







# facebook

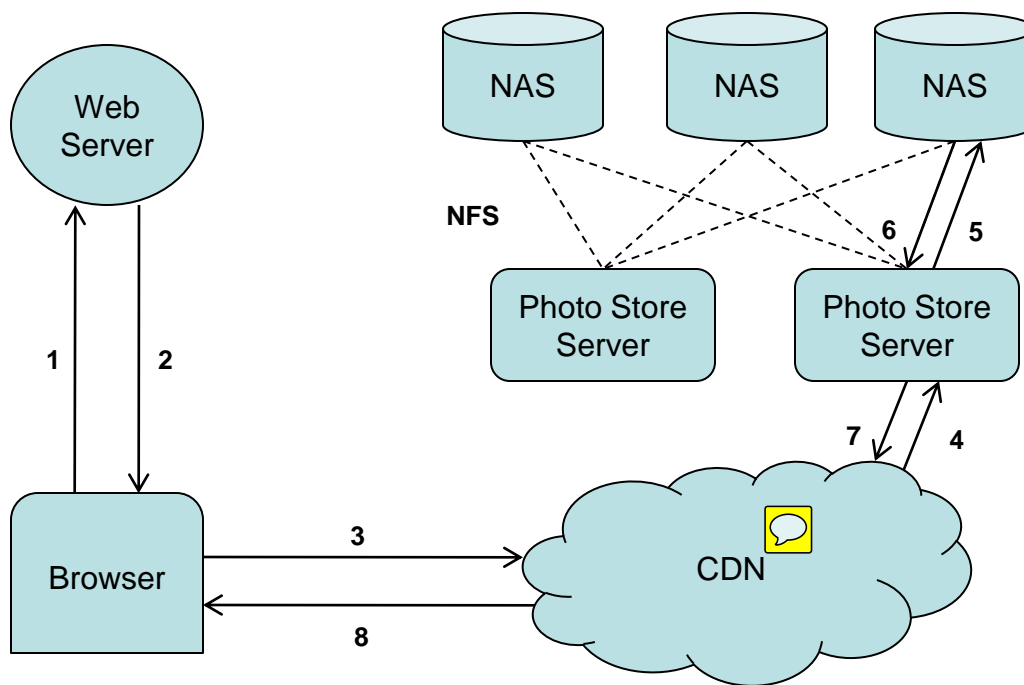
## Finding a needle in Haystack: Facebook's photo storage

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


## Photos @ Facebook

	April 2009	Current 
Total	15 billion photos 60 billion images  1.5 petabytes	65 billion photos 260 billion images  20 petabytes 
Upload Rate	220 million photos / week 25 terabytes	1 billion photos / week 60 terabytes
Serving Rate	 550,000 images / sec	 1 million images / sec

