage engine
- Multiple apps on same box, sharing memory
- 20,000 lines C code for the proxy and 1000 lines for the indexing policies

Implementation

- HashCache Storage Engine with plug-in policies
- HashCache Web proxy using storage engine
- Multiple apps on same box, sharing memory
- 20,000 lines C code for the proxy and 1000 lines for the indexing policies
- Event Driven implementation with non-blocking I/O

Implementation

- HashCache Storage Engine with plug-in policies
- HashCache Web proxy using storage engine
- Multiple apps on same box, sharing memory
- 20,000 lines C code for the proxy and 1000 lines for the indexing policies
- Event Driven implementation with non-blocking I/O
- Design similar to that of Flash Web Server. Helpers for I/O and DNS lookups

Implementation

- HashCache Storage Engine with plug-in policies
- HashCache Web proxy using storage engine
- Multiple apps on same box, sharing memory
- 20,000 lines C code for the proxy and 1000 lines for the indexing policies
- Event Driven implementation with non-blocking I/O
- Design similar to that of Flash Web Server. Helpers for I/O and DNS lookups
- Balances load across multiple disks easily and makes scaling obvious

Evaluation - Web Polygraph

- De-facto feature and performance testing tool for web proxies
- Compare all variants of HashCache with Squid and Tiger

Evaluation - Web Polygraph

- De-facto feature and performance testing tool for web proxies
- Compare all variants of HashCache with Squid and Tiger

Experiment Name	Setting	Configuration	Comparision

Evaluation - Web Polygraph

- De-facto feature and performance testing tool for web proxies
- Compare all variants of HashCache with Squid and Tiger

Experiment Name	Setting	Configuration	Comparision
Low End	Small School using Laptop	1.4 GHz 256 MB 60 GB SATA	HashCache vs Squid vs Tiger

Evaluation - Web Polygraph

- De-facto feature and performance testing tool for web proxies
- Compare all variants of HashCache with Squid and Tiger

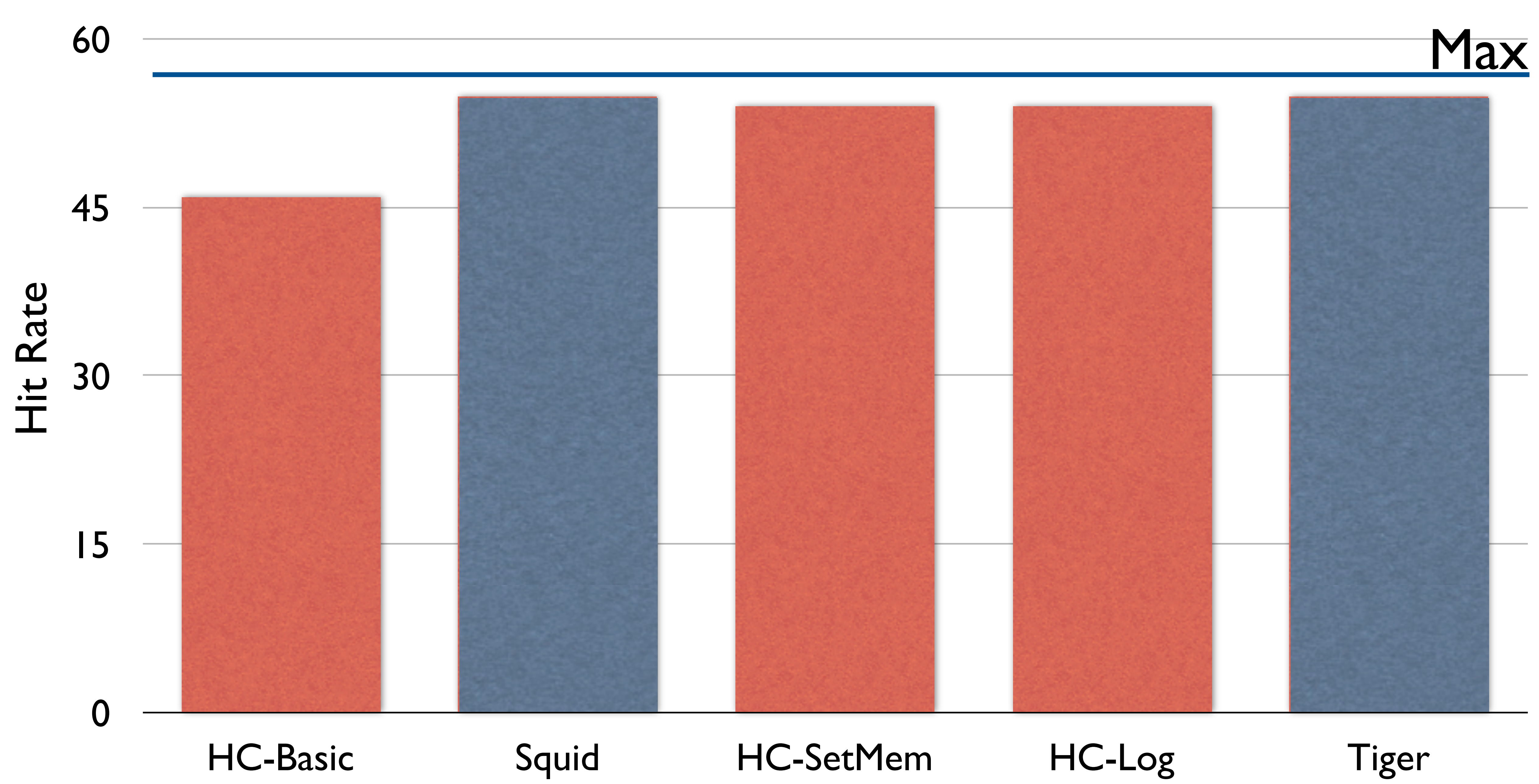
Experiment Name	Setting	Configuration	Comparision
Low End	Small School using Laptop	1.4 GHz 256 MB 60 GB SATA	HashCache vs Squid vs Tiger
High End	ISP with High-End Server	2 GHz 3.5 GB 5x18 GB SCSI	HashCache-Log vs Squid vs Tiger

