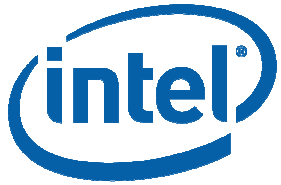
The Mystery of the Non-Linear Increase in Cache SER



April 15, 2009

Shubu Mukherjee

Principal Engineer

Director, SPEARS Group

Intel Massachusetts, Inc.

IEEE Fellow

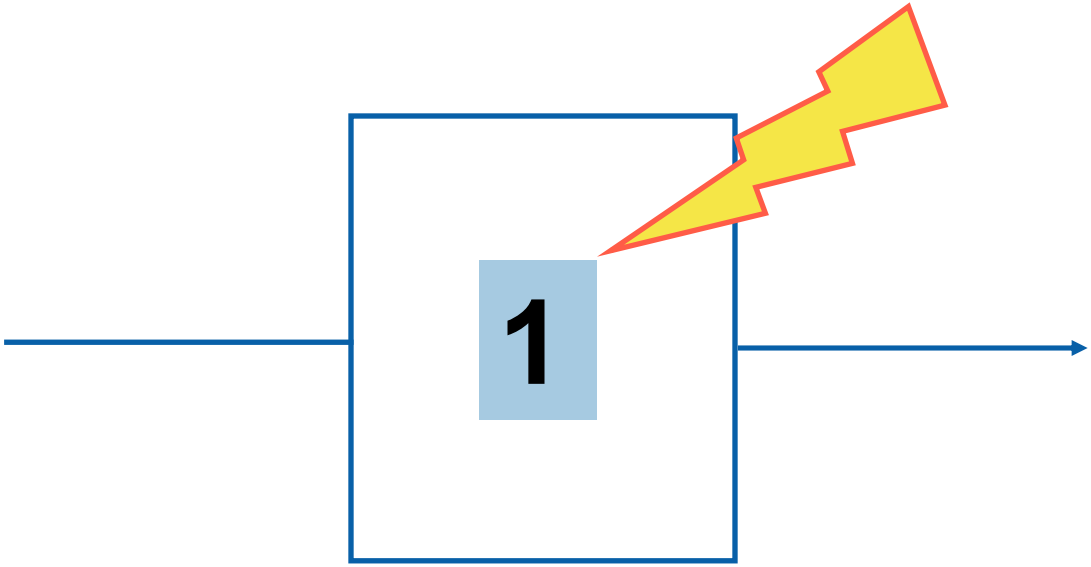
Adjunct Professor at Indian Institute of Technology, Kanpur

Joint work with:

Vinod Ambrose, Arijit Biswas, Leo Chan, Aamer Jaleel, Athanasios Papathanasiou, Mike Plaster, Charlie Recchia, Norbert Seifert

What is a Soft Error?

Neutron or Alpha Particle Strike Changes State of a Single Bit



Error is transient, bit is not permanently damaged.
No known feasible shielding or process technology eliminates this problem completely

Cosmic Ray Strikes: Evidence & Reaction

Publicly disclosed examples

- Error logs in large servers, E. Normand, "Single Event Upset at Ground Level," IEEE Trans. on Nucl Sci, Vol. 43, No. 6, Dec 1996.
- Sun Microsystems found cosmic ray strikes on L2 cache with defective error protection caused Sun's flagship servers to suddenly and mysteriously crash, R. Baumann, IRPS Tutorial on SER, 2000.
- Cypress Semiconductor reported in 2004 a single soft error brought a billion-dollar automotive factory to a halt once a month, Ziegler & Puchner, "SER – History, Trends, and Challenges," Cypress, 2004.

Typical server system data corruption targets 1000 years MTBF

- Very hard to achieve this goal in a cost-effective way
- Bossen, 2002 IRPS Workshop Talk

Server processors beginning to protect latches

- 80% of 200k latches protected with parity in 130 nm Fujitsu SPARC
- Uncore latches radiation-hardened in Intel's 65nm Tukwila processor

Evolution of a Product Team's Psyche

Shock

"SER is the crabgrass in the lawn of computer design"

Denial

"We will do the SER work two months before tapeout"

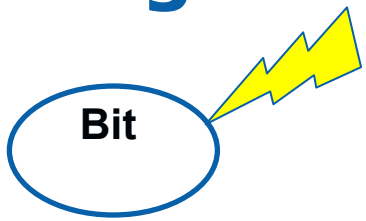
Anger

"Our reliability target is too ambitious"

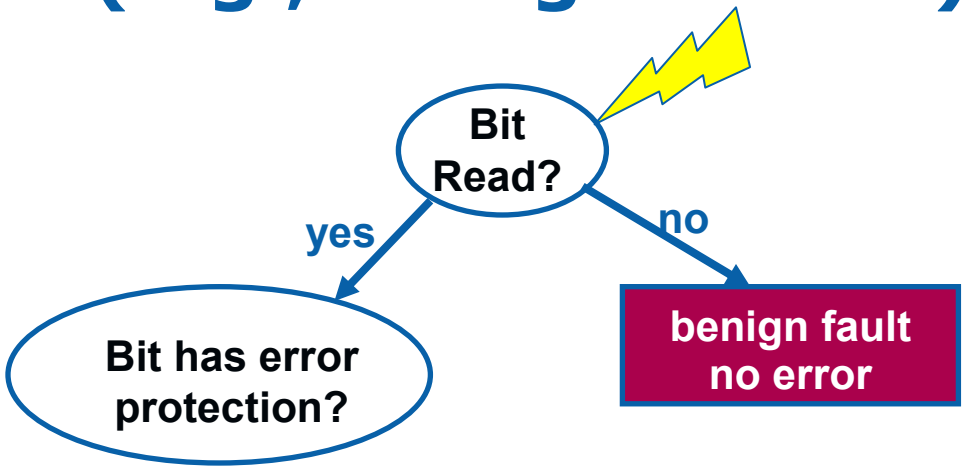
Acceptance

"You can deny physics only for so long"

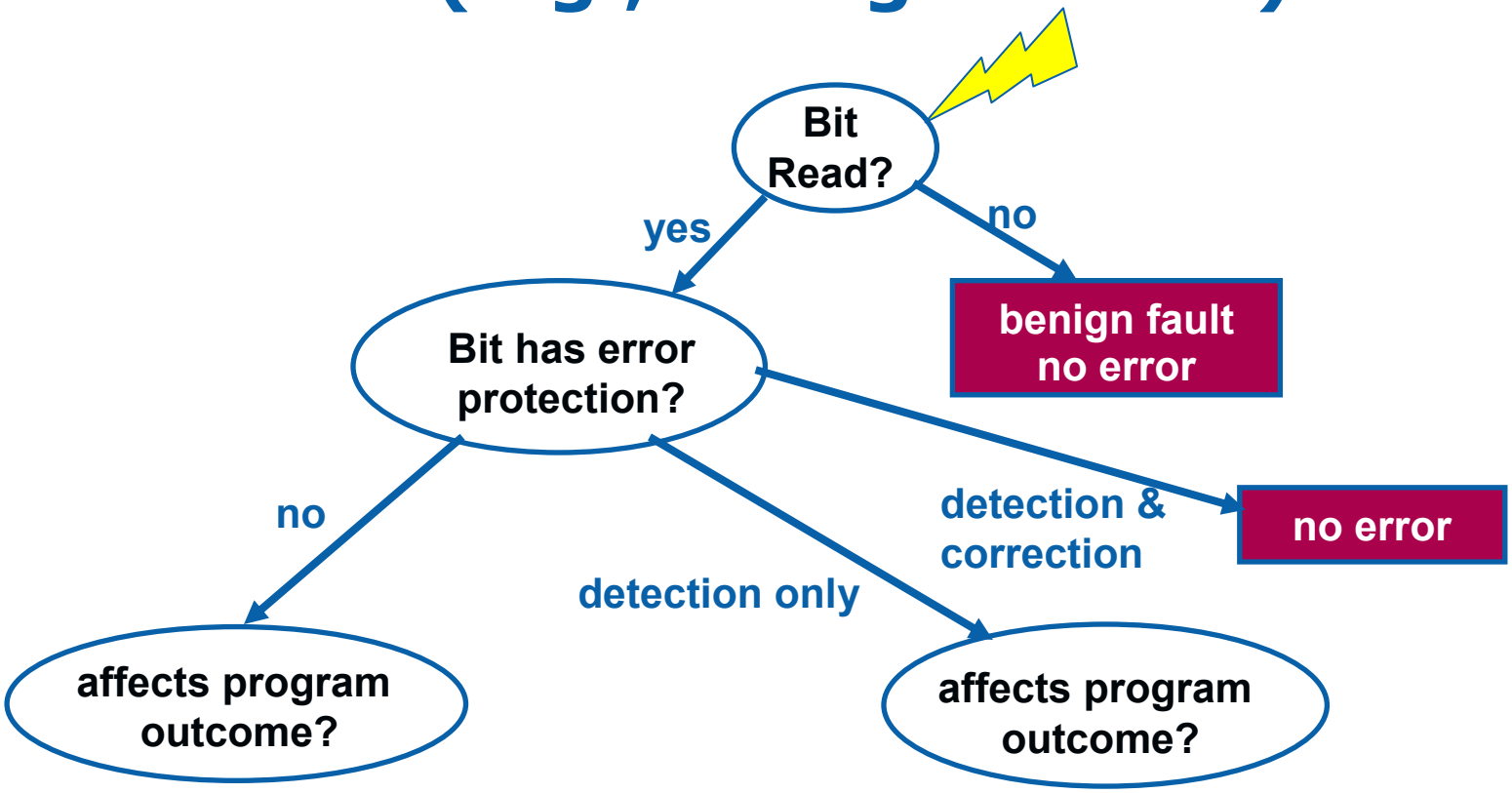
Strike on a bit (e.g., in register file)



