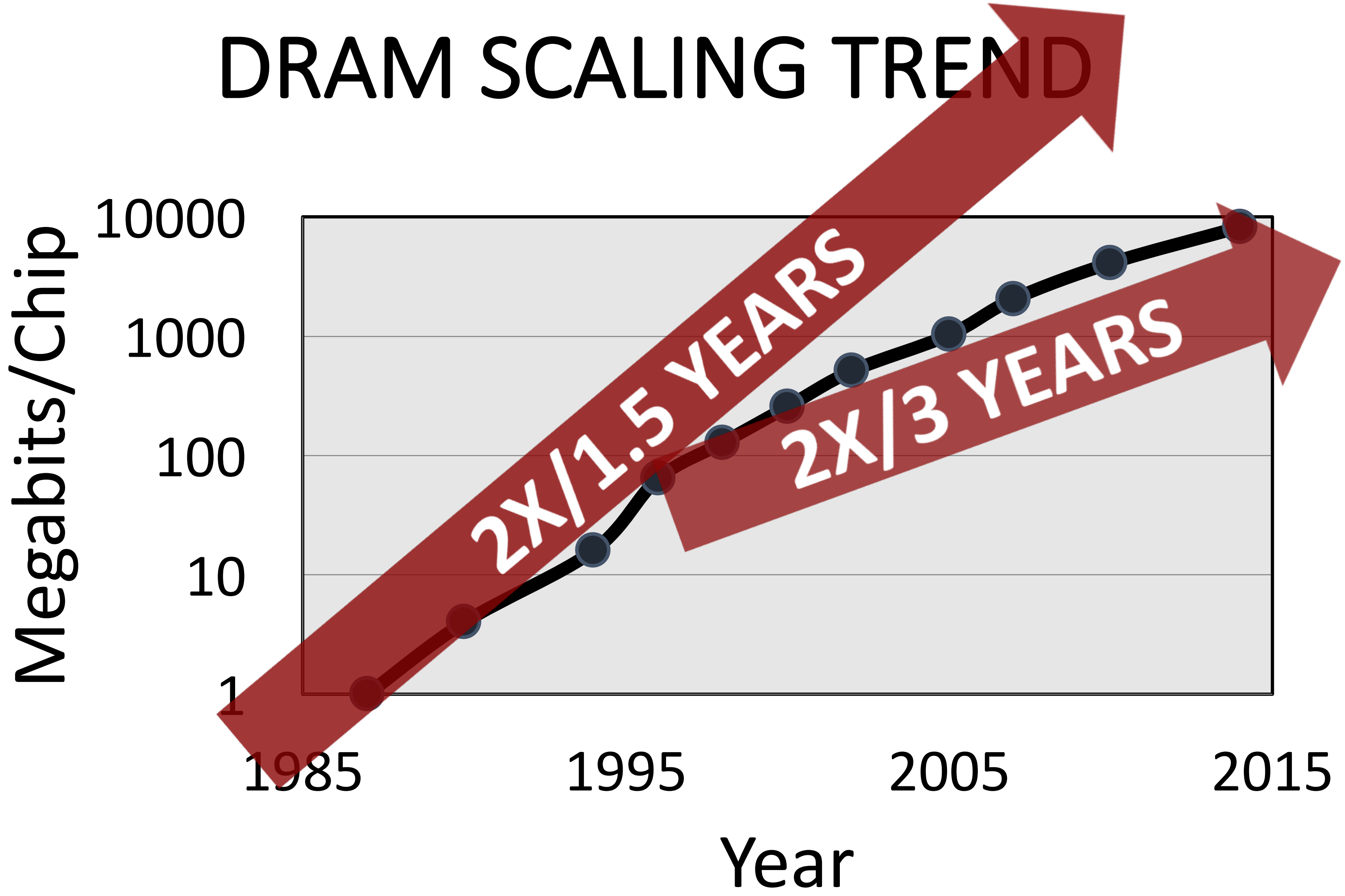
MEMCON
Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content

Samira Khan
Chris Wilkerson, Zhe Wang, Alaa Alameldeen, Donghyuk Lee, Onur Mutlu

DRAM SCALING TREND



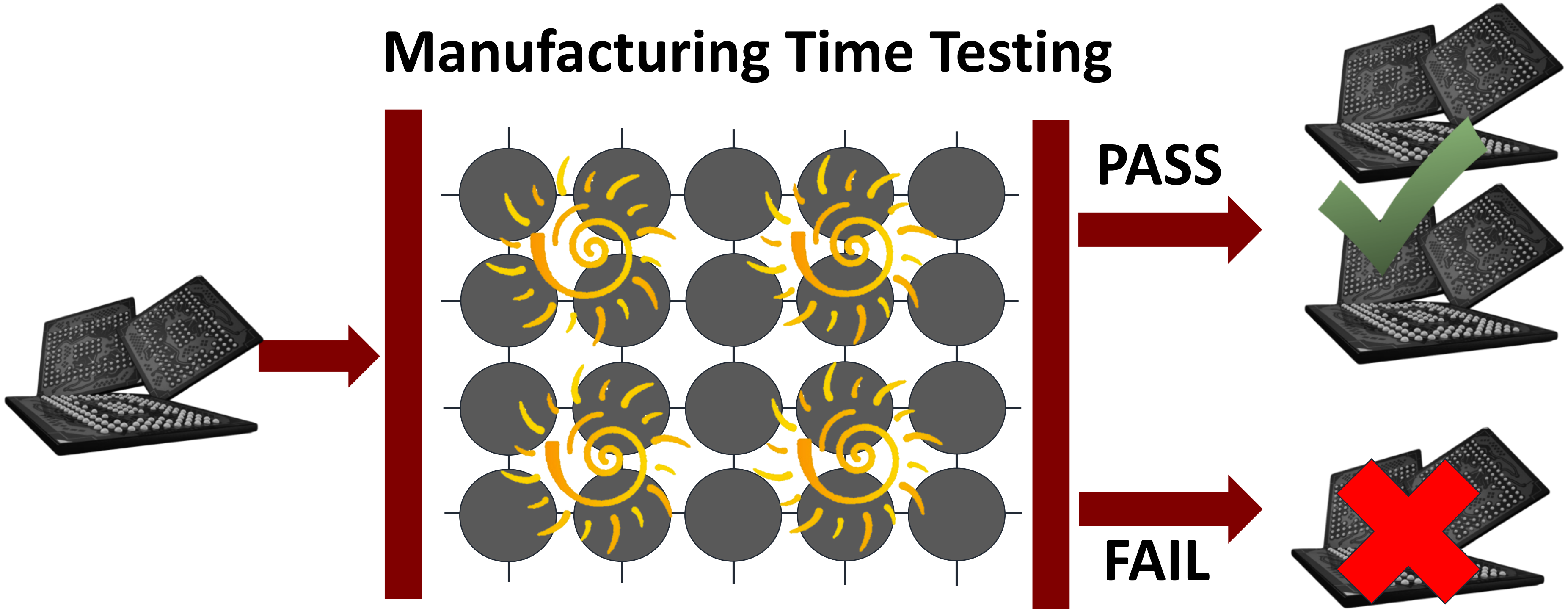
DRAM scaling is slowing down

Source: Flash Memory Summit 2013, Memcon 2014

DRAM SCALING CHALLENGE

Manufacturing reliable cells at low cost
is getting difficult

TRADITIONAL APPROACH TO ENABLE DRAM SCALING

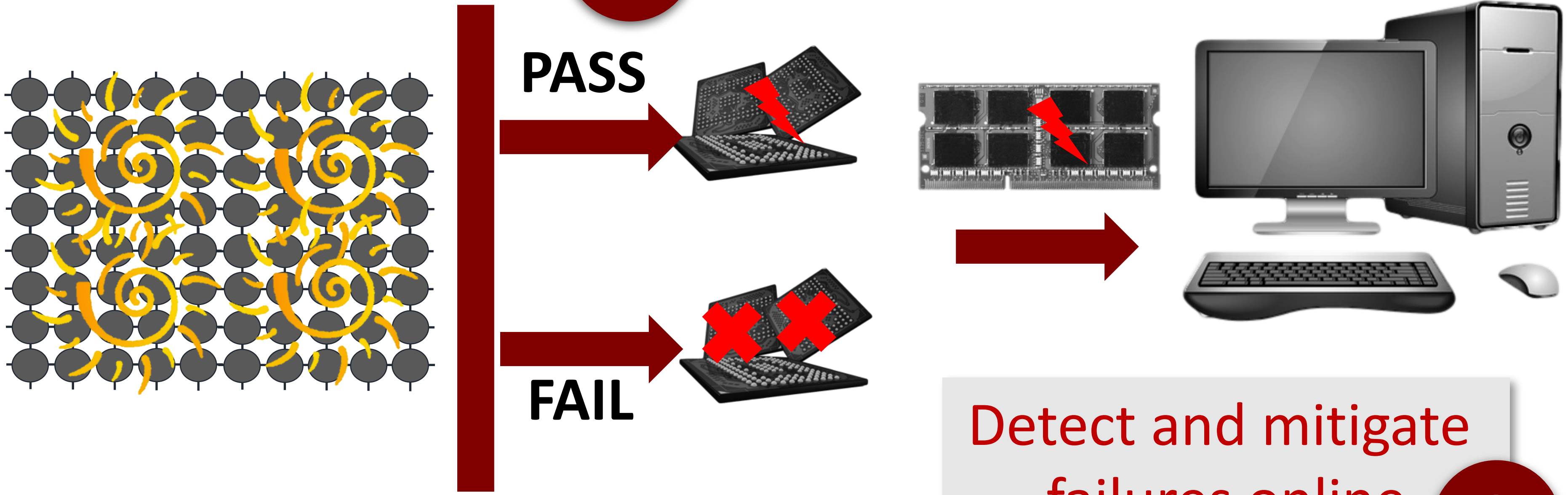


1. Manufacturers perform exhaustive testing of DRAM chips
2. Chips failing the tests are discarded

VISION: SYSTEM-LEVEL DETECTION AND MITIGATION

1 Not fully tested during manufacture-time

2 Ship modules with possible failures



3 Detect and mitigate failures online

Detect and mitigate errors after the system has become operational

ONLINE PROFILING

BENEFITS OF ONLINE PROFILING

1. Improves yield, reduces cost, enables scaling

Vendors can make cells smaller without a strong reliability guarantee

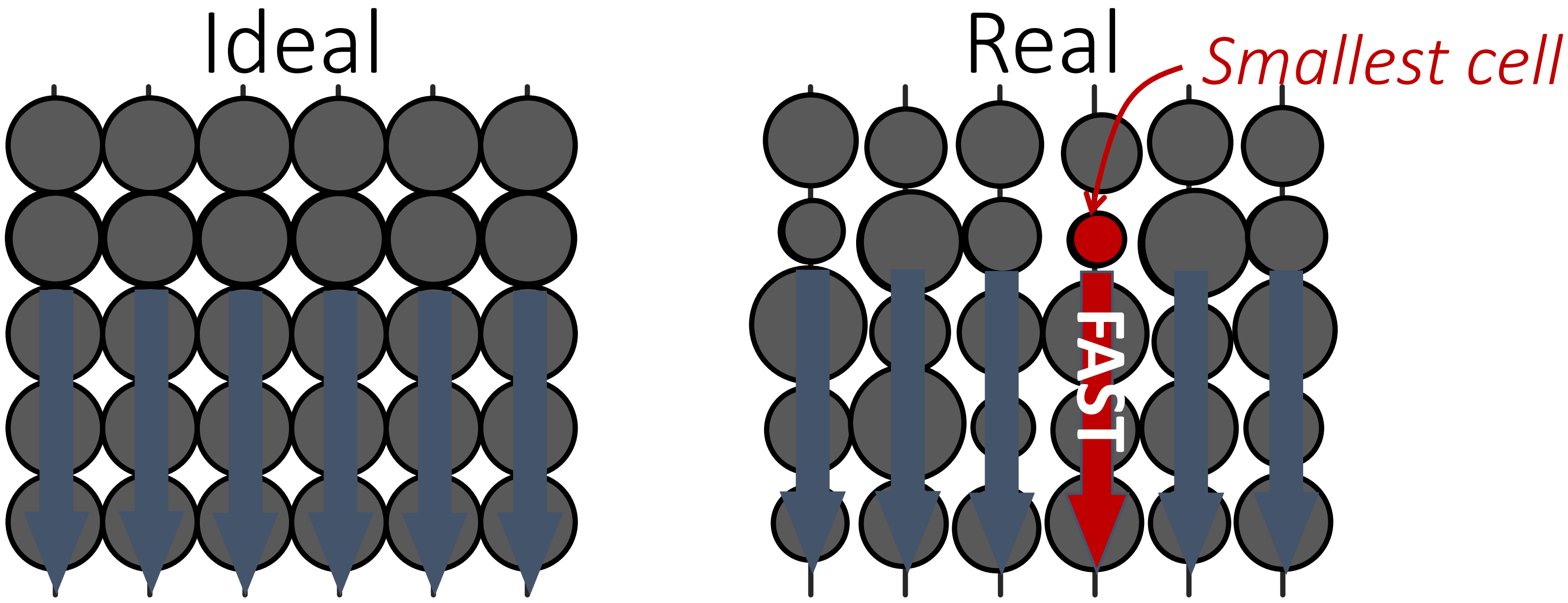
BENEFITS OF ONLINE PROFILING

1. Improves yield, reduces cost, enables scaling

Vendors can make cells smaller without a strong reliability guarantee

2. Improves performance and energy efficiency

DRAM CELLS ARE NOT EQUAL

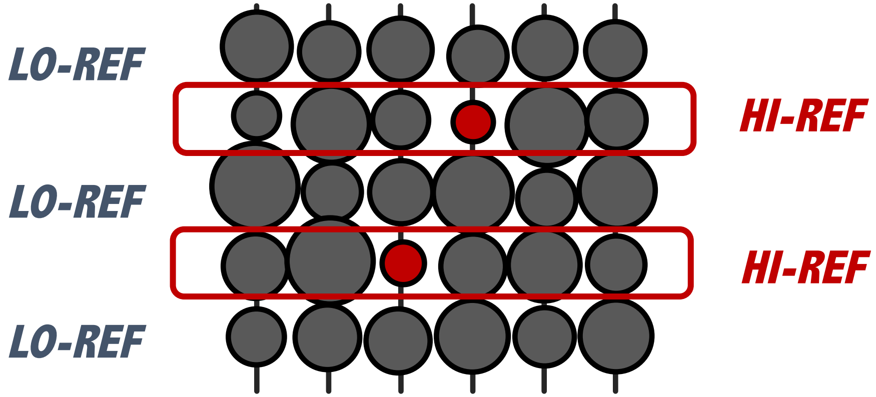


Large variation in retention time

Most cells have high retention time → can be refreshed at a lower rate without any failure

