# gem5 Tutorial

Sascha Bischoff

Andreas Hansson

# Outline

- Introduction
  - Why a system simulator?
  - Where it comes from?
  - What it can do?
  - High-level features
- Basics
  - Compiling
  - Running
- Using the simulator
  - Checkpoints
  - Sampling
  - Instrumenting
  - Results
- Debugging
  - Trace
  - Debugging the simulator
  - Debugging the execution

# Outline

- Memory System
  - Overview
  - Ports
  - Transport interfaces
  - Caches and Coherence
  - Interconnect components
- CPU Models
  - Simple
  - InOrder
  - Out-of-order
- Common Tasks
  - Adding a statistic, SimObject, or Instruction
- Conclusion

# INTRODUCTION

# Importance of System Simulation

- Why make it so complicated when I only care about
  - Benchmark run time
  - CPU performance
  - Interconnect latencies
  - DRAM controller scheduling
- CPU behavior depends on the memory system, and the behavior of the memory system depends on the CPUs
  - Complex interactions on many different levels, application, JIT, OS, caches, interconnect, memory controllers, devices
  - Gluing the pieces together, e.g. using traces, does not capture these dependencies.
- Solution: A system simulator

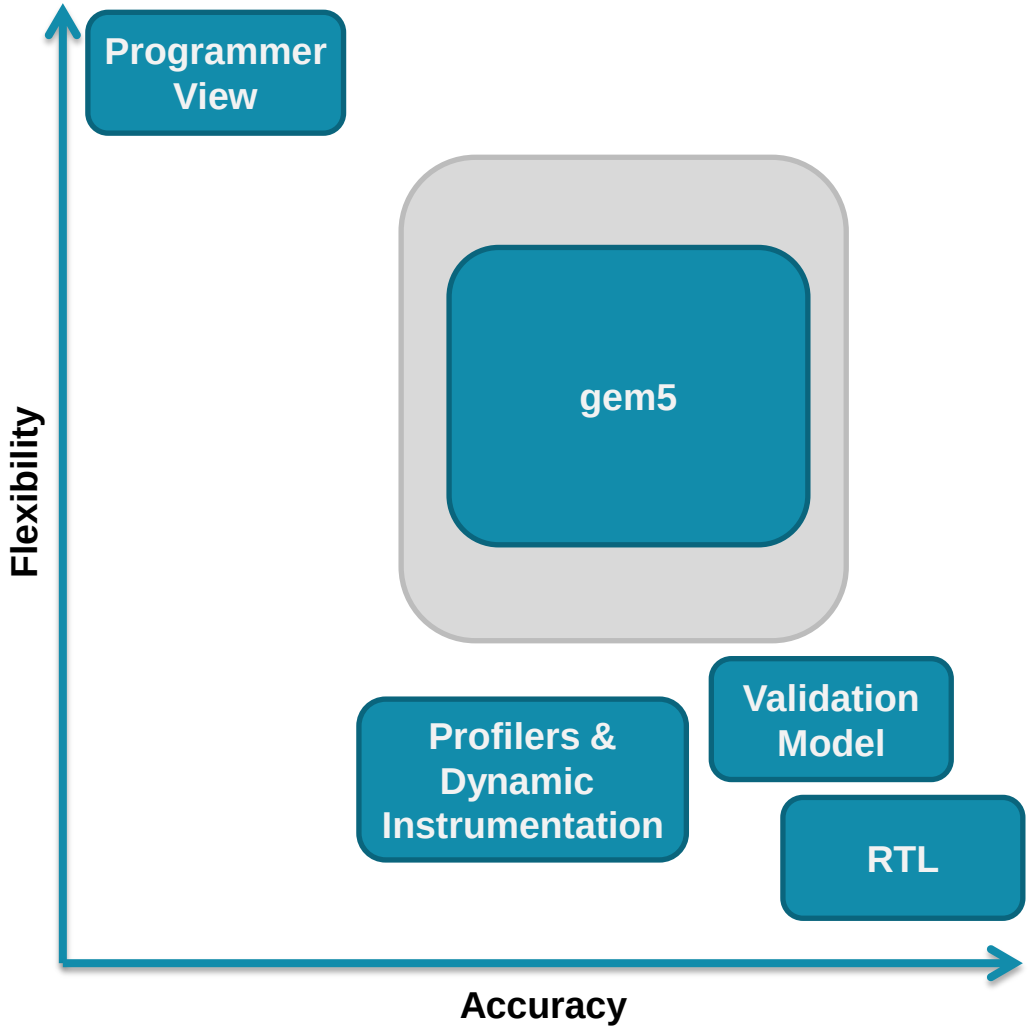


gem5

# System Simulator

- Built from a combination of M5 and GEMS
  - In doing so we lost all capitalization: gem5
- Self-contained simulation framework
  - Does not rely on many simulators glued together
    - Although you're welcome to glue things together
  - Built on a discrete-event simulation kernel
- Rich availability of modules in the framework
  - Out of the box it can model entire systems
    - Not just CPU intensive apps
    - Not just memory system with traces
    - Not DRAM in isolation
    - Not execution without I/O

