Erasing Belady's Limitations:
In Search of Flash Cache Offline Optimality

Yue Cheng[1,2] Fred Douglis[2] Philip Shilane[2] Michael Trachtman[2]
Grant Wallace[2] Peter Desnoyers[3] Kai Li[4]

[1]*Virginia Tech* [2]*EMC* [3]*Northeastern University* [4]*Princeton University*

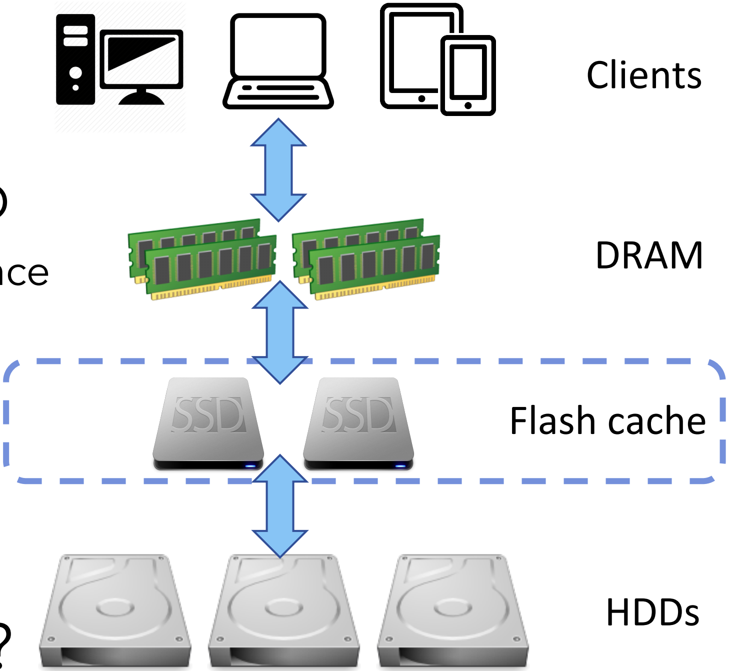
Background

- Unlike magnetic HDDs, flash SSDs erase data in large blocks
- Limited number of erasures before errors occur

… Want to make effective use of SSDs while being cognizant of erasure penalty

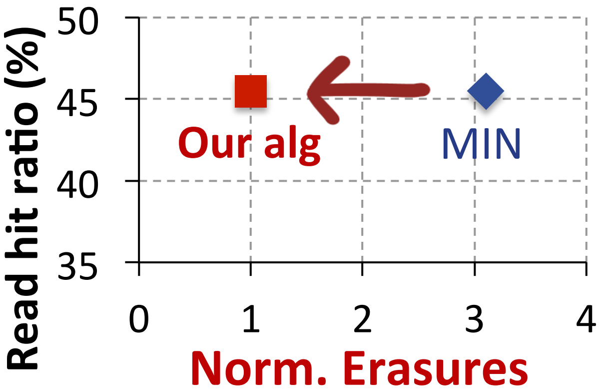
Flash Caching

- Using SSDs for an HDD cache
  - SSD cache as a tier btw. DRAM and HDD
    - Goal: to balance performance against endurance
  - Nitro [USENIX ATC '14], CacheDedup [USENIX FAST '16]
  - RIPQ [USENIX FAST '15]
  - Pannier [ACM Middleware '15]
- How do we know if we're doing well?
  - Comparison against an offline optimal **"best case"**
  - But what is the offline optimal for flash caches?



Offline Optimality

- Belady's MIN: A simple offline caching algorithm when the next access is known
  - Inserts a new entry into the cache, removing the entry that will be used furthest in the future
  - Yields the optimal read hit ratio (RHR)
- MIN is not able to provide the optimal erasures in the context of flash caching
  - MIN inserts data that ***won't actually be read***



