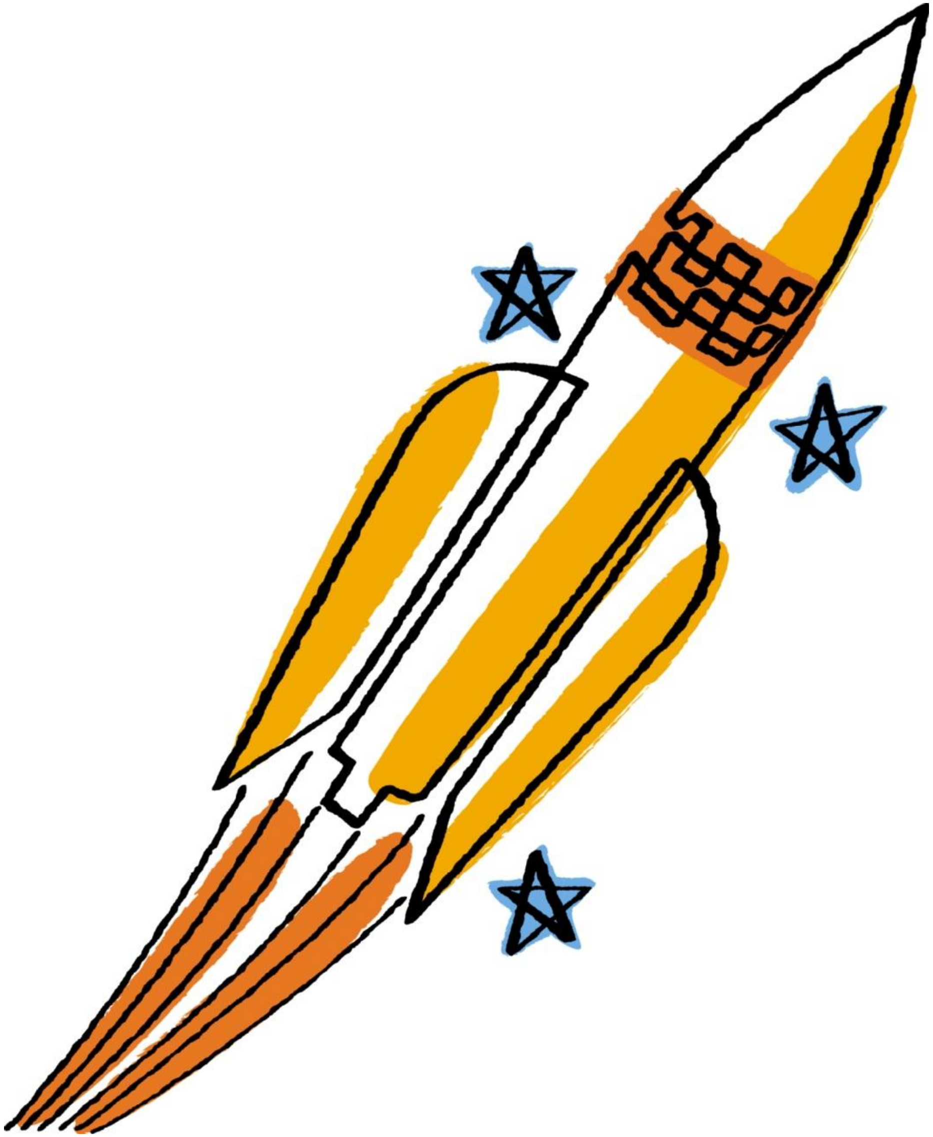
Go further, faster®

Mercury: Host-side Flash Caching for the Data Center

Steve Byan	**James Lentini**
Anshul Madan	Luis Pabón
Michael Condict	Jeff Kimmel
Steve Kleiman	Christopher Small
Mark Storer	

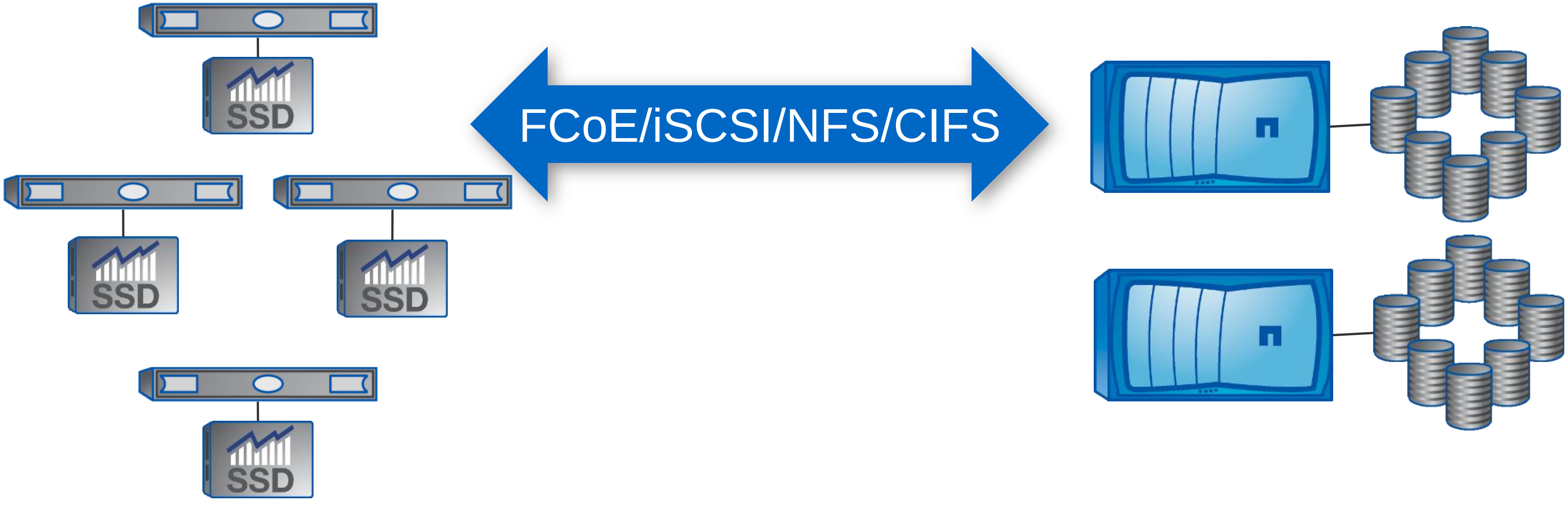
Advanced Technology Group
NetApp



28th IEEE Conference on Mass Storage Systems and Technologies (MSST)

April 20, 2012

Data Center with Flash SSDs



Shared Compute

Shared Storage

How do we make effective use of flash SSDs while preserving the benefits of shared storage?

Outline

- Part I: Architecture
- Part II: Design and Implementation
- Part III: Evaluation

Part I. Architecture

Four Architectural Goals

- Consistently High Performance
- Highly Available
- Correct and Consistent
- Simple Management Integration

Consistently High Performance

Goals

- Realize the low latency access
- Meet Service Level Objective (SLO) after restart

Consequences

- Direct-attached to host
- Persistent, preferably durable

Highly Available

Goal

- Never lose data in any situation

Consequence

- Write-through

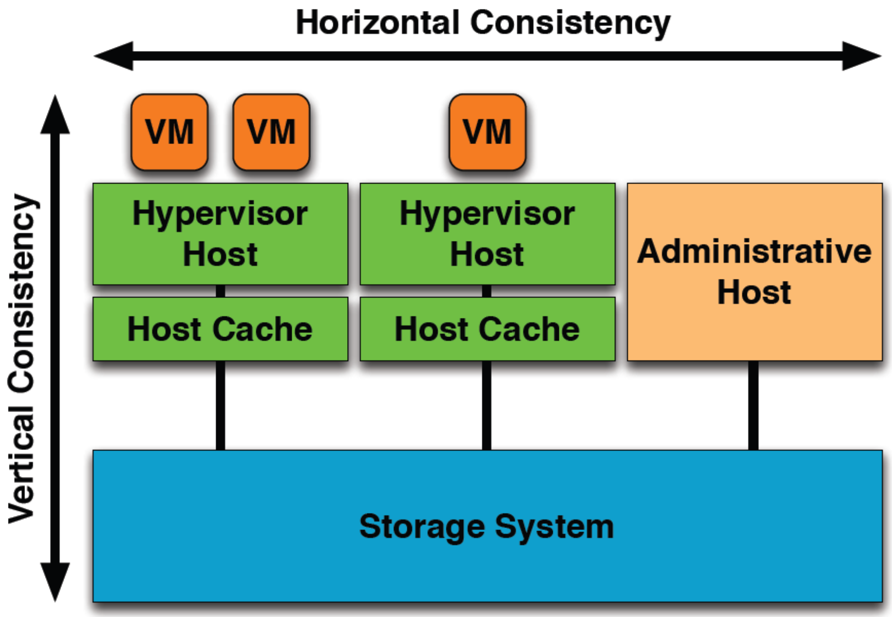
Correct and Consistent

Goals

- Consistency with storage array
- Consistent with peers

Consequences

- Cache non-shared objects
- Invalidate on migration, restore, etc.



