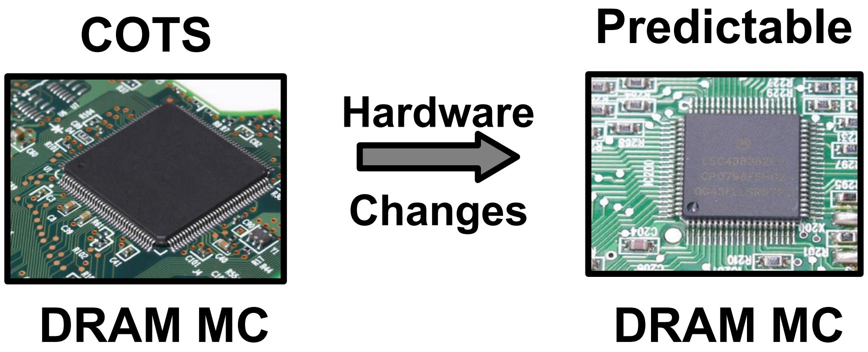
Reverse-engineering Embedded Memory Controllers through Latency-based Analysis

Mohamed Hassan, Anirudh M. Kaushik and Hiren Patel

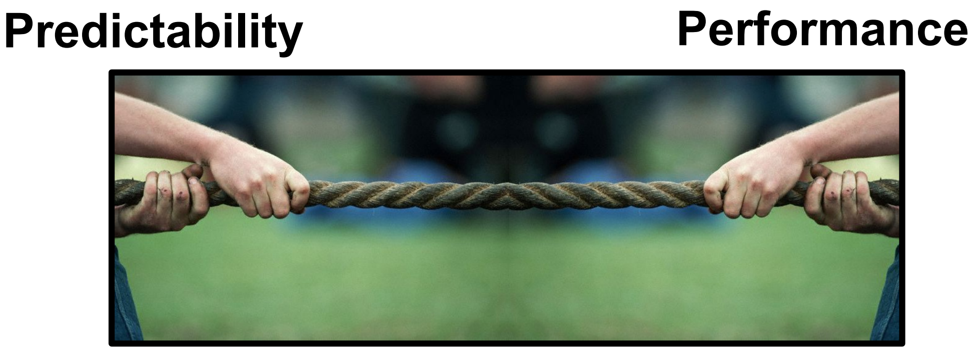
UNIVERSITY OF WATERLOO

Motivation

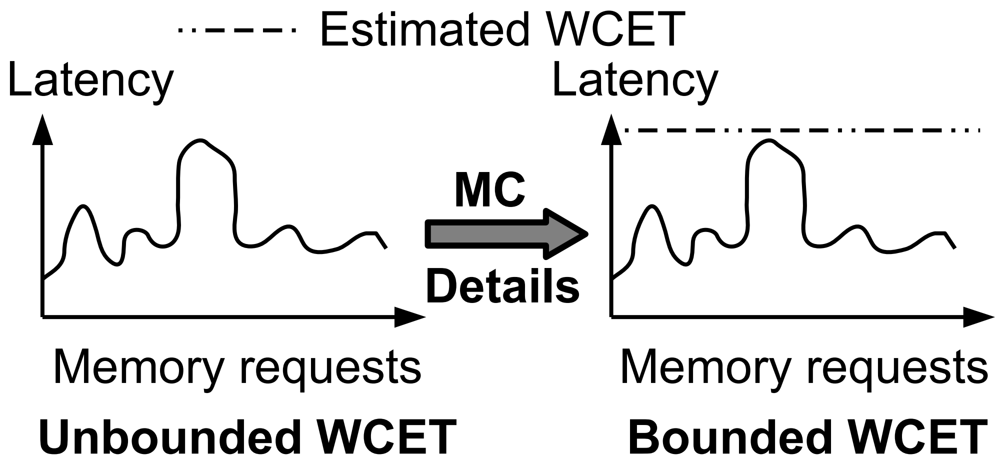
- COTS MC cannot be used for predictable real-time tasks *([DATE 2013], [CODESS 2007], [CODESS 2011], [ESL 2009])*



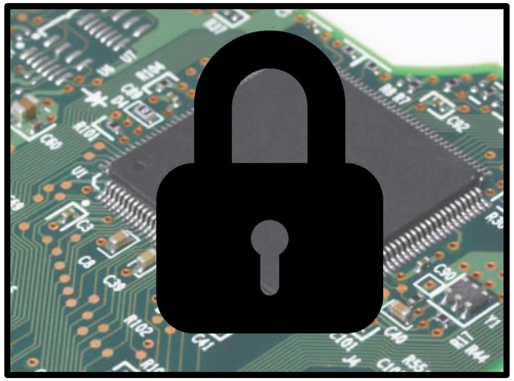
- Conflicting goals between predictability and performance



- Unbounded WCET for COTS MC *([RTAS 2014])*



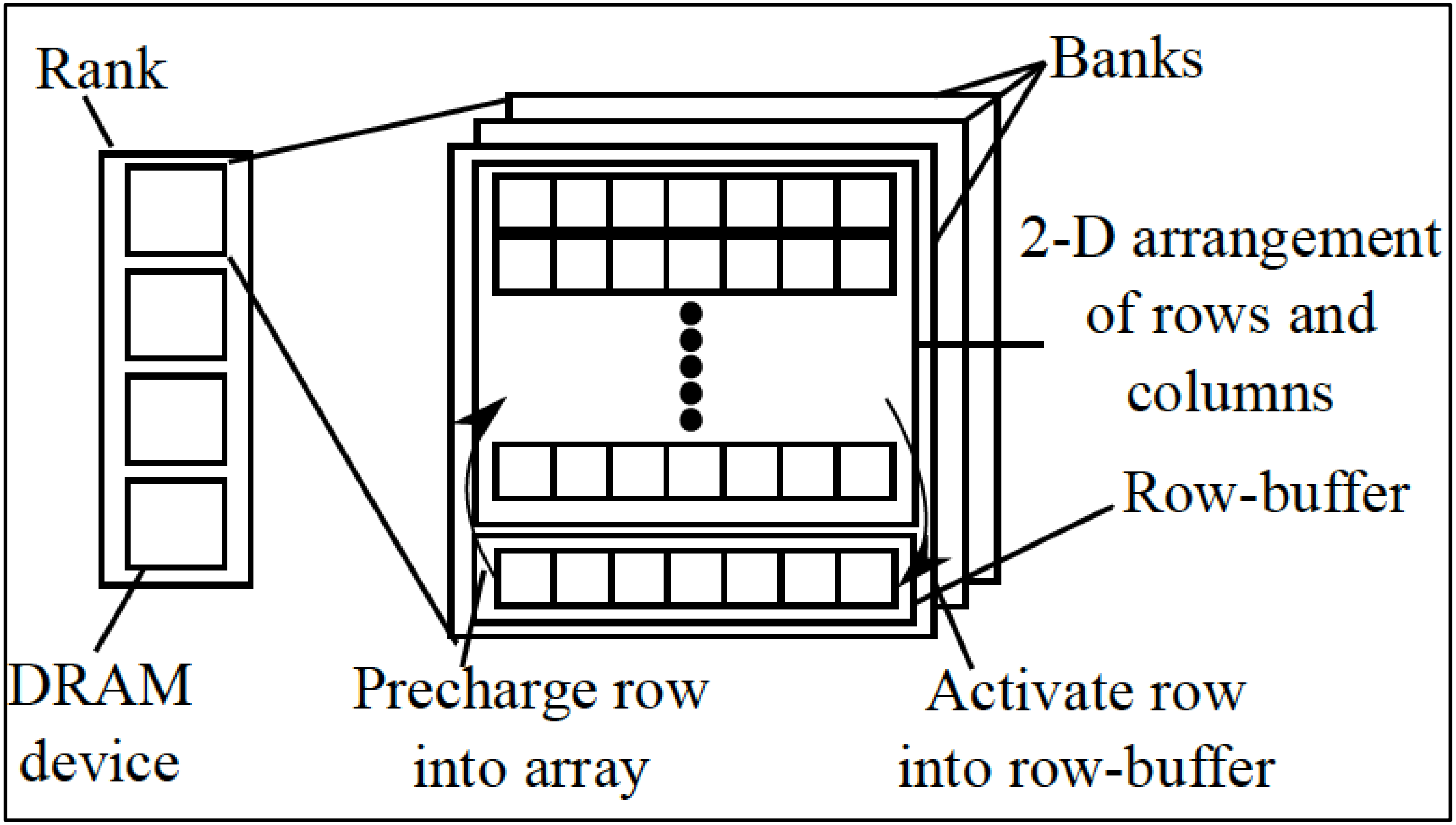
- COTS MC details are proprietary



Main contributions

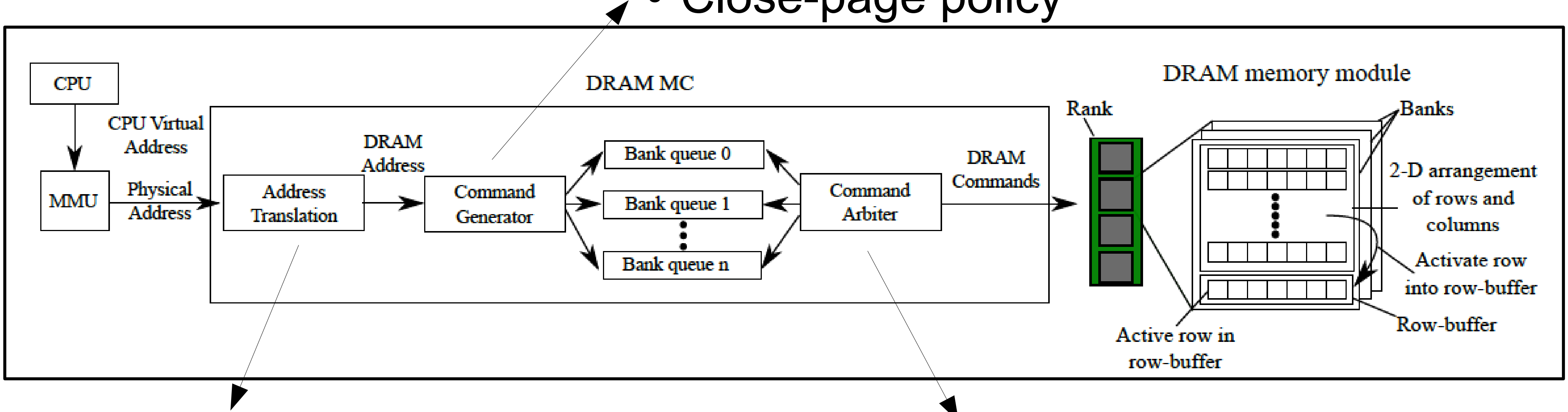
- We investigate whether we can reverse-engineer properties of the MC
  - Latency based analysis to understand DRAM memory accesses
  - Elicit worst-case and best-case bounds on access latencies for different DRAM memory access patterns
  - Develop inference rules to infer properties of MC using worst-case and best-case latency bounds
  - Validate inference rules on a set of commonly used DRAM MC configurations using a full system simulator

Background: DRAM architecture


Rank
Banks
2-D arrangement of rows and columns
Row-buffer
DRAM device
Precharge row into array
Activate row into row-buffer

Background: DRAM MC architecture

Page policies
- Open-page policy
- Close-page policy

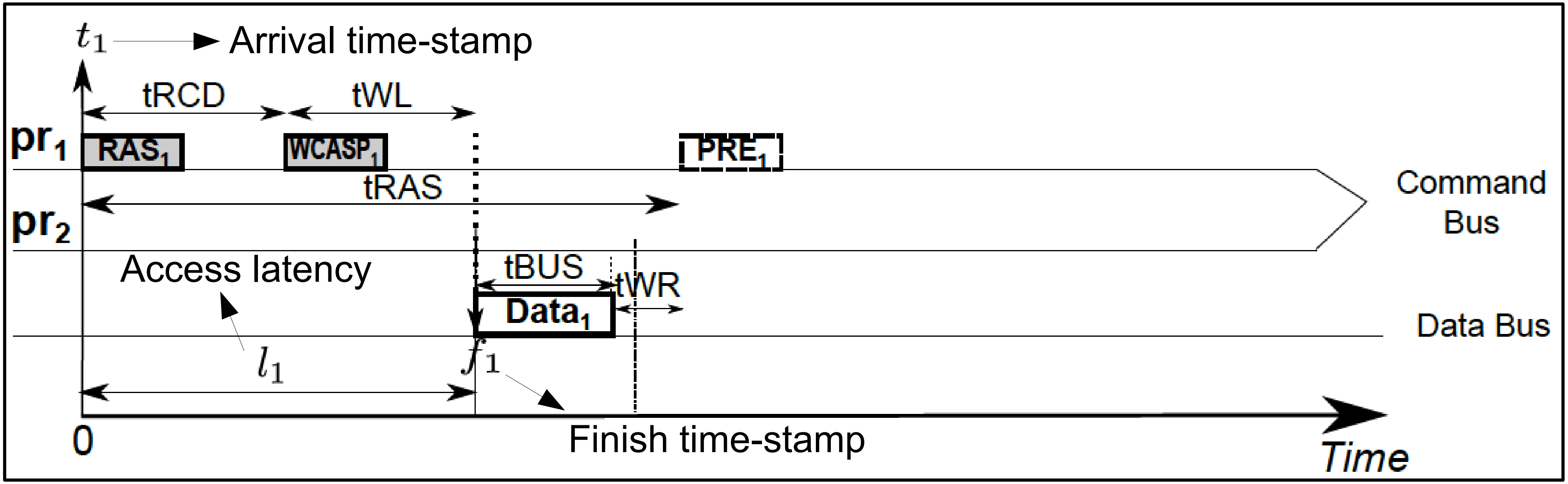


Address mapping schemes
- Baseline
  - <chn, rw, rnk, bnk, col>
  - <chn, rnk, bnk, col, row>
- XOR bank interleaving

Command arbitration schemes
- First-come-First-serve (FCFS/FIFO)
- Round-Robin (RR)
- First-Ready FCFS (FR-FCFS)

Background: Timing constraints

Write request ***pr*** $_1$ accessing DRAM



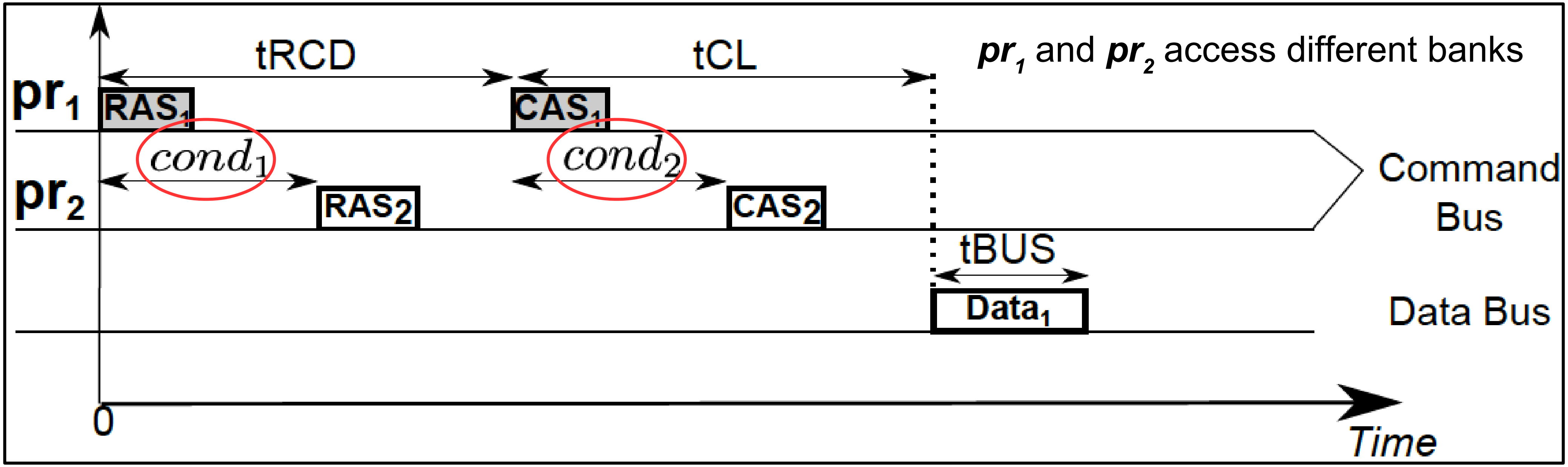
Parameter	Description
tCL (tWL/tRL)	Min time between CAS and data transfer
tRCD	RAS to CAS constraint to bring data into row-buffer
tBUS	Time to transfer data on the bus
tWR	Min time between data writing and PRE

- **RAS** : Row access strobe command
- **WCASP** : Column write command with auto-precharge
- **PRE** : Precharge command

UNIVERSITY OF WATERLOO

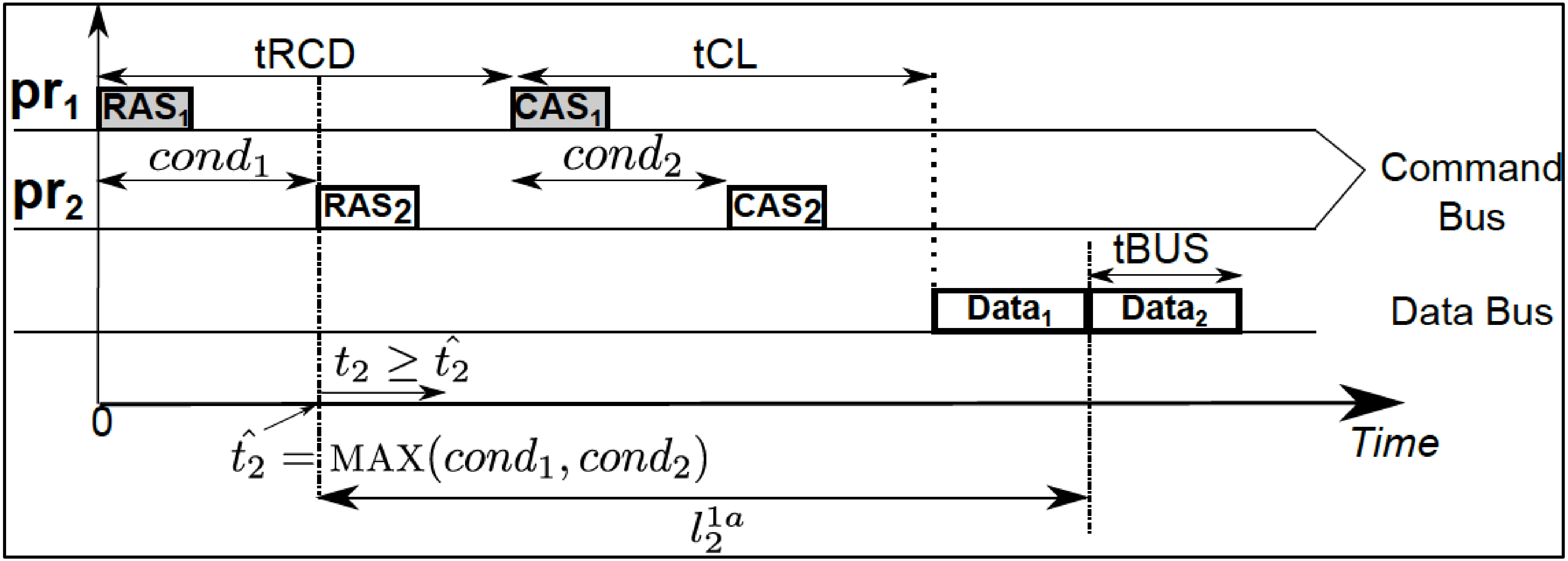
Latency analysis

Satisfy ***cond*$_1$** and ***cond*$_2$** for achieving best-case access latency for **pr$_2$**.