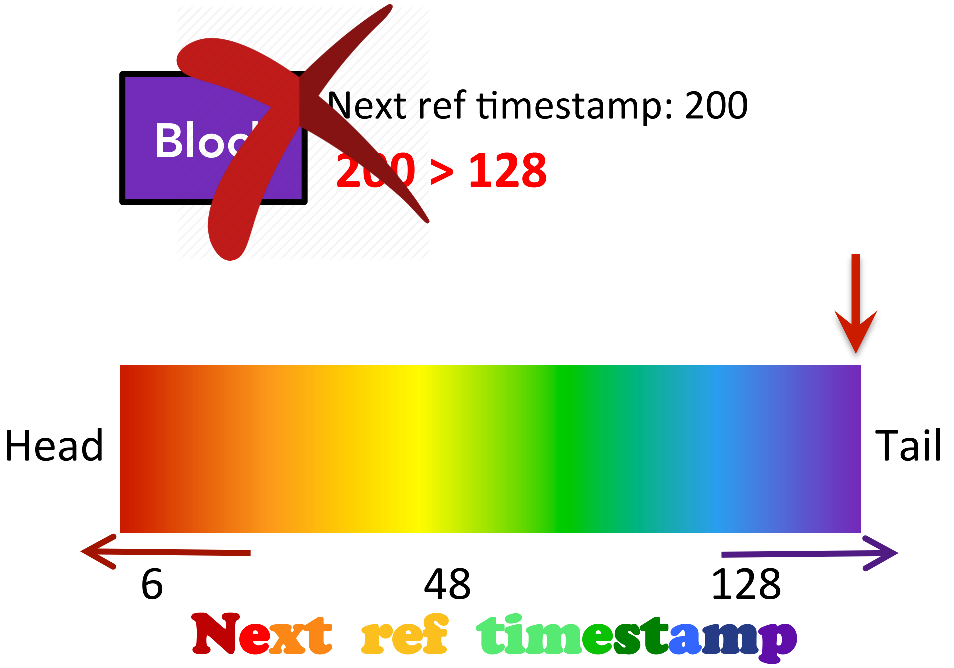
Flash Cache Offline Optimality

- Objectives:
  - O1: Minimize erasures s.t. maximal RHR
    - Never insert items if it does not increase RHR
  - O2: Maximize RHR s.t. an endurance limit
  - O3: Maximize combination of RHR and erasures
- True optimal
  - How to compute the offline optimal?
  - What is the complexity?
- Heuristics
  - How can we approximate?

The focus of our work

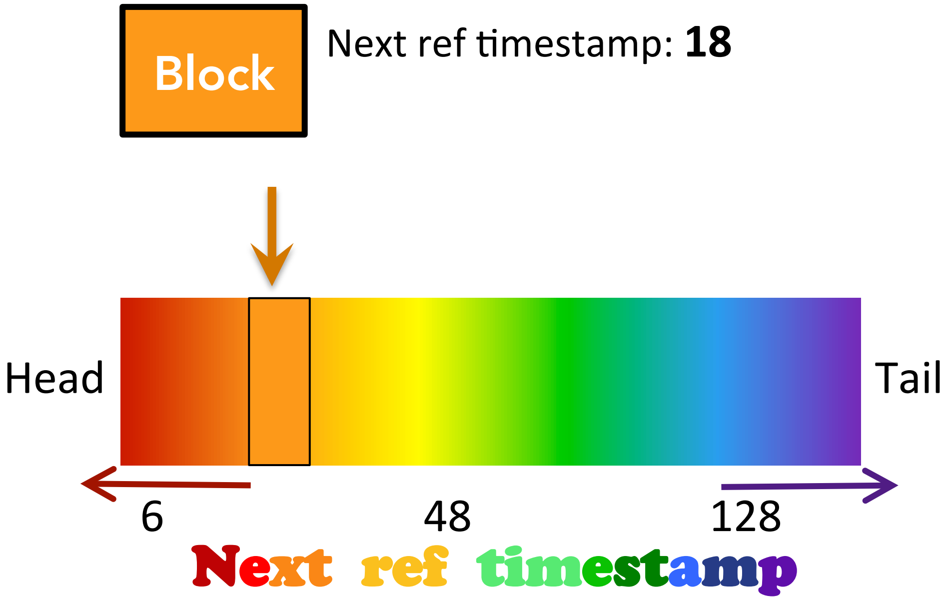
MIN

- **Blocks**
  – Minimum unit of access to the cache (e.g., 4KB)
- **MIN priority queue** (ranked based on next ref. timestamp)
  1. Don't insert blocks w/ furthest next ref.



