Tectonic-Shift: A Composite Storage Fabric for Large-Scale ML Training

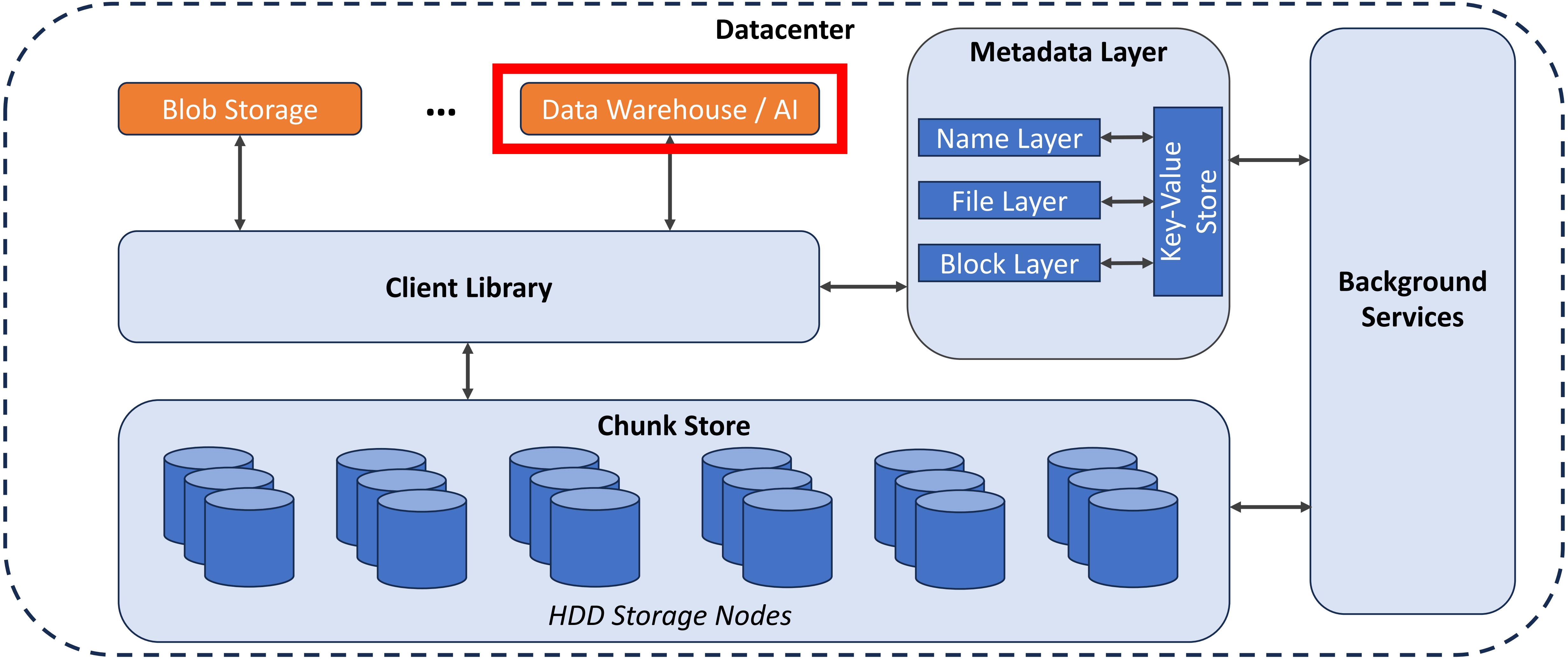
Mark Zhao, Satadru Pan, Niket Agarwal, Zhaoduo Wen, David Xu, Anand Natarajan, Pavan Kumar, Shiva Shankar P, Ritesh Tijoriwala, Karan Asher, Hao Wu, Aarti Basant, Daniel Ford, Delia David, Nezih Yigitbasi, Pratap Singh, Carole-Jean Wu, and Christos Kozyrakis

Stanford University, Meta

2023 USENIX Annual Technical Conference

Stanford MAST

Tectonic Filesystem: Meta's storage foundation

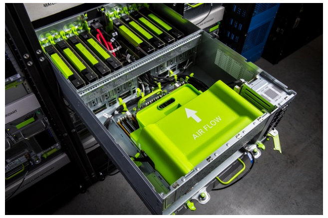


S. Pan, et al., *Facebook's Tectonic Filesystem: Efficiency from Exascale, FAST'21*

ML infrastructure scaling trends

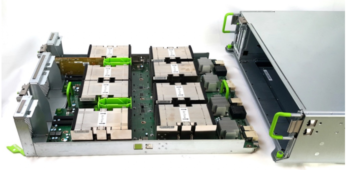
Training larger and more complex models (e.g., LLMs, DLRMs) requires…