Evaluation - Web Polygraph

- De-facto feature and performance testing tool for web proxies
- Compare all variants of HashCache with Squid and Tiger

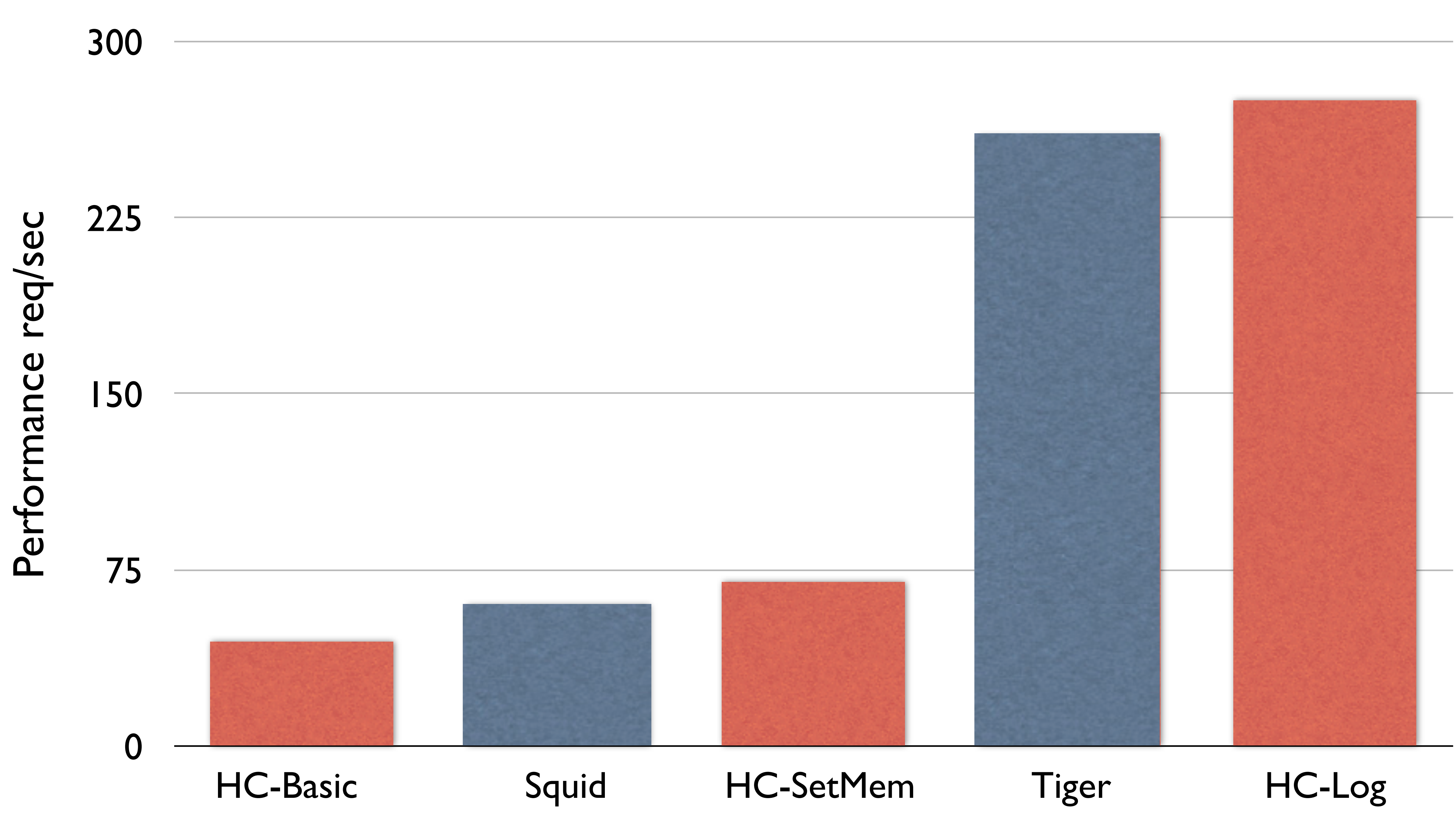
Experiment Name	Setting	Configuration	Comparision
Low End	Small School using Laptop	1.4 GHz 256 MB 60 GB SATA	HashCache vs Squid vs Tiger
High End	ISP with High-End Server	2 GHz 3.5 GB 5x18 GB SCSI	HashCache-Log vs Squid vs Tiger
Large Disk	Large School with Mini-Tower	1.4 GHz 2 GB 2x1TB USB	HashCache-Log vs HashCache-SetMem

