Using Commodity Memory Devices to Support Fundamental Security Primitives

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
25 March 2019
Bogazici University

SAFARI ETH zürich Carnegie Mellon

Brief Self Introduction

- Onur Mutlu
  - Full Professor @ ETH Zurich CS, since September 2015 (officially May 2016)
  - Strecker Professor @ Carnegie Mellon University ECE/CS, 2009-2016, 2016-…
  - PhD from UT-Austin, worked at Google, VMware, Microsoft Research, Intel, AMD
  - https://people.inf.ethz.ch/omutlu/
  - omutlu@gmail.com (Best way to reach me)
  - https://people.inf.ethz.ch/omutlu/projects.htm
- Research and Teaching in:
  - Computer architecture, computer systems, hardware security, bioinformatics
  - Memory and storage systems
  - Hardware security, safety, predictability; fault tolerance
  - Hardware/software cooperation
  - Architectures for bioinformatics, health, medicine
  - New computation, communication, storage paradigms
  - …

SAFARI Research Group

Think BIG, Aim HIGH!

https://safari.ethz.ch

SAFARI Group Members @ ETH Zurich


Dr. Mohammed Alser


Dr. Lois Orosa


Dr. Yaohua Wang


Dr. Juan Gómez-Luna

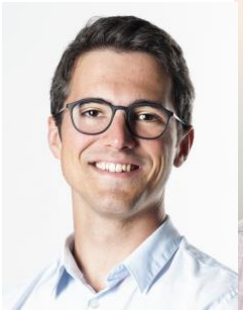
- 4 Post-doctoral Researchers
- 8 PhD Students + 4 at CMU
- 5 Interns
- 15 Master's and Bachelor's Researchers


Jeremie Kim


Hasan Hassan


Minesh Patel


Ivan Puddu


Lukas Breitwieser


Giray Yaglikci


Can Firtina


Geraldo F. de Oliveira

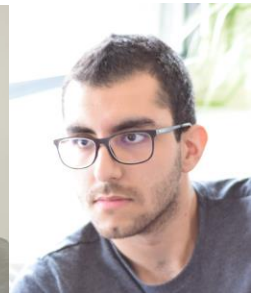

Nika Mansouri


Skanda Koppula


Konstantinos Kanellopoulos


Nisa Bostanci


Ataberk Olgun


Rokneddin Azizi


Christina Giannoula


Taha Shahroodi

Teaching: Accelerated Memory Course (~6.5 hours)

- ACACES 2018
  - Memory Systems and Memory-Centric Computing Systems
  - Taught by Onur Mutlu July 9-13, 2018
  - ~6.5 hours of lectures

- Website for the Course including Videos, Slides, Papers
  - https://people.inf.ethz.ch/omutlu/acaces2018.html
  - https://www.youtube.com/playlist?list=PL5Q2soXY2Zi-HXxomthrpDpMJm05P6J9x

- All Papers are at:
  - https://people.inf.ethz.ch/omutlu/projects.htm
  - Final lecture notes and readings (for all topics)

Teaching: Online Courses and Lectures

- **Freshman Digital Circuits and Computer Architecture Course Lecture Videos (2018, 2017)**
- **Graduate Computer Architecture Course Lecture Videos (2018, 2017, 2015, 2013)**
- **Undergraduate Computer Architecture Course Lecture Videos (2015, 2014, 2013)**
- **Parallel Computer Architecture Course Materials (Lecture Videos)**
- https://people.inf.ethz.ch/omutlu/teaching.html
- https://www.youtube.com/channel/UCIwQ8uOeRFgOEvBLYc3kc3g
- https://www.youtube.com/user/cmu18447

Research & Teaching: Some Overview Talks

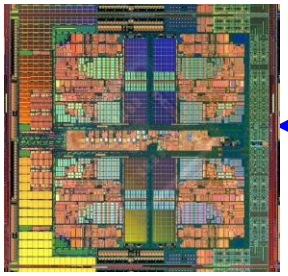
https://www.youtube.com/watch?v=kgiZlSOcGFM&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl

- Future Computing Architectures
  - https://www.youtube.com/watch?v=kgiZlSOcGFM&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=1
- Enabling In-Memory Computation
  - https://www.youtube.com/watch?v=oHqsNbxgdzM&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=7
- Accelerating Genome Analysis
  - https://www.youtube.com/watch?v=hPnSmfwu2-A&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=9
- Rethinking Memory System Design
  - https://www.youtube.com/watch?v=F7xZLNMIY1E&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=3

Research: Broad Perspective

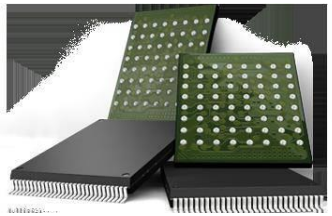
***Research Focus:* Computer architecture, HW/SW, security, bioinformatics**

- ***Memory and storage (DRAM, flash, emerging), interconnects***
- ***Heterogeneous & parallel systems, GPUs, systems for data analytics***
- ***System/architecture interaction, new execution models, new interfaces***
- ***Energy efficiency, fault tolerance, hardware security, performance***
- ***Genome sequence analysis & assembly algorithms and architectures***
- ***Biologically inspired systems & system design for bio/medicine***

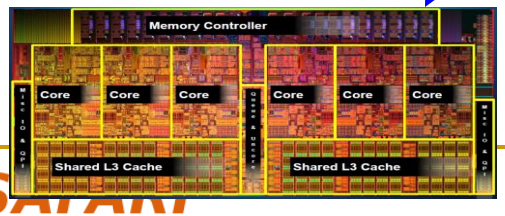


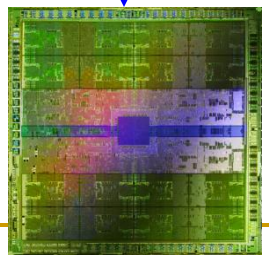
Heterogeneous Processors and Accelerators

Hybrid Main Memory



Persistent Memory/Storage





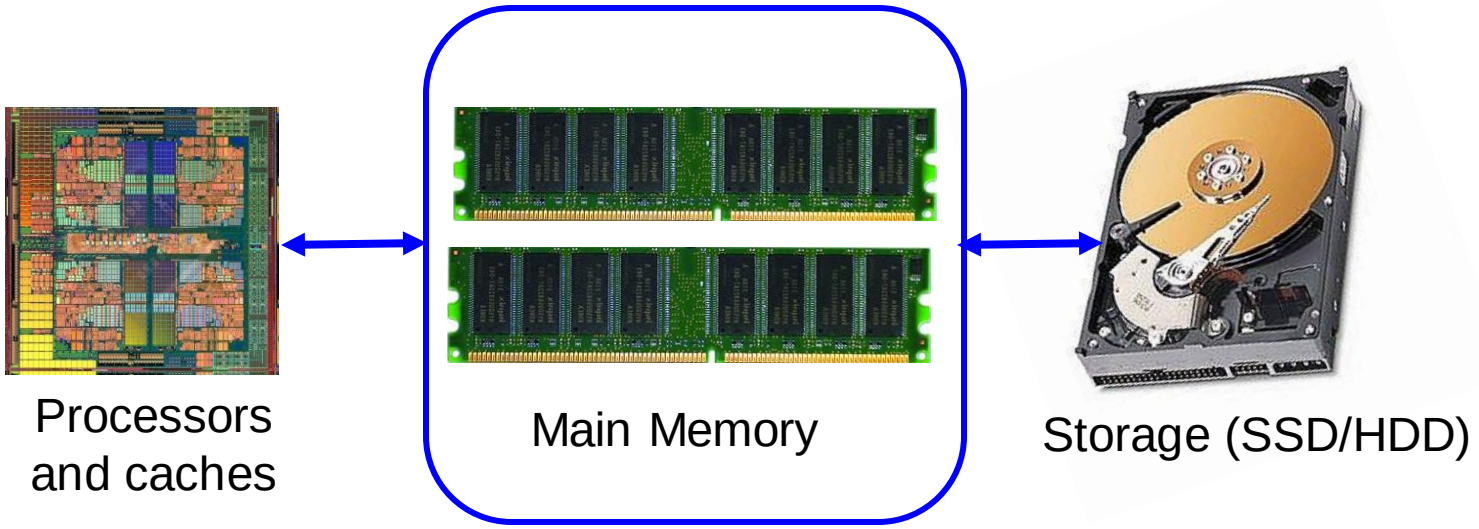
Graphics and Vision Processing

Broad research spanning apps, systems, logic with architecture at the center

Four Major Current Directions

- Fundamentally Secure/Reliable/Safe Architectures
- Fundamentally Energy-Efficient Architectures
  - Memory-centric (Data-centric) Architectures
- Fundamentally Low-Latency Architectures
- Architectures for Genomics, Medicine, Health

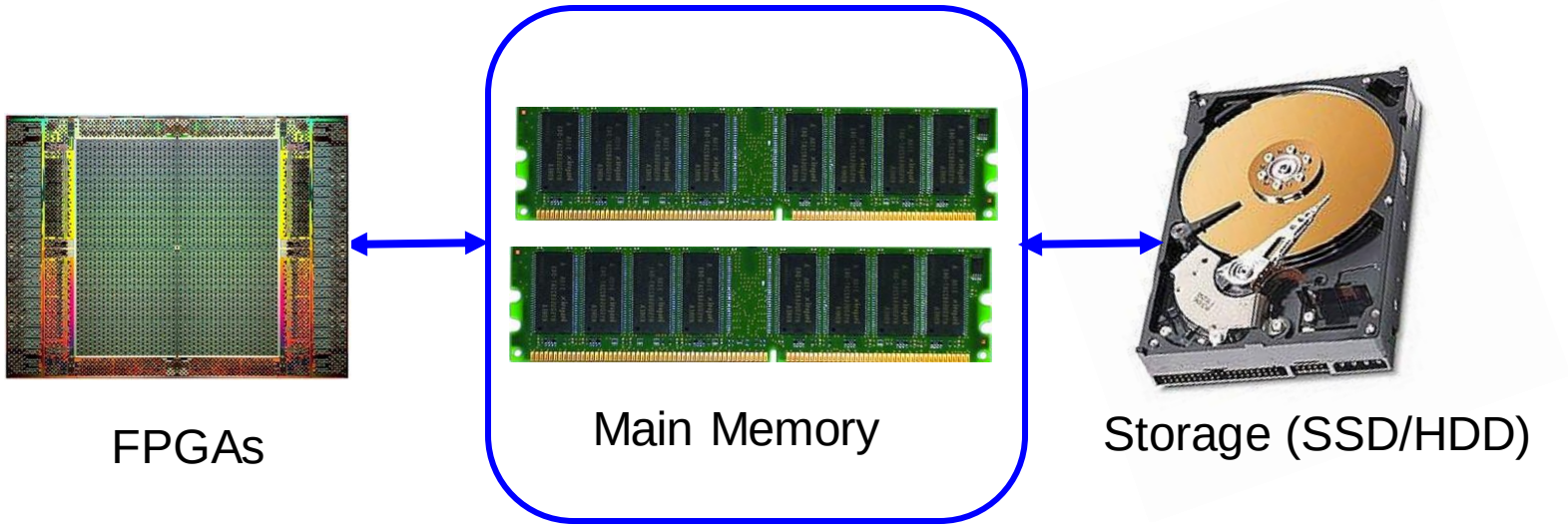
The Main Memory System



- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

- Devices that make up main memory are ubiquitous in all systems

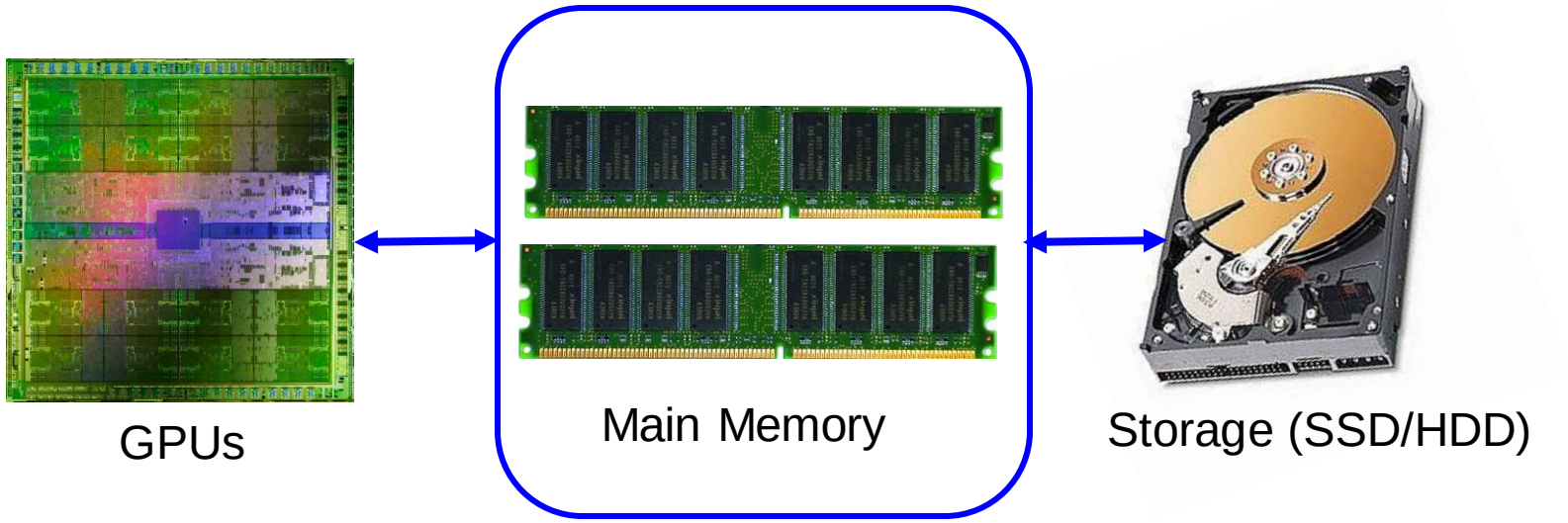
The Main Memory System



- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

- Devices that make up main memory are ubiquitous in all systems

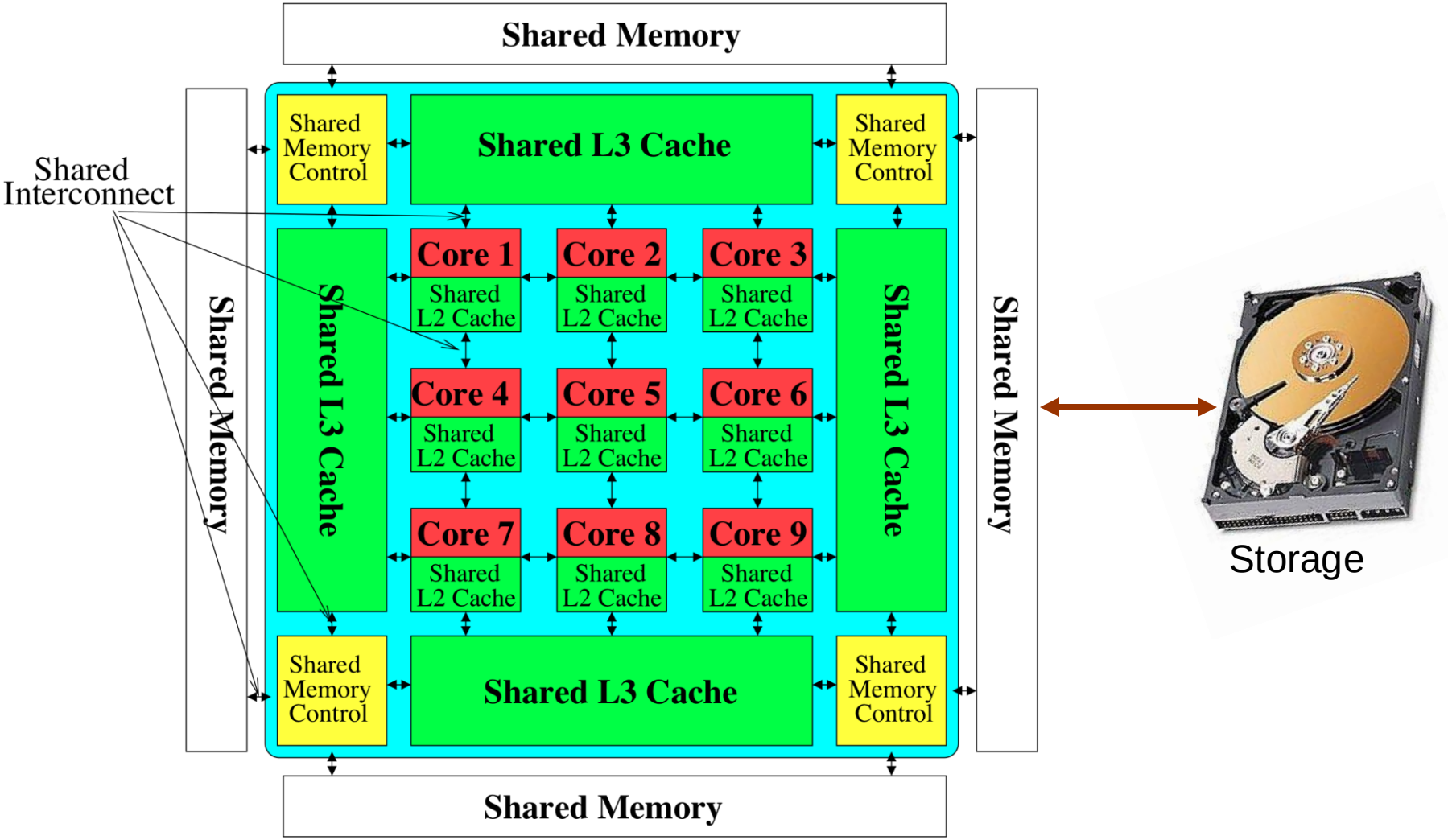
The Main Memory System



- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

- Devices that make up main memory are ubiquitous in all systems

The Main Memory System



Most of the system is dedicated to storing and moving data

The Shortcoming

- Most of the system is dedicated to main memory
- Main memory is in all systems

- Main memory devices are used to **only** store and move data

- Can we do better?
- Can we better take advantage of the main memory devices?

Doing Better with Memory Devices

- **Make the memory devices more intelligent**

  → minimize data movement, exploit parallel processing

  - Processing in memory
  - See my past Bogazici talks and many works:
    - RowClone [MICRO 2013], Ambit [MICRO 2017], Tesseract [ISCA 2015], PEI [ISCA 2015], TOM [ISCA 2016], EMC [ISCA 2016], Google Workloads [ASPLOS 2018], LazyPIM/CoNDA [ISCA 2019], …

Processing in Memory (I)

- Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu, **"Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks"** *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**)*, Williamsburg, VA, USA, March 2018.

62.7% of the total system energy is spent on data movement

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand[1] Saugata Ghose[1] Youngsok Kim[2]
Rachata Ausavarungnirun[1] Eric Shiu[3] Rahul Thakur[3] Daehyun Kim[4,3]
Aki Kuusela[3] Allan Knies[3] Parthasarathy Ranganathan[3] Onur Mutlu[5,1]

Processing Data Where It Makes Sense in Modern Computing Systems: Enabling In-Memory Computation

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu

Feb. 21st 2019

SAFARI ETH zürich Carnegie Mellon

Processing in Memory (III)

Processing Data Where It Makes Sense in Modern Computing Systems: Enabling In-Memory Computation

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
15 February 2019
GWU ECE Distinguished Lecture

SAFARI ETH zürich Carnegie Mellon

Processing in Memory (IV)

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu
onur@cmu.edu
http://users.ece.cmu.edu/~omutlu/
August 6, 2015
Bogazici University

Carnegie Mellon

SAFARI

Processing in Memory: Overview Paper

Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions

SAUGATA GHOSE, KEVIN HSIEH, AMIRALI BOROUMAND,
RACHATA AUSAVARUNGNIRUN
Carnegie Mellon University

ONUR MUTLU
ETH Zürich and Carnegie Mellon University

Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, Onur Mutlu, **"Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions"**
Invited Book Chapter, to appear in 2018.
[Preliminary arxiv.org version]

https://arxiv.org/pdf/1802.00320.pdf

Doing Better with Memory Devices

- **Make the memory devices more intelligent**

  → minimize data movement, exploit parallel processing

  - Processing in memory
  - See my past Bogazici talks and many works:
    - RowClone [MICRO 2013], Ambit [MICRO 2017], Tesseract [ISCA 2015], PEI [ISCA 2015], TOM [ISCA 2016], EMC [ISCA 2016], Google Workloads [ASPLOS 2018], LazyPIM/CoNDA [ISCA 2019], …

- **Use the memory devices to support key functions**

  - Security primitives
  - …

Key Goal

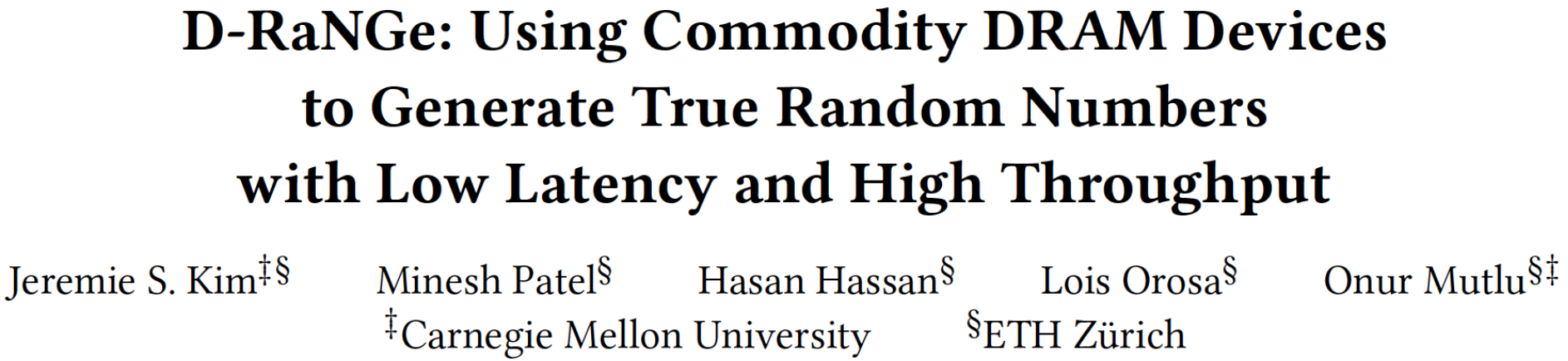
How to Use
Memory Devices
to Support Security

Using Memory for Security

- Generating True Random Numbers (using DRAM)
  - Kim et al., HPCA 2019
- Evaluating Physically Unclonable Functions (using DRAM)
  - Kim et al., HPCA 2018
- Quickly Destroying In-Memory Data (using DRAM)
  - Orosa et al., arxiv.org 2019

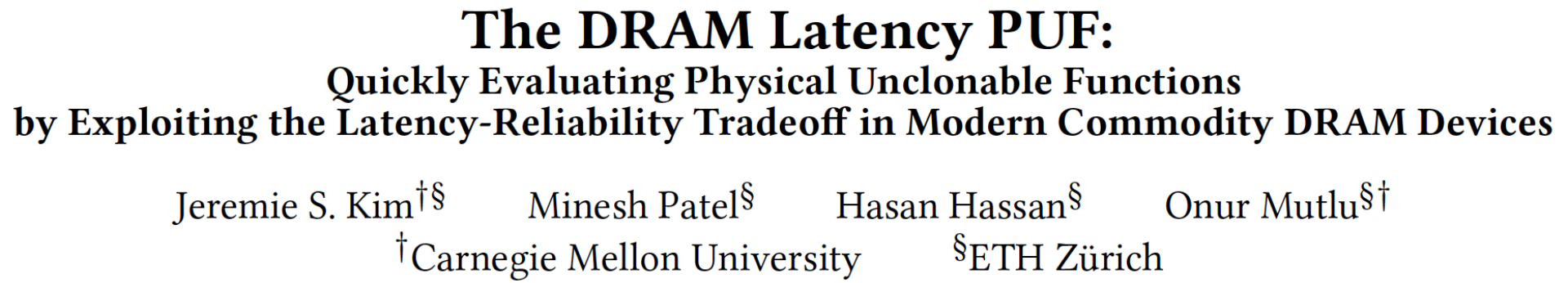
Generating True Random Numbers

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu, **"D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput"** *Proceedings of the 25th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Washington, DC, USA, February 2019. [Slides (pptx) (pdf)]
  [Full Talk Video (21 minutes)]



Evaluating Physically Unclonable Functions

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu,
  "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern DRAM Devices"
  *Proceedings of the 24th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Vienna, Austria, February 2018.
  [Lightning Talk Video]
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]



Quickly Destroying In-Memory Data

- Dataplant: In-DRAM Security Mechanisms for Low-Cost Devices
  - https://arxiv.org/pdf/1902.07344.pdf

Dataplant: In-DRAM Security Mechanisms for Low-Cost Devices

Lois Orosa[1] Yaohua Wang[1,2] Ivan Puddu[1] Mohammad Sadrosadati[1,3]
Kaveh Razavi[1,4] Juan Gómez-Luna[1] Hasan Hassan[1] Nika Mansouri-Ghiasi[1]
Arash Tavakkol[1] Minesh Patel[1] Jeremie Kim[1,5] Vivek Seshadri[6]
Uksong Kang[7] Saugata Ghose[5] Rodolfo Azevedo[8] Onur Mutlu[1,5]

[1]*ETH Zürich* [2]*National University of Defense Technology* [3]*Sharif University of Technology*
[4]*Vrije Universiteit Amsterdam* [5]*Carnegie Mellon University* [6]*Microsoft* [7]*SK Hynix* [8]*UNICAMP*

Using Memory for Security

- Generating True Random Numbers (using DRAM)
  - Kim et al., HPCA 2019

- Evaluating Physically Unclonable Functions (using DRAM)
  - Kim et al., HPCA 2018

- Quickly Destroying In-Memory Data (using DRAM)
  - Orosa et al., arxiv.org 2019

D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput

Jeremie S. Kim **Minesh Patel**

Hasan Hassan **Lois Orosa** **Onur Mutlu**

SAFARI

Carnegie Mellon

ETH zürich

Executive Summary

- **Motivation**: High-throughput true random numbers enable system security and various randomized algorithms.
  - Many systems (e.g., IoT, mobile, embedded) do not have dedicated **True Random Number Generator (TRNG)** hardware but have DRAM devices
- **Problem**: Current DRAM-based TRNGs either
  1. do **not** sample a fundamentally non-deterministic entropy source
  2. are **too slow** for continuous high-throughput operation
- **Goal**: A novel and effective TRNG that uses **existing** commodity DRAM to provide random values with 1) **high-throughput,** 2) **low latency** and 3) no adverse effect on concurrently running applications
- **D-RaNGe:** Reduce DRAM access latency **below reliable values** and exploit DRAM cells' failure probabilities to generate random values
- **Evaluation:**
  1. Experimentally characterize **282 real LPDDR4 DRAM devices**
  2. **D-RaNGe** (**717.4 Mb/s**) has significantly higher throughput (**211x**)
  3. **D-RaNGe** (**100ns**) has significantly lower latency (**180x**)

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

Motivation and Goal

- High throughput **True Random Numbers** are required for many real-world applications
  - Importantly **cryptography** for securely encrypting file systems, network packets, data in standard protocols (TLS/SSL/RSA…)
  - Others include randomized algorithms, scientific simulation, statistical sampling, recreational entertainment

- **True random numbers** can only be generated via **physical processes**
  - e.g., radioactive decay, thermal noise, shot noise
  - Systems rely on **dedicated TRNG Hardware** that samples non-deterministic **various physical phenomena**

Motivation and Goal

- Smaller devices (e.g., IoT, mobile, embedded) **require**, but **often lack**, a high throughput **True Random Number Generator (TRNG)**
- DRAM devices are available on most systems
- Mechanism that generates TRN using DRAM enables:
  1. applications that **require true random numbers** to now run on most systems
  2. other use-cases, e.g., **processing-in-memory applications** to generate true random numbers within memory itself
- **Our Goal:** to provide a **TRNG** using DRAM devices that satisfies the characteristics of an effective TRNG

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

Effective TRNG Characteristics

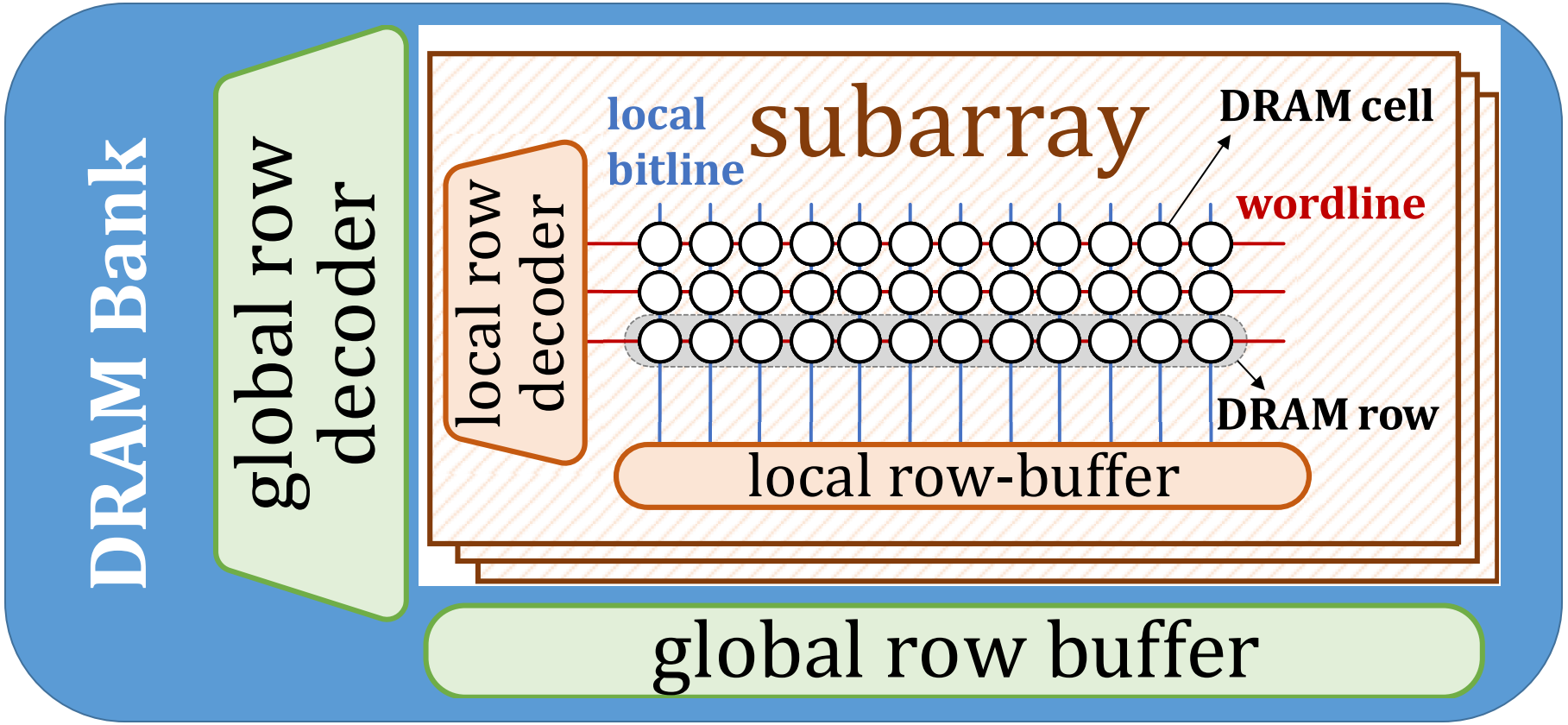
1. Low **implementation cost**
2. Fully **non-deterministic**
   - impossible to predict the next output given complete information about how the mechanism operates
3. Provide a continuous stream of true random numbers with **high throughput**
4. Provide true random numbers with **low latency**
5. Exhibit **low system interference**
   - not significantly slow down concurrently-running applications
6. Generate random values with **low energy overhead**

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

DRAM Organization

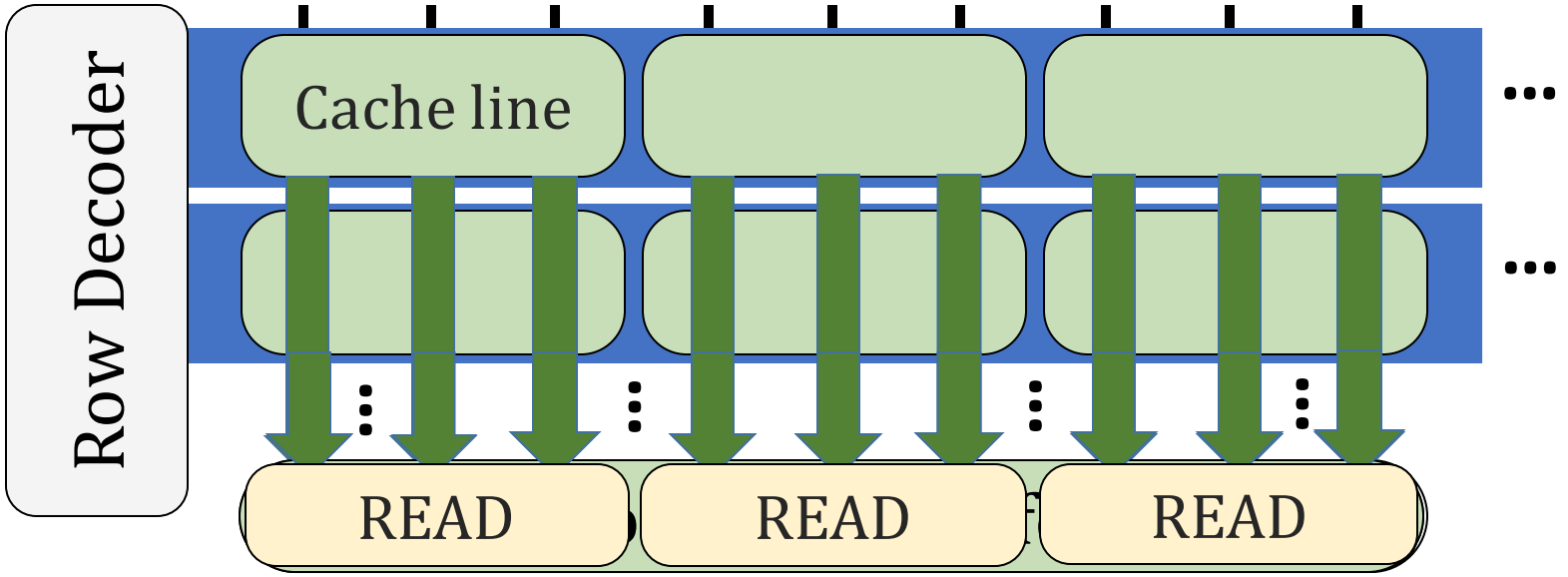
A DRAM bank is hierarchically organized into **subarrays**



