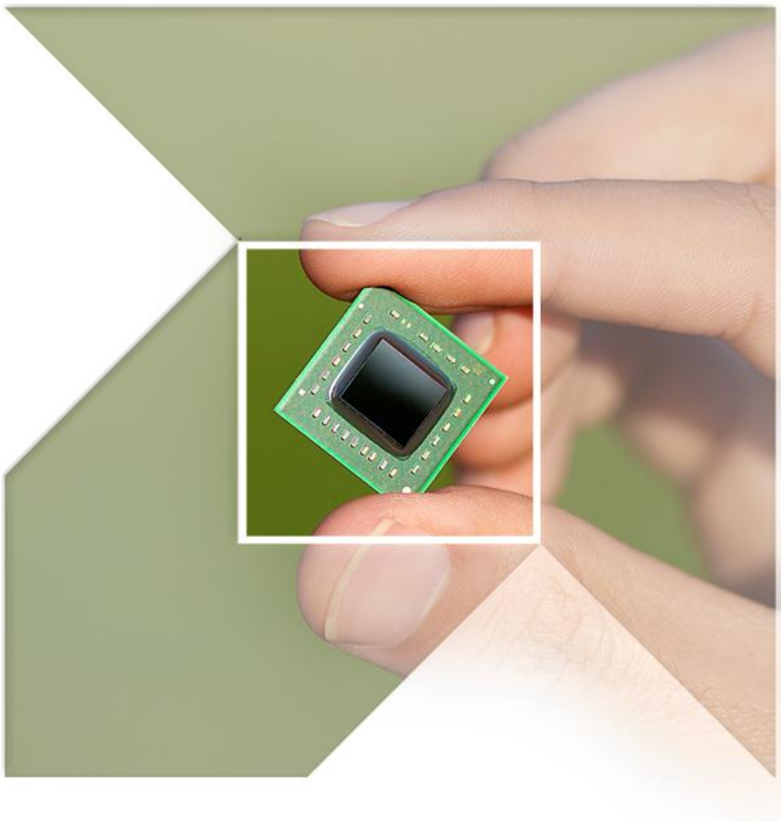


EFFICIENTLY ENABLING CONVENTIONAL BLOCK SIZES FOR VERY LARGE DIE-STACKED DRAM CACHES

MICRO 2011 @ Porte Alegre, Brazil

Gabriel H. Loh [1] and Mark D. Hill [2][1]

December 2011

[1] AMD Research
[2] University of Wisconsin-Madison

Hill's work largely performed while on sabbatical at [1].

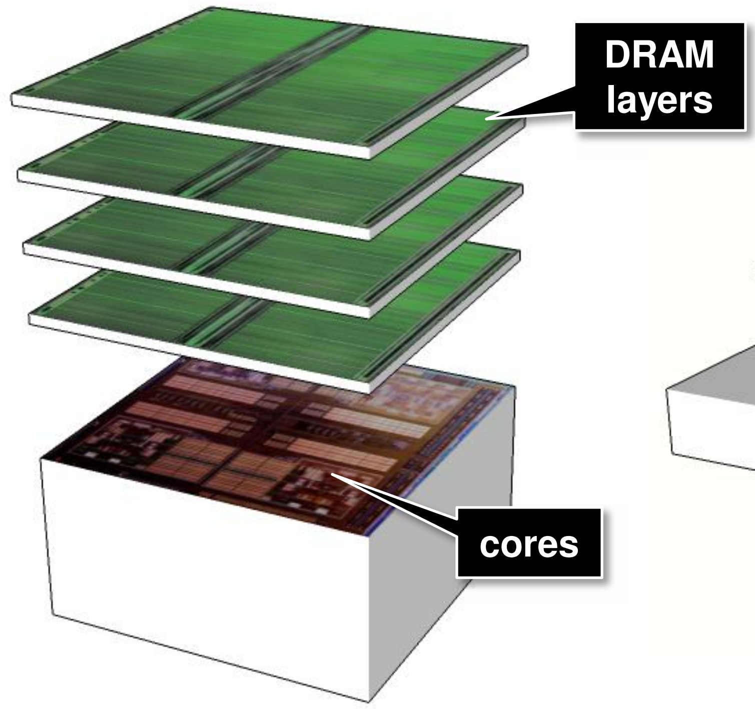
EXECUTIVE SUMMARY

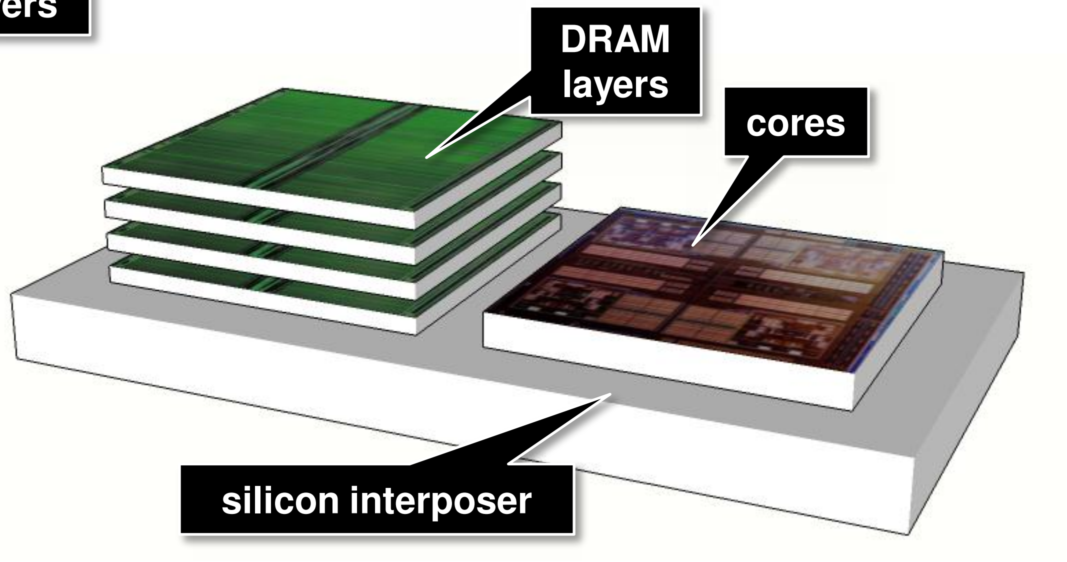
- Good use of stacked DRAM is cache, but:
  - Tags in stacked DRAM believed too slow
  - On-chip tags too large (e.g., 96 MB for 1 GB stacked DRAM cache)
- Solution put tags in stacked DRAM, but:
  - Faster Hits: Schedule together tag & data stacked DRAM accesses
  - Faster Miss: On-chip MissMap bypasses stacked DRAM on misses
- Result (e.g., 1 GB stacked DRAM cache w/ 2 MB on-chip MissMap)
  - 29-67% faster than naïve tag+data in stacked DRAM
  - Within 88-97% of stacked DRAM cache w/ impractical on-chip tags

OUTLINE

- Motivation
- Fast Hits via Compound Access Scheduling
- Fast Misses via MissMap
- Experimental Results
- Related Work and Summary

CHIP STACKING IS HERE





"Vertical"

"Horizontal"