Scale-Up Infrastructure



Big Sur

2016



Big Basin

2017

Big Basin v2

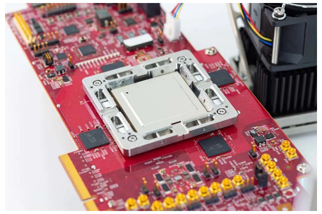
2018

Zion / ZionEX

2019-2021

Grand Teton

2022



MTIA

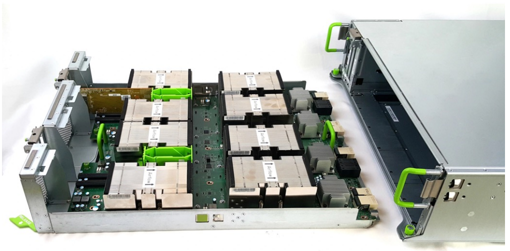
2023+

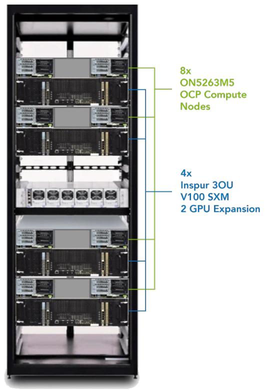
CPUs → **GPUs / ASICs**

ML infrastructure scaling trends

Training larger and more complex models (e.g., LLMs, DLRMs) requires…

Scale-Out Infrastructure





Node-Scale

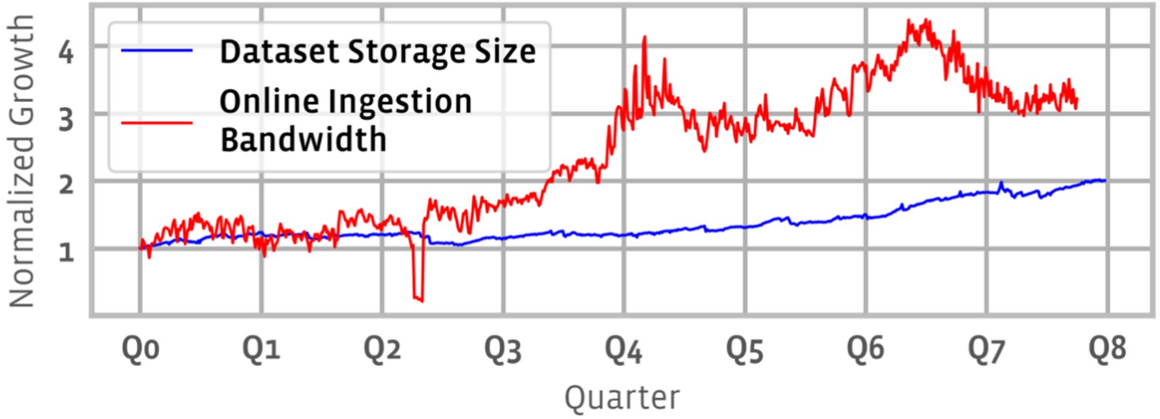
Rack-Scale

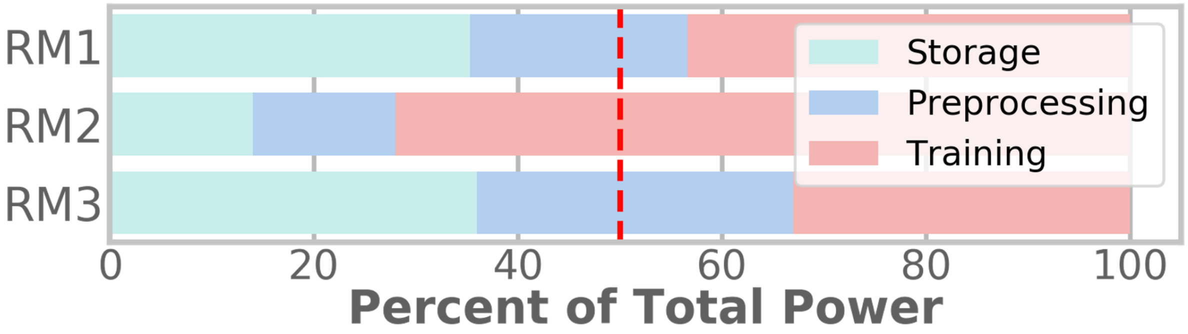
Warehouse-Scale

Single-Node Trainers → **Datacenter-Scale Training Clusters**

ML infrastructure needs IOPS scaling

Result: A *massive* growth in IOPS demand for ML training datasets





How do we scale Tectonic to meet exploding IOPS demands?

M. Zhao, et al., Understanding Data Storage and Ingestion for Large-Scale Deep Recommendation Model Training, ISCA'22

Hardware design space exploration

Need to provision storage fabric with *both* sufficient **storage** and **IOPS** capacity

Hardware design space exploration

Option 1: Scale Tectonic's HDD Chunk Store

	Storage Power	IOPS Power	Storage & IOPS Power
HDD Cluster	1.00	9.92	9.92

IOPS-Bound

Provision enough HDDs to meet IOPS demand

Hardware design space exploration

Option 2: Place ML datasets in flash

IOPS-Bound

	Storage Power	IOPS Power	Storage & IOPS Power
HDD Cluster	1.00	9.92	9.92
Flash Cluster	6.53	1.88	6.53

Storage-Bound

Provision enough SSDs to store datasets

Hardware design space exploration

Option 3: Composite storage

IOPS-Bound → HDD Cluster

Storage-Bound → Flash Cluster

	Storage Power	**IOPS Power**	**Storage & IOPS Power**
HDD Cluster	1.00	9.92	9.92
Flash Cluster	6.53	1.88	6.53
HDD + Flash Cluster	1.00	1.88	**2.69**

HDDs: Storage Efficient

Provision enough HDDs to store datasets

Hardware design space exploration

Option 3: Composite storage

	Storage Power	IOPS Power	Storage & IOPS Power
HDD Cluster	1.00	9.92	9.92
Flash Cluster	6.53	1.88	6.53
HDD + Flash Cluster	1.00	1.88	**2.69**

IOPS-Bound → HDD Cluster

Storage-Bound → Flash Cluster

Flash: IOPS Efficient

Provision SSDs to serve IOPS

Hardware design space exploration

Option 3: Composite storage

	Storage Power	IOPS Power	Storage & IOPS Power
HDD Cluster	1.00	9.92	9.92
Flash Cluster	6.53	1.88	6.53
HDD + Flash Cluster	1.00	1.88	**2.69**

