Architectural Exploration with gem5

ARM

Andreas Sandberg
Stephan Diestelhorst
William Wang

ARM Research

Xi'An: ASPLOS 2017
2017-04-09

This is an interactive presentation

Please ask questions!

Even if they are in:

- English
- Chinese
- Swedish
- German

Agenda

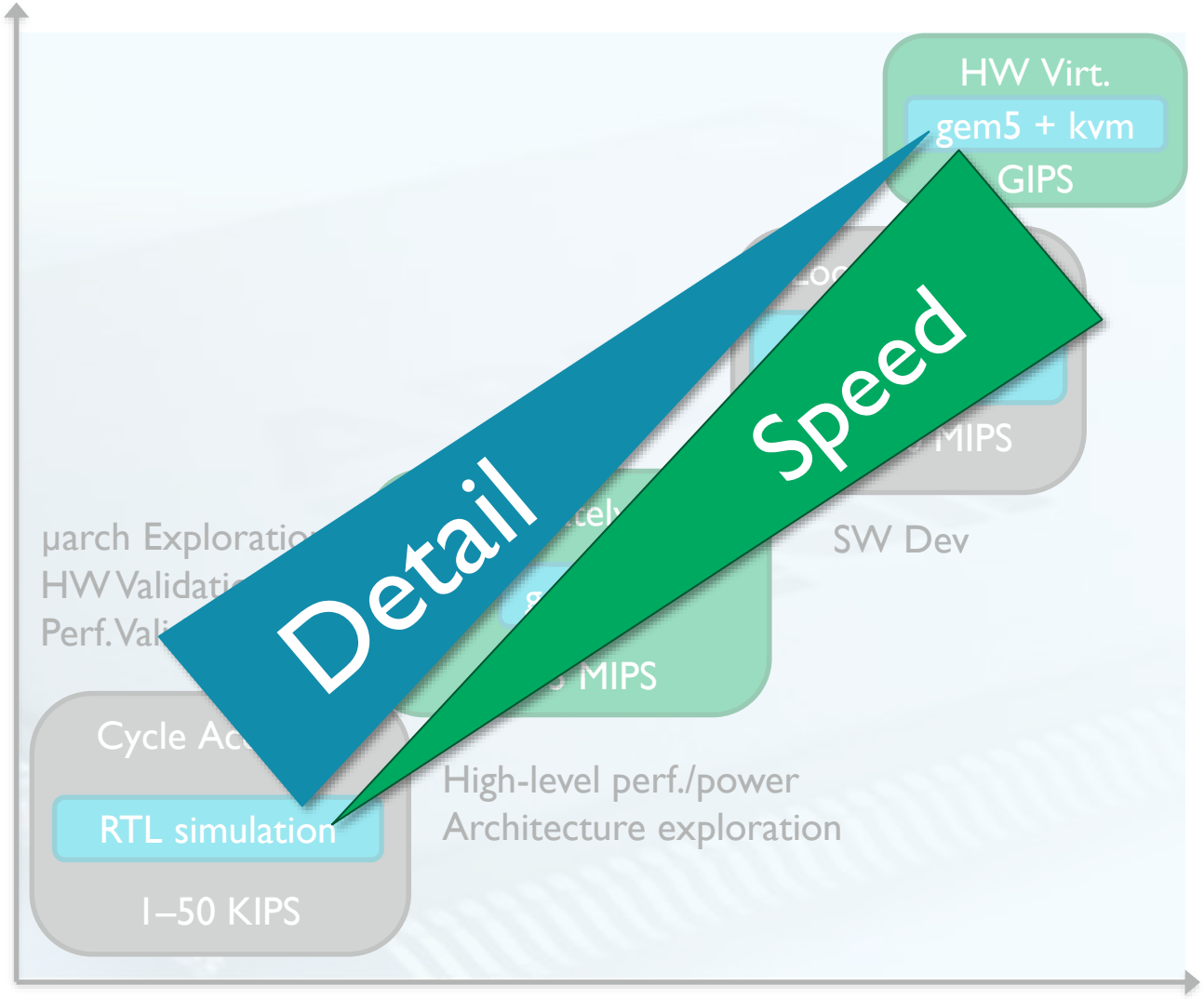
- Presenters: Andreas Sandberg, William Wang, Stephan Diestelhorst (ARM Cambridge, UK)

- 13:00 Introduction (10 min) – Stephan
- 13:10 Getting Started (15 min) – William
- 13:25 Configuration (25 min) – Andreas
- 13:50 Debug & Trace (20 min) – William
- 14:10 Creating SimObjects (20 min) – Andreas
- 14:30 Coffee Break (30 min)
- 15:00 Memory System (40 min) – Stephan
- 15:40 CPU Models (20 min) – Andreas
- 16:00 Advanced Features (45 min) – all
- 16:45 Contributing to gem5 (20 min) – Andreas

What is gem5?

Level of detail

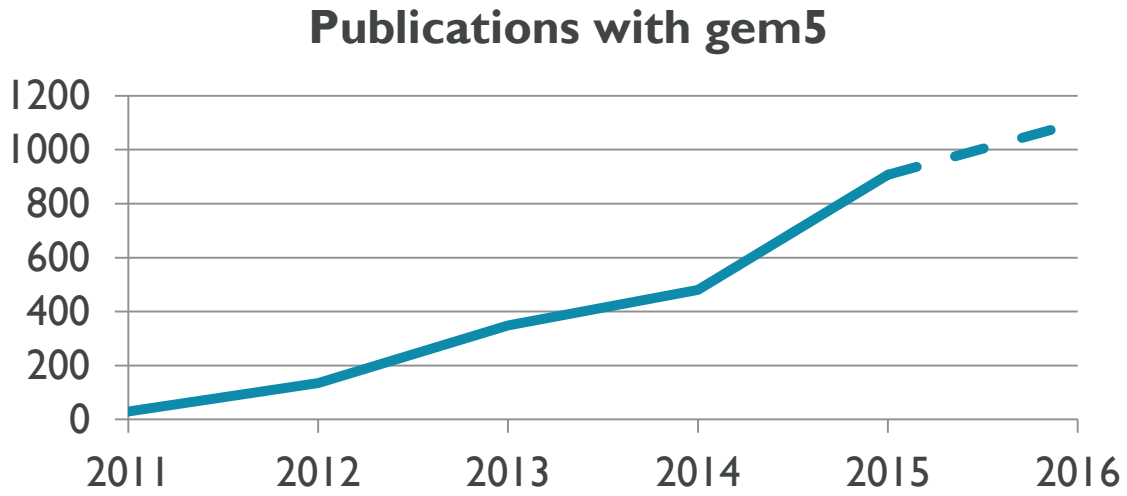
- HW Virtualization
  - Very no/limited timing
  - The same Host/guest ISA
- Functional mode
  - No timing, chain basic blocks of instructions
  - Can add cache models for warming
- Timing mode
  - Single time for execute and memory lookup
  - Advanced on bundle
- Detailed mode
  - Full out-of-order, in-order CPU models
  - Hit-under-miss, reodering, …



ARM

Users and contributors

- Widely used in academia and industry
- Contributions from
  - ARM, AMD, Google,…
  - Wisconsin, Cambridge, Michigan, BSC, …



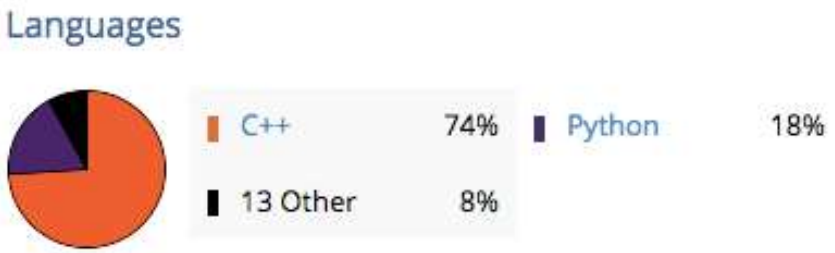
In a Nutshell, gem5...

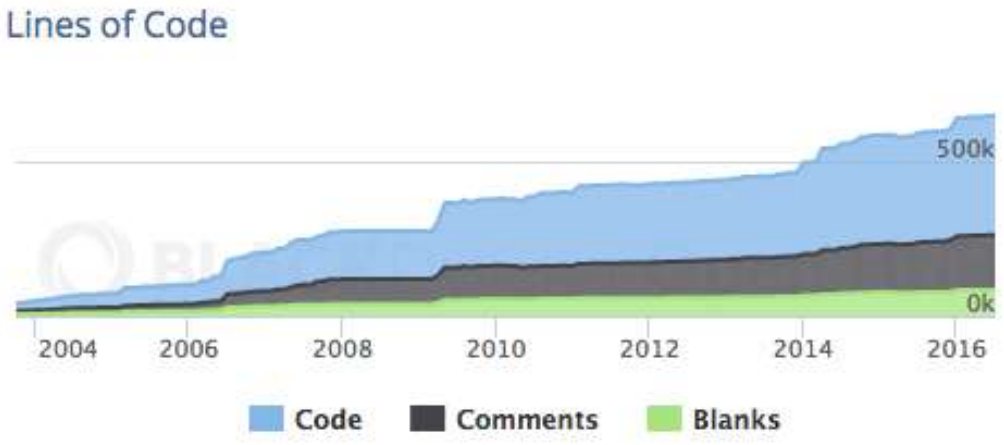
... has had 11,558 commits made by 193 contributors
representing 386,321 lines of code

... is mostly written in C++
with a well-commented source code

... has a well established, mature codebase
maintained by a very large development team
with stable Y-O-Y commits

... took an estimated 104 years of effort (COCOMO model)
starting with its first commit in October, 2003
ending with its most recent commit 14 days ago





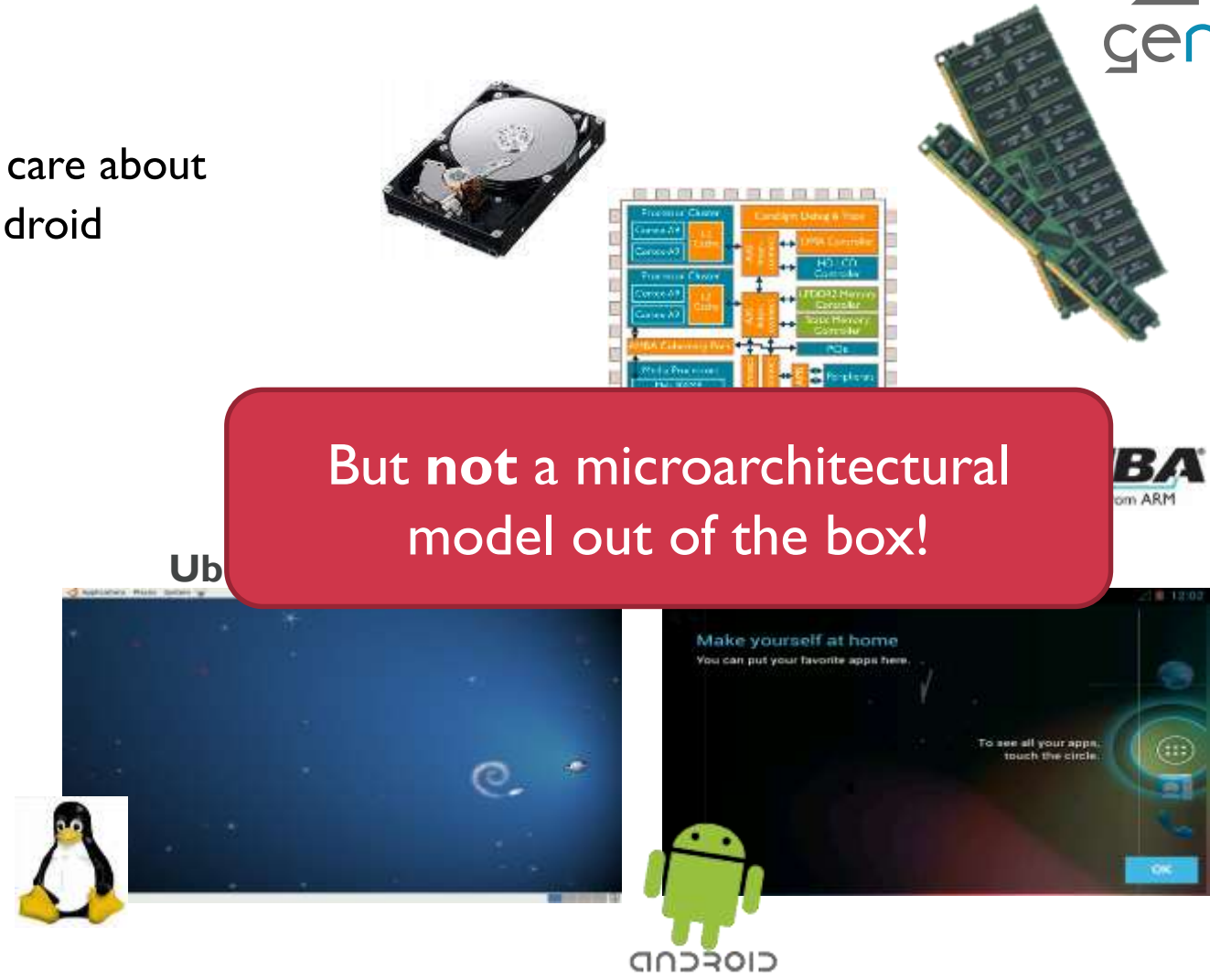
ARM

When not to use gem5

- Performance validation
  - gem5 is not a cycle-accurate microarchitecture model!
  - This typically requires more accurate models such as RTL simulation.
  - Commercial products such as **ARM CycleModels** operate in this space.
- Core microarchitecture exploration
  - *Only* do this if you have a custom, detailed, CPU model!
  - gem5's core models were not designed to replace more accurate microarchitectural models.
- To validate functional correctness or test bleeding-edge ISA improvements
  - gem5 is not as rigorously tested as commercial products.
  - New (ARMv8.0+) or optional instructions are sometimes not implemented
  - Commercial products such as **ARM FastModels** offer better reliability in this space.

Why gem5?

- Runs real workloads
  - Analyze workloads that customers use and care about
  - … including complex workloads such as Android
- Comprehensive model library
  - Memory and I/O devices
  - Full OS, Web browsers
  - Clients and servers
- Rapid *early* prototyping
  - New ideas can be tested quickly
  - System-level impact can be quantified
- System-level insights
  - Enables us to study complex memory-system interactions
- Can be wired to custom models
  - *Add detail where it matters, when it matters!*



Getting Started

William Wang

Prerequisites

- Operating system:
  - OSX, Linux
  - Limited support for Windows 10 with a Linux environment
- Software:
  - git
  - Python 2.7 (dev packages)
  - SCons
  - gcc 4.8 or clang 3.1 (or newer)
  - SWIG 2.0.4 or newer
  - make
- Optional:
  - dtc (to compile device trees)
  - ARMv8 cross compilers (to compile workloads)
  - python-pydot (to generate system diagrams)

Compiling gem5

```
$ scons build/ARM/gem5.opt
```

- Guest architecture
- Several architectures in the source tree.
- Most common ones are:
  - **ARM**
  - **NULL** – Used for trace-drive simulation
  - **X86** – Popular in academia, but *very* strange timing behavior

- Optimization level:
  - **debug**: Debug symbols, no/few optimizations
  - **opt**: Debug symbols + most optimizations
  - **fast**: No symbols + even more optimizations

Compiling gem5's device trees

1. `sudo apt install device-tree-compiler`
2. `make -C system/arm/dt`

- Device trees are used to describe hard-to-discover devices

- `armv8_gem5_v1_`**`N`**`cpu.dtb`
  - Traditional CMP/SMP configuration with **N** cores
  - Built from armv8.dts and platforms/vexpress_gem5_v1.dtsi
- `armv8_gem5_v1_big_little_`**`M`**`_`**`N`**`.dtb`
  - bigLittle configurations with **M** big cores and **N** small cores
  - Built from armv8.dts and platforms/vexpress_gem5_v1.dtsi

Compiling Linux for gem5

```
1. sudo apt install gcc-aarch64-linux-gnu
2. git clone -b gem5/v4.4  https://github.com/gem5/linux-arm-gem5
3. cd linux-arm-gem5
4. make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- gem5_defconfig
5. make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -j `nproc`
```

- Builds the default kernel configuration for gem5
  - Has support for most of the devices that gem5 supports

Example disk images

- Example kernels and disk images can be downloaded from gem5.org/Download
  - This includes pre-compiled boot loaders
  - Old but useful to get started
- Download and extract this into a new directory:
  - `wget http://www.gem5.org/dist/current/arm/aarch-system-2014-10.tar.xz`
  - `mkdir dist; cd dist`
  - `tar xvf ../aarch-system-2014-10.tar.xz`
- Set the M5_PATH variable to point to this directory:
  - `export M5_PATH=/path/to/dist`
- Most example scripts try to find files using `M5_PATH`
  - Kernels/boot loaders/device trees in `${M5_PATH}/binaries`
  - Disk images in `${M5_PATH}/disks`

Running an example script

```
$ build/ARM/gem5.opt configs/example/arm/fs_bigLITTLE.py \
    --kernel path/to/vmlinux \
    --cpu-type atomic \
    --dtb $PWD/system/arm/dt/armv8_gem5_v1_big_little_1_1.dtb \
    --disk your_disk_image.img
```

- Simulates a bL system with 1+1 cores
  - Uses a functional 'atomic' CPU model
  - Use the 'timing' CPU type for an example OoO + InO configuration

Demo

Configuration and Control

Andreas Sandberg

Design philosophy

- gem5 is conceptually a Python library implemented in C++
  - Configured by instantiating Python classes with matching C++ classes
  - Model parameters exposed as attributes in Python
  - Running is controlled from Python, but implemented in C++

- Configuration and running are two distinct steps
  - Configuration phase ends with a call to instantiate the C++ world
  - Parameters cannot be changed after the C++ world has been created

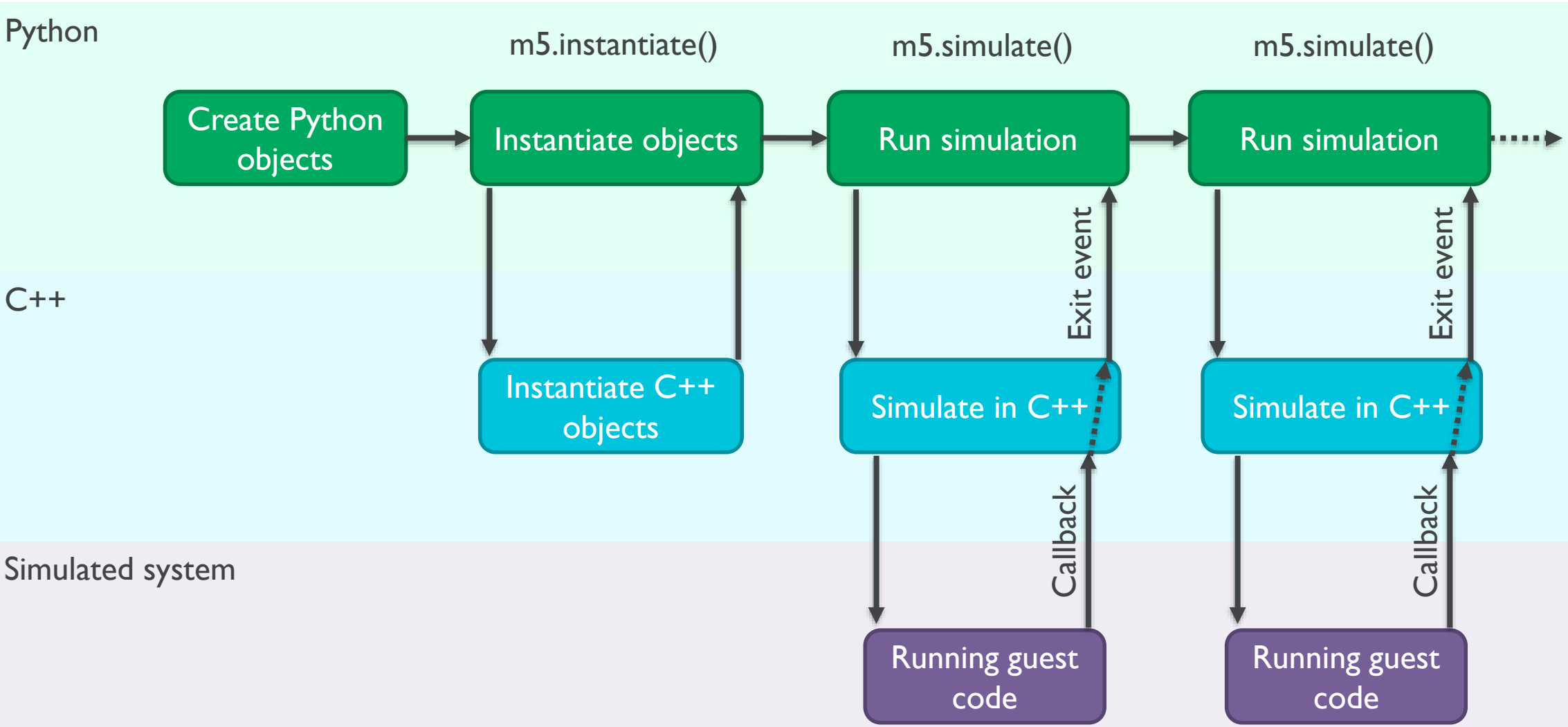
Useful tricks

- gem5 can be launched interactively
  - Use the `-i` option
  - Pretty prompt if ipython has been installed
  - Still requires a simulation script

- Ignore `configs/example/{fs,se}.py` and `configs/common/FSConfig.py`
  - Far too complex
  - Tries to handle every single use case in a single configuration file

- Good configuration examples:
  - `configs/learning_gem5/`
  - `configs/example/arm/`

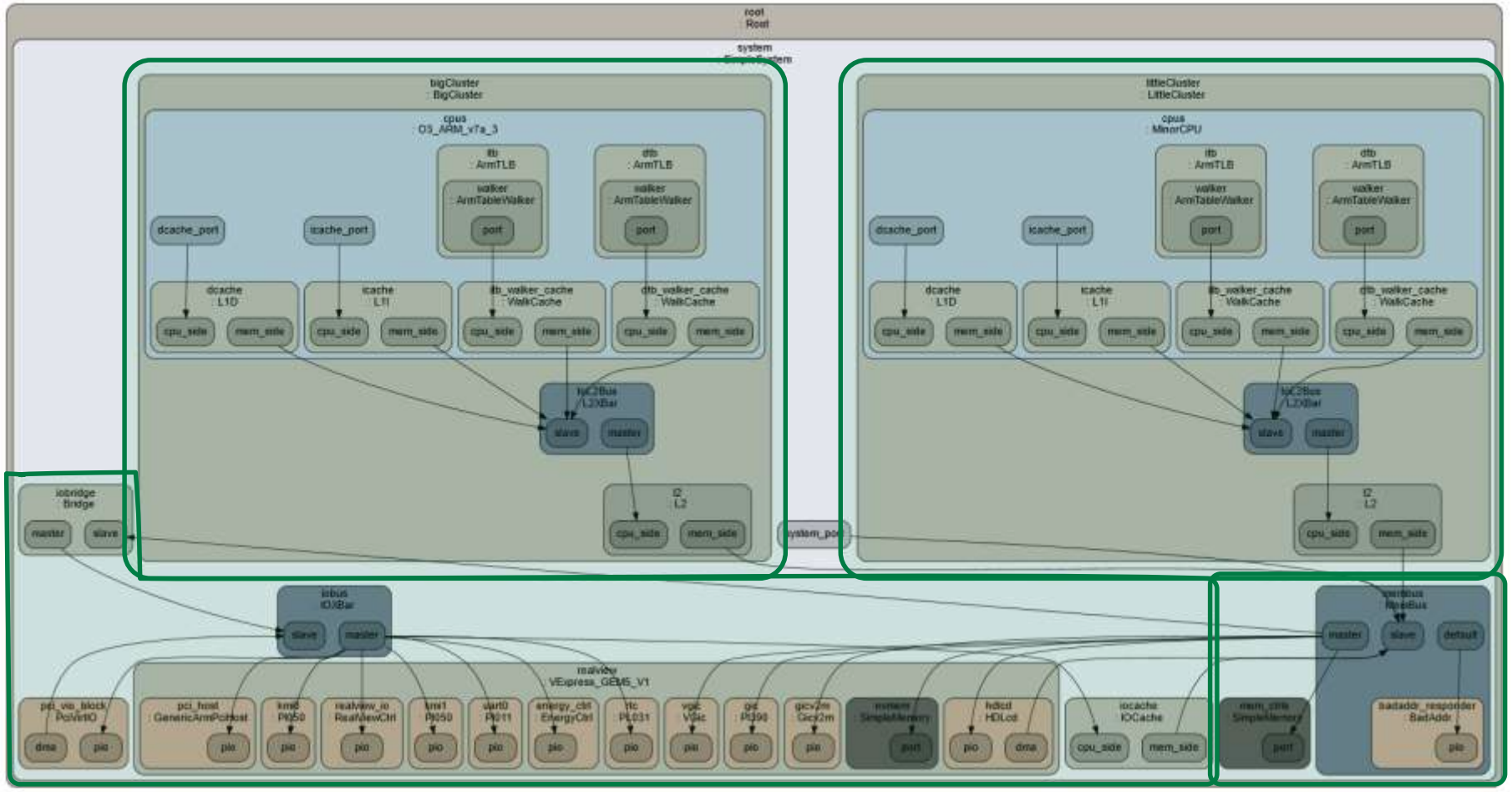
Control flow



General structure

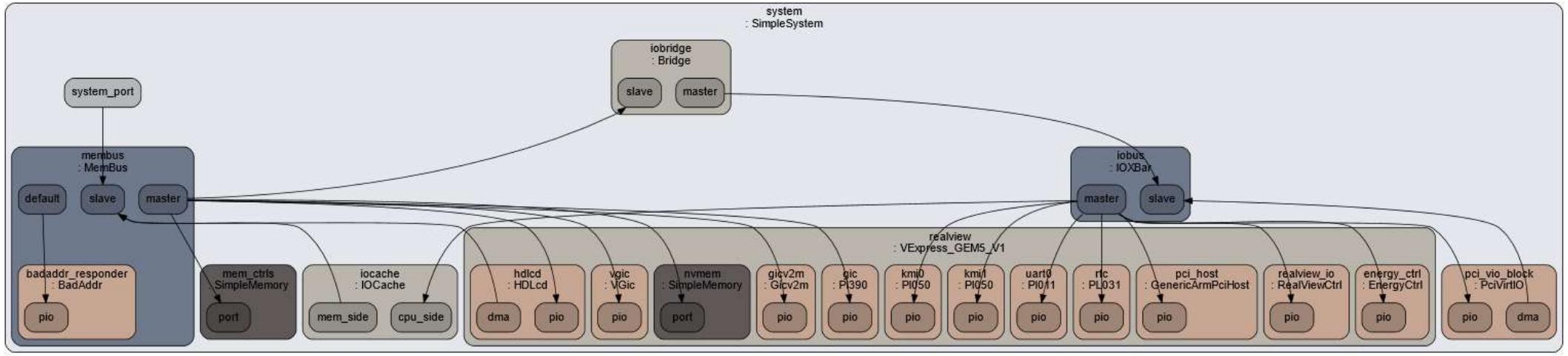
- The simulator contains exactly one Root object
  - Controls global configuration options
  - `root = Root(full_system=True)`
- The root object contains one or more System instances
  - A system represents a shared memory machine
  - Contains devices, CPUs, and memories
- Multiple system may be connected using network interfaces
  - Cluster on cluster simulation
  - Not within the scope of this presentation

System Overview



ARM

Creating a "simple" system



- The system contains basic platform devices
  - Interrupt controllers, PCI bridge, debug UART
  - Sets up the boot loader and kernel as well
- See examples in config/example/arm:
  - `SimpleSystem` (devices.py) defines a basic ARM system with PCI support
  - Instantiated by `createSystem()` in fs_bigLITTLE.py

Overriding model parameters

```
import m5
```

```
class L1DCache(m5.objects.Cache):
    assoc = 2
    size = '16kB'
```

- Use gem5's base Cache
- Override associativity
- Override size

```
class L1ICache(L1DCache):
    assoc = 16
```

- Use defaults from L1DCache
- Override associativity again

```
l1i = L1ICache(assoc=8,
        repl=m5.objects.RandomRepl())
```

- Override parameters at instantiation time
- We'll cover memory ports later

Running

```
m5.instantiate()
```

- Instantiate the C++ world

```
event = m5.simulate()
```

- Start the simulation

```
print 'Exiting @ tick %i: %s' \
    % (m5.curTick(),
       event.getCause())
```

- Print why the simulator exited
- Sometimes desirable to call m5.simulate() again.

```
m5.simulate(m5.tick.fromSeconds(0.1))
```

- Run for a fixed number of simulated seconds

Creating Checkpoints

```
m5.checkpoint('name.cpt')
```

- Checkpoints can be used to store the simulator's state
  - Can be used to implement SimPoints or similar methodologies
- Checkpoint limitations:
  - The act of taking a checkpoint affects system state!
  - Checkpoints don't store cache state
  - Checkpoints don't store pipeline state

Restoring Checkpoints

```
m5.instantiate('name.cpt')
```

- Instantiate system and load state from checkpoint

```
event = m5.simulate()
```

- Run in the same way as before

Guest to simulation script communication

```
system.exit_on_work_items = True
...
event = m5.simulate()
-----
```

- Work item handling in Python
- Exit event will contain information about work items

```
#include "m5op.h"
```

- Include the m5op header
- Remember to link with libm5.a

```
m5_work_begin(id, 0);
// Region of interest
m5_work_end(id, 0);
```

- Annotate your regions of interest

Exit Events

`event.getCause()`	`event.getCode()`	Description
user interrupt received	-	User pressed Ctrl+C
simulate() limit reached	-	gem5 reached the specified time limit
m5_exit instruction encountered	Exit code from guest	Guest executed m5_exit()
m5_fail instruction encountered	Failure code from guest	Guest executed m5_fail()
checkpoint	-	Guest executed m5_checkpoint()
workbegin/workend	Work item ID	Guest work item annotation