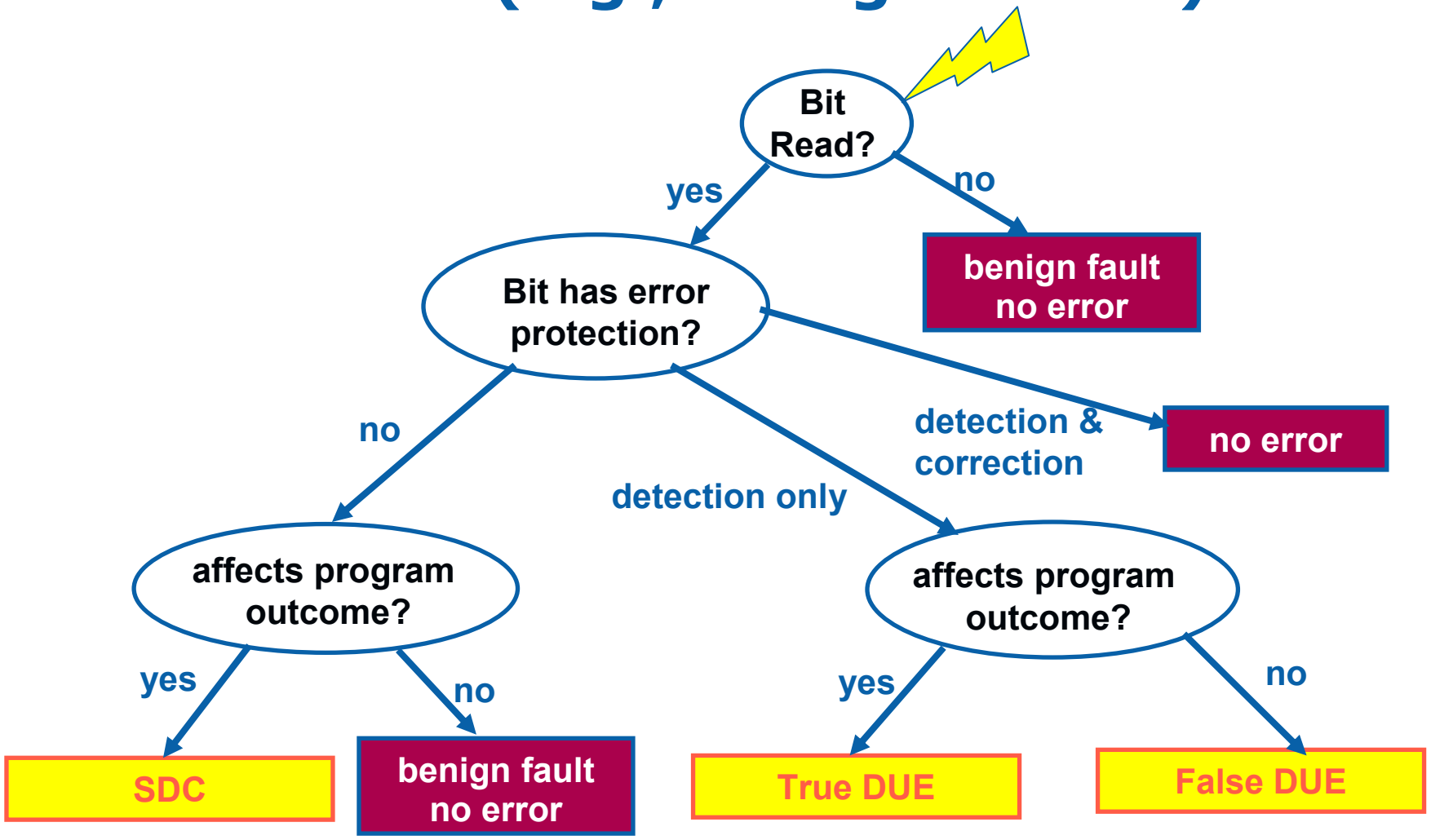
Strike on a bit (e.g., in register file)



Strike on a bit (e.g., in register file)



Strike on a bit (e.g., in register file)



SDC = Silent Data Corruption, DUE = Detected Unrecoverable Error

The Mystery:
Cache SER Increases Non-Linearly with Cache Size

Observations in the field showed a higher than expected rate of both correctable ECC events on the L1 Cache Data and uncorrectable Parity events on the L1 Cache Tags

- **Parity events are the biggest concern since these cause system halts (DUE)**

Intuition based on average behavior indicated cache error rate increases linearly with cache size (2x increase in SER for a 2x increase in cache size)

- **However, empirical data indicated a 5x to 10x error rate increase as cache size doubled across a variety of CPUs.**

The Investigation

The Detectives: Vinod Ambrose, Arijit Biswas, Leo Chan, Aamer Jaleel, Shubu Mukherjee, Athanasios Papathanasiou, Mike Plaster, Charlie Recchia, Norbert Seifert

- Suspects
- Clues & Leads
- Re-enacting the Crime
- The Smoking Gun
- Putting it all Together
- Book ‘em Dan-o
- Conclusions

Suspects – Potential Causes of the SER Increase

Electrical Issues

- Load line / power supply issue
- Circuit-level issue related to increased RC on word/bit lines

Design Issues

- Differences in SRAM cell design
- Differences in SRAM cell layout / decap density

Manufacturing Issues

- Process technology effects

Logical Issues

- Workload-based AVF effects
- AVF = probability a bit flip results in user-visible error

The Investigation

- Suspects
- Clues & Leads
- Re-enacting the Crime
- The Smoking Gun
- Putting it all Together
- Book ‘em Dan-o
- Conclusions

Clues & Leads – The Shakedown

Design sources check on design / layout issues

- SRAM cell design and layout identical between small and large cache parts

Manufacturing sources check on process issues

- No process tweaks or optical shrinks between small and large cache parts

Circuit design sources check parasitic RC issues

- SRAM cell SER insulated from parasitic RC effects of longer bit/word lines
- Parasitic RC on larger cache is not an issue

→Sources eliminate circuit, design, layout & process

Clues & Leads – "Elementary my dear Watson"

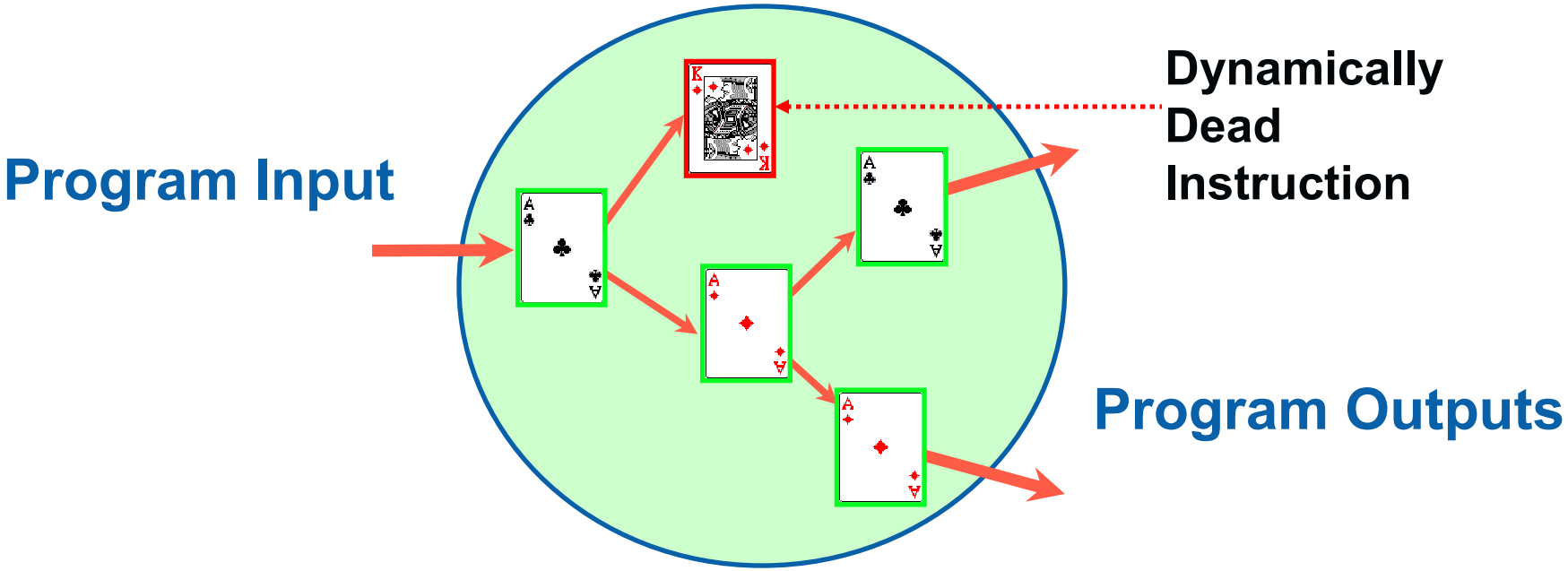
"...when you have eliminated the impossible, whatever remains, however improbable, must be the truth "
–Sir Arthur Conan Doyle

→Logical issues is the only remaining suspect, but how do we prove that workload-related AVF effects are the culprit?

