The gem5 Standard Library

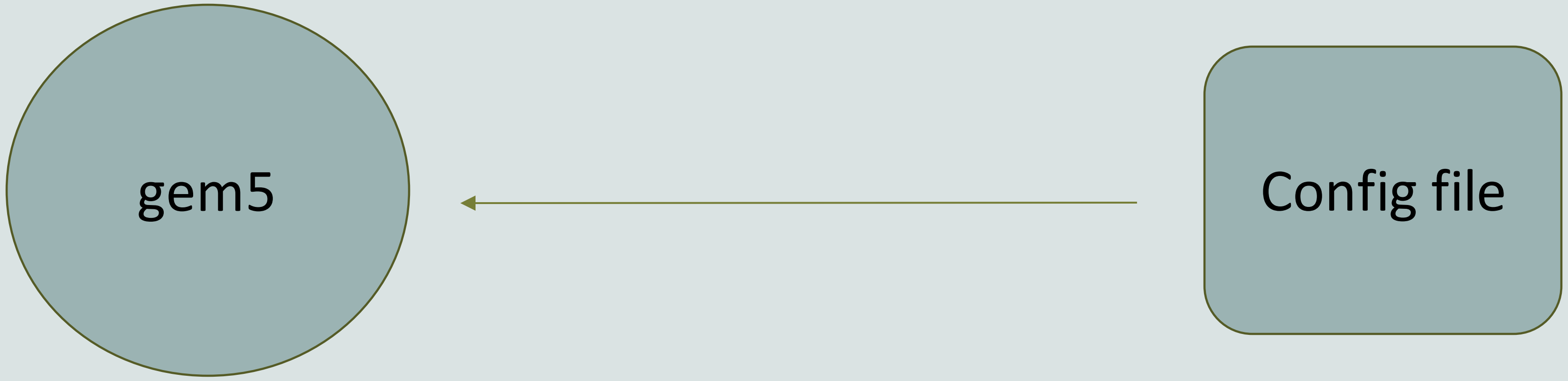
A presentation by

Bobby R. Bruce

UCDAVIS COMPUTER SCIENCE

DArchR DAVIS ARCHITECTURE RESEARCH

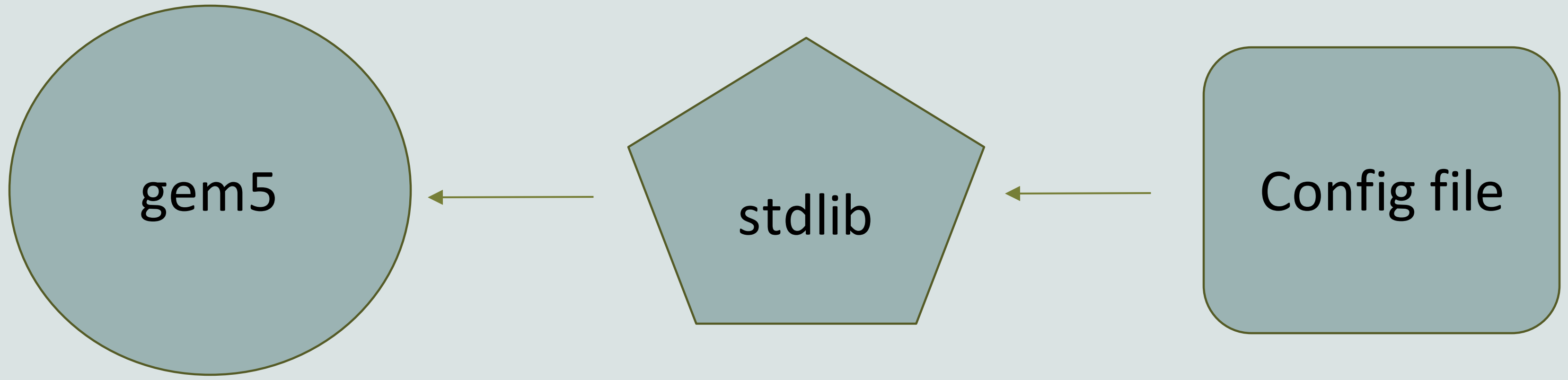
What is the standard library for?



When done without the library you must define *every part* of your simulation.

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

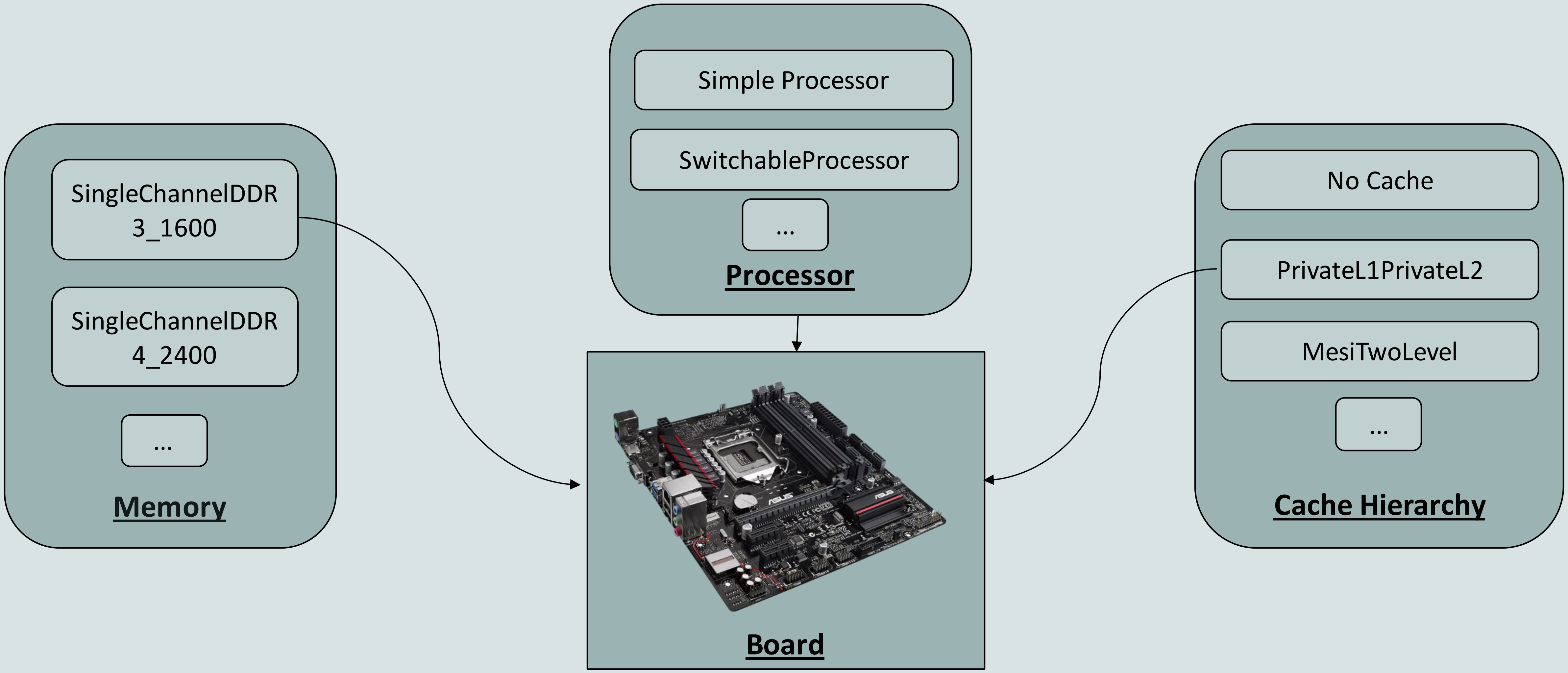
What is the standard library for?



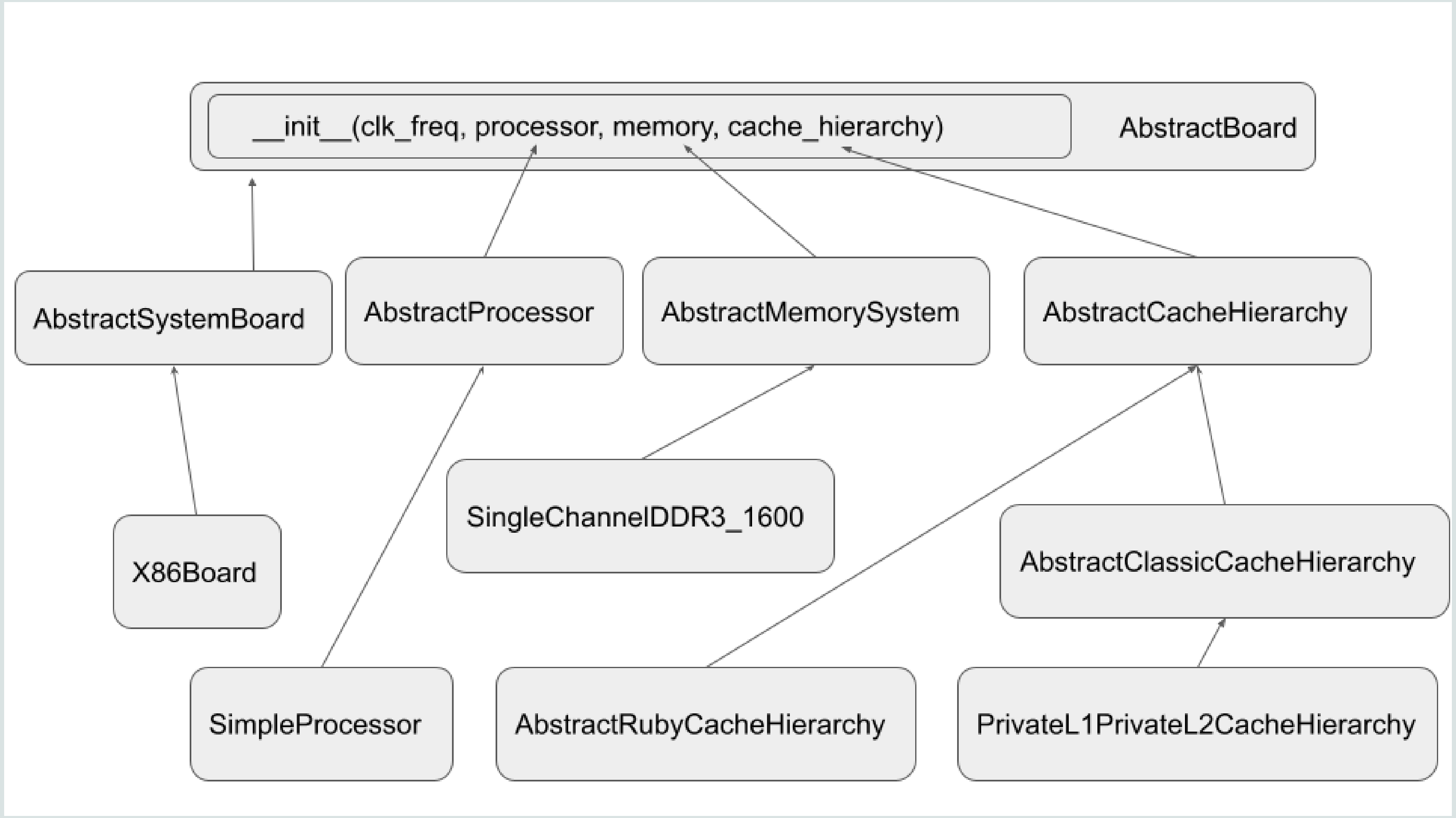
The stdlib is a library which allows for users to quickly create systems with pre-built components.

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.

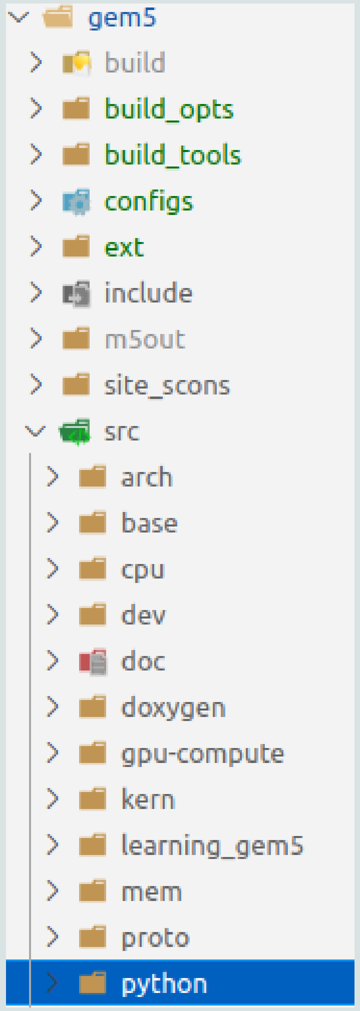
The stdlib modular metaphor



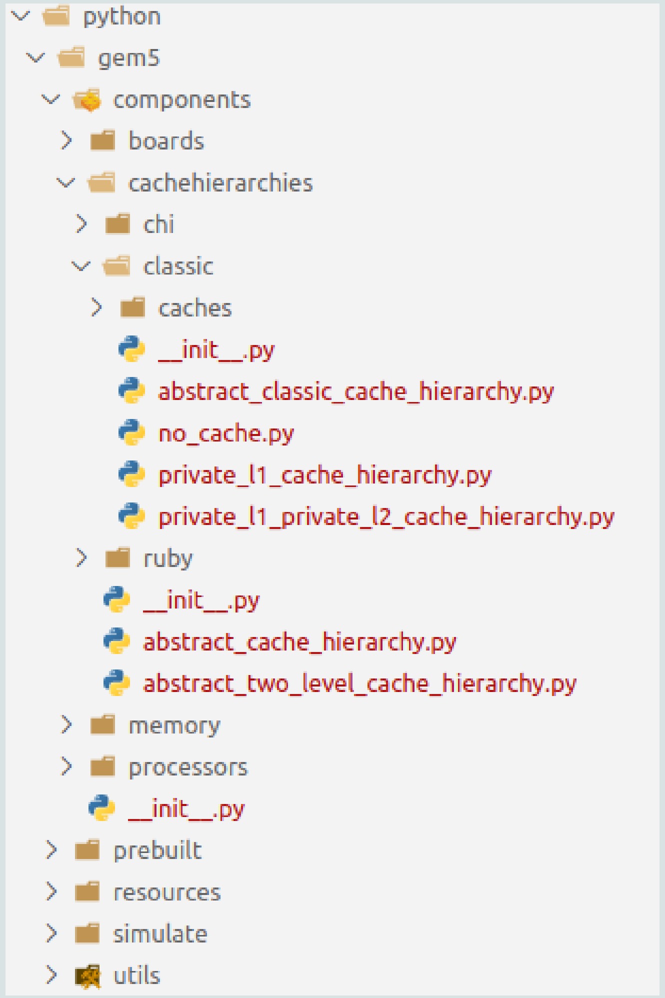
The modular architecture


__init__(clk_freq, processor, memory, cache_hierarchy)
AbstractBoard
AbstractSystemBoard
AbstractProcessor
AbstractMemorySystem
AbstractCacheHierarchy
X86Board
SingleChannelDDR3_1600
AbstractClassicCacheHierarchy
SimpleProcessor
AbstractRubyCacheHierarchy
PrivateL1PrivateL2CacheHierarchy

