EasyDRAM

An FPGA-based Infrastructure for Fast and Accurate End-to-End Evaluation of Emerging DRAM Techniques

Oğuzhan Canpolat Ataberk Olgun
David Novo Oğuz Ergin Onur Mutlu

https://arxiv.org/abs/2506.10441

SAFARI

ETH zürich TOBB ETÜ Ekonomi ve Teknoloji Üniversitesi LIRMM جامعة الشارقة UNIVERSITY OF SHARJAH

EasyDRAM Summary

Motivation

- Numerous DRAM techniques improve computing system throughput, reliability, and computing capabilities
  - Evaluating them require invasive changes across the compute stack
- Existing FPGA-based evaluation platforms require hardware design expertise and do not accurately model modern systems

Goal: Develop a configurable framework (EasyDRAM) that allows fast and accurate evaluation of emerging DRAM techniques using real DRAM chips

Key Ideas of EasyDRAM

- Easy to use (C++) programmable memory controller implemented in FPGA
- System state advanced to respect modern clock frequency ratios between processor and DRAM

Case Studies

- In-DRAM row-copy (RowClone) provides 15.0x speedup over CPU-based copy for copy-heavy microbenchmarks
- DRAM access latency reduction (SolarDRAM) improves end-to-end system performance by 2.8% avg. across PolyBench workloads

Talk Outline

I. Background

II. Motivation

III. EasyDRAM

IV. Case Studies

V. Conclusion

Talk Outline

I. **Background**

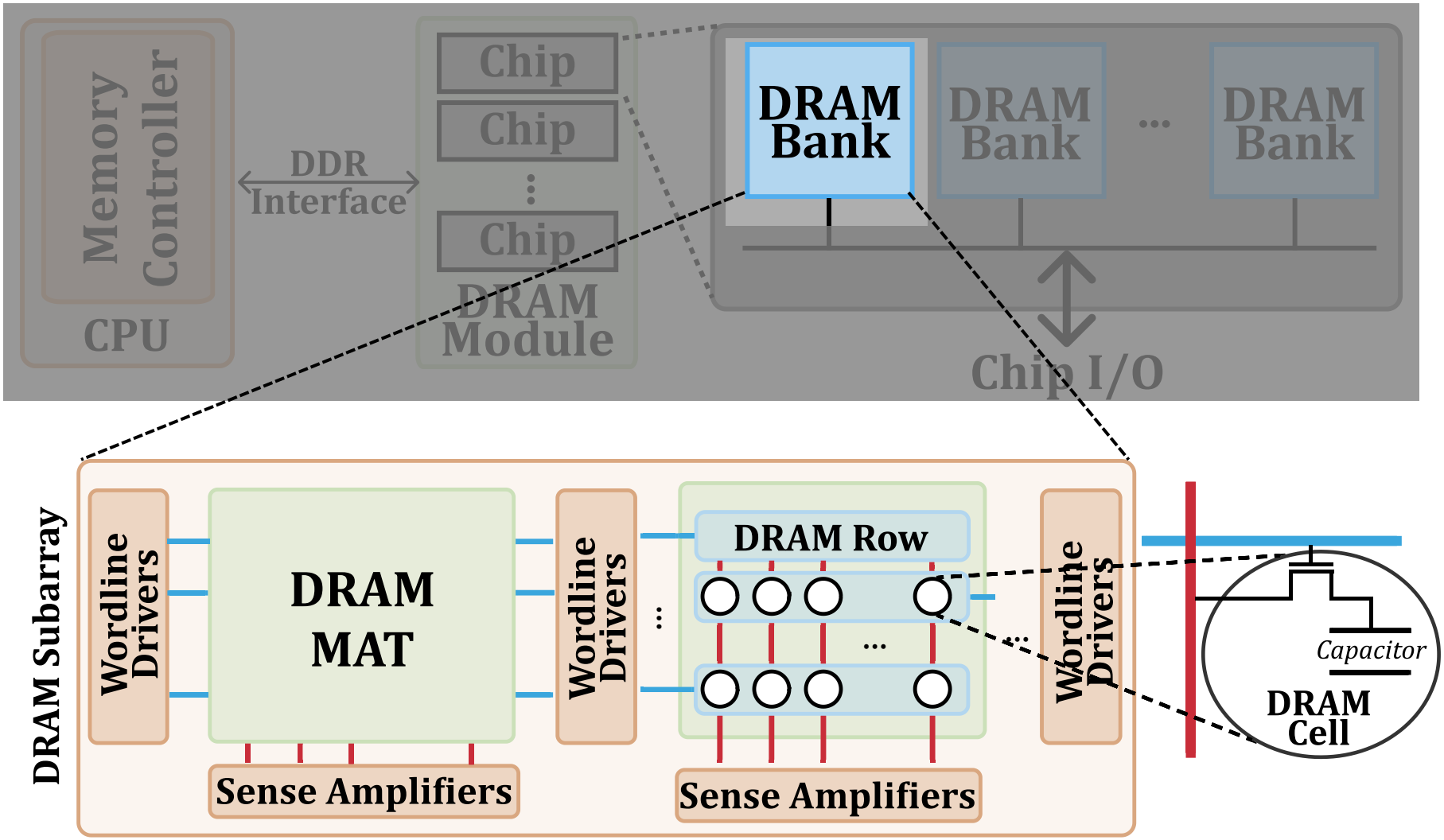
II. Motivation

III. EasyDRAM

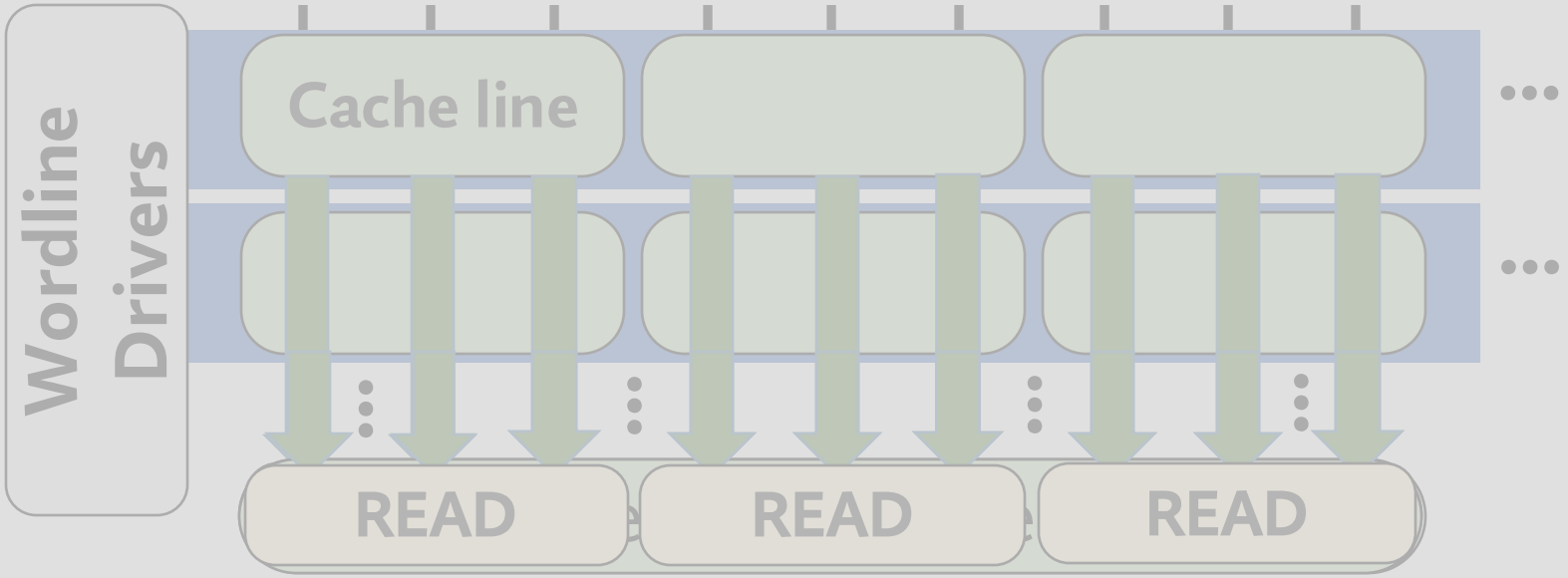
IV. Case Studies

V. Conclusion

DRAM Organization



DRAM Operation



[Kim+ HPCA'19]

DRAM Techniques

- DRAM techniques violate standard timing parameters to improve latency and computing capabilities of DRAM

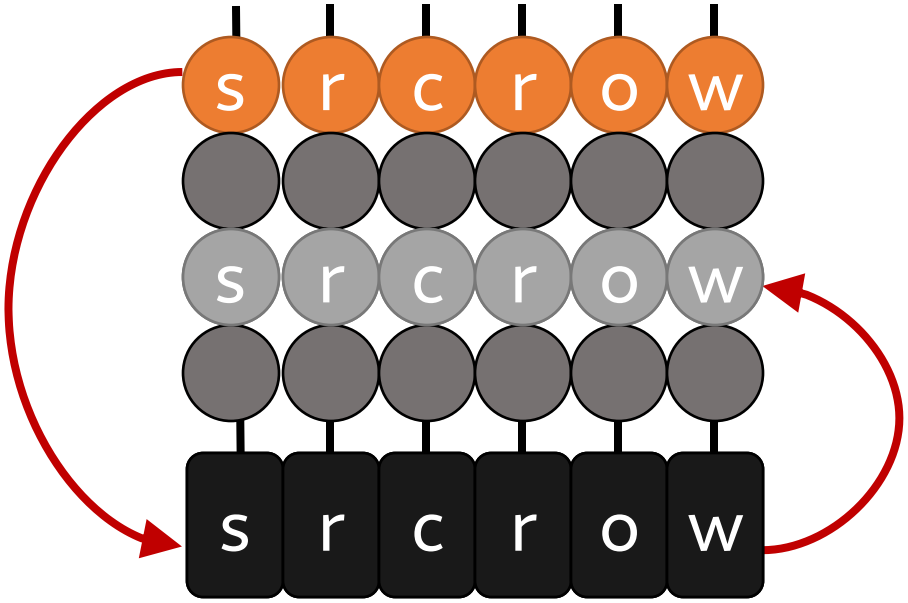
- Many examples
  - In-DRAM bulk data copy and initialization (RowClone [MICRO'13])
  - Bulk bitwise operations (Ambit [MICRO'17])
  - DRAM access latency reduction (e.g., SolarDRAM [ICCD'18])
  - Retention-aware intelligent refresh (e.g., RAIDR [ISCA'12])
  - Random number generation (e.g., QUAC-TRNG [ISCA'21])
  - ...

DRAM Techniques

- DRAM techniques violate standard timing parameters to improve latency and computing capabilities of DRAM

- Many examples
  - In-DRAM bulk data copy and initialization (RowClone [MICRO'13])
  - Bulk bitwise operations (Ambit [MICRO'17])
  - DRAM access latency reduction (e.g., SolarDRAM [ICCD'18])
  - Retention-aware intelligent refresh (e.g., RAIDR [ISCA'12])
  - Random number generation (e.g., QUAC-TRNG [ISCA'21])
  - ...

Row-Copy: Key Idea (RowClone)



Sense Amplifiers

1. Source row to sense amplifiers

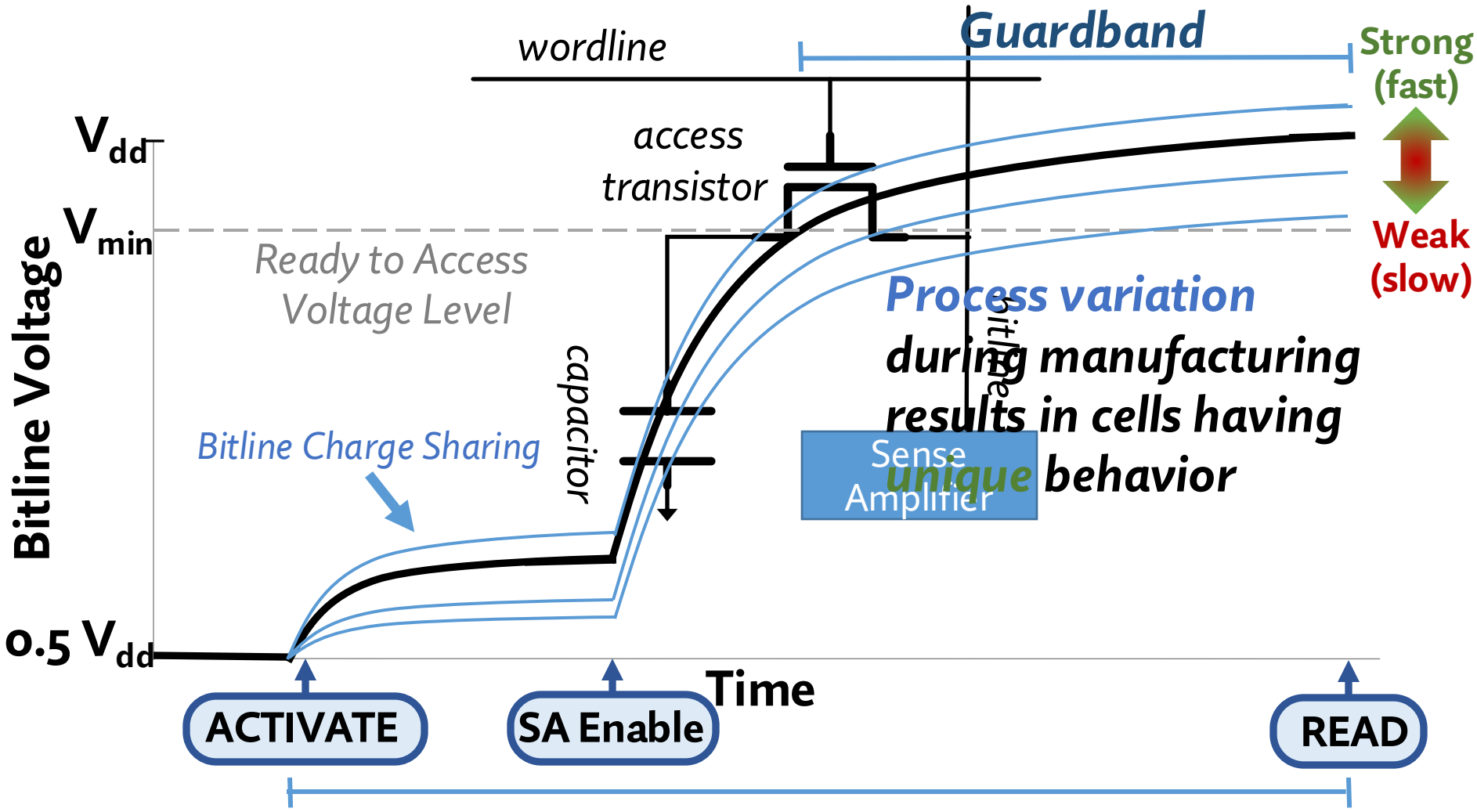
2. Sense amplifiers to destination row

DRAM Techniques

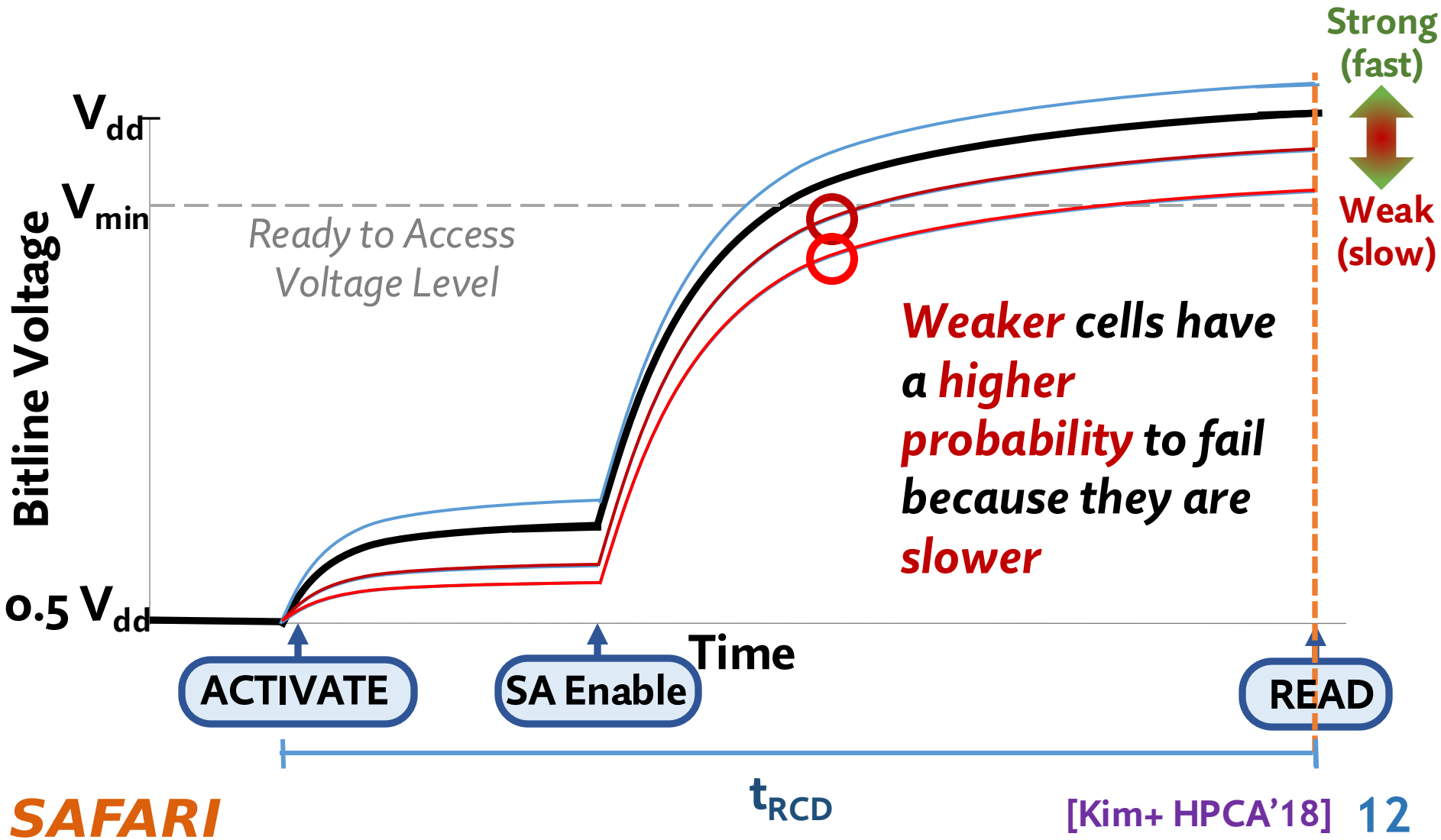
- DRAM techniques violate standard timing parameters to improve latency and computing capabilities of DRAM

- Many examples
  - In-DRAM bulk data copy and initialization (RowClone [MICRO'13])
  - Bulk bitwise operations (Ambit [MICRO'17])
  - DRAM access latency reduction (e.g., SolarDRAM [ICCD'18])
  - Retention-aware intelligent refresh (e.g., RAIDR [ISCA'12])
  - Random number generation (e.g., QUAC-TRNG [ISCA'21])
  - ...

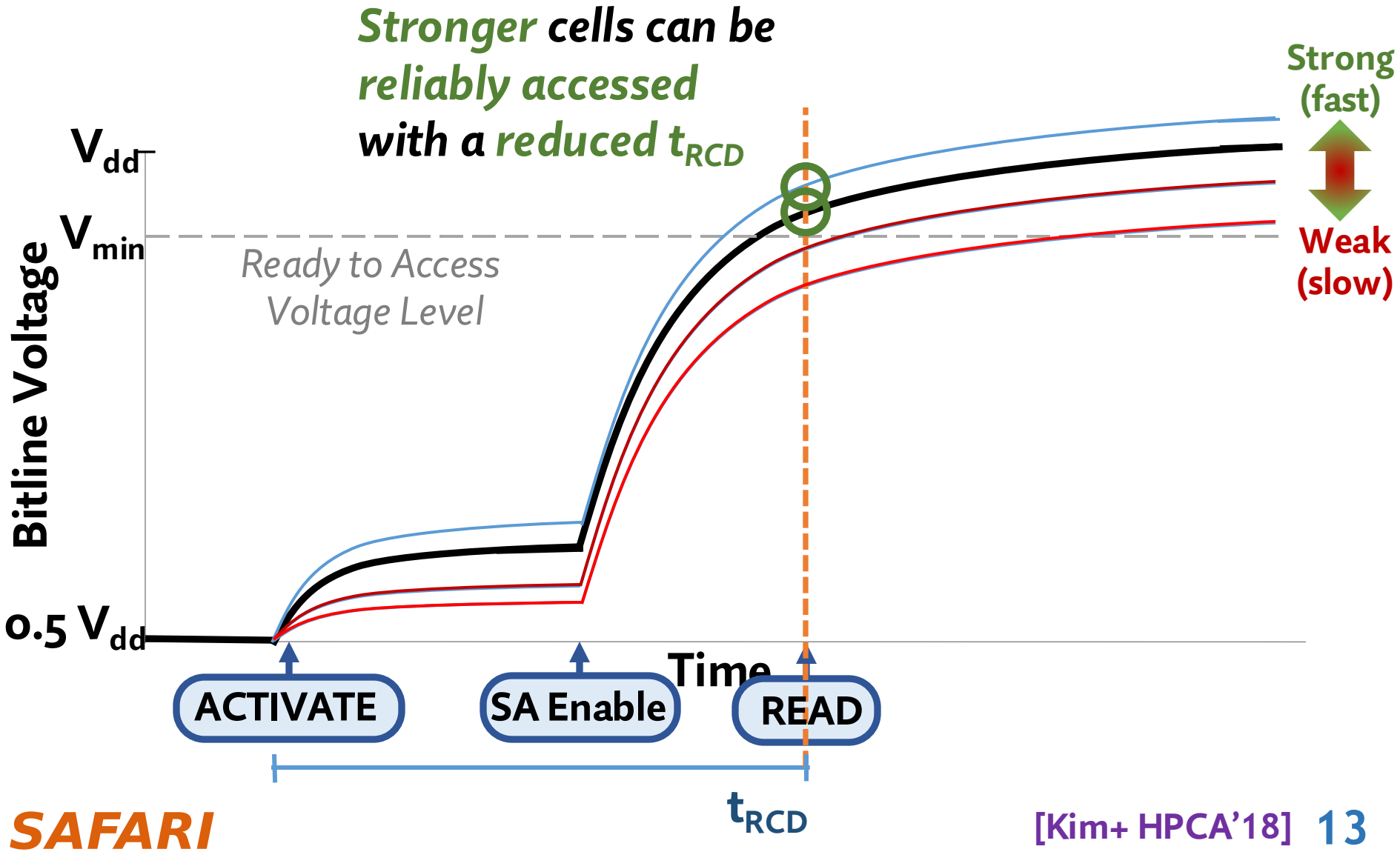
DRAM Accesses and Failures



DRAM Accesses and Failures



DRAM Access Latency Reduction: Key Idea



Talk Outline

I. Background

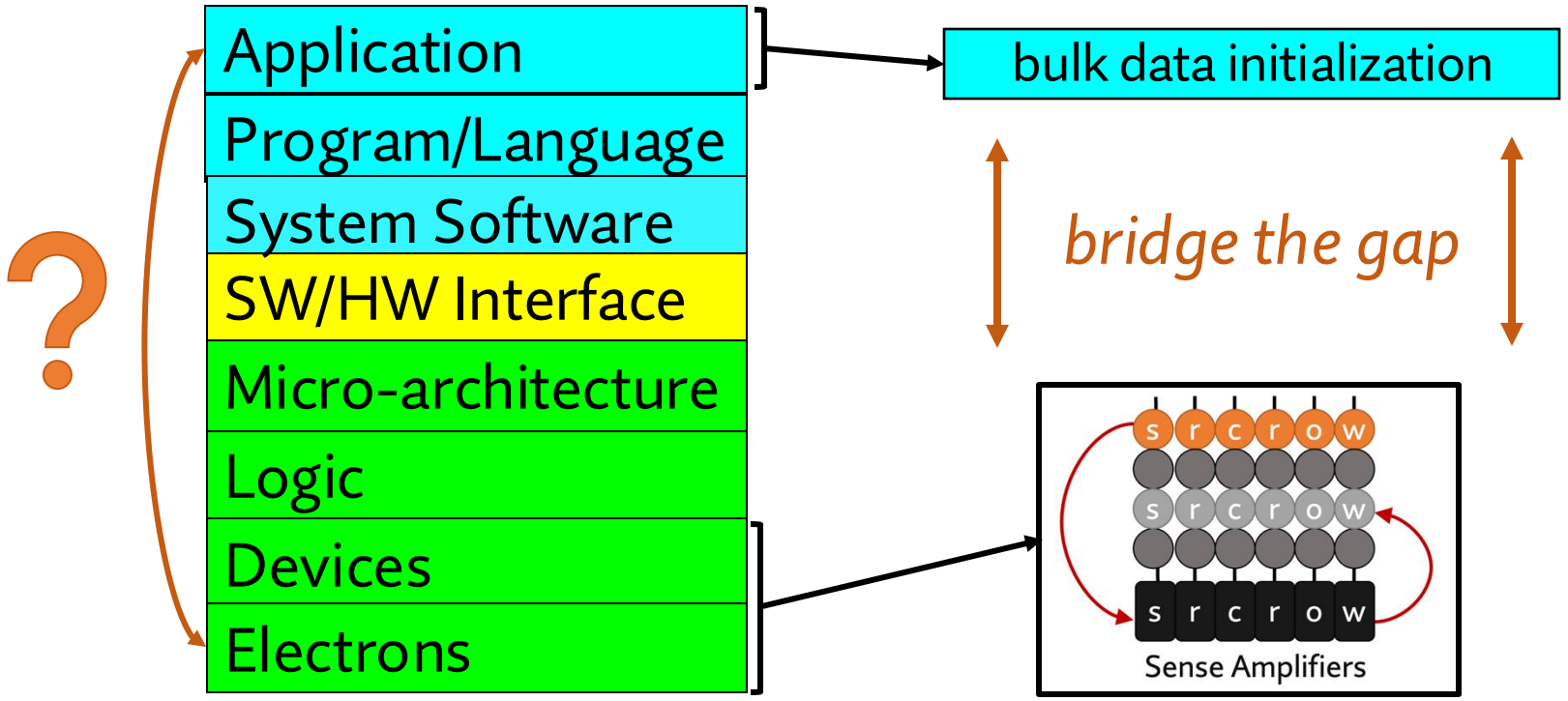
II. Motivation

III. EasyDRAM

IV. Case Studies

V. Conclusion

System Support for DRAM Techniques



Evaluating the benefits of DRAM techniques requires modifications across the stack

FPGA-Based Evaluation Platforms

- Good basis for evaluating DRAM techniques

Interface with real DRAM chips

Relatively short end-to-end execution time

Requires hardware design expertise

Disproportionally low CPU clock frequency

https://arxiv.org/pdf/2506.10441

FPGA-Based Evaluation Platforms

- Good basis for evaluating DRAM techniques

Interface with real DRAM chips

Relatively short end-to-end execution time

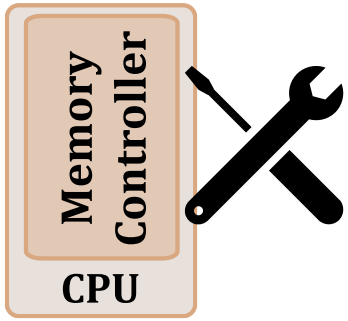
Requires hardware design expertise

Disproportionally low CPU clock frequency

https://arxiv.org/pdf/2506.10441

Hardware Design Expertise Requirement

- Implementing DRAM techniques requires memory controller modifications



- Deep hardware design expertise to reason about
  - Design clock frequency
  - Design area and hardware utilization
  - Functional correctness
  - Debugging

Barrier to entry for many researchers and designers

FPGA-Based Evaluation Platforms

- Good basis for evaluating DRAM techniques

Interface with real DRAM chips

Relatively short end-to-end execution time

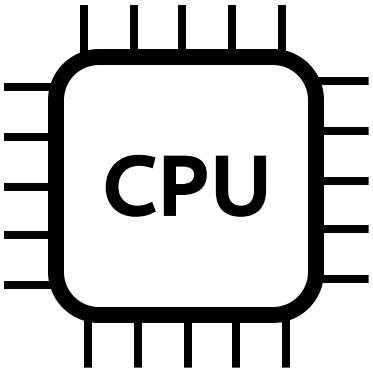
Requires hardware design expertise

Disproportionally low CPU clock frequency

https://arxiv.org/pdf/2506.10441

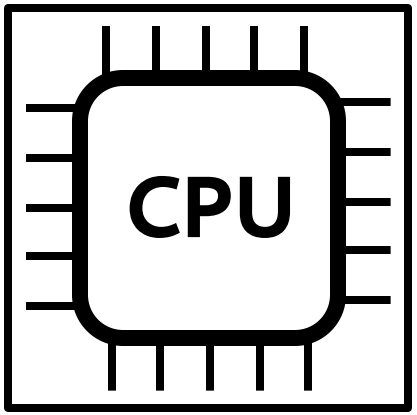
Disproportionately Low CPU Clock Frequency

