

P&S Ramulator

Designing and Evaluating Memory Systems and
Modern Software Workloads with Ramulator

Hasan Hassan

Prof. Onur Mutlu

ETH Zürich

Spring 2022

9 March 2022

P&S Ramulator: Content

- You will learn in detail how **modern memory systems** operate
- You will **design new DRAM and memory controller mechanisms** for improving overall system **performance, energy** consumption, and **reliability**
- You will **simulate and understand** the memory system behavior of **modern workloads** such as machine learning, graph analytics, genome analysis

P&S Ramulator: Key Takeaways

- This P&S is aimed at improving your
 - ❑ **Knowledge** in Computer Architecture and Memory Systems
 - ❑ **Technical skills** in simulating memory systems
 - ❑ **Critical thinking and analysis**
 - ❑ **Interaction** with a nice group of researchers
 - ❑ Familiarity with key **research directions**
 - ❑ **Technical presentation** of your project

P&S Ramulator: Key Goal

Learn how state-of-the-art memory controllers operate, design new DRAM and memory controller mechanisms, and evaluate your mechanisms using simulation

Prerequisites of the Course

- Digital Design and Computer Architecture (or equivalent course)
- A good knowledge in C/C++ programming language
- Interest in making things efficient and solving problems
- Interest in understanding software development and hardware design, and their interaction

Course Info: Who Are We? (I)

■ Onur Mutlu

- ❑ Full Professor @ ETH Zurich ITET (INFK), since September 2015
- ❑ Strecker Professor @ Carnegie Mellon University ECE/CS, 2009-2016, 2016-...
- ❑ PhD from UT-Austin, worked at Google, VMware, Microsoft Research, Intel, AMD
- ❑ <https://people.inf.ethz.ch/omutlu/>
- ❑ omutlu@gmail.com (Best way to reach me)
- ❑ <https://people.inf.ethz.ch/omutlu/projects.htm>



■ Research and Teaching in:

- ❑ Computer architecture, computer systems, hardware security, bioinformatics
- ❑ Memory and storage systems
- ❑ Hardware security, safety, predictability
- ❑ Fault tolerance
- ❑ Hardware/software cooperation
- ❑ Architectures for bioinformatics, health, medicine
- ❑ ...

Course Info: Who Are We? (II)

- Lead Supervisor:

- Hasan Hassan

- Supervisors:

- Geraldo de Oliveira

- Giray Yaglikci

- Ataberk Olgun

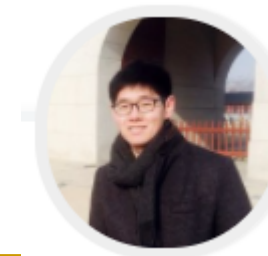
- Haocong Luo

- Jeremie Kim

- Minesh Patel

- Get to know us and our research

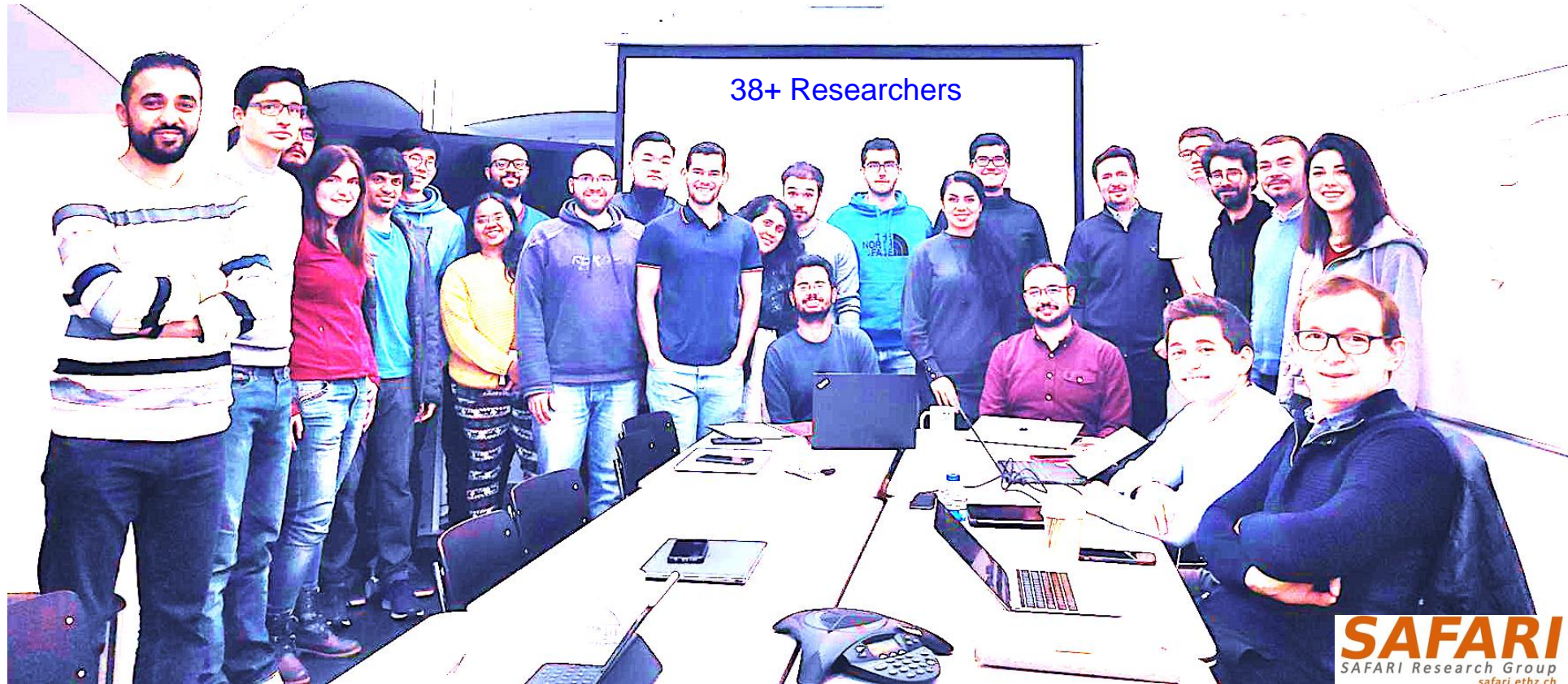
- <https://safari.ethz.ch/group-members>



