

Computer Architecture: Main Memory (Part II)

Prof. Onur Mutlu
Carnegie Mellon University

(reorged & cut by Seth)

Review: DRAM Controller: Functions

- Ensure correct operation of DRAM (refresh and timing)
- Service DRAM requests while obeying timing constraints of DRAM chips
 - Constraints: resource conflicts (bank, bus, channel), minimum write-to-read delays
 - Translate requests to DRAM command sequences
- Buffer and schedule requests to improve performance
 - Reordering, row-buffer, bank, rank, bus management
- Manage power consumption and thermals in DRAM
 - Turn on/off DRAM chips, manage power modes

2

DRAM Power Management

- DRAM chips have power modes
- Idea: When not accessing a chip power it down
- Power states
 - Active (highest power)
 - All banks idle
 - Power-down
 - Self-refresh (lowest power)
- Tradeoff: State transitions incur latency during which the chip cannot be accessed

3

Review: Why are DRAM Controllers Difficult to Design?

- Need to obey DRAM timing constraints for correctness
 - There are many (50+) timing constraints in DRAM
 - tWTR: Minimum number of cycles to wait before issuing a read command after a write command is issued
 - tRC: Minimum number of cycles between the issuing of two consecutive activate commands to the same bank
 - ...
- Need to keep track of many resources to prevent conflicts
 - Channels, banks, ranks, data bus, address bus, row buffers
- Need to handle DRAM refresh
- Need to optimize for performance (in the presence of constraints)
 - Reordering is not simple
 - Predicting the future?

4

Review: Many DRAM Timing Constraints

Latency	Symbol	DRAM cycles	Latency	Symbol	DRAM cycles
Precharge	t_{RP}	11	Activate to read/write	t_{RCD}	11
Read column address strobe	CL	11	Write column address strobe	CWL	8
Additive	AL	0	Activate to activate	t_{RC}	39
Activate to precharge	t_{RAS}	28	Read to precharge	t_{RTP}	6
Burst length	t_{BL}	4	Column address strobe to column address strobe	t_{CCD}	4
Activate to activate (different bank)	t_{RRD}	6	Four activate windows	t_{FAW}	24
Write to read	t_{WTR}	6	Write recovery	t_{WR}	12

Table 4. DDR3 1600 DRAM timing specifications

- From Lee et al., "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," HPS Technical Report, April 2010.

5

Review: More on DRAM Operation

- Kim et al., "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," ISCA 2012.
- Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

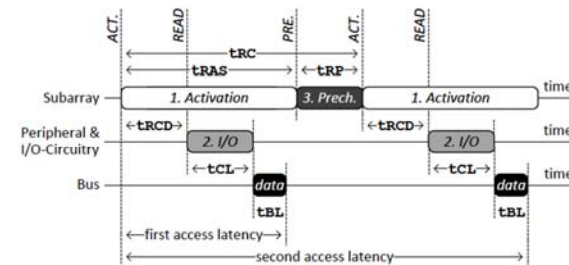


Figure 5. Three Phases of DRAM Access

Table 2. Timing Constraints (DDR3-1066) [43]

Phase	Commands	Name	Value
1	ACT → READ	t_{RCD}	15ns
	ACT → WRITE	t_{RCD}	15ns
	ACT → PRE	t_{RAS}	37.5ns
2	READ → data	t_{CL}	15ns
	WRITE → data	t_{CWL}	11.25ns
	data burst	t_{BL}	7.5ns
3	PRE → ACT	t_{RP}	15ns
1 & 3	ACT → ACT	t_{RC} ($t_{RAS} + t_{RP}$)	52.5ns

6

Self-Optimizing DRAM Controllers

- Problem: DRAM controllers difficult to design → It is difficult for human designers to design a policy that can adapt itself very well to different workloads and different system conditions
- Idea: Design a memory controller that adapts its scheduling policy decisions to workload behavior and system conditions using machine learning.
- Observation: Reinforcement learning maps nicely to memory control.
- Design: Memory controller is a reinforcement learning agent that dynamically and continuously learns and employs the best scheduling policy.

Self-Optimizing DRAM Controllers

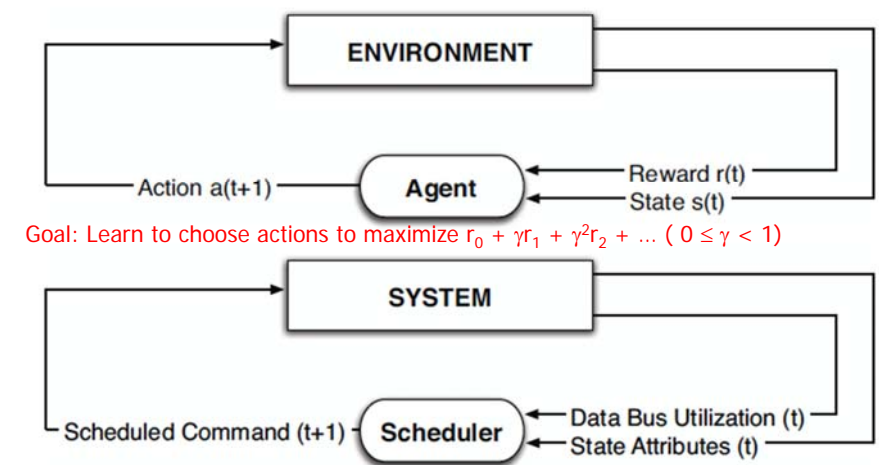
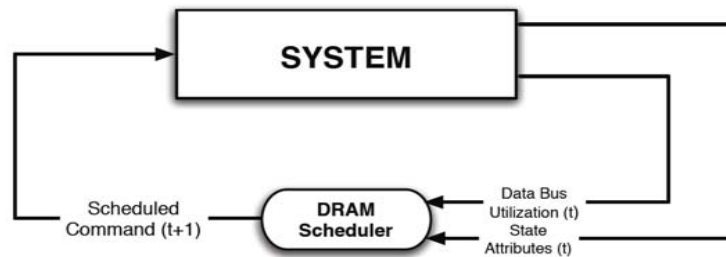


Figure 2: (a) Intelligent agent based on reinforcement learning principles; (b) DRAM scheduler as an RL-agent

Self-Optimizing DRAM Controllers

- Dynamically adapt the memory scheduling policy via interaction with the system at runtime
 - Associate system states and actions (commands) with long term reward values
 - Schedule command with highest estimated long-term value in each state
 - Continuously update state-action values based on feedback from system



9

Self-Optimizing DRAM Controllers

- Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana, **"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"**
Proceedings of the 35th International Symposium on Computer Architecture (ISCA), pages 39-50, Beijing, China, June 2008.

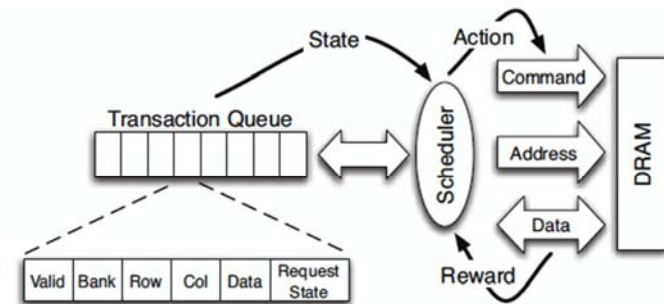


Figure 4: High-level overview of an RL-based scheduler.

10

States, Actions, Rewards

- | | | |
|--|--|---|
| <p>❖ Reward function</p> <ul style="list-style-type: none"> • +1 for scheduling Read and Write commands • 0 at all other times | <p>❖ State attributes</p> <ul style="list-style-type: none"> • Number of reads, writes, and load misses in transaction queue • Number of pending writes and ROB heads waiting for referenced row • Request's relative ROB order | <p>❖ Actions</p> <ul style="list-style-type: none"> • Activate • Write • Read - load miss • Read - store miss • Precharge - pending • Precharge - preemptive • NOP |
|--|--|---|

11

Performance Results

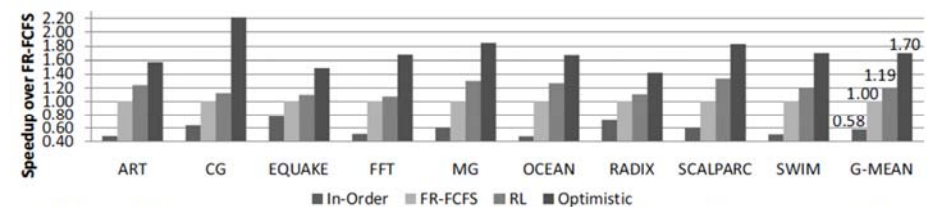


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

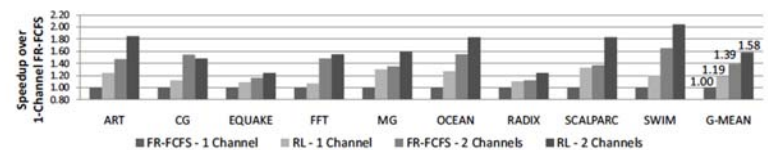


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

12

Self Optimizing DRAM Controllers

- Advantages
 - + Adapts the scheduling policy dynamically to changing workload behavior and to maximize a long-term target
 - + Reduces the designer's burden in finding a good scheduling policy. Designer specifies:
 - 1) What system variables might be useful
 - 2) What target to optimize, but not how to optimize it
- Disadvantages
 - Black box: designer much less likely to implement what she cannot easily reason about
 - How to specify different reward functions that can achieve different objectives? (e.g., fairness, QoS)

13

Trends Affecting Main Memory

Major Trends Affecting Main Memory (I)

- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending

Major Trends Affecting Main Memory (II)

- Need for main memory capacity, bandwidth, QoS increasing
 - **Multi-core**: increasing number of cores
 - **Data-intensive applications**: increasing demand/hunger for data
 - **Consolidation**: cloud computing, GPUs, mobile
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending

Major Trends Affecting Main Memory (III)

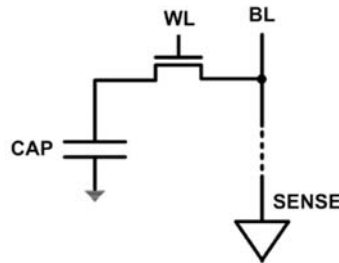
- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
 - ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer 2003]
 - DRAM consumes power even when not used (periodic refresh)
- DRAM technology scaling is ending

Major Trends Affecting Main Memory (IV)

- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending
 - ITRS projects DRAM will not scale easily below X nm
 - Scaling has provided many benefits:
 - higher capacity (density), lower cost, lower energy

The DRAM Scaling Problem

- DRAM stores charge in a capacitor (charge-based memory)
 - Capacitor must be large enough for reliable sensing
 - Access transistor should be large enough for low leakage and high retention time
 - Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]



- DRAM capacity, cost, and energy/power hard to scale

Solution 1: Tolerate DRAM

- Overcome DRAM shortcomings with
 - System-DRAM co-design
 - Novel DRAM architectures, interface, functions
 - Better waste management (efficient utilization)
- Key issues to tackle
 - Reduce refresh energy
 - Improve bandwidth and latency
 - Reduce waste
 - Enable reliability at low cost
- Liu, Jaiyen, Veras, Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim, Seshadri, Lee+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
- Lee+, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
- Liu+, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices" ISCA'13.
- Seshadri+, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," 2013.

Tolerating DRAM: System-DRAM Co-Design

New DRAM Architectures

- RAIDR: Reducing Refresh Impact
- TL-DRAM: Reducing DRAM Latency
- SALP: Reducing Bank Conflict Impact
- RowClone: Fast Bulk Data Copy and Initialization

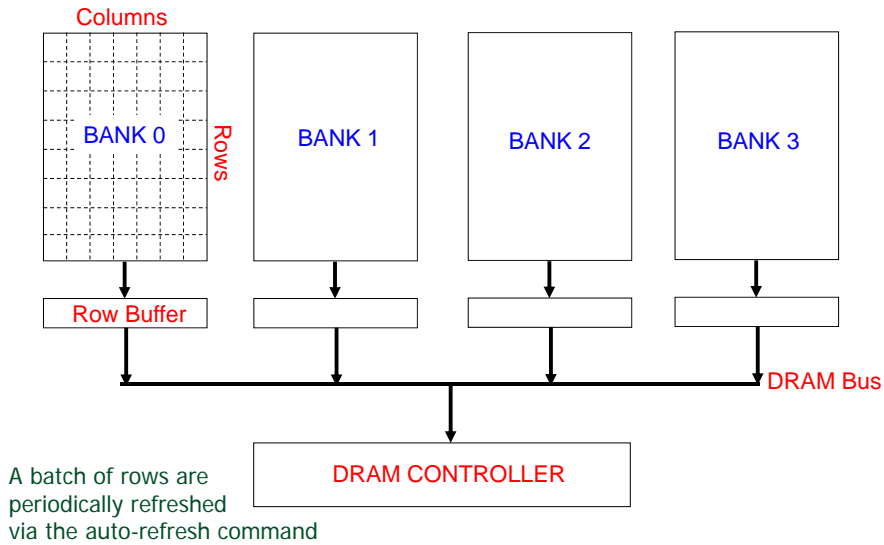
22

RAIDR: Reducing DRAM Refresh Impact

DRAM Refresh

- DRAM capacitor charge leaks over time
- The memory controller needs to refresh each row periodically to restore charge
 - Activate + precharge each row every N ms
 - Typical N = 64 ms
- Downsides of refresh
 - Energy consumption: Each refresh consumes energy
 - Performance degradation: DRAM rank/bank unavailable while refreshed
 - QoS/predictability impact: (Long) pause times during refresh
 - Refresh rate limits DRAM density scaling

