Ramulator 2.0 Summary

How to measure off-chip power?

❑ **Analysis based on the paper below**

BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows
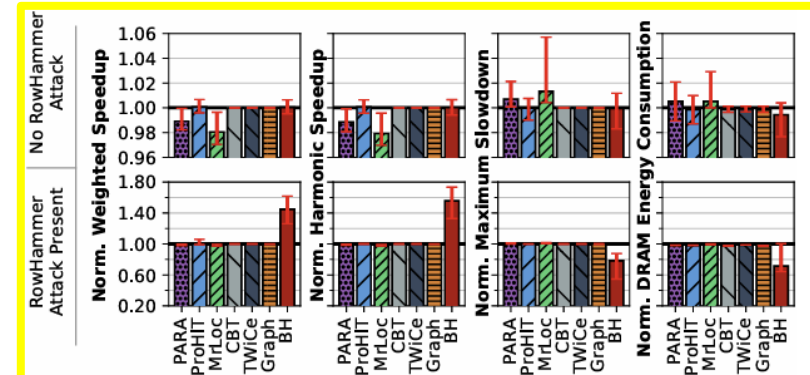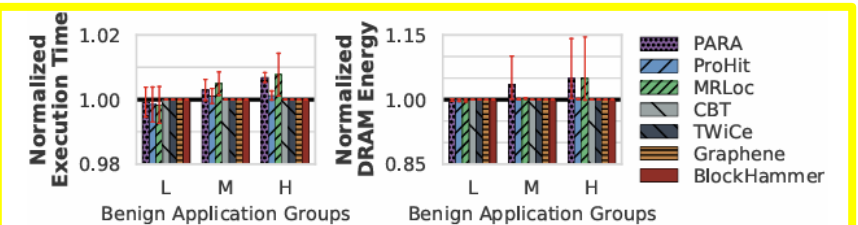
- **Section 6. Hardware Complexity Analysis**
  - ■ **In case of on-chip analysis**

6. Hardware Complexity Analysis

We evaluate BlockHammer's (1) chip area, static power, and access energy consumption using CACTI [99] and (2) circuit latency using Synopsys DC [143]. We demonstrate that Block-Hammer's physical costs are competitive with state-of-the-art RowHammer mitigation mechanisms.

6.2. Latency Analysis

We implement BlockHammer in Verilog HDL and synthe-size our design using Synopsys DC [143] with a 65 nm process technology to evaluate the latency impact on memory accesses. According to our RTL model, which we open source [124], BlockHammer responds to an *"Is this ACT RowHammer-safe?"* query (❶ in Figure 2) in only 0.97 ns. This latency can be hid-den because it is one-to-two orders of magnitude smaller than the row access latency (e.g., 45–50 ns) that DRAM standards (e.g., DDRx, LPDDRx, GDDRx) enforce [36, 53, 55].

Mitigation Mechanism	N_{RH}=32K*						N_{RH}=1K					
	SRAM	CAM	Area		Access Energy	Static Power	SRAM	CAM	Area		Access Energy	Static Power
	KB	KB	mm²	% CPU	(pJ)	(mW)	KB	KB	mm²	% CPU	(pJ)	(mW)
BlockHammer	**51.48**	**1.73**	**0.14**	**0.06**	**20.30**	**22.27**	**441.33**	**55.58**	**1.57**	**0.64**	**99.64**	**220.99**
Dual counting Bloom filters	48.00	-	0.11	0.04	18.11	19.81	384.00	-	0.74	0.30	86.29	158.46
H3 hash functions	-	-	< 0.01	< 0.01	-	-	-	-	< 0.01	< 0.01	-	-
Row activation history buffer	1.73	1.73	0.03	0.01	1.83	2.05	55.58	55.58	0.83	0.34	12.99	62.12
AttackThrottler counters	1.75	-	< 0.01	< 0.01	0.36	0.41	1.75	-	< 0.01	< 0.01	0.36	0.41
PARA [73]	**-**	**-**	**< 0.01**	**-**	**-**	**-**	**-**	**-**	**< 0.01**	**-**	**-**	**-**
ProHIT [137]*	**-**	**0.22**	**< 0.01**	**<0.01**	**3.67**	**0.14**	×	×	×	×	×	×
MrLoc [161]*	**-**	**0.47**	**< 0.01**	**<0.01**	**4.44**	**0.21**	×	×	×	×	×	×
CBT [132]	**16.00**	**8.50**	**0.20**	**0.08**	**9.13**	**35.55**	**512.00**	**272.00**	**3.95**	**1.60**	**127.93**	**535.50**
TWiCE [84]	**23.10**	**14.02**	**0.15**	**0.06**	**7.99**	**21.28**	**738.32**	**448.27**	**5.17**	**2.10**	**124.79**	**631.98**
Graphene [113]	**-**	**5.22**	**0.04**	**0.02**	**40.67**	**3.11**	**-**	**166.03**	**1.14**	**0.46**	**917.55**	**93.96**

* PRoHIT [137] and MRLoc [161] do *not* provide a concrete discussion on how to adjust their empirically-determined parameters for different N_{RH} values. Therefore, we (1) report their values for a fixed design point that each paper provides for N_{RH}=2K and (2) mark values we cannot estimate using an ×.

Table 4: Per-rank area, access energy, and static power of BlockHammer vs. state-of-the-art RowHammer mitigation mechanisms.

❑ **Analysis based on the paper below**

**BlockHammer: Preventing RowHammer at Low Cost
by Blacklisting Rapidly-Accessed DRAM Rows**

- **Section 6. Hardware Complexity Analysis**

  ▪ **In case of off-chip analysis**

7. Experimental Methodology

We evaluate BlockHammer's effect on a typical DDR4-based memory subsystem's performance and energy consumption as compared to six prior RowHammer mitigation mechanisms [73, 84, 113, 132, 137, 161]. We use Ramulator [77, 125] for performance evaluation and DRAMPower [18] to estimate DRAM energy consumption. We open-source our infrastructure, which implements both BlockHammer and six state-of-the-art Row-Hammer mitigation mechanisms [124]. Table 5 shows our system configuration.

Processor	3.2 GHz, {1,8} core, 4-wide issue, 128-entry instr. window
Last-Level Cache	64-byte cache line, 8-way set-associative, 16 MB
Memory Controller	64-entry each read and write request queues; Scheduling policy: FR-FCFS [122, 164]; Address mapping: MOP [60]
Main Memory	DDR4, 1 channel, 1 rank, 4 bank groups, 4 banks/bank group, 64K rows/bank

Table 5: Simulated system configuration.

Figure 4: Execution time and DRAM energy consumption for benign single-core applications, normalized to baseline.

Figure 5: Performance and DRAM energy consumption for multiprogrammed workloads, normalized to baseline.

DRAM Operations & States

❏ **DRAM Operations & States**
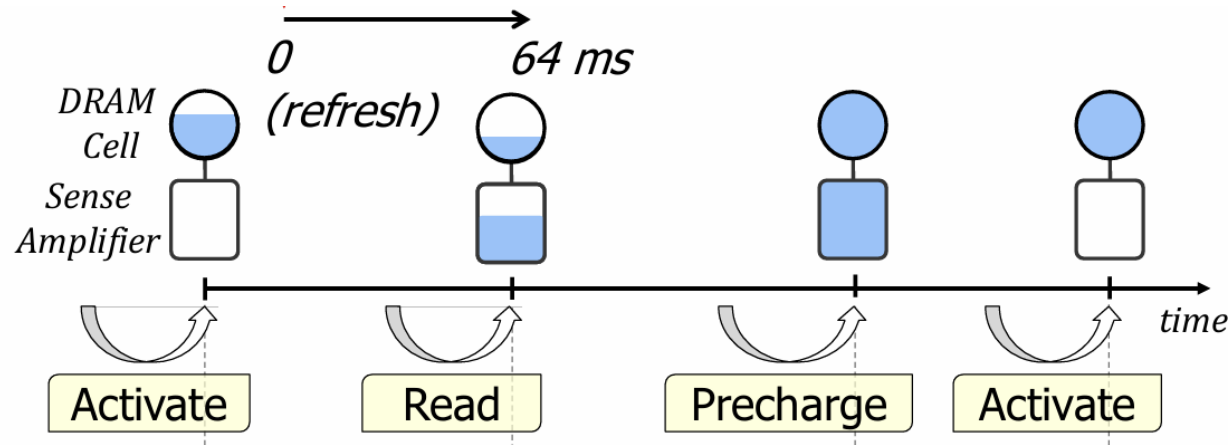
- **Main DRAM states**
  - **Activate**
  - **Read/Write**
  - **Precharge**

```
1   // DRAM.h
2   template <typename T>
3   class DRAM {
4     DRAM<T>* parent;
5     vector<DRAM<T>*>
6       children;
7     T::Level level;
8     int index;
9     // more code...
10  };
```

```
1   // DDR3.h/cpp
2   class DDR3 {
3     enum class Level {
4       Channel, Rank, Bank,
5       Row, Column, MAX
6     };
7
8
9     // more code...
10  };
```
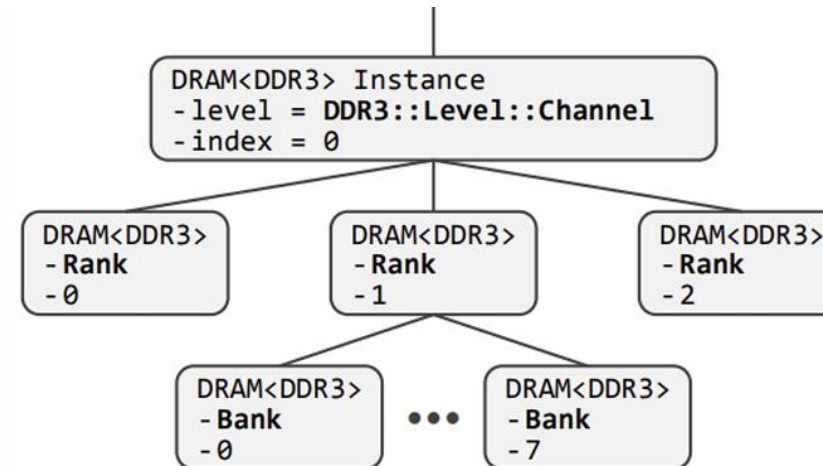
src files <=> DRAM Operation
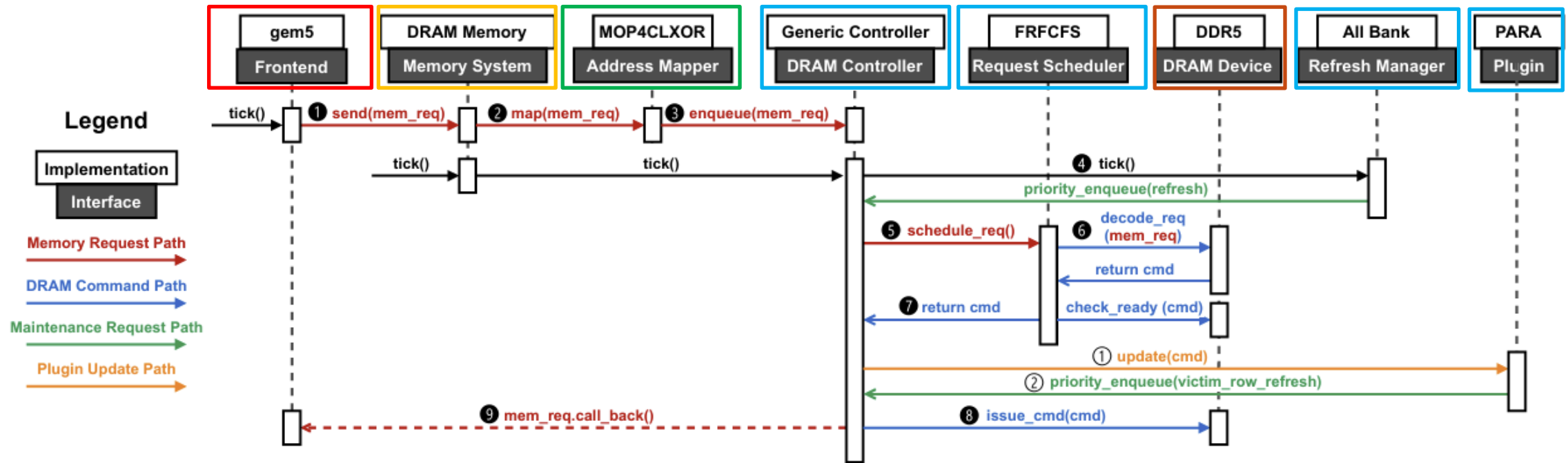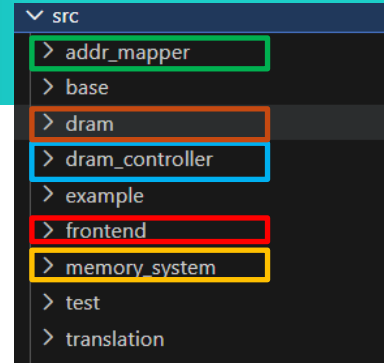
❑ **Simulation Configuration**

