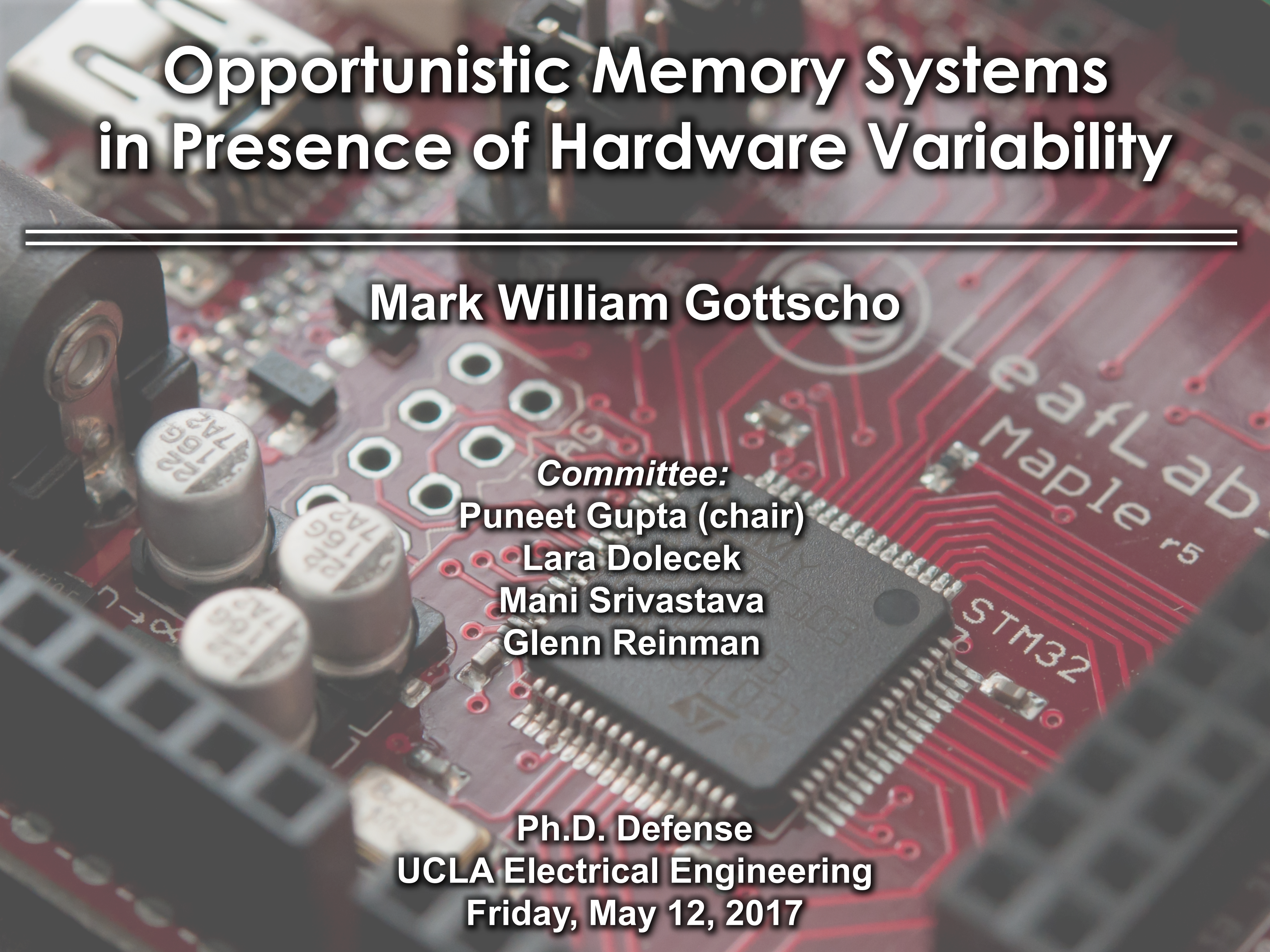


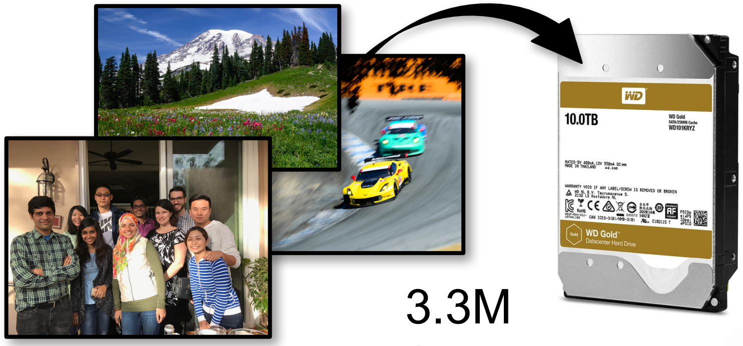
Opportunistic Memory Systems in Presence of Hardware Variability

Mark William Gottscho

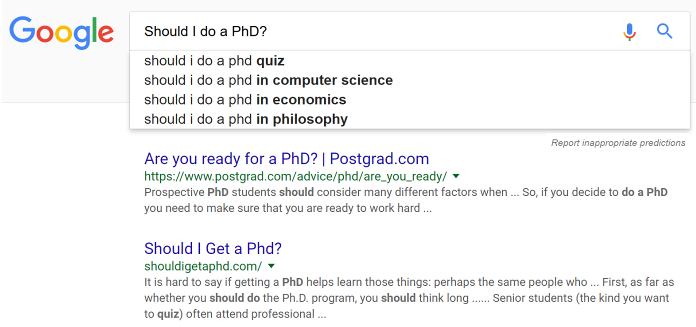
Committee:
Puneet Gupta (chair)
Lara Dolecek
Mani Srivastava
Glenn Reinman

Ph.D. Defense
UCLA Electrical Engineering
Friday, May 12, 2017

Memory is Essential



3.3M photos


Should I do a PhD?

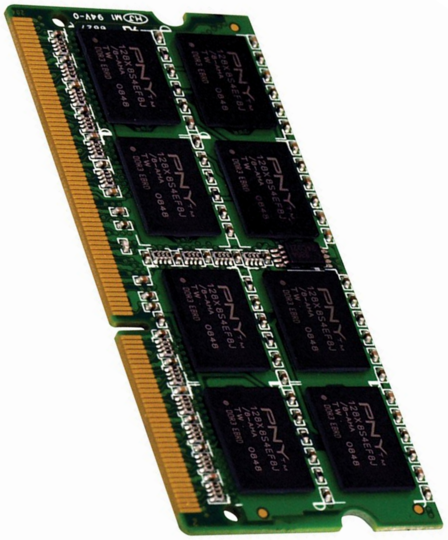
should i do a phd **quiz**
should i do a phd **in computer science**
should i do a phd **in economics**
should i do a phd **in philosophy**

Report inappropriate predictions

Are you ready for a PhD? | Postgrad.com
https://www.postgrad.com/advice/phd/are_you_ready/ ▾
Prospective **PhD** students **should** consider many different factors when ... So, if you decide to **do a PhD** you need to make sure that you are ready to work hard ...

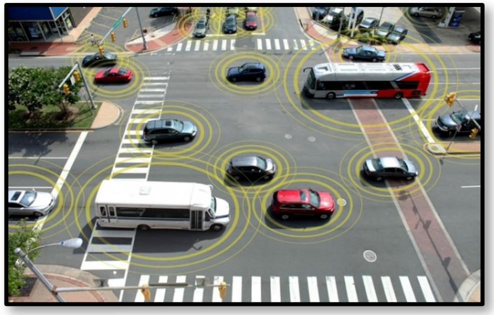
Should I Get a Phd?
shouldigetaphd.com/ ▾
It is hard to say if getting a **PhD** helps learn those things: perhaps the same people who ... First, as far as whether you **should do** the Ph.D. program, you **should** think long ...... Senior students (the kind you want to **quiz**) often attend professional ...

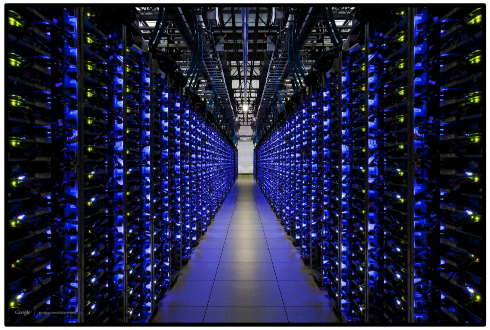
2.3M Google searches per second



YouTube

400 hours of video uploaded per minute





Hardware Variability in Memory

Hardware variability is particularly problematic for memories:

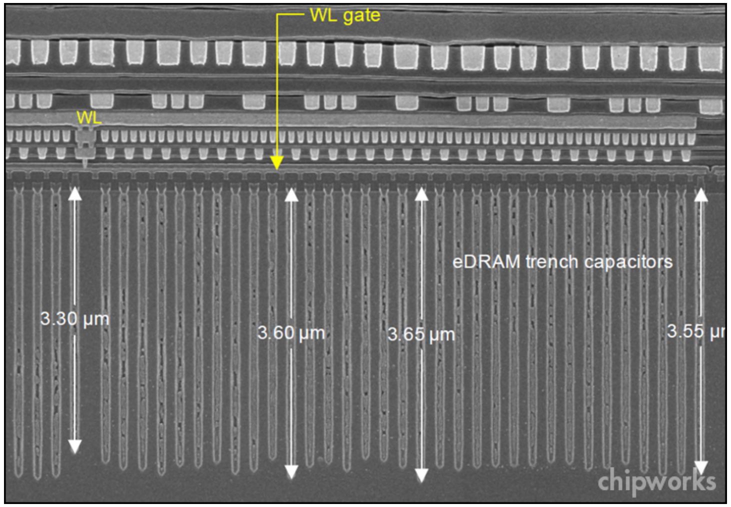
1. Smallest and densest device/circuit features
2. Large fraction of the chip area budget
3. Must permit instability in order to be rewritable

Memories are particularly susceptible to:

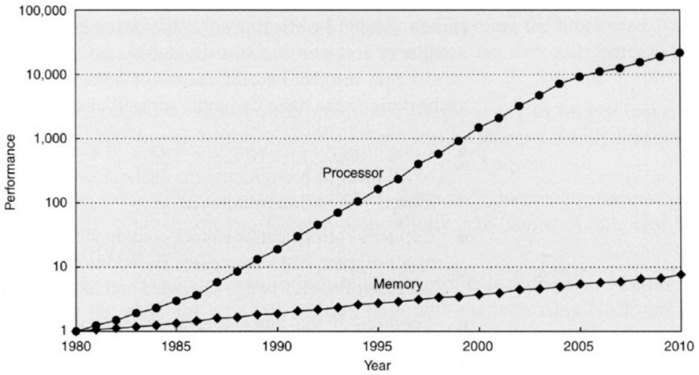
1. Manufacturing defects
2. Parametric variations
3. The operating environment

Memory wall often limits:

1. **Energy efficiency**
2. **System resiliency**



32nm eDRAM in the IBM Power 7 Processor [ChipWorks]

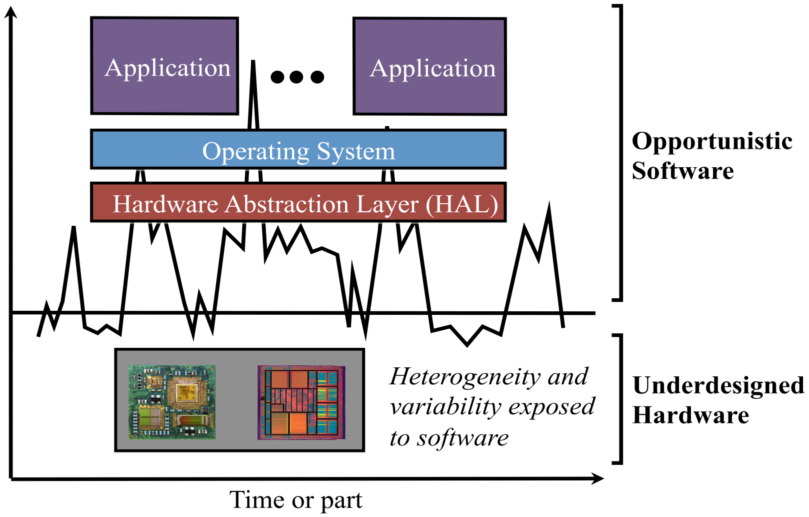


[Hennessy & Patterson '12]

Better-Than-Worst-Case Design

Underdesigned and Opportunistic Computing machines

[Gupta et al. TCAD'13]



Application

Application

Operating System

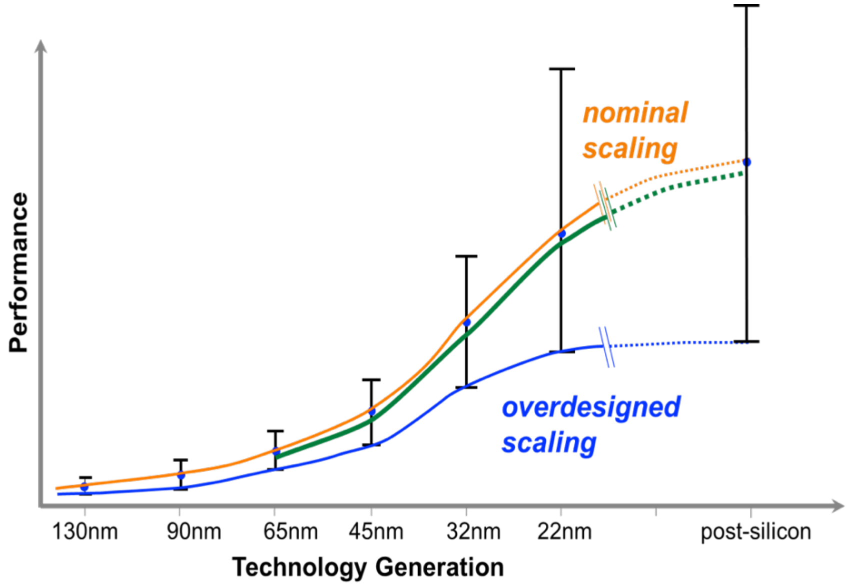
Hardware Abstraction Layer (HAL)

Heterogeneity and variability exposed to software

Opportunistic Software

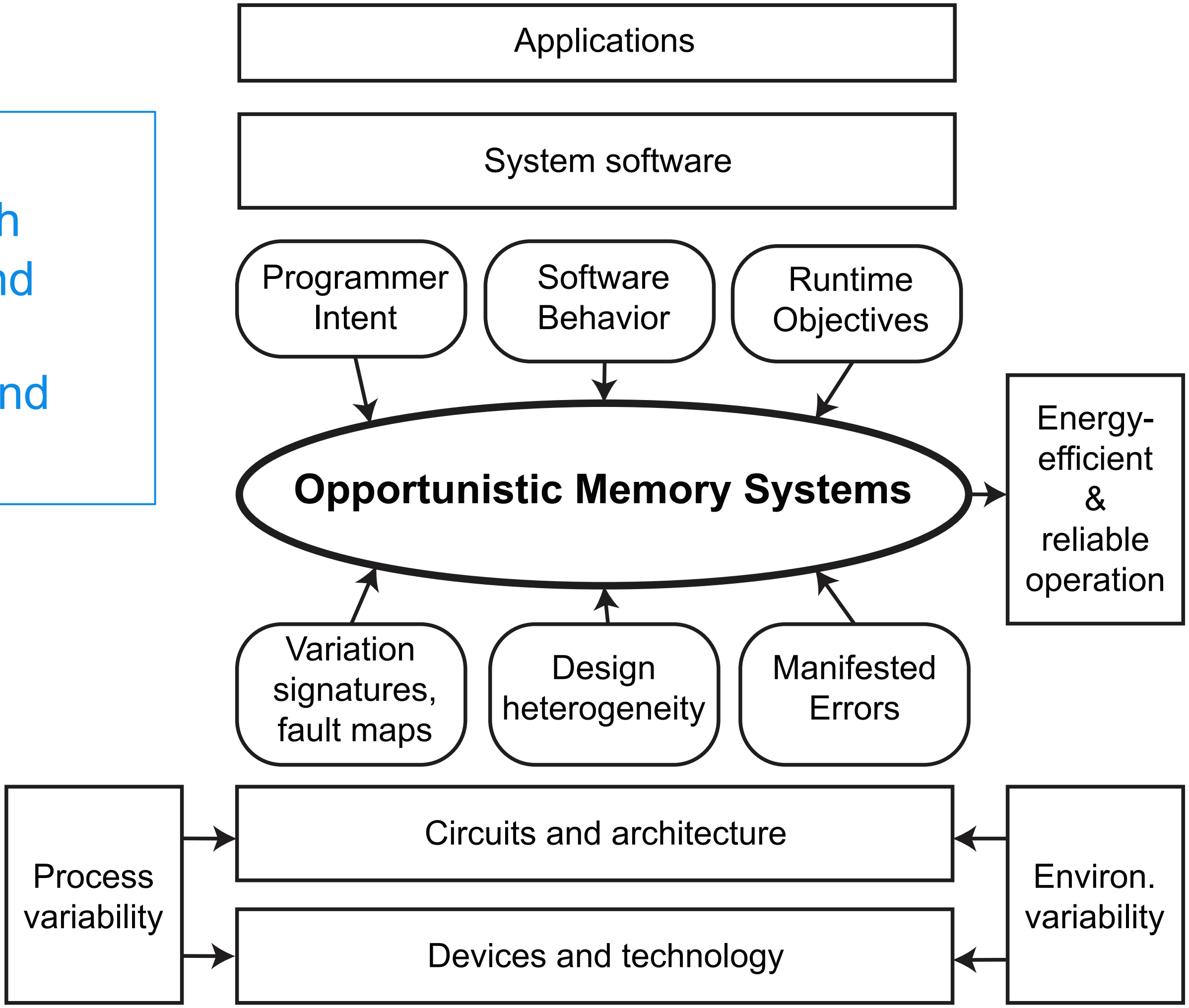
Underdesigned Hardware

Time or part



My Framework

Opportunistic Memory Systems exploit & cope with hardware variations within and across individual chips for improved energy efficiency and resiliency.



Overview of My Dissertation

Part 1: Opportunistically Exploiting Memory Variability

1. ViPZonE: Saving Energy in DRAM Main Memory
   with Power Variation-Aware Memory Management
2. DPCS: Saving Energy in SRAM Caches
   with Dynamic Power/Capacity Scaling
3. X-Mem: Case Studies on Memory Performance Variability
   with the new Extensible Memory Characterization Tool

Part 2: Opportunistically Coping with Memory Errors

4. Performability: Exploring the Impact of Corrected Memory Errors
   by quantifying and analytically modeling their performance effects
5. SDECC: Recovering from Detected-but-Uncorrectable Memory Errors
   with Software-Defined Error-Correcting Codes
6. ViFFTo: Improving Reliability of Embedded Scratchpad Memories
   with Virtualization-Free Fault Tolerance

Agenda

- Introduction
- Part 1: Exploiting Variability
  - ViPZonE
  - DPCS
  - X-Mem
- Part 2: Coping with Errors
  - Performability
  - SDECC
  - ViFFTo
- Conclusion and Directions for Future Work

ViPZonE: Saving Energy in DRAM Main Memory using Power Variation-Aware Memory Management

Collaborators:

Dr. Luis A. D. Bathen (UC Irvine)
Prof. Nikil Dutt (UC Irvine)
Prof. Alex Nicolau (UC Irvine)
Prof. Puneet Gupta (UCLA)

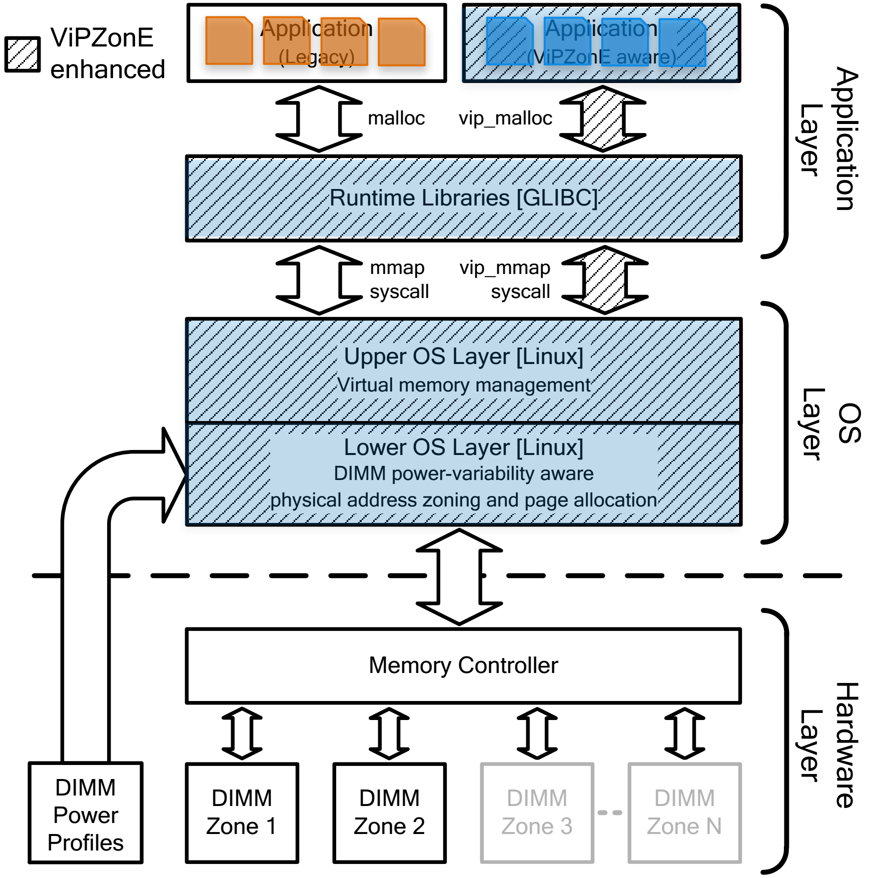
Publications:

Gottscho et al., ESL'12
Bathen et al., CODES+ISSS'12
Dutt et al., ASP-DAC'13
Gottscho et al., TC'15
Wanner et al., it'15

Chapter 2

Summary of ViPZonE

[Gottscho ESL'12, Bathen CODES+ISSS'12, Dutt ASP-DAC'13, Gottscho TC'15, Wanner it'15]



1 ViPZonE-enabled apps tell OS how to allocate virtual pages w/ special variant of `malloc()` in modified standard C library

2 ViPZonE-enabled glibc tells OS how to allocate virtual pages w/ special variant of `mmap()` syscall

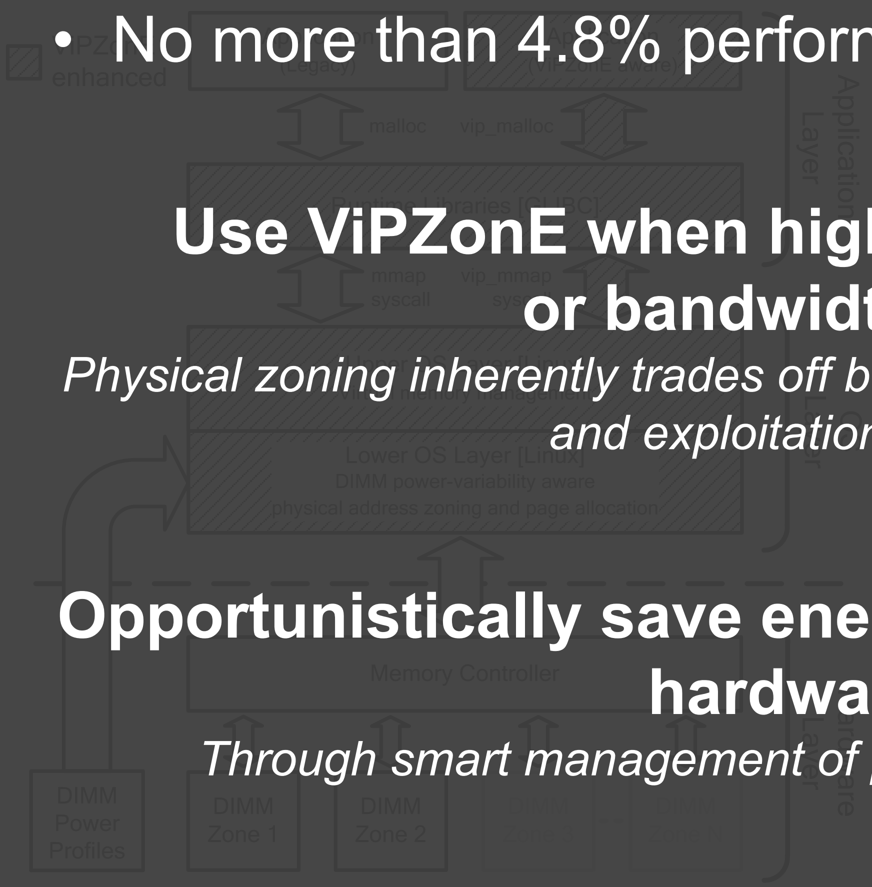
3 Kernel's physical page allocator attempts to map allocated virtual page in particular memory device

Legacy app does not exploit power variability!

ViPZonE app consolidates pages onto low power zones!

Summary of ViPZonE

- Up to 27.8% energy savings on Intel Sandy Bridge/DDR3 testbed desktop
- No more than 4.8% performance degradation



Use ViPZonE when high memory-level parallelism or bandwidth is not needed

Physical zoning inherently trades off benefits of striping for resource consolidation and exploitation of device variations

Opportunistically save energy in today's systems with no hardware changes

Through smart management of physical memory variation signatures

apps tell OS how to allocate virtual pages w/ special variant of `malloc()` in modified standard C library

ViPZonE-enabled glibc tells OS how to allocate virtual pages w/ special variant of `mmap()` syscall

Kernel's physical page allocator

page in particular memory device

DPCS: Saving Energy in SRAM Caches with Dynamic Power/Capacity Scaling

Collaborators:

Dr. Abbas BanaiyanMofrad (UC Irvine)
Prof. Nikil Dutt (UC Irvine)
Prof. Alex Nicolau (UC Irvine)
Prof. Puneet Gupta (UCLA)

Publications:

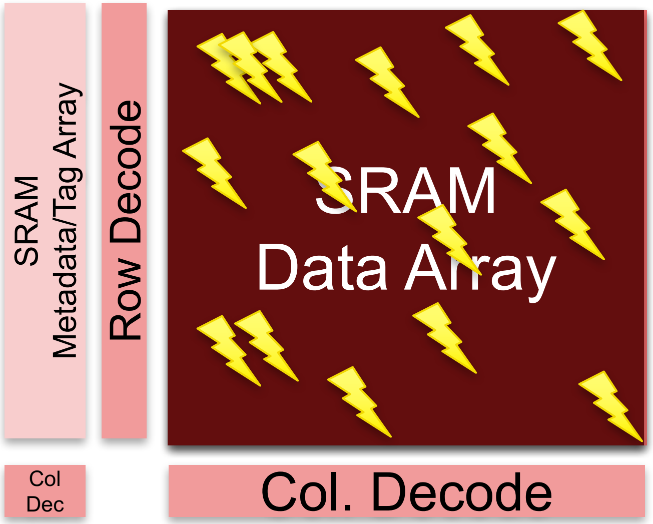
Gottscho et al., DAC'14
Dutt et al., DAC'14
Gottscho et al., TACO'15
Wanner et al., it'15

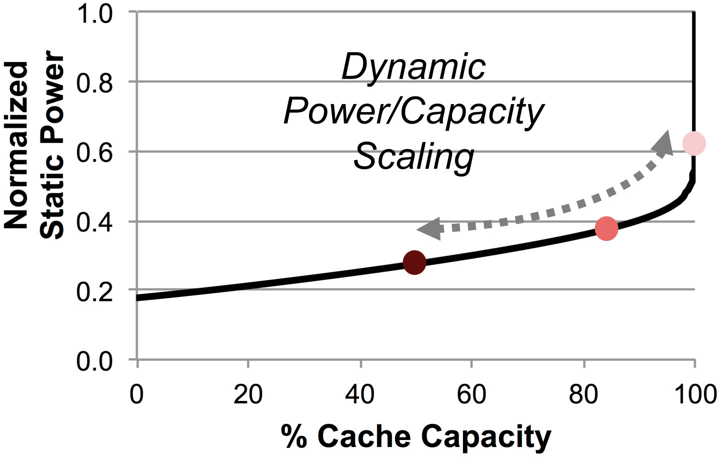
Chapter 3

Summary of DPCS

[Gottscho DAC'14, Dutt DAC'14, Gottscho TACO'15, Wanner it'15]

- Pre-characterize SRAM faults using BIST
- Encode min non-faulty VDD on per-block basis
  - Store in modified tag array with 2 extra bits per block



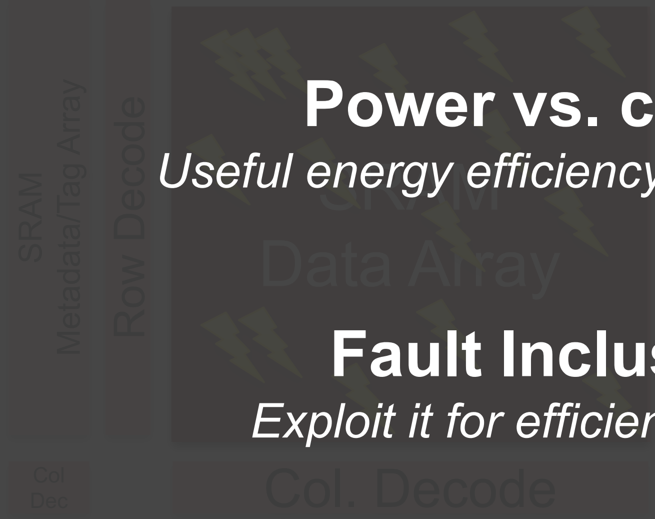


- High performance mode
  - Full VDD & cache capacity
- Low power mode
  - Reduced VDD, disabled faulty blocks

Summary of DPCS

[Gottscho DAC'14, Dutt DAC'14, Gottscho TACO'15, Wanner it'15]

- Up to 79% total cache energy savings
- Up to 26% total system energy savings
- Average 2.24 % performance overhead
- 6% total cache area overhead



Power vs. capacity tuning

Useful energy efficiency knob, complements DVFS

Fault Inclusion Property

Exploit it for efficient storage of fault maps

Opportunistic approach to energy-efficient caches

Leverage variability without harming reliability or performance

X-Mem: A New Tool for Case Studies on Memory Performance Variability

Collaborators:

Dr. Sriram Govindan (Microsoft)
Dr. Bikash Sharma (Microsoft)
Dr. Mohammed Shoaib (Microsoft Research)
Prof. Puneet Gupta (UCLA)

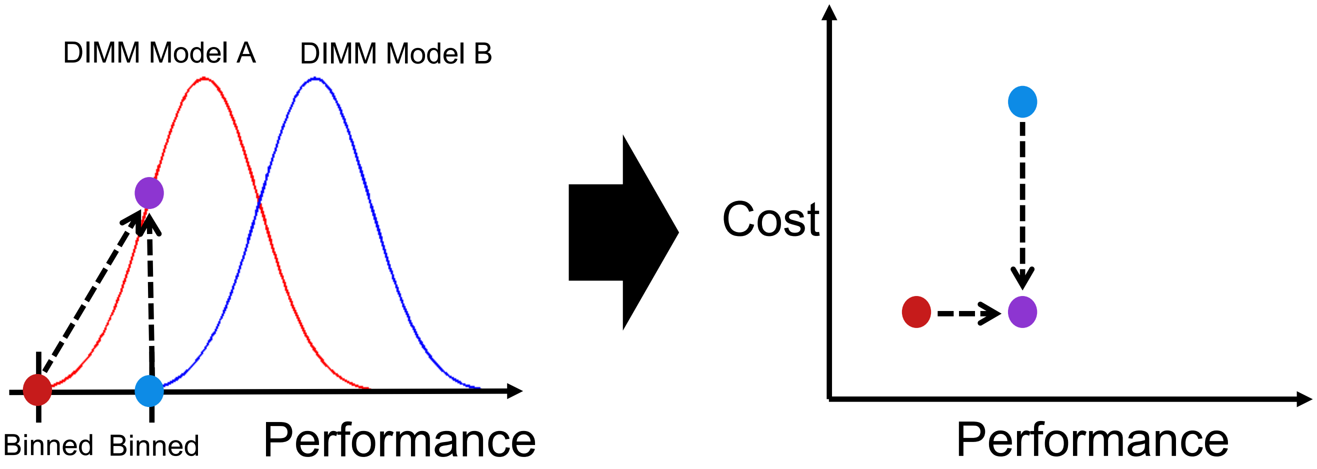
Publications:

Gottscho et al., ISPASS'16

Chapter 4

Summary of X-Mem

[Gottscho ISPASS'16]



Tool	Thru-put	Lat.	Loaded Lat.	Multi-Thrd.	NUMA	Lrg. Pages	Power	Cache & Mem.	Native Linux	Native Win.	x86	x86-64	ARM	Vector Inst.	Open Src.	Lang.	(A) Acc. Patt. Divers.	(B) Platf. Var.	(C) Metric Flex.	(D) Tool Extens.
STREAM v5.10 [13]	✓			○				○	✓		✓	○	○		✓	C, FORTRAN				
STREAM2 v0.1 [14]	✓			○				○	✓		✓	○	○		✓	FORTRAN				
lmbench3 [15]	✓	✓		✓				○	✓		✓	○	○		✓	C				
TinyMemBench v0.3.9 [16]	✓	✓				✓		✓	✓		✓	○	○	✓	✓	C				
mlc v2.3 [17]	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓								
X-Mem v2.2.3 [18], [19]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	C++	✓	✓	✓	✓

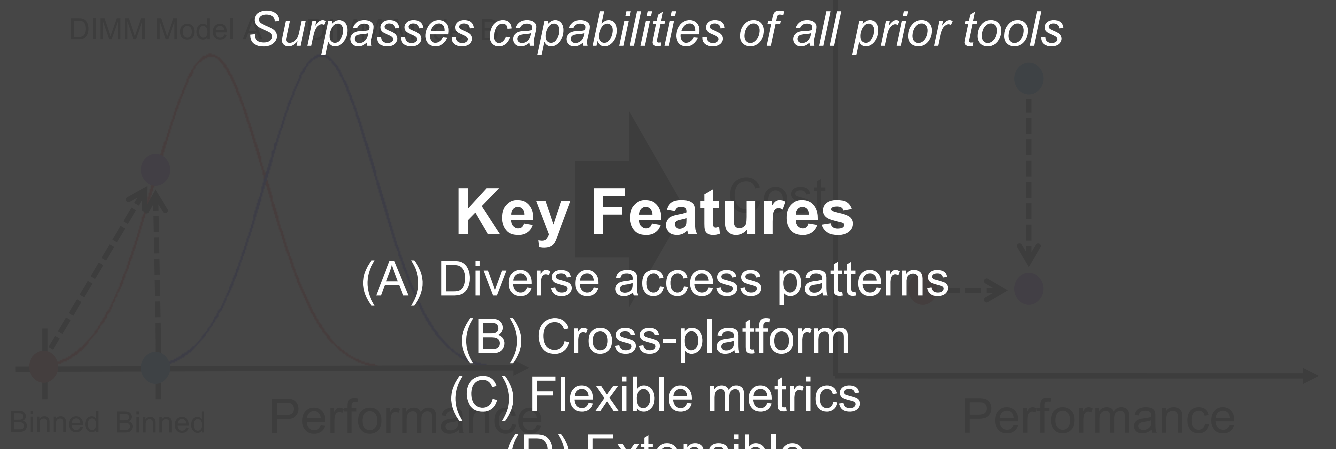
Code at http://nanocad-lab.github.io/X-Mem

Summary of X-Mem

[Gottscho ISPASS'16]

New flexible tool for characterizing memory systems

Surpasses capabilities of all prior tools



Key Features

(A) Diverse access patterns
(B) Cross-platform
(C) Flexible metrics
(D) Extensible

Three case studies

Explored efficacy of opportunistic variation-aware DRAM latency tuning

Tool	Thru-put	Lat.	Loaded Lat.	Multi-Thrd.	NUMA	Lrg. Pages	Power	Cache & Mem.	Native Linux	Native Win.	x86	x86-64	ARM	Vector Inst.	Open Src.	Lang.	(A) Acc. Patt. Divers.	(B) Platf. Var.	(C) Metric Flex.	(D) Tool Extens.
STREAM v5.10 [13]	✓			○		[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]			✓	C, FORTRAN				
STREAM2 v0.1 [14]	✓			○				○	✓		✓	○	○		✓	FORTRAN				
lmbench3 [15]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]				
TinyMemBench v0.3.9 [16]	✓	✓				✓		✓	✓		✓	○	○	✓	✓	C				
mlc v2.3 [17]	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓								
X-Mem v2.2.3 [18], [19]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	C++	✓	✓	✓	✓

Code at http://nanocad-lab.github.io/X-Mem

Agenda

- Introduction
- Part 1: Exploiting Variability
  - ViPZonE
  - DPCS
  - X-Mem
- Part 2: Coping with Errors
  - Performability
  - SDECC
  - ViFFTo
- Conclusion and Directions for Future Work

Performability: The Impact of Corrected Memory Errors on Performance

Collaborators:

Dr. Mohammed Shoaib (Microsoft Research)
Dr. Sriram Govindan (Microsoft)
Dr. Bikash Sharma (Microsoft)
Dr. Di Wang (Microsoft Research)
Prof. Puneet Gupta (UCLA)

Publications:

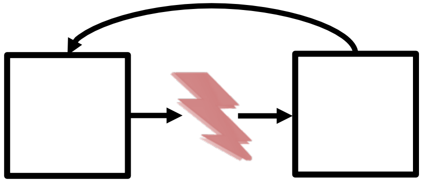
Gottscho et al., CAL'16

Chapter 5

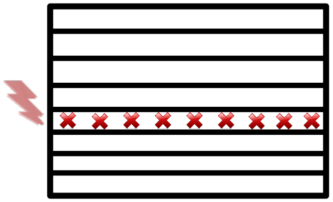
How Fault Tolerance Impacts Cloud Application Performance