# Why a Flexible Simulation Tool?



# Envisioned use-cases

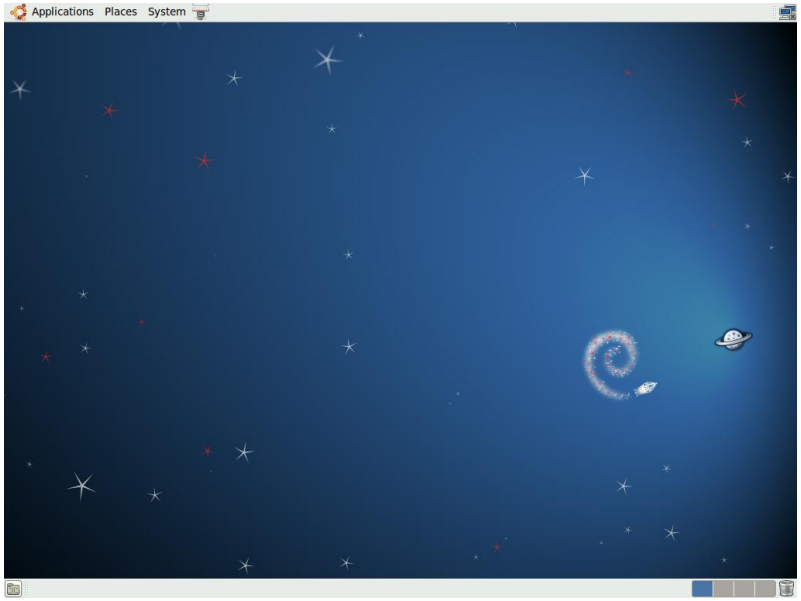
- SW development and verification
  - Binary-translation models (e.g. OVP/QEMU) are fast enough to do this and have a mature SW development environment
- HW/SW performance verification
  - Need performance measures of 1st order accuracy, capturing the things that actually matter
- Early Architectural Exploration
  - Need an environment where it is fast and easy to model and connect the key architectural components of hardware platform
- HW/SW functional verification
  - RTL is representative enough and has enough visibility and a mature methodology



gem5

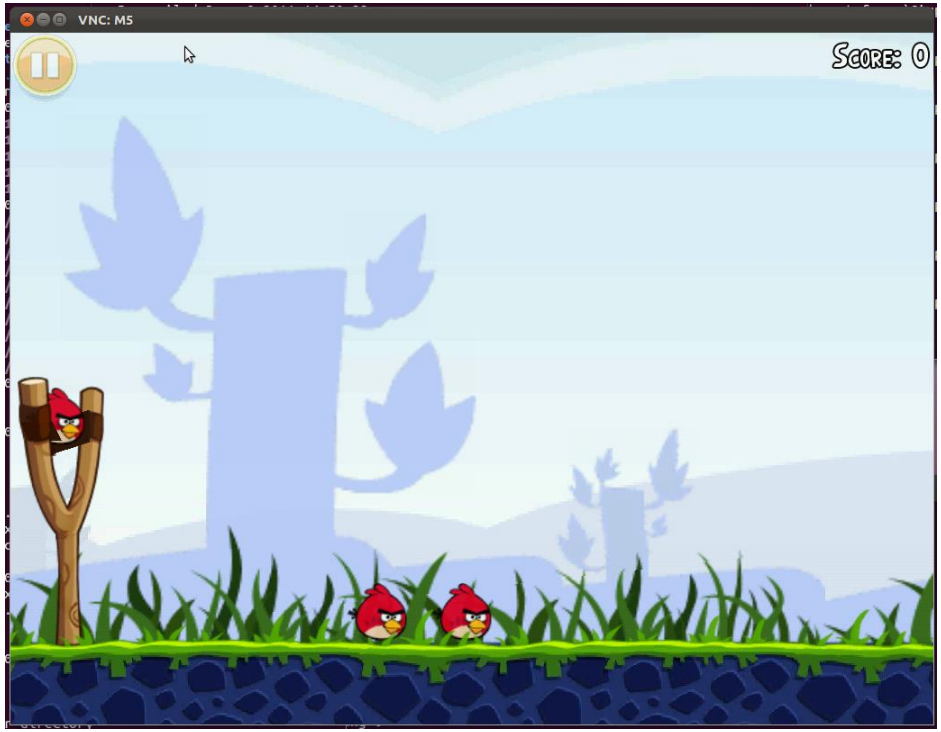
# Operating Systems & Apps

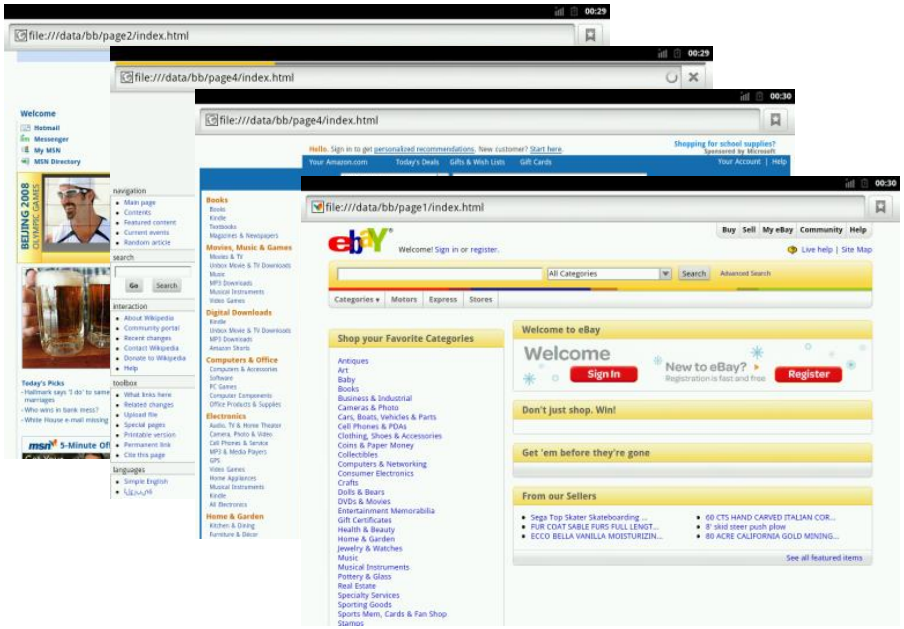
## Ubuntu 11.06 (Linux 2.6.35.8)




