DR-STRaNGe:

End-to-End System Design for DRAM-based True Random Number Generators

F. Nisa Bostancı

Ataberk Olgun Lois Orosa A. Giray Yağlıkçı

Jeremie S. Kim Hasan Hassan Oğuz Ergin Onur Mutlu

SAFARI

ETH zürich

kasırga

TOBB ETÜ
University of Economics & Technology

DR-STRaNGe Summary

Motivation:

- Random numbers are important for many applications
- DRAM-based True Random Number Generators (TRNGs) can provide **true random numbers at low cost** on **a wide range** of systems

Problem: There is no end-to-end system design for DRAM-based TRNGs

1. Interference between regular memory requests and RNG requests **significantly slows down** concurrently running applications
2. Unfair prioritization of RNG applications **degrades system fairness**
3. High latency of DRAM-based TRNGs **degrades the RNG applications' performance**

Goal: A **low-cost** and **high-performance** end-to-end system design for DRAM-based TRNGs

DR-STRaNGe: An end-to-end system design for DRAM-based TRNGs that

- **Reduces the interference between regular memory requests and RNG requests** by separating them in the memory controller
- **Improves fairness across applications** with an RNG-aware memory request scheduler
- **Hides the large TRNG latencies** using a random number buffering mechanism combined with a new DRAM idleness predictor

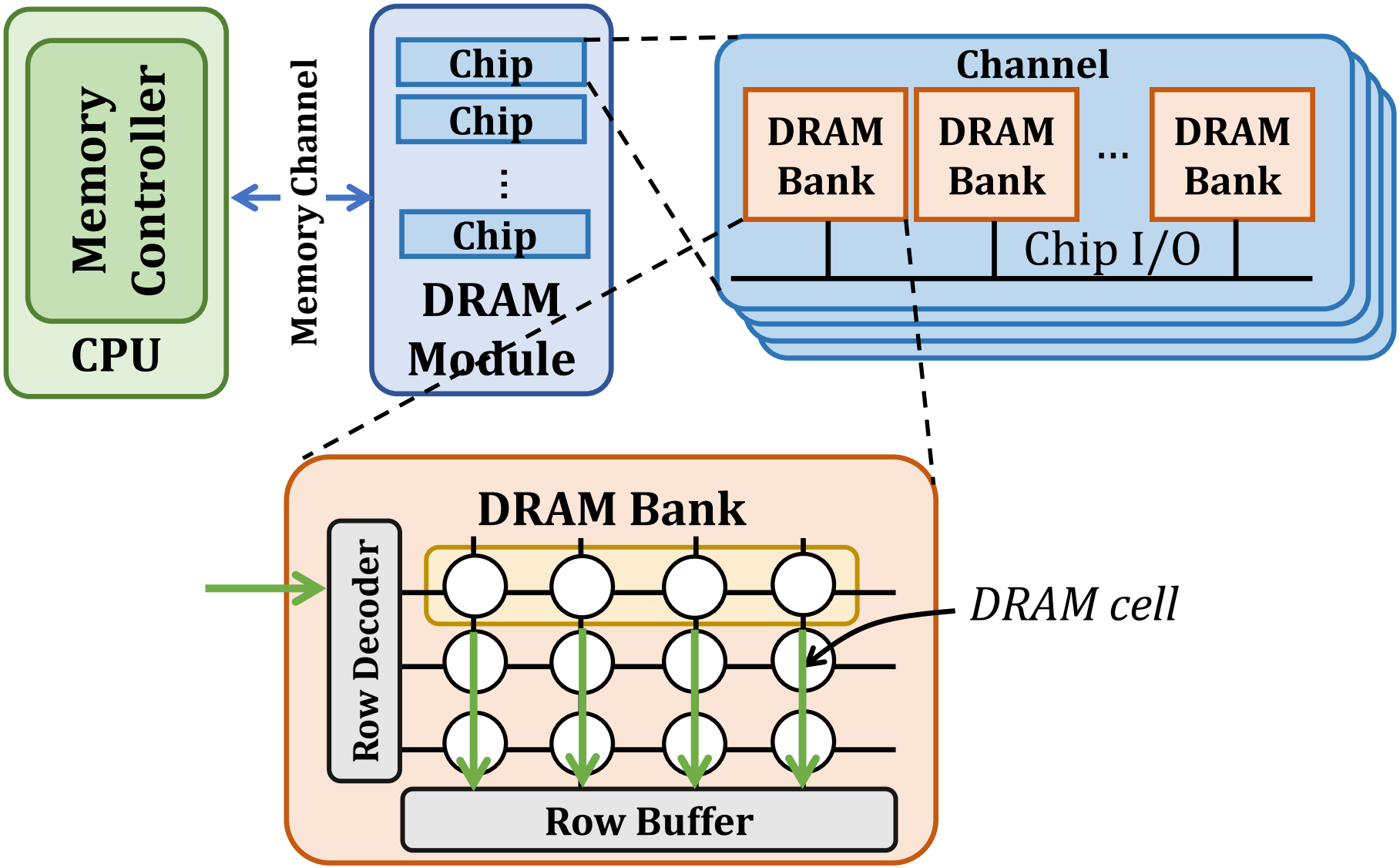
Results: DR-STRaNGe

- Improves the average performance of non-RNG (**17.9%**) and RNG (**25.1%**) applications
- Improves the average system fairness (**32.1%**) when generating random numbers at a 5 Gb/s throughput
- Reduces the average energy consumption (**21%**)

Outline

- Background
- Motivation and Goal
- DR-STRaNGe
  - Random Number Buffering Mechanism
  - RNG-Aware Scheduler
  - Application Interface
- Evaluation
- Conclusion

DRAM Organization

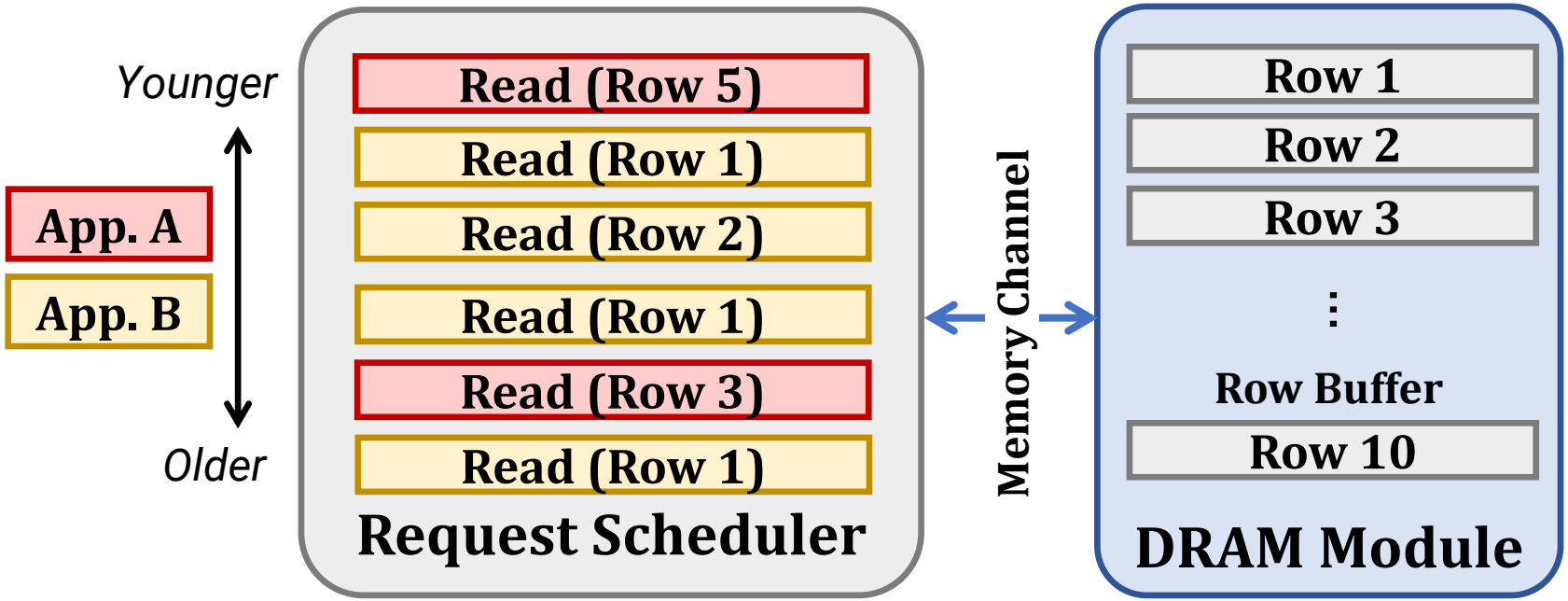


Memory Request Scheduling

Commonly used memory request schedulers aim to **maximize throughput** by leveraging **the row buffer locality**