Onur Mutlu's SAFARI Research Group

Computer architecture, HW/SW, systems, bioinformatics, security, memory

<https://safari.ethz.ch/safari-newsletter-april-2020/>



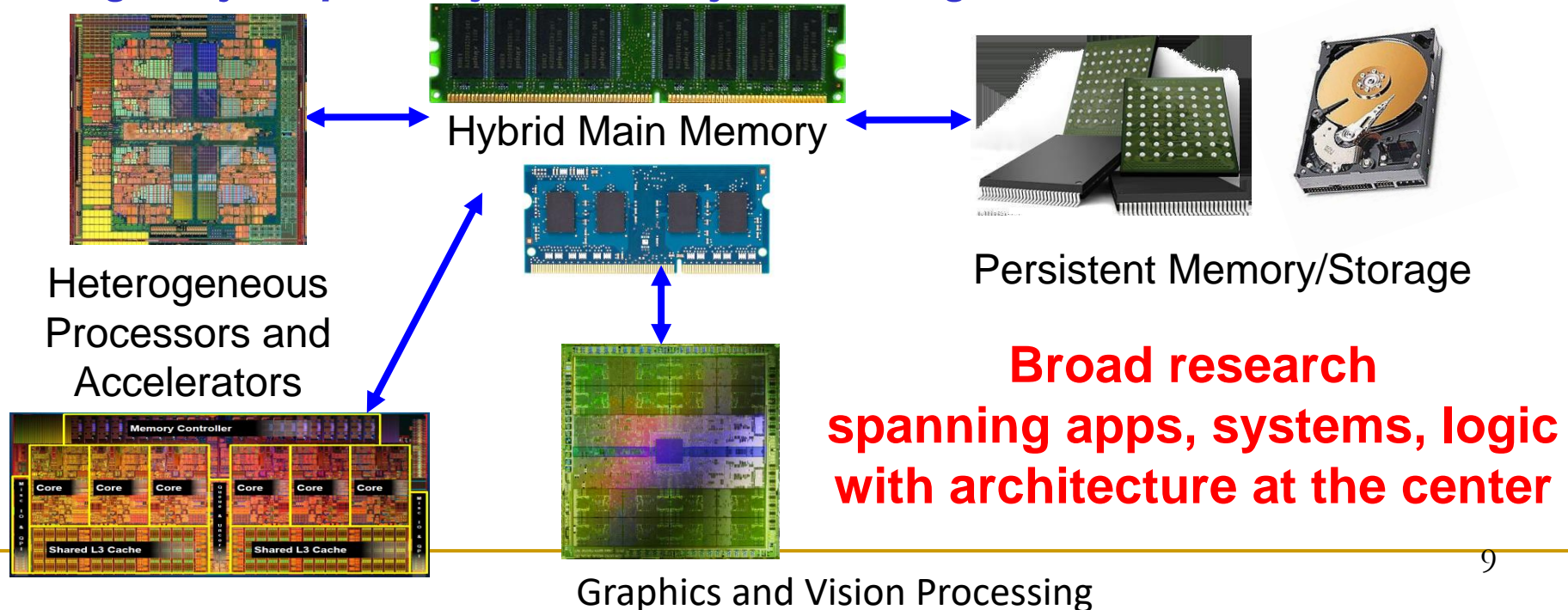
Think BIG, Aim HIGH!

<https://safari.ethz.ch>

Current Research Focus Areas

Research Focus: Computer architecture, HW/SW, bioinformatics

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



Course Info: How About You?