IOPS-Bound → HDD Cluster

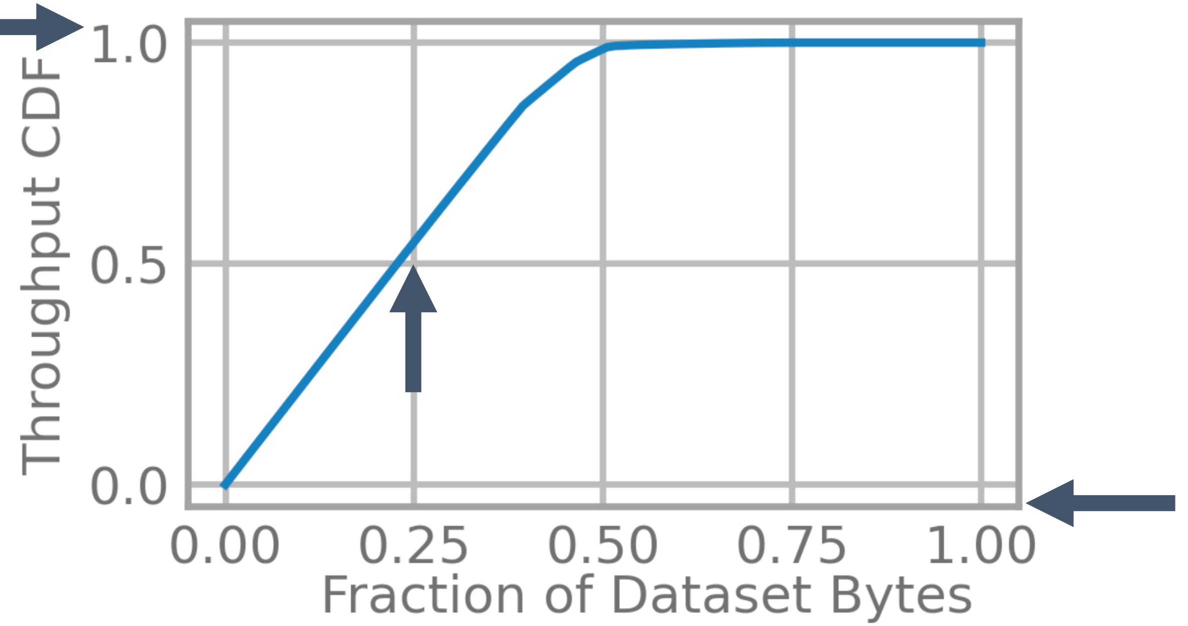
Storage-Bound → Flash Cluster

Composite Storage
Provision HDDs for storage, flash for IOPS not covered by HDDs

Software design space exploration

Goal: *Build a flash tier that **absorbs read IOPS** without storing the entire dataset.*

...contribute to y% total storage IOPS

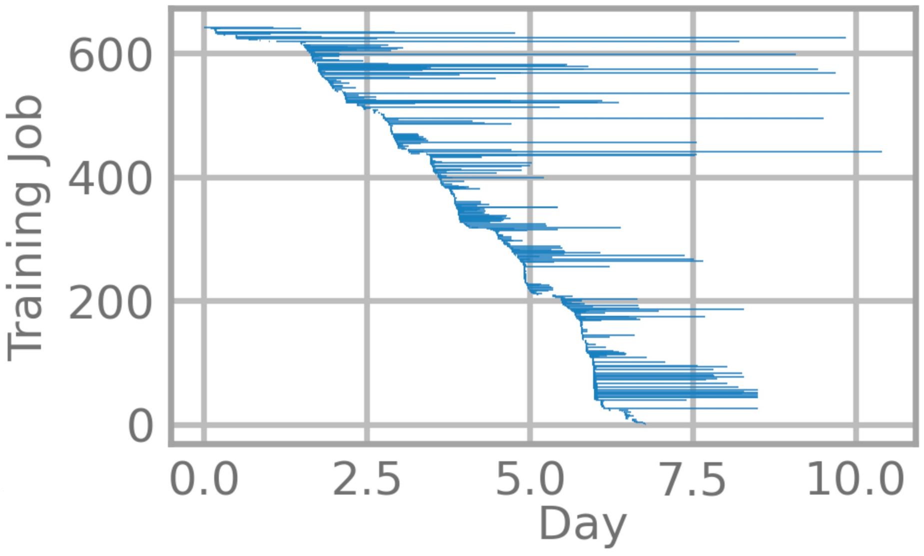


x% of stored bytes

Challenge: While our ML workloads exhibit skewed popularity, *current caches are ineffective at capturing their data reuse*.

Why current caches will not work

- ML jobs present challenging cache patterns
  - Scans: Large O(10-100PB), long-running single-epoch reads
  - Churn: data reuse *across* massive, asynchronous multi-tenant jobs
- *General-purpose LRU caches thrash*
- *ML caches focus on data reuse within multi-epoch jobs and single-tenant environments*




```
>>> dataset = tf.data.Dataset.range(5)
>>> dataset = dataset.map(lambda x: x**2)
>>> dataset = dataset.cache()
>>> # The first time reading through the data will generate the data using
>>> # `range` and `map`.
>>> list(dataset.as_numpy_iterator())
[0, 1, 4, 9, 16]
>>> # Subsequent iterations read from the cache.
>>> list(dataset.as_numpy_iterator())
[0, 1, 4, 9, 16]
```

D. G. Murray, et al., tf.data: A Machine Learning Data Processing Framework, VLDB vol. 14

Why current caches will not work

- ML jobs present challenging cache patterns
  - Scans: Large O(10-100PB), long-running single-epoch reads
  - Churn: data reuse *across* massive, asynchronous multi-tenant jobs
- *General-purpose LRU caches thrash*
- *ML caches focus on data reuse within multi-epoch jobs and single-tenant environments*