Smaller cells will fail to retain data at a lower refresh rate

BENEFITS OF ONLINE PROFILING



Unreliable DRAM Cells

Reduce refresh count by using a lower refresh rate, but use higher refresh rate for faulty cells

1. Improves yield, reduces cost, enables scaling

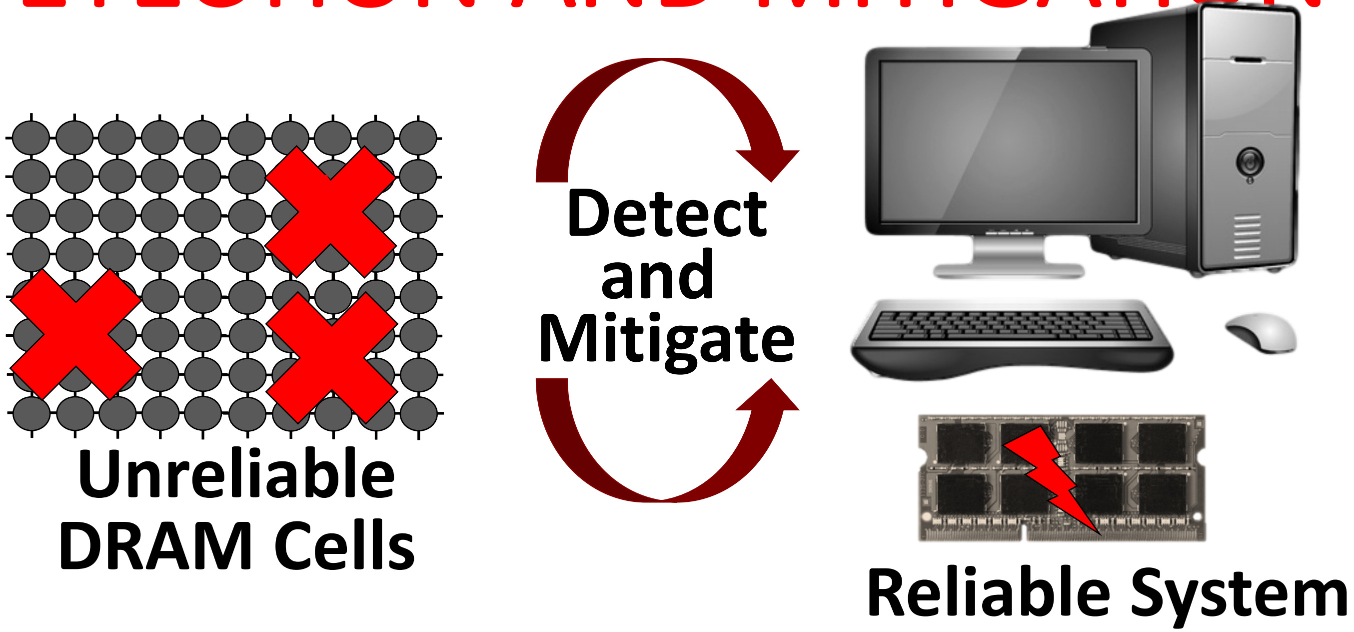
Vendors can make cells smaller without a strong reliability guarantee

2. Improves performance and energy efficiency

Reduce refresh rate, refresh faulty rows more frequently

In order to enable these benefits,
we need to ***detect*** *the failures*
at the system level

CHALLENGE IN SYSTEM-LEVEL DETECTION AND MITIGATION

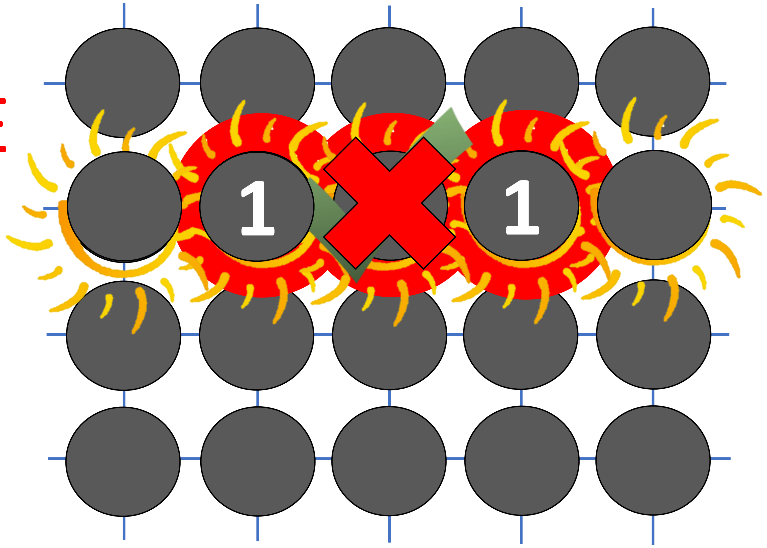


Depends on accurately *detecting* DRAM failures

If failures were permanent,
a simple boot up test would have worked,
but there are intermittent failures

DATA-DEPENDENT FAILURE

INTERFERENCE
DUE TO
COUPLING



NO
~~FAILURE~~
FAILURE

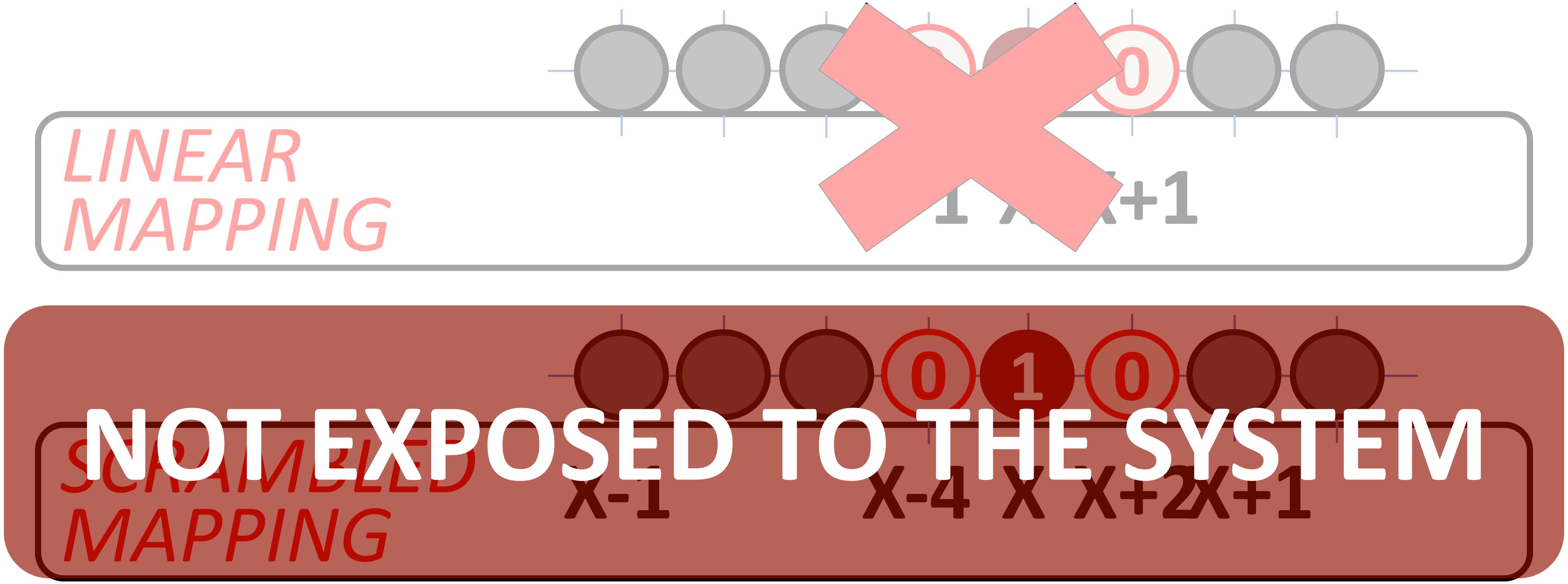
Some cells can fail depending on the data stored in neighboring cells

How can we detect these failures at the system level?

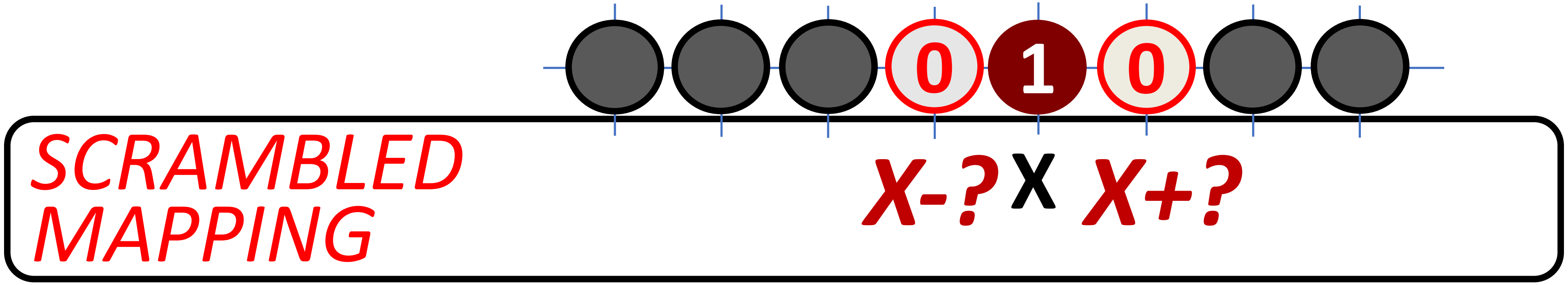
JSSC'88, MDTD'02

DETECTING DATA-DEPENDENT FAILURES

Testing with specific patterns in neighboring addresses



CHALLENGE IN DETECTION

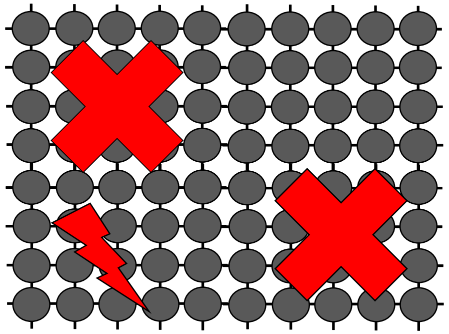


How to detect *data-dependent failures* when we even do not know which *cells are neighbors*?

PROBLEM: SCRAMBLED ADDRESS MAPPING

MEMCON: DRAM-Internal Independent Detection

CAL'16, MICRO'17



DRAM

System-Level Detection and Mitigation of Failures

GOAL

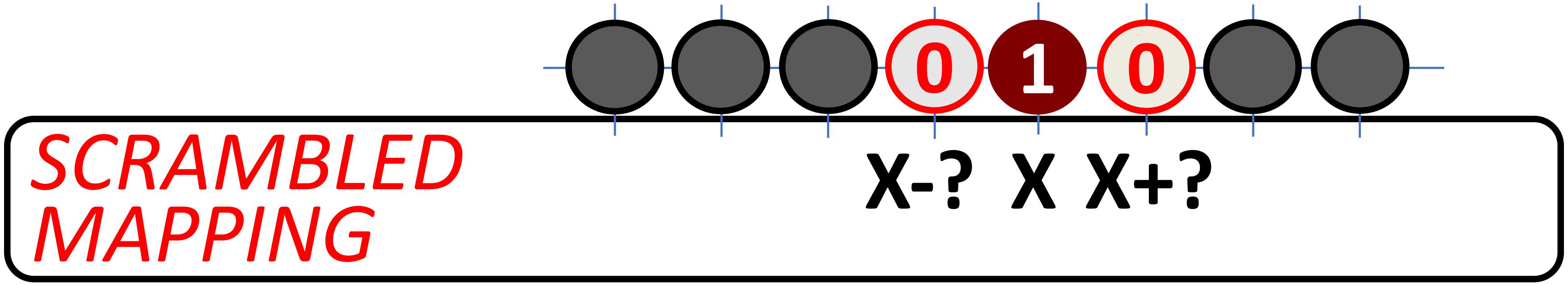
KEY IDEA

CHALLENGE

MECHANISM

RESULTS

GOAL: MEMCON



Detect data-dependent failures
without the knowledge of
the DRAM internal address mapping

CURRENT DETECTION MECHANISM

Initial Failure Detection and Mitigation	Execution of Applications

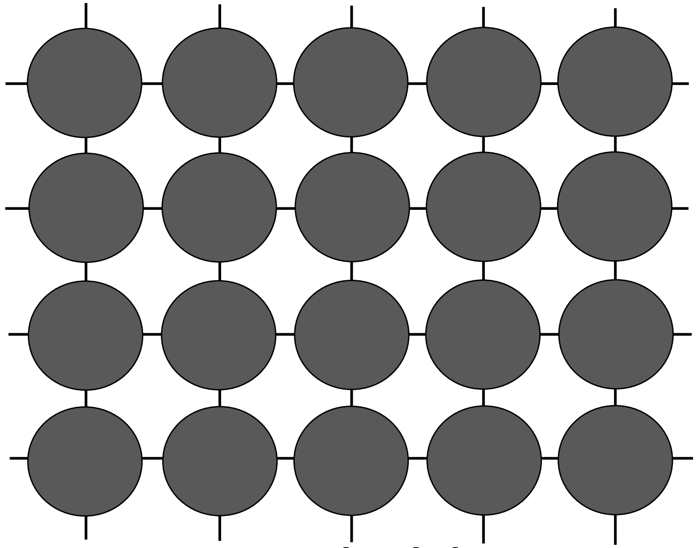
Detection is done with some initial testing isolated from system execution

1. Detect and mitigate all failures with every possible content
2. Only after that start program execution

RAPID HPCA'06, Rethinking Refresh IEEE Micro'08 , RAIDR ISCA'12, SECRET ICCD'12, Dtail SC'14, ProactiveDRAM ICCD'14, REAPER ISCA'17