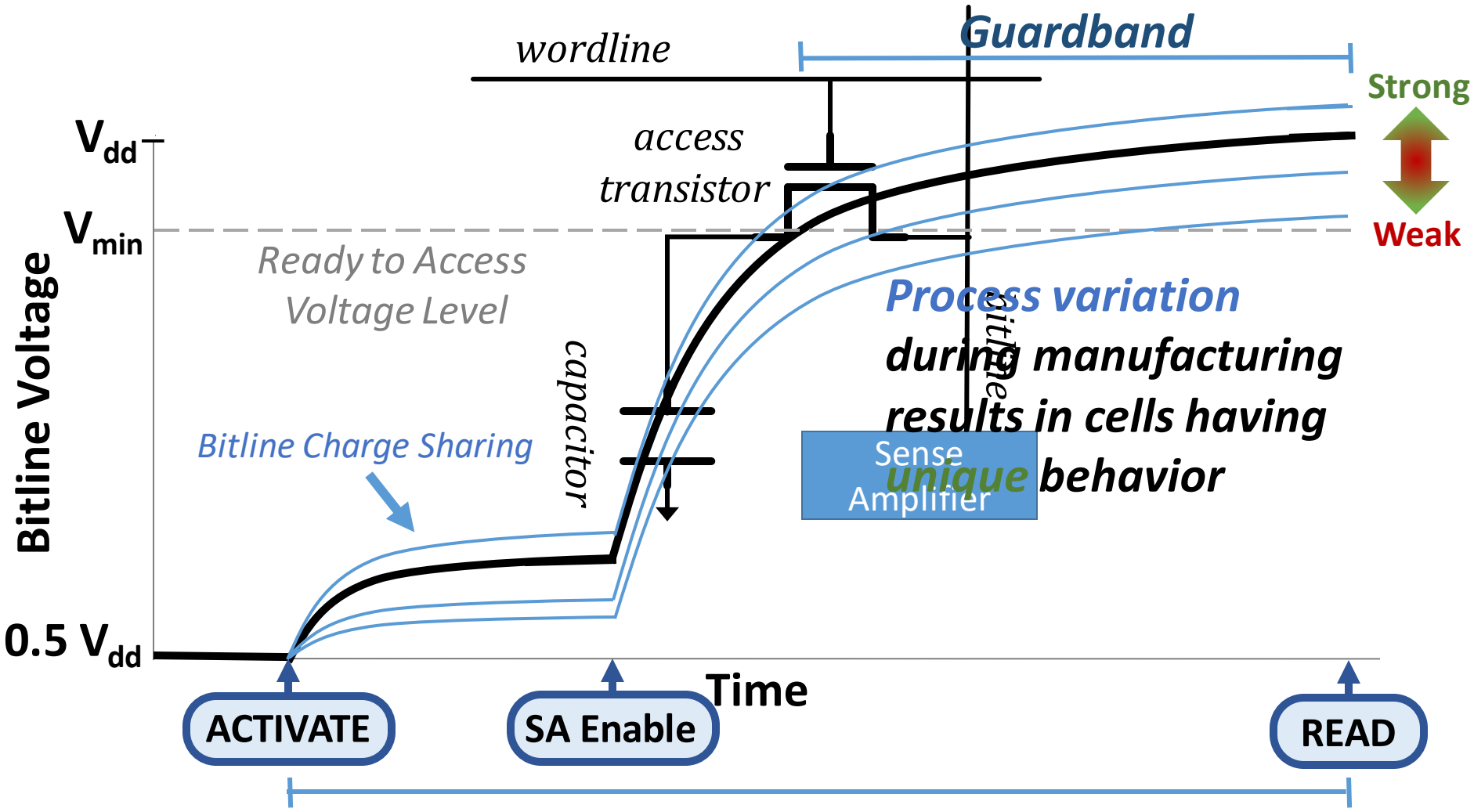
Columns of cells in subarrays share a **local bitline**

Rows of cells in a subarray share a **wordline**

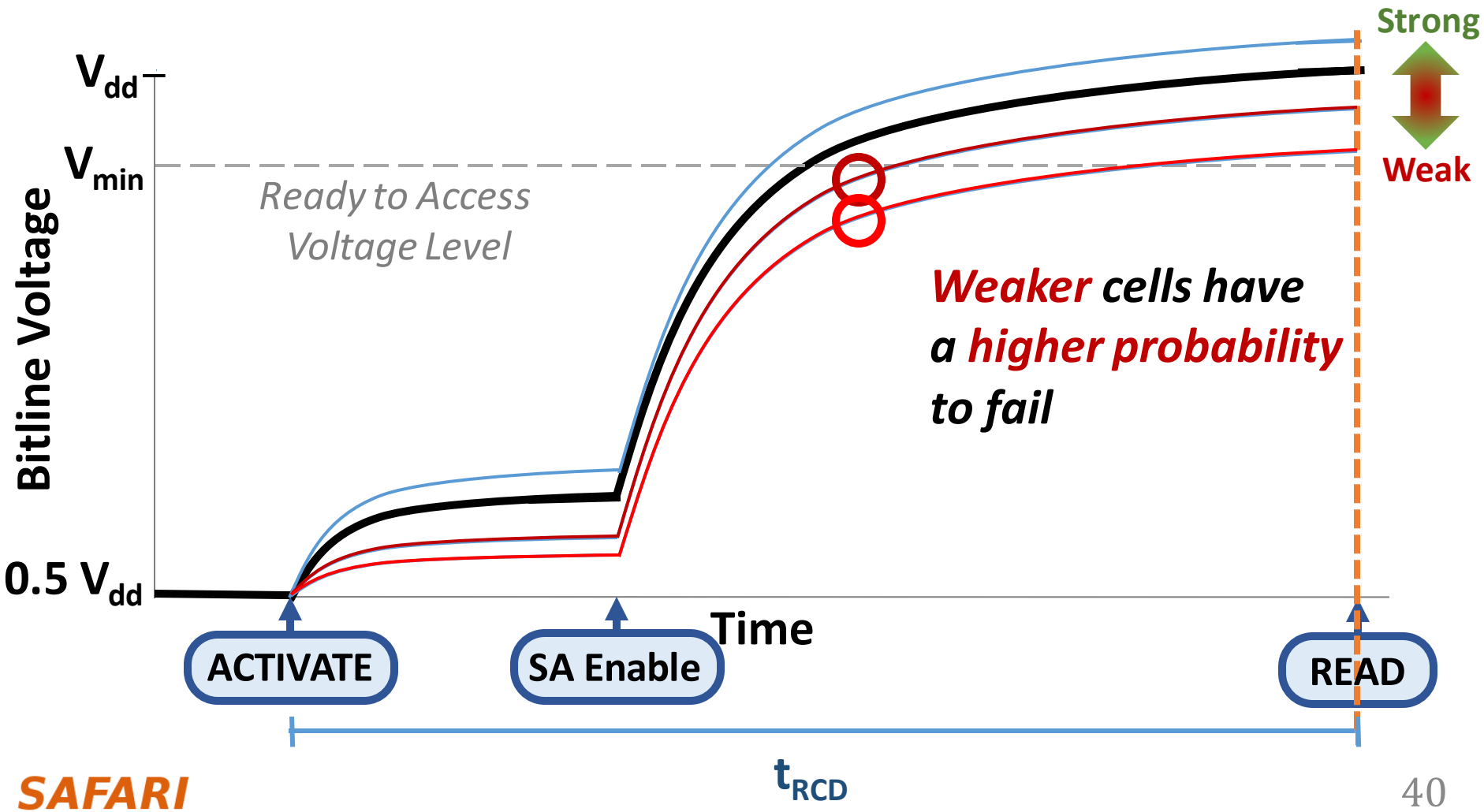
DRAM Operation



DRAM Accesses and Failures



DRAM Accesses and Failures

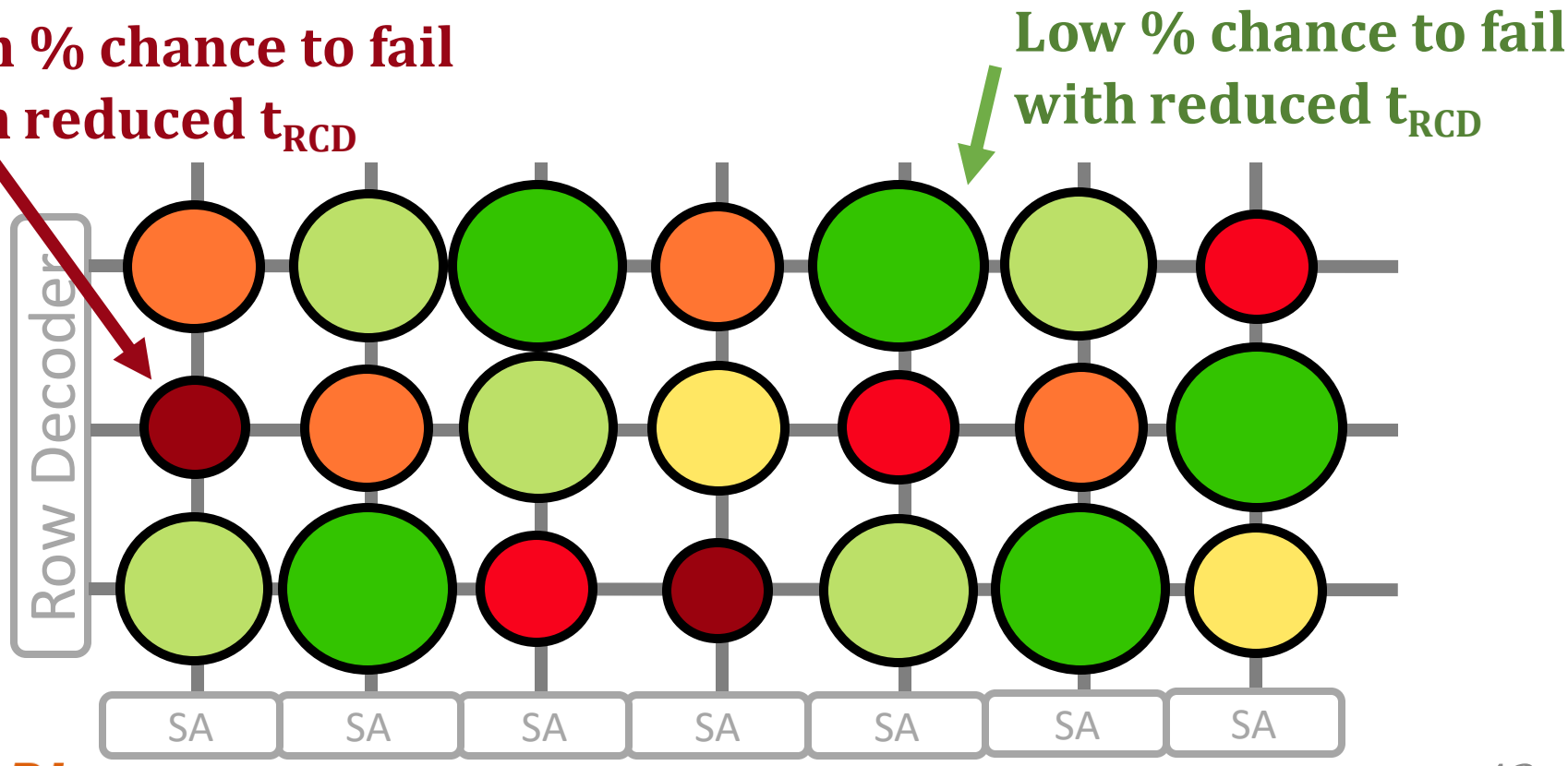


D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

D-RaNGe Key Idea

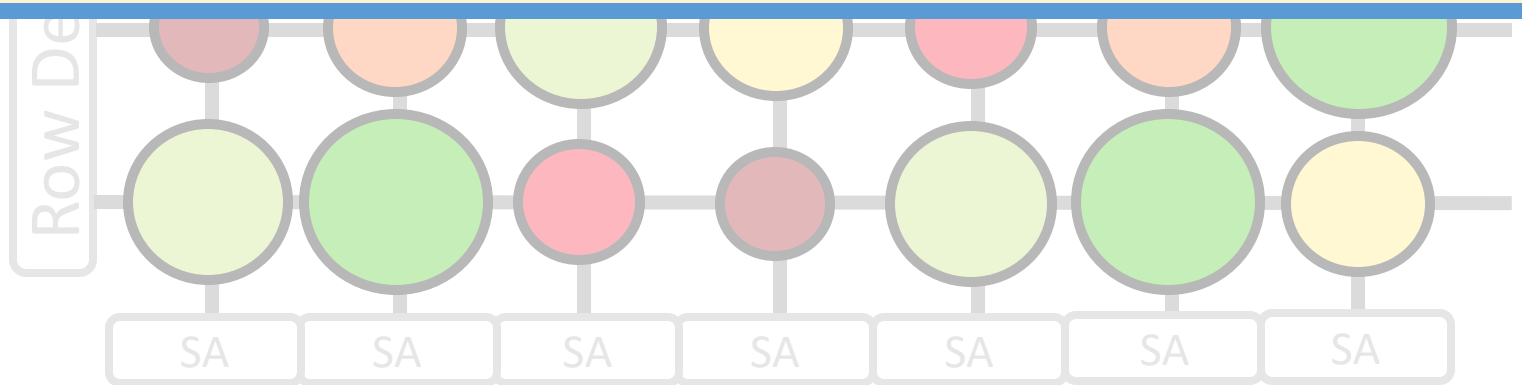
- A cell's latency failure probability is inherently related to **random process variation** from manufacturing
- We can extract **random values** by observing DRAM cells' latency failure probabilities



D-RaNGe Key Idea

- A cell's latency failure probability is inherently related to **random process variation** from manufacturing
- We can extract **random values** by observing DRAM

The key idea is to extract random values by sampling DRAM cells that fail truly randomly

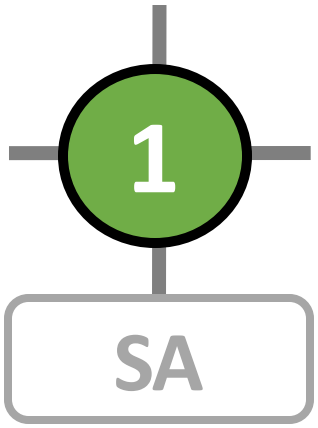


D-RaNGe: Extracting Random Values

Identify all DRAM cells that fail randomly when accessed with a reduced $\mathbf{t_{RCD}}$ (**RNG Cell**)

- When accessing an RNG Cell with a reduced $\mathbf{t_{RCD}}$, the values read will be truly random values

RNG Cell



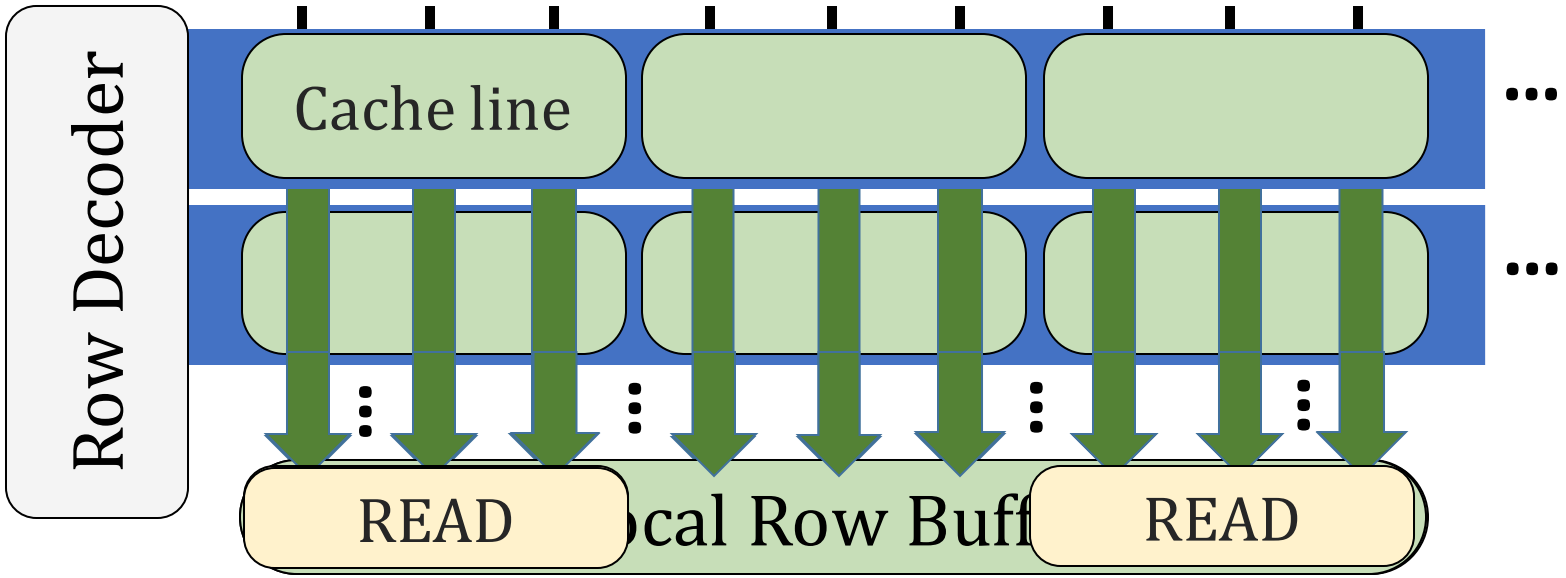
Random values when accessed with $\mathbf{t_{RCD}}$ reduced by **45%**

D-RaNGe: Identifying RNG Cells

- To identify RNG Cells, extract 1M values (**bitstream**) from each DRAM cell
- An **RNG Cell** is a DRAM cell whose output passes the NIST statistical test suite for randomness
- NIST tests **[Rukhin+, Tech report, 2001]** include tests for:
  - Unbiased output of 1's and 0's across entire bitstream
  - Unbiased output within smaller segments of the bitstream
  - Limited number of uninterrupted sequence of identical bits
  - Peak heights in the discrete fourier transform of bitstream
  - Even distribution of short sequences within bitstream
  - Cumulative sum always stays close to zero
  - …

D-RaNGe: Access Pattern

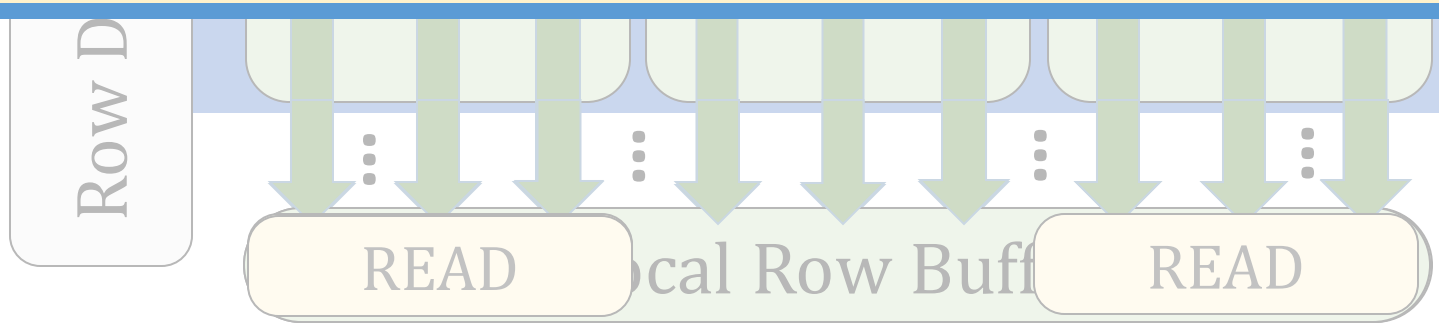
- To maximize the bits that are accessed **immediately following activation**, we alternate accesses to distinct rows **in each bank**
  - **quickly** generate tRCD failures within cache lines in two rows
  - **maximizes** tRCD failures when using reduced tRCD



D-RaNGe: Access Pattern

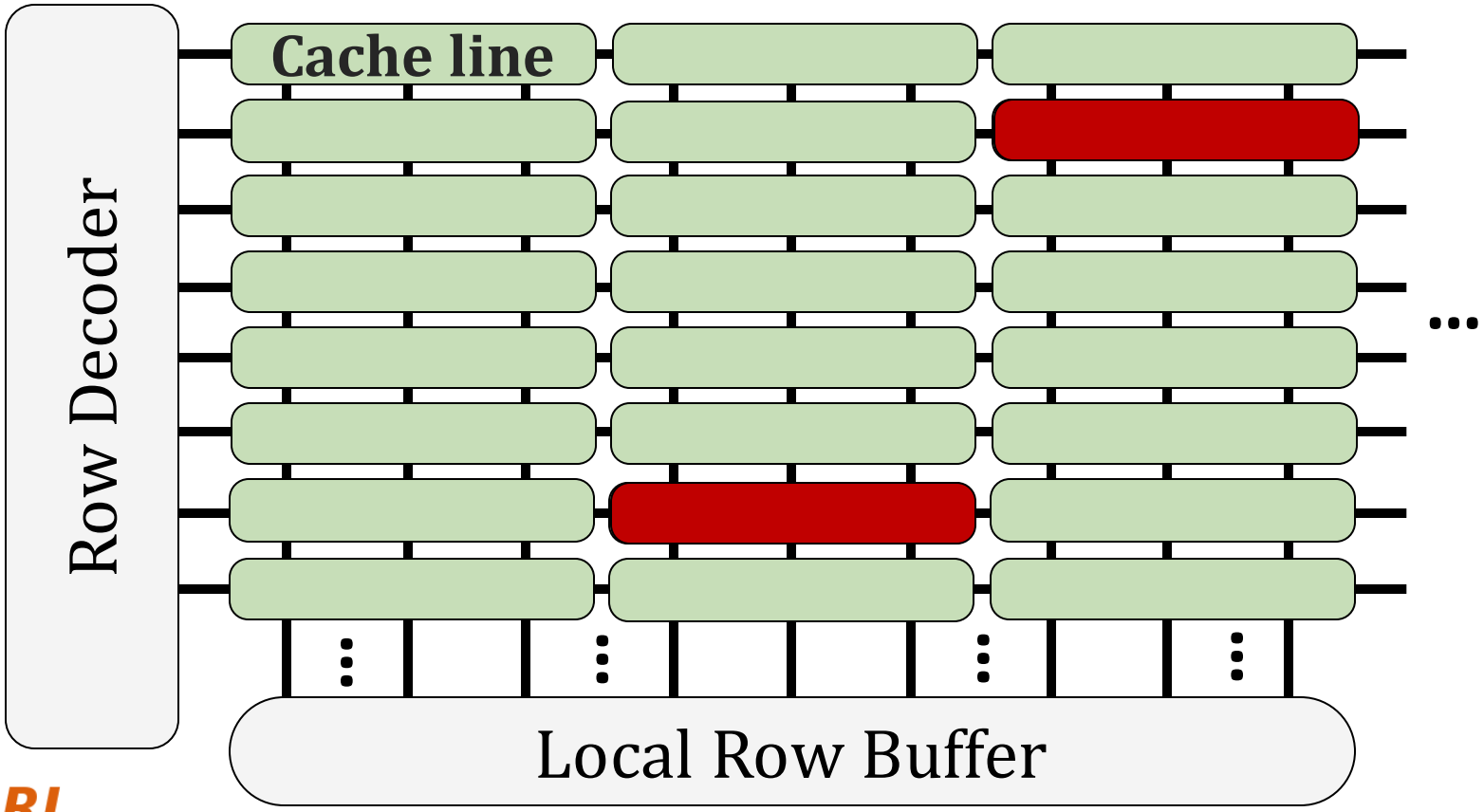
- To maximize the bits that are accessed **immediately following activation**, we alternate

Accessing cache lines containing more RNG cells will result in more random values



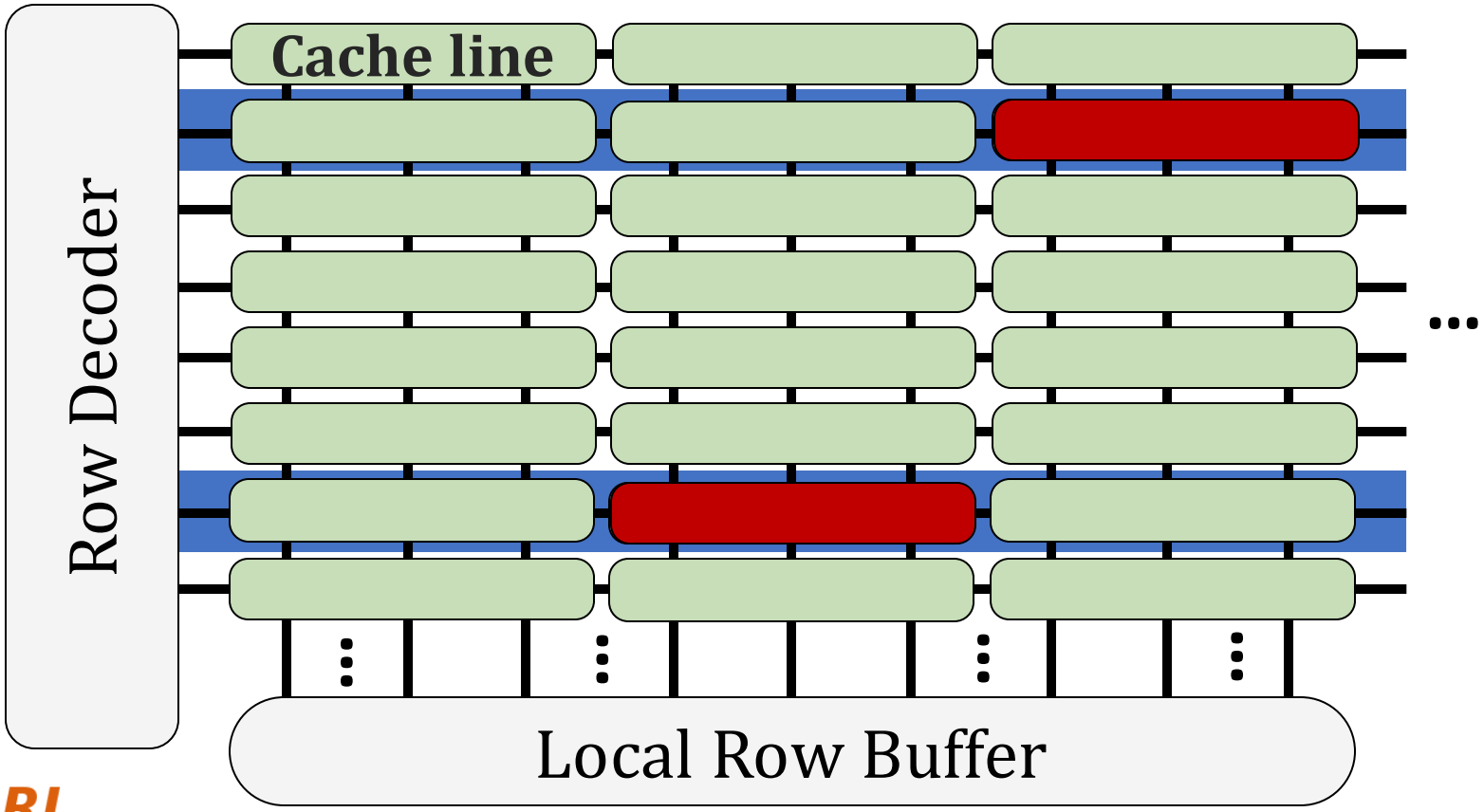
D-RaNGe: Exclusive Access

- To minimize system interference, D-RaNGe has **exclusive access** to RNG cells
- **In a bank**, find the **two cache lines** in distinct rows with the most number of RNG cells



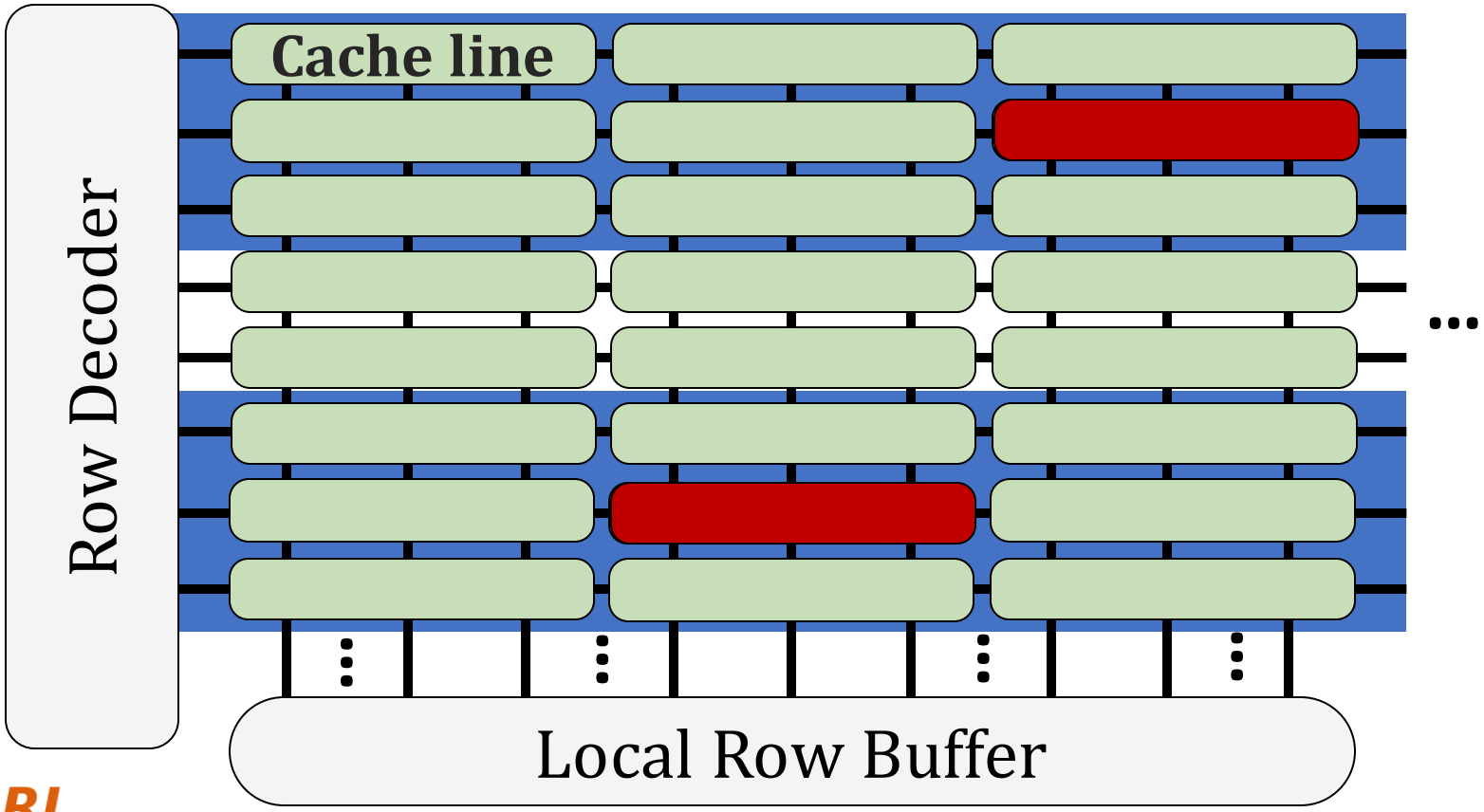
D-RaNGe: Exclusive Access

Reserve rows containing selected cache lines exclusively for D-RaNGe accesses to minimize interference



D-RaNGe: Exclusive Access

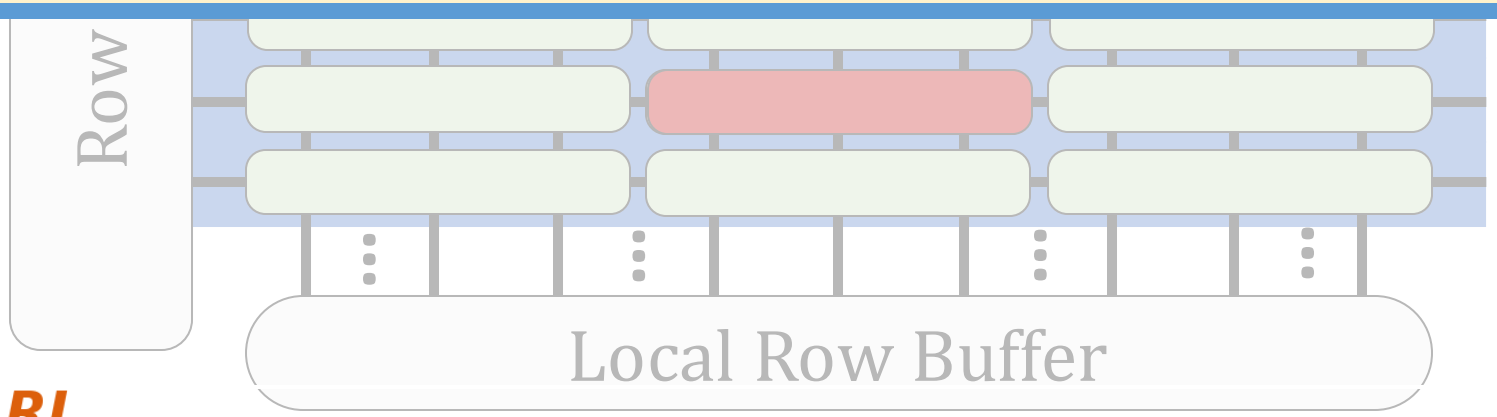
- **Reserve neighboring rows to minimize DRAM data pattern/read interference**



D-RaNGe: Exclusive Access

- Cache lines containing more RNG cells provide more random bits of data per access
- In a bank, find the **two cache lines** in distinct rows with

We can parallelize accesses across all available DRAM banks for higher throughput of random values



D-RaNGe: Example Implementation

- Memory controller **reserves rows** containing selected RNG cells and neighboring rows
- When system not accessing a bank, memory controller runs D-RaNGe firmware to generate random values in the bank
- Memory controller has **buffer of random data**
- Stores random values in memory controller buffer
- Expose **API** for returning random values from the buffer when requested by the user

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- **Methodology**
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

Methodology

- **282 2y-nm LPDDR4 DRAM devices**
  - **2GB** device size from **3 major DRAM manufacturers**
- **Thermally controlled testing chamber**
  - Ambient temperature range: **{40°C – 55°C} ± 0.25°C**
  - DRAM temperature is held at 15°C above ambient
- **Control over DRAM commands/timing parameters**
  - Test reduced latency effects by **reducing t_{RCD} parameter**
- **Cycle-level simulator:** Ramulator [Kim+, CAL'15]
  https://github.com/CMU-SAFARI/ramulator
  - SPEC CPU2006 workloads, 4-core
- **DRAM Energy:** DRAMPower [Chandrasekar+, '12]
  http://www.es.ele.tue.nl/drampower/
  - Using output from Ramulator

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

Results – NIST Randomness Tests

How do we know whether D-RaNGe is truly random?

