ECE4700J Lab 6
Introduction to Gem5

Haoyang Zhang

UM-SJTU Joint Institute

zhy-sjtu-jc@sjtu.edu.cn

June 24, 2022

Top 4 Conferences of Computer Architecture

ISCA

https://iscaconf.org/

The International Symposium on Computer Architecture

MICRO 2022

55th IEEE/ACM International Symposium on Microarchitecture®

https://www.microarch.org/

ASPLOS

ASPLOS: Architectural Support for Programming Languages and Operating Systems

https://asplos-conference.org/

High Performance Computer Architecture

HPCA

https://hpca-conf.org/

IEEE International Symposium on High-Performance Computer Architecture

Do Research? You Need to setup your experiment!

For Research in Computer Architecture: How to implement and test your hardware design?

- Use Self-Produced Real Chips (ASIC: application specific integrated circuit) (e.g. Google's TPU)

1. Chip Fabrication is **extremely expensive**
2. **Very long** Development Cycle
3. Designing ASIC is **too complicated**.

Strength: Results are absolutely accurate.

- Use Programmable Devices. E.g. **FPGA**

1. +: Enables people to do hardware designs with **low cost**.
2. +: **Shorter** Development Cycle (Is it really short?)
3. -: **Limited capacity and performance**. Imagine you really implement a complex CPU on FPGA… On that CPU, even boosting Linux may take ~minutes.
4. -: **Poor** software **environment/tools** (e.g. Vivado (- _ -)||)

Do Research? You Need to setup your experiment!

And another way:

- Use **Software Simulators**!

1. They can model **any hardware you want**
2. **Very good scalability**
3. Short Development Cycle
4. Good Environment Support (Since it's software! E.g. You can use printf/gdb to debug!)

However, at last they are still not real hardware…
There are things that they cannot model. (e.g. Rowhammer. Physical effects)

Nowadays, developing better simulators / framework for FPGA / Language for programming hardware are all hot topics in computer architecture research.

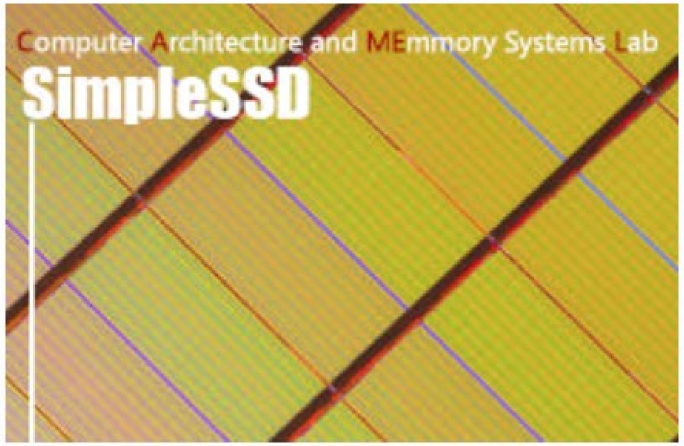
Simulators Examples

Processor Simulators

gem5

[gem5: The gem5 simulator system]

Memory Simulators



[SimpleSSD — SimpleSSD 2.0.12 documentation]

GPU Simulators

AccelSim

[Accel-Sim: The Accel-Sim Framework]

Learning gem5 – Part I

Getting started with gem5

Jason Lowe-Power
http://learning.gem5.org/
https://faculty.engineering.ucdavis.edu/lowepower/

What is gem5?

Michigan m5 + Wisconsin GEMS = gem5

“The gem5 simulator is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture.”

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. **The gem5 simulator**. *SIGARCH Comput. Archit. News* 39, 2 (August 2011), 1-7. DOI=http://dx.doi.org/10.1145/2024716.2024718

Tutorial and book are open source!

http://learning.gem5.org/

https://github.com/powerjg/learning_gem5

See a problem?

Open a pull request or issue

Want to add new material? Let me know!

Want to do your own version of this? See http://learning.gem5.org/book/#notes-for-presentations

This tutorial

This is going to interactive

Work along with me for best results

Ask questions!!

Building gem5

http://learning.gem5.org/book/part1/building.html

Let’s get started!

```
> git clone https://gem5.googlesource.com/public/gem5

> cd gem5

> scons build/X86/gem5.opt -j5
```

and now we wait (about 8 minutes)

```
> scons build/X86/gem5.opt -j5
```

scons: the build system that gem5 uses (like make). See http://scons.org/

build/X86/gem5.opt: "parameter" passed to scons. gem5's *Sconscript* interprets this. Also, the patch to the gem5 executable.