256 MB

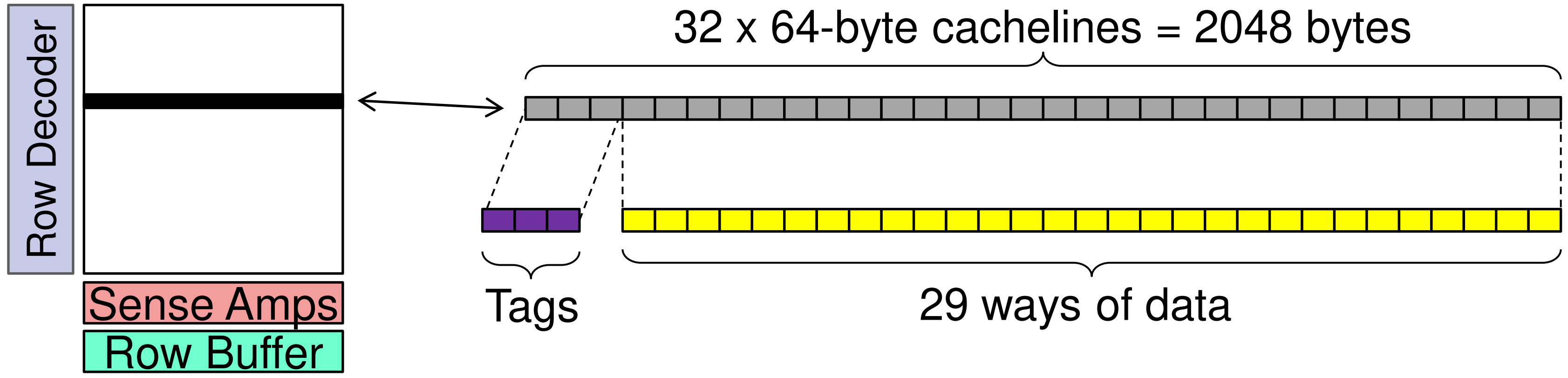
Samsung @ ISSCC'11: "A 1.2V 12.8Gb/s **2Gb** Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking"

HOW TO USE STACKED MEMORY?

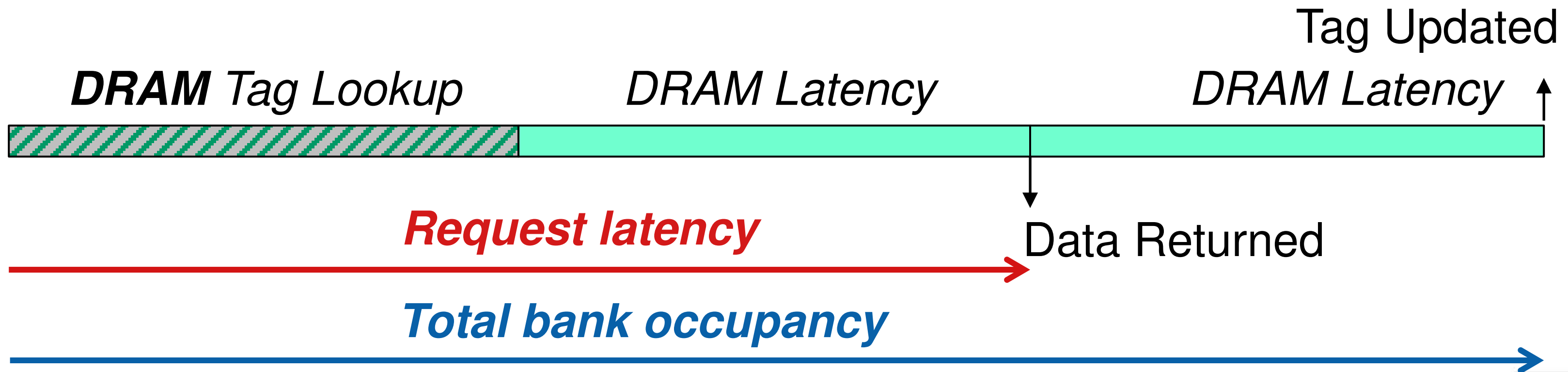
- Complete Main Memory
  - Few GB too small for all but some embedded systems
- OS-Managed NUMA Memory
  - Page-size fragmentation an issue
  - Requires OS-HW cooperation (across companies)
- Cache w/ Conventional Block (Line) Size (e.g., 64B)
  - But on-chip tags for 1 GB cache is impractical 96 MB! (TAKE 1)
- Sector/subblock Cache
  - Tag w/ 2KB block (sector) + state bits w/ each 64B subblock
  - Tags+state fits on-chip, but fragmentation issues (see paper)

TAG+DATA IN DRAM (CONVENTIONAL BLOCKS – TAKE 2)

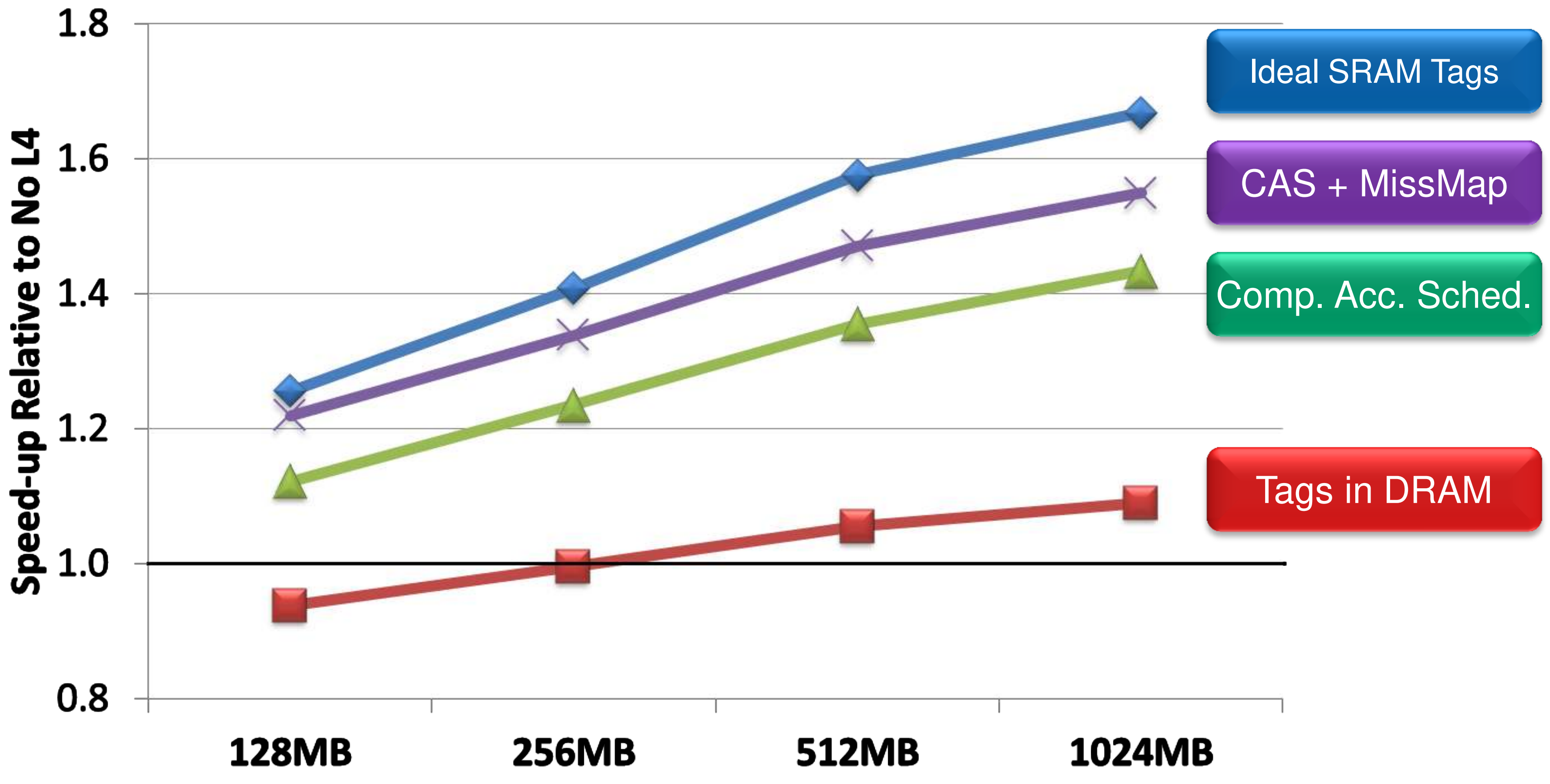
- Use 2K-Stacked-DRAM pages but replace 32 64B blocks with
  - 29 tags (48b) + 29 blocks