Latency analysis

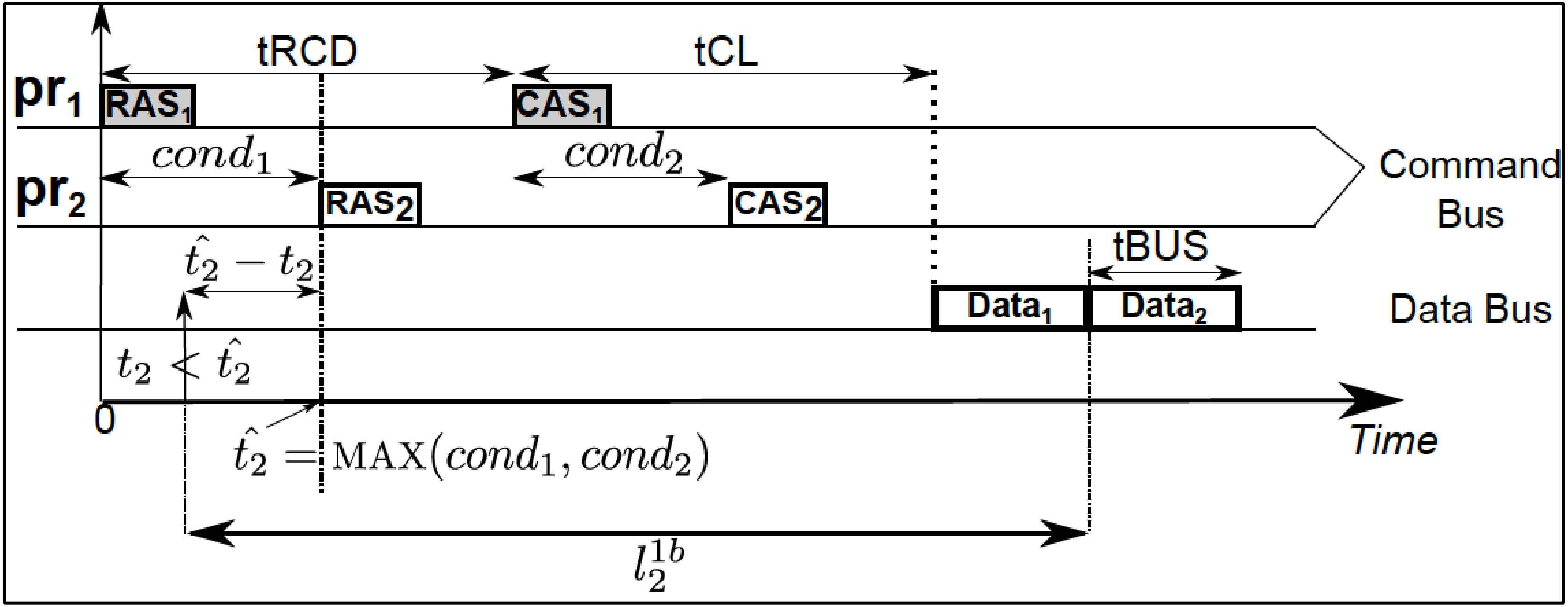
Case 1 : **pr$_2$** arrives after ***cond$_1$*** is satisfied i.e $t_2 \geq \hat{t}_2$



Best case access latency of **pr$_2$:** l_2^{1a}

Latency analysis

Case 2 : **pr$_2$** arrives before ***cond$_1$*** is satisfied i.e $t_2 < \hat{t}_2$

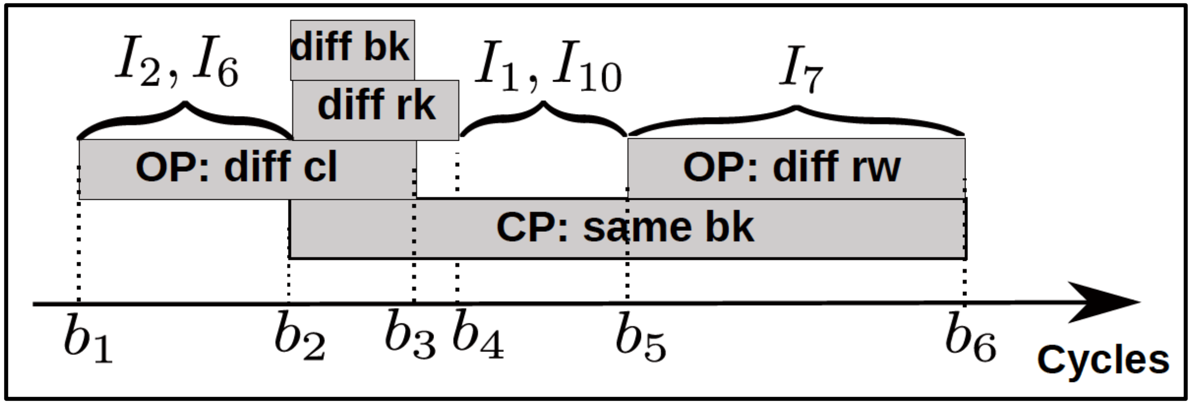


Access latency of **pr$_2$** : $l_2^{1b} = (\hat{t}_2 - t_2) + l_2^{1a}$

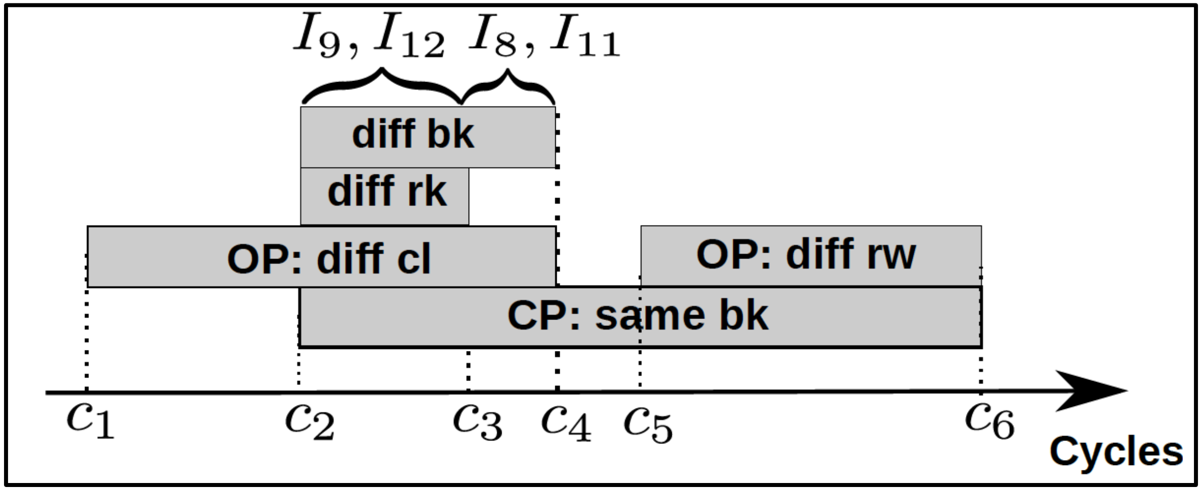
Worst-case access latency of **pr$_2$** : $\hat{t}_2 + l_2^{1a}$

Latency bounds

- Latency bounds for consecutive reads



- Latency bounds for write then read



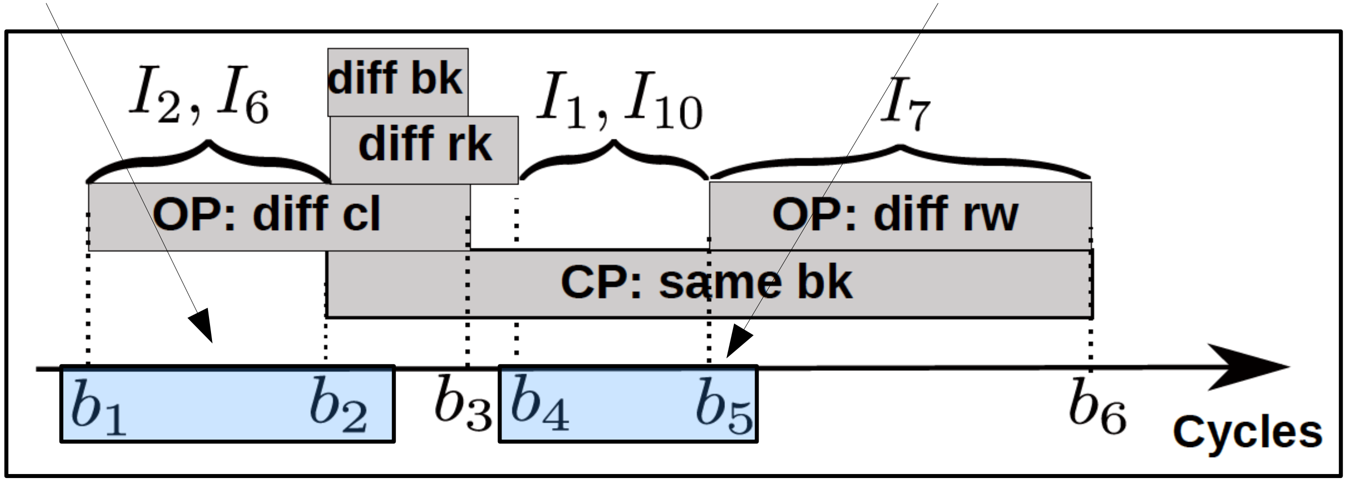
Notation	Cycles
b_1/c_1	tCL
b_2/c_2	tRCD + tCL
b_3	tRCD + tCL + tBUS
b_4	tRCD + tCL + tBUS + tRTRS
b_5/c_5	tRP + tRCD + tCL
b_6	tRC + tRCD + tCL
c_3	b_3 + tRTRS
c_4	tRCD + tCL + tWL + tBUS + tWTR
c_6	tRCD + tWL + tBUS + tWR + tRP + tRCD + tCL

Reverse-engineering MC: Intuition

Example : How to infer page-policy implemented by MC?

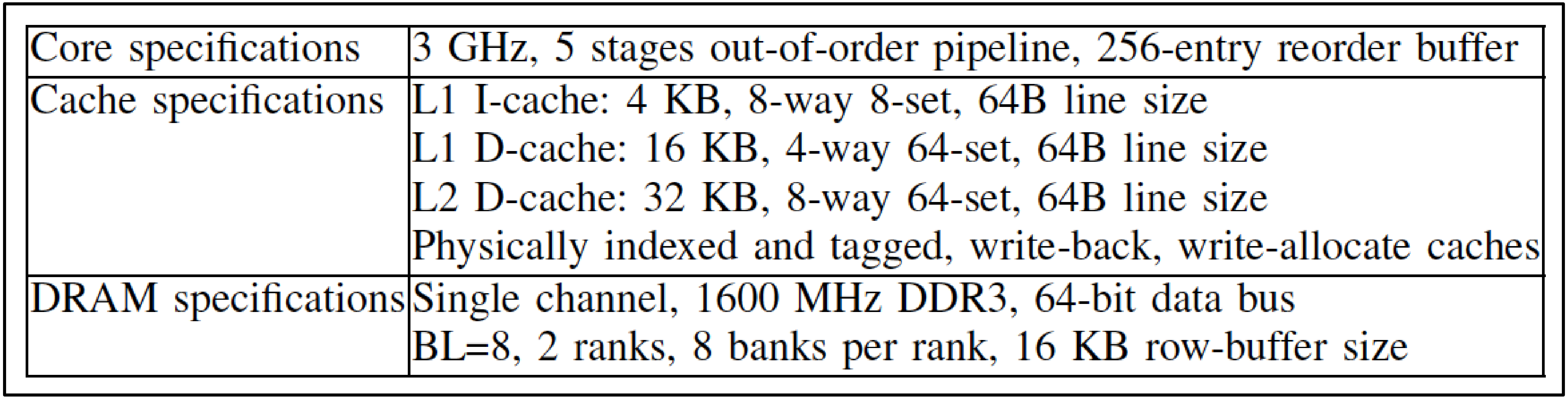
In open-page policy, successive read requests targeting same open row in row-buffer do not activate row-buffer

In close-page policy, successive read requests targeting same open row in row-buffer activate row-buffer

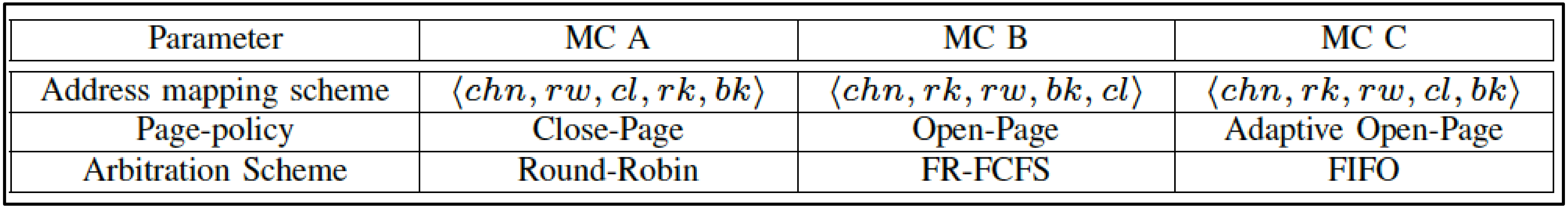


Different latency ranges for open and close-page policies

Experimental Framework



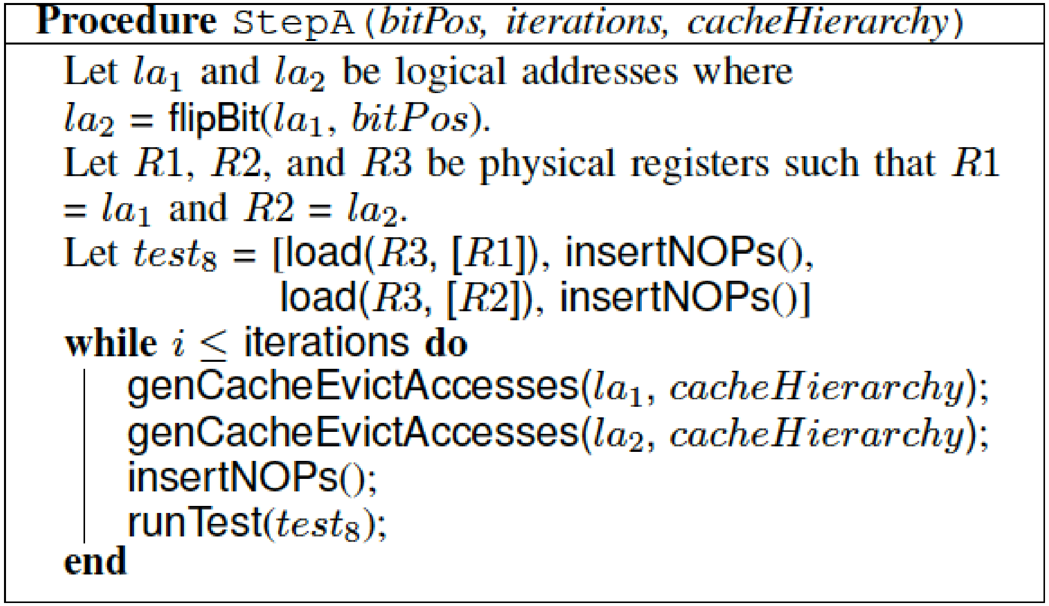
Core specifications	3 GHz, 5 stages out-of-order pipeline, 256-entry reorder buffer
Cache specifications	L1 I-cache: 4 KB, 8-way 8-set, 64B line size L1 D-cache: 16 KB, 4-way 64-set, 64B line size L2 D-cache: 32 KB, 8-way 64-set, 64B line size Physically indexed and tagged, write-back, write-allocate caches
DRAM specifications	Single channel, 1600 MHz DDR3, 64-bit data bus BL=8, 2 ranks, 8 banks per rank, 16 KB row-buffer size



Parameter	MC A	MC B	MC C
Address mapping scheme	$\langle chn, rw, cl, rk, bk \rangle$	$\langle chn, rk, rw, bk, cl \rangle$	$\langle chn, rk, rw, cl, bk \rangle$
Page-policy	Close-Page	Open-Page	Adaptive Open-Page
Arbitration Scheme	Round-Robin	FR-FCFS	FIFO