X86: Specifies the default build options. See `build_opts/*`

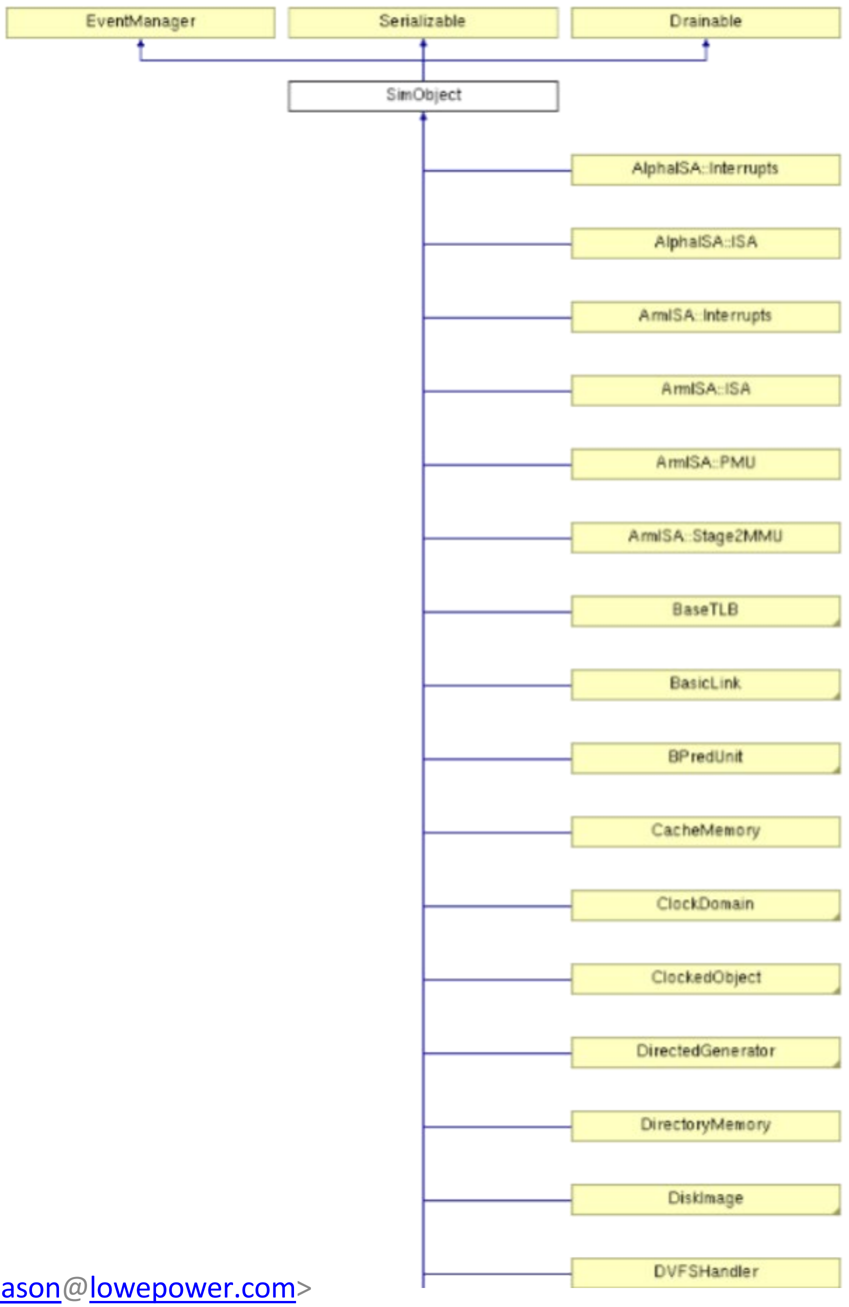
opt: version of executable to compile (one of debug, opt, perf, fast)

gem5 architecture

gem5 consists of "**SimObjects**"

Most C++ objects in gem5 inherit from **`class SimObject`**

Represent physical system components



gem5 configuration scripts

http://learning.gem5.org/book/part1/simple_config.html

http://learning.gem5.org/book/part1/cache_config.html

gem5 user interface

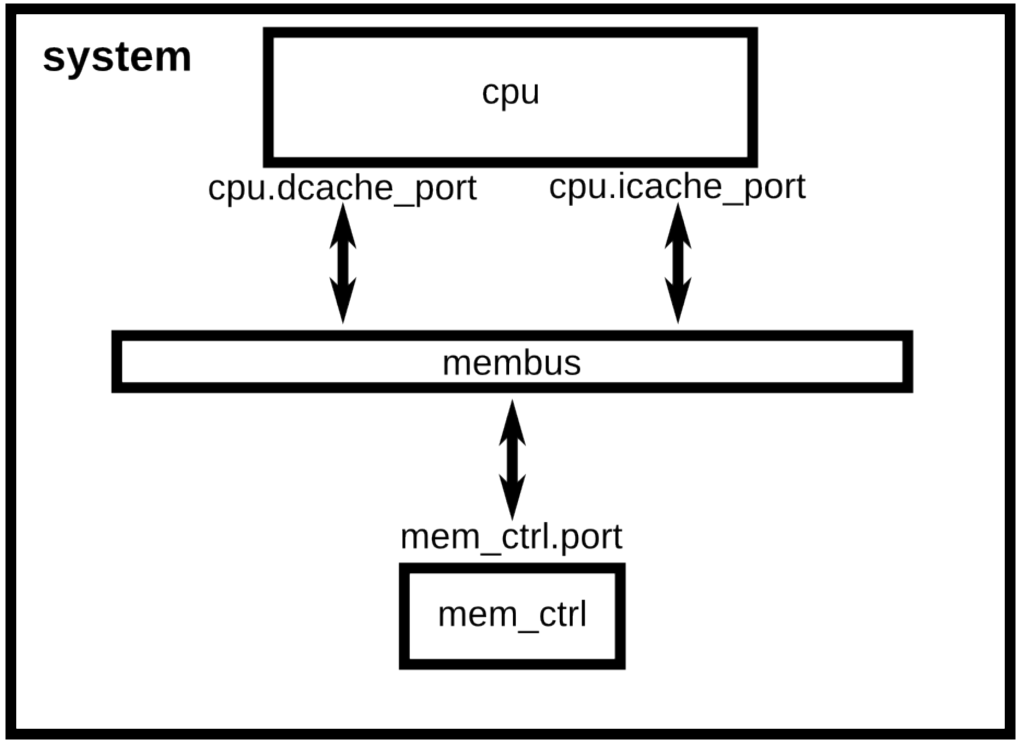
gem5 completely controlled by **Python scripts**

