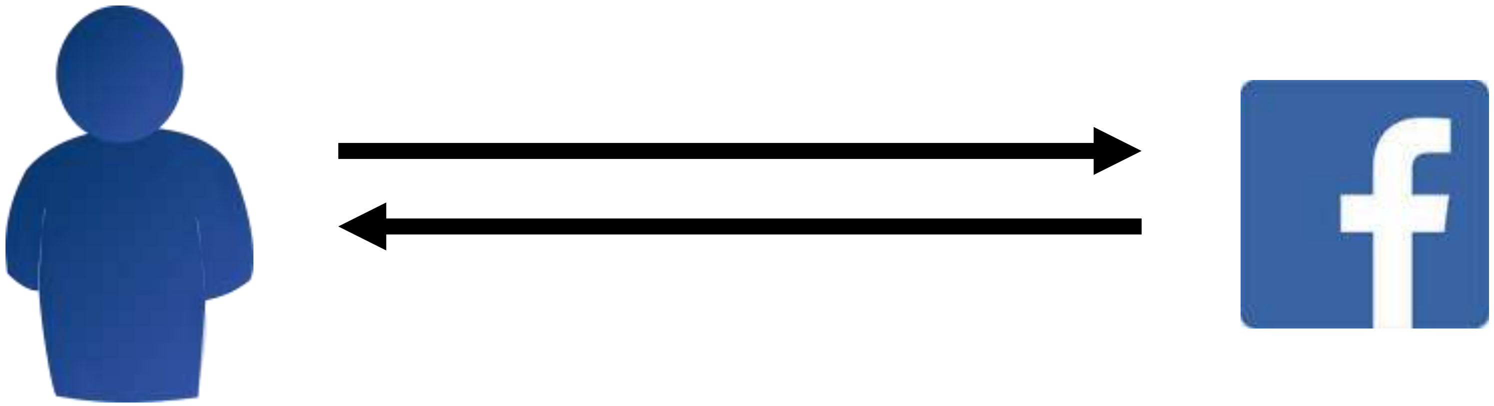
Serving Photos at *Scaaale*: Caching and Storage

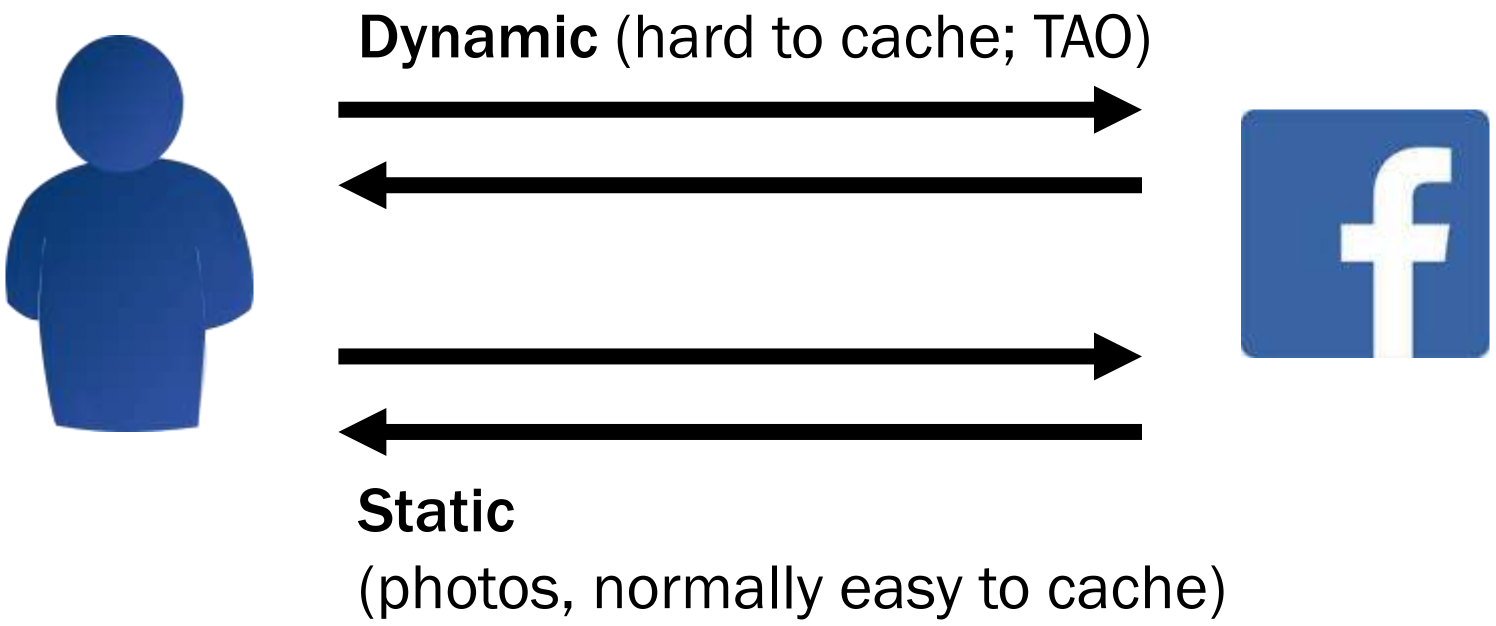
An Analysis of Facebook Photo Caching. Huang et al.

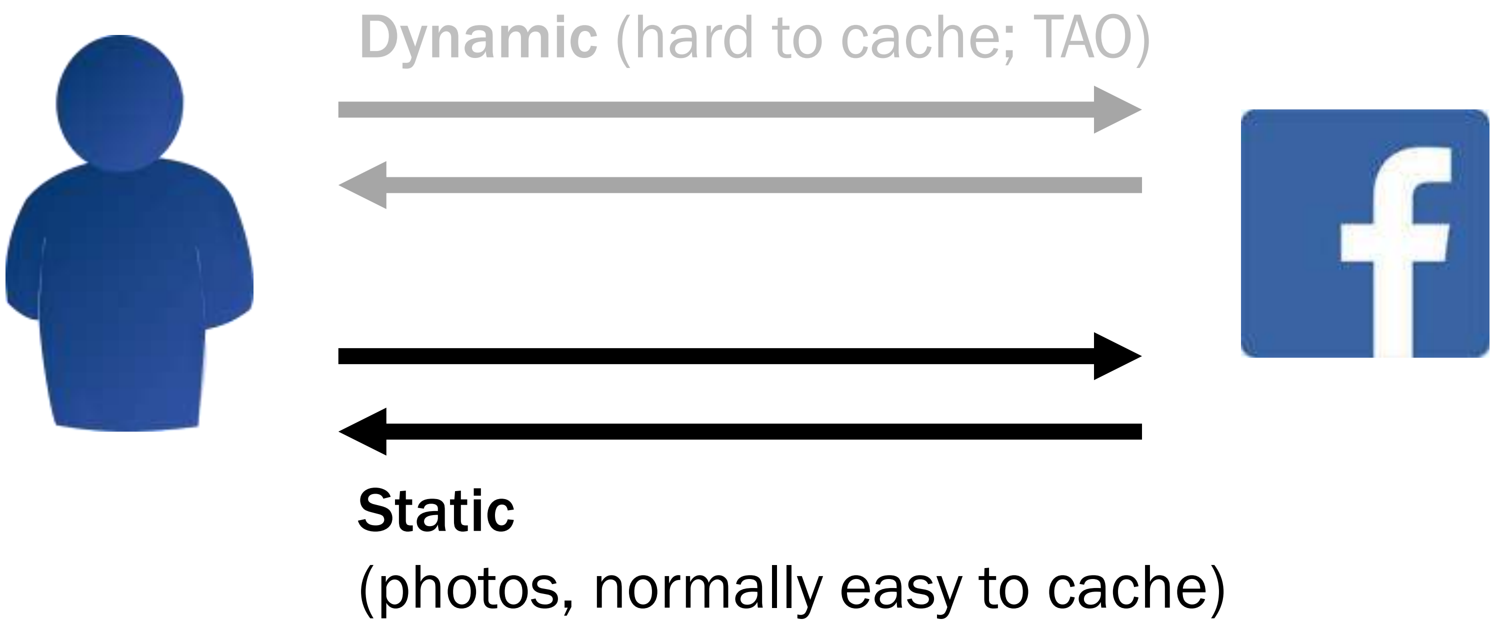
Finding a Needle in a Haystack. Beaver et al.

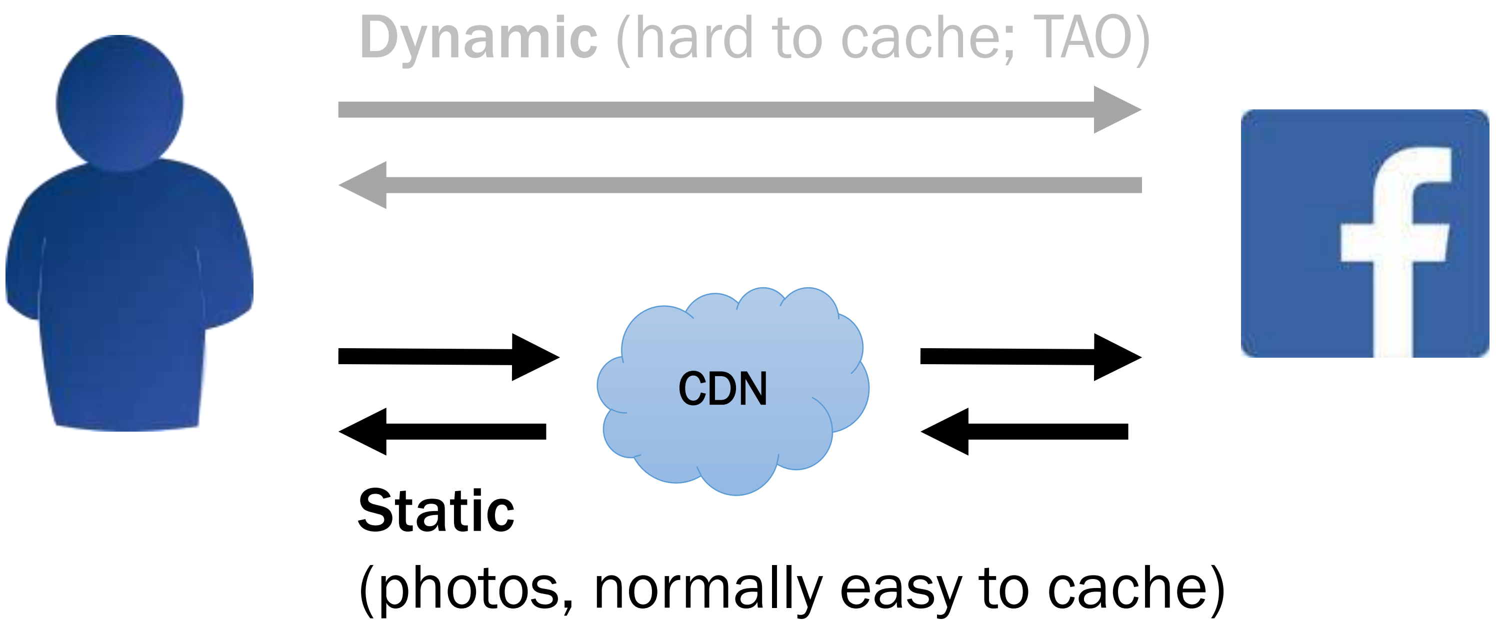
Vlad Niculae for CS6410

Most slides from Qi Huang (SOSP 2013) and Peter Vajgel (OSDI 2010)



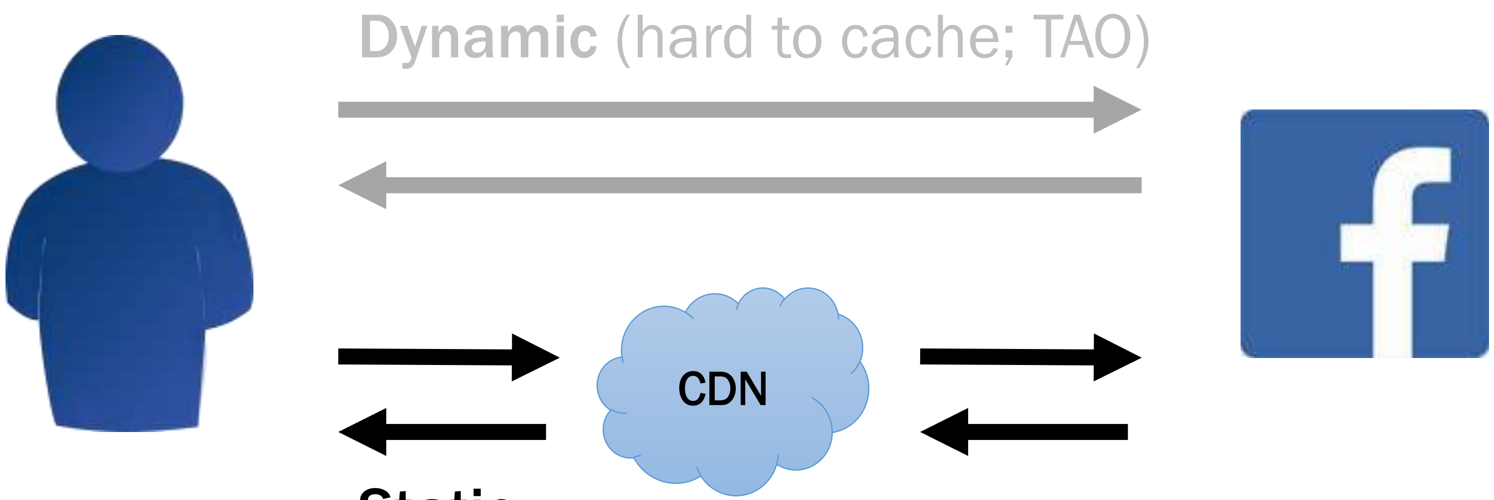

Dynamic (hard to cache; TAO)
Static
(photos, normally easy to cache)


Dynamic (hard to cache; TAO)
Static
(photos, normally easy to cache)


Dynamic (hard to cache; TAO)
CDN
Static
(photos, normally easy to cache)

An Analysis of Facebook Photo Caching

Qi Huang, Ken Birman, Robbert van Renesse (Cornell),
Wyatt Lloyd (Princeton, Facebook),
Sanjeev Kumar, Harry C. Li (Facebook)

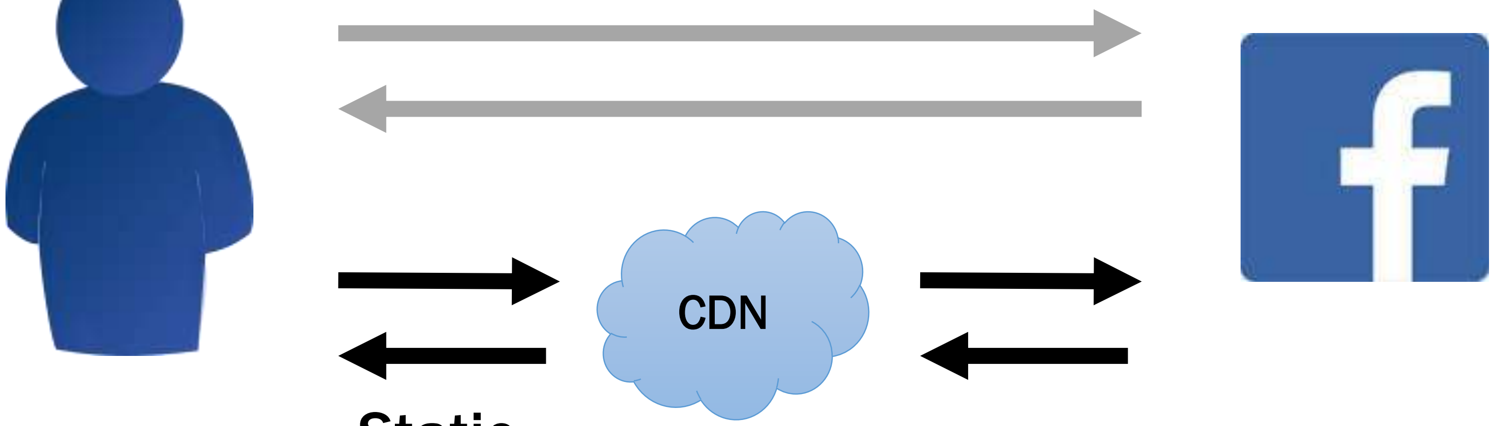


Static

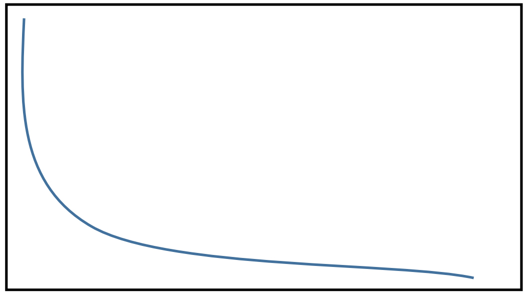
(photos, normally easy to cache)

