All things SimObject

A presentation by

Mahyar Samani

UCDAVIS COMPUTER SCIENCE

DArchR DAVIS ARCHITECTURE RESEARCH

Let's start with compiling :D

- Run the following command in gem5:
- "scons build/NULL/gem5.opt -j$(nproc)"

SimObject and Clocked Object

- Almost all the objects in the gem5 code base are SimObjects.
- A SimObject represents things that correspond to physical components and can be specified and instantiated via the config file (CPUs, caches, etc.).
- A ClockedObject extends the SimObject with a clock and accessor functions (nextCycle, clockEdge) to relate ticks (the unit of time in simulation) to its cycles.

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

Let's copy the boilerplates

- Run the following command in gem5-bootcamp-env:
- "cp -r materials/developing-gem5-models/02-simobj/bootcamp gem5/src/"

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_header: The filename for the C++ header file

cxx_class: The fully qualified C++ class name

gem5

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;
}
} //
```

gem5

Step 3: Register the SimObject and C++ file

Import: SConscript is just Python… but weird.

SConscript

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

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

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

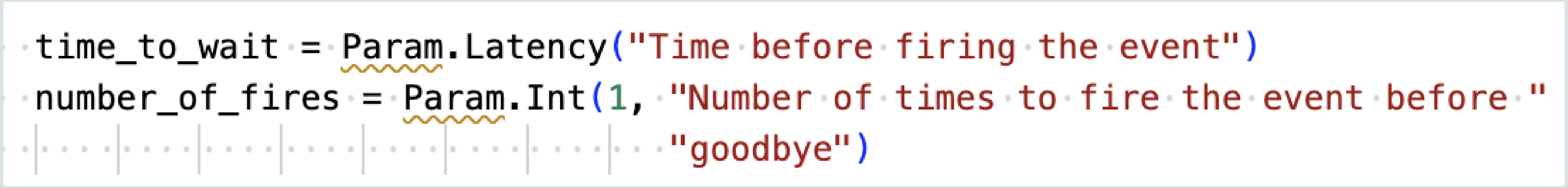
sim_objects: The SimObjects declared in the file (could be more than 1)

gem5

Step 4: (Re-)build gem5

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

Adding Parameters and events to the SimObject

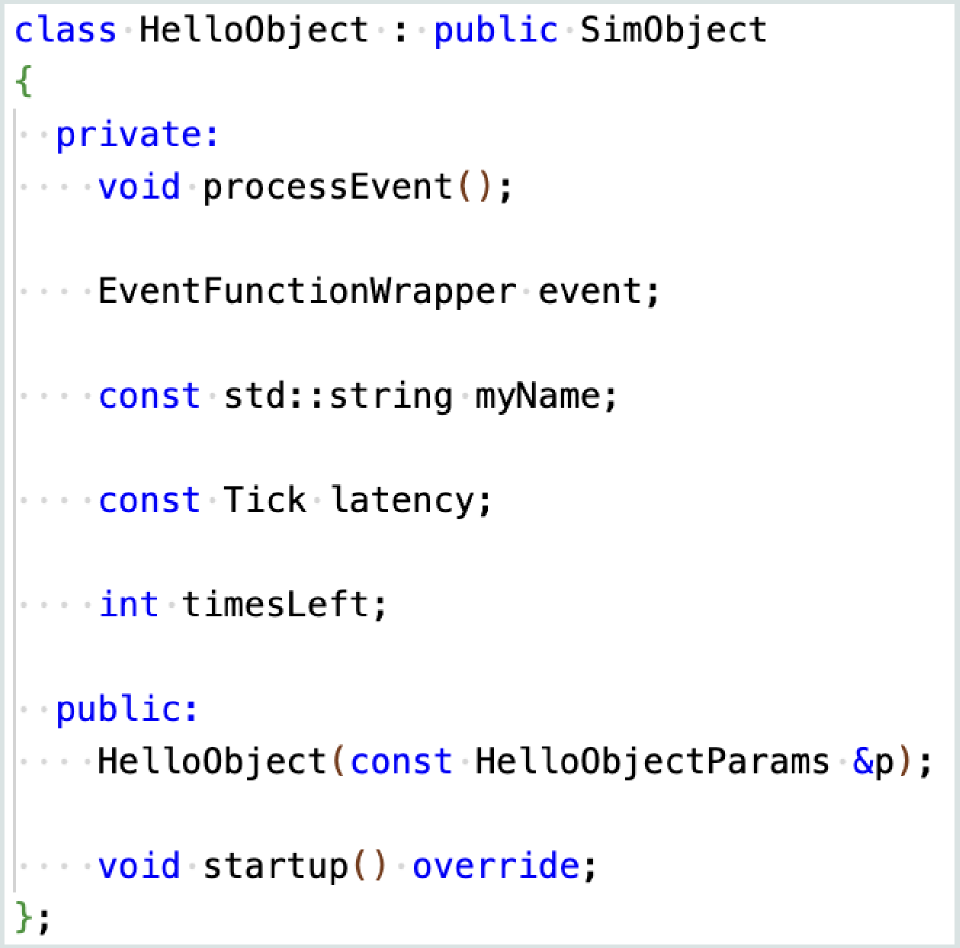


```
    time_to_wait = Param.Latency("Time before firing the event")
    number_of_fires = Param.Int(1, "Number of times to fire the event before "
                                "goodbye")
```

Add this to “HelloObject.py”.

This declares the parameters of the SimObject.

Adding Parameters and events to the SimObject



```cpp
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;
};
```

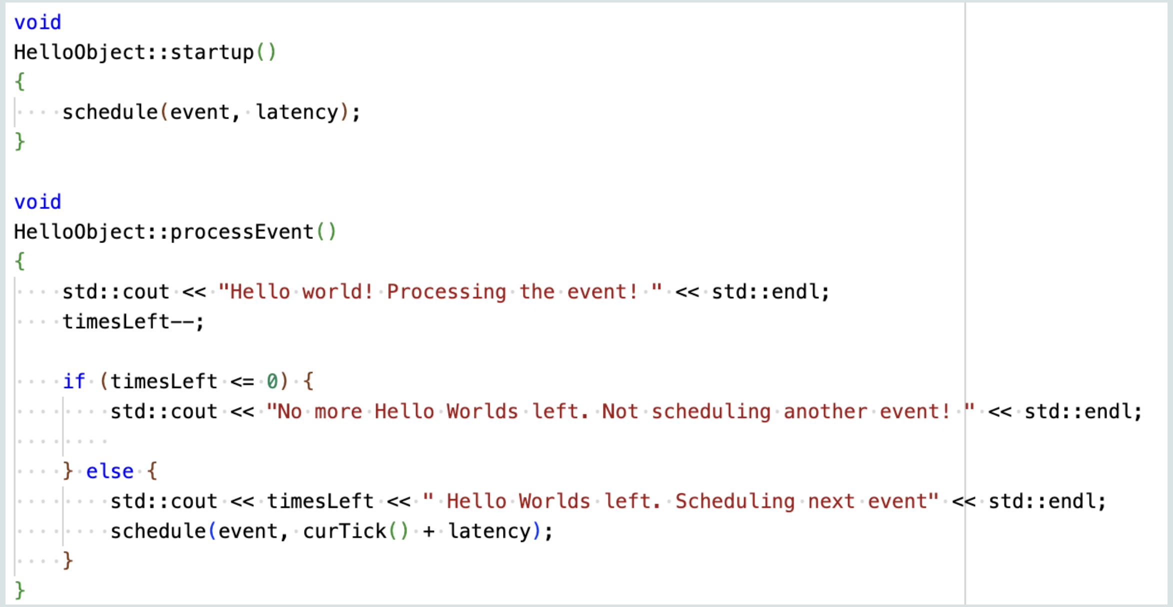
Update the “hello_object.hh” with the parameters and the event variables/functions.

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



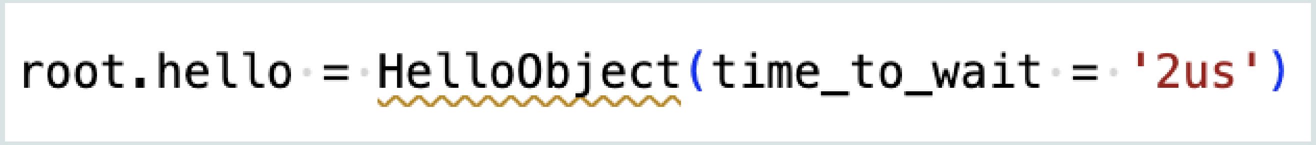
```
void
HelloObject::startup()
{
    schedule(event, latency);
}