Clock frequency
4~6 GHz



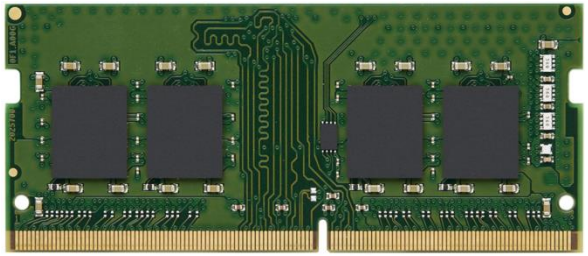
Integrated Circuit

50~200 MHz



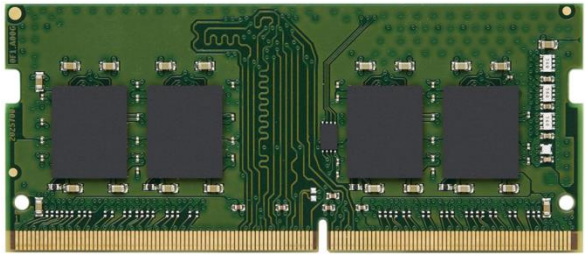
FPGA Implementation

1~4 GHz



DRAM

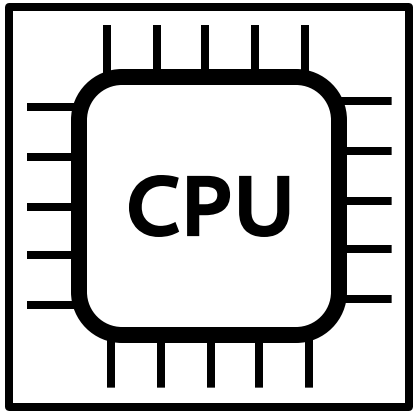
1~4 GHz



DRAM

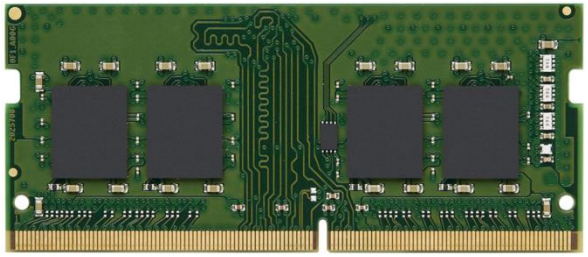
The Clock Frequency is Discrepant

50~200 MHz



FPGA Implementation

1~4 GHz



DRAM

Limits evaluation accuracy and skews the results

Problem

FPGA-based platforms

are difficult to modify

&

yield inaccurate system performance results

when used for DRAM technique evaluation

Our Goal

Enable rapid and accurate end-to-end evaluation of DRAM techniques

without needing deep hardware design expertise

Talk Outline

I. Background

II. Motivation

III. EasyDRAM

IV. Case Studies

V. Conclusion

EasyDRAM Key Ideas

- Evaluate DRAM techniques using real DRAM chips
  - without requiring deep hardware design expertise
  - with accurate system performance results

1

Programmable memory controller

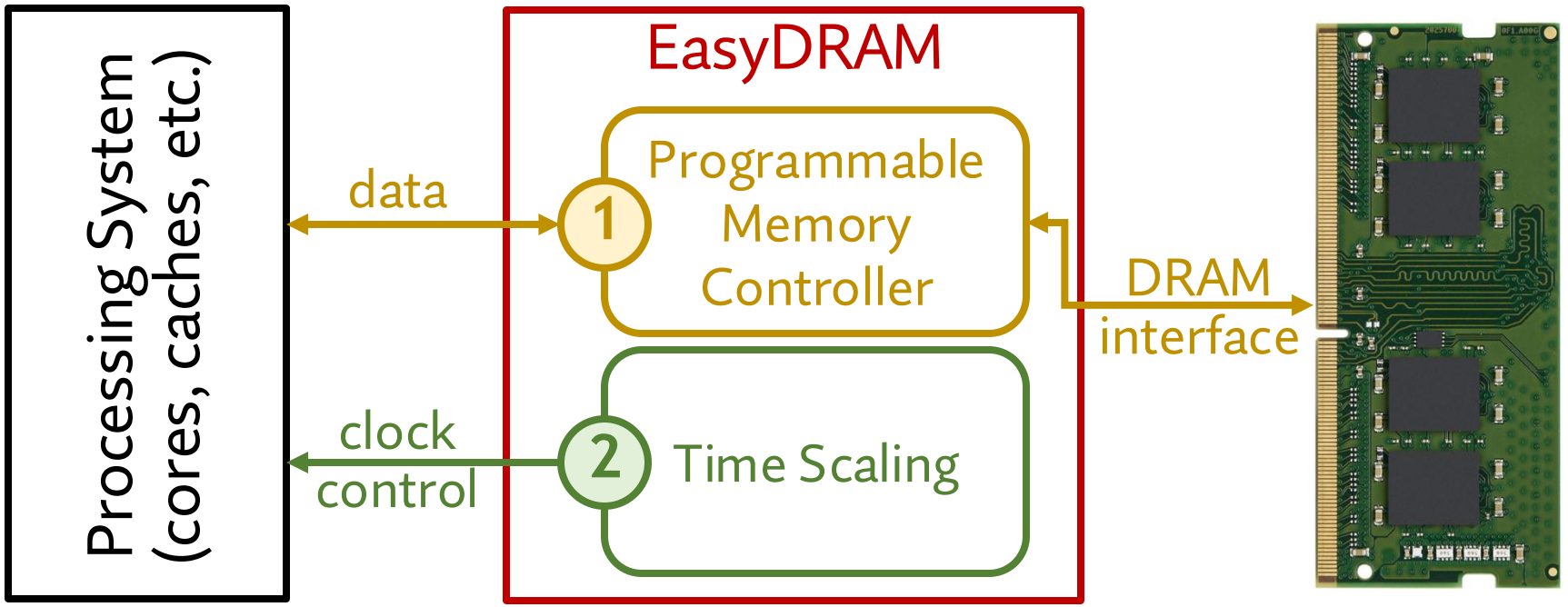
- C++ program defines memory controller
- Program runs on a RISC-V core

2

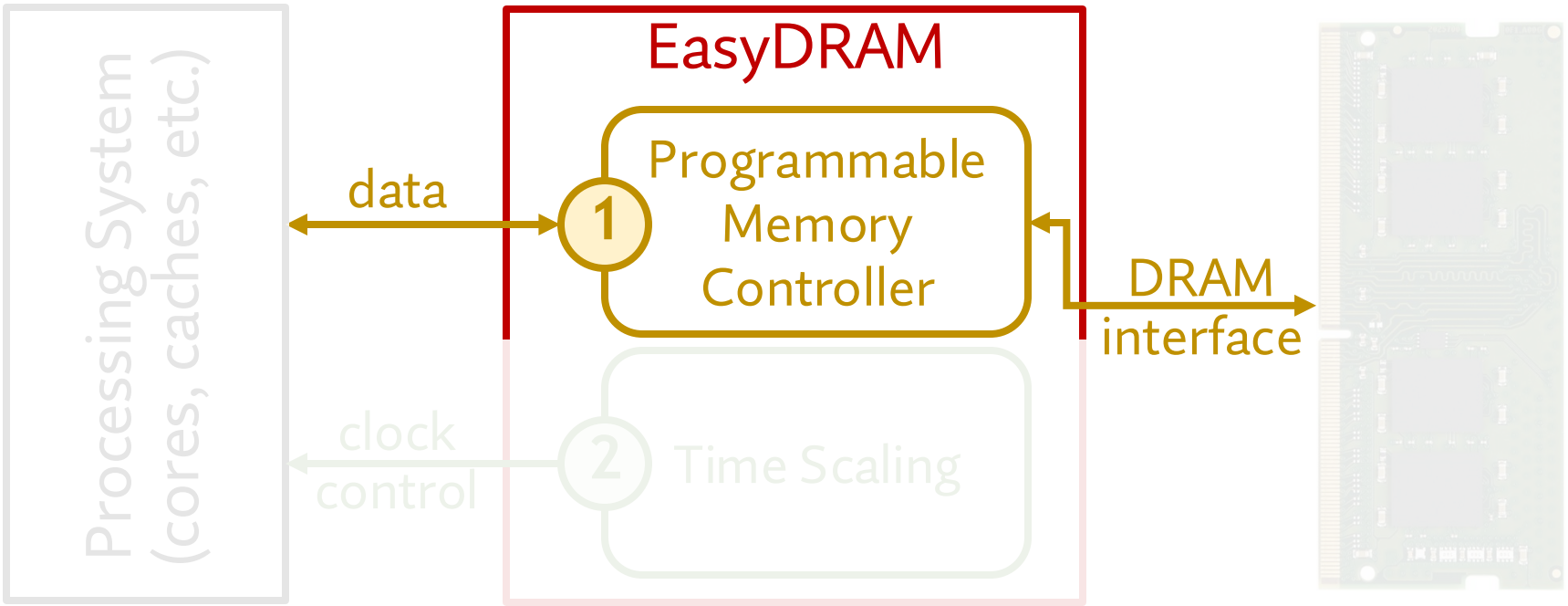
"Time Scaling" technique

- Split hardware components to emulation domains
- Control clock of each domain to match expectations

EasyDRAM Overview



EasyDRAM Overview



Programmable Memory Controller

- Fully-programmable general-purpose processor instead of a conventional RTL memory controller

```
while (request_queue_is_empty());
Request req = receive_memory_request();
auto pa = req.physical_address;
auto dram_address = translate_address(pa);
response = issue_read(dram_address);
send_response(req);
```

C++ Memory Controller Program

Programmable Memory Controller

- Fully-programmable general-purpose processor instead of a conventional RTL memory controller

```
1  while (request_queue_is_empty());
   Request req = receive_memory_request();
   auto pa = req.physical_address;
   auto dram_address = translate_address(pa);
   response = issue_read(dram_address);
   send_response(req);
```

wait for memory request to arrive

Programmable Memory Controller

- Fully-programmable general-purpose processor instead of a conventional RTL memory controller

```
while (request_queue_is_empty());
Request req = receive_memory_request();
auto pa = req.physical_address;
auto dram_address = translate_address(pa);
response = issue_read(dram_address);
send_response(req);
```

② process request to find DRAM address

Programmable Memory Controller

- Fully-programmable general-purpose processor instead of a conventional RTL memory controller

```
while (request_queue_is_empty());
Request req = receive_memory_request();
auto pa = req.physical_address;
auto dram_address = translate_address(pa);
response = issue_read(dram_address);
send_response(req);
```

3

issue request to DRAM and respond

Programmable Memory Controller

```
while (request_queue_is_empty());
Request req = receive_memory_request();
auto pa = req.physical_address;
auto dram_address = translate_address(pa);
response = issue_read(dram_address);
send_response(req);
```

Provides an easy means to perform
low-level memory controller operations

Programmable Memory Controller

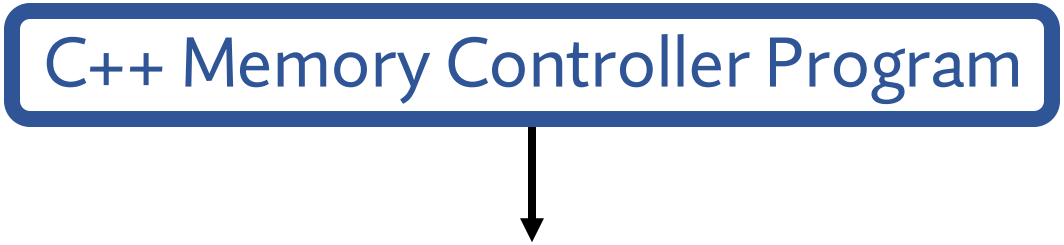
```
while (request_queue_is_empty());
Request req = receive_memory_request();
```

https://arxiv.org/pdf/2506.10441

Function Name	Description
set_scheduling_state(bool state)	Set critical mode register
get_request()	Read a request from the hardware request buffer
ddr_activate()/precharge()/read()...	Insert a DRAM command to the command batch
flush_commands()	Execute the DRAM commands in the command batch
add_request(Request& req)	Insert a memory request to the request table in programmable core memory
FRFCFS::schedule()	Select a request with the FR-FCFS scheduler
FCFS::schedule()	Select a request with the FCFS scheduler
rowclone(Address src, Address dst...)	Insert a RowClone command sequence to the command batch

Provides an easy means to perform
low-level memory controller operations

Using a Programmable Memory Controller in an FPGA-based System is Challenging



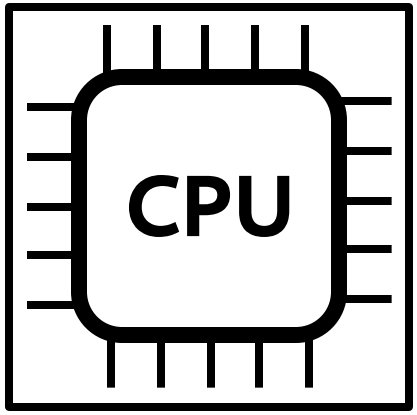
Executes many instructions
to simulate one memory controller clock cycle
(i.e., is slow)

1. Cannot issue DRAM commands at nanosecond granularity

2. The processor experiences lengthier stalls waiting for memory than in a real system

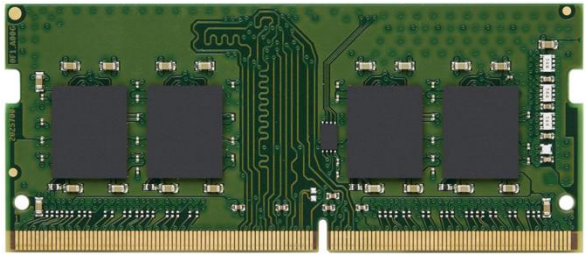
Recall: Clock Frequency Discrepancy

50~200 MHz



FPGA Implementation

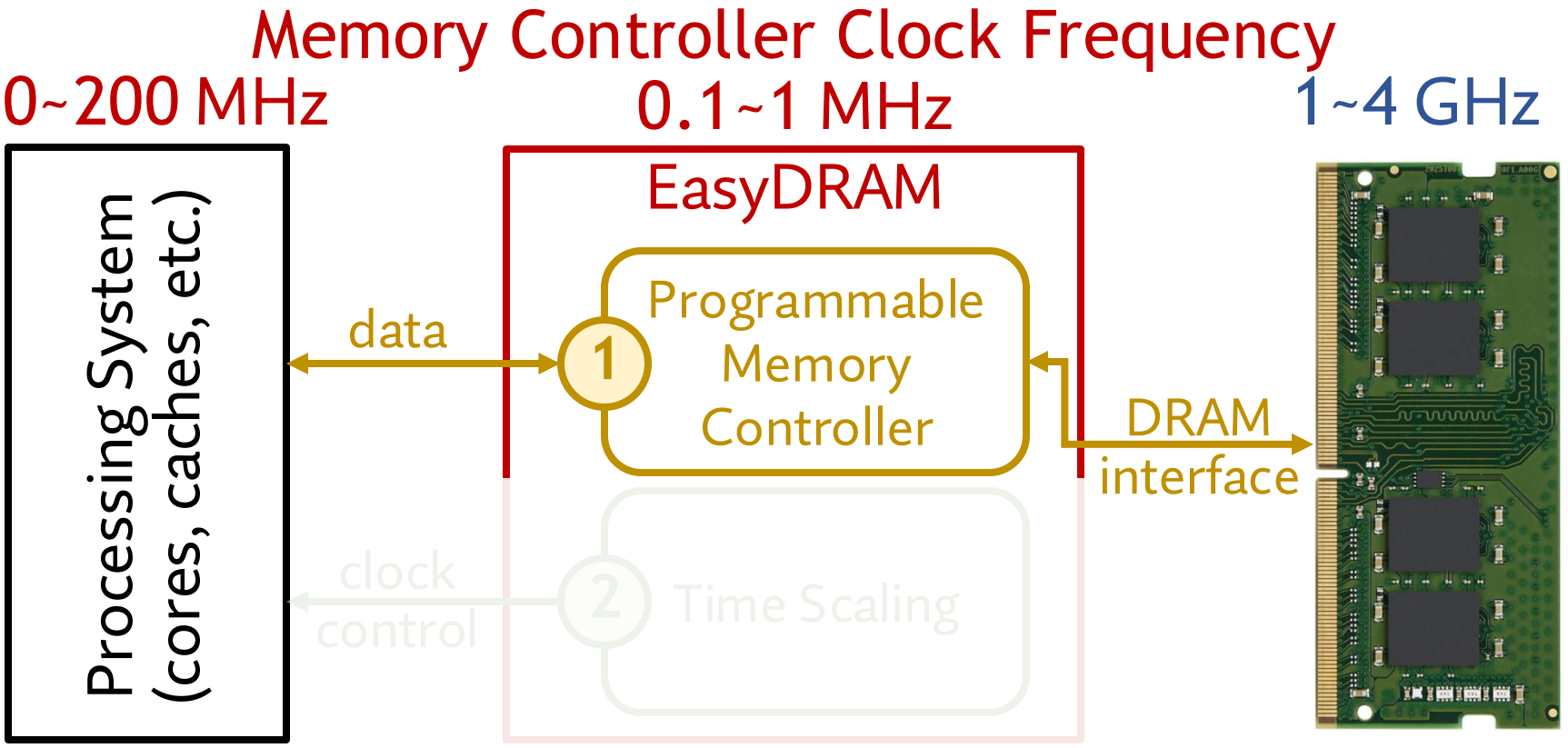
1~4 GHz



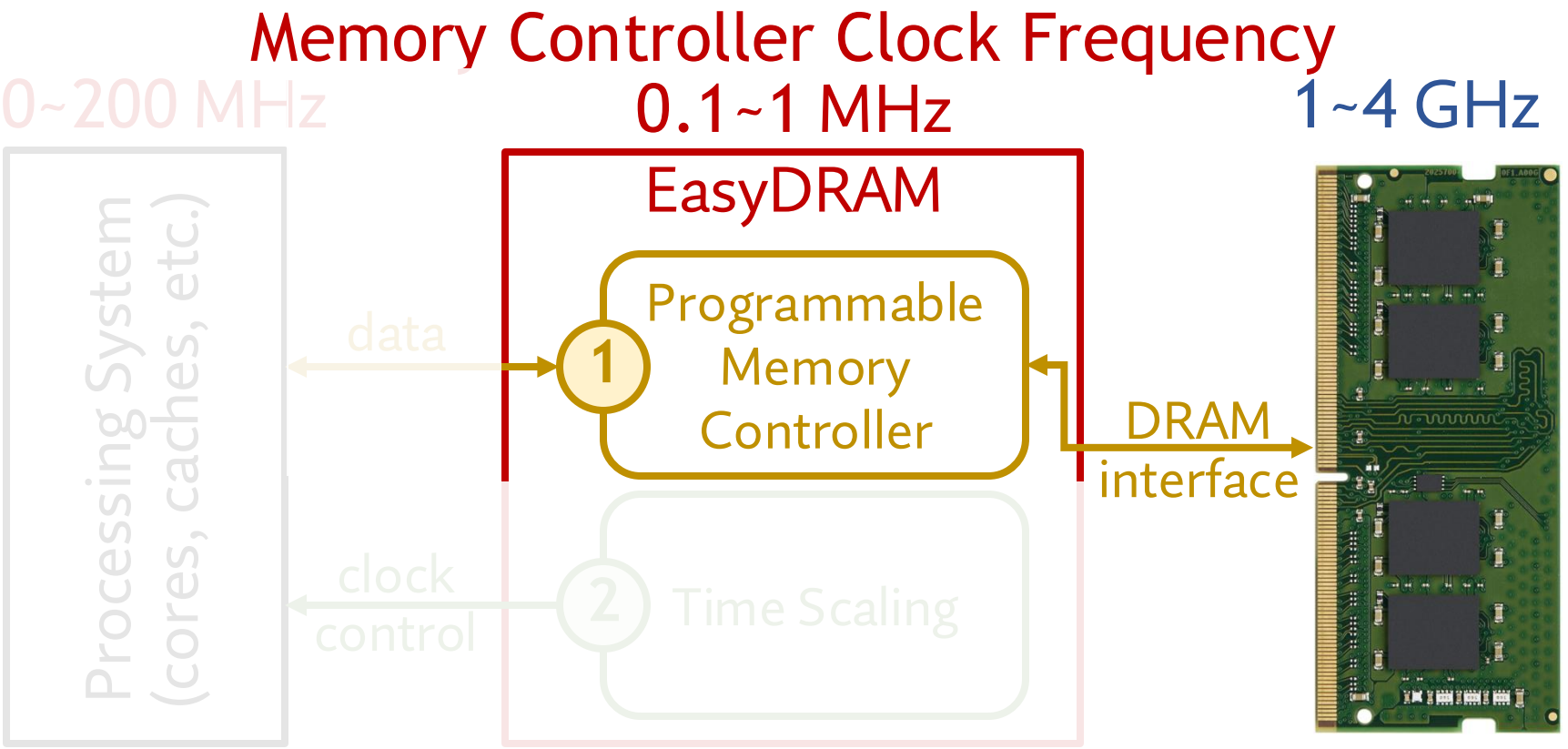
DRAM

Limits evaluation accuracy and skews the results

Another View of EasyDRAM



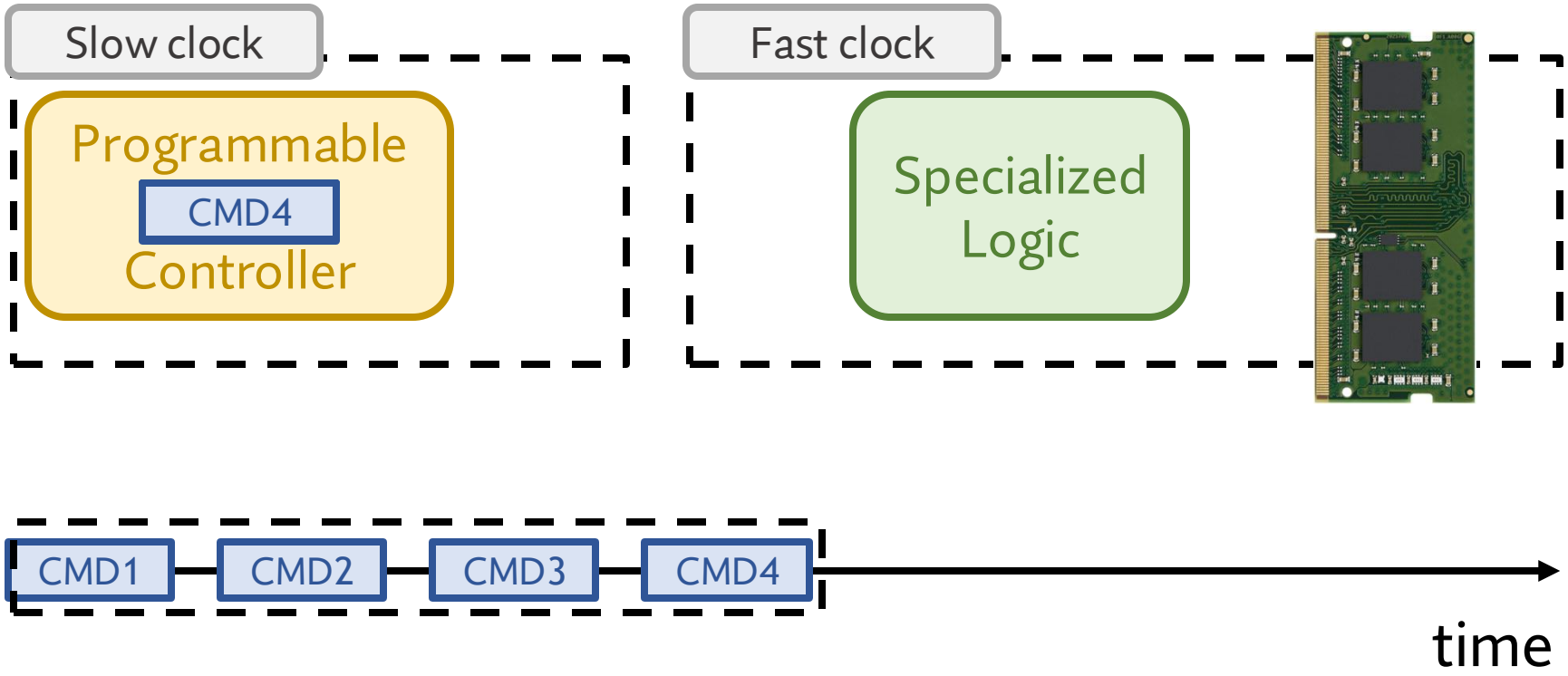
Another View of EasyDRAM



1 Cannot issue DRAM commands at nanosecond granularity

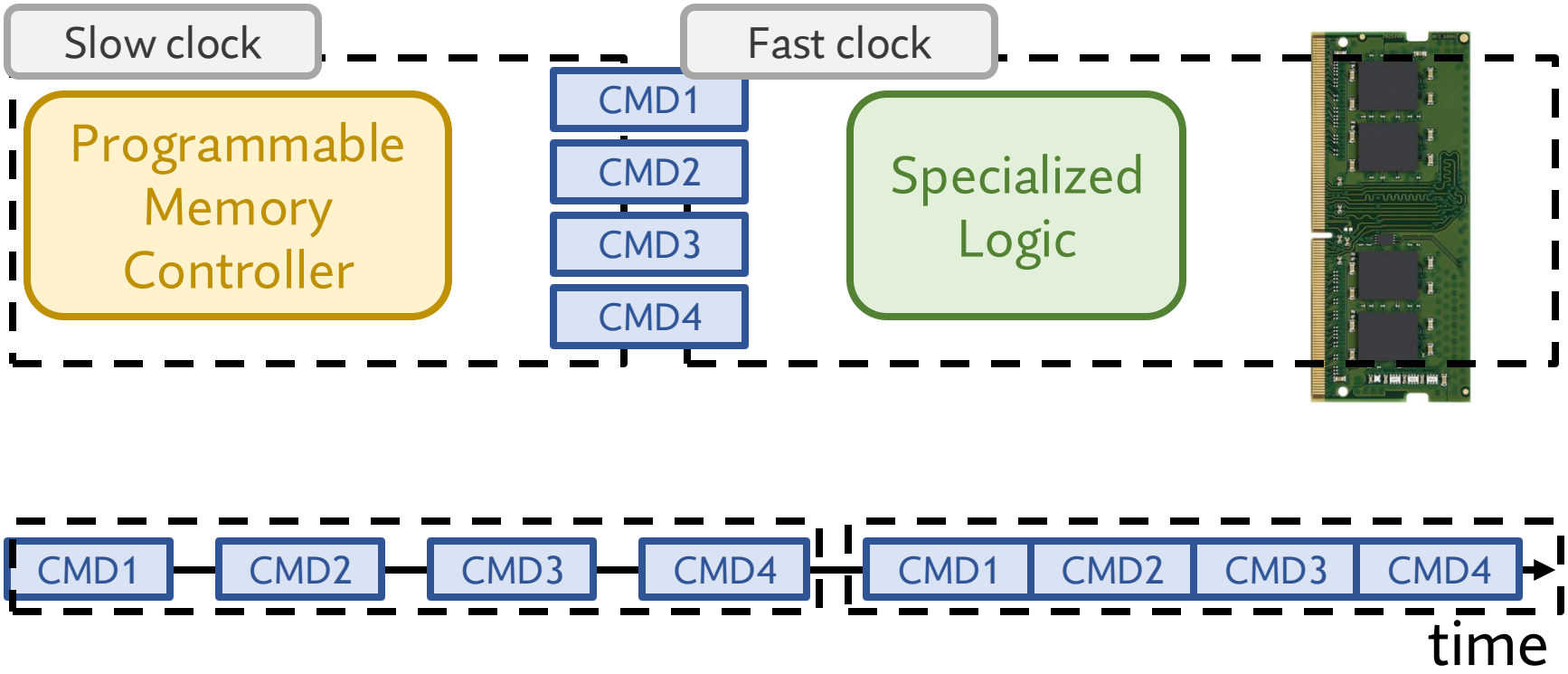
EasyDRAM Issues DRAM Commands at Nanosecond Granularity

- Key idea: Issue DRAM commands in batches



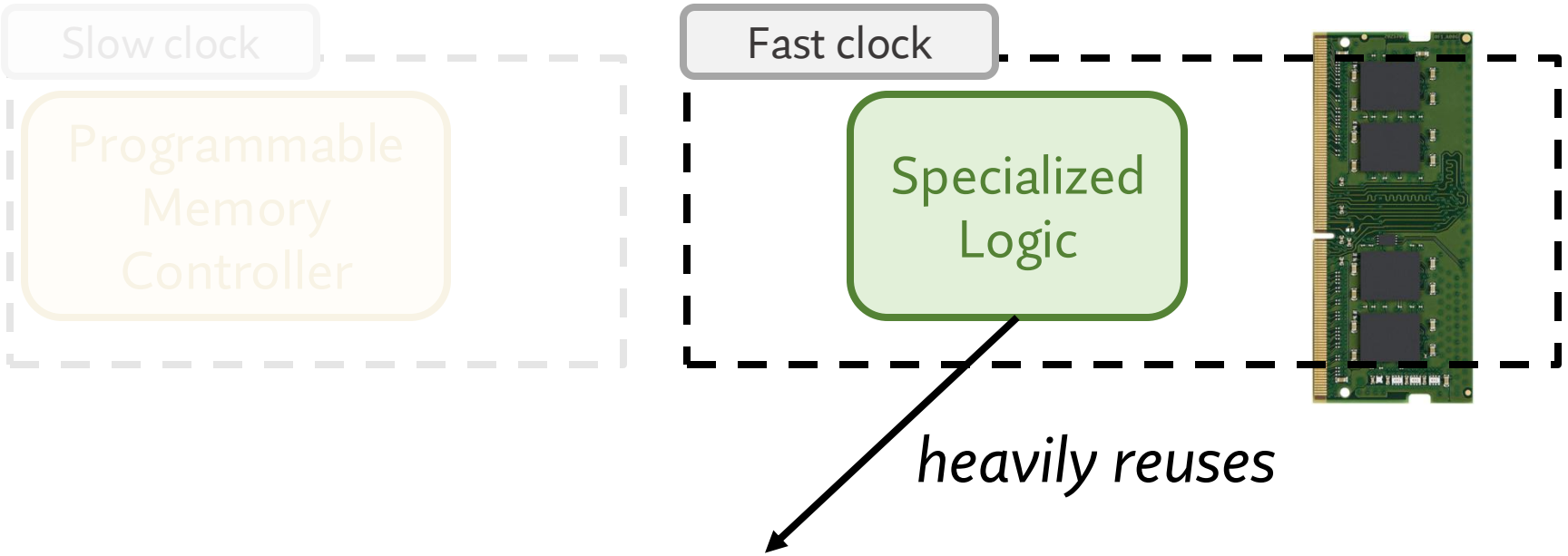
EasyDRAM Issues DRAM Commands at Nanosecond Granularity