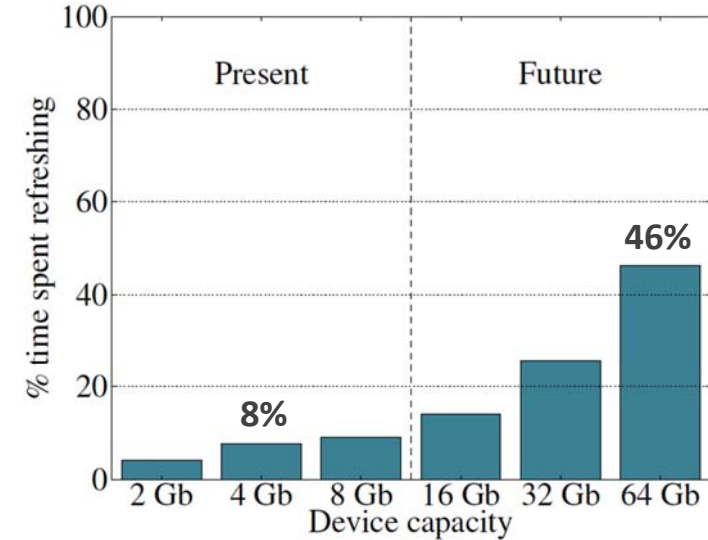
Refresh Today: Auto Refresh



SAFARI

25

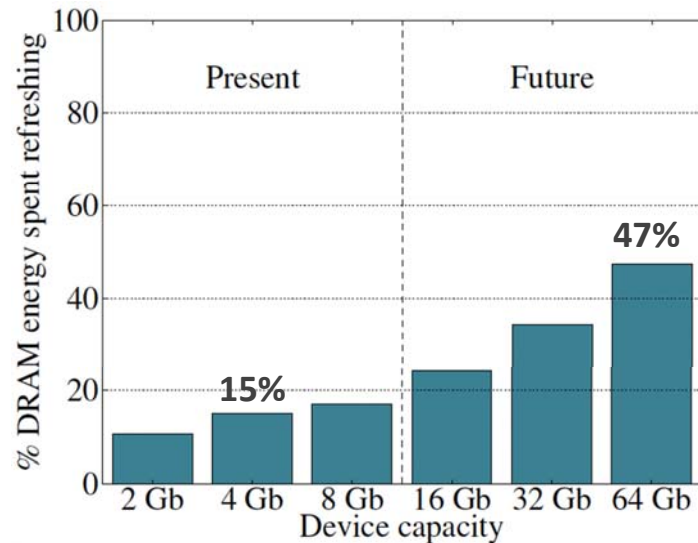
Refresh Overhead: Performance



SAFARI

26

Refresh Overhead: Energy

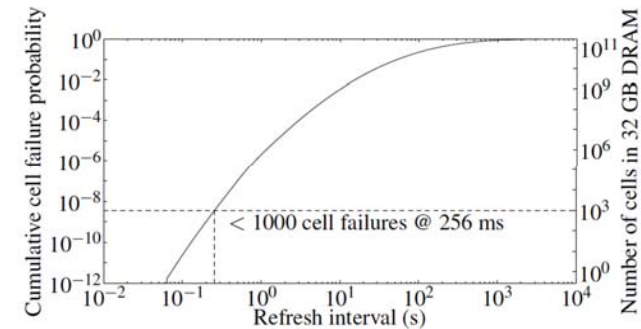


SAFARI

27

Problem with Conventional Refresh

- Today: Every row is refreshed at the same rate



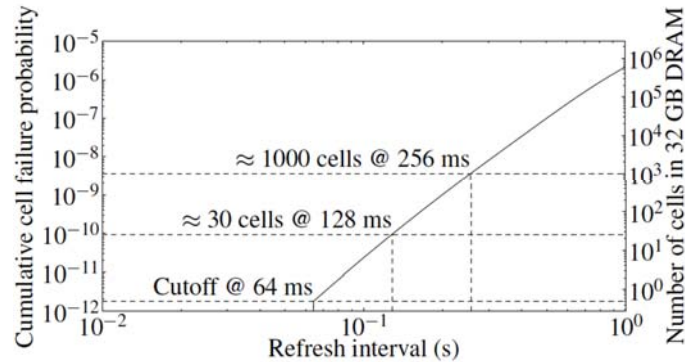
- Observation: Most rows can be refreshed much less often without losing data [Kim+, EDL'09]
- Problem: No support in DRAM for different refresh rates per row

SAFARI

28

Retention Time of DRAM Rows

- Observation: Only very few rows need to be refreshed at the worst-case rate

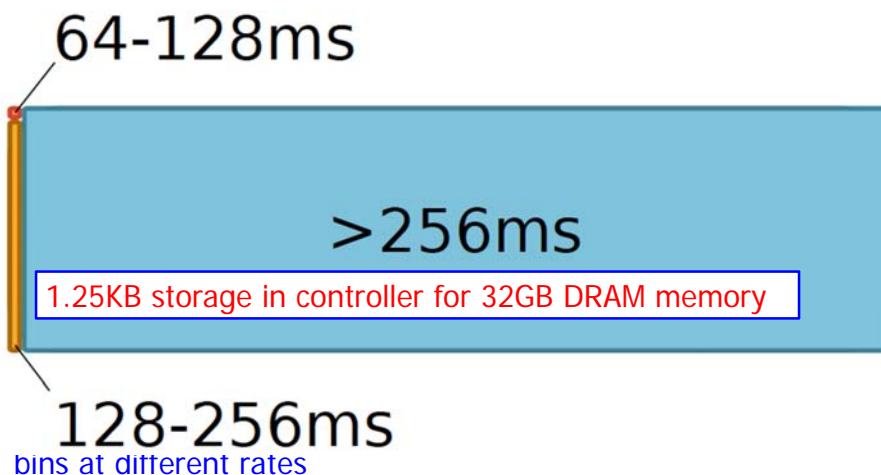


- Can we exploit this to reduce refresh operations at low cost?

Reducing DRAM Refresh Operations

- **Idea:** Identify the retention time of different rows and refresh each row at the frequency it needs to be refreshed
- **(Cost-conscious) Idea:** Bin the rows according to their minimum retention times and refresh rows in each bin at the refresh rate specified for the bin
 - e.g., a bin for 64-128ms, another for 128-256ms, ...
- **Observation:** Only very few rows need to be refreshed very frequently [64-128ms] → Have only a few bins → Low HW overhead to achieve large reductions in refresh operations
- Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

RAIDR: Mechanism



bins at different rates

→ probe Bloom Filters to determine refresh rate of a row

1. Profiling

To profile a row:

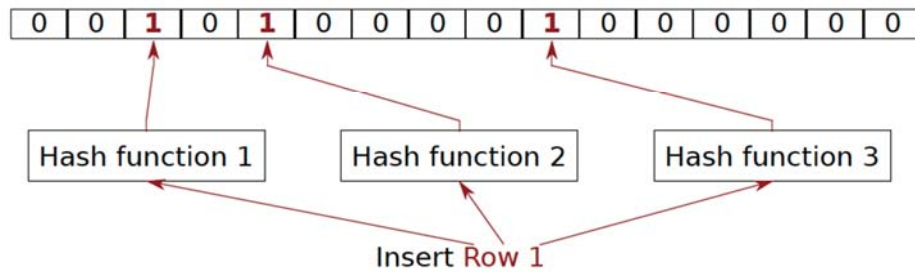
1. Write data to the row
2. Prevent it from being refreshed
3. Measure time before data corruption

	Row 1	Row 2	Row 3
Initially	11111111...	11111111...	11111111...
After 64 ms	11111111...	11111111...	11111111...
After 128 ms	11011111... (64-128ms)	11111111...	11111111...
After 256 ms		11111011... (128-256ms)	11111111... (>256ms)

2. Binning

- How to efficiently and scalably store rows into retention time bins?
- Use Hardware Bloom Filters [Bloom, CACM 1970]

Example with 64-128ms bin:

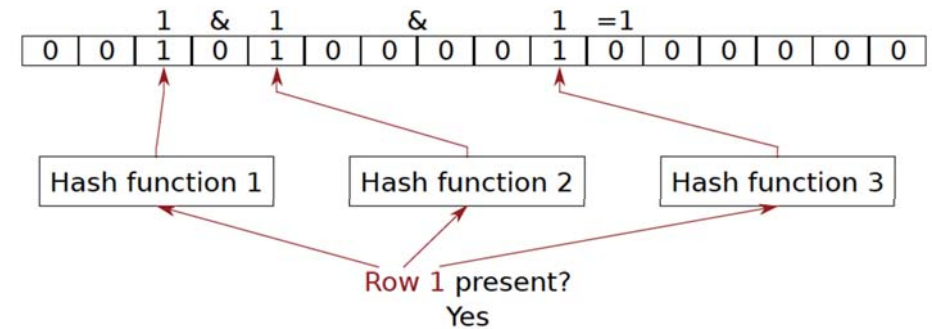


SAFARI

33

Bloom Filter Operation Example

Example with 64-128ms bin:

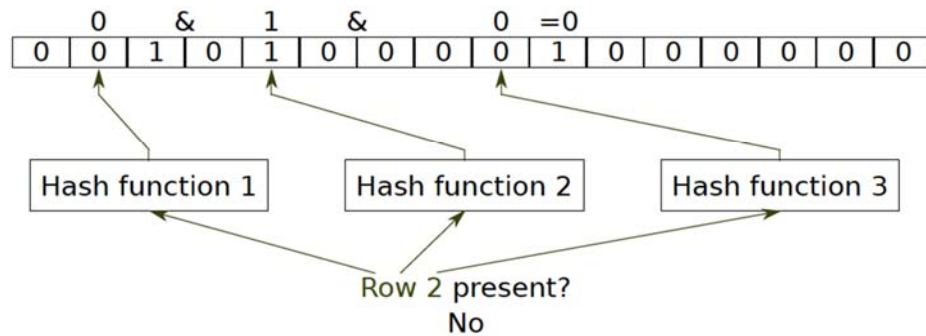


SAFARI

34

Bloom Filter Operation Example

Example with 64-128ms bin:

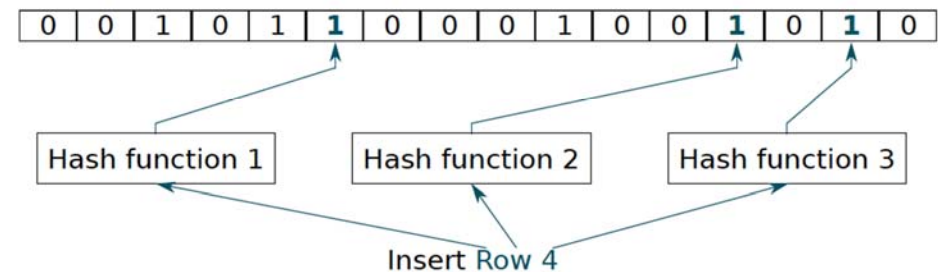


SAFARI

35

Bloom Filter Operation Example

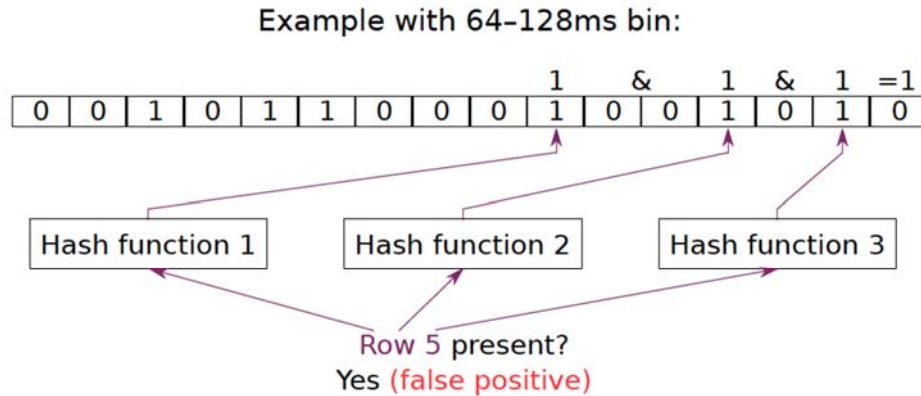
Example with 64-128ms bin:



SAFARI

36

Bloom Filter Operation Example



SAFARI

37

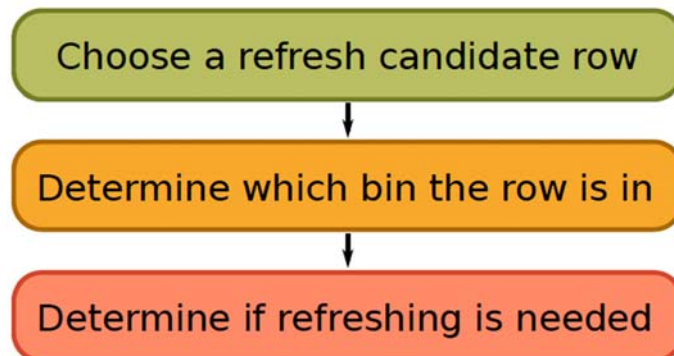
Benefits of Bloom Filters as Bins

- **False positives:** a row may be declared present in the Bloom filter even if it was never inserted
 - **Not a problem:** Refresh some rows more frequently than needed
- **No false negatives:** rows are never refreshed less frequently than needed (no correctness problems)
- **Scalable:** a Bloom filter never overflows (unlike a fixed-size table)
- **Efficient:** No need to store info on a per-row basis; simple hardware → 1.25 KB for 2 filters for 32 GB DRAM system