Fig. 1: High-level software architecture of Ramulator 2.0 using an example DDR5 system configuration

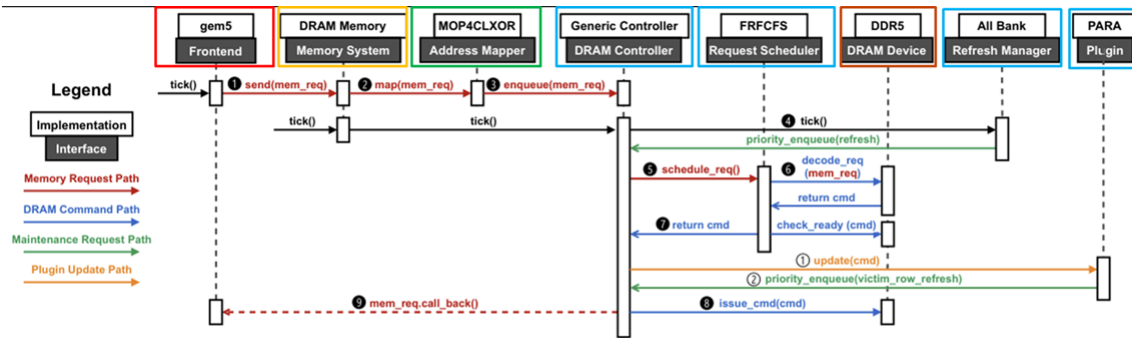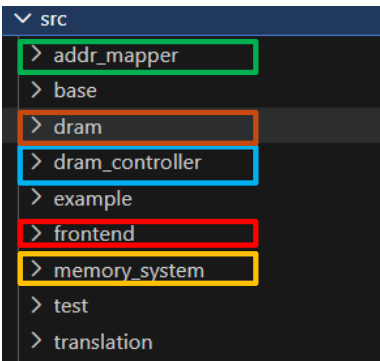# src files <=> DRAM Operation

❑ Simulation Configuration

Fig. 1: High-level software architecture of Ramulator 2.0 using an example DDR5 system configuration

1. **Requests are sent**: Front-end(trace file)에서 Memory Request를 보냄

2. **Memory Addresses are Mapped**: Address Mapper가 Request Address를 DRAM 구조에 맞게 변환

3. **Enqueue**: DRAM Ctrlr의 Buffer에 Request를 넣음

4. **DRAM Ctrlr - Ticking Refresh Manager**: Ctrlr가 Refresh Manager를 호출해 high-priority maintenance request(ex. Refresh)을 추가

5. **DRAM Ctrlr - Request Scheduling**: Request Scheduler에게 최적의 Request을 선택하라고 요청

6. **DRAM Device가 Request 확인**: Scheduler가 DRAM Device Model을 참조해 적합한 Command를 Decode

7. **Issue Command**: DRAM Ctrlr가 DRAM Command를 보냄

8. **Updates the behavior and timing information:** DRAM Command Issue시 State & Timing이 Update

9. **Notify the frontend**: Memory Request가 끝나면 callback으로 frontend에 알림

❑ **main.cpp**

```cpp
13 ∨ int main(int argc, char* argv[]) {
14      // Parse command line arguments
15      argparse::ArgumentParser program("Ramulator", "2.0");
16      program.add_argument("-c", "--config").metavar("\"dumped YAML configuration\"")
17        .help("String dump of the yaml configuration.");
18      program.add_argument("-f", "--config_file").metavar("path-to-configuration-file")
19        .help("Path to a YAML configuration file.");
20      program.add_argument("-p", "--param").metavar("KEY=VALUE")
21        .append()
22        .help("Specify parameter to override in the configuration file. Repeat this option to change multiple parameters.");
```

```cpp
88      // Connect the frontend and the memory system together,
89      // this recursively calls the "setup" function in all instaniated components
90      // so that they can get each other's parameters (if needed) after their initialization
91      frontend->connect_memory_system(memory_system);
92      memory_system->connect_frontend(frontend);
93
94      // Get the relative clock ratio between the frontend and memory system
95      int frontend_tick = frontend->get_clock_ratio();
96      int mem_tick = memory_system->get_clock_ratio();
97
98      int tick_mult = frontend_tick * mem_tick;
99
00      for (uint64_t i = 0;; i++) {
01        if (((i % tick_mult) % mem_tick) == 0) {
02          frontend->tick();
03        }
04
05        if (frontend->is_finished()) {
06          break;
07        }
08
09        if ((i % tick_mult) % frontend_tick == 0) {
10          memory_system->tick();
11        }
12      }
13
14      // Finalize the simulation. Recursively print all statistics from all components
15      frontend->finalize();
16      memory_system->finalize();
17
18      return 0;
19  }
```

main.cpp

1. **Argument 받는 부분**
   - **Options**
   1. -c: command line dump
   2. -f: YAML document
   3. -p: overriding parameters in a YAML document

2. **Long for loop를 통한 tick() 기반 simul**
   1. frontend(core)가 발행한 예상 instructions들을 모두 처리시 is_finished()가 true가 됨

❑example_config.yaml

```
1  ∨ Frontend:
2      impl: SimpleO3
3      clock_ratio: 8
4      num_expected_insts: 500000
5  ∨   traces:
6        - example_inst.trace
7
8  ∨  Translation:
9      impl: RandomTranslation
10     max_addr: 2147483648
11
```

```
13    MemorySystem:
14      impl: GenericDRAM
15      clock_ratio: 3
16
17      DRAM:
18        impl: DDR4
19        org:
20          preset: DDR4_8Gb_x8
21          channel: 1
22          rank: 2
23        timing:
24          preset: DDR4_2400R
25
26      Controller:
27        impl: Generic
28        Scheduler:
29          impl: FRFCFS
30        RefreshManager:
31          impl: AllBank
32        RowPolicy:
33          impl: ClosedRowPolicy
34          cap: 4
35        plugins:
36
37      AddrMapper:
38        impl: RoBaRaCoCh
```

1. **Frontend Interface(IFrontEnd) 부분**
   - **trace file에서 Instruction읽고, Memory Request 생성**
   - *impl: SimpleO3*
     ⇒ Simple Out-of-Order (O3) CPU
   - *clock_ratio: 8*
     ⇒ global CLK 대비 Frontend CLK 속도
   - *num_expected_insts: 500000*
     ⇒ Simulation이 해당 instruction 수에 도달 시 종료
   - *traces*
     ⇒ Instruction trace file(include memory access Inst)
   - *impl: RandomTranslation*
     ⇒ Physical Memory ↔ Virtual Memory 변환
     ⇒ System의 Page Table 등을 간단히 Modeling
   - *max_addr: 2147483648*
     ⇒ Translation 시 address overflow 방지

yaml file

❑ example_config.yaml

```
 1  ∨ Frontend:
 2      impl: SimpleO3
 3      clock_ratio: 8
 4      num_expected_insts: 500000
 5  ∨   traces:
 6        - example_inst.trace
 7
 8  ∨ Translation:
 9      impl: RandomTranslation
10      max_addr: 2147483648
11
```

```
13     MemorySystem:
14       impl: GenericDRAM
15       clock_ratio: 3
16
17     DRAM:
18       impl: DDR4
19       org:
20         preset: DDR4_8Gb_x8
21         channel: 1
22         rank: 2
23       timing:
24         preset: DDR4_2400R
25
26     Controller:
27       impl: Generic
28       Scheduler:
29         impl: FRFCFS
30       RefreshManager:
31         impl: AllBank
32       RowPolicy:
33         impl: ClosedRowPolicy
34         cap: 4
35       plugins:
36
37     AddrMapper:
38       impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- **Frontend의 Request를 받아 DRAM Ctrlr을 통해 처리**
- **Latency, en/dequeue, Timing Constraints 처리**

- *impl: GenericDRAM*
  ⇒ 기본 DRAM 기반 System, Ctrlr와 DRAM을 통합

- *clock_ratio: 3*
  ⇒ global CLK 대비 MemorySystem CLK 속도
  ⇒ 현재: DRAM이 CPU보다 느린 System (= 3:8)

yaml file

☐ example_config.yaml

```
 1 ∨ Frontend:
 2     impl: SimpleO3
 3     clock_ratio: 8
 4     num_expected_insts: 500000
 5 ∨   traces:
 6       - example_inst.trace
 7
 8 ∨ Translation:
 9     impl: RandomTranslation
10     max_addr: 2147483648
11

