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

D-RaNGe Outline

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

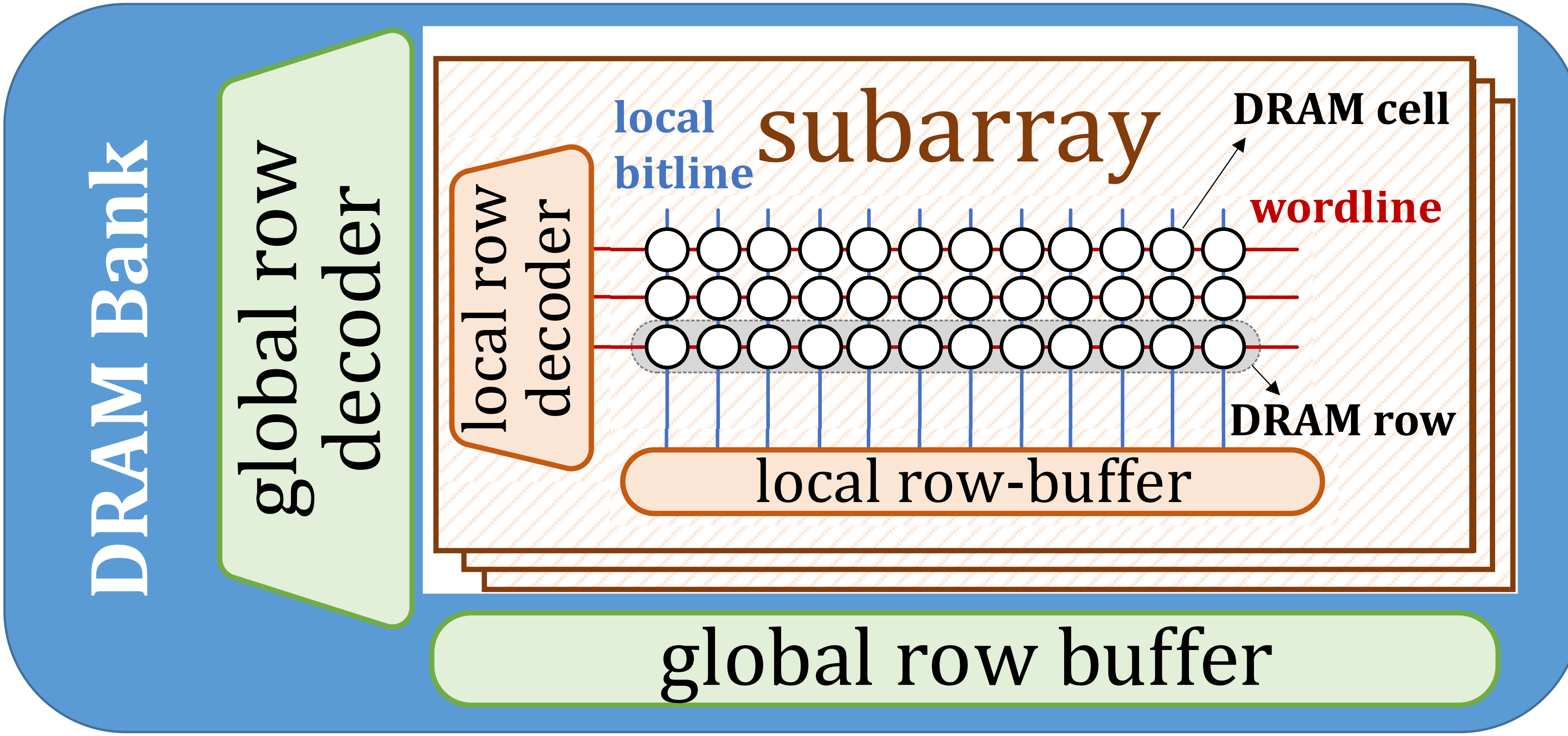
Effective TRNG Characteristics

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

DRAM Organization

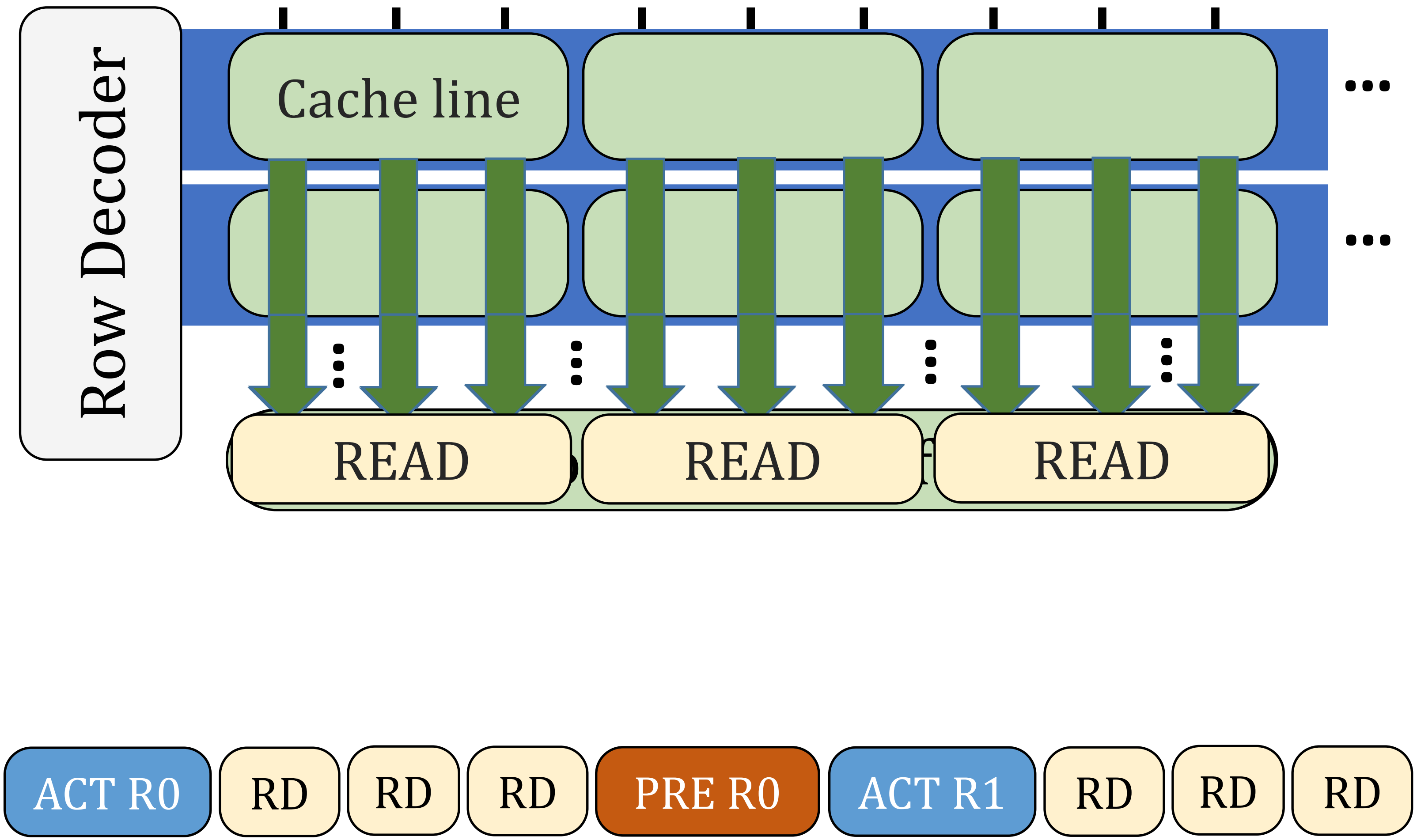
A DRAM bank is hierarchically organized into **subarrays**



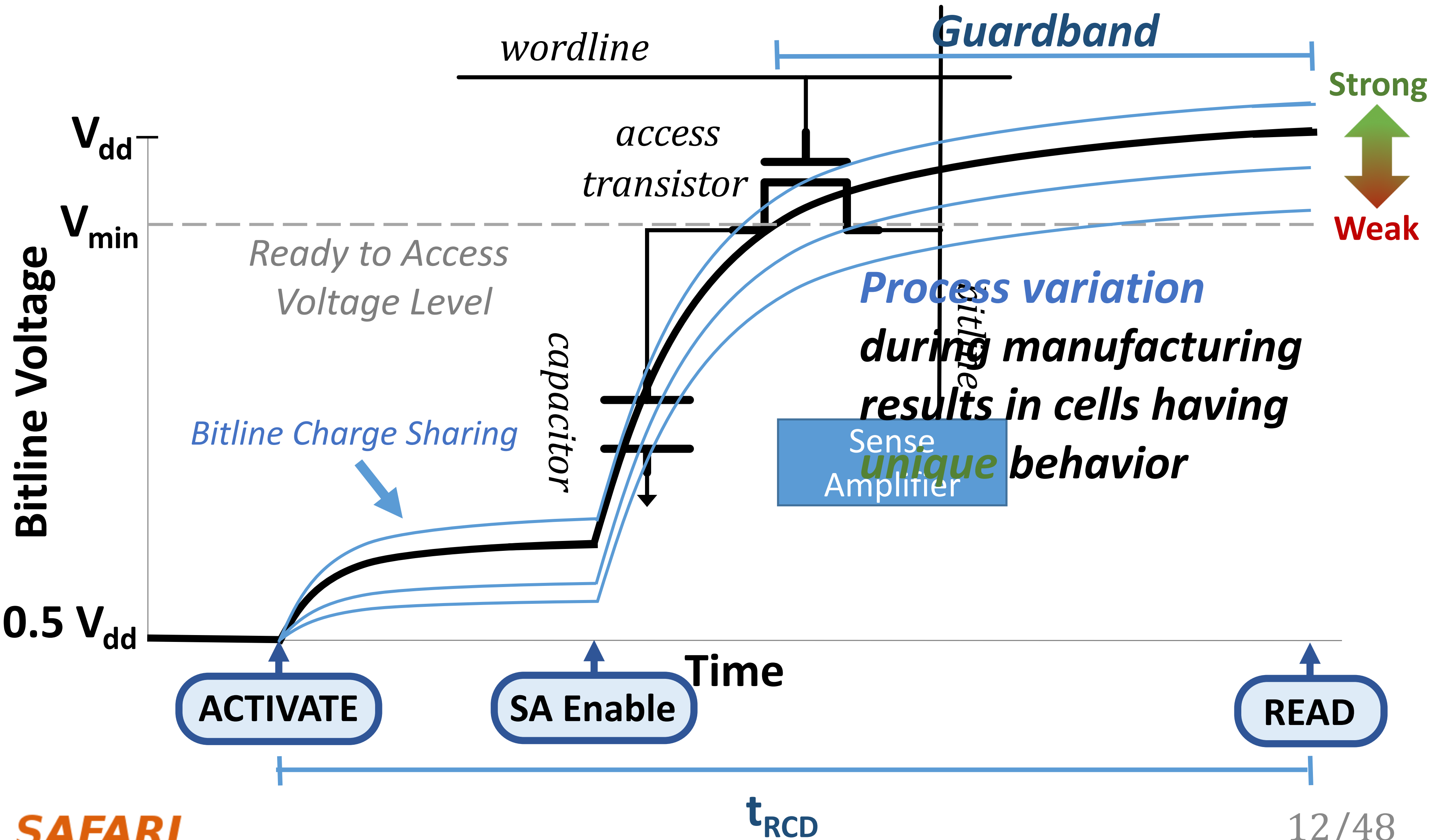
Columns of cells in subarrays share a **local bitline**