SAFARI

38

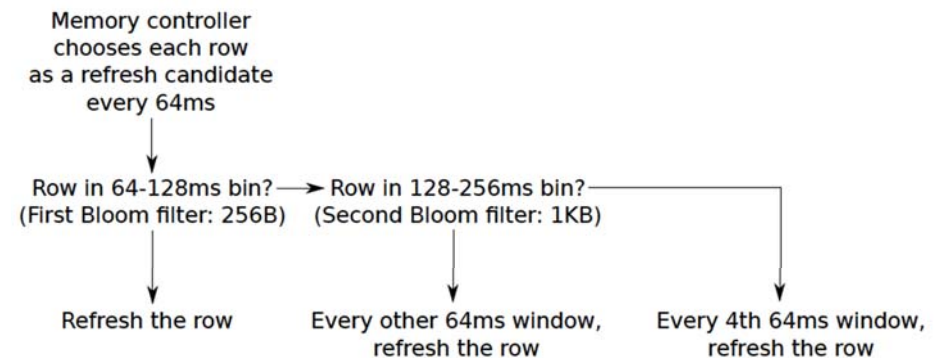
3. Refreshing (RAIDR Refresh Controller)



SAFARI

39

3. Refreshing (RAIDR Refresh Controller)



Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

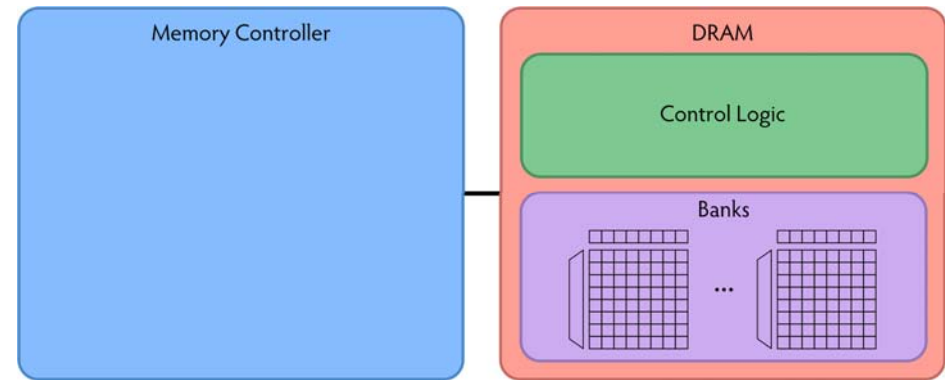
SAFARI

40

Tolerating Temperature Changes

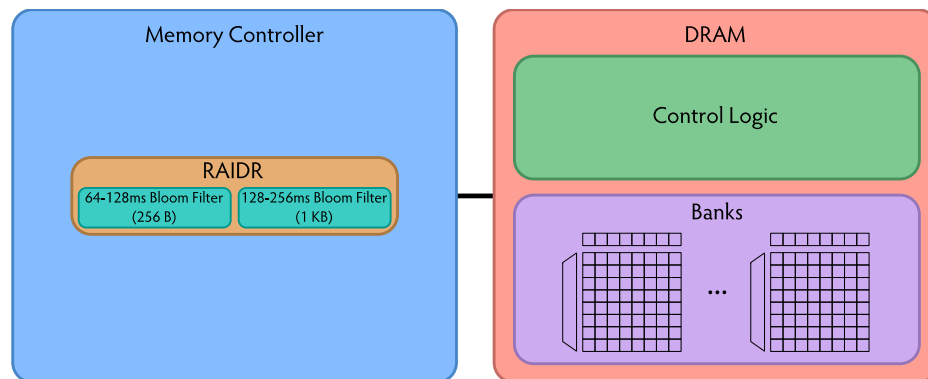
- Change in temperature causes retention time of all cells to change by a uniform and predictable factor
- **Refresh rate scaling**: increase the refresh rate for all rows uniformly, depending on the temperature
- Implementation: counter with programmable period
 - Lower temperature \Rightarrow longer period \Rightarrow less frequent refreshes
 - Higher temperature \Rightarrow shorter period \Rightarrow more frequent refreshes

RAIDR: Baseline Design



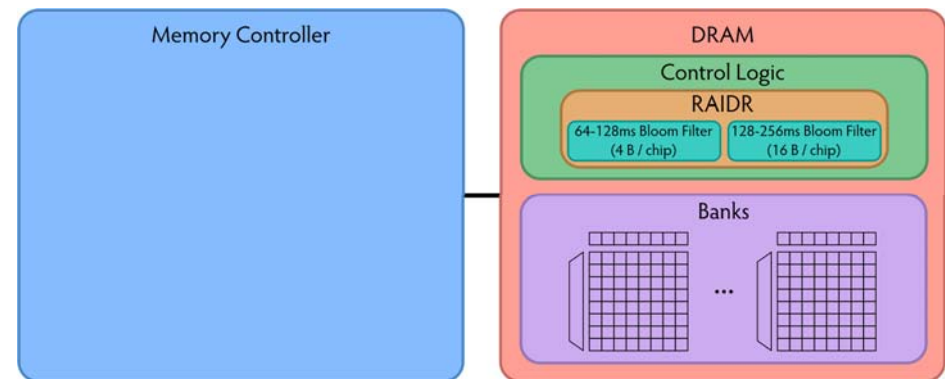
Refresh control is in DRAM in today's auto-refresh systems
RAIDR can be implemented in either the controller or DRAM

RAIDR in Memory Controller: Option 1



Overhead of RAIDR in DRAM controller:
1.25 KB Bloom Filters, 3 counters, additional commands issued for per-row refresh (all accounted for in evaluations)

RAIDR in DRAM Chip: Option 2

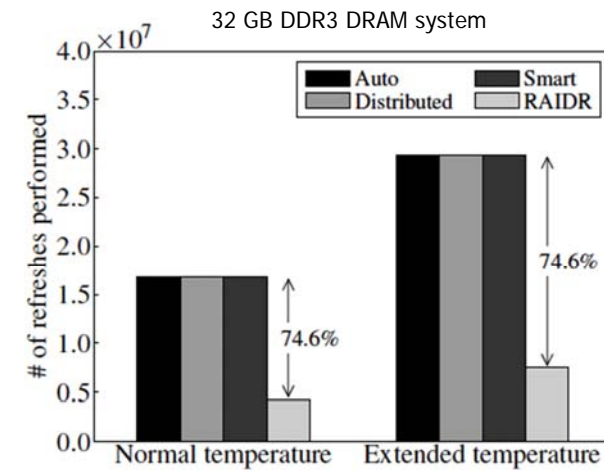


Overhead of RAIDR in DRAM chip:
Per-chip overhead: 20B Bloom Filters, 1 counter (4 Gbit chip)
Total overhead: 1.25KB Bloom Filters, 64 counters (32 GB DRAM)

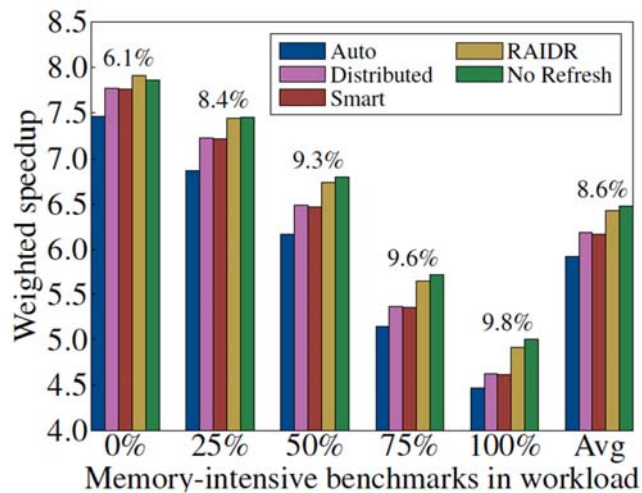
RAIDR Results

- Baseline:
 - 32 GB DDR3 DRAM system (8 cores, 512KB cache/core)
 - 64ms refresh interval for all rows
- RAIDR:
 - 64–128ms retention range: 256 B Bloom filter, 10 hash functions
 - 128–256ms retention range: 1 KB Bloom filter, 6 hash functions
 - Default refresh interval: 256 ms
- Results on SPEC CPU2006, TPC-C, TPC-H benchmarks
 - 74.6% refresh reduction
 - ~16%/20% DRAM dynamic/idle power reduction
 - ~9% performance improvement

RAIDR Refresh Reduction

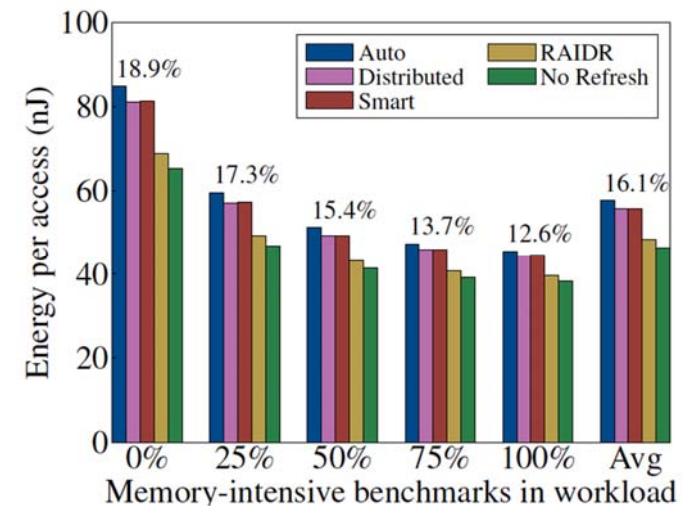


RAIDR: Performance



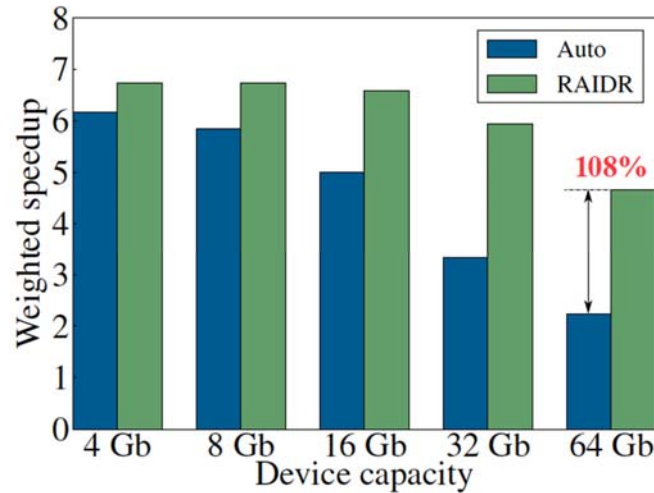
RAIDR performance benefits increase with workload's memory intensity

RAIDR: DRAM Energy Efficiency



RAIDR energy benefits increase with memory idleness

DRAM Device Capacity Scaling: Performance

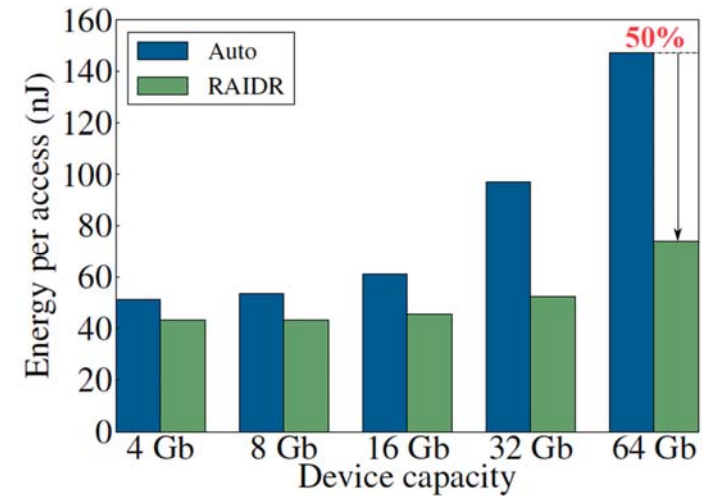


RAIDR performance benefits increase with DRAM chip capacity

SAFARI

49

DRAM Device Capacity Scaling: Energy



RAIDR energy benefits increase with DRAM chip capacity

[RAIDR slides](#)

SAFARI

50

More Readings Related to RAIDR

- Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and [Onur Mutlu](#), **"An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms"** *Proceedings of the 40th International Symposium on Computer Architecture (ISCA)*, Tel-Aviv, Israel, June 2013. [Slides \(pptx\)](#) [Slides \(pdf\)](#)

SAFARI

51

New DRAM Architectures

- RAIDR: Reducing Refresh Impact
- [TL-DRAM: Reducing DRAM Latency](#)
- SALP: Reducing Bank Conflict Impact
- RowClone: Fast Bulk Data Copy and Initialization

