The gem5 Standard Library

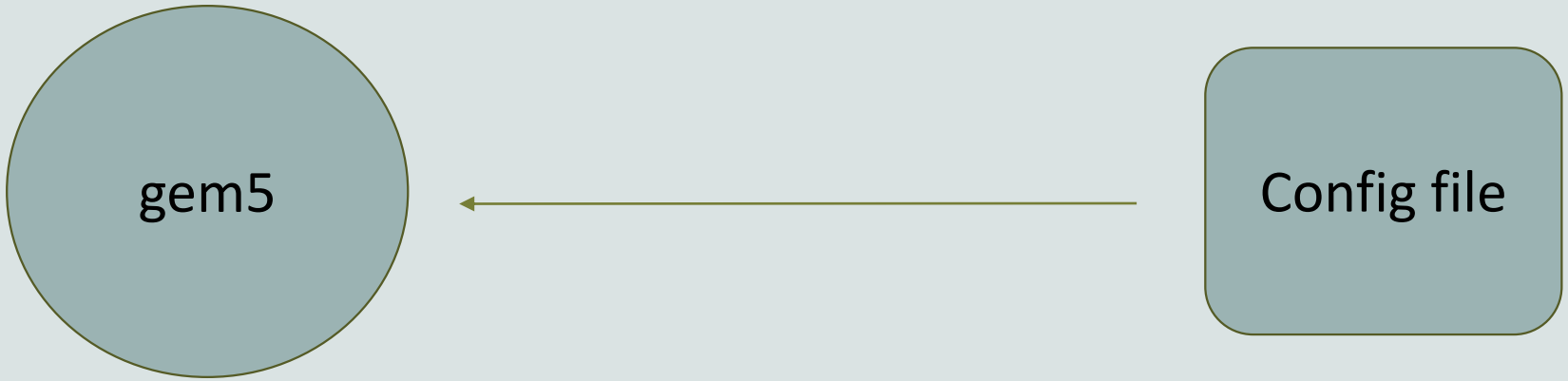
A presentation by

Bobby R. Bruce

UCDAVIS COMPUTER SCIENCE

DArchR DAVIS ARCHITECTURE RESEARCH

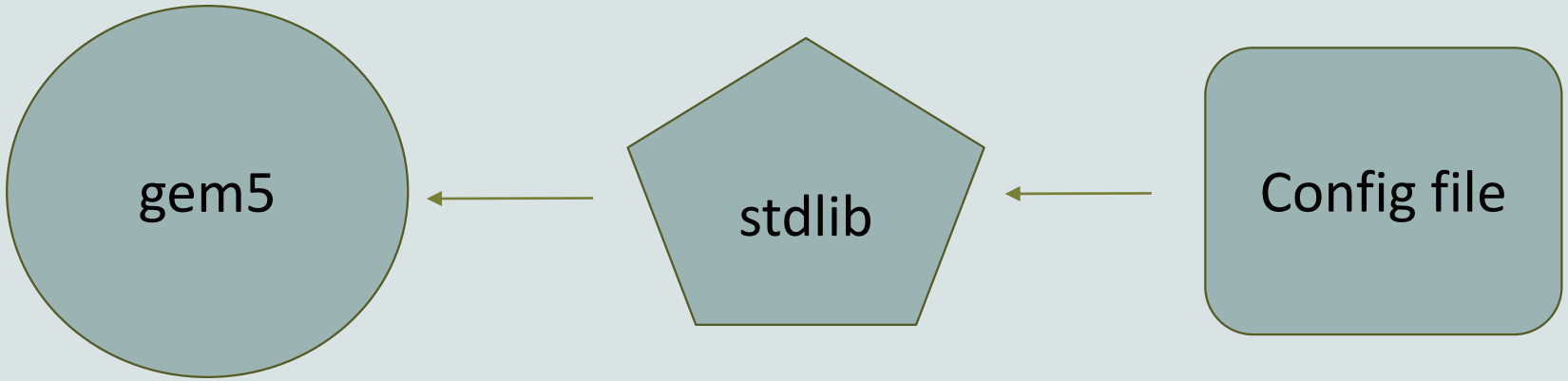
What is the standard library for?



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.