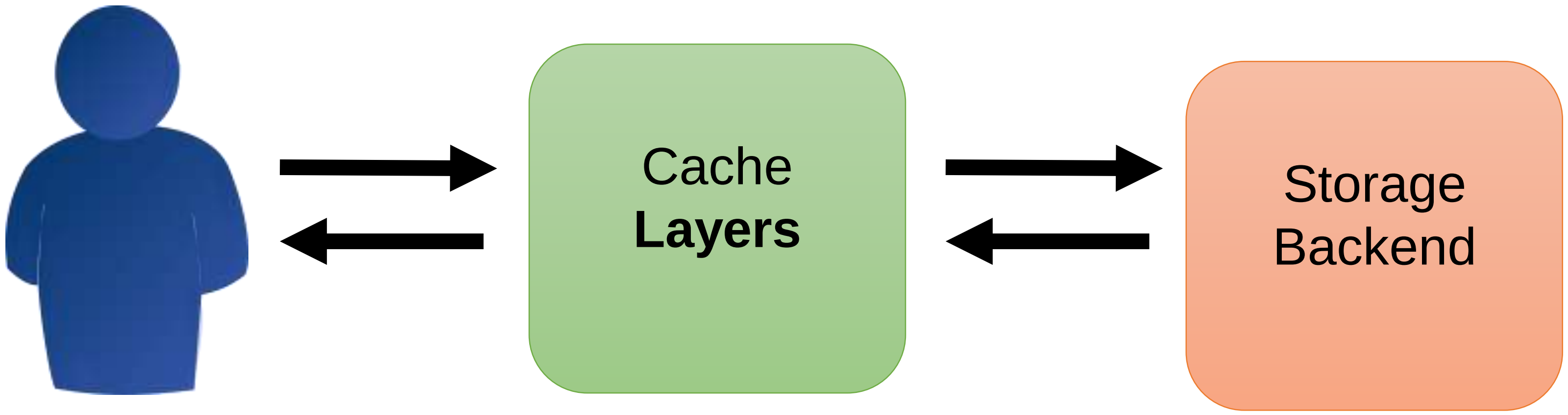
“Normal” site CDN hitrate ~99%

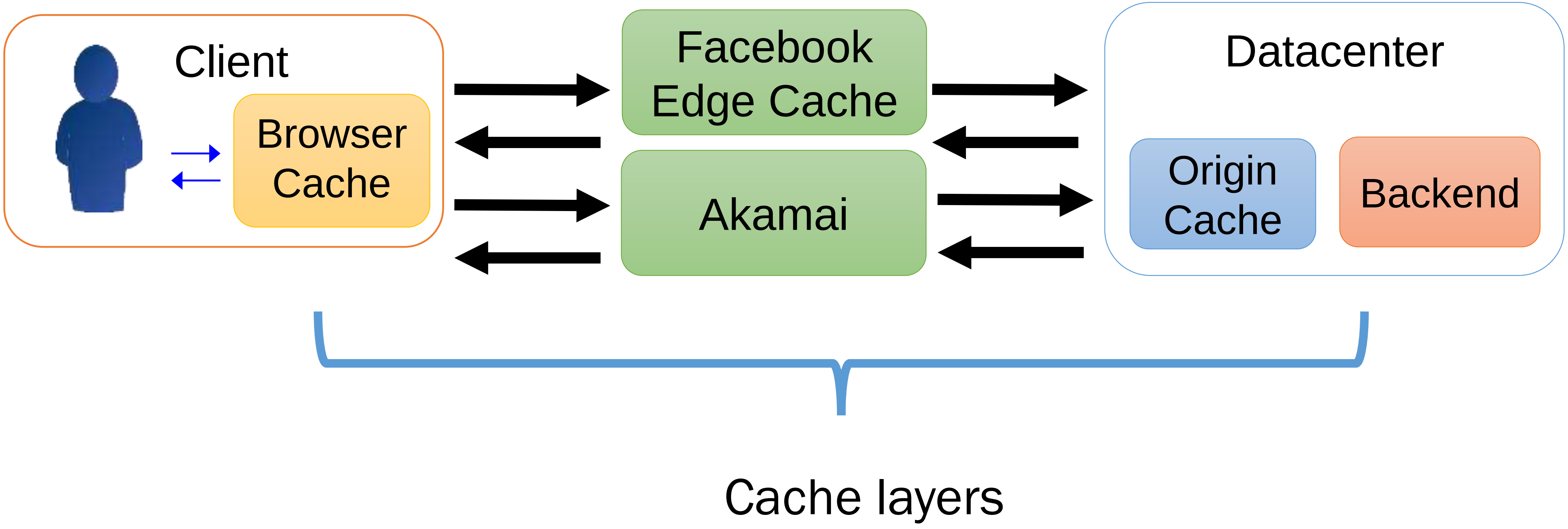
Dynamic (hard to cache; TAO)

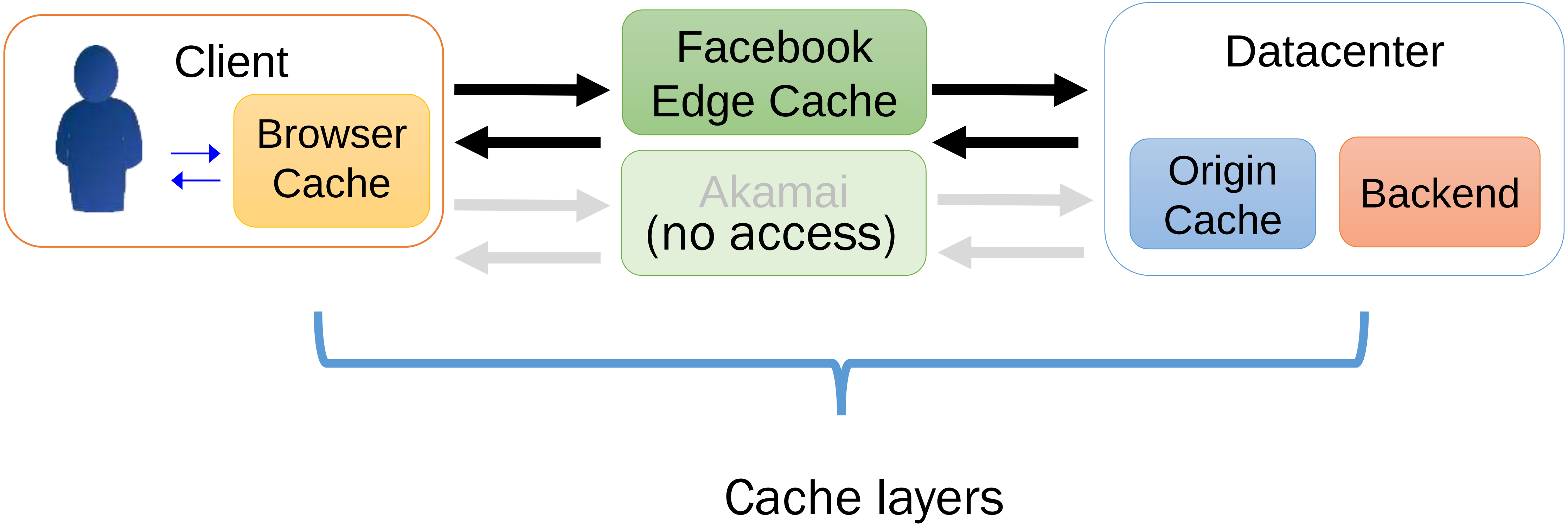


Static
(photos, normally easy to cache)
“Normal” site CDN hitrate ~99%
For Facebook, CDN hitrate ~80%

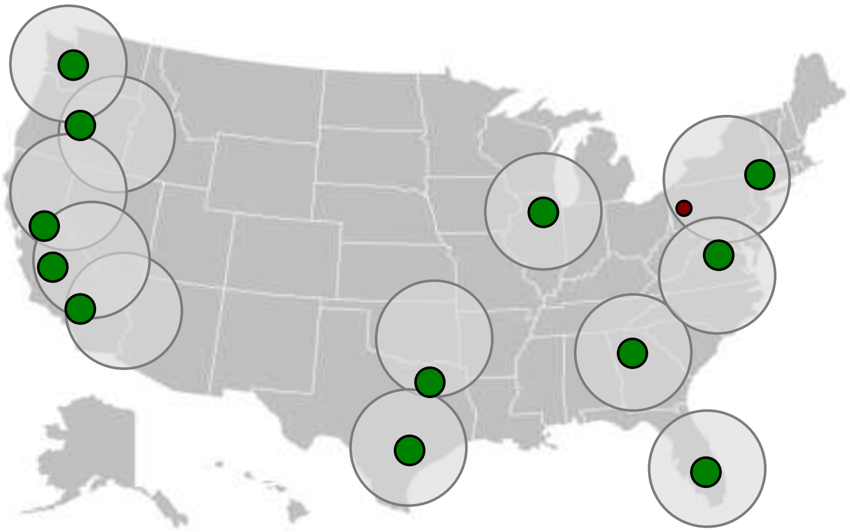


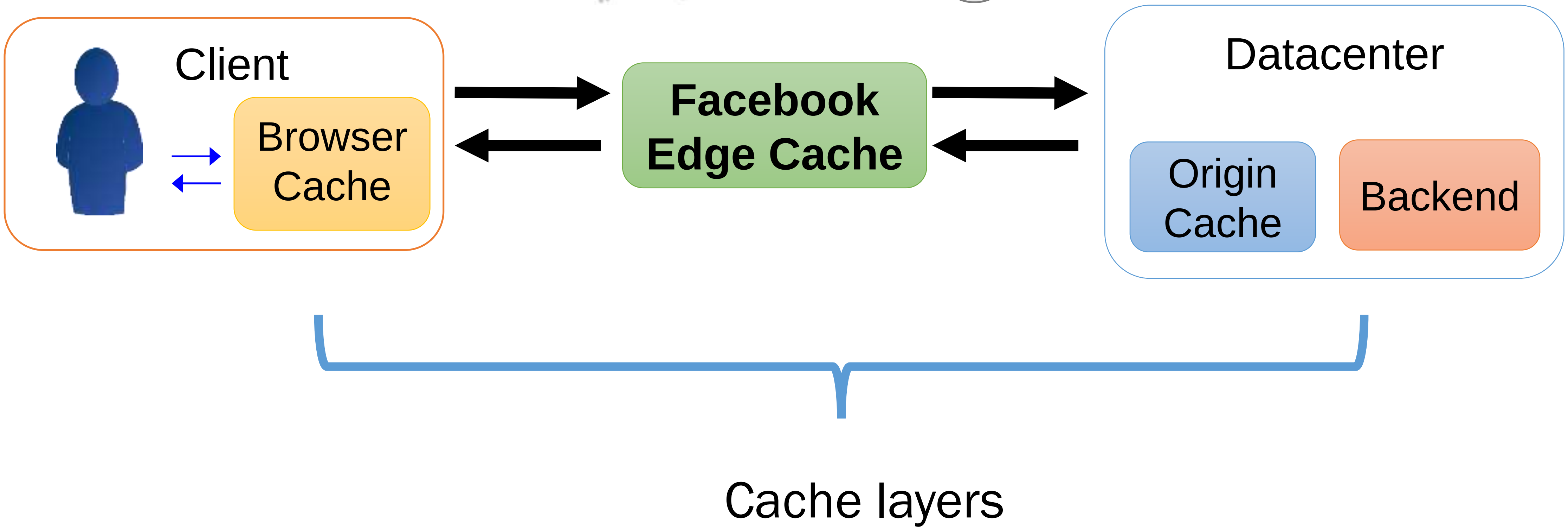

Cache
Layers
Storage
Backend


Client
Browser Cache
Facebook Edge Cache
Akamai
Datacenter
Origin Cache
Backend
Cache layers

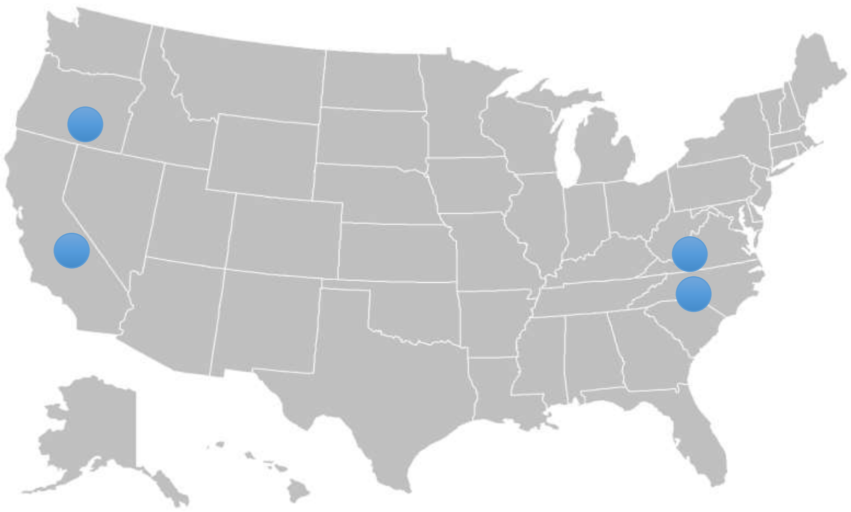

Client
Browser Cache
Facebook Edge Cache
Akamai
(no access)
Datacenter
Origin Cache
Backend
Cache layers

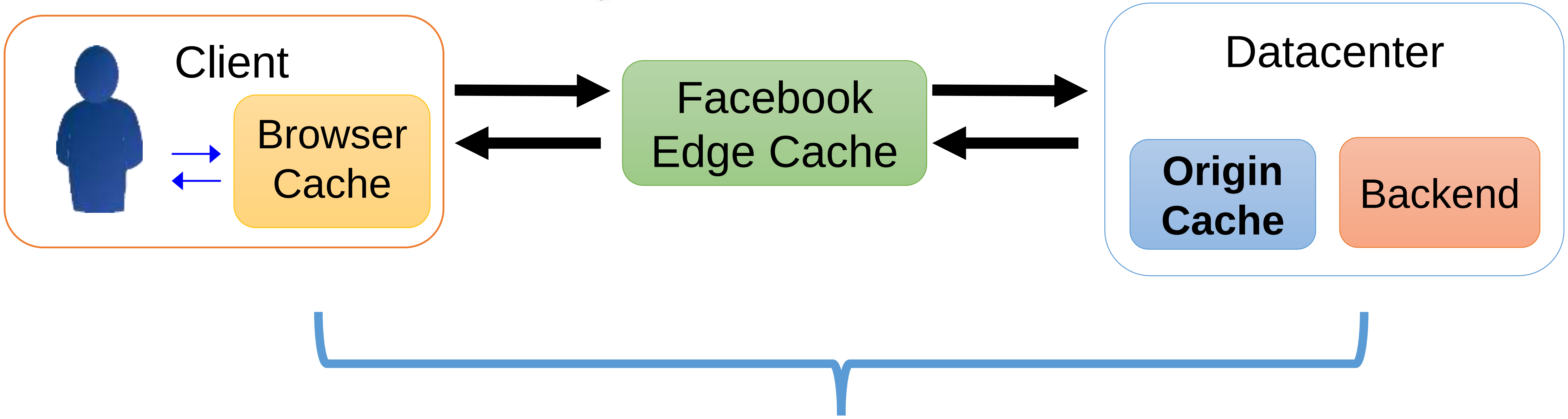
Points of presence:
Independent
FIFO
Main goal:
reduce bandwidth



