ETH 227-2210-00L Computer Architecture, Fall 2021

HW 3: Low-Latency Memory, Memory Controllers, Memory Interference and Quality of Service, Emerging Memory Technologies (SOLUTIONS)

Instructor: Prof. Onur Mutlu
TAs: Juan Gómez Luna, Mohammed Alser, Jisung Park, Lois Orosa Nogueira, Gagandeep Singh, Haiyu Mao, Behzad Salami, Nour Almadhoun Alserr, Mohammad Sadr, Hasan Hassan, Can Firtina Geraldo Francisco De Oliveira Junior, Abdullah Giray Yaglikci, Rahul Bera, Konstantinos Kanellopoulos, Nika Mansouri Ghiasi, Rakesh Nadig, João Dinis Ferreira, Haocong Luo, Roknoddin Azizibarzoki

Given: Saturday, Nov 13, 2021
Due: **Saturday, Nov 27, 2021**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture21/`. Please, check your inbox, you should have received an email with the password you should use to login. If you did not receive any email, contact comparch@lists.inf.ethz.ch. In the first page after login, you should click in "Computer Architecture Home", and then go to "any submitted paper" to see the list of papers.
- **Handin - Questions (2-6).** You should upload your answers to the Moodle Platform (`https://moodle-app2.let.ethz.ch/course/view.php?id=15536`) as a single PDF file.
- If you have any questions regarding this homework, please ask them the Moodle forum (`https://moodle-app2.let.ethz.ch/mod/moodleoverflow/view.php?id=662952`).
- Please note that the handin questions have a hard deadline. However, you can submit your paper reviews till the end of the semester.

1. Critical Paper Reviews [1,000 points]

You will do at least 4 readings for this homework, out of which 3 are tagged as **REQUIRED** papers. You may access them by *simply clicking on the QR codes below or scanning them.*

Required Reading 1 | Required Reading 2 | Required Reading 3

Write an approximately one-page critical review for the readings (i.e., papers from #1 to #3 **and** *at least* 1 of the remaining 23 papers, from #4 to #26). If you review a paper other than the 4 mandatory papers, you will receive 250 BONUS points on top of 1,000 points you may get from paper reviews (i.e., each additional submission is worth 250 BONUS points with a possibility to get up to 6500 points).

Please read the guideline slides for reviewing papers and watch Prof. Mutlu's guideline video on how to do a critical paper review. We also provide you with sample reviews which you can access using the QR code. A review with bullet point style is more appreciated. Try not to use very long sentences and paragraphs. Keep your writing and sentences simple. Make your points bullet by bullet, as much as possible. **We will give out extra credit that is worth 0.5% of your total course grade for each good review.**

Guideline Slides | Guideline Video | Sample Reviews

1. **(REQUIRED)** Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture", in Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), 2013. https://people.inf.ethz.ch/omutlu/pub/tldram_hpca13.pdf
2. **(REQUIRED)** Kim et al., "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput", in Proceedings of the 25th International Symposium on High-Performance Computer Architecture (HPCA), 2019. https://people.inf.ethz.ch/omutlu/pub/drange-dram-latency-based-true-random-number-generator_hpca19.pdf
3. **(REQUIRED)** Ipek et al., "Self Optimizing Memory Controllers: A Reinforcement Learning Approach", in Proceedings of the 35th International Symposium on Computer Architecture (ISCA), 2008. https://people.inf.ethz.ch/omutlu/pub/rlmc_isca08.pdf
4. Mahadev Satyanarayanan (Satya), "Edge Computing: A New Disruptive Force.", The 13th ACM International Systems and Storage Conference Keynote Talk (Vitrual), 2020. https://www.youtube.com/watch?v=7D2ZrMQWt7A
5. Luo et al., "CLR-DRAM: A Low-Cost DRAM Architecture Enabling Dynamic Capacity-Latency Trade-Off", in Proceedings of the 47th International Symposium on Computer Architecture (ISCA), 2020. https://people.inf.ethz.ch/omutlu/pub/CLR-DRAM_capacity-latency-reconfigurable-DRAM_isca20.pdf
6. Chang et al., "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization", in Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), 2016. https://people.inf.ethz.ch/omutlu/pub/understanding-latency-variation-in-DRAM-chips_sigmetrics16.pdf
7. Chang et al., "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM", in Proceedings of the 22nd International Symposium on High-Performance Computer Architecture (HPCA), 2016. https://people.inf.ethz.ch/omutlu/pub/lisa-dram_hpca16.pdf
8. Olgun et al., "QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips", in Proceedings of the 48th International Symposium on Computer Architecture (ISCA), 2021. https://people.inf.ethz.ch/omutlu/pub/QUAC-TRNG-DRAM_isca21.pdf
9. Hassan et al., "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality", in Proceedings of the 22nd International Symposium on High-Performance Computer Architecture (HPCA), 2016. https://people.inf.ethz.ch/omutlu/pub/chargecache_low-latency-dram_hpca16.pdf
10. Kim et al.,"A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM", in Proceedings of the 39th International Symposium on Computer Architecture (ISCA), 2012. https://people.inf.ethz.ch/omutlu/pub/salp-dram_isca12.pdf
11. Kim et al., "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers", in Proceedings of the 16th International Symposium on High Performance Computer Architecture (HPCA), 2010. https://people.inf.ethz.ch/omutlu/pub/atlas_hpca10.pdf
12. Kim et al., "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior", in Proceedings of the 43rd International Symposium on Microarchitecture (MICRO), 2010. https://people.inf.ethz.ch/omutlu/pub/tcm_micro10.pdf
13. Mutlu et al., "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems", in Proceedings of the 35th International Symposium on Computer Architecture (ISCA), 2008. https://people.inf.ethz.ch/omutlu/pub/parbs_isca08.pdf
14. Mutlu et al., "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors", in Proceedings of the 40th International Symposium on Microarchitecture (MICRO), 2007. https://people.inf.ethz.ch/omutlu/pub/stfm_micro07.pdf
15. Subramanian et al., "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling", IEEE Transactions on Parallel and Distributed Systems (TPDS), 2016. https://people.inf.ethz.ch/omutlu/pub/bliss-memory-scheduler_ieee-tpds16.pdf
16. Subramanian et al., "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems", in Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), 2013. https://people.inf.ethz.ch/omutlu/pub/mise-predictable_memory_performance-hpca13.pdf
17. Ebrahimi et al., "Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems", ACM Transactions on Computer Systems (TOCS), 2012. https://people.inf.ethz.ch/omutlu/pub/fairness-via-throttling_acm_tocs12.pdf