Assumption : Monitoring probes at MC to time-stamp start and finish times of requests

Benchmark Setup

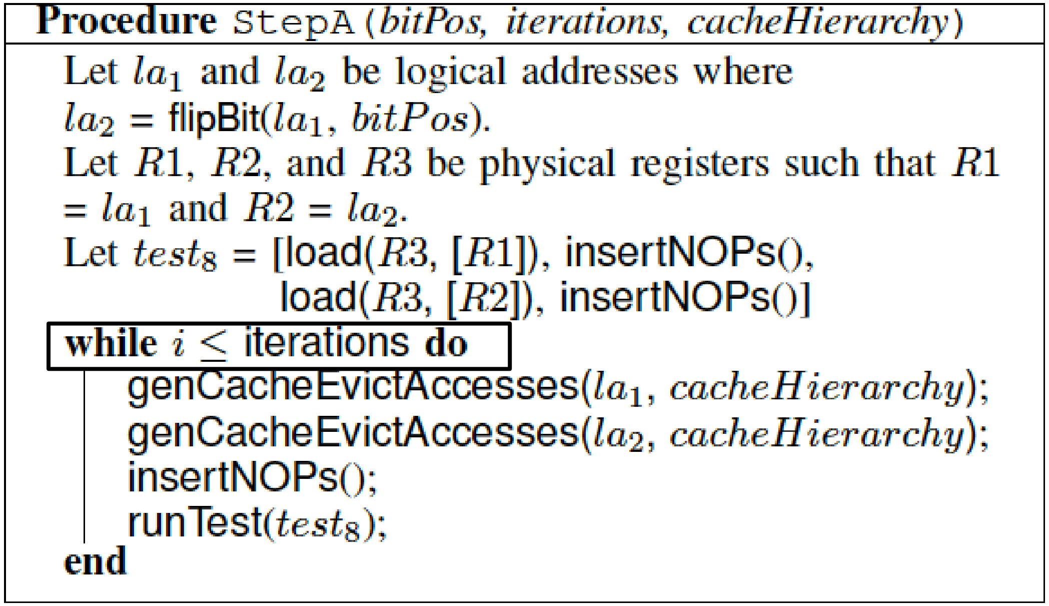


```
Procedure StepA(bitPos, iterations, cacheHierarchy)
  Let $la_1$ and $la_2$ be logical addresses where
  $la_2$ = flipBit($la_1$, $bitPos$).
  Let $R1$, $R2$, and $R3$ be physical registers such that $R1$
  = $la_1$ and $R2$ = $la_2$.
  Let $test_8$ = [load($R3$, [$R1$]), insertNOPs(),
                  load($R3$, [$R2$]), insertNOPs()]
  while $i \leq$ iterations do
      genCacheEvictAccesses($la_1$, $cacheHierarchy$);
      genCacheEvictAccesses($la_2$, $cacheHierarchy$);
      insertNOPs();
      runTest($test_8$);
  end
```

```
unsigned int iter = 0;
while(iter < 10000) {
  // Cache evictions
  asm volatile("movl %eax, offset_1(%rbx));
  asm volatile("movl %eax, offset_n(%rbx));
  asm volatile("movl %eax, offset_1(%rcx));
  asm volatile("movl %eax, offset_n(%rcx));
  asm volatile("movl %eax, offset_1(%rdx));
  asm volatile("movl %eax, offset_n(%rdx));
  asm volatile("nop"); x A times

  // Reverse-engineering requests
  asm volatile("movl (%rbx), %eax);
  asm volatile("nop"); x B times
  asm volatile("movl (%rcx), %eax);
  asm volatile("nop"); x C times
  iter++;
}
```

Benchmark Setup



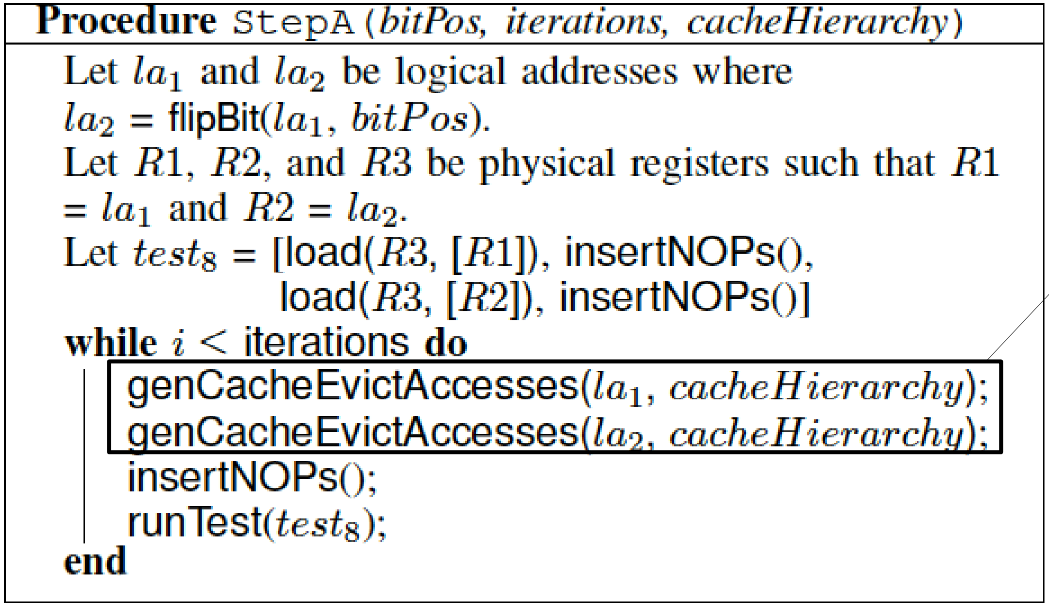
```
Procedure StepA(bitPos, iterations, cacheHierarchy)
  Let $la_1$ and $la_2$ be logical addresses where
  $la_2$ = flipBit($la_1$, $bitPos$).
  Let $R1$, $R2$, and $R3$ be physical registers such that $R1$
  = $la_1$ and $R2$ = $la_2$.
  Let $test_8$ = [load($R3$, [$R1$]), insertNOPs(),
                  load($R3$, [$R2$]), insertNOPs()]
  while $i \leq$ iterations do
      genCacheEvictAccesses($la_1$, $cacheHierarchy$);
      genCacheEvictAccesses($la_2$, $cacheHierarchy$);
      insertNOPs();
      runTest($test_8$);
  end
```

```
unsigned int iter = 0;
while(iter < 10000) {
  // Cache evictions
  asm volatile(“movl %eax, offset1(%rbx));
  asm volatile(“movl %eax, offsetn(%rbx));
  asm volatile(“movl %eax, offset1(%rcx));
  asm volatile(“movl %eax, offsetn(%rcx));
  asm volatile(“movl %eax, offset1(%rdx));
  asm volatile(“movl %eax, offsetn(%rdx));
  asm volatile(“nop”); x A times

  // Reverse-engineering requests
  asm volatile(“movl (%rbx), %eax);
  asm volatile(“nop”); x B times
  asm volatile(“movl (%rcx), %eax);
  asm volatile(“nop”); x C times
  iter++;
}
```

- Large number of iterations to extract stable access latencies

Benchmark Setup



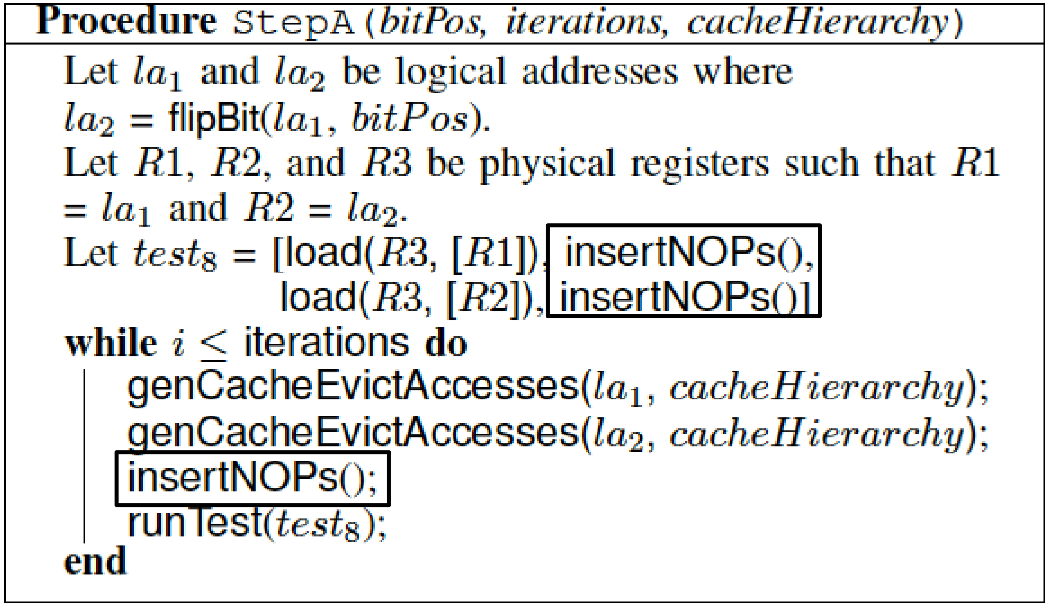
```
Procedure StepA(bitPos, iterations, cacheHierarchy)
  Let $la_1$ and $la_2$ be logical addresses where
  $la_2$ = flipBit($la_1$, $bitPos$).
  Let $R1$, $R2$, and $R3$ be physical registers such that $R1$
  = $la_1$ and $R2$ = $la_2$.
  Let $test_8$ = [load($R3$, [$R1$]), insertNOPs(),
                 load($R3$, [$R2$]), insertNOPs()]
  while $i <$ iterations do
      genCacheEvictAccesses($la_1$, $cacheHierarchy$);
      genCacheEvictAccesses($la_2$, $cacheHierarchy$);
      insertNOPs();
      runTest($test_8$);
  end
```

```
unsigned int iter = 0;
while(iter < 10000) {
  // Cache evictions
  asm volatile("movl %eax, offset_1(%rbx));
  asm volatile("movl %eax, offset_n(%rbx));
  asm volatile("movl %eax, offset_1(%rcx));
  asm volatile("movl %eax, offset_n(%rcx));
  asm volatile("movl %eax, offset_1(%rdx));
  asm volatile("movl %eax, offset_n(%rdx));
  asm volatile("nop"); x A times

  // Reverse-engineering requests
  asm volatile("movl (%rbx), %eax);
  asm volatile("nop"); x B times
  asm volatile("movl (%rcx), %eax);
  asm volatile("nop"); x C times
  iter++;
}
```

- Cache evictions to evict reverse-engineering requests from MC
- Details about the cache hierarchy known apriori or inferred
  - Cache hierarchy details can be reverse-engineered using [RTAS 2013]

Benchmark Setup



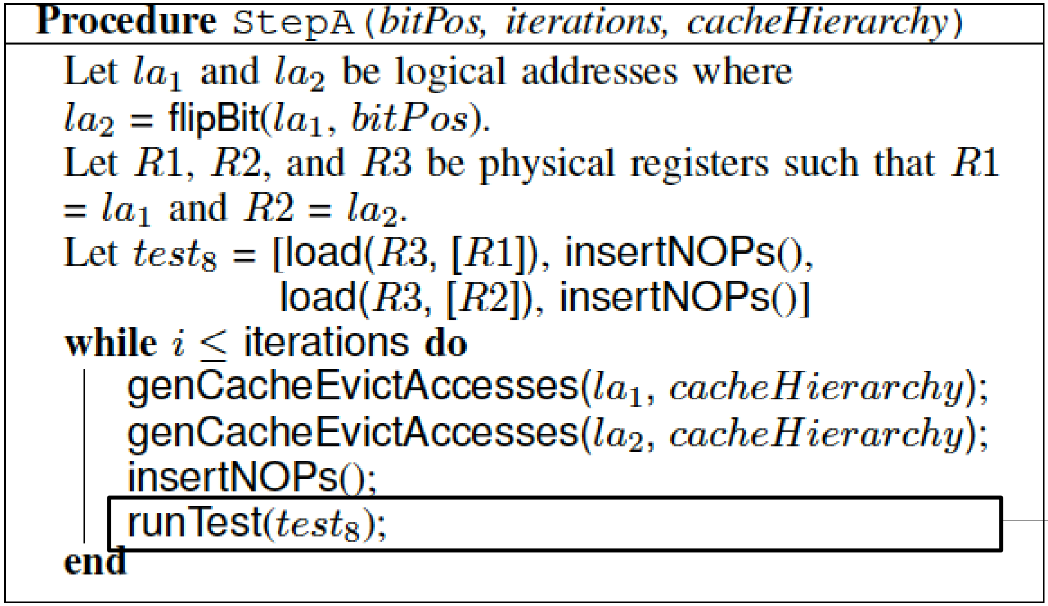
```
Procedure StepA(bitPos, iterations, cacheHierarchy)
  Let $la_1$ and $la_2$ be logical addresses where
  $la_2$ = flipBit($la_1$, $bitPos$).
  Let $R1$, $R2$, and $R3$ be physical registers such that $R1$
  = $la_1$ and $R2$ = $la_2$.
  Let $test_8$ = [load($R3$, [$R1$]), insertNOPs(),
                  load($R3$, [$R2$]), insertNOPs()]
  while $i \leq$ iterations do
      genCacheEvictAccesses($la_1$, $cacheHierarchy$);
      genCacheEvictAccesses($la_2$, $cacheHierarchy$);
      insertNOPs();
      runTest($test_8$);
  end
```

```
unsigned int iter = 0;
while(iter < 10000) {
   // Cache evictions
   asm volatile("movl %eax, offset_1(%rbx));
   asm volatile("movl %eax, offset_n(%rbx));
   asm volatile("movl %eax, offset_1(%rcx));
   asm volatile("movl %eax, offset_n(%rcx));
   asm volatile("movl %eax, offset_1(%rdx));
   asm volatile("movl %eax, offset_n(%rdx));
   asm volatile("nop"); x A times

   // Reverse-engineering requests
   asm volatile("movl (%rbx), %eax);
   asm volatile("nop"); x B times
   asm volatile("movl (%rcx), %eax);
   asm volatile("nop"); x C times
   iter++;
}
```

- NOPs inserted to vary arrival time of requests to MC
  - Key to achieving best-case and worst-case access latencies
- NOPs inserted to flush memory buffers of previous requests
- Iterative approach to determining number of NOP instructions

Benchmark Setup



```
Procedure StepA (bitPos, iterations, cacheHierarchy)
  Let $la_1$ and $la_2$ be logical addresses where
  $la_2$ = flipBit($la_1$, $bitPos$).
  Let $R1$, $R2$, and $R3$ be physical registers such that $R1$
  = $la_1$ and $R2$ = $la_2$.
  Let $test_8$ = [load($R3$, [$R1$]), insertNOPs(),
                  load($R3$, [$R2$]), insertNOPs()]
  while $i \leq$ iterations do
      genCacheEvictAccesses($la_1$, $cacheHierarchy$);
      genCacheEvictAccesses($la_2$, $cacheHierarchy$);
      insertNOPs();
      runTest($test_8$);
  end
```

```
unsigned int iter = 0;
while(iter < 10000) {
  // Cache evictions
  asm volatile(“movl %eax, offset_1(%rbx));
  asm volatile(“movl %eax, offset_n(%rbx));
  asm volatile(“movl %eax, offset_1(%rcx));
  asm volatile(“movl %eax, offset_n(%rcx));
  asm volatile(“movl %eax, offset_1(%rdx));
  asm volatile(“movl %eax, offset_n(%rdx));
  asm volatile(“nop”); x A times

  // Reverse-engineering requests
  asm volatile(“movl (%rbx), %eax);
  asm volatile(“nop”); x B times
  asm volatile(“movl (%rcx), %eax);
  asm volatile(“nop”); x C times
  iter++;
}
```

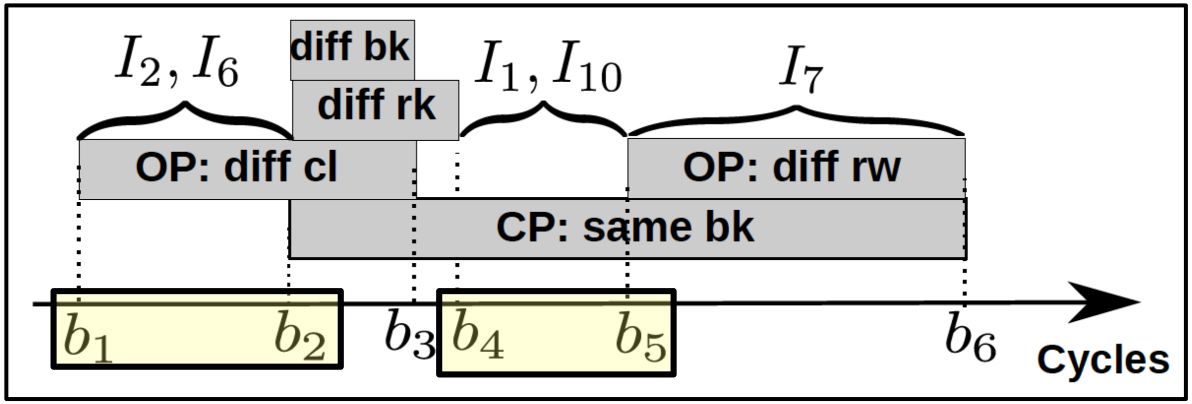
- Data hazard inserted to avoid instruction reordering for OoO pipeline
  - RAW hazard on register **eax** for inferring bank and rank bits
  - WAW hazard on register **eax** for inferring rest of the properties

Reverse-engineering MC

- 3-step process
  - Step 1: Reverse-engineer page-policy
  - Step 2: Reverse-engineer address mapping scheme
  - Step 3: Reverse-engineer command arbitration scheme

Reverse-engineering MC: Open/Close-page policy (Step 1)

Observation : Two non-overlapping ranges for consecutive read requests.