## Android Gingerbread

# Real Applications

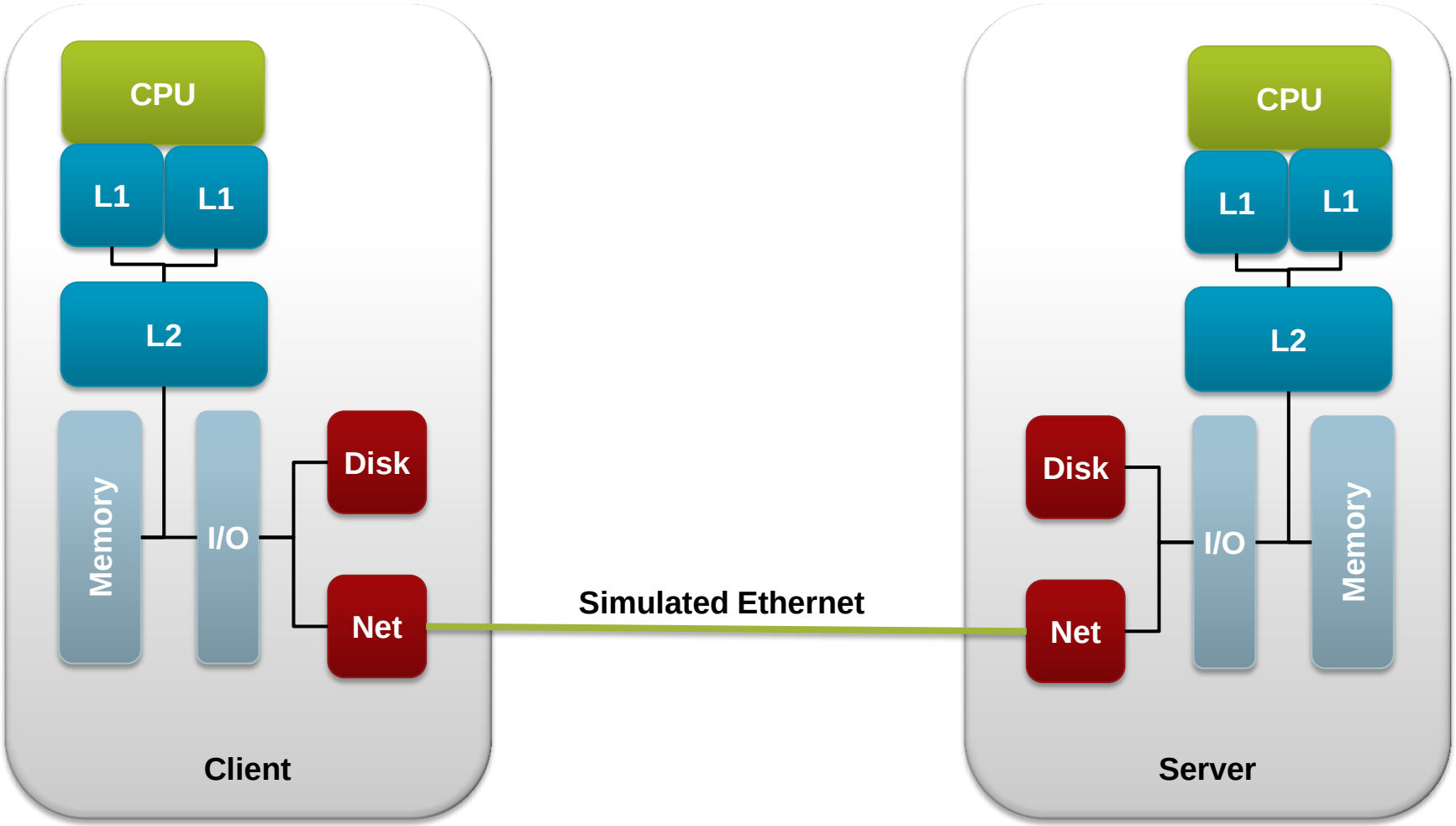





gem5

# Multiple System Simulations



# Main Goals

***Open source tool focused on architectural modeling***

- Flexibility
  - Multiple CPU models, memory systems, and device models
    - Across the speed vs. accuracy spectrum
- Availability
  - For both academic and corporate researchers
  - No dependence on proprietary code
  - BSD license
- Collaboration
  - Combined effort of many with different specialties
  - Active community leveraging collaborative technologies



gem5

# High-level Features

- Configurable CPU models
  - Simple one-IPC (SimpleAtomic/Timing)
  - Detailed in-order execution (InOrder)
  - Detailed out-of-order execution (O3)
  - Hardware-accelerated fast forwarding (KVM)
- Pluggable memory system
  - Stitch memory system together out of components
  - Use Wisconsin's Ruby
- Device Models
  - Enough device models to boot Linux, Android
- Boot real operating systems
  - Linux, Android
- Many ISAs

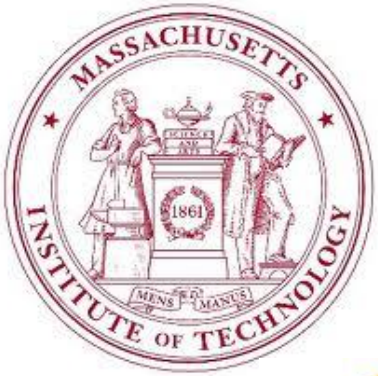


gem5

# What we would like gem5 to be

- Something that spares you the pain we've been through
  - A community resource
- Modular enough to localize changes
  - Contribute back, and spare others some pain
- A path to reproducible/comparable results
  - A common platform for evaluating ideas
- Simulator of choice for performance exploration