18. Cai et al., "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery", in Invited Book Chapter in Inside Solid State Drives, 2018. https://arxiv.org/pdf/1711.11427.pdf
19. Kim et al., "Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-Based Storage Systems", in Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2020. https://people.inf.ethz.ch/omutlu/pub/evanesco-secure-data-sanitization-for-flash-memory_asplos20.pdf
20. Park et al., "Reducing Solid-State Drive Read Latency by Optimizing Read-Retry", in Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2021. https://people.inf.ethz.ch/omutlu/pub/Reducing-SSD-Read-Latency-by-Optimizing-Read-Retry_asplos21.pdf
21. Lee et al., "Architecting phase change memory as a scalable dram alternative", in Proceedings of the 3th International Symposium on Computer Architecture (ISCA), 2009. https://people.inf.ethz.ch/omutlu/pub/pcm_isca09.pdf
22. Lee et al., "Phase Change Technology and the Future of Main Memory", IEEE Micro, Special Issue: Micro's Top Picks from Microarchitecture Conferences (MICRO TOP PICKS), 2010. https://people.inf.ethz.ch/omutlu/pub/pcm_ieee_micro10.pdf
23. Qureshi et al., "Scalable high performance main memory system using phase-change memory technology", in Proceedings of the 36th annual international symposium on Computer architecture (ISCA), 2009. https://dl.acm.org/doi/pdf/10.1145/1555754.1555760
24. Yoon et al.,"Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories", ACM Transactions on Architecture and Code Optimization (TACO), 2015 https://people.inf.ethz.ch/omutlu/pub/data-mapping-buffering-for-phase-change-memory_taco14.pdf
25. Kultursay et al., "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative", in Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2013. https://people.inf.ethz.ch/omutlu/pub/sttram_ispass13.pdf
26. Bera et al., "Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning", in Proceedings of the 54th International Symposium on Microarchitecture (MICRO), 2021. https://people.inf.ethz.ch/omutlu/pub/Pythia-customizable-hardware-prefetcher-using-reinforcement-learning_micro21.pdf

2. Tiered-difficulty [150 points]

Recall from your required reading on Tiered-Latency DRAM that there is a near and far segment, each containing some number of rows. Assume a very simplified memory model where there is just one bank and there are two rows in the near segment and four rows in the far segment. The time to activate and precharge a row is 25ns in the near segment and 50ns in the far segment. The time from start of activation to reading data is 10ns in the near segment and 15ns in the far segment. All other timings are negligible for this problem. Given the following memory request stream, determine the optimal assignment (minimize average latency of requests) of rows in the near and far segment (assume a fixed mapping where rows cannot migrate, a closed-row policy, and the far segment is inclusive).

```
time 0ns  : row 0 read
time 10ns : row 1 read
time 100ns: row 2 read
time 105ns: row 1 read
time 200ns: row 3 read
time 300ns: row 1 read
```

(a) What rows would you place in near segment? Hint: draw a timeline.

Rows 0 and 2.

Explanation. If you were to map 0 and 2 (this is the answer) to near segment:

```
 row 0:  activated at time = 0
row 0:  read at time = 10 (10ns latency)
row 1:  activated at time = 25
row 1:  read at time = 40 (30ns latency)
row 2:  activated at time = 100
row 2:  read at time = 110 (10ns latency)
row 1:  activated at time = 125
row 1:  read at time = 140 (35ns latency)
row 3:  activated at time = 200
row 3:  read at time = 215 (15ns latency)
row 1:  activated at time = 300
row 1:  read at time = 315 (15 ns latency)
```

total latency is **115ns**.