Dynamic FW/SW Error Reporting, Recovery, & Prevention

Error Logging

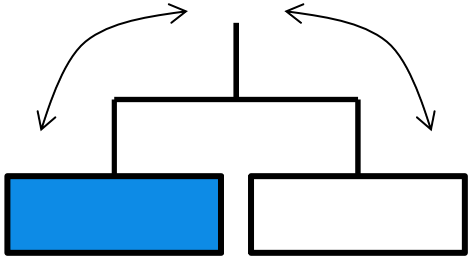


Checkpointing

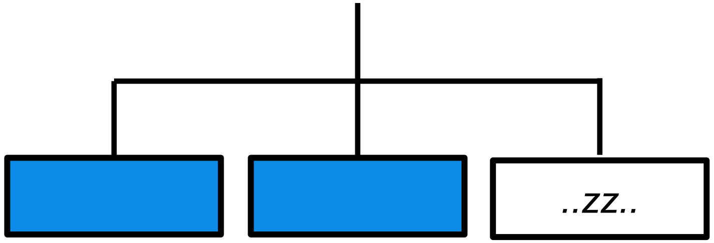


Page Retirement

Static HW Over-Provisioning

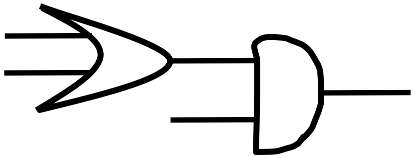


Mirroring



Sparing

Static HW EDAC Capabilities



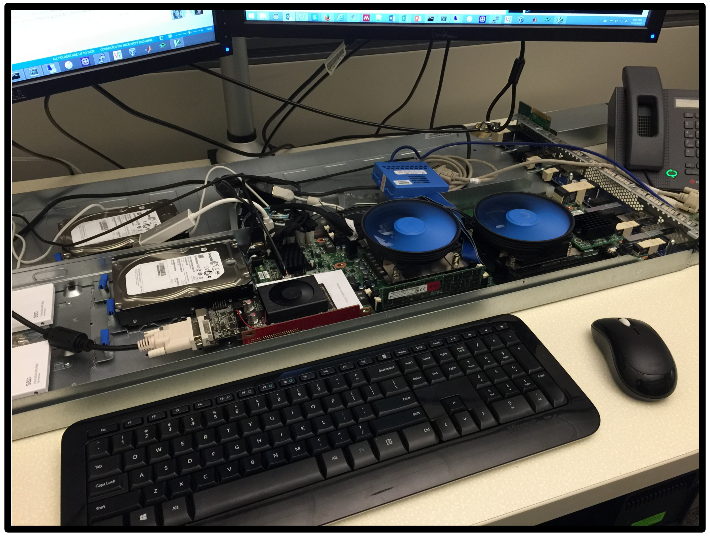
$Hc'=0$

ECC Encode/Decode

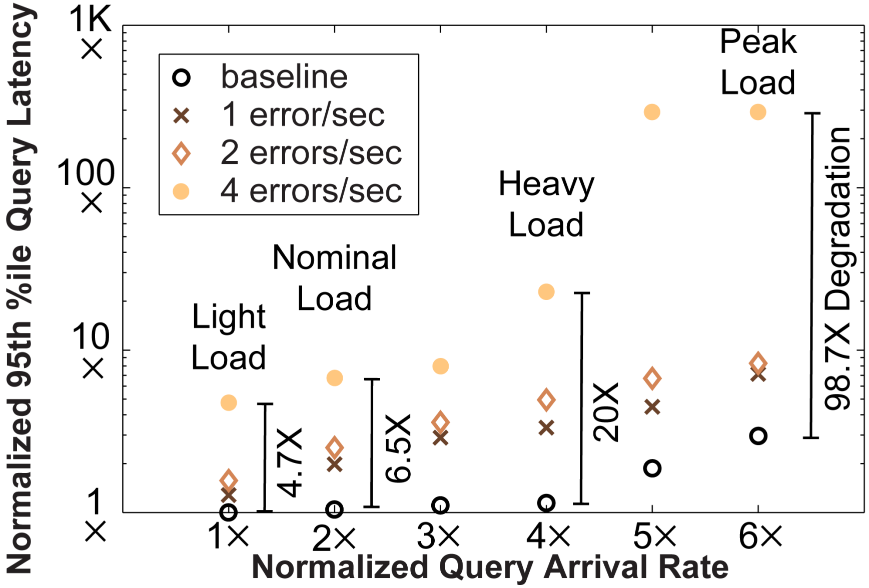
Measured Performance Degradation

[Gottscho CAL'16]

X-Mem extended: controlled injections of correctable memory errors in production-spec cloud server



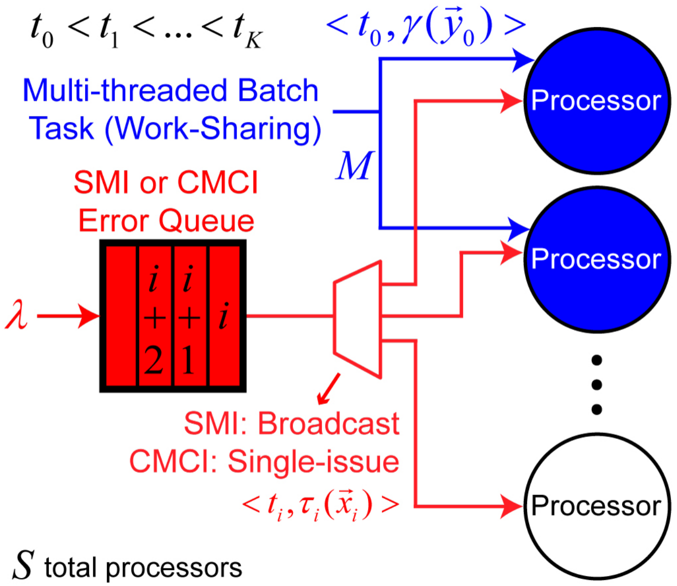
Interactive application (web search)

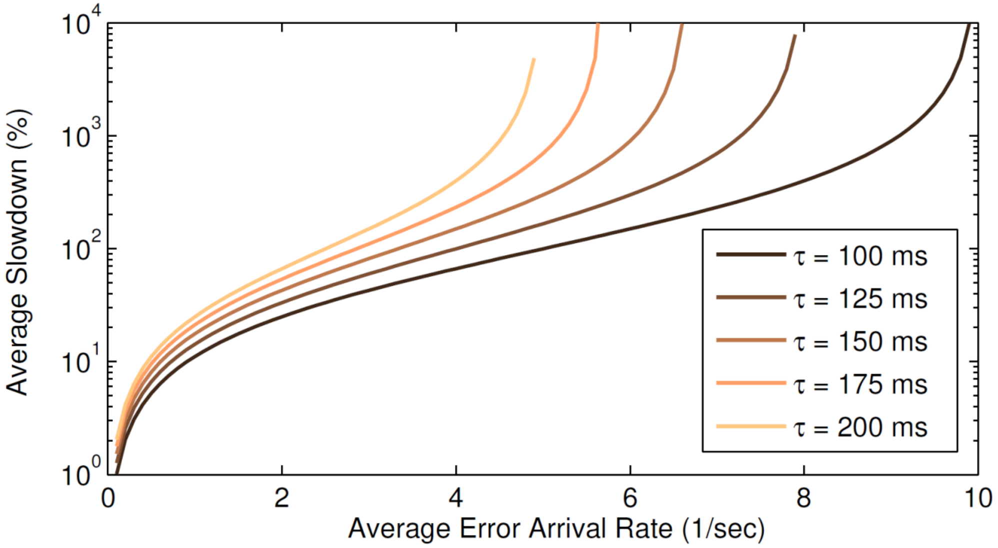


Corrected memory errors can have severe impact on application performance!

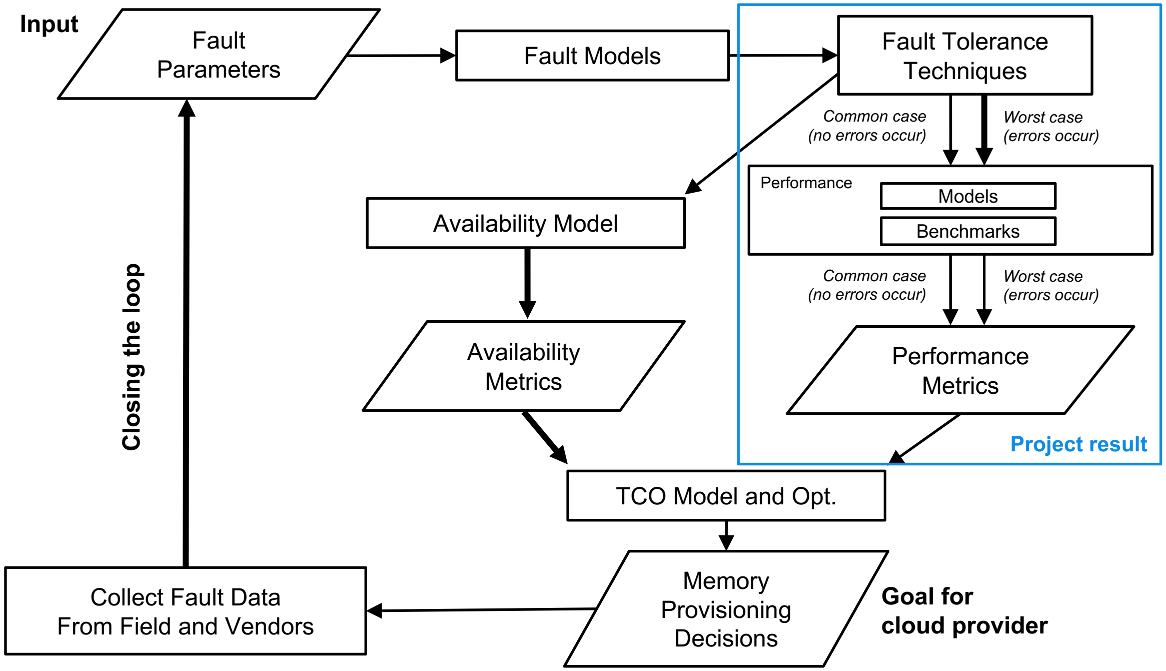
Queuing-Theoretic Models for Performance Degradation

Batch applications on multiprocessors with broadcast error handling





Summary of Performability


Input
Fault Parameters
Fault Models
Fault Tolerance Techniques
Common case (no errors occur)
Worst case (errors occur)
Performance
Models
Benchmarks
Common case (no errors occur)
Worst case (errors occur)
Performance Metrics
Project result
Availability Model
Availability Metrics
TCO Model and Opt.
Memory Provisioning Decisions
Goal for cloud provider
Collect Fault Data From Field and Vendors
Closing the loop

Recommendations

- *Integrate* performability models and empirical data into high-level TCO models
- *Reduce* the overhead of hardware error reporting via architecture/firmware/OS optimizations
- *Prevent* faults proactively using page retirement and variation-aware memory management

Agenda

- Introduction
- Part 1: Exploiting Variability
  - ViPZonE
  - DPCS
  - X-Mem
- Part 2: Coping with Errors
  - Performability
  - SDECC
  - ViFFTo
- Conclusion and Directions for Future Work

SDECC: Recovering from Detected-but-Uncorrectable Memory Errors using Software-Defined Error-Correcting Codes

Collaborators:

Clayton Schoeny (UCLA)
Prof. Lara Dolecek (UCLA)
Prof. Puneet Gupta (UCLA)

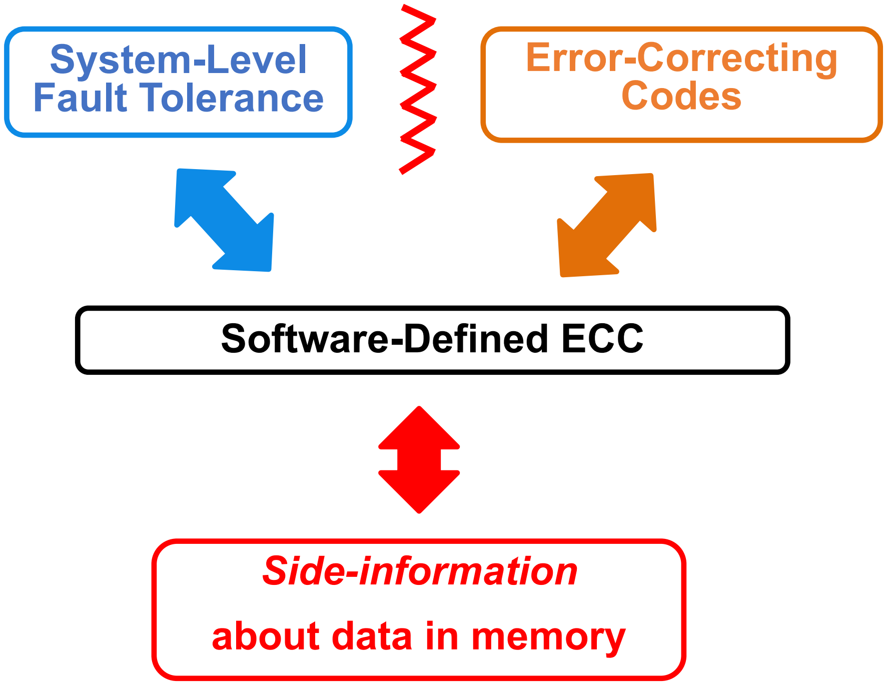
Publications:

Gottscho et al., SELSE’16
Gottscho et al., DSN-W’16
Gottscho et al., 2017 manuscript submitted and under peer review

Chapter 6

SDECC Concept

[Gottscho SELSE'16, Gottscho DSN-W'16, Gottscho '17]

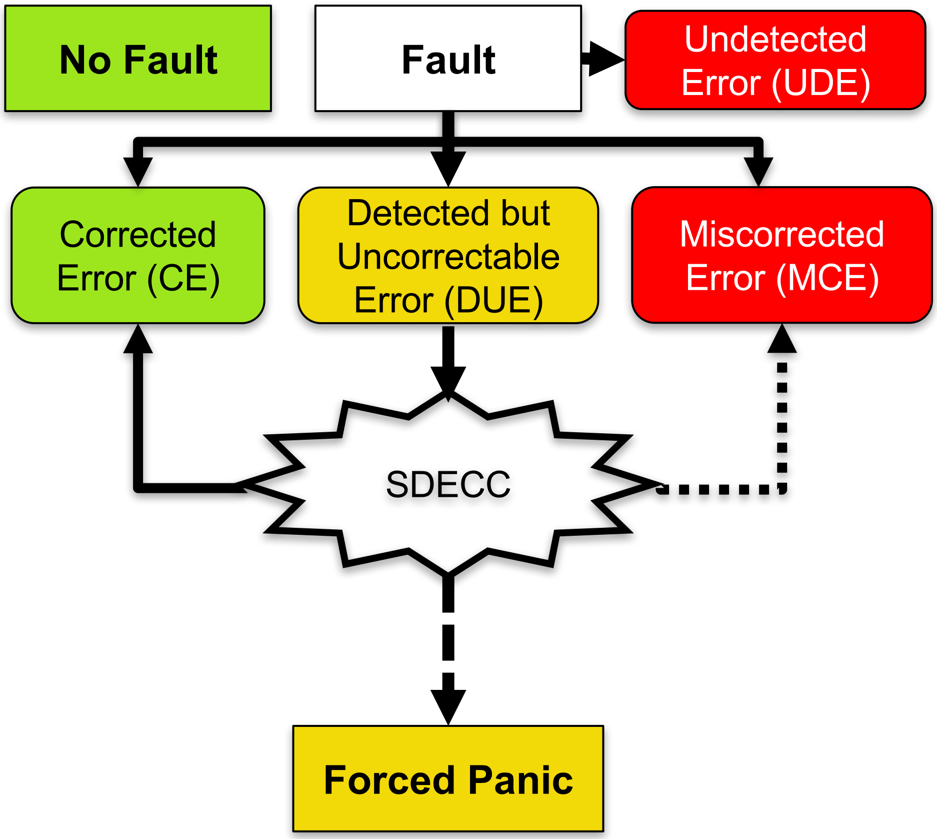


System-Level Fault Tolerance

Error-Correcting Codes

Software-Defined ECC

Side-information about data in memory



No Fault

Fault

Undetected Error (UDE)

Corrected Error (CE)

Detected but Uncorrectable Error (DUE)

Miscorrected Error (MCE)

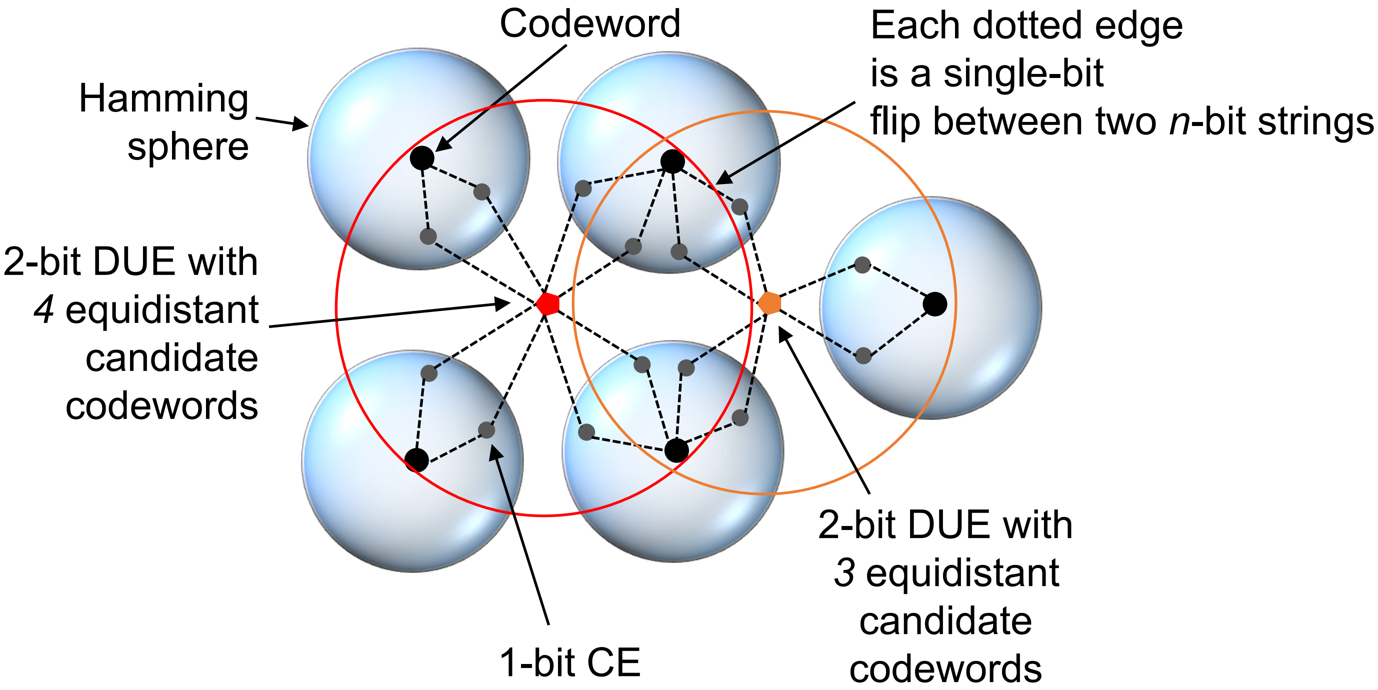
SDECC

Forced Panic

Candidate Codewords

[Gottscho SELSE'16, Gottscho DSN-W'16, Gottscho '17]

Example using SECDED (concept applies generally)


Codeword
Each dotted edge is a single-bit flip between two *n*-bit strings
Hamming sphere
2-bit DUE with *4* equidistant candidate codewords
1-bit CE
2-bit DUE with *3* equidistant candidate codewords

Analysis of Existing ECC Codes

[Gottscho '17]

Class of Code	Type of Code	n	k	t	q	# ways DUE	Avg. # C.C.	Baseline Prob. Success
32-bit SECDED	[Hsiao IBM Jour. '70]	39	32	1	2	741	12.04	8.50%
32-bit SECDED	[Davydov Trans.IT '91]	39	32	1	2	741	9.67	11.70%
64-bit SECDED	[Hsiao IBM Jour. '70]	72	64	1	2	2556	20.73	4.97%
64-bit SECDED	[Davydov Trans.IT '91]	72	64	1	2	2556	16.62	6.85%
32-bit DECTED	-	39	32	2	2	14190	4.12	28.20%
64-bit DECTED	-	79	64	2	2	79079	5.40	20.53%
128-bit SSCDSD (ChipKill-Correct)	[Kaneda Trans. Comp '82]	36	32	1	16	141750	3.38	39.88%

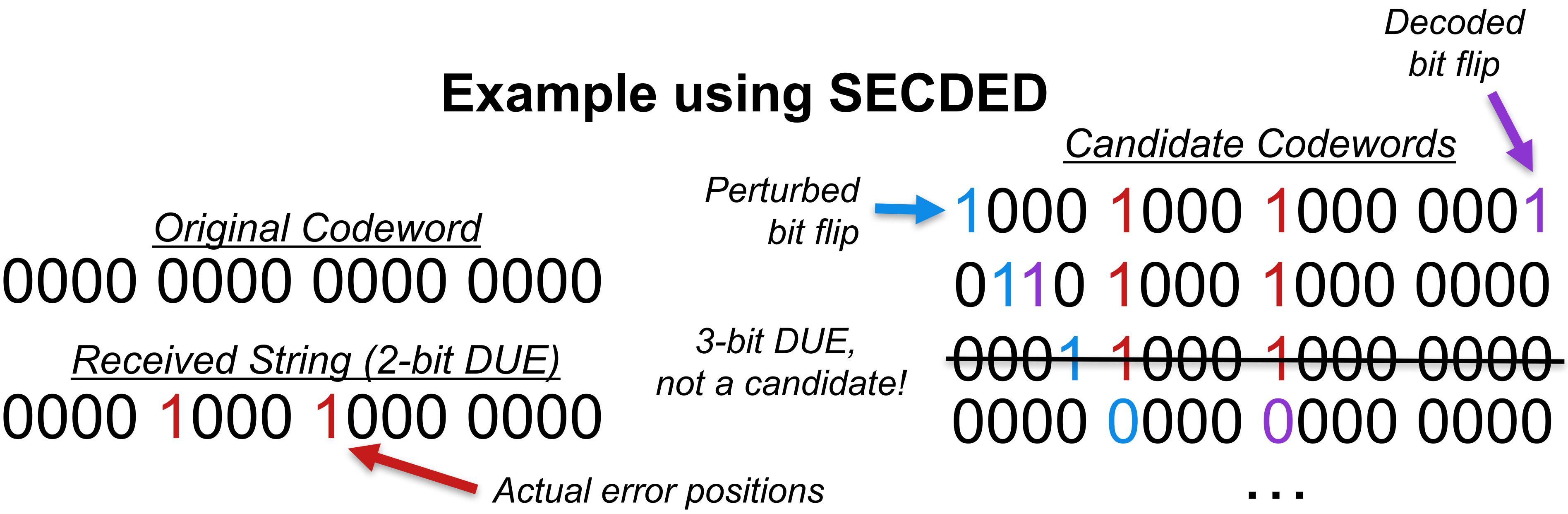
Computing Candidate Codewords

[Gottscho '17]

Algorithm

```
For each symbol-wise error position
    For each symbol-wise error value
        Perturb received string using current position/value
        ECC-decode the perturbed string
        If decoder produces a codeword
            Add codeword to list of candidates
```

Example using SECDED



Original Codeword

0000 0000 0000 0000

Received String (2-bit DUE)

0000 1000 1000 0000

Actual error positions

Perturbed bit flip

Decoded bit flip

Candidate Codewords

1000 1000 1000 0001

0110 1000 1000 0000

~~0001 1000 1000 0000~~

3-bit DUE, not a candidate!

0000 0000 0000 0000

. . .

Exploiting Data Side Information in Memory

[Gottscho SELSE'16, Gottscho DSN-W'16, Gottscho '17]

Data types

- `uint32_t`, `double`, pointers, packed arrays, classes…

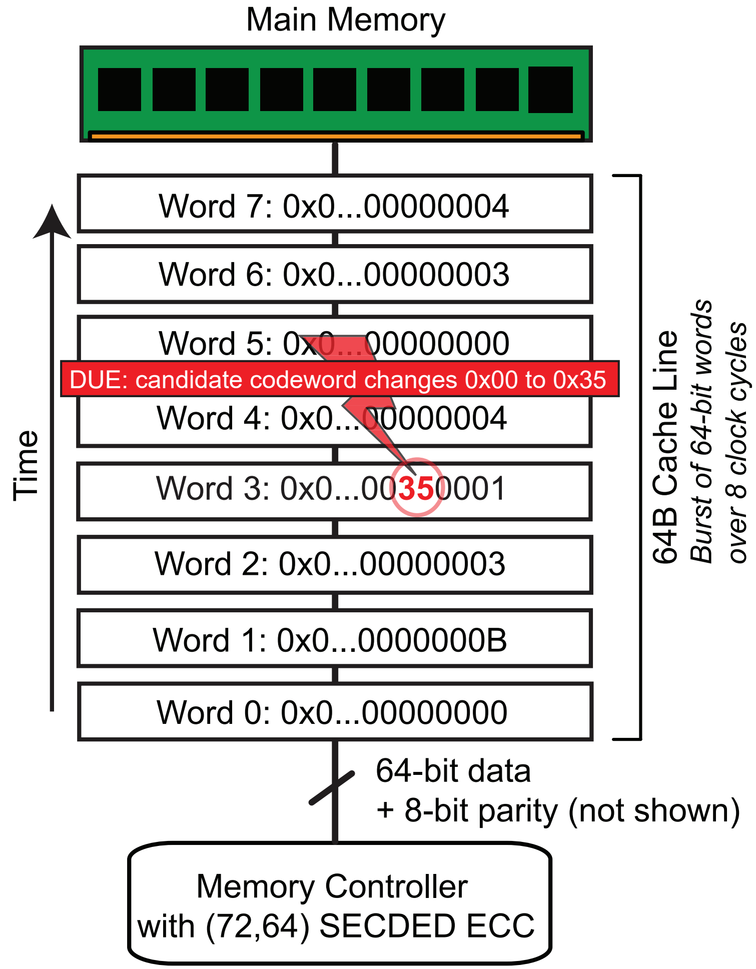
Object states

- Assertions, invalid pointers…

Data correlation

- Previously used for compression

[Yang MICRO'00, Alameldeen '04, Pekhimenko PACT'12]



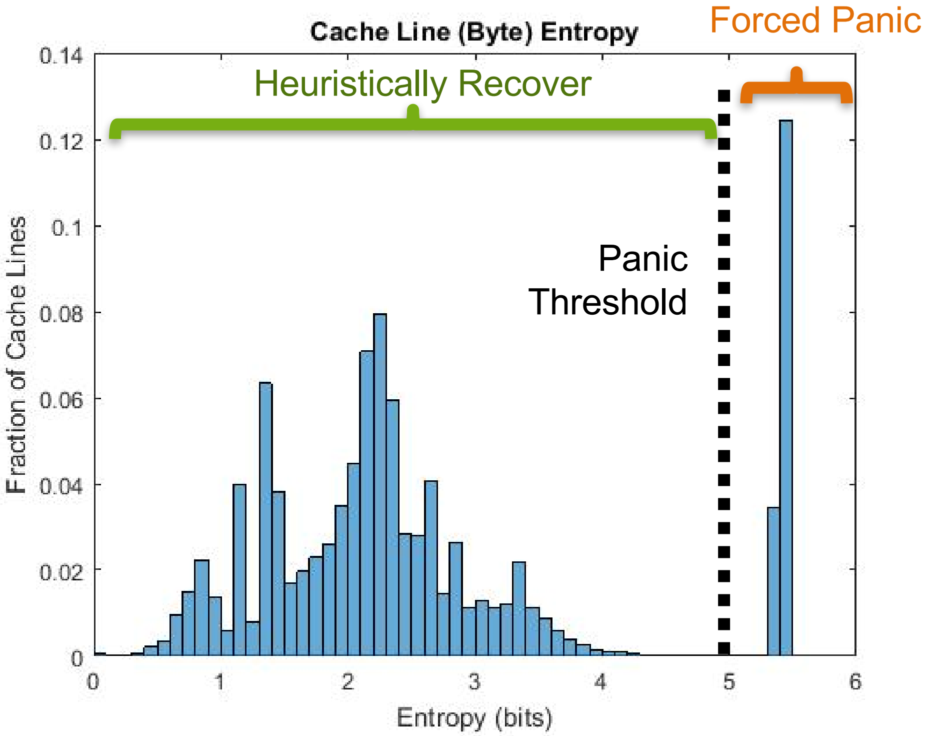
Data Entropy-based Recovery Policy

[Gottscho '17]

- **Use entropy to determine most-correlated candidate codeword**
  - High entropy detected → *force a panic*
  - Low entropy detected → *heuristically recover*

x_i: Value of byte *i* in 64B cache line

$$\boldsymbol{Entropy}:\ H(X) = -\sum_{i=1}^{64} P(x_i)\log_2 P(x_i)$$

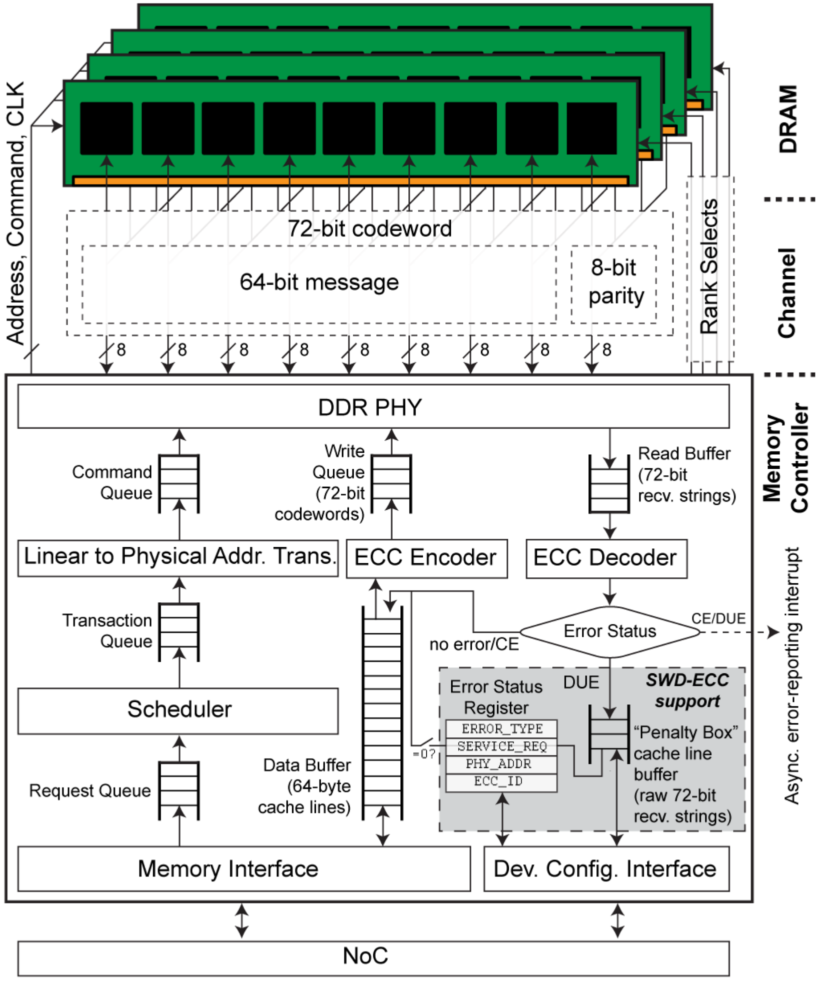


Architectural Support: SDECC for Main Memory

[Gottscho '17]

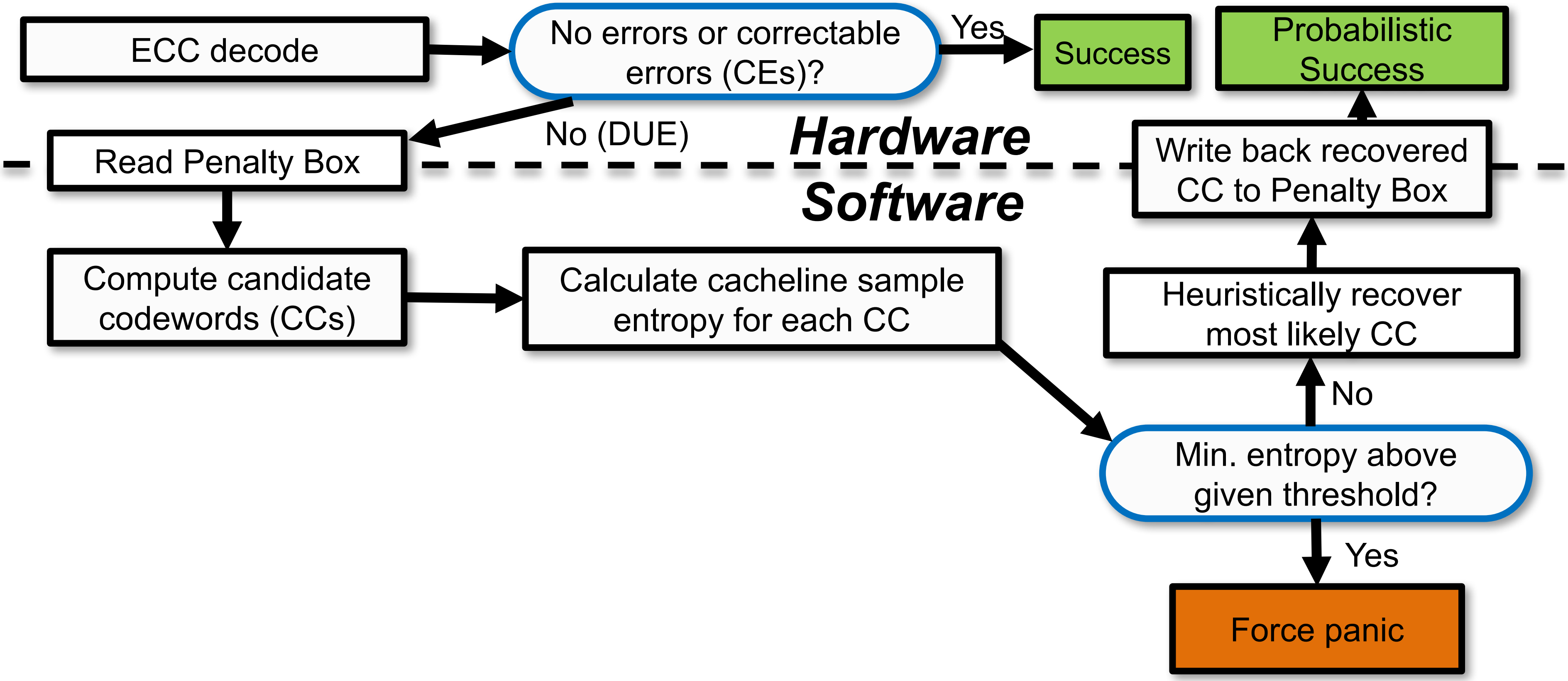
- **Existing DRAM systems already have most of the required support for SDECC**
  - ECC decoder
  - Error status registers
  - Error-reporting interrupts
- **We only need to expose the corrupted cacheline to system software!**
  - Extend functionality of existing error status registers and interrupt

No performance/energy overhead in common cases with no DUE!



Overall SDECC Approach

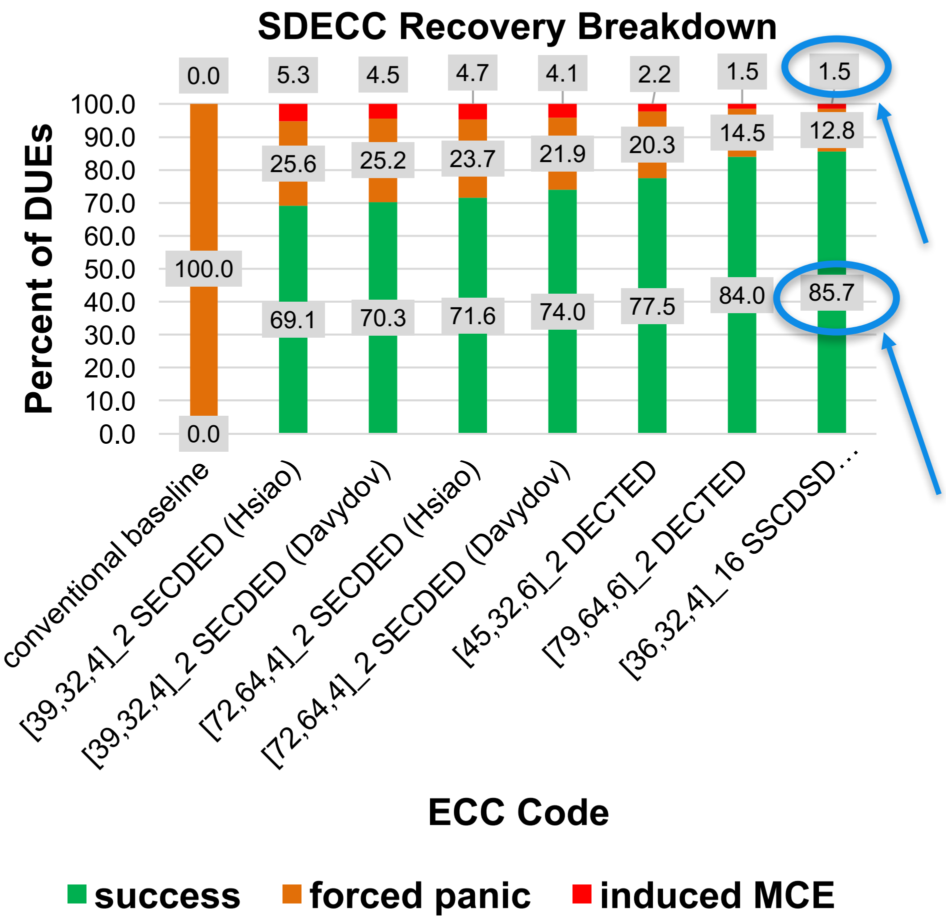
[Gottscho SELSE'16, Gottscho DSN-W'16, Gottscho '17]


ECC decode
No errors or correctable errors (CEs)?
Yes
Success
Probabilistic Success
No (DUE)
Hardware
Software
Read Penalty Box
Write back recovered CC to Penalty Box
Compute candidate codewords (CCs)
Calculate cacheline sample entropy for each CC
Heuristically recover most likely CC
No
Min. entropy above given threshold?
Yes
Force panic

Results: DUE Recovery Breakdown

[Gottscho '17]

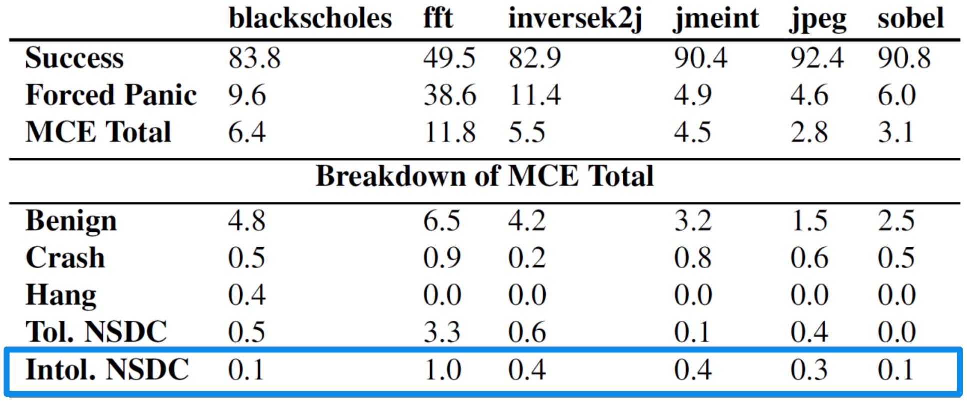
- Trace-based fault injection campaign
- 20 SPEC CPU2006 benchmarks
- RISC-V instruction set architecture



Results for Approximation-Tolerant Applications

[Gottscho '17]

[72,64,4]_2
Hsiao SECDED

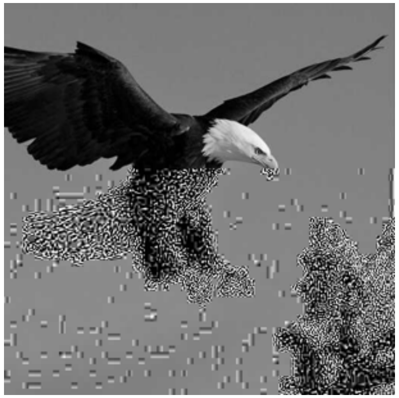


	blackscholes	fft	inversek2j	jmeint	jpeg	sobel
Success	83.8	49.5	82.9	90.4	92.4	90.8
Forced Panic	9.6	38.6	11.4	4.9	4.6	6.0
MCE Total	6.4	11.8	5.5	4.5	2.8	3.1
			Breakdown of MCE Total			
Benign	4.8	6.5	4.2	3.2	1.5	2.5
Crash	0.5	0.9	0.2	0.8	0.6	0.5
Hang	0.4	0.0	0.0	0.0	0.0	0.0
Tol. NSDC	0.5	3.3	0.6	0.1	0.4	0.0
Intol. NSDC	0.1	1.0	0.4	0.4	0.3	0.1

Original image (jpeg benchmark)

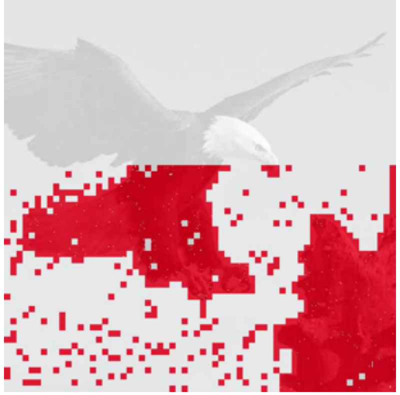
-

Worst-case corrupted image (out of 1000)



=

Pixel Delta



Pruning Candidates with Lightweight Hashes

[Gottscho '17]

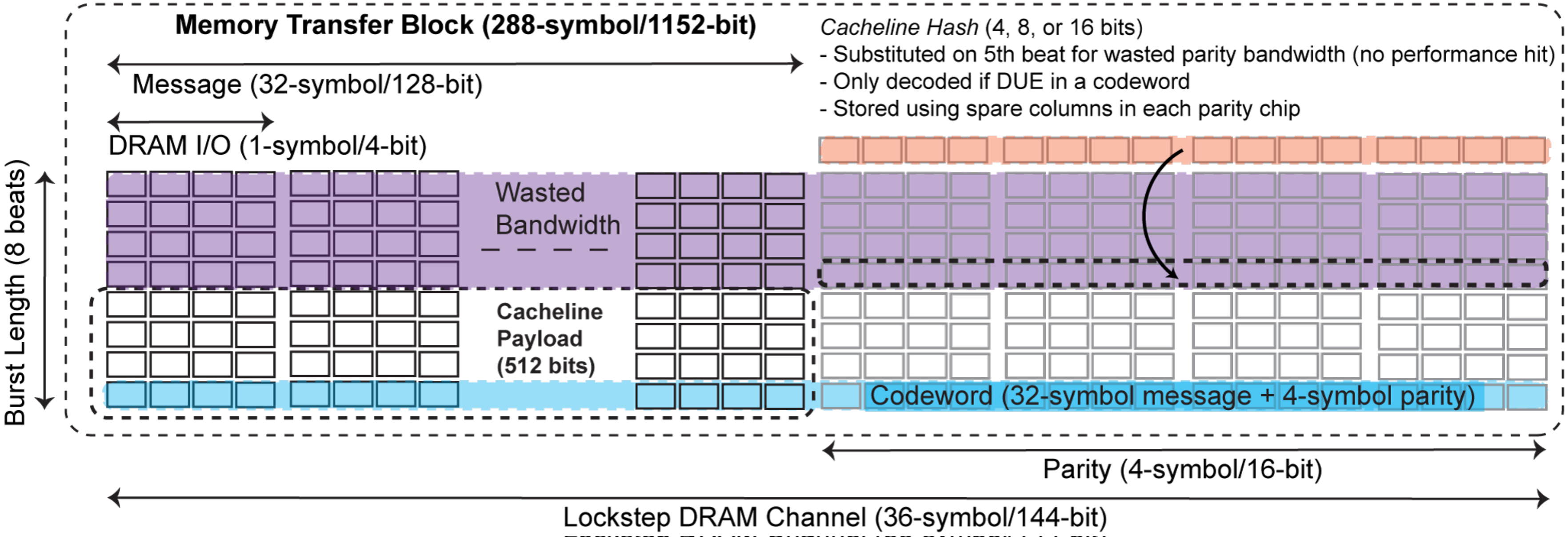
What if we could prune the list of candidate codewords to improve chance of recovery?

