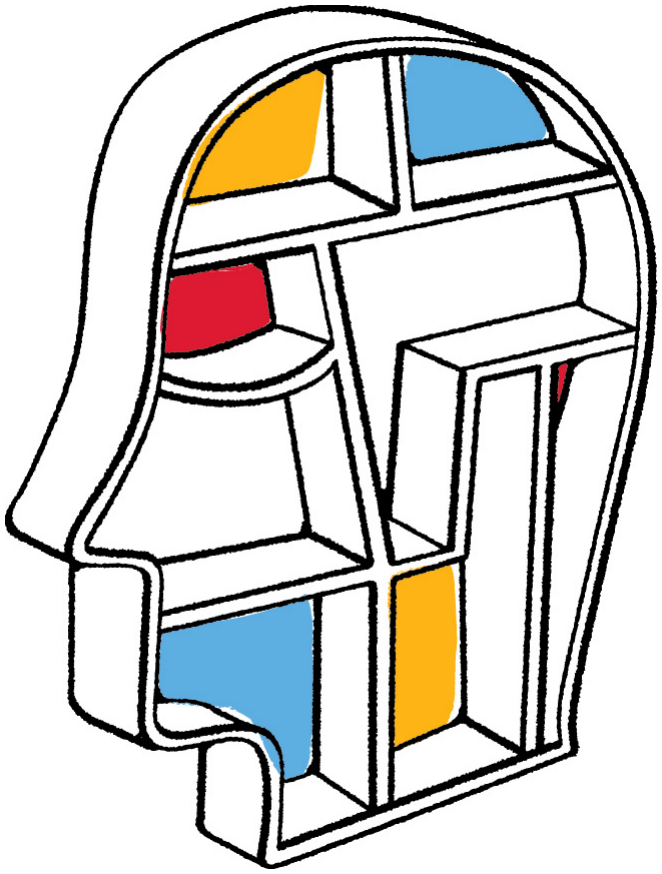
NetApp

Go further, faster®

iDedup

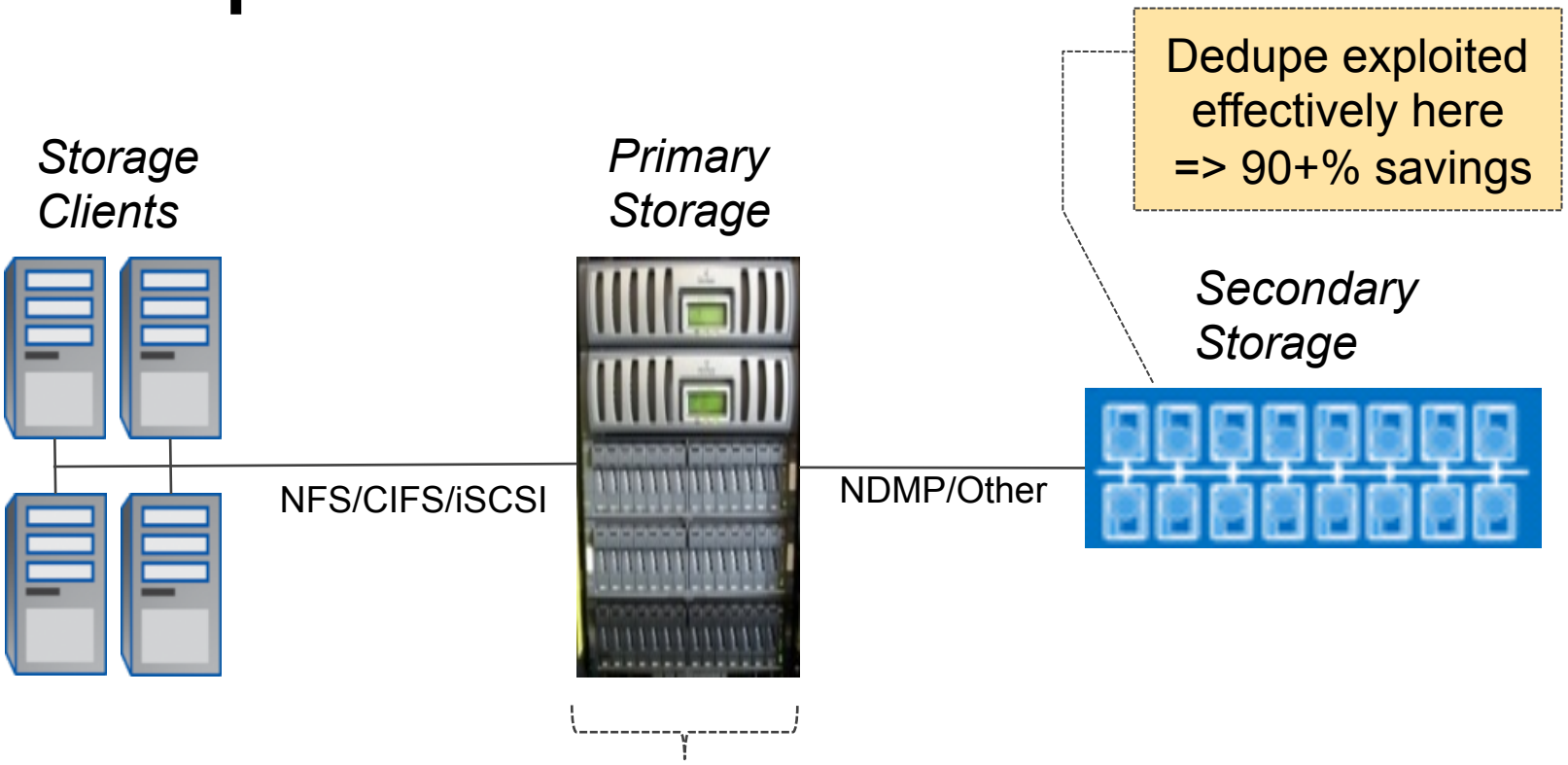
Latency-aware inline deduplication for primary workloads



Kiran Srinivasan, Tim Bisson
Garth Goodson, Kaladhar Voruganti

Advanced Technology Group
NetApp

iDedup – overview/context



iDedup

- Inline/foreground dedupe technique for primary
- Minimal impact on latency-sensitive workloads

Dedupe techniques (offline vs inline)

Offline dedupe

- First copy on stable storage is not deduped
- Dedupe is a post-processing/background activity

Inline dedupe

- Dedupe before first copy on stable storage
- Primary => Latency should not be affected!
- Secondary => Dedupe at ingest rate (IOPS)!