  - But previously dismissed as too slow



IMPRACTICAL IDEAL & OUR RESULT FORECAST



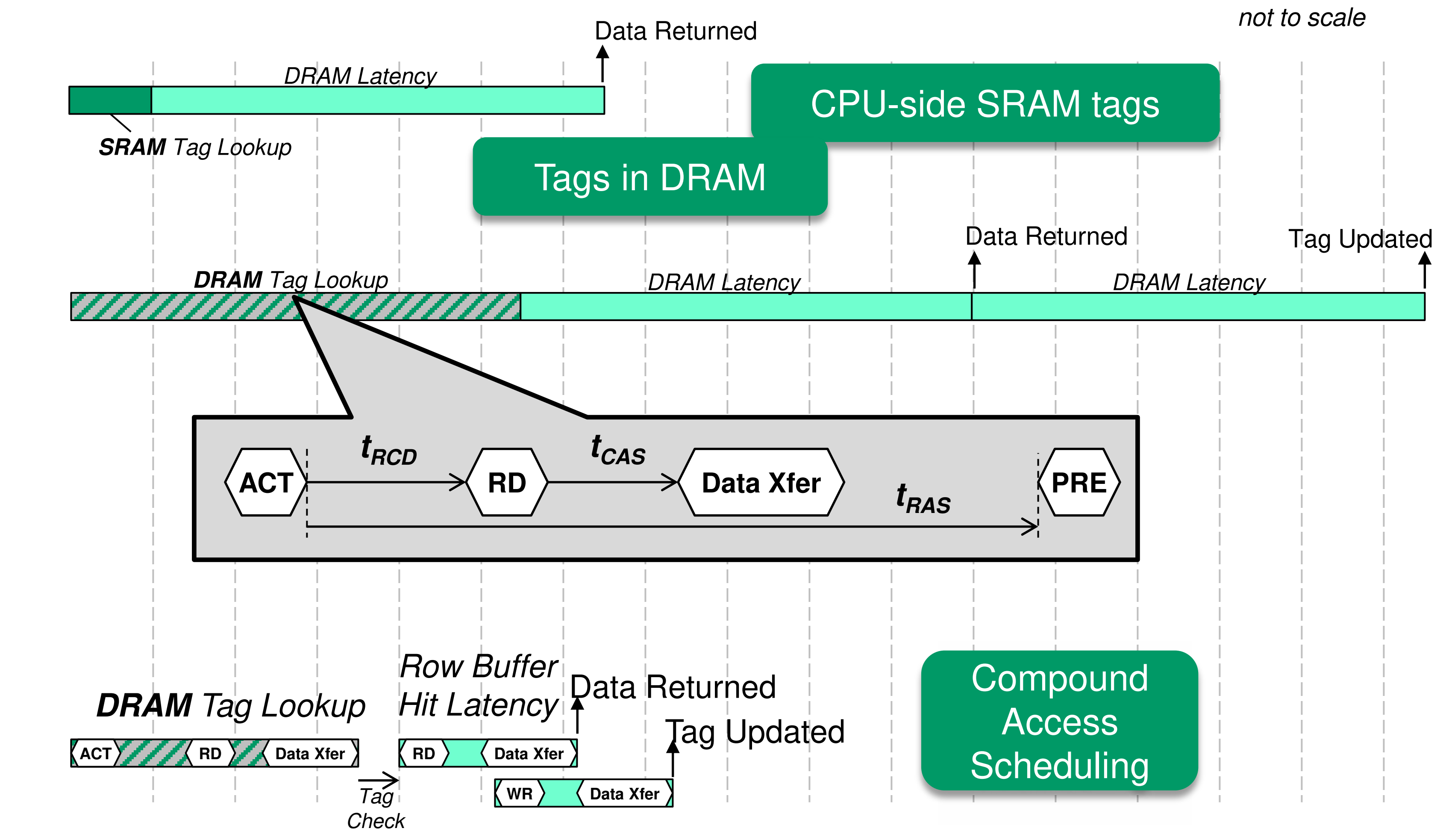
Compound Access Scheduling + MissMap → Approximate impractical on-chip SRAM tags

Methods Later; Avg of Web-Index, SPECjbb05, TPC-C, & SPECweb05

OUTLINE

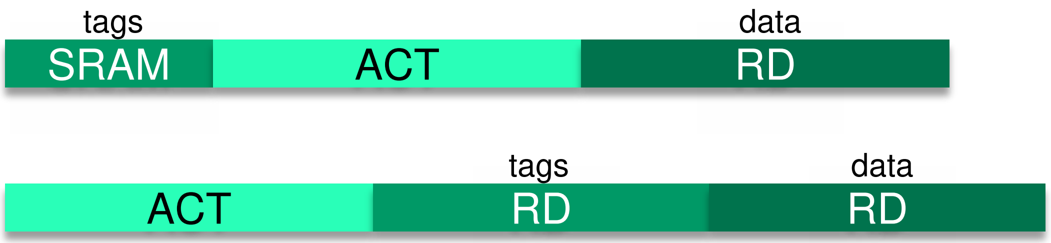
- Motivation
- **Fast Hits via Compound Access Scheduling**
- Fast Misses via MissMap
- Experimental Results
- Related Work and Summary

FASTER HITS (CONVENTIONAL BLOCKS – TAKE 3)



COMPOUND ACCESS SCHEDULING

- Reserve the bank for data access; guarantee row buffer hit
  - Approximately trading an SRAM lookup for a row-buffer hit:



- On a miss, unnecessarily holds bank open for the tag-check latency
  - Prevents tag lookup on another row in same bank
  - Effective penalty is minimal since t_{RAS} must elapse before closing this row, so bank will be unavailable anyway

OUTLINE

- Motivation
- Fast Hits via Compound Access Scheduling
- **Fast Misses via MissMap**
- Experimental Results
- Related Work and Summary

FASTER MISSES (CONVENTIONAL BLOCKS – TAKE 4)

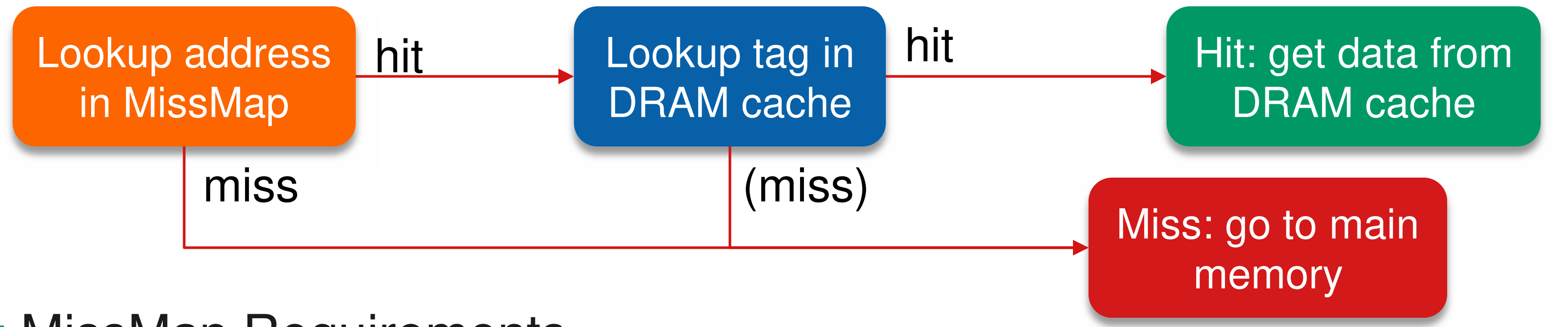
- Want to avoid delay & power of stacked DRAM access on miss

- Impractical on-chip tags answer
  - Q1 “Present:” Is block in stacked DRAM cache?
  - Q2 “Where:” Where in stacked DRAM cache (set/way)?

- New on-chip MissMap
  - Approximate impractical tags for practical cost
  - Answer Q1 “Present”
  - But NOT Q2 “Where”

MISSMAP

- On-chip structures to answer Q1: Is block in stacked DRAM cache?