Solution: lightweight hashes

- Compute small (4, 8, or 16-bit) universal hash of original cacheline, store in memory
- If-and-only-if DUE occurs:
  - Read out original hash
  - Compare it against computed candidate hashes

Lightweight Hash Implementation: ChipKill

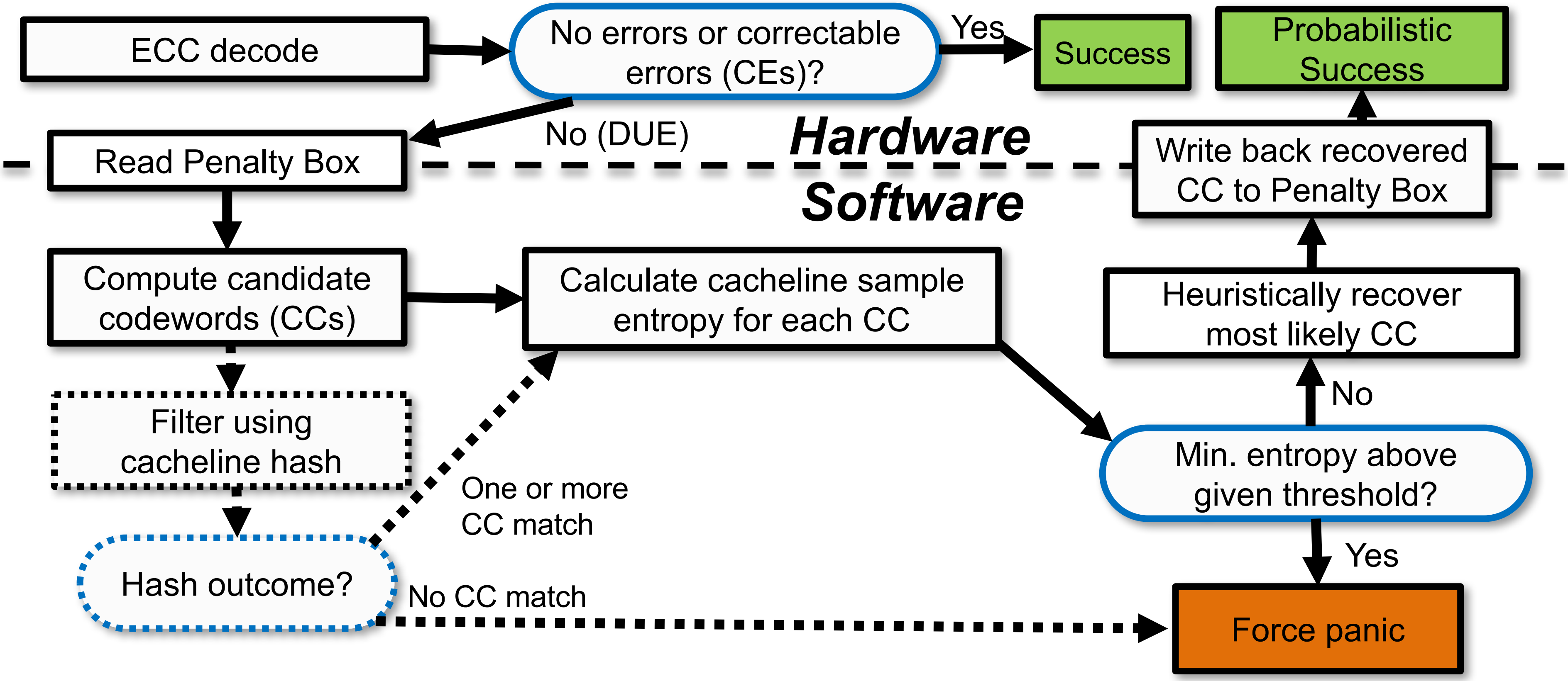
[Gottscho '17]


Memory Transfer Block (288-symbol/1152-bit)
Message (32-symbol/128-bit)
DRAM I/O (1-symbol/4-bit)
Burst Length (8 beats)
Wasted Bandwidth
Cacheline Payload (512 bits)
Codeword (32-symbol message + 4-symbol parity)
Parity (4-symbol/16-bit)
Lockstep DRAM Channel (36-symbol/144-bit)

Cacheline Hash (4, 8, or 16 bits)
- Substituted on 5th beat for wasted parity bandwidth (no performance hit)
- Only decoded if DUE in a codeword
- Stored using spare columns in each parity chip

Overall SDECC Approach ***With Hashes***

[Gottscho '17]


ECC decode
No errors or correctable errors (CEs)?
Yes
Success
Probabilistic Success
No (DUE)
Hardware
Software
Read Penalty Box
Write back recovered CC to Penalty Box
Compute candidate codewords (CCs)
Calculate cacheline sample entropy for each CC
Heuristically recover most likely CC
Filter using cacheline hash
No
Min. entropy above given threshold?
One or more CC match
Yes
Hash outcome?
No CC match
Force panic

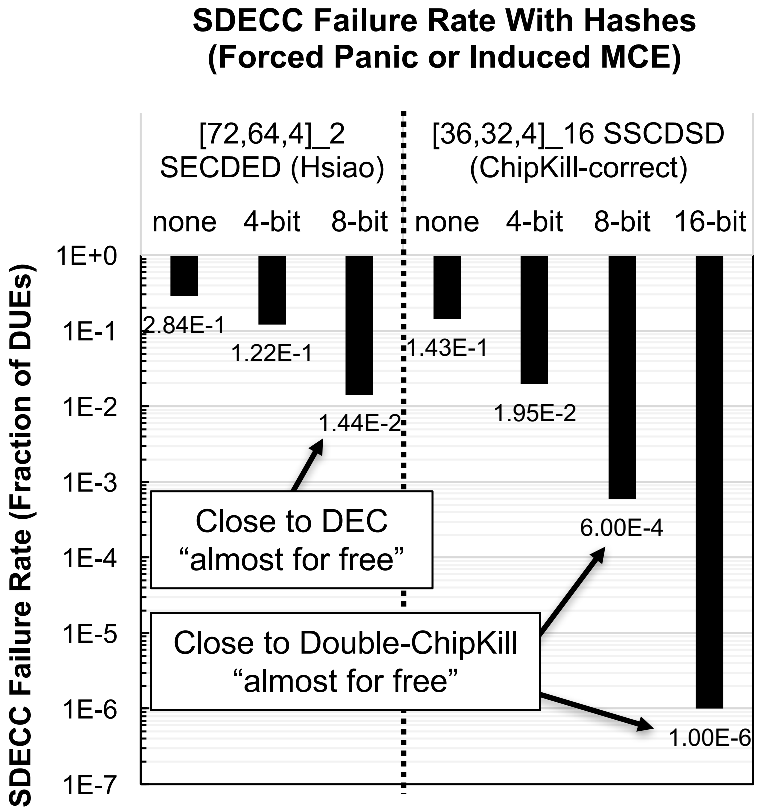
Results: DUE Recovery Breakdown ***with Hashes***

[Gottscho '17]

Lightweight hashes can improve SDECC recovery rates by orders of magnitude

Rates of Successful DUE Recovery

	baseline	none	4-bit	8-bit	16-bit
SECDED	5%	71.6%	87.8%	98.56%	N/A
ChipKill	39.9%	85.7%	98.05%	99.940%	99.9999%



SDECC Failure Rate With Hashes (Forced Panic or Induced MCE)

Close to DEC "almost for free"

Close to Double-ChipKill "almost for free"

Summary of SDECC

[Gottscho SELSE'16, Gottscho DSN-W'16, Gottscho '17]

- **Reliability Benefits**
  - Approximation-tolerant applications
    - Recover up to 92.4% of DUEs with [72,64,4]_2 SECDED
    - As low as 0.1% intolerable NSDC rate
  - Approximation-*intolerant* applications with 16-bit Lightweight Hash
    - Recover up to 99.9999% of DUEs with [36,32,4]_16 SSCDSD ChipKill-correct
    - MCE rate less than 0.2 ppm of DUEs

- **Applications to several domains**
  - *Supercomputing:* help reduce checkpoint frequency, saving time/energy
  - *Approximation-tolerant IoT devices:* support error correction at low cost
  - *Real-time embedded systems:* avoid missing deadlines when errors occur

Agenda

- Introduction
- Part 1: Exploiting Variability
  - ViPZonE
  - DPCS
  - X-Mem
- Part 2: Coping with Errors
  - Performability
  - SDECC
  - ViFFTo
- Conclusion and Directions for Future Work

ViFFTo: Virtualization-Free Fault Tolerance for Embedded Scratchpad Memories at Low Cost

Collaborators:

Irina Alam (UCLA)
Clayton Schoeny (UCLA)
Prof. Lara Dolecek (UCLA)
Prof. Puneet Gupta (UCLA)

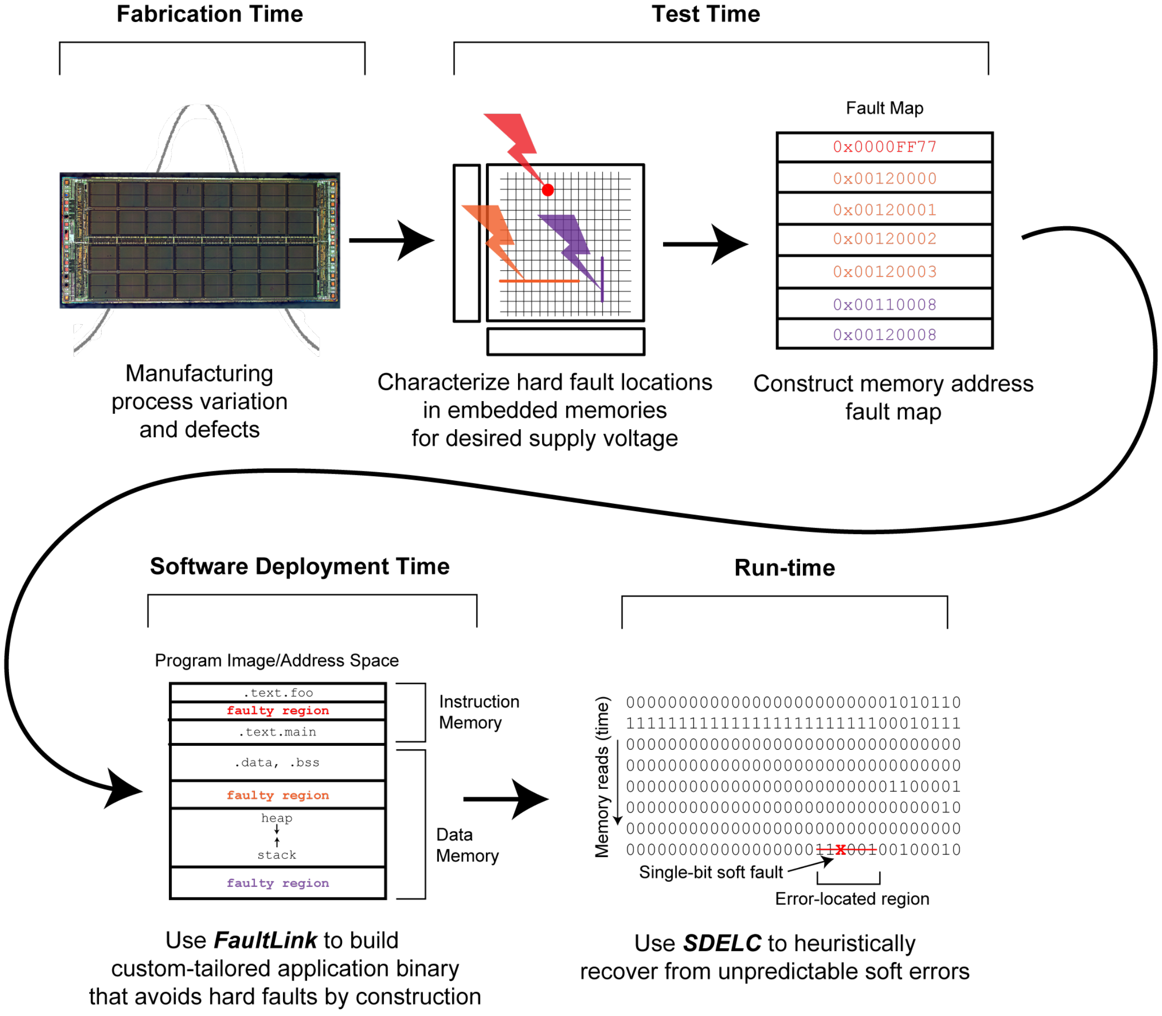
Publications:

Gottscho et al., 2017 manuscript submitted and under peer review

Chapter 7

ViFFTo Approach

[Gottscho '17]



Fabrication Time

Manufacturing
process variation
and defects

Test Time

Characterize hard fault locations
in embedded memories
for desired supply voltage

Fault Map

Fault Map
0x0000FF77
0x00120000
0x00120001
0x00120002
0x00120003
0x00110008
0x00120008

Construct memory address
fault map

Software Deployment Time

Program Image/Address Space

Program Image/Address Space
.text.foo
faulty region
.text.main
.data, .bss
faulty region
heap ↓ ↑ stack
faulty region

Use ***FaultLink*** to build
custom-tailored application binary
that avoids hard faults by construction

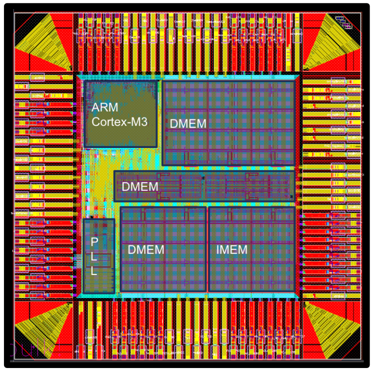
Run-time

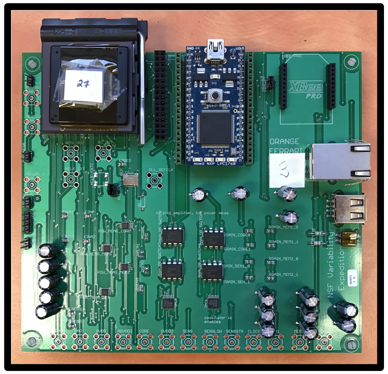
00000000000000000000000001010110
11111111111111111111111100010111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000001100001
00000000000000000000000000000010
00000000000000000000000000000000
000000000000000000~~11x001~~00100010

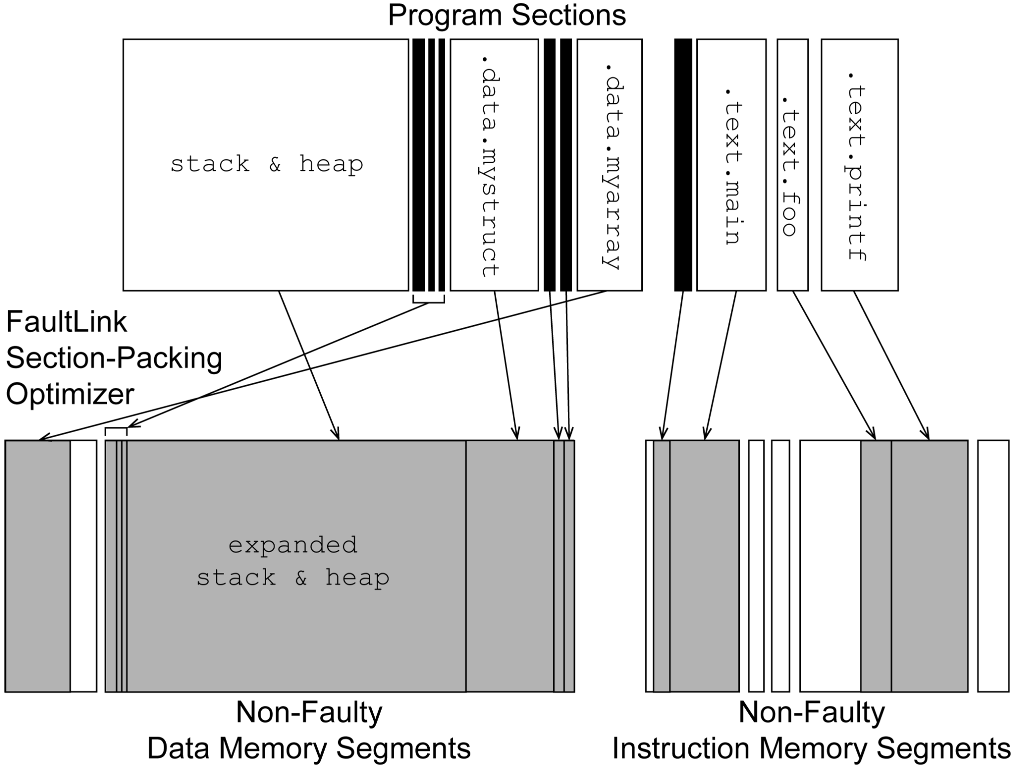
Use ***SDELC*** to heuristically
recover from unpredictable soft errors

FaultLink: Guarding Against Hard Faults at Link-Time

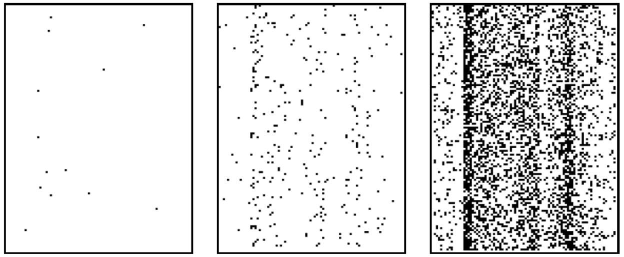
[Gottscho ‘17]







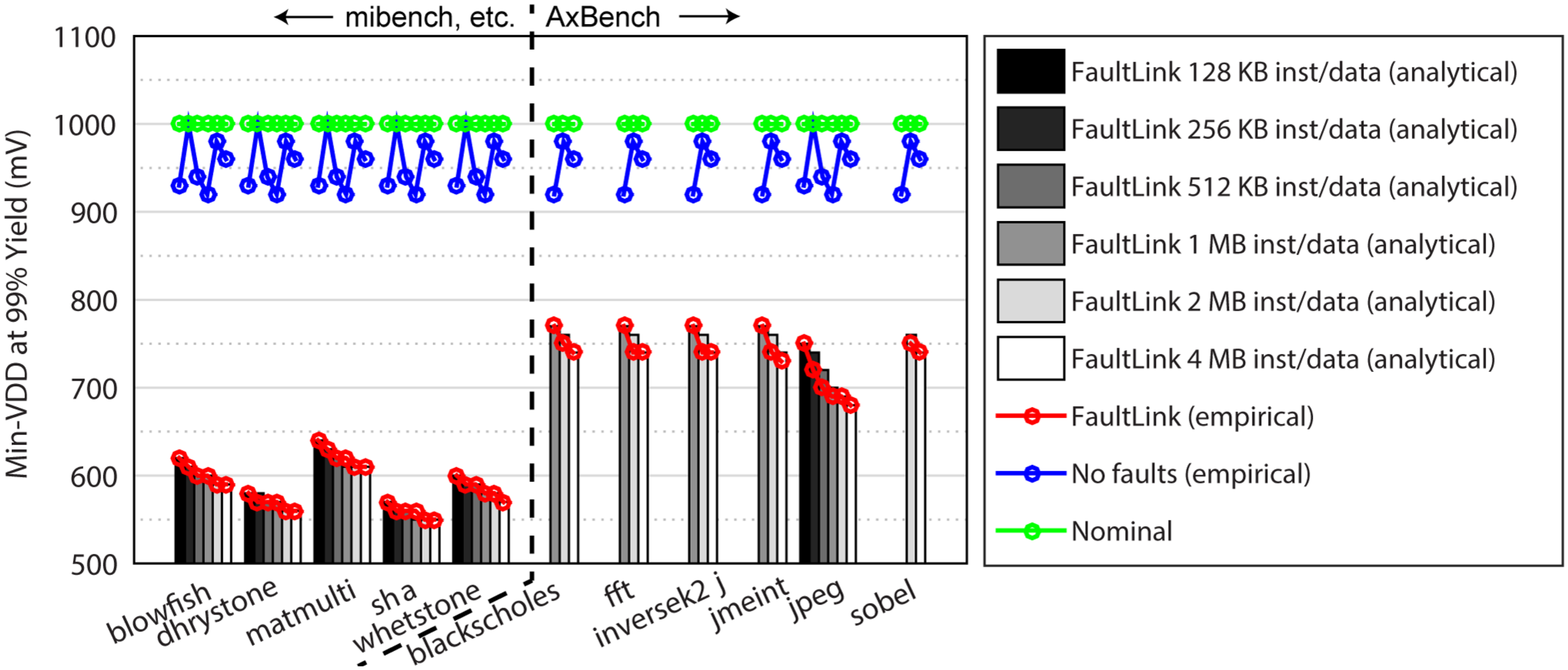
Test chip data SPM



750 mV   700 mV   650 mV

Results: Hard Faults

[Gottscho '17]



SDELC: Guarding Against Soft Faults at Run-Time

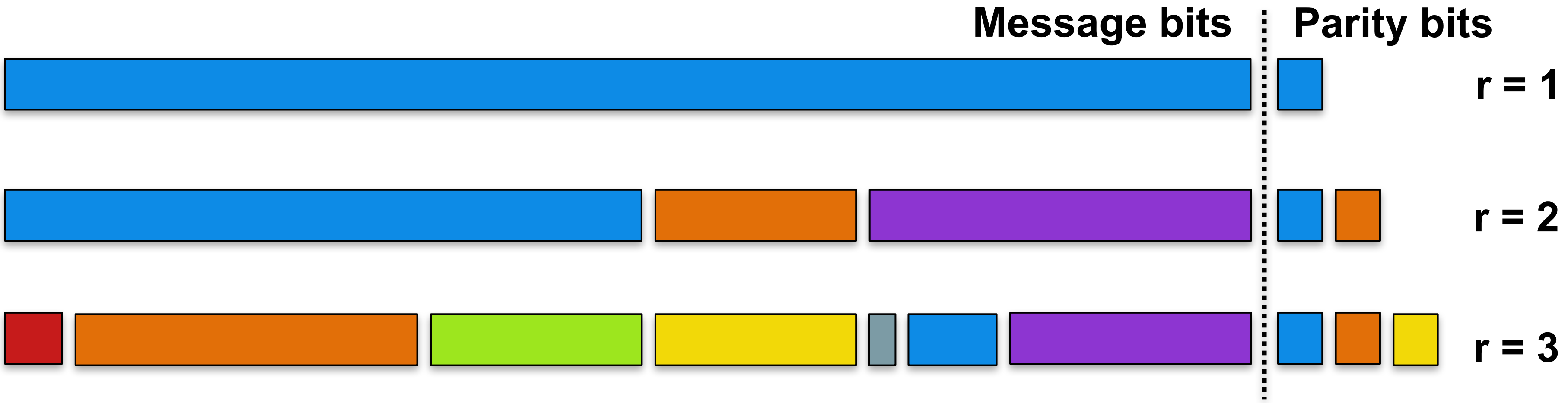
[Gottscho '17]

Software-Defined Error-Localizing Codes (SDELCs)

- Based on novel Ultra-Lightweight Error-Localizing Codes (UL-ELCs)
  - Between parity & Hamming code
  - Detect & localize 1-bit errors to specific chunk

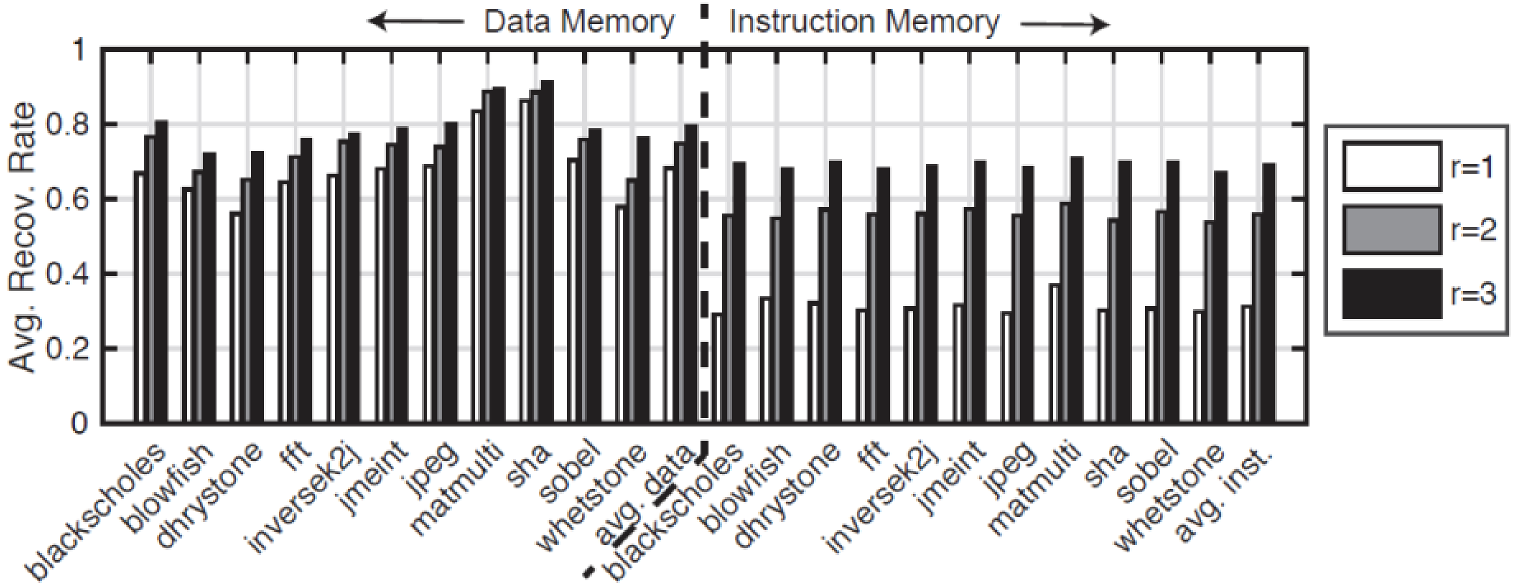
Software-Defined Recovery using Embedded C Library

- Application-driven data & instruction recovery policies



Results: Soft Faults

[Gottscho '17]



70% of single-bit errors can be recovered at less than half the cost of a standard Hamming code!

Summary of ViFFTo

[Gottscho ‘17]

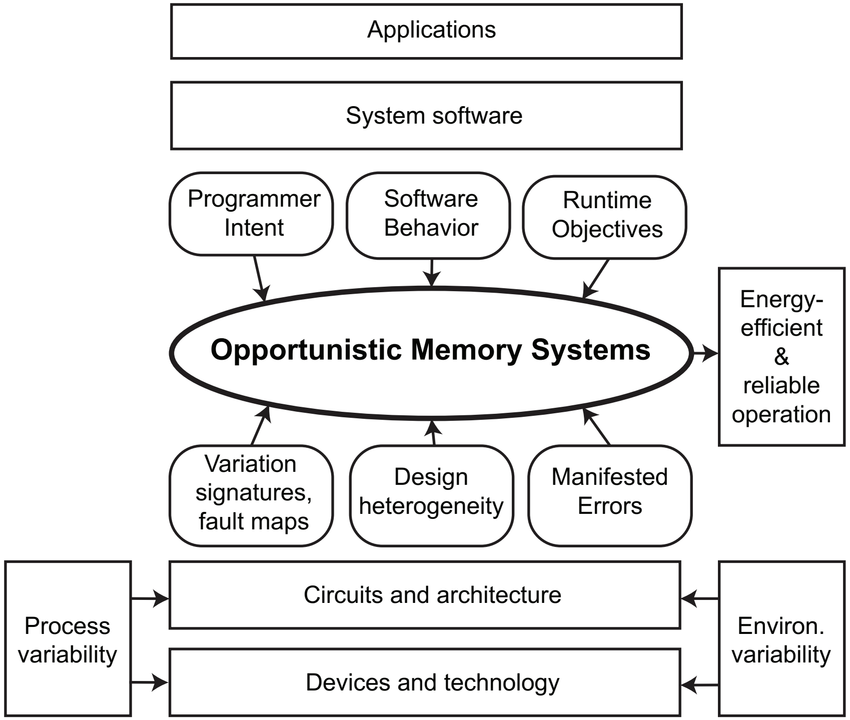
- ViFFTo opportunistically copes with memory errors in low-cost IoT devices
  - FaultLink can reduce VDD by up to 440 mV
  - SDELC can recover 70-90% of single-bit soft faults
- Minimal or no hardware overheads required
  - Improve yield (cost), energy, and reliability of IoT devices
  - Safest for approximation-tolerant applications

Agenda

- Introduction
- Part 1: Exploiting Variability
  - ViPZonE
  - DPCS
  - X-Mem
- Part 2: Coping with Errors
  - Performability
  - SDECC
  - ViFFTo
- Conclusion and Directions for Future Work

Summary of Dissertation

- Addressing energy efficiency and resiliency of memories is essential
- Opportunistic memory systems can help solve this problem!
- Part 1: ViPZonE, DPCS, X-Mem
  - Exploited hardware variability
- Part 2: Performability, SDECC, ViFFTo
  - Coped with memory errors



Open-source code available at https://github.com/nanocad-lab
Data available at http://nanocad.ee.ucla.edu/Main/DownloadForm

Directions for Future Work

- Short-term
  - Software-Defined ECC with fault models
  - Application-specific fault tolerance for hardware accelerators
  - Adapting techniques to emerging non-volatile memory devices

- Long-term
  - Joint abstractions for heterogeneity and variability
  - Checkerboard Architecture

- Vision
  - Demand for data + hardware specialization → Opportunistic Memory Systems

Acknowledgments

- Committee
  - Prof. Puneet Gupta (advisor)
  - Prof. Lara Dolecek
  - Prof. Mani Srivastava
  - Prof. Glenn Reinman
- UC Irvine
  - Prof. Nikil Dutt
  - Prof. Alexandru Nicolau
  - Dr. Luis A. D. Bathen
  - Dr. Abbas BanaiyanMofrad
- Microsoft
  - Dr. Mohammed Shoaib
  - Dr. Sriram Govindan
  - Dr. Bikash Sharma
  - Dr. Di Wang
  - Mike Andrewartha
  - Mark Santaniello
  - Dr. Jie Liu
  - Dr. Badriddine Khessib
  - Dr. Kushagra Vaid
- Qualcomm
  - Dr. Greg Wright
- UCLA doctoral students
  - Clayton Schoeny
  - Dr. Fred Sala
  - Salma Elmalaki
  - Dr. Lucas Wanner
- UCLA NanoCAD Lab
  - Irina Alam
  - Dr. Shaodi Wang
  - Dr. Liangzhen Lai
  - Yasmine Badr
  - Saptadeep Pal
  - Dr. Abde Ali Kagalwalla
  - Weiche Wang
  - Dr. Rani Ghaida
  - Dr. John Lee
- UCLA department staff
  - Deeona Columbia
  - Sandra Bryant
  - Mandy Smith
  - Ryo Arreola
- Funding
  - Qualcomm Innovation Fellowship
  - UCLA Dissertation Year Fellowship
  - US National Science Foundation Variability Expedition Grant No. CCF-1029783
  - UCLA Electrical Engineering Department PhD Fellowship

Publications

Mark Gottscho, Irina Alam, Clayton Schoeny, Lara Dolecek, and Puneet Gupta. "Low-Cost Memory Fault Tolerance for IoT Devices," 10 pages. Manuscript submitted and under review, April 2017.

Mark Gottscho, Clayton Schoeny, Lara Dolecek, and Puneet Gupta. "Software-Defined ECC: Recovery from Detected-but-Uncorrectable Memory Errors," 14 pages. Manuscript submitted and under review, April 2017.

Mark Gottscho, Mohammed Shoaib, Sriram Govindan, Bikash Sharma, Di Wang, and Puneet Gupta. "Measuring the Impact of Memory Errors on Application Performance," in IEEE Computer Architecture Letters (CAL), 4 pages. Pre-print available online August 2016. DOI: 10.1109/LCA.2016.2599513

Mark Gottscho, Clayton Schoeny, Lara Dolecek, and Puneet Gupta. "Software-Defined Error- Correcting Codes," in Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 276-282, Best of SELSE Special Session. Toulouse, France, June 2016. ISBN: 978-1-5090-3688-2, DOI: 10.1109/DSN-W.2016.67

Mark Gottscho, Sriram Govindan, Bikash Sharma, Mohammed Shoaib, and Puneet Gupta. "X-Mem: A Cross-Platform and Extensible Memory Characterization Tool for the Cloud," in Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 263-273. Uppsala, Sweden, April 2016. ISBN: 978-1-5090-1953-3, DOI: 10.1109/ISPASS.2016.7482101

Mark Gottscho, Clayton Schoeny, Lara Dolecek, and Puneet Gupta. "Software-Defined Error- Correcting Codes," in Silicon Errors in Logic – System Effects (SELSE) workshop, 6 pages. Austin, Texas, USA, March 2016. Best Paper Award.

Qixiang Zhang, Liangzhen Lai, **Mark Gottscho**, and Puneet Gupta. "Multi-Story Power Distribution Networks for GPUs," in IEEE Design, Automation, and Test in Europe (DATE), pp. 451-456, Dresden, Germany, March 2016. ISBN: 978-3-9815-3707-9

Mark Gottscho, Abbas BanaiyanMofrad, Nikil Dutt, Alex Nicolau, and Puneet Gupta. "DPCS: Dynamic Power/Capacity Scaling for SRAM Caches in the Nanoscale Era," in ACM Transactions on Architecture and Code Optimization (TACO), Vol. 12, No. 3, Article 27, 26 pages. Pre-print available online August 2015, in print October 2015. EISSN: 1544-3973, DOI: 10.1145/2792982

Lucas Wanner, Liangzhen Lai, Abbas Rahimi, **Mark Gottscho**, Pietro Mercati, Chu-Hsiang Huang, Frederic Sala, Yuvraj Agarwal, Lara Dolecek, Nikil Dutt, Puneet Gupta, Rajesh Gupta, Ranjit Jhala, Rakesh Kumar, Sorin Lerner, Subhashish Mitra, Alexandru Nicolau, Tajana Simunic Rosing, Mani B. Srivastava, Steve Swanson, Dennis Sylvester, and Yuanyuan Zhou. "NSF Expedition on Variability-Aware Software: Recent Results and Contributions," in De Gruyter Information Technology (it), Vol. 57, No. 3, pp. 181-198. Invited paper. Pre-print available online June 2015. DOI: 10.1515/itit-2014-1085

Salma Elmalaki, **Mark Gottscho**, Puneet Gupta, and Mani Srivastava. "A Case for Battery Charging-Aware Power Management and Deferrable Task Scheduling," in USENIX Workshop on Power-Aware Computing and Systems (HotPower), 6 pages. Bloomfield, Colorado, USA, October 2014.