If you were to map 1 and 2 (an example incorrect answer) to near segment:

```
 row 0:  activated at time = 0
row 0:  read at time = 15 (15ns latency)
row 1:  activated at time = 50
row 1:  read at time = 60 (50ns latency)
row 2:  activated at time = 100
row 2:  read at time = 110 (10ns latency)
row 1:  activated at time = 125
row 1:  read at time = 135 (30ns latency)
row 3:  activated at time = 200
row 3:  read at time = 215 (15ns latency)
row 1:  activated at time = 300
row 1:  read at time = 310 (10 ns latency)
```

total latency is **130ns**.

(b) What rows would you place in far segment?

Rows 1 and 3 (also rows 0 and 2 since inclusive).

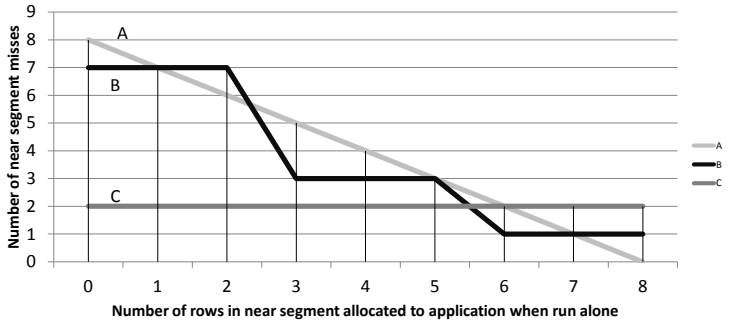
(c) In 15 words or less, describe the insight in your mapping?

See TL-DRAM's WMC policy – the first access in near simultaneous requests causes the second to wait activation + precharge time. minimizing this wait by caching first row in near segment is better than caching second row in near segment (this decreases only time to read from start of activation), even if second row is accessed more frequently (see example above)

(d) Assume now that the mapping is dynamic. What are the tradeoffs of an exclusive design vs. an inclusive design? Name one advantage and one disadvantage for each.

Exclusive requires swapping, but can use nearly full capacity of DRAM. Inclusive, the opposite.

(e) Assume now that there are eight (8) rows in the near segment. Below is a plot showing the number of misses to the near segment for three applications (A, B, and C) when run alone with the specified number of rows allocated to the application in the near segment. This is similar to the plots you saw in your Utility-Based Cache Partitioning reading except for TL-DRAM instead of a cache. Determine the optimal static partitioning of the near segment when all three of these applications are run together on the system. In other words, how many rows would you allocate for each application? Hint: this should sum to eight. Optimal for this problem is defined as minimizing total misses across all applications.



(1) How many near segment rows would you allocate to A?

5

(2) How many near segment rows would you allocate to B?

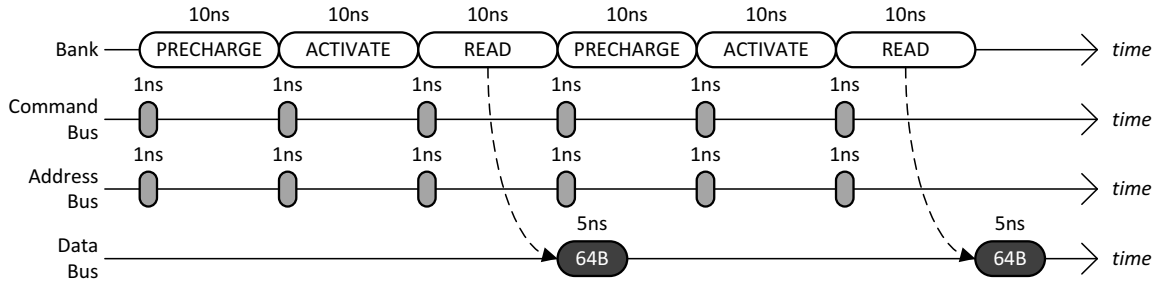
3

(3) How many near segment rows would you allocate to C?

0

3. Memory Interference and QoS [150 points]

Row-Buffer Conflicts. The following timing diagram shows the operation of a single DRAM channel and a single DRAM bank for two back-to-back reads that conflict in the row-buffer. Immediately after the bank has been busy for 10ns with a READ, data starts to be transferred over the data bus for 5ns.



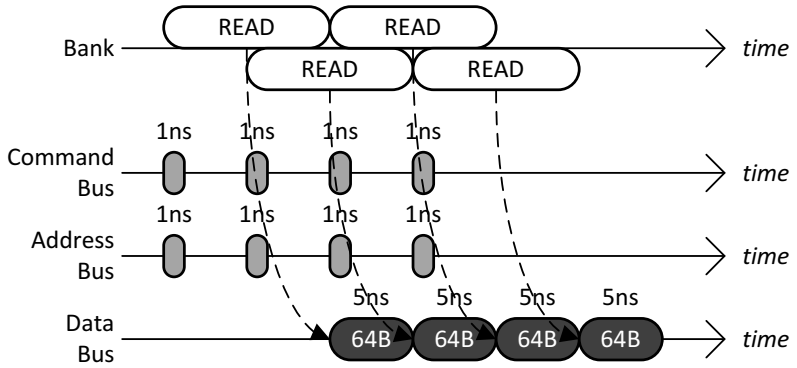
(a) Given a long sequence of back-to-back reads that always conflict in the row-buffer, what is the data throughput of the main memory system? Please state your answer in **gigabytes/second**.

