Computer Architecture

Lecture 12a: Research Presentation

Prof. Onur Mutlu
ETH Zürich

Fall 2020

30 October 2020

ETH Zürich

SAFARI

Bit-Exact ECC Recovery (BEER):

Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics

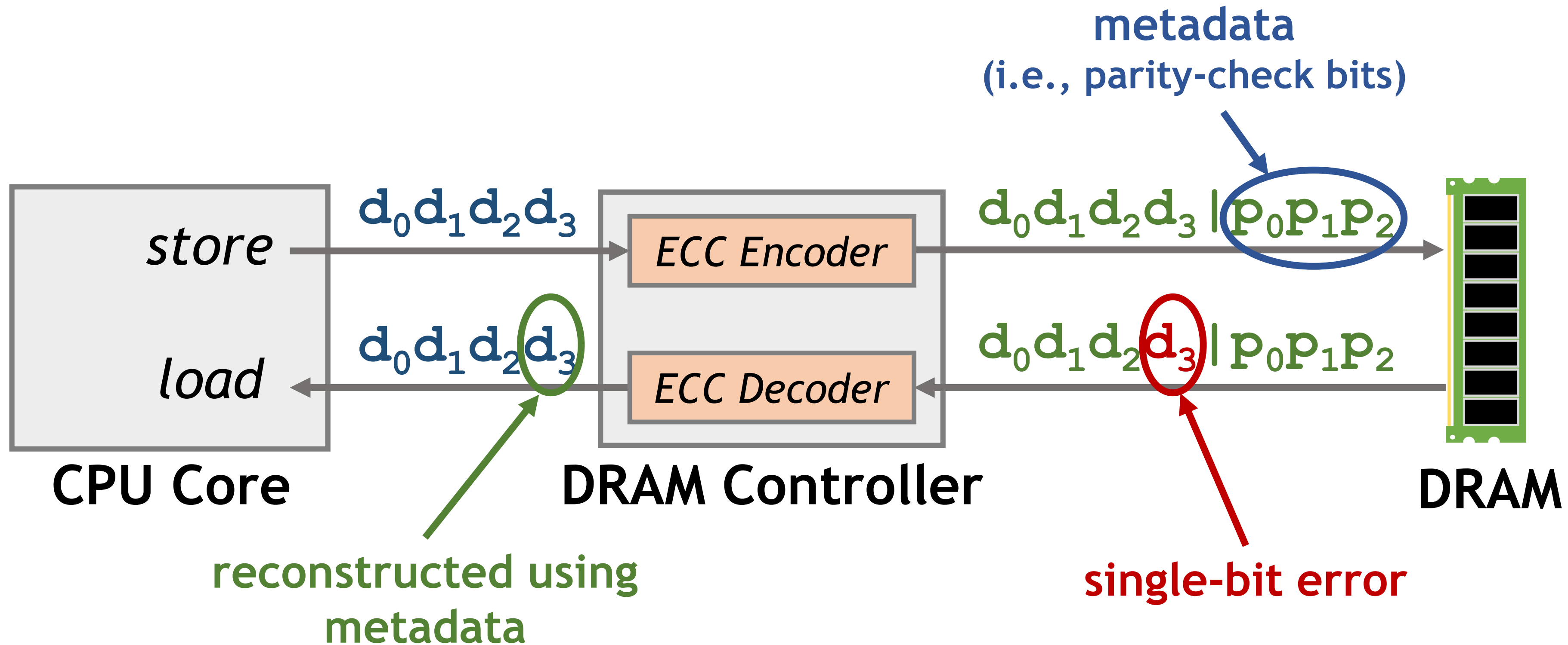
Minesh Patel, Jeremie S. Kim
Taha Shahroodi, Hasan Hassan, Onur Mutlu

MICRO 2020

Extended talk for Computer Architecture HS 2020

Error Correction Codes (ECCs)

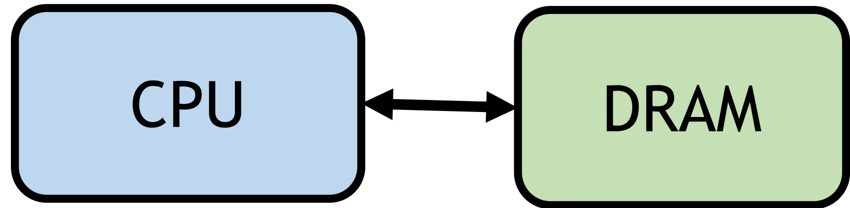
- **Key idea:** add **metadata** that allows the memory controller to reconstruct corrupt data on a bit flip



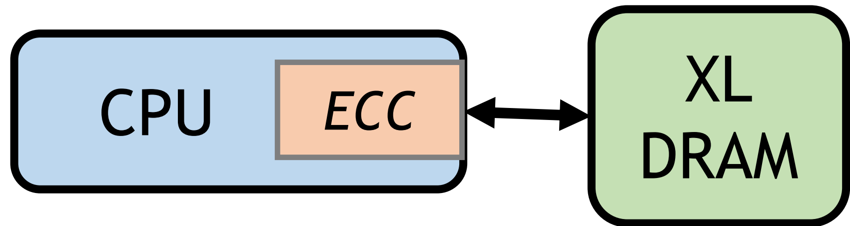
- More metadata allows correcting more errors

Three Types of DRAM Systems

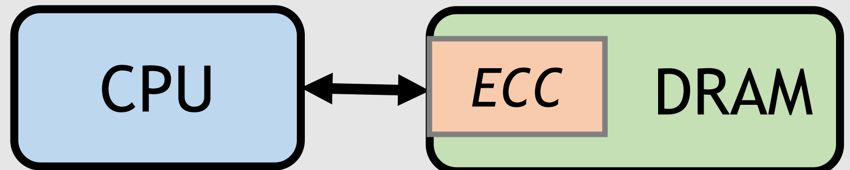
No-ECC



Rank-level ECC



On-Die-ECC



Executive Summary

Problem: DRAM on-die ECC **complicates** third-party reliability studies

- **Proprietary** design **obfuscates** raw bit errors in an **unpredictable** way
- **Interferes** with (1) design, (2) test & validation, and (3) characterization

Goal: understand **exactly how** on-die ECC obfuscates errors

Contributions:

1. **BEER**: new testing methodology that determines a DRAM chip's **unique on-die ECC function** (i.e., its parity-check matrix)
   - Exploits **ECC-function-specific** uncorrectable error patterns
   - Requires no hardware support, inside knowledge, or metadata access
2. **BEEP**: new error profiling methodology that infers the **raw bit error locations** of error-prone cells from the **observable uncorrectable errors**

BEER Evaluations:

- Apply BEER to 80 real LPDDR4 chips from 3 major DRAM manufacturers
- Show correctness in simulation for 115,300 codes (4-247b ECC words)

We hope BEER and BEEP enable valuable studies in the future

Talk Outline

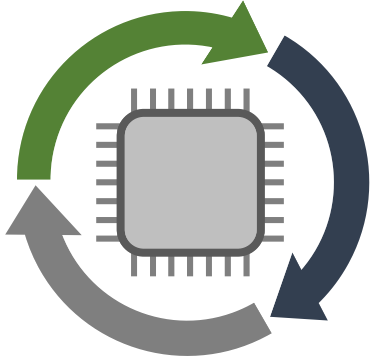
Challenges Caused by Unknown On-Die ECCs

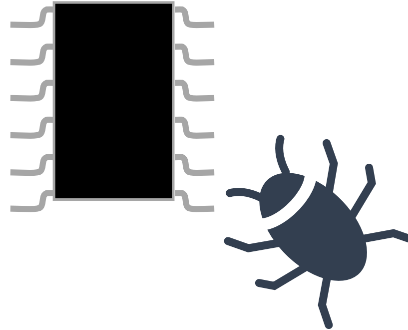
BEER: Determining the On-Die ECC Function

Evaluating BEER in Experiment and Simulation

BEEP and Other Practical Use Cases for BEER

Third-Party DRAM Users





System Architects
Design Error Mitigations

Test Engineers
Third-Party Testing

Research Scientists
Error-Characterization

Need to understand
a DRAM chip's **reliability characteristics**

Inter-chip variation?