Why inline dedupe for primary?

Provisioning/Planning is easier

- Dedupe savings are seen right away
- Planning is simpler as capacity values are accurate

No post-processing activities

- No scheduling of background processes
- No interference => front-end workloads are not affected
- Key for storage system users with limited maintenance windows

Efficient use of resources

- Efficient use of I/O bandwidth (offline has both reads + writes)
- File system's buffer cache is more efficient (deduped)

Performance challenges have been the key obstacle

- Overheads (CPU + I/Os) for both reads and writes hurt latency

iDedup – Key features

Minimizes inline dedupe performance overheads

- Leverages workload characteristics
- Eliminates almost all extra I/Os due to dedupe processing
- CPU overheads are minimal

Tunable tradeoff: dedupe savings vs performance

- Selective dedupe => some loss in dedupe capacity savings

iDedup can be combined with offline techniques

- Maintains same on-disk data structures as normal file system
- Offline dedupe can be run optionally

Related Work

Workload/ Method	Offline	Inline
Primary	NetApp ASIS EMC Celerra IBM StorageTank	**iDedup** zFS*
Secondary	(No motivation for systems in this category)	EMC DDFS, EMC Cluster DeepStore, NEC HydraStor, Venti, SiLo, Sparse Indexing, ChunkStash, Foundation, Symantec, EMC Centera

Outline

- Inline dedupe challenges
- Our Approach
- Design/Implementation details
- Evaluation results
- Summary

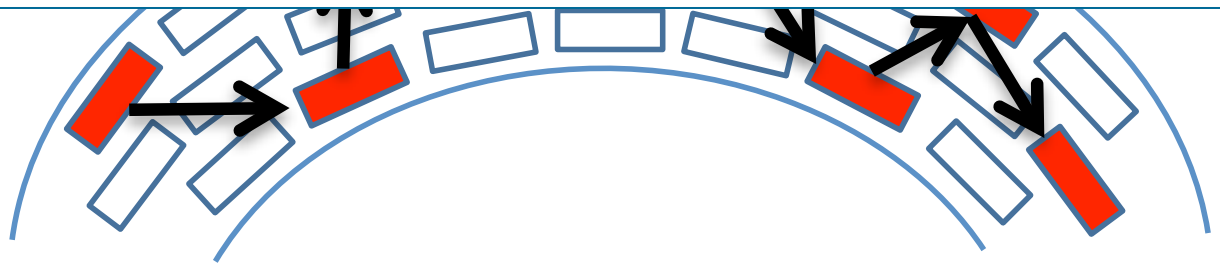
Inline Dedupe - Read Path challenges

Inherently, dedupe causes disk-level fragmentation !

- *Sequential reads turn random => more seeks => more latency*
- RPC based protocols (CIFS/NFS/iSCI) are latency sensitive

Primary workloads are typically read-intensive

- Usually the Read/Write ratio is ~70/30
- Inline dedupe must not affect read performance !