# NFS based Design



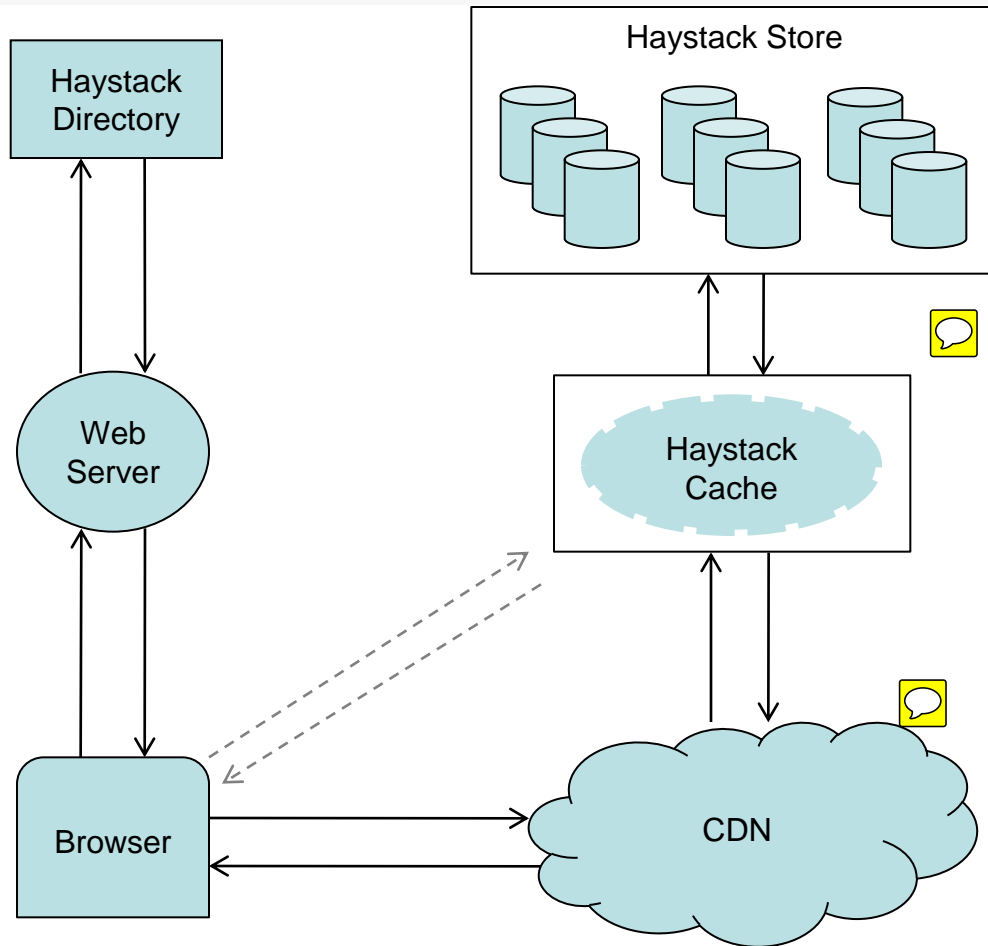
## NFS based Design

- Typical website
  - Small working set
  - Infrequent access of old content
  - ~99% CDN hit rate
- Facebook
  - Large working set
  - Frequent access of old content 
  - 80% CDN hit rate

## NFS based Design

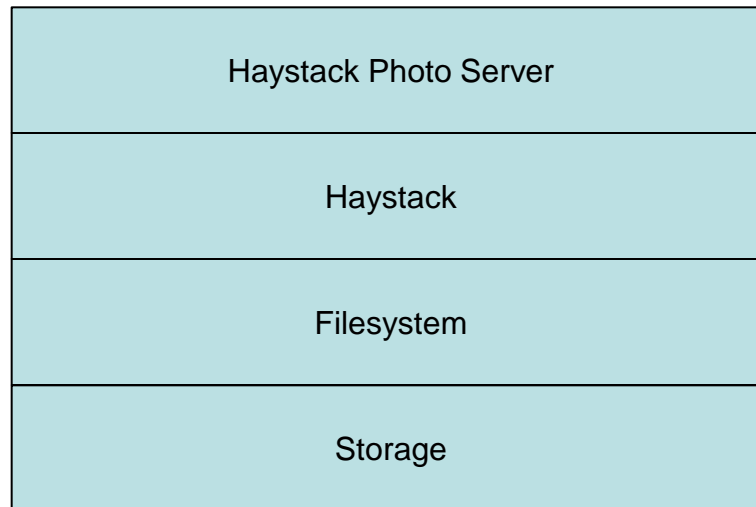
- Metadata bottleneck
  - Each image stored as a file
  - Large metadata size severely limits the metadata hit ratio
- Image read performance
  - ~10 iops / image read (large directories - thousands of files)
  - ~3 iops / image read (smaller directories - hundreds of files)
  - ~2.5 iops / image read (file handle cache)