Simple Management Integration

Goal

- Simple and transparent management

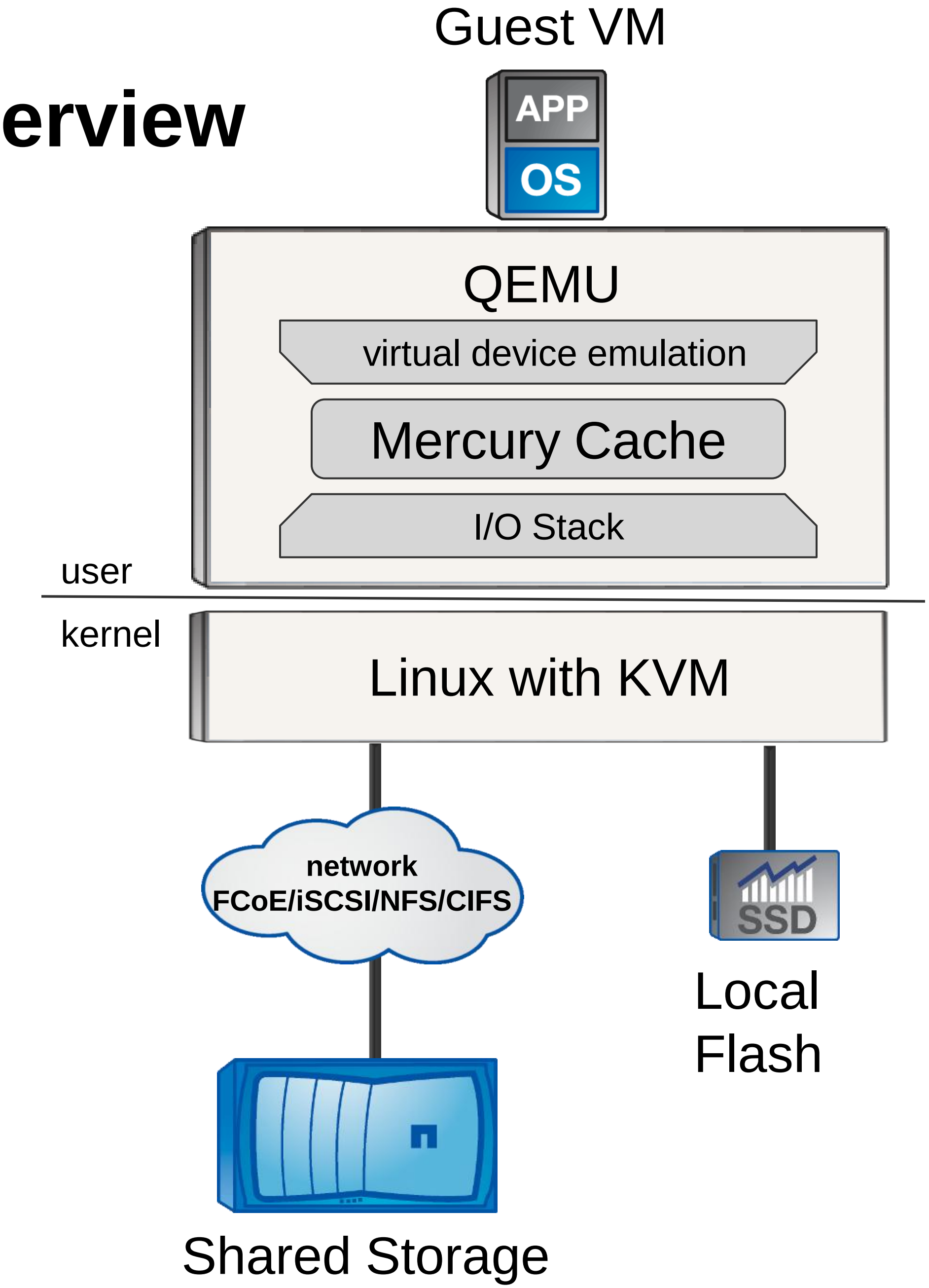
Consequence

- Hypervisor integration

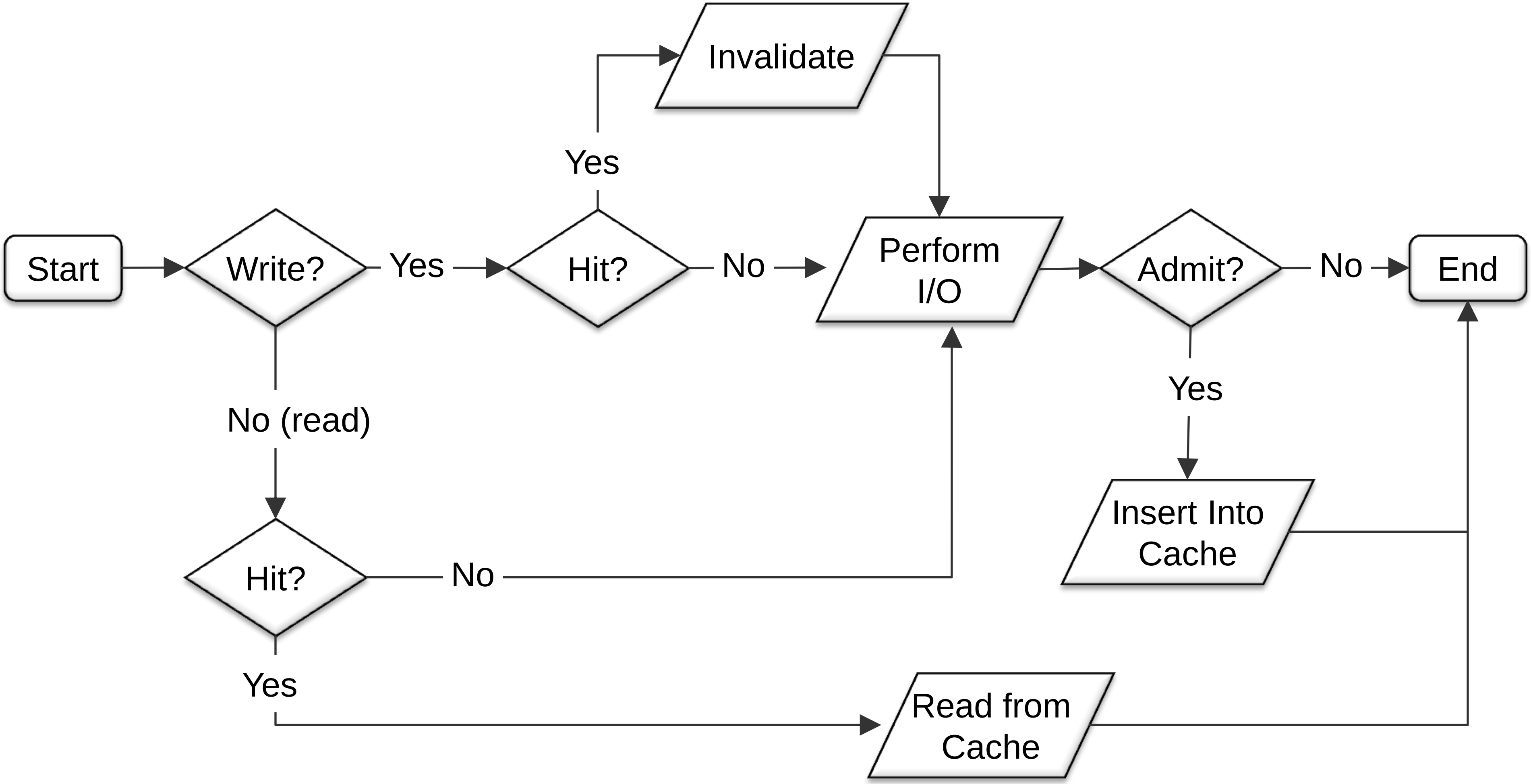
Part II. Design and Implementation

Implementation Overview

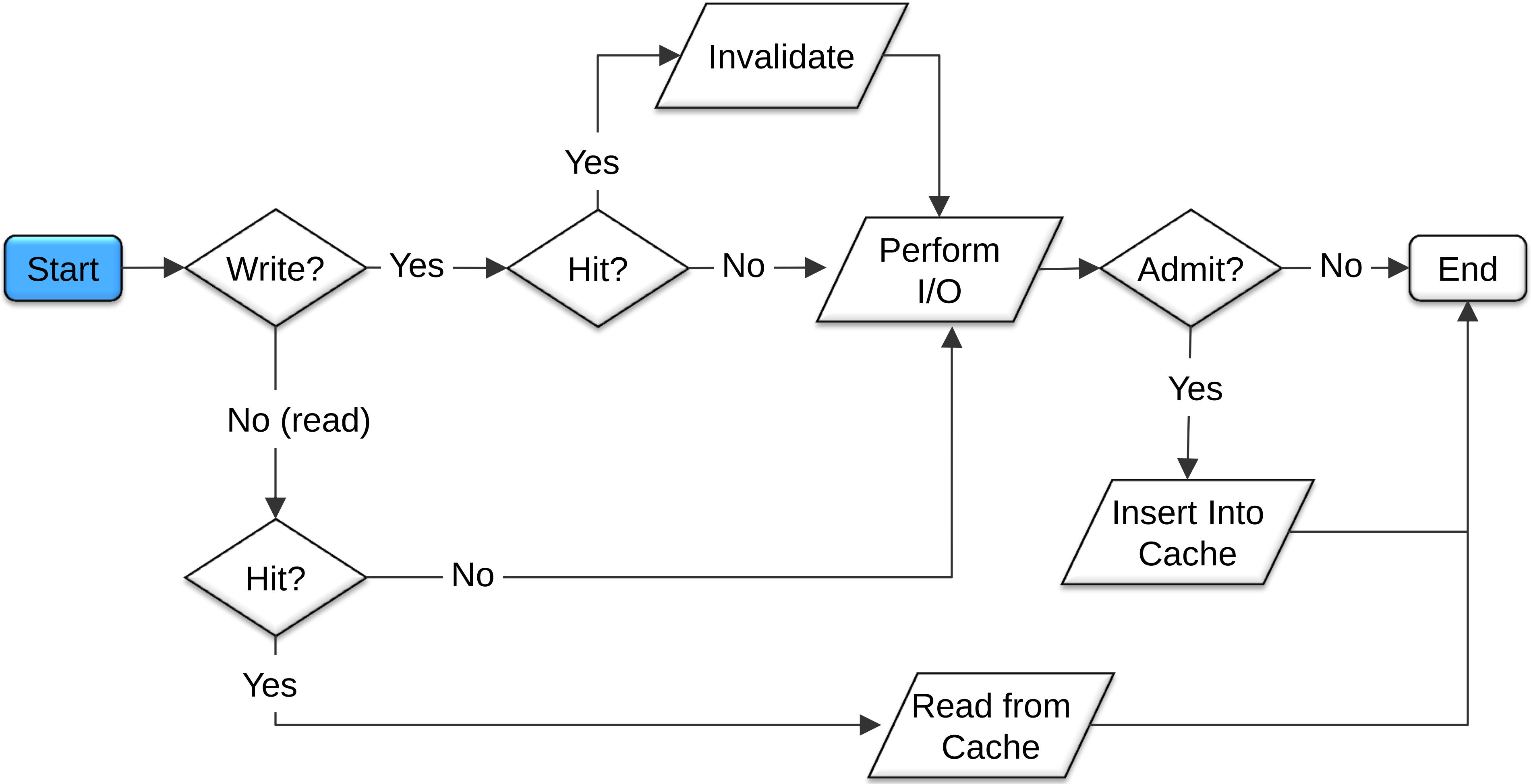
- Write-through
  - Simplifies cache consistency
- Persistent
  - Warm cache on restart
  - Cache durability after a crash is future work



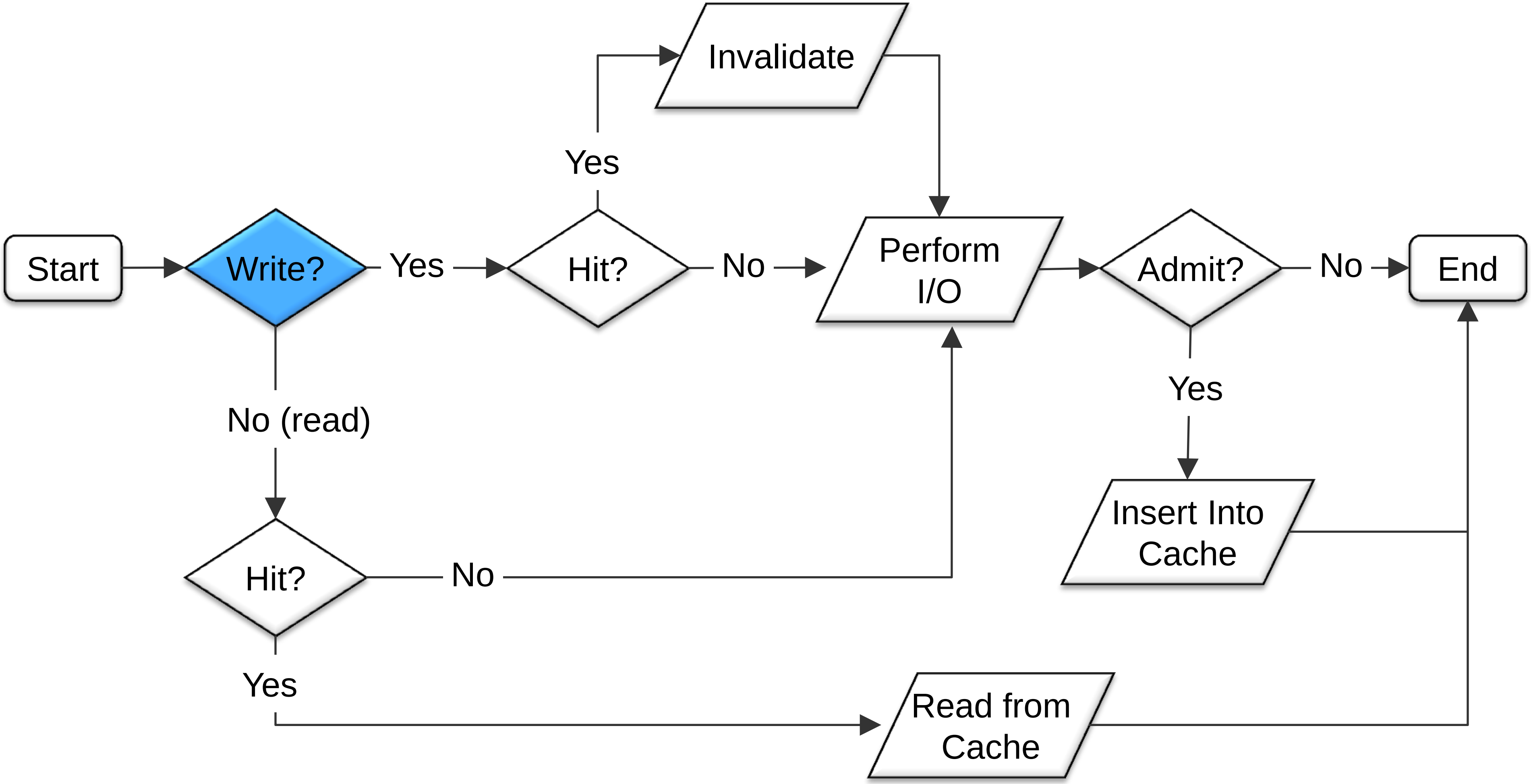
Simplified I/O Flow



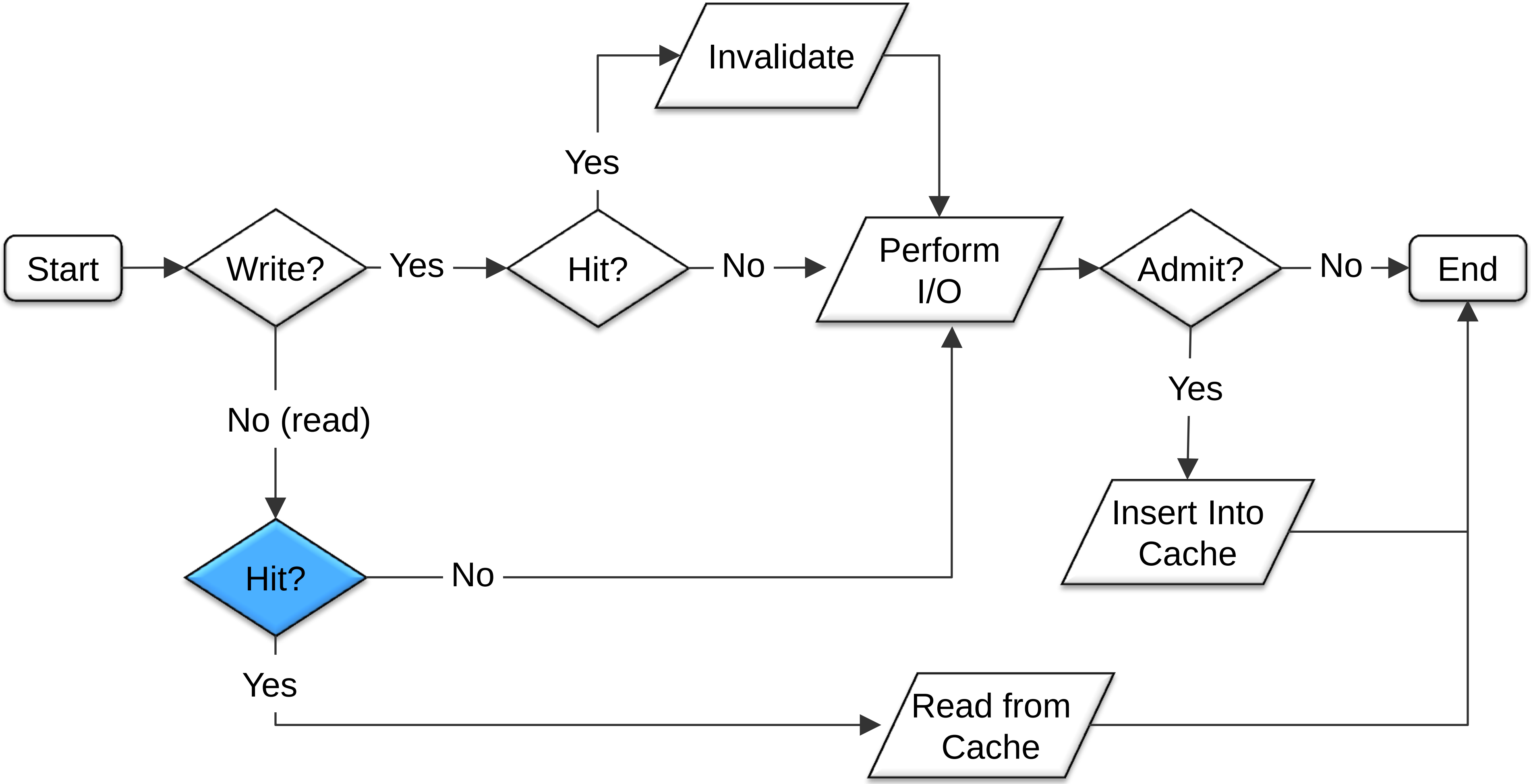
Read Processing Example (1 of 6)