Scripts define system to model

All (C++) SimObjects exposed to Python

So… let's make one!

Simple config script



Single CPU connected to a memory bus

Switch!

Simple config script

configs/learning_gem5/part1/simple.py

Running gem5

```
> build/X86/gem5.opt
      configs/tutorial/simple.py
```

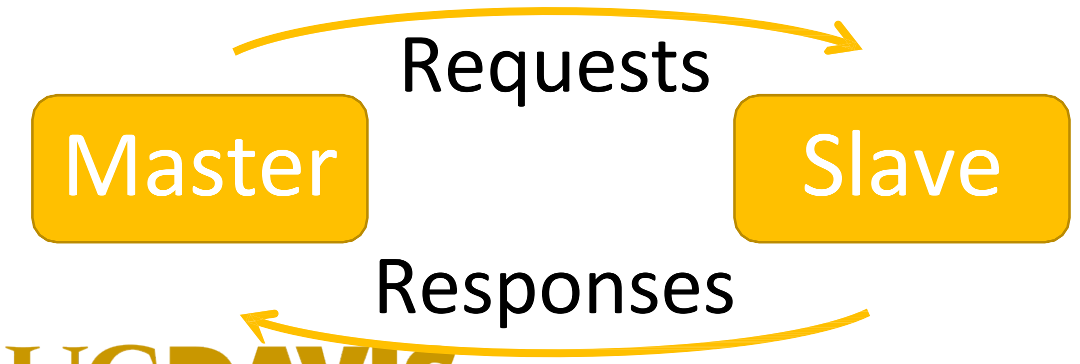
build/X86/gem5.opt:
the gem5 binary to run

configs/…/simple.py:
the configuration script (config script)

Port interface

```
system.cpu.icache_port = system.membus.slave
system.cpu.dcache_port = system.membus.slave
...
system.mem_ctrl.port = system.membus.master
```

Ports connect **MemObjects**



To register a connection between master and slave, use '=' in Python

Syscall Emulation (SE) mode

```
process = Process()
process.cmd = ['tests/.../hello']
system.cpu.workload = process
...
root = Root(full_system = False)
```

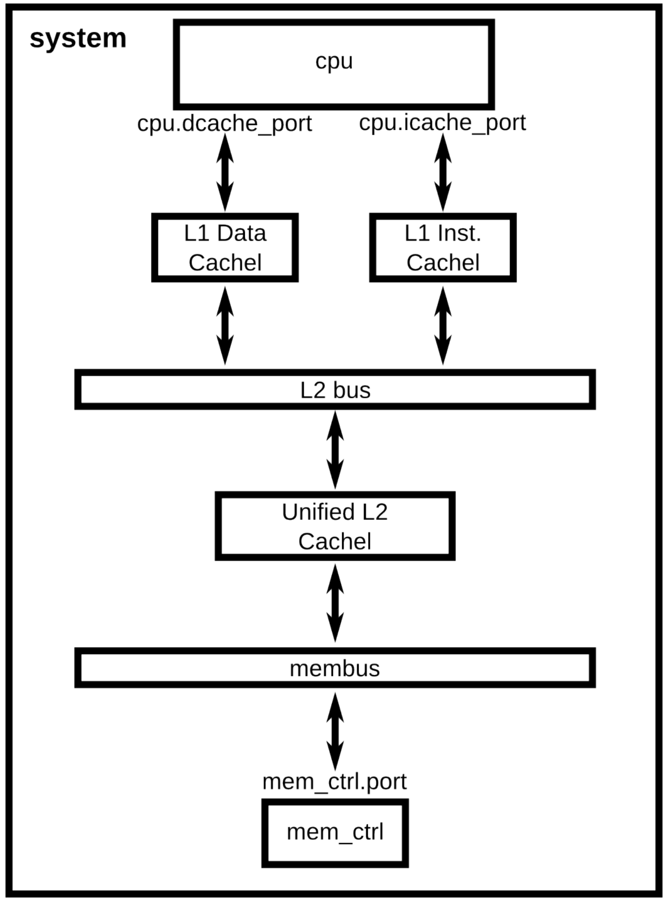
SE mode *emulates* the operating system (Linux) syscalls. No OS runs.

process: an *emulated* process with *emulated* page tables, file descriptors, etc.

Full system mode runs a full OS as if gem5 is a "bare metal" system. Like full virtualization.