Recall: AVF = probability a bit flip results in user-visible error

AVF = probability that a bit is necessary for Architecturally Correct Execution (ACE)

Architecturally Correct Execution (ACE)



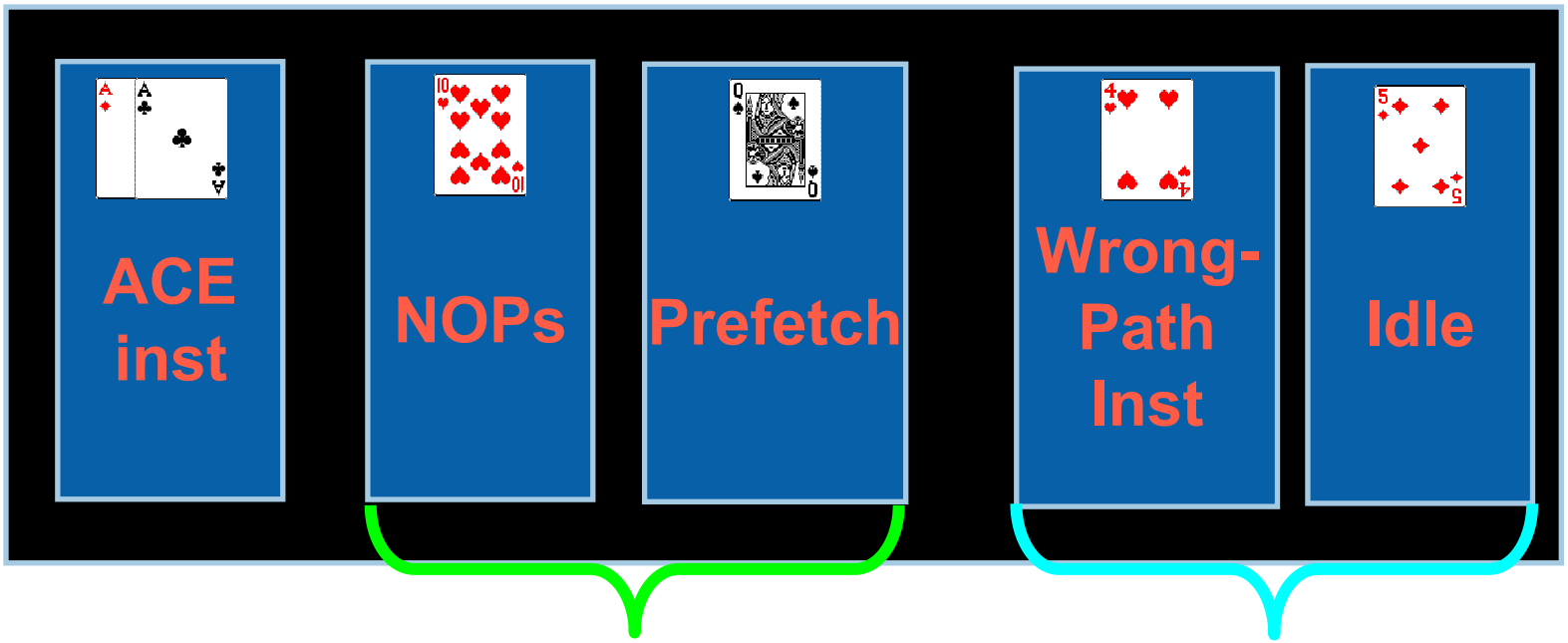
ACE path requires only a subset of values to flow correctly through the program's data flow graph (and the machine)

Anything else (un-ACE path) can be derated away

Most bits of an un-ACE instruction do not affect program output

Mapping ACE & un-ACE Instructions to the Instruction Queue

Instruction Queue Data Array



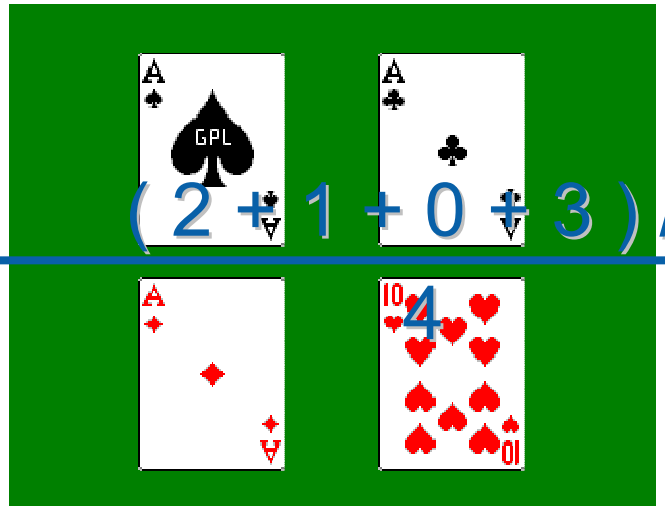
AVF = fraction of cycles a bit contains ACE state
= fraction of ACE bits in IQ in an average cycle

SER & AVF are properties of a bit so ACEness must be mapped to the bit level

intel

Computing AVF of a Structure

AVF = fraction of cycles a bit contains ACE state



$$= \frac{\text{Average number of ACE bits in a cycle}}{\text{Total number of bits in the structure}}$$

The Hypothesis

Similar SER increase seen on specific CPUs for correctable & uncorrectable cache errors

- **Correctable errors come from cache data (ECC-protected)**
- **Uncorrectable errors come from cache tags/status (parity-protected)**
  - **Writeback cache Tags are only ACE for dirty cache lines (clean lines are correctable by invalidate-refetch)**

Hypothesis: As the cache size increases, average residency of cache lines increase

- **Average # of ACE bits in a cycle increase as a result**
- **For tags, only dirty line residency is ACE**

One obvious indicator of increased cache residency is the # of cache misses

Why can Cache Misses be an Indicator of AVF?

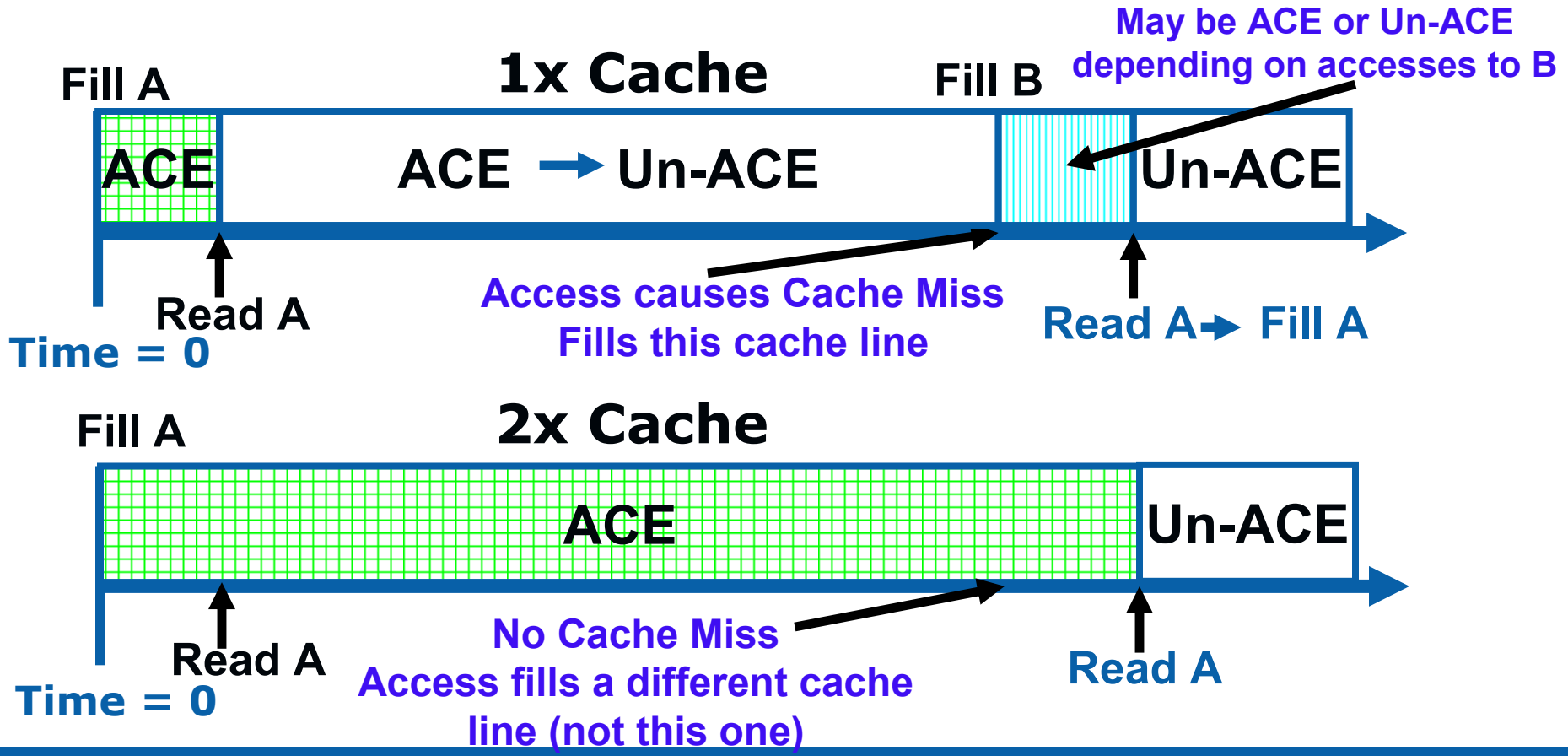
→ AVF increases can be inversely related to # of cache misses

- # of cache misses as a function of cache size is a property of the workload's memory footprint
- As the workload's data and instruction set fit into the cache, number of misses will decrease significantly, increasing the AVF by reducing the unACE time
- This workload-based cache miss phenomenon occurs independent of underlying architecture
- We observed this phenomenon in our AVF computation work for address-based structures for the Data TLB and L1 cache
  - Observed worst case increases of ~100x