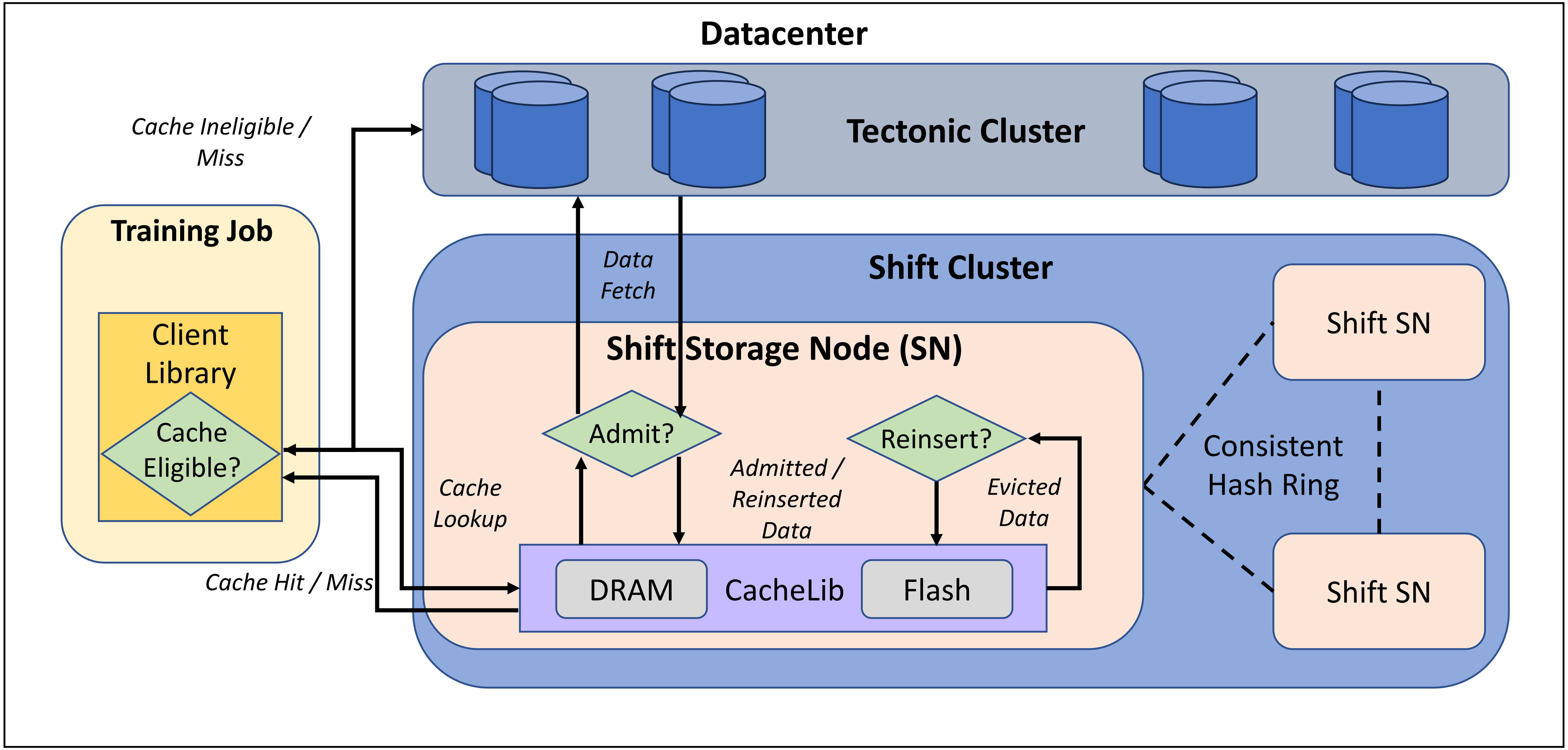
Need for a flash storage tier designed for industrial ML workloads.

Shift: A transparent, application-aware flash tier

A disaggregated flash storage tier that is…

- Transparent to end users
  - Exposes *Tectonic* API and semantics used across Meta
- Application-aware
  - Maximizes IOPS absorption using application metadata
- Simple
  - Builds upon *Tectonic*'s Metadata Layer and CacheLib
- Scalable and Fault Tolerant
  - Decentralized, DHT-based architecture