Temperature dependence?

'Weak' cell locations?

Statistical error properties?

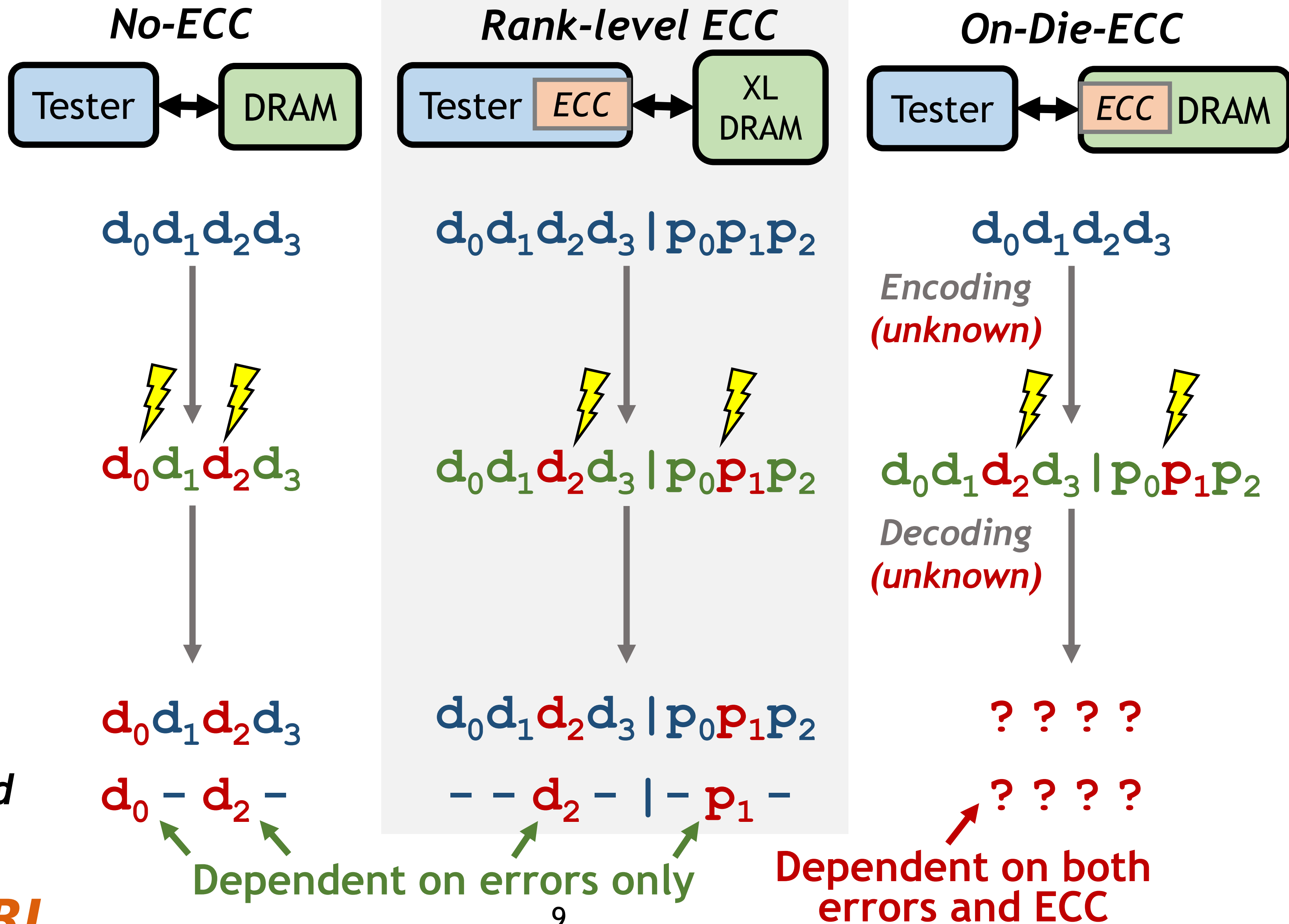
Aggregate failure rates?

Minimum operating timings?

Third-Party DRAM Users

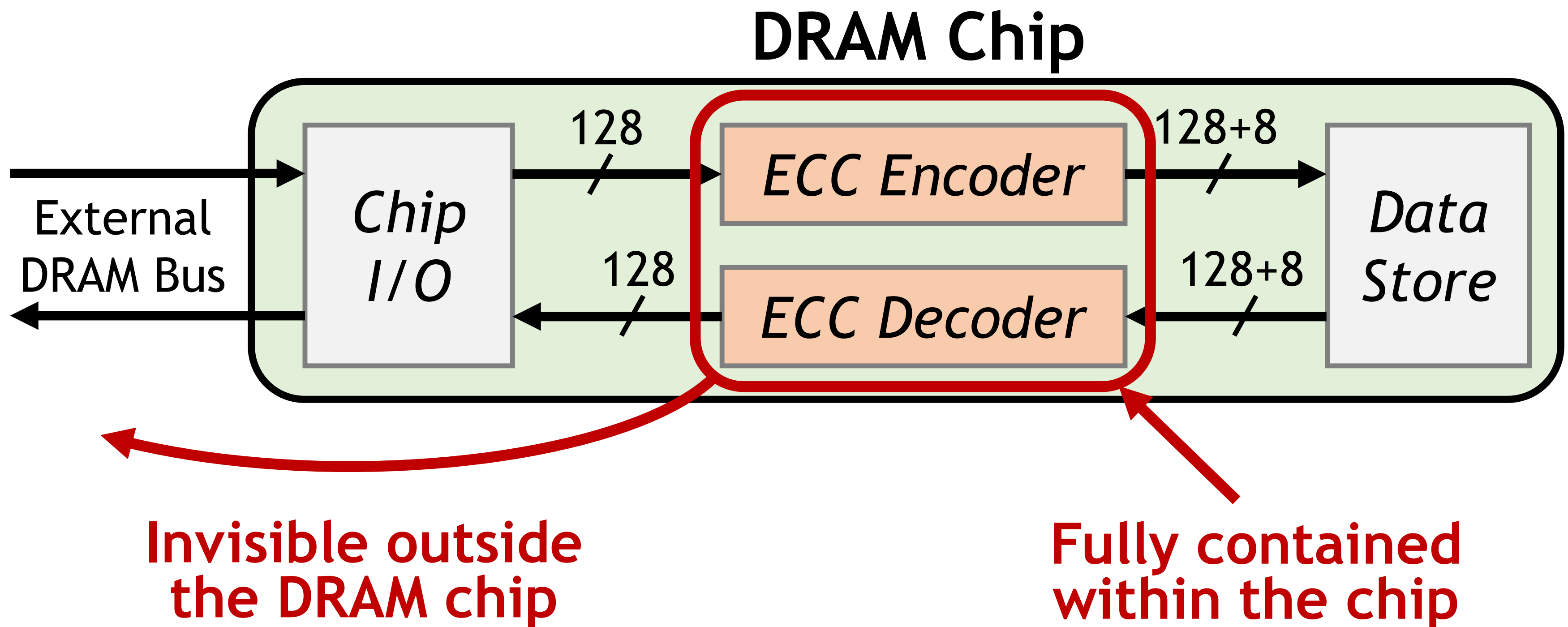
But how do we study
DRAM reliability characteristics?