Dumping statistics

- Can be requested from Python:
  - `m5.stats.dump()`: Dump statistics
  - `m5.stats.reset()`: Reset stat counters

- Guest command line:
  - `m5 dumpstats [[delay] [period]]`
  - `m5 dumpresetstas [[delay] [period]]`

- Guest code using libm5.a:
  - `m5_dump_stats(delay, periodicity)`: Dump statistics
  - `m5_dumpreset_stats(delay, periodicity)`: Dump & reset statistics

Examples

- Simple full system configuration file: ARM big.LITTLE configuration example
  - `configs/example/arm/{fs_bigLittle.py, devices.py}`
  - Demonstrates how to setup a single system
  - Reasonably small and well documented

- Distributed multi-system configuration:
  - `configs/example/arm/dist_bigLittle.py`
  - Reuses the configuration file above

- Simple syscall emulation mode example: Jason Lowe-Power's Learning gem5
  - `configs/learning_gem5/part1`

Debugging

William Wang

Debugging Facilities

- Tracing
  - Instruction tracing
  - Diffing traces
- Using gdb to debug gem5
  - Debugging C++ and gdb-callable functions
  - Remote debugging
- Pipeline viewer

Tracing/Debugging

- `printf()` is a nice debugging tool
  - Keep good print statements in code and selectively enable them
  - Lots of debug output can be a very good thing when a problem arises
  - Use `DPRINTF`s in code
  - `DPRINTF(TLB, "Inserting entry into TLB with pfn:%#x…)`
- Example flags:
  - `Fetch, Decode, Ethernet, Exec, TLB, DMA, Bus, Cache, O3CPUAll`
  - Print out all flags with `./build/ARM/gem5.opt -- debug-help`
- Enabled on the command line
  - `--debug-flags=Exec`
  - `--debug-start=30000`
  - `--debug-file=my_trace.out`
  - Enable the flag Exec; Start at tick `30000`; Write to `my_trace.out`

Sample Run with Debugging

Command Line:

```
22:44:28 [/work/gem5] ./build/ARM/gem5.opt --debug-flags=Decode --
debug-start=50000-- debug-file=my_trace.out configs/example/se.py  -c
tests/test-progs/hello/bin/arm/linux/hello
…
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 3107500 because target called exit()
```

my_trace.out:

```
2:44:47 [ /work/gem5] head m5out/my_trace.out
  50000:     system.cpu:  Decode:  Decoded cmps instruction:     0xe353001e
  50500:     system.cpu:  Decode:  Decoded ldr instruction:  0x979ff103
  51000:     system.cpu:  Decode:  Decoded ldr instruction:  0xe5107004
  51500:     system.cpu:  Decode:  Decoded ldr instruction:  0xe4903008
  52000:     system.cpu:  Decode:  Decoded addi_uop instruction: 0xe4903008
  52500:     system.cpu:  Decode:  Decoded cmps instruction:     0xe3530000
  53000:     system.cpu:  Decode:  Decoded b instruction:    0x1affff84
  53500:     system.cpu:  Decode:  Decoded sub instruction:  0xe2433003
  54000:     system.cpu:  Decode:  Decoded cmps instruction:     0xe353001e
  54500:     system.cpu:  Decode:  Decoded ldr instruction:  0x979ff103
```

Adding Your Own Flag

- Print statements put in source code
  - Encourage you to add ones to your models or contribute ones you find particularly useful
- Macros remove them from the **`gem5.fast`** binary
  - There is no performance penalty for adding them
  - To enable them you need to run **`gem5.opt`** or **`gem5.debug`**
- Adding one with an existing flag
  - `DPRINTF(<flag>, "normal printf %s\n", "arguments");`
- To add a new flag add the following in a `Sconscript`
  - `DebugFlag('MyNewFlag')`
  - `Include corresponding header, e.g. #include "debug/MyNewFlag.hh"`

Instruction Tracing

- Separate from the general debug/trace facility
  - But both are enabled the same way
- Per-instruction records populated as instruction executes
  - Start with PC and mnemonic
  - Add argument and result values as they become known
- Printed to trace when instruction completes
- Flags for printing cycle, symbolic addresses, etc.

```
2:44:47 [ /work/gem5] head m5out/my_trace.out
50000:   T0 : 0x14468 :   cmps   r3, #30             : IntAlu :  D=0x00000000
50500:   T0 : 0x1446c     :   ldrls   pc, [pc, r3 LSL #2]    : MemRead :  D=0x00014640 A=0x14480
51000:   T0 : 0x14640     :   ldr   r7, [r0, #-4]        : MemRead :  D=0x00001000 A=0xbefffff0c
51500:   T0 : 0x14644.0   :   ldr   r3, [r0] #8          : MemRead :  D=0x00000011 A=0xbefffff10
52000:   T0 : 0x14644.1   :   addi_uop   r0, r0, #8      : IntAlu :  D=0xbefffff18
52500:   T0 : 0x14648     :   cmps   r3, #0              : IntAlu :  D=0x00000001
53000:   T0 : 0x1464c     :   bne                           : IntAlu :
```

Using GDB with gem5

- Several gem5 functions are designed to be called from GDB
  - `schedBreakCycle()` – also with `--debug-break`
  - `setDebugFlag()/clearDebugFlag()`
  - `dumpDebugStatus()`
  - `eventqDump()`
  - `SimObject::find()`
  - `takeCheckpoint()`

Using GDB with gem5

```
2:44:47 [/work/gem5] gdb --args ./build/ARM/gem5.opt
configs/example/fs.py
GNU gdb Fedora (6.8-37.el5)
(gdb) b main
Breakpoint 1 at 0x4090b0: file build/ARM/sim/main.cc, line 40.
(gdb) run

Breakpoint 1, main (argc=2, argv=0x7fffa59725f8) at
build/ARM/sim/main.cc
    main(int argc, char **argv)

(gdb) call schedBreakCycle(1000000)
(gdb) continue
Continuing.

gem5 Simulator System
...
0: system.remote_gdb.listener: listening for remote gdb #0 on
port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0. Starting simulation...
Program received signal SIGTRAP, Trace/breakpoint trap.
0x0000003ccb6306f7 in kill () from /lib64/libc.so.6
```

Using GDB with gem5

```
(gdb) p _curTick
$1 = 1000000

(gdb) call setDebugFlag("Exec")
(gdb) call schedBreakCycle(1001000)
(gdb) continue
Continuing.

1000000: system.cpu T0 : @_stext+148. 1 : addi_uop r0, r0, #4 : IntAlu
: D=0x00004c30
1000500: system.cpu T0 : @_stext+152 : teqs r0, r6 : IntAlu :
D=0x00000000
Program received signal SIGTRAP, Trace/breakpoint trap.
    0x0000003ccb6306f7 in kill () from /lib64/libc.so.6
(gdb) print SimObject::find("system.cpu")
$2 = (SimObject *) 0x19cba130
(gdb) print (BaseCPU*)SimObject::find("system.cpu")
$3 = (BaseCPU *) 0x19cba130
(gdb) p $3->instCnt
$4 = 431
```

Diffing Traces

- Often useful to compare traces from two simulations
  - Find where known good and modified simulators diverge
- Standard diff only works on files (not pipes)
  - …but you really don't want to run the simulation to completion first
- `util/rundiff`
  - Perl script for diffing two pipes on the fly
- `util/tracediff`
  - Handy wrapper for using rundiff to compare gem5 outputs
  - `tracediff "a/gem5.opt|b/gem5.opt" -debug-flags=Exec`
    - Compares instructions traces from two builds of gem5
    - See comments for details

Advanced Trace Diffing

- Sometimes if you run into a nasty bug it's hard to compare apples-to-apples traces
  - Different cycles counts, different code paths from interrupts/timers

- Some mechanisms that can help:
  - `-ExecTicks` don't print out ticks
  - `-ExecKernel` don't print out kernel code
  - `-ExecUser` don't print out user code
  - `ExecAsid` print out ASID of currently running process

- State trace
  - PTRACE program that runs binary on real system and compares cycle-by-cycle to gem5
  - Supports ARM, x86, SPARC
  - See wiki for more information [http://gem5.org/Trace_Based_Debugging]

Checker CPU

- Runs a complex CPU model such as the O3 model in tandem with a special Atomic CPU model
- Checker re-executes and compares architectural state for each instruction executed by complex model at commit
- Used to help determine where a complex model begins executing instructions incorrectly in complex code

- Checker cannot be used to debug MP or SMT systems
- Checker cannot verify proper handling of interrupts
- Certain instructions must be marked unverifiable i.e. “wfi”

Remote Debugging

```
./build/ARM/gem5.opt configs/example/fs.py
gem5 Simulator System
...
command line: ./build/ARM/gem5.opt configs/example/fs.py
Global frequency set at 1000000000000 ticks per second
info: kernel located at: /dist/binaries/vmlinux.arm
Listening for system connection on port 5900
Listening for system connection on port 3456
0: system.remote_gdb.listener: listening for remote gdb #0 on
port 7000 info: Entering event queue @ 0. Starting
simulation...
```

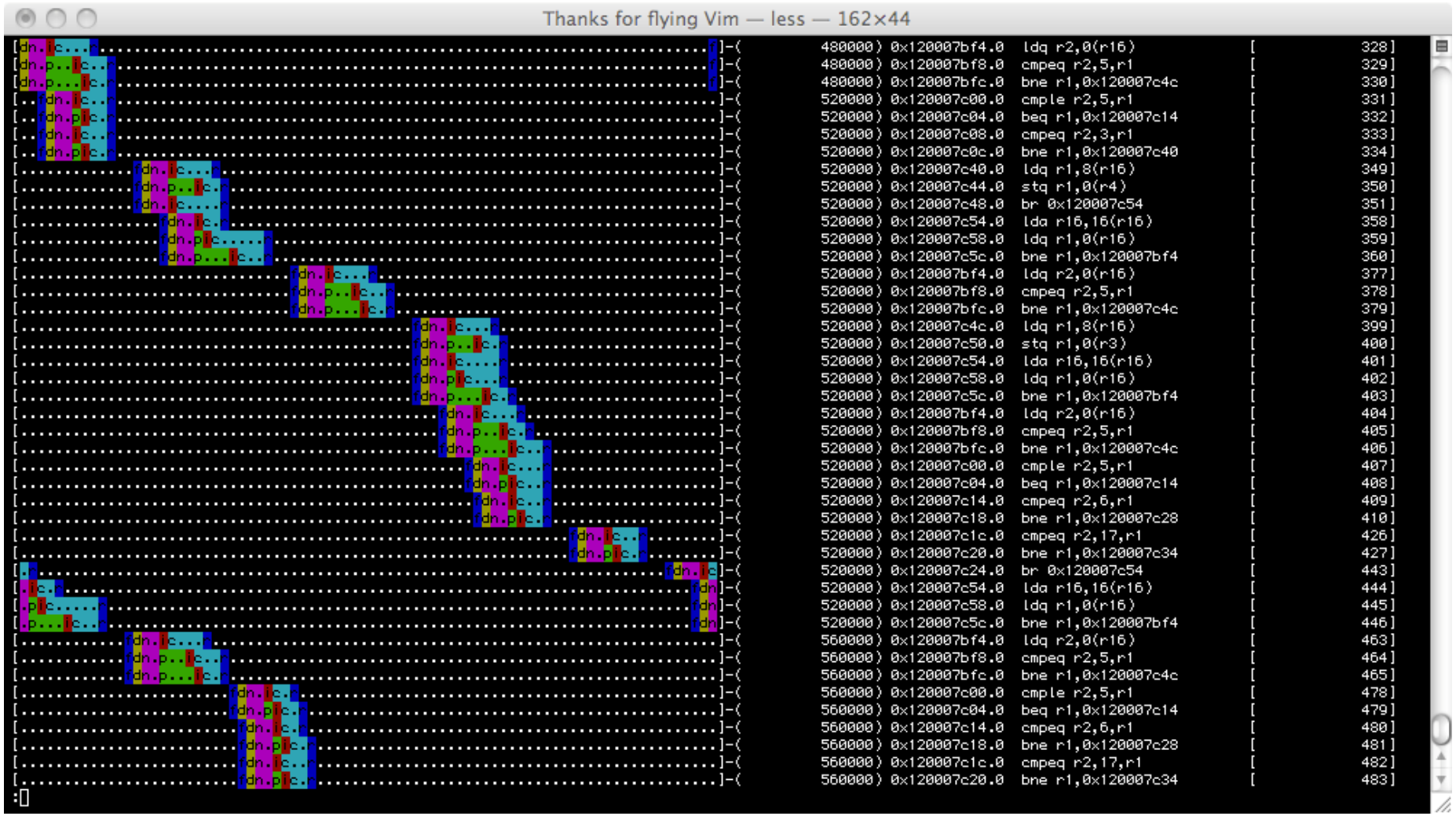
Remote Debugging

```
GNU gdb (Sourcery G++ Lite 2010.09-50) 7.2.50.20100908-cvs
Copyright (C) 2010 Free Software Foundation, Inc.
...
(gdb) symbol-file /dist/binaries/vmlinux.arm
Reading symbols from /dist/binaries/vmlinux.arm...done.
(gdb) set remote Z-packet on
(gdb) set tdesc filename arm-with-neon.xml
(gdb) target remote 127.0.0.1:7000
Remote debugging using 127.0.0.1:7000
cache_init_objs (cachep=0xc7c00240, flags=3351249472) at
mm/slab.c:2658
(gdb) step
sighand_ctor (data=0xc7ead060) at kernel/fork.c:1467
(gdb) info registers
r0  0xc7ead060   -940912544
r1  0x5201312
r2  0xc002f1e4   -1073548828
r3  0xc7ead060   -940912544
r4  0x00
r5  0xc7ead020   -940912608
…
```

ARMv7 only, ARMv8 doesn't need

O3 Pipeline Viewer

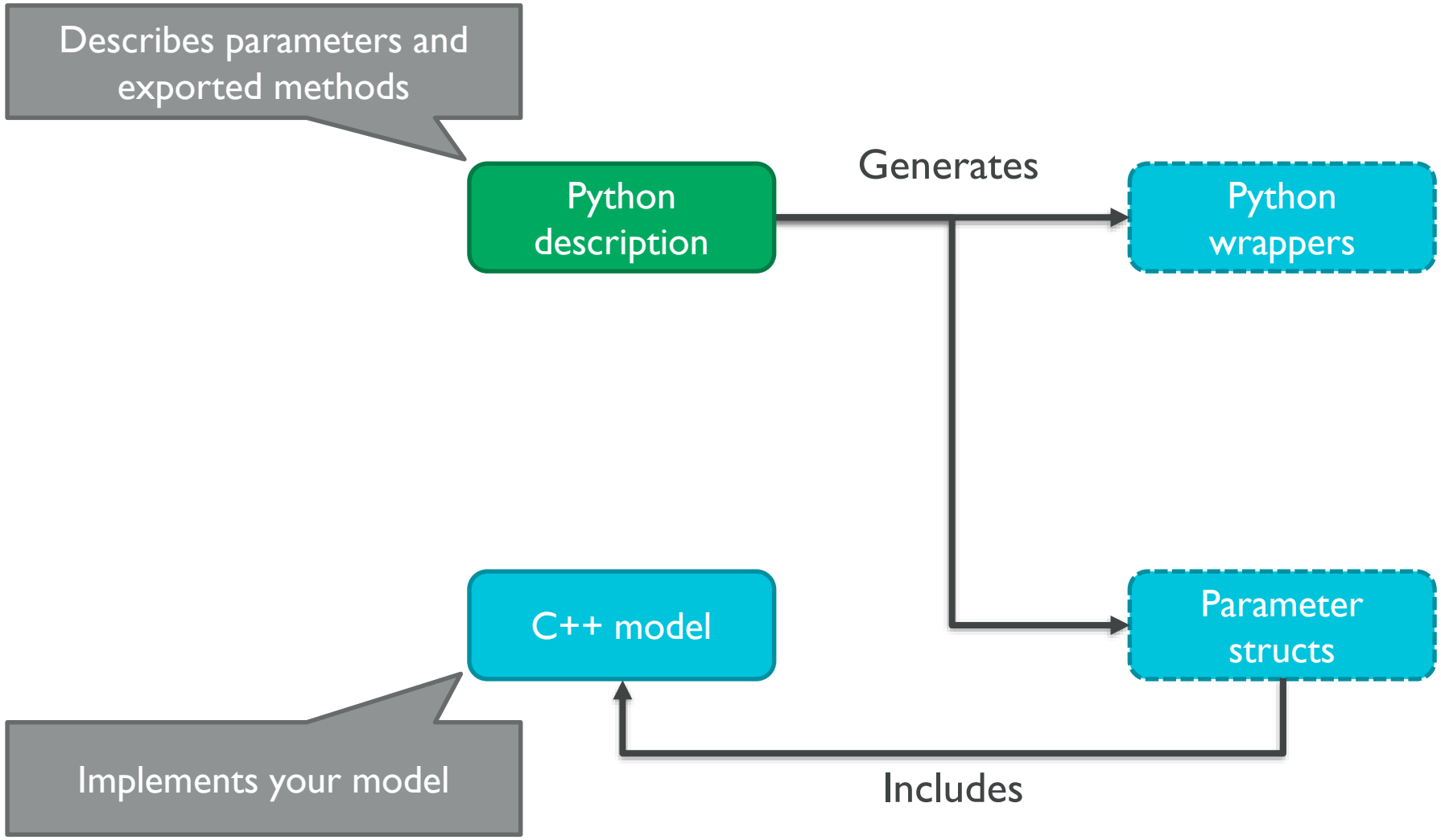
Use `--debug-flags=O3PipeView` and `util/o3-pipeview.py`



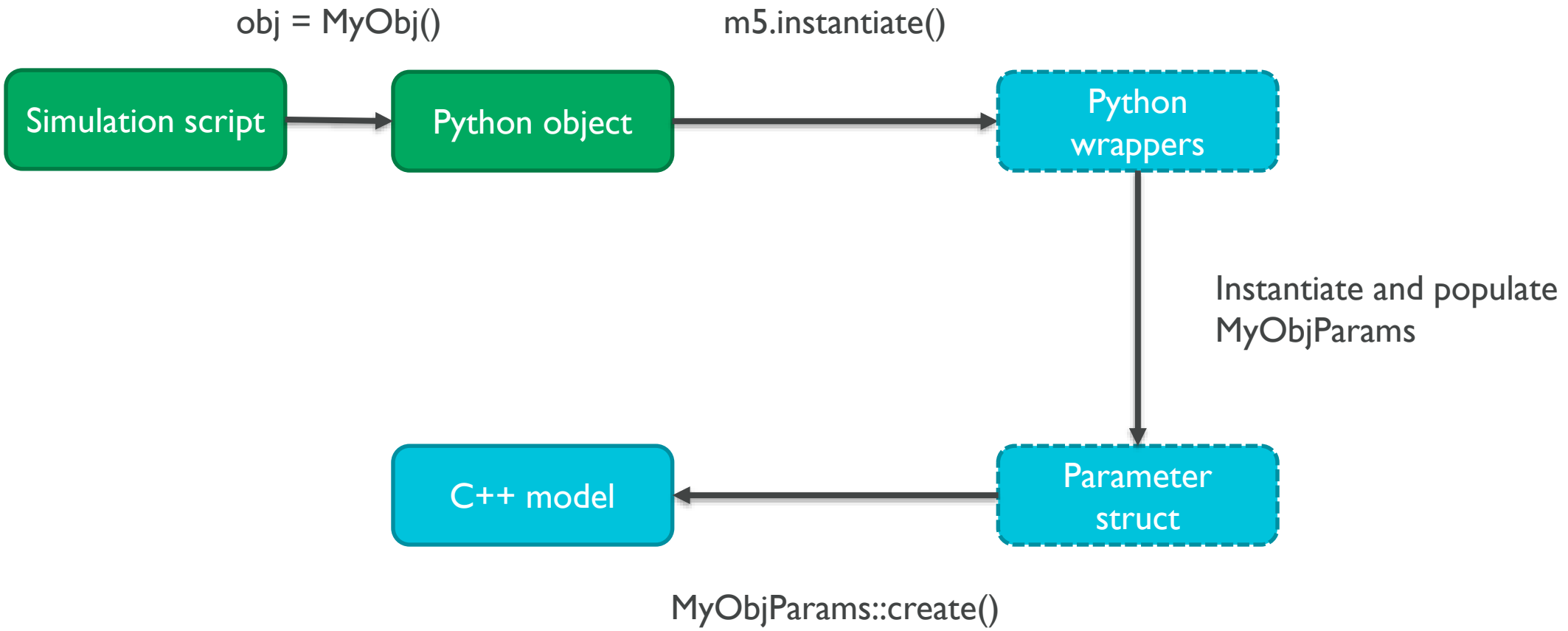
Adding new models

Andreas Sandberg

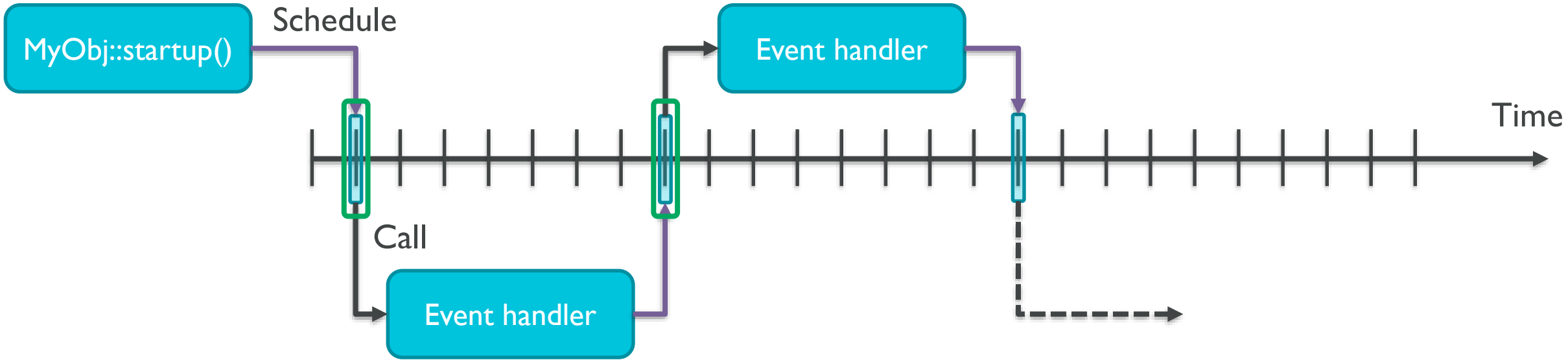
How are models implemented



How are models instantiated



Discrete event based simulation

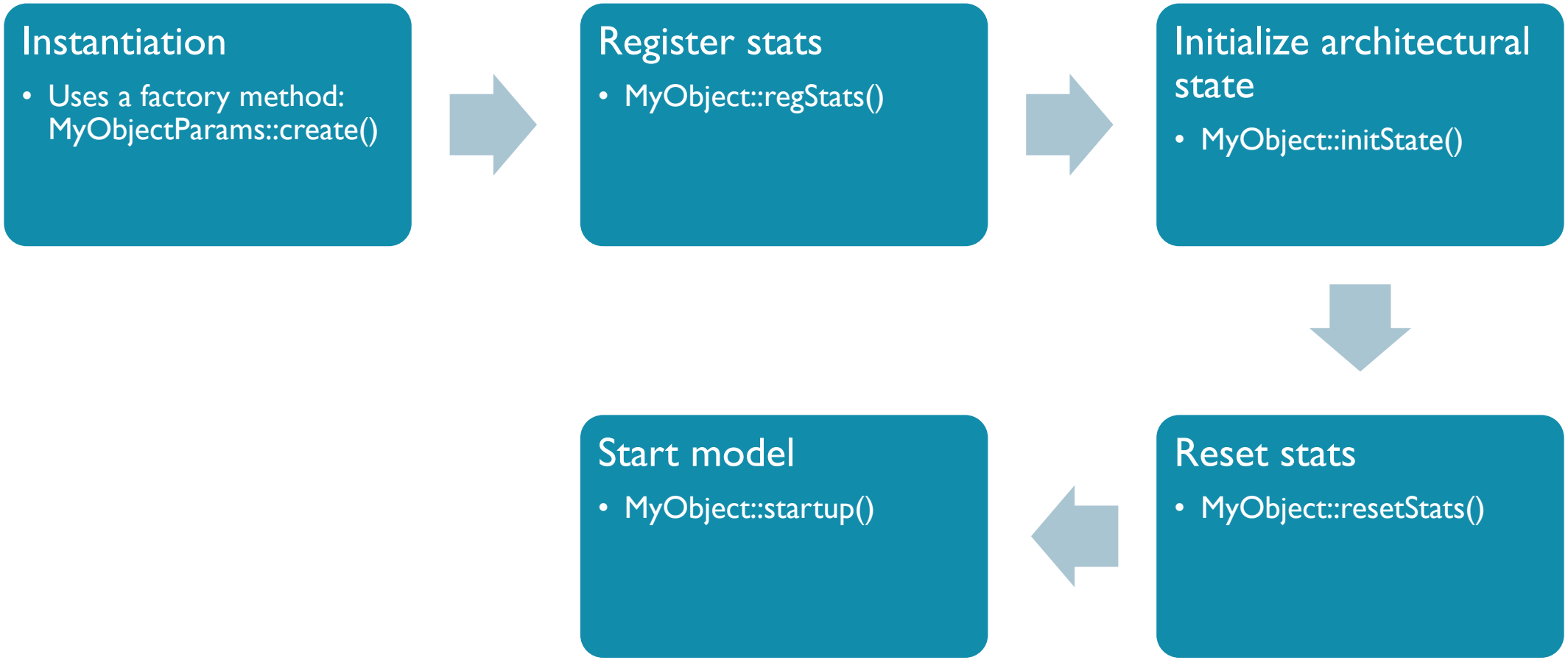


- Discrete: Handles time in discrete steps
  - Each step is a tick
  - Usually 1THz in gem5
- Simulator skips to the next event on the timeline

Creating a SimObject

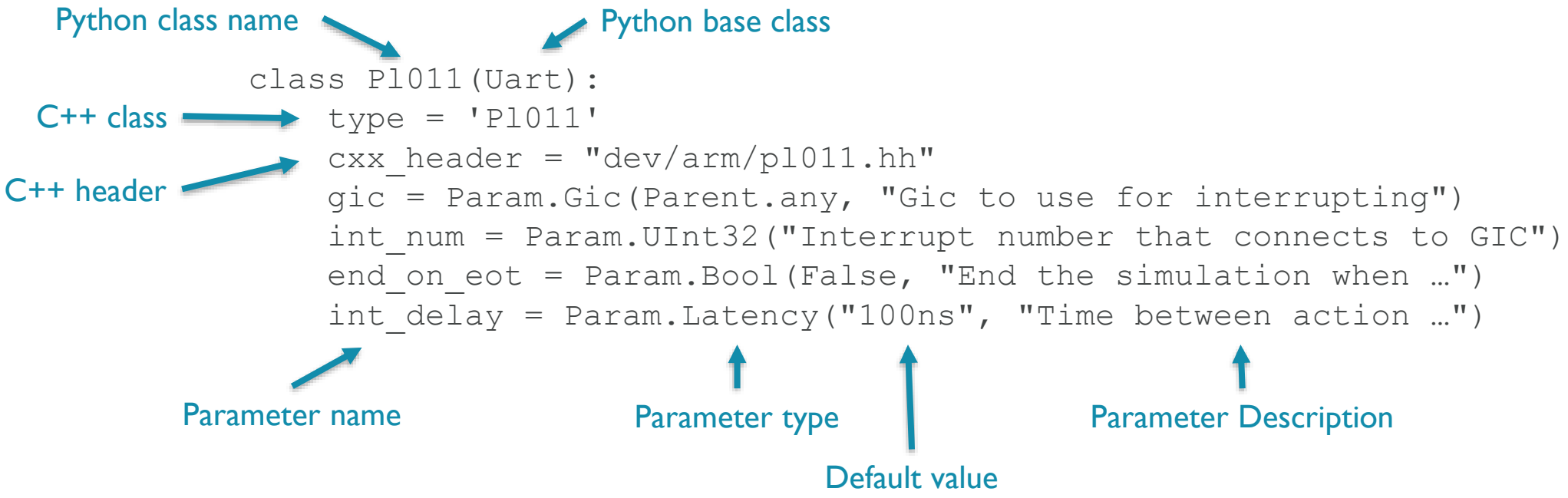
- Derive Python class from Python SimObject
  - Define parameters, ports and configuration
  - Parameters in Python are automatically turned into C++ struct and passed to C++ object
  - Add Python file to `SConscript`
    - Or, place it in an existing Python file
- Derive C++ class from C++ SimObject
  - Defines the simulation behavior
  - See `src/sim/sim_object.{cc,hh}`
  - Add C++ filename to `SConscript` in directory of new object
  - Need to make sure you have a create factory method for the object
    - Look at the bottom of an existing object for info
- Recompile

SimObject initialization



Parameters and SimObjects

- Parameters to SimObjects are synthesized from Python structures
  - Object hierarchy in Python reflects the C++ world
- This example is from src/dev/arm/Realview.py



SimObject Parameters

- Parameters can be:
  - Scalars – Param.Unsigned(5), Param.Float(5.0), Param.UInt32(42), …
  - Arrays – VectorParam.Unsigned([1,1,2,3])
  - SimObjects – Param.PhysicalMemory(…)
  - Arrays of SimObjects – VectorParam.PhysicalMemory(Parent.any)
  - Memory address ranges– Param.AddrRange(0,Addr.max))
- Normally converted from strings with units :
  - Latency – Param.Latency('15ns') Tick
  - Frequency – Param.Frequency('100MHz') -> Tick
  - MemorySize – Param.MemorySize('1GB') -> Bytes
  - Time – Param.Time('Mon Mar 25 09:00:00 CST 2012')
  - Ethernet Address – Param.EthernetAddr("90:00:AC:42:45:00")

Auto-generated Header file

```
#ifndef __PARAMS__Pl011__
#define __PARAMS__Pl011__

class Pl011;

#include <cstddef>
#include "base/types.hh”
#include "params/Gic.hh"
##include "base/types.hh"

#include "params/Uart.hh"

struct Pl011Params
    : public UartParams
{
    Pl011 * create();
    uint32_t int_num;
    Gic * gic;
    bool end_on_eot;
    Tick int_delay;
};
#endif // __PARAMS__Pl011__
```

Factory method

```
class Pl011(Uart):
    type = 'Pl011'
    gic = Param.Gic(Parent.any, …)
    int_num = Param.UInt32(…)
    end_on_eot = Param.Bool(False, "End …)
    int_delay = Param.Latency("100ns", "Time …")
```

How Parameters are used in C++

src/dev/arm/pl011.cc:

```
Pl011::Pl011(const Pl011Params *p)
    : Uart(p), …,
      intNum(p->int_num), gic(p->gic),
      endOnEOT(p->end_on_eot), intDelay(p->int_delay)
{
…
}
```

You can also access parameters through params() accessor after instantiation.