Notation	Cycles
b_1	10
b_2	20
b_4	25
b_5	30

Methodology : **test**$_1$: ***Read [addr$_1$] NOP() Read [addr$_2$]***

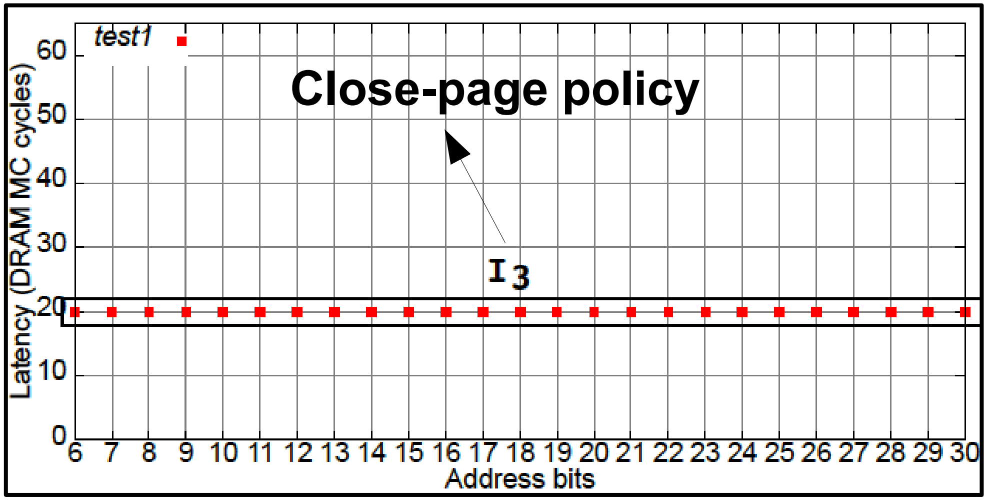
Execute **test**$_1$ for ***PW*** times with ***addr$_2$ = flipBit(addr$_1$, i)***

Inference rules :

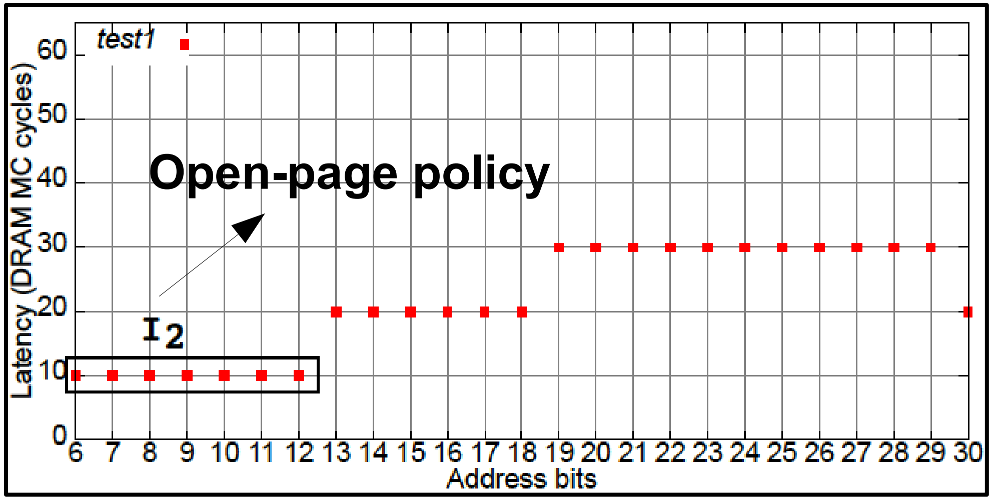
$$(I_1)\exists i \in [0, PW-1] : b_4 < l_2^i < b_5 \Rightarrow Close - page\ policy$$
$$(I_2)\exists i \in [0, PW-1] : b_1 \leq l_2^i < b_2 \Rightarrow Open - page\ policy$$
$$(I_3)\forall i \in [0, PW-1] : l_2^i = b_2 \Rightarrow Close - page\ policy$$

Results: Inferring Open/Close page-policy

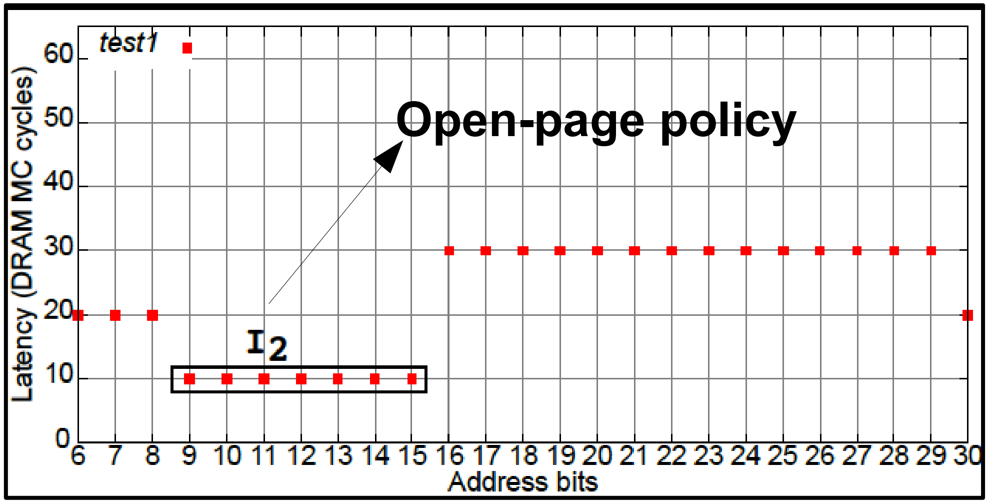
MC A



MC B

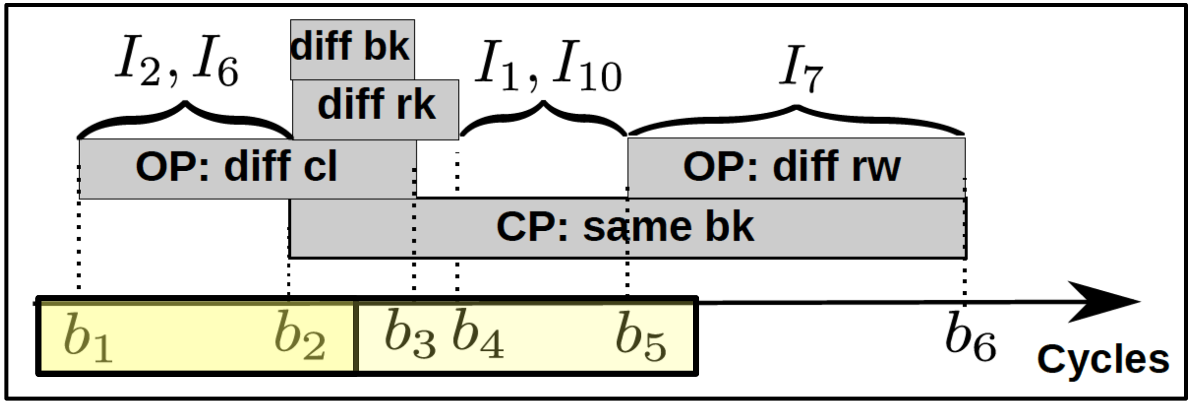


MC C



Reverse-engineering MC: Adaptive Open-page policy

Observation : Row-buffer keeps row open for a longer time when successive requests target different columns to the same row, and vice-versa when successive requests target different rows to the same bank.



Notation	Cycles
b_2	20
b_5	30

Methodology : test : Execute two sequences of requests

Sequence$_1$: Target different rows to same bank and rank

Read [addr$_1$] NOP Read[addr$_2$] NOP Read[addr$_n$]

Sequence$_2$:Target different columns to same rank, bank, and row

Read [addr$_{n+1}$] NOP Read[addr$_{n+2}$] NOP Read[addr$_{2n}$]

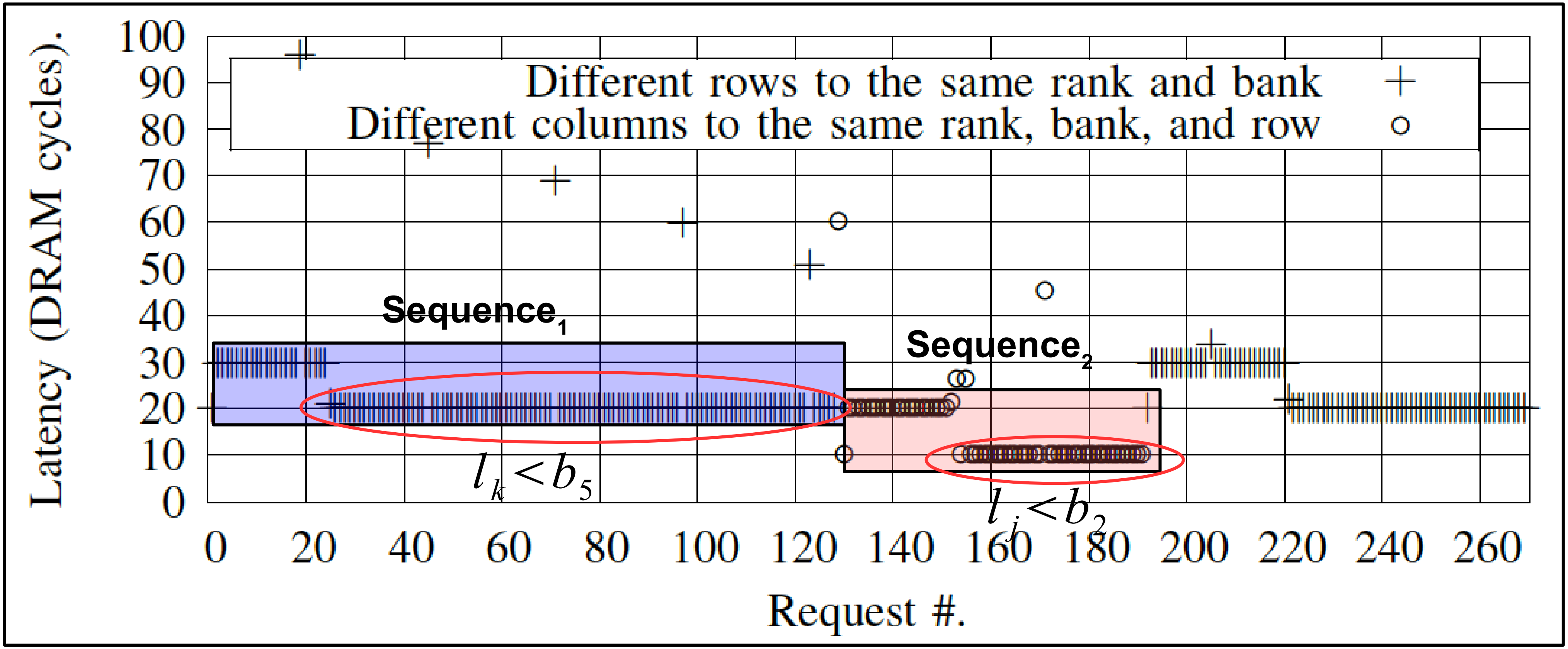
Inference rule :

$$(I_4)\exists k\in[1,n],\, j\in[n+1,2\mathrm{n}]:(l_k<b_5)\wedge(l_j<b_2)\Rightarrow Adaptive\,Open-page\ policy$$

UNIVERSITY OF WATERLOO

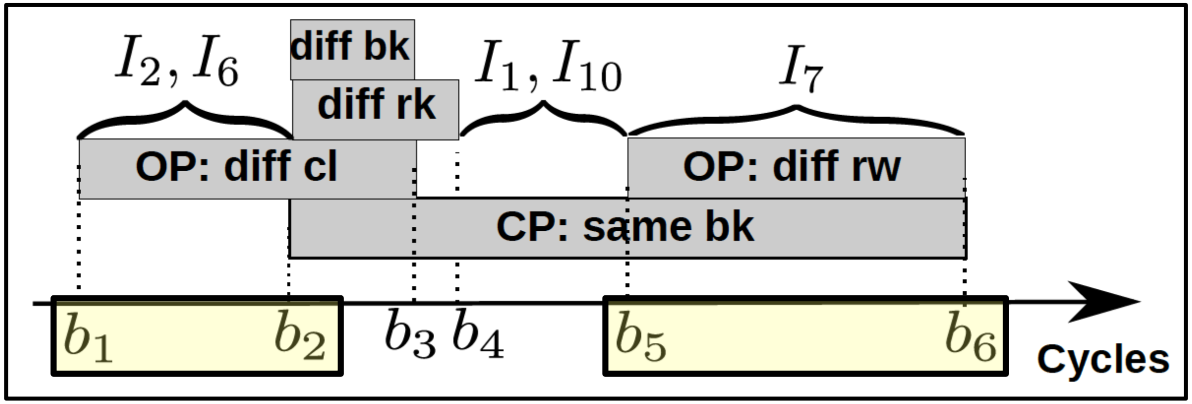
Results: Inferring Adaptive Open-page policy

MC C



Reverse-engineering MC: Address mapping for Open-page policy (Step 2)

Observation : Minimum latency for requests targeting open row in row-buffer and maximum latency for requests targeting different rows to the same bank and rank.



Notation	Cycles
b_1	10
b_2	20
b_5	30
b_6	41

Methodology : **test$_1$** : ***Read [addr$_1$] NOP()   Read [addr$_2$]***

Execute **test$_1$** for ***PW*** times with ***addr$_2$ = flipBit(addr$_1$, i)***

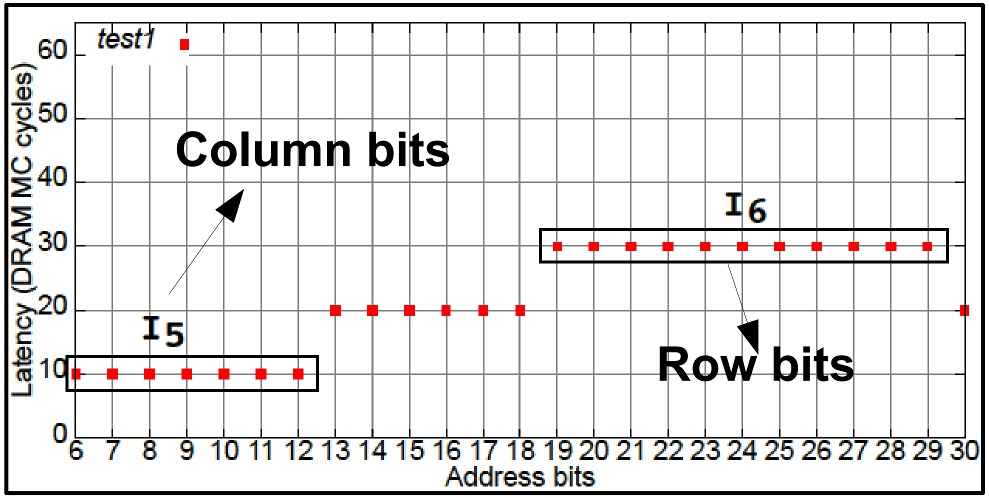
Inference rules :

$$(I_5)\forall i \in [0, PW-1] : b_1 \le l_2^i < b_2 \Rightarrow i\ is\ column\ bit$$

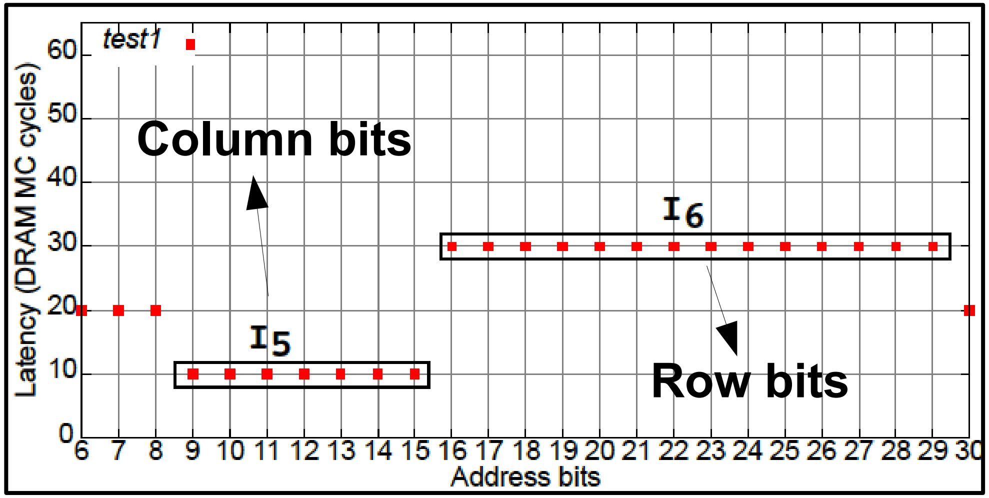
$$(I_6)\forall i \in [0, PW-1] : b_5 \le l_2^i \le b_6 \Rightarrow i\ is\ row\ bit$$

Results: Inferring column and row bits for Open-page policy

MC B

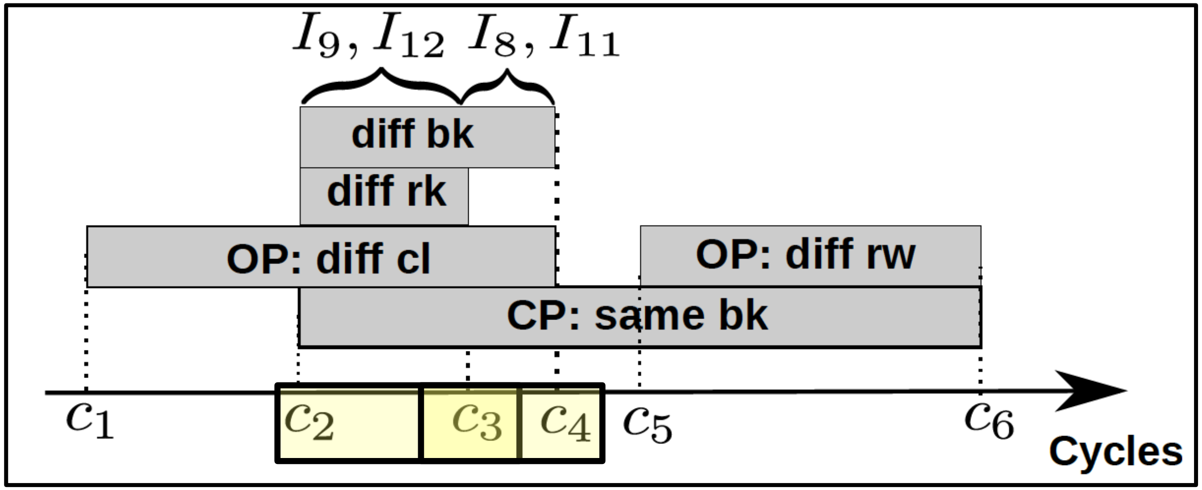


MC C



Reverse-engineering MC: Address mapping for Open-page policy (Step 2)

Observation : Shared data bus for banks in a rank. Switching delay to change bus direction from write to read and vice-versa. Switching overhead when changing between ranks.



