ECE 5730

Memory Systems

Spring 2009

More on Memory Scheduling

Cornell University

Announcements

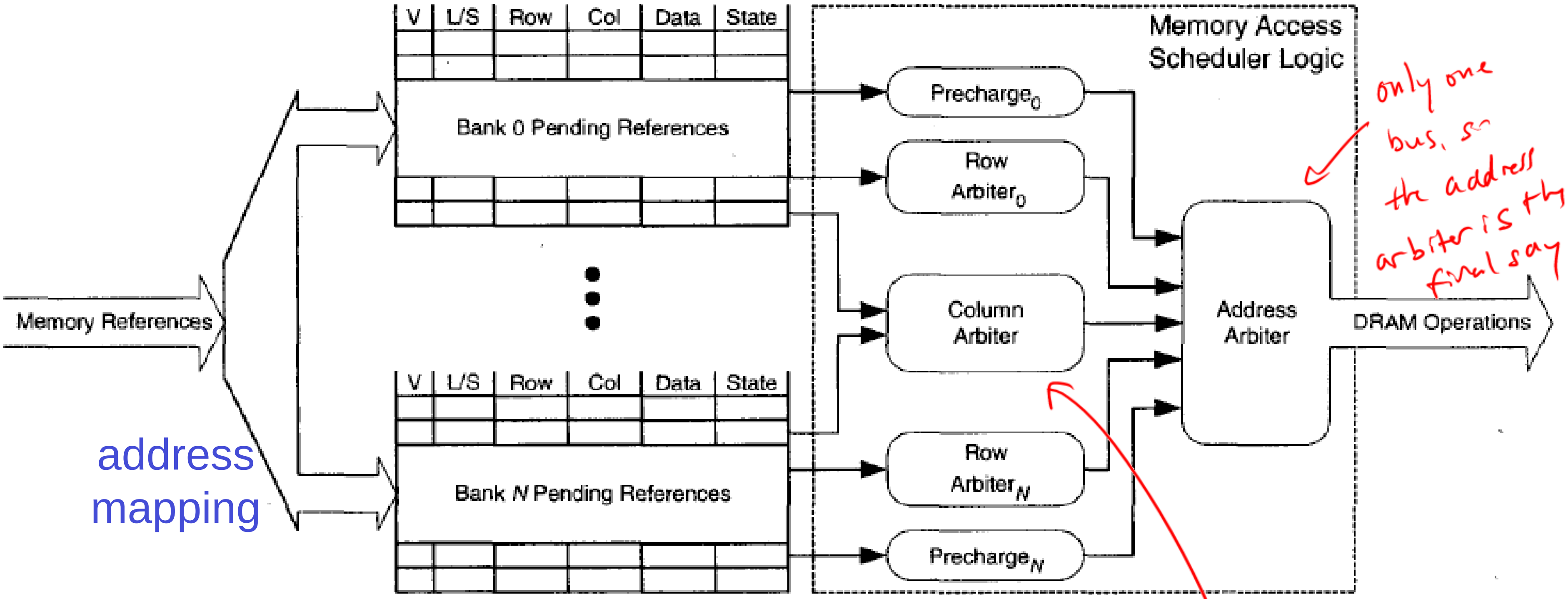
- Exam I average = ?

- Course project proposal
  - What you propose to investigate
  - What resources you plan to use (tools, benchmarks, machines)
  - The step-by-step approach you will take
  - Emailed to me by this Friday at 5pm EDT
  - 10 points off final project grade if late

Where We're Headed

- Memory controllers
  - Scheduling of read and write commands
  - Refresh management
- Memory power management
- Memory case studies

Memory Scheduler Organization



address mapping

only one bus, so the address arbiter is the final say

scheduler for a single rank

only one column arbiter, because banks can conflict within a rank on I/O circuitry

[Rixner00]

FR-FCFS Scheduler

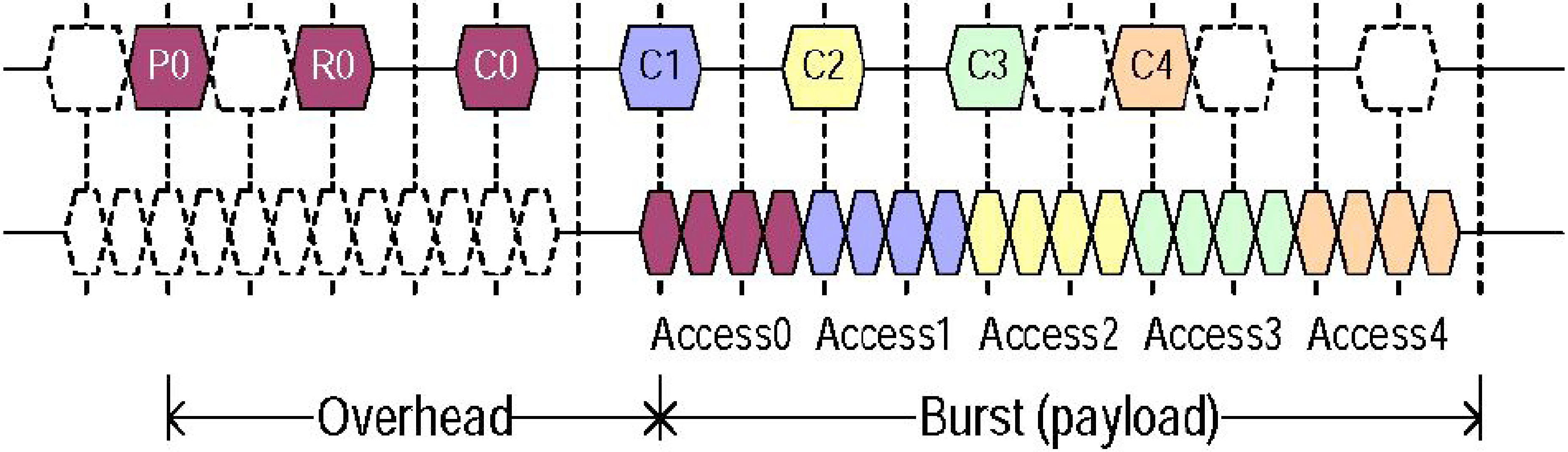
- First-ready, first-come-first-service [Rixner00]
- Widely compared with new scheduling ideas

→ actually pretty good other schemes do only marginally better

- Column commands have priority over row
- In case of a tie, select oldest command

