Prototyping a Hybrid Main Memory Using a Virtual Machine Monitor

Authors

Dong Ye[§], Aravind Pavuluri[*], Carl Waldspurger[*]

Brian Tsang[¥], Bohuslav Rychlik[¥], Steven Woo[¥]

[§]Northeastern University, [*]VMware, Inc., [¥]Rambus, Inc.,

Presented at ICCD by

Brian Tsang

10-13-08

Contents

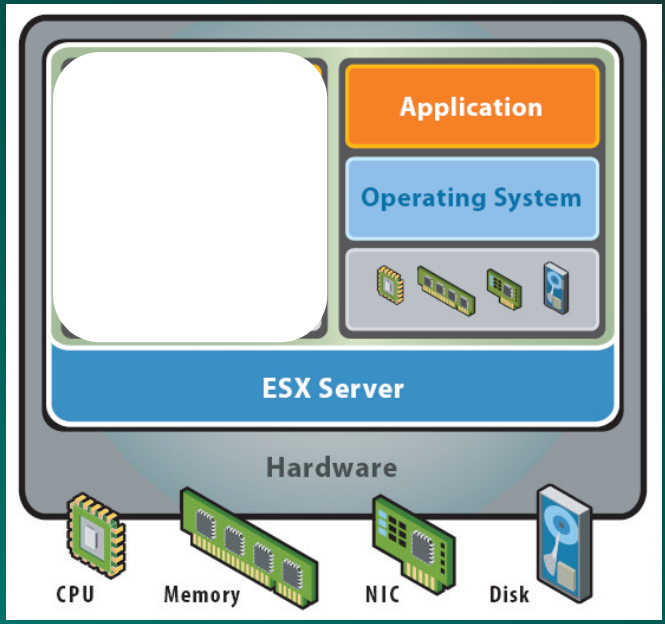
- Introduction
- VMM-based Model
- Experimental Method
- Case Study Results
- Limitations
- Conclusions

Introduction

- Virtualization typically used for:
  1. Consolidate servers
  2. Enhance security
  3. Centralize Desktop

- We use virtualization to do *performance modeling*…

- Why?
  1. Fast evaluation speed
  2. Freedom in target workload
  3. Complete execution



Case Study: "Analyze Hybrid Main Memory System"

- **Memory technologies such as flash may present opportunity to plug the gap between DRAM and disk in the memory hierarchy**

	Latency (in CPU clocks)	Bandwidth	Cost	Access Size
Registers	1	Very high	$$$$$$	4-8B
L1 Cache	1-2	Very high	$$$$$	16-32B
L2 Cache	4-12	High	$$$$	64-128B
DDR3 1600	120-1000	Medium	$$$	4-32B
Example: Flash Memory	~1,000,000	?	$$	?
Disk	>15,000,000	Low	$	512B-4KB

- **We generalize to any multi-tier main memory system**

NEED MULTI-TIER MEMORY ARCH DIAGRAM HERE
(can use the one from paper with clearly marked M1 and M2)

VMM-based Model

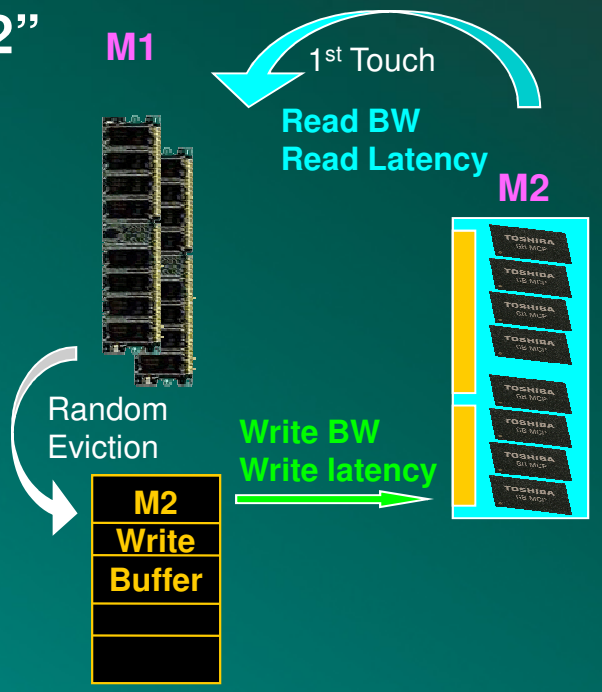
VMware ESX VMM is well-suited to model a hybrid main memory:

→ Already contains an extra level of memory indirection

→ Has a general purpose memory protection mechanism called ***tracing***

We modify the VMM as follows:

- **Mark each memory pages as either "in M2" or "in M1"**
- **Install custom trace handler which fires upon access of any page "in M2"**
- **Insert a delay based on:**
  - Read Latency & BW
  - Write Latency & BW
  - Write Buffer Size
  - Inclusive/Exclusive caching
- **Mark page as now "in M1"**
- **Mark random victim from M1 as "in M2"**



Experimental Method

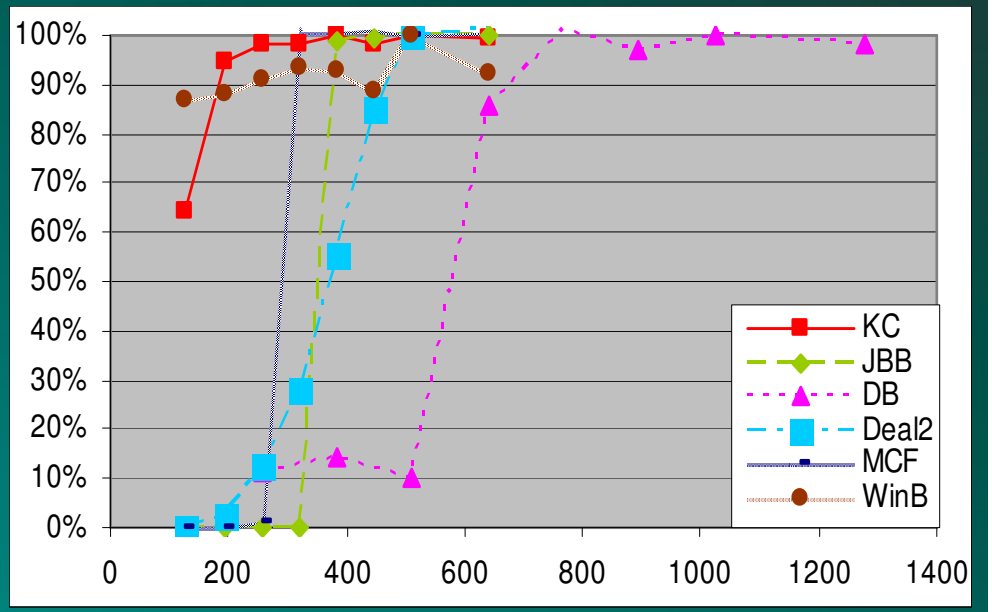
M1+M2 memory size = each benchmark's working set size
Baseline performance: M1 = working set size
M2 = 0

Benchmarks

Enterprise Computing
- 64b Linux - Oracle DB (640 MB)
- 32b Linux - SPEC JBB (384 MB)

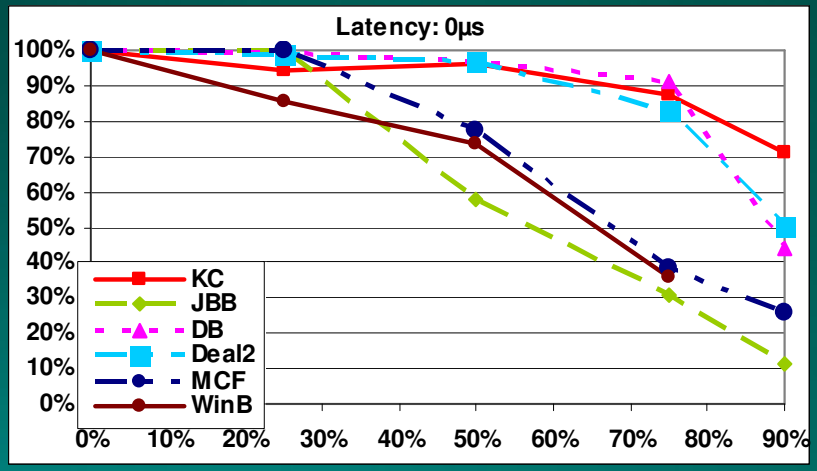
Desktop productivity
- 32b Win - Business Winstone '02 (128MB)

