

Plan for the week

Monday

Introduction

- Getting started with gem5: using, develop, and simulation

Using gem5

- gem5 standard library

Tuesday

Using gem5

- General using
- gem5 models: caches, CPUs, memory

- Full system sim
- Accelerating simulation

Wednesday

gem5 devel

- First SimObject, params, events, memory ops

- Instruction execution
- Adding an instruction

Thursday

gem5 devel

- Classic caches
- Ruby and SLICC
- OCN and Garnet

- gem5's GPGPU

Friday

Extra topics

- Contributing to gem5

- Using other simulators w/ gem5

- **Whatever you want!**





Running Things on gem5

A presentation by
Maryam Babaie



OOO Action Item 😊

Launch codespace and run the following commands:

```
cd gem5
```

```
scons build/X86/gem5.debug -j14
```

Table of Contents

1. Intro. to Syscall Emulation Mode
2. The m5 Utility
 - i. Examples on m5 Utility
 - ii. SE mode uses hosts for most things
3. Cross-compiling
4. Traffic Generator

Intro. to Syscall Emulation Mode



Previously on gem5: how to build & use

Building with Scons:

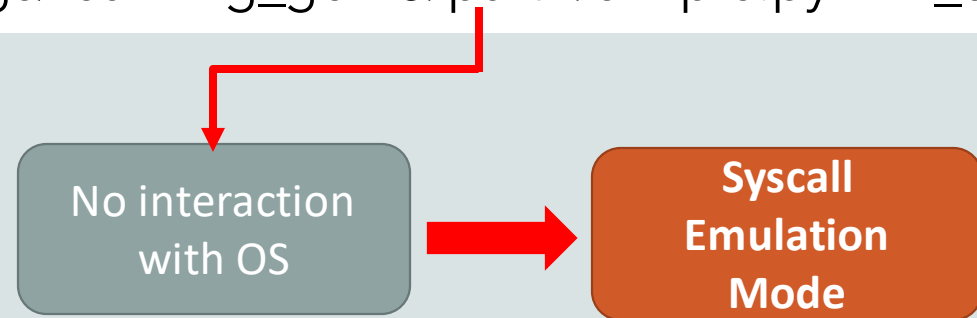
```
scons build/{ISA}/gem5.{variant}-j {cpus}
```

Once compiled, gem5 can then be run using:

```
build/{ISA}/gem5.{variant}[gem5 options] {simulation script}[script options]
```

Example:

```
build/X86/gem5.fast --outdir=simple_out configs/learning_gem5/part1/simple.py --l1i_size=32kB
```

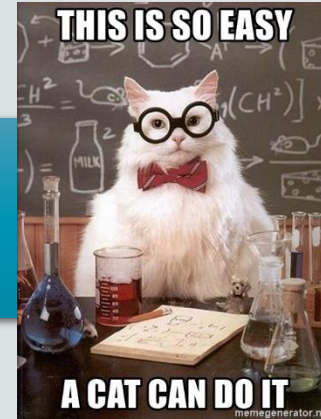


What is Syscall Emulation?

Syscall Emulation (SE) mode does not model all the devices in a system.

It focuses on simulating the **CPU** and **memory** system.

SE mode is much easier to configure.



However, SE only emulates Linux system calls, and only models user-mode code.

When to use/avoid Syscall Emulation?

If you do not need to model the OS, and you want extra performance, then you should use **SE mode**.

**HEADS
UP!**

However, if you need high fidelity modeling of the system, or if OS interactions like page table walks are important, then you should use **FS mode**.

The m5 Utility



The m5 Utility API

“*m5ops*” are the special opcodes that can be used in m5 to issue special instructions.

- Usage: checkpointing, exiting simulation, etc.

The *m5 utility* is the API providing these functionalities/options.

Options include:

- **exit (delay)**: Stop the simulation in delay nanoseconds.
- **resetstats (delay, period)**: Reset simulation statistics in delay nanoseconds; repeat this every period nanoseconds.
- **dumpstats (delay , period)**: Save simulation statistics to a file in delay nanoseconds; repeat this every period nanoseconds.
- **dumpresetstats (delay ,period)**: same as dumpstats; resetstats;
- Full list of options can be found [here](#).