Where to find stuff: The directory structure

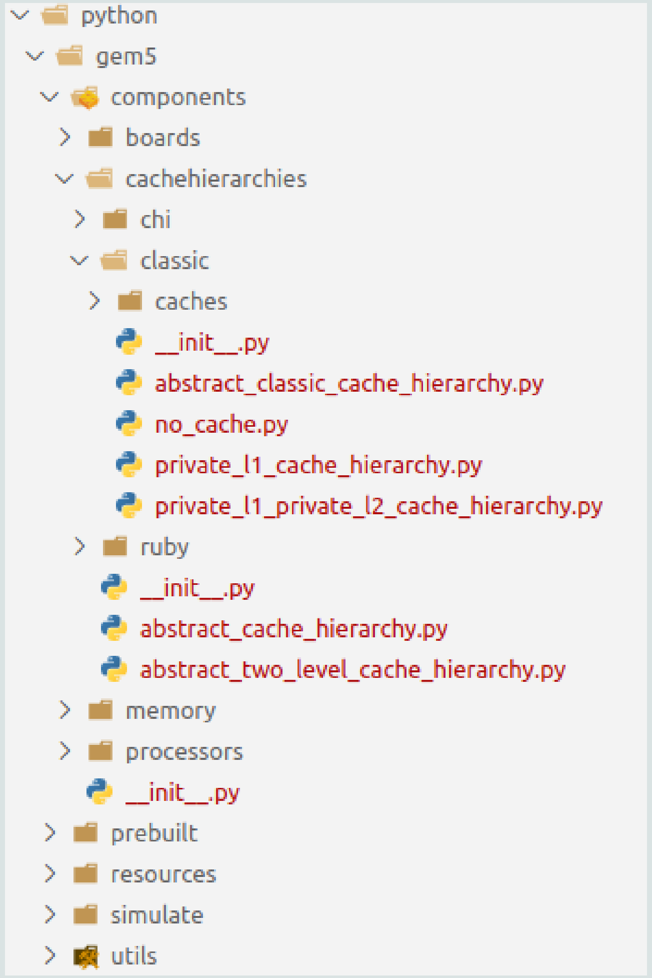


```
gem5
  build
  build_opts
  build_tools
  configs
  ext
  include
  m5out
  site_scons
  src
    arch
    base
    cpu
    dev
    doc
    doxygen
    gpu-compute
    kern
    learning_gem5
    mem
    proto
    python
```

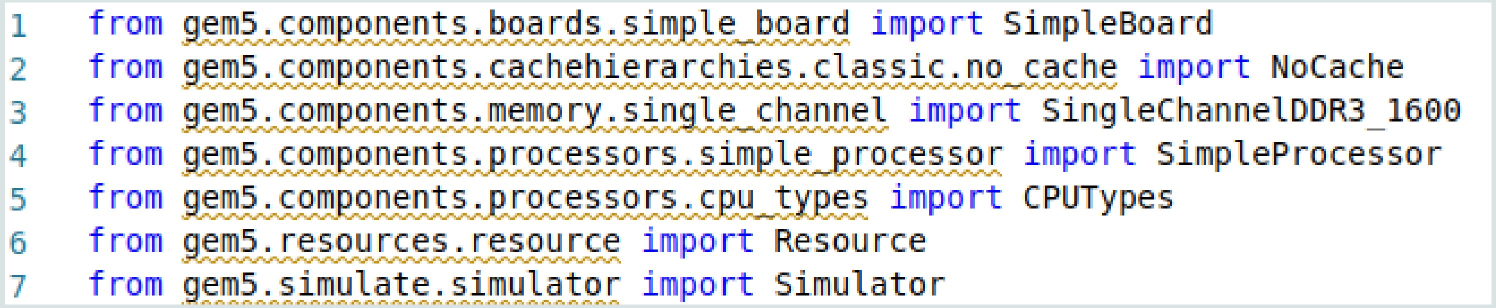


```
python
  gem5
    components
      boards
      cachehierarchies
        chi
        classic
          caches
          __init__.py
          abstract_classic_cache_hierarchy.py
          no_cache.py
          private_l1_cache_hierarchy.py
          private_l1_private_l2_cache_hierarchy.py
        ruby
        __init__.py
        abstract_cache_hierarchy.py
        abstract_two_level_cache_hierarchy.py
      memory
      processors
      __init__.py
    prebuilt
    resources
    simulate
    utils
```

Where to find stuff : Importing in a script



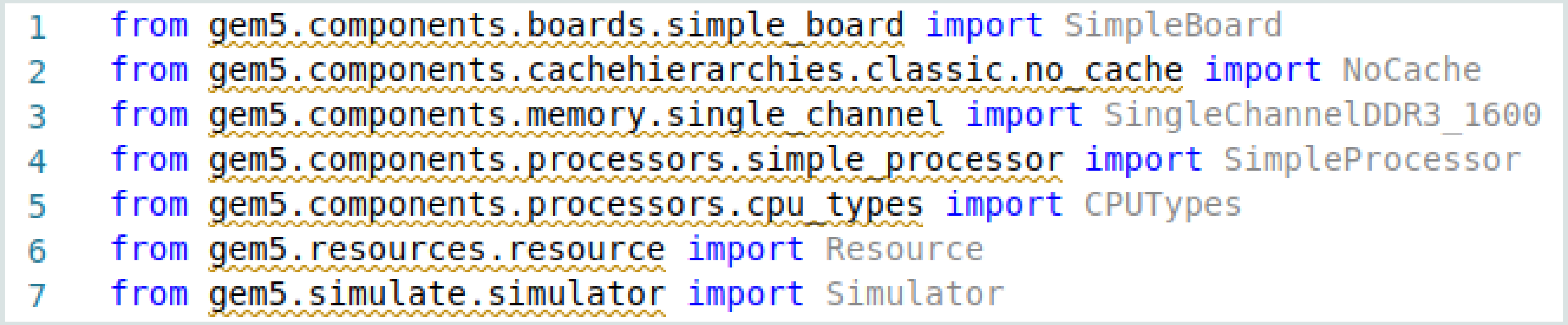
```
python
  gem5
    components
      boards
      cachehierarchies
        chi
        classic
          caches
          __init__.py
          abstract_classic_cache_hierarchy.py
          no_cache.py
          private_l1_cache_hierarchy.py
          private_l1_private_l2_cache_hierarchy.py
        ruby
        __init__.py
        abstract_cache_hierarchy.py
        abstract_two_level_cache_hierarchy.py
      memory
      processors
      __init__.py
    prebuilt
    resources
    simulate
    utils
```



```python
from gem5.components.boards.simple_board import SimpleBoard
from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.simple_processor import SimpleProcessor
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator
```

Getting started: Creating a "Hello World" in the stdlib

materials/using-gem5/02-stdlib/hello-world.py

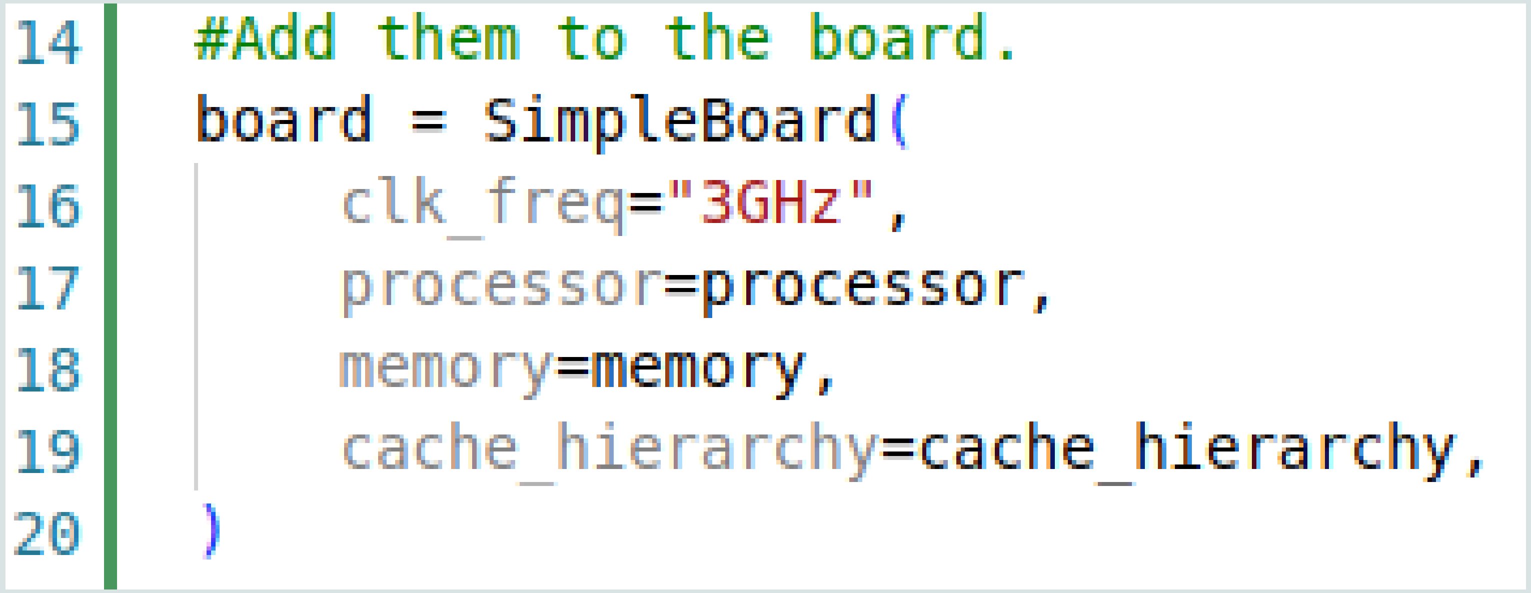


```python
from gem5.components.boards.simple_board import SimpleBoard
from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.simple_processor import SimpleProcessor
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator
```

Getting started: Creating a "Hello World" in the stdlib

```python
# Obtain the components.
cache_hierarchy = NoCache()
memory = SingleChannelDDR3_1600("1GiB")
processor = SimpleProcessor(cpu_type=CPUTypes.TIMING, num_cores=1)
```

Getting started: Creating a "Hello World" in the stdlib



```
#Add them to the board.
board = SimpleBoard(
    clk_freq="3GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)
```

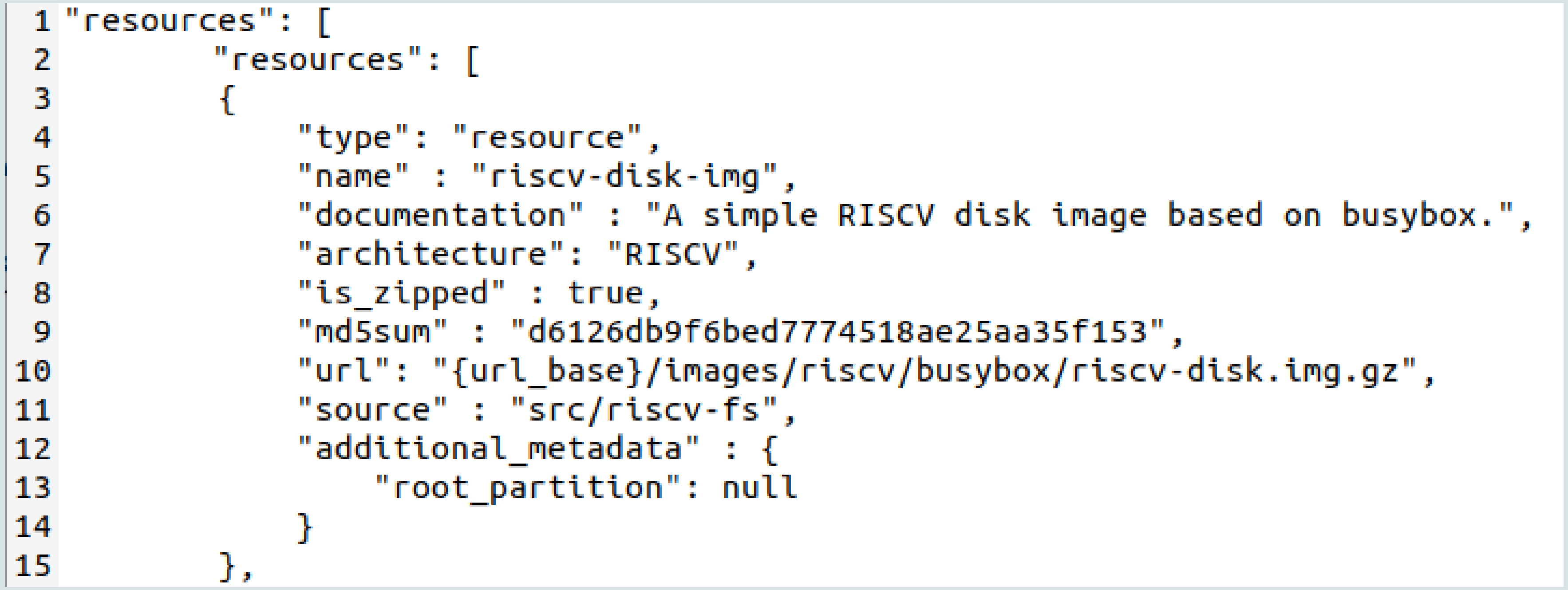
gem5

gem5 Resources

- gem5 resources is a repository providing resources that are known to be compatible with gem5.
- These resources are not necessary for the compilation or running gem5 but may aid users in running simulations. E.g.: disk images, kernels, applications, cross-compilers, etc.
- Resources are held on gem5's Google Cloud Bucket, and sources for these resources are found at: https://gem5.googlesource.com/public/gem5-resources/
- The stdlib can be used to automatically obtain and use these resources.
- https://resources.gem5.org/resources.json