Example of Cache Misses Affecting Cache Data AVF

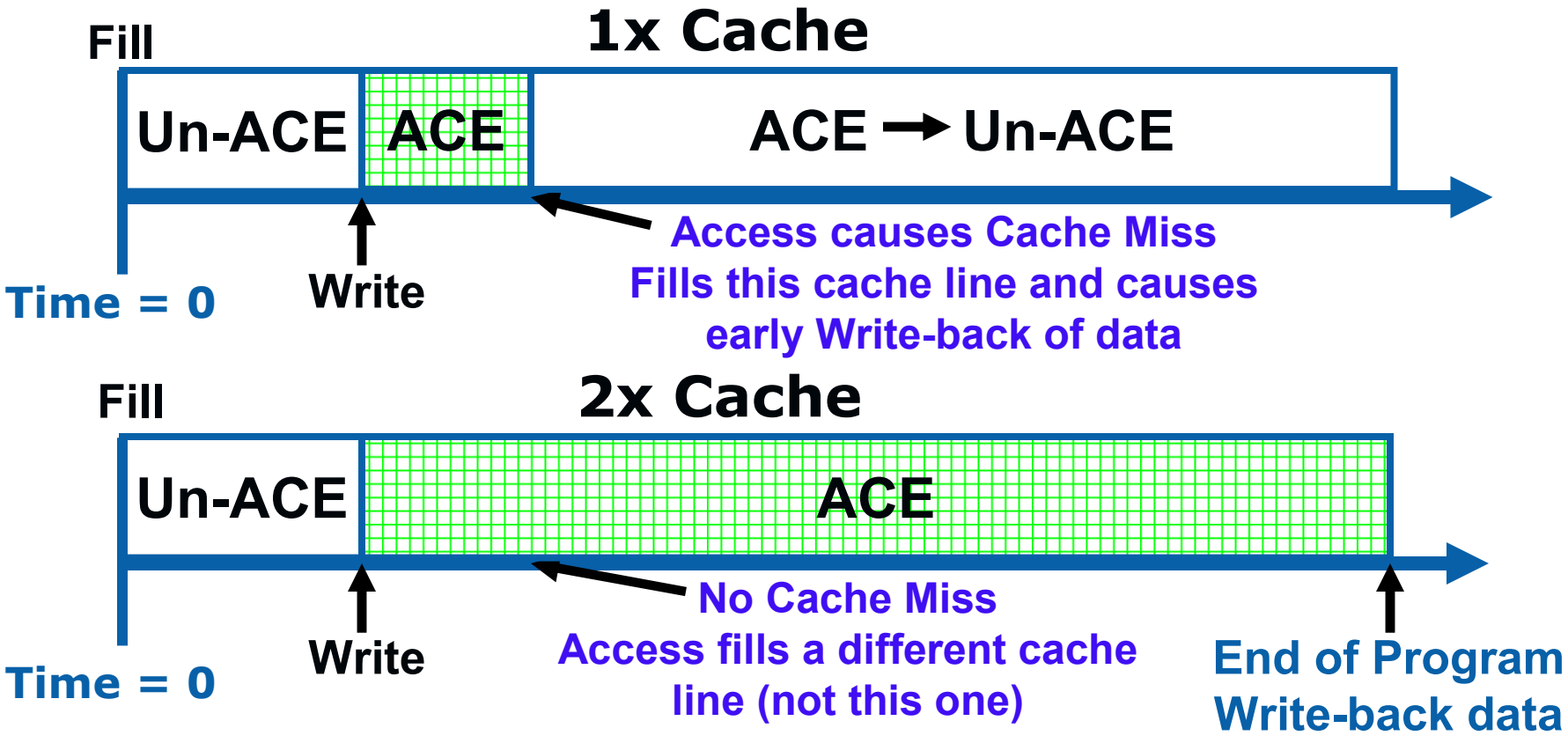
Main concept:

Time from last ACE Read to next Fill (post-eviction) is unACE



Example of Cache Misses Affecting Cache Tag AVF

Main concept:
Time from first Write to Evict (Write-back) is ACE



The Investigation

- Suspects
- Clues & Leads
- Re-enacting the Crime
- The Smoking Gun
- Putting it all Together
- Book ‘em Dan-o
- Conclusions

How do we prove our hypothesis?

Choosing the tools and methodology

Proton beam experiment is conducted to re-create the workload effect

- Target systems are irradiated by accelerated proton beam
- Target workloads are running on target systems while they are irradiated
- Incidence of Detected Unrecoverable Errors (DUE) are counted

Choosing the systems and benchmarks

Systems

(Chose 2 CPUs with the exact same core design but different size cache sizes: 1 MB and 2 MB size L2 caches)

- Target system 1: Processor 1X (1 MB)
- Target system 2: Processor 2x (2 MB)

Benchmarks

- Target workload 1: art (Spec2000)
- Target workload 2: libquantum (Spec2006)
- Target workload 3: swim (Spec2000)

Workload Cache Miss Profiles

3 workloads were specifically chosen to test the cache miss hypothesis

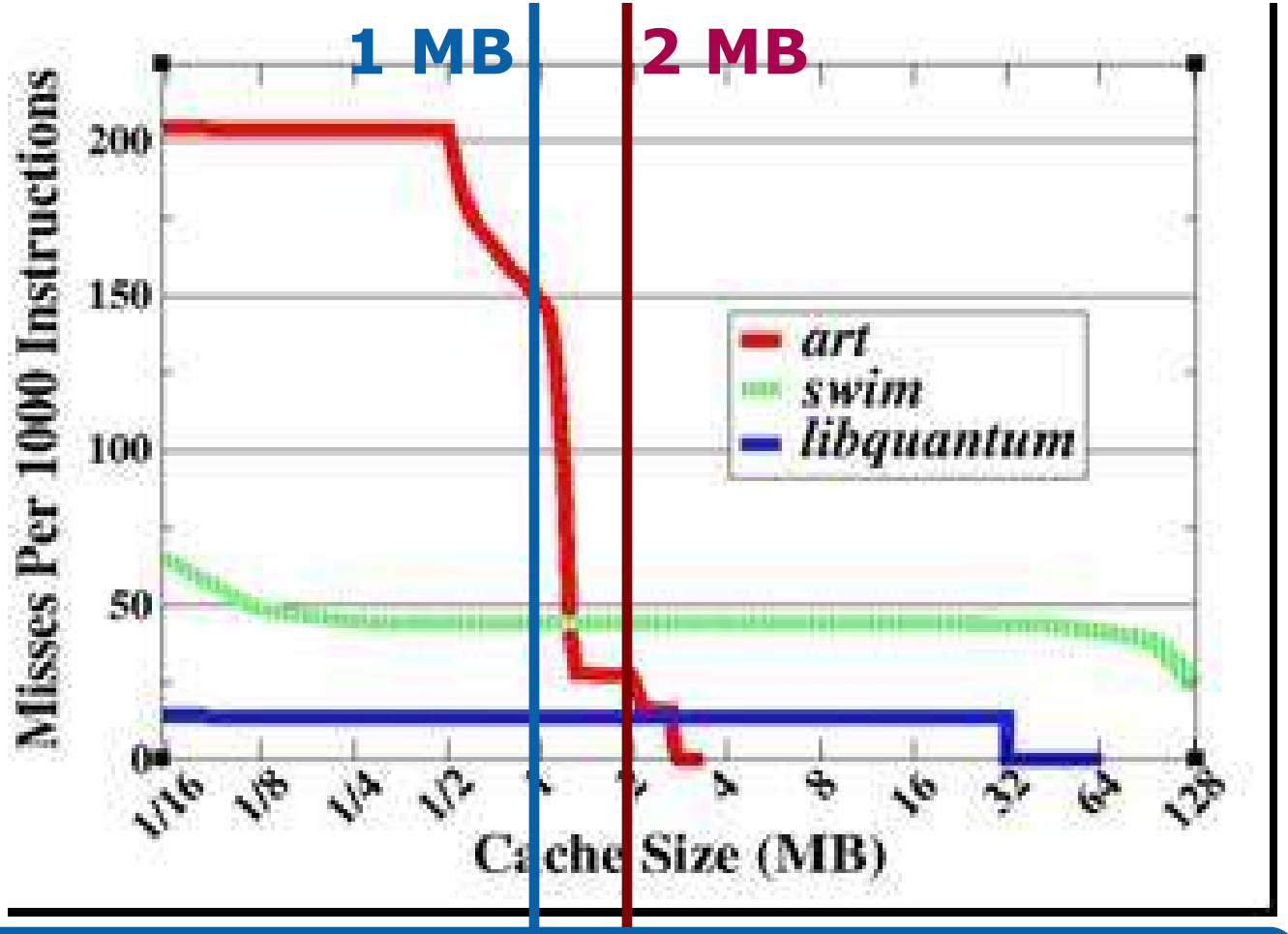
art
Spec2000

Has large change change in #misses between 1 and 2 MB

Libquantum
Spec 2006

Swim
Spec2000

Have no change in #misses between 1 and 2 MB



Expect art to show a large increase in AVF
Expect libquantum & swim may show little to no change in AVF

The Investigation

- Suspects
- Clues & Leads
- Re-enacting the Crime
- The Smoking Gun
- Putting it all Together
- Book ‘em Dan-o
- Conclusions

The Smoking Gun

Benchmark	DUE SER Ratio: (Processor 2x / Processor 1x)
Art	4.25
Swim	3.87
Libquantum	1.2

- Art & Swim show a ~4x increase in SER
- Libquantum shows negligibe increase in SER

AVF can cause a non-linear increase in SER
All failures are DUE so we focus on Tags

Putting it all Together

Tools of the Trade

- **VTUNE™ software used to verify modeled cache miss behavior**
  - VTUNE provides a SW interface to HW performance counters
- **AVF simulation studies**
  - L1 cache AVF studies indicate that up to 100x SER increase for a 2x cache size increase is possible
  - Indicated possible scenarios for non-linear AVF increase
- **Cache study simulations**
  - Provided architecture-independent cache access statistics for the 3 benchmarks

Explaining the Data: Art

Art showed a ~4X SER **increase** from 1X to 2X

From 1X to 2X:
Dirty Reads **increased**
Misses **decreased**
Writebacks **decreased**

These are all indications of significant **increases** in the residency time of dirty lines as cache size increases

Vtune™ Simulation	**Ratio: 2X / 1X**
MPKI	1/6
Dirty Reads	1.56
Write Misses	1/4
Read Misses	1/3
Write-backs	1/33

Measurement results for art show the expected SER increase

Explaining the Data: Libquantum & Swim

Swim showed a ~4X SER **increase** from 1X to 2X

Libquantum showed a negligible difference in SER

From 1X to 2X:
Dirty Reads are **similar**
Misses are **similar**
Writebacks are **similar**