64B/30ns = 32B/15ns = 32GB/15s = 2.13 GB/s

(b) To increase the data throughput, the main memory designer is considering adding more DRAM banks to the single DRAM channel. Given a long sequence of back-to-back reads to all banks that always conflict in the row-buffers, what is the minimum number of banks that is required to achieve the maximum data throughput of the main memory system?

30ns/5ns = 6

Row-Buffer Hits. The following timing diagram shows the operation of the single DRAM channel and the single DRAM bank for four back-to-back reads that hit in the row-buffer. It is important to note that row-buffer hits to the same DRAM bank are pipelined: while each READ keeps the DRAM bank busy for 10ns, up to at most **half** of this latency (5ns) can be overlapped with another read that hits in the row-buffer.



(c) Given a long sequence of back-to-back reads that always hits in the row-buffer, what is the data throughput of the main memory system? Please state your answer in **gigabytes/second**.

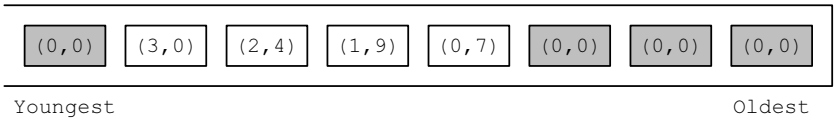
64B/5ns = 64GB/5s = 12.8GB/s

(d) When the maximum data throughput is achieved for a main memory system that has a single DRAM channel and a single DRAM bank, what is the bottleneck that prevents the data throughput from becoming even larger? **Circle** all that apply.

BANK COMMAND BUS ADDRESS BUS DATA BUS

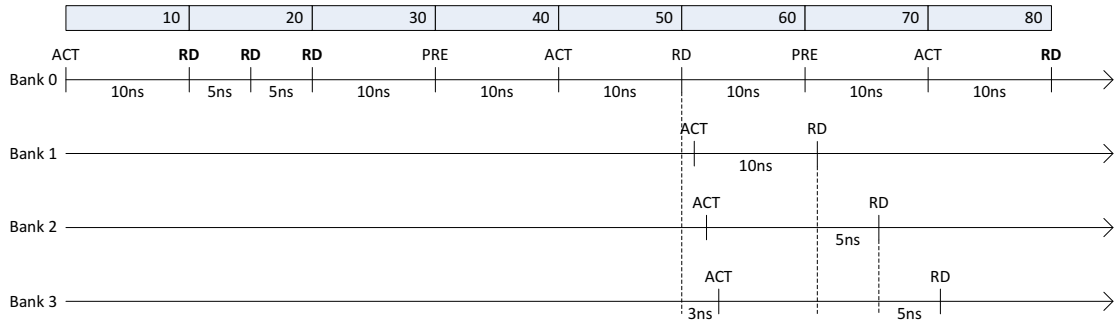
Memory Scheduling Policies. The diagram below shows the memory controller's *request queue* at time 0. The shaded rectangles are read requests generated by thread *T0*, whereas the unshaded rectangles are read requests generated by thread *T1*. Within each rectangle, there is a pair of numbers that denotes the request's (*BankAddress, RowAddress*). Assume that the memory system has a **single** DRAM channel and four DRAM banks. Further assume the following.

- All the row-buffers are **closed** at time 0.
- Both threads start to stall at time 0 because of memory.
- A thread continues to stall until it receives the data for all of its requests.
- Neither thread generates more requests.



We provide two sets of answers. The correct way to solve the problem is to model contention in the banks as well as in all of the buses (address/command/data). The answer that is given in the answer boxes is for the case you modeled contention in only the banks.

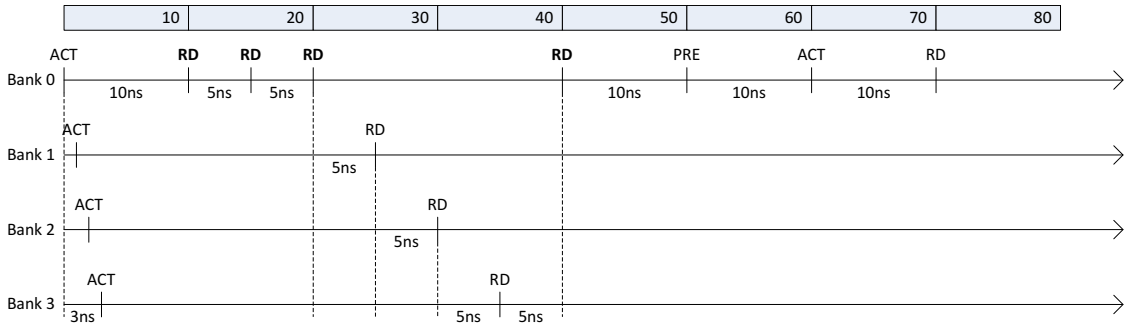
(f) For the $FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.



