



The gem5 Tutorial @HPCA 2024

Presentation by Bobby R. Bruce

Materials designed for gem5 v23.1

A little about me



Started in Sep. 2019

Started as a Postdoc

In Jan 2021 I moved up to being Project Scientist

Spent almost all that time working on the gem5 project.

*So, am I an expert in
gem5?*



What is gem5?

The gem5 architecture simulator provides a platform for evaluating computer systems by modeling the behavior of the underlying hardware. It enables researchers to simulate the performance and behavior of complex computer systems, including the CPU, memory system, and interconnects. This makes it possible to study the performance of different microarchitectural and architectural choices, as well as the effects of different workloads, without having to build and test real systems.

By ChatGPT



What are we going to cover today?

- A short history of gem5.
- Getting gem5 setup on your system: Compilation, etc..
- Using prebuilt systems.
- An overview of event-based simulation and gem5's software structure
- Simulator outputs and how to interpret it.
- Creating a system using stdlib components.
- Statistical outputs .
- SE-mode and FS-mode simulations.
- Checkpoints, lower-fidelity components, KVM mode, and Sampling
- Creating your own SimObject.
- Creating your own stdlib component.
- Gem5 Resources.
- How to continue to gem5.



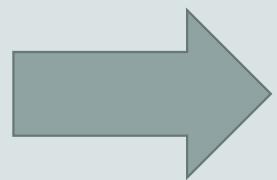
A little bit of history



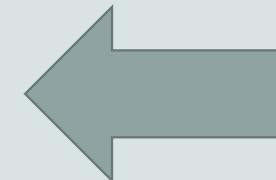
The m5 Simulator

"A tool for simulating
systems"

(~2002)



(2011)



The GEMS simulator

Provided a detailed memory system.

(~2000)



A little bit of history



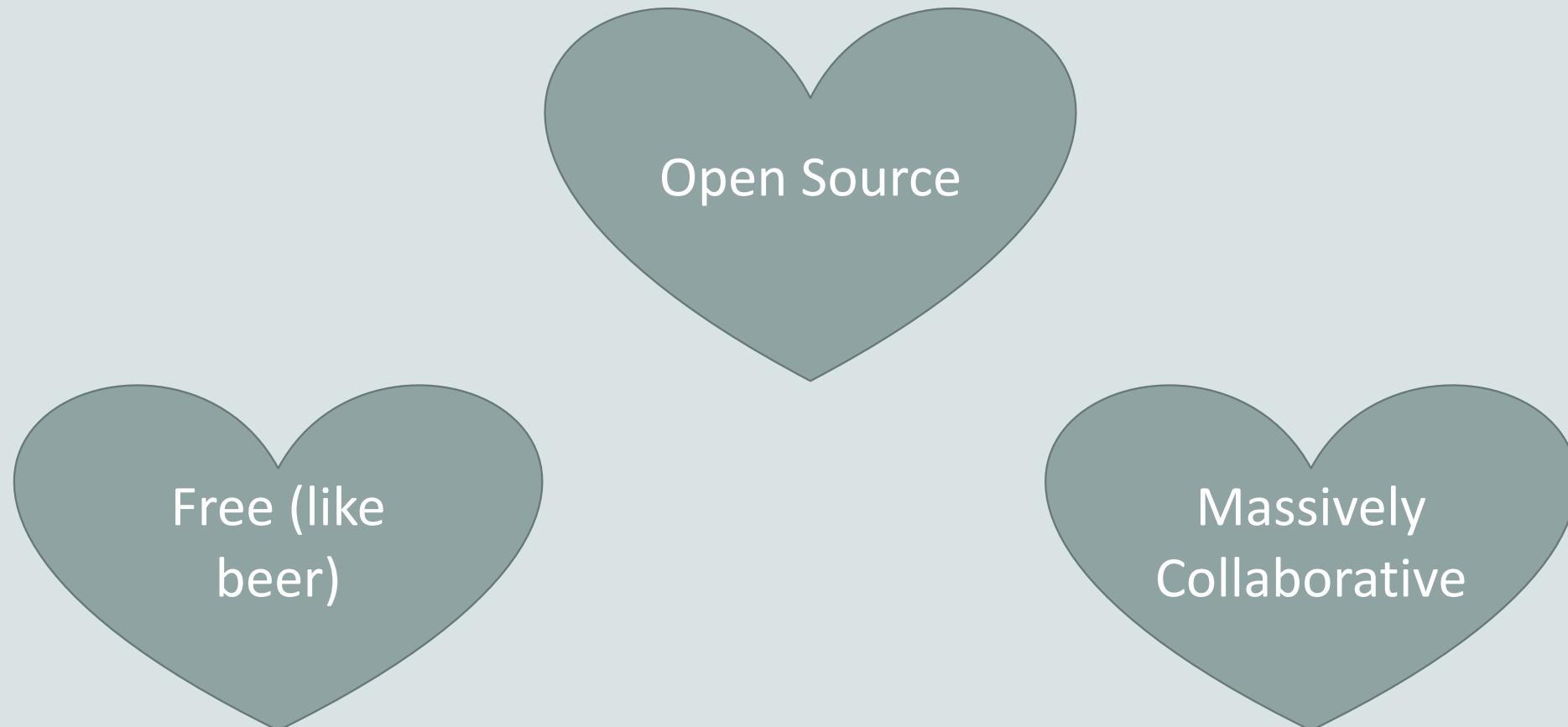
Total Commit*: 21,000

Unique Contributions*: 421



* As of v23.1

A true public infrastructure project



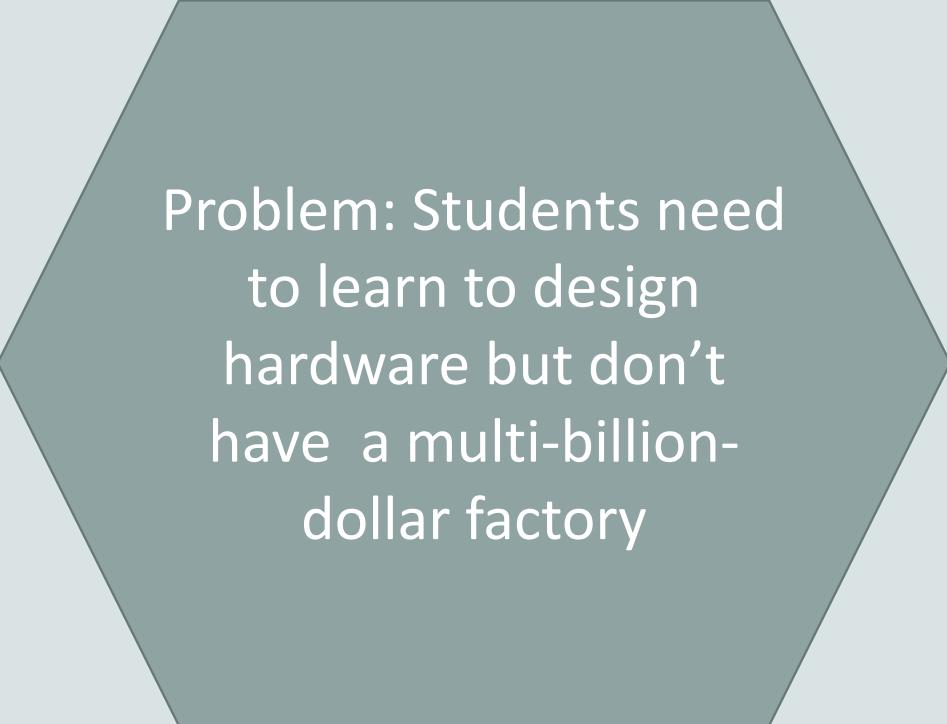
Who uses gem5, and why?

Education

Academic
Research

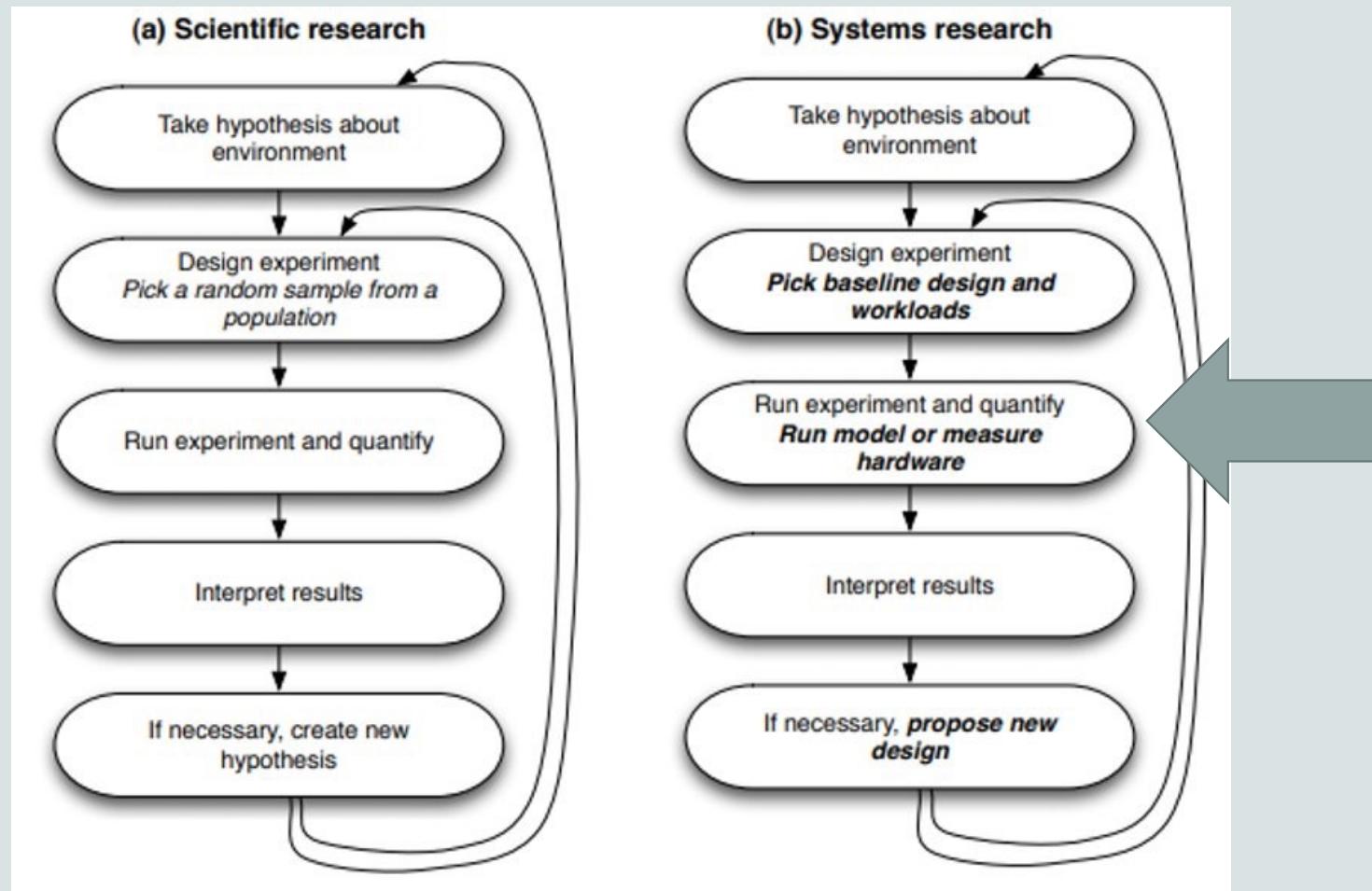
Industrial
R&D

Education



Problem: Students need
to learn to design
hardware but don't
have a multi-billion-
dollar factory

Academic Research



Academic Researchers

We surveyed the top architecture conferences and found:

- 70% of all computer architecture research utilizes simulation.
- The gem5 simulator is by-far the most popular.

Room for improvement: Most users still “roll their own” simulation software. Only 20% use gem5 directly. We want to go above 50% by 2027.

Industry

?

Really, we don't know exactly. We don't track users and industrial users seldom make themselves known.

Industry

Big players we know use it



ARM

AMD The AMD logo, featuring the word "AMD" in a bold black sans-serif font next to a green diamond-shaped icon.

gem5

Let's hit the ground running

This example will show:

1. How someone obtains gem5.
2. How you build it.
3. Running a very basic "Hello World" simulation.



- Getting and compiling gem5 is often the hardest part...
- There's a lot of complicated things happening behind the scenes. I will explain them later.



Typical Downloading



DON'T DO
THIS!

```
> git clone https://github.com/gem5/gem5  
> cd gem5
```

stable: The default branch for gem5. Updated at stable releases. Currently v23.1.

develop: The branch in which new features, improvements, etc. are added regularly for the next release.

In this tutorial we're going to use codes paces with a repo which includes some example materials.

Though all the gem5 code is v23.1.