Testing and Error Characterization

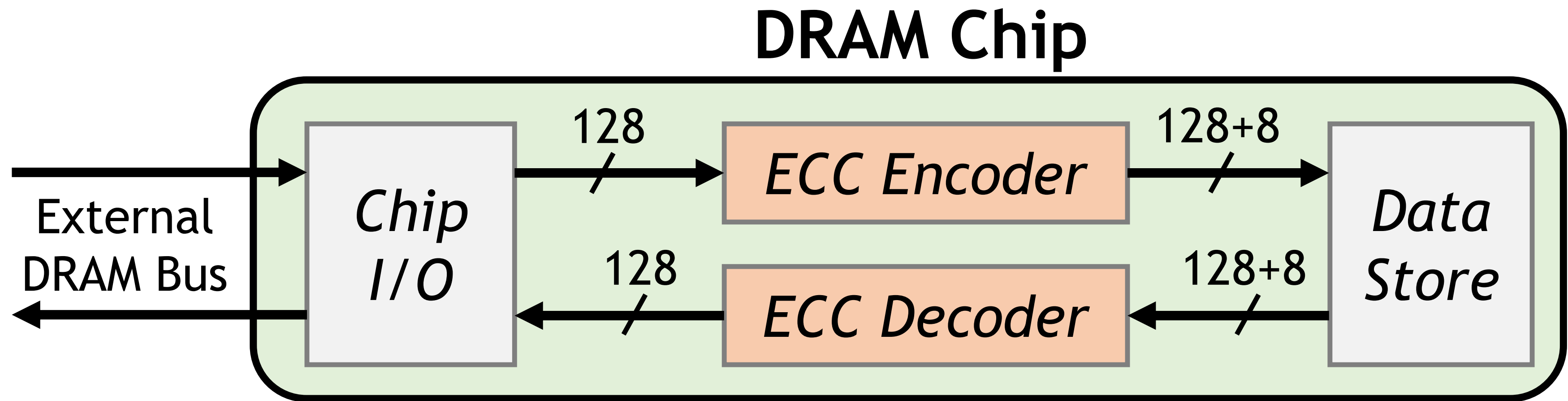


A Typical DRAM On-Die ECC Design

- 128-bit single-error correcting (SEC) Hamming code



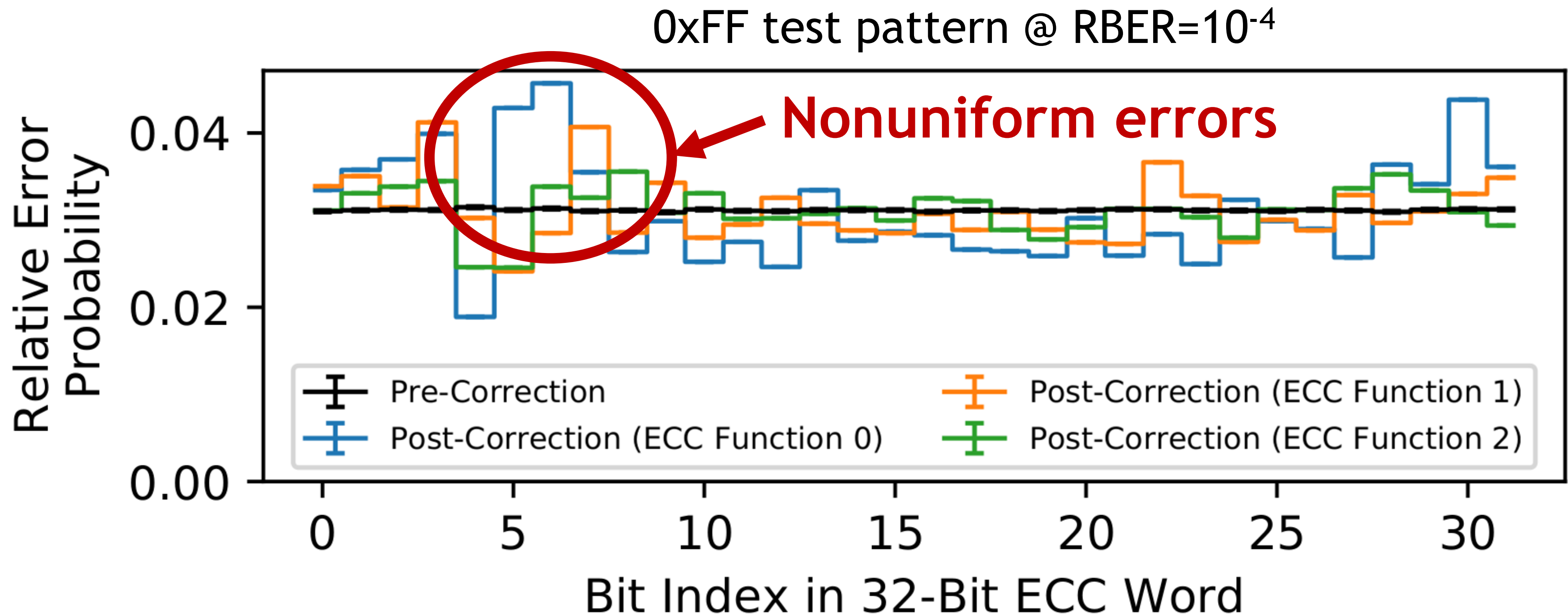
A Typical DRAM On-Die ECC Design



- Many ways to implement a 128-bit Hamming code
  - Different **ECC functions**
    - Known as **parity-check matrices** (i.e., ***H*-matrices**)
    - All correct 1 error, but act **differently** on 2+ errors
- Manufacturers are free to choose any design
  - Circuit optimization goals (e.g., area, power)
  - Details are **highly proprietary** (even under NDA)

Effect of Different On-Die ECC Designs

- Simulating **uniform-random errors** in a 32b ECC word



- 32-bit single-error correction Hamming codes
- Three different parity-check matrices

Effect of Different On-Die ECC Designs

The **same** error characteristics
can appear very **different**
with different ECC functions

Challenges for Third Parties

System Architects: Designing Error Mitigations

- On-die ECC forces system architects to support **unpredictable**, **chip-dependent** memory reliability characteristics

Test/Validation Engineers: Post-Manufacturing Testing

- On-die ECC **hides** the root-causes of uncorrectable errors and **defeats** test patterns designed to target physical cells

Research Scientists: Error-Characterization Studies

- On-die ECC **conflates** raw bit errors with ECC artifacts, effectively **obfuscating** the true physical cell characteristics

Challenges for Third Parties

These challenges all arise
from the **inability** to predict
how ECC transforms error patterns

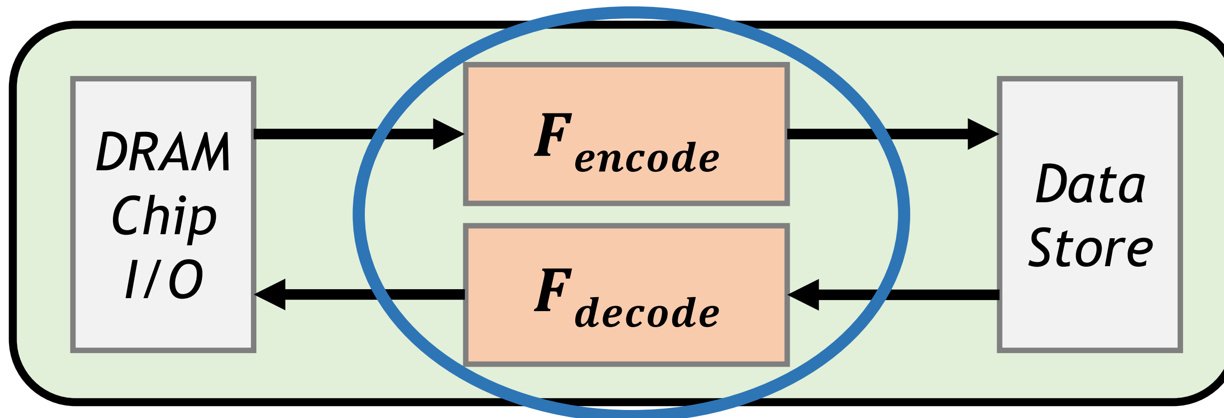
Overcoming Challenges of On-Die ECC

Our goal: Determine the on-die ECC function **without**:

(1) hardware support or tools

(2) prior knowledge about on-die ECC

(3) access to ECC metadata (e.g., syndromes)



- Reveals how on-die ECC scrambles errors (BEER)
- Allows inferring raw bit error locations (BEEP)

Talk Outline

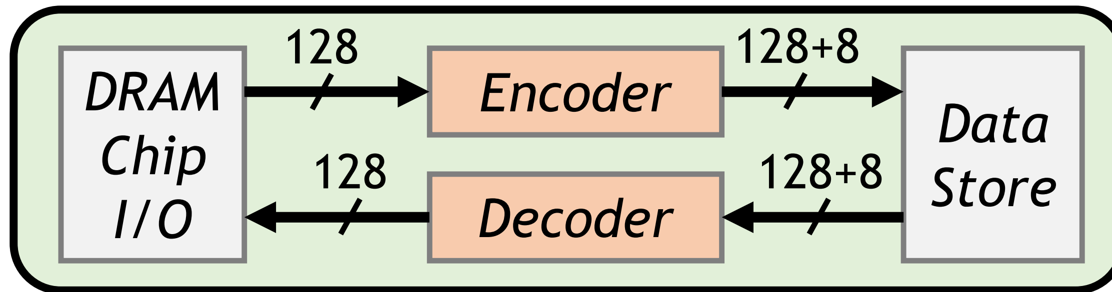
Challenges Caused by Unknown On-Die ECCs

BEER: Determining the On-Die ECC Function

Evaluating BEER in Experiment and Simulation

BEEP and Other Practical Use Cases for BEER

Typical On-Die ECC Function



- Encoder and decoder both use **linear operations**

Encoder

dataword $\boldsymbol{d} = [\bullet\ \bullet\ \bullet\ \bullet]$

$\boldsymbol{d}$ $[\bullet\ \bullet\ \bullet\ \bullet] * \boldsymbol{G}$

codeword $\boldsymbol{c} = [\bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet]$

Define the ECC function

Decoder

dataword′ $\boldsymbol{d'} = [\bullet\ \bullet\ \bullet\ \bullet]$

$\boldsymbol{H} * \boldsymbol{c'}$ $[\bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet]^T$

codeword′ $\boldsymbol{c'} = [\bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet]$

Error Correction During Decoding

- Two-step decoding algorithm: **syndrome decoding**
  1. Calculate an **error syndrome** that points to error(s)
  2. Correct detected errors (if any)

Correctable Errors

$$H * c' = s$$

s points to the error location (if any)

Uncorrectable Errors

$$H * c' = s$$

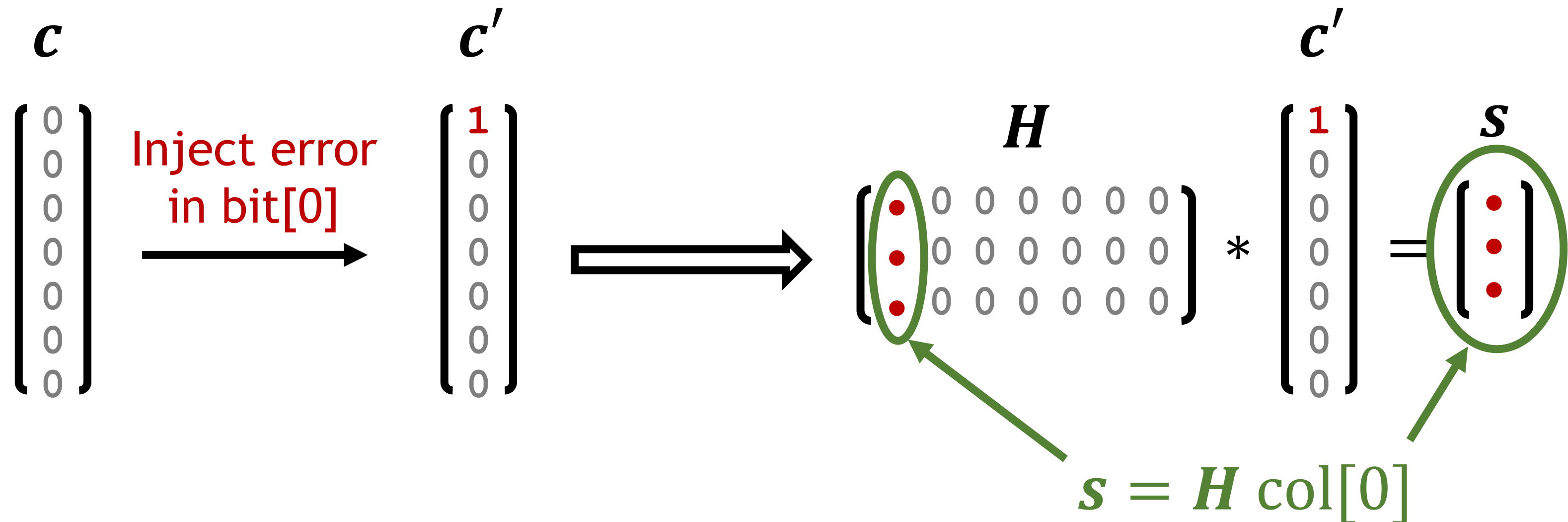
s points to an arbitrary H-dependent position

Error Correction During Decoding

Key idea: exploit the $\boldsymbol{H}$-dependence of uncorrectable errors to disambiguate ECC functions

Determining the On-Die ECC Function

- **Approach:** iteratively isolate linear components of $\boldsymbol{H}$
  - Demonstrated by [Cojocar+, SP'19] for rank-level ECC



- Can systematically extract each column of $\boldsymbol{H}$
- Determine entire $\boldsymbol{H}$ by extracting all columns