NIST Test Name	P-value	Status
monobit	0.675	PASS
frequency_within_block	0.096	PASS
runs	0.501	PASS
longest_run_ones_in_a_block	0.256	PASS
binary_matrix_rank	0.914	PASS
dft	0.424	PASS
non_overlapping_template_matching	>0.999	PASS
overlapping_template_matching	0.624	PASS
maurers_universal	0.999	PASS
linear_complexity	0.663	PASS
serial	0.405	PASS
approximate_entropy	0.735	PASS
cumulative_sums	0.588	PASS
random_excursion	0.200	PASS
random_excursion_variant	0.066	PASS

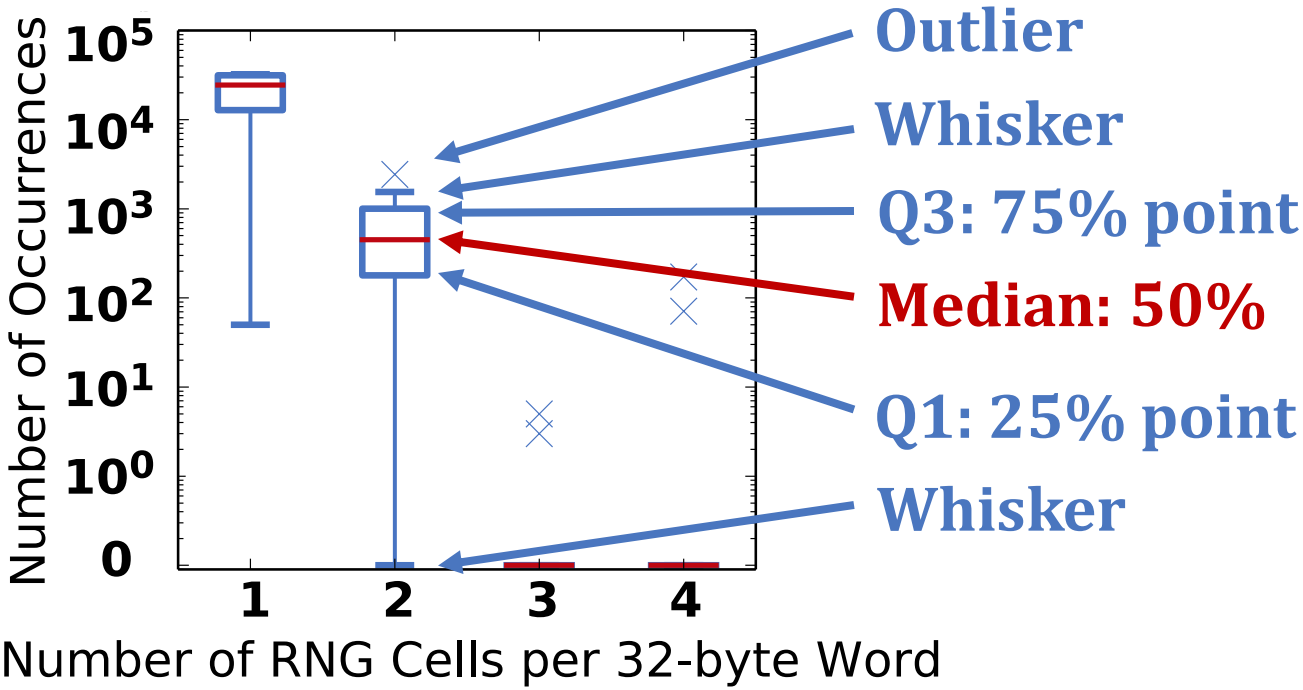
[Rukhin+, Tech report, 2001]

Passes all tests in NIST test suite for randomness!

More details in our HPCA 2019 paper

Results – 64-bit TRN Latency

Latency is related to density of available RNG cells per cache line

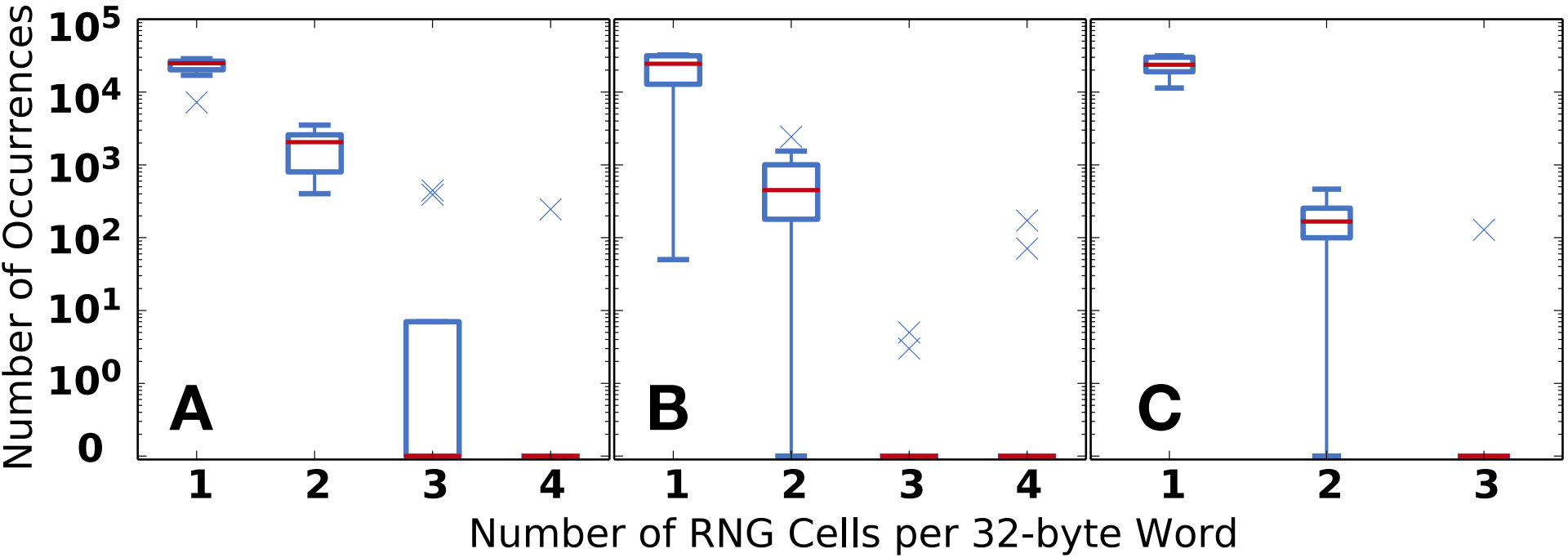


Across our devices, we analyze **availability of RNG cells** per cache line in a bank. Each point is the number of occurrences in a bank.

We plot the distribution across many banks as box-and-whisker plot

Results – 64-bit TRN Latency

Latency is related to density of available RNG cells per cache line



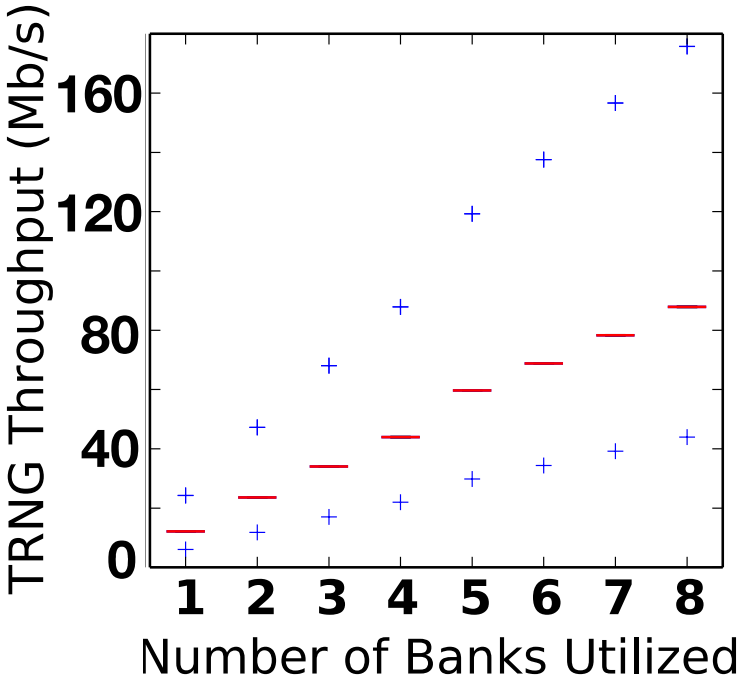
Maximum latency: **960 ns**

assuming 1 RNG cell / cache line from **a single bank**

Minimum empirical latency: **100 ns**

assuming 4 RNG cell / cache line in **all 32 banks in 4-channels**

Results – Single Channel Throughput

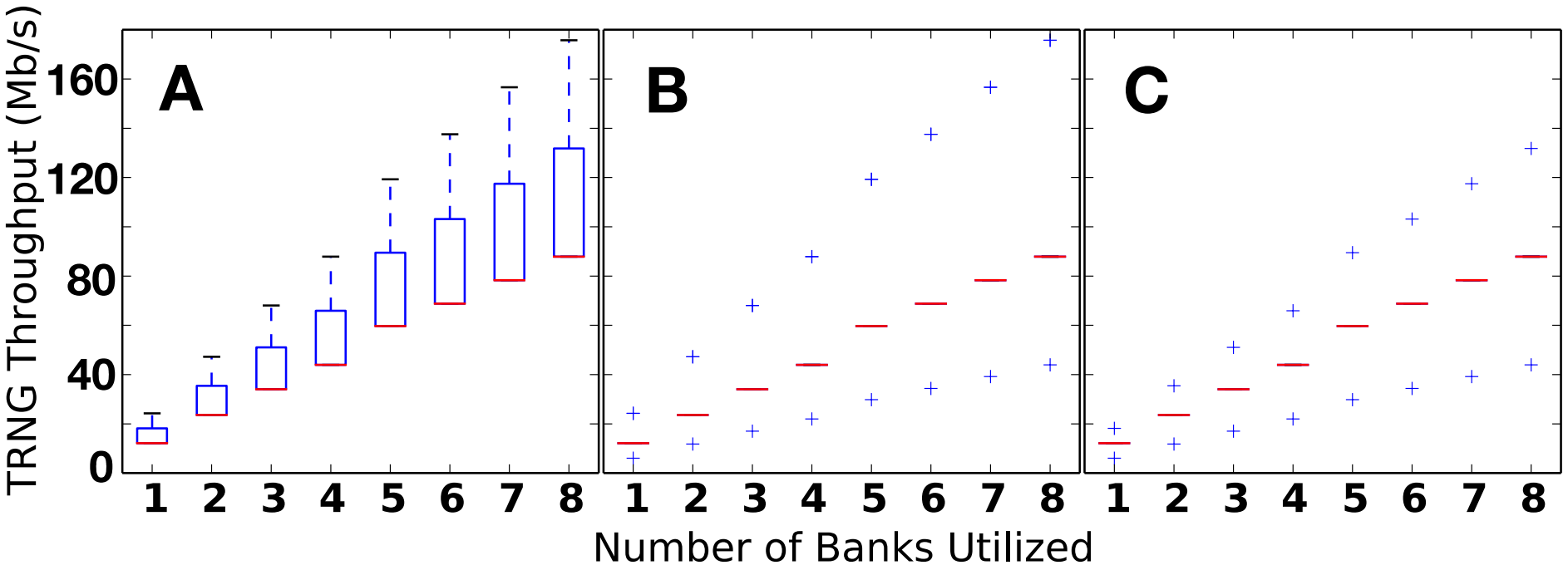


We determine **throughput** using the RNG cell densities found

For each bank utilized (x-axis), select the two cache lines containing the **most** number of RNG cells

$$Throughput = \frac{Accesses}{Second} \times \left(\sum_{i}^{selected\ cache\ lines} RNG\ Cell\ Density i\right)$$

Results – Single Channel Throughput



Since there are only between 1 and 4 RNG cells per cache line, there are a limited number of possible throughputs

- At least **40 Mb/s** when using all **8 banks** in a single channel
- Maximum throughput for **A/B/C: 179.4/179.4/134.5 Mb/s**
- 4-channel max (avg) throughput: **717.4 Mb/s (435.7 Mb/s)**

Results

- **System Interference**
  - **Capacity overhead:** 6 DRAM rows per DRAM bank (**~0.018%**)
  - D-RaNGe is flexible and can adjust its level of interference
  - D-RaNGe throughput with SPEC CPU2006 workloads in the **pessimistic** case where D-RaNGe only issues accesses to a DRAM bank when it is idle (no interference)
    - Average throughput of **83.1 Mb/s**
- **Energy Consumption**
  - **4.4 nJ/bit**
  - **Determined by Ramulator + DRAMPower**
    - https://github.com/CMU-SAFARI/ramulator
    - http://www.es.ele.tue.nl/drampower/

Other Results in the Paper

- **LPDDR4 DRAM Activation Failure Characterization**
  - Spatial distribution, data pattern dependence, temperature effects, variation over time
- **A detailed analysis on:**
  - Devices of **the three major DRAM manufacturers**
  - D-RaNGe energy consumption, 64-bit latency, throughput
- **Further discussion on:**
  - Algorithm for D-RaNGe to effectively generate random values
  - **Design considerations** for D-RaNGe
  - D-RaNGe overhead analysis
  - Analysis of NIST statistical test suite results
  - Detailed comparison against prior work

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

Prior Work: Command Scheduling

[Pyo+, IET, 2009]

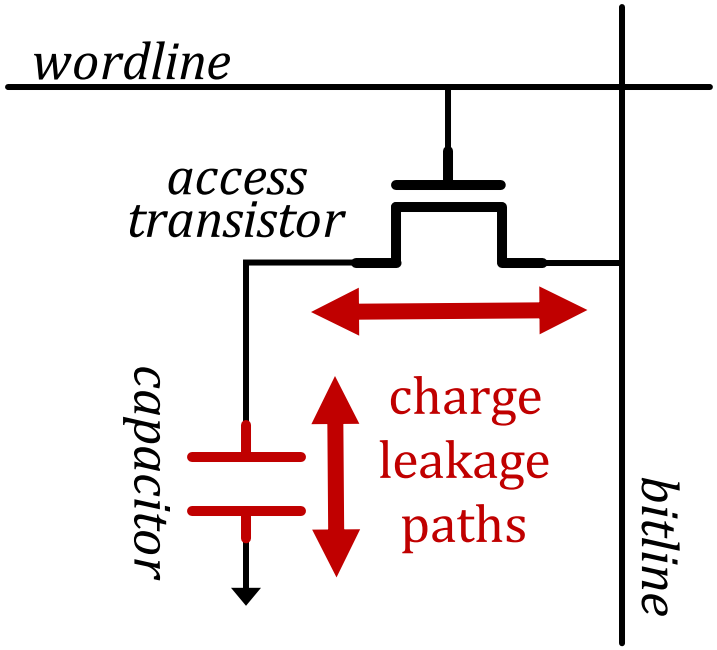
- **Randomness source**: time it takes to run a code segment of many DRAM accesses
  - Since time to access DRAM is **unpredictable** due to memory conflicts, refresh operations, calibration, etc.
  - Lower bits of the cycle timer used as random values
- Can produce random numbers at **3.4 Mb/s**
- **D-RaNGe** can produce TRNs at **>700Mb/s (211x higher)**
- **Downsides of DRAM Command Scheduling based TRNGs**
  - Randomness source is **not truly random**: depends on memory controller implementation and concurrently running applications
  - Much lower TRN throughput than **D-RaNGe**

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

DRAM Cell Leakage

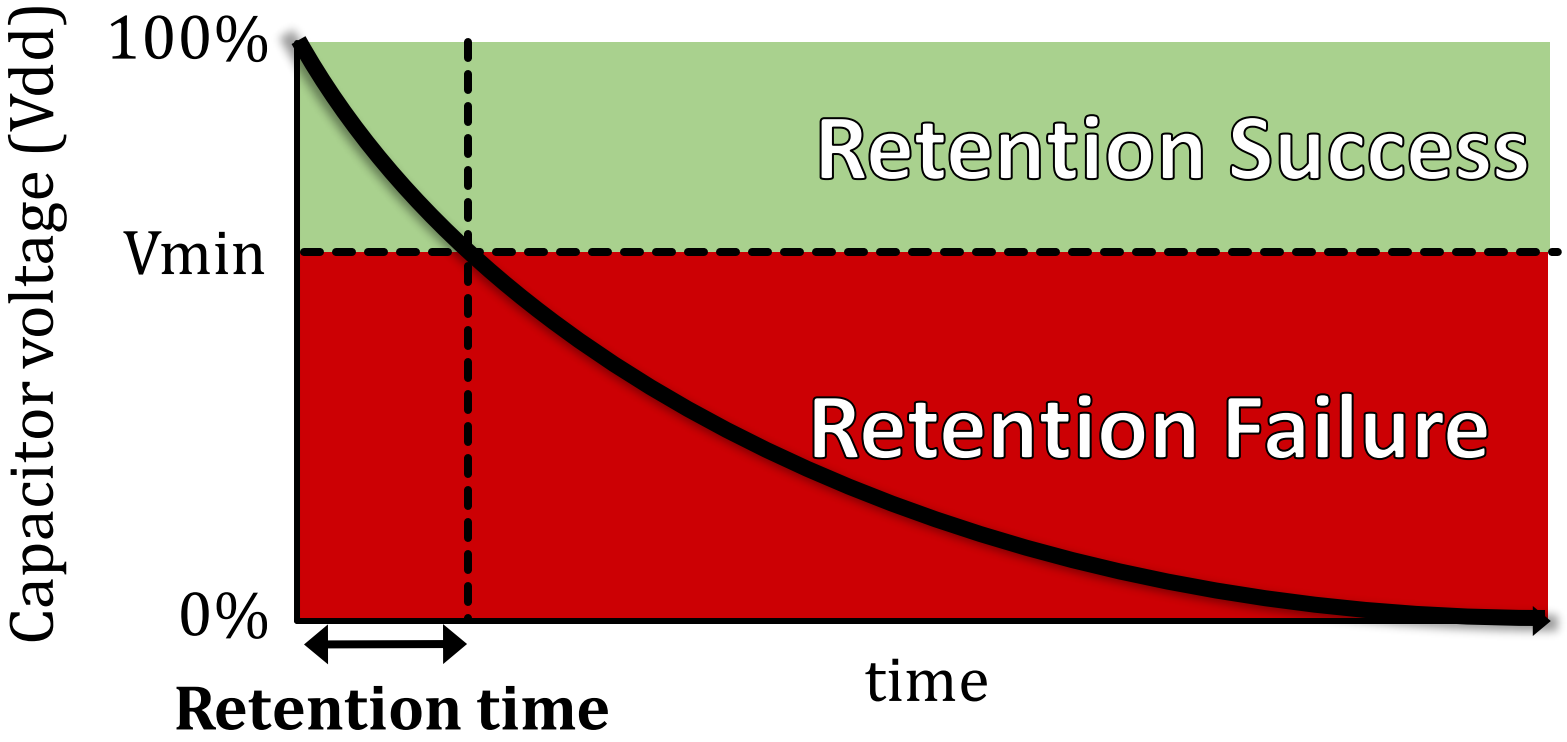
DRAM encodes information in **leaky** capacitors



Stored data is **corrupted** if too much charge leaks (i.e., the capacitor voltage degrades too much)

[Patel et al., REAPER, ISCA'17]

DRAM Cell Retention



Retention failure – when leakage corrupts stored data

Retention time – how long a cell holds its value

[Patel et al., REAPER, ISCA'17]

Data Retention in DRAM Cells [ISCA 2013]

- Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu, **"An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms"** *Proceedings of the 40th International Symposium on Computer Architecture (**ISCA**)*, Tel-Aviv, Israel, June 2013. Slides (ppt) Slides (pdf)

An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms

Jamie Liu*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
jamiel@alumni.cmu.edu

Ben Jaiyen*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
bjaiyen@alumni.cmu.edu

Yoongu Kim
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
yoonguk@ece.cmu.edu

Chris Wilkerson
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95054
chris.wilkerson@intel.com

Onur Mutlu
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
onur@cmu.edu

Data Retention in DRAM Cells [ISCA 2017]

- Minesh Patel, Jeremie S. Kim, and Onur Mutlu,
  "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions"
  *Proceedings of the 44th International Symposium on Computer Architecture (**ISCA**)*, Toronto, Canada, June 2017.
  [Slides (pptx) (pdf)]
  [Lightning Session Slides (pptx) (pdf)]

- First experimental analysis of (mobile) LPDDR4 chips
- Analyzes the complex tradeoff space of retention time profiling
- Idea: enable fast and robust profiling at higher refresh intervals & temperatures

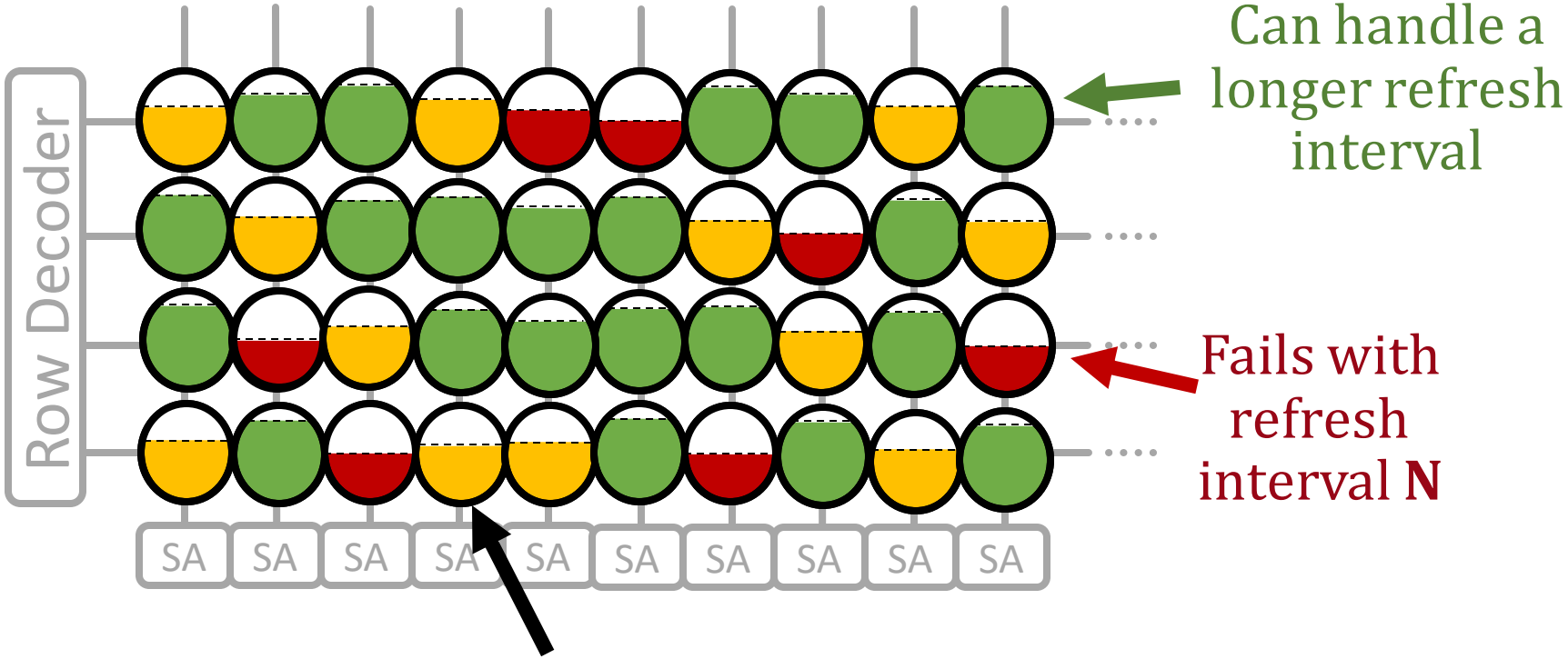
The Reach Profiler (REAPER):
Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions

Minesh Patel[§‡] Jeremie S. Kim[‡§] Onur Mutlu[§‡]
[§]ETH Zürich [‡]Carnegie Mellon University

Retention-based TRNGs

[Keller+, ISCAS, 2014] [Hashemian, DATE, 2015] [Sutar+, TECS, 2018]

Generate random values using data from cells that **fail randomly** with a **refresh interval *N***



After time N, some cells leak close to Vmin.

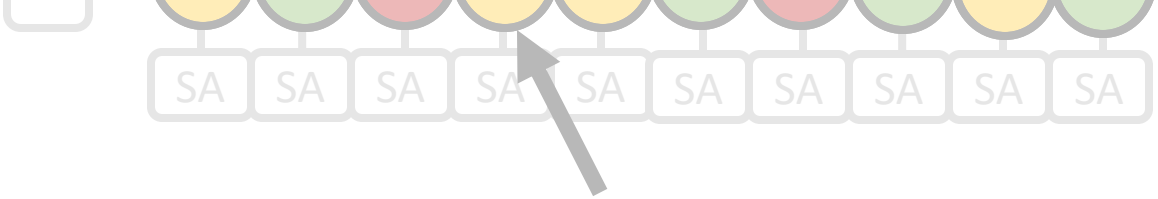
These RNG cells fail randomly

Retention-based TRNGs

[Keller+, ISCAS, 2014] [Hashemian, DATE, 2015] [Sutar+, TECS, 2018]

Generate random values using data from cells that **fail randomly** with a **refresh interval *N***

The key idea is to extract random values by aggregating values from RNG cells after every *increased* refresh interval N



After time N, some cells leak close to Vmin.

These RNG cells fail randomly

DRAM Retention TRNG Weaknesses

High latency

- Prior work shows that **40 sec** refresh interval results in 256 random bits of data per 4MiB DRAM block
- **D-RaNGe's** latency is **100ns** (>9 orders of magnitude faster)

Low Throughput / High DRAM capacity overhead

- Requires more capacity for higher throughput
  - Fully reserving a **32GB** DRAM device results in **0.05 Mb/s**
- **D-RaNGe** has **14,000x** higher throughput with a fixed capacity overhead **(384 KB)**

High energy consumption

- **6.8mJ/bit** mainly due to long idle periods
- **D-RaNGe**: **4.4 nJ/bit** (>7 orders of magnitude lower)

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

Start-up Values as Random Numbers

[Tehranipoor, HOST, 2016]

- When a device is powered up, some DRAM cells have **random values** due to interaction between
  - precharge logic
  - row decoder logic
  - column select lines
- Prior works propose **power cycling DRAM** to extract the random data resident in those cells

- **Downsides of DRAM Start-up value based TRNGs**
  - Must power cycle DRAM to generate random values:
    - **High latency:** based on power cycle time and data migration
    - **High storage cost:** all data must be migrated or will be lost

D-RaNGe Comparison against Prior Work

- **Compared to Command Scheduling, D-RaNGe:**
  - samples a **truly random entropy source**
  - **211x** higher throughput
  - **180x** lower latency
- **Compared to Retention Time, D-RaNGe:**
  - **>5** orders of magnitude higher throughput
  - **>9** orders of magnitude lower latency
  - **>7** orders of magnitude more energy efficient
- **Compared to Startup Values, D-RaNGe:**
  - **continuously** produces random values
  - does not require a system restart

D-RaNGe Outline

- Motivation
- Effective True Random Number Generators
- D-RaNGe
  - DRAM Operation
  - Key Idea
- Methodology
- Results
- Prior work on DRAM-based TRNGs
  - Command scheduling
  - Cell charge retention
  - Start-up values
- Summary

Summary and Conclusion

- **Motivation**: High-throughput true random numbers enable system security and various randomized algorithms.
  - Many systems (e.g., IoT, mobile, embedded) do not have dedicated **True Random Number Generator (TRNG)** hardware but have DRAM devices
- **Problem**: Current DRAM-based TRNGs either
  1. do **not** sample a fundamentally non-deterministic entropy source
  2. are **too slow** for continuous high-throughput operation
- **Goal**: A novel and effective TRNG that uses **existing** commodity DRAM to provide random values with 1) **high-throughput,** 2) **low latency** and 3) no adverse effect on concurrently running applications
- **D-RaNGe:** Reduce DRAM access latency **below reliable values** and exploit DRAM cells' failure probabilities to generate random values
- **Evaluation:**
  1. Experimentally characterize **282 real LPDDR4 DRAM devices**
  2. **D-RaNGe** (**717.4 Mb/s**) has significantly higher throughput (**211x**)
  3. **D-RaNGe** (**100ns**) has significantly lower latency (**180x**)

D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput

Jeremie S. Kim **Minesh Patel**

Hasan Hassan **Lois Orosa** **Onur Mutlu**

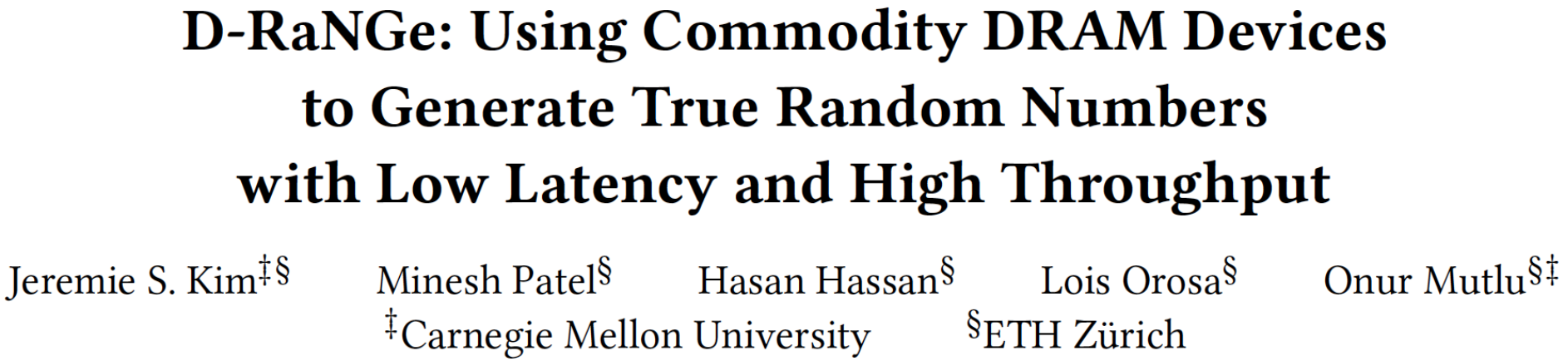
SAFARI

Carnegie Mellon

ETH zürich

More on D-RaNGe

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu, **"D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput"** *Proceedings of the 25th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Washington, DC, USA, February 2019. [Slides (pptx) (pdf)] [Full Talk Video (21 minutes)]



Using Memory for Security

- Generating True Random Numbers (using DRAM)
  - Kim et al., HPCA 2019

- Evaluating Physically Unclonable Functions (using DRAM)
  - Kim et al., HPCA 2018

- Quickly Destroying In-Memory Data (using DRAM)
  - Orosa et al., arxiv.org 2019

The DRAM Latency PUF:

Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices

Jeremie S. Kim **Minesh Patel**

Hasan Hassan **Onur Mutlu**

SAFARI

ETH Zürich

Carnegie Mellon

Executive Summary

- **Motivation**:
  - We can authenticate a system via **unique signatures** if we can evaluate a **Physical Unclonable Function (PUF)** on it
  - Signatures **(PUF response)** reflect inherent properties of a device
  - DRAM is a promising substrate for PUFs because it is **widely** used
- **Problem**: Current DRAM PUFs are 1) very slow, 2) require a DRAM reboot, or 3) require additional custom hardware
- **Goal**: To develop a novel and effective PUF for **existing** commodity DRAM devices with **low-latency evaluation time** and **low system interference** across **all operating temperatures**
- **DRAM Latency PUF:** Reduce DRAM access latency **below reliable values** and exploit the resulting error patterns as **unique identifiers**
- **Evaluation:**
  1. Experimentally characterize **223 real LPDDR4 DRAM devices**
  2. **DRAM latency PUF** (88.2 ms) achieves a speedup of **102x/860x** at 70°C/55°C over prior DRAM PUF evaluation mechanisms