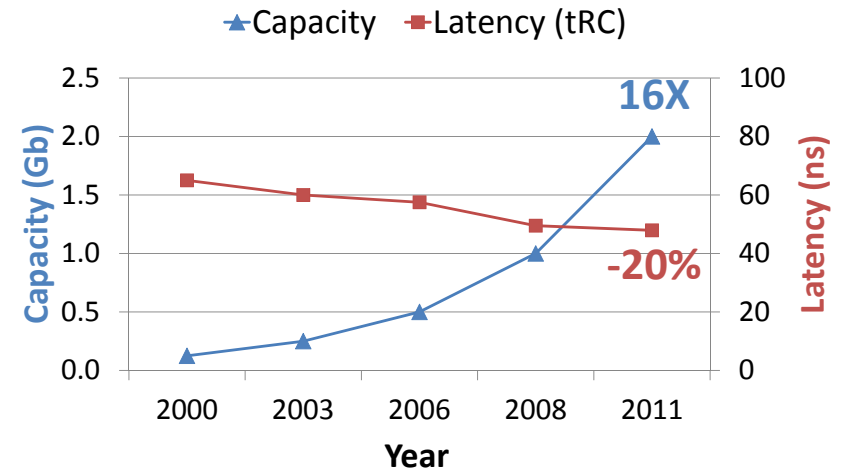
SAFARI

52

Tiered-Latency DRAM: Reducing DRAM Latency

Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu,
"Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture"
19th International Symposium on High-Performance Computer Architecture (HPCA),
 Shenzhen, China, February 2013. [Slides \(pptx\)](#)

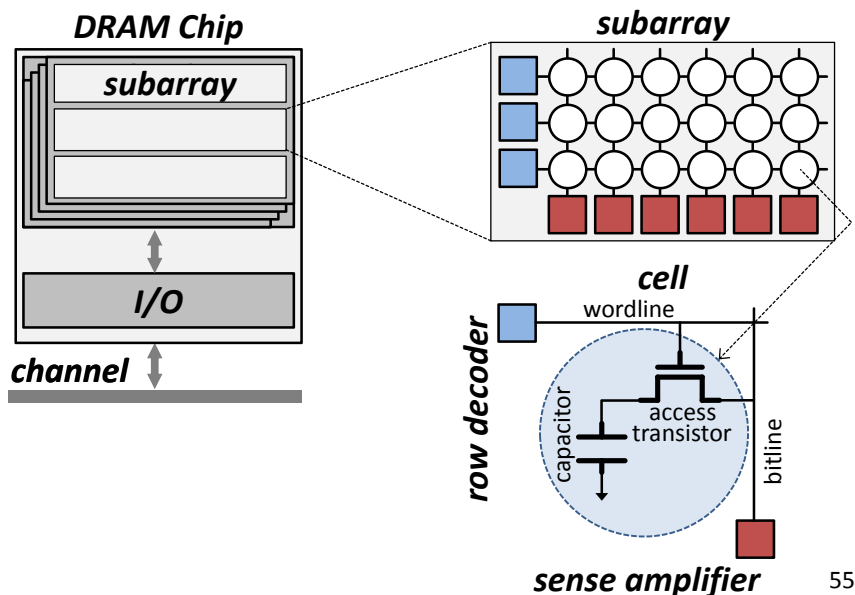
Historical DRAM Latency-Capacity Trend



DRAM latency continues to be a critical bottleneck

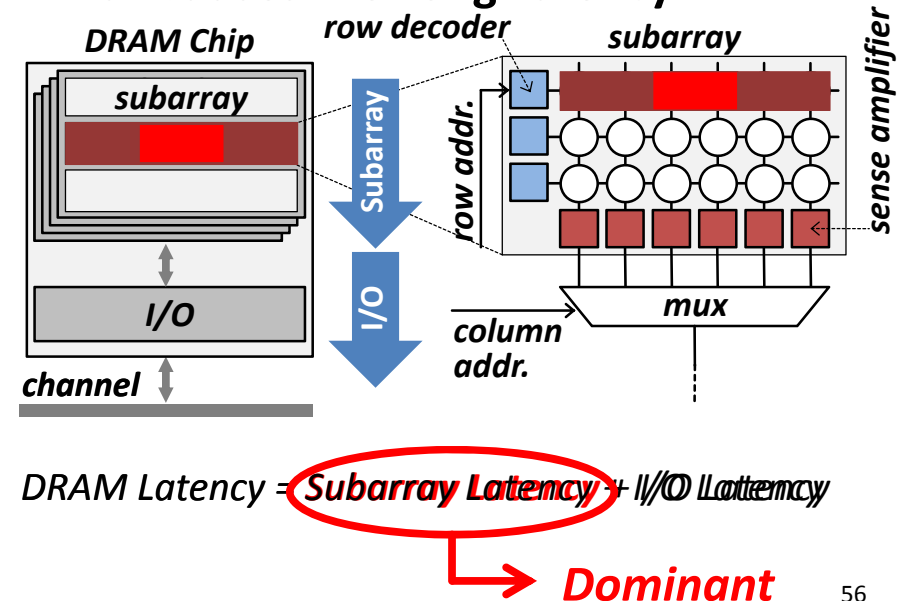
54

What Causes the Long Latency?



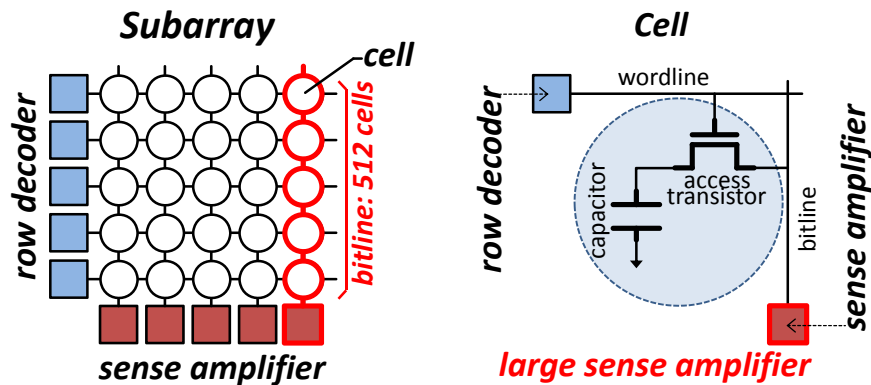
55

What Causes the Long Latency?



56

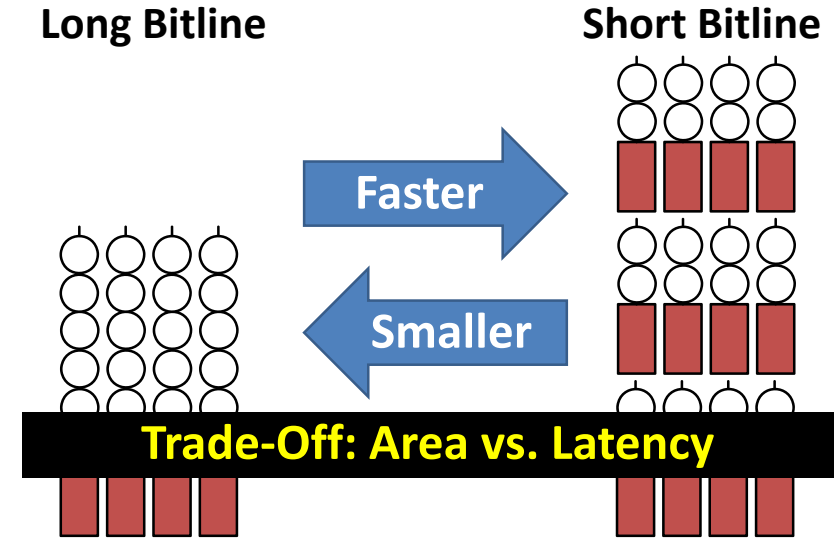
Why is the Subarray So Slow?



- Long bitline
 - Amortizes sense amplifier cost → Small area
 - Large bitline capacitance → High latency & power