T0: (10 + 5 + 5 + 10 + 10 + 10) + 10 + 10 + 10 + 10 + 5 = 95ns

T1: (10 + 5 + 5 + 10 + 10 + 10) + 1 + 10 + 5 + 5 + 10 + 5 = 86ns

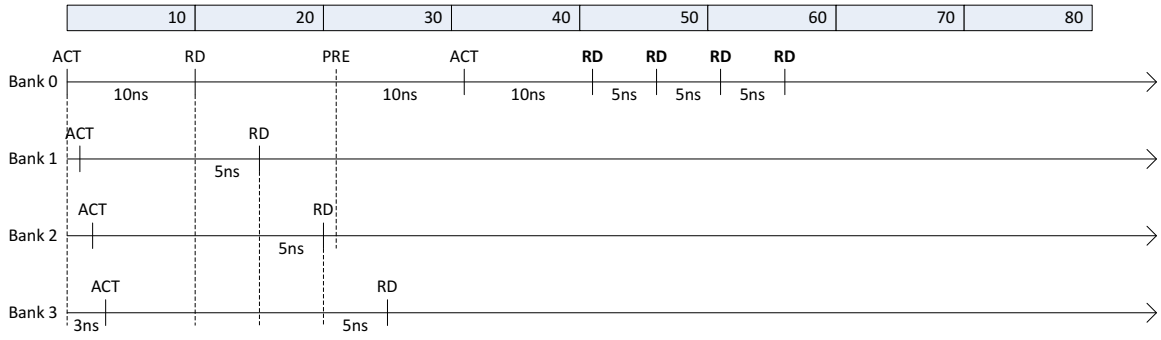
(g) For the $FR-FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.



T0: (10 + 5 + 5 + 5 + 5 + 5 + 5) + 10 + 5 = 55ns

T1: (10 + 5 + 5 + 5 + 5 + 5 + 5) + 10 + 10 + 10 + 10 + 5 = 85ns

(h) For the $PAR-BS$ scheduling policy, calculate the memory stall time of *T0* and *T1*. Assume that all eight requests are included in the same batch.



T0: (10 + 5 + 5) + 1 + 10 + 10 + 5 + 5 + 5 + 10 + 5 = 71ns

T1: (10 + 5 + 5) + 5 + 10 + 5 = 40ns

(f) For the $FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.

T0:

Bank 0 is the critical path for both threads.

T0 = Closed + Pipelined-Hit + Pipelined-Hit + Conflict + Conflict + Data
= (ACT+RD)+(RD/2)+(RD/2)+(PRE+ACT+RD)+(PRE+ACT+RD)+DATA
= 20ns + 5ns + 5ns + 30ns + 30ns + 5ns
= 95ns

T1:

T1 = Closed + Pipelined-Hit + Pipelined-Hit + Conflict + Data
= (ACT+RD)+(RD/2)+(RD/2)+(PRE+ACT+RD)+DATA
= 20ns + 5ns + 5ns + 30ns + 5ns
= 65ns

(g) For the $FR-FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.

T0:

Bank 0 is the critical path for both threads. First, we serve all four shaded requests since they are row-buffer hits. Lastly, we serve the unshaded request.

T0 = Closed + Pipelined-Hit + Pipelined-Hit + Pipelined-Hit + Data
= (ACT+RD)+(RD/2)+(RD/2)+(RD/2)+DATA
= 20ns + 5ns + 5ns + 5ns + 5ns
= 40ns

T1:

T1 = Closed + Pipelined-Hit + Pipelined-Hit + Pipelined-Hit + Conflict + Data
= (ACT+RD)+(RD/2)+(RD/2)+(RD/2)+(PRE+ACT+RD)+DATA
= 20ns + 5ns + 5ns + 5ns + 30ns + 5ns
= 70ns

(h) For the $PAR-BS$ scheduling policy, calculate the memory stall time of *T0* and *T1*. Assume that all eight requests are included in the same batch.

T0:

First, we serve all four unshaded requests in parallel across the four banks. Then, we serve all four shaded requests in serial.

T0 = Closed + Conflict + Pipelined-Hit + Pipelined-Hit + Pipelined-Hit + Data
= (ACT+RD)+(PRE+ACT+RD)+(RD/2)+(RD/2)+(RD/2)+DATA
= 20ns + 30ns + 5ns + 5ns + 5ns + 5ns
= 70ns

T1:

T1 = Closed + Data
= (ACT+RD)+DATA
= 20ns + 5ns
= 25ns

4. Memory Scheduling [150 points]

In lectures, we introduced a variety of ways to tackle memory interference. In this problem, we will look at the Blacklisting Memory Scheduler (BLISS) to reduce unfairness. There are two key aspects of BLISS that you need to know.

- When the memory controller services η consecutive requests from a particular application, this application is blacklisted. We name this non-negative integer η the **Blacklisting Threshold**.
- The blacklist is cleared periodically every **10000** cycles starting at $t = 0$.