Looking up gem5 Resources

https://resources.gem5.org/resources.json

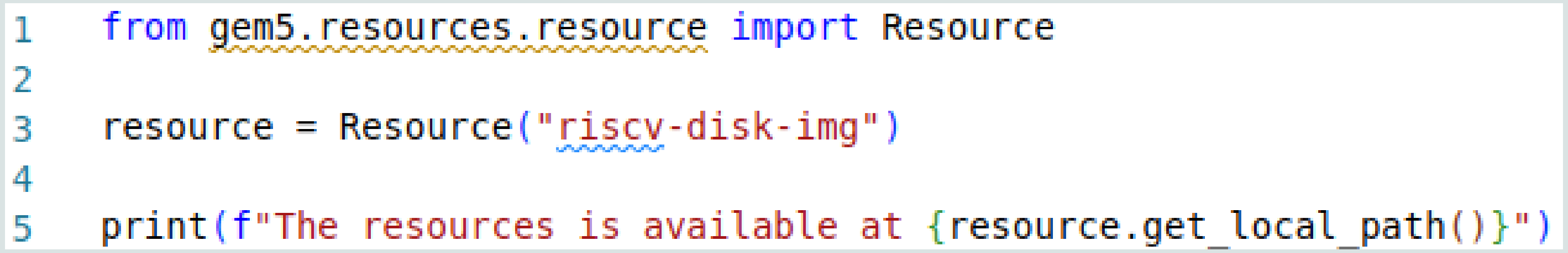


```
"resources": [
        "resources": [
        {
            "type": "resource",
            "name" : "riscv-disk-img",
            "documentation" : "A simple RISCV disk image based on busybox.",
            "architecture": "RISCV",
            "is_zipped" : true,
            "md5sum" : "d6126db9f6bed7774518ae25aa35f153",
            "url": "{url_base}/images/riscv/busybox/riscv-disk.img.gz",
            "source" : "src/riscv-fs",
            "additional_metadata" : {
                "root_partition": null
            }
        },
```

This is all machine-reachable for now. We're working on a web-portal.

Obtaining Resources in the stdlib

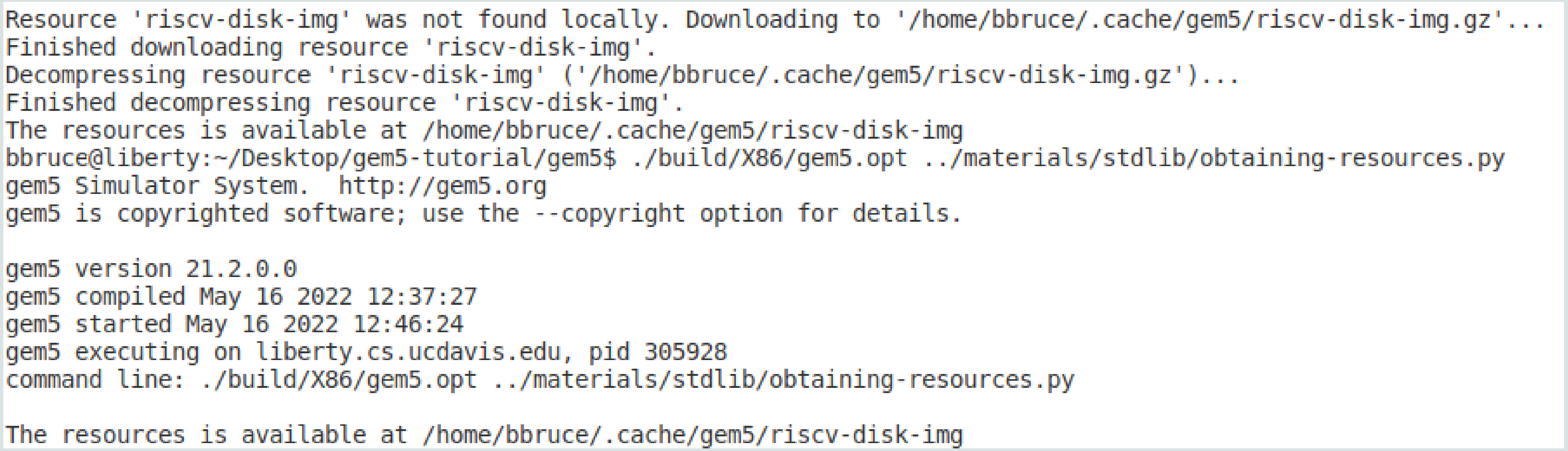
materials/using-gem5/02-stdlib/obtaining-resources.py



```python
from gem5.resources.resource import Resource

resource = Resource("riscv-disk-img")

print(f"The resources is available at {resource.get_local_path()}")
```

```
> gem5-x86 materials/using-gem5/02-stdlib/obtaining-resources.py
```

Obtaining Resources in the stdlib



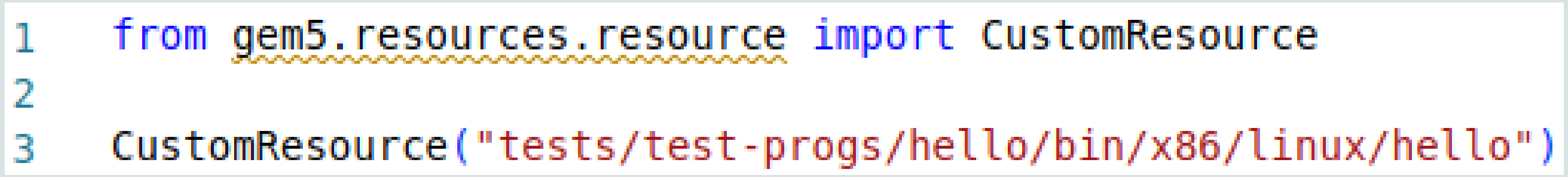
```
Resource 'riscv-disk-img' was not found locally. Downloading to '/home/bbruce/.cache/gem5/riscv-disk-img.gz'...
Finished downloading resource 'riscv-disk-img'.
Decompressing resource 'riscv-disk-img' ('/home/bbruce/.cache/gem5/riscv-disk-img.gz')...
Finished decompressing resource 'riscv-disk-img'.
The resources is available at /home/bbruce/.cache/gem5/riscv-disk-img
bbruce@liberty:~/Desktop/gem5-tutorial/gem5$ ./build/X86/gem5.opt ../materials/stdlib/obtaining-resources.py
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 21.2.0.0
gem5 compiled May 16 2022 12:37:27
gem5 started May 16 2022 12:46:24
gem5 executing on liberty.cs.ucdavis.edu, pid 305928
command line: ./build/X86/gem5.opt ../materials/stdlib/obtaining-resources.py

The resources is available at /home/bbruce/.cache/gem5/riscv-disk-img
```

The stdlib will use the cached resources if already downloaded.
Run the script twice and see for yourself.

Using a Custom Resource

You don't need to use the gem5 resources

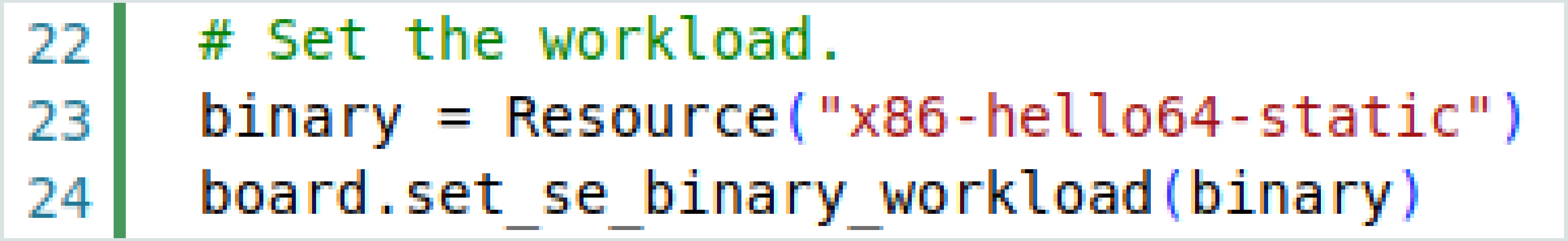
You can specify a local resources (e.g., your own disk image)



```python
from gem5.resources.resource import CustomResource

CustomResource("tests/test-progs/hello/bin/x86/linux/hello")
```

Getting started: Creating a "Hello World" in the stdlib

Back to "materials/using-gem5/02-stdlib/hello-world.py", add the following:

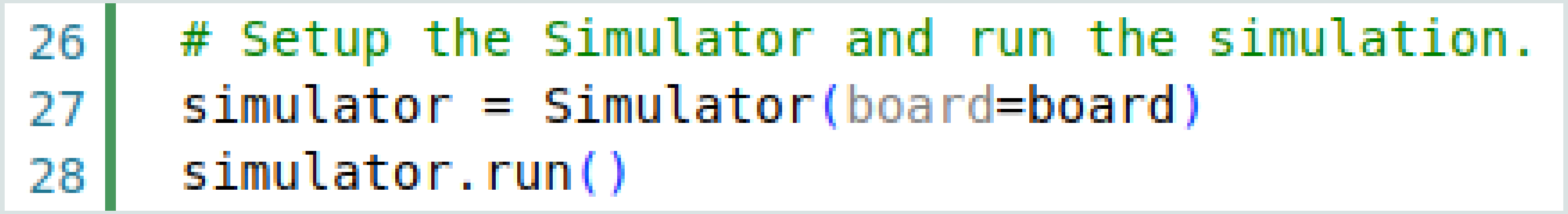


```python
# Set the workload.
binary = Resource("x86-hello64-static")
board.set_se_binary_workload(binary)
```

`set_se_binary_workload` is used to run a board in Syscall Emulation mode, with a single binary

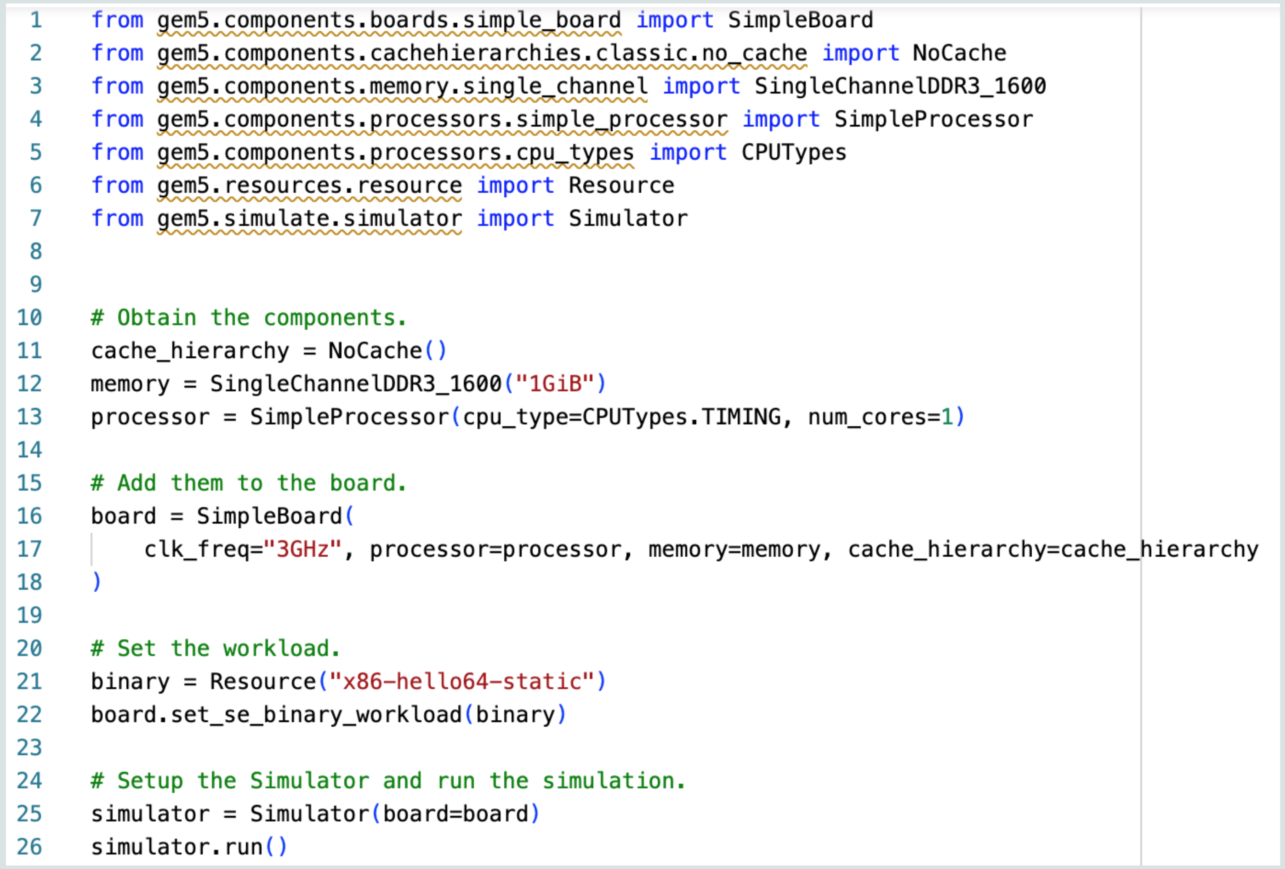
Getting started: Creating a "Hello World" in the stdlib