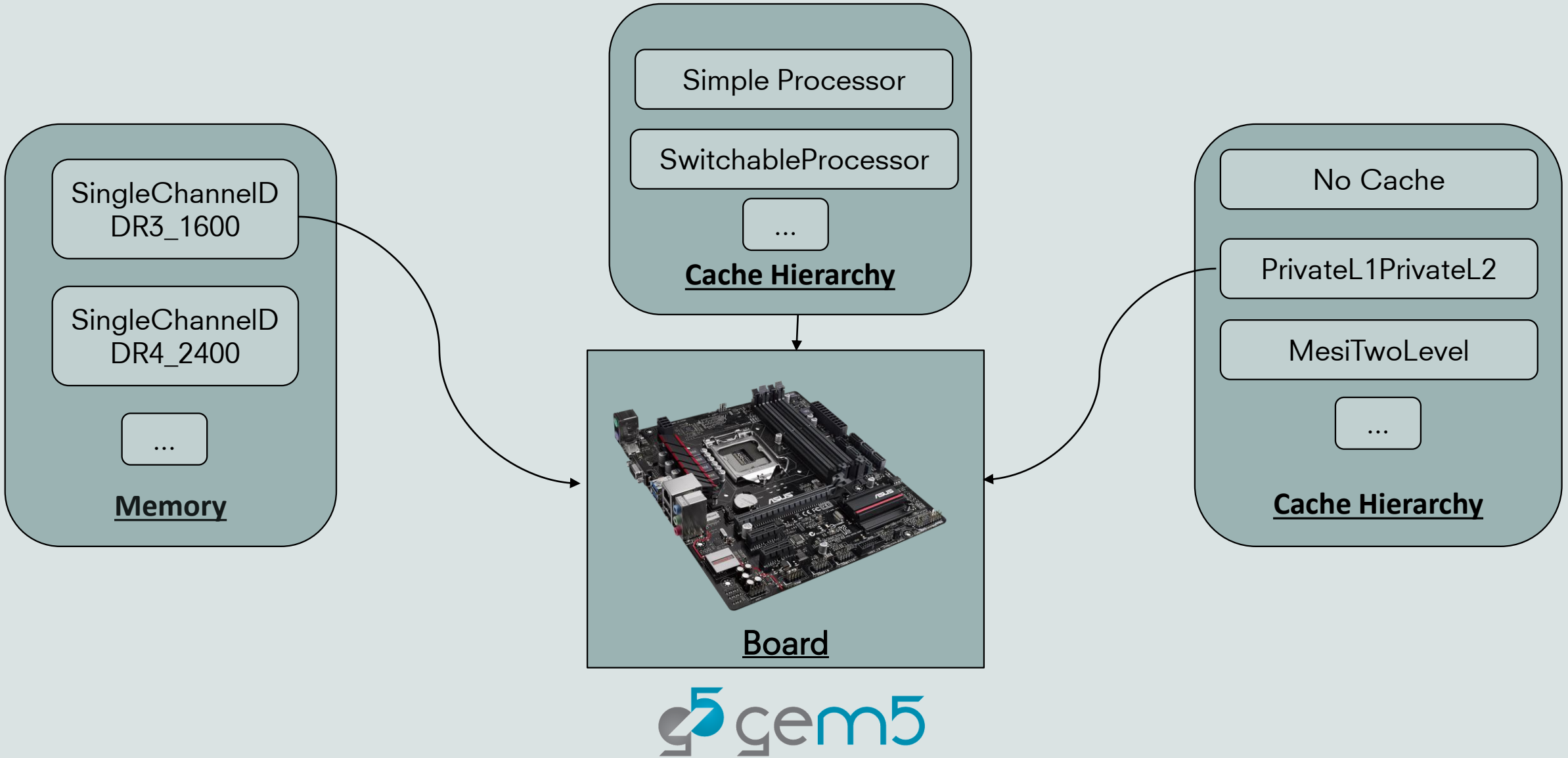
What is the standard library for?



