


 北京大学信息科学技术学院本科生课程

**计算机组成与
系统结构实习**

 **存储架构
缓存+主存**

北京大学高能效计算与应用中心

孙广宇

2017-11-20

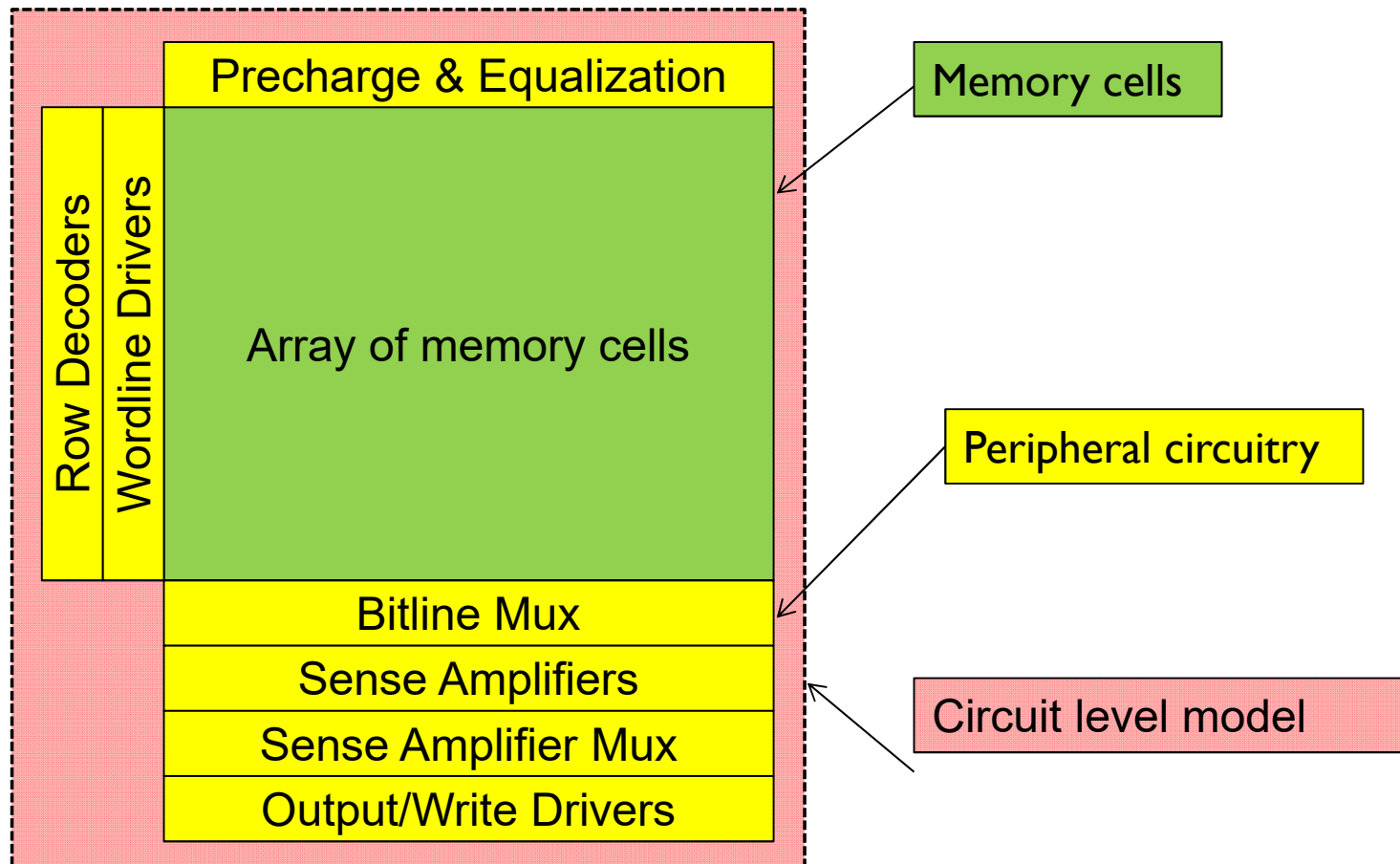
OUTLINE

- **Cache (SRAM)**
- **Main Memory (DRAM)**
- **Emerging Memory**

OUTLINE

- **Cache (SRAM)**
 - Low Level Cache Organization
 - Cache Content Management
- Main Memory
- Emerging Technologies

ABSTRACT OF A SRAM DESIGN



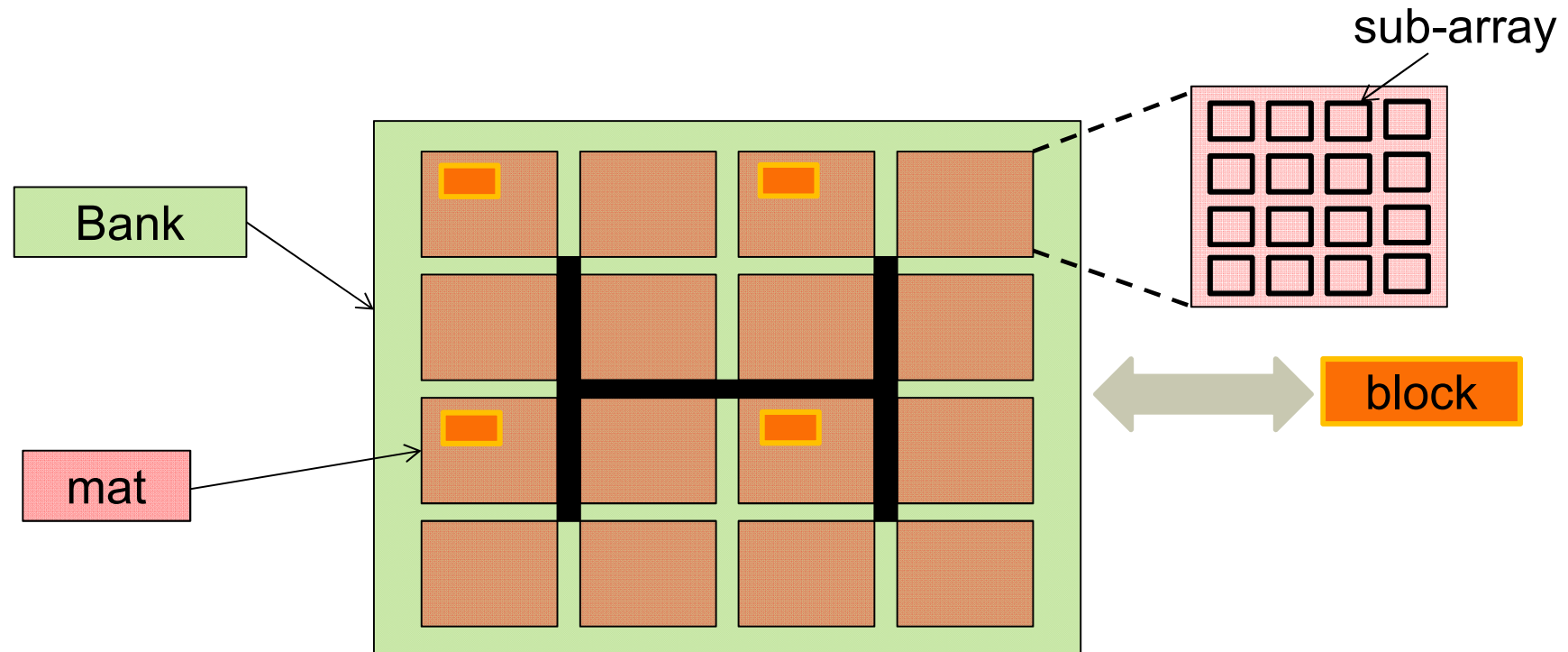
RAM ORGANIZATION

bank: fully-functional memory unit

sub-array: elementary unit in mat

mat: simultaneously operated building block in a bank

block: data storage unit



APPROXIMATE RELATIONSHIP

● Terminology

- Area: A
- Capacity: C
- Latency: T

● Some approximation

- $A \propto C$ (Tag's density is lower, consumes 10%~20% area)
- $T \propto C^{1/a}$ (cache access can also be pipelined)
- Cell latency is dominating for small SRAM (low capacity)
- Wire latency is dominating for large SRAM (high capacity)
- Density is more important for large RAM design

CACTI: AN OPEN SOURCE PLATFORM

<http://quid.hpl.hp.com:9081/cacti/>

CACTI 5.3

Normal Interface	Cache Size (bytes)	<input type="text"/>
Detailed Interface	Line Size (bytes)	<input type="text"/>
Pure RAM Interface	Associativity	<input type="text"/>
FAQ	Nr. of Banks	<input type="text"/>
	Technology Node (nm)	<input type="text"/>
	<input type="button" value="Submit"/>	

CACTI is not that accurate, but we will use it in our lab 😊

OUTLINE

- **Cache (SRAM)**
 - Low Level Cache Organization
 - Cache Content Management
- Main Memory (DRAM)
- Emerging Memory

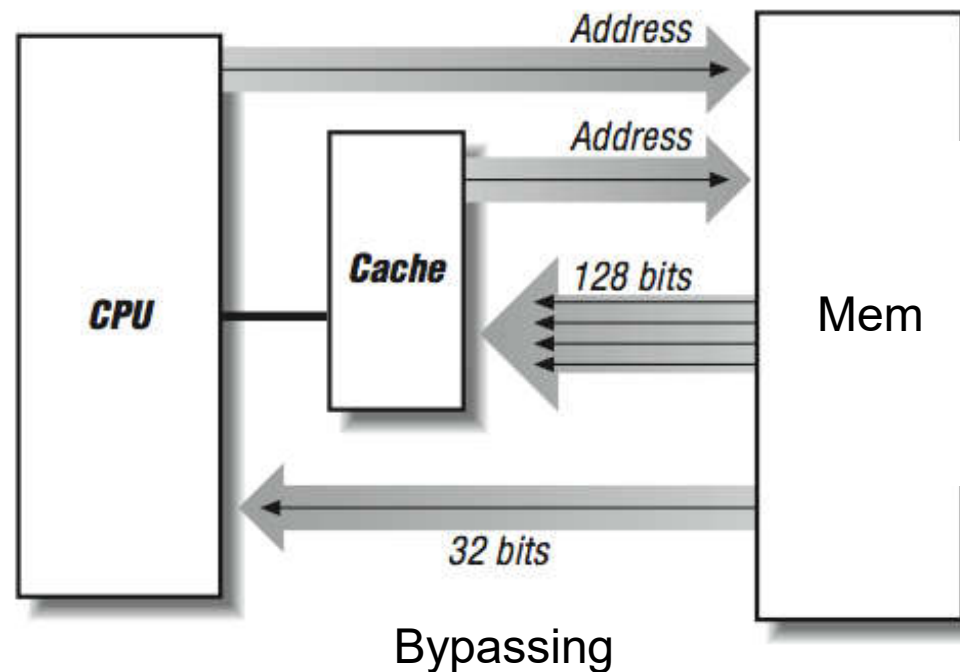
CACHE ACCESS TIME = HIT TIME + MISS RATE * MISS PENALTY

Goal: select proper data to cache

- In other words: some data **bypass** the cache
- Based on what metrics?
 - Reuse distance
 - Reuse count
 - Miss rate
 - Private data
 - Congestion
 - ...
- Prediction is needed
 - Based on single operation
 - Based on statistical operations

CACHE BYPASSING

- **Basic idea**
 - Bypass a block to upper levels if it is predicted to be less useful than the blocks currently in the cache.



APPLICATION

- 应用场景：GPU的广泛使用
- GPU cache 由于GPU 大规模多线程的特性，导致局部性很差。
- GPU cache同样有抖动出现，这些thrashing是由于warp之间竞争引起的，当一些warp被调度到同一个SIMT核时，就造成了thrashing的出现。由于应用的工作集通常比cache大小大很多，一些高级的替换策略也没办法解决GPU的这个问题。
- GPU同时有成百上千线程执行，每个线程分到的cache资源更少，CPU cache line生命周期更短，很多cache line在重用之前就被替换。
- 为防止有用的cache line被过早的evict出去，选择性地让某些内存请求绕过cache，从而为其他请求省下了cache空间，从而达到减少竞争的目的。
- 坏处：会有很多cache miss发生，从而影响性能。

Duong N, Zhao D, Kim T, et al. Improving Cache Management Policies Using Dynamic Reuse Distances[C]// Ieee/acm International Symposium on Microarchitecture. IEEE, 2012:389-400.

SOME ISSUES OF CACHE BYPASSING

- **Performance and Energy Benefits**
 - Frequent eviction for poor-locality applications.
 - Improve hit rate for some replacement policies (e.g. RR).
- **Challenges for CBT**
 - Implementation Overhead
 - Memory Bandwidth & Performance Overhead
 - Challenges in inclusive caches

A SIMPLE EXAMPLE (RELATED WITH LAB 3.2)

Cache bypassing based on cache miss behavior

- Two types of Cache miss
 - Conflict miss
 - Capacity miss (should bypass the cache)
- How to detect data causing capacity miss?
 - Store the Tag of most recently evicted data for each set
 - Check the Tag when the next cache miss happens
 - If the tag of new coming data matches the Tag stored, it is considered as a conflict miss, otherwise it is considered as capacity miss
 - Such conflict misses can be avoid by increasing associativity by 1
- and places data in a bypass. Bypasses any capacity miss buffer.

REPLACEMENT POLICY

$$\text{CACHE ACCESS TIME} = \text{HIT TIME} + \text{MISS RATE} * \text{MISS PENALTY}$$

What we have learned?

- Random replacement
- LRU replacement (is it practical?)
- FIFO

Some other options”

- LRU (Least Recently Used), pseudo
- LFU (Least Frequently Used)
- NMRU (Not Most Recently Used)
- NRU (Not Recently Used)
- Pseudo
- Optimal
- ...

Q: What is an OPT replacement policy?

A SIMPLE EXAMPLE

A Cache with only 4 blocks, full associative

For example, the visit sequence is as followed:

3 2 3 1 2 0 3 1

If use LRU,

the replaced block is 2.

If use LFU,

the replaced block is 0.