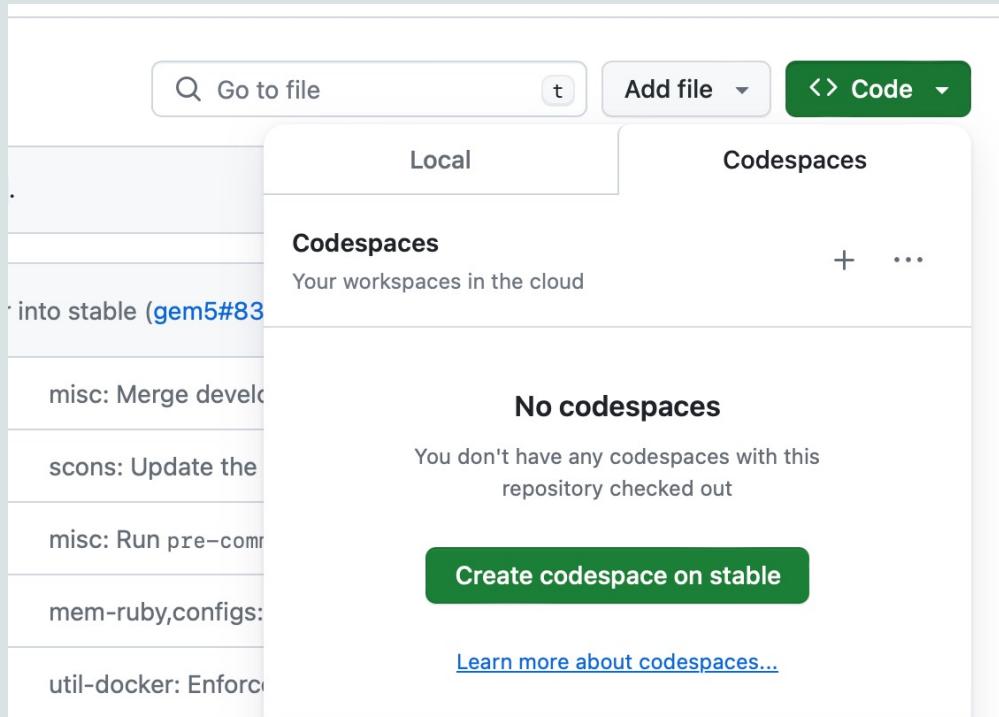
Using CodeSpaces

Step 1: Go to <https://github.com/gem5-hpca-2024/gem5>



Using CodeSpaces

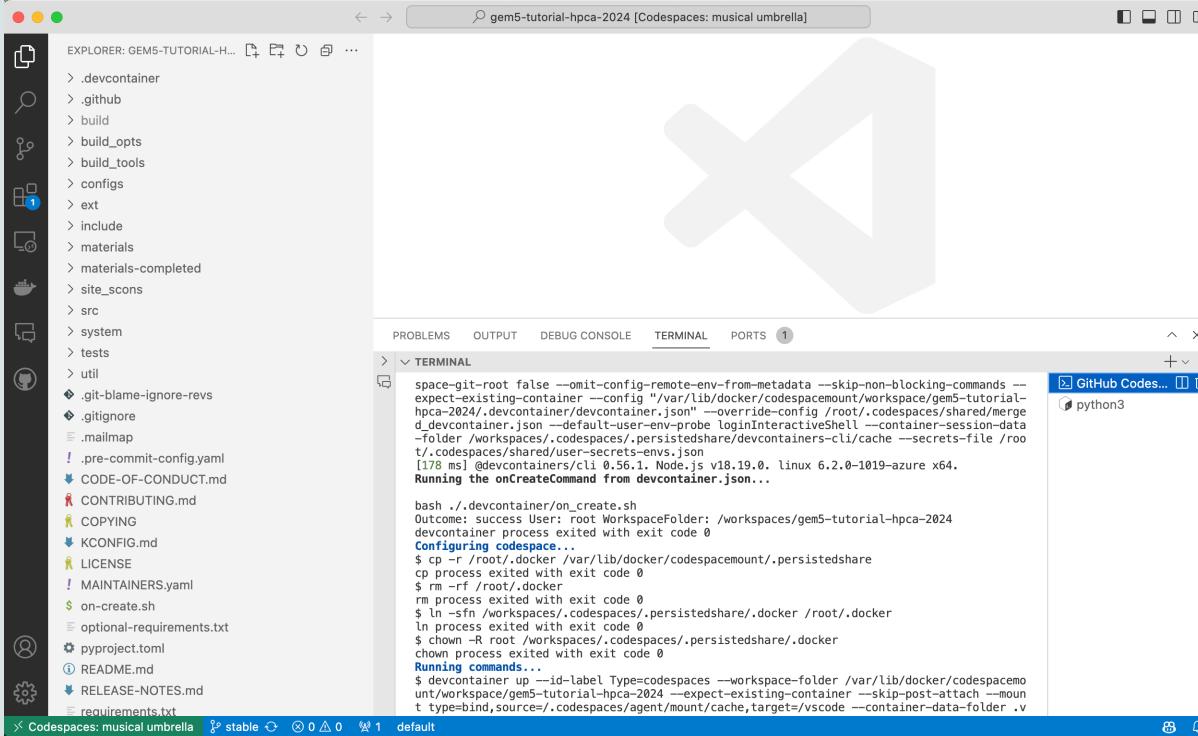
Step 2: Click “Code” -> “Create codespace on stable”



Some may be given the option to open this in a local instance of Visual Studio. This is fine. If you receive no option, you'll run it through the browser. The interface is identical.

Using CodeSpaces

Step 3: Wait for your environment to load. Then you're done



Building gem5

```
> scons build/ALL/gem5.opt -j`nproc`
```

Let's start by writing a simulation configuration

```
from gem5.prebuilt.demo.x86_demo_board import X86DemoBoard  
from gem5.resources.resource import obtain_resource  
from gem5.simulate.simulator import Simulator
```

Open “materials/01-basic.py”. You’ll see the above already prepared for you. Do your work here.

Let's be lazy and use a prebuild board

The X86DemoBoard has the following properties:

- Single Channel DDR3, 2GB Memory.
- A 4 core 3GHz processor (using gem5's 'timing' model).
- A MESI Two Level Cache Hierarchy, with 32kB data and instruction case and a 1MB L2 Cache.
- Will be run as a Full-System simulation.

```
board = X86DemoBoard()
```

Source:

“src/python/gem5/prebuilt/demo/x86_demo_board.py”

Let's load some software!

(And be lazy again... let's use
something pre-made)

```
board.set_workload(obtain_resource("x86-ubuntu-18.04-boot"))
```

“obtain_resource” downloads the files needed to run the specified workload. In this case the “x86-ubuntu-18.04-boot” workload:

1. Boots 18.04 with linux 5.4.49
2. Upon boot will exit the simulation.

<https://resources.gem5.org/resources/x86-ubuntu-18.04-boot?version=2.0.0>

The gem5 Resources Web portal

<https://resources.gem5.org/resources/x86-ubuntu-18.04-boot?version=2.0.0>

gem5-resources /

x86-ubuntu-18.04-boot □

Category: [workload](#)

.x86 VERSION 2.0.0 TAGS None

[Readme](#) [Changelog](#) [Usage](#) [Example](#) [Versions](#) [Raw](#)

A full boot of Ubuntu 18.04 with Linux 5.4.49 for X86. It runs an `m5 exit` command when the boot is completed unless the readfile is specified. If specified the readfile will be executed after booting.

Author
[Unknown](#)

License
Unknown

Properties
Kernel
[x86-linux-kernel-5.4.49](#)

Disk_image
[x86-ubuntu-18.04-img](#)

Function
[set_kernel_disk_workload](#)

Back to the configuration: Put the board in the simulator

```
simulator = Simulator(board=board)
simulator.run(max_ticks=10**10) ... # Run for 10 billion ticks.
```

Note: We're setting “max_ticks” here to stop the simulation after 10 billion simulation ticks. This is just so our simulation stops in a reasonable time `simulator.run()` will have it run to completion.

There! We're done!

```
from gem5.prebuilt.demo.x86_demo_board import X86DemoBoard
from gem5.resources.resource import obtain_resource
from gem5.simulate.simulator import Simulator

# Here we setup the board. The prebuilt X86DemoBoard allows for Full-System X86
# simulation.
board = X86DemoBoard()

# We then set the workload. Here we use the "x86-ubuntu-18.04-boot" workload.
# This boots Ubuntu 18.04 with Linux 5.4.49. If the required resources are not
# found locally, they will be downloaded.
board.set_workload(obtain_resource("x86-ubuntu-18.04-boot"))

# The board is then passed to the simulator and it is run.
simulator = Simulator(board=board)
simulator.run(max_ticks=10**10) # Run for 10 billion ticks.
```

The completed configuration can be found in “materials-completed/01-basic.py”



Run the simulation

```
> ./build/ALL/gem5.opt materials/01-basic.py
```

```
gem5 Simulator System. https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.1.0.0
gem5 compiled Feb 13 2024 15:23:03
gem5 started Feb 13 2024 15:50:24
gem5 executing on page, pid 29650
command line: ./build/ALL/gem5.opt materials-completed/01-basic.py

warn: The X86DemoBoard is solely for demonstration purposes. This board is not known to be representative of any real-world system.
      Use with caution.
info: Using default config
Global frequency set at 100000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:692: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (2048 Mbytes)
src/sim/kernel_workload.cc:46: info: kernel located at: /Users/bobbyrbruce/.cache/gem5/x86-linux-kernel-5.4.49
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
          0: board.pc.south_bridge.cmos rtc: Real-time clock set to Sun Jan 1 00:00:00 2012
board.pc.com_1.device: Listening for connections on port 3456
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
src/dev/intel_8254_timer.cc:128: warn: Reading current count from inactive timer.
board.remote_gdb: Listening for connections on port 7000
src/sim/simulate.cc:199: info: Entering event queue @ 0. Starting simulation...
src/mem/ruby/system/Sequencer.cc:670: warn: Replacement policy updates recently became the responsibility of SLICC state machines. Make sure to setMRU() near callbacks in .sm files!
build/ALL/arch/x86/generated/exec-ns.cc.inc:27: warn: instruction 'fninit' unimplemented
src/dev/x86/pc.cc:117: warn: Don't know what interrupt to clear for console.
```

Let's check other outputs here...

- m5out/
---- board.pc.com_1.device

The terminal output
of the simulated
system.

```
Linux version 5.4.49 (aakahlow@amarillo) (gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #8 SMP Mon Jun 29 14:40:17 PDT 2020
Command line: earlyprintk=ttyS0 console=ttyS0 lpj=7999923 root=/dev/hda1 disk_device=/dev/hda
x86/fpu: x87 FPU will use FXSAVE
BIOS-provided physical RAM map:
BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
BIOS-e820: [mem 0x000000000009fc00-0x00000000000fffff] reserved
BIOS-e820: [mem 0x0000000000100000-0x000000007fffffff] usable
BIOS-e820: [mem 0x00000000ffff0000-0x00000000ffffffff] reserved
printk: bootconsole [earlyser0] enabled
NX (Execute Disable) protection: active
SMBIOS 2.5 present.
DMI: , BIOS: 06/08/2008
tsc: Fast TSC calibration using PIT
tsc: Detected 3003.010 MHz processor
last_pfn = 0x80000 max_arch_pfn = 0x400000000
Disabled
x86/PAT: MTRRs disabled, skipping PAT initialization too.
CPU MTRRs all blank -- virtualized system.
x86/PAT: Configuration [0-7]: WB WT UC UC WB WT UC UC
found SMP MP-table at [mem 0x000f0050-0x000f005f]
ACPI: Early table checksum verification disabled
ACPI: RSDP:0x00000000000F01F0 000024 (v02-*****)
ACPI: XSDT:0x00000000000F0238 000024 (v01-***** 00000000 00000000)
ACPI BIOS Error (bug): Invalid table length 0x24 in RSDT/XSDT (20190816/tbutils-291)
```

Let's check other outputs here...

- m5out/
---- board.pc.com_1.device
---- config.{ini/json}

A record of the simulated system in both JSON and INI format.

```
[board]
type=System
children=cache_hierarchy clk_domain dvfs_handler iobus memory pc processor workload
auto_unlink_shared_backstore=false
cache_line_size=64
eventq_index=0
exit_on_work_items=true
init_param=0
m5ops_base=4294901760
mem_mode=timing
mem_ranges=0:2147483648-3221225472:3222274048
memories=board.memory.mem_ctrl.dram
mmap_using_noreserve=false
multi_thread=false
num_work_ids=16
readfile=
redirect_paths=
shadow_rom_ranges=
shared_backstore=
symbolfile=
thermal_components=
thermal_model=NULL
work_begin_ckpt_count=0
work_begin_cpu_id_exit=-1
work_begin_exit_count=0
work_cpus_ckpt_count=0
work_end_ckpt_count=0
work_end_exit_count=0
work_item_id=-1
workload=board.workload
system_port=board.cache_hierarchy.ruby_system.sys_port_proxy.i
```

```
        "type": "Root",
        "cxx_class": "gem5::Root",
        "name": null,
        "path": "root",
        "eventq_index": 0,
        "full_system": true,
        "sim_quantum": 0,
        "time_sync_enable": false,
        "time_sync_period": 1000000000000,
        "time_sync_spin_threshold": 100000000,
        "board": {
            "type": "System",
            "cxx_class": "gem5::System",
            "name": "board",
            "path": "board",
            "auto_unlink_shared_backstore": false,
            "cache_line_size": 64,
            "eventq_index": 0,
            "exit_on_work_items": true,
            "init_param": 0,
            "m5ops_base": 4294901760,
            "mem_mode": "timing",
            "mem_ranges": [
                "0:2147483648",
                "3221225472:3222274048"
            ],
            "memories": [
                "board.memory.mem_ctrl.dram"
            ],
            "mmap_using_noreserve": false,
            "multi_thread": false,
            "num_work_ids": 16,
            "readfile": "",
            "redirect_paths": [],
            "shadow_rom_ranges": []}
```



Let's check other outputs here...

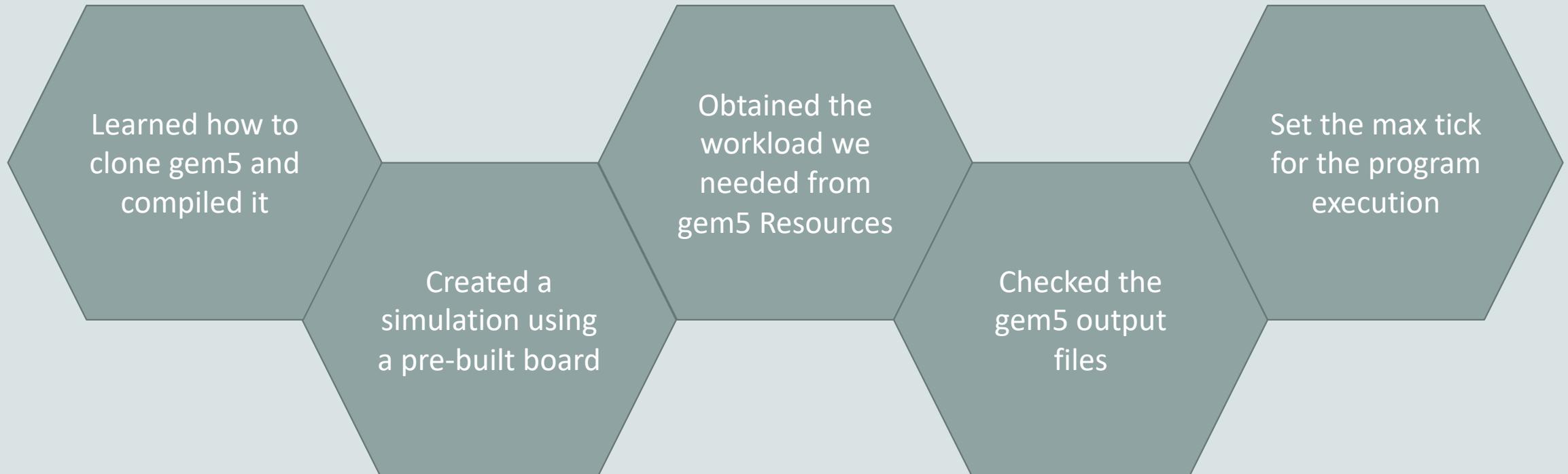
- m5out/
- board.pc.com_1.device
- config.{ini/json}
- stats.txt

The gem5
statistic output

----- Begin Simulation Statistics -----	
simSeconds.....	0.010000
simTicks.....	100000000000
finalTick.....	10000000000
simFreq.....	1000000000000
hostSeconds.....	7.92
hostTickRate.....	1262226239
hostMemory.....	406582560
simInsts.....	1473828
simOps.....	18070009
hostInstRate.....	186030
hostOpRate.....	2280835

Wait, did we do?