Creating/Using Events

- One of the most common things in an event driven simulator is scheduling events
  - Declaring events and handlers is easy:

```
/* Handle when a timer event occurs */
void timerHappened();
EventWrapper<MyClass, &MyClass::timerHappend> event;
```

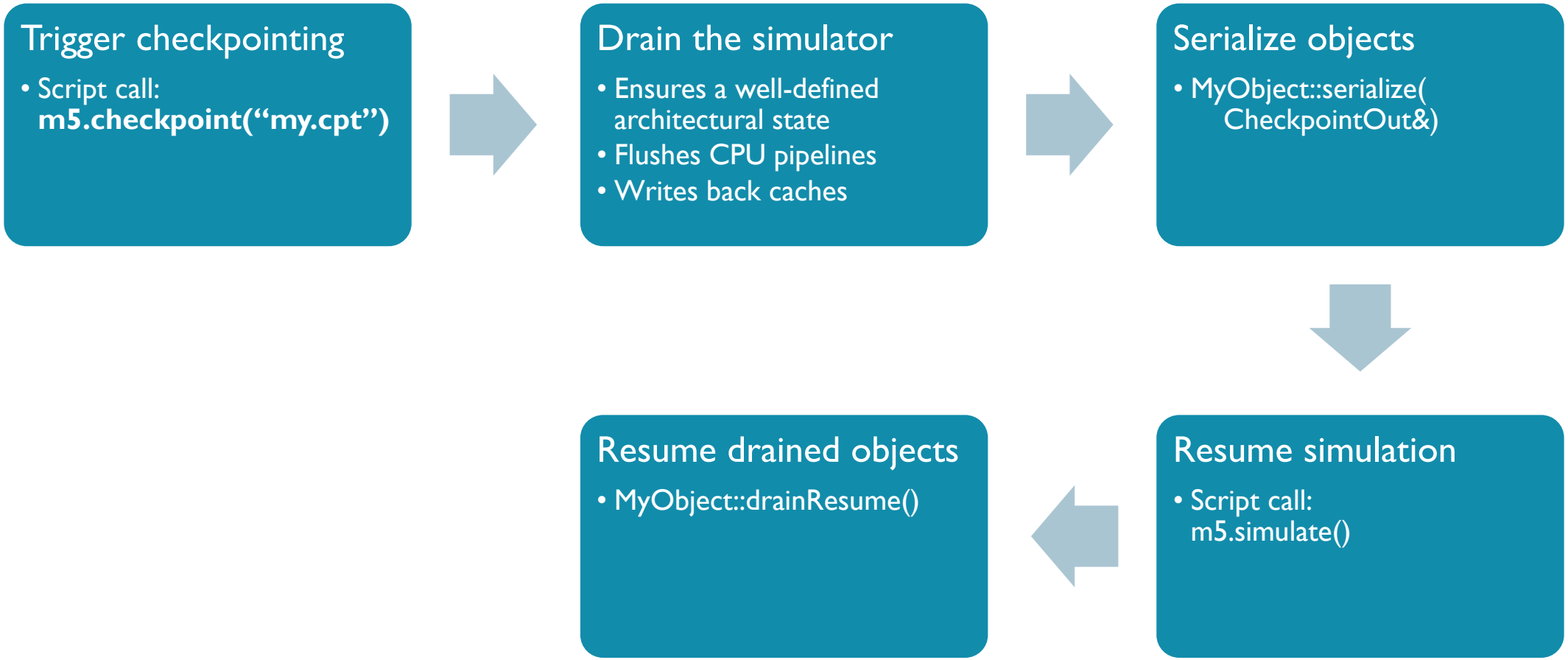
  - Scheduling them is easy too:

```
/* something that requires me to schedule an event  at time t*/
if (event.scheduled())
     reschedule(event, curTick() + t);
else
     schedule(event, curTick() + t);
```

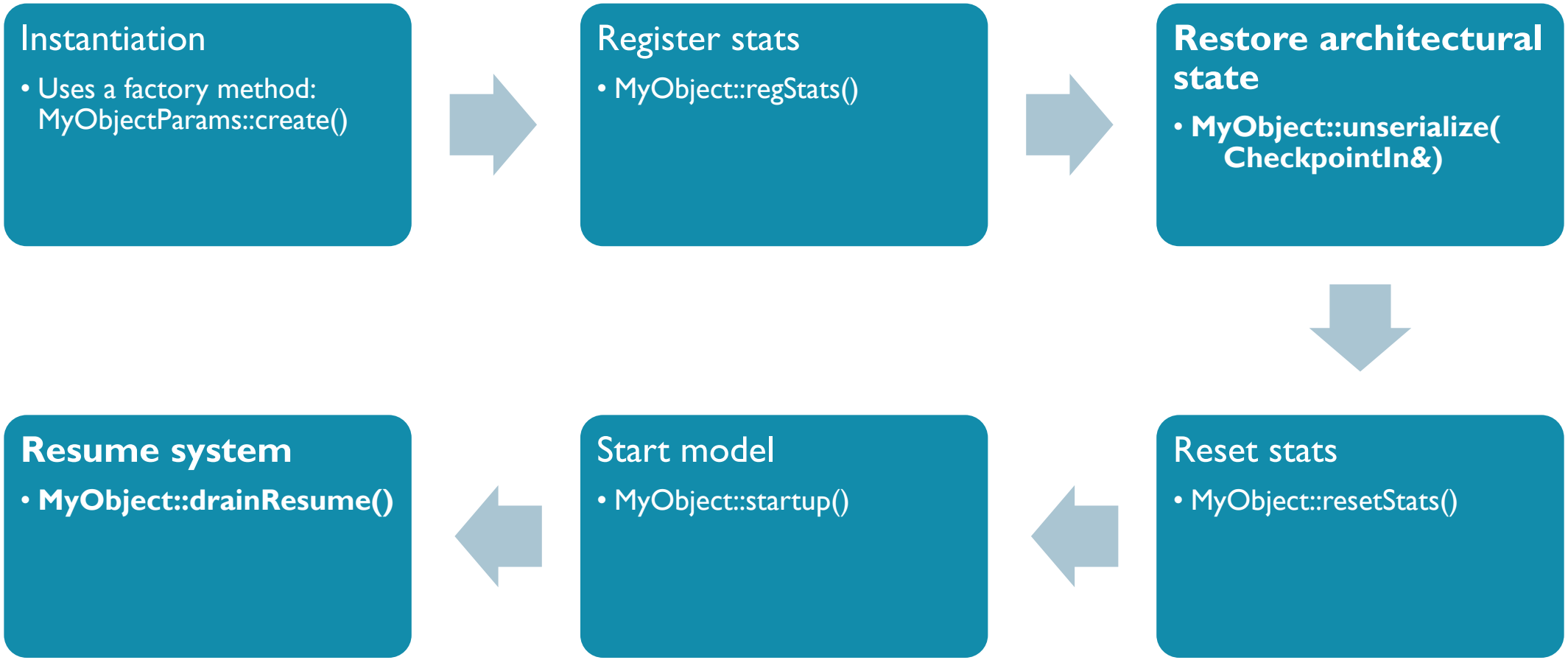
Checkpointing SimObject State

- If your object has state, that needs to be written to the checkpoint
- Checkpointing takes place on a *drained* simulator
  - Draining ensures that microarchitectural state is flushed
  - Models may need to flush pipelines and wait for outstanding requests to finish.
- Checkpoint implemented by overriding `SimObject::serialize(CheckpointOut &)`
  - Save necessary state
  - No need to store parameters from the config systyem!
  - Use `SERIALIZE_*()` macros or paramOut
- To implement restore, override `SimObject::unserialize(CheckpointIn &)`
  - Use `UNSERIALIZE_*()` macros or paramIn

Creating a checkpoint

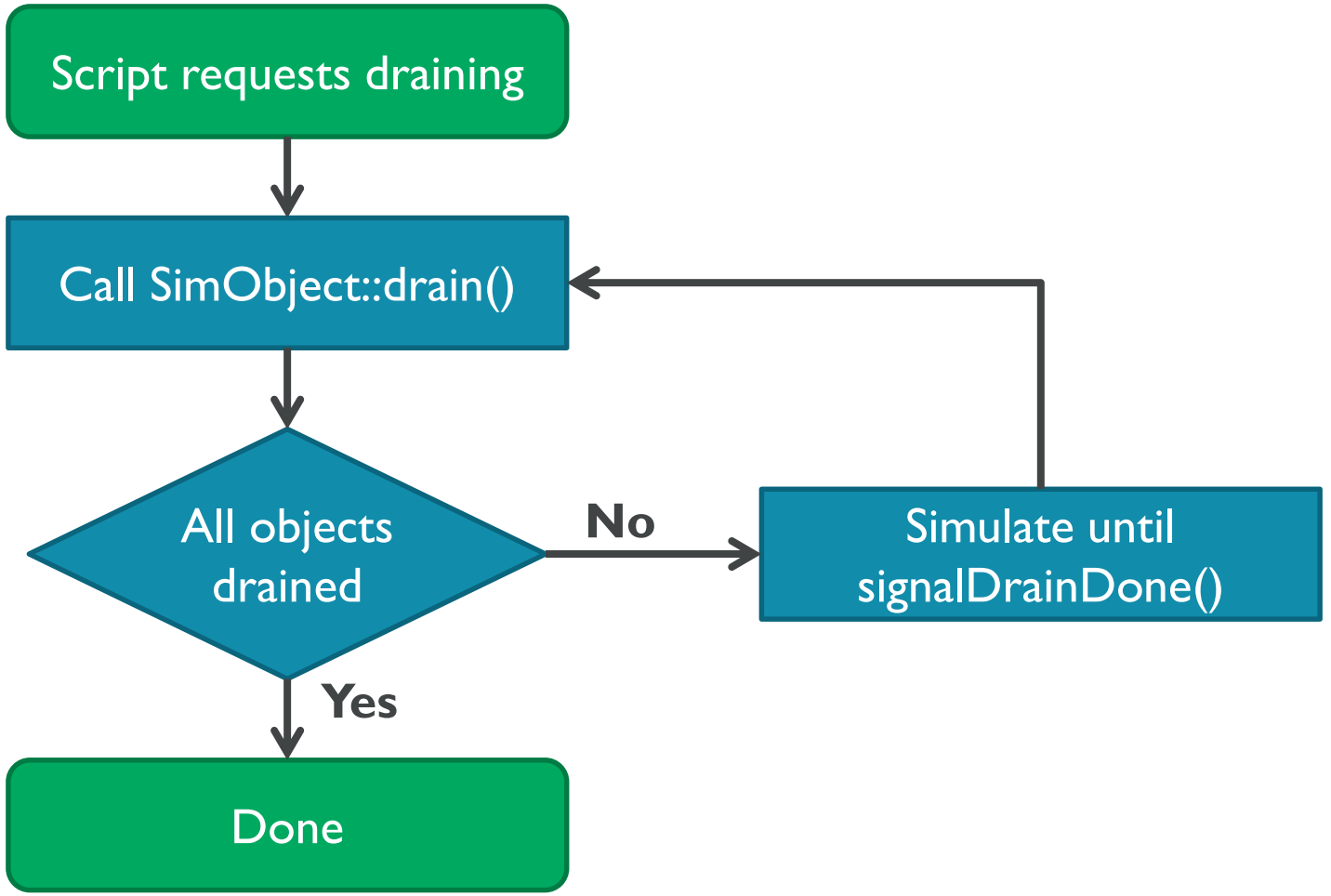


Restoring from a checkpoint



Draining

- Flush internal state
- Stop producing new messages



Checkpointing Example

```
// uint16_t control;
void
Pl011::serialize(CheckpointOut &cp) const
{
    SERIALIZE_SCALAR(control);
}

void
Pl011::unserialize(CheckpointIn &cp)
{
    UNSERIALIZE_SCALAR(control);
}
```

Good Examples

- Simple IO devices: `IsaFake`
  - See: `src/dev/isa_fake.{cc,hh}` and `src/dev/Device.py`
  - Demonstrates a basic memory-mapped device using the `BasicPioDevice` base class
- PCI devices: `PciVirtIO`
  - See: `src/dev/virtio/pci.{cc,hh}` and `src/dev/VirtIO.py`
  - PCI device with a single BAR and interrupts
- More complex PCI device: `CopyEngine`
  - See: `src/dev/pci/copy_engine.{cc,hh}` and `src/dev/pci/CopyEngine.py`
  - PCI device with DMA support
- Python exports: `PowerModelState`
  - See: `src/sim/power/PowerModelState.py`
  - Exports two methods (`getDynamicPower` & `getStaticPower`) to Python

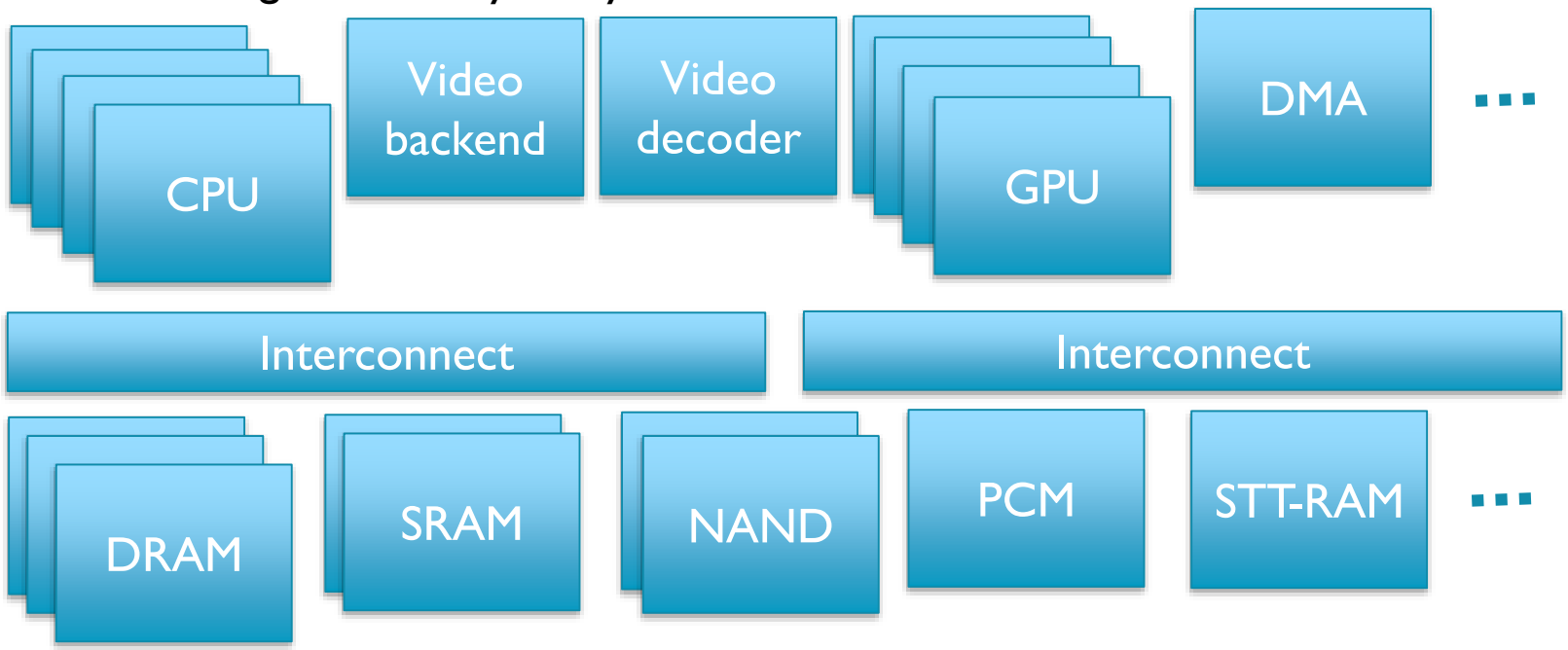
<Insert coffee break here>

Memory System

Stephan Diestelhorst

Goals

- Model a system with heterogeneous applications, running on a set of heterogeneous processing engines, using heterogeneous memories and interconnect
  - CPU centric: capture memory system behaviour accurate enough
  - Memory centric: Investigate memory subsystem and interconnect architectures

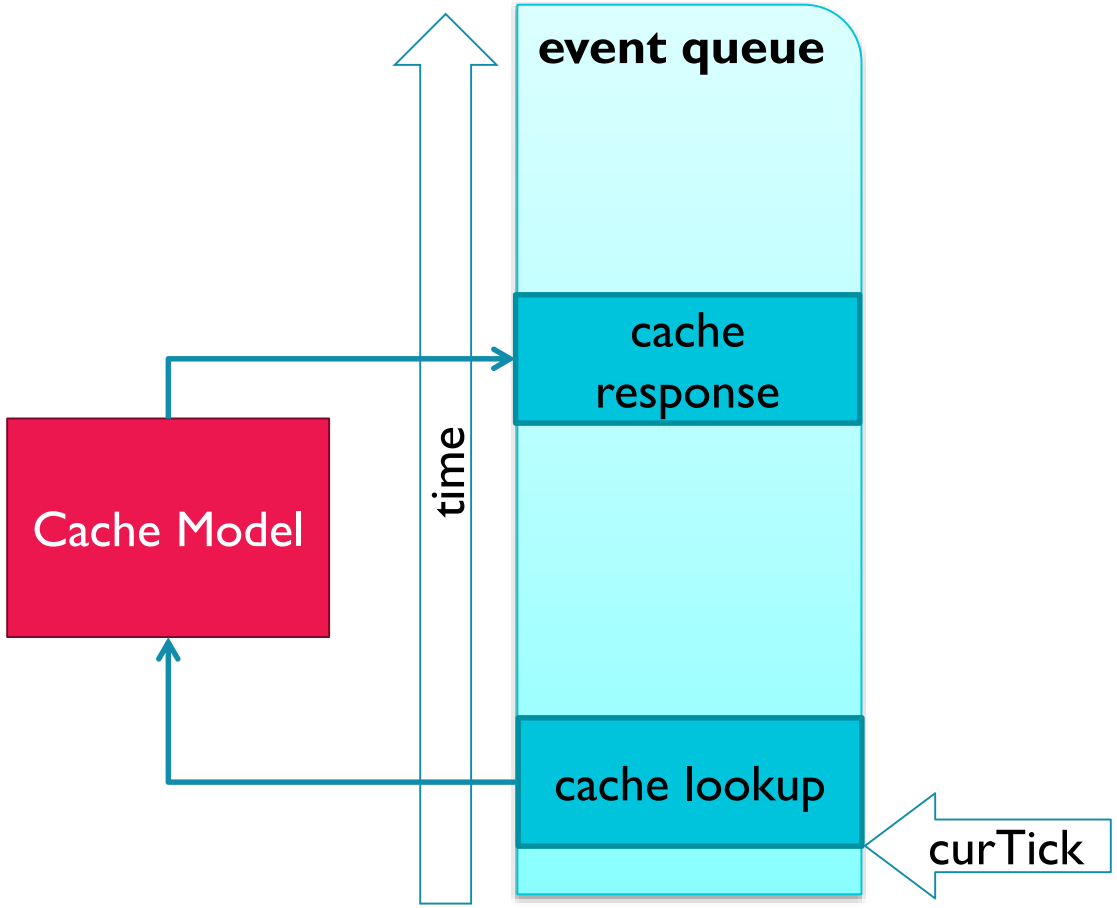


Goals, contd.

- Two worlds...
  - Computation-centric simulation
    - e.g. SimpleScalar, Asim etc
    - More behaviourally oriented, with ad-hoc ways of describing parallel behaviours and intercommunication
  - Communication-centric simulation
    - e.g. SystemC+TLM2 (IEEE standard)
    - More structurally oriented, with parallelism and interoperability as a key component
- gem5 is trying to balance
  - Easy to extend (flexible)
  - Easy to understand (well defined)
  - Fast enough (to run full-system simulation at MIPS)
  - Accurate enough (to draw the right conclusions)

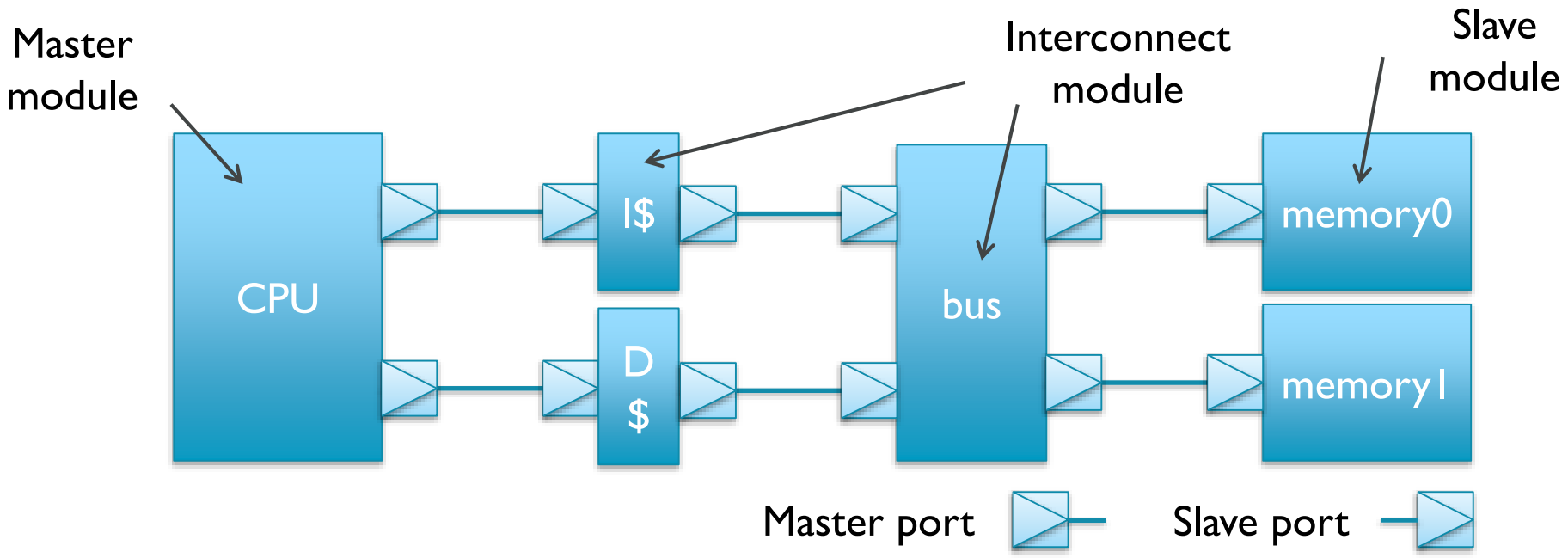
Event Simulation

- Event-driven
  - no activity -> no clocking
  - event queue
- Deterministic
  - fixed random number seed
  - no dependence on host addresses
- Multi-Queue
  - multiple workers



Ports, Masters and Slaves

- MemObjects are connected through master and slave ports
- A master module has at least one master port, a slave module at least one slave port, and an interconnect module at least one of each
  - A master port always connects to a slave port
  - Similar to TLM-2 notation



Transport interfaces

- Atomic
  - Similar to loosely timed in TLM
  - Blocking: Requests completes in a single call chain
  - Each component along the way adds latency to the request
- Timing
  - Similar to approximately timed in TLM
  - Asynchronous: One call to send a packet, callback when response is ready.
- Functional
  - Debug interface that doesn't affect coherency states.
  - Blocking: Requests complete within a single call chain.

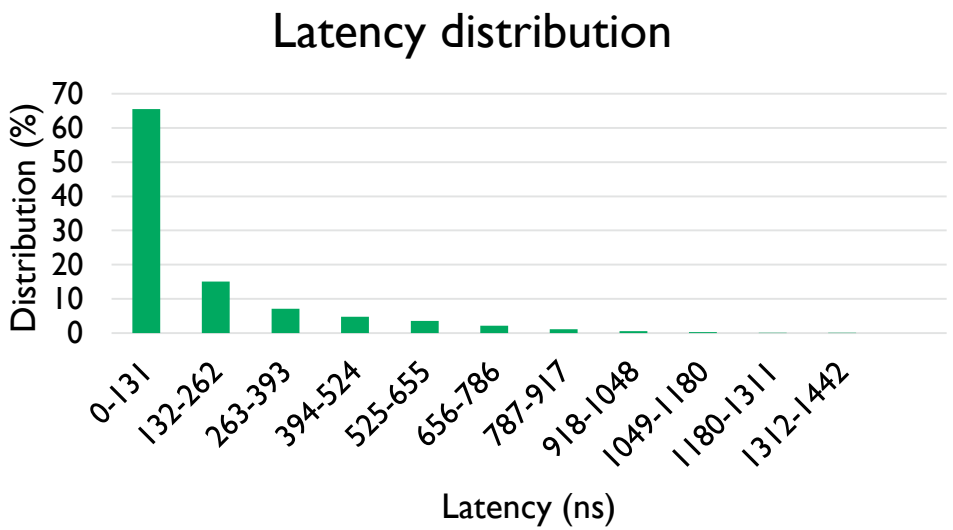
The Atomic and Timing interfaces are mutually exclusive

Communication Monitor

- Insert as a structural component where stats are desired

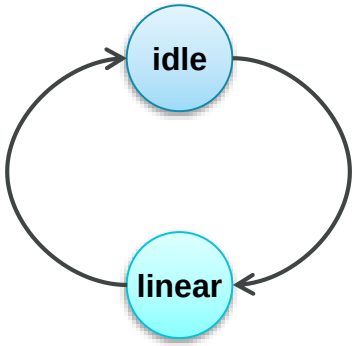
```
memmonitor = CommMonitor()
membus.master = memmonitor.slave
memmonitor.master = memctrl.slave
```

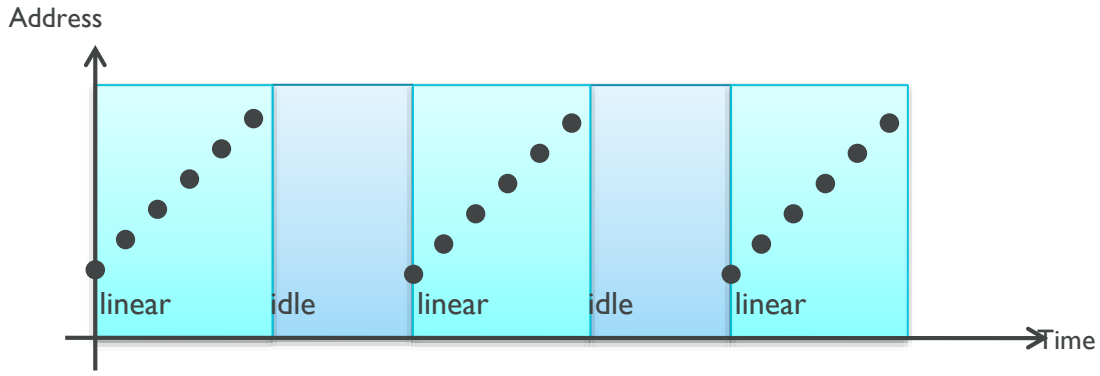
- A wide range of communication stats
  - bandwidth, latency, inter-transaction (read/write) time, outstanding transactions, address heatmap, etc
- Provides an attachment point for communication probes:
  - Tracing (using protobuf)
  - Stack distance monitoring
  - Footprint estimation



Traffic generator

- Test scenarios for memory system regression and performance validation
  - High-level of control for scenario creation
- Black-box models for components that are not yet modeled
  - Video/baseband/accelerator for memory-system loading
- Inject requests based on (probabilistic) state-transition diagrams
  - Idle, random, linear and trace replay states