- Key idea: Issue DRAM commands in batches



EasyDRAM Issues DRAM Commands at Nanosecond Granularity

- Key idea: Issue DRAM commands in batches

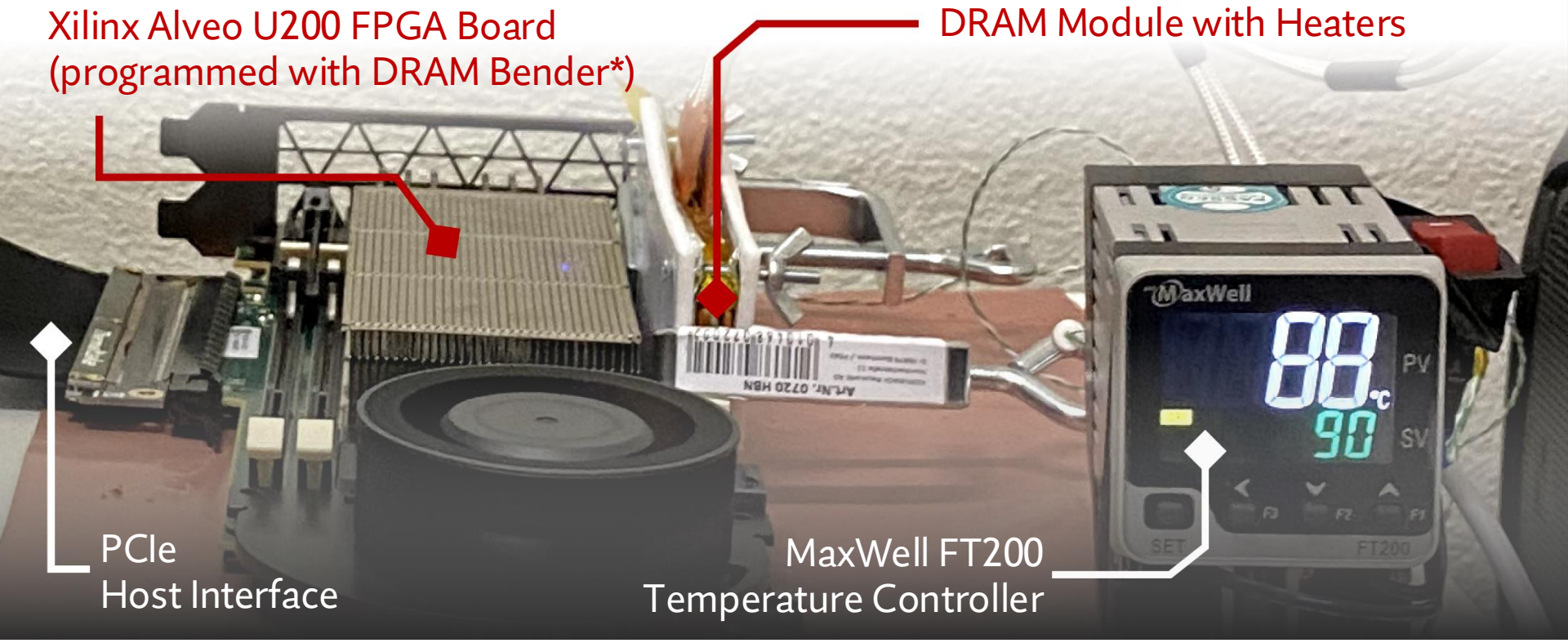


DRAM Bender DDR3/4 and HBM2 Testing Infrastructure

*Olgun et al., "DRAM Bender: An Extensible and Versatile FPGA-Based Infrastructure to Easily Test State-of-the-Art DRAM Chips," TCAD, 2023. [GitHub: https://github.com/CMU-SAFARI/DRAM-Bender]

DDR4 DRAM Testing Infrastructure

DRAM Bender on a Xilinx Virtex UltraScale+ XCU200

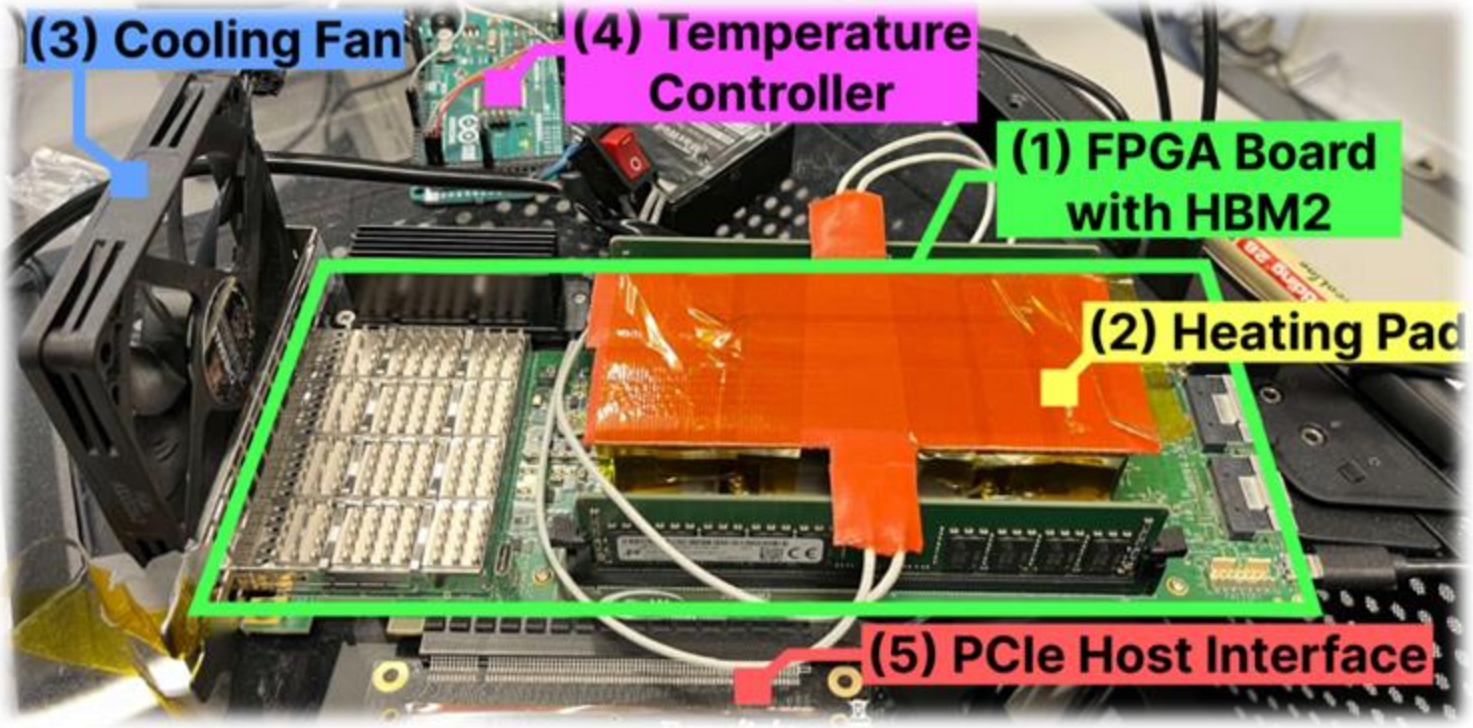


Fine-grained control over DRAM commands,
timing parameters (±1.5ns), and temperature (±0.5°C)

*Olgun et al., "DRAM Bender: An Extensible and Versatile FPGA-Based Infrastructure to Easily Test State-of-the-Art DRAM Chips," TCAD, 2023. [GitHub: https://github.com/CMU-SAFARI/DRAM-Bender]

HBM2 DRAM Testing Infrastructure

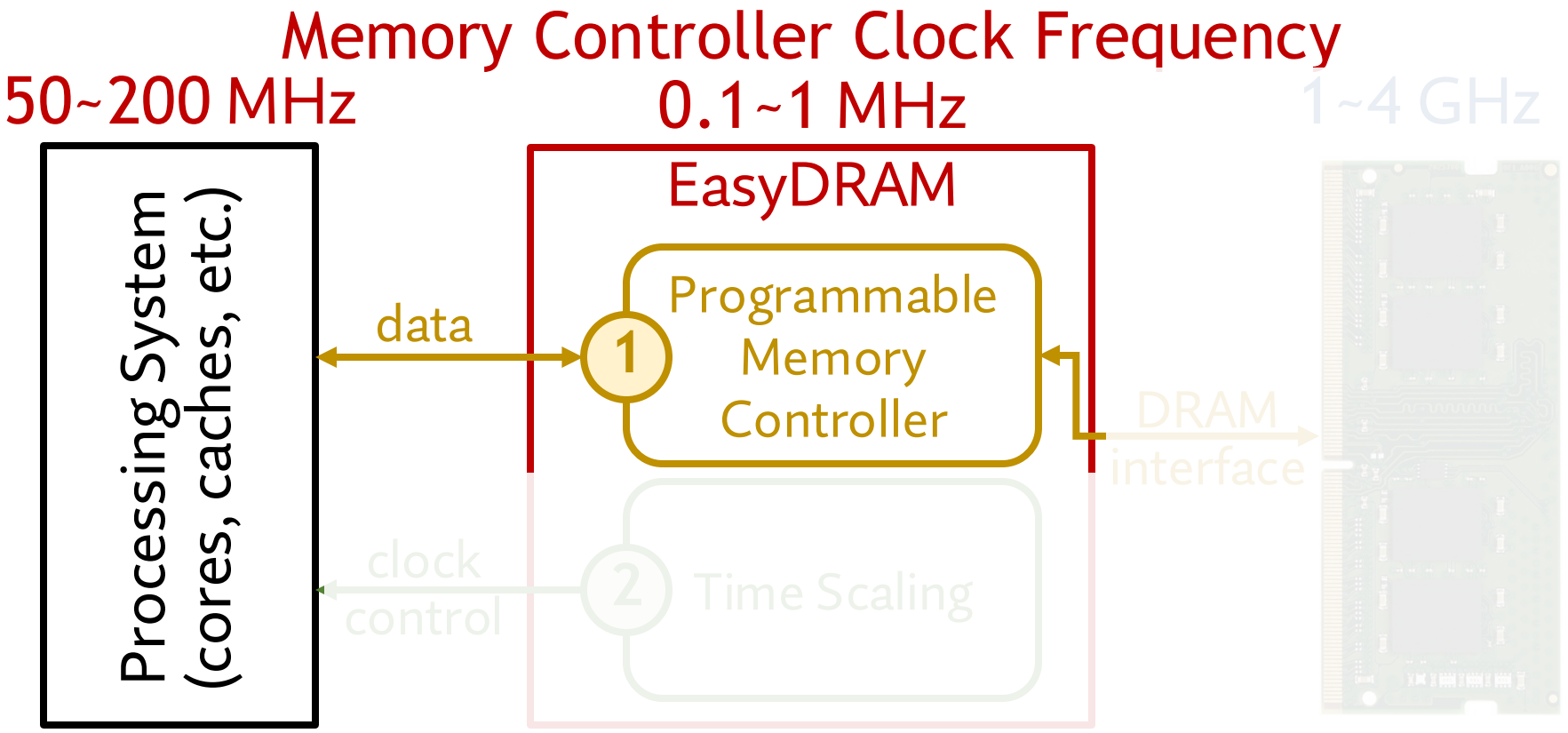
DRAM Bender on a Bittware XUPVVH



Fine-grained control over DRAM commands,
timing parameters (±1.67ns), and temperature (±0.5°C)

*Olgun et al., "DRAM Bender: An Extensible and Versatile FPGA-Based Infrastructure to Easily Test State-of-the-Art DRAM Chips," TCAD, 2023. [GitHub: https://github.com/CMU-SAFARI/DRAM-Bender]

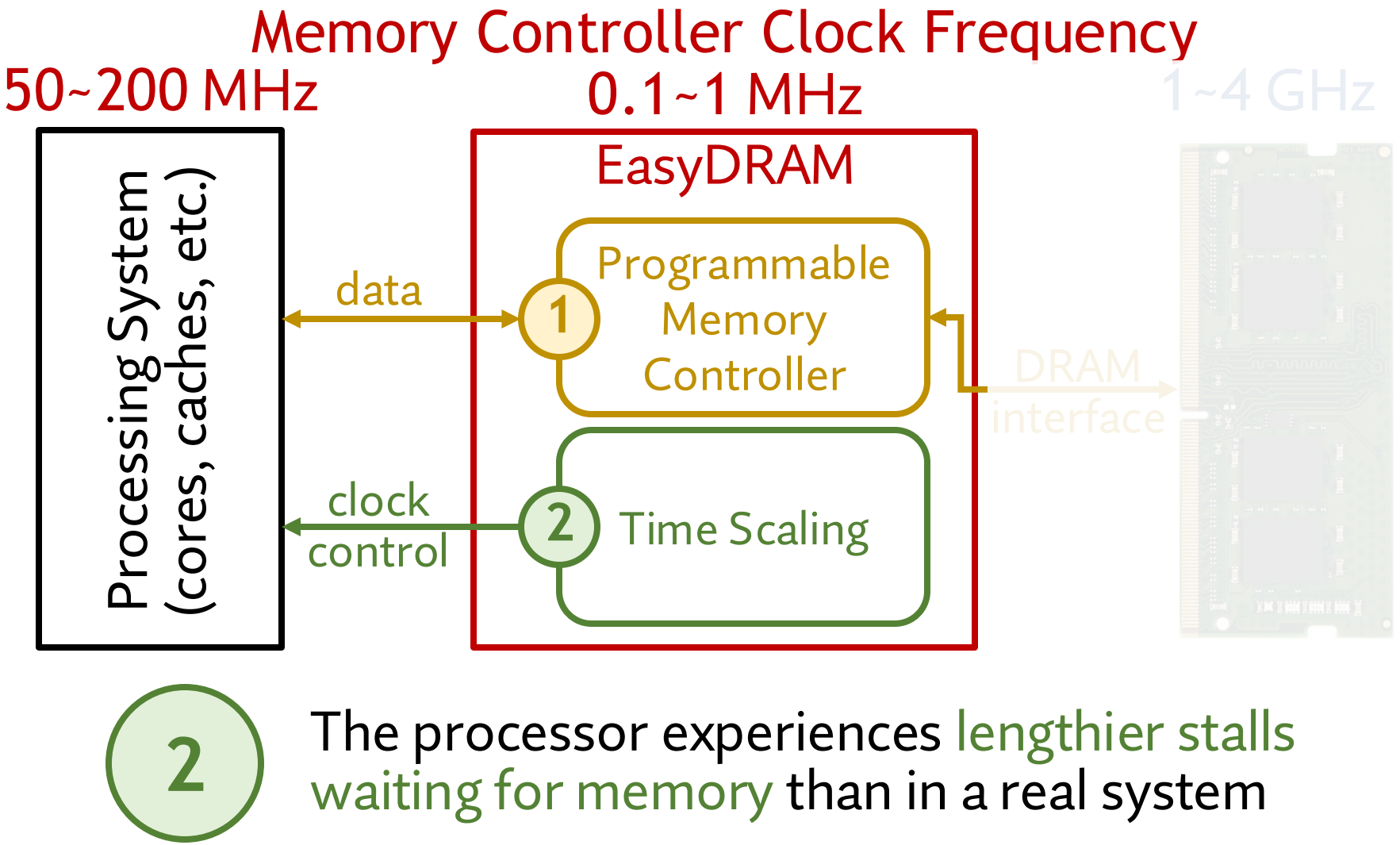
Another View of EasyDRAM



2

The processor experiences lengthier stalls waiting for memory than in a real system

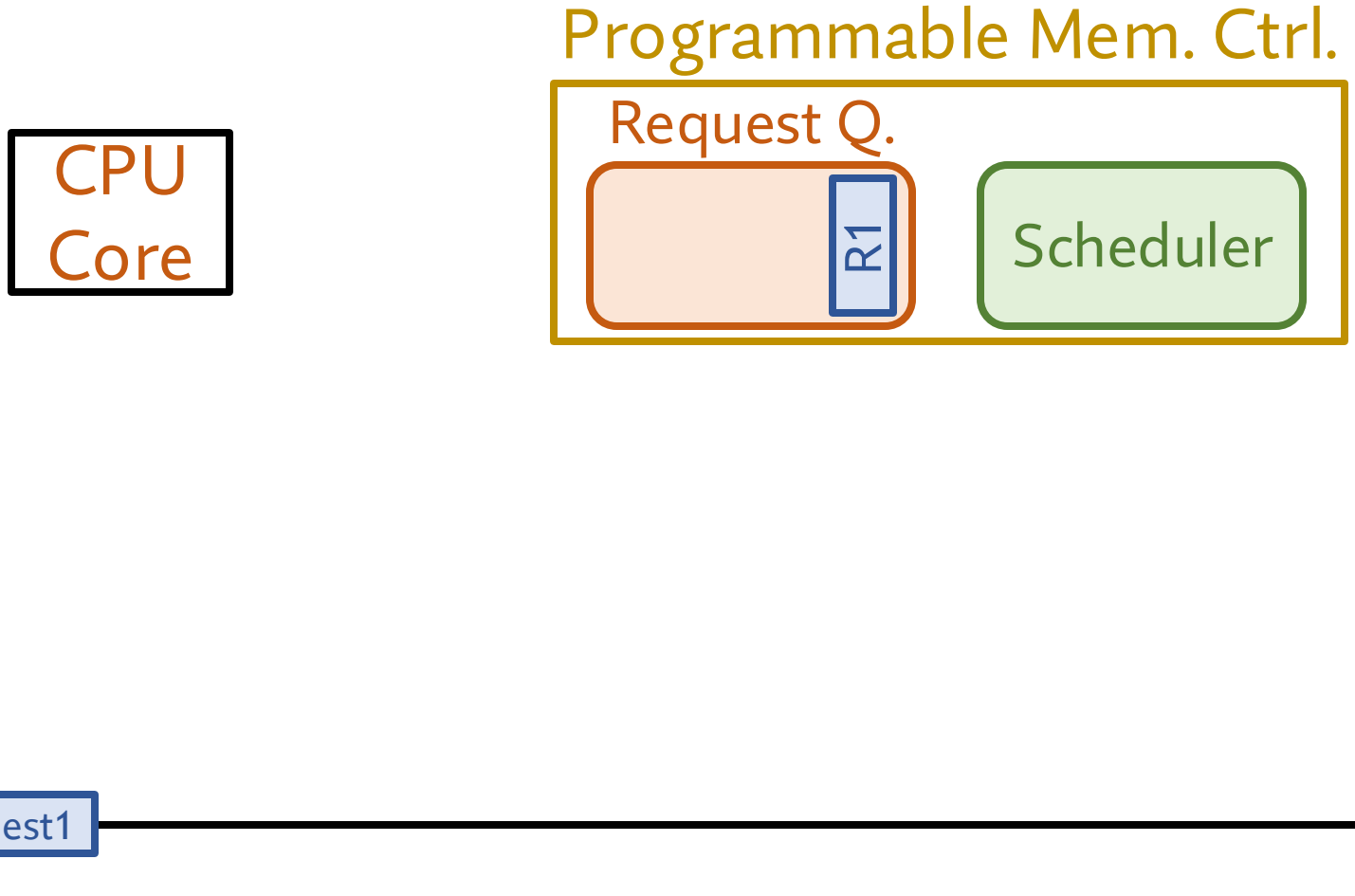
Another View of EasyDRAM



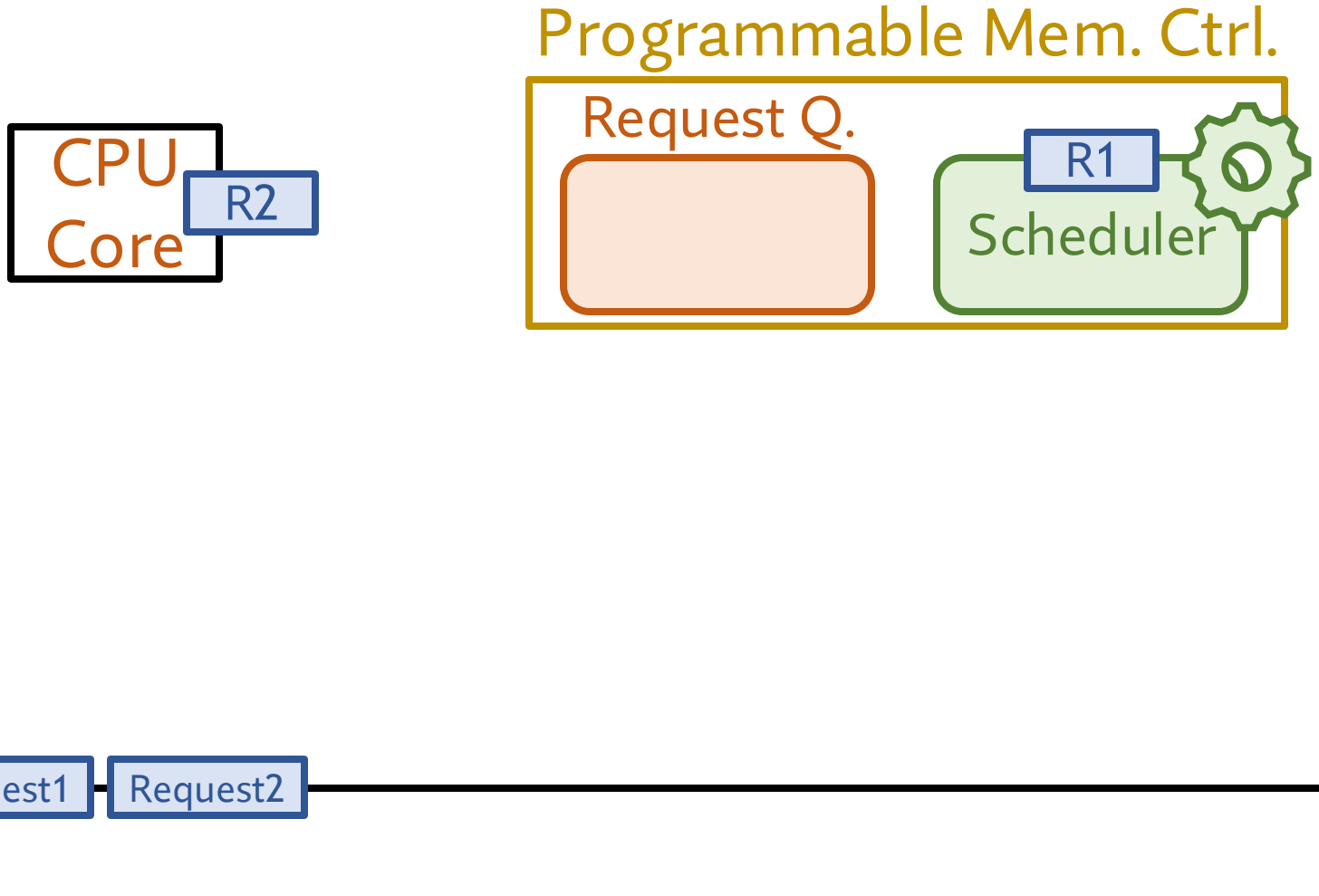
2 The processor experiences lengthier stalls waiting for memory than in a real system