CURRENT DETECTION MECHANISM

Detect every possible failure with all content before execution



Unreliable
DRAM Cells

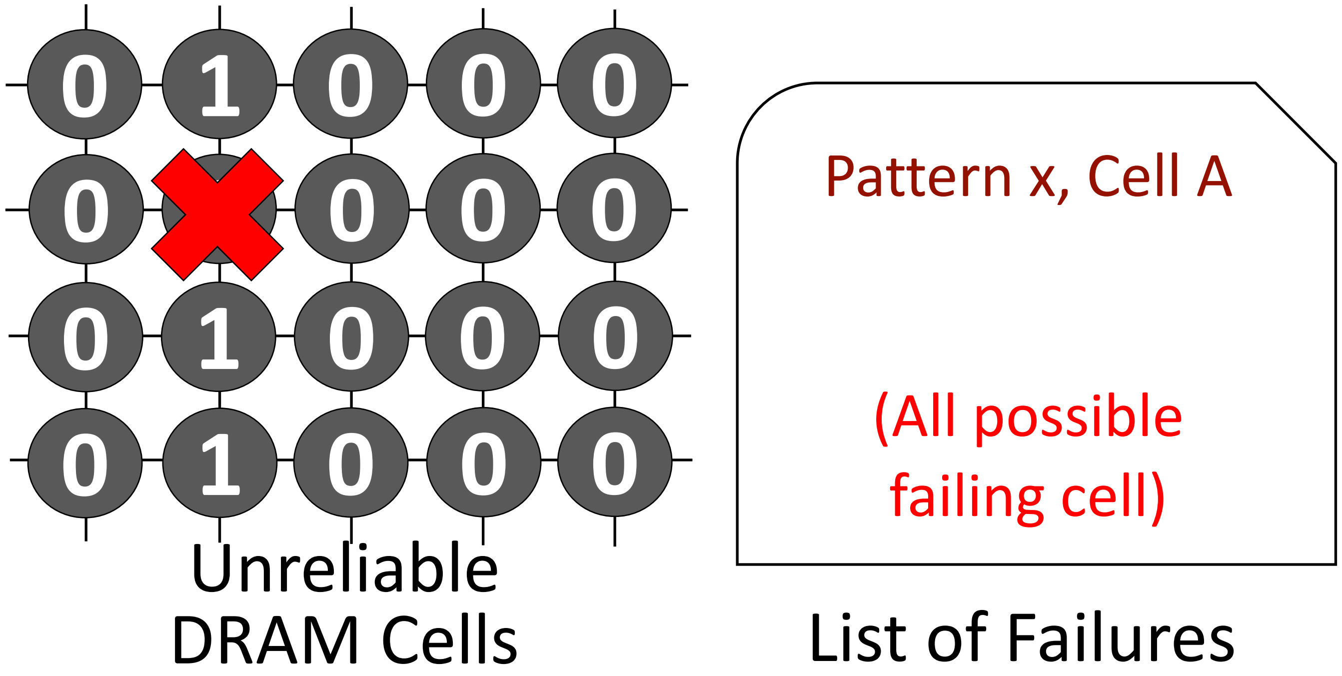
(All possible
failing cell)

List of Failures

Initial Failure Detection and Mitigation

CURRENT DETECTION MECHANISM

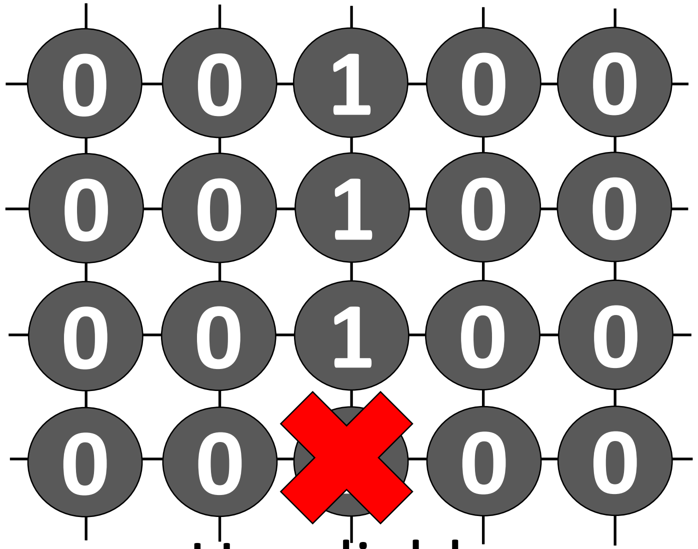
Detect every possible failure with all content before execution



Initial Failure Detection and Mitigation

CURRENT DETECTION MECHANISM

Detect every possible failure with all content before execution



Unreliable
DRAM Cells

Pattern x, Cell A
Pattern y, Cell B

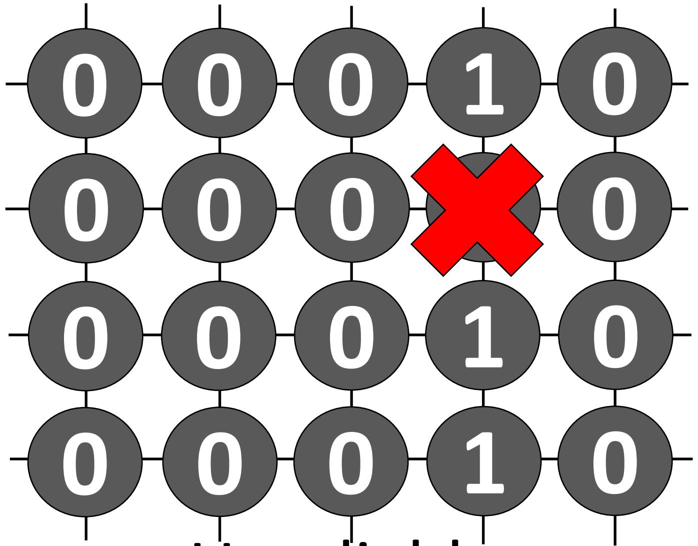
(All possible failing cell)

List of Failures

Initial Failure Detection and Mitigation

CURRENT DETECTION MECHANISM

Detect every possible failure with all content before execution



Unreliable DRAM Cells

Pattern x, Cell A
Pattern y, Cell B
Pattern z, Cell C
…
(All possible failing cell)

List of Failures

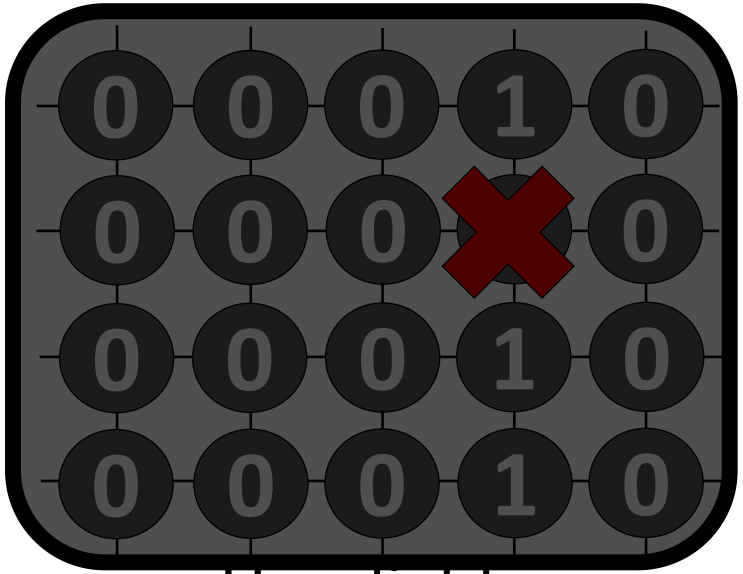
Applications

Initial Failure Detection and Mitigation

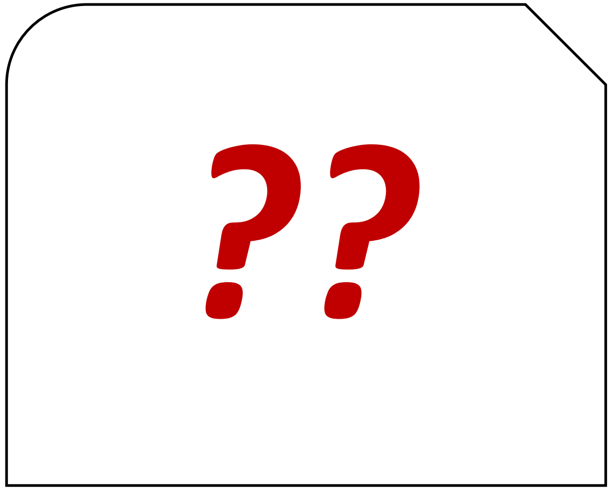
Execution of Applications

CURRENT DETECTION MECHANISM

Detect every possible failure with all content before execution



Unreliable
DRAM Cells



List of Failures

Applications

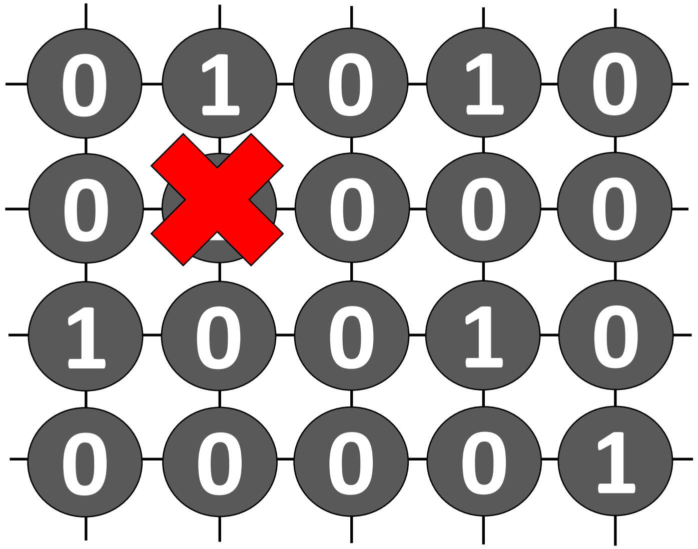
Initial Failure Detection and Mitigation

Execution of Applications

The problem is cannot detect ***all*** failures
when we do not know the address mapping

MEMCON: MEMORY CONTENT-BASED DETECTION AND MITIGATION

NO NEED TO DETECT EVERY POSSIBLE FAILURE



Unreliable DRAM Cells
with Program Content

**Current content,
Cell A**

List of Failures

Application

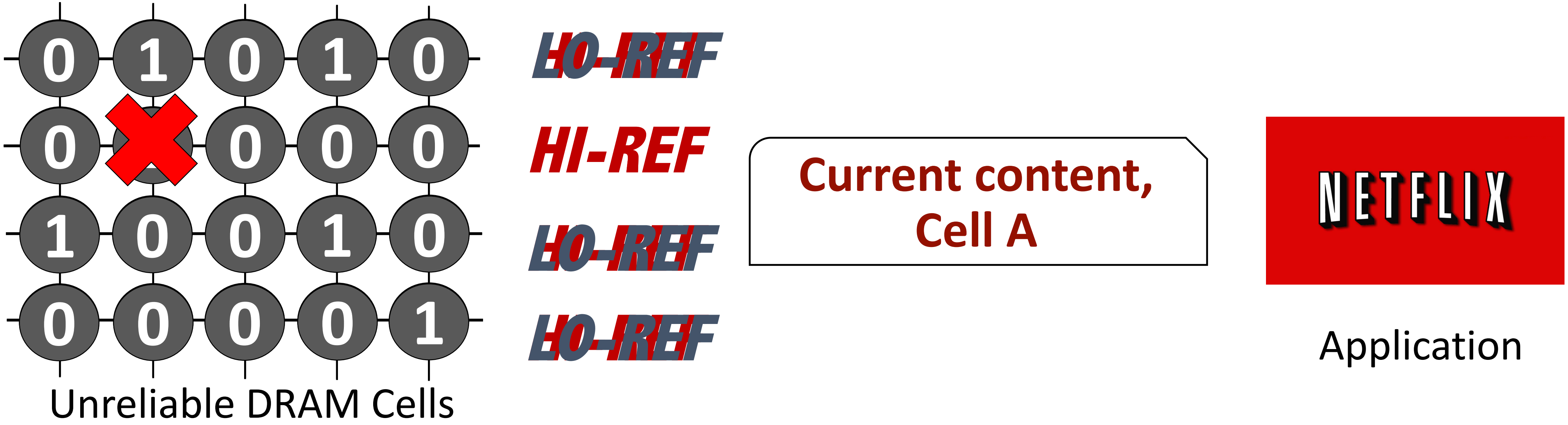
Simultaneous Detection and Execution

Based on current memory content of running applications

Need to detect and mitigate
only with the current content

MEMCON: HIGH-LEVEL DESIGN

Simultaneous Detection and Execution



1. No initial detection and mitigation
2. Start running the application with a high refresh rate
3. Detect failures with the current memory content
    - If no failure found, use a low refresh rate

PROBLEM: SCRAMBLED ADDRESS MAPPING

MEMCON: DRAM-Internal Independent Detection

CAL'16, MICRO'17

GOAL

KEY IDEA

CHALLENGE

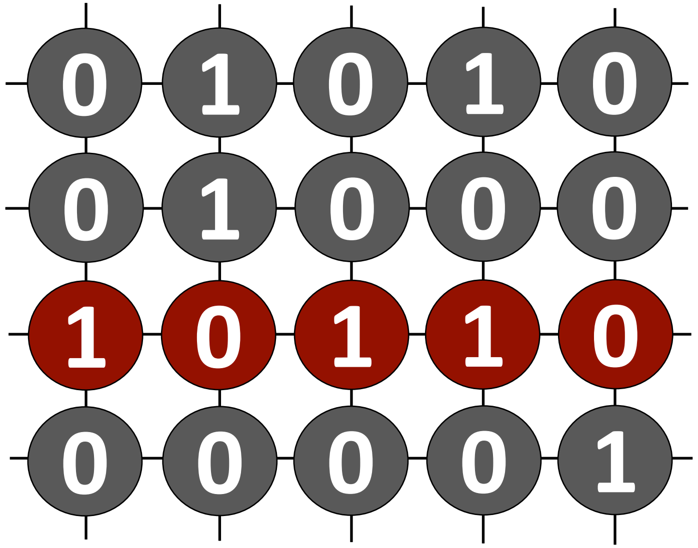
MECHANISM

RESULTS

CHALLENGE WITH MEMCON

NEED TO DETECT FAILURE AT A WRITE

Write access to row A



Unreliable DRAM Cells with Program Content

Current content, Cell ??

NETFLIX

Testing at every write is expensive!!!