Notation	Cycles
c_2	20
c_3	24
c_4	54

Methodology : **test$_2$** : ***Write [addr$_1$] NOP() Read [addr$_2$]***

Execute **test$_2$** for ***PW*** times with ***addr$_2$ = flipBit(addr$_1$, i)***

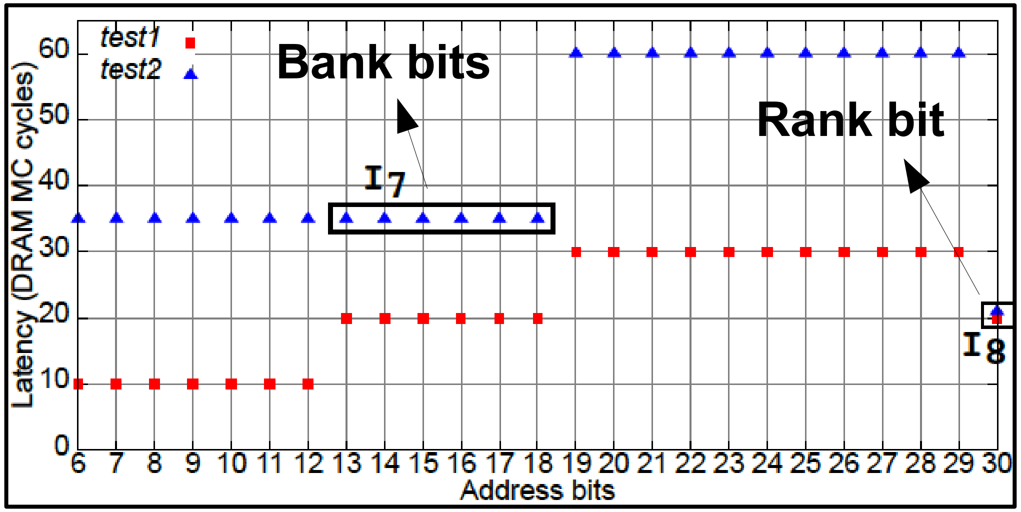
Inference rules :

$$(I_7)\,\forall\, i \in [0, PW-1] : (i \text{ is not column bit}) \wedge (c_3 < l_2^i < c_4) \Rightarrow i \text{ is bank bit}$$

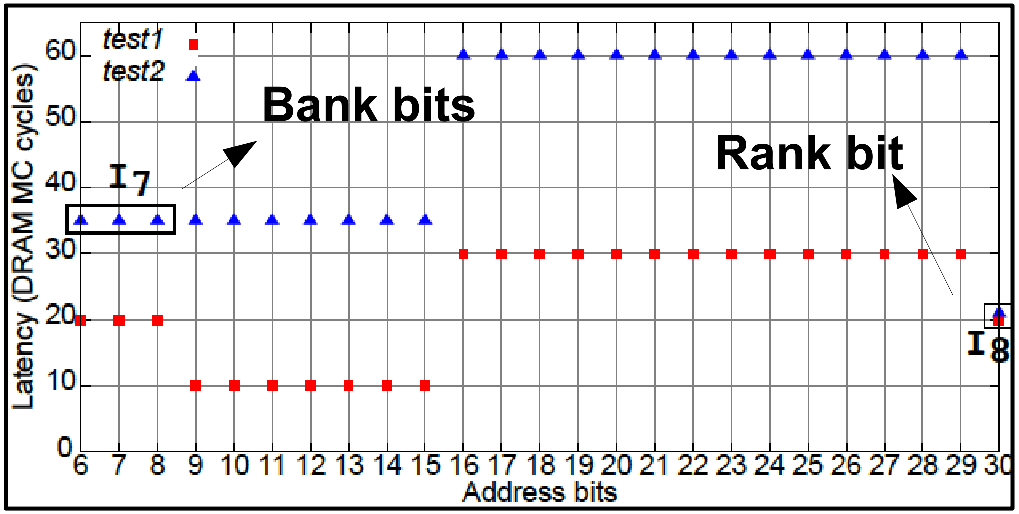
$$(I_8)\,\forall\, i \in [0, PW-1] : (i \text{ is not column} \vee \text{bank bit}) \wedge (c_2 < l_2^i < c_3) \Rightarrow i \text{ is rank bit}$$

Results: Inferring rank and bank bits for Open-page policy

MC B



MC C



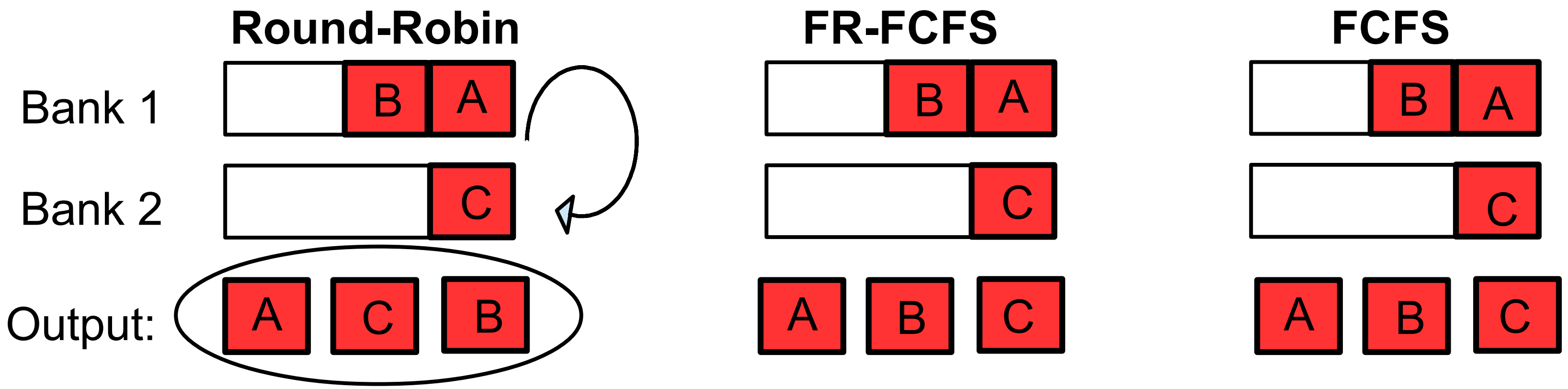
More bits than number of banks inferred as bank bits for MC B
=> XOR bank interleaving

Reverse-engineering MC: Command arbitration schemes (Step 3)

test$_4$ Input : A B C

A and B target same bank and different rows

C targets different bank



Round-Robin

Bank 1 B A

Bank 2 C

Output: A C B

FR-FCFS

B A

C

A B C

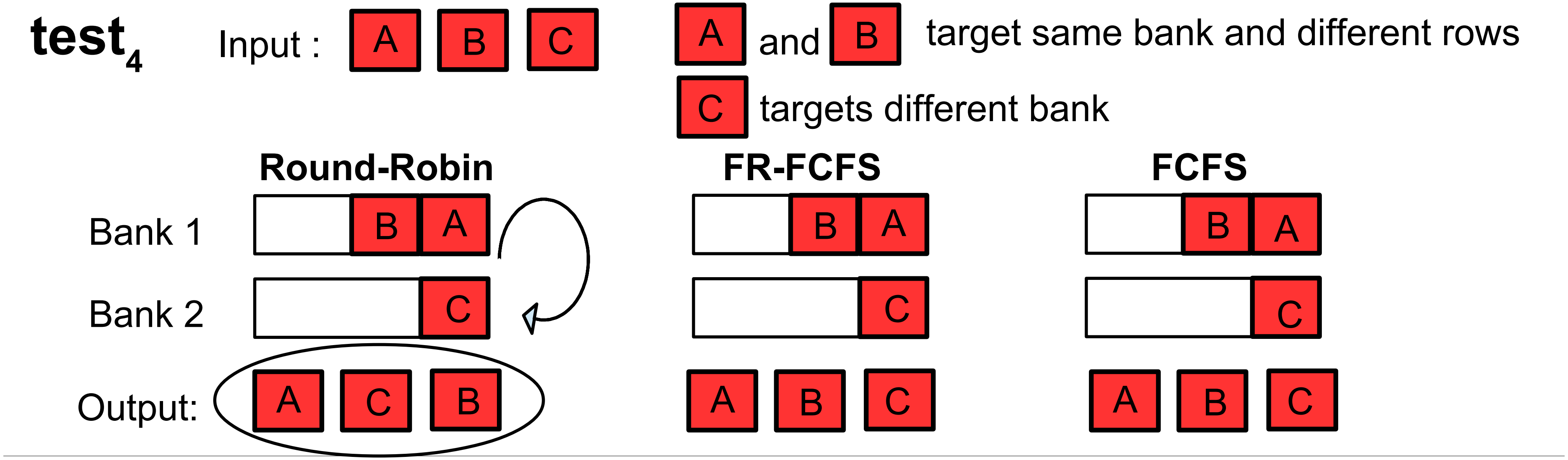
FCFS

B A

C

A B C

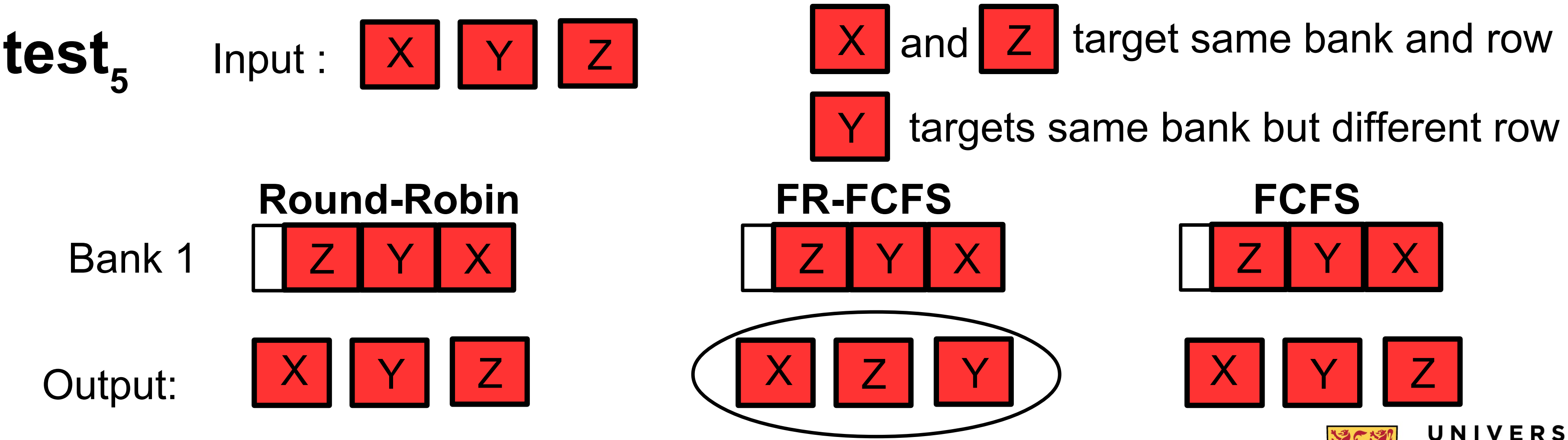
Reverse-engineering MC: Command arbitration schemes (Step 3)



test_4 Input : A B C

A and B target same bank and different rows

C targets different bank



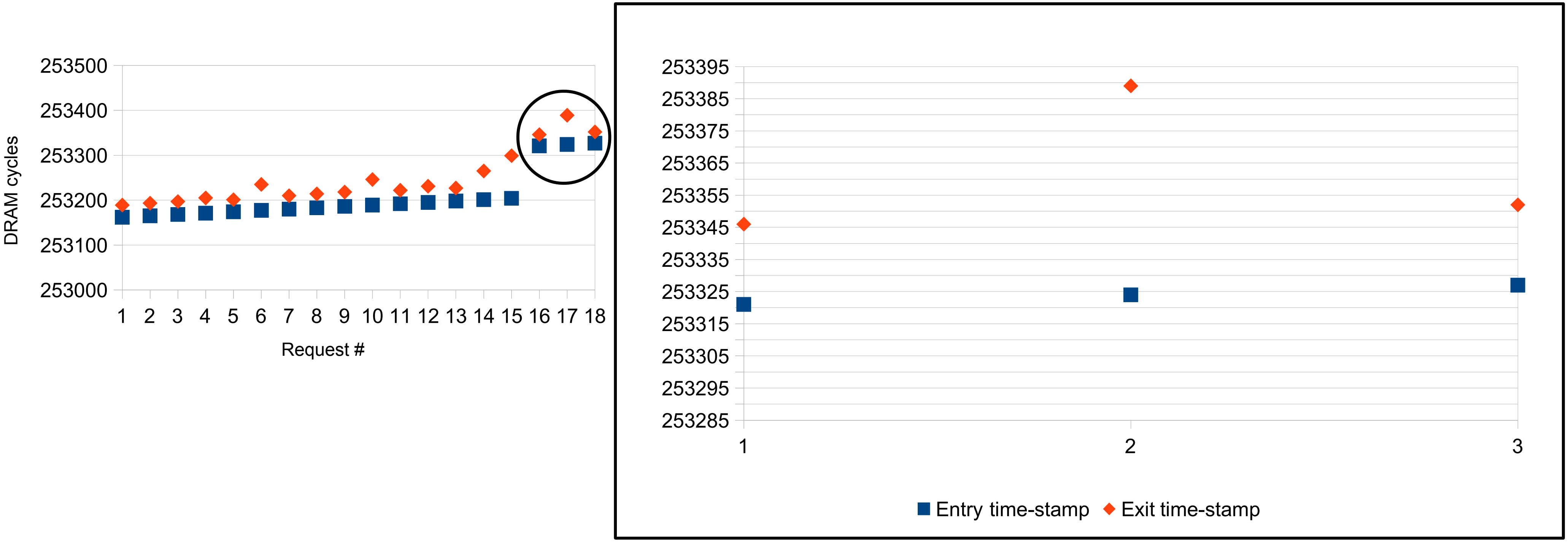
test_5 Input : X Y Z

X and Z target same bank and row

Y targets same bank but different row

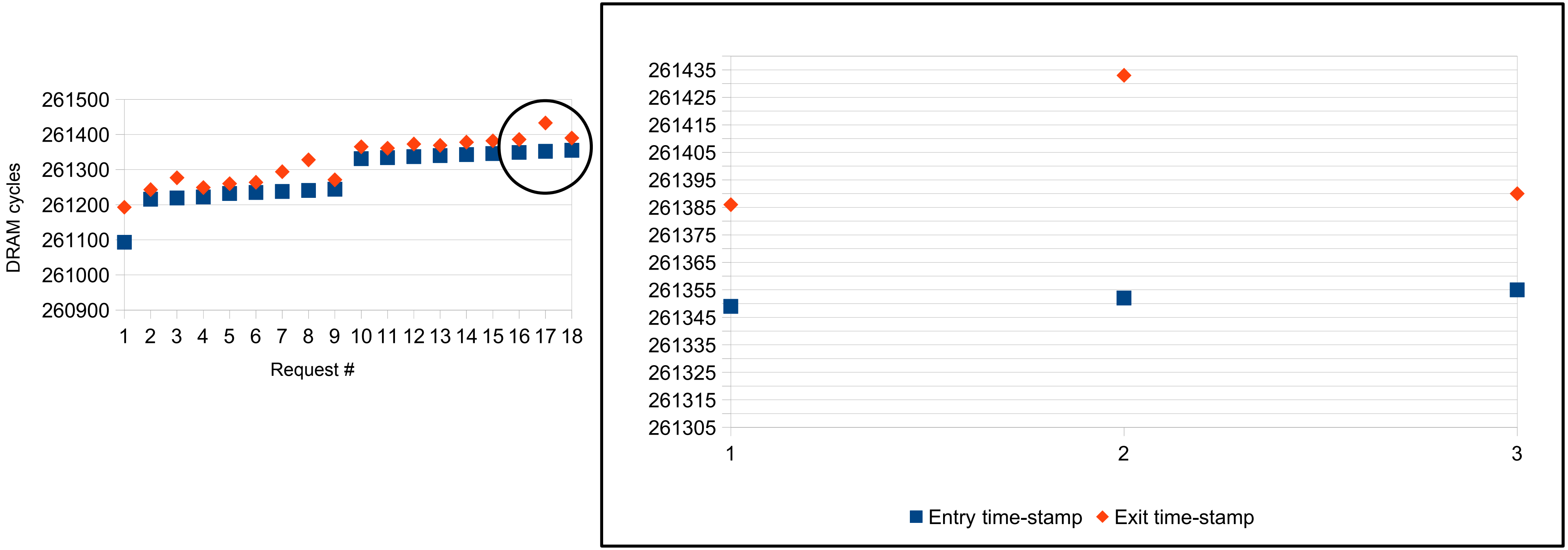
Results: Inferring command arbitration schemes

