18-742:
Computer Architecture & Systems

Row Clone:
Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Prof. Phillip Gibbons

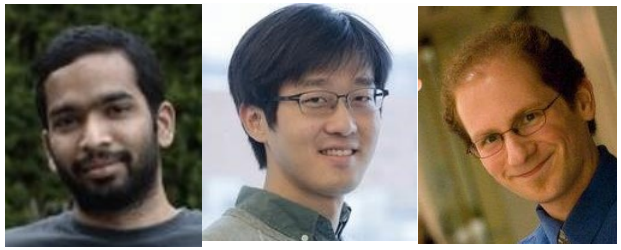
Spring 2025, Lecture 9

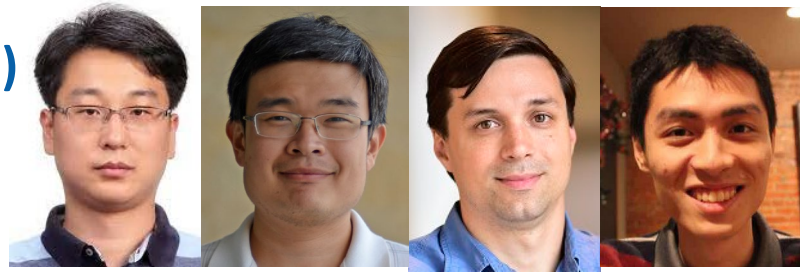
"RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"

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

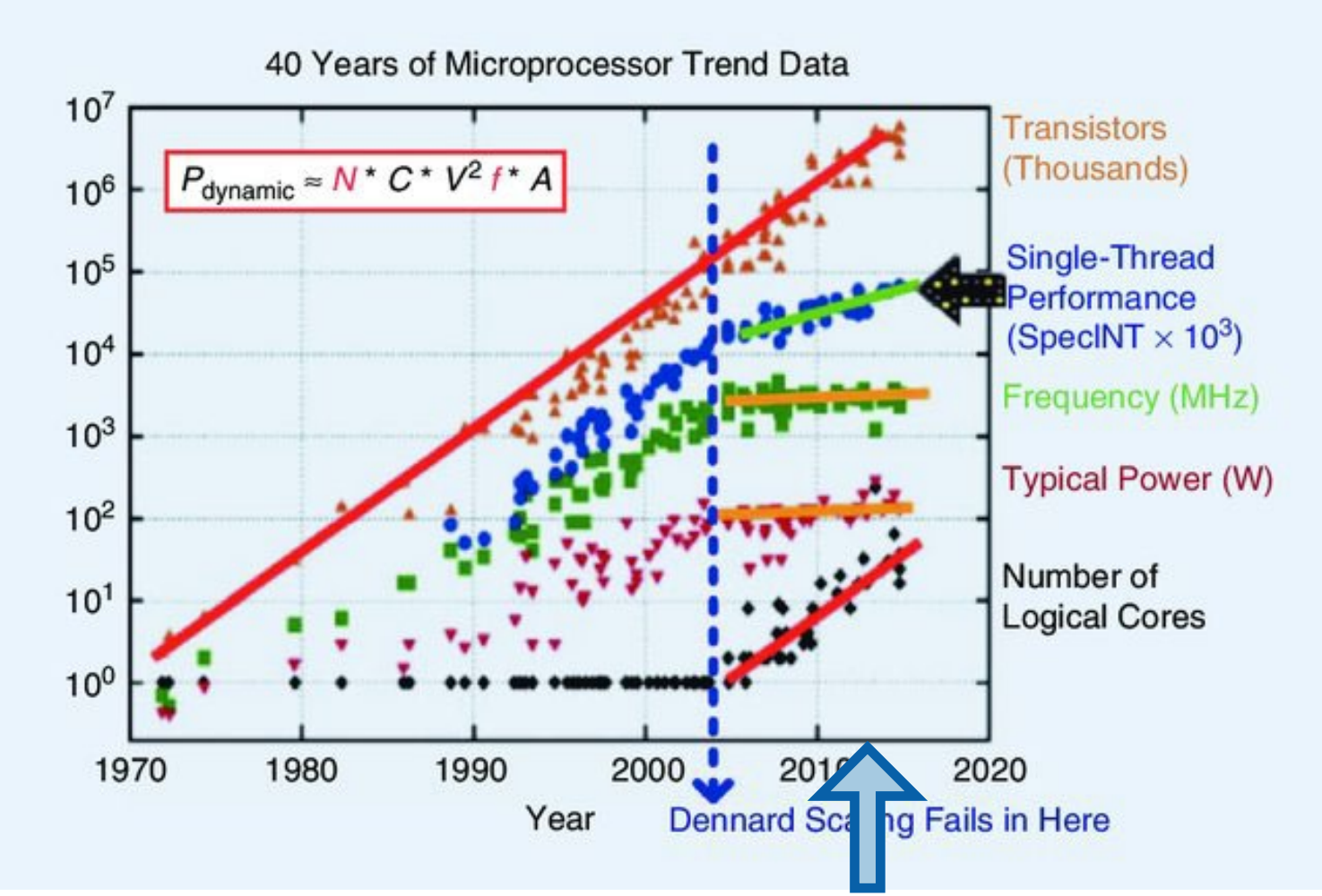
2013

- **Vivek:** CMU PhD, Principal Researcher MSR India
- **Yoongu:** CMU PhD, now Google
- **Chris:** CMU PhD, now Fastly
- **Donghyuk:** CMU PhD, now Nvidia
- **Rachata:** CMU PhD, now TGGS prof (Thailand)
- **Gennady:** CMU PhD, now U.Toronto Prof
- **Yixin:** CMU PhD, now Google
- **Onur:** CMU prof, now ETH Prof, many awards
- **Phil:** PE Intel, now standing in front of you
- **Michael:** Principal Engineer (PE) Intel
- **Todd:** CMU Prof, ACM Fellow





Moore's Law w/o Dennard Scaling



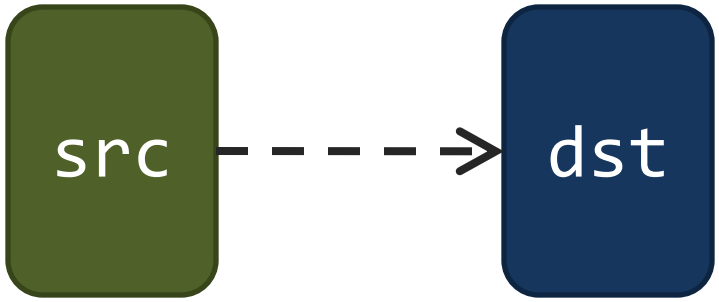
Executive Summary

- Bulk data copy and initialization
  - Unnecessarily moves data on the memory channel
  - Degrades system performance and energy efficiency
- **RowClone** – perform copy in DRAM with low cost
  - Uses row buffer to copy large quantity of data
  - **Source row → row buffer → destination row**
  - 11X lower latency and 74X lower energy for a bulk copy
- Accelerate Copy-on-Write and Bulk Zeroing
  - Forking, checkpointing, zeroing (security), VM cloning
- Improves performance and energy efficiency at low cost
  - 27% and 17% for 8-core systems (0.01% DRAM chip area)

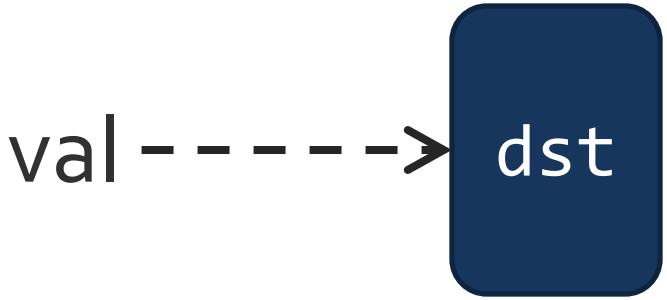
Slides adapted from our Micro'13 talk

Bulk Data Copy and Initialization

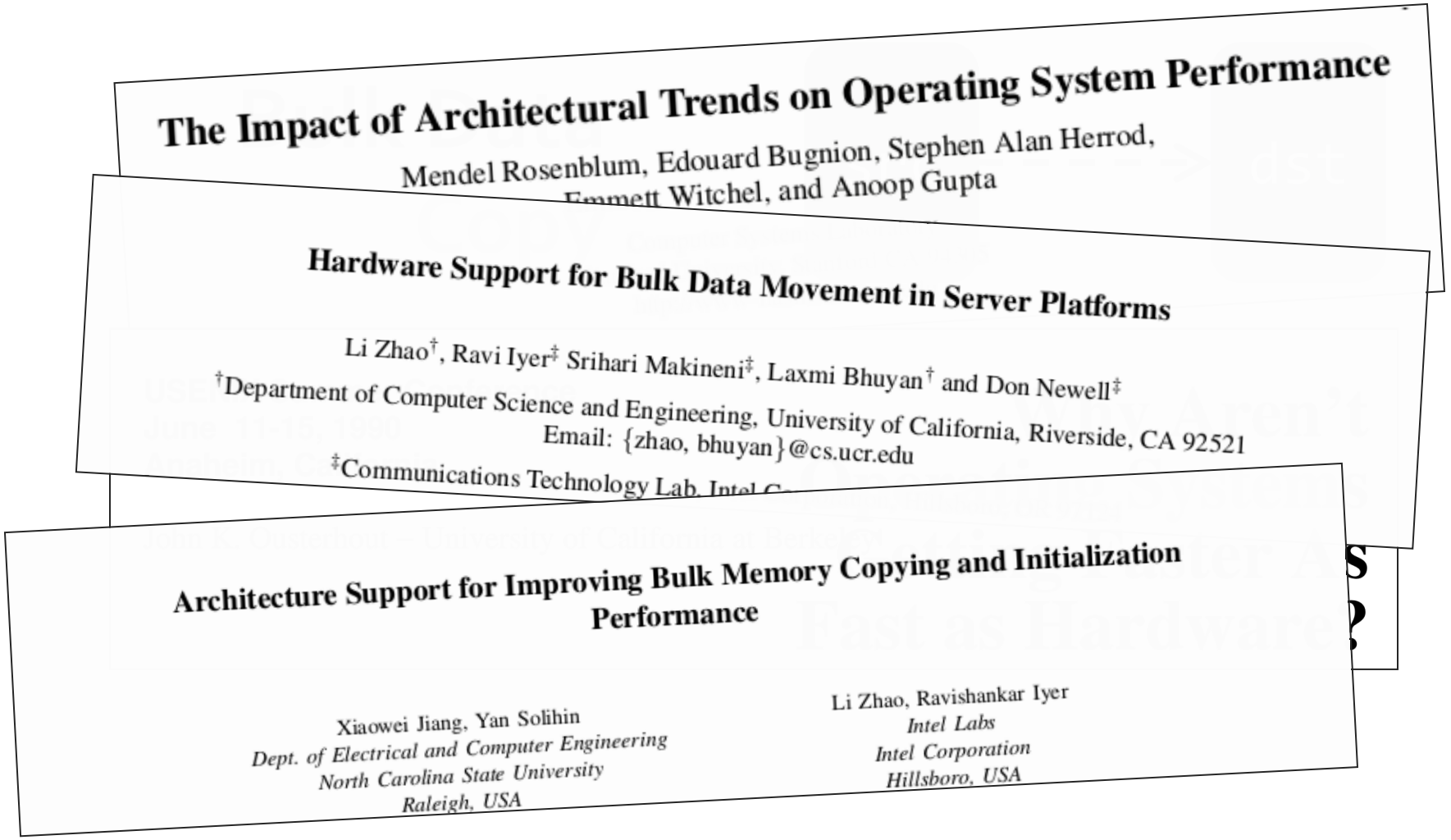
Bulk Data Copy



Bulk Data Initialization



Bulk Data Copy and Initialization



The Impact of Architectural Trends on Operating System Performance

Mendel Rosenblum, Edouard Bugnion, Stephen Alan Herrod,
Emmett Witchel, and Anoop Gupta

Hardware Support for Bulk Data Movement in Server Platforms

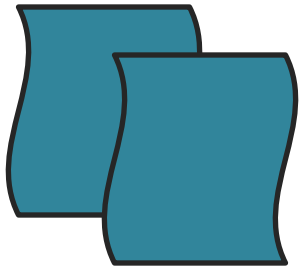
Li Zhao†, Ravi Iyer‡ Srihari Makineni‡, Laxmi Bhuyan† and Don Newell‡
†Department of Computer Science and Engineering, University of California, Riverside, CA 92521
Email: {zhao, bhuyan}@cs.ucr.edu
‡Communications Technology Lab, Intel C

Architecture Support for Improving Bulk Memory Copying and Initialization Performance

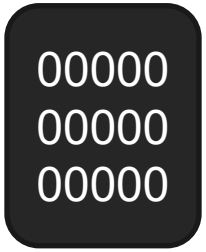
Xiaowei Jiang, Yan Solihin
Dept. of Electrical and Computer Engineering
North Carolina State University
Raleigh, USA

Li Zhao, Ravishankar Iyer
Intel Labs
Intel Corporation
Hillsboro, USA

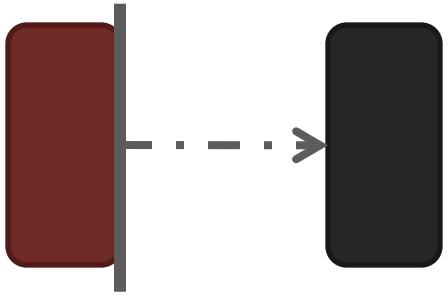
Bulk Copy and Initialization – Applications