13 MemorySystem:
14   impl: GenericDRAM
15   clock_ratio: 3
16
17   DRAM:
18     impl: DDR4
19     org:
20       preset: DDR4_8Gb_x8
21       channel: 1
22       rank: 2
23     timing:
24       preset: DDR4_2400R
25
26   Controller:
27     impl: Generic
28     Scheduler:
29       impl: FRFCFS
30     RefreshManager:
31       impl: AllBank
32     RowPolicy:
33       impl: ClosedRowPolicy
34       cap: 4
35     plugins:
36
37   AddrMapper:
38     impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- **DRAM Section**
- *impl: DDR4*
  - ⇒ tick() 시 Timing Check / Command Issue 실행.
- *org: DDR4_8Gb_x8*
  - ⇒ 현재 DRAM preset - 8Gb 용량, x8bit data bus
  - ⇒ Channel/Rank 설정 시 기본 Preset설정을 Override 함
- *timing: DDR4_2400R*
  - ⇒ Timing preset - nRCD등의 Timing Contraint 정의
  - ⇒ 이를 이용해 tick() 시 Latency 계산

```
26     inline static const std::map<std::string, std::vector<int>> timing_presets = {
27       // name       rate  nBL nCL nRCD nRP nRAS nRC  nWR nRTP nCWL nCCDS nCCDL nRRDS nRRDL nWTRS nW
28       {"DDR4_1600J", {1600,  4, 10, 10, 10, 28, 38, 12,  6,  9,   4,   5,  -1,  -1,   2,
29       {"DDR4_1600K", {1600,  4, 11, 11, 11, 28, 39, 12,  6,  9,   4,   5,  -1,  -1,   2,
30       {"DDR4_1600L", {1600,  4, 12, 12, 12, 28, 40, 12,  6,  9,   4,   5,  -1,  -1,   2,
31       {"DDR4_1866L", {1866,  4, 12, 12, 12, 32, 44, 14,  7, 10,   4,   5,  -1,  -1,   3,
32       {"DDR4_1866M", {1866,  4, 13, 13, 13, 32, 45, 14,  7, 10,   4,   5,  -1,  -1,   3,
33       {"DDR4_1866N", {1866,  4, 14, 14, 14, 32, 46, 14,  7, 10,   4,   5,  -1,  -1,   3,
34       {"DDR4_2133N", {2133,  4, 14, 14, 14, 36, 50, 16,  8, 11,   4,   6,  -1,  -1,   3,
35       {"DDR4_2133P", {2133,  4, 15, 15, 15, 36, 51, 16,  8, 11,   4,   6,  -1,  -1,   3,
36       {"DDR4_2133R", {2133,  4, 16, 16, 16, 36, 52, 16,  8, 11,   4,   6,  -1,  -1,   3,
37       {"DDR4_2400P", {2400,  4, 15, 15, 15, 39, 54, 18,  9, 12,   4,   6,  -1,  -1,   3,
38       {"DDR4_2400R", {2400,  4, 16, 16, 16, 39, 55, 18,  9, 12,   4,   6,  -1,  -1,   3,
39       {"DDR4_2400U", {2400,  4, 17, 17, 17, 39, 56, 18,  9, 12,   4,   6,  -1,  -1,   3,
40       {"DDR4_2400T", {2400,  4, 18, 18, 18, 39, 57, 18,  9, 12,   4,   6,  -1,  -1,   3,
```

```cpp
class DDR4 : public IDRAM, public Implementation {
  RAMULATOR_REGISTER_IMPLEMENTATION(IDRAM, DDR4, "DDR4", "DDR4 Device Model")

public:
  inline static const std::map<std::string, Organization> org_presets = {
    //  name          density   DQ  Ch Ra Bg Ba  Ro     Co
    {"DDR4_2Gb_x4",   {2<<10,    4,  {1, 1, 4, 4, 1<<15, 1<<10}}},
    {"DDR4_2Gb_x8",   {2<<10,    8,  {1, 1, 4, 4, 1<<14, 1<<10}}},
    {"DDR4_2Gb_x16",  {2<<10,   16,  {1, 1, 2, 4, 1<<14, 1<<10}}},
    {"DDR4_4Gb_x4",   {4<<10,    4,  {1, 1, 4, 4, 1<<16, 1<<10}}},
    {"DDR4_4Gb_x8",   {4<<10,    8,  {1, 1, 4, 4, 1<<15, 1<<10}}},
    {"DDR4_4Gb_x16",  {4<<10,   16,  {1, 1, 2, 4, 1<<15, 1<<10}}},
    {"DDR4_8Gb_x4",   {8<<10,    4,  {1, 1, 4, 4, 1<<17, 1<<10}}},
    {"DDR4_8Gb_x8",   {8<<10,    8,  {1, 1, 4, 4, 1<<16, 1<<10}}},
    {"DDR4_8Gb_x16",  {8<<10,   16,  {1, 1, 2, 4, 1<<16, 1<<10}}},
    {"DDR4_16Gb_x4",  {16<<10,   4,  {1, 1, 4, 4, 1<<18, 1<<10}}},
    {"DDR4_16Gb_x8",  {16<<10,   8,  {1, 1, 4, 4, 1<<17, 1<<10}}},
    {"DDR4_16Gb_x16", {16<<10,  16,  {1, 1, 2, 4, 1<<17, 1<<10}}},
  };
```

```cpp
class DDR4 : public IDRAM, public Implementation {
    void tick() override {

    void init() override {
      RAMULATOR_DECLARE_SPECS();
      set_organization();
      set_timing_vals();

      set_actions();
      set_preqs();
      set_rowhits();
      set_rowopens();
      set_powers();

      create_nodes();
    };
```

yaml file

☐ example_config.yaml

```
 1  ∨ Frontend:
 2       impl: SimpleO3
 3       clock_ratio: 8
 4       num_expected_insts: 500000
 5  ∨    traces:
 6         - example_inst.trace
 7
 8  ∨  Translation:
 9       impl: RandomTranslation
10       max_addr: 2147483648
11

13  MemorySystem:
14      impl: GenericDRAM
15      clock_ratio: 3
16
17      DRAM:
18        impl: DDR4
19        org:
20          preset: DDR4_8Gb_x8
21          channel: 1
22          rank: 2
23        timing:
24          preset: DDR4_2400R
25
26      Controller:
27        impl: Generic
28        Scheduler:
29          impl: FRFCFS
30        RefreshManager:
31          impl: AllBank
32        RowPolicy:
33          impl: ClosedRowPolicy
34          cap: 4
35        plugins:
36
37      AddrMapper:
38        impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- **Controller Section**
- *impl: Generic*
  - ⇒ Generic - 기본 Ctrlr
  - ⇒ Request Queue/Scheduling 등 관리
- *Scheduler - impl: FRFCFS*
  - ⇒ FRFCFS - First-Ready First-Come-First-Serve
    - ⇒ 준비된 Request를 Queue에서 꺼내 우선 처리
- *RefreshManager - impl: AllBank*
  - ⇒ AllBank - 모든 Bank simultaneous Refresh
- *RowPolicy - impl: ClosedRowPolicy*
  - ⇒ ClosedRowPolicy - 사용 후 Row 즉시 닫음(Precharge)
  - ⇒ cap:4 - 열려있는 Row 최대 수 제한
- *plugins*
  - ⇒ 현재 Ramulator에서는 Row Hammering 완화 기법을 plugin으로 제공해줌

yaml file

❑ example_config.yaml

```
1  ∨ Frontend:
2       impl: SimpleO3
3       clock_ratio: 8
4       num_expected_insts: 500000
5  ∨   traces:
6           - example_inst.trace
7
8  ∨ Translation:
9       impl: RandomTranslation
10      max_addr: 2147483648
11
13  MemorySystem:
14      impl: GenericDRAM
15      clock_ratio: 3
16
17      DRAM:
18        impl: DDR4
19        org:
20          preset: DDR4_8Gb_x8
21          channel: 1
22          rank: 2
23        timing:
24          preset: DDR4_2400R
25
26      Controller:
27        impl: Generic
28        Scheduler:
29          impl: FRFCFS
30        RefreshManager:
31          impl: AllBank
32        RowPolicy:
33          impl: ClosedRowPolicy
34          cap: 4
35        plugins:
36
37      AddrMapper:
38        impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- **AddrMapper Section**

- *impl: RoBaRaCoCh*
  - ⇒ Row-Bank-Rank-Column-Channel Mapping Scheme
  - ⇒ Requested Addres 변환(Physical → DRAM Vector)

  [Physical → DRAM Vector Example]
  - ⇒ Physical Address: 0x12345678
  - ⇒ DRAM Vector:
    [Channel:0, Rank:1, Bank:2, Row:128, Column:512]

trace file

❑ example_inst.trace & trace.cpp & core.cpp

- **simpleO3 CPU model기준**

  - **Ramulator는 "Memory" Simulator**

  - **Memory 명령어만 취급하기에, 3가지로만 Instruction을 분리한다.**

    1. **Not Memory Operation**

    2. **Load**

    3. **Store**

- **따라서, simpleO3 기반 trace파일:**
  - **1st column은 Not Memory Operation Cycle 수 (or ticks 수)**
  - **2nd column은 load operation address**
  - **3rd column은 store operation address**


```
≡ example_inst.trace
1   3  20734016
2   1  20846400
3   6  20734208
4   1  20846400
5   8  20841280   20841280
6   0  20734144
7   2  20918976   20734016
8   1  20846400
9
```

- **각 line은 load(& store)동작을 나타낸다.**

- **1st line : 3cycle동안 stall → load**

- **5th line : 8cycle동안 stall → load → store**

trace file

☐ example_inst.trace & trace.cpp & core.cpp

- **Trace file을 arg로 받아, Frontend (ex: simpleO3.cpp)에서 처리되어 메모리 접근 request를 생성**

- **simpleO3 CPU model기준: trace의 token[0]: bubble / token[1]: load address / token[2]: store address**

trace file result

❑ ./ramulator2 -f ./example_config.yaml

```
root@947e591ed45c:/workspace# ./ramulator2 -f ./example_config.yaml
Frontend:
  impl: SimpleO3
  memory_access_cycles_recorded_core_0: 61
  cycles_recorded_core_0: 216815
  llc_mshr_unavailable: 0
  llc_read_misses: 37
  llc_read_access: 133336
  llc_write_misses: 8
  llc_write_access: 33334
  llc_eviction: 0
  num_expected_insts: 500000
  Translation:
    impl: RandomTranslation


MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 0
  total_num_read_requests: 6
  memory_system_cycles: 81306
  DRAM:
    impl: DDR4
  AddrMapper:
    impl: RoBaRaCoCh
```

```
Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 46.5
  read_queue_len_avg_0: 0.00232455181
  write_queue_len_0: 0
  queue_len_0: 245
  num_other_reqs_0: 0
  num_write_reqs_0: 0
  read_latency_0: 279
  priority_queue_len_avg_0: 0.000688756059
  row_hits_0: 2
  priority_queue_len_0: 56
  row_misses_0: 4
  row_conflicts_0: 0
  read_row_misses_0: 4
  queue_len_avg_0: 0.00301330769
  read_row_conflicts_core_0: 0
  read_row_hits_0: 2
  write_queue_len_avg_0: 0
  read_row_conflicts_0: 0
  write_row_misses_0: 0
  write_row_conflicts_0: 0
  read_queue_len_0: 189
  write_row_hits_0: 0
  read_row_hits_core_0: 2
  read_row_misses_core_0: 4
  num_read_reqs_0: 6
  Scheduler:
    impl: FRFCFS
  RefreshManager:
    impl: AllBank


RowPolicy:
  impl: ClosedRowPolicy
  num_close_reqs: 0
```

- 기본 yaml file setting을 이용.
  => trace file의 Inst 실행결과가 분석됨

- 1. Frontend:
  CPU ↔ Memory<Cache> 접근 분석

- 2. MemorySystem<DRAM> 관점 분석
  - Request 수
  - Cycle 수
  - Latency
  - Queue 길이
  - Row Hit / Miss

trace file result analysis

☐ **./ramulator2 -f ./example_config.yaml**

- 분석 시 사용한 **trace file**은 오른쪽의 **example_inst**이다.