Append the following:



```python
# Setup the Simulator and run the simulation.
simulator = Simulator(board=board)
simulator.run()
```

Getting started: Creating a "Hello World" in the stdlib

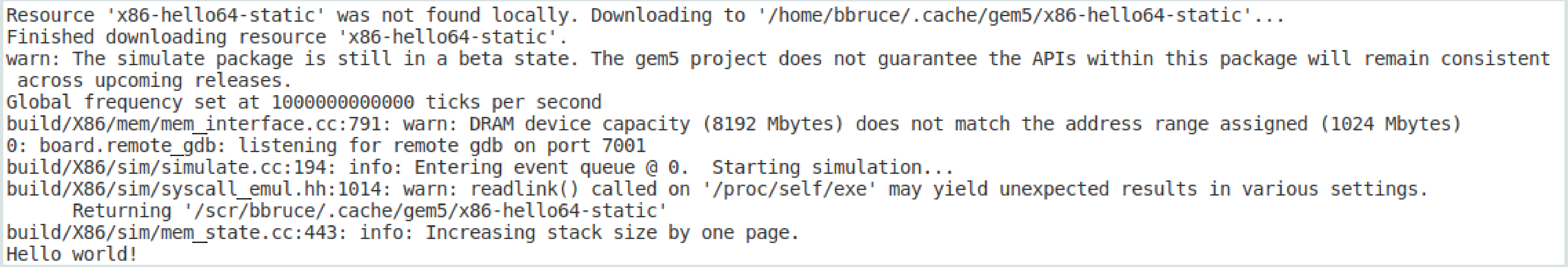


```python
from gem5.components.boards.simple_board import SimpleBoard
from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.simple_processor import SimpleProcessor
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator


# Obtain the components.
cache_hierarchy = NoCache()
memory = SingleChannelDDR3_1600("1GiB")
processor = SimpleProcessor(cpu_type=CPUTypes.TIMING, num_cores=1)

# Add them to the board.
board = SimpleBoard(
    clk_freq="3GHz", processor=processor, memory=memory, cache_hierarchy=cache_hierarchy
)

# Set the workload.
binary = Resource("x86-hello64-static")
board.set_se_binary_workload(binary)

# Setup the Simulator and run the simulation.
simulator = Simulator(board=board)
simulator.run()
```

Getting started: Creating a "Hello World" in the stdlib

Save the file!!!

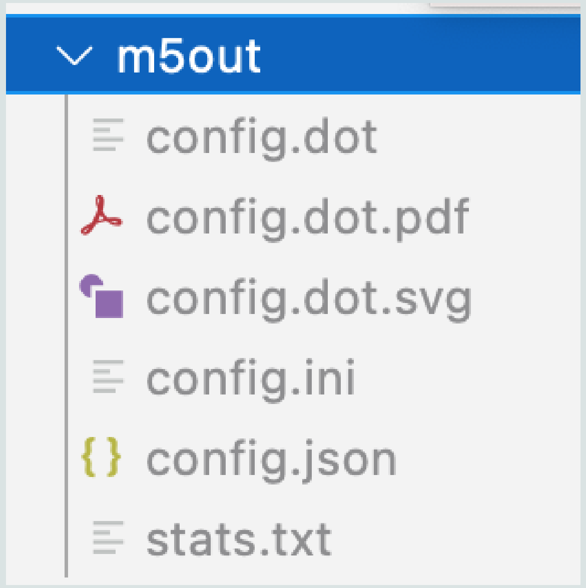
```
> gem5-x86 materials/using-gem5/02-stdlib/hello-world.py
```



```
Resource 'x86-hello64-static' was not found locally. Downloading to '/home/bbruce/.cache/gem5/x86-hello64-static'...
Finished downloading resource 'x86-hello64-static'.
warn: The simulate package is still in a beta state. The gem5 project does not guarantee the APIs within this package will remain consistent
 across upcoming releases.
Global frequency set at 1000000000000 ticks per second
build/X86/mem/mem_interface.cc:791: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (1024 Mbytes)
0: board.remote_gdb: listening for remote gdb on port 7001
build/X86/sim/simulate.cc:194: info: Entering event queue @ 0.  Starting simulation...
build/X86/sim/syscall_emul.hh:1014: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
      Returning '/scr/bbruce/.cache/gem5/x86-hello64-static'
build/X86/sim/mem_state.cc:443: info: Increasing stack size by one page.
Hello world!
```

More detailed output

Look into the more the ”gem5/m5out” directory



```
m5out
  config.dot
  config.dot.pdf
  config.dot.svg
  config.ini
  config.json
  stats.txt
```

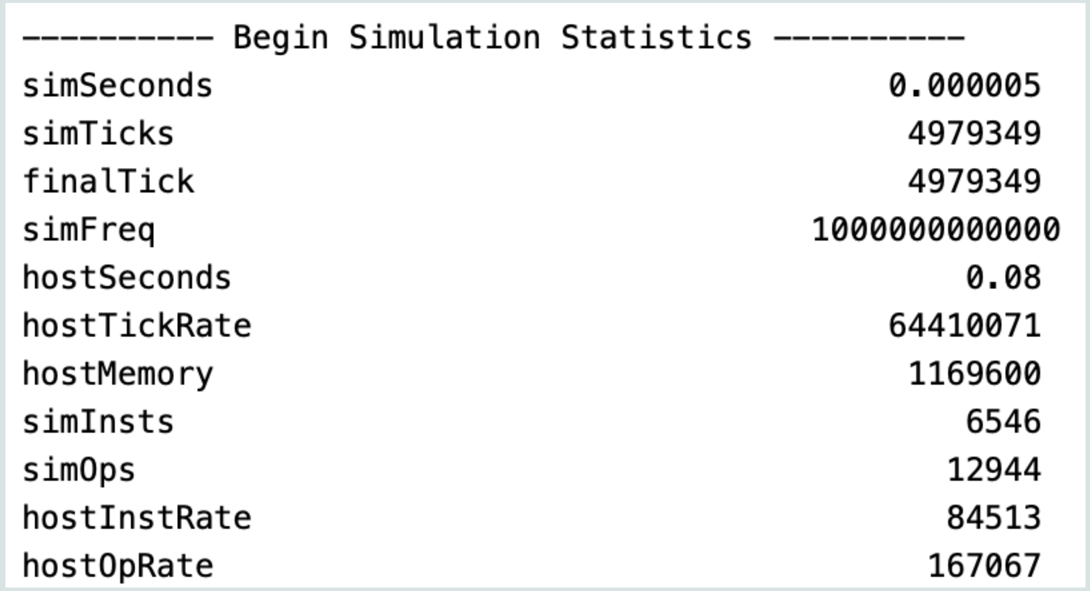
- The “config” files detail your system configuration (various formats, ”config.ini” most human-readable.

- The stats.txt shows the various simulation statistics.

- In Full-System simulations the terminal output can be found in this directory.

More detailed output

Look into the more the ”gem5/m5out/stats.txt” file



```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.000005
simTicks                                      4979349
finalTick                                     4979349
simFreq                                 1000000000000
hostSeconds                                      0.08
hostTickRate                                 64410071
hostMemory                                    1169600
simInsts                                         6546
simOps                                          12944
hostInstRate                                    84513
hostOpRate                                     167067
```

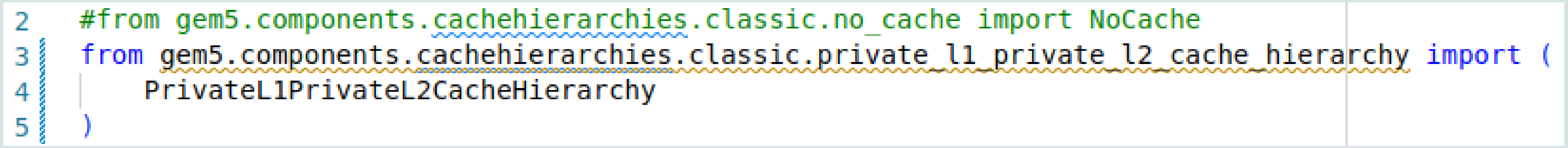
Extending our design

Remember: gem5 is modular!

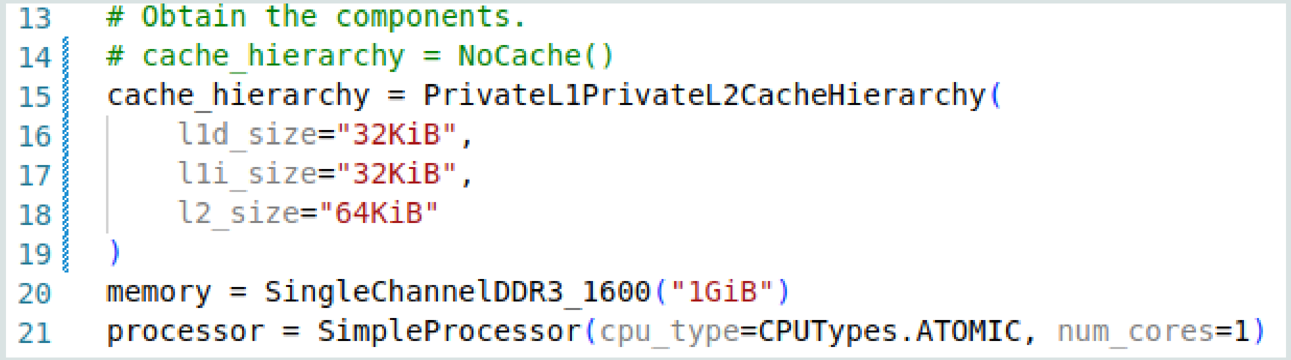
In general, you can replace components with components of the same type.

Let's add a real cache implementation to our design!

Extending our design



```python
#from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.cachehierarchies.classic.private_l1_private_l2_cache_hierarchy import (
    PrivateL1PrivateL2CacheHierarchy
)
```



```python
# Obtain the components.
# cache_hierarchy = NoCache()
cache_hierarchy = PrivateL1PrivateL2CacheHierarchy(
    l1d_size="32KiB",
    l1i_size="32KiB",
    l2_size="64KiB"
)
memory = SingleChannelDDR3_1600("1GiB")
processor = SimpleProcessor(cpu_type=CPUTypes.ATOMIC, num_cores=1)
```

Save the file again.

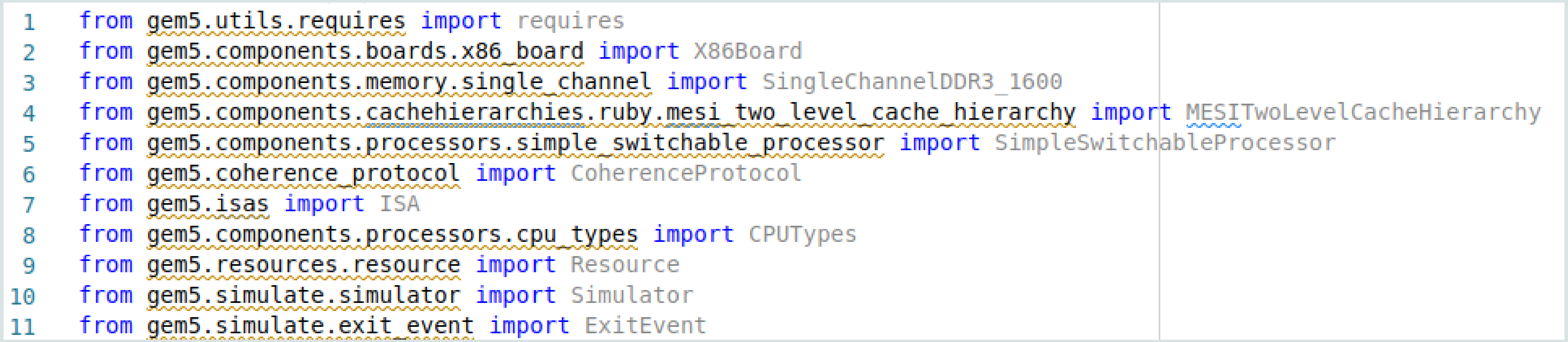
```
> gem5-x86 materials/using-gem5/02-stdlib/hello-world.py
```

Extending our design

Check the output in “m5out/stats.txt” and see how the Simulated Seconds and Simulated Ticks varies when using and not using a cache.