Determining the On-Die ECC Function

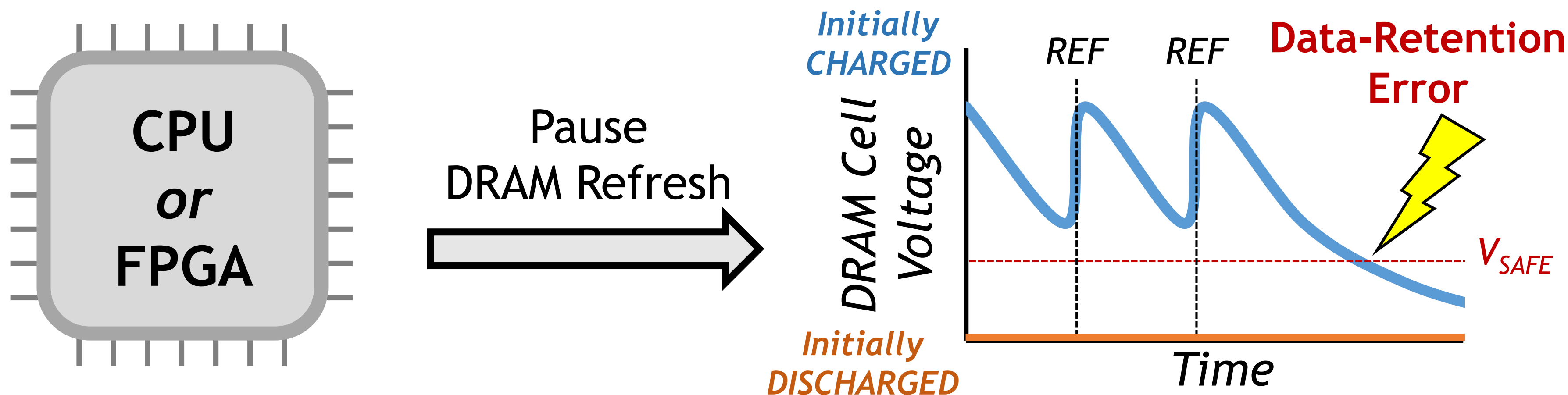
$\boldsymbol{H}$

On-die ECC causes two **challenges:**

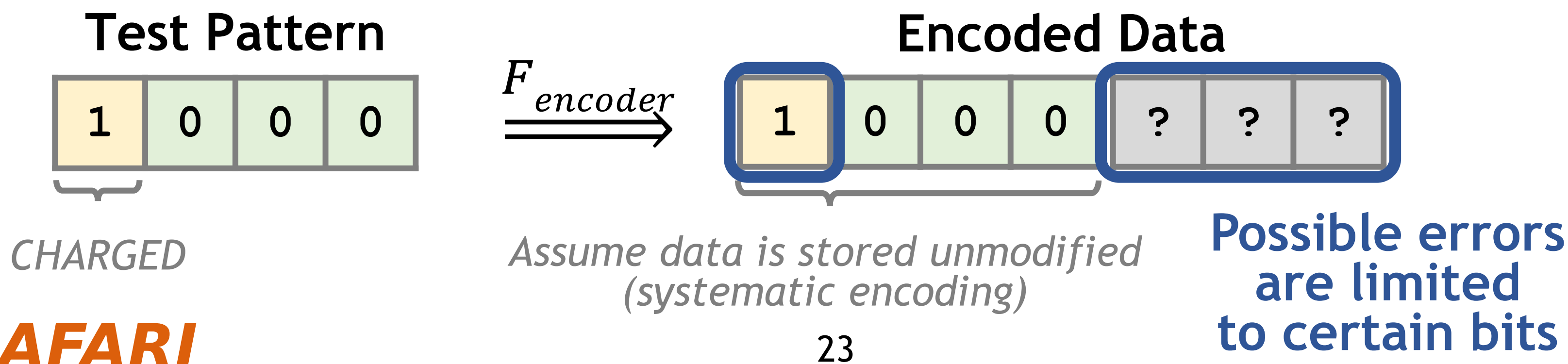
1. No way to inject errors in `bit[n]`
2. No way to observe error syndromes

Challenge 1: Injecting Errors

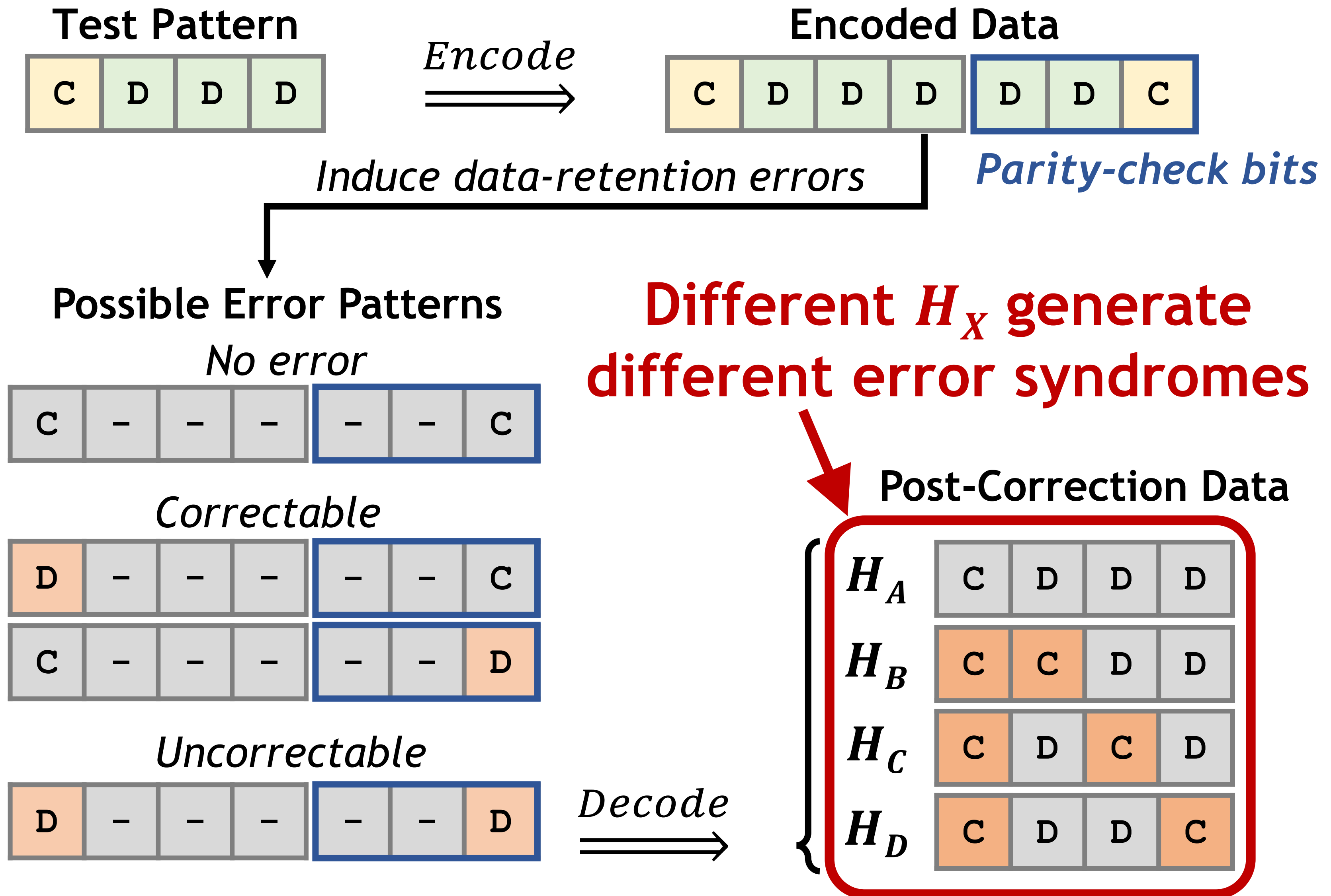
- **Key idea:** deliberately induce data-retention errors



- Difference between CHARGED and DISCHARGED cells allows us to restrict errors to specific bit positions



Challenge 2: Inferring Error Syndromes

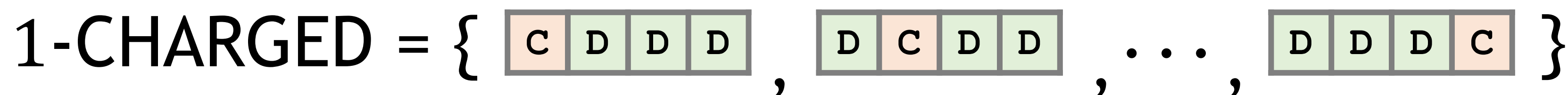


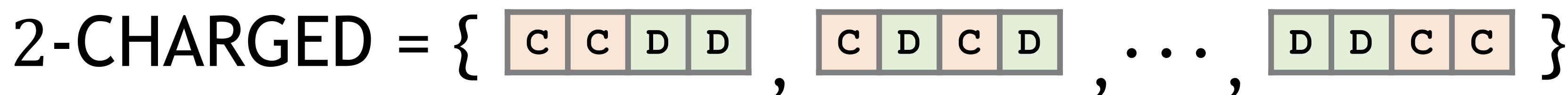
Challenge 2: Inferring Error Syndromes

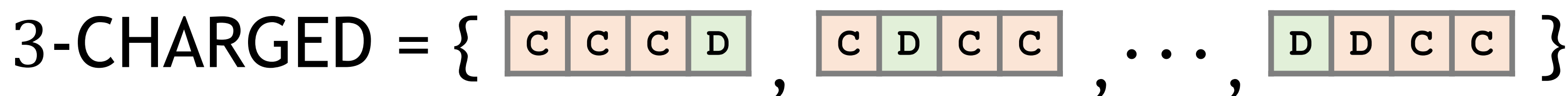
We can **differentiate** error syndromes from **uncorrectable error** patterns