Mark Gottscho, Abbas BanaiyanMofrad, Nikil Dutt, Alex Nicolau, and Puneet Gupta. "Power / Capacity Scaling: Energy Savings With Simple Fault-Tolerant Caches," in Proceedings of the ACM/IEEE Design Automation Conference (DAC), 6 pages. San Francisco, California, USA, June 2014. ISBN: 978-1-4503-2730-5, DOI: 10.1145/2593069.2593184

Nikil Dutt, Puneet Gupta, Alex Nicolau, **Mark Gottscho**, and Majid Shoushtari. "Multi-Layer Memory Resiliency," in Proceedings of the ACM/IEEE Design Automation Conference (DAC), 6 pages. Invited paper. San Francisco, California, USA, June 2014. ISBN: 978-1-4503-2730-5, DOI: 10.1145/2593069.2596684

Mark Gottscho, Luis A. D. Bathen, Nikil Dutt, Alex Nicolau, and Puneet Gupta. "ViPZonE: Hardware Power Variability-Aware Virtual Memory Management for Energy Savings," in IEEE Transactions on Computers (TC), Vol. 64, No. 5, pp. 1483-1496. Pre-print available online June 2014, in print May 2015. ISSN: 0018-9340, DOI: 10.1109/TC.2014.2329675

Nikil Dutt, Puneet Gupta, Alex Nicolau, Luis A. D. Bathen, **Mark Gottscho**. "Variability-Aware Memory Management for Nanoscale Computing," in Proceedings of the ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 125-132. Invited paper. Yokohama, Japan, January 2013. ISBN: 978-1-4673-3029-9, ISSN: 2153-6961, DOI: 10.1109/ASPDAC. 2013.6509584

Luis A. D. Bathen, **Mark Gottscho**, Nikil Dutt, Alex Nicolau, and Puneet Gupta. "ViPZonE: OS-Level Memory Variability-Aware Physical Address Zoning for Energy Savings," in Proceedings of the ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 33-42. Tampere, Finland, October 2012. ISBN: 978-1-4503-1426-8, DOI: 10.1145/2380445.2380457

Mark Gottscho, Abde Ali Kagalwalla, and Puneet Gupta. "Power Variability in Contemporary DRAMs," in IEEE Embedded Systems Letters (ESL), Vol. 4, No. 2, pp. 37-40. Pre-print available April 2012, in print June 2012. ISSN: 1943-0663, DOI: 10.1109/LES.2012.2192414

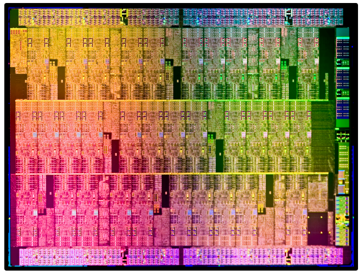
Questions?

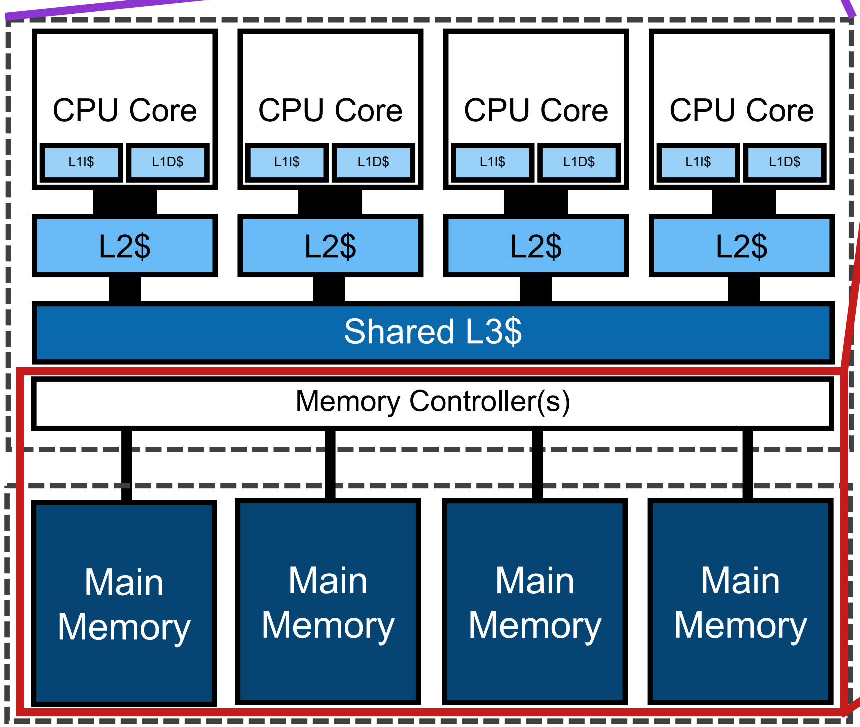
BONUS SLIDES

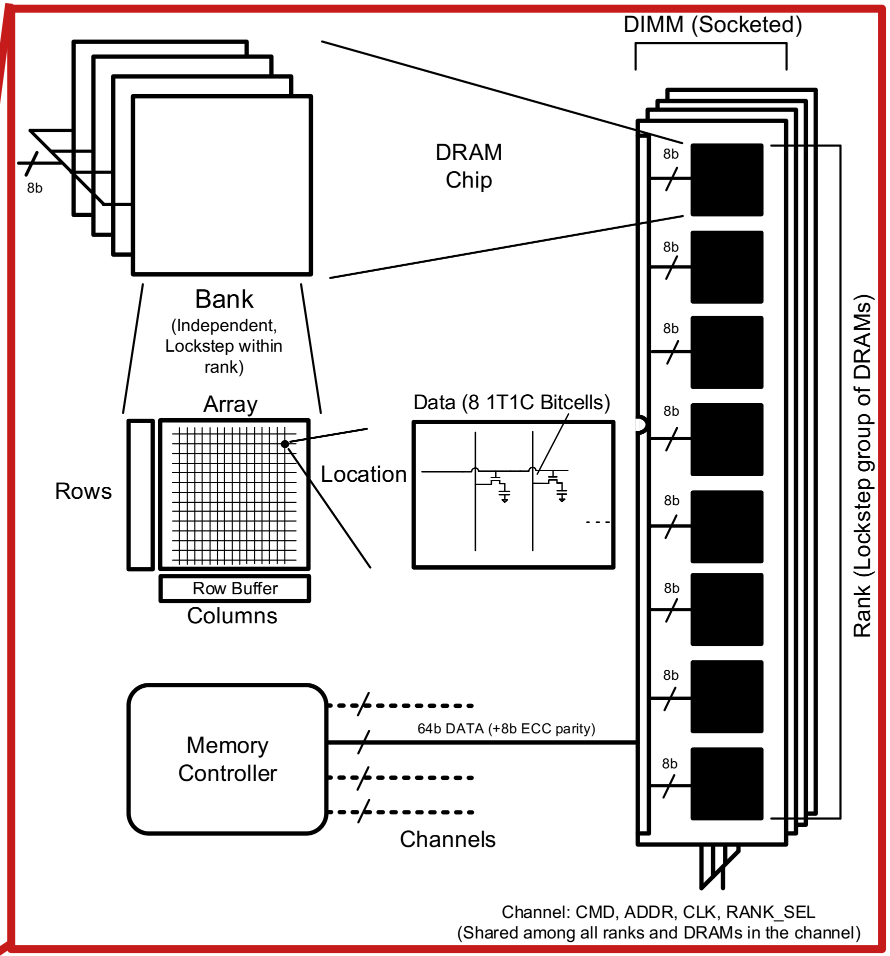
Bonus Slides

1. Introduction

Main Memory System

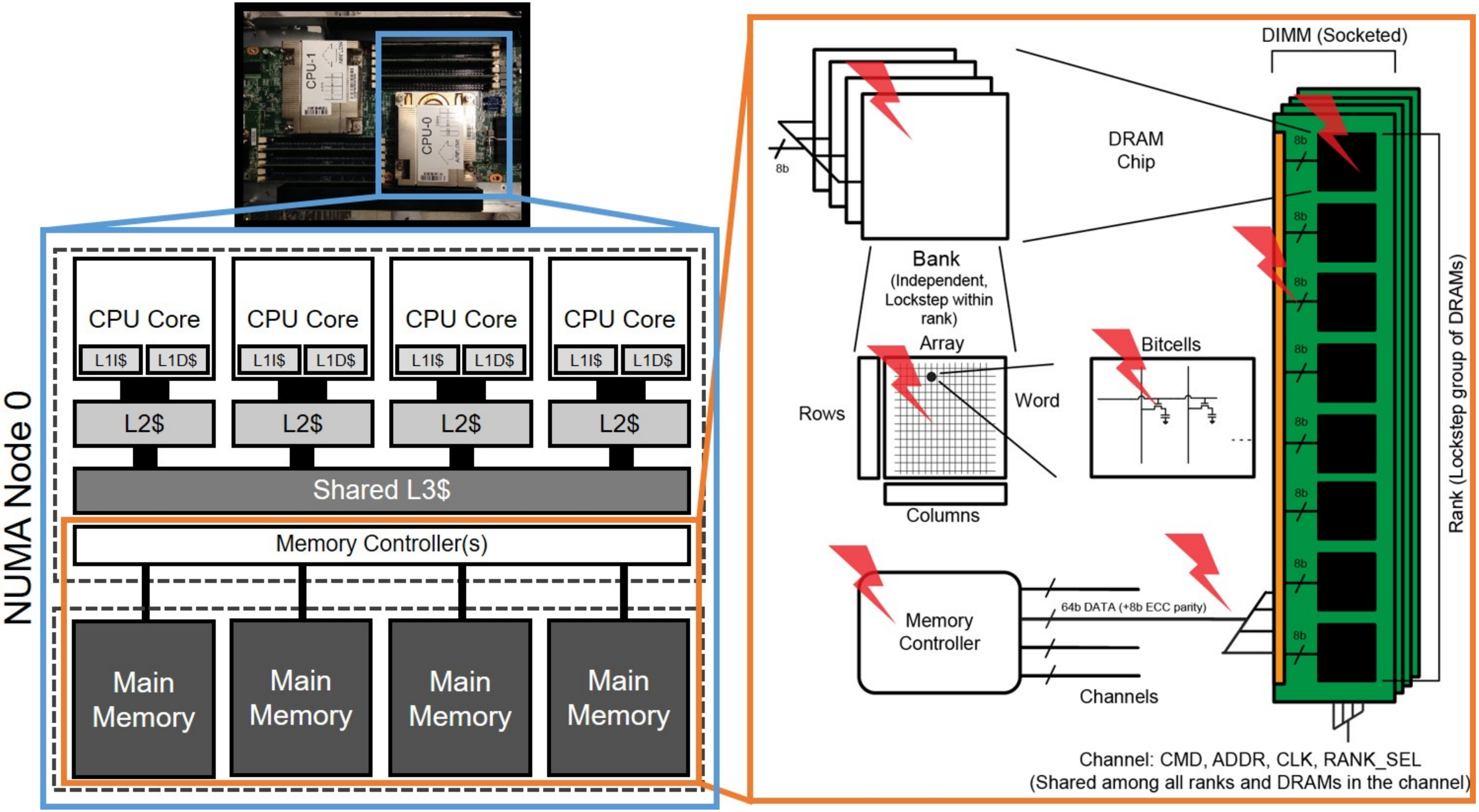





DIMM (Socketed)
8b
DRAM Chip
Bank
(Independent, Lockstep within rank)
Array
Rows
Row Buffer
Columns
Location
Data (8 1T1C Bitcells)
8b
8b
8b
8b
8b
8b
8b
8b
Rank (Lockstep group of DRAMs)
Memory Controller
64b DATA (+8b ECC parity)
Channels
Channel: CMD, ADDR, CLK, RANK_SEL
(Shared among all ranks and DRAMs in the channel)

Breakdown of Main Memory

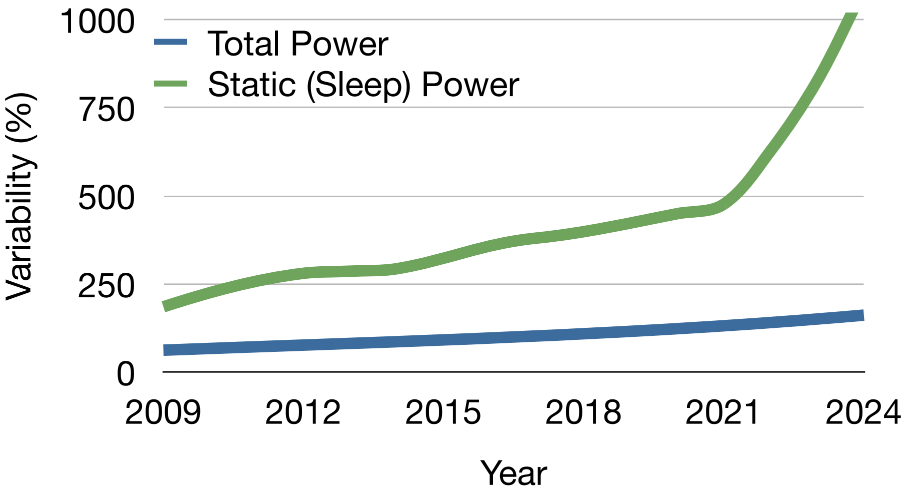
Faults in the Memory Hierarchy



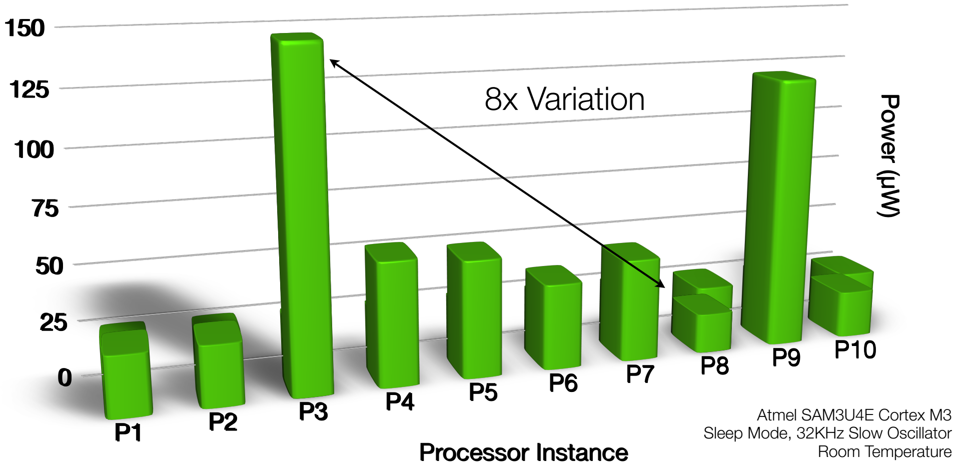
How Much Hardware Variability is There?

Individual figures courtesy of
Prof. Lucas Wanner's UCLA PhD Defense, 2014

ITRS Projection of Power Variability

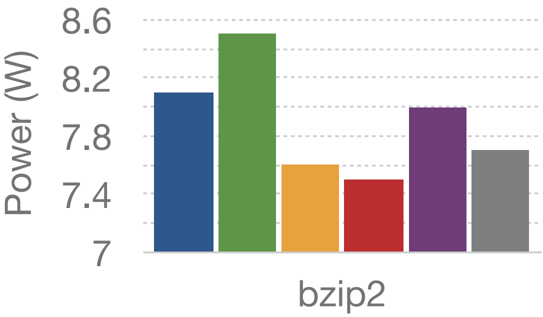


Measured Sleep Power in 10 Cortex M3 Processors

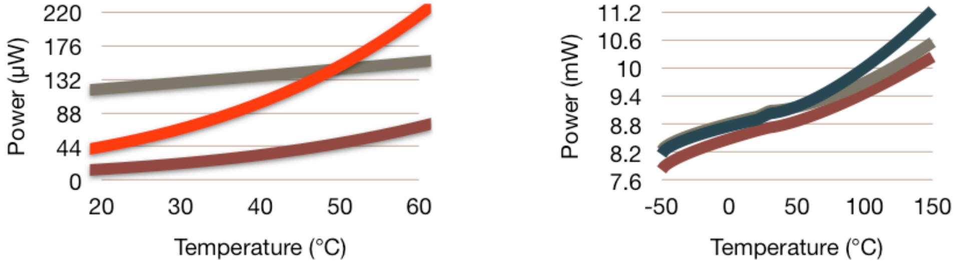


Atmel SAM3U4E Cortex M3
Sleep Mode, 32KHz Slow Oscillator
Room Temperature

Measured Power Consumption of
5 Intel Core i5 CPUs [Balaji et al. HotPower'12]



Measured Power Variations
With Respect to Temperature



SAM3U Sleep: 14x

SAM3U Active: 44%

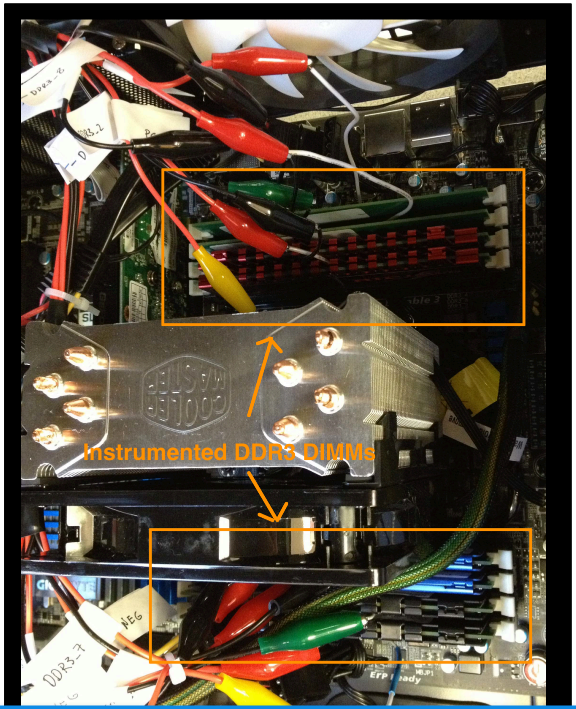
Bonus Slides

2. ViPZonE

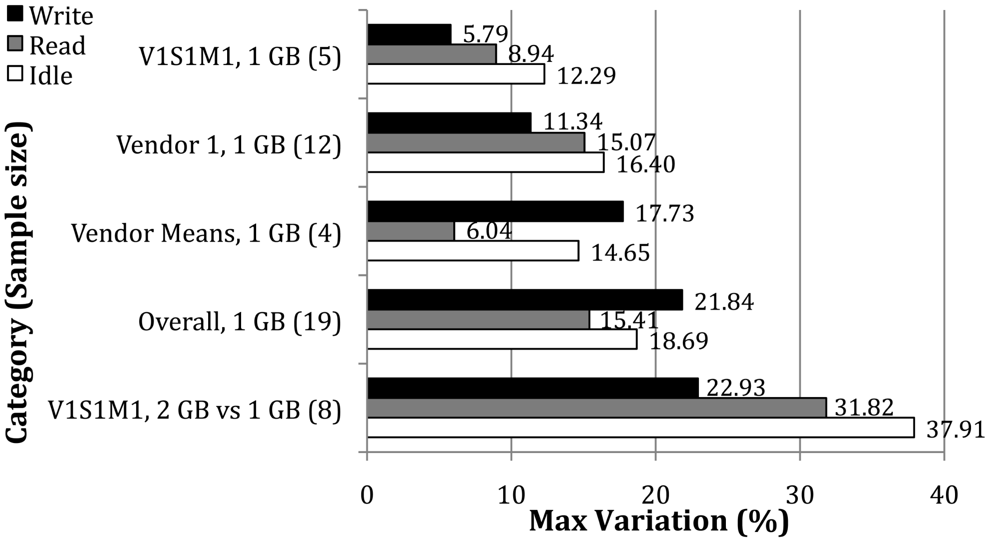
Motivation: Power Variability in Contemporary DRAMs

[Gottscho et al. ESL'12]

Significant power variations measured in off-the-shelf DDR3 memory



Instrumented DDR3 DIMMs



Systems could save energy by exploiting active memory power variability!

Related Work

[Bathen et al. CODES+ISSS'12, Gottscho et al. TC'15]

- **Power-aware memory systems**
  - Page allocation [Lebeck et al. ASPLOS'00]
  - Scheduler-based [Delaluz et al. DAC'02]
  - Page miss rates [Zhou et al. ASPLOS'04]
  - Adaptive architecture [Zheng et al. MICRO'08]
  - Independent DRAMs [Ahn et al. CAL'08]
- **Variation-aware circuits and systems**
  - Task scheduling [Wang et al. ICCAD'07]
  - Speed binning multicore processors [Sartori et al. ISQED'10]
  - Embedded sensing [Wanner et al. HotPower'10, DATE'11]
  - Quality adaptation [Pant et al. GLSVLSI'10]

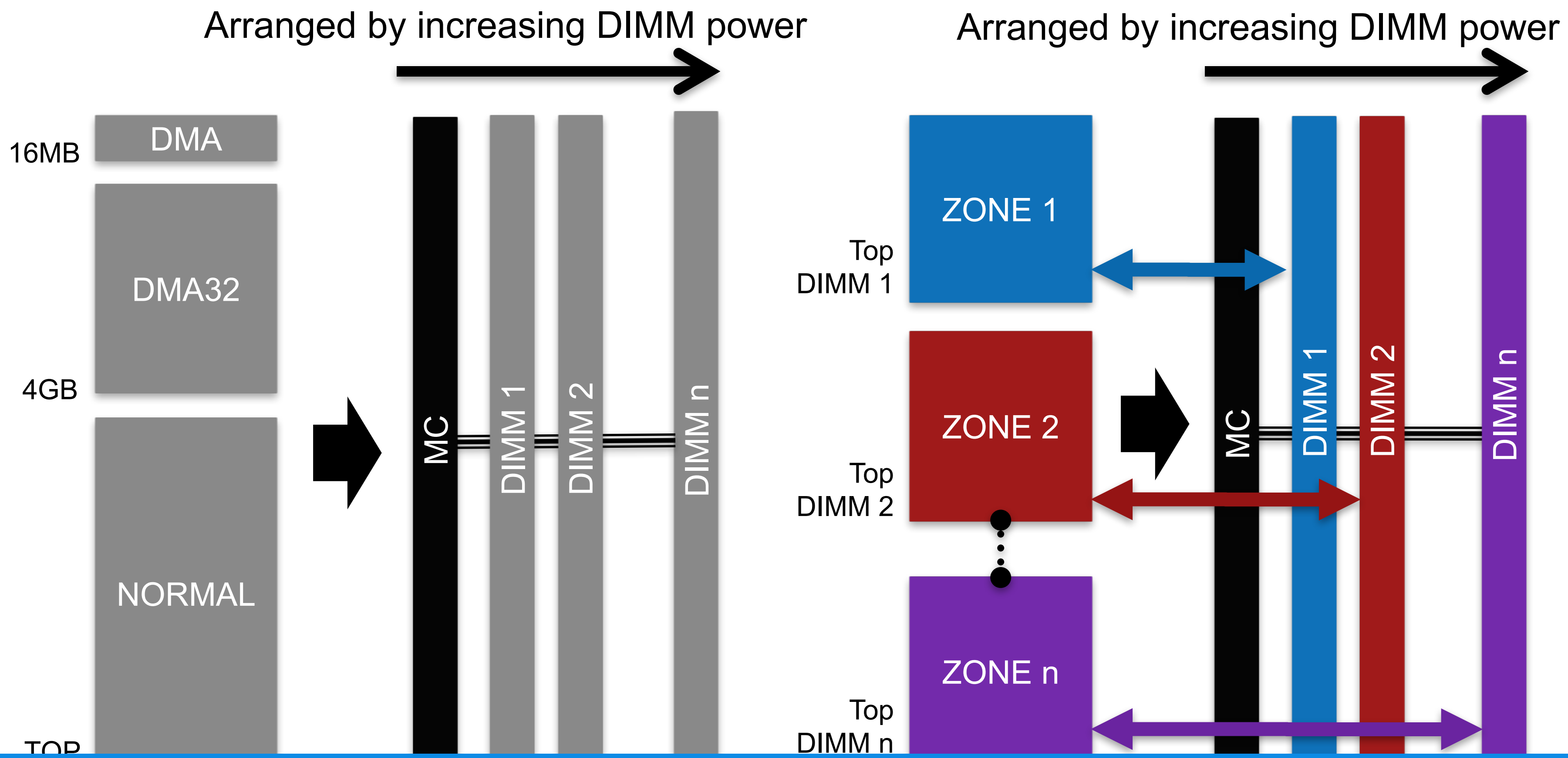
No prior work on SW-based variation-aware memory management except VaMV [Bathen et al. DATE'12]

3

Implementation: Lower OS Layer

Physical address zoning

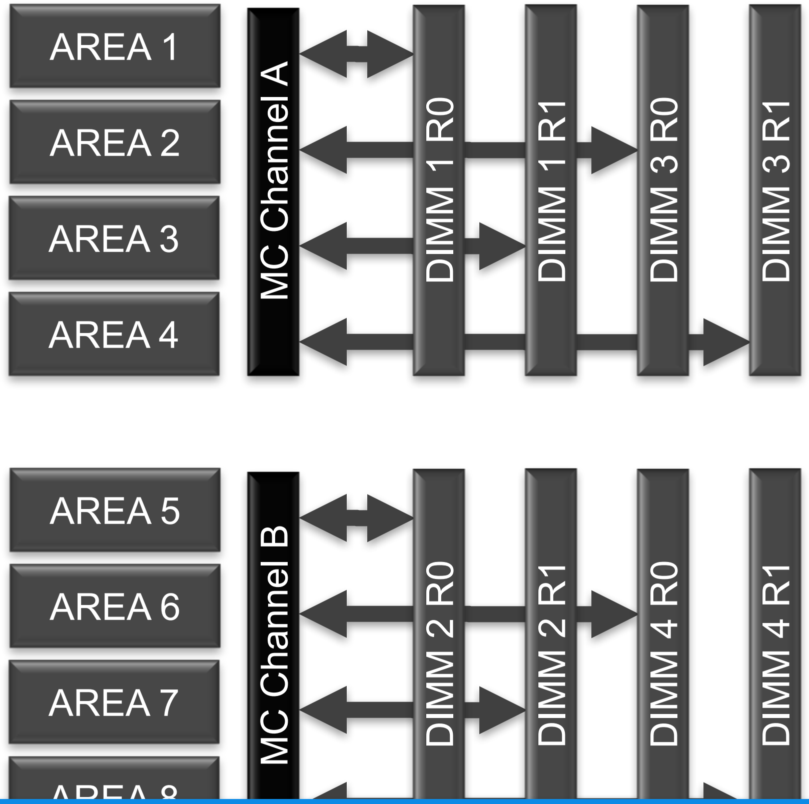
[Bathen et al. CODES+ISSS’12, Gottscho et al. TC’15]



“ViPZonEs” have different power characteristics because they are directly mapped to DIMMs exhibiting variation

DRAM Channel and Rank Interleaving

[Gottscho et al. TC'15]

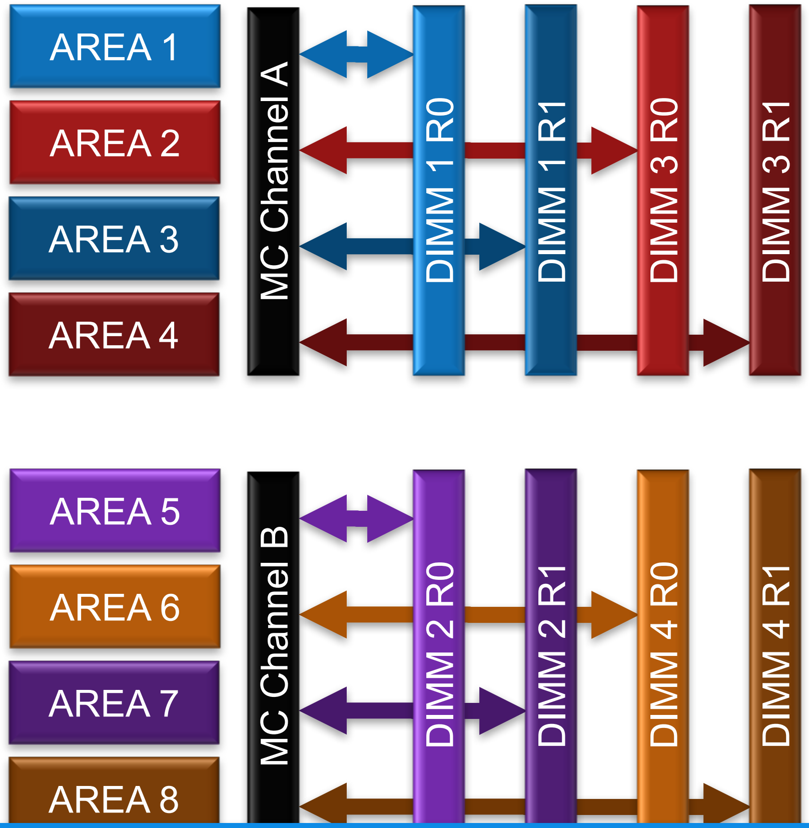


- Assume DDR3 with:
  - 2 channels
  - 2 DIMMs per channel
  - 2 ranks per DIMM
  - All rank capacities equal
- Assume data mapping:
  - Data striped channels, DIMMs, and ranks @ cache line granularity
  - Stripe size < page size, e.g. 64B vs 4KB

Conventional interleaving is good for memory-level parallelism for within-page access patterns

Interleaving Disabled

[Gottscho et al. TC'15]



- No striping of adjacent cache lines
- Single-page access = single-rank access
- Non-accessed ranks can enter low power states more often
- BUT: reduced memory-level parallelism for access to adjacent cache lines

Disabling interleaving allows ViPZonE to work but could impact baseline performance

1 Implementation: Application Layer

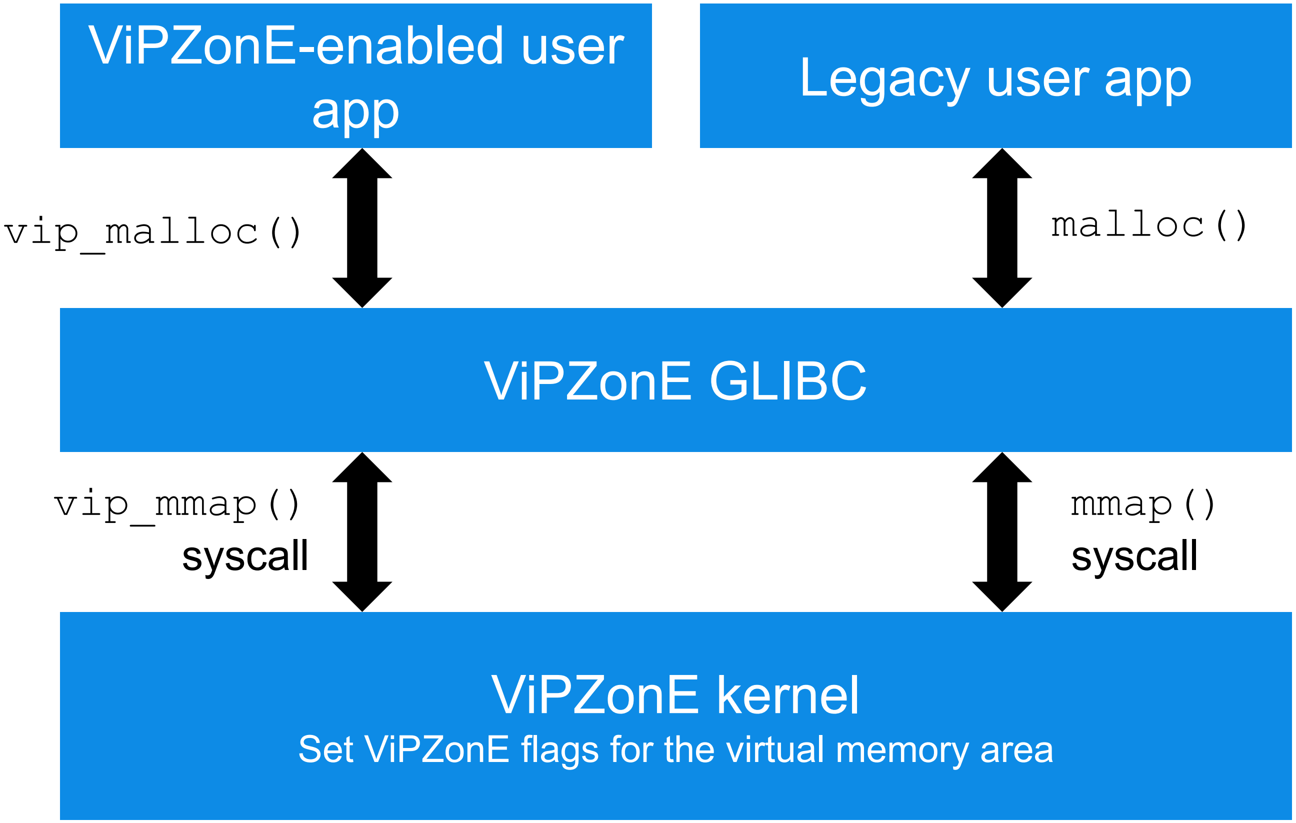
[Bathen et al. CODES+ISSS'12, Gottscho et al. TC'15]

```
#include <stdlib.h> //Special ViPZonE GLIBC with ViPZonE Linux kernel

//…some code…

void foo(size_t arraySize) {
    int *data_ptr = NULL;

    /* Possible vip_malloc() flags:
     * One of: VIP_WRITE or VIP_READ
     * One of: VIP_HIGH_UTIL or VIP_LOW_UTIL
     * Programmer is responsible to decide
     */
    data_ptr = (int *) vip_malloc(sizeof(int)*arraySize,
        VIP_WRITE | VIP_HIGH_UTIL);

    //…some write-heavy operations…
```

`vip_malloc()` abstracts memory power variability in a user-friendly way

2 Implementation: Upper OS Layer

[Bathen et al. CODES+ISSS'12, Gottscho et al. TC'15]



ViPZonE-enabled user app

Legacy user app

vip_malloc()

malloc()

ViPZonE GLIBC

vip_mmap() syscall

mmap() syscall

ViPZonE kernel

Set ViPZonE flags for the virtual memory area

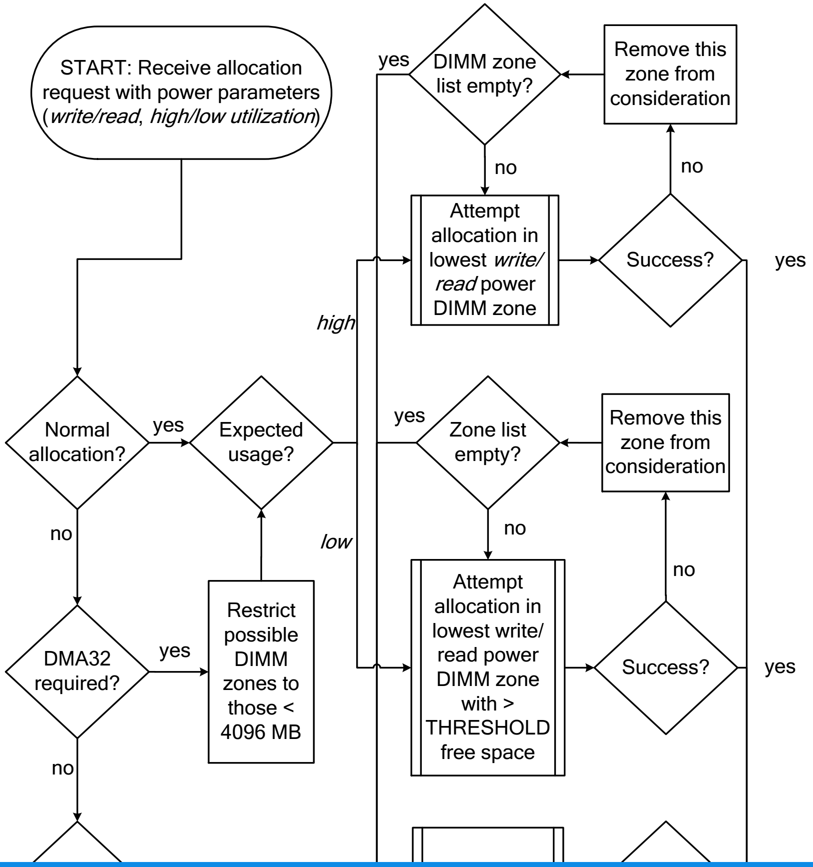
New apps can exploit ViPZonE, legacy apps work the same

3

Implementation: Lower OS Layer

Physical page allocator

[Bathen et al. CODES+ISSS'12, Gottscho et al. TC'15]


START: Receive allocation request with power parameters (*write/read*, *high/low utilization*)
Normal allocation?
yes
no
Expected usage?
high
low
DMA32 required?
yes
no
Restrict possible DIMM zones to those < 4096 MB
DIMM zone list empty?
yes
no
Attempt allocation in lowest *write/ read* power DIMM zone
Success?
yes
no
Remove this zone from consideration
Zone list empty?
yes
no
Attempt allocation in lowest write/ read power DIMM zone with > THRESHOLD free space
Success?
yes
no
Remove this zone from consideration

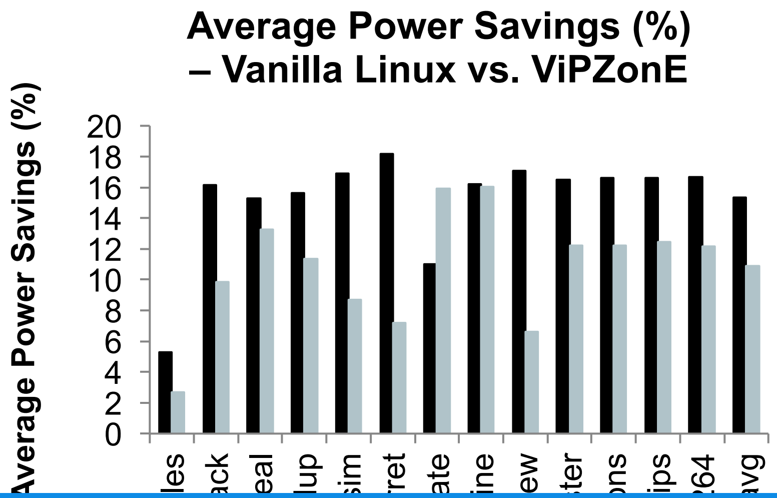
Simplicity → Fast kernel ☺

Simulation Results: Promising Power Savings

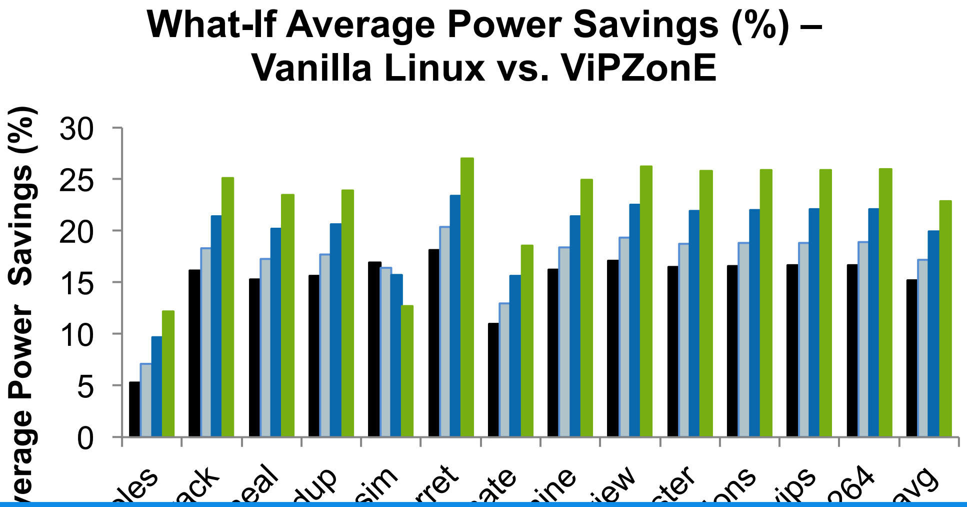
[Bathen et al. CODES+ISSS'12]

