Cache, Cache Everywhere, Does The Miss Rate Ever Shrink
Cache, Cache Everywhere, Flushing All Hits Down The Sink

On Exclusivity in Multilevel, Hybrid Caches

Raja Appuswamy, David C. van Moolenbroek,
Andew S. Tanenbaum

Vrije Univeristeit, Amsterdam

Traditional Flash Cache

- Assumptions
  - Memory is always volatile
  - |RAM| << |Flash|
    - Cost/GB (RAM) >> Cost/GB (Flash)
    - Cheaper to provision more flash
- Design Decision
  - Independently managed layers
  - Inclusive caching hierarchy (RAM/SSD)

Changes in the Hardware Landscape (1)

- DRAM price free fall (2011)
  - Tablet love, Windows Peak, Recession…
  - STEC ZEUSIOPS 200GB 2.5IN SAS MLC = $1,260.59[1]
  - 192GB using 16 GB DDR3 SDRAM = $1,300[2]
- Multi-core & more memory
  - 32-core, 100 GB servers now common place
  - RAMCloud, memcached widely used
- Volatility is subjective
  - Storage servers equipped with redundant PSUs
  - Enough residual power to checkpoint/restart
- Promise of Storage Class Memories
  - Scale, nonvolatile storage with performance ~ RAM

[1] http://www.kernelsoftware.com/products/catalog/stec.html , [2] www.newegg.com

Changes in the Hardware Landscape (2)

- Revisiting Assumptions

  ~~Memory is always volatile~~

  ~~Density (Flash) >> Density (RAM)~~

- Properties of next-generation caches
  - Multilevel and physical collocated
  - All levels are persistent
  - All levels of comparable density

Traditional Caching With Modern Hardware

- Inclusive caching reduces cache effectiveness
  - Same data cached in both RAM and SSD
- Well known problem in distributed caching
  - Redundancy between client/server caches

Exclusive Caching: Primer

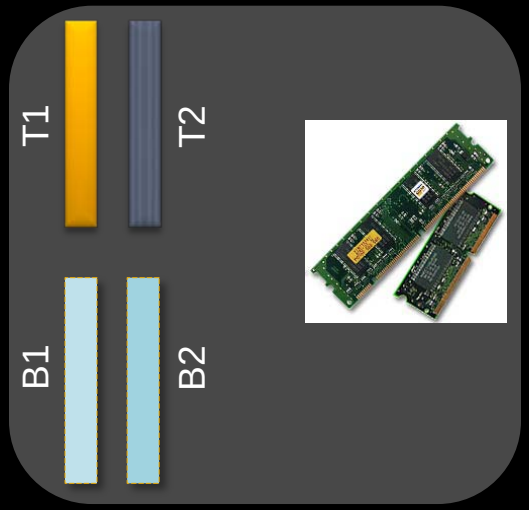
- Admission only into client cache on miss
  - Inclusive allocates in both server/client
- Admission into server cache on client eviction a.k.a “Demotions”
- Data removed from server on L2 hit
  - Inclusive caching classifies data as MRU
- However, maybe hard to realize
  - Need to unify management of physically distributed caches

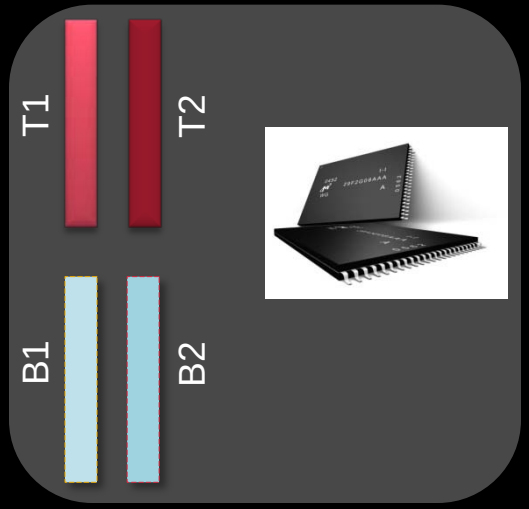
Exclusivity in Direct-attached Caches

- Client = RAM, Server = SSD
  - Easy to implement centralized caching
  - Demotions are cheap (no network)
    - But not free as each demotion = SSD write
  - Second level is flash based
    - SSD write endurance must be addressed
  - All levels are persistent
    - How do we build an exclusive read/write cache?