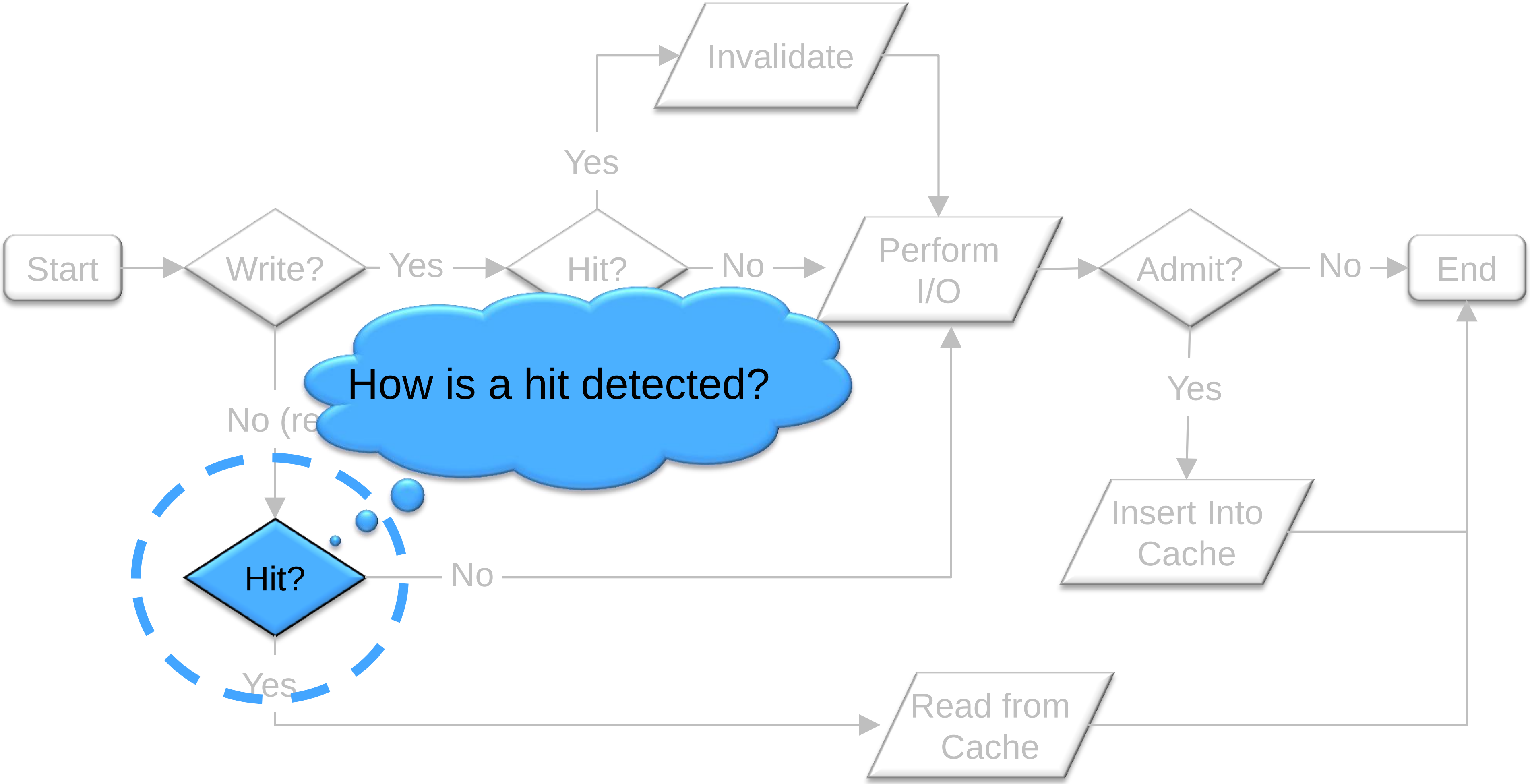
Read Processing Example (2 of 6)



Read Processing Example (3 of 6)



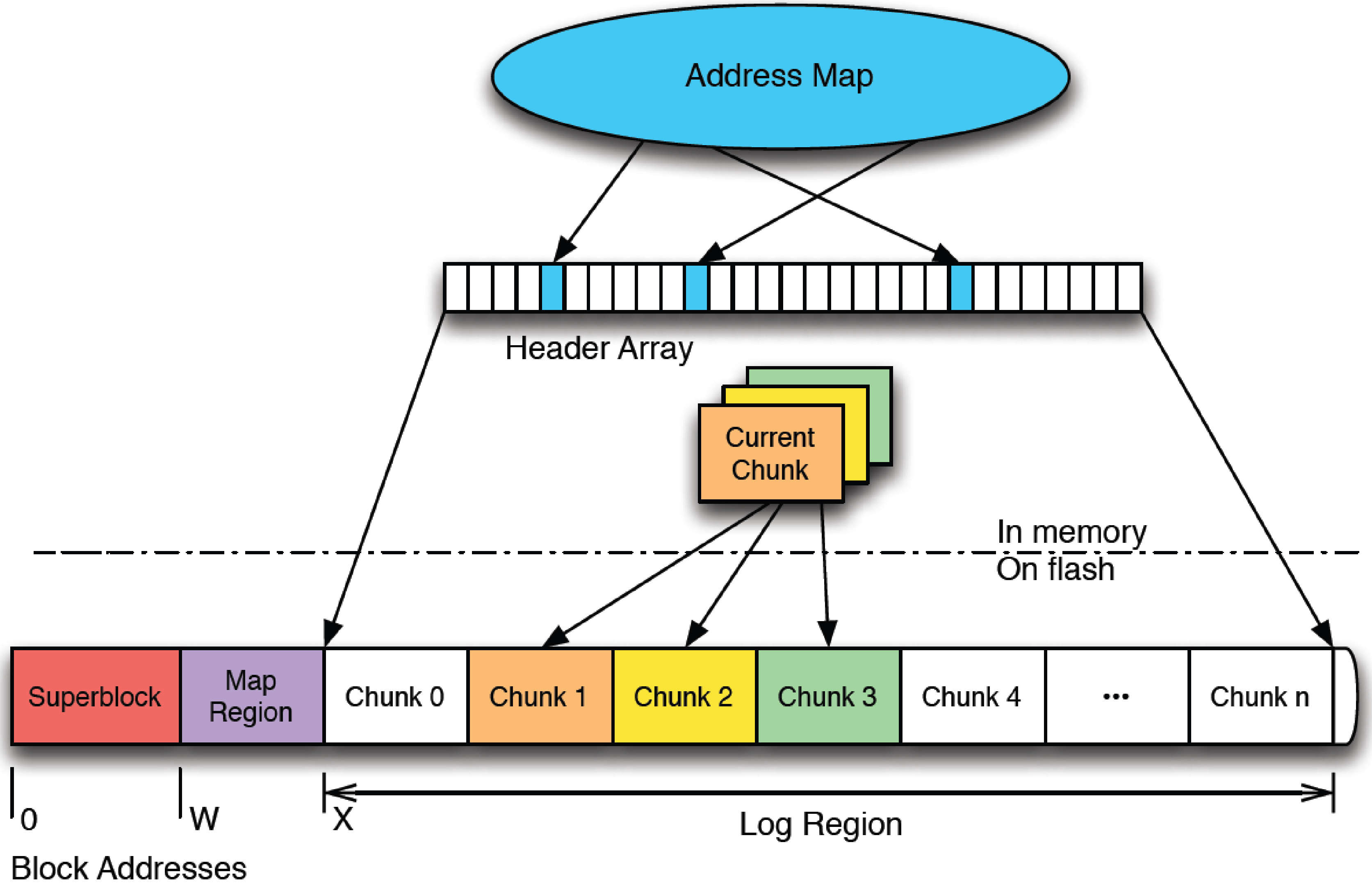
Read Processing Example (4 of 6)



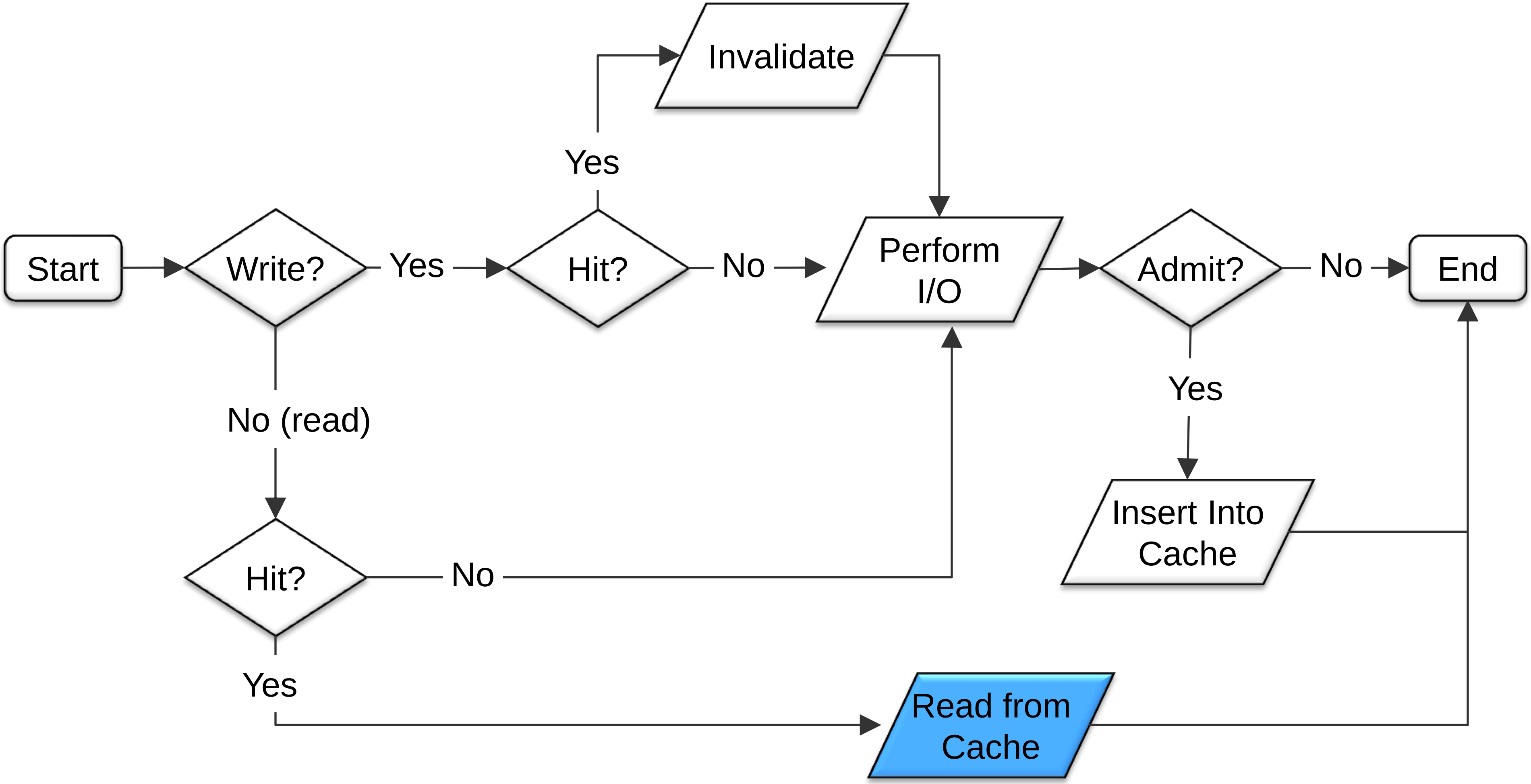
Detecting Cache Hits

- All cache metadata in RAM for speed.
  - Mercury is a second-level cache → modest hit rate → minimize cache overhead
  - Memory-to-cache ratio is 0.5% (e.g., 500 GB cache requires 2.5 GB of RAM)
- Cache headers
  - One header for each block in the cache
  - Implemented as a simple array
- Address Map
  - Dictionary maps (primary storage, LBA) keys to header index values
  - Implemented with hash table, O(1) lookup time