- Simulations show that memory power savings could be up to ~20%
  - Using the 1GB DIMM variability data shown earlier

- Memory power savings could increase to ~30% if future DIMM variability increases to 100%
- Performance overhead was expected to be modest



Average Power Savings (%) – Vanilla Linux vs. ViPZonE



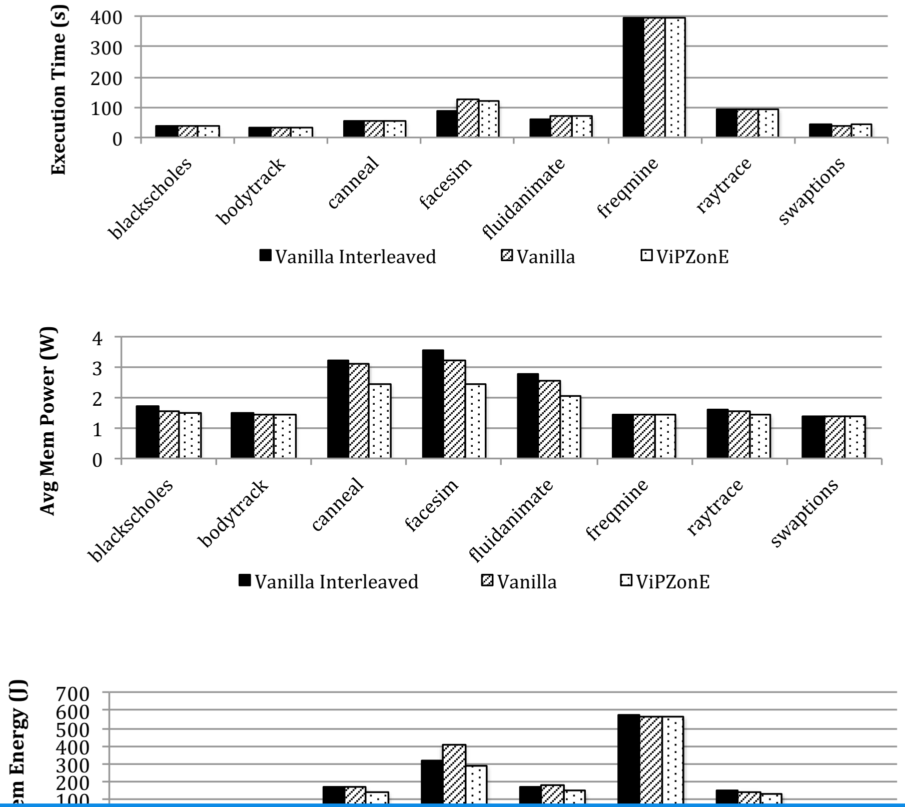
What-If Average Power Savings (%) – Vanilla Linux vs. ViPZonE

Detailed simulations indicate promising power savings

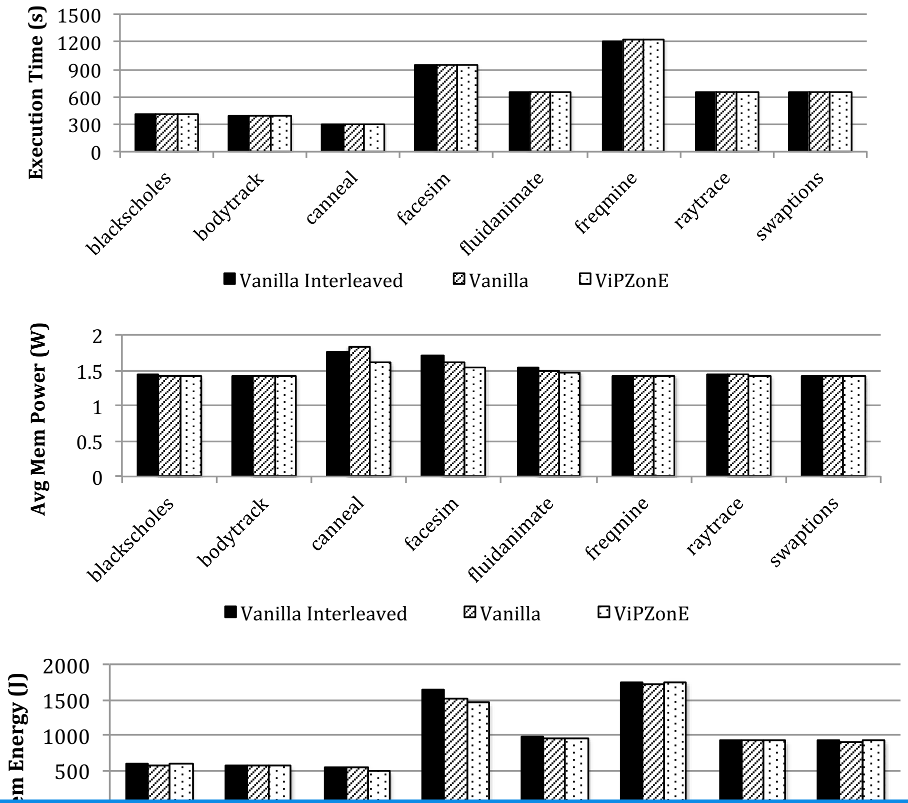
Measured Testbed Results

[Gottscho et al. TC’15]

Fast2 Hardware Config



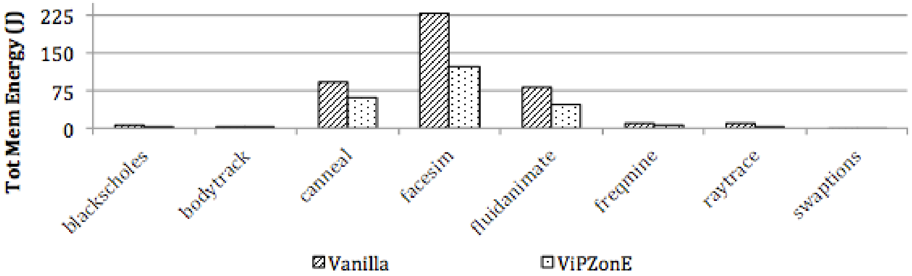
Slow2 Hardware Config



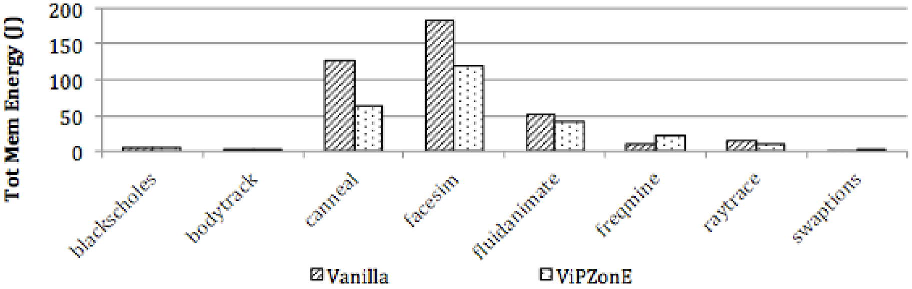
Good energy savings for non-bandwidth-intensive applications

Hypothetical Benefits for NVMs

[Gottscho et al. TC'15]



(a) *Fast2* Memory Energy, idle energy removed



(b) *Slow2* Memory Energy, idle energy removed

Idle power is the limiting factor for ViPZonE on current hardware

Summary

[Bathen et al. CODES+ISSS'12, Gottscho et al. TC'15]

Use when high memory-level parallelism or bandwidth not needed

Physical zoning inherently trades off benefits of striping for resource consolidation and exploitation of device variations

Opportunistically save energy in today's systems with no hardware changes

Through smart management of physical memory variation signatures

Bonus Slides

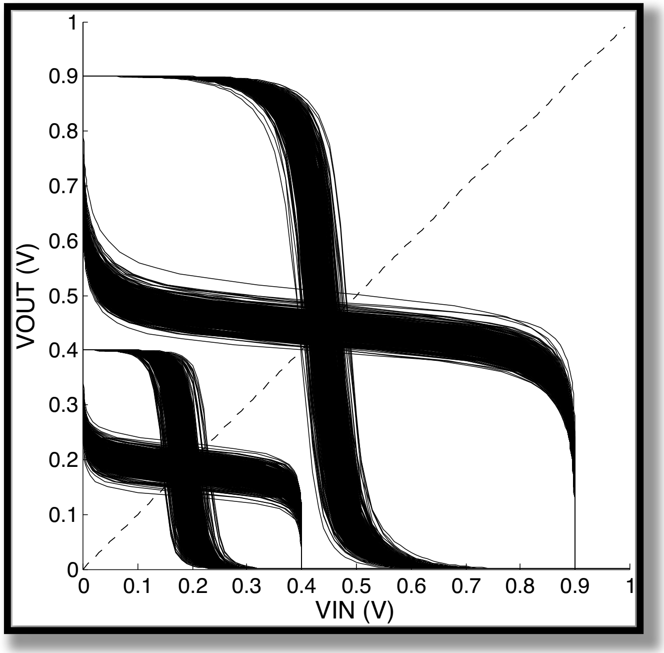
3. DPCS

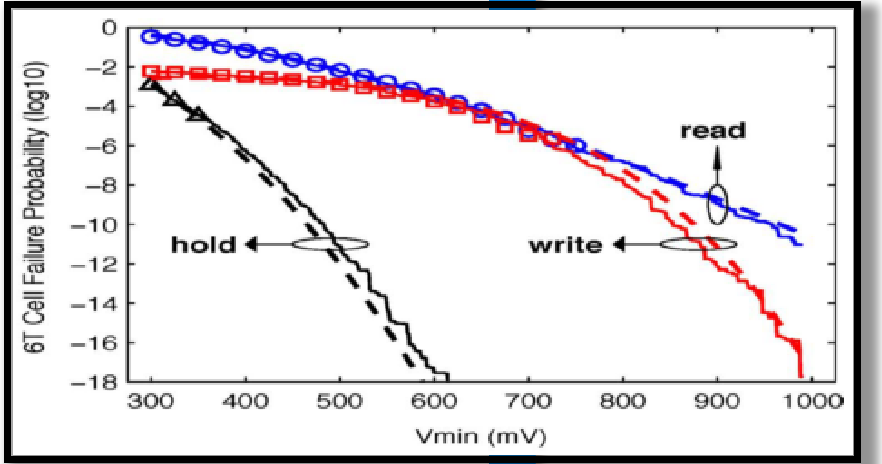
Motivation: Increasing Process Variability Limits SRAM Voltage Scaling

[Gottscho et al. DAC'14]

Process variability ↑ → V_T variations ↑

SRAM σ_{SNM} ↑





[Wang and Calhoun TVLSI'11]

Limited min-VDD/yield, leakage-dominated caches, increasing portion of overall power

Related Work

- Rich body of work for fault-tolerant voltage-scalable (FTVS) cache memories in nanoscale era
  - Leakage reduction (famously: [Powell et al. ISLPED'00, Flautner et al. ISCA'02])
  - Fault tolerant circuits/architecture/ECC [Shirvani & McCluskey VLSI Test '99, Agarwal et al. TVLSI'05, Ansari et al. MICRO'09, Alameldeen et al. TC'11, etc.]
  - Memory power/performance scaling [Fan et al. '05, Deng et al. ASPLOS'11, David et al. ICAC'11, Deng et al. MICRO'12] for

DPCS is the first FTVS scheme that efficiently leverages multiple voltage levels and power gating of disabled blocks, and supplements DVFS for logic

Question

[Gottscho et al. DAC'14, TACO'15]

How to optimize SRAM for the "best" *system-level* tradeoffs in energy, reliability, performance, & area?

There are many possible fault-tolerant cache design schemes that can be used!

Amdahl's Law Re-Formulated

[Gottscho et al. DAC’14, TACO’15]

$$PowerReduction_{FTVS,\ overall} = \frac{1}{1 - Fraction_{VS} + \boxed{Fraction_{FT\ overhead}} + \frac{Fraction_{VS}}{PowerReduction_{VS}}} \quad (1)$$

<u>Save energy via</u>

<u>simple & low-overhead</u>

fault-tolerant, voltage-scalable (FTVS)

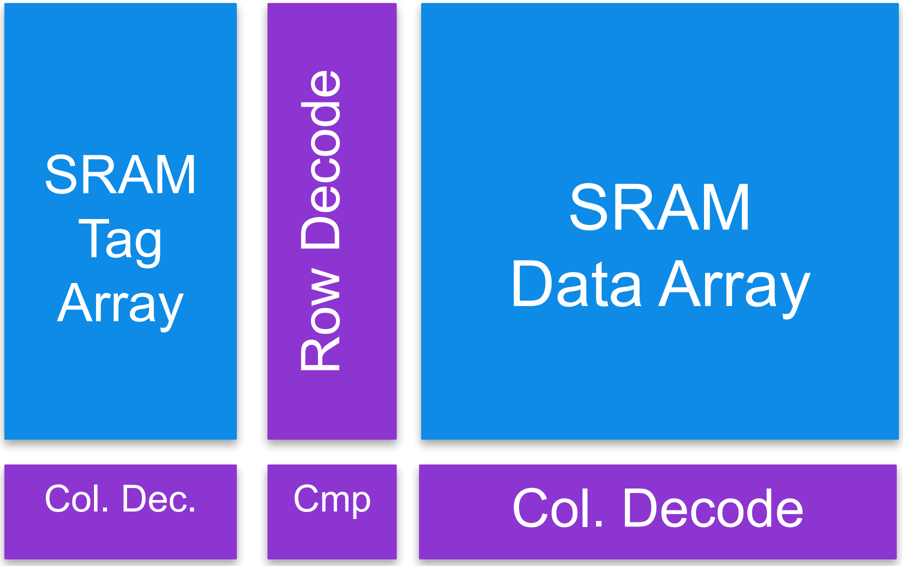
SRAM cache architecture

Using Fault Tolerance to Achieve Lower min-VDD

[Gottscho et al. DAC'14]

Many fault-tolerant, voltage-scalable (FTVS) approaches lower min-VDD using sophisticated fault tolerance methods

Baseline Cache @ Nominal VDD – No Fault Tolerance



PURPLE = periphery @ full VDD

BLUE = SRAM cells (bright is higher VDD)

Using Fault Tolerance to Achieve Lower min-VDD

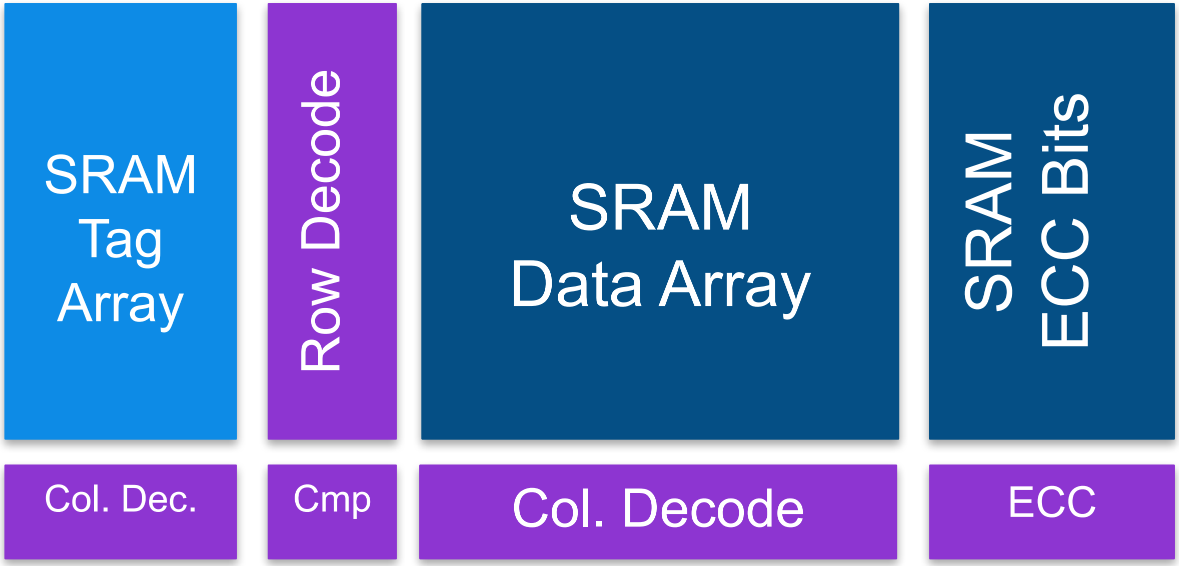
[Gottscho et al. DAC'14]

Many fault-tolerant, voltage-scalable (FTVS) approaches lower min-VDD using sophisticated fault tolerance methods

ECC Cache, Data Array @ 0.7 VDD

PURPLE = periphery @ full VDD

BLUE = SRAM cells (bright is higher VDD)



Using Fault Tolerance to Achieve Lower min-VDD

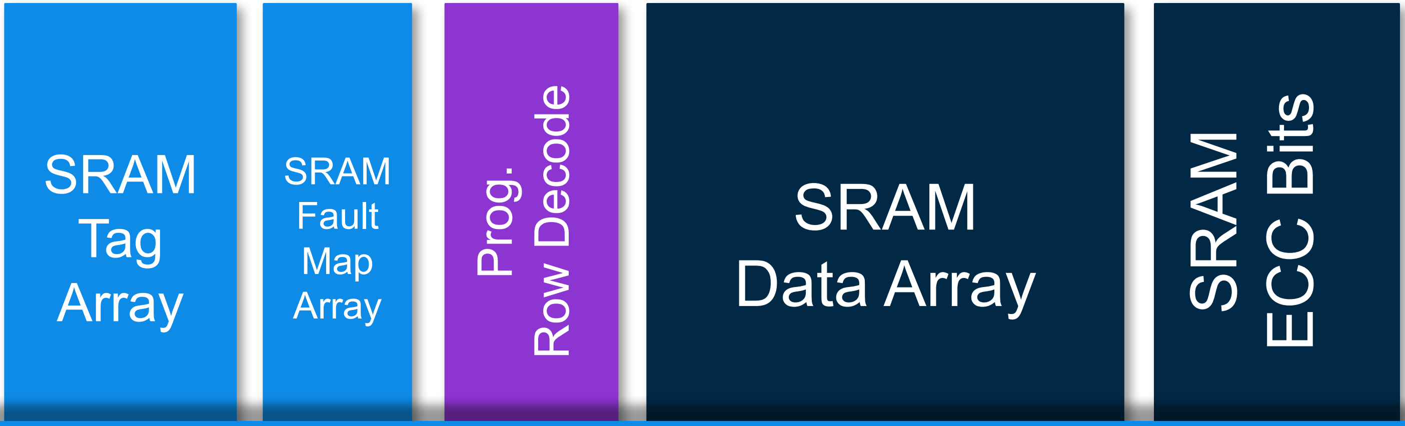
[Gottscho et al. DAC'14]

Many fault-tolerant, voltage-scalable (FTVS) approaches lower min-VDD using sophisticated fault tolerance methods

ECC + Faulty Set Remapping Cache, Data Array @ 0.5 VDD

PURPLE = periphery @ full VDD

BLUE = SRAM cells (bright is higher VDD)

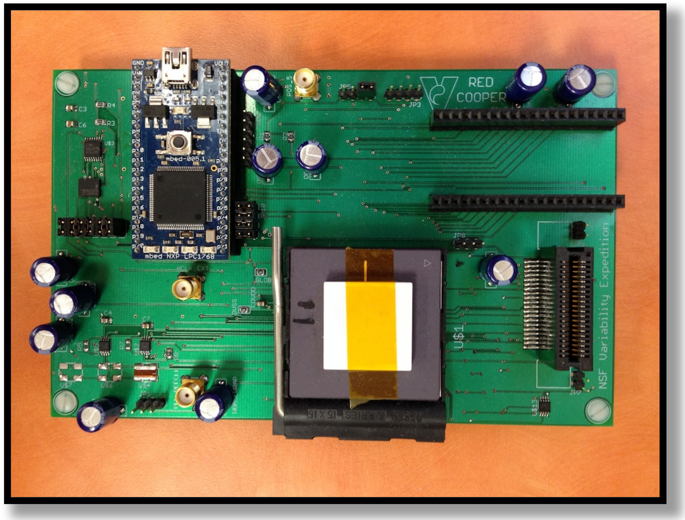


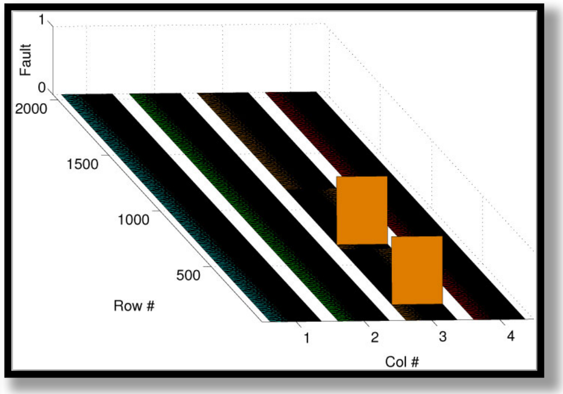
Min-VDD can be a misleading metric…

SRAM “Fault Inclusion Property”

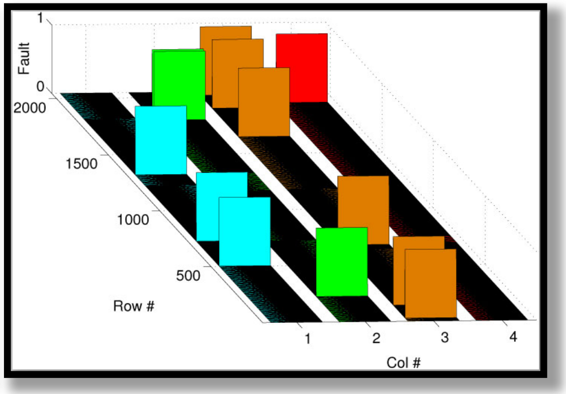
[Gottscho et al. DAC’14, TACO’15]

NSF Variability Expedition
“Red Cooper” test chips[1]
based on ARM Cortex M3

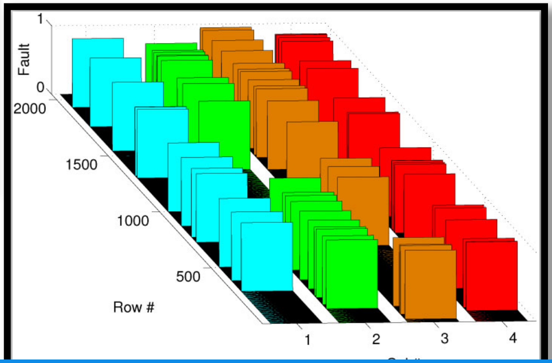


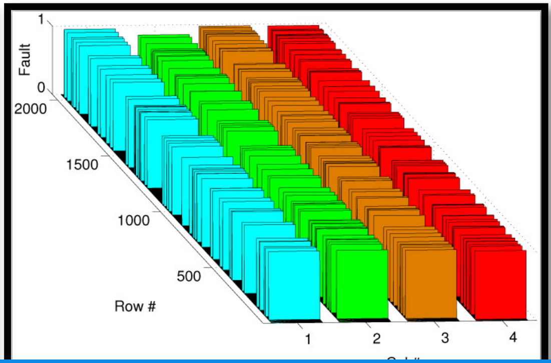


550 mV



525 mV

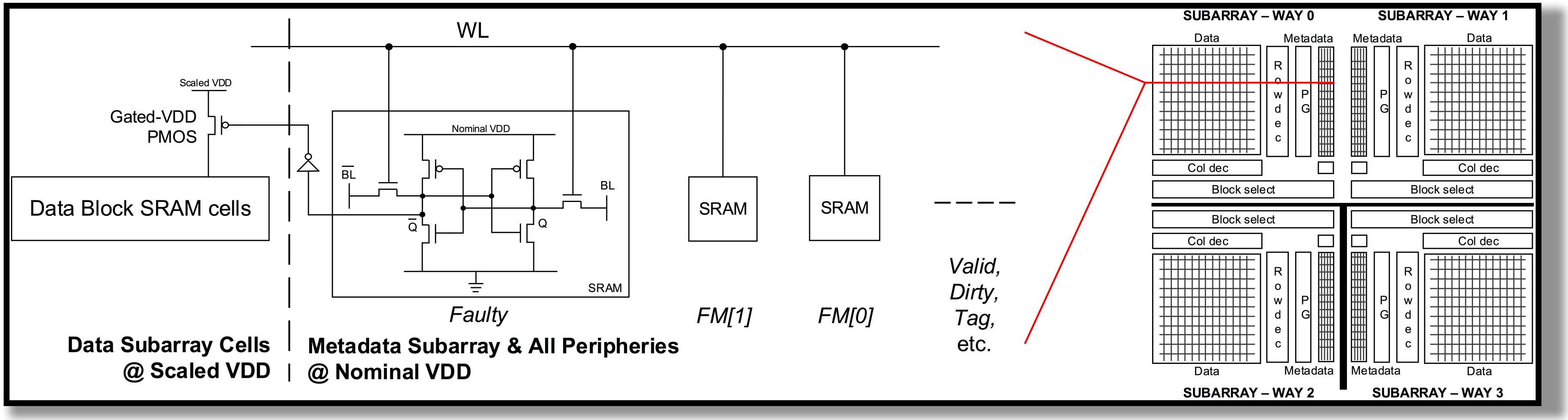




We can now efficiently store multi-VDD fault maps with low overhead... Trade off cache capacity and power dynamically!

Architectural Mechanism

[Gottscho et al. DAC'14, TACO'15]



- No redundancy – just sacrifice faulty blocks as VDD scales
  - # good blocks fall off a "cliff" anyway
    - Redundancy can only do so much
  - Negligible area overhead

Simplicity is key to low overheads

Power/Capacity Scaling

[Gottscho et al. DAC'14, TACO'15]

- To adjust data array VDD
  - Temporarily stall accesses
  - Cache controller finds the blocks that will become faulty at next VDD using *FM* bits
    - Flush those blocks that are also *Valid* & *Dirty*
    - Then set *Faulty* bits, power gating them
  - Adjust VDD, wait for voltage to settle
  - Resume operations
- Two general types of runtime policies
  - Static (SPCS)

Power gate cache blocks that are disabled for extra power savings

Static & Dynamic Power/Capacity Scaling Policies

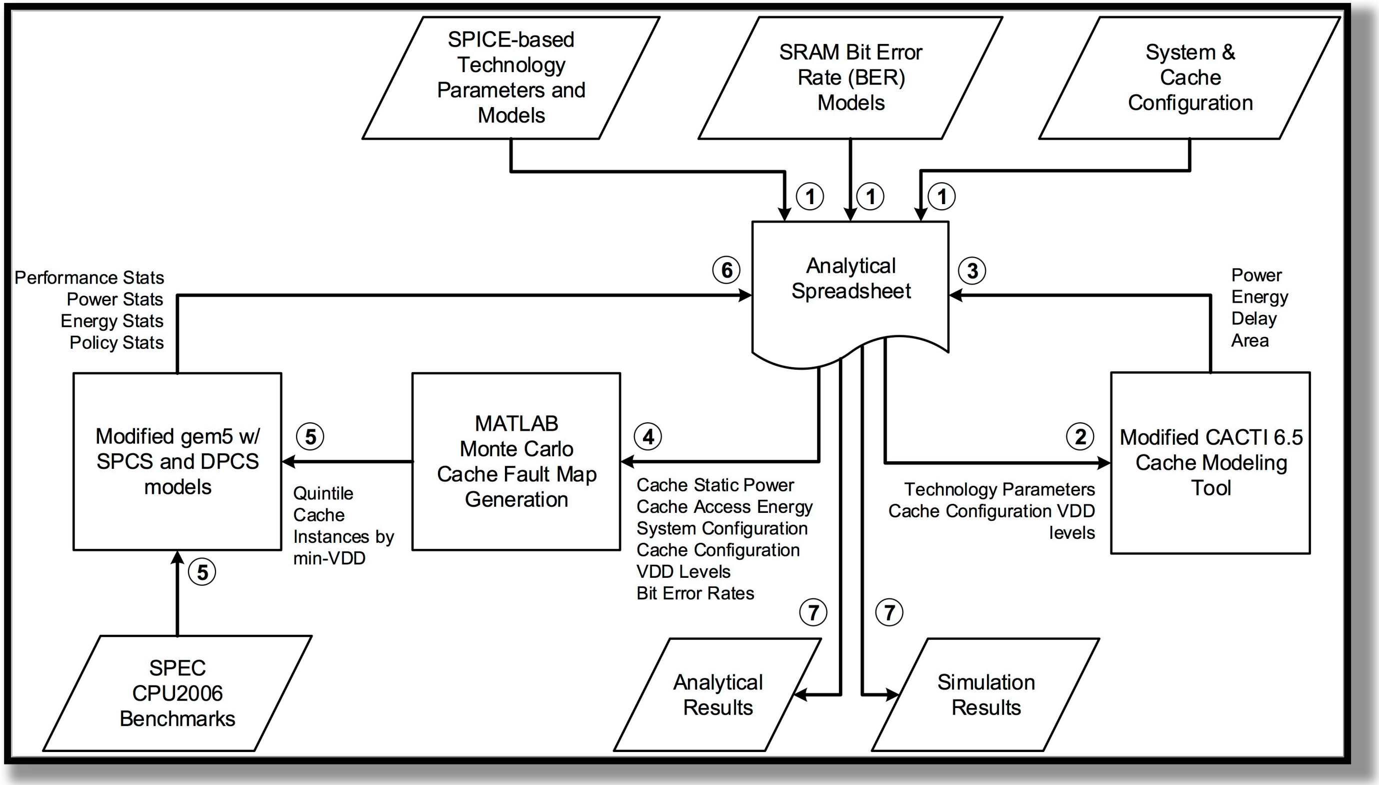
- *Static (SPCS) Policy*: Choose single optimal VDD at design, test, or boot time
- *Dynamic Policy 1 (DPCS1)*: Based on *access diversity <----> spatial locality*
- *Dynamic Policy 2 (DPCS2): Based on average access time <----> temporal locality*

DPCS: Performance OK → lower VDD, etc.
Adapt within and across applications

Evaluation Setup

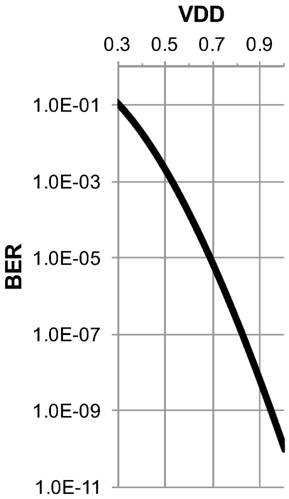
[Gottscho et al. DAC’14, TACO’15]

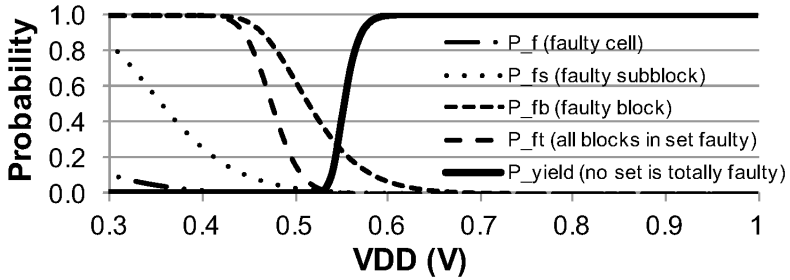
- 45nm SOI
- 2 system/cache configurations for L1 & L2
- 3 permitted VDD levels
- SPEC CPU2006

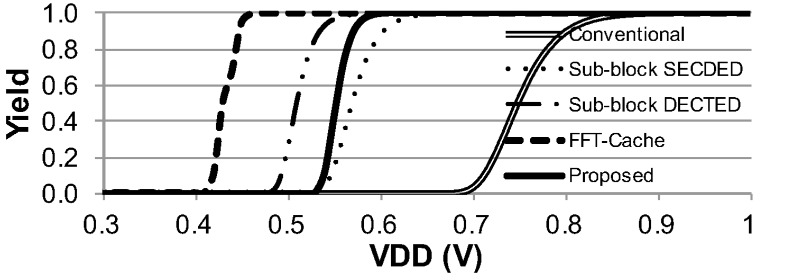


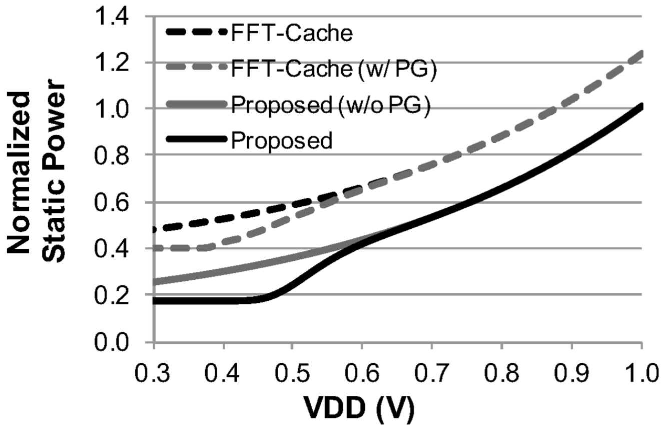
SPICE-based Technology Parameters and Models
SRAM Bit Error Rate (BER) Models
System & Cache Configuration
1
1
1
Analytical Spreadsheet
6
3
Performance Stats
Power Stats
Energy Stats
Policy Stats
Power
Energy
Delay
Area
Modified gem5 w/ SPCS and DPCS models
5
MATLAB Monte Carlo Cache Fault Map Generation
4
2
Modified CACTI 6.5 Cache Modeling Tool
Quintile Cache Instances by min-VDD
Cache Static Power
Cache Access Energy
System Configuration
Cache Configuration
VDD Levels
Bit Error Rates
Technology Parameters
Cache Configuration VDD levels
5
7
7
SPEC CPU2006 Benchmarks
Analytical Results
Simulation Results

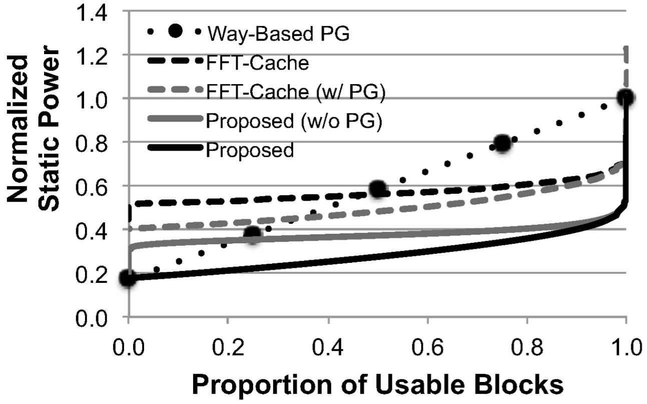
Analytical Results



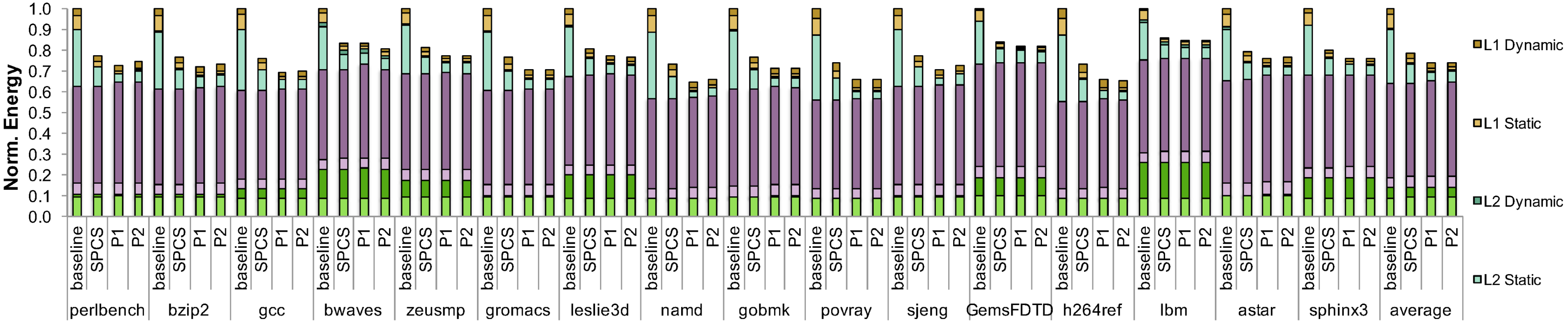


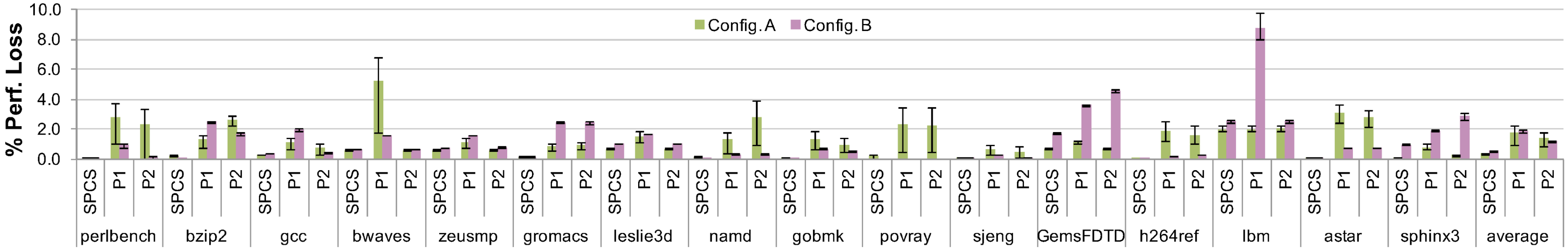


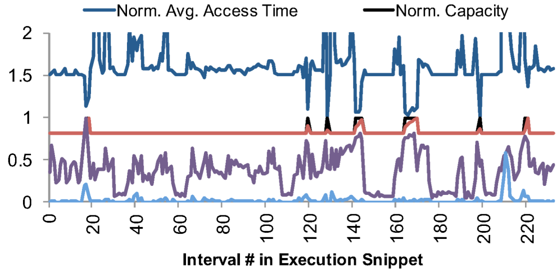


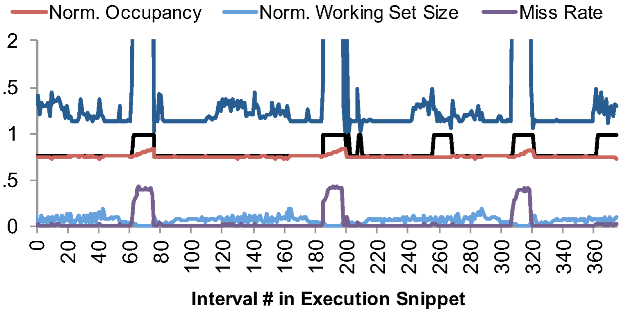


Simulation Results









Summary

[Gottscho et al. DAC'14, TACO'15]

Power vs. capacity tuning

Useful energy efficiency knob, complements DVFS

Fault Inclusion Property

Exploit it for efficient storage of variation signatures

Opportunistic cache energy savings

Leverage variability without harming reliability or performance

Against baseline of SRAM cache @ 1V:

- SPCS: 62% (22%) total cache (system) energy savings
- DPCS: 79% (26%) total cache (system) energy savings
- DPCS: average 2.24 % performance overhead
- 6% area overhead