Going further: Adding caches

http://learning.gem5.org/book/part1/cache_config.html



Extending SimObjects in Python config

Object-oriented config files

Switch!

Adding command-line parameters

It’s just Python!

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

class L1ICache(L1Cache):
    def connectCPU(self, cpu):
        self.cpu_side = cpu.icache_port
...
```

Use good object-oriented design!

Debugging config files is easy. Just add some print statements!

`See text for details`

Use Python builtins to provide support for command line parameters.

Understanding gem5 output

http://learning.gem5.org/book/part1/gem5_stats.html

Understanding gem5 output

```
> ls m5out
config.ini   config.json   stats.txt
```

config.ini: Dumps all of the parameters of all SimObjects. This shows exactly what you simulated.

config.json: Same as config.ini, but in json format.

stats.txt: Detailed statistic output. Each SimObject defines and updates statistics. They are printed here at the end of simulation.

UCDAVIS

stats.txt

Switch!

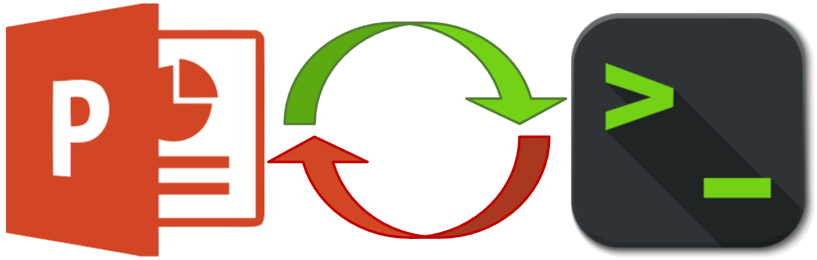
```
---------- Begin Simulation Statistics ----------
sim_seconds     0.000346       # Number of seconds simulated
sim_ticks       345518000      # Number of ticks simulated
final_tick      345518000      # Number of ticks from beginning ...
sim_freq        1000000000000  # Frequency of simulate
...
sim_insts       5712           # Number of instructions simulated
sim_ops         10314          # Number of ops (including micro ...
...
system.mem_ctrl.bytes_read::cpu.inst   58264   # Nu
system.mem_ctrl.bytes_read::cpu.data   7167    # Number of bytes ...
...
system.cpu.committedOps              10314   # Number o
system.cpu.num_int_alu_accesses 10205   # Number of integer ...
```

sim_seconds: name of stat. This shows *simulated guest* time

Every SimObject can have its own stats. Names are what you used in the Python config file

Example scripts

Switch!



Questions?

We covered

- gem5 history
- Downloading and building gem5
- gem5’s user interface: python
- How to write a configuration script
- gem5’s output
- Using the example scripts

Learning gem5 – Part II

Modifying and Extending gem5

Jason Lowe-Power
http://learning.gem5.org/
https://faculty.engineering.ucdavis.edu/lowepower/

A simple SimObject

http://learning.gem5.org/book/part2/helloobject.html

gem5’s coding guidelines

Follow the style guide (http://www.gem5.org/Coding_Style)

- Install the style guide when scons asks
- Don’t ignore style errors

Use good development practices

- Historically mercurial queues
- Now: ***git branches***

Adding a new SimObject

Step 1: Create a Python class

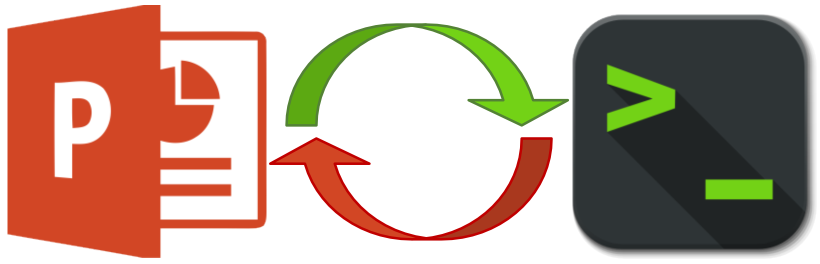
Step 2: Implement the C++

Step 3: Register the SimObject and C++ file

Step 4: (Re-)build gem5

Step 5: Create a config script

Switch!



Step 1: Create a Python class

HelloObject.py

```
from m5.params import *
from m5.SimObject import SimObject

class HelloObject(SimObject):
    type = ‘HelloObject’
    cxx_header = ‘learning_gem5/hello_object.hh’
```

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

Step 2: Implement the C++

hello_object.hh

```
#include "params/HelloObject.hh"
#include "sim/sim_object.hh"
class HelloObject : public SimObject
{
  public:
    HelloObject(HelloObjectParams *p);
};
```

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

Constructor has one parameter, the generated params object.

Step 2: Implement the C++

hello_object.cc

```
HelloObject::HelloObject(HelloObjectParams *params)
    : SimObject(params)
{
    std::cout << "Hello World! From a SimObject!" << std::endl;
}
HelloObject*
HelloObjectParams::create()
{
    return new HelloObject(this);
}
```

HelloObjectParams: when you specify a **Param** in the Hello.py file, it will be a member of this object.

You must **define** this function (you'll get a linker error otherwise). This is how Python config creates the C++ object.

Step 3: Register the SimObject and C++ file

SConscript