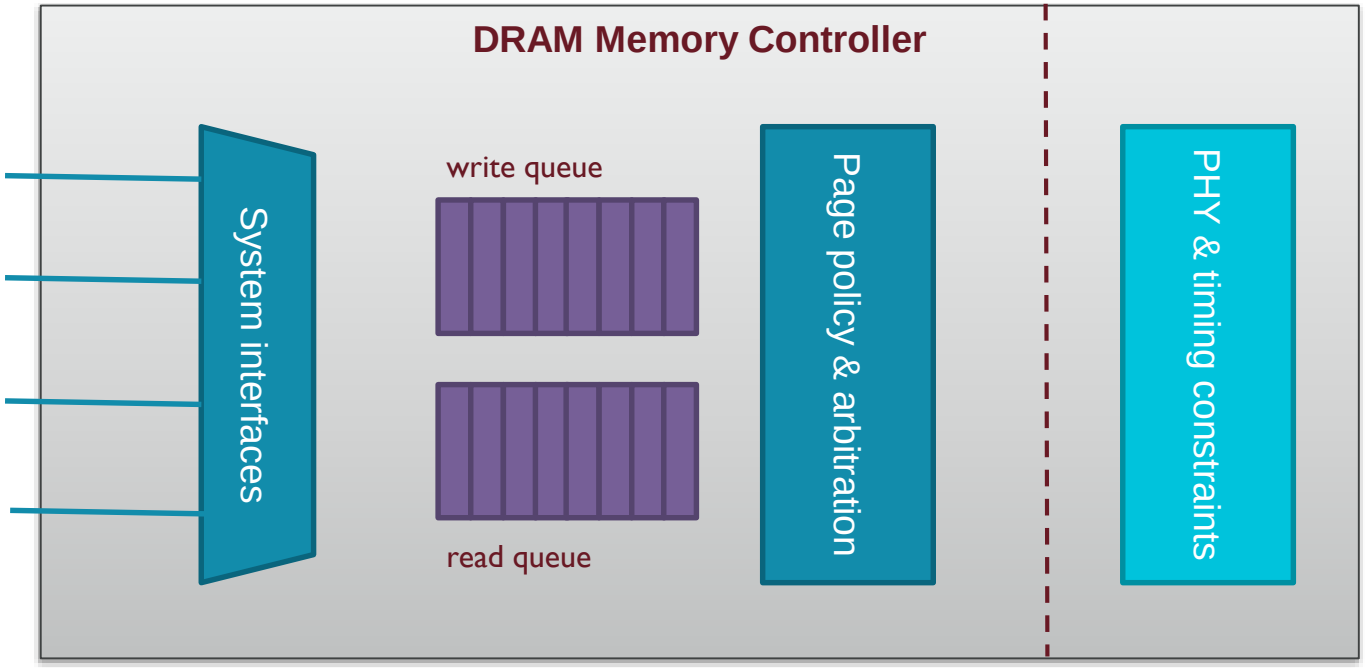


Memory controllers

- All memories in the system inherit from AbstractMemory
  - Basic single-channel memory controller
    - Instantiate multiple times if required
    - Interleaving support added in the bus/crossbar (to be posted)
- SimpleMemory
  - Fixed latency (possibly with a variance)
  - Fixed throughput (request throttling without buffering)
- SimpleDRAM
  - High-level configurable DRAM controller model to mimic DDRx, LPDDRx, WideIO, HBM etc
    - Memory organization: ranks, banks, row-buffer size
    - Controller architecture: Read/write buffers, open/close page, mapping, scheduling policy
    - Key timing constraints: tRCD, tCL, tRP, tBURST, tRFC, tREFI, tTAW/tFAW

Top-down controller model

- Don't model the actual DRAM, only the timing constraints
  - DDR3/4, LPDDR2/3/4, WIO1/2, GDDR5, HBM, HMC, even PCM
  - See *src/mem/DRAMCtrl.py* and *src/mem/dram_ctrl.{hh, cc}*



Device width
Burst length
#ranks, #banks
Page size

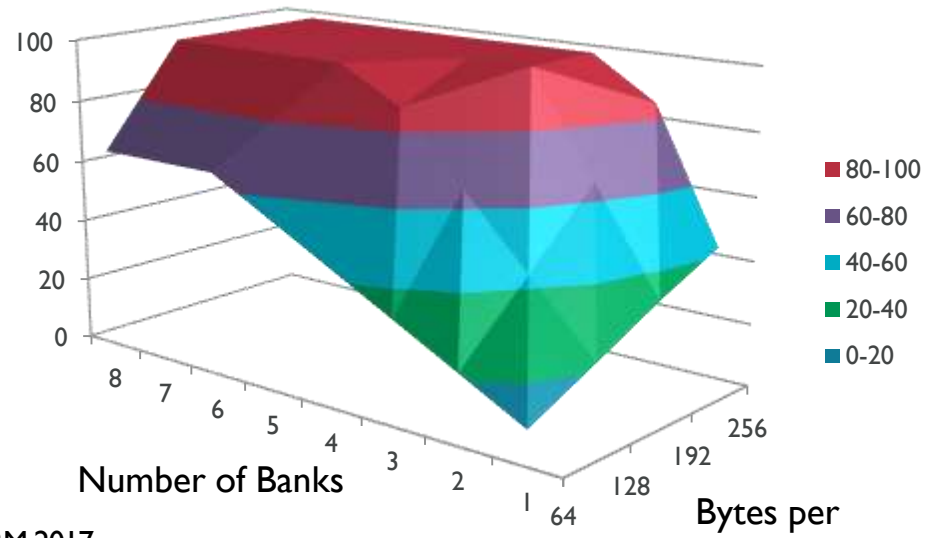
tRCD
tCL
tRP
tRAS
tBURST
tRFC & tRFEI
tWTR
tRRD
tFAW/tTAW
…

Hansson et al, *Simulating DRAM controllers for future system architecture exploration*, ISPASS'14

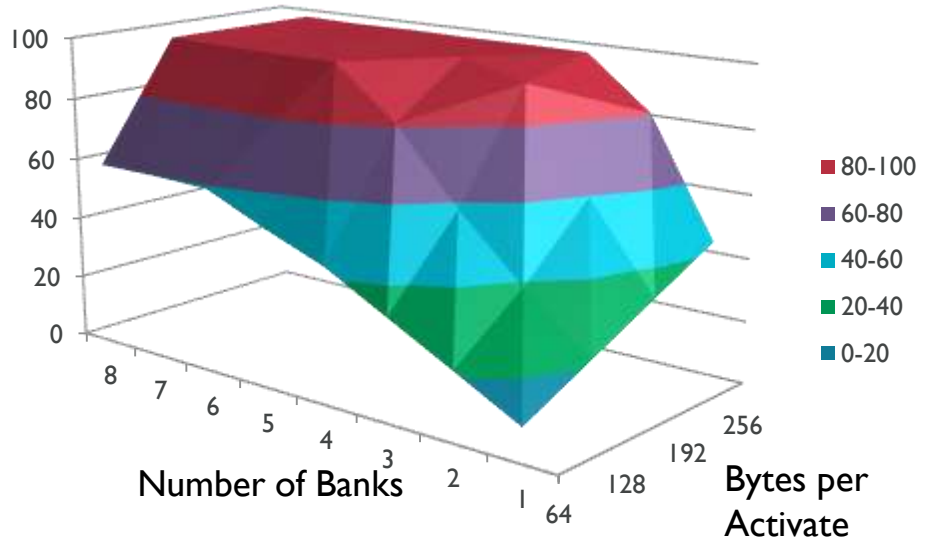
Controller model correlation

- Comparing with a real memory controller
  - Synthetic traffic sweeping bytes per activate and number of banks
    - *See configs/dram/sweep.py* and *util/dram_sweep_plot.py*

gem5 model

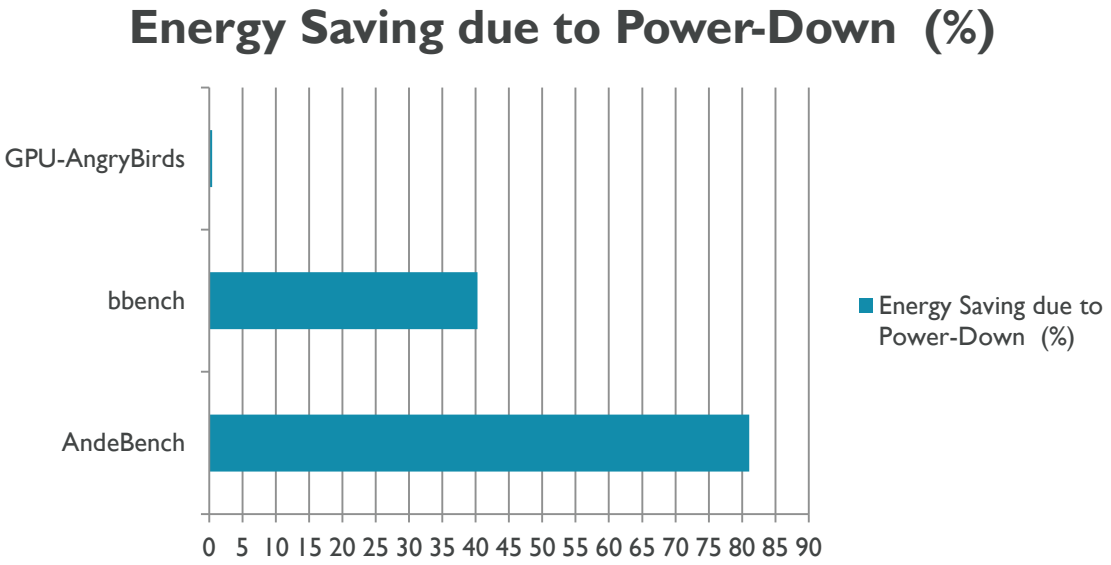


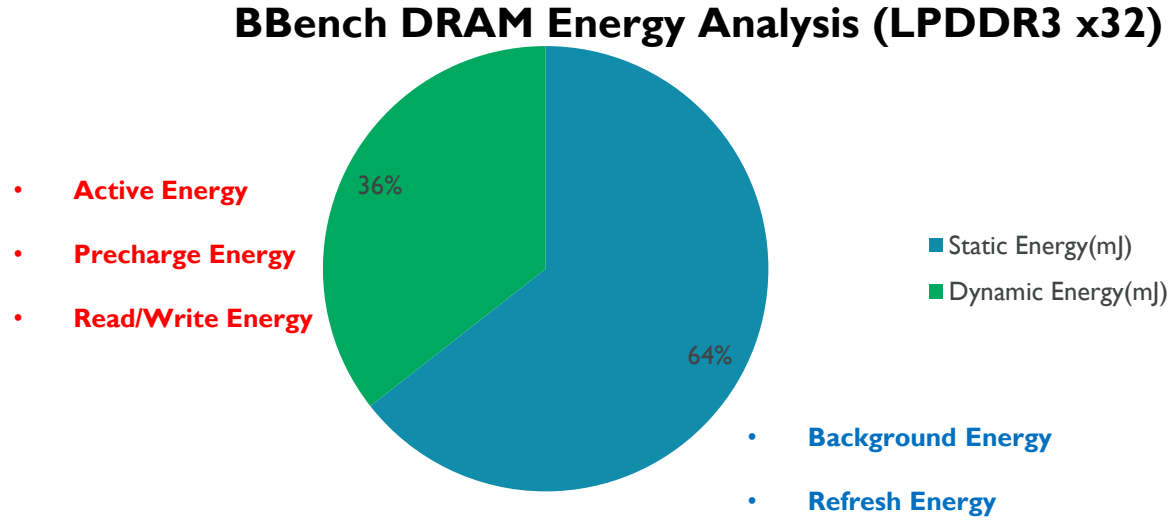
Real memory controller



DRAM power modeling

- DRAM accounts for a large portion of system power
  - Need to capture power states, and system impact
- Integrated model opens up for developing more clever strategies
  - DRAMPower adapted and adopted for gem5 use-case

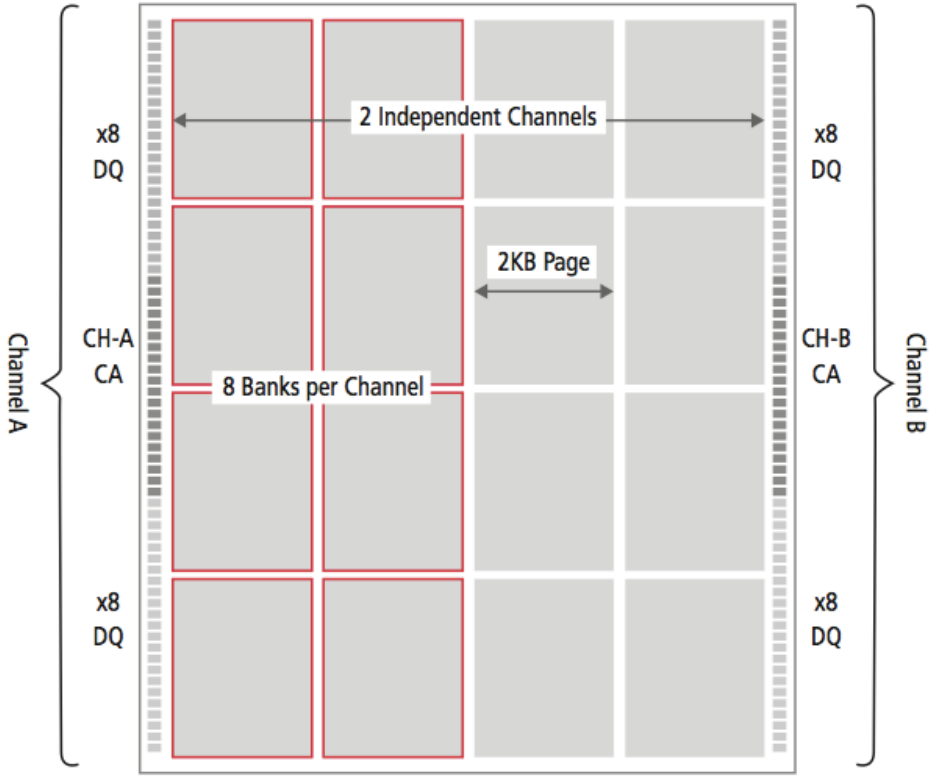




Naji et al, *A High-Level DRAM Timing, Power and Area Exploration Tool*, SAMOS'15

Address interleaving

- Multi-channel memory support is essential
  - Emerging DRAM standards are multi-channel by nature (LPDDR4, WIO1/2, HBM1/2, HMC)
- Interleaving support added to address range
  - Understood by memory controller and interconnect
  - See *src/base/addr_range.hh* for matching and *src/mem/xbar.{hh, cc}* for actual usage
  - Interleaving not visible in checkpoints
- XOR-based hashing to avoid imbalances
  - Simple yet effective, and widely published
  - See *configs/common/MemConfig.py* for system configuration

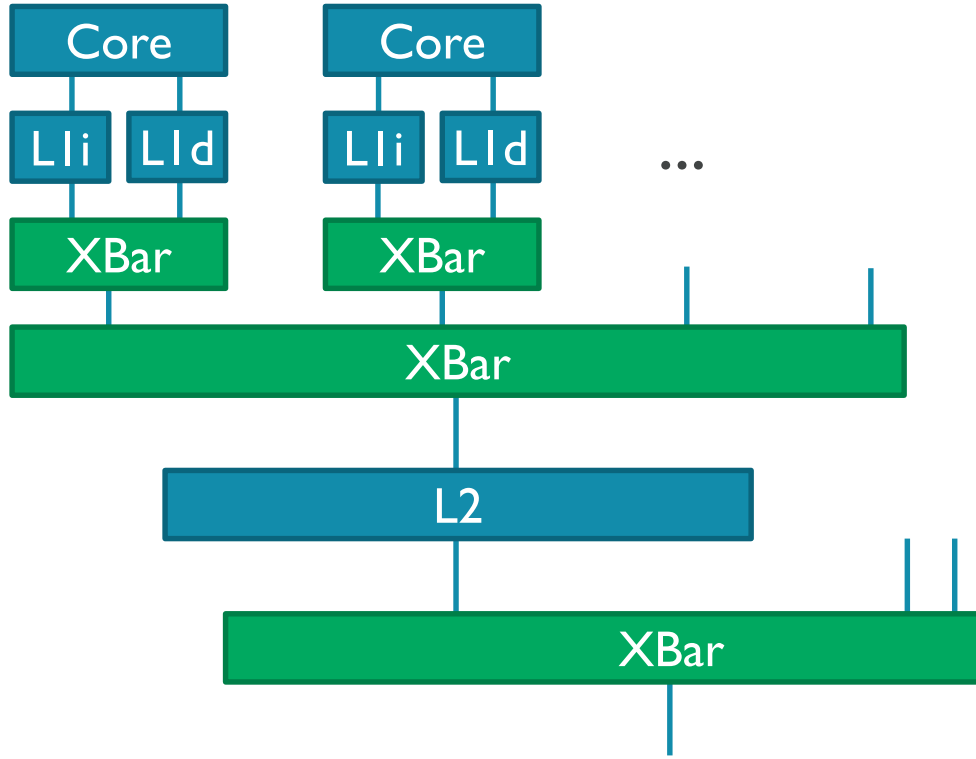


Source: Micron

ARM

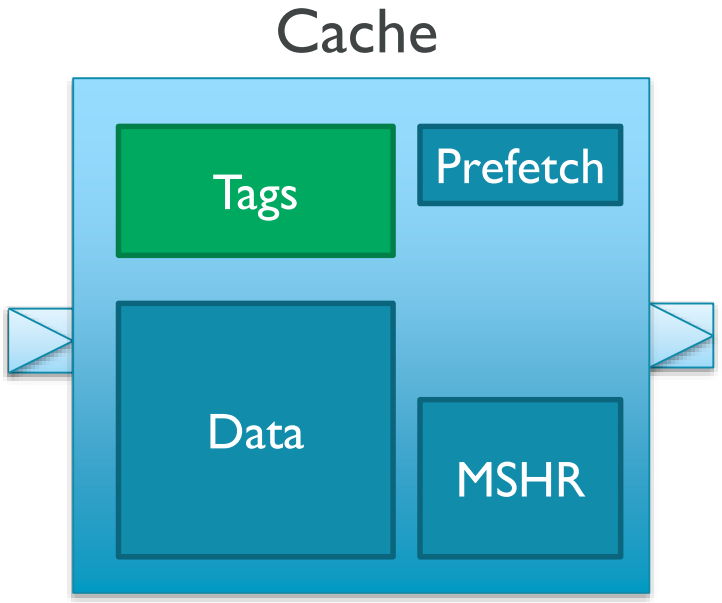
Crossbars& Bridges

- Create rich system interconnect topologies using a simple bus model and bus bridge
- Crossbars do address decoding and arbitration
  - Distributes snoops and aggregates snoop responses
  - Routes responses
  - Configurable width and clock speed
- Bridges connects two buses
  - Queues requests and forwards them
  - Configurable amount of queuing space for requests and responses



Caches

- Single cache model with several components:
  - Cache: request processing, miss handling, coherence
  - Tags: data storage and replacement (LRU, Random, etc.)
  - Prefetcher: N-Block Ahead, Tagged Prefetching, Stride Prefetching
  - MSHR & MSHRQueue: track pending/outstanding requests
    - Also used for write buffer
  - Parameters: size, hit latency, block size, associativity, number of MSHRs (max outstanding requests)

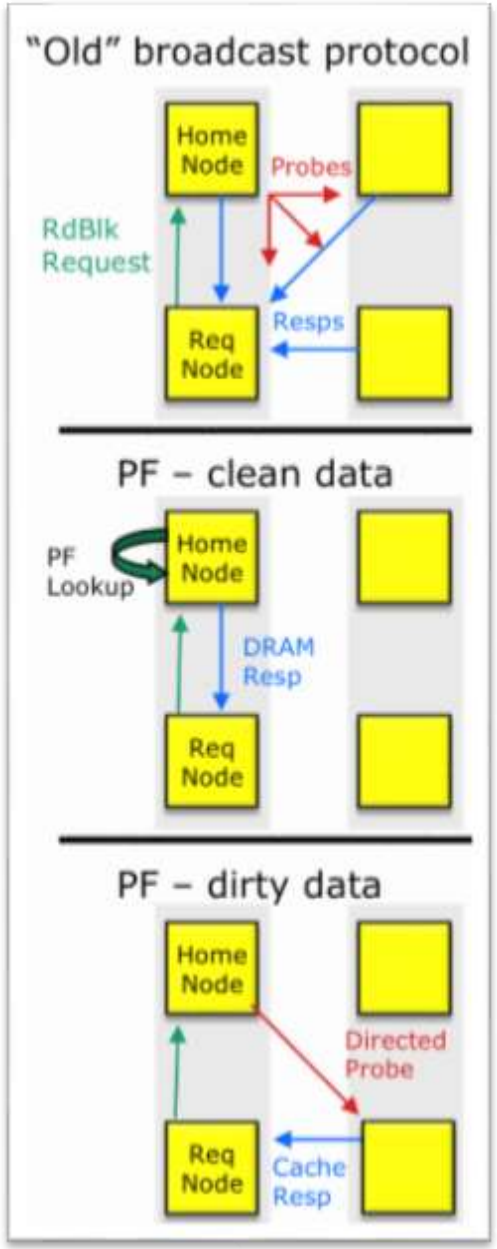


Coherence protocol

- MOESI bus-based snooping protocol
  - Support nearly arbitrary multi-level hierarchies at the expense of some realism
- Does not enforce inclusion
- Magic "express snoops" propagate upward in zero time
  - Avoid complex race conditions when snoops get delayed
  - Timing is similar to some real-world configurations
    - L2 keeps copies of all L1 tags
    - L2 and L1s snooped in parallel

Snoop (probe) filtering

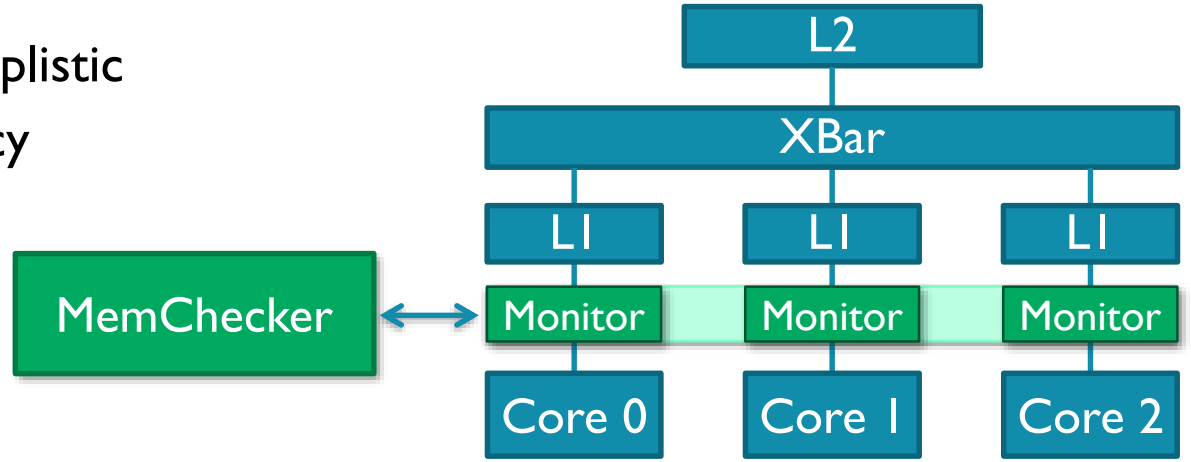
- Broadcast-based coherence protocol
  - Incurs performance and power cost
  - Does not reflect realistic implementations
- Snoop filter goes one step towards directories
  - Track sharers, based on writeback and clean eviction
  - Direct snoops and benefit from locality
- Many possible implementations
  - Currently ideal (infinite), no back invalidations
  - Can be used with coherent crossbars on any level
  - *See src/mem/SnoopFilter.py* and *src/mem/snoop_filter.{hh, cc}**



Source: AMD

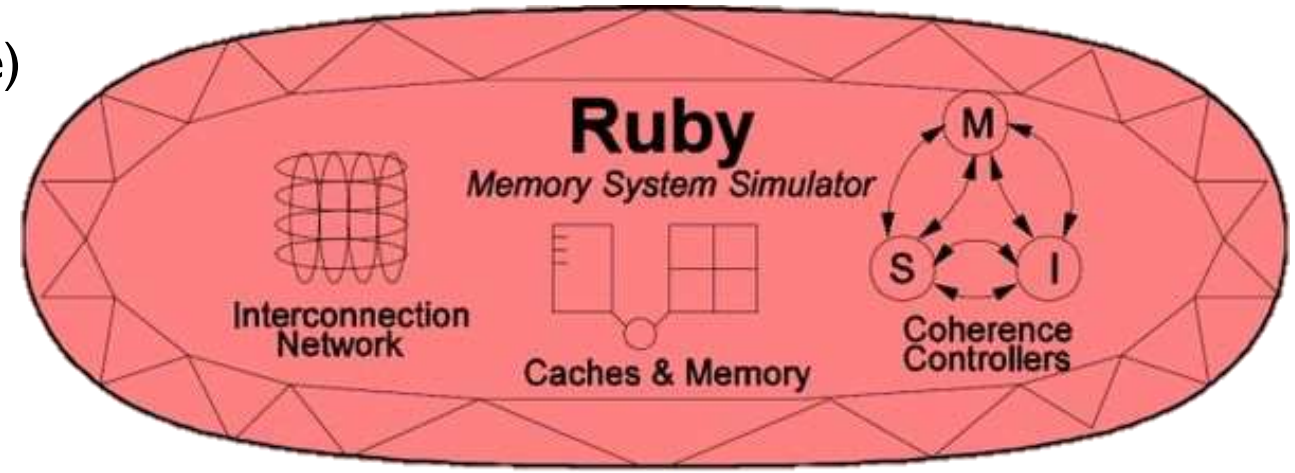
Memory system verification

- Check adherence to consistency model
  - Notion of functional reference memory is too simplistic
  - Need to track valid values according to consistency model
- Memory checker and monitors
  - Tracking in *src/mem/MemChecker.py* and *src/mem/mem_checker.{hh, cc}*
  - Probing in *src/mem/mem_checker_monitor.{hh, cc}*
- Revamped testing
  - Complex cache (tree) hierarchies in *configs/examples/{memtest, memcheck}.py*
  - Randomly generated soak test in *util/memtest-soak.py*
  - For any changes to the memory system, please use these



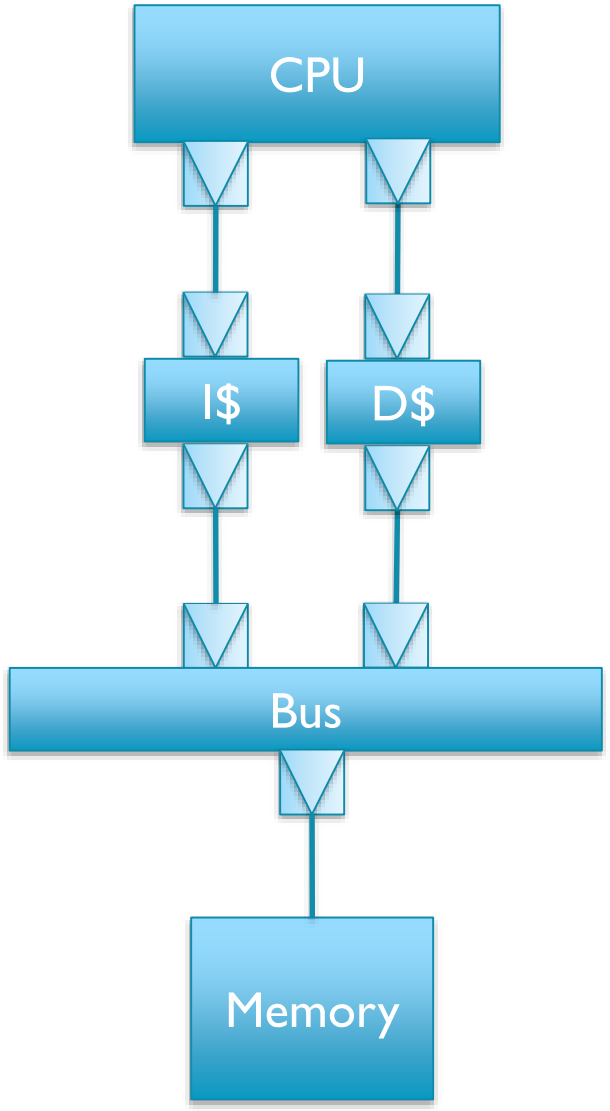
Ruby for Networks and Coherence

- As an alternative to its native memory system gem5 also integrates Ruby
- Create networked interconnects based on domain-specific language (SLICC) for coherence protocols
- Detailed statistics
  - e.g., Request size/type distribution, state transition frequencies, etc...
- Detailed component simulation
  - Network (fixed/flexible pipeline and simple)
  - Caches (Pluggable replacement policies)
- Supports Alpha and x86
  - Limited ARM support about to be added
  - Limited support for functional accesses



Instantiating and Connecting Objects

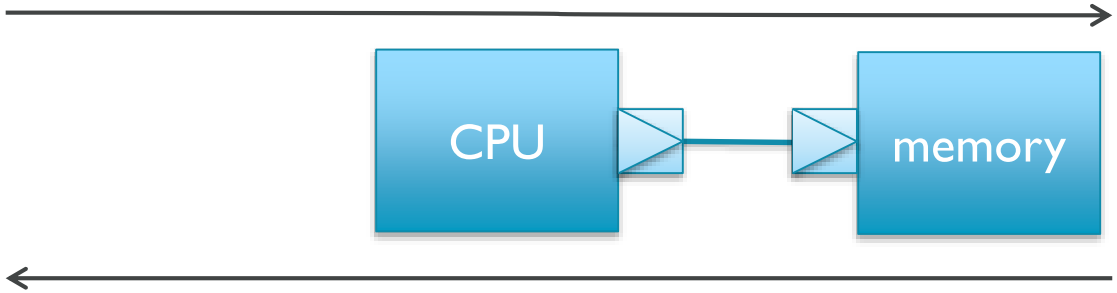
```
class BaseCPU(MemObject):
    icache_port = MasterPort("Instruction Port")
    dcache_port = MasterPort("Data Port")
    …
class BaseCache(MemObject):
    cpu_side = SlavePort("Port on side closer to CPU")
    mem_side = MasterPort("Port on side closer to MEM")
    ...
class Bus(MemObject):
    slave = VectorSlavePort("vector port for connecting masters")
    master = VectorMasterPort("vector port for connecting slaves")
    …
system.cpu.icache_port = system.icache.cpu_side
system.cpu.dcache_port = system.dcache.cpu_side

system.icache.mem_side = system.l2bus.slave
system.dcache.mem_side = system.l2bus.slave
```



Requests & Packets

- Protocol stack based on Requests and Packets
  - Uniform across all MemObjects (with the exception of Ruby)
  - Aimed at modelling general memory-mapped interconnects
  - A master module, e.g. a CPU, changes the state of a slave module, e.g. a memory through a Request transported between master ports and slave ports using Packets

```
Request req(addr, size, flags, masterId);
Packet* req_pkt = new Packet(req, MemCmd::ReadReq);
...
```




```
if (req_pkt->needsResponse()) {
   req_pkt->makeResponse();
} else {
   delete req_pkt;
}
...
```

```
...
delete resp_pkt;
```