Entry 0
Entry 1
:
Entry 3

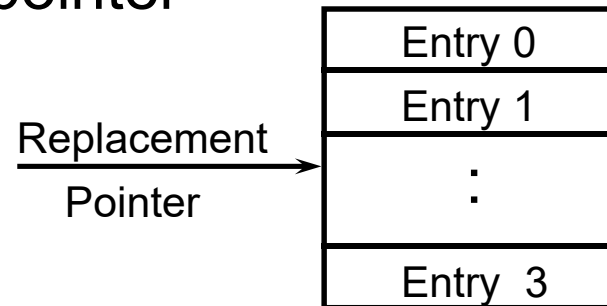
TRUE LRU SHORTCOMINGS

- Streaming data/scans: x_0, x_1, \dots, x_n
 - Effectively no temporal reuse
- Thrashing: reuse distance $> a$
 - Temporal reuse exists but LRU fails
- All blocks march from MRU to LRU
 - Other conflicting blocks are pushed out
- For $n > a$ no blocks remain after scan/thrash
 - Incur many conflict misses after scan ends
- Pseudo-LRU sometimes helps a little bit

A SIMPLE PSEUDO LRU

Example of a Simple “Pseudo” Least Recently Used Implementation:

- Assume 4 Fully Associative Entries
- Hardware replacement pointer points to one cache entry
- Whenever an access is made to the entry the pointer points to:
 - Move the pointer to the next entry
- Otherwise: do not move the pointer



BINARY TREE BASED PSEUDO LRU

- Assume 4 Fully Associative Entries
- Rather than true LRU, use binary tree
- Each line has 3-bit($N-1$, N : the number of ways) status bit, named PLRU bit
- Initial, all PLRU bits are set 0. For example, $B_0B_1B_2=000$
- If there is one block is invalid, replace it.
else choose B1 or B2 according B0.
- ...
- Replace the line which is selected.

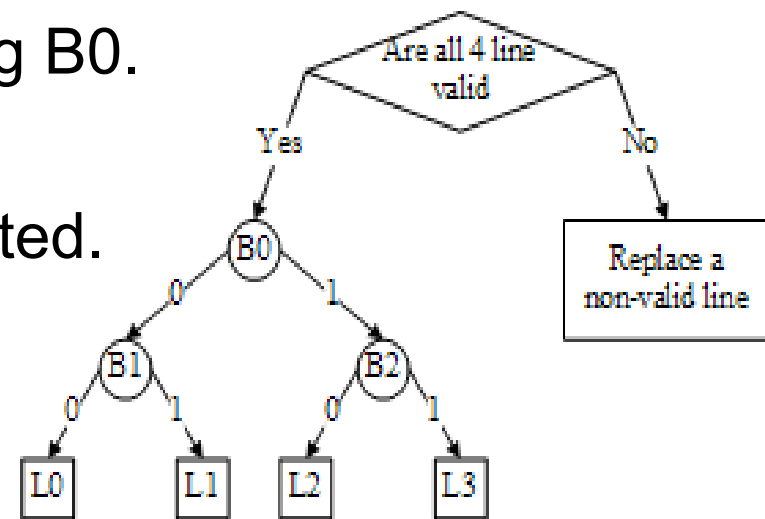


图 2-15 Tree-Based PLRU 替换算法

PLRU BITS UPDATE RULES

- When evicted the cache line, the PLRU bits are updated
- When the cache line is hit, the PLRU bits are also updated
- The update rules are:

Current Access	New State of the PLRU Bits		
	B0	B1	B2
L0	1	1	No Change
L1	1	0	No Change
L2	0	No Change	1
L3	0	No Change	0

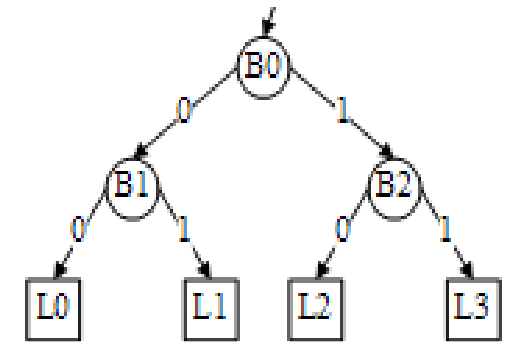


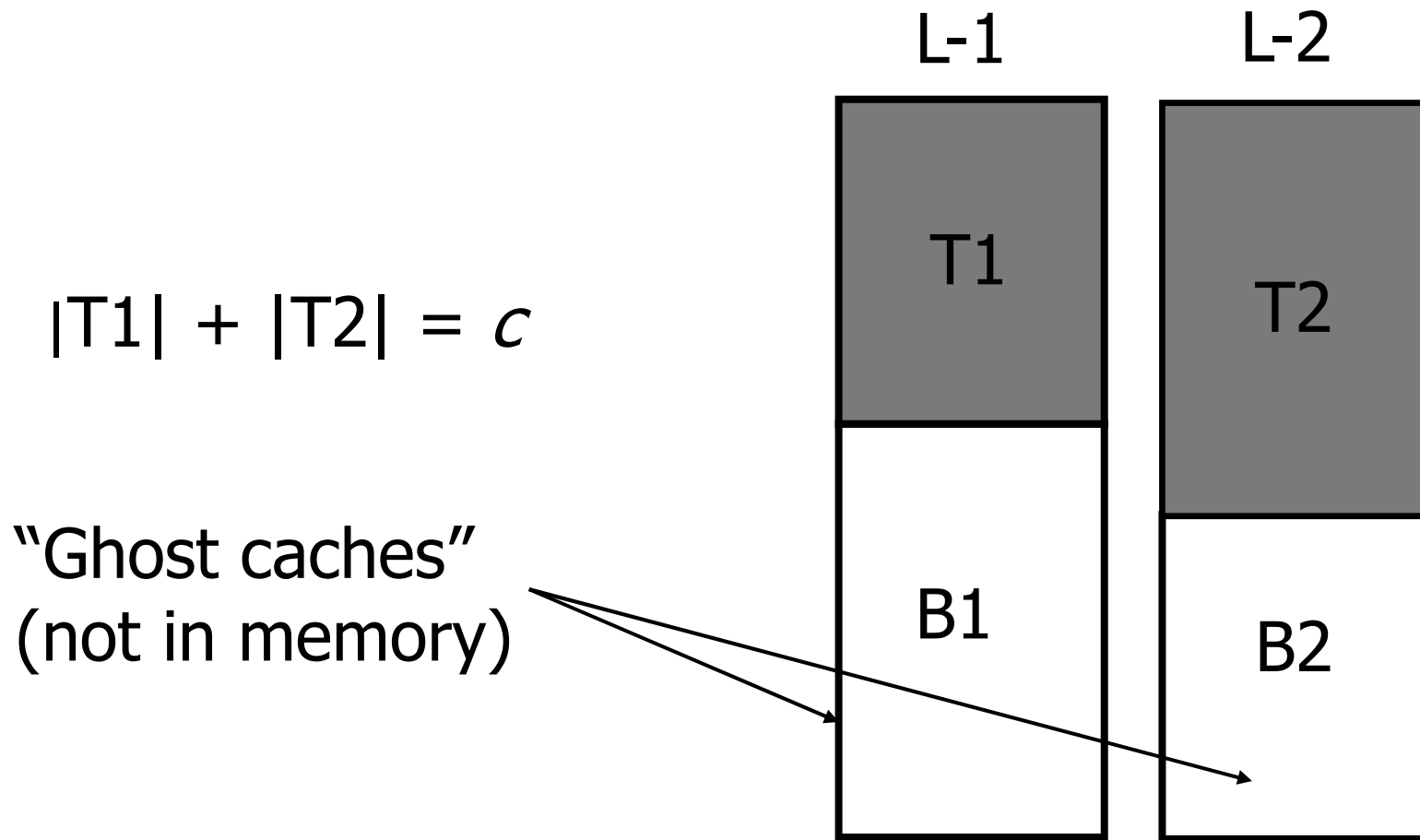
图 2-15 Tree-Based PLRU 替

Q: 3 2 3 1 2 0 3 1

ARC: ADAPTIVE REPLACEMENT CACHE

- **Basic idea**
 - Keeping track of both frequently used and recently used pages plus a recent eviction history for both.
- **Maintains two LRU lists**
 - Data that have been referenced only once (L1)
 - Data that have been referenced at least twice (L2)
- **Cache contains tops of both lists: T1 and T2**
 - Each list has same length c as cache
 - Bottoms B1 and B2 are not in cache

ARC: ADAPTIVE REPLACEMENT CACHE

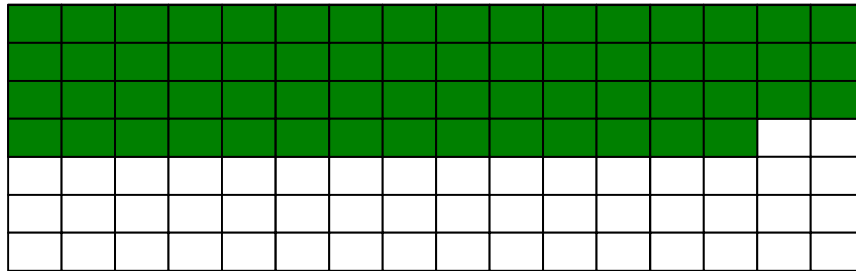


ONLINE REPLACEMENT POLICIES

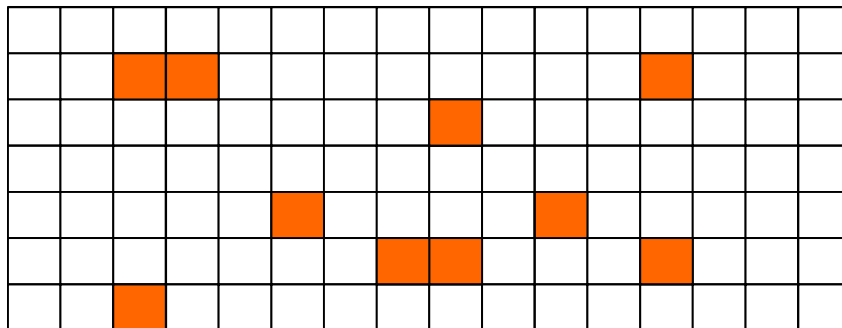
- **Other important policies**
 - SLRU, LFU, LFUDA, LIRS, ARC, CAR, MQ, ...
 - Published in MICRO, ISCA, PLDI, FAST, SIGMOD
 - ...
 - https://en.wikipedia.org/wiki/Cache_replacement_policies
- **Design considerations**
 - Hardware overhead
 - Latency
 - Miss rate
 - Power consumption
 - ...

CACHE PREFETCHING

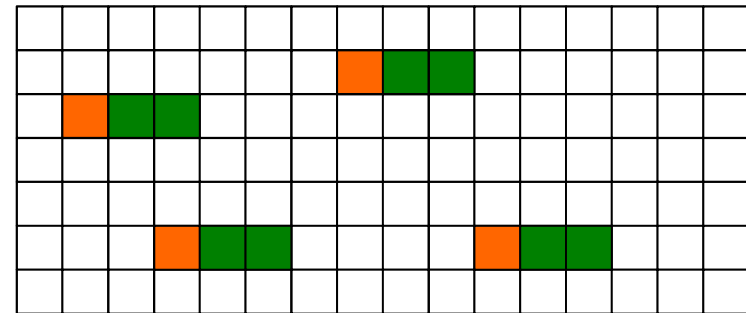
$$\text{CACHE ACCESS TIME} = \text{HIT TIME} + \text{MISS RATE} * \text{MISS PENALTY}$$



■ = Regular Data Access



■ = Irregular Data Access



■ =
Regular
Data
Access

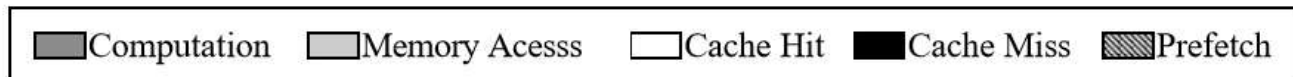
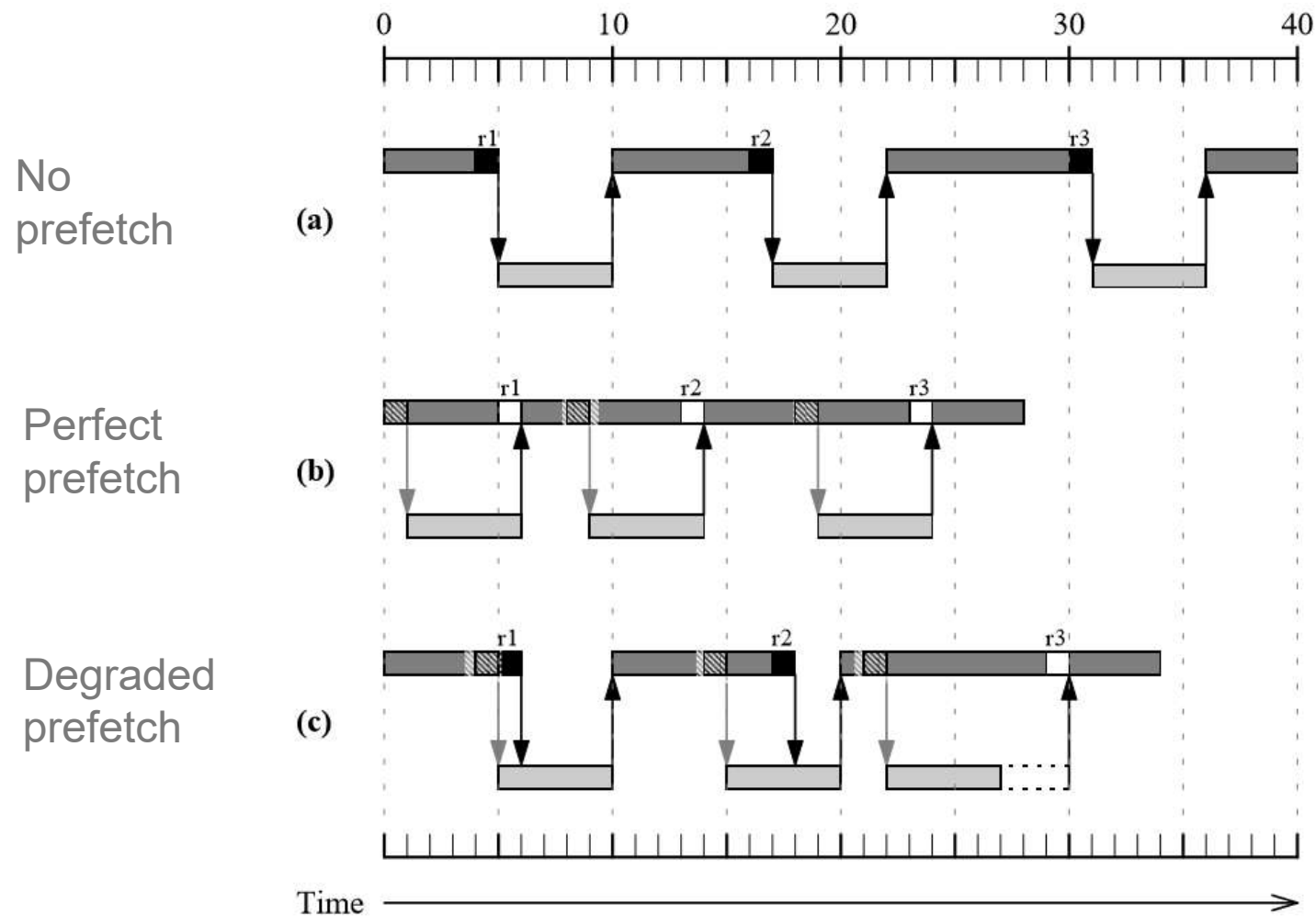
■ =
Irregular
Data
Access