MC A: Executing test_4 on Round-Robin



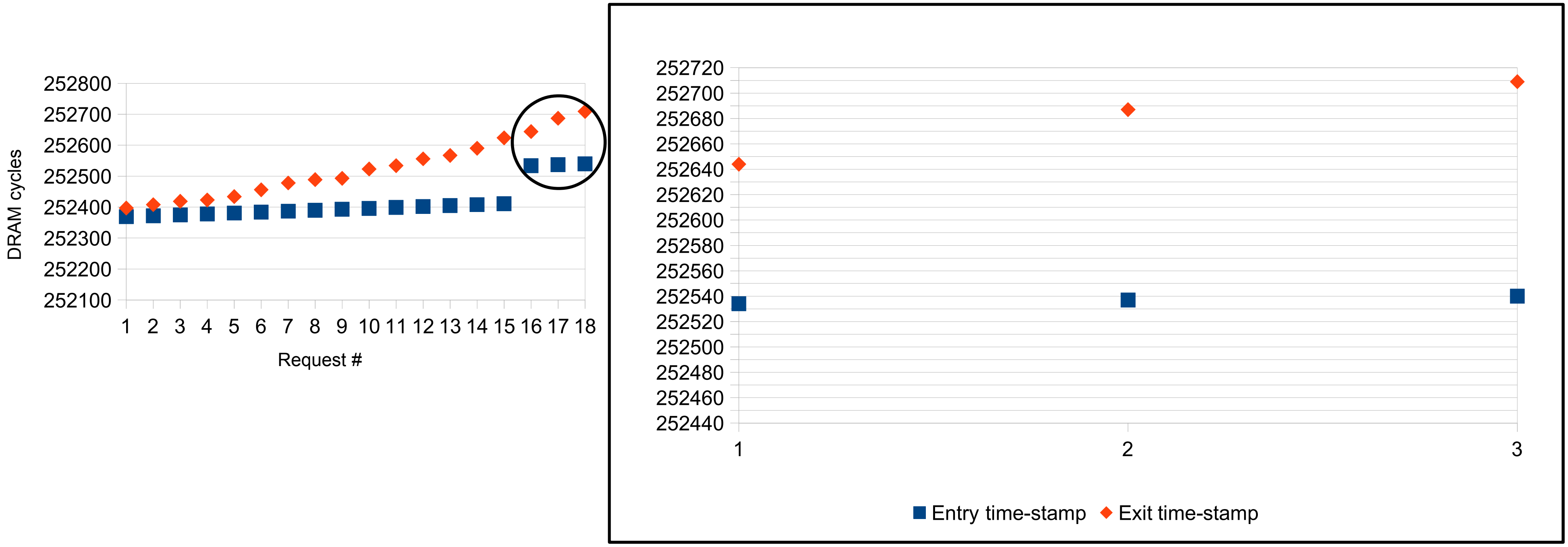
Results: Inferring command arbitration schemes

MC B: Executing test_5 on FR-FCFS



Results: Inferring command arbitration schemes

MC C: Executing test_4 and test_5 on FCFS/FIFO



Summary of MC properties

- **Address mapping schemes**
  - ✓ Baseline
  - ✓ XOR bank interleaving
- **Page policies**
  - ✓ Open-Page
  - ✓ Close-Page
  - ✓ Adaptive Open-Page
- **Arbitration schemes**
  - ✓ FCFS/FIFO
  - ✓ Round-Robin
  - ✓ FR-FCFS
- ✓ **FR-FCFS threshold**

Conclusion and future work

- Shown that we can reverse-engineer some key properties of common MC configurations using latency based analysis
- Future work include
  - Extending analysis to include more MC properties
  - Applying inference rules and methodology on real hardware platforms

Thank you!

Questions?

References

- [DATE 2013]: S. Goossens, B. Akesson, and K.Goossens, “Conservative open-page policy for mixed time-criticality memory controllers,” in Design Automation Test in Europe Conference Exhibition (DATE), 2013
- [CODESS 2007]: B. Akesson, K. Goossens, and M. Ringhofer, “Predator: A predictable SDRAM memory controller,” in Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis.
- [CODESS 2011]: J. Reineke, I. Liu, H. Patel, S. Kim, and E. A. Lee, “PRET DRAM controller: Bank privatization for predictability and temporal isolation,” in Proceedings of the Seventh International Conference on Hardware/Software Codesign and System Synthesis.
- [ESL 2009]: M. Paolieri, E. Quiones, F. Cazorla, and M. Valero, “An analyzable memory controller for hard real-time CMPs,” Embedded Systems Letters, 2009
- [RTAS 2013]: A. Abel and J. Reineke, “Measurement-based modeling of the cache replacement policy,” 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)
- [RTAS 2014]: H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. R.Rajkumar, “Bounding memory interference delay in COTS-based multicore systems,” Technical Report CMU/SEI-2014-TR-003, Software Engineering Institute, Carnegie Mellon University, Tech. Rep., 2014.

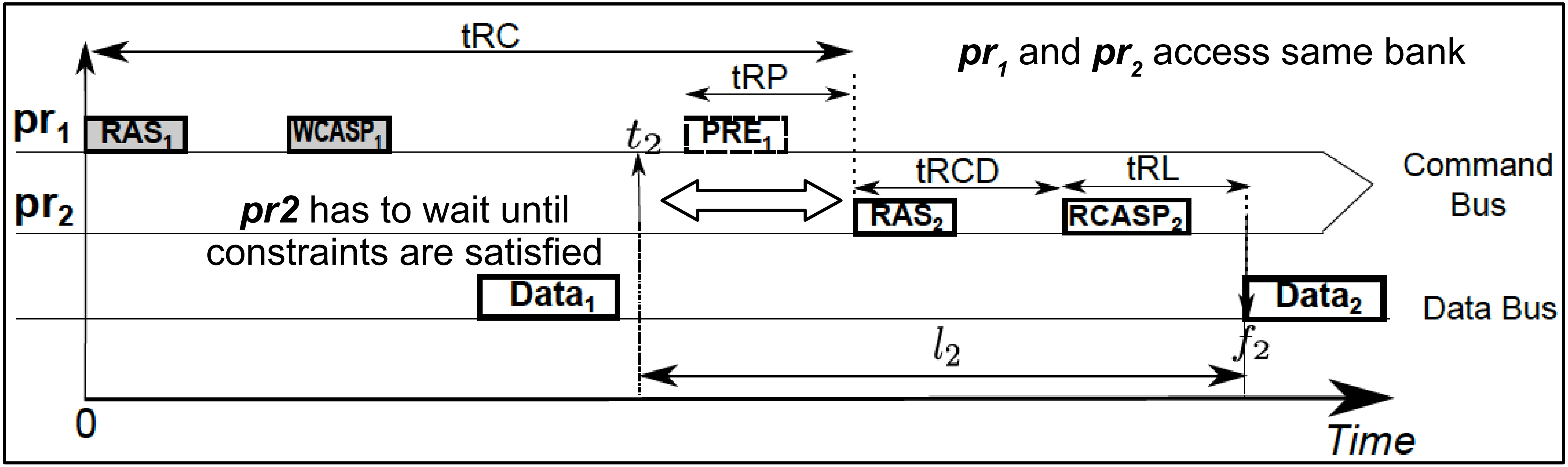
Backup

Related work

- **Measurement based analysis**
  - Inferring cache hierarchy details
    - Performance counters : Abel and Reineke [2013], Dongarra et al. [2004], John and Baumgartl [2007].
    - Latency based analysis : Yotov et al. [2005], Wong et al. [2010], Thomborson and Yu [2000].
  - Inferring DRAM MC details
    - Some properties of address mapping scheme : Yun et al. [2014], Park et al. [2013]
- **Hardware customizations to DRAM MC**
  - For achieving predictability
    - Conservative open-page by Goosens et al. [2013], Private banks by Reineke et al. [2011], Predictable command arbitration schemes by Akesson et al. [2007]
  - For achieving performance
    - XOR bank interleaving by Zhang et al. [2000], FR-FCFS scheduling by Rixner et al. [2000]
- **Software customizations**
  - For achieving predictability Potential future work
  - For achieving performance
    - Random page allocator by Park et al. [2013], Improving row-buffer hits for irregular applications Ding et al. [2014]

Background : Timing constraints

Read request ***pr***$_2$ following write request ***pr***$_1$ accessing DRAM

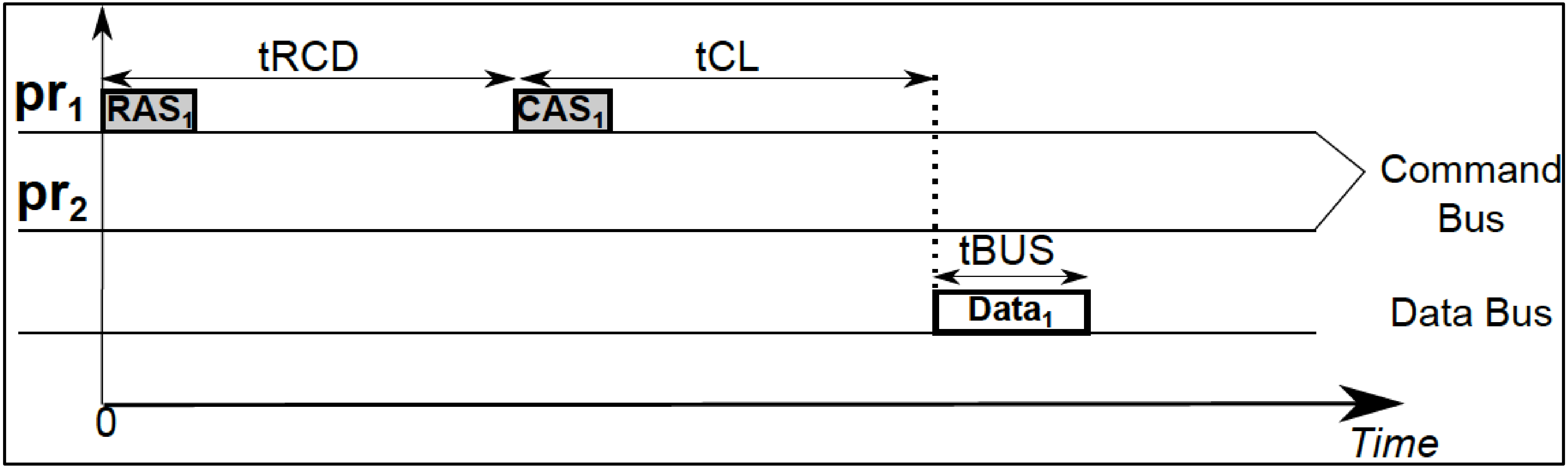


Parameter	Description
tRP	Min time between PRE and following RAS command
tRC	Minimum time between two RAS commands to same bank

- **RAS** : Row access strobe command
- **RCASP** : Column read command with auto-precharge
- **PRE** : Precharge command

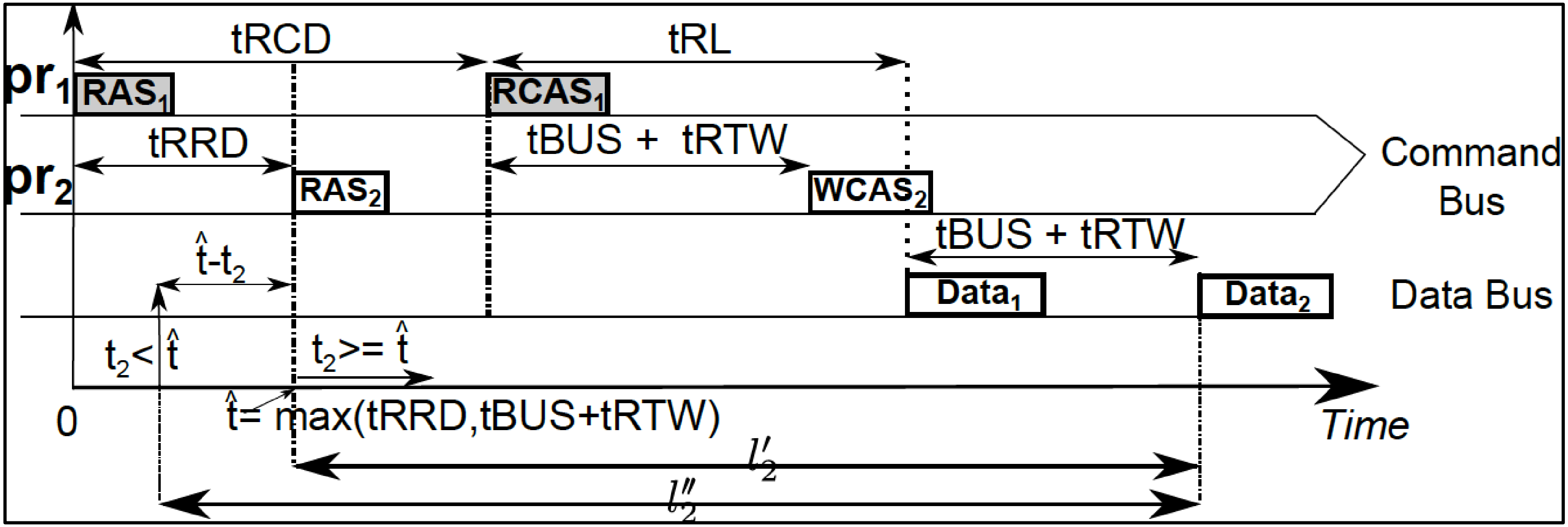
Latency analysis

Worst-case access latency : Maximum access latency for a memory request
Best-case access latency : Minimum access latency for a memory request



Latency analysis : Example

- Open/Close-page policy, different banks, read followed by write request

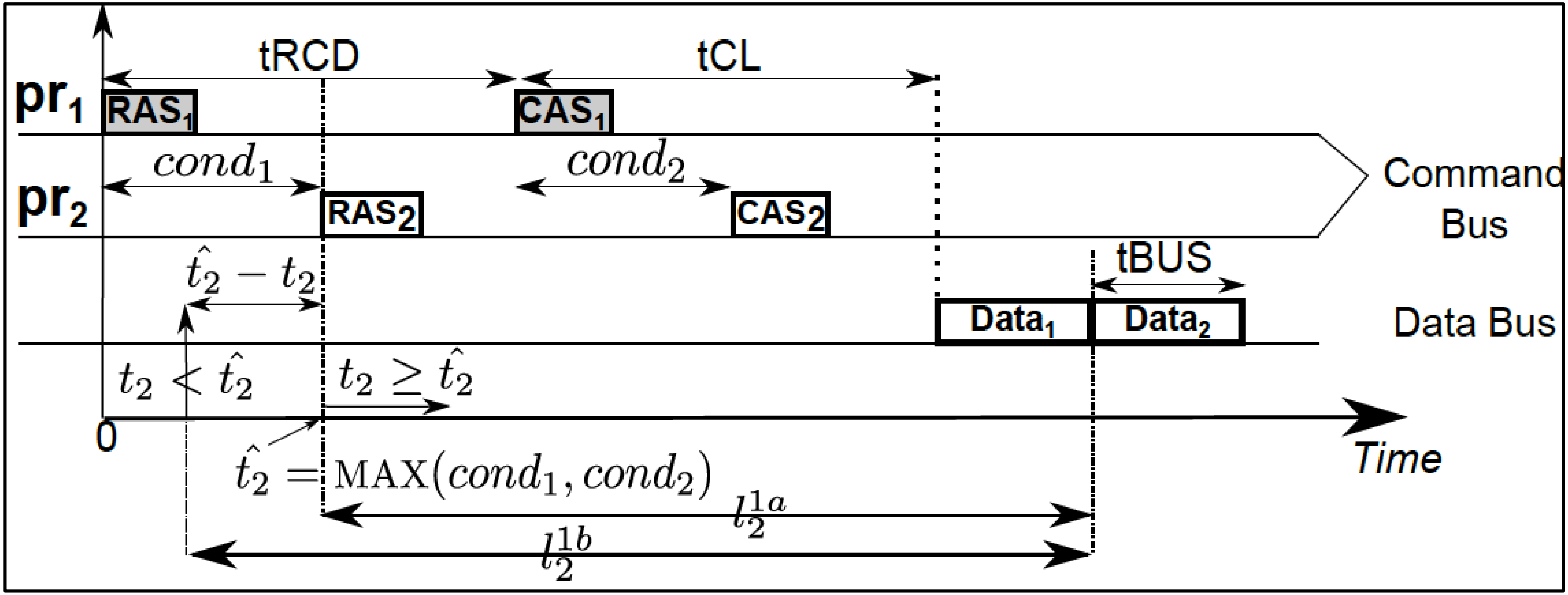


Best-case access latency l'_2 : tRCD + tWL

Worst-case access latency l''_2 : max(tRRD, tBUS + tRTW) + tRCD + tWL

Latency analysis

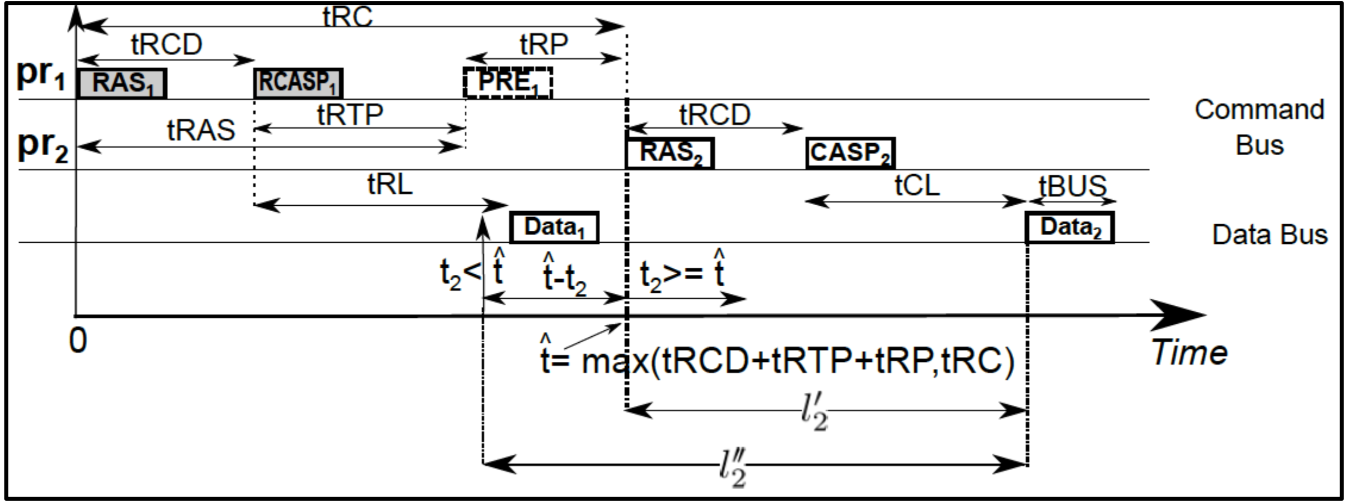
Combining **Case 1** and **Case 2** :