Hit Rate Comparison

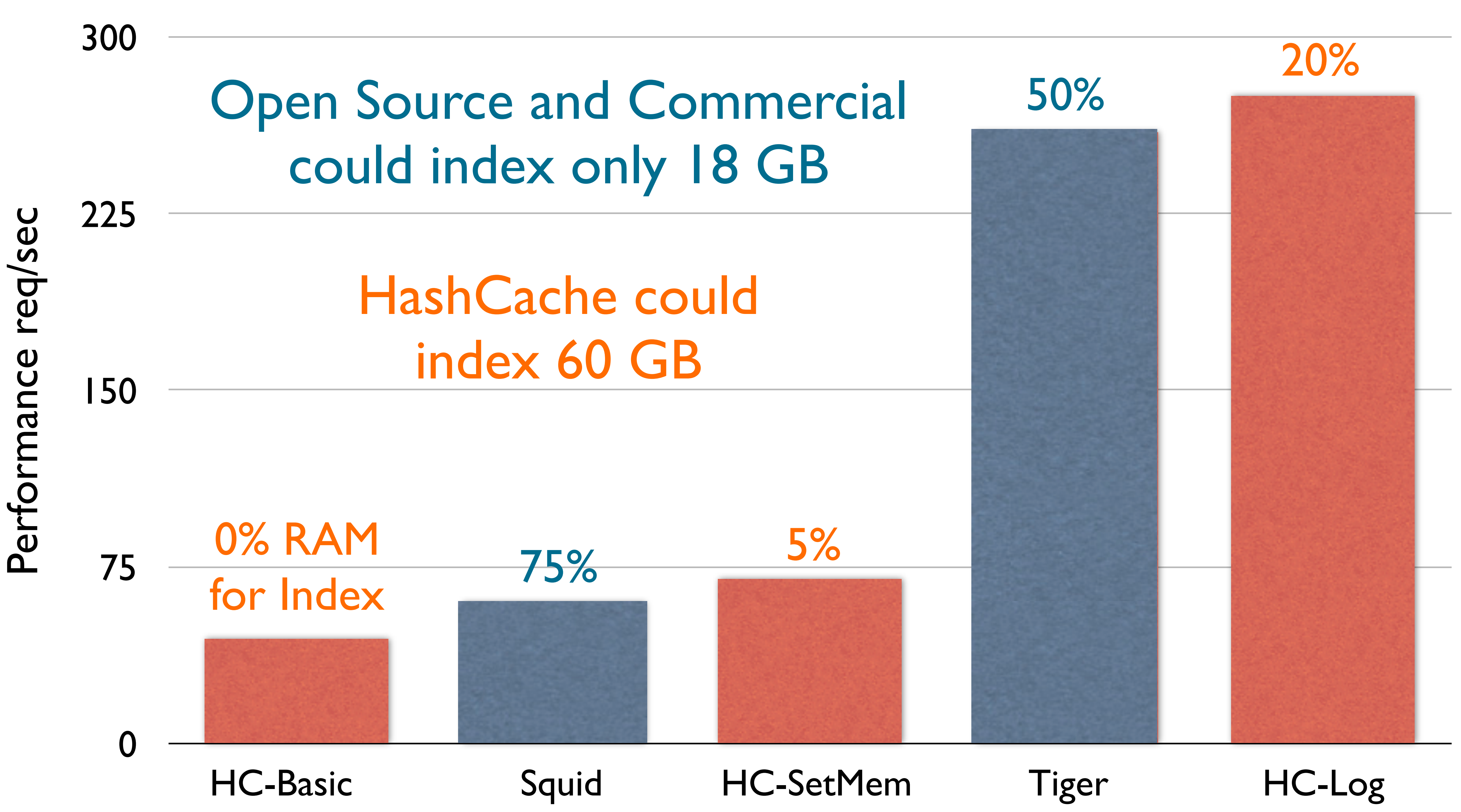


Low End Configuration

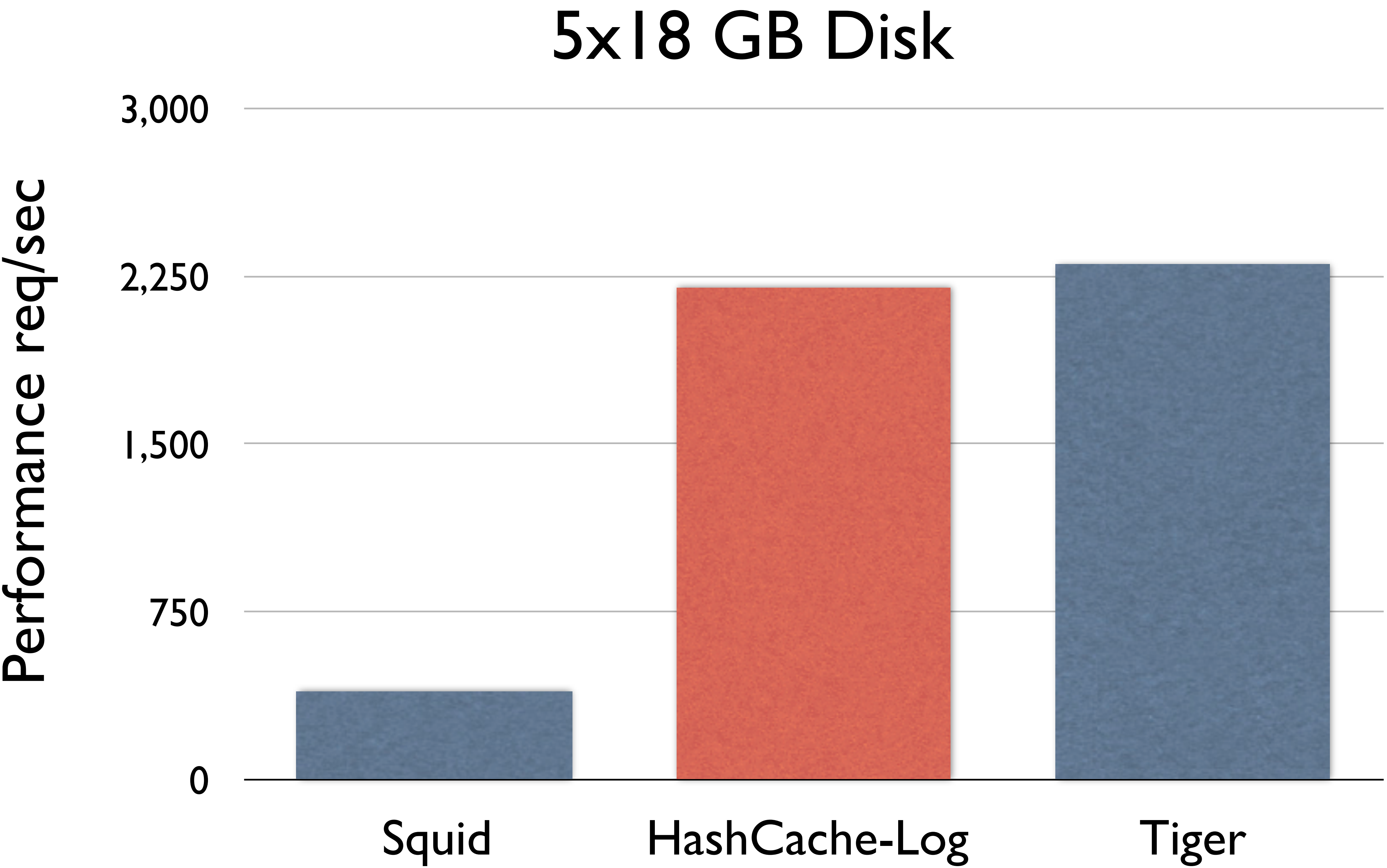
Low End Configuration