MEMCON: COST-BENEFIT ANALYSIS

The cost of testing

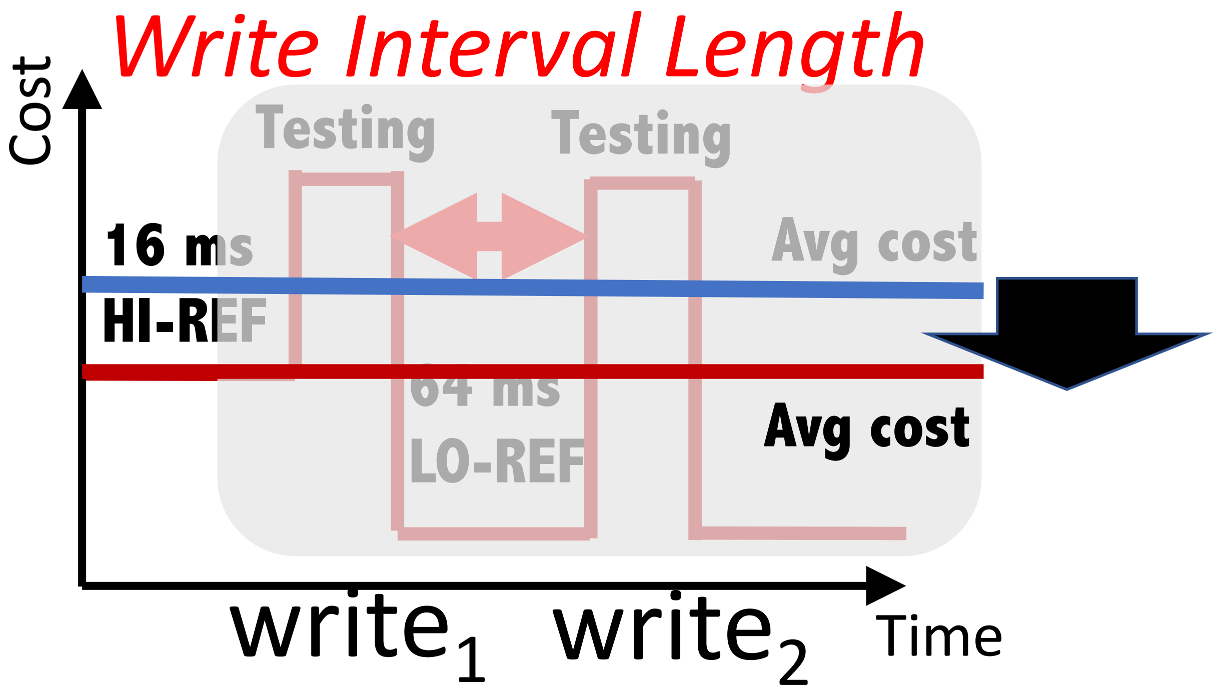
Extra memory accesses to read and write rows

The benefit of testing

If no failure found, can reduce refresh rate

MEMCON: COST-BENEFIT ANALYSIS

Frequent testing introduces high cost

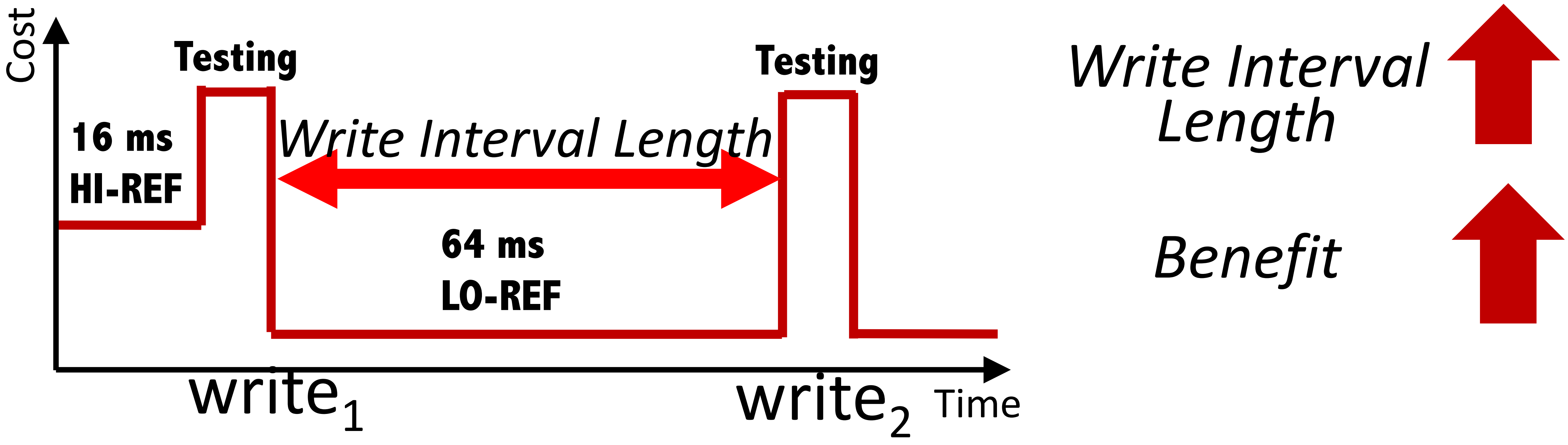


Short write interval → High average cost

Using ***HI-REF*** is better when write interval length is short

MEMCON: COST-BENEFIT ANALYSIS

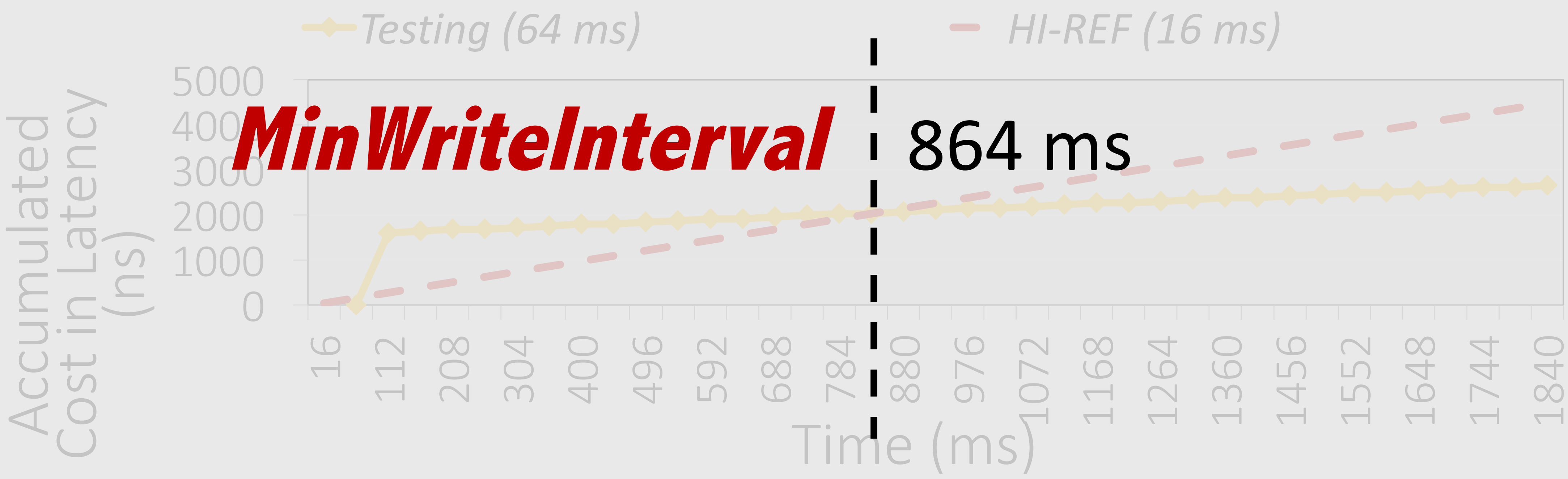
The higher the write interval length, the higher is the benefit



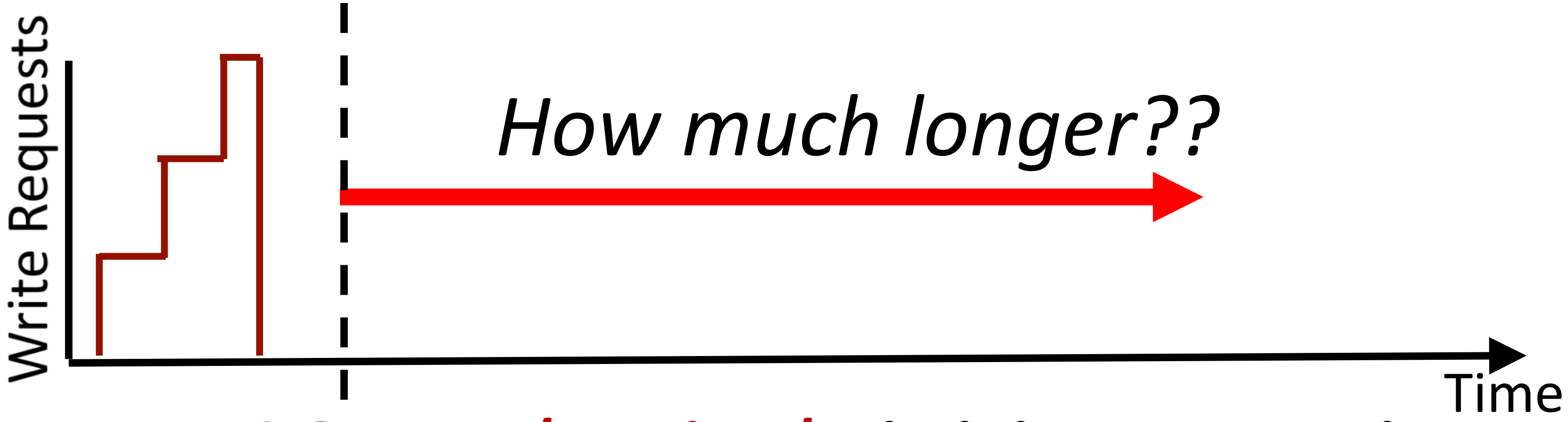
Can amortize the cost of testing with long intervals

MEMCON: COST BENEFIT ANALYSIS

What is the write interval that can amortize the cost?



MEMCON: MECHANISM



MEMCON *selectively* initiates testing when the write interval is *long enough to amortize* the cost of testing

How do we predict the interval on a write access?