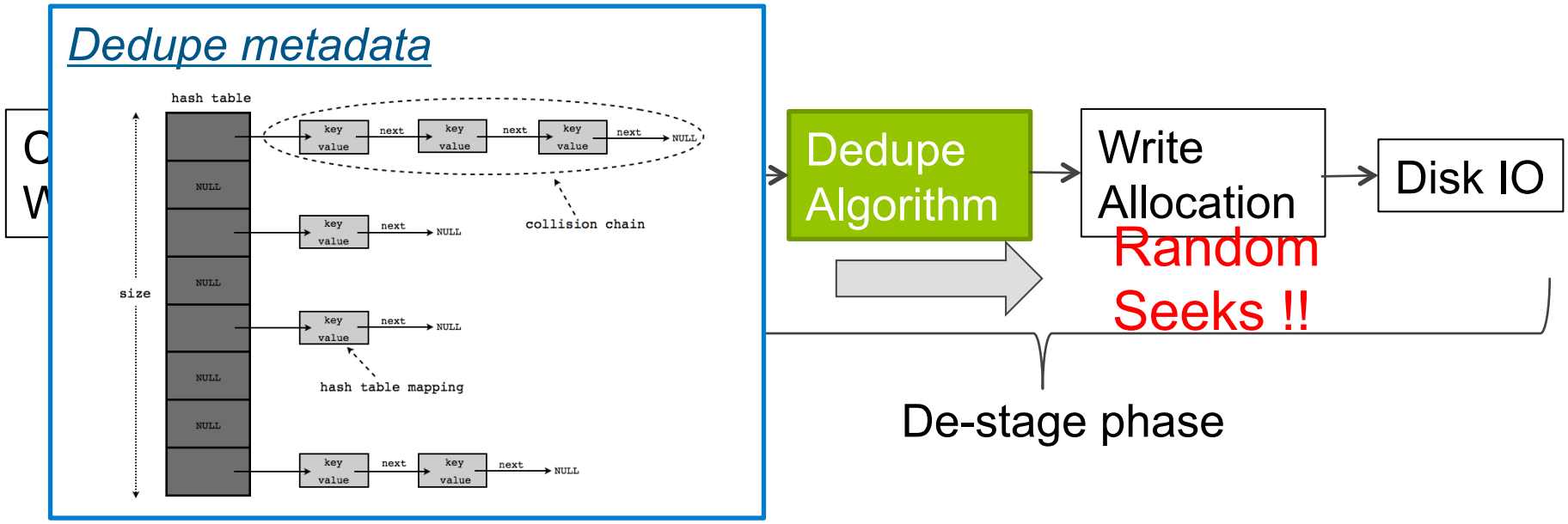


Fragmentation with random seeks

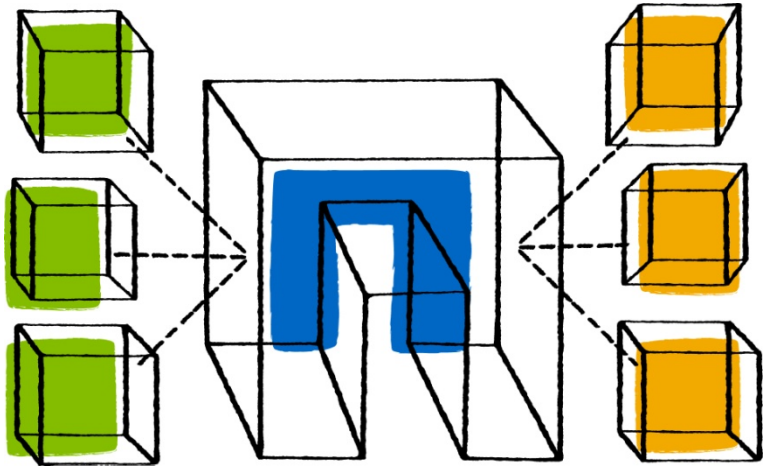
Inline Dedupe – Write Path Challenges

Extra random I/Os in the write path due to dedupe algorithm

- Dedupe metadata (Finger Print DB) lookups and updates
- Updating the reference counts of blocks on disk



Our Approach



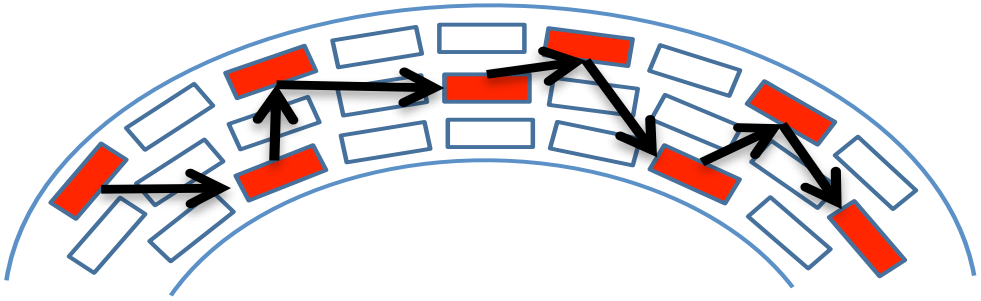
iDedup – Intuition

Is there a good tradeoff between capacity savings and latency performance?

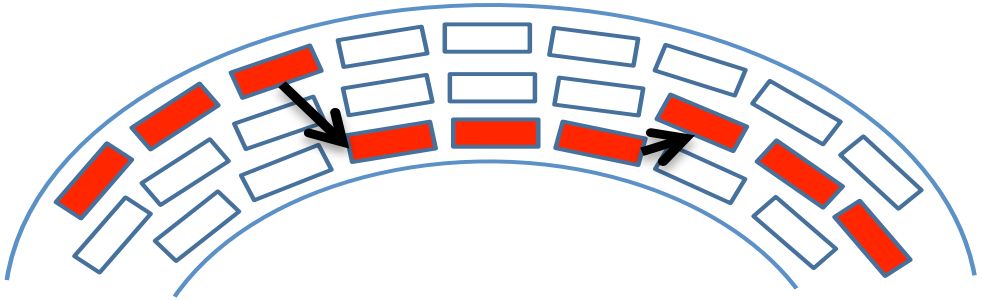
iDedup – Solution to Read Path issues

Insight 1: Dedupe only sequences of disk blocks

- Solves fragmentation => amortized seeks during reads
- Selective dedupe, leverages *spatial locality*
- Configurable minimum sequence length



