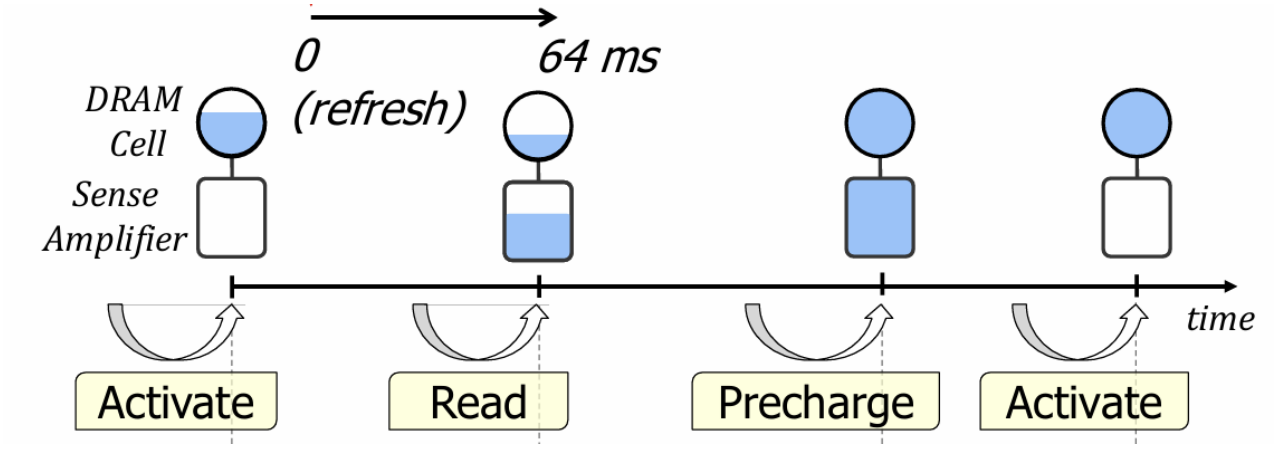
Ramulator 2.0 Summary

Intelligent System

Laboratory

❑ DRAM Operations & States

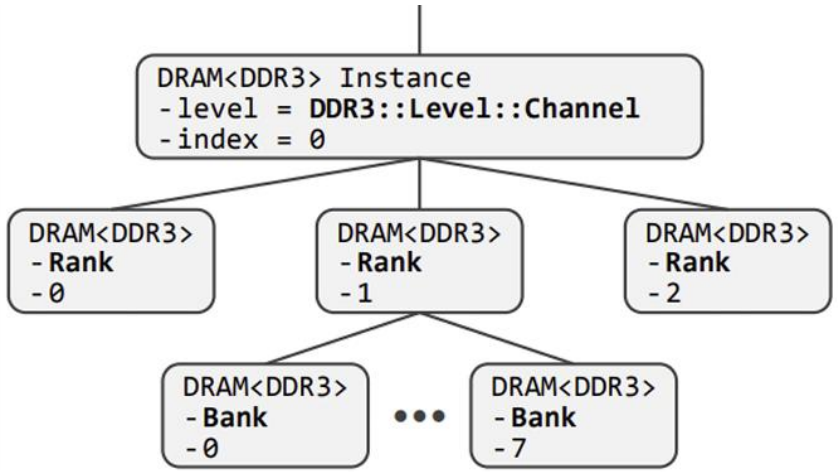


- **Main DRAM states**
  - **Activate**
  - **Read**
  - **Precharge**

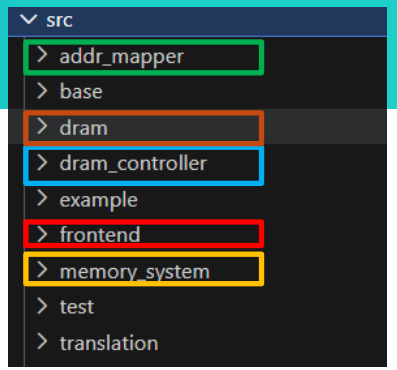
```cpp
// DRAM.h
template <typename T>
class DRAM {
  DRAM<T>* parent;
  vector<DRAM<T>*>
    children;
  T::Level level;
  int index;
  // more code...
};
```

```cpp
// DDR3.h/cpp
class DDR3 {
  enum class Level {
    Channel, Rank, Bank,
    Row, Column, MAX
  };


  // more code...
};
```