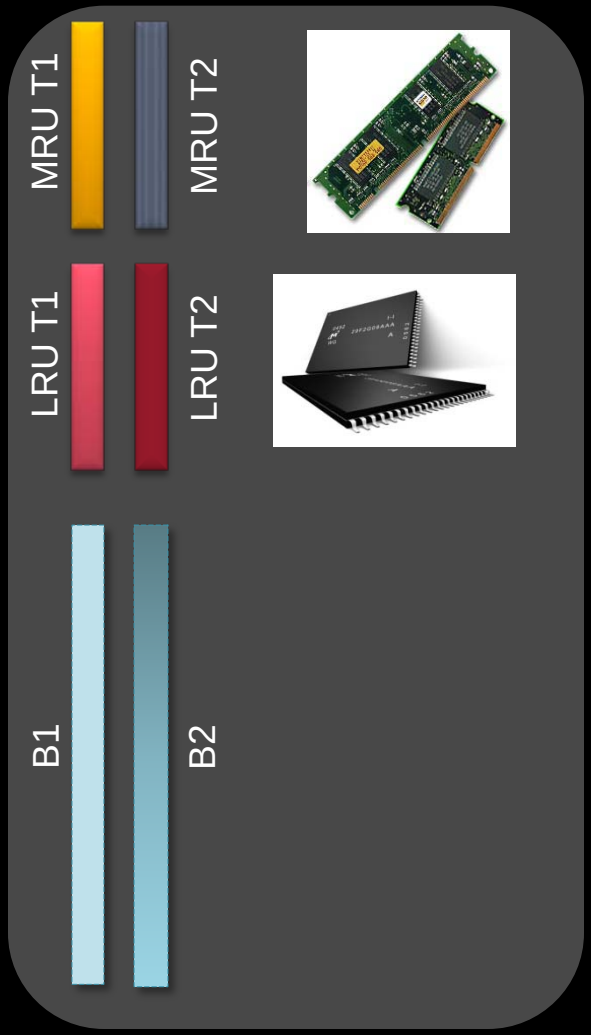
Inclusive vs Exclusive: Boundaries

Inclusive





Exclusive



Simulation Environment

- Trace-driven simulator
  - Block-level, ARC simulation
    - Supports Inclusive (IARC), and Exclusive ARC (UARC)
  - Measures hit rate at each level, execution time

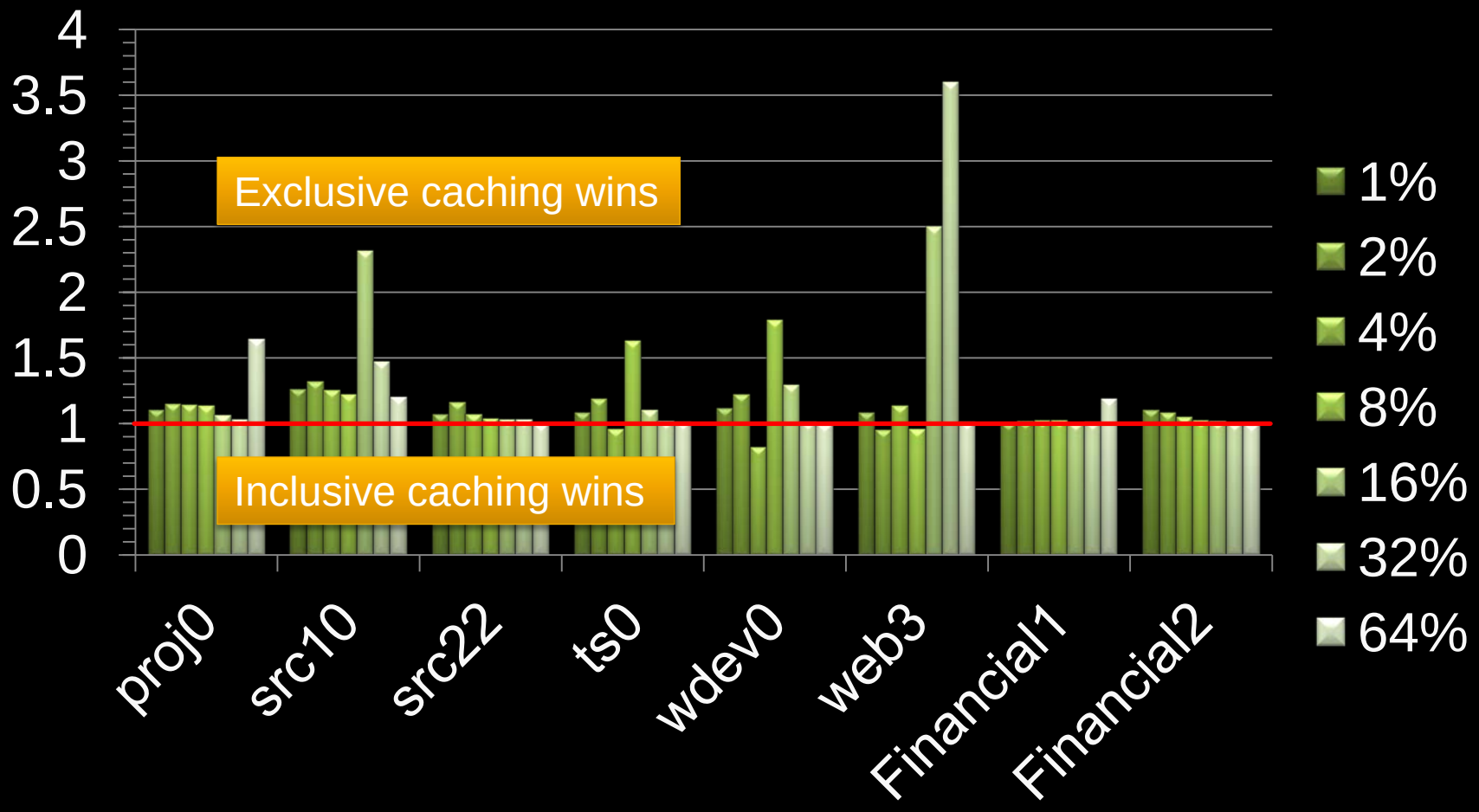
$$\text{Virtual Execution Time} = |\text{SSD read hits}| \times T_{SSDR} + |\text{SSD demotions}| \times T_{SSDW} + |\text{read misses}| \times T_{HDD} + |\text{flushes}| \times T_{HDD}$$