Requests & Packets

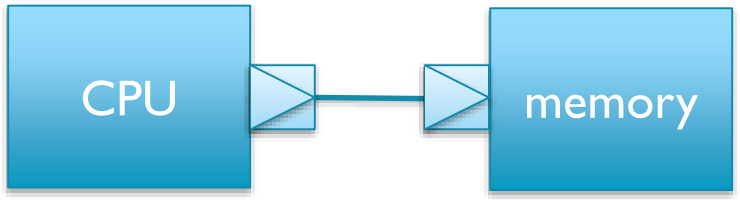
- Requests contain information persistent throughout a transaction
  - Virtual/physical addresses, size
  - MasterID uniquely identifying the module initiating the request
  - Stats/debug info: PC, CPU, and thread ID
- Requests are transported as Packets
  - Command (ReadReq, WriteReq, ReadResp, etc.) (MemCmd)
  - Address/size (may differ from request, e.g., block aligned cache miss)
  - Pointer to request and pointer to data (if any)
  - Source & destination port identifiers (relative to interconnect)
    - Used for routing responses back to the master
    - Always follow the same path
  - SenderState opaque pointer
    - Enables adding arbitrary information along packet path

Functional transport interface

- On a master port we send a request packet using sendFunctional
- This in turn calls recvFunctional on the connected slave port
- For a specific slave port we implement the desired functionality by overloading recvFunctional
  - Typically check internal (packet) buffers against request packet
  - For a slave module, turn the request into a response (without altering state)
  - For an interconnect module, forward the request through the appropriate master port using sendFunctional
    - Potentially after performing snoops by issuing sendFunctionalSnoop

```
masterPort.sendFunctional(pkt);
// packet is now a response
```

```
MySlavePort::recvFunctional(PacketPtr pkt)
{
   ...
```



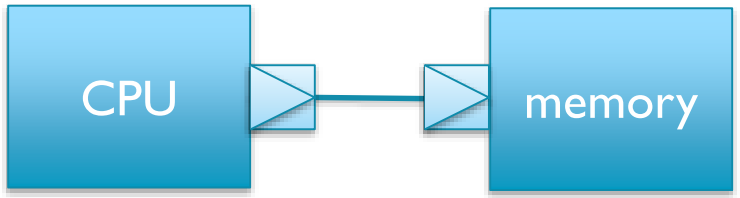
Atomic transport interface

- On a master port we send a request packet using sendAtomic
- This in turn calls recvAtomic on the connected slave port
- For a specific slave port we implement the desired functionality by overloading recvAtomic
  - For a slave module, perform any state updates and turn the request into a response
  - For an interconnect module, perform any state updates and forward the request through the appropriate master port using sendAtomic
    - Potentially after performing snoops by issuing sendAtomicSnoop
  - Return an approximate latency

```
Tick latency = masterPort.sendAtomic(pkt);
// packet is now a response
```

→

```
MySlavePort::recvAtomic(PacketPtr pkt)
{
   ...
   return latency;
}
```



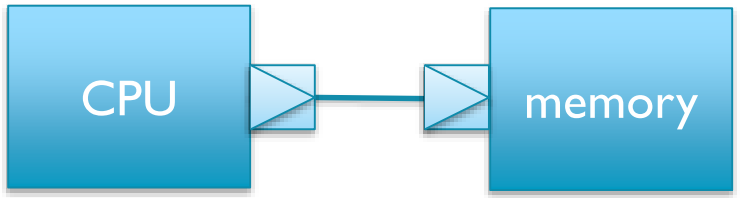
Timing transport interface

- On a master port we try to send a request packet using sendTimingReq
- This in turn calls recvTiming on the connected slave port
- For a specific slave port we implement the desired functionality by overloading recvTimingReq
  - Perform state updates and potentially forward request packet
  - For a slave module, typically schedule an action to send a response at a later time
- A slave port can choose not to accept a request packet by returning false
  - The slave port later has to call sendRetryReq to alert the master port to try again

```
bool success = masterPort.sendTimingReq(pkt);
if (success) {
   // request packet is sent
   ...
} else {
   // failed, wait for recvReqRetry from slave port
   ...
}
```

→

```
MySlavePort::recvTimingReq(PacketPtr pkt)
{
   assert(pkt->isRequest());
   ...
   return true/false;
}
```

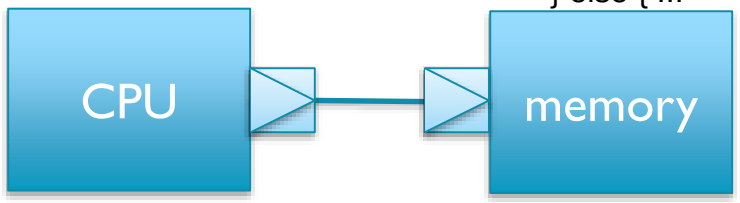


Timing transport interface (cont'd)

- Responses follow a symmetric pattern in the opposite direction
- On a slave port we try to send a response packet using sendTiming
- This in turn calls recvTiming on the connected master port
- For a specific master port we implement the desired functionality by overloading recvTiming
  - Perform state updates and potentially forward response packet
  - For a master module, typically schedule a succeeding request
- A master port can choose not to accept a response packet by returning false
  - The master port later has to call sendRetryResp to alert the slave port to try again

```
MyMasterPort::recvTimingResp(PacketPtr pkt)
{
   assert(pkt->isResponse());
   ...
   return true/false;
}
```

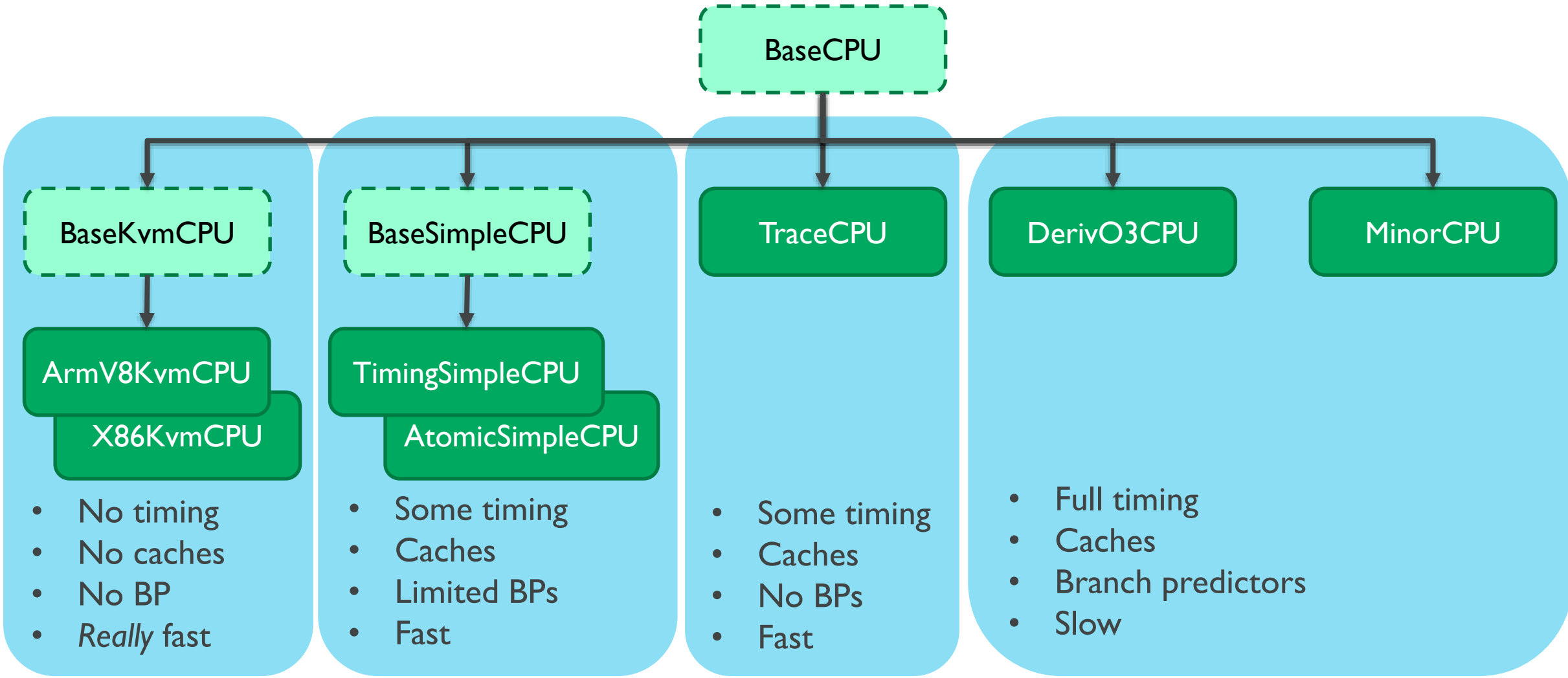
```
bool success = slavePort.sendTimingResp(pkt);
if (success) {
   // response packet is sent
   ...
} else { ...
```



CPU Models

Andreas Sandberg

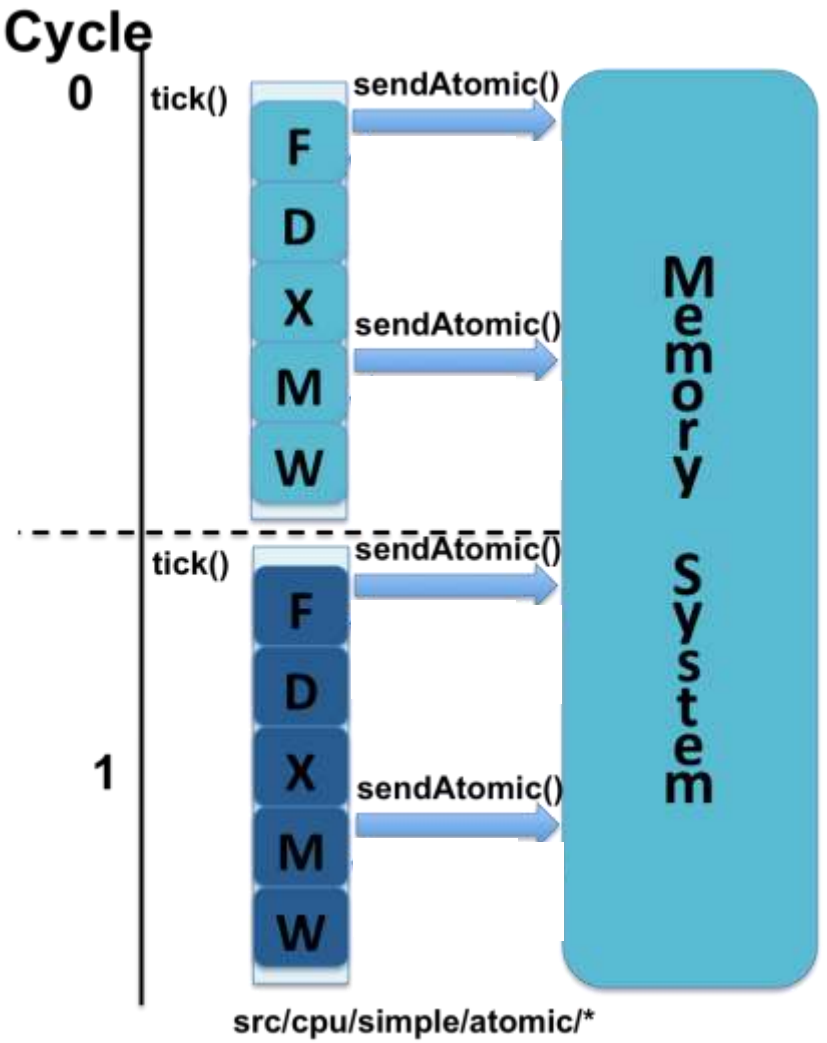
CPU models overview



ARM

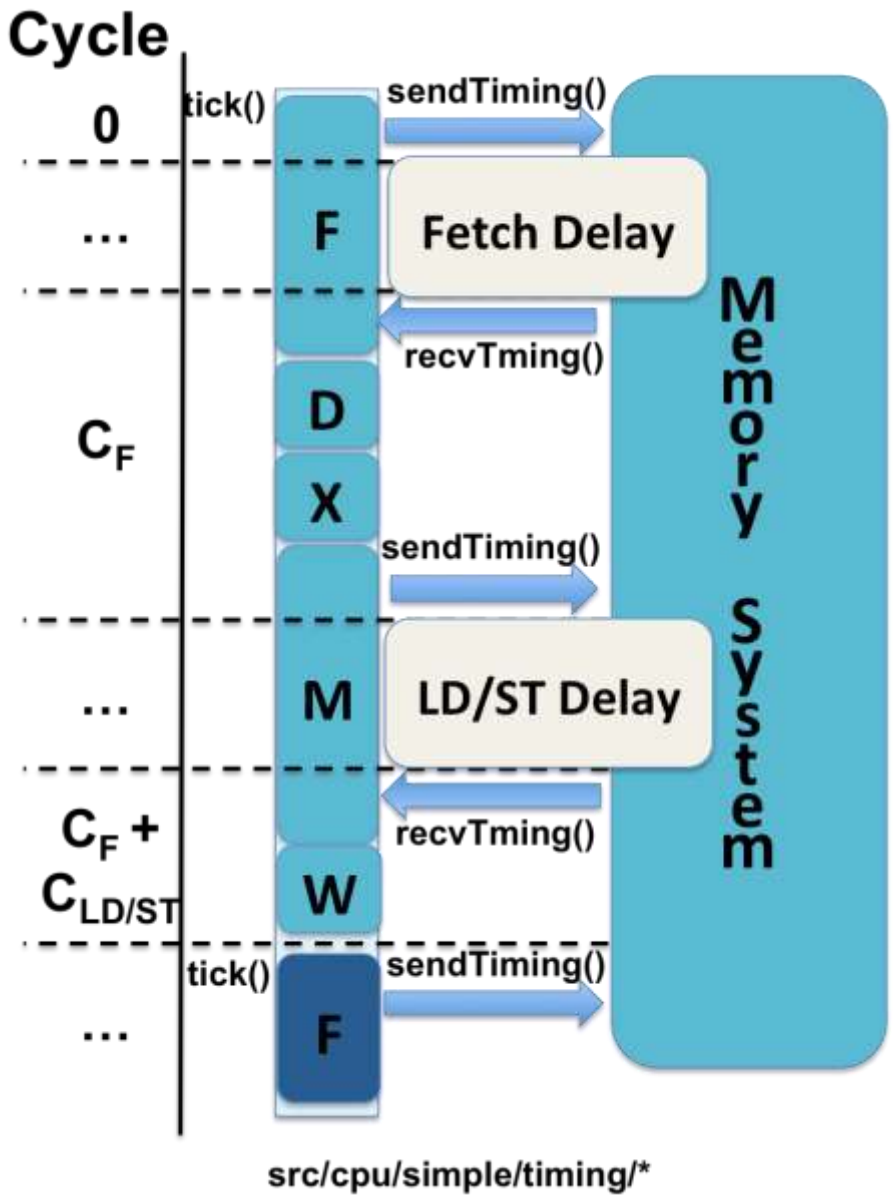
Atomic Simple CPU

- On every CPU tick() perform all operations for an instruction
- Memory accesses use atomic methods
- Fastest functional simulation
  - Except for KVM-accelerated CPUs



Timing Simple CPU

- Memory accesses use timing path
- CPU waits until memory access returns
- Fast, provides some level of timing



Detailed CPU Models

- Parameterizable pipeline models w/SMT support
- Two Types
  - MinorCPU – Parameterizable in-order pipeline model
  - O3CPU – Parameterizable out-of-order pipeline model
- "Execute in Execute", detailed modeling
  - Roughly an order-of-magnitude slower than Simple
  - Models the timing for each pipeline stage
  - Forces both timing and execution of simulation to be accurate
  - Important for Coherence, I/O, Multiprocessor Studies, etc

In-Order CPU Model

- Models a “standard” 4-stage pipeline
  - Fetch1, Fetch2, Decode, Execute
- Key Resources
  - Cache, Execution, BranchPredictor, etc.
  - Pipeline stages

Out-of-Order (O3) CPU Model

- Defaults to a 7-stage pipeline
  - Fetch, Decode, Rename, Issue, Execute, Writeback, Commit
  - Model varying amount of stages by changing the delay between them
    - For example: fetchToDecodeDelay
- Key Resources
  - Physical Registers, IQ, LSQ, ROB, Functional Units

Important CPU interfaces

- BaseCPU
  - Base class for all CPU models
  - Provides a common interface for checkpointing/switching/interrupts/…
  - Even used by KVM-based CPUs
- ThreadContext
  - Interface for accessing total architectural state of a single thread (PC, registers, etc.)
  - Holds pointers to important structures (TLB, CPU, etc.)
  - CPU models typically implement custom versions or use SimpleThread
- ExecContext
  - Abstract interface defining how an instruction interface with the CPU model

StaticInst

- Represents a decoded instruction
  - Has classifications of the inst
  - Corresponds to the binary machine inst
  - Only has static information

- Has all the methods needed to execute an instruction
  - Tells which regs are source and dest
  - Contains the execute() function
  - ISA parser generates execute() for all insts

DynInst

- Complex CPU models need to track resources used by instructions
- Dynamic version of StaticInst
  - Used to hold extra information for in-flight instructions
  - Holds PC, Results, Branch Prediction Status
  - Interface for TLB translations
- Specialized versions for detailed CPU models

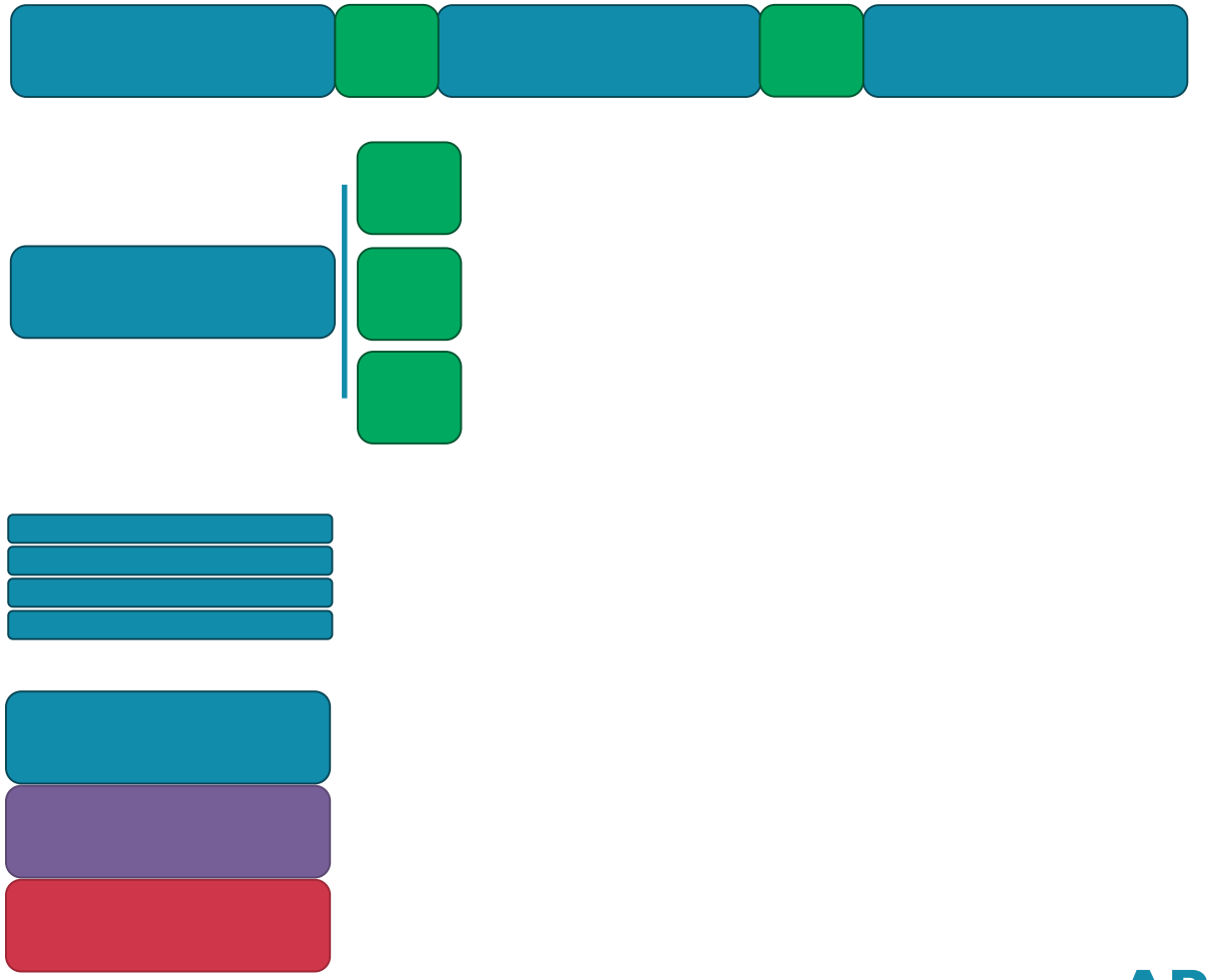
Examples

- Virtualization-based CPU: `BaseKvmCPU`
  - See: `src/cpu/kvm/base.{cc,hh}` and `src/cpu/kvm/BaseKvmCPU.py`
  - Implements the basic interfaces required by all CPU model
  - Reasonably small and well documented
  - Does *not* simulate instructions or implement `ExecContext`
- Simplest possible simulated CPU: `AtomicSimpleCPU`
  - See: `src/cpu/simple/{base.cc,base.hh,atomic.cc,atomic.hh, AtomicSimpleCPU.py}`
  - Minimal simulated CPU that includes SMT
- Simplest "real" model: `MinorCPU`
  - See `src/cpu/minor/*`
  - Implements a pipelined in-order CPU

Advanced Features & Capabilities

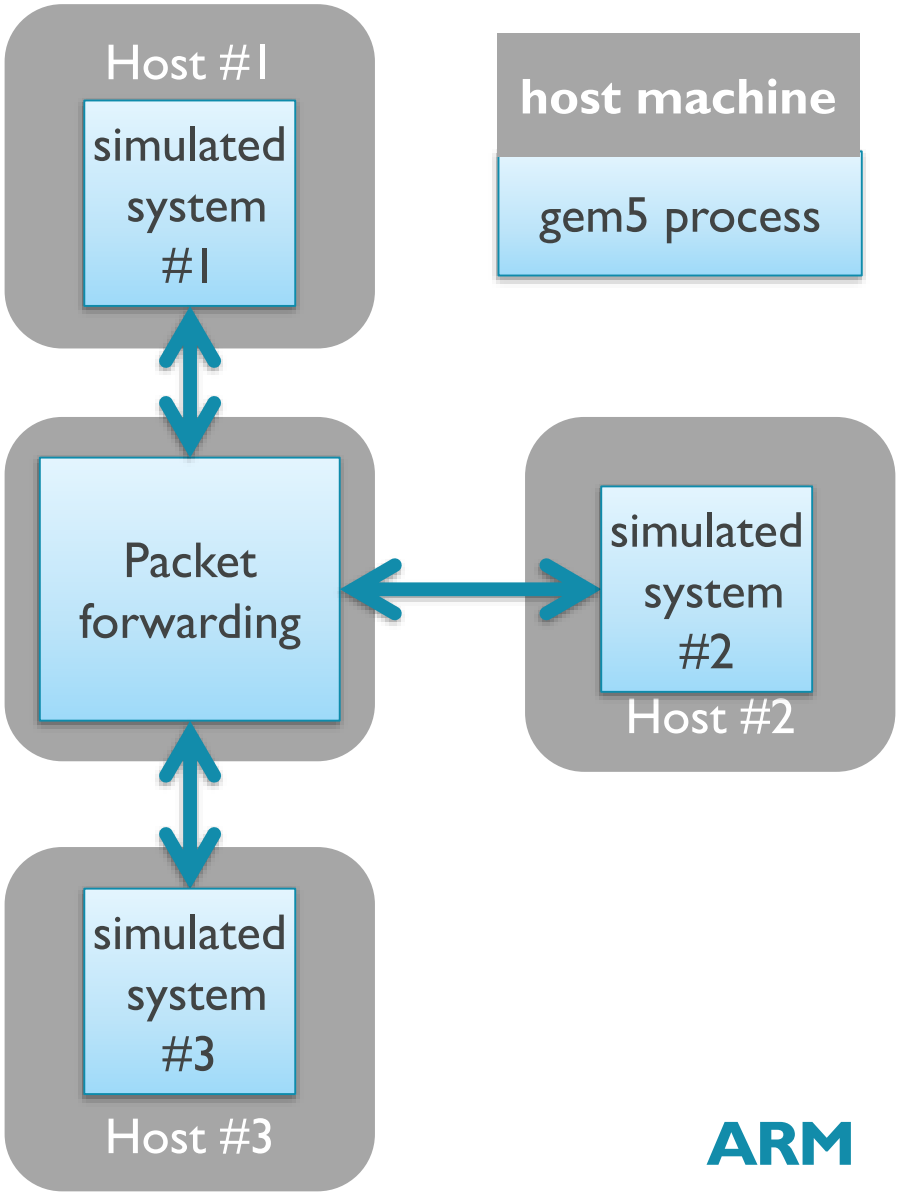
Accelerating gem5

- Switching modes
  - (kvm +) functional + timing / detailed
- Checkpoints
  - boot Linux -> checkpoint
  - run multiple configurations in parallel
  - run multiple checkpoints in parallel
- Multi-threading
  - multiple queues
  - multiple workers execute events
  - data sharing and tight coupling limits speedup
- Multi-processed gem5
  - for design space explorations



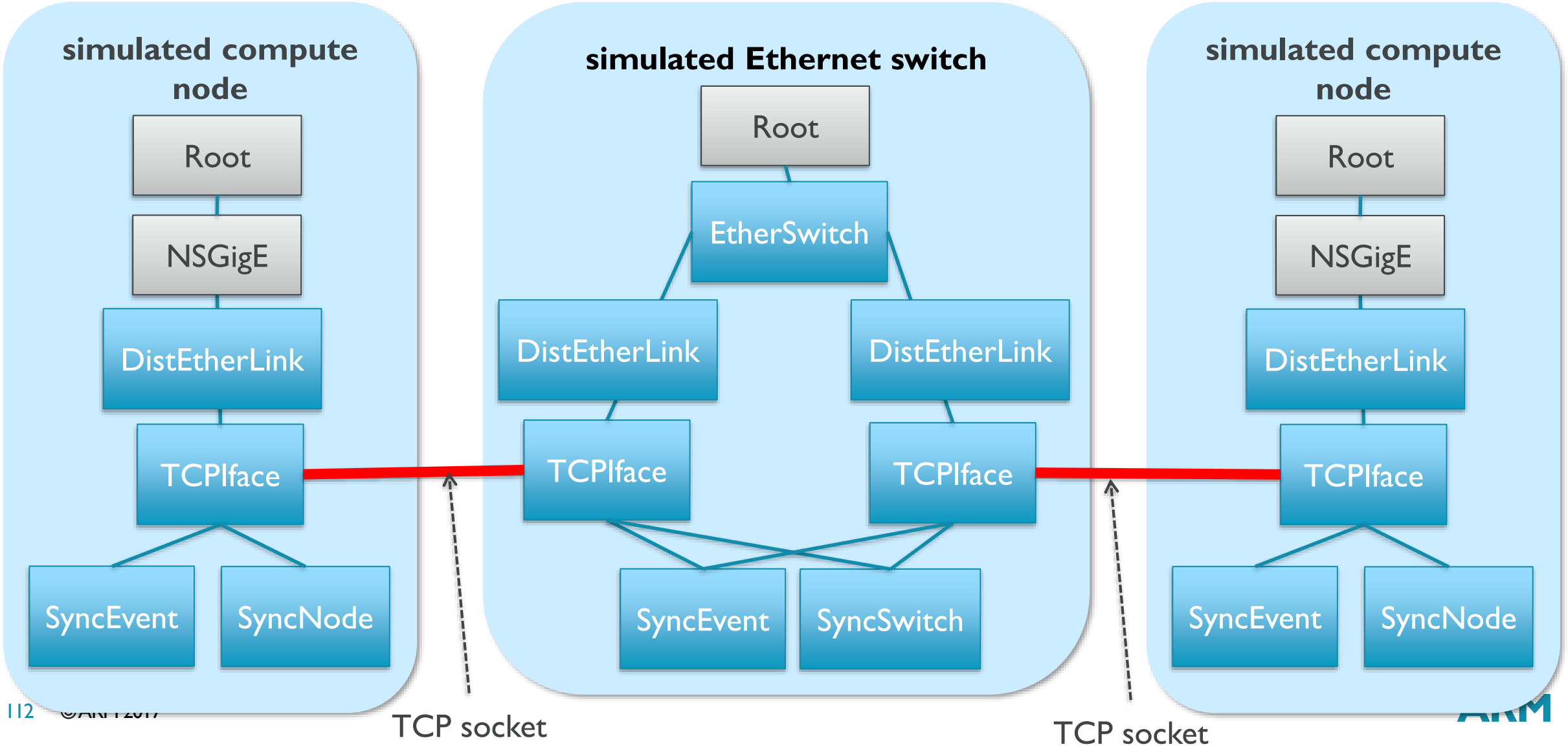
Distributed gem5 simulation

- gem5 running in parallel on a cluster of host machines
- Packet forwarding engine
  - Forward packets among the simulated systems
  - Synchronize the distributed simulation
  - Simulate network topology
- Tested with ~30 nodes, 100s planned