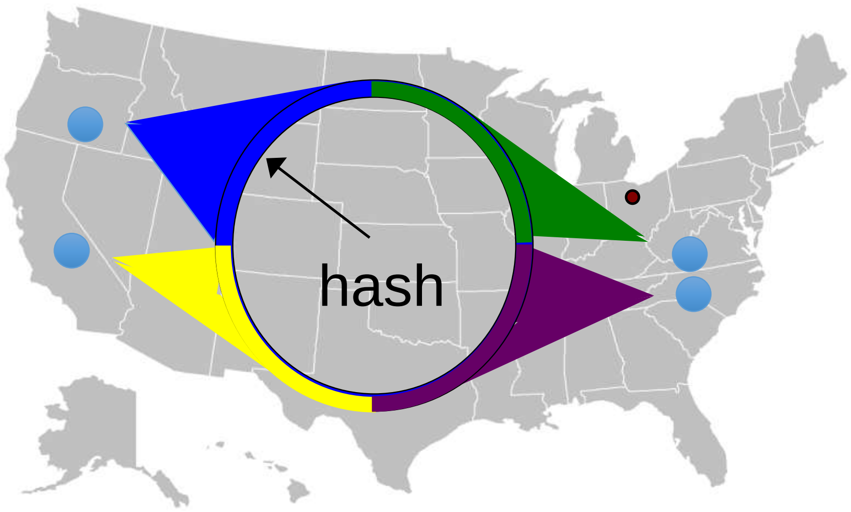


Origin:
Coordinated
FIFO
Main goal:
traffic sheltering





Cache layers



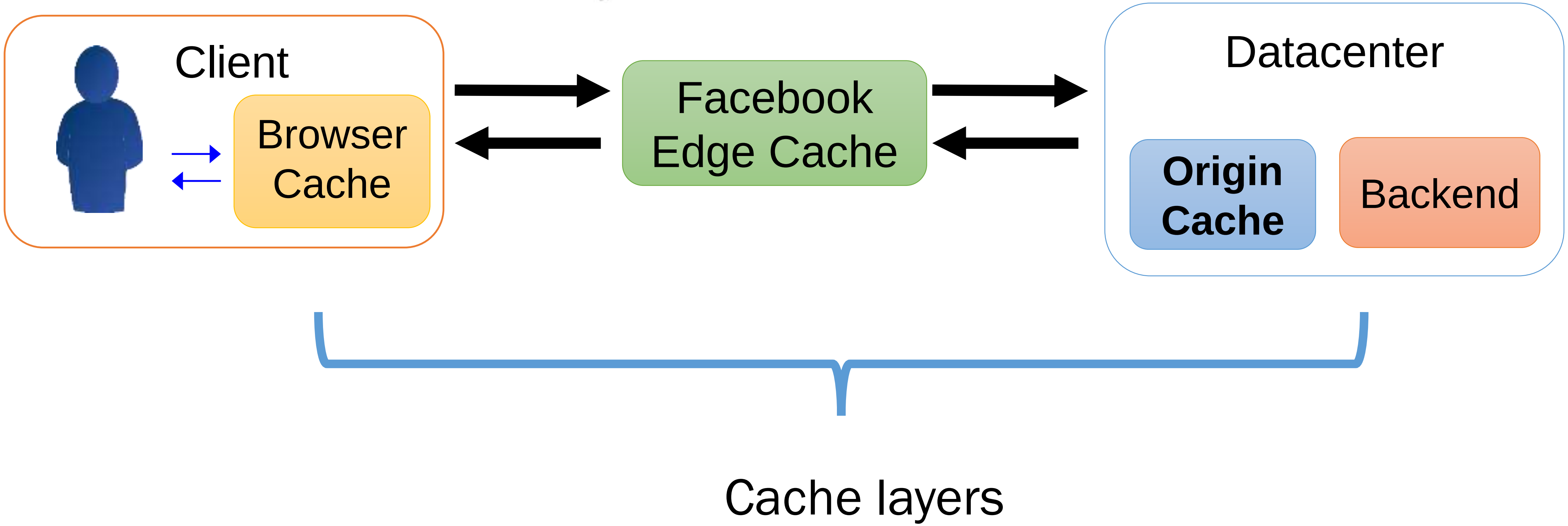
Origin:

Coordinated.

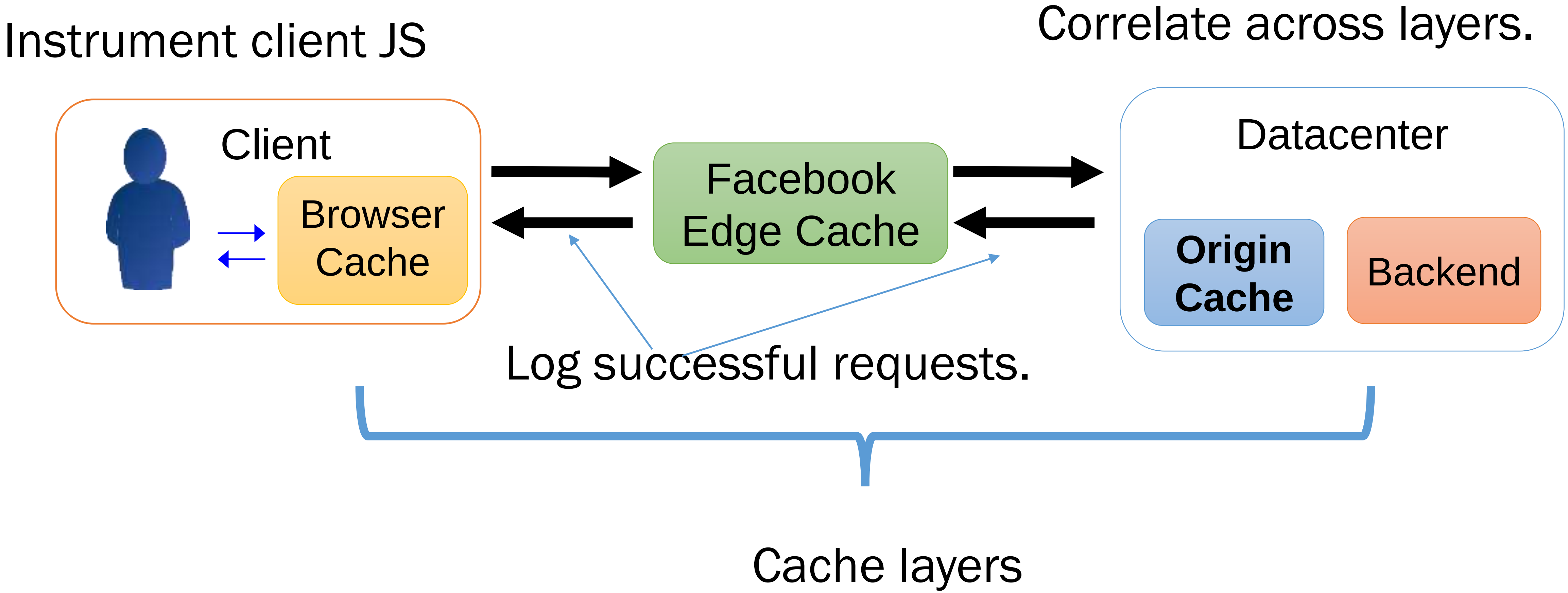
FIFO

Main goal:

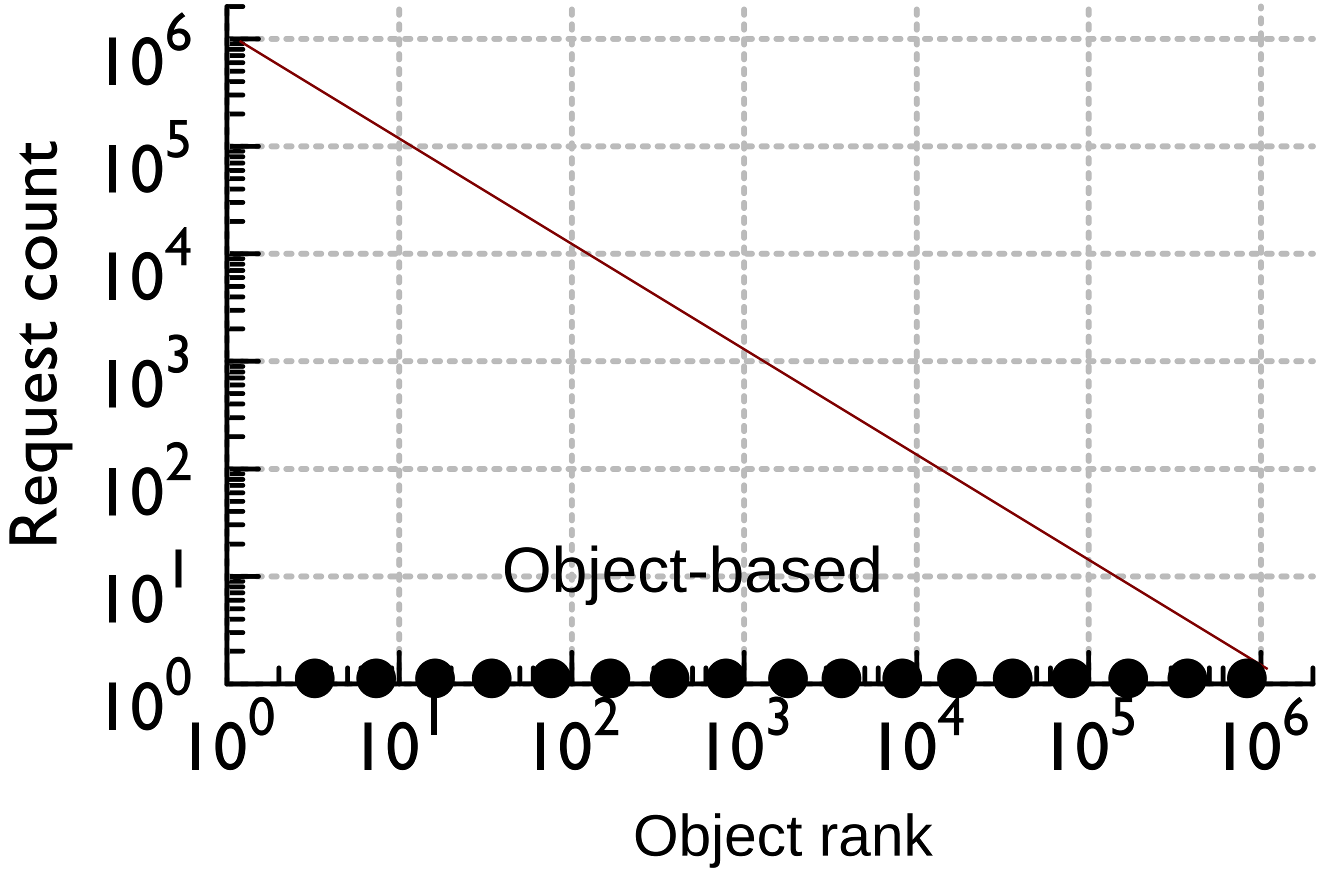
traffic sheltering



Analyze traffic in production!