- **예상 simulation결과**는 다음과 같아야 한다 → *이를 실제 Results와 비교한다.*

- **8번의 load (address: 0x20846400가 3번 나온다)**

- **2번의 write (1st Inst의 address, 5th Inst의 address에 Write [이미 load된 address 접근 → cache hit])**

    ▪ **DRAM 관점**

    → **이를 통해서 6번의 DRAM Read Access를 요구하는 것을 예측 가능**

    • **나머지 중복 load들은 cache line에 hit되어 DRAM Access X**

    → **Write의 경우 이미 Load된 address에 write하므로 cache이용 : 0번의 DRAM Write Access**

    ▪ **SRAM(cache) 관점**

    → **적은 memory 주소만 반복 접근 하므로 cache eviction X (이전에 load된 addr들은 다 hit 됨)**

    → **Initial Read Misses: 6번 / Initial Write Misses: 0번**

trace file result analysis

❑ ./ramulator2 -f ./example_config.yaml

```
root@947e591ed45c:/workspace# ./ramulator2 -f ./example_config.yaml
Frontend:
  impl: SimpleO3
  memory_access_cycles_recorded_core_0: 61
  cycles_recorded_core_0: 216815
  llc_mshr_unavailable: 0
  llc_read_misses: 37
  llc_read_access: 133336
  llc_write_misses: 8
  llc_write_access: 33334
  llc_eviction: 0
  num_expected_insts: 500000
  Translation:
    impl: RandomTranslation

MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 0
  total_num_read_requests: 6
  memory_system_cycles: 81306
  DRAM:
    impl: DDR4
  AddrMapper:
    impl: RoBaRaCoCh
```

```
Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 46.5
  read_queue_len_avg_0: 0.00232455181
  write_queue_len_0: 0
  queue_len_0: 245
  num_other_reqs_0: 0
  num_write_reqs_0: 0
  read_latency_0: 279
  priority_queue_len_avg_0: 0.000688756059
  row_hits_0: 2
  priority_queue_len_0: 56
  row_misses_0: 4
  row_conflicts_0: 0
  read_row_misses_0: 4
  queue_len_avg_0: 0.00301330769
  read_row_conflicts_core_0: 0
  read_row_hits_0: 2
  write_queue_len_avg_0: 0
  read_row_conflicts_0: 0
  write_row_misses_0: 0
  write_row_conflicts_0: 0
  read_queue_len_0: 189
  write_row_hits_0: 0
  read_row_hits_core_0: 2
  read_row_misses_core_0: 4
  num_read_reqs_0: 6
  Scheduler:
    impl: FRFCFS
  RefreshManager:
    impl: AllBank


  RowPolicy:
    impl: ClosedRowPolicy
    num_close_reqs: 0
```

```
52    const SimpleO3Core::Trace::Inst& SimpleO3Core::Trace::get_next_inst(
53      const Inst& inst = m_trace[m_curr_trace_idx];
54      m_curr_trace_idx = (m_curr_trace_idx + 1) % m_trace_length;
55      return inst;
56    }
```

짧은 trace지만 반복 실행 되기에 많은 access 발생

1. **Frontend**
   - **Last Level Cache(LLC) 분석**
     - **Core 0의 Memory(Cache 포함) 접근 Cycles**

     - **LLC의 Read Request Cache Miss 수**
     - **LLC의 Read Access 수**
     - **Miss Status Handling Register(MSHR) unavailable cnt**
       → **MSHR가 가득 차서 Request 거부된 횟수**

     - **LLC의 Write Request Cache Miss 수**
     - **LLC의 Write Access 수**
     - **LLC의 Eviction 수**
       → **Cache Line 교체된 수 (DRAM Write 유발)**

이전의 예상치와 다르게 LLC의 miss수가 훨씬 많음
→ **dependency check, reorder buffer 등 때문일 수 있음**
→ **LLC operation 코드 분석이 필요 → OoO CPU의 특성 때문임**
→ **bool SimpleO3LLC::send(Request req)**

trace file result analysis

❑ ./ramulator2 -f ./example_config.yaml

```
root@947e591ed45c:/workspace# ./ramulator2 -f ./example_config.yaml
Frontend:
  impl: SimpleO3
  memory_access_cycles_recorded_core_0: 61
  cycles_recorded_core_0: 216815
  llc_mshr_unavailable: 0
  llc_read_misses: 37
  llc_read_access: 133336
  llc_write_misses: 8
  llc_write_access: 33334
  llc_eviction: 0
  num_expected_insts: 500000
Translation:
  impl: RandomTranslation

MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 0
  total_num_read_requests: 6
  memory_system_cycles: 81306
DRAM:
  impl: DDR4
AddrMapper:
  impl: RoBaRaCoCh
```

```
Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 46.5
  read_queue_len_avg_0: 0.00232455181
  write_queue_len_0: 0
  queue_len_0: 245
  num_other_reqs_0: 0
  num_write_reqs_0: 0
  read_latency_0: 279
  priority_queue_len_avg_0: 0.000688756059
  row_hits_0: 2
  priority_queue_len_0: 56
  row_misses_0: 4
  row_conflicts_0: 0
  read_row_misses_0: 4
  queue_len_avg_0: 0.00301330769
  read_row_conflicts_core_0: 0
  read_row_hits_0: 2
  write_queue_len_avg_0: 0
  read_row_conflicts_0: 0
  write_row_misses_0: 0
  write_row_conflicts_0: 0
  read_queue_len_0: 189
  write_row_hits_0: 0
  read_row_hits_core_0: 2
  read_row_misses_core_0: 4
  num_read_reqs_0: 6
Scheduler:
  impl: FRFCFS
RefreshManager:
  impl: AllBank

RowPolicy:
  impl: ClosedRowPolicy
  num_close_reqs: 0
```
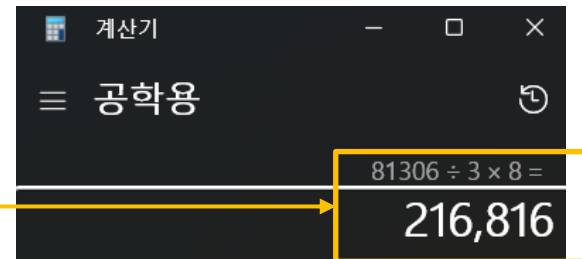
2. MemorySystem Interface
- **DRAM 분석**
  - Read/Write 외 other requests 수   *이전의 예상치와 일치*
  - **write requests 수 → 현재 0 (cache eviction 없었기에)**
  - **read requests 수 → 현재 6 (initial cache line load)**
  - **Memory system cycles**
    → **Memory CLK과 Proc CLK이 다르기에 (= 3:8)**

DRAM CLK : Processor CLK = 3:8

```
계산기
≡ 공학용
                              81306 ÷ 3 × 8 =
                              216,816
```

```
108    inline static constexpr ImplDef m_requests = {
109      "read", "write", "all-bank-refresh", "open-row", "close-row"
110    };
111
112    inline static const ImplLUT m_request_translations = LUT (
113      m_requests, m_commands, {
114        {"read", "RD"}, {"write", "WR"}, {"all-bank-refresh", "REFab"},
115        {"open-row", "ACT"}, {"close-row", "PRE"}
116      }
117    );
```

Other requests: read/write를 제외한 open-row와 같은 request등을 의미

trace file result analysis

❑ **./ramulator2 -f ./example_config.yaml**

```
● root@947e591ed45c:/workspace# ./ramulator2 -f ./example_config.yaml
Frontend:
  impl: SimpleO3
  memory_access_cycles_recorded_core_0: 61
  cycles_recorded_core_0: 216815
  llc_mshr_unavailable: 0
  llc_read_misses: 37
  llc_read_access: 133336
  llc_write_misses: 8
  llc_write_access: 33334
  llc_eviction: 0
  num_expected_insts: 500000
  Translation:
    impl: RandomTranslation


MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 0
  total_num_read_requests: 6
  memory_system_cycles: 81306
  DRAM:
    impl: DDR4
  AddrMapper:
    impl: RoBaRaCoCh
```

```
Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 46.5
  read_queue_len_avg_0: 0.00232455181
  write_queue_len_0: 0
  queue_len_0: 245
  num_other_reqs_0: 0
  num_write_reqs_0: 0
  read_latency_0: 279
  priority_queue_len_avg_0: 0.000688756059
  row_hits_0: 2
  priority_queue_len_0: 56
  row_misses_0: 4
  row_conflicts_0: 0
  read_row_misses_0: 4
  queue_len_avg_0: 0.00301330769
  read_row_conflicts_core_0: 0
  read_row_hits_0: 2
  write_queue_len_avg_0: 0
  read_row_conflicts_0: 0
  write_row_misses_0: 0
  write_row_conflicts_0: 0
  read_queue_len_0: 189
  write_row_hits_0: 0
  read_row_hits_core_0: 2
  read_row_misses_core_0: 4
  num_read_reqs_0: 6
  Scheduler:
    impl: FRFCFS
  RefreshManager:
    impl: AllBank


  RowPolicy:
    impl: ClosedRowPolicy
    num_close_reqs: 0
```