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



```python
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.

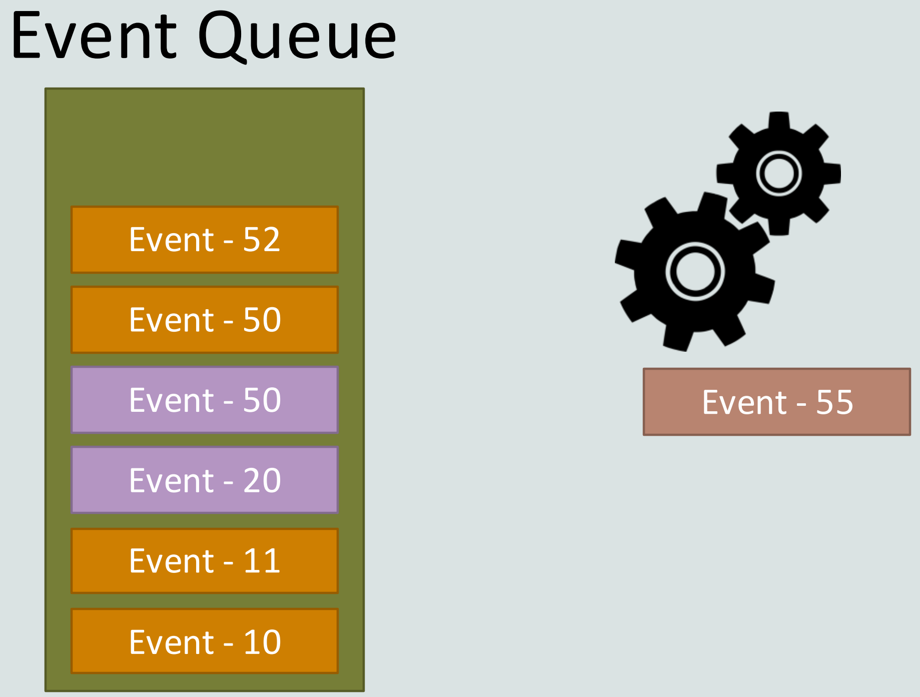
A little more to do in your own time

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 your interested in SimObjects and creating your own this is worth doing..

gem5 architecture: Simulating

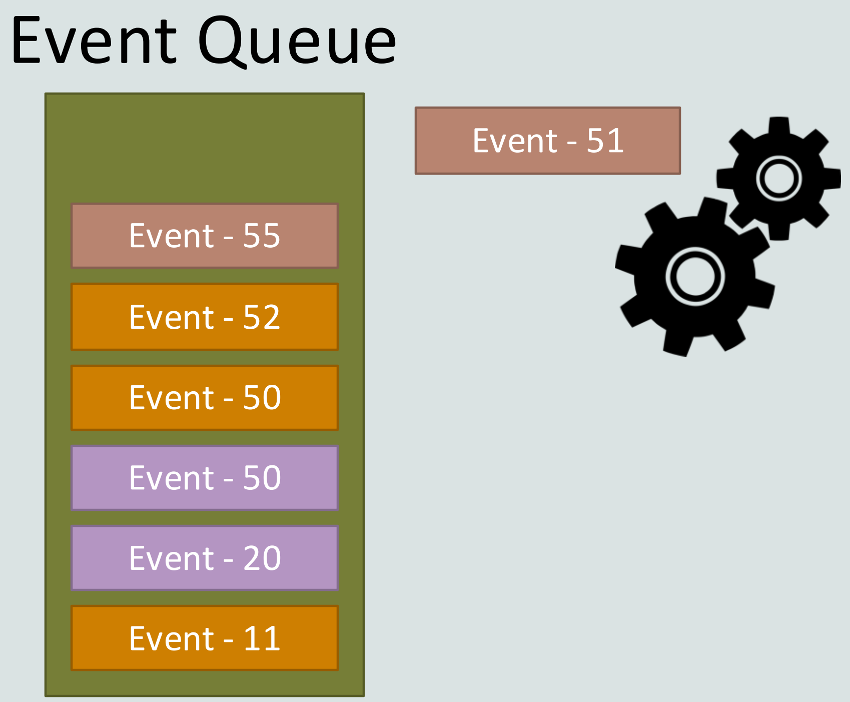
gem5 is a **discrete event simulator**



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

gem5 architecture: Simulating

gem5 is a **discrete event simulator**

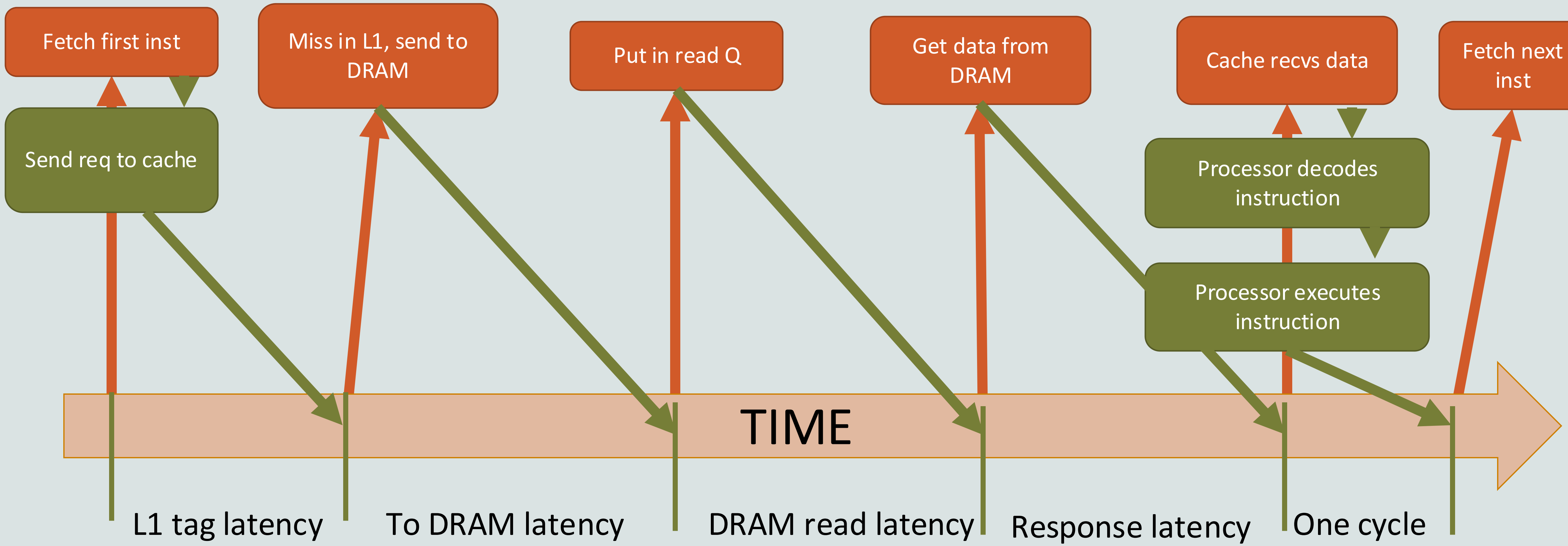


1) Event at head dequeued
2) Event executed
3) More events queued

We'll cover more later

All SimObjects can enqueue events to the event queue

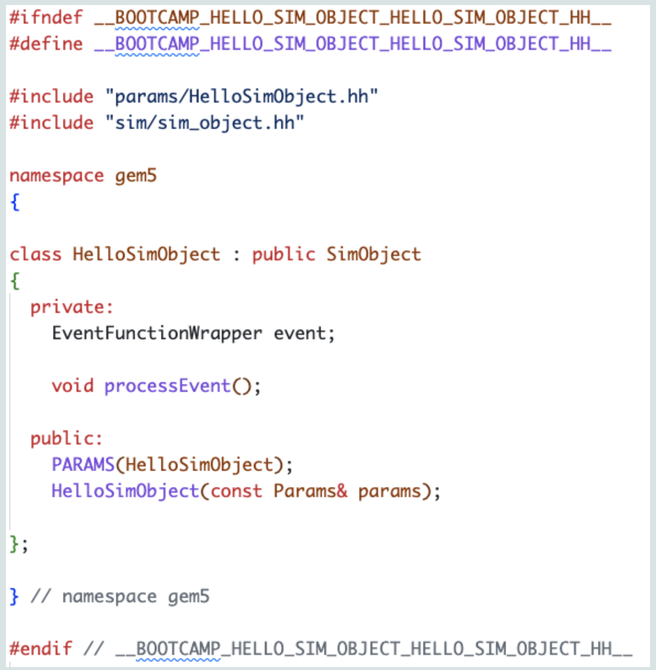
Discrete event simulation example



Let’s talk about events

Events at the high level represent different types of interactions within/between SimObjects. Each event executes a function that is called at the tick when the respective event is scheduled.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.hh"

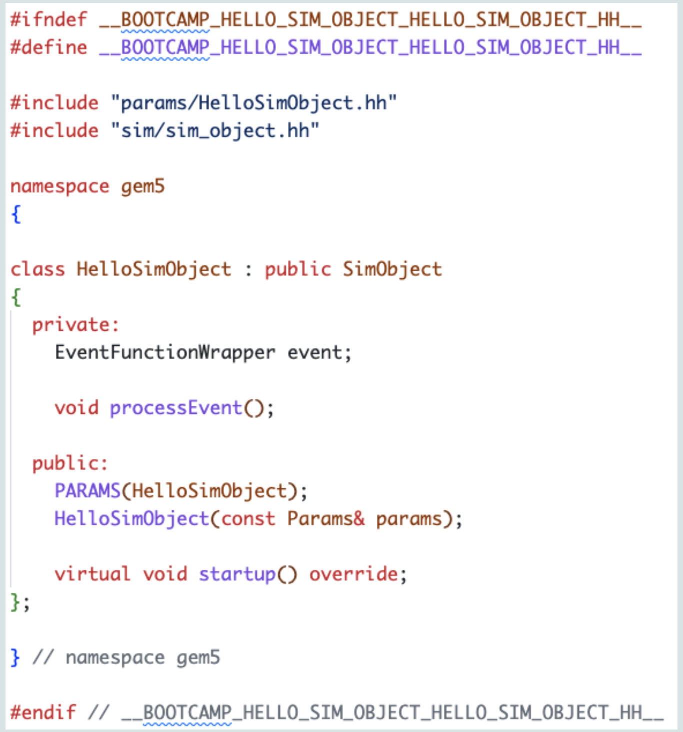


```
#ifndef __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
#define __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__

#include "params/HelloSimObject.hh"
#include "sim/sim_object.hh"

namespace gem5
{

class HelloSimObject : public SimObject
{
  private:
    EventFunctionWrapper event;

    void processEvent();

  public:
    PARAMS(HelloSimObject);
    HelloSimObject(const Params& params);

};

} // namespace gem5

#endif // __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
```

virtual void SimObject::startup()

Final initialization call before simulation starts. All state is initialized. This function is the correct place to schedule initial events.

"gem5/src/bootcamp/hello-sim-object/hello_sim_object.hh"



```
#ifndef __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
#define __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__

#include "params/HelloSimObject.hh"
#include "sim/sim_object.hh"

namespace gem5
{

class HelloSimObject : public SimObject
{
  private:
    EventFunctionWrapper event;

    void processEvent();

  public:
    PARAMS(HelloSimObject);
    HelloSimObject(const Params& params);

    virtual void startup() override;
};