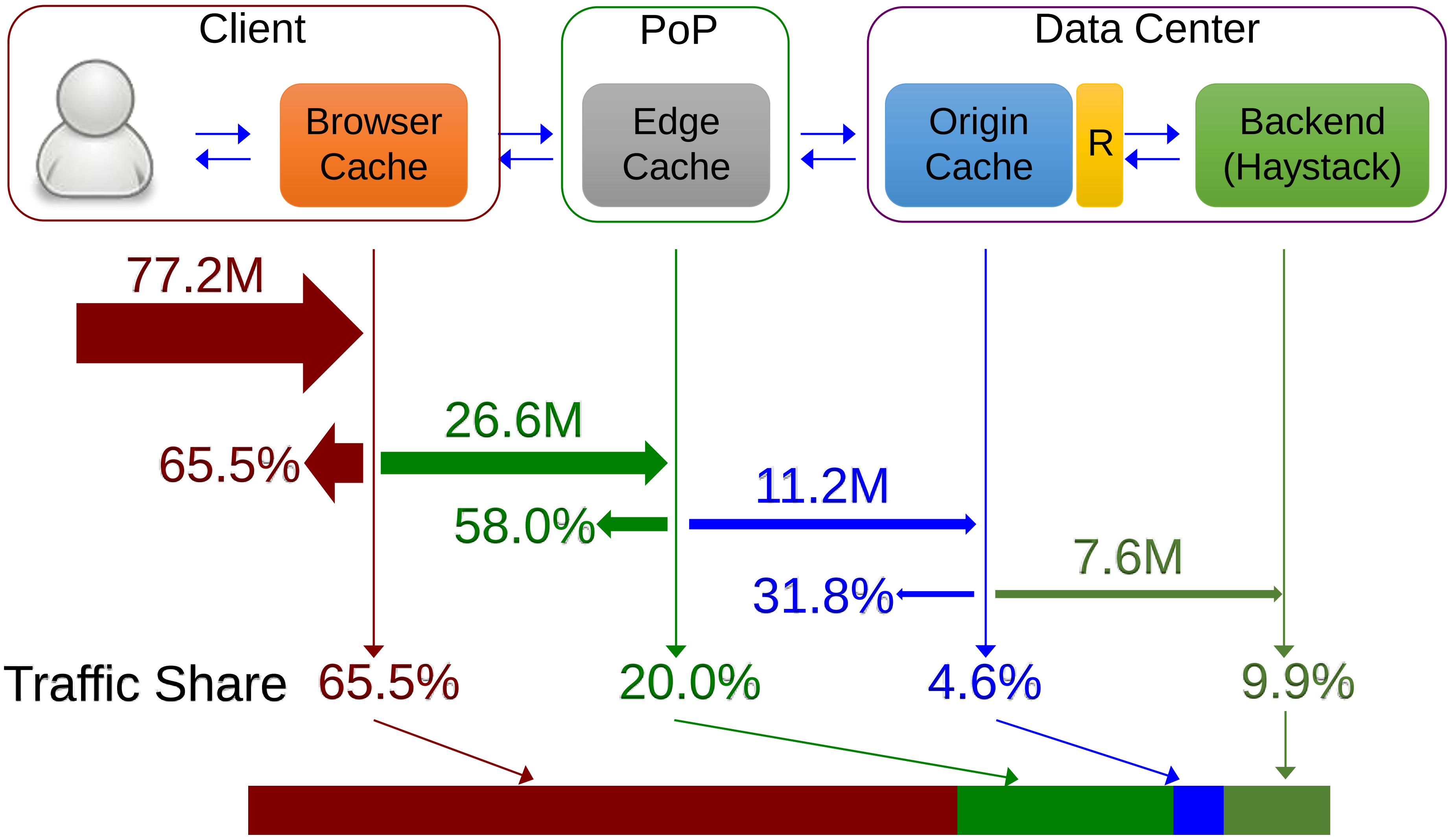


Sampling on Power-law

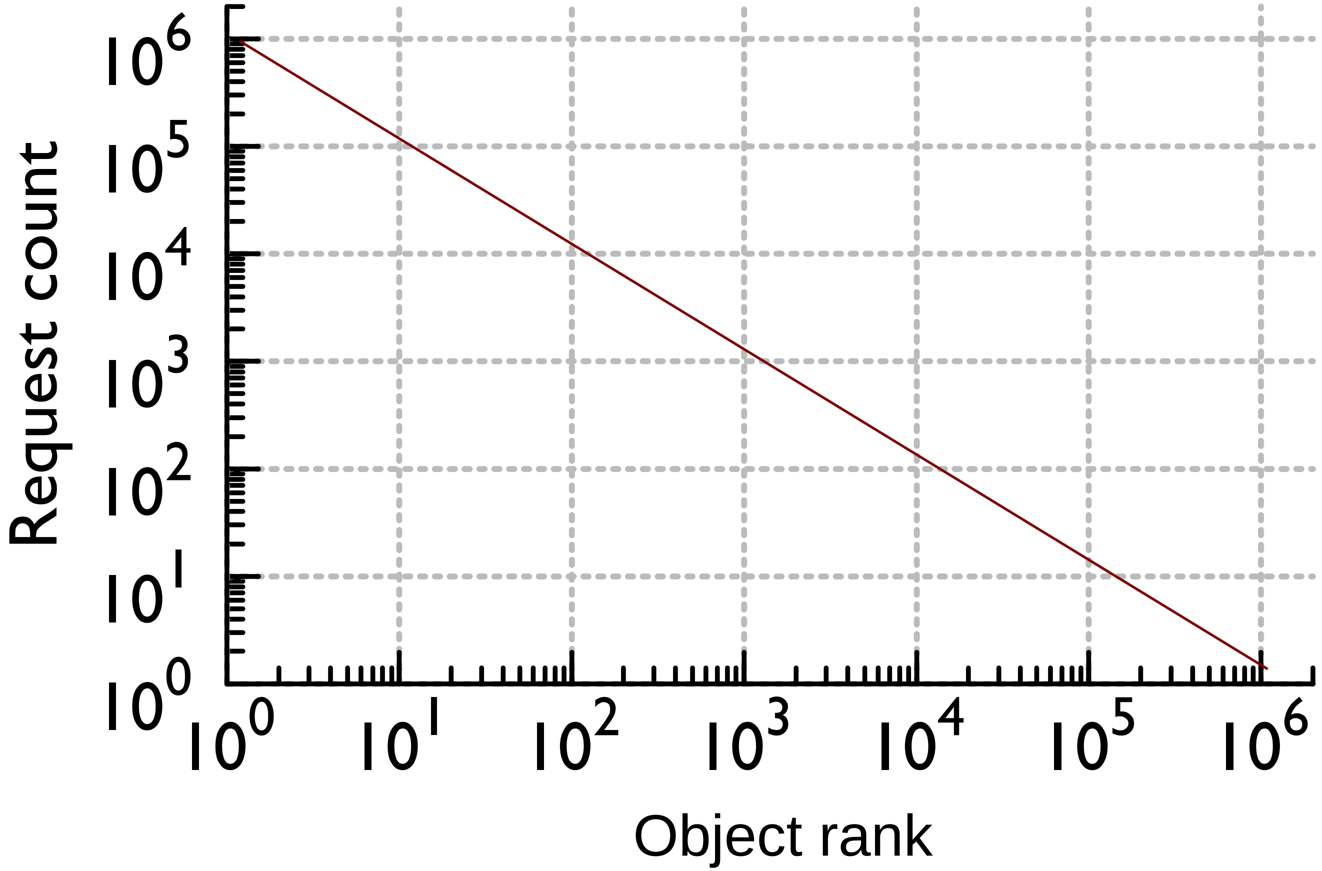


- Object-based: fair coverage of unpopular content
- Sample 1.4M photos, 2.6M photo objects

Data analysis

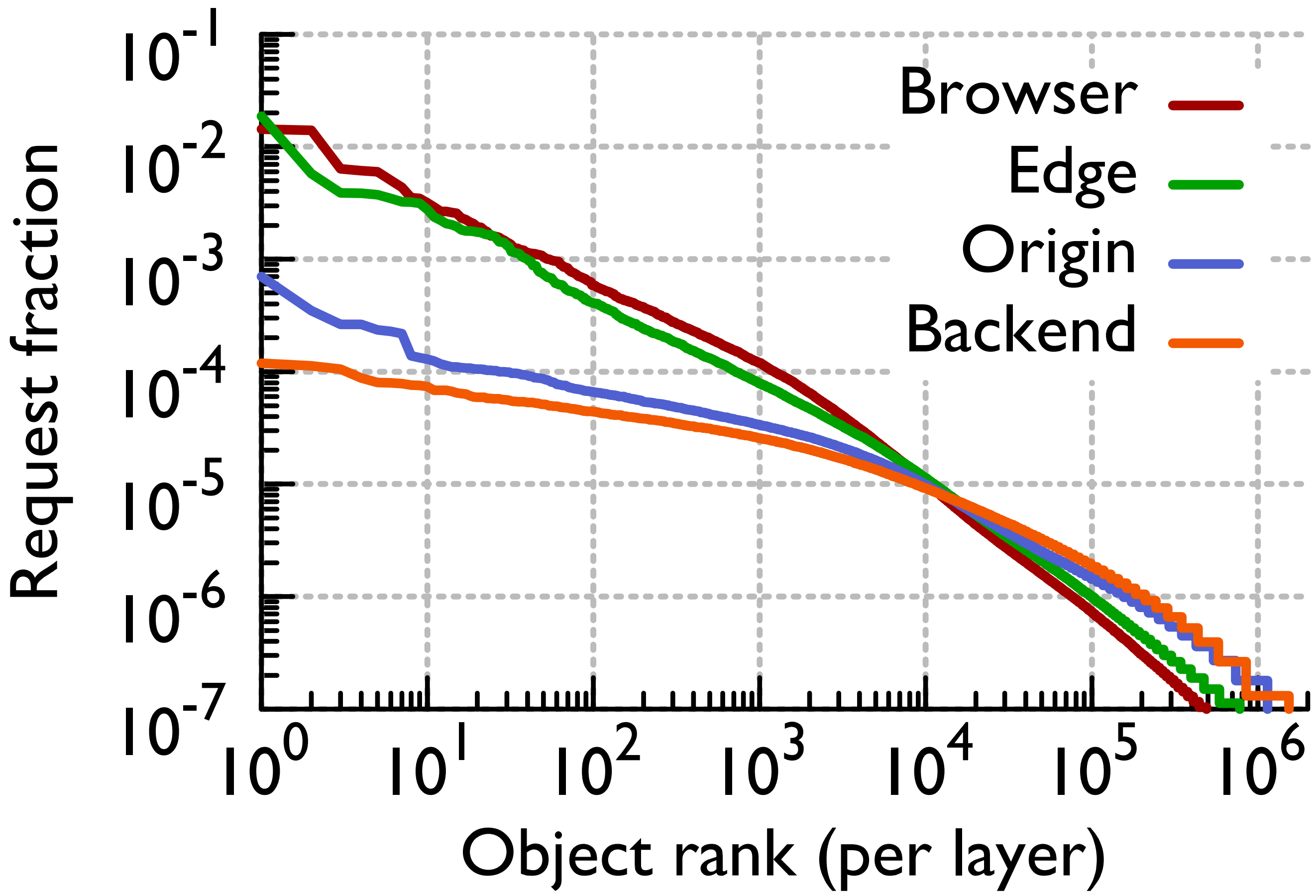


Client
Browser Cache
PoP
Edge Cache
Data Center
Origin Cache
R
Backend (Haystack)
77.2M
65.5%
26.6M
58.0%
11.2M
31.8%
7.6M
Traffic Share
65.5%
20.0%
4.6%
9.9%



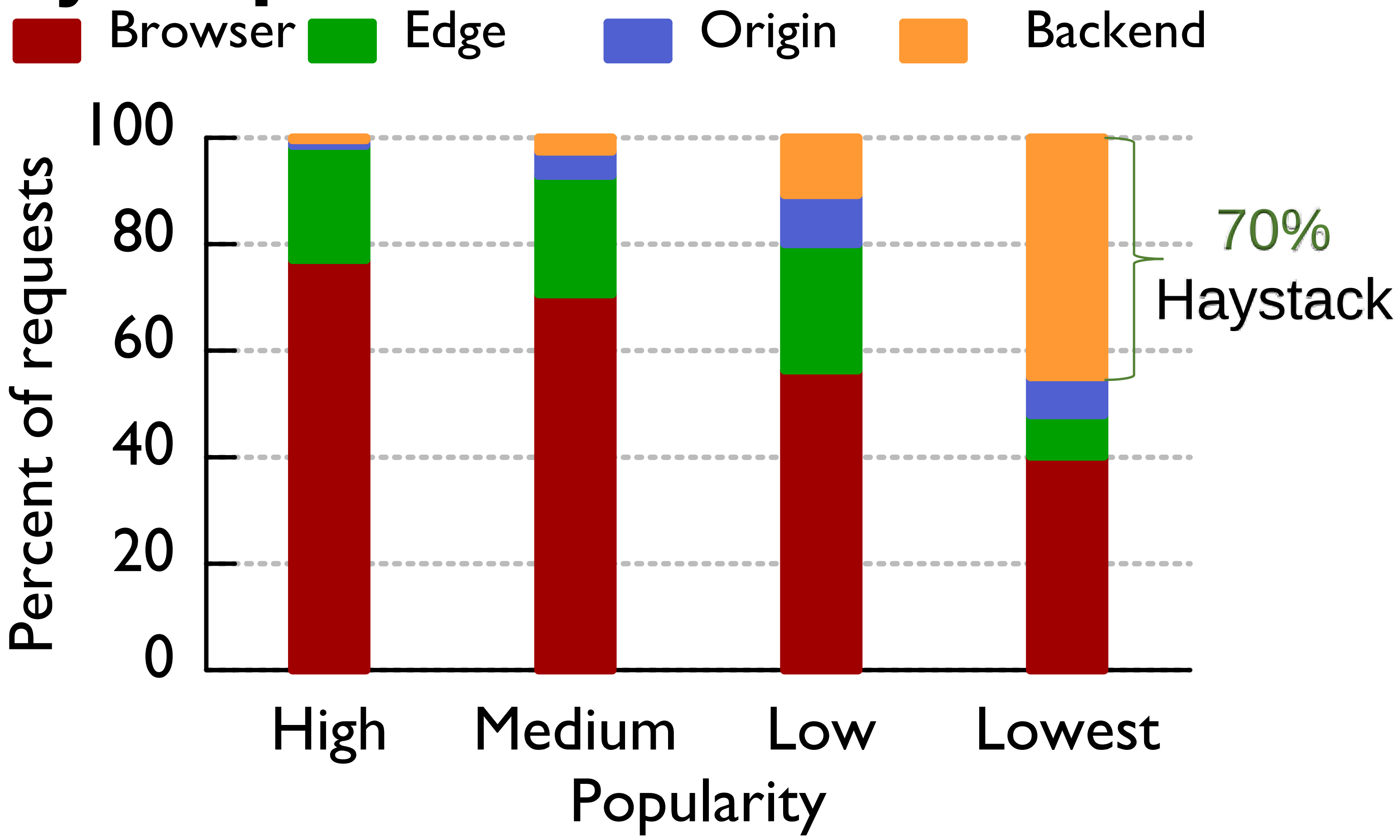
Request count
10^6
10^5
10^4
10^3
10^2
10^1
10^0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
Object rank

Popularity Distribution

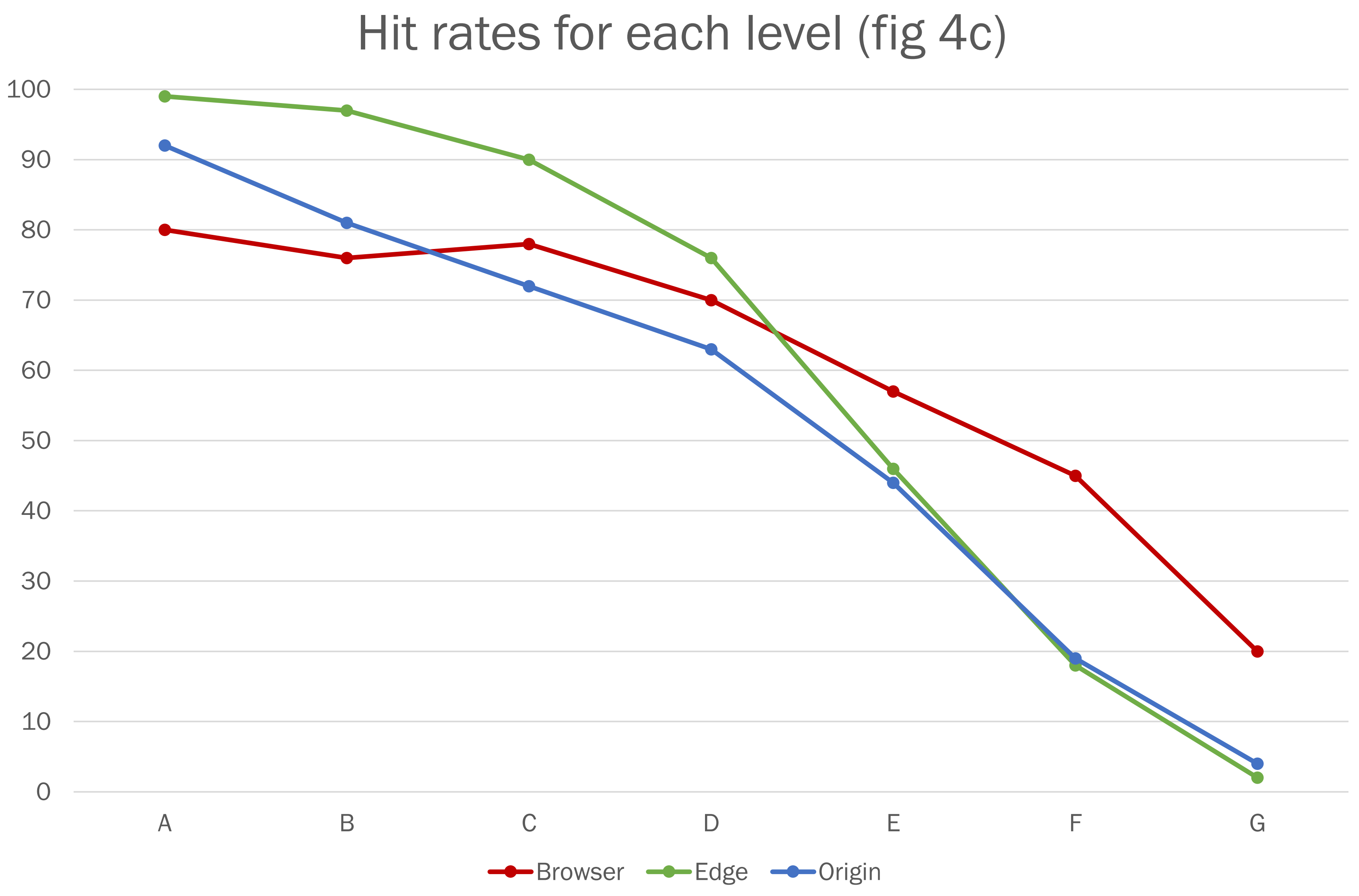


- Backend resembles a stretched exponential dist.