Fragmentation with random seeks



Sequences, with amortized seeks

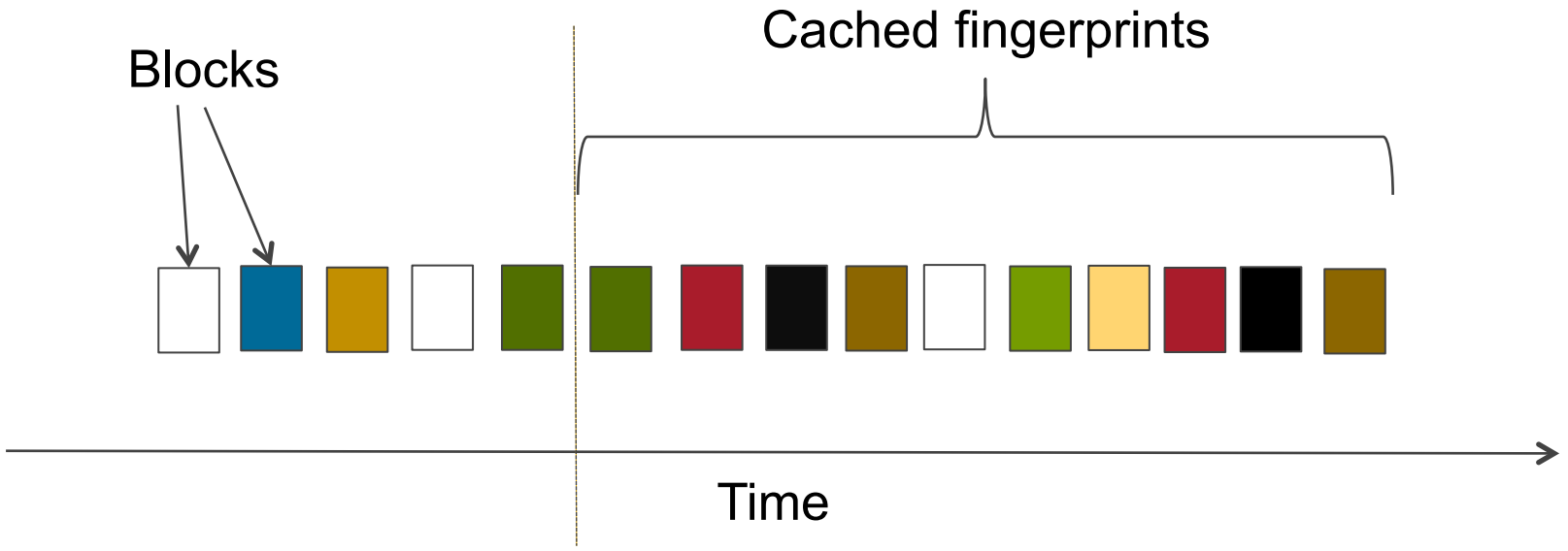
iDedup – Write Path issues

How can we reduce dedupe metadata I/Os?
Flash(?) - read I/Os are cheap, but frequent updates are expensive