We...



Ok, but how does it all work?

Modern systems are very complex, and the design of gem5 simulations reflects this.

However, at its core, the simulator builds on a relatively simple model.

Nomenclature

Host: the actual hardware you're using

Running things directly on the hardware:

Native execution

Guest: Code running on top of “fake” hardware

OS in virtual machine is guest OS

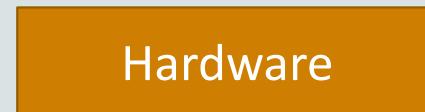
Running “on top of” hypervisor

Hypervisor is emulating hardware

Your system



Operating Sys.



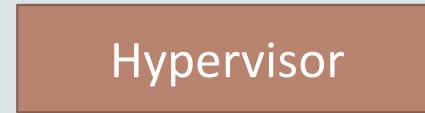
Host

Hardware

Virtual machines

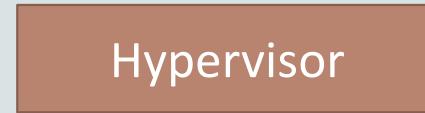


Operating Sys.



Guest

App



Host

Hardware

Nomenclature

Host: the actual hardware you're using

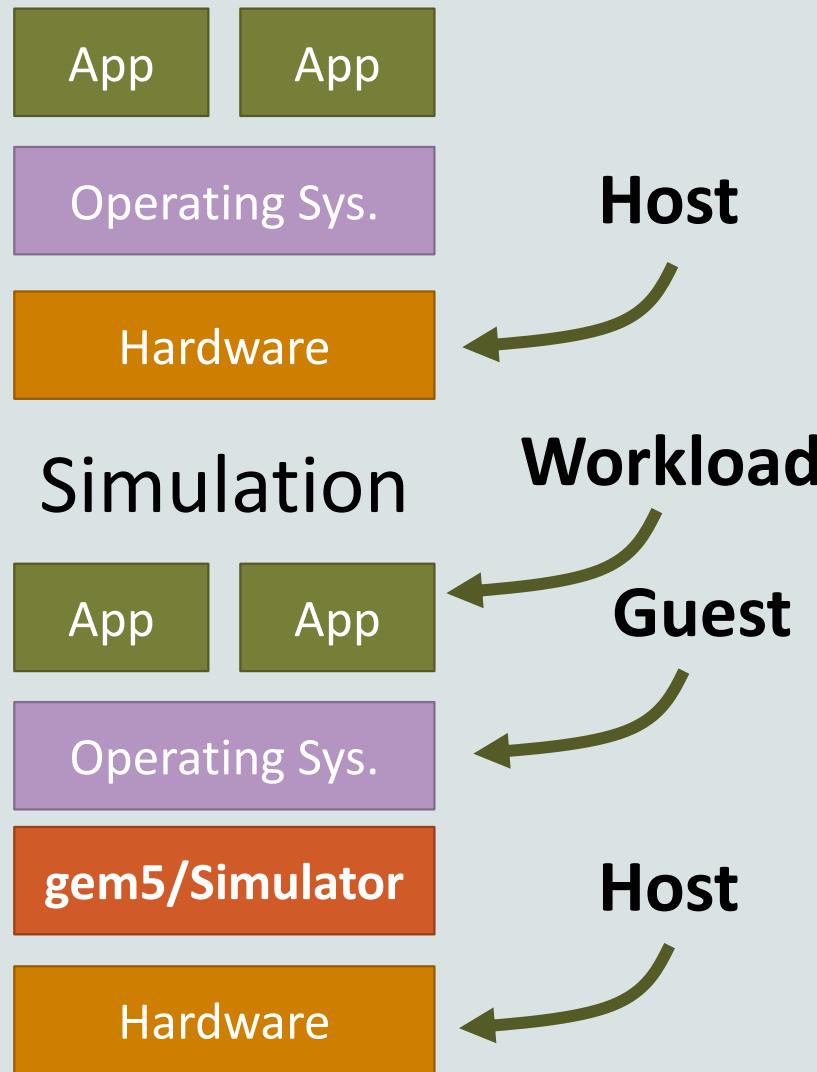
Simulator: Runs on the host
Exposes hardware to the guest

Guest: Code running on *simulated* hardware
OS running on gem5 is guest OS
gem5 is simulating hardware

Simulator's code: Runs natively
executes/emulates the guest code

Guest's code: (or benchmark, workload, etc.)
Runs on gem5, not on the host.

Your system



Nomenclature

Host: the actual hardware you're using

Simulator: Runs on the host
Exposes hardware to the guest

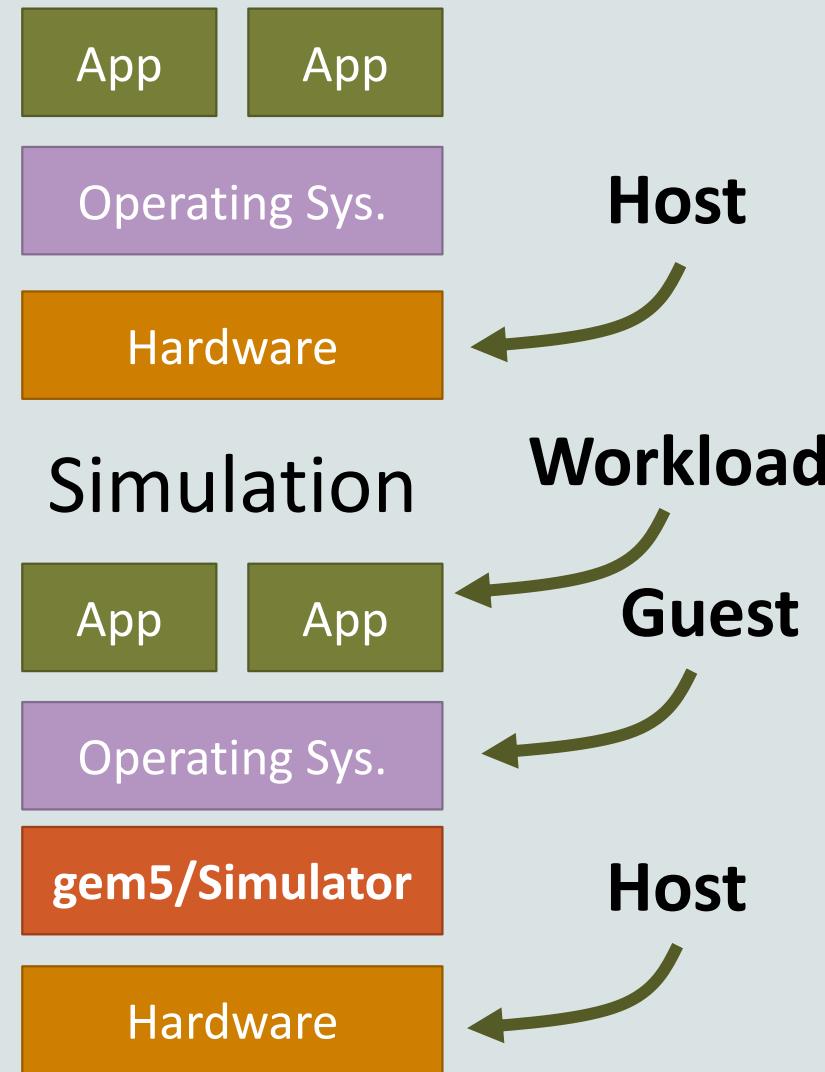
Simulator's performance:

Time to run the simulation on host
Wallclock time as you perceive it

Simulated performance:

Time predicted by the simulator
Time for guest code to run on
simulator

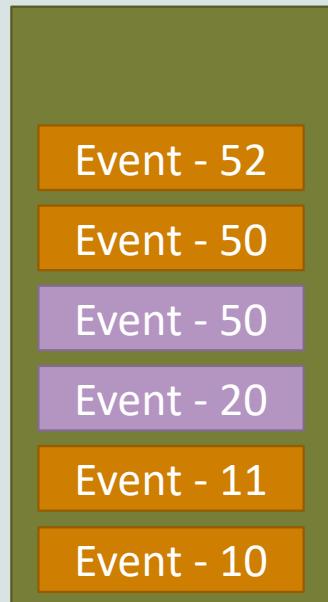
Your system



At its core: it's a discrete event simulator

gem5 is a **discrete event simulator**

Event Queue

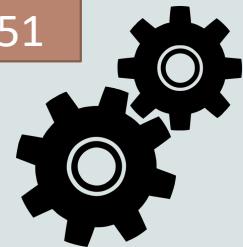
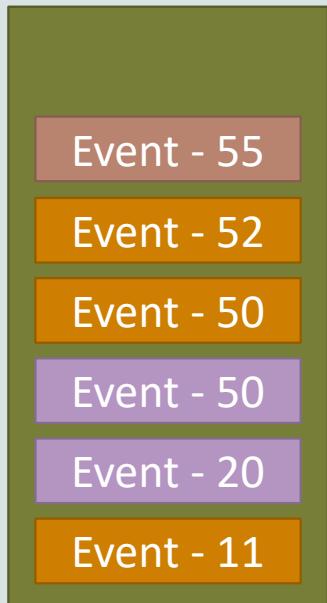


- 1) Event at head dequeued
- 2) Event executed
- 3) More events queued

At its core: it's a discrete event simulator

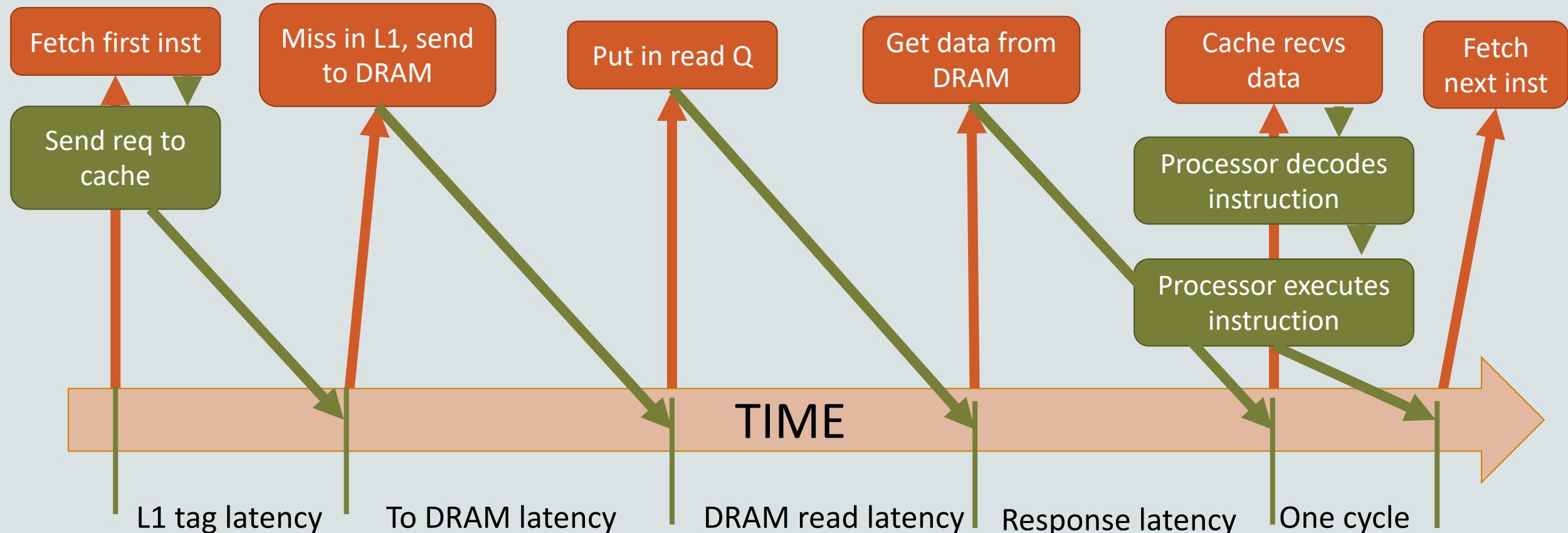
gem5 is a **discrete event simulator**

Event Queue



- 1) Event at head dequeued
- 2) Event executed
- 3) More events queued

Discrete event simulation example



Discrete event simulation

"Time" needs a unit

In gem5, we use a unit called "Tick"

Need to convert a simulation "tick" to user-understandable time

E.g., seconds

This is the global simulation tick rate

Usually this is 1 ps per tick or 10^{12} ticks per second

Ok, but how do you schedule these events?