} // namespace gem5

#endif // __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
```

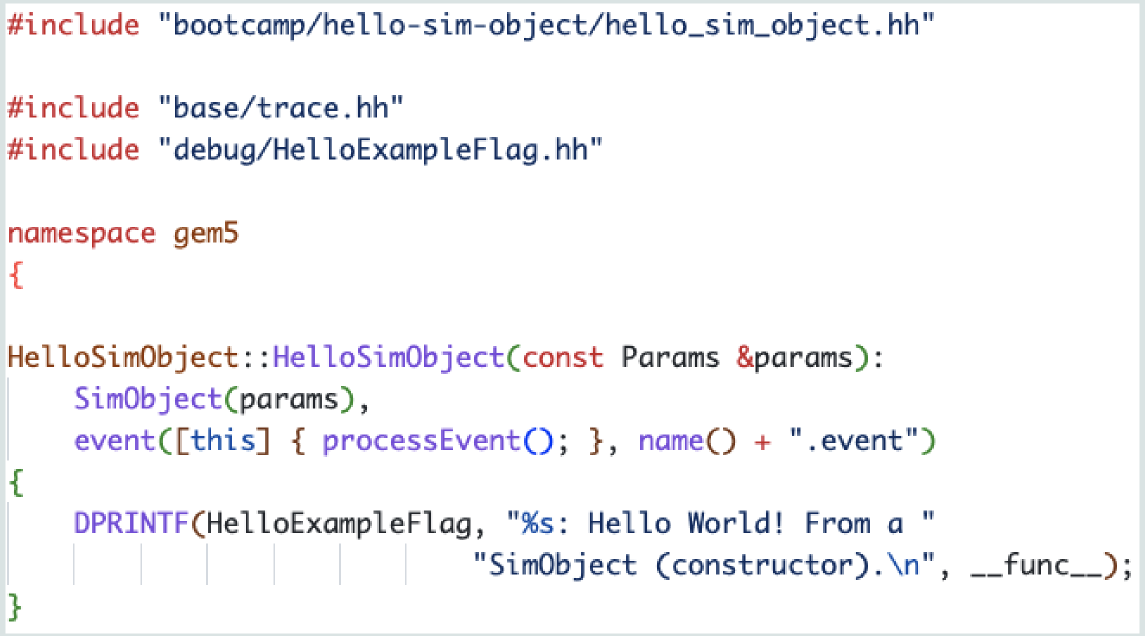
Other initialization functions

- virtual void SimObject::init(): First initialization call. All SimObjects are instantiated, and all ports are connected.
- virtual void SimObject::initState(): Called after init() only when simulating afresh. i.e. not called when restoring a checkpoint
- virtual void SimObject::loadState() Called after init() and only when restoring from a checkpoint.

Let’s code

We will look at an example on how to initialize an event. Then students will follow as I change the constructor of HelloObject to initialize event.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"



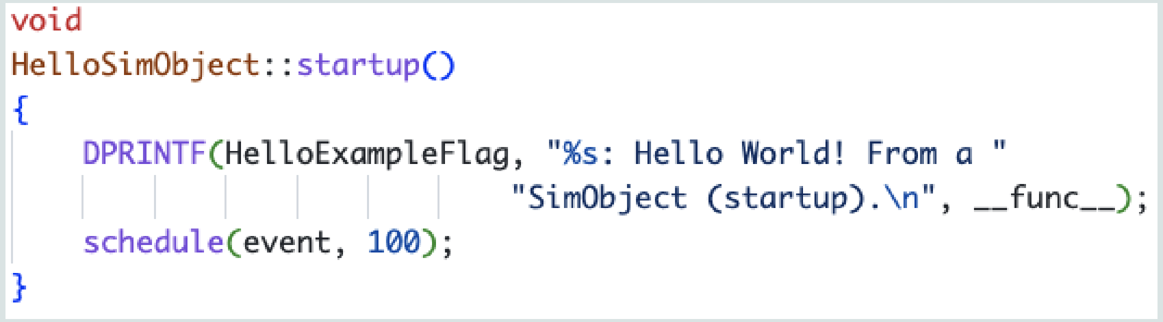
```
#include "bootcamp/hello-sim-object/hello_sim_object.hh"

#include "base/trace.hh"
#include "debug/HelloExampleFlag.hh"

namespace gem5
{

HelloSimObject::HelloSimObject(const Params &params):
    SimObject(params),
    event([this] { processEvent(); }, name() + ".event")
{
    DPRINTF(HelloExampleFlag, "%s: Hello World! From a "
                            "SimObject (constructor).\n", __func__);
}
```

Schedule(...)

Function inherited from EventManager (SimObject is EventManager). Schedules an event on a specific tick.

Args: Event*, Tick: takes absolute time in ticks.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"



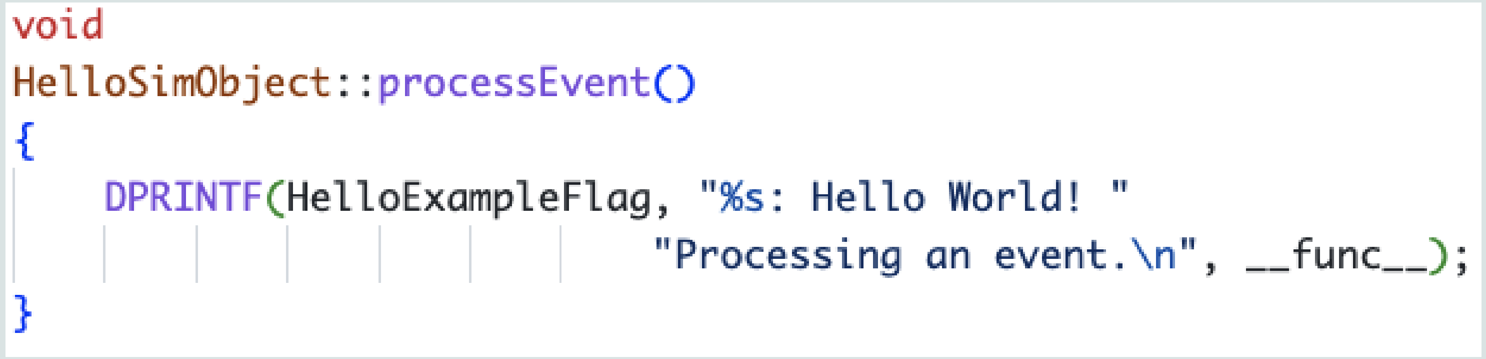
```
void
HelloSimObject::startup()
{
    DPRINTF(HelloExampleFlag, "%s: Hello World! From a "
                              "SimObject (startup).\n", __func__);
    schedule(event, 100);
}
```

Let’s code

Now, we have to implement the callback function for our event.

Students will follow as I implement processEvent.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"



```
void
HelloSimObject::processEvent()
{
    DPRINTF(HelloExampleFlag, "%s: Hello World! "
                              "Processing an event.\n", __func__);
}
```

Let's recompile

- Run the following command in gem5:
- "scons build/NULL/gem5.opt -j$(nproc)"

Let’s sim

Run the following commands in gem5:

1- "build/NULL/gem5.opt src/bootcamp/hello-sim-object/run_hello.py"

2- "build/NULL/gem5.opt --debug-flags=HelloExampleFlag src/bootcamp/hello-sim-object/run_hello.py"

Let’s sim

Run the following commands in gem5:

1- "build/NULL/gem5.opt src/bootcamp/hello-sim-object/run_hello.py"

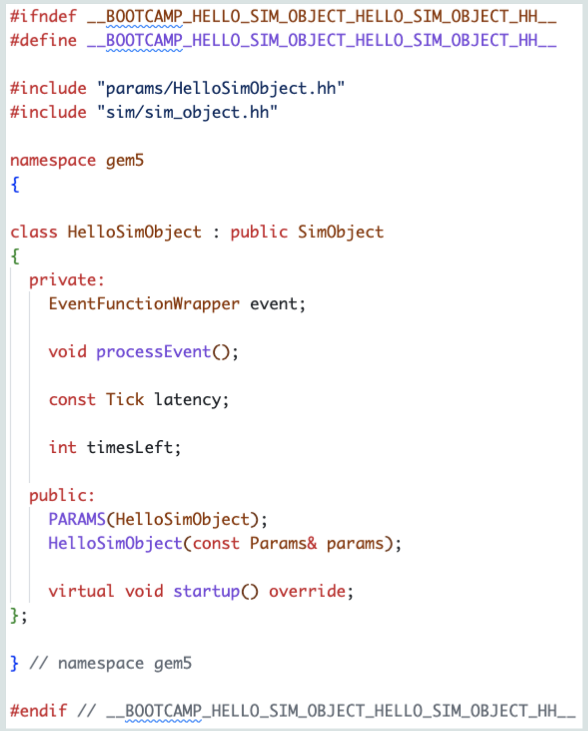
2- "build/NULL/gem5.opt --debug-flags=HelloExampleFlag src/bootcamp/hello-sim-object/run_hello.py"

Do you see a difference in the outputs?

Let’s make it more interesting.

Let’s make our SimObject print “Hello world! Processing the event!” *n times, every L ticks.*

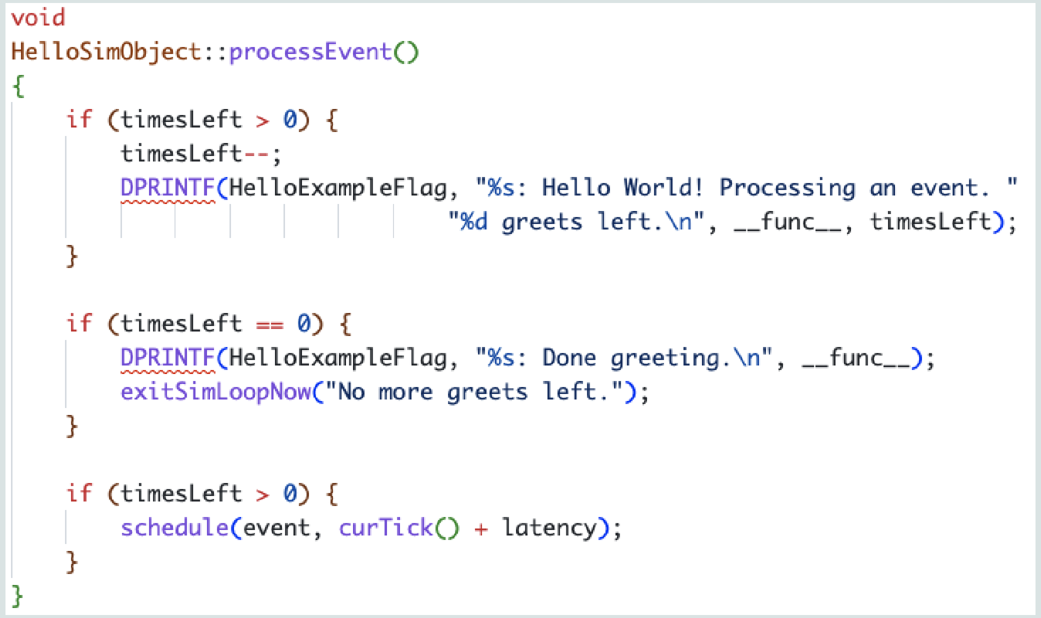
"gem5/src/bootcamp/hello-sim-object/hello_sim_object.hh"