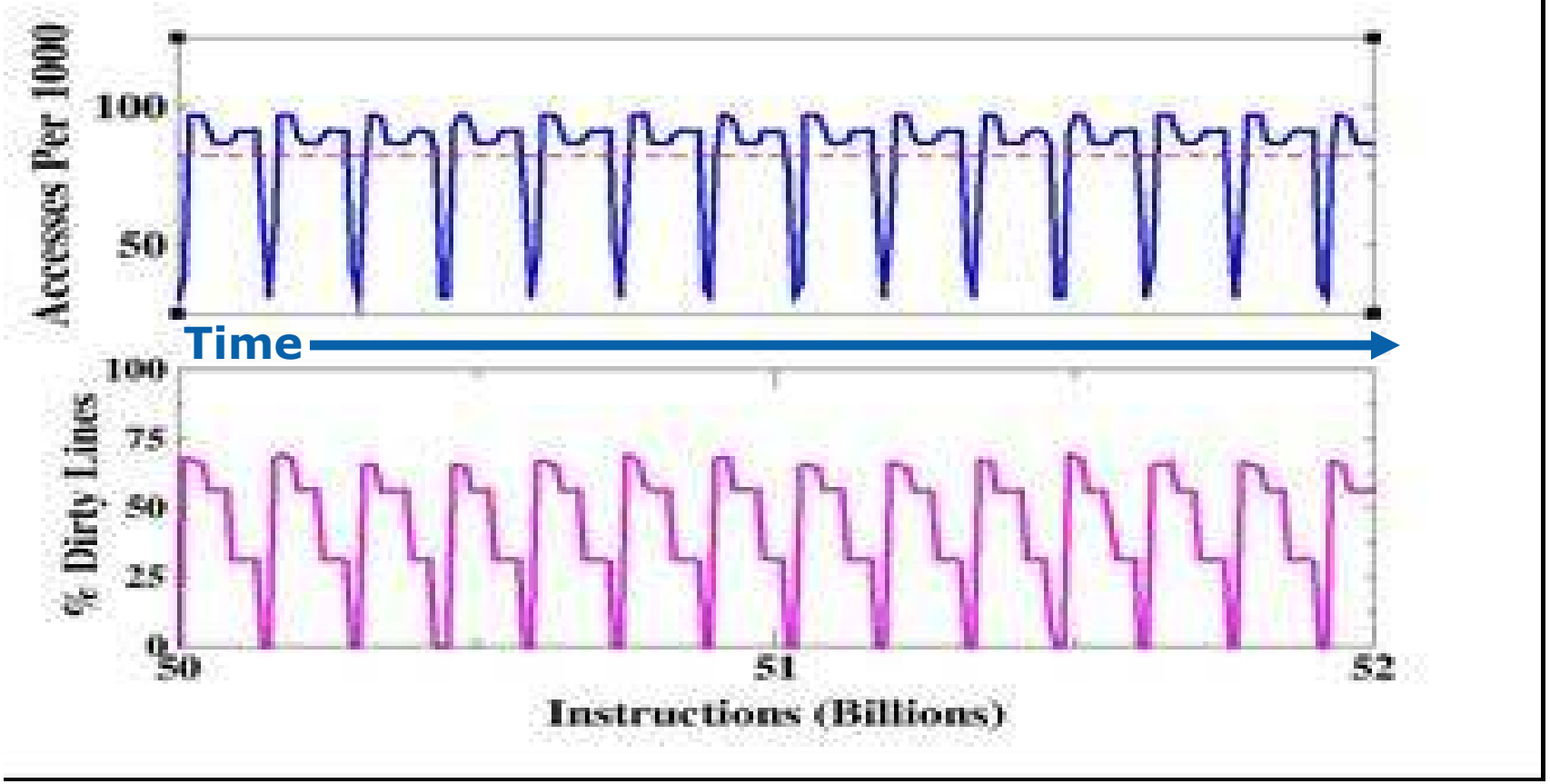
Vtune™ Simulation	**Swim Ratio: 2X / 1X**	**Libquantum Ratio: 2X / 1X**
MPKI	1	1
Dirty Reads	0.98	1.06
Write Misses	1	1
Read Misses	1/1.16	1/1.14
Write-backs	1.00	1.01

So why do we see an SER increase in Swim but not Libquantum?
Both were expected to show no increase

Swim: Strided Cache Access Patterns

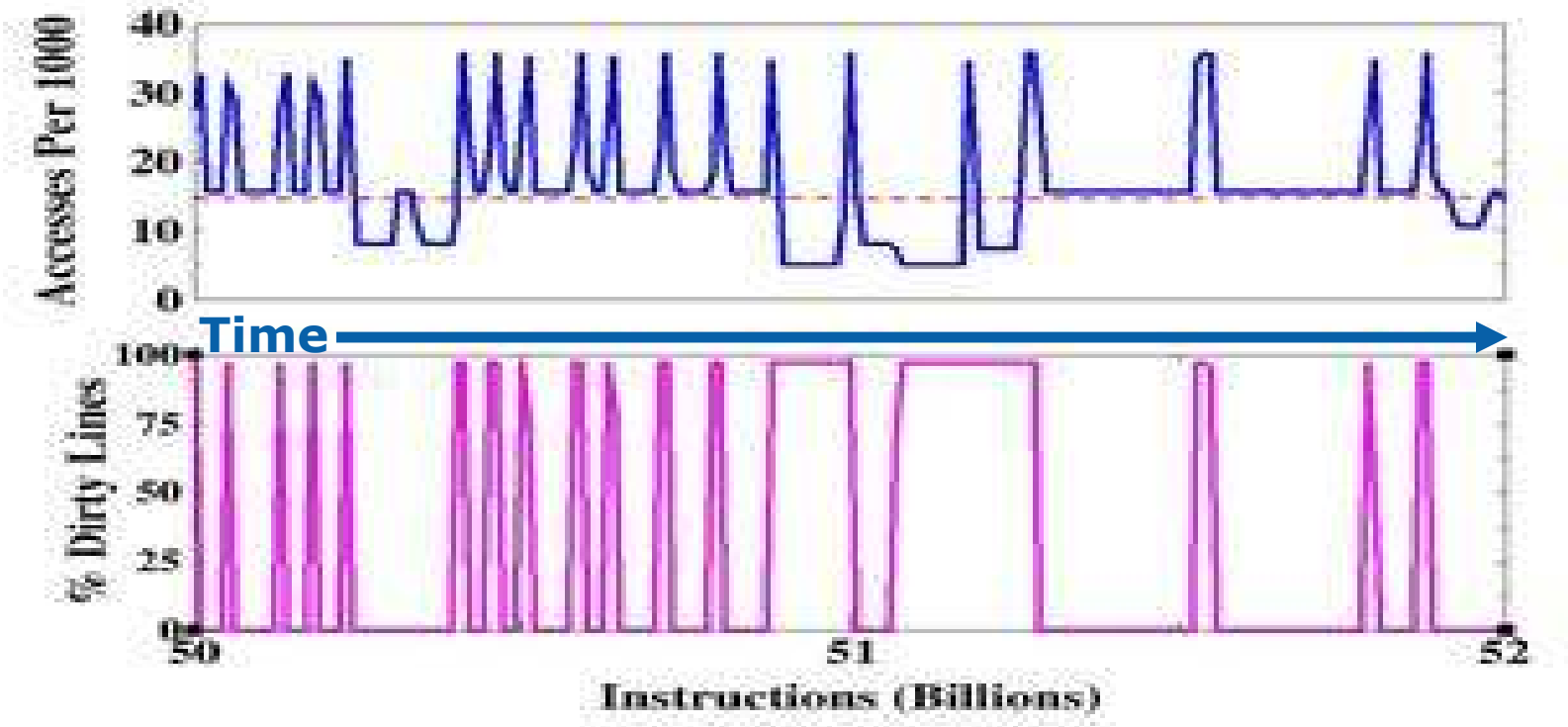
Swim has a higher AVF due to strided access pattern

- Strided access patterns "step" through memory



Libquantum: Random Cache Access Patterns

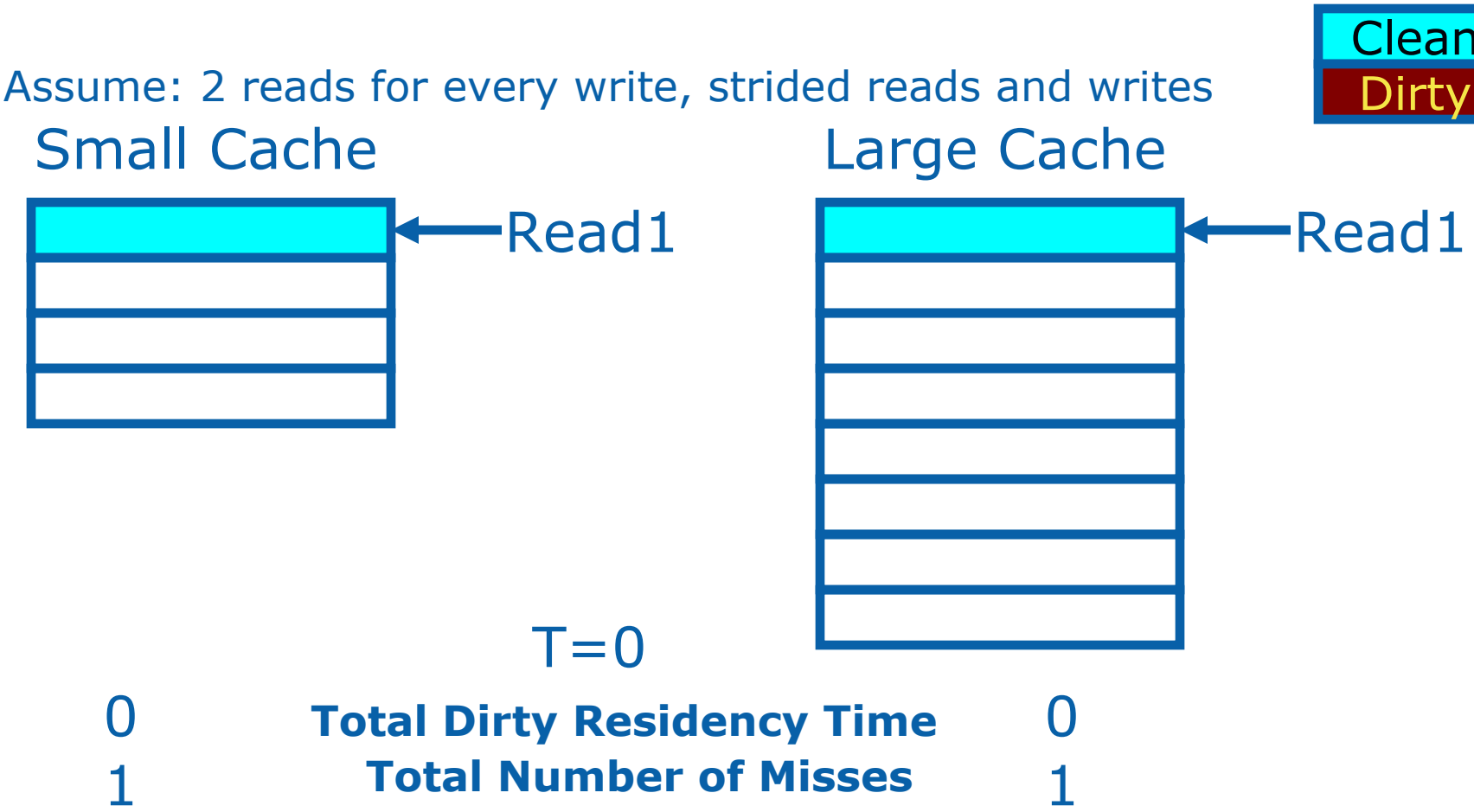
Libquantum has a more random access pattern & no cache locality (99% miss rate for 1MB & 2MB)