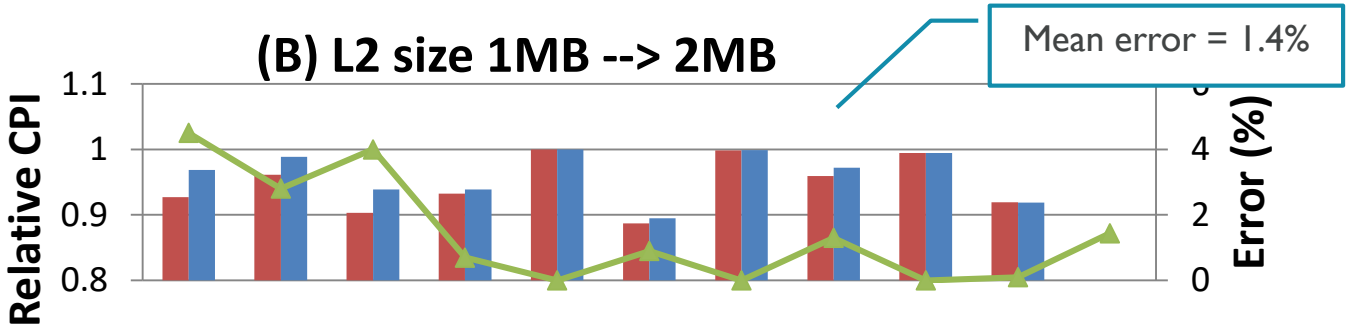
ARM

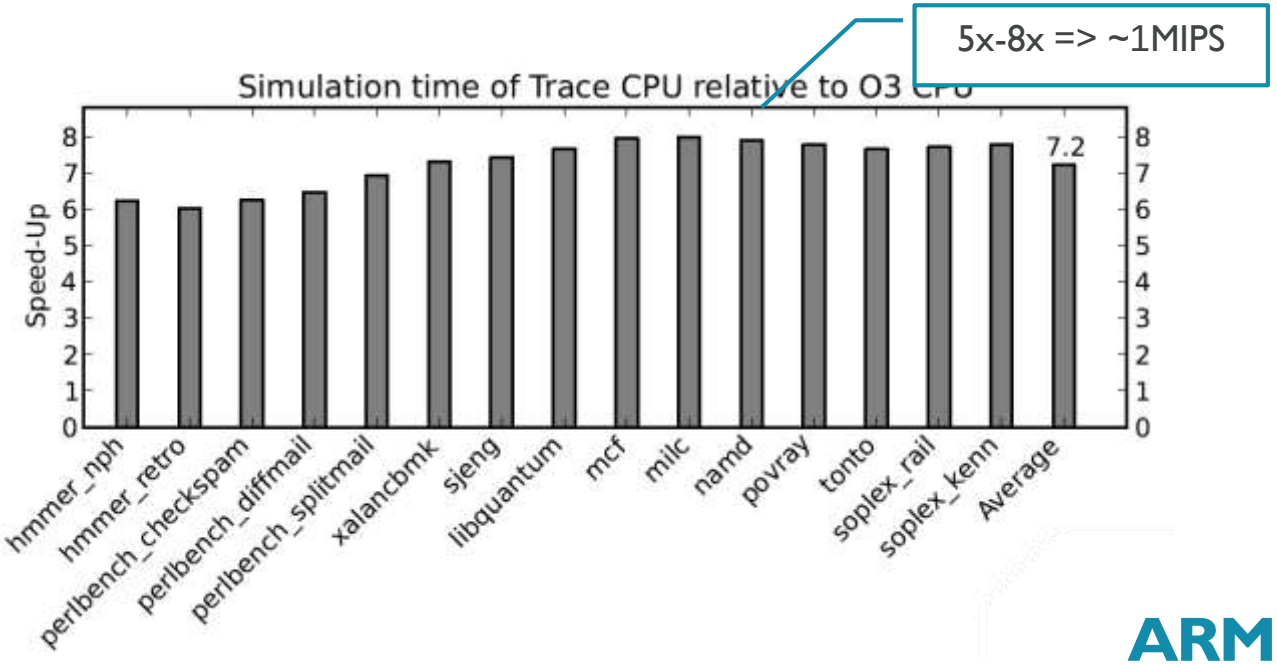
Object Diagram : Simulating a 2-node Cluster Example



Elastic Traces – fast, realistic memory exploration

- High-level OOO core model → speedy simulation
  - Capture data dependencies and MLP
  - Elastic replay
- High-level synchronisation event capture
  - Predict scalability for SMPs
  - Additional 10x speedup





ARM

Data Profiling and Heterogeneous Memory

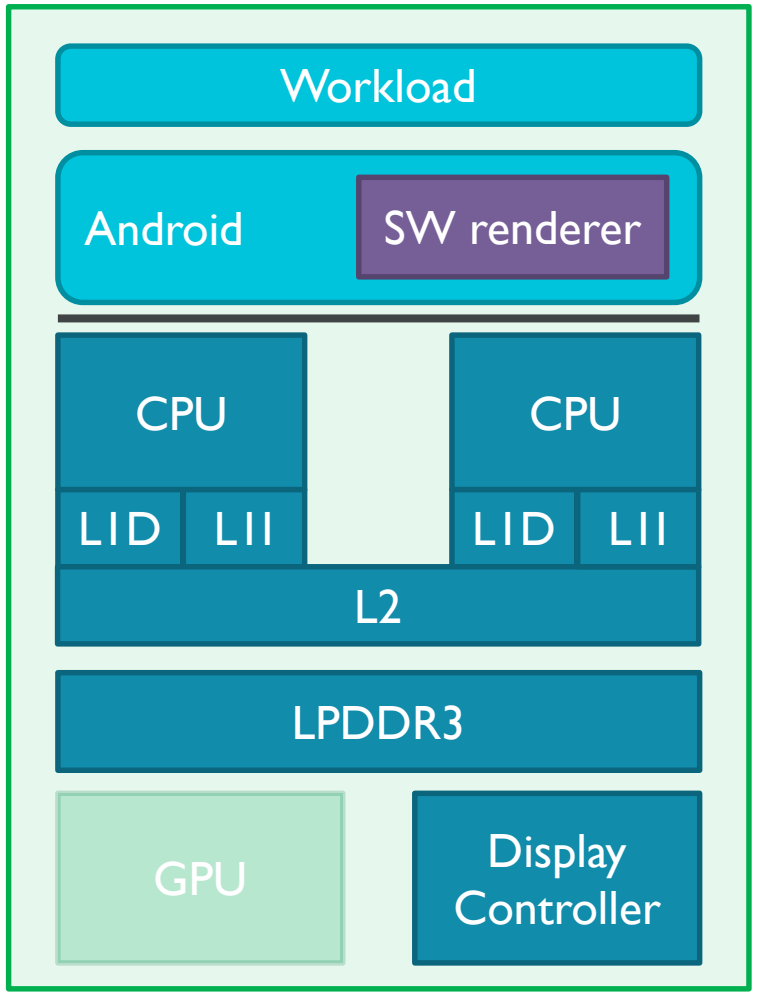
- Address rising cost of communication
- Optimize data structures to improve cache utilization and efficiency
- Optimize data storage onto heterogeneous memories

Data variable (Size) (filename : linenum)	Accesses	Read (	L1$	Cs	Cf	Cc	Ts	Fs)	Write (	L1$	Cs	Cf	Cc	Ts	Fs)
fib.n (4) (fibonacci.c : 18)	1306	1241 (	6,	0,	0,	6,	0,	0)	65 (	24,	22,	0,	2,	0,	0)
fib.dict (8) (fibonacci.c : 23)	508	445 (	0,	0,	0,	0,	0,	0)	63 (	0,	0,	0,	0,	0,	0)
fib.f (8) (fibonacci.c : 27)	389	227 (	0,	0,	0,	0,	0,	0)	162 (	12,	0,	1,	11,	0,	0)
dictionary (8) (fibonacci.c : 14)	106	105 (	7,	0,	7,	0,	0,	0)	1 (	1,	0,	0,	1,	0,	0)
main.n (4) (fibonacci.c : 36)	8	7 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[0] (8) (fibonacci.c : 14)	7	4 (	0,	0,	0,	0,	0,	0)	3 (	0,	0,	0,	0,	0,	0)
dictionary[1] (8) (fibonacci.c : 14)	6	6 (	0,	0,	0,	0,	0,	0)	0 (	0,	0,	0,	0,	0,	0)
main.f (8) (fibonacci.c : 42)	5	2 (	0,	0,	0,	0,	0,	0)	3 (	0,	0,	0,	0,	0,	0)
dictionary[8] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[6] (8) (fibonacci.c : 14)	3	2 (	1,	0,	0,	1,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[64] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[62] (8) (fibonacci.c : 14)	3	2 (	1,	1,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[60] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[58] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[56] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[54] (8) (fibonacci.c : 14)	3	2 (	1,	1,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[52] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[50] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[4] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[48] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[46] (8) (fibonacci.c : 14)	3	2 (	1,	1,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[44] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[42] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[40] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[38] (8) (fibonacci.c : 14)	3	2 (	2,	1,	1,	0,	0,	0)	1 (	1,	0,	1,	0,	0,	0)
dictionary[36] (8) (fibonacci.c : 14)	3	2 (	1,	0,	1,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)
dictionary[34] (8) (fibonacci.c : 14)	3	2 (	0,	0,	0,	0,	0,	0)	1 (	0,	0,	0,	0,	0,	0)

Graphics & Android Andreas

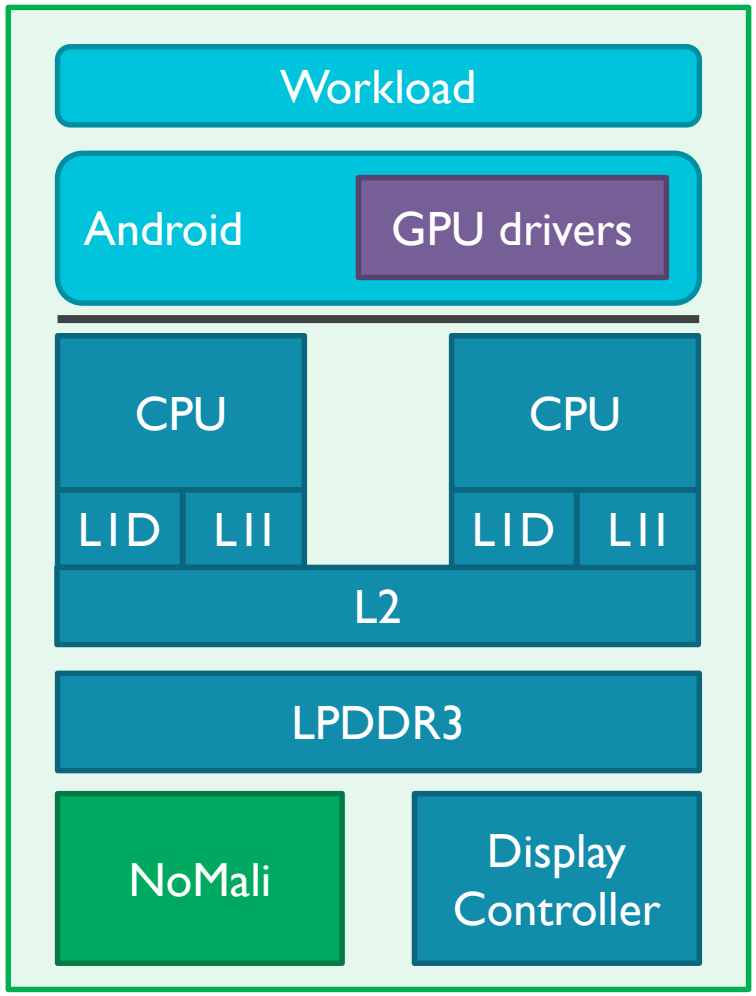
ARM

Common Approach: CPU-Centric



- Software renderer instead of a real GPU
  - Optimization friendly code
  - Can be vectorized
  - Easy-to-predict branches
  - Large memory foot print
- Doesn't simulate the driver
  - Known to be the bottleneck for some workloads
  - Horrible code
- Workload and software renderer compete for resources
  - Can significantly skew core behavior
- Affects 2D applications *and* 3D applications

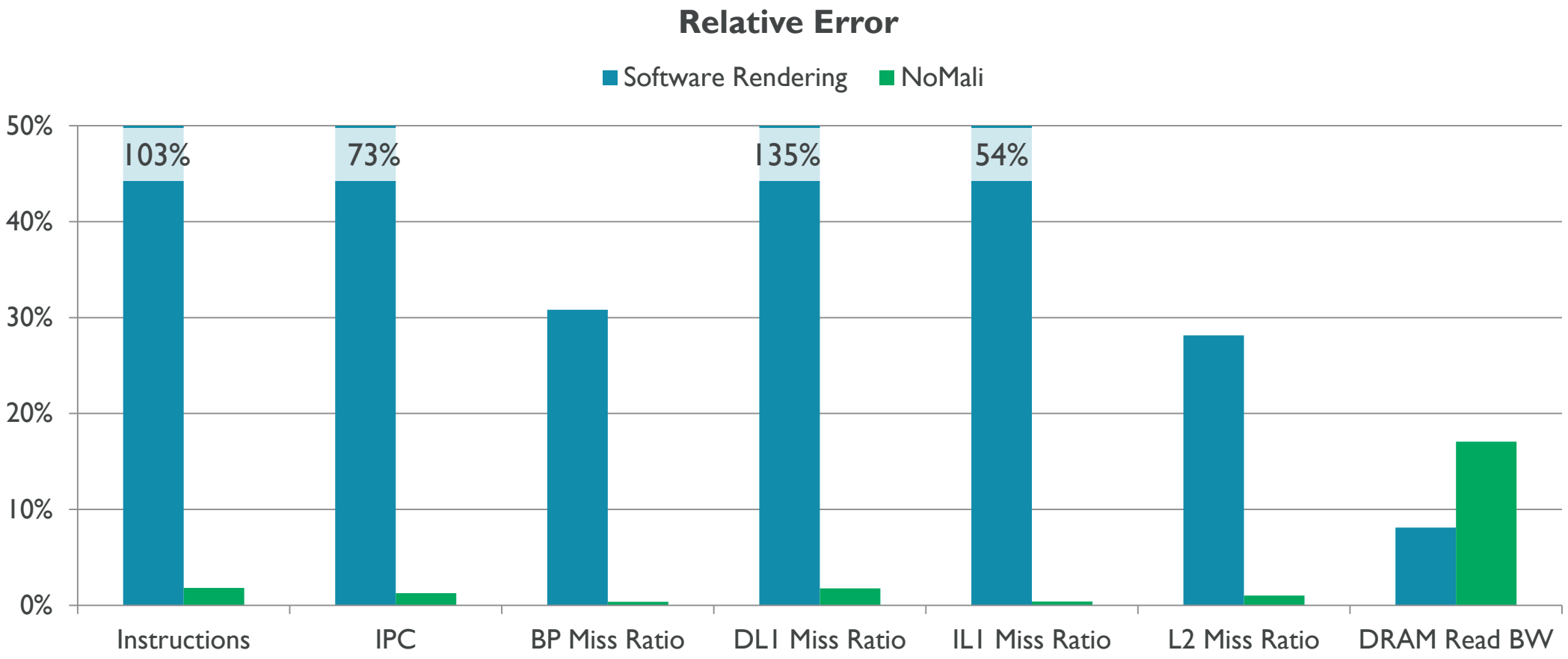
Full system NoMali modelling



- Passes the duck test (almost)
  - Most GPU integration tests work (no pixels)
  - Implements the Mali register interface & interrupts
  - Accurate CPU+GPU interactions
- Runs the full driver stack
  - Complex software with significant CPU component
- Limitations:
  - Doesn't produce any display output
  - No memory system interactions
  - Requires a properly optimized driver stack
- Use cases:
  - CPU-centric studies (driver performance)
  - Fast-forward (boot / long traces)

De Jong, Rene, and Andreas Sandberg. "NoMali: Simulating a Realistic Graphics Driver Stack Using a Stub GPU." *ISPASS 2016*

Why do you care?



bbench on Android K (real GPU as reference)

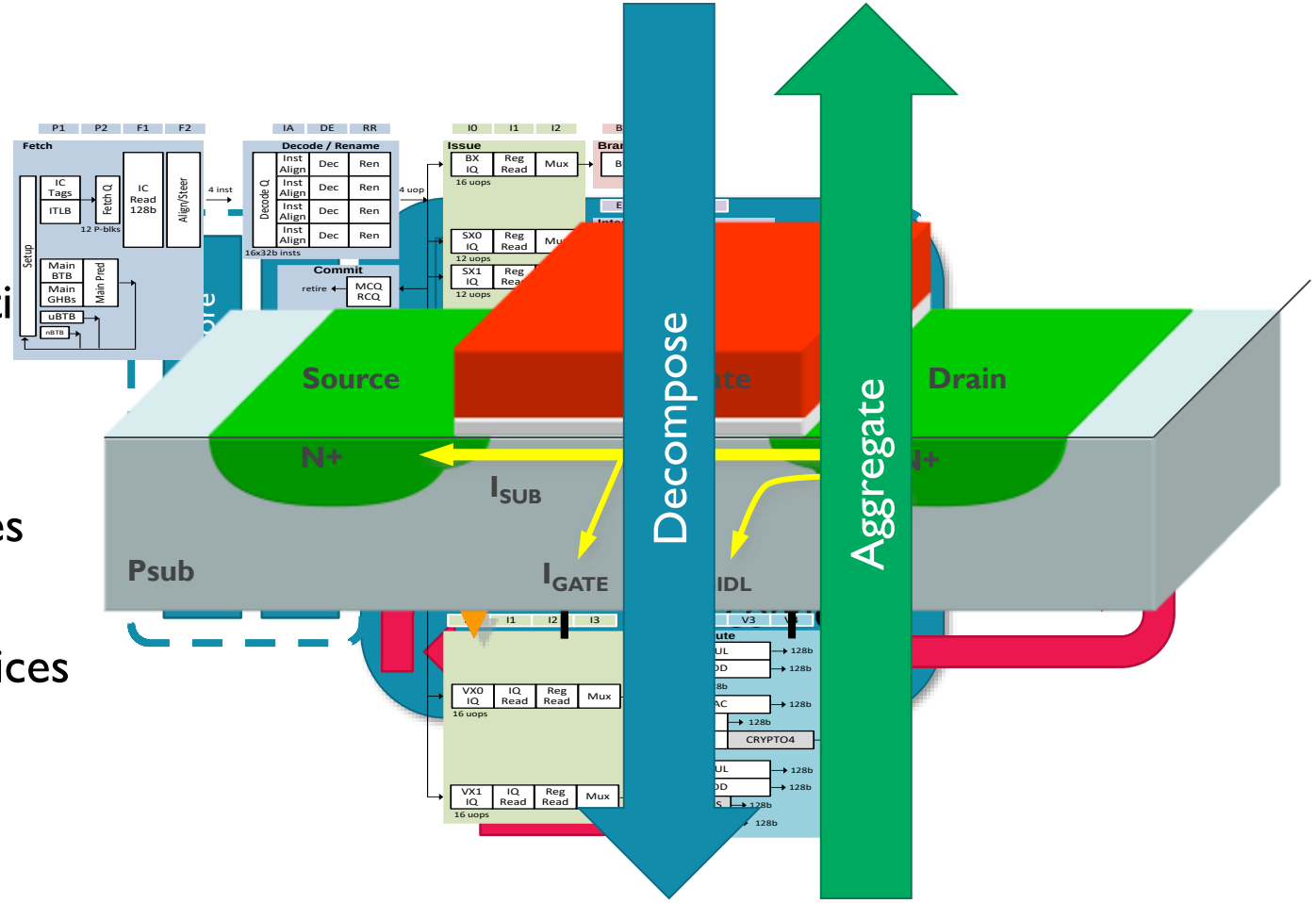
ARM

Power Modelling Stephan

ARM

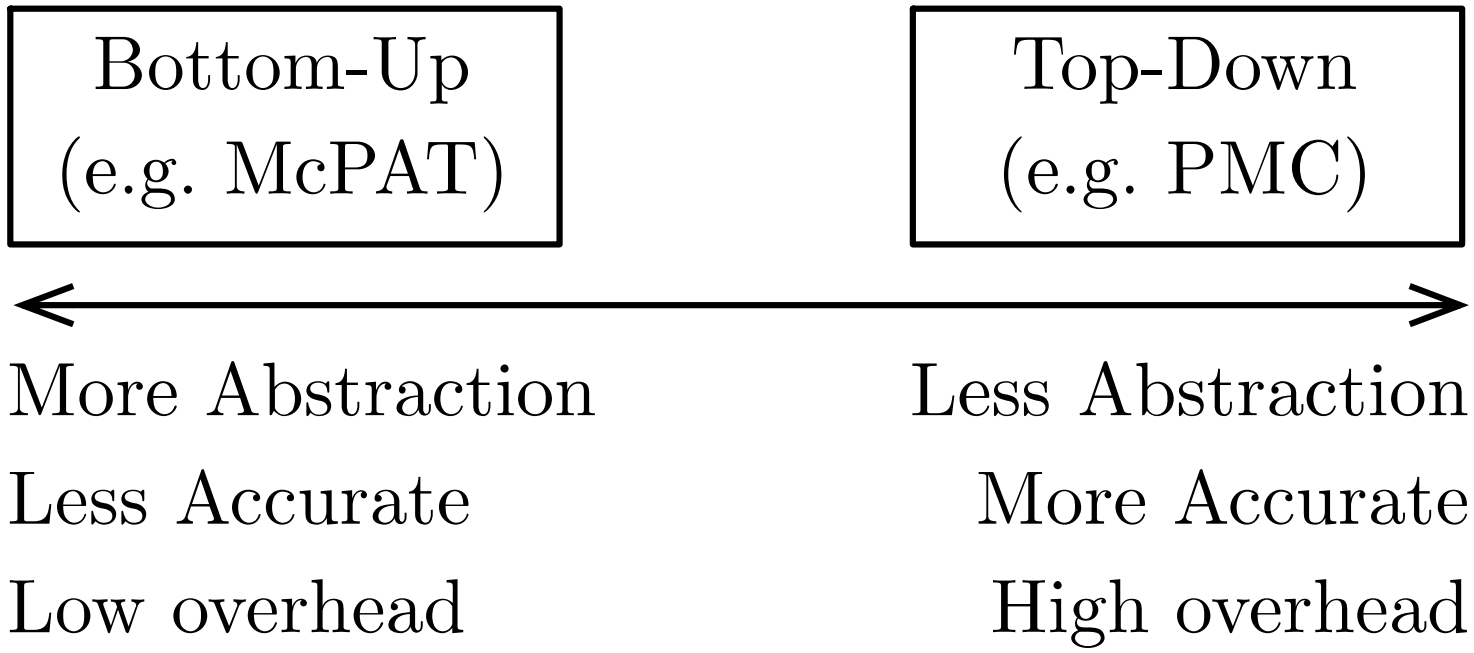
Power Models

- bottom-up
  - simulate gates
  - toggle rates
  - complex aggregati
- top-down
  - high level activities
  - few voltage rails
  - measure real devices



ARM

Top Down vs. Bottom Up



Top-down also has uses in design-space exploration – accurate reference

Top Down Power Models

- Built experimentally
- Often uses regression
- Extremely accurate
- Inflexible, often tied to a specific platform

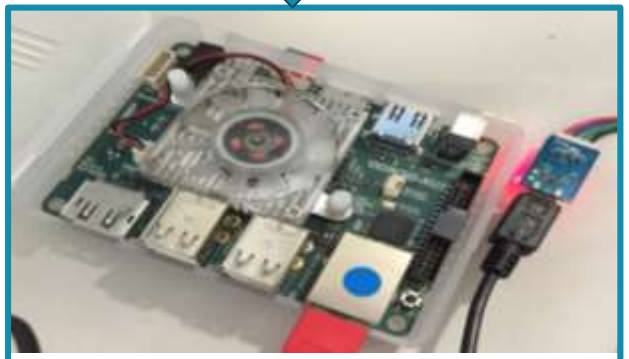
Bottom Up Power Models

- Built on theory
  - E.g. McPAT – Power Area and Timing Multi- and Many- core modelling framework
- Good for design-space exploration
- Large errors (largely due to abstraction)
- Relatively slow (not suitable for run-time management)

Power Modeling Based on Existing Hardware

1. Run: workloads
@ different DVFS level
@ different affinities

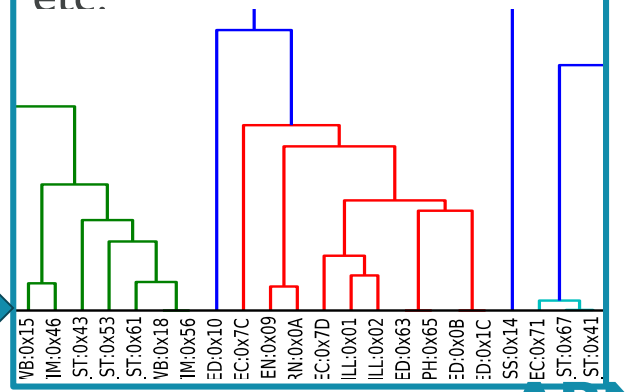
60 workloads used:
MiBench, MediaBench,
LMbench, NEON, OpenMP



ODROID-XU3
Exynos-5422
4x Cortex-A7
4x Cortex-A15

2. Record:
- Performance Counters (PMCS)
- Voltage, Power

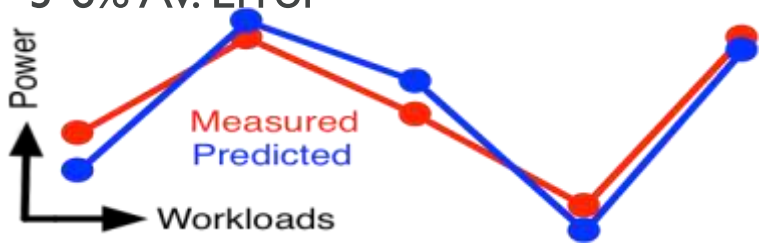
3. Choose PMCs:
Hierarchical cluster analysis, correlation matrix analysis, exhaustive search etc.



4. Build Model
- OLS multiple linear regression
- Deals with PMC multicollinearity
- Considers heteroscedasticity

$$P_{cluster} = \left(\sum_{n=0}^{N} \beta_{an} E_n V^2 f\right) + \beta_b V + \beta_c f + \beta_d$$

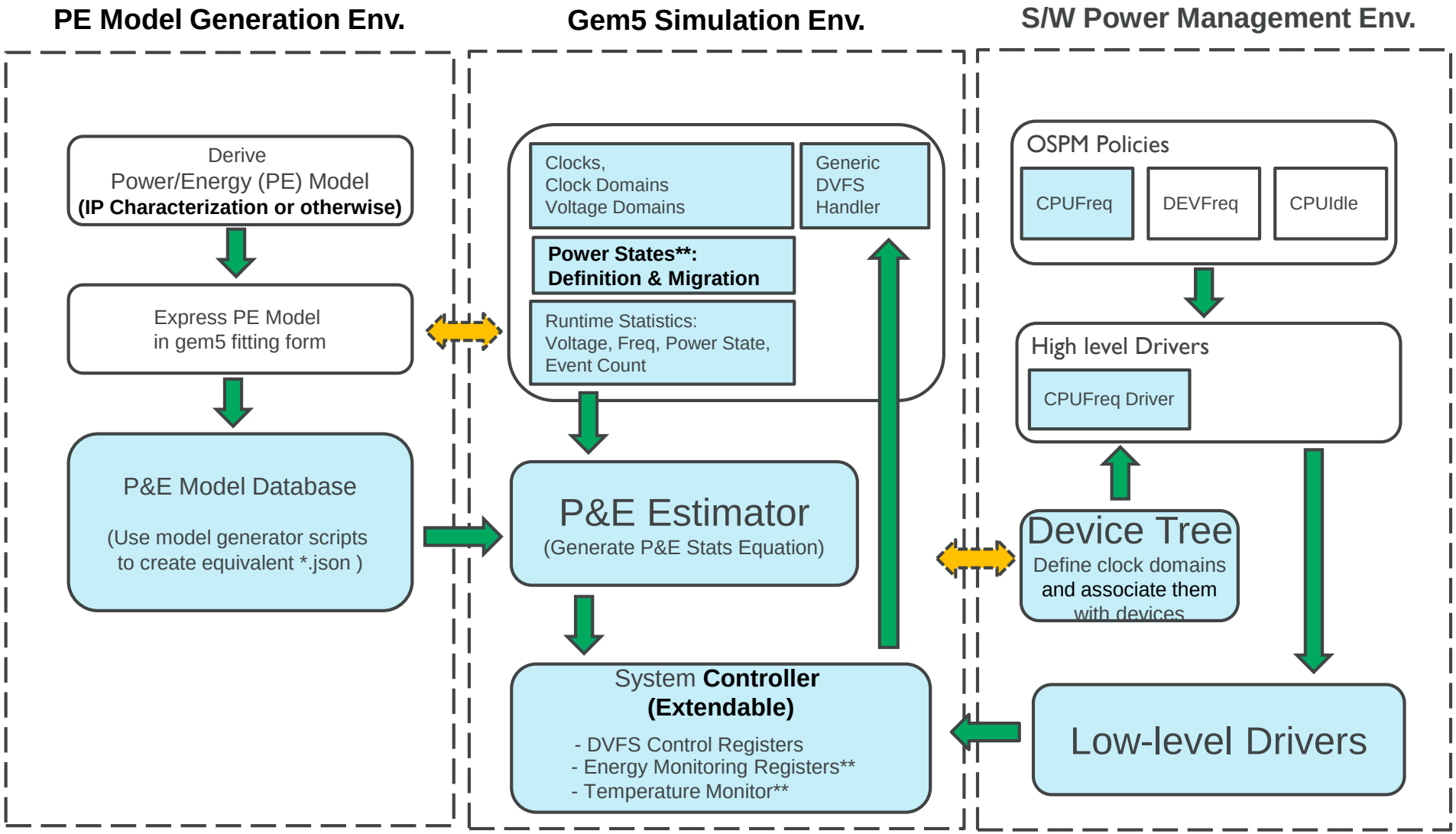
5. Validate
- K-fold cross validation
- R^2: ~0.99
- 3-6% Av. Error



6. Uses
- OS run-time management
- Reference for research
- gem5 add-on

ARM

Power&Energy Framework Overview



Ongoing activities within P&E framework

** Needs to be spec'ed out

ARM

Why are CPU power models important?