How to use the m5 utility?

It is best to insert the option(s) directly in the source code of the application.

m5ops.h header file has prototypes for all the functionalities/options must be included.

The application should be linked with the appropriate ***m5 & libm5.a*** files.



Building m5 and libm5

The m5 utility is in “gem5/util/m5/” directory.

To build **m5** and **libm5.a**, run the following command in the gem5/util/m5/ directory.

```
scons build/{TARGET_ISA}/out/m5
```

Target ISA must be in **lower case**:

- x86
- arm
- thumb
- sparc
- arm64
- Riscv

This will generate **libm5.a** and **m5** binaries in the *util/m5/build/{TARGET_ISA}/out/* directory.



Building m5 and libm5

Note: if you are using a x86 system for other ISAs, you need to have the *cross-compiler*

Cross-compiler for each target ISA:

- arm : **arm-linux-gnueabihf-gcc**
- thumb : **arm-linux-gnueabihf-gcc**
- sparc : **sparc64-linux-gnu-gcc**
- arm64 : **aarch64-linux-gnu-gcc**
- riscv : **riscv64-linux-gnu-gcc**

See [util/m5/README.md](#) for more details.



Linking m5 to C/C++ code

After building the m5 and libm5.a as described, link them to your code:

1. Include **gem5/m5ops.h** in your source file(s).
2. Add **gem5/include** to your compiler's include search path.
3. Add **gem5/util/m5/build/{TARGET_ISA}/out** to the linker search path.
4. Link against **libm5.a**.



Example 1: print in std out

```
materials > using-gem5 > 03-running > example1 > se_example.cpp
1  #include <unistd.h>
2  // #include "gem5/m5ops.h"
3
4  int main()
5  {
6      // m5_reset_stats(0, 0);
7
8      write(1, "This will be output to standard out\n", 36);
9
10     // m5_exit(0);
11
12     return 0;
13 }
```

Example1 code:

materials/using-gem5/03-running/example1/se_example.cpp

Config file:

materials/using-gem5/03-running/simple.py

Commands

Compile the code: **gcc materials/using-gem5/03-running/example1/se_example.cpp -o exampleBin**

Run workload: **./exampleBin**

Run gem5: **gem5-x86 materials/using-gem5/03-running/simple.py**



Example 1

Include `gem5/m5ops.h`

Adding m5 util option

Adding m5 util option

```
materials > using-gem5 > 03-running > example1 > se_example.cpp > ...
1  #include <unistd.h>
2  #include "gem5/m5ops.h"
3
4  int main()
5  {
6      m5_reset_stats(0, 0);
7
8      write(1, "This will be output to standard out\n", 36);
9
10     m5_exit(0);
11
12     return 0;
13 }
```



Example 1: building x86 m5 utility

```
cd gem5/util/m5
```

```
scons build/x86/out/m5
```



Example 1

Add **gem5/include** to your compiler's include search path.

```
gcc materials/using-gem5/03-running/example1/se_example.cpp -o exampleBin  
-I gem5/include/  
-lm5 Link against libm5.a.  
-Lgem5/util/m5/build/x86/out
```

Add **gem5/util/m5/build/{TARGET_ISA}/out** to the linker search path.

Note: if you try to locally run the output binary in your host, it will generate error:

```
Illegal instruction (core dumped)
```