Latency of **pr$_2$** at any given arrival time ***t$_2$*** : $l_2 = MAX(\hat{t}_2 - t_{2,} 0) + l_2^{1a}$
when $\hat{t}_2 = MAX(cond_{1,} cond_2)$

Proof Strategy : Examples

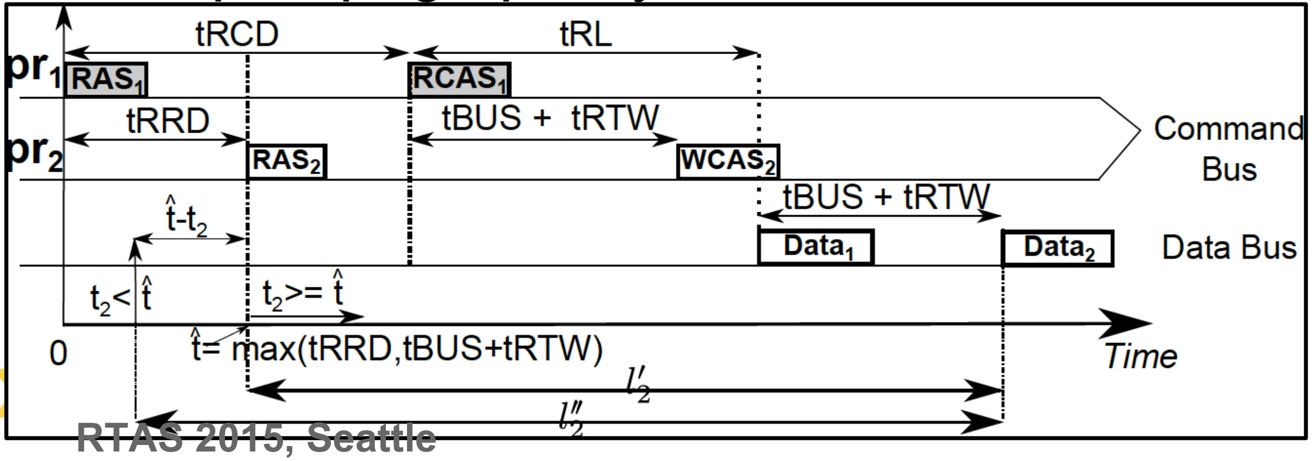
- Close-page policy, same bank/rank, read followed by write/read request



l_2': tRP + tRCD + tCL

l_2'': $\hat{t}_2$+ tRCD + tCL

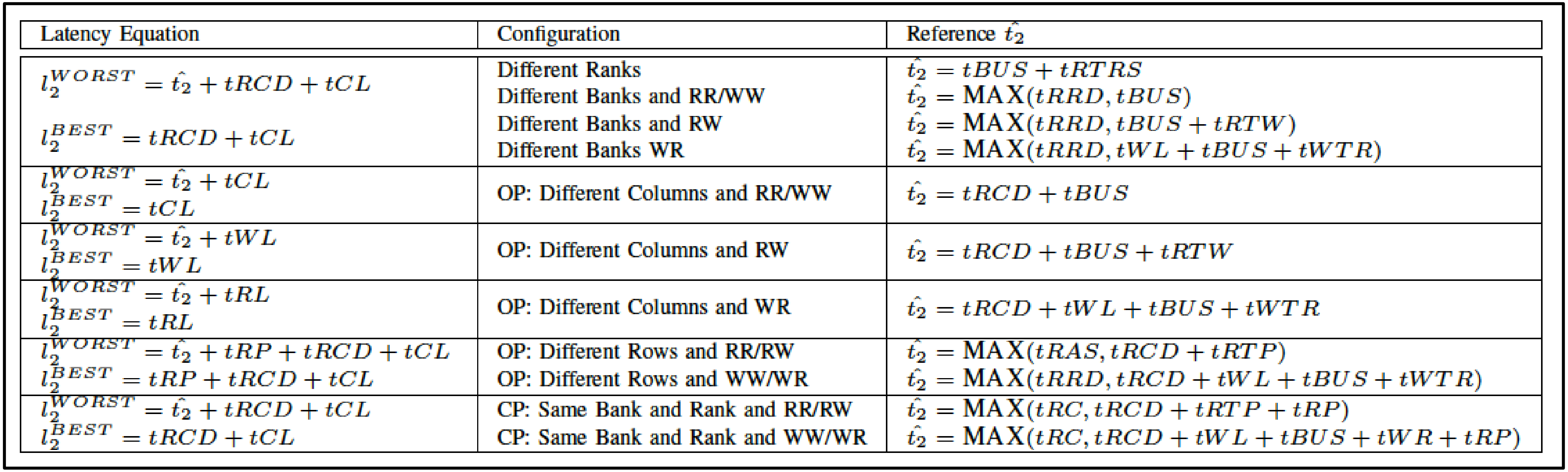
- Close/Open-page policy, different banks, read followed by write request



l_2' : tRCD + tCL

l_2'' : $\hat{t}_2$+ tRCD + tCL

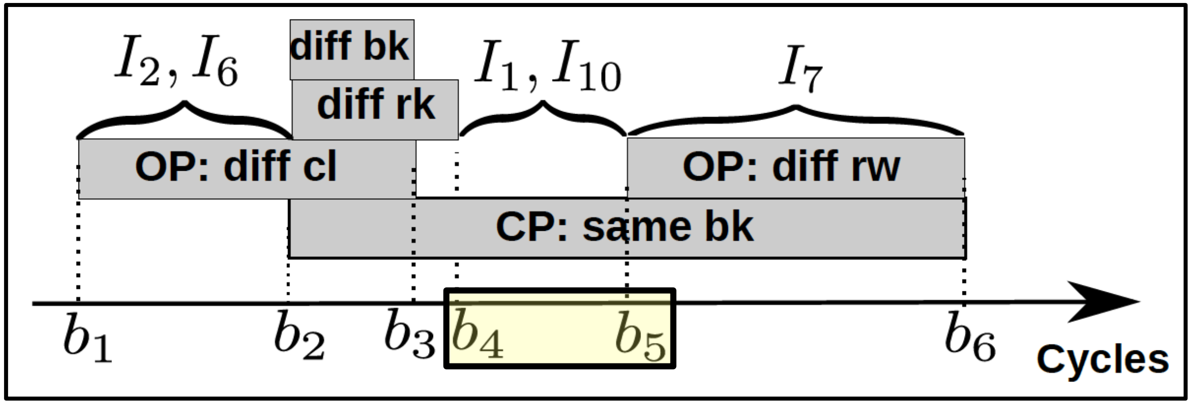
Latency analysis : Summary



Latency Equation	Configuration	Reference $\hat{t_2}$
$l_2^{WORST} = \hat{t_2} + tRCD + tCL$ $l_2^{BEST} = tRCD + tCL$	Different Ranks Different Banks and RR/WW Different Banks and RW Different Banks WR	$\hat{t_2} = tBUS + tRTRS$ $\hat{t_2} = \mathrm{MAX}(tRRD, tBUS)$ $\hat{t_2} = \mathrm{MAX}(tRRD, tBUS + tRTW)$ $\hat{t_2} = \mathrm{MAX}(tRRD, tWL + tBUS + tWTR)$
$l_2^{WORST} = \hat{t_2} + tCL$ $l_2^{BEST} = tCL$	OP: Different Columns and RR/WW	$\hat{t_2} = tRCD + tBUS$
$l_2^{WORST} = \hat{t_2} + tWL$ $l_2^{BEST} = tWL$	OP: Different Columns and RW	$\hat{t_2} = tRCD + tBUS + tRTW$
$l_2^{WORST} = \hat{t_2} + tRL$ $l_2^{BEST} = tRL$	OP: Different Columns and WR	$\hat{t_2} = tRCD + tWL + tBUS + tWTR$
$l_2^{WORST} = \hat{t_2} + tRP + tRCD + tCL$ $l_2^{BEST} = tRP + tRCD + tCL$	OP: Different Rows and RR/RW OP: Different Rows and WW/WR	$\hat{t_2} = \mathrm{MAX}(tRAS, tRCD + tRTP)$ $\hat{t_2} = \mathrm{MAX}(tRRD, tRCD + tWL + tBUS + tWTR)$
$l_2^{WORST} = \hat{t_2} + tRCD + tCL$ $l_2^{BEST} = tRCD + tCL$	CP: Same Bank and Rank and RR/RW CP: Same Bank and Rank and WW/WR	$\hat{t_2} = \mathrm{MAX}(tRC, tRCD + tRTP + tRP)$ $\hat{t_2} = \mathrm{MAX}(tRC, tRCD + tWL + tBUS + tWR + tRP)$

Reverse-engineering MC: Address mapping for Close-page policy

Observation : Highlighted access range is unique for close-page policy and moreover for row/column bits.

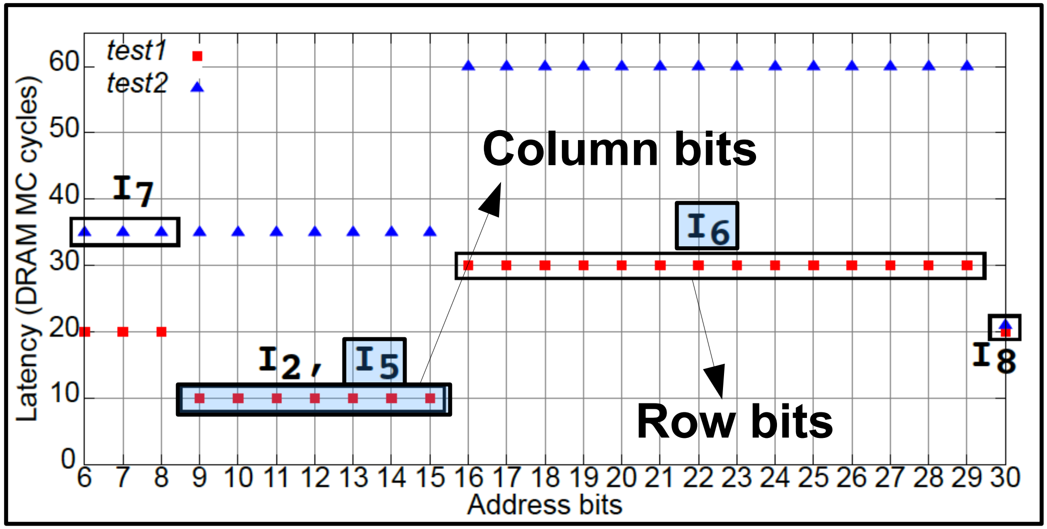


Methodology : **test$_1$** : ***Read [addr$_1$] NOP() Read [addr$_2$]***

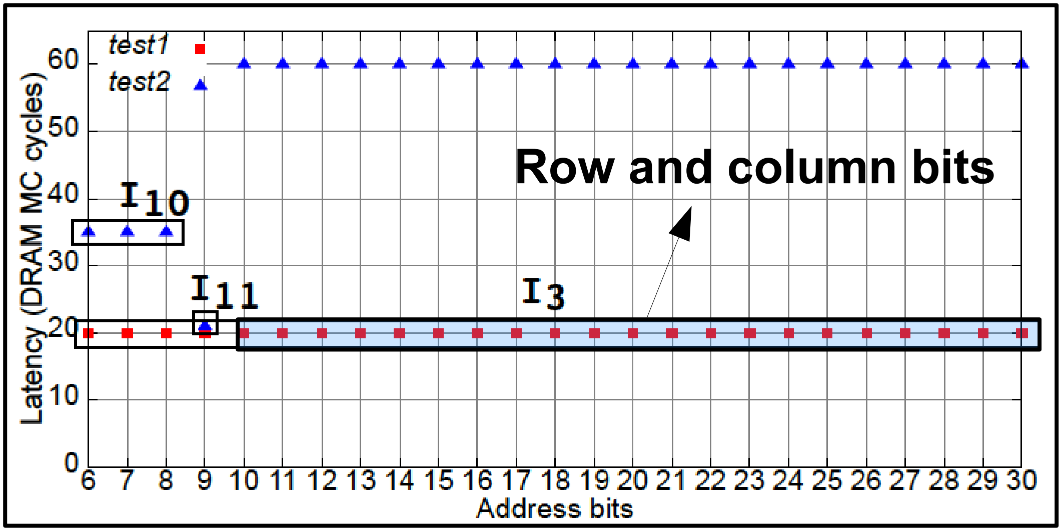
Execute **test$_1$** for ***PW*** times with ***addr$_2$ = flipBit(addr$_1$, i)***

Inference rules : $(I_9)\,\forall i \in [0, PW-1] : b_4 < l_2^i < b_5 \Rightarrow i\ is\ row \vee column\ bit$

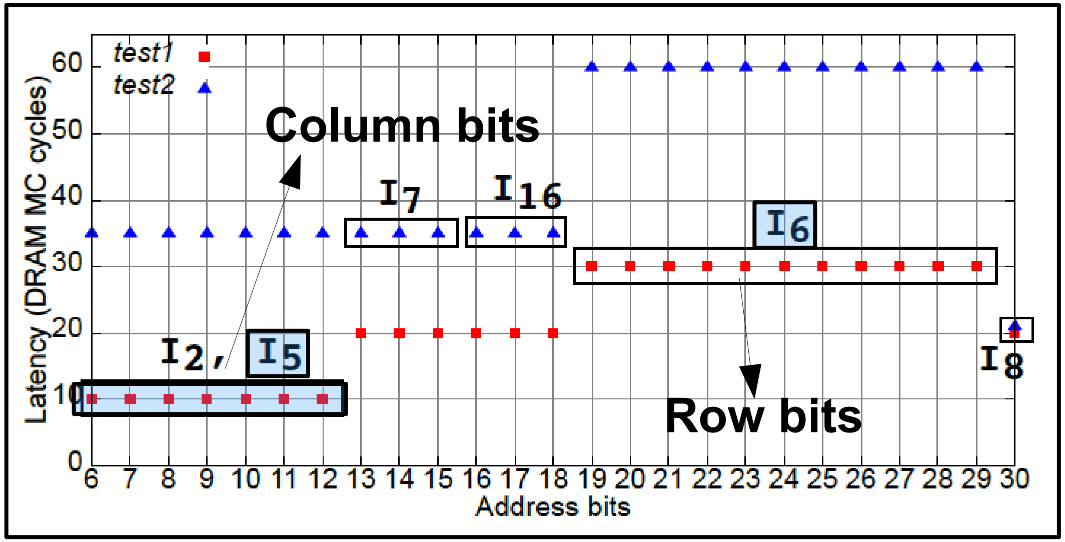
Results



test1
test2
Column bits
Row bits
I7
I6
I2, I5
I8
Latency (DRAM MC cycles)
0
10
20
30
40
50
60
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Address bits



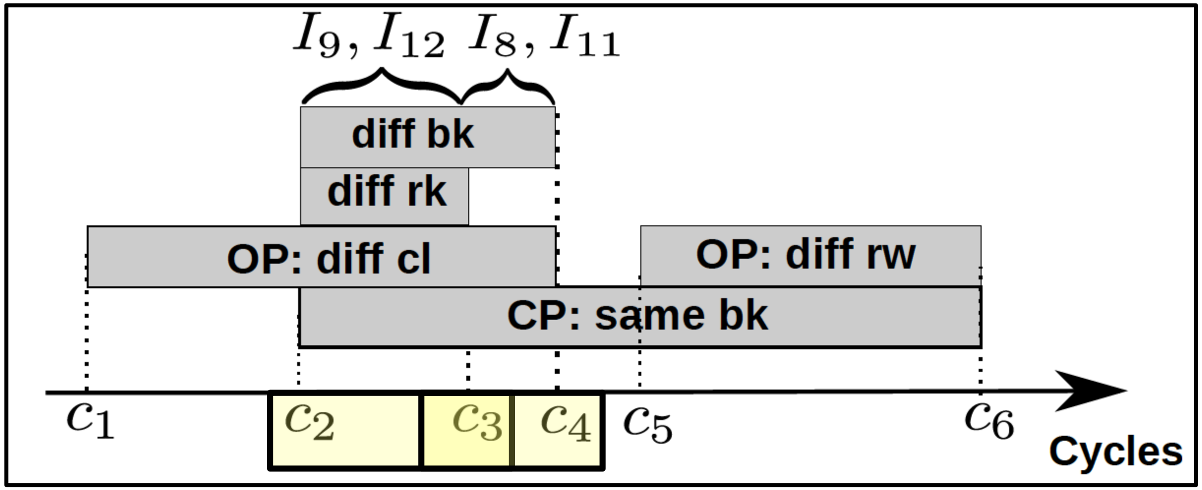
test1
test2
Row and column bits
I10
I11
I3
Latency (DRAM MC cycles)
0
10
20
30
40
50
60
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Address bits



test1
test2
Column bits
Row bits
I7
I16
I6
I2, I5
I8
Latency (DRAM MC cycles)
0
10
20
30
40
50
60
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Address bits

Reverse-engineering MC: Address mapping for Close-page policy

Observation : Shared data bus for banks in a rank. Switching delay to change bus direction from write to read and vice-versa. Switching overhead changing between ranks.