- Design space exploration
  - To see the effect of making architectural changes
- Run-time management
  - CPU employs power-saving techniques (DVFS, DPM, asymmetric multi-core e.g. ARM big.LITTLE)
  - Need accurate power estimations to make performance-power trade-off

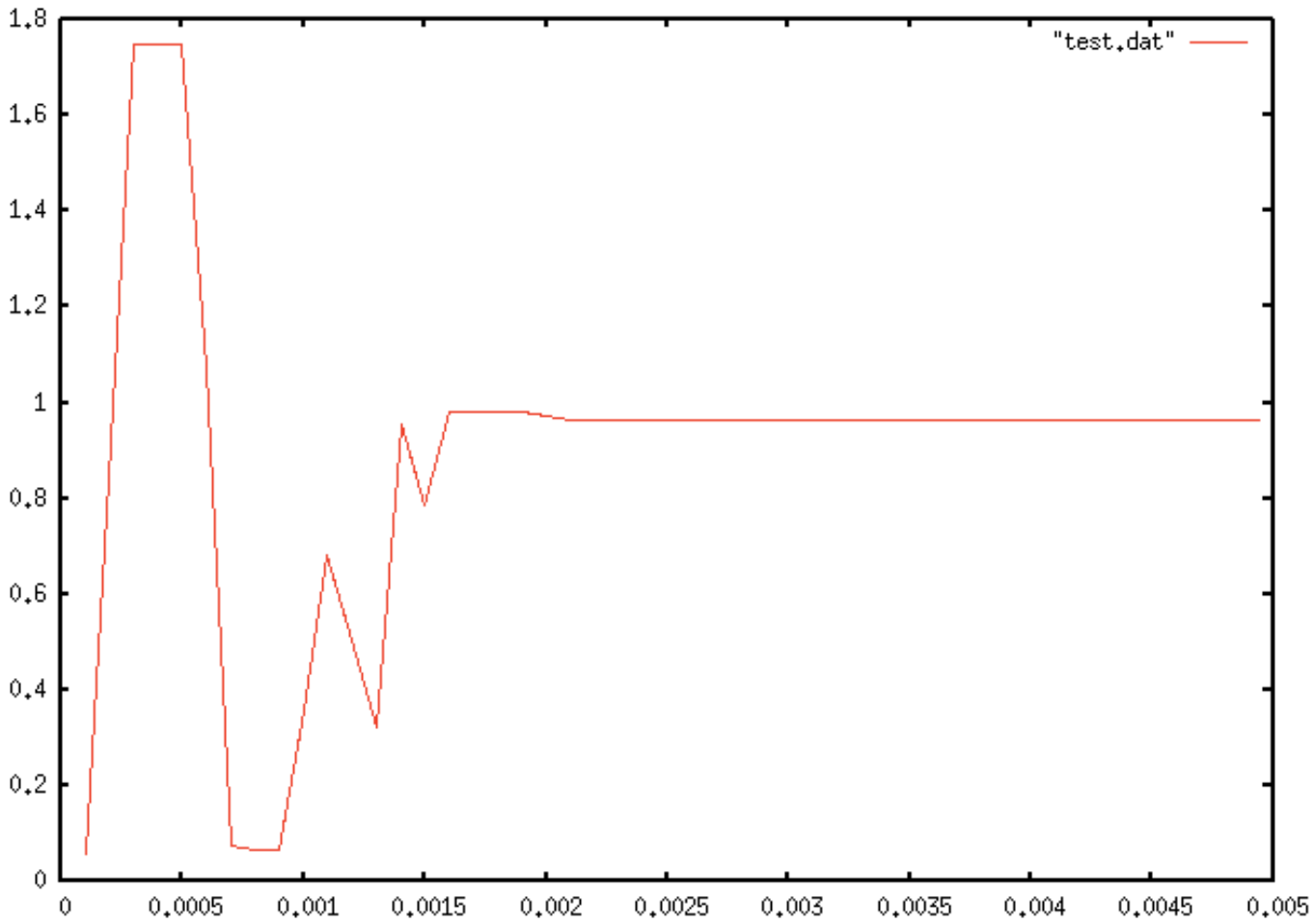
Enable Power Modelling in gem5

- `configs/example/arm/fs_power.py`
  - `dyn = "voltage * (2 * ipc + 3 * 0.000000001 * dcache.overall_misses / sim_seconds)”`
  - `st = "4 * temp"`

- `gem5.opt configs/example/arm/fs_power.py \`
  `--caches --kernel vmlinux`

- `grep pm0.dynamic_power m5out/stats.txt`
  - system.bigCluster.cpus.power_model.pm0.dynamic_power 0.057501 #Dynamic power for this object (Watts)
  - ...

And it wiggles!

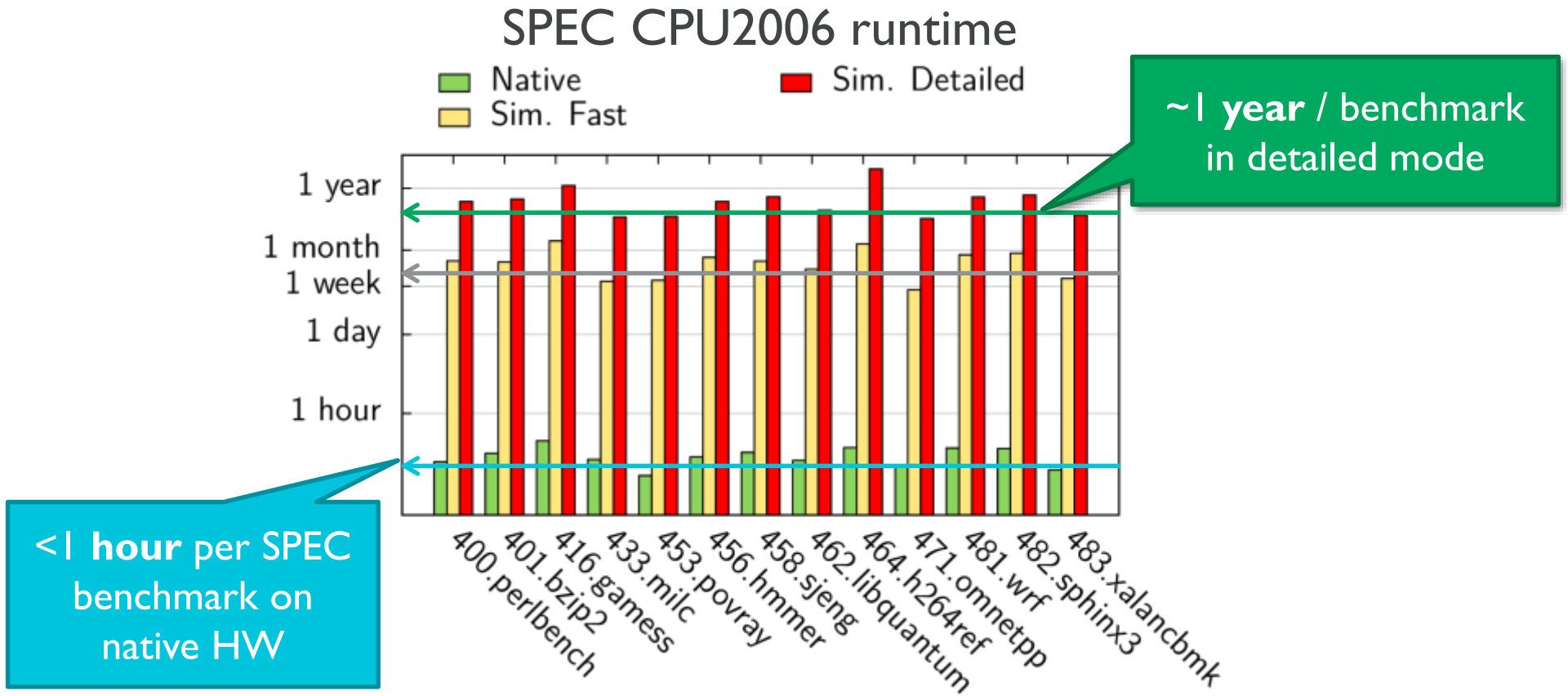


KVM

Andreas

ARM

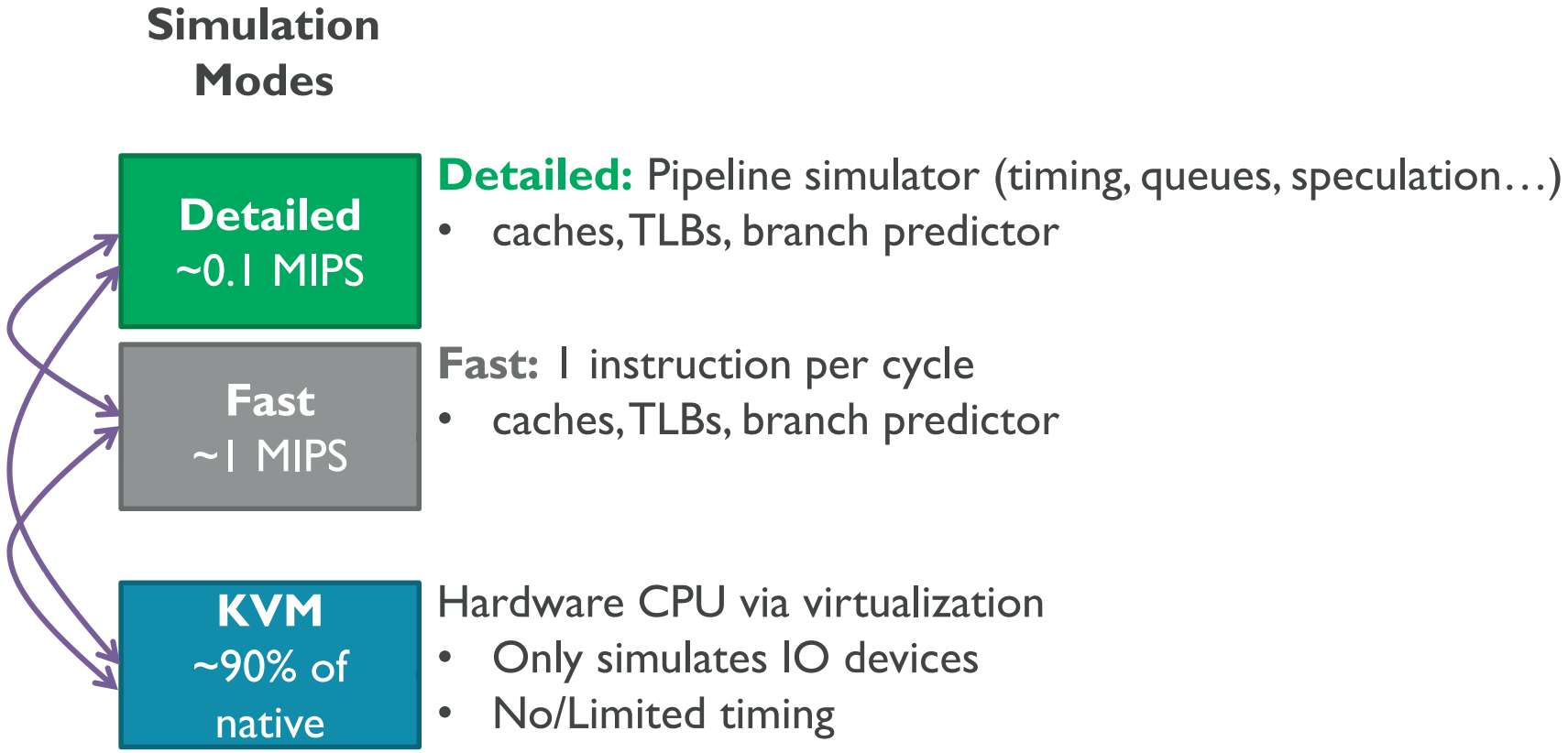
Problem: Simulation is Slow



Native	Fast	Detailed
3,000 MIPS	1 MIPS	0.1 MIPS

ARM

A KVM-Based CPU Model



Can switch between modes during simulation

Current state of KVM on ARM

- Requirements
  - Server-class ARMv8-based system
  - RAM: 4+ GiB
  - Host system and kernel with KVM support
- Known-working:
  - Running full-systems with simulated devices
  - Able to boot Android N
- Limited-support:
  - Multiple CPUs
  - Graphics, KMI
  - CPU switching
  - Checkpointing

Already in use despite known limitations

How Do I Use KVM?

- Supported by config/example/fs.py and config/example/arm/fs_bigLITTLE.py
  - Only the bL configuration supports multi-core!
- Behaves like a "normal" CPU model

```
./build/ARM/gem5.opt \
  configs/example/arm/fs_bigLITTLE.py
  --cpu-type kvm \
  --kernel vmlinux --disk my_disk.img \
  --big-cpus 1 --little-cpus 0 \
  --dtb
$GEM5/system/arm/dt/armv8_gem5_v1_1cpu.dtb
```

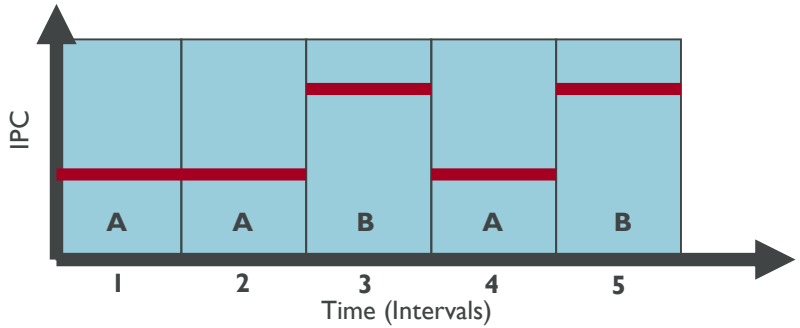
Demo

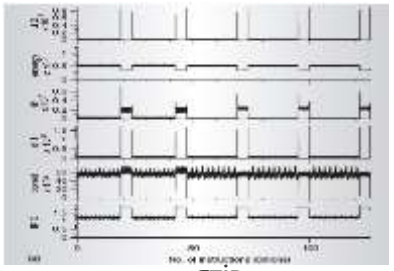
Methodology

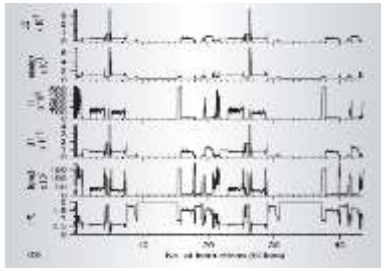
William

SimPoints

- Generate wieldable, representative slices of full benchmarks
- Terminology:
  - Intervals – slices in time, sampling granularity (e.g. 10K instructions)
  - Phases – intervals with similar behavior that often recur periodically



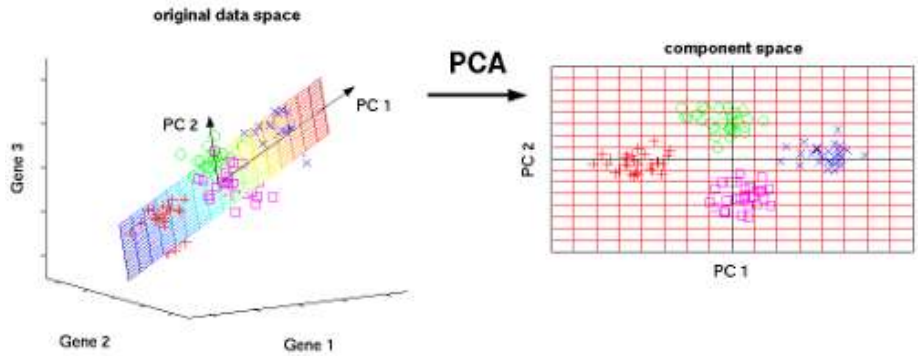

gzip


gcc

- Output from SimPoint analysis are slices and weights for each slice (choose a clustering within 5% of CPI of full run)
- Gem5 is instrumented to capture SimPoints
  - Run one time to analyze basic block vectors
  - Second time generates gem5 checkpoints at every identified phase
  - Runs can be repeated with different experimental configuration

Principal Component Analysis (PCA)

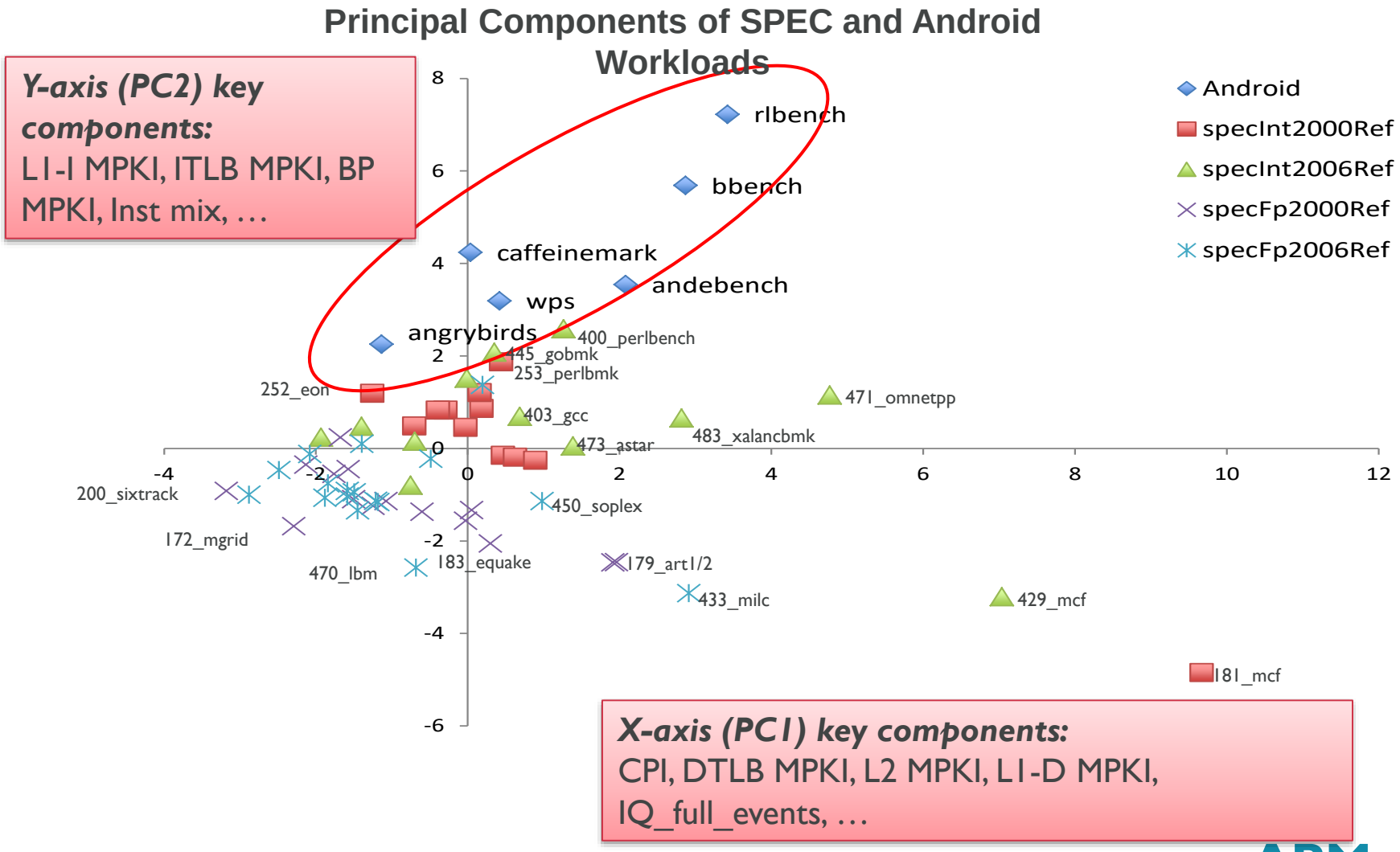
- Find the ***most important parameters*** from a large data set automatically
- How to describe "most important" using math?
  - High variance
- How do we represent our data so that the most important features can be extracted easily?
  - Change of basis
- Can infer similarities and dissimilarities of workloads
  - Based on distance on projected component space



PCA reveals the ***internal structure of the data that best explains the variance*** in the data**!**

Studying Complex Software is Important

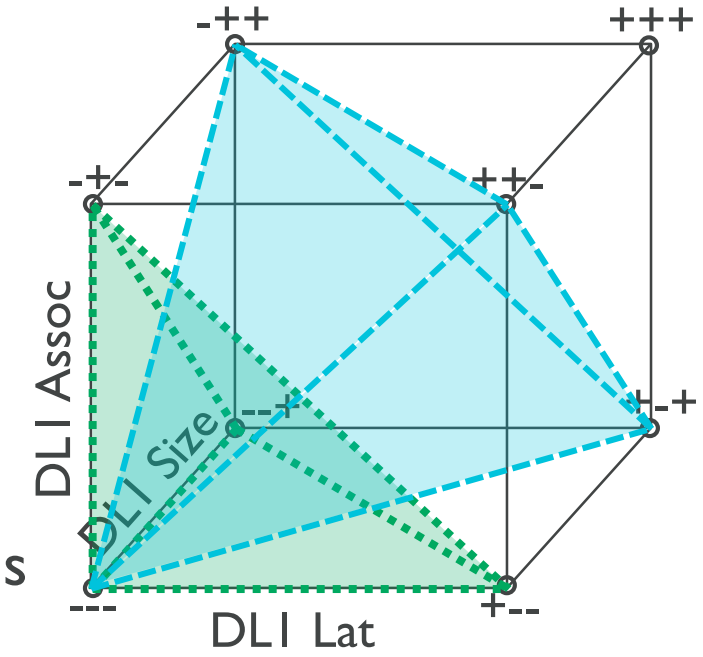
- Android workloads stress the Instruction-side aspects of a system
- The popular SPEC benchmarks primarily stress only the Data-side
  - Very limited coverage of full mobile systems' behavior



ARM

Fractional Factorial Designs

- Balanced experiment distribution
- Identify important factors
- 2^{N-M} experiments << 2^N

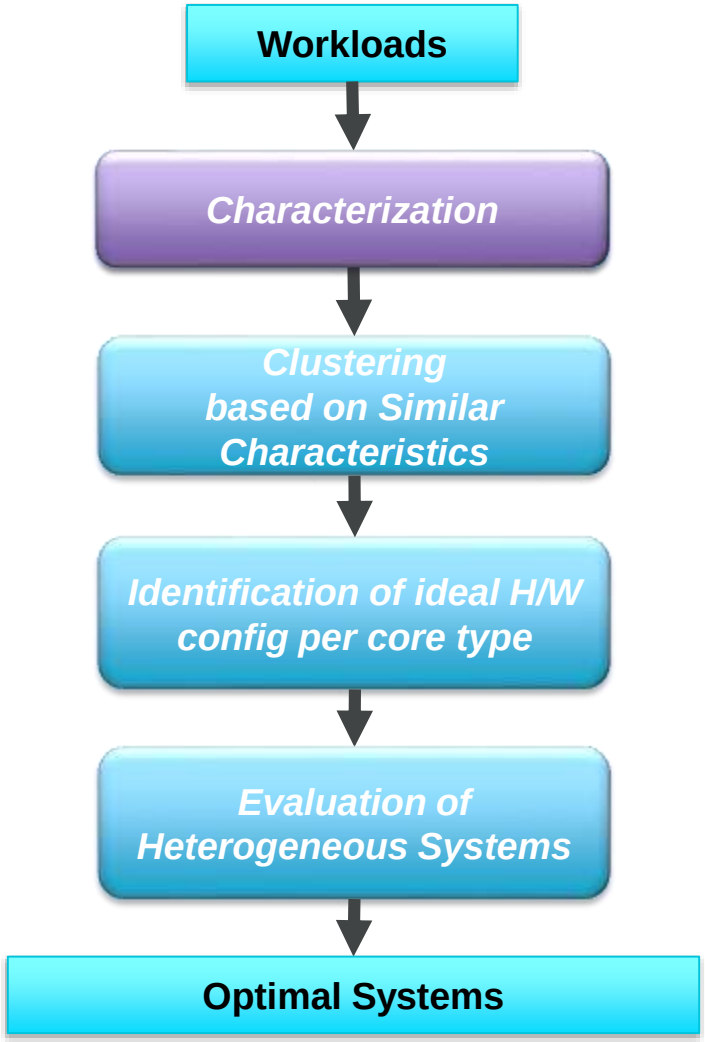


- Looks for parameters where the average '+' run is very different from '-'
  - Experiments are tolerant to noise
- Does not identify what are the best options
  - Narrows design space to what matters most

DL1 Lat	DL1 Size	DL1 Assoc
-	-	-
+	-	+
-	+	+
+	+	+

ARM

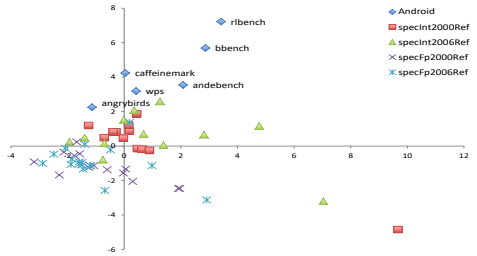
Methodology



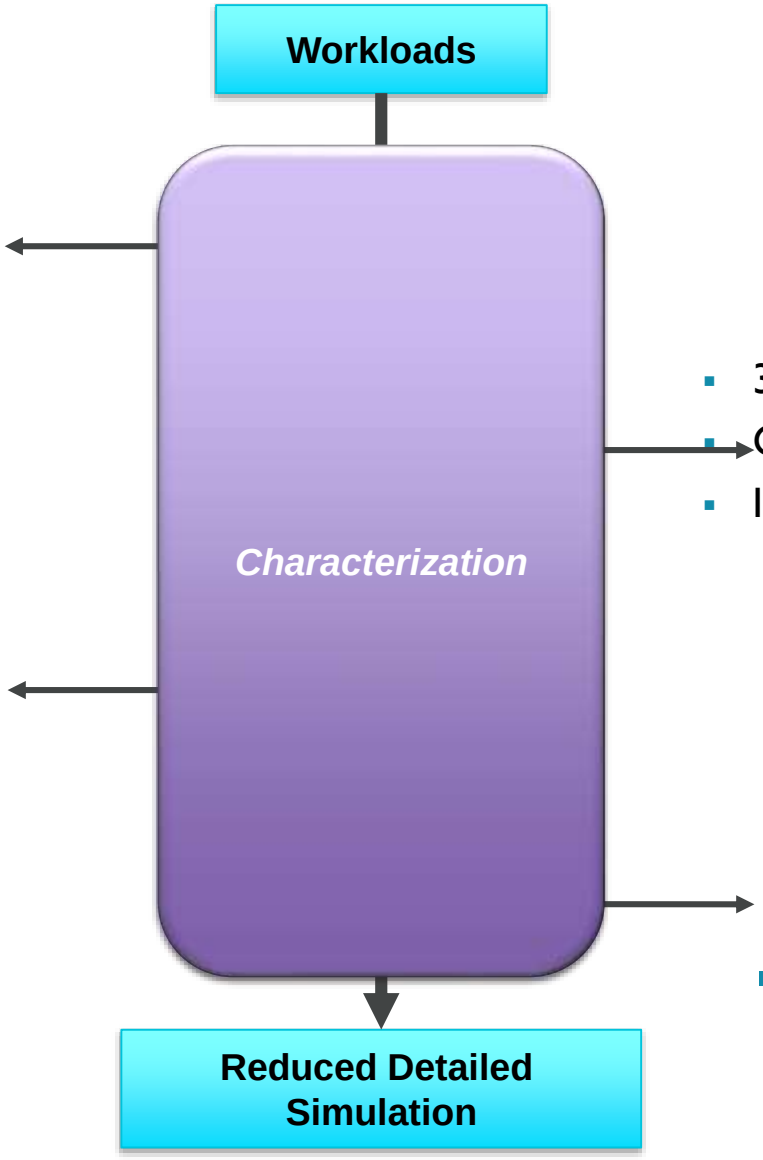
- Objective: To find the ideal heterogeneous system for a given set of workloads and hardware parameters
- Characterize and cluster workload phases
  - Cluster based on performance sensitivity to various hardware parameters
- Selectively enable or disable hardware parameters per cluster of similar workload phases to improve their efficiency

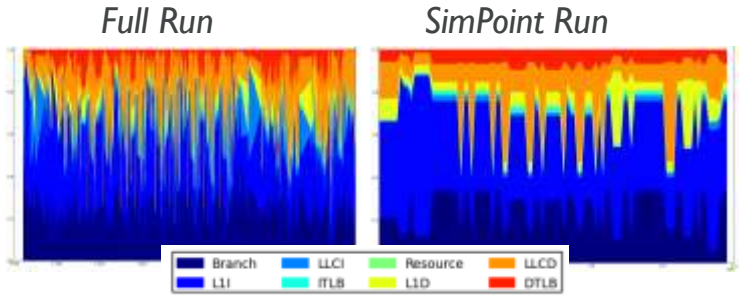
Characterization Methodology

- Record and deterministically playback GUI interactions

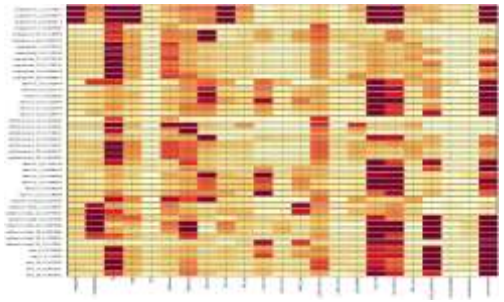


- Quickly and automatically expose differences in elements of a large data set
- Compare and contrast phase behavior





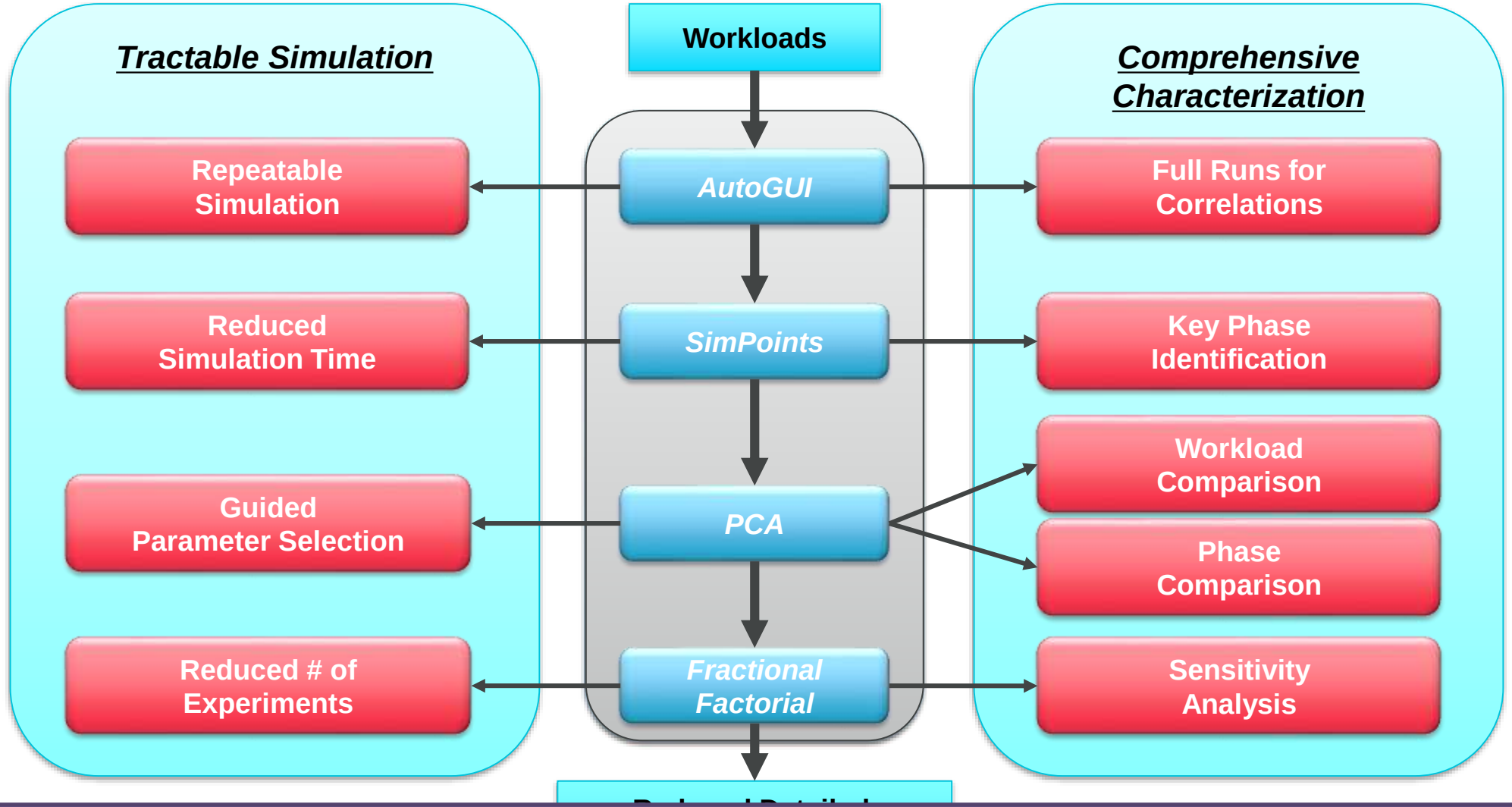
- 300x speedup of our simulations
- Good correlation to full runs for statistics of interest
- Identifies unique phases of software behavior



- Perform high-level coverage architectural exploration using a limited set of experiments

ARM

Characterization Methodology



Sunwoo, et al. "A Structured Approach to the Simulation, Analysis and Characterization of Smartphone Applications." Published at IISWC 2013.