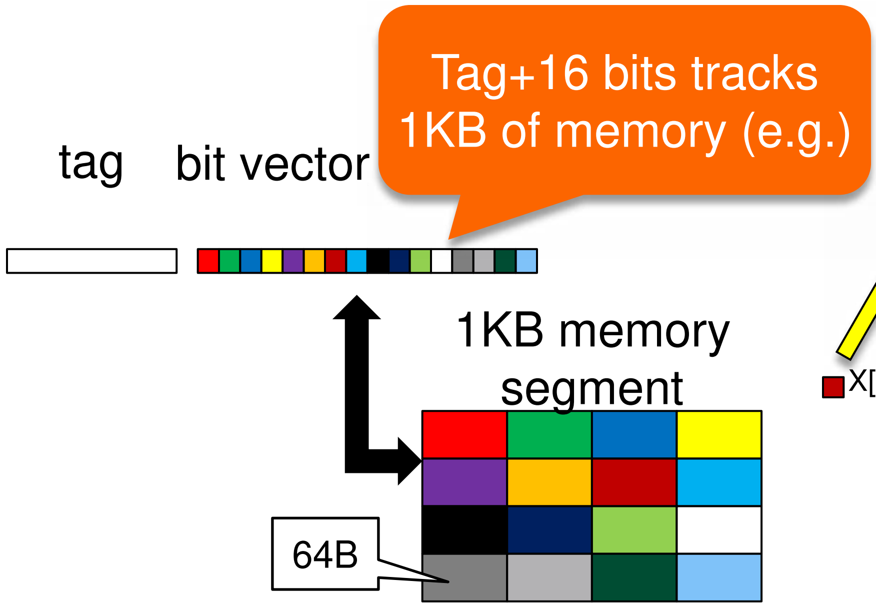


- MissMap Requirements
  – Add block in miss; remove block on victimization
  – No false negatives: If says, “not present” ➔ must be not present
  – False positives allowed: If says, “present” ➔ may (rarely) miss
- Sounds like a Bloom Filter?

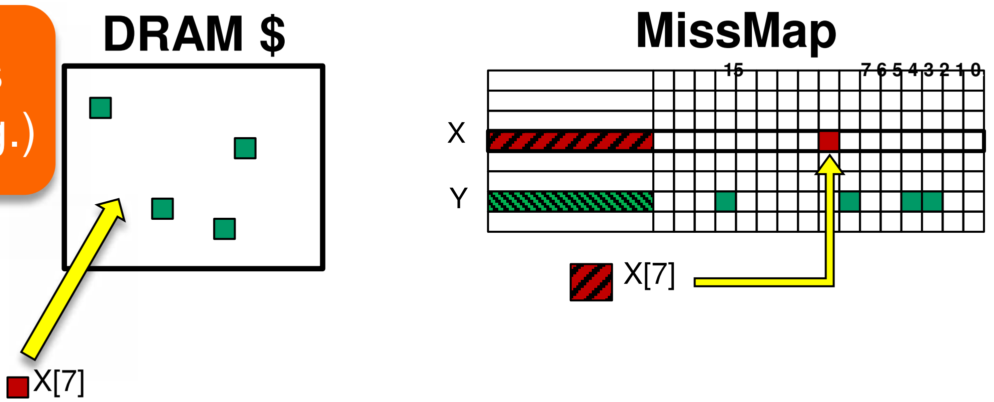
- But our implementation is precise – no false negatives or positives
  – Extreme subblocking with over-provisioning

MISSMAP IMPLEMENTATION

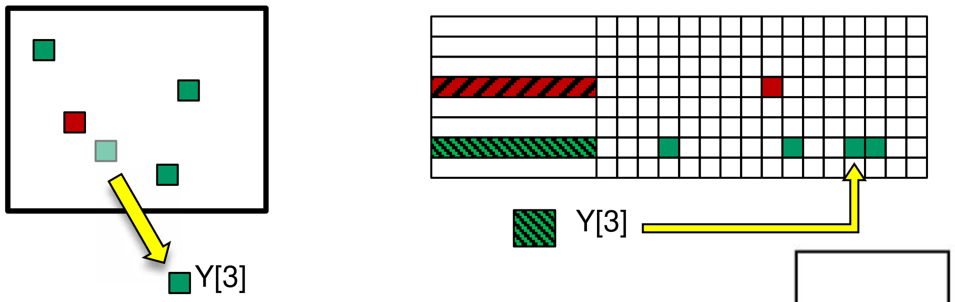
MissMap entry



Installing a line in the DRAM $

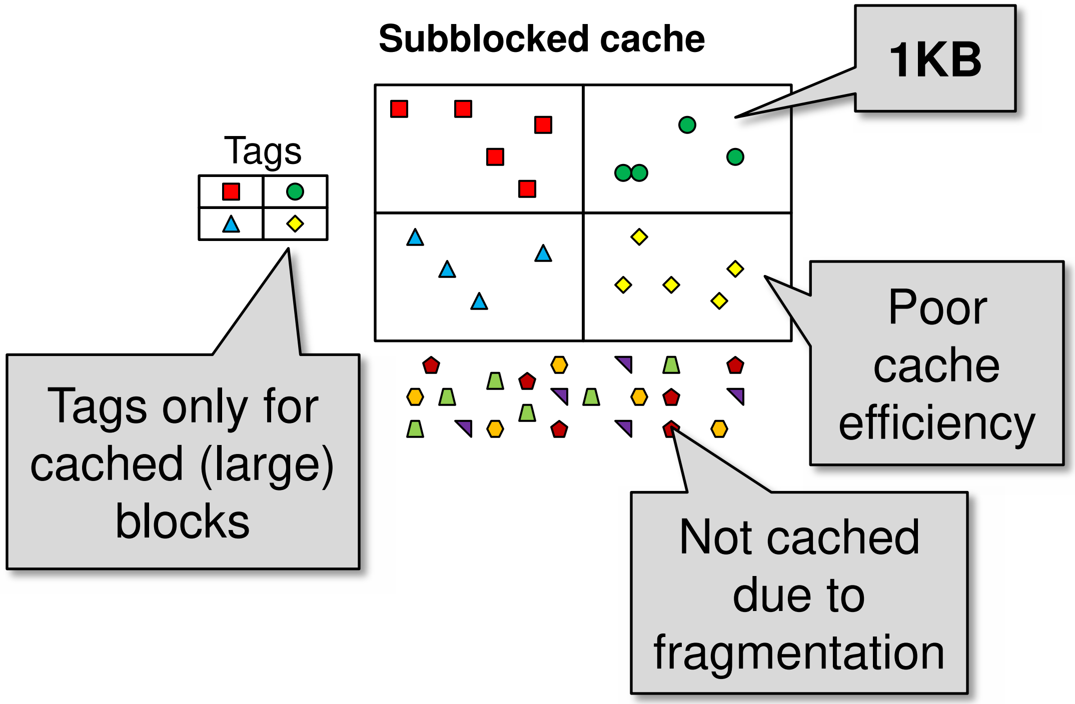


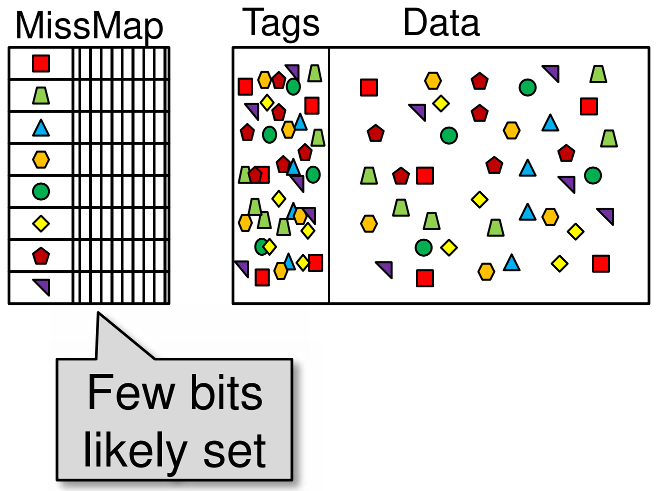
Evicting a line from the DRAM $



- Key 1: Extreme Subblocking

MISSMAP IMPLEMENTATION


Subblocked cache
1KB
Tags
Tags only for cached (large) blocks
Poor cache efficiency
Not cached due to fragmentation