CACHE PREFETCHING

$$\text{CACHE ACCESS TIME} = \text{HIT TIME} + \text{MISS RATE} * \text{MISS PENALTY}$$

- **Basic idea**
 - Fetch data from their original storage in slower memory to a faster local memory **before they are actually needed**.
 - Explore spatial locality & Reduce cache miss
 - Hardware & Software approaches
 - Prefetch instruction
 - Prefetch mechanism
- **How to measure a prefetch algorithm?**
 - **Coverage**: the fraction of total misses that are eliminated because of prefetching
 - **Accuracy**: the fraction of total prefetches that were useful
 - **Timeliness**: how early a block is prefetched versus when it is actually referenced

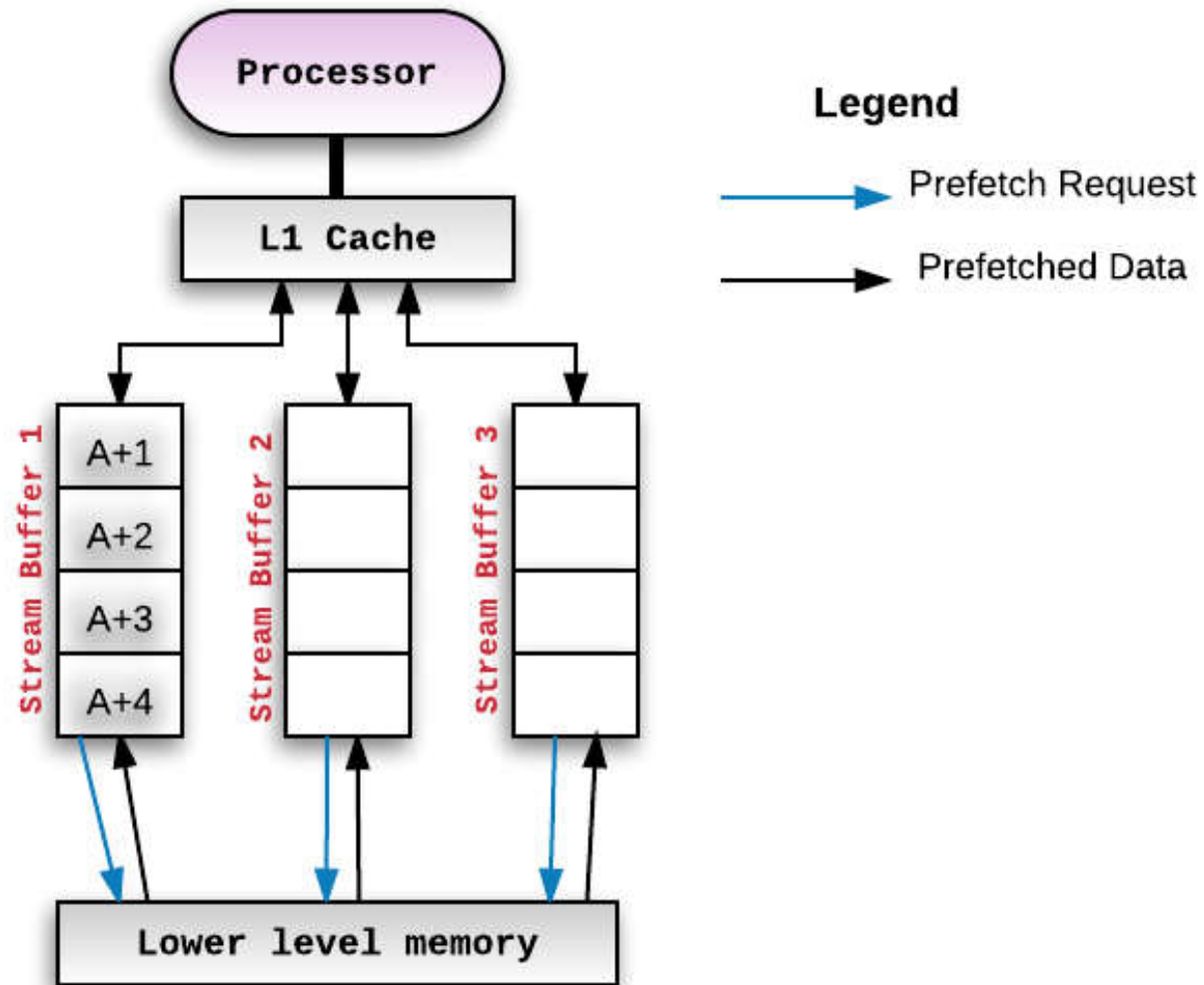
PREFETCH OVERHEAD



STREAM BUFFERS

- **Stream buffer: The most common technique in use**
 - Detect special patterns (e.g. constant-stride access)!
 - The cache miss address (and k subsequent addresses) are fetched into a separate buffer of depth k .
 - The processor then consumes data or instructions from the **stream buffer** if the address matches.

STREAM BUFFERS



PITFALL: CACHE FILTERING EFFECT

- Upper level caches (L1, L2) hide reference stream from lower level caches
- Blocks with “no reuse” @ LLC could be very hot (never evicted from L1/L2)
- Evicting from LLC often causes L1/L2 eviction (due to inclusion)
- Could hurt performance even if LLC miss rate improves

OUTLINE

- Cache Advanced Topics
- **Main Memory**
 - A detailed view
 - Design considerations
- Emerging Technologies

A DETAILED VIEW OF AN ARRAY

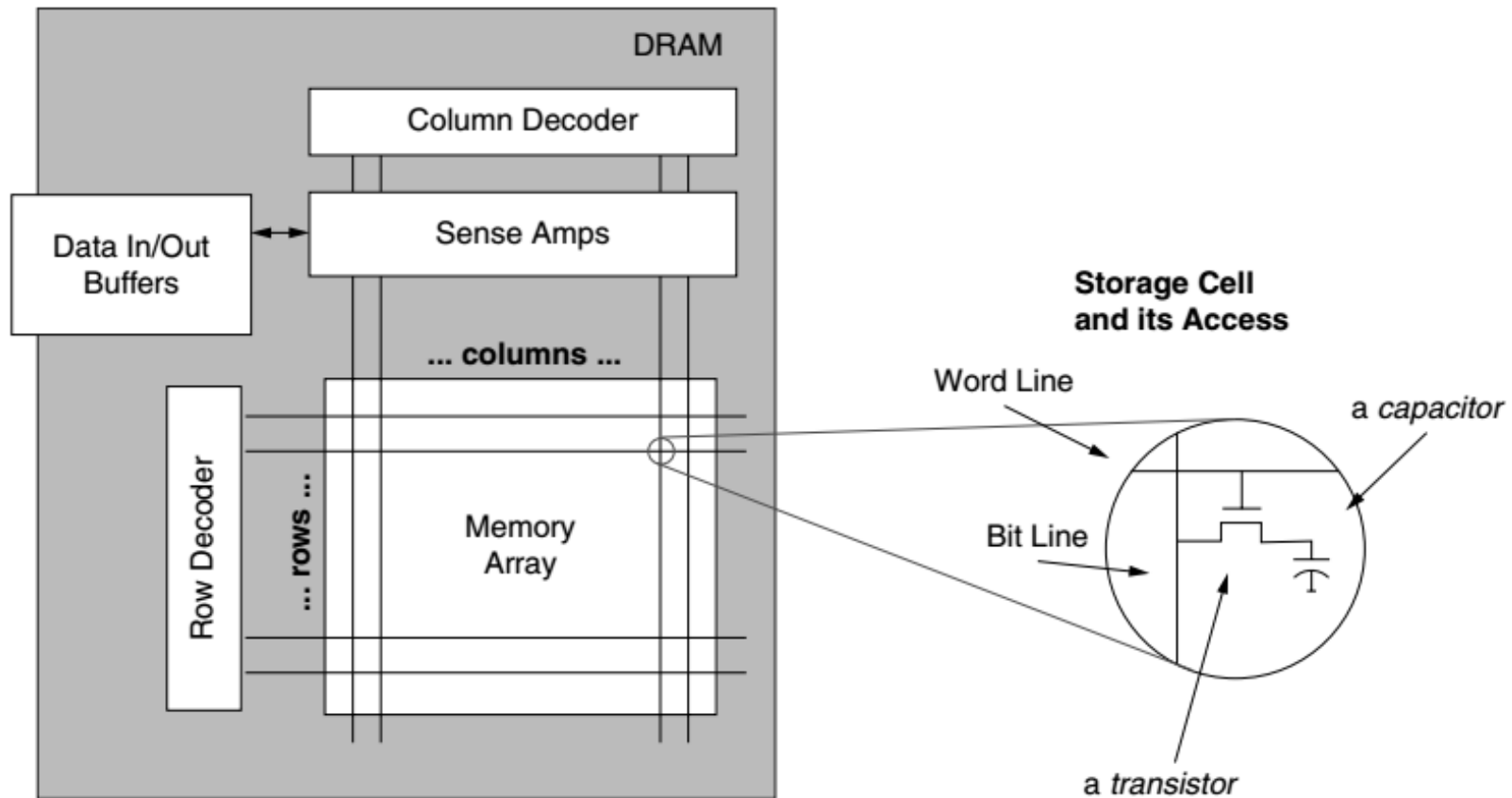


FIGURE 7.3: Basic organization of DRAM internals. The DRAM memory array is a grid of storage cells, where one bit of data is stored at each intersection of a *row* and a *column*.

ACCESS FLOW OVERVIEW

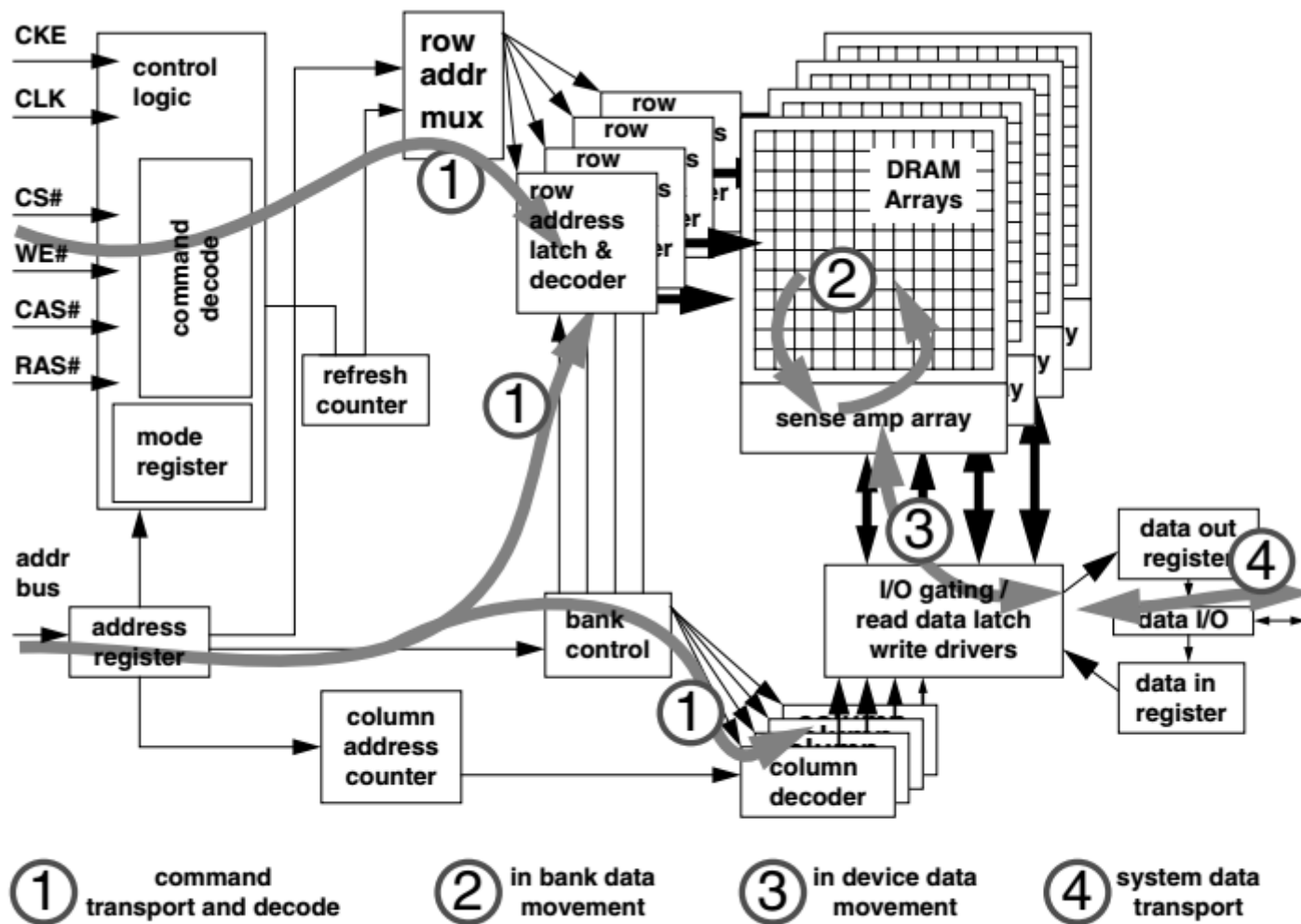


FIGURE 11.1: Command and data movement on a generic SDRAM device.

