Simple DRAM and Virtual Memory Abstractions for Highly Efficient Memory Systems

Thesis Oral

Vivek Seshadri

Committee:

Todd Mowry (Co-chair)

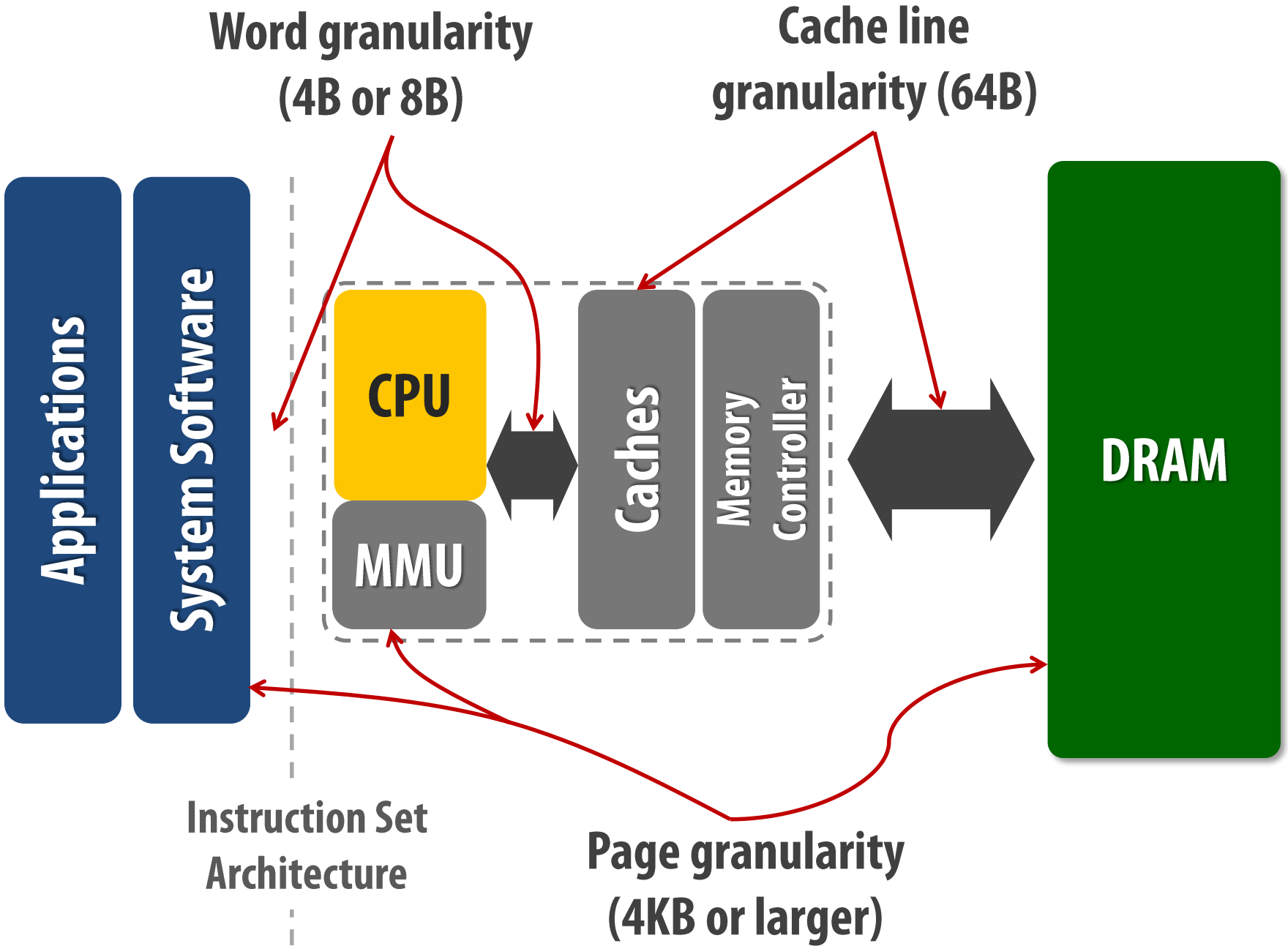
Onur Mutlu (Co-chair)

Phillip B. Gibbons

David Andersen

Rajeev Balasubramonian, University of Utah

Presented in partial fulfillment of the requirements for the degree of Doctor of Philosophy


Word granularity
(4B or 8B)
Cache line
granularity (64B)
Applications
System Software
CPU
MMU
Caches
Memory
Controller
DRAM
Instruction Set
Architecture
Page granularity
(4KB or larger)

The Curse of Multiple Granularities

- **Fine-grained memory capacity management**
  - Existing virtual memory frameworks result in unnecessary work
  - High memory redundancy and low performance
- **Bulk data operations**
  - Existing interfaces require large data transfers
  - High latency, bandwidth, and energy
- **Non-unit strided access patterns**
  - Poor spatial locality
  - Existing systems optimized to transfer cache lines
  - High latency, bandwidth, and energy

Thesis Statement

Our thesis is that

exploiting the untapped potential of existing hardware structures (processor and DRAM) by augmenting them with simple, low-cost features can enable significant improvement in the efficiency of different memory operations

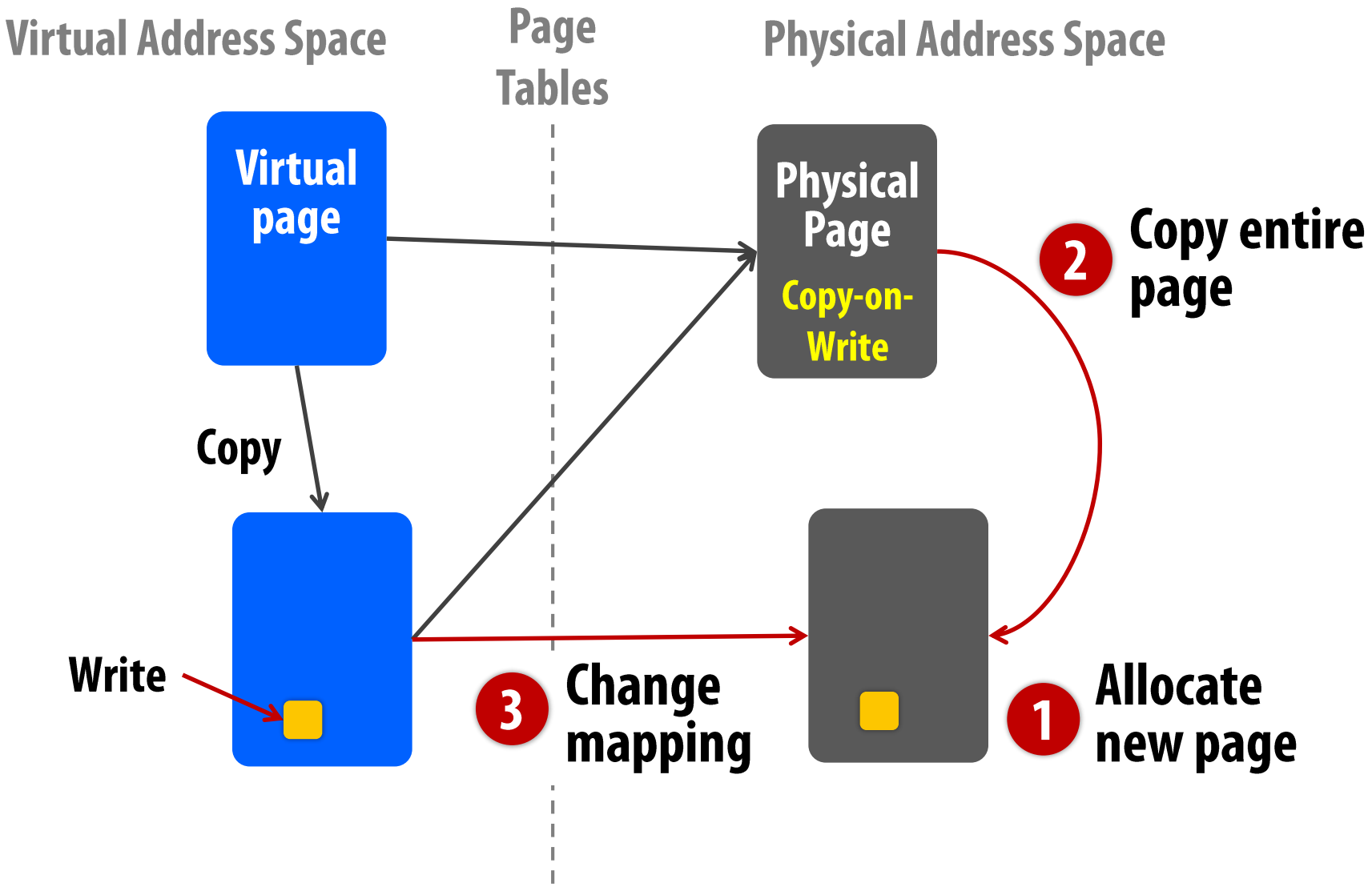
Contributions of this Dissertation

1. **Page Overlays** – a new virtual memory framework to enable fine-grained memory management
2. **RowClone** – a mechanism to perform bulk data copy and initialization completely inside DRAM
3. **Buddy RAM** – a mechanism to perform bulk bitwise operations completely inside DRAM
4. **Gather-Scatter DRAM** – a mechanism to exploit DRAM architecture to accelerate strided access patterns
5. **Dirty-Block Index** – a mechanism to restructure dirty bits in on-chip caches to suit queries for dirty blocks

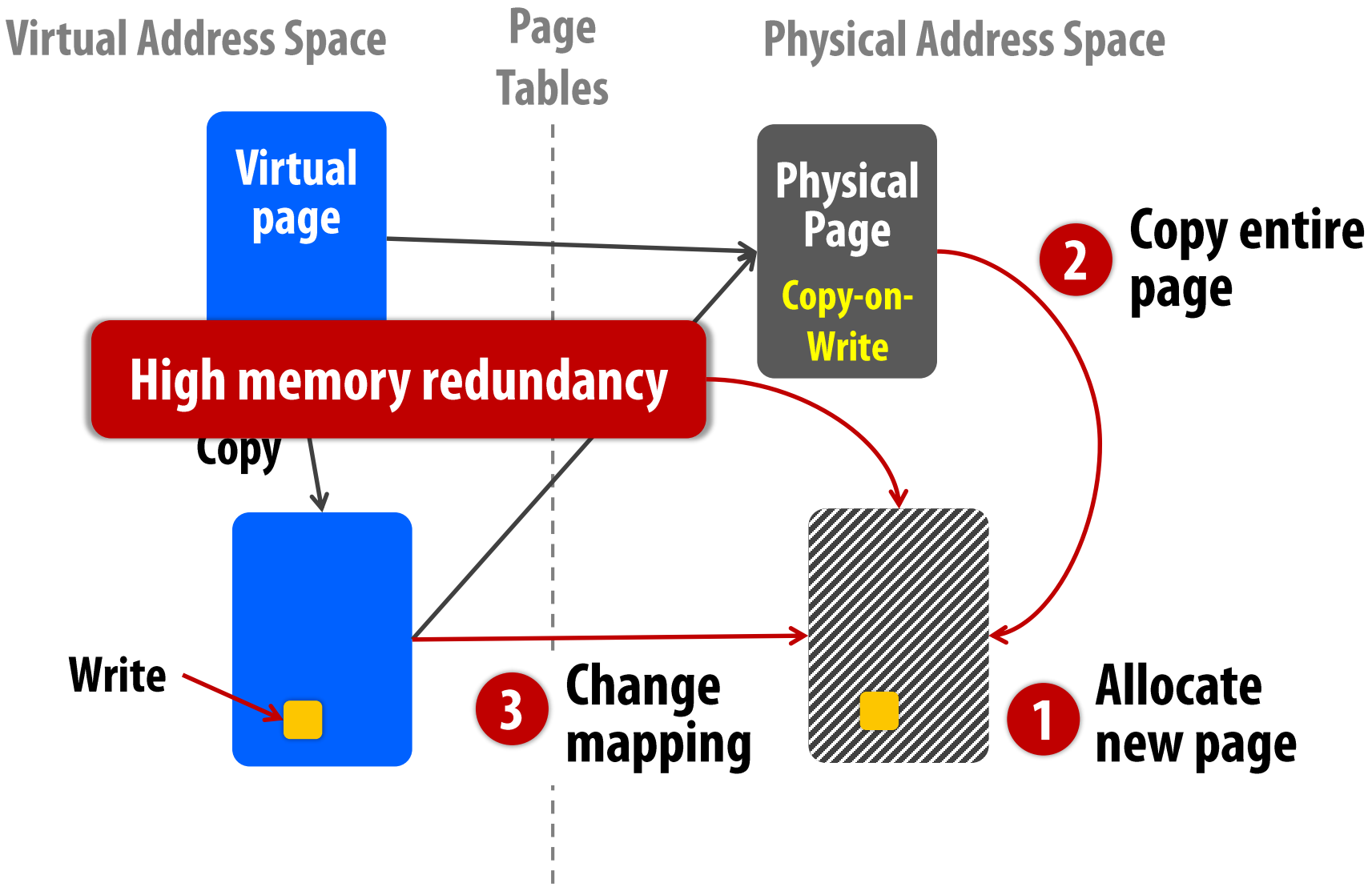
Outline for the Talk

1. Page Overlays
   - efficient fine-grained memory management
2. Gather-Scatter DRAM
   - accelerating strided access patterns
3. RowClone + Buddy RAM
   - in-DRAM bulk copy + bitwise operations
4. Dirty-Block Index
   - maintaining coherence of dirty blocks

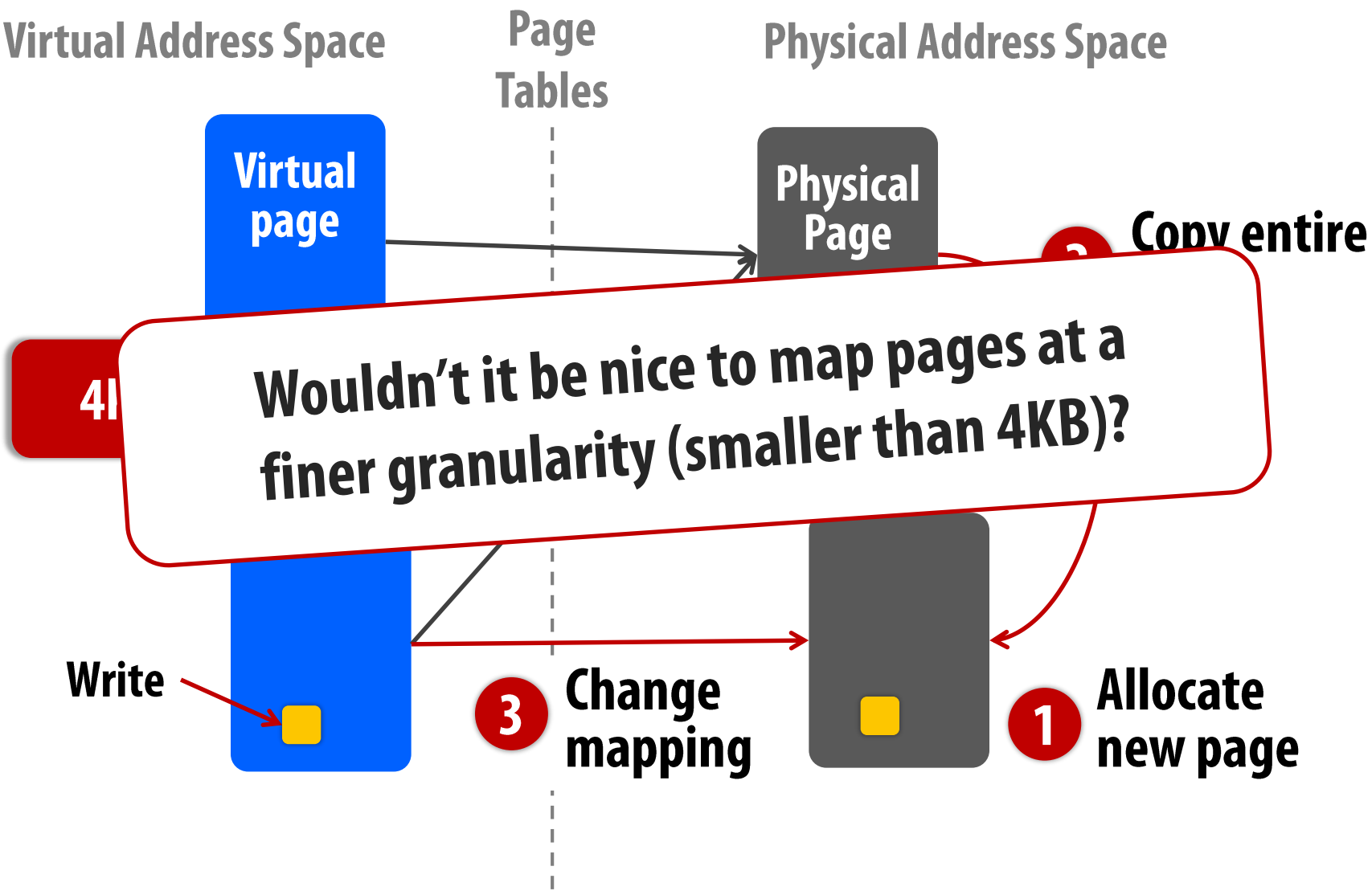
Copy-on-Write



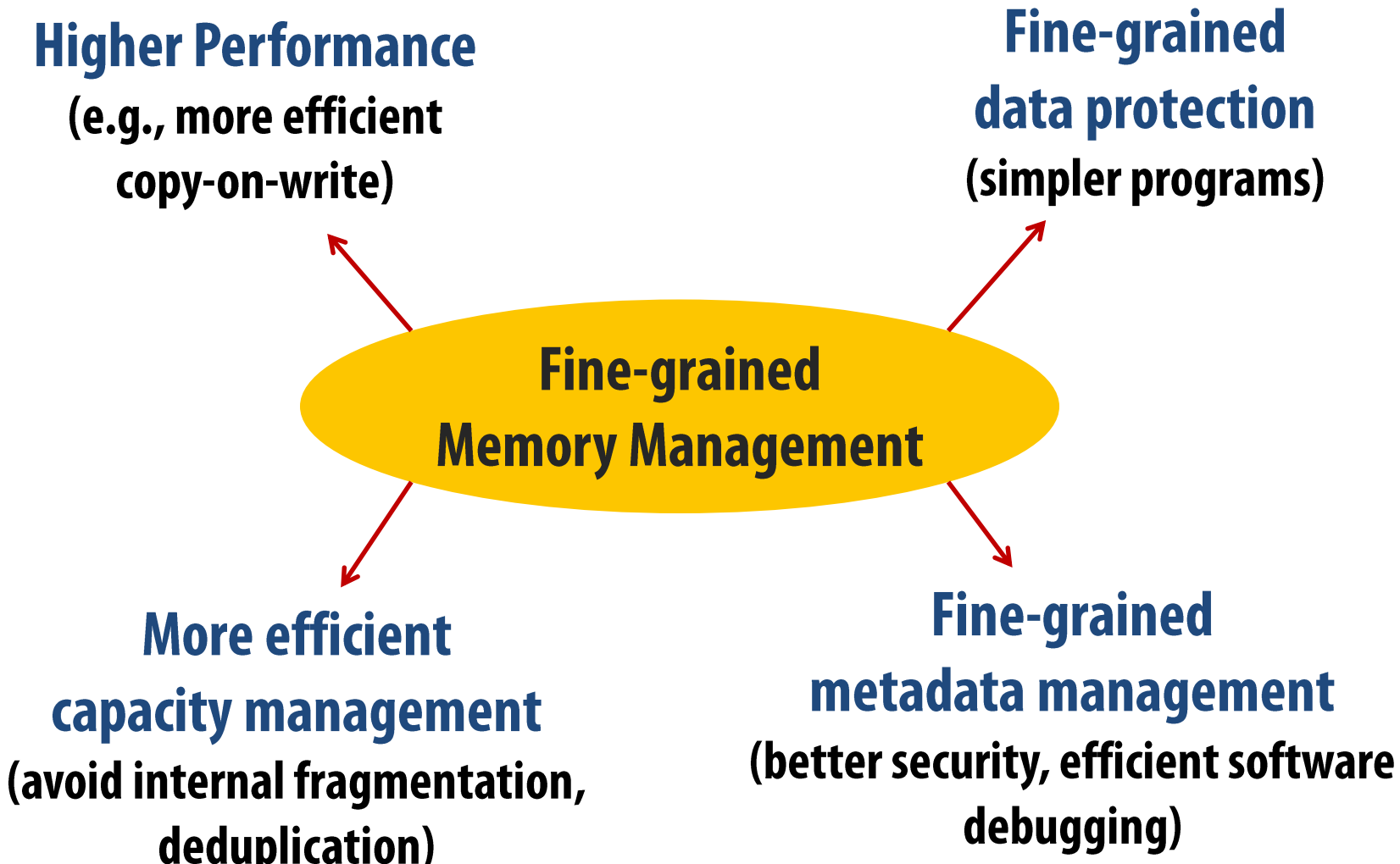
Shortcomings of Page-granularity Management