PROBLEM:
SCRAMBLED ADDRESS MAPPING

MEMCON: DRAM-Internal
Independent Detection

CAL’16, MICRO’17

GOAL

KEY IDEA

CHALLENGE

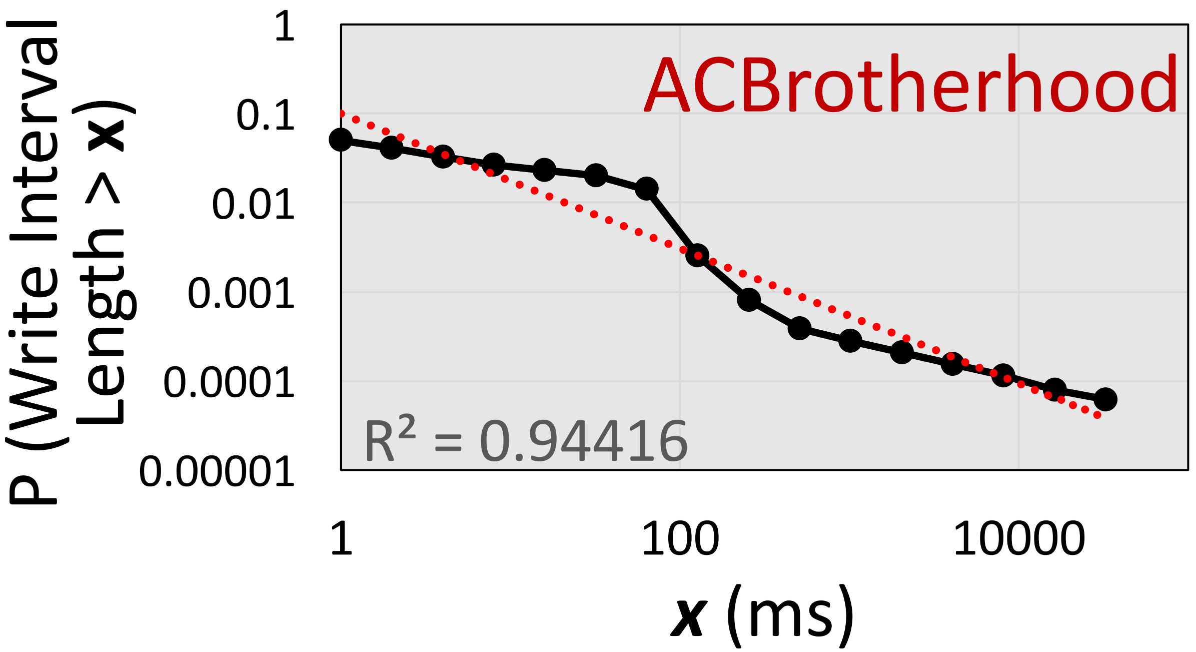
MECHANISM

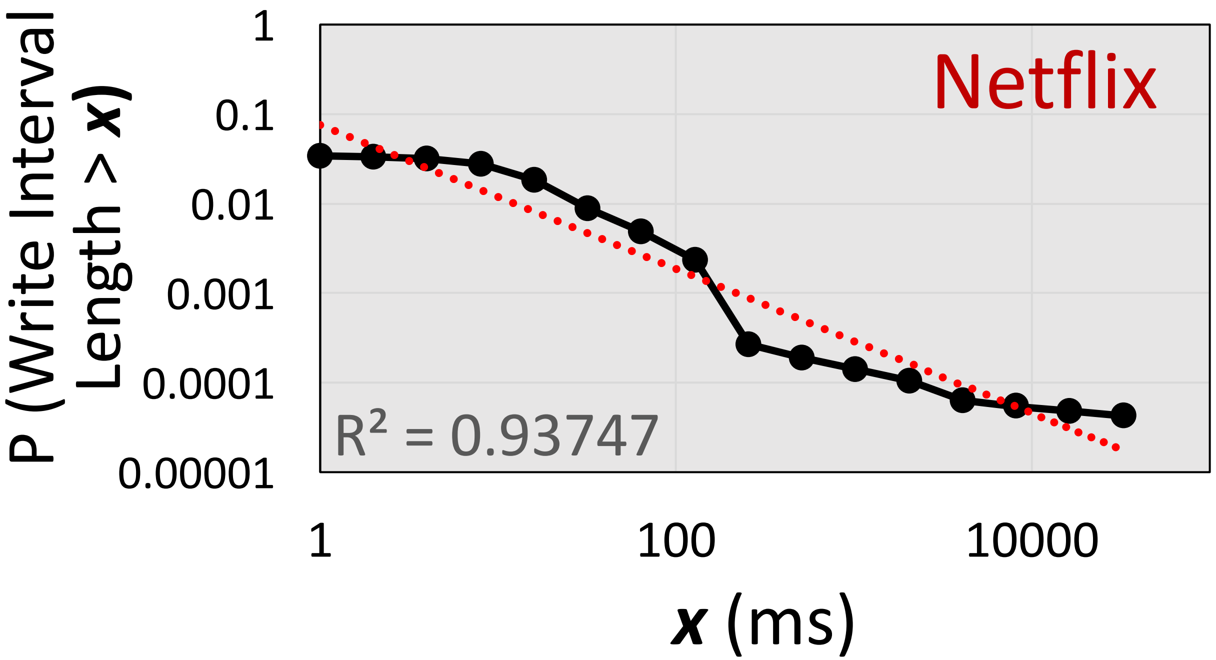
RESULTS

32 seconds of execution on a real machine

Profiled with a custom FPGA-based infrastructure

WRITE INTERVAL CHARACTERISTICS



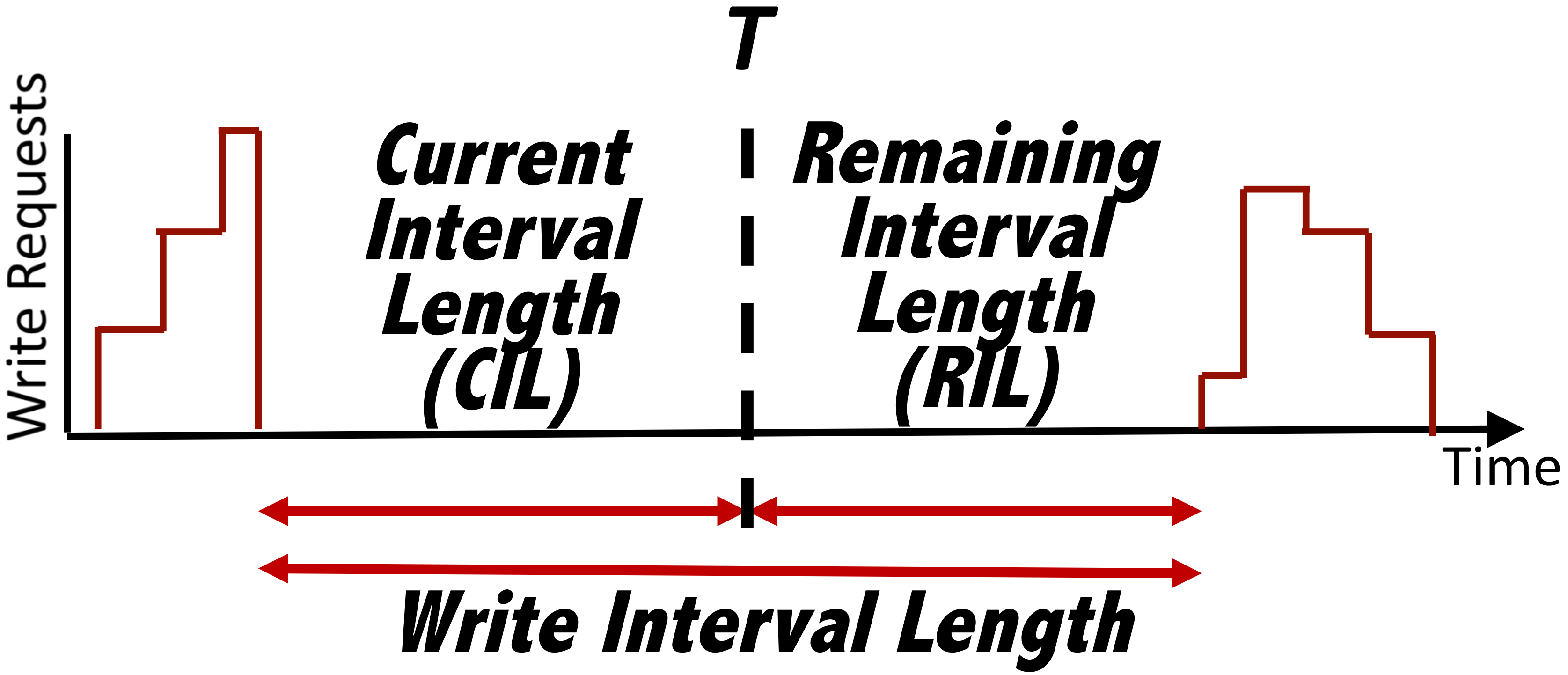


Write intervals follow a Pareto distribution

The longer the elapsed time after a write

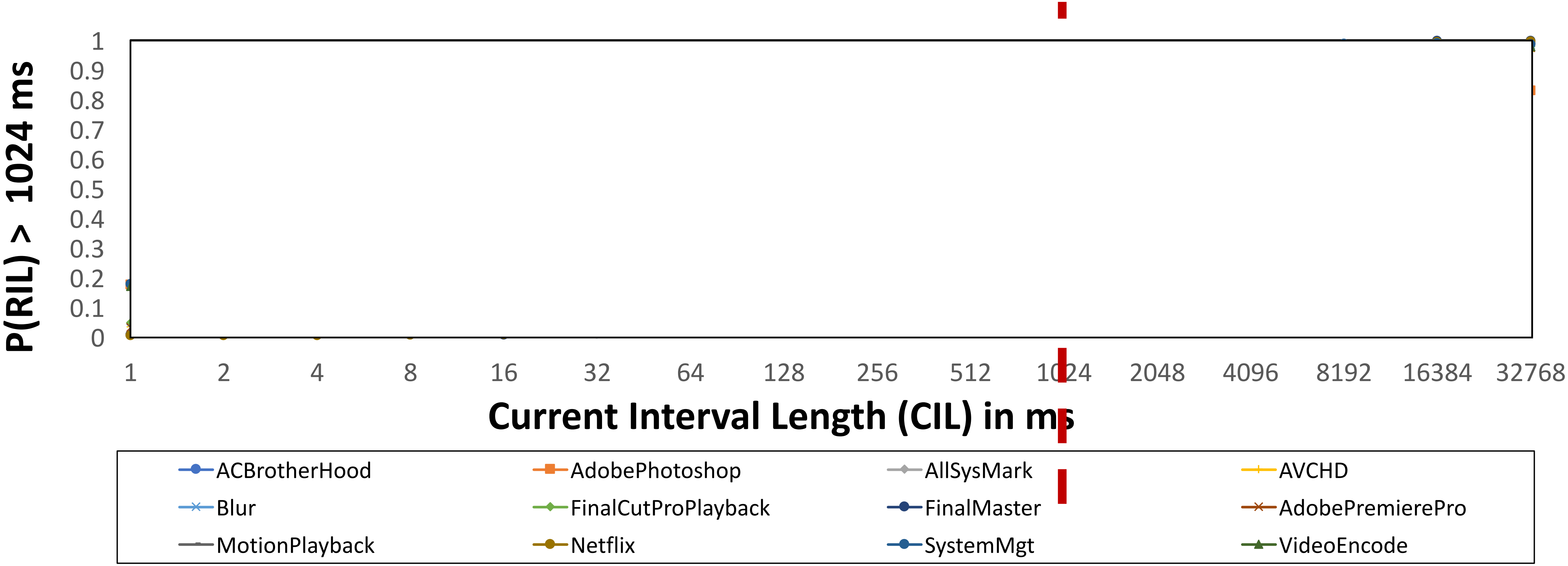
→ The longer the write interval

WRITE INTERVAL CHARACTERISTICS



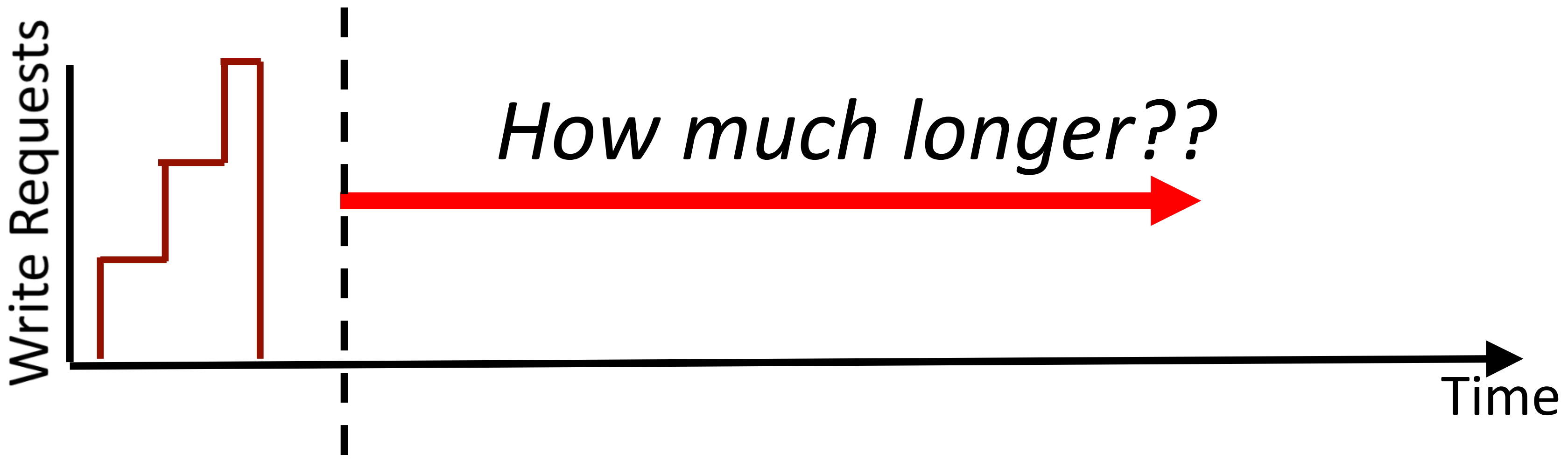
The longer the ***CIL***
→ It is expected that the longer the ***RIL***

WRITE INTERVAL CHARACTERISTICS

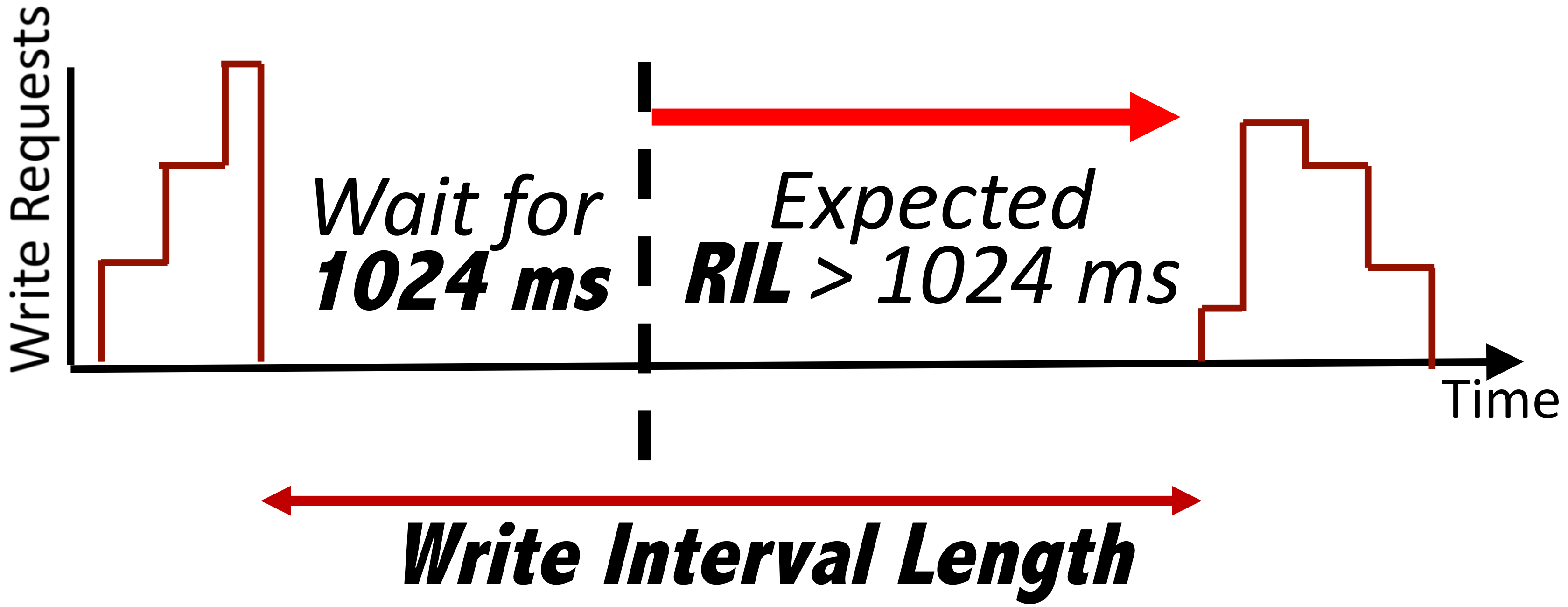


If the interval is already 1024 ms long, the probability that the remaining interval is greater than 1024 ms is on average 76%

How do we predict the interval on a write access?