# Where did it come from





ARM®


AMD



hp®
invent

gem5

# BASICS

# Building gem5

- Platforms
  - Linux, BSD, MacOS X, Solaris, etc
  - 64 bit machines help quite a bit
- Tools
  - GCC/G++ 4.4+ (or clang 2.9+)
  - Python 2.4+
  - SCons 0.98.1+
    - http://www.scons.org
  - SWIG 1.3.40+
    - http://www.swig.org
- If using Ubuntu install
  - apt-get install python-dev scons m4 build-essential g++ swig zlib-dev



gem5

# Compile Targets

- build/<isa>/<binary>

- ISAs:
  - ARM, ALPHA, MIPS, SPARC, POWER, X86

- Binaries
  - **gem5.debug** debug build, symbols, tracing, assert
  - **gem5.opt** optimized build, symbols, tracing, assert
  - **gem5.fast** optimized build, no debugging, no symbols, no tracing, no assertions
  - **gem5.prof** gem5.fast + profiling support

# Sample Compile

```
21:36:01 [/work/gem5] scons build/ARM/gem5.opt –j4
scons: Reading SConscript files ...
Checking for leading underscore in global variables...(cached) yes
Checking for C header file Python.h... (cached) yes
Checking for C library dl... (cached) yes
Checking for C library python2.7... (cached) yes
Checking for accept(0,0,0) in C++ library None... (cached) yes
Checking for zlibVersion() in C++ library z... (cached) yes
Checking for clock_nanosleep(0,0,NULL,NULL) in C library None... (cached) no
Checking for clock_nanosleep(0,0,NULL,NULL) in C library rt... (cached) no
Can't find library for POSIX clocks.
Checking for C header file fenv.h... (cached) yes
Reading SConsopts
Building in /work/gem5/build/ARM
Using saved variables file /work/gem5/build/variables/ARM
Generating LALR tables
WARNING: 1 shift/reduce conflict
scons: done reading SConscript files.
scons: Building targets ...
 [     CXX] ARM/sim/main.cc -> .o
 [ TRACING]  -> ARM/debug/Faults.hh
 [GENERATE]  -> ARM/arch/interrupts.hh
 [GENERATE]  -> ARM/arch/isa_traits.hh
 [GENERATE]  -> ARM/arch/microcode_rom.hh
 [ CFG ISA]  -> ARM/config/the_isa.hh
```

# Running Simulation

```
21:58:32 [ /work/gem5] ./build/ARM/gem5.opt -h
Usage
=====
  gem5.opt [gem5 options] script.py [script options]

 gem5 is copyrighted software; use the --copyright option for details.

Options
=======
--version                 show program's version number and exit
--help, -h                show this help message and exit
--build-info, -B          Show build information
--copyright, -C           Show full copyright information
--readme, -R              Show the readme
--outdir=DIR, -d DIR      Set the output directory to DIR [Default: m5out]
--redirect-stdout, -r     Redirect stdout (& stderr, without -e) to file
--redirect-stderr, -e     Redirect stderr to file
--stdout-file=FILE        Filename for -r redirection [Default: simout]
--stderr-file=FILE        Filename for -e redirection [Default: simerr]
--interactive, -i         Invoke the interactive interpreter after running the
                          script
--pdb                     Invoke the python debugger before running the script
--path=PATH[:PATH], -p PATH[:PATH]
                          Prepend PATH to the system path when invoking the
                          script
```

# Running Simulation

```
Statistics Options
------------------
--stats-file=FILE          Sets the output file for statistics [Default: stats.txt]

Configuration Options
---------------------
--dump-config=FILE     Dump configuration output file [Default: config.ini]
--json-config=FILE     Create JSON output of the configuration [Default: config.json]

Debugging Options
-----------------
--debug-break=TIME[,TIME]
                           Cycle to create a breakpoint
--debug-help               Print help on trace flags
--debug-flags=FLAG[,FLAG]
                           Sets the flags for tracing (-FLAG disables a flag)
--remote-gdb-port=REMOTE_GDB_PORT
                           Remote gdb base port (set to 0 to disable listening)

Trace Options
-------------
--trace-start=TIME         Start tracing at TIME (must be in ticks)
--trace-file=FILE          Sets the output file for tracing [Default: cout]
--trace-ignore=EXPR        Ignore EXPR sim objects
```

# gem5 has two fundamental modes

- Full system (FS)
  - For booting operating systems
  - Models bare hardware, including devices
  - Interrupts, exceptions, privileged instructions, fault handlers
  - Simulated UART output
  - Simulated frame buffer output
- Syscall emulation (SE)
  - For running individual applications, or set of applications on MP
  - Models user-visible ISA plus common system calls
  - System calls emulated, typically by calling host OS
  - Simplified address translation model, no scheduling
- Now dependent on how you run the binary
  - No longer need to compile different binaries