Strided Cache Access Example

Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

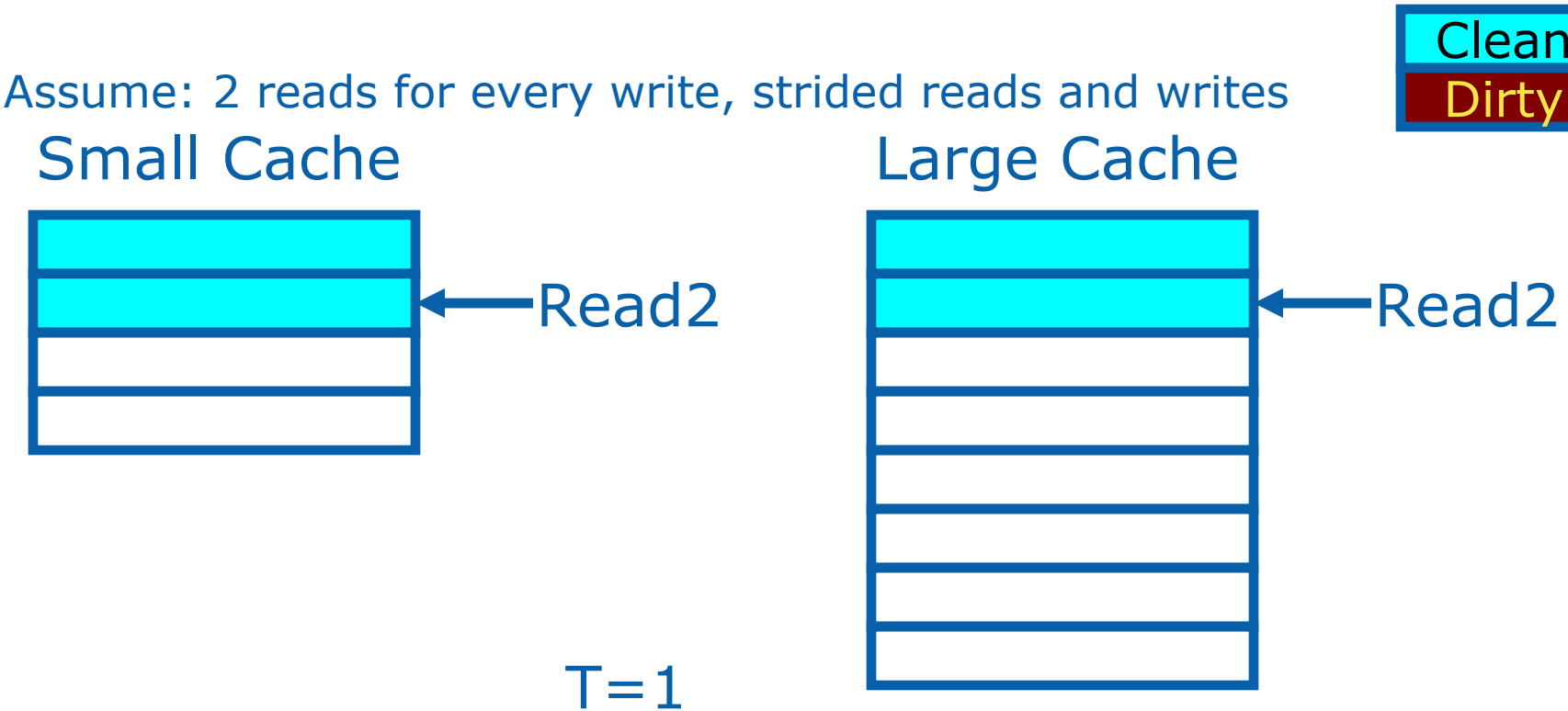
Assume: 2 reads for every write, strided reads and writes



Strided Cache Access Example

Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

Assume: 2 reads for every write, strided reads and writes

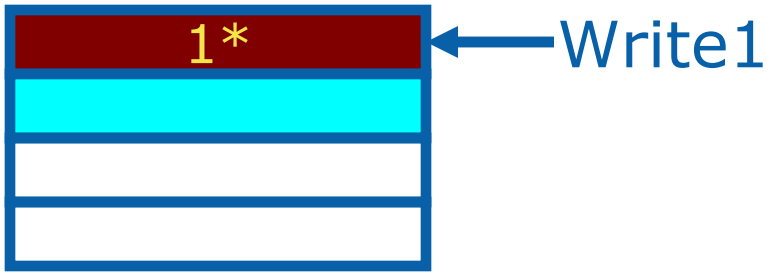


Strided Cache Access Example

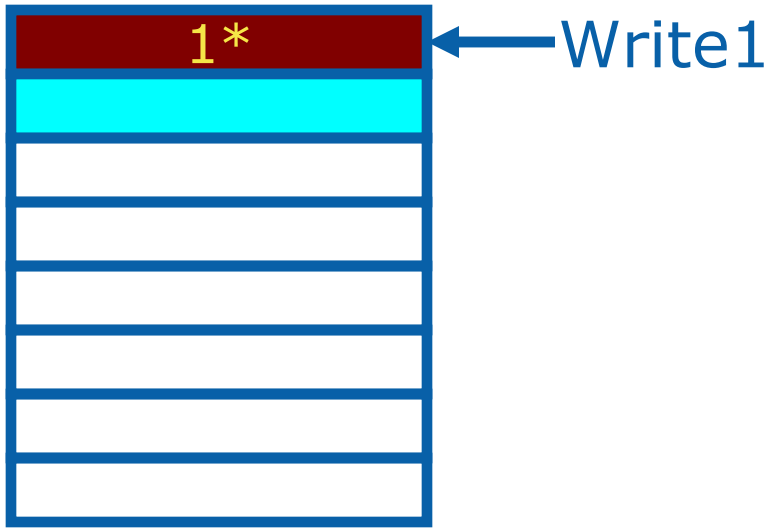
Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

Assume: 2 reads for every write, strided reads and writes

Small Cache



Large Cache



* Numbers represent cumulative dirty cycles

T=2

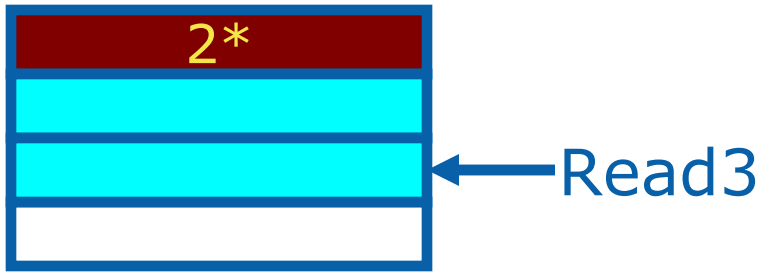
$$\frac{1}{2} \quad \textbf{Total Dirty Residency Time} \quad \textbf{Total Number of Misses} \quad \frac{1}{2}$$

Strided Cache Access Example

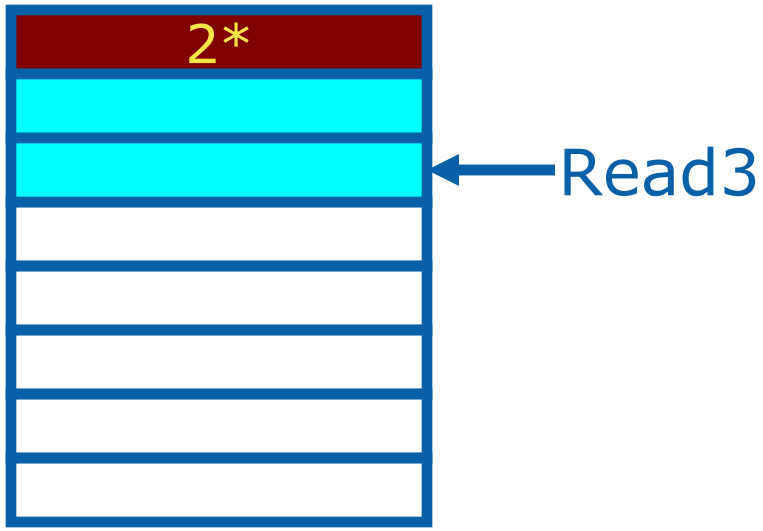
Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