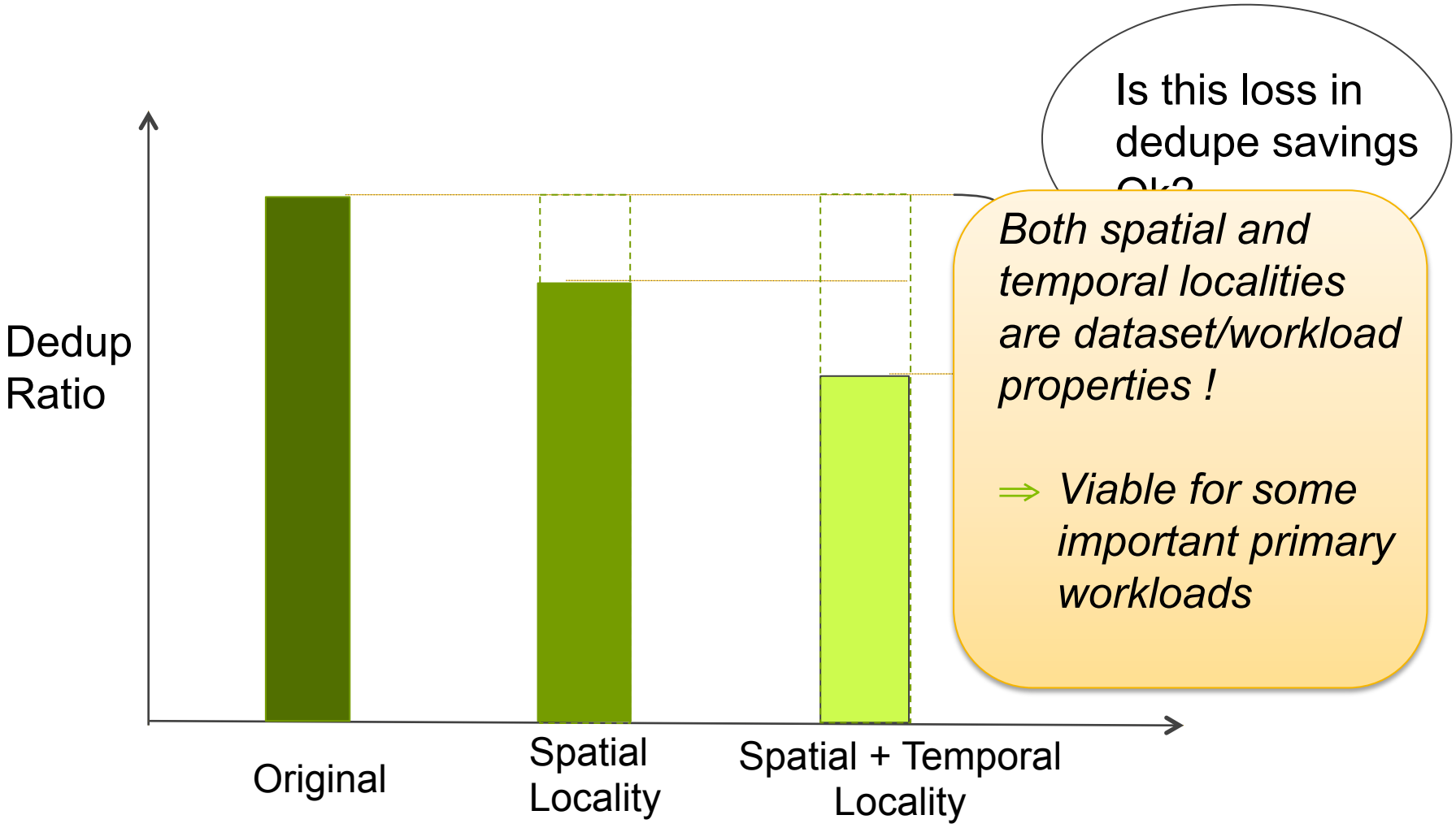
iDedup – Solution to Write Path issues

Insight 2: Keep a smaller dedupe metadata as an in-memory cache

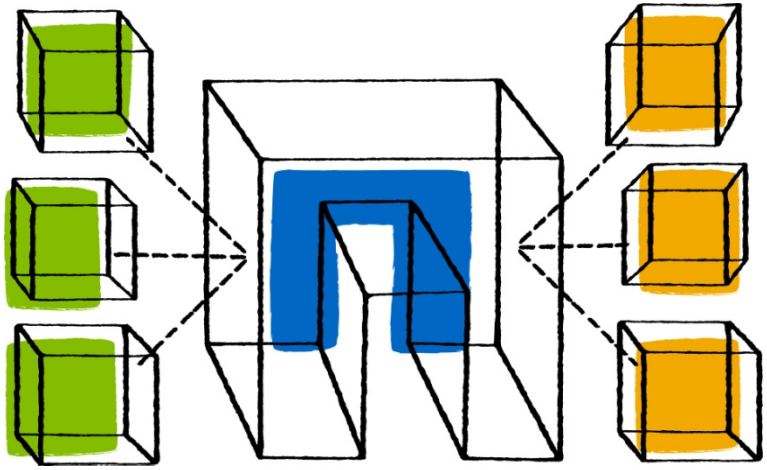
- No extra IOs
- Leverages *temporal locality* characteristics in duplication
- Some loss in dedupe (a subset of blocks are used)



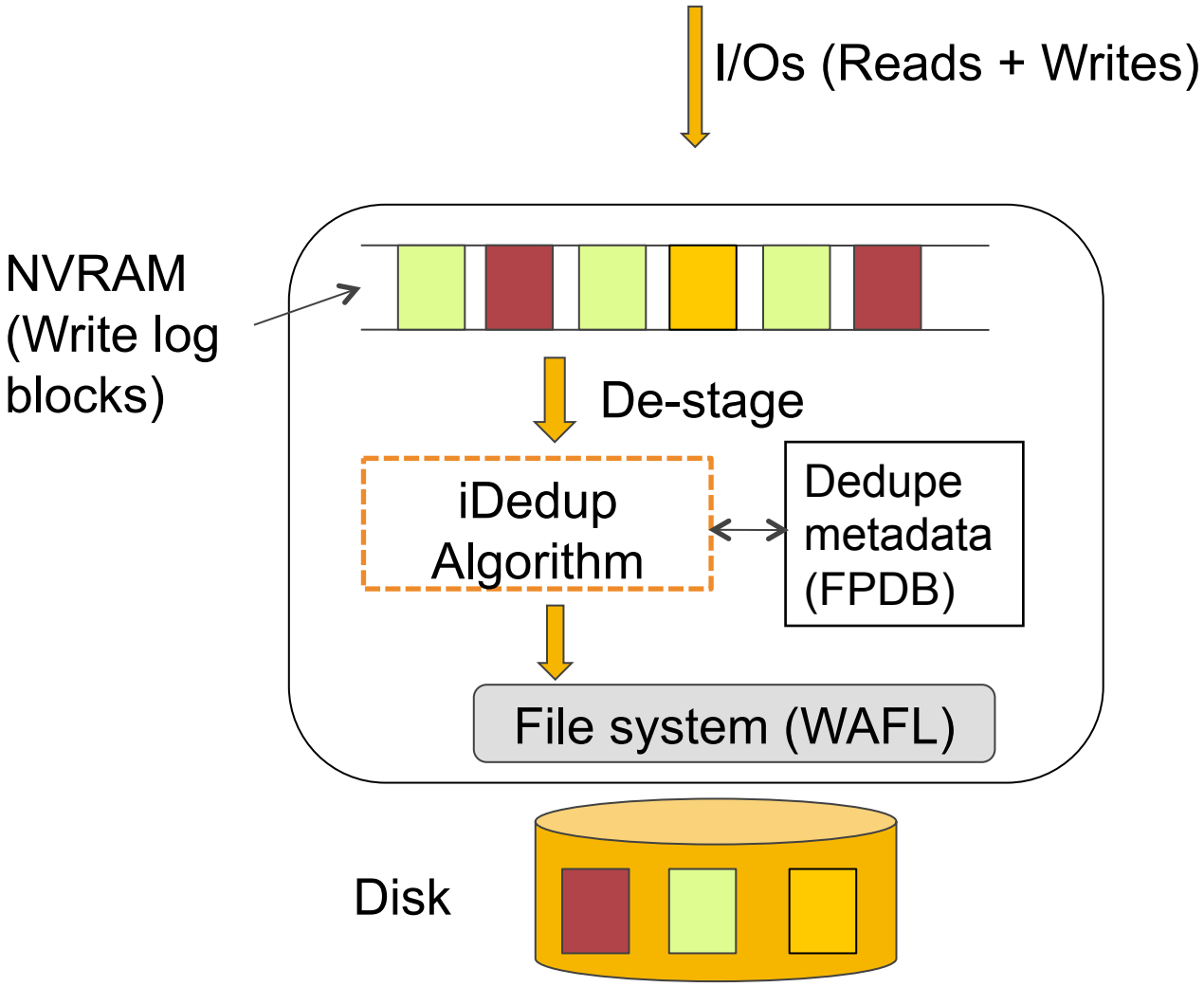
iDedup - Viability



Design and Implementation



iDedup Architecture



iDedup - Two key tunable parameters

Minimum sequence length – Threshold

- Minimum number of sequential duplicate blocks on disk
- Dataset property => ideally set to expected fragmentation
- Different from larger block size – variable vs fixed
- Knob between performance (fragmentation) and dedupe

Dedupe metadata (Fingerprint DB) cache size

- Workload's working set property
- Increase in cache size => decrease in buffer cache
- Knob between performance (cache hit ratio) and dedupe