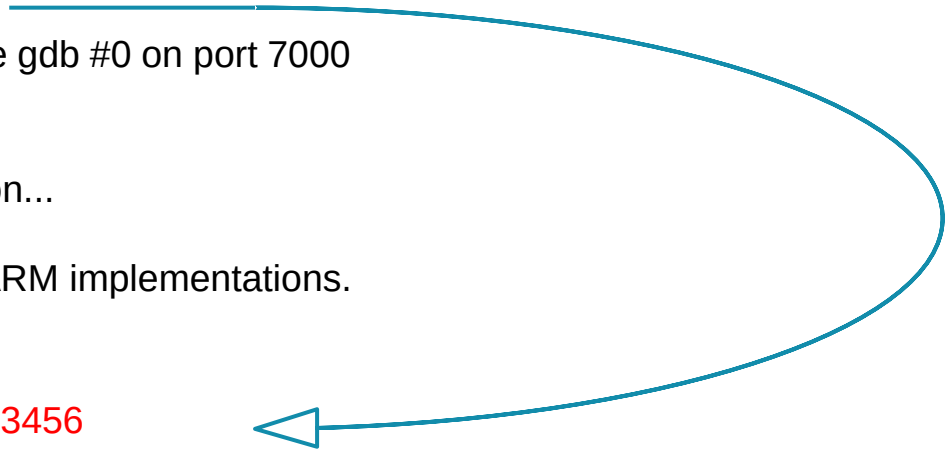
# Sample Run – Syscall Emulation

```
2:08:12 [/work/gem5] ./build/ARM/gem5.opt configs/example/se.py  \
                          -c tests/test-progs/hello/bin/arm/linux/hello

gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Mar 18 2012 21:58:16
gem5 started Mar 18 2012 22:10:24
gem5 executing on daystrom
command line: ./build/ARM/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/arm/linux/hello

Global frequency set at 1000000000000 ticks per second
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 3107500 because target called exit()
```

# Sample Run – Full System

**Command Line:**

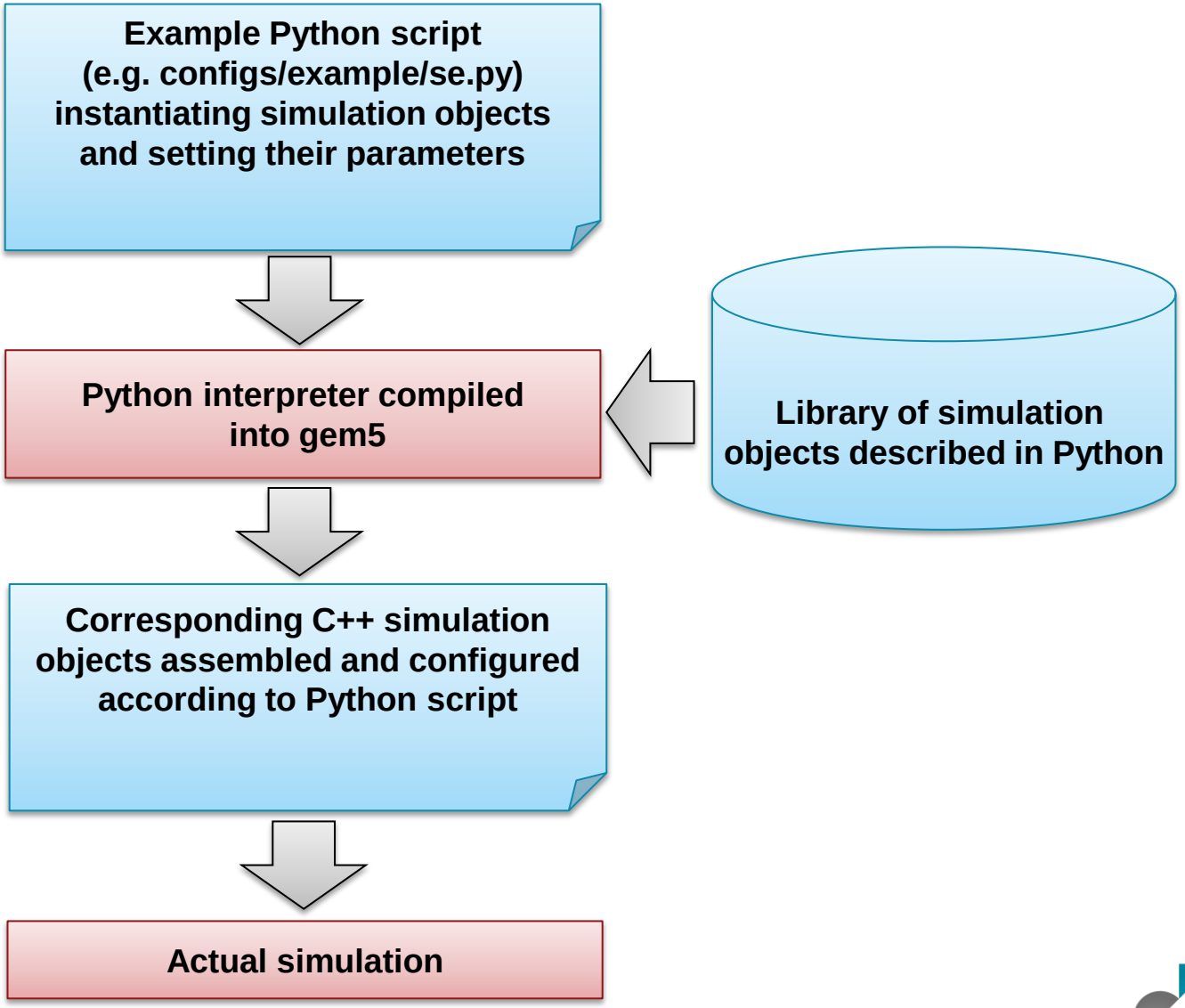
```
22:13:19 [/work/gem5] ./build/ARM/gem5.opt configs/example/fs.py
…
info: kernel located at: /dist/binaries/vmlinux.arm.smp.fb.2.6.38.8
Listening for system connection on port 5900
Listening for system connection on port 3456
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
info: Using bootloader at address 0x80000000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
warn: The clidr register always reports 0 caches.
warn: clidr LoUIS field of 0b001 to match current ARM implementations.
```