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

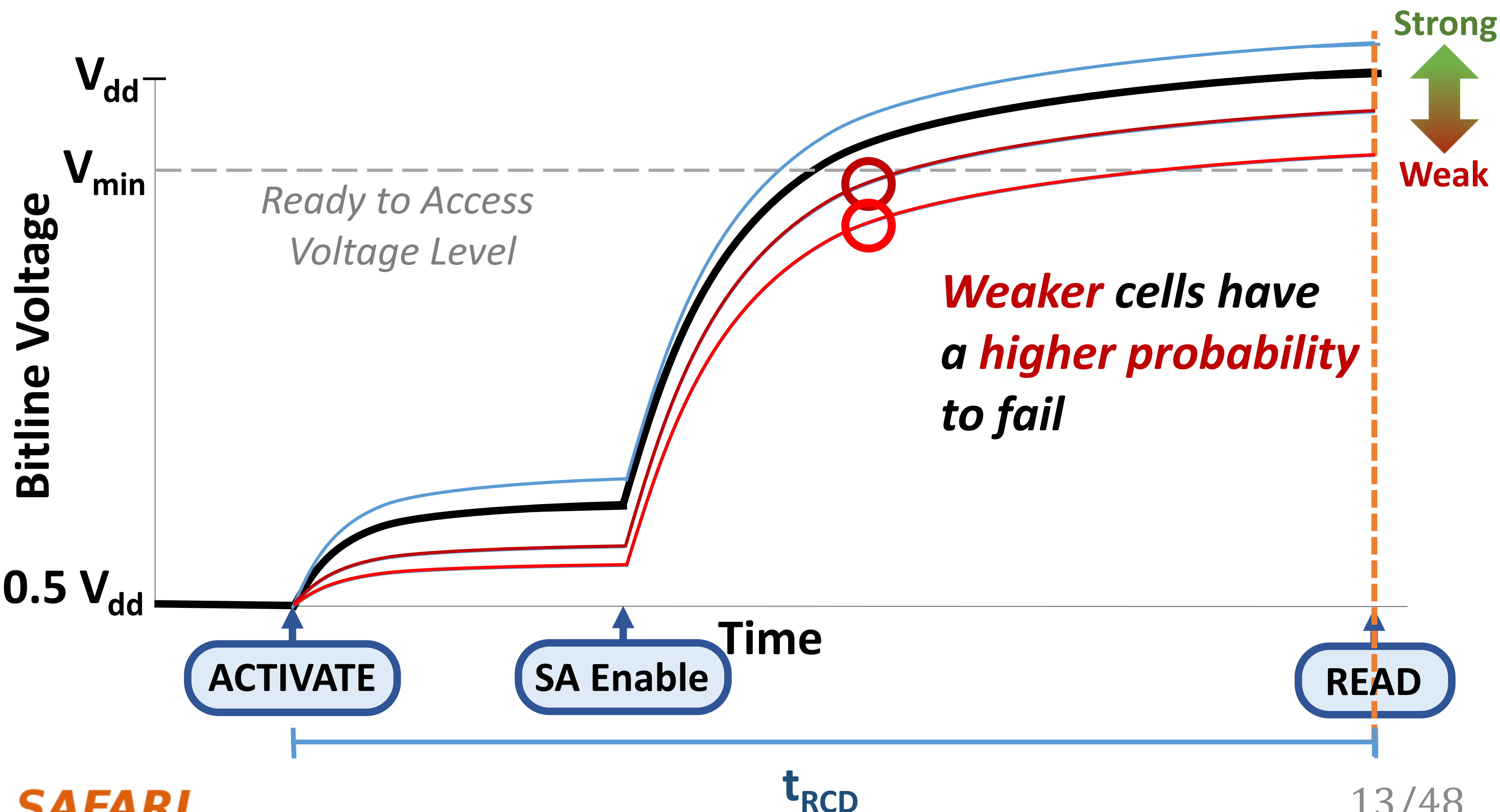
DRAM Operation



DRAM Accesses and Failures



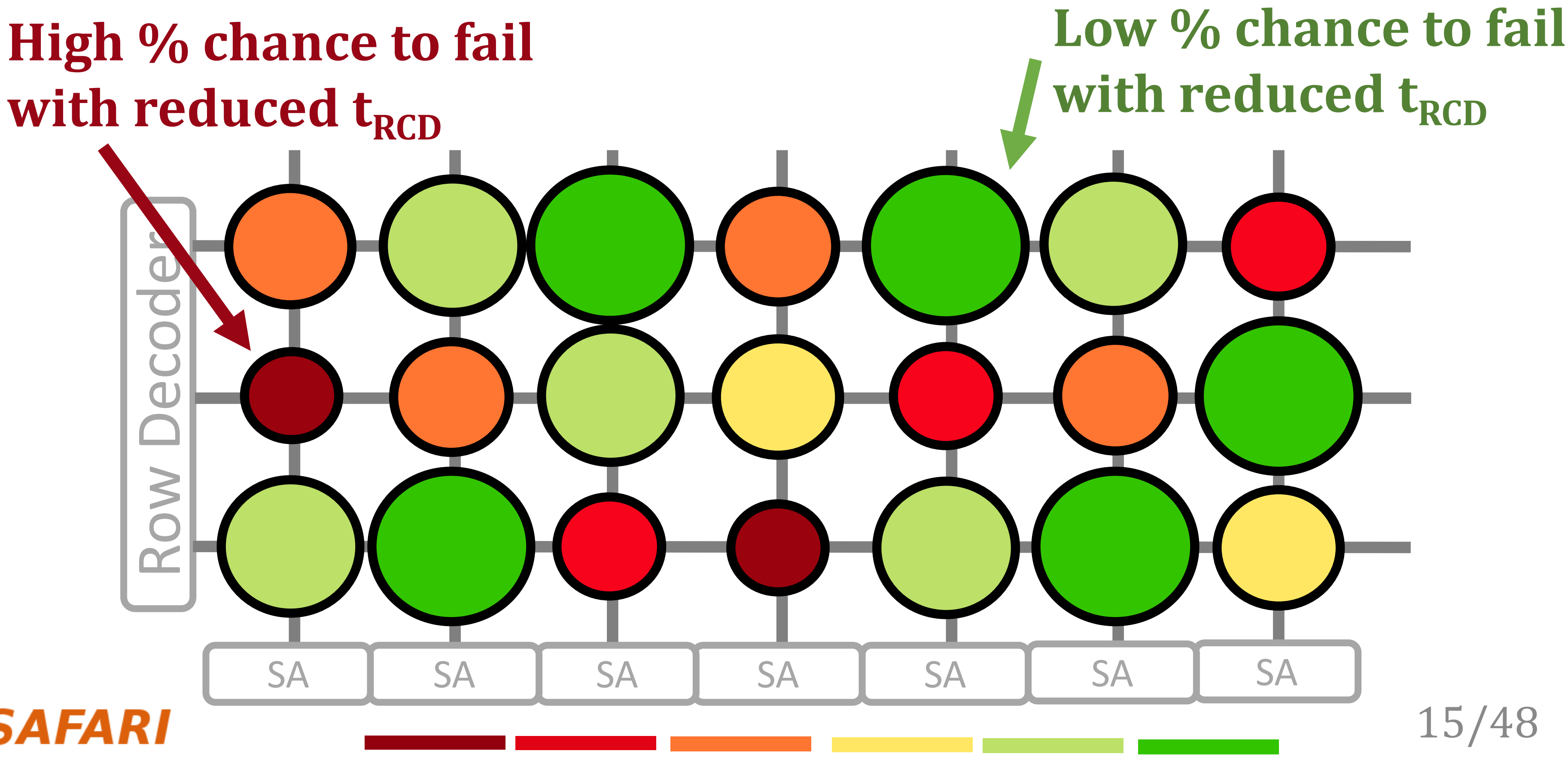
DRAM Accesses and Failures



D-RaNGe Outline

D-RaNGe Key Idea

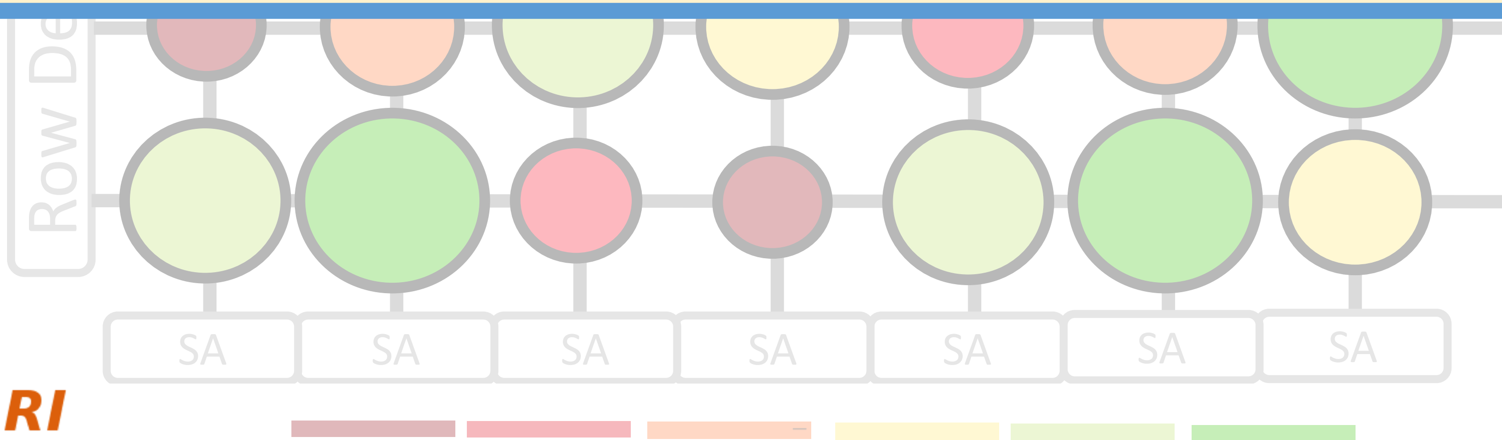
- A cell's latency failure probability is inherently related to **random process variation** from manufacturing
- We can extract **random values** by observing DRAM cells' latency failure probabilities



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

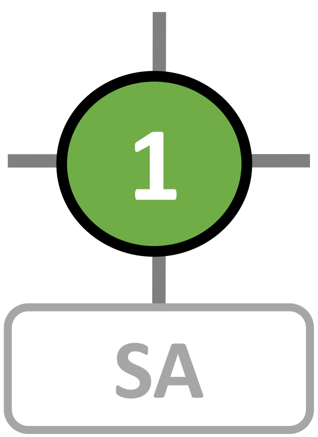


D-RaNGe: Extracting Random Values

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



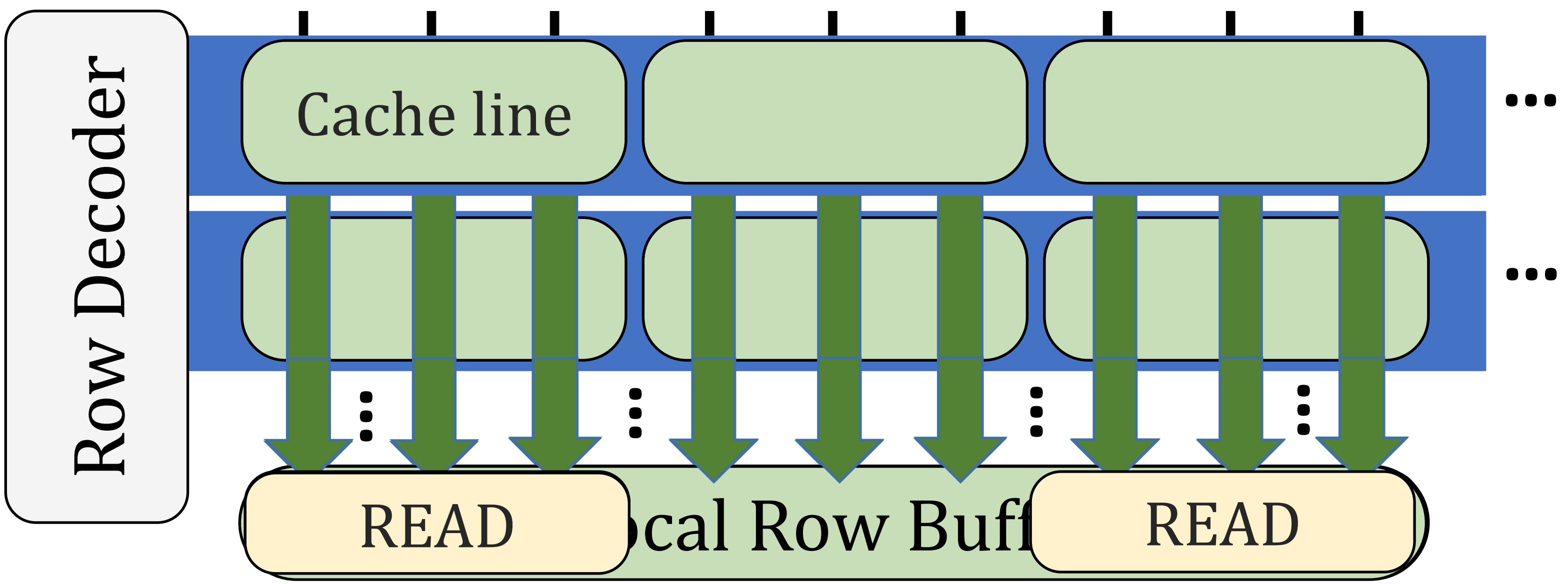
Random values when accessed with $\mathbf{t_{RCD}}$ reduced by **45%**

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

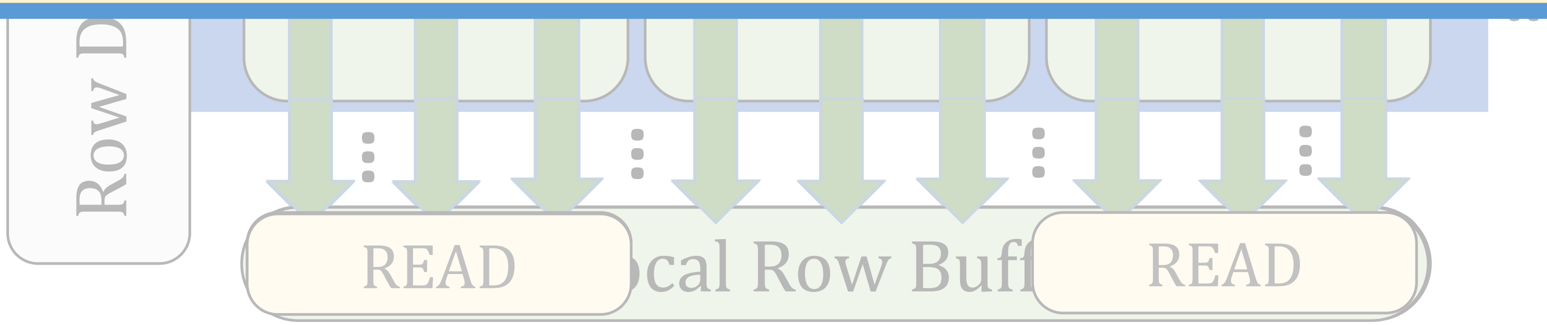
- To maximize the bits that are accessed **immediately following activation**, we alternate accesses to distinct rows **in each bank**
  - **quickly** generate tRCD failures within cache lines in two rows
  - **maximizes** tRCD failures when using reduced tRCD



