The Storage Hierarchy is Not a Hierarchy: Optimizing Caching on Modern Devices with Orthus

Kan Wu, Zhihan Guo, Guanzhou Hu, Kaiwei Tu,
Ramnatthan Alagappan, Rathijit Sen, Kwanghyun Park, Andrea Arpaci-Dusseau and Remzi Arpaci-Dusseau

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON
vmware®
Microsoft

Storage Hierarchy

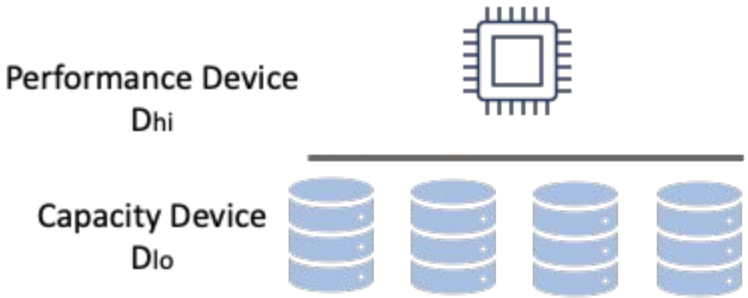
- Long been central to system designs

"Ideally one would desire an indefinitely large memory capacity ... It does not seem possible to achieve such a capacity. We are therefore forced to recognize the possibility of constructing a hierarchy of memories ..."

--- "**Preliminary Discussion of the Logical Design of an Electronic Computing Instrument**" (1946),
by Burks, Goldstine, and von Neumann.

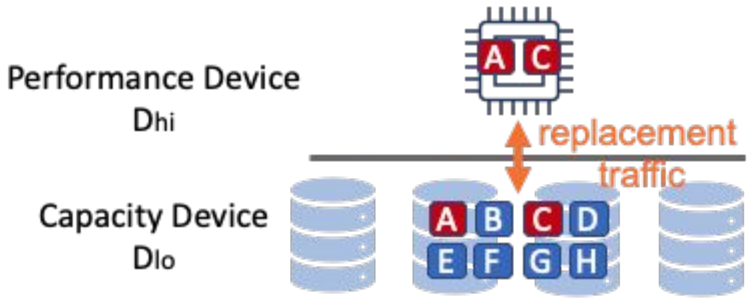
Storage Hierarchy

- Simplified two-layer hierarchy
  - **Performance Device:** fast, expensive, small
  - **Capacity Device:** slow, cheap, large



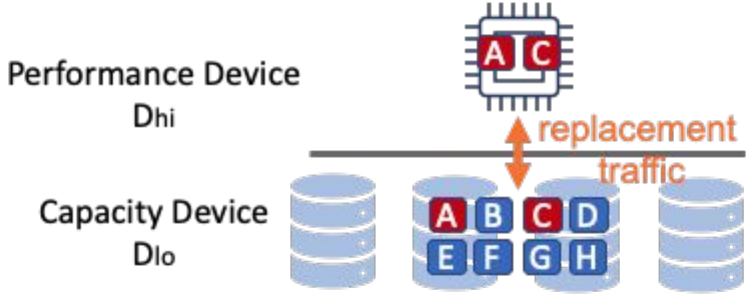
Caching

- Replicating popular items in **Performance Device**



Caching Wisdom: Maximizing Hit Rates

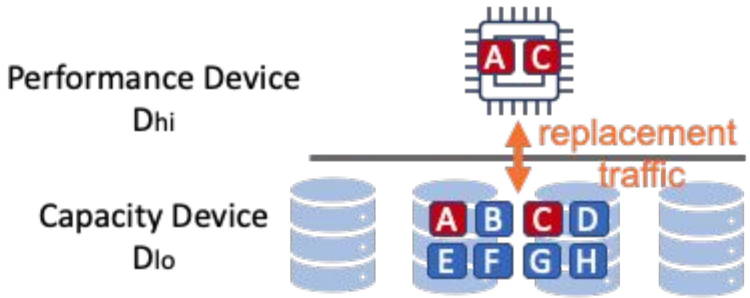
- Strives to direct most accesses to **Performance Device**



Caching Wisdom: Maximizing Hit Rates

- Strives to direct most accesses to **Performance Device**

- Caching delivers ~**Performance Device** speed along with **Capacity Device** capacity
- Traditionally, very good!
  - **Performance:** Performance Device >> Capacity device
  - E.g. DRAM vs. HDD (100x differences)



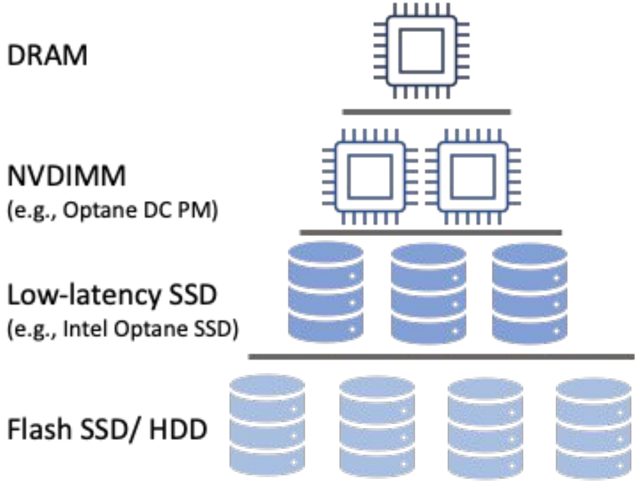
Problem: Caching is Insufficient in Modern Storage Hierarchies

- **Insight**:

  the assumption (Performance device >> Capacity device) is broken

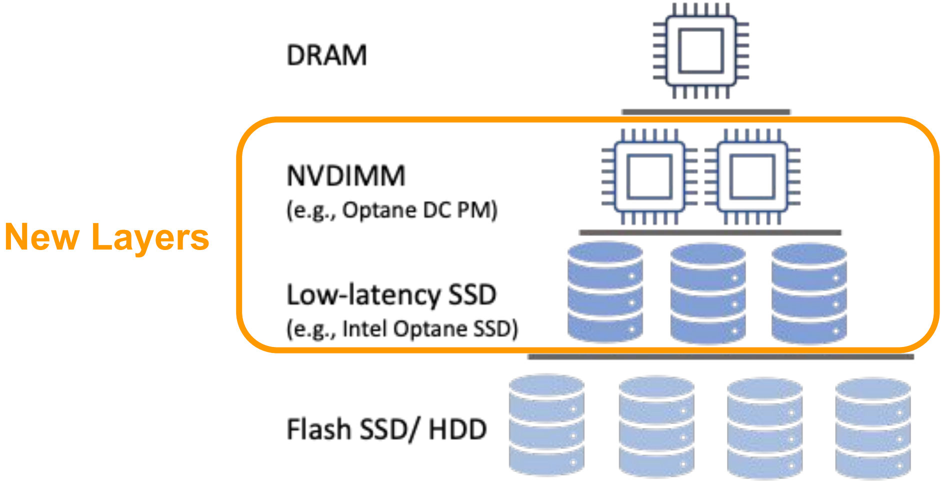
The Modern Storage Hierarchies

- Non-Volatile Memory-based devices are filling the performance gap



The Modern Storage Hierarchies

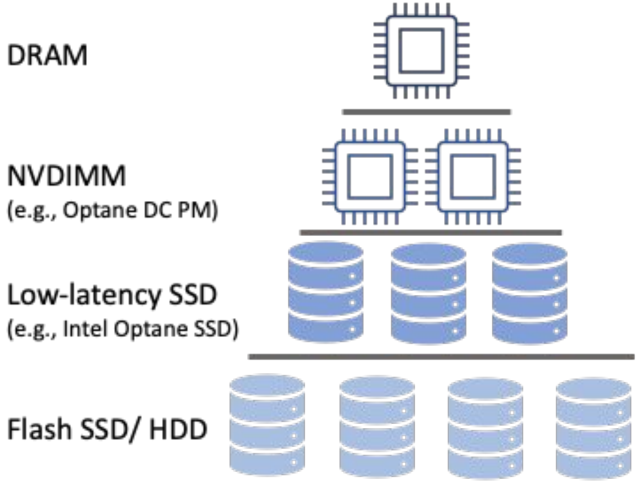
- Non-Volatile Memory-based devices are filling the performance gap
  - NVDIMM (300ns, ~7GB/s)
  - Low-latency SSD (10us, ~3GB/s)