EasyDRAM *without* Time Scaling

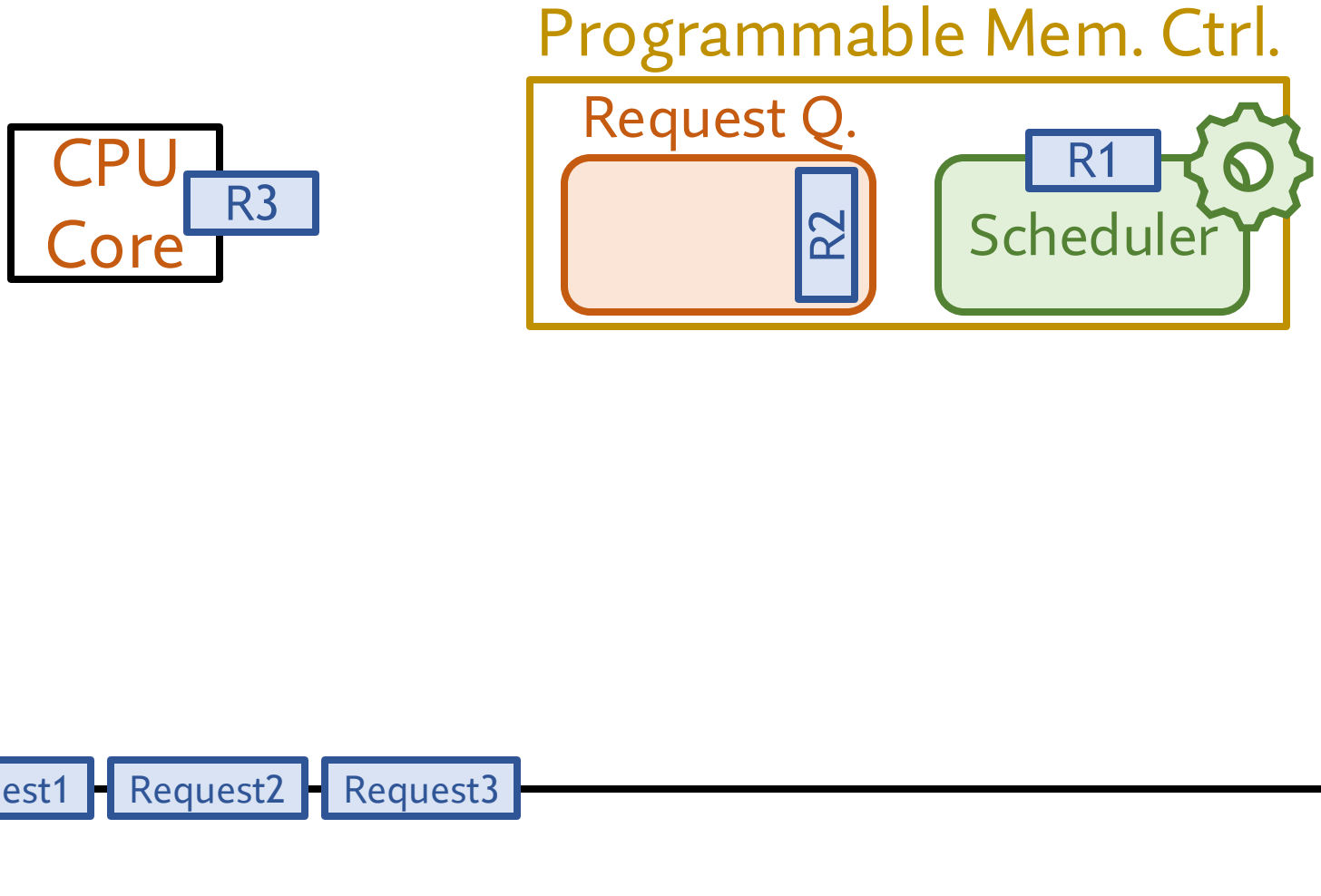
EasyDRAM *without* Time Scaling



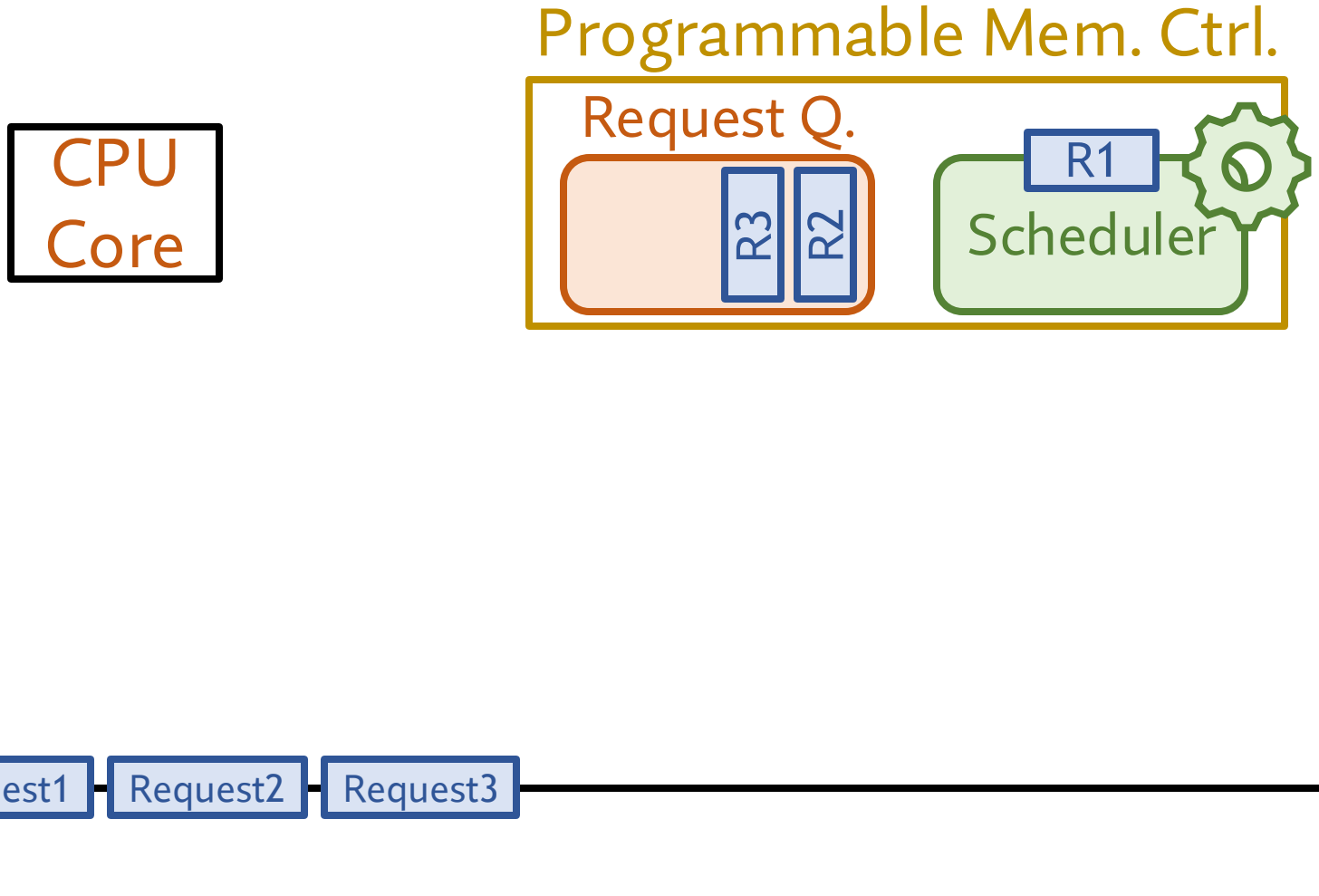
EasyDRAM *without* Time Scaling



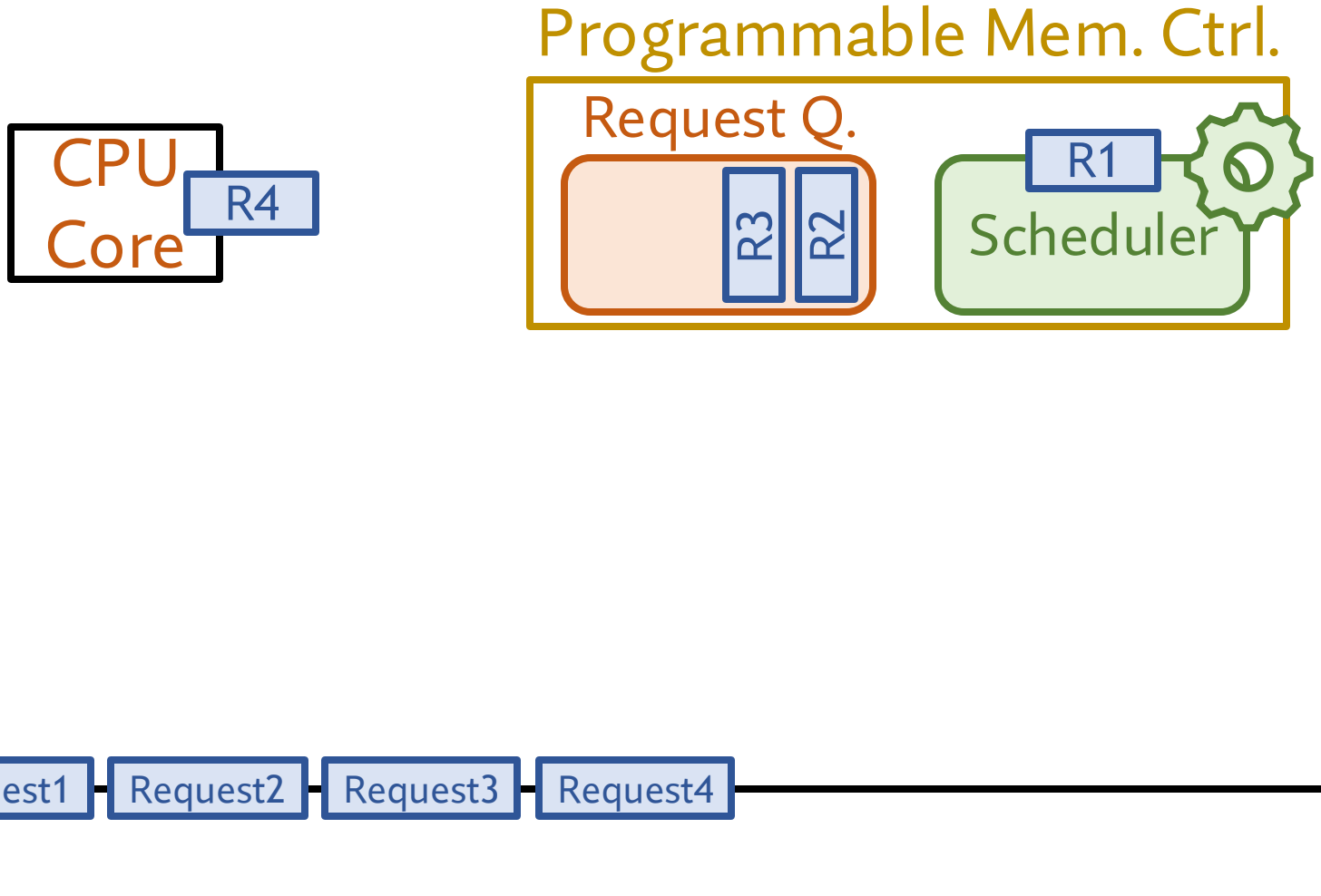
EasyDRAM *without* Time Scaling



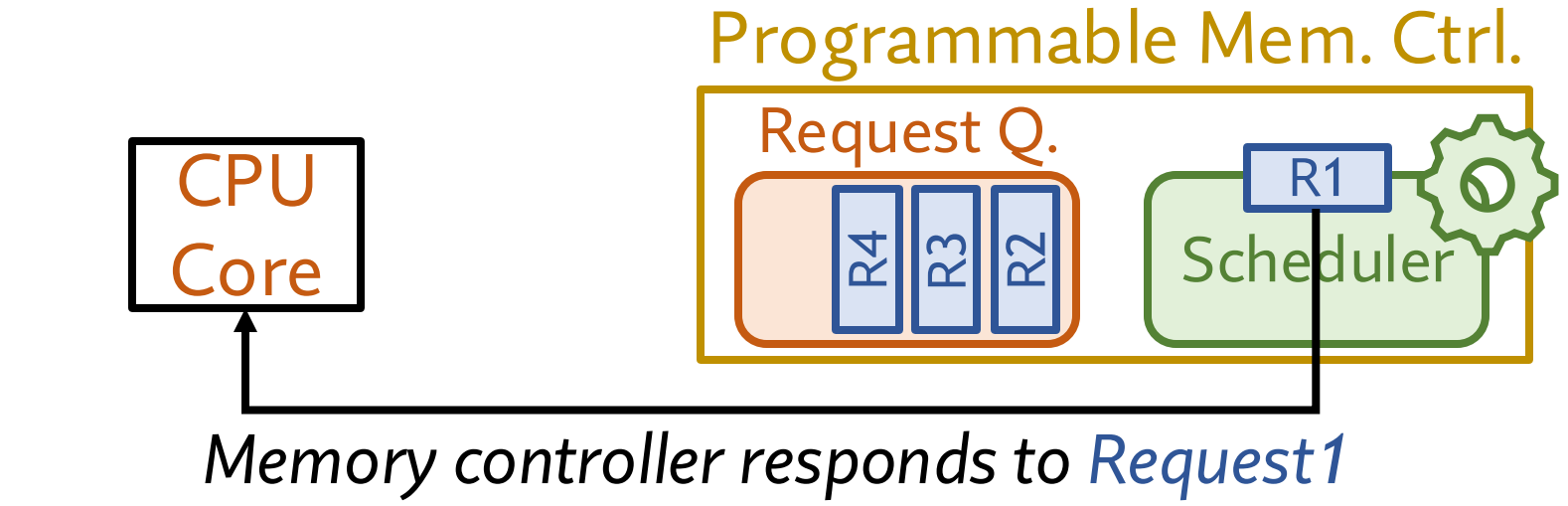
EasyDRAM *without* Time Scaling

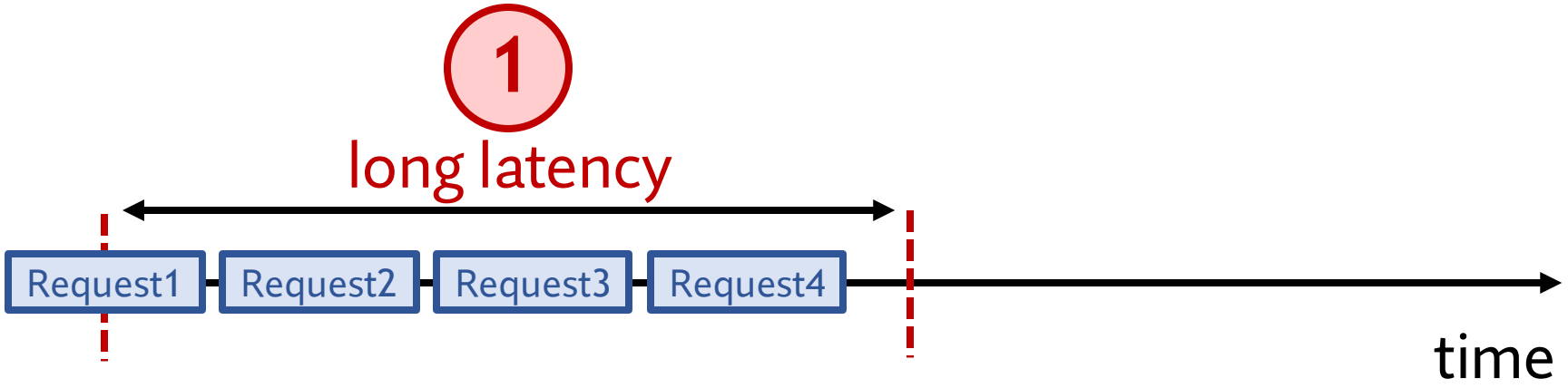


EasyDRAM *without* Time Scaling

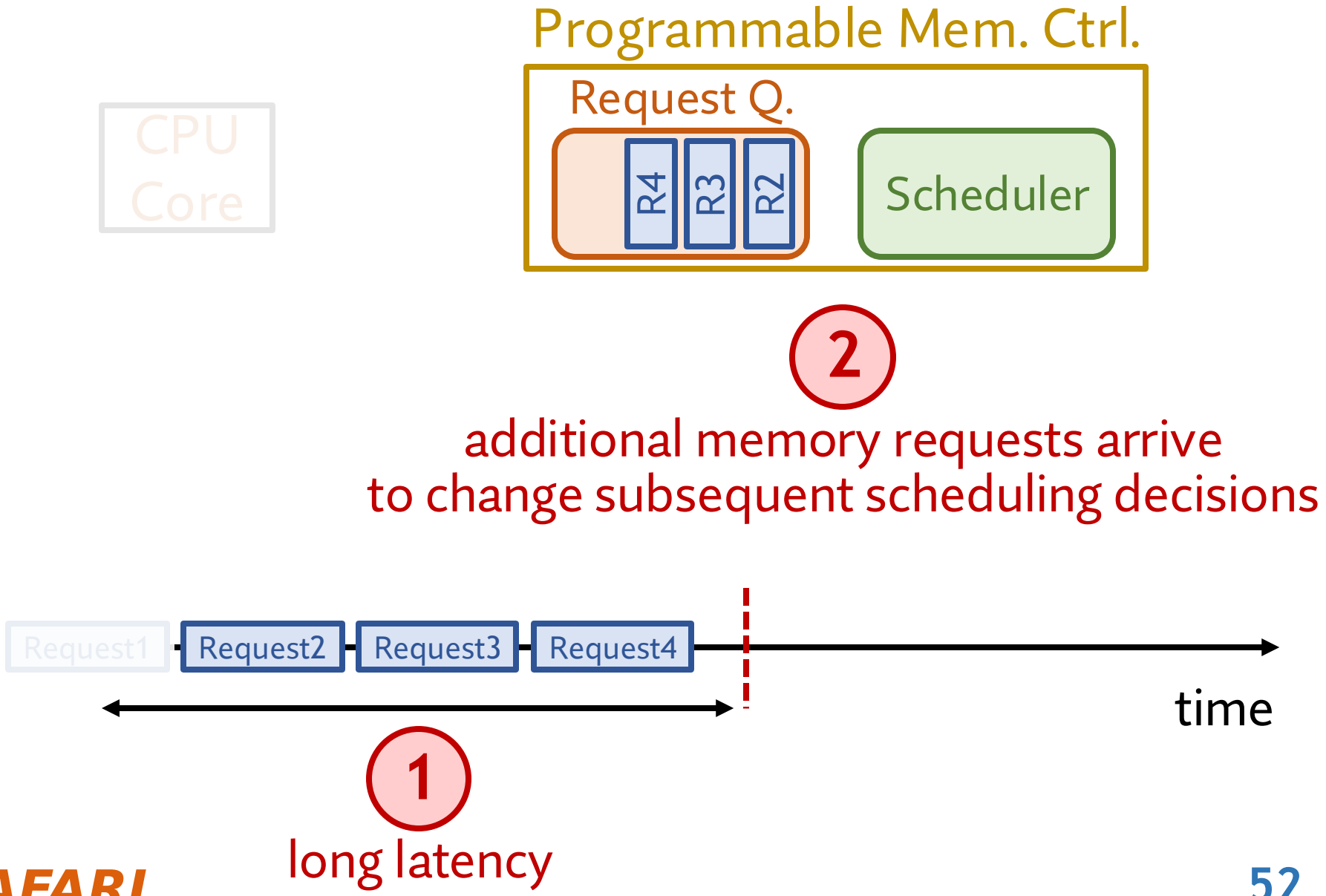


EasyDRAM *without* Time Scaling

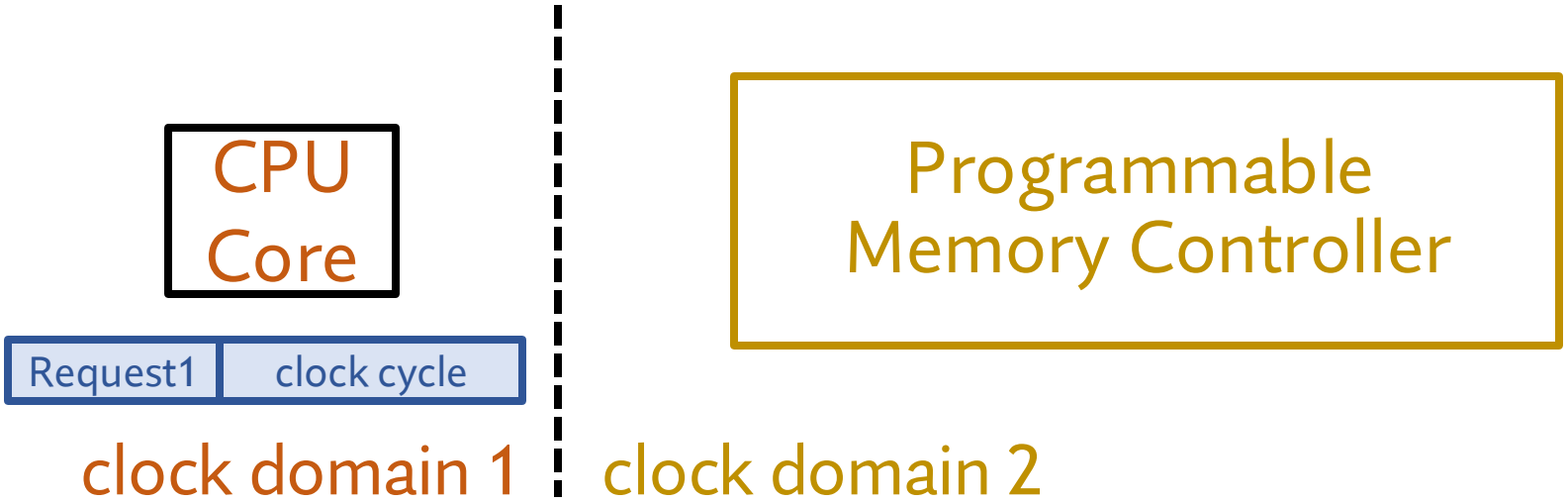




EasyDRAM *without* Time Scaling



Time Scaling Key Ideas

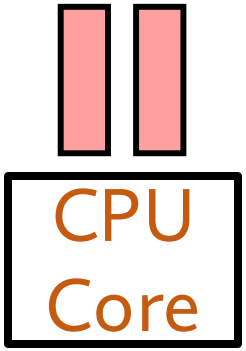


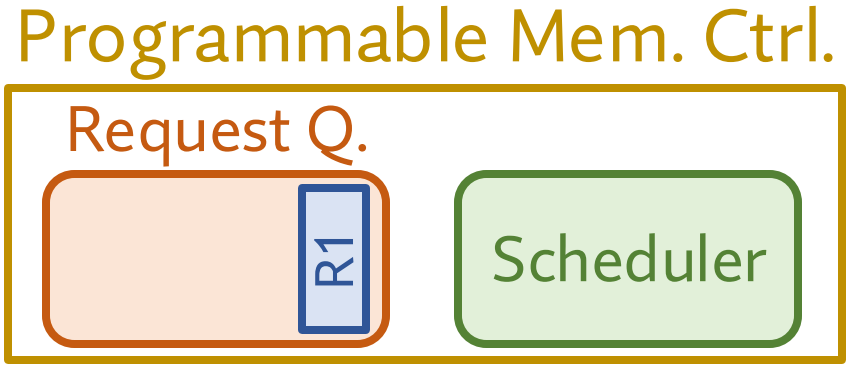
1. Split clock signal into emulation domains
   - "pause" CPU core during long latency memory requests

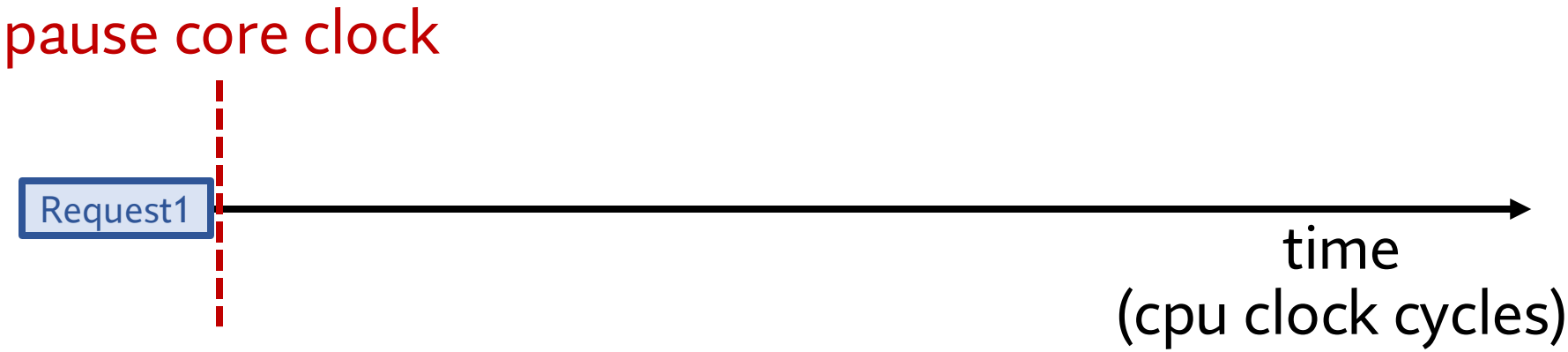
2. Tag memory requests with a clock cycle stamp
   - "ignore" the additional memory requests

EasyDRAM *with* Time Scaling

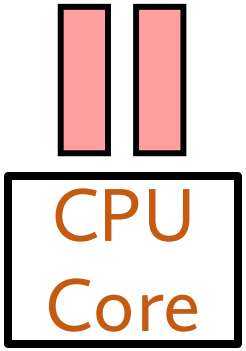
EasyDRAM *with* Time Scaling

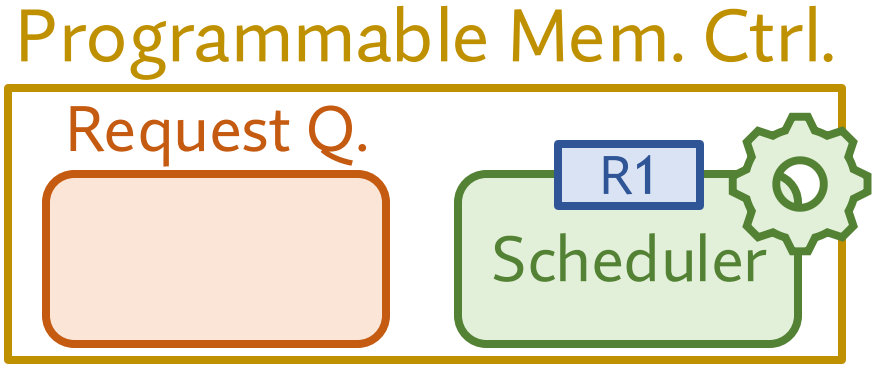




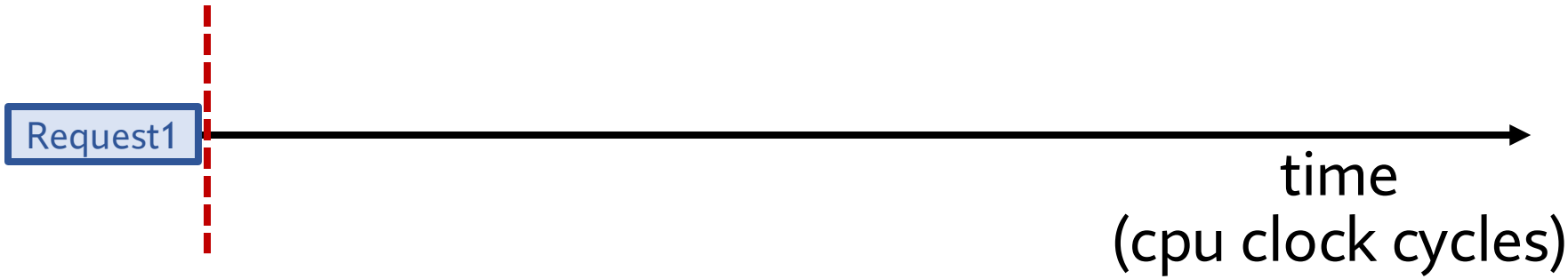


EasyDRAM *with* Time Scaling

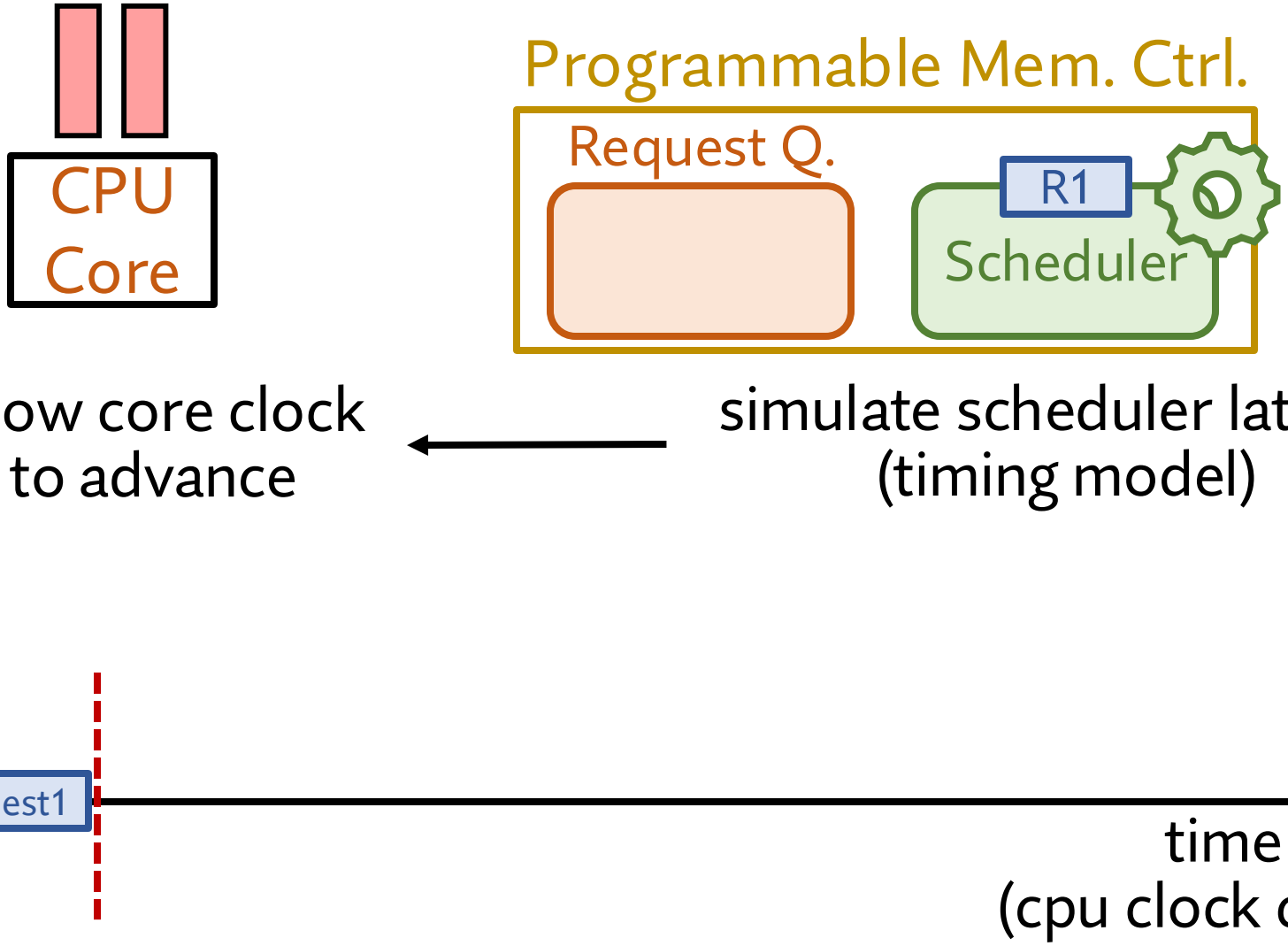




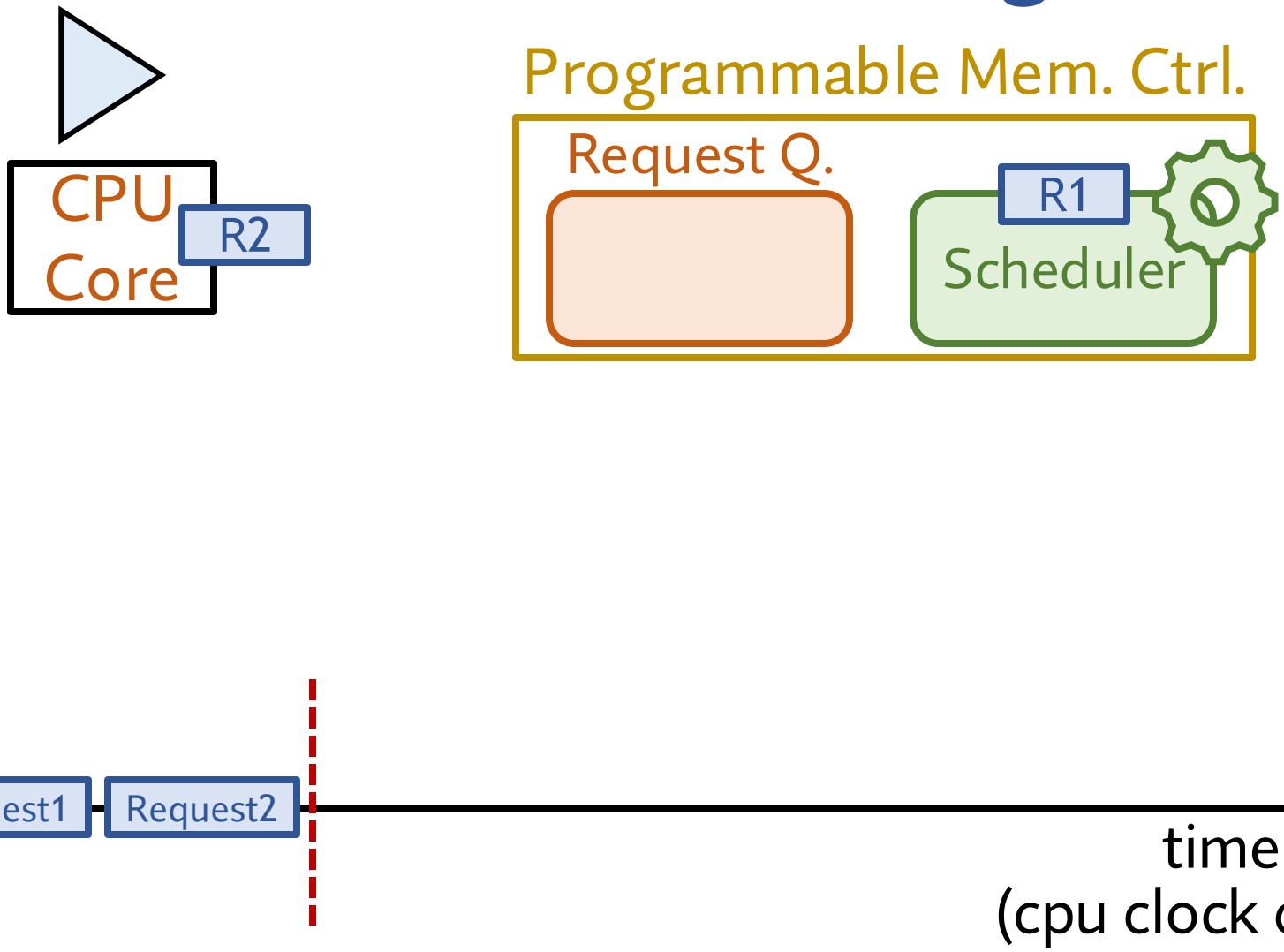
simulate scheduler latency
(timing model)