```
#ifndef __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
#define __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__

#include "params/HelloSimObject.hh"
#include "sim/sim_object.hh"

namespace gem5
{

class HelloSimObject : public SimObject
{
  private:
    EventFunctionWrapper event;

    void processEvent();

    const Tick latency;

    int timesLeft;

  public:
    PARAMS(HelloSimObject);
    HelloSimObject(const Params& params);

    virtual void startup() override;
};

} // namespace gem5

#endif // __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
```

Let’s make it more interesting.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"



```
void
HelloSimObject::processEvent()
{
    if (timesLeft > 0) {
        timesLeft--;
        DPRINTF(HelloExampleFlag, "%s: Hello World! Processing an event. "
                                  "%d greets left.\n", __func__, timesLeft);
    }

    if (timesLeft == 0) {
        DPRINTF(HelloExampleFlag, "%s: Done greeting.\n", __func__);
        exitSimLoopNow("No more greets left.");
    }

    if (timesLeft > 0) {
        schedule(event, curTick() + latency);
    }
}
```

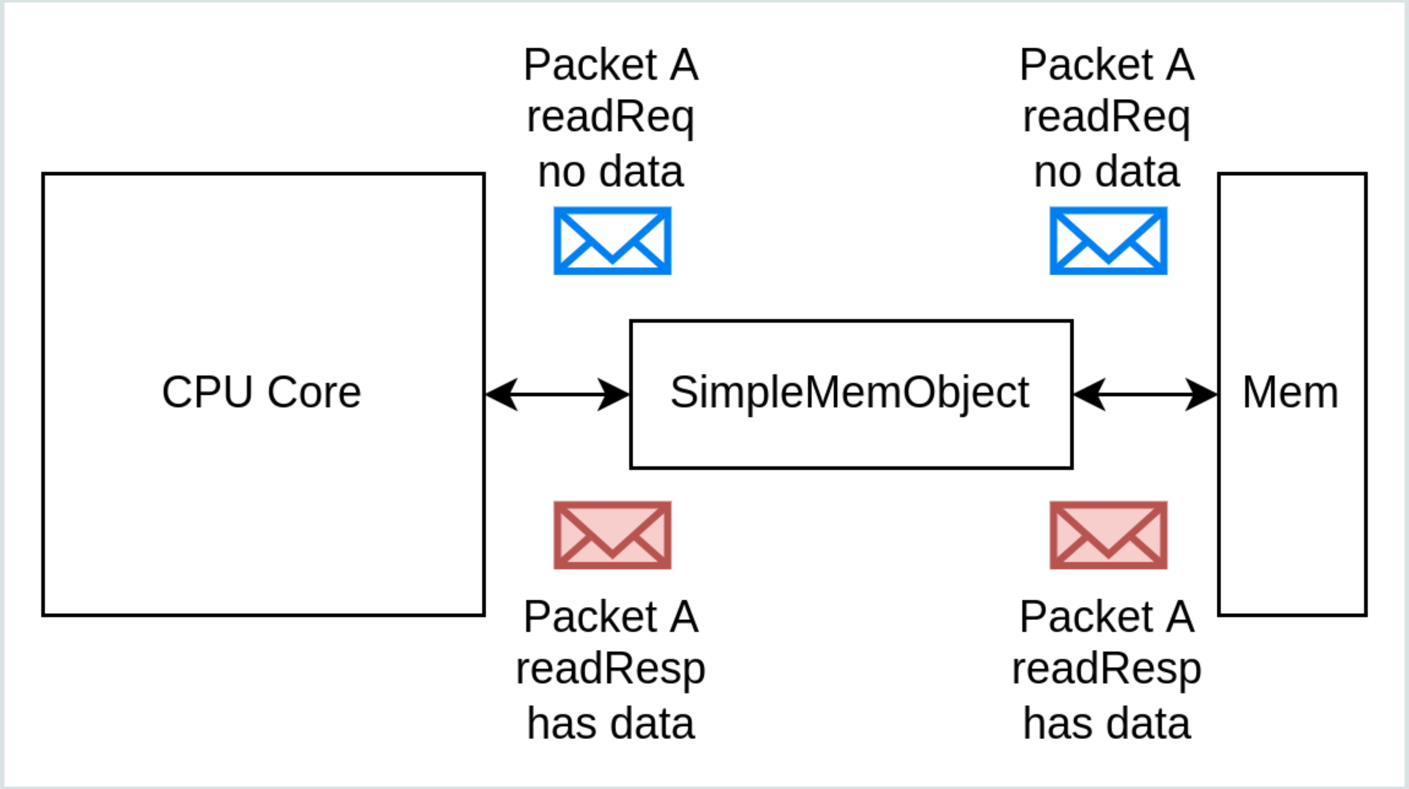
Interacting with memory

- Let's build a simple memory object with the following specs:
- Sits between CPU and memory. Forwards requests from CPU to memory and responses from memory to CPU one **Packet** at a time.
- Separate interface for instruction and data requests.

Packets

- Encapsulation of information required to interact with memory. Some included info are:
- MemCmd: readReq, readResp, writeReq, writeResp
- RequestorID: ID for the Requestor SimObject.
- Addr: Address of the data requested.
- Data: Depending on the MemCmd packet might have data. e.g. Request packets for write, and response for read.

A high-level overview of interacting with memory



Packets are moved around through ports.

Ports and Accesses

All memory objects are connected to each other through ports. Ports facilitate the movement of data/information between different objects. There are 3 different types of accesses that ports allow: **timing**, **atomic**, and **functional.**

- **Timing:** timing accesses move the time (simulated interactions take time). They are the only mode of access that result in correct simulation results. (We will focus on this)
- **Atomic:** Used for fast-forwarding. No events are scheduled in the memory system. Memory is accessed through a long chain of function calls.
- **Functional:** Used for debugging purposes. It is used for things like reading data from the host to the simulator.

Request and Response Ports

Request ports facilitate requesting data from another SimObject. Important methods to note:

- sendTimingReq
- recvTimingResp
- recvReqRetry

Response ports provide Request ports with the data requested. Important methods to note:

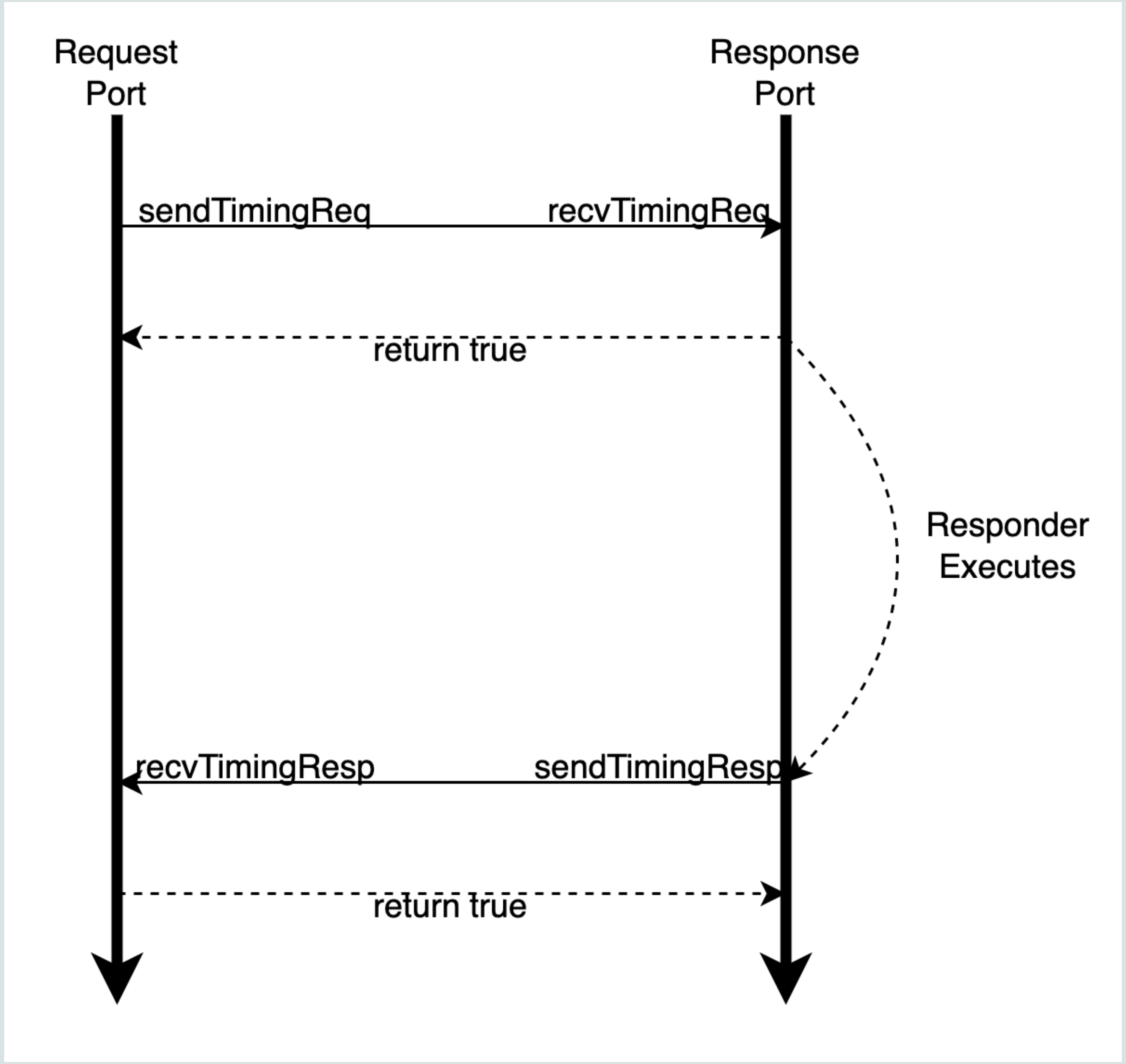
- recvTimingReq
- sendTimingResp
- recvRespRetry

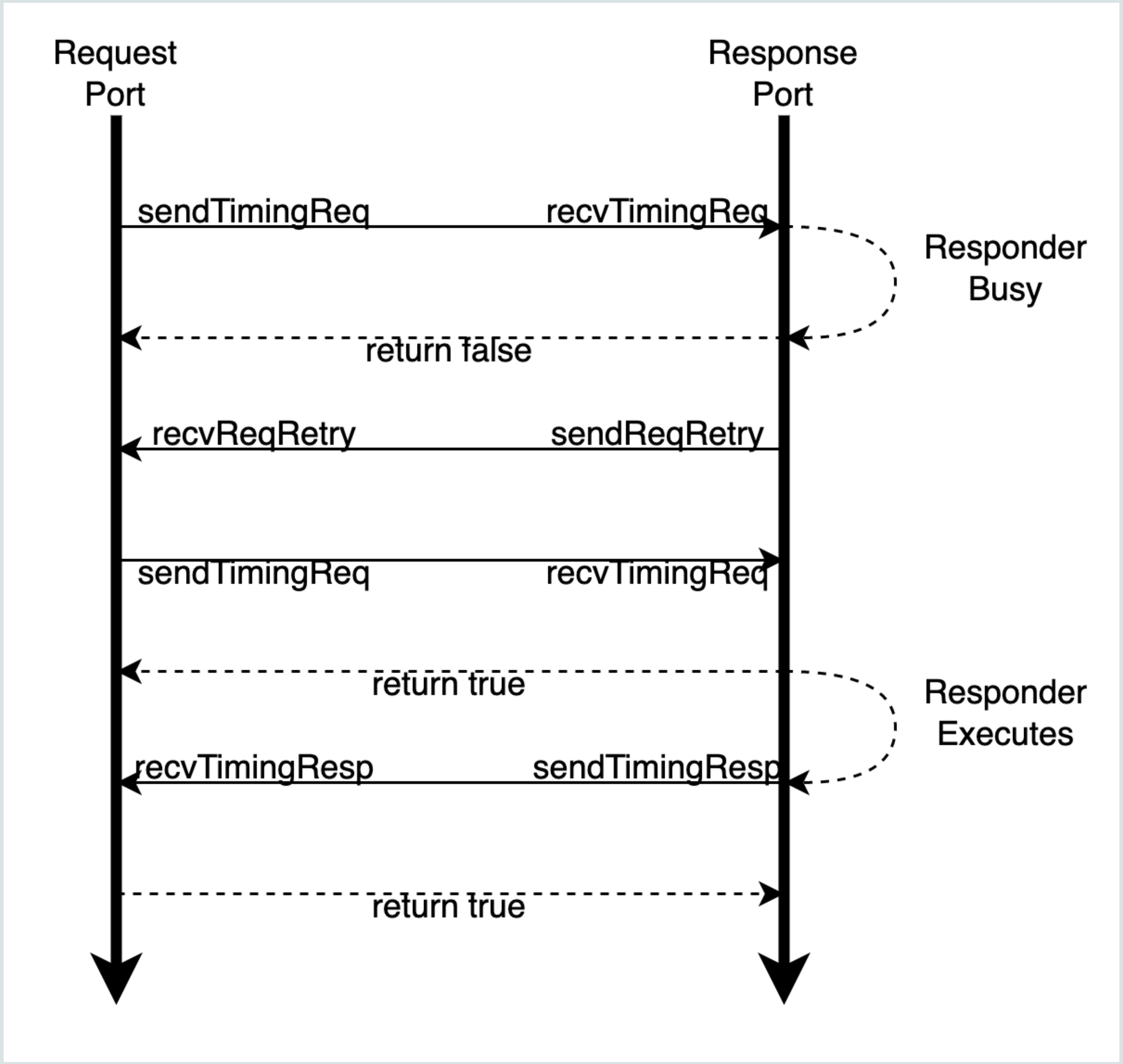
NOTE: Only ports of different types could be connected to each other.

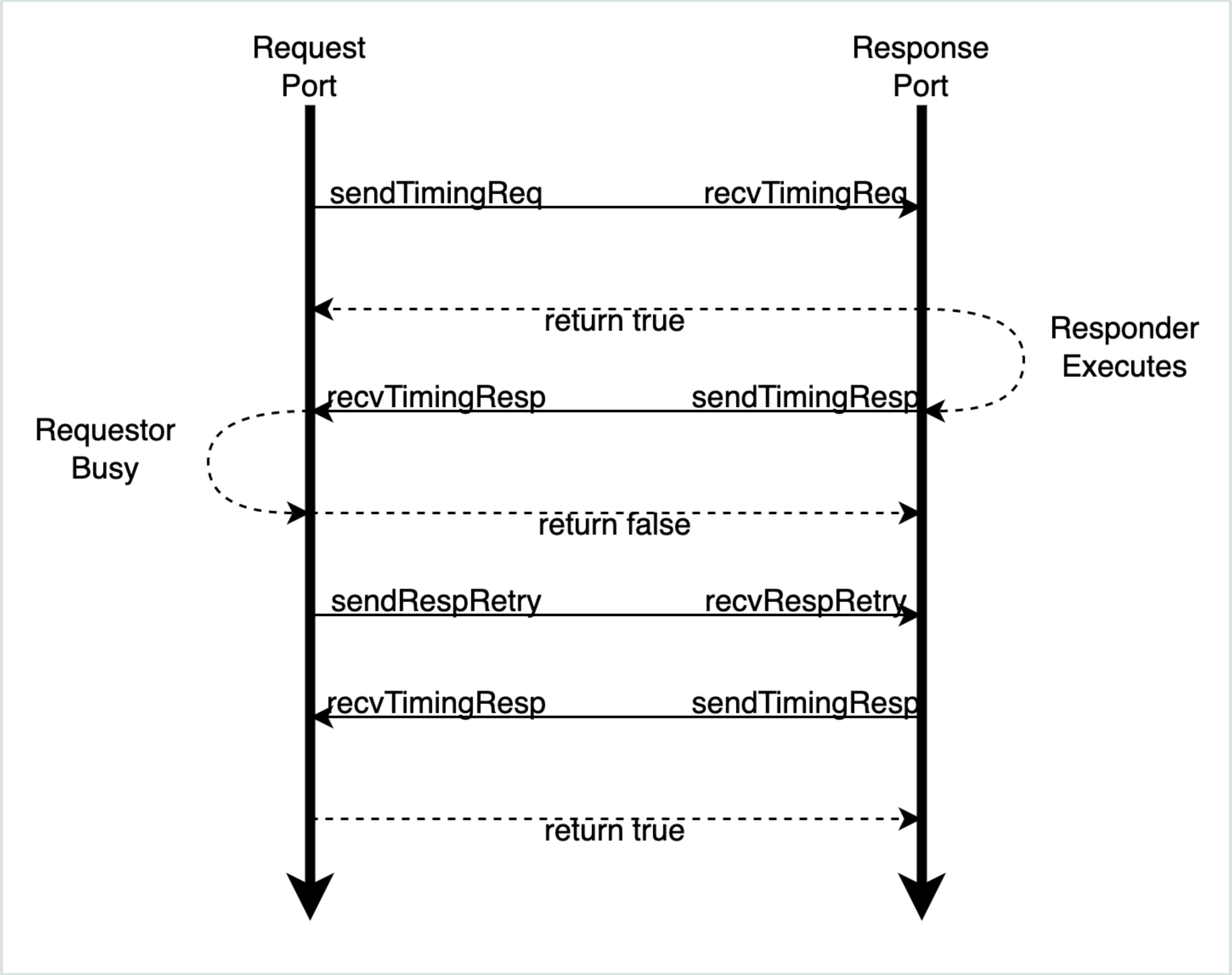
Port Connection and interaction