To reduce unfairness, memory requests in BLISS are prioritized in the following order:

- Non-blacklisted applications' requests
- Row buffer hit requests
- Older requests

The memory system for this problem consists of 2 channels with 2 banks each. Tables 1 and 2 show the memory request stream in the same bank for both applications at different times ($t = 0$ and $t = 10$). For both tables, a request on the left-hand side is older than a request on the right-hand side in the same table. The applications do not generate more requests than those shown in Tables 1 and 2. The memory requests are labeled with numbers that represent the row position of the data within the accessed bank. Assume the following for all questions:

- A row buffer *hit* takes **100 cycles**.
- A row buffer *miss* (i.e., opening a row in a bank with a closed row buffer) takes **200 cycles**.
- A row buffer *conflict* (i.e., closing the currently open row and opening another one) takes **250 cycles**.
- All row buffers are closed at time $t = 0$

Application A (Channel 0, Bank 0)								
Application B (Channel 0, Bank 0)	Row 2	Row 2	Row 2	Row 2	Row 2	Row 3	Row 3	Row 4

Table 1. Memory requests of the two applications at $t = 0$

Application A (Channel 0, Bank 0)	Row 3	Row 7	Row 2	Row 0	Row 5	Row 3	Row 8	Row 9
Application B (Channel 0, Bank 0)	Row 2	Row 2	Row 2	Row 2	Row 2	Row 3	Row 3	Row 4

Table 2. Memory requests of the two applications at $t = 10$. Note that none of the Application B's existing requests are serviced yet.

(a) Compute the slowdown of each application using the FR-FCFS scheduling policy after both threads ran to completion. We define:

$slowdown = \frac{memory\ latency\ of\ the\ application\ when\ run\ together\ with\ other\ applications}{memory\ latency\ of\ the\ application\ when\ run\ alone}$

Show your work.

$slowdown_A = \sim 1.53$
$slowdown_B = 1.25$

Explanation:
For both applications, the first request will incur row buffer miss penalty, and the rest of the requests will either be hits or conflicts.
Application A (alone) $= 200 + 100 + 250 * 6 = 1800$ *cycles*
Application B (alone) $= 200 + 100 * 4 + 250 + 100 + 250 = 1200$ *cycles*

Applications A (with B, FR-FCFS) $= 200+100*4+100+250+100+100*2+250+250*5 = 2750$ *cycles*
Applications B (with A, FR-FCFS) $= 200+100*4+100+250+100+100*2+250 = 1500$ *cycles*

From the two tables above we know that all requests of application B were issued before any of the application A's requests were issued. Thus, all requests of B are prioritized unless there is a row hit for A's requests.
$slowdown_A = \frac{2750}{1800} = \sim 1.53$

$slowdown_B = \frac{1500}{1200} = 1.25$

(b) If we use the BLISS scheduler, for what value(s) of η (the Blacklisting Threshold) will the slowdowns of **both** applications be equal to those obtained with FR-FCFS?

For $\eta \geq 6$ or $\eta = 0$.

Explanation:
We want both A and B to complete without blacklisting or to complete both blacklisted, thus $\eta \geq 6$ and $\eta = 0$, respectively.

(c) For what value(s) of η (the Blacklisting Threshold) will the slowdown of A be < 1.5?

Impossible. Slowdown for A will always be ≥ 1.5

Explanation: For the give memory requests, it is not possible to find η that blacklists B but not A. Thus, the smallest slowdown for A is the case explained in the solution of part (b).

(d) For what value(s) of η (the Blacklisting Threshold) will B experience the maximum slowdown it can possibly experience with the Blacklisting Scheduler?

For $\eta = 5$.

Explanation: We already know that the slowdowns will be equal to the slowdown with FR-FCFS when $\eta \geq 6$ or $\eta = 0$. If we execute the memory requests for the rest of possible η values, we find that $\eta = 5$ causes application B to complete after 2150 cycles, which is the largest.

(e) What is a simple mechanism (that we discussed in lectures) that we can use instead of BLISS to make the slowdowns of both A and B equal to 1.00?

Memory Channel Partitioning (MCP)

Explanation: With MCP, each application will operate on an independent channel, without any interference with the other application.

5. DRAM Scheduling and Latency [150 points]

You would like to understand the configuration of the DRAM subsystem of a computer using reverse engineering techniques. Your current knowledge of the particular DRAM subsystem is limited to the following information:

- The physical memory address is 16 bits.
- The DRAM subsystem consists of a single channel, 2 banks, and 64 rows per bank.
- The DRAM is byte-addressable.
- The most-significant bit of the physical memory address determines the bank. The following 6 bits of the physical address determine the row.
- The DRAM command bus operates at 1 GHz frequency.
- The memory controller issues commands to the DRAM in such a way that *no command* for servicing a *later* request is issued before issuing a READ command for the current request, which is the oldest request in the request buffer. For example, if there are requests A and B in the request buffer, where A is the older request and the two requests are to different banks, the memory controller does *not* issue an ACTIVATE command to the bank that B is going to access *before* issuing a READ command to the bank that A is accessing.
- The memory controller services requests in order with respect to each bank. In other words, for a given bank, the memory controller first services the oldest request in the *request buffer* that targets the same bank. If all banks are ready to service a request, the memory controller first services the oldest request in the request buffer.