Forking

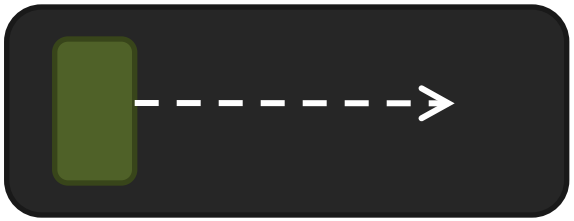


Zero initialization
(e.g., security)



Checkpointing

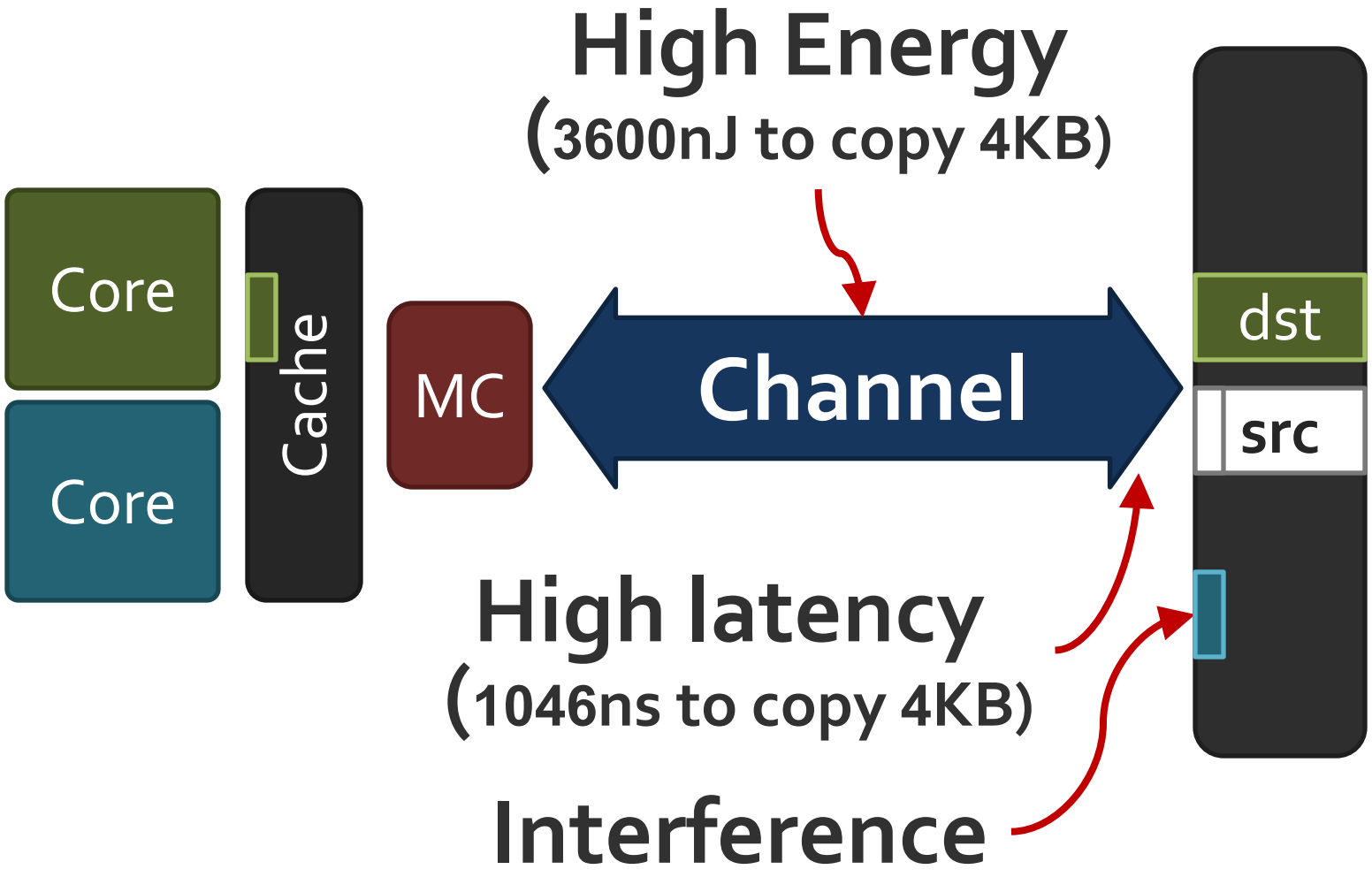
VM Cloning
Deduplication



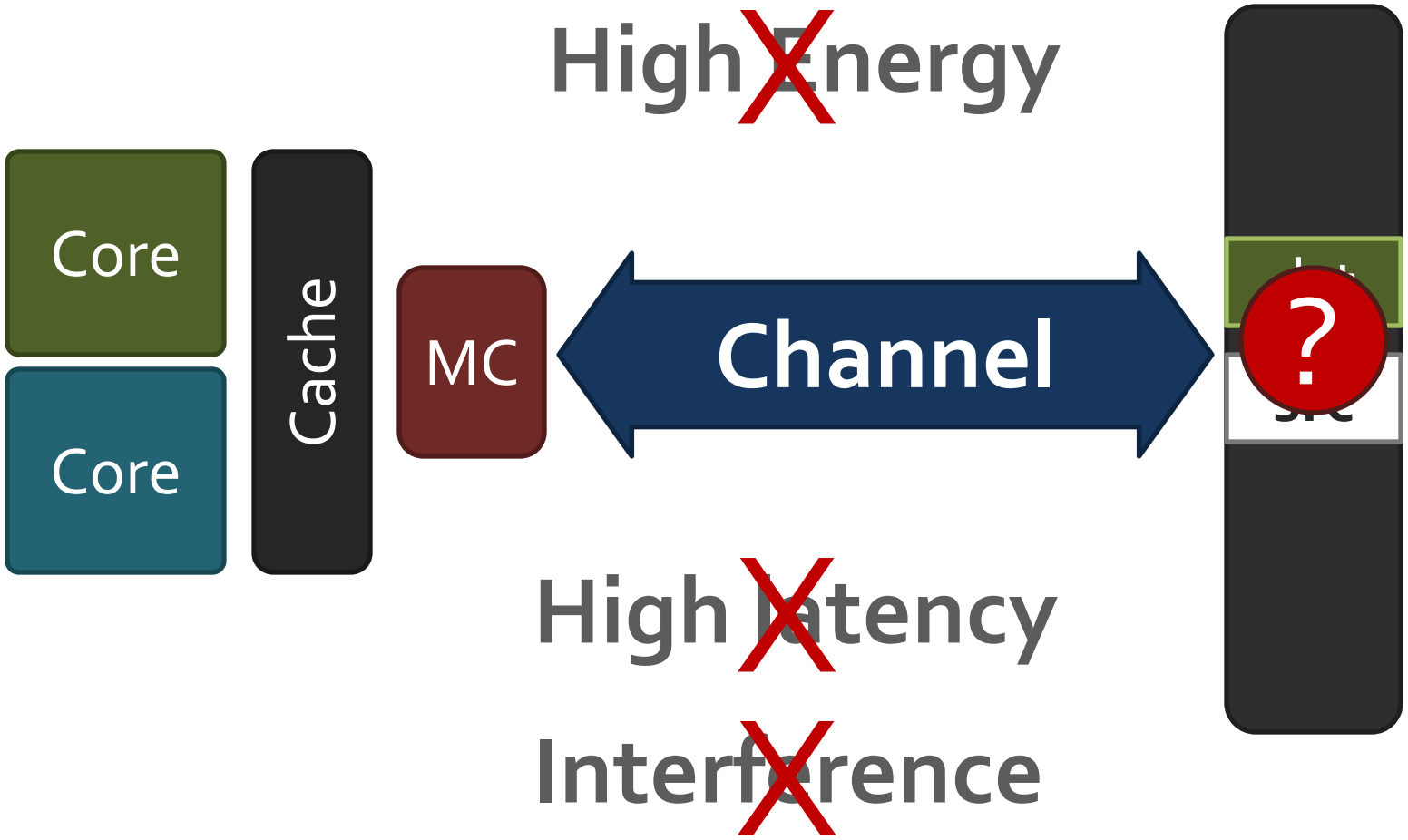
Page Migration

• • •
Many more

Shortcomings of Existing Copy/Init Approach



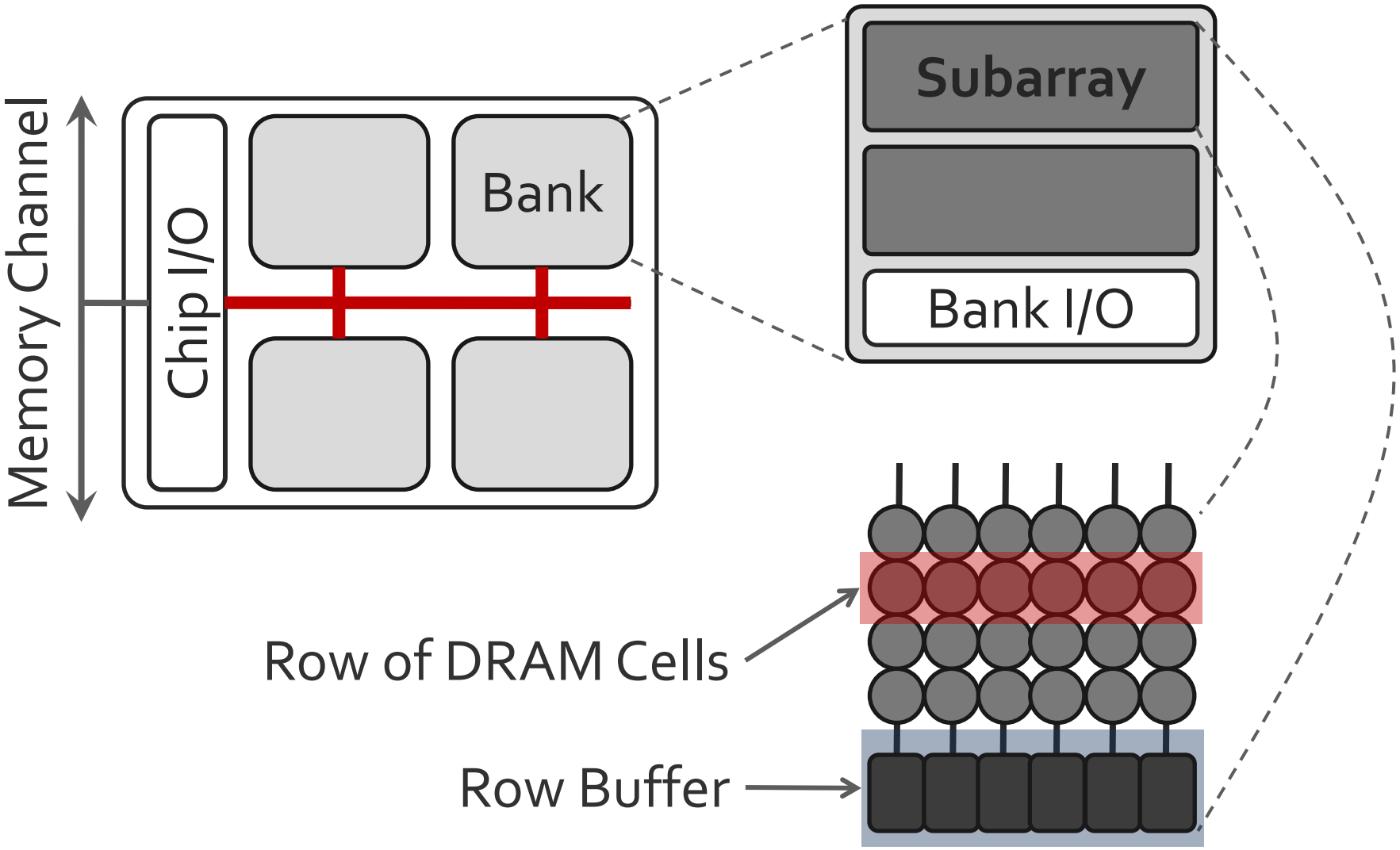
Our Approach: In-DRAM Copy with Low Cost