While some are incredibly complex, at their core they only do two things:

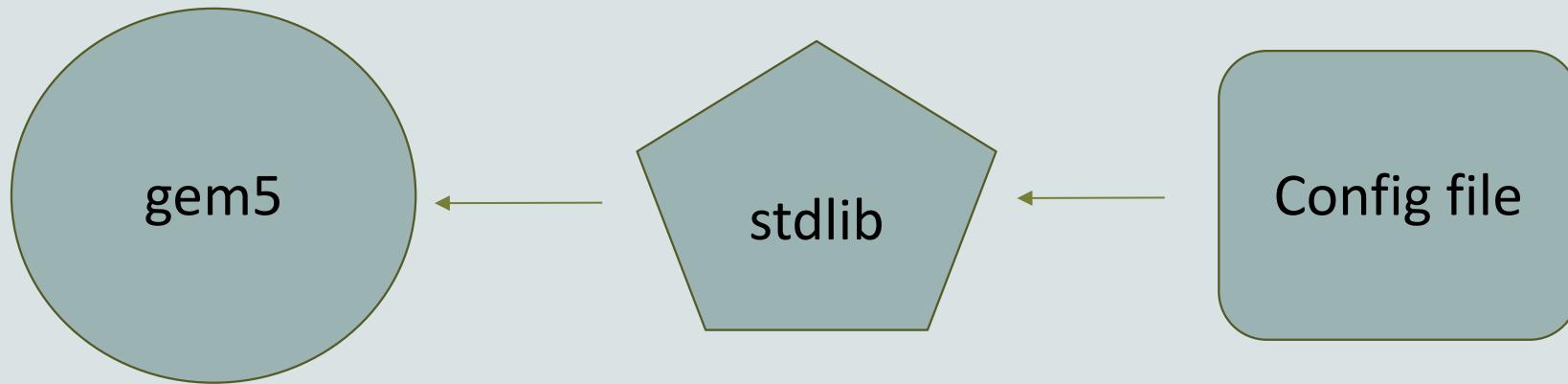
1. Schedule events and process events.
2. Talk to other SimObjects.

This is great, but it takes a lot to get things done



This allows for maximum flexibility but can mean creating 100s of lines of Python to create even a basic simulation.

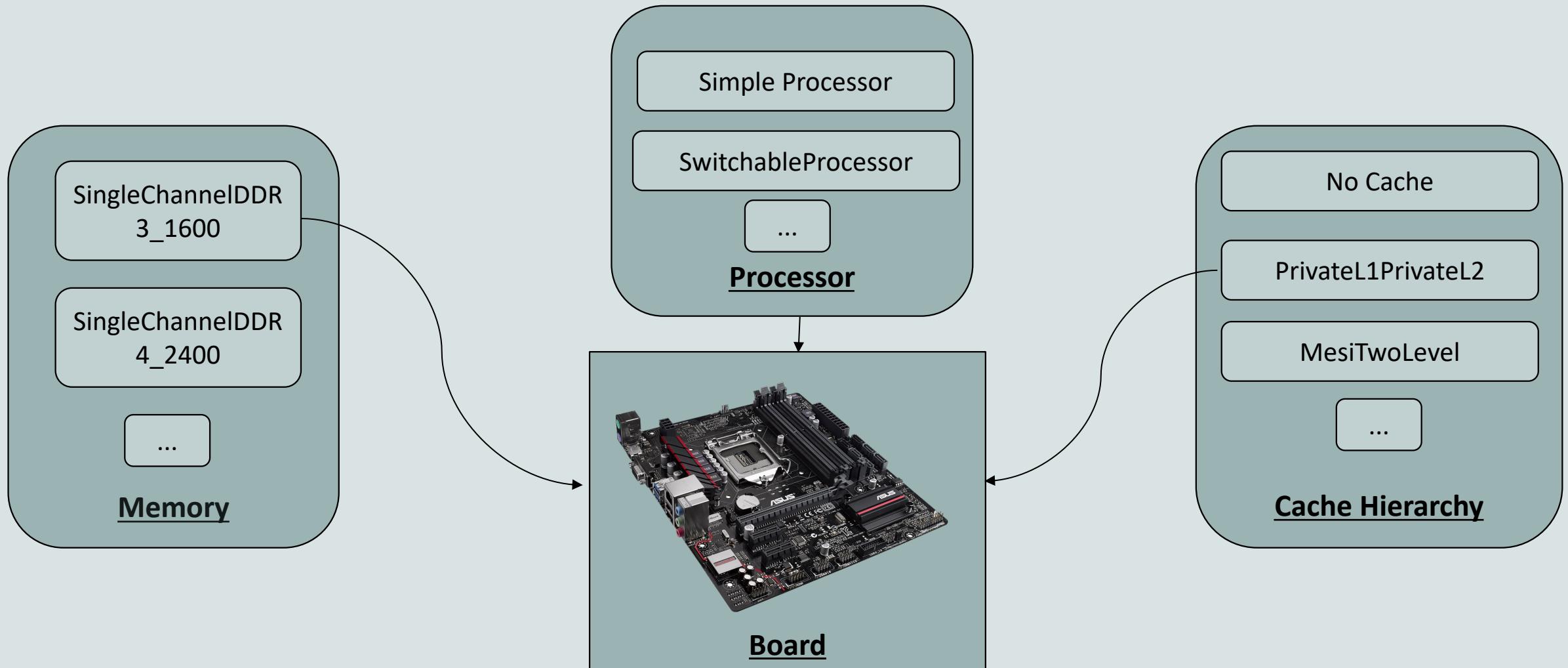
What is the standard library for?



The stdlib is a library which allows for users to quickly create systems with pre-built components.

The stdlib's module architecture allows for components (e.g. a memory system or a cache hierarchy setup) to be quickly swapped in and out without radical redesign.

The stdlib modular metaphor



Let's build using components!

```
from gem5.coherence_protocol import CoherenceProtocol
from gem5.components.boards.x86_board import X86Board
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.cpu_types import CPUTypes
from gem5.components.processors.simple_switchable_processor import (
    SimpleSwitchableProcessor,
)
from gem5.isas import ISA
from gem5.resources.resource import obtain_resource
from gem5.simulate.exit_event import ExitEvent
from gem5.simulate.simulator import Simulator
from gem5.utils.requires import requires
```

Open “materials/02-components.py” You’ll see the above already prepared for you.

Let's choose a cache hierarchy and memory system

```
cache_hierarchy = MESITwoLevelCacheHierarchy(  
    l1d_size="16kB",  
    l1d_assoc=8,  
    l1i_size="16kB",  
    l1i_assoc=8,  
    l2_size="256kB",  
    l2_assoc=16,  
    num_l2_banks=1,  
)
```

```
memory = SingleChannelDDR3_1600(size="3GB")
```

And a processor!

```
processor = SimpleSwitchableProcessor(  
    ... starting_core_type=CPUTypes.TIMING,  
    ... switch_core_type=CPUTypes.03,  
    ... isa=ISA.X86,  
    ... num_cores=2,  
)
```

The SimpleSwitchingProcessor allows for different types of cores to be swapped during a simulation with `processor.switch()`.

This can be useful when wanting to switch to and from a detailed form of simulation. More on this later.

Then let's plug them into a board

```
board = X86Board(  
    ... clk_freq="3GHz",  
    ... processor=processor,  
    ... memory=memory,  
    ... cache_hierarchy=cache_hierarchy,  
)
```

We add the components to the board, in this case an `X86Board`.

Load in our workload to the board

```
command := (
    "m5 exit;" +
    "echo 'This is running on 03 CPU cores.';" +
    "sleep 1;" +
    "m5 exit;"
)

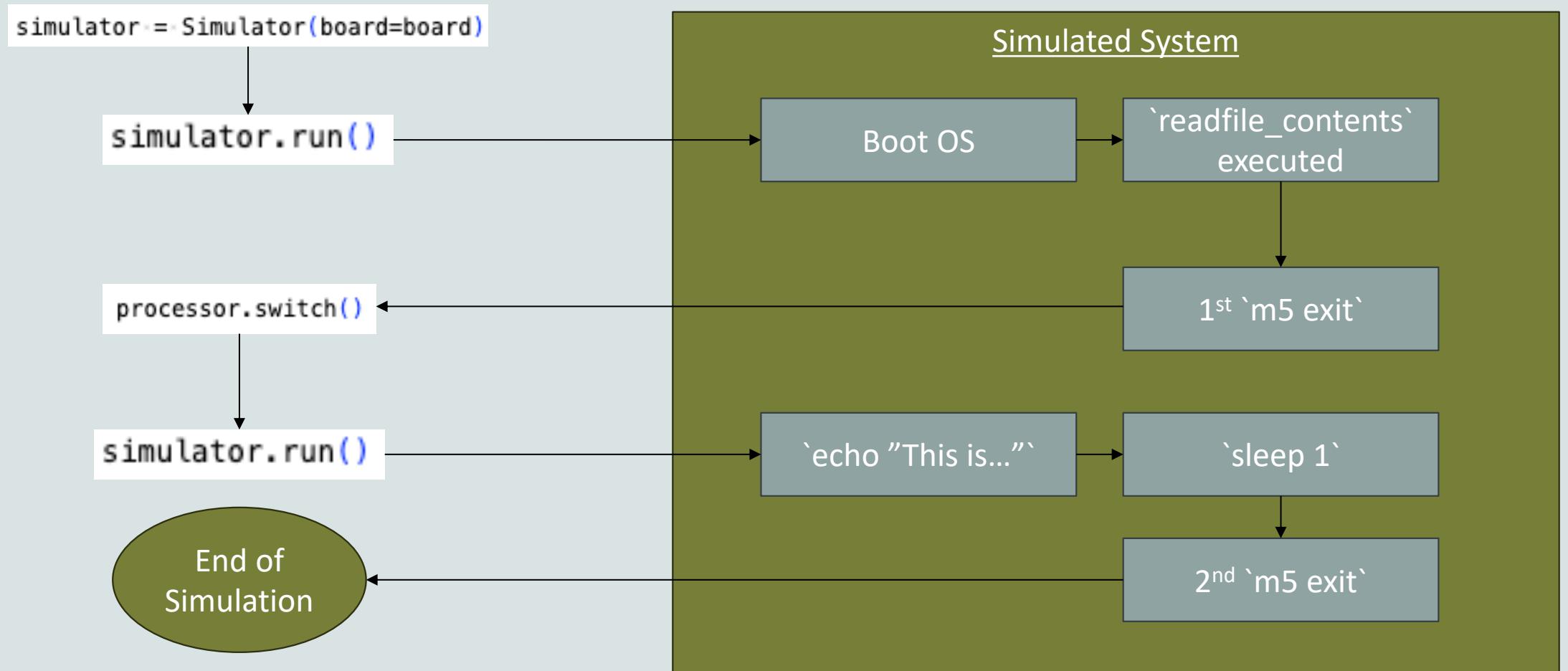
board.set_kernel_disk_workload(
    kernel=obtain_resource("x86-linux-kernel-4.4.186"),
    disk_image=obtain_resource("x86-ubuntu-18.04-img"),
    readfile_contents=command,
)
```

The `set_kernel_disk_workload` function accepts a kernel binary resources and a disk images resources.

In addition, here we are passing a command via the `readfile_contents` parameter which will run after the boot is complete.

The `m5 exit` command will exit the simulation loop.

Run the simulator



Run the simulator

```
> ./build/ALL/gem5.opt materials/02-components.py
```

A completed version of the configuration can be found in “materials-completed/02-01-components-initial.py”

We can do better...

```
simulator = Simulator(  
    ...  
    board=board,  
    ...  
    on_exit_event={  
        ...  
        | ExitEvent.EXIT : (func() for func in [processor.switch])  
        ...  
    }  
  
    simulator.run()
```

Here we can specify exactly what to do on each exit event type via Python generators.

The Simulator had default behavior for these events, but they can be overridden.

- ExitEvent.EXIT
- ExitEvent.CHECKPOINT
- ExitEvent.FAIL
- ExitEvent.SWITCHCPU
- ExitEvent.WORKBEGIN
- ExitEvent.WORKEND
- ExitEvent.USER_INTERRUPT
- ExitEvent.MAX_TICK

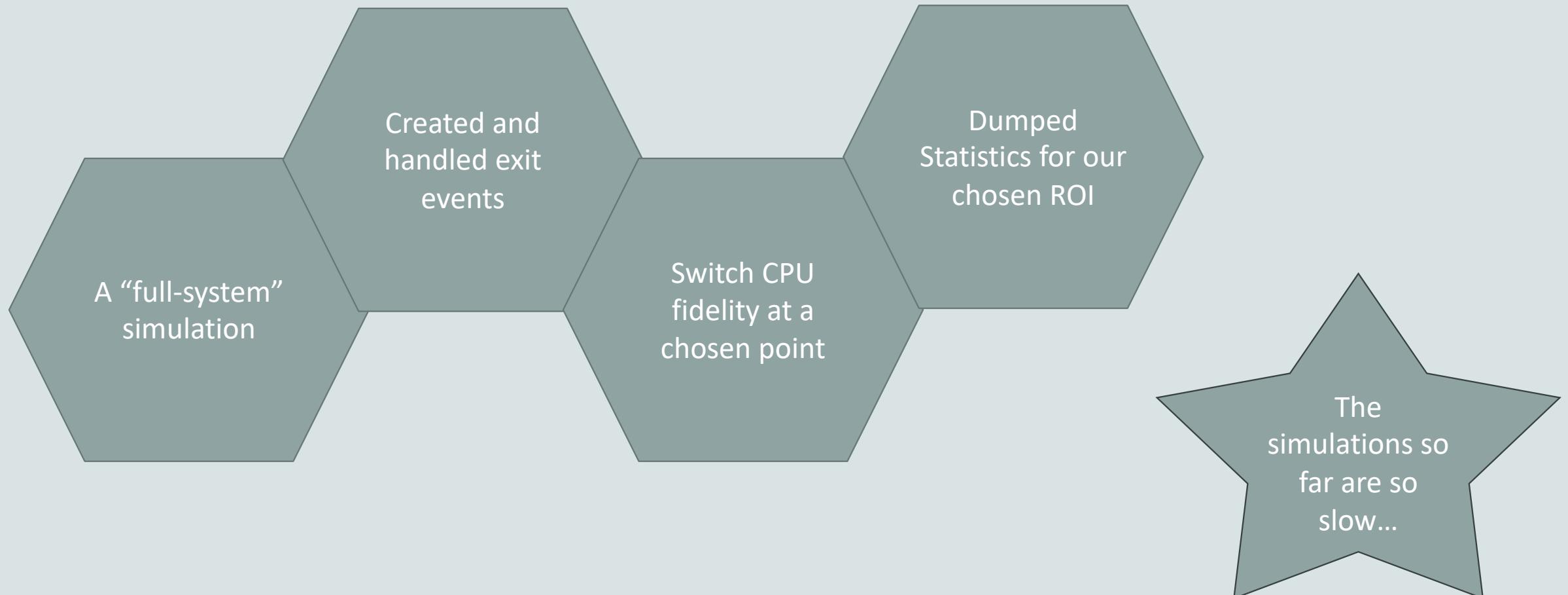
To run

```
> ./build/ALL/gem5.opt materials/02-components.py
```

This will take a while to run but you can
see the terminal output in “m5out”
while this is running.