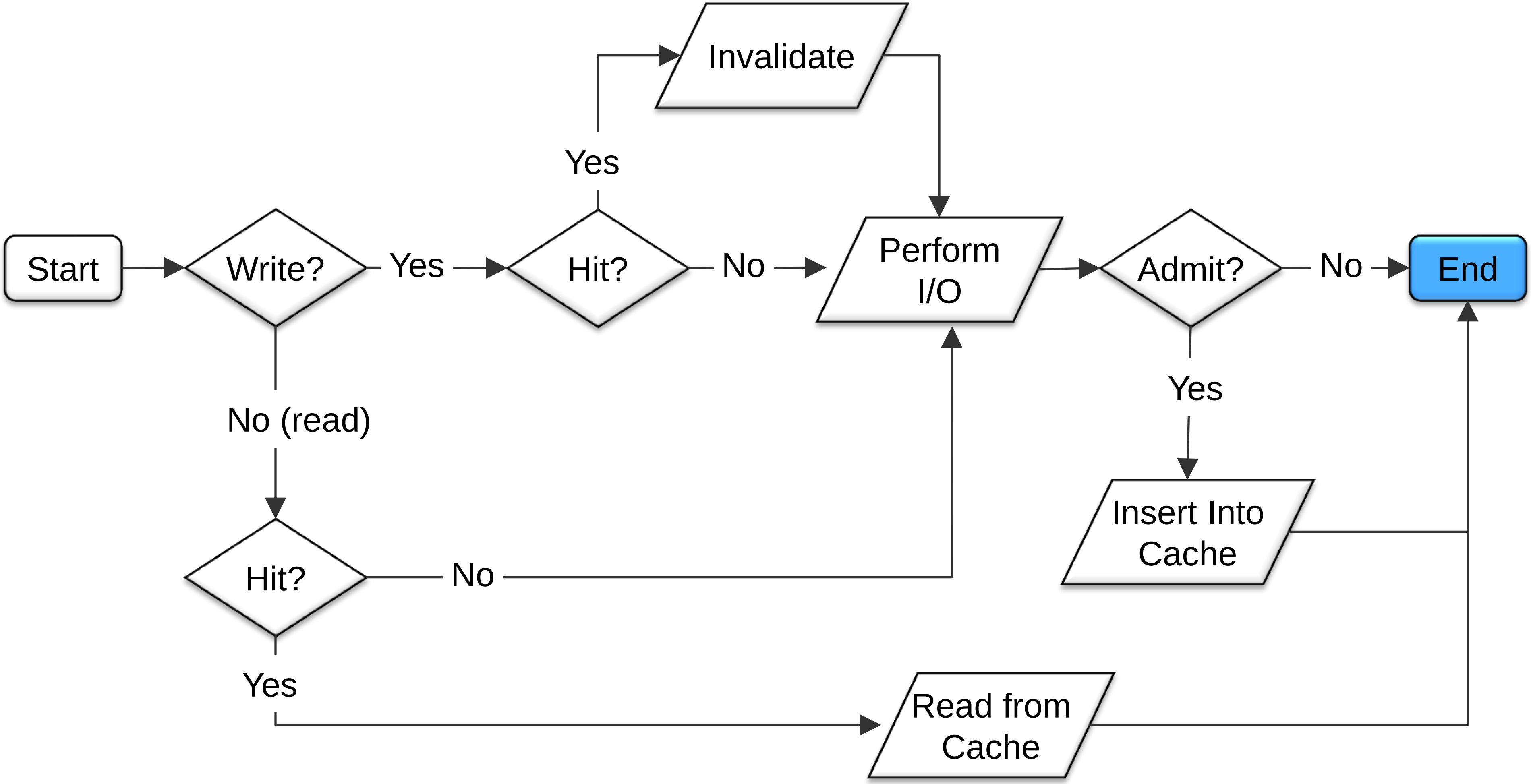
Data Structures



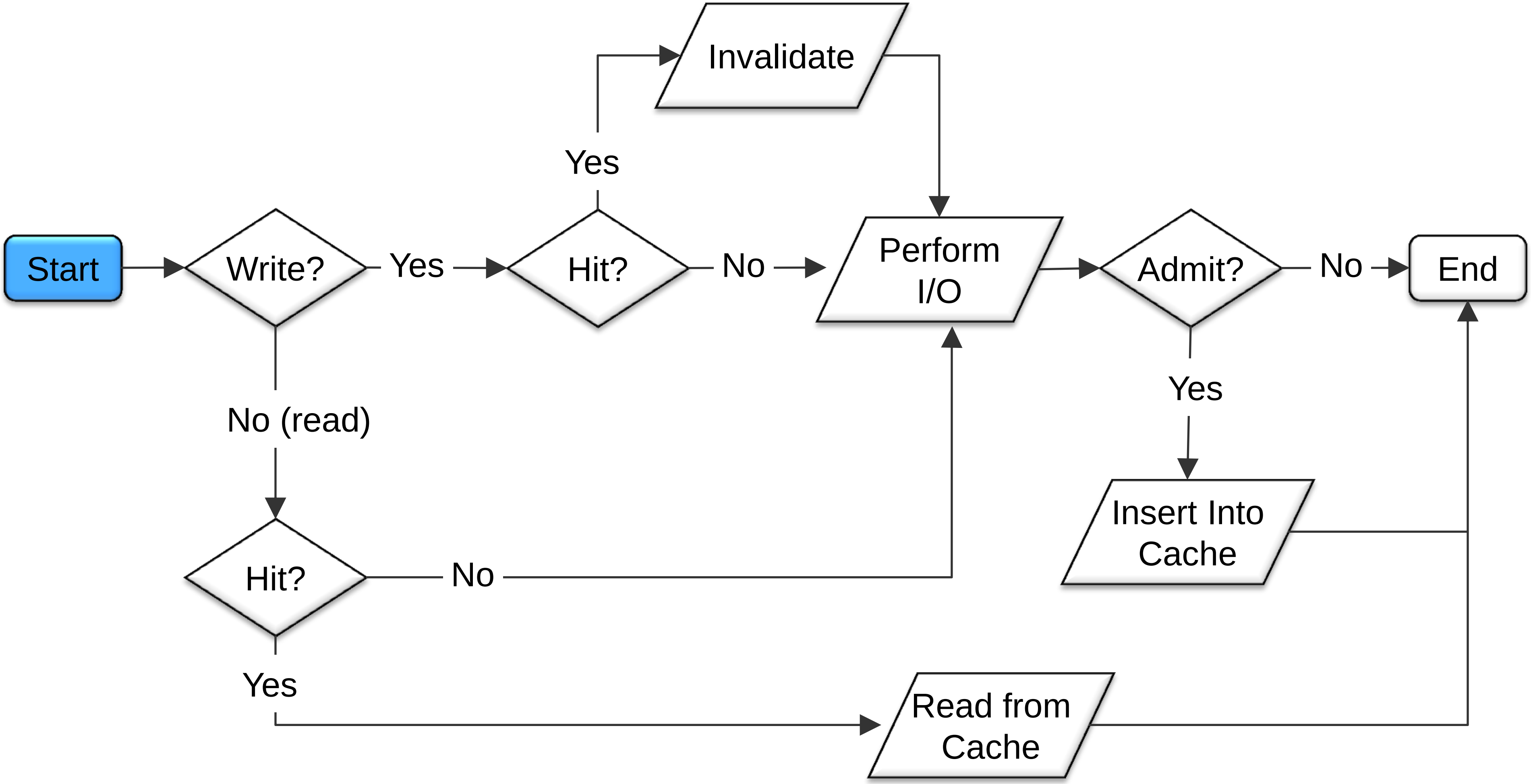
Read Processing Example (5 of 6)



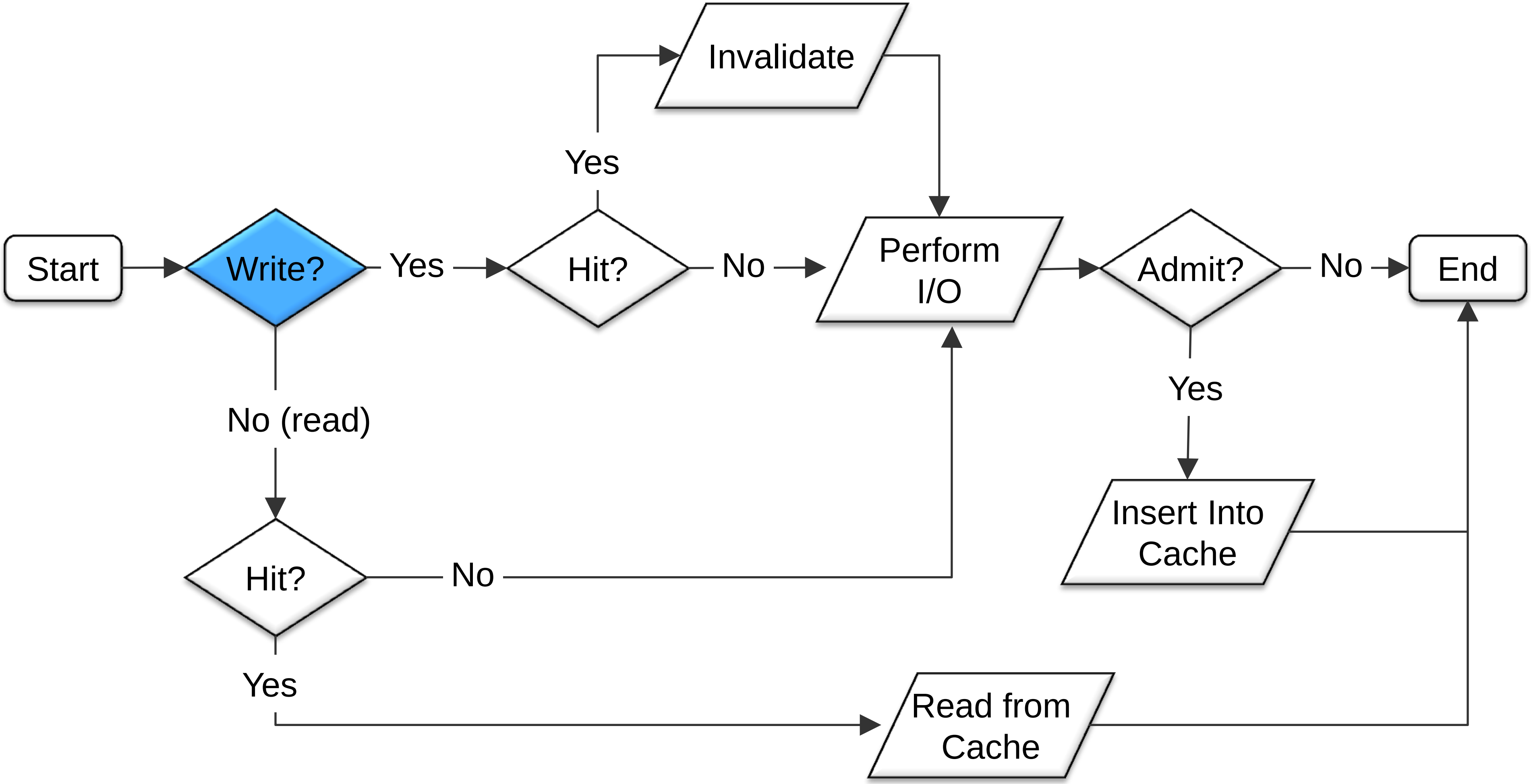
Read Processing Example (6 of 6)