- Key 2: Over-provisioning
- Key 3: Answer Q1 “Present” NOT Q2 “Where”
  - 36b tag + 64b vector = 100b
  - NOT 36b tag + 5*64b vector = 356b (3.6x)

Example:
2MB MissMap
4KB pages
Each entry is ~12.5 bytes
(36b tag, 64b vector)
167,000 entries total
Best case, tracks ~640MB

OUTLINE

- Motivation
- Fast Hits via Compound Access Scheduling
- Fast Misses via MissMap
- **Experimental Results**
- Related Work and Summary

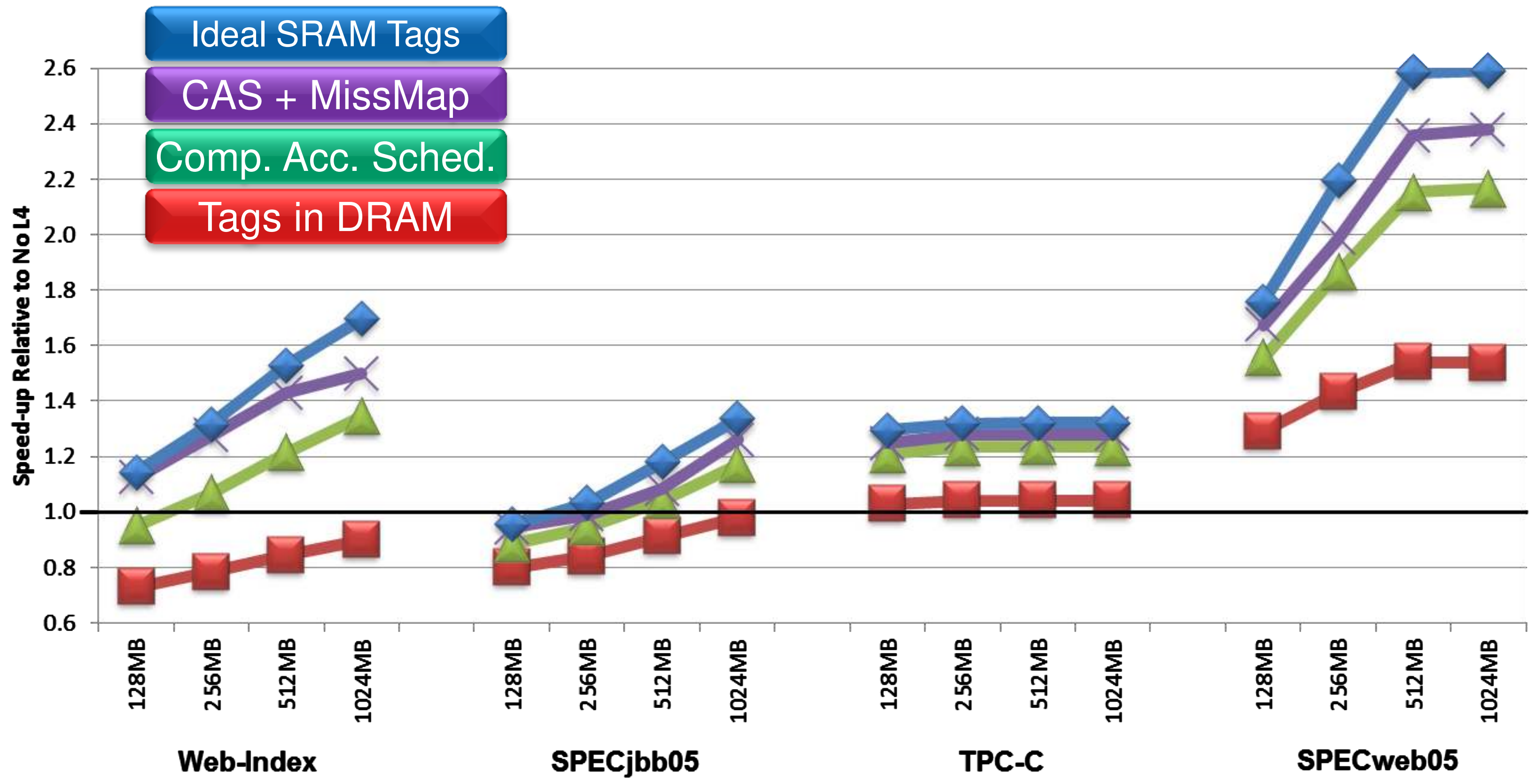
METHODOLOGY (SEE PAPER FOR DETAILS)

- Workloads (footprint)
  - Web-Index (2.98 GB) // SPECjbb05 (1.20 GB)
  - TPC-C (1.03 GB) // SPECweb05 (1.02 GB)

- Base Target System
  - 8 3.2 GHz cores with 1 IPC peak w/ 2-cycle 2-way 32KB I\$ + D\$
  - 10-cyc 8-way 2MB L2 for 2 cores + 24-cyc 16-way 8MB shared L3
  - Off-chip DRAM: DDR3-1600, 2 channels
- Enhanced Target System
  - 12-way 6MB shared L3 + 2MB MissMap
  - Stacked DRAM: 4 channels, 2x freq (~½ latency), 2x bus width

- gem5 simulation infrastructure (= Wisconsin GEMS + Michigan M5)

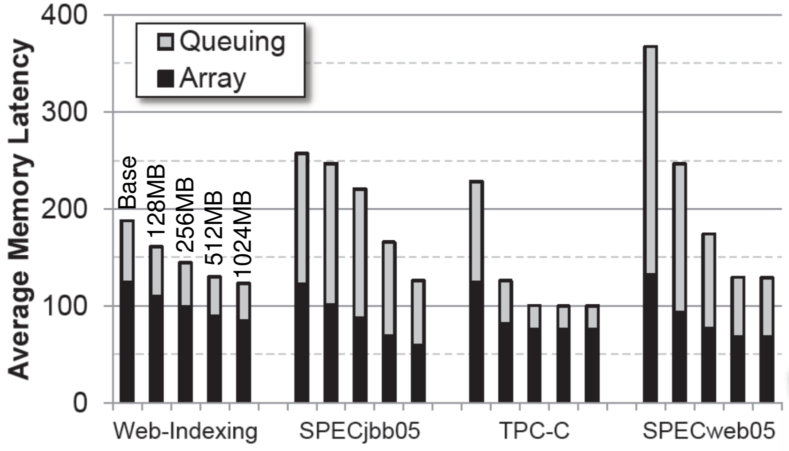
KEY RESULT: COMPOUND SCHEDULING + MISSMAP WORK



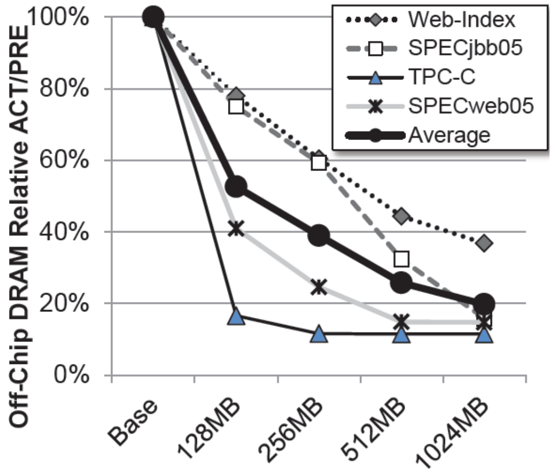
Compound Access Scheduling + MissMap → Approximate impractical on-chip SRAM tags