The Modern Storage Hierarchies

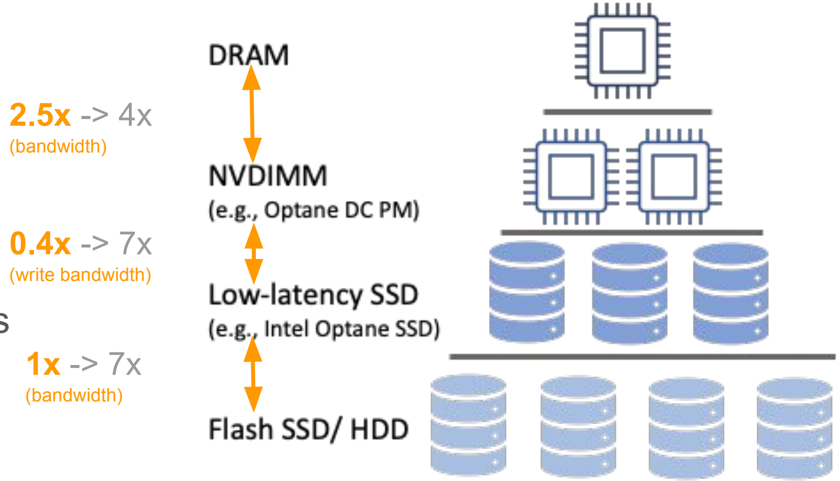
- Non-Volatile Memory-based devices are filling the performance gap
  - NVDIMM (300ns, ~7GB/s)
  - Low-latency SSD (10us, ~3GB/s)

- The differences between today's neighboring layers are less clear and even overlapping (depending on workloads)



The Modern Storage Hierarchies

- Non-Volatile Memory-based devices are filling the performance gap
  - NVDIMM (300ns, ~7GB/s)
  - Low-latency SSD (10us, ~3GB/s)

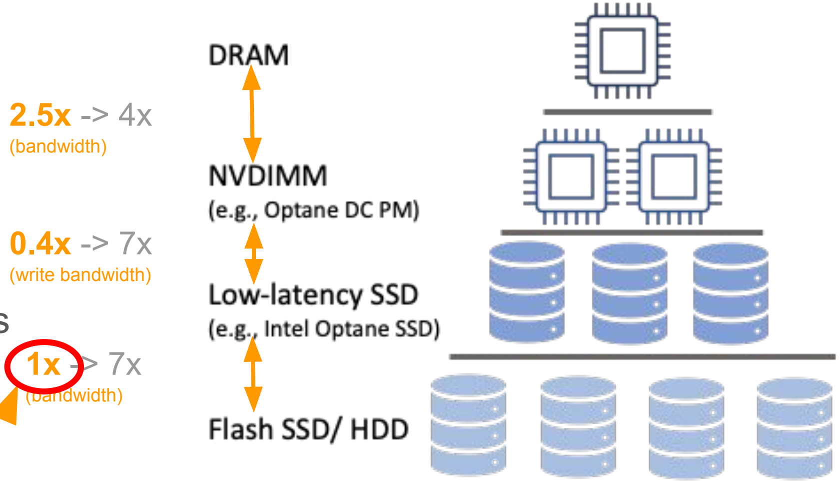


- The differences between today's neighboring layers are less clear and even overlapping (depending on workloads)

The Modern Storage Hierarchies

- Non-Volatile Memory-based devices are filling the performance gap
  - NVDIMM (300ns, ~7GB/s)
  - Low-latency SSD (10us, ~3GB/s)

- The differences between today’s neighboring layers are less clear and even overlapping (depending on workloads)
- E.g., serving reads with high parallelism, Optane SSD ~ Flash SSD, caching leaves huge performance available in Flash SSD **unexploited**



It’s imperative to rethink how modern hierarchies should be managed.

Our Approach:
Non-Hierarchical Caching (NHC)

- Insight: we should treat modern hierarchy in a less hierarchical manner
  - The available performance in capacity devices should be exploited

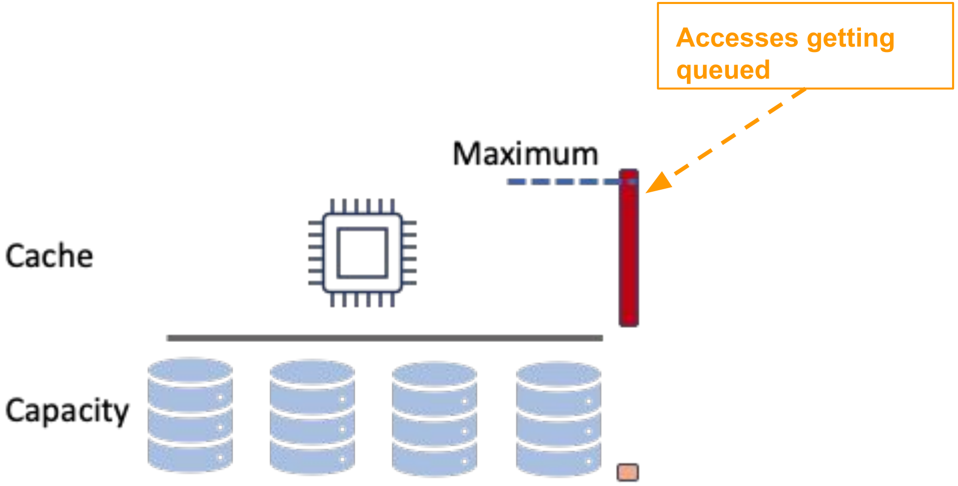
Our Approach:
Non-Hierarchical Caching (NHC)

- Insight: we should treat modern hierarchy in a less hierarchical manner
  - The available performance in capacity devices should be exploited

- Key idea: augmenting caching with dynamic load **admission** and request **offloading**

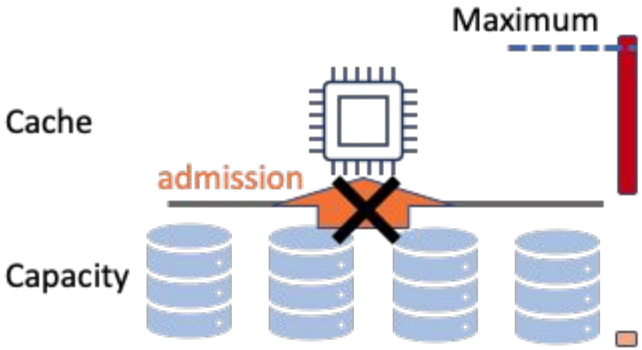
Augmenting Caching with Dynamic Load Admission and Request offloading

- **Intuition:** we should avoid **excess load** to cache device when it is **saturated**