- Let us know your background, interests
- Why did you join this P&S?
- Please submit HW0

Course Requirements and Expectations

- Attendance required for all meetings
- Study the learning materials
- Each student will carry out a hands-on project
 - Build, implement, code, and design with close engagement from the supervisors
- Participation
 - Ask questions, contribute thoughts/ideas
 - Read relevant papers

We will help in all projects!

If your work is really good, you may get it published!

Course Website

- https://safari.ethz.ch/projects_and_seminars/doku.php?id=ramulator
- Useful information about the course
- Check your email frequently for announcements

Meeting 1

Learning materials:

- The github version of Ramulator: <https://github.com/CMU-SAFARI/ramulator>
- Original Ramulator paper: https://people.inf.ethz.ch/omutlu/pub/ramulator_dram_simulator-ieee-cal15.pdf
- An example study of modern workloads and DRAM architectures using Ramulator: https://people.inf.ethz.ch/omutlu/pub/Workload-DRAM-Interaction-Analysis_sigmetrics19_pomacs19.pdf
- An example recent study of a new DRAM architecture using Ramulator: https://people.inf.ethz.ch/omutlu/pub/CLR-DRAM_capacity-latency-reconfigurable-DRAM_isca20.pdf
- An example recent study of a new virtual memory system architecture using Ramulator: https://people.inf.ethz.ch/omutlu/pub/VBI-virtual-block-interface_isca20.pdf
- Three examples of new ideas enabled by Ramulator based evaluation:
 - https://people.inf.ethz.ch/omutlu/pub/rowclone_micro13.pdf
 - https://people.inf.ethz.ch/omutlu/pub/salp-dram_isca12.pdf
 - https://people.inf.ethz.ch/omutlu/pub/raidr-dram-refresh_isca12.pdf

Meeting 2 (TBD)

- We will **announce the projects** and will give you some description about them
- You will have a week to submit your project preferences
- The supervisors would like to help you with selecting a project that matches your interests, skills, and background
- It is important that you **study the learning materials** before our next meeting!

Tentative Weekly Schedule

- Week 1 – Logistics & Intro to Simulating Memory Systems using Ramulator
- Week 2 – Tutorial on Using Ramulator | Available Projects
- Week 3 – BlockHammer [**HPCA'21**] | **PU**
- Week 4 – CLR-DRAM [**ISCA'20**] | **PU**
- Week 5 – CODIC [**ISCA'21**] | **PU**
- Week 6 – SIMDram [**ASPLOS'17**] | **PU**
- Week 7 – DAMOV [**IEEE Access'21**] | **PU**
- Week 8 – Synchron [**HPCA'21**] | **PU**
- Week 9+ - **PU**

- **PU** = Project Updates

- **How about meeting every Thursday at 18:00?**

Performance Assessment

We expect you to:

- **Learn** how DRAM operates and how to analyze performance of memory systems using simulation
- **Achieve the goals** of your project
- **Deliver** your **code and results** with sufficient documentation
- Prepare a **final presentation** and present your work to SAFARI

An Introduction to Simulating Memory Systems with Ramulator

Motivation

- DRAM and Memory Controller landscape is changing
- Many new and upcoming standards
- Many new controller designs
- A fast and easy-to-extend simulator is very much needed

<i>Segment</i>	<i>DRAM Standards & Architectures</i>
Commodity	DDR3 (2007) [14]; DDR4 (2012) [18]
Low-Power	LPDDR3 (2012) [17]; LPDDR4 (2014) [20]
Graphics	GDDR5 (2009) [15]
Performance	eDRAM [28], [32]; RLDram3 (2011) [29]
3D-Stacked	WIO (2011) [16]; WIO2 (2014) [21]; MCDRAM (2015) [13]; HBM (2013) [19]; HMC1.0 (2013) [10]; HMC1.1 (2014) [11]
Academic	SBA/SSA (2010) [38]; Staged Reads (2012) [8]; RAIDR (2012) [27]; SALP (2012) [24]; TL-DRAM (2013) [26]; RowClone (2013) [37]; Half-DRAM (2014) [39]; Row-Buffer Decoupling (2014) [33]; SARP (2014) [6]; AL-DRAM (2015) [25]