Traces

Trace	I/O (Millions)	Read (%)	Write (%)	R-WSS (GB)	RW-WSS (GB)
Financial1	36	15	85	1.11	3.66
Financial2	18	78	22	0.82	1.17
proj0	40	6	94	1.76	3.16
src10	406	47	53	120.76	121.26
src22	17	36	64	20.31	20.31
ts0	4	26	74	0.5	0.91
wdev0	3	27	73	0.2	0.53
web3	0.5	60	40	0.22	0.59

Is Exclusive Caching Beneficial?

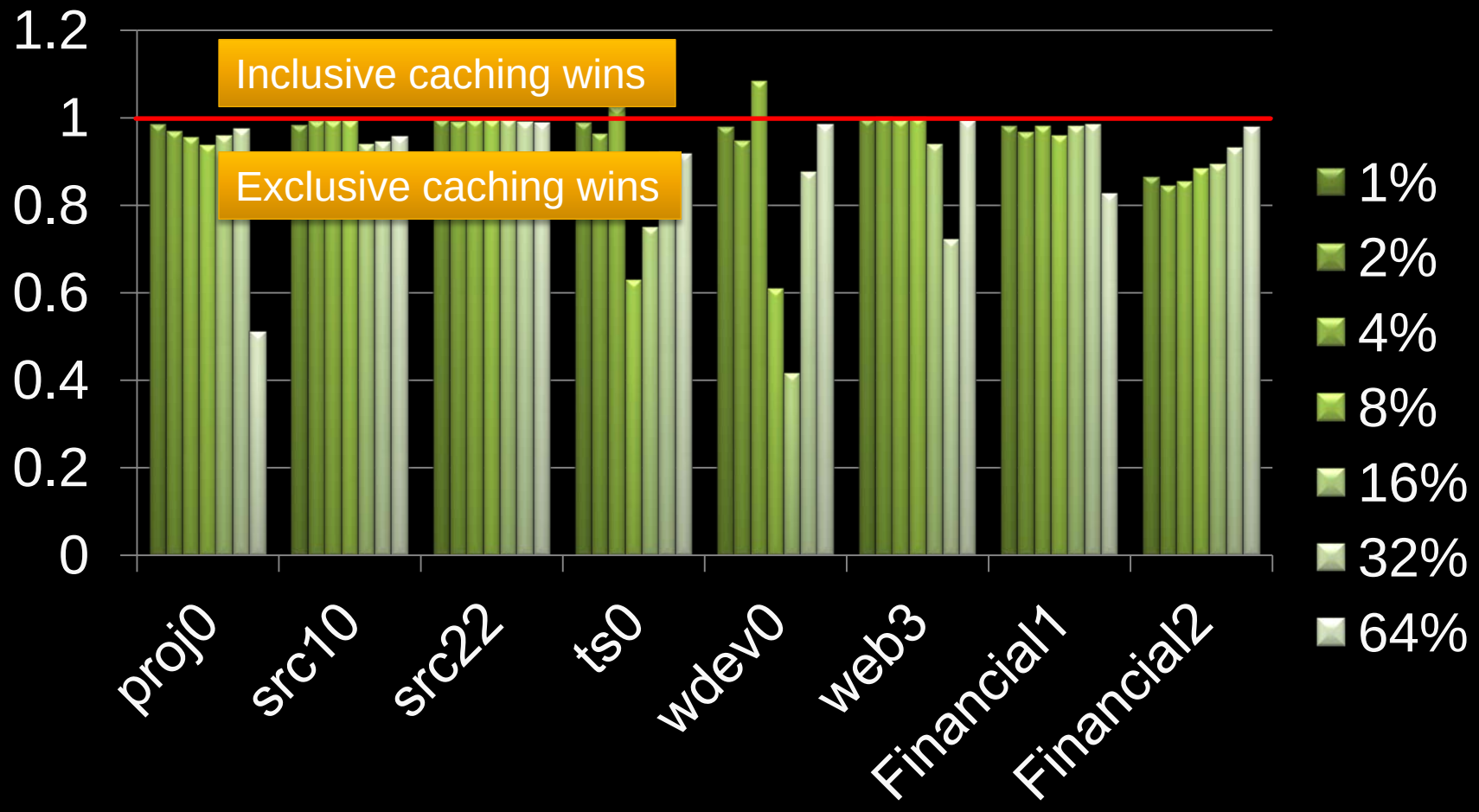
Normalized (wrt IARC) UARC Hit Rate



Exclusive caching improves cache effectiveness

Is Exclusive Caching Beneficial?

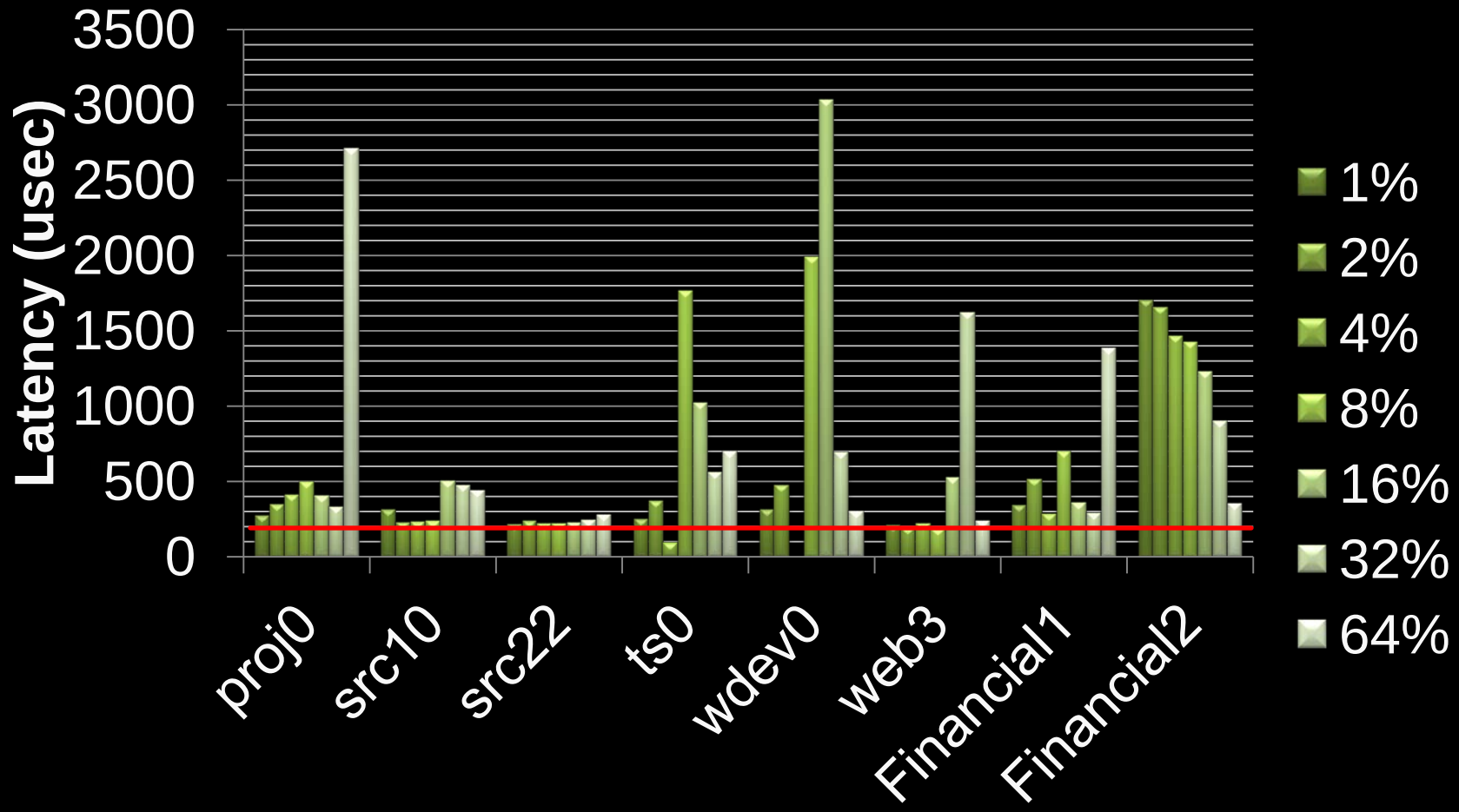
Normalized UARC Execution Time



Exclusive caching improves performance

SSD Performance/Cost Tradeoff

SSD Write Performance Impact



Exclusive caching enables cost/performance/lifetime tradeoffs

RAM/SSD Size Sensitivity

Trace	1%	2%	4%	8%	16%	32%	64%
Financial1	-	-	-	-	-	-	-
Financial2	-	-	10% (1.031)	10% (1.0216)	10% (1.0404)	20% (1.0543)	20% (1.0311)
hm0	-	20% (1.0132)	-	-	-	-	-
proj0	10% (1.0144)	10% (1.0113)	-	-	-	-	30% (1.0378)
src10	-	-	-	-	-	-	-
src22	-	-	-	-	-	-	-
web3	-	-	-	-	-	10% (1.0111)	40% (1.0189)