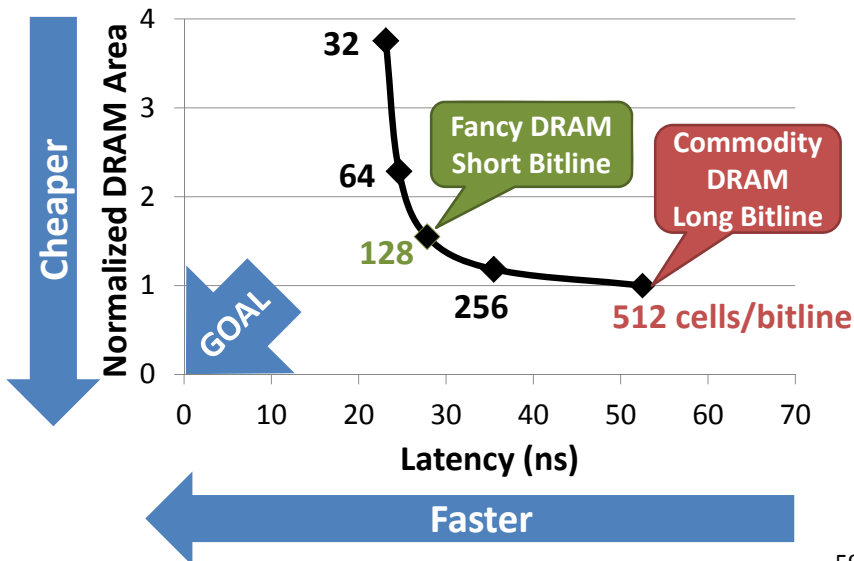
57

Trade-Off: Area (Die Size) vs. Latency



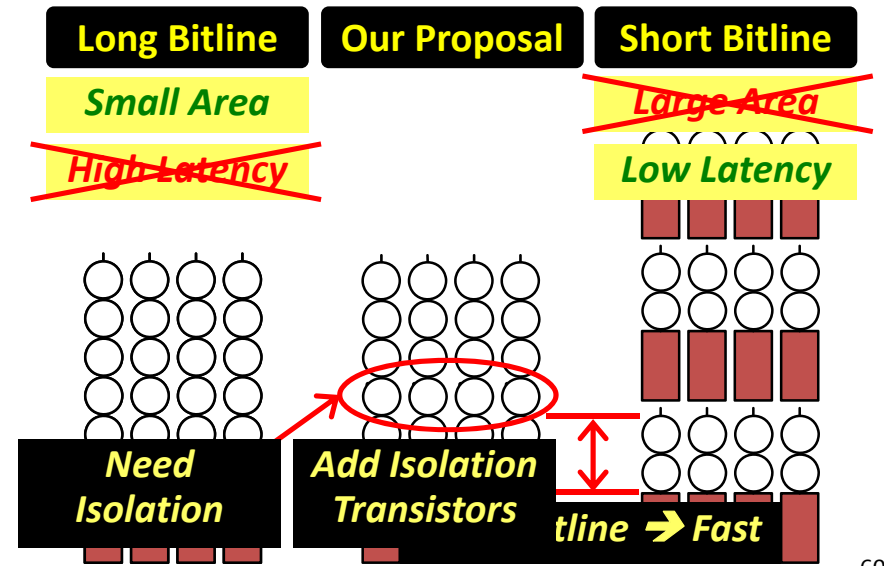
58

Trade-Off: Area (Die Size) vs. Latency



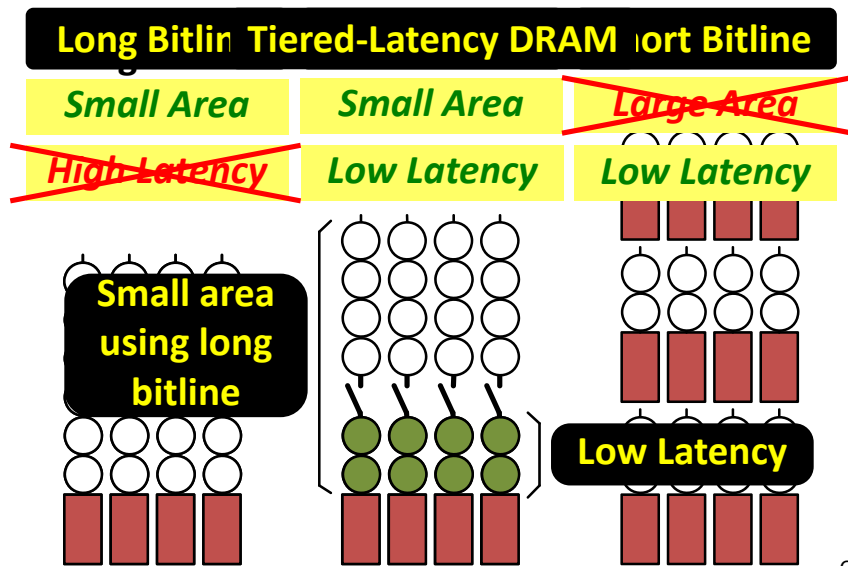
59

Approximating the Best of Both Worlds



60

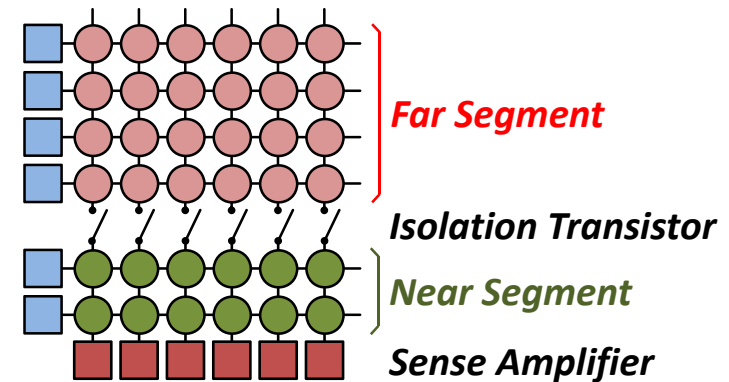
Approximating the Best of Both Worlds



61

Tiered-Latency DRAM

- Divide a bitline into two segments with an isolation transistor

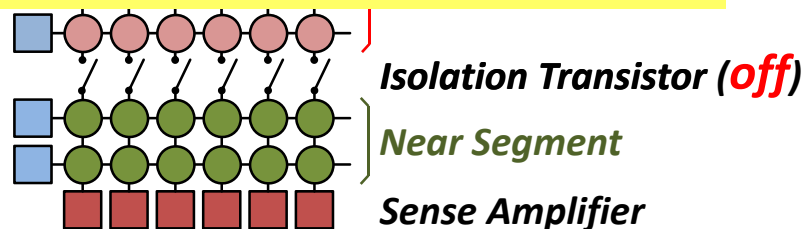


62

Near Segment Access

- Turn **off** the isolation transistor

Reduced bitline length
Reduced bitline capacitance
→ Low latency & low power

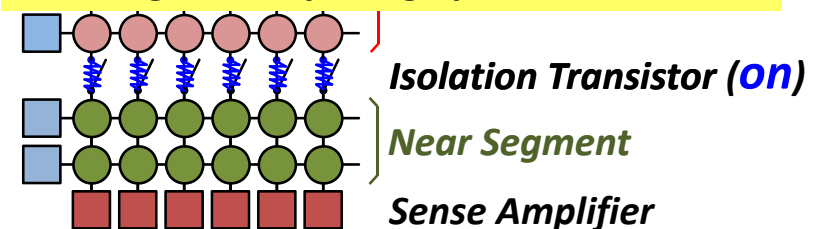


63

Far Segment Access

- Turn **on** the isolation transistor

Long bitline length
Large bitline capacitance
Additional resistance of isolation transistor
→ High latency & high power



64

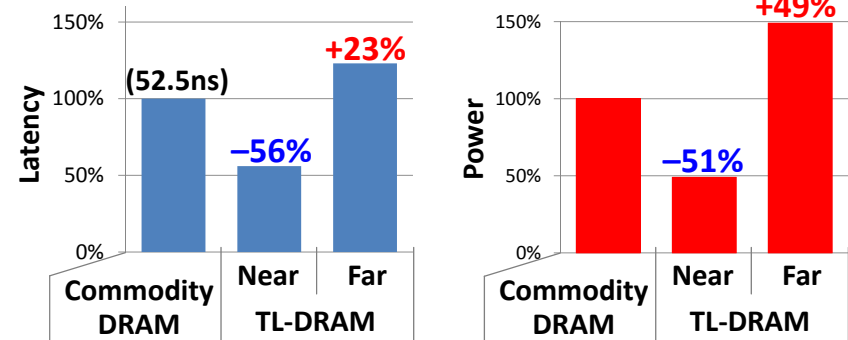
Latency, Power, and Area Evaluation

- **Commodity DRAM:** 512 cells/bitline
- **TL-DRAM:** 512 cells/bitline
 - Near segment: 32 cells
 - Far segment: 480 cells
- **Latency Evaluation**
 - SPICE simulation using circuit-level DRAM model
- **Power and Area Evaluation**
 - DRAM area/power simulator from Rambus
 - DDR3 energy calculator from Micron

65

Commodity DRAM vs. TL-DRAM

- **DRAM Latency (tRC)** • **DRAM Power**

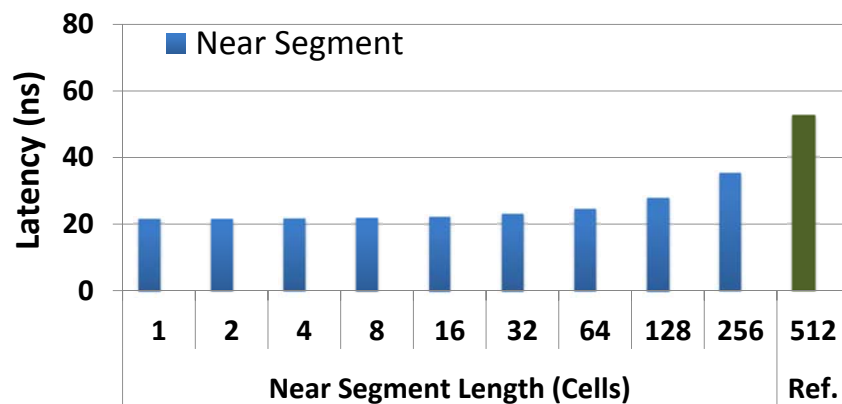


- **DRAM Area Overhead**

~3%: mainly due to the isolation transistors

66

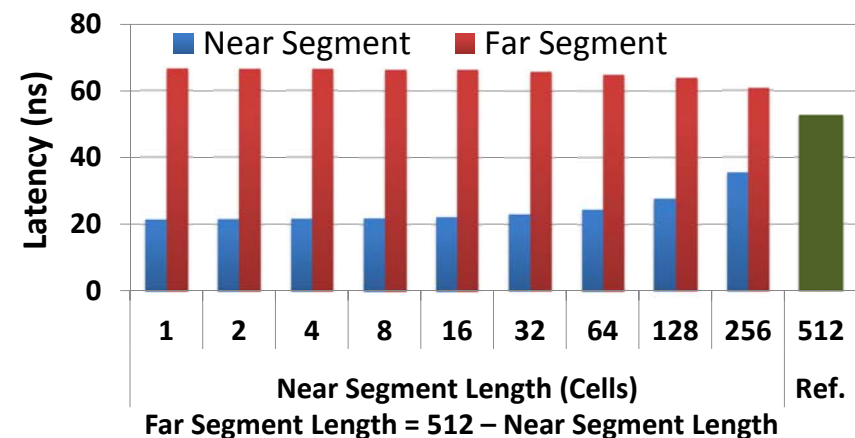
Latency vs. Near Segment Length



Longer near segment length leads to higher near segment latency

67

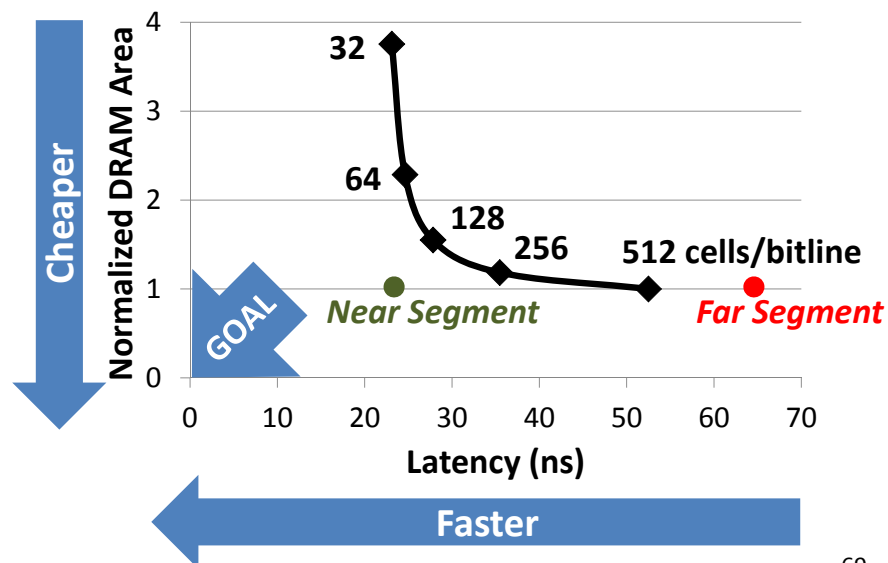
Latency vs. Near Segment Length



Far segment latency is higher than commodity DRAM latency

68

Trade-Off: Area (Die-Area) vs. Latency



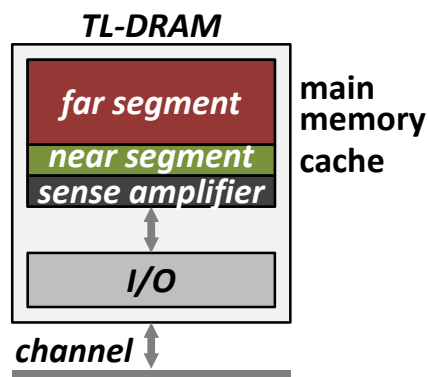
69

Leveraging Tiered-Latency DRAM

- TL-DRAM is a **substrate** that can be leveraged by the hardware and/or software
- Many potential uses
 1. Use near segment as hardware-managed **inclusive** cache to far segment
 2. Use near segment as hardware-managed **exclusive** cache to far segment
 3. Profile-based page mapping by operating system
 4. Simply replace DRAM with TL-DRAM

70

Near Segment as Hardware-Managed Cache

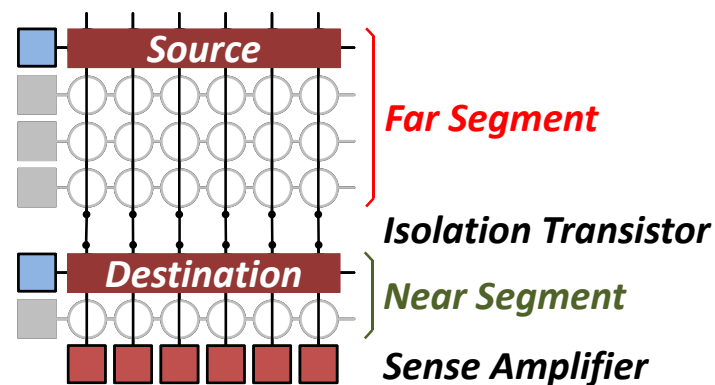


- Challenge 1:** How to efficiently migrate a row between segments?
- Challenge 2:** How to efficiently manage the cache?

71

Inter-Segment Migration

- Goal:** Migrate source row into destination row
- Naïve way:** Memory controller reads the source row *byte by byte* and writes to destination row *byte by byte* → **High latency**

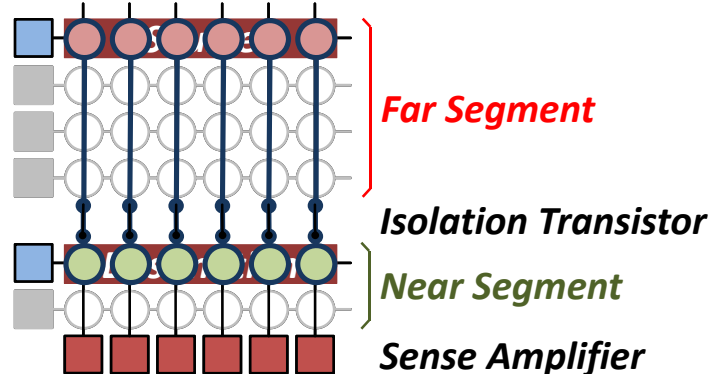


72

Inter-Segment Migration

- Our way:

- Source and destination cells *share bitlines*
- Transfer data from source to destination across *shared bitlines* concurrently

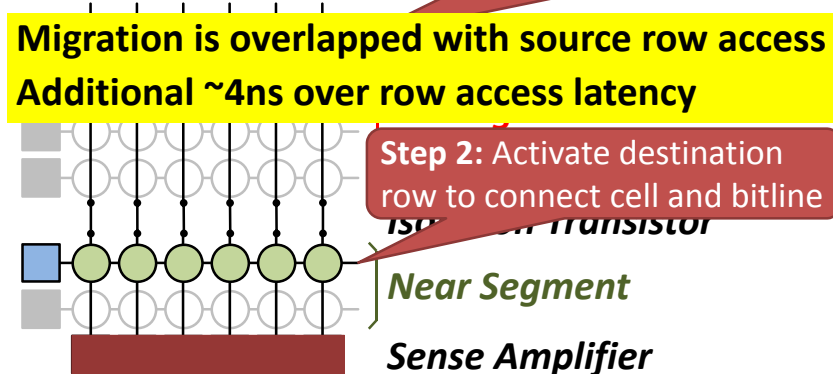


73

Inter-Segment Migration

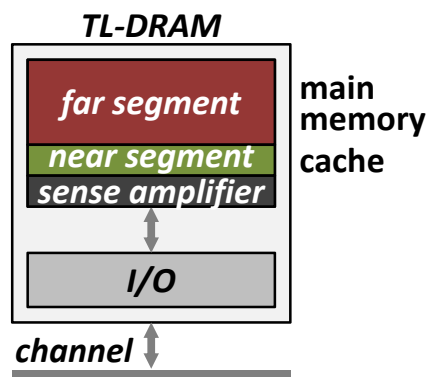
- Our way:

- Source and destination cells *share bitlines*
- Transfer data from source to destination across *shared bitlines* concurrently



74

Near Segment as Hardware-Managed Cache



- **Challenge 1:** How to efficiently migrate a row between segments?
- **Challenge 2:** How to efficiently manage the cache?