D-RaNGe: Access Pattern

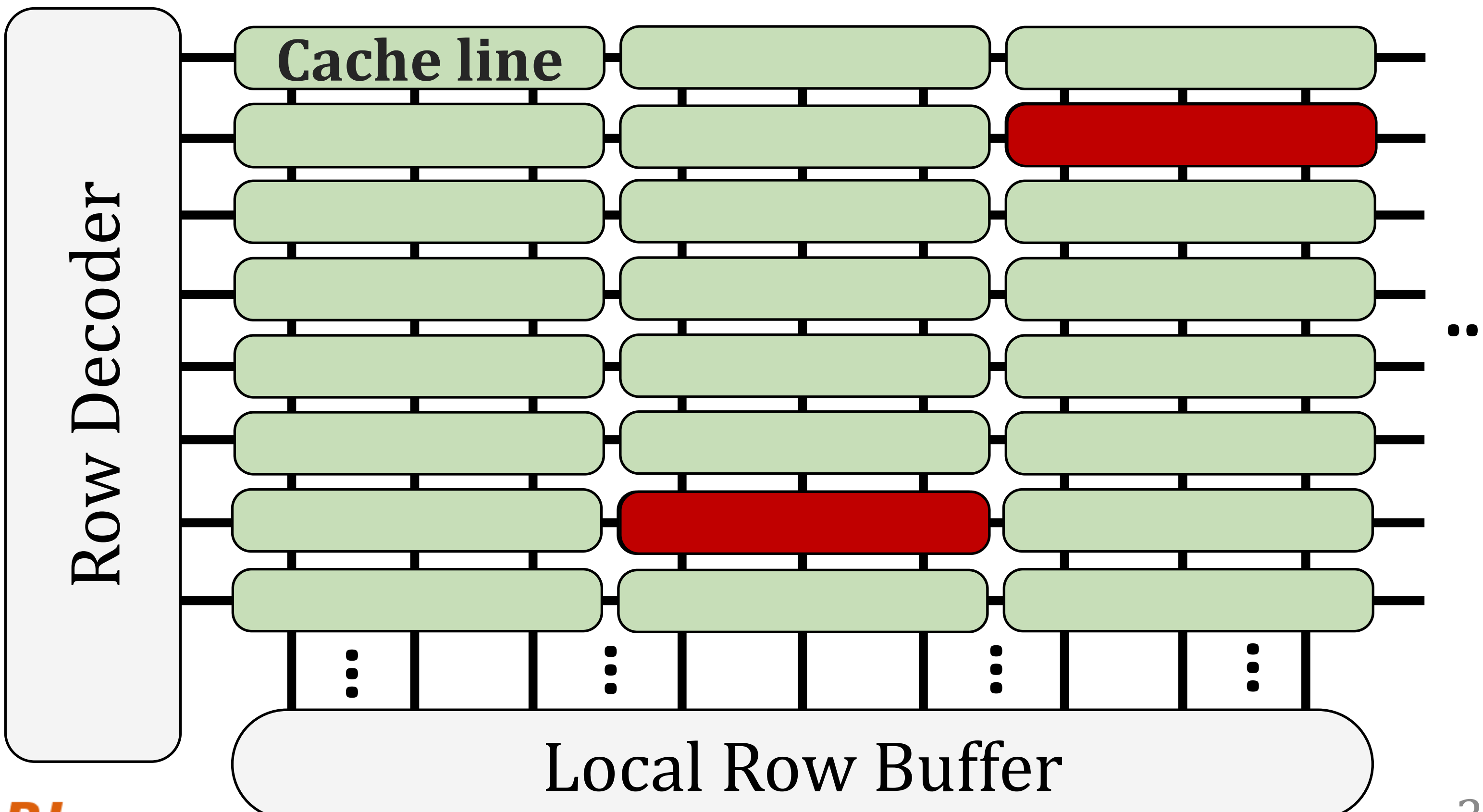
- To maximize the bits that are accessed **immediately following activation**, we alternate

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



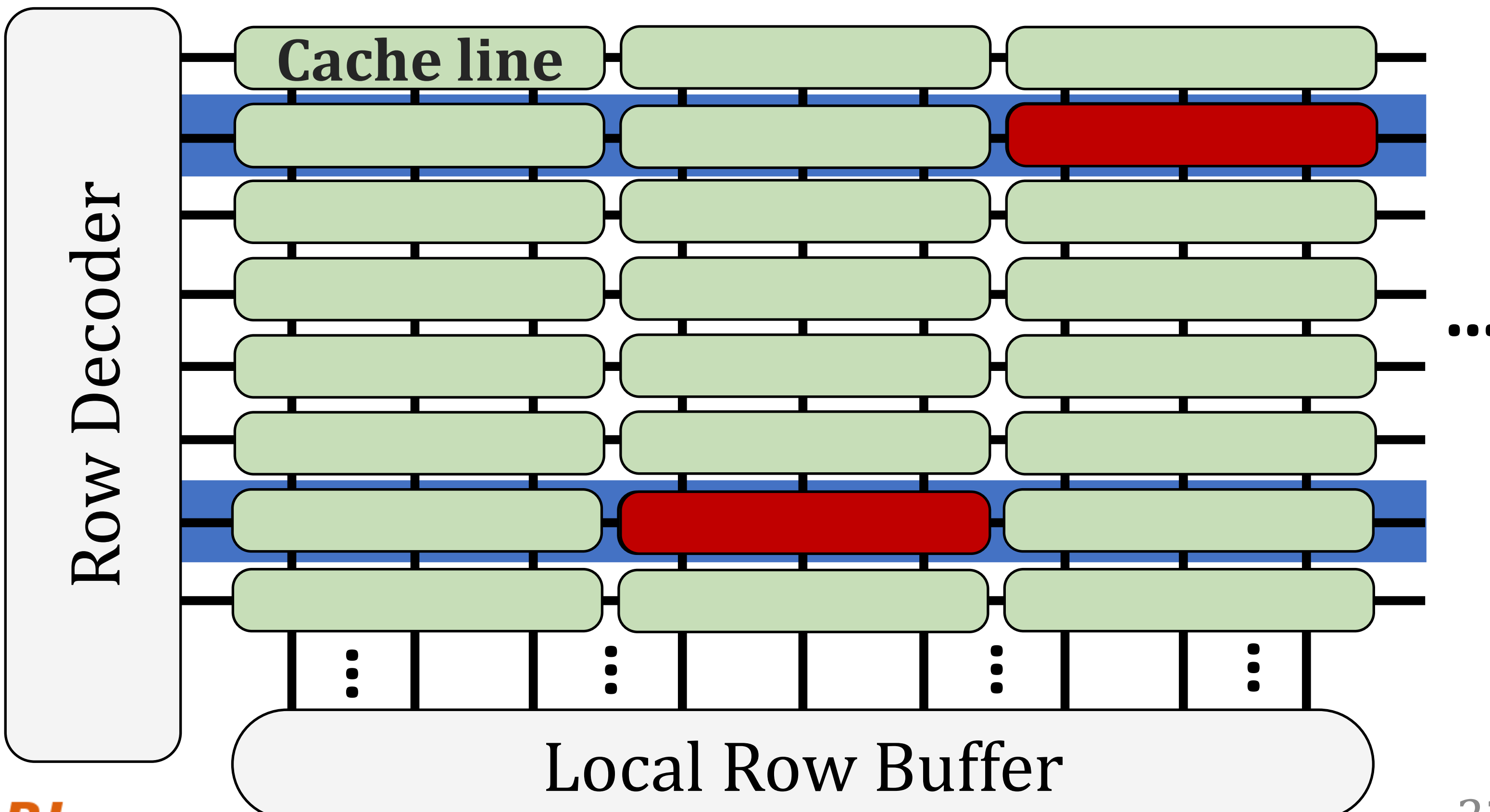
D-RaNGe: Exclusive Access

- 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

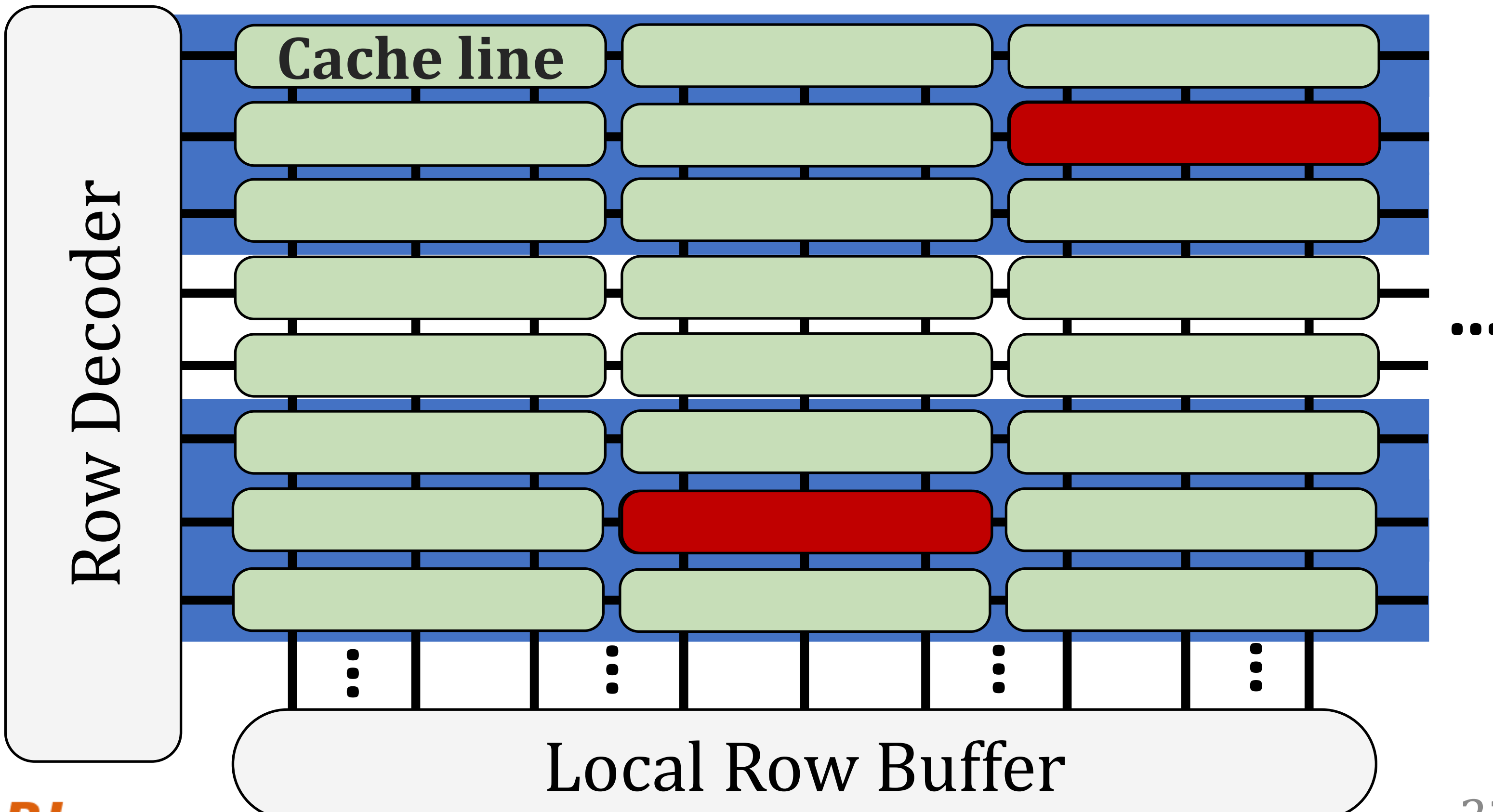
Reserve rows containing selected cache lines exclusively for D-RaNGe accesses to minimize interference



D-RaNGe: Exclusive Access

Reserve neighboring rows to minimize DRAM data pattern/read interference

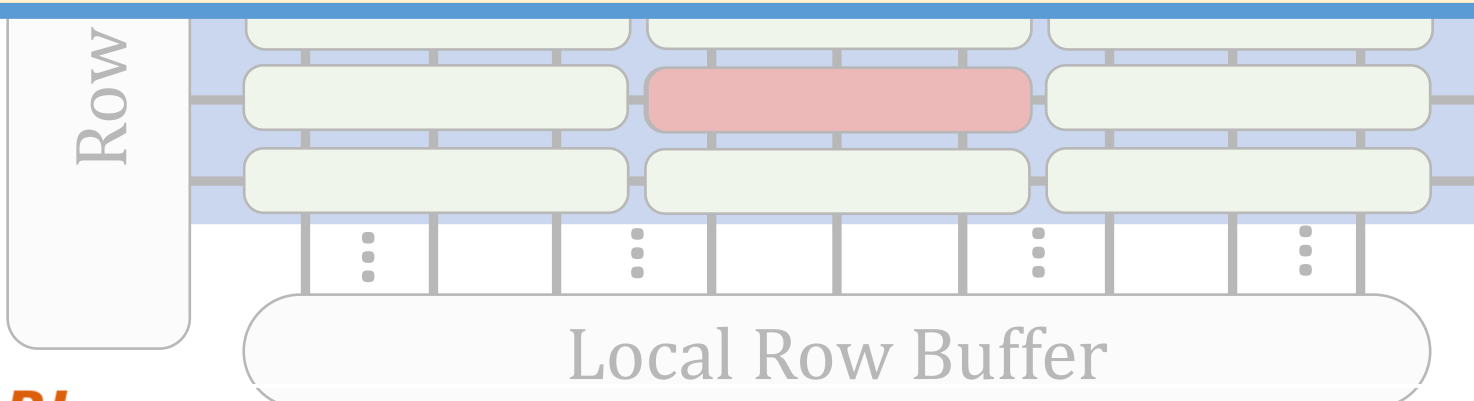
the most number of RNG cells



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

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

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

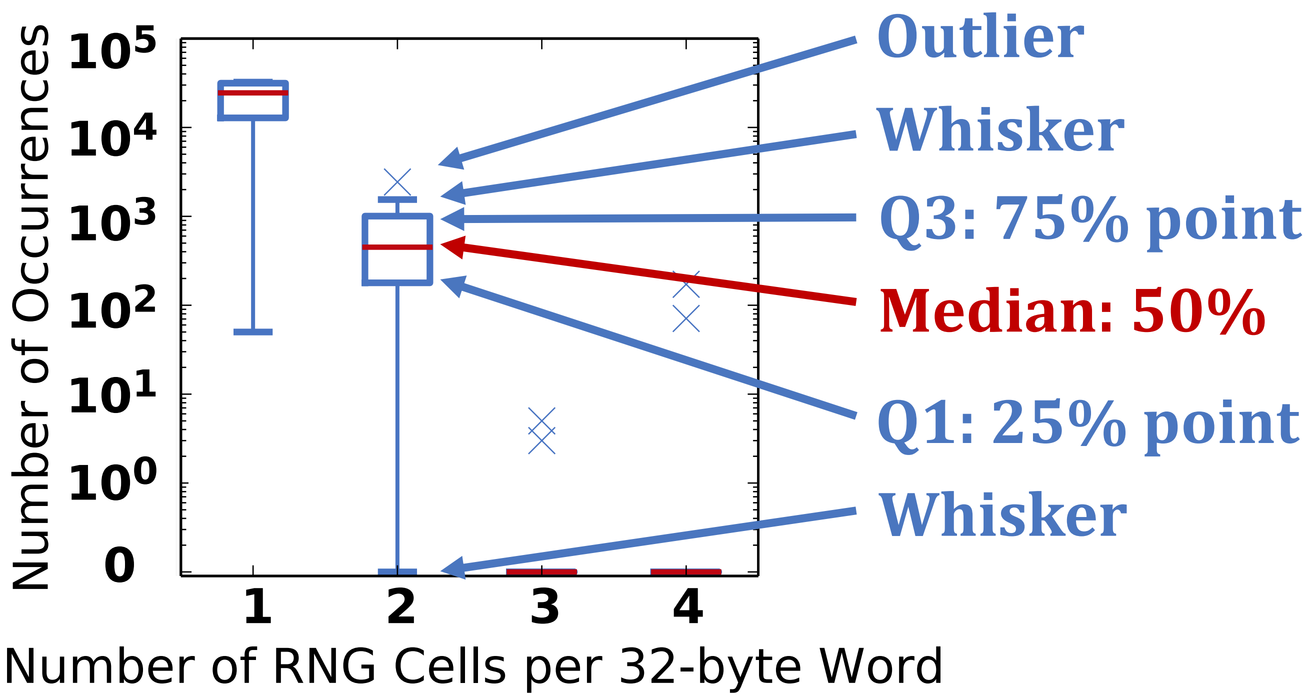
[Rukhin+, Tech report, 2001]

Passes all tests in NIST test suite for randomness!