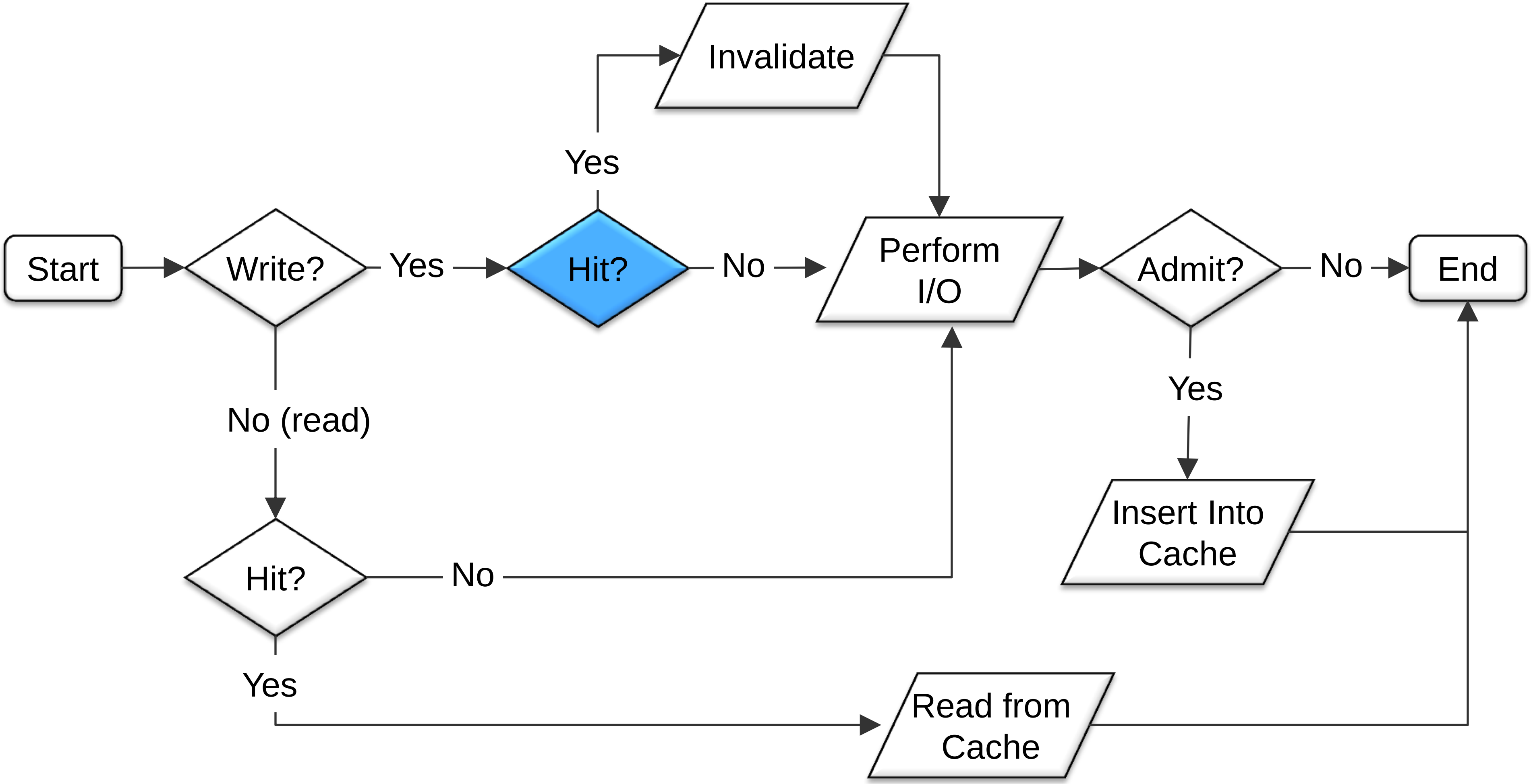
Write Processing Example (1 of 11)



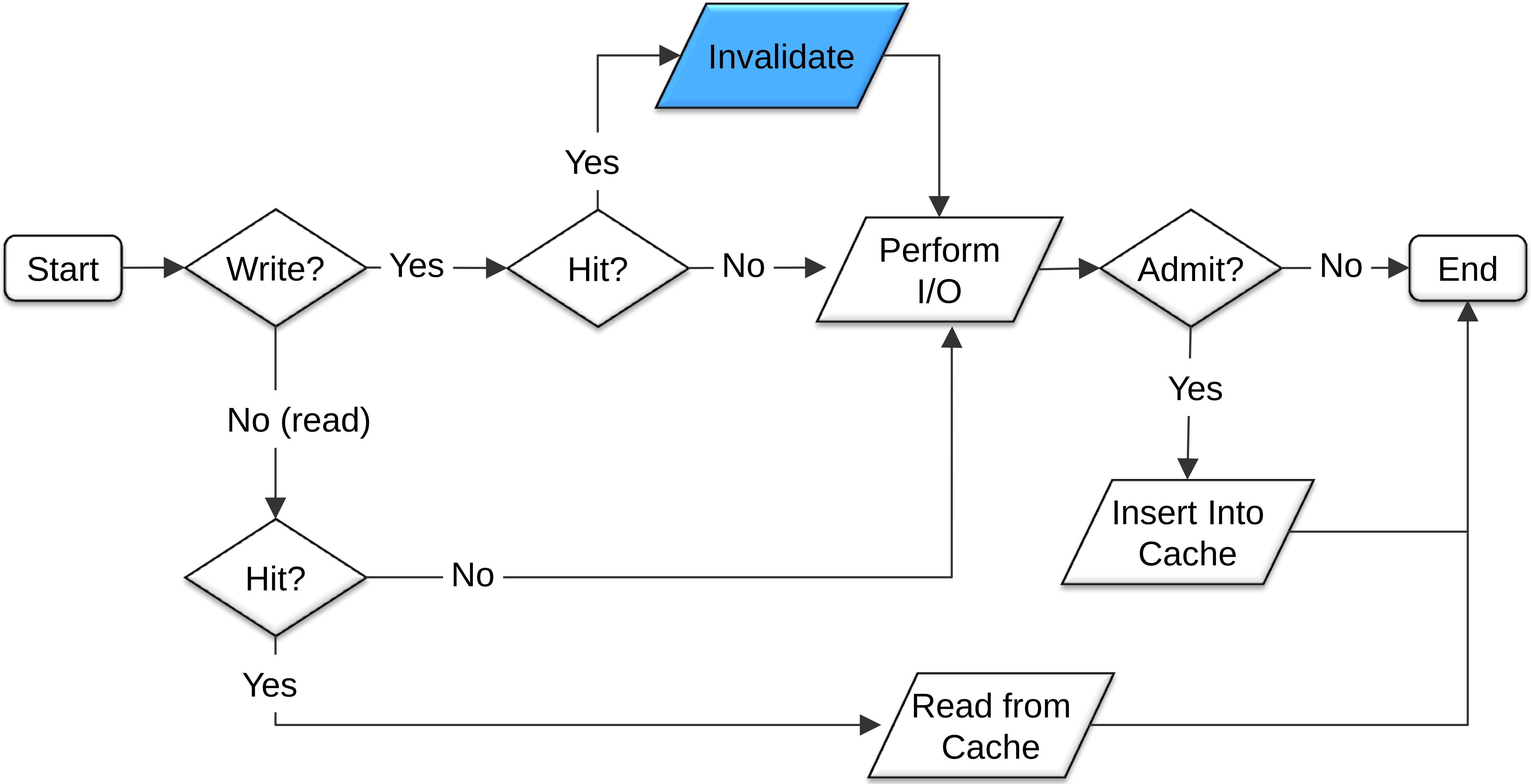
Write Processing Example (2 of 11)



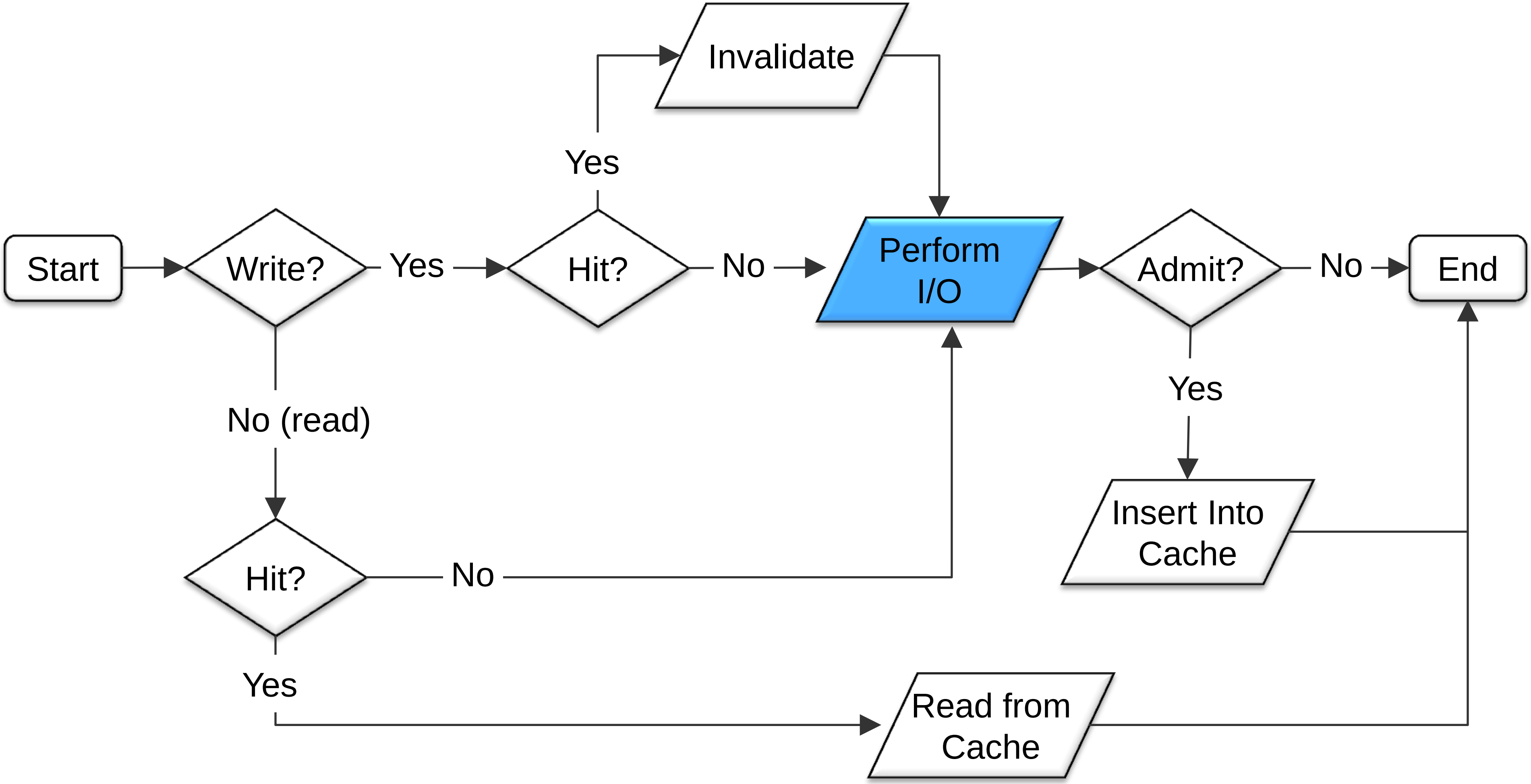
Write Processing Example (3 of 11)



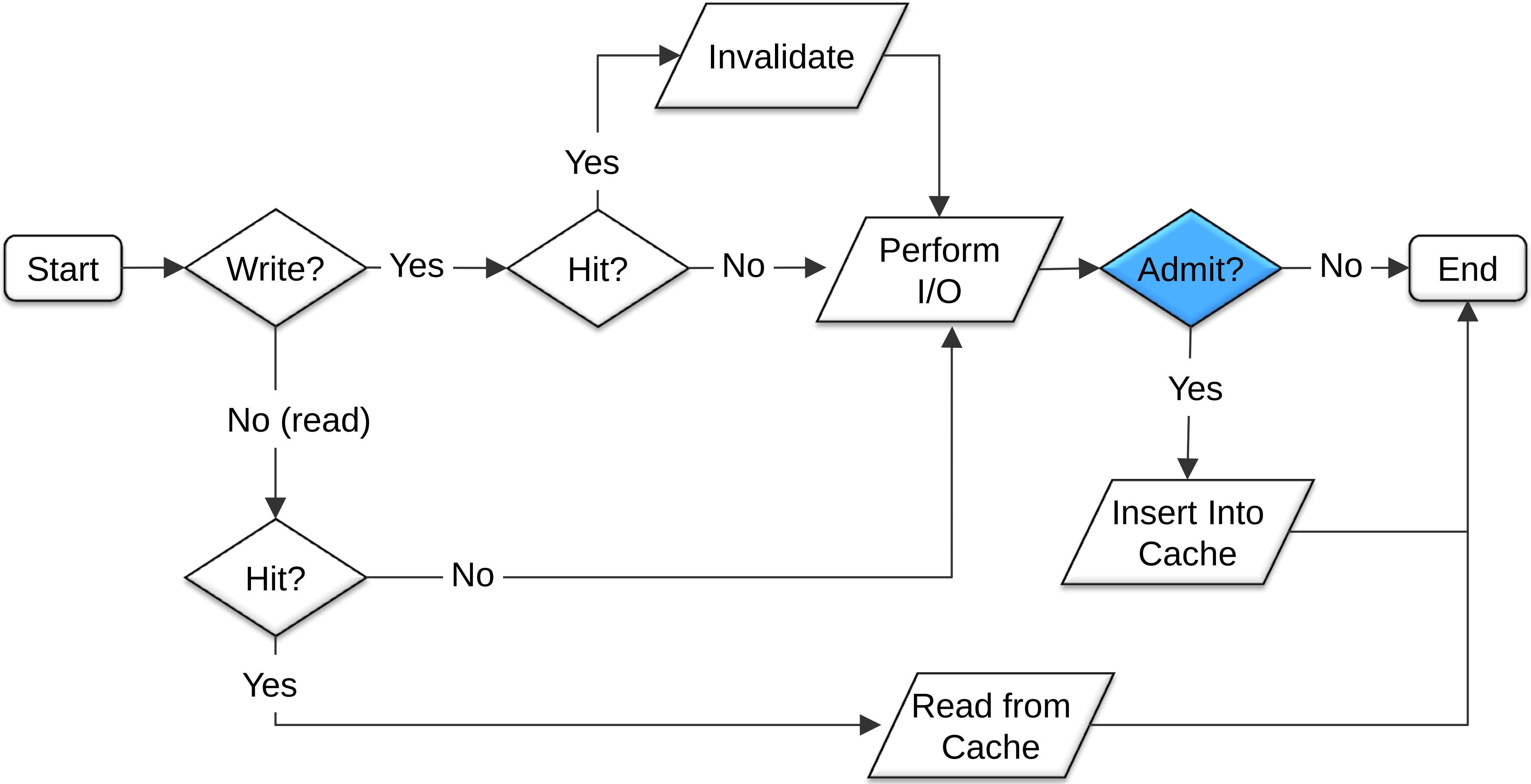
Write Processing Example (4 of 11)