Burst Scheduling

- Accesses to same bank and row are grouped into bursts



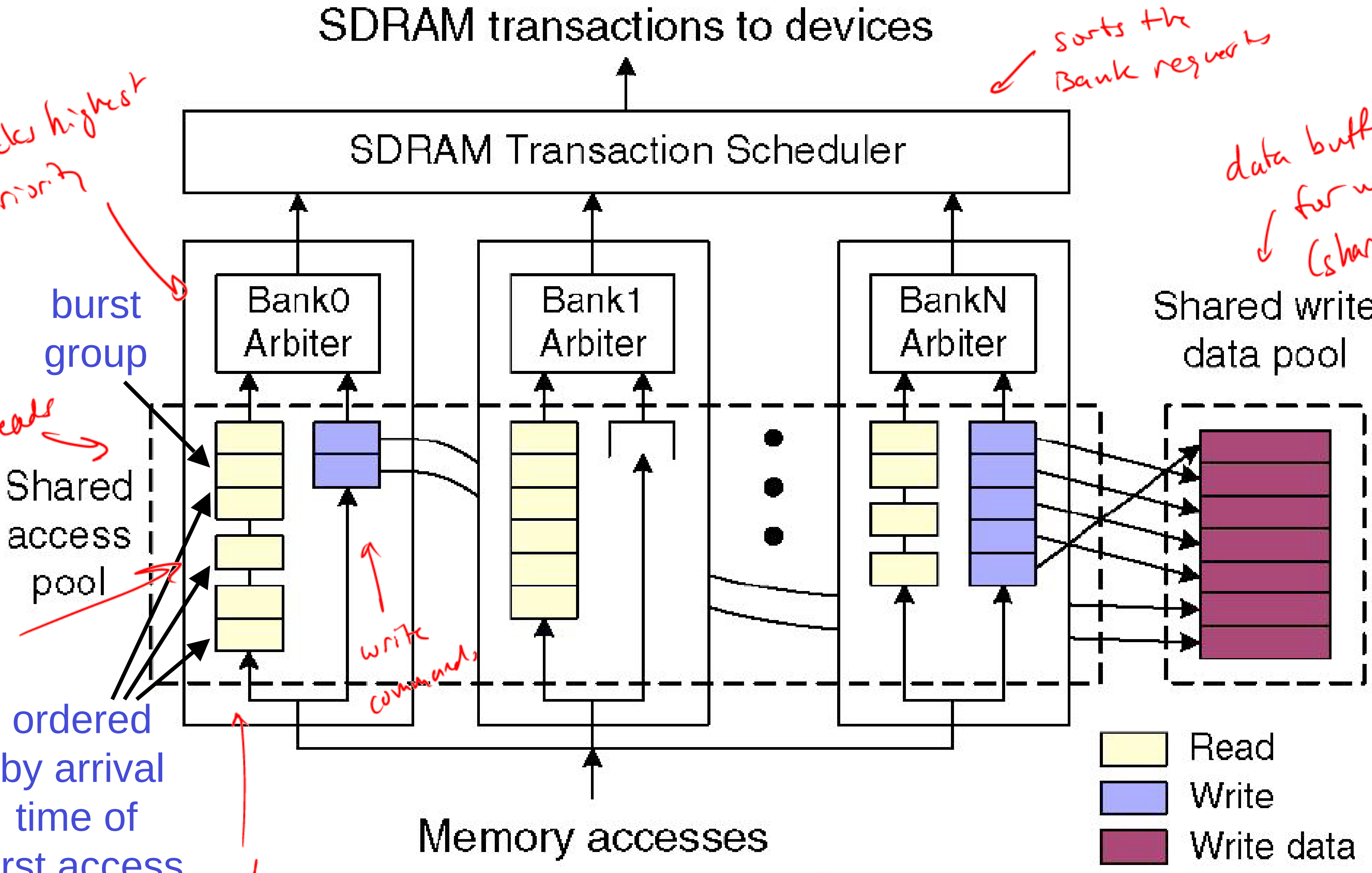
how to take advantage of open rows? prioritize and group!

we have to guarantee some degree of fairness.

can't let commands sit idle forever, etc

[Shao07]

Burst Scheduler Organization



sorts the Bank requests

picks highest priority

data buffer for writes (shared)

group of reads

single read

write commands

read commands

give priority to older commands

[Shao07]

Burst Scheduler Organization

- Read and write queues implemented as a global pool of queue entries
- Newly arriving requests join an existing burst group or create new one
- Bank arbiters select a request for each bank, and the scheduler selects one of these requests
- Older bursts generally receive priority, but long newer bursts may delay shorter older ones

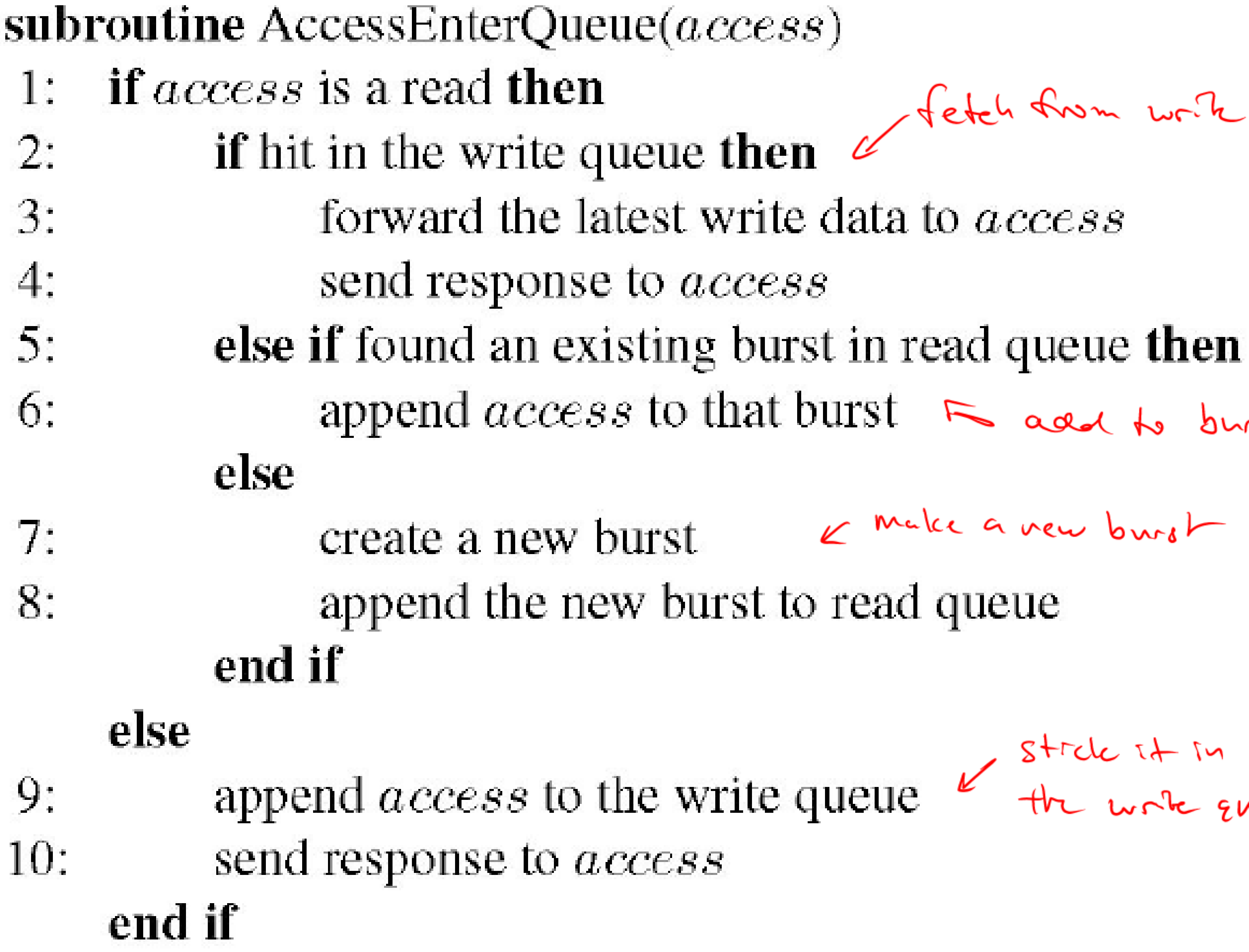
3 different algorithms

Burst Scheduler Components

- Access enter queue
  - Determines actions when a new request arrives
- Bank arbiter
  - Selects one request from the queue for its bank

    ⇒ highest priority
- Transaction scheduler
  - Selects one transaction to be sent on the channel