More details in the paper

Results – 64-bit TRN Latency

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

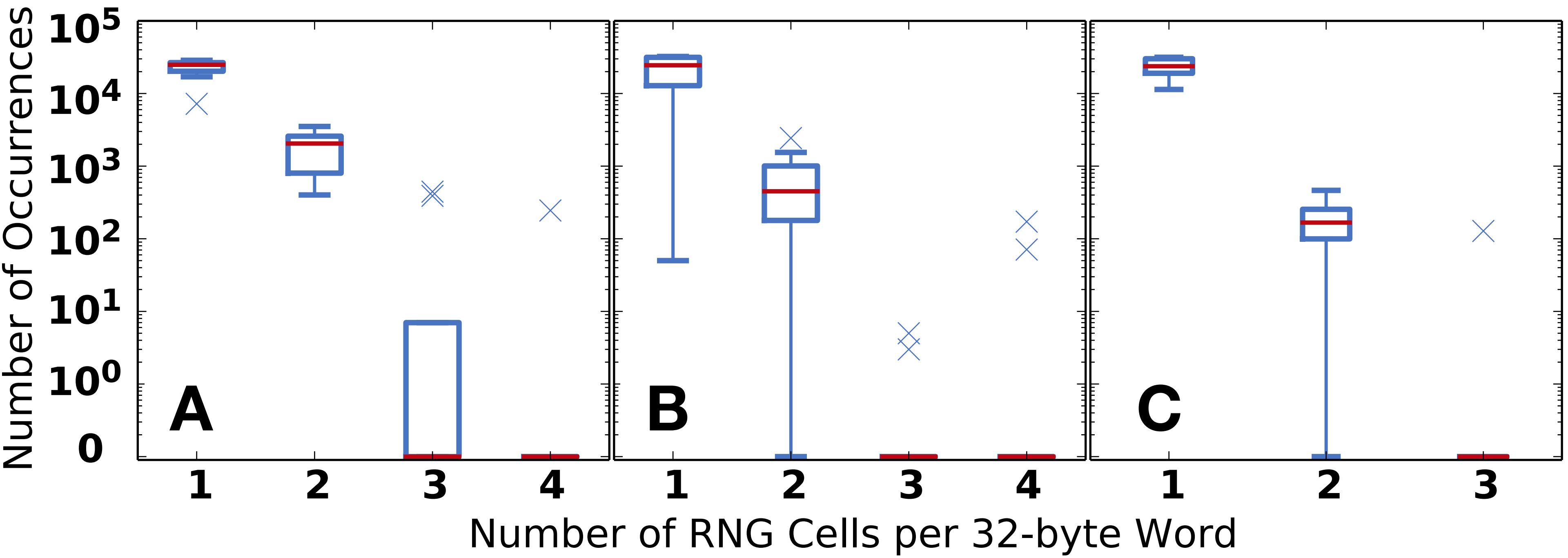


Across our devices, we analyze **availability of RNG cells** per cache line in a bank. Each point is the number of occurrences in a bank.

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



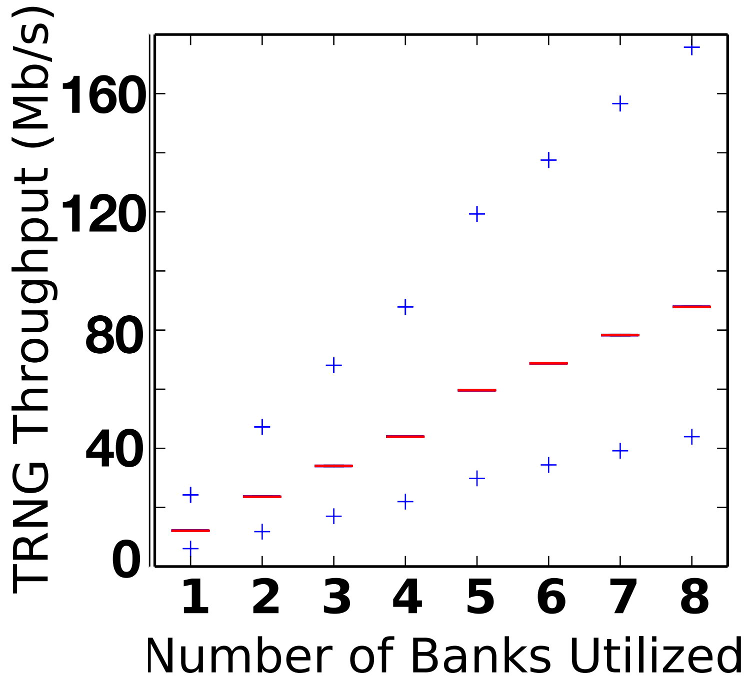
Maximum latency: **960 ns**

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

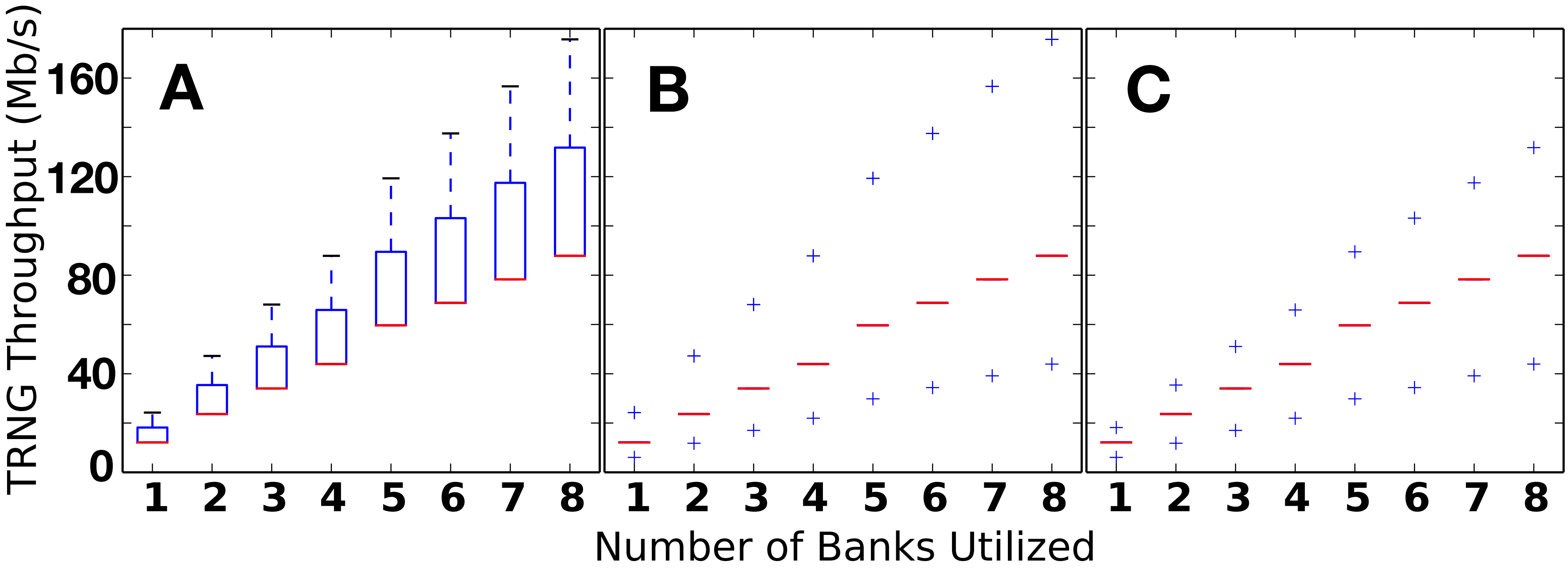


We determine **throughput** using the RNG cell densities found

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

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)**

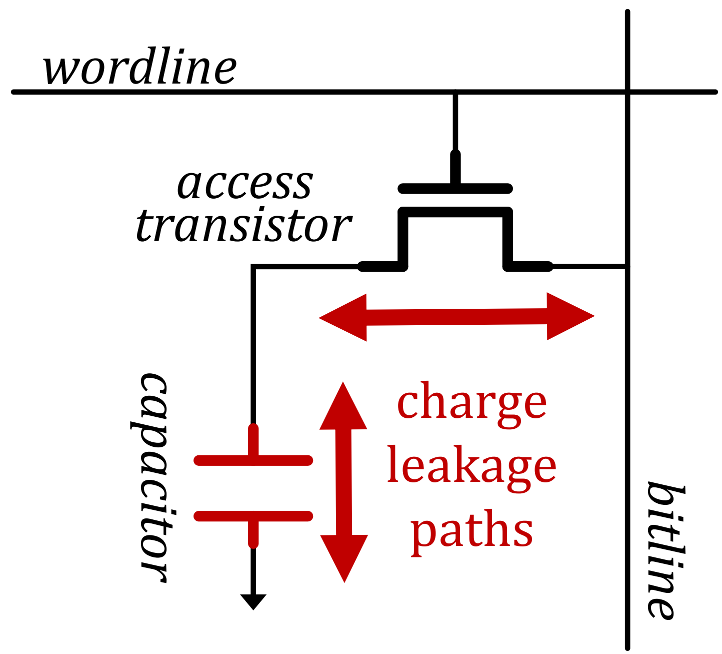
- **Downsides of DRAM Command Scheduling based TRNGs**
  - Randomness source is **not truly random**: depends on memory controller implementation and concurrently running applications
  - Much lower TRN throughput than **D-RaNGe**

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

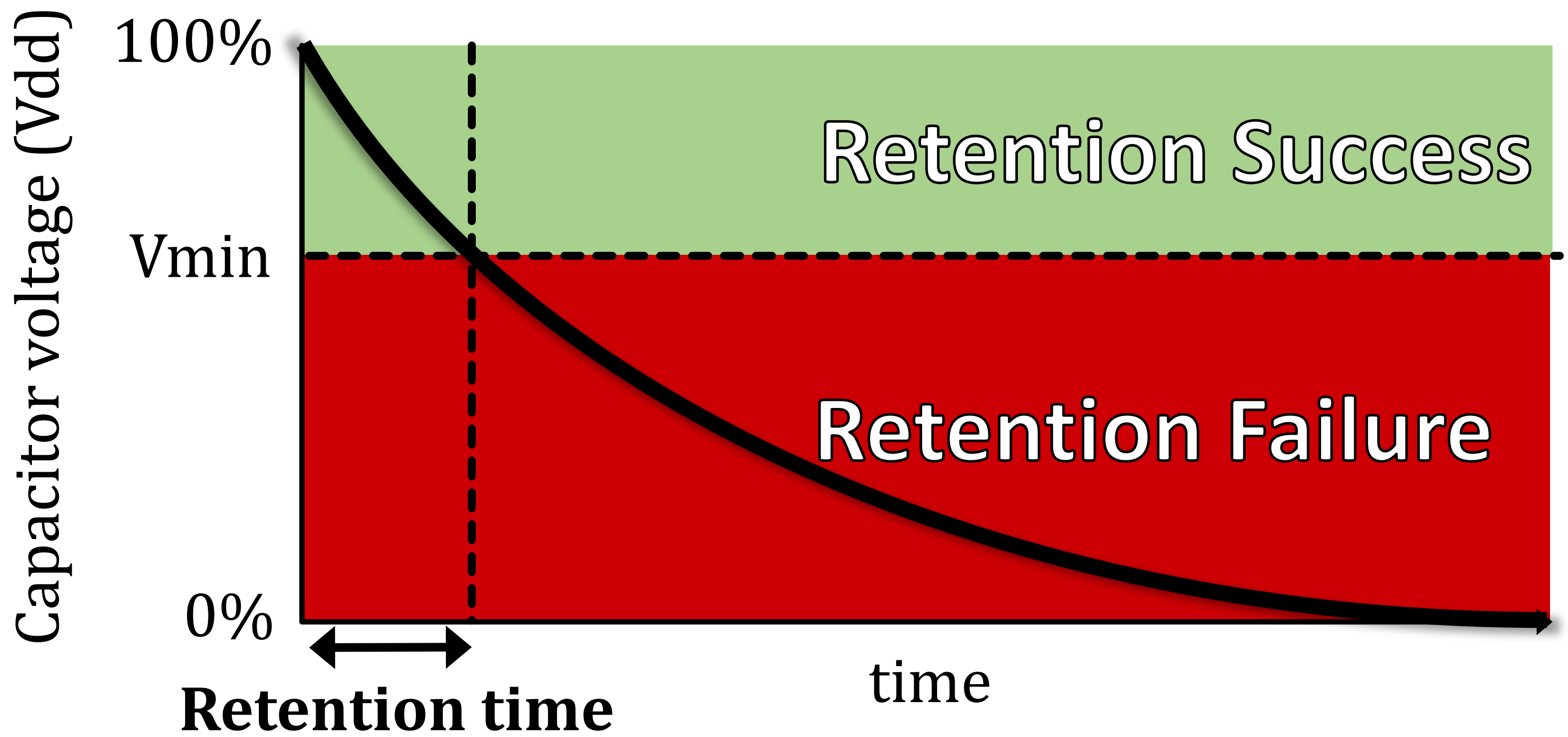
DRAM encodes information in **leaky** capacitors