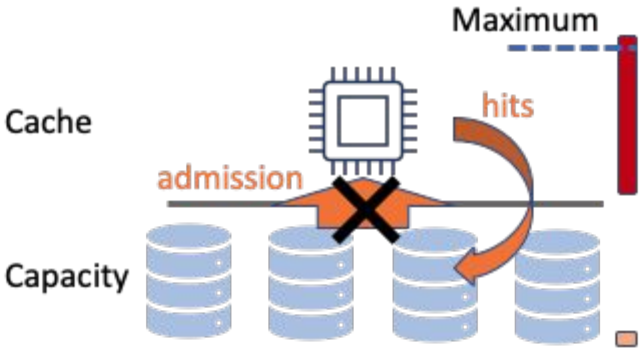
Augmenting Caching with Dynamic Load Admission and Request offloading

- **Intuition:** we should avoid **excess load** to cache device when it is **saturated**
- **Excess load examples:**
  - Data admission to further improve hit rate



Augmenting Caching with Dynamic Load Admission and Request offloading

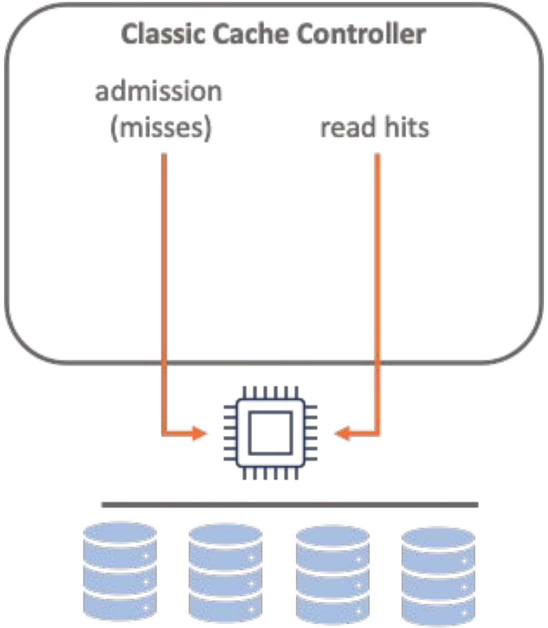
- **Intuition:** we should avoid **excess load** to cache device when it is **saturated**
- **Excess load examples:**
  - Data admission to further improve hit rate
  - Too many cache hits

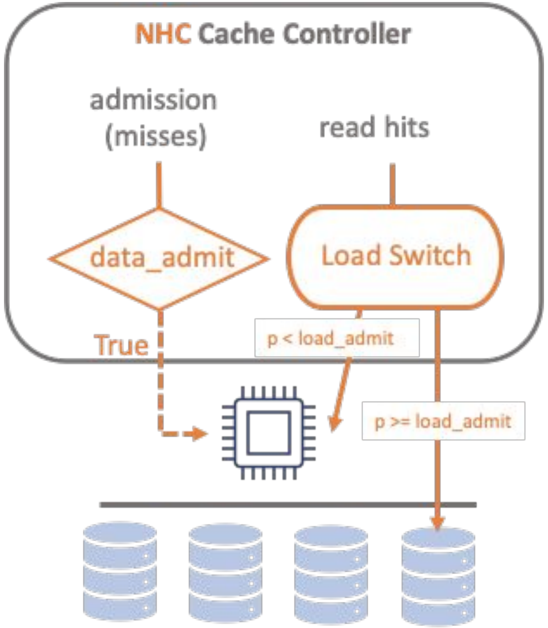


Design: Non-Hierarchical Caching

Design: Non-Hierarchical Caching

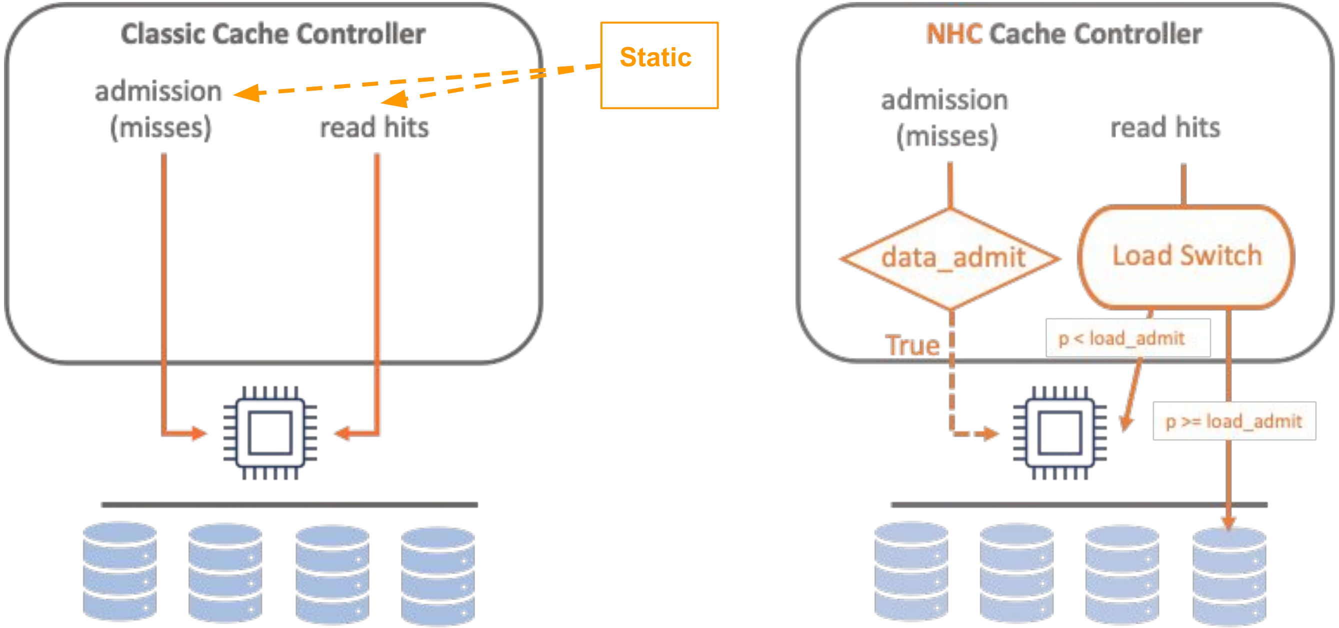
- Enable offloading: **tunable caching behaviors**
  - Classic caching is (data_admit = true, load_admit_ratio = 100%)