Access Enter Queue Algorithm



```
subroutine AccessEnterQueue($access$)
if $access$ is a read then
    if hit in the write queue then
        forward the latest write data to $access$
        send response to $access$
    else if found an existing burst in read queue then
        append $access$ to that burst
    else
        create a new burst
        append the new burst to read queue
    end if
else
    append $access$ to the write queue
    send response to $access$
end if
```

fetch from write buffer

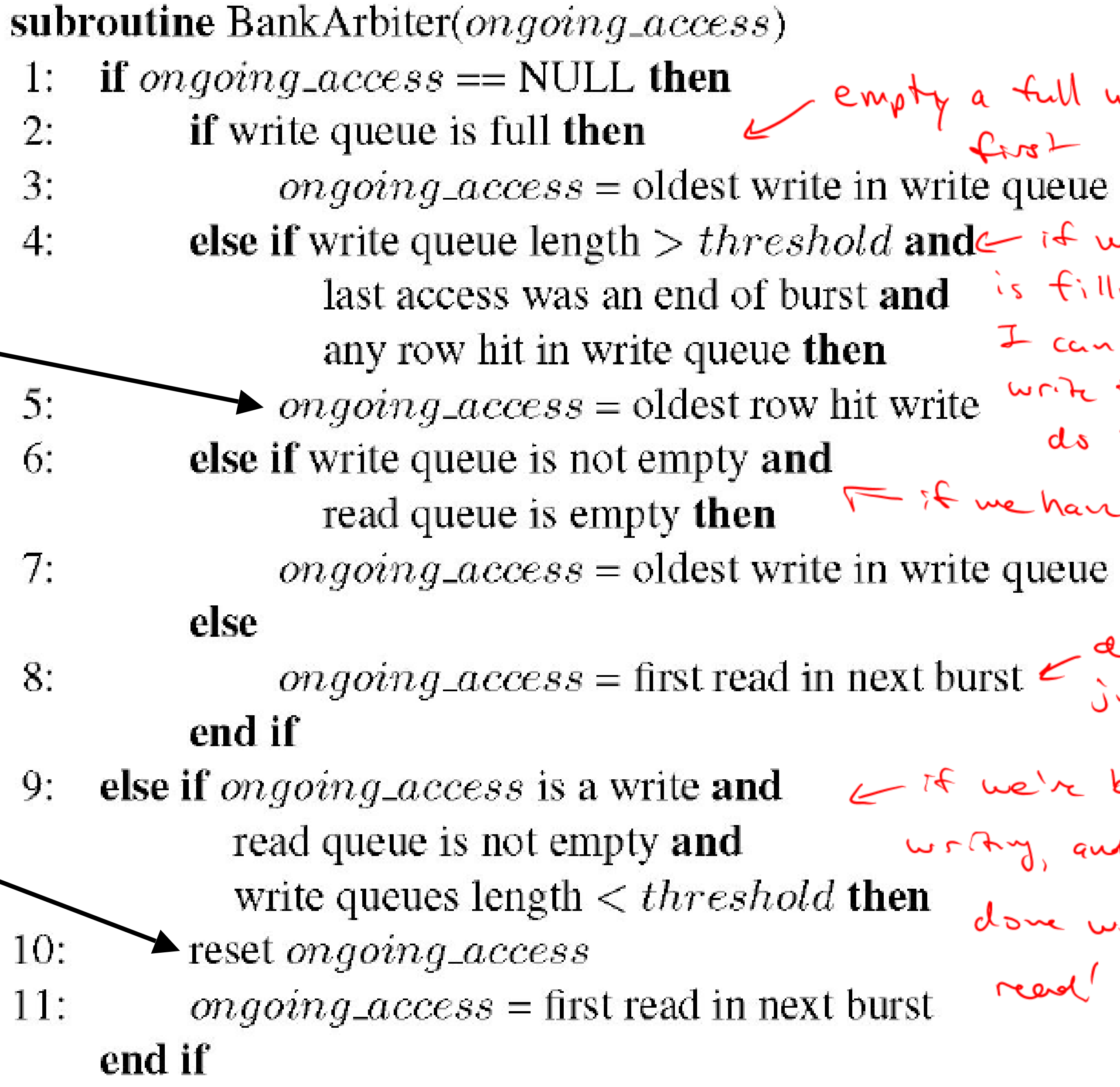
add to burst

make a new burst

stick it in the write queue

[Shao07]

Bank Arbiter Algorithm



```
subroutine BankArbiter($ongoing\_access$)
if $ongoing\_access$ == NULL then
    if write queue is full then
        $ongoing\_access$ = oldest write in write queue
    else if write queue length > $threshold$ and
            last access was an end of burst and
            any row hit in write queue then
        $ongoing\_access$ = oldest row hit write
    else if write queue is not empty and
            read queue is empty then
        $ongoing\_access$ = oldest write in write queue
    else
        $ongoing\_access$ = first read in next burst
    end if
else if $ongoing\_access$ is a write and
        read queue is not empty and
        write queues length < $threshold$ then
    reset $ongoing\_access$
    $ongoing\_access$ = first read in next burst
end if
```

empty a full write queue first

if write queue is filling up and I can append the write to the burst do it!

write piggybacking

if we have no reads, just write

default case just read

if we've been writing, and we're mostly done writing, then read! kill the write!

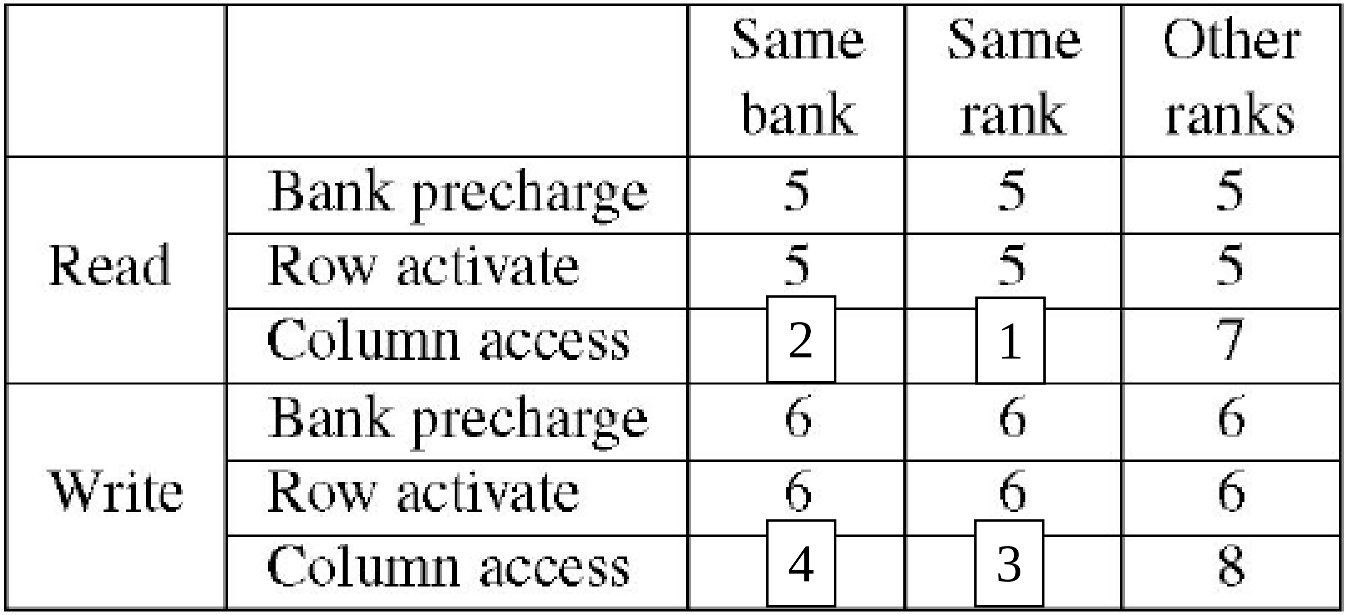
read preemption

[Shao07]

Transaction Scheduler Algorithm

- Queued cmd is *unblocked* if it can be issued without violating DRAM timing constraints
- Unblocked cmd priority (1 = highest)