How to Contribute to gem5

Andreas Sandberg

Prerequisites

- gem5's is distributed under a 3-clause BSD license
  - See LICENSE in the repository
- New code must have this license as well!
- It's your responsibility to:
  - Ensure that your contribution is covered by the license.
  - Ensure that you have the right to submit the code
  - Ensure that the right copyright notices are in place

Best practice
“How to operate your friendly reviewer”

How to structure your change

- What characterizes a good change?
  - Small: Smaller changes are easier to review and understand.
  - Well-defined: One commit == logical change
  - No unrelated changes: Don't sneak bug fixes into feature commits
  - Descriptive commit message
  - Always use your real name and email in the commit meta data

- What characterizes a change that makes reviewers cringe?
  - Multiple changes going into the same commit "various bug fixes in Foo"
  - Large changes that could have been broken into incremental changes
  - Poorly written commit messages

The structure of a commit message

Summary:

python: Move native wrappers to the _m5 namespace

Body:

Swig wrappers for native objects currently share the _m5.internal name space with Python code. This is undesirable if we ever want to switch from Swig to some other framework for native binding (e.g., PyBind11 or Boost::Python). This changeset moves all of such wrappers to the _m5 namespace, which is now reserved for native code.

Meta data:

Change-Id: I2d2bc12dbc05b57b7c5a75f072e08124413d77f3
Signed-off-by: Andreas Sandberg <andreas.sandberg@arm.com>
Reviewed-by: Curtis Dunham <curtis.dunham@arm.com>
Reviewed-by: Jason Lowe-Power <jason@lowepower.com>

Commit message: Summary line

Summary: python: Move native wrappers to the _m5 namespace

- *Short* summary of your change (max 65 characters)
  - Think of it as a subject in an email
- Should uniquely identify your change
- Typically the first thing a potential reviewer sees
- Sometimes the only information shown about a change

- Keywords used to identify affected components
  - See the wiki for details

Commit message: Body

Body: Swig wrappers for native objects currently share the _m5.internal name space with Python code. ...

- Should describe your change in detail – think of it as documentation
  - Reviewers will read this before they see any code

- Describe *what* the change does and *why*
  - Not necessarily how, that should be clear from the code
- Describe any implementation trade-offs
- Describe known limitations

Commit message: Metadata

Meta data:

```
Change-Id: I2d2bc12dbc05b57b7c5a75f072e08124413d77f3
Signed-off-by: Andreas Sandberg <andreas.sandberg@arm.com>
Reviewed-by: Curtis Dunham <curtis.dunham@arm.com>
Reviewed-by: Jason Lowe-Power <jason@lowepower.com>
```

- **Change-Id:** Unique ID used by Gerrit to identify the change (generated)
- **Signed-off-by:** It's complicated…
- **Reviewed-by:** Use this to acknowledge reviewers (generated by Gerrit)
- **Reviewed-on:** Link to review request (generated by Gerrit)
- **Reported-by:** Use this to acknowledge users that report bugs
- **Tested-by:** Can be used to acknowledge testers

Developer Certificate of Origin

Proposed Flow

- By making a contribution to this project, I certify that:
  a) The contribution was … by me and I have the right to submit it…; or
  b) … is based upon previous work that … is covered under an appropriate open source license and I have the right under that license to submit that work with modifications… ; or
  c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.
  d) I understand and agree that this project and the contribution are public and that a record of the contribution … is maintained indefinitely and may be redistributed…

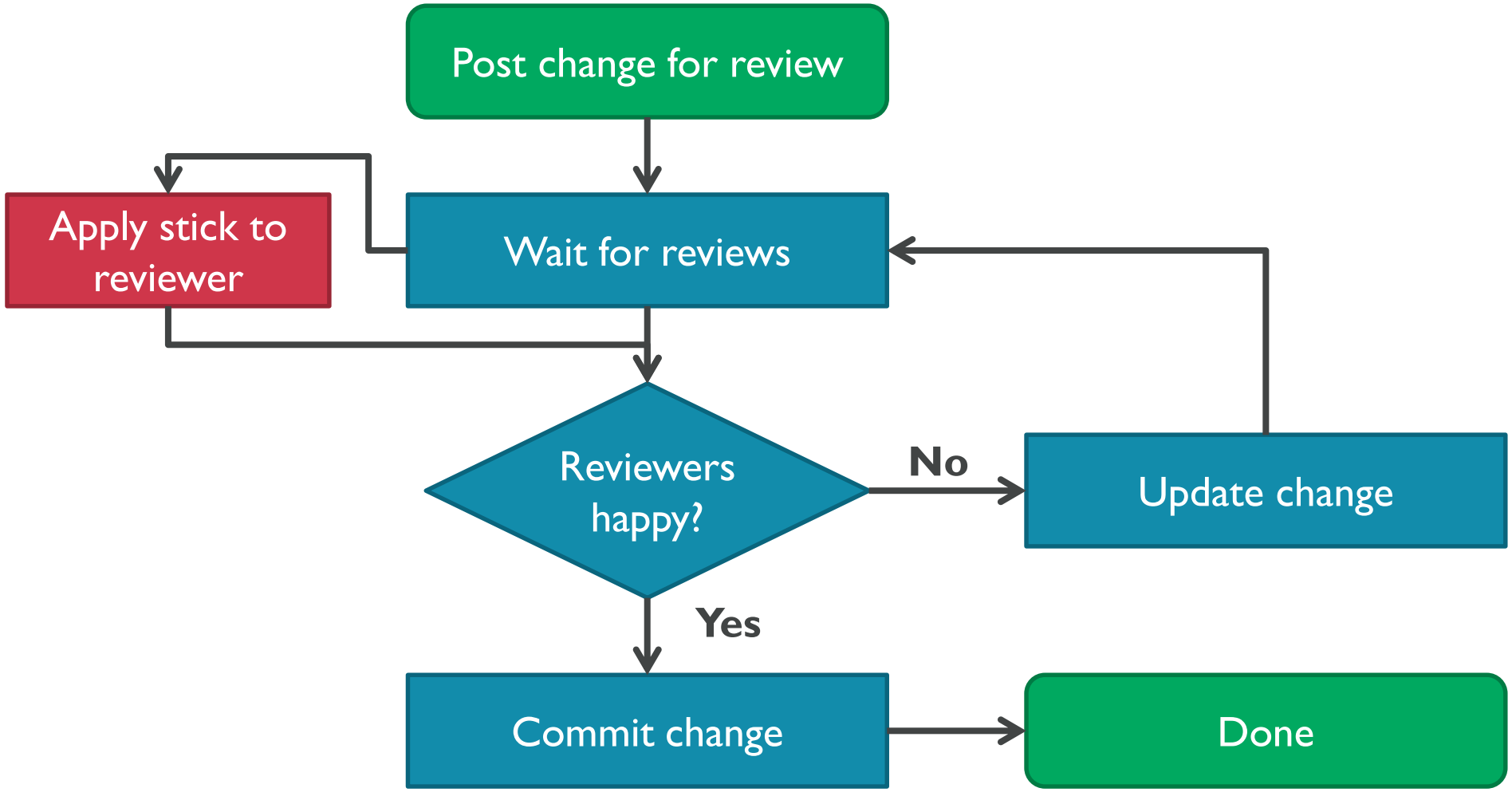
- See the https://developercertificate.org/ for the full version.
- A **Signed-off-by:** tag indicates that you understand and agree to the DCO.

New flow!

Submitting Code:
How to use the new Gerrit-based flow

ARM

Code submission flow



The job of a reviewer

- Evaluate technical aspects
  - Is it doing what it says in the commit message?
  - Is a technically sound implementation?
- Evaluate implementation aspects
  - Is the commit message describing the change?
  - Is it following the style guidelines?
- Legal aspects
  - Patch author's responsibility, but reviewers should look out for obvious issues.

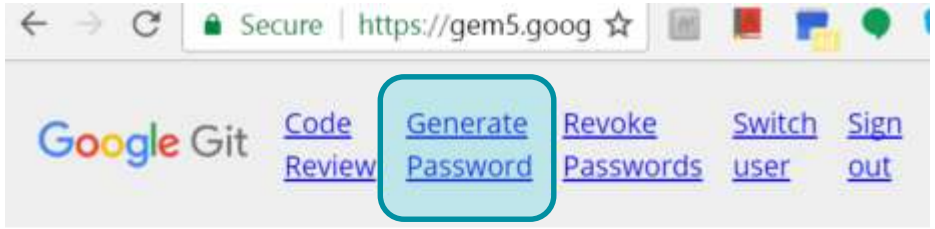
You are the reviewers!

gem5 is changing

- Recently switched from Mercurial to Git
  - Canonical repository on http://gem5.googlesource.com
  - Mirror on GitHub: http://github.com/gem5
- Recently switched from ReviewBoard to Gerrit
  - Automates code submission
  - Tightly integrated with git
  - Google (e.g., GMail) accounts for authentication
  - Will integrate support automatic testing

Setting up gerrit & git

- Prerequisites
  - Google account registered with the email address you use for contributions
- Where to start:
  - http://gem5.googlesource.com
- Git authentication
  - Required to push changes for review
  - Uses https unlike most other installations
  - Requires an authentication cookie



Git repositories on gem5

Name	Description
All-Projects	
All-Users	
public	
public/gem5	
public/testing	

Posting a change for review

- Push to a "magical" git ref:
  - `refs/for/<branch>`: Create a review request
  - `refs/drafts/<branch>`: Create a draft review

- Pushes either updates an existing review or creates a new one
- More advanced usage described in the Gerrit manual

- Tips and tricks:
  - Make sure that you assign one or more reviewers to the change
  - Assign a topic name to related changes

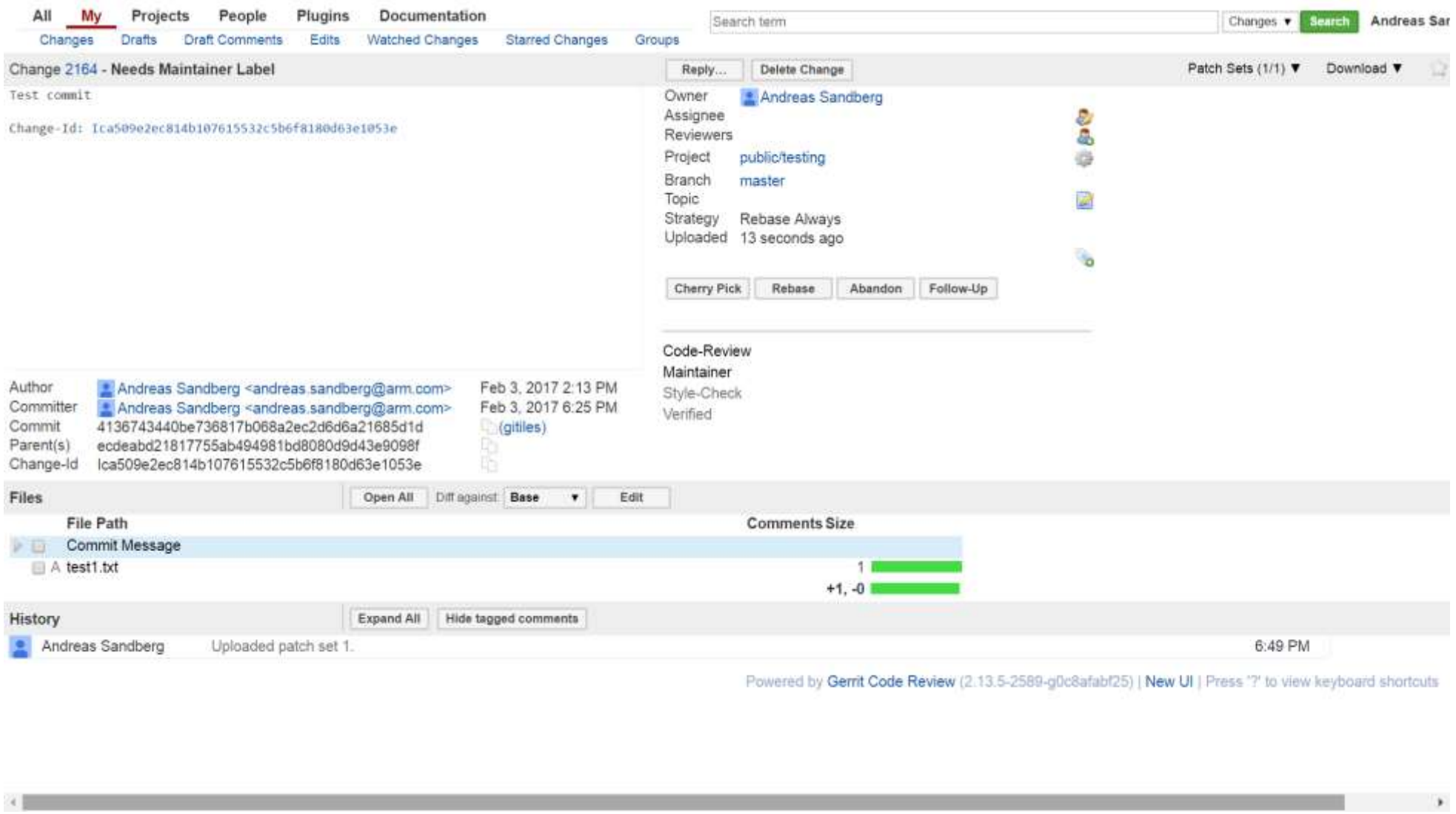
Simple Example

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

```
$ git add -i
$ git commit -m "test commit"
```

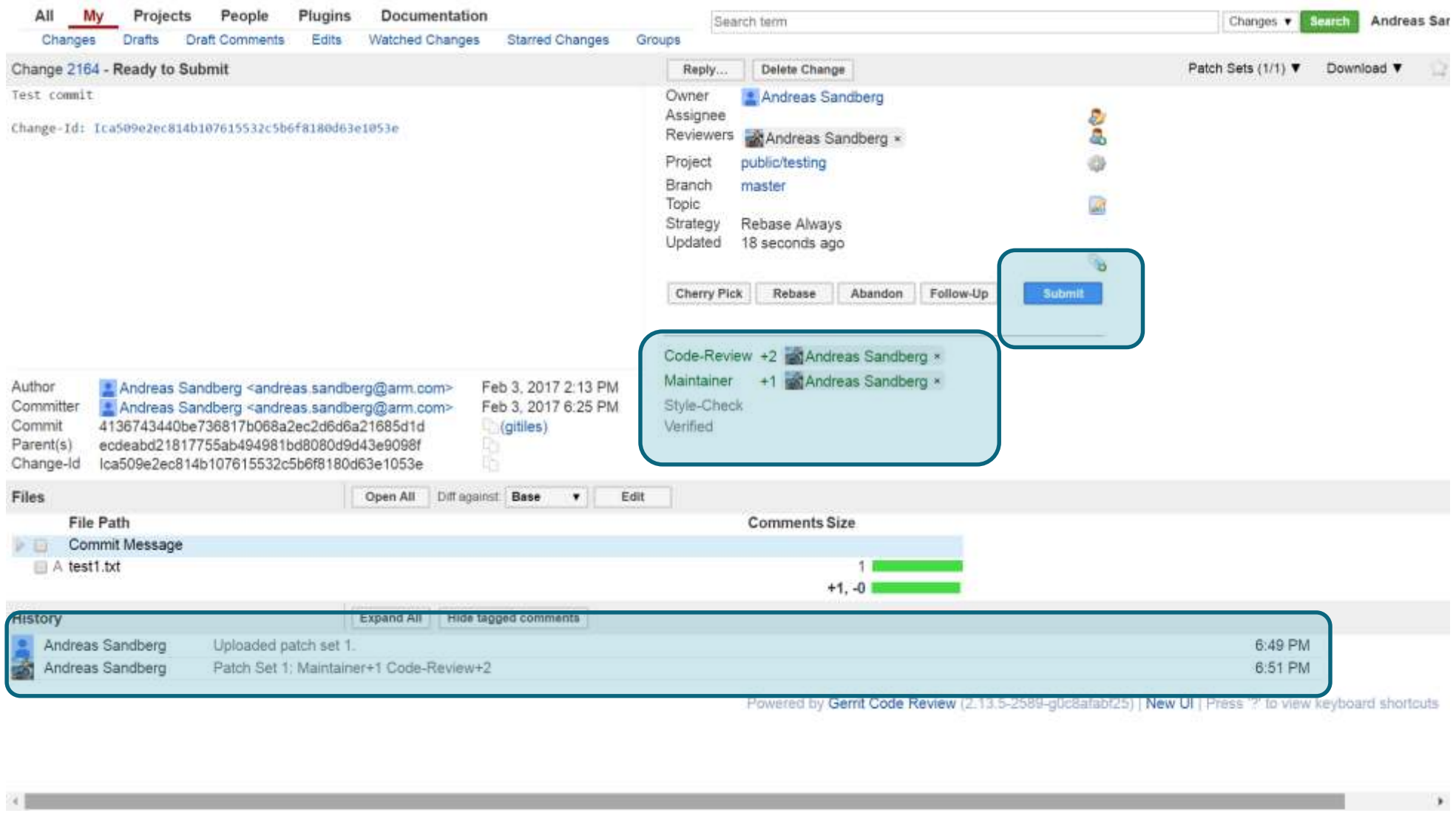
```
$ git push origin HEAD:refs/for/master
…
remote: New Changes:
remote:   https://gem5-review.googlesource.com/2160 Test commit
remote:
To https://gem5.googlesource.com/public/gem5
 * [new branch]      HEAD -> refs/for/master
```

https://gem5-review.googlesource.com/2160



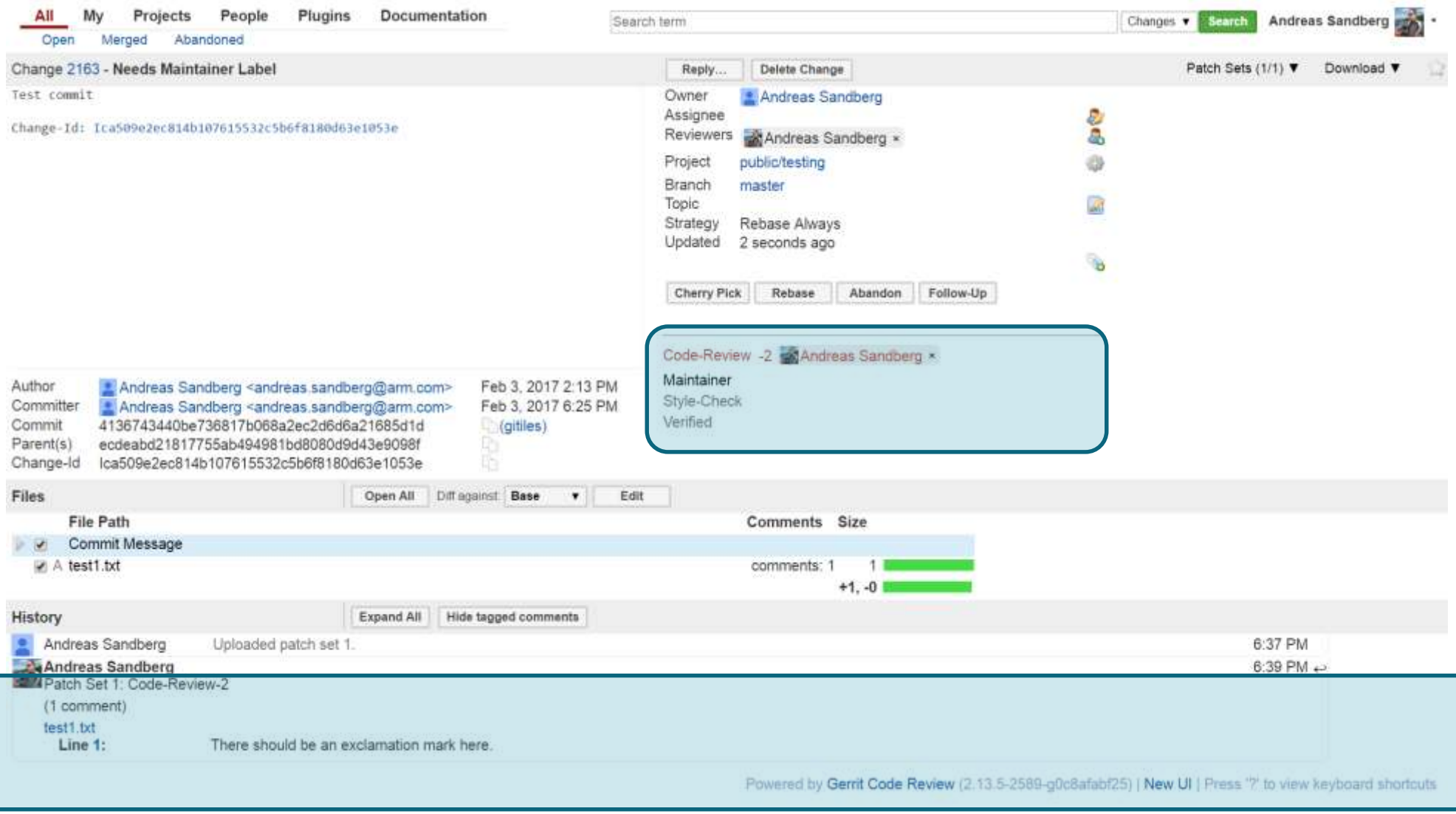
ARM

https://gem5-review.googlesource.com/2160



ARM

https://gem5-review.googlesource.com/2160



ARM

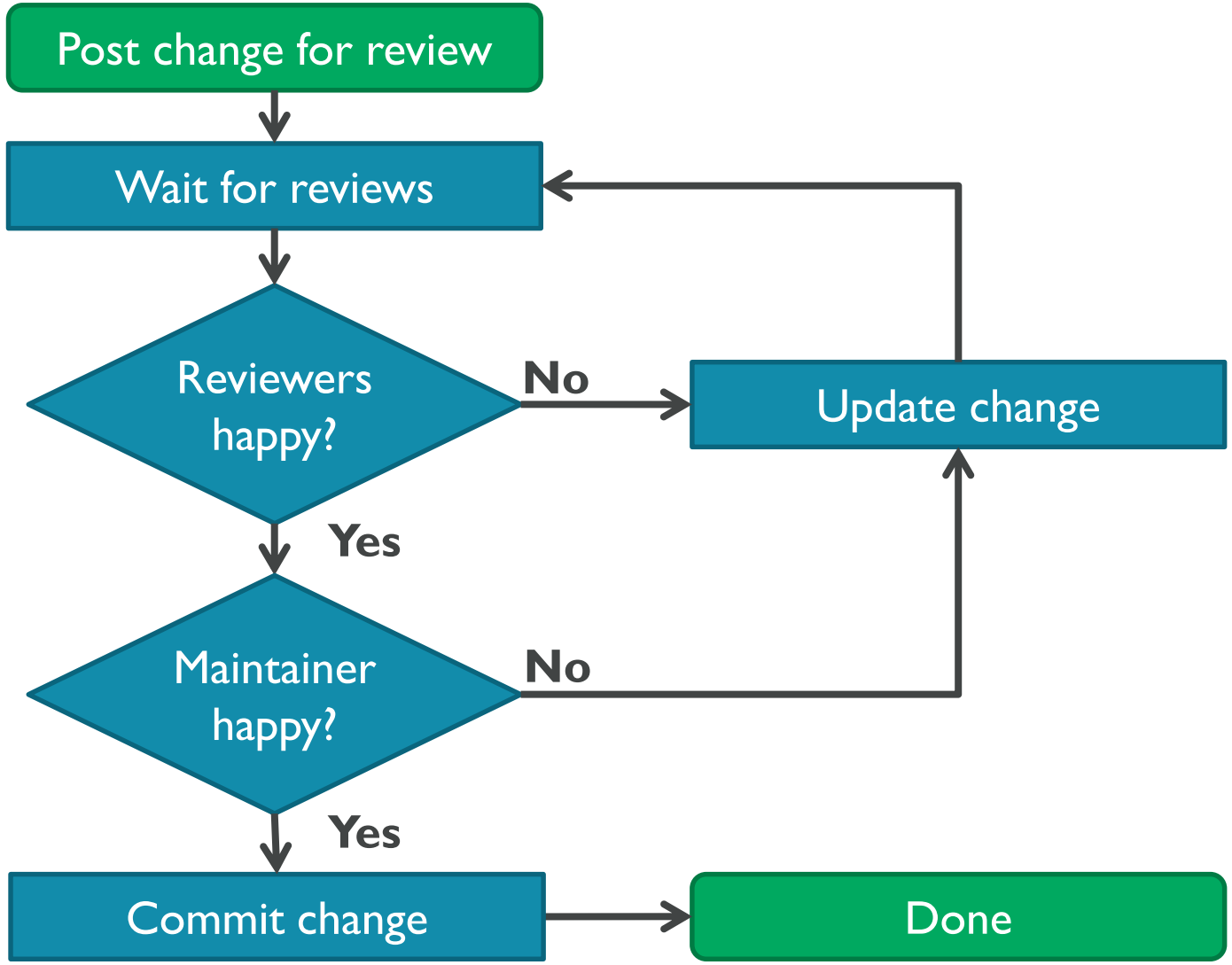
Reviewing code in Gerrit

- Changes can only be submitted if they have been:
  - Reviewed
  - Accepted by a maintainer
  - Passed automatic testing

- Gerrit uses *labels* to enforce these policies:
  - **Code-Review**: Normal code reviews, anyone can use these.
  - **Maintainer**: Only available to maintainers, required for submission.
  - **Verified**: Used by CI system to accept/reject depending on test outcomes
  - **Style-Check**: Automatic style checking

- Maintainers can override labels if they are obviously wrong

Code submission flow



How to review code

- Start with the commit message
  - Does it make sense?
  - Is it a change that makes sense in gem5? Why/Why not?
- Look at the code
  - Is it solving the problem in the description?
  - Is the implementation technically sound? Are there obvious bugs?
- Comment on the code and submit a review score
  - -2: Don't submit under any circumstances (blocks submission)
  - …
  - +2: Looks good, approved!
- Be polite and kind
  - Developers and reviewers are people too!

Further information - gem5 related papers from ARM Research

- Sunwoo, Dam, et al. "A structured approach to the simulation, analysis and characterization of smartphone applications." IISWC'13
- Gutierrez, Anthony, et al. "Sources of error in full-system simulation." ISPASS'14
- Hansson, Andreas, et al. "Simulating DRAM controllers for future system architecture exploration." ISPASS'14
- De Jong, Rene, and Andreas Sandberg. "NoMali: Simulating a realistic graphics driver stack using a stub GPU." ISPASS'16
- Rusitoru, Roxana. "ARMv8 micro-architectural design space exploration for high performance computing using fractional factorial." PMBS'15
- Vasileios Spiliopoulos, et.al. "Introducing DVFS-Management in a Full-System Simulator." MASCOTS '13
- Matthew J. Walker, et al. "Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs." IEEE Trans. on CAD of Integrated Circuits and Systems 36'2017

Further information - gem5 related papers from ARM Research

- Jagtap, Radhika, et al. "Elastic traces for fast and accurate system performance exploration." ISPASS'16
- Mohammad Alian, et al. "dist-gem5: Distributed simulation of computer clusters." ISPASS'17

ARM Research Summit

11-13 September 2017
Robinson College, Cambridge, UK

Submission deadline - 30 April 2017
Early-bird discount ends - 30 June 2017