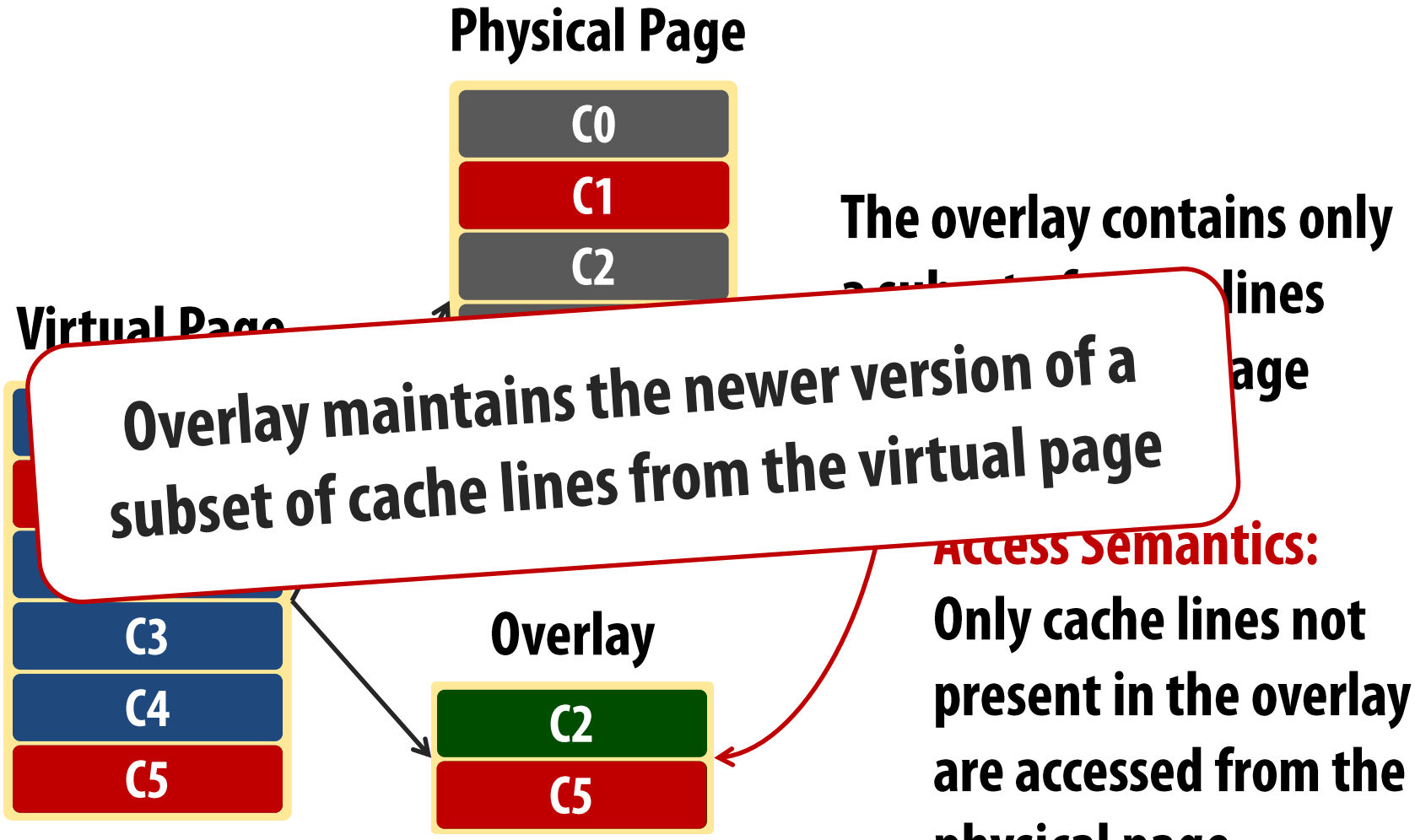
Shortcomings of Page-granularity Management



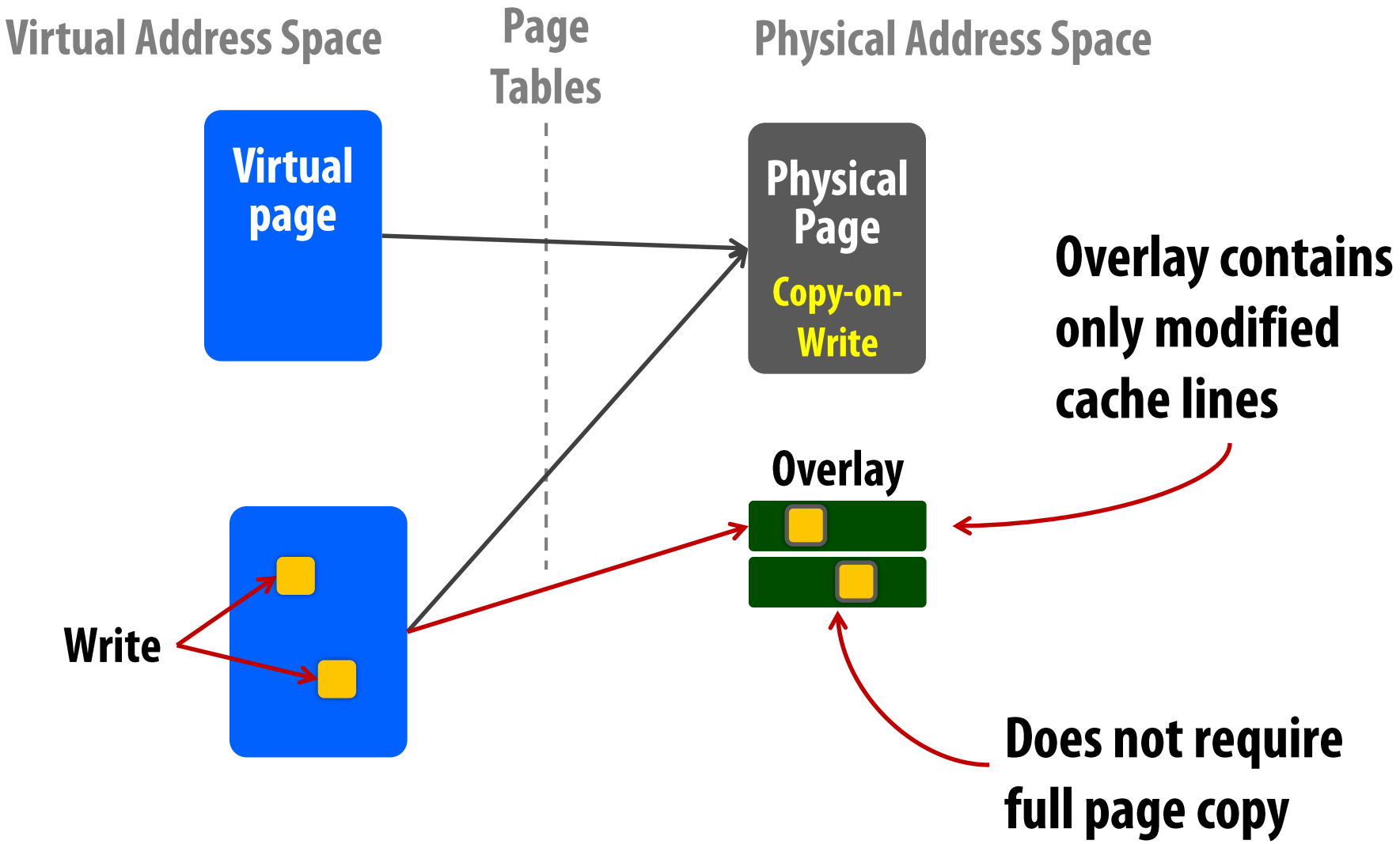
Fine-grained Memory Management



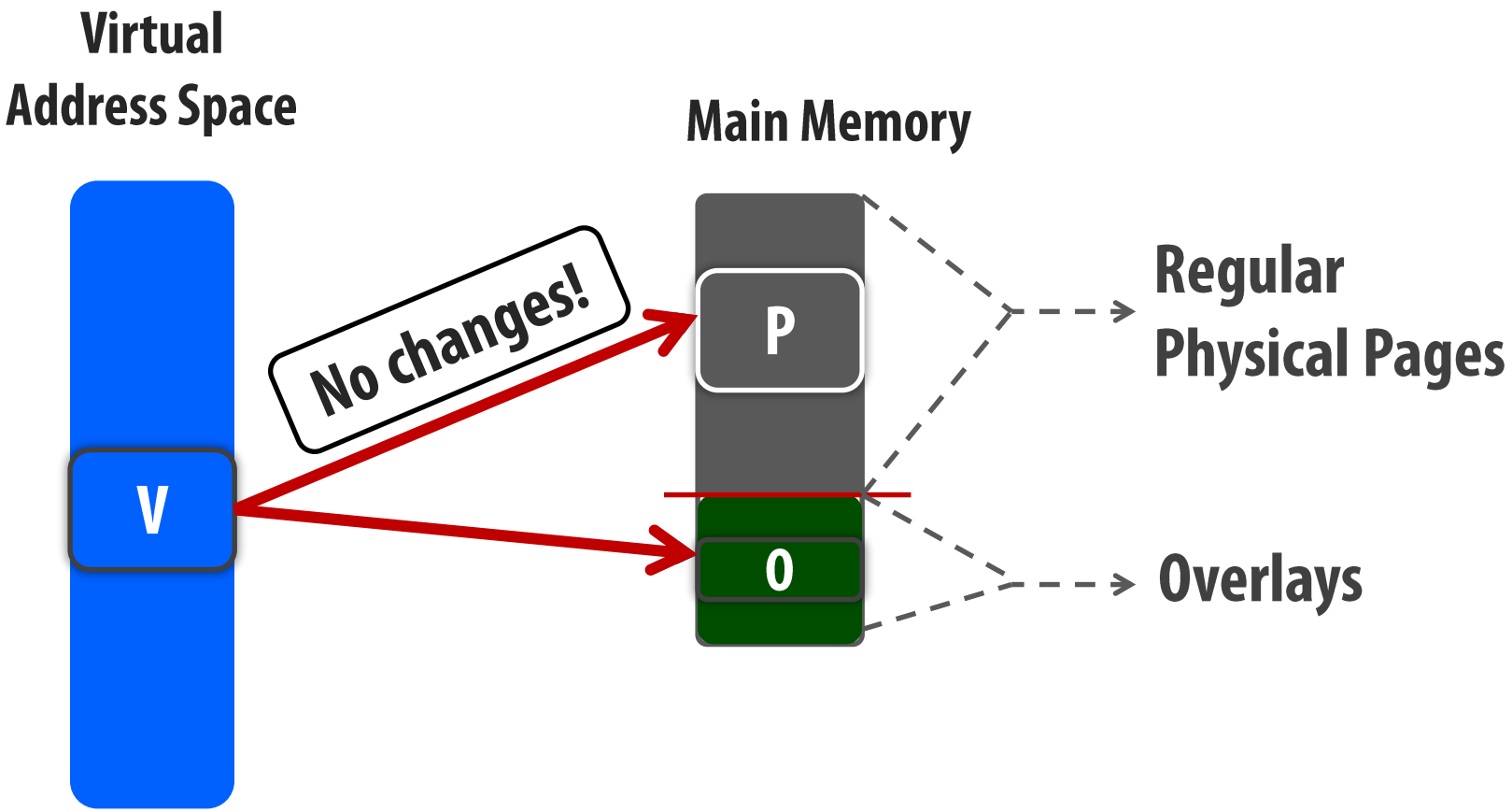
The Page Overlay Framework



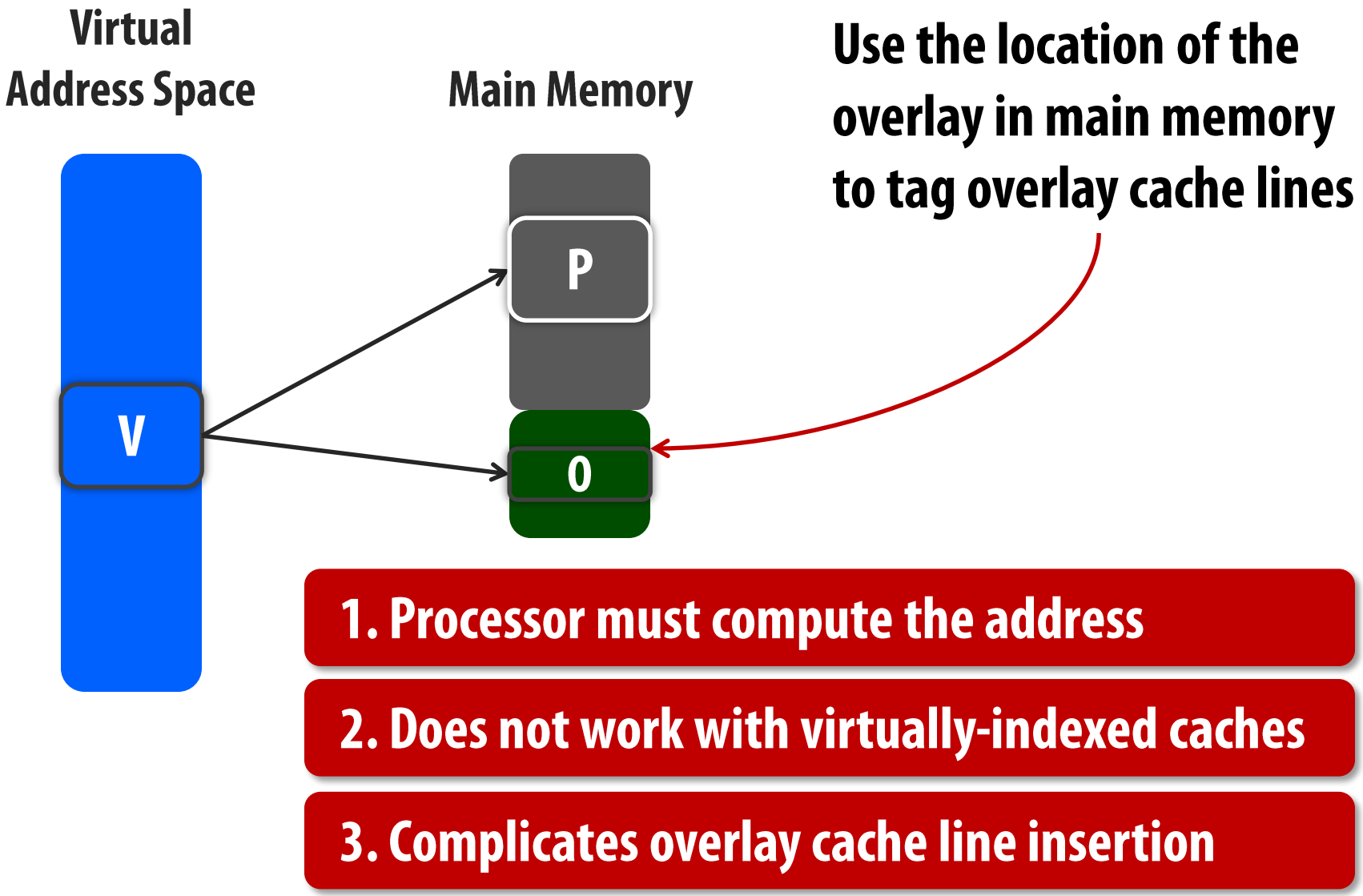
Overlay-on-Write: An Efficient Copy-on-Write



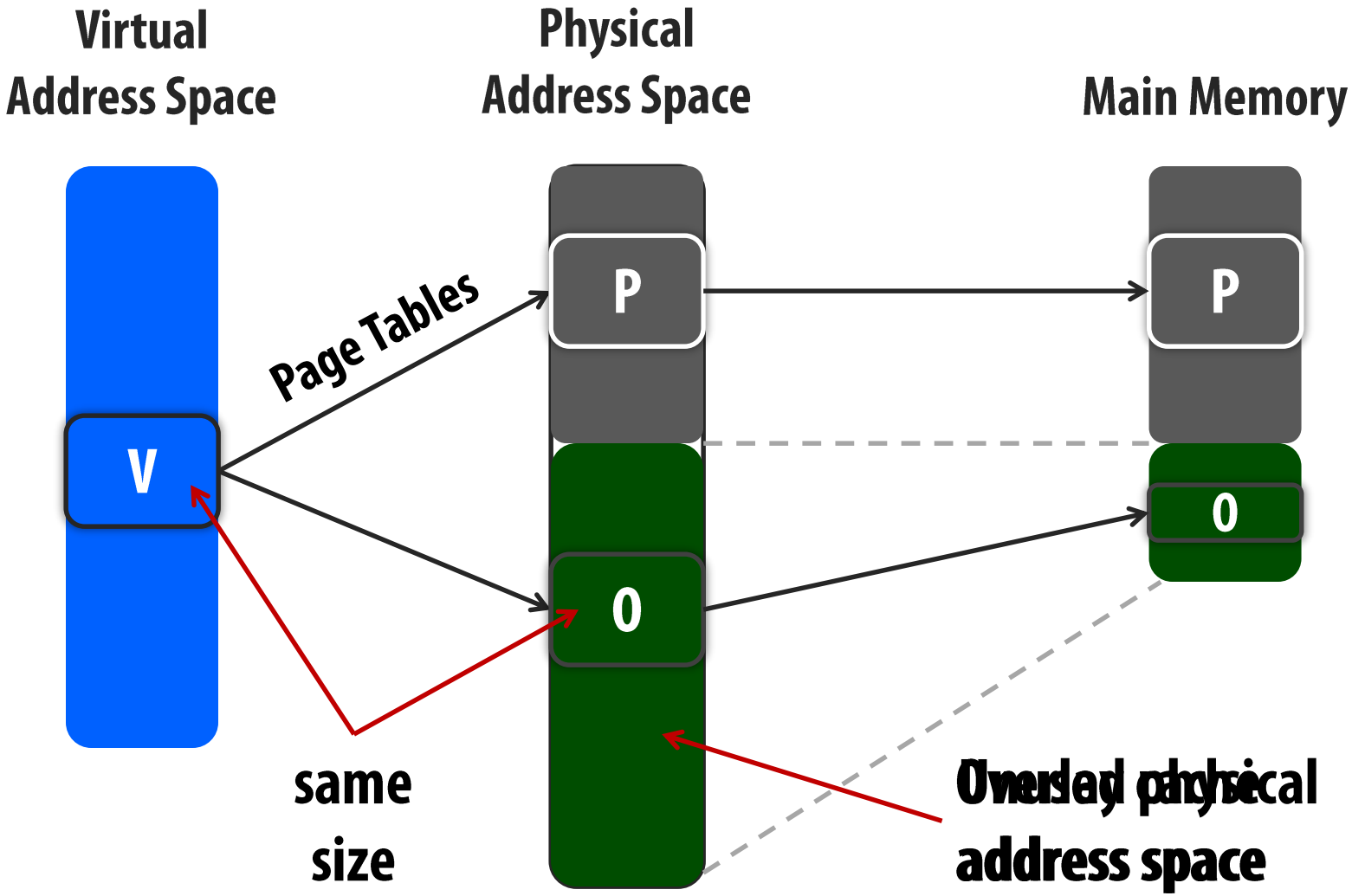
Implementation Overview



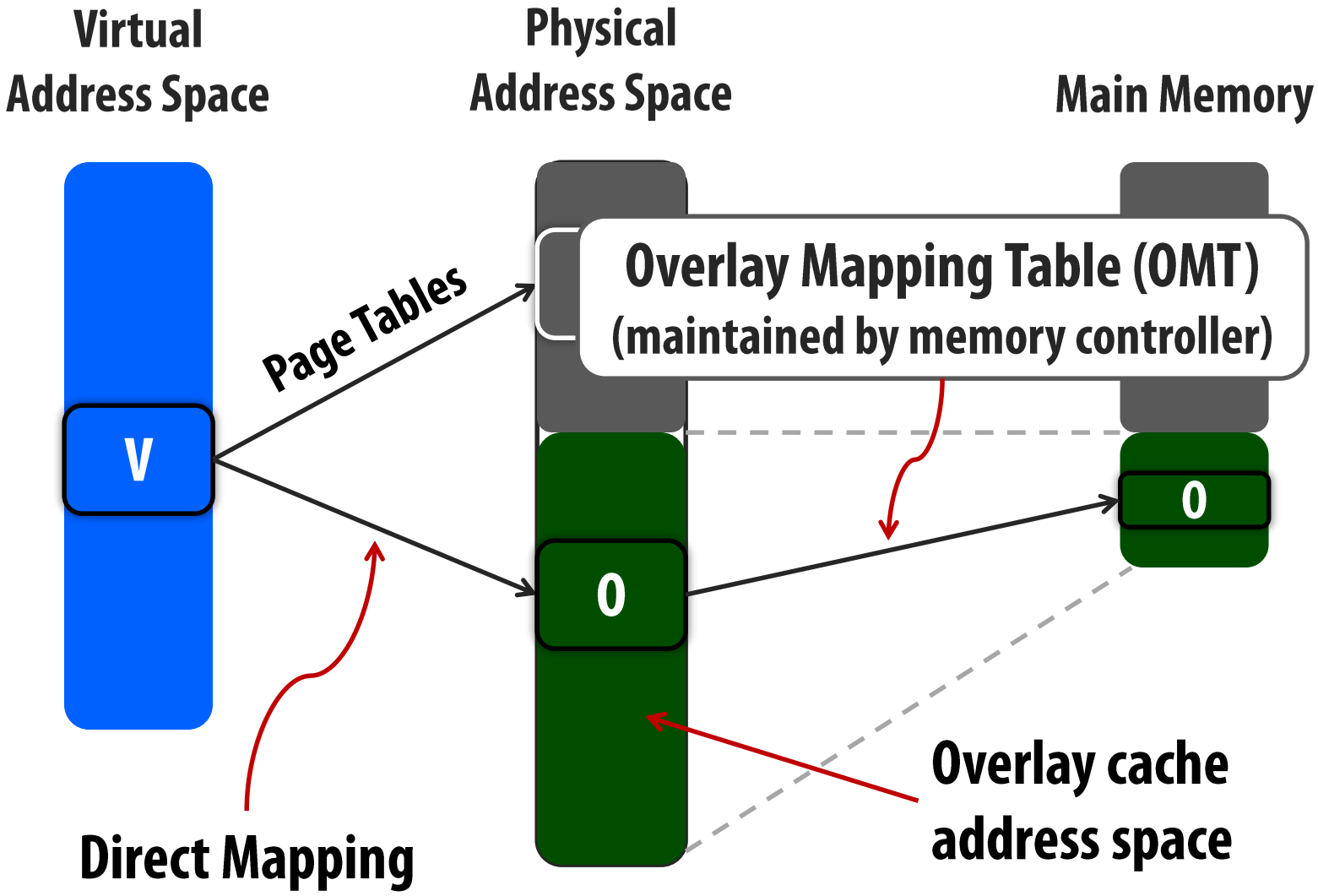
Addressing Overlay Cache Lines: Naïve Approach