2. **MemorySystem Interface**
   - **DRAM Controller 분석**
     - **abcd**

trace file result

❏ power는 어떻게 측정하느냐면

- yaml file에 아래 drampower_enable 옵션을 넣음
- dram.h -> DDR4.cpp 의 power함수 enable됨

```
17 ∨    DRAM:
18         impl: DDR4
19 ∨       org:
20           preset: DDR4_8Gb_x8
21           channel: 1
22           rank: 2
23 ∨       timing:
24           preset: DDR4_2400R
25         drampower_enable: true
26         # power_debug: true   # option: debug log
27 ∨       voltage:
28           preset: Default  # option: voltage preset
29 ∨       current:
30           preset: Default  # option: current preset
31
```

```
DRAM:
    impl: DDR4
    active_cycles_rank1: 9361
    pre_background_energy_rank1: 3850.965247500003
    total_background_energy_rank0: 4424.0171849999997
    idle_cycles_rank1: 68489
    total_cmd_energy: 3203.0342735999993
    total_cmd_energy_rank0: 1601.8769927999997
    total_energy_rank0: 6025.8941777999989
    act_background_energy_rank0: 572.15229750000003
    pre_background_energy_rank0: 3851.8648874999999
    active_cycles_rank0: 9345
    act_background_energy_rank1: 573.13190550000002
    total_energy: 12051.1486116
    idle_cycles_rank0: 68505
    total_energy_rank1: 6025.2544338000007
    total_background_energy_rank1: 4424.097153000006
    total_background_energy: 8848.1143379999994
    total_cmd_energy_rank1: 1601.1572807999996
AddrMapper:
    impl: RoBaRaCoCh
```

trace file result

❏ power는 어떻게 측정하느냐면

- **power_debug option enable 시**
- **오른쪽과 같은 power log 확인 가능**

```
17 ∨    DRAM:
18        impl: DDR4
19 ∨      org:
20          preset: DDR4_8Gb_x8
21          channel: 1
22          rank: 2
23 ∨      timing:
24          preset: DDR4_2400R
25        drampower_enable: true
26        # power_debug: true  # option: debug log
27 ∨      voltage:
28          preset: Default  # option: voltage preset
29 ∨      current:
30          preset: Default  # option: current preset
31
```

```
root@947e591ed45c:/workspace# ./ramulator2 -f ./example_config_power_en.yaml
[Power] Rank0 ------ACT------ @ 19
[Power] Rank0 Rank is idle. idle_cycles: 19    active_start_cycle: 19 @ 19
[Power] Rank0 Bank0 Incrementing ACT counter. @ 19
[Power] Rank1 ------ACT------ @ 20
[Power] Rank1 Rank is idle. idle_cycles: 20    active_start_cycle: 20 @ 20
[Power] Rank1 Bank2 Incrementing ACT counter. @ 20
[Power] Rank0 ------ACT------ @ 23
[Power] Rank0 Bank3 Incrementing ACT counter. @ 23
[Power] Rank1 ------ACT------ @ 24
[Power] Rank1 Bank3 Incrementing ACT counter. @ 24
[Power] Rank0 Bank0 Incrementing RD counter. @ 35
[Power] Rank0 Bank3 Incrementing RD counter. @ 39
[Power] Rank0 Bank0 Incrementing RD counter. @ 43
[Power] Rank1 Bank2 Incrementing RD counter. @ 49
[Power] Rank1 Bank3 Incrementing RD counter. @ 53
[Power] Rank0 Bank0 Incrementing RD counter. @ 59
[Power] Rank0 ------PREA------ @ 9364
[Power] Rank0 Incrementing PRE counter. @ 9364
[Power] Rank0 Rank is not idle. active_cycles: 9345    idle_start_cycle: 9364 @ 9364
[Power] Rank0 ------REFab------ @ 9380
[Power] Rank0 Refresh starts. idle_cycles: 35 @ 9380
[Power] Rank1 ------PREA------ @ 9381
[Power] Rank1 Incrementing PRE counter. @ 9381
[Power] Rank1 Rank is not idle. active_cycles: 9361    idle_start_cycle: 9381 @ 9381
[Power] Rank1 ------REFab------ @ 9397
[Power] Rank1 Refresh starts. idle_cycles: 36 @ 9397
[Power] Rank0 ------REFab_end------ @ 9812
[Power] Rank0 Refresh ends. idle_start_cycle: 9812 @ 9812
[Power] Rank1 ------REFab_end------ @ 9829
[Power] Rank1 Refresh ends. idle_start_cycle: 9829 @ 9829
[Power] Rank0 ------REFab------ @ 18728
[Power] Rank0 Refresh starts. idle_cycles: 8951 @ 18728
[Power] Rank1 ------REFab------ @ 18729
[Power] Rank1 Refresh starts. idle_cycles: 8936 @ 18729
[Power] Rank0 ------REFab_end------ @ 19160
[Power] Rank0 Refresh ends. idle_start_cycle: 19160 @ 19160
[Power] Rank1 ------REFab_end------ @ 19161
[Power] Rank1 Refresh ends. idle_start_cycle: 19161 @ 19161
[Power] Rank0 ------REFab------ @ 28092
```

yaml/trace file for direct Read/Write Request

❑ **example_config_dram.yaml & example_dram.trace & readwrite_trace.cpp**

```yaml
! example_config_dram.yaml
1    Frontend:
2      impl: ReadWriteTrace
3      path: example_dram.trace  # trace 파일 경로 (상대/절대)
4      clock_ratio: 1
5
6      Translation:
7        impl: RandomTranslation
8        max_addr: 2147483648
9
10
11   MemorySystem:
12     impl: GenericDRAM
13     clock_ratio: 1
14
15     DRAM:
16       impl: DDR4
17       org:
18         preset: DDR4_8Gb_x8
```

```
≡ example_dram.trace
1    R 0,0,0,0,0
2    W 0,0,0,0,128
3    R 0,0,0,1,0
4    W 0,0,0,1,256
5    R 0,0,1,0,0
6    W 0,0,1,0,512
7    R 0,0,0,1024,0
8    W 0,0,0,1024,0
9
```

```cpp
39   void tick() override {
40       const Trace& t = m_trace[m_curr_trace_idx];
41       m_memory_system->send({t.addr_vec, t.is_write ? Request::Type::Write : Request::Type::Read});
42       // m_curr_trace_idx = (m_curr_trace_idx + 1) % m_trace_length;
43       m_curr_trace_idx++; // revised by MinsungKim yonsei ISL lab - 2026.01.09.
44   };
```

- **Trace file을 arg로 받아, Frontend (ex: readwrite_trace.cpp)에서 처리되어 메모리 접근 request를 생성**

– **Read-Write Trace:** This frontend expects a trace file containing read (R) or store (W) requests with a *DRAM address vector* (i.e., which channel, rank, bank, row, column). Each line in the trace has the following format: **R/W ⟨comma-separated addr vector⟩:**

❑ Simulation Result

결과가 나오긴하는데 말이 안되는 수치를 가짐

```
Frontend:
  impl: ReadWriteTrace

MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 4
  total_num_read_requests: 4
  memory_system_cycles: 7
  DRAM:
    impl: DDR4
  AddrMapper:
    impl: RoBaRaCoCh
```

```
Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 4.5
  read_queue_len_avg_0: 0.571428597
  write_queue_len_0: 12
  queue_len_0: 16
  num_other_reqs_0: 0
  num_write_reqs_0: 4
  read_latency_0: 18
  priority_queue_len_avg_0: 0
  row_hits_0: 0
  priority_queue_len_0: 0
  row_misses_0: 1
  row_conflicts_0: 0
  read_row_misses_0: 1
  queue_len_avg_0: 2.28571439
  read_row_conflicts_core_0: 0
  read_row_hits_0: 0
  write_queue_len_avg_0: 1.71428573
  read_row_conflicts_0: 0
  write_row_misses_0: 0
  write_row_conflicts_0: 0
  read_queue_len_0: 4
  write_row_hits_0: 0
  read_row_hits_core_0: 0
  read_row_misses_core_0: 0
  num_read_reqs_0: 4
  Scheduler:
    impl: FRFCFS
  RefreshManager:
    impl: AllBank


  RowPolicy:
    impl: ClosedRowPolicy
    num_close_reqs: 0
```
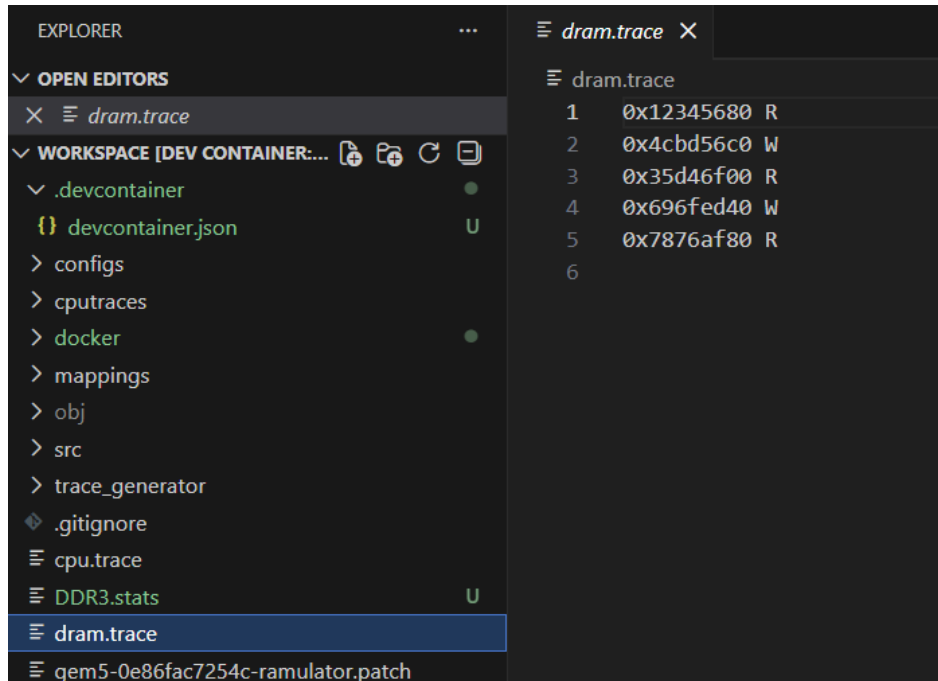
Currently, Ramulator 2 implements the following three frontends:

- **Memory Trace Frontend:** This frontend reads an input trace file that contains *main memory requests* of an application. It *sequentially* sends these requests to the memory system. This mode does not model any system in sufficient detail to perform timing simulations. Because of that, the Memory Trace Frontend is better suited for testing/debugging the functionality of newly added features. In the main tasks of this assignment, we will *not* use this frontend. Still, feel free to use this mode to test and debug your newly implemented features. Currently, Ramulator 2 implements the following two memory trace frontends:

- **Ramulator2.0은 Read/Write Request의 functionality만 검증**
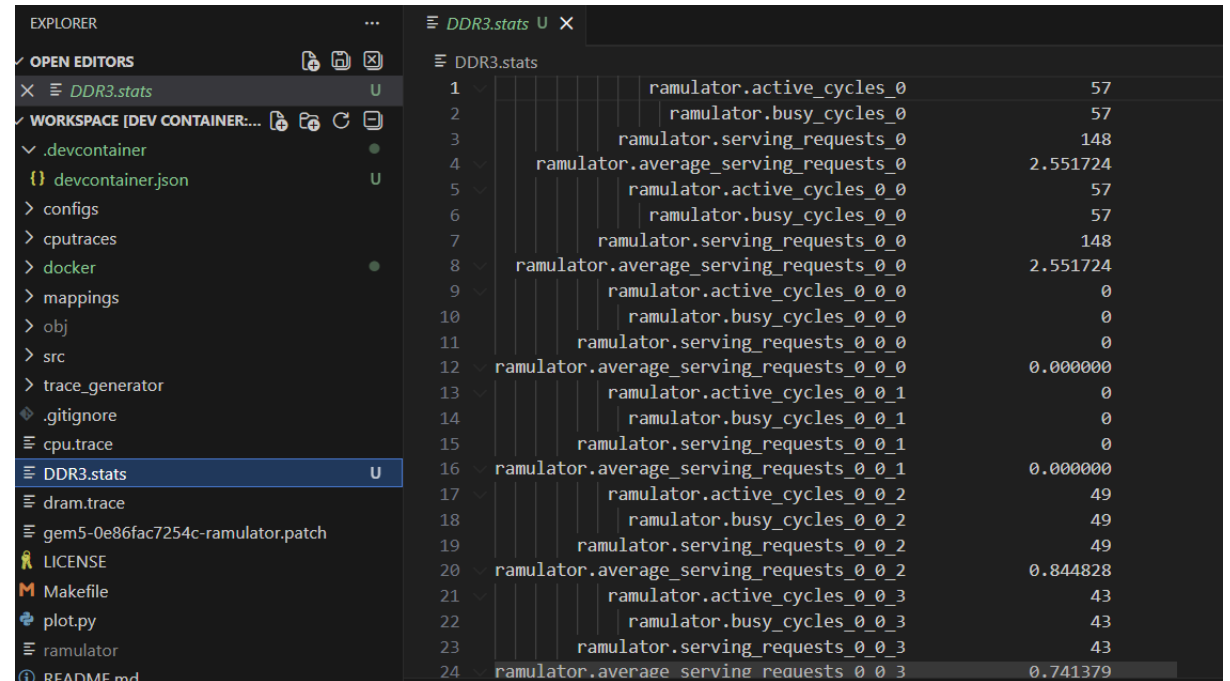
- **Cycle Accurate하지 않음**

yaml/trace file for direct Read/Write Request

❑ **Simulation Result of Ramulator1.0**

Ramulator1.0은 납득할 수준의 결과가 나옴

Ramulator2.0의 전체 시스템을 끝까지 리뷰 후 Ramulator1.0의 DRAM subsystem만 simulation한 결과를 바탕으로 다시 분석 예정

src/frontend folder

❑ simpleO3 Core Operations

• **class SimpleO3 – init()**

```cpp
void init() override {
    m_clock_ratio = param<uint>("clock_ratio").required();

    // Core params
    std::vector<std::string> trace_list = param<std::vector<std::string>>("traces").desc("A list of traces.").required();
    m_num_cores = trace_list.size();

    int ipc   = param<int>("ipc").desc("IPC of the SimpleO3 core.").default_val(4);
    int depth = param<int>("inst_window_depth").desc("Instruction window size of the SimpleO3 core.").default_val(128);

    // LLC params
    int llc_latency        = param<int>("llc_latency").desc("Aggregated latency of the LLC.").default_val(47);
    int llc_linesize_bytes = param<int>("llc_linesize").desc("LLC cache line size in bytes.").default_val(64);
    int llc_associativity  = param<int>("llc_associativity").desc("LLC set associativity.").default_val(8);
    int llc_capacity_per_core = parse_capacity_str(param<std::string>("llc_capacity_per_core").desc("LLC capacity per core.").default_val("2MB"));
    int llc_num_mshr_per_core = param<int>("llc_num_mshr_per_core").desc("Number of LLC MSHR entries per core.").default_val(16);

    // Simulation parameters
    m_num_expected_insts = param<int>("num_expected_insts").desc("Number of instructions that the frontend should execute.").required();

    // Create address translation module
    m_translation = create_child_ifce<ITranslation>();

    // Create the LLC
    m_llc = new SimpleO3LLC(llc_latency, llc_capacity_per_core * m_num_cores, llc_linesize_bytes, llc_associativity, llc_num_mshr_per_core * m_num_cores);
    // m_llc->deserialize(serialization_filename);
    // m_llc->serialize(serialization_filename);

    // Create the cores
    for (int id = 0; id < m_num_cores; id++) {
        SimpleO3Core* core = new SimpleO3Core(id, ipc, depth, m_num_expected_insts, trace_list[id], m_translation, m_llc);
        core->m_callback = [this](Request& req){return this->receive(req);} ;
        m_cores.push_back(core);
    }

    m_logger = Logging::create_logger("SimpleO3");

    // Register the stats
    register_stat(m_num_expected_insts).name("num_expected_insts");
    register_stat(m_llc->s_llc_eviction).name("llc_eviction");
    register_stat(m_llc->s_llc_read_access).name("llc_read_access");
    register_stat(m_llc->s_llc_write_access).name("llc_write_access");
    register_stat(m_llc->s_llc_read_misses).name("llc_read_misses");
    register_stat(m_llc->s_llc_write_misses).name("llc_write_misses");
    register_stat(m_llc->s_llc_mshr_unavailable).name("llc_mshr_unavailable");

    for (int core_id = 0; core_id < m_cores.size(); core_id++) {
        // register_stat(m_cores[core_id]->s_insts_retired).name("cycles_retired_core_{}", core_id);
        register_stat(m_cores[core_id]->s_cycles_recorded).name("cycles_recorded_core_{}", core_id);
        register_stat(m_cores[core_id]->s_mem_access_cycles).name("memory_access_cycles_recorded_core_{}", core_id);
    }
}
```

1.1. Core Parameters를 yaml file에서 가져옴 → clock_ratio / traces (ipc, depth 하드 코딩)

1.2. LLC Parameters (latency, cache line size, 몇 assosiative cahce인지, MSHR 수, cache size) 하드 코딩

1.3. Address Translation / LLC / Cores(trace file의 수만큼 생성 됨) Instance 생성
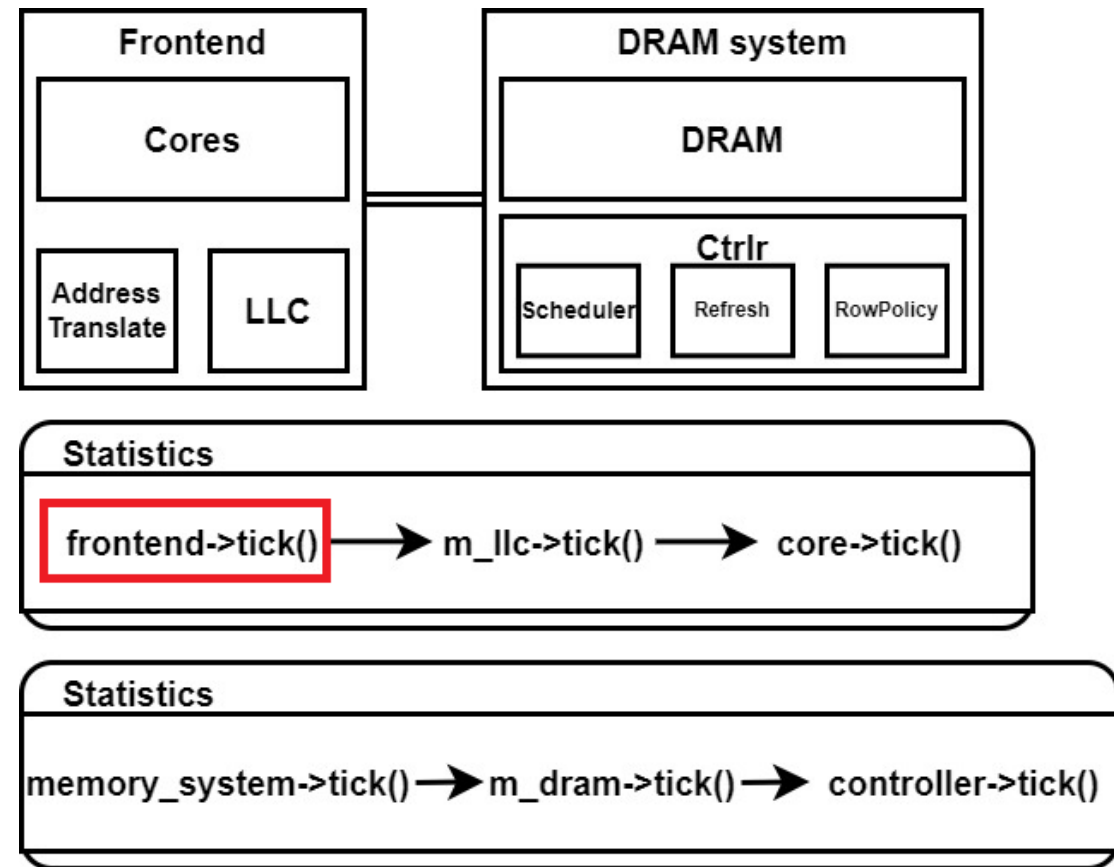
1.4. LLC의 동작 통계 변수들은 registration class에 등록

❑ simpleO3 Core Operations

- **class SimpleO3 – tick() → frontend->tick()**

```
81    void tick() override {
82      m_clk++;
83
84      if(m_clk % 10000000 == 0) {
85        m_logger->info("Processor Heartbeat {} cycles.", m_clk);
86      }
87
88      m_llc->tick();
89      for (auto core : m_cores) {
90        core->tick();
91      }
92    }
```

- **m_clk (global clk) 증가**

- **아래의 순서로 frontend 동작**

- **Last Level Cache tick() → Core tick()**
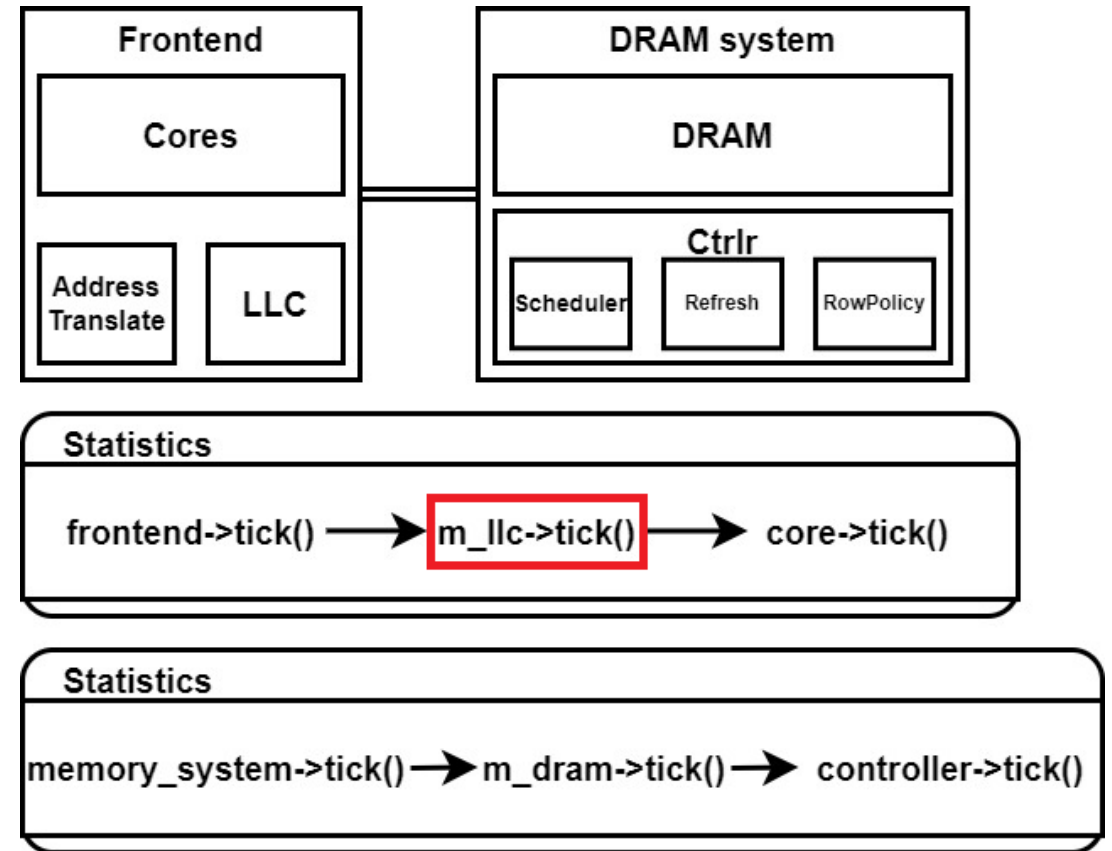
❑ simpleO3 Core Operations