(1) Requests to **open rows** over all requests

(2) *Older* requests over the *younger* ones

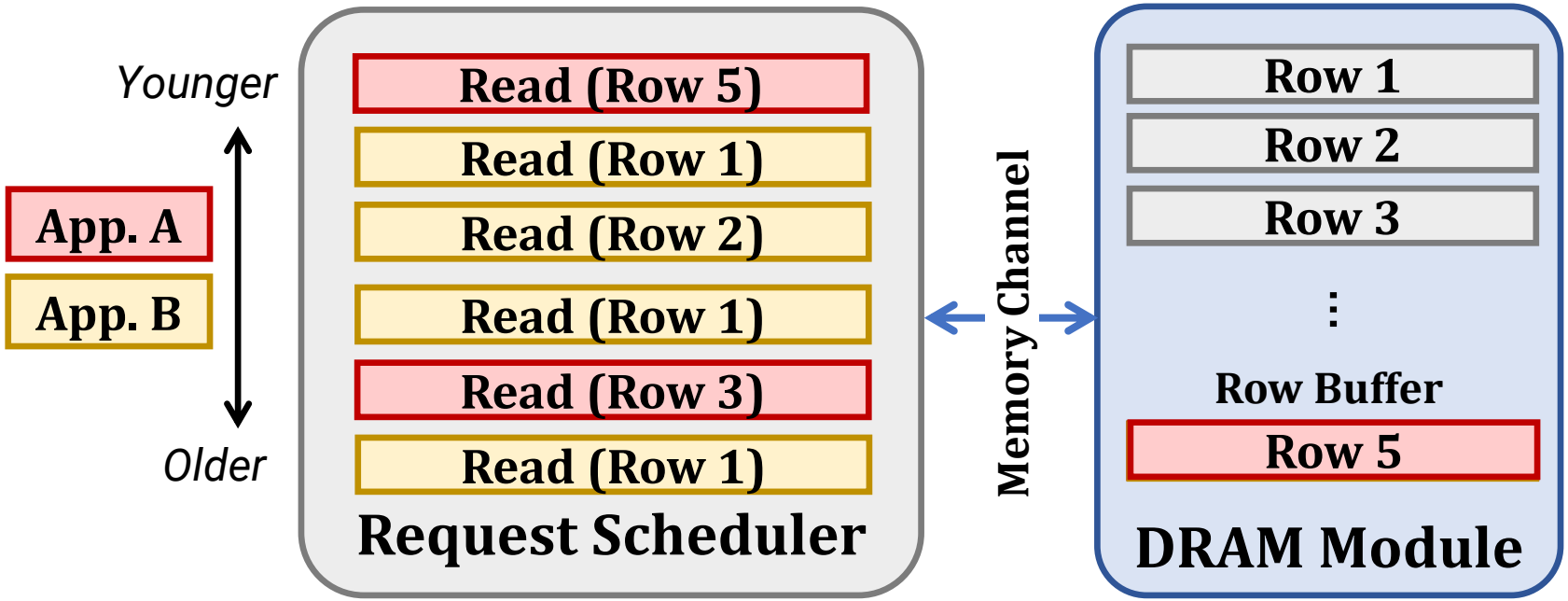


Memory Request Scheduling

Commonly used memory request schedulers aim to **maximize throughput** by leveraging **the row buffer locality**

(1) Requests to **open rows** over all requests

(2) *Older* requests over the *younger* ones

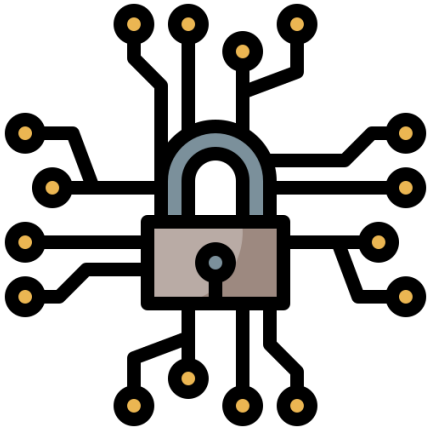


Outline

- Background
- Motivation and Goal
- DR-STRaNGe
  - Random Number Buffering Mechanism
  - RNG-Aware Scheduler
  - Application Interface
- Evaluation
- Conclusion

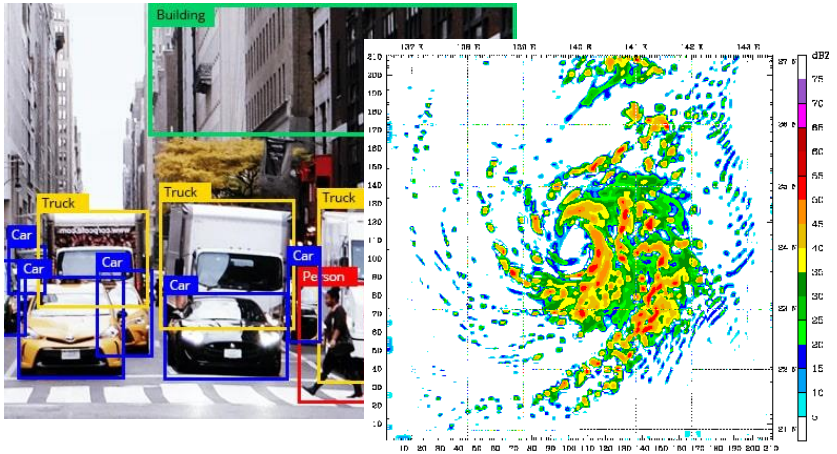
True Random Numbers (TRN)

- True random numbers are widely used in real world



Security Applications:

- Cryptographic key generation, authentication, countermeasures against hardware attacks, …
- Emerging protocols require **a very high TRNG throughput (~ Gb/s)**



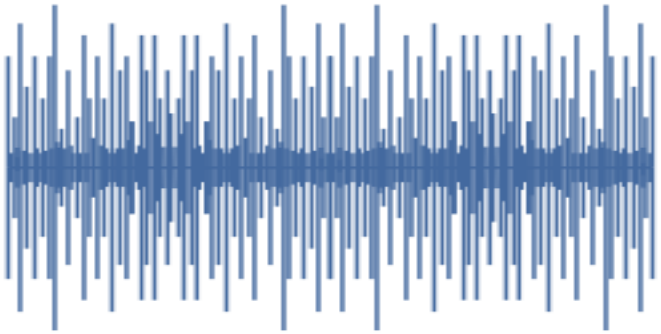
Other:

- Randomized algorithms, scientific simulation, statistical sampling, blockchain applications, …

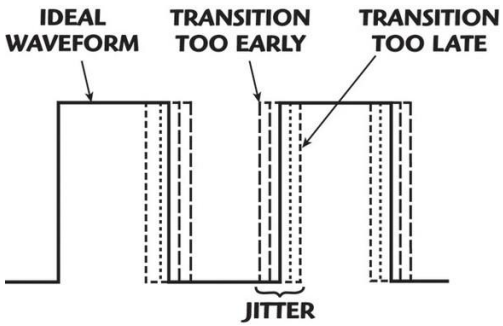
True Random Numbers (TRN)

- True random numbers are generated by harnessing entropy resulting from **random physical processes**

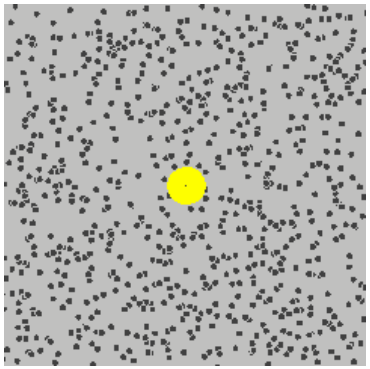
Thermal Noise



Clock Jitter

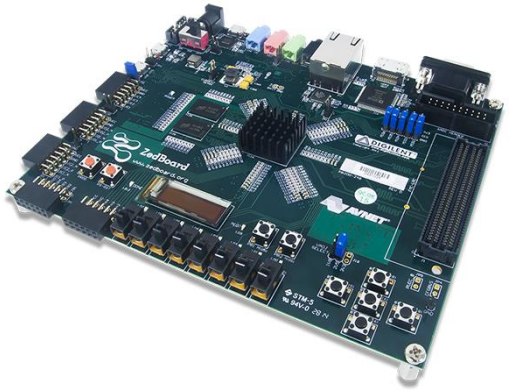


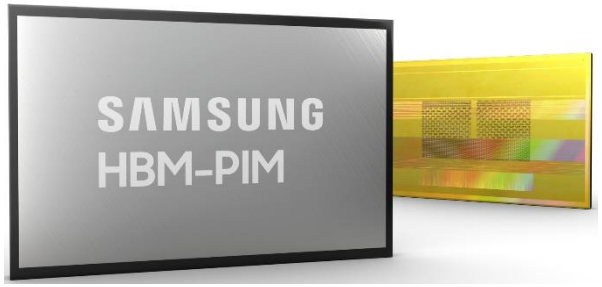
Brownian Motion



True Random Number Generators

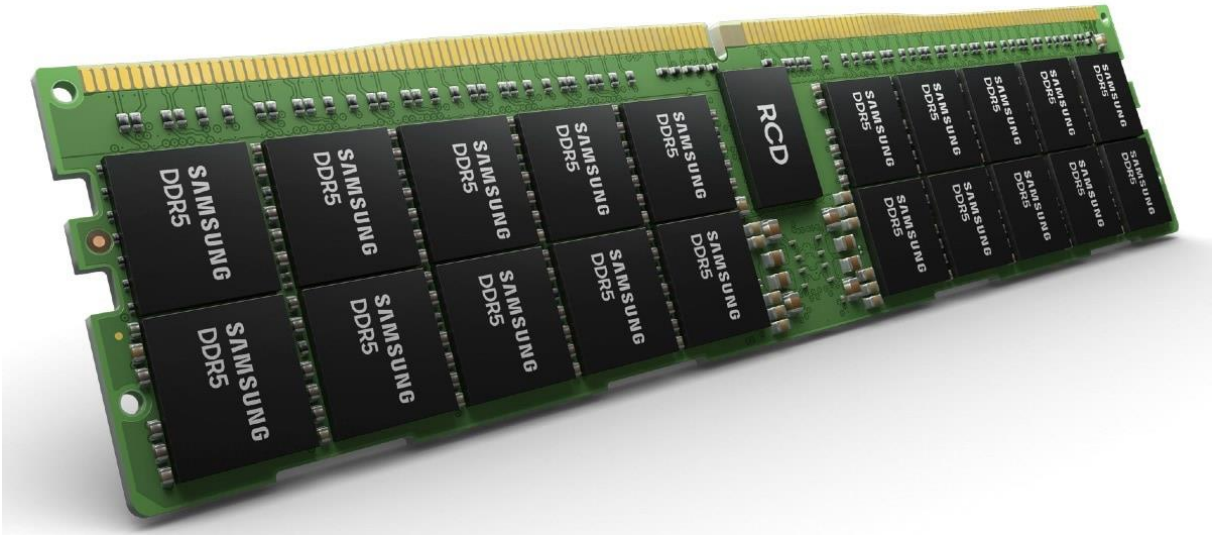
- Systems can generate true random numbers with **dedicated hardware true random number generators (TRNGs)**
  - Sample non-deterministic various physical phenomena
  - Not suitable for all systems





Why DRAM-based TRNGs?