- In most cases, UARC is always better
  - "-" entries in the table

Exclusive caching is effective even at low RAM/SSD ratios

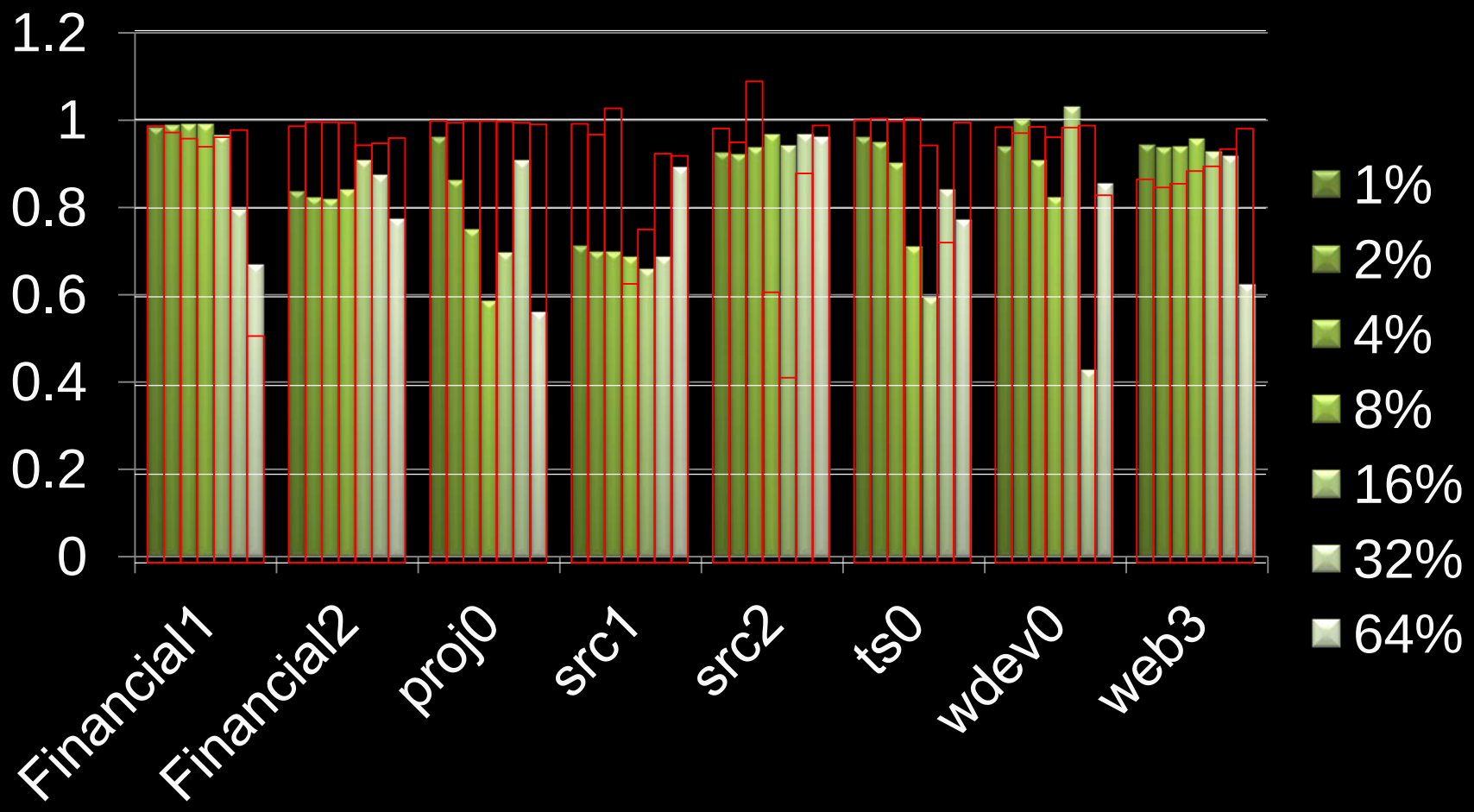
Summary

- Exclusive caching is worth the effort
  - Improves performance
  - Can reduce CAPEX by using cheap SSDs
  - Is beneficial across most RAM/SSD size ratios
- But
  - How do we deal with dirty data?
  - How do we deal with poor lifetime of cheap SSDs?