# Haystack based Design





# Haystack Store

- Replaces Storage and Photo Server in NFS based Design

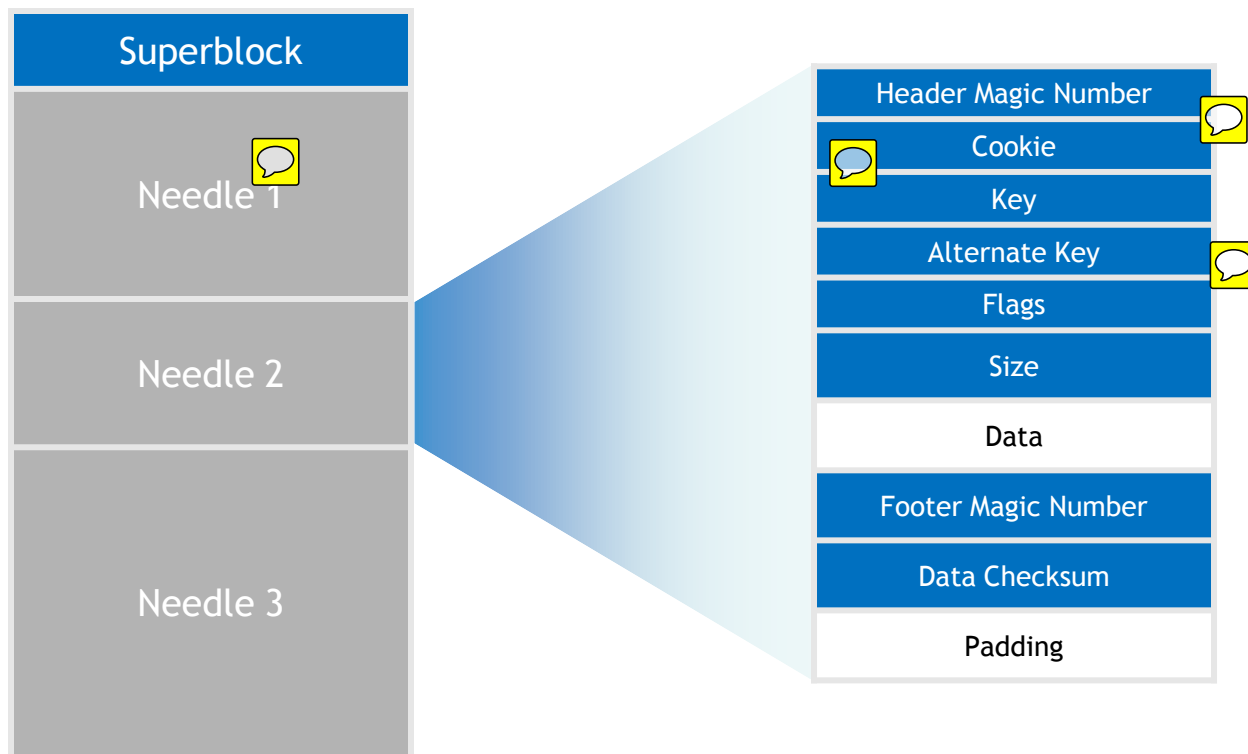


# Haystack Store

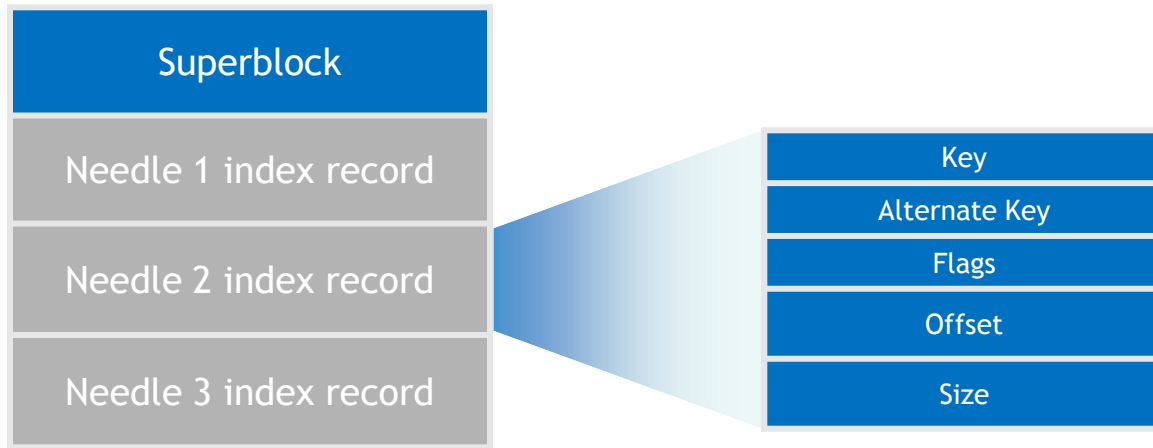
- Storage
  - 12x 1TB SATA, RAID6
- Filesystem 
  - Single ~10TB xfs filesystem
- Haystack
  - Log structured, append only object store containing needles as object abstractions
  - 100 haystacks  per node each 100GB in size



# Haystack Store - Haystack file Layout



# Haystack Store - Haystack Index File Layout



## 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
1 <sup>st</sup> scaled image 32-bit offset / 16-bit size
2 <sup>nd</sup> scaled image 32-bit offset / 16-bit size
3 <sup>rd</sup> scaled image 32-bit offset / 16-bit size
4 <sup>th</sup> scaled image 32-bit offset / 16-bit size