DRAM is everywhere



DRAM-based TRNGs enable true random number generation **within widely available DRAM chips**

DRAM-based TRNGs

1. **Retention Failures**
   Fundamentally Slow: cells leak charge slowly

2. **Start-up Values**
   Fundamentally Slow: requires power-cycle

3. **Timing Failures**
   Fast: enabled by reducing DRAM command latencies

DRAM-based TRNGs

1 **Retention Failures**

Fundamentally Slow: cells leak charge slowly

2 **Start-up Values**

Fundamentally Slow: requires power-cycle

3 **Timing Failures**

Fast: enabled by reducing DRAM command latencies

Integration of DRAM-based TRNGs into Real Systems

No prior work provides
an **end-to-end system design**
to enable DRAM-based TRNGs
in **real systems**

Three Key Challenges

1. RNG Interference
significantly slows down concurrently-running applications

2. Unfair Prioritization
degrades overall system fairness

3. High TRNG Latency
degrades RNG applications' performance

Three Key Challenges

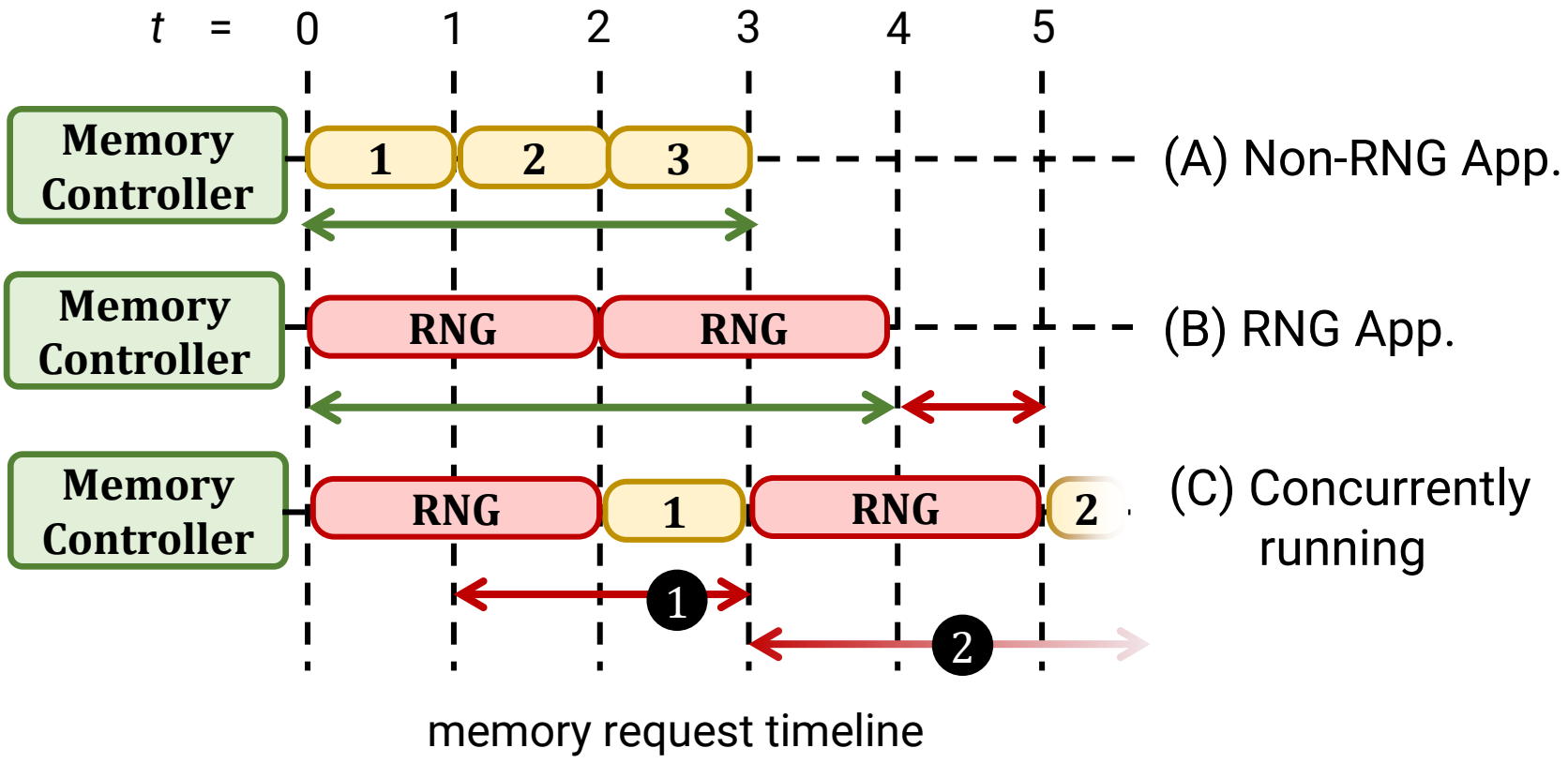
1. RNG Interference
significantly slows down concurrently-running applications

2. Unfair Prioritization
degrades overall system fairness

3. High TRNG Latency
degrades RNG applications' performance

RNG Interference

- TRNG in DRAM can be **intrusive** in current systems that use DRAM as main memory and **stall** memory requests



Three Key Challenges

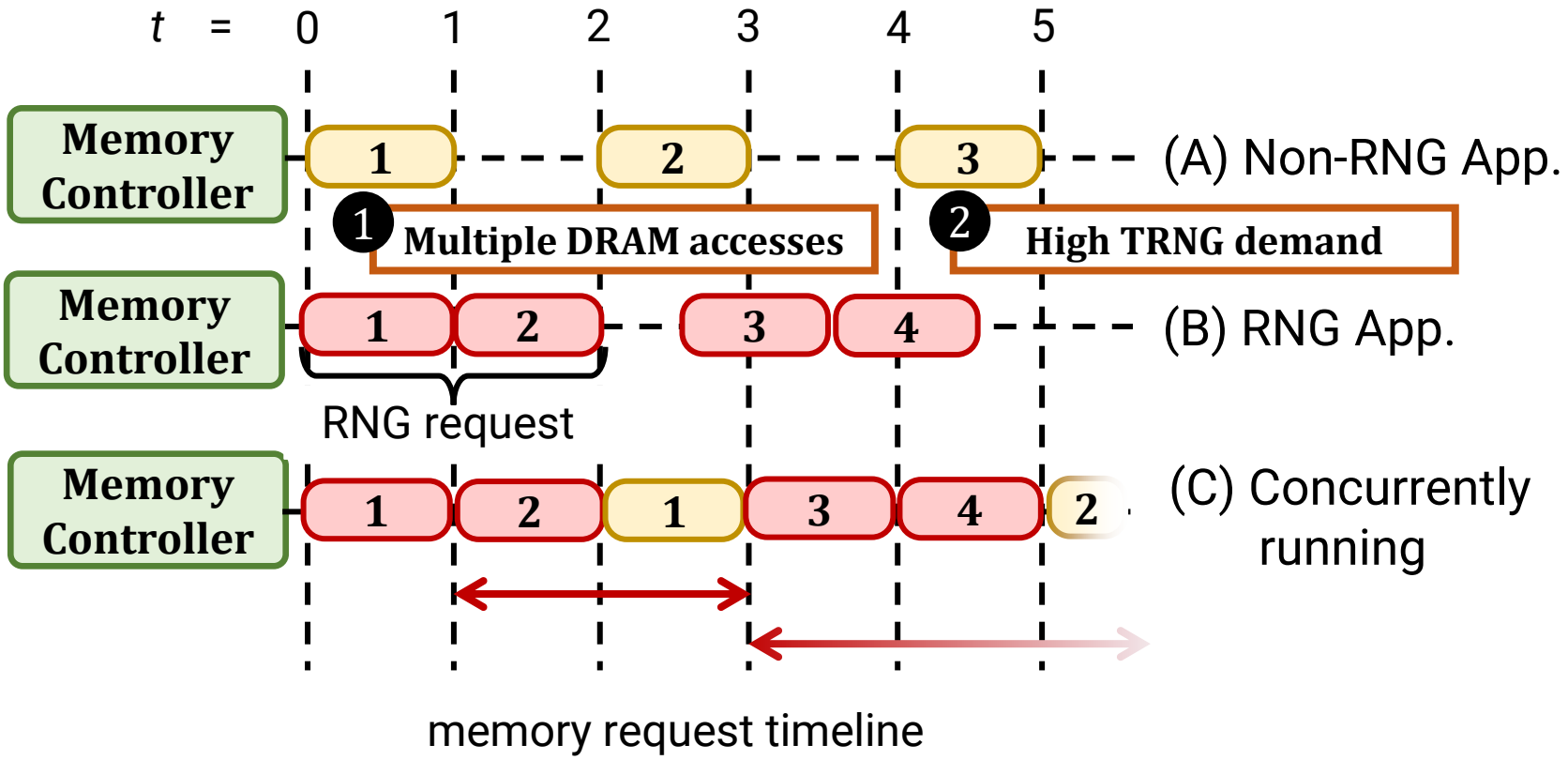
1. **RNG Interference**
   significantly slows down concurrently-running applications

2. **Unfair Prioritization**
   degrades overall system fairness

3. **High TRNG Latency**
   degrades RNG applications' performance

Unfair Prioritization

- Memory request schedulers can prioritize RNG applications to achieve **high throughput**
- Stalls **the non-RNG applications** more and creates **unfairness**



Three Key Challenges

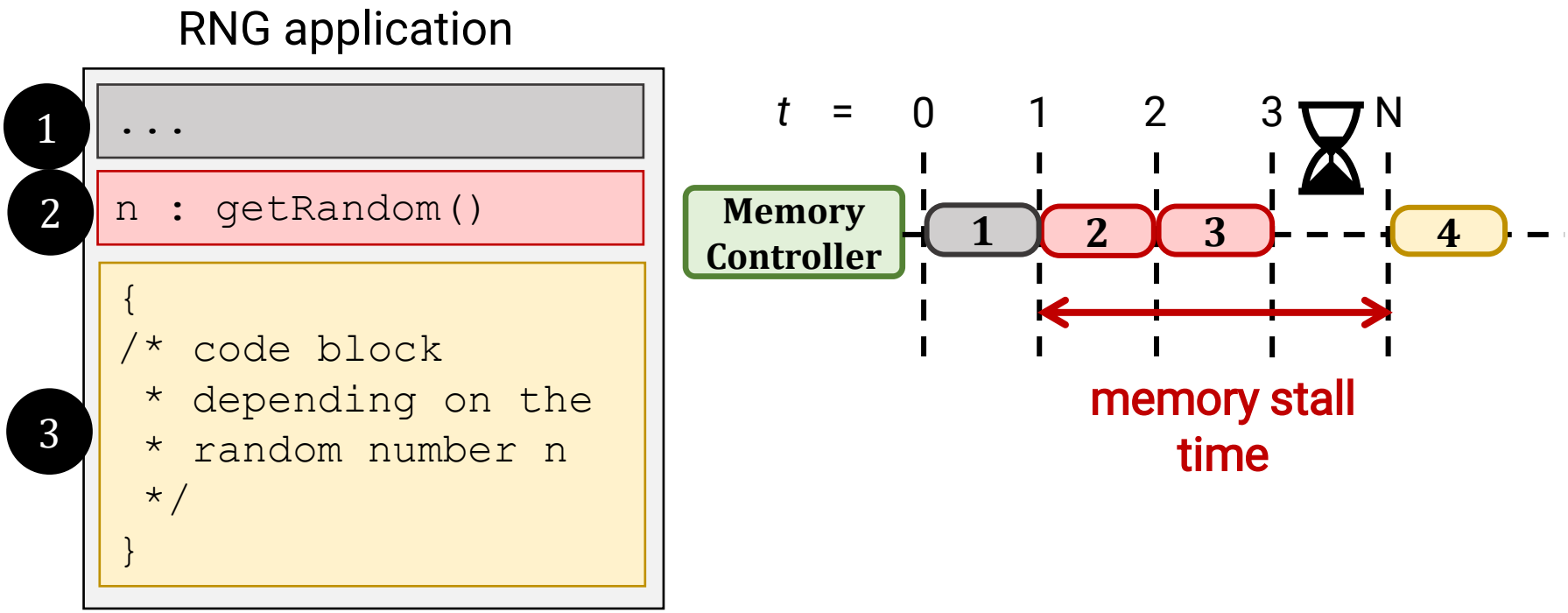
1. **RNG Interference**
   significantly slows down concurrently-running applications