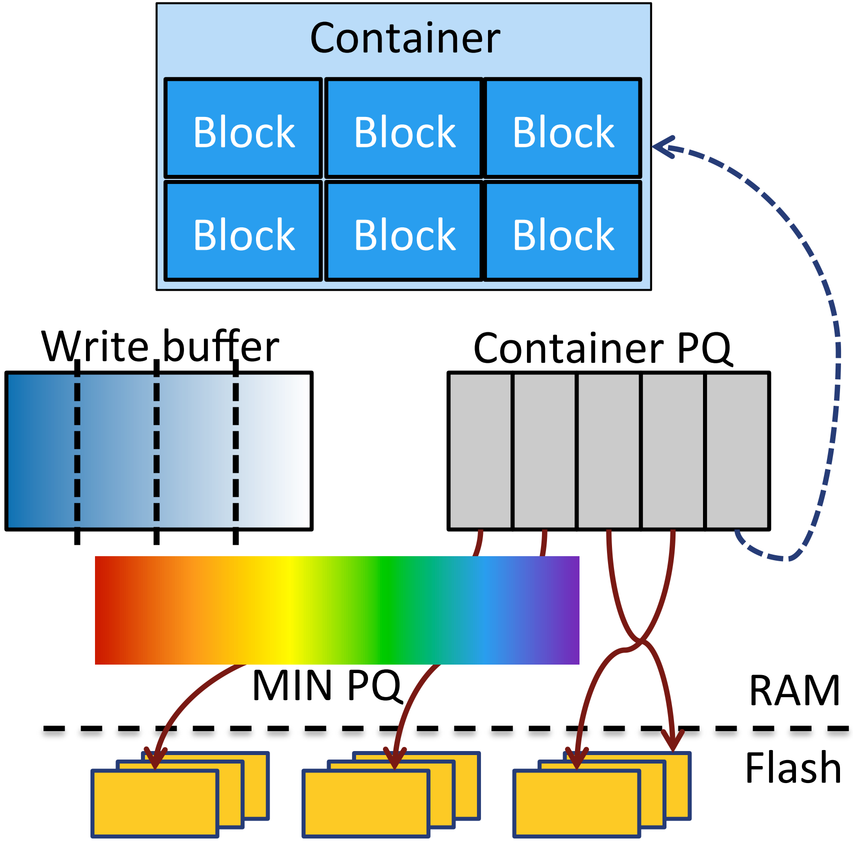
MIN

- **Blocks**
  - Minimum unit of access to the cache (e.g., 4KB)
- **MIN priority queue** (ranked based on next ref. timestamp)
  1. Don't insert blocks w/ furthest next ref.
  2. A new block is inserted



Key Components

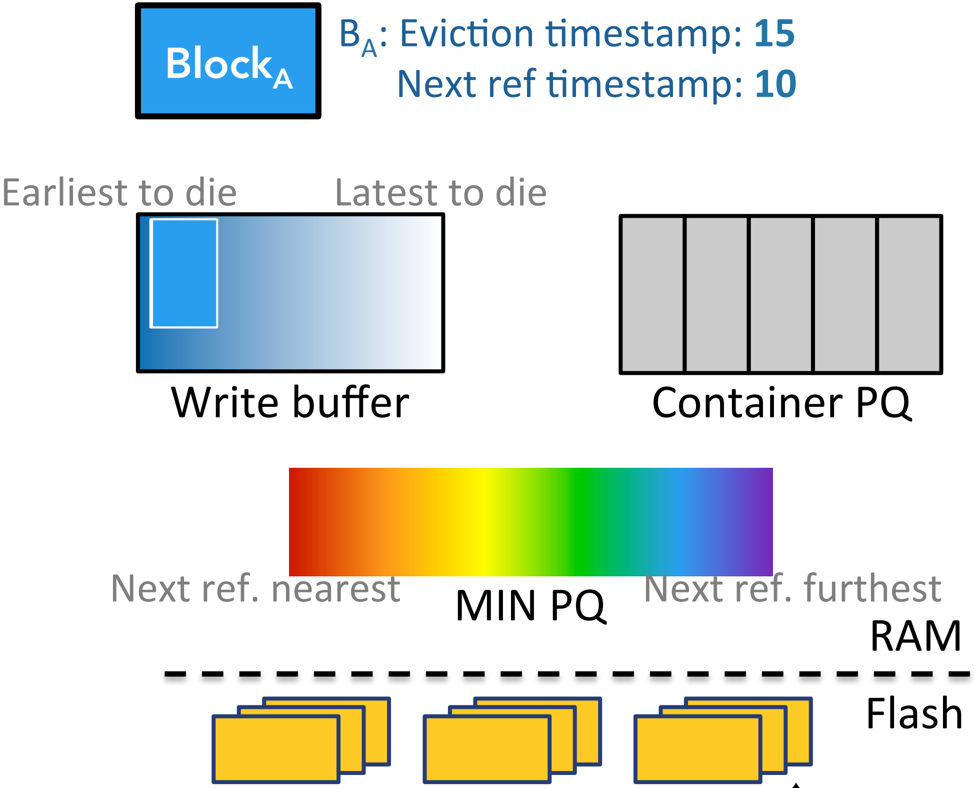
- **MIN priority queue**
  - In-RAM structure tracking runtime status at block granularity
- **Containers**
  - Unit of insertion & eviction for flash cache (e.g., 2MB)
- **Write buffer**
  - Packs blocks into in-RAM write buffer (e.g., 8MB)
- **Container priority queue**
  - In-RAM structure tracking flash containers