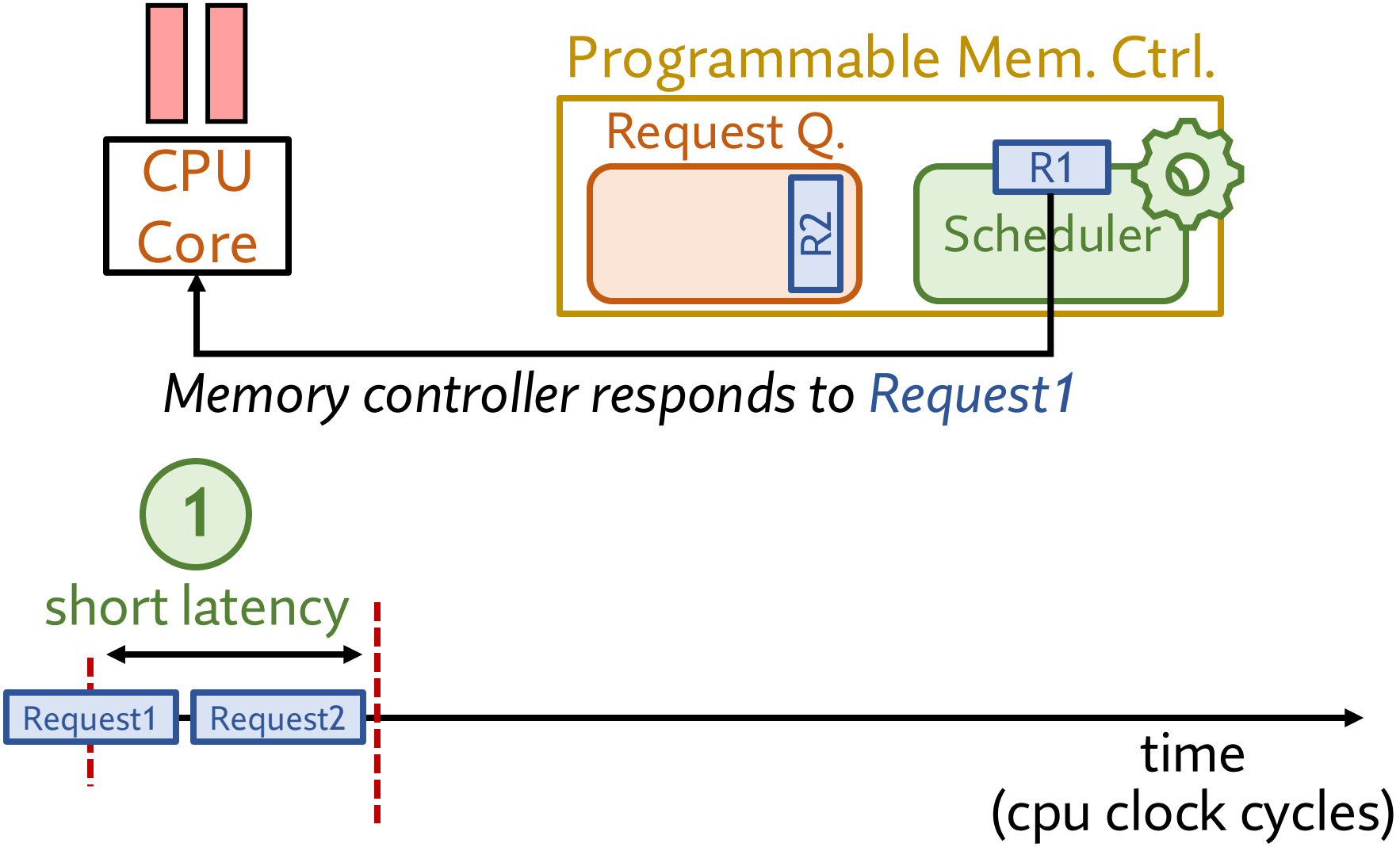
EasyDRAM *with* Time Scaling



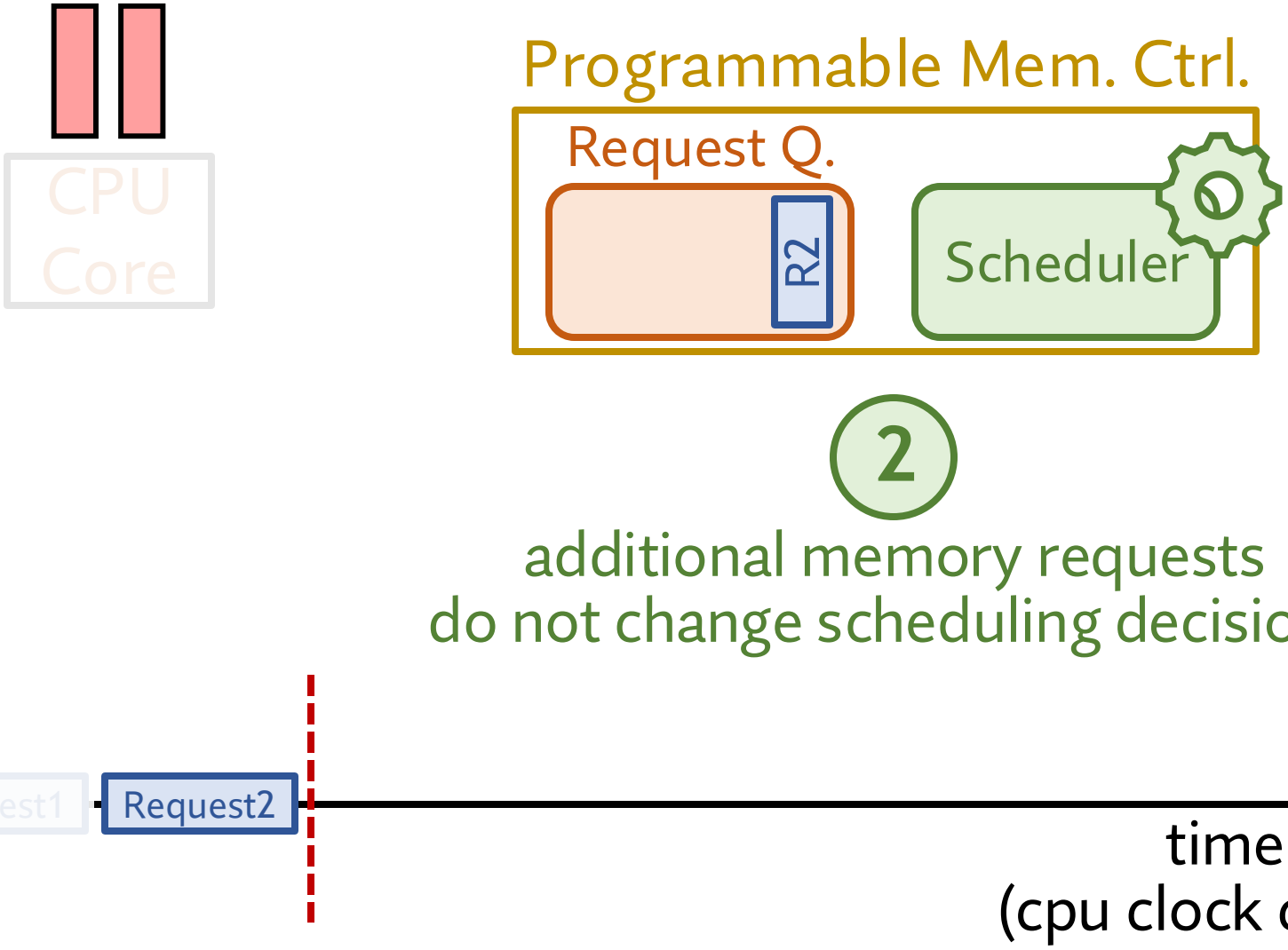
EasyDRAM *with* Time Scaling



EasyDRAM *with* Time Scaling



EasyDRAM *with* Time Scaling



EasyDRAM *with* Time Scaling

https://arxiv.org/pdf/2506.10441

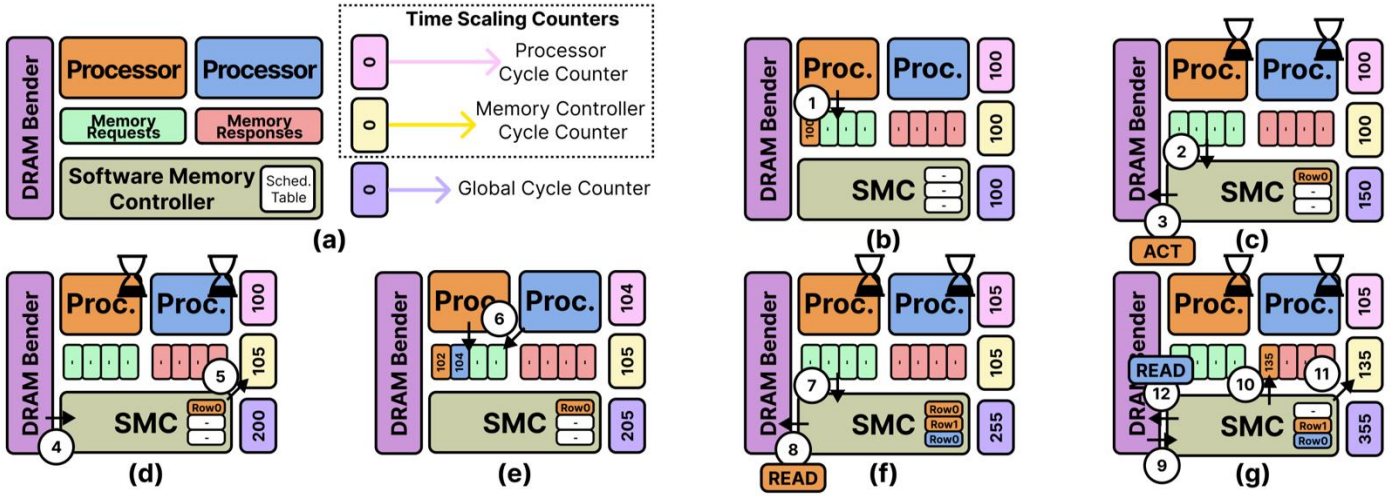


Figure 5: Time scaled execution of EasyDRAM at different emulation steps

and time spent for execution (9). SMC tags the response with a value (in terms of processor cycle count) that indicates when a processor is allowed to consume that response and writes the response to the hardware response buffer (10). Doing so ensures that the processor does *not* observe a memory request response earlier than expected and execute additional instructions that depend on this response ahead of time. The duration spent on scheduling a memory request is converted to the number of emulation cycles at the emulated system's clock frequency and the memory controller cycle counter is updated (11). The processors do *not* send new requests and SMC issues another

mands 7, then returns the data and the number of cycles taken for execution 8. SMC finalizes the memory request response, tags the response with the processor cycle count value indicating when the response can be consumed by the processor, and transfers the response to the request buffers 9. Lastly, SMC advances the memory controller time scaling counter 10 and exits critical mode. As the execution continues, the processor time scaling counter reaches the response's tagged counter value. The response is transferred to the memory bus 11, completing the request's lifetime by reaching the processor 12.

Time Scaling is Validated (I)

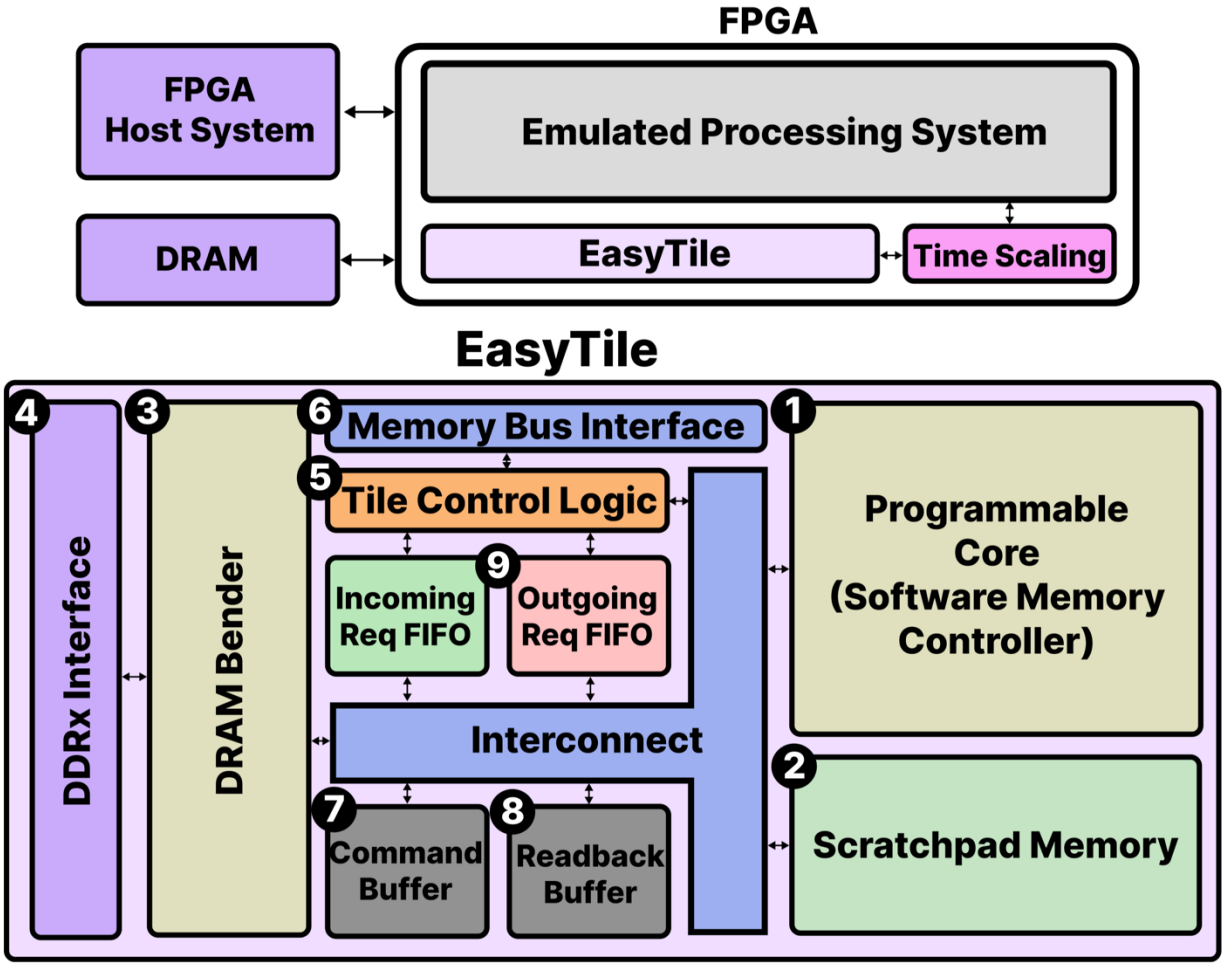
- Use a two-level approach

1. RTL simulation of EasyDRAM

2. Memory latency profile comparison to a real system

EasyDRAM Implementation

https://arxiv.org/pdf/2506.10441



EasyDRAM's FPGA Prototype

Full system prototype on Xilinx VCU108 FPGA board

- **RISC-V System:** Out-of-order Boom Core in Rocket Chip with 512 KiB L2 cache
- **Real DDR4 DRAM:** Micron EDY4016AABG-DR-F-D, 4 GiB, 1,333 MT/s

Time Scaling is Validated (II)

- Use a two-level approach

1

RTL simulation of EasyDRAM

1. RTL simulation: core clocked at 1 GHz + **No** Time Scaling
2. EasyDRAM: core clocked at 100 MHz + Time Scaling

EasyDRAM's w/ Time Scaling execution time within 0.1% of that of RTL simulation average across 29 microbenchmarks

Time Scaling is Validated (III)

2

NVIDIA Jetson Nano SoC (ARM Cortex A57 CPU)

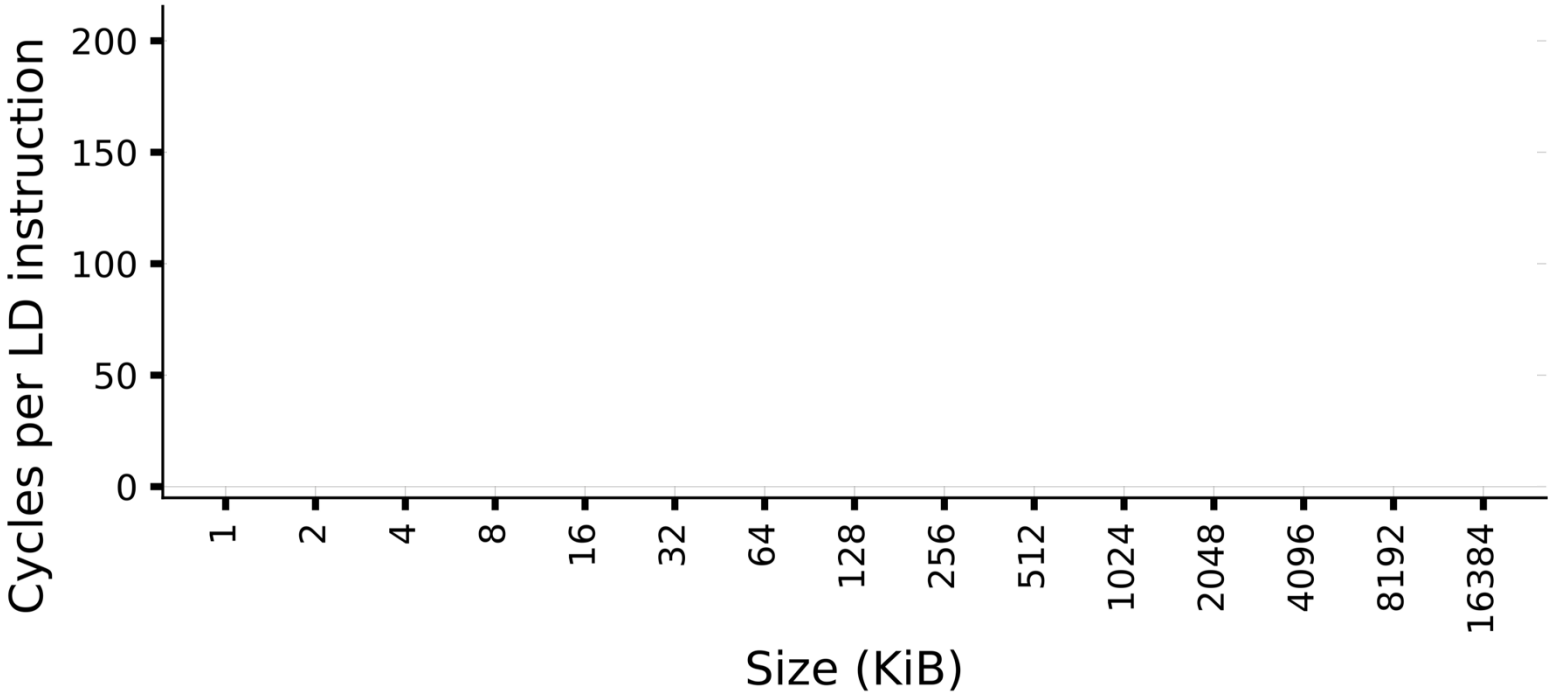
- Configure EasyDRAM to match the SoC system
- Compare memory latency profile to EasyDRAM

Time Scaling is Validated (III)

2

NVIDIA Jetson Nano SoC (ARM Cortex A57 CPU)

- Configure EasyDRAM to match the SoC system
- Compare memory latency profile to EasyDRAM

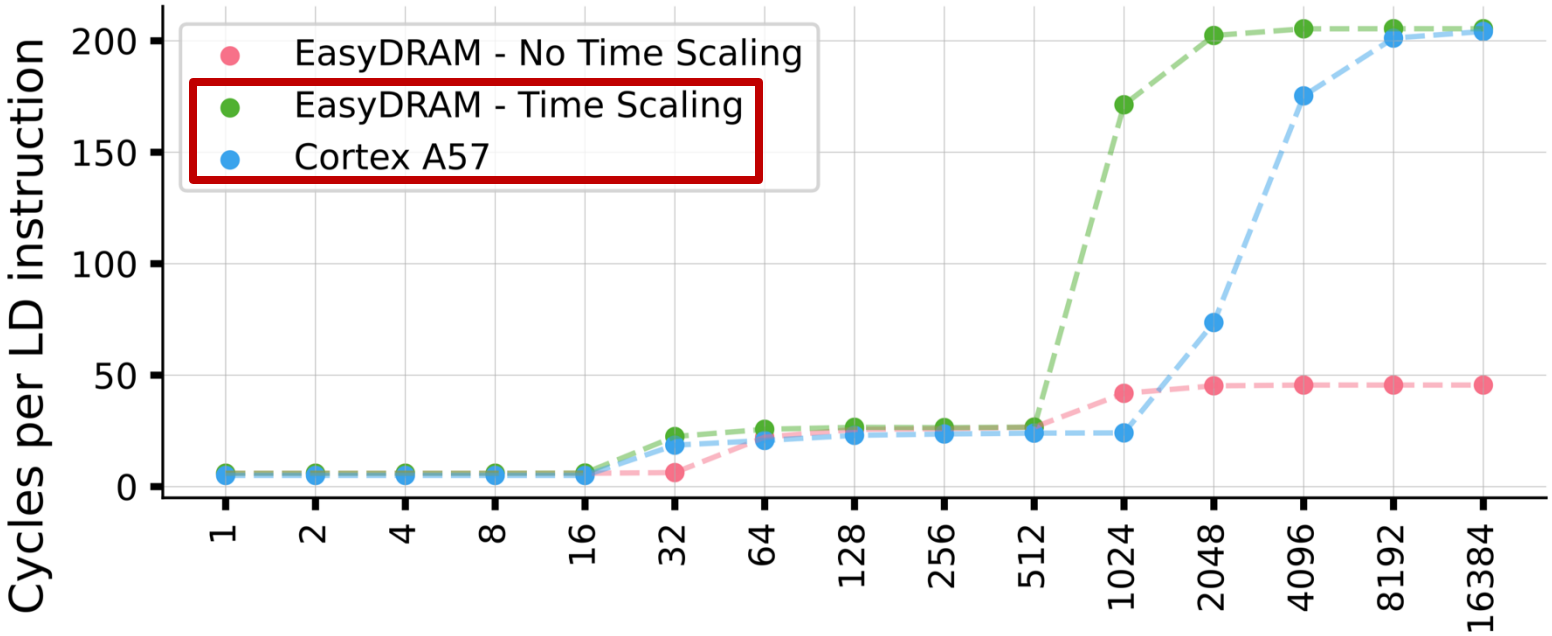


Time Scaling is Validated (III)

2

NVIDIA Jetson Nano SoC (ARM Cortex A57 CPU)

- Configure EasyDRAM to match the SoC system
- Compare memory latency profile to EasyDRAM



EasyDRAM with Time Scaling exhibits similar latency profile to the Cortex A57 system

Talk Outline

I. Background

II. Motivation

III. EasyDRAM

IV. Case Studies

V. Conclusion

Two Case Studies

- Realistic performance evaluations of

1. In-DRAM Row Copy (RowClone)

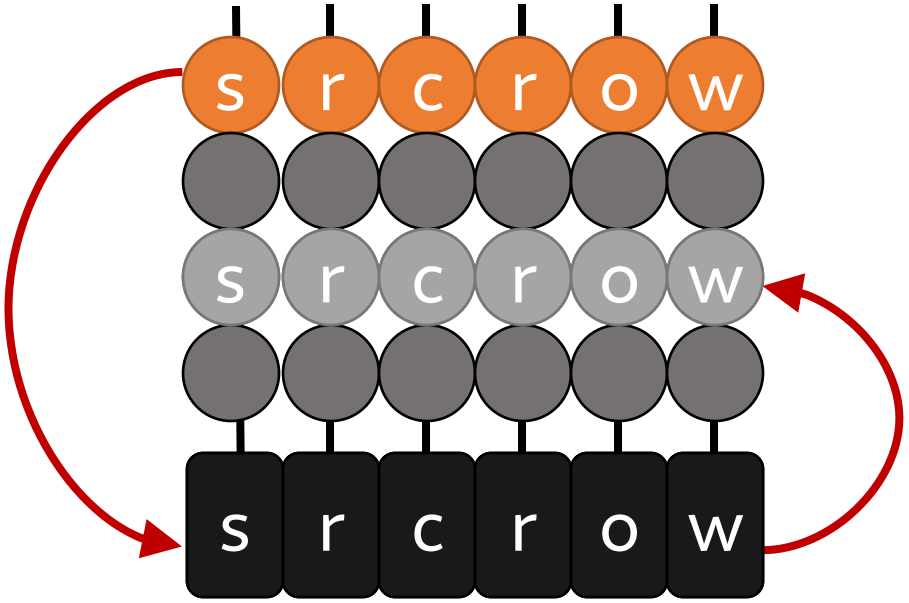
2. DRAM Access Latency Reduction

Two Case Studies

- Realistic performance evaluations of

1. **In-DRAM Row Copy (RowClone)**
2. DRAM Access Latency Reduction

Recall: RowClone Key Idea



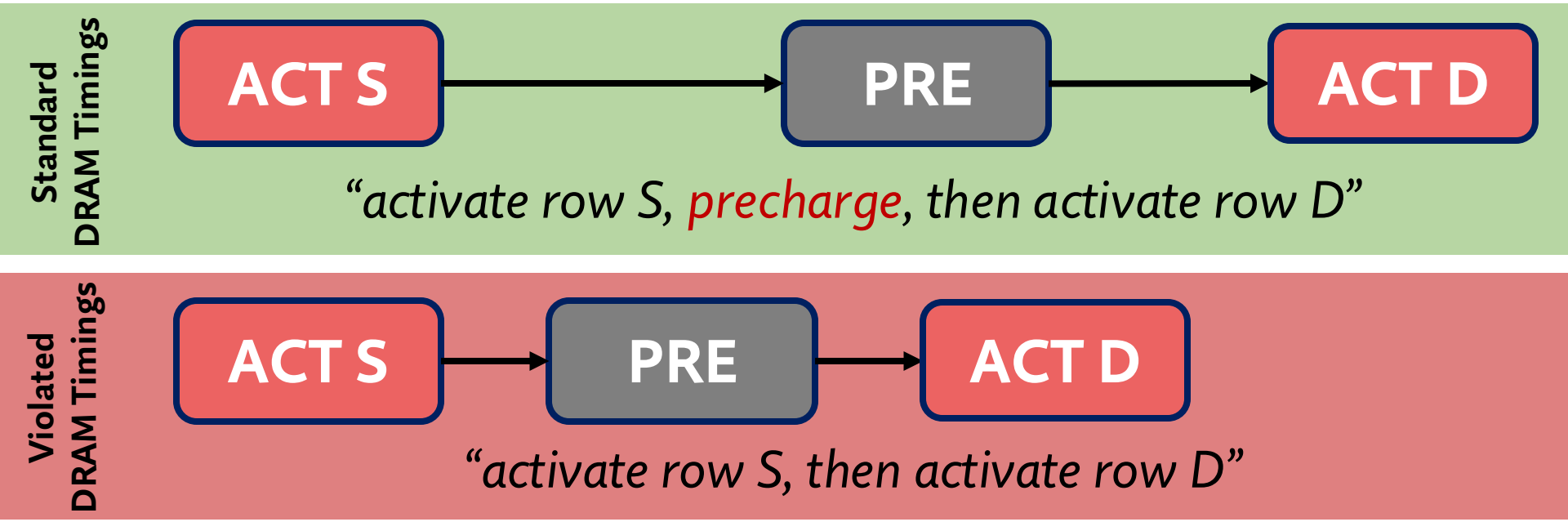
1. Source row to sense amplifiers

2. Sense amplifiers to destination row

RowClone in Real DRAM Chips

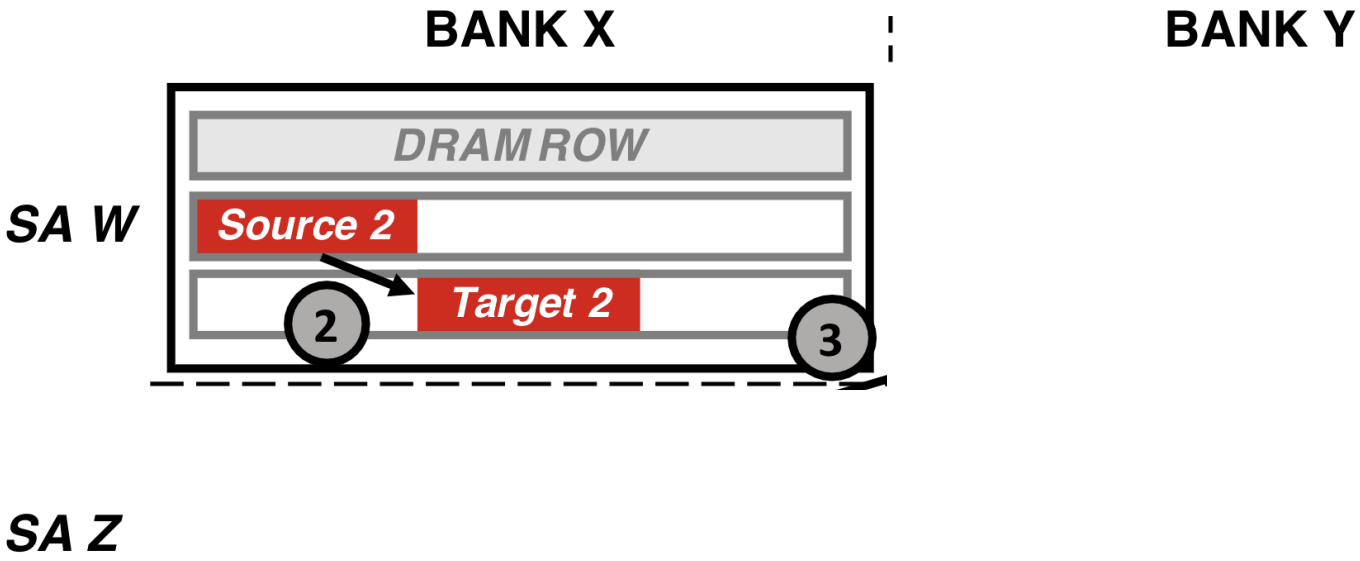
Key Idea: Use carefully created DRAM command sequences