Table 1. Landscape of DRAM-based memory

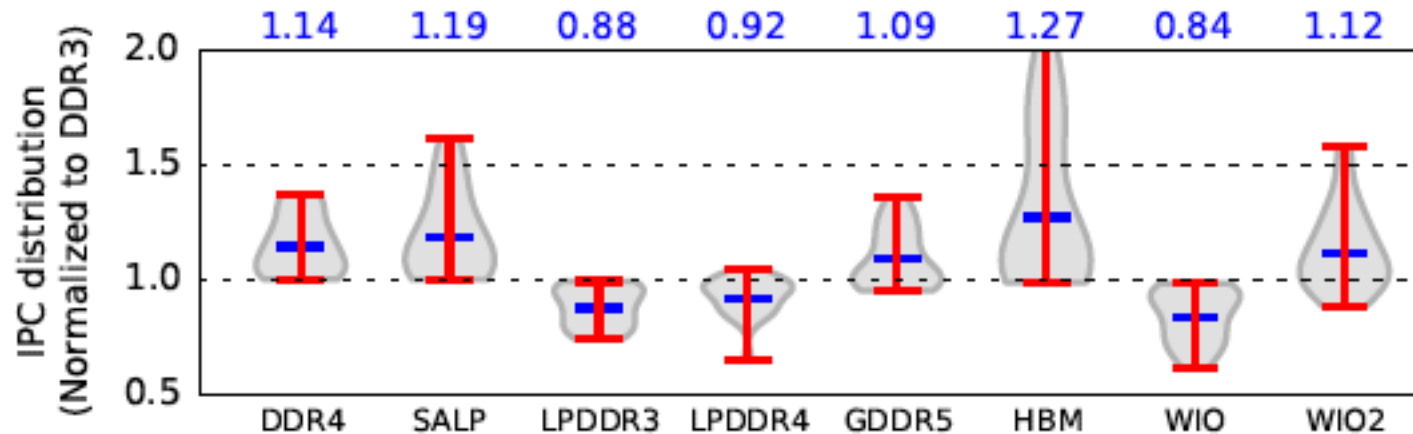
Ramulator: A Fast and Extensible DRAM Simulator

- Provides out-of-the box support for many DRAM standards:
 - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, HMC, and academic proposals (SALP, AL-DRAM, TLDRAM, RowClone, and SARP)
 - Models timing of non-volatile memories (PCM, STT-MRAM)
- Supports multiple scheduling and row buffer management policies
- Modular and extensible to different standards
- Can be paired with other simulators, e.g., gem5 and DRAMPower
- Written in C++11
- ~2.5X faster than fastest open-source simulator

<i>Simulator</i> (clang -O3)	<i>Cycles (10⁶)</i>		<i>Runtime (sec.)</i>		<i>Req/sec (10³)</i>		<i>Memory</i> (MB)
	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	
Ramulator	652	411	752	249	133	402	2.1
DRAMSim2	645	413	2,030	876	49	114	1.2
USIMM	661	409	1,880	750	53	133	4.5
DrSim	647	406	18,109	12,984	6	8	1.6
NVMain	666	413	6,881	5,023	15	20	4,230.0

Case Study: Comparison of DRAM Standards

<i>Standard</i>	<i>Rate (MT/s)</i>	<i>Timing (CL-RCD-RP)</i>	<i>Data-Bus (Width×Chan.)</i>	<i>Rank-per-Chan</i>	<i>BW (GB/s)</i>
DDR3	1,600	11-11-11	64-bit × 1	1	11.9
DDR4	2,400	16-16-16	64-bit × 1	1	17.9
SALP [†]	1,600	11-11-11	64-bit × 1	1	11.9
LPDDR3	1,600	12-15-15	64-bit × 1	1	11.9
LPDDR4	2,400	22-22-22	32-bit × 2*	1	17.9
GDDR5 [12]	6,000	18-18-18	64-bit × 1	1	44.7
HBM	1,000	7-7-7	128-bit × 8*	1	119.2
WIO	266	7-7-7	128-bit × 4*	1	15.9
WIO2	1,066	9-10-10	128-bit × 8*	1	127.2



Across 22 workloads, simple CPU model

Figure 2. Performance comparison of DRAM standards

Another Example Study with Ramulator

- Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu,
"Demystifying Workload–DRAM Interactions: An Experimental Study"
*Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (**SIGMETRICS**), Phoenix, AZ, USA, June 2019.*
[\[Preliminary arXiv Version\]](#)
[\[Abstract\]](#)
[\[Slides \(pptx\) \(pdf\)\]](#)