WRITE INTERVAL PREDICTION



After a write, wait for a CIL, where P(RIL) > 1024 is high

If idle, predict the interval will last more than 1024 ms

PROBLEM: SCRAMBLED ADDRESS MAPPING

MEMCON: DRAM-Internal Independent Detection

CAL’16, MICRO’17

GOAL

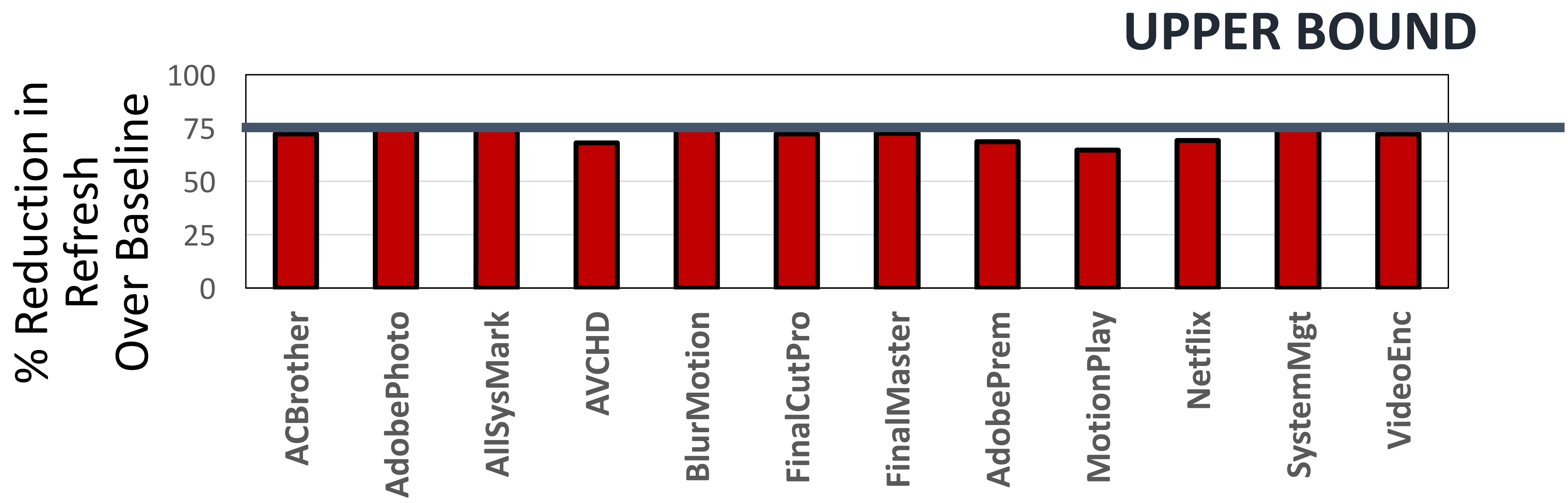
KEY IDEA

CHALLENGE

MECHANISM

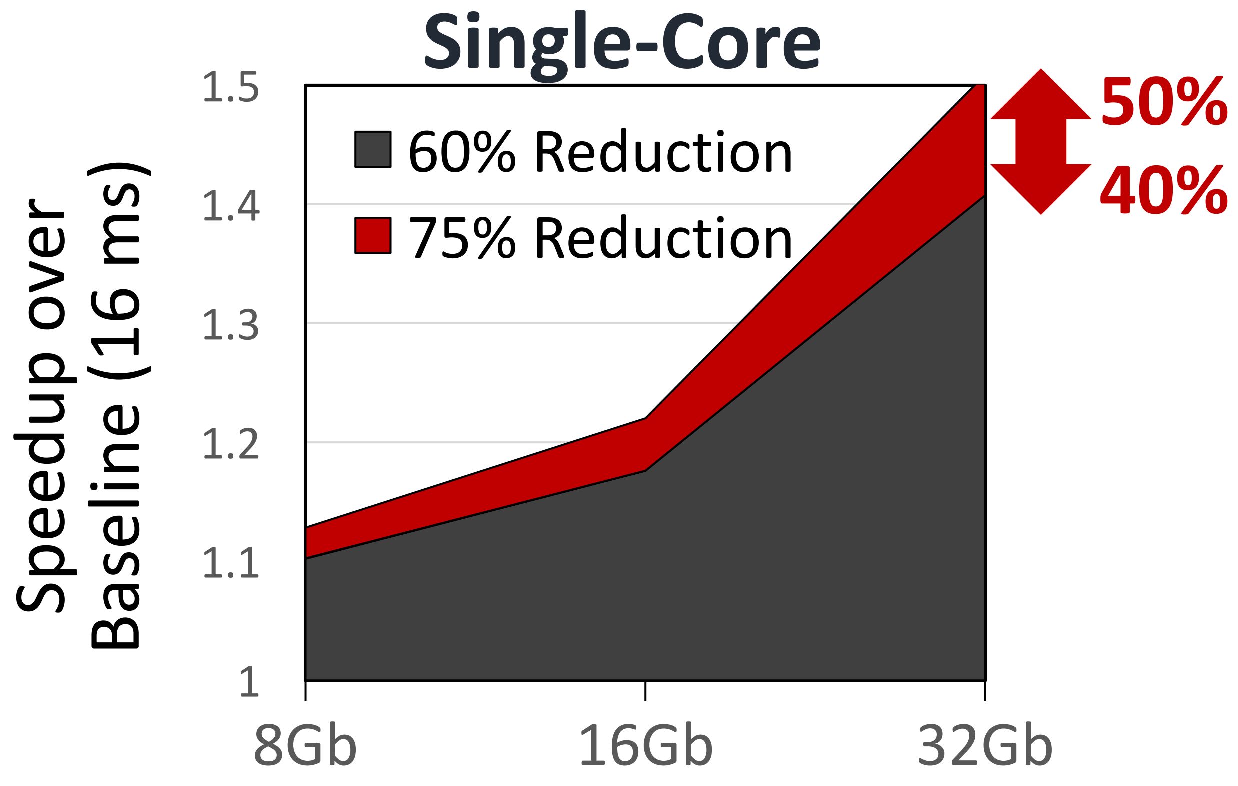
RESULTS

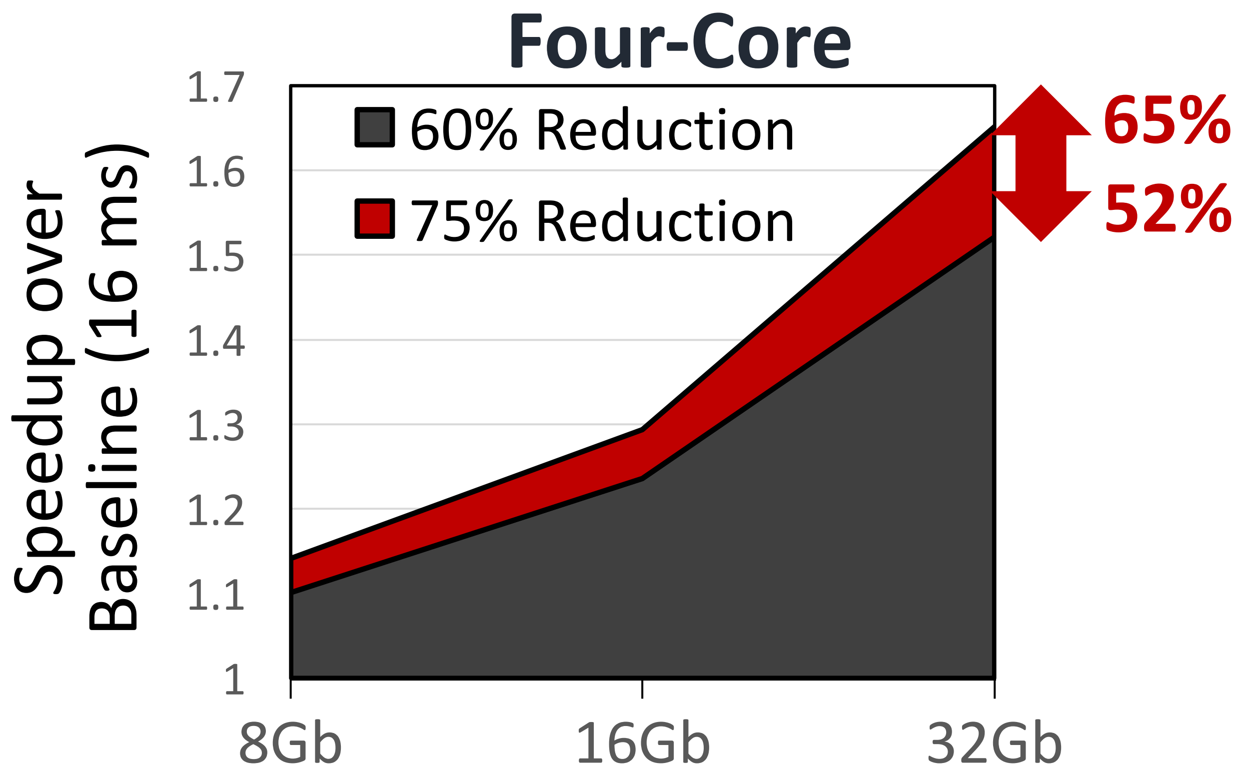
MEMCON: REDUCTION IN REFRESH COUNT



71% average reduction in refresh count,
very close to the upper bound of 75%

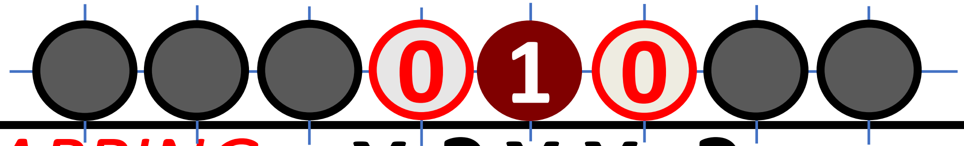
MEMCON: PERFORMANCE IMPROVEMENT DUE TO REDUCTION IN REFRESH COUNT





Refresh reduction leads to significant performance improvement

MEMCON: SUMMARY



Problem: SCRAMBLED MAPPING **X-? X X+?**

Goal: MEMCON ***detects and mitigates*** data-dependent failures ***without*** the knowledge of the DRAM internal address mapping

Key Idea: Instead of detecting every possible failure, detects failure based on the ***current content***

***Challenge:* Content changes with writes, testing at every write is expensive**

Mechanism: Selective testing mechanism based on write interval length

- Initiates a test only when can amortize the cost

Results:

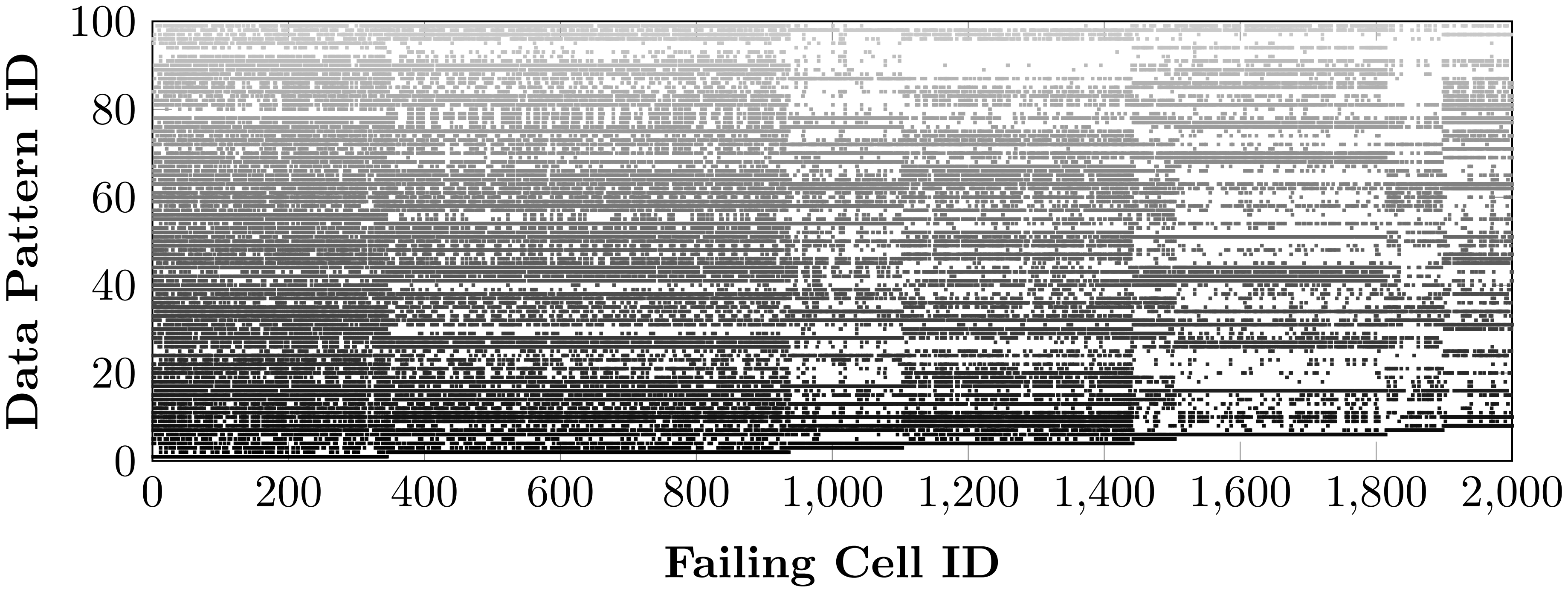
65%-74%
Reduction in refresh count

40%-50%
Performance improvement using 32Gb DRAM (1 core)

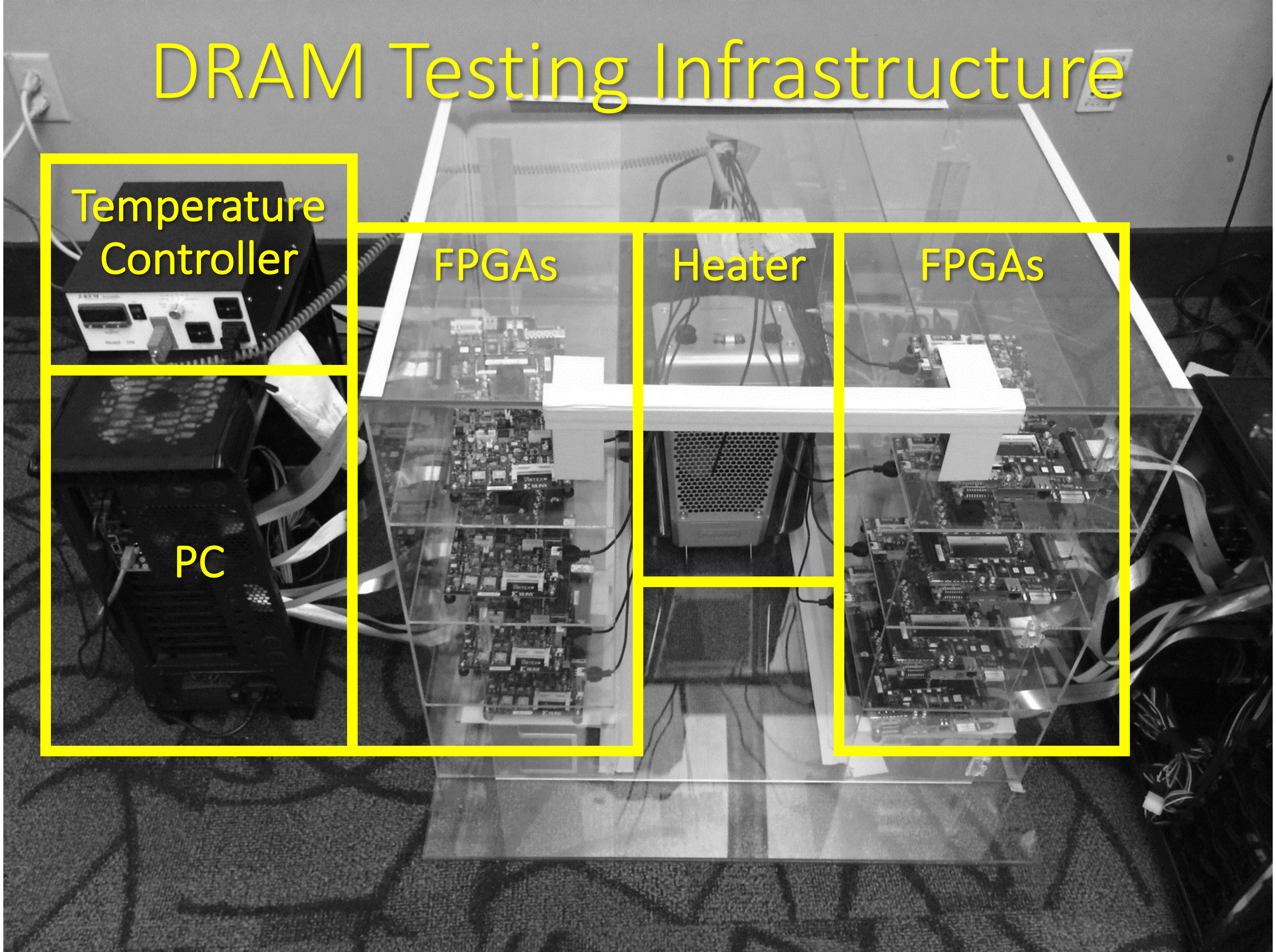
MEMCON
Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content