- ComputeDRAM **[Gao+, MICRO'19]** demonstrates in-DRAM copy operations in real DDR3 chips
- ACT → PRE → ACT command sequence with greatly reduced DRAM timing parameters

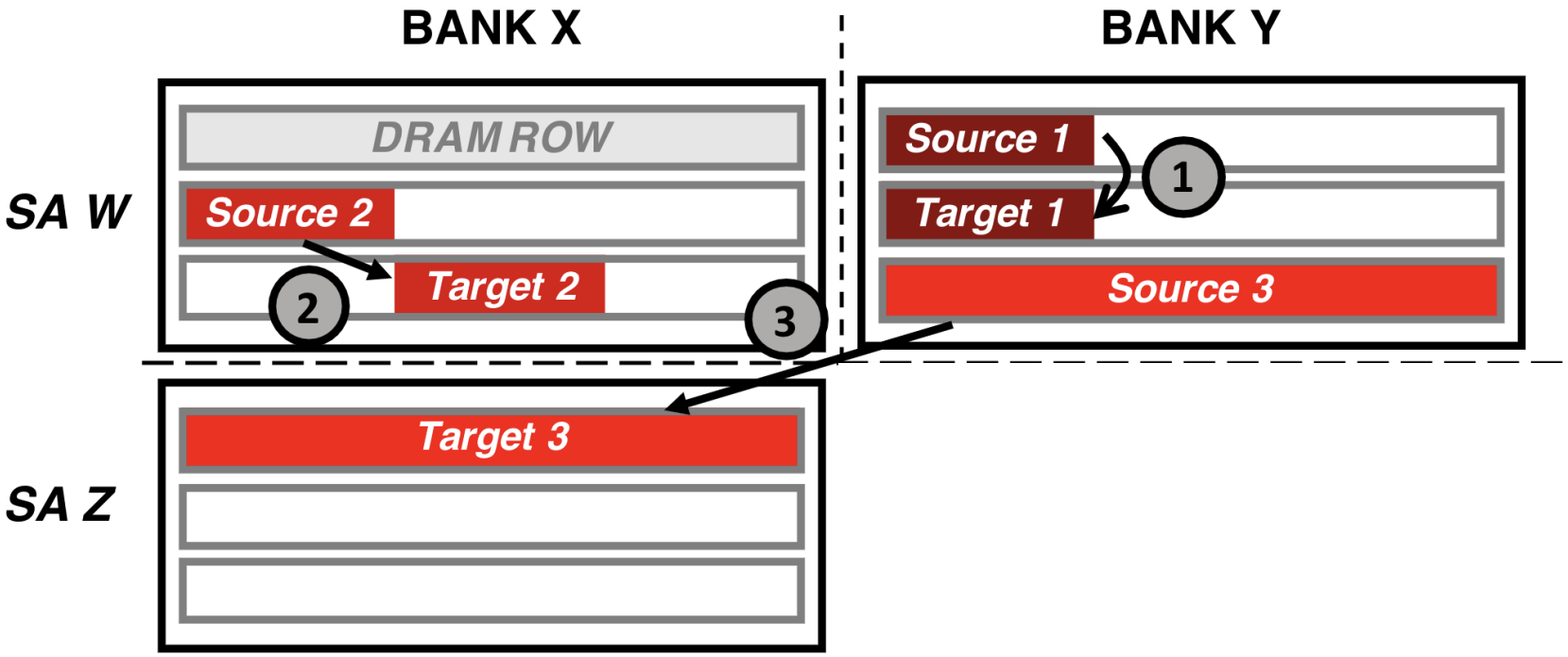


RowClone Memory Allocation Constraints

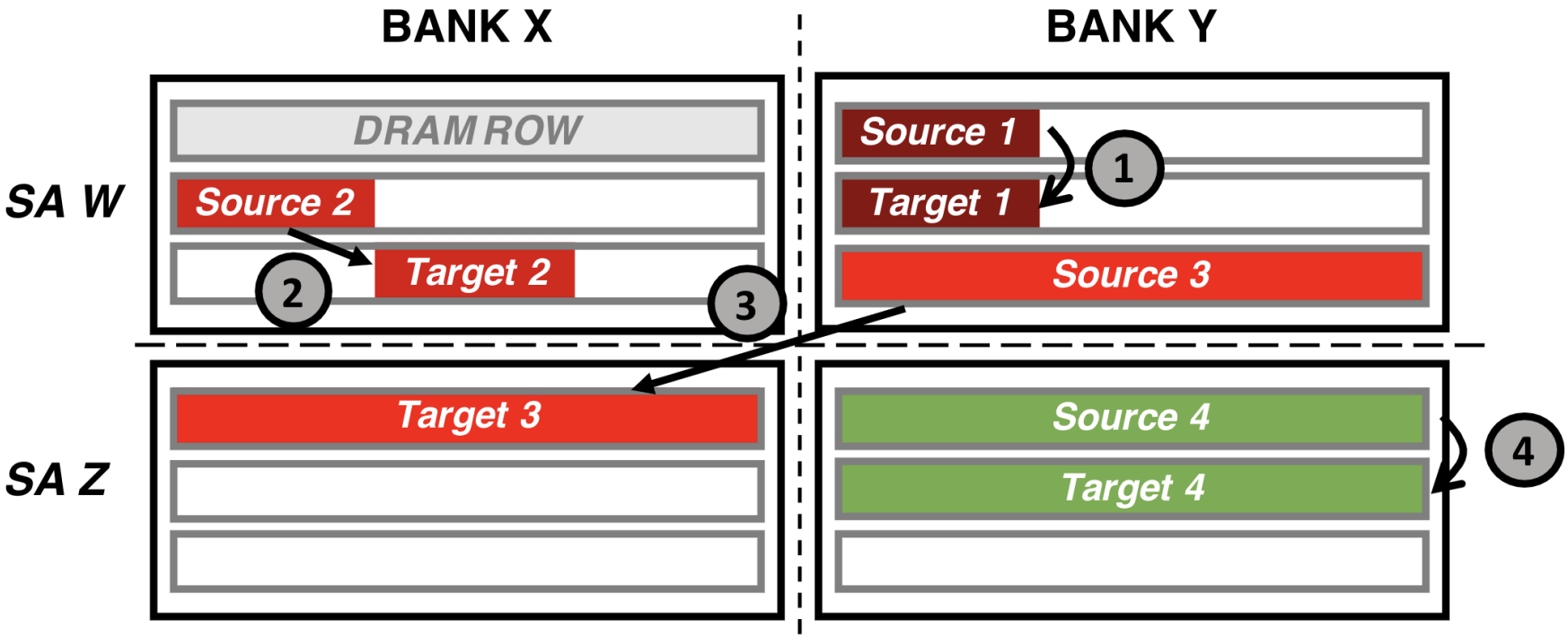
RowClone Memory Allocation Constraints



RowClone Memory Allocation Constraints



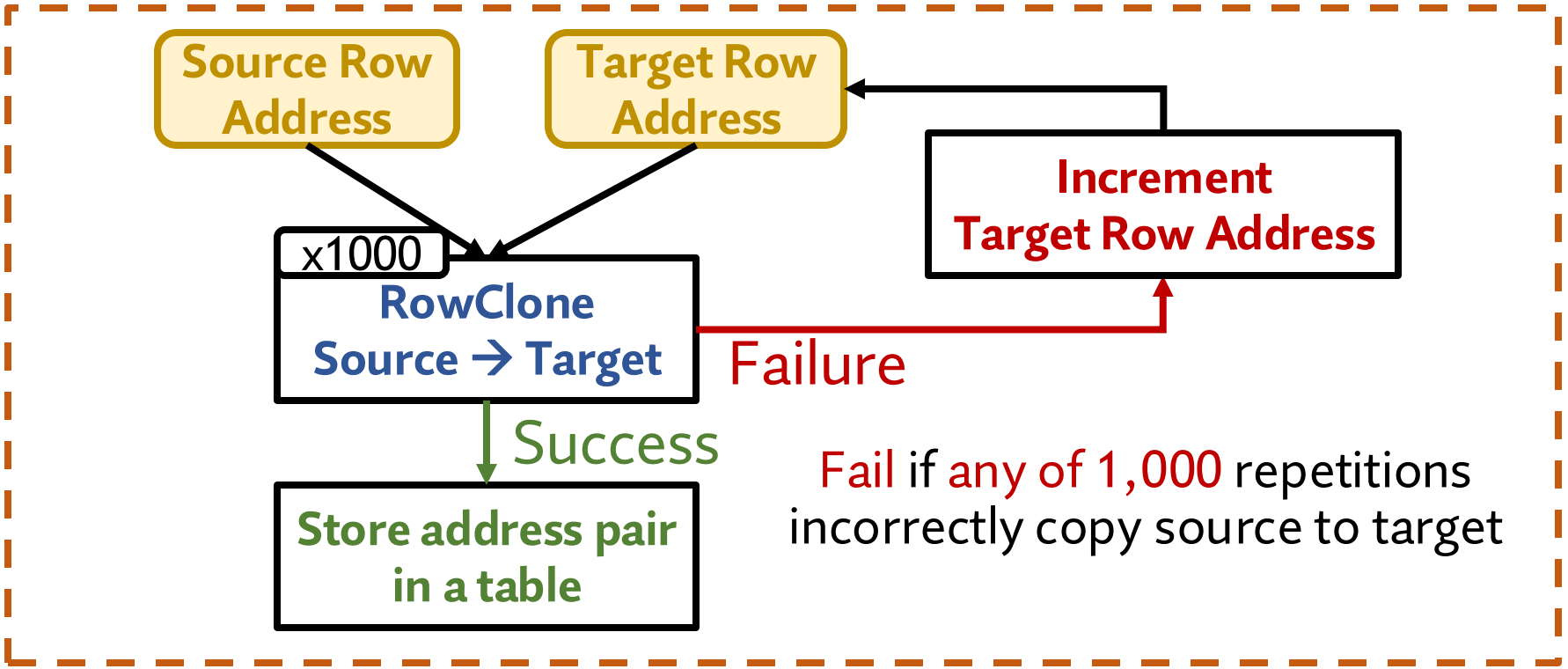
RowClone Memory Allocation Constraints



4 Memory allocation that works for RowClone

RowClone Implementation

- Initialization/one-time cost to find reliable source-target row pairs for RowClone



Repeated for all source row addresses in the DRAM bank

RowClone Evaluation

- Evaluated system configurations
  - EasyDRAM – No Time Scaling
  - EasyDRAM – Time Scaling
  - Ramulator 2.0 (configured to model EasyDRAM's system)

- Workloads
  - Copy: Copy N-bytes from source array to target array
  - Init: Initialize an N-byte array with data
  - CPU version: Use load/store instructions
  - RowClone version: Use in-DRAM row copy
    - RowClone – No Flush: Data is up to date in DRAM (no cached copies)
    - RowClone – CLFLUSH: Cached data must be flushed/invalidated to DRAM

RowClone Evaluation

- Evaluated system configurations
  - EasyDRAM – No Time Scaling
  - EasyDRAM – Time Scaling
  - Ramulator 2.0 (configured to model EasyDRAM's system)

- Workloads
  - Copy: Copy N-bytes from source array to target array
  - Init: Initialize an N-byte array with data
  - CPU version: Use load/store instructions
  - RowClone version: Use in-DRAM row copy
    - RowClone – No Flush: Data is up to date in DRAM (no cached copies)
    - RowClone – CLFLUSH: Cached data must be flushed/invalidated to DRAM

RowClone Evaluation

- Evaluated system configurations
  - EasyDRAM – No Time Scaling
  - EasyDRAM – Time Scaling
  - Ramulator 2.0 (configured to model EasyDRAM's system)

- Workloads
  - Copy: Copy N-bytes from source array to target array
  - Init: Initialize an N-byte array with data
  - CPU version: Use load/store instructions
  - RowClone version: Use in-DRAM row copy
    - RowClone – No Flush: Data is up to date in DRAM (no cached copies)
    - RowClone – CLFLUSH: Cached data must be flushed/invalidated to DRAM

https://arxiv.org/pdf/2506.10441

Key Takeaways

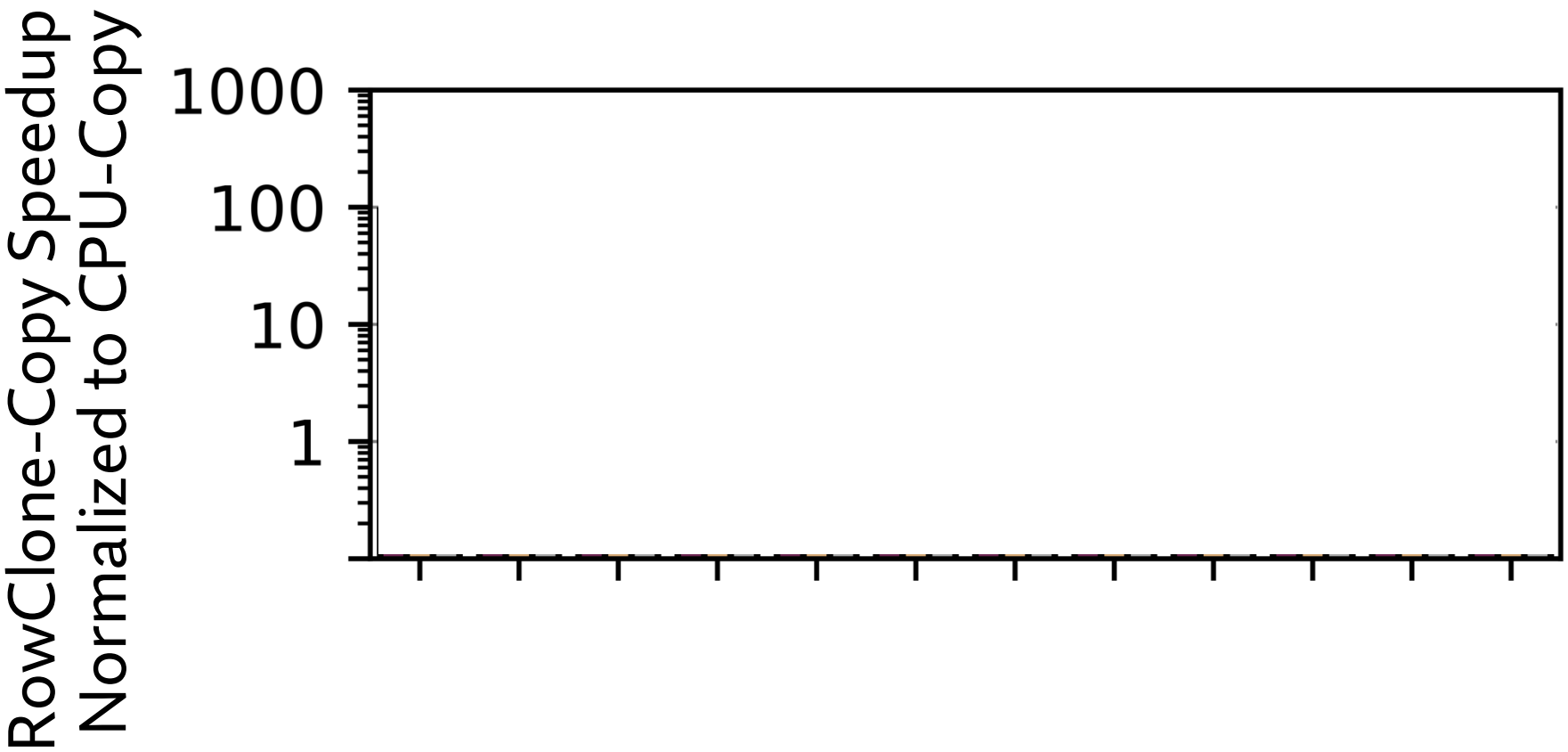
Takeaway 1

DRAM technique evaluation platforms that
do not faithfully model a modern processor
report high benefits in favor of DRAM techniques

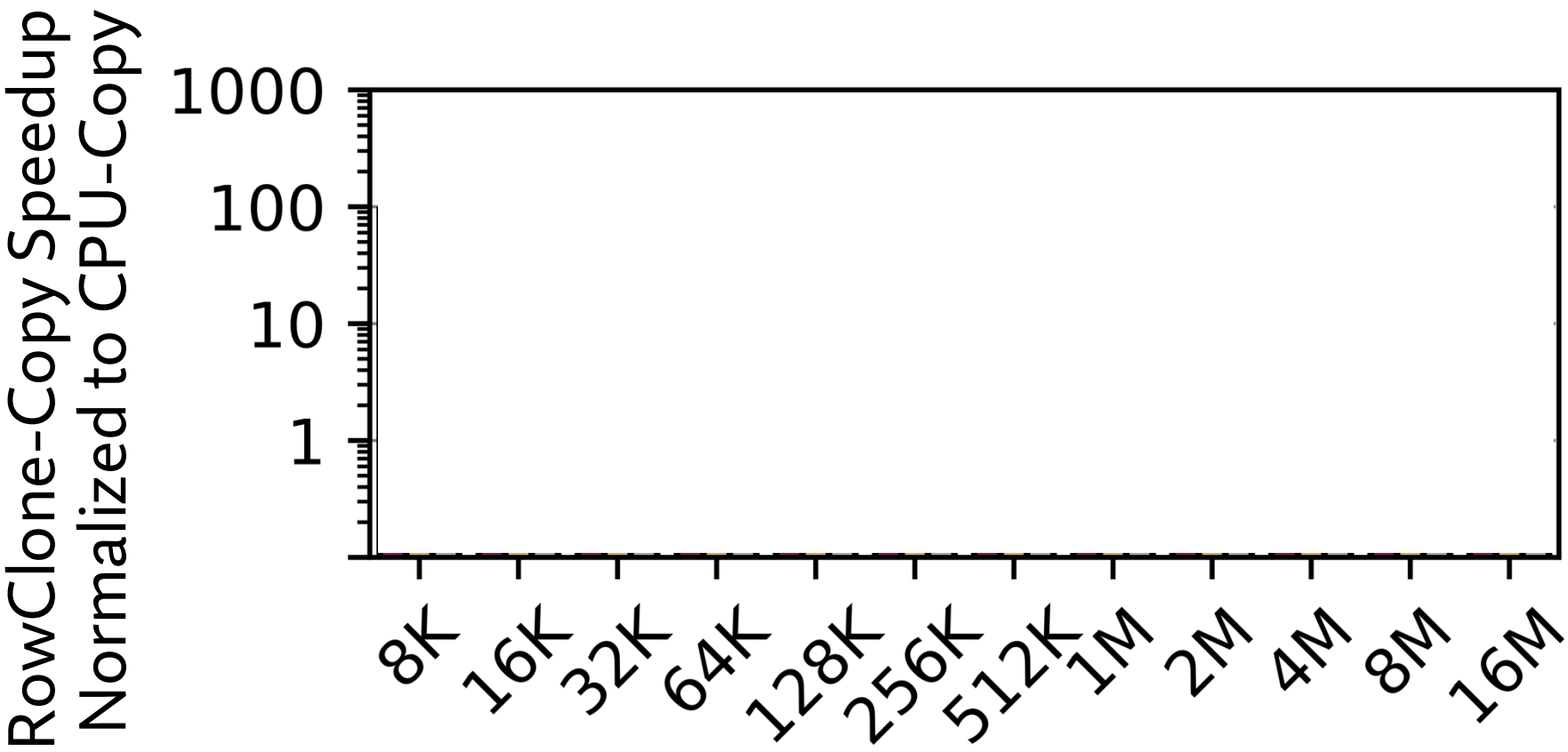
Takeaway 2

RowClone (still) significantly improves performance
when compared to a modern CPU baseline

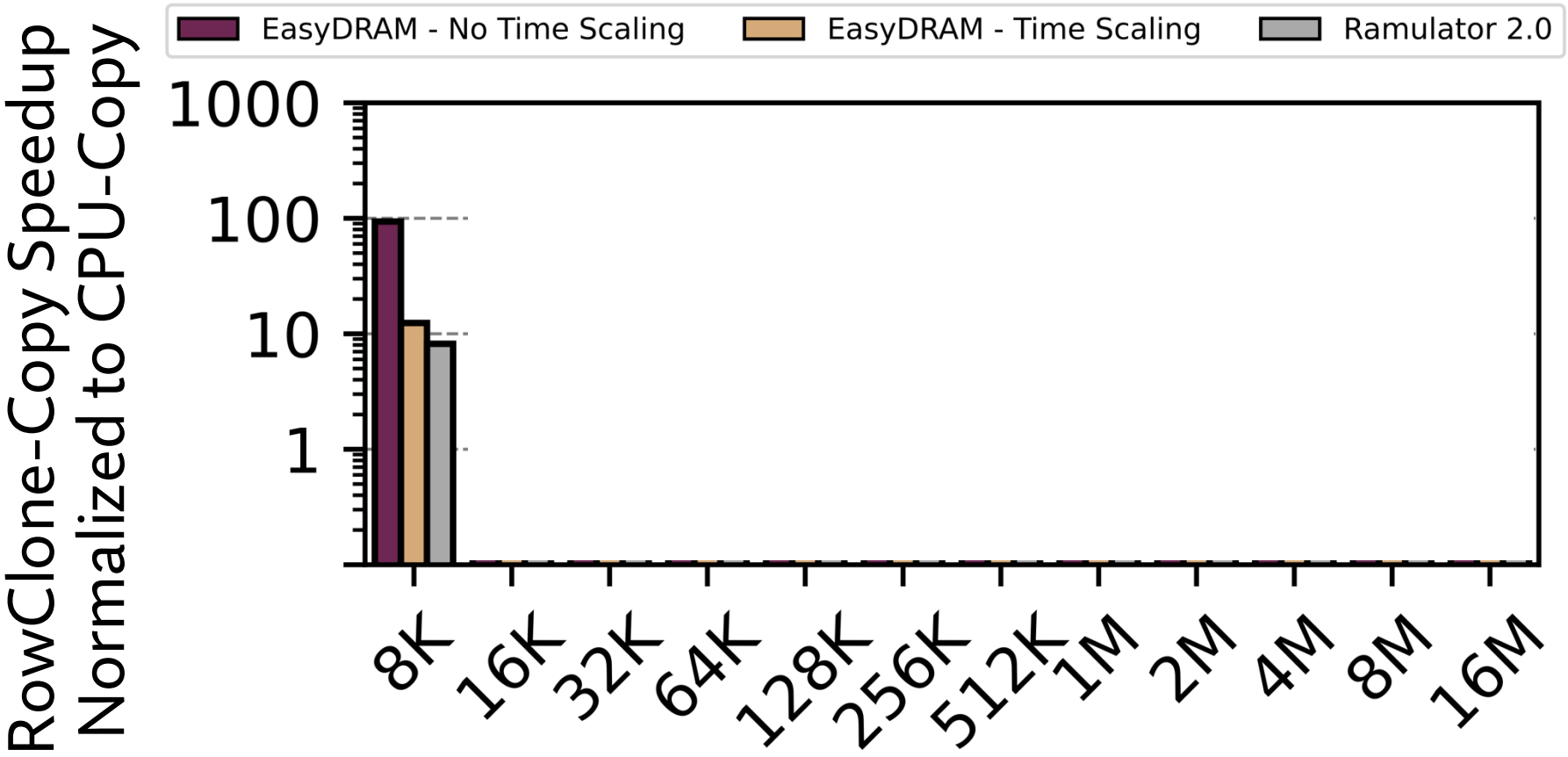
Execution Time (RowClone – No Flush)



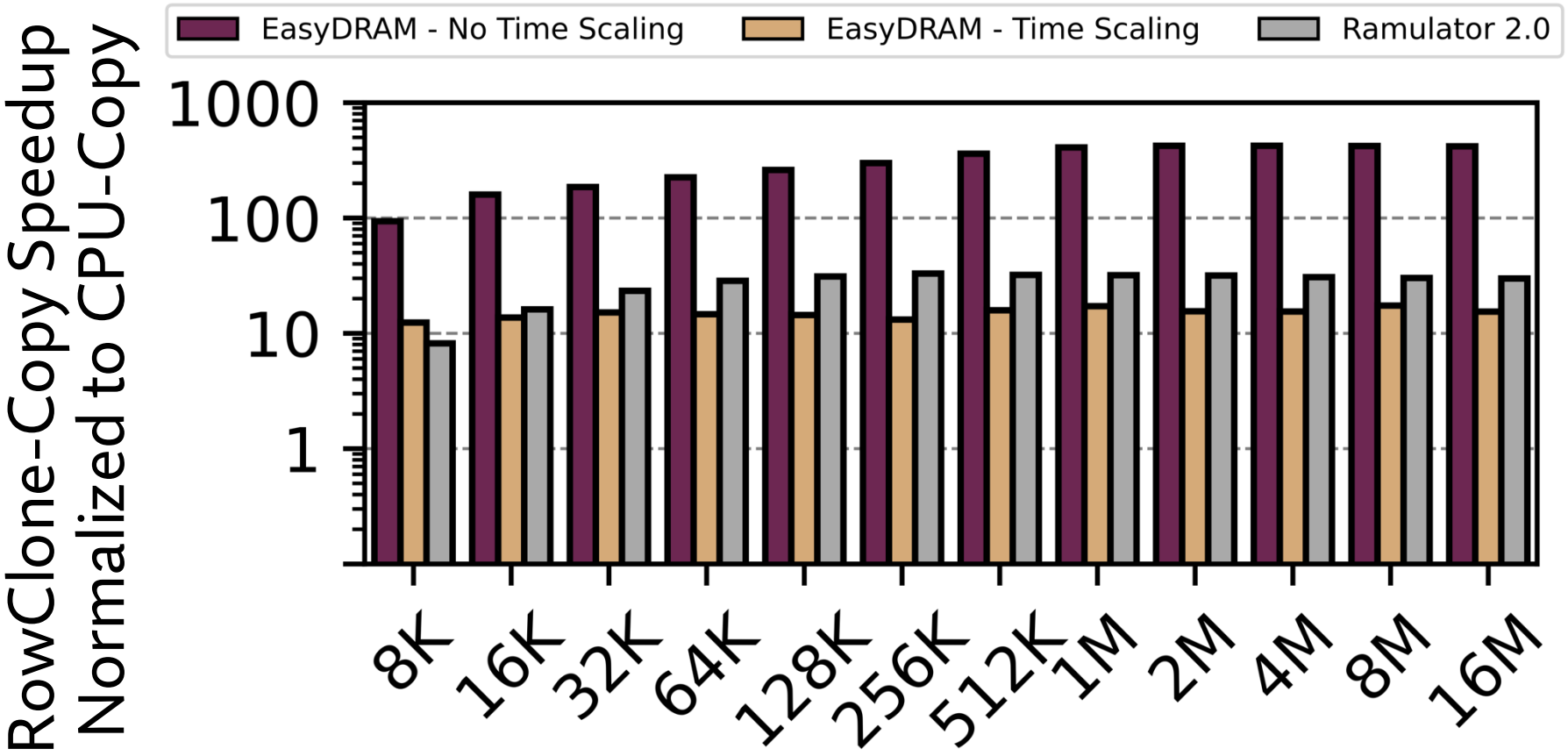
Execution Time (RowClone – No Flush)



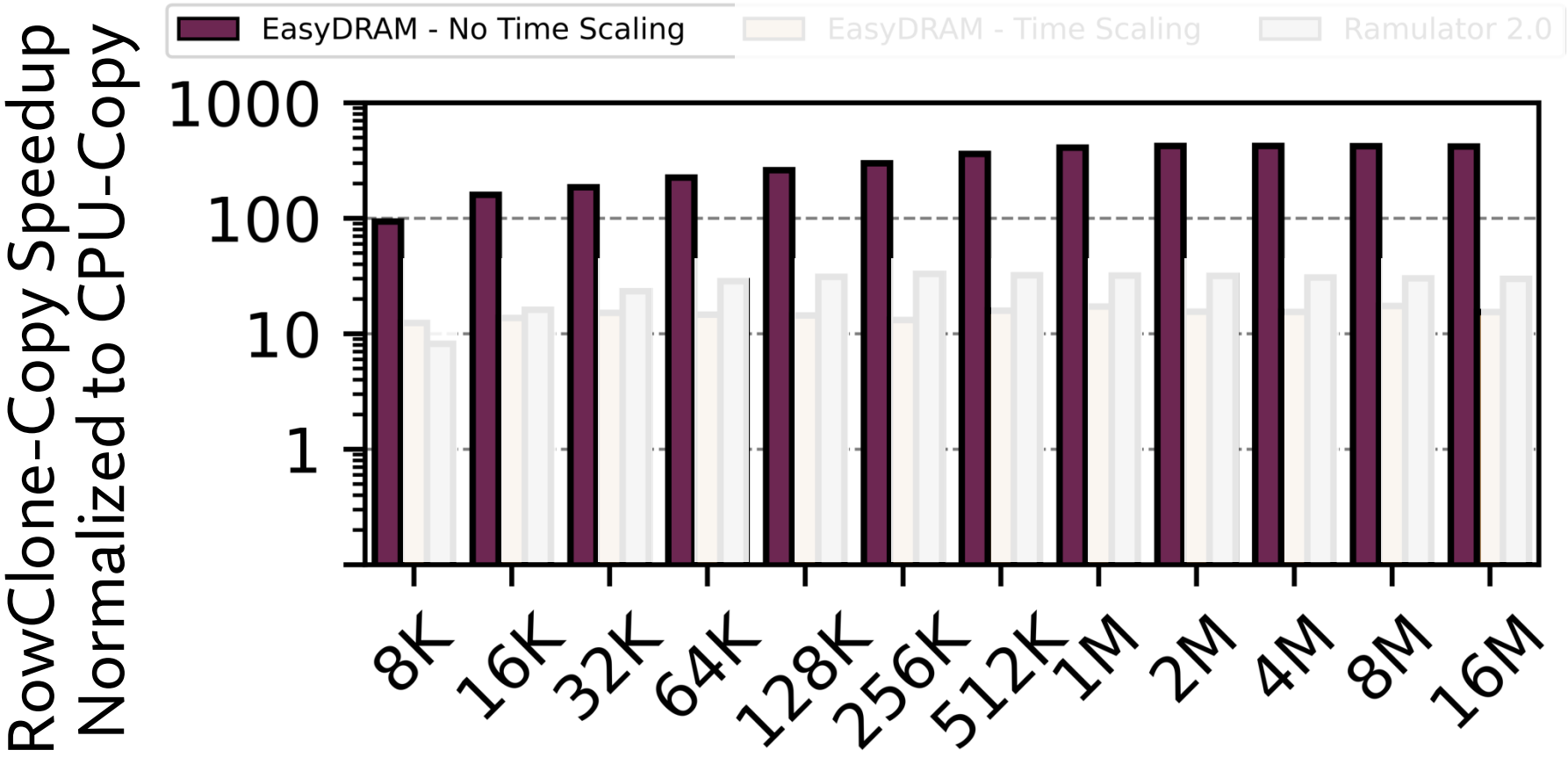
Execution Time (RowClone – No Flush)