Addressing Overlay Cache Lines: Dual Address Design



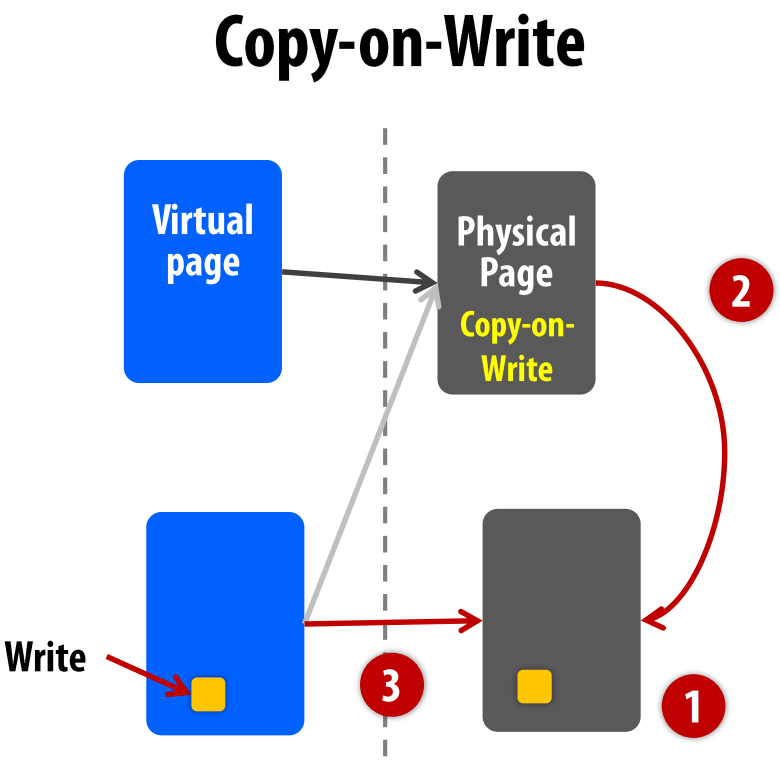
Virtual-to-Overlay Mappings

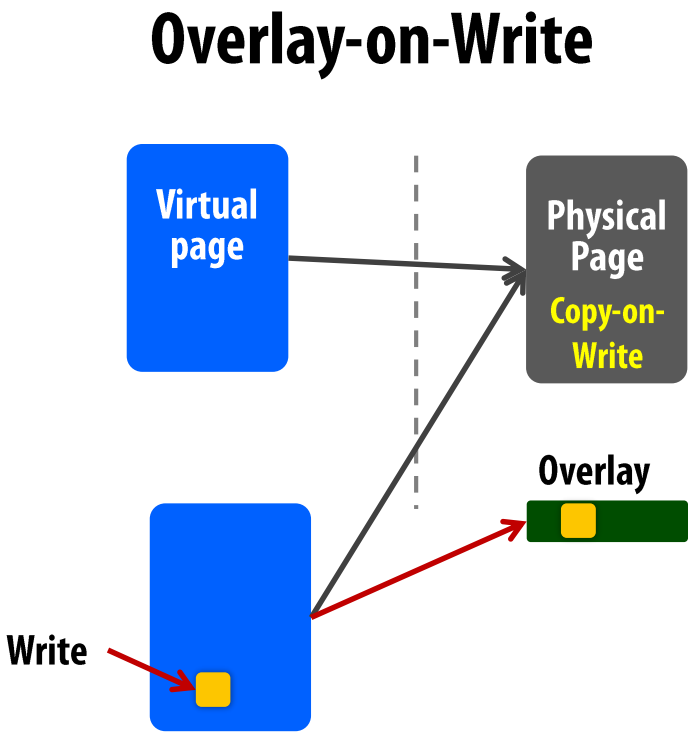


Methodology

- Memsim memory system simulator [Seshadri+ PACT 2012]
- 2.67 GHz, single core, out-of-order, 64 entry instruction window
- **64-entry L1 TLB, 1024-entry L2 TLB**
- **64KB L1 cache, 512KB L2 cache, 2MB L3 cache**
- Multi-entry Stream Prefetcher [Srinath+ HPCA 2007]
- Open row, FR-FCFS, 64 entry write buffer, drain when full
- **64-entry OMT cache**
- DDR3 1066 MHz, 1 channel, 1 rank, 8 banks

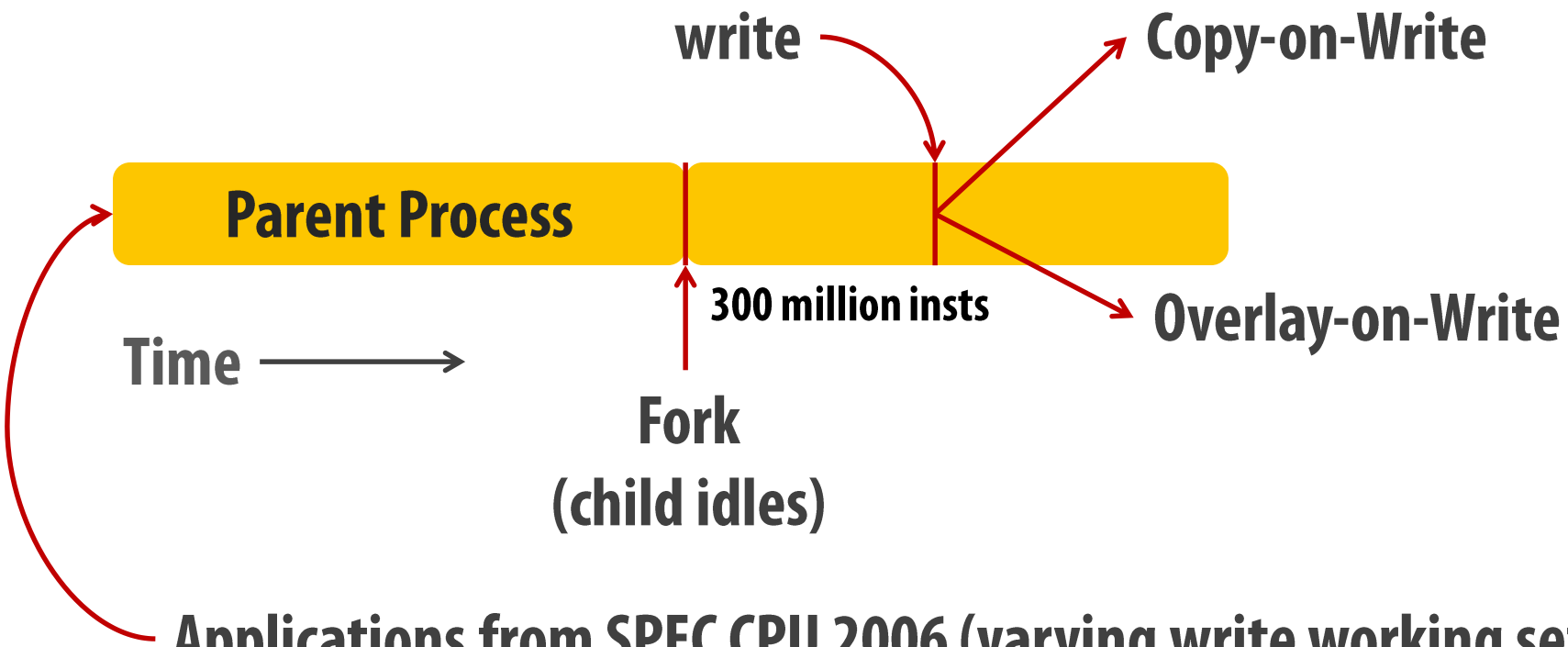
Overlay-on-Write





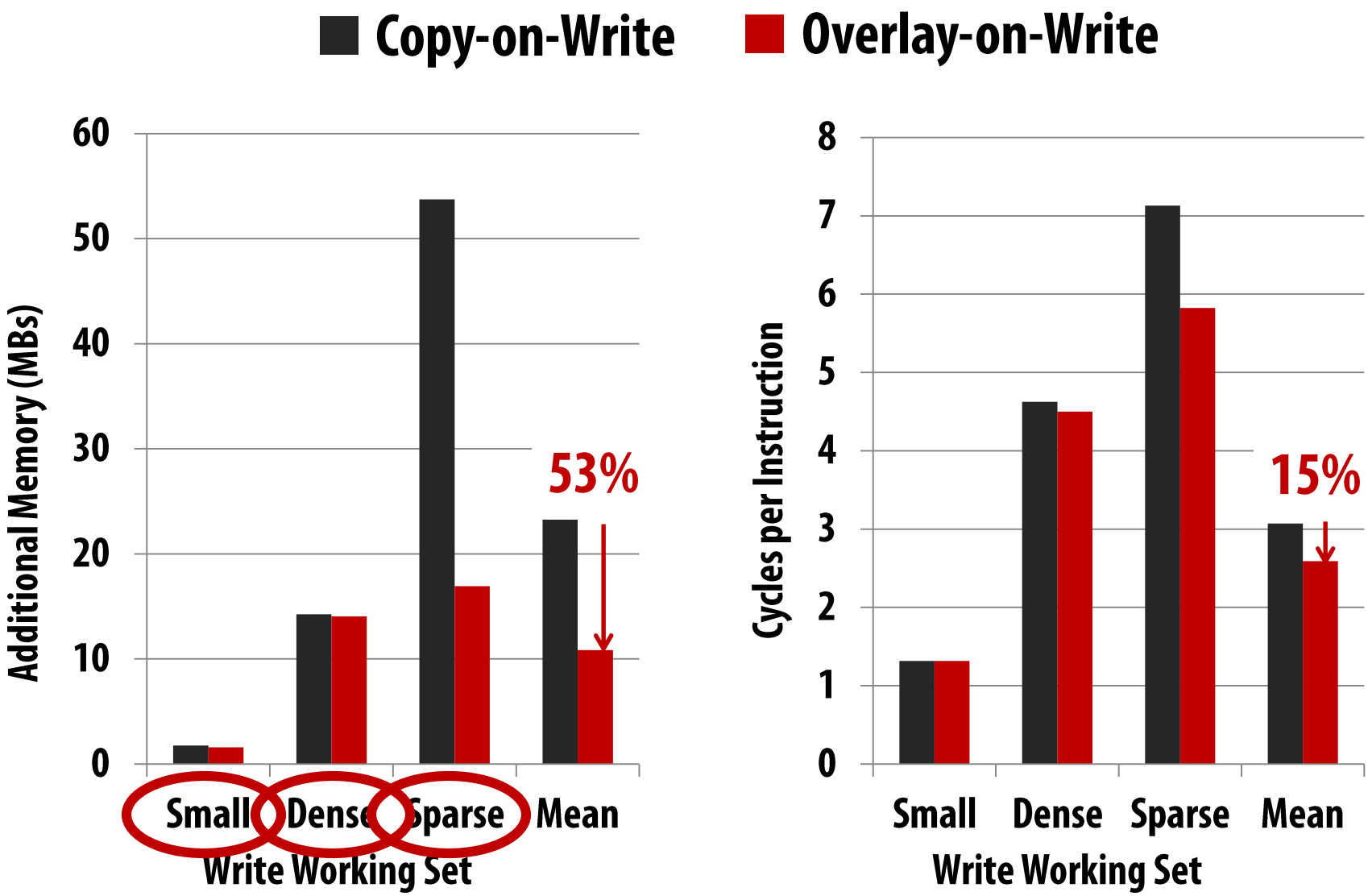
- Lower memory redundancy
- Lower latency

Fork Benchmark



- Additional memory consumption
- Performance (cycles per instruction)

Overlay-on-Write vs. Copy-on-Write on Fork



Page Overlays: Applications

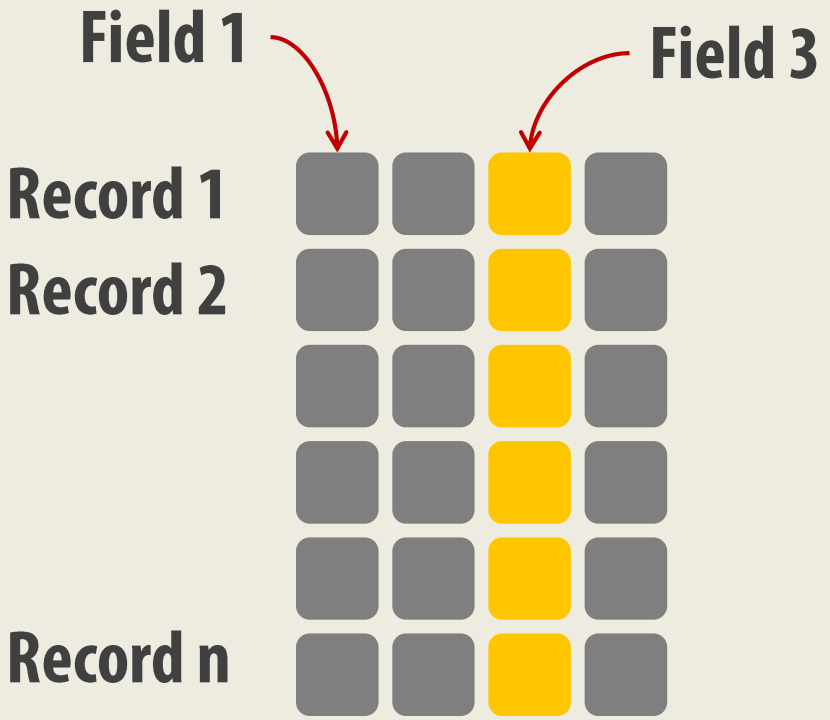
- Overlay-on-Write
- Sparse Data Structure Representation
  - Exploit any degree of cache line sparsity
  - Allow dynamic insertion of non-zero values
- Flexible Super-page Management
  - Share super pages across processes
- Fine-grained Deduplication
- Memory Checkpointing
- Virtualizing Speculation
- Fine-grained Metadata Management

Outline for the Talk

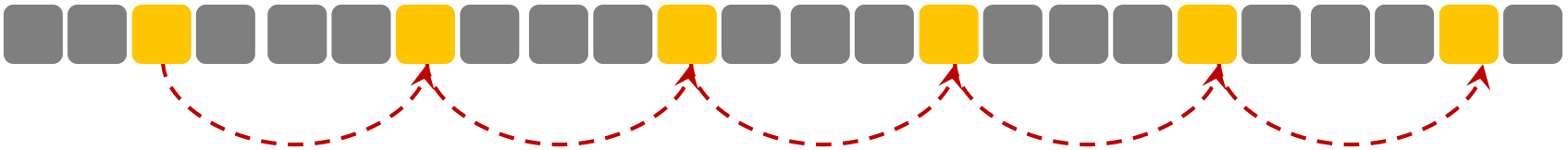
1. Page Overlays
   - efficient fine-grained memory management
2. Gather-Scatter DRAM
   - accelerating strided access patterns
3. RowClone + Buddy RAM
   - in-DRAM bulk copy + bitwise operations
4. Dirty-Block Index
   - maintaining coherence of dirty blocks

Example 2: Strided access pattern

In-Memory Database Table



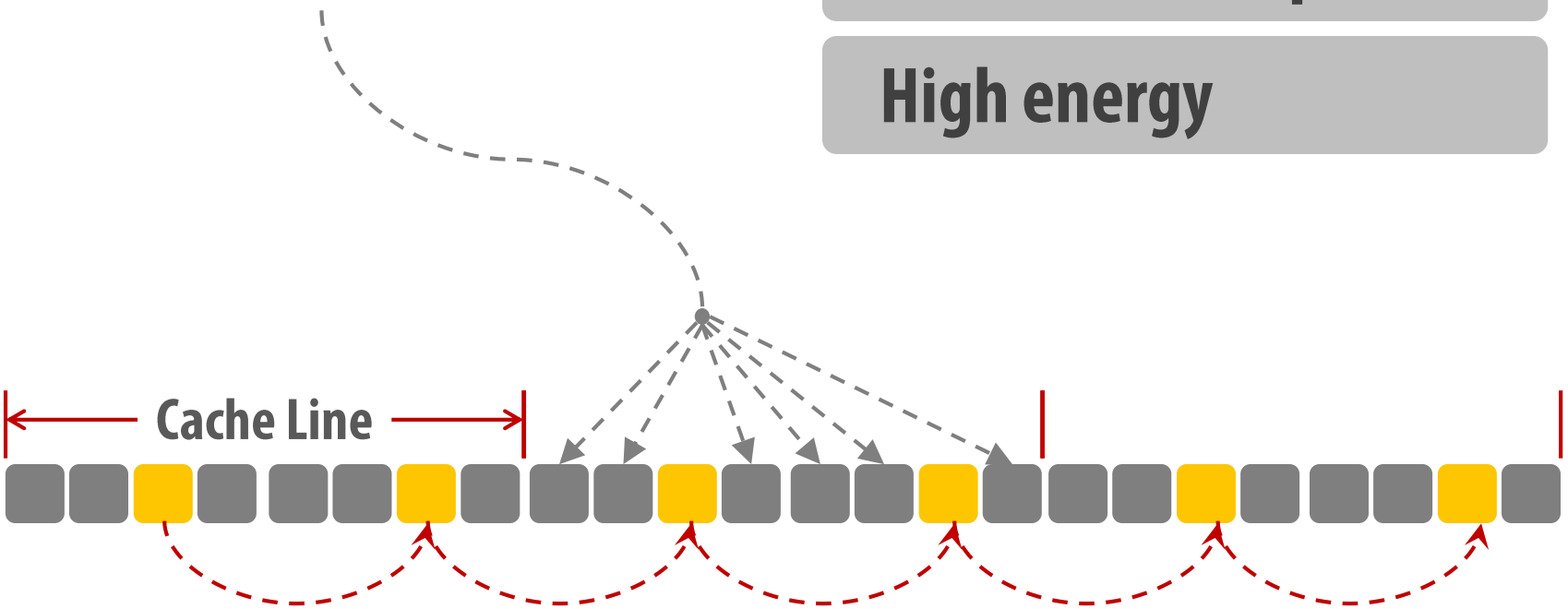
Physical layout of the data structure (row store)