VirginiaTech EMC² Northeastern PRINCETON UNIVERSITY

Container-optimized Heuristic

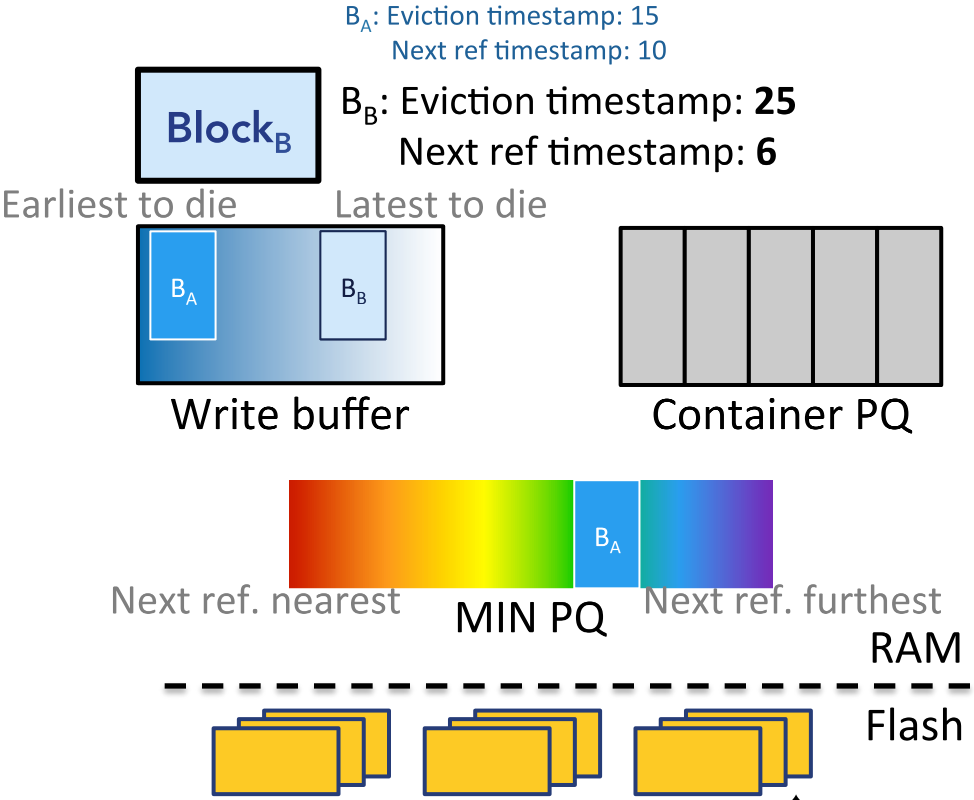
- Insert a block
  - Into the write buffer (sorted by *eviction timestamp*)
  - Into the MIN queue (sorted by *next ref. timestamp*)



VirginiaTech EMC² Northeastern PRINCETON UNIVERSITY

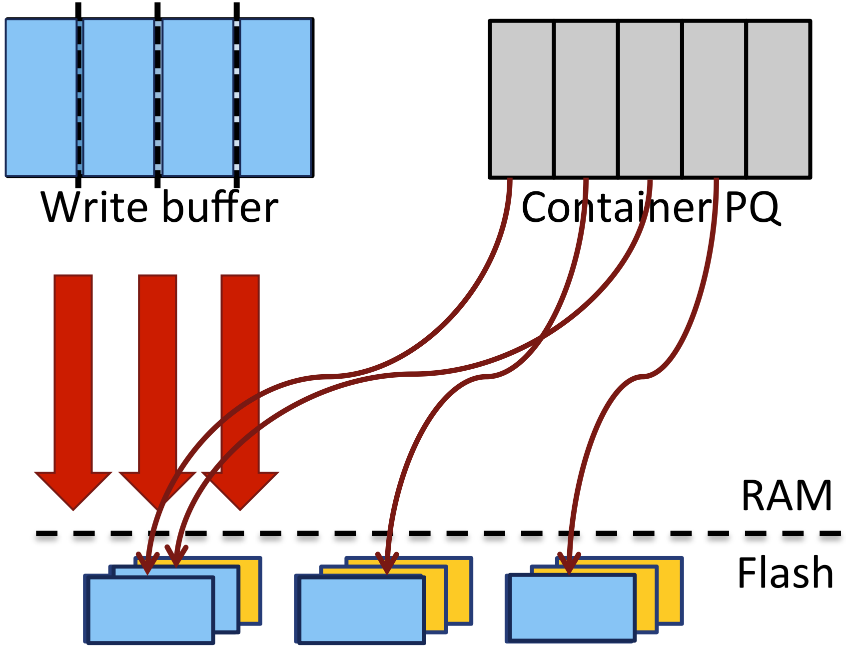
Container-optimized Heuristic

- Insert a block
  - Into the write buffer (sorted by *eviction timestamp*)
  - Into the MIN queue (sorted by *next ref. timestamp*)