2ND KEY RESULT: OFF-CHIP CONTENTION REDUCED

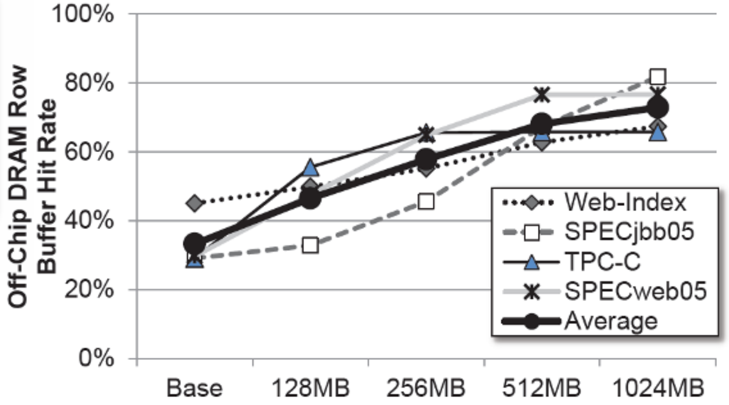
- For requests that miss, main memory is more responsive



Fewer requests → lower queuing delay



Fewer requests → More row-buffer hits → lower DRAM latency



OTHER RESULTS IN PAPER

- Impact on all off-chip DRAM traffic (activate, read, write, precharge)
- Dynamic active memory footprint of the DRAM cache
- Additional traffic due to MissMap evictions
- Cacheline vs. MissMap lifetimes
- Sensitivity to how L3 is divided between data and the MissMap
- Sensitivity to MissMap segment size
- Performance against sub-blocked caches

OUTLINE

RELATED WORK

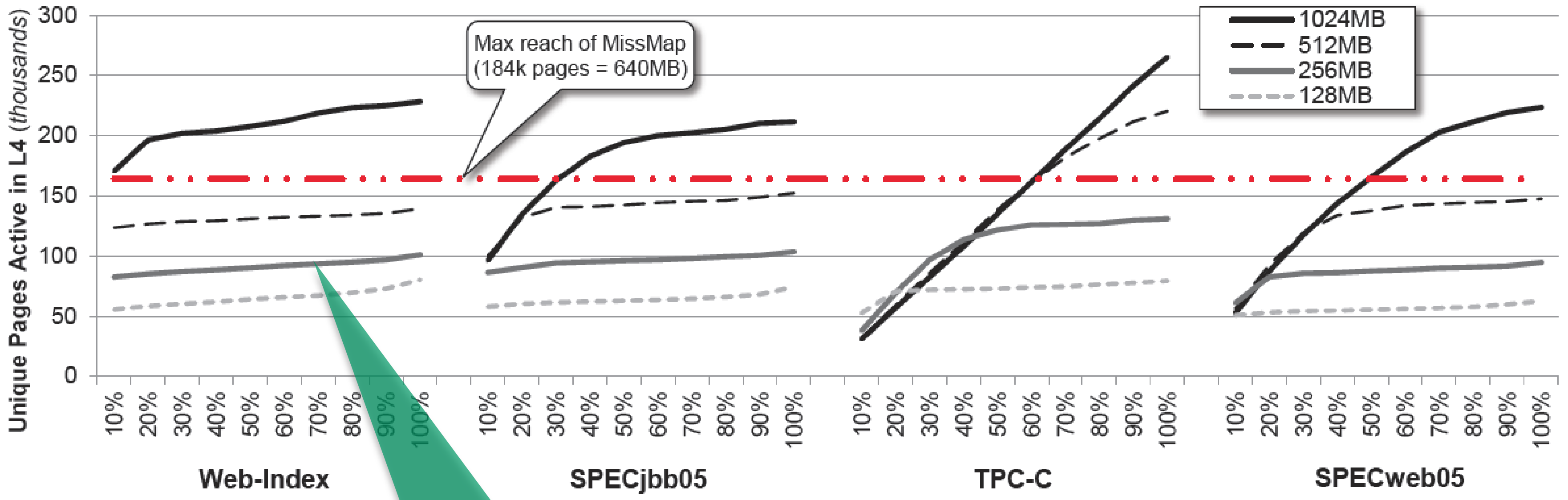
- Stacked DRAM as main memory
  - Mostly assumes all of main memory can be stacked *[Kgil+ ASPLOS'06, Liu+ IEEE D&T'05, Loh ISCA'08, Woo+ HPCA'10]*
- As a large cache
  - Mostly assumes tag-in-DRAM latency too costly *[Dong+ SC'10, Ghosh+ MICRO'07, Jiang+ HPCA'10, Loh MICRO'09, Zhao+ ICCD'07]*
- Other stacked approaches (NVRAM, hybrid technologies, etc.)
  - *[Madan+ HPCA'09, Zhang/Li PACT'09]*
- MissMap related
  - Subblocking *[Liptay IBMSysJ'68, Hill/Smith ISCA'84, Seznec ISCA'94, Rothman/Smith ICS'99]*
  - "Density Vector" for prefetch suppression *[Lin+ ICCD'01]*
  - Coherence optimization *[Moshovos+ HPCA'01, Cantin+ ISCA'05]*

EXECUTIVE SUMMARY

- Good use of stacked DRAM is cache, but:
  - Tags in stacked DRAM believed too slow
  - On-chip tags too large (e.g., 96 MB for 1 GB stacked DRAM cache)
- Solution put tags in stacked DRAM, but:
  - Faster Hits: Schedule together tag & data stacked DRAM accesses
  - Faster Miss: On-chip MissMap bypasses stacked DRAM on misses
- Result (e.g., 1 GB stacked DRAM cache w/ 2 MB on-chip MissMap)
  - 29-67% faster than naïve tag+data in stacked DRAM
  - Within 88-97% of stacked DRAM cache w/ impractical on-chip tags