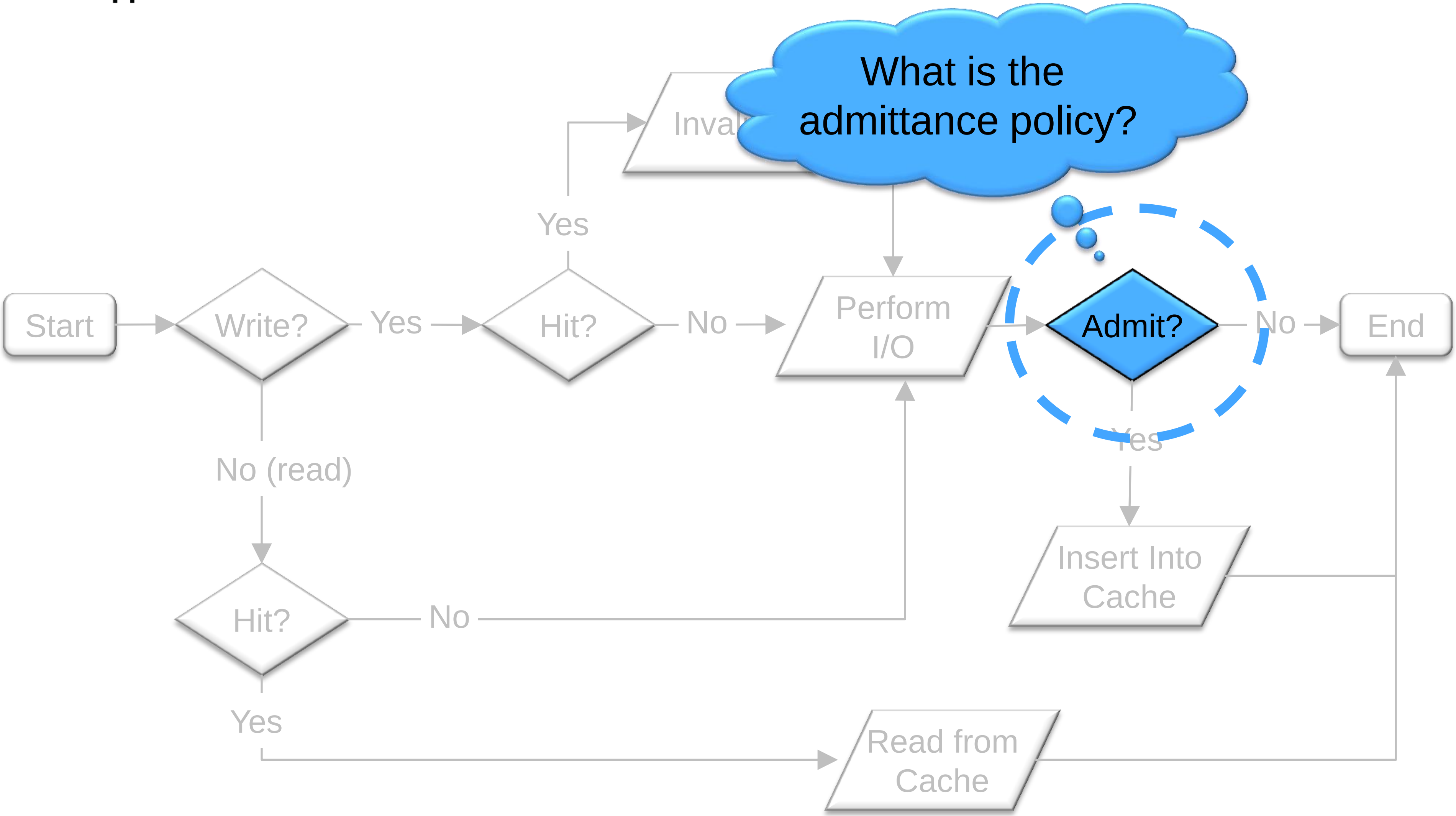
Write Processing Example (5 of 11)



Write Processing Example (6 of 11)



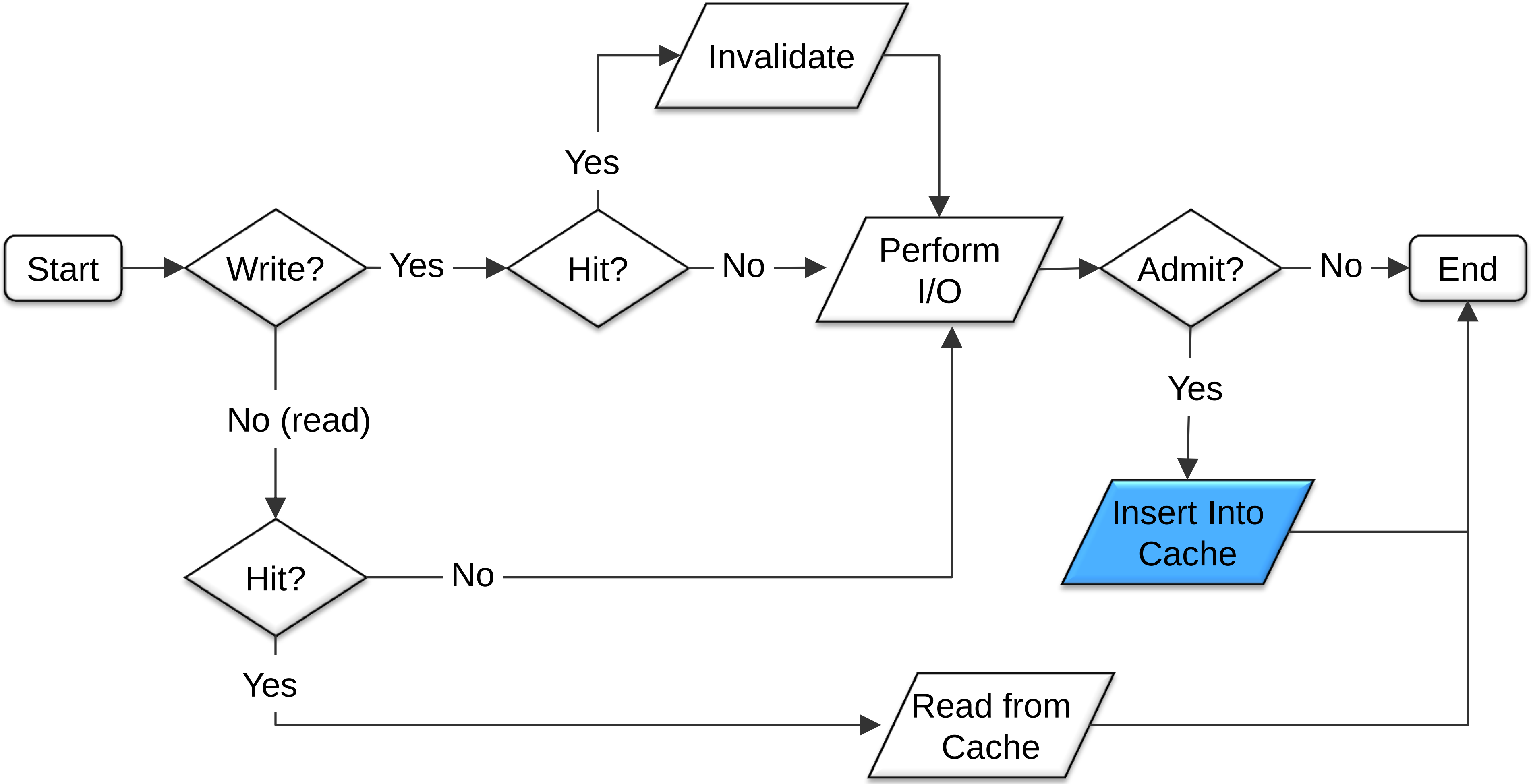
Write Processing Example (7 of 11)



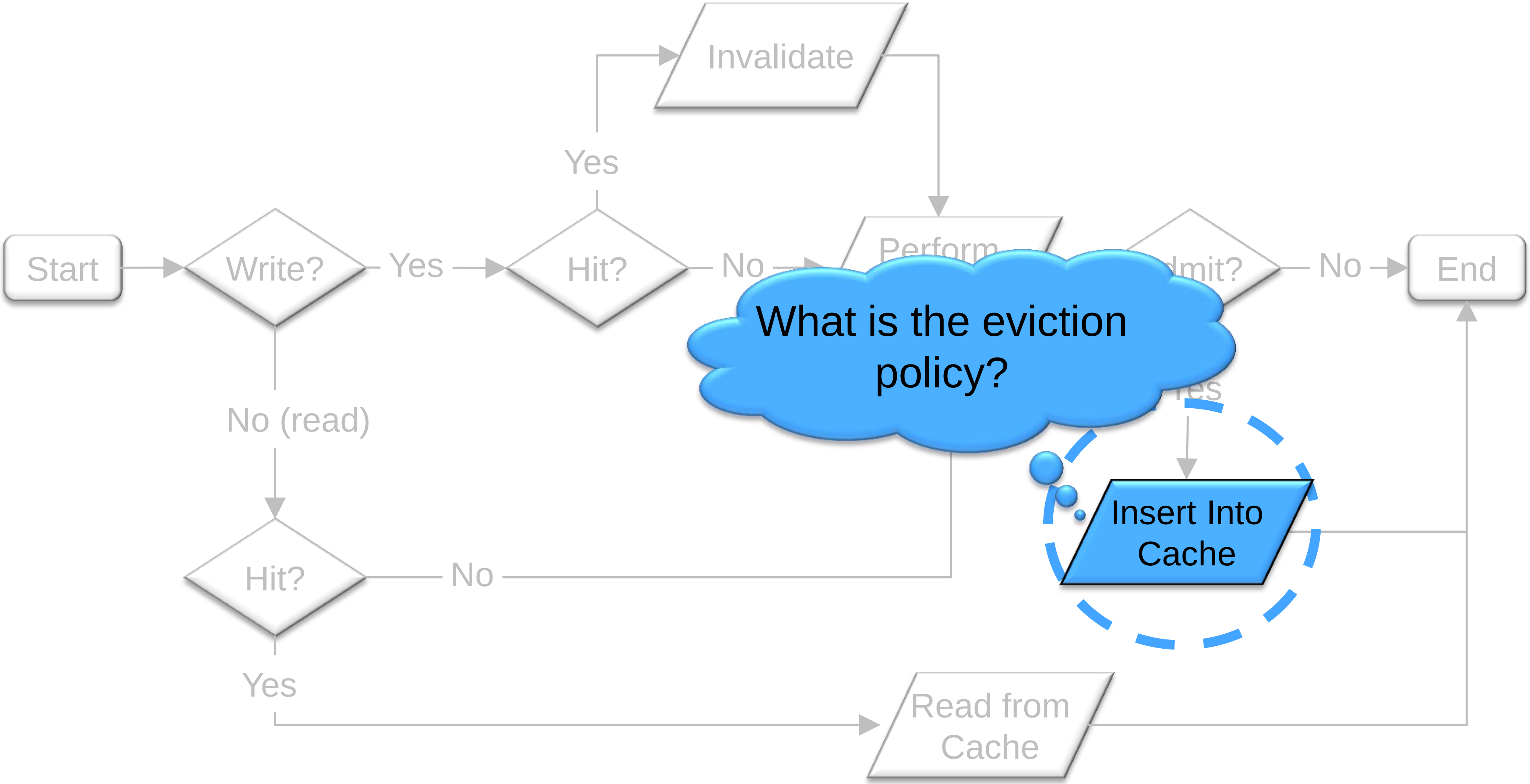
Admittance Policies

- Unrestricted (default)
  - All writes and read misses are inserted into the cache
- Write-Around
  - writes skip the cache
- Sequential I/O Bypass (future work)
  - Sequential reads, writes, or both skip the cache

Write Processing Example (8 of 11)



Write Processing Example (10 of 11)



Invalidate

Yes

Start

Write?

Yes

Hit?

No

Perform

dmit?

No

End

What is the eviction policy?

es

Insert Into Cache

No (read)

Hit?