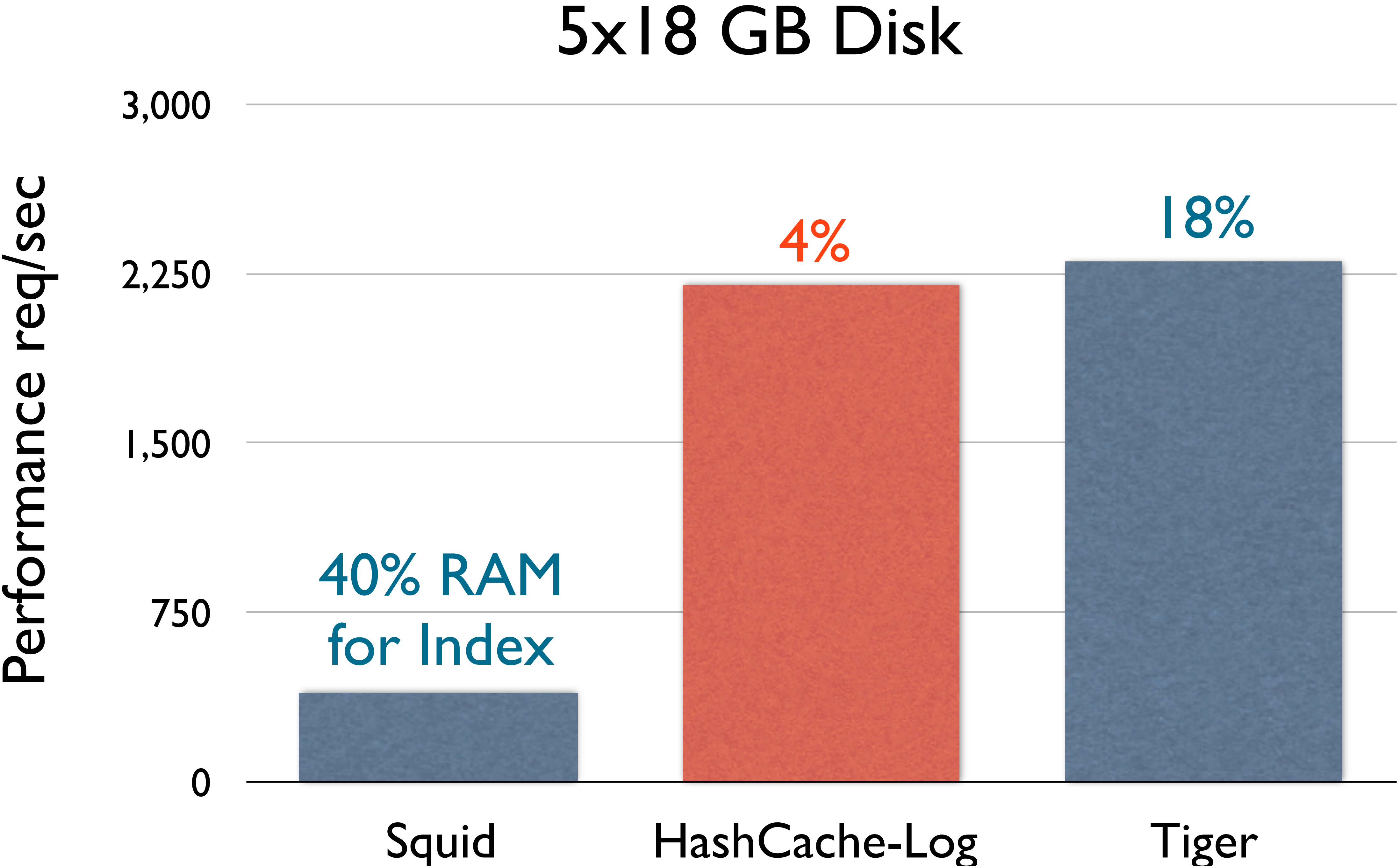
Low End Configuration



High End Configuration

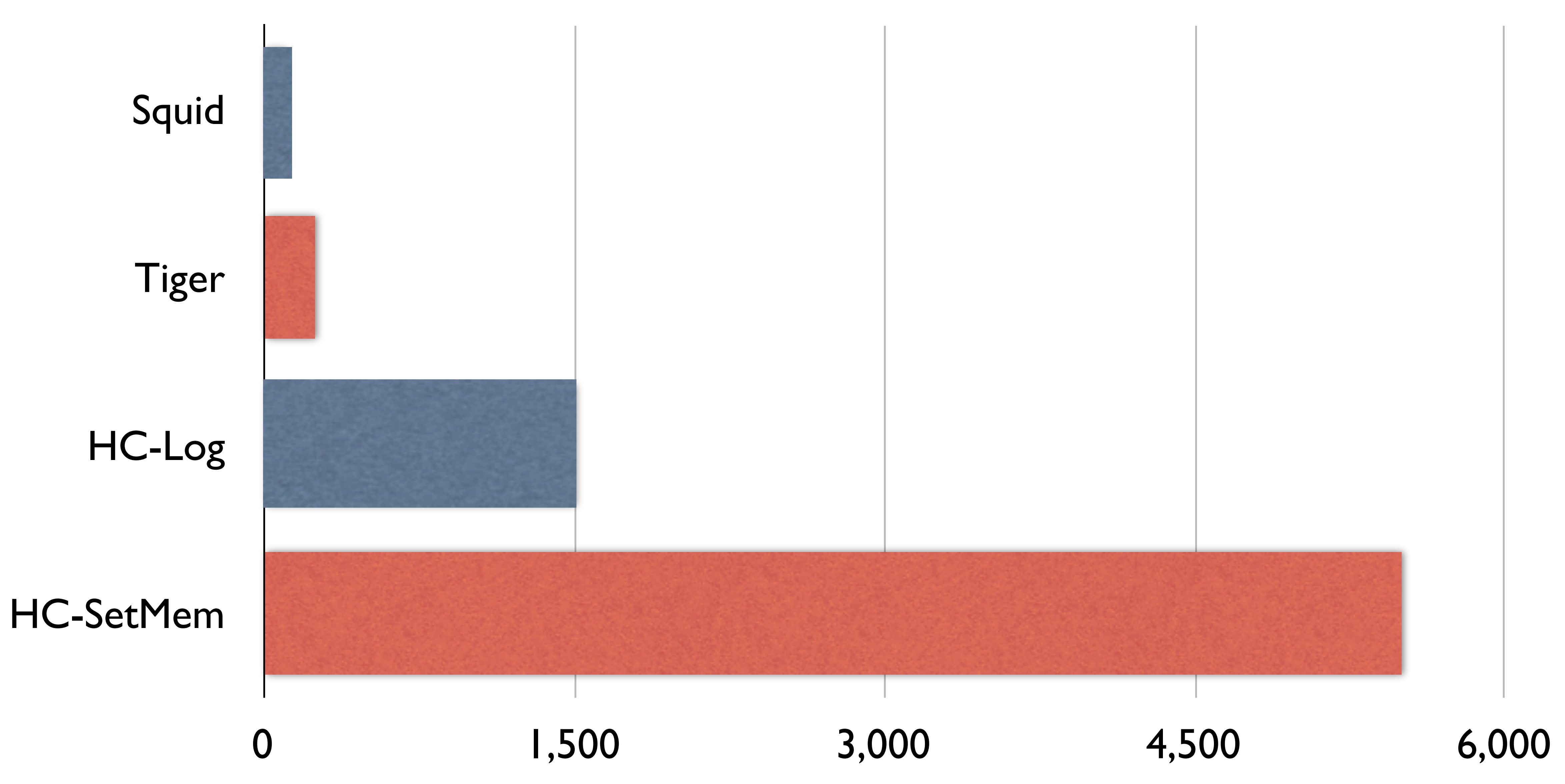


High End Configuration



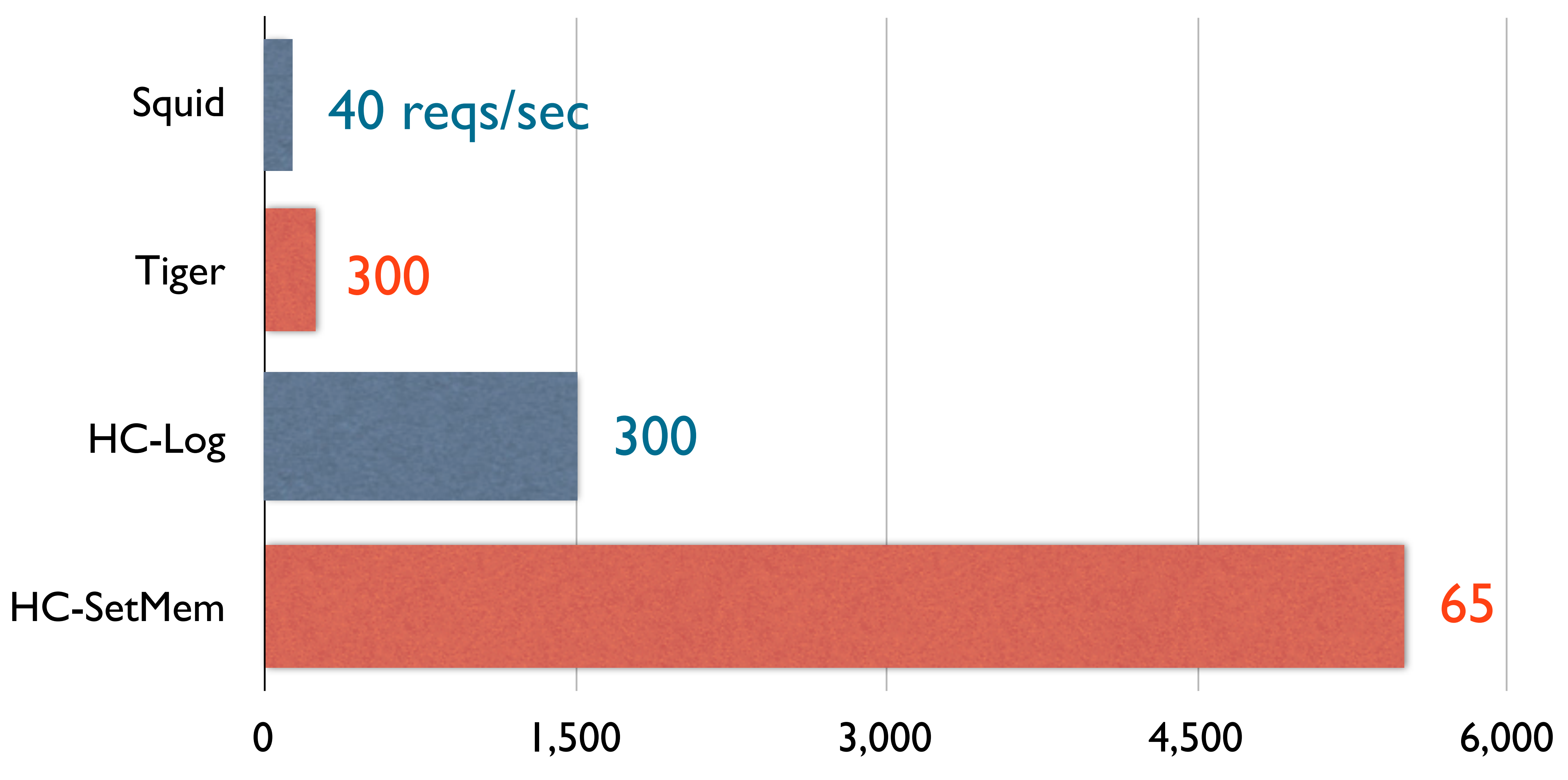