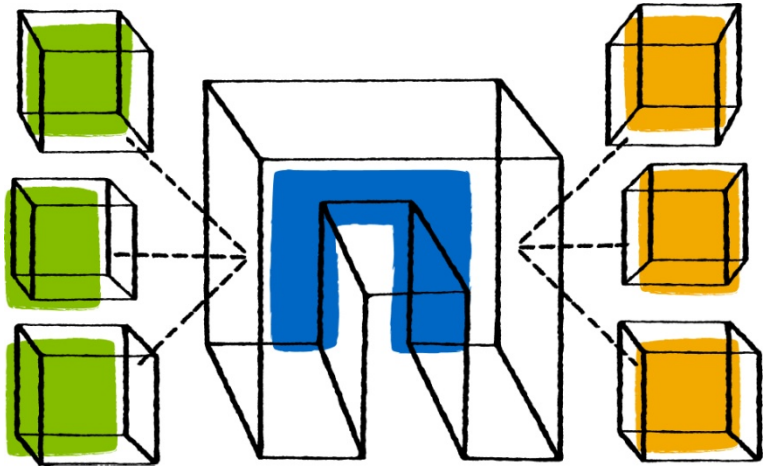
iDedup Algorithm

The iDedup algorithm works in 4 phases for every file:

- Phase 1 (per file):Identify blocks for iDedup
  - Only full, pure data blocks are processed
  - Metadata blocks, special files ignored
- Phase 2 (per file) : Sequence processing
  - Uses the dedupe metadata cache
  - Keeps track of multiple sequences
- Phase 3 (per sequence): Sequence pruning
  - Eliminate short sequences below threshold
  - Pick among overlapping sequences via a heuristic
- Phase 4 (per sequence): Deduplication of sequence

NetApp

Evaluation

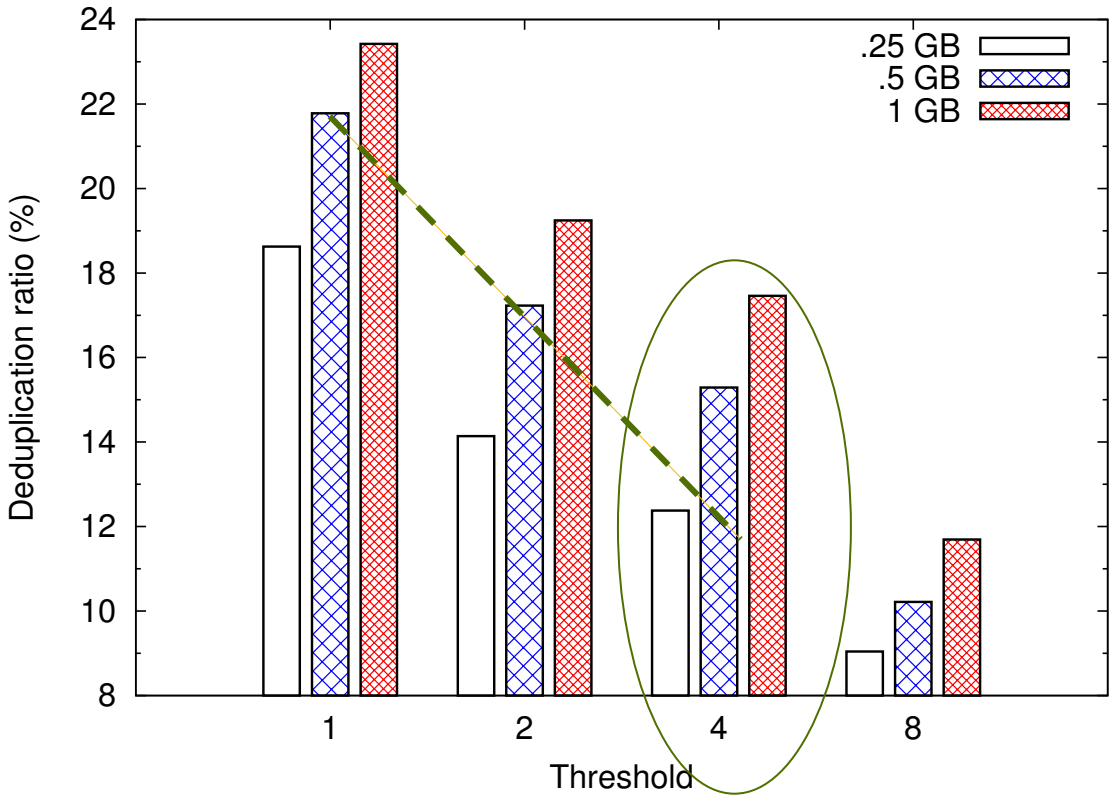


Evaluation Setup

- NetApp FAS 3070, 8GB RAM, 3 disk RAID0
- Evaluated by replaying real-world CIFS traces
  - Corporate filer traces in NetApp DC (2007)
    - Read data: 204GB (69%), Write data: 93GB
  - Engineering filer traces in NetApp DC (2007)
    - Read data: 192GB (67%), Write data: 92GB
- Comparison points
  - Baseline: System with no iDedup
  - Threshold-1: System with full dedupe (1 block)
- Dedupe metadata cache: 0.25, 0.5 & 1GB

Results: Deduplication ratio vs Threshold

Dedupe ratio vs Thresholds, Cache sizes (Corp)