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

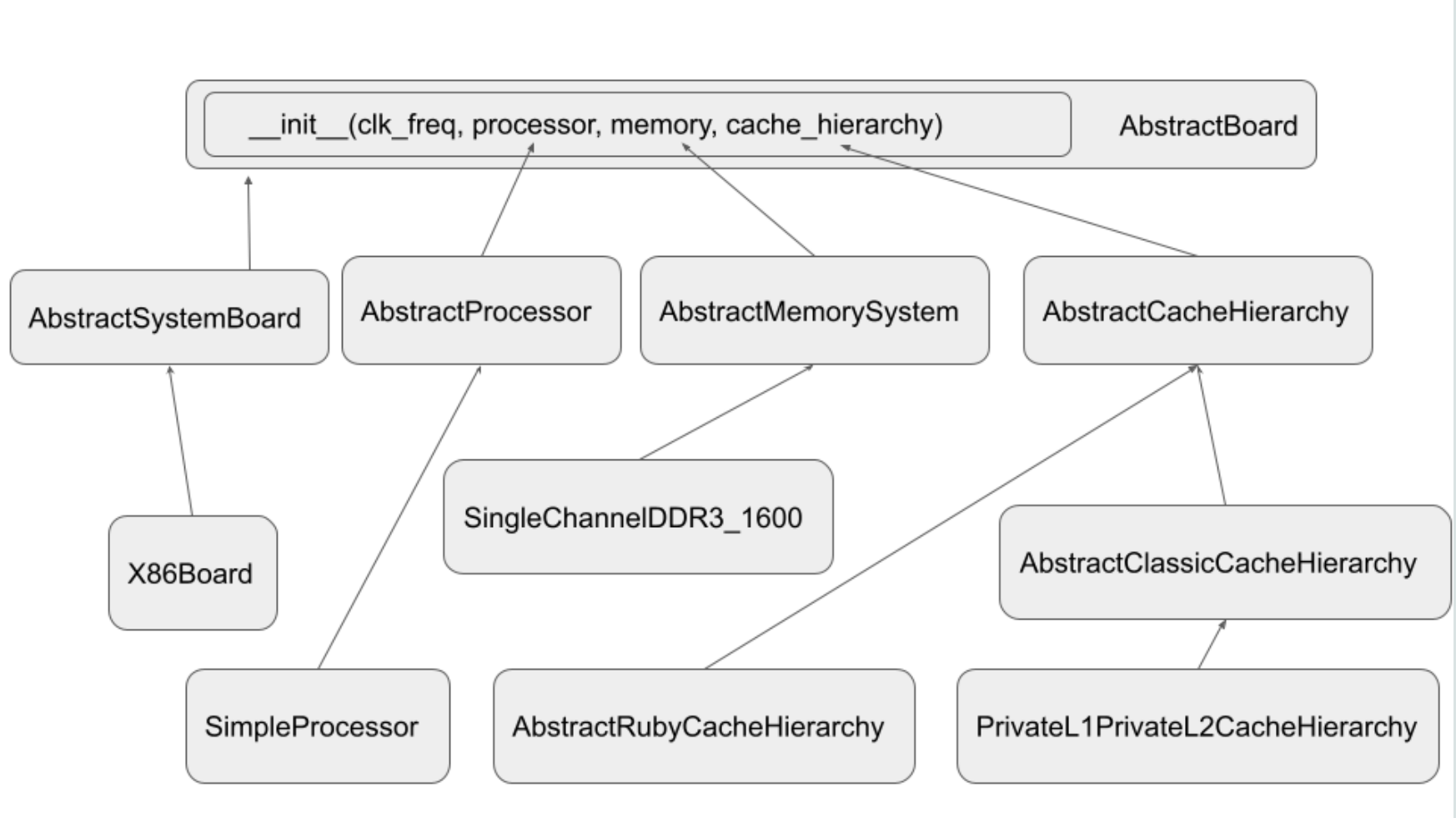
The stdlib's module architecture allows for components (e.g. a memory system or a cache hierarchy setup) to be quickly swapped in and out without radical redesign.