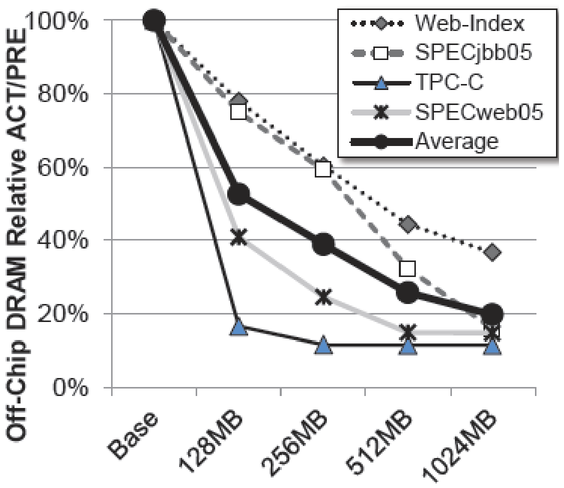
BACKUP SLIDES

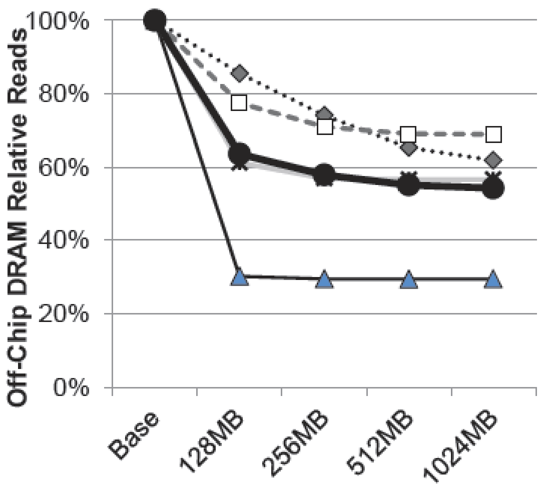
UNIQUE PAGES IN L4 VS. MISSMAP REACH

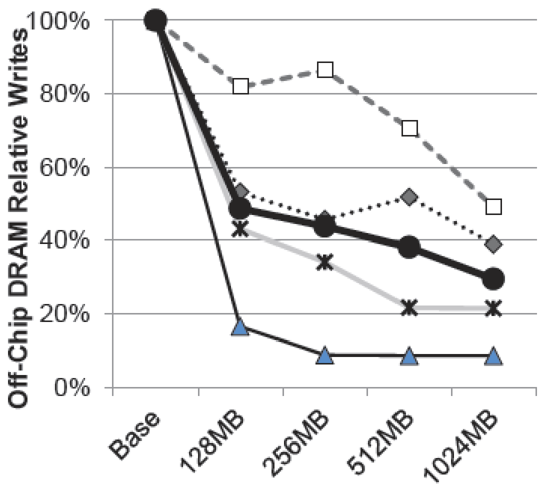


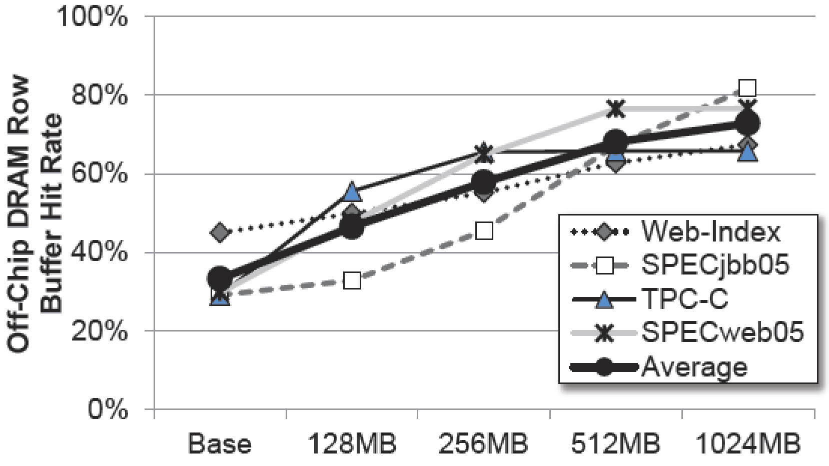
Ex. 70% of the time a 256MB cache held ~90,000 or fewer unique pages

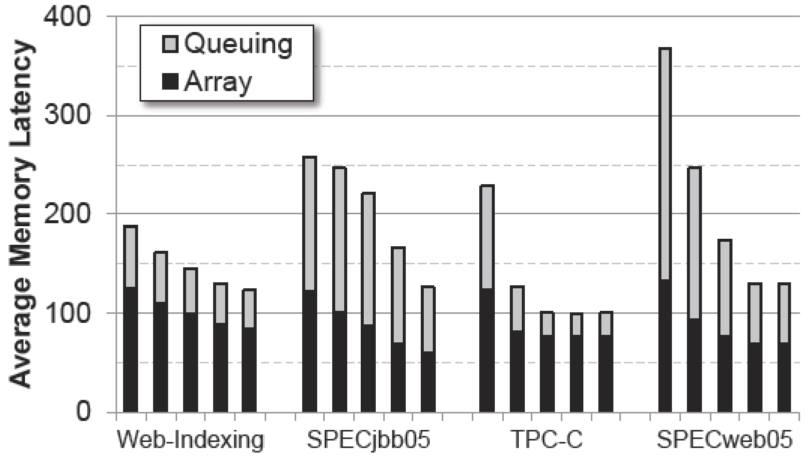
IMPACT ON OFF-CHIP DRAM ACTIVITY







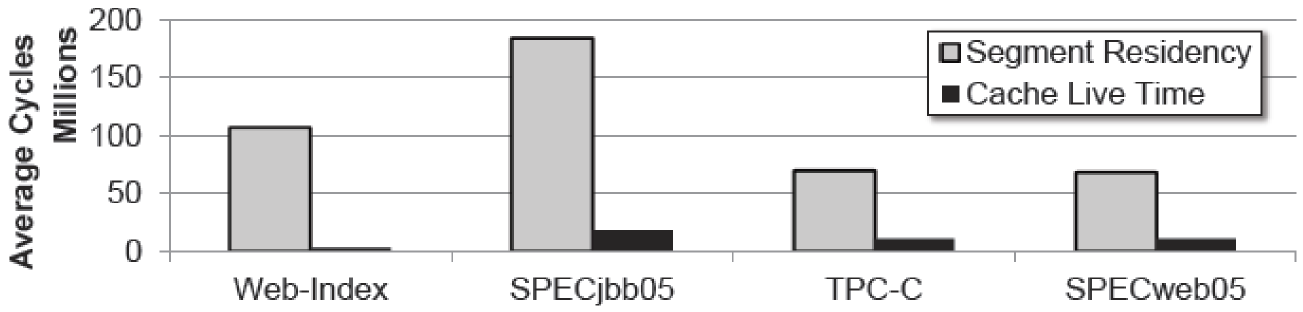




MISSMAP EVICTION TRAFFIC

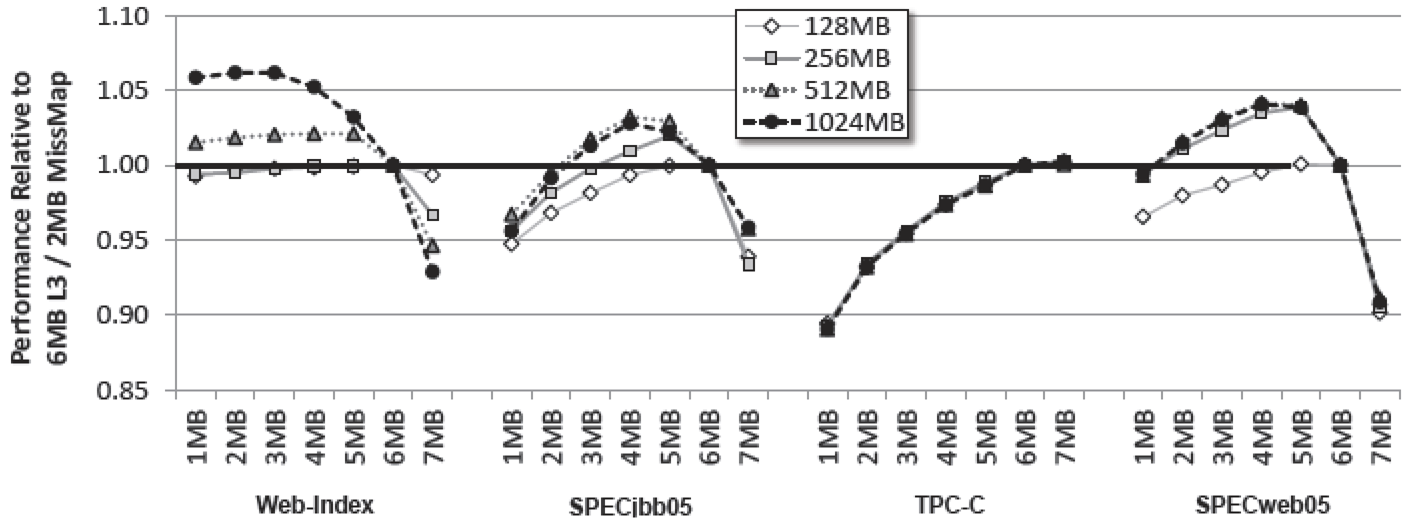
Workload Name	% Clean MissMap Evictions	Dirty Lines per Dirty Segment	
		128MB	1GB
Web-Index	99.7%	0.02	26.8
SPECjbb05	55.7%	1.06	29.3
TPC-C	74.7%	3.80	4.93
SPECweb05	72.2%	1.61	16.2

- Many MissMap evictions correspond to clean pages (e.g., no writeback traffic from the L4)