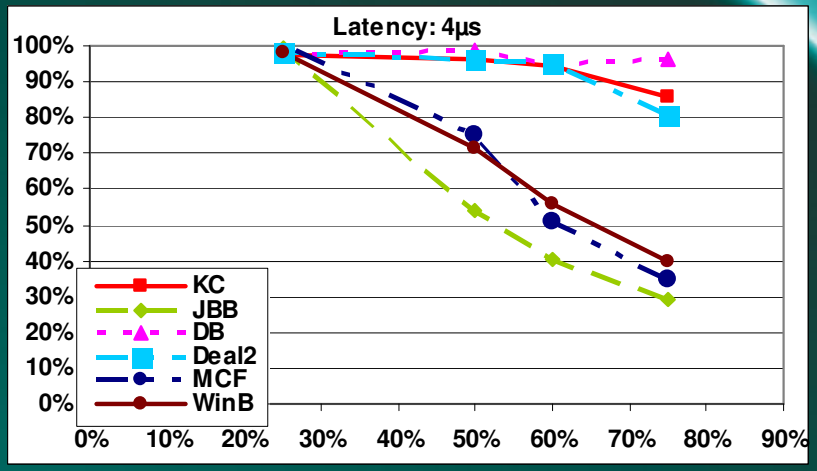
Technical computing
- 32b Linux - Deal II (SPEC 2006) (448 MB)
- 32b Linux - MCF (SPEC 2006) (320 MB)
- 32b Linux - Kernel Compile (192 MB)

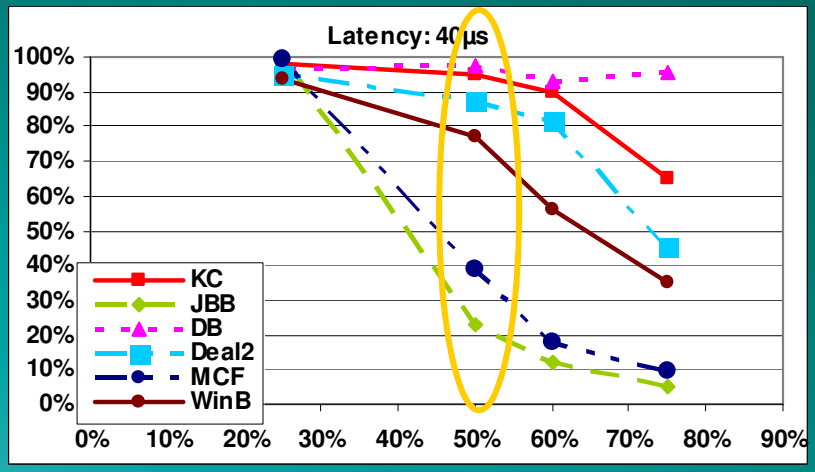


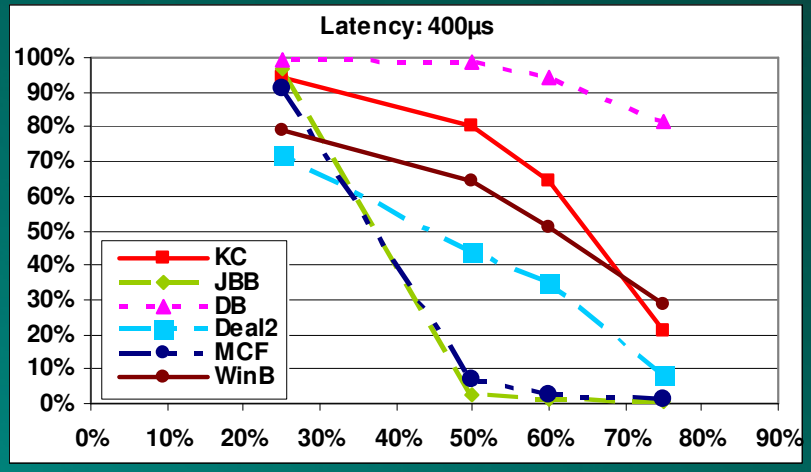
PUT AXIS LABELS DIRECTLY ON GRAPH

M2 Latency









Most benchmarks can handle a 40 µs latency even when M2 is 50% of total memory size