User needs to connec ports to each other in the python using "=". PyBind takes care of peer ports being connected to each other in C++.

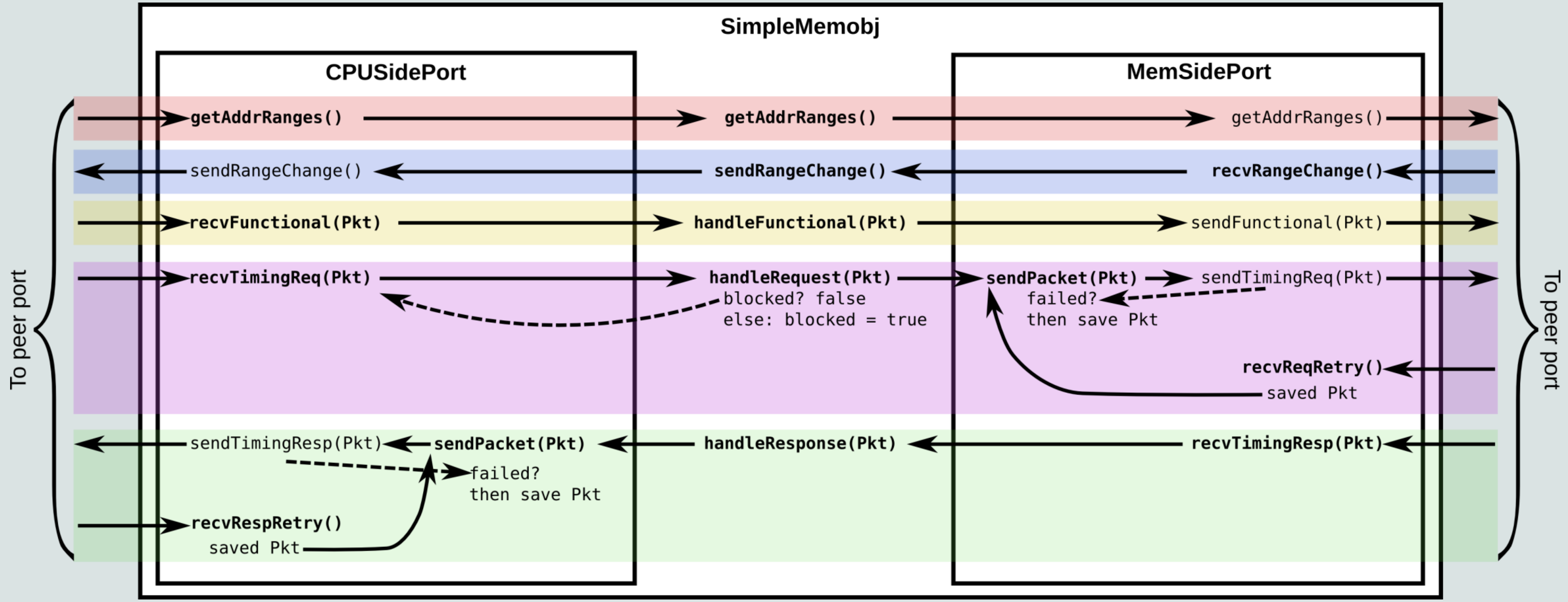
IMPORTANT NOTE: Only ports of opposite type could be connected to each other.


Request Port
Response Port
sendTimingReq
recvTimingReq
return true
Responder Executes
recvTimingResp
sendTimingResp
return true


Request Port
Response Port
sendTimingReq
recvTimingReq
Responder Busy
return false
recvReqRetry
sendReqRetry
sendTimingReq
recvTimingReq
return true
Responder Executes
recvTimingResp
sendTimingResp
return true


Request Port
Response Port
sendTimingReq
recvTimingReq
return true
Responder Executes
recvTimingResp
sendTimingResp
Requestor Busy
return false
sendRespRetry
recvRespRetry
recvTimingResp
sendTimingResp
return true

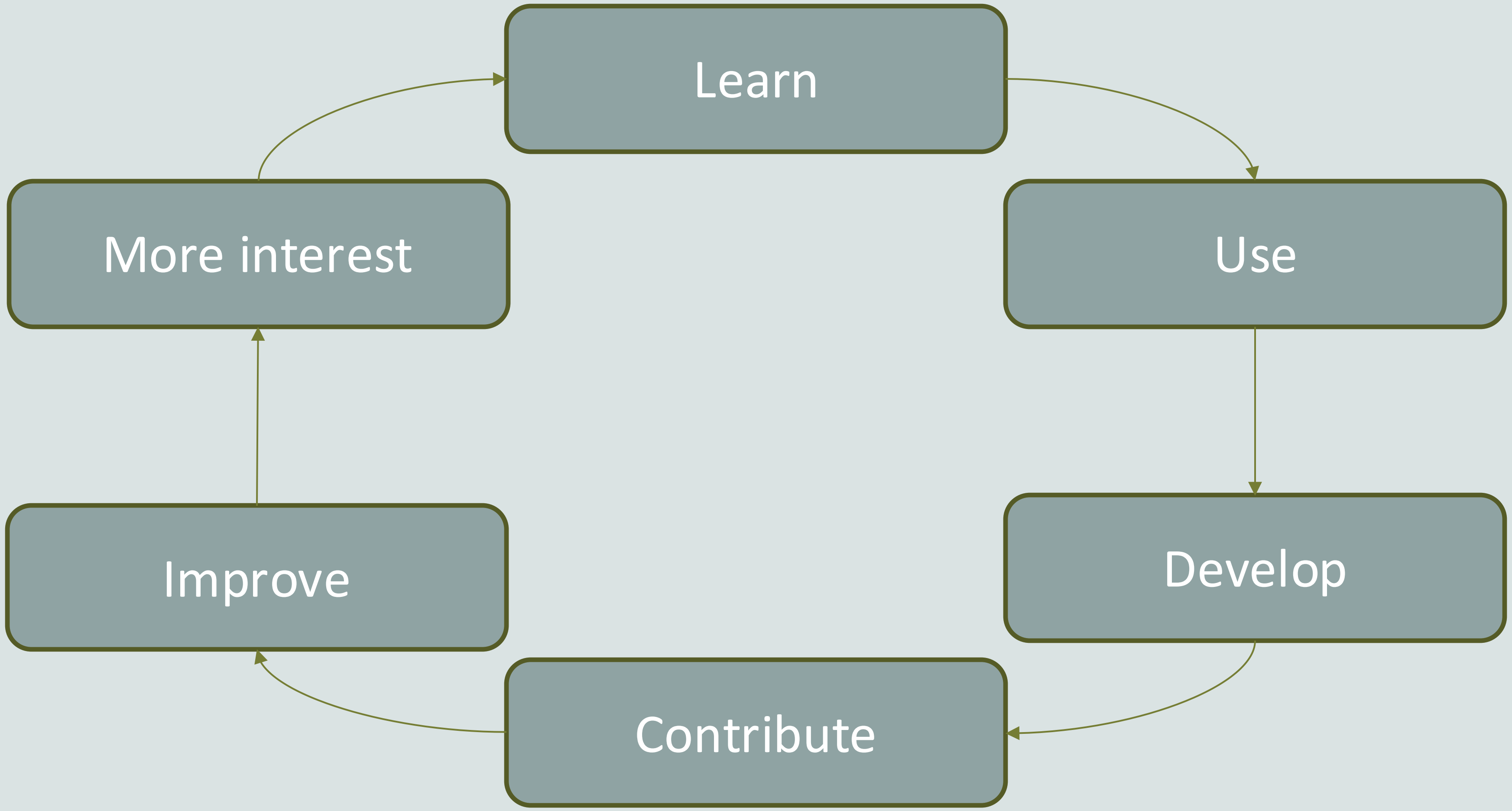
What does it look like?



Lots more to do!

- See materials/…

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.

gem5

“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?

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'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.

3. It doesn't confirm 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

Doxygen 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.

gem5

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

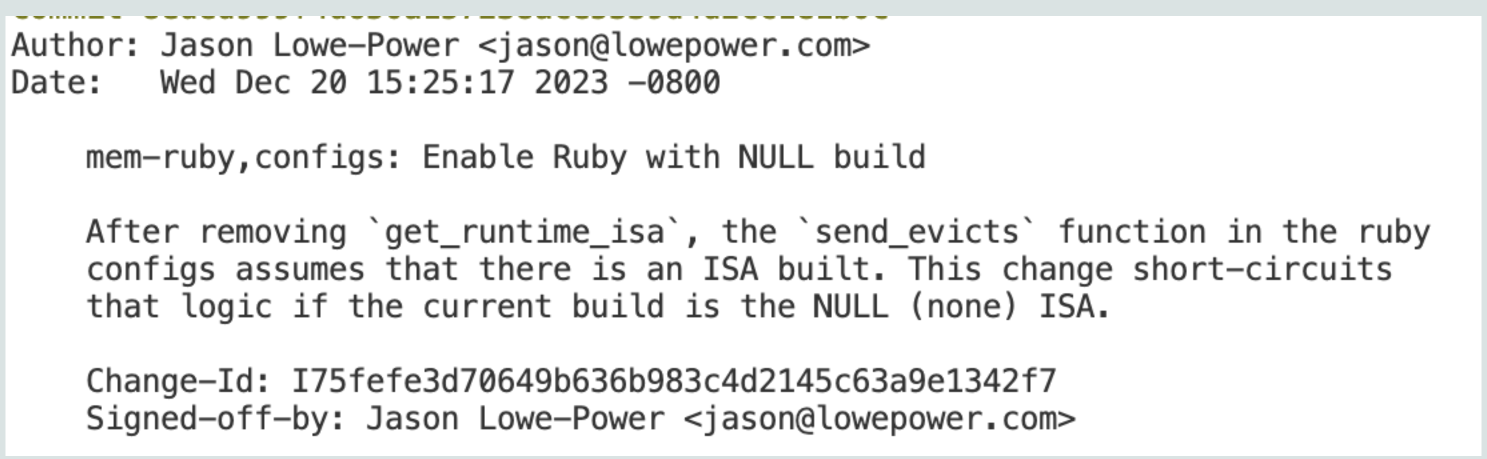
We have some unique rules for gem5:

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



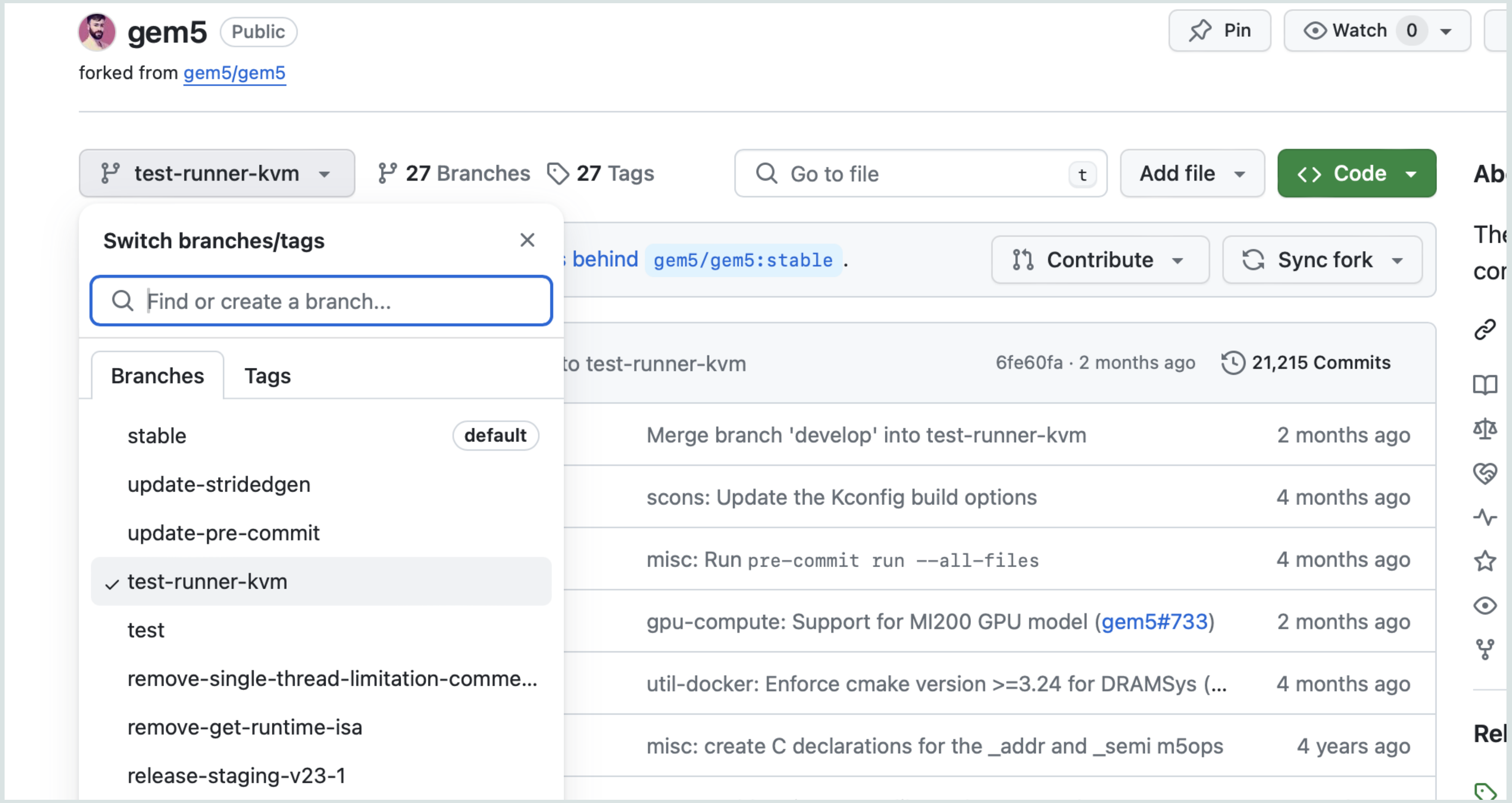
```
Author: Jason Lowe-Power <jason@lowepower.com>
Date:   Wed Dec 20 15:25:17 2023 -0800