Cntr+C to exit the simulation.

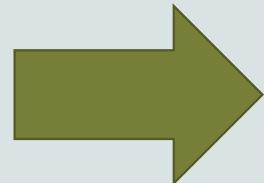
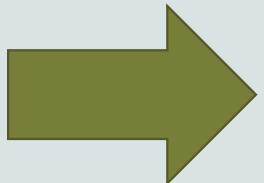
What did we just do?



gem5 is slloooww

(Not our fault: It's the nature of simulation)

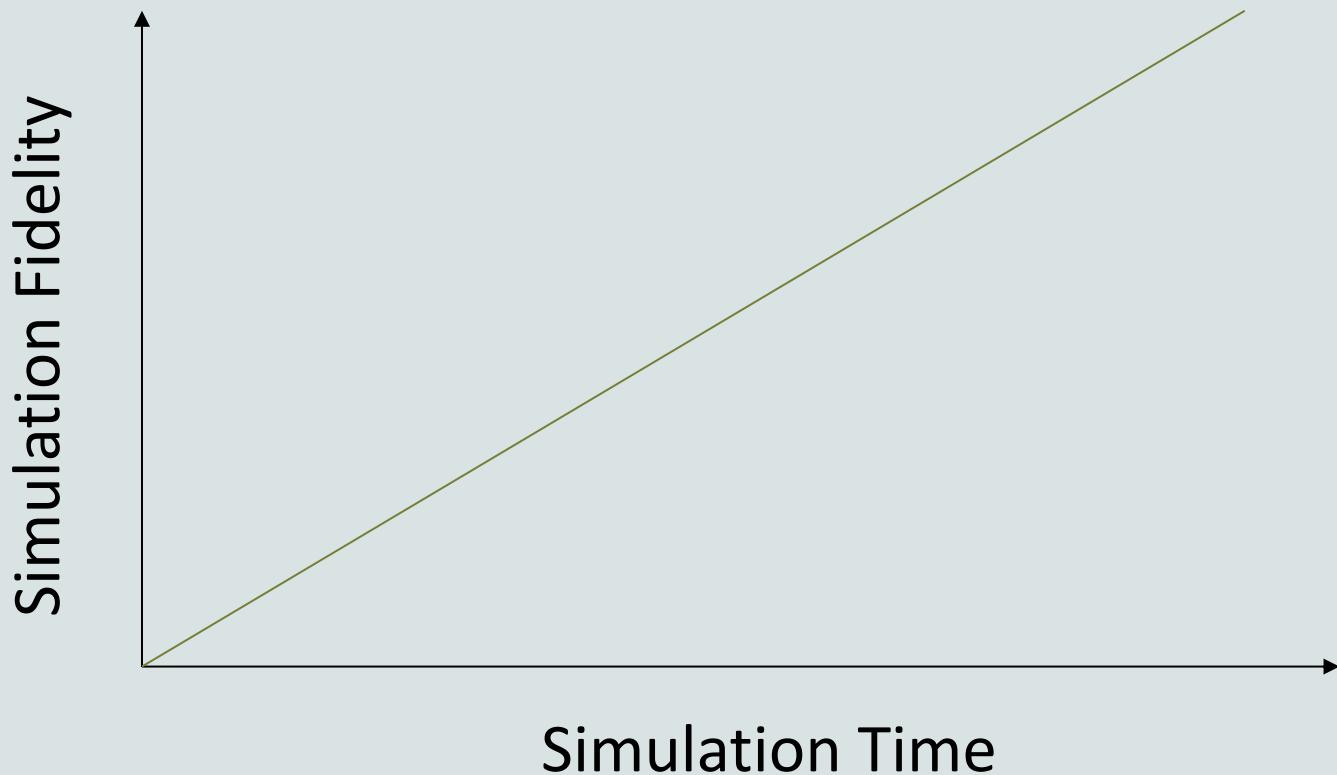
Simulating
1 Second



>> 100k
seconds on the
host



Fortunately, there are some work arounds

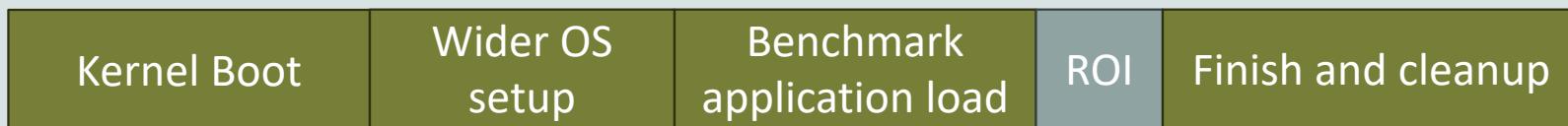


Key idea: You don't need to simulate everything perfectly, or at all.

Simulations can always be made faster by simulating less

```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
              // guaranteed to be random.
}
```

This isn't always a bad thing... a lot of a simulation is of no interest to us



Some techniques we provide

CPU Models

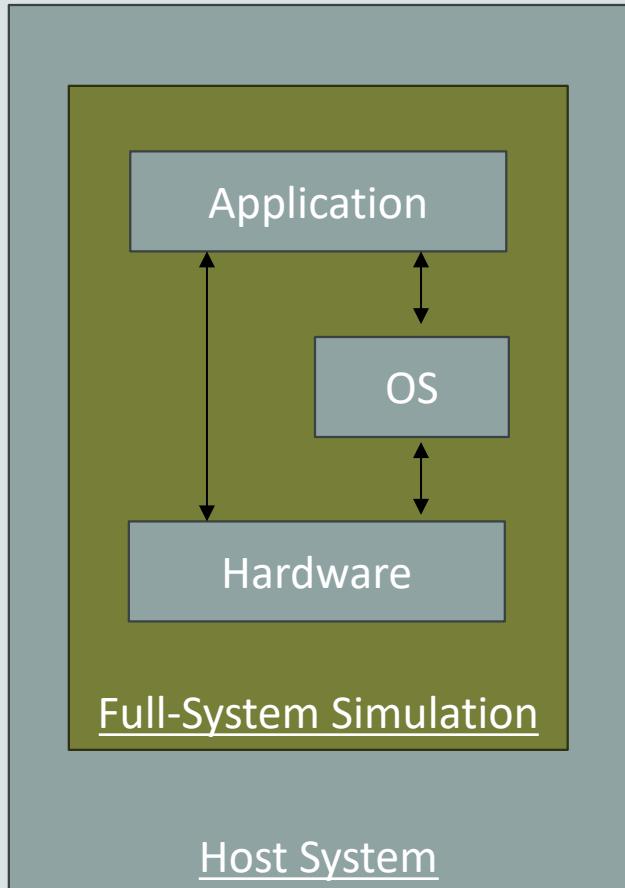
KVM Mode

SE Mode

Checkpoints

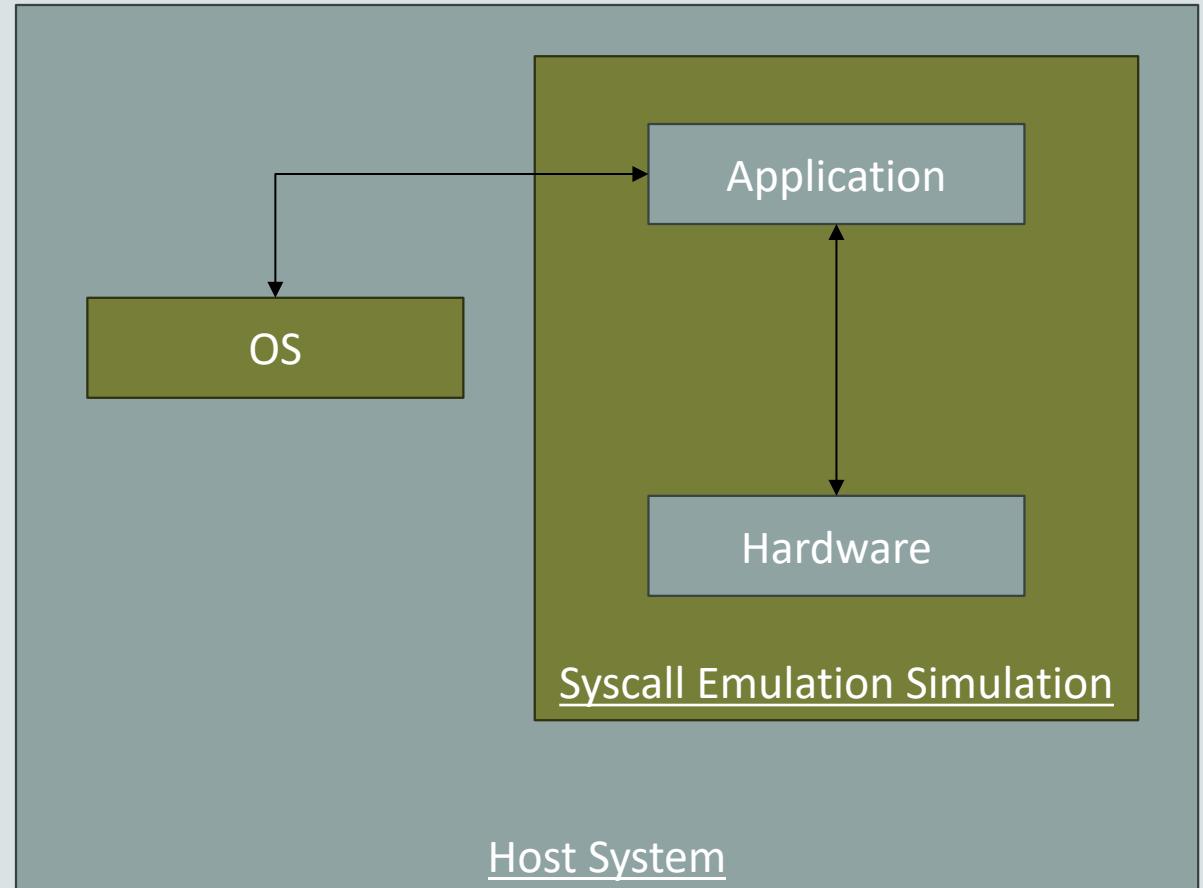
Sampling:
Simpoints
and
Loopoints

SE mode vs FS mode

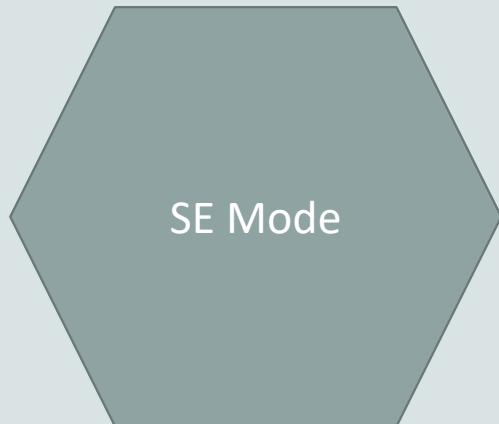


SE Mode relays application syscalls to host OS. This means we don't need to simulate an OS for applications to run

In addition, we can access host resources such as files or libraries to dynamically link in.



SE mode example



A pre-made script which uses SE-mode can be found in “materials/03-se-mode.py”.

It’s very similar to the scripts you’ve created thus far but uses the ‘SimpleBoard’.

Let’s try to understand the script first before demoing SE Mode. It includes some features we’ve yet to explain.

Suites (and argparse)

```
parser = argparse.ArgumentParser(  
    description="This script shows how to use a suite. In this example, we"  
    "will use the ARM Getting Started Benchmark Suite, and show"  
    "the different functionalities of the suite.",  
)  
  
# Obtain the ARM "Getting Started" Benchmark Suite.  
microbenchmarks = obtain_resource("arm-getting-started-benchmark-suite")  
  
# Give these as an option to the user to select and run the benchmark of their  
# choice.  
parser.add_argument(  
    "benchmark",  
    type=str,  
    choices=[benchmark.get_id() for benchmark in microbenchmarks],  
    help=f"The benchmark from the {microbenchmarks.get_id()} suite to run.",  
)  
  
# Get the arguments from the command line parser.  
args = parser.parse_args()
```

The require function: Making sure we built the right thing

```
requires(isa_required=ISA.ARm)
```

SE mode

```
> ./build/ALL/gem5.opt materials/03-se-mode.py \
    arm-matrix-multiply-run
```

This will take about a minute to run.

This would've taken hours to run in FS, but in SE ignore the entire OS simulation

You can try other CLI options too.