You realize that you can observe the memory requests that are waiting to be serviced in the request buffer. At a particular point in time, you take the snapshot of the request buffer and you observe the following requests in the request buffer (in descending order of request age, where the oldest request is on the top):

time ↓

```
Read 0xD780
Read 0x280C
Read 0xE4D0
Read 0x2838
```

At the same time you take the snapshot of the request buffer, you start probing the DRAM command bus. You observe the DRAM command type and the cycle (relative to the first command) at which the command is seen on the DRAM command bus. The following are the DRAM commands you observe on the DRAM bus while the requests above are serviced.

```
Cycle 0  --- READ
Cycle 1  --- PRECHARGE
Cycle 8  --- PRECHARGE
Cycle 13 --- ACTIVATE
Cycle 18 --- READ
Cycle 20 --- ACTIVATE
Cycle 22 --- READ
Cycle 25 --- READ
```

Answer the following questions using the information provided above.

(a) What are the following DRAM timing parameters used by the memory controller, in terms of nanoseconds? If there is not enough information to infer the value of a timing parameter, write *unknown*.

i) ACTIVATE-to-READ latency:

5 ns.

Explanation. After issuing the ACTIVATE command at cycle 13, the memory controller waits until cycle 18, which indicates that the ACTIVATE-to-READ latency is 5 cycles. The command bus operates at 1 GHz, so it has 1 ns clock period. Thus, the ACTIVATE-to-READ is $5 * 1 = 5$ ns.

ii) ACTIVATE-to-PRECHARGE latency:

Unknown.

Explanation. In the command sequence above, there is not a PRECHARGE command that follows an ACTIVATE command with a known issue cycle. Thus, we cannot determine the ACTIVATE-to-PRECHARGE latency.

iii) PRECHARGE-to-ACTIVATE latency:

12 ns.

Explanation. The PRECHARGE-to-ACTIVATE latency can be easily seen in the first two commands at cycles 1 and 13. The PRECHARGE-to-ACTIVATE latency is 12 cycles = 12 ns.

iv) READ-to-PRECHARGE latency:

8 ns.

Explanation. The READ command at cycle 0 is followed by a PRECHARGE command to the same bank at cycle 8. There are idle cycles before cycle 8, which indicates that the memory controller delayed the PRECHARGE command until cycle 8 because the timing constaints but not because the command bus was busy. Thus, the READ-to-PRECHARGE is 8 cycles, which is $8 * 1 = 8$ ns for the 1 GHz DRAM command bus.

v) READ-to-READ latency:

4 ns.

Explanation. Bank 0 receives back-to-back reads at cycles 18 and 22. The READ-to-READ latency is 4 cycles, which is $4 * 1 = 4$ ns for the 1 GHz DRAM command bus.

(b) What is the status of the banks *prior* to the execution of any of the above requests? In other words, which rows from which banks were open immediately prior to issuing the DRAM commands listed above? Fill in the table below indicating whether a bank has an open row, and if there is an open row, specify its address. If there is not enough information to infer the open row address, write *unknown*.

	Open or Closed?	**Open Row Address**
Bank 0	Open	Unknown
Bank 1	Open	43

Explanation. By decoding the accessed addresses we can find which bank and row each access targets. Looking at the commands issued for those requests, we can determine which requests needed PRECHARGE (row buffer conflict, the initially open row is unknown in this case), ACTIVATE (the bank is initially closed), or directly READ (the bank is initially open and the open row is the same as the one that the request targets).

time ↓

`0xD780` → `Bank: 1, Row: 43` (Row hit, so Bank 1 must have row 43 open.)
`0x280C` → `Bank: 0, Row: 20` (PRECHARGE first. Any row other than 20 might have been open.)
`0xE4D0` → `Bank: 1, Row: 50`
`0x2838` → `Bank: 0, Row: 20`

(c) To improve performance, you decide to implement the idea of Tiered-Latency DRAM (TL-DRAM) in the DRAM chip. Assume that a bank consists of a single subarray. With TL-DRAM, an entire bank is divided into a near segment and far segment. When accessing a row in the near segment, the ACTIVATE-to-READ latency *reduces* by 1 cycle and the ACTIVATE-to-PRECHARGE latency reduces by 3 cycles. When precharging a row in the near segment, the PRECHARGE-to-ACTIVATE latency reduces by 3 cycles. When accessing a row in the far segment, the ACTIVATE-to-READ latency *increases* by 1 cycle and the ACTIVATE-to-PRECHARGE latency increases by 2 cycles. When precharging a row in the far segment, the PRECHARGE-to-ACTIVATE latency increases by 2 cycles. The following table summarizes the changes in the affected latency parameters.