Shortcomings of existing systems

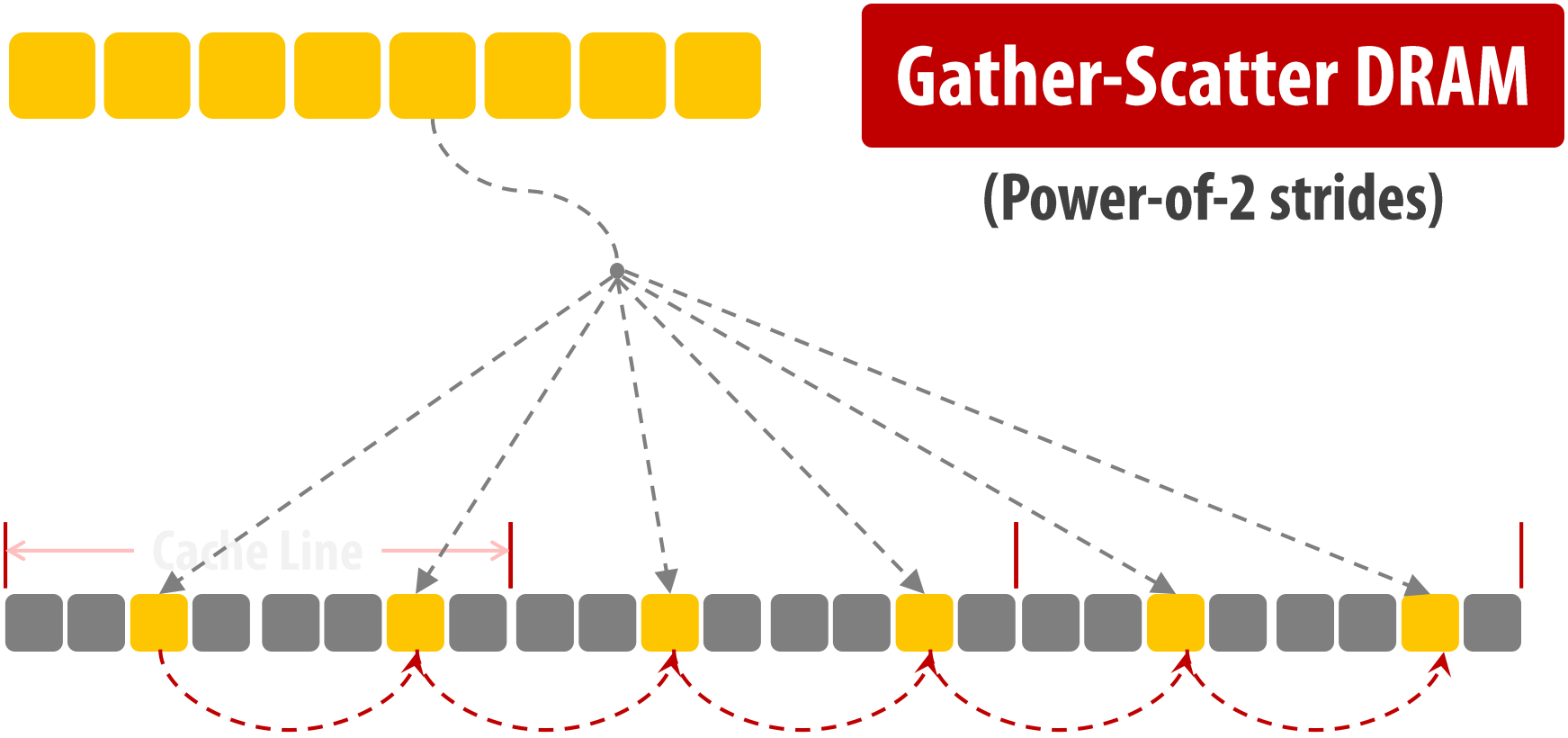
Data **unnecessarily** transferred on the **memory channel** and stored in **on-chip cache**

- High latency
- Wasted bandwidth
- Wasted cache space
- High energy

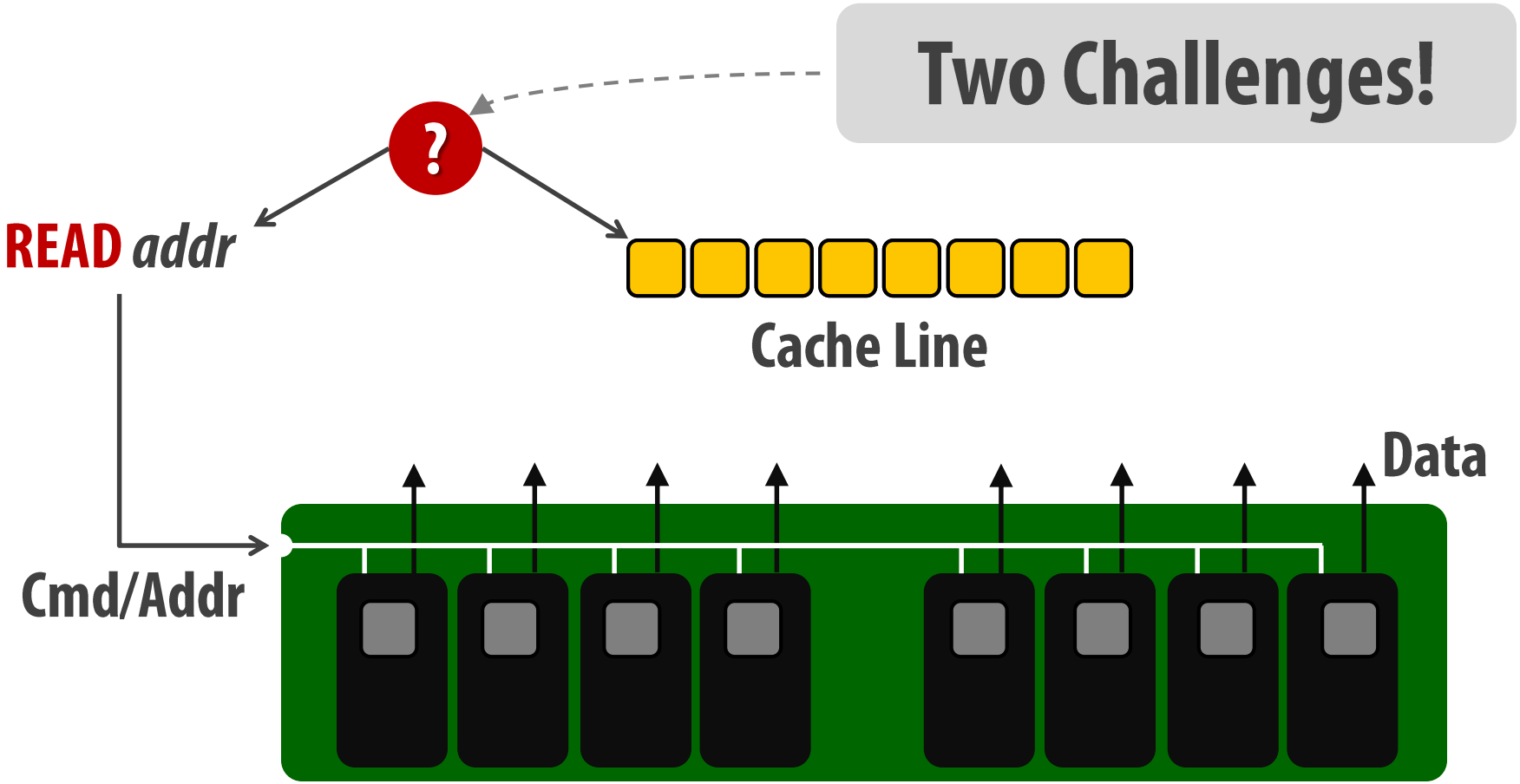


Goal: Eliminate inefficiency

Can we retrieve only useful data from memory?

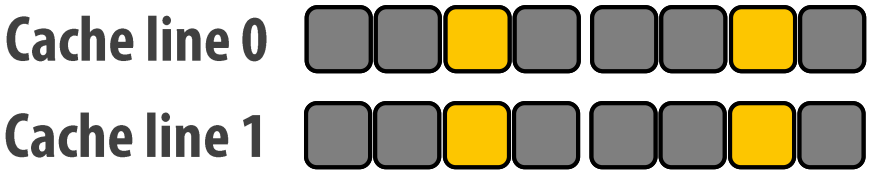


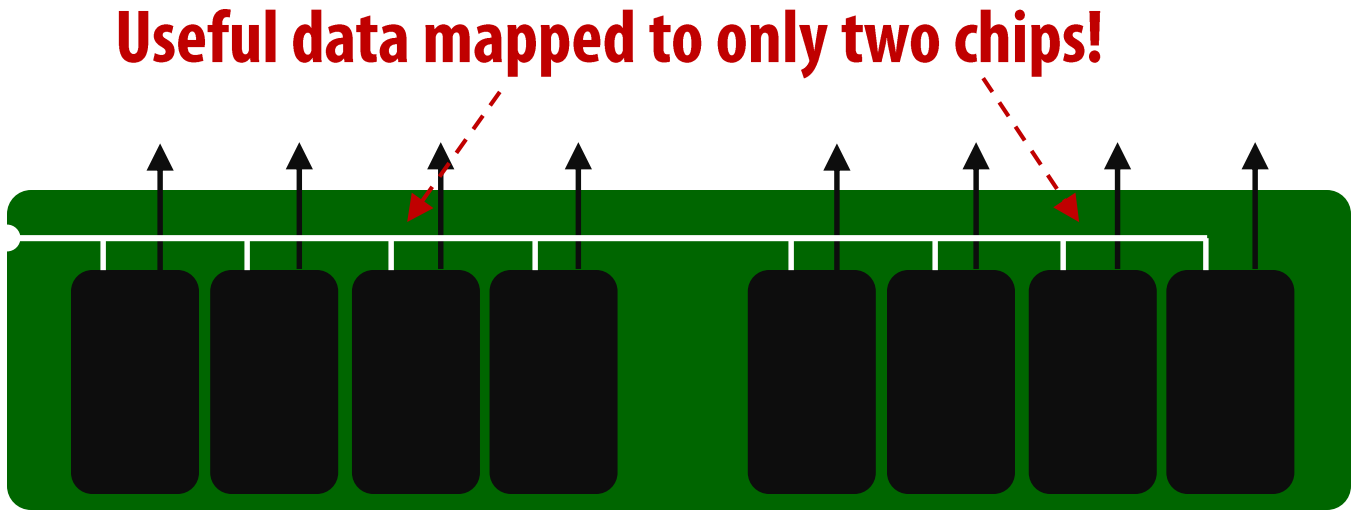
DRAM modules have multiple chips



Challenge 1: Chip conflicts

Data of each cache line is spread across all the chips!



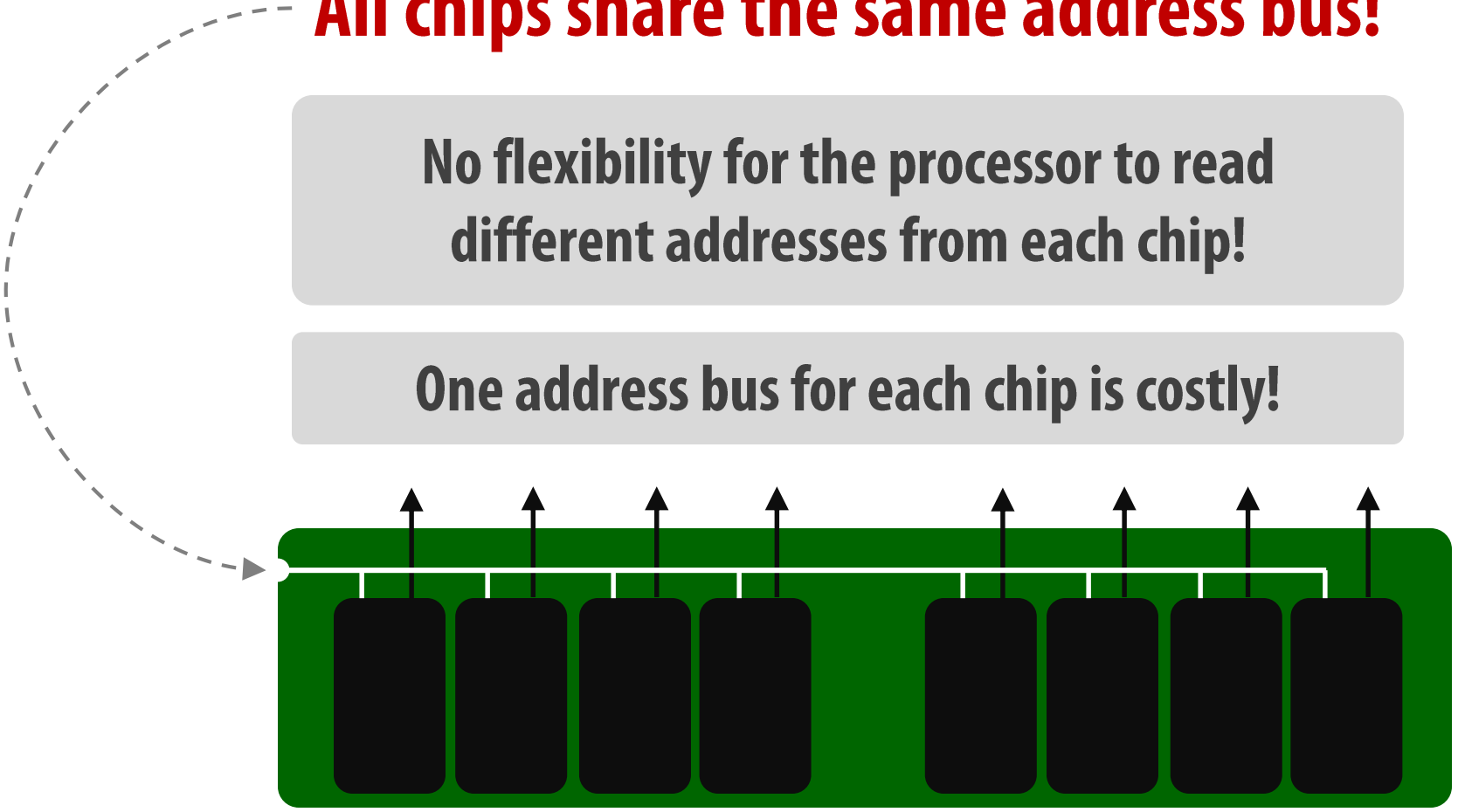


Challenge 2: Shared address bus

All chips share the same address bus!

No flexibility for the processor to read different addresses from each chip!

One address bus for each chip is costly!



Gather-Scatter DRAM

Challenge 1: Minimizing chip conflicts

Cacheline-ID-based data shuffling

(shuffle data of each cache line differently)

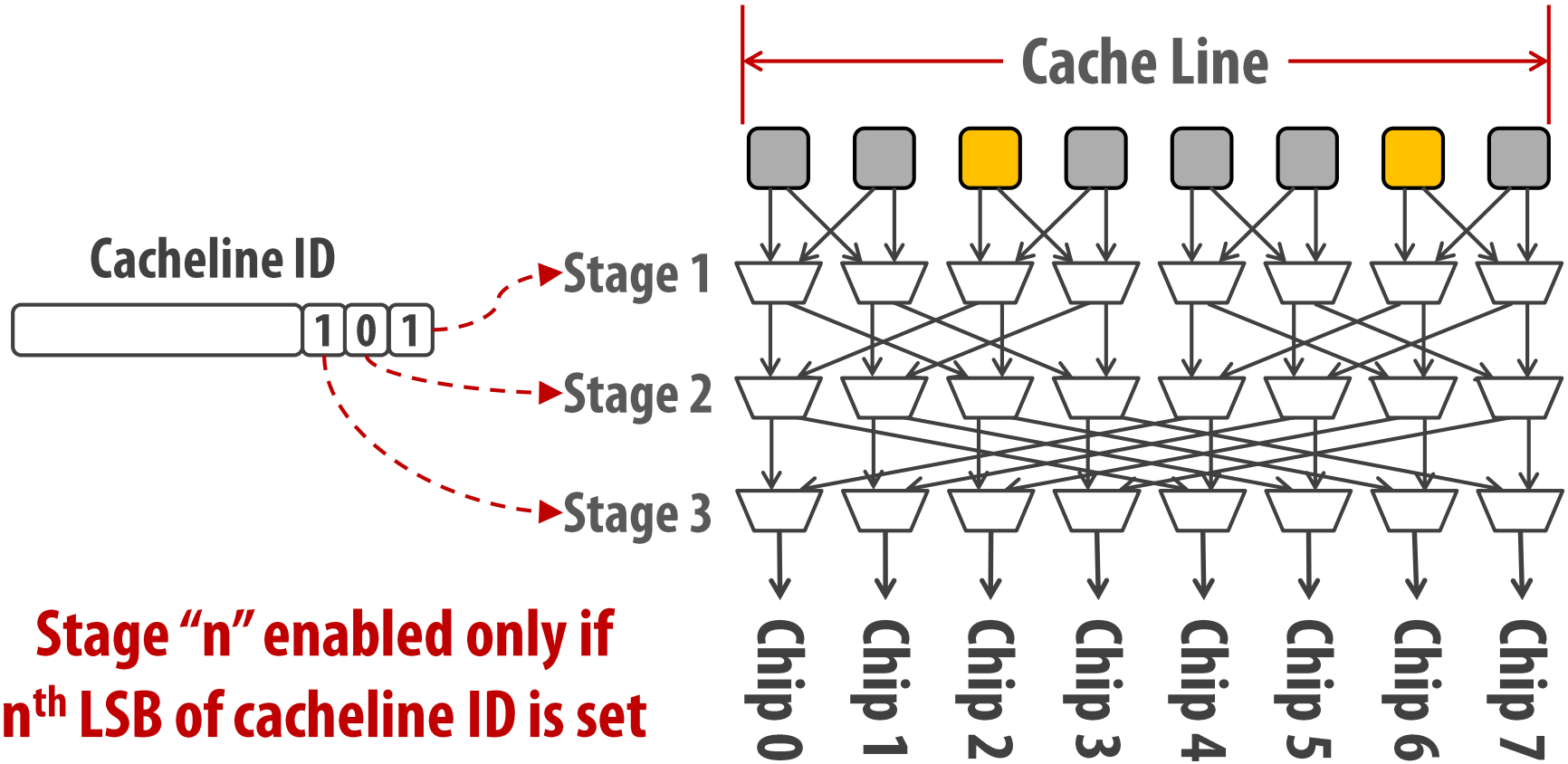
Challenge 2: Shared address bus

Pattern ID – In-DRAM address translation

(locally compute column address at each chip)

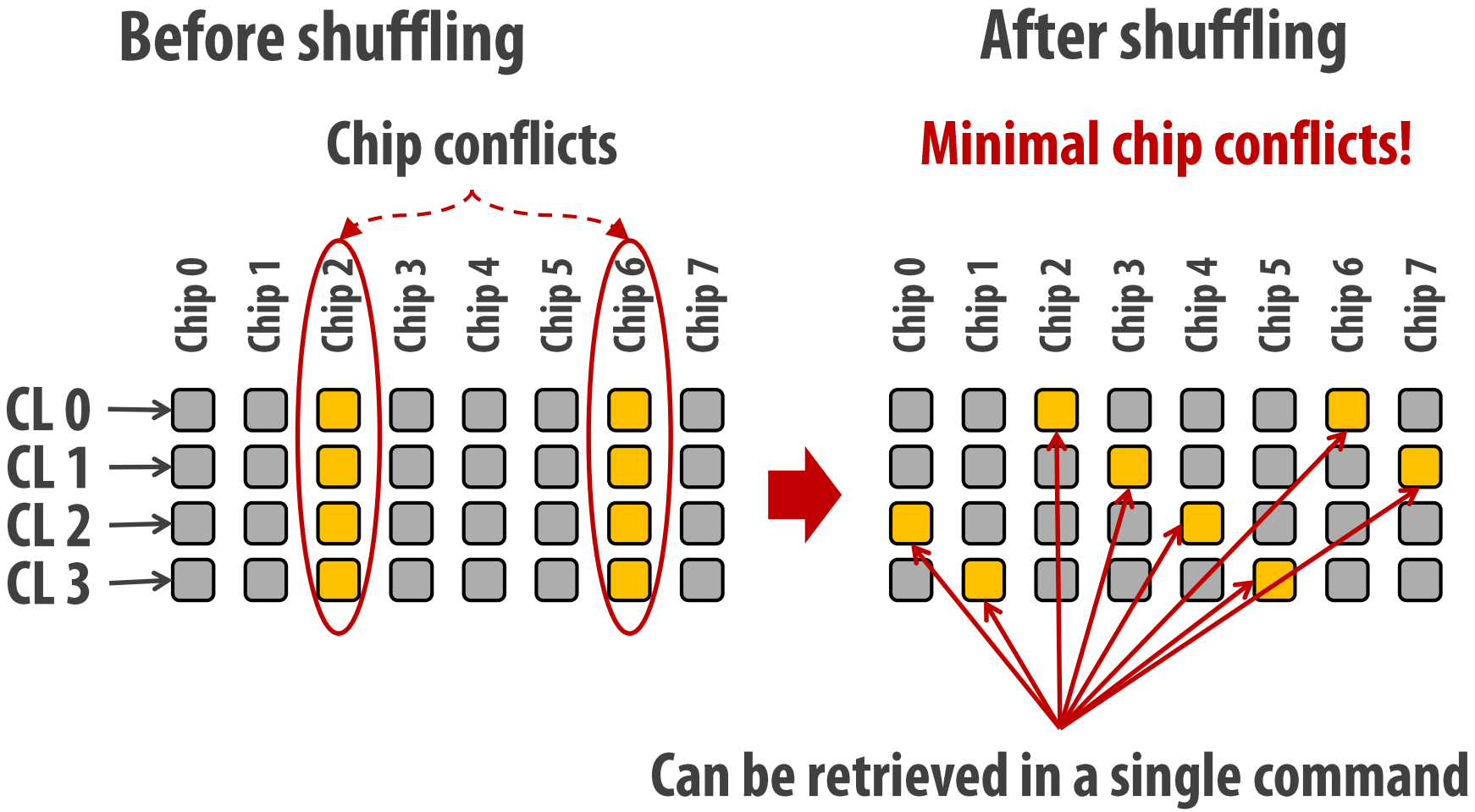
Cacheline-ID-based data shuffling

(implemented in the memory controller)