Less than linear decrease in dedupe savings

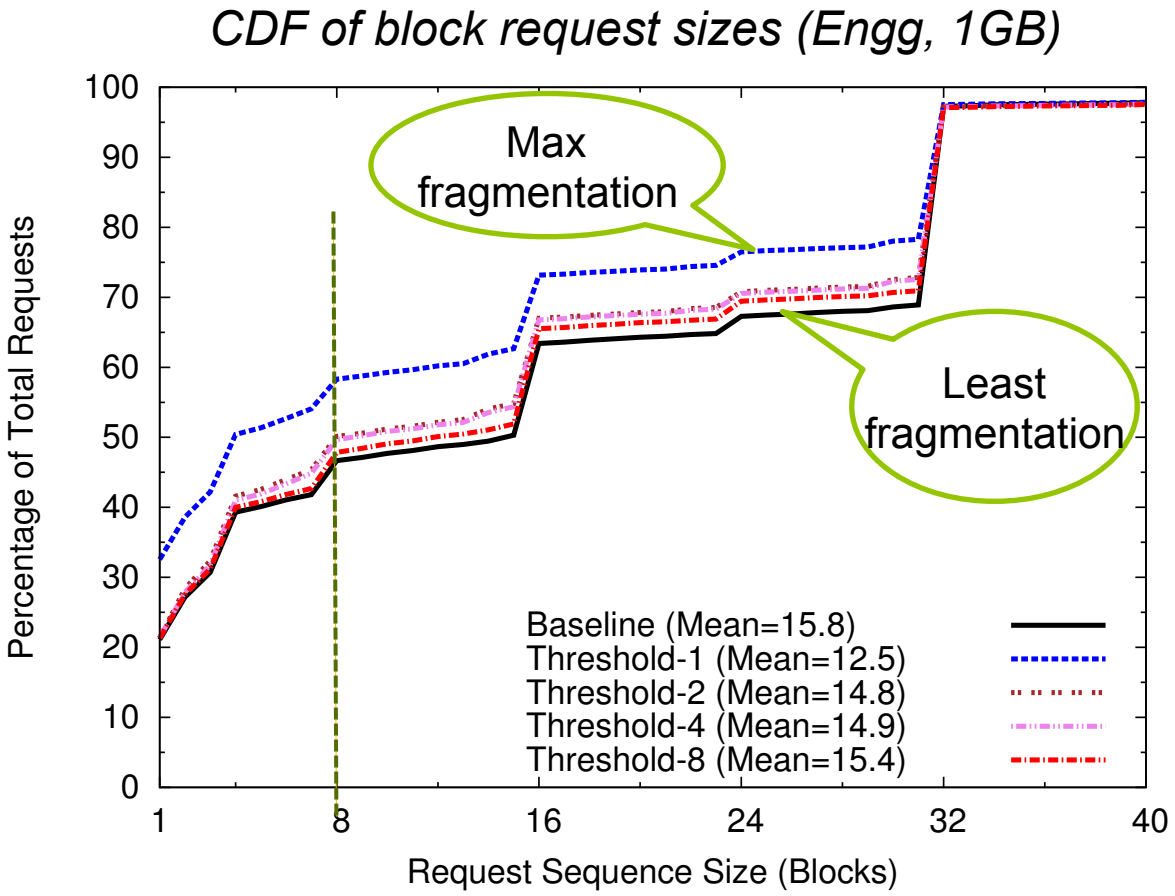
⇒ *Spatial locality in dedupe*

Ideal Threshold = Biggest threshold with least decrease in dedupe savings

⇒ *Threshold-4*

⇒ *~60% of max*

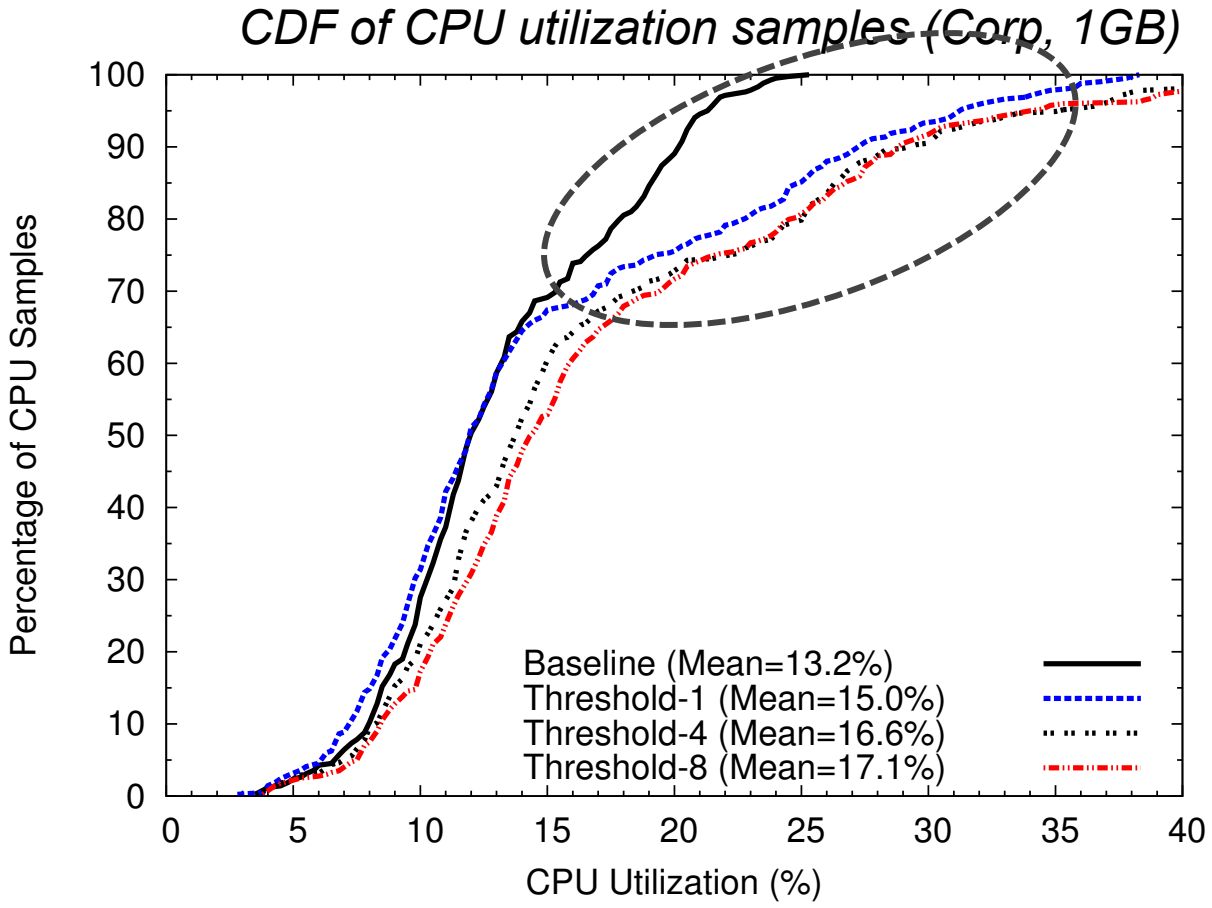
Results: Disk Fragmentation (req sizes)