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



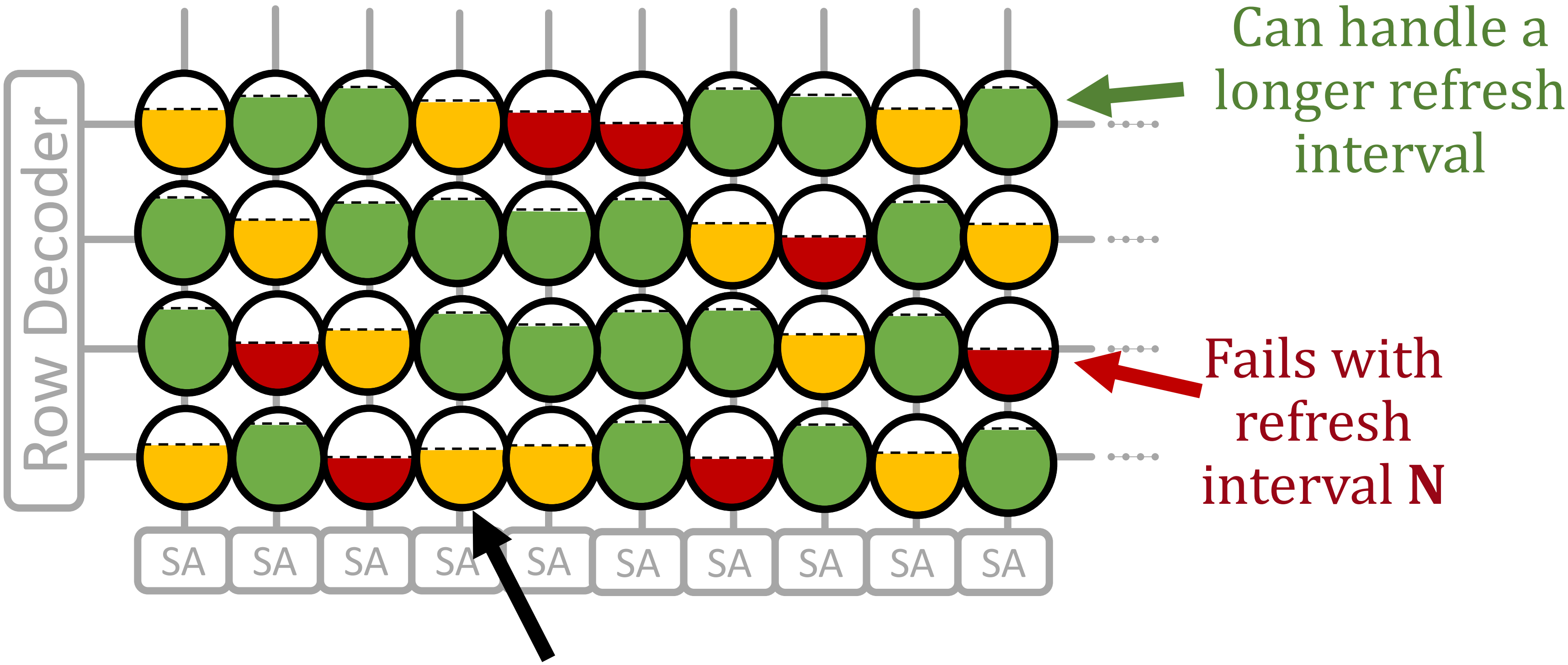
Retention failure – when leakage corrupts stored data

Retention time – how long a cell holds its value

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.

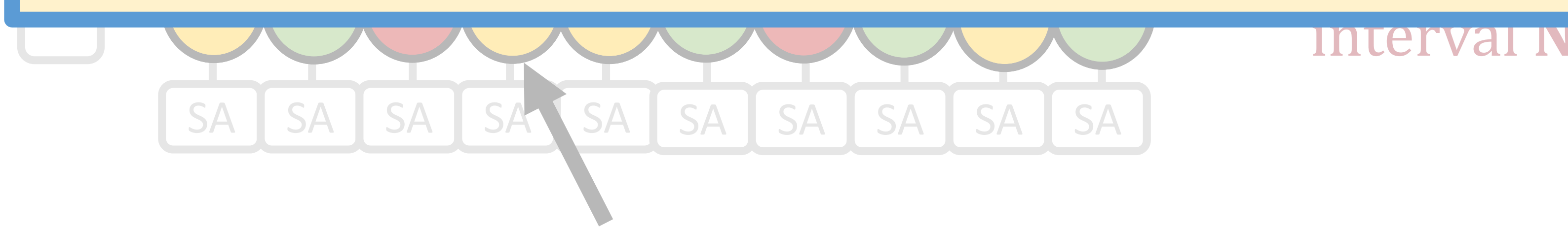
These RNG cells fail randomly

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

SAFARI

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

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

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**)

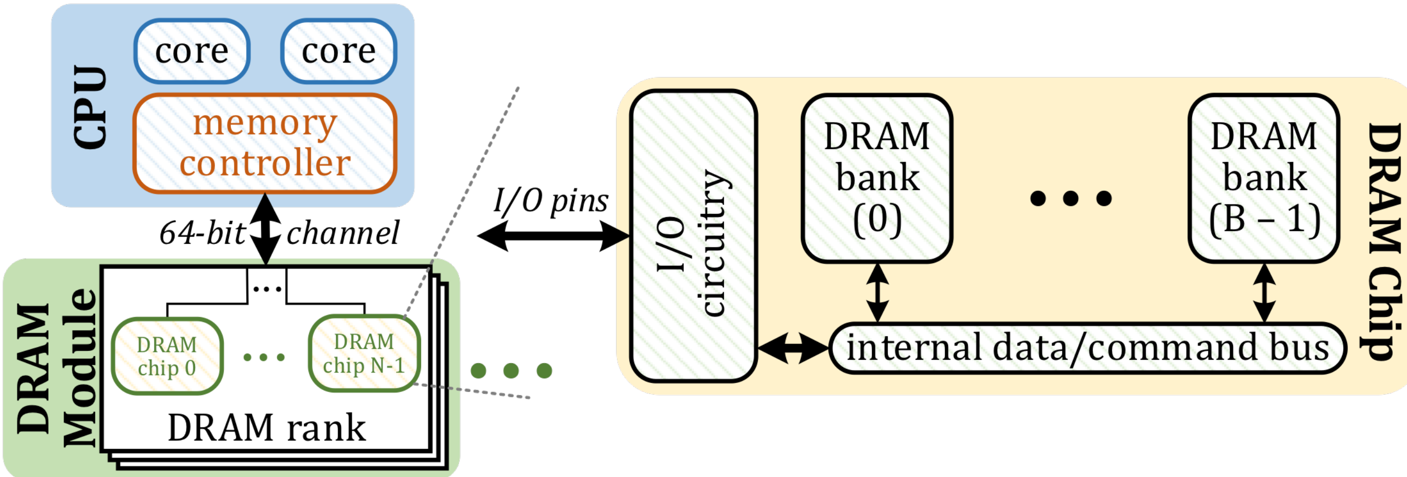
D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput

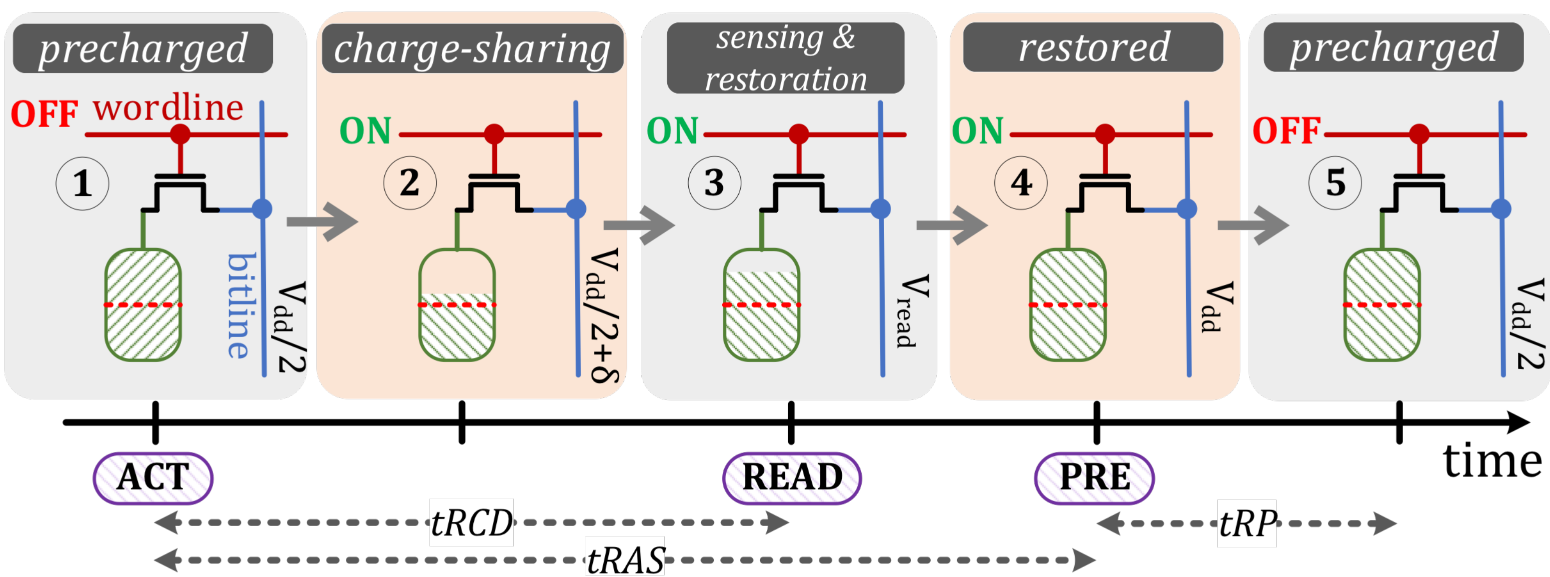
Jeremie S. Kim **Minesh Patel**
Hasan Hassan **Lois Orosa** **Onur Mutlu**

SAFARI

Carnegie Mellon ETH Zürich

DRAM Organization + Operation





DRAM Activation Failure Testing

Algorithm 1: DRAM Activation Failure Testing

```
DRAM_ACT_failure_testing(data_pattern, DRAM_region):
    write data_pattern (e.g., solid 1s) into all cells in DRAM_region
    set low tRCD for ranks containing DRAM_region
    foreach col in DRAM_region:
        foreach row in DRAM_region:
            activate(row)     // fully refresh cells
            precharge(row)    // ensure next access activates the row
            activate(row)
            read(col)         // induce activation failure on col
            precharge(row)
            record activation failures to storage
    set default tRCD for DRAM ranks containing DRAM_region
```