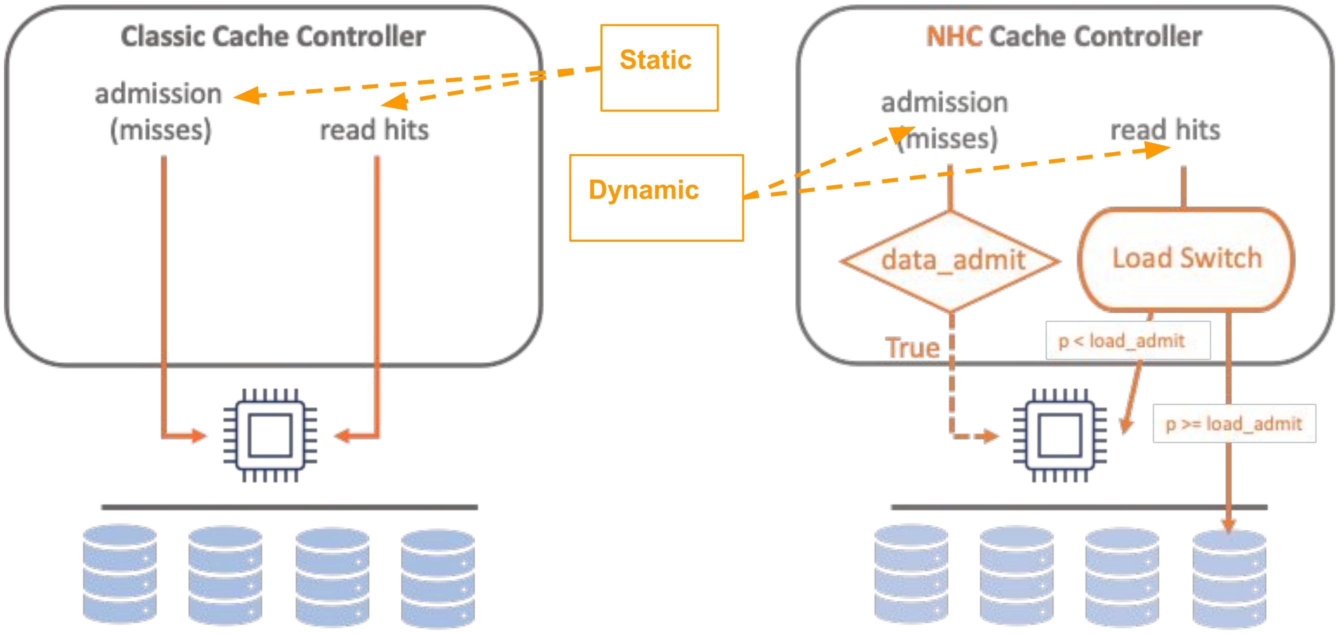
Design: Non-Hierarchical Caching

- Enable offloading: **tunable caching behaviors**
  - Classic caching is (data_admit = true, load_admit_ratio = 100%)



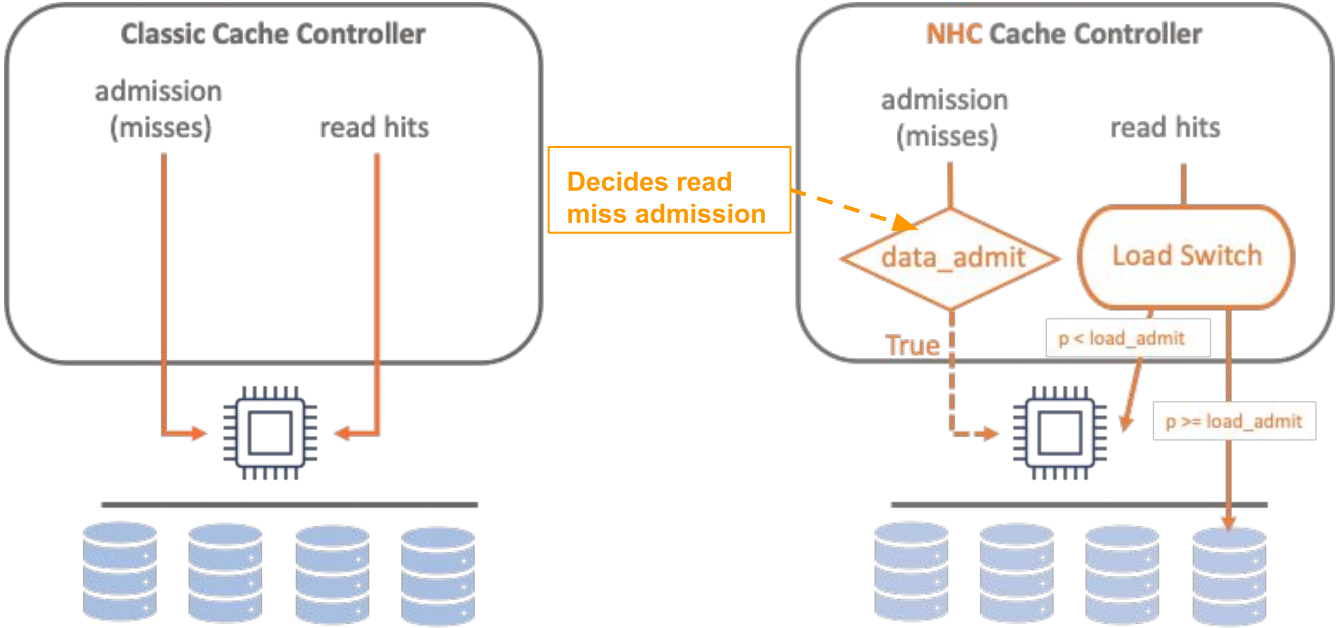
Design: Non-Hierarchical Caching

- Enable offloading: **tunable caching behaviors**
    - Classic caching is (data_admit = true, load_admit_ratio = 100%)



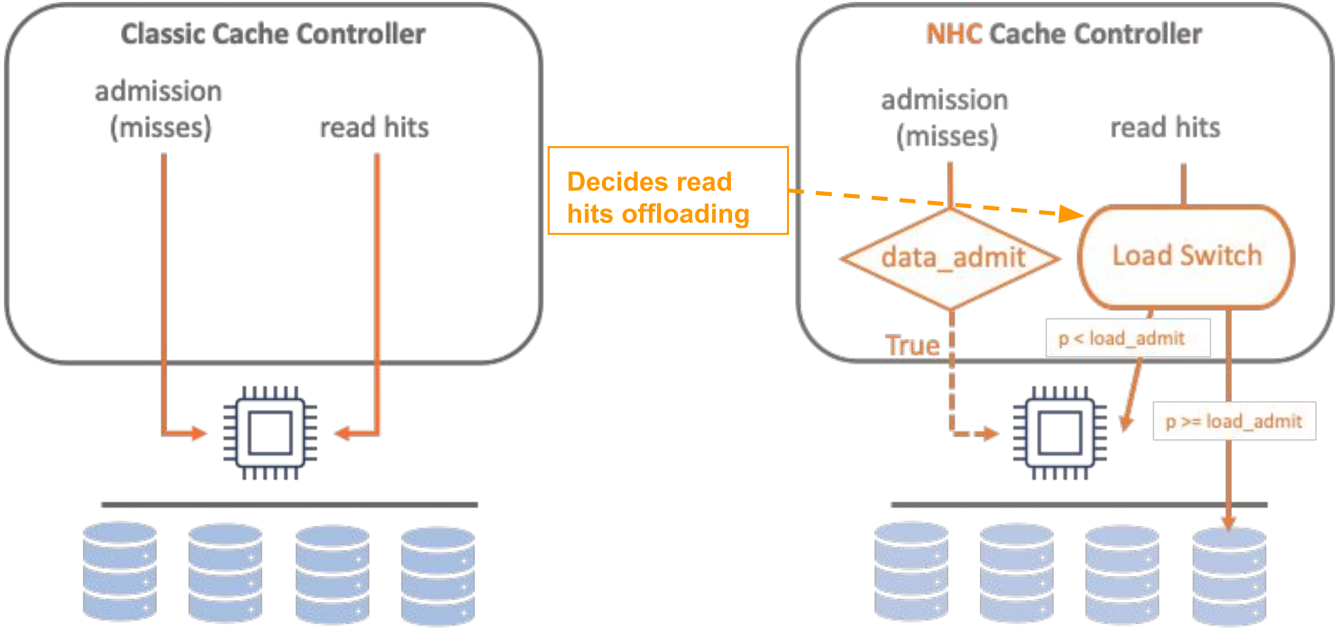
Design: Non-Hierarchical Caching

- Enable offloading: **tunable caching behaviors**
  - Classic caching is (data_admit = true, load_admit_ratio = 100%)



Design: Non-Hierarchical Caching

- Enable offloading: **tunable caching behaviors**
  - Classic caching is (data_admit = true, load_admit_ratio = 100%)