PUT LABELS ON AXES

INCLUSIVE OR EXCLUSIVE?
BE READY FOR THIS QUESTION

M2 Bandwidth

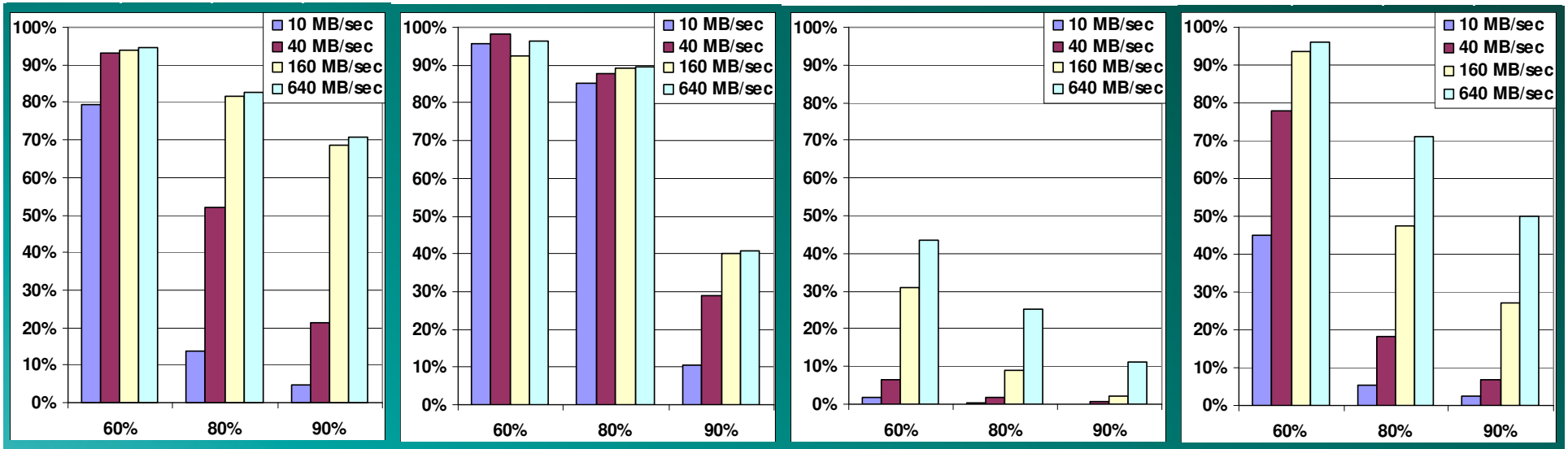
- Wide variety in M2 bandwidth demand (5MB/s – 500 MB/s)
- As M2 fraction increases, the demand for M2 bandwidth increases
- Diminishing returns beyond 640 MB/s BW limit