Samira Khan
Chris Wilkerson, Zhe Wang, Alaa Alameldeen, Donghyuk Lee, Onur Mutlu

DATA-DEPENDENT FAILURE

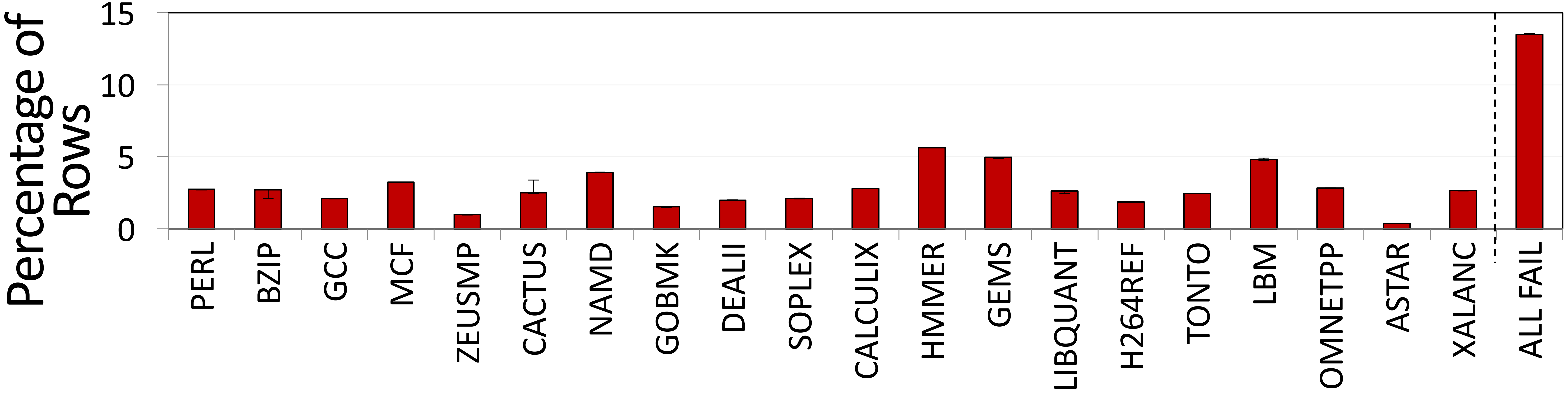


DRAM Testing Infrastructure



NUMBER OF FAILING ROWS WITH PROGRAM CONTENT

Tested with program content in real DRAM chips



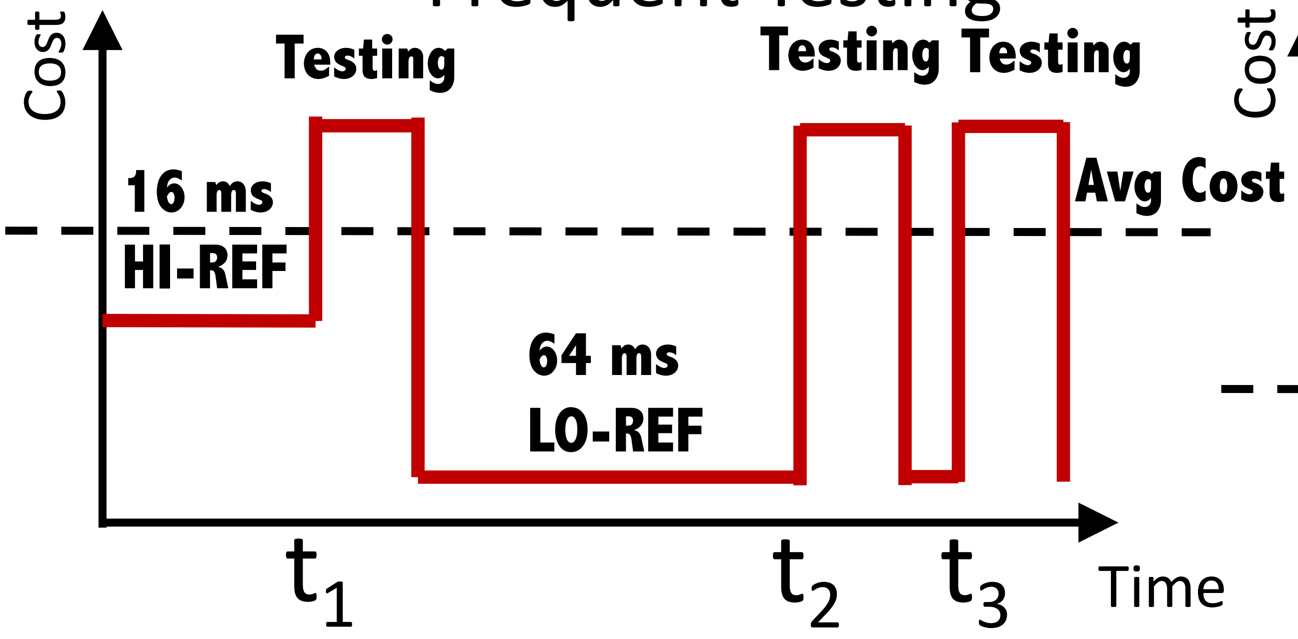
Program content exhibits significantly less failures

MEMCON: COST-BENEFIT ANALYSIS

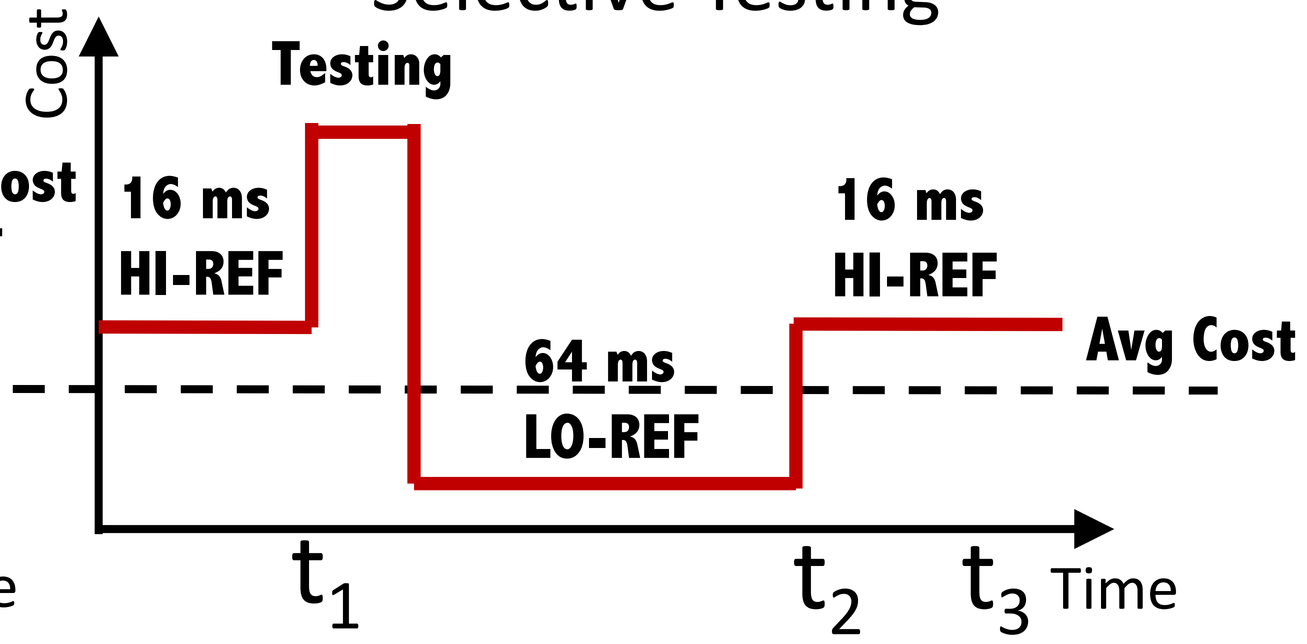
Cost: Extra memory accesses to read and write rows

Benefit: If no failure found, can reduce refresh rate

Frequent Testing

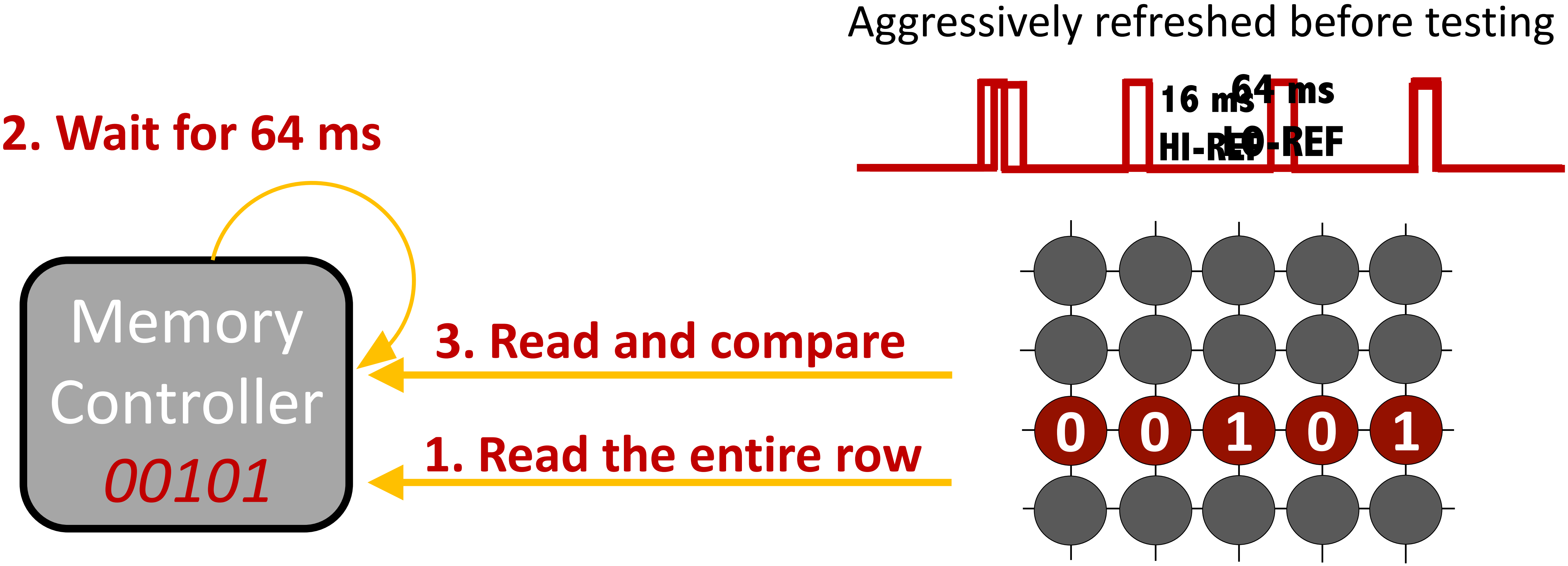


Selective Testing



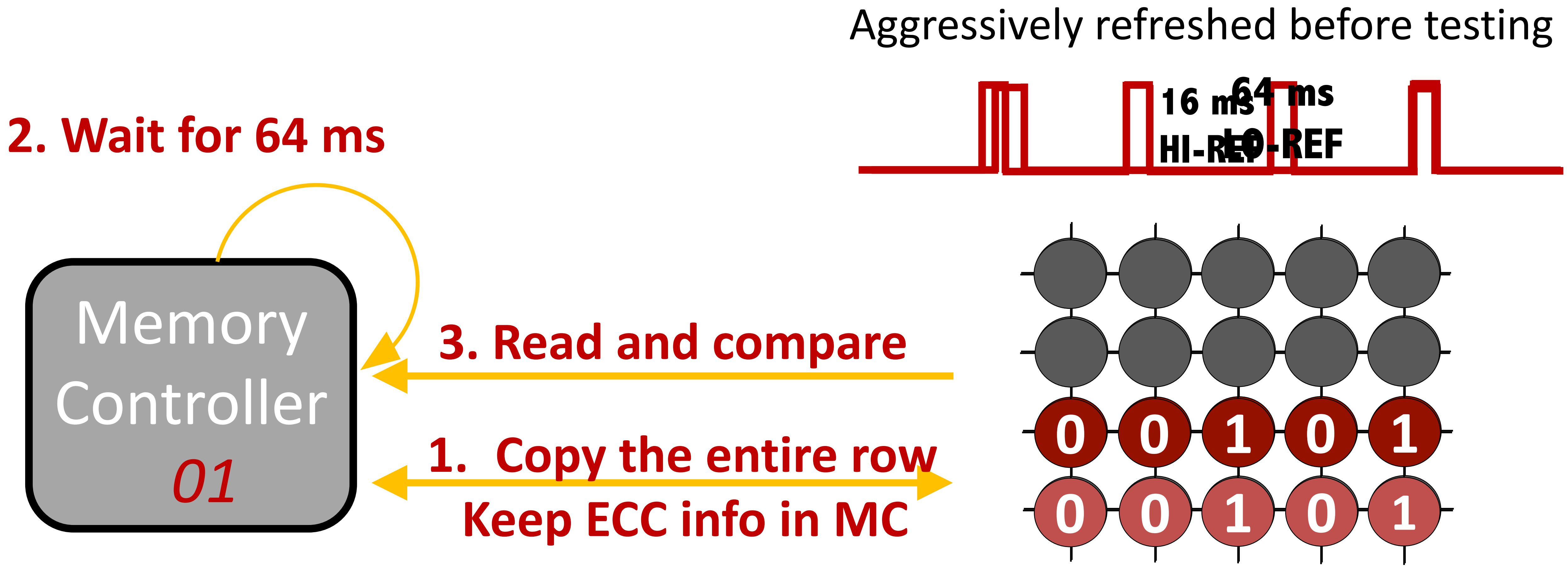
Initiate a test only when the cost can amortized