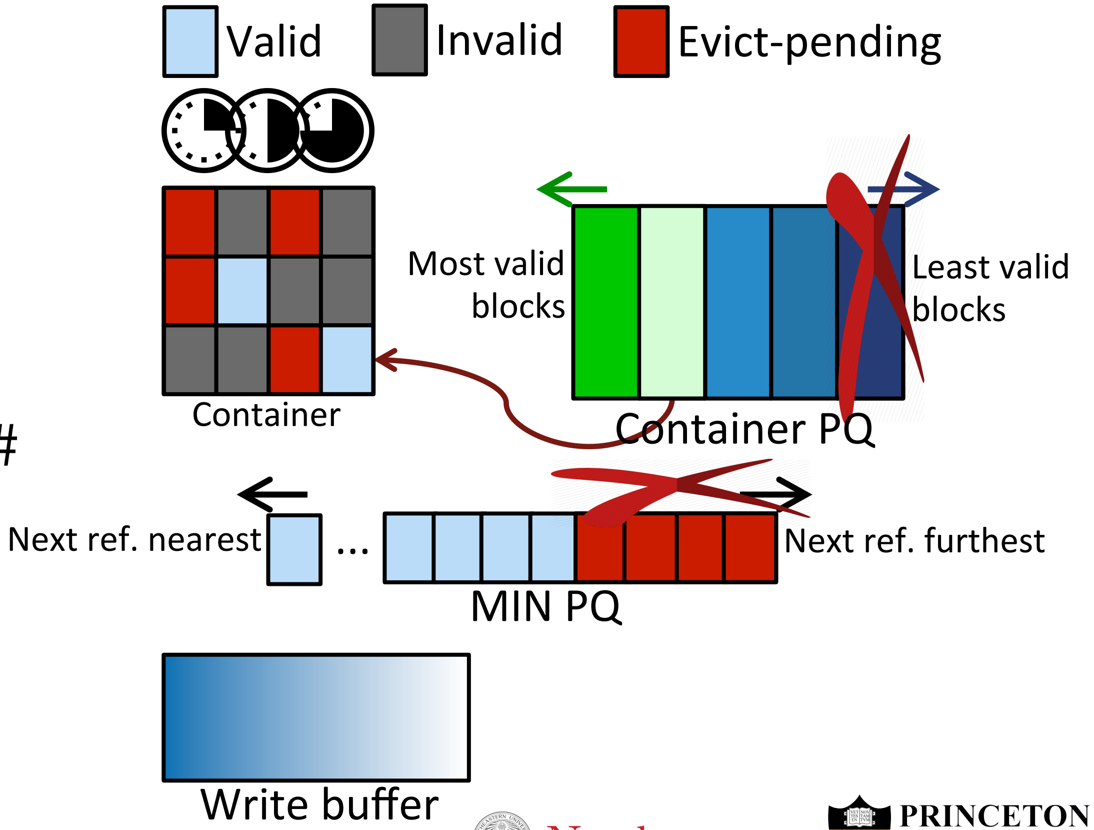
Container-optimized Heuristic

- Write buffer is dispersed into containers when it is **FULL**
- The containers are written to the flash cache



Container-optimized Heuristic

- Update block status on **invalidation** or when MIN would evict block as furthest in future ("**evict-pending**")
  - Remains in container until that is GC'd
- Rank container queue by # valid blocks
- Evict the tail container to make room for new data
- Copy forward valid blocks to the write buffer



Optimizations

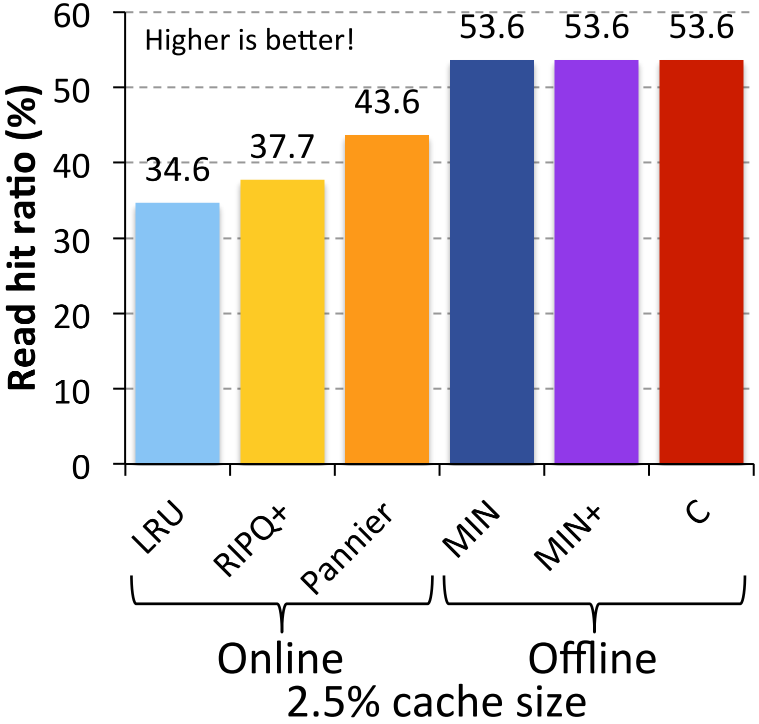
- **R_1:** Only insert blocks read at least once before eviction
- **TRIM:** Skips **dead** blocks during GC
  - Dead: overwritten or never reaccessed
- **Copy-forwarding reduction:** Eliminates wasted CF blocks
- **E:** Segregates blocks by eviction timestamp