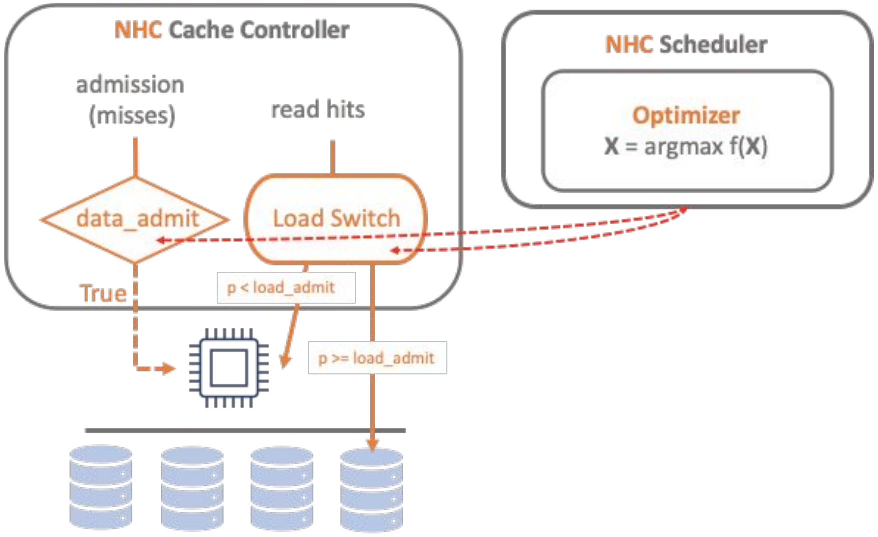


How much load to offload?

- **Observation**: different hierarchies, different workloads desire different split of load to devices (for best performance)
- Handle complexities: **feedback-based cache scheduler**

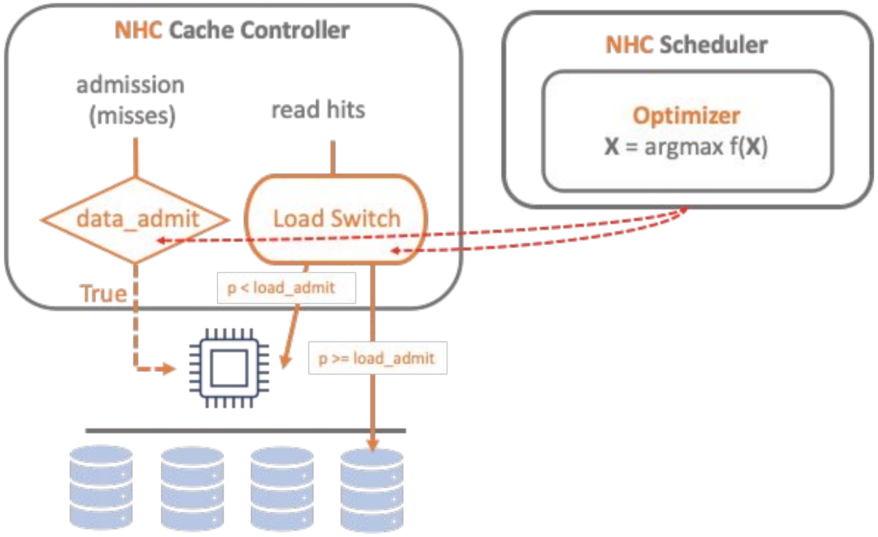
Design: Non-Hierarchical Caching

- feedback-based cache scheduler
  - Adjust tuning knobs (e.g., data_admit flag, load_admit ratio)



Design: Non-Hierarchical Caching

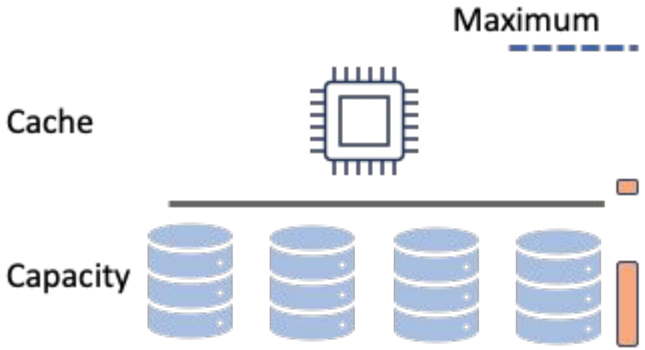
- feedback-based cache scheduler
    - **Optimize a target performance metric**
    - Target metric: user/device; throughput/ latency/ tail latency
        - **f(X):** a function to measure/compute the target metric



Example NHC Scheduling States

Example NHC Scheduling States

- State 1: begin with classic caching



Example NHC Scheduling States

- State 1: begin with classic caching
  - Ends when hit rate becomes “stable”

Example NHC Scheduling States

- State 2: adjust load between devices

Example NHC Scheduling States

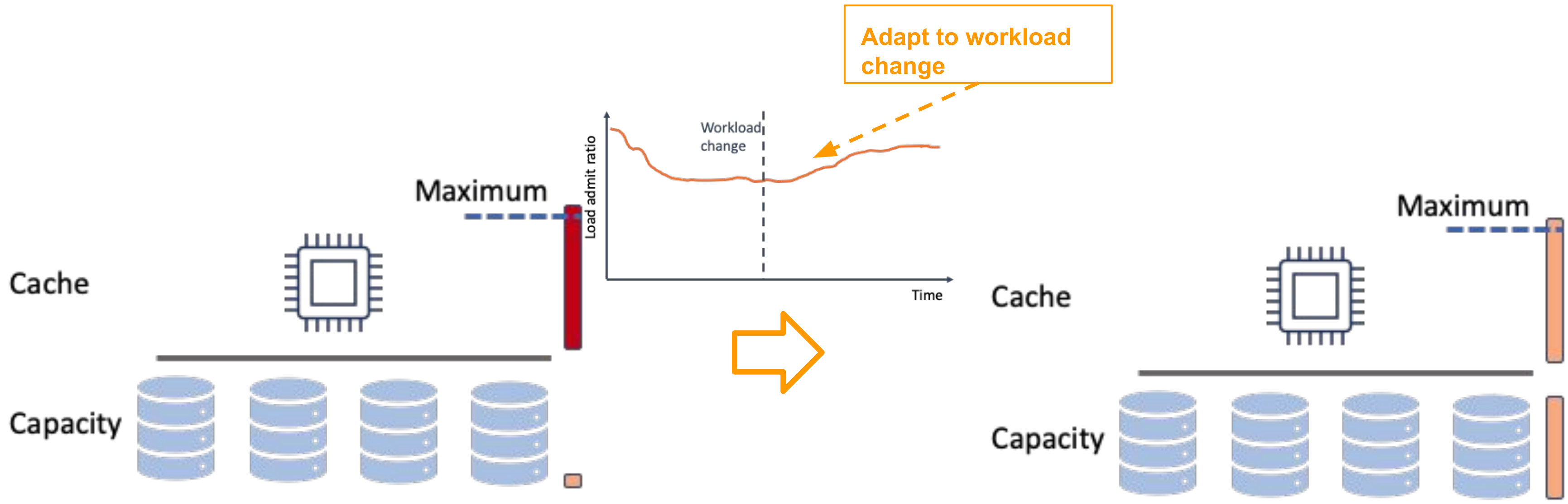
- **State 2: adjust load between devices**
  - Turn off data admission for read misses
  - Start to tune load admit ratio (base point 100%)

Example NHC Scheduling States

- **State 2: adjust load between devices**
  - Turn off data admission for read misses
  - Start to tune load admit ratio (base point 100%)