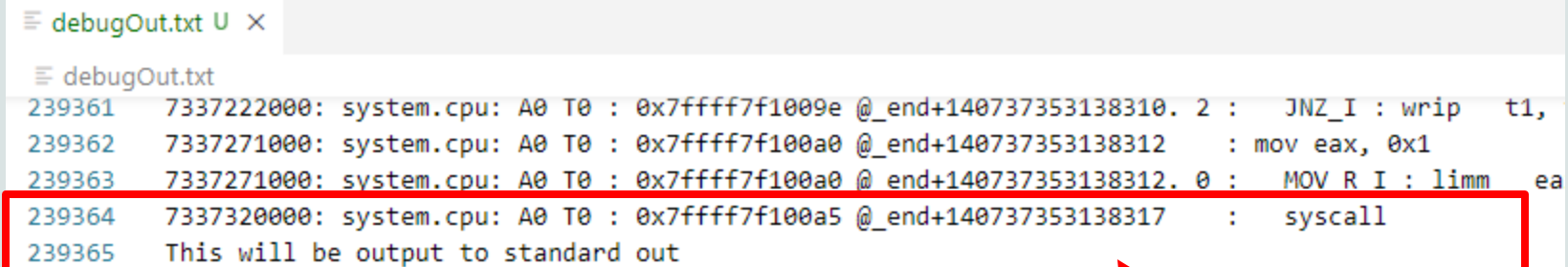


SE mode uses the host for many things.

SE mode treats a system call as one instruction for the **guest**.

```
1 #include <unistd.h>
2 #include "gem5/m5ops.h"
3
4 int main()
5 {
6     m5_reset_stats(0, 0);
7     write(1, "This will be output to standard out\n", 36);
8     m5_exit(0);
9     return 0;
10 }
11
12
13
```

```
debugOut.txt U X
debugOut.txt
239361 7337222000: system.cpu: A0 T0 : 0x7ffff7f1009e @_end+140737353138310. 2 : JNZ_I : wrrip t1,
239362 7337271000: system.cpu: A0 T0 : 0x7ffff7f100a0 @_end+140737353138312 : mov eax, 0x1
239363 7337271000: system.cpu: A0 T0 : 0x7ffff7f100a0 @_end+140737353138312. 0 : MOV R I : limm ea
239364 7337320000: system.cpu: A0 T0 : 0x7ffff7f100a5 @_end+140737353138317 : syscall
239365 This will be output to standard out
```



Run gem5:

```
gem5/build/X86/gem5.debug --debug-flags=ExecAll materials/using-gem5/03-running/simple.py > debugOut.txt
```

Example 2: checking a directory

```
materials > using-gem5 > 03-running > example2 > dir_example.cpp > ...
1  #include<iostream>
2  #include<dirent.h>
3
4  using namespace std;
5
6  int main()
7  {
8      struct dirent *d;
9      DIR *dr;
10     dr = opendir("/workspaces/gem5-bootcamp-env/materials/using-gem5/03-running");
11     if (dr!=NULL) {
12         std::cout<<"List of Files & Folders:\n";
13         for (d=readdir(dr); d!=NULL; d=readdir(dr)) {
14             std::cout<<d->d_name<< " ";
15         }
16         closedir(dr);
17     }
18     else {
19         std::cout<<"\nError Occurred!";
20     }
21     std::cout<<endl;
22     return 0;
23 }
```

Example2 code:

materials/using-gem5/03-running/example2/dir_example.cpp

Config file:

materials/using-gem5/03-running/simple.py

Commands

Compile the code:

```
g++ materials/using-gem5/03-running/example2/dir_example.cpp -o exampleBin
```

Run gem5:

```
gem5-x86 materials/using-gem5/03-running/simple.py
```



SE mode uses the host for many things.

For things like creating/reading a file, it will create/read files on the **host**.

```
materials > using-gem5 > 03-running > example2 > dir_example.cpp > ...
1  #include<iostream>
2  #include<dirent.h>
3
4  using namespace std;
5
6  int main()
7  {
8      struct dirent *d;
9      DIR *dr;
10     dr = opendir("/workspaces/gem5-bootcamp-env/materials/using-gem5/03-running");
11     if (dr!=NULL) {
12         std::cout<<"List of Files & Folders:\n";
13         for (d=readdir(dr); d!=NULL; d=readdir(dr)) {
14             std::cout<<d->d_name<< " , ";
15         }
16         closedir(dr);
17     }
18     else {
19         std::cout<<"\nError Occurred!";
20     }
21     std::cout<<endl;
22     return 0;
23 }
```

```
gem5: connecting to workspace root@root: port 7000
command line: gem5/build/X86/gem5.debug materials/using-gem5/03-running/simple.py

Global frequency set at 1000000000000 ticks per second
build/X86/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not m
0: system.remote_gdb: listening for remote gdb on port 7000
Beginning simulation!
build/X86/sim/simulate.cc:194: info: Entering event queue @ 0. Starting simulation...
build/X86/sim/mem_state.cc:443: info: Increasing stack size by one page.
build/X86/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
build/X86/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
build/X86/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
build/X86/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
build/X86/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
build/X86/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
build/X86/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
List of Files & Folders:
simple.py, example3, example1, ., example2, ..,
Exiting @ tick 180191021000 because exiting with last active thread context
root@codespaces-45cda2:/workspaces/gem5-bootcamp-env#
```