FS and SE mode: Common Fit falls

“FS mode takes too long so I just switched over to SE mode”

*You **must** understand what you’re simulating and whether
it’ll impact results.*

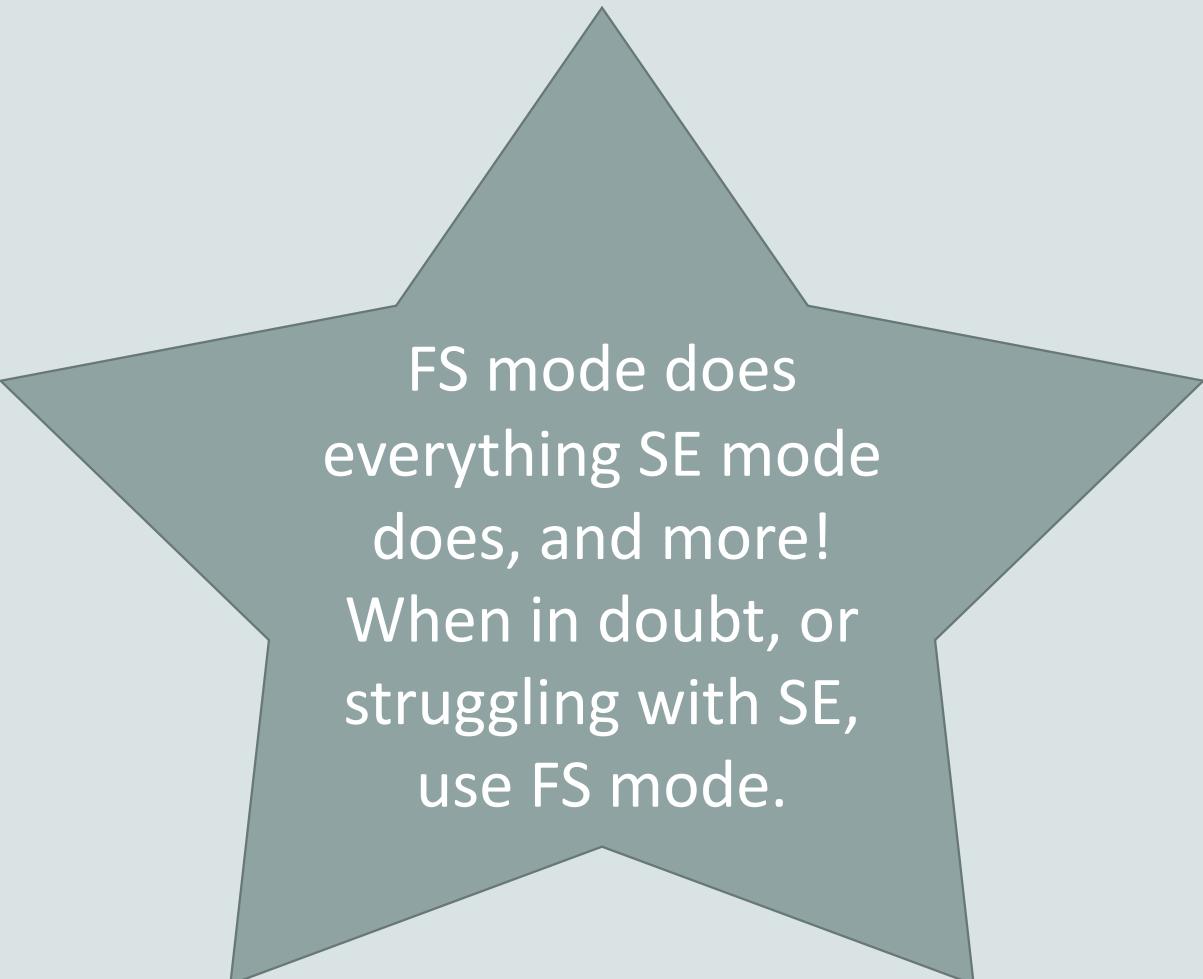
“gem5 Won’t let me run a binary with elevated instructions”

*SE-mode will only allow you to simulate user-space instructions.
We can’t just pass elevated instructu*

“I can’t run my program in SE-Mode because a syscall isn’t
implemented correctly”

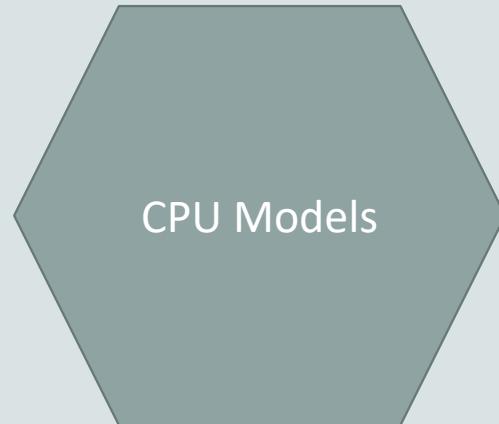
We’d love to have all the syscalls implemented but it’s too much work.

FS and SE mode



FS mode does
everything SE mode
does, and more!
When in doubt, or
struggling with SE,
use FS mode.

CPU Models



Memory Accesses

How memory are simulated

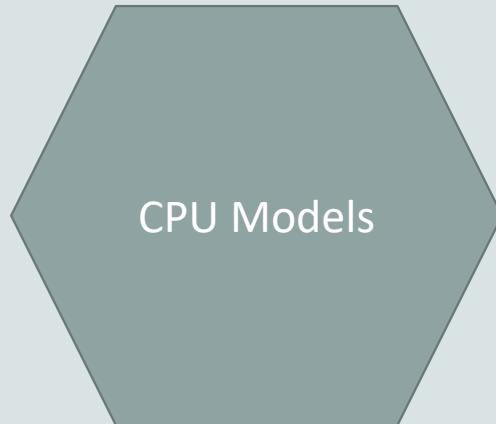
- Timing
- Atomic

CPU Design

The type of CPU design. The simpler the faster.

- O3
- Pipeline
- Simple

CPU Models: Memory Accesses



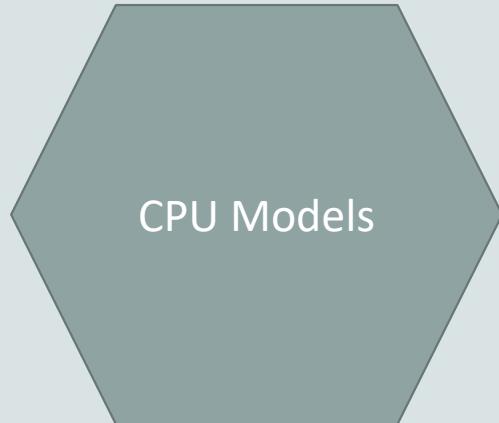
Atomic Accesses

When a memory is accessed, the CPU gets the data instantly and latency is simply estimated for statistics. I.e., everything is happening in one function call from the CPU, in a single event in the Queue.

Low fidelity but fast!

Only suitable if you don't care about memory access data (kind of rare for computer architects...).

CPU Models: Memory Accesses



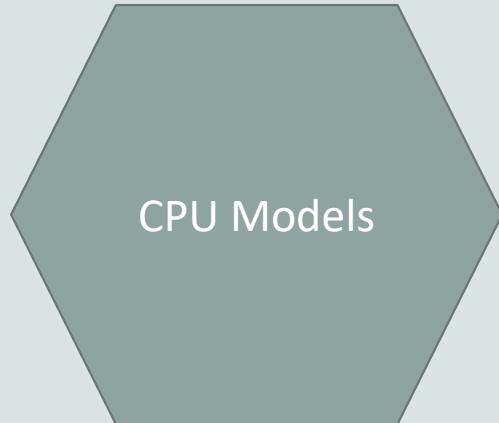
Timing Accesses

Memory requests are responded to by scheduling events in the future in the event queue.

I.e., a request is made by the CPU to memory and the response is scheduled in the event queue.

Low but more accurate.

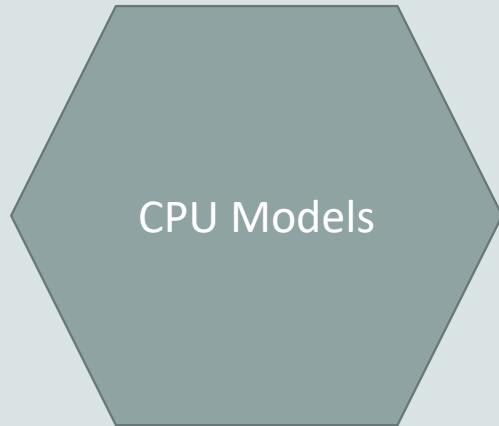
CPU Models: Simple



Simple CPU

A basic in-order CPU model. Due to its relative simplicity, it is relatively easy to simulate compare to other CPU model types.

CPU Models: Simple



Minor CPU

A CPU model for simulating pipelines. The Minor CPU is highly configurable.

Let's play around with a few different CPU models

Go back to the “materials/03-se-mode.py” script.

Try changing the `cpu_type` field of the `SimpleProcessor` and see how it impacts simulation execution time (see ‘host seconds’ in “m5out/stats.txt” for this information)

CPUTypes.TIMING

A Simple CPU using the Timing memory access model.

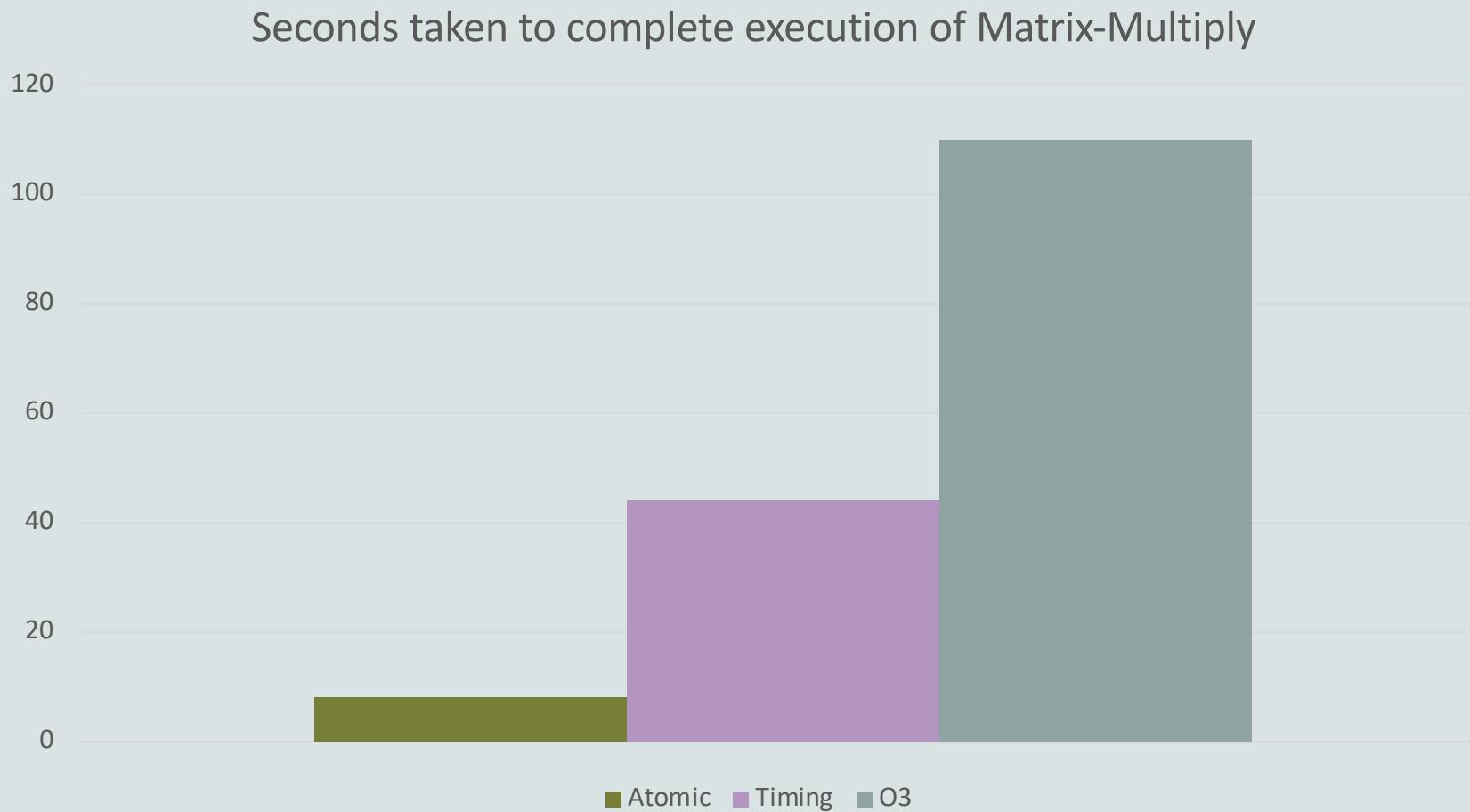
CPUTypes.ATOMIC

A Simple CPU using the Atomic memory access model.

:CPUTypes.03

A O3 CPU using the Timing memory access model.

What I observed with matrix-multiply



KVM Mode: The other special CPU Type

Due to GitHub Codespace restrictions, we cannot use this special CPU type.

If you later wish to check your host system's compatibility and how to setup KVM see:

https://www.gem5.org/documentation/general_docs/using_kvm/

The KVM CPU utilizes the host systems Kernel Virtual Machine to utilize the host hardware to execute instructions.

The host must have KVM enabled and be of the same ISA type as that being emulated.

KVM always runs with Atomic memory accesses. It is simulating almost nothing and is therefore very fast.

```
processor = SimpleProcessor(cpu_type=CPUTypes.KVM, isa=ISA.ARM, num_cores=1)
```

Checkpointing

Checkpointing allows for a simulation to save its state then load to that state later.

This is useful in cases where you want to quickly jump to points in .

This is literally like a checkpoint/save in a video game.

The simulated system that restores the checkpoint can differ from that which created it.

There are some restrictions:

1. You can't restore to a system with less memory.
2. You can't restore with a different workload.
3. Ruby Cache Hierarchy's are wiped. Ergo you start with a clean cache from restore.

Saving a Checkpoint

Open “materials/04-01-saving-a-checkpoint.py”.

Let’s first go through the file to understand what this configuration is doing.