- **class SimpleO3 – tick() → m_llc->tick()**

```cpp
20  void SimpleO3LLC::tick() {
21      m_clk++;
22
23      // Send miss requests to the memory system when LLC latency is met
24      // TODO: Optimization by assuming in-order issue?
25      auto it = m_miss_list.begin();
26      while (it != m_miss_list.end()) {
27          if (m_clk >= it->first) {
28              if (!m_memory_system->send(it->second)) {
29                  it++;
30              }
31              else {
32                  it = m_miss_list.erase(it);
33              }
34          } else {
35              it++;
36          }
37      }
38
39      // call hit request callback when LLC latency is met
40      it = m_hit_list.begin();
41      while (it != m_hit_list.end()) {
42          if (m_clk >= it->first) {
43              std::vector<Request> _req_v{it->second};
44              m_receive_requests[it->second.addr] = _req_v;
45
46              it->second.callback(it->second);
47              it = m_hit_list.erase(it);
48          }
49          else {
50              it++;
51          }
52      }
53  };
```

```cpp
207  void SimpleO3LLC::evict_line(CacheSet_t& set, CacheSet_t::iterator victim_it) {
208      DEBUG_LOG(DSIMPLEO3LLC, m_logger, "Evicting {}.", victim_it->addr);
209      s_llc_eviction++;
210
211      // Generate writeback request if victim line is dirty
212      if (victim_it->dirty) {
213          Request writeback_req(victim_it->addr, Request::Type::Write);
214          m_miss_list.push_back(std::make_pair(m_clk + m_latency, writeback_req))
215
216          DEBUG_LOG(DSIMPLEO3LLC, m_logger, "Writeback Request will be issued at Clk={}.", m_clk + m_latency);
217      }
218
219      set.erase(victim_it);
220  }
```

- **Miss List에서 지연 끝난 Missed Request을 MemorySystem으로 전달**
- **LLC latency(m_latency) Simulation & Send missed req to DRAM**
- **현재: LLC latency modeling 값 → m_latency = 47**

❑ simpleO3 Core Operations

- **class SimpleO3 – tick() → m_llc->tick()**

```
20  void SimpleO3LLC::tick() {
21      m_clk++;
22
23      // Send miss requests to the memory system when LLC latency is met
24      // TODO: Optimization by assuming in-order issue?
25      auto it = m_miss_list.begin();
26      while (it != m_miss_list.end()) {
27          if (m_clk >= it->first) {
28              if (!m_memory_system->send(it->second)) {
29                  it++;
30              }
31              else {
32                  it = m_miss_list.erase(it);
33              }
34          } else {
35              it++;
36          }
37      }
38
39      // call hit request callback when LLC latency is met
40      it = m_hit_list.begin();
41      while (it != m_hit_list.end()) {
42          if (m_clk >= it->first) {
43              std::vector<Request> _req_v = it->second;
44              m_receive_requests[it->second.addr] = _req_v;
45
46              it->second.callback(it->second);
47              it = m_hit_list.erase(it);
48          }
49          else {
50              it++;
51          }
52      }
53  };
```

```
64      if (auto line_it = check_set_hit(set, req.addr); line_it != set.end()) {
65          // Hit in the set
66          DEBUG_LOG(DSIMPLEO3LLC, m_logger,
67          "[Clk={}] Request Source: {}, Type: {}, Addr: {}, Index: {}, Tag: {}. Hit, will finish at Clk={}",
68          m_clk, req.source_id, req.type_id, req.addr, get_index(req.addr), get_tag(req.addr), m_clk, m_clk + m_latency
69          );
70
71          // Update the LRU status
72          set.push_back({req.addr, get_tag(req.addr), line_it->dirty || (req.type_id == Request::Type::Write), true});
73          set.erase(line_it);
74
75          // Add to the hit list to callback when finished
76          m_hit_list.push_back(std::make_pair(m_clk + m_latency, req));
77          return true;
78      } else {
```

```
void receive(Request& req) {
    m_llc->receive(req);

    // TODO: LLC latency for the core to receive the request?
    for (auto r : m_llc->m_receive_requests[req.addr]) {
        r.arrive = req.arrive;
        r.depart = req.depart;
        m_cores[r.source_id]->receive(r);
    }
    m_llc->m_receive_requests[req.addr].clear();
};
```

```
56      // Create the cores
57      for (int id = 0; id < m_num_cores; id++) {
58          SimpleO3Core* core = new SimpleO3Core(id, ipc, depth, m_num_expected_insts, trace_list
59          core->m_callback = [this](Request& req){return this->receive(req);} ;
60          m_cores.push_back(core);
61      }
```

- **Hit List에서 지연 끝난 Request들을 m_receive_requests배열에 저장**
  - 여러 Core가 같은 주소를 가지는 Requests를 기다리고 있을 때, 완료되면 모든 Core에게 통지할 수 있도록 Mapping 하는 것
  - ex. m_receive_requests[0x1000] = [Request(addr=0x1000, source_id=0)]
- **callback이 호출됨 (이 callback은 SimpleO3::receive를 가리킴)**
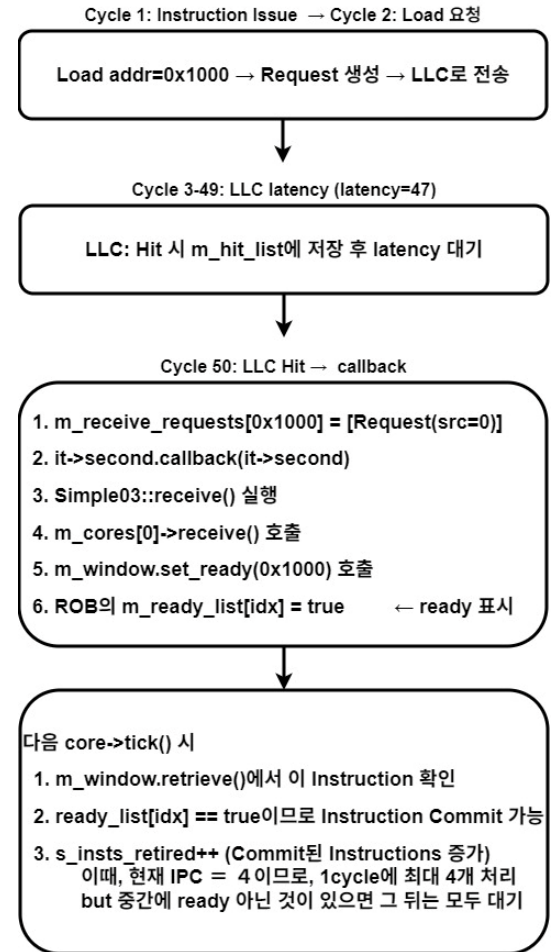
❑ simpleO3 Core Operations

- class SimpleO3 – tick() → m_llc->tick() → **callback**

```
56        // Create the cores
57        for (int id = 0; id < m_num_cores; id++) {
58          SimpleO3Core* core = new SimpleO3Core(id, ipc, depth, m_num_expected_insts, trace_list
59          core->m_callback = [this](Request& req){return this->receive(req);} ;
60          m_cores.push_back(core);
61        }
```

```
void receive(Request& req) {
  m_llc->receive(req);      → cache 내 MSLR 처리

  // TODO: LLC latency for the core to receive the request?
  for (auto r : m_llc->m_receive_requests[req.addr]) {
    r.arrive = req.arrive;
    r.depart = req.depart;
    m_cores[r.source_id]->receive(r);
  }
  m_llc->m_receive_requests[req.addr].clear();
}
```

```
184  void SimpleO3Core::receive(Request& req) {
185    m_window.set_ready(req.addr);
186
187    if (req.arrive != -1 && req.depart > m_last_mem_cycle) {
188      if (!reached_expected_num_insts) {
189        s_mem_access_cycles += (req.depart - std::max(m_last_mem_cycle, req.arrive));
190        m_last_mem_cycle = req.depart;
191      }
192    }
193  }
```

```
90  ∨ void SimpleO3Core::InstWindow::set_ready(Addr_t addr) {
91        if (m_load == 0) return;
92
93        int index = m_tail_idx;
94  ∨     for (int i = 0; i < m_load; i++) {
95  ∨       if (m_addr_list[index] == addr) {
96            m_ready_list[index] = true;
97          }
98          index++;
99  ∨       if (index == m_depth) {
100           index = 0;
101         }
102       }
103     }
```

Cycle 1: Instruction Issue → Cycle 2: Load 요청

Load addr=0x1000 → Request 생성 → LLC로 전송

Cycle 3-49: LLC latency (latency=47)

LLC: Hit 시 m_hit_list에 저장 후 latency 대기

Cycle 50: LLC Hit → callback

1. m_receive_requests[0x1000] = [Request(src=0)]
2. it->second.callback(it->second)
3. SimpleO3::receive() 실행
4. m_cores[0]->receive() 호출
5. m_window.set_ready(0x1000) 호출
6. ROB의 m_ready_list[idx] = true   ← ready 표시

다음 core->tick() 시

1. m_window.retrieve()에서 이 Instruction 확인
2. ready_list[idx] == true이므로 Instruction Commit 가능
3. s_insts_retired++ (Commit된 Instructions 증가)
   이때, 현재 IPC = 4 이므로, 1cycle에 최대 4개 처리
   but 중간에 ready 아닌 것이 있으면 그 뒤는 모두 대기

- **Out of order Processor는 Instruction의 Dependency를 고려하여 Instruction Re-Ordering진행**
- **ROB에 쌓인 이미 완료된 Instruction은 이제 sequential하게 기존 Instruction 순서에 맞게 commit됨**
- **항상 Processor는 User입장에서 Sequential하게 보여야 함** - The MicroArchitecture of SuperScalar Processors(1995)