Methodology : **test**$_3$: ***Write [addr$_1$] NOP()   Read [addr$_2$]***

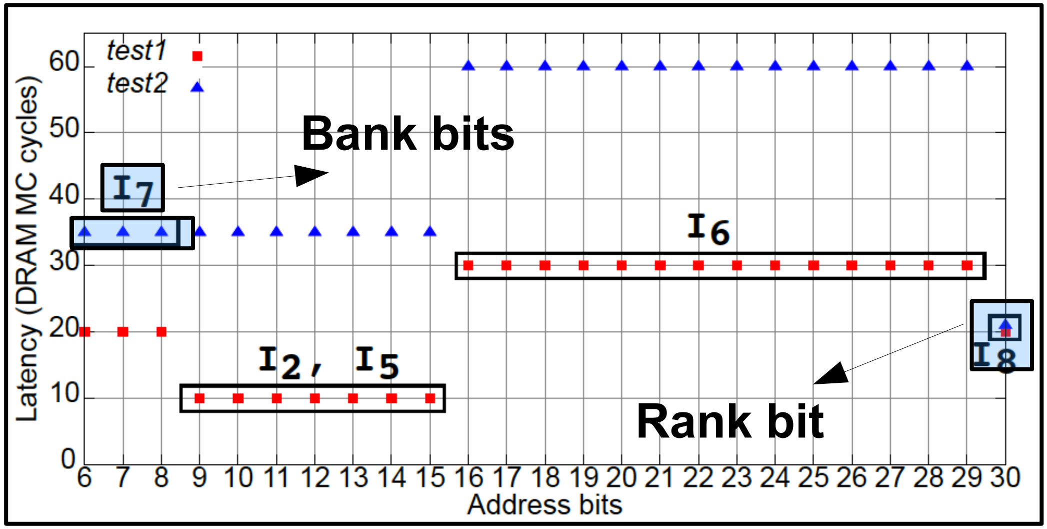
Execute **test**$_3$ for ***PW*** times with ***addr$_2$* = *flipBit(addr$_1$, i)***

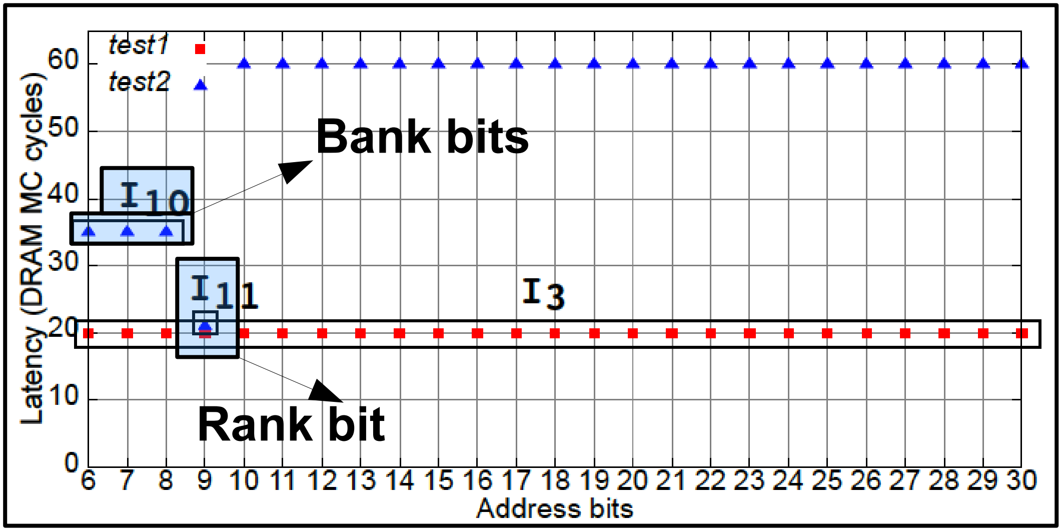
Inference rules :

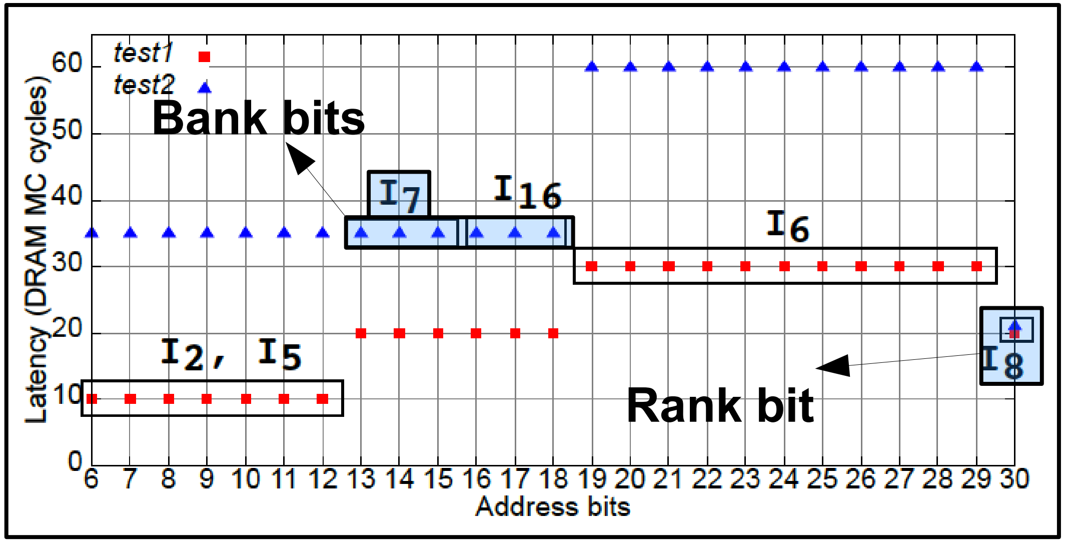
$$(I_{10})\,\forall\, i \in [0, PW-1] : (i\ is\ not\ column \vee row\ bit) \wedge (c_3 < l_2^i < c_4) \Rightarrow i\ is\ bank\ bit$$

$$(I_{11})\,\forall\, i \in [0, PW-1] : (i\ is\ not\ column\ , row \vee bank\ bit) \wedge (c_2 < l_2^i < c_3) \Rightarrow i\ is\ rank\ bit$$

Results


test1
test2
Latency (DRAM MC cycles)
0
10
20
30
40
50
60
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Address bits
Bank bits
I_7
I_6
I_2, I_5
I_8
Rank bit


test1
test2
Latency (DRAM MC cycles)
0
10
20
30
40
50
60
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Address bits
Bank bits
I_{10}
I_{11}
I_3
Rank bit


test1
test2
Latency (DRAM MC cycles)
0
10
20
30
40
50
60
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Address bits
Bank bits
I_7
I_{16}
I_6
I_2, I_5
I_8
Rank bit

Reverse-engineering MC: XOR Address mapping scheme

Observation : XOR address mapping scheme performs XOR operation on row bits with bank bits to convert requests targeting different rows, same banks to different banks. Hence more bits will appear to be bank bits when executing ***test*$_1$*.***

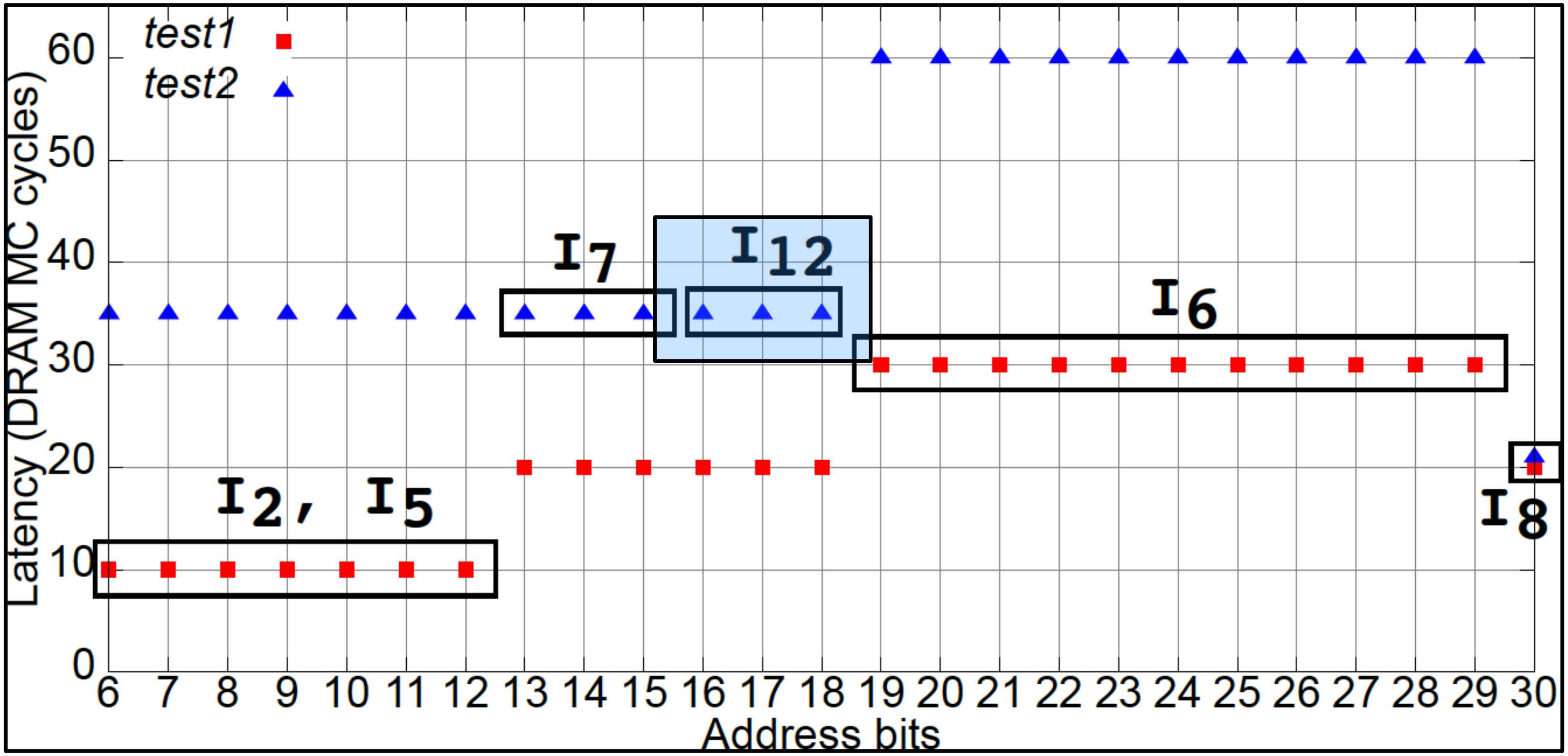
Methodology : **test$_4$** : ***Read [addr$_1$] NOP() Read [addr$_2$]***

Execute **test$_4$** for ***PW*** times with ***addr$_2$ = flipBit(addr$_1$, i)***

Inference rules : $(I_{12}) \forall i \in [0, PW-1]: inferred\ bank\ bits > actual\ bank\ bits$

$$\Rightarrow XOR\ address\ mapping\ scheme$$

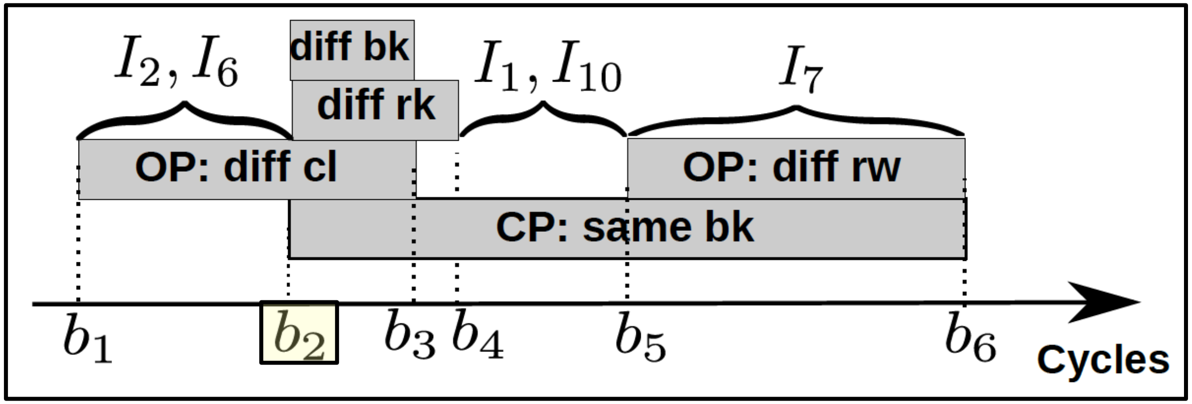
Results



test1 🟥
test2 ▲
Latency (DRAM MC cycles)
0
10
20
30
40
50
60
Address bits
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
I_2, I_5
I_7
I_{12}
I_6
I_8

Reverse-engineering MC: FR-FCFS threshold depth

Observation : Some implementations of FR-FCFS limit the number of requests targeting an open row in the row-buffer that can be reordered and prioritized to avoid starving other requests. Once threshold is satisfied, precharge command is executed to close the row in row-buffer.

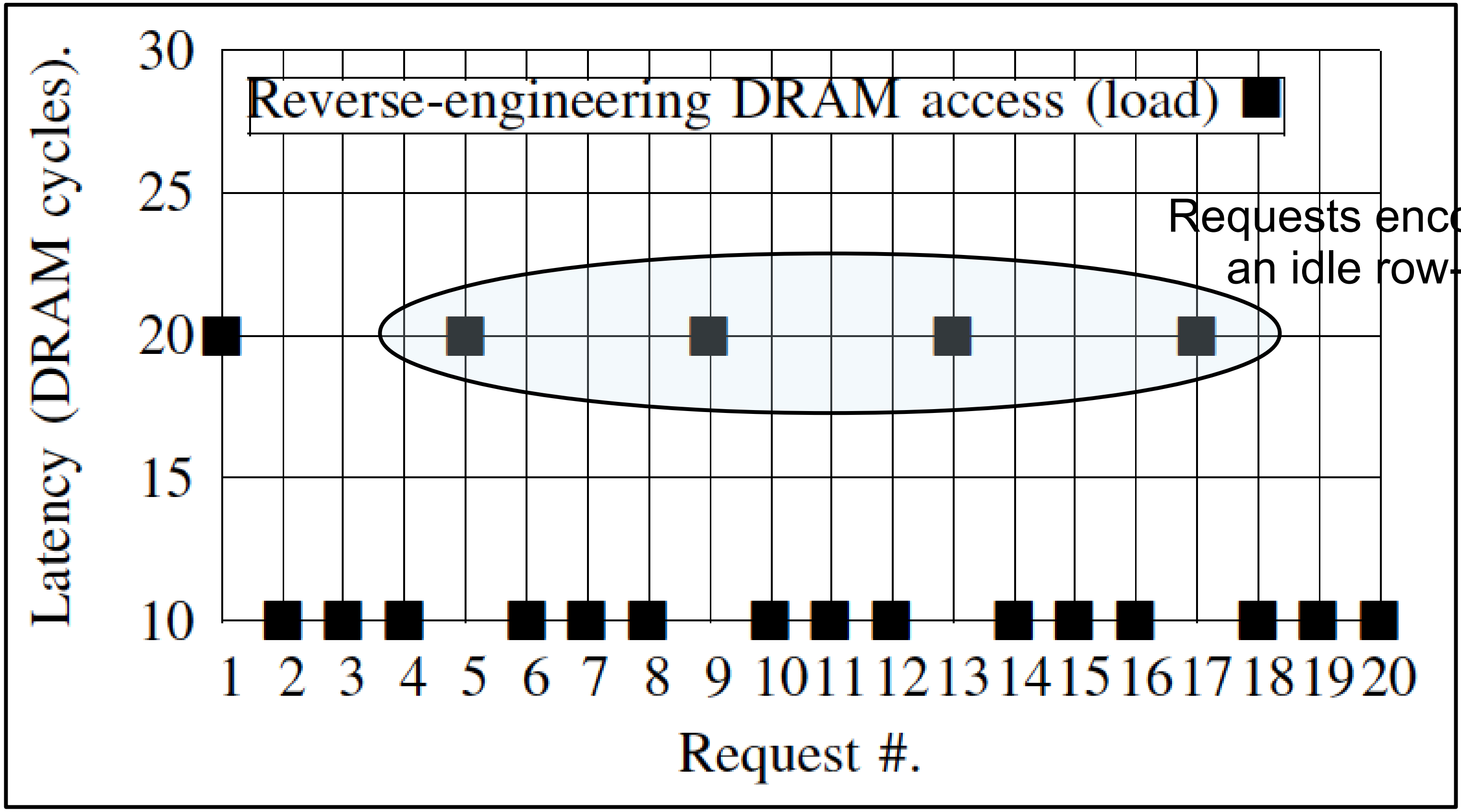


Methodology : Execute a sequence of requests that target same row, bank, and rank.

Read [addr$_1$]   NOP   Read[addr$_2$]   NOP   Read[addr$_n$]

Inference rules : $\exists l_2 : l_2 \geq b_2$

Results



Requests encountering an idle row-buffer

Hardware considerations for reverse-engineering MC

- ✓ Single-core/requestor architecture with memory buffers
  - Load-store buffers or store buffers for accumulating requests to MC
- ✓ Multi-core/multi-requestor architecture
  - Shared MC between multiple cores/requestors
  - Simultaneous requests from multiple cores/requestors accumulate requests to MC

Hardware considerations for reverse-engineering MC

- ✓ Out-of-Order Pipeline architecture
  - Re-order buffer aids in accumulating requests to MC
- ✓ Multi-core/multi-requestor architecture
  - Shared MC between multiple cores/requestors
  - Simultaneous requests from multiple cores/requestors accumulate requests to MC
- In-Order Pipeline architecture
  - ✗ Stalls on every memory access
  - ✓ Load-store buffer or store buffer