# 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

# Haystack Store Operations

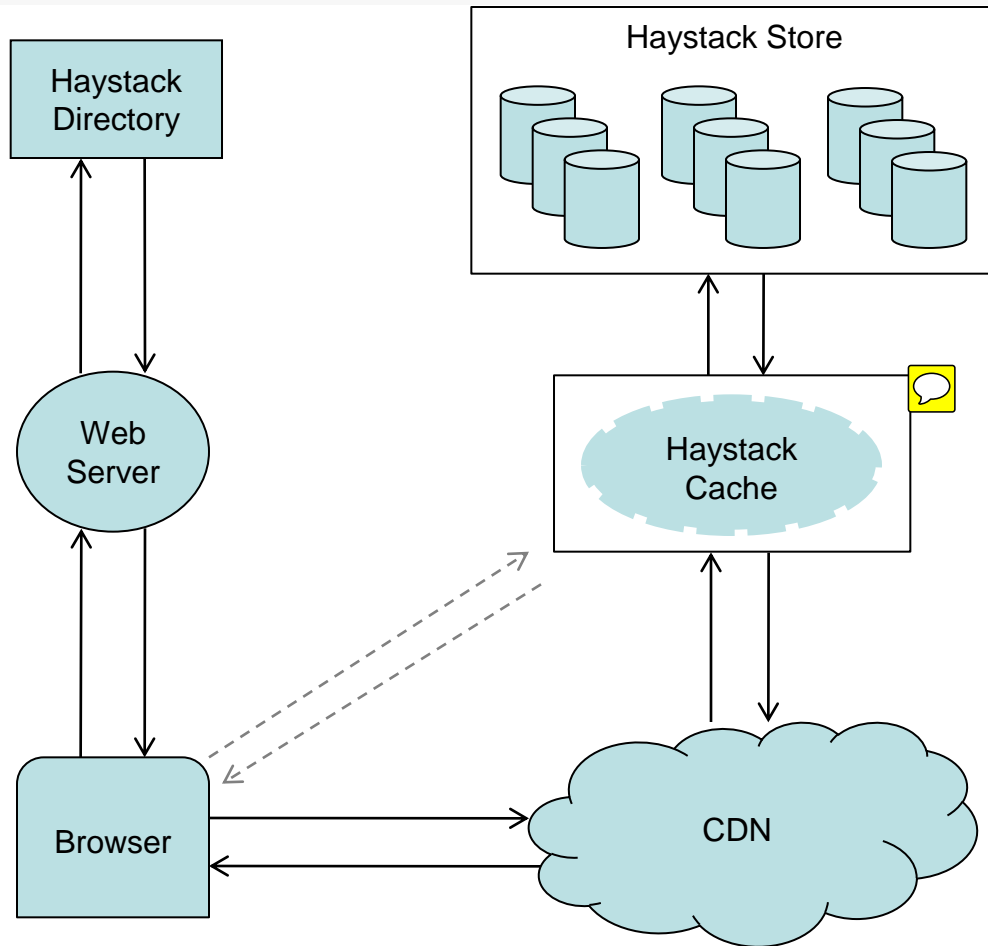
- Delete

- 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

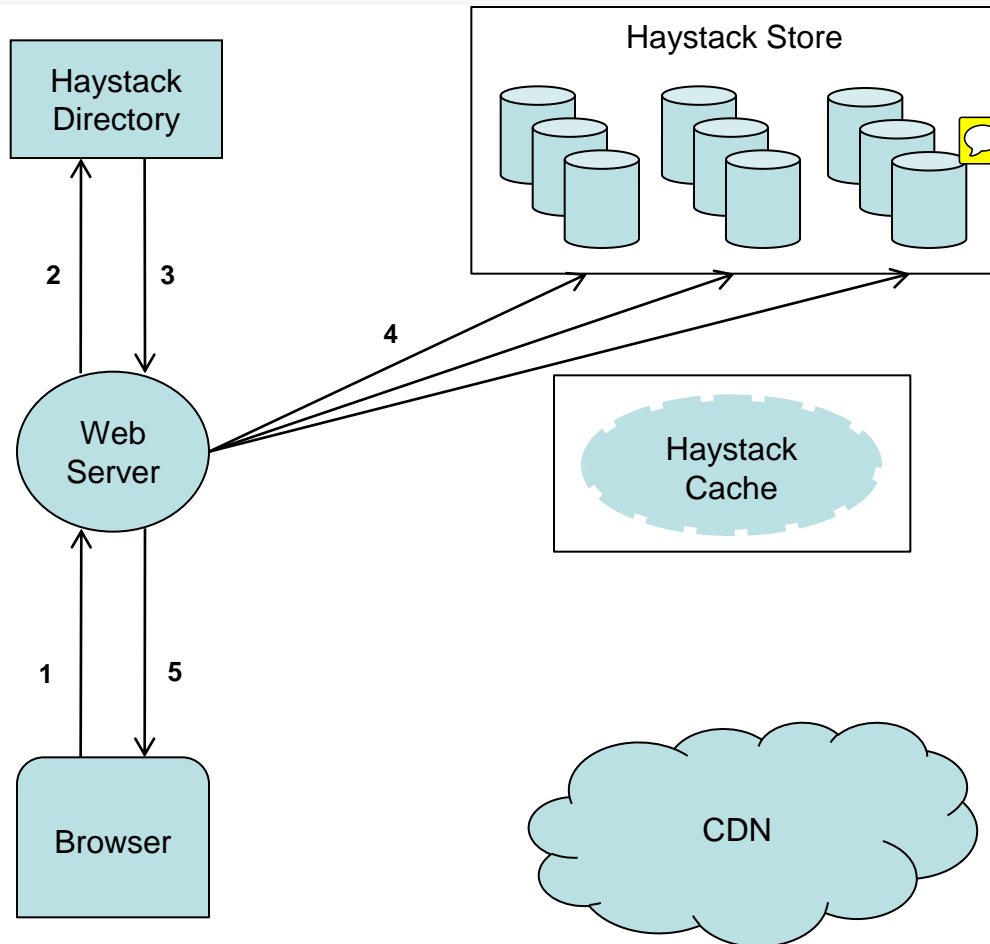
# Haystack based Design



## Haystack Directory

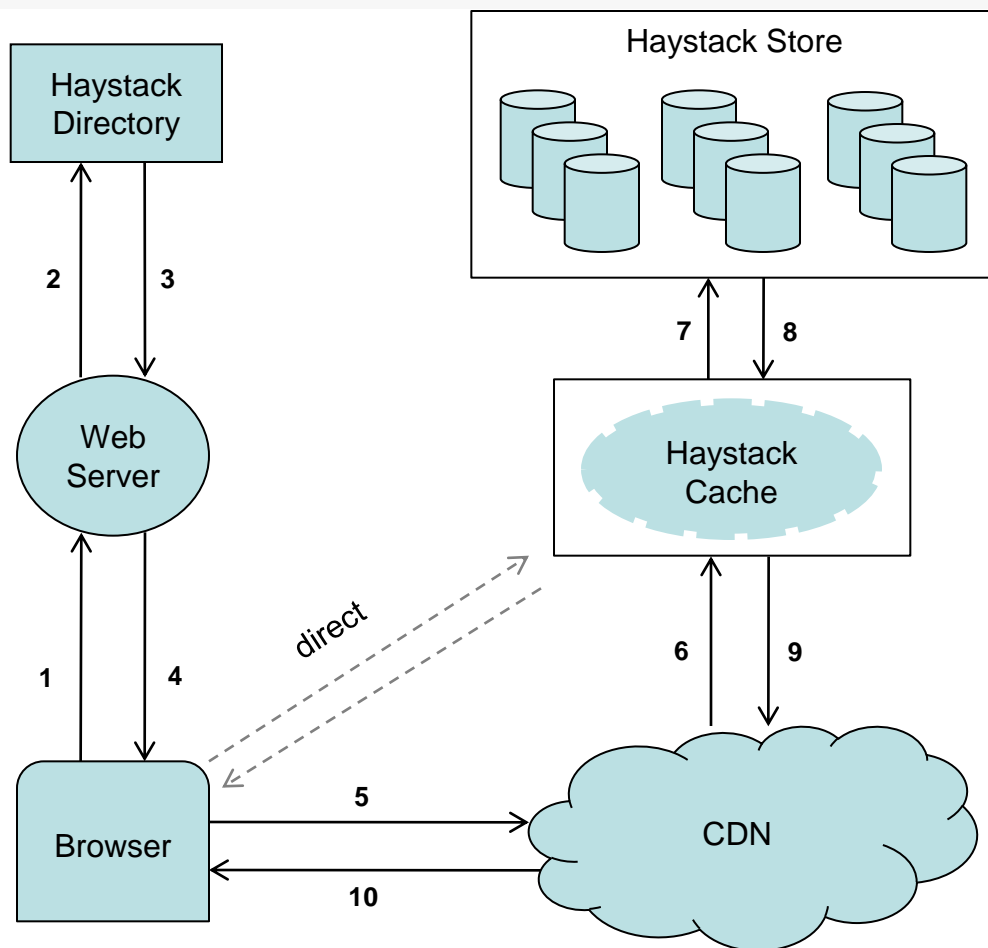
- Logical to physical volume mapping
  - 3 physical haystacks (on 3 nodes) per one logical volume
- URL generation
  - `http://<CDN>/<Cache>/<Node>/<Logical volume id, Image id>`
- Load Balancing
  - Writes across logical volumes
  - Reads across physical haystacks
- Caching strategy
  - External CDN or Local cache?

# Haystack based Design - Photo Upload





# Haystack based Design - Photo Download



## 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

## Q&A

- Thanks!