    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



gem5 Public

forked from gem5/gem5

Pin | Watch 0

test-runner-kvm | 27 Branches | 27 Tags | Go to file t | Add file | Code

Switch branches/tags

Find or create a branch...

Branches | Tags

- stable default
- update-stridedgen
- update-pre-commit
- ✓ test-runner-kvm
- test
- remove-single-thread-limitation-comme...
- remove-get-runtime-isa
- release-staging-v23-1

behind gem5/gem5:stable.

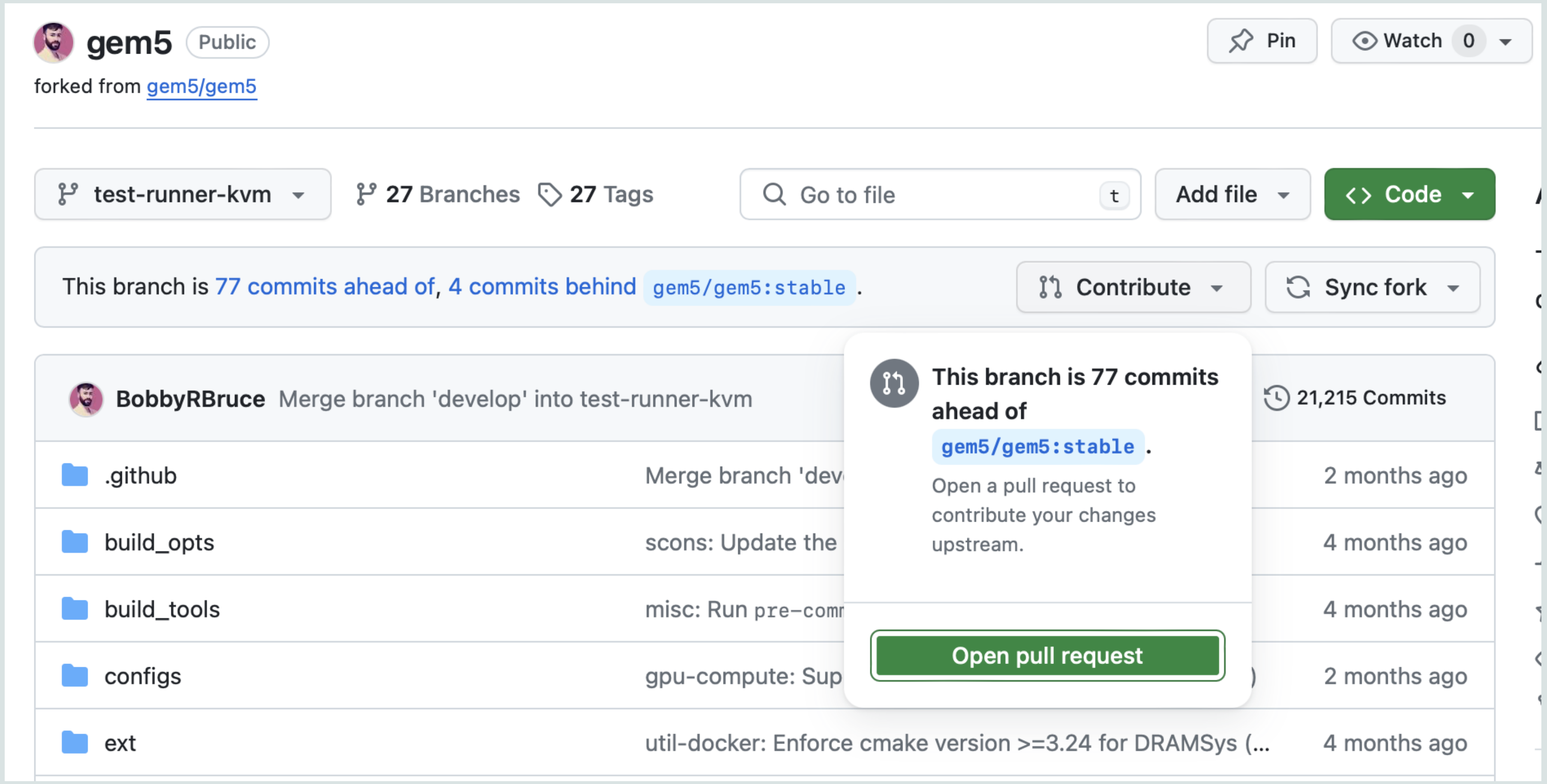
Contribute | Sync fork

to test-runner-kvm 6fe60fa · 2 months ago 21,215 Commits

Merge branch 'develop' into test-runner-kvm	2 months ago
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



This branch is 77 commits ahead of, 4 commits behind gem5/gem5:stable.

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

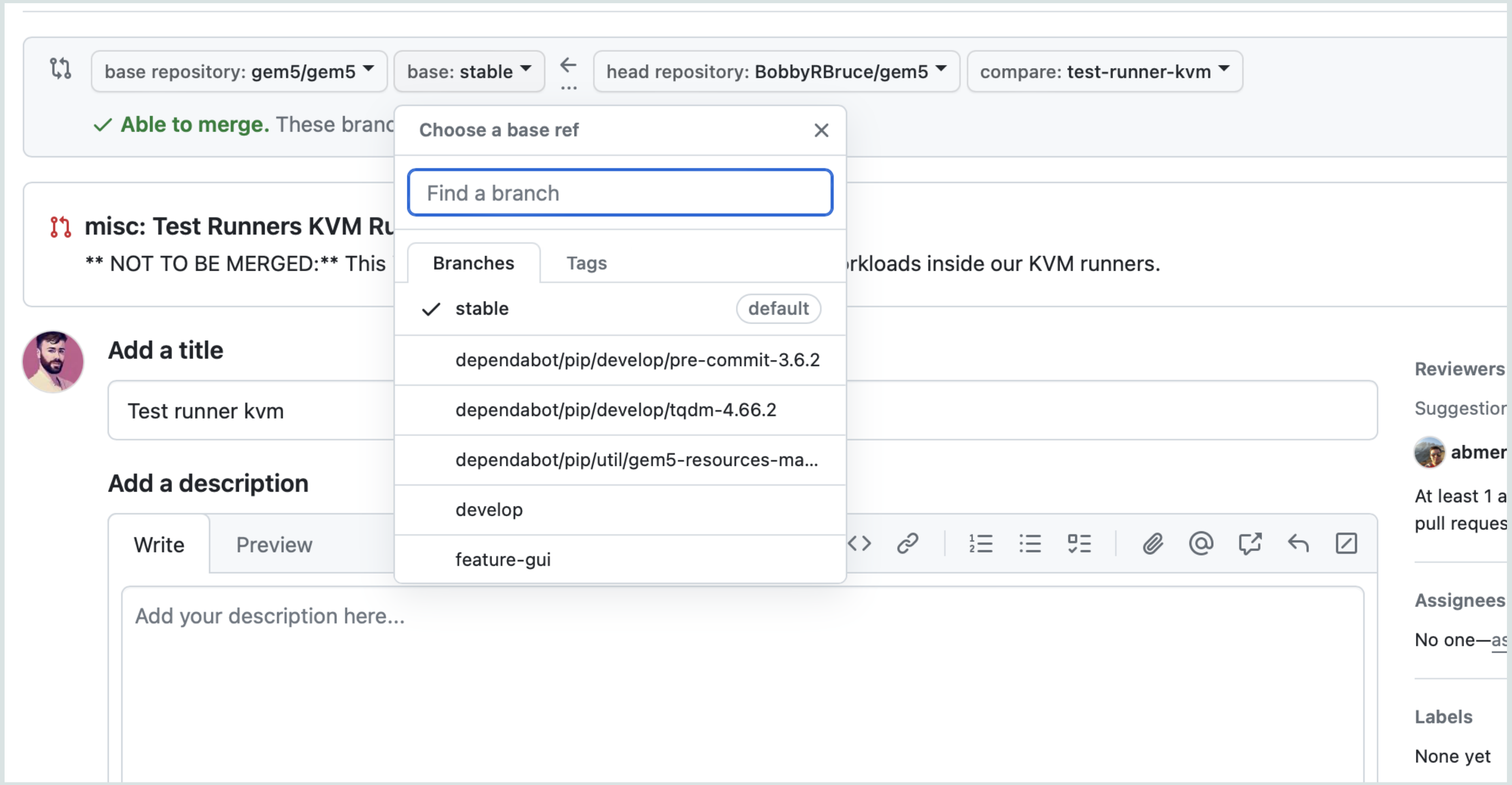
Open a pull request to contribute your changes upstream.