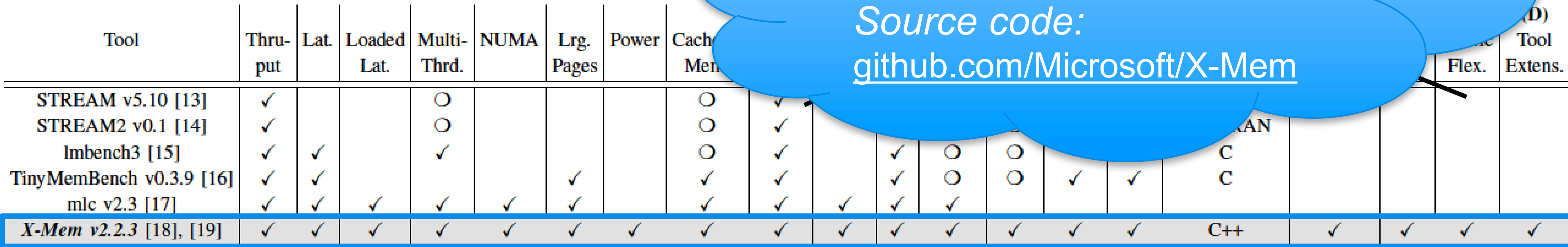
Bonus Slides

4. X-Mem

Motivation: Memory is Important in Cloud Computing

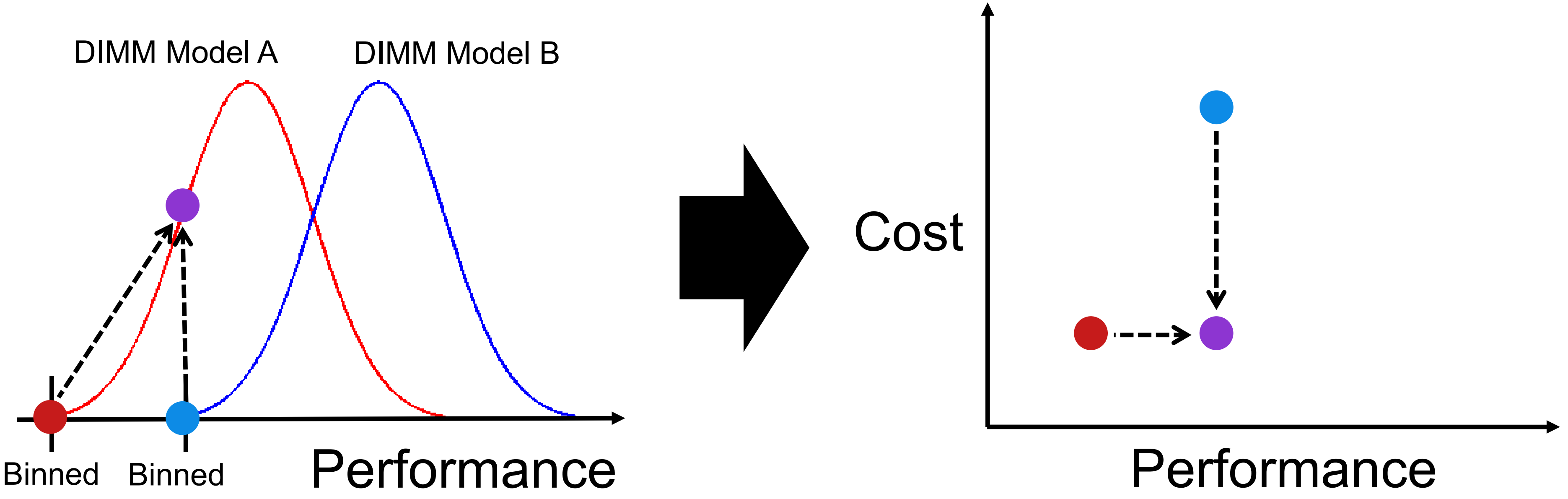
- *Cloud subscribers* want to maximize app. performance
- *Cloud providers* want to minimize CapEx/OpEx given SLAs
- Needs pressure memory hierarchy: characterization is critical
- Memory benchmarking tools don't meet key requirements
  - (A) Access pattern diversity
  - (B) Platform variability
  - (C) Metric flexibility
  - (D) Tool extensibility

We propose X-Mem, a new tool!
Project homepage:
nanocad-lab.github.io/X-Mem
Source code:
github.com/Microsoft/X-Mem

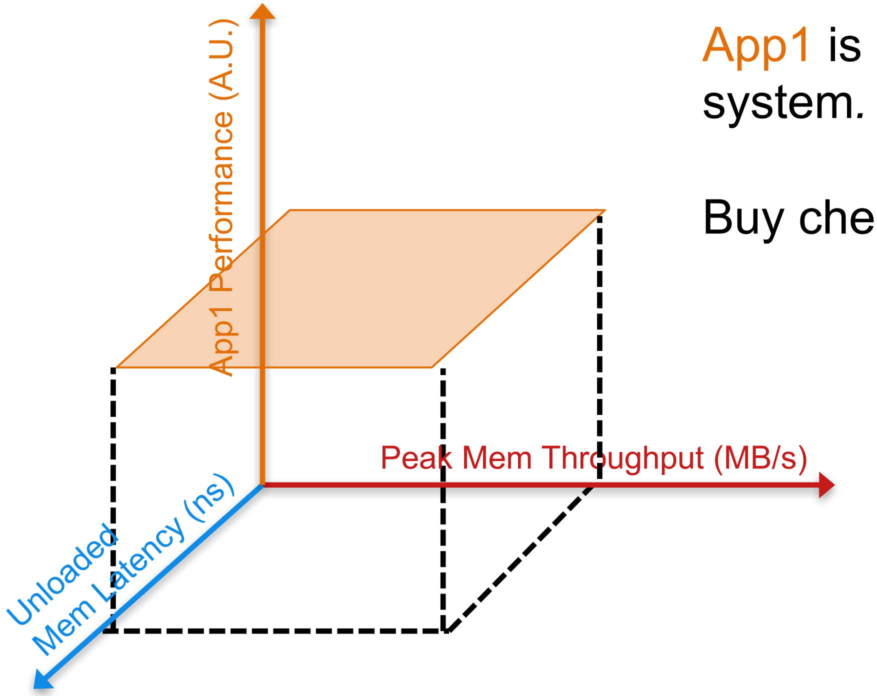


Tool	Thru-put	Lat.	Loaded Lat.	Multi-Thrd.	NUMA	Lrg. Pages	Power	Cach Men	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Flex.	(D) Tool Extens.
STREAM v5.10 [13]	✓			○				○	✓	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		
STREAM2 v0.1 [14]	✓			○				○	✓	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	RAN				
lmbench3 [15]	✓	✓		✓				○	✓		✓	○	○	[illegible]	[illegible]	C				
TinyMemBench v0.3.9 [16]	✓	✓				✓		✓	✓		✓	○	○	✓	✓	C				
mlc v2.3 [17]	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓								
X-Mem v2.2.3 [18], [19]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	C++	✓	✓	✓	✓

Idea: Exploit Memory Process Variation for Higher Performance/Watt at Lower Cost

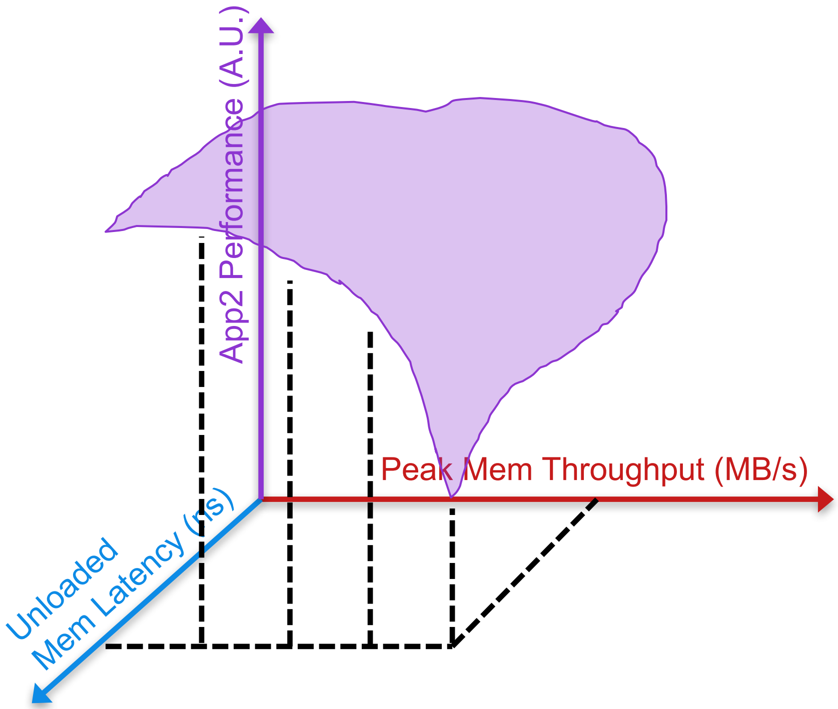


Idea: DIMM Provisioning



App1 is *insensitive to memory performance* on this system.

Buy cheaper, lower performance DIMMs.



But App2 is *sensitive to memory performance* on this system.

Buy higher performance DIMMs, which come at higher cost.

Related Work

- My own prior work showed up to 25% power variation across DDR3 DIMMs of same specs [Gottscho et al. ESL'12]
- ViPZonE exploited power variation for energy savings [Bathen et al. CODES+ISSS'12, Gottscho et al. TC'15]
- A recent study proposed variation-aware tuning of DRAM timings [Chandrakesar et al. DATE'14]
  - They found up to 25-35% latency and/or bandwidth improvements possible at DRAM level
  - Problems: Their approach is not scalable & system-level impact was not evaluated
  - Recently followed up by AL-DRAM [Lee et al. HPCA'15], which was done concurrently with this work

Question: How to evaluate efficacy of variation-aware DRAM performance tuning?

Objective

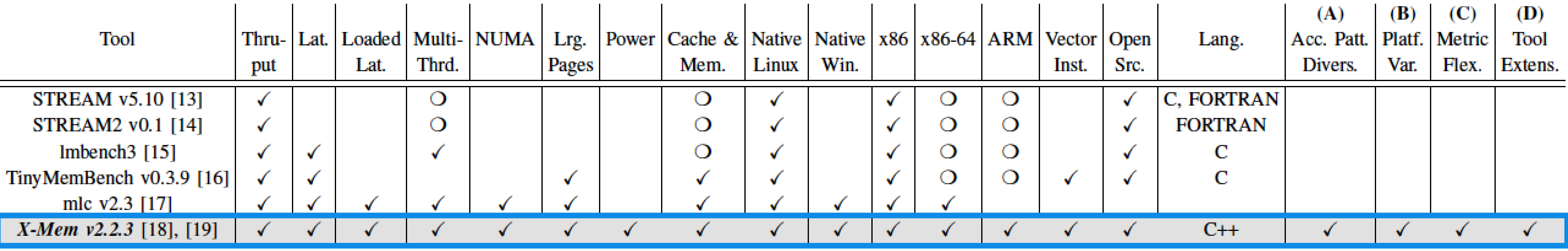
[Gottscho et al. ISPASS'16]

Develop a new software tool

that can evaluate memory variation-aware solutions

for improving energy efficiency

and support other uses by the community.

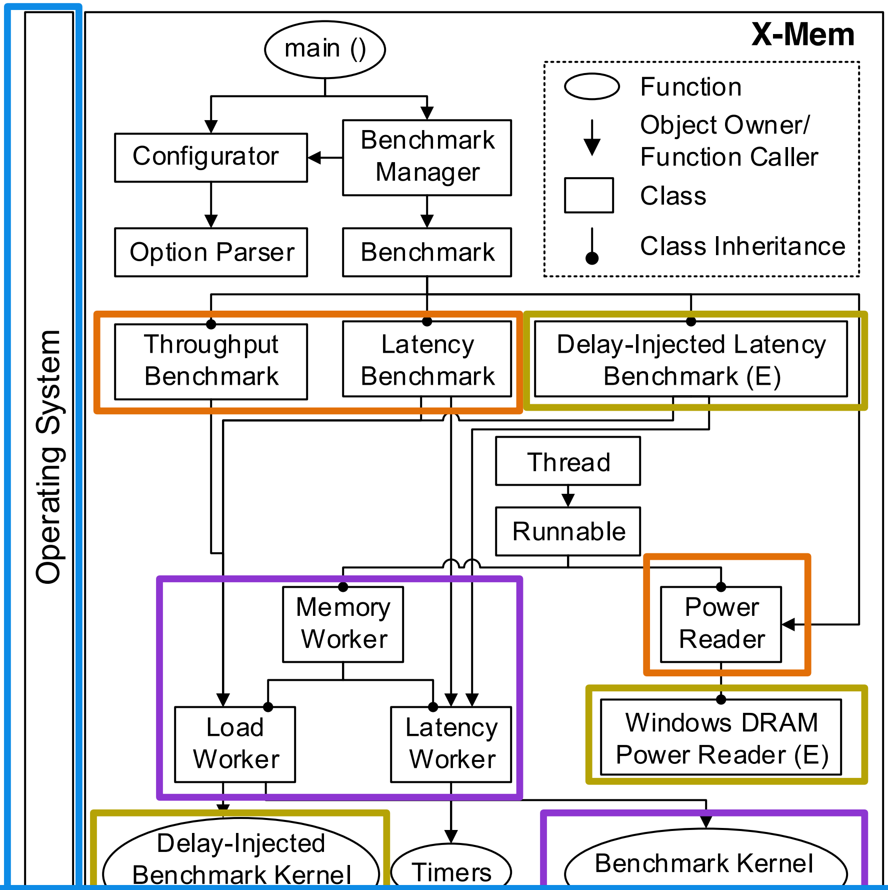


Tool	Thru-put	Lat.	Loaded Lat.	Multi-Thrd.	NUMA	Lrg. Pages	Power	Cache & Mem.	Native Linux	Native Win.	x86	x86-64	ARM	Vector Inst.	Open Src.	Lang.	(A) Acc. Patt. Divers.	(B) Platf. Var.	(C) Metric Flex.	(D) Tool Extens.
STREAM v5.10 [13]	✓			○				○	✓		✓	○	○		✓	C, FORTRAN				
STREAM2 v0.1 [14]	✓			○				○	✓		✓	○	○		✓	FORTRAN				
lmbench3 [15]	✓	✓		✓				○	✓		✓	○	○		✓	C				
TinyMemBench v0.3.9 [16]	✓	✓				✓		✓	✓		✓	○	○	✓	✓	C				
mlc v2.3 [17]	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓								
X-Mem v2.2.3 [18], [19]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	C++	✓	✓	✓	✓

X-Mem Design

[Gottscho et al. ISPASS'16]

- Object-oriented C++
- Caches through DRAM
- (A) Access pattern diversity
- (B) Platform variability
- (C) Metric flexibility
- (D) Tool extensibility
- Open-source
- User-friendly CLI & documentation



Latest SW, documentation, data available @ https://nanocad-lab.github.io/X-Mem

X-Mem Feature:
(A) Access Pattern Diversity

[Gottscho et al. ISPASS’16]

6 Degrees of Freedom

1.	Access granularity	32, 64, 128, and 256-bit chunk sizes
2.	Access types	Read or write
3.	Access patterns	Random, sequential and strided in ± 2^{0-4} chunks
4.	Parallelism	Multithreaded
5.	Page sizes	Large and normal
6.	Topologies	CPU and memory NUMA nodes, core affinity

- **(D) Tool Extensibility**: Developers can easily add specialized patterns through new benchmark kernel functions

X-Mem Feature:
(B) Platform Abstractions

[Gottscho et al. ISPASS'16]

1.	OS Support	Windows, GNU/Linux
2.	Architectural support	x86, x86-64 with(out) AVX SIMD extensions ARMv7 with(out) NEON SIMD extensions, ARMv8

- All OS and hardware-specific implementation details are abstracted via OOP techniques and preprocessor macros
  - Includes benchmark kernels, high-resolution timers, power measurement *etc.*
- Portable SCons-based build system using Python
- **(D) Tool Extensibility**: Ports to other OSes and architectures possible with relatively little effort. Enables apples-to-apples memory hierarchy comparisons.

X-Mem Feature:
(C) Metric Flexibility

[Gottscho et al. ISPASS'16]

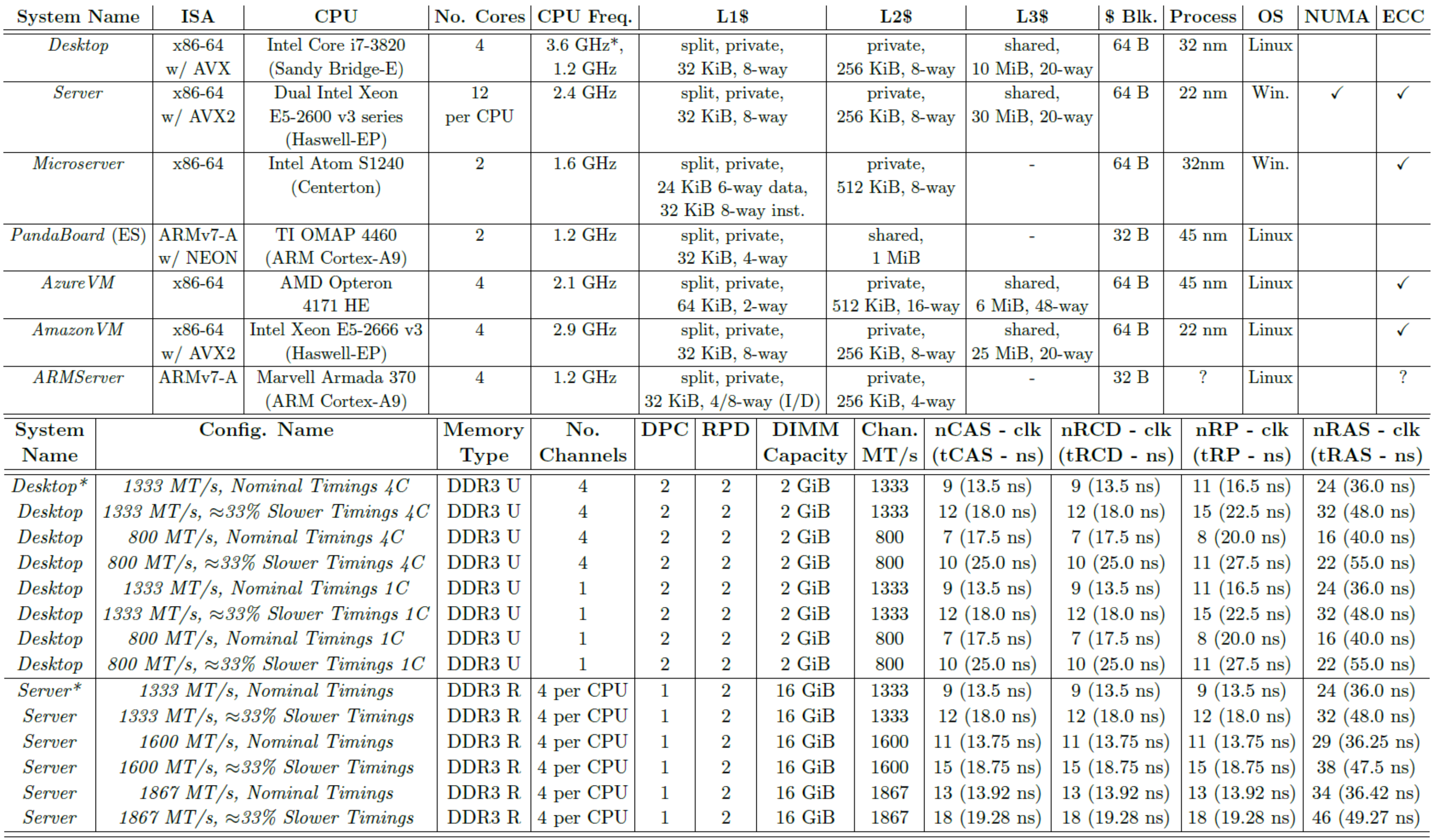
- **Performance:** X-Mem measures real performance of the memory hierarchy as could be seen by an application
  - Average aggregate throughput
  - Average unloaded latency
  - Average loaded latency

- **Power**
  - Average and peak DRAM power
  - Simple software hooks for custom power measurement hardware

- **(D) Tool Extensibility:** shared-data throughput, percentile statistics, variance, data-aware power/performance bookkeeping for NVMs *etc.*

Experimental Platform Details

[Gottscho et al. ISPASS'16]



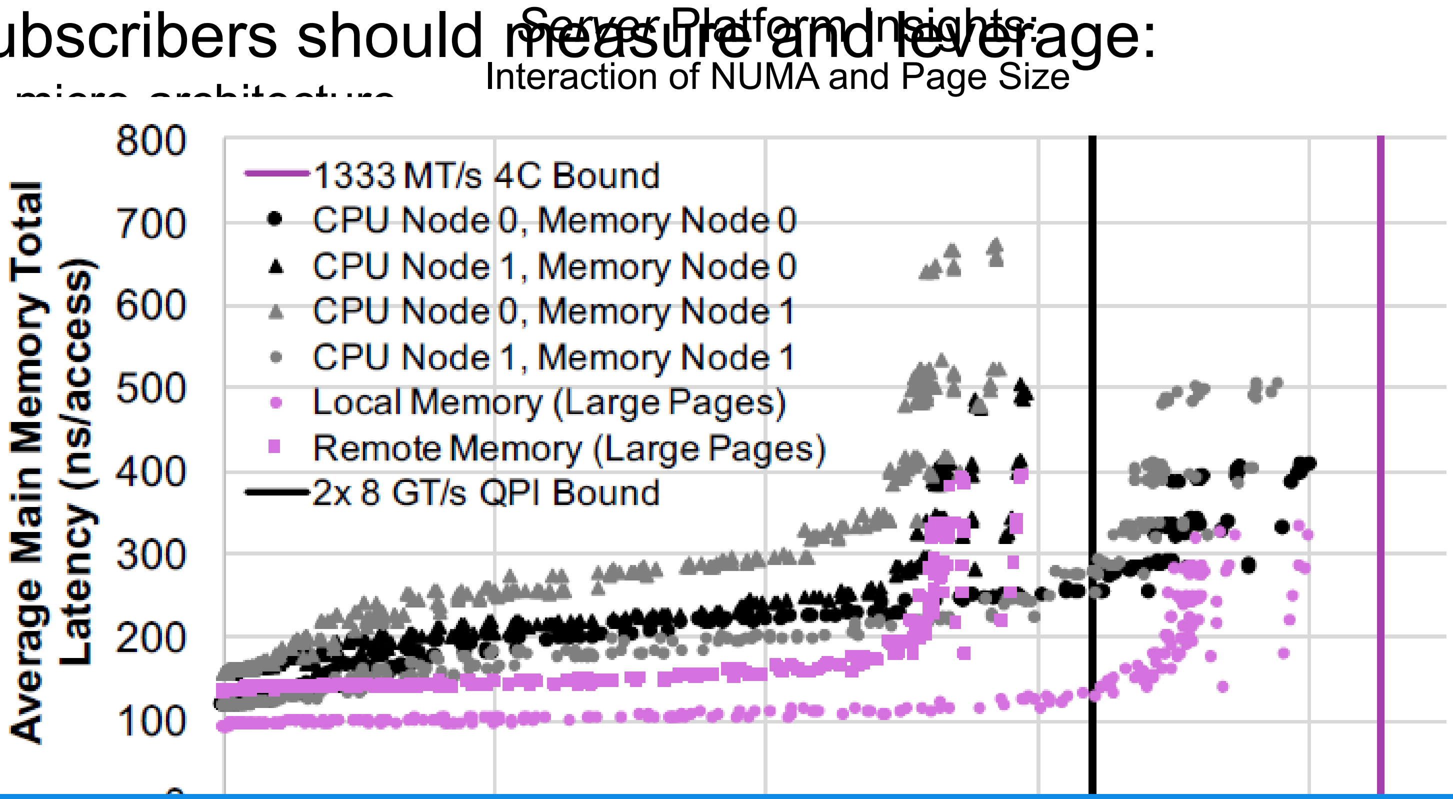
System Name	ISA	CPU	No. Cores	CPU Freq.	L1$	L2$	L3$	$ Blk.	Process	OS	NUMA	ECC
Desktop	x86-64 w/ AVX	Intel Core i7-3820 (Sandy Bridge-E)	4	3.6 GHz*, 1.2 GHz	split, private, 32 KiB, 8-way	private, 256 KiB, 8-way	shared, 10 MiB, 20-way	64 B	32 nm	Linux		
Server	x86-64 w/ AVX2	Dual Intel Xeon E5-2600 v3 series (Haswell-EP)	12 per CPU	2.4 GHz	split, private, 32 KiB, 8-way	private, 256 KiB, 8-way	shared, 30 MiB, 20-way	64 B	22 nm	Win.	✓	✓
Microserver	x86-64	Intel Atom S1240 (Centerton)	2	1.6 GHz	split, private, 24 KiB 6-way data, 32 KiB 8-way inst.	private, 512 KiB, 8-way	-	64 B	32nm	Win.		✓
PandaBoard (ES)	ARMv7-A w/ NEON	TI OMAP 4460 (ARM Cortex-A9)	2	1.2 GHz	split, private, 32 KiB, 4-way	shared, 1 MiB	-	32 B	45 nm	Linux		
AzureVM	x86-64	AMD Opteron 4171 HE	4	2.1 GHz	split, private, 64 KiB, 2-way	private, 512 KiB, 16-way	shared, 6 MiB, 48-way	64 B	45 nm	Linux		✓
AmazonVM	x86-64 w/ AVX2	Intel Xeon E5-2666 v3 (Haswell-EP)	4	2.9 GHz	split, private, 32 KiB, 8-way	private, 256 KiB, 8-way	shared, 25 MiB, 20-way	64 B	22 nm	Linux		✓
ARMServer	ARMv7-A	Marvell Armada 370 (ARM Cortex-A9)	4	1.2 GHz	split, private, 32 KiB, 4/8-way (I/D)	private, 256 KiB, 4-way	-	32 B	?	Linux		?

System	Config. Name	Memory	No.	DPC	RPD	DIMM	Chan.	nCAS - clk	nRCD - clk	nRP - clk	nRAS - clk
Name		Type	Channels			Capacity	MT/s	(tCAS - ns)	(tRCD - ns)	(tRP - ns)	(tRAS - ns)
*Desktop**	*1333 MT/s, Nominal Timings 4C*	DDR3 U	4	2	2	2 GiB	1333	9 (13.5 ns)	9 (13.5 ns)	11 (16.5 ns)	24 (36.0 ns)
Desktop	*1333 MT/s, ≈33% Slower Timings 4C*	DDR3 U	4	2	2	2 GiB	1333	12 (18.0 ns)	12 (18.0 ns)	15 (22.5 ns)	32 (48.0 ns)
Desktop	*800 MT/s, Nominal Timings 4C*	DDR3 U	4	2	2	2 GiB	800	7 (17.5 ns)	7 (17.5 ns)	8 (20.0 ns)	16 (40.0 ns)
Desktop	*800 MT/s, ≈33% Slower Timings 4C*	DDR3 U	4	2	2	2 GiB	800	10 (25.0 ns)	10 (25.0 ns)	11 (27.5 ns)	22 (55.0 ns)
Desktop	*1333 MT/s, Nominal Timings 1C*	DDR3 U	1	2	2	2 GiB	1333	9 (13.5 ns)	9 (13.5 ns)	11 (16.5 ns)	24 (36.0 ns)
Desktop	*1333 MT/s, ≈33% Slower Timings 1C*	DDR3 U	1	2	2	2 GiB	1333	12 (18.0 ns)	12 (18.0 ns)	15 (22.5 ns)	32 (48.0 ns)
Desktop	*800 MT/s, Nominal Timings 1C*	DDR3 U	1	2	2	2 GiB	800	7 (17.5 ns)	7 (17.5 ns)	8 (20.0 ns)	16 (40.0 ns)
Desktop	*800 MT/s, ≈33% Slower Timings 1C*	DDR3 U	1	2	2	2 GiB	800	10 (25.0 ns)	10 (25.0 ns)	11 (27.5 ns)	22 (55.0 ns)
*Server**	*1333 MT/s, Nominal Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1333	9 (13.5 ns)	9 (13.5 ns)	9 (13.5 ns)	24 (36.0 ns)
Server	*1333 MT/s, ≈33% Slower Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1333	12 (18.0 ns)	12 (18.0 ns)	12 (18.0 ns)	32 (48.0 ns)
Server	*1600 MT/s, Nominal Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1600	11 (13.75 ns)	11 (13.75 ns)	11 (13.75 ns)	29 (36.25 ns)
Server	*1600 MT/s, ≈33% Slower Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1600	15 (18.75 ns)	15 (18.75 ns)	15 (18.75 ns)	38 (47.5 ns)
Server	*1867 MT/s, Nominal Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1867	13 (13.92 ns)	13 (13.92 ns)	13 (13.92 ns)	34 (36.42 ns)
Server	*1867 MT/s, ≈33% Slower Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1867	18 (19.28 ns)	18 (19.28 ns)	18 (19.28 ns)	46 (49.27 ns)

Case Study 1: Characterization of the Memory Hierarchy for Cloud Subscribers

[Gottscho et al. ISPASS'16]

- Cloud subscribers should measure and leverage:
  - Cache micro-architecture
  - Syster
- Underst performa
  - Worklc
  - Workir
  - Data a
  - When,



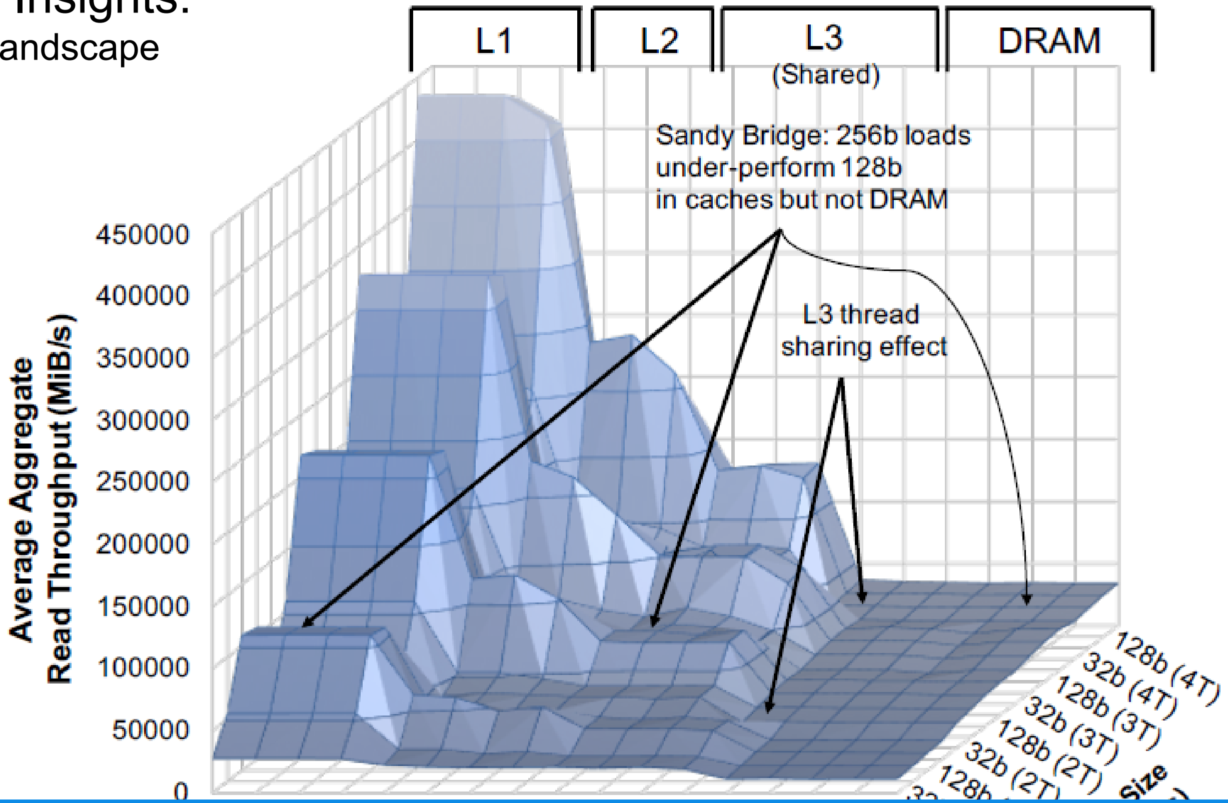
X-Mem can uncover performance effects that only manifest at a system level

Case Study 1: Characterization of the Memory Hierarchy for Cloud Subscribers

[Gottscho et al. ISPASS'16]

Desktop Platform Insights:

Memory Hierarchy Landscape



Sandy Bridge: 256b loads under-perform 128b in caches but not DRAM

L3 thread sharing effect

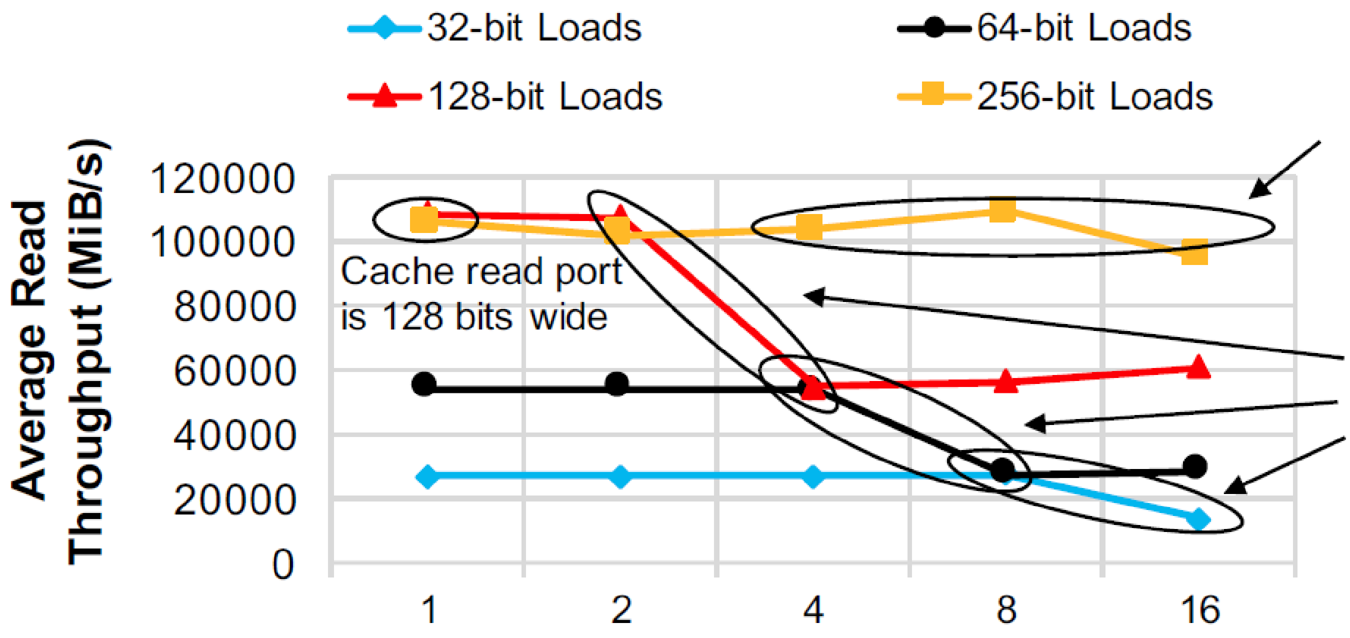
X-Mem can quantify various aspects of performance for cache and memory architectures

Case Study 1: Characterization of the Memory Hierarchy for Cloud Subscribers

[Gottscho et al. ISPASS'16]

Desktop Platform Insights:

L1 Data Cache Architecture



Cache read port is 128 bits wide

256-bit reads are 2x 128-bit reads, don't suffer from striding

Drops occur at 64 B strides, revealing block size

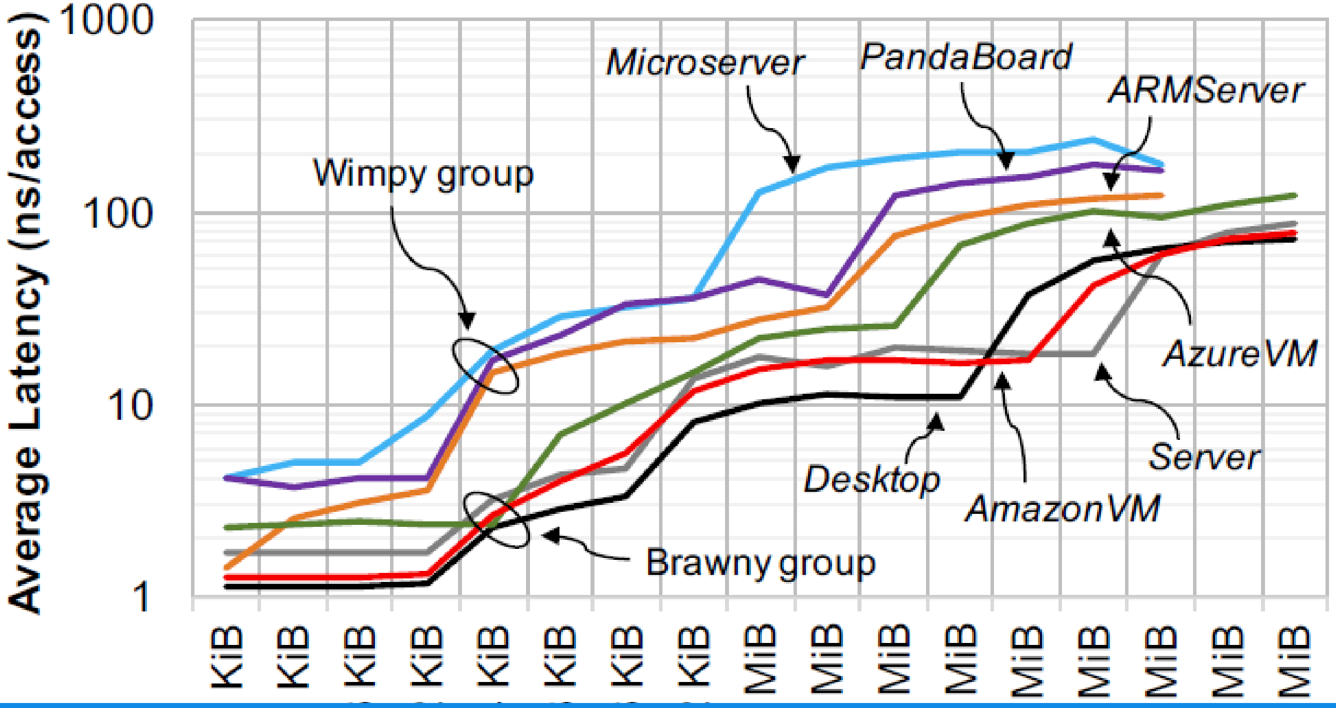
X-Mem can reveal hidden details of cache and memory micro-architectures

Case Study 2: Cross-Platform Insights for Cloud Subscribers

[Gottscho et al. ISPASS'16]

- Cloud subscribers can use X-Mem to directly compare memory pe
  - x86 vs. AR
  - Virtual vs.
  - Wimpy vs.
  - Apples-to-
- This capab
  - Choose a