KC	60%	80%	90%
AVG	4.72	19.0	75.9
PEAK	79.3	83.9	171

DB	60%	80%	90%
AVG	0.13	0.9	44.8
PEAK	19.4	36.8	195

JBB	60%	80%	90%
AVG	191	296	357
PEAK	389	479	471

Deal2	60%	80%	90%
AVG	16	80	256
PEAK	370	516	476

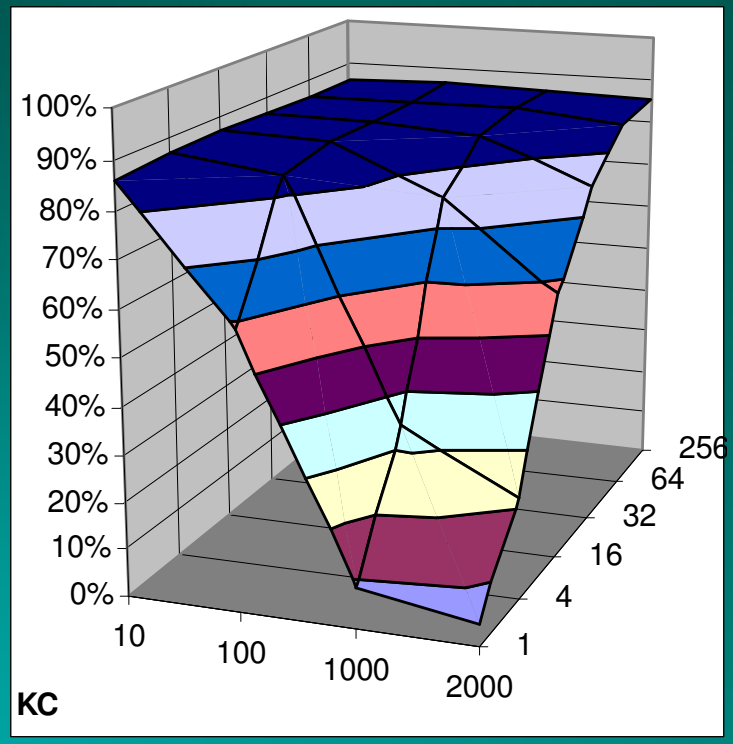


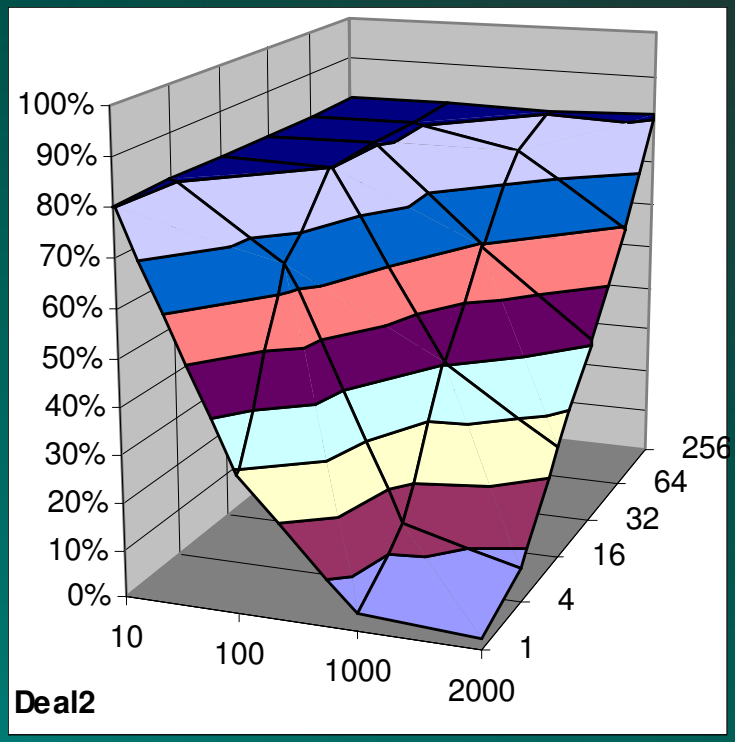
Write BW limit must be >2X average write BW to prevent performance loss

X-axis- Each group is a specific % of M2
-Within each group are varying M2 bandwidth limits
Y-axis -normalized performance

M2 Write Buffering

- Modest write buffering effectively hides even long M2 write latency





256 page buffer hides 2000 µS of write latency

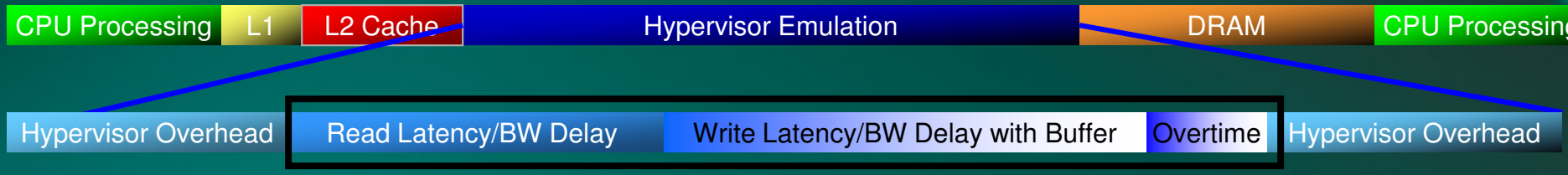
64 page buffer hides 1000 µS of write latency