No

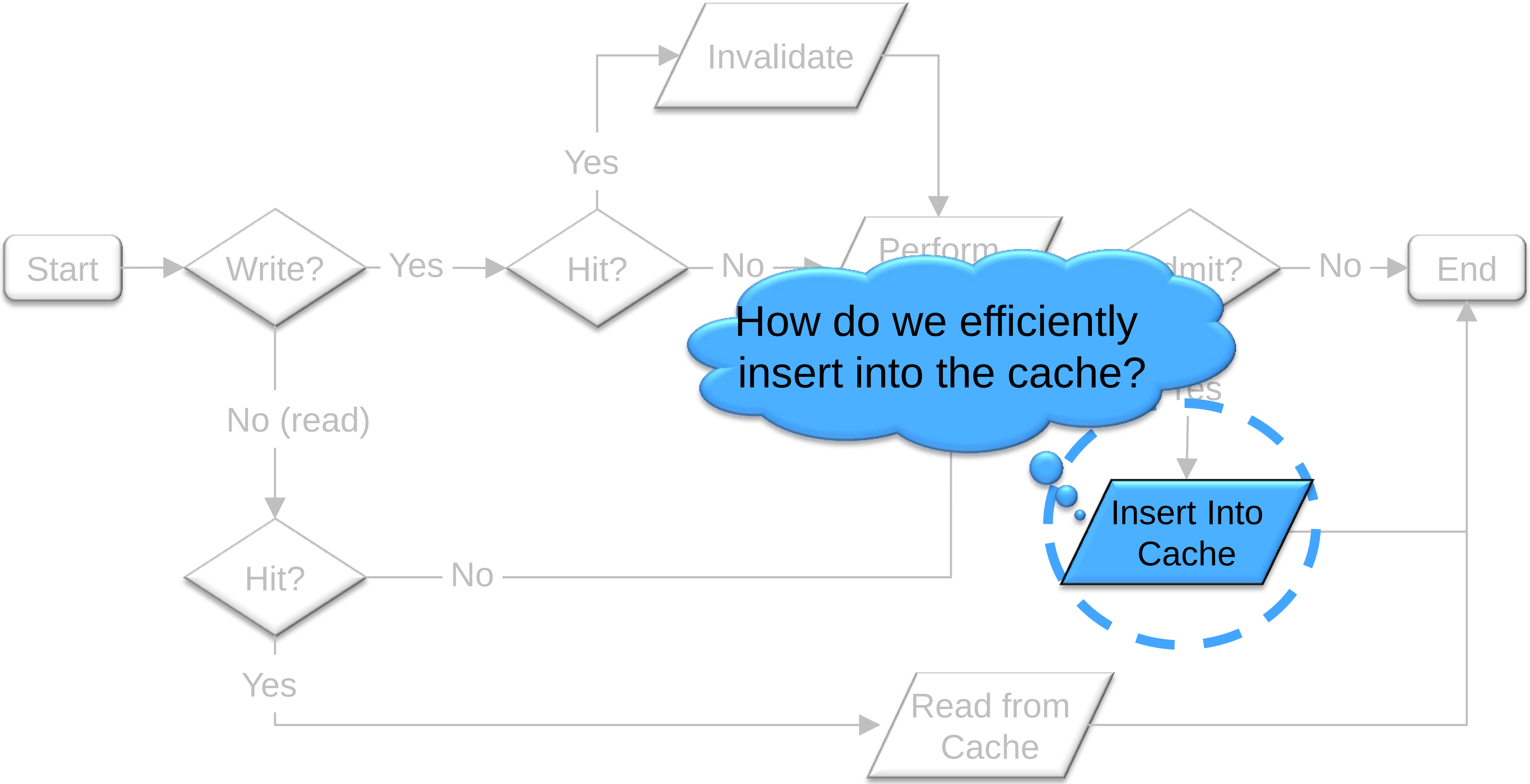
Yes

Read from Cache

Eviction Policies

- First In First Out (FIFO)
  - – Less I/O to clean log, but lower hit rate
    - Eliminates reads during log cleaning.
- CLOCK
  - – Higher hit rate, but more expensive log cleaning compared to FIFO.

Write Processing Example (9 of 11)



Invalidate

Yes

Start

Write?

Yes

Hit?

No

Perform

dmit?

No

End

How do we efficiently insert into the cache?

es

Insert Into Cache

No (read)

Hit?

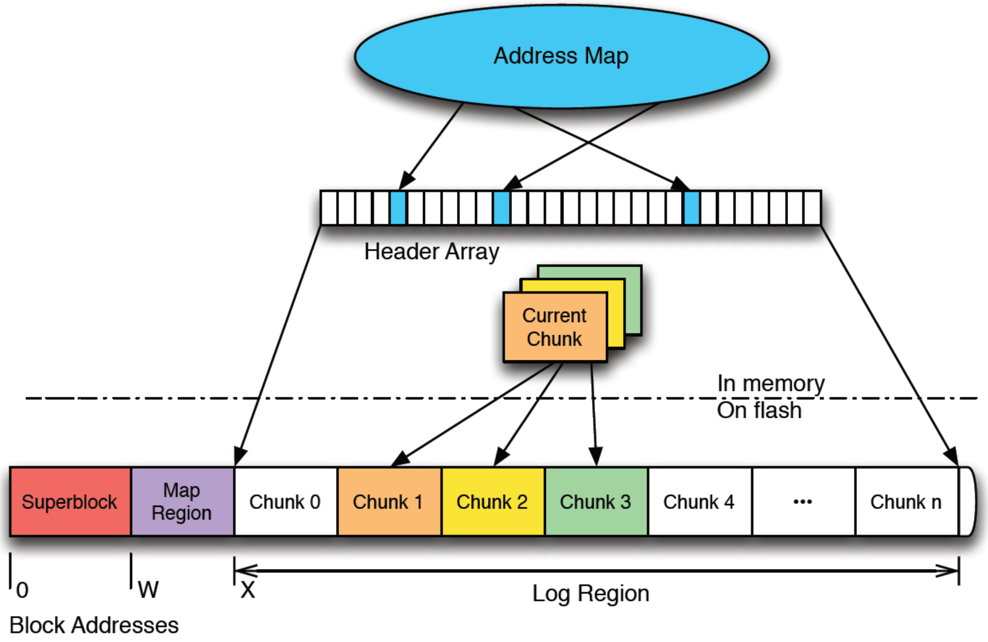
No

Yes

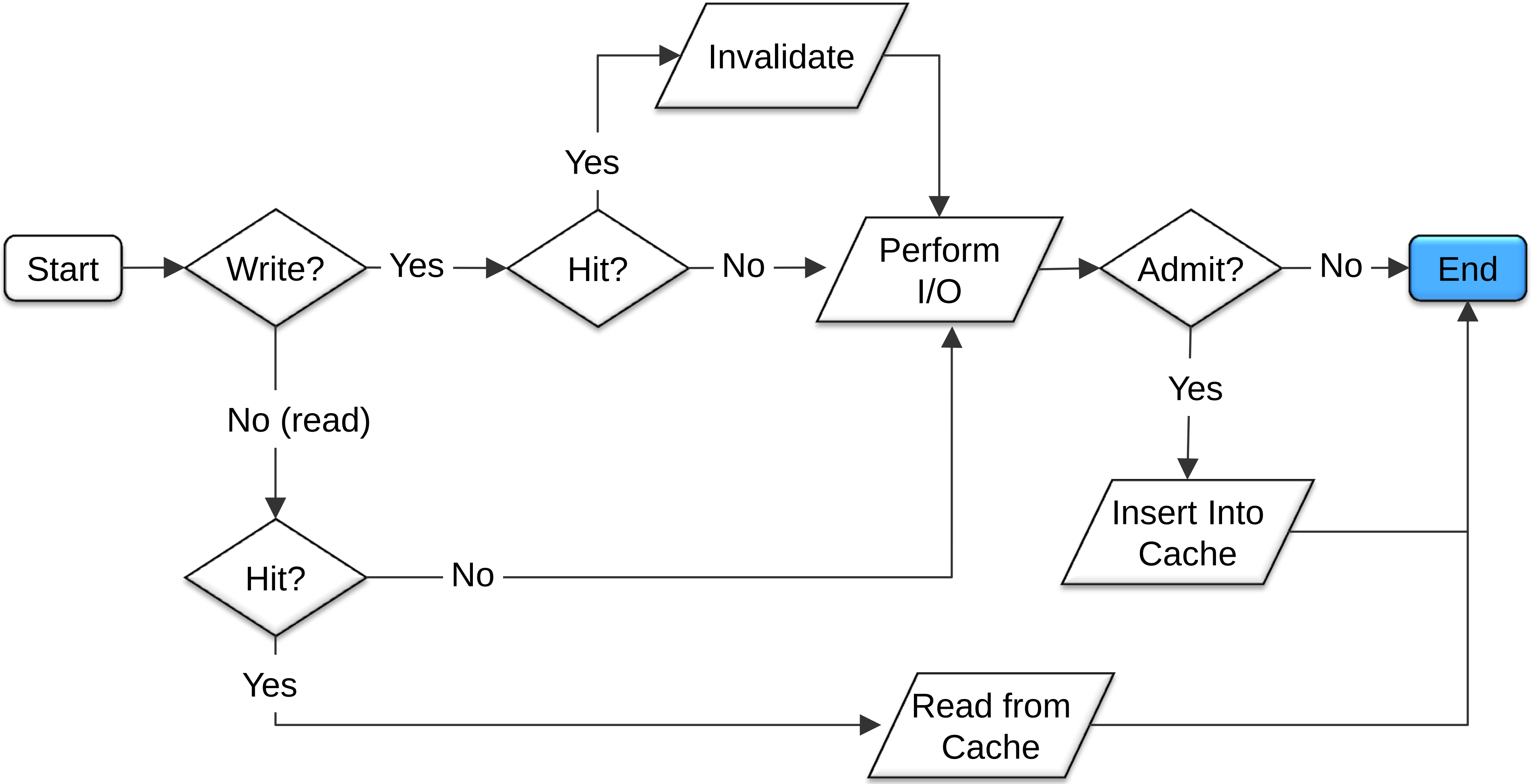
Read from Cache

Cache Insertion

- Specialize I/O access patterns for flash
  - Log-structured writes with erase block size chunks to minimizes SSD FTL's (flash translation layer) cleaning



Write Processing Example (11 of 11)



Go further, faster®

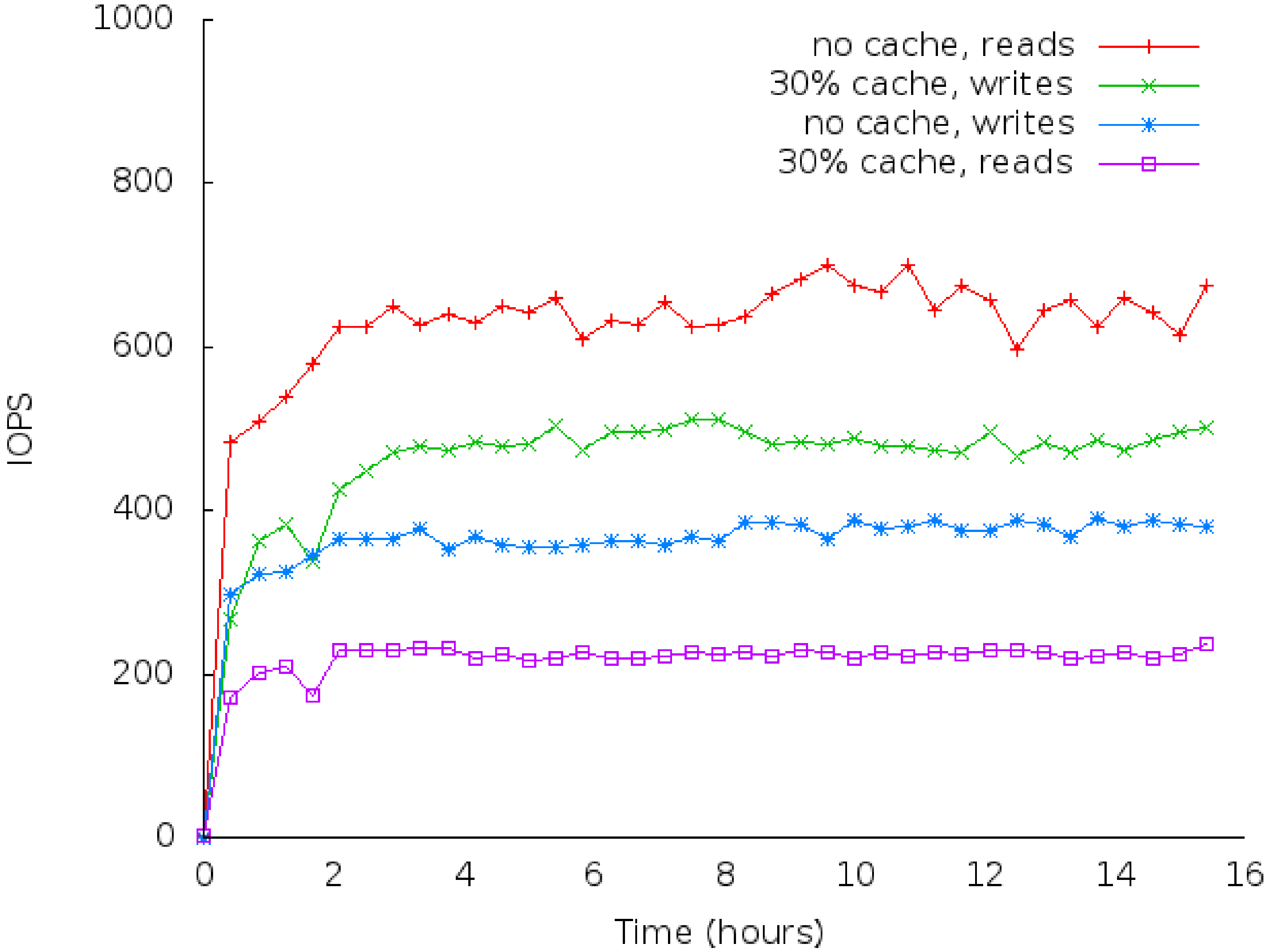
Part III. Evaluation

Evaluation

- Two workloads:
  - Microsoft® Exchange Jetstress
  - NetApp® Enterprise Workload[1]
- PCIe device with SLC (single-level cell) flash
  - Paper contains SLC and MLC SSD results
- x86 Server with Linux, KVM/QEMU
- NetApp FAS3270 with iSCSI LUN(s)