SAFARI

Activation Failure Spatial Distribution

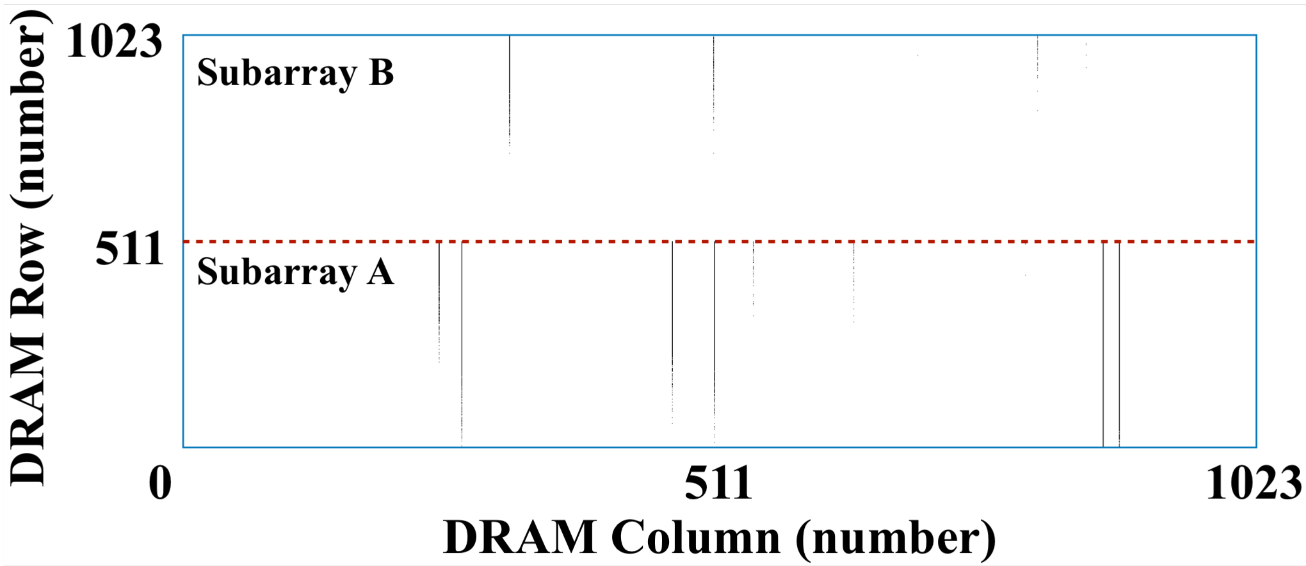


Figure 4: Activation failure bitmap in 1024×1024 **cell array.**

Activation Failure Temperature Dependence

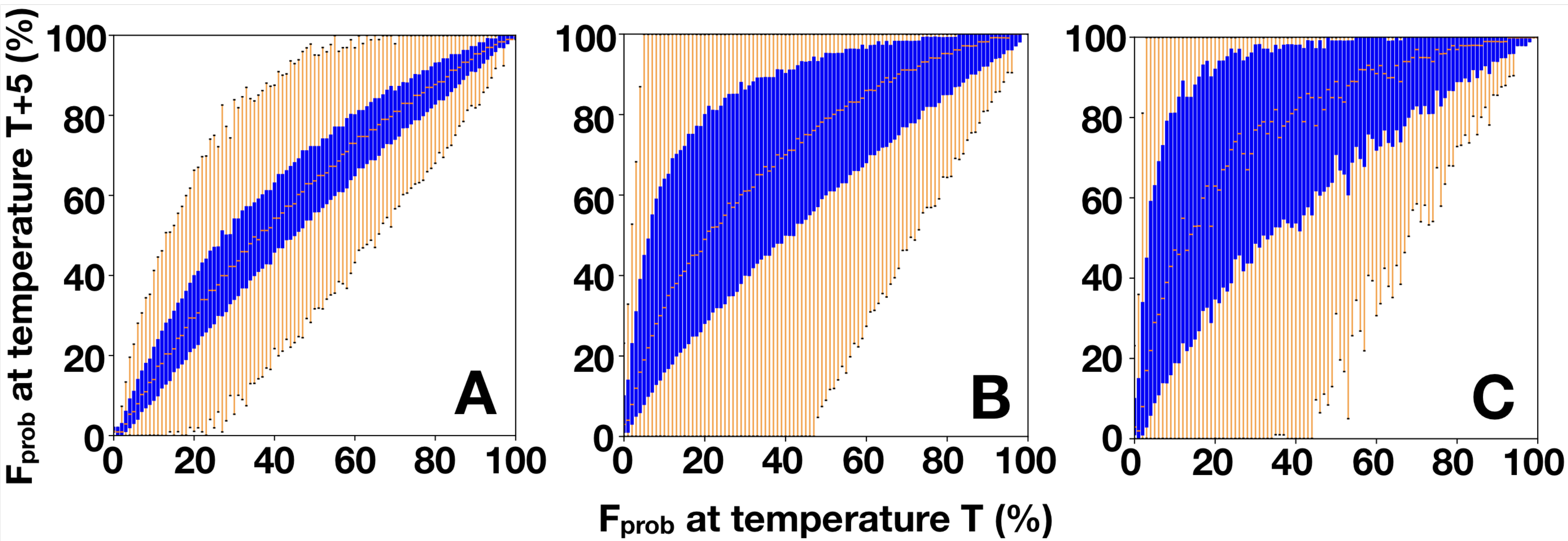


Figure 6: Effect of temperature variation on failure probability

Full D-RaNGe Algorithm

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

D-RaNGe(*num_bits*): // *num_bits*: number of random bits requested
    DP: a known data pattern that results in high entropy
    select 2 DRAM words with RNG cells in distinct rows in each bank
    write *DP* to chosen DRAM words and their neighboring cells
    get exclusive access to rows of chosen DRAM words and nearby cells
    set low t_{RCD} for DRAM ranks containing chosen DRAM words
    for each bank:
        read data in DW_1 // induce activation failure
        write the read value of DW_1's RNG cells to *bitstream*
        write original data value back into DW_1
        memory barrier // ensure completion of write to DW_1
        read data in DW_2 // induce activation failure
        write the read value of DW_2's RNG cells to *bitstream*
        write original data value back into DW_2
        memory barrier // ensure completion of write to DW_2
        if $bitstream_{size} \geq num\_bits$:
            break
    set default t_{RCD} for DRAM ranks of the chosen DRAM words
    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	✓	✓	$40s$	$6.8mJ/bit$	$0.05Mb/s$
Tehranipoor+ [144]	2016	Startup Values	✓	✗	$> 60ns$ (optimistic)	$> 245.9pJ/bit$ (optimistic)	N/A
Sutar+ [141]	2018	Data Retention	✓	✓	$40s$	$6.8mJ/bit$	$0.05Mb/s$
D-RaNGe	2018	Activation Failures	✓	✓	$100ns < x < 960ns$	$4.4nJ/bit$	$717.4Mb/s$

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

DRAM Data Pattern Dependence

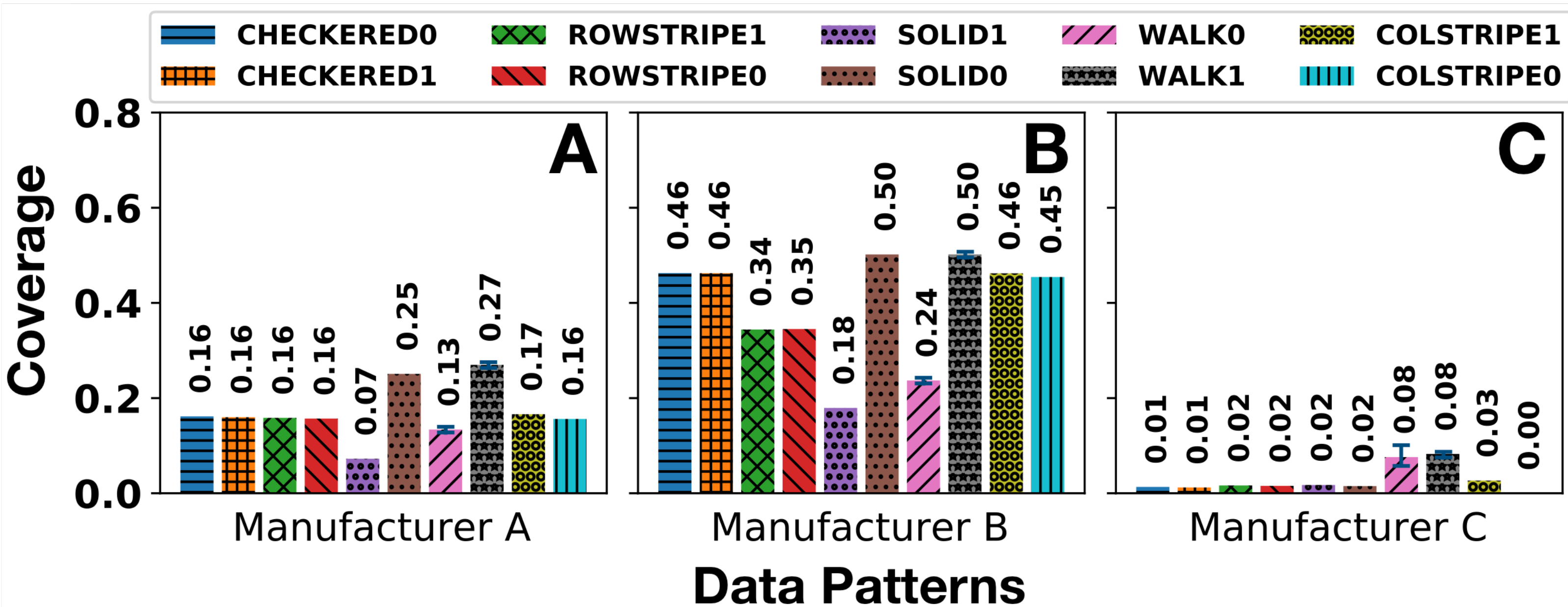
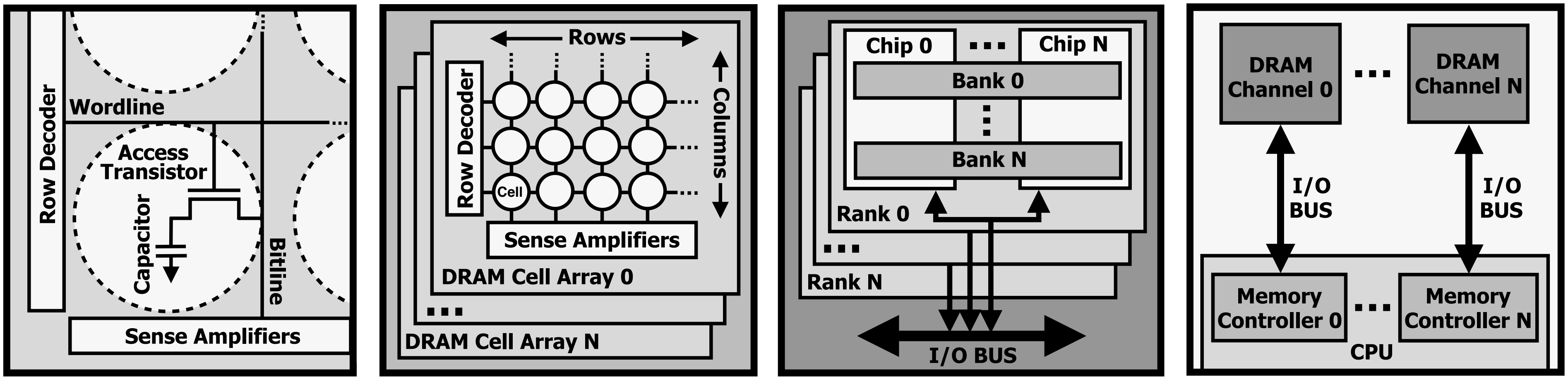


Figure 5: Data pattern dependence of DRAM cells prone to activation failure over 100 iterations

DRAM Architecture Background



(a) DRAM Cell Array (b) DRAM Bank (c) DRAM Channel (d) DRAM-Based System

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

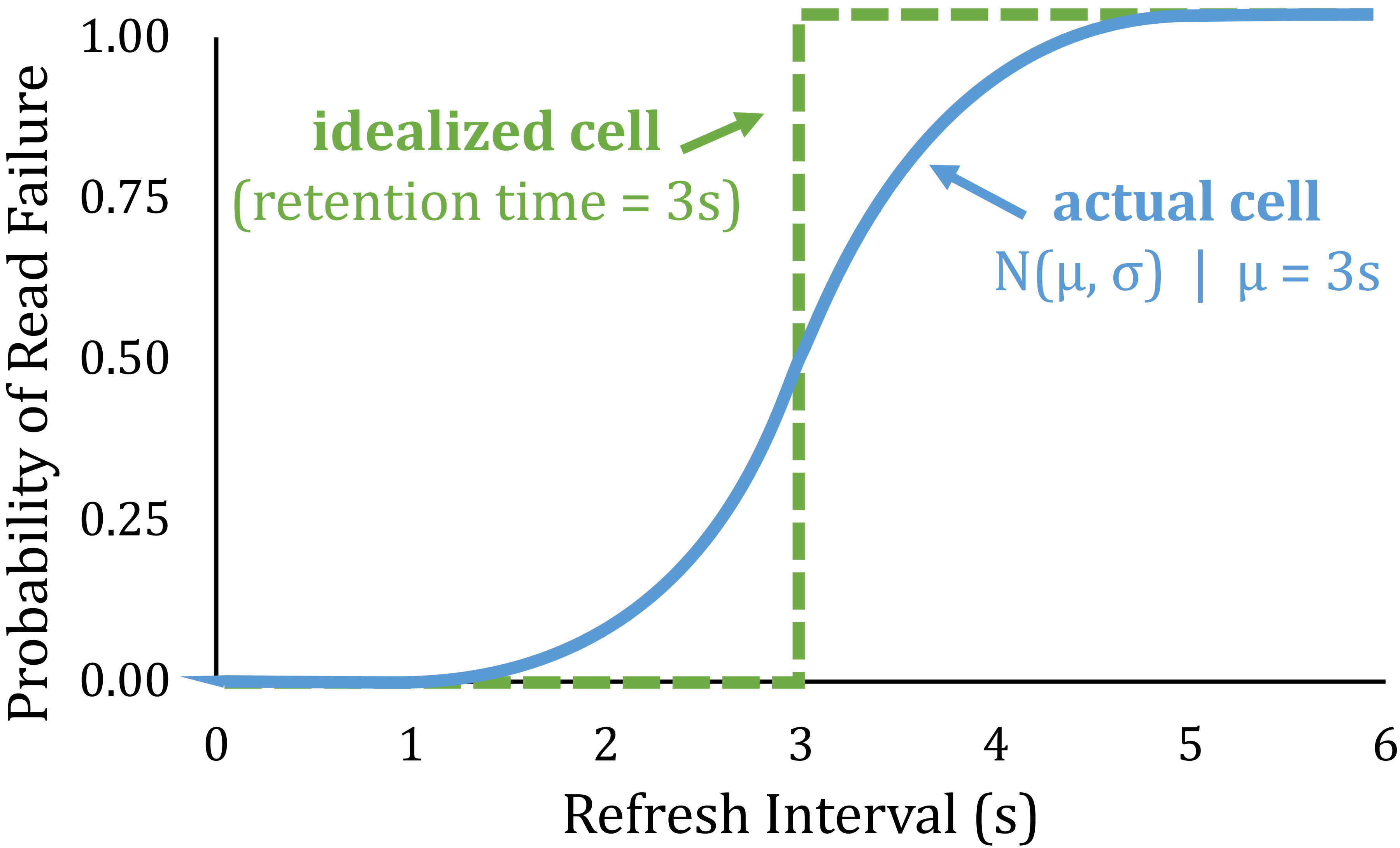
- **Variable retention time (VRT)**
  - Retention time changes **randomly (unpredictably)**
  - Due to a combination of various circuit effects

Long-term Continuous Profiling

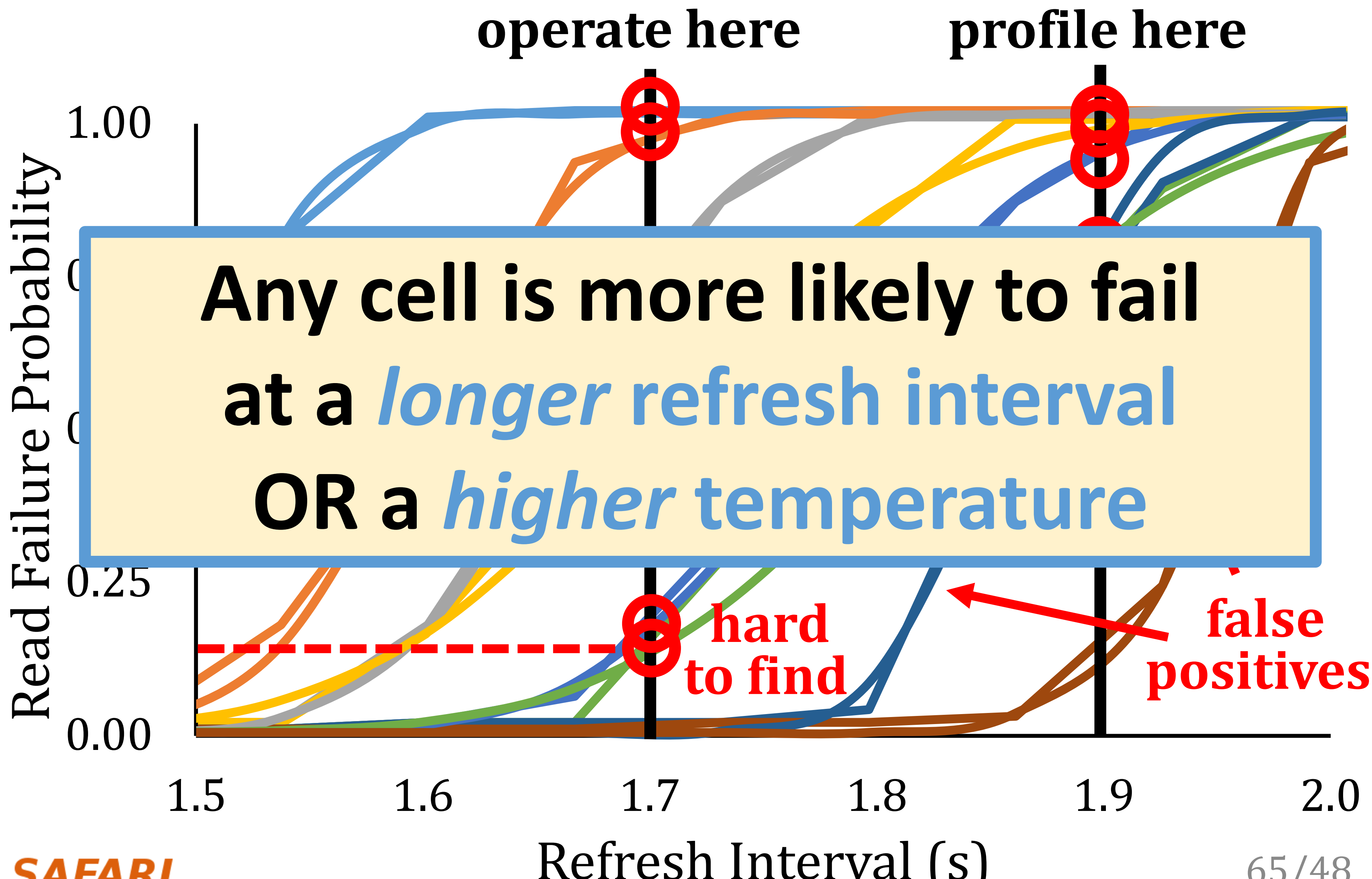
Error correction codes (ECC) and online profiling are *necessary* to manage new failing cells