Cross-Platform Insights:
Unloaded Latency of Caches and DRAM



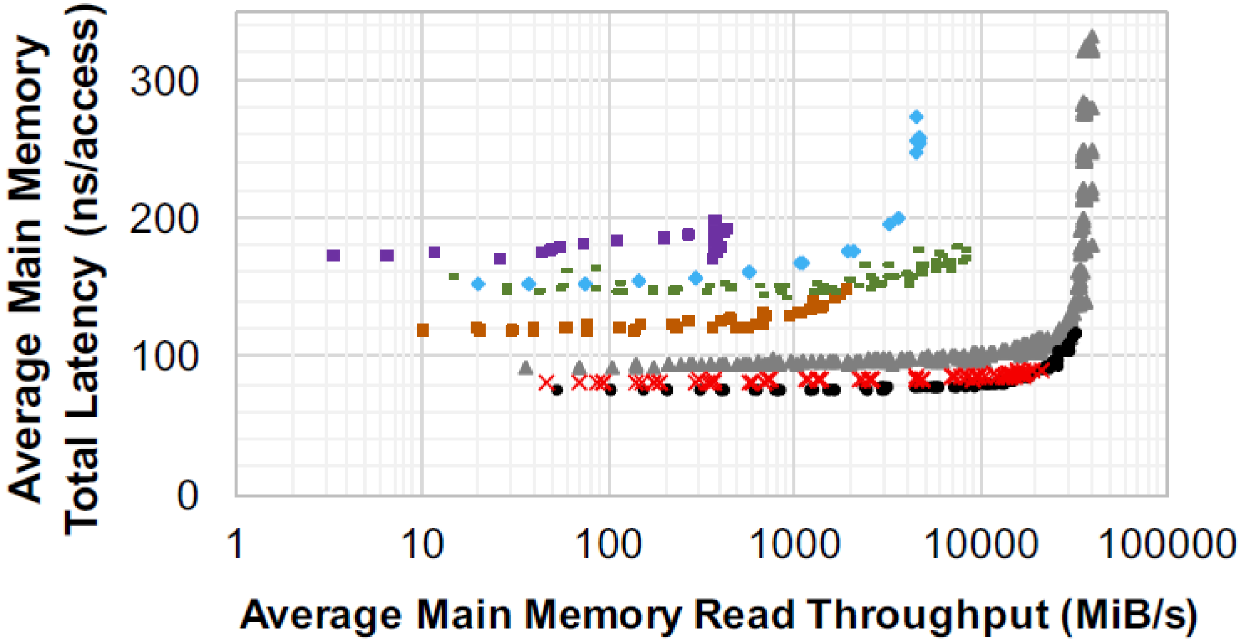
X-Mem can perform apples-to-apples comparisons between diverse platforms

Case Study 2: Cross-Platform Insights for Cloud Subscribers

[Gottscho et al. ISPASS'16]

Cross-Platform Insights:

Main Memory Loaded Latency

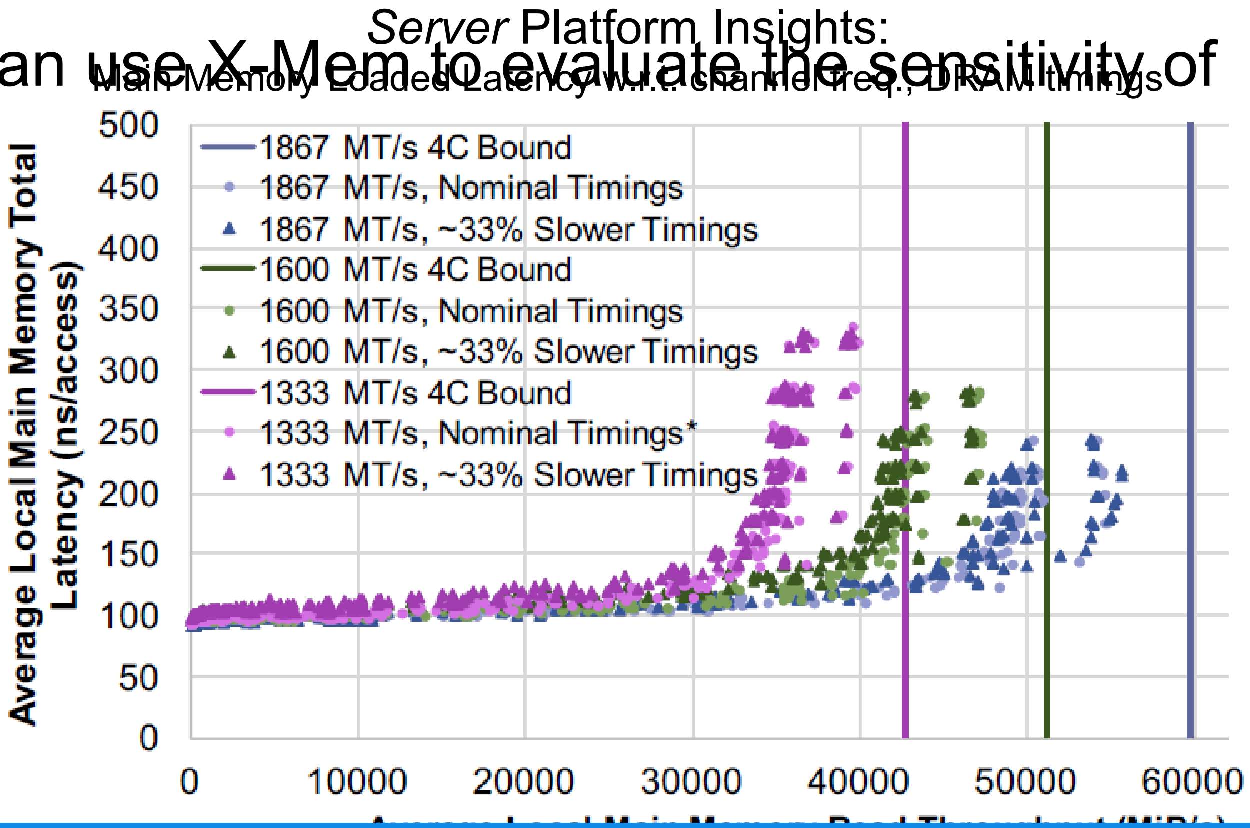


X-Mem can perform apples-to-apples comparisons between diverse platforms

Case Study 3: Impact of Variation-Aware Tuning of Platform Configurations for Cloud Providers

[Gottscho et al. ISPASS'16]

- Cloud providers can use X-Mem to evaluate the sensitivity of system-level perf configurations
  - Number of DRAM
  - DRAM timing para
    [Gottscho ESL'12, CODI
  - Analyze throughpu etc.
- This capability en
  - Optimally configure
  - Maximize performa

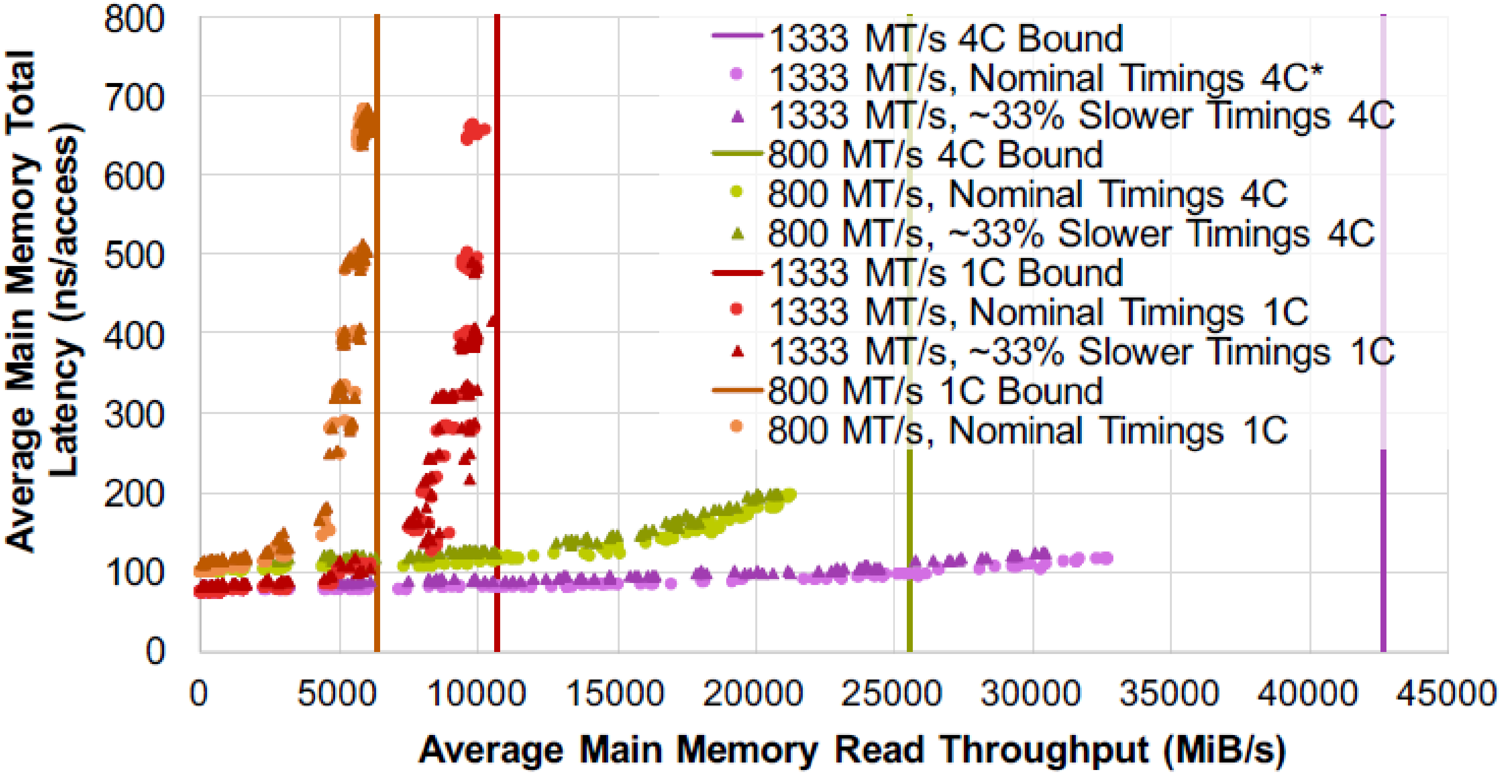


X-Mem can facilitate studies of platform configurations and impact of variation-aware tuning

Case Study 3: Impact of Variation-Aware Tuning of Platform Configurations for Cloud Providers

[Gottscho et al. ISPASS'16]

Desktop @ 3.6 GHz Platform

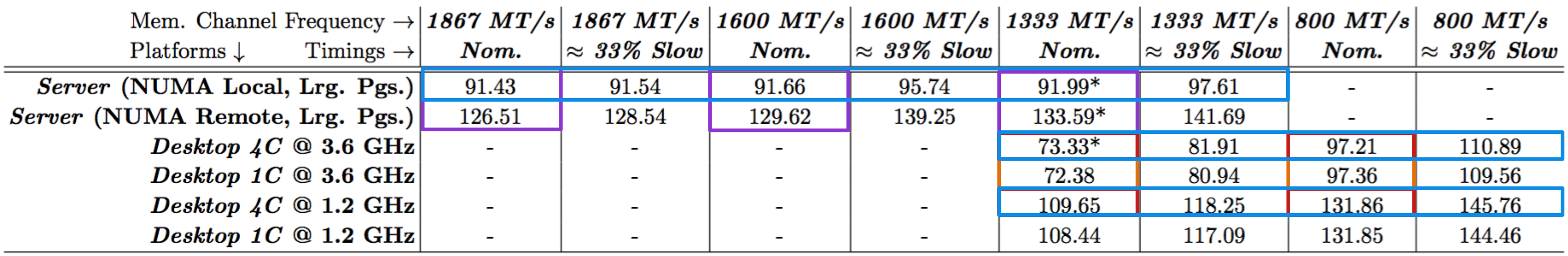


Case Study 3: Impact of Variation-Aware Tuning of Platform Configurations for Cloud Providers

[Gottscho et al. ISPASS'16]

Remote access: Up to 45% slower

channels: no impact



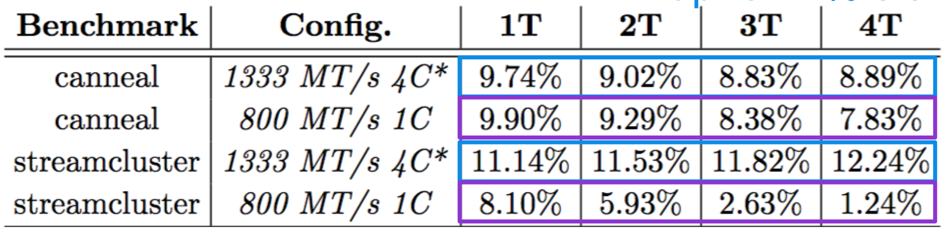
Mem. Channel Frequency →	*1867 MT/s*	*1867 MT/s*	*1600 MT/s*	*1600 MT/s*	*1333 MT/s*	*1333 MT/s*	*800 MT/s*	*800 MT/s*
Platforms ↓ Timings →	*Nom.*	*≈ 33% Slow*	*Nom.*	*≈ 33% Slow*	*Nom.*	*≈ 33% Slow*	*Nom.*	*≈ 33% Slow*
***Server* (NUMA Local, Lrg. Pgs.)**	91.43	91.54	91.66	95.74	91.99*	97.61	-	-
***Server* (NUMA Remote, Lrg. Pgs.)**	126.51	128.54	129.62	139.25	133.59*	141.69	-	-
***Desktop 4C* @ 3.6 GHz**	-	-	-	-	73.33*	81.91	97.21	110.89
***Desktop 1C* @ 3.6 GHz**	-	-	-	-	72.38	80.94	97.36	109.56
***Desktop 4C* @ 1.2 GHz**	-	-	-	-	109.65	118.25	131.86	145.76
***Desktop 1C* @ 1.2 GHz**	-	-	-	-	108.44	117.09	131.85	144.46

Figure: Sensitivity of unloaded latency (ns/access) w.r.t. CPU & DDR3 frequency, DRAM timing, # DDR3 channels

CPU underclocked 3X: 50% higher DRAM lat.

DRAM timings 33% slower
→ up to 12% slower overall

Benchmarks are memory BW starved; *relative* impact of DRAM timings is LESS w/ more threads



Benchmark	Config.	1T	2T	3T	4T
canneal	*1333 MT/s 4C**	9.74%	9.02%	8.83%	8.89%
canneal	*800 MT/s 1C*	9.90%	9.29%	8.38%	7.83%
streamcluster	*1333 MT/s 4C**	11.14%	11.53%	11.82%	12.24%
streamcluster	*800 MT/s 1C*	8.10%	5.93%	2.63%	1.24%

Figure: Impact of 33% slower DRAM timings

Memory has enough BW; benchmarks appear latency-bound

X-Mem shows that variation-aware DRAM perf. tuning makes sense only when BW bottlenecks are removed

Summary

[Gottscho et al. ISPASS'16]

Case Study 1: expose micro-architectural and system-level details for precise engineering

Case Study 2: compare memory hierarchies of different platforms for software optimization

Case Study 3: tune memory performance using variability for specific applications

New flexible tool for characterizing memory systems

Surpasses capabilities of all prior tools

Several key features enable broad usability

(A) Access pattern diversity, (B) Platform variability,
(C) Metric flexibility, (D) Tool extensibility

Characterization is critical to opportunistic memory systems

Data-driven exploitation of performance and power variability

Bonus Slides

5. Performability

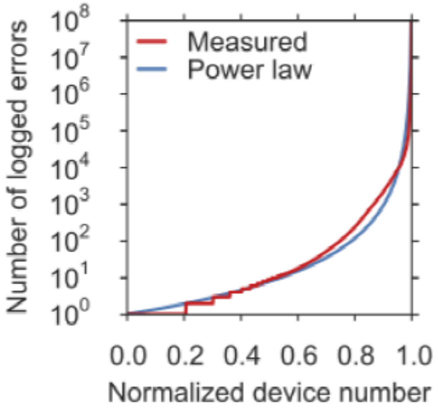
Motivation & Related Work

- Datacenters are growing in size
  - Prolific demand for memory
  - Increasing DRAM error rates observed in the field [Li et al. '10, Schroeder et al. CACM'11, Sridharan & Liberty '12, Hwang et al. ASPLOS'12, DeBardeleben et al. SELSE'14, Meza et al. DSN'15]
- Memory errors cause significant loss of availability and higher TCO [Meza et al. DSN'15, Nikolaou et al. MICRO'15]
- …Even *corrected* errors do! [Meza et al. DSN'15]
  - Why? Apparently, performance impacts caused by "avalanches" of errors

Need controlled analysis of memory errors to answer the field studies' call for action

Memory is Important in Cloud

- Main memory in cloud:
  - Impacts providers: capital and operational expenditures
  - Impacts subscribers: application performance
- Deep understanding of memory system can help minimize cost-to-benefit ratio for both
- DRAM faults and fault-tolerance techniques affect:
  - Performance and availability of servers
  - Total cost of ownership (TCO) of datacenter



[Meza et al. DSN’15]

How to optimally provision, manage, and retire DRAM to minimize datacenter TCO while satisfying performance and availability SLAs?

DRAM Fault Models

Fault Classification

Granularity

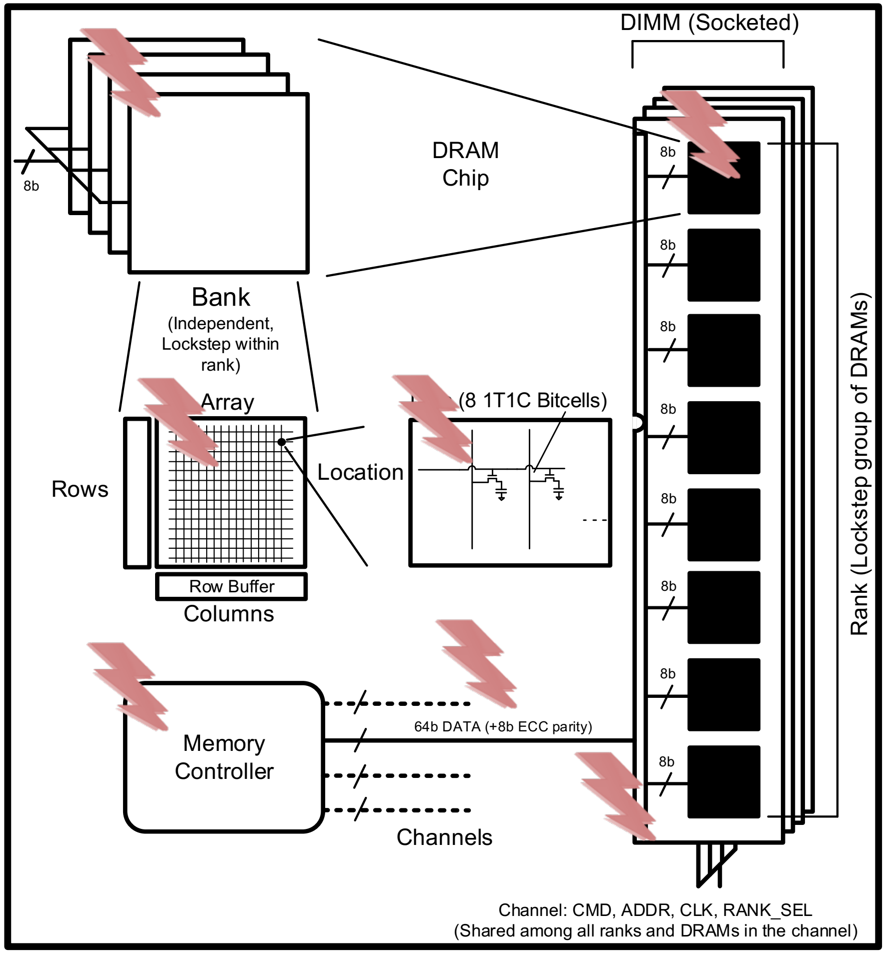
- Socket
- Channel
- Rank
- Chip
- Bank
- Row
- Column
- Multi-bit
- Single-bit

Time

- Permanent
- Intermittent
- Single-events

Space

- Within-pages
- Neighboring pages
- Across pages


DIMM (Socketed)
DRAM Chip
8b
Bank
(Independent, Lockstep within rank)
Array
(8 1T1C Bitcells)
Rows
Location
Row Buffer
Columns
Rank (Lockstep group of DRAMs)
Memory Controller
64b DATA (+8b ECC parity)
Channels
Channel: CMD, ADDR, CLK, RANK_SEL
(Shared among all ranks and DRAMs in the channel)

DRAM Fault Tolerance Techniques Available on Current Cloud Servers (Haswell-EP)

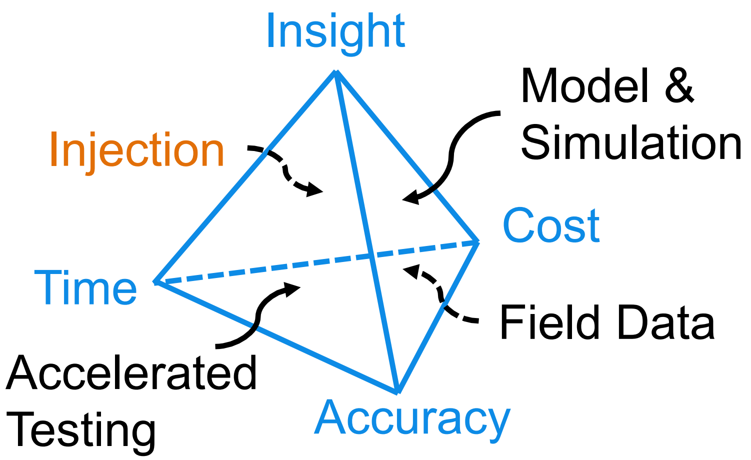
Active SW Mgmt.	**Prevention** • Page retirement (PFA)	**Recovery** • Clean page fault (pseudo-checkpoint) – req. kernel mods
HW/FW Mgmt.	**Reactive** • Rank sparing • Channel mirroring • Bit-steering/device tag	**Active** • Demand scrubbing • Patrol scrubbing
HW Coding	**Detection Only** • Address/command parity • Data parity	**Detection and Correction** • SECDED ECC • ChipKill/SDDC ECC

When and how to use these techniques?

Approach: How to Study Memory Resiliency in the Real World?

- Faults are considered rare events
  - Reliability engineering is a challenge

Four approaches to study resiliency



Unfortunate tetrahedron: Choose 1 face

Method	Advantages	Disadvantages
Field data	Ground truth, big-data statistics	Insight, post-facto analysis
Accelerated testing	Accuracy, hardware-in-the-loop	Cost, design space
Simulated modeling	Transparency, control	Time, scalability
Fault injection	Tractable, pre-deployment	Accuracy, assumptions

Measuring Performance Impact of Injected Memory Errors

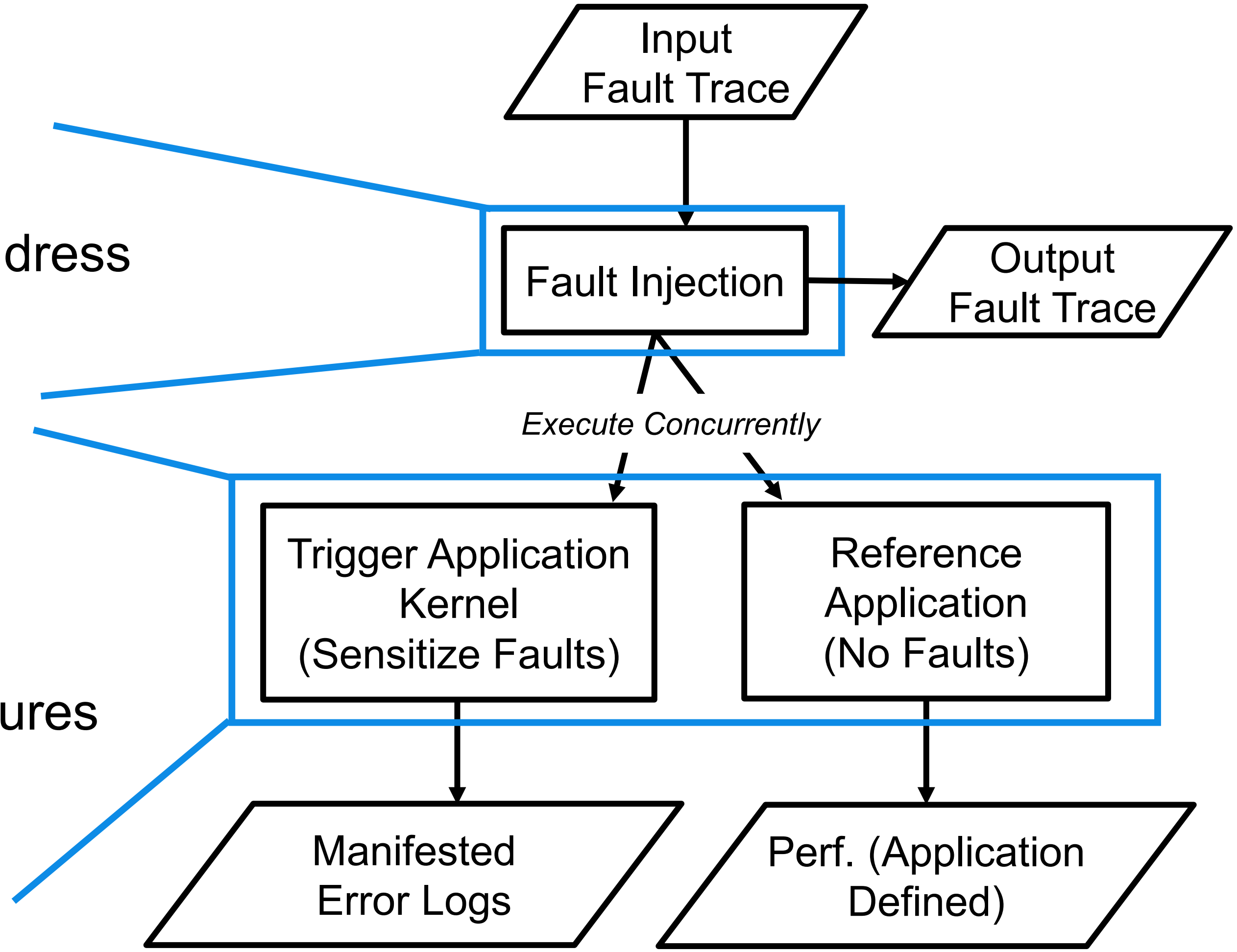
X-Mem modified for fault injection

Steps:

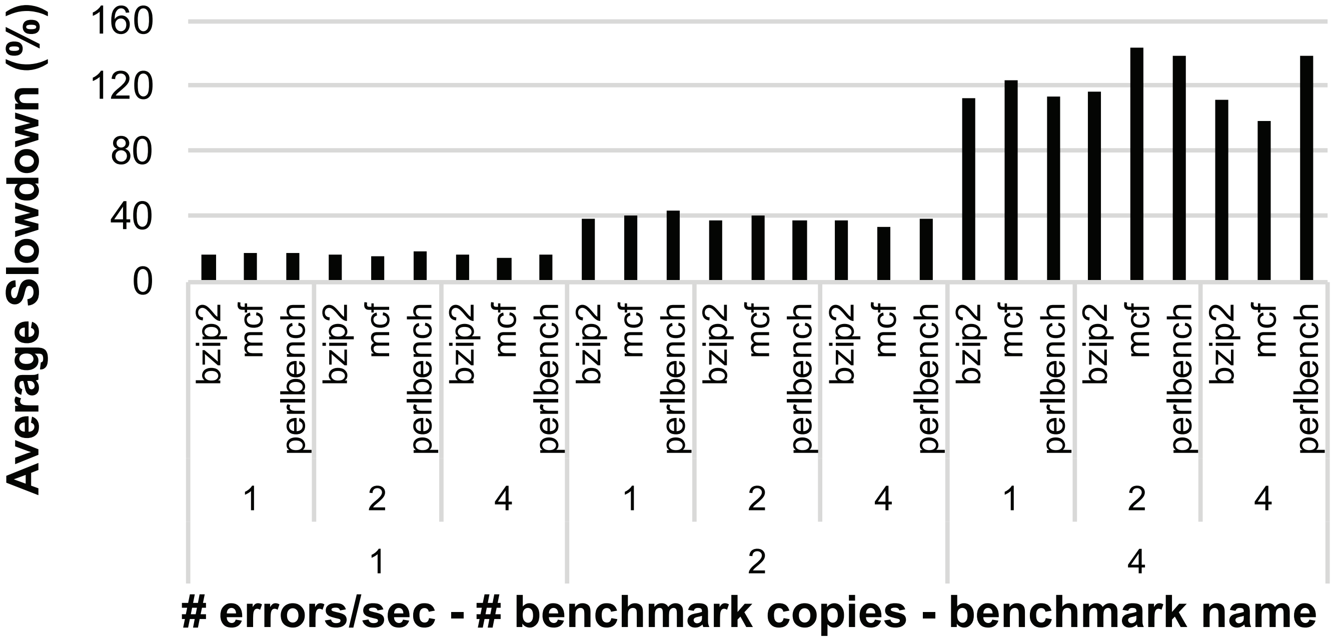
1. Virtual → physical address
2. Inject faults

Steps:

1. Trigger pings faulty memory
2. Reference app measures performance


Input Fault Trace
Fault Injection
Output Fault Trace
Execute Concurrently
Trigger Application Kernel (Sensitize Faults)
Reference Application (No Faults)
Manifested Error Logs
Perf. (Application Defined)

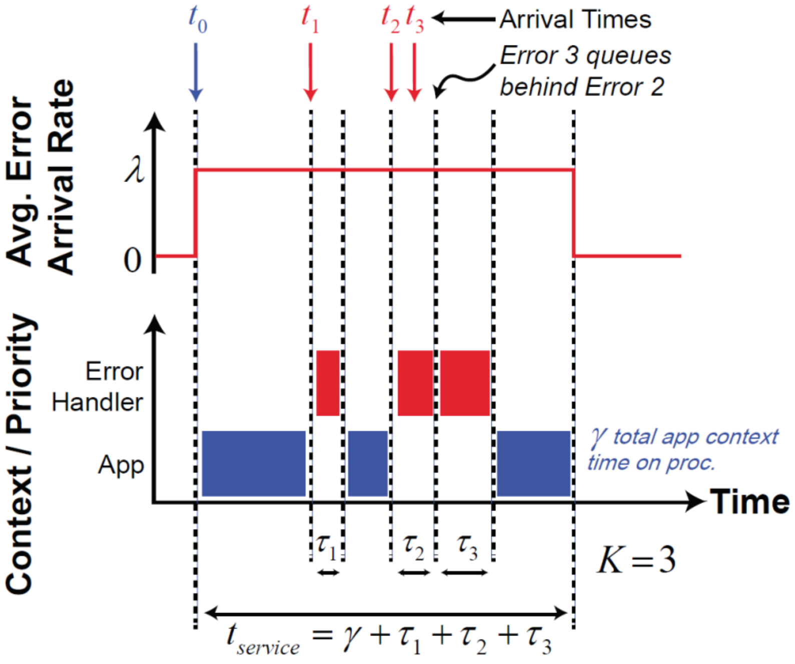
Performance of Batch Applications with SMI Errors



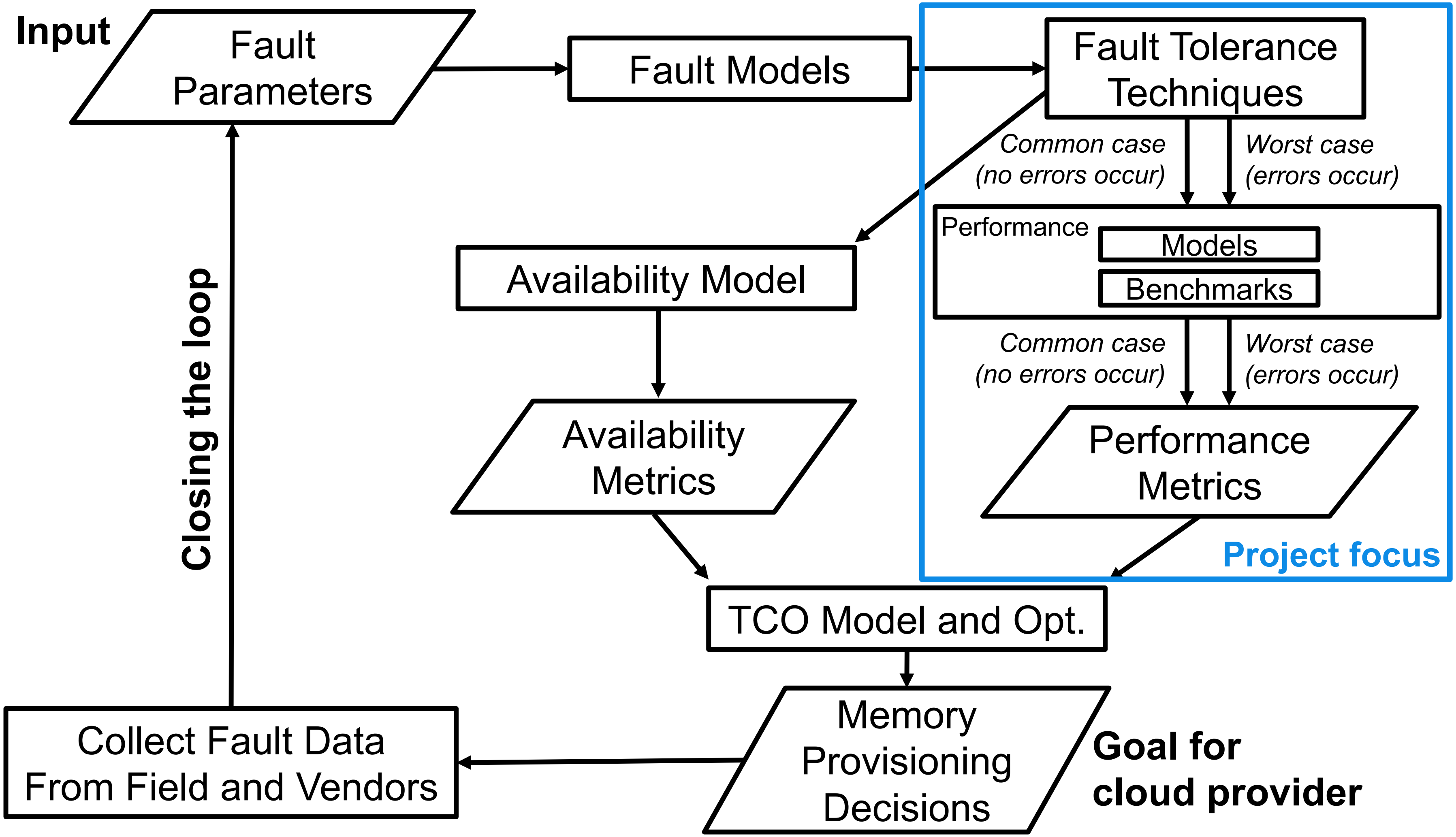
Application Thread-Servicing Time

- Modeled impact of corrected errors on general application performance
- Model combinations
  - System: uniprocessor vs. multiprocessor
  - Error-reporting scheme: broadcast vs. single-issue
  - Application type: batch (throughput-oriented) vs. interactive (latency-oriented)
- Models are built on derived model of application thread-servicing time (ATST) in presence of errors

Application thread-servicing time (ATST)



Framework


Input
Fault Parameters
Fault Models
Fault Tolerance Techniques
Common case (no errors occur)
Worst case (errors occur)
Performance
Models
Benchmarks
Common case (no errors occur)
Worst case (errors occur)
Performance Metrics
Project focus
Availability Model
Availability Metrics
TCO Model and Opt.
Memory Provisioning Decisions
Goal for cloud provider
Collect Fault Data From Field and Vendors
Closing the loop

Outlook

- Datacenter operations
  - Common/worst-case performance impact of memory errors
  - *Optimal servicing of faulty hardware*
  - *Variation-aware memory provisioning*

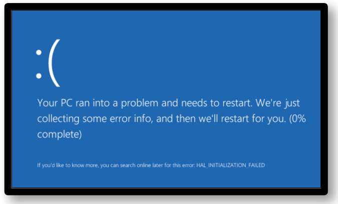
- System design
  - Efficient error-reporting architectures
  - Modeling impact of corrected errors on different applications

Understanding impact of corrected errors is useful for opportunistic memory provisioning

Bonus Slides

6. SDECC

Motivation: Memory Errors are a Major Problem



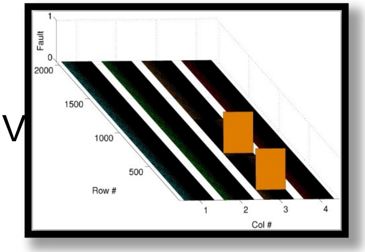
:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

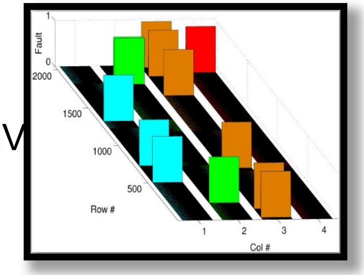
If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION_FAILED

- **System-level effects from embedded to HPC**
  - System crashes
  - Silent data corruption
- **DRAM reliability worsens** with density
  - Google: 70,000 FIT/Mb in commodity DRAM; 8% of modules affected per year;
    4% of servers crash per year [Schroeder CACM'11]
  - Facebook: 2.5% of machines see DRAM errors per month [Meza DSN'15]
- **SRAM stops working** at low voltage
  - 6X fault rate measured from 600mV to 525mV [Gottscho TACO'15]
- **Flash wears out** with usage
  - NASA's Opportunity Mars rover had to reformat its flash in 2014
- **STT-RAM is unpredictable**
  - Stochastic write & thermal instability [Zhao Microelec. Rel.'12]
- **Memory errors will continue to be a challenge!**

550 mV



525 mV



[Gottscho TACO'15]

Motivation & Related Work

Historically separate abstractions:

- Error-correcting codes (ECCs)
  - e.g., SECDED [Hsiao IBM Journal'70], DECTED, ChipKill [Dell IBM'97], SEC-DAEC [Dutta et al. VLSI Test'07], VS-ECC [Alameldeen et al. ISCA'11]
- System-level fault tolerance techniques
  - Checkpoint & recovery
  - Mirroring/sparing

Is there room for anything in between?