```
| Import(*)
| SimObject('Hello.py')
| Source('hello_object.cc')
```

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

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

Step 4: (Re-)build gem5

Step 5: Create a config script

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

```
| ...
| system.hello = HelloObject()
| ...
```

```
> build/X86/gem5.opt configs/learning_gem5/hello.py
...
Hello world! From a SimObject!
...
```

Simple SimObject code

gem5/src/learning_gem5/part2/hello_object.cc

gem5/src/learning_gem5/part2/hello_object.hh

gem5/src/learning_gem5/part2/HelloObject.py

gem5/configs/learning_gem5/part2/hello_run.py

Debug support in gem5

http://learning.gem5.org/book/part2/debugging.html

Adding debug flags

SConscript

Switch!

```
DebugFlag(‘Hello’)
```

hello_object.cc

```
DPRINTF(Hello, “Created the hello object”);
```

Declare the flag: add the debug flag to the SConscript file in the current directory

DPRINTF: macro for d ug statements in gem5

Hello: the debug flag declared in the SConscript. Found in “debug/hello.hh”

Debug string: Any C format string

Debugging gem5

```
> build/X86/gem5.opt --debug-flags=Hello configs/tutorial/hello.py
...
          0: system.hello: Hello world! From a debug statement
```

debug-flags: Comma separated list of flags to enable. Other options include --debug-start=<tick>, --debug-ignore=<simobj name>, etc. See gem5.opt --help

SimObject parameters

http://learning.gem5.org/book/part2/parameters.html

Adding parameters

Switch!

```
class HelloObject(SimObject):
    type = 'HelloObject'
    cxx_header = "learning_gem5/hello_object.hh"