Experimental Methodology

- Storage traces (34): limit to large enough datasets
  - EMC VMAX: 25
  - MSR Cambridge: 3
  - MS Production Servers: 6
- Implementation
  - Full-system flash cache simulator
  - Vary cache size as function of unique data accessed in trace: 1-10%
- Metrics
  - Performance: Read hit ratio (RHR)
  - Endurance: Erasures per block per day (EPBPD)
  - Function of RHR and EPBPD: Weighted flash usage effectiveness (WFUE)

Comparing Algorithms

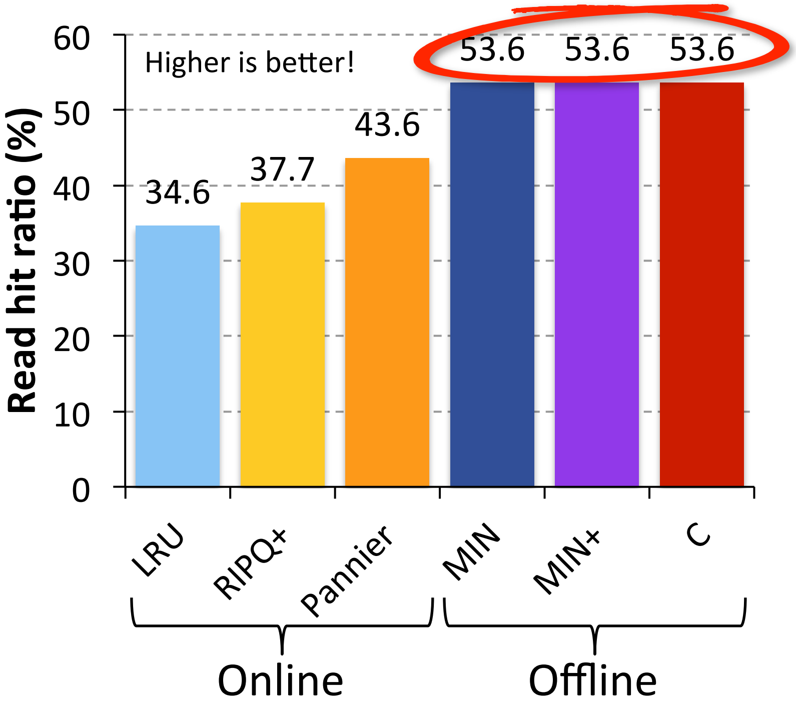
All 3 offline algorithms achieve the same (optimal) RHR



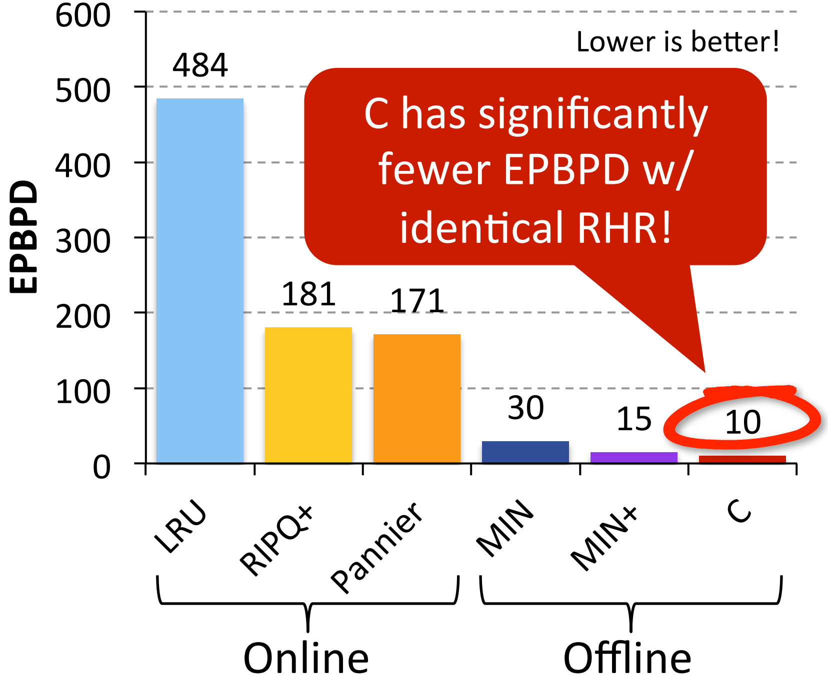
(O=Offline, C=Container-optimized)