DRAM DESIGN CONSIDERATIONS

Goal

- High Capacity
- High throughput, low latency,
- Low power
- High reliability

Management:

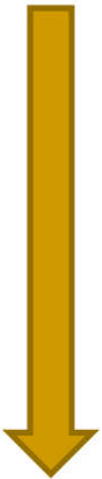
- Address mapping
- Requests scheduling
- Page open/close
- Refreshing
- ECC

THREE PRINCIPLES OF MEMORY SCHEDULING

Prioritize row-buffer-hit requests[Rixner+, ISCA'00]

- To maximize memory bandwidth

Older



Newer

Req 1	Row A
Req 2	Row B
Req 3	Row C
Req 4	Row A
Req 5	Row B

Currently open row
B


THREE PRINCIPLES OF MEMORY SCHEDULING

Prioritize row-buffer-hit requests[Rixner+, ISCA'00]

- To maximize memory bandwidth

Prioritize latency-sensitive applications[Kim+, HPCA'10]

Application	Memory Intensity (MPKI)
1	5
2	1
3	2
4	10



THREE PRINCIPLES OF MEMORY SCHEDULING

Prioritize row-buffer-hit requests [Rixner+, ISCA'00]

- To maximize memory bandwidth

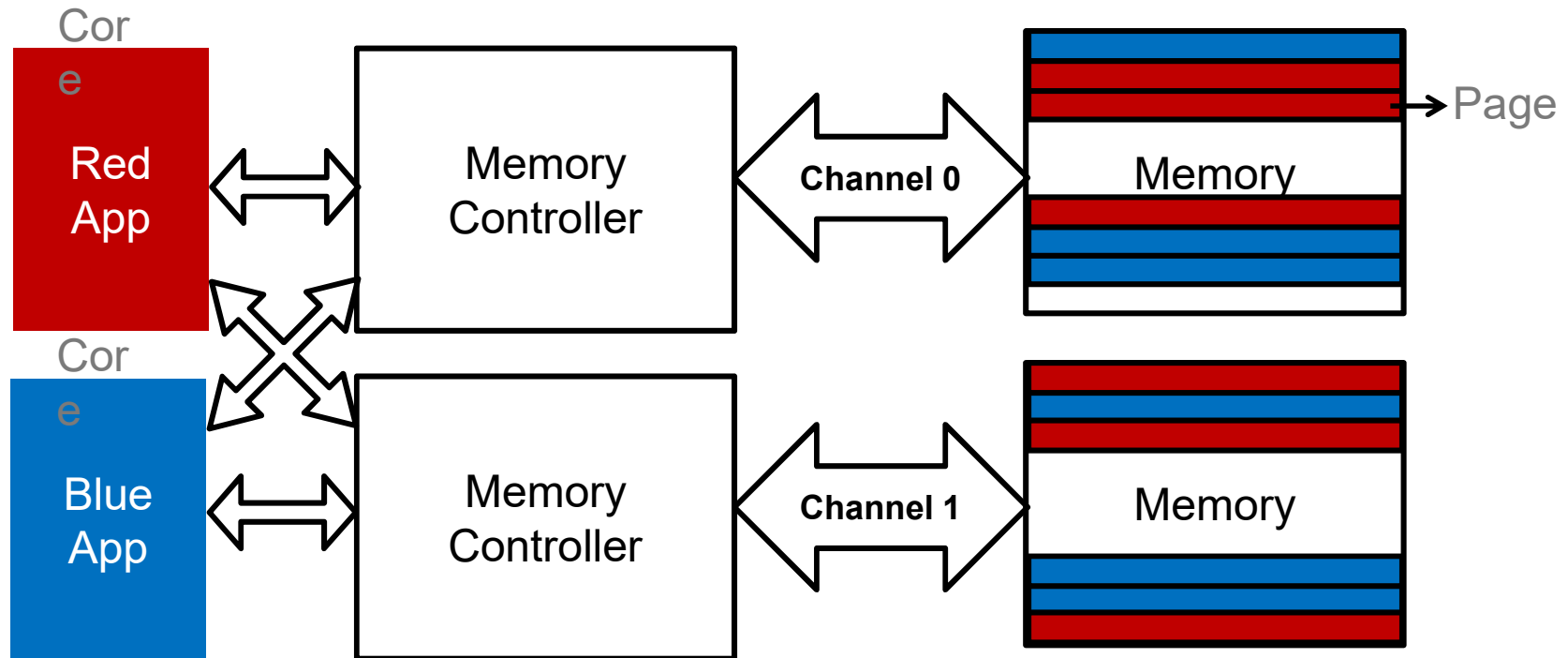
Prioritize latency-sensitive applications [Kim+, HPCA'10]

- To maximize system throughput

Ensure that no application is starved [Mutlu+, MICRO'07]