The stdlib modular metaphor

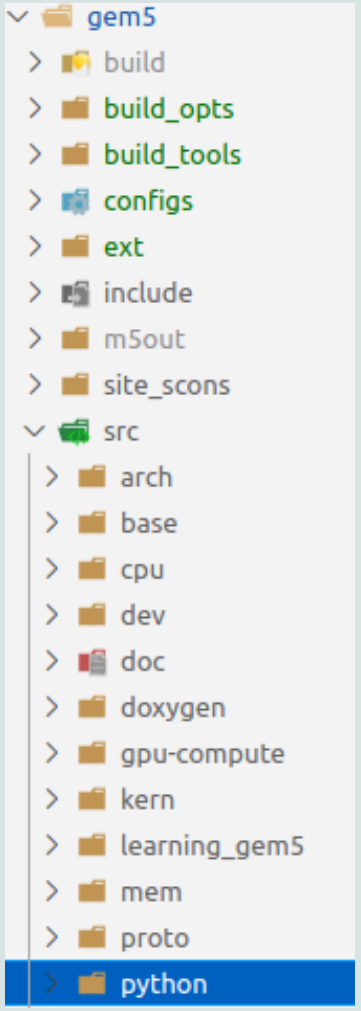


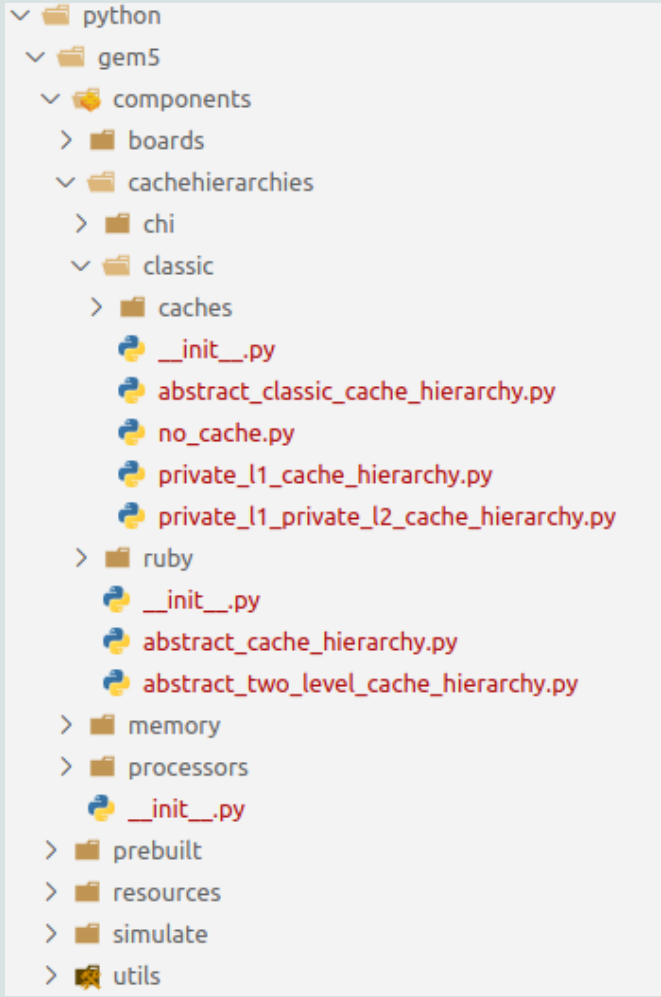
gem5

The modular architecture

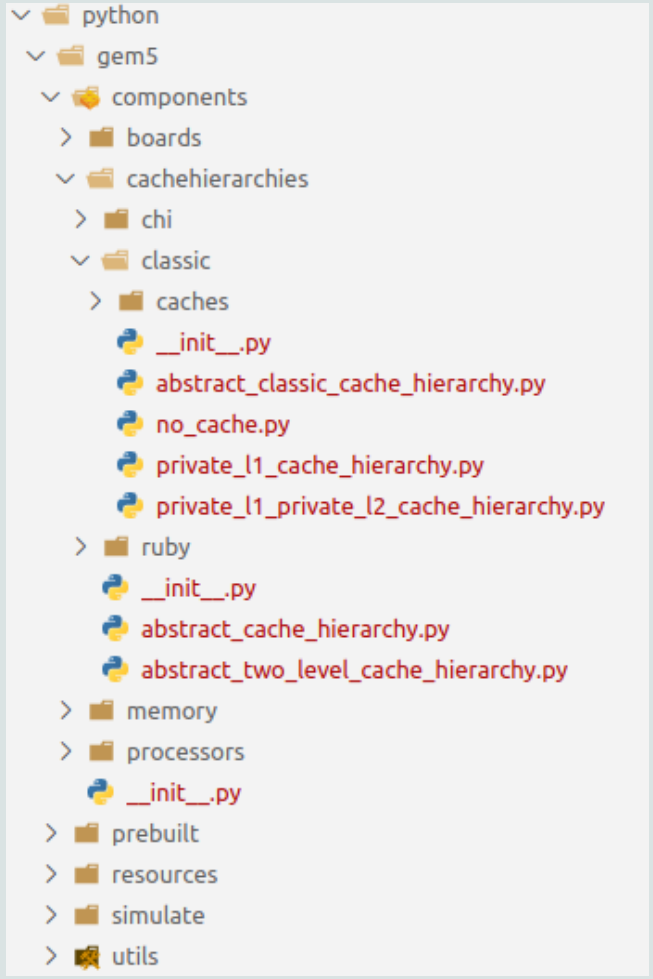


Where to find stuff: The directory structure





Where to find stuff : Importing in a script




```
1  from gem5.components.boards.simple_board import SimpleBoard
2  from gem5.components.cachehierarchies.classic.no_cache import NoCache
3  from gem5.components.memory.single_channel import SingleChannelDDR3_1600
4  from gem5.components.processors.simple_processor import SimpleProcessor
5  from gem5.components.processors.cpu_types import CPUTypes
6  from gem5.resources.resource import Resource
7  from gem5.simulate.simulator import Simulator
```

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

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

You should see the following:

```
1  from gem5.components.boards.simple_board import SimpleBoard
2  from gem5.components.cachehierarchies.classic.no_cache import NoCache
3  from gem5.components.memory.single_channel import SingleChannelDDR3_1600
4  from gem5.components.processors.simple_processor import SimpleProcessor
5  from gem5.components.processors.cpu_types import CPUTypes
6  from gem5.resources.resource import Resource
7  from gem5.simulate.simulator import Simulator
```

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

```
 9  # Obtain the components.
10  cache_hierarchy = NoCache()
11  memory = SingleChannelDDR3_1600("1GiB")