Assume: 2 reads for every write, strided reads and writes

Small Cache



Large Cache



* Numbers represent cumulative dirty cycles

T=3

2 **Total Dirty Residency Time** 2

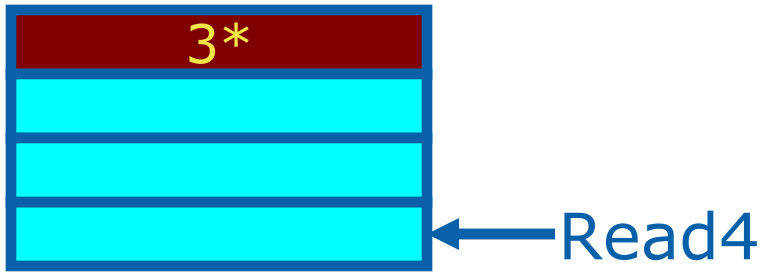
3 **Total Number of Misses** 3

Strided Cache Access Example

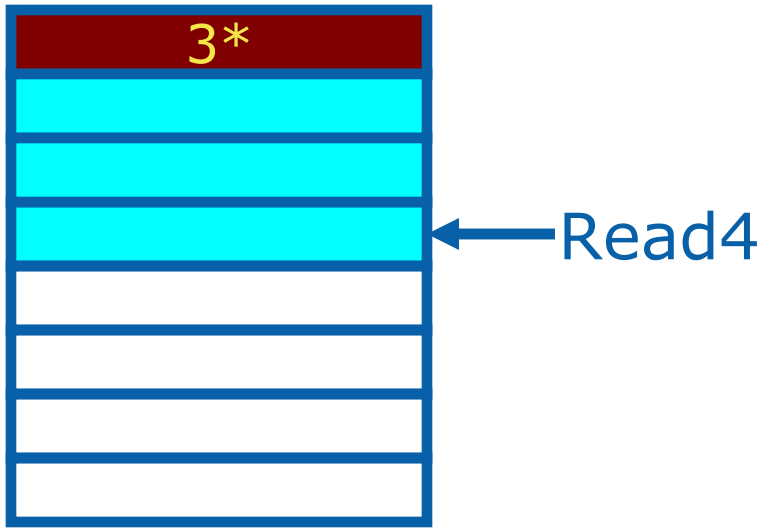
Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

Assume: 2 reads for every write, strided reads and writes

Small Cache



Large Cache



* Numbers represent cumulative dirty cycles

T=4

3 **Total Dirty Residency Time** 3

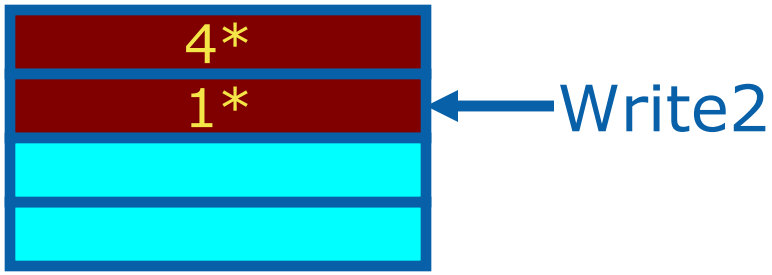
4 **Total Number of Misses** 4

Strided Cache Access Example

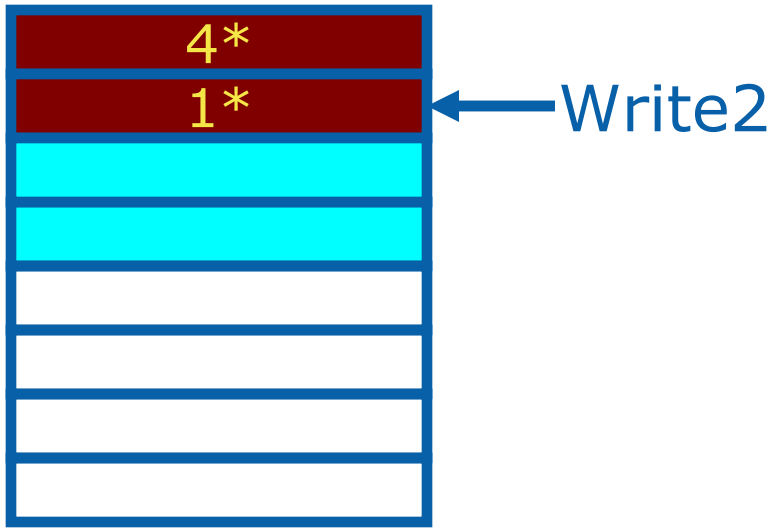
Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

Assume: 2 reads for every write, strided reads and writes

Small Cache



Large Cache



* Numbers represent cumulative dirty cycles

T=5

5 **Total Dirty Residency Time** 5

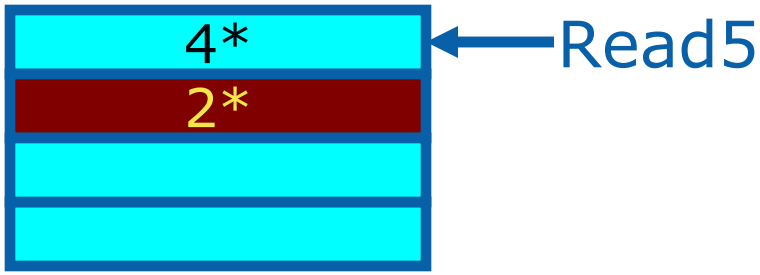
4 **Total Number of Misses** 4

Strided Cache Access Example

Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

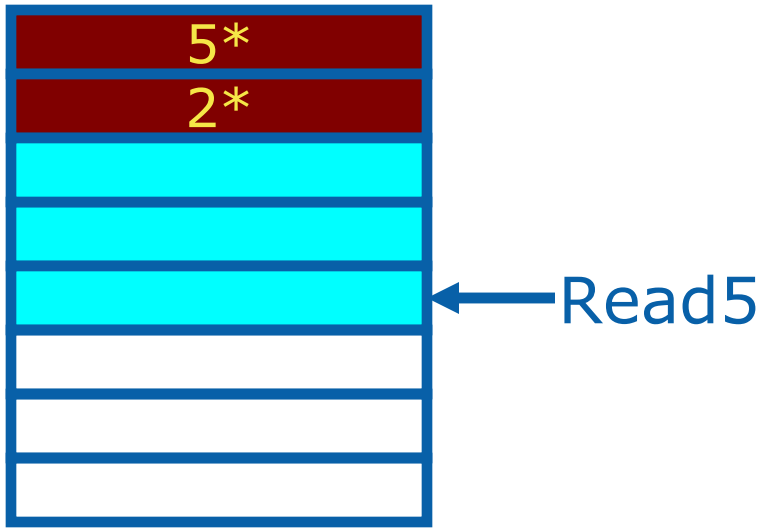
Assume: 2 reads for every write, strided reads and writes

Small Cache



Read5 wraps around and evicts address 1

Large Cache



* Numbers represent cumulative dirty cycles

T=6

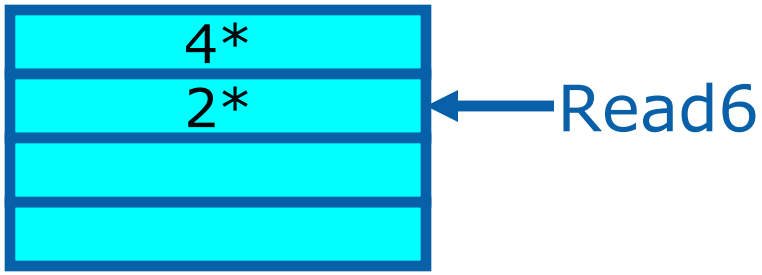
6	**Total Dirty Residency Time**	7
5	**Total Number of Misses**	5

Strided Cache Access Example

Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

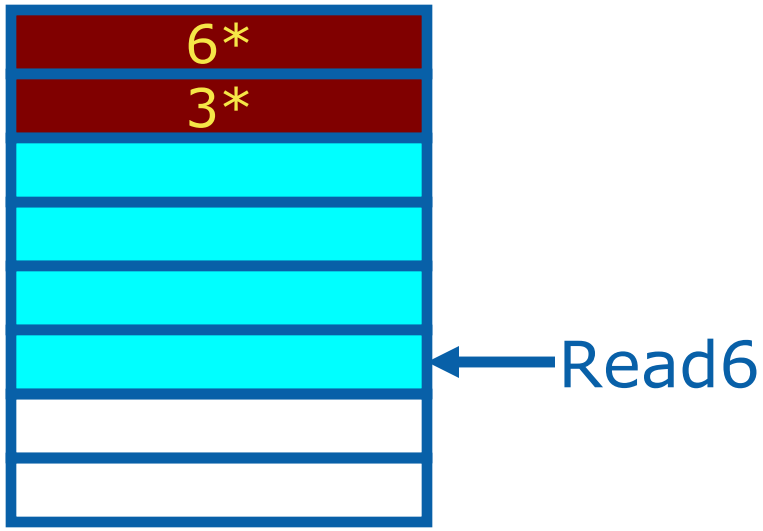
Assume: 2 reads for every write, strided reads and writes