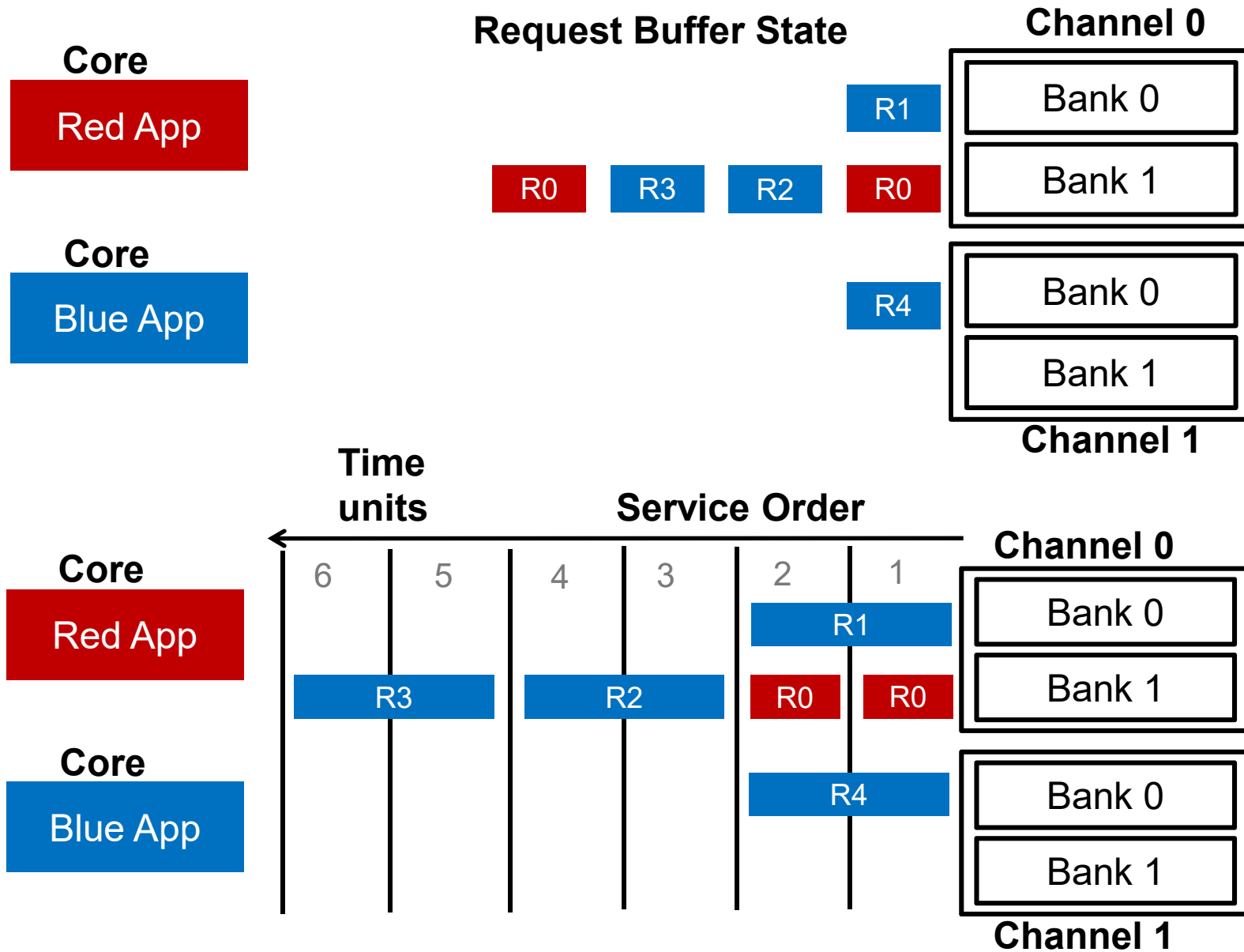
- To minimize unfairness

MODERN SYSTEMS HAVE MULTIPLE CHANNELS

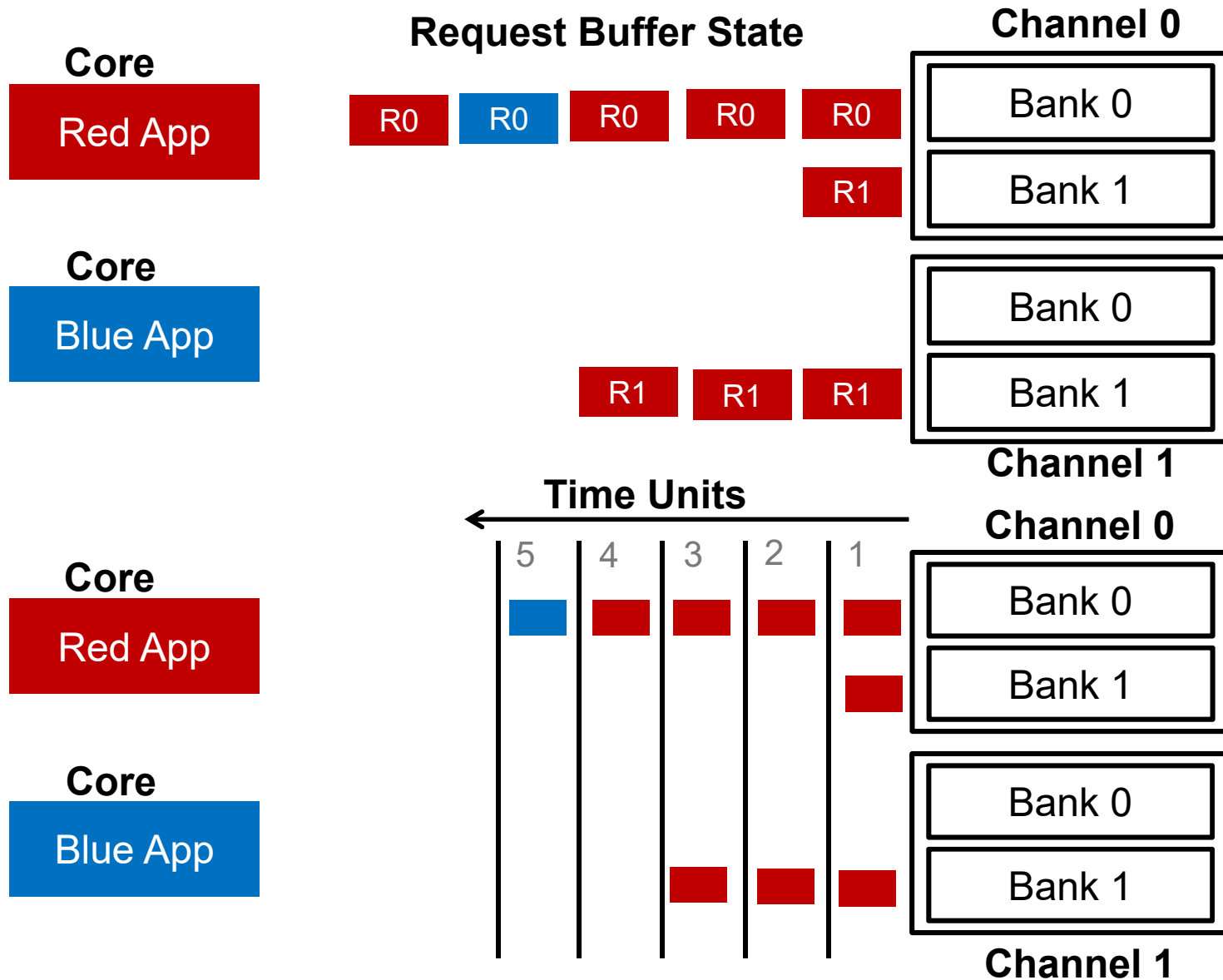


- **Memory Controller**
- **Channel**
- **page**

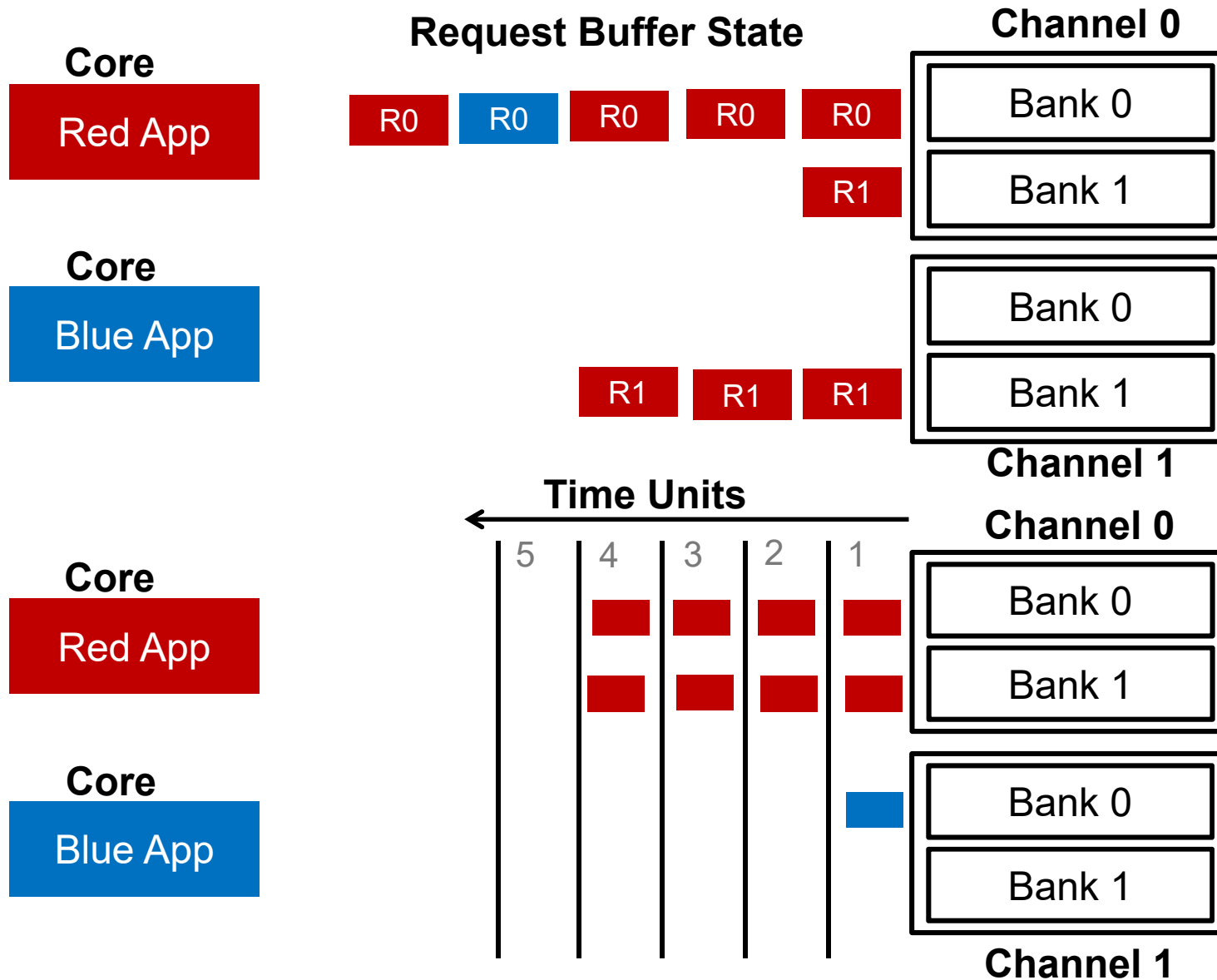
PRIORITIZE ROW-BUFFER-HIT REQUESTS



PRIORITIZE LATENCY-SENSITIVE APPLICATIONS



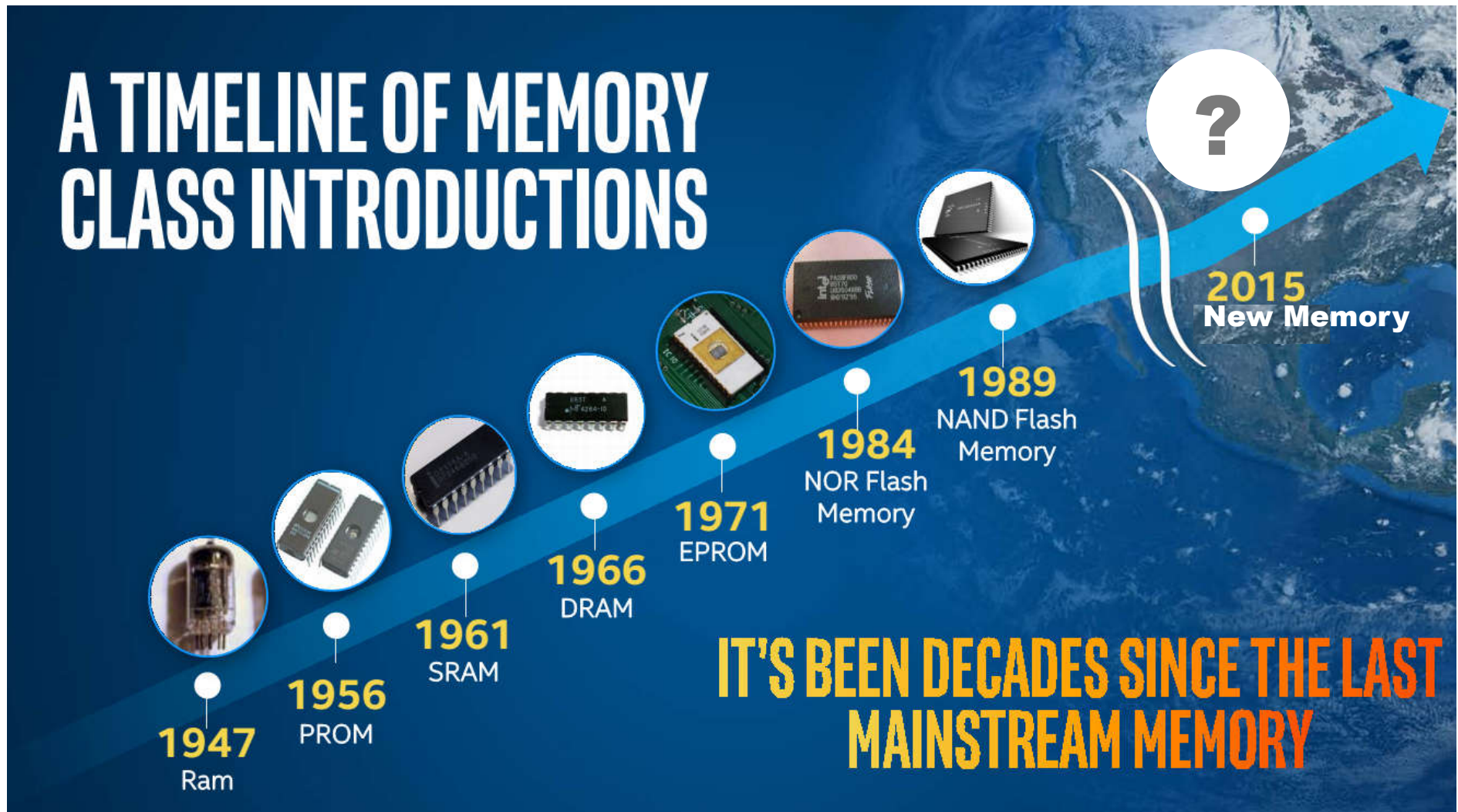
PRIORITIZE LATENCY-SENSITIVE APPLICATIONS



OUTLINE

- Cache Advanced Topics
- Main Memory
- **Emerging Memory**

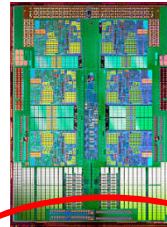
MEMORY TECHNOLOGY TIMELINE



Source: Intel

TRADITIONAL MEMORY HIERARCHIES

Need more
on-chip
memory



On-chip memory
(SRAM)

Latency:
(Cycles)

1~30

Bandwidth



Main memory
(DRAM)

100~300



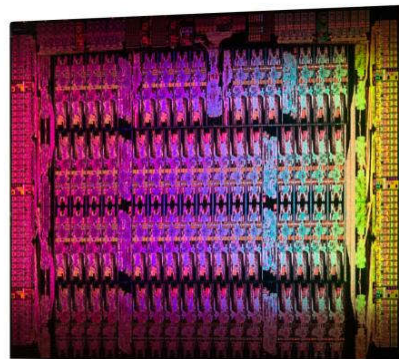
Solid State Disk
(Flash Memory)

25000~2000000



HDD

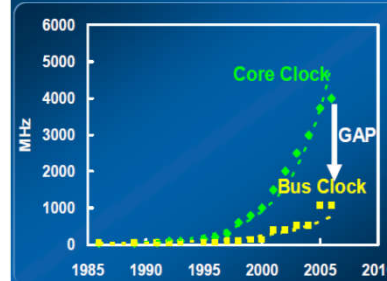
>5000000



Intel Xeon Phi (60-core)

Source: S. Borkar

Memory BW Gap



Busses have become
wider to deliver
necessary memory
BW (10 to 30 GB/sec)

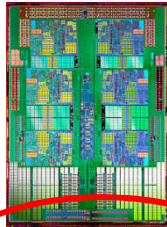
Yet, memory BW is not
enough

Many Core System will
demand 100 GB/sec
memory BW

How do you feed the beast?

TRADITIONAL MEMORY HIERARCHIES

Need more
on-chip
memory



On-chip memory
(SRAM)

Latency:
(Cycles)

1~30



Main memory
(DRAM)

100~300



Solid State Disk
(Flash Memory)

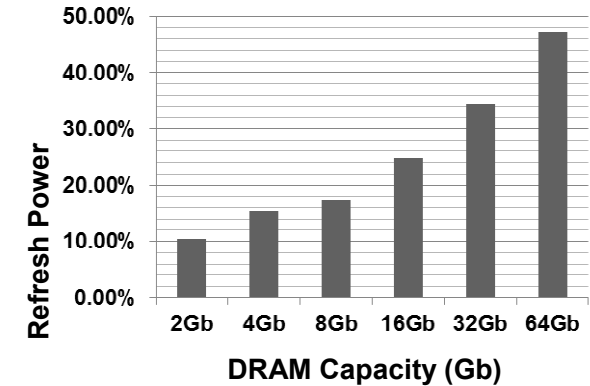
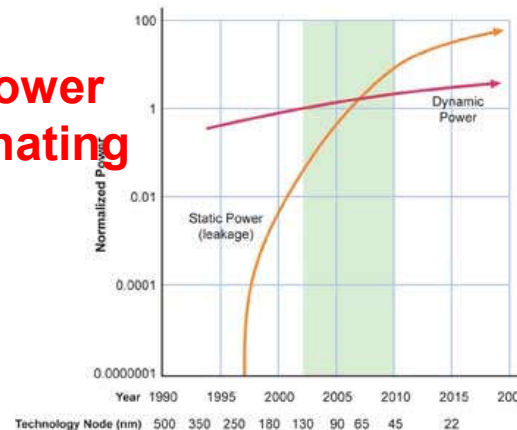
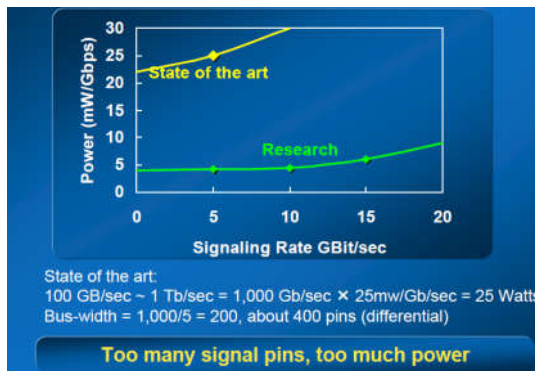
25000~2000000



HDD

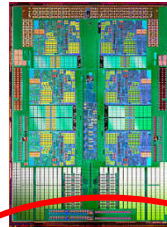
>5000000

Static power
is dominating



TRADITIONAL MEMORY HIERARCHIES

Need more
on-chip
memory



On-chip memory
(SRAM)



Main memory
(DRAM)



Solid State Disk
(Flash Memory)



HDD

Latency:
(Cycles)

1~30

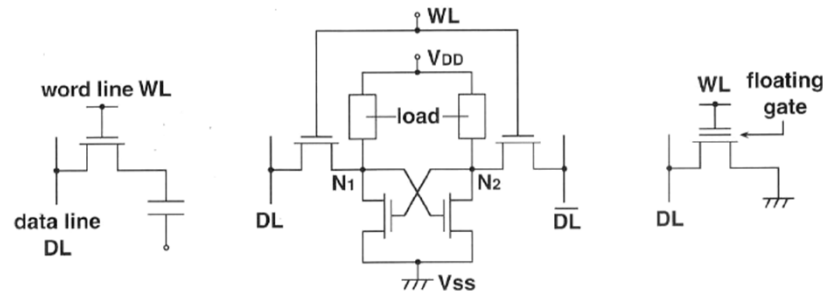
100~300

25000~2000000

>5000000

**Static power
is dominating**

Charge memory: scaling challenges



EMERGING NON-VOLATILE MEMORIES

Magnetic RAM (MRAM)

- EverSpin (130nm, up to 16Mb)

Spin-Transfer Torque RAM (STTRAM)

- Grandis (54nm, acquired by Samsung)

Phase-Change RAM (PCRAM)

- Samsung (20nm, diode, up to 8Gb)

Resistive RAM (RRAM)

- Panasonic (180nm process, 4-layer xpoint)
- Unity Semi(64MB, acquired by Rambus 2012.2))

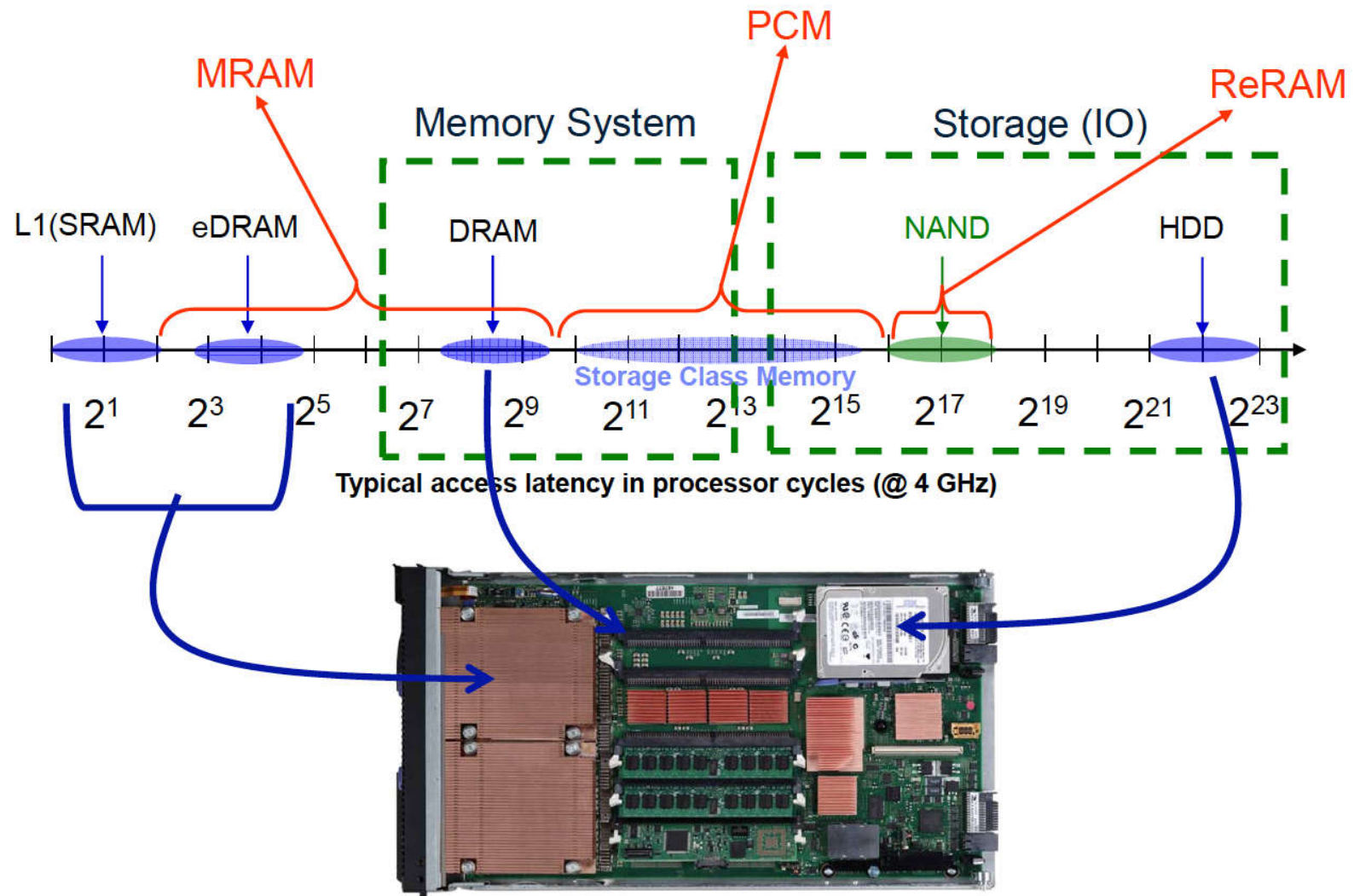


COMPARISON OF MEMORY TECHNOLOGIES

- ❑ **STT-MRAM**: Spin-Transfer-Torque Magnetic Random Access Memory
- ❑ **PCRAM**: Phase Change Random Access Memory
- ❑ **RRAM**: Resistive Radom Access Memory