Dealing With Dirty Data

- Three approaches in designing a r/w cache
  - No special handling of dirty data
  - Partition cache into separate clean/dirty regions
  - Make demand-based algorithms write aware
- Default UARC
  - Clean and dirty managed data together
- P-UARC
  - Two independent, statically-sized UARC partitions
- Cost-Aware UARC (CA-UARC)
  - Adapting ARC algorithm to be cost aware

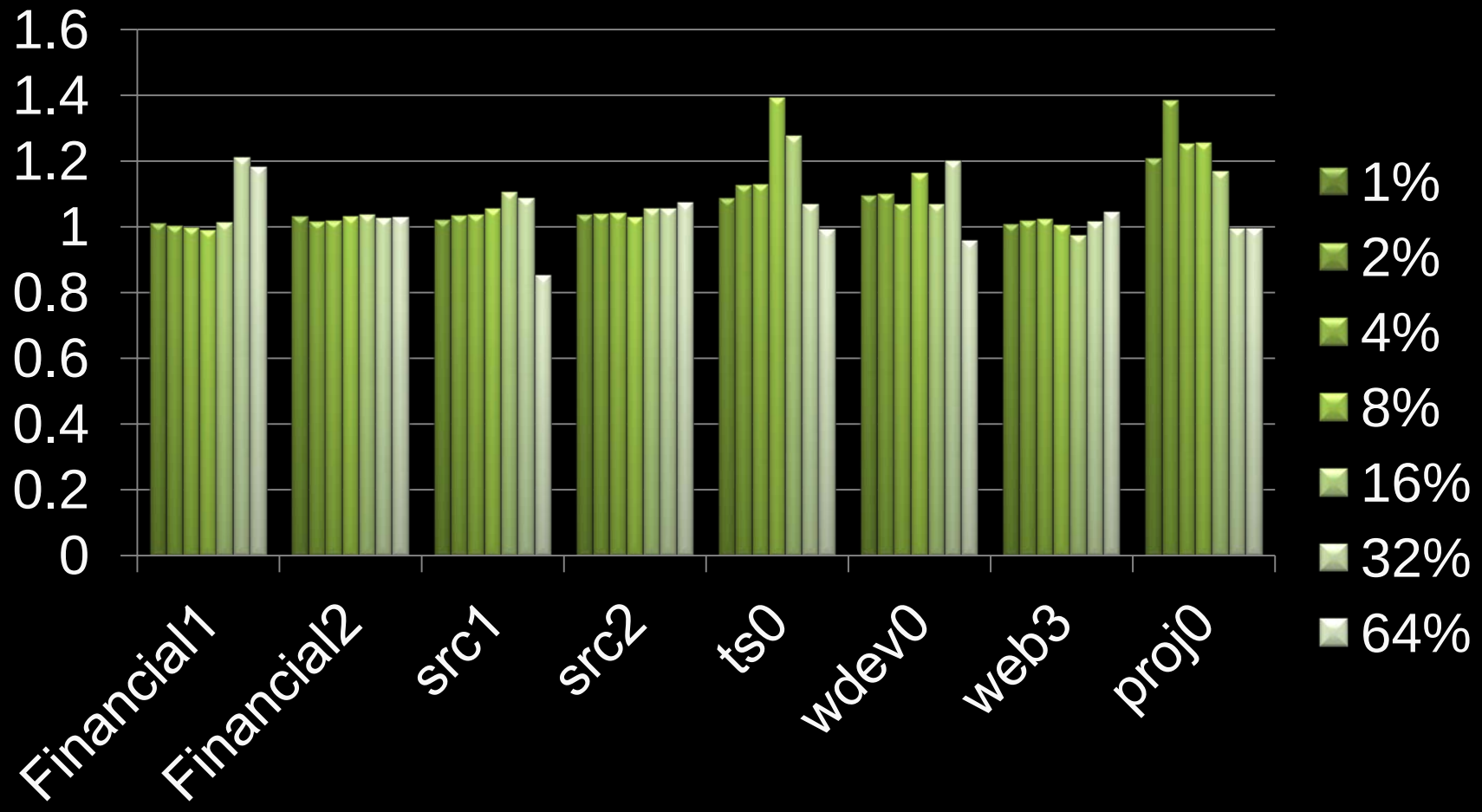
Default-UARC R/W Cache
Normalized Execution Time



Exclusion offers substantial benefits in a read-write cache

P-UARC R/W Cache

Normalized Execution Time



Partitioning the cache into read/write regions offers no improvement

CA-UARC (1)