75

Evaluation Methodology

- System simulator

- CPU: Instruction-trace-based x86 simulator
- Memory: Cycle-accurate DDR3 DRAM simulator

- Workloads

- 32 Benchmarks from TPC, STREAM, SPEC CPU2006

- Performance Metrics

- Single-core: Instructions-Per-Cycle
- Multi-core: Weighted speedup

76

Configurations

• System configuration

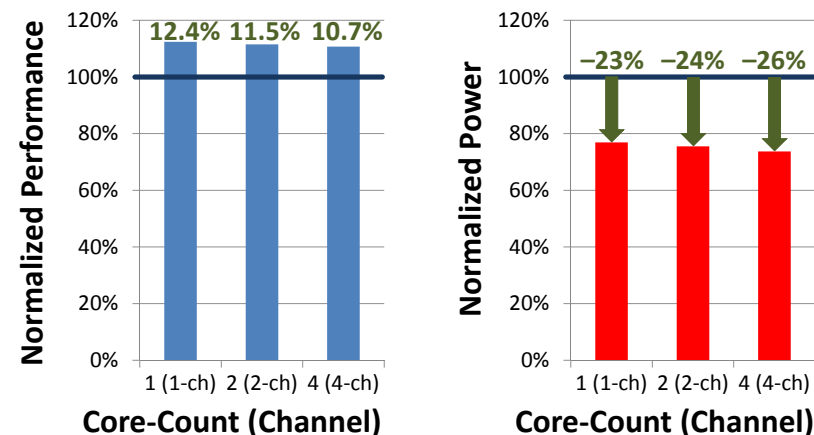
- CPU: 5.3GHz
- LLC: 512kB private per core
- **Memory: DDR3-1066**
 - 1-2 channel, 1 rank/channel
 - 8 banks, 32 subarrays/bank, **512 cells/bitline**
 - Row-interleaved mapping & closed-row policy

• TL-DRAM configuration

- Total bitline length: **512 cells/bitline**
- Near segment length: 1-256 cells
- Hardware-managed inclusive cache: near segment

77

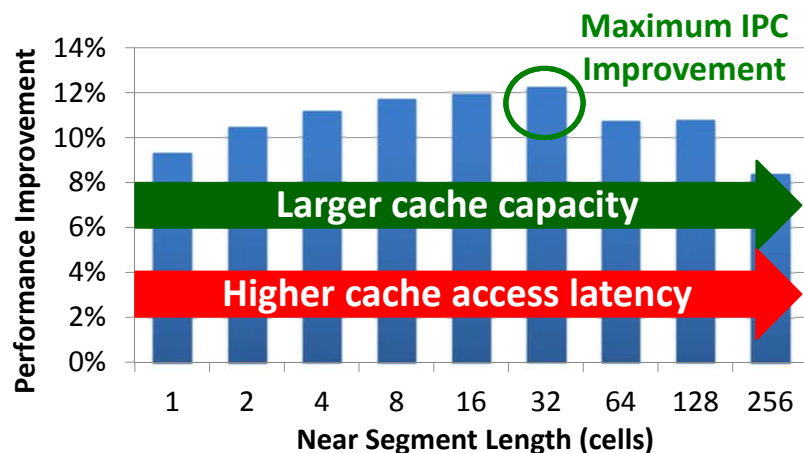
Performance & Power Consumption



Using near segment as a cache improves performance and reduces power consumption

78

Single-Core: Varying Near Segment Length



By adjusting the near segment length, we can trade off cache capacity for cache latency

79

Other Mechanisms & Results

- **More mechanisms** for leveraging TL-DRAM
 - Hardware-managed **exclusive** caching mechanism
 - Profile-based page mapping to near segment
 - TL-DRAM improves performance and reduces power consumption with other mechanisms
- **More than two tiers**
 - Latency evaluation for three-tier TL-DRAM
- **Detailed circuit evaluation** for DRAM latency and power consumption
 - Examination of tRC and tRCD
- **Implementation details and storage cost analysis** in memory controller

in

80

Summary of TL-DRAM

- **Problem:** DRAM latency is a critical performance bottleneck
- **Our Goal:** Reduce DRAM latency with low area cost
- **Observation:** Long bitlines in DRAM are the dominant source of DRAM latency
- **Key Idea:** Divide long bitlines into two shorter segments
 - Fast and slow segments
- **Tiered-latency DRAM:** Enables latency heterogeneity in DRAM
 - Can leverage this in many ways to improve performance and reduce power consumption
- **Results:** When the fast segment is used as a cache to the slow segment → Significant performance improvement (>12%) and power reduction (>23%) at low area cost (3%)

81

New DRAM Architectures

- RAIDR: Reducing Refresh Impact
- TL-DRAM: Reducing DRAM Latency
- SALP: Reducing Bank Conflict Impact
- RowClone: Fast Bulk Data Copy and Initialization

SAFARI

82

Subarray-Level Parallelism: Reducing Bank Conflict Impact

Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu,
"A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM"
Proceedings of the 39th International Symposium on Computer Architecture (ISCA),
Portland, OR, June 2012. [Slides \(pptx\)](#)

The Memory Bank Conflict Problem

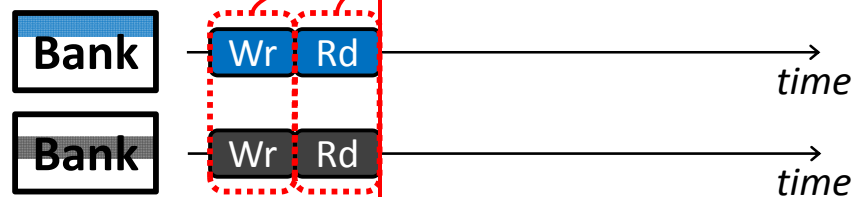
- Two requests to the same bank are serviced serially
- Problem: Costly in terms of performance and power
- Goal: We would like to reduce bank conflicts without increasing the number of banks (at low cost)
- Idea: Exploit the internal sub-array structure of a DRAM bank to parallelize bank conflicts
 - By reducing global sharing of hardware between sub-arrays
- Kim, Seshadri, Lee, Liu, Mutlu, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.

SAFARI

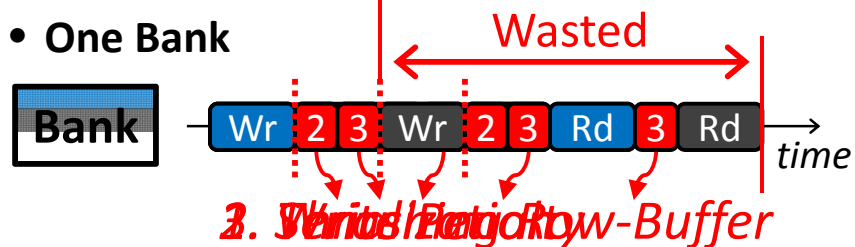
84

The Problem with Memory Bank Conflicts

- **Two Banks**



- **One Bank**

**SAFARI**

85

Goal

- **Goal:** *Mitigate the detrimental effects of **bank conflicts** in a cost-effective manner*
- **Naïve solution:** Add more banks
 - **Very expensive**
- **Cost-effective solution:** Approximate the benefits of more banks without

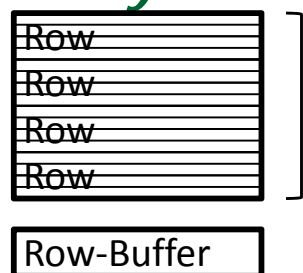
the

86

Key Observation #1

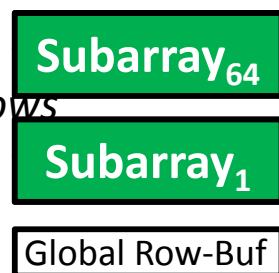
A DRAM bank is divided into

Logical Bank



A single row-buffer
cannot drive *all* rows

Physical Bank



Many *local row-buffers*,
one at each *subarray*

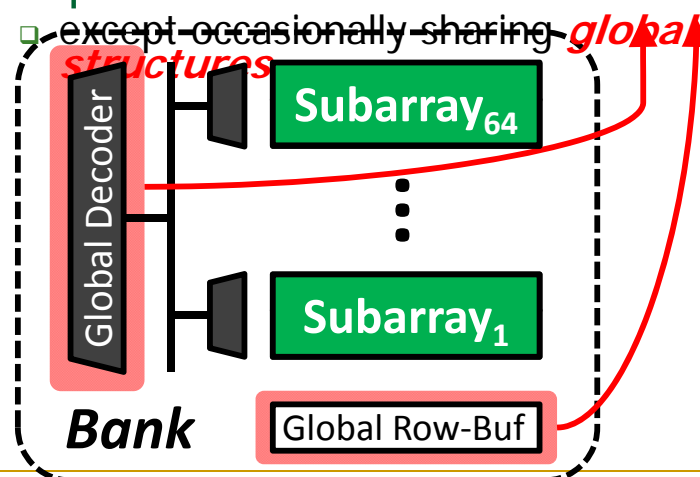
SAFARI

87

Key Observation #2

Each subarray is mostly independent...

- except occasionally sharing *global structures*

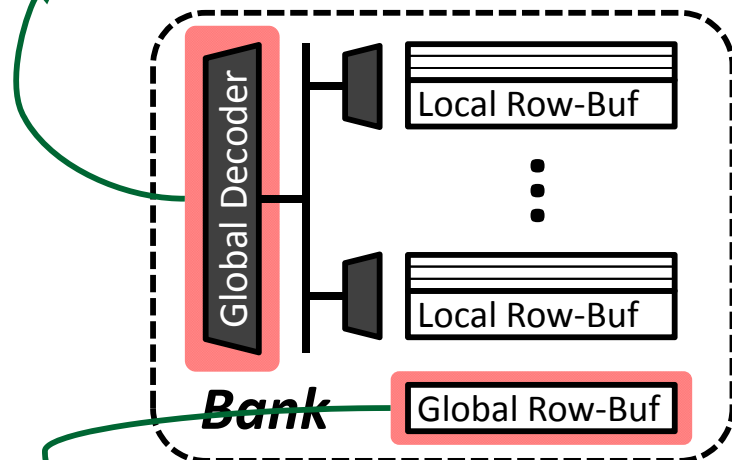


SAFARI

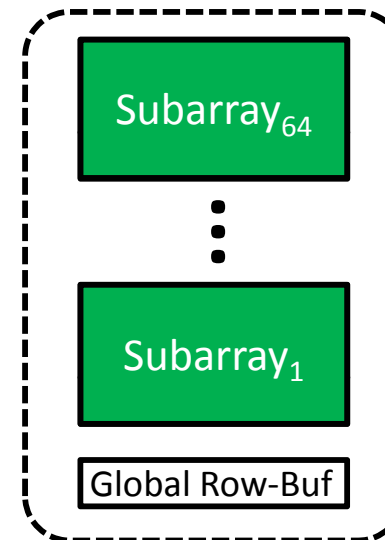
88


Key Idea: Reduce Sharing of Globals

1. Parallel access to subarrays



Overview of Our Mechanism



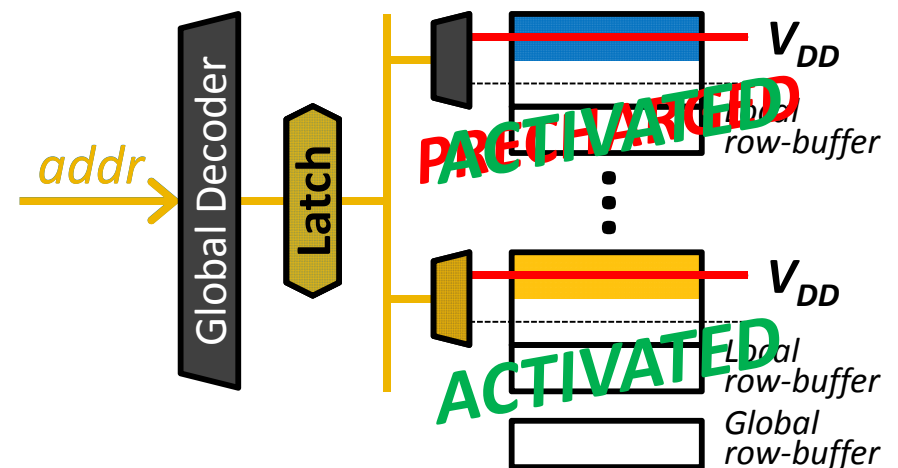
1. Parallelize
2. Utilize multiple  To some bankers but diff. subarrays