	SRAM	DRAM	FLASH	STT-RAM	PCRAM	RRAM
cell area (F ²)	50-200	6	<4 if 3D	6-20	4-10	<4 if 3D
multi-bit	1	1	3	2	2	2
scalability	<20nm	<20nm	<20nm	~10nm	<20nm	<10nm
voltage	<1V	<1V	>20V	<2V	<3V	<3V
read speed	<1ns	~10ns	~10μs	~ns	~50ns	<10ns
energy/bit	~fJ	~pJ	~nJ	~0.1pJ	~10pJ	~0.1pJ
standby power	High	High	None	None	None	None
endurance	>1E16	>1E16	<1E5	>1E15	1E8-1E12	1E6-1E12
retention	volatile	~64ms	>10years	>10years	>10years	>10years
Vulnerability	High	High	Low	Low	Low	Low

OVERVIEW OF ADOPTION



Source : IBM

CHALLENGES

Hardware

- Controller
- Wear leveling/Reliability
- Allocation

Software

- OS/File System (Linux DAX extension, PMFS)
- Dedicated library (PMEM library)
- Data management
- Programming model

Thanks

OPTIMAL REPLACEMENT POLICY?

[Belady, IBM Systems Journal, 1966]

- **Evict block with longest reuse distance**
i.e. next reference to block is farthest in future
Requires knowledge of the future!
- **Can't build it, but can model it with trace**
Process trace in reverse
[Sugumar&Abraham] describe how to do this in
one pass over the trace with some lookahead
(Cheetah simulator)
- **Useful, since it reveals opportunity**
Normally used as the target for optimization

SEGMENTED OR PROTECTED LRU

- [I/O: Karedla, Love, Wherry, IEEE Computer 27(3), 1994]
- [Cache: Wilkerson, Wade, US Patent 6393525, 1999]
- Partition LRU list into **filter** and **reuse** lists
- On insert, block goes into filter list
- On reuse (hit), block promoted into reuse list
 - Hit = MRU?
- Provides scan & some thrash resistance
- Blocks without reuse get evicted quickly
- Blocks with reuse are protected from scan/thrash blocks
- No storage overhead, but LRU update slightly more complicated

OPTIMIZATION TECHNIQUES

Distance/latency aware placement

Data migration?

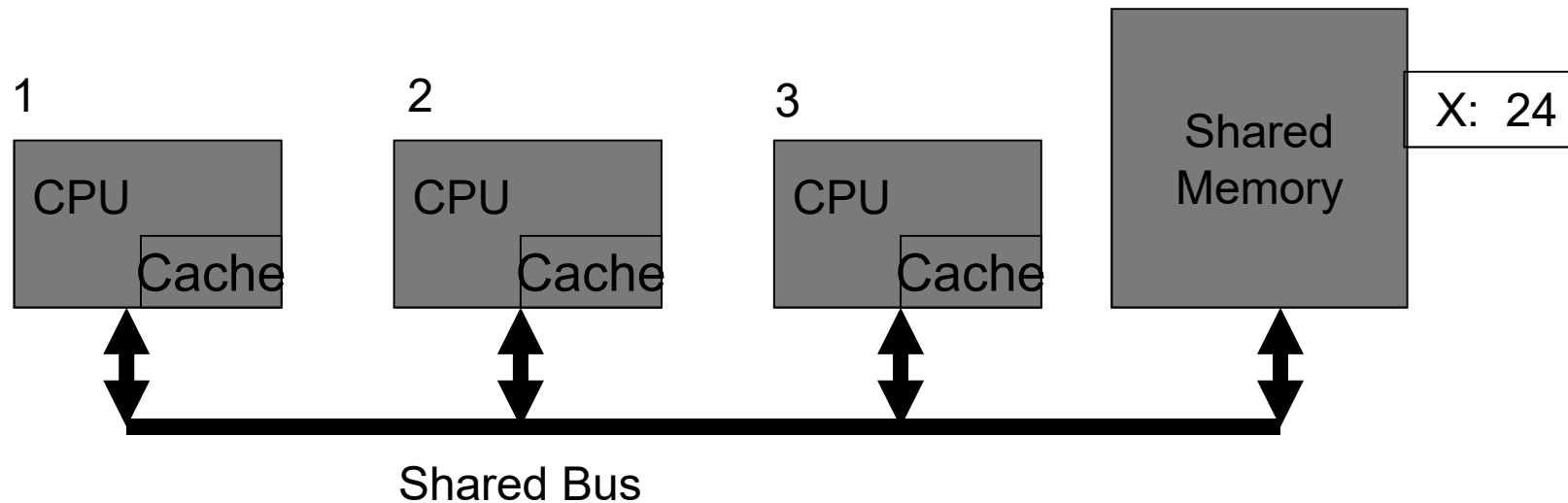
Data replication?

Q: What is the problem of data duplication?

OUTLINE

- **Cache (SRAM)**
 - Low Level Cache Organization
 - Cache Content Management
 - Cache Coherence Management
- Main Memory

EXAMPLE



Processor 1 reads X: obtains 24 from memory and caches it
Processor 2 reads X: obtains 24 from memory and caches it
Processor 1 writes 32 to X: its locally cached copy is updated
Processor 3 reads X: what value should it get?
Memory and processor 2 think it is 24
Processor 1 thinks it is 32

Notice that having write-through caches is not good enough

BUS SNOOPING

Scheme where every CPU knows who has a copy of its cached data is far too complex.

So each CPU (cache system) 'snoops' (i.e. watches continually) for write activity concerned with data addresses which it has cached.

This assumes a bus structure which is 'global', i.e all communication can be seen by all.

More scalable solution: 'directory based' coherence schemes

SNOOPING PROTOCOLS

Write Invalidate

- CPU wanting to write to an address, grabs a bus cycle and sends a 'write invalidate' message
- All snooping caches invalidate their copy of appropriate cache line
- CPU writes to its cached copy (assume for now that it also writes through to memory)
- Any shared read in other CPUs will now miss in cache and re-fetch new data.

SNOOPING PROTOCOLS

Write Update

- CPU wanting to write grabs bus cycle and broadcasts new data as it updates its own copy
- All snooping caches update their copy

Note that in both schemes, problem of simultaneous writes is taken care of by bus arbitration - only one CPU can use the bus at any one time.

UPDATE OR INVALIDATE?

Update looks the simplest, most obvious and fastest, but:

- Invalidate scheme is usually implemented with write-back cache:
 - Multiple writes to same word (no intervening read) need only one invalidate message but would require an update for each
 - Writes to same block in (usual) multi-word cache block require only one invalidate but would require multiple updates.

Due to both spatial and temporal locality, previous cases occur often

Bus bandwidth is a precious commodity in shared memory multi-processors

Experience has shown that invalidate protocols use significantly less bandwidth.

Will consider implementation details only of invalidate.

MESI PROTOCOL (1)

A practical multiprocessor invalidate protocol which attempts to minimize bus usage.

Allows usage of a 'write back' scheme - i.e. main memory not updated until 'dirty' cache line is displaced

Extension of usual cache tags, i.e. invalid tag and 'dirty' tag in normal write back cache.

MESI PROTOCOL (2)

Any cache line can be in one of 4 states (2 bits)

Modified - cache line has been modified, is different from main memory - is the only cached copy.
(multiprocessor 'dirty')

Exclusive - cache line is the same as main memory and is the only cached copy

Shared - Same as main memory but copies may exist in other caches.

Invalid - Line data is not valid (as in simple cache)

MESI PROTOCOL (3)

Cache line changes state as a function of memory access events.

Event may be either

- Due to local processor activity (i.e. cache access)
- Due to bus activity - as a result of snooping

Cache line has its own state affected only if address matches

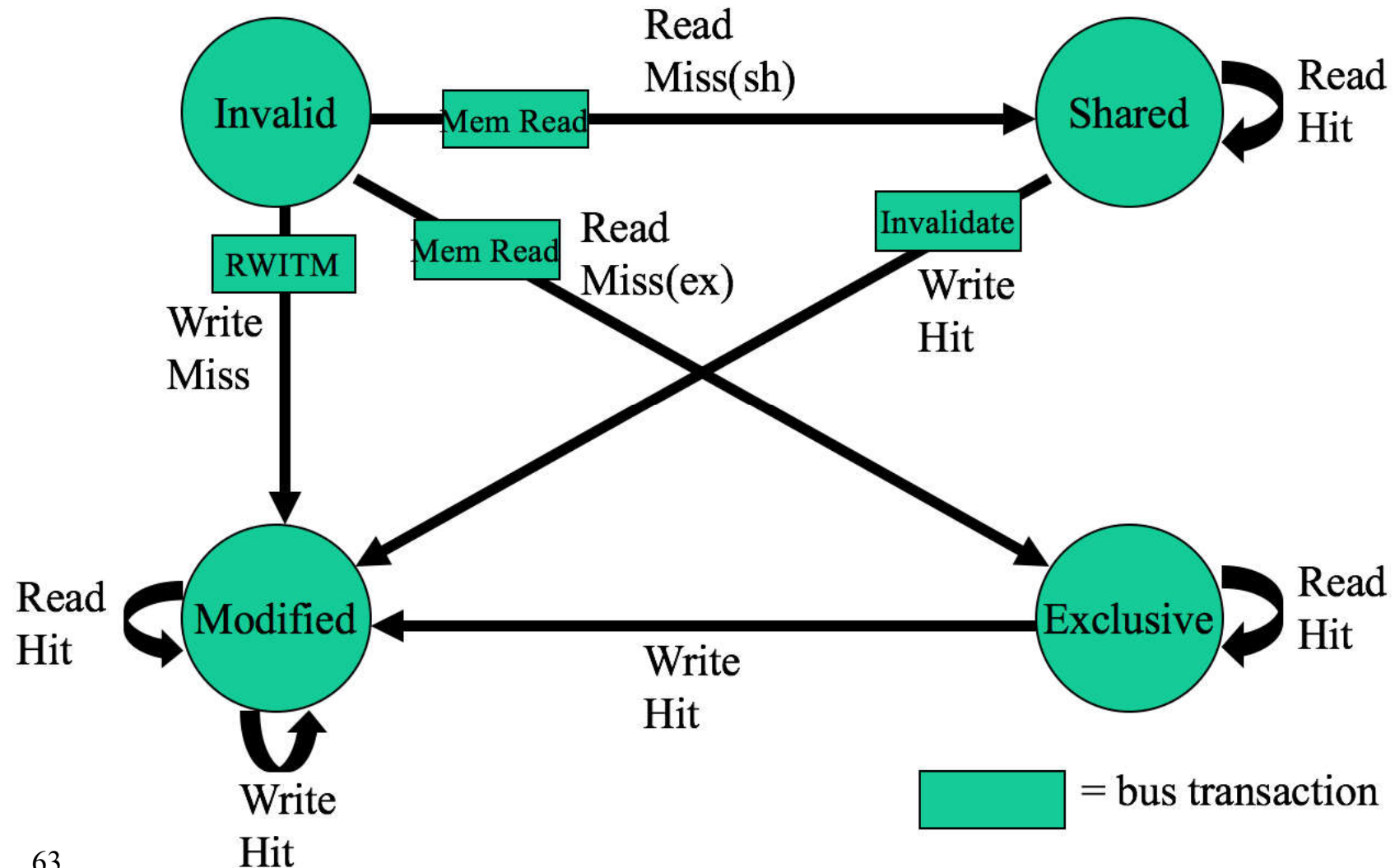
MESI PROTOCOL (4)

Operation can be described informally by looking at action in local processor

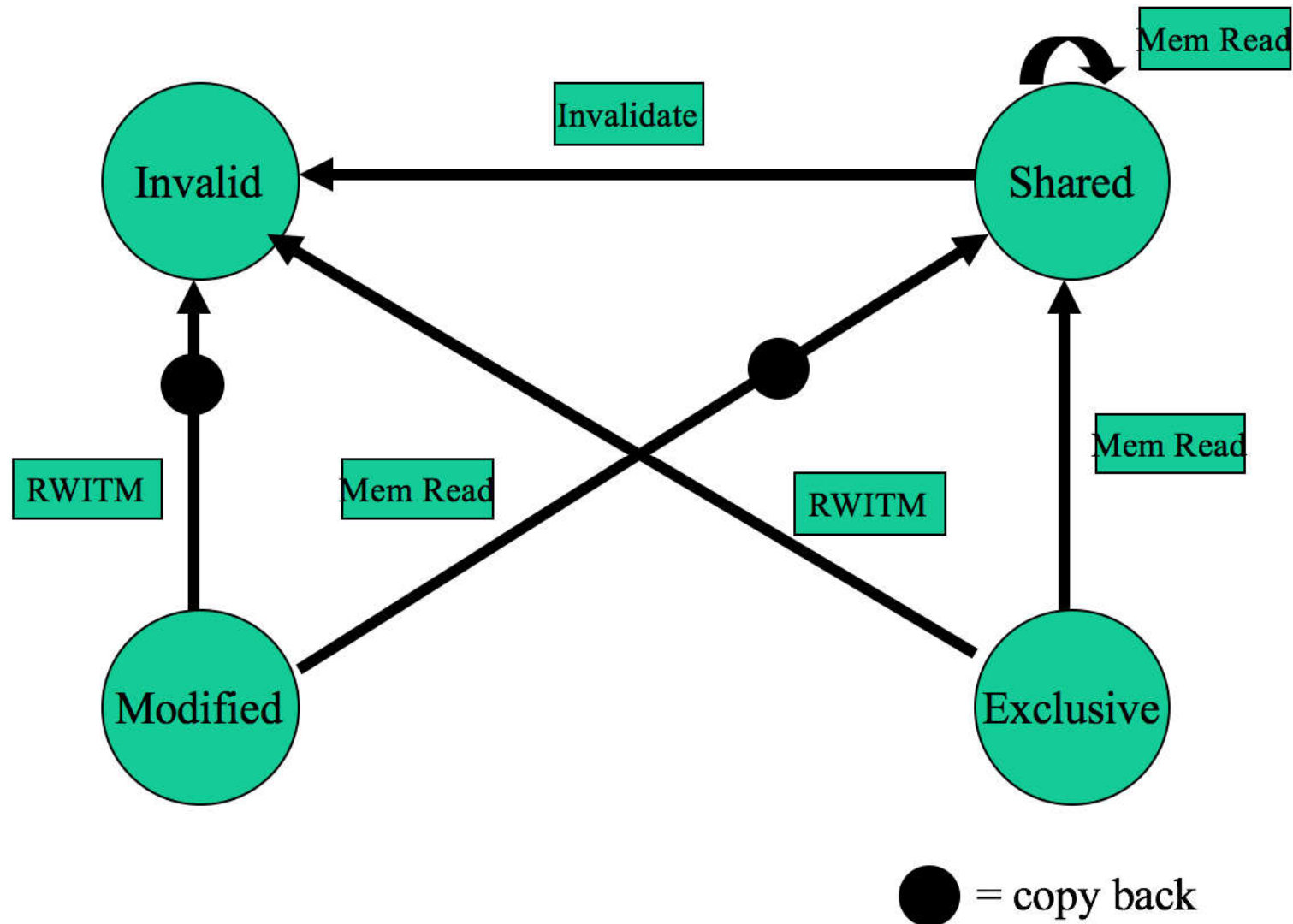
- Read Hit
- Read Miss
- Write Hit
- Write Miss