❑ **simpleO3 Core Operations**

- **class SimpleO3 – tick() → core->tick()**

- abcd
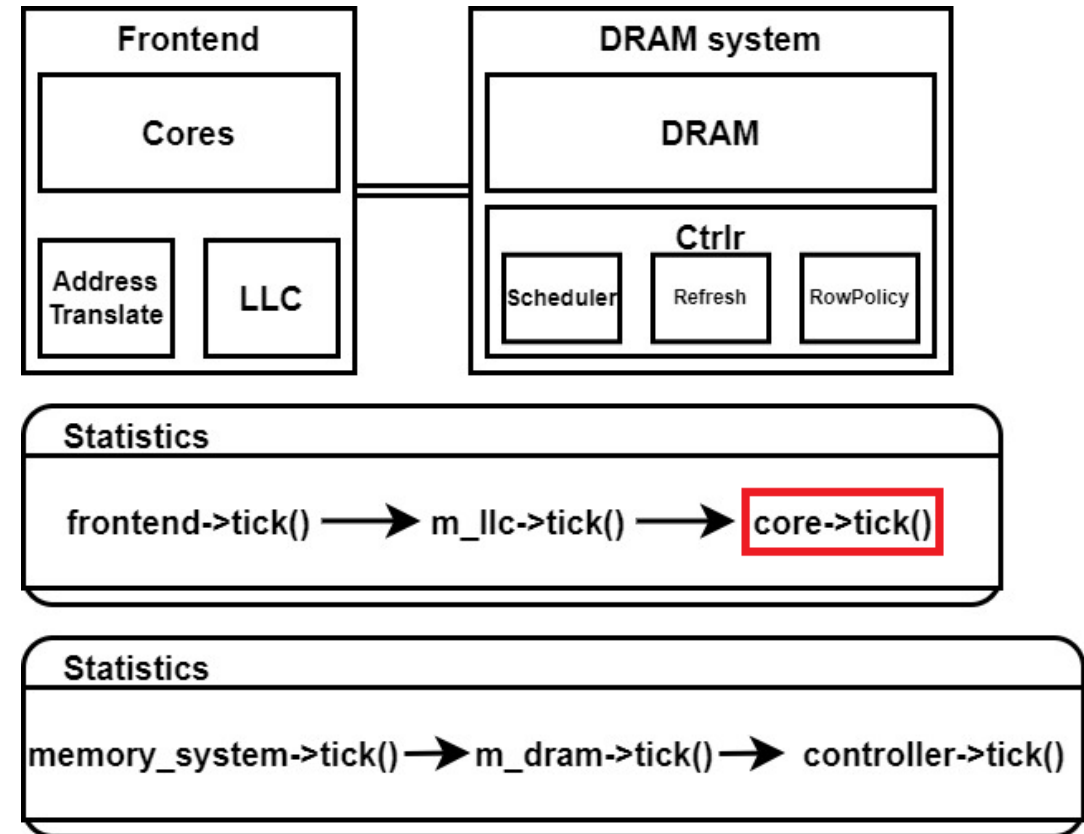
❑ **memory_system**

- **abcd**

❑ **memory_system**

- **abcd**

❏ **addr_mapper**

- **abcd**

❑ **addr_mapper**

- **abcd**

src/translation folder

❏ **translation**

- **abcd**

src/translation folder

❏ **translation**

  • **abcd**

src/translation folder

❏ **translation**

- **abcd**

src/dram folder

❑ **dram**

- **abcd**

src/dram folder

❑ **dram**

• **abcd**

❑ **dram**

- **abcd**

src/dram_controller folder

❑ DRAM Controller Operations

- **class GenericDRAMController – init()**

(src\dram_controller\impl\ generic_dram_controller.cpp)

```cpp
void init() override {
    m_wr_low_watermark =  param<float>("wr_low_watermark").desc("Threshold for switching back to read mode.").default_val(0.2f)
    m_wr_high_watermark = param<float>("wr_high_watermark").desc("Threshold for switching to write mode.").default_val(0.8f);

    m_scheduler = create_child_ifce<IScheduler>();
    m_refresh = create_child_ifce<IRefreshManager>();
    m_rowpolicy = create_child_ifce<IRowPolicy>();

    if (m_config["plugins"]) {
        YAML::Node plugin_configs = m_config["plugins"];
        for (YAML::iterator it = plugin_configs.begin(); it != plugin_configs.end(); ++it) {
            m_plugins.push_back(create_child_ifce<IControllerPlugin>(*it));
        }
    }
};
```

1.1. YAML file을 통해 write buffer priority threshold를 초기화. → Scheduler가 사용할 buffer를 결정

1.2. Scheduler, Refresh Manager, Row Policy를 생성.

1.3. Plugin을 load하여 Controller를 확장

```cpp
// 2.2.1    If no request to be scheduled in the priority buffer, check the read and write buffers.
if (!request_found) {
    // Query the write policy to decide which buffer to serve
    set_write_mode();
    auto& buffer = m_is_write_mode ? m_write_buffer : m_read_buffer;
    if (req_it = m_scheduler->get_best_request(buffer); req_it != buffer.end()) {
        request_found = m_dram->check_ready(req_it->command, req_it->addr_vec);
        req_buffer = &buffer;
    }
}
```

❑ DRAM Controller Operations

- **class GenericDRAMController – setup()**

```
 70 ∨     void setup(IFrontEnd* frontend, IMemorySystem* memory_system) override {
 71         m_dram = memory_system->get_ifce<IDRAM>();
 72         m_bank_addr_idx = m_dram->m_levels("bank");
 73         m_priority_buffer.max_size = 512*3 + 32;
 74
 75         m_num_cores = frontend->get_num_cores();
 76
 77         s_read_row_hits_per_core.resize(m_num_cores, 0);
 78         s_read_row_misses_per_core.resize(m_num_cores, 0);
 79         s_read_row_conflicts_per_core.resize(m_num_cores, 0);
 80
 81         register_stat(s_row_hits).name("row_hits_{}", m_channel_id);
 82         register_stat(s_row_misses).name("row_misses_{}", m_channel_id);
 83         register_stat(s_row_conflicts).name("row_conflicts_{}", m_channel_id);
 84         register_stat(s_read_row_hits).name("read_row_hits_{}", m_channel_id);
 85         register_stat(s_read_row_misses).name("read_row_misses_{}", m_channel_id);
 86         register_stat(s_read_row_conflicts).name("read_row_conflicts_{}", m_channel_id);
 87         register_stat(s_write_row_hits).name("write_row_hits_{}", m_channel_id);
 88         register_stat(s_write_row_misses).name("write_row_misses_{}", m_channel_id);
 89         register_stat(s_write_row_conflicts).name("write_row_conflicts_{}", m_channel_id);
 90
 91 ∨       for (size_t core_id = 0; core_id < m_num_cores; core_id++) {
 92           register_stat(s_read_row_hits_per_core[core_id]).name("read_row_hits_core_{}", core_id);
 93           register_stat(s_read_row_misses_per_core[core_id]).name("read_row_misses_core_{}", core_id);
 94           register_stat(s_read_row_conflicts_per_core[core_id]).name("read_row_conflicts_core_{}", core_id);
 95         }

 96
 97         register_stat(s_num_read_reqs).name("num_read_reqs_{}", m_channel_id);
 98         register_stat(s_num_write_reqs).name("num_write_reqs_{}", m_channel_id);
 99         register_stat(s_num_other_reqs).name("num_other_reqs_{}", m_channel_id);
100         register_stat(s_queue_len).name("queue_len_{}", m_channel_id);
101         register_stat(s_read_queue_len).name("read_queue_len_{}", m_channel_id);
102         register_stat(s_write_queue_len).name("write_queue_len_{}", m_channel_id);
103         register_stat(s_priority_queue_len).name("priority_queue_len_{}", m_channel_id);
104         register_stat(s_queue_len_avg).name("queue_len_avg_{}", m_channel_id);
105         register_stat(s_read_queue_len_avg).name("read_queue_len_avg_{}", m_channel_id);
106         register_stat(s_write_queue_len_avg).name("write_queue_len_avg_{}", m_channel_id);
107         register_stat(s_priority_queue_len_avg).name("priority_queue_len_avg_{}", m_channel_id);
108
109         register_stat(s_read_latency).name("read_latency_{}", m_channel_id);
110         register_stat(s_avg_read_latency).name("avg_read_latency_{}", m_channel_id);
111       };
```

2.1. DRAM Interface 연결 - (ex. dram.h → DDR4)
2.2. Bank level의 index를 조회
- DRAM Address Vector → ex. addr_vec[0] = channel, [1] = rank, [2] = bankgroup, [3] = bank, [4] = row, [5] = column
- 이 경우, Bank level index = 3
- 아래는 Row Hit 판별을 위해 Bank level index가 어떻게 사용되는지를 나타냄
- m_bank_addr_idx = 3 → bank = addr_vec[3] = 2 → Row Hit Check: open_rows[bank=2]의 row=1024 check → hit?
2.3. priority buffer Size를 하드 코딩

❏ DRAM Controller Operations

- **class GenericDRAMController – setup()**

2.4. Frontend에서 core 수를 조회하여 m_num_cores에 저장

2.5. Read Row Hit Per Core, … , Read Row Conflict Per Core 등의 배열 초기화 → multi core 지원을 위해 초기화

→ 위 배열들은 통계를 수집하고 출력 용도로만 사용됨

2.6. 다양한 통계 variable를 register하여 DRAM controller의 동작을 분석 → 통계를 수집하고 출력

❏ DRAM Controller Operations

- **class GenericDRAMController – send()**

```cpp
bool send(Request& req) override {
    req.final_command = m_dram->m_request_translations(req.type_id);

    switch (req.type_id) {
        case Request::Type::Read: {
            s_num_read_reqs++;
            break;
        }
        case Request::Type::Write: {
            s_num_write_reqs++;
            break;
        }
        default: {
            s_num_other_reqs++;
            break;
        }
    }

    // Forward existing write requests to incoming read requests
    if (req.type_id == Request::Type::Read) {
        auto compare_addr = [req](const Request& wreq) {
            return wreq.addr == req.addr;
        };
        if (std::find_if(m_write_buffer.begin(), m_write_buffer.end(), compare_addr) != m_write_buffer.end()) {
            // The request will depart at the next cycle
            req.depart = m_clk + 1;
            pending.push_back(req);
            return true;
        }
    }

    // Else, enqueue them to corresponding buffer based on request type id
    bool is_success = false;
    req.arrive = m_clk;
    if      (req.type_id == Request::Type::Read) {
        is_success = m_read_buffer.enqueue(req);
    } else if (req.type_id == Request::Type::Write) {
        is_success = m_write_buffer.enqueue(req);
    } else {
        throw std::runtime_error("Invalid request type!");
    }
    if (!is_success) {
        // We could not enqueue the request
        req.arrive = -1;
        return false;
    }

    return true;
};
```

❑ **dram_controller**

- **abcd**

src/base folder

❑ **base**

- **abcd**