SE mode deos NOT implement many things!

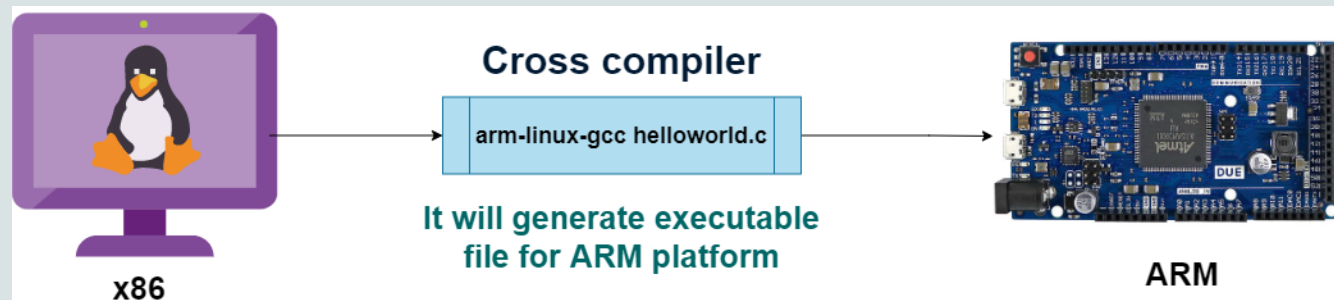
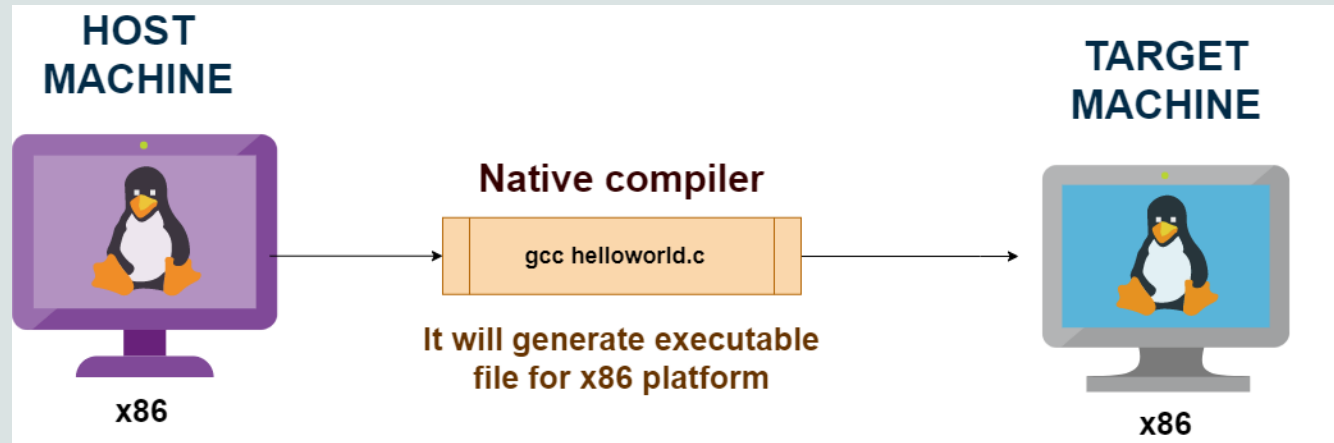
- Filesystem
- Most of systemcalls
- I/O devices
- Interrupts
- TLB misses → Page table walks
- Context switches
- multiple threads

You may have a multithreaded execution, but there's no context switches & no spin locks

Cross-compiling



Cross-compiling from one ISA to another.