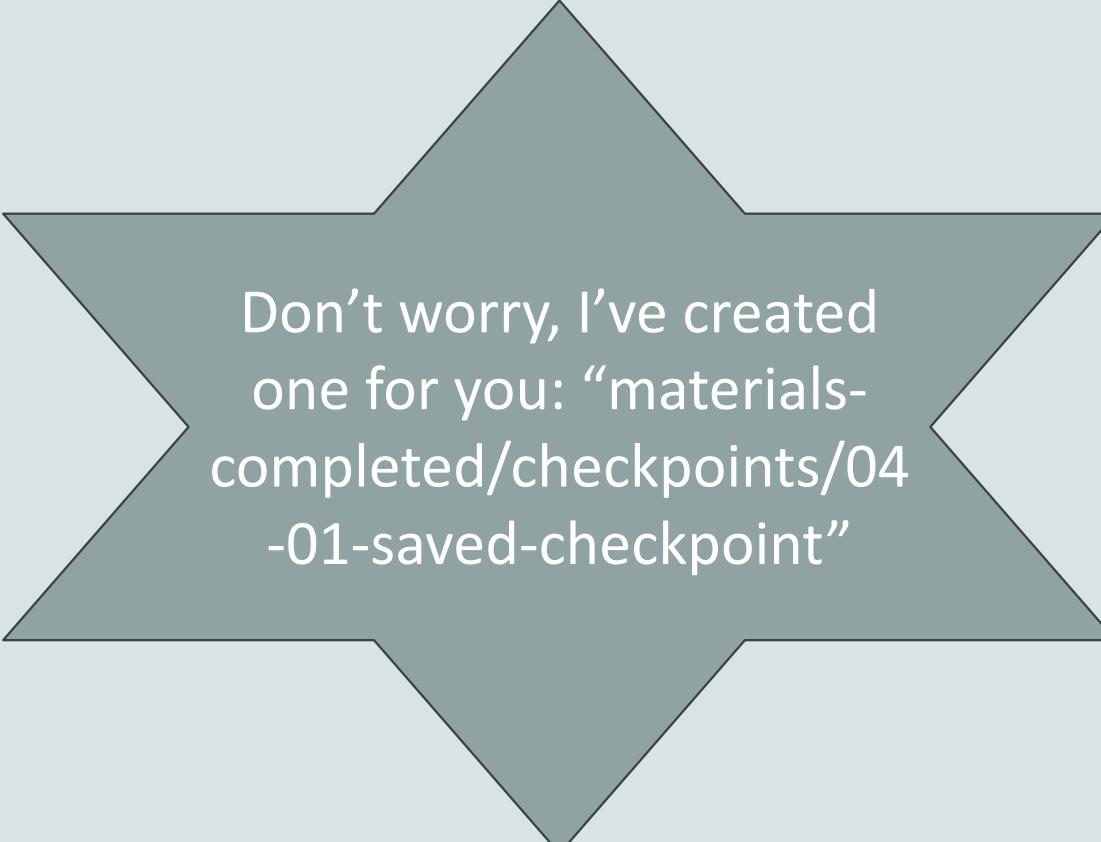
Saving a Checkpoint

```
def save_checkpoint(self, checkpoint_dir: Path) -> None:  
    """  
        This function will save the checkpoint to the specified directory.  
  
        :param checkpoint_dir: The path to the directory where the checkpoint  
        will be saved.  
    """  
    m5.checkpoint(str(checkpoint_dir))
```

This is a function definition in “src/gem5/python/simulate/simulator.py”. Using this info, append to the configuration file to make it create a checkpoint after the first `m5 exit`.

When done, run the script. It should take about 20 minutes.

Saving a Checkpoint



Don't worry, I've created
one for you: "materials-
completed/checkpoints/04
-01-saved-checkpoint"

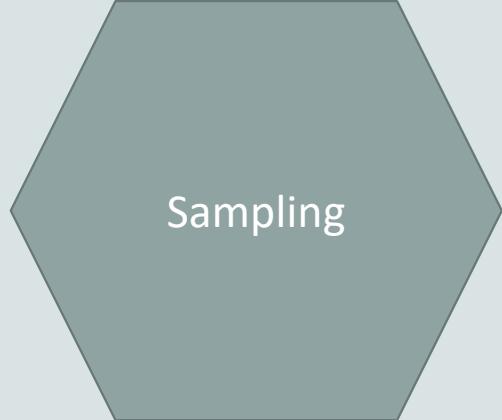
Restoring from a checkpoint

Open “materials/04-02-saving-a-checkpoint.py”.

Again, let’s go through this file. Note the differences.

This file is complete. Run it and see the restore in action.

Sampling!



Sampling is only simulating small parts of an execution then extrapolating data from there.

SimPoint and Looppoint are partially integrated into gem5.

There are some toy examples in
“configs/example/gem5_library/looppoints”.

Right now, a bit experimental.
Advanced users only!

You can now

Obtain,
compile, and
run gem5

Setup
simulations
using the gem5
stdlib

Understand
Event-based
Simulation

Use Exit Events
to return data
and alter
parameters at
specific points
in time

Understand SE
and FS mode

Reduce your
simulation
time with a set
of tricks to
lower and
increase
fidelity

What's next?

While we strive to provide a tool which allows for easy simulation of typical hardware setups via modular connections between components provided by the project...

Research and development requires creation of something new. We must therefore know how to create new components, SimObjects, and incorporate them into our designs.

We provide you with 99% of a simulated system so you can fuss over that 1% that's unique to your project

SimObjects at a glance

Model

C++ code in src/

Parameters

Python code in src/

In SimObject declaration file

Instance or configuration

A particular choice for the parameters

In standard library, your extensions, or python runscript

Adding a new SimObject

Step 1: Create a Python class (SimObject description file)

Step 2: Implement the C++

Step 3: Register the SimObject and C++ file

Step 4: (Re-)build gem5

Step 5: Create a config script

Adding a new SimObject

Create a new directory in “src” called “simobject-example”.

All the files will go there.

Step 1: Create a Python class

HelloObject.py

```
| from m5.params import *
| from m5.SimObject import SimObject
|
| class HelloObject(SimObject):
|     type = "HelloObject"
|     cxx_header = "simobject-example/hello_object.hh"
|     cxx_class = "gem5::HelloObject"
```

m5.params: Things like
MemorySize, Int, etc.

Import the objects we need

type: The C++ class name

cxx_class: The fully qualified
C++ class name

cxx_header: The filename for the
C++ header file

Step 2: Implement the C++ hello_object.hh

```
| #include "params/HelloObject.hh"  
| #include "sim/sim_object.hh"  
| namespace gem5{  
| class HelloObject : public SimObject  
| {  
|     public:  
|         HelloObject(const HelloObjectParams &p);  
|     };  
| } // namespace gem5
```

params/*.hh generated automatically. Comes from Python SimObject definition

Constructor has one parameter, the generated params object.
Must be a **const reference**

Step 2: Implement the C++

hello_object.cc

```
#include "simobject-example/hello_object.hh"
#include <iostream>
Namespace gem5 {
HelloObject::HelloObject(const HelloObjectParams &params)
    : SimObject(params)
{
    std::cout << "Hello World! From a SimObject!" << std::endl;
}
} //
```

Step 3: Register the SimObject and C++ file

SConscript

```
| Import('*')
| SimObject(HelloObject.py', sim_objects=[HelloObject'])
| Source(hello_object.cc')
```

Source(): Tell scons to compile this file (e.g., with g++).

sim_objects: The SimObjects declared in the file (could be more than 1)

Import: SConscript is just Python... but weird.

SimObject(): Says that this Python file contains a SimObject. Note: you can put pretty much any Python in here

Step 4: (Re-)build gem5

```
> scons build/ALL/gem5.opt -j`nproc`
```

Step 5: Create a config script

```
import m5
from m5.objects import *
root = Root(full_system=False)
root.hello = HelloObject()

m5.instantiate()
exit_event = m5.simulate()
print(f"Exiting @ tick {m5.curTick()} because"
      "{exit_event.getCause()}")
```

All simulations
require a **Root**

Instantiate the new object that
you created in the config file
(e.g., simple.py)

Simulate the system as
configured!

Instantiate all the SimObjects
(create the C++ instances)

> build/ALL/gem5.opt run-hello.py

...

Hello world! From a SimObject!

Adding Parameters and events to the SimObject

```
time_to_wait = Param.Latency("Time before firing the event")
number_of_fires = Param.Int(1, "Number of times to fire the event before"
                           "goodbye")
```

Add this to “HelloObject.py”.

This declares the parameters of the SimObject.

Adding Parameters and events to the SimObject

```
class HelloObject : public SimObject
{
    private:
        void processEvent();
        EventFunctionWrapper event;
        const std::string myName;
        const Tick latency;
        int timesLeft;

    public:
        HelloObject(const HelloObjectParams &p);
        void startup() override;
};
```

Update the “hello_object.hh” with the parameters and the event variables/functions.

The “processEvent” function will handle the event.

The “event” variable will wrap the “processEvent’ function.

The other variables store the stage of the object and the variables

Adding Parameters and events to the SimObject

```
HelloObject::HelloObject(const HelloObjectParams &params) :  
    SimObject(params),  
    event([this]{ processEvent(); }, name() + ".event"),  
    myName(params.name),  
    latency(params.time_to_wait),  
    timesLeft(params.number_of_fires)  
{  
    std::cout << "Created the HelloObject with name " << myName << std::endl;  
}
```

Now we updated "hello_object.cc" to set the variables and functions.

We've also changed the `std::cout` to output the Object name.

Adding Parameters and events to the SimObject

```
void
HelloObject::startup()
{
    schedule(event, latency);
}

void
HelloObject::processEvent()
{
    std::cout << "Hello world! Processing the event!" << std::endl;
    timesLeft--;

    if (timesLeft <= 0) {
        std::cout << "No more Hello Worlds left. Not scheduling another event!" << std::endl;
    }
    else {
        std::cout << timesLeft << " Hello Worlds left. Scheduling next event" << std::endl;
        schedule(event, curTick() + latency);
    }
}
```

Implement the `startup` and `processEvent` functions in "hello_object.cc"

The `startup` schedules the event before the simulation starts with `latency`.

The `processEvent` will continue to reschedule the event `timeLeft` times.



Adding Parameters and events to the SimObject

```
root.hello = HelloObject(time_to_wait = '2us')
```

Because `time_to_wait` does not have a default value we must set it in the config script.

Add this to your run script.

Adding Parameters and events to the SimObject

```
scons build/ALL/gem5.opt -j`nproc`  
./build/ALL/gem5.opt <your run script>
```

After, why don't you try running changing the default
'number_of_fires' parameter to something more interesting.

A little more to do in your own time

https://www.gem5.org/documentation/learning_gem5/part2/parameters/ contains an additional part of this tutorial which involves adding another SimObject: “GoodbyeObject” which schedules a Goodbye event when the HelloObject events cease scheduling

If you’re interested in SimObjects and creating your own this is worth doing..

Let's see how SimObject objects are configured, utilized and create new components



Let's do it
live! Try to
follow along!

(an example can be found in “materials-completed/07-our-silly-cache”)

gem5 Resources

A resources is something that runs on a simulated system.

While the underlying simulated hardware remains the same, what resources are used in the simulation impact what the simulated system runs and how the simulator runs it.

Typically, this is software, disk images, or data to be loaded into the simulated system's memory/storage.

It may also be things like “checkpoints” for the simulation to use, or sampling data.

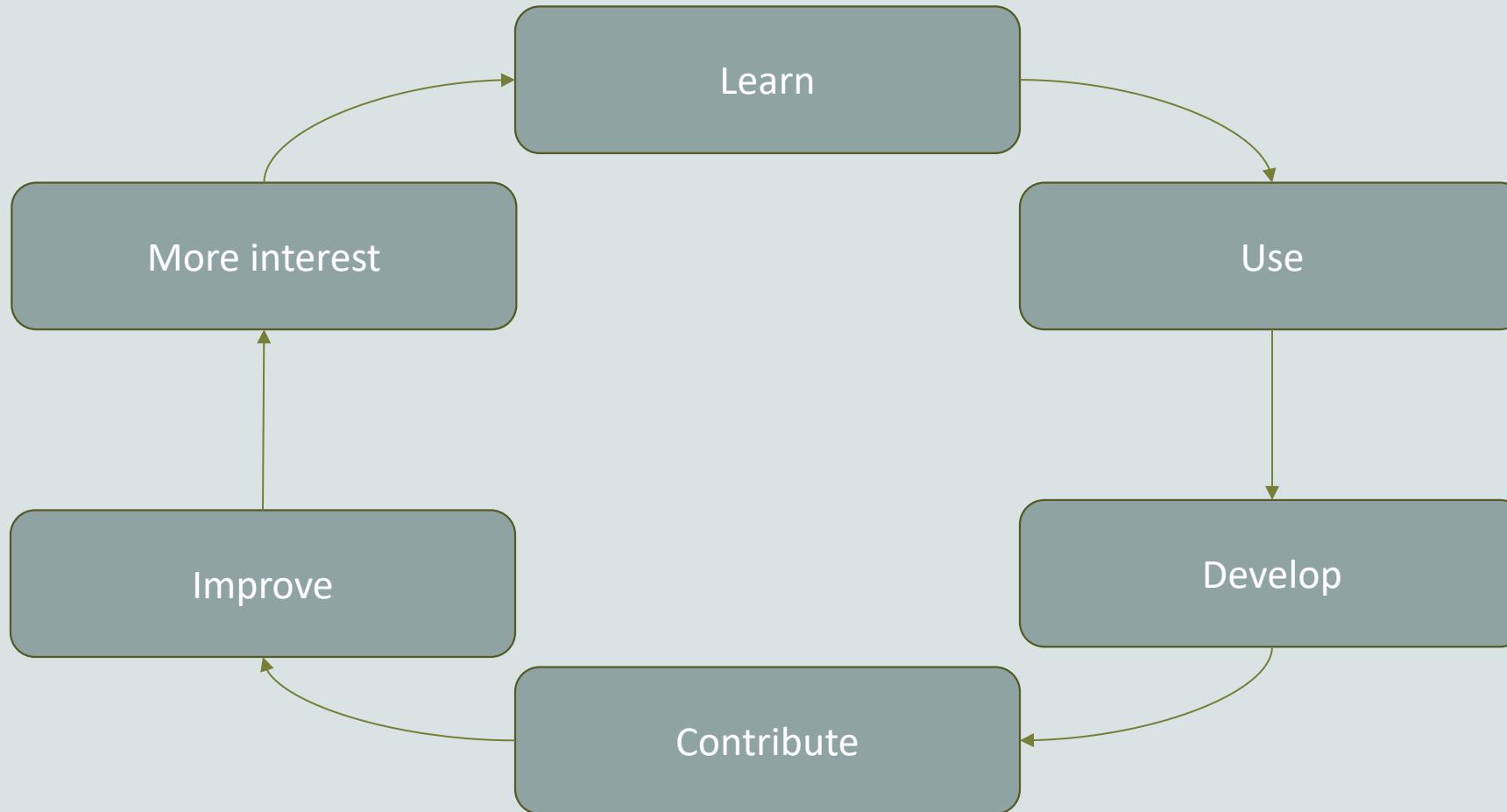
They are files, directories, programs, and generally thing that doesn't impact a simulated configuration but does impact it's running.

Contributing to gem5

You've learned a thing or two about gem5, why not work to make it better?



Our Strategy



Why should I contribute to gem5?

You're Nice!

- You've found a bug and have a fix.
- You've developed something useful and want to share it.

Fame!

- Get yourself known in the project.
- Good PR for your research to have it incorporated in gem5.

Fortune!

- Looks good on your CV.
- Companies contribute to gem5 all the time.

“I’m scared”

Understandable...

Very few patches get in straight away. Most patches are only accepted after requests for changes.

We try our best to keep feedback as constructive as possible (don’t take it personally!).

The purpose of this session is to make it less scary!

What can I contribute?

Your own changes (bug fixes are very welcome!)