src files <=> DRAM Operation



❑ Simulation Configuration

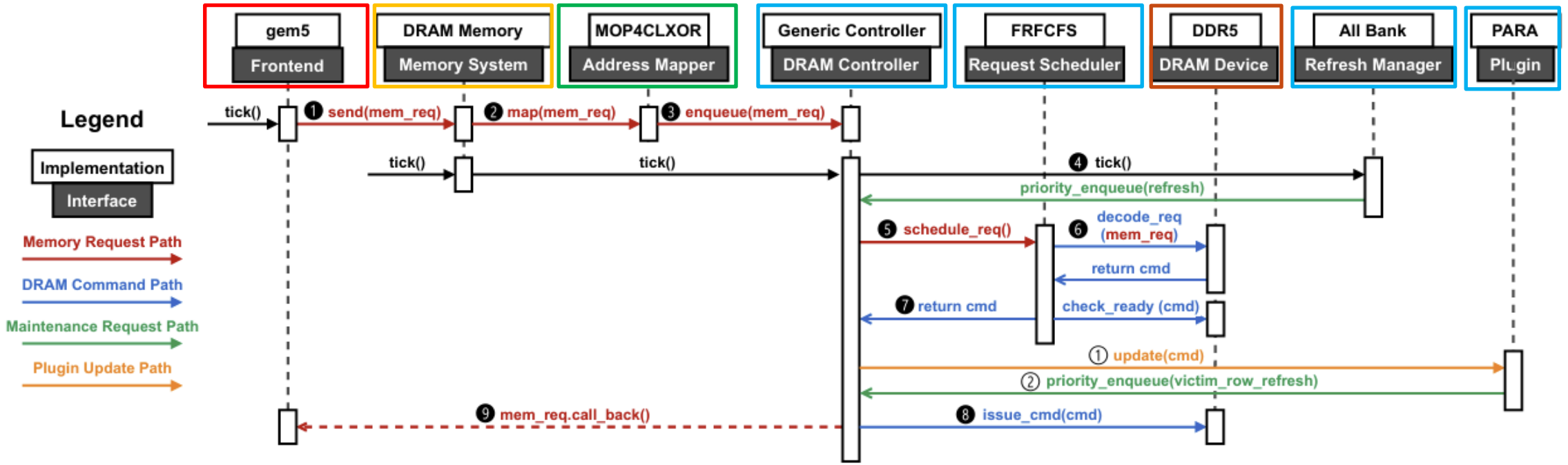


Fig. 1: High-level software architecture of Ramulator 2.0 using an example DDR5 system configuration