Stage "n" enabled only if n^{th} LSB of cacheline ID is set

Effect of data shuffling



Gather-Scatter DRAM

Challenge 1: Minimizing chip conflicts

Cacheline-ID-based data shuffling

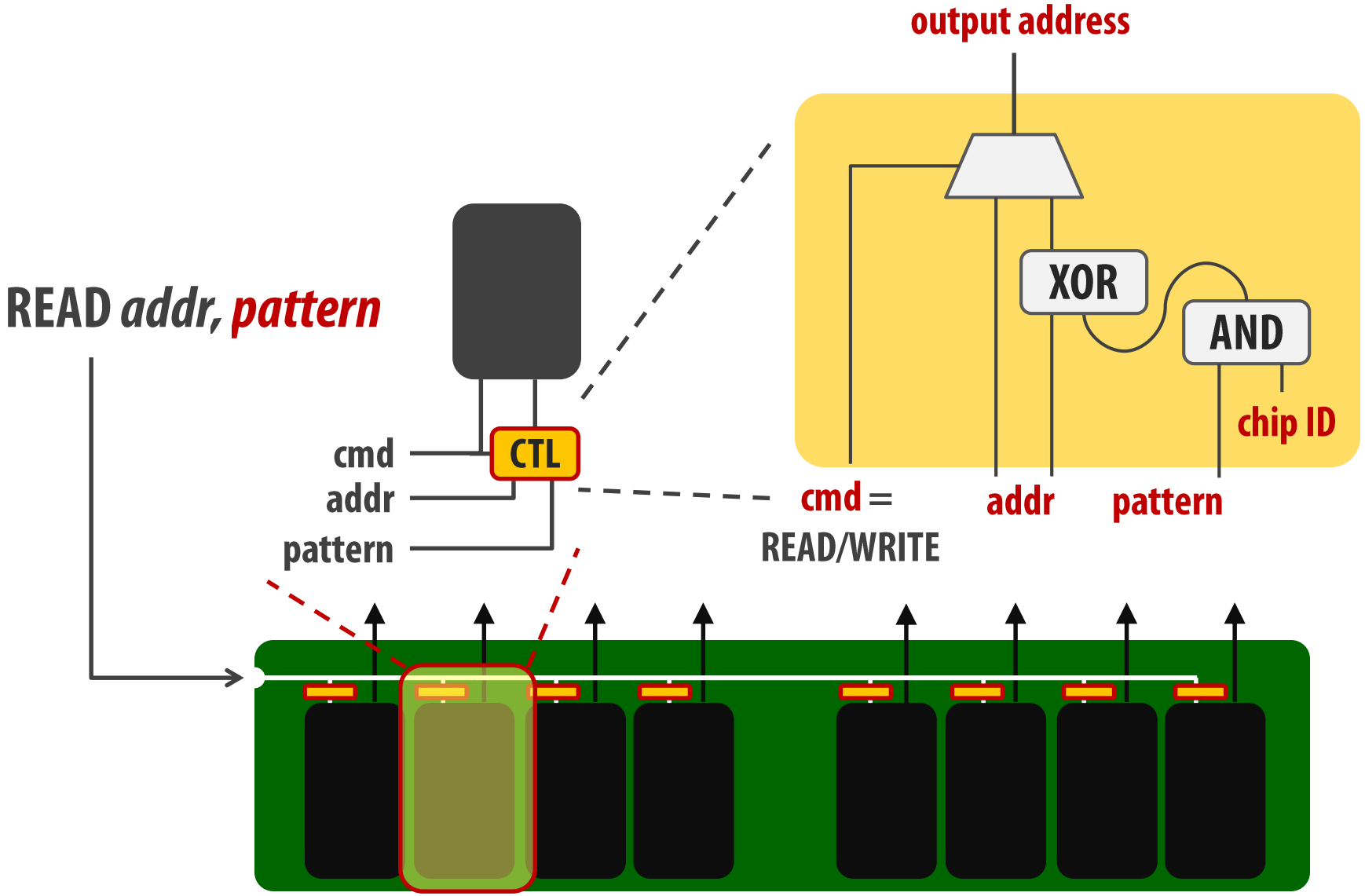
(shuffle data of each cache line differently)

Challenge 2: Shared address bus

Pattern ID – In-DRAM address translation

(locally compute the cacheline address at each chip)

Per-chip column translation logic



Gather-Scatter DRAM (GS-DRAM)

32 values contiguously stored in DRAM (from address 0)

read addr **0**, pattern **0** (stride = 1, default operation)

read addr **0**, pattern **1** (stride = 2)

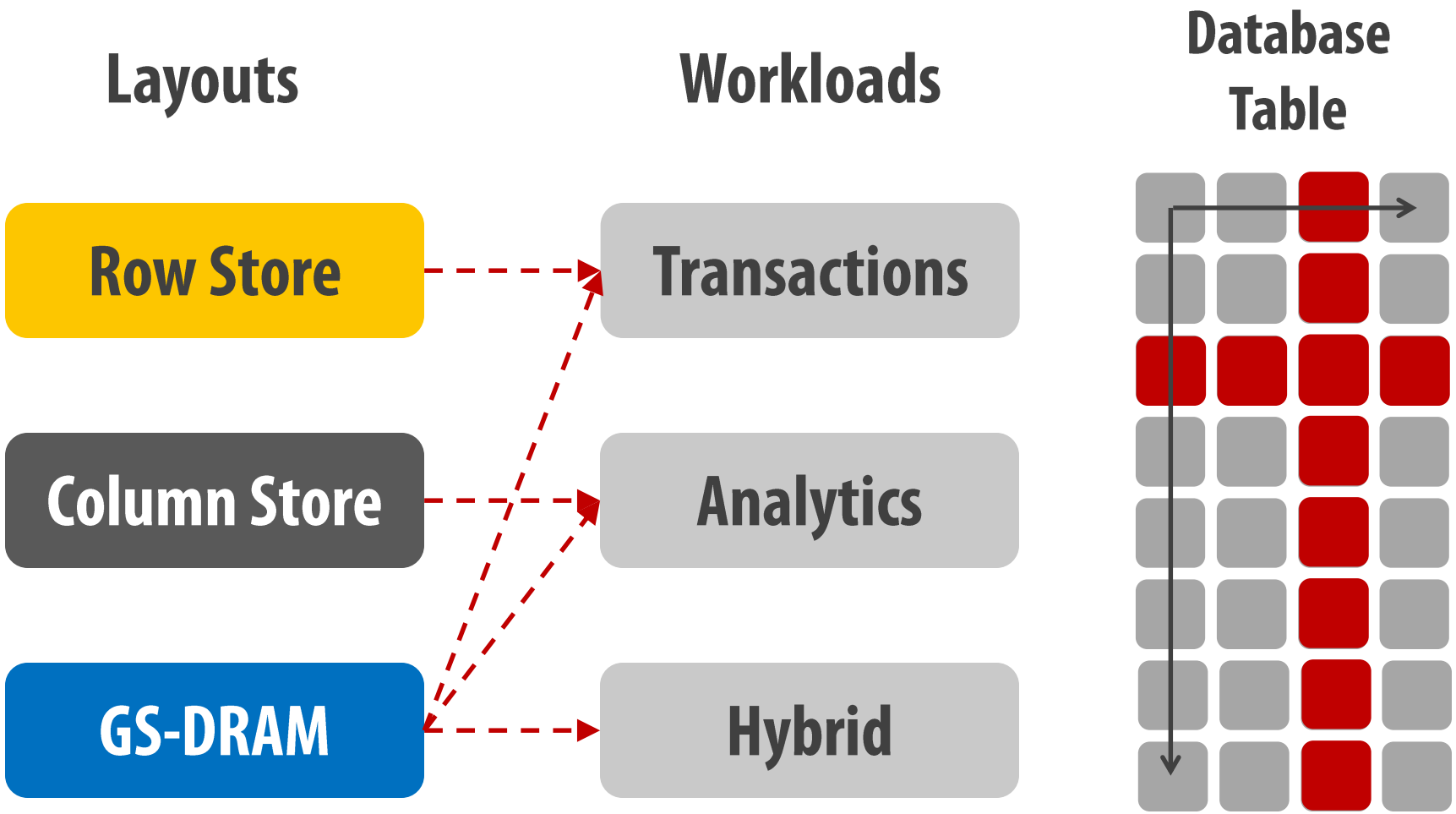
read addr **0**, pattern **3** (stride = 4)

read addr **0**, pattern **7** (stride = 8)

Methodology

- **Simulator**
  - Gem5 x86 simulator
  - Use "prefetch" instruction to implement pattern load
  - Cache hierarchy
    - 32KB L1 D/I cache, 2MB shared L2 cache
  - Main Memory: DDR3-1600, 1 channel, 1 rank, 8 banks
- **Energy evaluations**
  - McPAT + DRAMPower

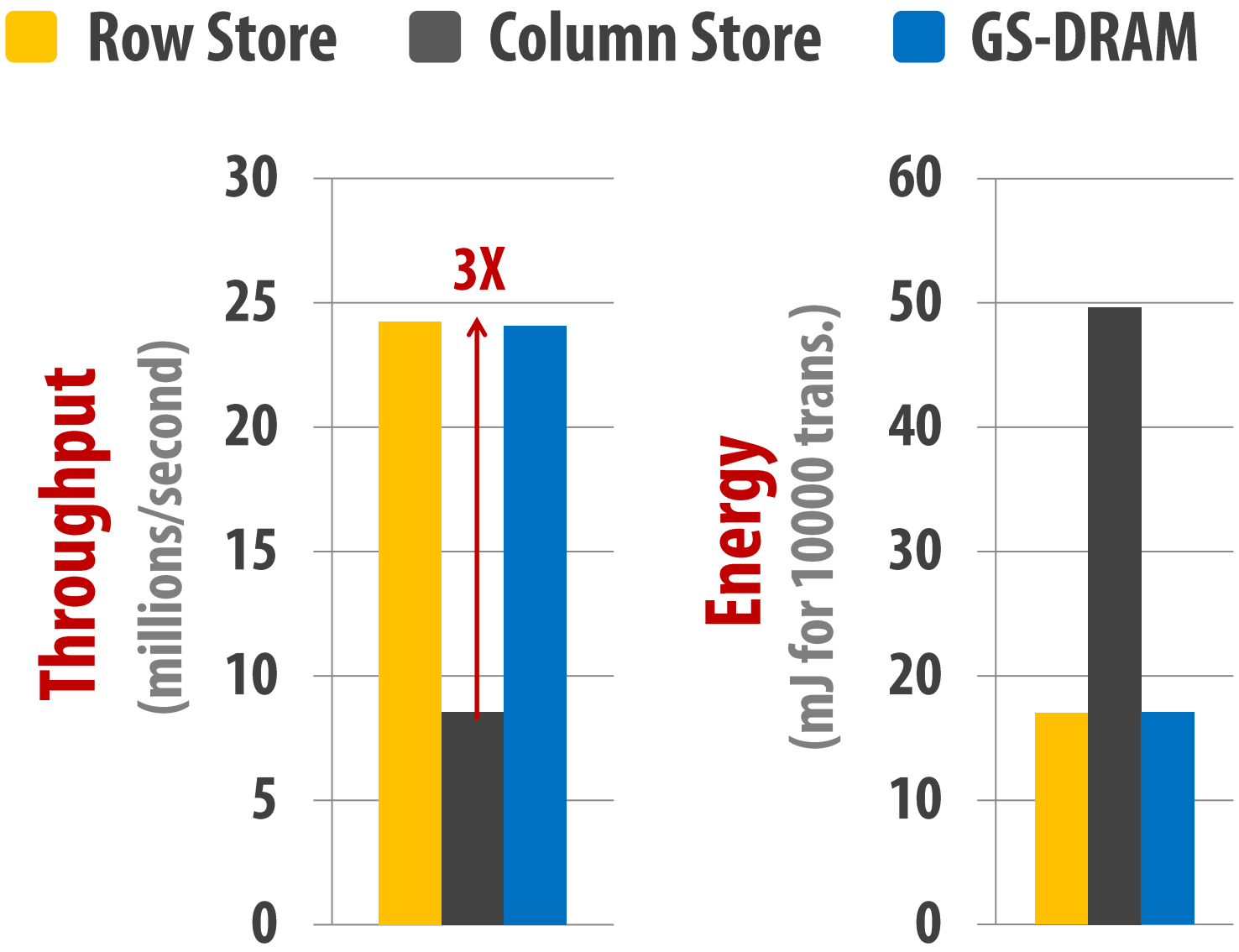
In-memory databases



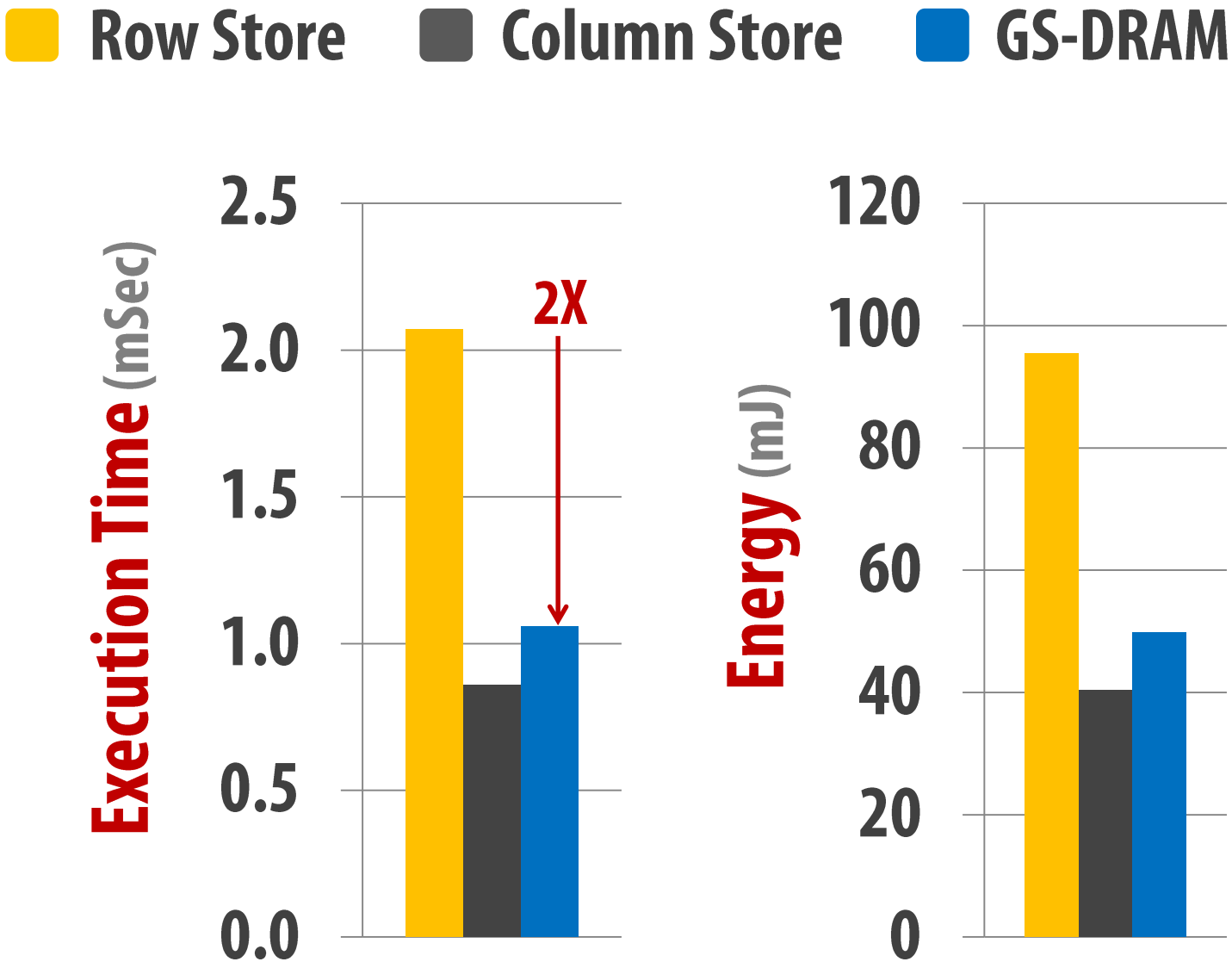
Workload

- **Database**
  - 1 table with million records
  - Each record = 1 cache line
- **Transactions**
  - Operate on a random record
  - Varying number of read-only/write-only/read-write fields
- **Analytics**
  - Sum of k columns
- **Hybrid**
  - Transactions thread: random records with 1 read-only, 1 write-only
  - Analytics thread: sum of one column

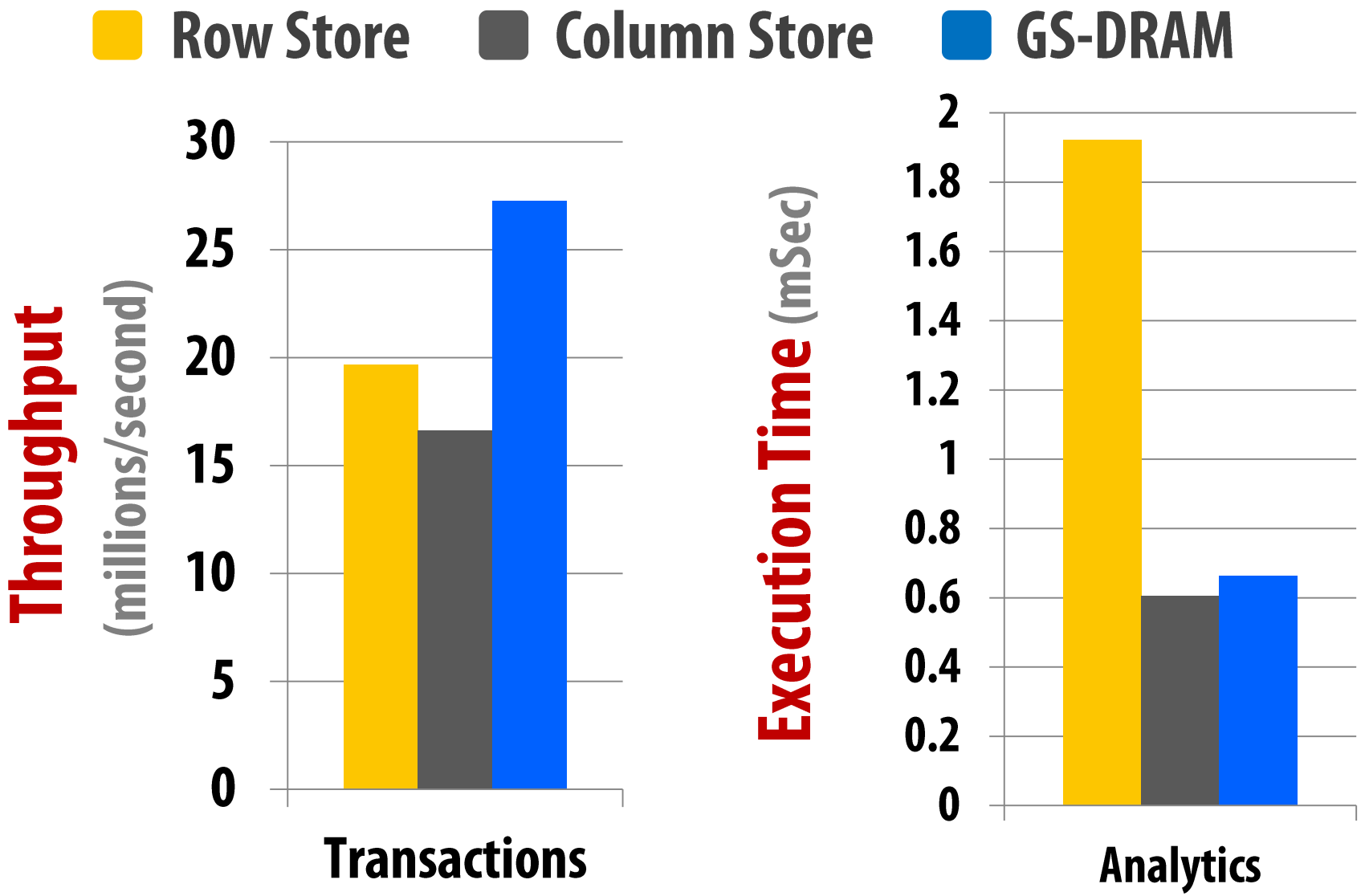
Transaction throughput and energy



Analytics performance and energy



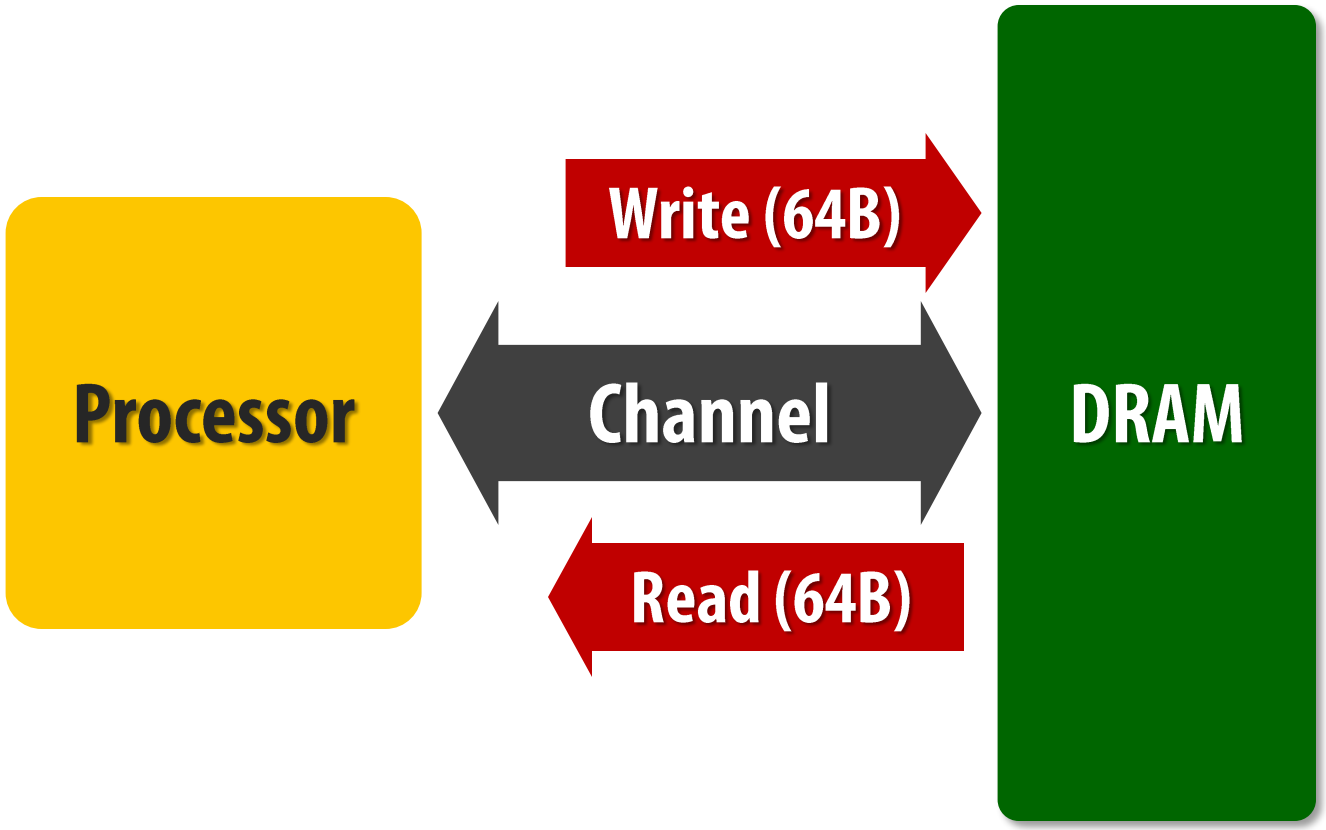
Hybrid Transactions/Analytical Processing