Popularity Impact on Caches



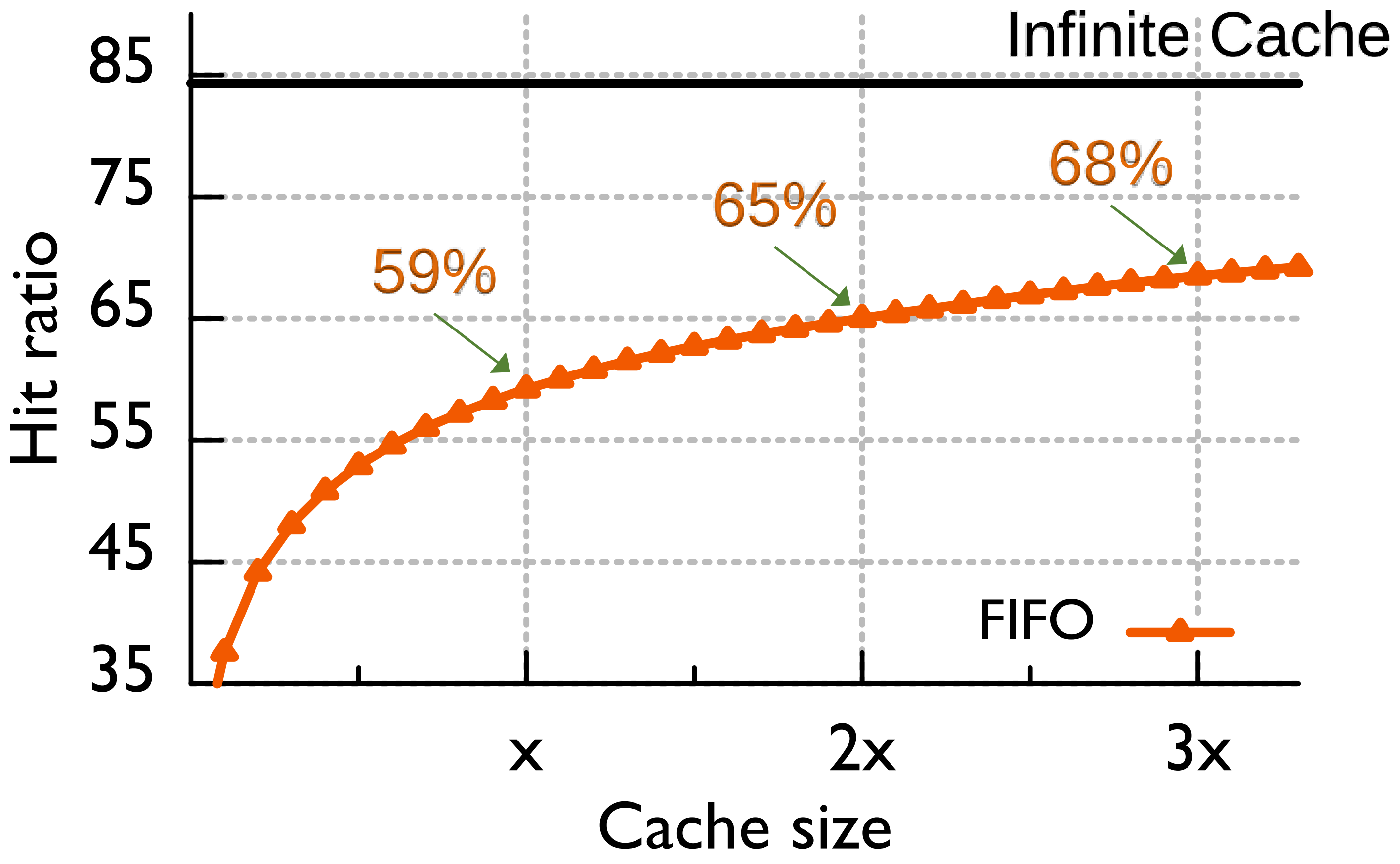
- Backend serves the tail


Hit rates for each level (fig 4c)
100
90
80
70
60
50
40
30
20
10
0
A
B
C
D
E
F
G
Browser
Edge
Origin

What if?

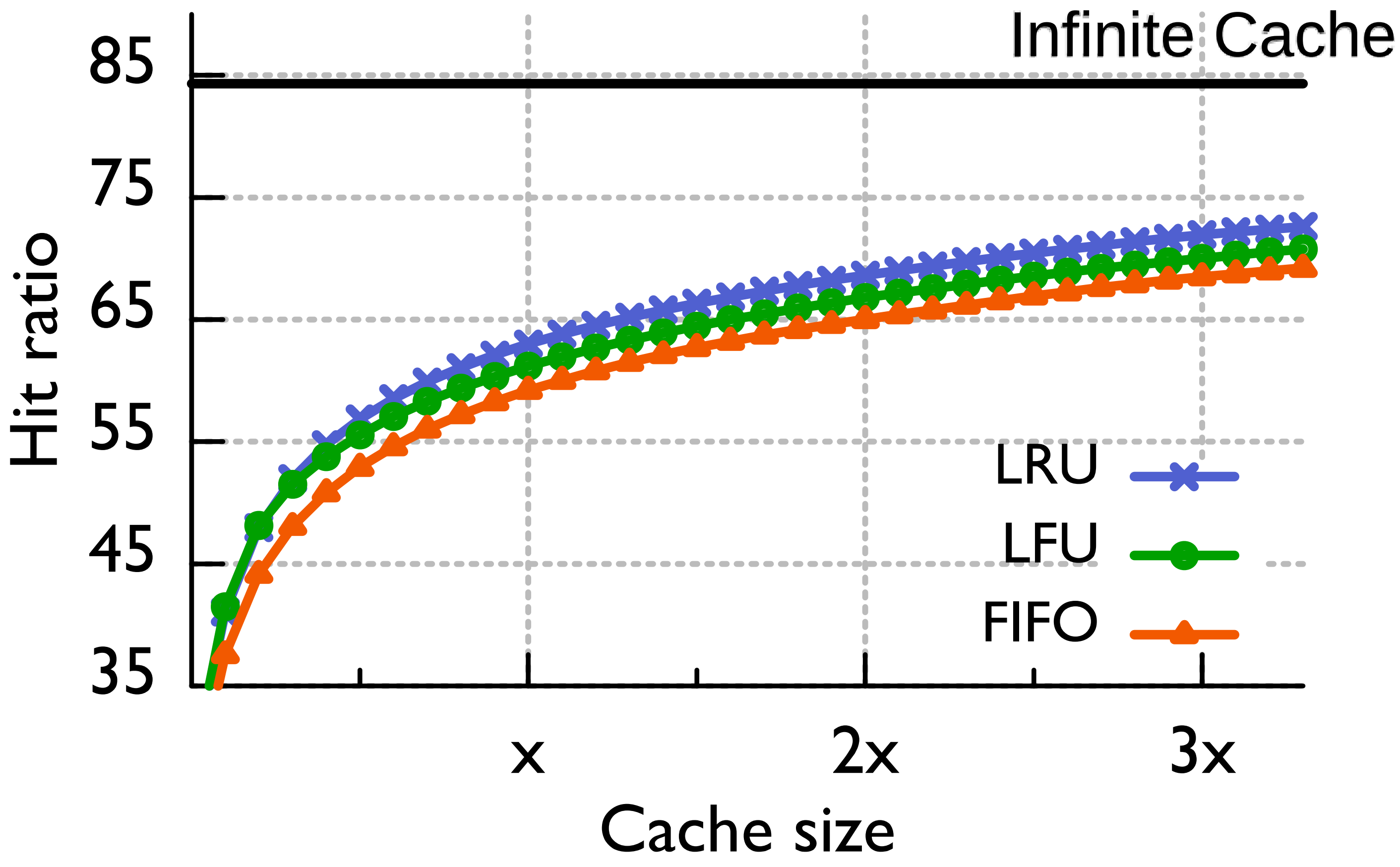
- Picked San Jose edge (high traffic, median hit ratio)

Edge Cache with Different Sizes



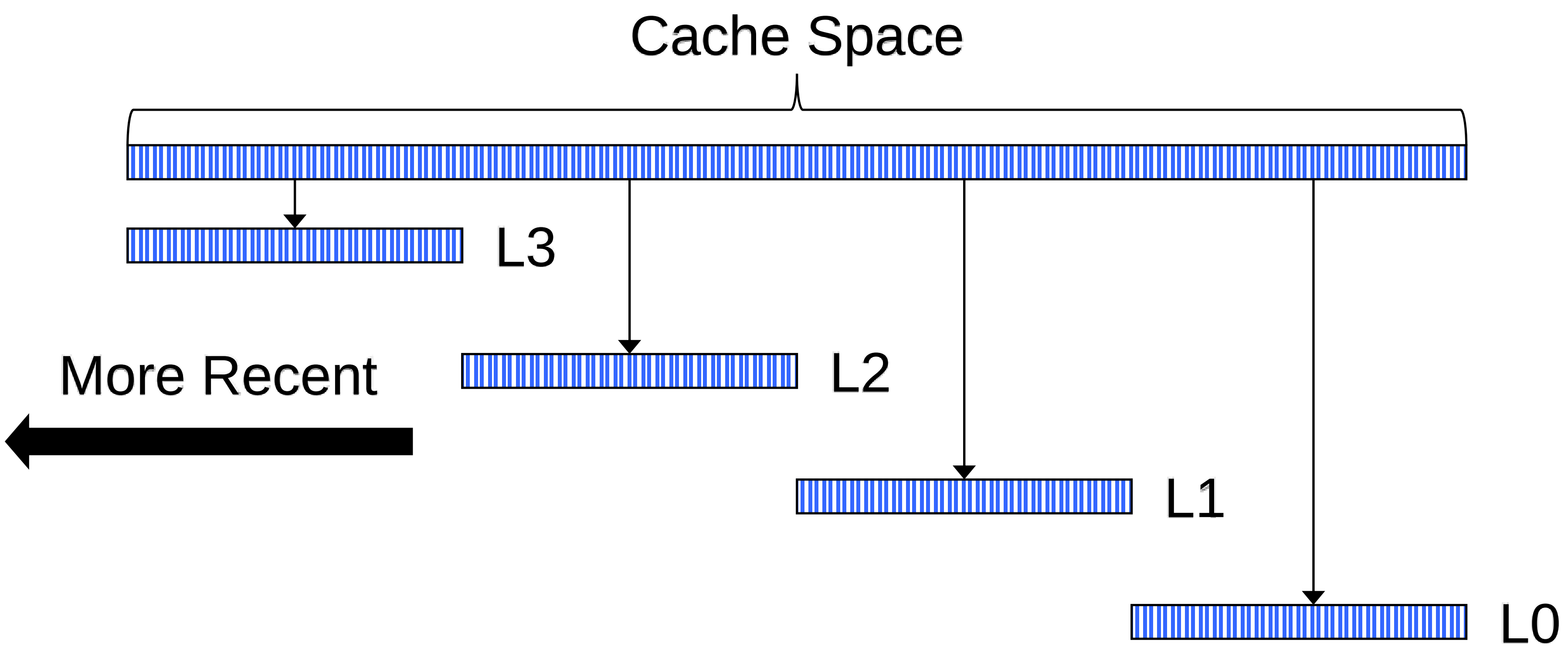
- “Infinite” size ratio needs 45x of current capacity