12  processor = SimpleProcessor(cpu_type=CPUTypes.ATOMIC, num_cores=1)
```

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

```
14  #Add them to the board.
15  board = SimpleBoard(
16      clk_freq="3GHz",
17      processor=processor,
18      memory=memory,
19      cache_hierarchy=cache_hierarchy,
20  )
```

gem5 Resources

- gem5 resources is a repository providing sources for artifacts 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

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

```
1  from gem5.resources.resource import Resource
2
3  resource = Resource("riscv-disk-img")
4
5  print(f"The resources is available at {resource.get_local_path()}")
```

Run this with `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.

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)

```
1   from gem5.resources.resource import CustomResource
2
3   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:

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

An `se_binary_workload` is running a board in Syscall Emulation mode, with a single binary

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

Append the following:

```
26  # Setup the Simulator and run the simulation.
27  simulator = Simulator(board=board)
28  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.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 = 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

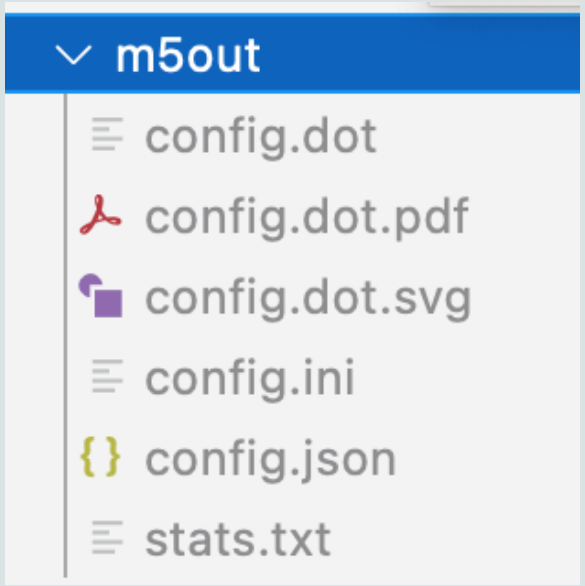
Save the file.