Execution Time (RowClone – No Flush)

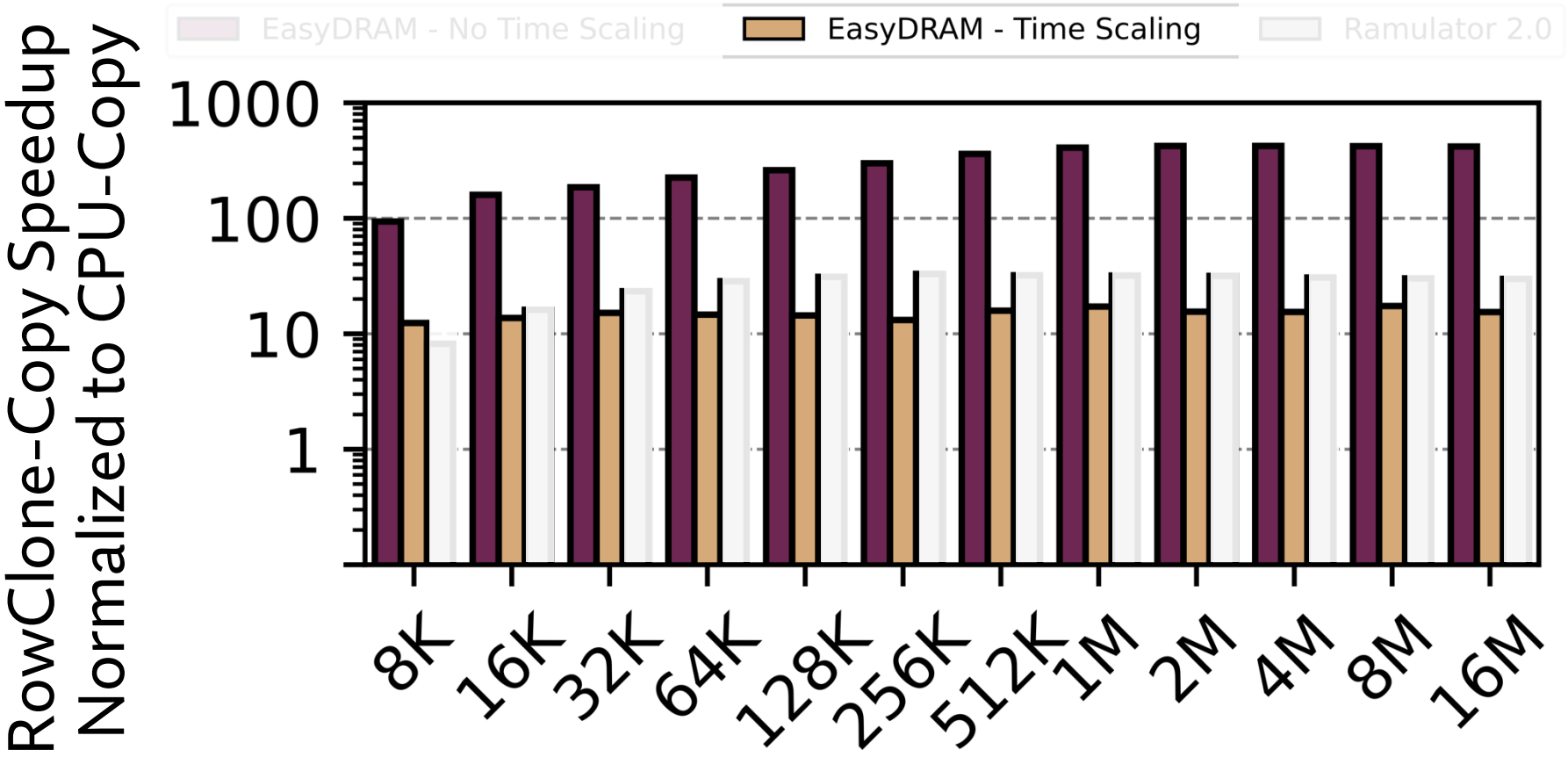


Execution Time (RowClone – No Flush)



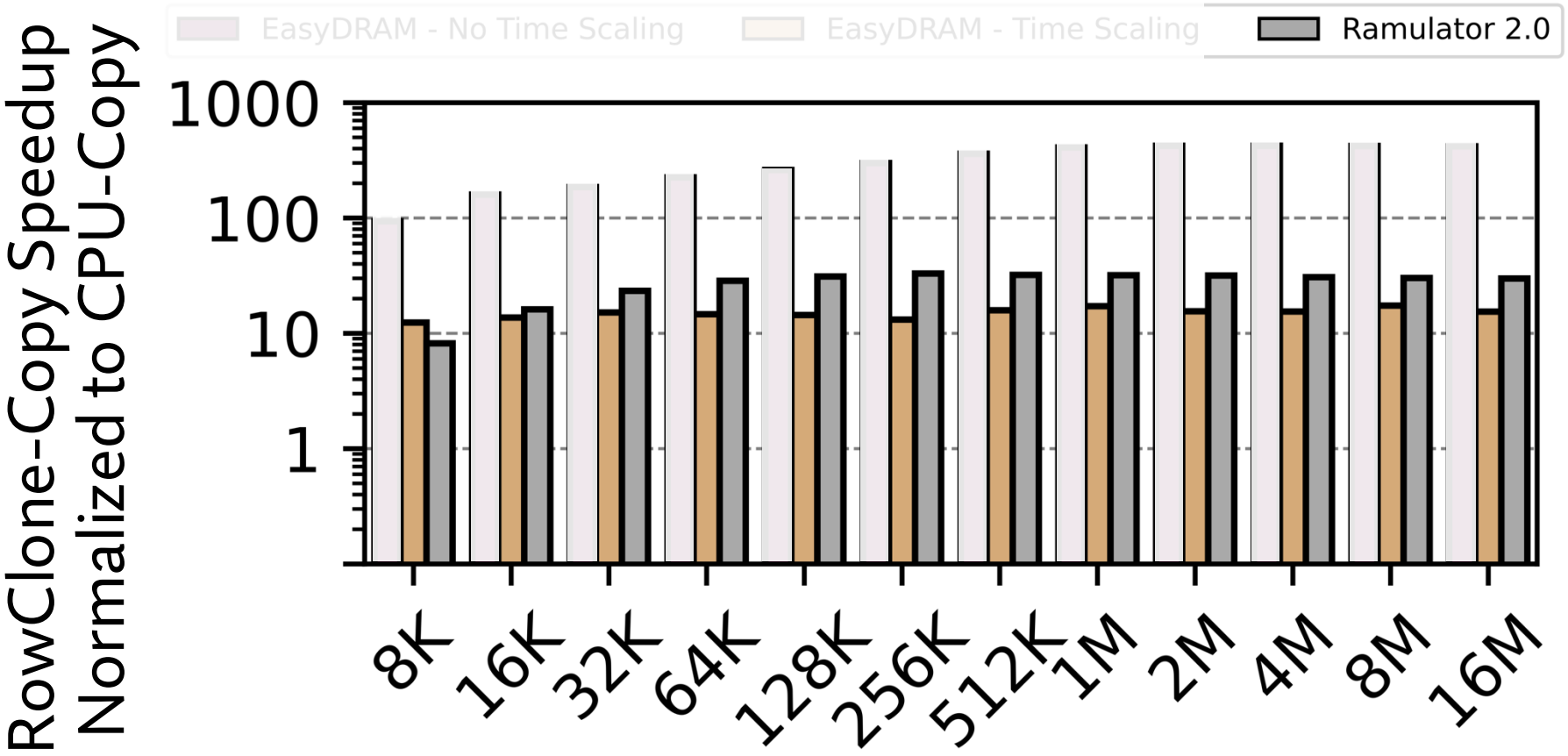
EasyDRAM – No Time Scaling demonstrates
307x (423x) average (maximum) speedup
across tested array sizes

Execution Time (RowClone – No Flush)



EasyDRAM –Time Scaling demonstrates
15.0x (17.4x) average (maximum) speedup
across tested array sizes

Execution Time (RowClone – No Flush)



Ramulator 2.0 demonstrates
27.2x (33.0x) average (maximum) speedup
across tested array sizes

Key Takeaways

Takeaway 1

DRAM technique evaluation platforms that
do not faithfully model a modern processor
report high benefits in favor of DRAM techniques

Takeaway 2

RowClone (still) significantly improves performance
when compared to a modern CPU baseline

Two Case Studies

- Realistic performance evaluations of

1. **In-DRAM Row Copy (RowClone)**
2. DRAM Access Latency Reduction

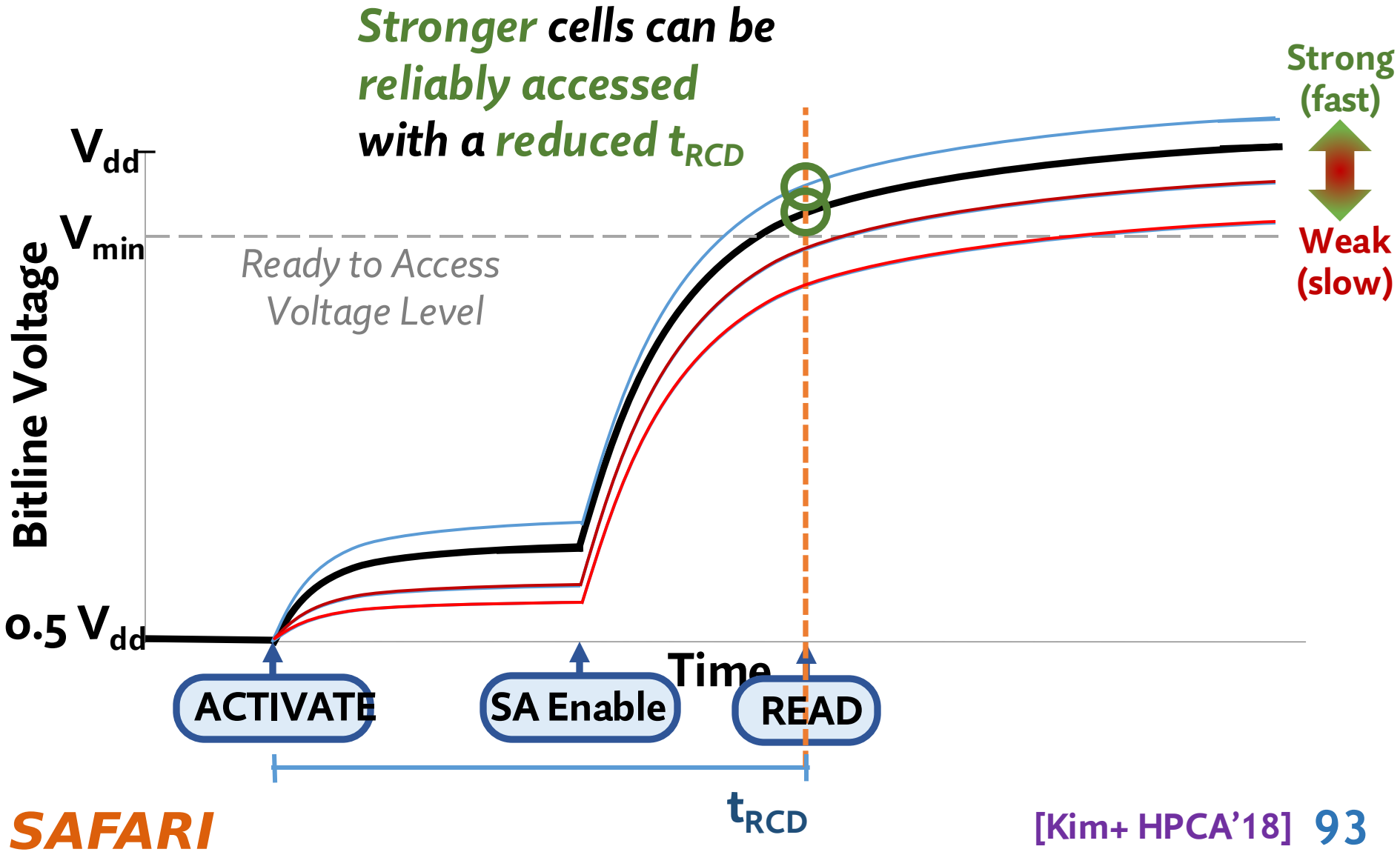
Two Case Studies

- Realistic performance evaluations of

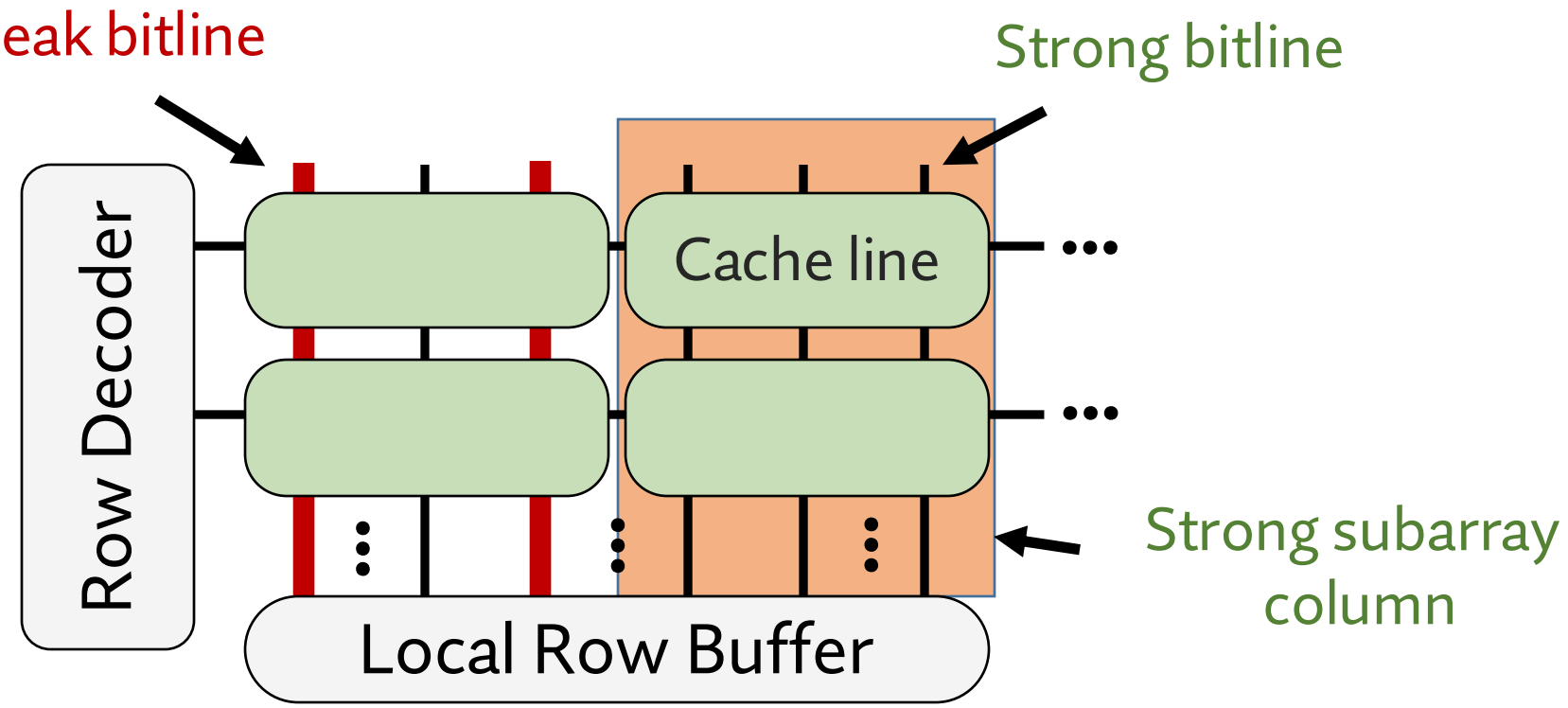
1 In-DRAM Row Copy (RowClone)

2 **DRAM Access Latency Reduction**

DRAM Access Latency Reduction: Key Idea



Solar-DRAM: Variable Latency Cache Lines (VLC)



Identifies subarray columns comprised of strong bitlines

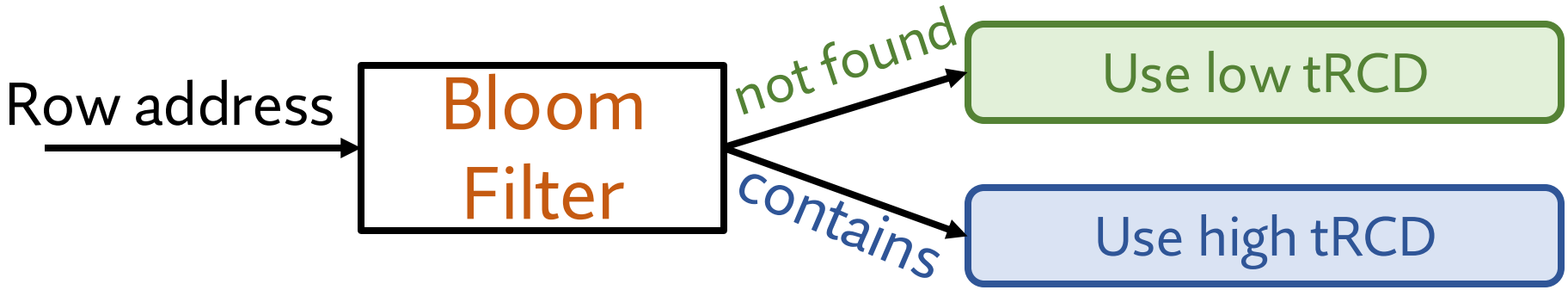
Access cache lines in strong subarray columns with a reduced t_{RCD}

SolarDRAM VLC Implementation

- Storing the minimum t_{RCD} value of all cache lines is not scalable
  - 64M such cache lines in a 4 GiB module

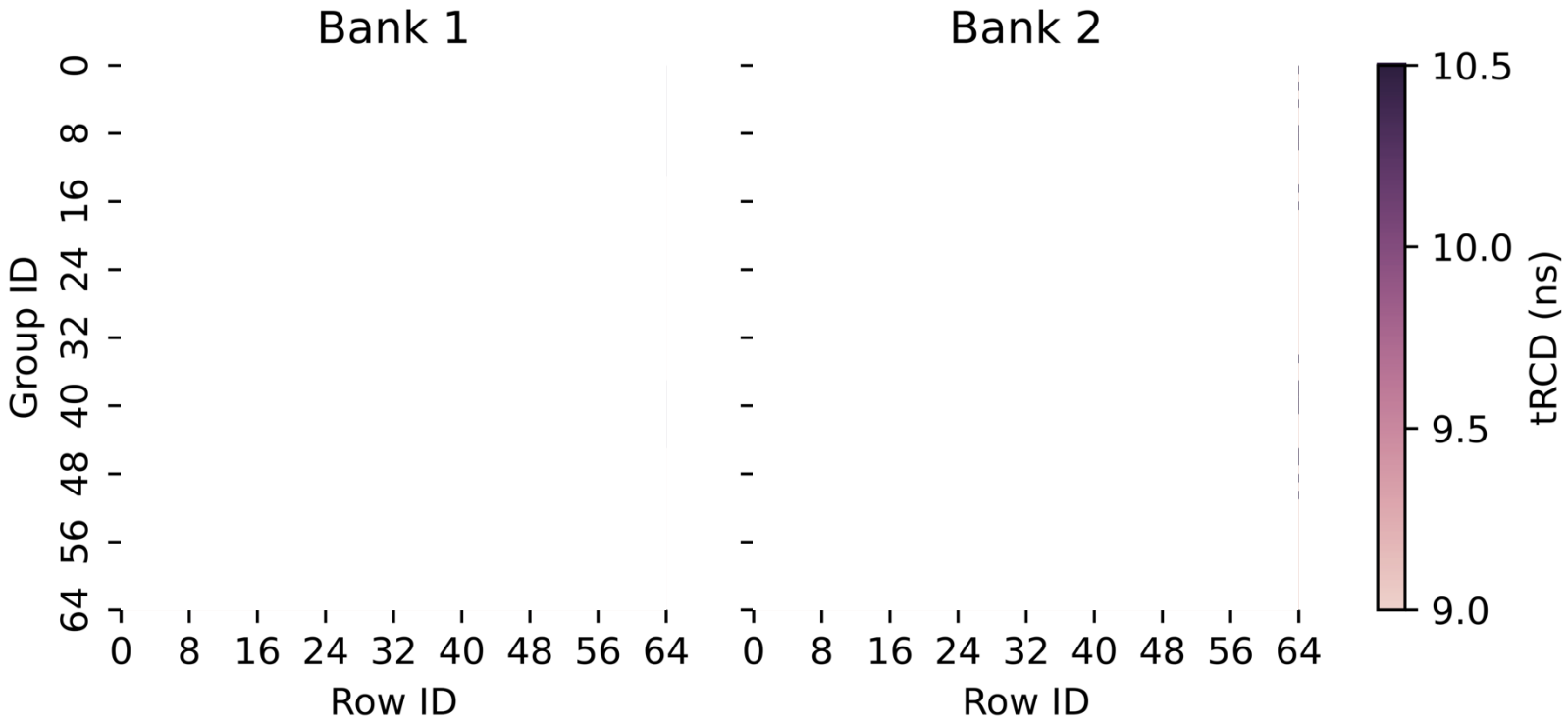
- Reduce storage overhead

1. Use t_{RCD} of weakest cache line in a row for that row
   - Factor of 128 storage reduction
2. Use storage-efficient Bloom filter to store weak row IDs



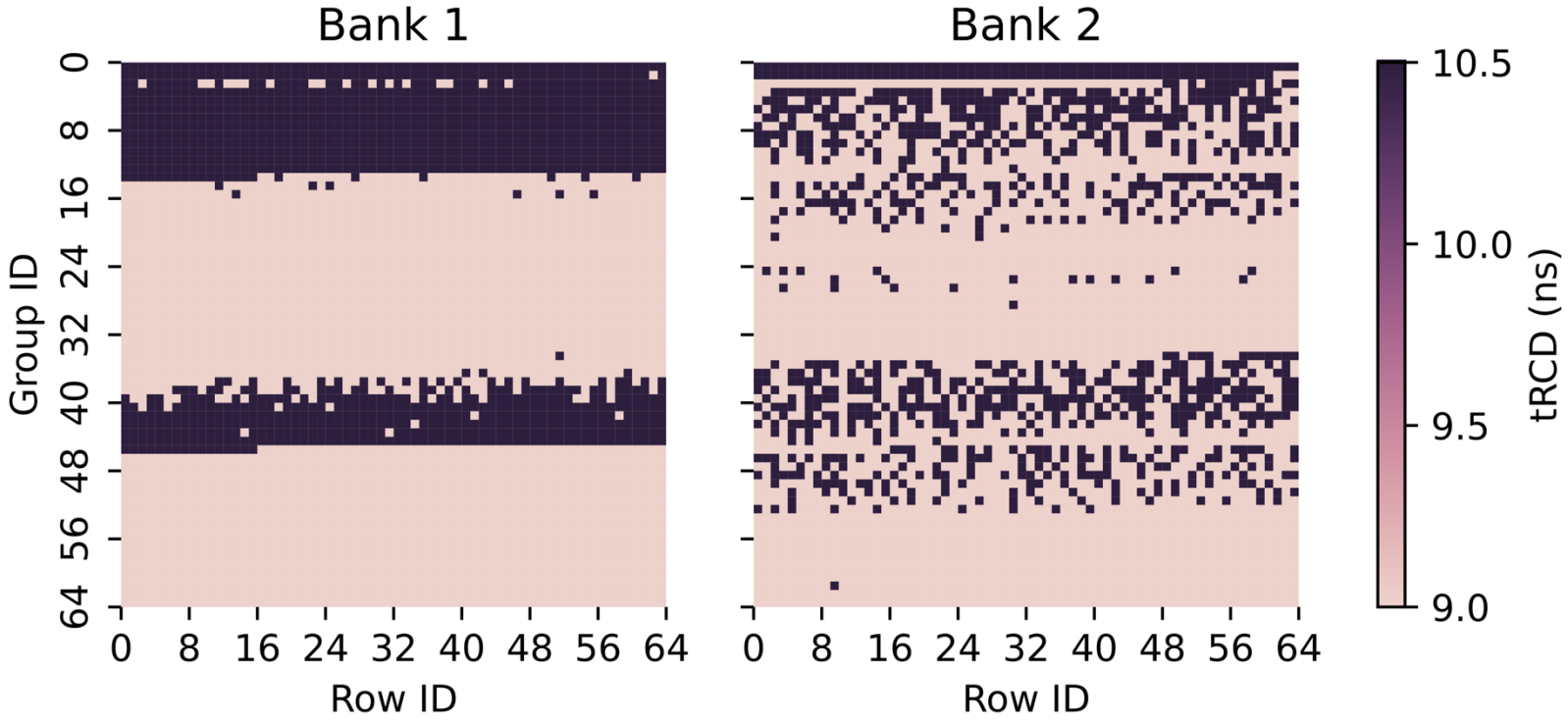
EasyDRAM VLC Profile

- Characterize 4K DRAM rows in 2 banks for access latency failures
  - Nominal t_{RCD} for our module is 13.5 ns

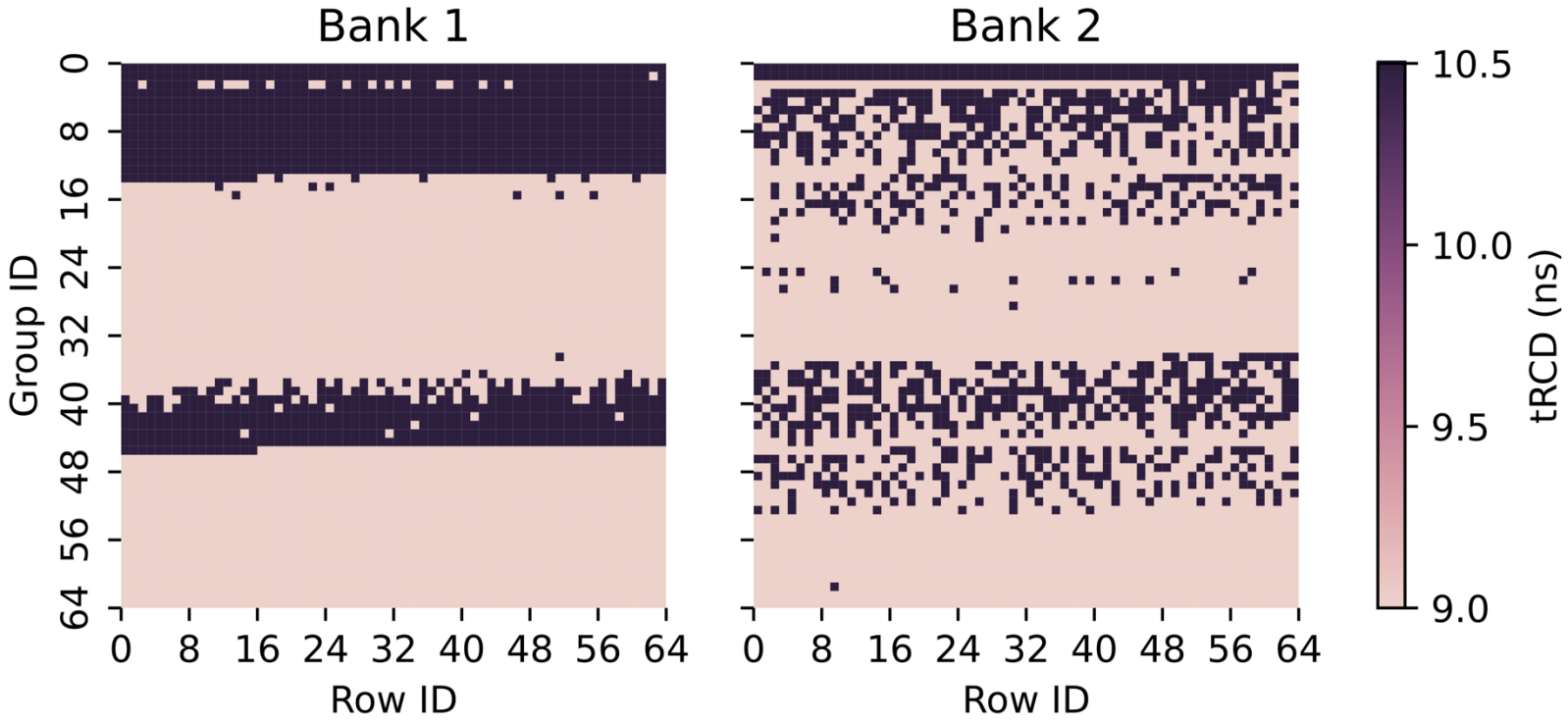


EasyDRAM VLC Profile

- Characterize 4K DRAM rows in 2 banks for access latency failures
  - Nominal t_{RCD} for our module is 13.5 ns