Edge Cache with Different Algos

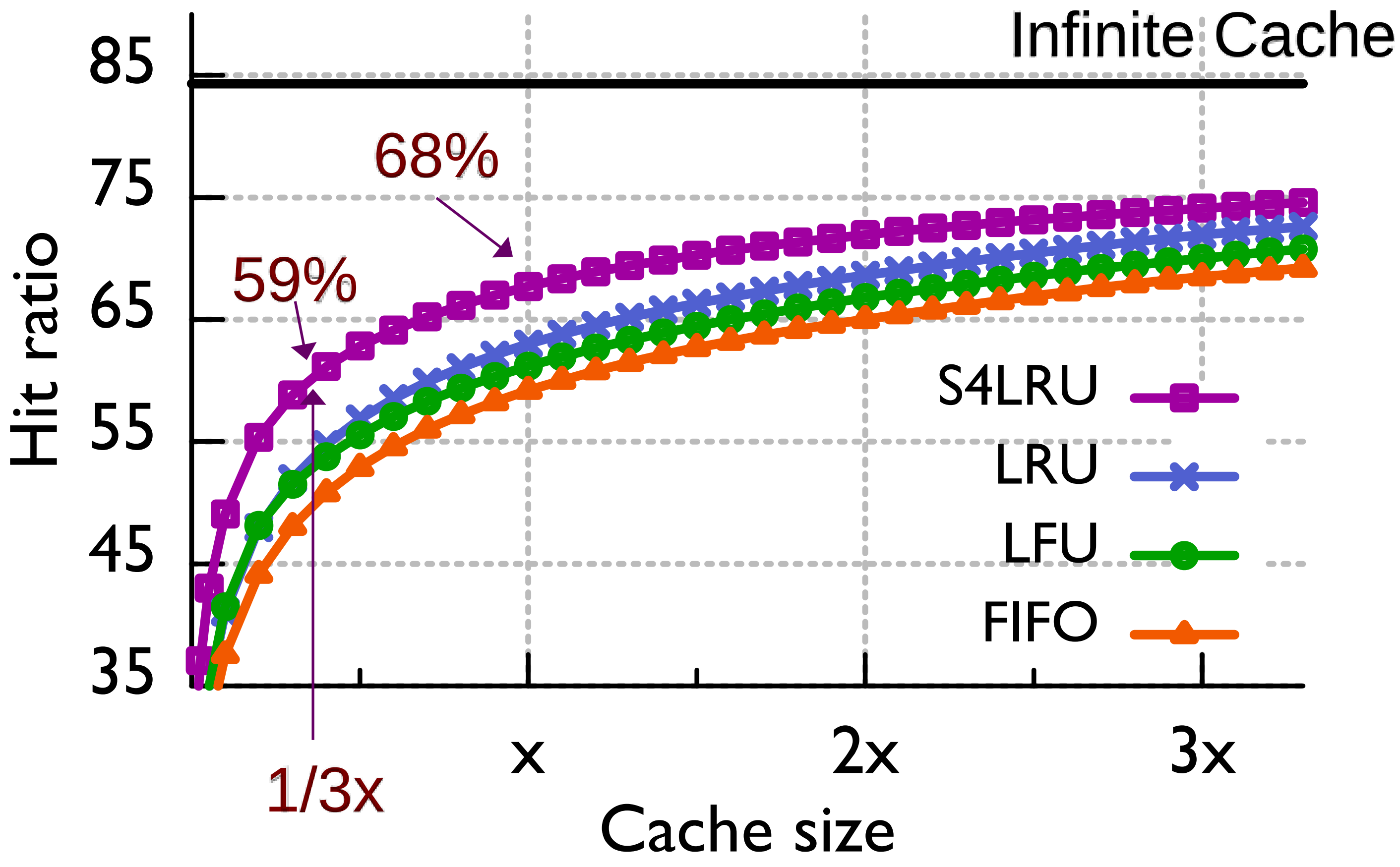


- Both LRU and LFU outperform FIFO slightly

S4LRU

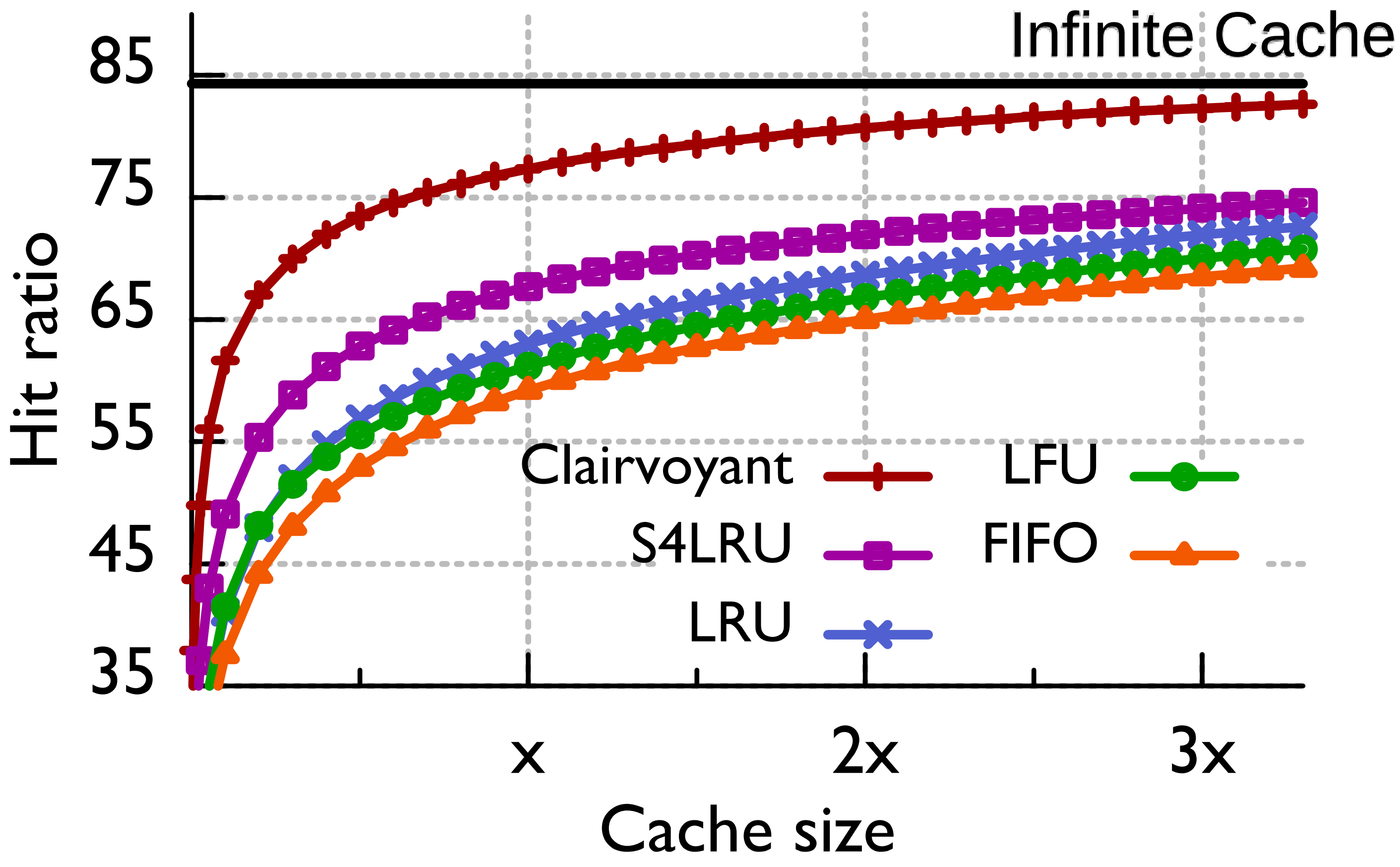


Edge Cache with Different Algos



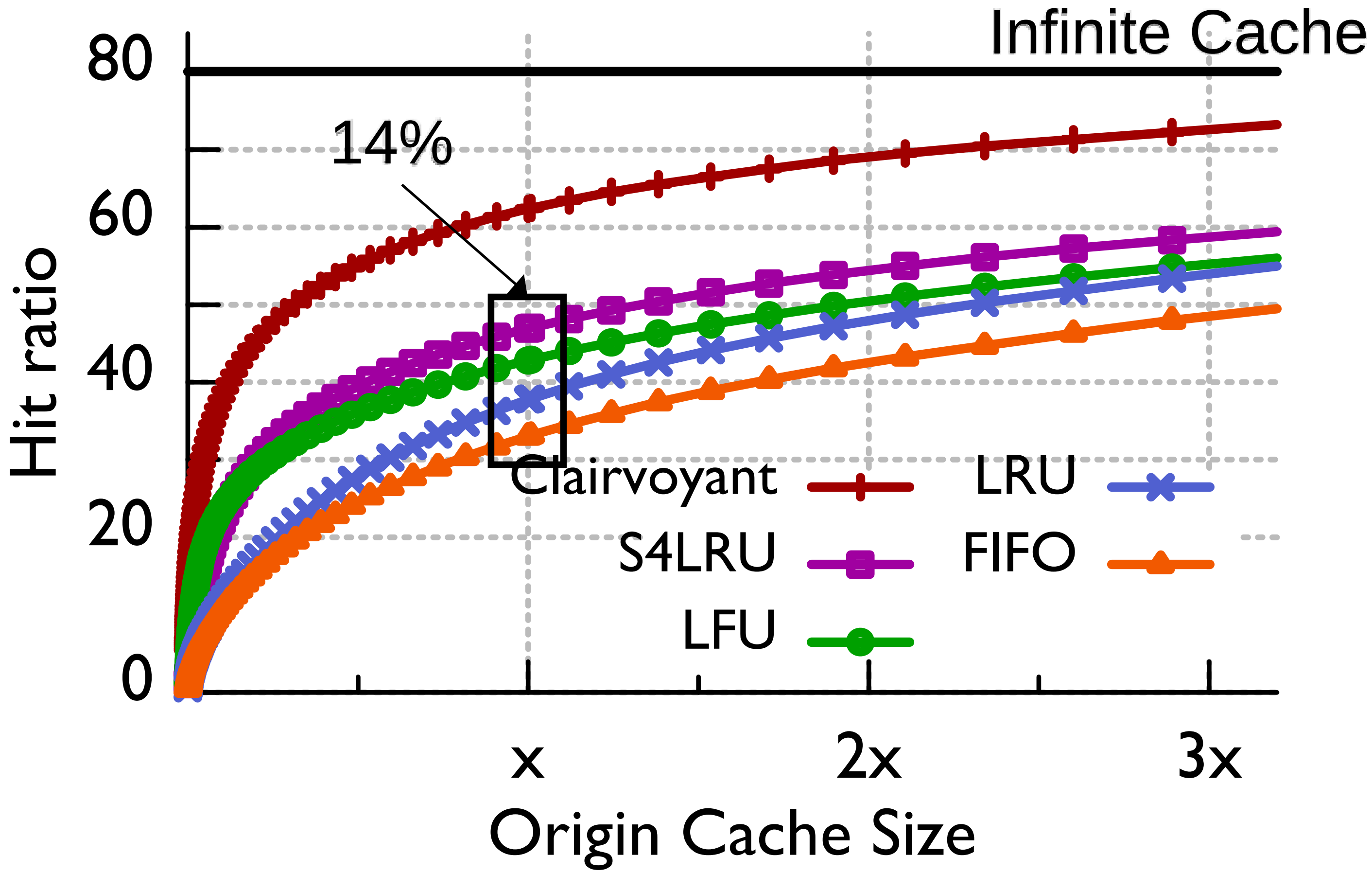
- S4LRU improves the most

Edge Cache with Different Algos



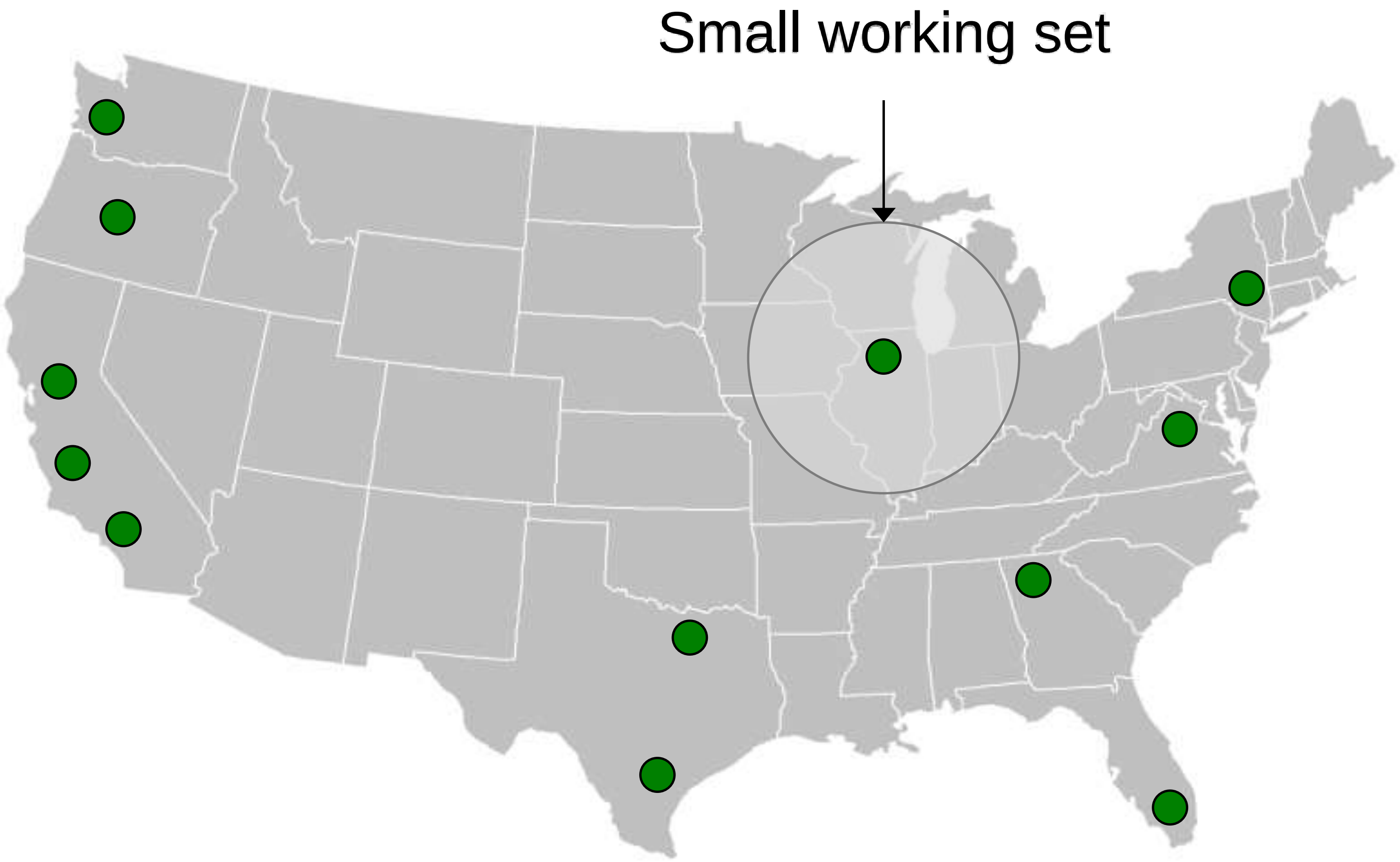
- Clairvoyant => room for algorithmic improvement.

Origin Cache



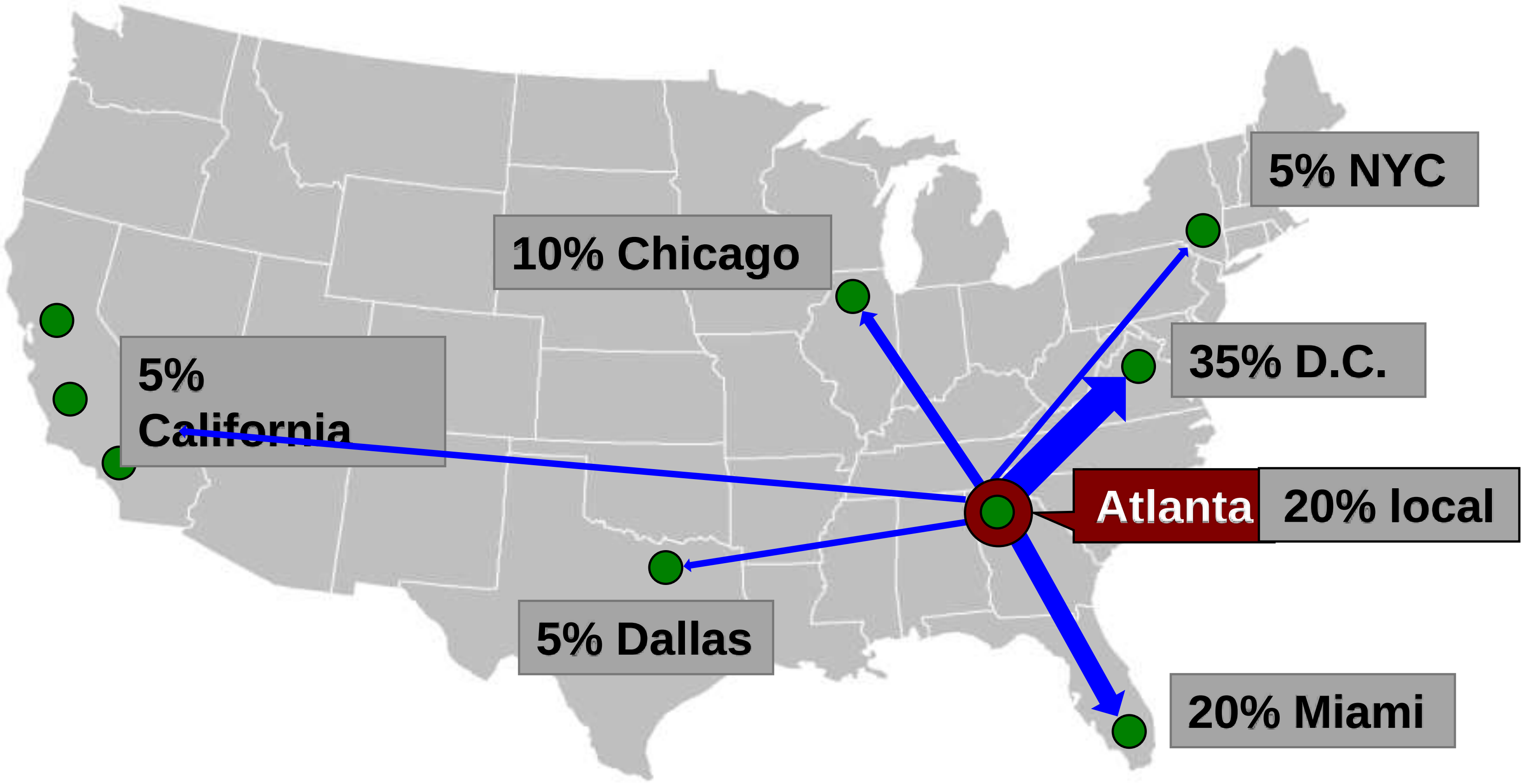
- S4LRU improves Origin more than Edge

Geographic Coverage of Edge

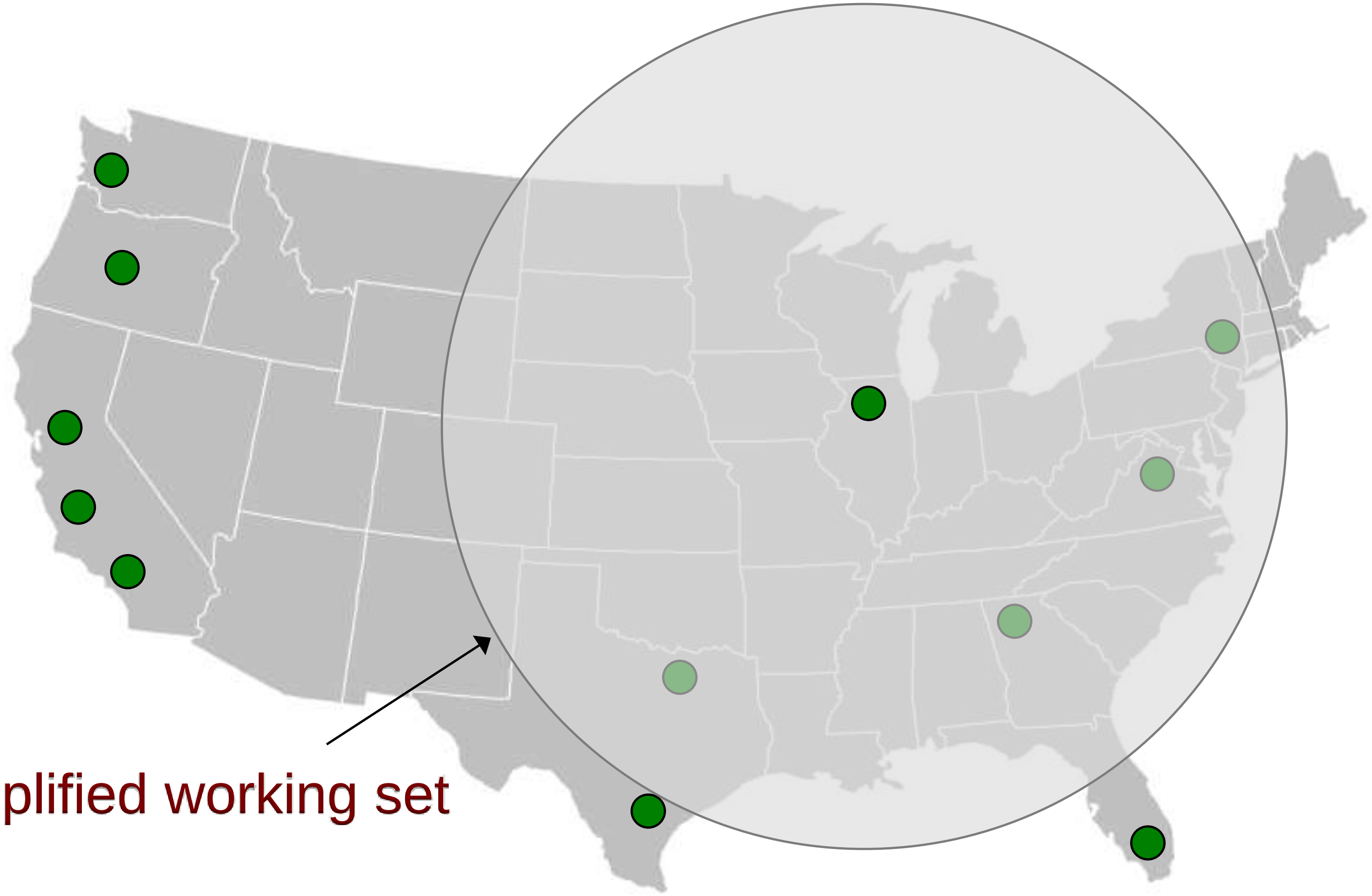


Geographic Coverage of Edge

- Atlanta has 80% requests served by remote Edges. Not uncommon!