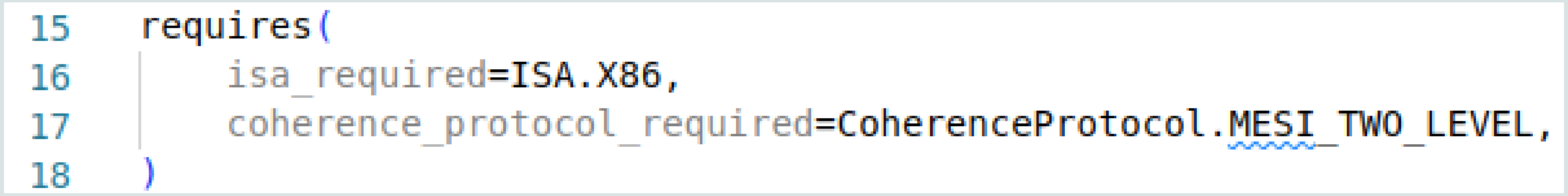
An X86 full-system simulation

"materials/using-gem5/02-stdlib/x86-full-system.py”



```python
from gem5.utils.requires import requires
from gem5.components.boards.x86_board import X86Board
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.cachehierarchies.ruby.mesi_two_level_cache_hierarchy import MESITwoLevelCacheHierarchy
from gem5.components.processors.simple_switchable_processor import SimpleSwitchableProcessor
from gem5.coherence_protocol import CoherenceProtocol
from gem5.isas import ISA
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator
from gem5.simulate.exit_event import ExitEvent
```

An X86 full-system simulation

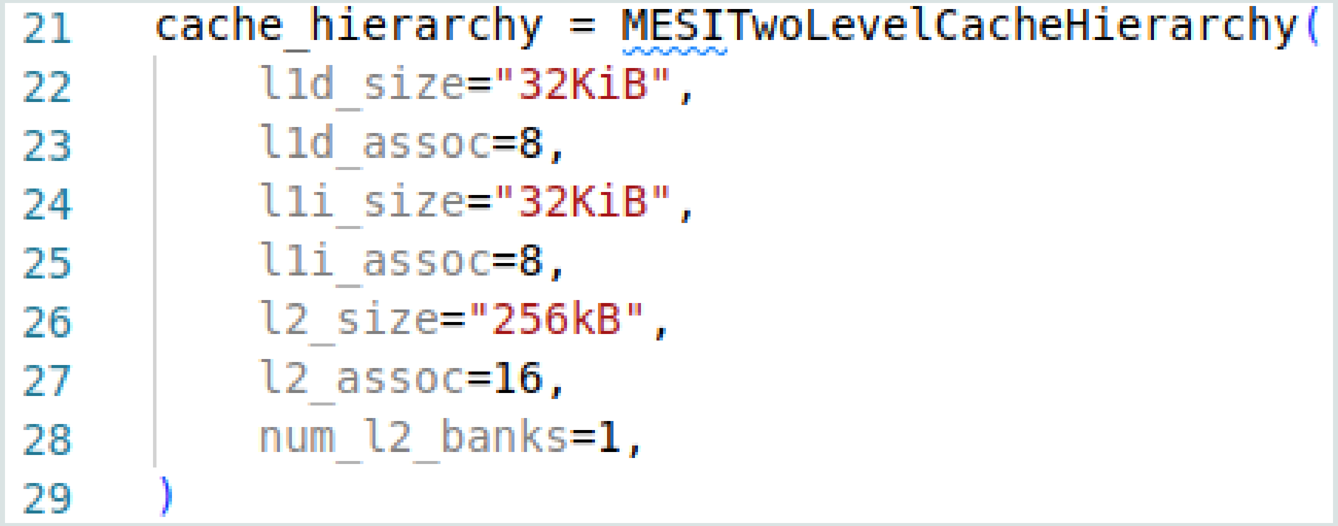


```
requires(
    isa_required=ISA.X86,
    coherence_protocol_required=CoherenceProtocol.MESI_TWO_LEVEL,
)
```

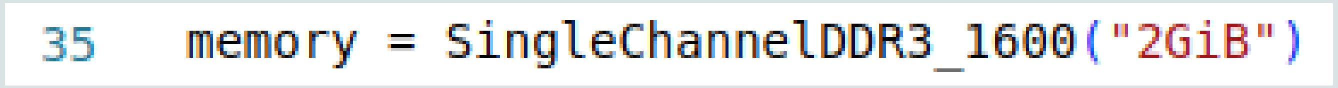
This adds a check for the gem5 binary parsing the script. In this case:

1. The binary supports the X86 ISA.
2. The binary supports the MESI Two Level coherence protocol.

An X86 full-system simulation

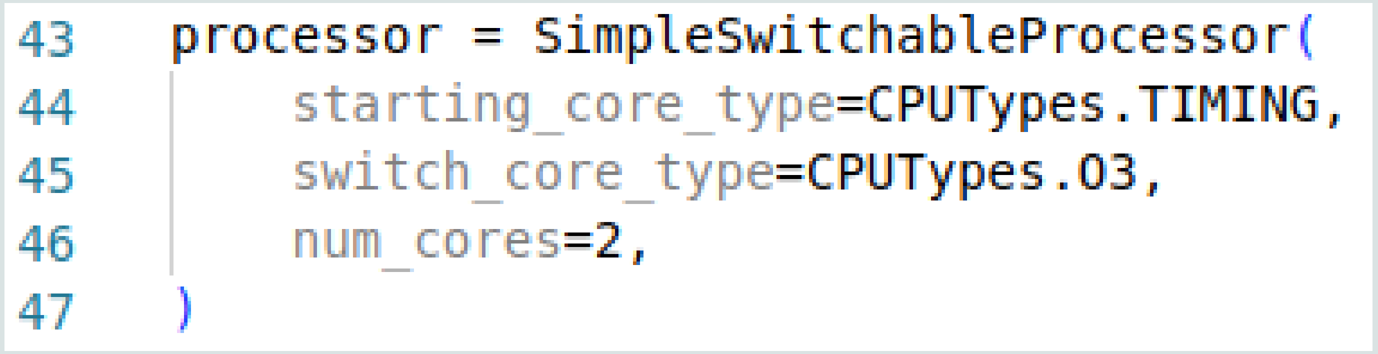


```
cache_hierarchy = MESITwoLevelCacheHierarchy(
    l1d_size="32KiB",
    l1d_assoc=8,
    l1i_size="32KiB",
    l1i_assoc=8,
    l2_size="256kB",
    l2_assoc=16,
    num_l2_banks=1,
)
```



```
memory = SingleChannelDDR3_1600("2GiB")
```

An X86 full-system simulation

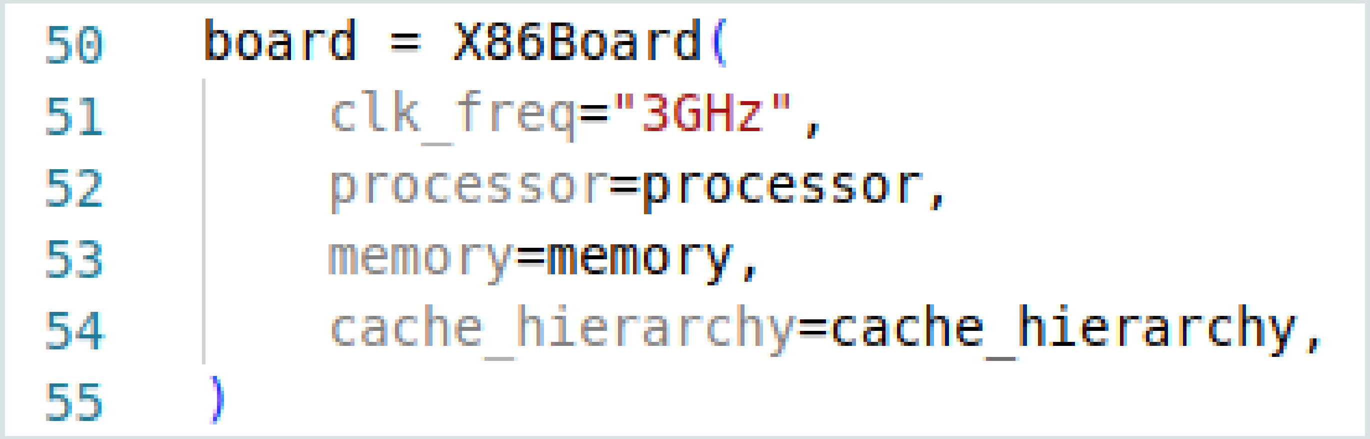


```
processor = SimpleSwitchableProcessor(
    starting_core_type=CPUTypes.TIMING,
    switch_core_type=CPUTypes.O3,
    num_cores=2,
)
```

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.

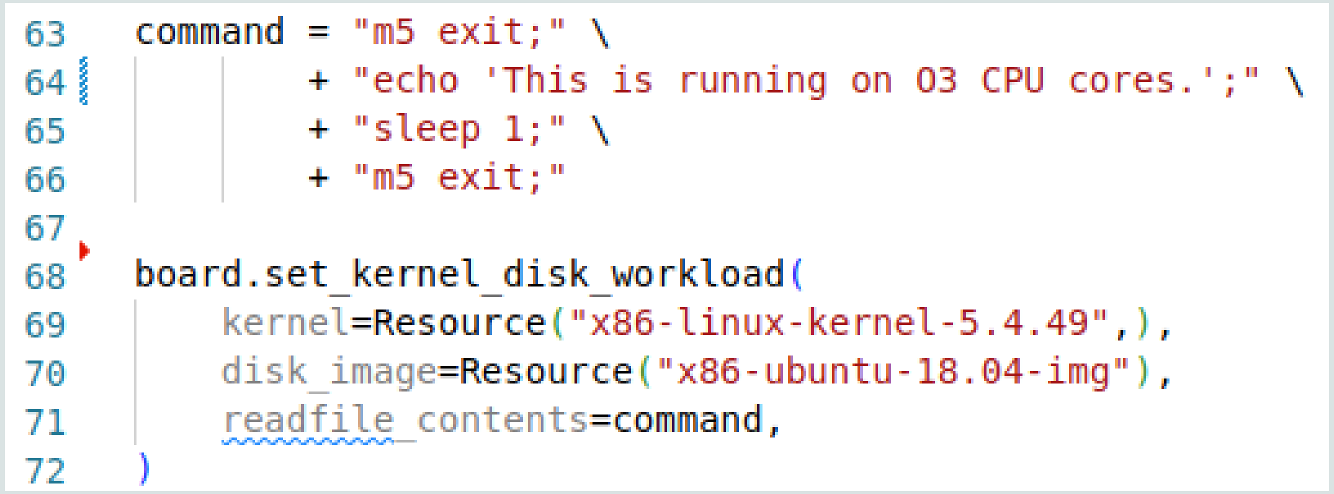
An X86 full-system simulation



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

As usual, we add the components to the board, in this case an `X86Board`.

An X86 full-system simulation



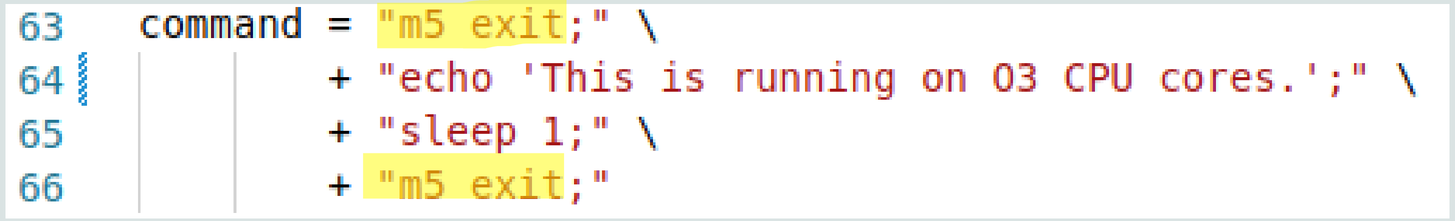
```
command = "m5 exit;" \
        + "echo 'This is running on O3 CPU cores.';" \
        + "sleep 1;" \
        + "m5 exit;"

board.set_kernel_disk_workload(
    kernel=Resource("x86-linux-kernel-5.4.49",),
    disk_image=Resource("x86-ubuntu-18.04-img"),
    readfile_contents=command,
)
```

The 'set_kernel_disk_workload` function is used to run a full system workload.

You must specify the `kernel` resource to use and the `disk_image` resource.

In this case we can set the command to run on boot.

The Simulator Module



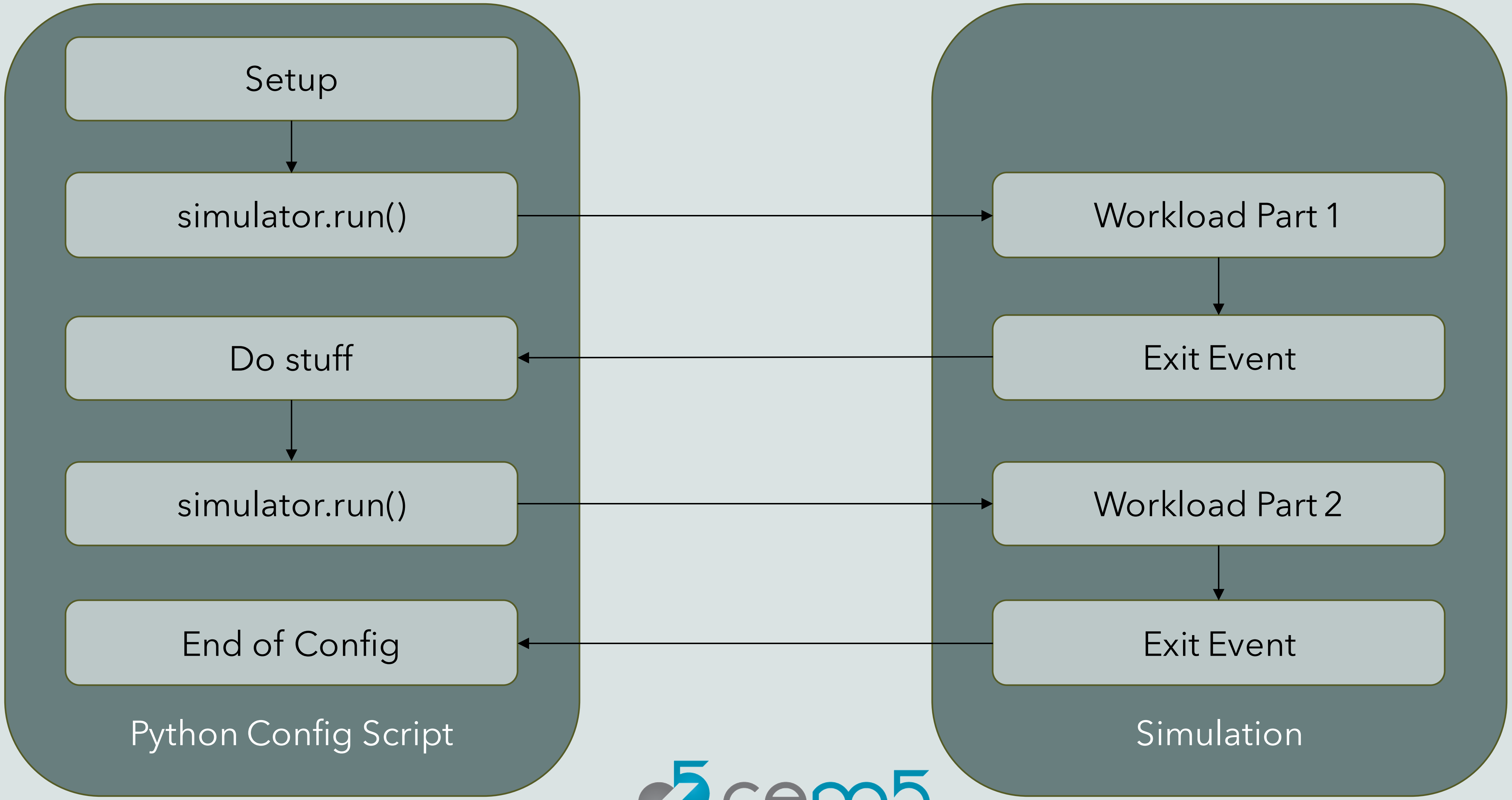
```
command = "m5 exit;" \
        + "echo 'This is running on O3 CPU cores.';" \
        + "sleep 1;" \
        + "m5 exit;"
```

During a simulation you can have "Exit Events".