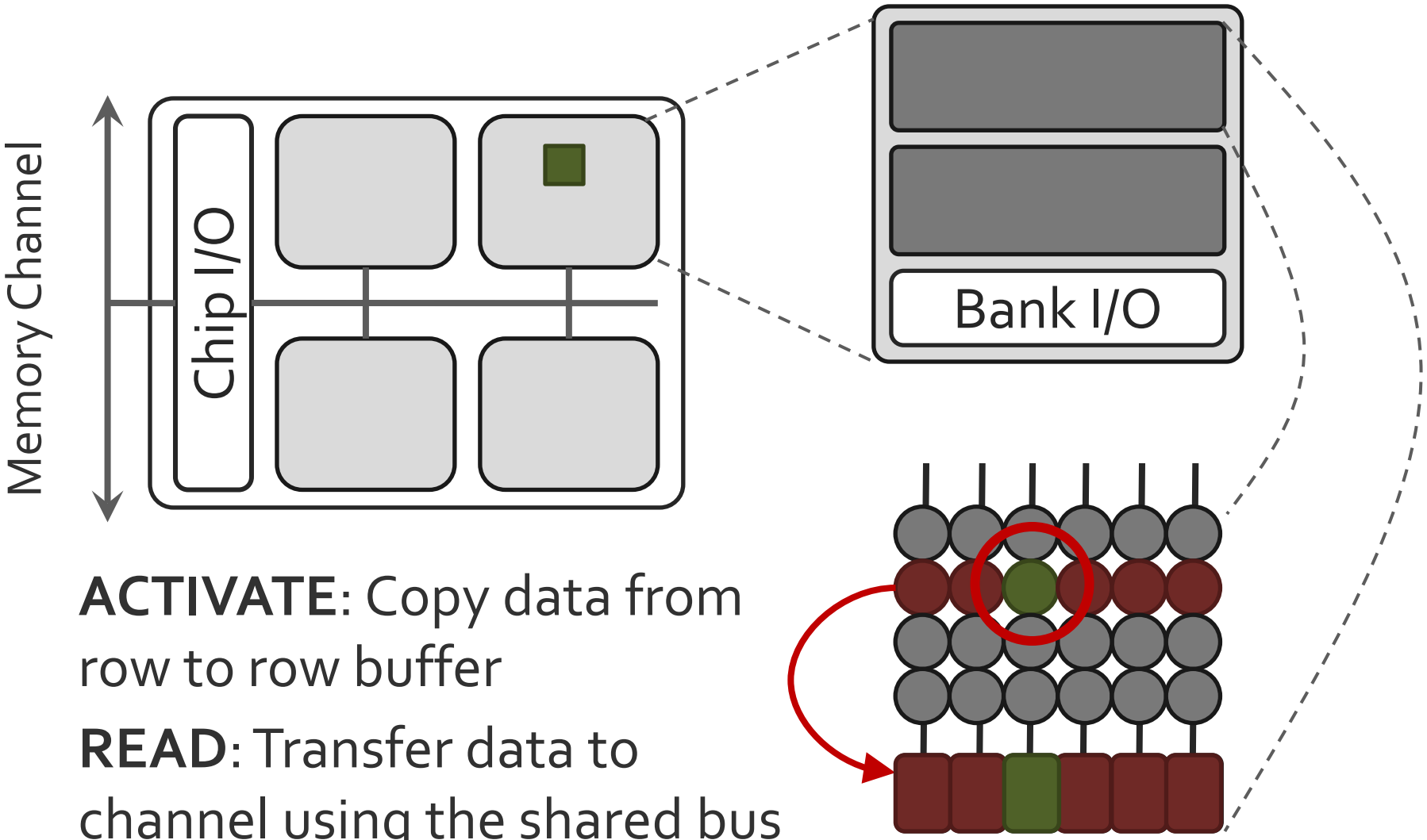
Outline

- ✓ Introduction
- DRAM Background
- RowClone
  - Fast Parallel Mode
  - Pipelined Serial Mode
- End-to-end Design
- Evaluation

DRAM Chip Organization



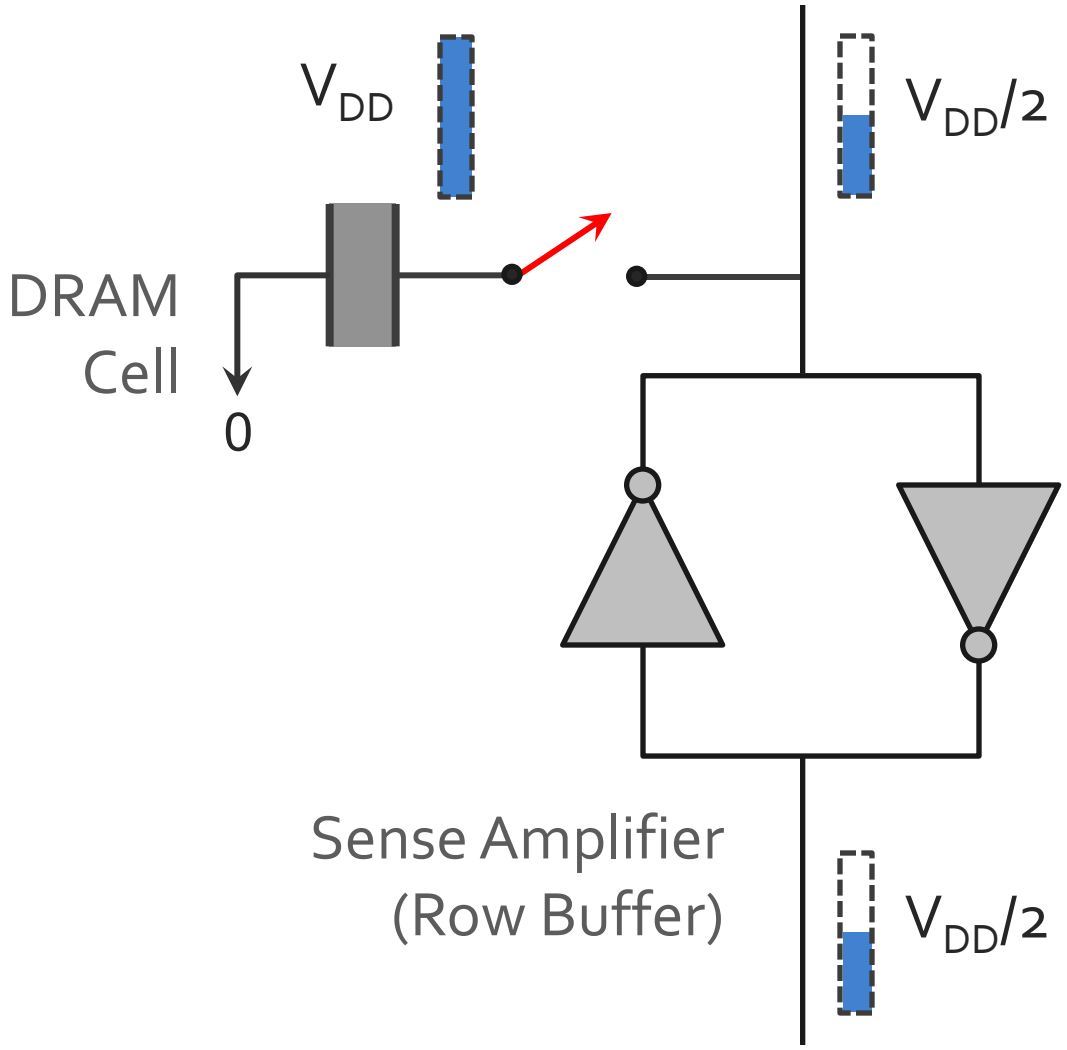
DRAM Read Operation



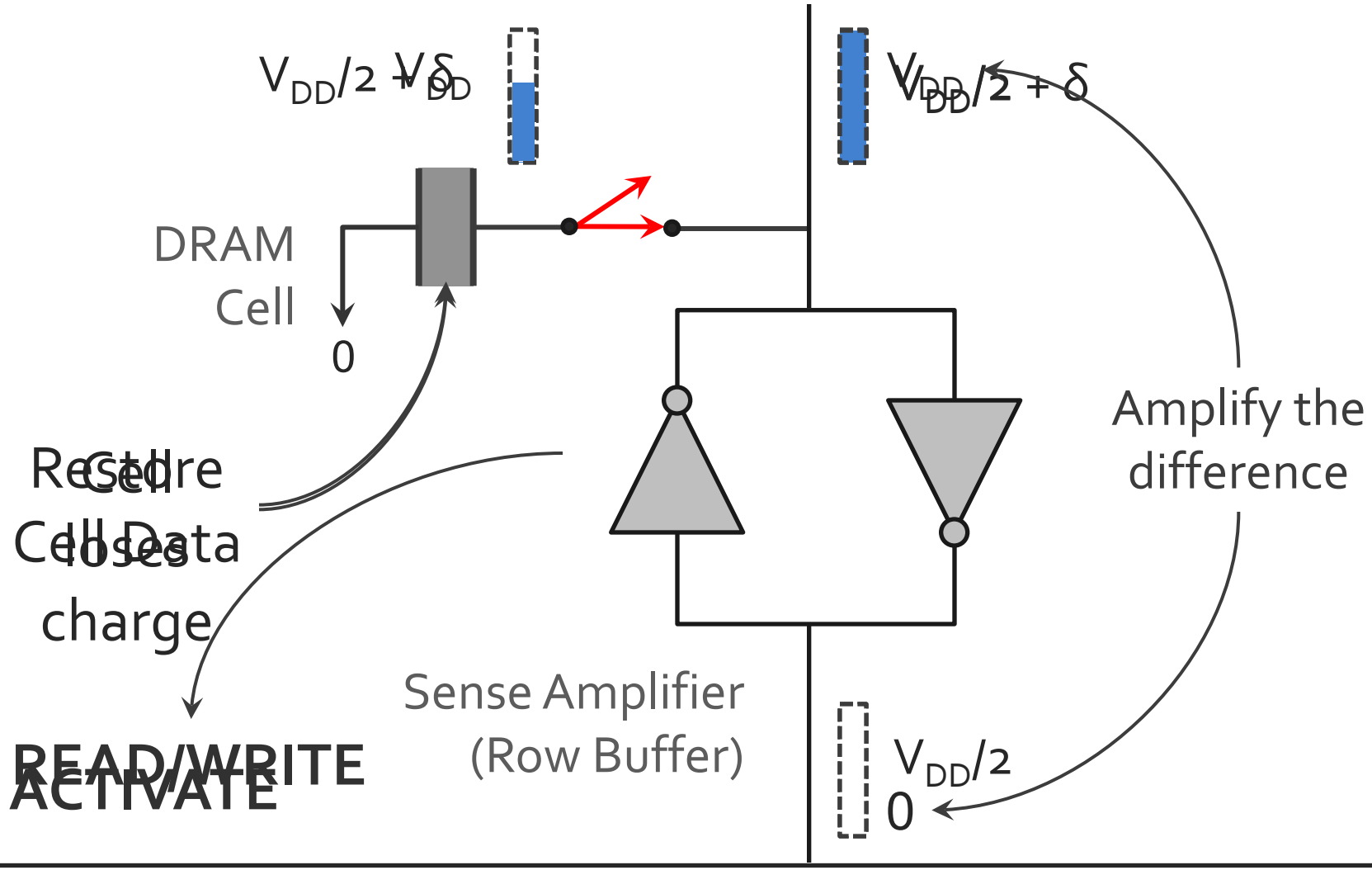
ACTIVATE: Copy data from row to row buffer

READ: Transfer data to channel using the shared bus

DRAM Cell Operation



DRAM Cell Operation

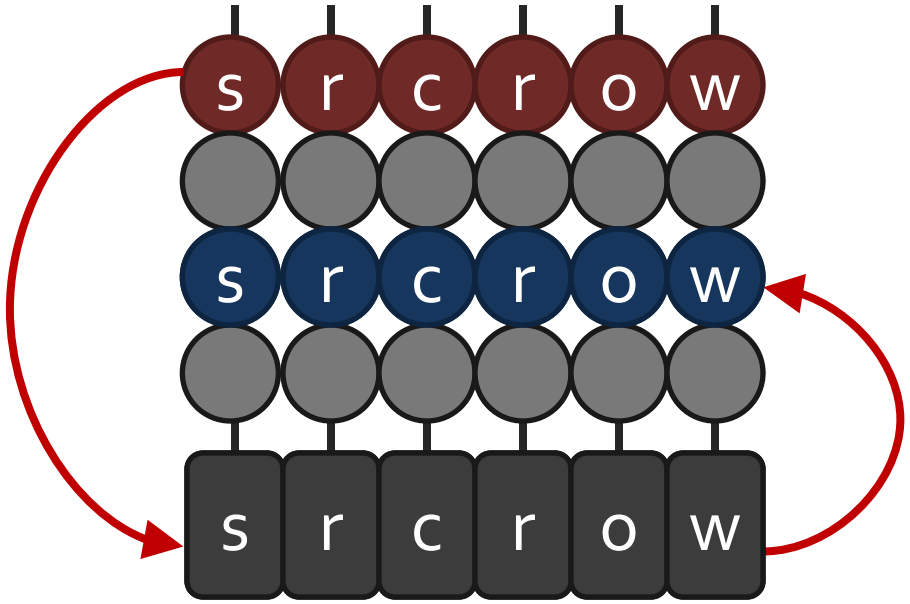


In the stable state, sense amplifier drives the cell

Outline

- ✓ Introduction
- ✓ DRAM Background
- RowClone
  - Fast Parallel Mode
  - Pipelined Serial Mode
- End-to-end Design
- Evaluation

RowClone: Fast Parallel Mode (FPM)