The DRAM Latency PUF Outline

- Motivation
- Effective PUF Characteristics
- DRAM Latency PUF
  - DRAM Operation
  - Key Idea
- Prior Best DRAM PUF: DRAM Retention PUF
  - DRAM Cell Retention
  - Key Idea
  - Weaknesses
- Methodology
- Results
- Summary

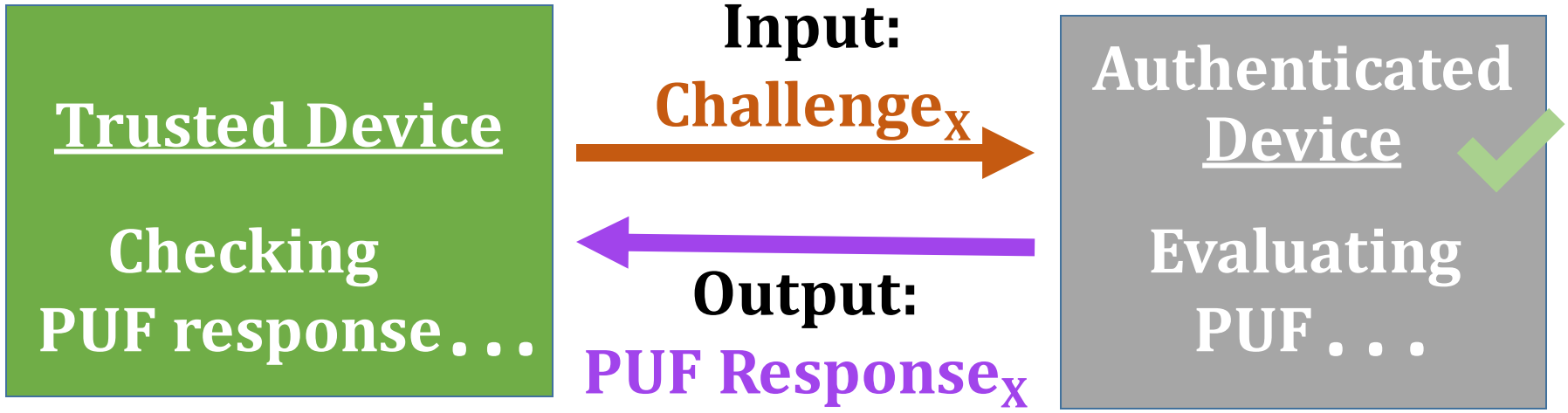
The DRAM Latency PUF Outline

- **Motivation**
- Effective PUF Characteristics
- DRAM Latency PUF
  - DRAM Operation
  - Key Idea
- Prior Best DRAM PUF: DRAM Retention PUF
  - DRAM Cell Retention
  - Key Idea
  - Weaknesses
- Methodology
- Results
- Summary

Motivation

We want a way to ensure that a system's components are not **compromised**

- **Physical Unclonable Function (PUF):** a function we **evaluate** on a device to **generate** a **signature** **unique** to the device
- We refer to the unique signature as a **PUF response**
- Often used in a **Challenge-Response Protocol (CRP)**



Motivation

1. We want a **runtime-accessible** PUF
   - Should be evaluated **quickly** with **minimal** impact on concurrent applications
   - Can protect against **attacks that swap system components with malicious parts**

2. DRAM is a **promising substrate** for evaluating PUFs because it is **ubiquitous** in modern systems
   - Unfortunately, current DRAM PUFs are **slow** and get **exponentially slower** at lower temperatures

The DRAM Latency PUF Outline

- Motivation
- **Effective PUF Characteristics**
- DRAM Latency PUF
  - DRAM Operation
  - Key Idea
- Prior Best DRAM PUF: DRAM Retention PUF
  - DRAM Cell Retention
  - Key Idea
  - Weaknesses
- Methodology
- Results
- Summary

Effective PUF Characteristics

1. Repeatability

Effective PUF Characteristics

1. Repeatability
2. Diffuseness

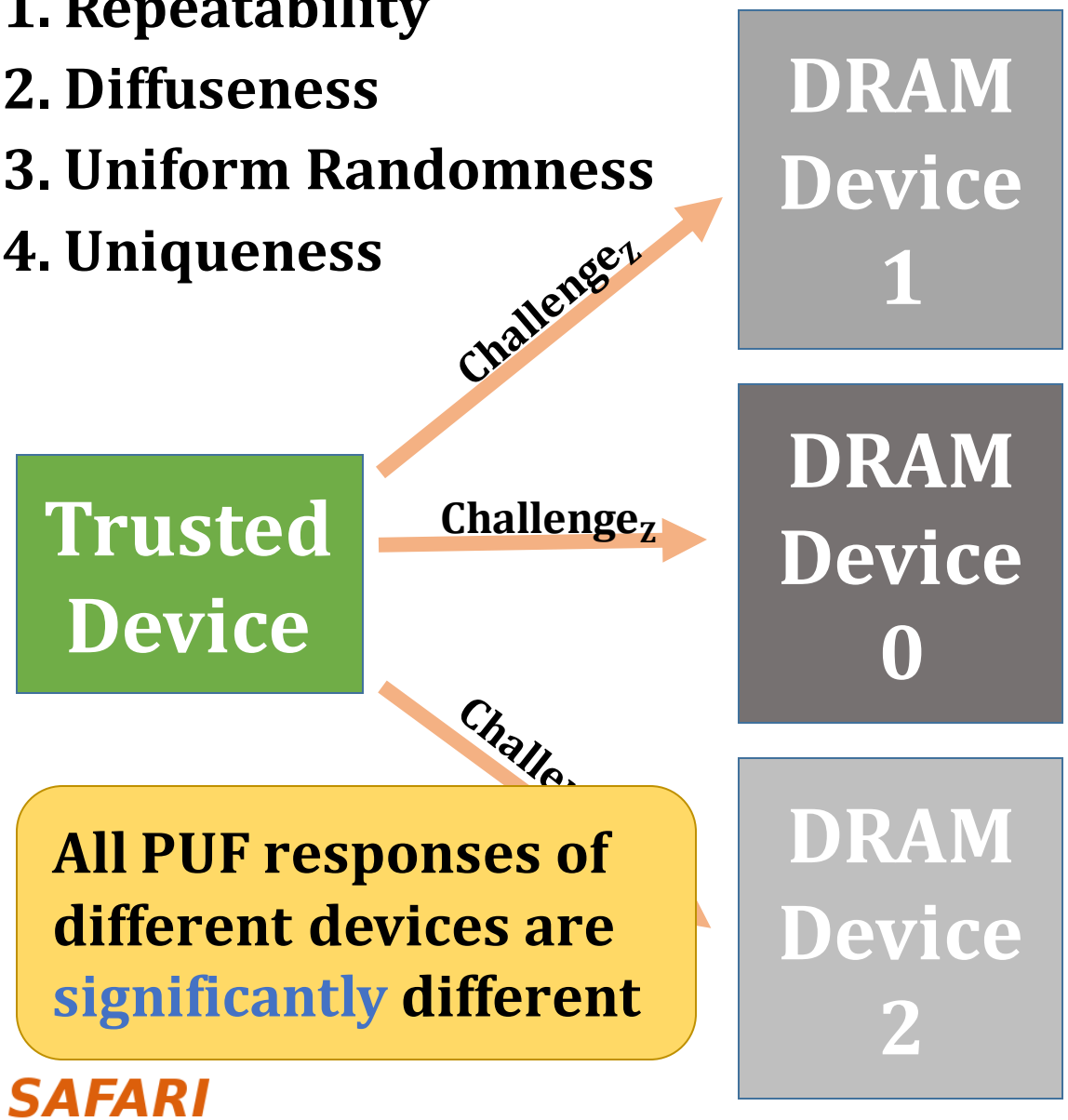
Effective PUF Characteristics

1. Repeatability
2. Diffuseness
3. Uniform Randomness

Cannot use multiple challenge-response pairs to guess another

Effective PUF Characteristics

1. Repeatability
2. Diffuseness
3. Uniform Randomness
4. Uniqueness



Effective PUF Characteristics

1. Repeatability
2. Diffuseness
3. Uniform Randomness
4. Uniqueness
5. Unclonability

Trusted Device

DRAM Device 0

Effective PUF Characteristics

1. Repeatability
2. Diffuseness
3. Uniform Randomness
4
5

More analysis
of the effective PUF characteristics
in the paper

Effective PUF Characteristics

Runtime-accessible PUFs must have

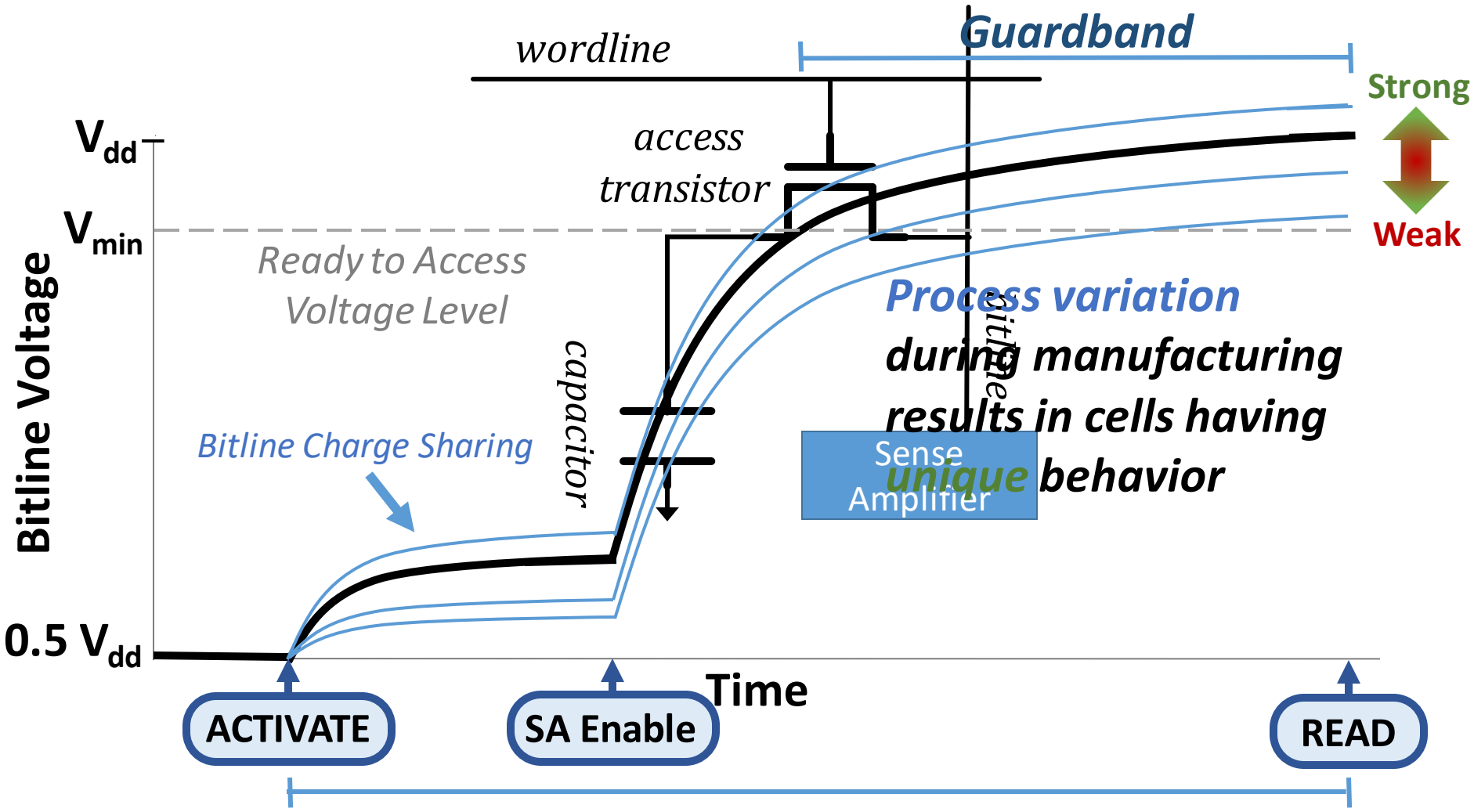
1. **Low Latency**
   - Each device can **quickly** generate a PUF response

2. **Low System Interference**
   - PUF evaluation **minimally affects performance** of concurrently-running applications

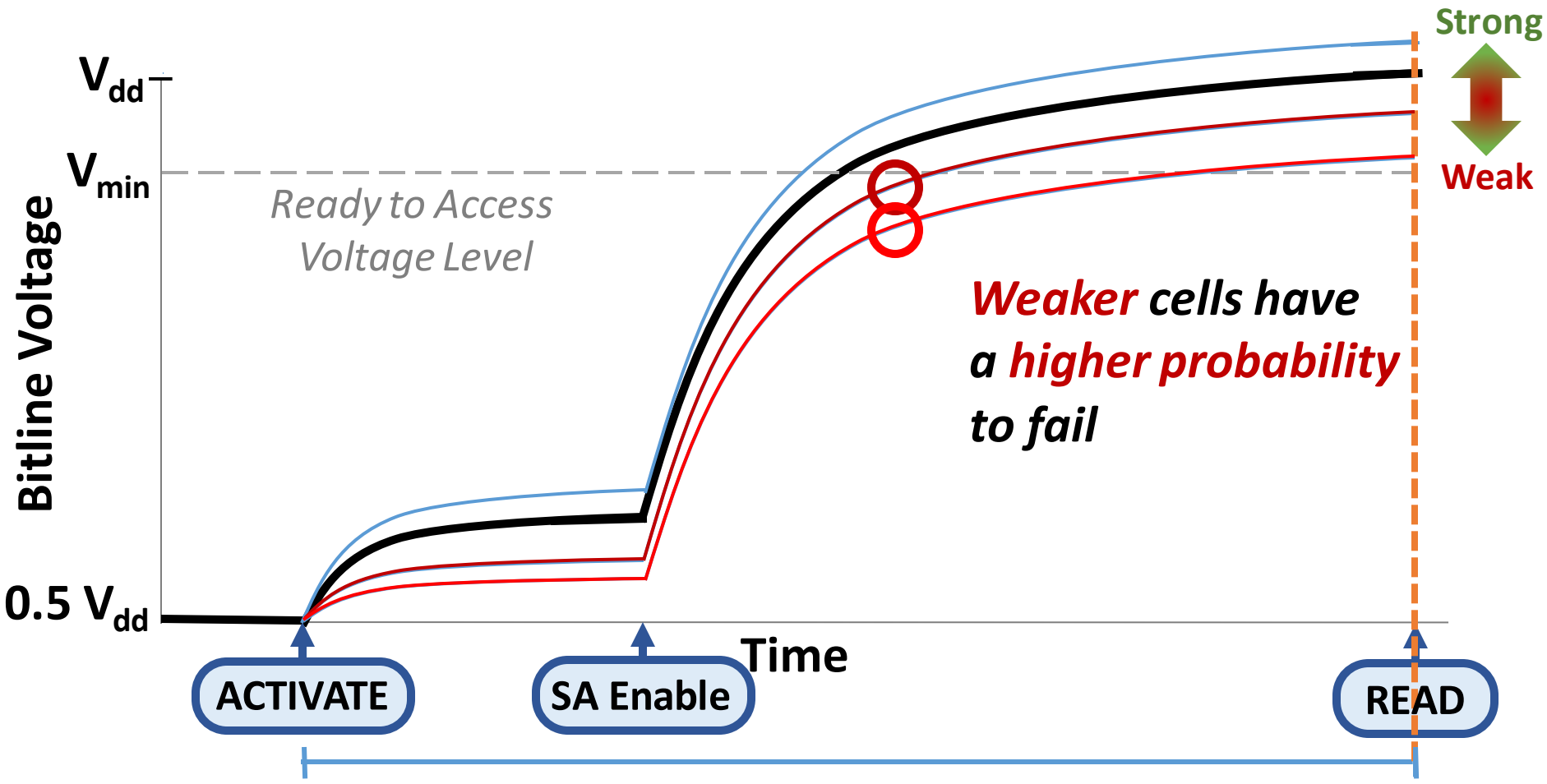
The DRAM Latency PUF Outline

- Motivation
- Effective PUF Characteristics
- **DRAM Latency PUF**
  - **DRAM Operation**
  - Key Idea
- Prior Best DRAM PUF: DRAM Retention PUF
  - DRAM Cell Retention
  - Key Idea
  - Weaknesses
- Methodology
- Results
- Summary

DRAM Accesses and Failures



DRAM Accesses and Failures

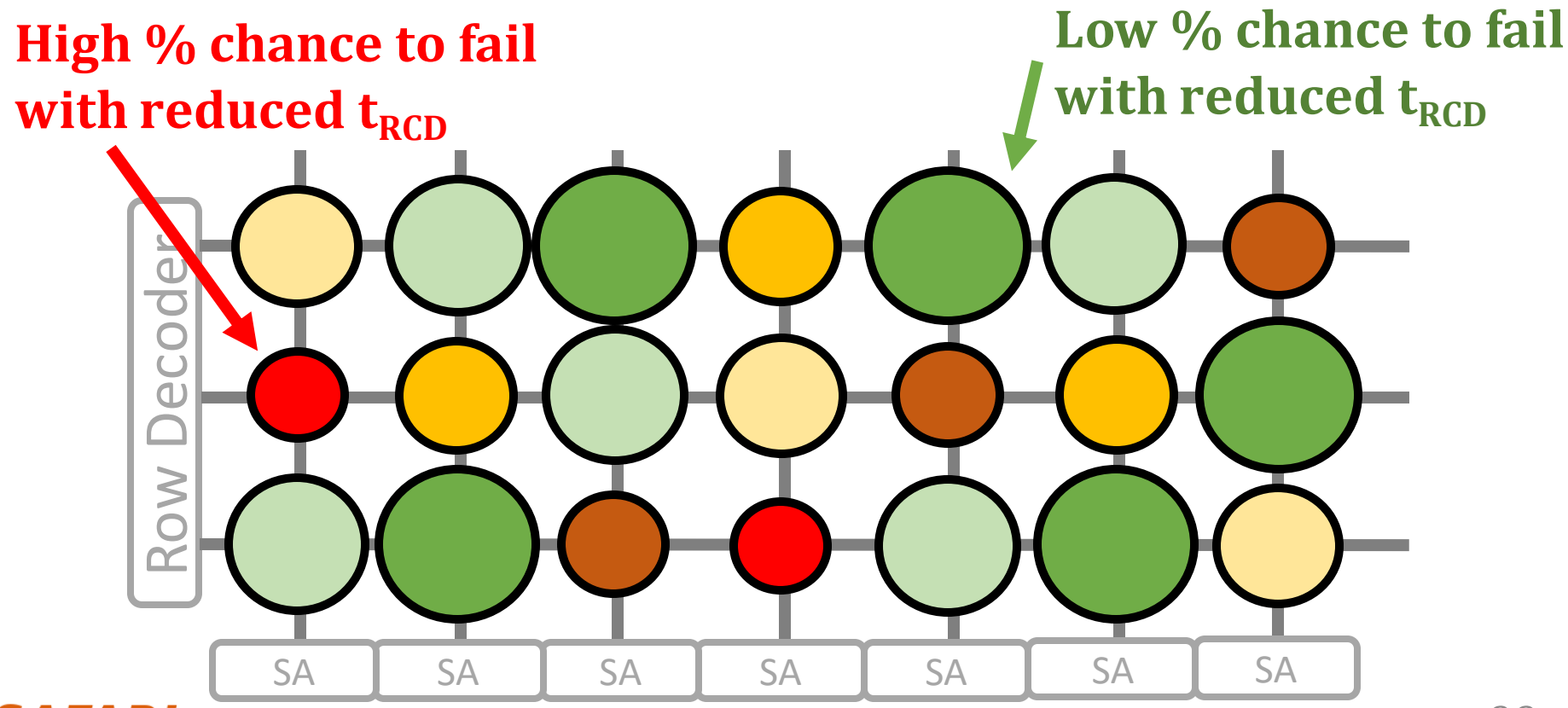


The DRAM Latency PUF Outline

- Motivation
- Effective PUF Characteristics
- **DRAM Latency PUF**
  - DRAM Operation
  - **Key Idea**
- Prior Best DRAM PUF: DRAM Retention PUF
  - DRAM Cell Retention
  - Key Idea
  - Weaknesses
- Methodology
- Results
- Summary

DRAM Latency PUF Key Idea

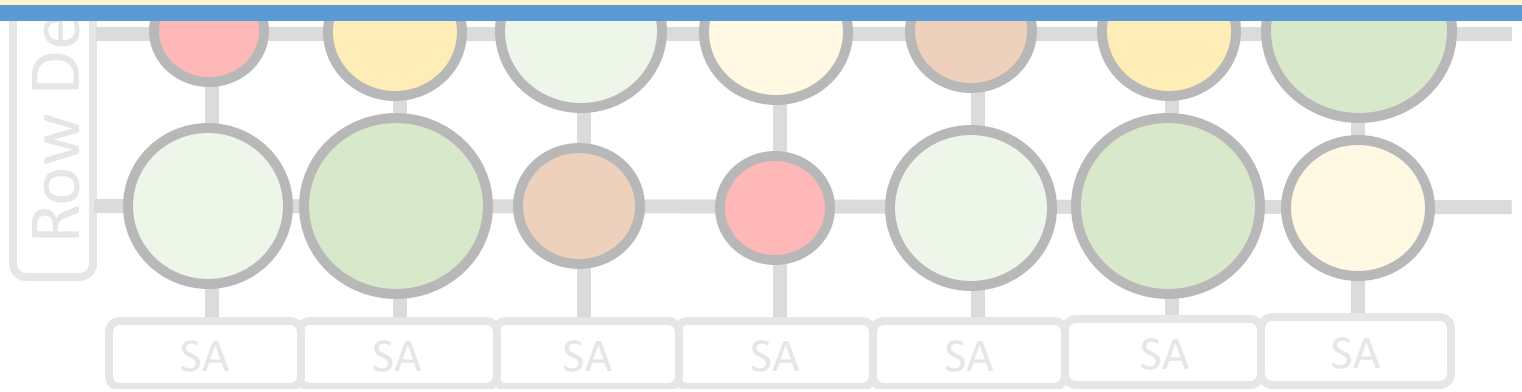
- A cell's latency failure probability is inherently related to **random process variation** from manufacturing
- We can provide **repeatable and unique device signatures** using latency error patterns



DRAM Latency PUF Key Idea

- A cell's latency failure probability is inherently related to **random process variation** from manufacturing
- We can provide **repeatable and unique device**

The key idea is to compose a PUF response using the DRAM cells that fail with high probability

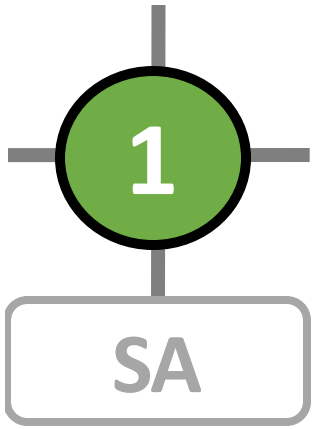


Evaluating a DRAM Latency PUF

Determine whether a **single cell's location** should be included in a DRAM latency PUF response

- **Include** if the cell **fails** with a probability greater than a **chosen threshold** when accessed with a reduced $\mathbf{t_{RCD}}$

Chosen Threshold: 50%



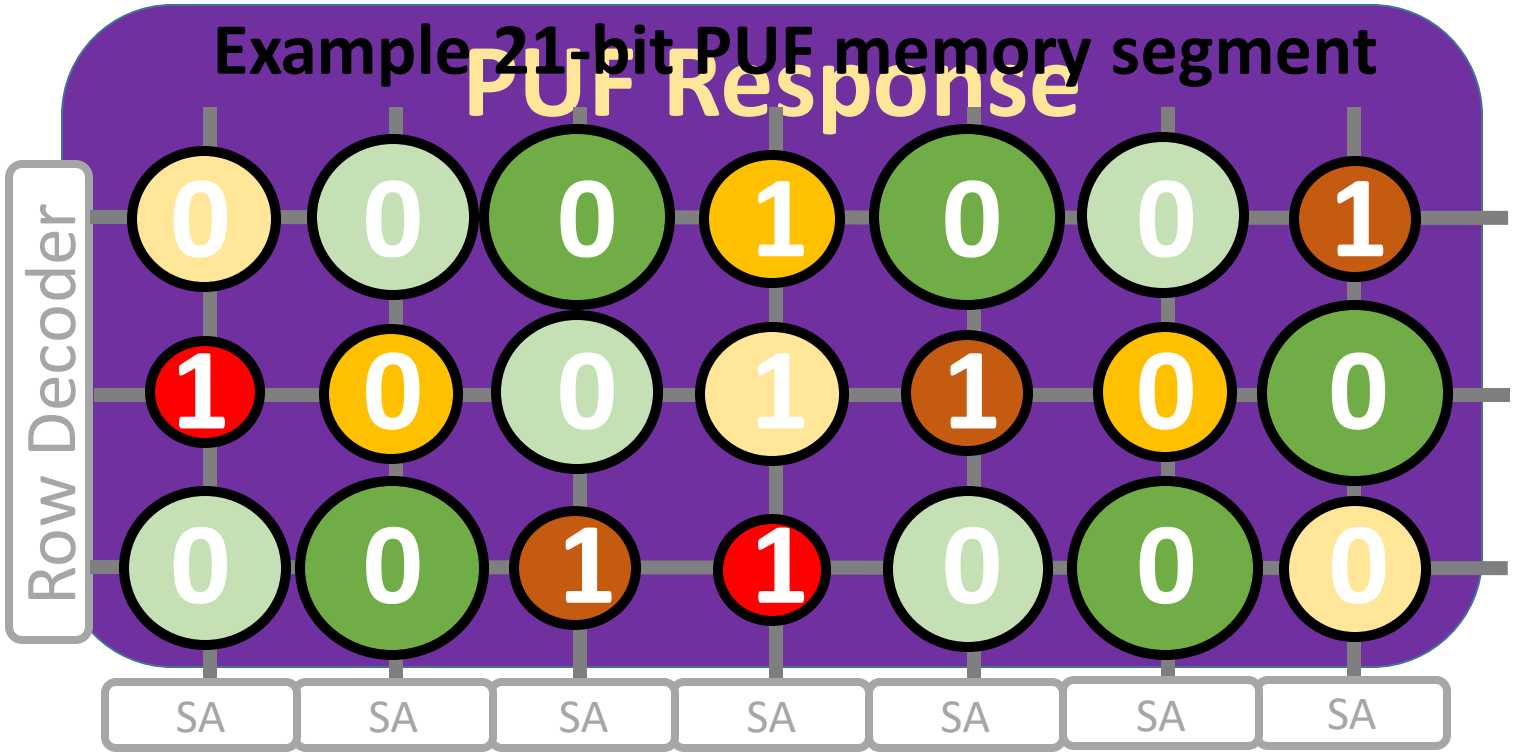
This Cell's Failure Rate: 60%

Failure rate is greater than the chosen threshold, so the cell's location should be included

✗ ✗ ✗✗ ✗✗

Evaluating a DRAM Latency PUF

- We induce latency failures **100 times** and use a **threshold of 10%** (i.e., use cells that fail > 10 times)
- We do this for every cell in a continuous **8KiB** memory region, that we refer to as a **PUF memory segment**



Evaluating a DRAM Latency PUF

- We induce latency failures **100 times** and use a **threshold of 10%** (i.e., use cells that fail > 10 times)
- We do this for every cell in a continuous 8KiB memory

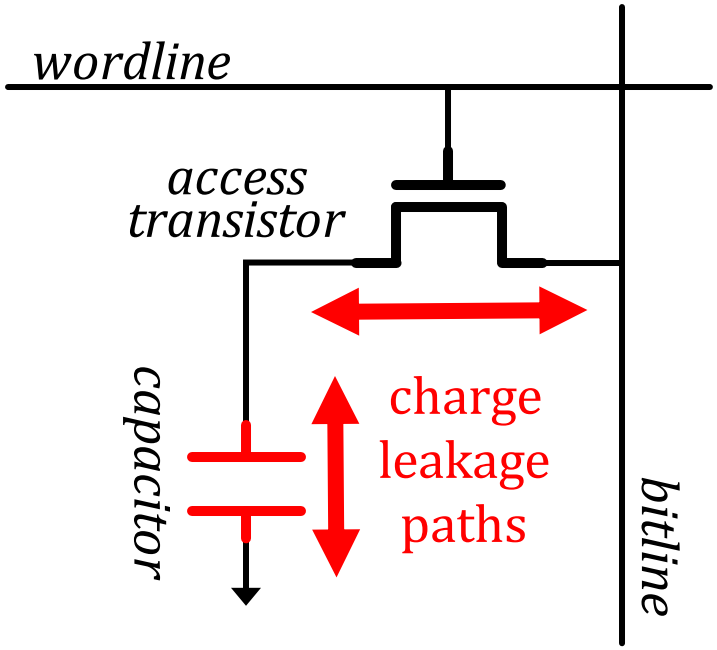
We can evaluate the DRAM latency PUF in only 88.2ms on average regardless of temperature!