Outline for the Talk

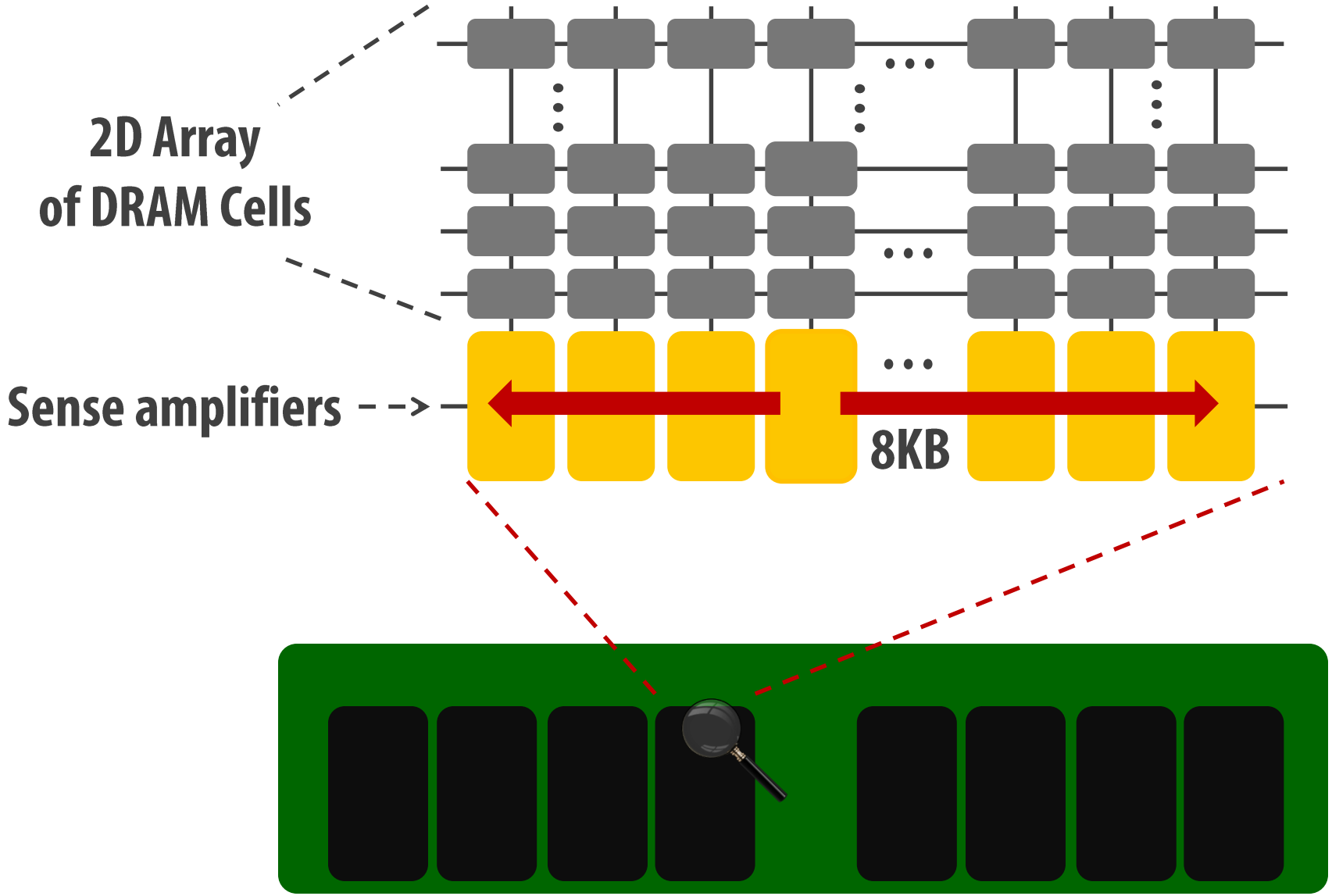
1. Page Overlays
   - efficient fine-grained memory management
2. Gather-Scatter DRAM
   - accelerating strided access patterns
3. RowClone + Buddy RAM
   - in-DRAM bulk copy + bitwise operations
4. Dirty-Block Index
   - maintaining coherence of dirty blocks

Today, DRAM is just a storage device!