Choosing a Set of Test Patterns

- We consider the "n-CHARGED" test patterns:







- Our paper explains that the combined {1,2}-CHARGED patterns are sufficient to identify the ECC function
- For each test pattern, we find **all possible** uncorrectable errors that can occur
  - Exploit **uniform-randomness** of data-retention errors
  - Even one DRAM chip provides millions of samples
    - E.g., 2 GiB DRAM module yields 2^{24} 128-bit words

BEER: Bit-Exact ECC Recovery

1. Experimentally induce data-retention errors using {1,2}-CHARGED test patterns

2. For each test pattern, identify all possible uncorrectable errors

3. Solve for the ECC function with the observed behavior using a SAT solver

Talk Outline

Challenges Caused by Unknown On-Die ECCs

BEER: Determining the On-Die ECC Function

Evaluating BEER in Experiment and Simulation

BEEP and Other Practical Use Cases for BEER

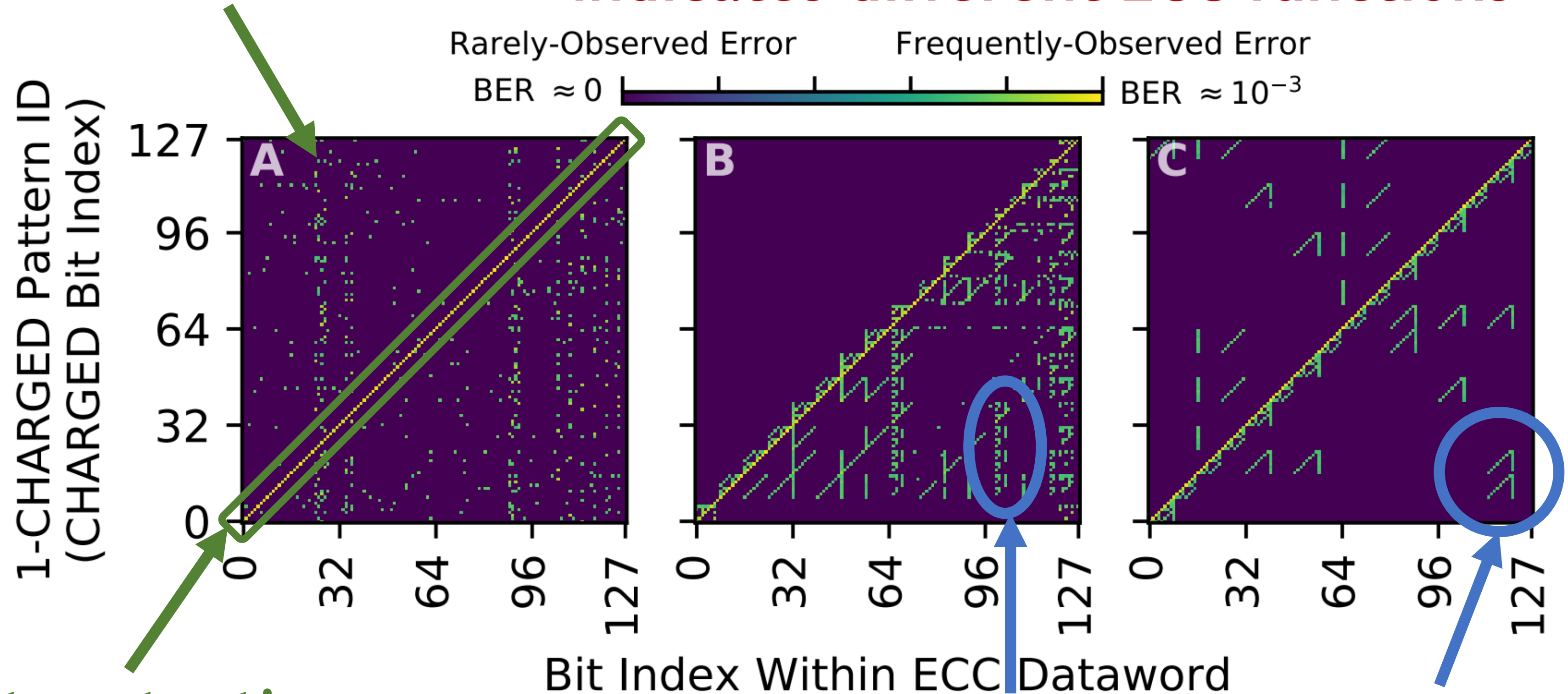
Experimental Methodology

- 80 LPDDR4 chips from 3 DRAM manufacturers
  - Manufacturers **anonymized** as ‘A’, ‘B’, and ‘C’
  - Temperature-controlled testing infrastructure
  - Control over DRAM timings (including refresh)

- **Refresh windows** between 1-30 minutes at 30-80°C
  - Leads to **bit error rates** (BERs) between 10^{-7} and 10^{-3}
  - BERs **far larger** than those of unwanted soft errors

Applying BEER to LPDDR4 Chips

- Study the uncorrectable errors in the 1-CHARGED patterns



Applying BEER to LPDDR4 Chips

1. **Different manufacturers** appear to use **different** on-die ECC functions

2. Chips of the **same model number** appear to use **identical** ECC functions

(shown in our paper)

Solving for the ECC Function

- We use the **Z3**[†] **SAT solver** to identify the $\boldsymbol{H}$-matrix
  - We demonstrate BEER for SEC Hamming codes, but it should readily extend to **all** linear block codes (e.g., BCH)
- We **open-source** our BEER implementation on **GitHub**
  - https://github.com/CMU-SAFARI/BEER
- Unfortunately, we face two **limitations** to validation:
  1. No way to check the **final results** since we cannot see into the on-die ECC implementation
  2. We cannot share our final matrices due to **confidentiality** reasons

[†]L. De Moura and N. Bjørner, "Z3: An Effient SMT Solver," TACAS, 2008.

Solving for the ECC Function

We validate BEER in **simulation** to:

1. Evaluate **correctness**
2. **Overcome** confidentiality issues
3. Test a **larger** set of ECC codes

Simulation Methodology

- We use the EINSim† DRAM error-correction simulator

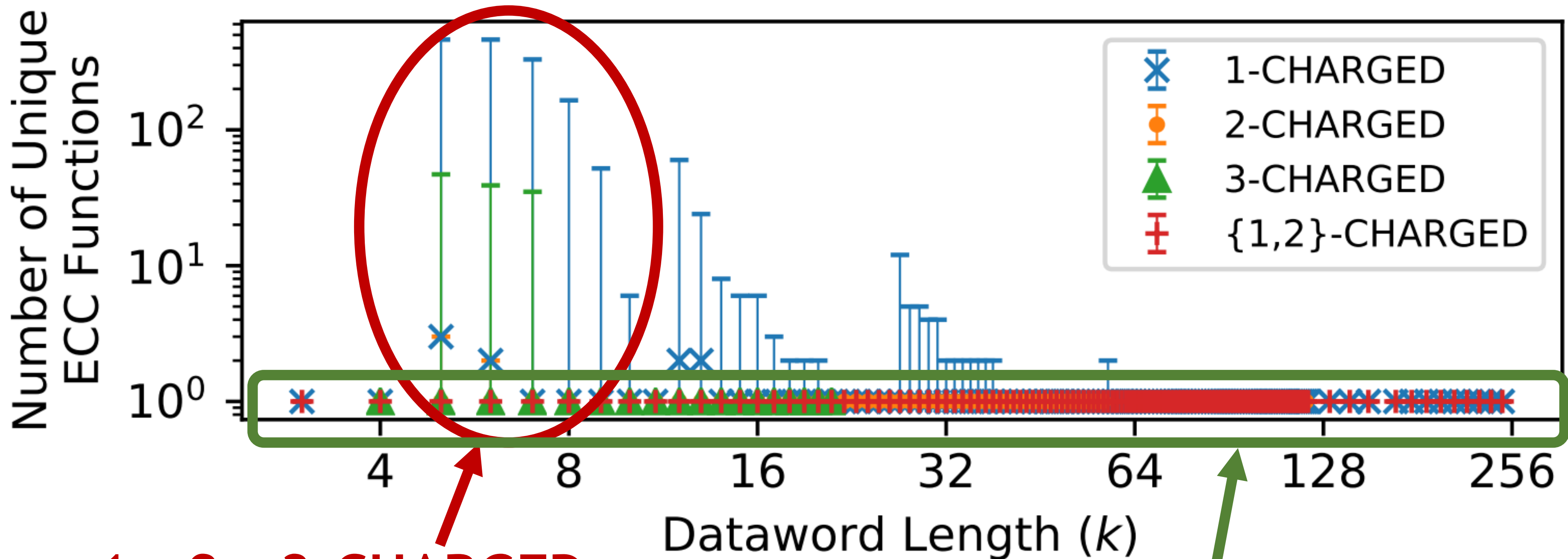
- We simulate 115,300 different SEC Hamming codes
  - ECC dataword lengths from 4 to 247 bits
  - 1-, 2-, 3-, and {1,2}-CHARGED test patterns