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

.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

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 branc

misc: Test Runners KVM Ru

** NOT TO BE MERGED:** This rkloads inside our KVM runners.

Choose a base ref ✕

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
Suggestion
abmer
At least 1 a
pull reques

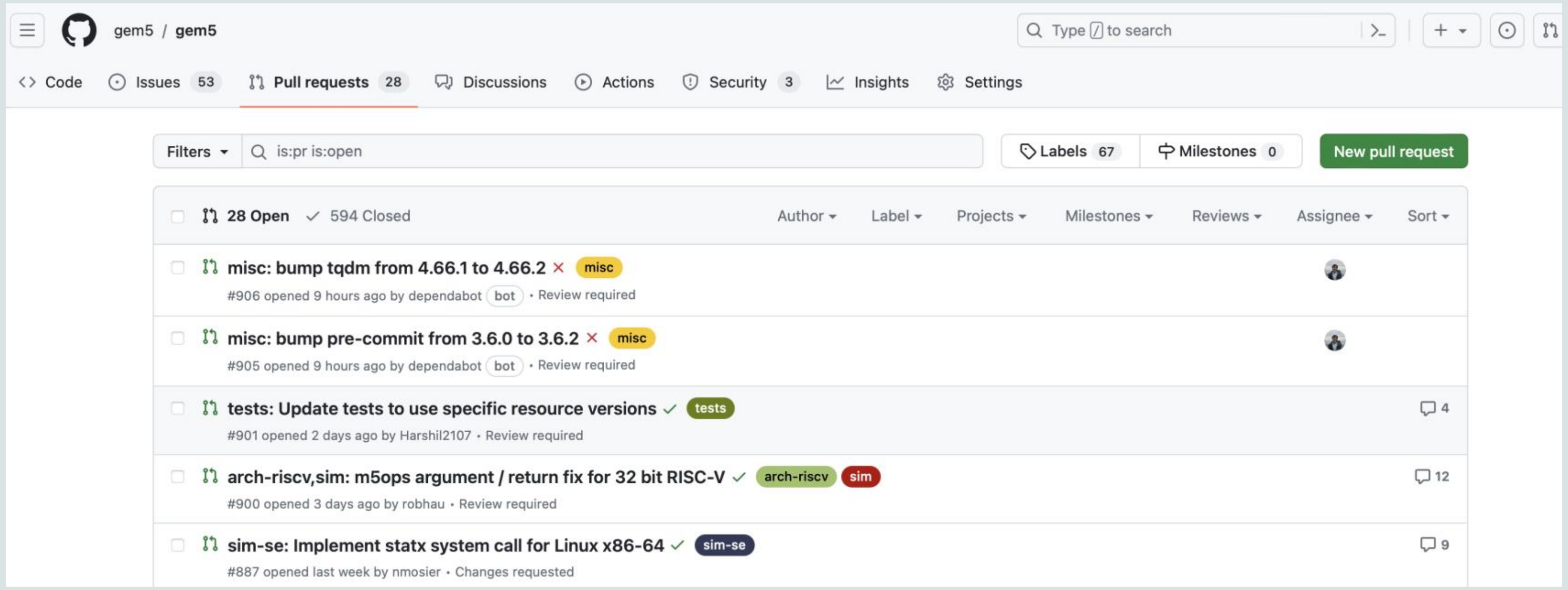
Assignees
No one—as

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



is:pr is:open

28 Open ✓ 594 Closed

misc: bump tqdm from 4.66.1 to 4.66.2 ✗ misc
#906 opened 9 hours ago by dependabot bot • Review required

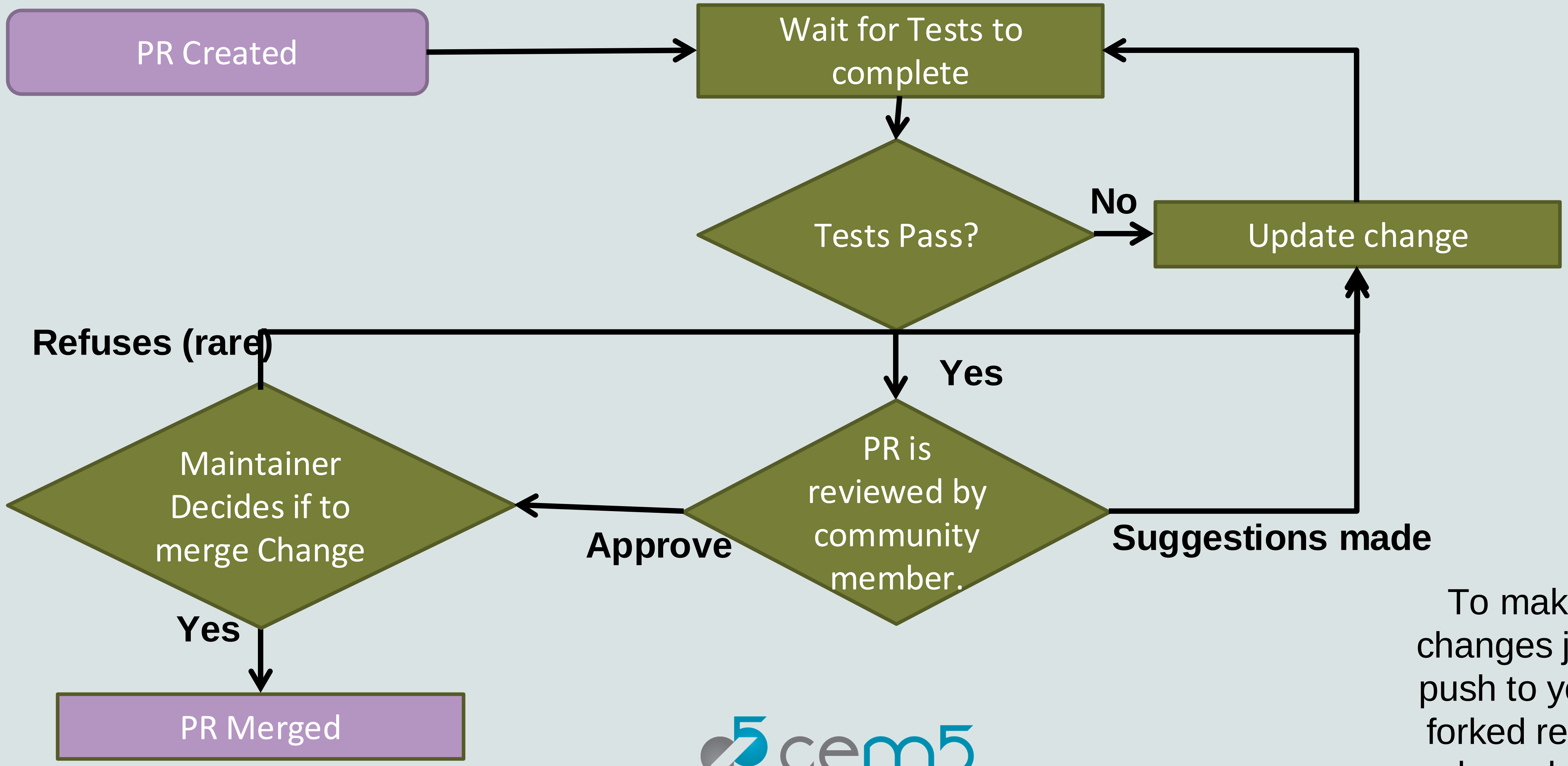
misc: bump pre-commit from 3.6.0 to 3.6.2 ✗ misc
#905 opened 9 hours ago by dependabot bot • Review required

tests: Update tests to use specific resource versions ✓ tests — 4
#901 opened 2 days ago by Harshil2107 • Review required

arch-riscv,sim: m5ops argument / return fix for 32 bit RISC-V ✓ arch-riscv sim — 12
#900 opened 3 days ago by robhau • Review required

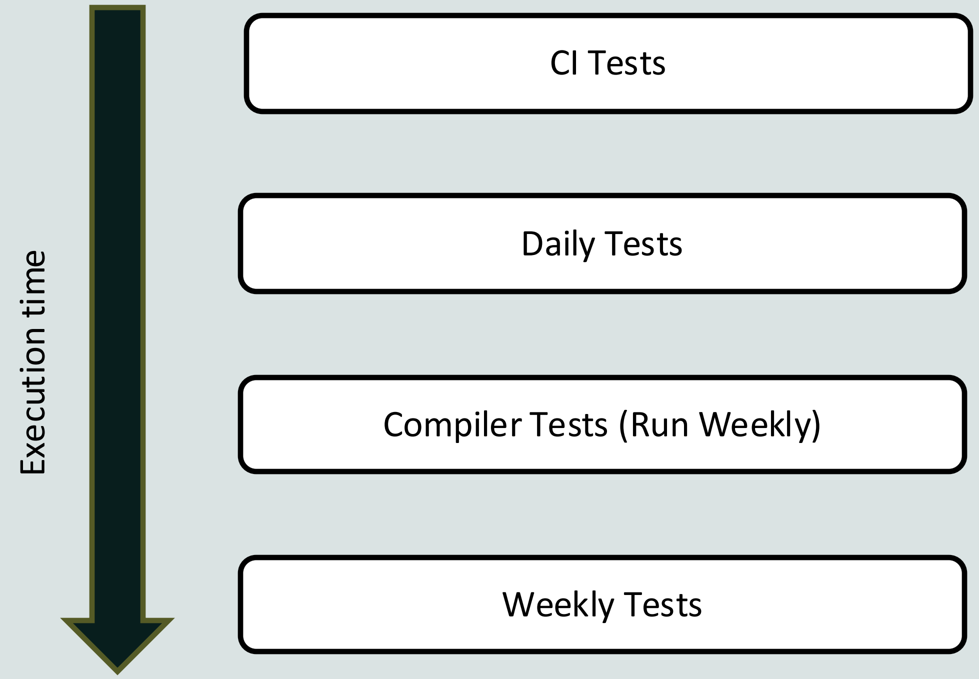
sim-se: Implement statx system call for Linux x86-64 ✓ sim-se — 9
#887 opened last week by nmosier • Changes requested

PR Review Process


PR Created
Wait for Tests to complete
Tests Pass?
No
Update change
Yes
Refuses (rare)
PR is reviewed by community member.
Approve
Suggestions made
Maintainer Decides if to merge Change
Yes
PR Merged

To make changes just push to your forked repo branch!

Testing overview



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.