More formally by state transition diagram

MESI – LOCALLY INITIATED ACCESSES



MESI – REMOTELY INITIATED ACCESSSES



CACHE PARTITIONING

$$\text{CACHE ACCESS TIME} = \text{HIT TIME} + \text{MISS RATE} * \text{MISS PENALTY}$$

- **An important observation**
 - Different threads (datasets) can adversely impact on each other.
 - A single thread can easily “pollute” the cache with its data, causing higher miss rate for other threads.
- **Communist, Utilitarian, and Capitalist Cache Policies on CMPs: Caches as a Shared Resource [PACT’06]**
 - **Fairness:** Each thread bears an equal portion of the cache sharing penalty.
 - **Performance:** Maximum the total throughput.
 - **Hardware Overhead:** Unregulated free-for-all design.

CACHE PARTITIONING APPROACHES

$$\text{CACHE ACCESS TIME} = \text{HIT TIME} + \text{MISS RATE} * \text{MISS PENALTY}$$

- **Provide configurable cache hardware**
 - Modifying the cache placement algorithm by restricting where data can be placed in the cache.
 - Might increase the cache access time due to having to locate the correct partition.
- **Modify the cache replacement algorithm**
 - Partitioning the cache is incremental.
 - A replacement only occurs on cache misses, this approach does not add to cache access time.

CACHE UTILIZATION WITH MULTIPLE APPS

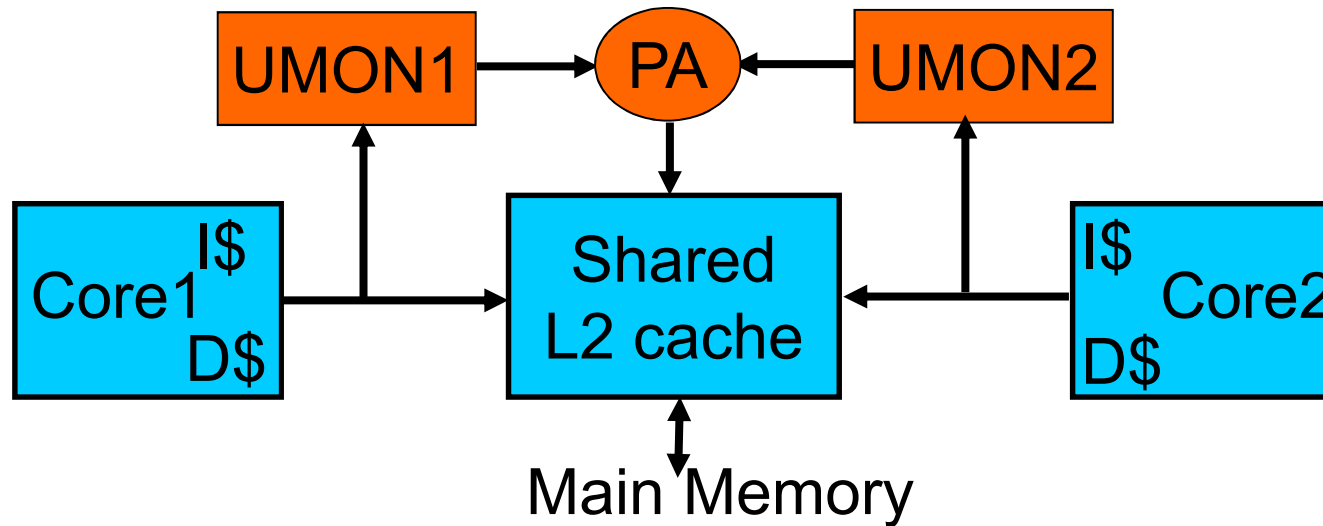
Cache partitioning

- Naturally enabled (locality)
- Fixed partitioning
- Adaptive partitioning

Purposes of partitioning

- Overall Performance?
- Priority?
- Fairness?
 - How to define fairness?

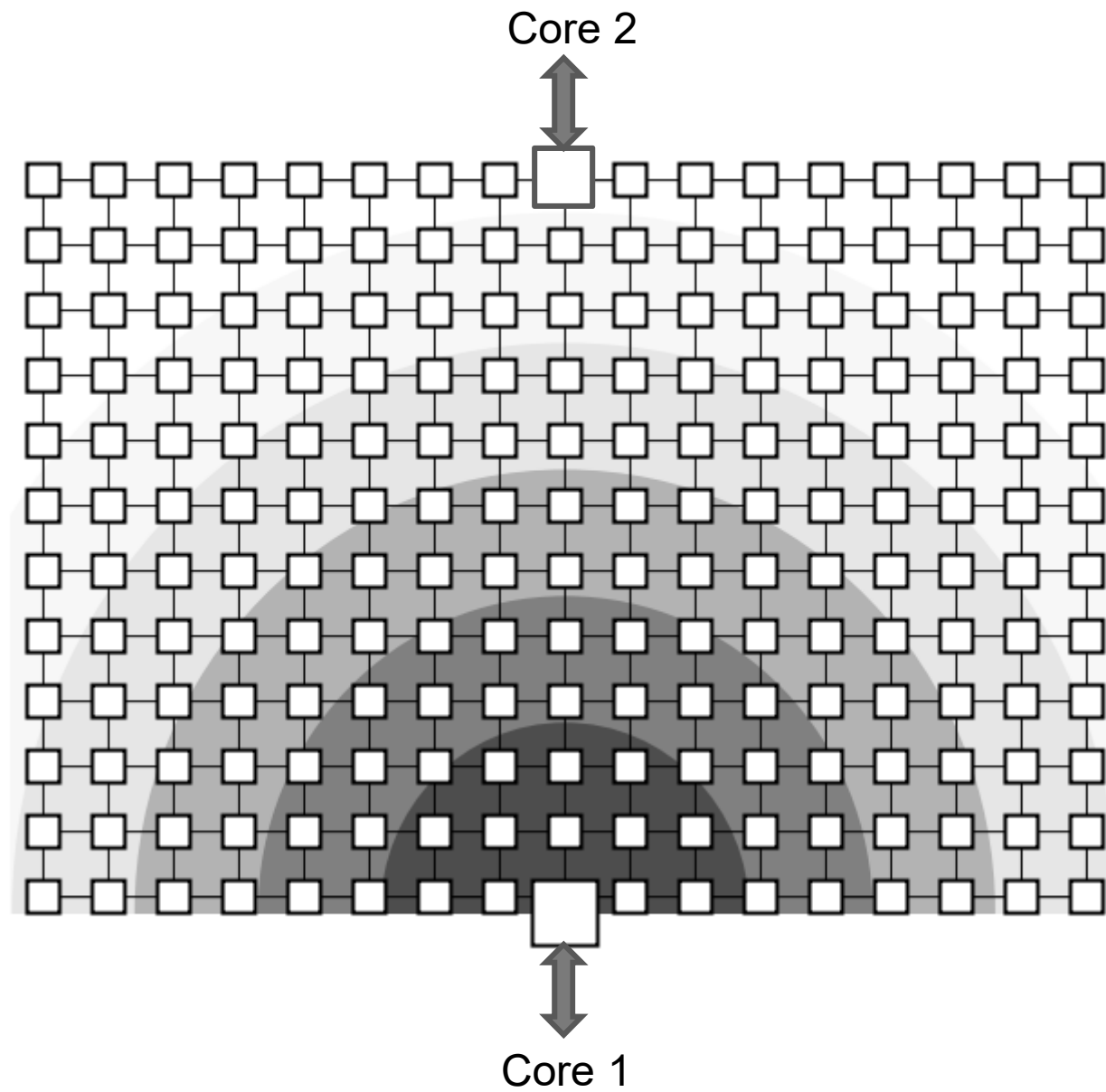
UCP: UTILIZATION BASED PARTITIONING



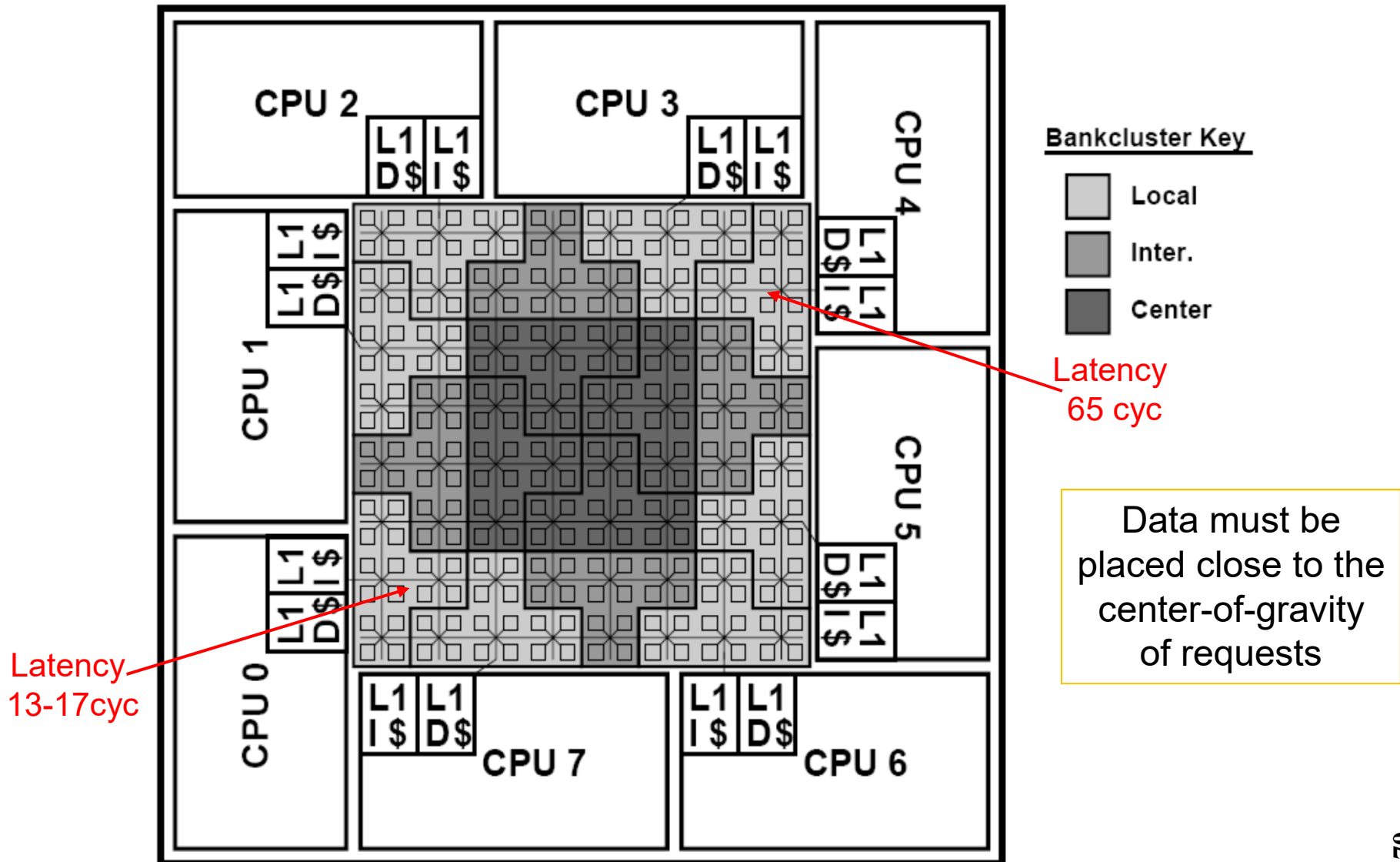
Three components:

- ☐ Utility Monitors (UMON) per core
- ☐ Partitioning Algorithm (PA)
- ☐ Replacement support to enforce partitions