The DRAM Latency PUF Outline

- Motivation
- Effective PUF Characteristics
- DRAM Latency PUF
  - DRAM Operation
  - Key Idea
- **Prior Best DRAM PUF: DRAM Retention PUF**
  - **DRAM Cell Retention**
  - Key Idea
  - Weaknesses
- Methodology
- Results
- Summary

DRAM Cell Leakage

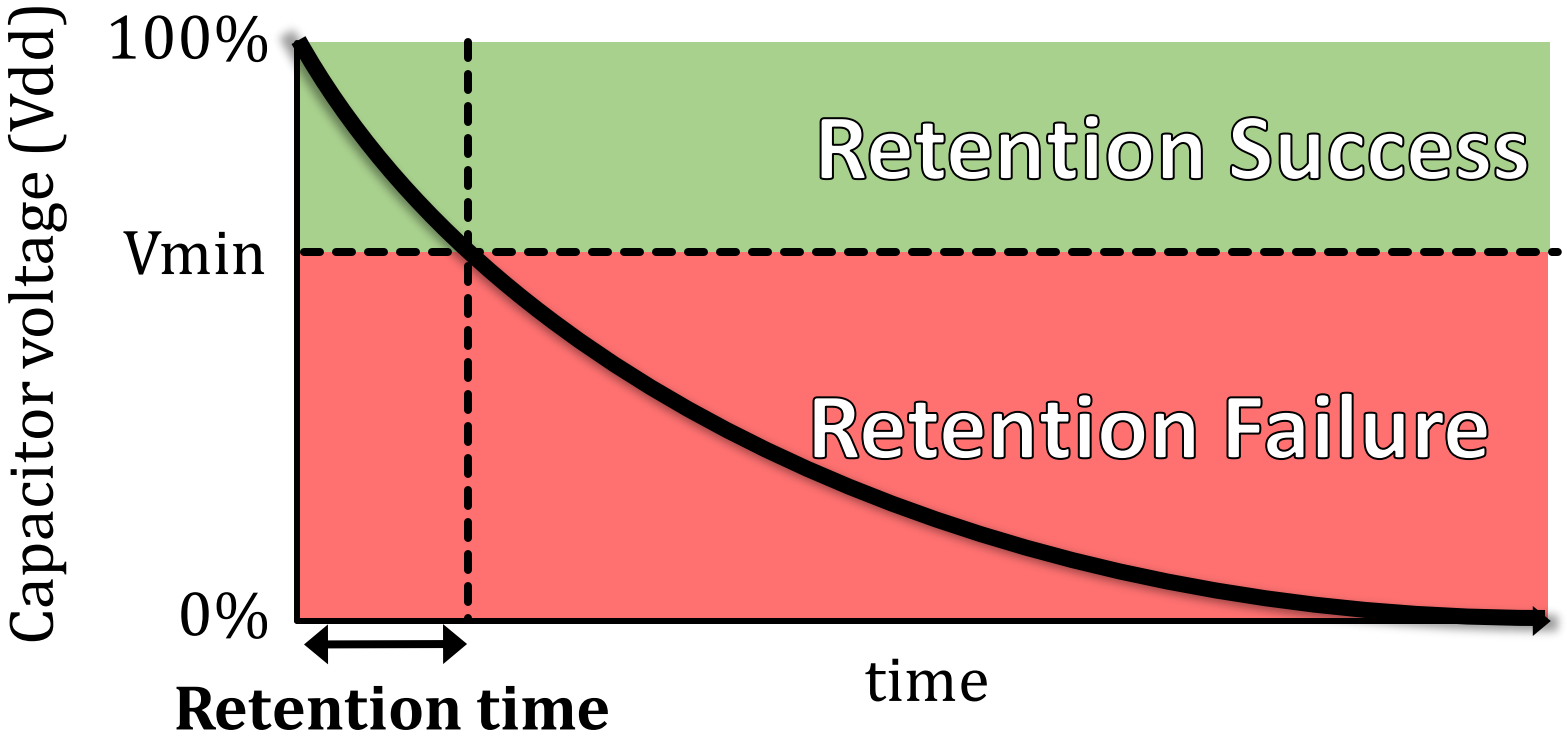
DRAM encodes information in leaky capacitors



Stored data is corrupted if too much charge leaks (i.e., the capacitor voltage degrades too much)

[Patel et al., REAPER, ISCA'17]

DRAM Cell Retention

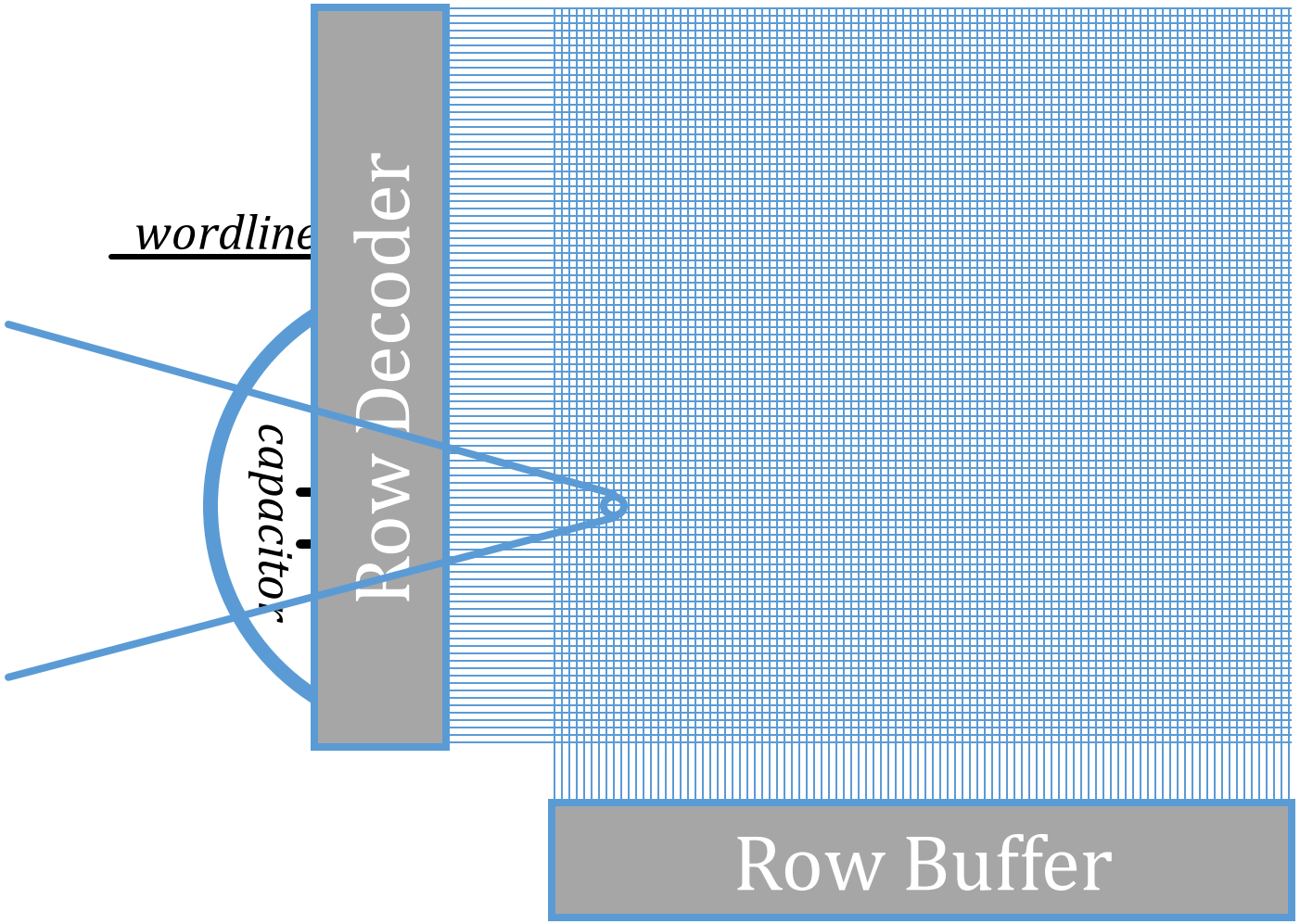


Retention failure – when leakage corrupts stored data

Retention time – how long a cell holds its value

[Patel et al., REAPER, ISCA'17]

Each Cell has a Different Retention Time



8GB DRAM = 6.4e10 cells

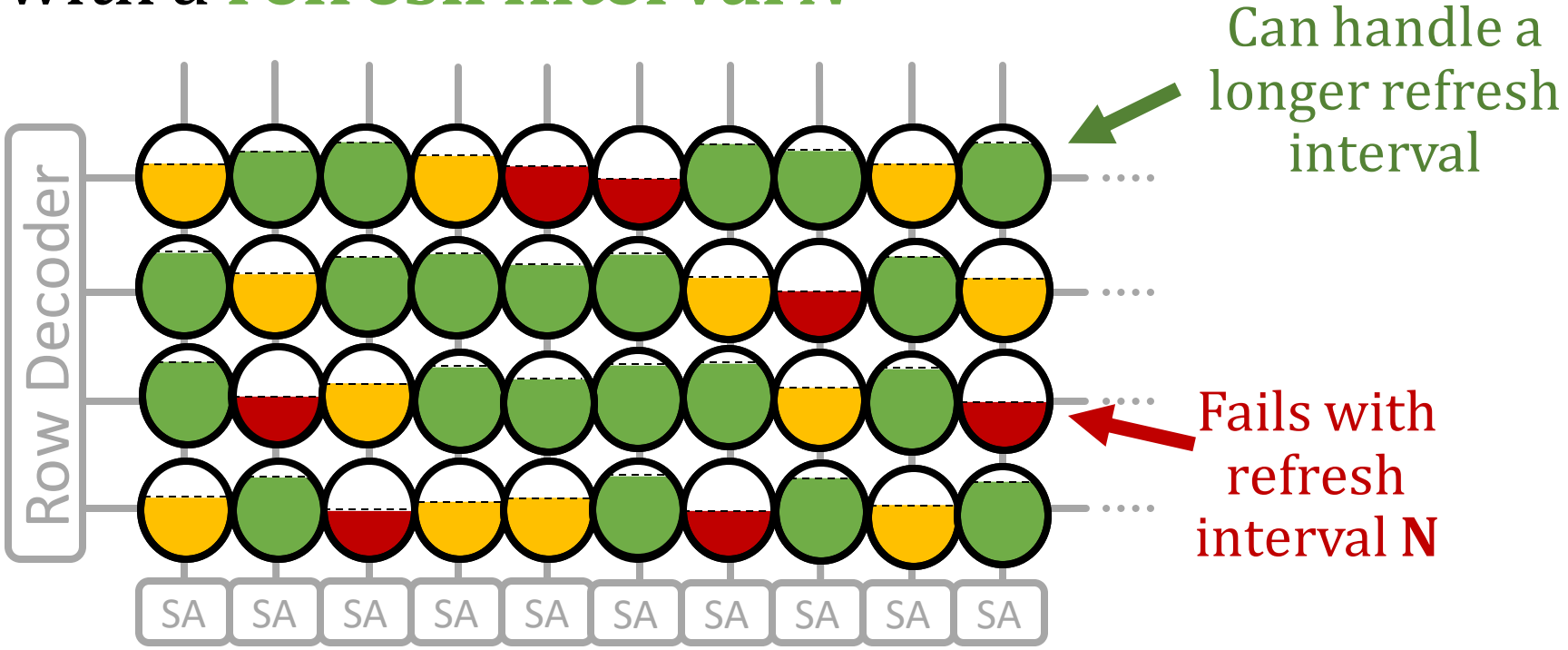
[Patel et al., REAPER, ISCA'17]

The DRAM Latency PUF Outline

- Motivation
- Effective PUF Characteristics
- DRAM Latency PUF
  - DRAM Operation
  - Key Idea
- **Prior Best DRAM PUF: DRAM Retention PUF**
  - DRAM Cell Retention
  - **Key Idea**
  - Weaknesses
- Methodology
- Results
- Summary

Evaluating a DRAM Retention PUF

Generate a **PUF response** with locations of cells in a **PUF memory segment** that **fail** with a **refresh interval *N***



The pattern of retention failures across a segment of DRAM is **unique** to the device

Evaluating a DRAM Retention PUF

Generate a **PUF response** with locations of cells in a **PUF memory segment** that **fail** with a **refresh interval *N***

Can handle a

We use the best methods from prior work and optimize the retention PUF for our devices

DRAM is unique to the device

The DRAM Latency PUF Outline

- Motivation
- Effective PUF Characteristics
- DRAM Latency PUF
  - DRAM Operation
  - Key Idea
- **Prior Best DRAM PUF: DRAM Retention PUF**
  - DRAM Cell Retention
  - Key Idea
  - **Weaknesses**
- Methodology
- Results
- Summary

DRAM Retention PUF Weaknesses

DRAM Retention PUF evaluation time is **very long** and leads to **high system interference**

Long evaluation time:

1. Most DRAM cells are strong → need to wait for long time to drain charge from capacitors
2. Especially at low temperatures

High system interference:

1. DRAM refresh can only be disabled at a **channel granularity** **(512MB – 2GB)**
2. Must issue **manual refreshes** to maintain data correctness in the rest of the channel **during entire evaluation time**
3. Manually refreshing DRAM consumes **significant** bandwidth on the DRAM bus

DRAM Retention PUF Weaknesses

Long evaluation time could be ameliorated in 2 ways:

1. **Increase temperature –** higher rate of charge leakage

   → **Observe failures faster**

Unfortunately:

1. Difficult to control DRAM temperature in the field
2. Operating at high temperatures is undesirable

2. **Increase PUF memory segment size –** more cells with low retention time in PUF memory segment

   → **Observe more failures faster**

Unfortunately:

- Large PUF memory segment

  → **high DRAM capacity overhead**

The DRAM Latency PUF Outline

- Motivation
- Effective PUF Characteristics
- DRAM Latency PUF
  - DRAM Operation
  - Key Idea
- Prior Best DRAM PUF: DRAM Retention PUF
  - DRAM Cell Retention
  - Key Idea
  - Weaknesses
- **Methodology**
- Results
- Summary

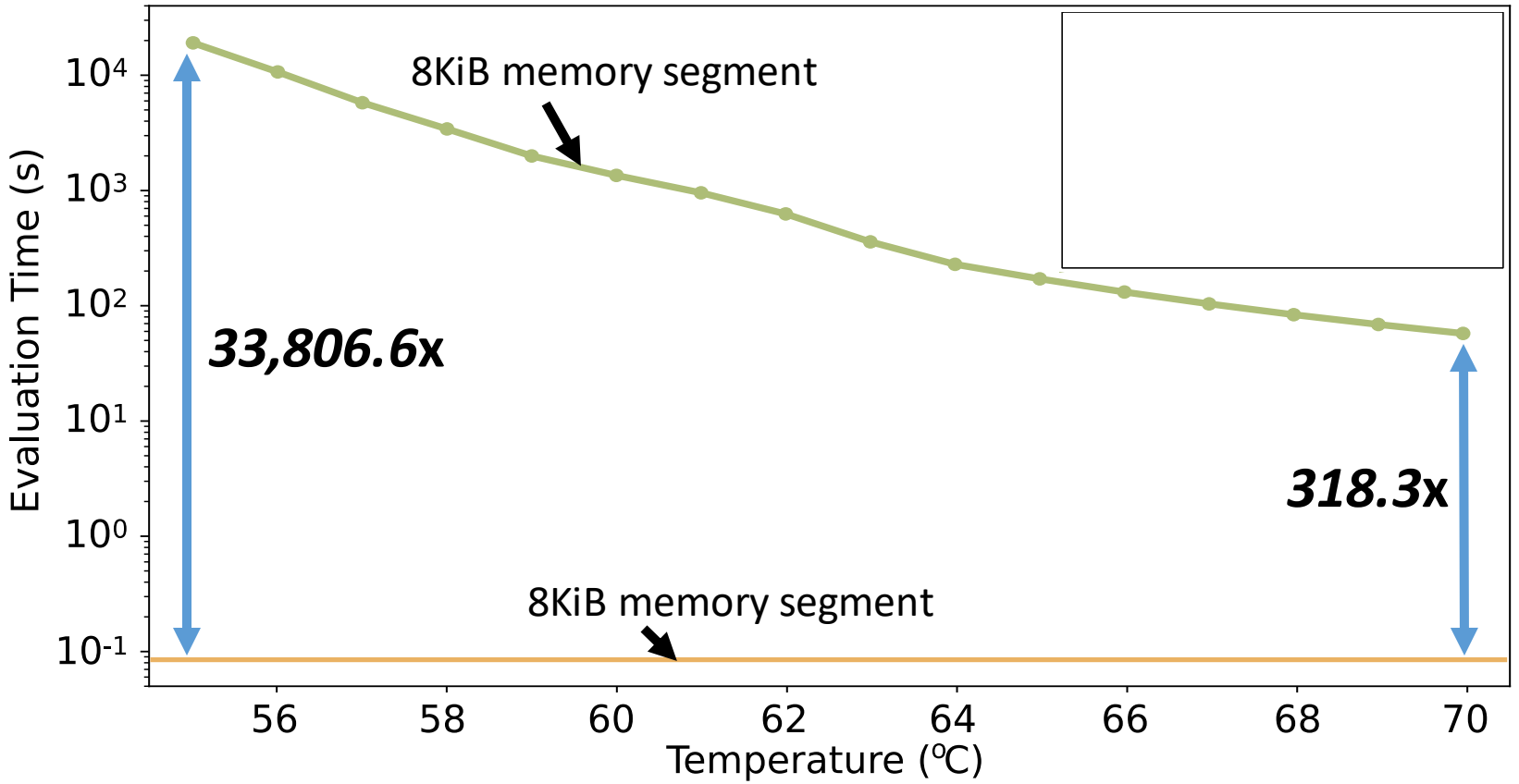
Methodology

- **223 2y-nm LPDDR4 DRAM devices**
  - **2GB** device size
  - From **3 major DRAM manufacturers**
- **Thermally controlled testing chamber**
  - Ambient temperature range: **{40°C – 55°C} ± 0.25°C**
  - DRAM temperature is held at 15°C above ambient
- **Precise control over DRAM commands and timing parameters**
  - Test retention time effects by **disabling refresh**
  - Test reduced latency effects by **reducing t_{RCD} parameter**

The DRAM Latency PUF Outline

- Motivation
- Effective PUF Characteristics
- DRAM Latency PUF
  - DRAM Operation
  - Key Idea
- Prior Best DRAM PUF: DRAM Retention PUF
  - DRAM Cell Retention
  - Key Idea
  - Weaknesses
- Methodology
- **Results**
- Summary

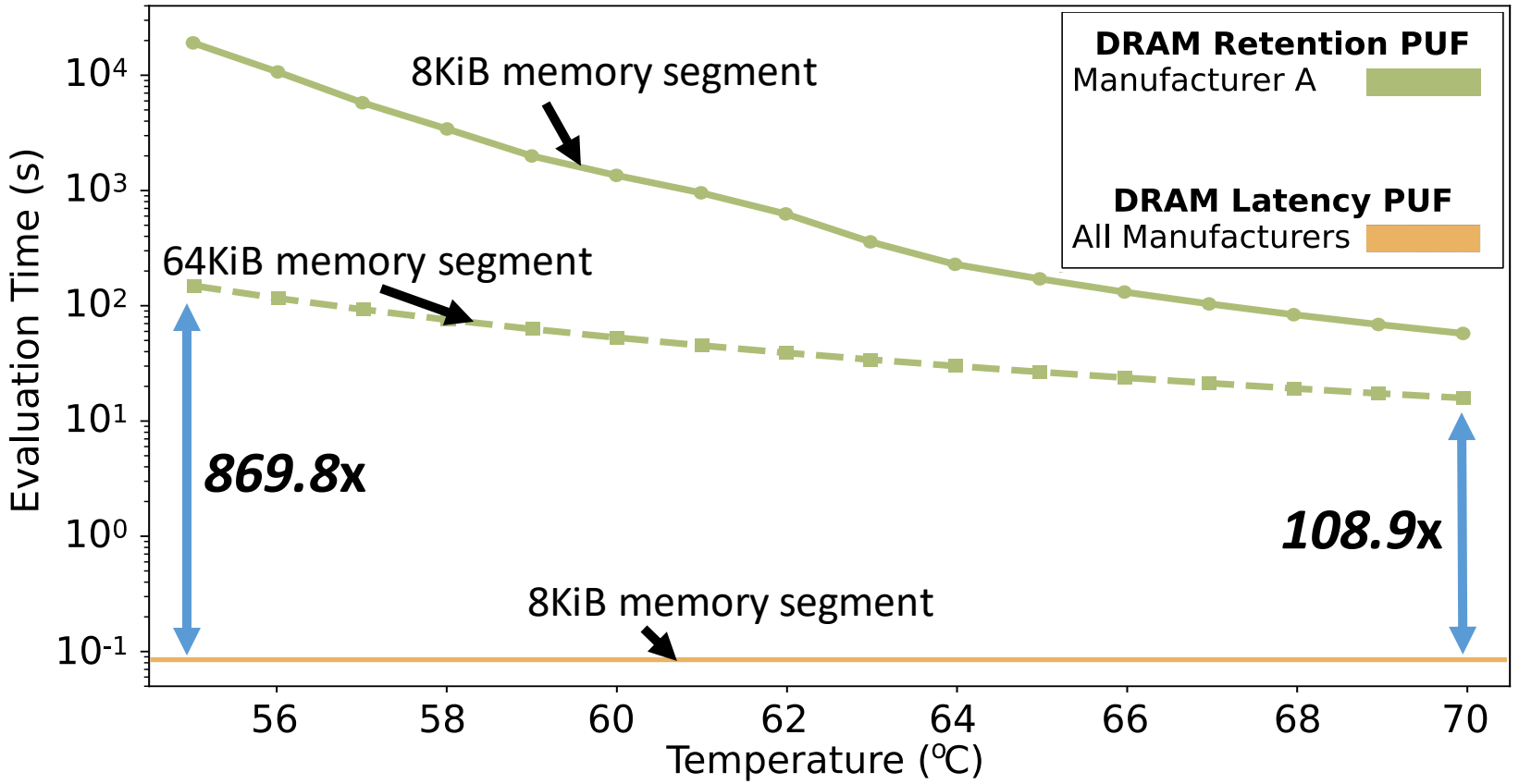
Results – PUF Evaluation Latency



DRAM latency PUF is

1. **Fast and constant latency (88.2ms)**

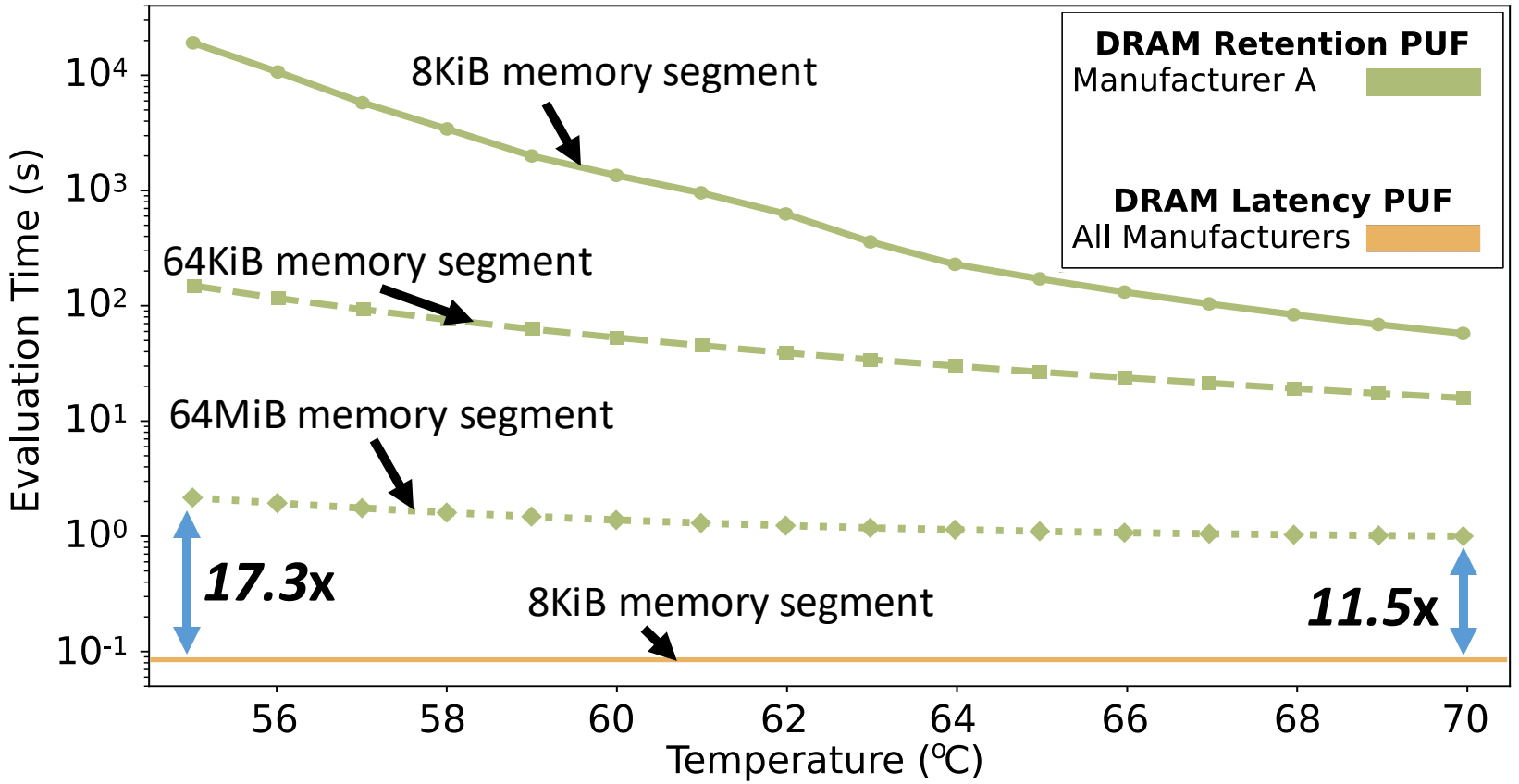
Results – PUF Evaluation Latency



DRAM latency PUF is

1. **Fast and constant latency (88.2ms)**

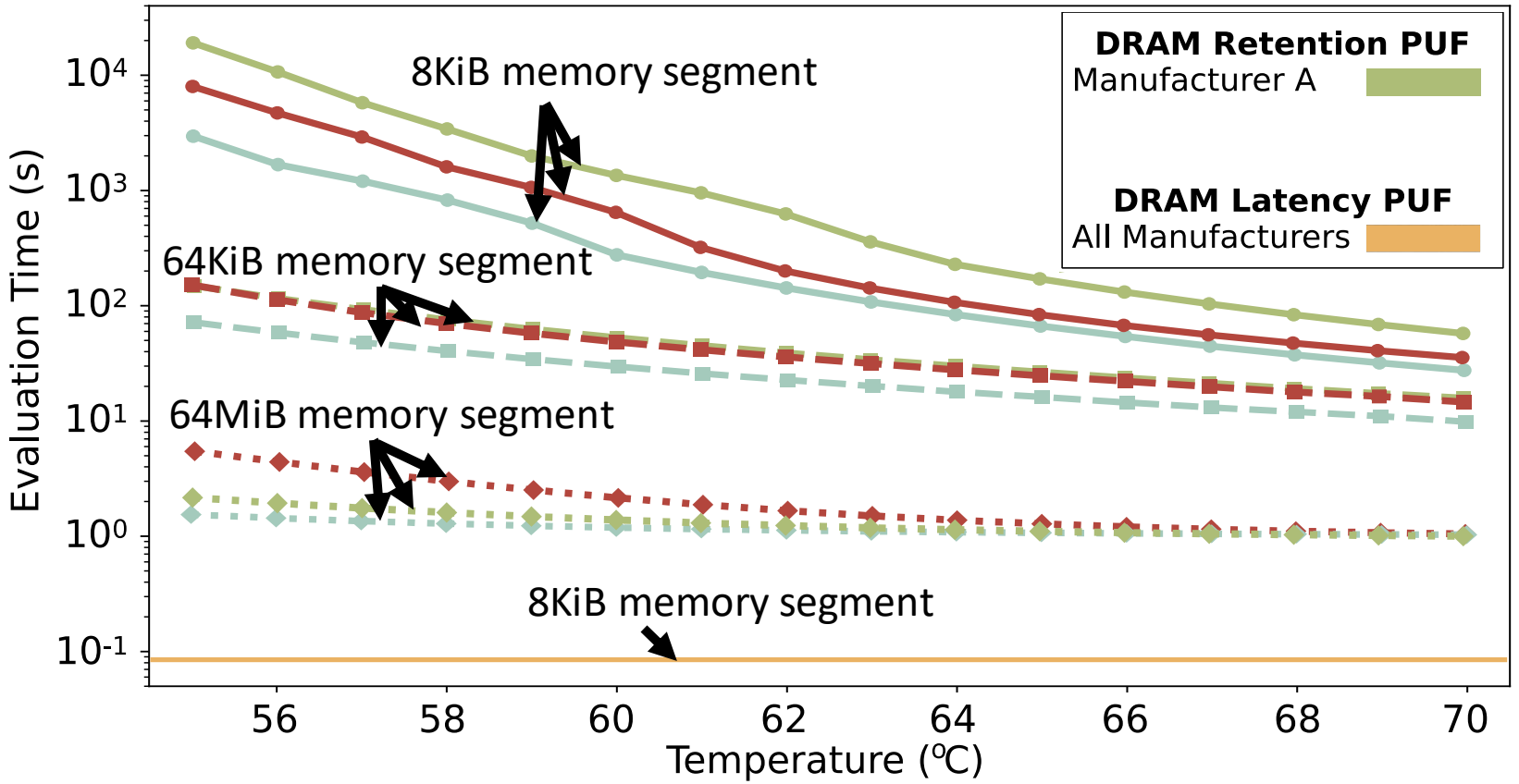
Results – PUF Evaluation Latency



DRAM latency PUF is

1. **Fast and constant latency (88.2ms)**

Results – PUF Evaluation Latency



DRAM latency PUF is

1. **Fast and constant latency (88.2ms)**
2. **On average, 102x/860x faster than the previous DRAM PUF with the same DRAM capacity overhead (64KiB)**

Results – System Interference

During PUF evaluation on commodity devices:

- **The DRAM Retention PUF**
  - Disables refresh at channel granularity **(~512MB – 2GB)**
    - **Issue manual refresh operations** to rows in channel but not in PUF memory segment to prevent data corruption
  - Has **long evaluation time** at low temperatures

- **The DRAM Latency PUF**
  - Does not require disabling refresh
  - Has short evaluation time **at any operating temperature**

Other Results in the Paper

- **How the DRAM latency PUF meets the basic requirements for an effective PUF**
- **A detailed analysis on:**
  - Devices of **the three major DRAM manufacturers**
  - The **evaluation time** of a PUF
- **Further discussion on:**
  - **Optimizing** retention PUFs
  - **System interference** of DRAM retention and latency PUFs
  - Algorithm to **quickly and reliably** evaluate DRAM latency PUF
  - **Design considerations** for a DRAM latency PUF
  - The DRAM Latency PUF overhead analysis

The DRAM Latency PUF Outline

- Motivation
- Effective PUF Characteristics
- DRAM Latency PUF
  - DRAM Operation
  - Key Idea
- Prior Best DRAM PUF: DRAM Retention PUF
  - DRAM Cell Retention
  - Key Idea
  - Weaknesses
- Methodology
- Results
- **Summary**

Executive Summary

- **Motivation**:
  - We can authenticate a system via **unique signatures** if we can evaluate a **Physical Unclonable Function (PUF)** on it
  - Signatures **(PUF response)** reflect inherent properties of a device
  - DRAM is a promising substrate for PUFs because it is **widely** used
- **Problem**: Current DRAM PUFs are 1) very slow, 2) require a DRAM reboot, or 3) require additional custom hardware
- **Goal**: To develop a novel and effective PUF for **existing** commodity DRAM devices with **low-latency evaluation time** and **low system interference** across **all operating temperatures**
- **DRAM Latency PUF:** Reduce DRAM access latency **below reliable values** and exploit the resulting error patterns as **unique identifiers**
- **Evaluation:**
  1. Experimentally characterize **223 real LPDDR4 DRAM devices**
  2. **DRAM latency PUF** (88.2 ms) achieves a speedup of **102x/860x** at 70°C/55°C over prior DRAM PUF evaluation mechanisms

The DRAM Latency PUF:

Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices

Jeremie S. Kim **Minesh Patel**

Hasan Hassan **Onur Mutlu**

SAFARI

ETH Zürich

Carnegie Mellon

More on the DRAM Latency PUF

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu,
  "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern DRAM Devices"
  *Proceedings of the 24th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Vienna, Austria, February 2018.
  [Lightning Talk Video]
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]

The DRAM Latency PUF:
Quickly Evaluating Physical Unclonable Functions
by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices

Jeremie S. Kim[†§] Minesh Patel[§] Hasan Hassan[§] Onur Mutlu[§†]
[†]Carnegie Mellon University [§]ETH Zürich

Using Memory for Security

- Generating True Random Numbers (using DRAM)
  - Kim et al., HPCA 2019

- Evaluating Physically Unclonable Functions (using DRAM)
  - Kim et al., HPCA 2018

- Quickly Destroying In-Memory Data (using DRAM)
  - Orosa et al., arxiv.org 2019

For Another Time …

- Dataplant: In-DRAM Security Mechanisms for Low-Cost Devices
  - https://arxiv.org/pdf/1902.07344.pdf

Dataplant: In-DRAM Security Mechanisms for Low-Cost Devices

Lois Orosa[1] Yaohua Wang[1,2] Ivan Puddu[1] Mohammad Sadrosadati[1,3]
Kaveh Razavi[1,4] Juan Gómez-Luna[1] Hasan Hassan[1] Nika Mansouri-Ghiasi[1]
Arash Tavakkol[1] Minesh Patel[1] Jeremie Kim[1,5] Vivek Seshadri[6]
Uksong Kang[7] Saugata Ghose[5] Rodolfo Azevedo[8] Onur Mutlu[1,5]

[1]*ETH Zürich* [2]*National University of Defense Technology* [3]*Sharif University of Technology*
[4]*Vrije Universiteit Amsterdam* [5]*Carnegie Mellon University* [6]*Microsoft* [7]*SK Hynix* [8]*UNICAMP*

Conclusion

- Memory devices have inherent capability to support key security primitives
  - True Random Number Generation
  - Physically Unclonable Functions
  - Fast Destruction/Randomization of Data
  - …
- It is time for us to treat memory as an intelligent device
  - that does more than simply storing and supplying data…
  - **Producing security primitives is one example**
- We can reinvent computing
  - with a **memory-centric design perspective**

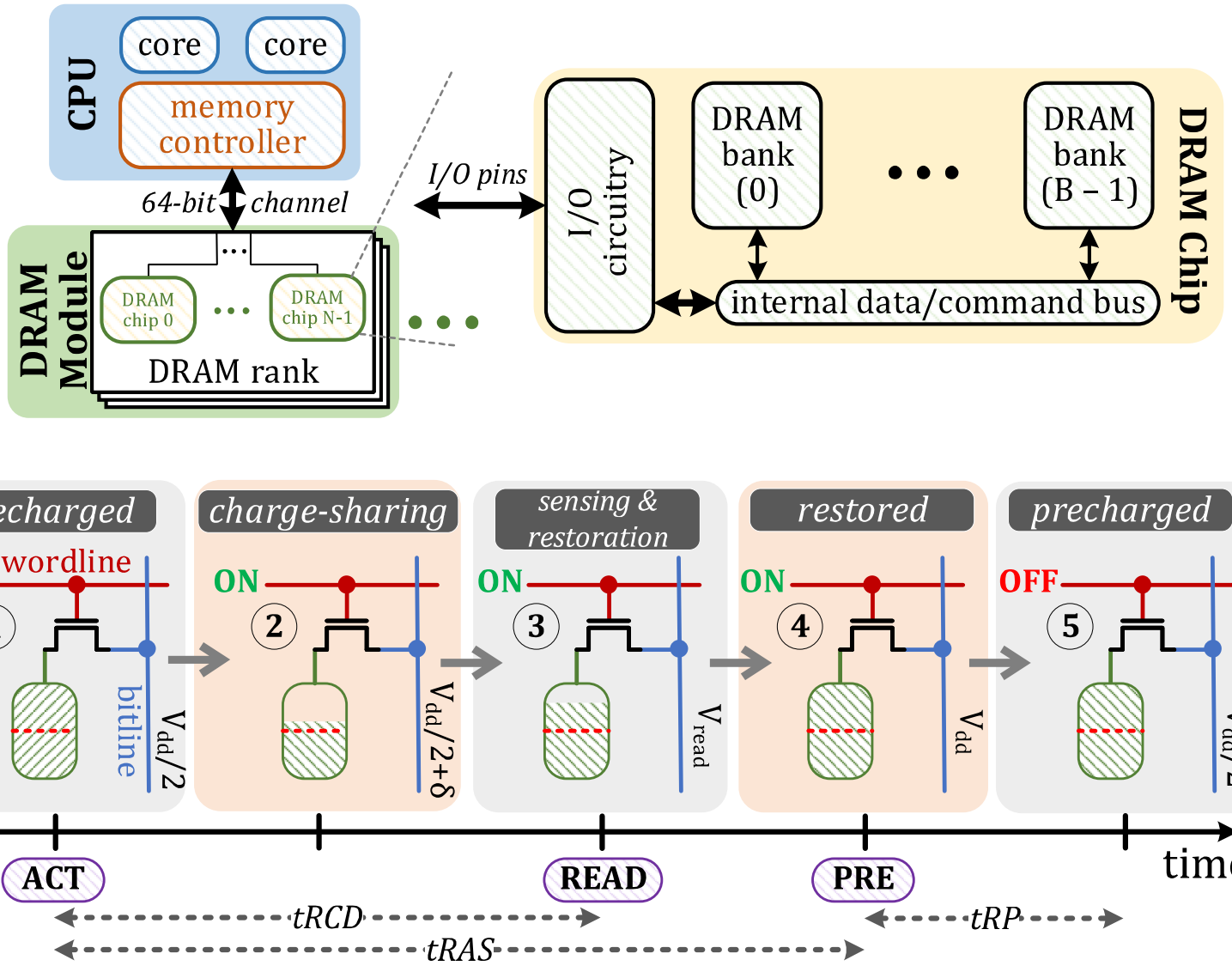
Using Commodity Memory Devices to Support Fundamental Security Primitives

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
25 March 2019
Bogazici University

SAFARI ETH zürich Carnegie Mellon

DRaNGe Backup Slides

DRAM Organization + Operation



DRAM Activation Failure Testing

Algorithm 1: DRAM Activation Failure Testing

```
1  DRAM_ACT_failure_testing(data_pattern, DRAM_region):
2      write data_pattern (e.g., solid 1s) into all cells in DRAM_region
3      set low t_RCD for ranks containing DRAM_region
4      foreach col in DRAM_region:
5          foreach row in DRAM_region:
6              activate(row)      // fully refresh cells
7              precharge(row)     // ensure next access activates the row
8              activate(row)
9              read(col)          // induce activation failure on col
10             precharge(row)
11             record activation failures to storage
12     set default t_RCD for DRAM ranks containing DRAM_region
```

Activation Failure Spatial Distribution

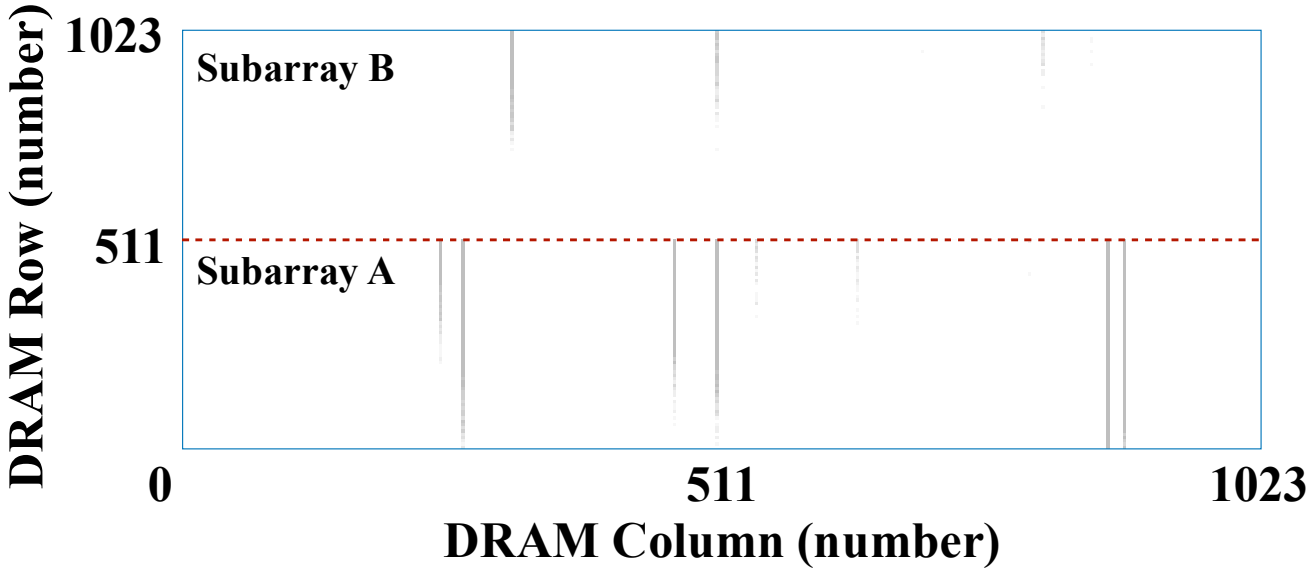


Figure 4: Activation failure bitmap in 1024 ◊ 1024 cell array.