Index Efficiency

Index Efficiency



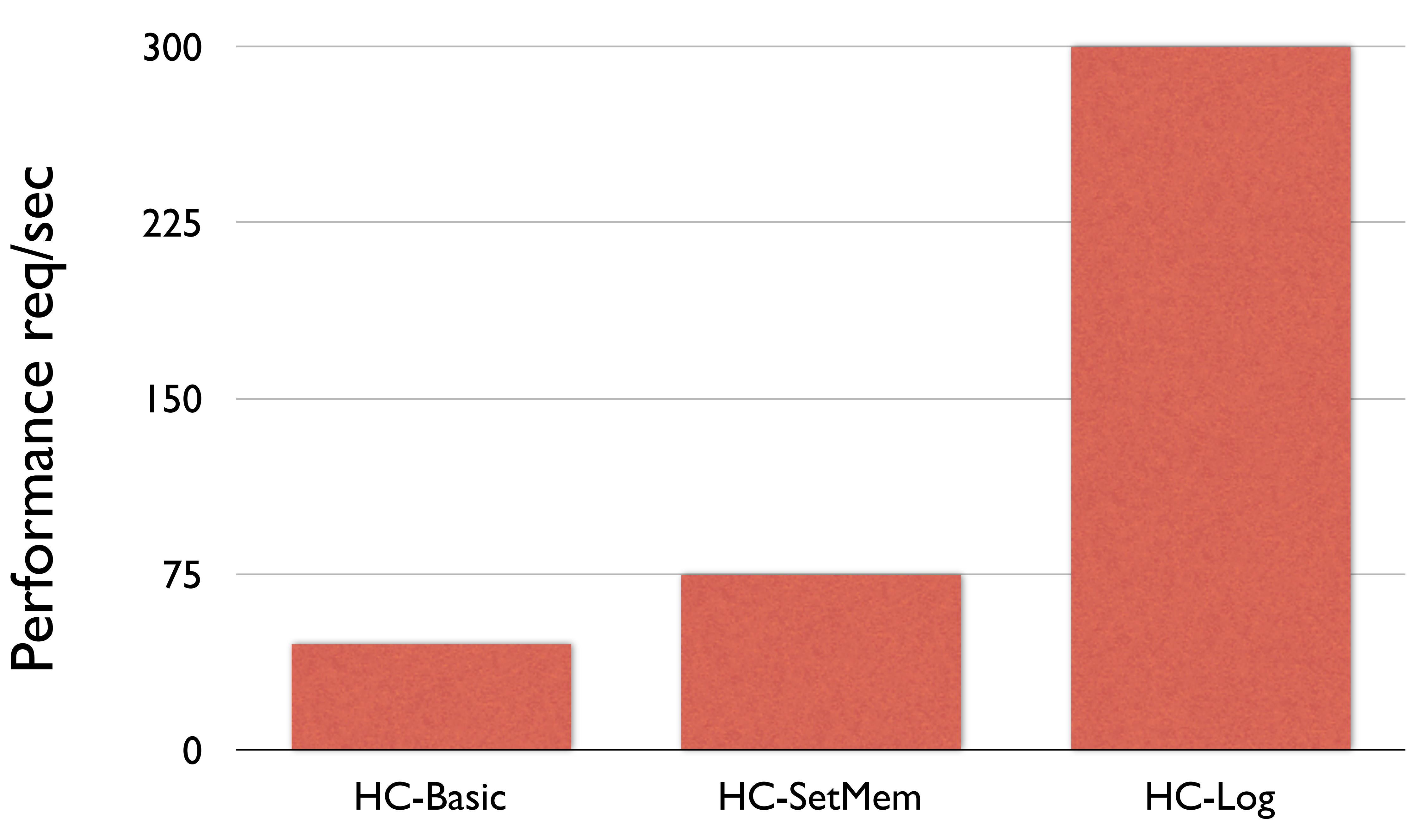
Max Disk for 1GB RAM

Index Efficiency

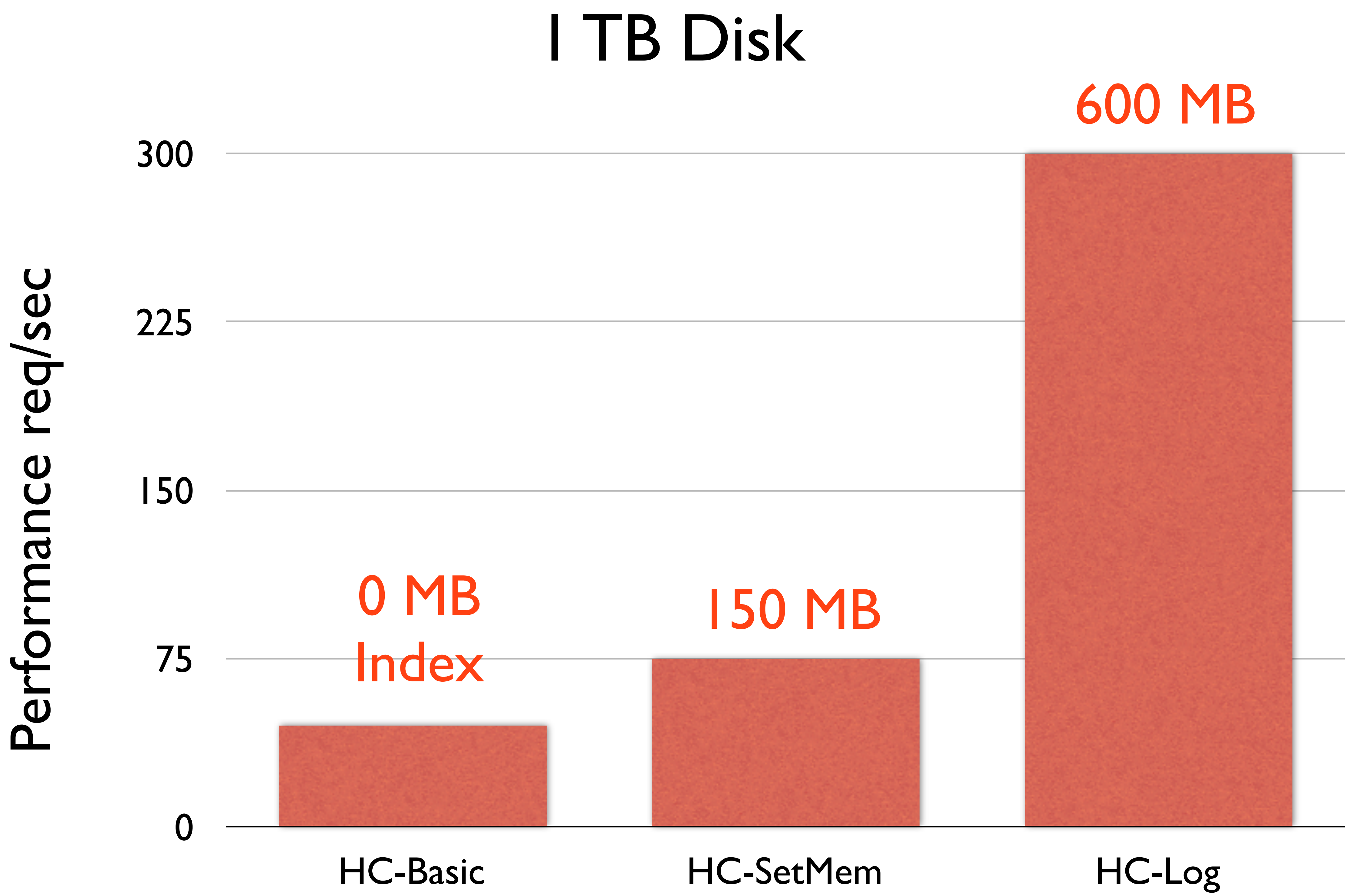


Max Disk for 1GB RAM

Large Disk Configuration



Large Disk Configuration



Conclusions & Status

- New Storage Engine & Web Cache
  - From no RAM per object to tiny no. of bits/obj
  - 6-10x better than Tiger, 20-50x vs Squid
  - Enables large disk w/ only laptop-class machine
  - More policies, details in paper
- Suitable for developing-world usage
  - Current deployments: Ghana, Nigeria
  - Working w/ school supplier on new deployments

Thank You!

abadam@cs.princeton.edu

http://tinyurl.com/hashcache