Demystifying Complex Workload–DRAM Interactions: An Experimental Study

SAUGATA GHOSE, Carnegie Mellon University, USA

TIANSHI LI, Carnegie Mellon University, USA

NASTARAN HAJINAZAR, Simon Fraser University, Canada & ETH Zürich, Switzerland

DAMLA SENOL CALI, Carnegie Mellon University, USA

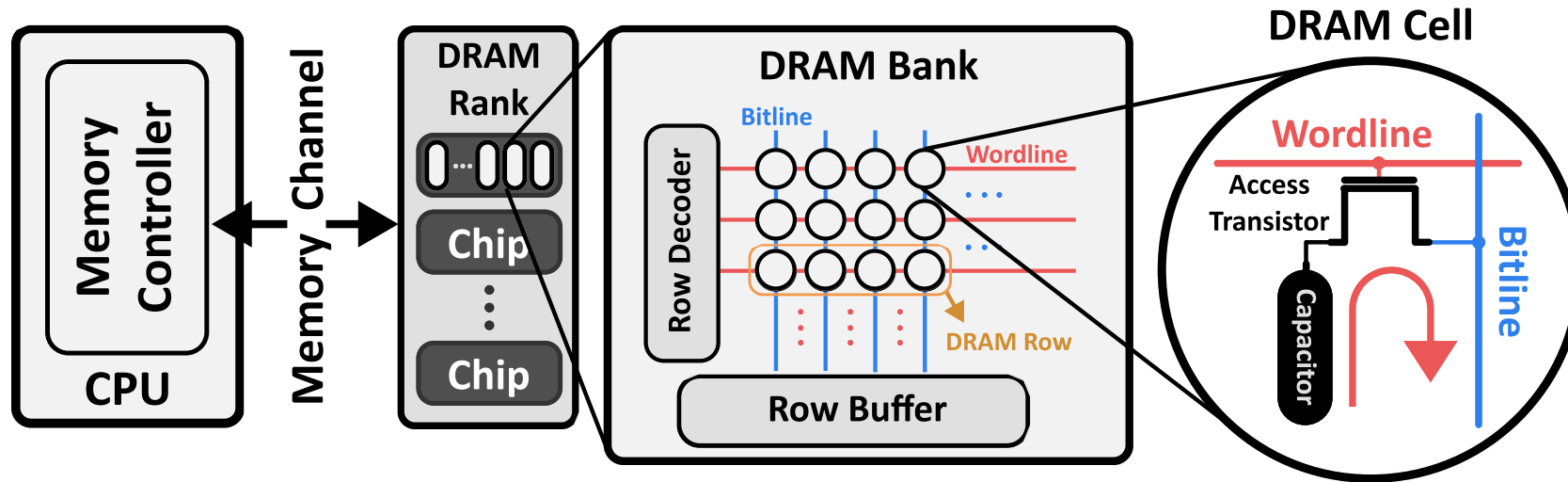
ONUR MUTLU, ETH Zürich, Switzerland & Carnegie Mellon University, USA