- Classify blocks based on I/O cost
  - Need an oracle to predict next access

Level	Status	Next Access	I/O Cost
RAM	Clean	R	2 (1 SSD W + 1 HDD/SSD R)
RAM	Clean	W	1 (1 SSD W)
RAM	Dirty	R	2 (1 SSD W + 1 HDD/SSD R)
RAM	Dirty	W	1 (1 SSD W)
SSD	Clean	R	1 (1 HDD R)
SSD	Clean	W	0
SSD	Dirty	R	2 (1 HDD W + 1 HDD R)
SSD	Dirty	W	1 (1 HDD W)

CA-UARC (2)

- Always evict block with least I/O cost
  - ARC picks the target list (T1 or T2)
  - I/O cost determines the target block
- Performance dependent on predictor
  - Exec. time drops (27%) with ideal predictor
  - No improvement with online predictors
    - Poor prediction => evicting “hot” but low-cost data

Summary

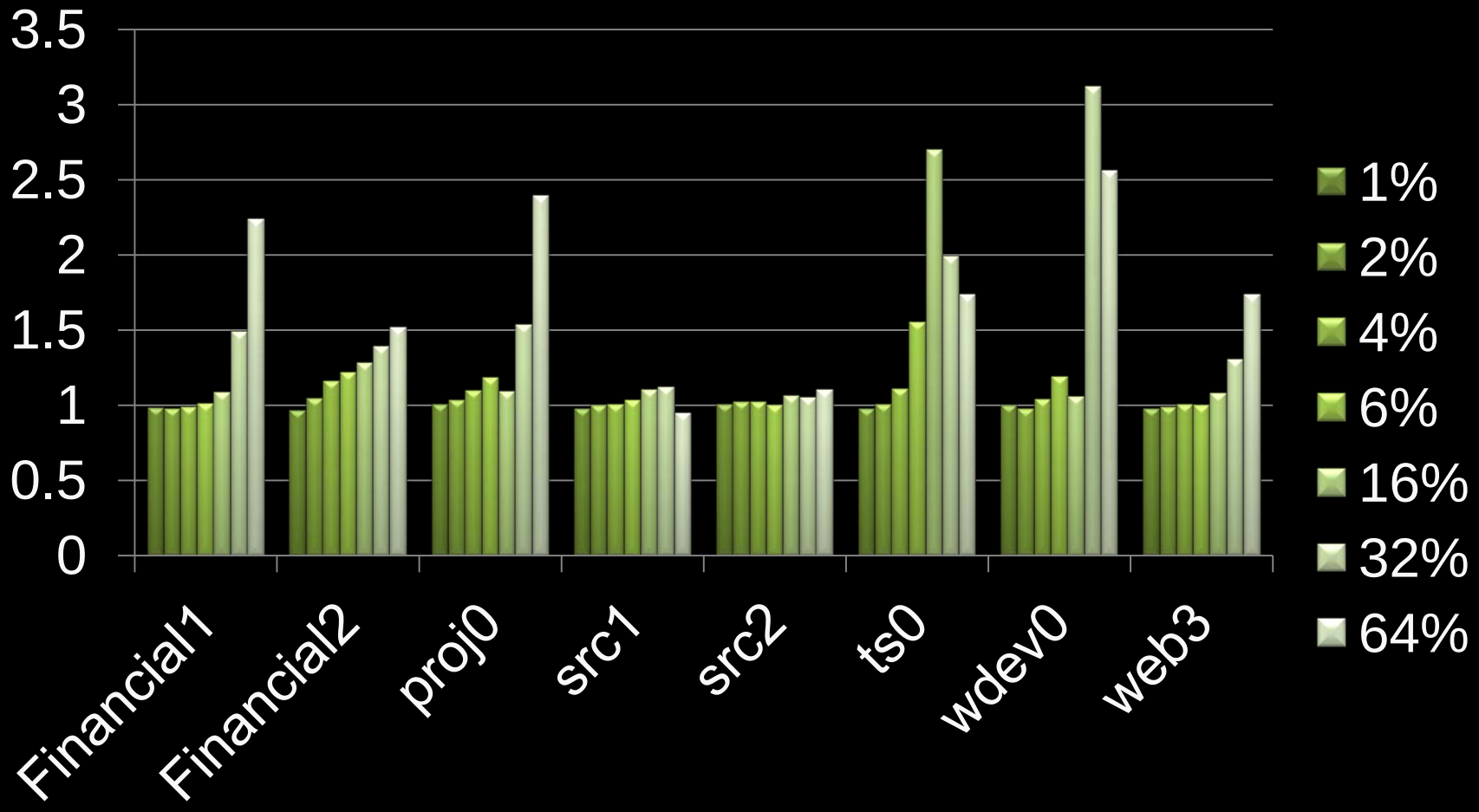
- Exclusively caching dirty data improves performance
- Cost-aware demand-based algorithms win
  - Partitioning fails to improve performance