Small Cache



Read6 wraps around and evicts address 2

Large Cache



* Numbers represent cumulative dirty cycles

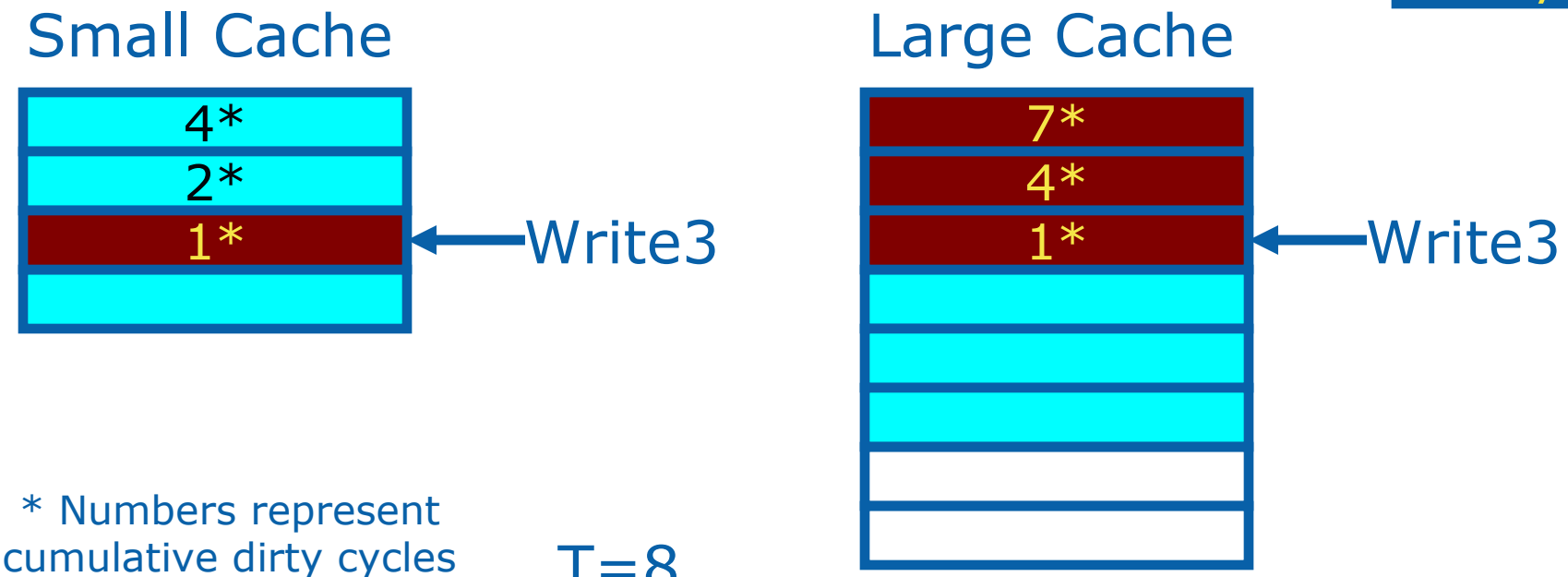
T=7

6	**Total Dirty Residency Time**	9
6	**Total Number of Misses**	6

Strided Cache Access Example

Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

Assume: 2 reads for every write, strided reads and writes



* Numbers represent cumulative dirty cycles

T=8

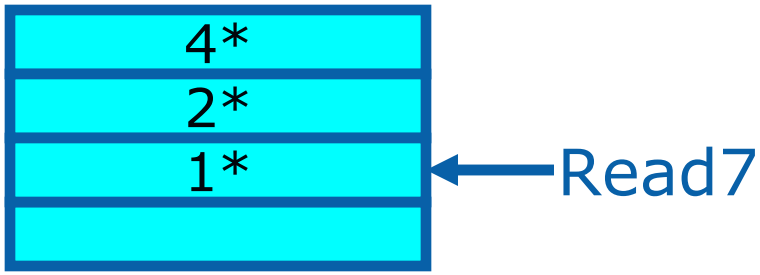
7	**Total Dirty Residency Time**	12
6	**Total Number of Misses**	6

Strided Cache Access Example

Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

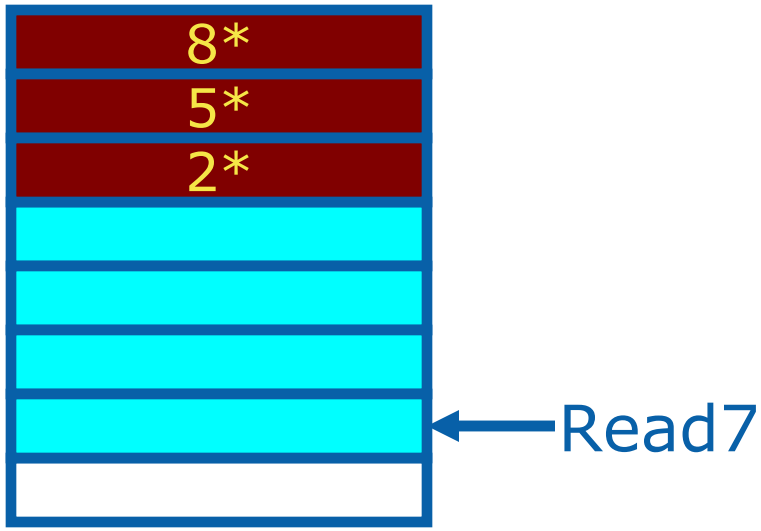
Assume: 2 reads for every write, strided reads and writes

Small Cache



Read7 wraps around and evicts address 3

Large Cache



* Numbers represent cumulative dirty cycles

T=9

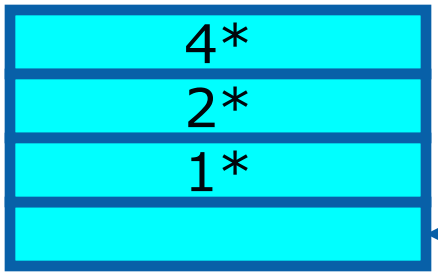
7	**Total Dirty Residency Time**	15
7	**Total Number of Misses**	7

Strided Cache Access Example

Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

Assume: 2 reads for every write, strided reads and writes

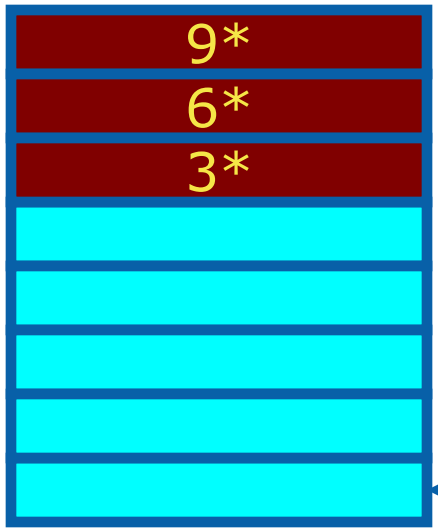
Small Cache



Read8

Read8 wraps around and evicts address 4

Large Cache



Read8

* Numbers represent cumulative dirty cycles

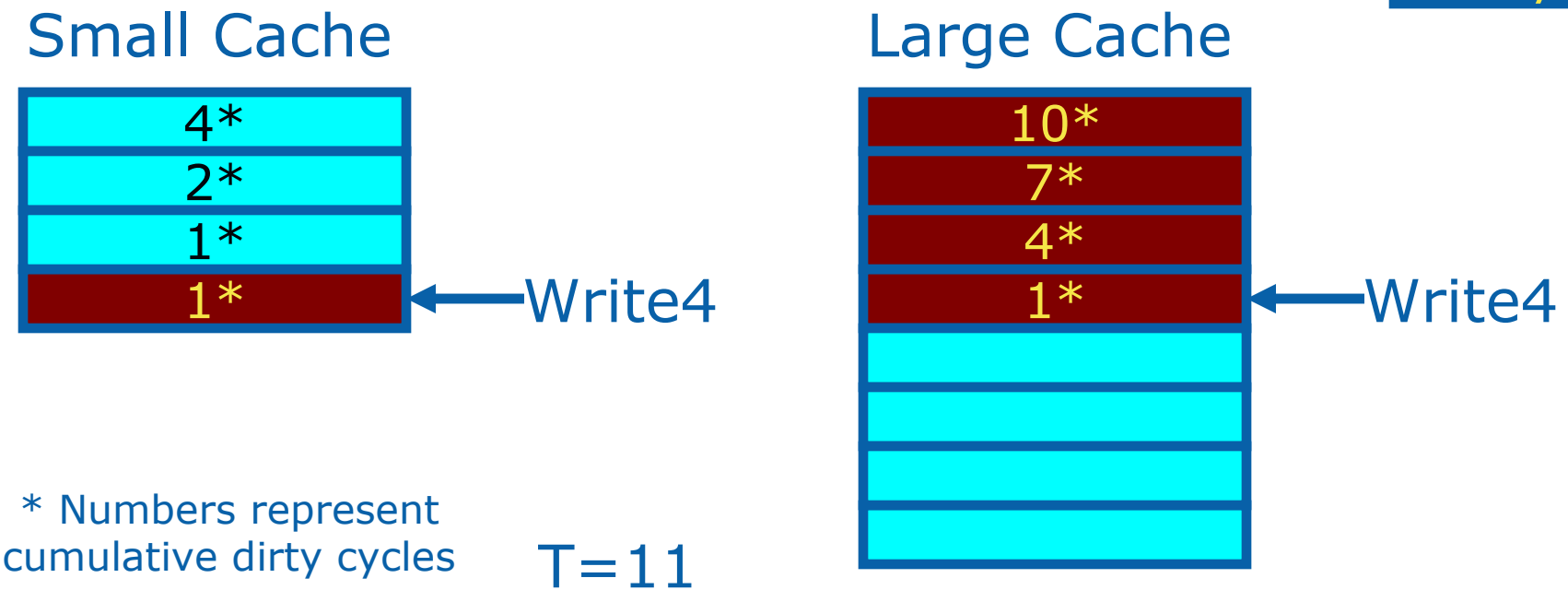
T=10

7	**Total Dirty Residency Time**	18
8	**Total Number of Misses**	8

Strided Cache Access Example

Strided cache access pattern can increase dirty residency times with cache size while keeping event counts and ratios the same

Assume: 2 reads for every write, strided reads and writes



* Numbers represent cumulative dirty cycles

T=11

8	**Total Dirty Residency Time**	22
10	**Total Number of Misses**	9

Cache Stride Example

Cache stride pattern caused a ~3x increase in dirty residency

- No dirty reads in either case
- Same # of writebacks since all dirty lines will be written back at the end of the program
- # of misses are roughly the same
- Different Read/Write ratios and strides will affect the rate of SER increase
  - Swim has a 4:1 ratio of loads:stores

The Investigation

- Suspects
- Clues & Leads
- Re-enacting the Crime
- The Smoking Gun
- Putting it all Together
- Book ‘em Dan-o
- Conclusions

Book 'em Dan-o

Vtune™ Simulation Proton Beam	Art Ratio: 2X / 1X	Swim Ratio: 2X / 1X	Libquantum Ratio: 2X / 1X
MPTF	4.85	3.87	1.2
MPKI	1/6	1	1
Dirty Reads	1.56	0.98	1.06
Write Misses	1/4	1	1
Read Misses	1/3	1/1.16	1/1.14
Write-backs	1/33	1.00	1.01

Art sees a ~4x increase in SER due to dirty residency time increases brought on by lower miss rates

Swim sees a ~4x increase in SER due to dirty residency time increases brought on by a strided access pattern

Conclusions

SER can scale non-linearly due to AVF

- Driven by workload-based cache behavior
- Tag AVF driven by residency time of dirty lines
  - Decreasing miss rates (Art)
  - Strided access patterns (Swim)

What can we do to account for this behavior?

- Recognize and accept that cache AVF behavior is based on residency times of valid lines (dirty lines for tag AVF)
- Provide recovery (correction) of cache errors (e.g. ECC) thereby reducing cache tag DUE to zero
- Increased SER seen on 2X processor was still within spec. Rather, the SER seen on the 1X processor was significantly under spec.