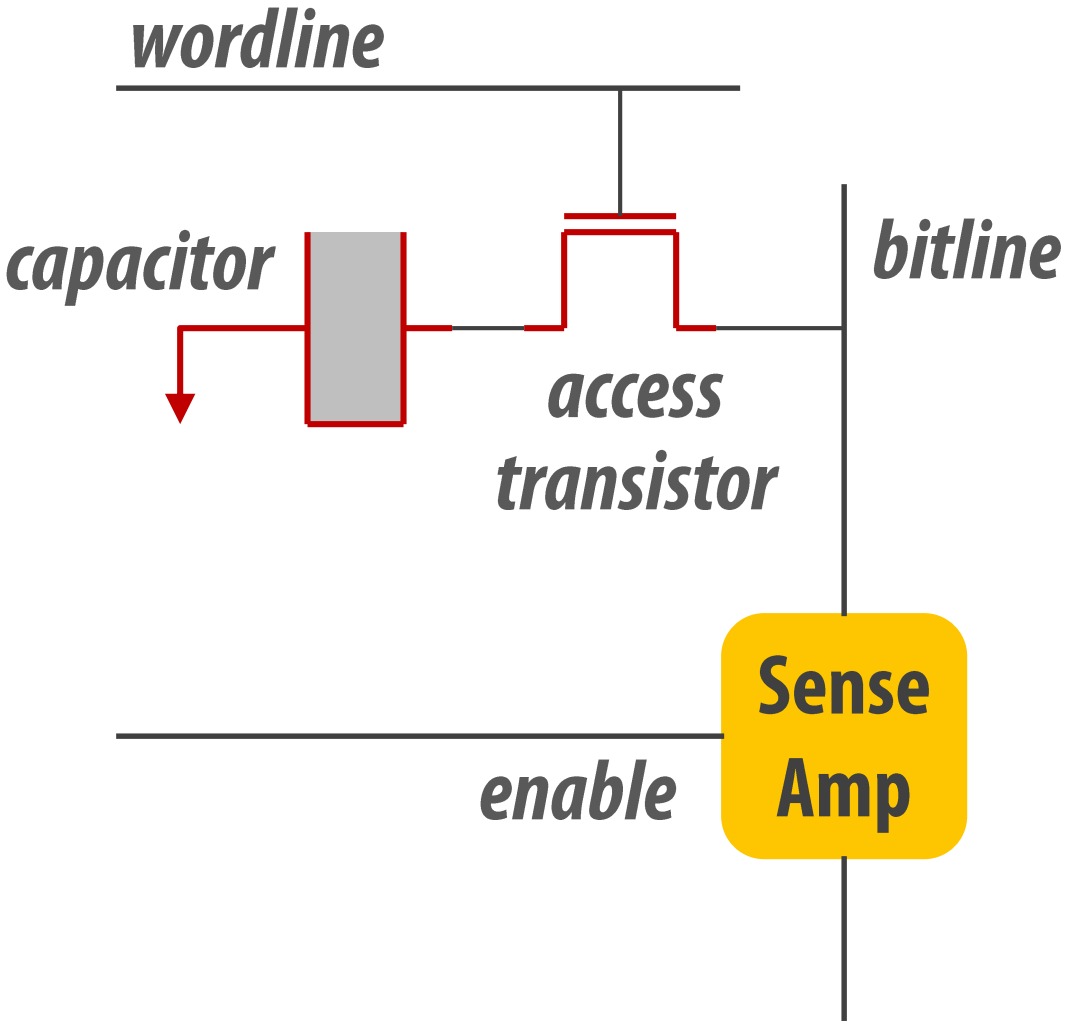


Bulk data operations => many data transfers

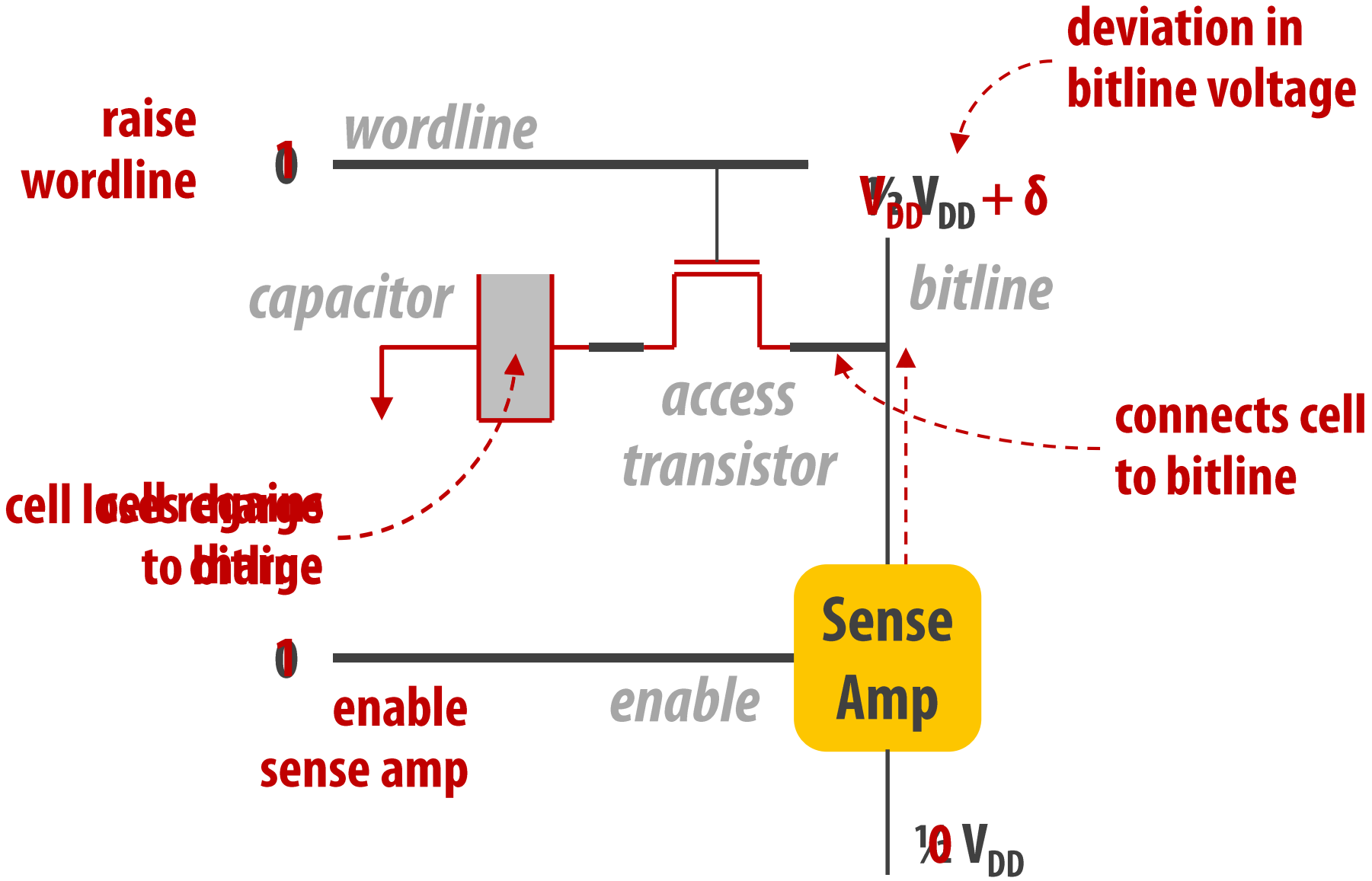
Inside a DRAM Chip



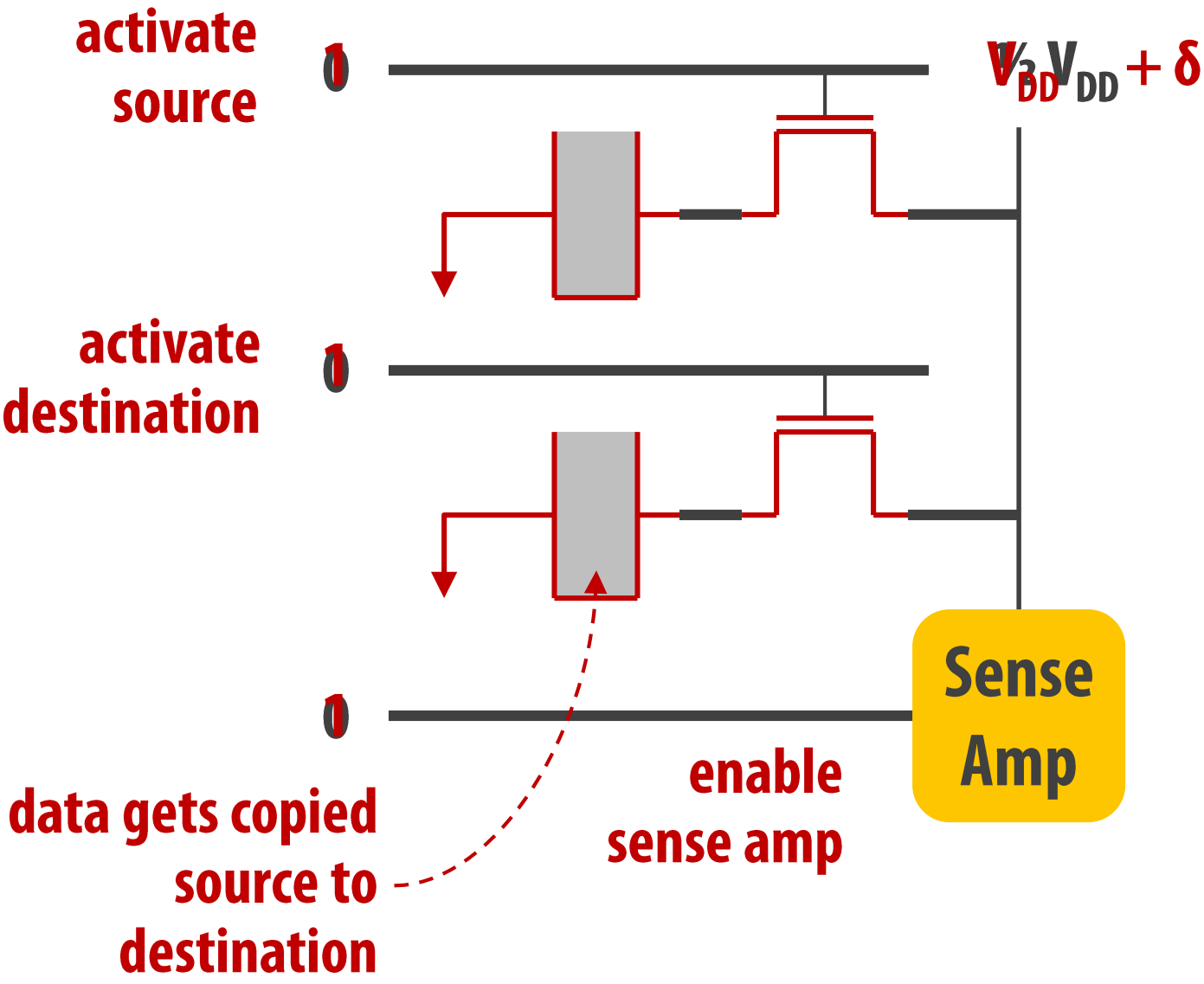
DRAM Cell Operation



DRAM Cell Operation

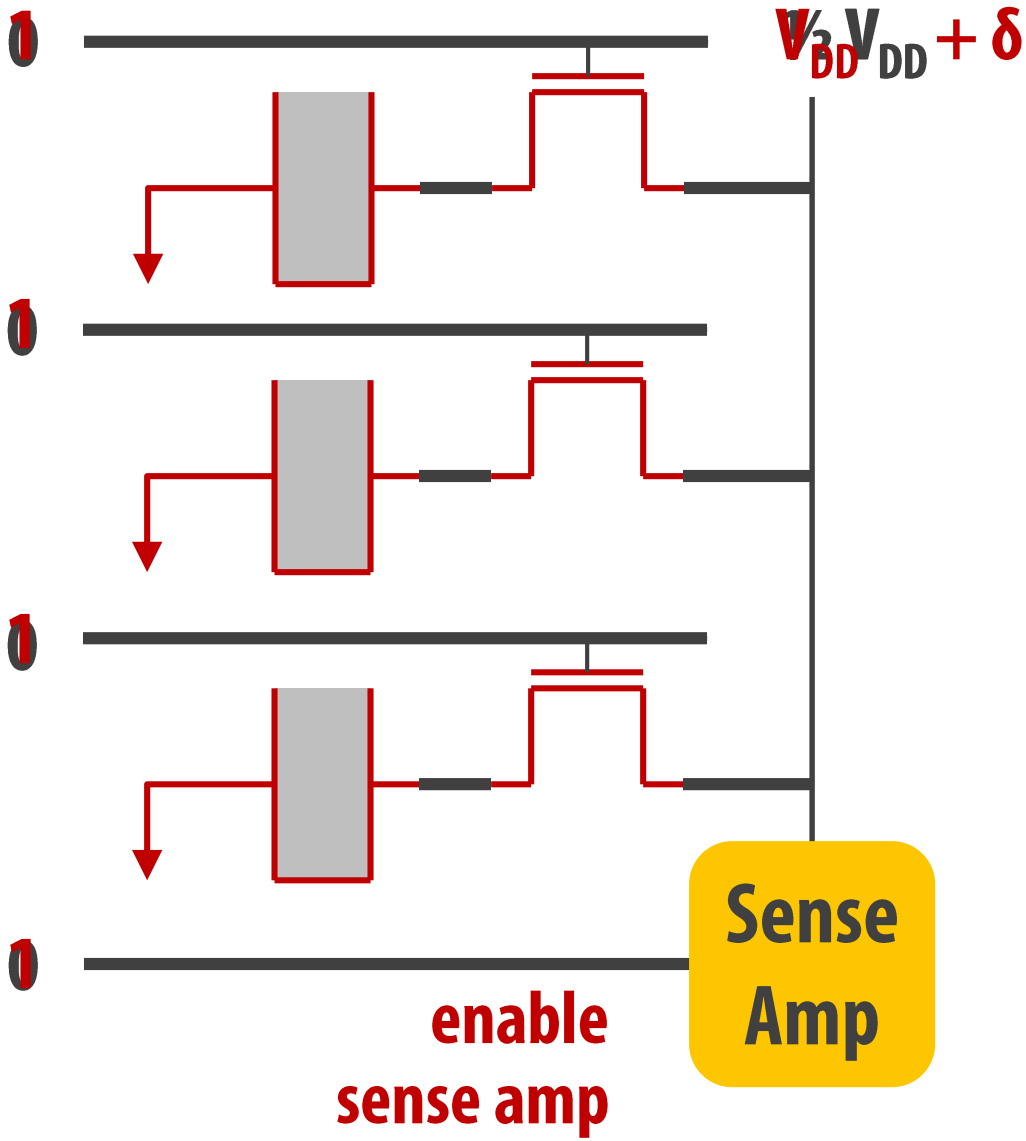


RowClone: In-DRAM Bulk Data Copy

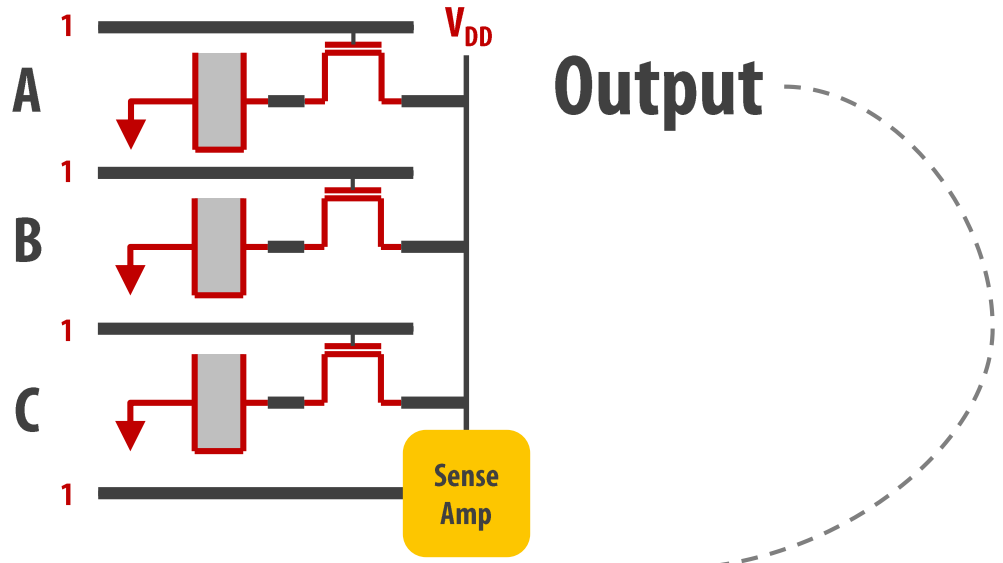


Triple-Row Activation

activate all three rows

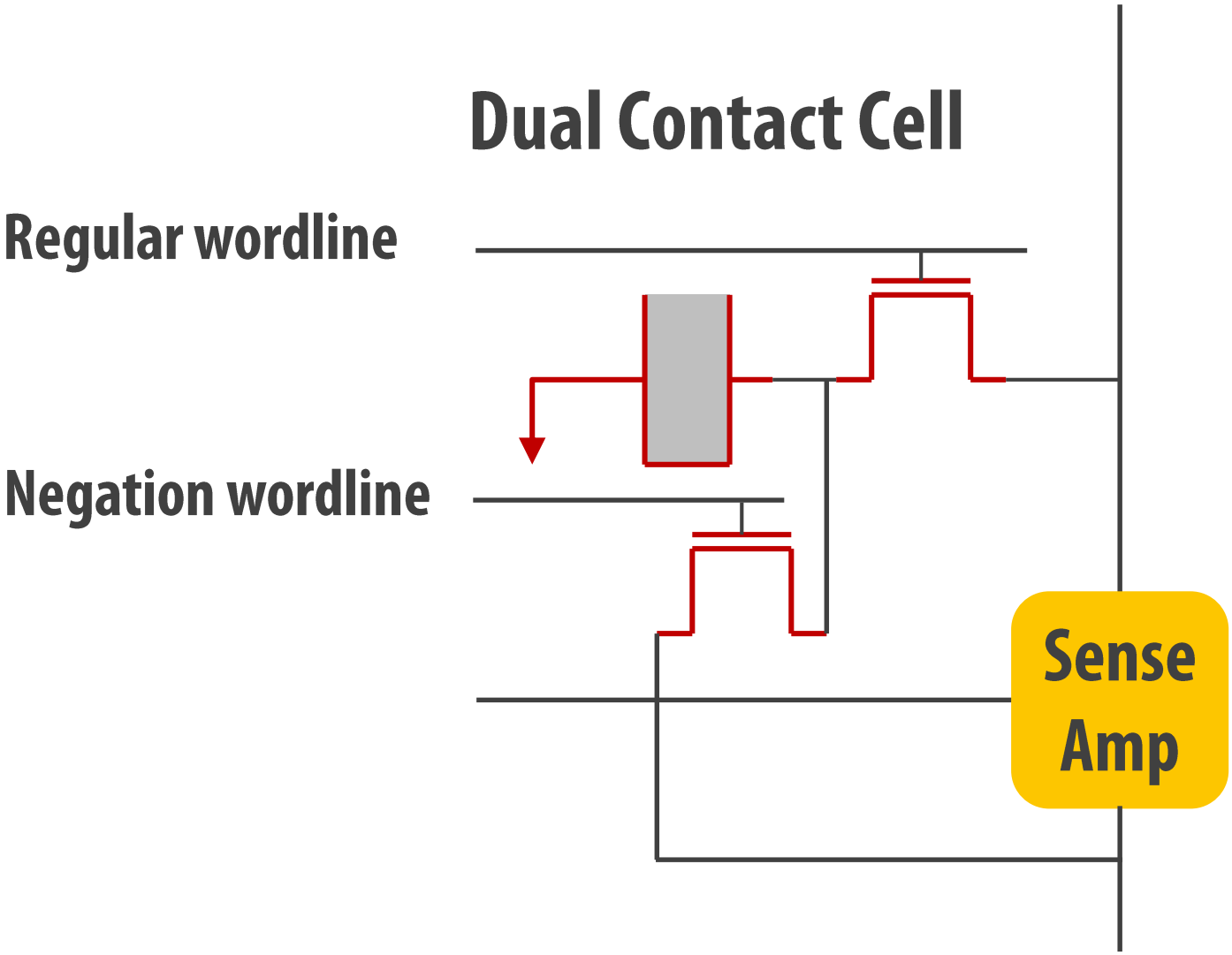


Bitwise AND/OR Using Triple-Row Activation

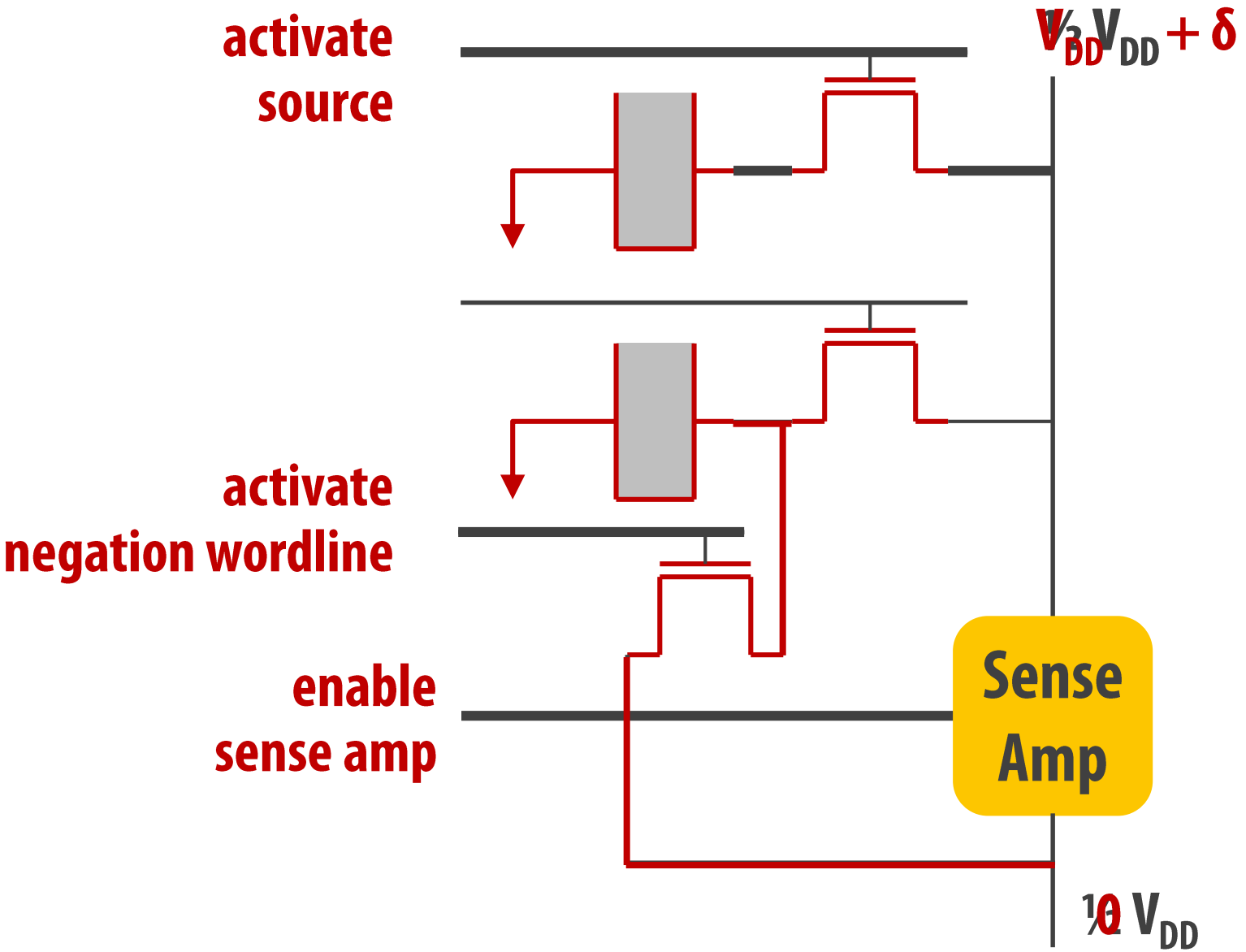



```
Output = AB + BC + CA
       = C (A OR B) + ~C (A AND B)
```