- For each test pattern:
  - Simulate 10^9 ECC words (≈14.9 GiB for 128-bit words)
  - Simulate data-retention errors with BER between 10^{-5} and 10^{-2}

†Patel et al., "*Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices*," DSN, 2019.

BEER Correctness Evaluation

- Evaluate the number of SAT solutions found by BEER
  - Shows whether the 'unique' solution is identified



1-, 2-, 3-CHARGED patterns individually do not always succeed

{1,2} -CHARGED patterns succeed for all test cases

BEER Correctness Evaluation

BEER **successfully** identifies the ECC function using the **{1,2}-CHARGED** test patterns

Two Other Evaluations in the Paper

1. Practicality of BEER's SAT problem
   - Measure SAT problem runtime and memory consumption
   - Negligible for short codes (i.e., < 1 minute, < 1 MiB RAM)
   - Realistic for long codes given that BEER is run offline
     - e.g., 57.1 hours + 6.3 GiB RAM for 128-bit code

2. Analytical experimental runtime analysis
   - Majority time is spent waiting for data-retention errors
   - 4.2 hours of testing per chip in our experiments

Talk Outline

Challenges Caused by Unknown On-Die ECCs

BEER: Determining the On-Die ECC Function

Evaluating BEER in Experiment and Simulation

BEEP and Other Practical Use Cases for BEER

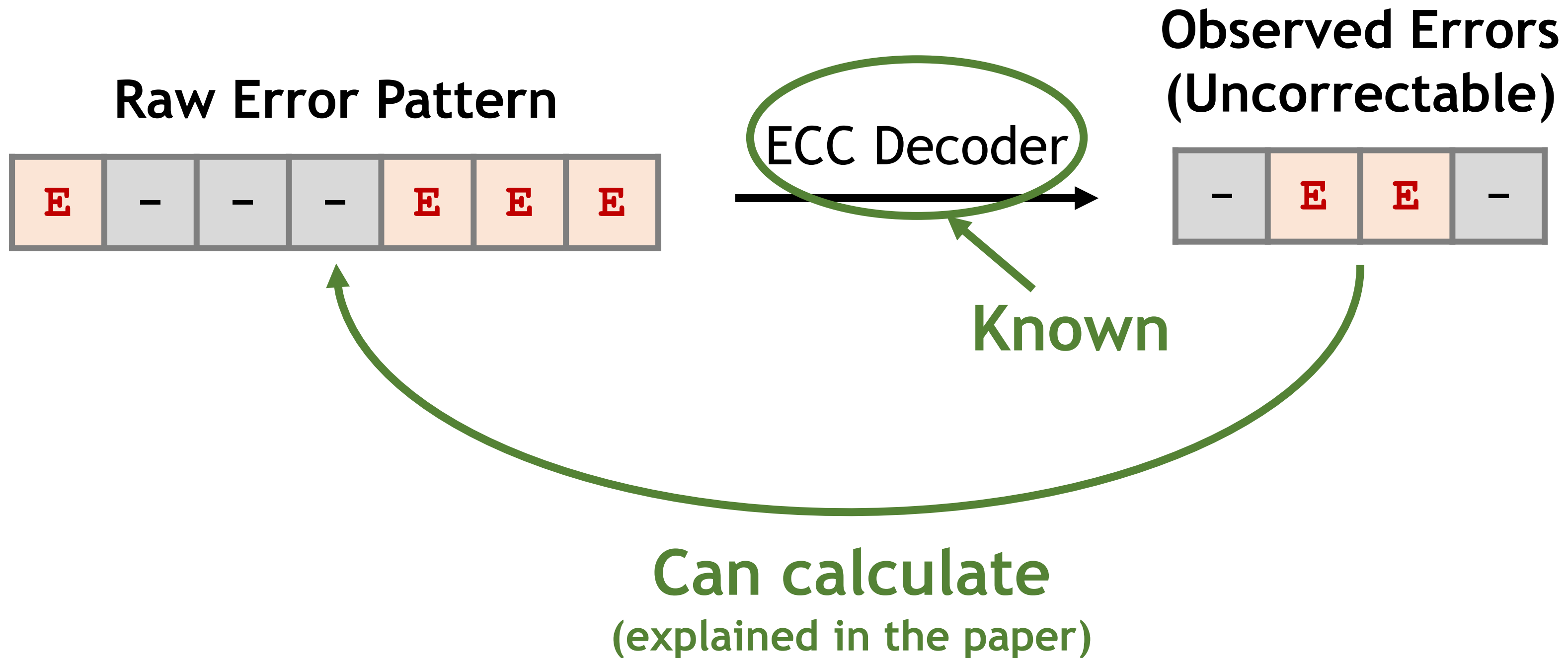
Practical Use Cases for BEER

Error Profiling

BEEP: identifying **raw bit error** locations corresponding to observed **post-correction errors**