✓ priority table

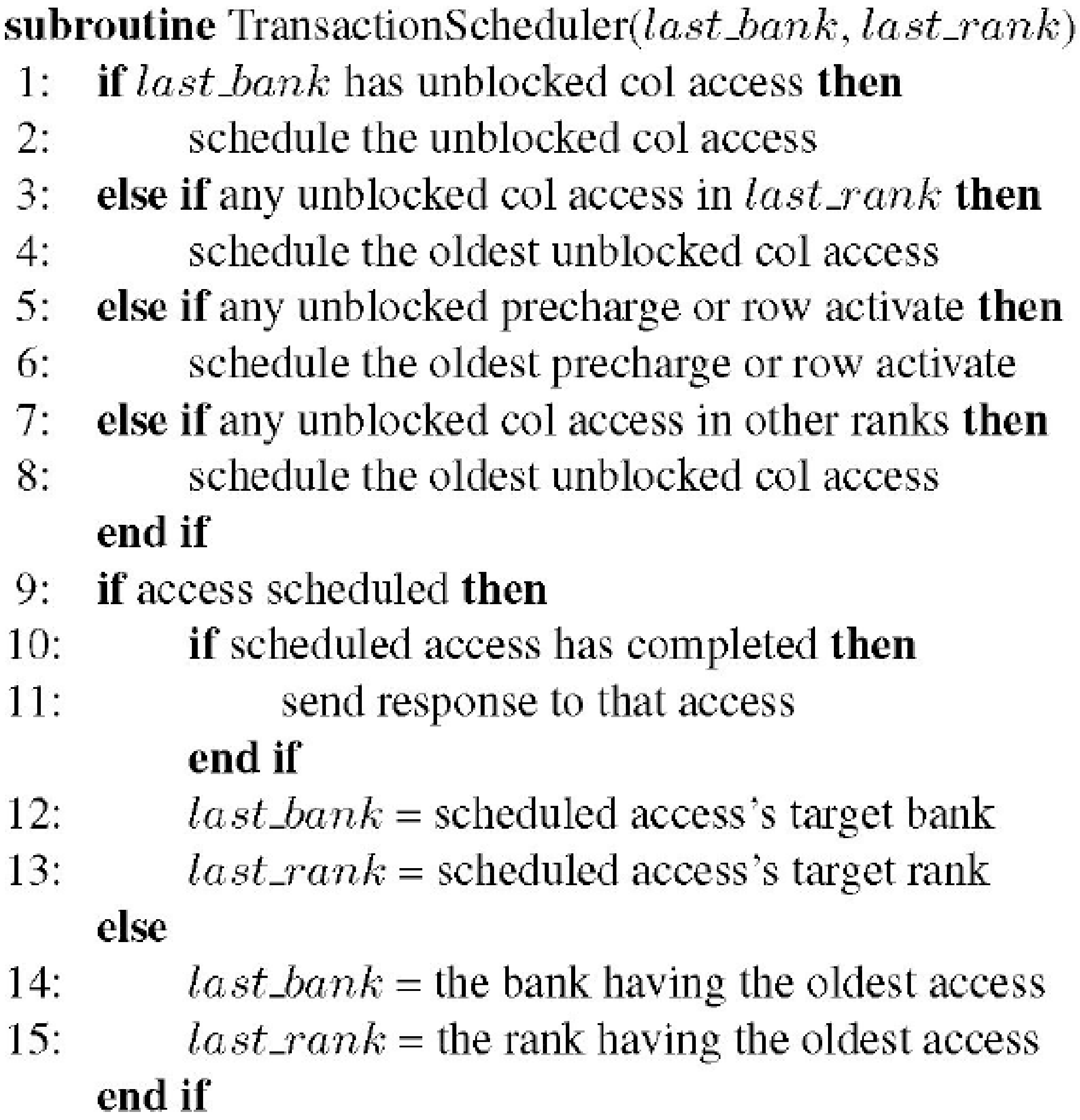


		Same bank	Same rank	Other ranks
Read	Bank precharge	5	5	5
	Row activate	5	5	5
	Column access	2	1	7
Write	Bank precharge	6	6	6
	Row activate	6	6	6
	Column access	4	3	8

- Oldest cmd selected if a tie

[Shao07]

Transaction Scheduler Algorithm



```
subroutine TransactionScheduler($last\_bank$, $last\_rank$)
if $last\_bank$ has unblocked col access then
    schedule the unblocked col access
else if any unblocked col access in $last\_rank$ then
    schedule the oldest unblocked col access
else if any unblocked precharge or row activate then
    schedule the oldest precharge or row activate
else if any unblocked col access in other ranks then
    schedule the oldest unblocked col access
end if
if access scheduled then
    if scheduled access has completed then
        send response to that access
    end if
    $last\_bank$ = scheduled access's target bank
    $last\_rank$ = scheduled access's target rank
else
    $last\_bank$ = the bank having the oldest access
    $last\_rank$ = the rank having the oldest access
end if
```

basically the same as the table

[Shao07]

Avoiding Hazards

read after write

- **RAW: Incoming reads that hit in the write queue have results forwarded to them**

write after read

- **WAR: Within bursts, writes are always piggybacked after reads**

write after write

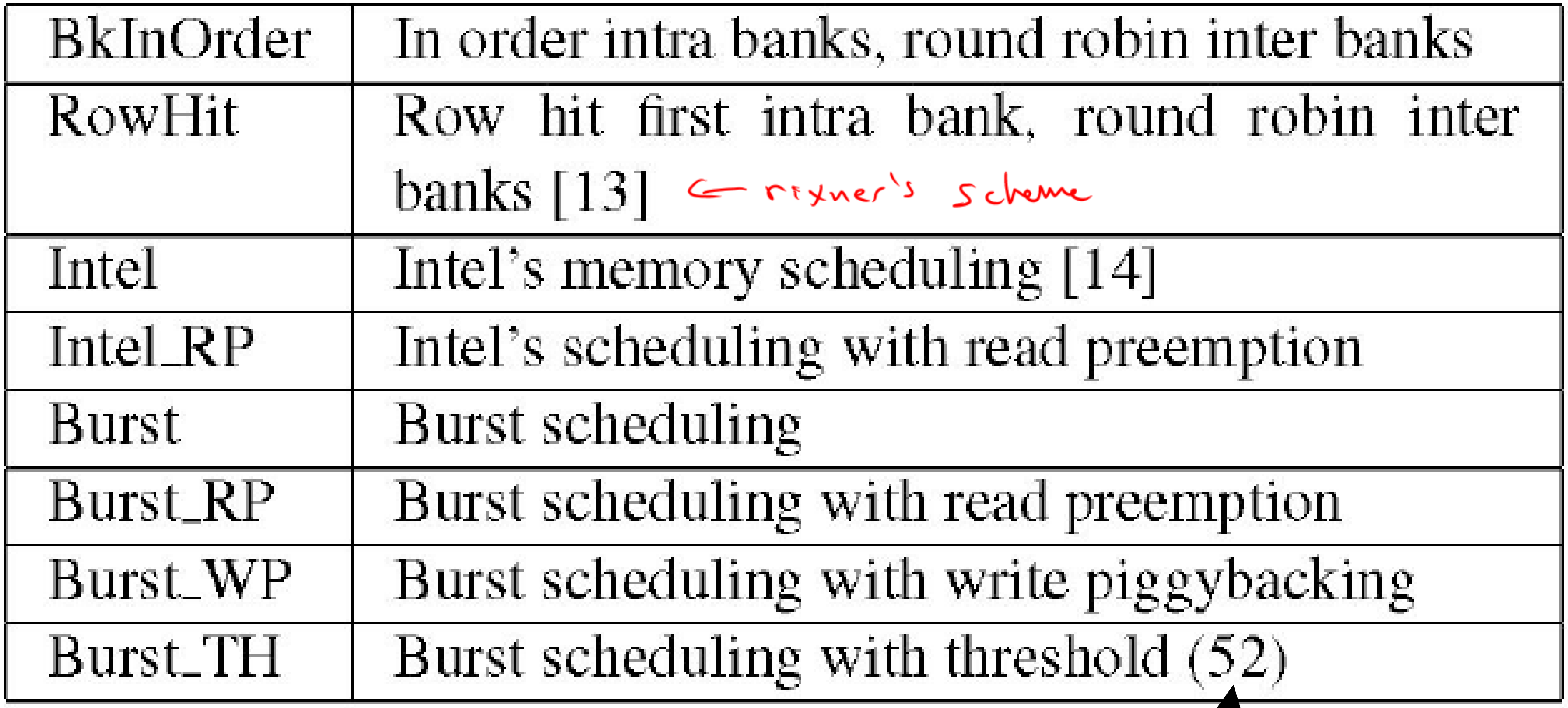
- **WAW: Within bursts, newer writes are always piggybacked after older ones**