Activation Failure Temperature Dependence

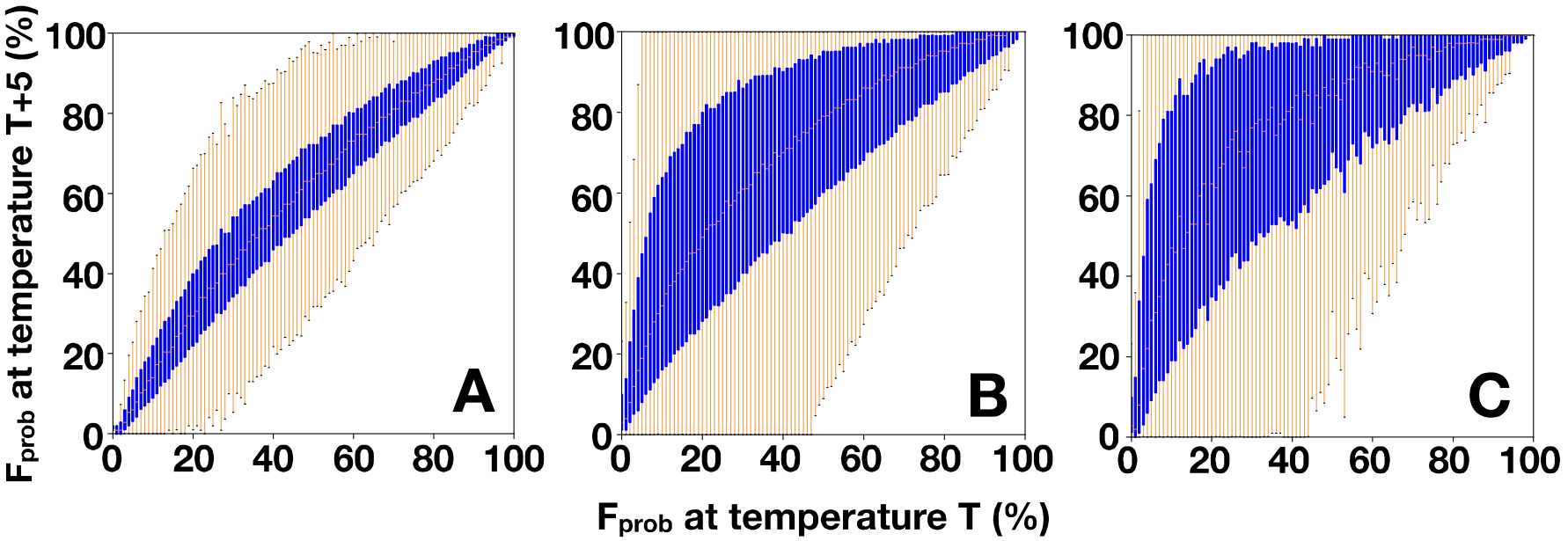


Figure 6: Effect of temperature variation on failure probability

Full D-RaNGe Algorithm

Algorithm 2: D-RaNGe: A DRAM-based TRNG

```
1  D-RaNGe(num_bits):   // num_bits: number of random bits requested
2     DP: a known data pattern that results in high entropy
3     select 2 DRAM words with RNG cells in distinct rows in each bank
4     write DP to chosen DRAM words and their neighboring cells
5     get exclusive access to rows of chosen DRAM words and nearby cells
6     set low t_RCD for DRAM ranks containing chosen DRAM words
7     for each bank:
8        read data in DW_1   // induce activation failure
9        write the read value of DW_1's RNG cells to bitstream
10       write original data value back into DW_1
11       memory barrier    // ensure completion of write to DW_1
12       read data in DW_2  // induce activation failure
13       write the read value of DW_2's RNG cells to bitstream
14       write original data value back into DW_2
15       memory barrier    // ensure completion of write to DW_2
16       if bitstream_size ≥ num_bits:
17          break
18    set default t_RCD for DRAM ranks of the chosen DRAM words
19    release exclusive access to rows of chosen words and nearby cells
```

Summary Comparison Table

Proposal	Year	Entropy Source	True Random	Streaming Capable	64-bit TRNG Latency	Energy Consumption	Peak Throughput
Pyo+ [116]	2009	Command Schedule	✗	✓	$18\mu s$	N/A	$3.40Mb/s$
Keller+ [65]	2014	Data Retention	✓	✓	$40s$	$6.8m\mathcal{J}/bit$	$0.05Mb/s$
Tehranipoor+ [144]	2016	Startup Values	✓	✗	$> 60ns$ (optimistic)	$> 245.9p\mathcal{J}/bit$ (optimistic)	N/A
Sutar+ [141]	2018	Data Retention	✓	✓	$40s$	$6.8m\mathcal{J}/bit$	$0.05Mb/s$
D-RaNGe	2018	Activation Failures	✓	✓	$100ns < x < 960ns$	$4.4n\mathcal{J}/bit$	$717.4Mb/s$

Table 2: Comparison to previous DRAM-based TRNG proposals.

DRAM Data Pattern Dependence

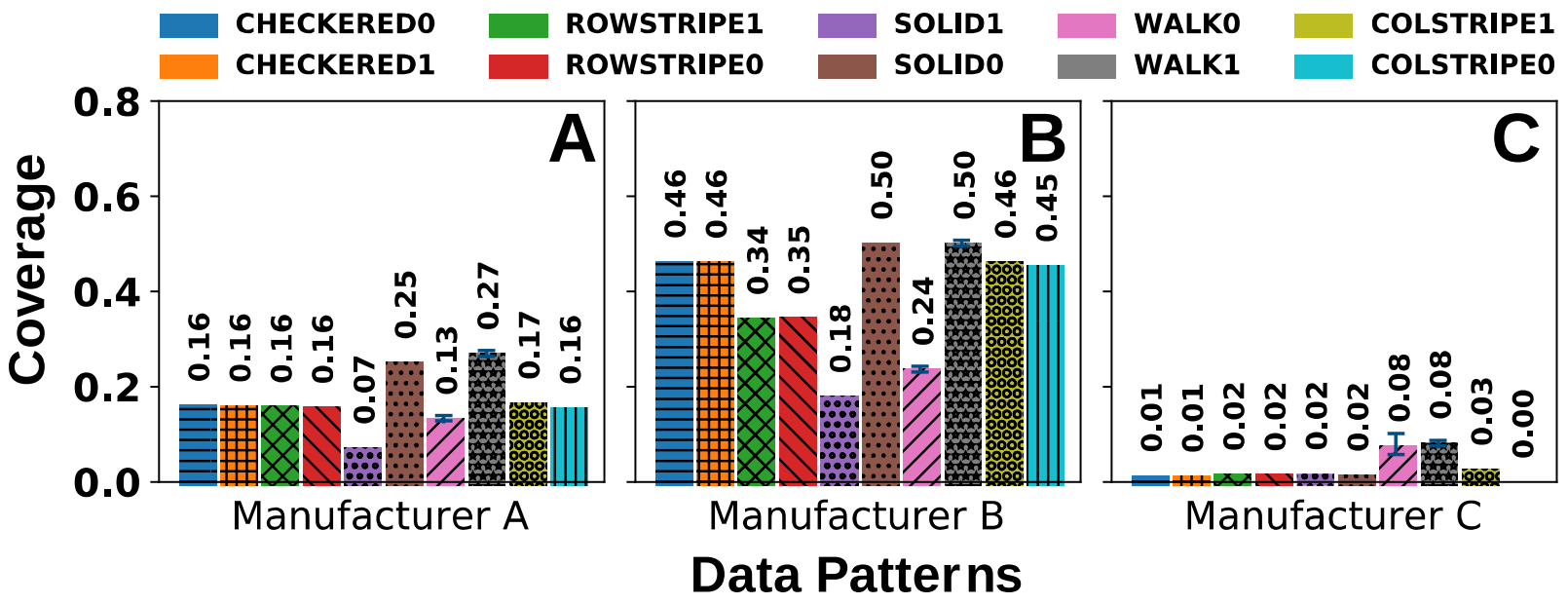
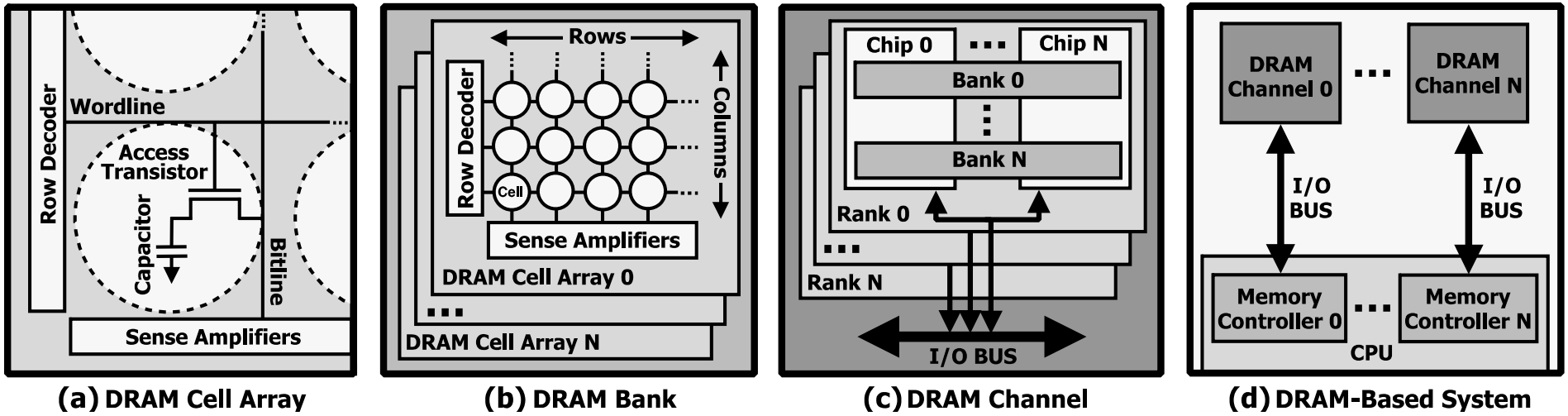


Figure 5: Data pattern dependence of DRAM cells prone to activation failure over 100 iterations

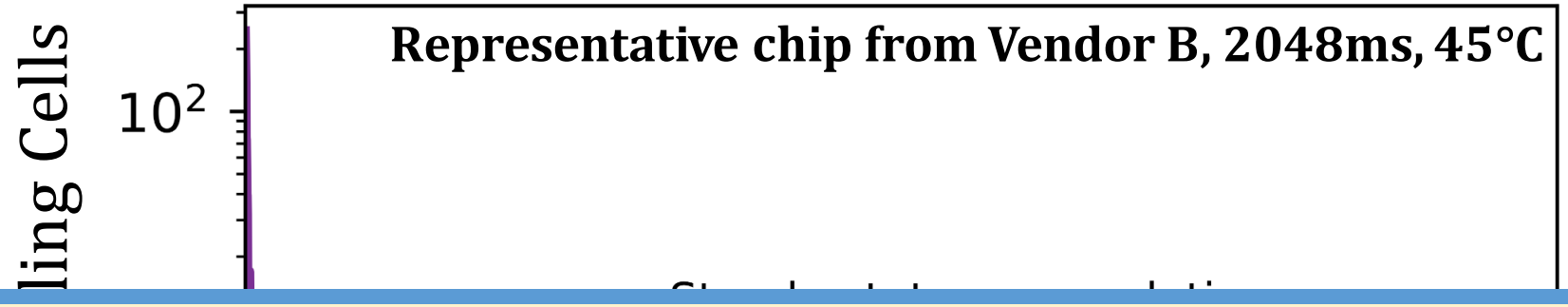
DRAM Architecture Background



Sources of Retention Time Variation

- **Process/voltage/temperature**
- **Data pattern dependence (DPD)**
  - Retention times **change with data** in cells/neighbors
  - e.g., all 1's vs. all 0's
- **Variable retention time (VRT)**
  - Retention time changes **randomly (unpredictably)**
  - Due to a combination of various circuit effects

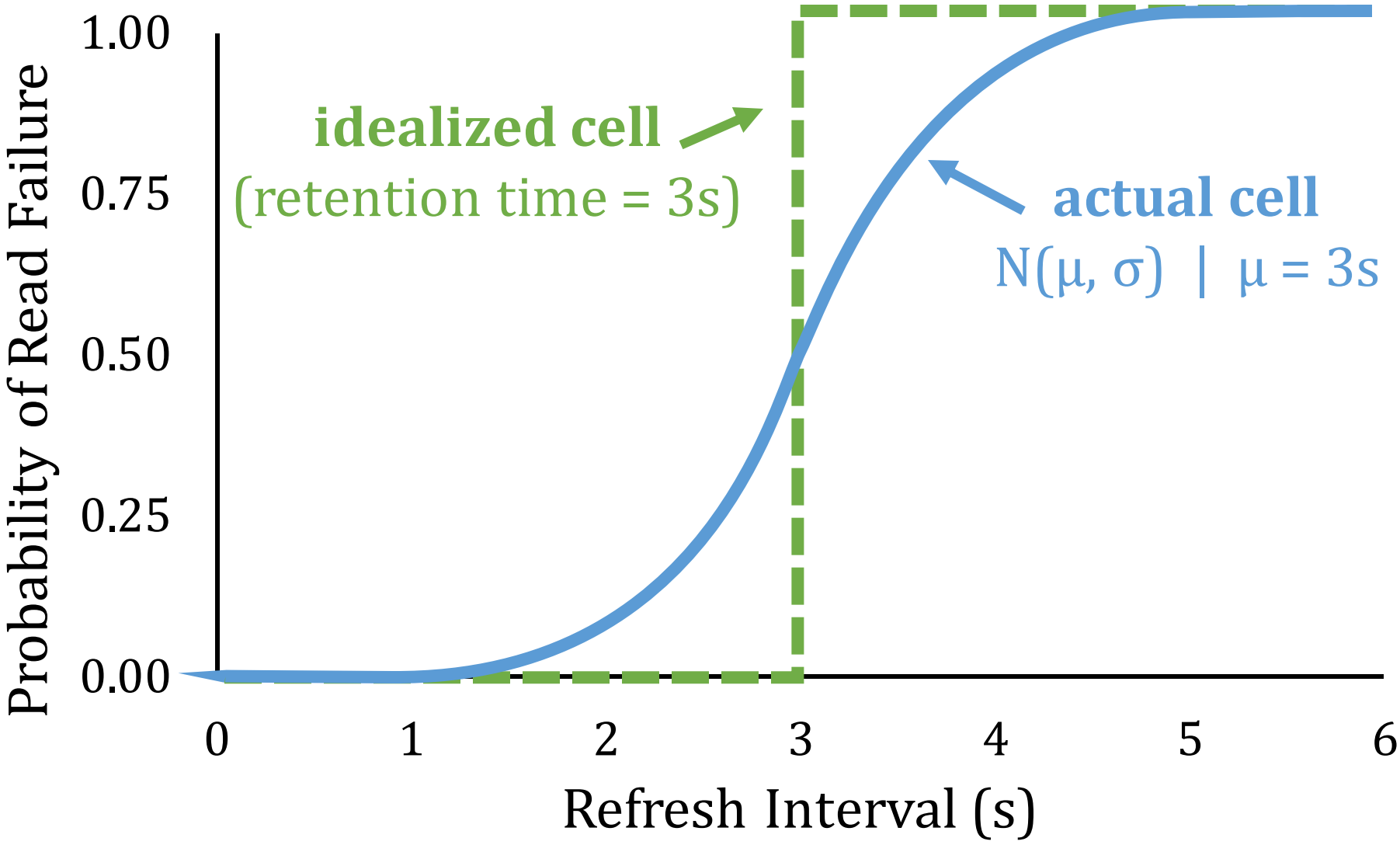
Long-term Continuous Profiling



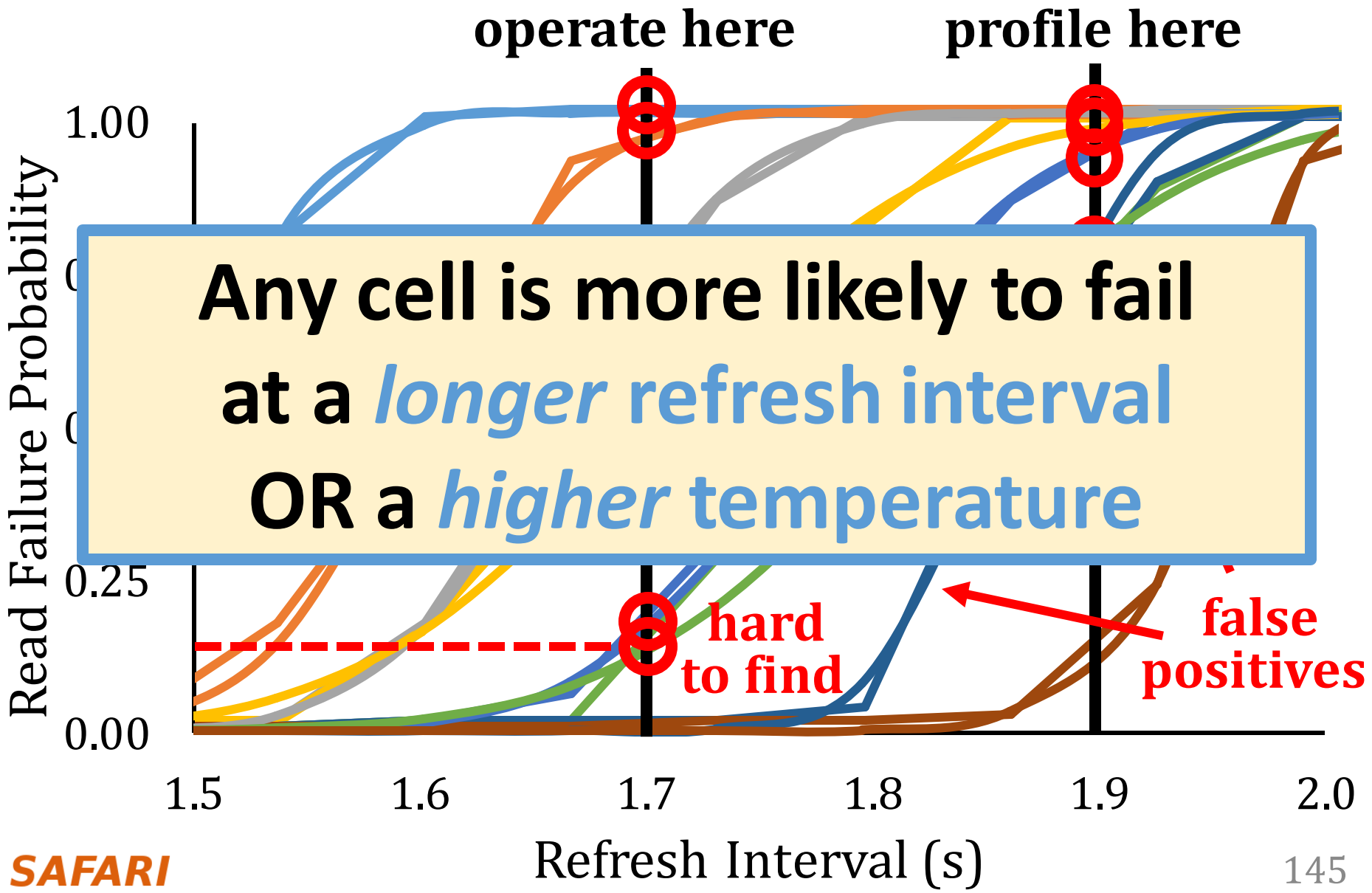
Error correction codes (ECC) and online profiling are *necessary* to manage new failing cells

- New failing cells continue to appear over time
  - Attributed to **variable retention time (VRT)**
- The set of failing cells changes over time

Single-cell Failure Probability (Cartoon)



Single-cell Failure Probability (Real)



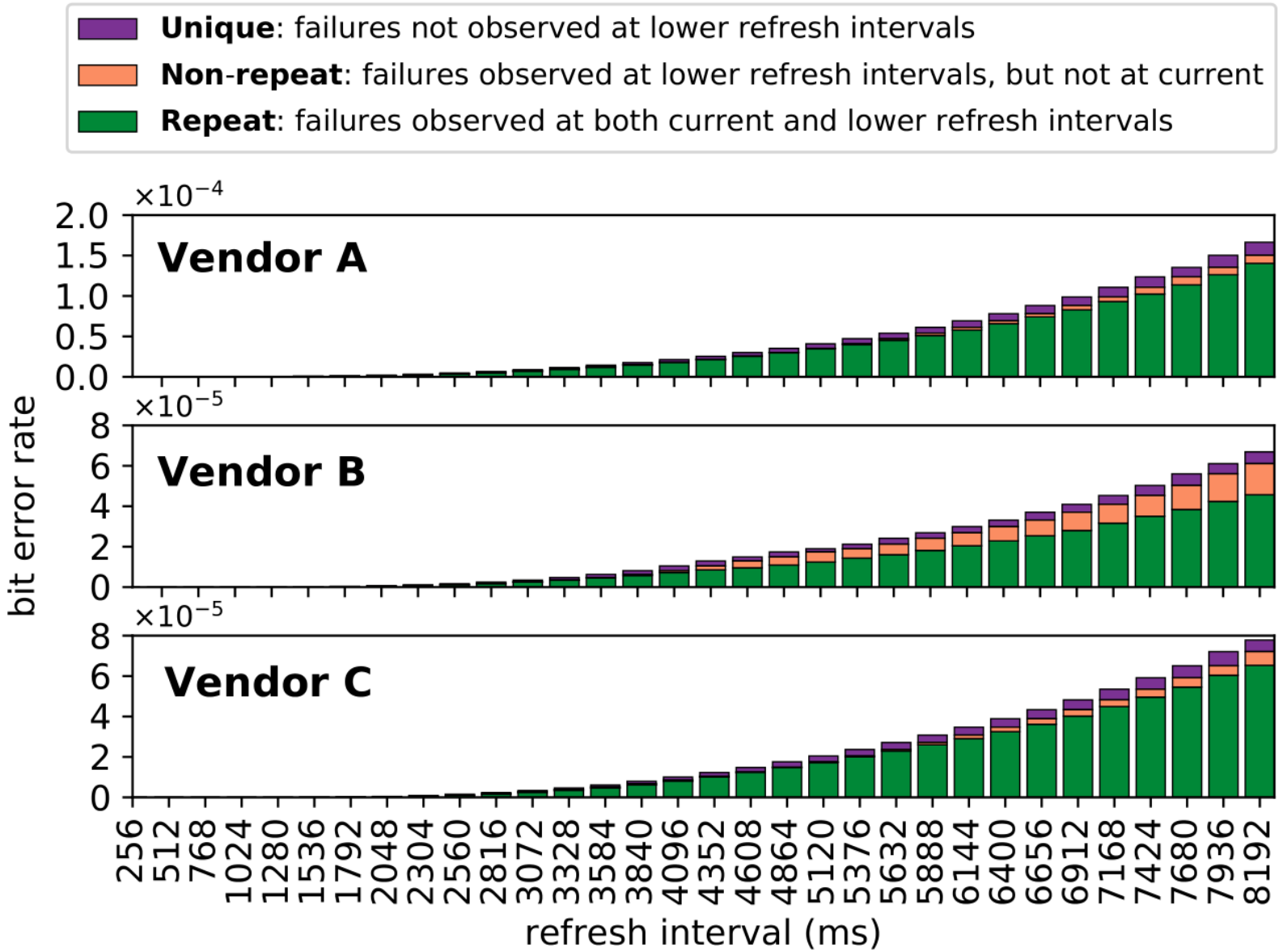
Temperature Relationship

- Well-fitting exponential relationship:

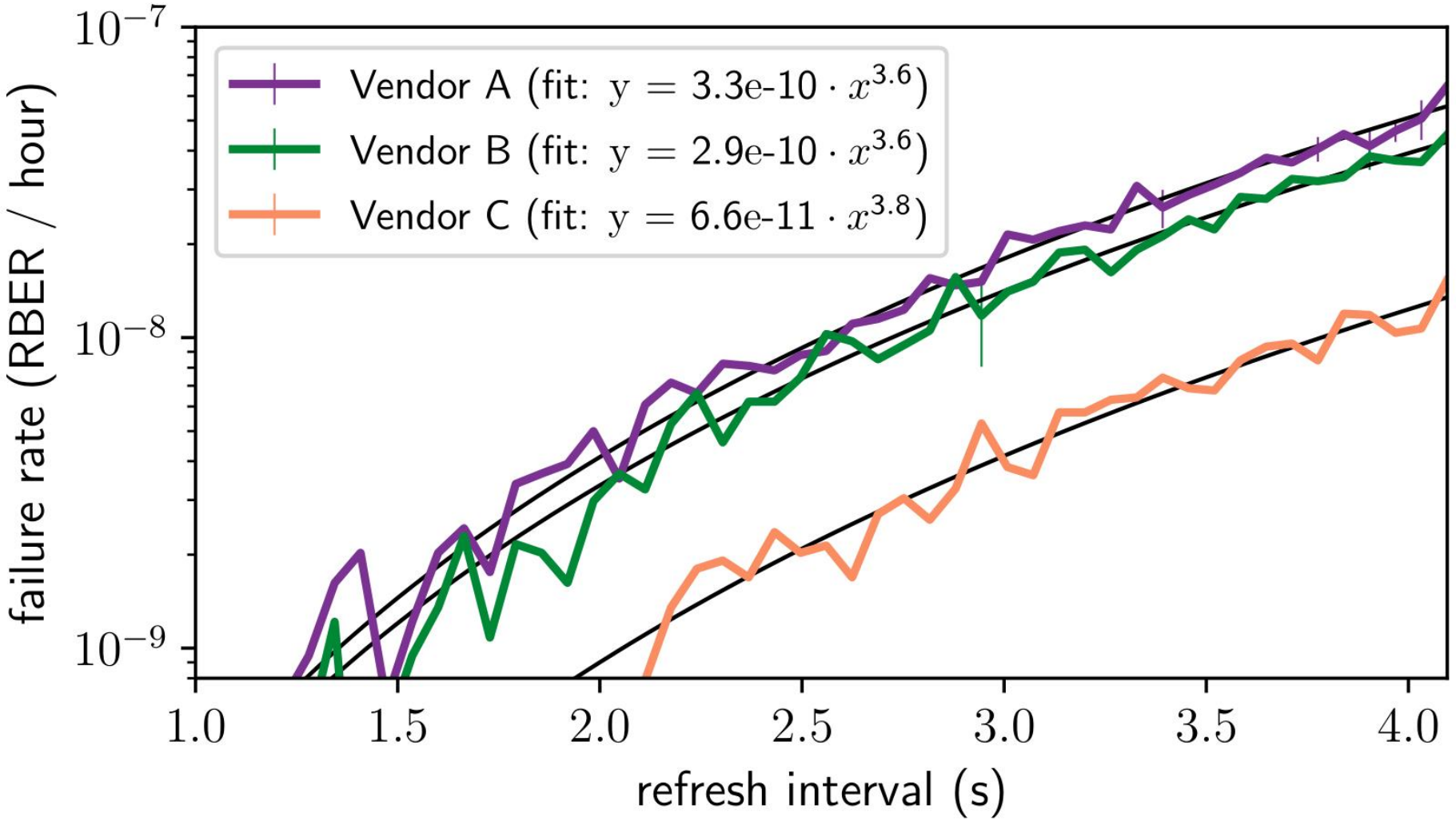
$$R_A \propto e^{0.22\Delta T} \qquad R_B \propto e^{0.20\Delta T} \qquad R_C \propto e^{0.26\Delta T}$$