Tectonic-Shift: Meta's ML storage fabric

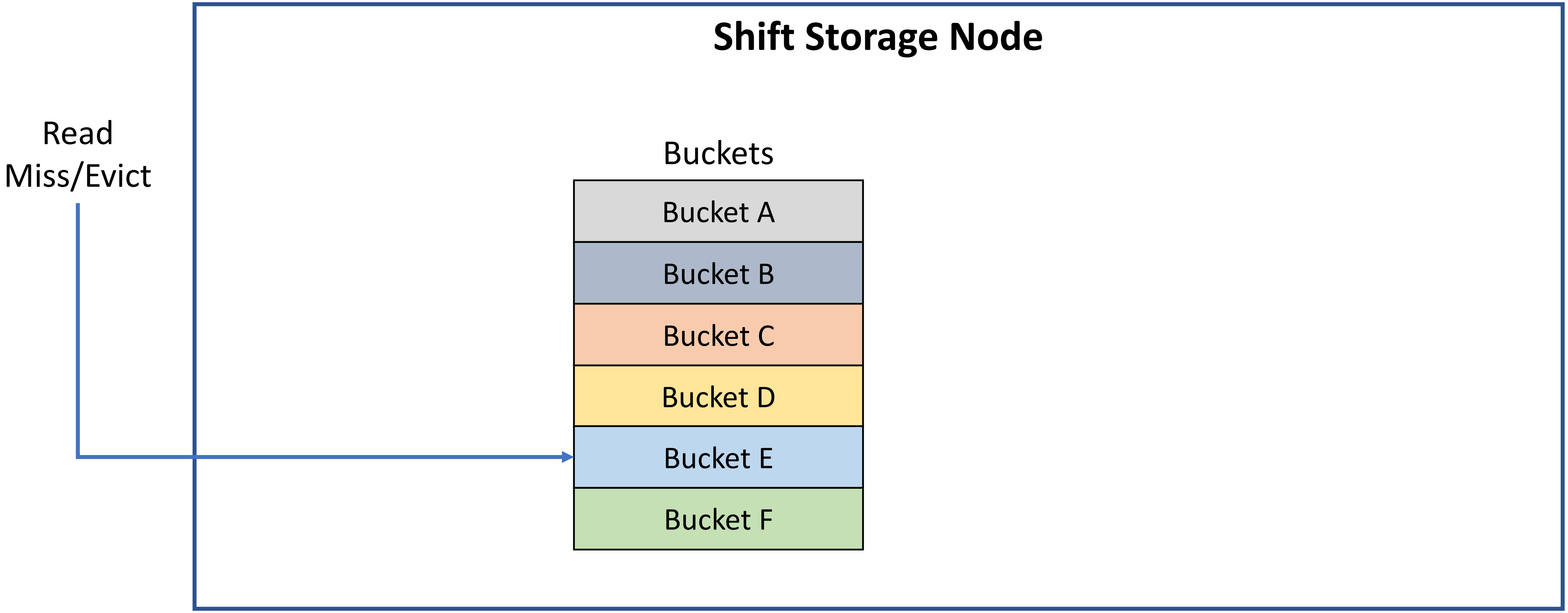


Absorbing IOPS with intelligent *Shift* policies

Each *Shift* Storage Node implements cache policies *on top of CacheLib* to **maximize absorbed IOPS:**

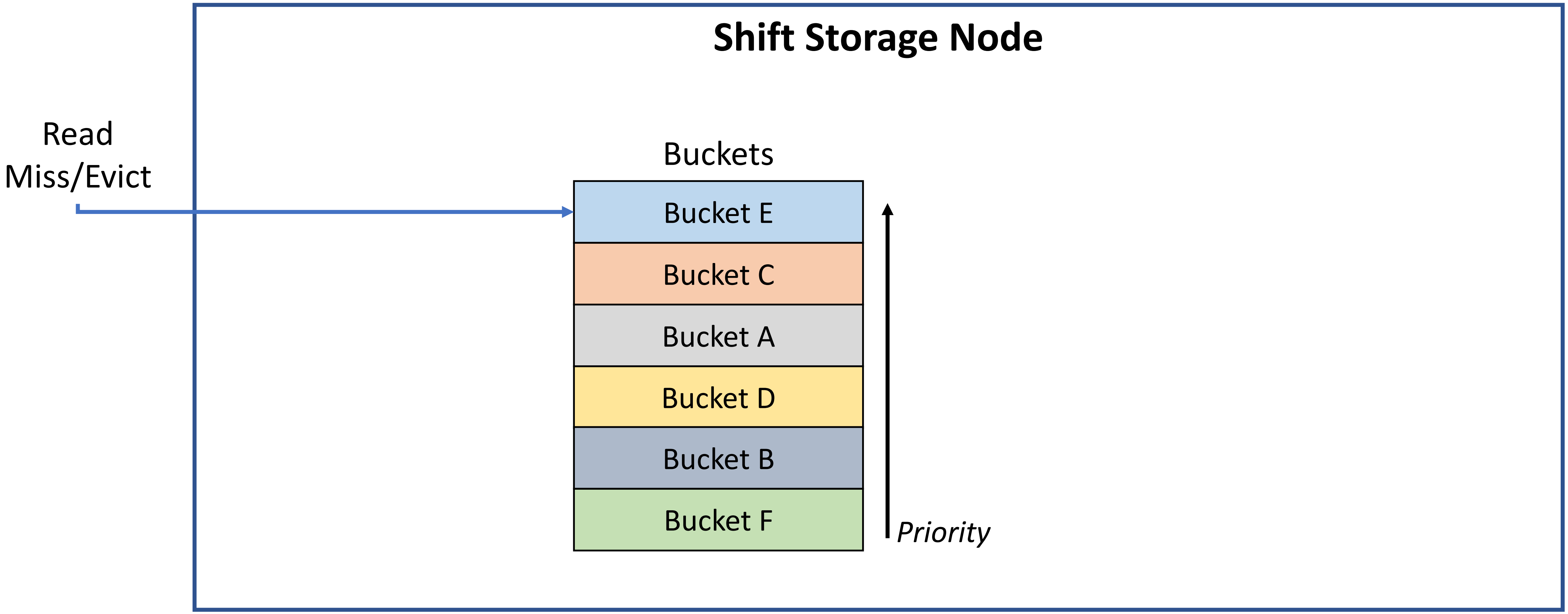
Absorbing IOPS with intelligent *Shift* policies

1. Group similar accesses (e.g., table partition) to ***buckets***



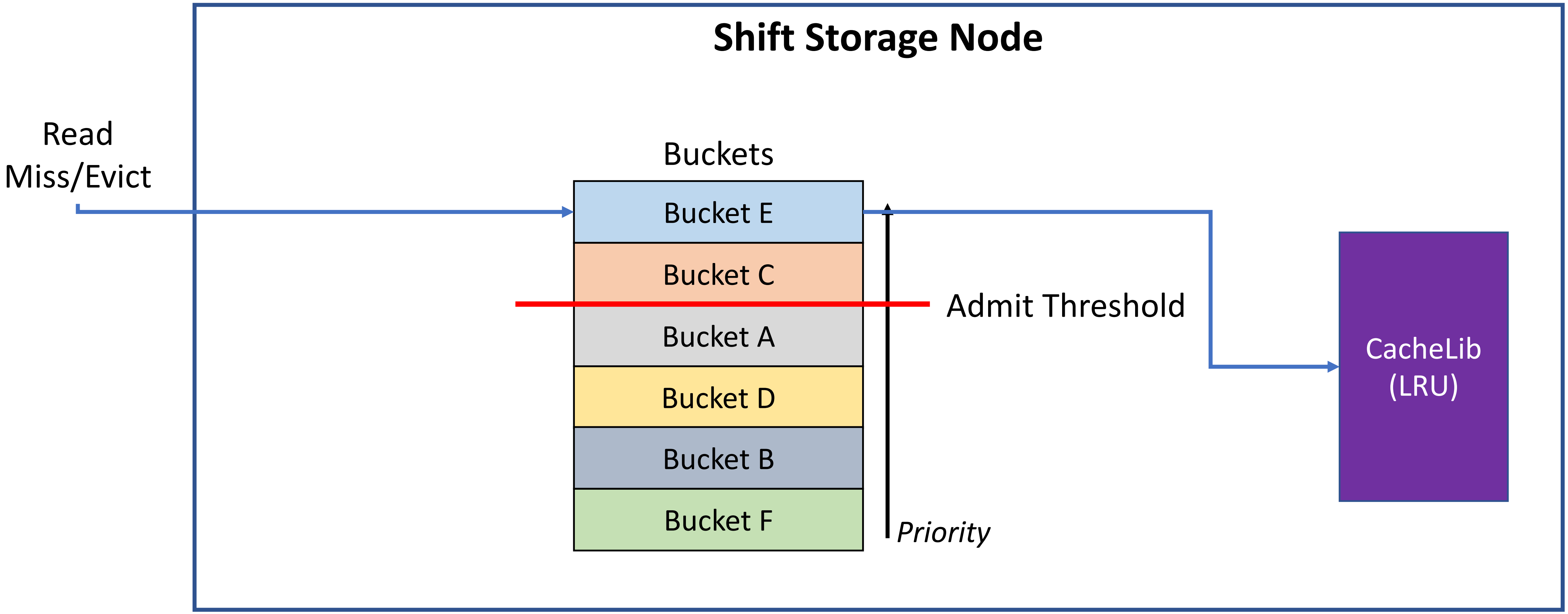
Absorbing IOPS with intelligent *Shift* policies