Run this example using: `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



- 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

```
#from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.cachehierarchies.classic.private_l1_cache_hierarchy import PrivateL1CacheHierarchy
```

```
10  # Obtain the components.
11  #cache_hierarchy = NoCache()
12  cache_hierarchy = PrivateL1CacheHierarchy(l1d_size="32kB", l1i_size="32kB")
13  memory = SingleChannelDDR3_1600("1GiB")
14  processor = SimpleProcessor(cpu_type=CPUTypes.ATOMIC, num_cores=1)
```

Save the file again and run `gem5-x86 materials/using-gem5/02-stdlib/hello-world.py`

More complex designs: An X86 full system simulation in the stdlib

Move to "materials/02-stdlib/x86-full-system.py. You should see the following provided for you:

```
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
```

Adding the 'requires' function

```
15  requires(
16      isa_required=ISA.X86,
17      coherence_protocol_required=CoherenceProtocol.MESI_TWO_LEVEL,
18  )
```

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.

Extending the gem5 library

```
21  cache_hierarchy = MESITwoLevelCacheHierarchy(
22      l1d_size="32KiB",
23      l1d_assoc=8,
24      l1i_size="32KiB",
25      l1i_assoc=8,
26      l2_size="256kB",
27      l2_assoc=16,
28      num_l2_banks=1,
29  )
```

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

Extending the gem5 library

```
43    processor = SimpleSwitchableProcessor(
44        starting_core_type=CPUTypes.TIMING,
45        switch_core_type=CPUTypes.O3,
46        num_cores=2,
47    )
```

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.

Extending the gem5 library

```
50  board = X86Board(
51      clk_freq="3GHz",
52      processor=processor,
53      memory=memory,
54      cache_hierarchy=cache_hierarchy,
55  )
```

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

Extending the gem5 library

```
19  command = "m5 exit;" \
20          + "echo 'This is running on Timing CPU cores.';" \
21          + "sleep 1;" \
22          + "m5 exit;"
23
24  board.set_kernel_disk_workload(
25      kernel=Resource("x86-linux-kernel-5.4.49",),
26      disk_image=Resource("x86-ubuntu-18.04-img"),
27      readfile_contents=command,
28  )
```

The 'set_kernel_disk_workload` 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 value of

Extending the gem5 library

```
#!/bin/bash

# Copyright (c) 2021 The University of Texas at Austin.
# SPDX-License-Identifier: BSD 3-Clause

echo "Starting gem5 init... reading run script file."
if ! m5 readfile > /tmp/script; then
    echo "Failed to run m5 readfile, exiting!"
    rm -f /tmp/script
    if ! m5 exit; then
        # Useful for booting the disk image in (e.g.,) qemu for debugging
        echo "m5 exit failed, dropping to shell."
        /bin/sh
    fi
else
    echo "Running m5 script from /tmp/script"
    chmod 755 /tmp/script
    /tmp/script
    echo "Done running script, exiting."
    rm -f /tmp/script
    m5 exit
fi
```

Here is what's being run when the disk image is booted.

If `m5 readfile` returns a script, it's executed. Otherwise `m5 exit` is called.

The Simulator Module

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

```
19  command = "m5 exit;" \
20          + "echo 'This is running on Timing CPU cores.';" \
21          + "sleep 1;" \
22          + "m5 exit;"
```

During a simulation you can have "Exit Events".

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

The Simulator Module

```
31  simulator = Simulator(board=board)
32
33  simulator.run() # Runs up to the first `m5 exit`` event`
34
35  # Here we can do things between the exit event
36
37  processor.switch()
38
39  simulator.run() # Run up to the final `m5 exit` event
```

Here we can run up to an exit event, do things, and then continue the run.

In this case we want to switch the CPU cores.