- E.g., 10°C ~ 10x more failures

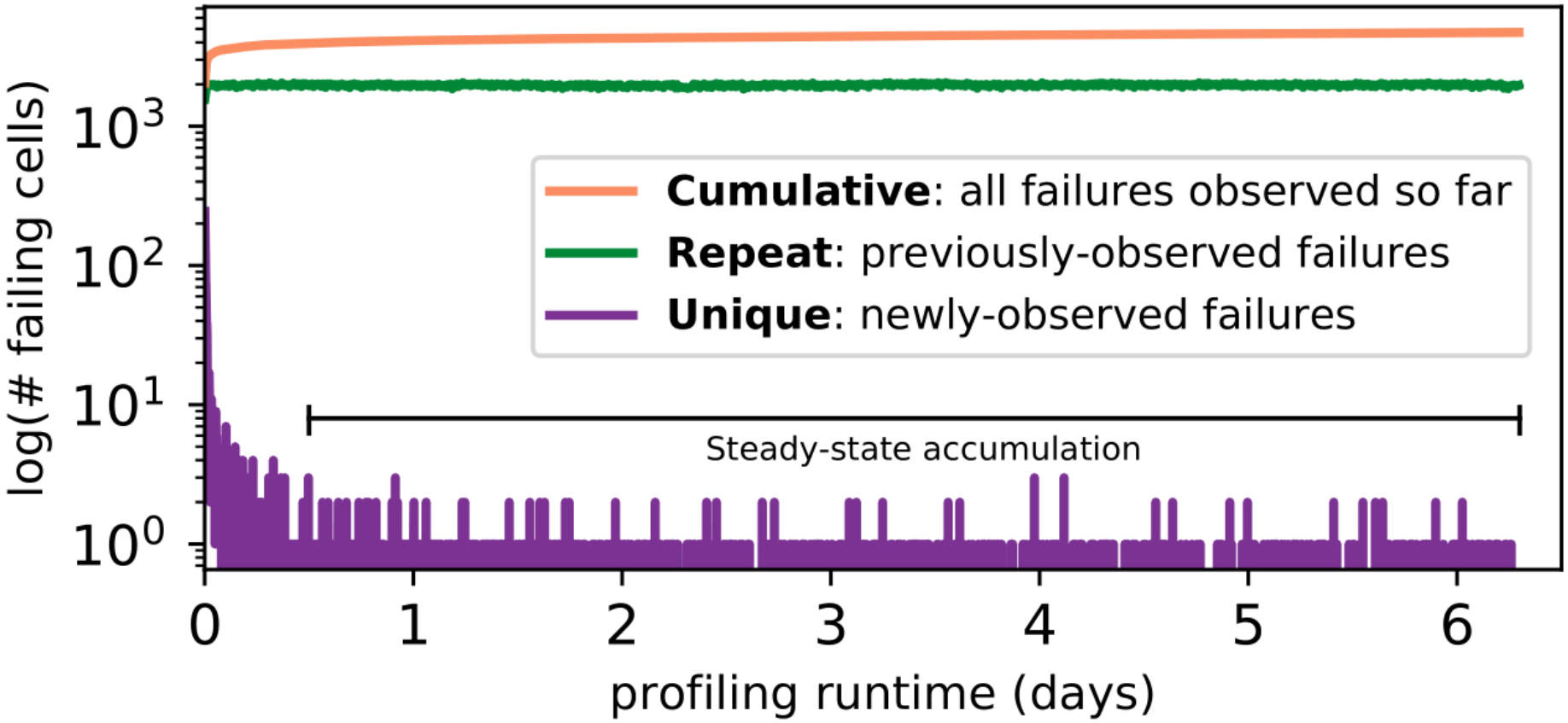
Retention Failures @ 45°C



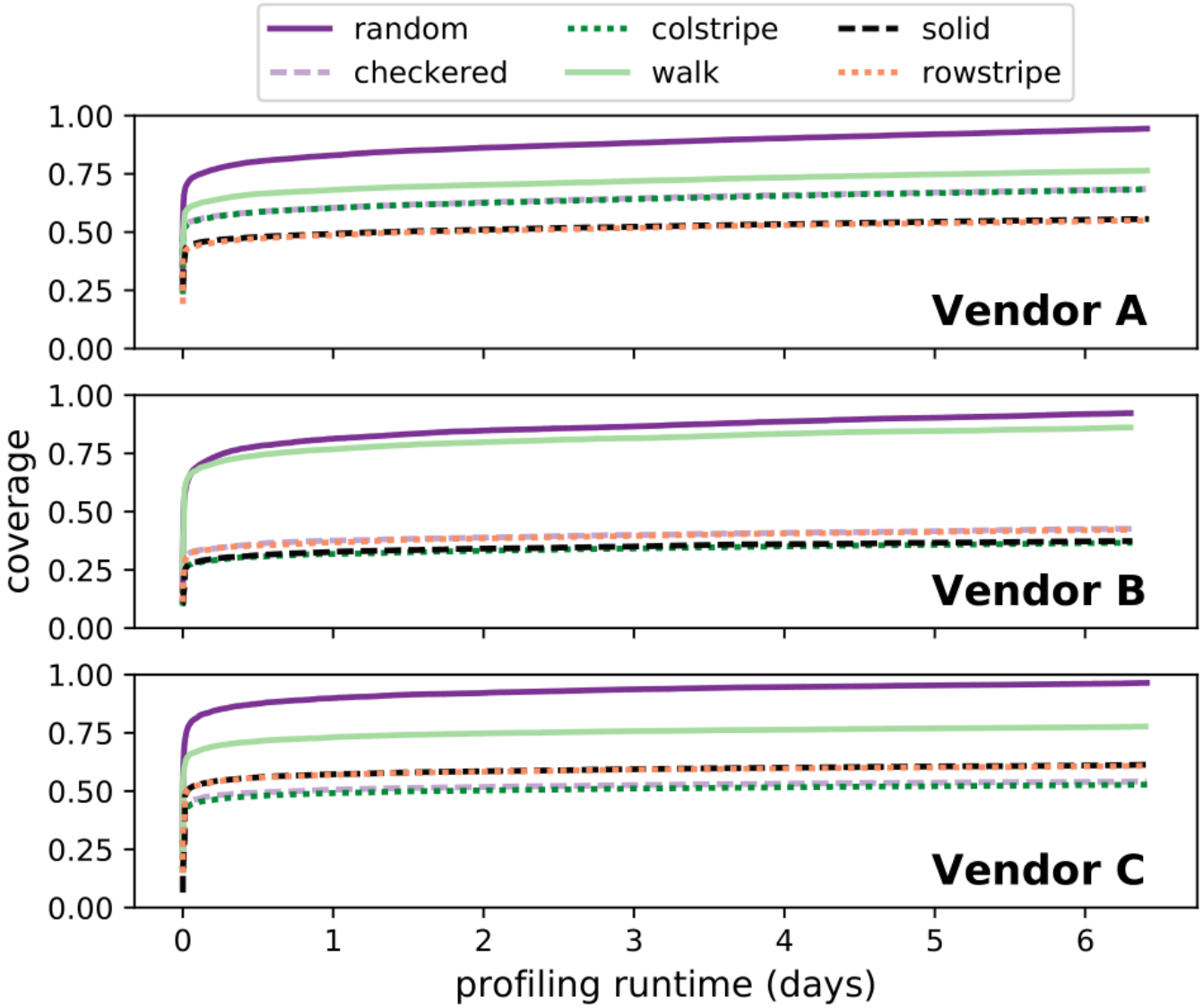
VRT Failure Accumulation Rate



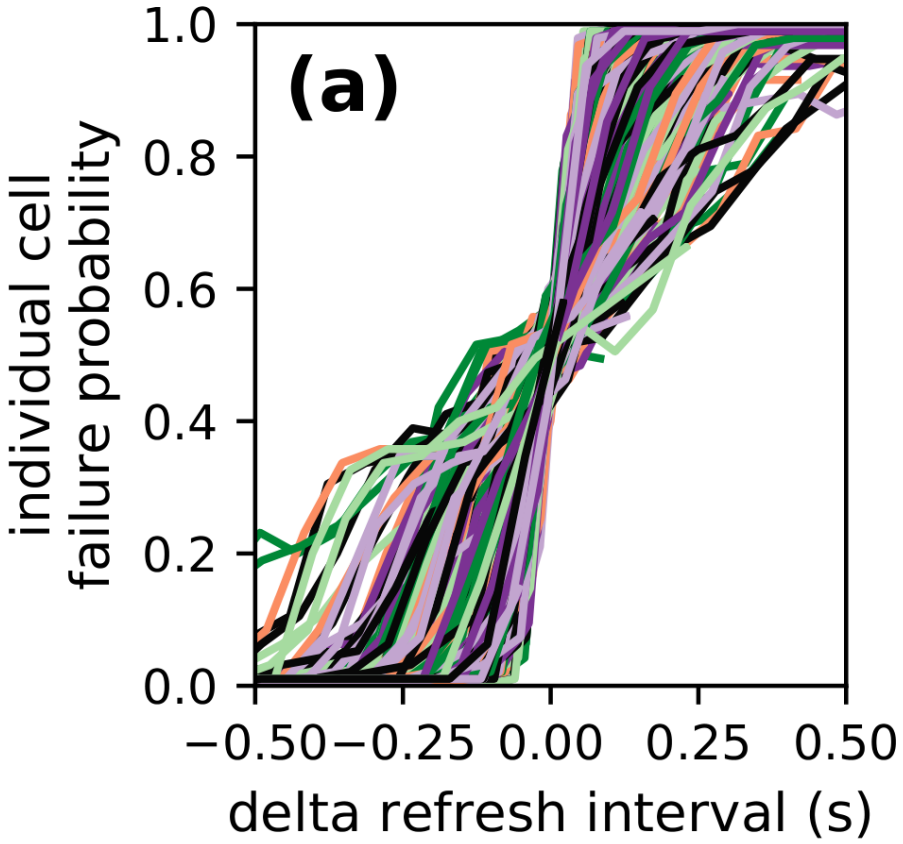
800 Rounds of Profiling @ 2048ms, 45°C

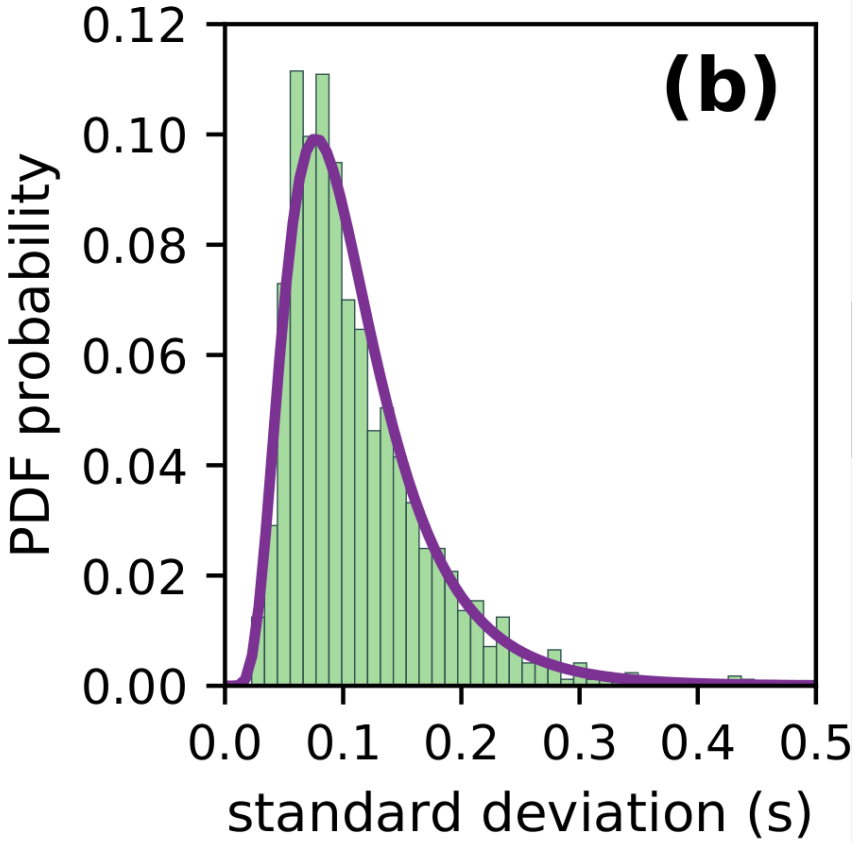


800 Rounds of Profiling @ 2048ms, 45°C



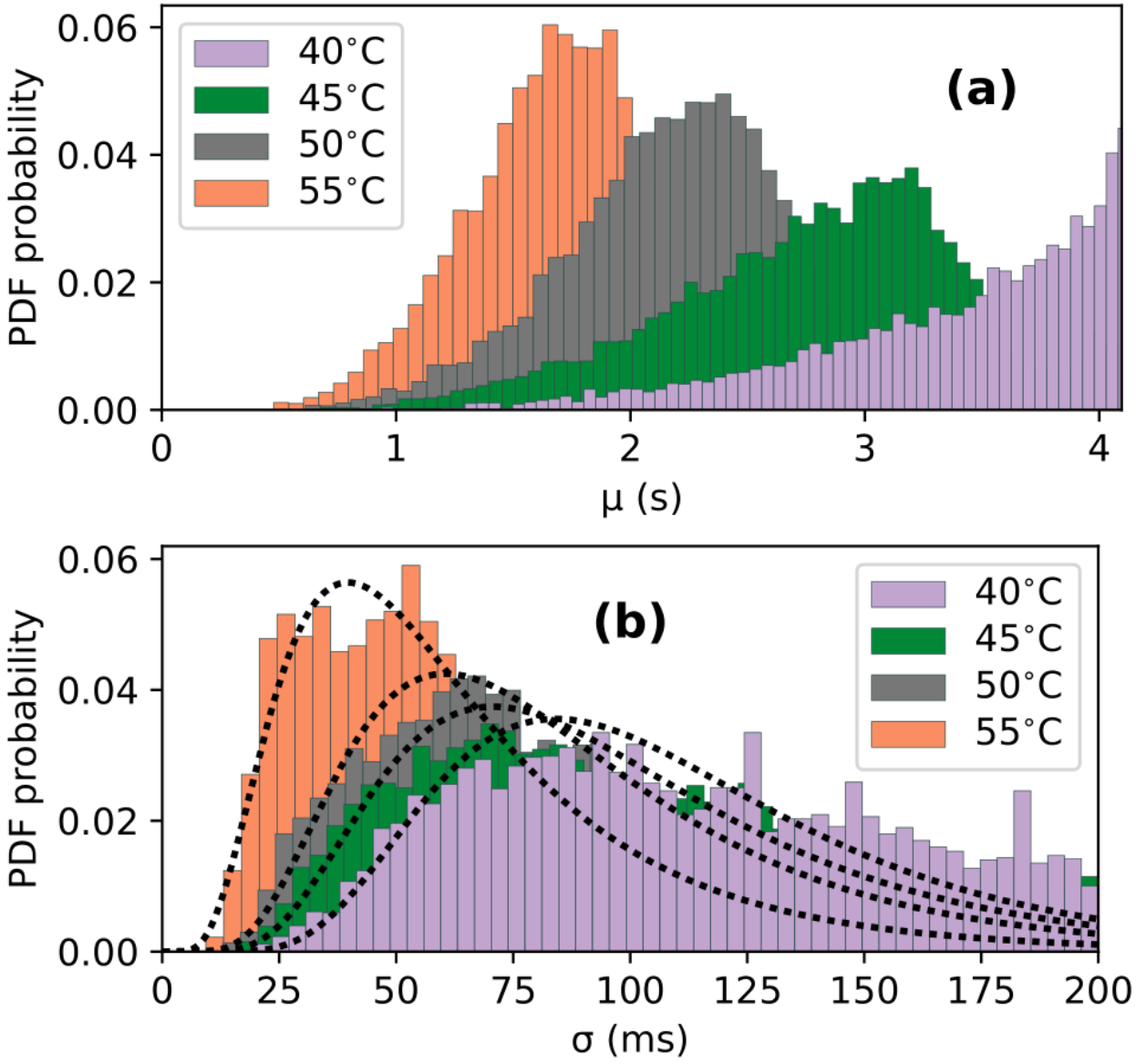
Individual Cell Failure Probabilities



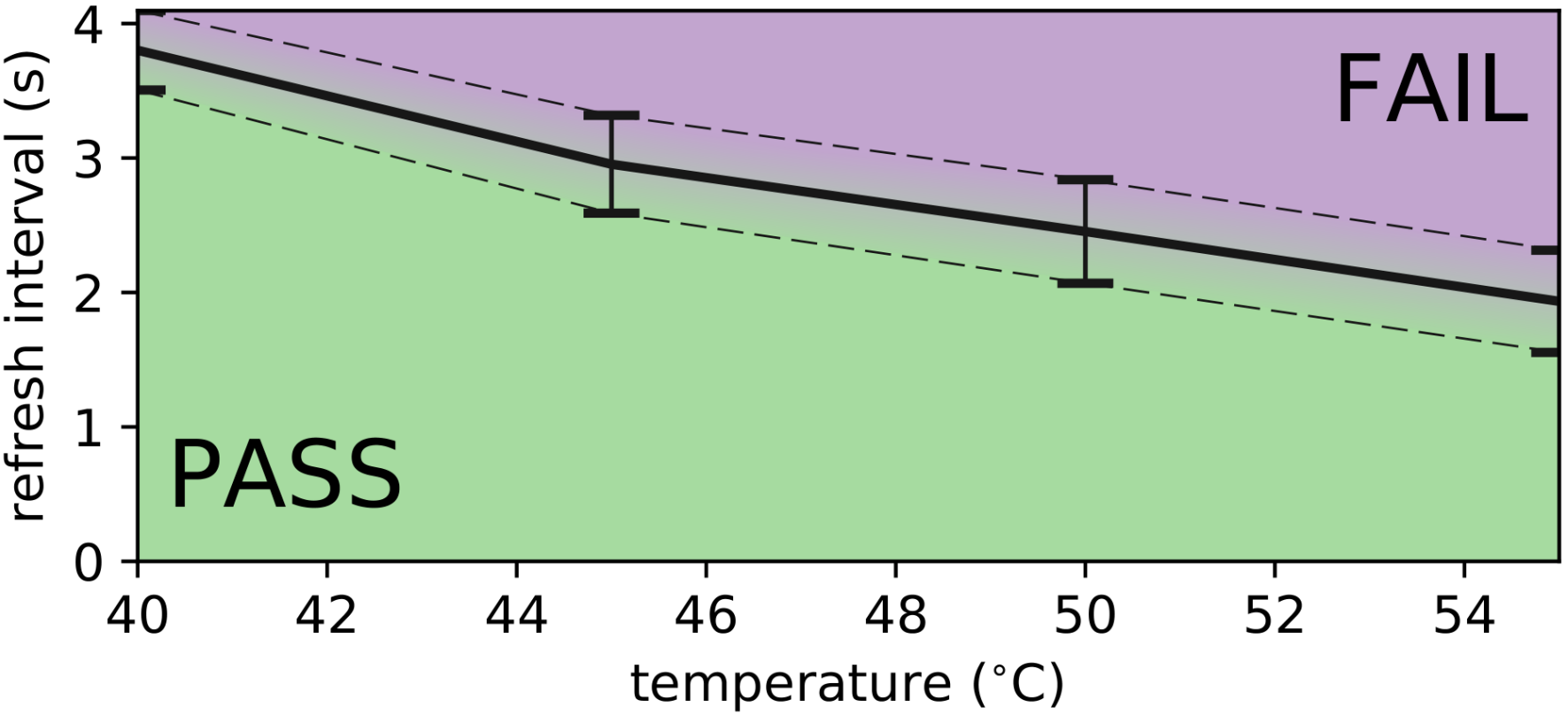


- Single representative chip of Vendor B at 40° C
- Refresh intervals ranging from 64ms to 4096ms

Individual Cell Failure Distributions



Single-cell Failures With Temperature

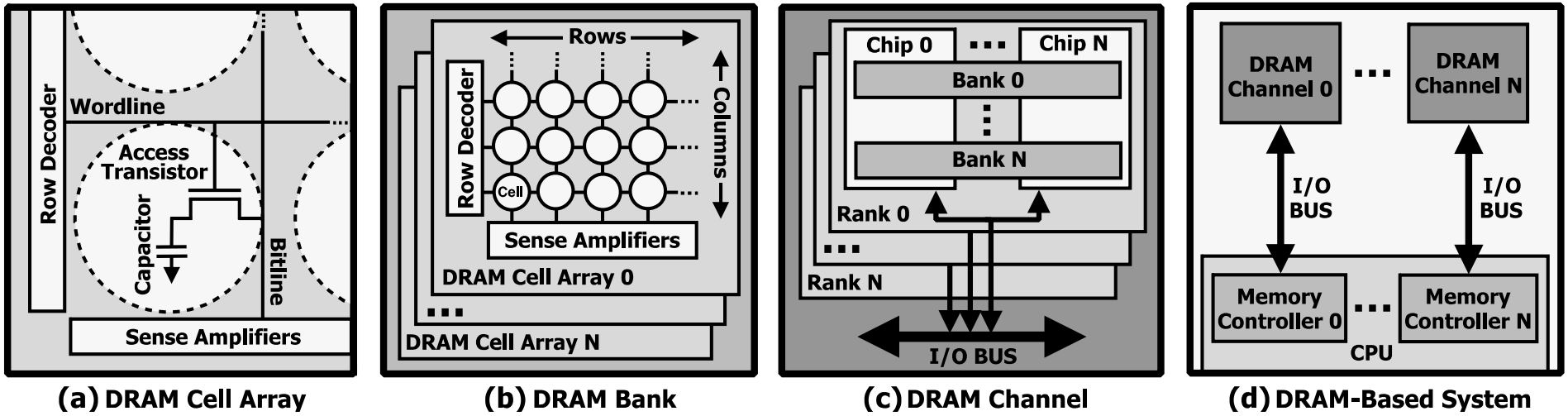


- Single representative chip of Vendor B
- {mean, std} for cells between 64ms and 4096ms

DRAM Latency PUF

Backup Slides

DRAM Architecture Background



Evaluating DRAM Retention PUFs

Algorithm 1: Evaluate Retention PUF [103, 120, 121, 124, 135]

```
1  evaluate_DRAM_retention_PUF(seg_id, wait_time):
2      rank_id ← DRAM rank containing seg_id
3      disable refresh for Rank[rank_id]
4      start_time ← current_time()
5      while current_time() - start_time < wait_time:
6          foreach row in Rank[rank_id]:
7              if row not in Segment[seg_id]:
8                  issue refresh to row                    // refresh all other rows
9      enable refresh for Rank[rank_id]
10     return data at Segment[seg_id]
```

	#Chips	#Tested Memory Segments
A	91	17,408
B	65	12,544
C	67	10,580

Table 1: The number of tested PUF memory segments across the tested chips from each of the three manufacturers.

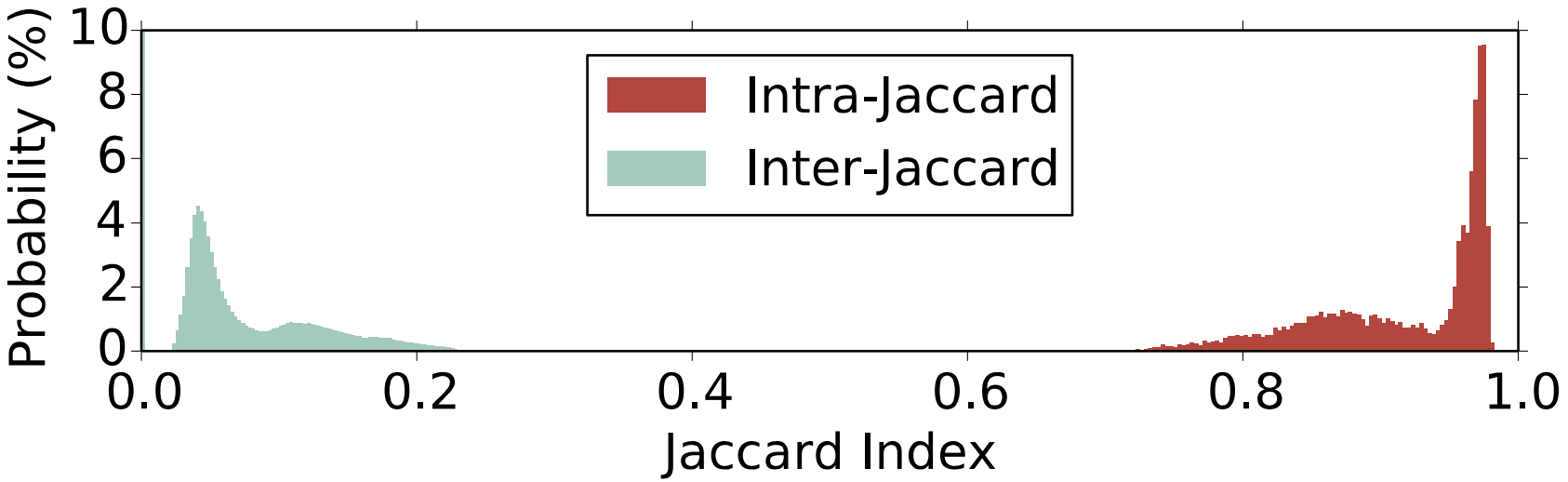


Figure 3: Distributions of Jaccard indices calculated across every possible pair of PUF responses across all tested PUF memory segments from each of 223 LPDDR4 DRAM chips.

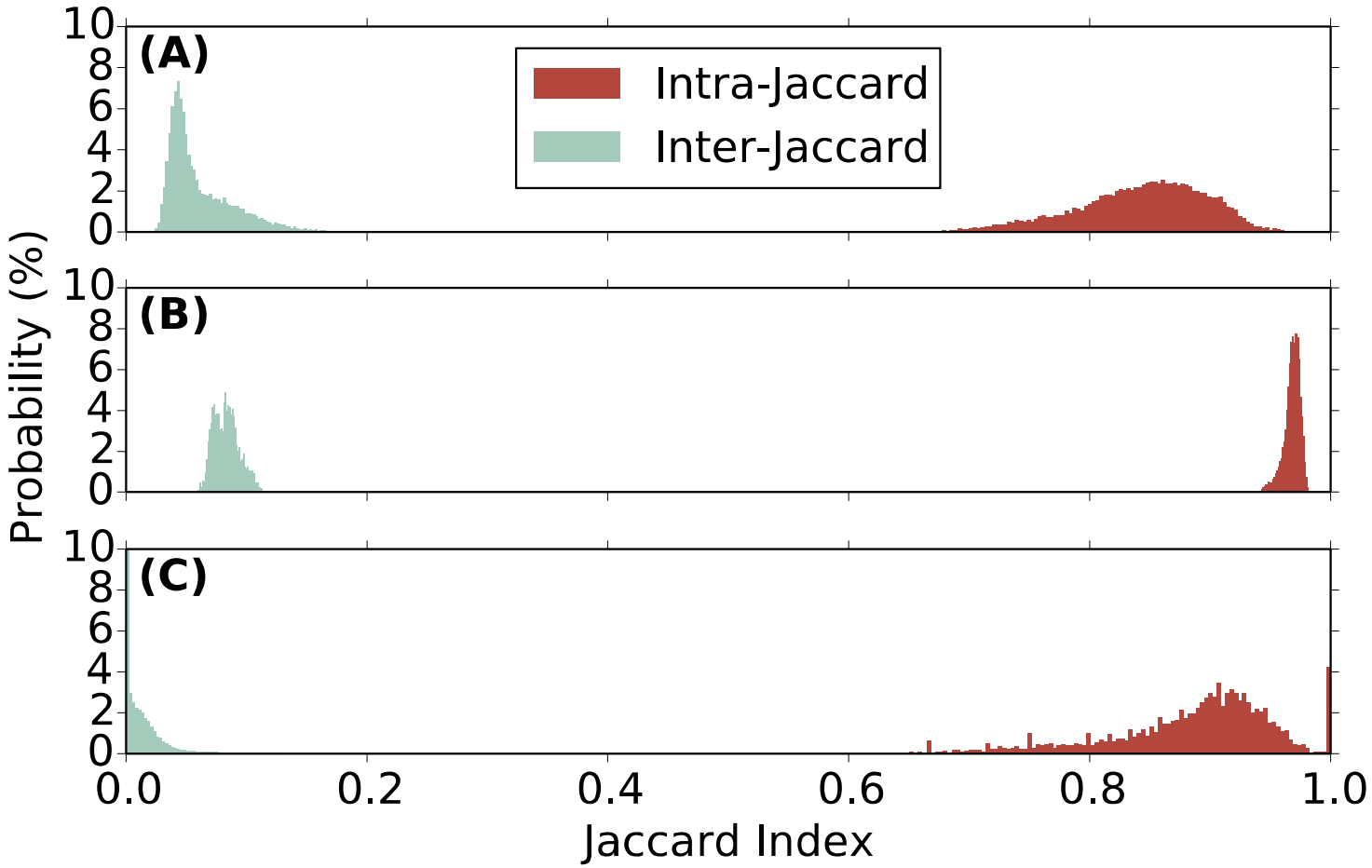


Figure 4: Distributions of Jaccard indices calculated between PUF responses of DRAM chips from a single manufacturer.

	#Chips	#Total Memory Segments
A	19	589,824
B	12	442,879
C	14	437,990

Table 2: Number of PUF memory segments tested for 30 days.

	%Memory Segments per Chip	
	Intra-Jaccard index range <0.1	Intra-Jaccard index range <0.2
A	100.00 [99.08, 100.00]	100.00 [100.00, 100.00]
B	90.39 [82.13, 99.96]	96.34 [95.37, 100.00]
C	95.74 [89.20, 100.00]	96.65 [95.48, 100.00]

Table 3: Percentage of PUF memory segments per chip with Intra-Jaccard index ranges <0.1 or 0.2 over a 30-day period. Median [minimum, maximum] values are shown.

Temperature Effects

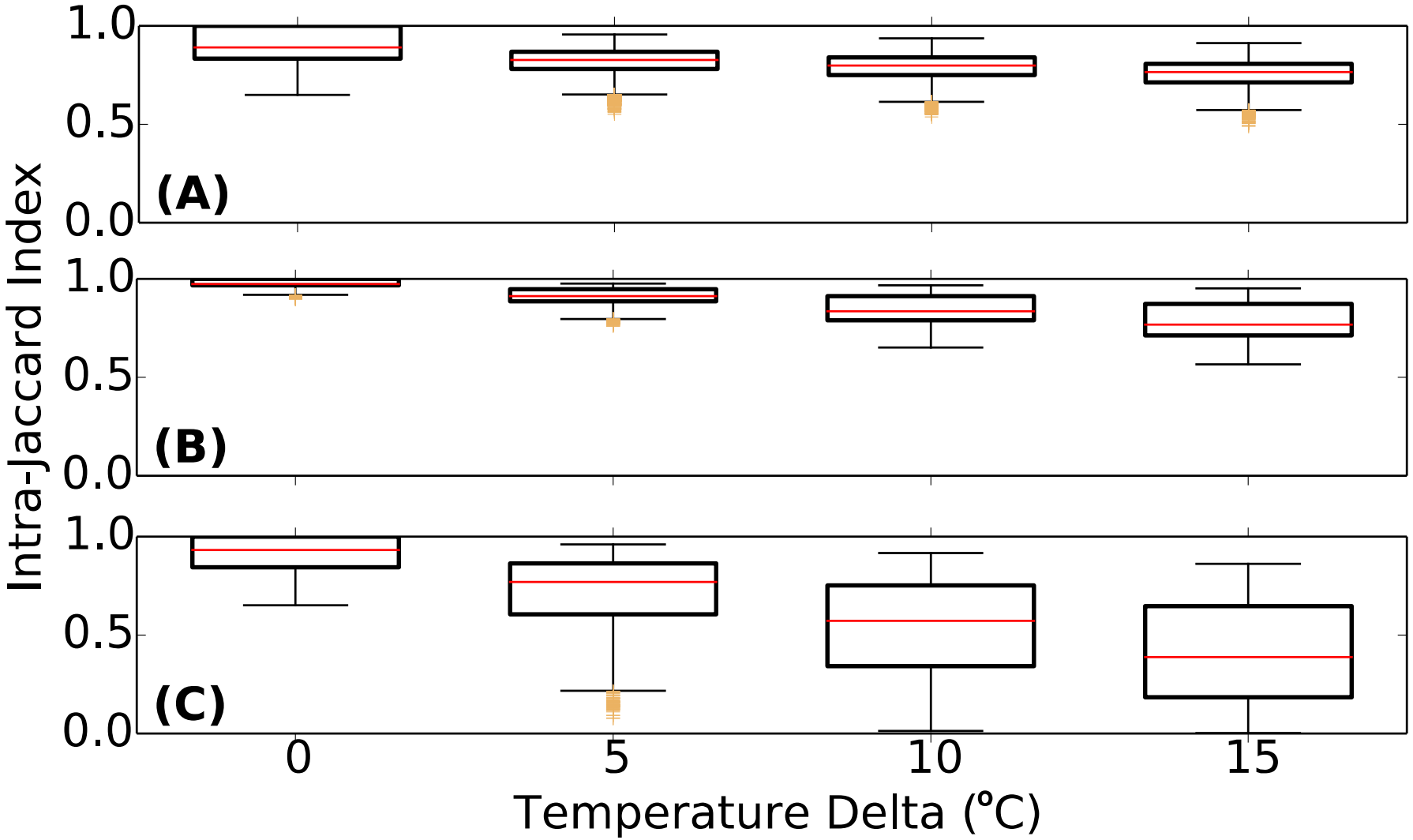


Figure 6: DRAM latency PUF repeatability vs. temperature.

Evaluating a DRAM Latency PUF

Algorithm 2: Evaluate DRAM latency PUF

```
1  evaluate_DRAM_latency_PUF(seg_id):
2      write known data (all 1's) to Segment[seg_id]
3      rank_id ← DRAM rank containing seg_id
4      obtain exclusive access to Rank[rank_id]
5      set low t_RCD for Rank[rank_id]
6      for i = 1 to num_iterations :
7          for col in Segment[seg_id]
8              for row in Segment[seg_id]:          // column-order reads
9                  read()                           // induce read failures
10                 memory_barrier()                 // one access at a time
11                 count_failures()                 // record in another rank
12     set default t_RCD for Rank[rank_id]
13     filter the PUF memory segment                // See Filtering Mechanism
14     release exclusive access to Rank[rank_id]
15     return error pattern at Segment[seg_id]
```

Memory Footprint. Equation 2 provides the memory footprint required by PUF evaluation:

$$mem_{total} = (size_{mem\_seg}) + (size_{counter\_buffer}) \quad (2)$$

where $size_{mem\_seg}$ is the size of the PUF memory segment and $size_{counter\_buffer}$ is the size of the counter buffer. The size of the counter buffer can be calculated using Equation 3:

$$size_{counter\_buffer} = (size_{mem\_seg}) \times \lceil \log_2 N_{iters} \rceil \quad (3)$$

	#Chips	Good Memory Segments per Chip (%)
A	19	100.00 [100.00, 100.00]
B	12	100.00 [64.06, 100.00]
C	14	30.86 [19.37, 95.31]

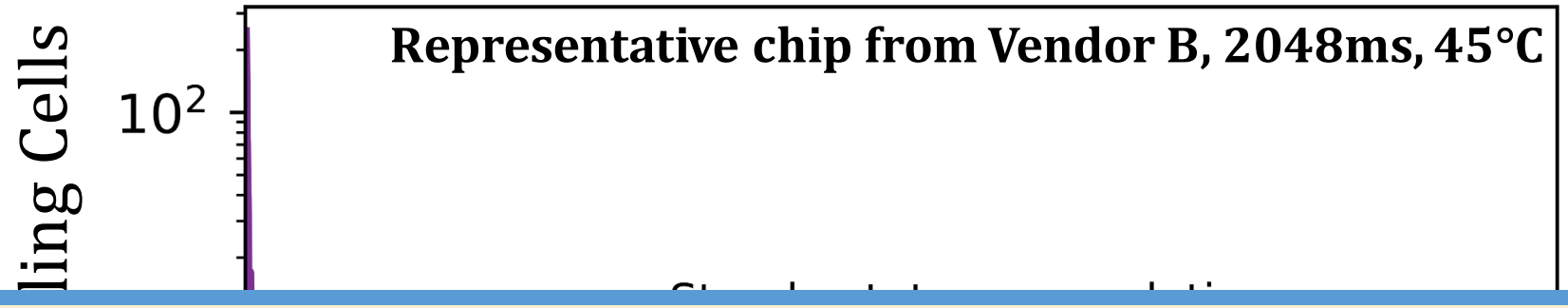
Table 4: Percentage of *good* memory segments per chip across manufacturers. Median [min, max] values are shown.