not hard, just some comparators and logic

Performance Evaluation

- **Evaluated scheduling policies**

naive scheduler

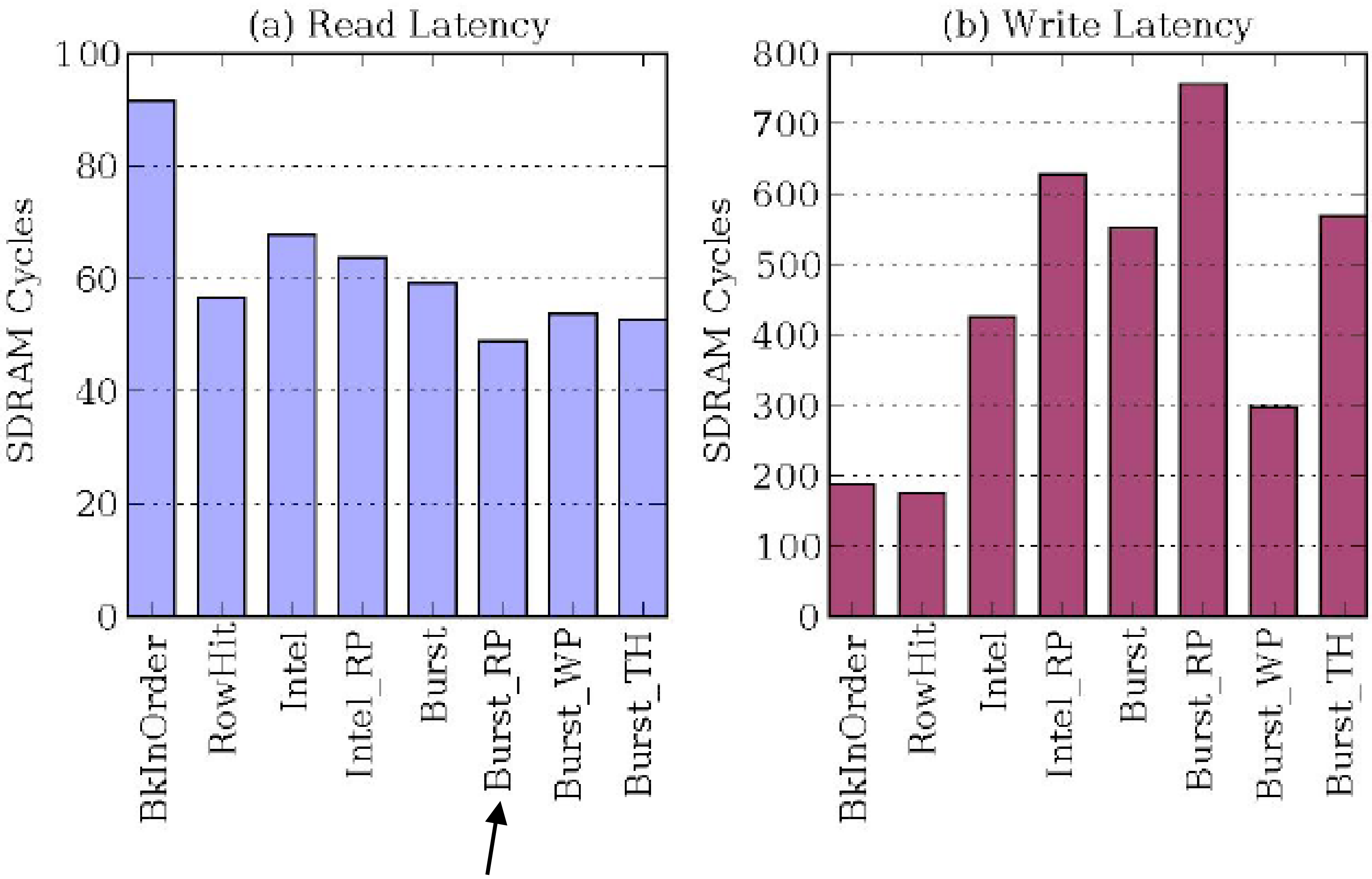


BkInOrder	In order intra banks, round robin inter banks
RowHit	Row hit first intra bank, round robin inter banks [13] ← rixner's scheme
Intel	Intel's memory scheduling [14]
Intel_RP	Intel's scheduling with read preemption
Burst	Burst scheduling
Burst_RP	Burst scheduling with read preemption
Burst_WP	Burst scheduling with write piggybacking
Burst_TH	Burst scheduling with threshold (52)

with write queue of 64 entries

[Shao07]

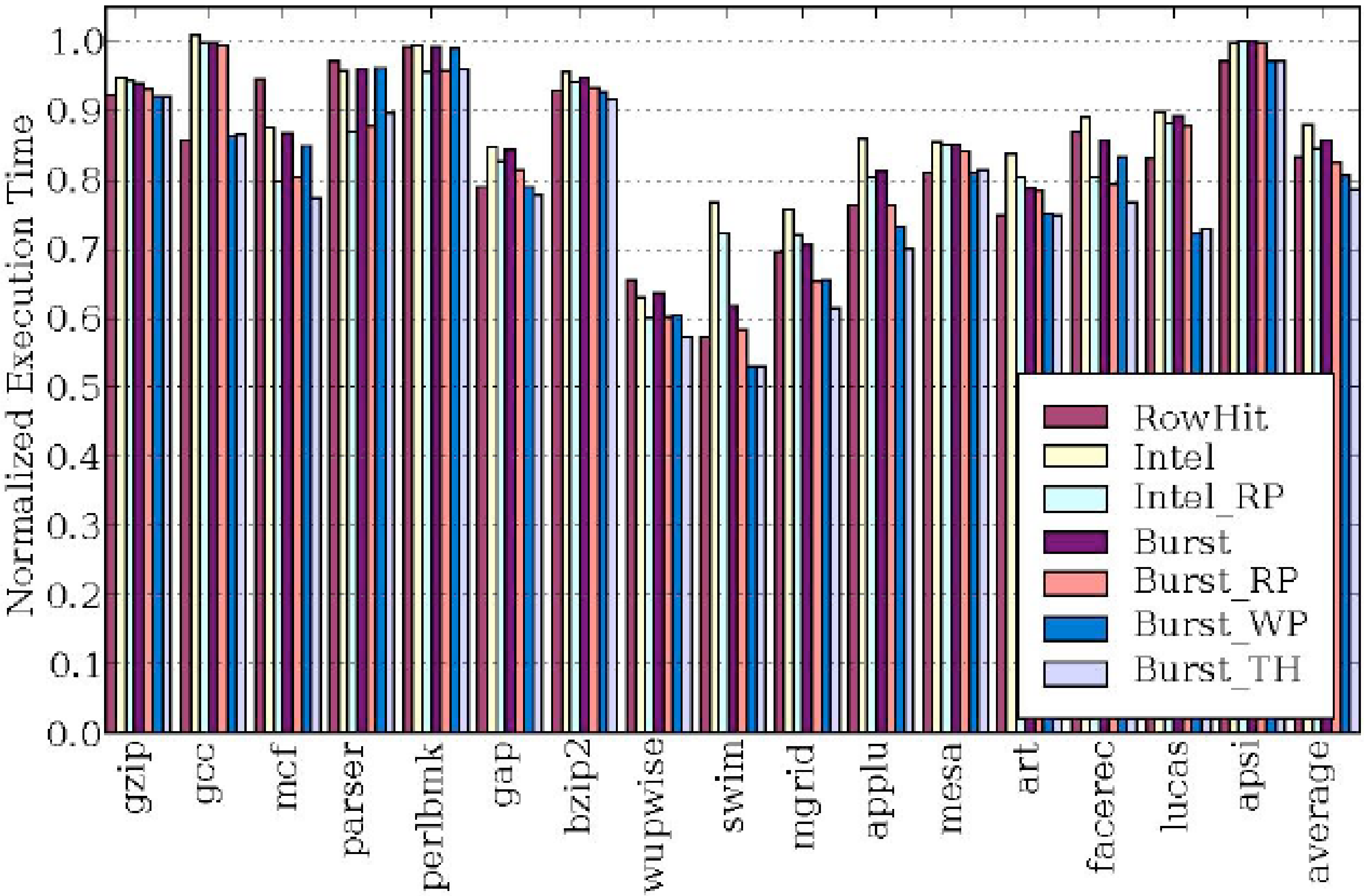
Access Latency Comparison



but more write queue stalls!