In this example there are two. These exit the simulation loop and return to the Python Script.

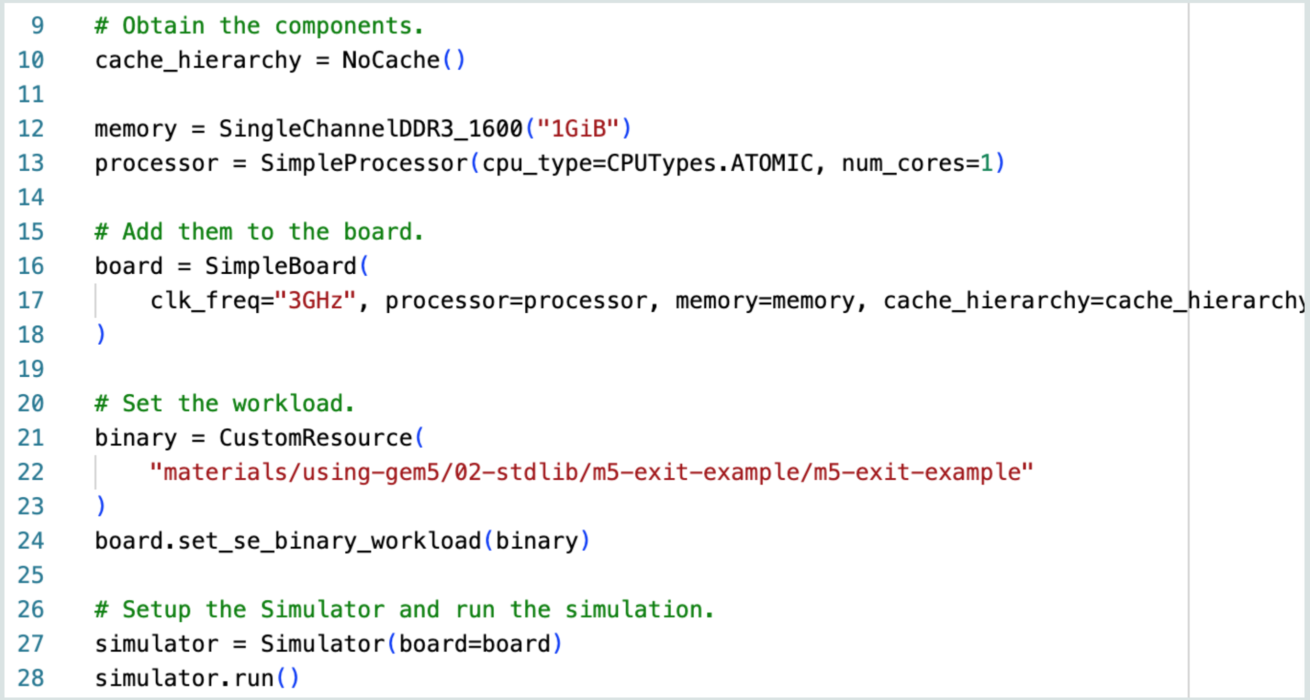
The Simulator Module is used to handle these events. Let's play with some examples to see how.

The Simulator Module



The Simulator Module

Go to “materials/using-gem5/02-stdlib/simulator-use.py”

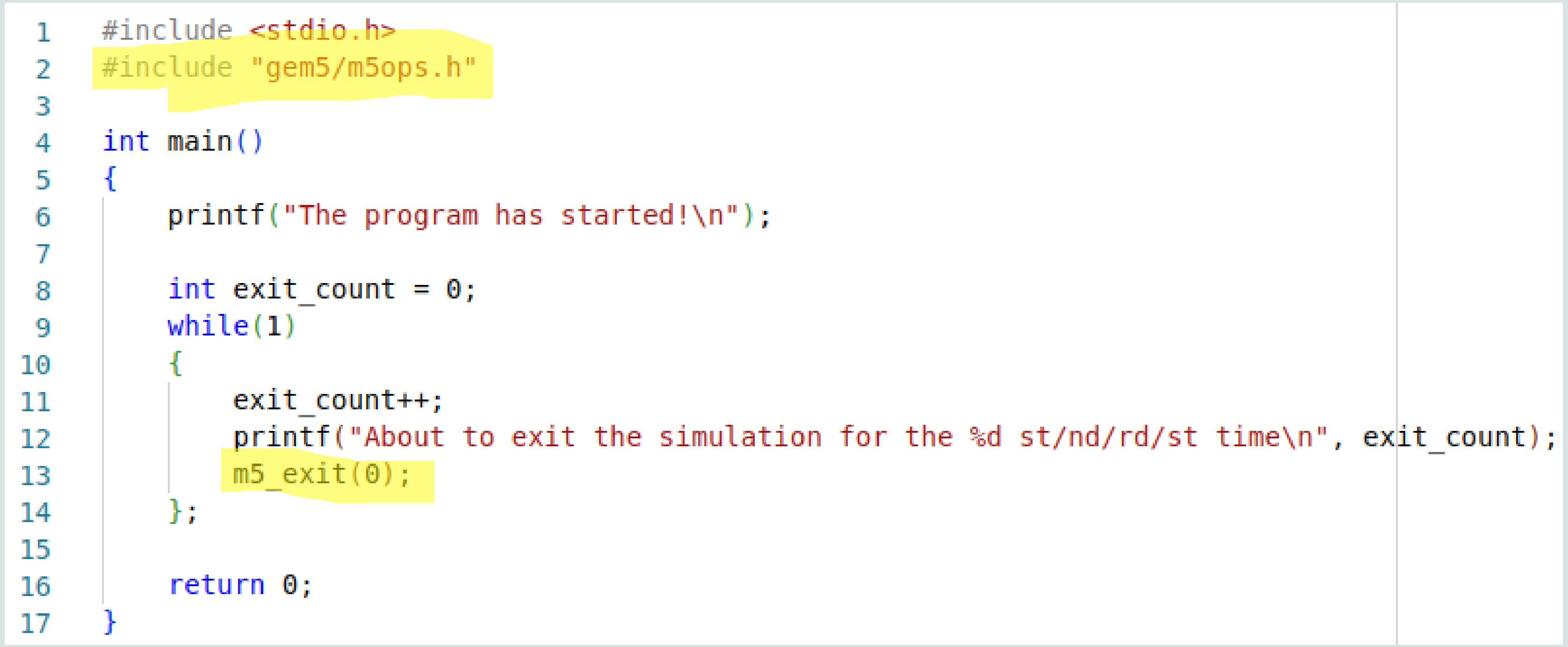


```
# Obtain the components.
cache_hierarchy = NoCache()

memory = SingleChannelDDR3_1600("1GiB")
processor = SimpleProcessor(cpu_type=CPUTypes.ATOMIC, num_cores=1)

# Add them to the board.
board = SimpleBoard(
    clk_freq="3GHz", processor=processor, memory=memory, cache_hierarchy=cache_hierarchy
)

# Set the workload.
binary = CustomResource(
    "materials/using-gem5/02-stdlib/m5-exit-example/m5-exit-example"
)
board.set_se_binary_workload(binary)

# Setup the Simulator and run the simulation.
simulator = Simulator(board=board)
simulator.run()
```

This is a pretty normal looking script but we are running this “m5-exit-example” binary. Let’s look into it

The Simulator Module

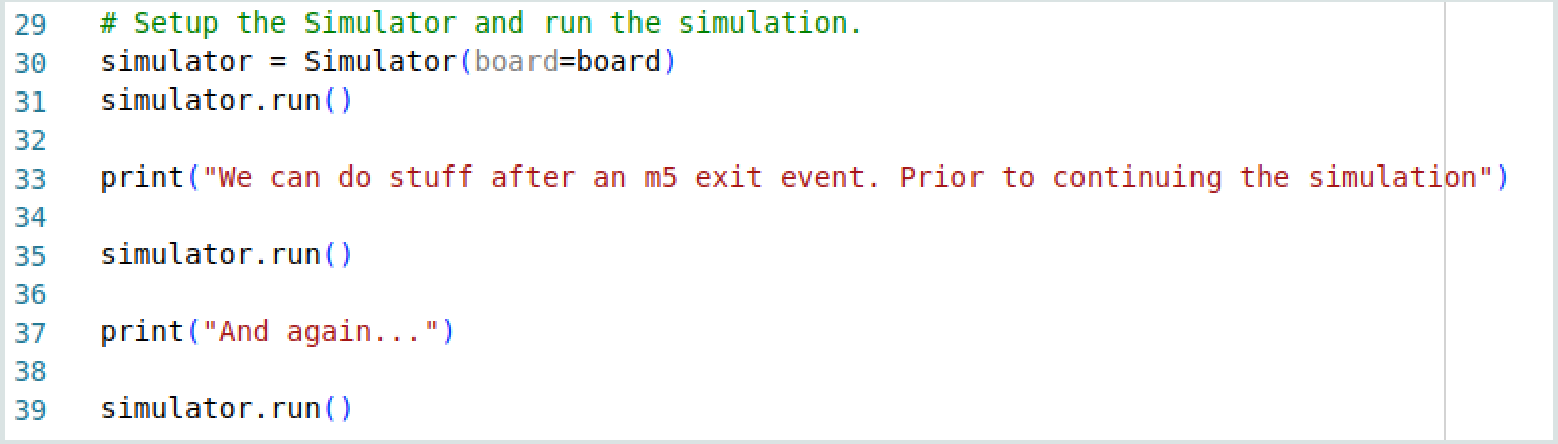
Go to “materials/using-gem5/02-stdlib/m5-exit-example/m5-exit-example.c”



```c
#include <stdio.h>
#include "gem5/m5ops.h"

int main()
{
    printf("The program has started!\n");

    int exit_count = 0;
    while(1)
    {
        exit_count++;
        printf("About to exit the simulation for the %d st/nd/rd/st time\n", exit_count);
        m5_exit(0);
    };

    return 0;
}
```

The Simulator Module

Go back to “materials/using-gem5/02-stdlib/simulator-use.py”

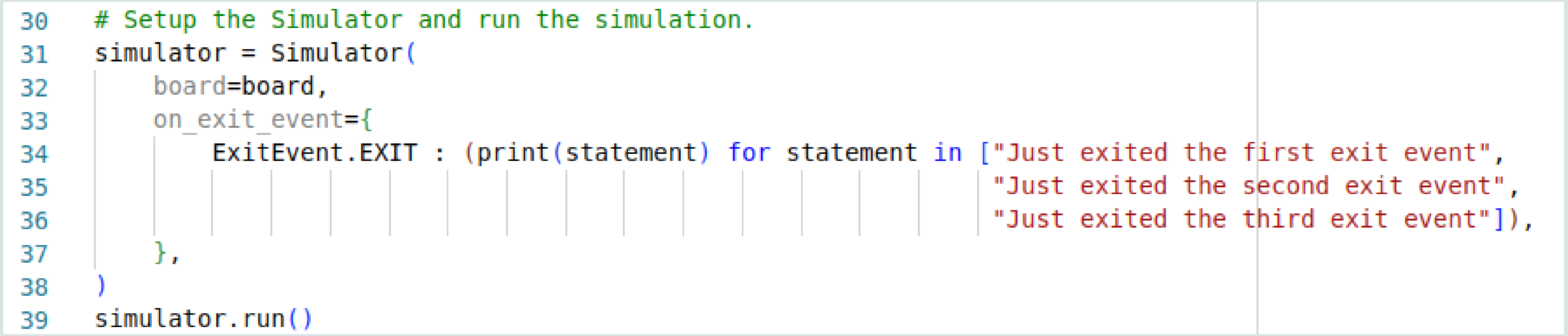


```python
# Setup the Simulator and run the simulation.
simulator = Simulator(board=board)
simulator.run()

print("We can do stuff after an m5 exit event. Prior to continuing the simulation")

simulator.run()

print("And again...")

simulator.run()
```

```
> gem5-x86 materials/using-gem5/02-stdlib/simulator-use.py
```

The Simulator Module

Let’s be a bit cleverer…



```python
# Setup the Simulator and run the simulation.
simulator = Simulator(
    board=board,
    on_exit_event={
        ExitEvent.EXIT : (print(statement) for statement in ["Just exited the first exit event",
                                                              "Just exited the second exit event",
                                                              "Just exited the third exit event"]),
    },
)
simulator.run()
```

```
> gem5-x86 materials/using-gem5/02-stdlib/simulator-use.py
```

The Simulator Module

Here we’re only covering the “Exit” type exit event, but there are other types.