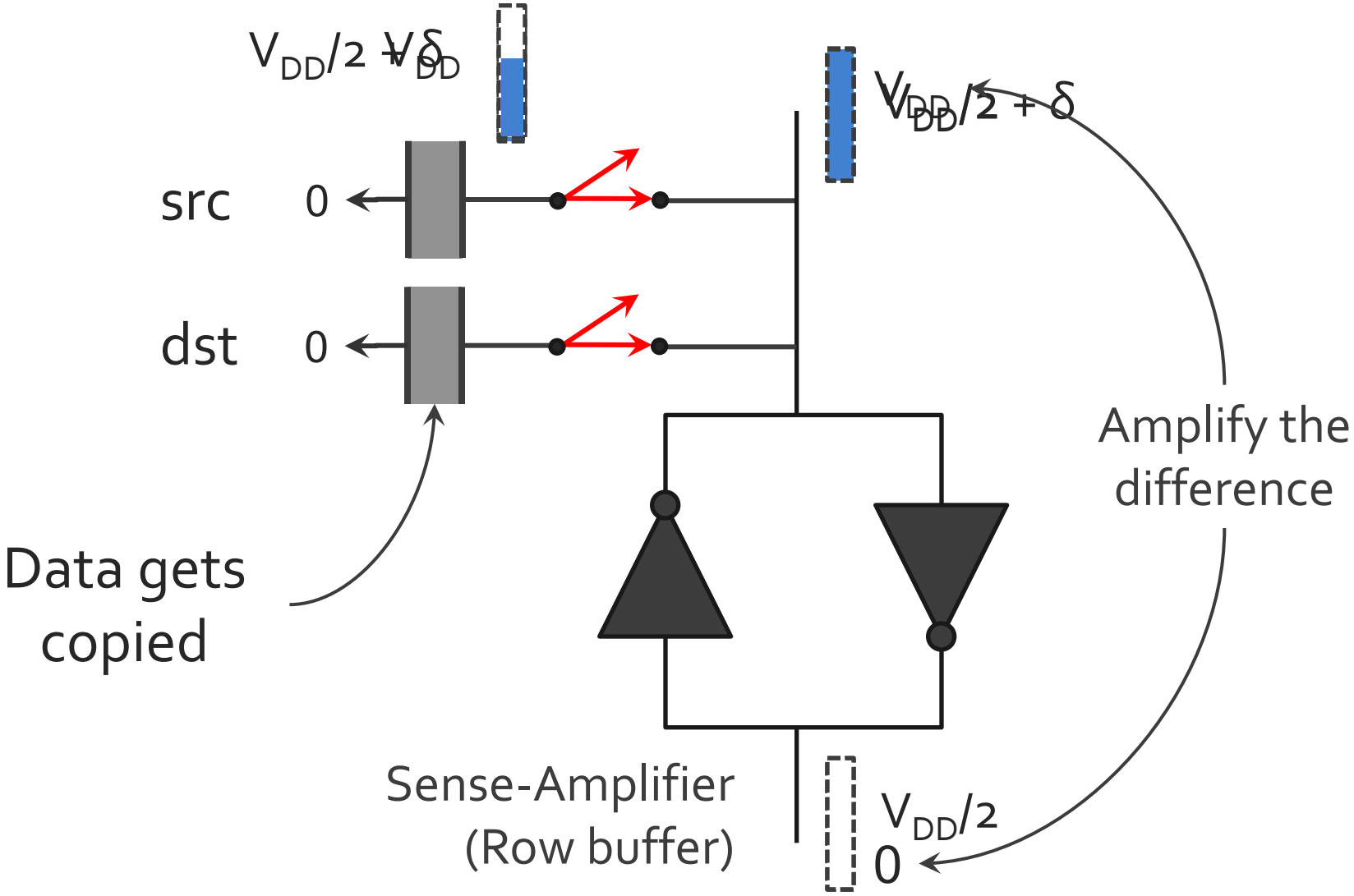


Row Buffer

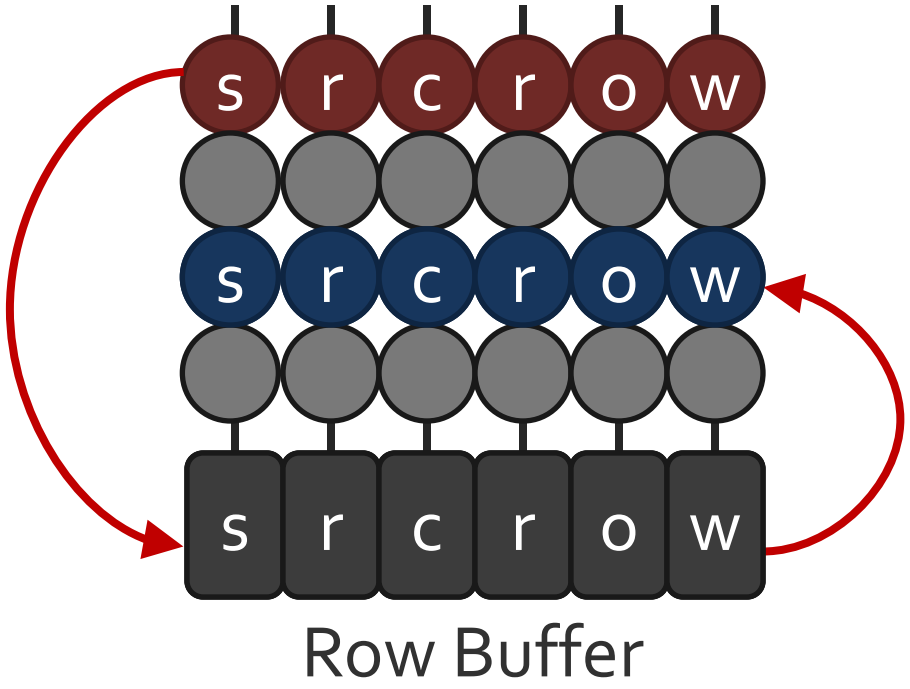
✓ 1. Source row to row buffer

? 2. Row buffer to destination row

Fast Parallel Mode: Implementation



Fast Parallel Mode: Implementation



1. **Activate** src row (copy data from src to row buffer)

2. **Activate** dst row (disconnect src from row buffer, connect dst – copy data from row buffer to dst)

Fast Parallel Mode: Benefits

Bulk Data Copy

1046ns to 90ns

3600nJ to 40nJ

No bandwidth consumption

Very little changes to the DRAM chip

Fast Parallel Mode: Constraints

- Location of source/destination
  - Both should be in the same subarray
- Size of the copy
  - Copies *all* the data from source row to destination

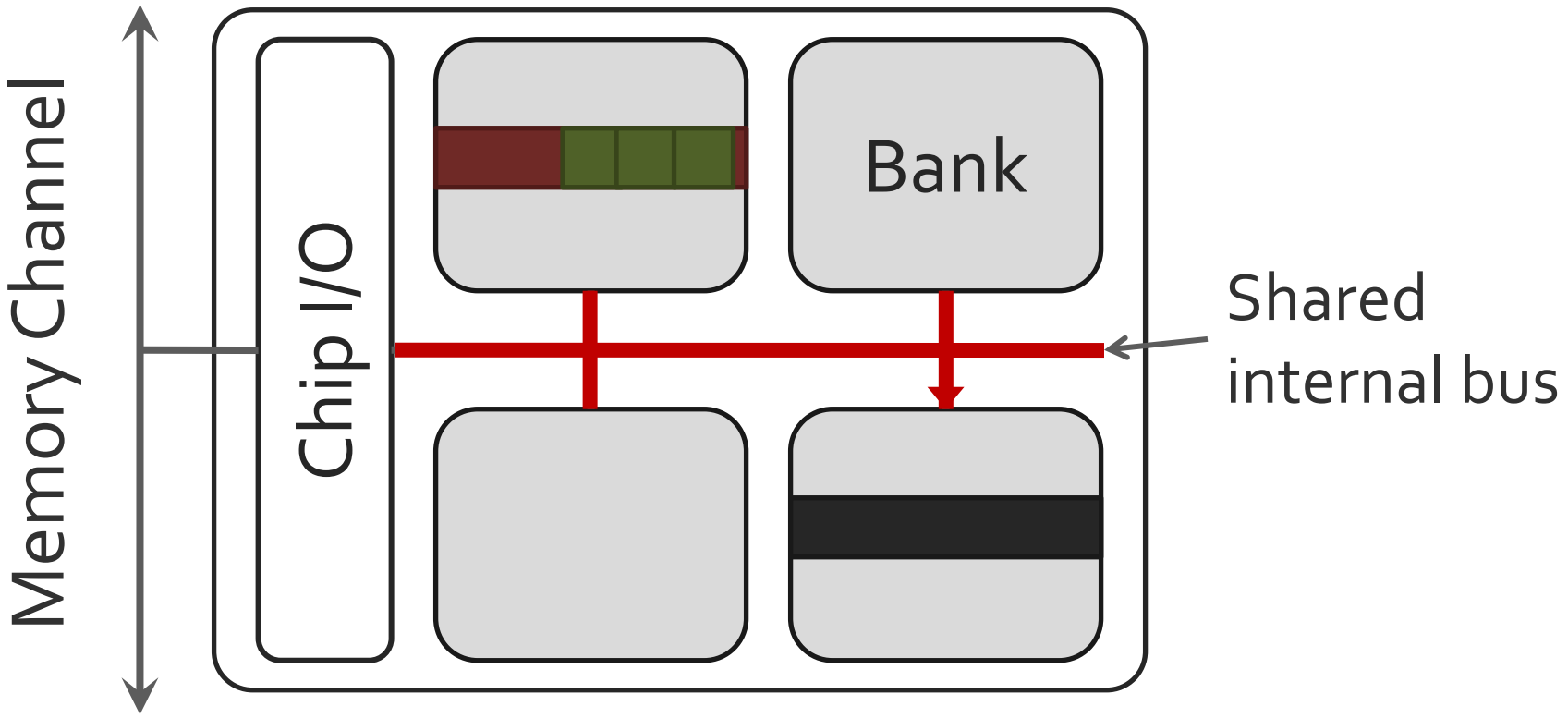
Discussion: Summary Question #1

What Did the Paper Get Right?

State the 3 most important things the paper says.

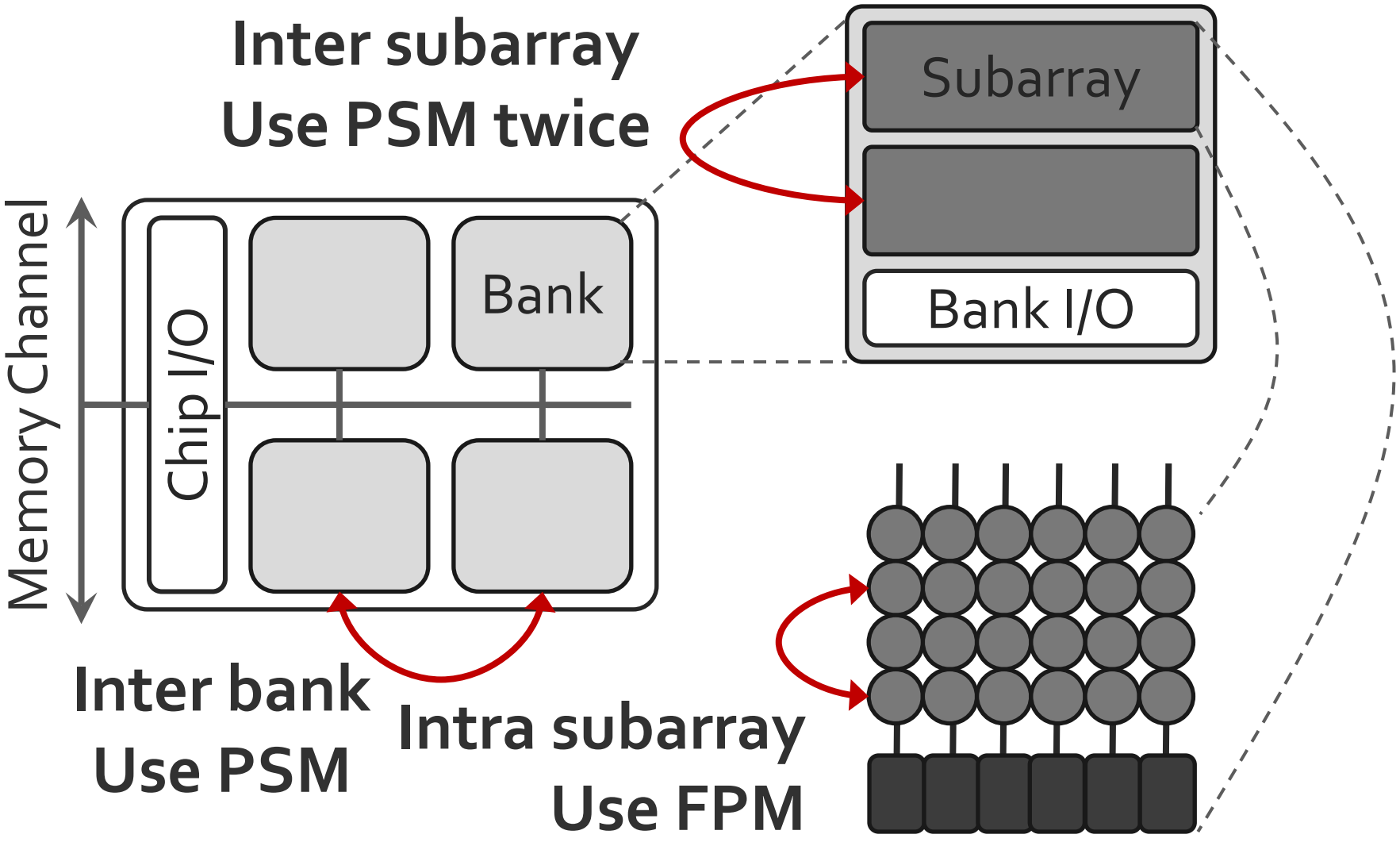
These could be some combination of the motivations, observations, interesting parts of the design, or clever parts of the implementation.