**Terminal:**

```
22:13:19 [/work/gem5] ./util/term/m5term 127.0.0.1 3456
==== m5 slave terminal: Terminal 0 ====
[    0.000000] Linux version 2.6.38.8-gem5 (saidi@zeep) (gcc version 4.5.2 (Sourcery G++ Lite 2011.03-41) )
#1 SMP Mon Aug 15 21:18:38 EDT 2011
[    0.000000] CPU: ARMv7 Processor [350fc000] revision 0 (ARMv7), cr=10c53c7f
[    0.000000] CPU: VIPT nonaliasing data cache, VIPT nonaliasing instruction cache
[    0.000000] Machine: ARM-RealView PBX
…
starting pid 354, tty '': '/sbin/getty -L ttySA0 38400 vt100'

AEL login:
```

# Sample Run – Behind the scenes



# Objects

- Everything you care about is an object (C++/Python)
  - Assembled using Python, simulated using C++
  - Derived from SimObject base class
  - Common code for creation, configuration parameters, naming, checkpointing, etc.
- Uniform method-based APIs for object types
  - CPUs, caches, memory, etc.
  - Plug-compatibility across implementations
    - Functional vs. detailed CPU
    - Conventional vs. indirect-index cache
- Easy replication: cores, multiple systems, . . .



gem5

# Events

- Standard discrete-event timing model
  - Global logical time in “ticks”
  - No fixed relation to real time
  - Constants in src/sim/core.hh always relate ticks to real time
- Picoseconds used in our examples
  - Objects schedule their own events
- Flexibility for detail vs. performance trade-offs
  - E.g., a CPU typically schedules event at regular intervals
- Every cycle or every n picoseconds
  - Won’t schedule self if stalled/idle



gem5

# Ports

- Used for connecting MemObjects together
  - e.g. enable a CPU to issue reads/writes to a memory
- Correspond to real structural ports on system components
  - e.g. CPU has an instruction and a data port
- Ports have distinct roles, and always appear in pairs
  - A MasterPort is connected to a SlavePort
  - Similar to TLM-2 initiator and target socket
- Send and receive function pairs transport packets
  - sendAtomic() on a MasterPort calls recvAtomic() on connected SlavePort
  - Implementation of recvAtomic is left to SlavePort subclass
- Result: class-specific handling with arbitrary connections and only a single virtual function call

# Transport interfaces

- Three transport interfaces: Functional, Atomic, Timing
  - All have their own transport functions on the ports
  - sendFunctional(), sendAtomic(), sendTiming()

- Functional:
  - Used for loading binaries, debugging, introspection, etc.
  - Accesses happen instantaneously
    - Reads get the “newest” copy of the data
    - Writes update data everywhere in the memory system
  - Completes a transaction in a single function call
    - Requests complete before sendFunctional() returns
  - Equivalent to TLM-2 debug transport
  - Objects that buffer packets must be queried and updated as well

# Transport interfaces (cont'd)

- Atomic:
  - Completes a transaction in a single function call
    - Requests complete before sendAtomic() returns
  - Models state changes (cache fills, coherence, etc.)
  - Returns approximate latency w/o contention or queuing delay
  - Similar to TLM-2 blocking transport (without *wait*)
  - Used for loosely-timed simulation (fast forwarding) or warming caches

- Timing:
  - Models all timing/queuing in the memory system
  - Split transaction into multiple phases
    - sendTiming() initiates send of request to slave
    - Slave later calls sendTiming() to send response packet
  - Similar to TLM-2 non-blocking transport
  - Used for approximately-timed simulation

- Atomic and Timing accesses cannot coexist in the same system

# Statistics

- Wide variety of statistics available
  - Scalar
  - Average
  - Vector
  - Formula
  - Histogram
  - Sparse Histogram
  - Distribution
  - Vector Distribution

- Currently output text
  - Soon to output SQLite database
    - or any other format you wish to add

# Checkpointing

- Simulator can create checkpoints
  - Restore from them at a later time
  - Normally create checkpoint in atomic memory mode
    - After reaching the ROI
  - Restore from checkpoint and change the system to be more detailed

- Constraints
  - Original simulation and test simulations must have
  - Same ISA; number of cores; memory map
  - We don’t currently checkpoint cache state

# Fast Forwarding

- Traditionally fast forwarded using the atomic CPU model
  - Checkpoint created when ROI reached

- KVM acceleration is now possible
  - Use the host CPU to execute guest instructions natively
  - Massive speedup, even compared to atomic CPU
    - Can interact with simulated system!
  - One caveat:
    - Simulation ISA must match host ISA
    - Currently working on ARM ISA

# RUNNING AN EXPERIMENT

# Running a Syscall Emulation Experiment

- Compiling a benchmark
- Running a benchmark in SE mode w/atomic CPU
- Running a benchmark with a detailed CPU
- Stats output
- Instrumenting and creating a checkpoint
- Running from that checkpoint

# Compiling a benchmark for SE

- Do all these experiments with queens.c
  - Very old benchmark, but it's easy to get and understand

[/work/gem5] wget https://llvm.org/svn/llvm-project/test-suite/tags/RELEASE_14/SingleSource/Benchmarks/McGill/queens.c

[/work/gem5] arm-linux-gnueabi-gcc –DUNIX –o queens queens.c –static

- All binaries must be compiled with static flag
  - In principle you could run a dynamic linker, but no one has done the work yet