Number of Candidate Codewords

[Gottscho '17]

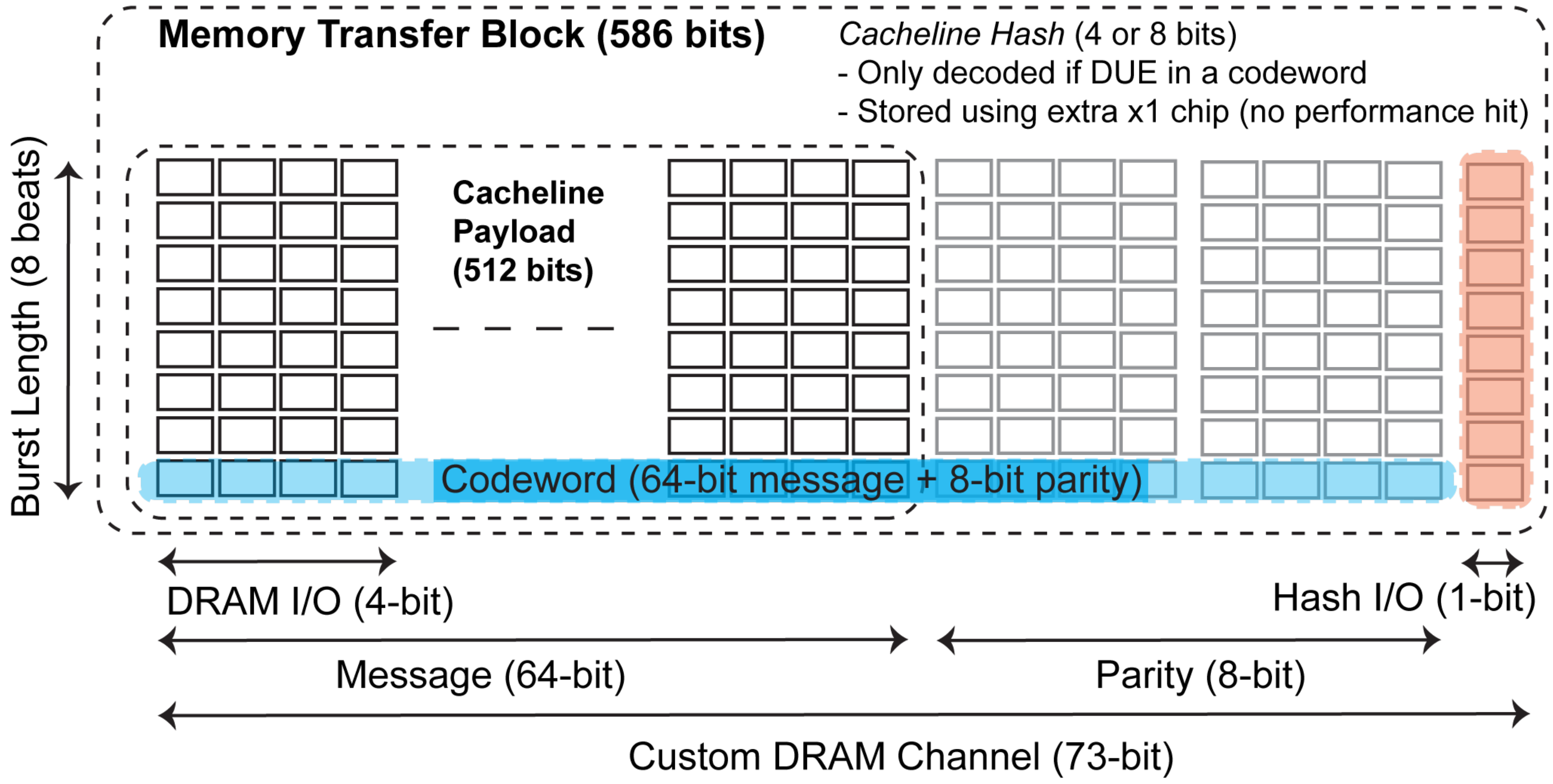
- Surprisingly small number of candidate codewords for any ECC that corrects *t* symbol-wise errors and detects *t+1* errors
- We proved that the average number of candidate codewords is:

$$\mu(n,t,q) = \frac{\binom{2t+2}{t+1} W_q(2t+2)}{\binom{n}{t+1}(q-1)^{t+1}} + 1.$$

- $W_q(2t+2)$: number of min. weight codewords.
- $\binom{2t+2}{t+1}$: number of DUEs distance of exactly $(t+1)$ from each min. weight codeword.
- $\binom{n}{t+1}(q-1)^{t+1}$: number of ways to produce a min. weight DUE.
- $+1$: for the original *correct* message.

Lightweight Hash Implementation: SECDED

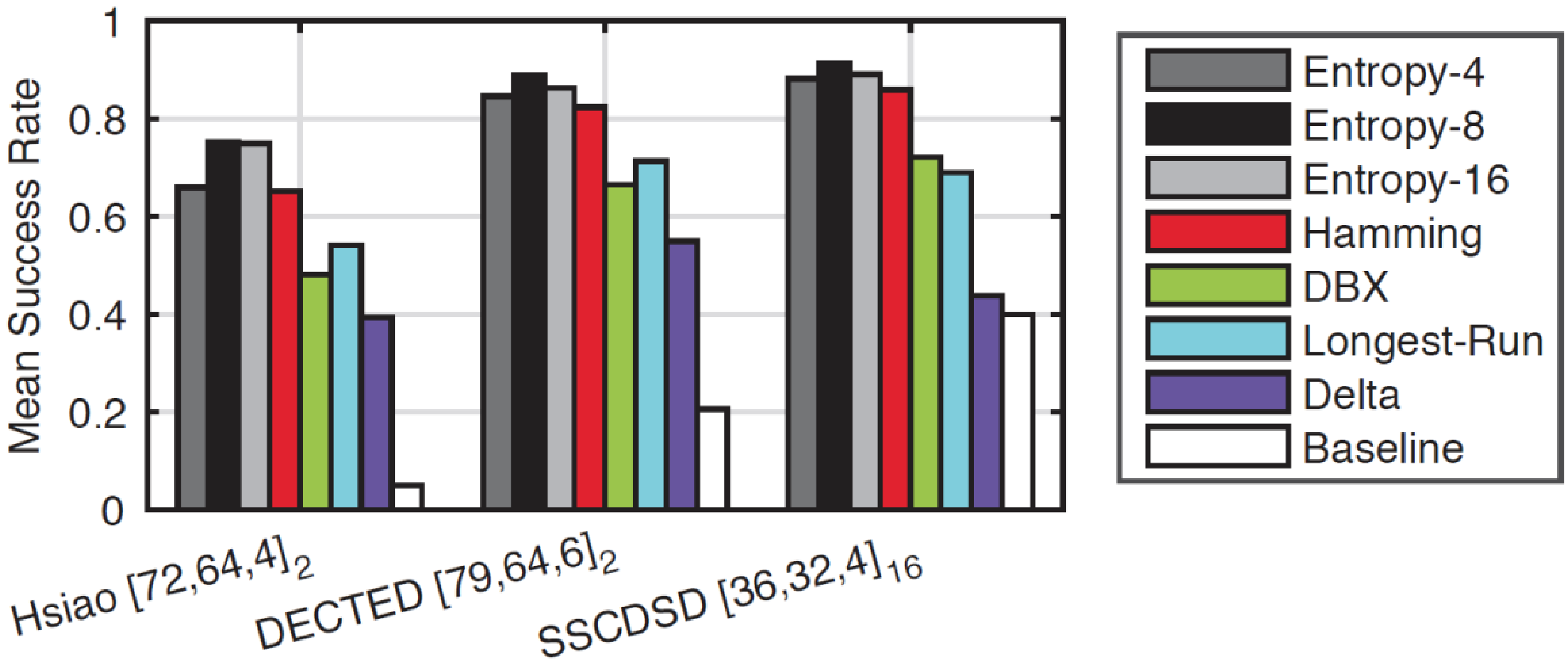
[Gottscho '17]



Memory Transfer Block (586 bits)
Cacheline Hash (4 or 8 bits)
- Only decoded if DUE in a codeword
- Stored using extra x1 chip (no performance hit)
Burst Length (8 beats)
Cacheline Payload (512 bits)
Codeword (64-bit message + 8-bit parity)
DRAM I/O (4-bit)
Hash I/O (1-bit)
Message (64-bit)
Parity (8-bit)
Custom DRAM Channel (73-bit)

Data Recovery Policies Comparison

[Gottscho '17]



What if the Lightweight Hash has an Error?

[Gottscho '17]

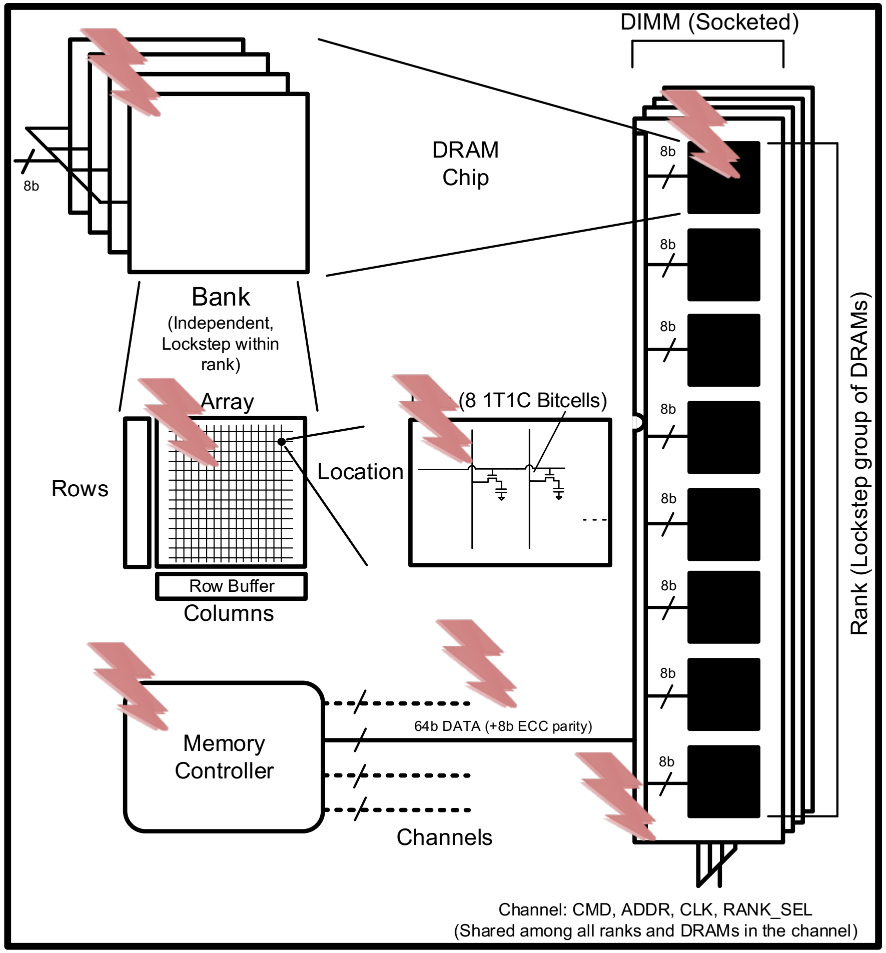
- **Outcome 1 (likely):** hash does not match any candidate, fall back to normal SDECC
- **Outcome 2 (unlikely):** hash collides with wrong candidate, guaranteed miscorrection
  - 0.003% chance for 16-bit hash

Fault Classification

Granularity
- Socket
- Channel
- Rank
- Chip
- Bank
- Row
- Column
- Multi-bit
- Single-bit

Time
- Permanent
- Intermittent
- Single-events

Space
- Within-pages
- Neighboring pages
- Across pages


DIMM (Socketed)
8b
DRAM Chip
Bank
(Independent, Lockstep within rank)
Array
(8 1T1C Bitcells)
Rows
Location
Row Buffer
Columns
Rank (Lockstep group of DRAMs)
Memory Controller
64b DATA (+8b ECC parity)
Channels
Channel: CMD, ADDR, CLK, RANK_SEL
(Shared among all ranks and DRAMs in the channel)

Results: DUE Recovery Breakdown ***with Hashes in Error***

[Gottscho '17]

Suppose 20% of all double-chip DUEs also have random 16-bit hash error

Then actual DUE recovery rate: 99.9999% → 97.2767%

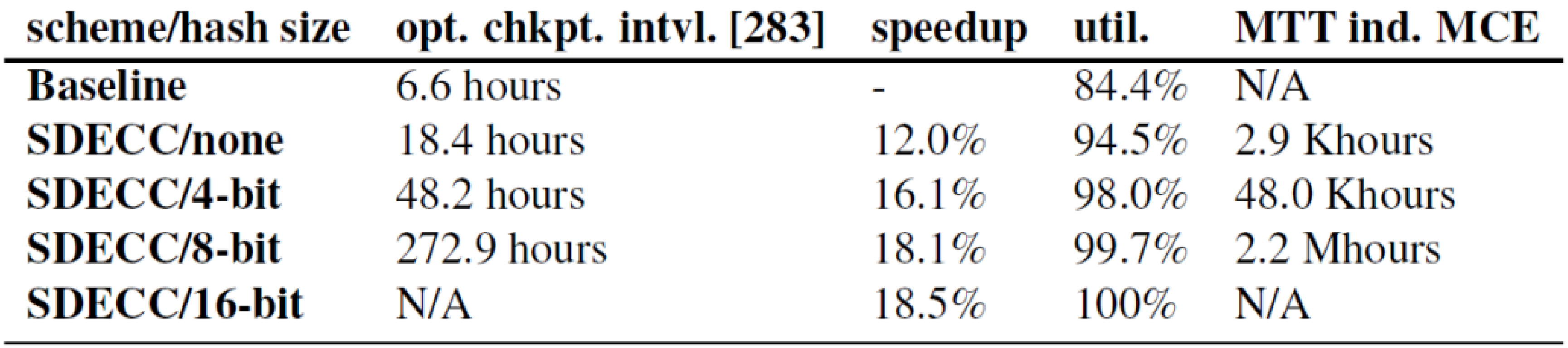
Speedup: 15.6%

Avg. util: 97.6%

MTT ind. MCE: 5.5 Mhours

System-Level Benefits

[Gottscho ‘17]



scheme/hash size	opt. chkpt. intvl. [283]	speedup	util.	MTT ind. MCE
Baseline	6.6 hours	-	84.4%	N/A
SDECC/none	18.4 hours	12.0%	94.5%	2.9 Khours
SDECC/4-bit	48.2 hours	16.1%	98.0%	48.0 Khours
SDECC/8-bit	272.9 hours	18.1%	99.7%	2.2 Mhours
SDECC/16-bit	N/A	18.5%	100%	N/A

Bonus Slides

7. ViFFTo

Motivation

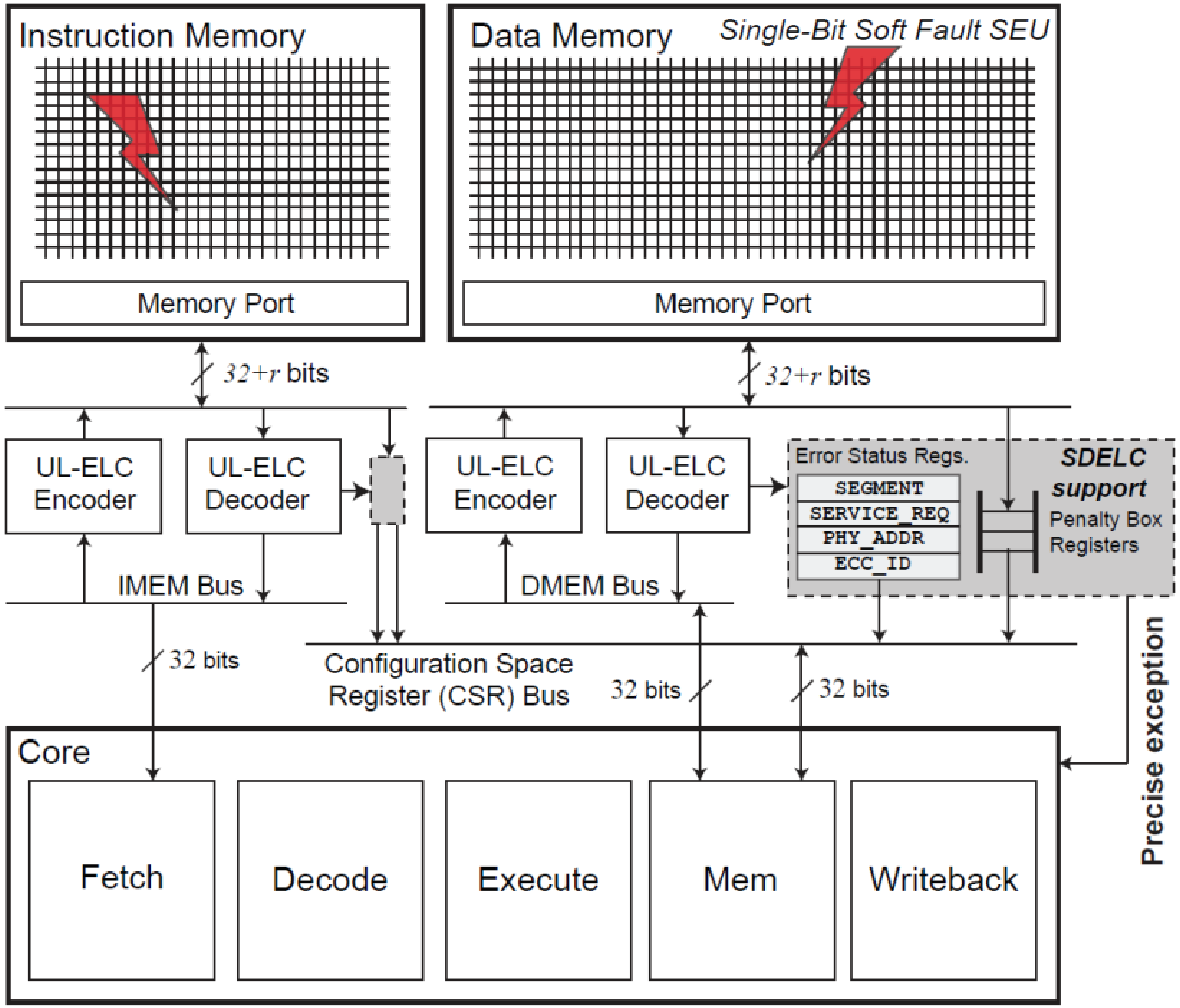
[Gottscho '17]

- Memory resiliency is a key challenge for embedded edge devices in IoT
- Conventional EDAC techniques are too costly and inefficient
- Many embedded systems lack "real" OS w/ virtual memory support

Opportunistic solution for coping with hard memory defects could reduce cost of IoT

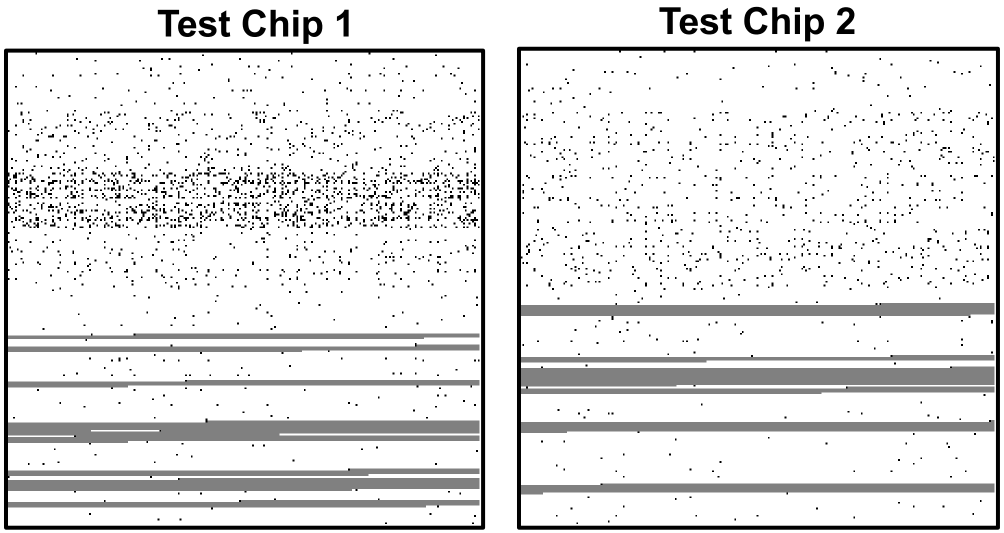
SDELC Architecture

[Gottscho '17]



Results: Hard Faults

[Gottscho '17]



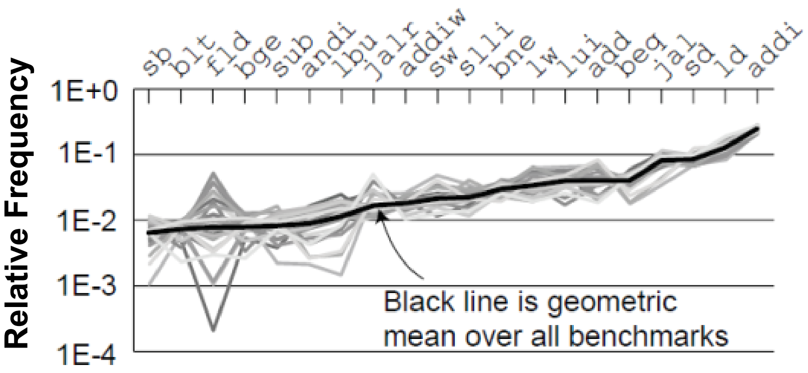
sha packed in inst SPM

SDELC Instruction Recovery Insights

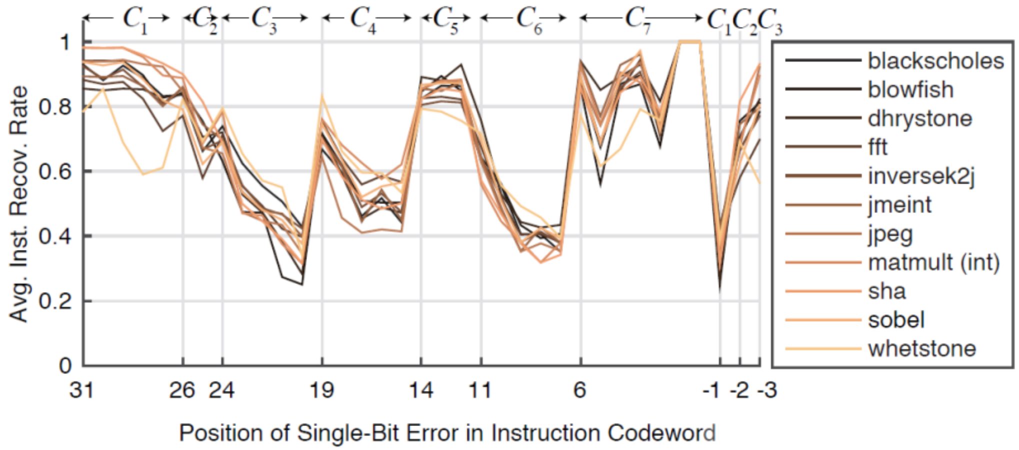
[Gottscho ‘17]

bit →	31 27	26 25	24 20	19 15	14 12	11 7	6 0	-1 -3
Type-U	imm[31:12]					rd	opcode	parity
Type-UJ	imm[20|10:1|11|19:12]					rd	opcode	parity
Type-I	imm[11:0]			rs1	funct3	rd	opcode	parity
Type-SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	parity
Type-S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	parity
Type-R	funct7		rs2	rs1	funct3	rd	opcode	parity
Type-R4	rs3	funct2	rs2	rs1	funct3	rd	opcode	parity

Chunk	C_1 (shared)	C_2 (shared)	C_3 (shared)	C_4	C_5	C_6	C_7	C_3	C_2	C_1
Parity-	00000	00	11111	00000	111	11111	1111111	1	0	0
Check	00000	11	00000	11111	000	11111	1111111	0	1	0
H	11111	00	00000	11111	111	00000	1111111	0	0	1



Black line is geometric mean over all benchmarks



Bonus Slides

8. Conclusion and Directions for Future Work

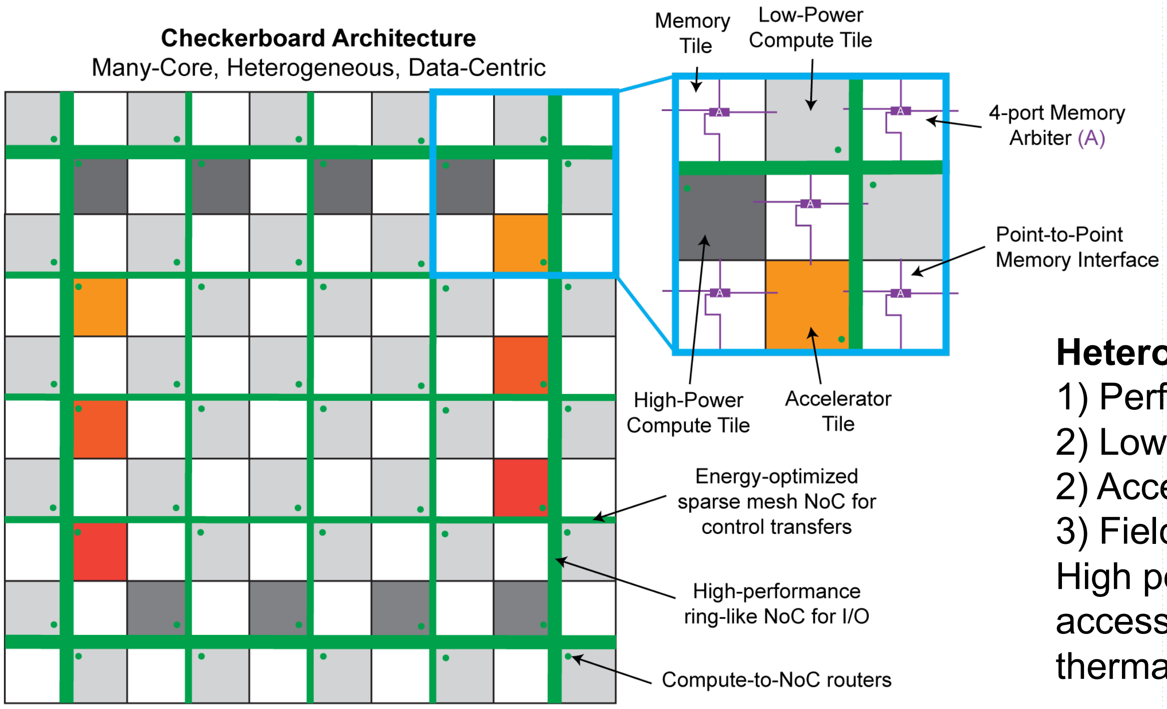
Many-Core Checkerboard Architecture for the Cloud

- Motivation: Datacenter-on-Chip
- Datacenter & cloud apps are often task-parallel or data-parallel
- But currently, they are deployed on commodity hardware
  - General-purpose
  - Also designed for other classes of applications
- Given the scale of datacenter services, it makes sense to consider customized architectures
- Question: Why do we need many-core processors to share a unified address space?
  - Corollary: How could we build better datacenter-specialized chips that allow for greater scale-out capabilities?
  - Not necessarily the wimpy node idea…
- Question: How could we build datacenter-specialized chips given the opportunities presented by emerging device and integration technology?

2D Checkerboard Architecture: High-Level

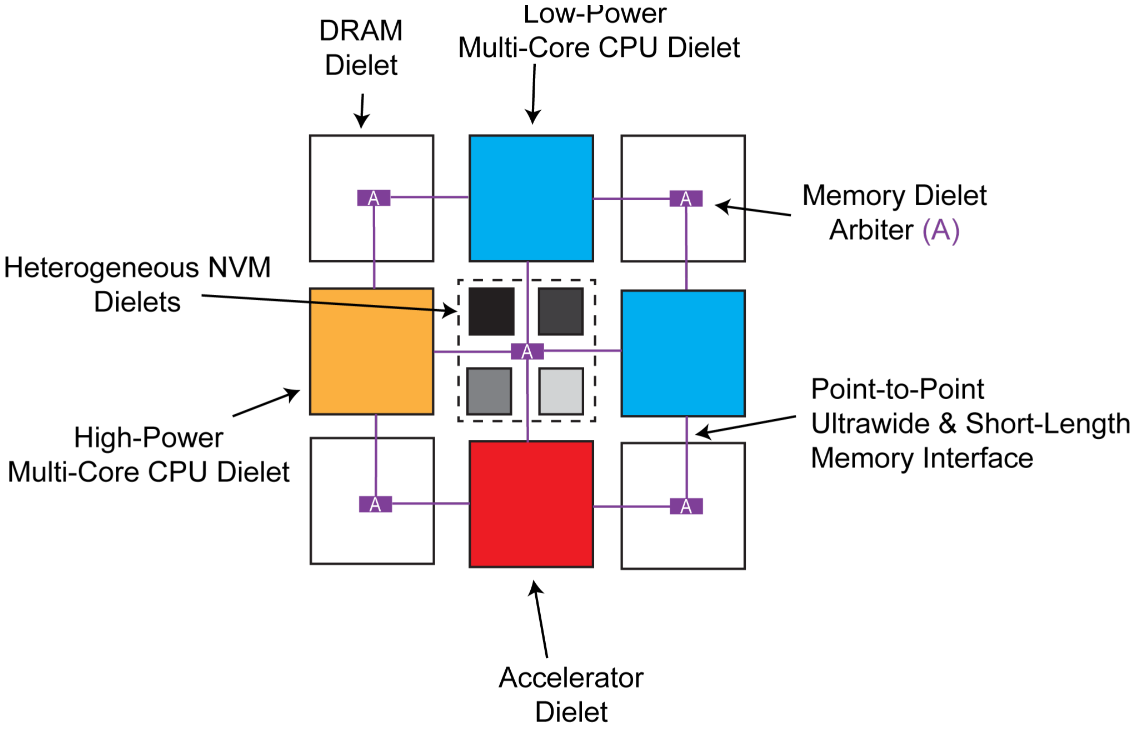
Heterogeneous memory tiles:
1) eDRAM
2) STT-RAM
3) 3D X-Point

Minimized data movement:
- On-chip dense scratchpads
- Local access by physically adjacent compute tiles only
- 4-way arbiter per memory tile
- No cache hierarchy
- No virtual memory
- Thread communication via core-to-core lightweight message passing, control migrations,
locally-shared memory with up to 4 tiles


Checkerboard Architecture
Many-Core, Heterogeneous, Data-Centric
Memory Tile
Low-Power Compute Tile
4-port Memory Arbiter (A)
A
Point-to-Point Memory Interface
High-Power Compute Tile
Accelerator Tile
Energy-optimized sparse mesh NoC for control transfers
High-performance ring-like NoC for I/O
Compute-to-NoC routers

Heterogeneous compute tiles:
1) Performance CPU
2) Low power CPU
2) Accelerators
3) Field-programmable fabric
High performance tiles near edges for I/O accessibility,
thermal footprint

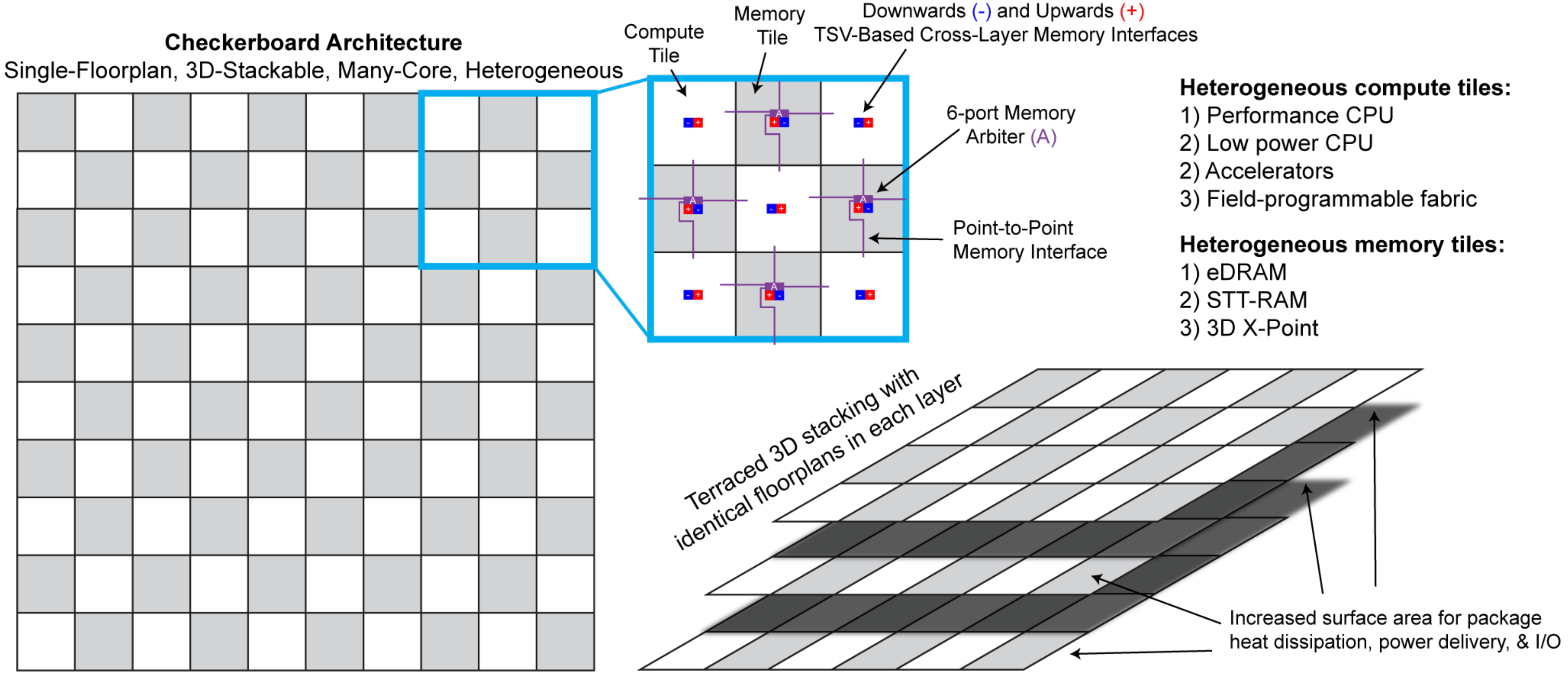
Dielet Example



Dense inter-dielet wiring pitch

- high-BW, low-latency integration of
  disparate technologies
- Simplified dielet I/O
- Highly modular
- Heterogeneous compute and memory
- System-on-wafer
  - Less compromised than SoC
  - Denser, faster, lower energy than PCB

3D Checkerboard Architecture: High-Level


Checkerboard Architecture
Single-Floorplan, 3D-Stackable, Many-Core, Heterogeneous
Compute Tile
Memory Tile
Downwards (-) and Upwards (+) TSV-Based Cross-Layer Memory Interfaces
6-port Memory Arbiter (A)
Point-to-Point Memory Interface
A
+
-
Terraced 3D stacking with identical floorplans in each layer
Increased surface area for package heat dissipation, power delivery, & I/O

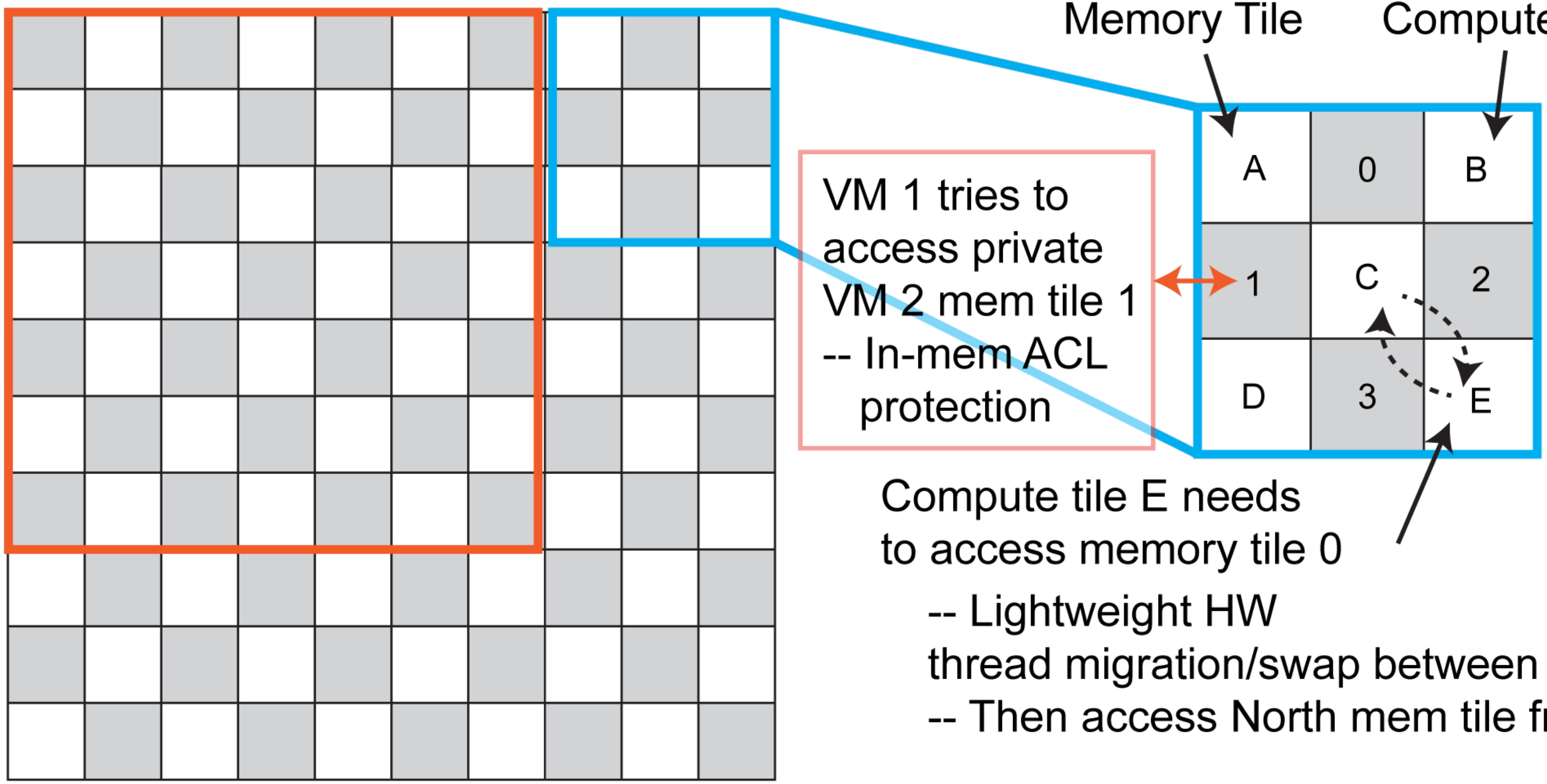
Heterogeneous compute tiles:
1) Performance CPU
2) Low power CPU
2) Accelerators
3) Field-programmable fabric

Heterogeneous memory tiles:
1) eDRAM
2) STT-RAM
3) 3D X-Point

Programming a Checkerboard Architecture

- Overlap-Clustered memory address space
    - Compute tiles: can only address adjacent memory tiles
    - Memory tiles: can only be accessed by adjacent compute tiles
    - *Remote memory access is forbidden*
        - Instead, HW-migrate lightweight threads as needed
    - *No virtual memory*
        - Instead, build relocatable programs – each memory tile has a base address offset
        - In-memory tile access control (e.g., forbid access from West compute tile)
    - *Allocate memory tiles, which come with adjacent compute tiles*
        - …instead of compute threads allocating memory, as is normally done
- System Benefits
    - Clusters have completely independent memory hierarchies
    - Lightweight or eliminated cache coherence – local tiles have shared nearby memory
    - Reduced global communication
    - Many-core scalability for running many task/data-parallel workloads
    - "Datacenter-on-chip" – well suited for isolated multi-core VMs in the cloud

Programming a Checkerboard Architecture: Example



VM 1 mapped to cluster 1

VM 2 mapped to cluster 2

Memory Tile

Compute Tile

VM 1 tries to access private VM 2 mem tile 1
-- In-mem ACL protection

Compute tile E needs to access memory tile 0

-- Lightweight HW thread migration/swap between E & C

-- Then access North mem tile from C