Simulator Architecture

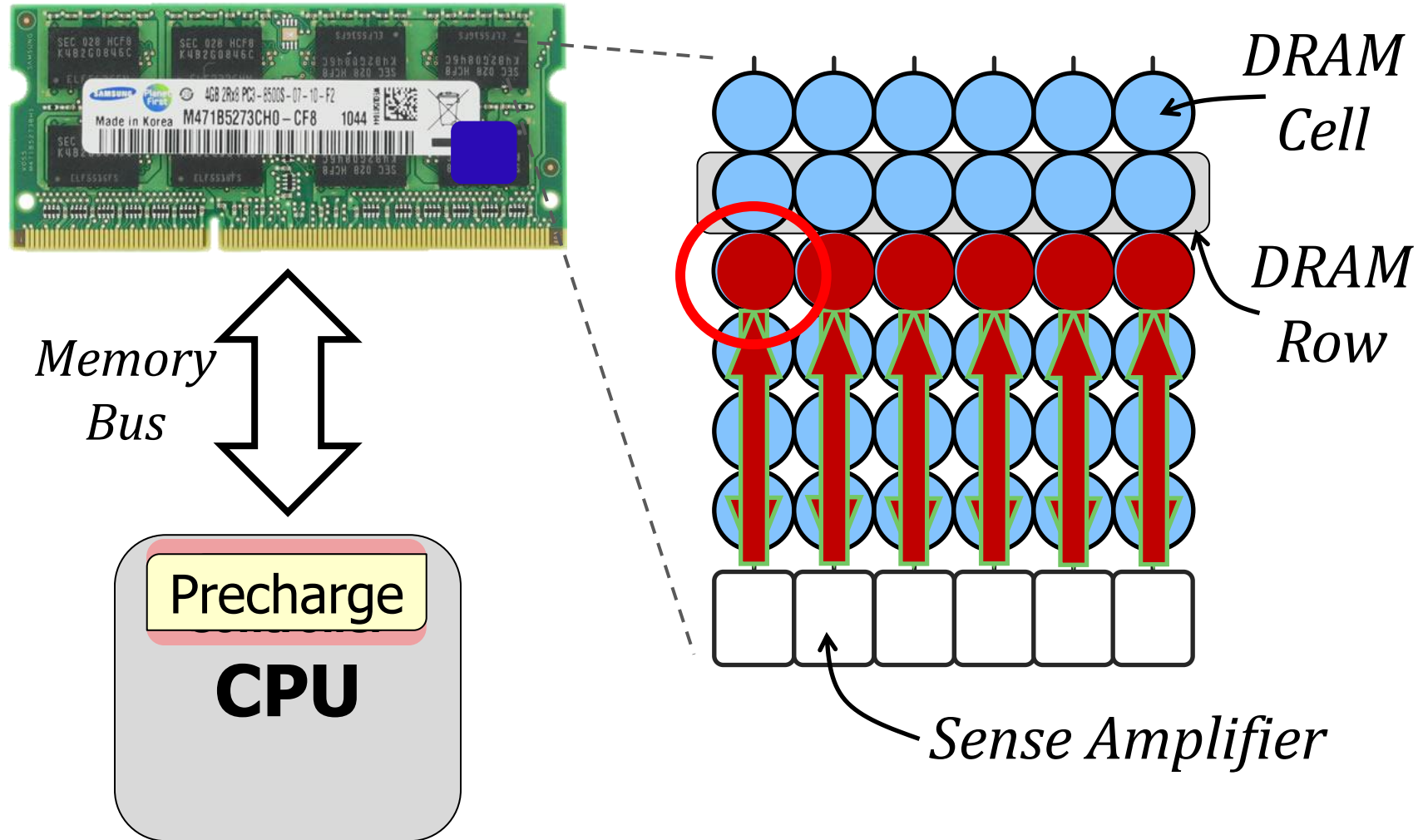
Design Objective: Extensibility

- Treats *extensibility* as a first-class citizen
- **Observation:** DRAM can be abstracted as a hierarchy of *state machines*
- Provides a *standard-agnostic* state machine
 - Paired with any standard at compile time