[Shao07]

Execution Time Comparison



[Shao07]

moar different scheme! ☺

Adaptive History-Based Scheduling

- Command selection based on past command history and queued unblocked commands

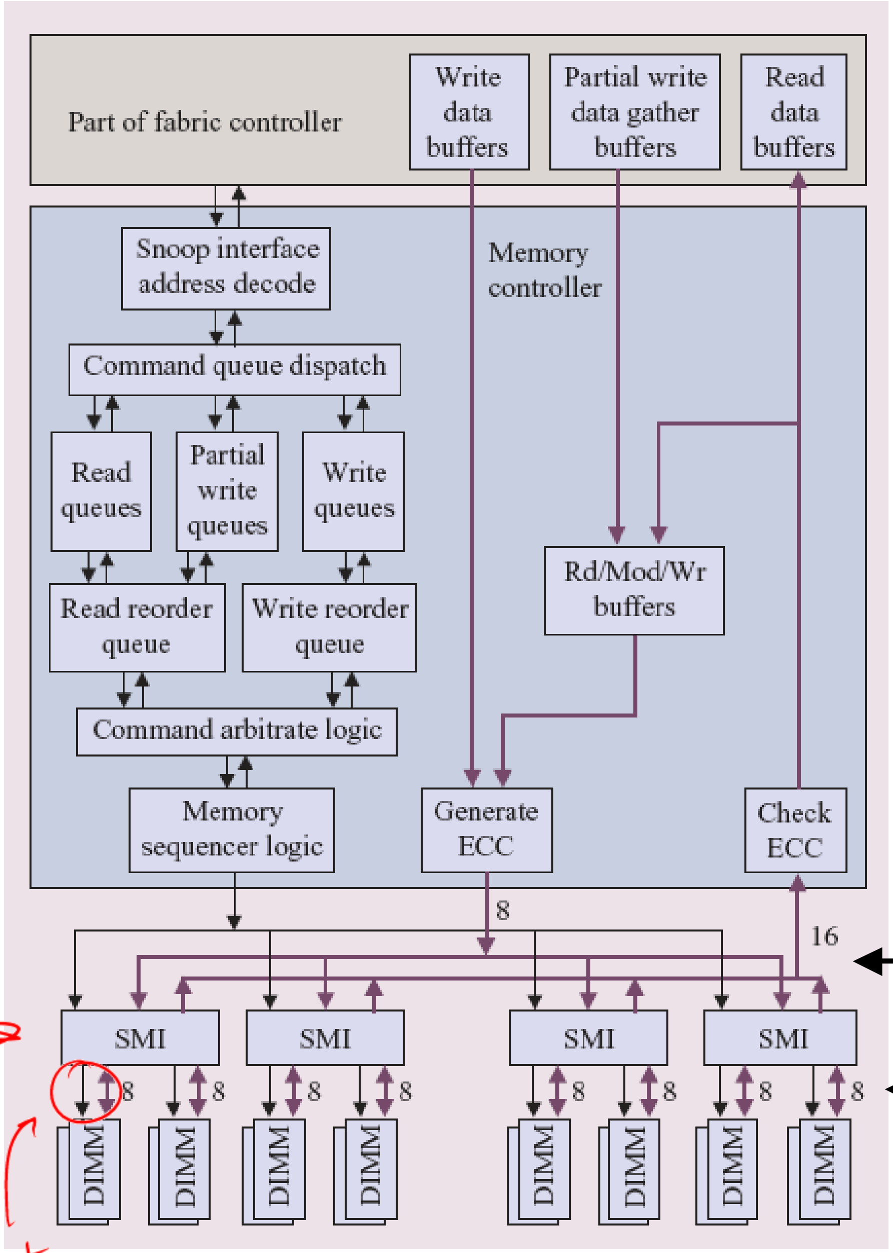
finite state machines
→

- Multiple arbiter algorithms implemented as FSMs

↳ different algorithms for different situations

- One of the FSMs selected to determine which cmd to issue

Power5 Memory System



MC connects to SMI chips

2 *ports*/SMI

sequential memory interface (2 ports)

port, 2x channels

[Sinharoy05]

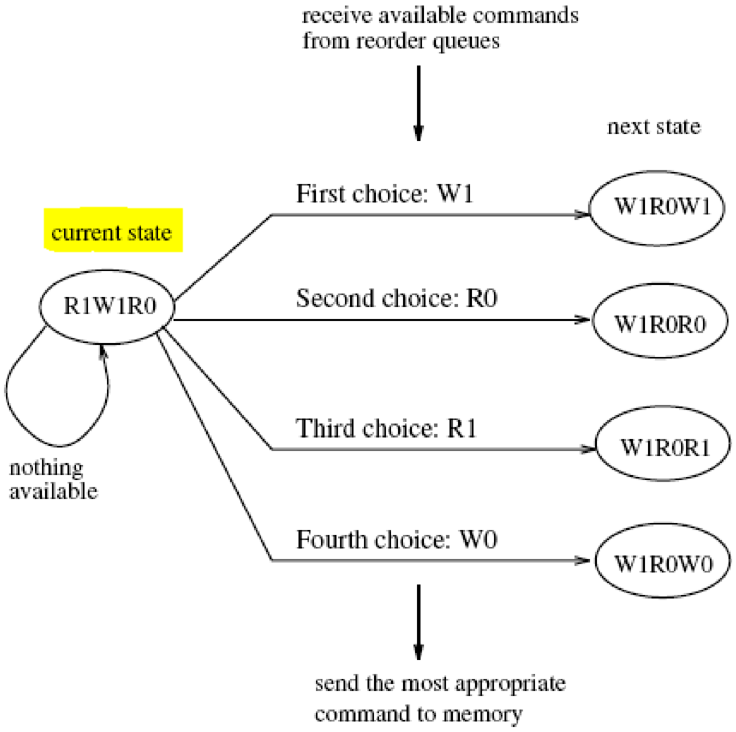
Example Arbiter FSM

- Assume MC connects to one external SMI chip that connects to two DIMM ports
  - Four possible commands: R0, R1, W0, W1

4x4x4

- Keep history of last 3 cmds (64 state FSM)

- Partial state diagram



[Hur04]

won best paper in '04

Arbiter Algorithms

- #1 **Command pattern**
  - Tries to achieve some ratio of reads to writes

    - using the state machine to issue commands to get a certain ratio

- #2 **Expected latency**
  - Selects the cmd that can be issued the soonest considering conflicts with recently issued commands