Challenges: Global Structures

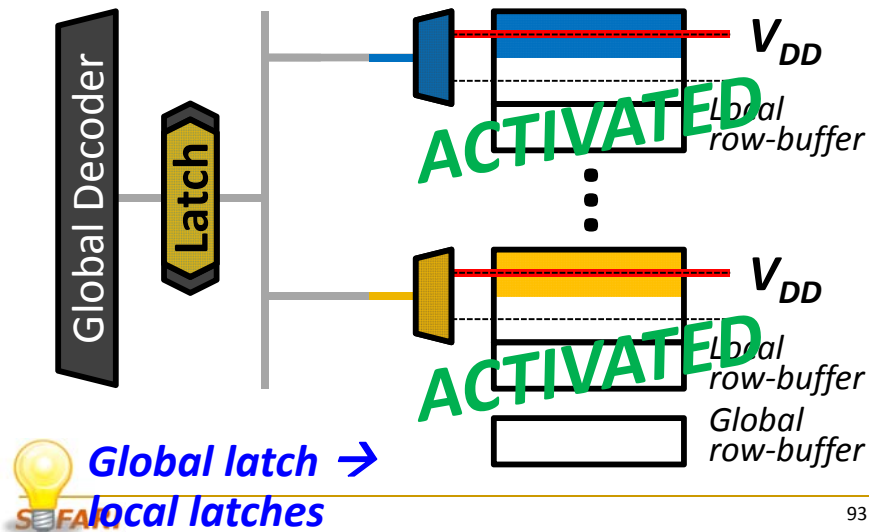
1. Global Address Latch

2. Global Bitlines

Challenge #1. Global Address Latch



Solution #1. Subarray Address Latch



93

Challenges: Global Structures

1. Global Address Latch

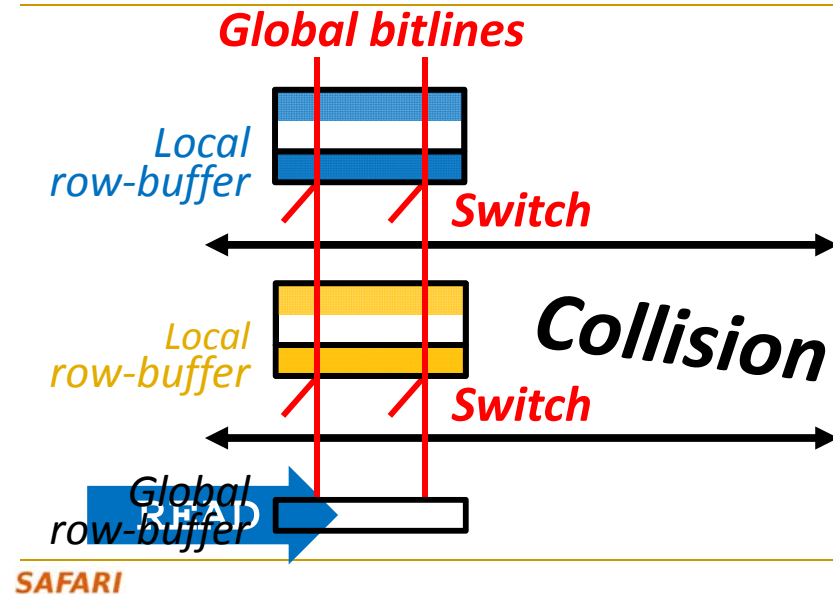
- Problem: Only one raised wordline
- Solution: **Subarray Address Latch**

2. Global Bitlines

SAFARI

94

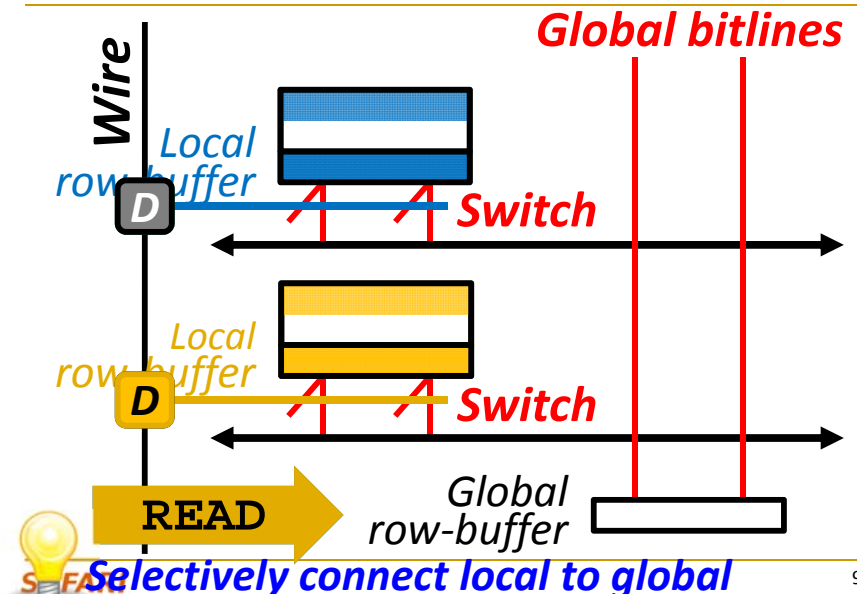
Challenge #2. Global Bitlines



SAFARI

95

Solution #2. Designated-Bit Latch



SAFARI

96

Challenges: Global Structures

1. Global Address Latch

- Problem: Only one raised wordline
- Solution: **Subarray Address Latch**

2. Global Bitlines

- Problem: Collision during access

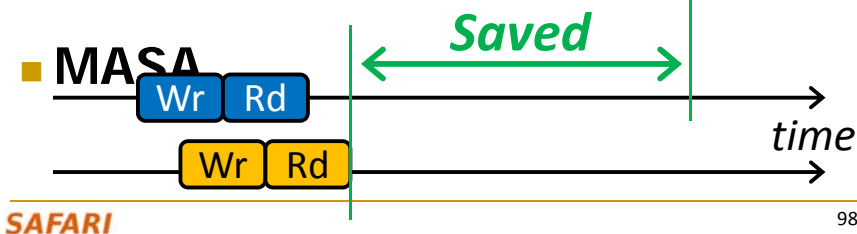
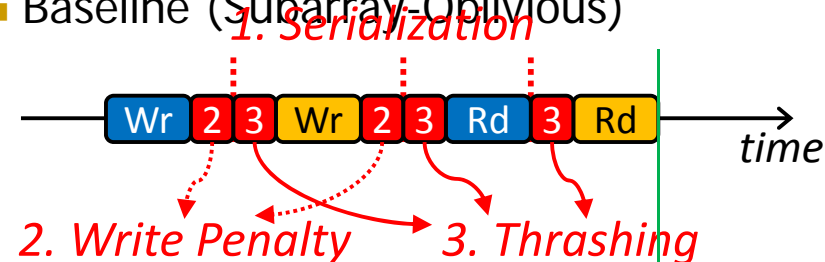
MASA (Multitude of Activated Subarrays)

SAFARI

97

MASA: Advantages

- Baseline (Subarray-Oblivious)



SAFARI

98

MASA: Overhead

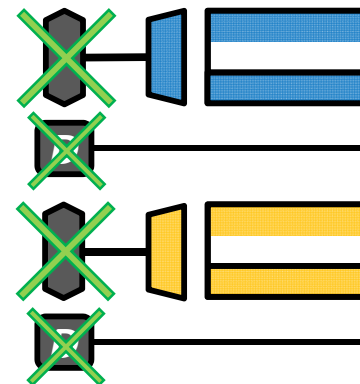
- **DRAM Die Size:** 0.15% increase
 - Subarray Address Latches
 - Designated-Bit Latches & Wire
- **DRAM Static Energy:** Small increase
 - 0.56mW for each activated subarray
 - *But saves dynamic energy*
- **Controller:** Small additional storage
 - Keep track of subarray status (< 256B)
 - Keep track of new timing constraints

SAFARI

99

Cheaper Mechanisms

Latches



1. *Serialization*

2. *Wr-Penalty*

3. *Thrashing*

MASA

SALP-2

SALP-1

SAFARI

100

System Configuration

System Configuration

- CPU: 5.3GHz, 128 ROB, 8 MSHR
- LLC: 512kB per-core slice

Memory Configuration

- DDR3-1066
- (default) 1 channel, 1 rank, 8 banks, 8 subarrays-per-bank
- (sensitivity) 1-8 chans, 1-8 ranks, 8-64 banks, 1-128 subarrays

Mapping & Row-Policy

- (default) Line-interleaved & Closed-row
- (sensitivity) Row-interleaved & Open-row

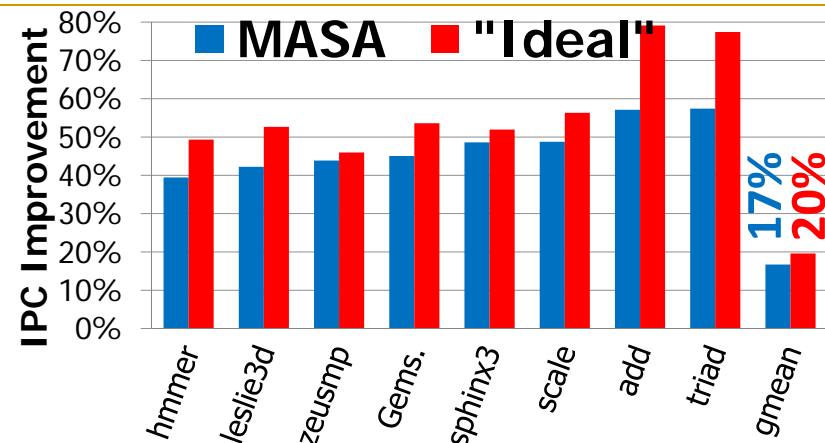
DRAM Controller Configuration

- 64-/64-entry read/write queues per-channel
- RR-FCFS, batch scheduling for writes

SAFARI

101

SALP: Single-core Results

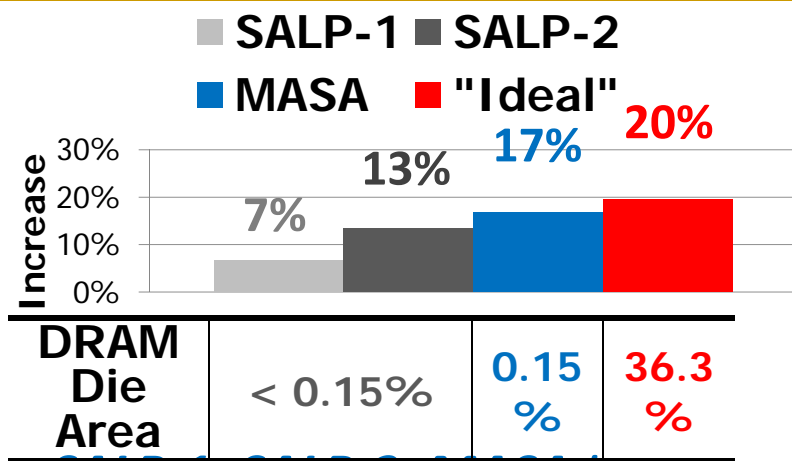


MASA achieves most of the benefit of having more banks ("Ideal")

SAFARI

102

SALP: Single-Core Results

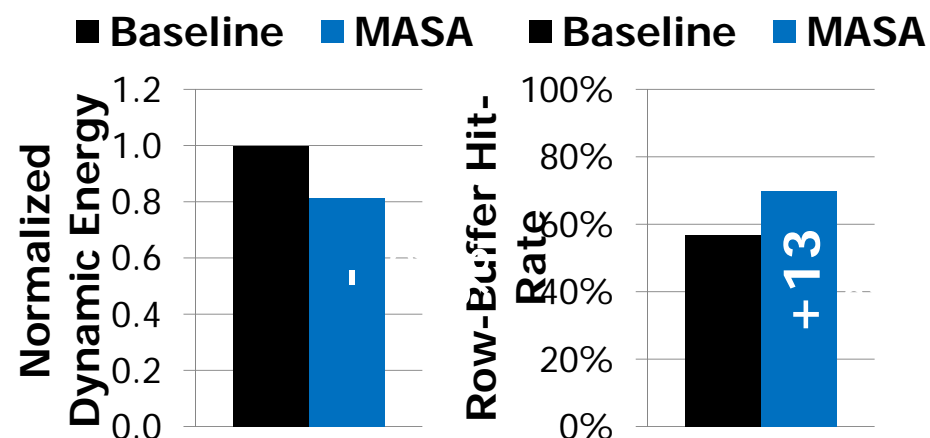


SALP-1, SALP-2, MASA improve performance at low cost

SAFARI

103

Subarray-Level Parallelism: Results



MASA increases energy-efficiency

SAFARI

104

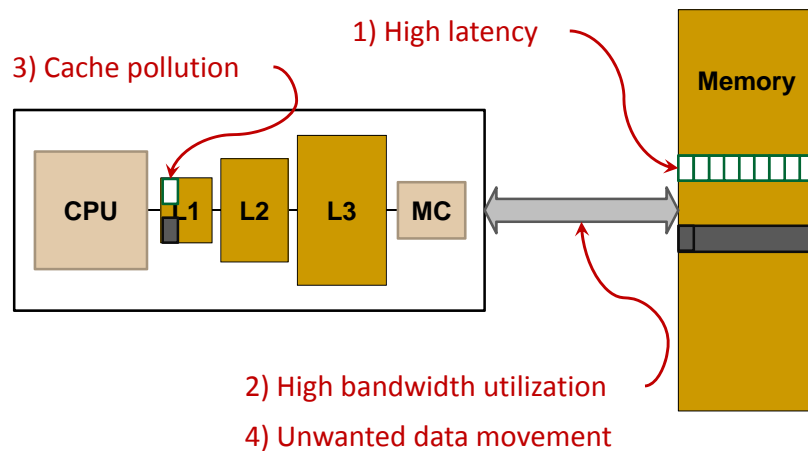
New DRAM Architectures

- RAIDR: Reducing Refresh Impact
- TL-DRAM: Reducing DRAM Latency
- SALP: Reducing Bank Conflict Impact
- RowClone: Fast Bulk Data Copy and Initialization

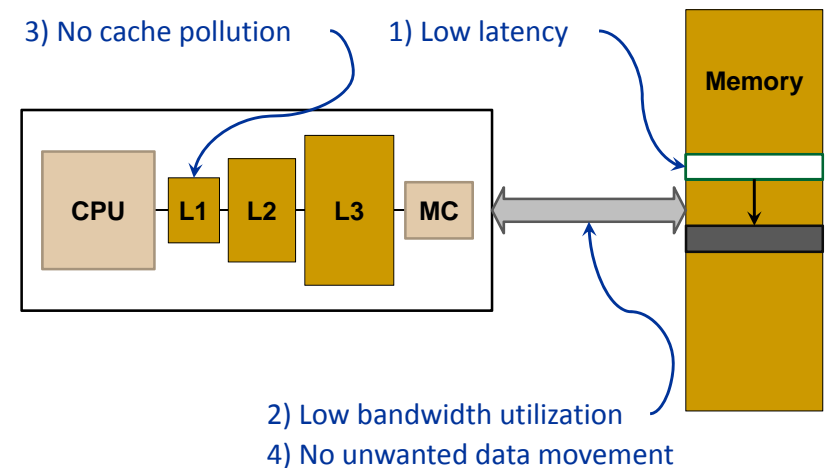
RowClone: Fast Bulk Data Copy and Initialization

Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, Todd C. Mowry, **"RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data"**
CMU Computer Science Technical Report, CMU-CS-13-108, Carnegie Mellon University, April 2013.