2. Prioritize buckets based on ***historic*** and ***derived future*** accesses



Absorbing IOPS with intelligent *Shift* policies

3. Admit buckets based on **threshold** to avoid thrashing and flash burn



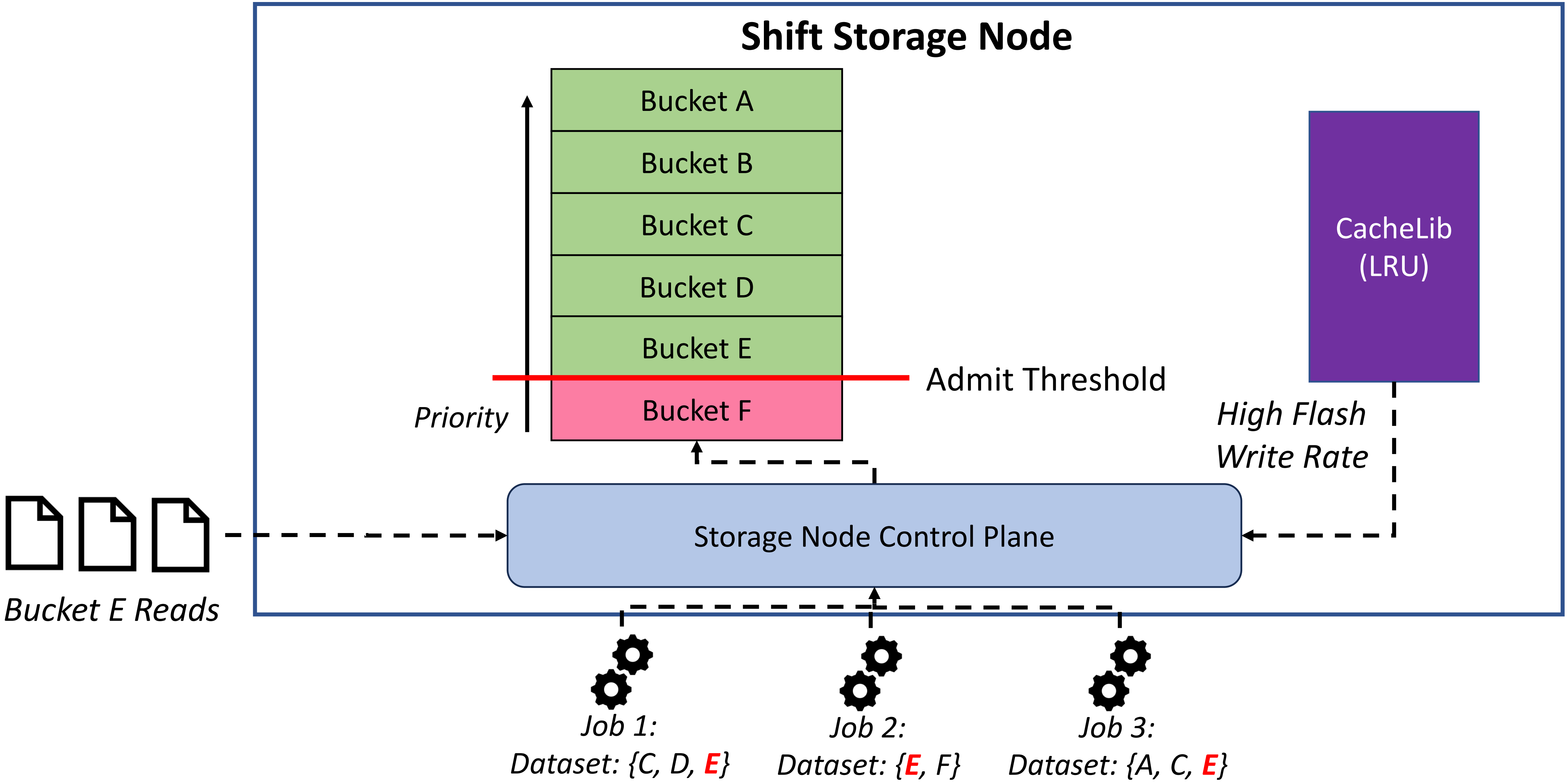
Bucket priorities: Predicting the future

Calculate bucket priorities based on...