You can override different types for different things.

The Simulator module has default behavior for each (see “gem5/src/python/gem5/simulate/exit_event_generators.py”)

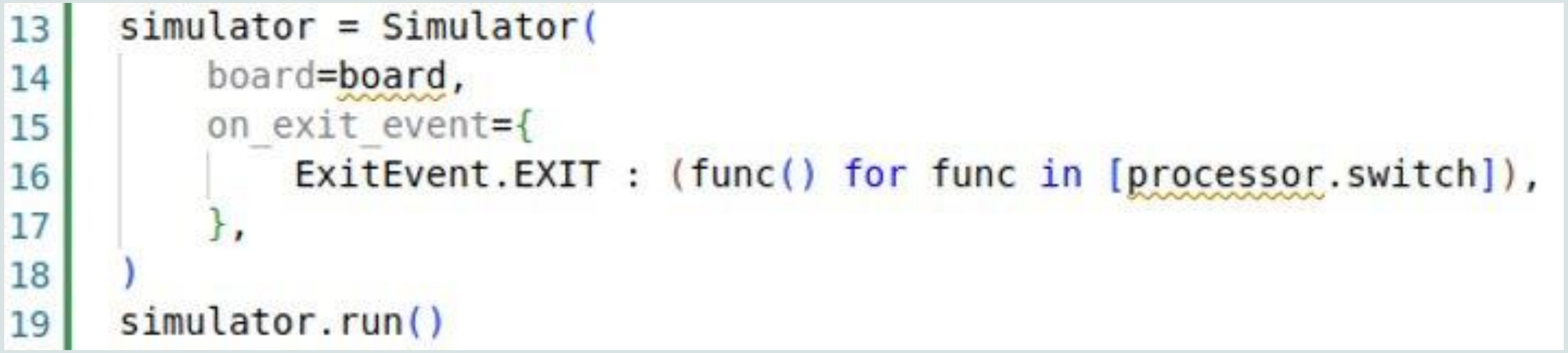
- ExitEvent.EXIT
- ExitEvent.CHECKPOINT
- ExitEvent.FAIL
- ExitEvent.SWITCHCPU
- ExitEvent.WORKBEGIN
- ExitEvent.WORKEND
- ExitEvent.USER_INTERRUPT
- ExitEvent.MAX_TICK

The Simulator Module

Note: This is module is still considered to be in Beta. The API may change in future versions of gem5

An X86 full-system simulation

Let’s go back to “materials/using-gem5/02-stdlib/x86-full-system.py”



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

An X86 full-system simulation

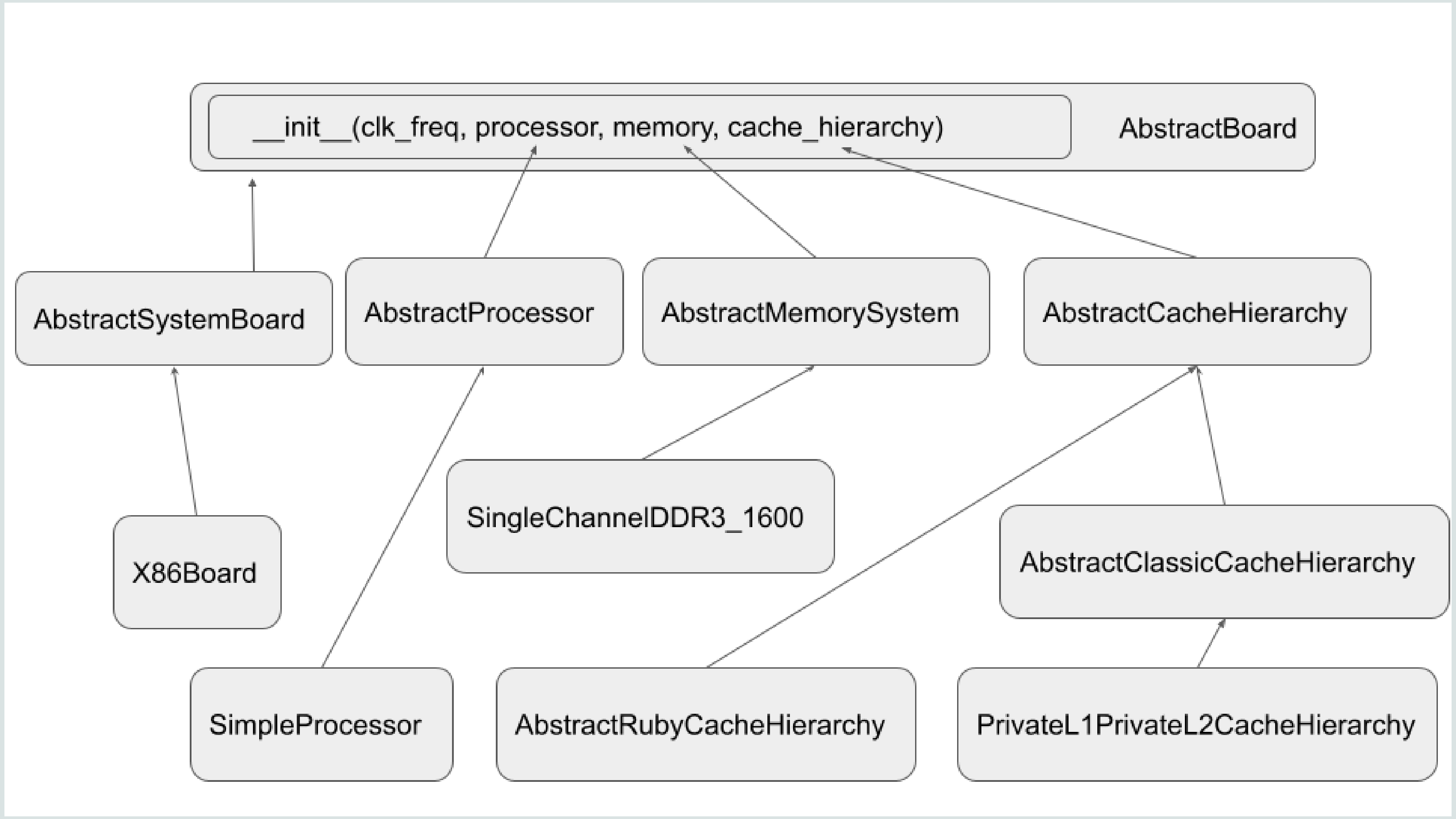
We’re done! You now have a full-system simulation!

```
> gem5-x86 materials/using-gem5/02-stdlib/x86-full-system.py
```

Warning: This will take a long time
to complete execution.

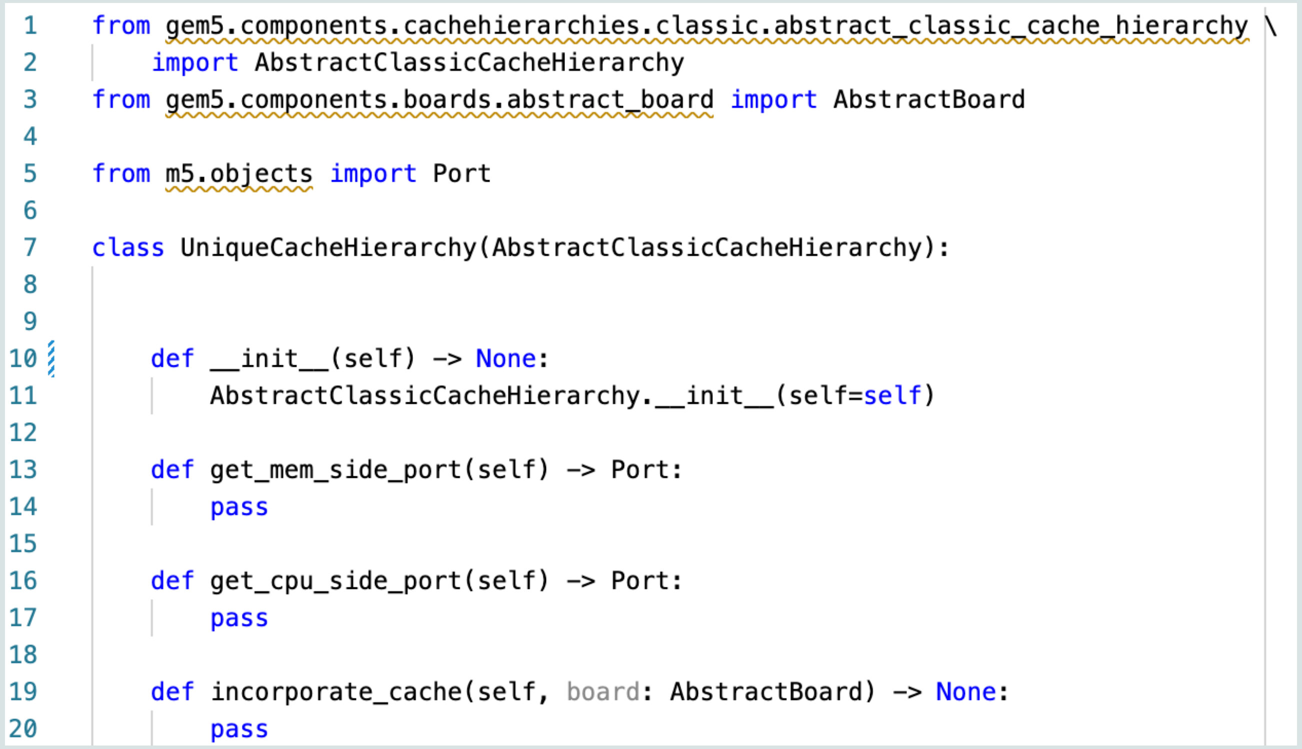
Cntl+C to exit this.

Extending the library


__init__(clk_freq, processor, memory, cache_hierarchy)
AbstractBoard
AbstractSystemBoard
AbstractProcessor
AbstractMemorySystem
AbstractCacheHierarchy
X86Board
SingleChannelDDR3_1600
AbstractClassicCacheHierarchy
SimpleProcessor
AbstractRubyCacheHierarchy
PrivateL1PrivateL2CacheHierarchy

Extending the library

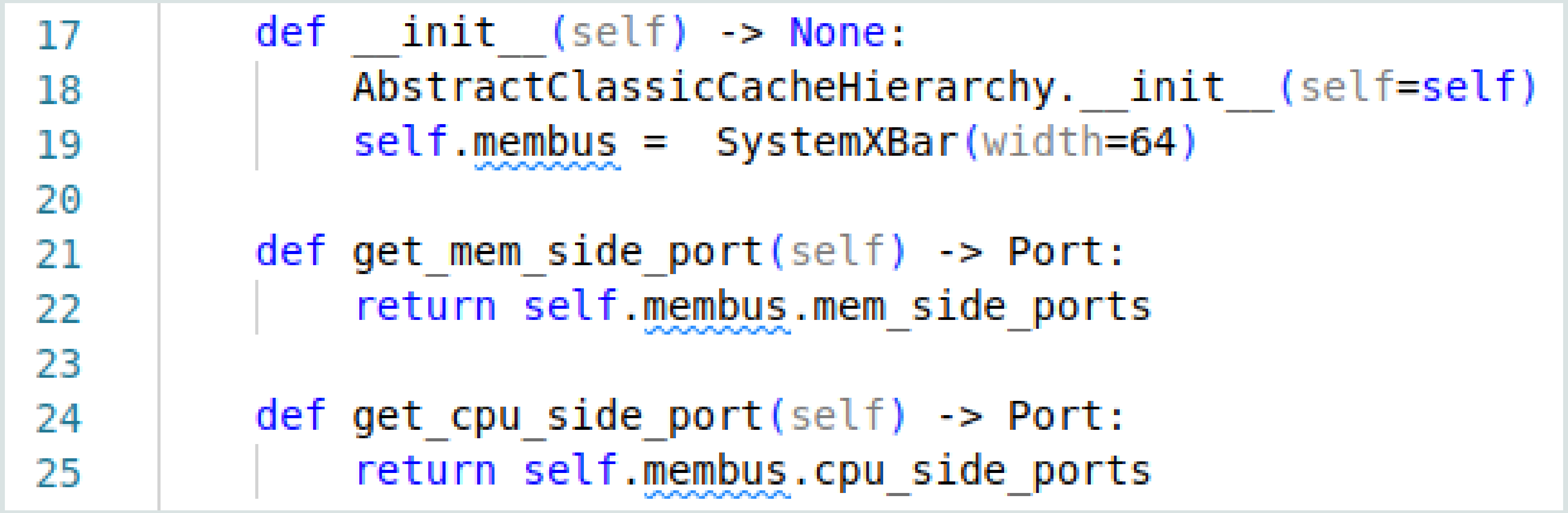
Open “materials/using-gem5/02-stdlib/unique_cache_hierarchy/unique_cache_hierarchy.py”



```python
from gem5.components.cachehierarchies.classic.abstract_classic_cache_hierarchy \
    import AbstractClassicCacheHierarchy
from gem5.components.boards.abstract_board import AbstractBoard

from m5.objects import Port

class UniqueCacheHierarchy(AbstractClassicCacheHierarchy):


    def __init__(self) -> None:
        AbstractClassicCacheHierarchy.__init__(self=self)

    def get_mem_side_port(self) -> Port:
        pass

    def get_cpu_side_port(self) -> Port:
        pass

    def incorporate_cache(self, board: AbstractBoard) -> None:
        pass
```