2. **Unfair Prioritization**
   degrades overall system fairness

3. **High TRNG Latency**
   degrades RNG applications' performance

High TRNG Latency

- DRAM-based true random number generation has **high latency** and can **degrade** the performance of applications that use TRNGs



Our Goal

To develop a low-cost and high-performance
end-to-end system design for DRAM-based TRNGs

Outline

- Background
- Motivation and Goal
- DR-STRaNGe
  - Random Number Buffering Mechanism
  - RNG-Aware Scheduler
  - Application Interface
- Evaluation
- Conclusion

DR-STRaNGe: Overview

Random Number Buffering Mechanism

Predicts and utilizes idle DRAM channels to generate random numbers

Stores the generated random numbers in a buffer to be served to upcoming RNG requests

Serves RNG requests with low latency

RNG-Aware Memory Request Scheduler

Accumulates RNG and regular memory requests in separate queues

Schedules requests based on the priority levels set by the OS

Reduces the RNG interference and improves system fairness

Application Interface

Exposes a secure interface to applications that use random numbers

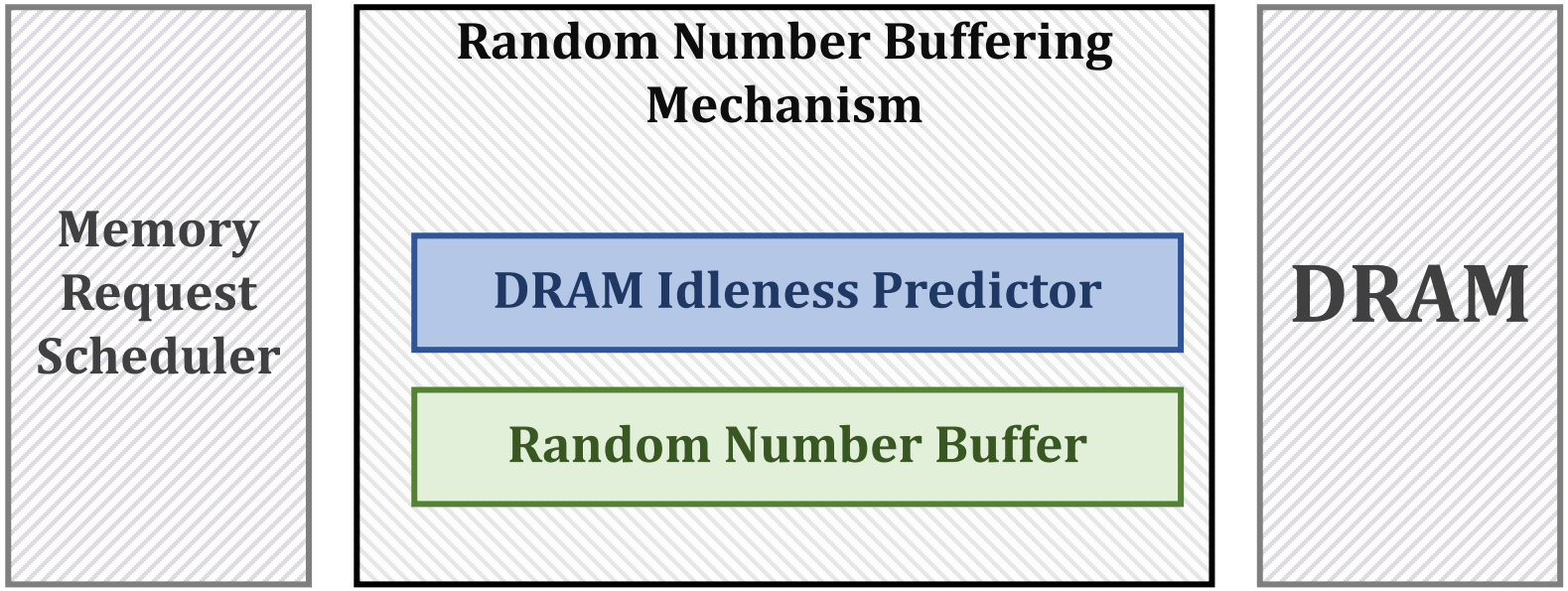
Completes the end-to-end system design and ensures security

Outline

- Background
- Motivation and Goal
- DR-STRaNGe
  - Random Number Buffering Mechanism
  - RNG-Aware Scheduler
  - Application Interface
- Evaluation
- Conclusion

Random Number Buffering Mechanism

- Generates and stores random numbers before they are requested, to be served in the future with **low latency**
- **Predicts** DRAM idleness to generate random numbers with **low overhead** and **stores** them in **a secure buffer**



When to Generate Random Numbers

- Before they are requested by an application
- **Two key metrics** to determine when to generate random numbers:

1. Low DRAM Utilization

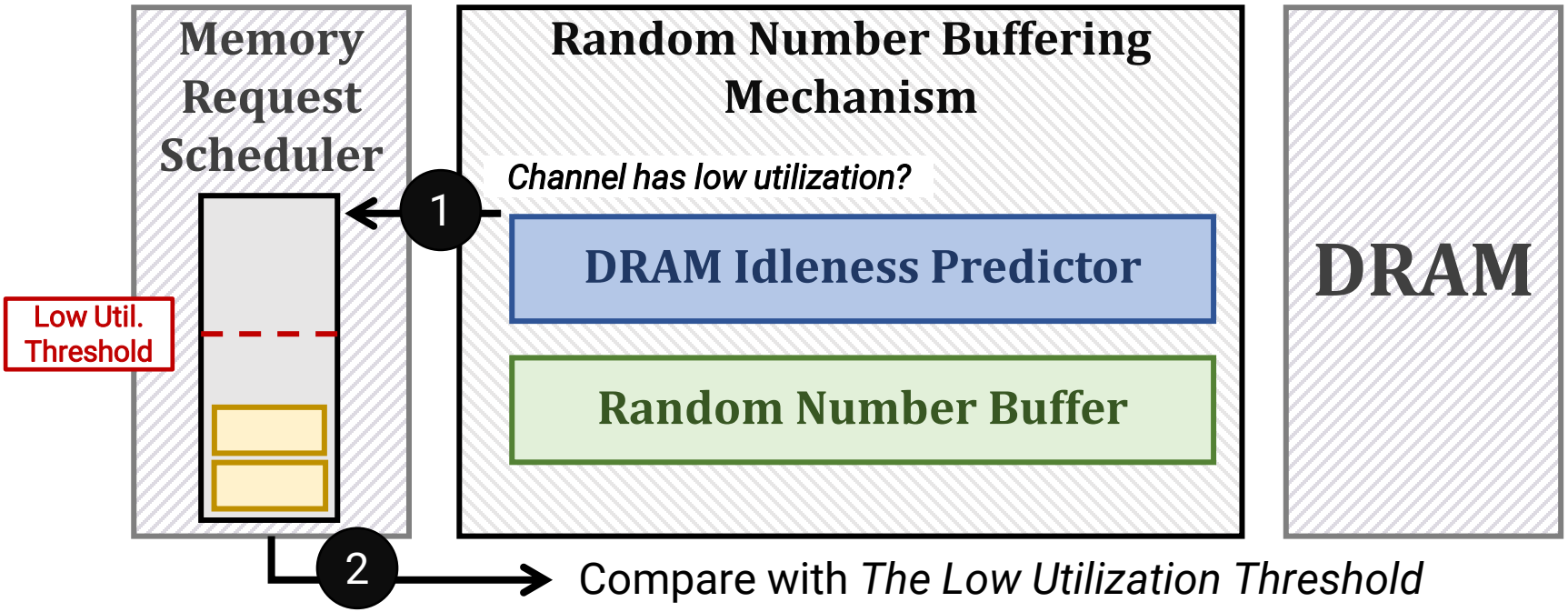
Low channel utilization due to the low rate of memory accesses