DRAM Characterization

Sources of Retention Time Variation

- **Process/voltage/temperature**
- **Data pattern dependence (DPD)**
  - Retention times **change with data** in cells/neighbors
  - e.g., all 1's vs. all 0's
- **Variable retention time (VRT)**
  - Retention time changes **randomly (unpredictably)**
  - Due to a combination of various circuit effects

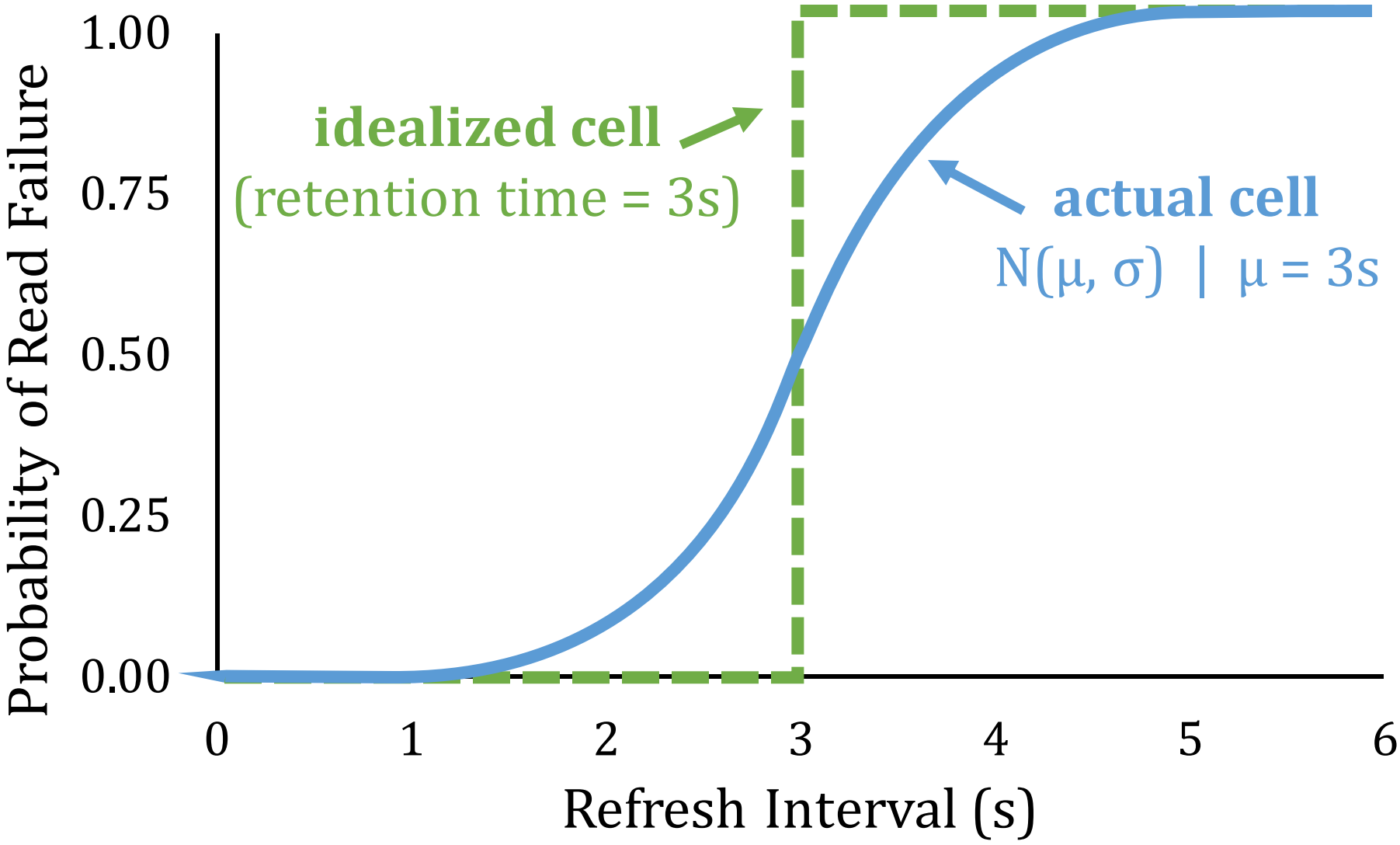
Long-term Continuous Profiling



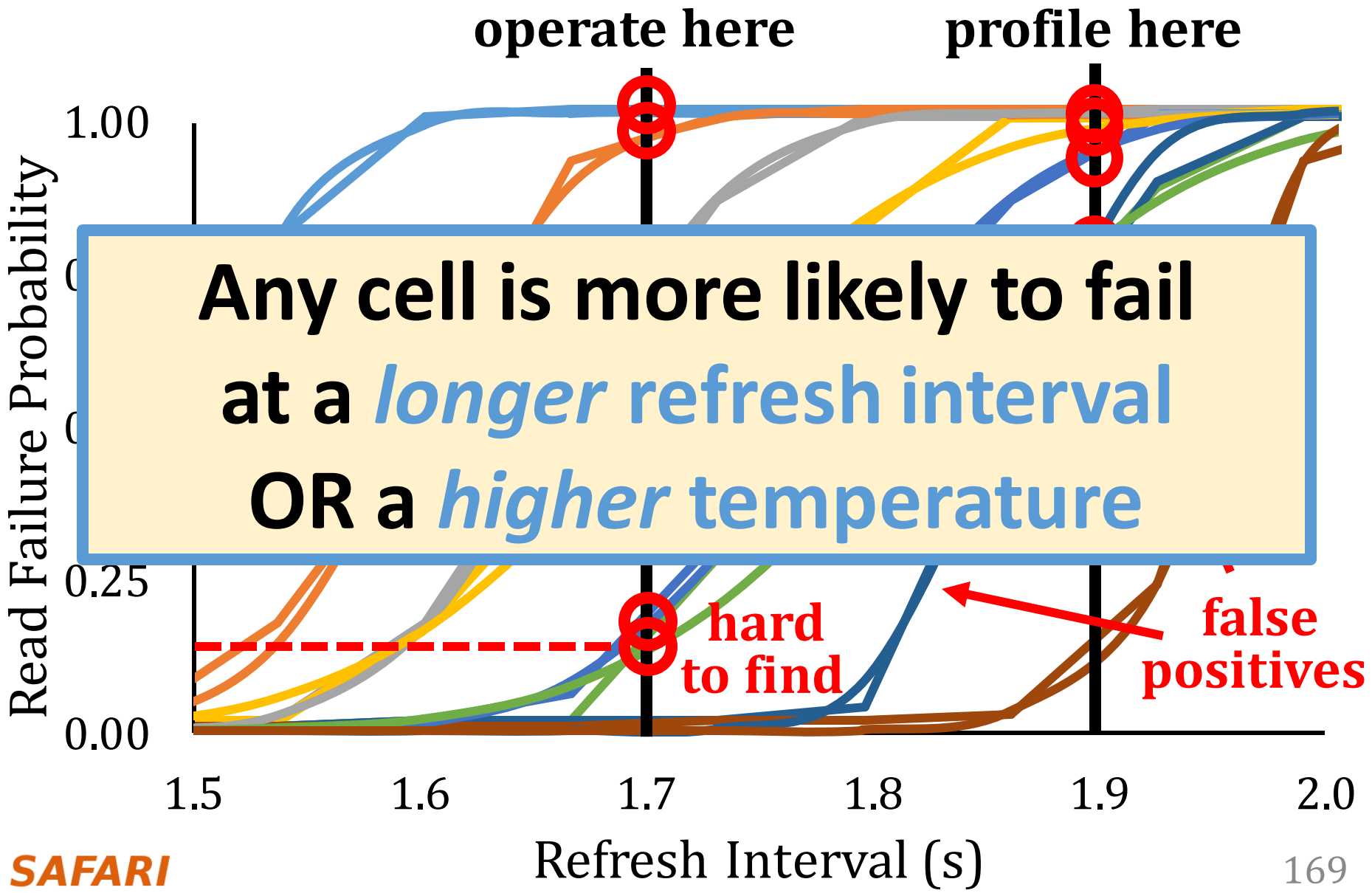
Error correction codes (ECC) and online profiling are *necessary* to manage new failing cells

- New failing cells continue to appear over time
  - Attributed to **variable retention time (VRT)**
- The set of failing cells changes over time

Single-cell Failure Probability (Cartoon)



Single-cell Failure Probability (Real)



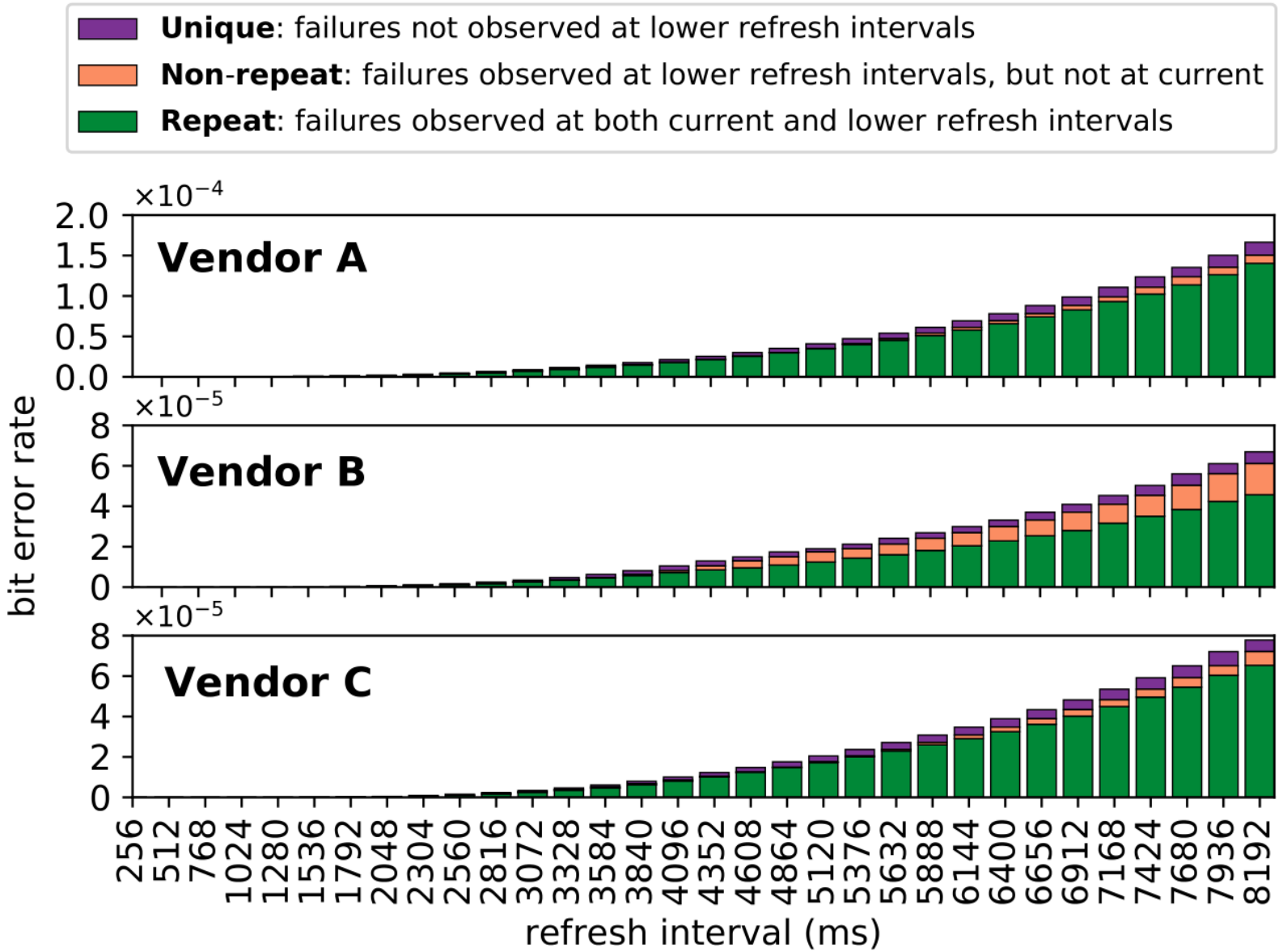
Temperature Relationship

- Well-fitting exponential relationship:

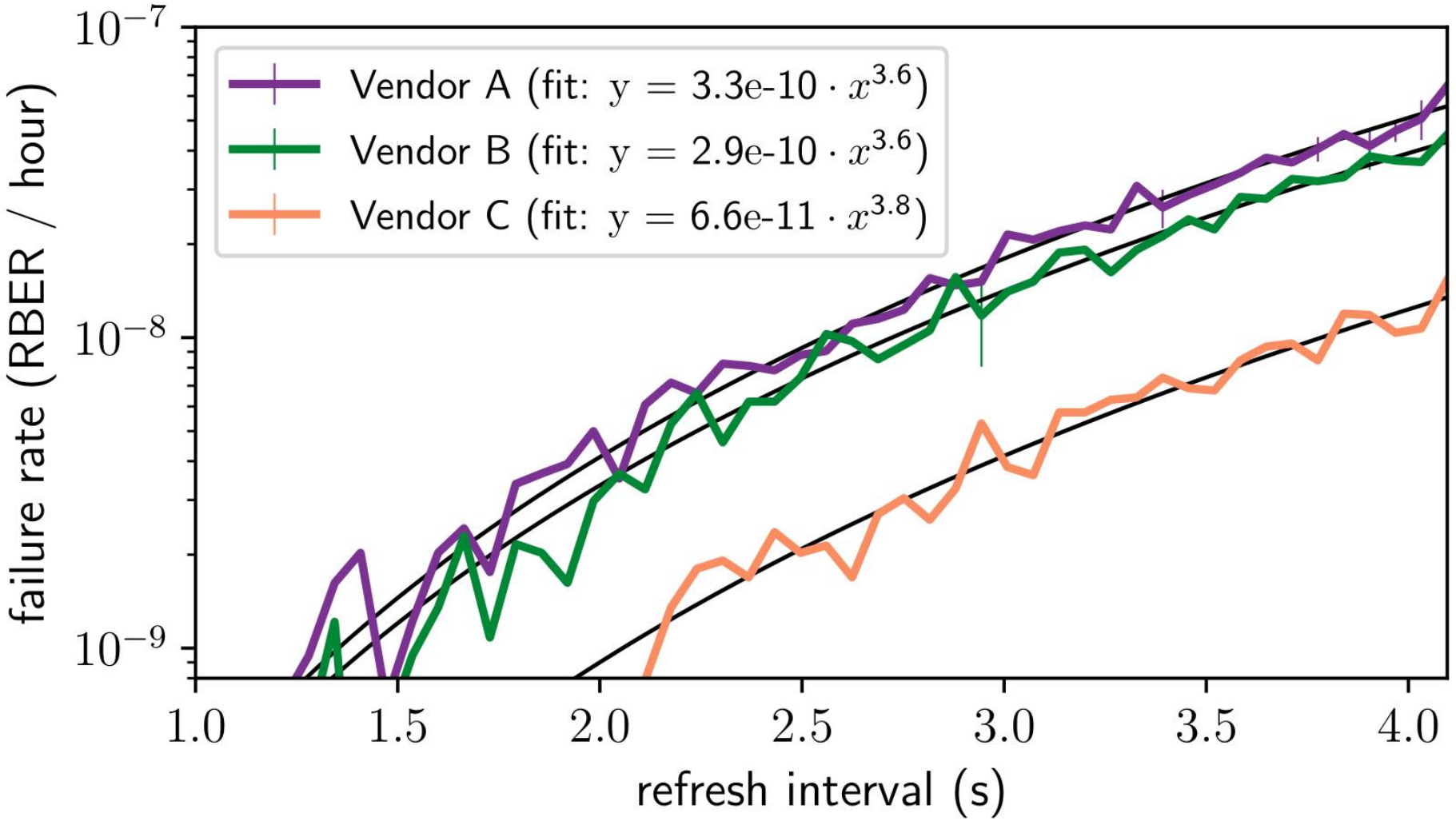
$$R_A \propto e^{0.22\Delta T} \qquad R_B \propto e^{0.20\Delta T} \qquad R_C \propto e^{0.26\Delta T}$$

- E.g., 10°C ~ 10x more failures

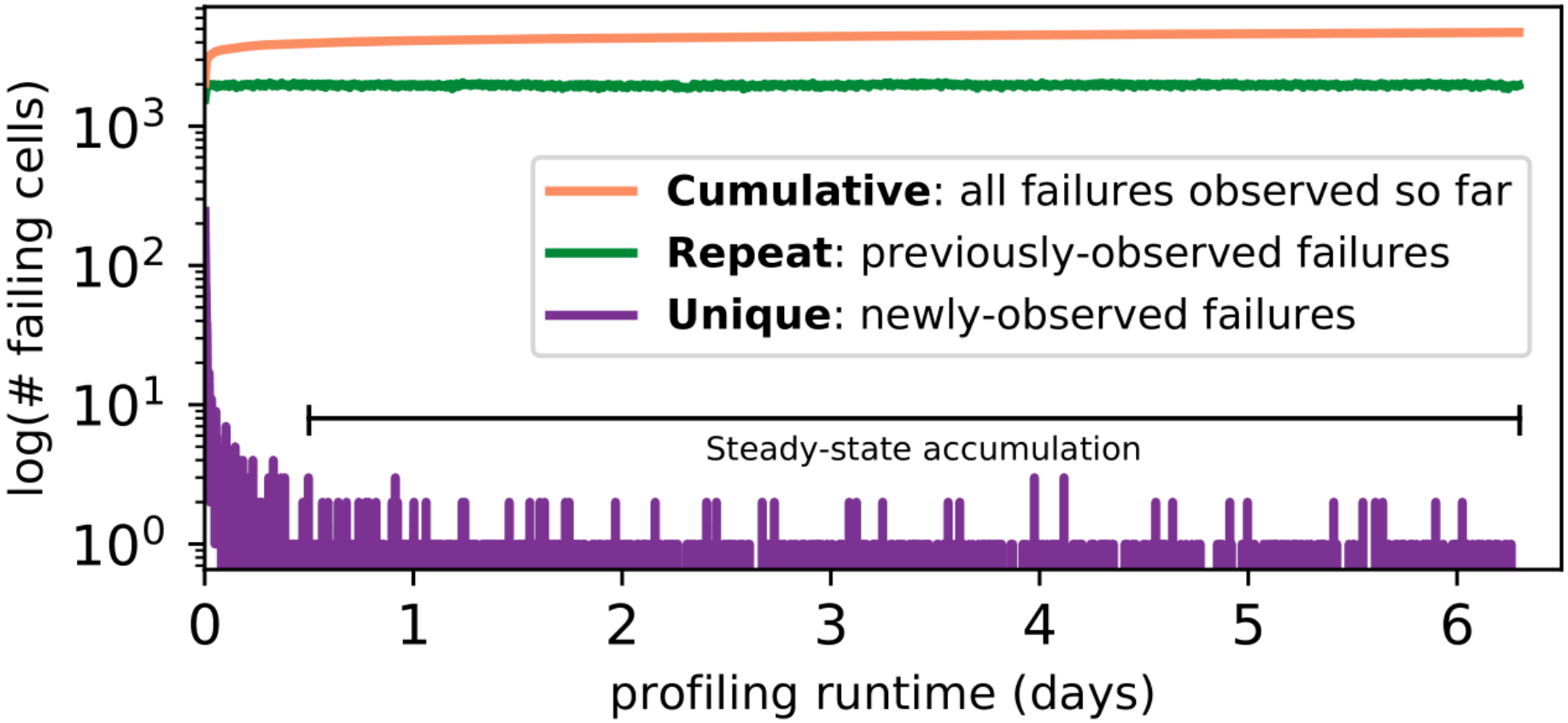
Retention Failures @ 45°C



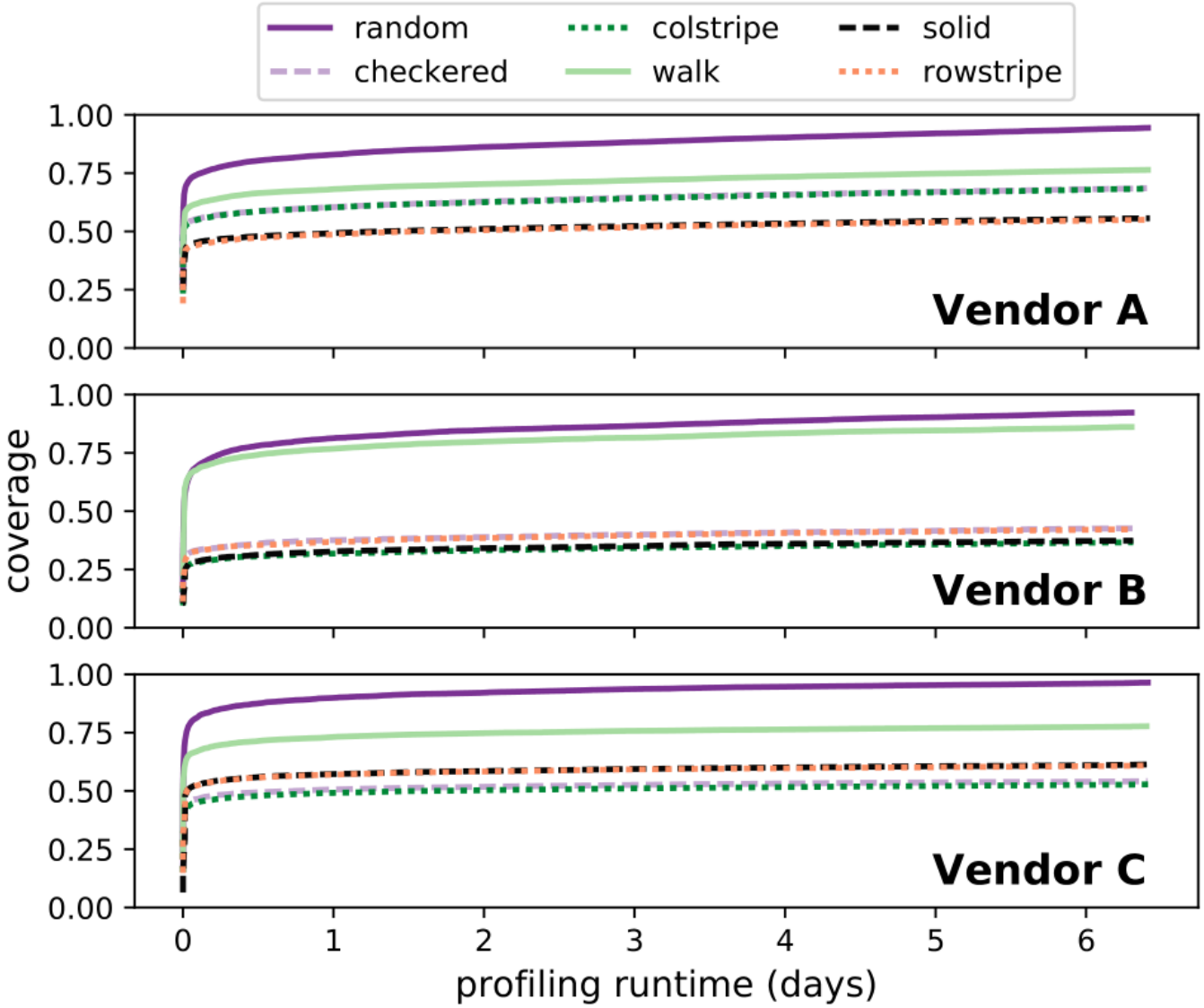
VRT Failure Accumulation Rate



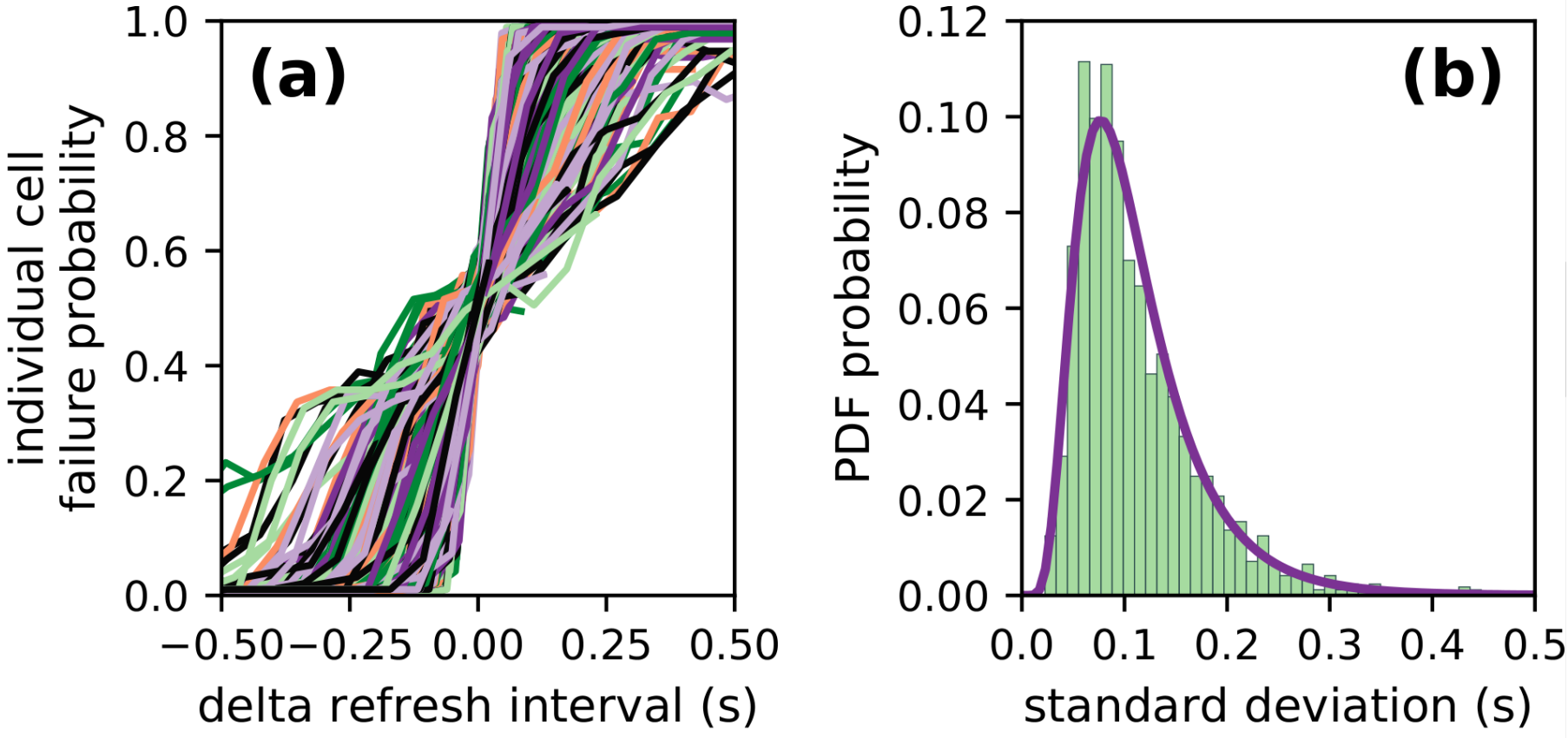
800 Rounds of Profiling @ 2048ms, 45°C



800 Rounds of Profiling @ 2048ms, 45°C

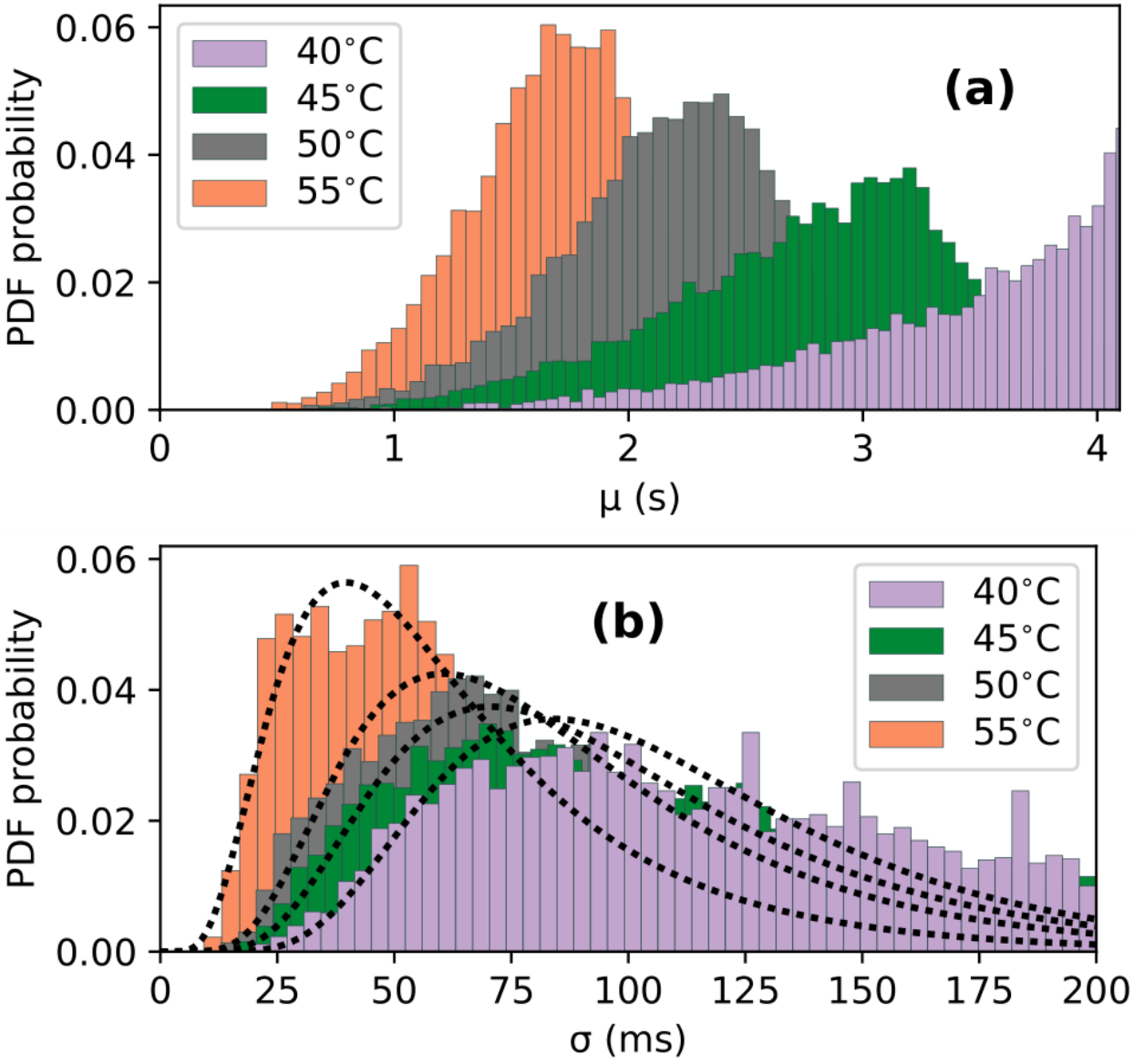


Individual Cell Failure Probabilities

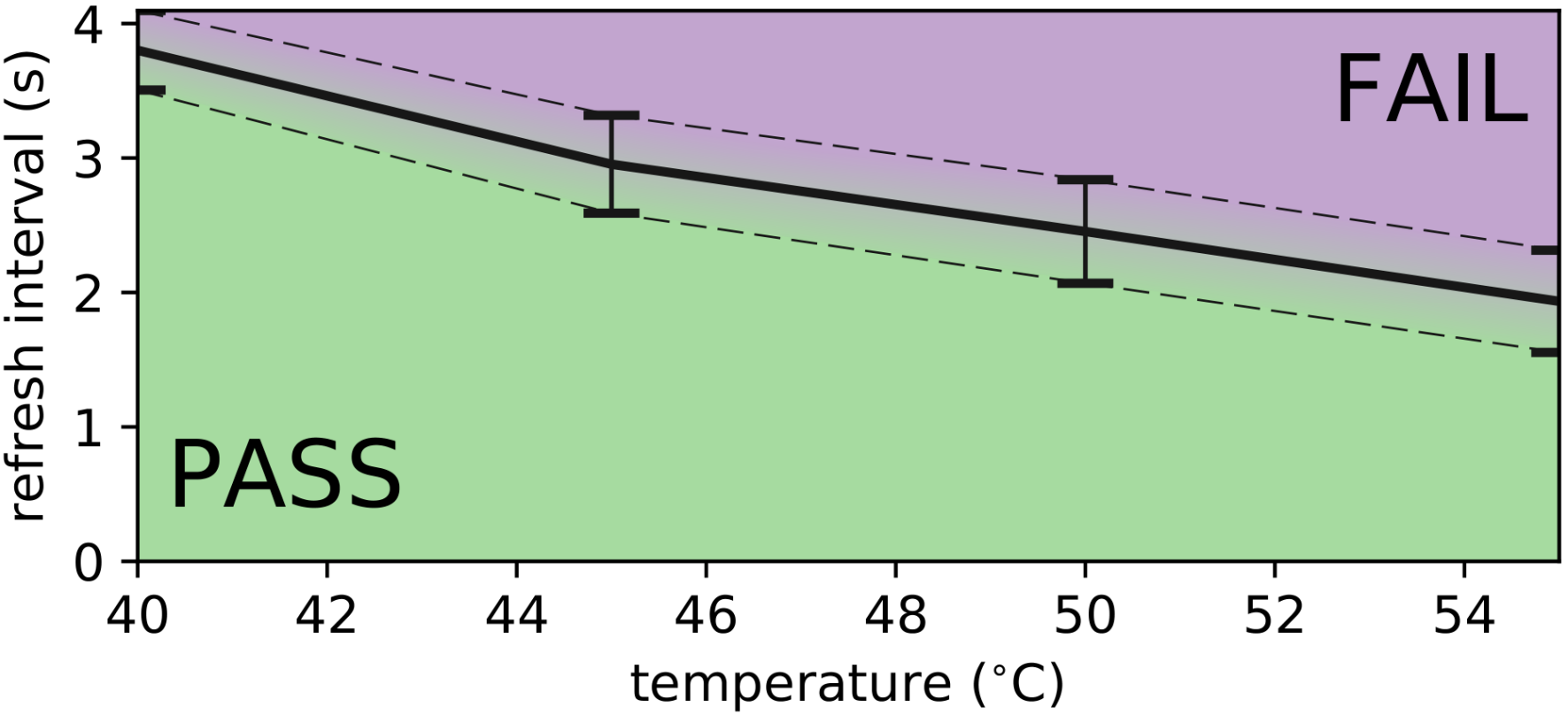


- Single representative chip of Vendor B at 40° C
- Refresh intervals ranging from 64ms to 4096ms

Individual Cell Failure Distributions



Single-cell Failures With Temperature



- Single representative chip of Vendor B
- {mean, std} for cells between 64ms and 4096ms