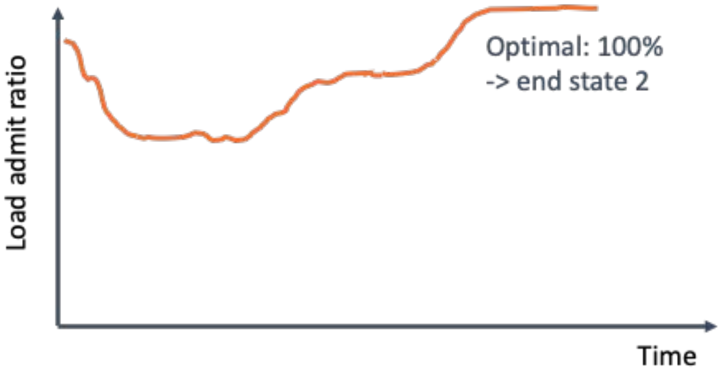
Example NHC Scheduling States

- State 2: adjust load between devices



Example NHC Scheduling States

- **State 2: adjust load between devices**
  - **End state 2 when: -> back to State 1 (classic caching)**
    - Workload hit rate significantly changed
    - Find 100% load admit rate is always optimal



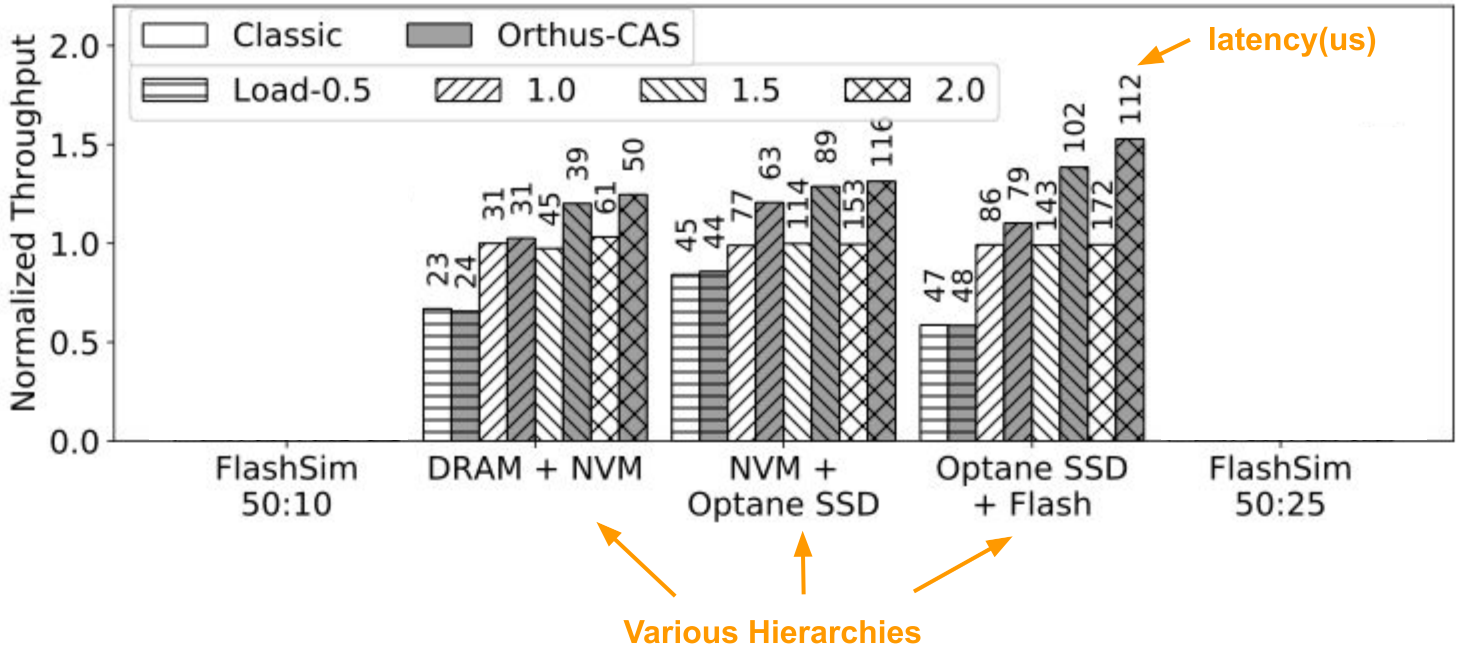
NHC - Key Properties

- Compatible with all classic caching implementations/ policies
- Require no prior knowledge of devices and workloads
- Robust to dynamic workloads

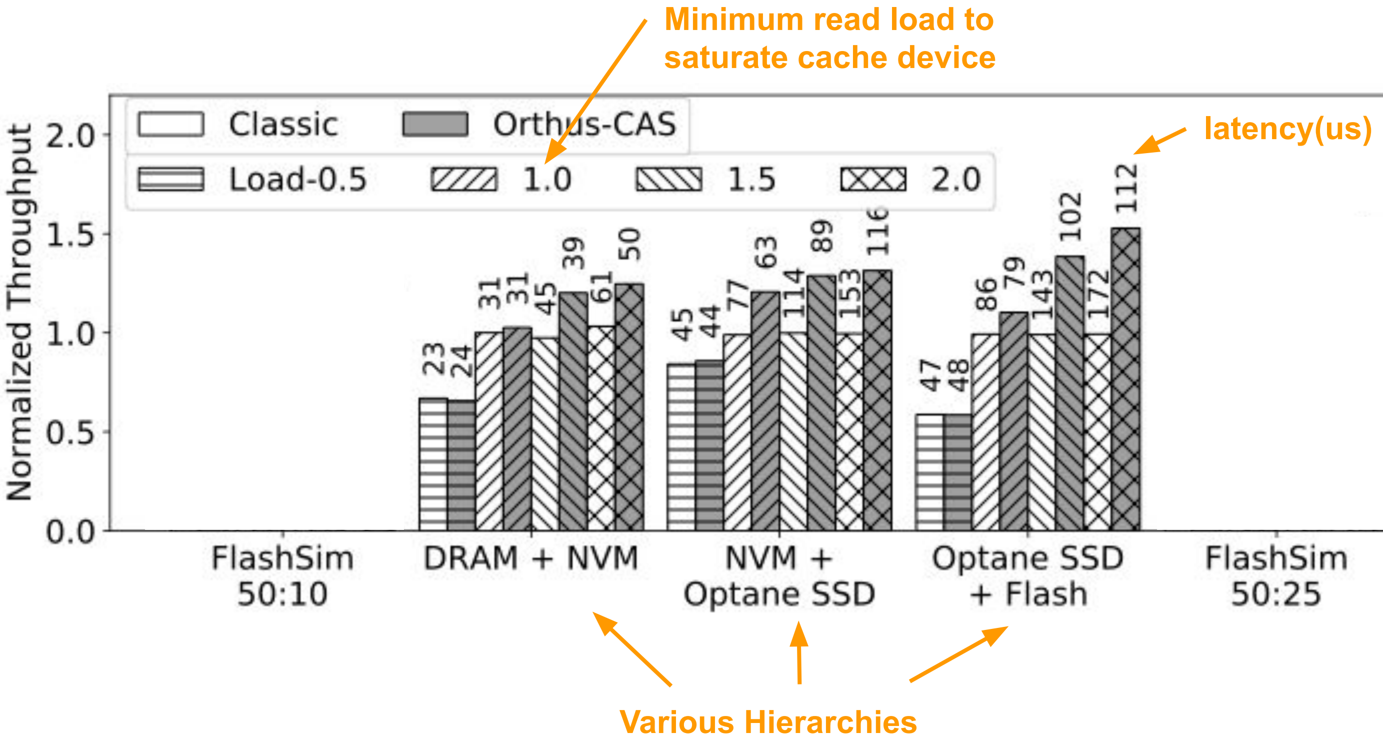
Implementation

- **Implementation (Orthus):**
  - **Orthus-CAS:** block-layer caching kernel module, based on Intel Open CAS framework
  - **Orthus-KV:** user-level caching layer for Wisckey [FAST' 16] (a LSM-tree based K/V store)
- **Supported target metrics:**
  - Throught
  - Avg. latency
  - P99 latency
- **Evaluated hierarchies:**
  - DRAM/Optane DC PM
  - Optane DC PM/Optane SSD
  - Optane SSD/Flash SSD