2. DRAM Idle Period Length

Number of idle DRAM cycles due to no memory accesses

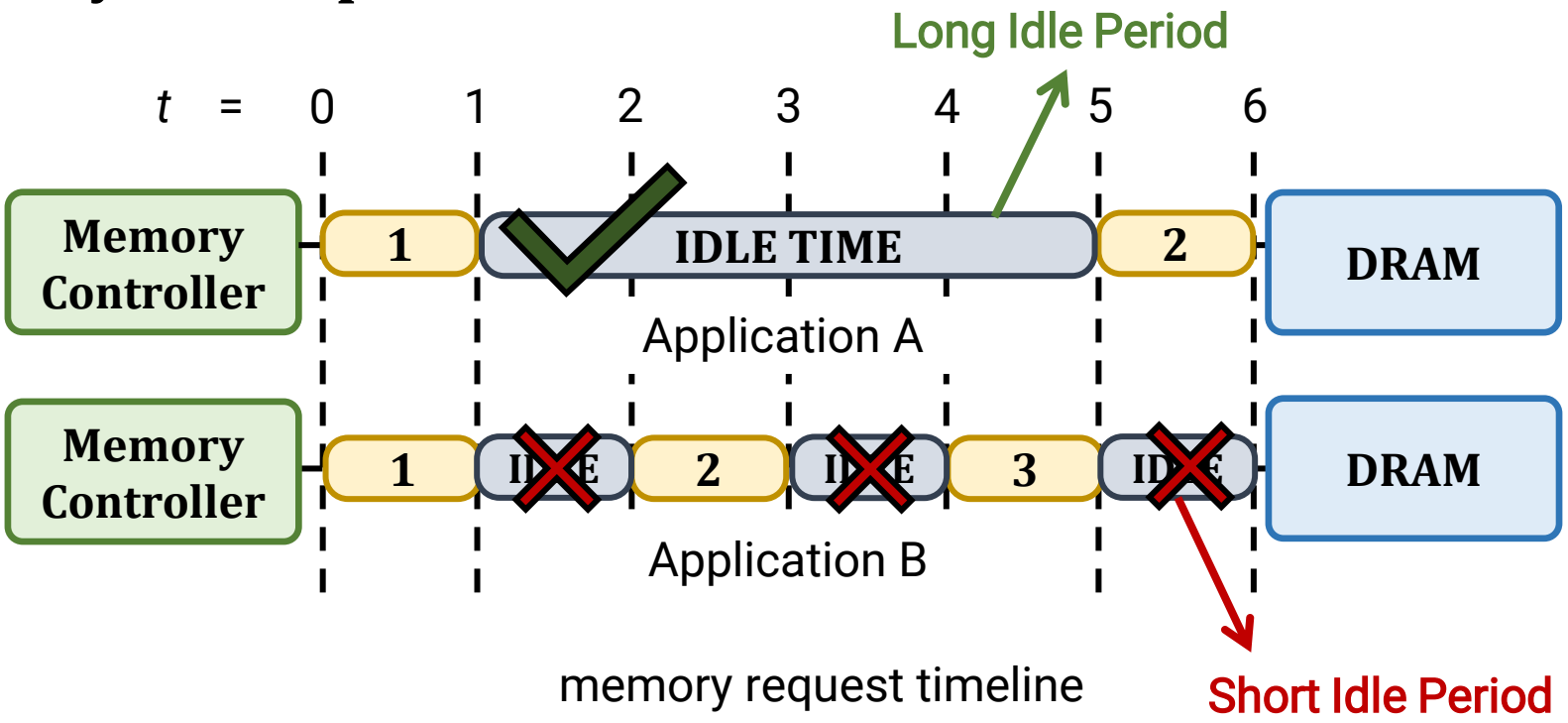
Metric 1: Low DRAM Utilization

- Determine if a channel has **low utilization** based on **the number of queued memory requests** in the memory request scheduler



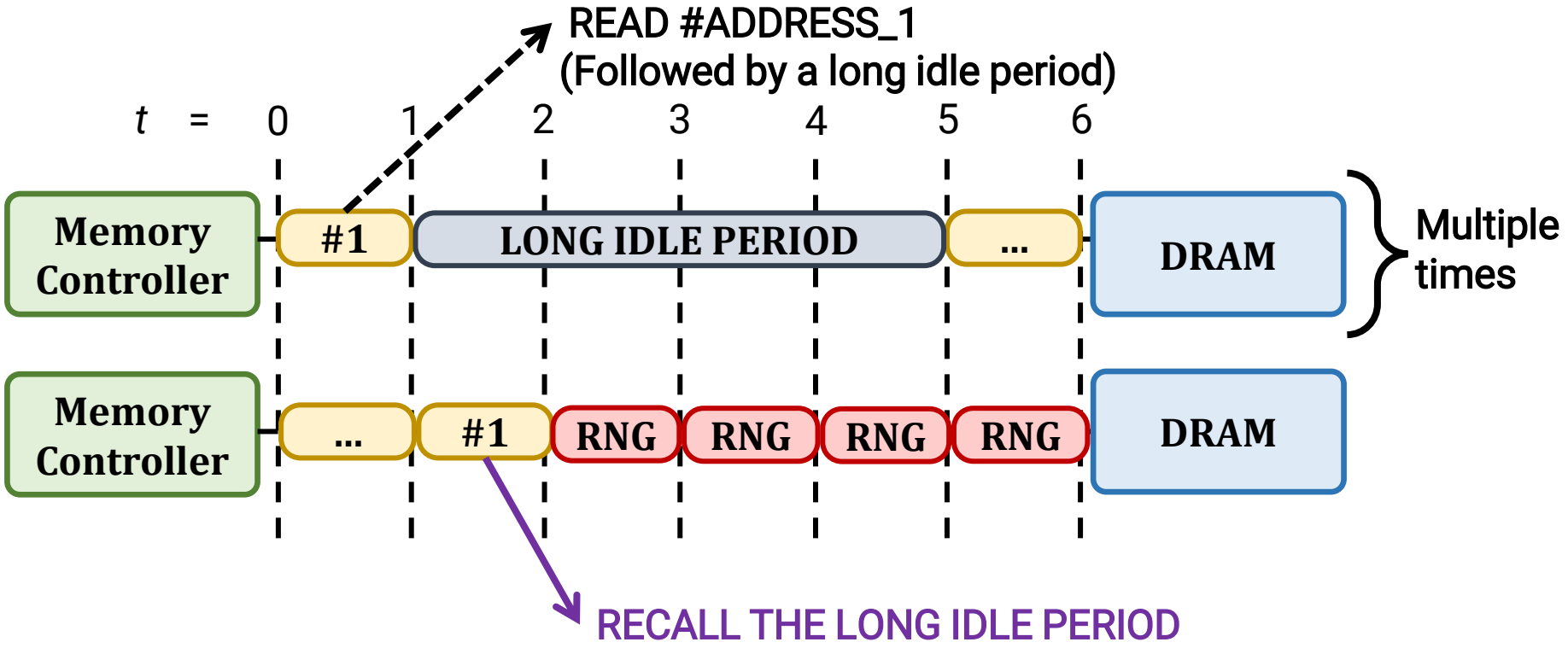
Metric 2: DRAM Idle Period Length

- Applications often do not fully utilize the DRAM bandwidth and this creates **idle periods**
- Idle period lengths **differ** across applications based on **the memory access pattern**



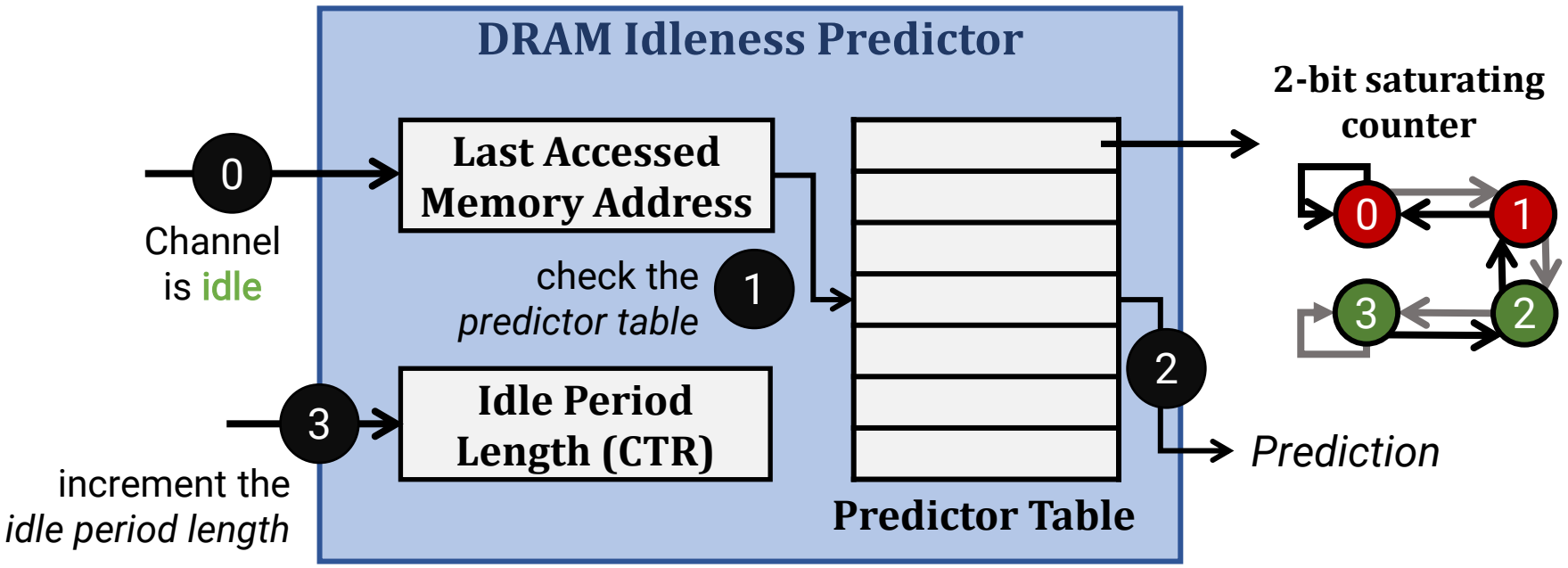
DRAM Idleness Predictor

- **Key Idea:** Use **the last accessed memory address** to **predict the length of the idle period**