src files <=> DRAM Operation

❑ Simulation Configuration

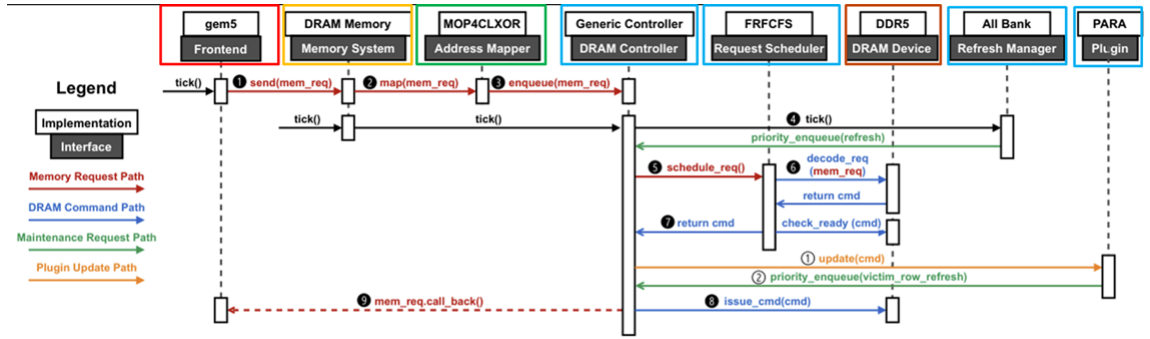
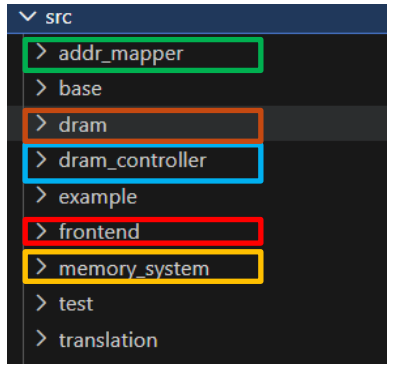


Fig. 1: High-level software architecture of Ramulator 2.0 using an example DDR5 system configuration



1. **Requests are sent**: Front-end(trace file)에서 메모리 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 function

❑ main.cpp

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

•
•
•

```cpp
  // Connect the frontend and the memory system together,
  // this recursively calls the "setup" function in all instaniated components
  // so that they can get each other's parameters (if needed) after their initialization
  frontend->connect_memory_system(memory_system);
  memory_system->connect_frontend(frontend);

  // Get the relative clock ratio between the frontend and memory system
  int frontend_tick = frontend->get_clock_ratio();
  int mem_tick = memory_system->get_clock_ratio();

  int tick_mult = frontend_tick * mem_tick;

  for (uint64_t i = 0;; i++) {
    if (((i % tick_mult) % mem_tick) == 0) {
      frontend->tick();
    }

    if (frontend->is_finished()) {
      break;
    }

    if ((i % tick_mult) % frontend_tick == 0) {
      memory_system->tick();
    }
  }

  // Finalize the simulation. Recursively print all statistics from all components
  frontend->finalize();
  memory_system->finalize();

  return 0;
}
```

main.cpp

1. **Argument 받는 부분**
   - **Options**
     1. -c: commandline 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

yaml file

❑ example_config.yaml

```yaml
Frontend:
  impl: SimpleO3
  clock_ratio: 8
  num_expected_insts: 500000
  traces:
    - example_inst.trace

  Translation:
    impl: RandomTranslation
    max_addr: 2147483648

MemorySystem:
  impl: GenericDRAM
  clock_ratio: 3

  DRAM:
    impl: DDR4
    org:
      preset: DDR4_8Gb_x8
      channel: 1
      rank: 2
    timing:
      preset: DDR4_2400R

  Controller:
    impl: Generic
    Scheduler:
      impl: FRFCFS
    RefreshManager:
      impl: AllBank
    RowPolicy:
      impl: ClosedRowPolicy
      cap: 4
    plugins:

  AddrMapper:
    impl: RoBaRaCoCh
```

yaml file

1. **Frontend 부분**
2. **MemorySystem 부분**

src/base folder

- base
  - abcd

src/base folder

- ❑ base
  - abcd

src/base folder

- base
  - abcd

src/frontend folder

- ❑ frontend
  - abcd

src/frontend folder

- ❑ frontend
  - abcd

src/frontend folder

- frontend

  - abcd

src/memory_system folder

- memory_system
  - abcd

src/memory_system folder

- memory_system
  - abcd

src/addr_mapper folder

- addr_mapper
  - abcd

src/addr_mapper folder

- 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

src/dram folder

- dram
  - abcd

src/dram_controller folder

- dram_controller

  - abcd

src/dram_controller folder

- dram_controller

  - abcd

src/dram_controller folder

- dram_controller

  - abcd

src/dram_controller folder

- dram_controller

  - abcd