The Simulator Module: We can do better

```
13  simulator = Simulator(
14      board=board,
15      on_exit_event={
16          ExitEvent.EXIT : (func() for func in [processor.switch]),
17      },
18  )
19  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.

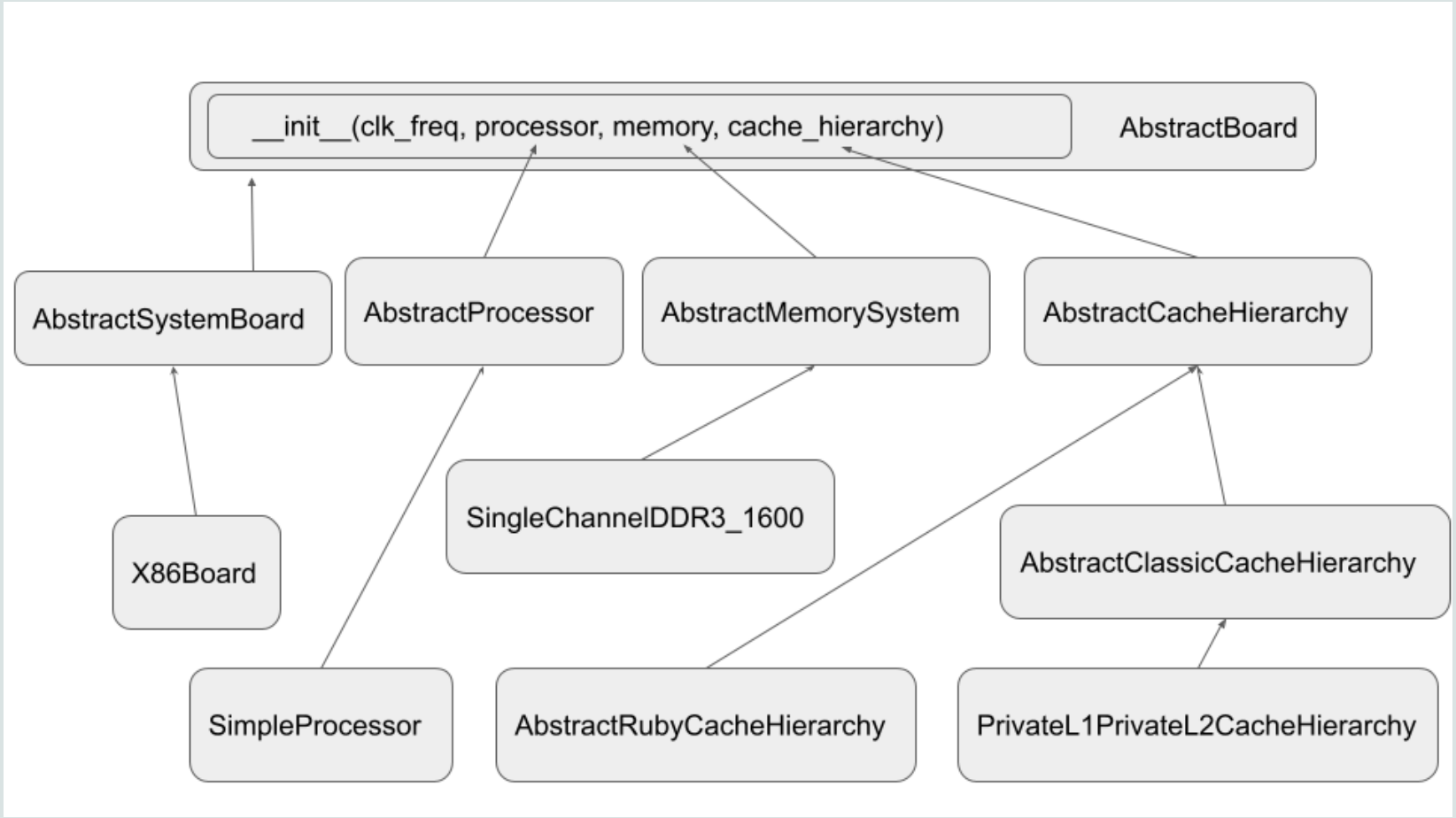
- ExitEvent.EXIT
- ExitEvent.CHECKPOINT
- ExitEvent.FAIL
- ExitEvent.SWITCHCPU
- ExitEvent.WORKBEGIN
- ExitEvent.WORKEND
- ExitEvent.USER_INTERRUPT
- ExitEvent.MAX_TICK

Your done! You can now run your full-system simulation

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

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

Expanding your design



Expanding your design

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