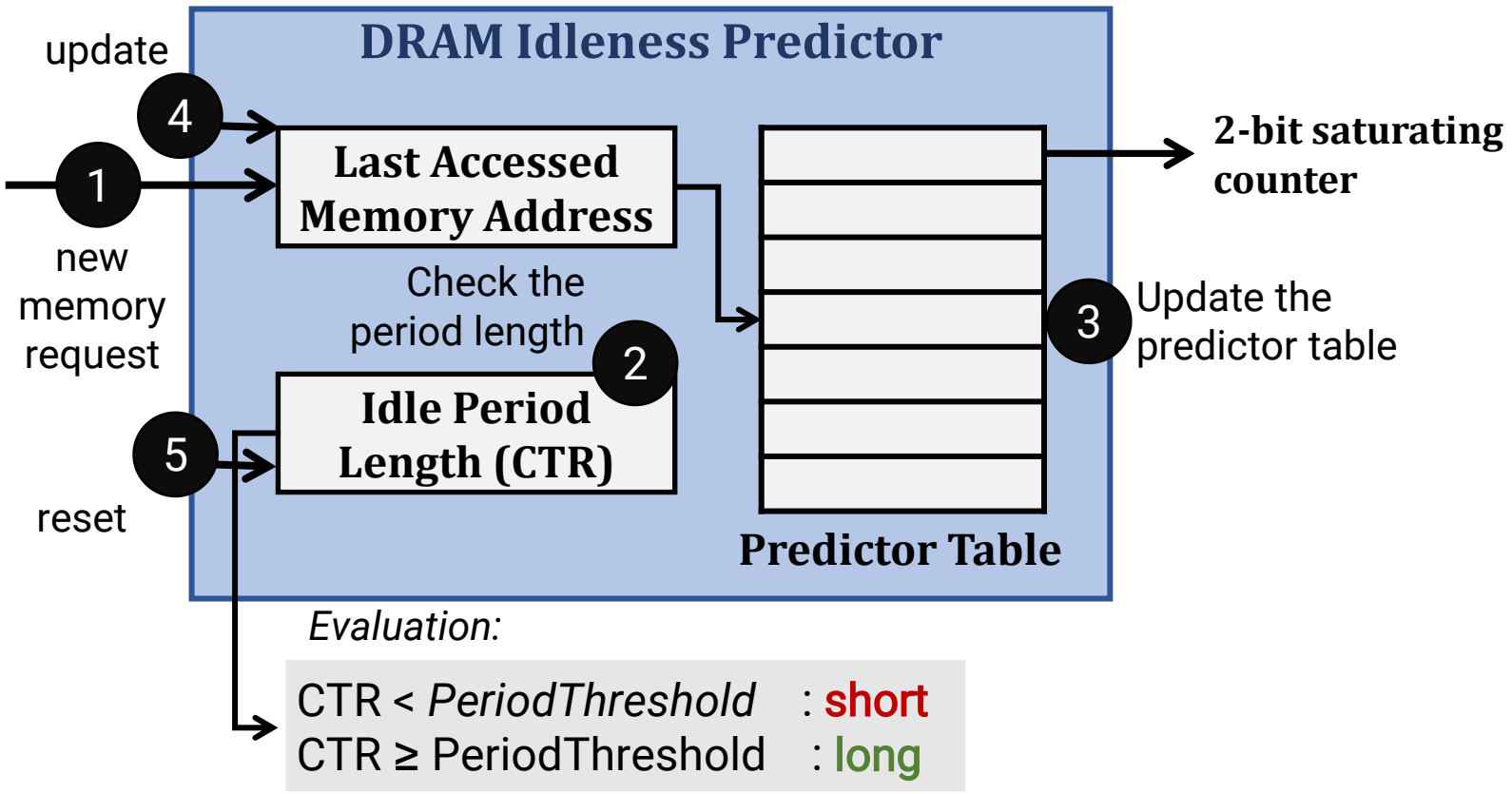
DRAM Idleness Predictor

- **Key Idea:** Use **the last accessed memory address** to **predict the length of the idle period**

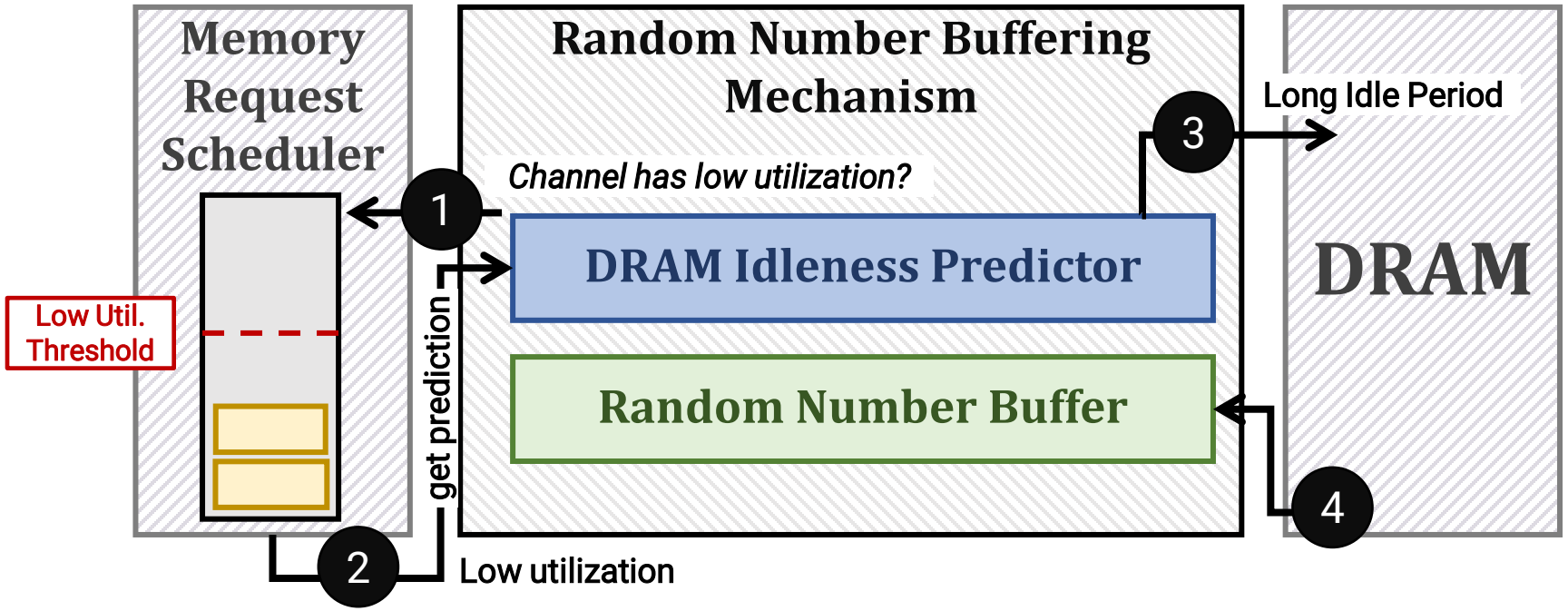


DRAM Idleness Predictor

- **Key Idea:** Use **the last accessed memory address** to **predict the length of the idle period**



Random Number Buffering Mechanism



Generates random numbers **only** if **DRAM is not fully utilized** and **the idle period is predicted to be a long idle period**

Outline

- Background
- Motivation and Goal
- DR-STRaNGe
  - Random Number Buffering Mechanism
  - RNG-Aware Scheduler
  - Application Interface
- Evaluation
- Conclusion

RNG-Aware Scheduler

- **Goal:** To schedule RNG requests **without significantly stalling regular memory requests**
- Two main issues with prior RNG-oblivious schedulers:

1. RNG requests **block** regular memory requests due to the shared scheduler queue space

2. Memory controller **frequently switches between RNG and regular memory requests** due to the RNG-oblivious scheduling decisions