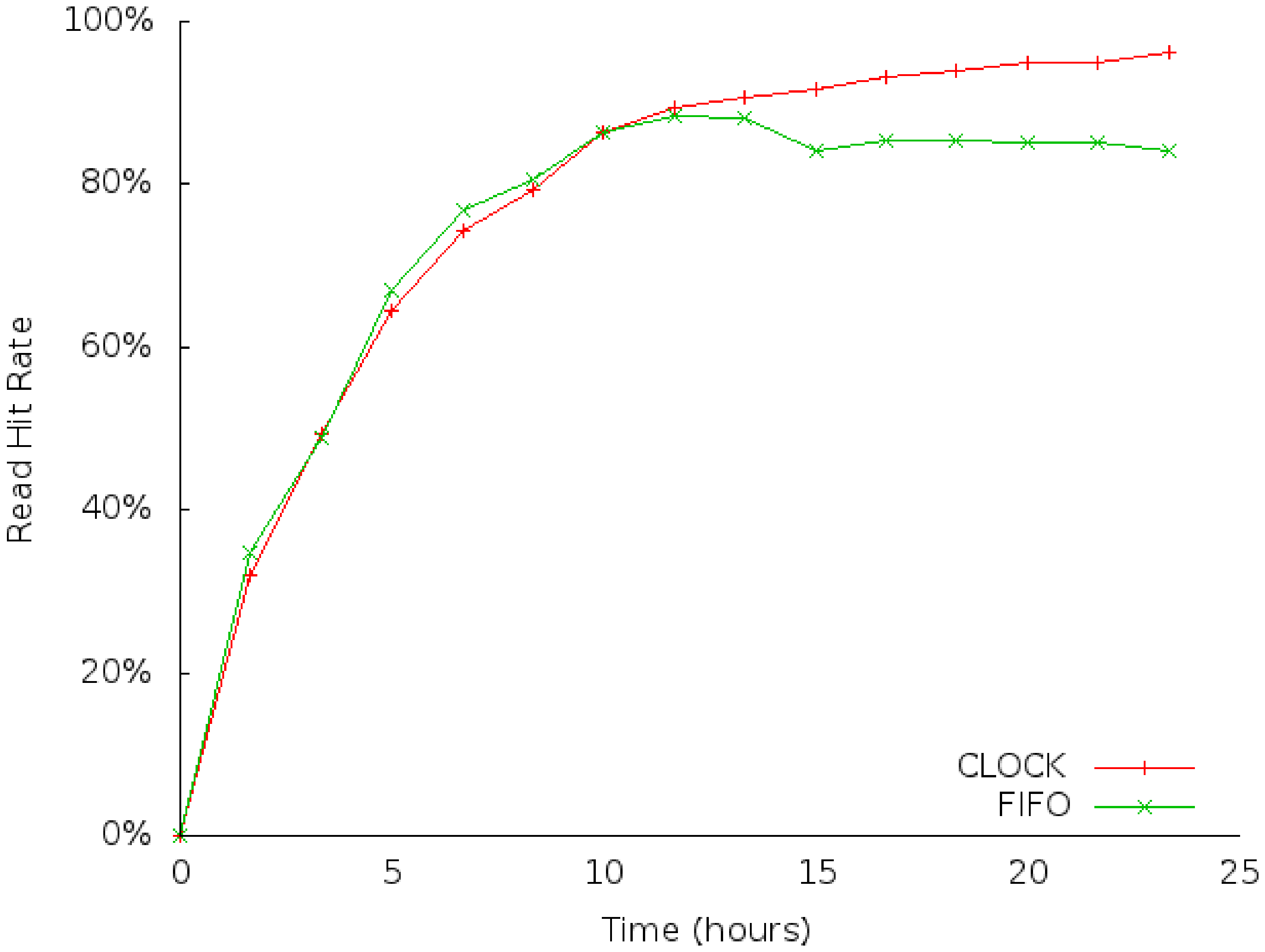
[1] S. Daniel et al., *A portable, open-source implementation of the SPC-1 workload*.

Cache reduces access to network storage



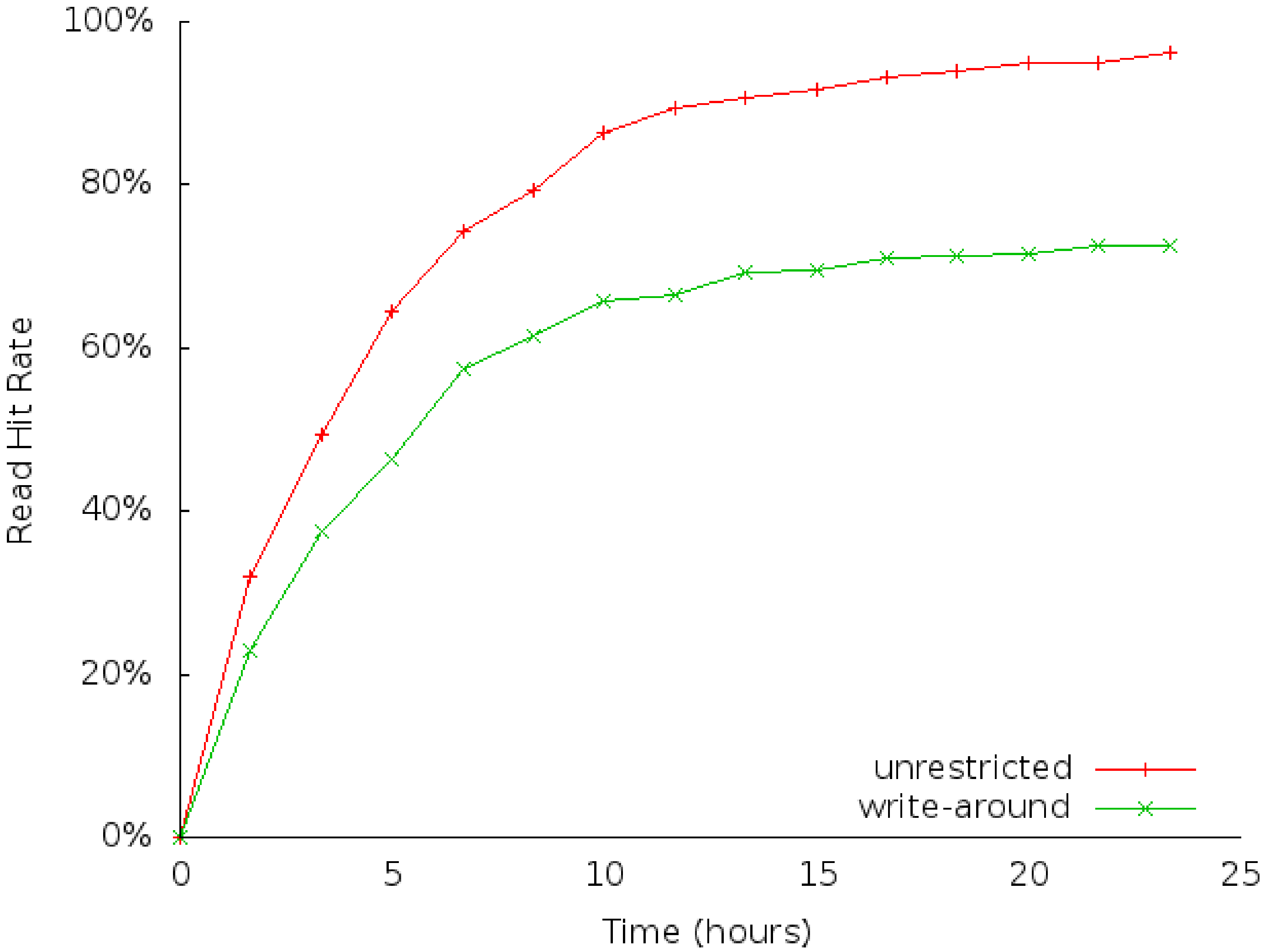
Jetstress workload. Unrestricted admittance policy. FIFO eviction policy. PCIe flash device.

Warming the cache takes a long time



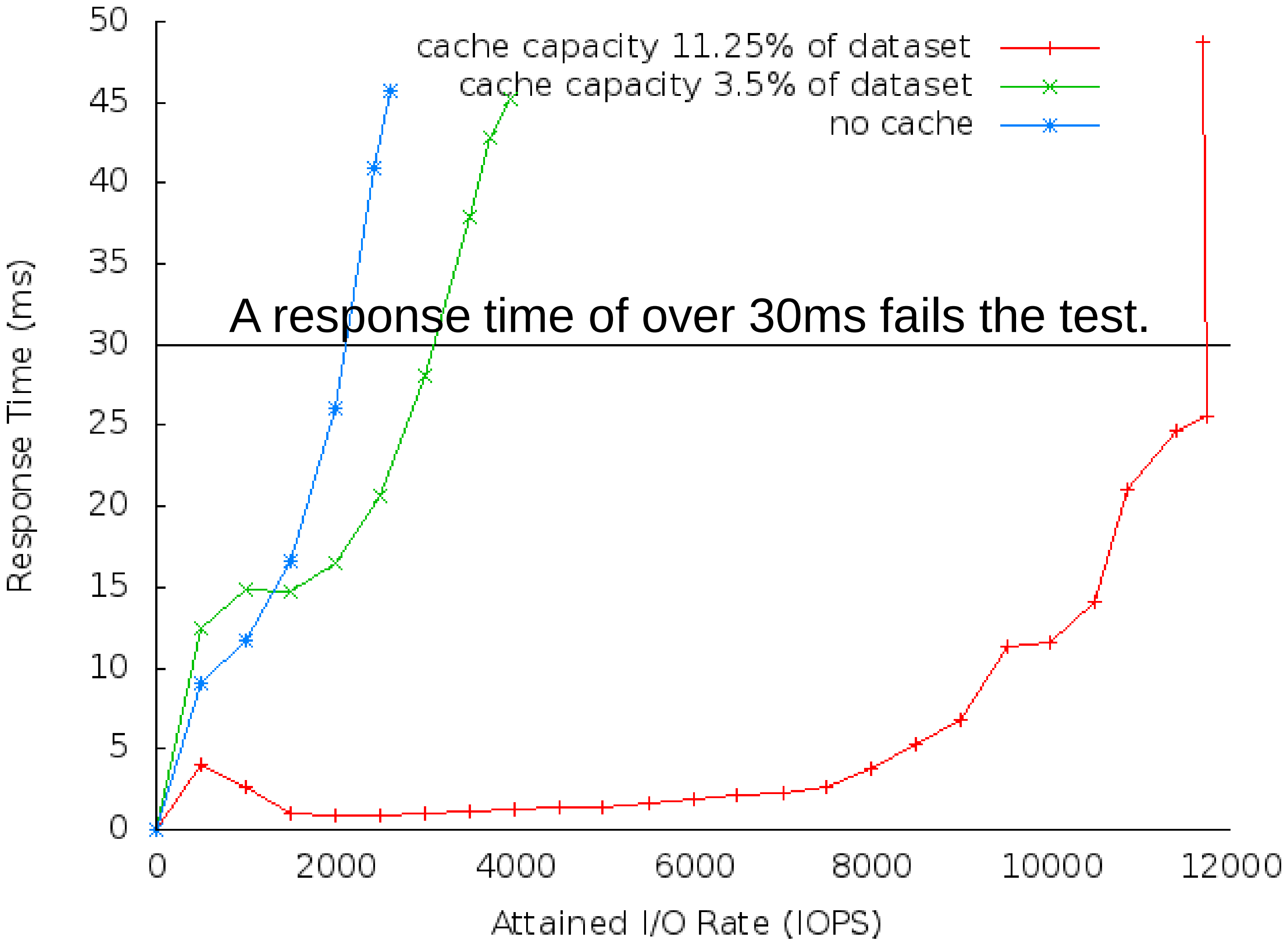
Enterprise workload. Unrestricted admittance policy. PCIe flash device capacity 11.25% of datase

Unrestricted Beats Write-Around



Enterprise workload. CLOCK eviction policy. PCIe flash device capacity 11.25% of dataset.

Significant Response Time Improvement



Enterprise workload. Unrestricted admittance policy. CLOCK eviction policy. PCIe flash device.

Summary

- **Host-side flash**
  - – minimizes flash access latency
- **Hypervisor I/O cache**
  - – simplifies deployment
- **Persistent**
  - – cache is warm on a restart
- **Write-through**
  - – consistent with primary storage

NetApp®

Thank you