    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")
```

Param.<TYPE>: Specifies a parameter of type <TYPE> for the SimObject

Param.<TYPE>(): First parameter: default value. Second parameter: “help”

Going further: More parameters

http://learning.gem5.org/book/part2/parameters.html

Included types (e.g., MemorySize, MemoryBandwidth, Latency)

Using a SimObject as a parameter

SimObject-SimObject interaction

src/learning_gem5/part2/hello_object.cc & hello_object.hh
src/learning_gem5/part2/goodbye_object.cc & goodbye_object.hh
src/learning_gem5/part2/HelloObject.py & GoodbyeObject.py

Questions?

We covered

- How to build a SimObject
- How to schedule events
- Debug statements in gem5
- Adding parameters to SimObjects

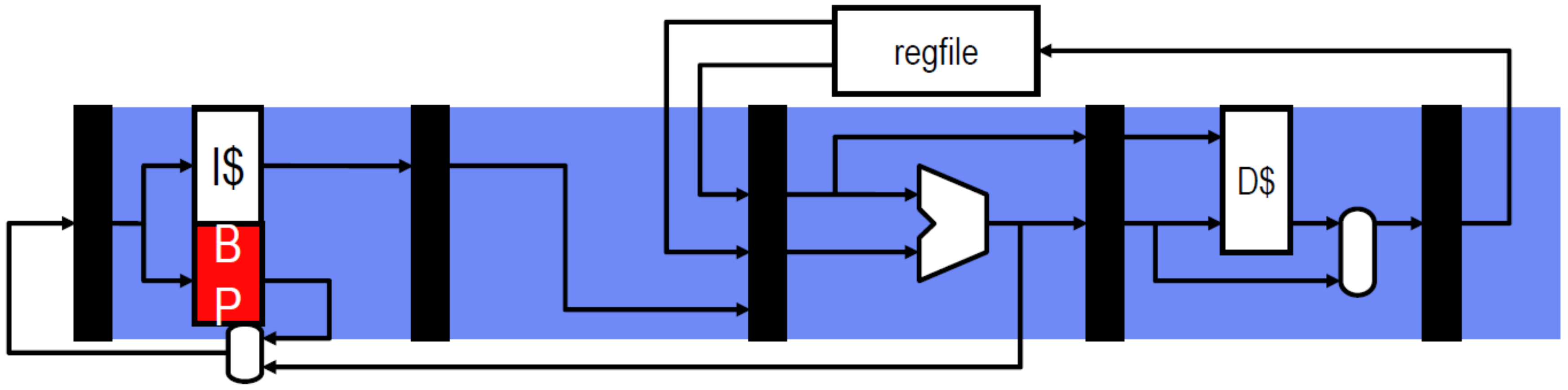
Lab 6 Assignment

Successfully run the **Spectre V1 Attack** on Gem5 Out-of-order CPU core, and visualize it's pipeline output. (Don't worry, it's really easy, you don't need to write any code)

What is **Spectre Attack**?

Meltdown and Spectre (spectreattack.com)

Some Backgrounds: Branch Prediction



BP part I: **target predictor**

- Applies to all control transfers
- Supplies target PC, tells if insn is a branch prior to decode

+ Easy

BP part II: **direction predictor**

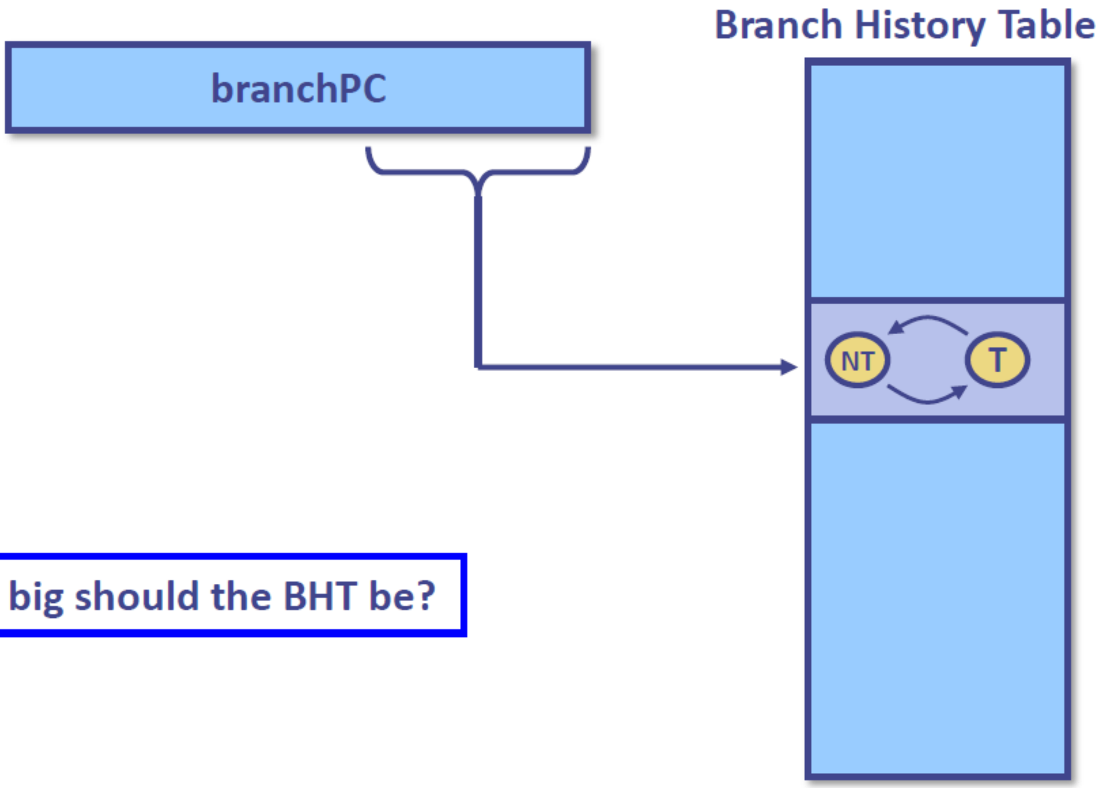
- Applies to conditional branches only
- Predicts taken/not-taken

– Harder

Some Backgrounds: Branch Prediction

Branch history table (BHT): simplest direction predictor

- PC indexes table of bits (0 = N, 1 = T), no tags *(why not?)*
- Essentially: branch will go same way it went last time



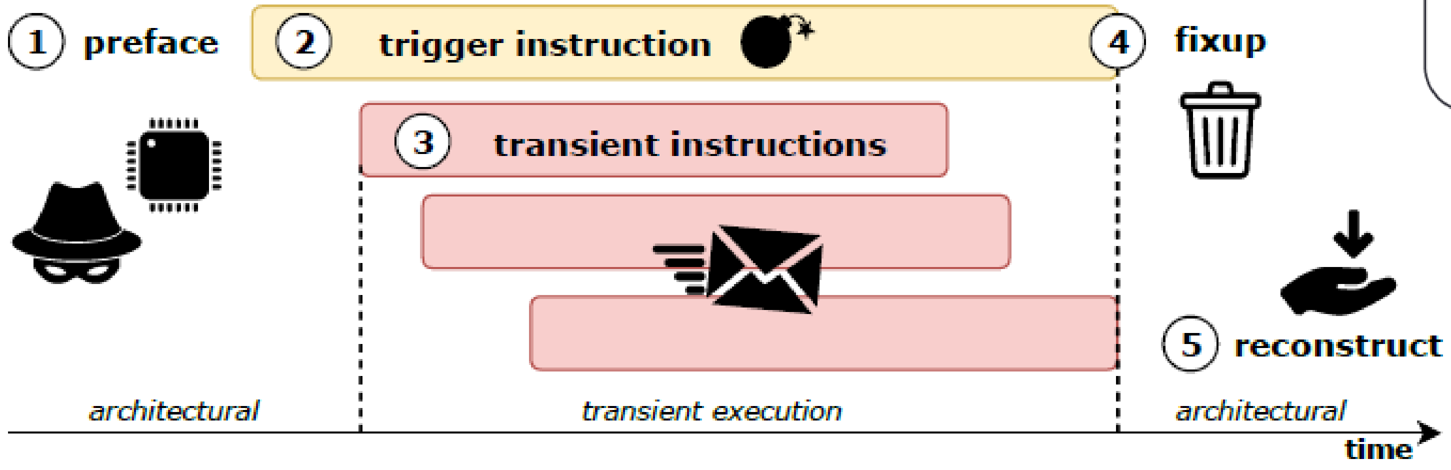
How big should the BHT be?

What Is Transient Execution Attack?

- Term explanation:
  - **Transient instructions**: executed, but result never committed
  - **Architectural** States vs. **Microarchitectural** States: can be observed by developers vs. cannot
    - Arch: Regs, Main Memory
    - MicroArch: Caches, TLB, BTB
  - **Cache Convert Channels**

Changes in microarchitectural states during transient instructions are not discarded!!