Policy	Description	O	C
LRU	Least recently used	✗	✗
RIPQ+	Static web content	✗	✗
Pannier	Handles divergent containers	✗	✓
MIN	Don't insert data w/ furthest next ref	✓	✗
MIN+	Don't insert data evicted w/o read	✓	✓
C	Our container-optimized heuristic	✓	✓

Comparing Algorithms



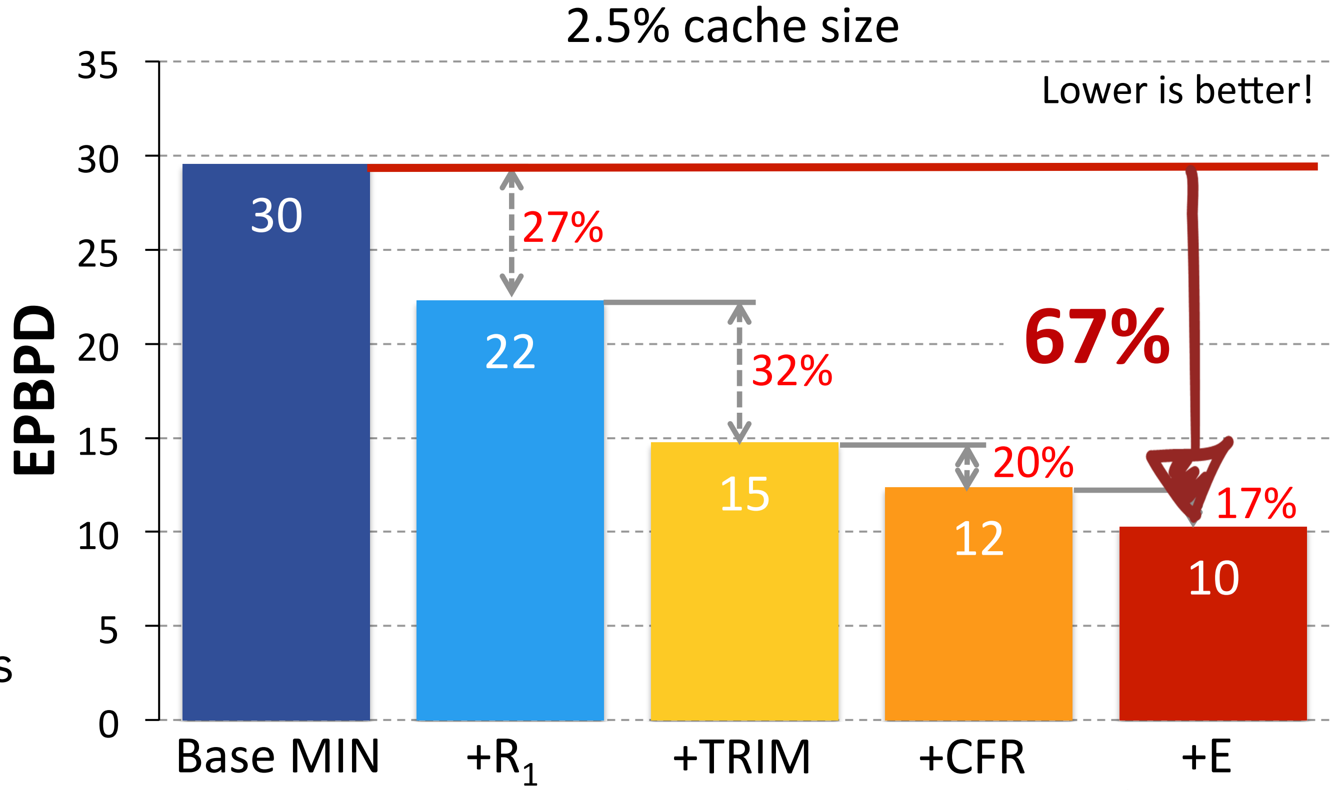
2.5% cache size



VirginiaTech EMC² Northeastern PRINCETON UNIVERSITY

Evaluating Heuristic Techniques

- **R_1:** Omit insertion with no reread
- **TRIM:** Notify GC to omit dead blocks
- **CFR:** Avoid wasted CF blocks
- **E:** Segregate blocks by evict timestamp