# Running Compiled Program

**Command Line:**

[/work/gem5] ./build/ARM/gem5.opt configs/example/se.py -c queens –o 16
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.
…
**** REAL SIMULATION ****
info: Entering event queue @ 0. Starting simulation...
16 queens on a 16x16 board...

```
Q - - - - - - - - - - - - - - -
- - Q - - - - - - - - - - - - -
- - - - Q - - - - - - - - - - -
- Q - - - - - - - - - - - - - -
- - - - - - - - - - - - Q - - -
- - - - - - - - Q - - - - - - -
- - - - - - - - - - - - - Q - -
- - - - - - - - - - - Q - - - -
- - - - - - - - - - - - - - Q -
- - - - - Q - - - - - - - - - -
- - - - - - - - - - - - - - - Q
- - - - - - Q - - - - - - - - -
- - - Q - - - - - - - - - - - -
- - - - - - - - - - Q - - - - -
- - - - - - - Q - - - - - - - -
- - - - - - - - - Q - - - - - -
```

**SE mode output is printed on the terminal**

Exiting @ tick 33345000 because target called exit()

# Statistics Output

[/work/gem5] cat m5out/stats.txt

```
---------- Begin Simulation Statistics ----------
sim_seconds                                 0.002038                       # Number of seconds simulated
sim_ticks                                2038122000                       # Number of ticks simulated
final_tick                               2038122000                       # Number of ticks from beginning of simulation
sim_freq                                 1000000000000                    # Frequency of simulated ticks
host_inst_rate                                2581679                       # Simulator instruction rate (inst/s)
host_op_rate                                  2781442                       # Simulator op (including micro ops) rate(op/s)
…
system.physmem.bytes_read                    17774713                       # Number of bytes read from this memory
system.physmem.bytes_written                   656551                       # Number of bytes written to this memory
…
system.cpu.numCycles                          4076245                       # number of cpu cycles simulated
system.cpu.committedInsts                     2763927                       # Number of instructions committed
system.cpu.committedOps                       2977829                       # Number of ops (including micro ops) committed
```

# Running with caches and detailed CPU

[/work/gem5] ./build/ARM/gem5.opt configs/example/se.py -c queens –o 16 --caches --l2cache \
--cpu-type=arm_detailed

…

16 queens on a 16x16 board...

```
Q - - - - - - - - - - - - - - -
- - Q - - - - - - - - - - - - -
- - - - Q - - - - - - - - - - -
- Q - - - - - - - - - - - - - -
- - - - - - - - - - - - Q - - -
- - - - - - - - Q - - - - - - -
- - - - - - - - - - - - - Q - -
- - - - - - - - - - - Q - - - -
- - - - - - - - - - - - - - Q -
- - - - - Q - - - - - - - - - -
- - - - - - - - - - - - - - - Q
- - - - - - Q - - - - - - - - -
- - - Q - - - - - - - - - - - -
- - - - - - - - - - Q - - - - -
- - - - - - - Q - - - - - - - -
- - - - - - - - - Q - - - - - -
```

Exiting @ tick 1686872500 because target called exit()

# Stats Output

[/work/gem5] cat m5out/stats.txt

```
---------- Begin Simulation Statistics ----------
sim_seconds                                 0.001687                       # Number of seconds simulated
sim_ticks                                1686872500                       # Number of ticks simulated
final_tick                               1686872500                       # Number of ticks from beginning of simulation
sim_freq                                 1000000000000                     # Frequency of simulated ticks
host_inst_rate                                 103418                       # Simulator instruction rate (inst/s)
host_op_rate                                   111421                       # Simulator op (including micro ops) rate(op/s)
…
system.physmem.bytes_read                       43968                       # Number of bytes read from this memory
system.physmem.bytes_written                        0                       # Number of bytes written to this memory
…
system.cpu.numCycles                          4076245                       # number of cpu cycles simulated
system.cpu.committedInsts                     2763927                       # Number of instructions committed
system.cpu.committedOps                       2977829                       # Number of ops (including micro ops) committed
…
system.cpu.commit.branchMispredicts             93499                       # The number of times a branch was mispredicted
system.cpu.cpi                                1.220635                       # CPI: Cycles Per Instruction
…
```

# Check pointing at the Region of Interest

- Edit queens.c
  - #include “util/m5/m5op.h”
    - Contains various op codes that cause the simulator to take action
  - Work happens in:

```
/* Find all solutions (begin recursion) */
   m5_checkpoint(0,0);
   find(0);

    …
    if (level == queens) {                  /* Placed all queens?  Stop. */
          ++solutions;                      /* Congrats, this is a solution! */
          m5_dumpreset_stats(0,0);
```

  - Recompile the binary when done:

```
[/work/gem5] arm-linux-gnueabi-gcc –DUNIX –o queens-w-chkpt queens.c    \
                    util/m5/m5op_arm.S --static
```

# Create a Checkpoint

**Command Line:**

[/work/gem5] ./build/ARM/gem5.opt configs/example/se.py -c queens –o 16
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.
…
**** REAL SIMULATION ****
info: Entering event queue @ 0. Starting simulation...
Writing checkpoint
info: Entering event queue @ 6805000. Starting simulation...
…
Exiting @ tick 2038122000because target called exit()

**Directory:**

[/work/gem5] ls m5out
config.ini config.json cpt.6805000 stats.txt

# Running from the checkpoint

**Command Line:**

```
[/work/gem5] ./build/ARM/gem5.opt configs/example/se.py  -c queens –o 16 --caches --l2cache \
                                                --cpu-type=arm_detailed  --checkpoint-dir=m5out -r 1
…
Switch at curTick count:10000
info: Entering event queue @ 6805000.  Starting simulation...
Switched CPUS @ tick 6815000
Changing memory mode to timing
switching cpus
**** REAL SIMULATION ****
info: Entering event queue @ 6815000.  Starting simulation...
```

**Stats:**

```
[/work/gem5] cat m5out/stats.txt
---------- Begin Simulation Statistics ----------
sim_seconds                                 0.001595
system.switch_cpus.cpi                      1.191434
…
---------- End Simulation Statistics   ----------

---------- Begin Simulation Statistics ----------
sim_seconds                                 0.000064
system.switch_cpus.cpi                      1.662081
…
---------- End Simulation Statistics   ----------
```