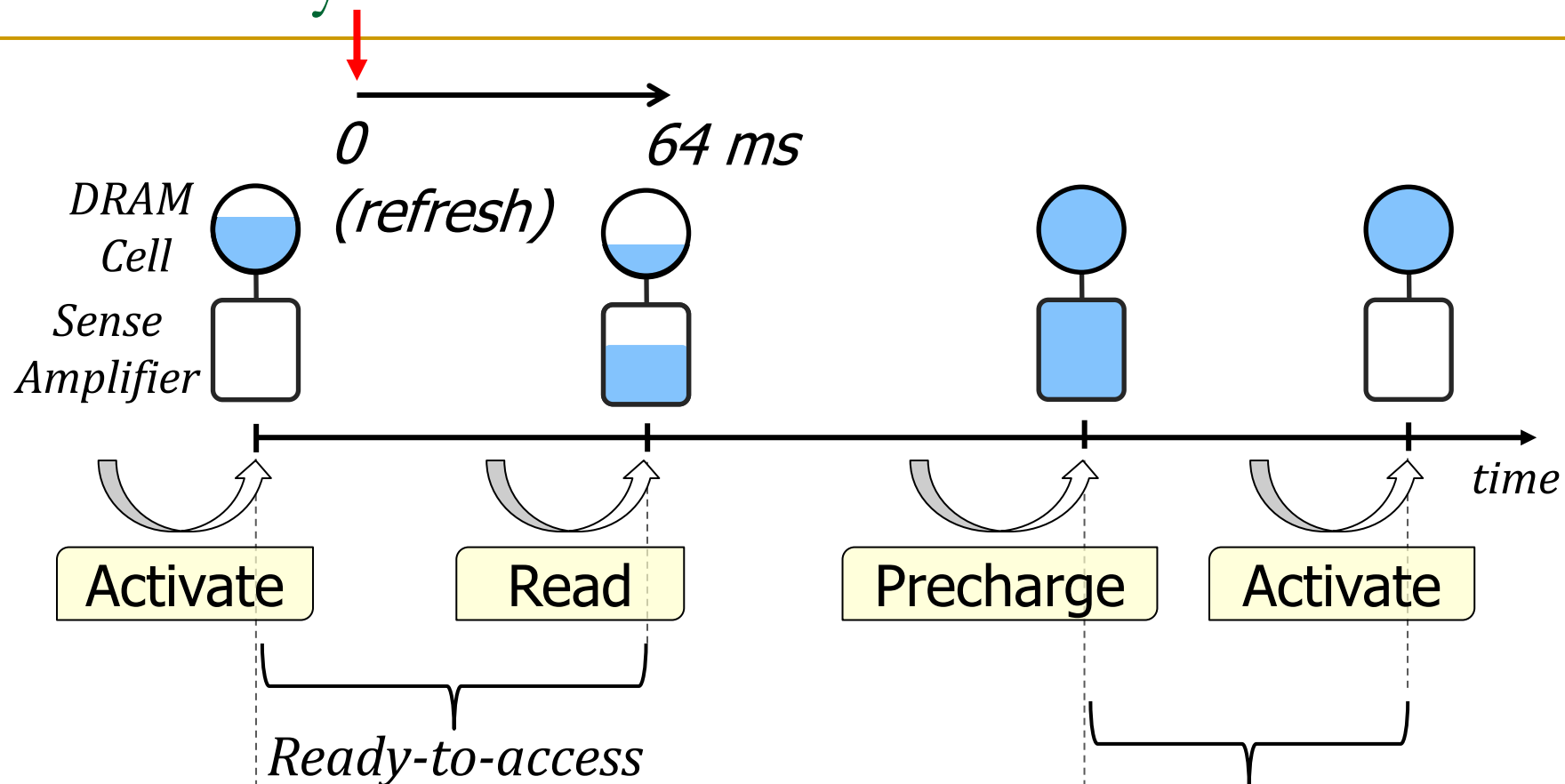
DRAM Organization



DRAM Operations



DRAM Latency



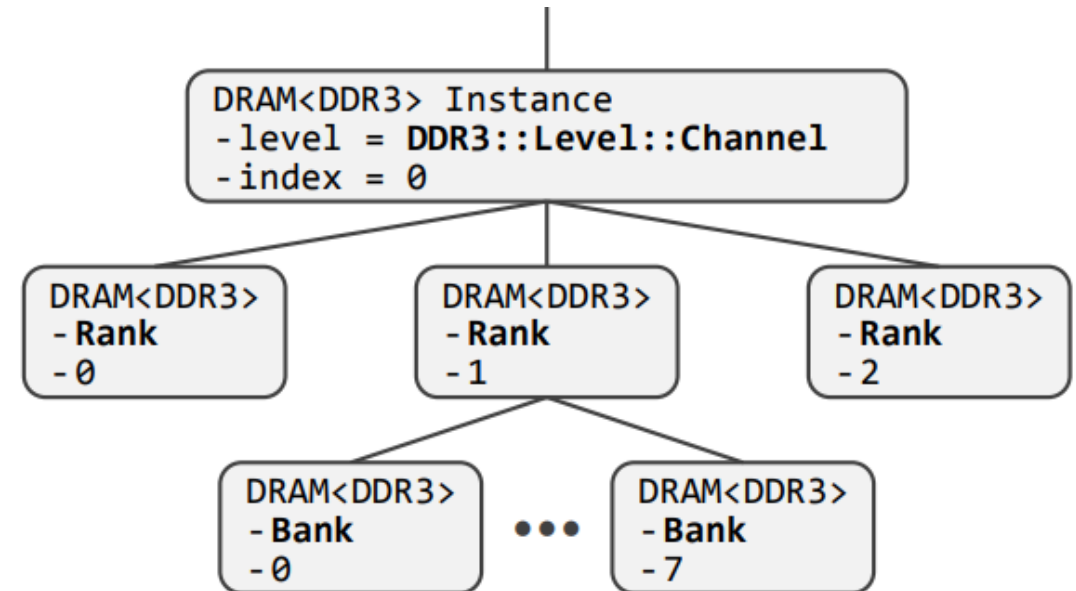
Retention Time: The interval during which the data is retained correctly in the DRAM cell without accessing it

High-Level Design: Hierarchy of State Machines

- **The 'DRAM' class:** a template for building a hierarchy of state machines (i.e., nodes)

```
1 // DRAM.h
2 template <typename T>
3 class DRAM {
4     DRAM<T>* parent;
5     vector<DRAM<T>*>
6         children;
7     T::Level level;
8     int index;
9     // more code...
10 };
```

```
1 // DDR3.h/cpp
2 class DDR3 {
3     enum class Level {
4         Channel, Rank, Bank,
5         Row, Column, MAX
6     };
7
8     // more code...
9
10 };
```



DRAM Node States

- **status**: may change when the node receives one of the DDR3 commands
- **next**: a lookup table specifying the earliest time the node can receive each command (for honoring DDR3 timing parameters).

```
1 // DRAM.h
2 template <typename T>
3 class DRAM {
4     // states (queried/updated by functions below)
5     T::Status status;
6     long horizon[T::Command::MAX];
7     map<int, T::Status> leaf_status; // for bank only
8     // functions (recursively traverses down tree)
9     T::Command decode(T::Command cmd, int addr[]);
10    bool check(T::Command cmd, int addr[], long now);
11    void update(T::Command cmd, int addr[], long now);
12 };
```

Currently named 'next'

```
1 // DDR3.h/cpp
2 class DDR3 {
3     enum class Status {Open, Closed, ..., MAX};
4     enum class Command {ACT, PRE, RD, WR, ..., MAX};
5 };
```


DRAM Functions

The memory controller relies on three recursive functions to serve a memory request:

- **decode():** returns the prerequisite command (e.g., ACT for a closed bank)
- **check():** returns whether or not the DRAM is ready to accept a given command (i.e., timing violation check)
- **update():** updates the node state based on the issued command

```
1 // DRAM.h
2 template <typename T>
3 class DRAM {
4     // states (queried/updated by functions below)
5     T::Status status;
6     long horizon[T::Command::MAX];
7     map<int, T::Status> leaf_status; // for bank only
8     // functions (recursively traverses down tree)
9     T::Command decode(T::Command cmd, int addr[]);
10    bool check(T::Command cmd, int addr[], long now);
11    void update(T::Command cmd, int addr[], long now);
12 };
```