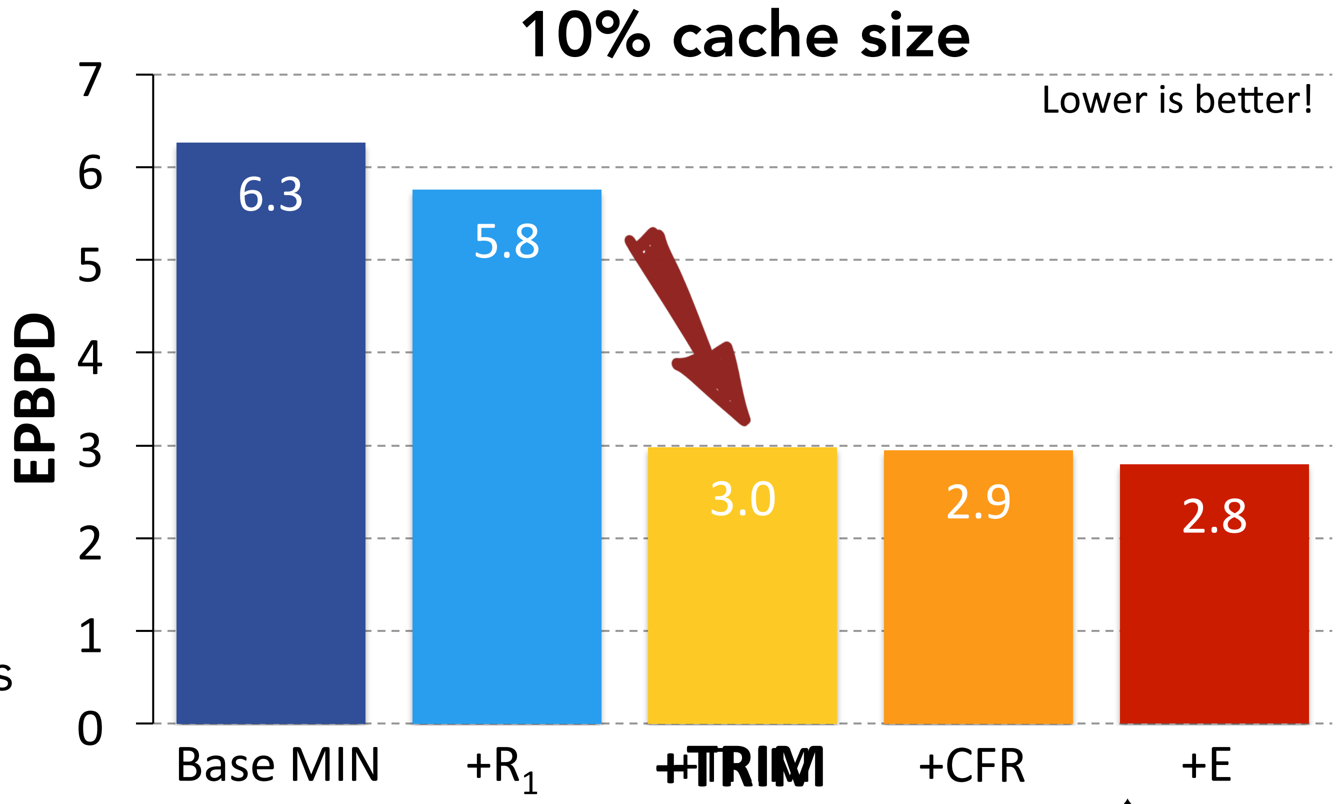


VirginiaTech EMC²

Northeastern PRINCETON UNIVERSITY

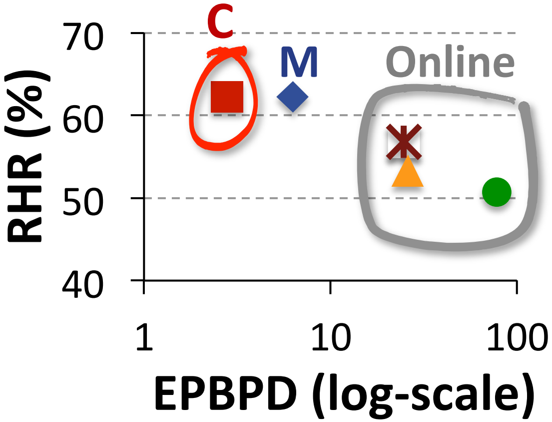
Evaluating Heuristic Techniques

- **R_1:** Omit insertion with no reread
- **TRIM:** Notify GC to omit dead blocks
- **CFR:** Avoid wasted CF blocks
- **E:** Segregate blocks by evict timestamp



Conclusion

- Important to have a baseline for the offline optimal considering both RHR and endurance
- Our container-optimized heuristic ***maintains the optimal RHR*** while reducing erasures by ***up to 67%***



- Additional optimizations may be possible to move this heuristic to the ***true*** optimal

Erasing Belady's Limitations:
In Search of Flash Cache Offline Optimality

Thank you! Q & A

Acknowledgments:

Dan Arnon, Cheng Li, Stephen Manley, Darren Sawyer, Dan Tsafrir, Kevin Xu, and Sanjeev Arora and his students from Princeton University

VirginiaTech EMC² Northeastern PRINCETON UNIVERSITY