Timing Parameter	**Near Segment Latency**	**Far Segment Latency**
ACTIVATE-to-READ	−1	+1
ACTIVATE-to-PRECHARGE	−3	+2
PRECHARGE-to-ACTIVATE	−3	+2

Assume that the rows in the near segment have smaller row ids compared to the rows in the far segment. In other words, physical memory row addresses 0 through $N-1$ are the near-segment rows, and physical memory row addresses N through 63 are the far-segment rows.
If the above DRAM commands are issued 2 cycles faster with TL-DRAM compared to the baseline (the last command is issued in cycle 23), how many rows are in the near segment, i.e., what is N? Show your work.

The rows in the range of [0-43] should definitely be in the near segment. Row 50 should definitely be in the far segment. Thus, N is a number between [44-50].

Explanation. There should be at least 44 rows in the near segment (rows 0 to 43) since rows until row id 43 need to be accessed with low latency to get 2 cycle reduction. The unknown open row in bank 0 should be in the near segment to get the 2 cycle improvement. Row 50 is in the far segment because if it was in the near segment, the command would have been finished in cycle 21, i.e., 4 cycles sooner instead of 2 cycles sooner. Thus, the number of rows in the near segment N is a number between 44 and 50.
Here is the new command trace:

```
Cycle 0 -- READ - Bank 1
Cycle 1 -- PRECHARGE - Bank 0, an unknown row in the near segment
Cycle 8 -- PRECHARGE - Bank 1, row 43, which is in the near segment
Cycle 10 -- ACT - Bank 0, row 20, which is in the near segment
Cycle 14 -- READ - Bank 0
Cycle 17 -- ACTIVATE - Bank 1, Row 50, which is in the far segment
Cycle 18 -- READ - Bank 0
Cycle 23 -- READ - Bank 1, Row 0
```

6. Emerging Memory Technologies [150 points]

Researchers at Lindtel developed a new memory technology, L-RAM, which is non-volatile. The access latency of L-RAM is close to that of DRAM while it provides higher density compared to the latest DRAM technologies. L-RAM has one shortcoming, however: it has limited endurance, i.e., a memory cell stops functioning after 10^6 writes are performed to the cell (known as cell wear-out).

(a) Lindtel markets a new computer system with L-RAM to have a lifetime of 2 years and the following specifications:

- 4GBytes of L-RAM as main memory with a *perfect* wear-leveling mechanism, i.e., writes are equally distributed over all the cells of L-RAM.
- The processor is in-order and there is no memory-level parallelism.
- It takes 4ns to send a memory request from the processor to the memory controller and it takes 20ns to send the request from the memory controller to L-RAM. The write latency of L-RAM is 40ns.
- L-RAM is word-addressable. Thus, each write request writes 8 bytes to memory.

A student at ETH tests the lifetime of the system and finds that this new computer system *cannot* guarantee a lifetime of 2 years. She writes a program to wear out the entire L-RAM device as quickly as possible. How fast is she able to wear out the device? Show all work.

$t_{wear\_out} = \frac{2^{32}}{2^3} \times 10^6 \times (40 + 4 + 20)$
$t_{wear\_out} = 2^{35} \times 10^6$ ns
$t_{wear\_out} \approx 397.68$ days

Explanation:

- Each memory cell should receive 10^6 writes.
- Since L-RAM is word addressable, the required number of writes is equal to $\frac{2^{32}}{2^3} \times 10^6$.
- The processor is in-order and there is no memory-level parallelism, so the total latency of each memory access is equal to $40 + 4 + 20$.

(b) L-RAM works in the multi-level cell (MLC) mode in which each memory cell stores 2 bits. The student decides to improve the lifetime of L-RAM cells by using the single-level cell (SLC) mode. When L-RAM is used in SLC mode, the lifetime of each cell improves by a factor of 10 and the write latency decreases by 75%. What is the lifetime of the system using the SLC mode, if we repeat the experiment in part (a), with all else remaining the same in the system? Show your work.

$t_{wear\_out} = \frac{2^{31}}{2^3} \times 10^7 \times (10 + 4 + 20) \times 10^{-9}$
$t_{wear\_out} = 91268055.04 \approx 1056.34$ days

Explanation:

- Each memory cell should receive $10 \times 10^6 = 10^7$ writes.
- The memory capacity is reduced by 50% since we are using SLC: $Capacity = 2^{32}/2 = 2^{31}$
- The required number of writes is equal to $\frac{2^{31}}{2^3} \times 10^7$.
- The SLC write latency is $0.25 \times t_{write\_MLC}$: $t_{write\_SLC} = 0.25 \times 40 = 10$

(c) Provide a mechanism that would increase the guaranteed lifetime of the computer system without changing the physical circuitry of L-RAM. From the baseline computer system in part (a), describe the changes required to guarantee a computer system lifetime of 2 years, with your mechanism. Be concrete and precise.

Artificially increase the time to either (1) send a memory request from the memory controller to L-RAM or (2) send a request from the processor to the memory controller by 54 ns.
$730 * 3600 * 24 < \frac{2^{32}}{2^3} \times 10^6 \times (40 + 4 + 20 + x)$
$x > 53.4$ns