BEEP: Profiling for Raw Bit Errors

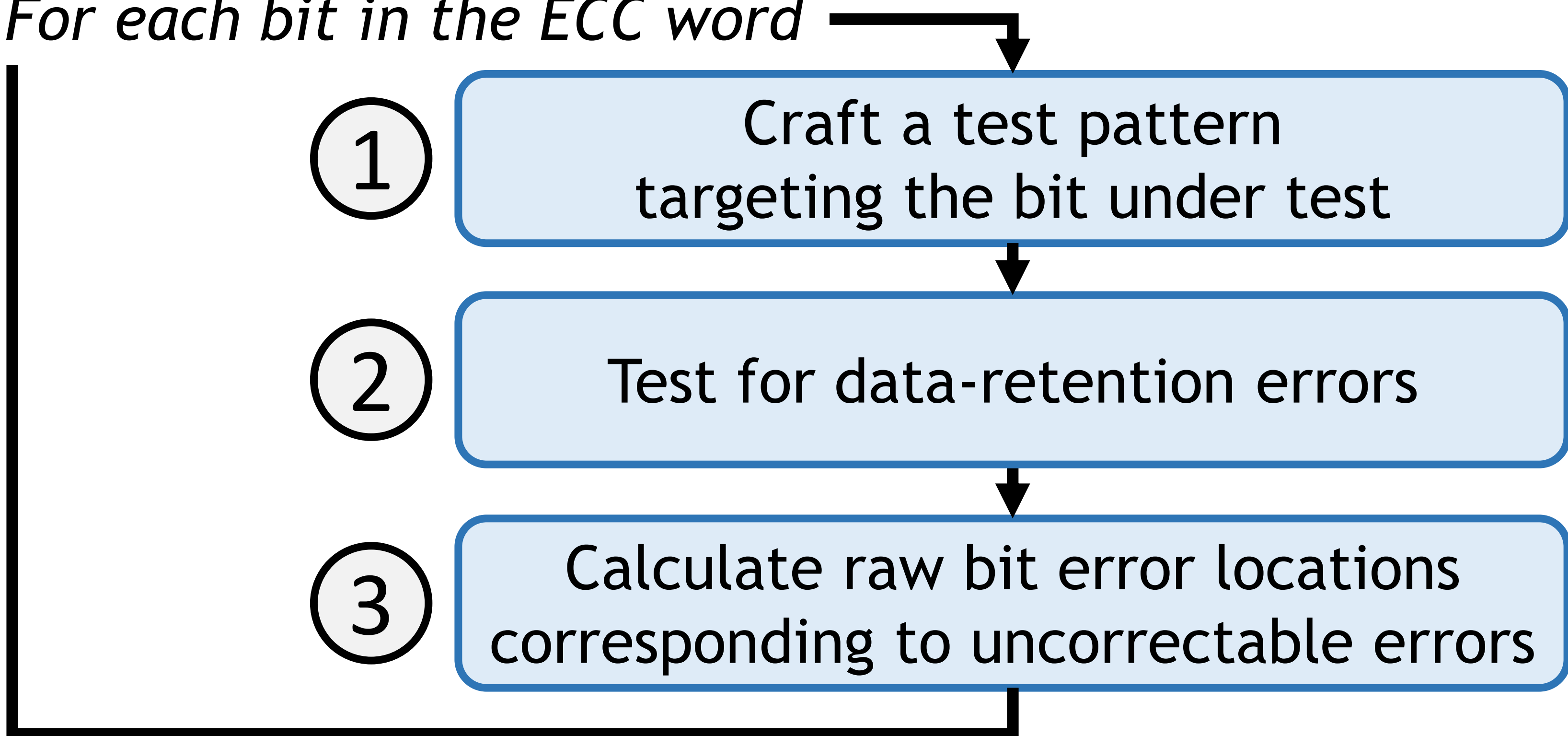
- **Key idea:** knowing the ECC function (i.e., via BEER) enables calculating raw bit error positions



- BEEP infers which **physical cells** are susceptible to data-retention errors using **only** the observed errors

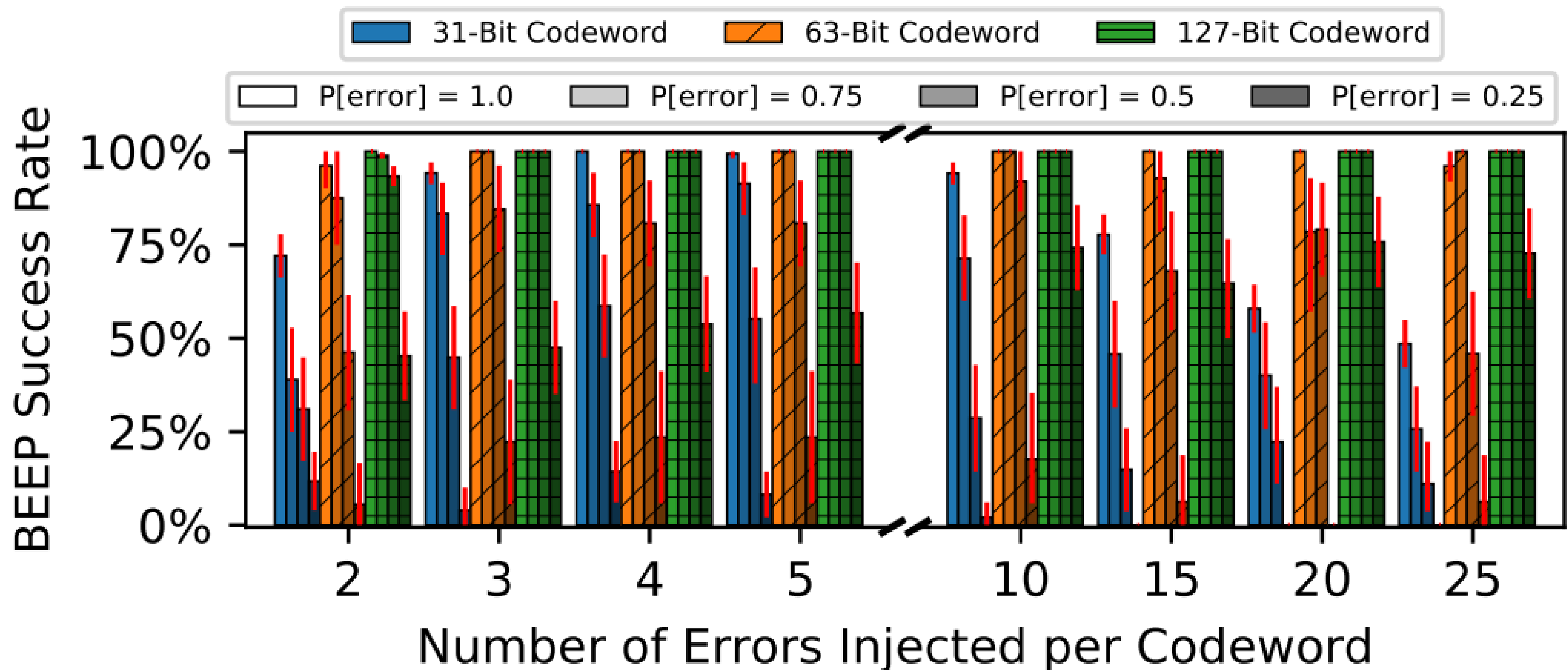
BEEP: High-Level Algorithm

- Iteratively test each bit in the ECC word and keep track of the error-prone cells it identifies



Evaluating BEEP's Accuracy

- We evaluate BEEP's **success rate** of identifying raw bit errors in simulation
  - Varying ECC word lengths and bit error rates
  - 100 ECC words simulated per measurement



SAFARI

Evaluating BEEP's Accuracy

BEEP is **more successful** for:

1. **Longer** ECC words
2. **Higher-probability** errors

Other Information in the Paper

- **Formalism** for BEER and the n-CHARGED test patterns
- **BEER** evaluations using **experiment** and **simulation**
  - Sensitivity to experimental noise
  - Analysis of experimental runtime
  - Practicality of the SAT problem (i.e., runtime, memory)
- **BEEP** evaluations in **simulation**
  - Accuracy at different error rates
  - Sensitivity to different ECC codes and word sizes
- Detailed discussion of **use-cases** for BEER
- Discussion on BEER's **requirements** and **limitations**

Executive Summary

Problem: DRAM on-die ECC **complicates** third-party reliability studies

- **Proprietary** design **obfuscates** raw bit errors in an **unpredictable** way
- **Interferes** with (1) design, (2) test & validation, and (3) characterization

Goal: understand **exactly how** on-die ECC obfuscates errors

Contributions:

1. **BEER**: new testing methodology that determines a DRAM chip's **unique on-die ECC function** (i.e., its parity-check matrix)
2. **BEEP**: new error profiling methodology that infers the **raw bit error locations** of error-prone cells from the **observable uncorrectable errors**

BEER Evaluations:

- Apply BEER to 80 real LPDDR4 chips from 3 major DRAM manufacturers
- Show correctness in simulation for 115,300 codes (4-247b ECC words)

https://github.com/CMU-SAFARI/BEER

We hope that both BEER and BEEP enable many valuable studies going forward

ETH Zürich

SAFARI

Bit-Exact ECC Recovery (BEER):

Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics

Minesh Patel, Jeremie S. Kim
Taha Shahroodi, Hasan Hassan, Onur Mutlu

MICRO 2020

Extended talk for Computer Architecture HS 2020