Example: Cross-compiling

Host = X86 → Target: ARM64

(1) Build m5 utility for arm64

```
cd gem5/util/m5
```

```
scons arm64.CROSS_COMPILE=aarch64-linux- build/arm64/out/m5
```

(2) Cross-compile the program with m5 utility

```
aarch64-linux-g++ materials/using-gem5/03-running/example1/se_example.cpp -o exampleBin  
-I gem5/include/ -lm5 -Lgem5/util/m5/build/arm64/out -static
```

(3) Run gem5

```
gem5-arm materials/using-gem5/03-running/simple.py
```

```
materials > using-gem5 > 03-running > example1 > se_example.cpp > ...  
1  #include <unistd.h>  
2  #include "gem5/m5ops.h"  
3  
4  int main()  
5  {  
6      m5_reset_stats(0, 0);  
7  
8      write(1, "This will be output to standard out\n", 36);  
9  
10     m5_exit(0);  
11  
12     return 0;  
13 }
```



Example: Cross-compiling (Dynamic)

(1) Build m5 utility for ARM, as shown before.

(2) Cross-compile the program with m5 utility

```
materials > using-gem5 > 03-running > example1 > se_example.cpp > ...
1  #include <unistd.h>
2  #include "gem5/m5ops.h"
3
4  int main()
5  {
6      m5_reset_stats(0, 0);
7
8      write(1, "This will be output to standard out\n", 36);
9
10     m5_exit(0);
11
12     return 0;
13 }
```

```
aarch64-linux-g++ materials/using-gem5/03-running/example1/se_example.cpp -o exampleBin  
-I gem5/include/ -lm5 -Lgem5/util/m5/build/arm64/out
```

Also, you need to let gem5 know where the libraries associated with the guest ISA are located, using “**redirect**”.

Example: Cross-compiling (Dynamic)

You should modify the config file (*simple.py*) as follows:

```
from m5.core import setInterpDir
```

```
binary = "/workspaces/gem5-bootcamp-env/exampleBin"
```

```
setInterpDir("/usr/aarch64-linux-genu/")
```

```
system.redirect_paths = [RedirectPath(app_path="/lib", host_paths=["/usr/aarch64-linux-genu/lib"])]
```