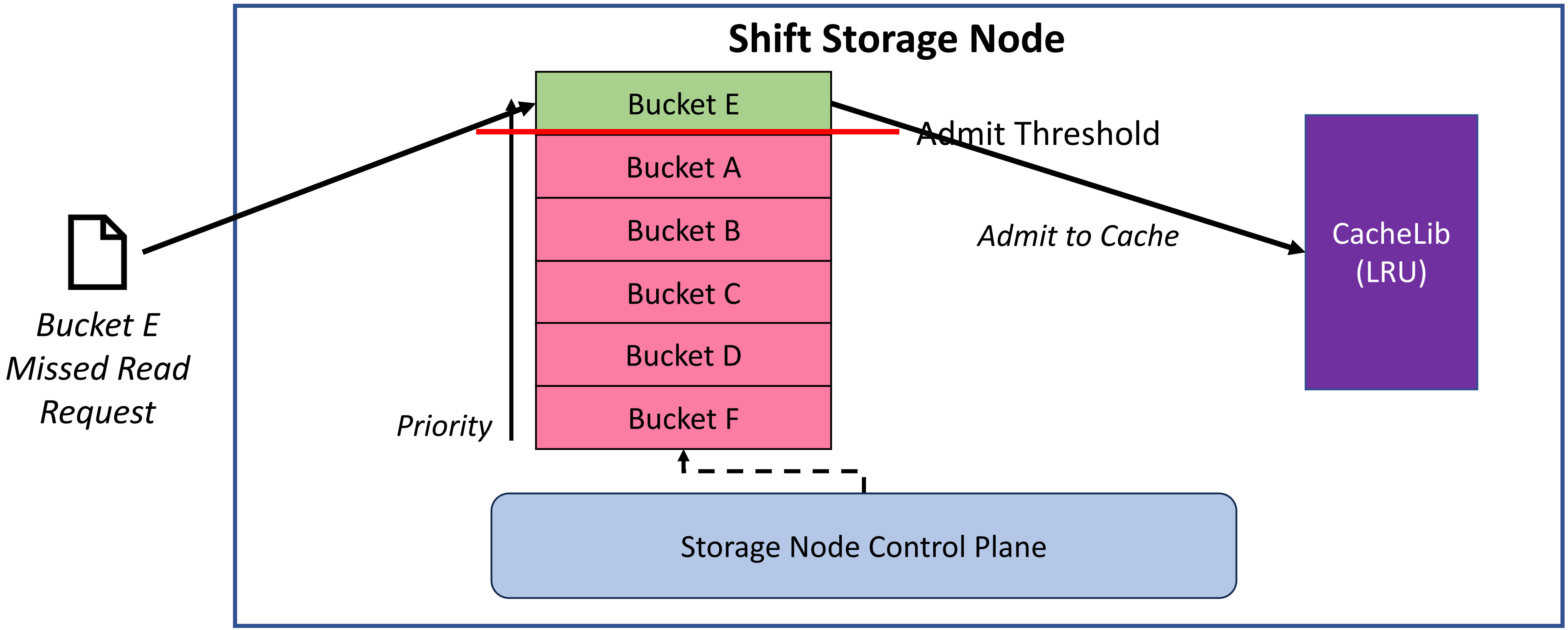
- Historic accesses
  - Log of recent per-bucket accesses
- Key insight: Future accesses
  - Derived from **dataset specifications**

```
class DLRMDataset(...):
  def __init__(self, table, rows, cols):
    ...
  def __iter__(self):
    # return iterator over table rows/cols
...

ds = DLRMDataset(
  table_t,
  [date_d, ...],
  [feature_f, ...]
)
loader = DataLoader(ds, ...) # DPP client

for sample in loader:
  # read sample from storage
  # train model
```