- 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)



Any cell is more likely to fail at a *longer* refresh interval OR a *higher* temperature

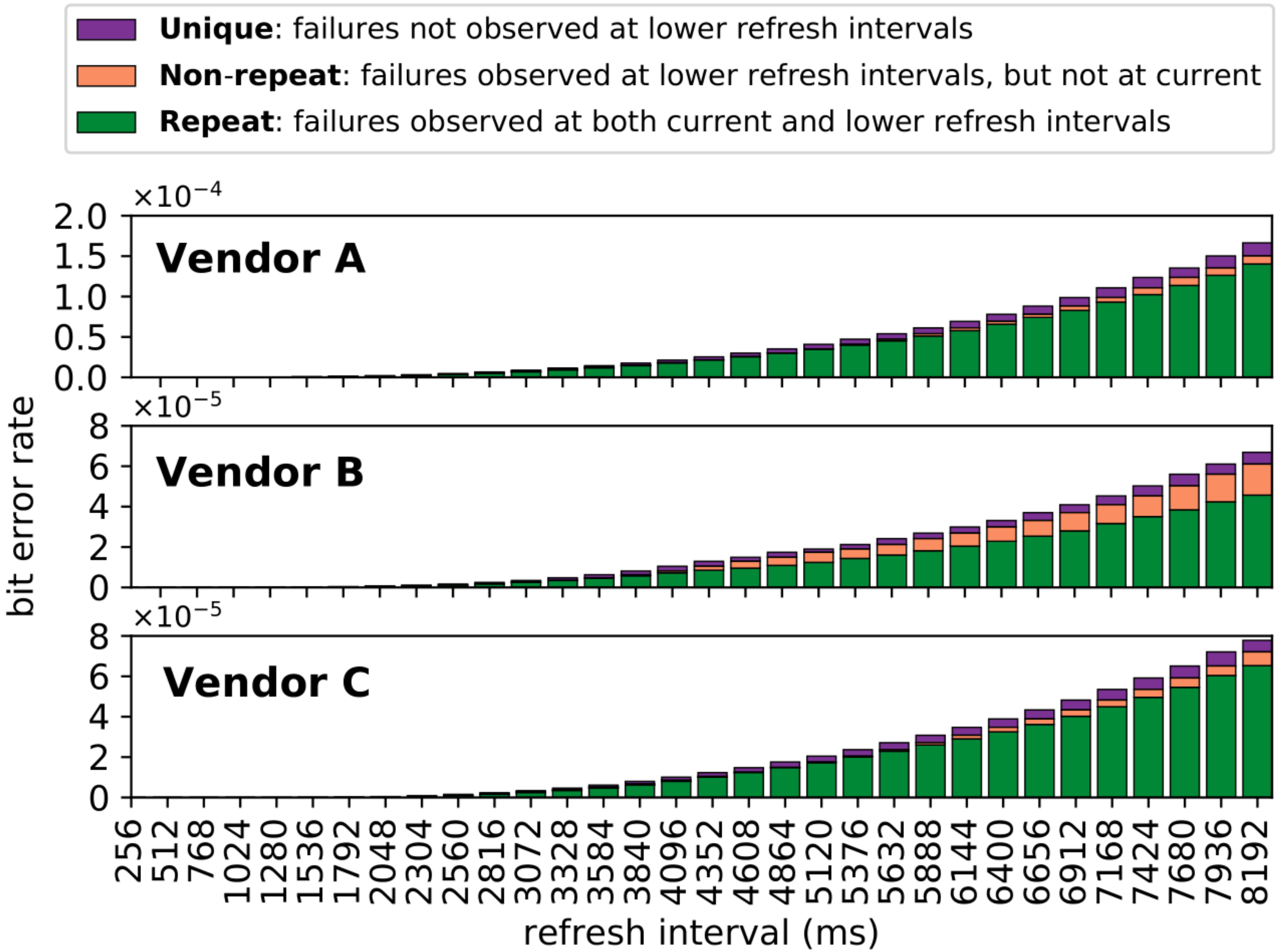
Temperature Relationship

- Well-fitting exponential relationship:

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

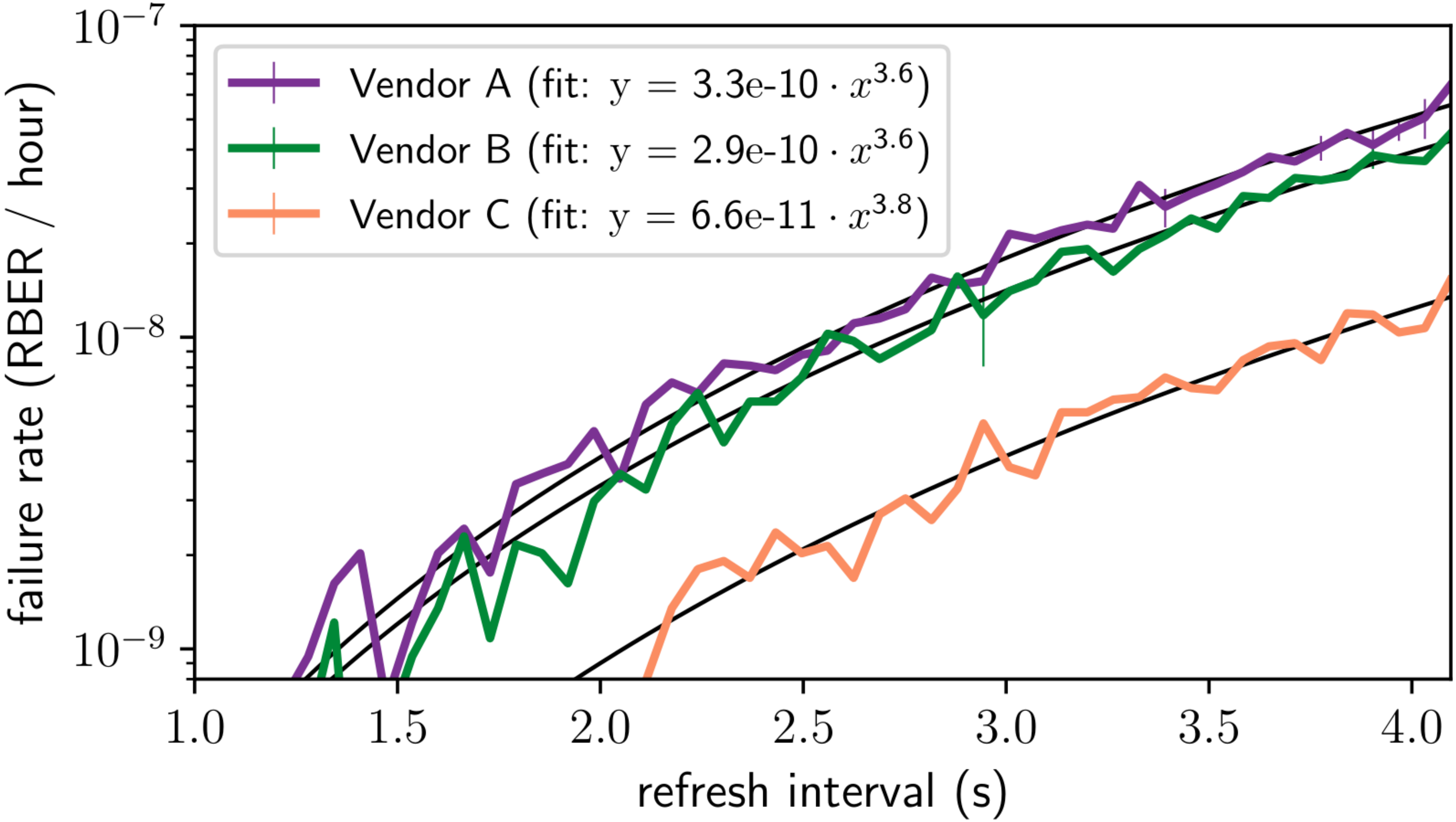
- E.g., 10°C ~ 10x more failures

Retention Failures @ 45°C

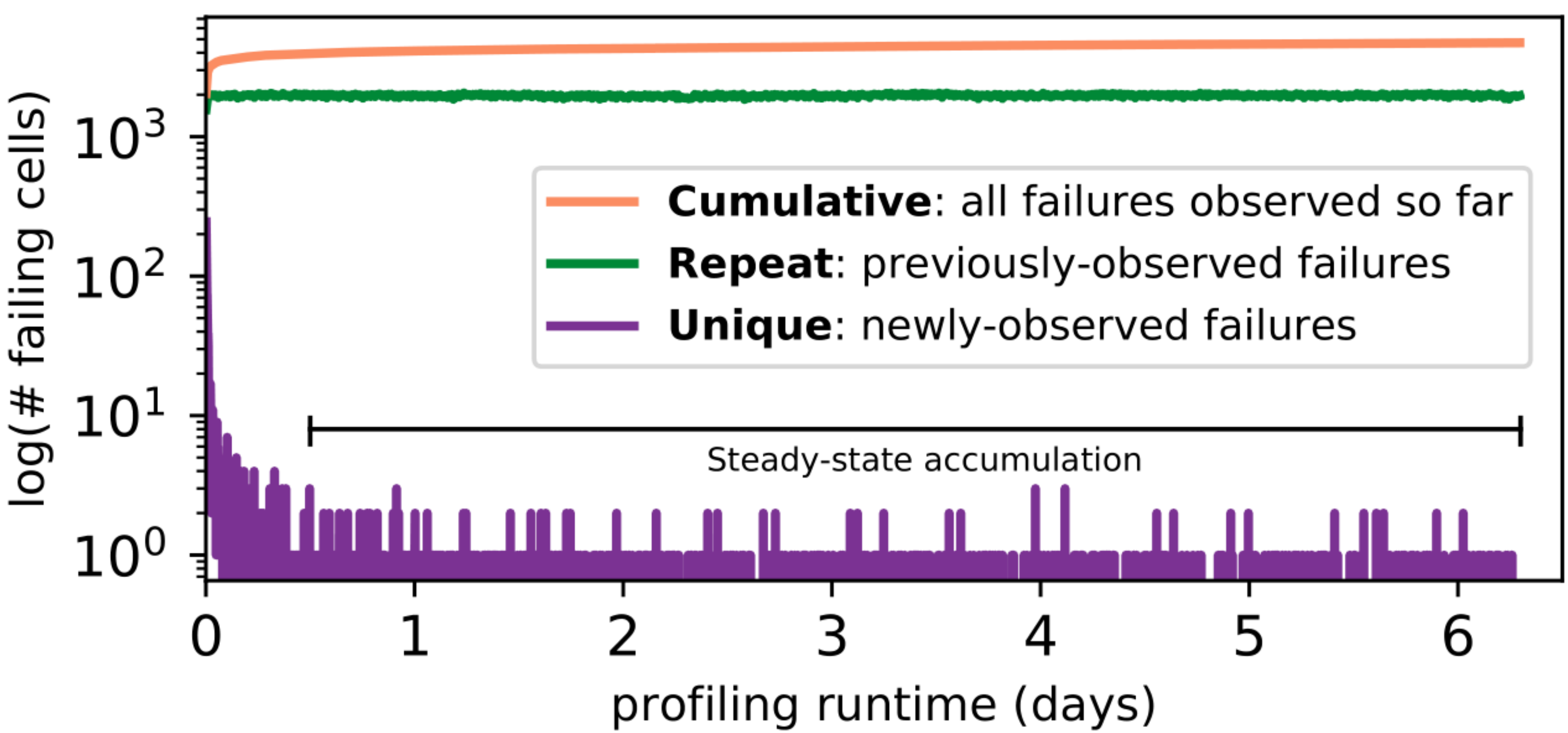


SAFARI

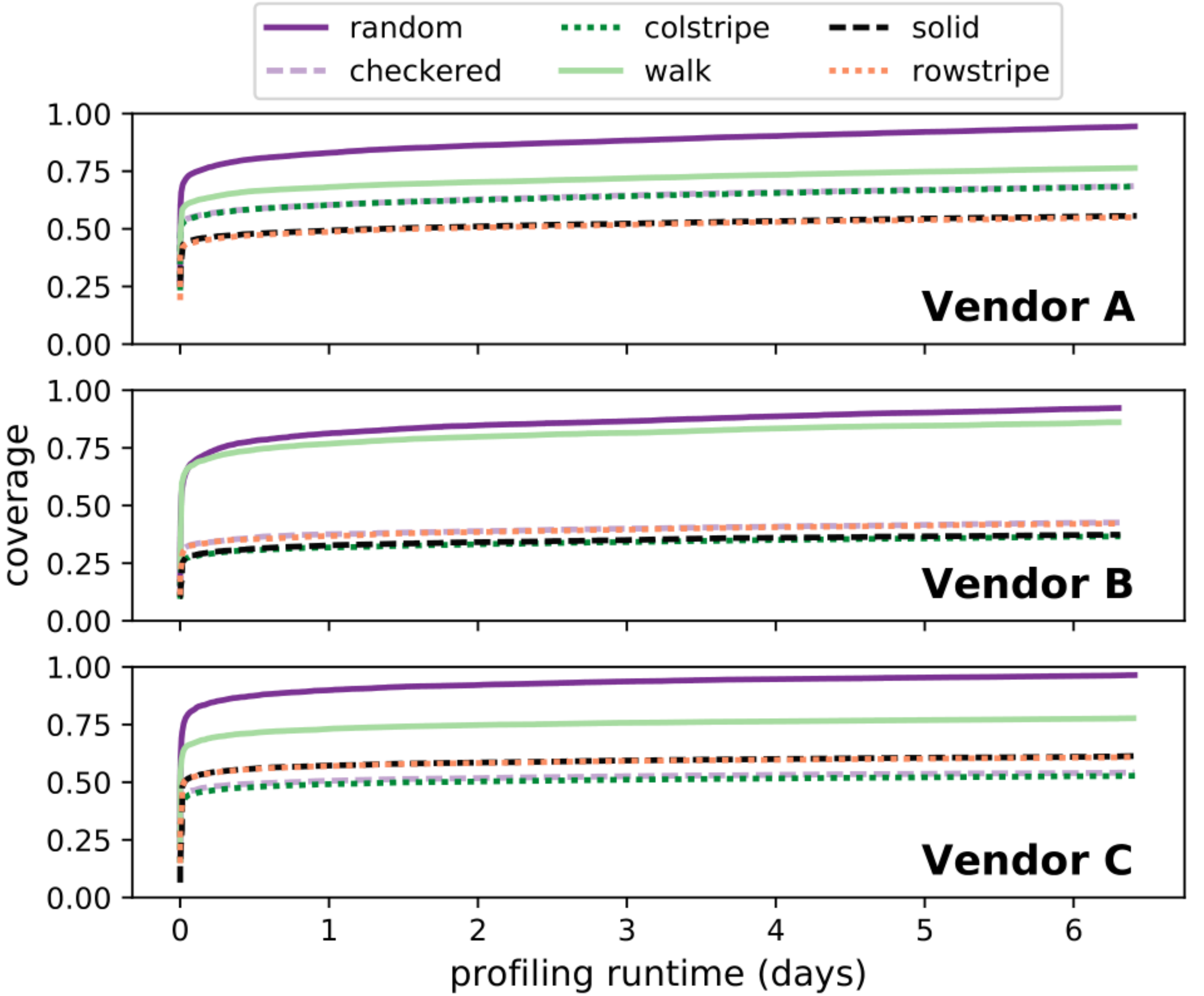
VRT Failure Accumulation Rate



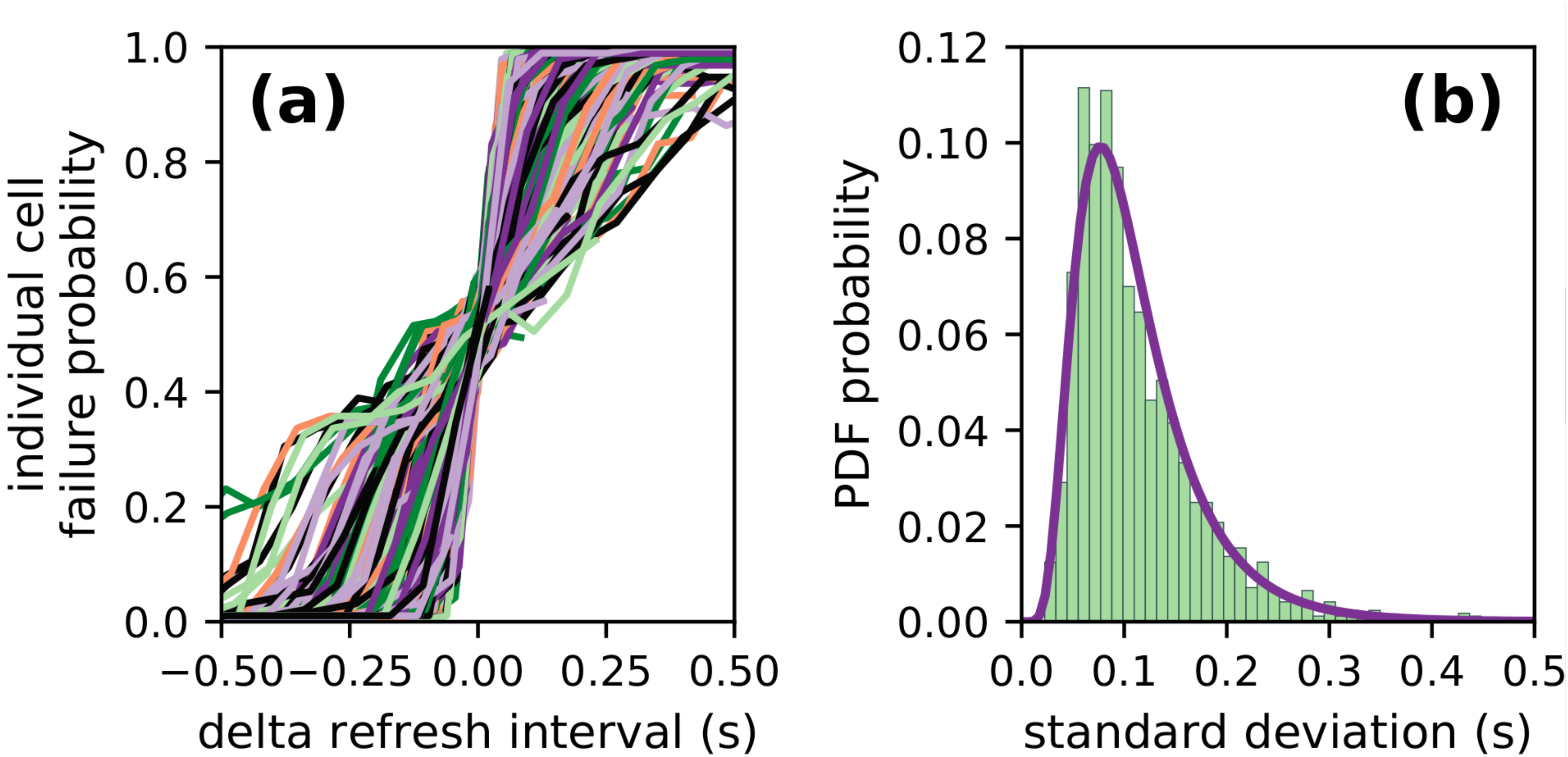
800 Rounds of Profiling @ 2048ms, 45°C



800 Rounds of Profiling @ 2048ms, 45°C

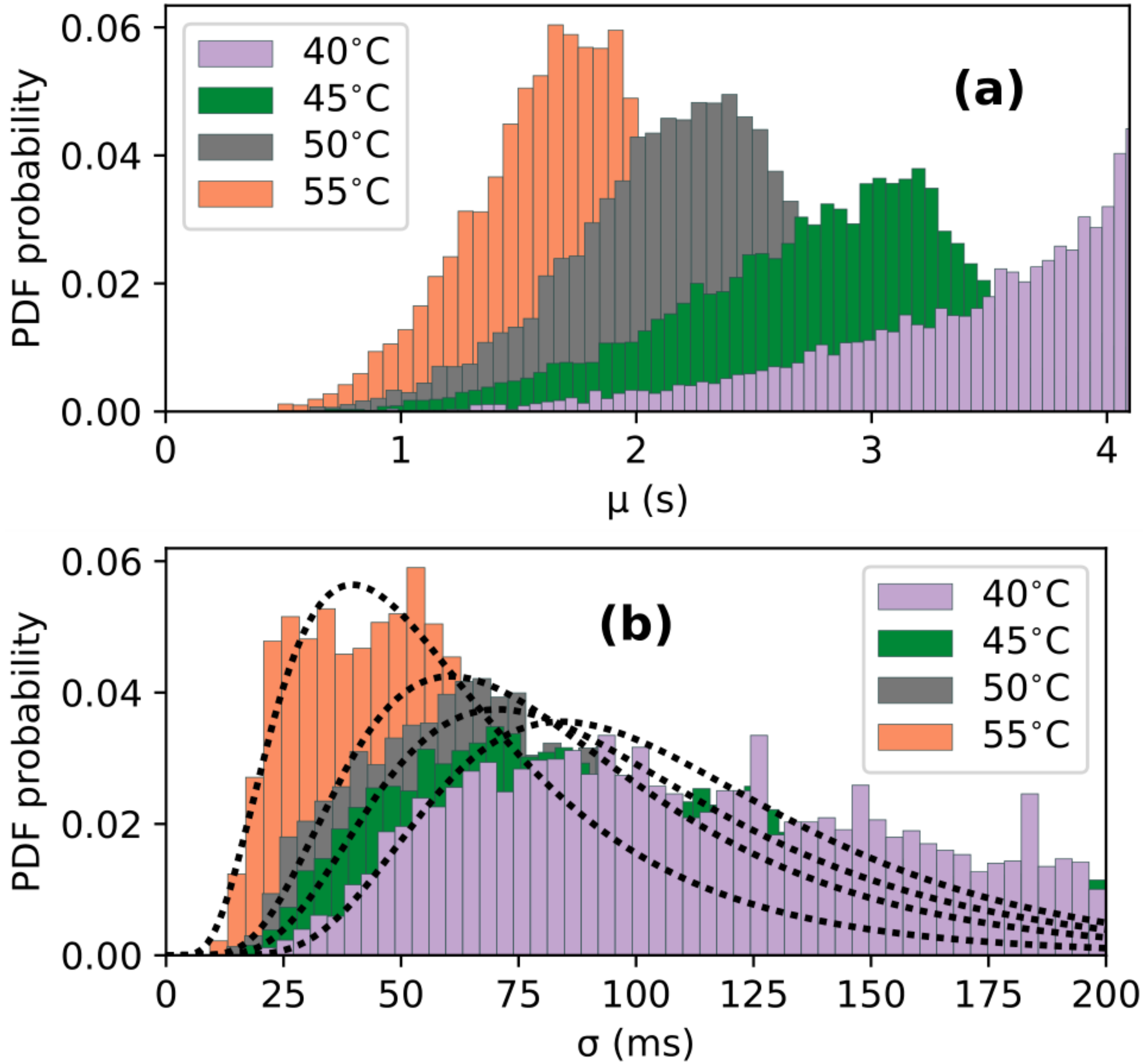


Individual Cell Failure Probabilities

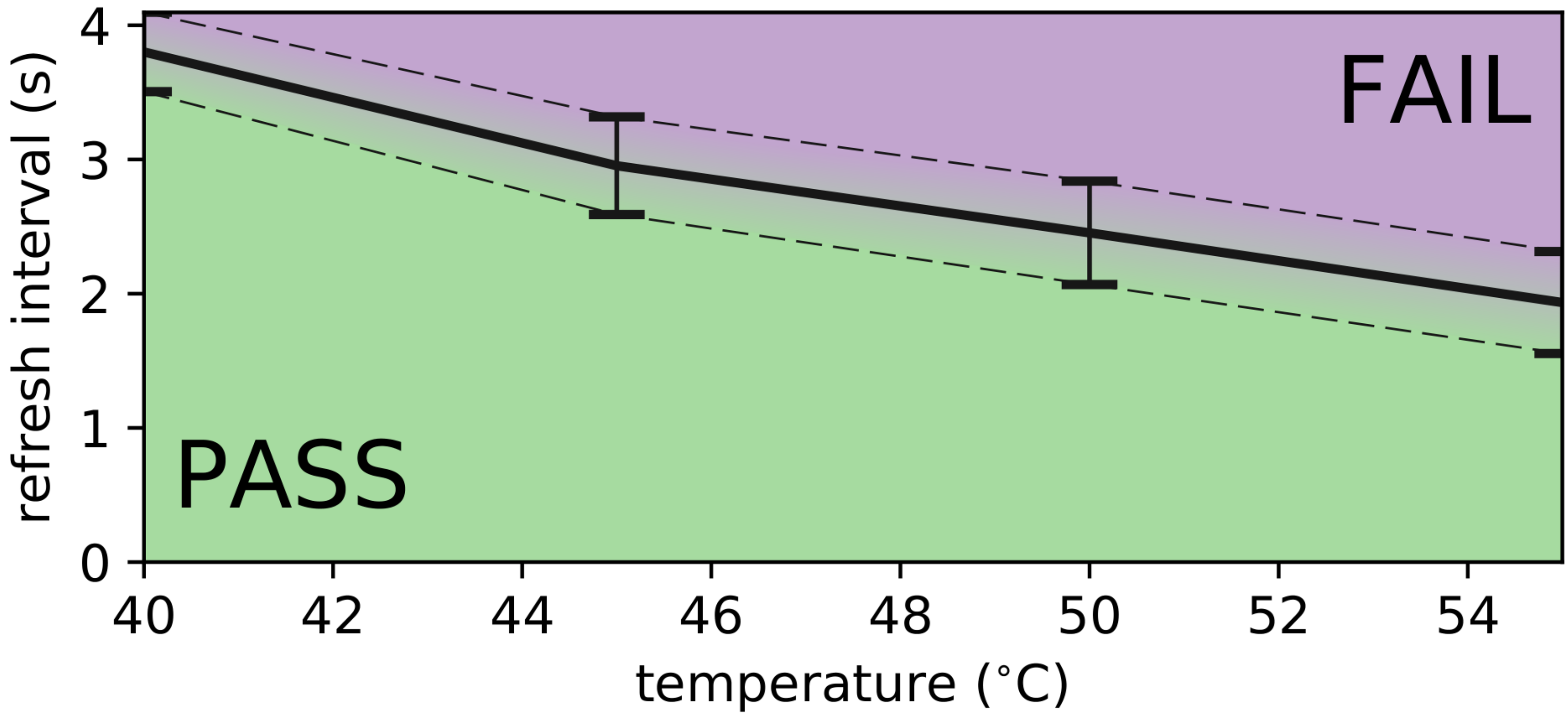


- Single representative chip of Vendor B at 40° C
- Refresh intervals ranging from 64ms to 4096ms

Individual Cell Failure Distributions



Single-cell Failures With Temperature



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