(3) Run gem5

```
gem5-arm materials/using-gem5/03-running/simple.py
```



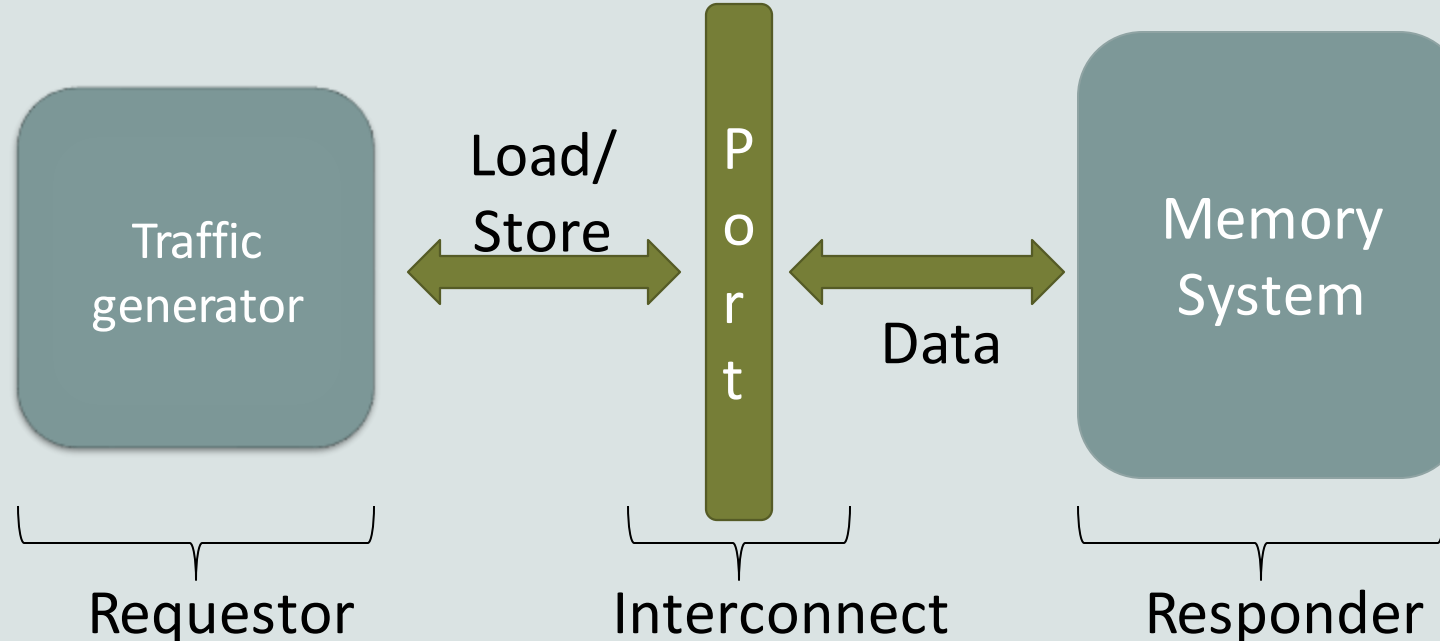
Traffic Generator in gem5



Traffic Generator

A traffic generator module generates stimuli for the memory system.

Used for creating test cases for caches, interconnects, and memory controllers, etc.