DATA PLACEMENT



Beckmann and Wood, MICRO'04



Examples: Frequency of Accesses

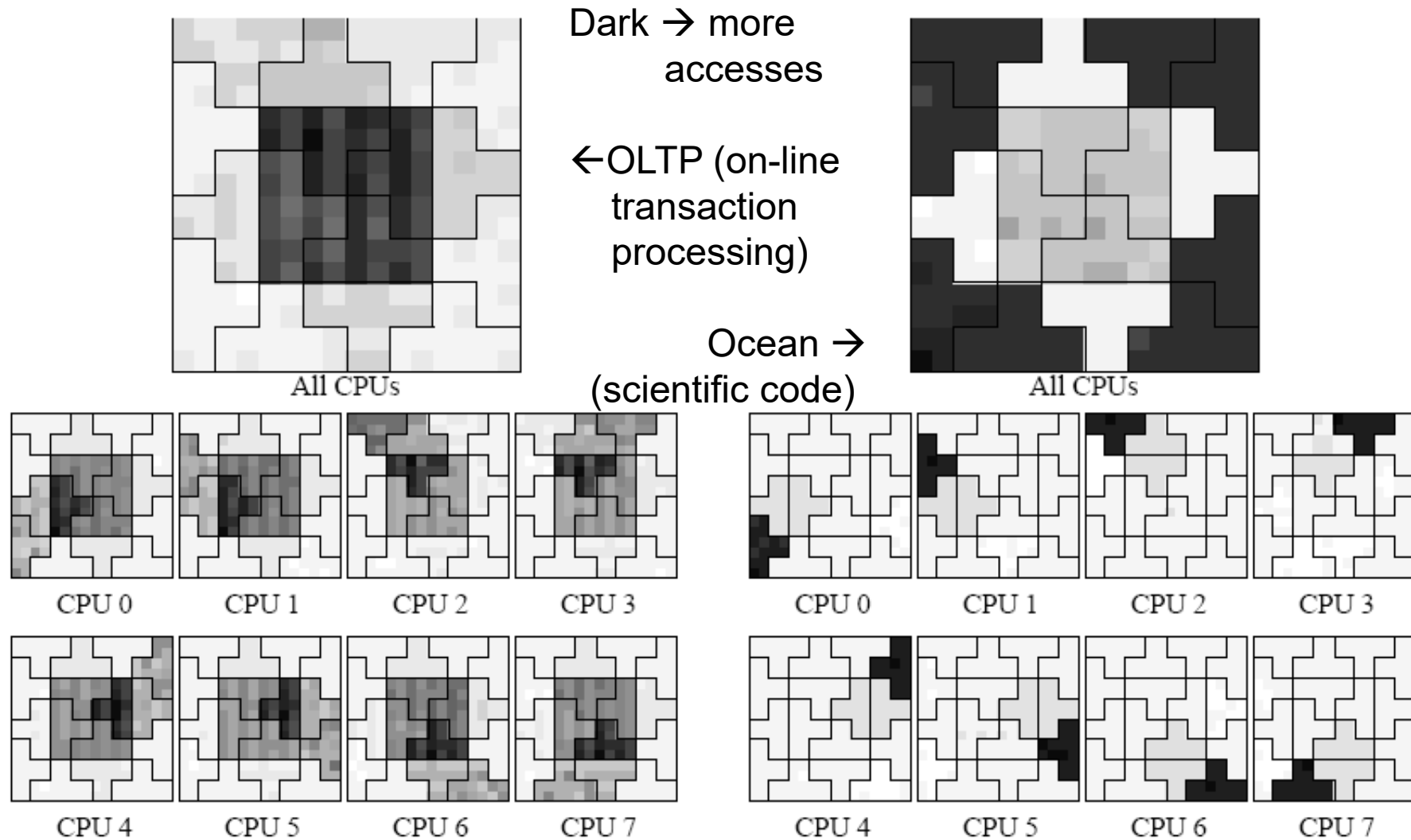


Figure 10. oltp L2 Hit Distribution

Figure 11. ocean L2 Hit Distribution

AN EXAMPLE OF ROW ACCESS

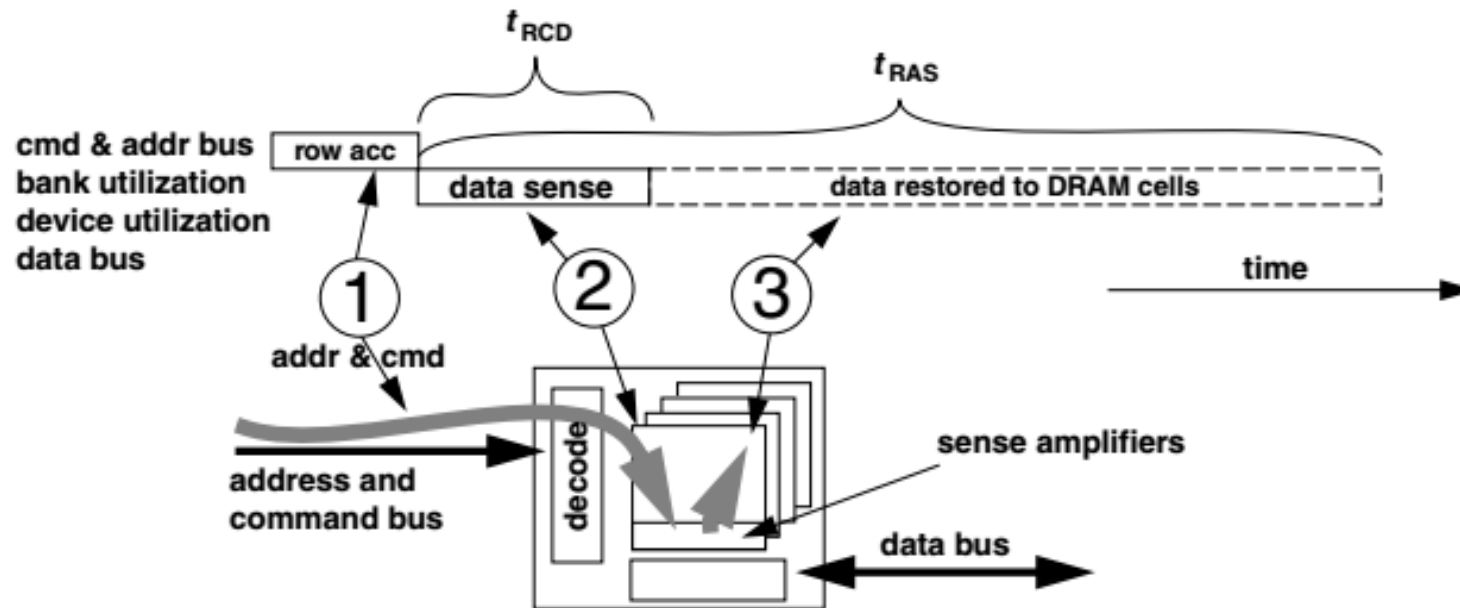


FIGURE 11.3: Row access command and timing.

AN EXAMPLE OF COLUMN READ

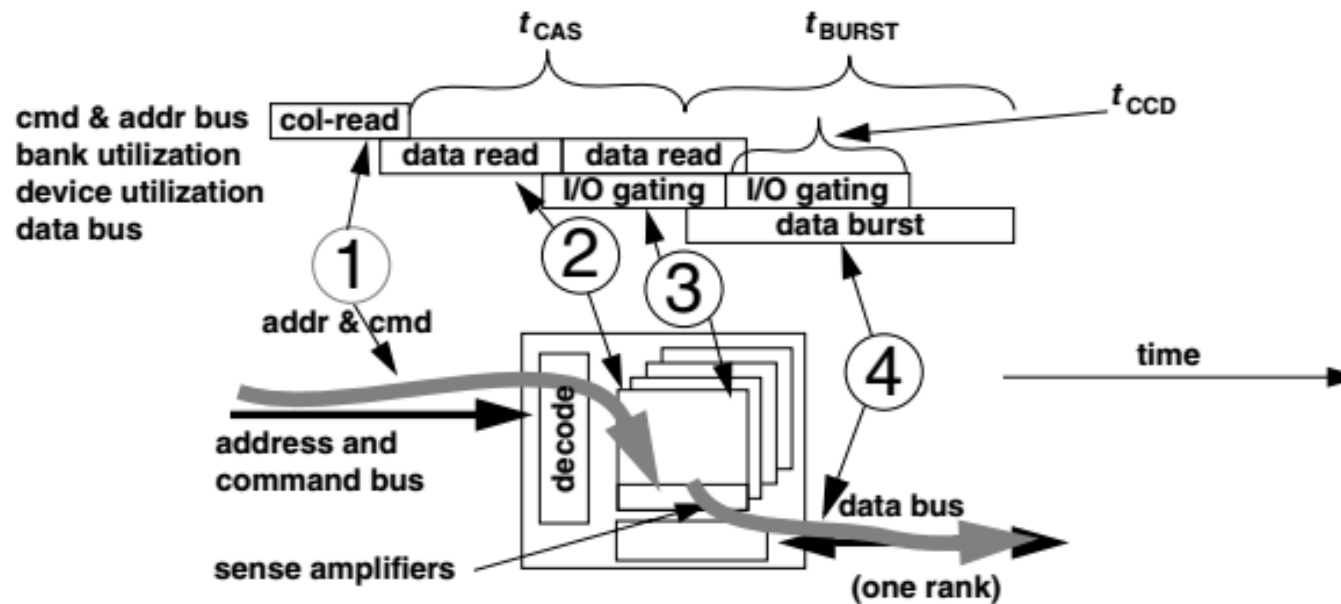
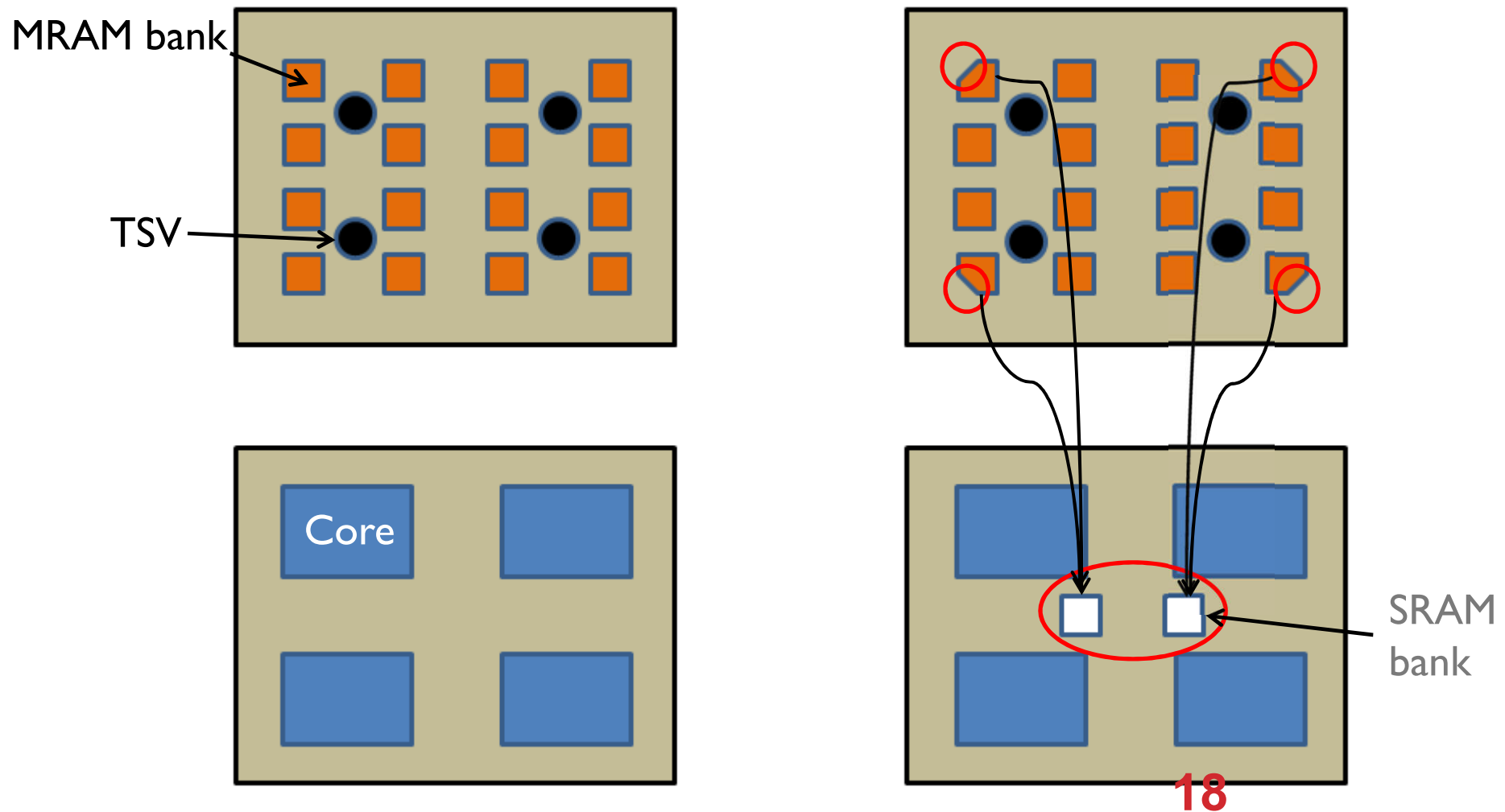


FIGURE 11.4: Column-read command and timing.

EXAMPLE: SRAM-MRAM HYBRID CACHE

31 way MRAM caches + one way SRAM cache.

(Hybrid Structure)

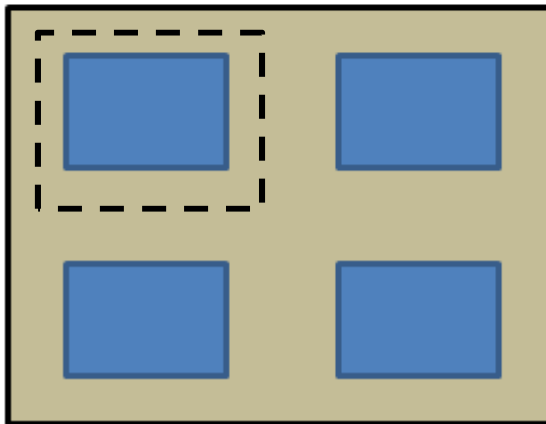
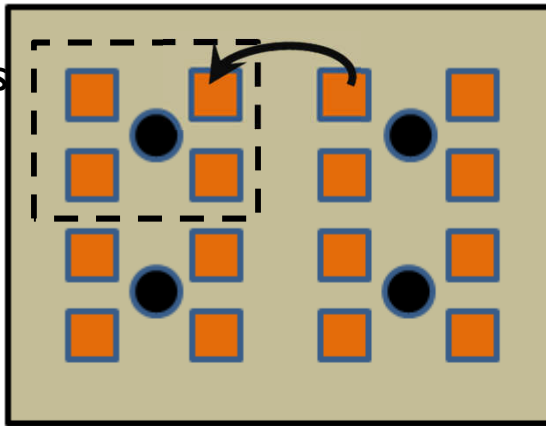


EXAMPLE: SRAM-MRAM HYBRID CACHE

- Migrate data frequently written to the SRAM cache banks.
(Reduce Write Operations)

Reduce data migrations among MRAM cache banks.

Home region
No migrations



Migration from MRAM to SRAM