    - try to go fast & avoid conflicts

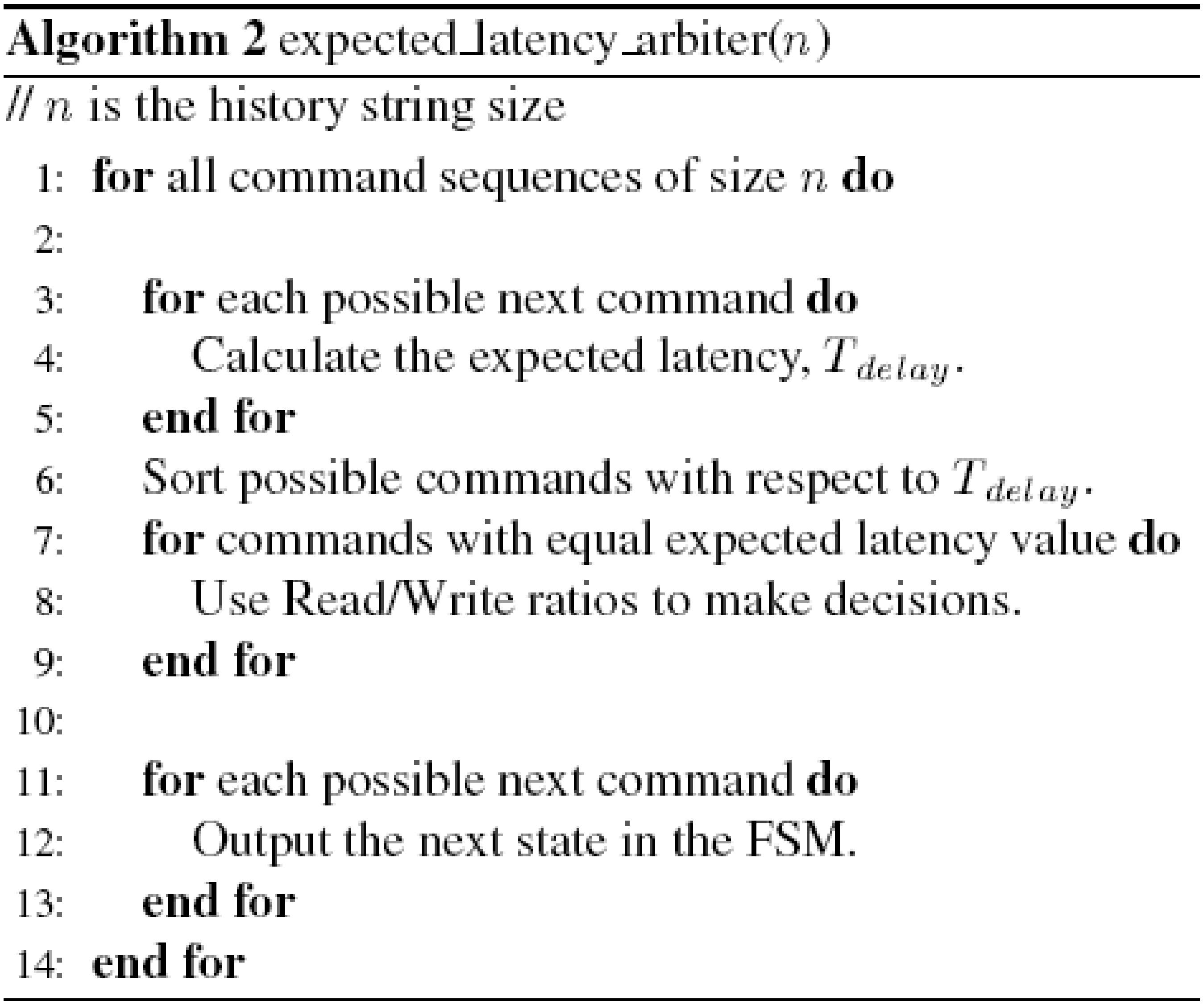
- **Probabilistic arbiter**
  - Probabilistically chooses one of the two algorithms

    - Chooses between #1 and #2

Command Pattern Algorithm

- Goal is to achieve on average some ratio of reads to writes
- History of last n commands is kept
- Cmd that (when issued) gives closest to the desired ratio is chosen
  - Example: R0, R1, W0 ⇒ W0 and W1 have priority if goal is to achieve 1/1 ratio of reads to writes
- Ties broken through another criteria, e.g., lowest expected latency

Expected Latency Algorithm



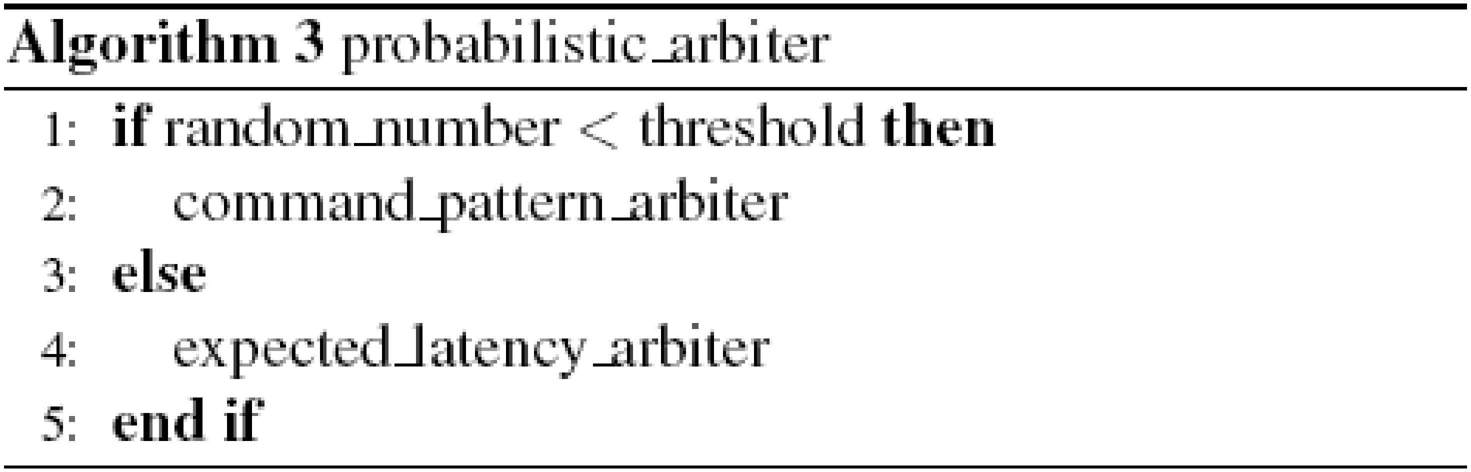
Algorithm 2 expected_latency_arbiter(n)

```
// $n$ is the history string size
for all command sequences of size $n$ do

    for each possible next command do
        Calculate the expected latency, $T_{delay}$.
    end for
    Sort possible commands with respect to $T_{delay}$.
    for commands with equal expected latency value do
        Use Read/Write ratios to make decisions.
    end for

    for each possible next command do
        Output the next state in the FSM.
    end for
end for
```

[Hur04]

Probabilistic Arbiter

- Probabilistically chooses between command pattern and expected latency algorithms