Pipelined Serial Mode (PSM)



Overlap the latency of the read and the write
1.9X latency reduction, 3.2X energy reduction

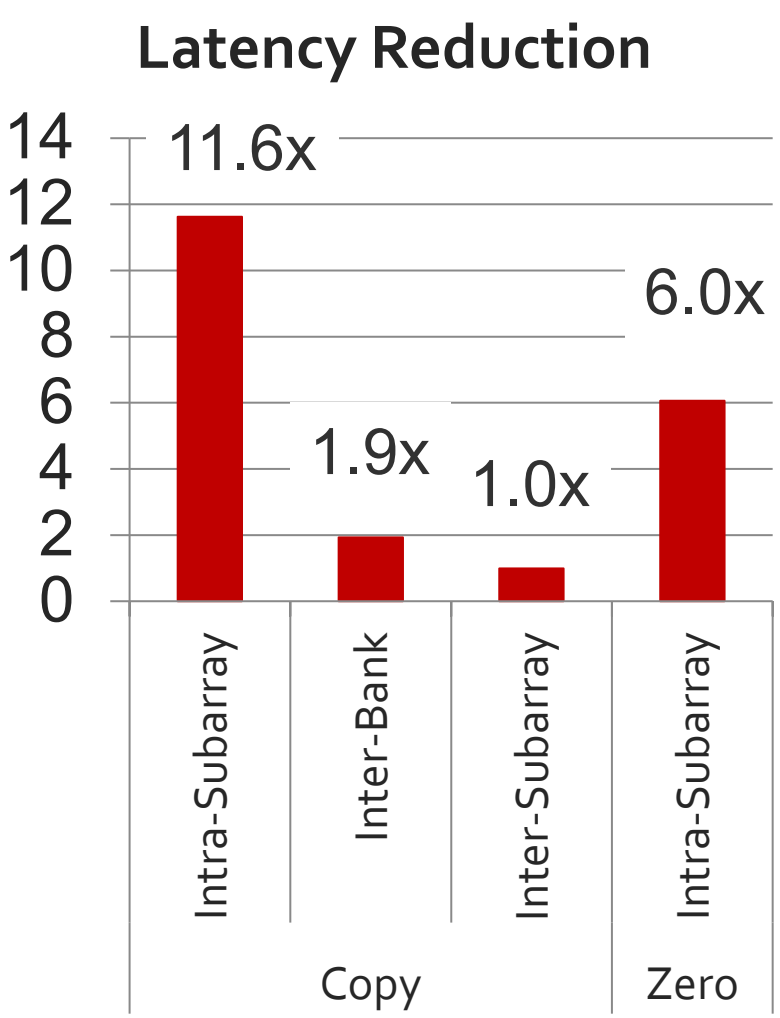
Bulk Copy using RowClone

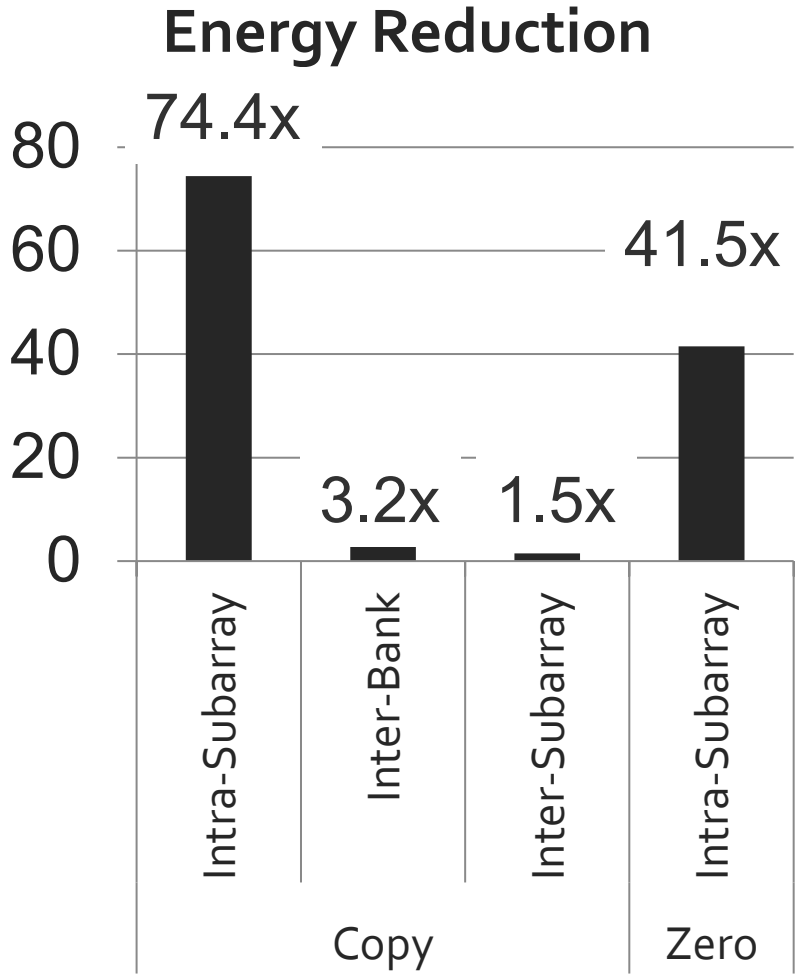


Bulk Initialization

- Initialization with arbitrary data
  - Initialize one row
  - Copy the data to other rows
- Zero initialization (most common)
  - Reserve a row in each subarray (always zero)
  - Copy data from reserved row (FPM mode)
  - **6.0X** lower latency, **41.5X** lower DRAM energy
  - 0.2% loss in capacity

Latency and Energy Benefits



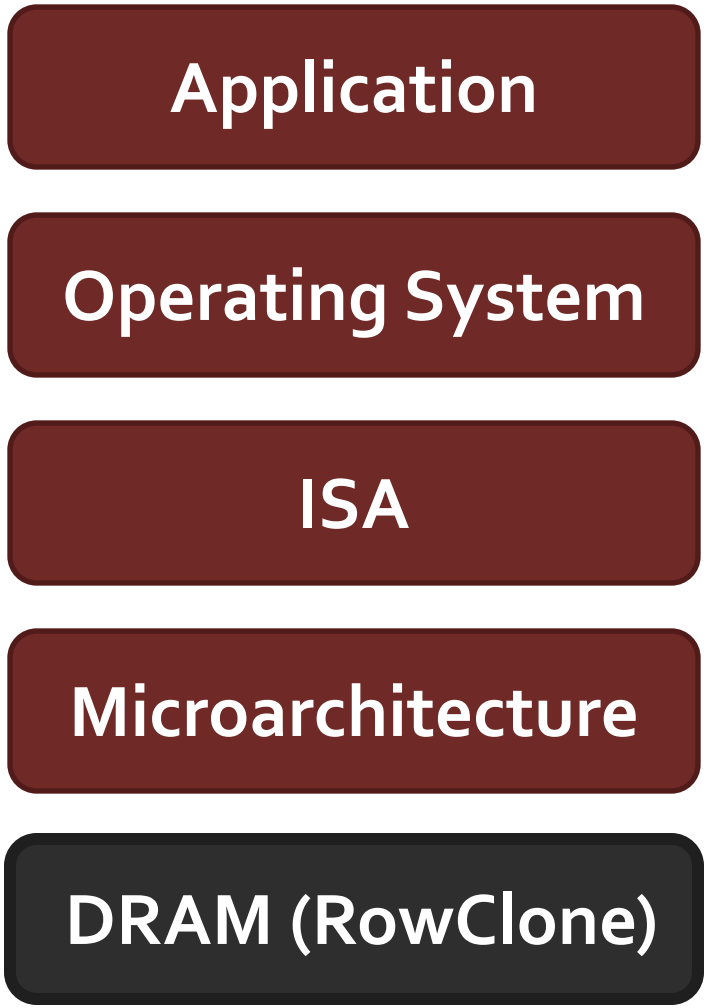


+ Very low cost: 0.01% increase in die area

Outline

- ✓ Introduction
- ✓ DRAM Background
- ✓ RowClone
  - Fast Parallel Mode
  - Pipelined Serial Mode
- End-to-end Design
- Evaluation

End-to-end System Design



How does the software communicate occurrences of bulk copy/initialization to hardware?

How to ensure cache coherence?

How to maximize use of the Fast Parallel Mode?

Handling data reuse after zero initialization?

1. Hardware/Software Interface

- Two new instructions
  - memcopy and meminit
  - Similar instructions present in existing ISAs

- Microarchitecture Implementation
  - Checks if instructions can be sped up by RowClone
  - Export instructions to the memory controller

2. Managing Cache Coherence

- RowClone modifies data in memory
  - Need to maintain coherence of cached data
- Similar to DMA
  - Source and destination in memory
  - Can leverage hardware support for DMA
- Additional optimizations
  - Create an in-cache copy of dirty source line with tag of destination line

3. Maximizing Use of Fast Parallel Mode

- Make operating system subarray-aware

- Primitives amenable to use of FPM
  - **Copy-on-Write**
    - Allocate destination in same subarray as source
    - Use FPM to copy
  - **Bulk Zeroing**
    - Use FPM to copy data from reserved zero row

4. Handling Data Reuse After Zeroing

- Data reuse after zero initialization
  - Phase 1: OS zeroes out the page
  - Phase 2: Application uses cachelines of the page
- RowClone
  - Avoids misses in phase 1
  - But incurs misses in phase 2
- **RowClone-Zero-Insert (RowClone-ZI)**
  - Insert clean zero cachelines into processor cache

Outline

- ✓ Introduction
- ✓ DRAM Background
- ✓ RowClone
  - Fast Parallel Mode
  - Pipelined Serial Mode
- ✓ End-to-end Design
- ▪ Evaluation

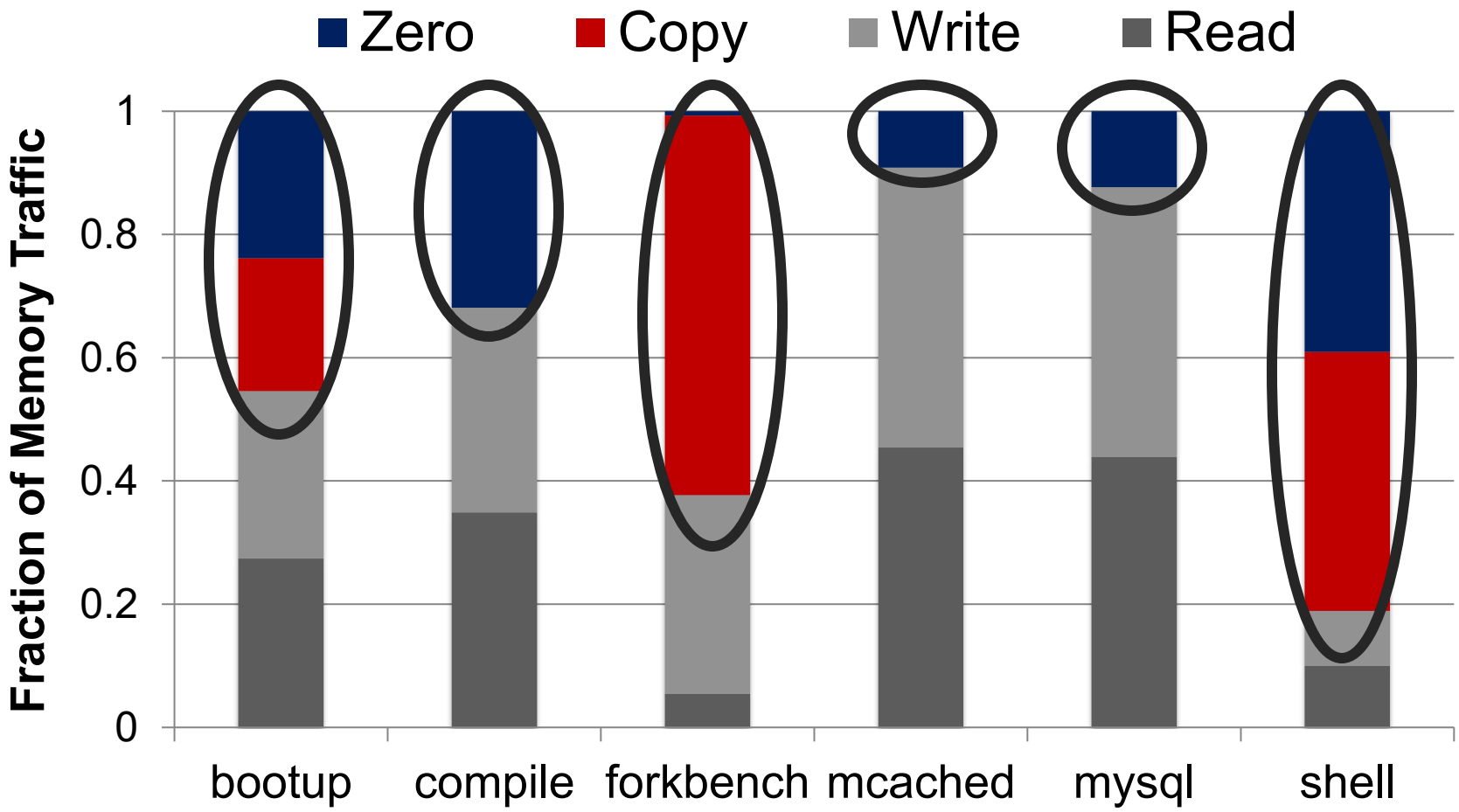
Methodology

- Out-of-order multi-core simulator
- 1MB/core last-level cache
- Cycle-accurate DDR3 DRAM simulator
- 6 Copy/Initialization intensive applications
  + SPEC CPU2006 for multi-core
- Performance
  - Instruction Throughput for single-core
  - Weighted Speedup for multi-core