Currently named 'next'



```
1 // DDR3.h/cpp
2 class DDR3 {
3     enum class Status {Open, Closed, ..., MAX};
4     enum class Command {ACT, PRE, RD, WR, ..., MAX};
5 };
```

decode(): Determining the *Prerequisite*

```
1 // DRAM.h
2 template <typename T>
3 class DRAM {
4     T::Command decode(T::Command cmd, int addr[]) {
5         if (prereq[level][cmd]) {
6             // consult lookup-table to decode command
7             T::Command p = prereq[level][cmd](this);
8             if (p != T::Command::MAX)
9                 return p; // decoded successfully
10        }
11        if (children.size() == 0) // lowest-level
12            return cmd; // decoded successfully
13        // use addr[] to identify target child...
14        // invoke decode() at the target child...
15    }
16};
```

```
1 // DDR3.h/cpp
2 class DDR3 {
3     // declare 2D lookup-table of lambdas
4     function<Command(DRAM<DDR3>*)>
5         prereq[Level::MAX][Command::MAX];
6     // populate an entry in the table
7     prereq[Level::Rank][Command::REF] =
8         [] (DRAM<DDR3>* node) -> Command {
9         for (auto bank : node->children)
10             if (bank->status == Status::Open)
11                 return Command::PREA;
12             return Command::REF;
13         };
14     // populate other entries...
15};
```

check(): Satisfying the DRAM *Timing*

```
265 // Check
266 template <typename T>
267 bool DRAM<T>::check(typename T::Command cmd, const int* addr, long clk)
268 {
269     if (next[int(cmd)] != -1 && clk < next[int(cmd)])
270         return false; // stop recursion: the check failed at this level
271
272     int child_id = addr[int(level)+1];
273     if (child_id < 0 || level == spec->scope[int(cmd)] || !children.size())
274         return true; // stop recursion: the check passed at all levels
275
276     // recursively check my child
277     return children[child_id]->check(cmd, addr, clk);
278 }
```

- Verifies whether $next[cmd] \leq now$ for every node affected by cmd

update(): *Transitioning*

Defined in DDR3.cpp

```
324 // Update
325 template <typename T>
326 void DRAM<T>::update(typename T::Command cmd, const int* addr, long clk)
327 {
328     cur_clk = clk;
329     update_state(cmd, addr);
330     update_timing(cmd, addr, clk);
331 }
```

```
334 // Update (State)
335 template <typename T>
336 void DRAM<T>::update_state(typename T::Command cmd, const int* addr)
337 {
338     int child_id = addr[int(level)+1];
339     if (lambda[int(cmd)])
340         lambda[int(cmd)](this, child_id); // update this level
341
342     if (level == spec->scope[int(cmd)] || !children.size())
343         return; // stop recursion: updated all levels
344
345     // recursively update my child
346     children[child_id]->update_state(cmd, addr);
347 }
```

```
350 // Update (Timing)
351 template <typename T>
352 void DRAM<T>::update_timing(typename T::Command cmd, const int* addr, long clk)
353 {
354     // I am not a target node: I am merely one of its siblings
355     if (id != addr[int(level)]) {
356         for (auto& t : timing[int(cmd)])
357             if (!t.sibling)
358                 continue; // not an applicable timing parameter
359
360         assert (t.dist == 1);
361
362         long future = clk + t.val;
363         next[int(t.cmd)] = max(next[int(t.cmd)], future); // update future
364     }
365
366     return; // stop recursion: only target nodes should be recursed
367 }
```

Defined in DDR3.cpp

Ramulator Paper and Source Code

- Yoongu Kim, Weikun Yang, and Onur Mutlu,
"Ramulator: A Fast and Extensible DRAM Simulator"
IEEE Computer Architecture Letters (CAL), March 2015.
[Source Code]
- Source code is released under the liberal MIT License
 - <https://github.com/CMU-SAFARI/ramulator>
 - <https://github.com/CMU-SAFARI/ramulator-pim>
 - *ZSim+Ramulator: a framework for design space exploration of general-purpose Processing-in-Memory (PIM) architectures*

Conclusion

- Provides out-of-the box support for many DRAM standards:
 - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, HMC, and academic proposals (SALP, AL-DRAM, TLDRAM, RowClone, and SARP)
 - Models timing of non-volatile memories (PCM, STT-MRAM)
- Supports multiple scheduling and row buffer management policies
- Modular and extensible to different standards
- Can be paired with other simulators, e.g., gem5 and DRAMPower
- Written in C++11
- ~2.5X faster than fastest open-source simulator

<i>Simulator</i> <i>(clang -O3)</i>	<i>Cycles (10⁶)</i>		<i>Runtime (sec.)</i>		<i>Req/sec (10³)</i>		<i>Memory</i> <i>(MB)</i>
	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	
Ramulator	652	411	752	249	133	402	2.1
DRAMSim2	645	413	2,030	876	49	114	1.2
USIMM	661	409	1,880	750	53	133	4.5
DrSim	647	406	18,109	12,984	6	8	1.6
NVMain	666	413	6,881	5,023	15	20	4,230.0

P&S Ramulator

Designing and Evaluating Memory Systems and
Modern Software Workloads with Ramulator

Hasan Hassan

Prof. Onur Mutlu

ETH Zürich

Spring 2022

9 March 2022