MEMCON: **READ AND COMPARE**



Benefit: If no failure found, can reduce refresh rate

MEMCON: **COPY AND COMPARE**



Benefit: If no failure found, can reduce refresh rate

MEMCON: COST-BENEFIT ANALYSIS

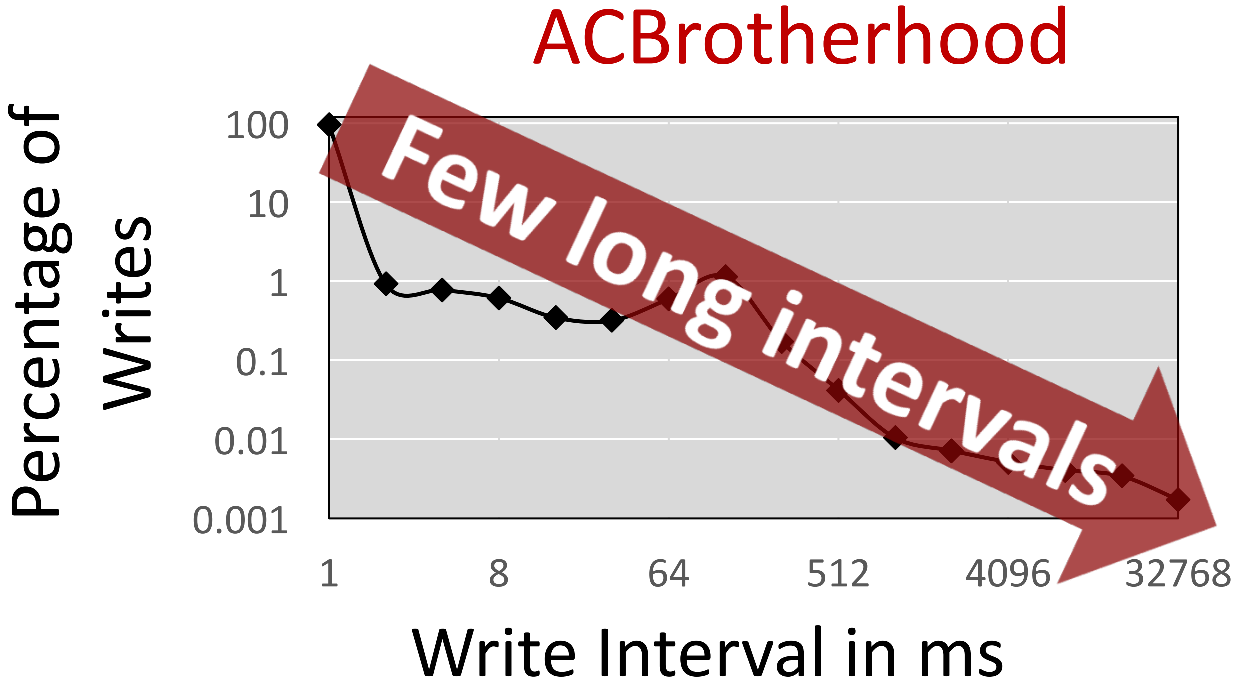
Cost: Extra memory accesses
to read and write rows while testing

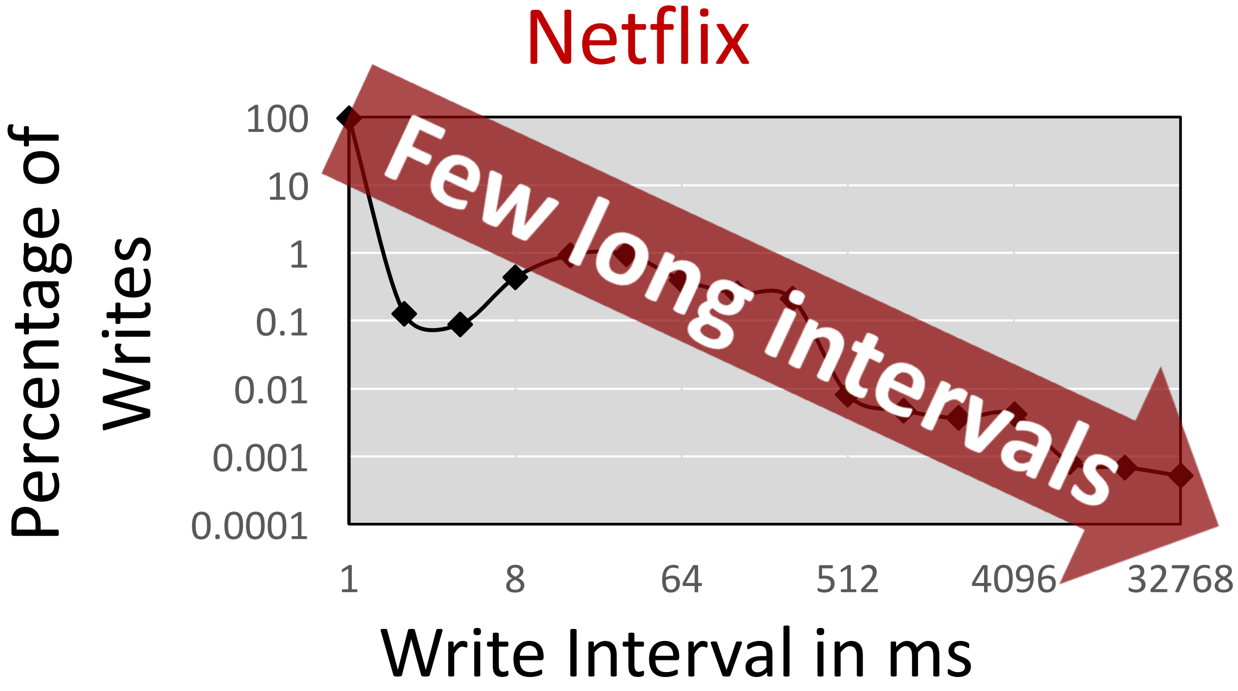
Benefit: If no failure found,
can reduce refresh rate

1. **READ AND COMPARE**
   – Read in-test row in the memory controller

2. **COPY AND COMPARE**
   – Copy in-test row in some other region in memory

WRITE INTERVAL CHARACTERISTICS

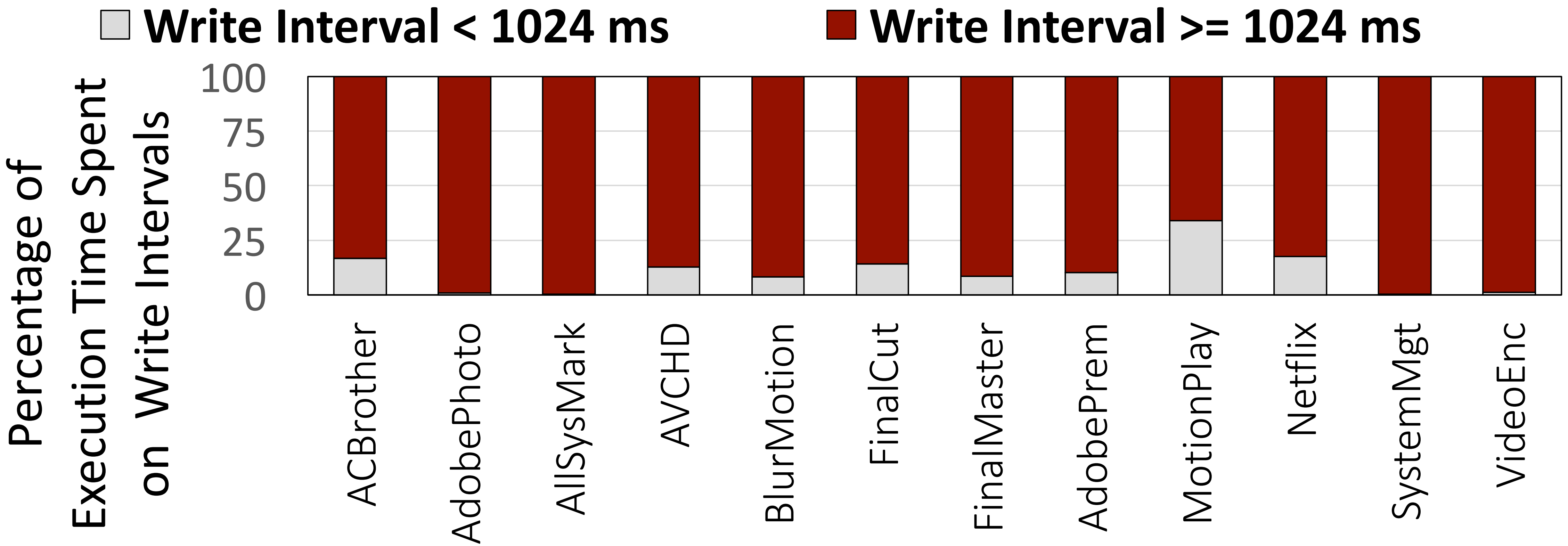




Only a small fraction of the writes exhibit long intervals

<u>but</u>

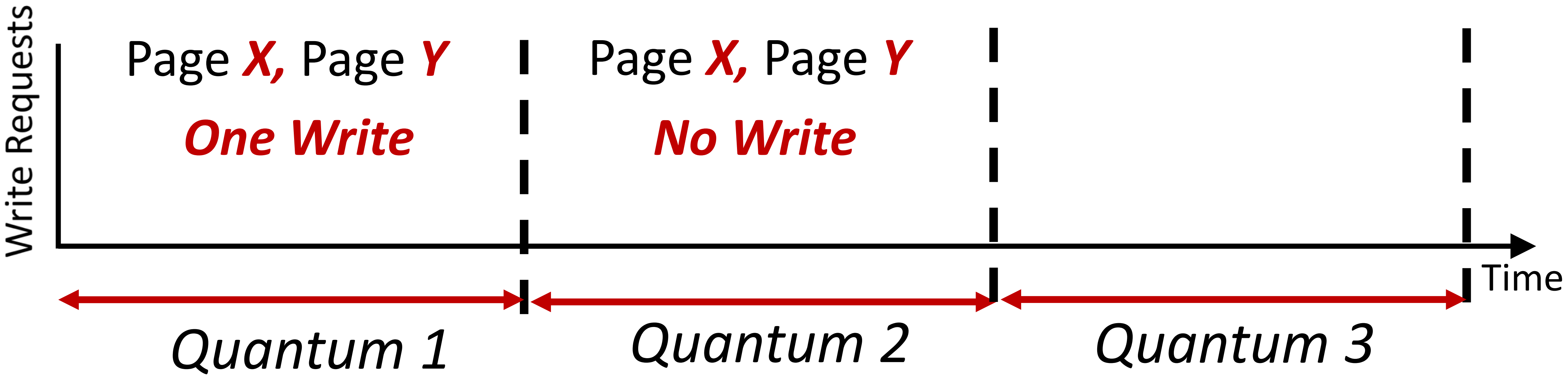
WRITE INTERVAL CHARACTERISTICS



These long intervals constitute the ***majority of the time*** spent on write intervals

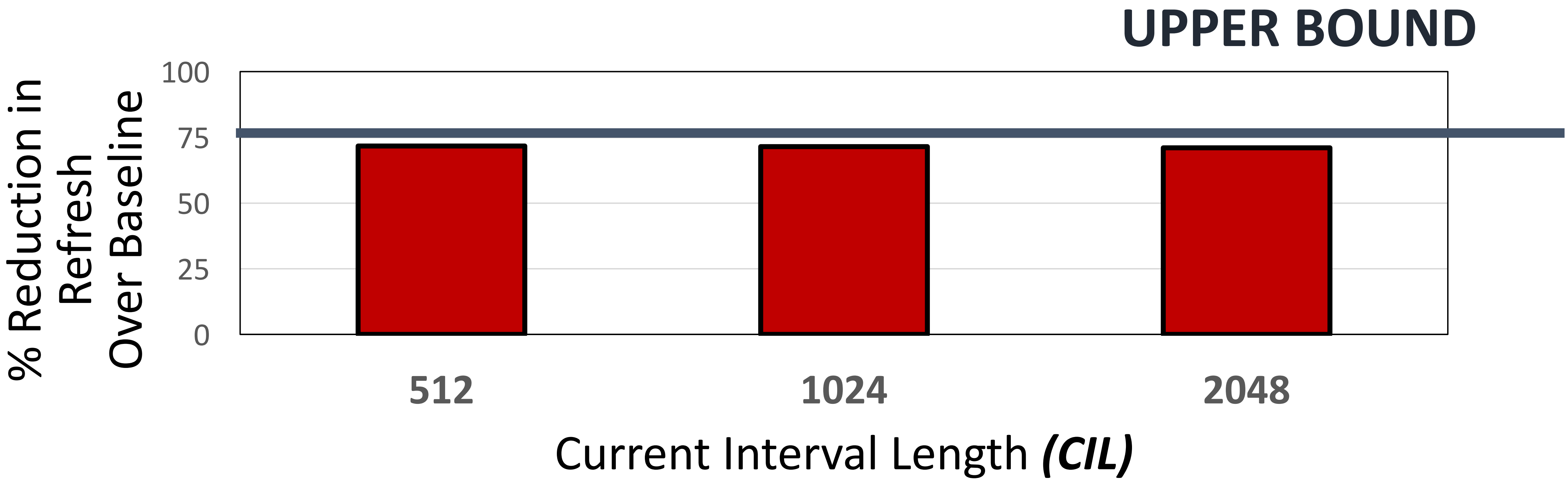
PREDICTING WRITE INTERVAL

Quantum Length = ***CIL*** *(e.g, 1024 ms)*



Predicted to have a long write interval if
*Receives a **single write** in quantum 1*
*And **no write** in quantum 2*

MEMCON: REDUCTION IN REFRESH COUNT



On average 71% reduction in refresh count, very close to the upper bound of 75%