Dealing With Lifetime

- Sieving accesses to improve lifetime
  - Per-block access count tracking
  - SSD allocation iff access count > sieve threshold
- Simulated both SE-IARC and SE-UARC
  - What is the worst-case performance impact?
  - Does SE-UARC improve lifetime over SE-IARC?
  - How does sieving interact with cost awareness? (details in the paper)

Performance Impact Of Sieving

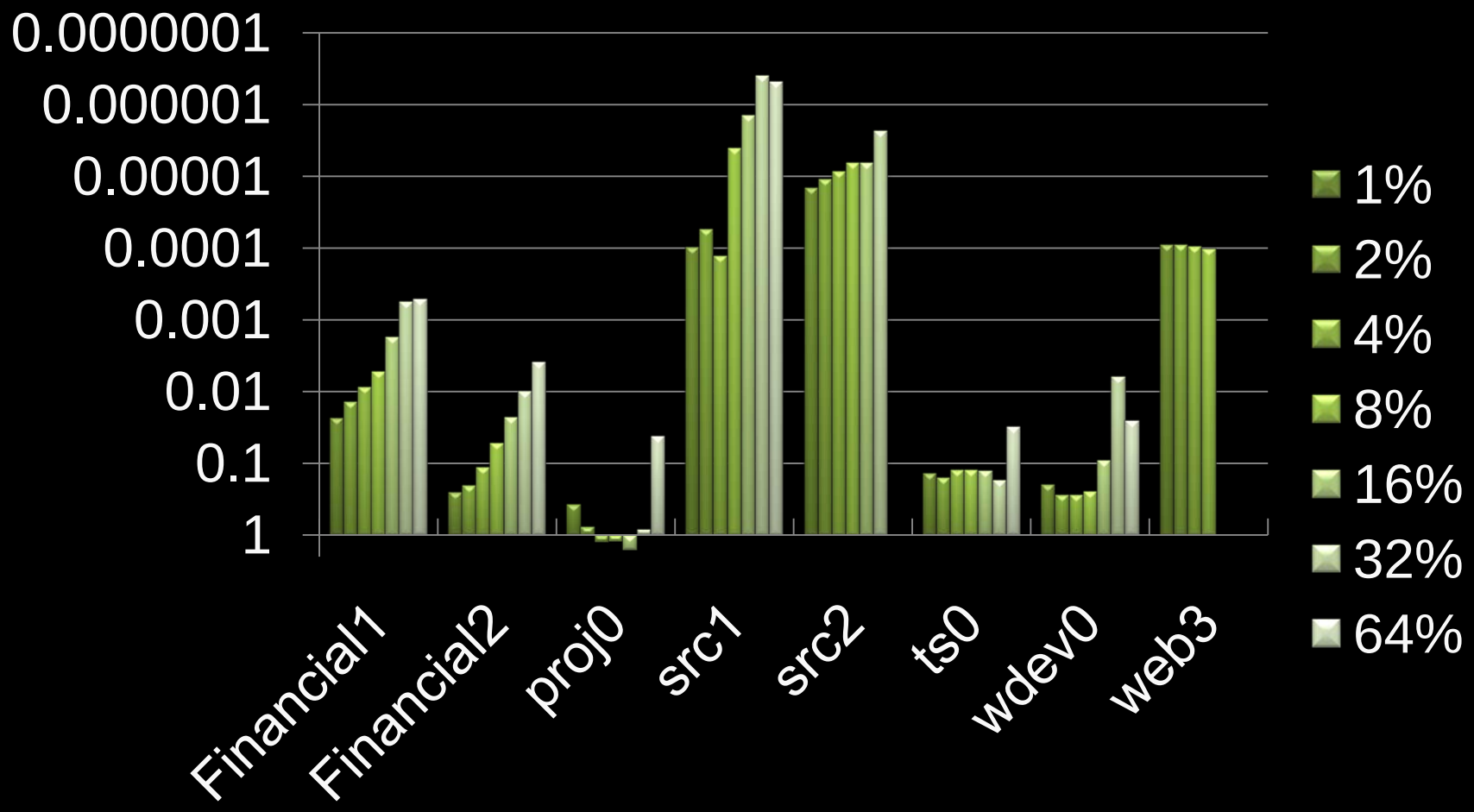
Normalized Exec. Time At Highest Threshold



In several cases, sieving improves lifetime with negligible perf. impact

SE-UARC vs SE-IARC

Normalized SE-UARC Allocation Count



Exclusivity improves SSD lifetime by several orders of magnitude

Summary

- Sieving SSD allocations is useful
  - Even a highly selective admission policy has only little performance impact
- Exclusive caching improves SSD lifetime
  - Hot data cached in RAM not allocated in SSD

Conclusion

- Persistent, terabyte-sized, multilevel, direct-attached caches
  - Exclusion MUST be a first-class design factor
- Exclusion in the storage stack
  - Can we extend FS-block level interface?
  - Do we implement new file systems?

EOF