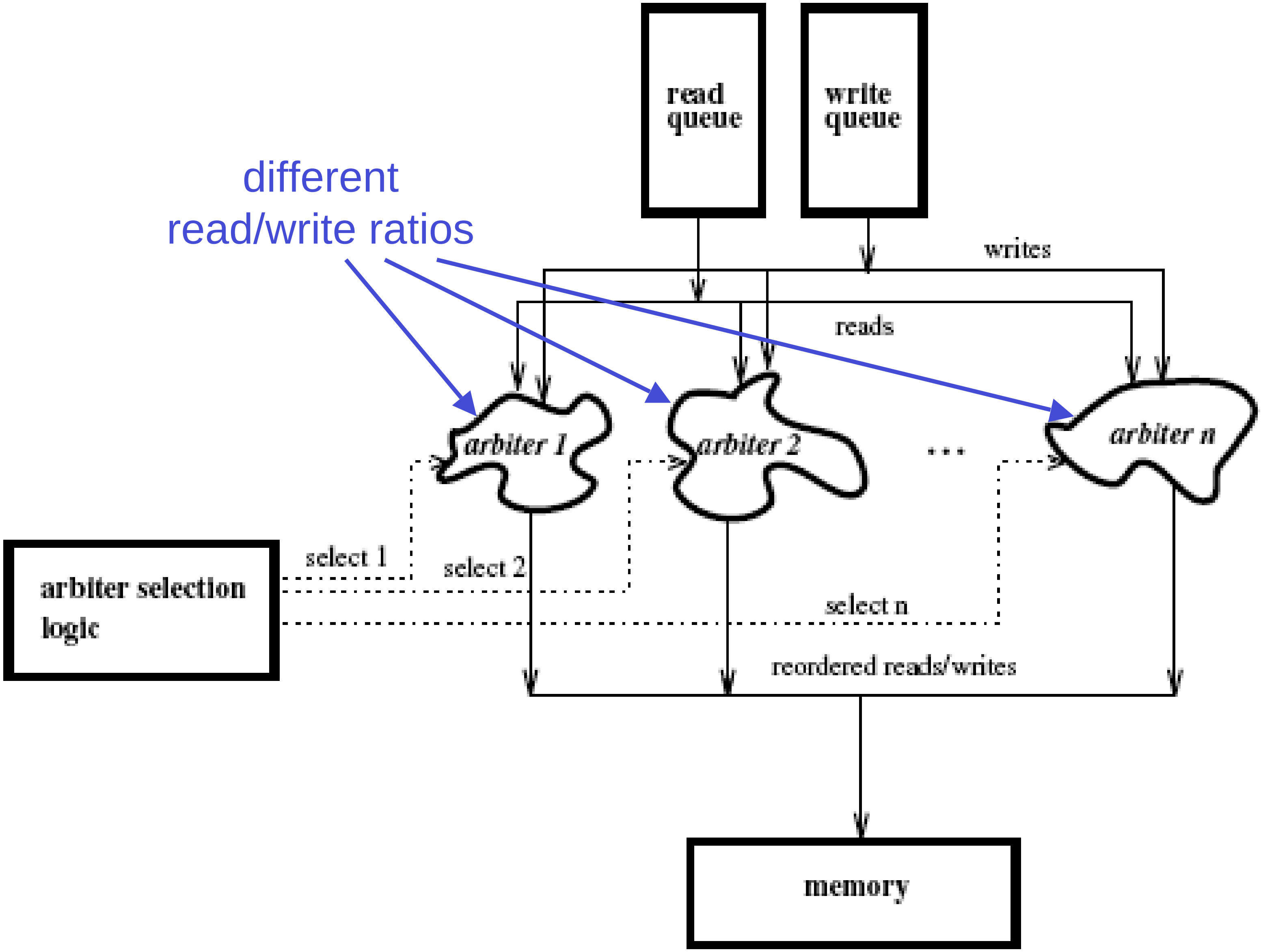


Algorithm 3 probabilistic_arbiter

```
if random_number < threshold then
    command_pattern_arbiter
else
    expected_latency_arbiter
end if
```

[Hur04]

Adaptive Selection of Arbiters



different read/write ratios

[Hur04]

Adaptive Selection of Arbiters

- Three arbiters that differ in R/W ratio
  - 2R/1W, 1R/1W, 1R/2W

- Calculate R/W ratio every 10K processor cycles

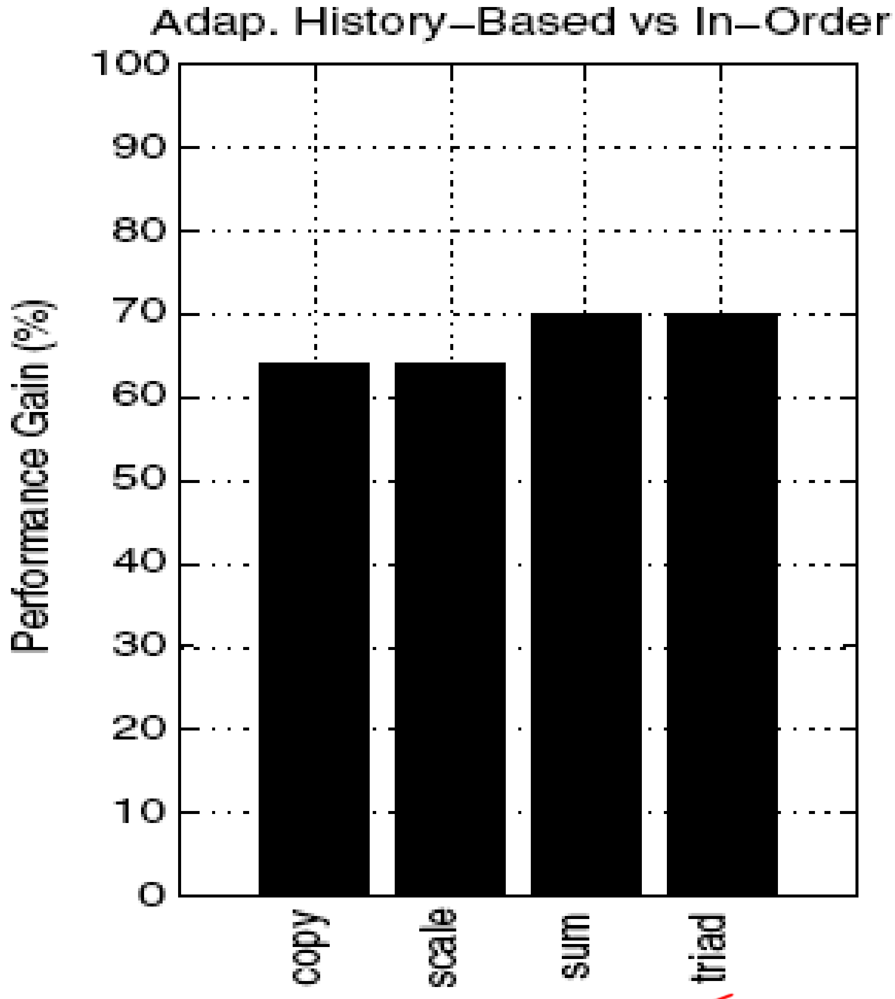
- If R/W > 1.2, pick 2R/1W
- If R/W < 0.8, pick 1R/2W
- Else, pick 1R/1W

Adaptive + FR-FCFS

- FR-FCFS selects cmd in each rank in each port

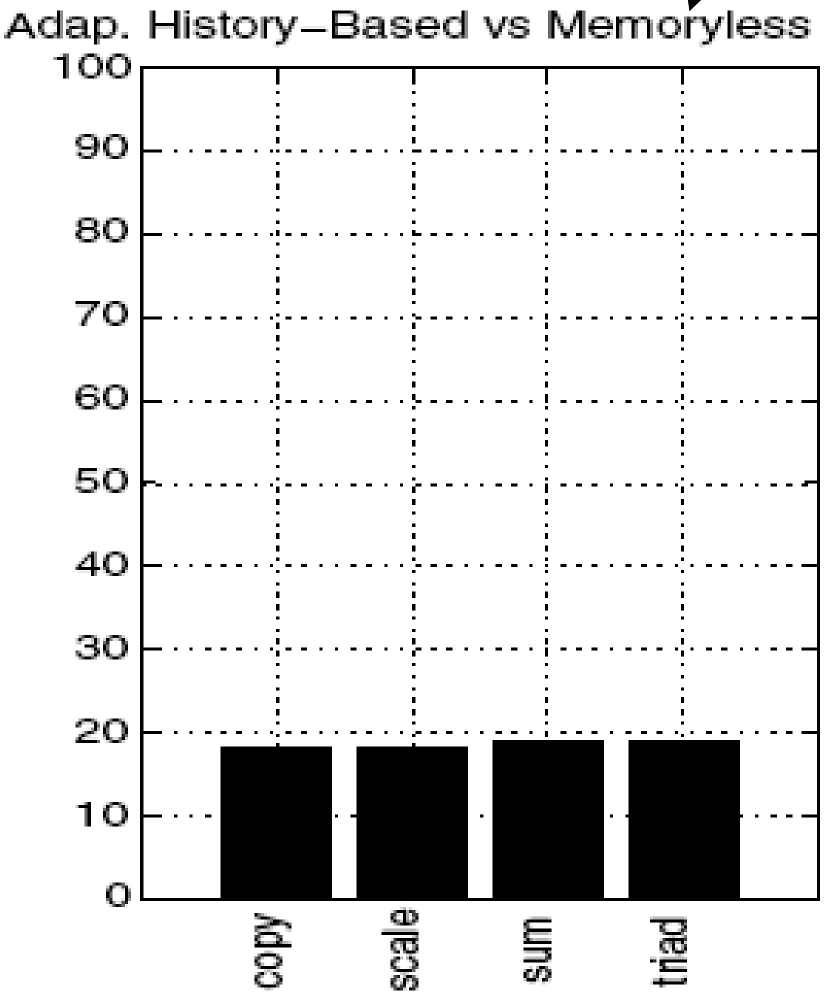
- Adaptive history-based scheduler chooses among selected channel and rank cmds

Performance Improvement



Stream memory benchmarks

FR-FCFS



[Hur04]

Next Time

Memory Scheduling For CMPs