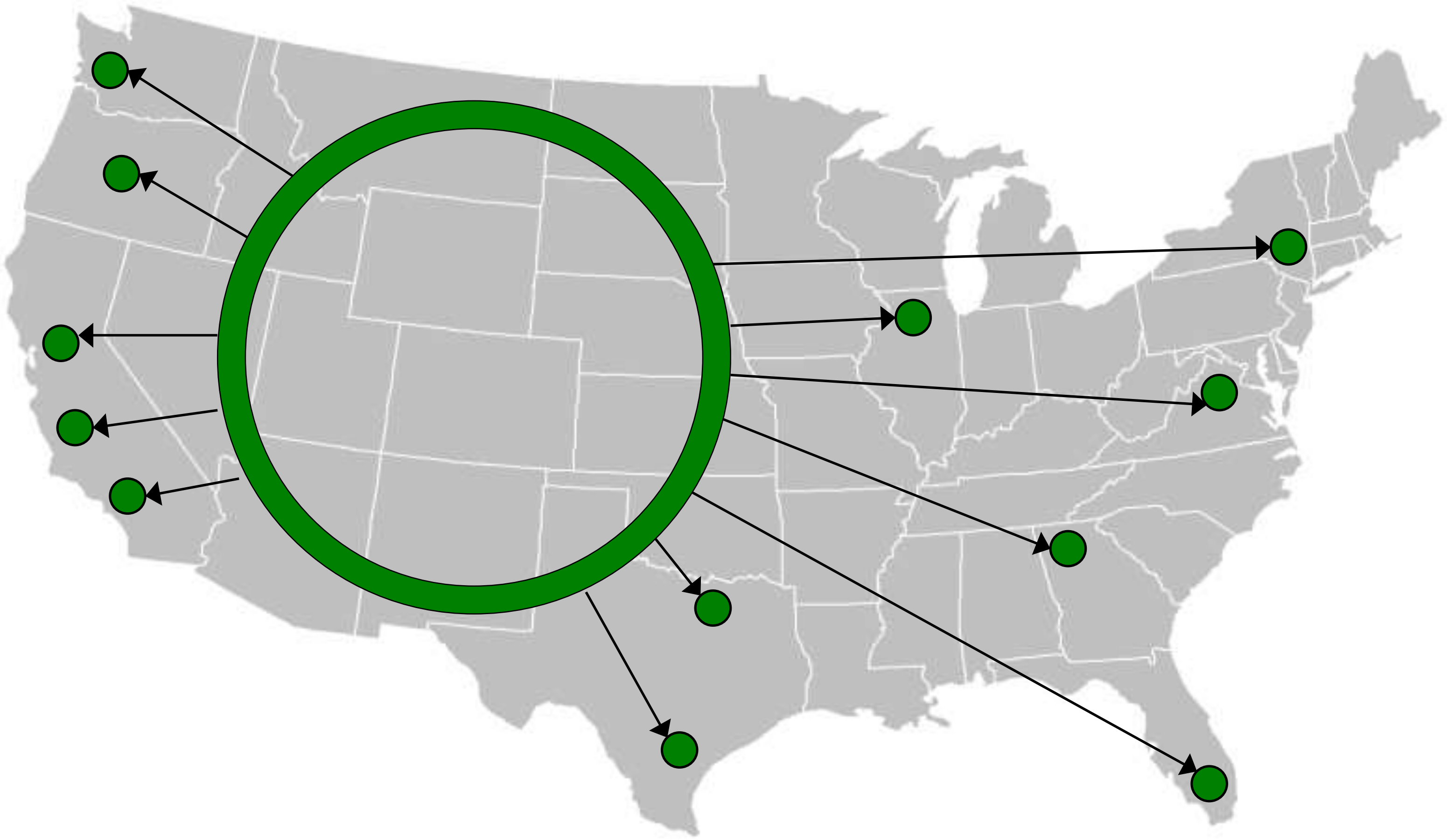


Geographic Coverage of Edge

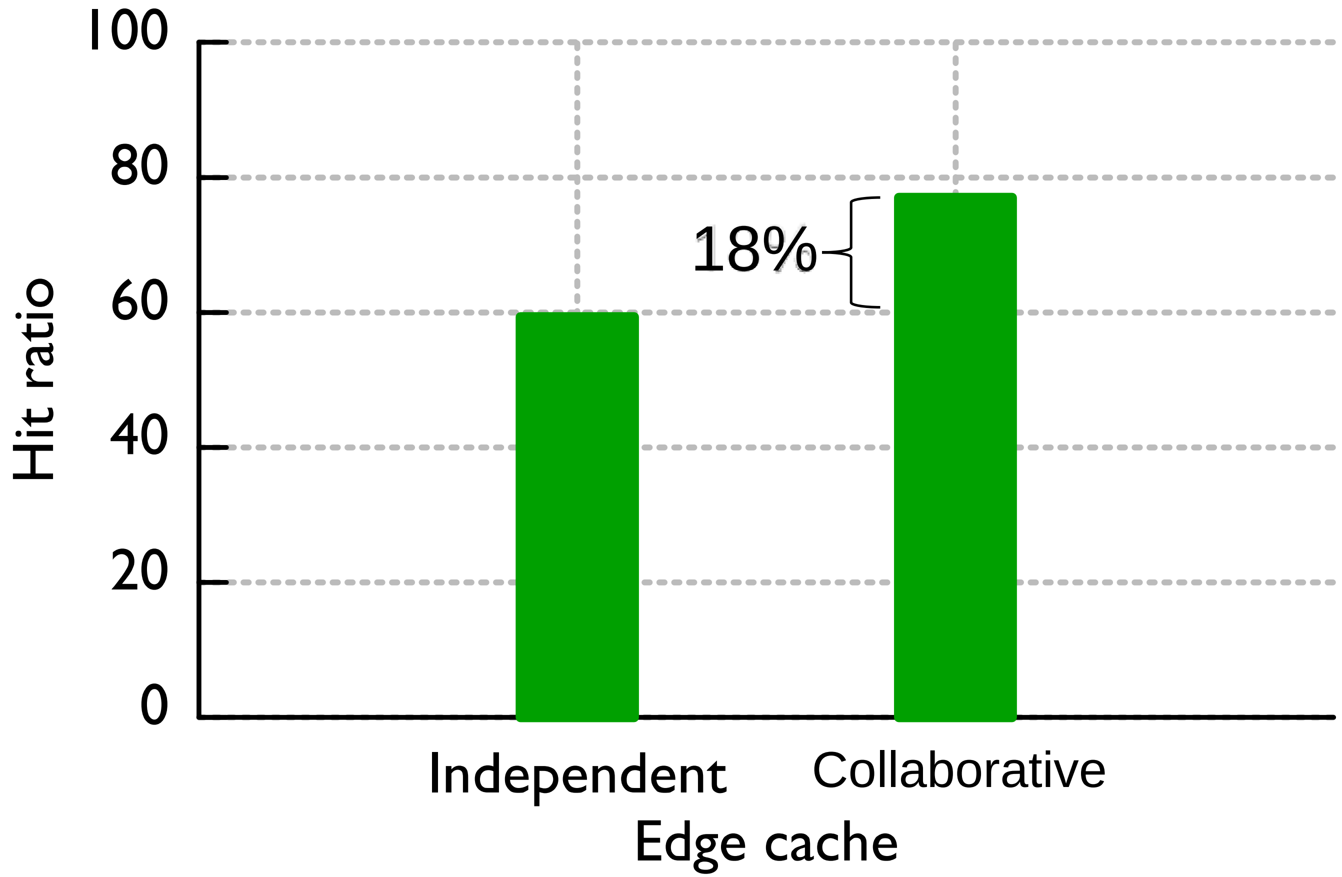


Amplified working set

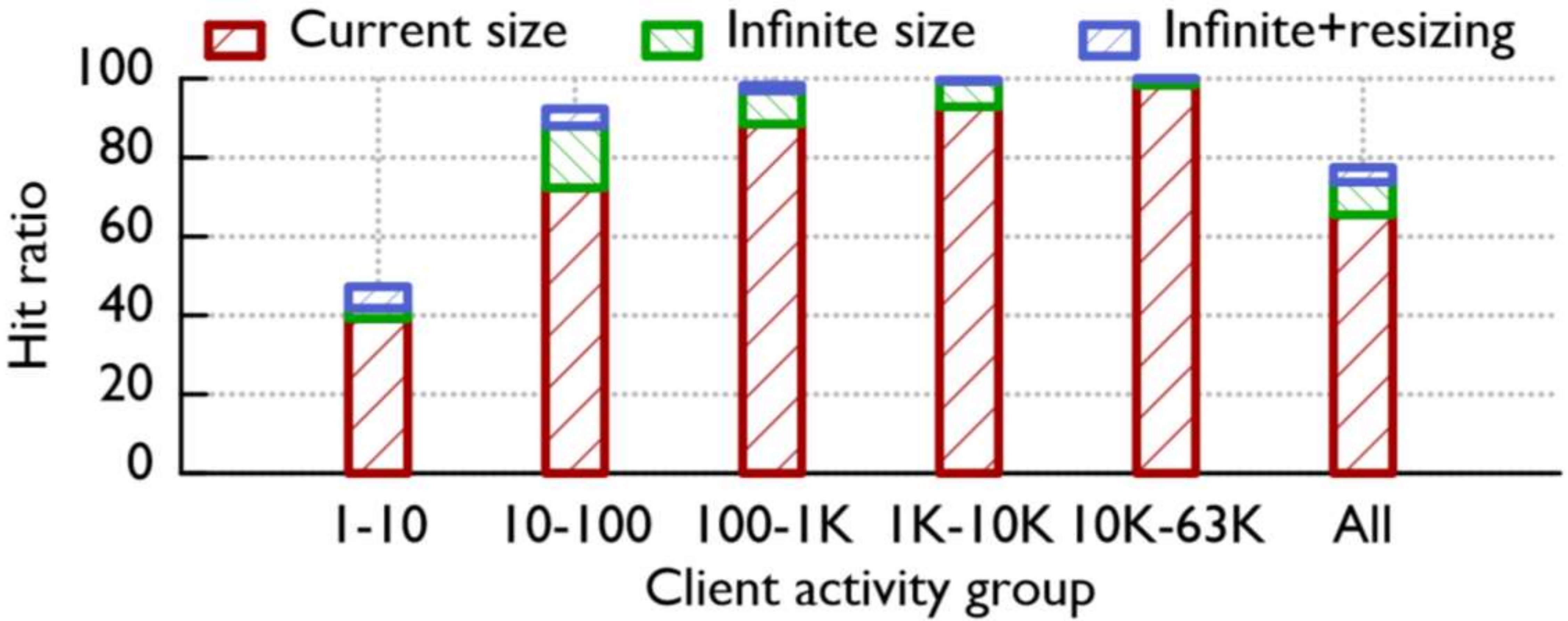
Collaborative Edge



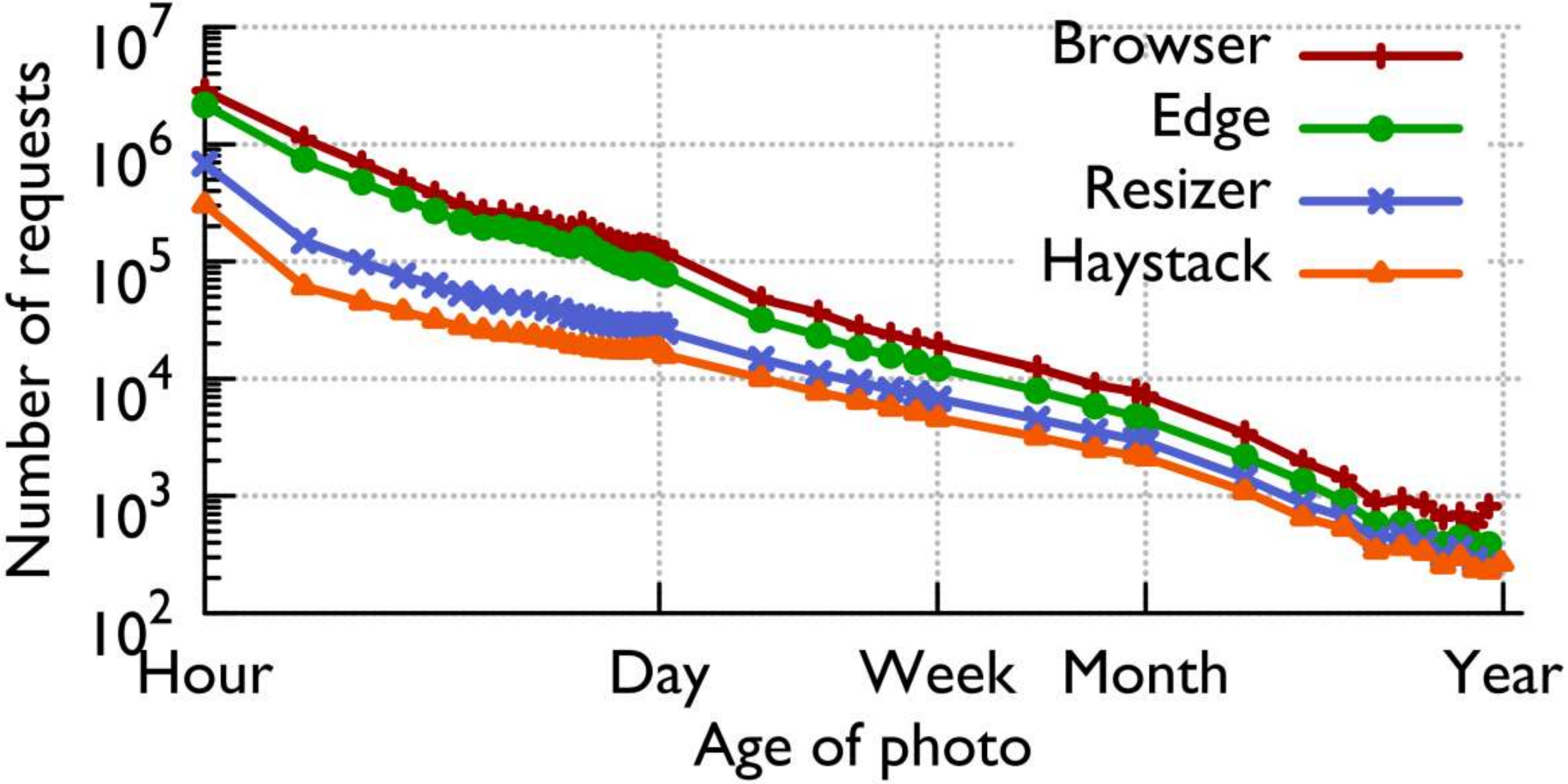
Collaborative Edge



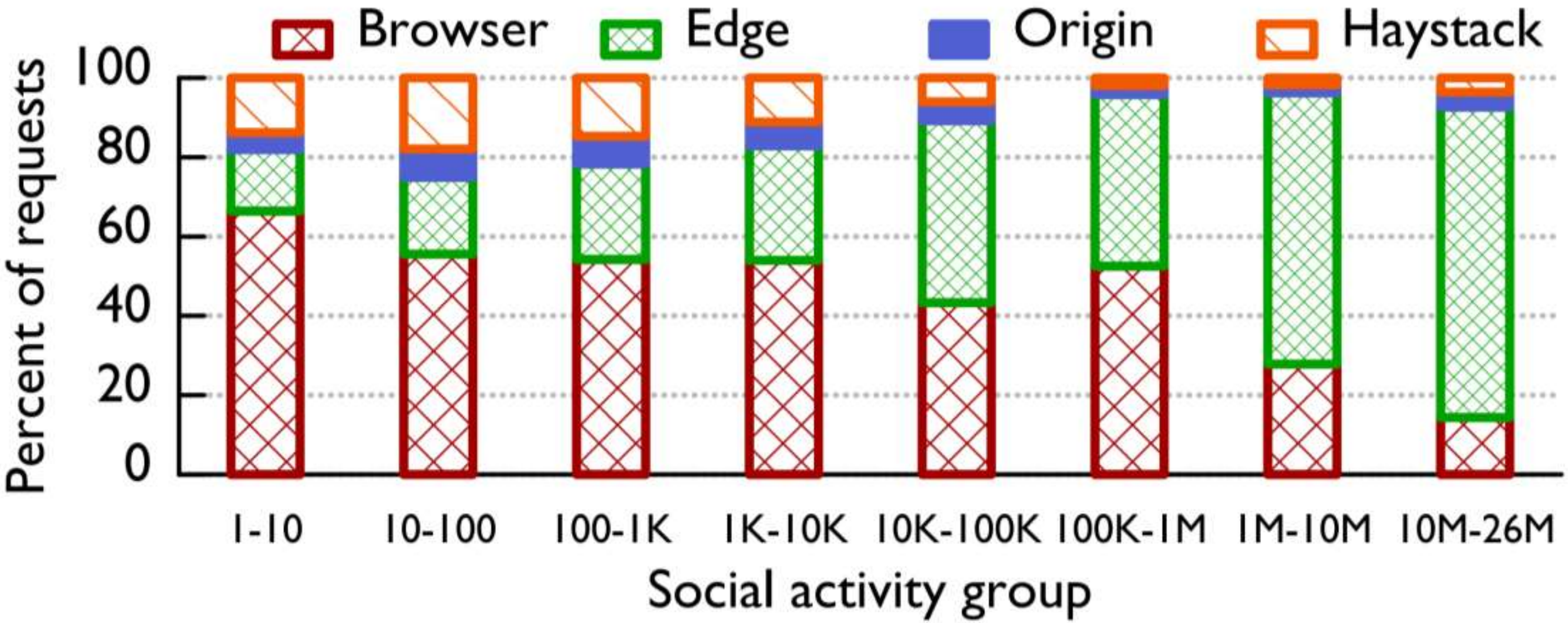
- "Collaborative" Edge increases hit ratio by 18%



Current size
Infinite size
Infinite+resizing
100
80
60
40
20
0
Hit ratio
1-10
10-100
100-1K
1K-10K
10K-63K
All
Client activity group



Browser
Edge
Resizer
Haystack
Number of requests
10^7
10^6
10^5
10^4
10^3
10^2
Hour
Day
Week
Month
Year
Age of photo


Browser
Edge
Origin
Haystack
Percent of requests
100
80
60
40
20
0
1-10
10-100
100-1K
1K-10K
10K-100K
100K-1M
1M-10M
10M-26M
Social activity group

What Facebook Could Do:

- Improve cache algorithm (+invest in cache algo research)
- Coordinate Edge caches
- Let some phones resize their own photos
- Use more machine learning at this layer!

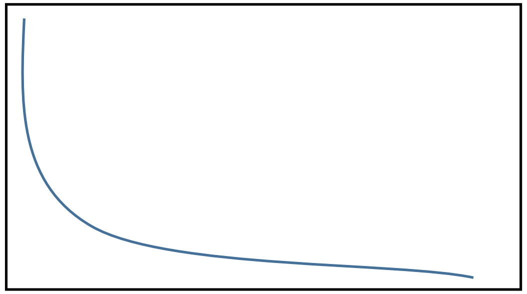
facebook

Finding a needle in Haystack: Facebook's photo storage

Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel

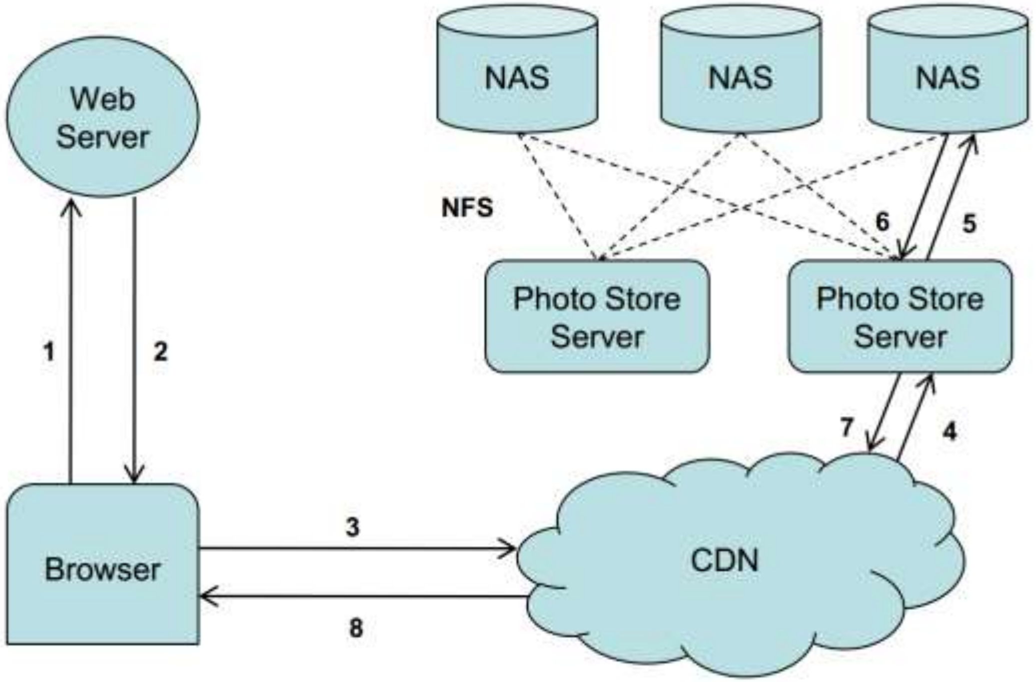
Backend storage for blobs

- Some requests are bound to miss the caches.
- Reads >> writes >> deletes.
- Writes often come in batches (Photo Albums)
- In this regime, Facebook found default solutions not to work.



facebook

NFS based Design



NFS based Design

- Metadata bottleneck
  - Each image stored as a file
  - Large metadata size severely limits the metadata hit ratio

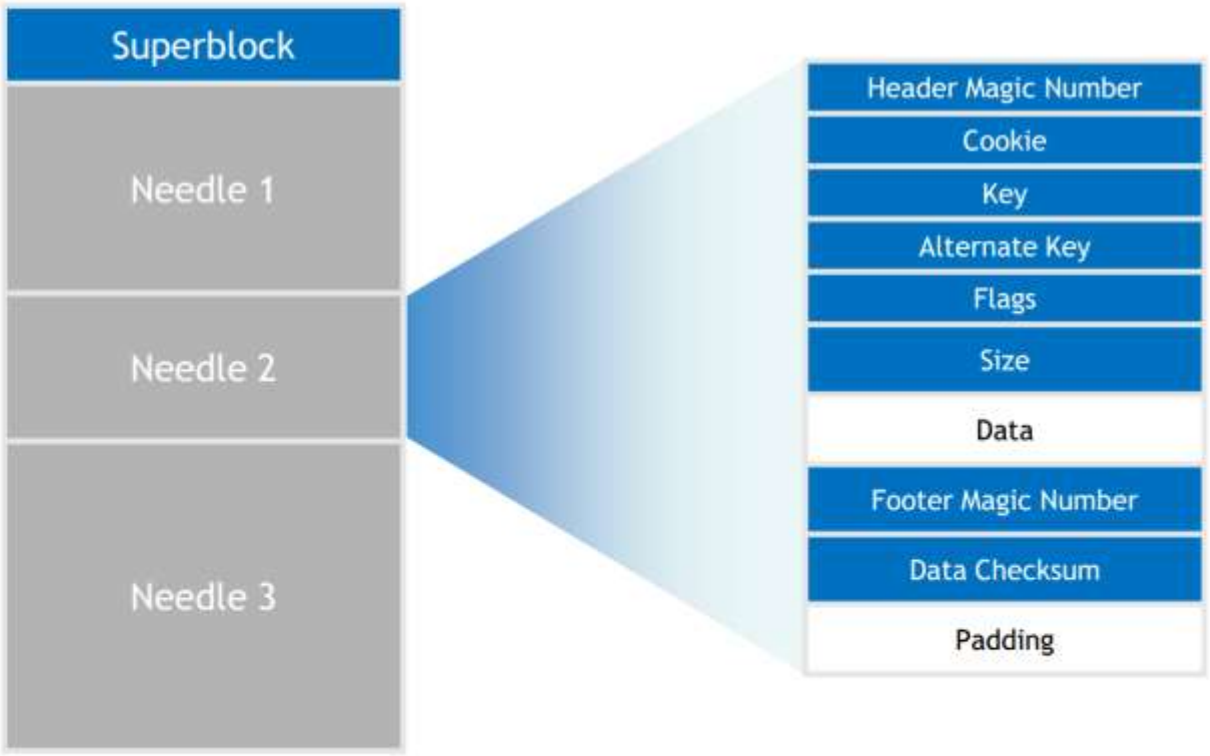
- Image read performance

  ~10 iops / image read (large directories – thousands of files)

  ~3 iops / image read (smaller directories – hundreds of files)

  ~2.5 iops / image read (file handle cache)

Haystack Store – Haystack file Layout