RNG-Aware Scheduler

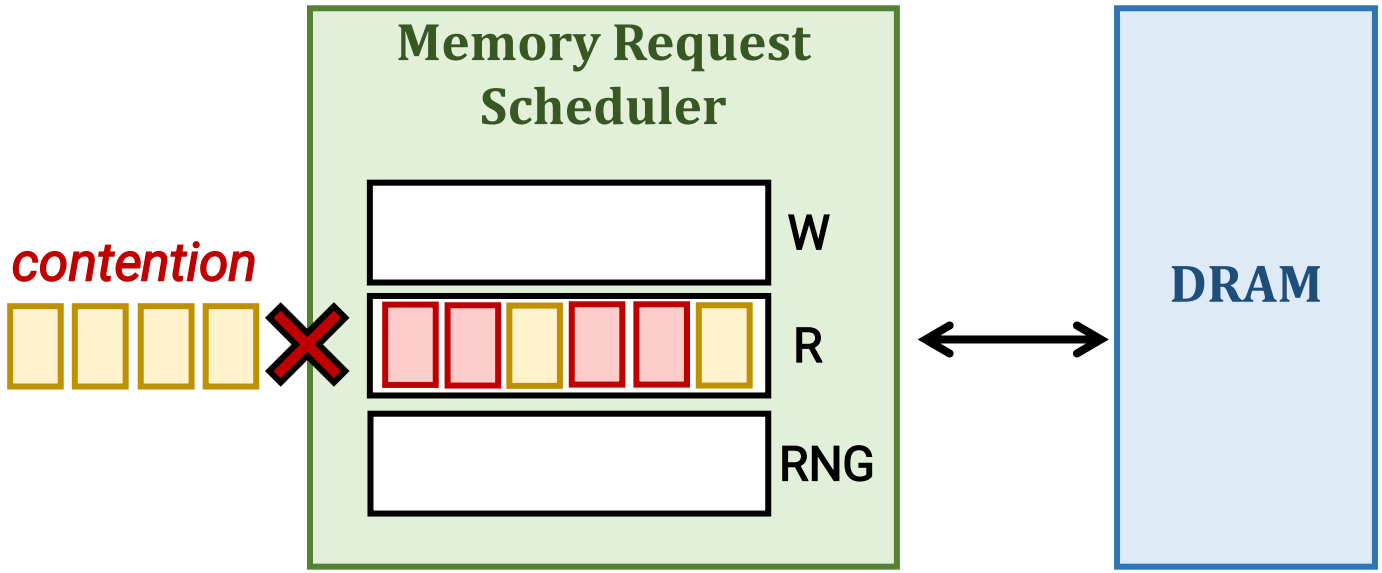
- **Two key ideas:**

1. Accumulate RNG requests in a separate scheduler queue to **reduce contention for queue space**

2. Use **application priority levels** to schedule RNG and regular memory requests

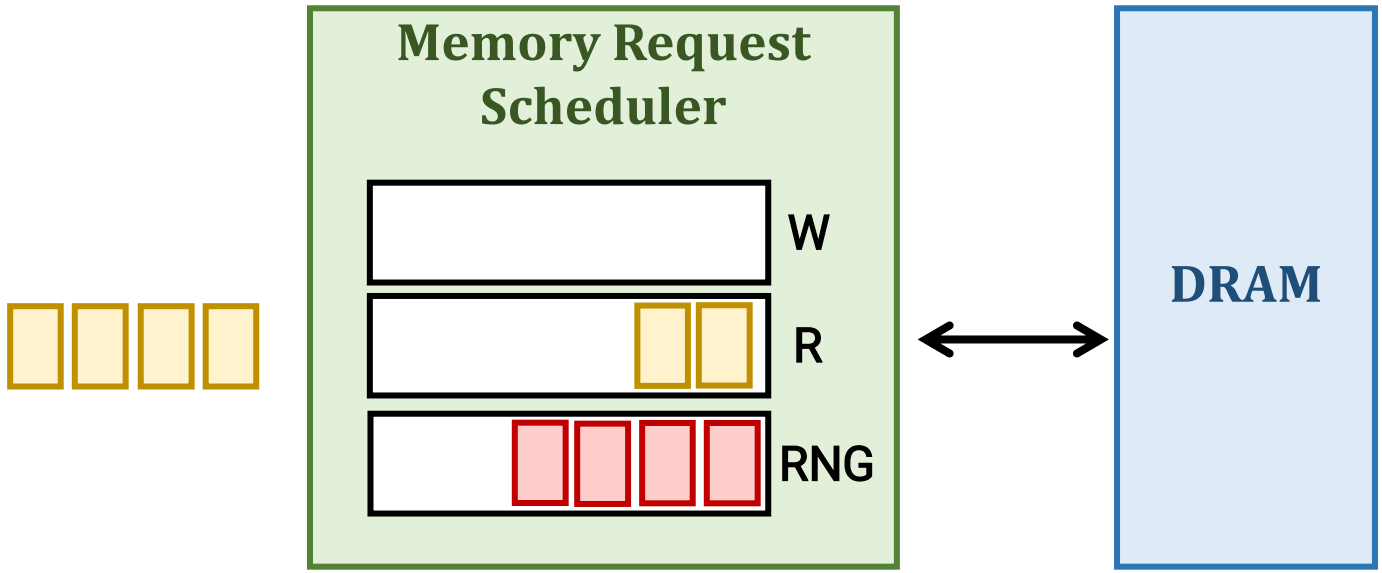
RNG-Aware Scheduler: Key Idea 1

1. Accumulate RNG requests in a separate scheduler queue to **reduce contention for queue space**



RNG-Aware Scheduler: Key Idea 1

1. Accumulate RNG requests in a separate scheduler queue to **reduce contention for queue space**



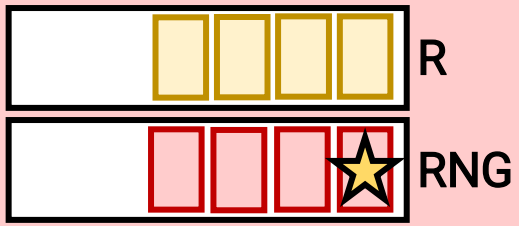
RNG-Aware Scheduler: Key Idea 2

2. Use **application priority levels** to schedule RNG and regular memory requests

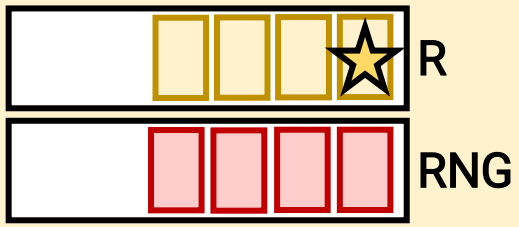
- The operating system manages hardware resources based on priority levels of applications
- RNG-Aware scheduler
  - Can use these priority levels
  - Can identify applications that use TRNGs

RNG-Aware Scheduler: Key Idea 2

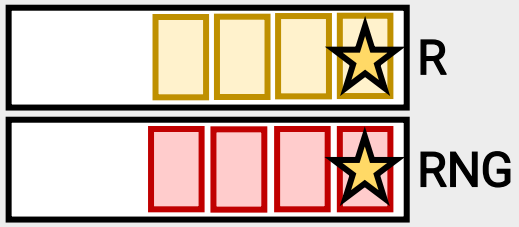
Three possible cases:



RNG Prioritized.
RNG queue is prioritized to minimize the memory stall time of RNG application



Non-RNG Prioritized.
Regular read queue is prioritized to minimize non-RNG application's memory stall time



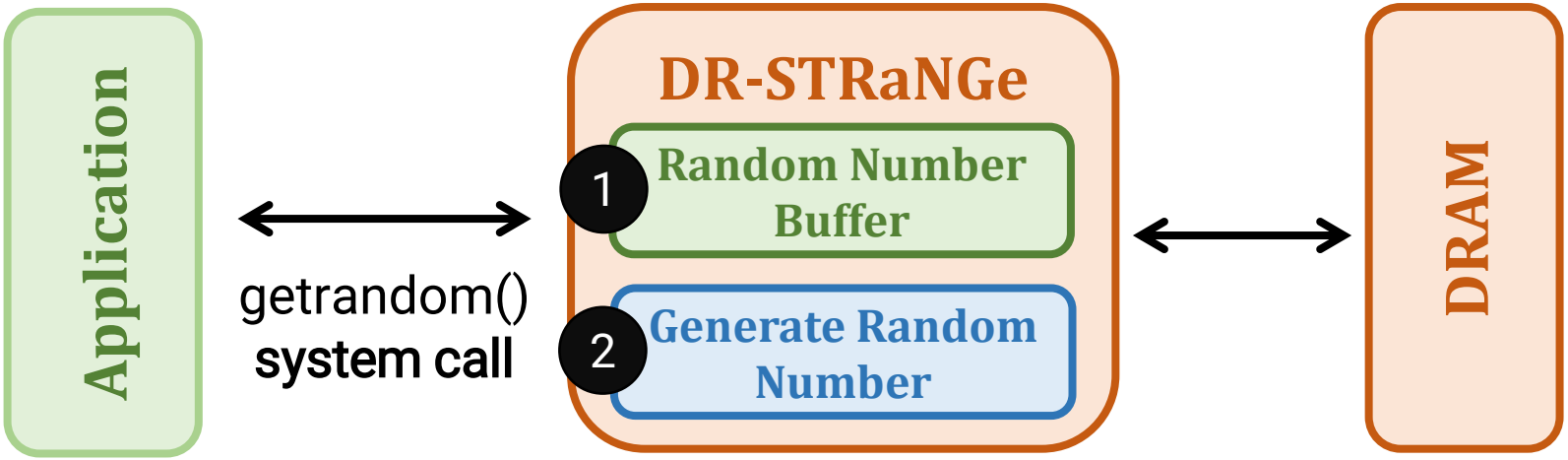
Equal Priorities.
RNG queue is prioritized to quickly serve the RNG requests and minimize the RNG interference

Outline

- Background
- Motivation and Goal
- DR-STRaNGe
  - Random Number Buffering Mechanism
  - RNG-Aware Scheduler
  - Application Interface
- Evaluation
- Conclusion

Application Interface

- Applications use a system call to request a random number
- DR-STRaNGe serves the request **from the random number buffer with low latency** if enough random bits are present
- DR-STRaNGe **generates** random numbers **with low RNG interference** and serve the request otherwise



Outline

- Background
- Motivation and Goal
- DR-STRaNGe
  - Random Number Buffering Mechanism
  - RNG-Aware Scheduler
  - Application Interface
- **Evaluation**
- Conclusion

Evaluation

- Performance, fairness, energy efficiency, and area overhead
- Cycle-level simulations using **Ramulator** [Kim+, CAL'16] and **DRAMPower** [Chandrasekar+]
- **System configuration:**

Processor	1-,2-,4-,8-,16-core, 4 GHz clock frequency, 3-wide issue, 128-entry instruction window
DRAM	DDR3-1600, 800Mhz bus frequency, 4 channels, 1 rank/channel, 8 banks/rank, 64K rows/bank
Memory Controller	32-entry read/write queues, FR-FCFS with a column cap of 16

- **D-RaNGe** [Kim+, HPCA'19], and **QUAC-TRNG** [Olgun+, ISCA'21]

Evaluation

- 43 single-core applications from **four benchmark suites:**
  - SPEC CPU2006, TPC, MediaBench, YCSB
- Synthetic RNG benchmarks with varying **required TRNG throughputs**
  - 640 *Mb/s*, 1280 *Mb/s*, 2560 *Mb/s*, 5120 *Mb/s*
- **Multi-core workloads**

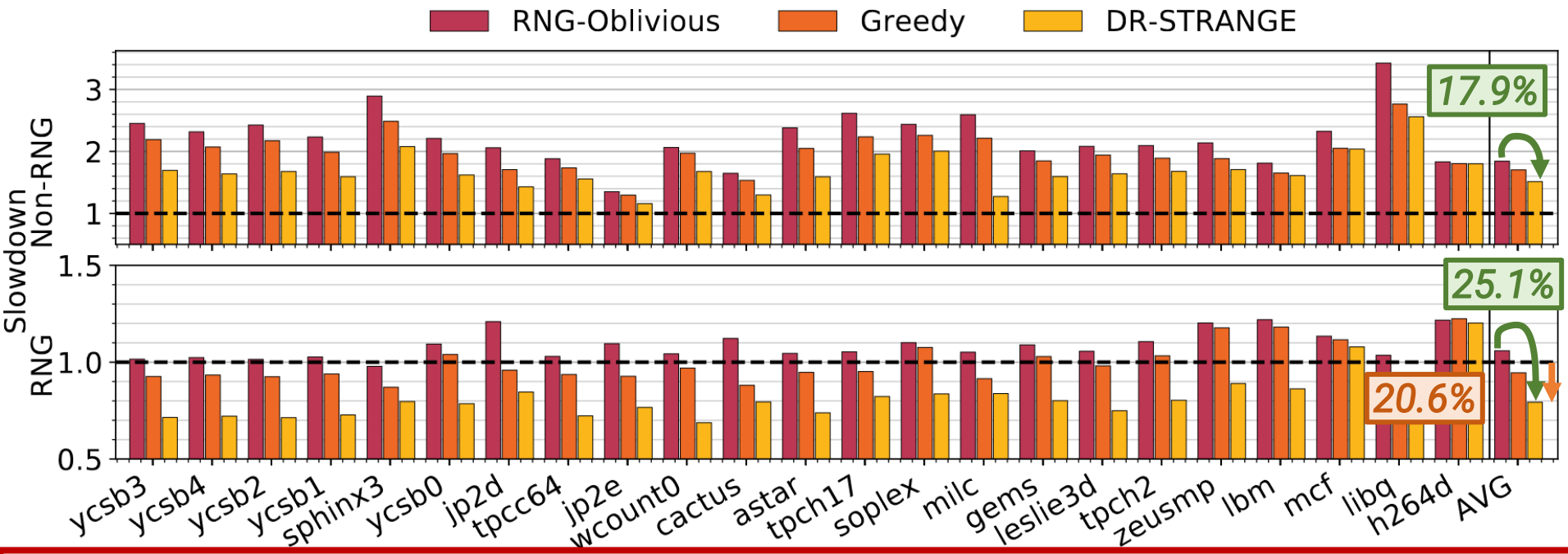
Number of Cores	Non-RNG Applications/Workloads	RNG Applications
2-core	43 non-RNG applications	x1 RNG application with 5120 *Mb/s* RNG throughput requirement x1 RNG application with 640 *Mb/s* RNG throughput requirement
4-core	40 workloads consisting of non-RNG applications (4 memory-intensity groups, 10 workloads each.)	x1 RNG application with 5120 *Mb/s* RNG throughput requirement
8-core	30 workloads consisting of non-RNG applications (3 memory-intensity groups, 10 workloads each.)	x1 RNG application with 5120 *Mb/s* RNG throughput requirement
16-core	30 workloads consisting of non-RNG applications (3 memory-intensity groups, 10 workloads each.)	x1 RNG application with 5120 *Mb/s* RNG throughput requirement