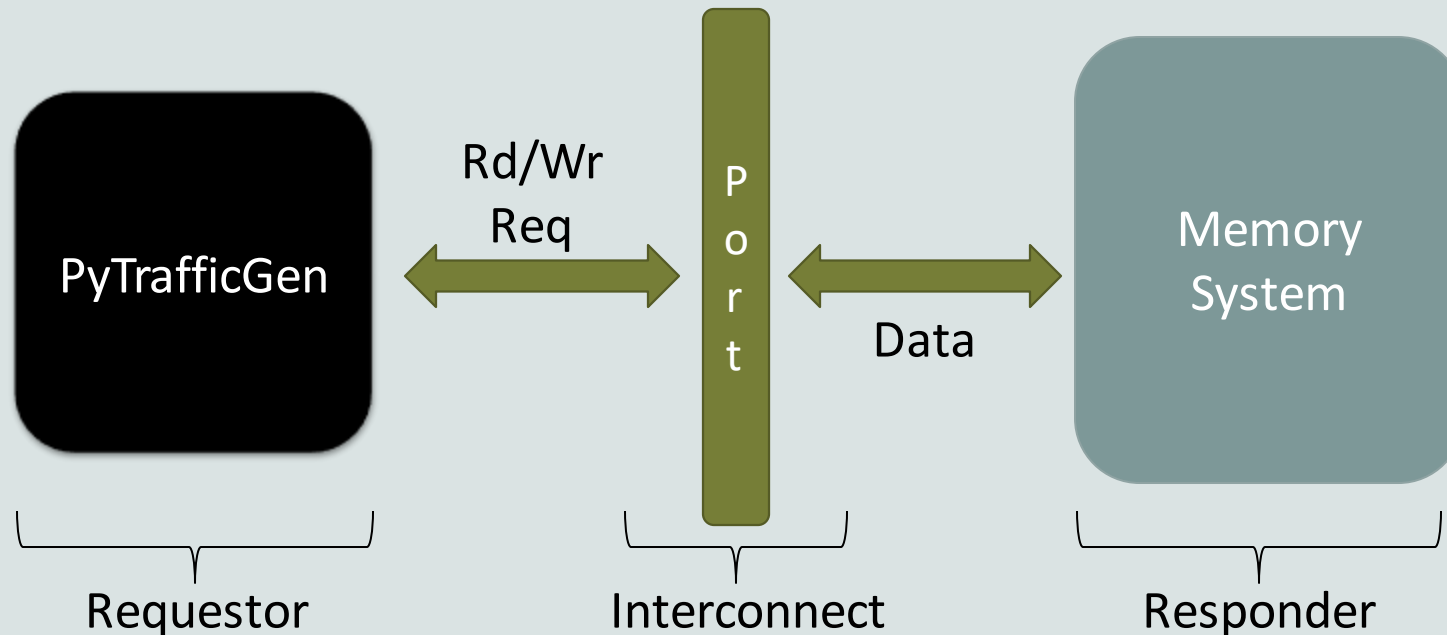


gem5's Traffic Gen: *PyTrafficGen*

PyTrafficGen is a traffic generator module (SimObject) located in:

`"gem5/src/cpu/testers/traffic_gen"`

Used as a black box replacement for any generator of read/write requestor.

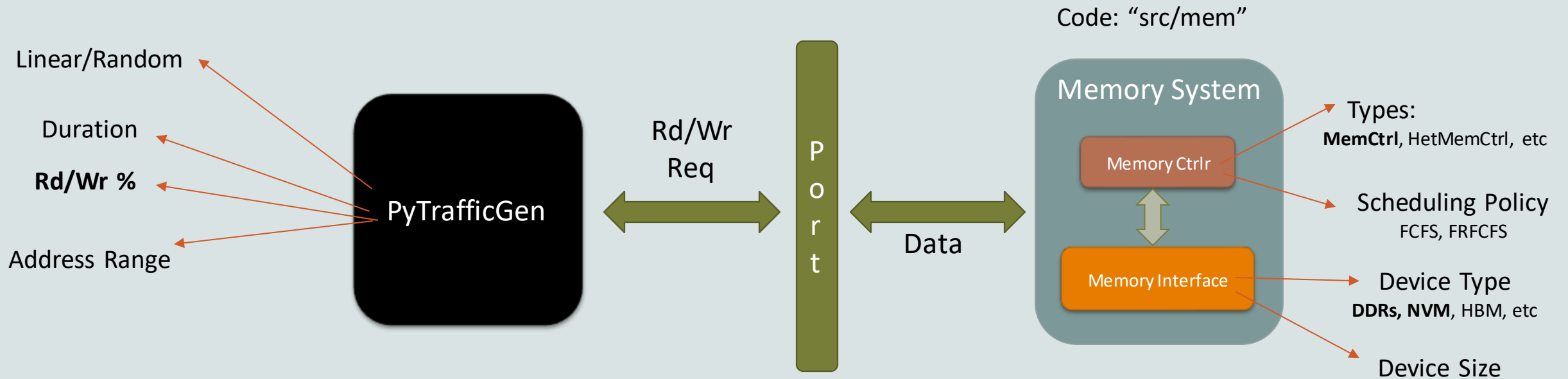


PyTrafficGen: Params

PyTrafficGen's parameters allow you to control the characteristics of the generated traffic.

Parameter	Definition
pattern	The pattern of generated addresses: linear/ random
duration	The duration of generating requests in ticks (quantum of time in gem5).
start address	The lower bound for addresses that the synthetic traffic will access.
end address	The upper bound for addresses that the synthetic traffic will access.
minimum period	The minimum timing difference between two consecutive requests in ticks.
maximum period	The maximum timing difference between two consecutive requests in ticks.
request size	The number of bytes that are read/written by each request.
read percentage	The percentage of reads among all the requests, the rest of requests are write requests.

Example3: *PyTrafficGen*



Command to run tests for this examples:

```
./materials/using-gem5/03-running/example3/traffGen_run.sh
```


Summary

- SE mode is easy to configure and fast for development purposes, if OS is not involved.
- m5 utility API is a useful tool for simulation behavior and performance analysis.
- Cross compilers should be used if the host and guest ISAs are different.
- Traffic generator can abstract away the details of a data requestor such as CPU for generating test cases for memory systems.