Dynamic priority and threshold tuning

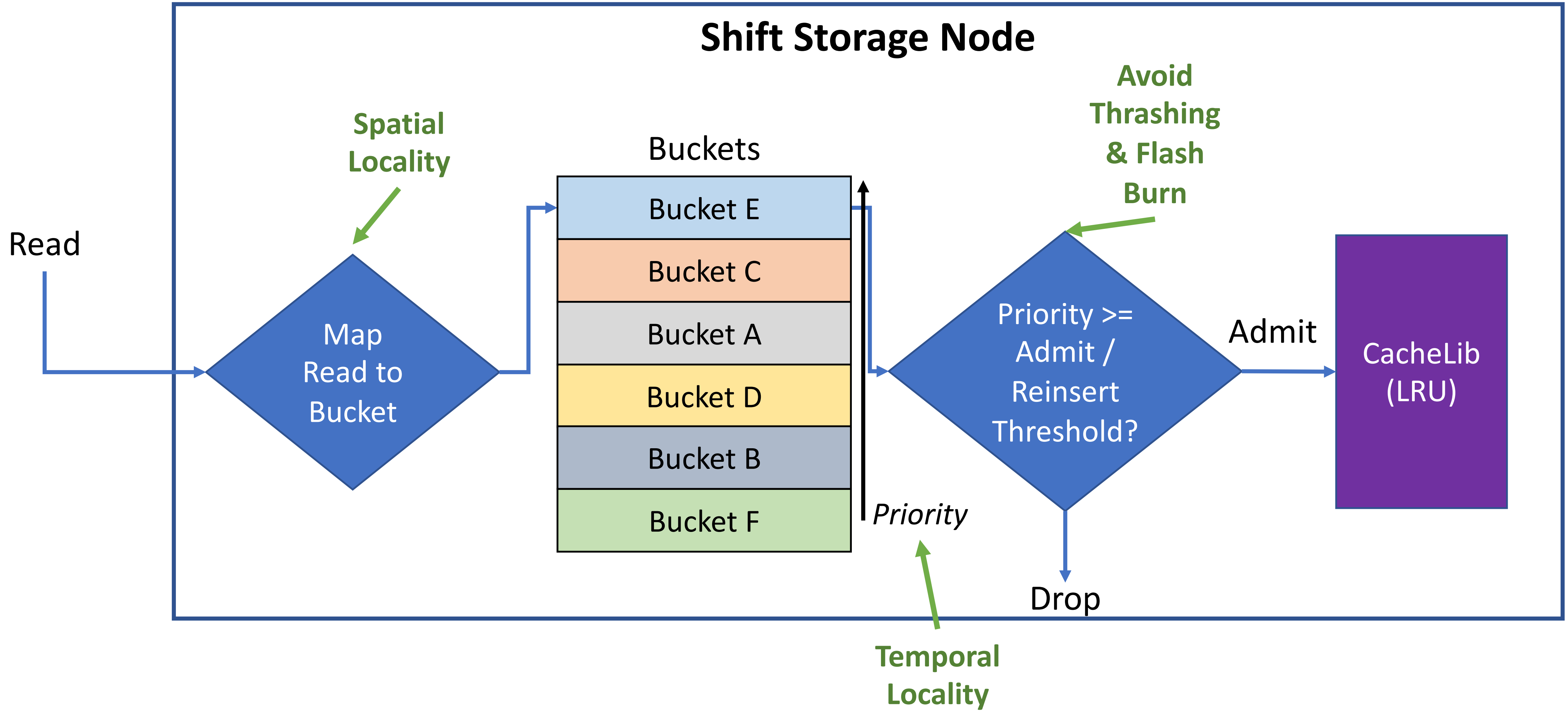


Dynamic priority and threshold tuning



Shift dynamically adjusts admission policies to keep high-priority data in cache, while minimizing thrashing and flash writes.

Putting it all together



Shift admission policies improve IOPS absorption

- Benchmark setup
  - Three production DLRM training workloads
  - 6-node *Shift* cluster
- Policy evaluation
  - CacheLib LRU, FIFO eviction only
  - Historic admission: bucket priority from recent accesses
  - Future admission: bucket priority from future accesses derived from Dataset
  - Historic & Future admission: bucket priority from max of Historic, Future

Average Normalized IO Absorption Across Benchmarks				
LRU Eviction	FIFO Eviction	Historic Admission + LRU Eviction	Future Admission + LRU Eviction	Historic & Future Admission + LRU Eviction
1.00	1.31	1.51	3.28	1.67

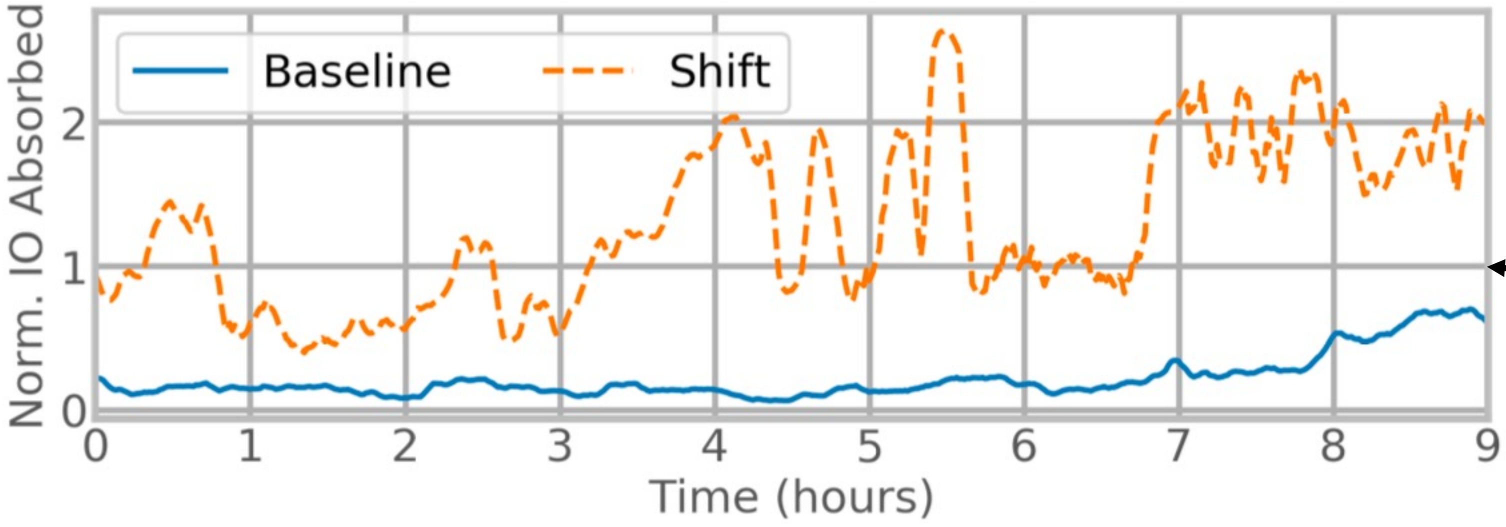
Shift admission policies manage flash endurance

- Need to limit flash write rates in production
  - Evaluation: 100 MB/s average write rate limit

Average IO Absorption & NVM Write Rate for *Synchronized* Workload						
	CacheLib Dynamic Admission	Reject First	Admit All	Historic Admission	Future Admission	Historic & Future Admission
IO Absorption (norm. to Dynamic)	1.00	1.51	2.66	2.14	3.07	2.99
NVM Write Rate (norm. to 100 MB/s limit)	0.96	8.39	22.05	1.01	1.01	1.00

Production deployment

Shift has been deployed across DCs at PB scale since early 2022, saving significant amounts storage infrastructure power.



Shift saves 29% power relative to only HDDs. Massive @ DC scale.

Conclusion

- Modern ML training clusters require massive storage IOPS.
- *Tectonic-Shift* meets IOPS demand by combining *Tectonic* with *Shift*, an IOPS-efficient flash storage tier.
- *Shift* maximizes absorbed IOPS (1.5-3.3x over LRU) using intelligent policies leveraging historic and derived future access patterns.
- *Tectonic-Shift* serves as Meta's ML storage fabric, improving storage efficiency (29% in our trace) across multiple datacenters.

myzhao@cs.stanford.edu