Comparison Points

- **System-level Comparison Points:**
  - **RNG-Oblivious Baseline Design**
  - **Greedy Idle Design**
    - **Perfect Idleness Predictor:** predicts idle period lengths with **100% accuracy**
    - Generates random numbers for the random number buffer **without any overhead**
    - Uses RNG-Aware scheduling

- **Memory Request Scheduler Comparison Points:**
  - FR-FCFS + Column cap of 16 [Mutlu+, MICRO'07]
  - BLISS [Subramanian+, ICCD'14]

System Performance (Dual-Core)



DR-STRaNGe improves the performance of both **non-RNG** and **RNG** applications

DR-STRaNGe **outperforms** both baseline designs by leveraging the **idle** and **low utilization periods** to generate random numbers

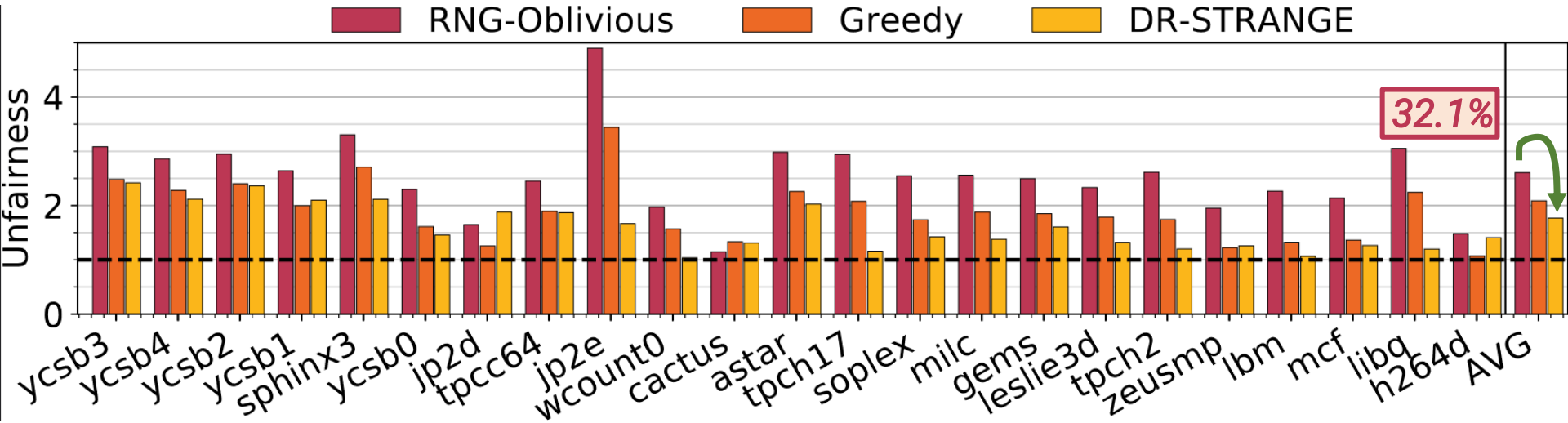
System Performance (Multi-Core)

- 4-, 8-, 16-core evaluation of system performance

DR-STRaNGe **outperforms** both baseline designs significantly

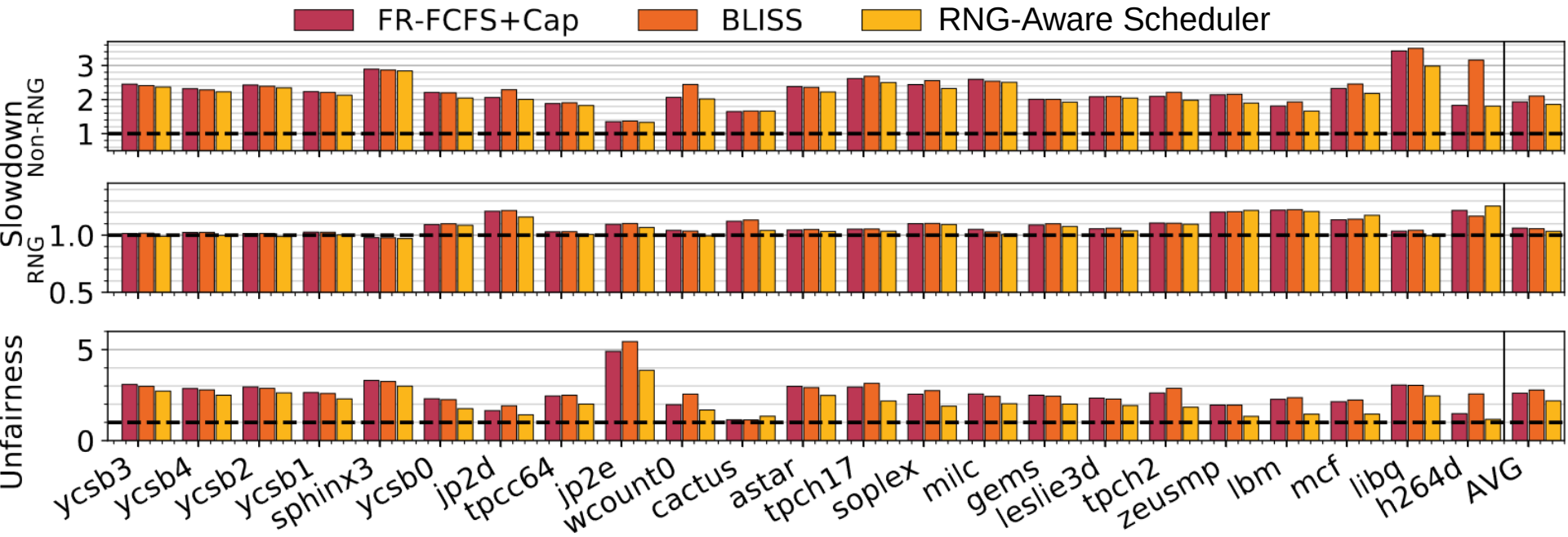
DR-STRaNGe's performance improvement **increases** with the number of **memory-intensive** applications in the workload mix

System Fairness



DR-STRaNGe **outperforms both designs**
by **employing an RNG-aware scheduling policy**
and **effectively reducing and controlling the RNG interference**

Impact of Memory Request Scheduling



RNG-Aware Scheduler **outperforms** both
FR-FCFS+Cap [Mutlu+, MICRO'07] and **BLISS [Subramanian+, ICCD'14]**

RNG-Aware Scheduler **improves average fairness by 16.1%**

Area and Energy Analysis

- **Area:**
  - CACTI [Muralimanohar+, HPL Tech. Report'09]
  - DR-STRaNGe incurs **minor area overhead**:
    - 0.0022mm^2 (0.00048% of an Intel Cascade Lake CPU Core)

- **Energy:**
  - DRAMPower [Chandrasekar+]
  - DR-STRaNGe **reduces average energy consumption** by 21%

More in the Paper

- Security Analysis of DR-STRaNGe
  - Security of Random Numbers
  - Timing Side-Channel Attacks
  - Covert Channel Attacks
  - Denial of Service Attacks

More in the Paper

- More Results
  - Impact of DRAM Idleness Predictor
    - Comparison to a Q-learning-based RL agent
  - Impact of the Random Number Buffer
  - Impact of Priority-based Scheduling
  - Impact of the Low Utilization Prediction
  - Experiments using QUAC-TRNG [Olgun+, ISCA'21]
  - Results of RNG Applications with Low RNG Demand

More in the Paper

- Security Analysis of DR-STRaNGe
  - Security of Random Numbers
  - Timing Side-Channel Attacks
- Impact of
- Experiments using QUAC-TRNG [Olgun+, ISCA'21]
- Results of RNG Applications with Low RNG Demand

DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators

F. Nisa Bostancı[†§] Ataberk Olgun[†§] Lois Orosa[§] A. Giray Yağlıkçı[§]
Jeremie S. Kim[§] Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]

[†]*TOBB University of Economics and Technology* [§]*ETH Zürich*

https://arxiv.org/abs/2201.01385

Outline

- Background
- Motivation and Goal
- DR-STRaNGe
  - Random Number Buffering Mechanism
  - RNG-Aware Scheduler
  - Application Interface
- Evaluation
- Conclusion

DR-STRaNGe Summary

Motivation:

- Random numbers are important for many applications
- DRAM-based True Random Number Generators (TRNGs) can provide **true random numbers at low cost** on **a wide range** of systems

Problem: There is no end-to-end system design for DRAM-based TRNGs

1. Interference between regular memory requests and RNG requests **significantly slows down** concurrently running applications
2. Unfair prioritization of RNG applications **degrades system fairness**
3. High latency of DRAM-based TRNGs **degrades the RNG applications' performance**

Goal: A **low-cost** and **high-performance** end-to-end system design for DRAM-based TRNGs

DR-STRaNGe: An end-to-end system design for DRAM-based TRNGs that

- **Reduces the interference between regular memory requests and RNG requests** by separating them in the memory controller
- **Improves fairness across applications** with an RNG-aware memory request scheduler
- **Hides the large TRNG latencies** using a random number buffering mechanism combined with a new DRAM idleness predictor

Results: DR-STRaNGe

- Improves the average performance of non-RNG (**17.9%**) and RNG (**25.1%**) applications
- Improves the average system fairness (**32.1%**) when generating random numbers at a 5 Gb/s throughput
- Reduces the average energy consumption (**21%**)

DR-STRaNGe:

End-to-End System Design for DRAM-based True Random Number Generators

F. Nisa Bostancı

Ataberk Olgun Lois Orosa A. Giray Yağlıkçı

Jeremie S. Kim Hasan Hassan Oğuz Ergin Onur Mutlu

SAFARI

ETH zürich

kasırga

TOBB ETÜ
University of Economics & Technology