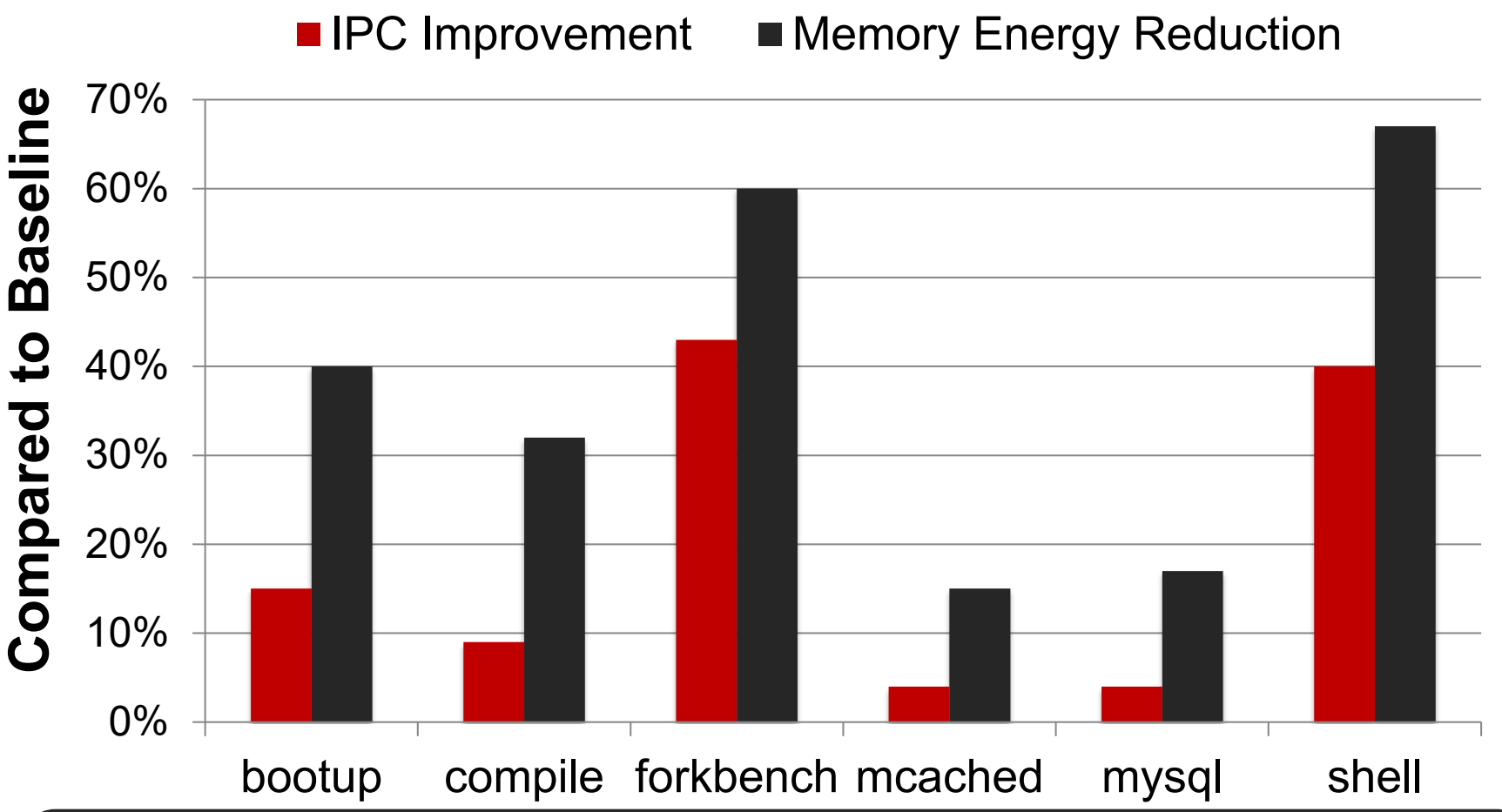
Copy/Initialization Intensive Applications

- **System bootup** (Booting the Debian OS)
- **Compile** (GNU C compiler – executing cc1)
- **Forkbench** (A fork microbenchmark)
- **Memcached** (Inserting a large number of objects)
- **MySql** (Loading a database)
- **Shell** script (`find` with `ls` on each subdirectory)

Memory Traffic due to Copy/Initialization



Single-Core Performance & Energy



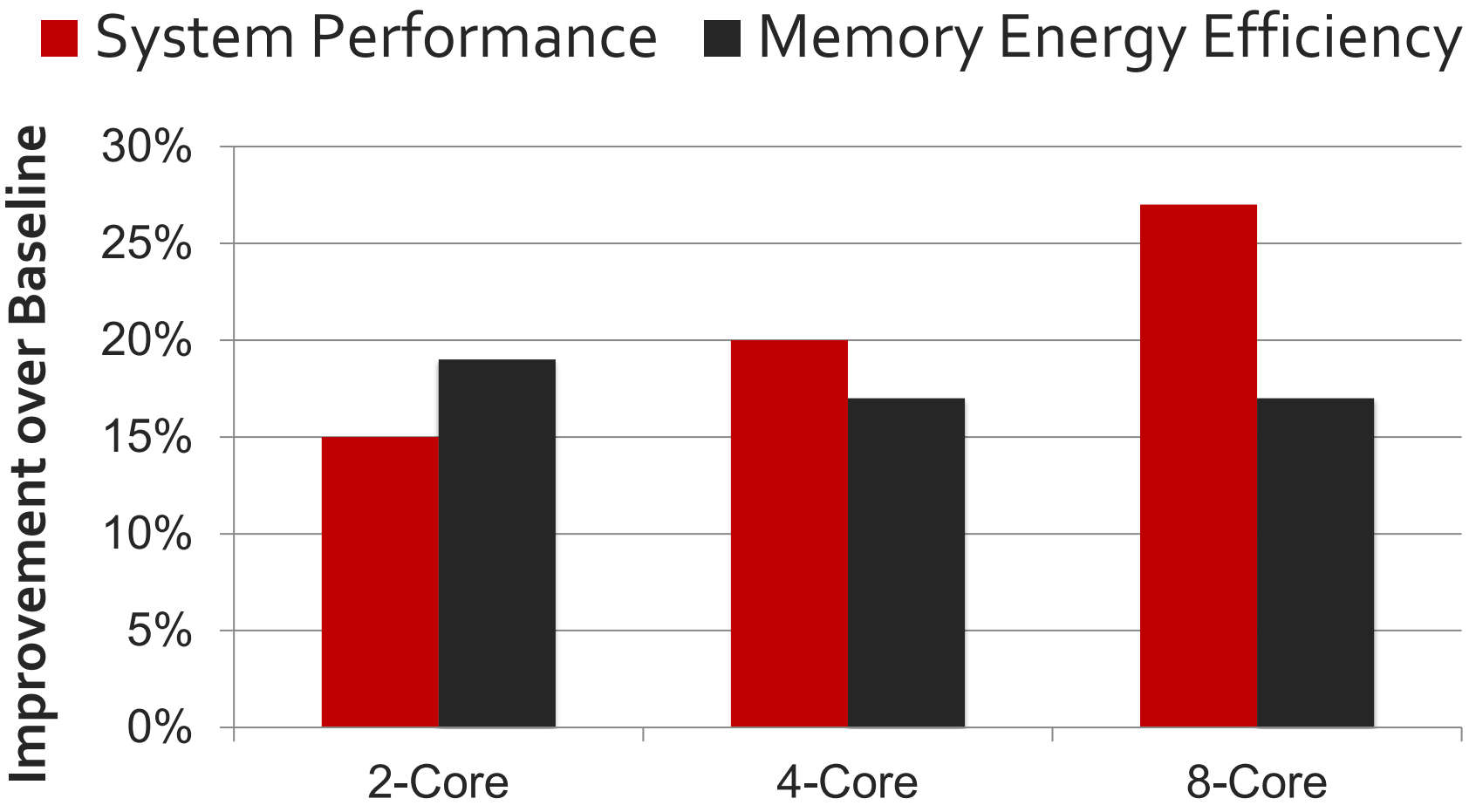
Improvements correlate with fraction of memory traffic due to copy/initialization

Multi-Core Systems

- Reduced bandwidth consumption benefits all applications.

- Run copy/initialization intensive applications with memory intensive SPEC applications.
  - Half the cores each

Multi-Core Results: Summary



Improvement increases w/ increasing core count

Consistent improvement in energy/instruction

Discussion: Summary Question #2

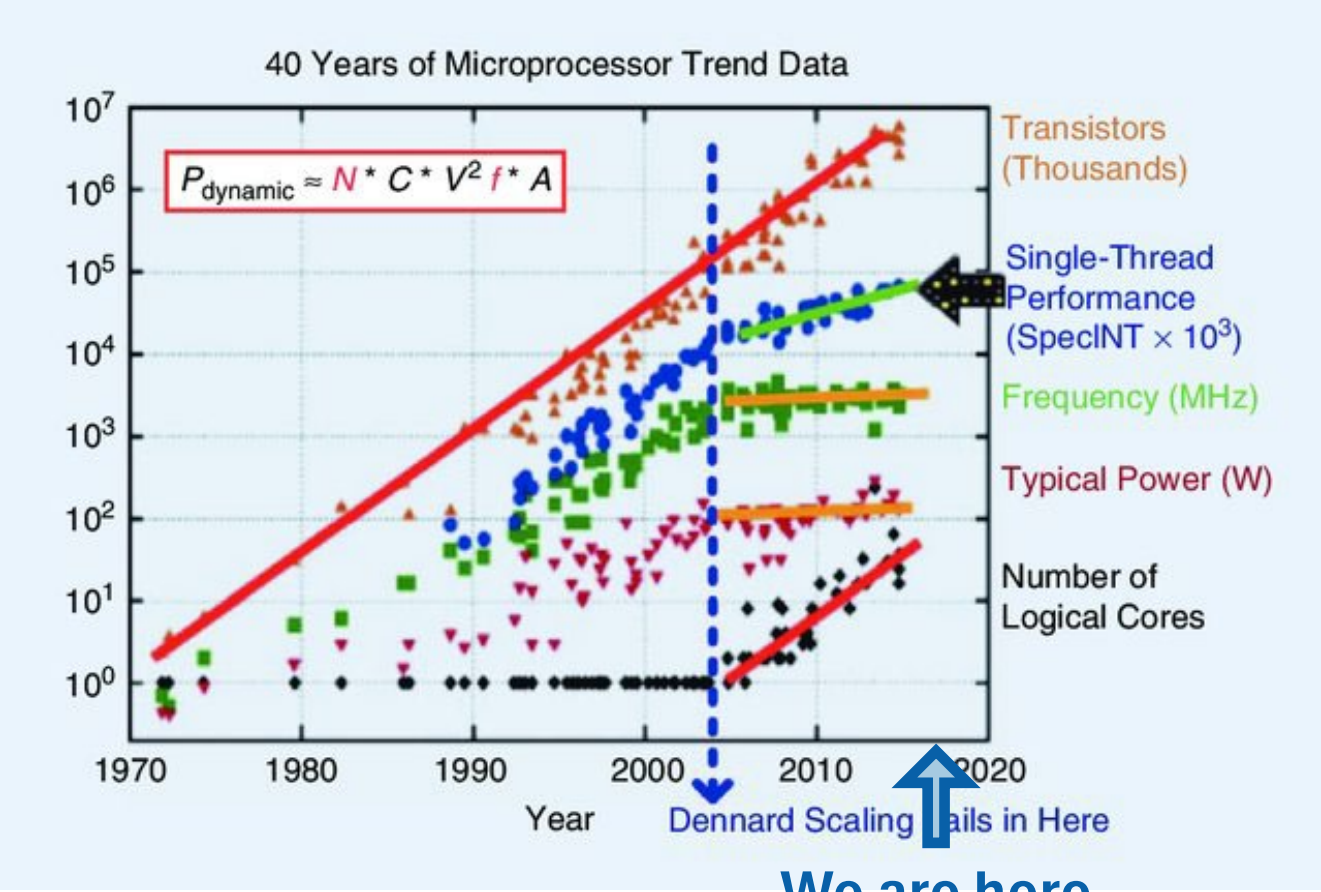
What Did the Paper Get Wrong?

Describe the paper's single most glaring deficiency.

Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

“Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology”

Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, Todd C. Mowry 2017



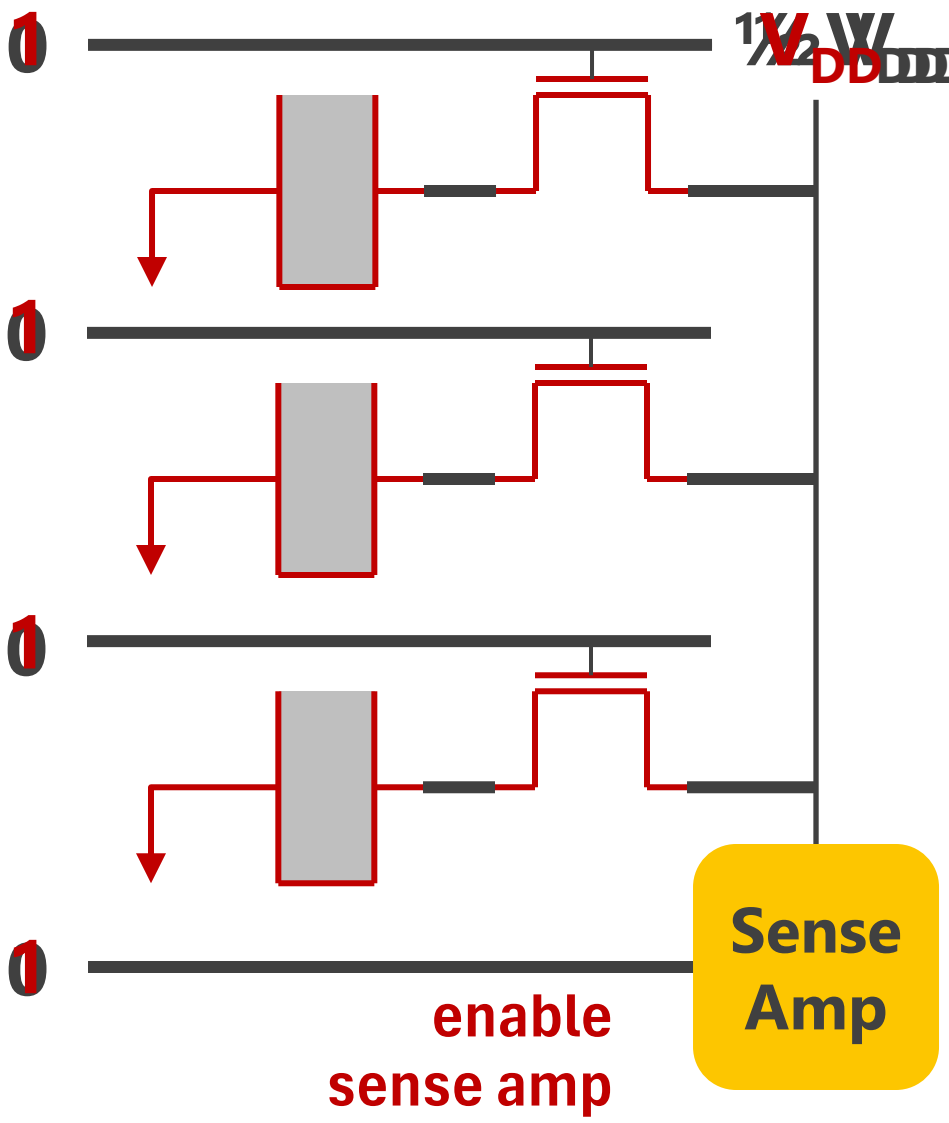
Executive Summary

- **Problem: Bulk bitwise operations**
  - present in many applications, e.g., databases, search filters
  - existing systems are memory bandwidth limited
- **Our Proposal: Ambit**
  - perform bulk bitwise operations completely inside DRAM
  - bulk bitwise AND/OR: simultaneous activation of three rows
  - bulk bitwise NOT: inverters in sense amplifiers
- **Results compared to state-of-the-art baseline**
  - average across seven bulk bitwise operations
    - 32X performance improvement (over GPU), 35X energy reduction
  - 3X-7X performance for real-world data-intensive applications

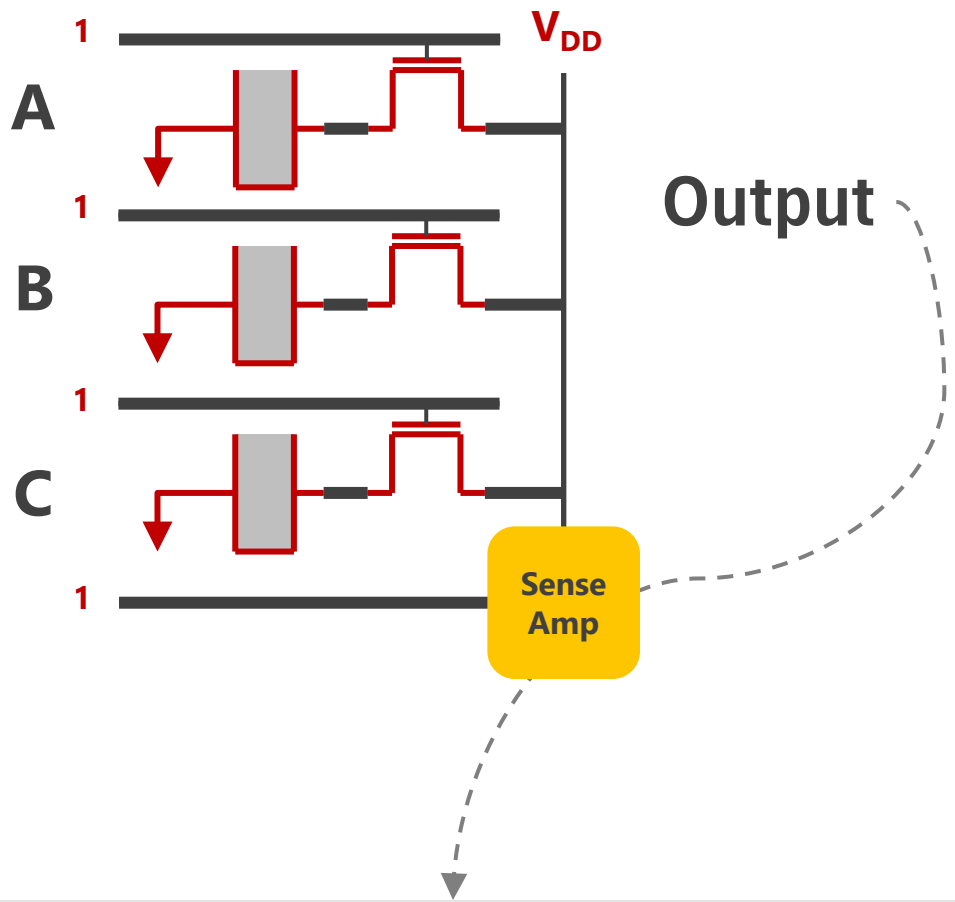
Slides adapted from our Micro'17 talk

Triple-Row Activation: Majority Function

activate all three rows



Bitwise AND/OR Using Triple-Row Activation




```
Output = AB + BC + CA
       = C (A OR B) + ~C (A AND B)
```

Potential Concerns with Triple-Row Activation

1. With three cells, bitline deviation may not be enough
2. Process variation: all cells are not equal

Spice simulations put these concerns to rest.

3. Cells leak charge
4. Memory controller may have to send three addresses
5. Source data gets destroyed

Address these challenges through implementation (next slide)

Bulk Bitwise AND/OR in DRAM

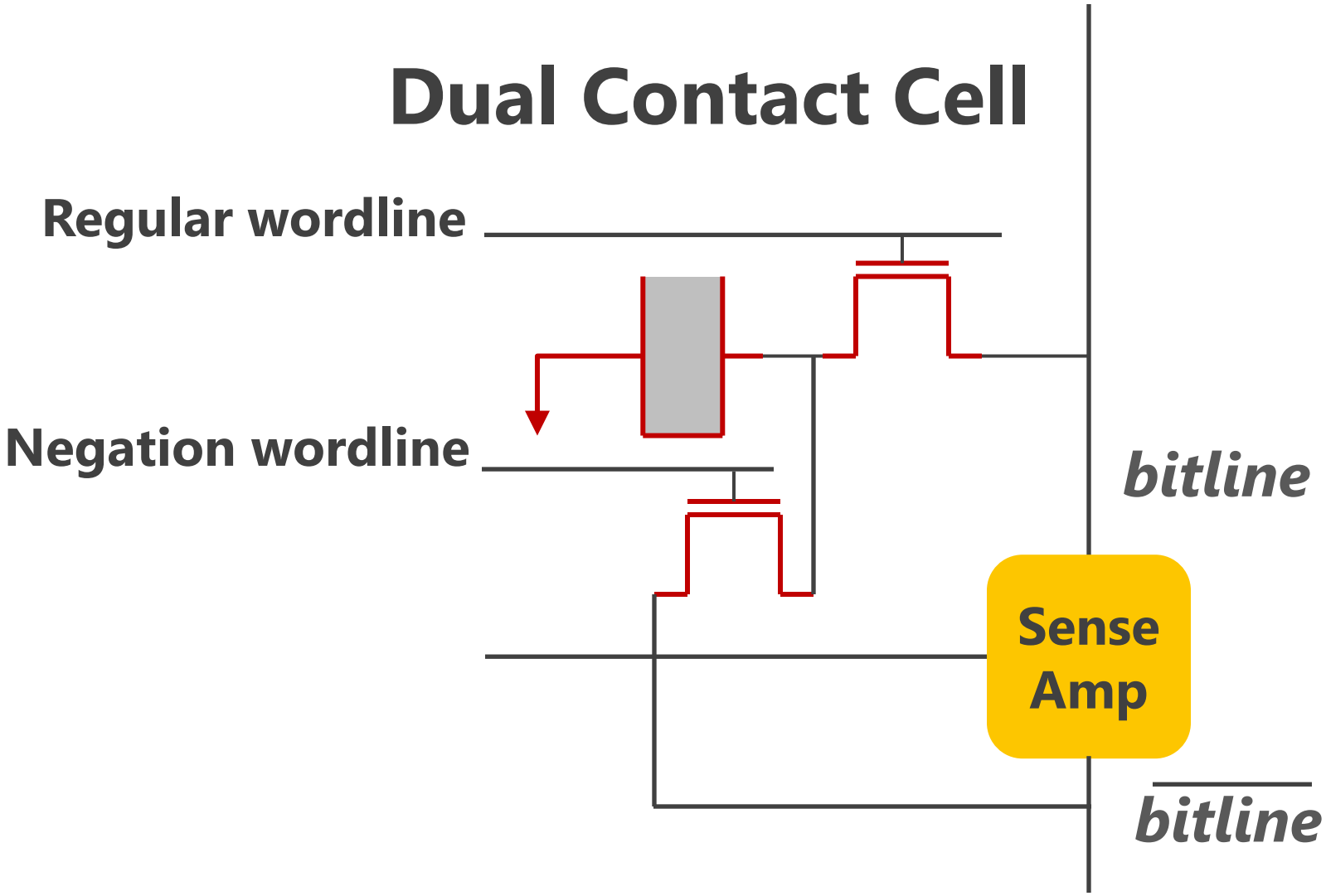
Result = row A **AND/OR** row B

Statically reserve three designated rows **t1**, **t2**, and **t3**

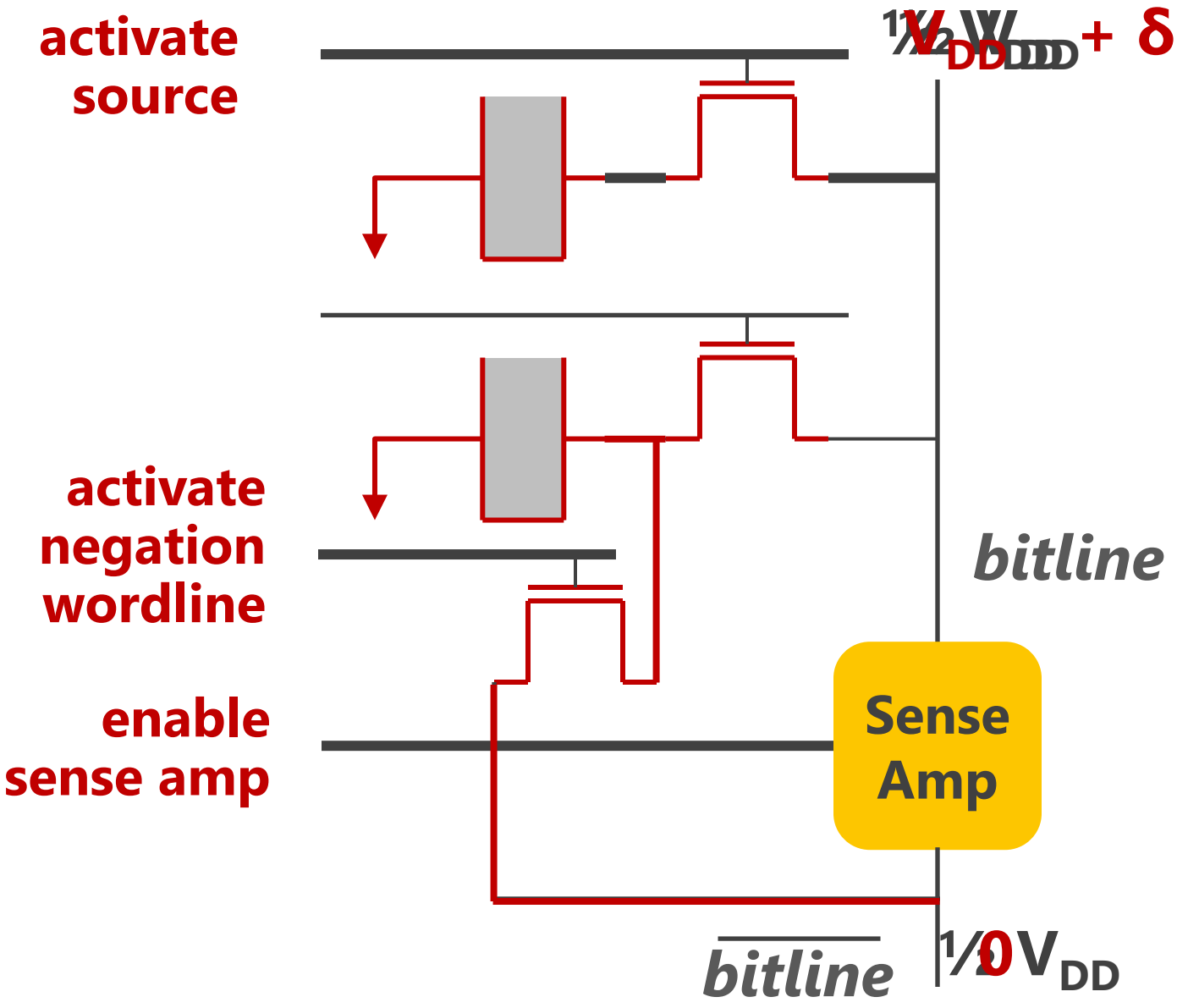
1. **Copy** data of row **A** to row **t1**
2. **Copy** data of row **B** to row **t2**
3. **Initialize** data of row **t3** to **0/1**
4. **Activate** rows **t1/t2/t3** simultaneously
5. **Copy** data of row **t1/t2/t3** to **Result** row

Use **RowClone** to perform **copy** and **initialization** operations completely in DRAM!