Today's Memory: Bulk Data Copy

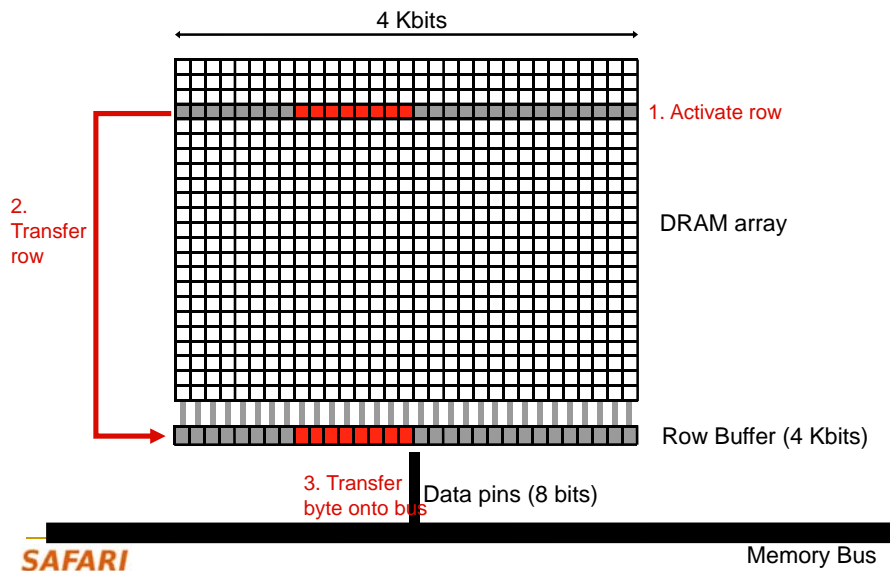


Future: RowClone (In-Memory Copy)

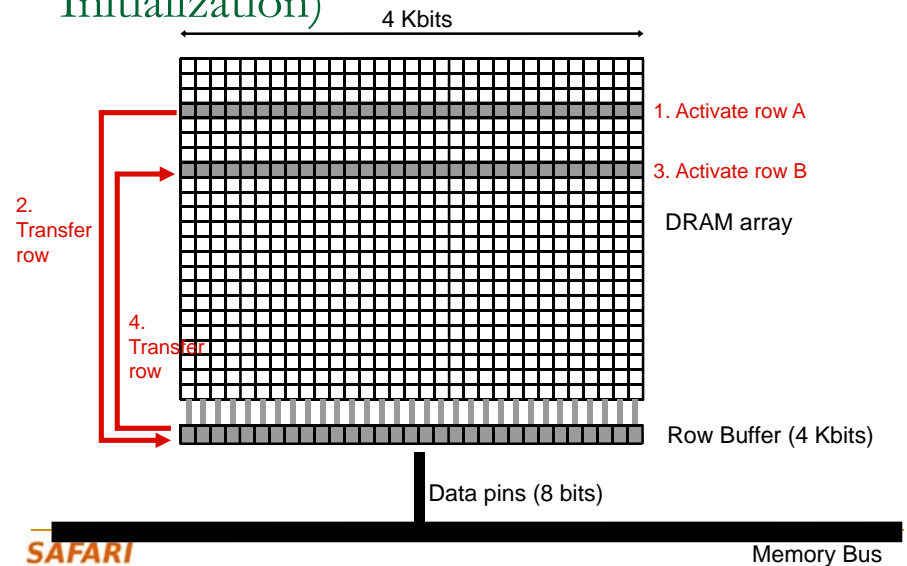


Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," CMU Tech Report 2013.

DRAM operation (load one byte)



RowClone: in-DRAM Row Copy (and Initialization)

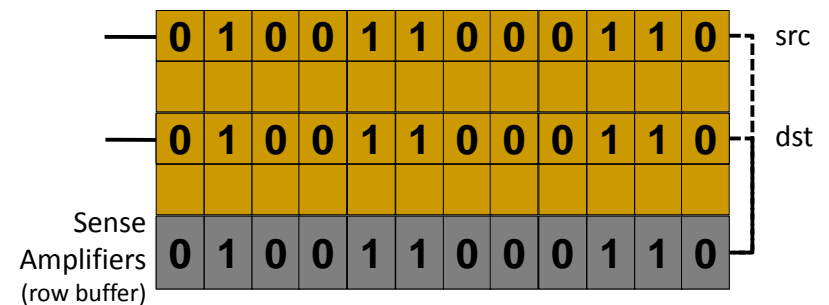


RowClone: Key Idea

- DRAM banks contain
 1. Multiple rows of DRAM cells – row = 8KB
 2. A row buffer shared by the DRAM rows
- Large scale copy
 1. Copy data from source row to row buffer
 2. Copy data from row buffer to destination row

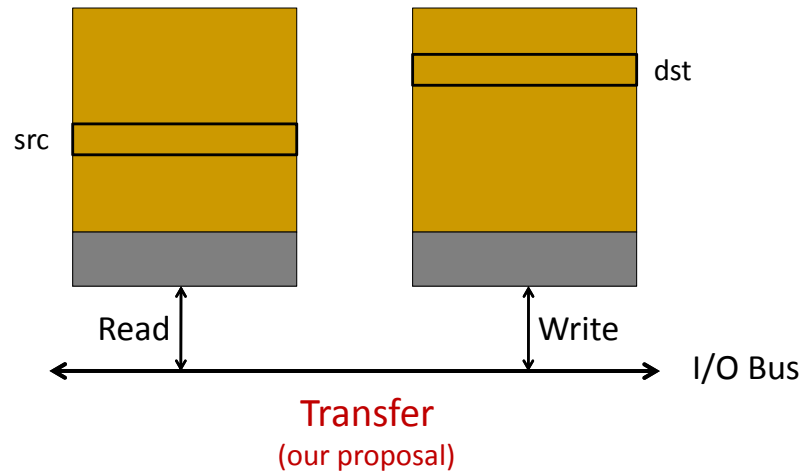
Can be accomplished by two consecutive **ACTIVATES** (if source and destination rows are in the same subarray)

RowClone: Intra-subarray Copy



Activate (src) → Deactivate (our proposal) → Activate (dst)

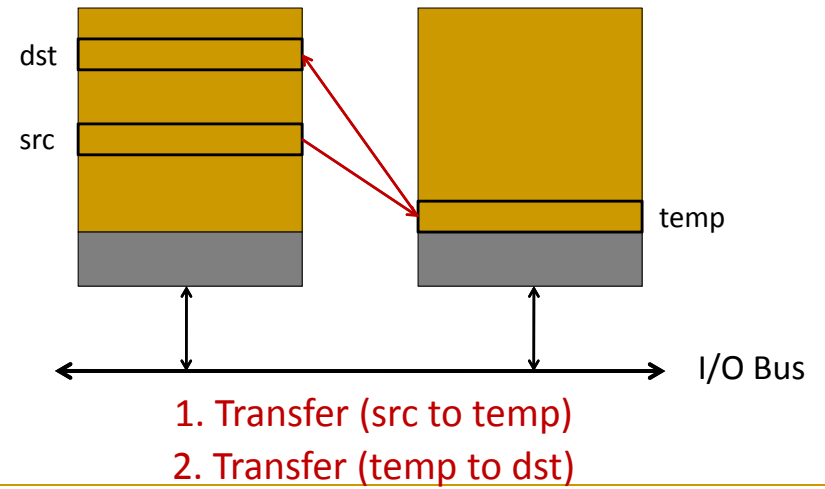
RowClone: Inter-bank Copy



SAFARI

113

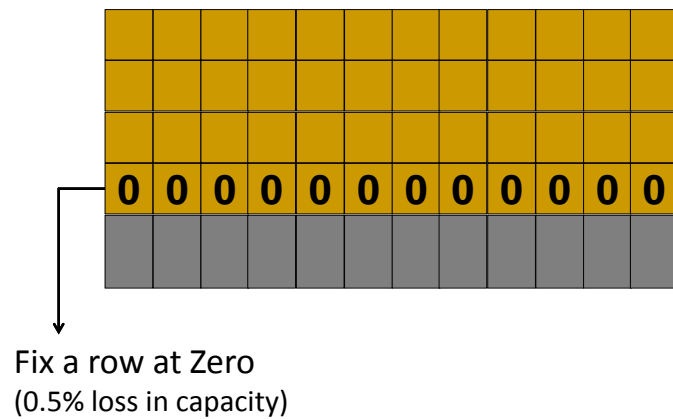
RowClone: Inter-subarray Copy



SAFARI

114

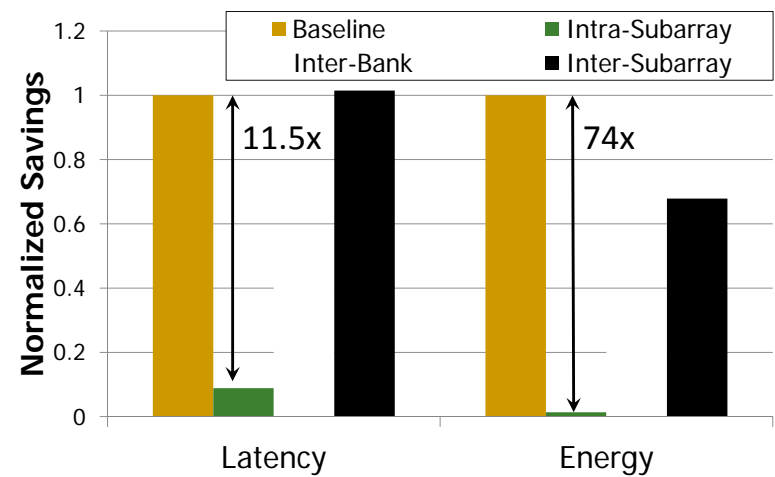
Fast Row Initialization



SAFARI

115

RowClone: Latency and Energy Savings



Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," CMU Tech Report 2013.

116

RowClone: Latency and Energy Savings

Mechanism	Absolute		Reduction	
	Latency (ns)	Energy (μ J)	Latency	Energy
4KB Copy				
Baseline	1046	3.6	1.00	1.0
Intra-subarray	90	0.04	11.62	74.4
Inter-Bank - PSM	540	1.1	1.93	3.2
Intra-Bank - PSM	1050	2.5	0.99	1.5
4KB Zeroing				
Baseline	546	2.0	1.00	1.0
Intra-subarray	90	0.05	6.06	41.5

Table 3: Latency and energy reductions due to RowClone

SAFARI

117

RowClone: Overall Performance

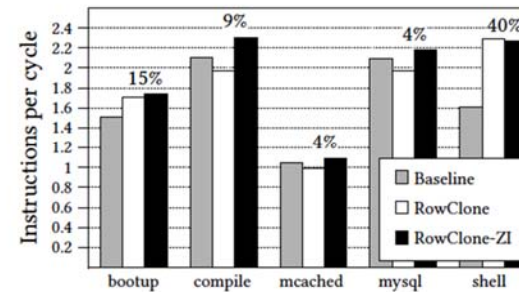


Figure 10: Performance improvement of RowClone-ZI. Value on top indicates percentage improvement compared to baseline.

Application	bootup	compile	mcached	mysql	shell
Energy Reduction	40%	32%	15%	17%	67%

	Number of Cores		
	2	4	8
Number of Workloads	138	50	40
Weighted Speedup Improvement	15%	20%	27%
Energy per Instruction Reduction	19%	17%	17%

SAFARI

Summary

- Major problems with DRAM scaling and design: high refresh rate, high latency, low parallelism, bulk data movement
- Four new DRAM designs
 - RAIDR: Reduces refresh impact
 - TL-DRAM: Reduces DRAM latency at low cost
 - SALP: Improves DRAM parallelism
 - RowClone: Reduces energy and performance impact of bulk data copy
- All four designs
 - Improve both performance and energy consumption
 - Are low cost (low DRAM area overhead)
 - Enable new degrees of freedom to software & controllers
- Rethinking DRAM interface and design essential for scaling
 - Co-design DRAM with the rest of the system

SAFARI

119

Computer Architecture: Main Memory (Part III)

Prof. Onur Mutlu
Carnegie Mellon University