EasyDRAM VLC Profile



All cache lines reliably operate at < nominal tRCD

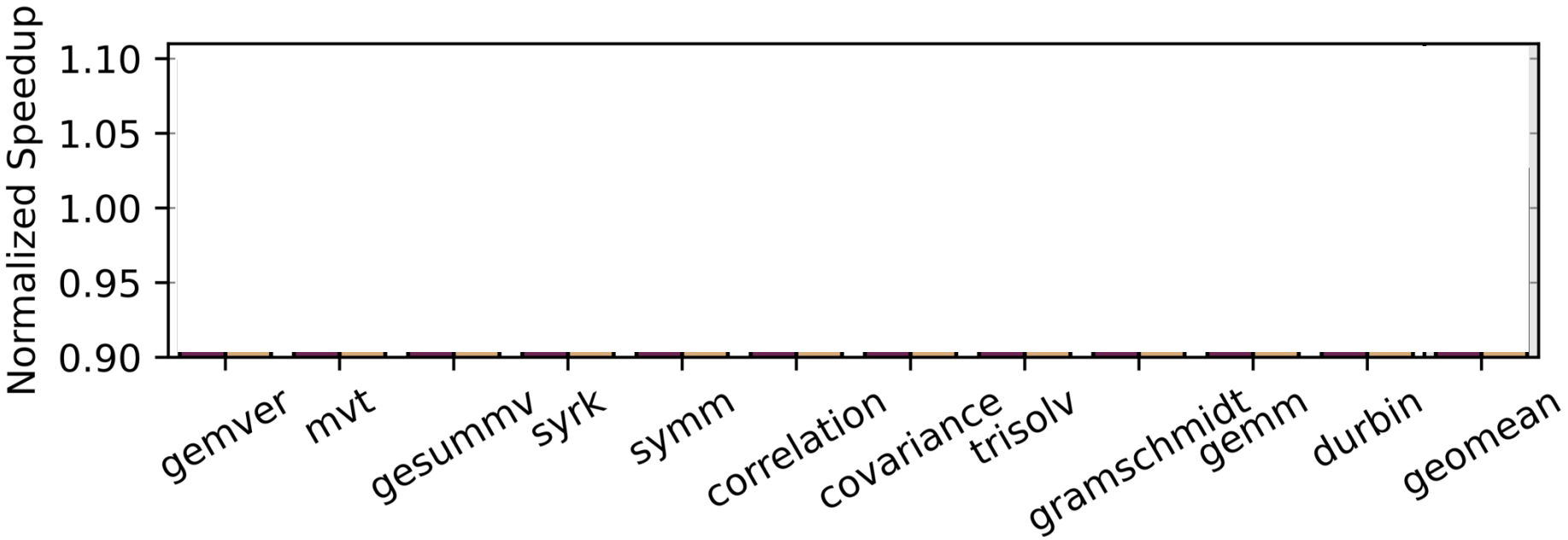
The fraction (84.5%) of "stronger" cache lines (t_{RCD} = 9.0 ns) far exceed that of "weaker" cache lines (t_{RCD} = 10.5 ns)

SolarDRAM Evaluation

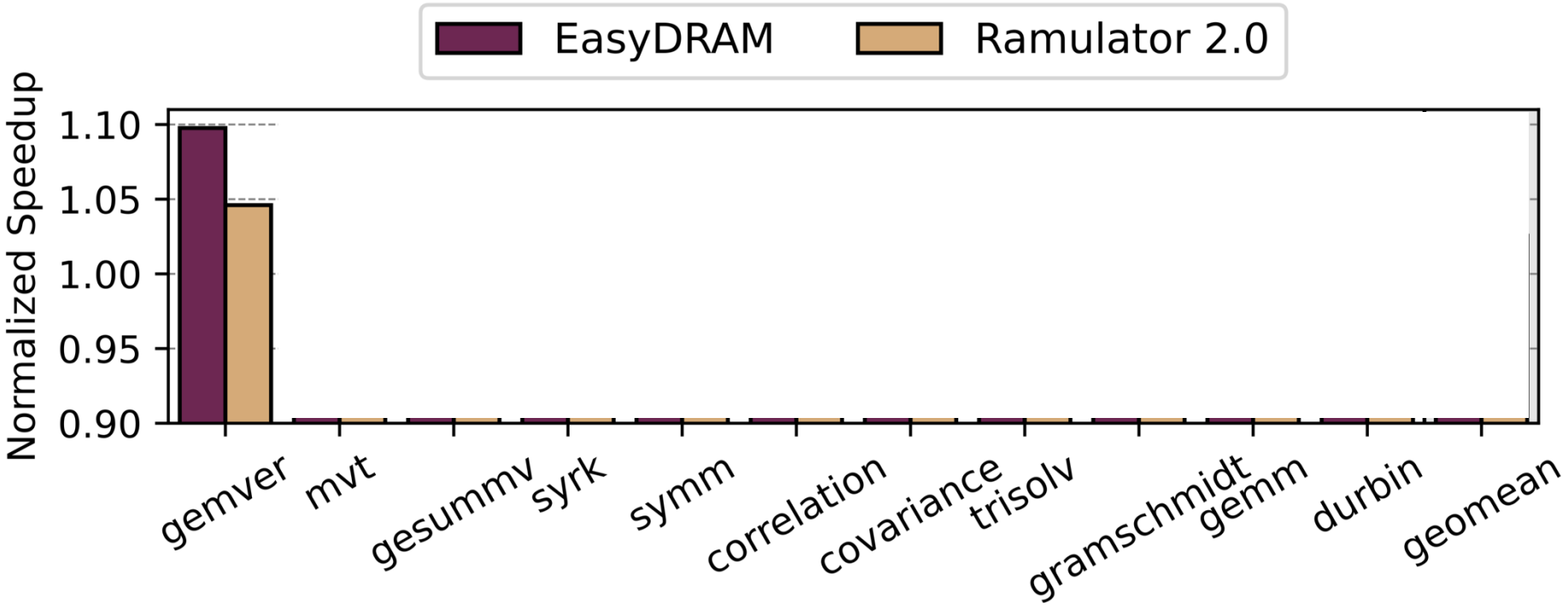
- Evaluated system configurations
  - EasyDRAM – Time Scaling
  - Ramulator 2.0 (configured to model EasyDRAM's system)

- PolyBench workload suite
  - Executed end-to-end in EasyDRAM
  - First 500M instructions executed for Ramulator 2.0

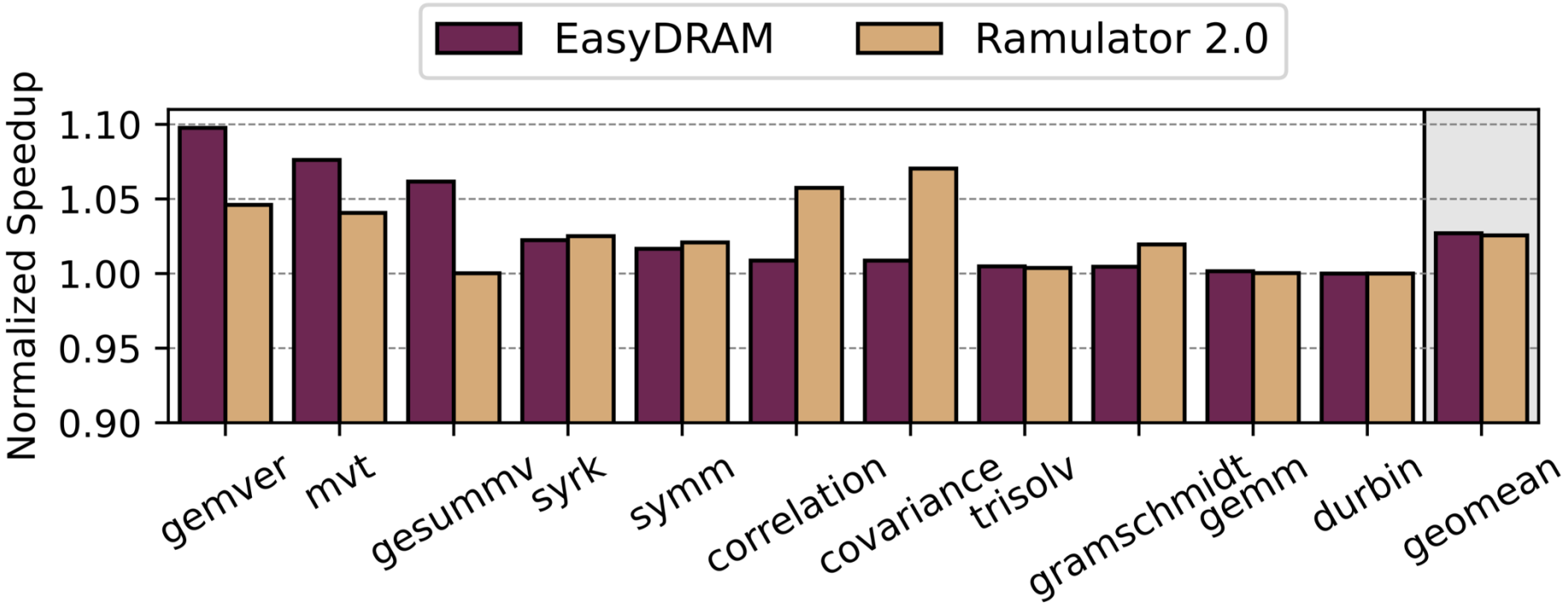
SolarDRAM Results



SolarDRAM Results



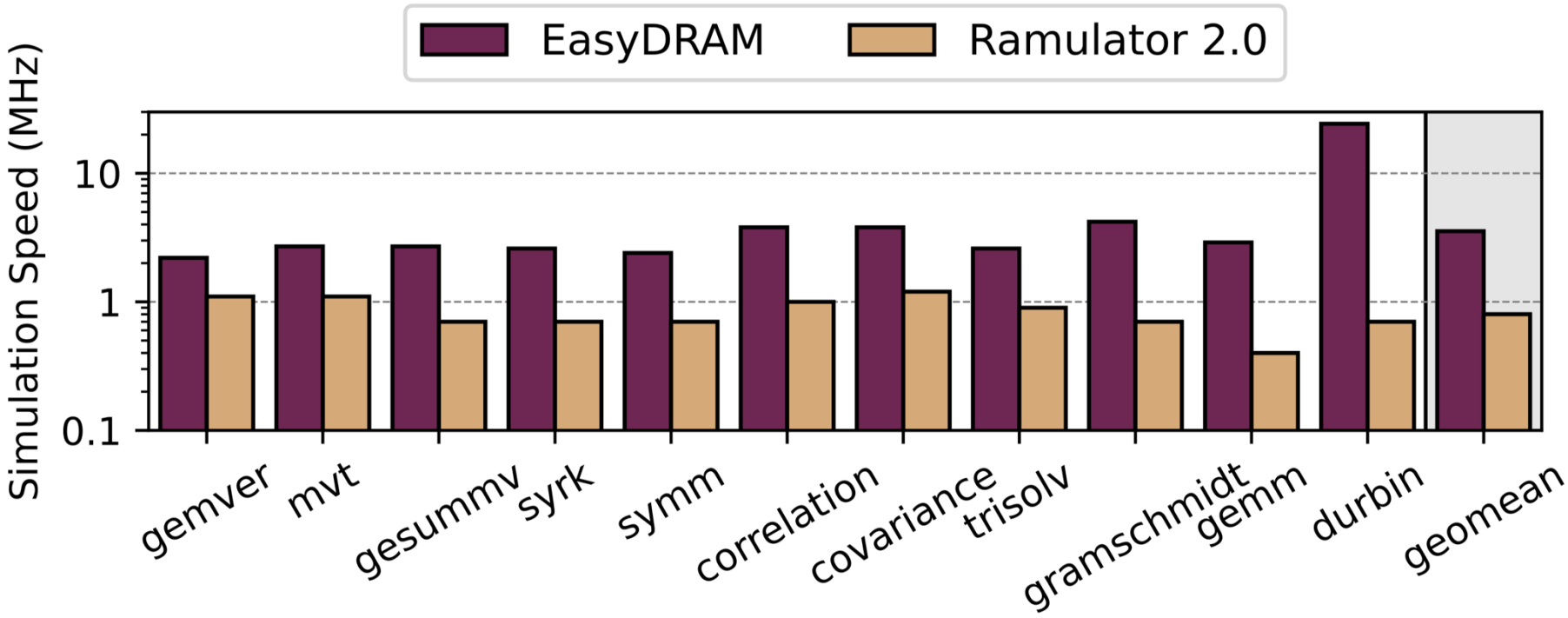
SolarDRAM Execution Time



SolarDRAM improves system performance
by 2.8% (9.8%) on average (maximum) across tested workloads

Benefits expected to increase with memory intensity

SolarDRAM Simulation Time



Average (maximum) simulation speed
5.9x (20.3x) faster than Ramulator 2.0

Talk Outline

I. Background

II. Motivation

III. EasyDRAM

IV. Case Studies

V. Conclusion

EasyDRAM Conclusion

EasyDRAM

Easy to use FPGA-based infrastructure
to accurately evaluate DRAM techniques

Easy to use (C++) programmable memory controller

Time Scaling for accurate evaluation

We conduct two case studies to showcase EasyDRAM's ease of use and ability to accurately evaluate DRAM techniques

1. In-DRAM row copy (RowClone)
2. DRAM access latency reduction (SolarDRAM)

We hope EasyDRAM enables
innovative ideas in memory system design to rapidly come to fruition

Extended Version on arXiv

https://arxiv.org/pdf/2506.10441

arXiv > cs > arXiv:2506.10441

Search... All fields Search

Help | Advanced Search

Computer Science > Hardware Architecture

[Submitted on 12 Jun 2025]

EasyDRAM: An FPGA-based Infrastructure for Fast and Accurate End-to-End Evaluation of Emerging DRAM Techniques

Oğuzhan Canpolat, Ataberk Olgun, David Novo, Oğuz Ergin, Onur Mutlu

DRAM is a critical component of modern computing systems. Recent works propose numerous techniques (that we call DRAM techniques) to enhance DRAM-based computing systems' throughput, reliability, and computing capabilities (e.g., in-DRAM bulk data copy). Evaluating the system-wide benefits of DRAM techniques is challenging as they often require modifications across multiple layers of the computing stack. Prior works propose FPGA-based platforms for rapid end-to-end evaluation of DRAM techniques on real DRAM chips. Unfortunately, existing platforms fall short in two major aspects: (1) they require deep expertise in hardware description languages, limiting accessibility; and (2) they are not designed to accurately model modern computing systems.

We introduce EasyDRAM, an FPGA-based framework for rapid and accurate end-to-end evaluation of DRAM techniques on real DRAM chips. EasyDRAM overcomes the main drawbacks of prior FPGA-based platforms with two key ideas. First, EasyDRAM removes the need for hardware description language expertise by enabling developers to implement DRAM techniques using a high-level language (C++). At runtime, EasyDRAM executes the software-defined memory system design in a programmable memory controller. Second, EasyDRAM tackles a fundamental challenge in accurately modeling modern systems: real processors typically operate at higher clock frequencies than DRAM, a disparity that is difficult to replicate on FPGA platforms. EasyDRAM addresses this challenge by decoupling the processor-DRAM interface and advancing the system state using a novel technique we call time scaling, which faithfully captures the timing behavior of the modeled system. We believe and hope that EasyDRAM will enable innovative ideas in memory system design to rapidly come to fruition. To aid future research EasyDRAM implementation is open sourced at this https URL.

Comments: Extended version of our publication at DSN 2025
Subjects: **Hardware Architecture (cs.AR)**
Cite as: arXiv:2506.10441 **[cs.AR]**
(or arXiv:2506.10441v1 **[cs.AR]** for this version)
https://doi.org/10.48550/arXiv.2506.10441

Access Paper:
View PDF
HTML (experimental)
TeX Source
Other Formats
view license

Current browse context: **cs.AR**
< prev | next >
new | recent | 2025-06
Change to browse by: cs

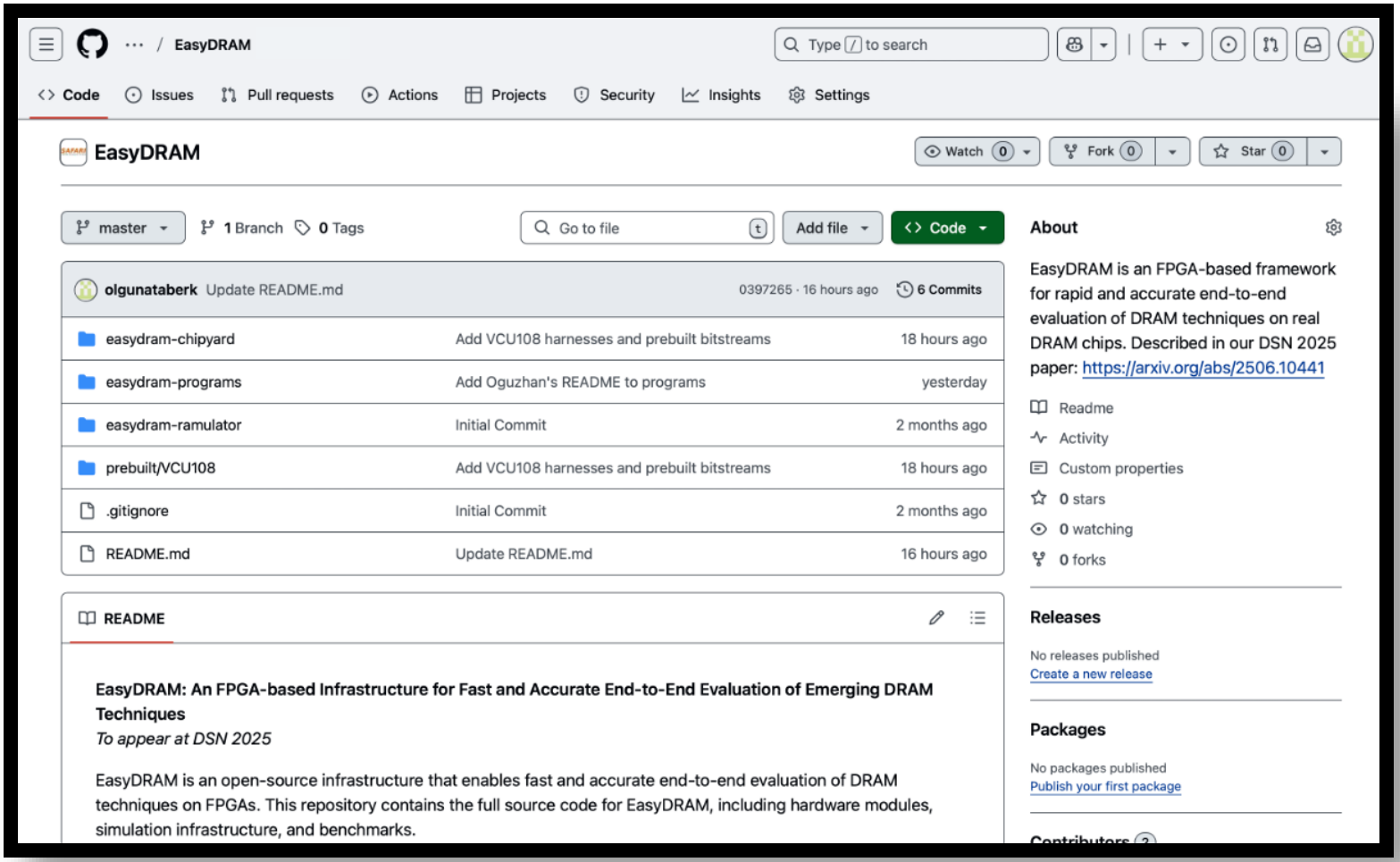
References & Citations
NASA ADS
Google Scholar
Semantic Scholar

Export BibTeX Citation

Bookmark

EasyDRAM is Open Source

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



EasyDRAM

An FPGA-based Infrastructure for Fast and Accurate End-to-End Evaluation of Emerging DRAM Techniques

Oğuzhan Canpolat Ataberk Olgun
David Novo Oğuz Ergin Onur Mutlu

https://arxiv.org/abs/2506.10441

SAFARI

ETH zürich TOBB ETÜ Ekonomi ve Teknoloji Üniversitesi LIRMM جامعة الشارقة UNIVERSITY OF SHARJAH

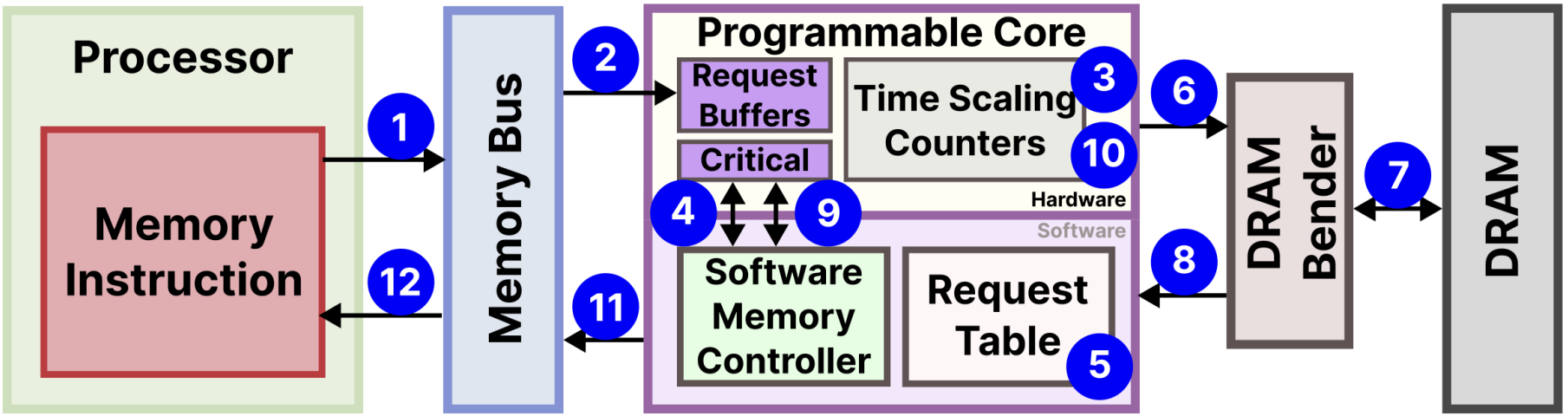
EasyDRAM
Backup Slides

EasyDRAM vs. Other Platforms

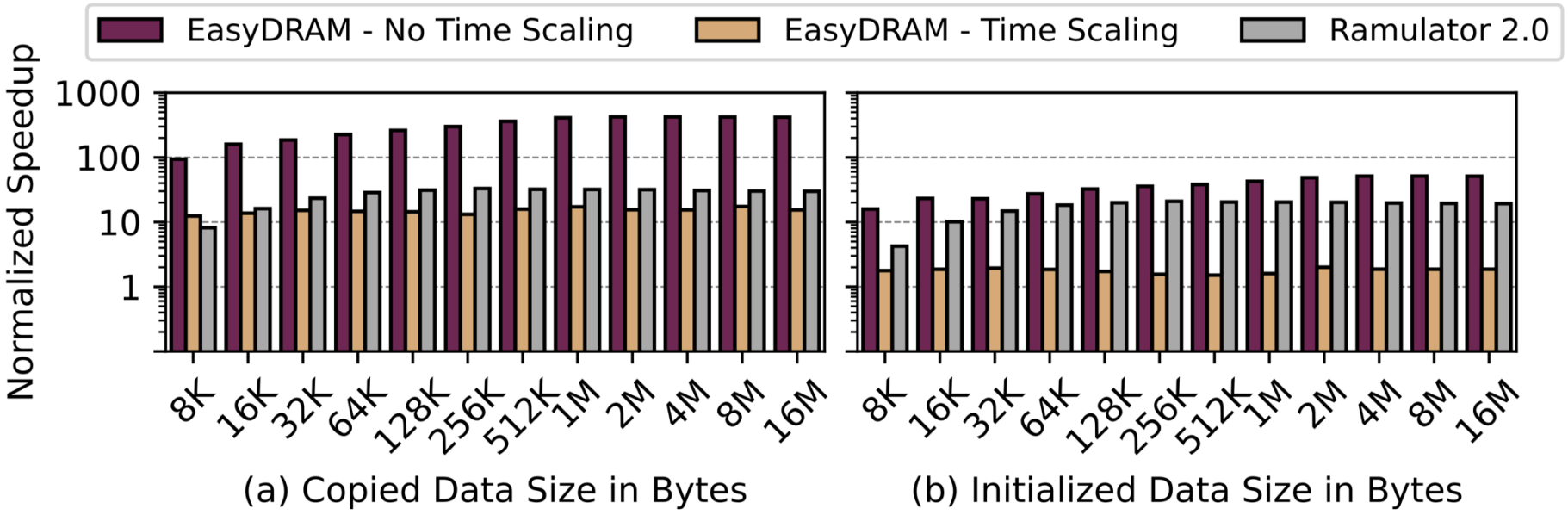
Table 1: Comparison of EasyDRAM with related state-of-the-art prototyping and evaluation platforms.

Platforms	Interface with real DRAM chips	Flexible memory controller for DRAM techniques	Evaluated CPU clock cycles per second	Accurate performance evaluation	Easily configurable system
Commercial computing systems	✓	✗	Billions	✓	✗
Software simulators [44–48]	✗	✓(C/C++)	≈ 10K - ≈ 1M	✓	✓
FPGA-based simulators [56, 57]	✗	✗	≈ 4M - ≈ 100M	✓	✓
DRAM testing platforms [41, 55]	DDR3/4	✗	N/A	✗	✗
FPGA-based emulators [40, 54, 58, 59]	DDR3/4	HDL	50M - 200M	✗	✓
EasyDRAM (this work)	DDR4	✓(C/C++)	≈ 10M	✓	✓

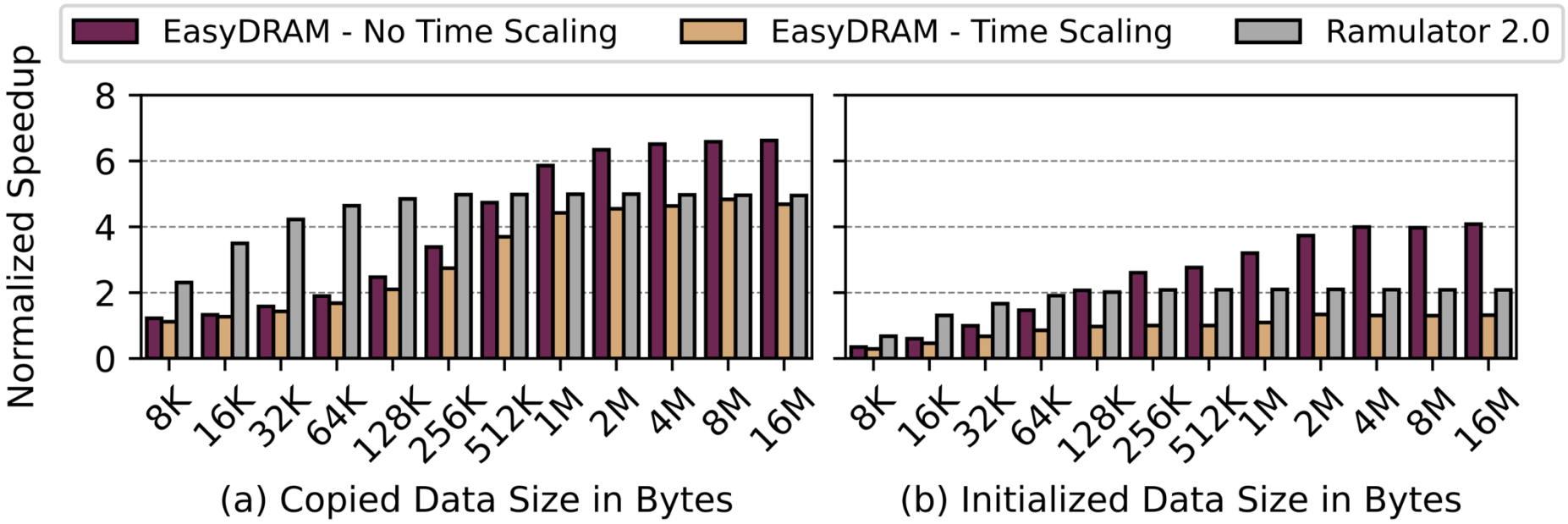
Lifetime of a Memory Request



RowClone and Init – No Flush



RowClone and Init – CLFLUSH



How Does a Program Issue RowClone Operations?

- We use memory-mapped registers and an internal DRAM technique instruction encoding

HLS vs. EasyDRAM Programmable Memory Controller

- HLS still defines hardware
- There will come a time for you to reason about hardware design
- You can still build the programmable memory ctrl. using HLS
- Not a replacement for the programmable memory controller

Why not use the hard ARM core in the FPGA?

- Good idea
- Does **not** complement EasyDRAM as a whole

+ Higher performance core
- Not in every FPGA

What is new?

- EasyDRAM as a tool is new
- Reproduction of prior results

- This platform allows users to easily prototype fault tolerant DRAM technique implementations

- Reliability – performance tradeoff for DRAM
  - Typically abstracted away by the DRAM standard

- System designers can make conscious design decisions to extract the most performance out of their memory systems