facebook

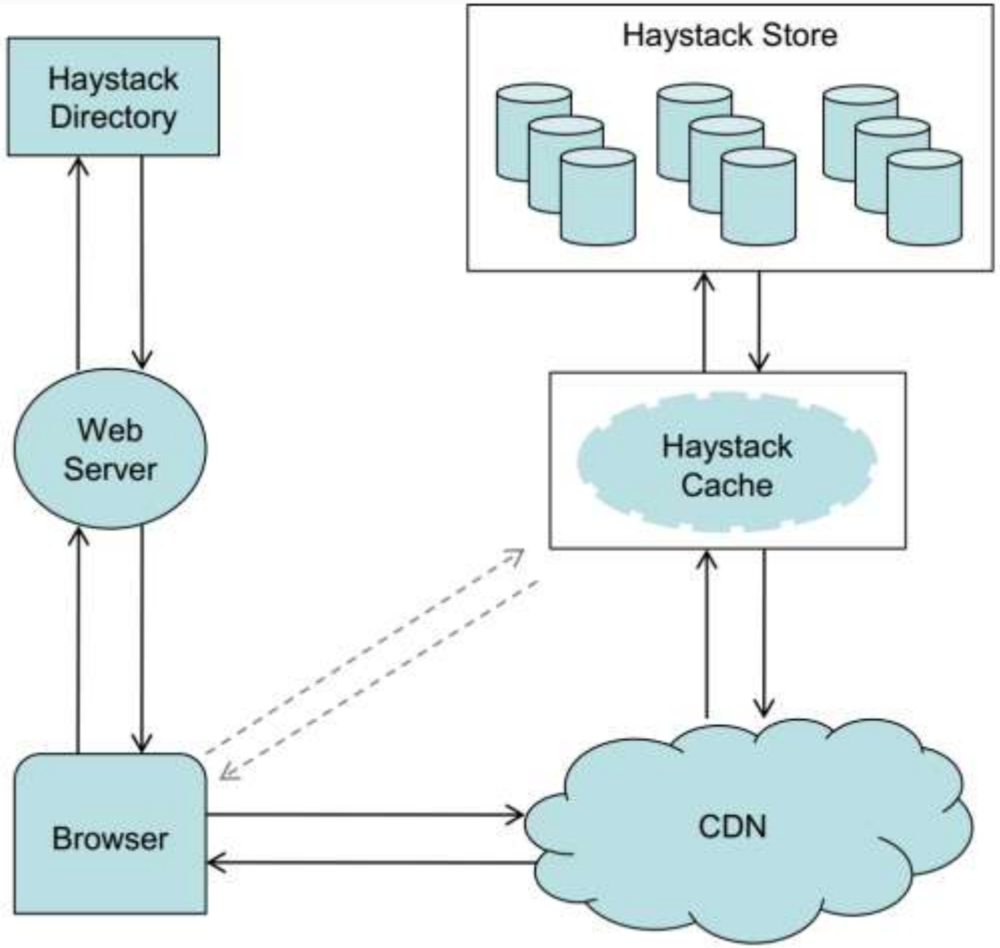
Haystack Store - Photo Server

- Accepts HTTP requests and translates them to corresponding Haystack operations
- Builds and maintains an incore index of all images in the Haystack
- 32 bytes per photo (8 bytes per image vs. ~600 bytes per inode)
- ~5GB index / 10TB of images

64-bit photo key
1st scaled image 32-bit offset / 16-bit size
2nd scaled image 32-bit offset / 16-bit size
3rd scaled image 32-bit offset / 16-bit size
4th scaled image 32-bit offset / 16-bit size

facebook

Haystack based Design



Haystack Store

- Storage
  - 12x 1TB SATA, RAID6

- Filesystem
  - Single ~10TB xfs filesystem

- Haystack
  - Log structured, append only object store containing needles as object abstractions
  - 100 haystacks per node each 100GB in size

facebook

Haystack Store Operations

- Read
  - Lookup offset / size of the image in the incore index
  - Read data (~1 iop)
- Multiwrite (Modify)
  - Asynchronously append images one by one to the haystack file
  - Flush haystack file
  - Asynchronously append index records to the index file
  - Flush index file if too many dirty index records
  - Update incore index

facebook

Haystack Store Operations

- Delete
  - Lookup offset of the image in the incore index
  - Synchronously mark image as “DELETED” in the needle header
  - Update incore index

- Compaction
  - Infrequent online operation
  - Create a copy of haystack skipping duplicates and deleted photos

Conclusion

- Haystack – simple and effective storage system
  - Optimized for random reads (~1 I/O per object read)
  - Cheap commodity storage
  - 8,500 LOC (C++)
  - 2 engineers 4 months from inception to initial deployment
- Future work
  - Software RAID6
  - Limit dependency on external CDN
  - Index on flash