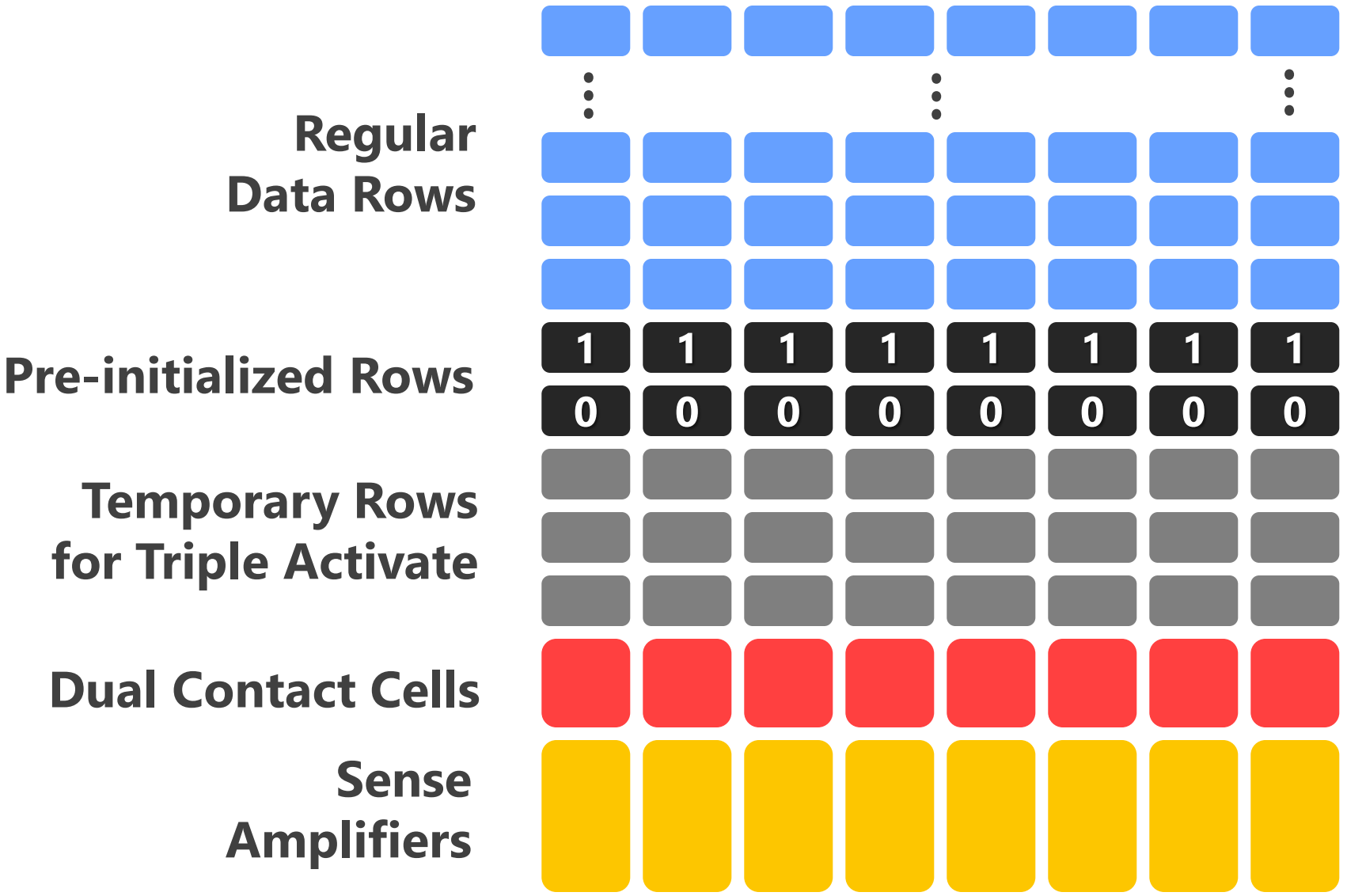
Negation Using the Sense Amplifier



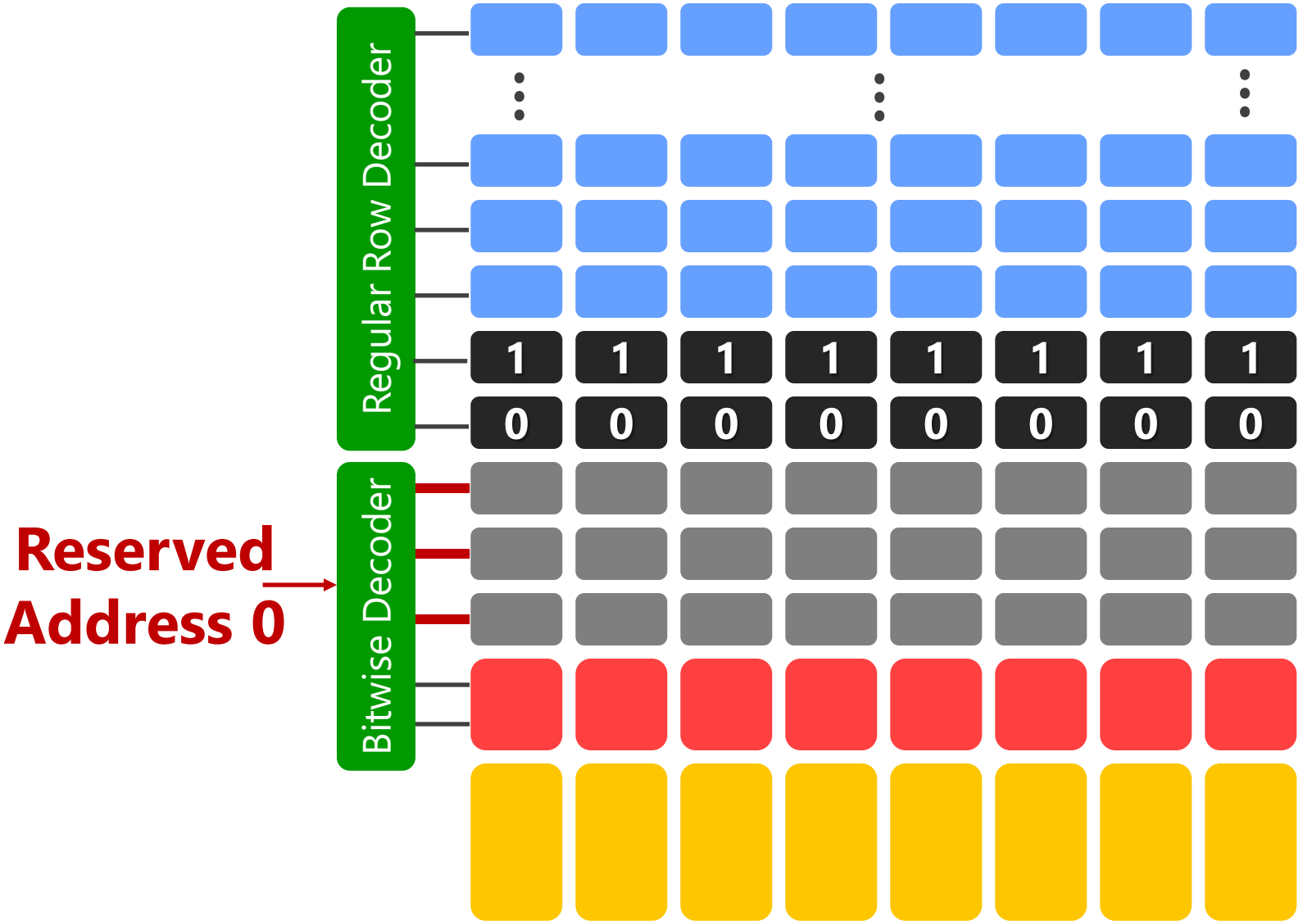
Negation Using the Sense Amplifier



Ambit – Implementation

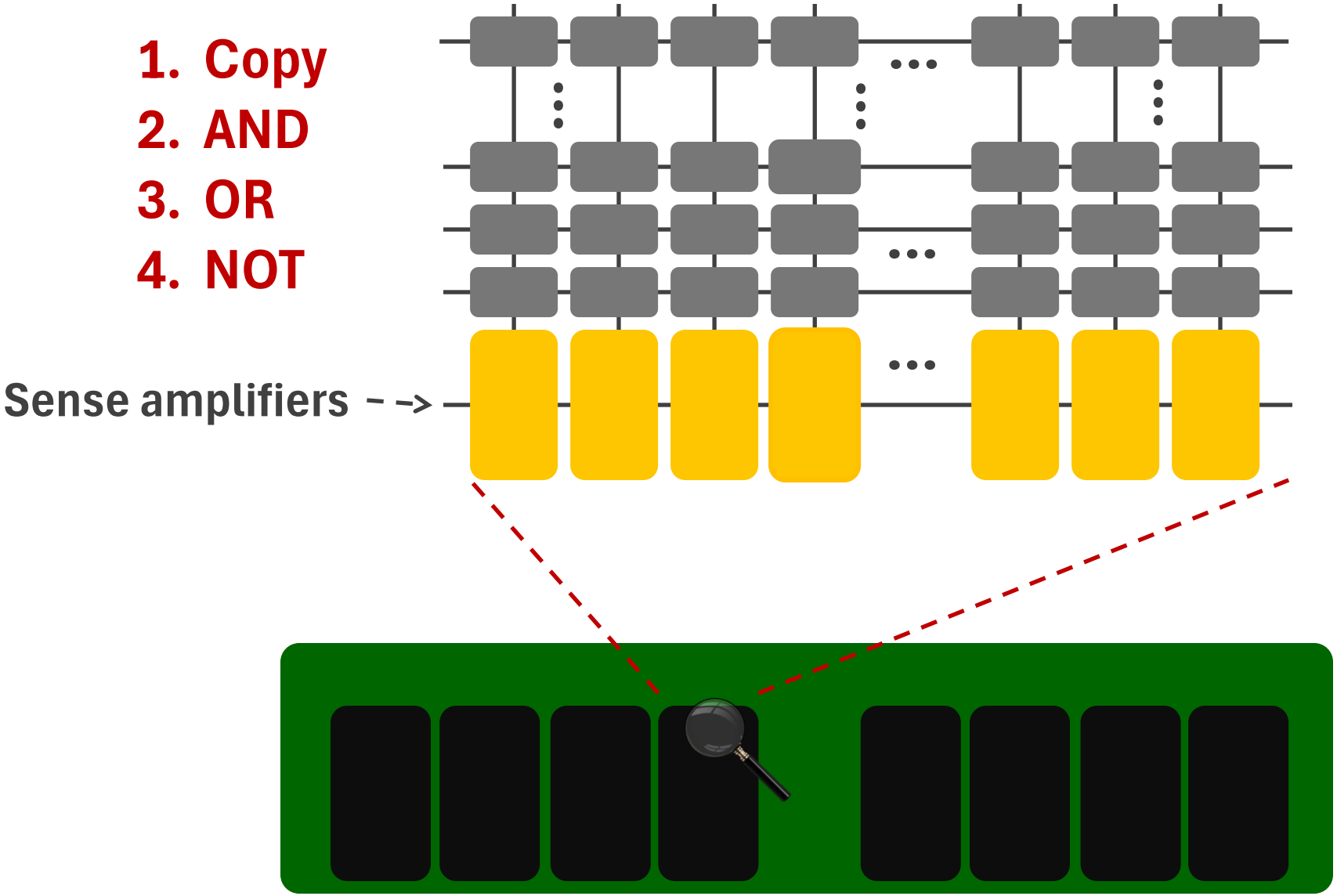


Ambit – Implementation



Ambit: Summary of Bulk Bitwise Ops

1. Copy
2. AND
3. OR
4. NOT



To Read for Wednesday

"Livia: Data-Centric Computing Throughout the Memory Hierarchy"

Elliot Lockerman, Axel Feldmann, Mohammad Bakhshalipour, Alexandru Stanescu, Shashwat Gupta, Daniel Sanchez, Nathan Beckmann 2020

Optional Further Reading:

"Leviathan: A Unified System for General-Purpose Near-Data Computing"

Brian C. Schwedock, Nathan Beckmann 2024