Extending the library

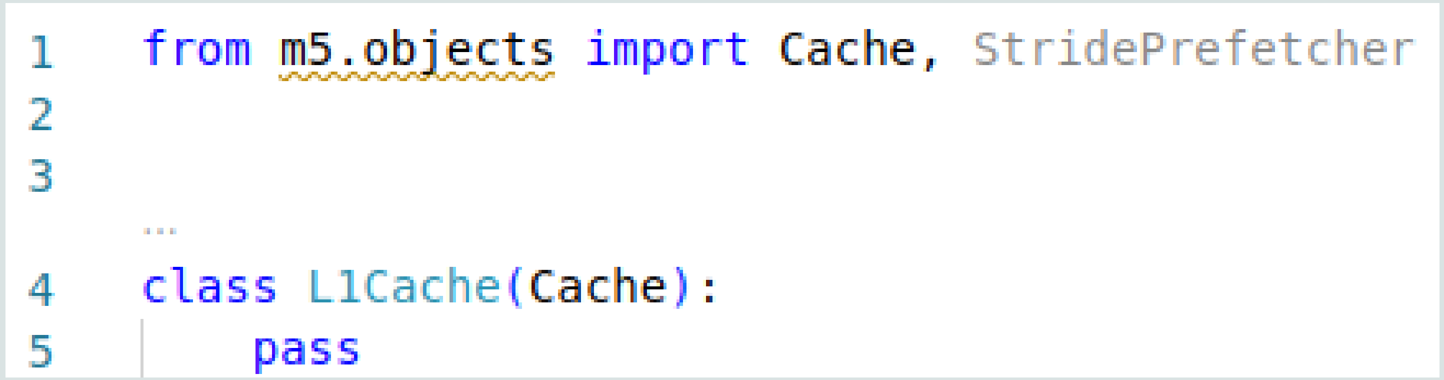
Complete the constructor and declare the mem-side and cpu-side ports



```python
    def __init__(self) -> None:
        AbstractClassicCacheHierarchy.__init__(self=self)
        self.membus =  SystemXBar(width=64)

    def get_mem_side_port(self) -> Port:
        return self.membus.mem_side_ports

    def get_cpu_side_port(self) -> Port:
        return self.membus.cpu_side_ports
```

Extending the library

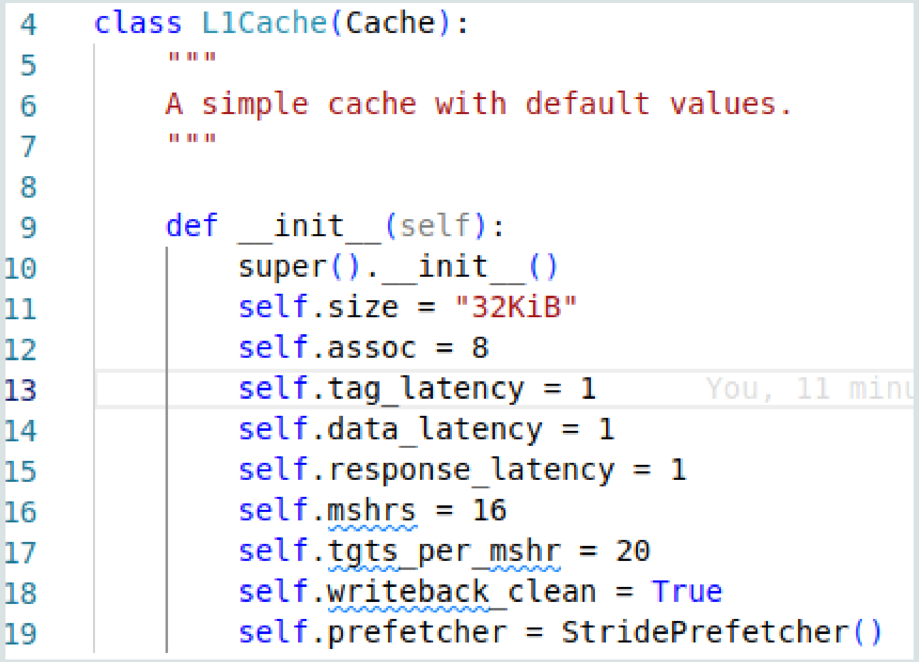
Next, open ”materials/using-gem5/02-stdlib/unique_cache_hierarchy/l1cache.py”



```
from m5.objects import Cache, StridePrefetcher


...
class L1Cache(Cache):
    pass
```

Extending the library

Let's extend the "Cache" SimObject to customize it for our purposes



```python
class L1Cache(Cache):
    """
    A simple cache with default values.
    """

    def __init__(self):
        super().__init__()
        self.size = "32KiB"
        self.assoc = 8
        self.tag_latency = 1        You, 11 minu
        self.data_latency = 1
        self.response_latency = 1
        self.mshrs = 16
        self.tgts_per_mshr = 20
        self.writeback_clean = True
        self.prefetcher = StridePrefetcher()
```

Important note:

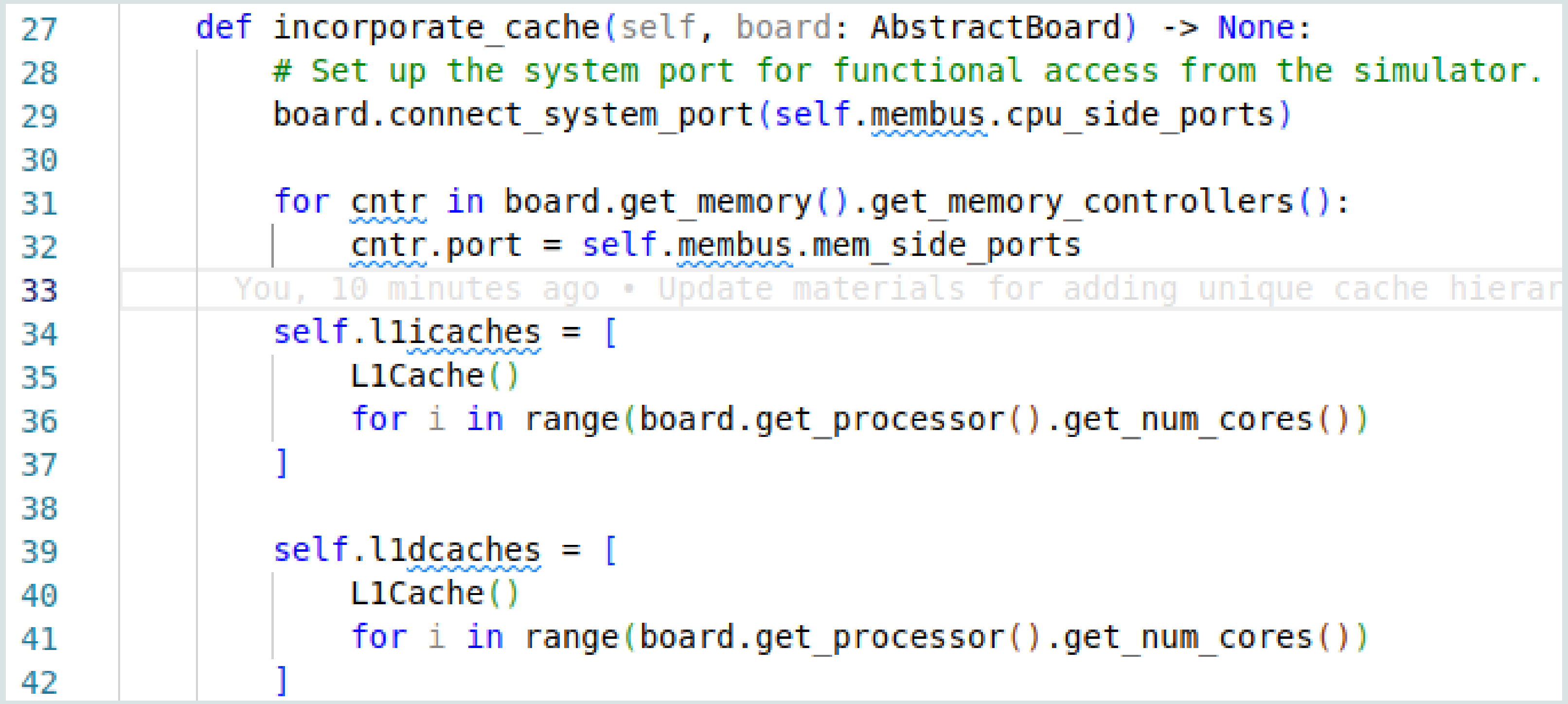
SimObject member variables are special. You can only set SimObject variables. Code like `self.custom_variable =7` will cause an error.

If you want create a non-SimObject variable, the variable name must have a preceding underscore:

`self._custom_variable = 7`

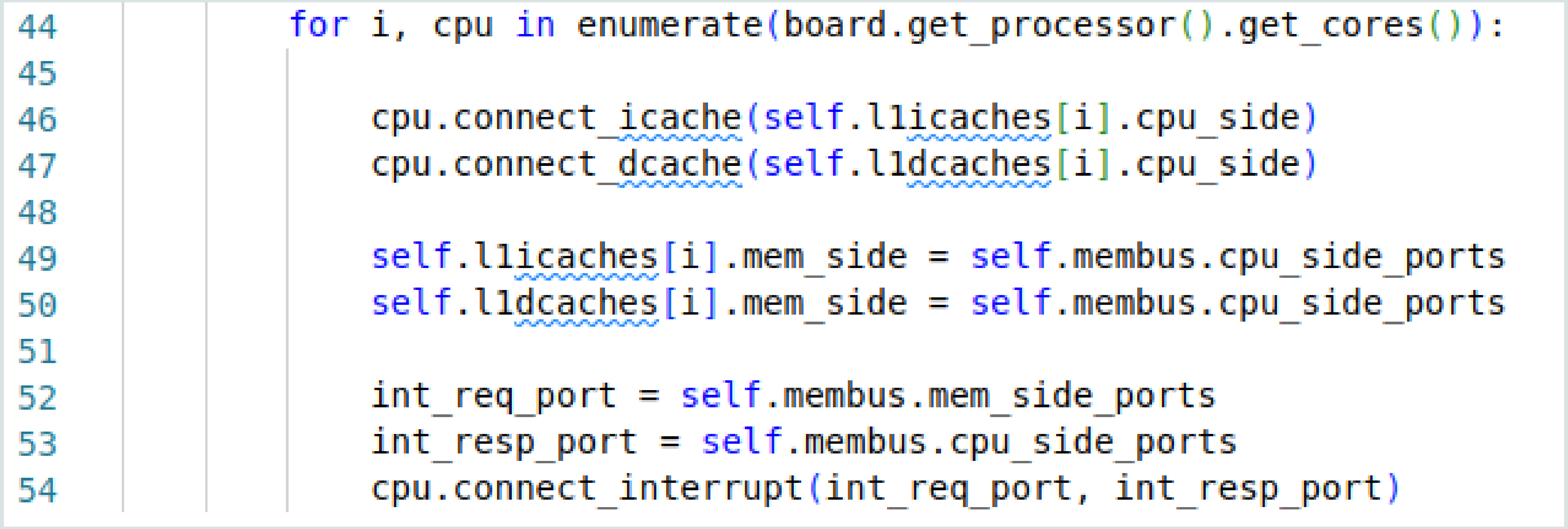
Extending the library

Go back to
“materials/using-gem5/02-stdlib/unique_cache_hierarchy/unique_cache_hierarchy.py”



```
    def incorporate_cache(self, board: AbstractBoard) -> None:
        # Set up the system port for functional access from the simulator.
        board.connect_system_port(self.membus.cpu_side_ports)

        for cntr in board.get_memory().get_memory_controllers():
            cntr.port = self.membus.mem_side_ports
          You, 10 minutes ago • Update materials for adding unique cache hierar
        self.l1icaches = [
            L1Cache()
            for i in range(board.get_processor().get_num_cores())
        ]

        self.l1dcaches = [
            L1Cache()
            for i in range(board.get_processor().get_num_cores())
        ]
```

Extending the library



```python
        for i, cpu in enumerate(board.get_processor().get_cores()):

            cpu.connect_icache(self.l1icaches[i].cpu_side)
            cpu.connect_dcache(self.l1dcaches[i].cpu_side)

            self.l1icaches[i].mem_side = self.membus.cpu_side_ports
            self.l1dcaches[i].mem_side = self.membus.cpu_side_ports

            int_req_port = self.membus.mem_side_ports
            int_resp_port = self.membus.cpu_side_ports
            cpu.connect_interrupt(int_req_port, int_resp_port)
```

Extending the library

Try adding this cache hierarchy to your “hello-world.py” example.

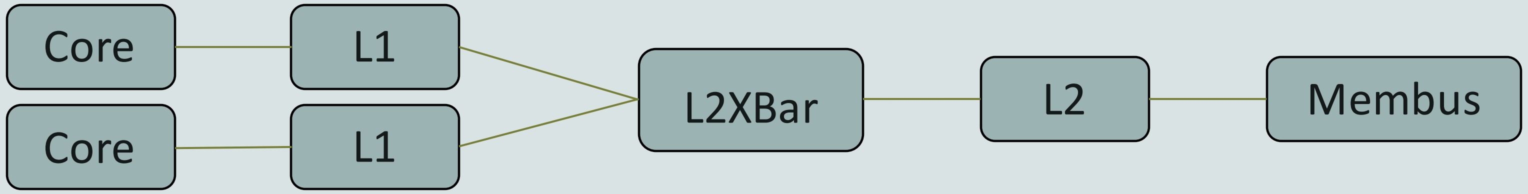
It can be done via a standard Python import.

Create your own cache!!

As a challenge, create your own gem5 Private L1 Shared L2 Cache hierarchy.

Once done, modify your "hello-world.py" program to see the performance difference of using the Private L1 with a Private L2 Cache hierarchy vs a Private L1 with a Shared L2 Cache.

Feel free to be creative



Helpful: `from m5.objects import L2XBar`

"src/python/gem5/components/cachehierarchies/classic" contains the code for a Private L1 Cache Hierarchy and a PL1/PL2 Cache Hierarchy. It may be helpful to reference them.