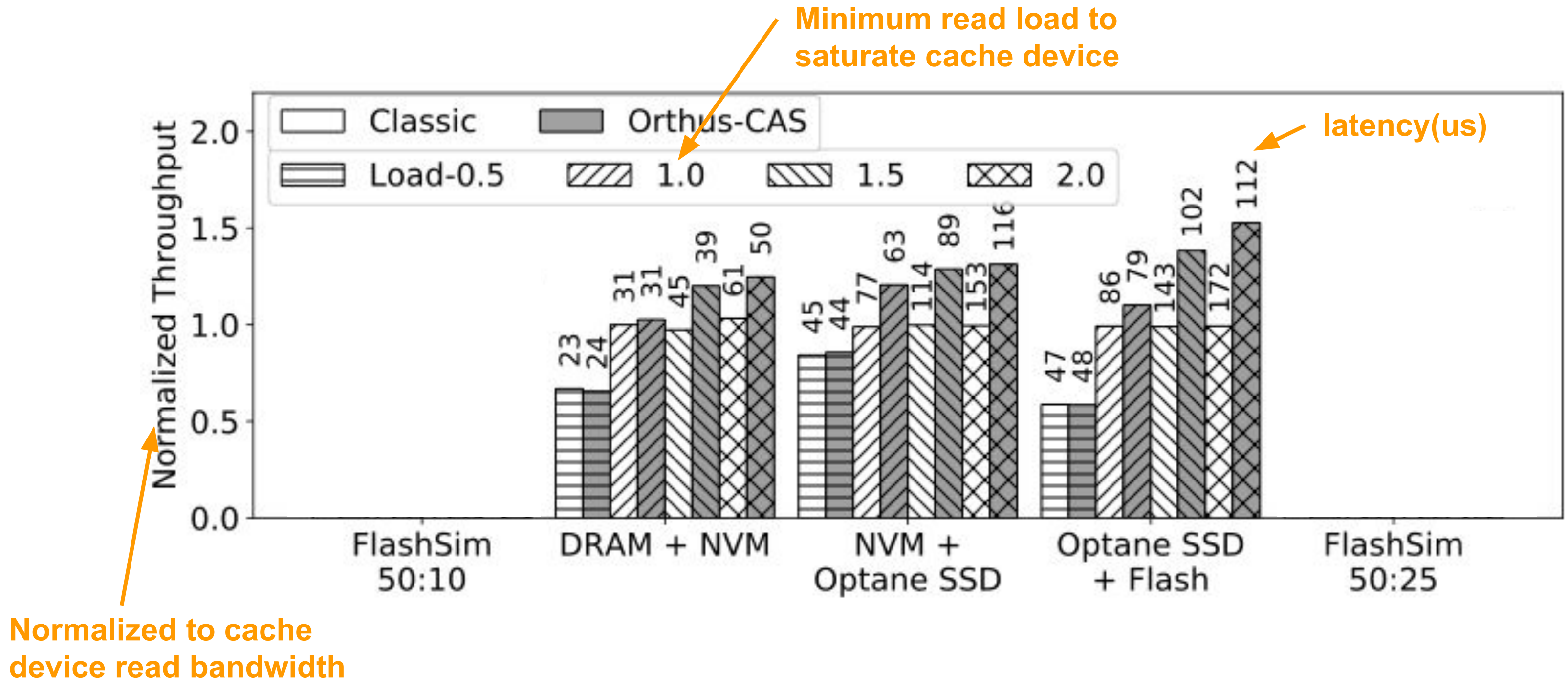
Ability to utilize capacity device performance



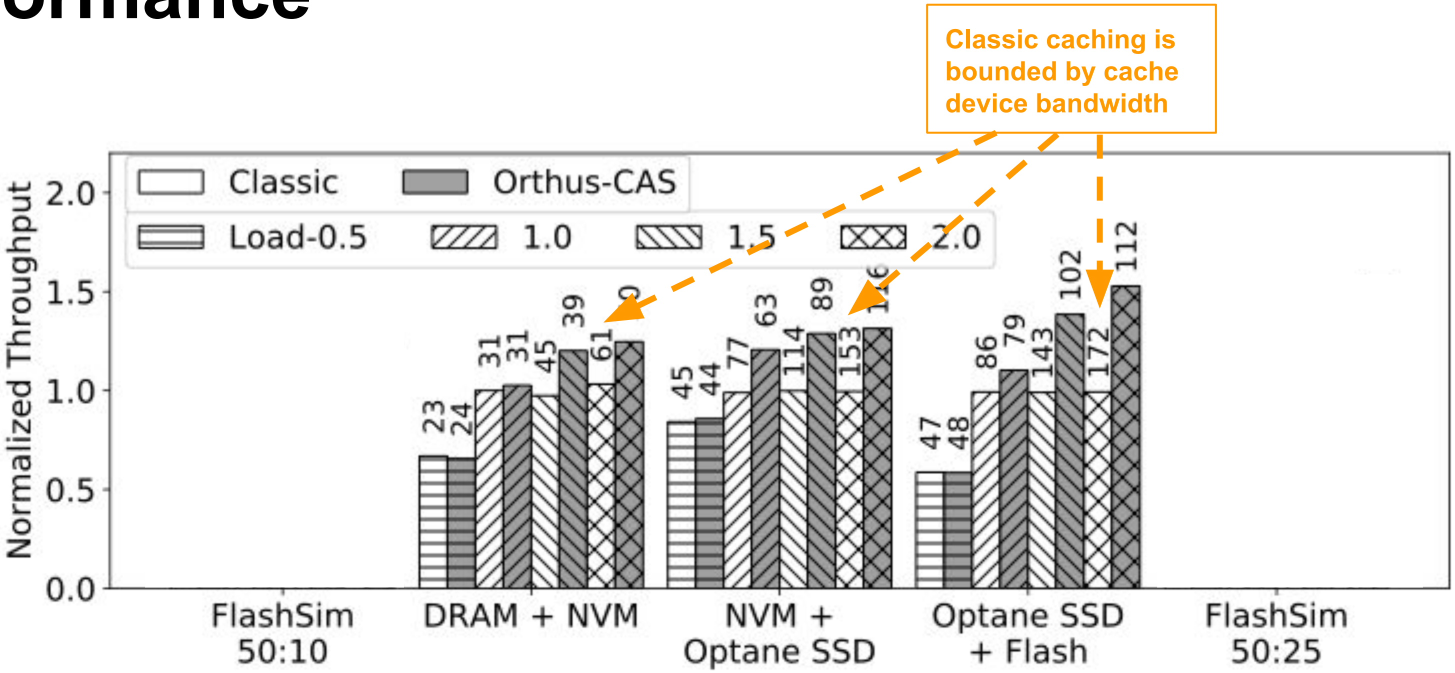
Ability to utilize capacity device performance