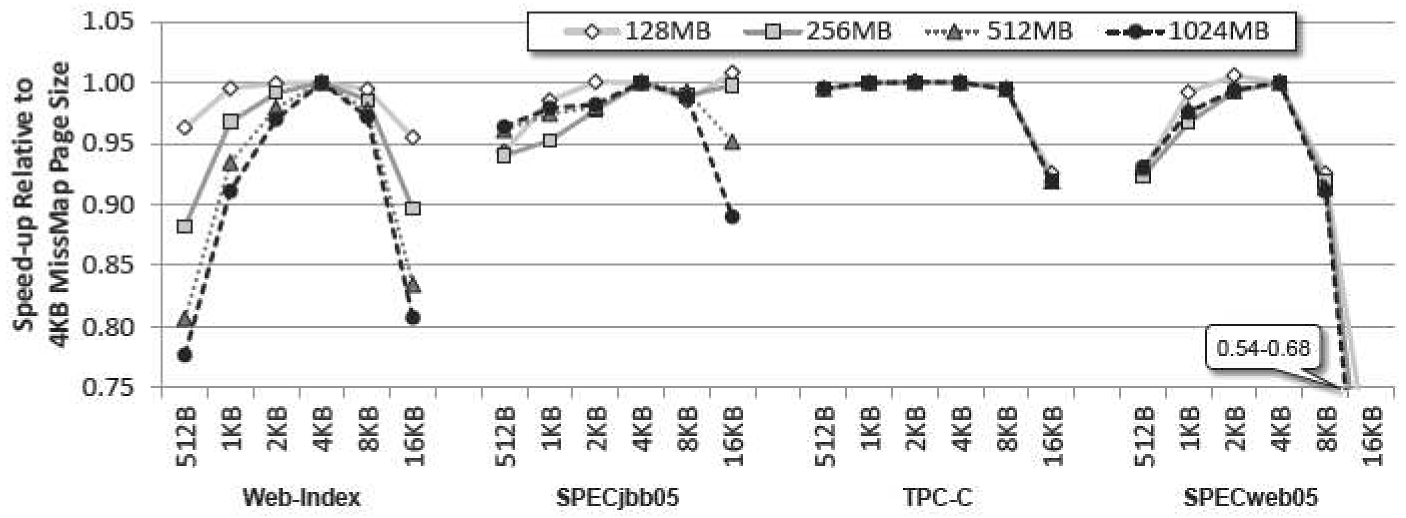
- By the time a MissMap entry is evicted, most of its cachelines have are long past dead/evicted.

SENSITIVITY TO MISSMAP VS. DATA ALLOCATION OF L3



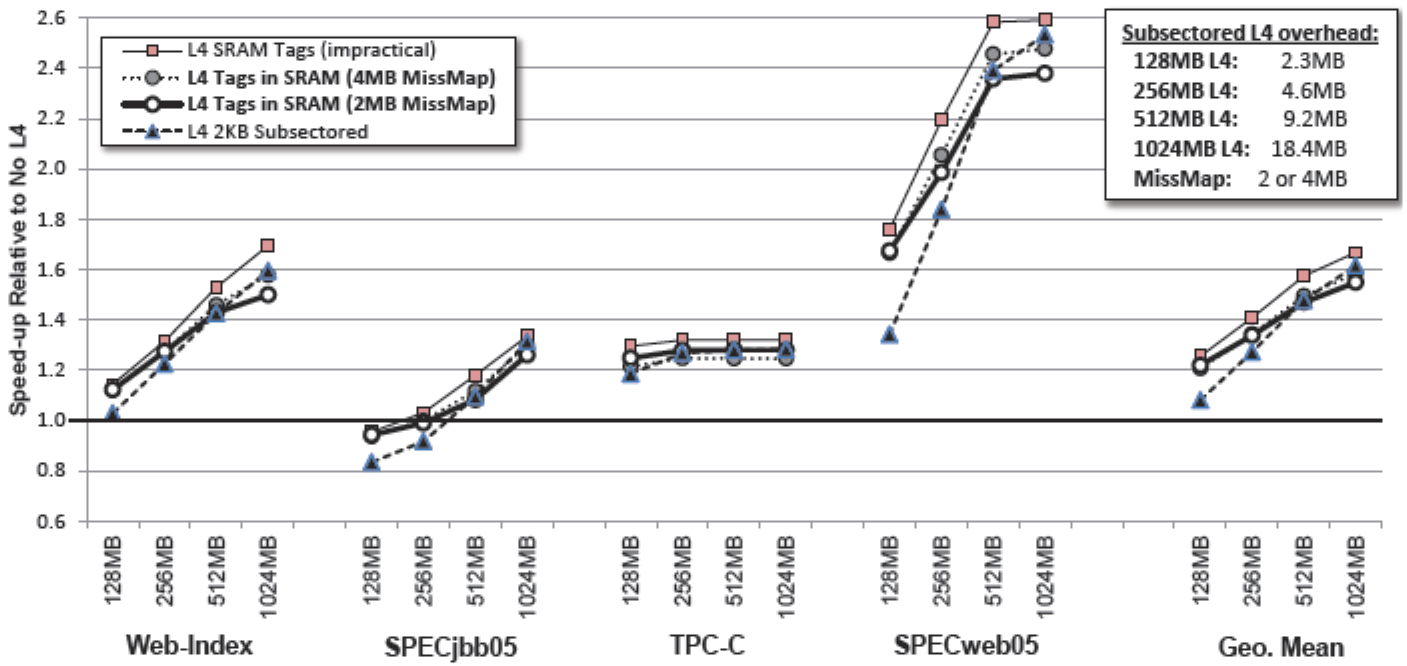
- 2MB MissMap + 6MB Data provides good performance
- 3MB MissMap + 5MB Data slightly better, but can hurt server workloads that are more sensitive to L3 capacity.

SENSITIVITY TO MISSMAP SEGMENT SIZE



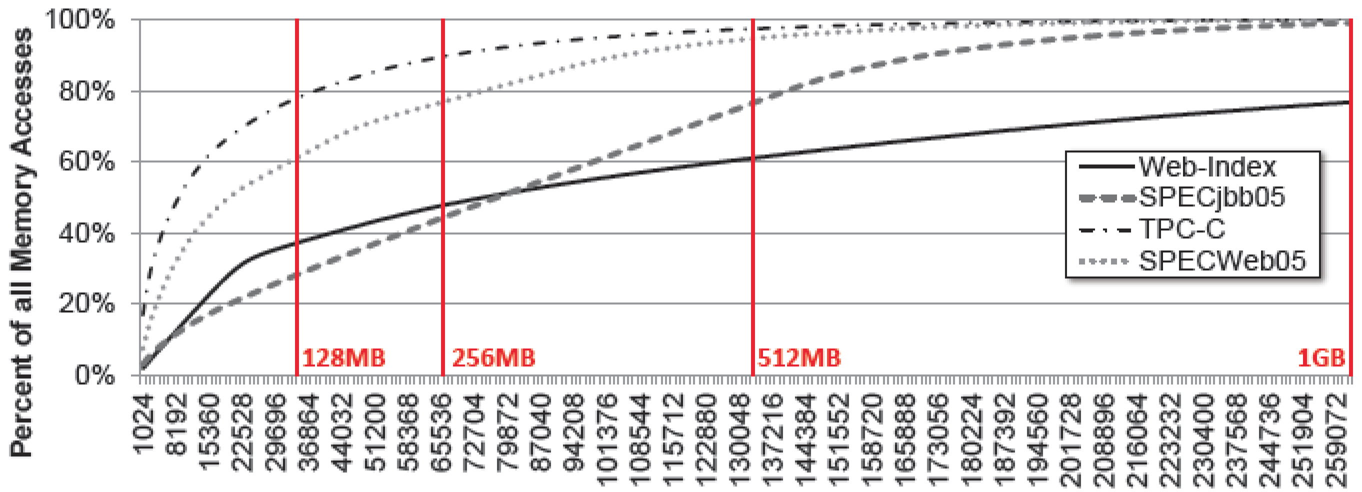
- 4KB segment size works the best
- Our simulations make use of physical addresses, so consecutive virtual pages can be mapped to arbitrary physical pages

COMPARISON TO SUB-BLOCKED CACHE



- Beyond 128MB, overhead is greater than MissMap
- At largest sizes (512MB, 1GB), sub-blocked cache delivers similar performance to our approach, but at substantially higher cost

BENCHMARK FOOTPRINTS



- TPC-C: ~80% of accesses served by hottest 128MB worth of pages
- SPECWeb05: ~80% accesses served by 256MB
- SPECjbb05: ~80% accesses served by 512MB
- Web-Index: huge active footprint