Check the GitHub Issue Tracker:

<https://github.com/gem5/gem5/issues>

What can I contribute?

Some stuff we're always needing more of:

Tests

Incorporating Syscalls for SE mode

Unimplemented ISA instructions/extensions

Useful stdlib components

Useful gem5 resources

Updating documentation on the gem5 website

Even fixing typos is helpful!

What can't I contribute?

1. Anything that'll burden the community with too much maintenance overhead.

Yeah, you've developed something nice for us, but it's big and complex: are you going to stick around to help us maintain it? Is it engineered for that to be easy?

- 3..It's dangerous

You've changed a lot of code and haven't proven you've not yet broken anything. Tests are required at a minimum.

2. Something overly niche and lacks general applicability

The component you made to carry out your research may be interesting to you but adding it to the codebase may just be bloat to most users: Consider sharing such things on your own git repos.

3. It doesn't conform to our standards

The code appears fine, but you've not conformed to our style guide.

Forking and cloning

Step 1: Go to <https://github.com/gem5/gem5>

Step 2: Fork the repo

Step 3: Clone the forked repo

Where do I make changes?

```
>git switch develop  
>git switch -c my-change
```

Making changes: CPP

Full style guide here:

https://www.gem5.org/documentation/general_docs/development/coding_style/

High-level overview: <https://www.gem5.org/contributing#making-modifications>

Doxxygen is highly recommended

<http://doxygen.gem5.org>

Making changes: Python

```
> pip install black  
> black <python file>
```

For variable/method/etc. Naming conventions please follow the PEP 8 naming convention recommendations: <https://peps.python.org/pep-0008/#naming-conventions>

While we try to enforce naming conventions across the gem5 project, we are aware there are instances where they are not.

In such cases please **follow the convention of the code you are modifying**.

The biggest gotchas!

- Whitespace at the end of a line.
- Indentation not 4 space characters (please, no tabs)
- Lines too long (for CPP, no more than 79 characters!)

When in doubt, follow the style around you!

We have a style checker which should stop you committing if you've done something wrong, but it's not perfect and can be side-stepped.

Ensure pre-commit is installed

```
> ./util/pre-commit-install.sh
```

Pre-commit ensures when you are about to commit a change a series of checks are run on your code to ensure it conforms to our style guide

Using git

```
> git add <files to add>
```

```
> git commit
```

Commit message rules

We have some unique rules for gem5:

1. The header must lead with tags (see MAINTAINERS.yaml for a list of tags).
2. Headers should be clear, short descriptions of what a patch will do.
3. Headers should be **no longer than 65 characters**
4. A blank line separates the header and the patch description.
5. Descriptions can span multiple paragraphs, but lines **should not exceed 72 characters** (this is lax rule, it's acceptable to exceed this if you're quoting code, or including a URL).

View the Git Log

```
> git log
```

Example commit message

```
Author: Jason Lowe-Power <jason@lowepower.com>
```

```
Date: Wed Dec 20 15:25:17 2023 -0800
```

```
mem-ruby,configs: Enable Ruby with NULL build
```

```
After removing `get_runtime_isa`, the `send_evicts` function in the ruby  
configs assumes that there is an ISA built. This change short-circuits  
that logic if the current build is the NULL (none) ISA.
```

```
Change-Id: I75fefe3d70649b636b983c4d2145c63a9e1342f7
```

```
Signed-off-by: Jason Lowe-Power <jason@lowepower.com>
```

How do I push?

```
> git push
```

This pushes to your forked repo



Create a pull request

The screenshot shows a GitHub repository page for the forked repository "gem5". The repository has 27 branches and 27 tags. A modal dialog titled "Switch branches/tags" is open, showing a search bar and a list of branches. The branch "test-runner-kvm" is selected and highlighted with a blue border. The main repository page lists several commits from the "test-runner-kvm" branch, including updates to Kconfig build options, GPU support, and DRAMSys enforcement.

Commit Message	Date	Commits
Merge branch 'develop' into test-runner-kvm	6fe60fa · 2 months ago	21,215 Commits
scons: Update the Kconfig build options	4 months ago	
misc: Run pre-commit run --all-files	4 months ago	
gpu-compute: Support for MI200 GPU model (gem5#733)	2 months ago	
util-docker: Enforce cmake version >=3.24 for DRAMSys (...)	4 months ago	
misc: create C declarations for the _addr and _semi m5ops	4 years ago	

Select the branch you just pushed



Create a pull request

The screenshot shows a GitHub repository page for the forked repository `gem5`. The repository has 27 branches and 27 tags. A comparison summary indicates that the branch is 77 commits ahead of `gem5/gem5:stable`. A tooltip over the 'Contribute' button provides instructions to open a pull request for upstream contribution. A green button labeled 'Open pull request' is visible.

This branch is 77 commits ahead of `gem5/gem5:stable`.

BobbyRBruce Merge branch 'develop' into test-runner-kvm

File	Commit Message	Time Ago
.github	Merge branch 'dev...	2 months ago
build_opts	scons: Update the...	4 months ago
build_tools	misc: Run pre-comm...	4 months ago
configs	gpu-compute: Sup...	2 months ago
ext	util-docker: Enforce cmake version >=3.24 for DRAMSys (...)	4 months ago

This branch is 77 commits ahead of `gem5/gem5:stable`. Open a pull request to contribute your changes upstream.

Open pull request

Click “Open Pull Request” under “Contribute”



Create a pull request

base repository: gem5/gem5 ▾ base: stable ▾ ... head repository: BobbyRBruce/gem5 ▾ compare: test-runner-kvm ▾

✓ Able to merge. These branches can be merged into the current branch.

Find a branch

Branches Tags

✓ stable default

dependabot/pip/develop/pre-commit-3.6.2

dependabot/pip/develop/tqdm-4.66.2

dependabot/pip/util/gem5-resources-ma...

develop

feature-gui

Add a title

Test runner kvm

Add a description

Write Preview

Add your description here...

Reviewers

Suggestions

abmer

At least 1 reviewer is required for a pull request.

Assignees

No one—assign later

Labels

None yet

Fill out the form. Of note: Change the base branch to “develop”.
Once filled, click “Create Pull Request”



Pull Request waiting for testing and review

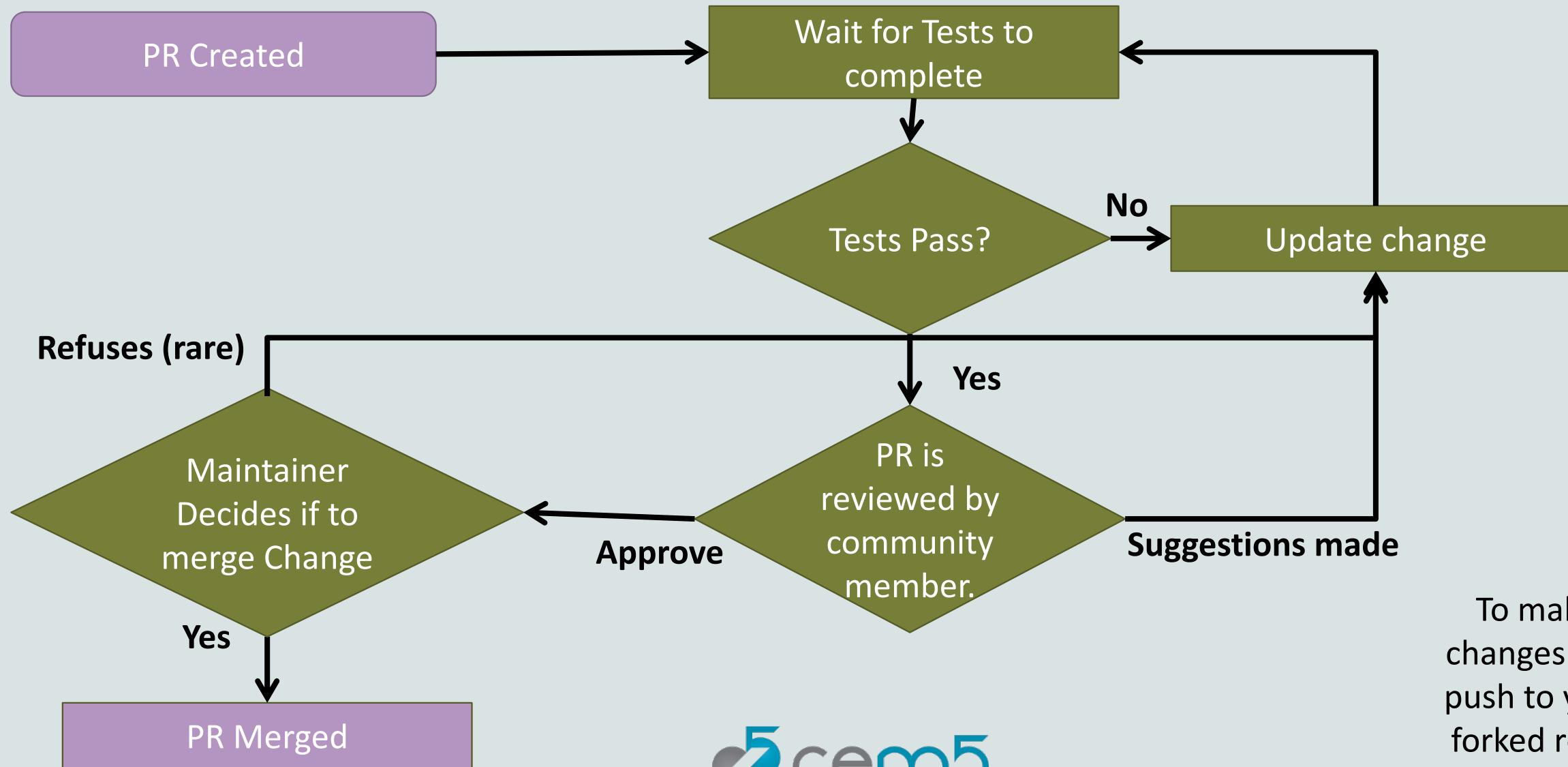
<https://github.com/gem5/gem5/pulls>

The screenshot shows the GitHub interface for the gem5 repository. The top navigation bar includes links for Code, Issues (53), Pull requests (28, highlighted in red), Discussions, Actions, Security (3), Insights, and Settings. A search bar at the top right contains the placeholder "Type ⌘ to search". Below the navigation, there are filters: "Filters" dropdown, a search bar with the query "is:pr is:open", a "Labels" button (67), a "Milestones" button (0), and a "New pull request" button. The main area displays a list of 28 open pull requests. Each pull request entry includes a checkbox, the title, the number of reviews (indicated by a green checkmark or a red minus sign), the author, the label, the project, the milestone, and the assignee. The first few pull requests are as follows:

- misc: bump tqdm from 4.66.1 to 4.66.2 (Review required)
- misc: bump pre-commit from 3.6.0 to 3.6.2 (Review required)
- tests: Update tests to use specific resource versions (Review required)
- arch-riscv,sim: m5ops argument / return fix for 32 bit RISC-V (Review required)
- sim-se: Implement statx system call for Linux x86-64 (Changes requested)



PR Review Process



Testing overview

Execution time



CI Tests

Daily Tests

Compiler Tests (Run Weekly)

Weekly Tests

The most direct are the “CI Tests”, you cannot merge your change into develop without these passing on your PR.

The rest run either daily or weekly. It is therefore possible your PR breaks gem5 but there's a delay in us finding out (so keep an eye on these tests).

Badges are shown on the repo main page:

<https://github.com/gem5/gem5?tab=readme-ov-file#testing-status>

What about the other gem5 repos?

gem5 Resources

<https://github.com/gem5/gem5-resources>

The Sources for the gem5 Resources

Build atop “stable” to make changes for the current release.

Built atop “develop” to make changes for the upcoming release.

gem5 Website

<https://github.com/gem5/website>

The www.gem5.org sources.

Changes made to the “stable” branch are live.

Changes made to “develop” will be merged into stable on the next gem5 release.

Neither of these are held up to the same standards as the gem5 repo but changes to them are reviewed.



Some useful resources

<https://www.gem5.org/contributing>

CONTRIBUTING.md in the gem5 directory

Sometimes using git is the biggest hurdle:

- <https://git-scm.com/book/en/v2> : The git book
- https://dev.to/milu_franz/git-explained-the-basics-igc : I think this is a good tutorial but is very GitHub-centric (we don't use GitHub for gem5). Still, going through it would be beneficial.
- https://wiki.spheredev.org/index.php/Git_for_the_lazy : Does a quick run through of the basic Git commands. Can be good for reference.
- <http://marklodato.github.io/visual-git-guide/index-en.html>: A bit more complex but tries to introduce the git data structures involved in git
- <https://towardsdatascience.com/git-help-all-2d0bb0c31483>: Another resource outlining both the commands and explaining how git works.

Caveats

gem5 is a tool, not a panacea

Most models are not validated against
“real” hardware

“All models are wrong but some are useful”

See “Architectural Simulators Considered Harmful” by Nowatzki et al. (2015).

There are bugs!



Bobby's Advice

Learn git. By that, I mean beyond “git add” and “git commit”.

Get comfortable with Object Oriented design. The gem5 codebase depends heavily on it. Learn it and incorporate it in your work.

Don't modify, extend! Hacking what's already there will cause problems. Create new SimObjects, components, scripts as needed.

Understand the data you need before trying to make gem5 go faster. SE mode, checkpoints, faster CPU models etc. are tempting but they have trade-offs.

Do not configure your system via the command line: Configurations exist in your configuration file and associated components, SimObjects, etc.



References

- Martin et al. 2005. **Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset.** ACM SIGARCH Computer Architecture News.
<https://doi.org/10.1145/1105734.1105747>
- Binkert et al. 2006. **The M5 simulator: Modeling Networked Systems.** IEEE Micro.
<https://doi.org/10.1109/MM.2006.82>
- Binkert, et al. 2011. **The gem5 simulator.** *SIGARCH Comput. Archit. News* 39
<http://dx.doi.org/10.1145/2024716.2024718>
- Lowe-Power et al 2021. **The gem5 Simulator: Version 20.0+.** ArXiv Preprint ArXiv:2007.03152, 2021. <https://doi.org/10.48550/arXiv.2007.03152>
- Nowatzki et al. 2015. **Architectural simulators considered harmful.** IEEE Micro.
<https://doi.org/10.1109/MM.2015.74>