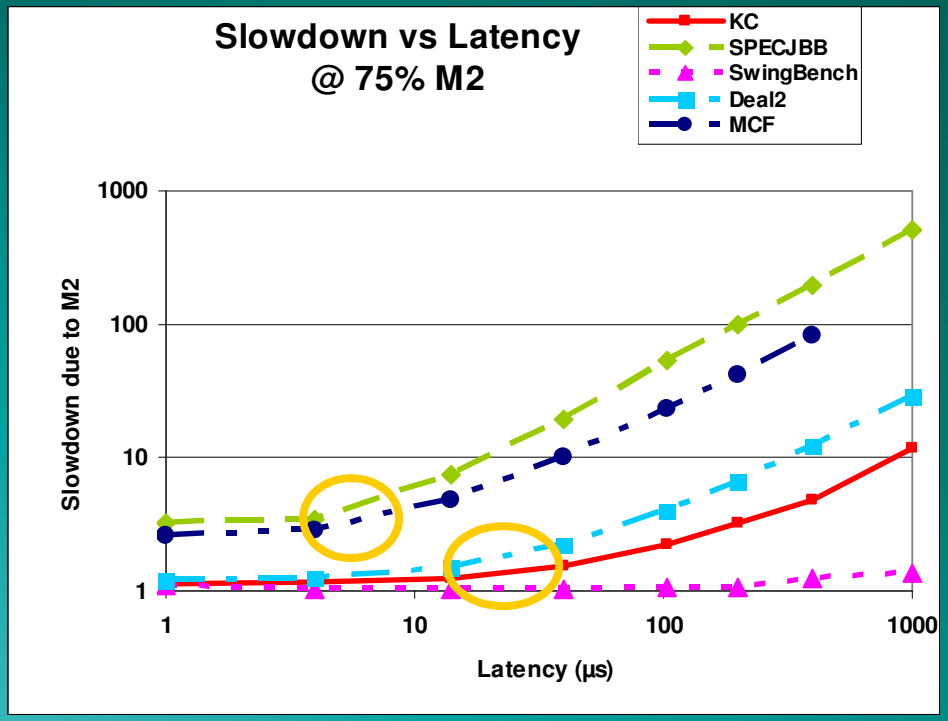
16 page buffer hides 100 µS of write latency

LABELS ON AXES!

Limitations

M2 memory access in hybrid memory systems





- Overhead -The time it takes for the program to enter the trace handler
  - Difficult to quantify
- Overtime – Occurs when incurred delay is greater than desired. Most common with low M2 latency settings
- Minimum observable latency:
  - For high bandwidth benchmarks, it is 4 µs.
  - For low bandwidth benchmarks, it is ~10 µs

Key Results

- Demonstrated use of virtualization platform for computer architecture performance modeling
  - Full workload simulation at high speed
  - Workloads spanning multiple operating systems
  - Modest additional development
  - Some limits on explorable design space

- Case Study: Hybrid Memory System
  - Wide variety in benchmark sensitivity to M2 read latency
  - Modest write buffering hides very long M2 write latency
  - Only certain workloads are amenable to hybrid memory systems, *when total memory is tightly provisioned*
    - → Real systems not usually as tightly provisioned

Q & A?

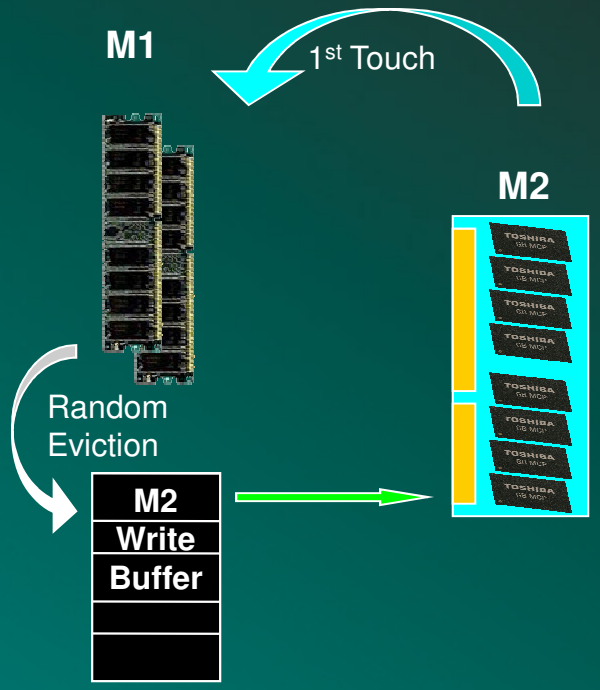
- If you have questions, feel free to contact me
  btsang@rambus.com

Backup

Assumptions

Our assumptions :

1. CPU only uses memory from M1
   On first touch, memory from M2 must be moved to M1
2. Reads and writes will not have the same performance
3. To mitigate the impact of slow writes, a write buffer would be implemented
4. Flash is accessed in units of the architecture pagesize (4KB)
5. A random page is evicted from M1



M1 (DRAM) memory access

M2 memory access in hybrid memory systems

Measuring performance

1. Configure VM
2. Run the benchmark
3. Record relevant metrics and statistics
   - Execution time is measured through an external time source
   - Benchmarks may report performance by rate or by execution time
4. Compare metrics to the baseline configuration
   - Baseline configuration- “M1 only” specifically sized for the benchmark

The host machine runs only 1 VM bound to a dedicated host processor core with ample memory