```
flush_caches();
throw new someException();
unsigned val = kernelArr[index1];
unsigned index2 = ((val&1)*0x100);
unsigned value2 = userArr[index2];
```



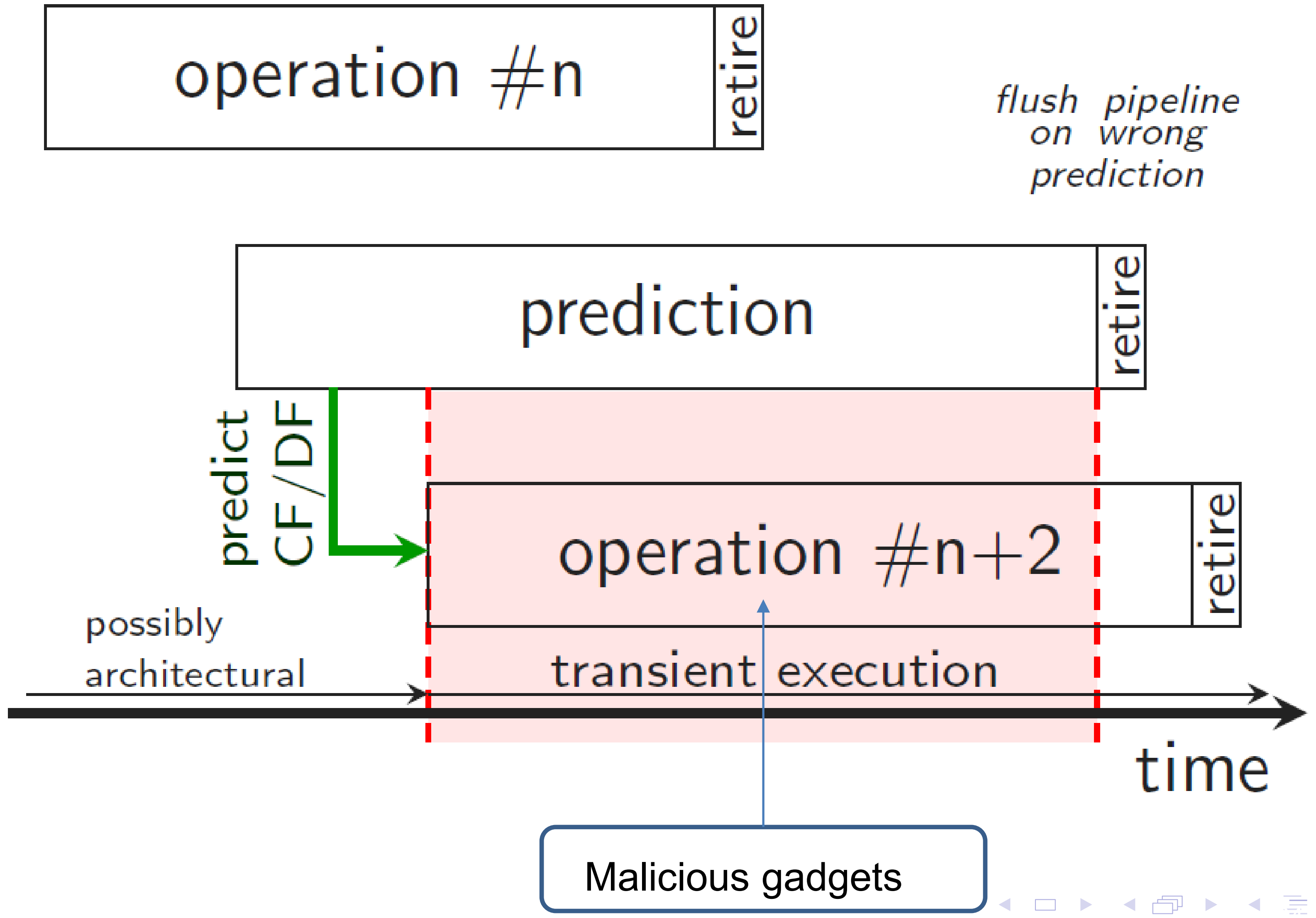
Speculative Covert Channel Attacks

- Induce victim to incorrect speculative execution path
  - Branch prediction
  - Memory speculation
- Construct a long time window for incorrect speculative execution
- Infer secrets from side-channel information leakages

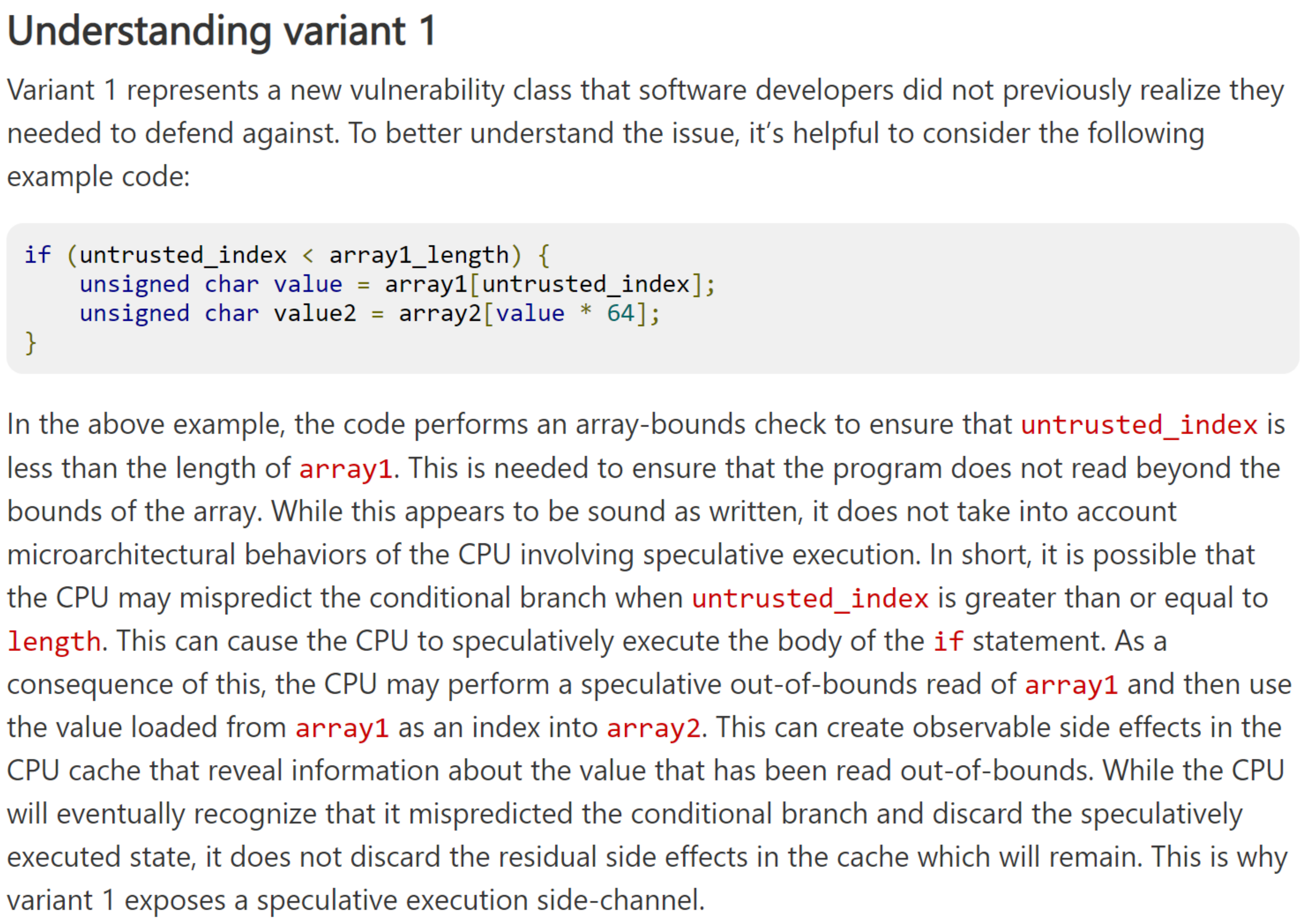
```
flush_caches();
throw new someException();
unsigned val = kernelArr[index1];
unsigned index2 = ((val&1)*0x100);
unsigned value2 = userArr[index2];
```

Focus of this work: Cache side channel attacks

Spectre Attacks



Spectre V1 Example



Understanding variant 1

Variant 1 represents a new vulnerability class that software developers did not previously realize they needed to defend against. To better understand the issue, it's helpful to consider the following example code:

```
if (untrusted_index < array1_length) {
    unsigned char value = array1[untrusted_index];
    unsigned char value2 = array2[value * 64];
}
```

In the above example, the code performs an array-bounds check to ensure that `untrusted_index` is less than the length of `array1`. This is needed to ensure that the program does not read beyond the bounds of the array. While this appears to be sound as written, it does not take into account microarchitectural behaviors of the CPU involving speculative execution. In short, it is possible that the CPU may mispredict the conditional branch when `untrusted_index` is greater than or equal to `length`. This can cause the CPU to speculatively execute the body of the `if` statement. As a consequence of this, the CPU may perform a speculative out-of-bounds read of `array1` and then use the value loaded from `array1` as an index into `array2`. This can create observable side effects in the CPU cache that reveal information about the value that has been read out-of-bounds. While the CPU will eventually recognize that it mispredicted the conditional branch and discard the speculatively executed state, it does not discard the residual side effects in the cache which will remain. This is why variant 1 exposes a speculative execution side-channel.

References

➔ Learning Gem5. http://learning.gem5.org/

➔ Spectre mitigations in MSVC. Spectre mitigations in MSVC - C++ Team Blog (microsoft.com)

➔ Kocher, Paul, et al. "Spectre attacks: Exploiting speculative execution." *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019.