Negation Using the Sense Amplifier

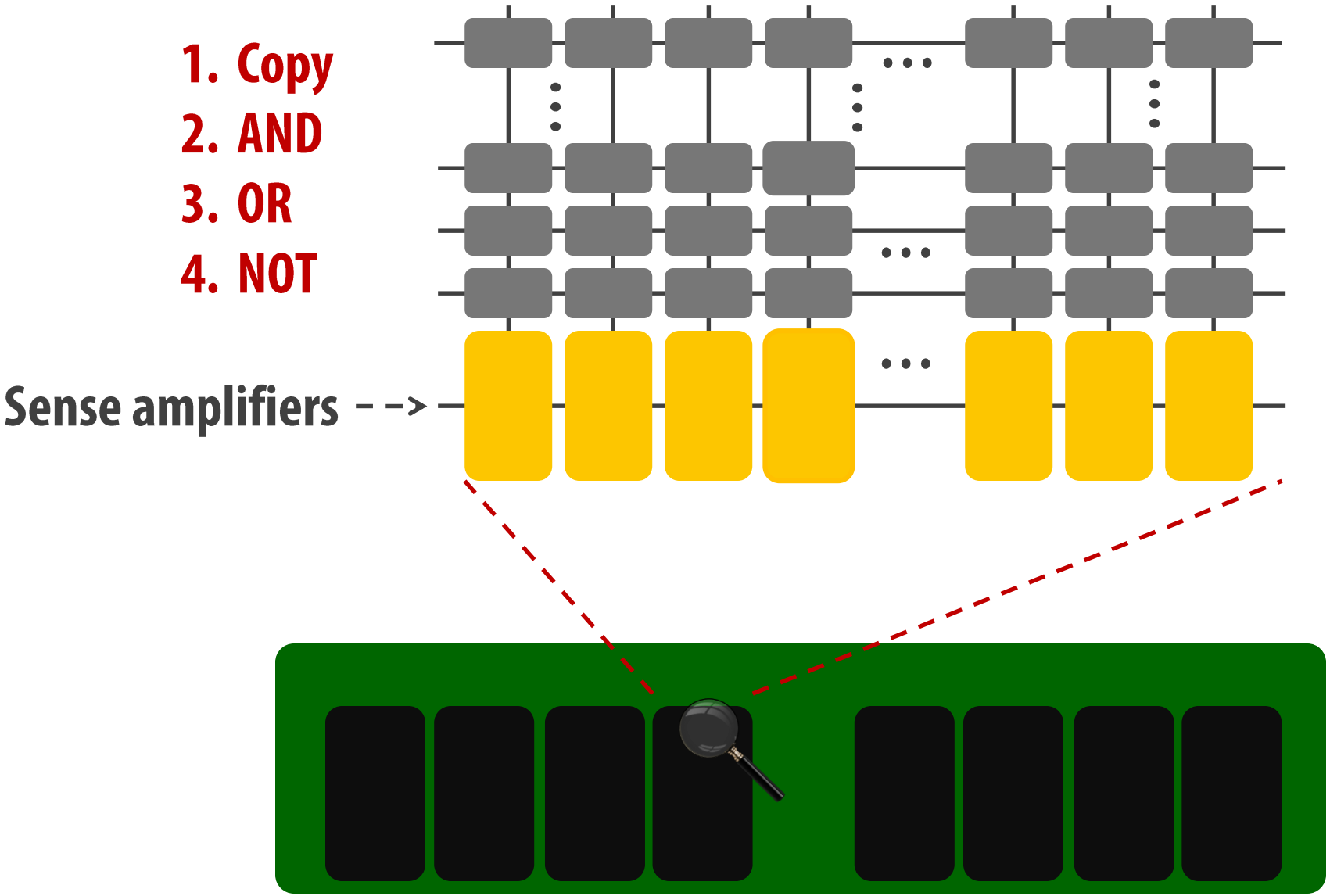


Negation Using the Sense Amplifier

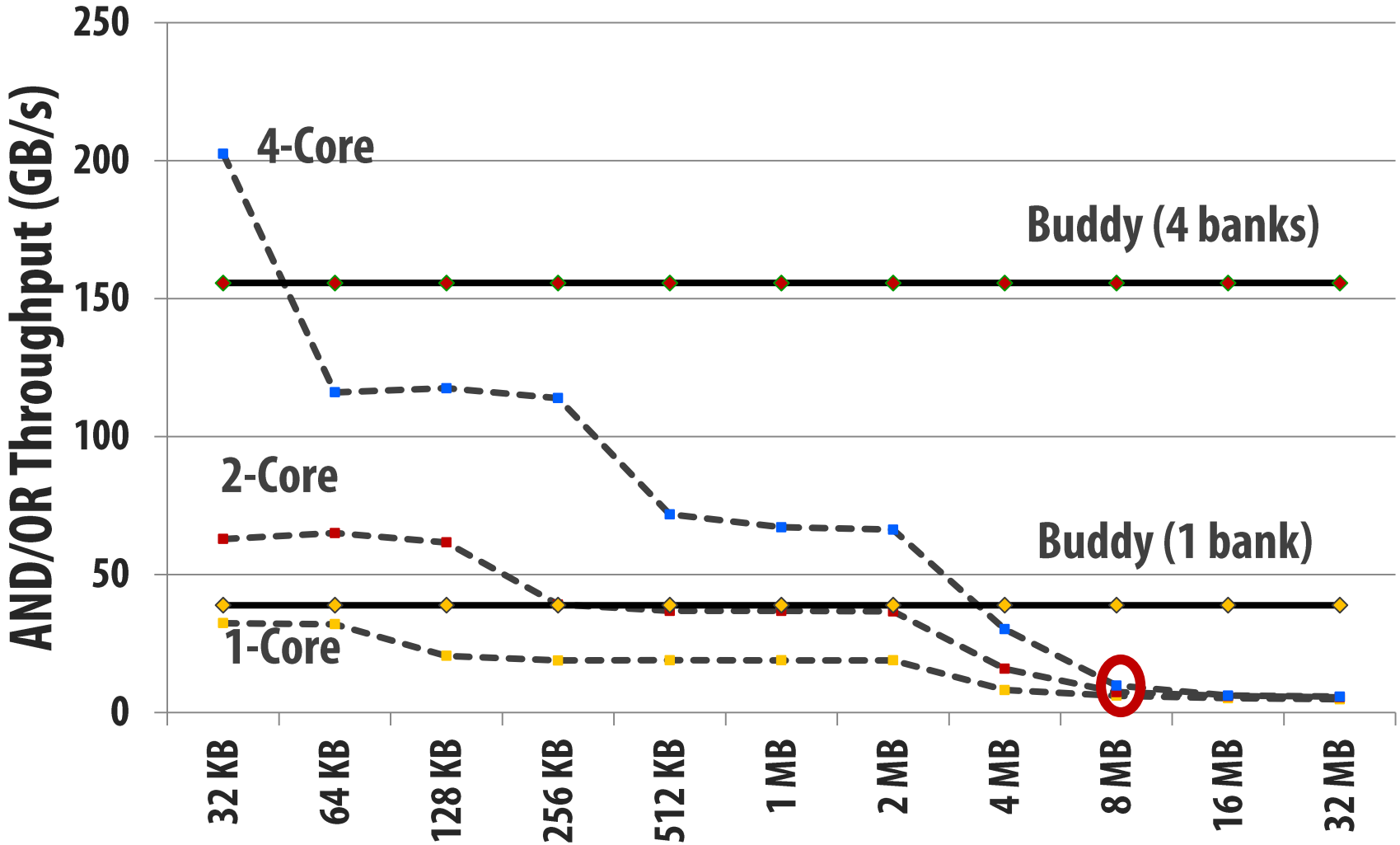


Summary of operations

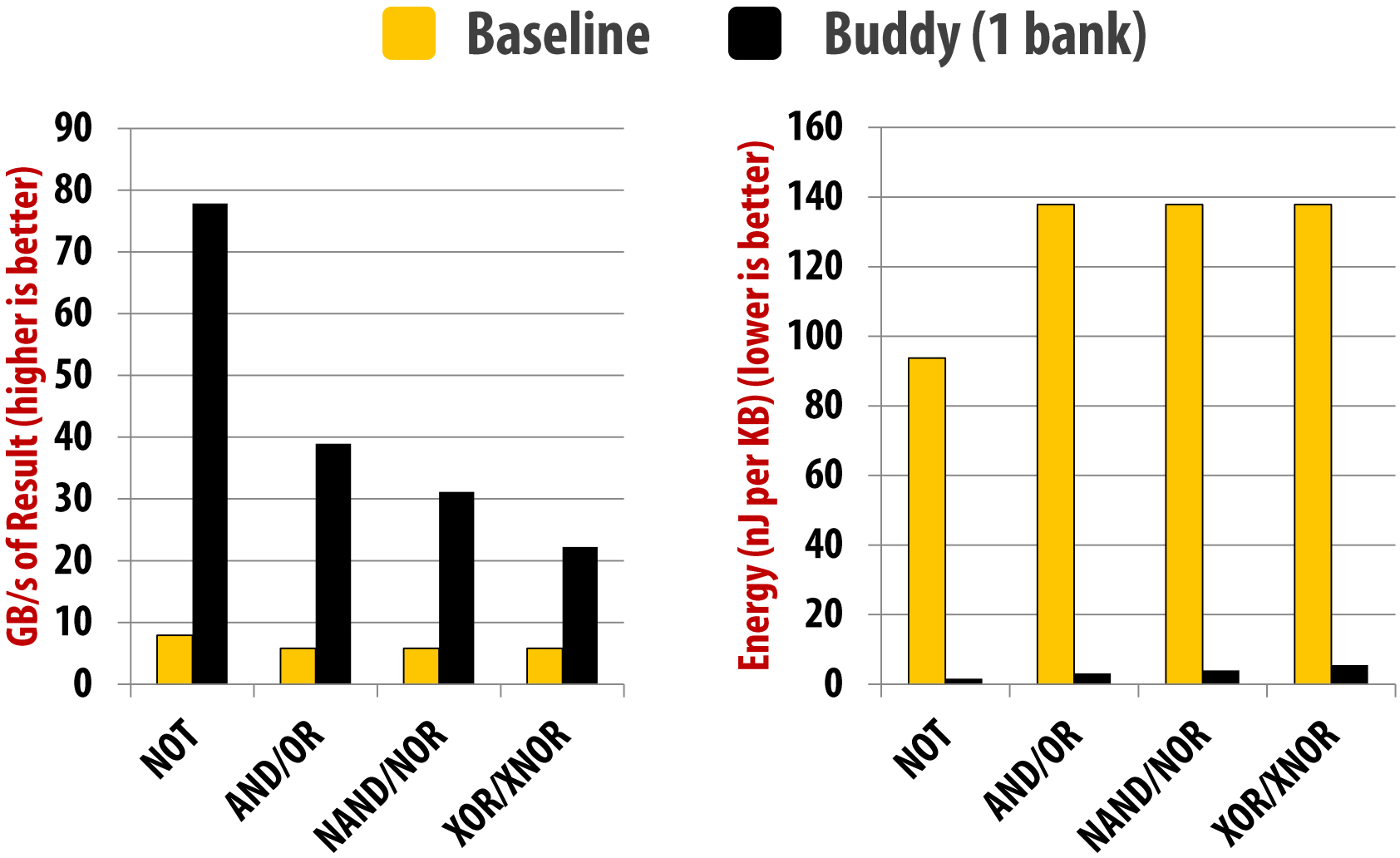
1. Copy
2. AND
3. OR
4. NOT



Throughput Comparison: Bitwise AND/OR



Throughput and Energy



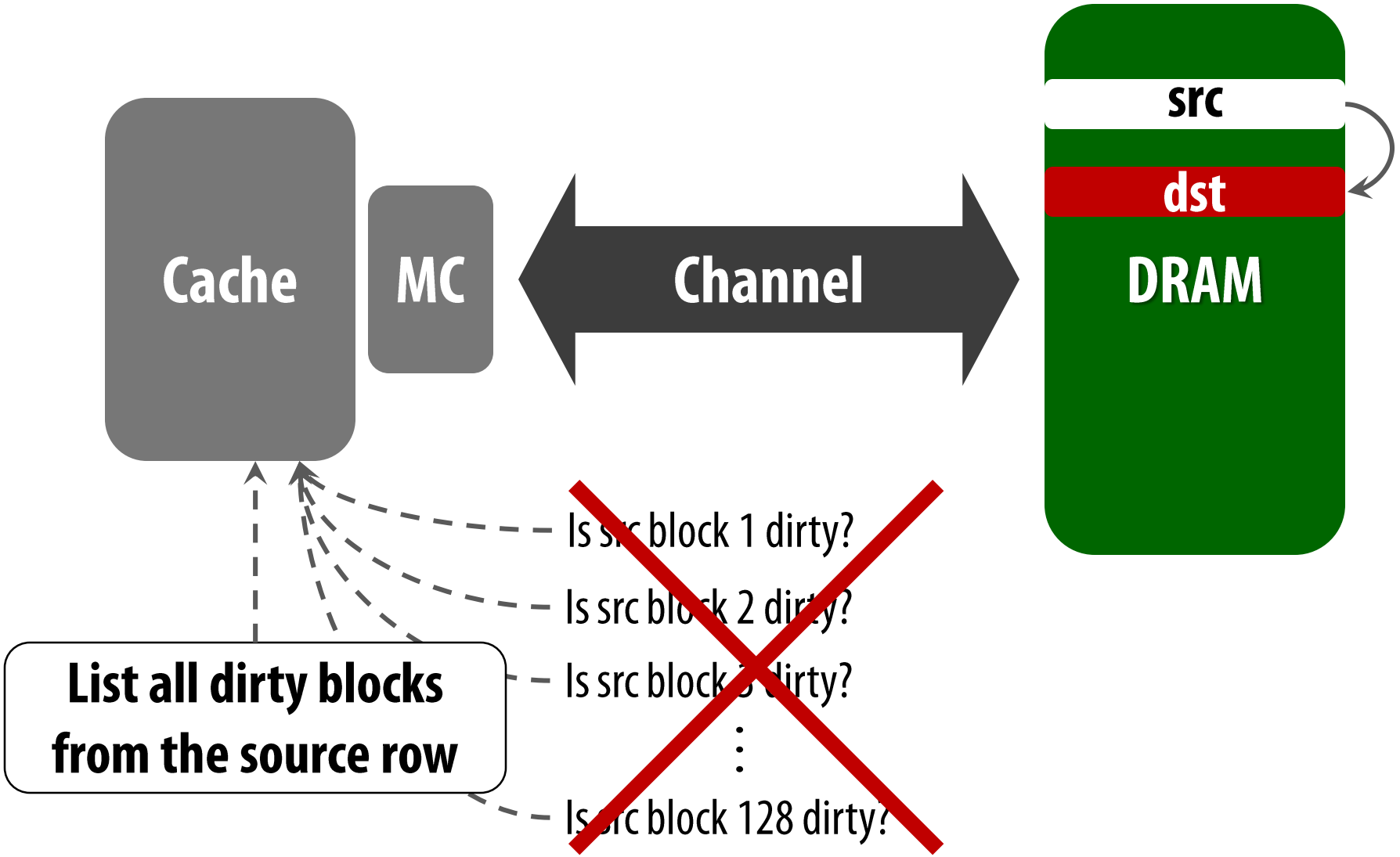
Applications

- Implementation of set operations
  - Buddy accelerates bitvector based implementations
  - More attractive than red-black trees
    - Insert, lookup, union, intersection, difference
- In-memory bitmap indices
  - Used by many online web applications
  - Buddy improves query performance by 6X

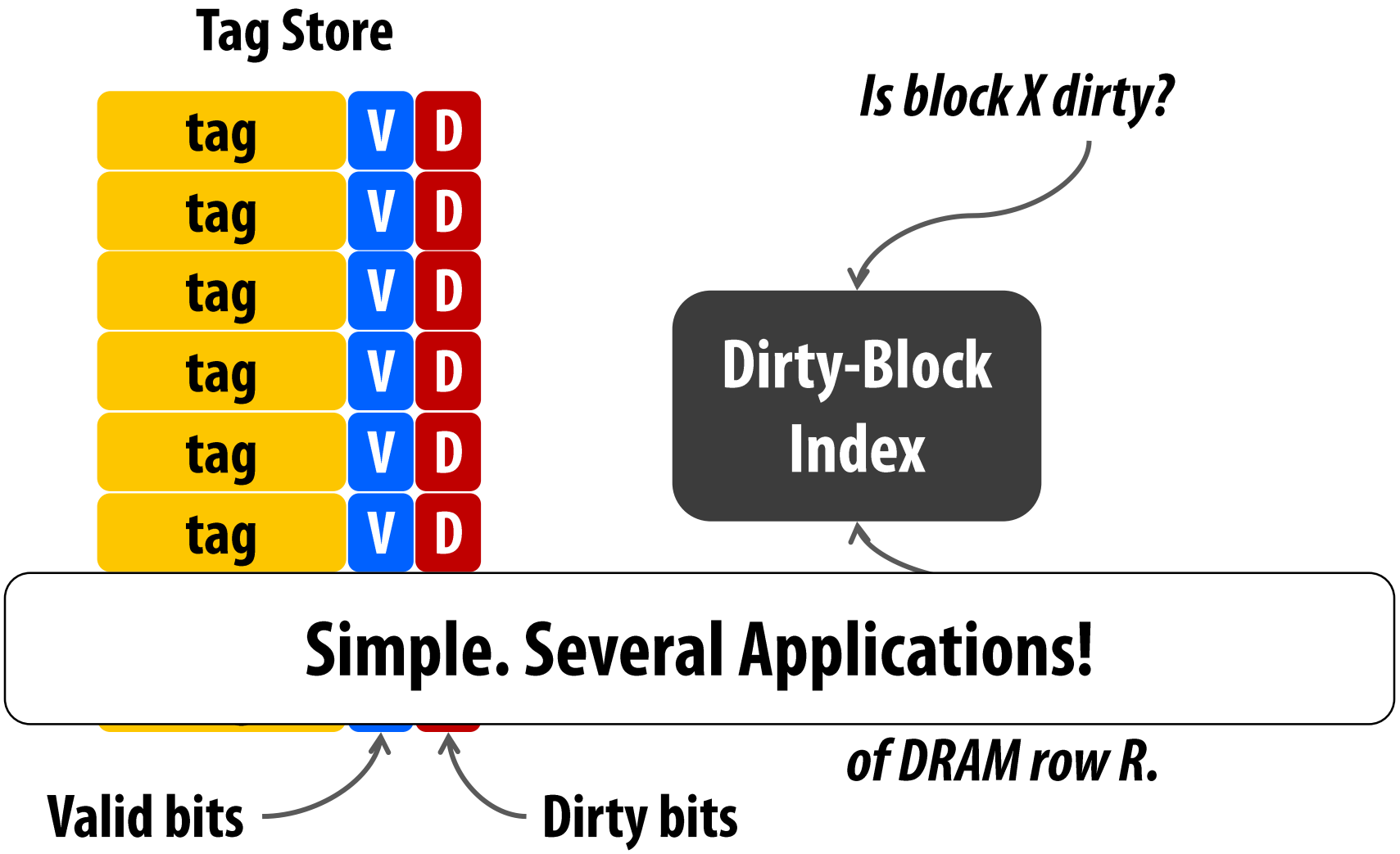
Outline for the Talk

1. Page Overlays
   - efficient fine-grained memory management
2. Gather-Scatter DRAM
   - accelerating strided access patterns
3. RowClone + Buddy RAM
   - in-DRAM bulk copy + bitwise operations
4. Dirty-Block Index
   - maintaining coherence of dirty blocks

Cache Coherence Problem



Dirty-Block Index



Acknowledgments

- Todd Mowry and Onur Mutlu
- Phil Gibbons and Mike Kozuch
- Dave Andersen and Rajeev Balasubramonian
- LBA and SAFARI
- Deb!
- CALCM and PDL
- Squash partners
- Friends
- Family – parents and brother

Conclusion

- **Different memory resources are managed and accessed at different granularities**
  - Results in significant inefficiency for many operations
- **Simple, low-cost abstractions**
  - New virtual memory framework for fine-grained management
  - Techniques to use DRAM to do more than store data
- **Our mechanisms eliminate unnecessary work**
  - Reduce memory capacity consumption
  - Improve performance
  - Reduce energy consumption

Simple DRAM and Virtual Memory Abstractions for Highly Efficient Memory Systems

Thesis Oral

Vivek Seshadri

Committee:

Todd Mowry (Co-chair)

Onur Mutlu (Co-chair)

Phillip B. Gibbons

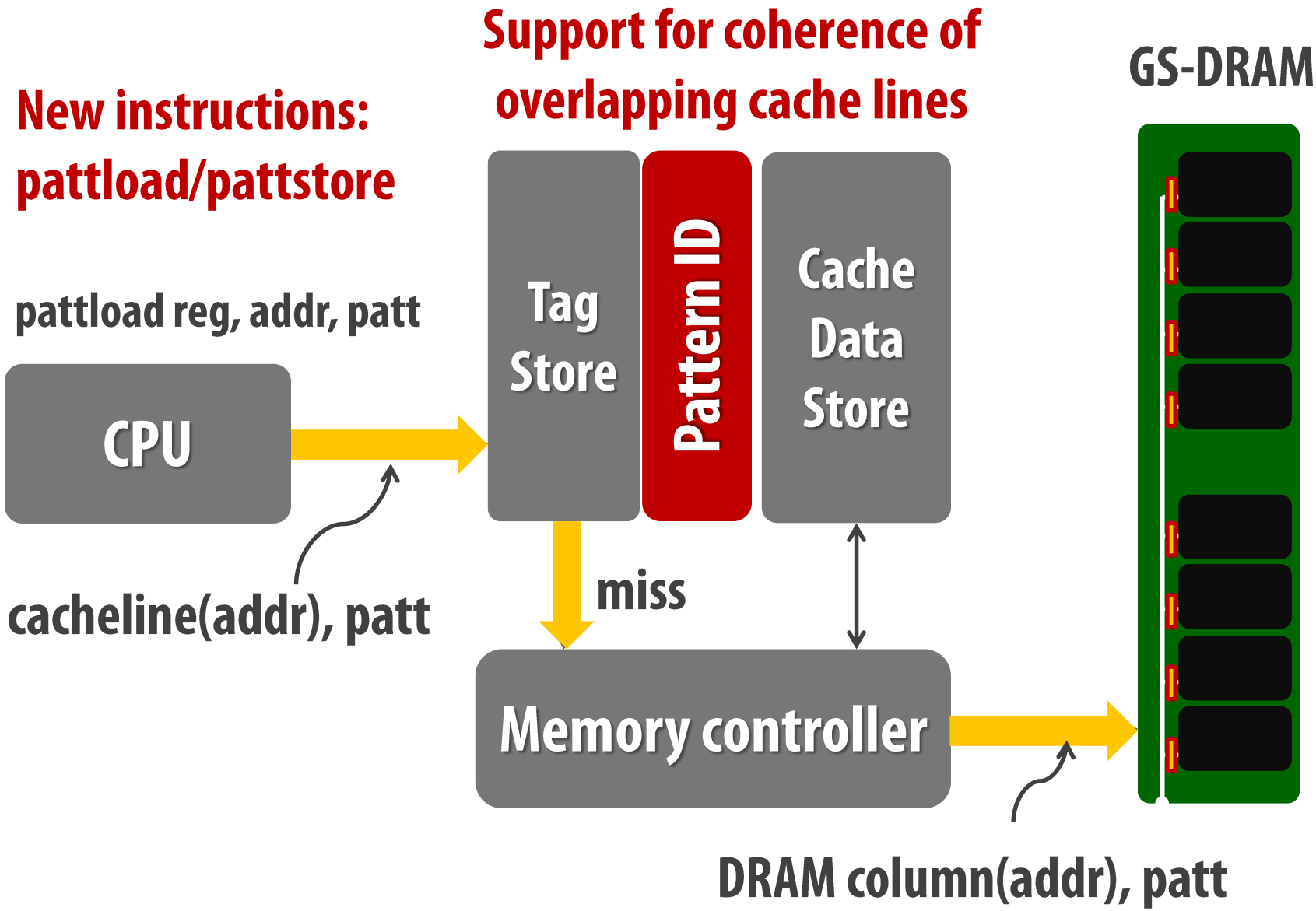
David Andersen

Rajeev Balasubramonian, University of Utah

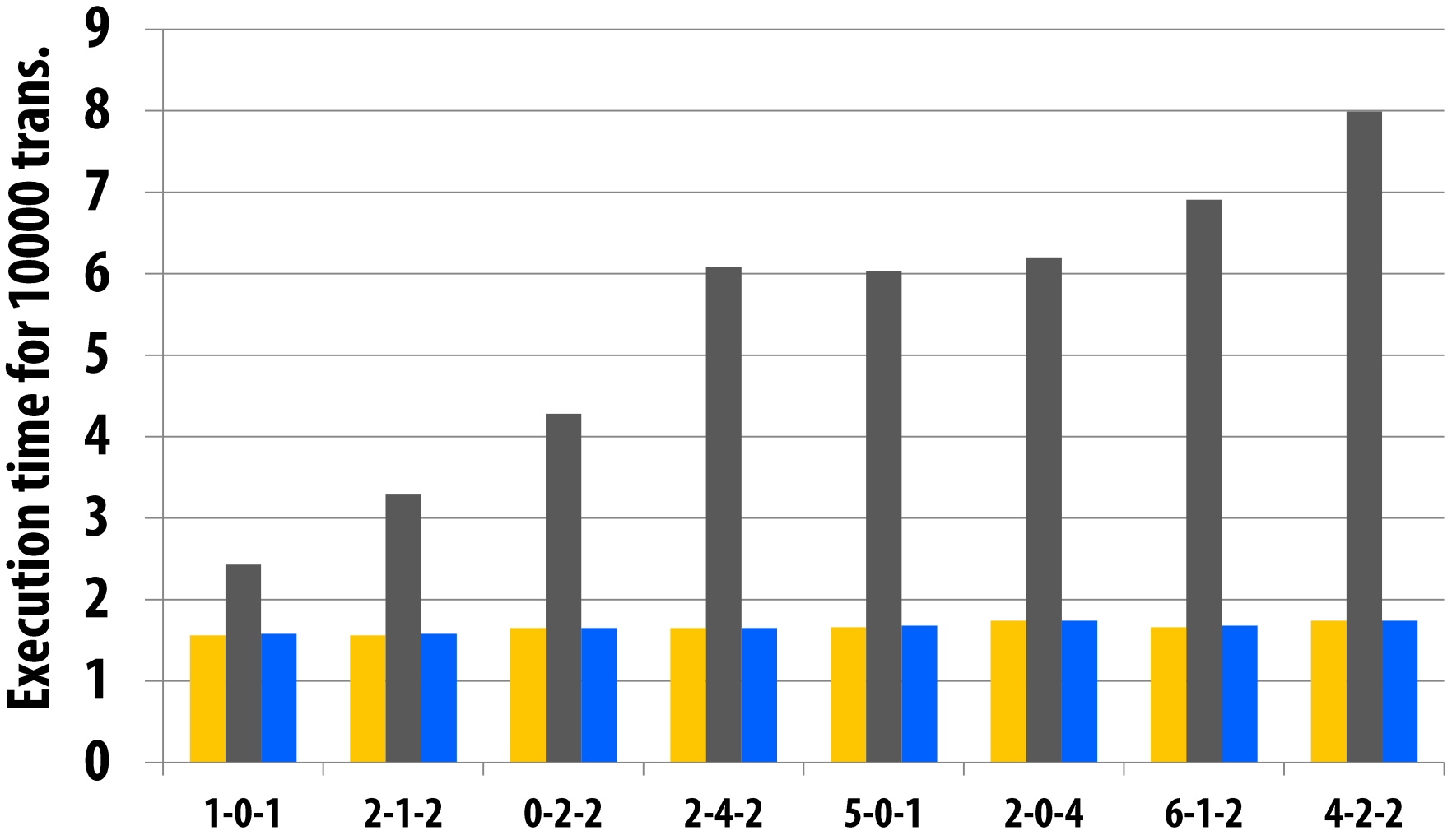
Presented in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Backup Slides

End-to-end system support for GS-DRAM



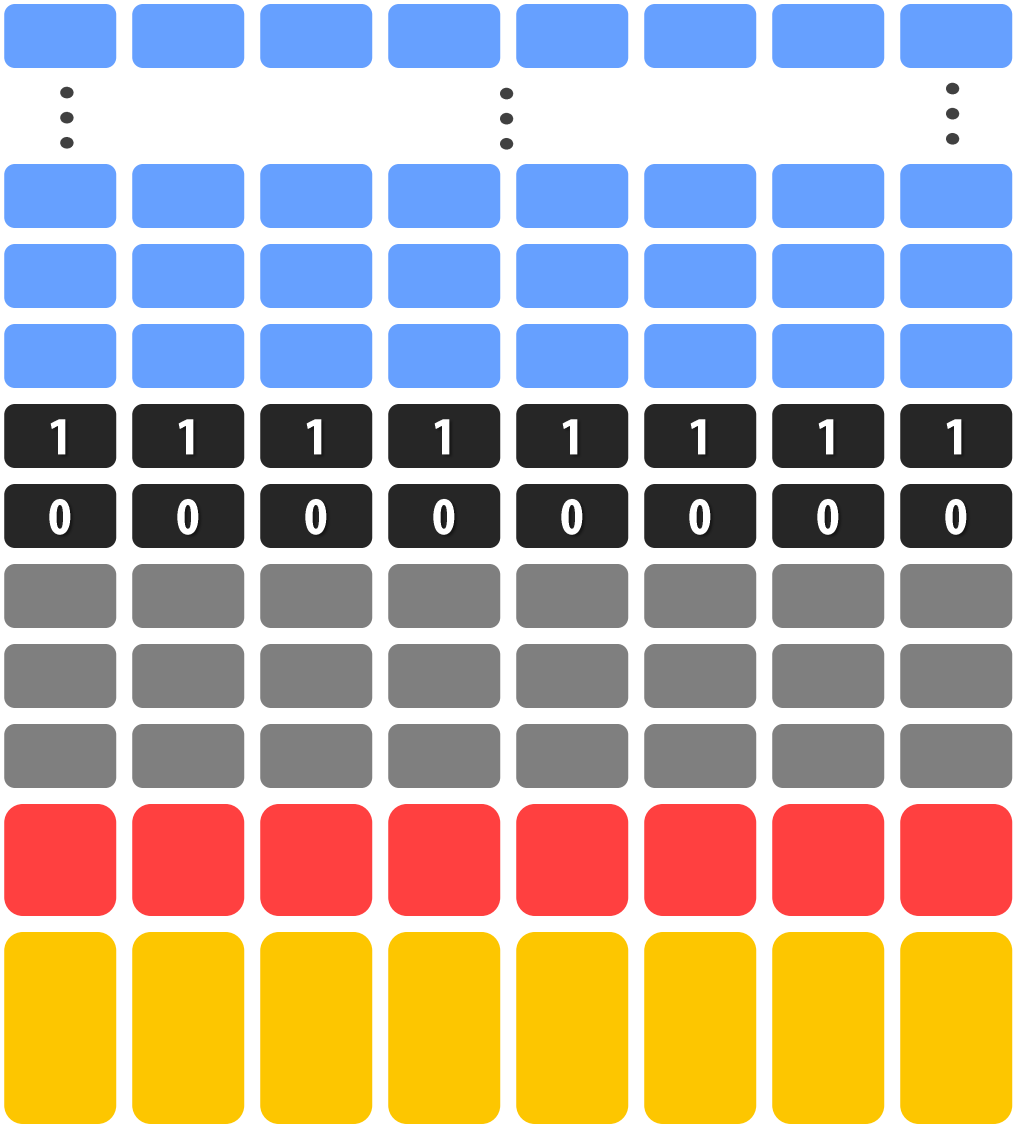
GS-DRAM improves transaction performance



GS-DRAM with Odd Stride

- Odd stride => minimal chip conflicts
  - No shuffling required
- Alignment problem
  - Objects do not fit perfectly into a DRAM row
  - Data may be part of different DRAM rows
- Addressing problem
  - A value may be part of different addresses in the same pattern
  - Difficult to maintain coherence

Buddy RAM – Implementation



Buddy RAM – Activate Activate Precharge (AAP)

- AAP copies result of first activation into row corresponding to second activation
  - Can be used to perform a simple copy
  - Bitwise operation if first activation correspond to triple row activation
- Example: Bitwise AND
  - AAP(d1, t1) -> copy row d1 to temporary row t1
  - AAP(d2, t2) -> copy row d2 to temporary row t2
  - AAP(c0, t3) -> copy control row zero to temporary row t3
  - AAP(t1/t2/t3, d3) -> copy t1&t2 to d3

Bitmap Indices: Buddy improves performance