```
1  from gem5.components.cachehierarchies.classic.abstract_classic_cache_hierarchy \
2      import AbstractClassicCacheHierarchy
3  from gem5.components.boards.abstract_board import AbstractBoard
4
5  from m5.objects import Port
6
7  class UniqueCacheHierarchy(AbstractClassicCacheHierarchy):
8
9
10     def __init__(self) -> None:
11         AbstractClassicCacheHierarchy.__init__(self=self)
12
13     def get_mem_side_port(self) -> Port:
14         pass
15
16     def get_cpu_side_port(self) -> Port:
17         pass
18
19     def incorporate_cache(self, board: AbstractBoard) -> None:
20         pass
```

Expanding your design

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

```
10      def __init__(self) -> None:
11          AbstractClassicCacheHierarchy.__init__(self=self)
12          self.membus =  SystemXBar(width=64)
13          self.membus.badaddr_responder = BadAddr()
14          self.membus.default = self.membus.badaddr_responder.pio
15
16      def get_mem_side_port(self) -> Port:
17          return self.membus.mem_side_ports
18
19      def get_cpu_side_port(self) -> Port:
20          return self.membus.cpu_side_ports
```

Expanding your design

Let's add a function to create an IO cache

```
25      def _setup_io_cache(self, board: AbstractBoard) -> None:
26          """Create a cache for coherent I/O connections"""
27          self.iocache = Cache(
28              assoc=8,
29              tag_latency=50,
30              data_latency=50,
31              response_latency=50,
32              mshrs=20,
33              size="1kB",
34              tgts_per_mshr=12,
35              addr_ranges=board.mem_ranges,
36          )
37          self.iocache.mem_side = self.membus.cpu_side_ports
38          self.iocache.cpu_side = board.get_mem_side_coherent_io_port()
```

Expanding your design

Finally, let's implement "incorporate_cache"

```
23      def incorporate_cache(self, board: AbstractBoard) -> None:
24          # Set up the system port for functional access from the simulator.
25          board.connect_system_port(self.membus.cpu_side_ports)
26
27          for cntr in board.get_memory().get_memory_controllers():
28              cntr.port = self.membus.mem_side_ports
29
30          self.l1icaches = [
31              L1ICache(size="32KiB")
32              for i in range(board.get_processor().get_num_cores())
33          ]
34
35          self.l1dcaches = [
36              L1DCache(size="32KiB")
37              for i in range(board.get_processor().get_num_cores())
38          ]
39          # ITLB Page walk caches
40          self.iptw_caches = [
41              MMUCache(size="8KiB") for _ in range(board.get_processor().get_num_cores())
42          ]
43          # DTLB Page walk caches
44          self.dptw_caches = [
45              MMUCache(size="8KiB") for _ in range(board.get_processor().get_num_cores())
46          ]
```

Expanding your design

```
48          if board.has_coherent_io():
49              self._setup_io_cache(board)
50
51          for i, cpu in enumerate(board.get_processor().get_cores()):
52
53              cpu.connect_icache(self.l1icaches[i].cpu_side)
54              cpu.connect_dcache(self.l1dcaches[i].cpu_side)
55
56              self.l1icaches[i].mem_side = self.membus.cpu_side_ports
57              self.l1dcaches[i].mem_side = self.membus.cpu_side_ports
58
59              self.iptw_caches[i].mem_side = self.membus.cpu_side_ports
60              self.dptw_caches[i].mem_side = self.membus.cpu_side_ports
61
62              cpu.connect_walker_ports(
63                  self.iptw_caches[i].cpu_side, self.dptw_caches[i].cpu_side
64              )
65
66              int_req_port = self.membus.mem_side_ports
67              int_resp_port = self.membus.cpu_side_ports
68              cpu.connect_interrupt(int_req_port, int_resp_port)
```

Expanding your design

To use this code, a user can import it as they would any other Python module.

As long as this code is in gem5's python search path, you can import it.

You can also add:
`import sys; sys.path.append(<path to new component>)`
at the beginning of your gem5 runscript to add the path of this new component to the python search path.

Try using this cache with your "hello-world.py" script