Fragmentation for other thresholds are between Baseline and Thresh-1

⇒ *Tunable fragmentation*

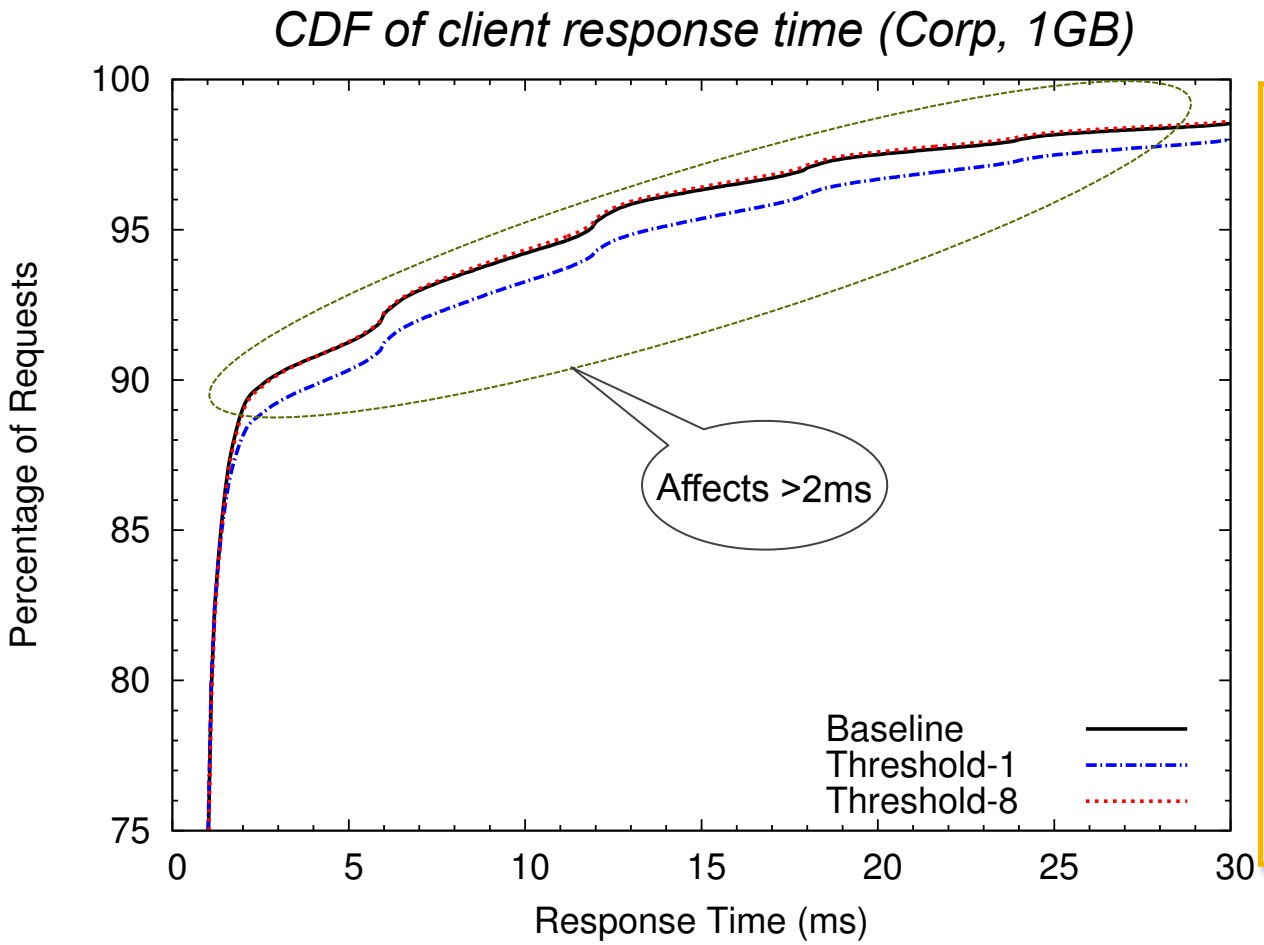
Results: CPU Utilization



Larger variance (long tail) compared to baseline

⇒ *But, mean difference is less than 4%*

Results: Latency Impact



Latency impact for longer response times (> 2ms)

⇒ *Thresh-1 mean latency affected by ~13% vs baseline*

⇒ *Diff between Thresh-8 and baseline <4%!*

Summary

- Inline dedupe has significant performance challenges
  - Reads : Fragmentation, Writes: CPU + Extra I/Os
- iDedup creates tradeoffs b/n savings and performance
  - Leverage dedupe locality properties
  - Avoid fragmentation – dedupe only sequences
  - Avoid extra I/Os – keep dedupe metadata in memory
- Experiments for latency-sensitive primary workloads
  - Low CPU impact – < 5% on the average
  - ~60% of max dedupe, ~4% impact on latency
- Future work: Dynamic threshold, more traces
- Our traces are available for research purposes

Acknowledgements

- NetApp WAFL Team
  - Blake Lewis, Ling Zheng, Craig Johnston, Subbu PVS, Praveen K, Sriram Venketaraman
- NetApp ATG Team
  - Scott Dawkins, Jeff Heller
- Shepherd
  - John Bent
- Our Intern (from UCSC)
  - Stephanie Jones

Thank you