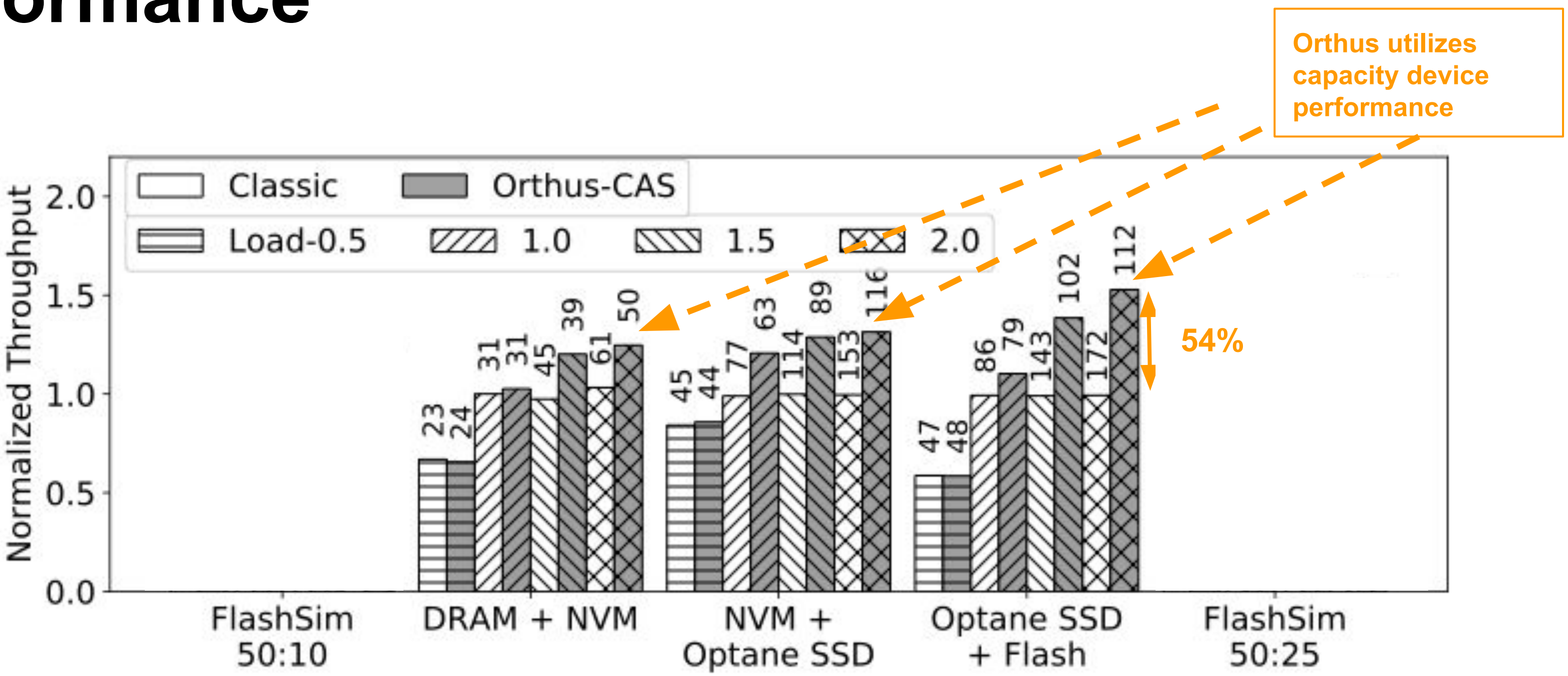
Ability to utilize capacity device performance



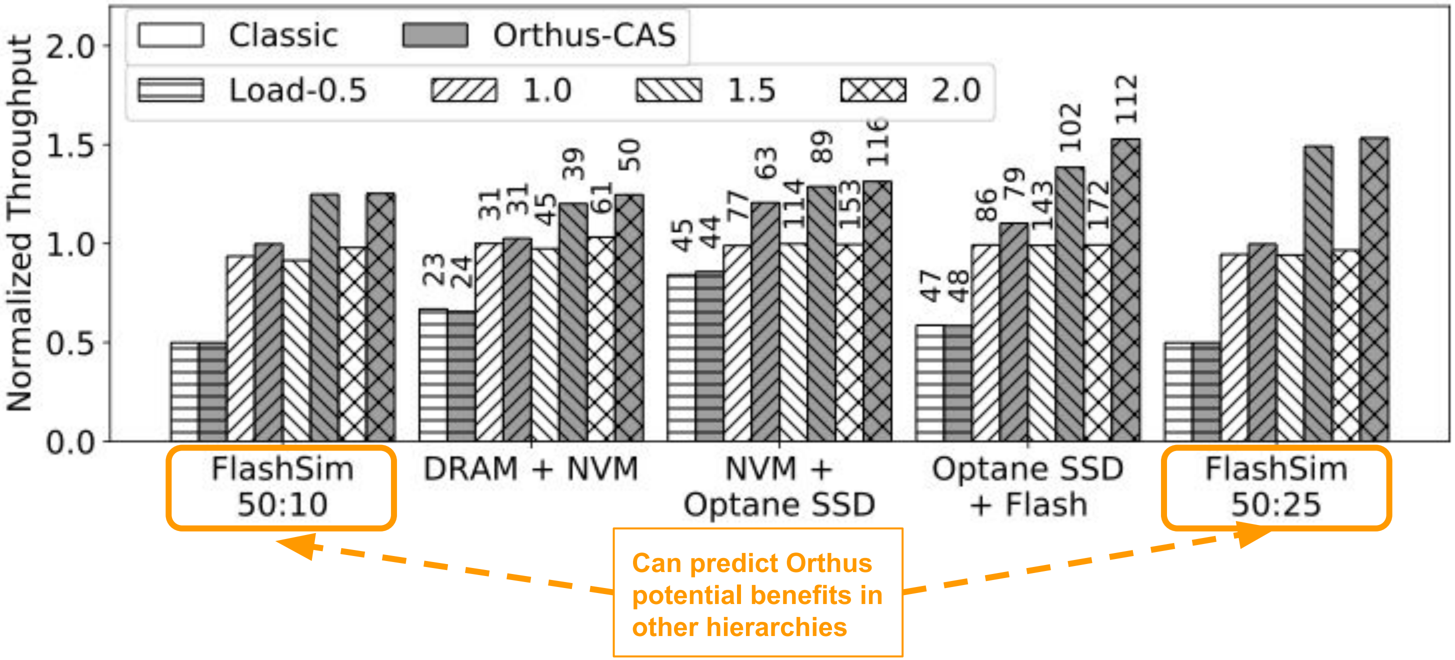
Ability to utilize capacity device performance



Ability to utilize capacity device performance



Ability to utilize capacity device performance



Other Experiments in the Paper

- Orthus improves with various caching **policies**
- Orthus optimizes different **target metrics** (e.g., tail latency)
- Orthus improves YCSB workloads
- Orthus improves **dynamic workloads**, such as Facebook ZippyDB workloads [FAST' 20]
- ...

Conclusion

- Evolving storage hierarchies have strong implications for caching
  - Quantitative comparisons across modern storage devices
  - Characterizing caching performance in both classic and modern hierarchies
- Orthus optimizes classic caching, by dynamic load admission and request offloading
  - Is compatible with all classic caching policies
  - Requires no prior knowledge of devices and workloads
  - Adapts to dynamic workloads

  - Can improve performance (throughput, tail latency) by up to 2X over classic caching in various storage hierarchies, under a range of realistic workloads

Thank you
&
Questions?

Contact: kanwu@cs.wisc.edu