**Stats within find(0);**

**Stats for when printing happened**

# Running a Full System Experiment

- Mounting disk images and putting files on them
- Creating scripts that run an experiment
  - Creating a checkpoint from within the simulation
- Running the experiment
  - Using m5term
- Running experiments from this checkpoint

# Mounting a Disk Image

- To mount a disk image you need to be root
  - You can do it within a VM

- Mount command:

```
[/work/gem5] mount –o loop,offset=32256 linux-arm-ael.img /mnt

[/work/gem5] ls /mnt
bin boot dev etc home lib lost+found media mnt proc root sbin sys tmp usr var writable

[/work/gem5] cp queens /mnt

[/work/gem5] cp queens-w-chkpt /mnt
```

- Make sure you unmount before you use the image

```
[/work/gem5] umount /mnt
```

# Create a Boot Script

- Scripts are executed by startup scripts on images distributed with gem5
  - Files are read from *host* system after booting
  - Written into simulated file system
  - Executed like a shell script

**configs/boot/queens.rcS:**

```
#!/bin/sh

# Wait for system to calm down
sleep 10

# Take a checkpoint in 100000 ns
m5 checkpoint 100000

# Reset the stats
m5 resetstats

# Run queuens
/queens 16

# Exit the simulation
m5 exit
```

# gem5 Terminal

- Default output from full-system simulation is on a UART
  - m5term is a terminal emulator that lets you connect to it
- Code is in src/util/term
  - Run make in that directory and make install
- Binary takes two parameters
  - ./m5term <host> <port>
- If you're running it locally, use the loopback interface
  - 127.0.0.1
- Port number is printed when gem5 starts
  - Tries 3456 and increments until it find a free port
  - So if you're running multiple copies on a single machine you might find 3457, 3458, …

# Running in Full System Mode

**Command Line:**

```
[/work/gem5] export LINUX_IMAGE=/tmp/linux-arm-ael.img
[/work/gem5] ./build/ARM/gem5.opt configs/example/fs.py --script=./configs/boot/queens.rcS
gem5 Simulator System.  http://gem5.org
…
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
…
Writing checkpoint
info: Entering event queue @ 32358957649500.  Starting simulation...
Exiting @ tick 32358957649500 because m5_exit instruction encountered
```

**Terminal:**

```
[/work/gem5] ./util/term/m5term 127.0.0.1 3456
==== m5 slave terminal: Terminal 0 ====
[    0.000000] Linux version 2.6.38.8-gem5 (saidi@zeep) (gcc version 4.5.2 (Sourcery G++ Lite
[    0.000000] CPU: ARMv7 Processor [350fc000] revision 0 (ARMv7), cr=10c53c7f
…
init started: BusyBox v1.15.3 (2010-05-07 01:27:07 BST)
starting pid 331, tty '': '/etc/rc.d/rc.local'
warning: can't open /etc/mtab: No such file or directory
Thu Jan  1 00:00:02 UTC 1970
S: devpts
Thu Jan  1 00:00:02 UTC 1970
16 queens on a 16x16 board...
 Q - - - - - - - - - - - - - - -
```

# Restoring from Checkpoint

**Command Line:**

```
[/work/gem5] ./build/ARM/gem5.opt configs/example/fs.py  --caches --l2cache \
                                                --cpu-type=arm_detailed   -r 1
…
Switch at curTick count:10000
info: Entering event queue @ 32344924619000.  Starting simulation...
Switched CPUS @ tick 32344924619000
Changing memory mode to timing
switching cpus
**** REAL SIMULATION ****
info: Entering event queue @ 32344924629000.  Starting simulation...
…
Exiting @ tick 32394507487500 because m5_exit instruction encountered
```

**Terminal:**

```
[/work/gem5] ./util/term/m5term 127.0.0.1 3456
==== m5 slave terminal: Terminal 0 ====
16 queens on a 16x16 board...
 Q - - - - - - - - - - - - - - -
 - - Q - - - - - - - - - - - - -
 - - - - Q - - - - - - - - - - -
 - Q - - - - - - - - - - - - - -
 - - - - - - - - - - - - Q - - -
 - - - - - - - - Q - - - - - - -
…
```

# What output is generated?

- Files describing the configuration
  - config.ini – ini formatted file that has all the objects and their parameters
  - config.json – json formatted file which is easy to parse for input into other simulators (e.g. power)
  - config.dot(.pdf) – system layout as a dot graph showing port connections
- Statistics
  - stats.txt – You've seen several examples of this
- Checkpoints
  - cpt.<cycle number> -- Each checkpoint has a cycle number. The –r N parameter restores the Nth checkpoint in the directory
- Output
  - *.terminal – Serial port output from the simulation
  - frames_<system> – Framebuffer output

# DEBUGGING

# 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 DPRINTFs 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 –debug-help
- Enabled on the command line
  - --debug-flags=Exec
  - --trace-start=30000
  - --trace-file=my_trace.out
  - Enable the flag Exec; start at tick 30000; Write to my_trace.out



gem5

# Sample Run with Debugging

## Command Line:

22:44:28 [/work/gem5] ./build/ARM/gem5.opt --debug-flags=Decode--trace-start=50000 --trace-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!
hack: be nice to actually delete the event here
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=0xbeffff0c
51500:      T0 : 0x14644.0        :   ldr   r3, [r0] #8         : MemRead :  D=0x00000011 A=0xbeffff10
52000:      T0 : 0x14644.1        :   addi_uop   r0, r0, #8                : IntAlu :  D=0xbeffff18
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
```



gem5

# 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



gem5

# 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
…
```



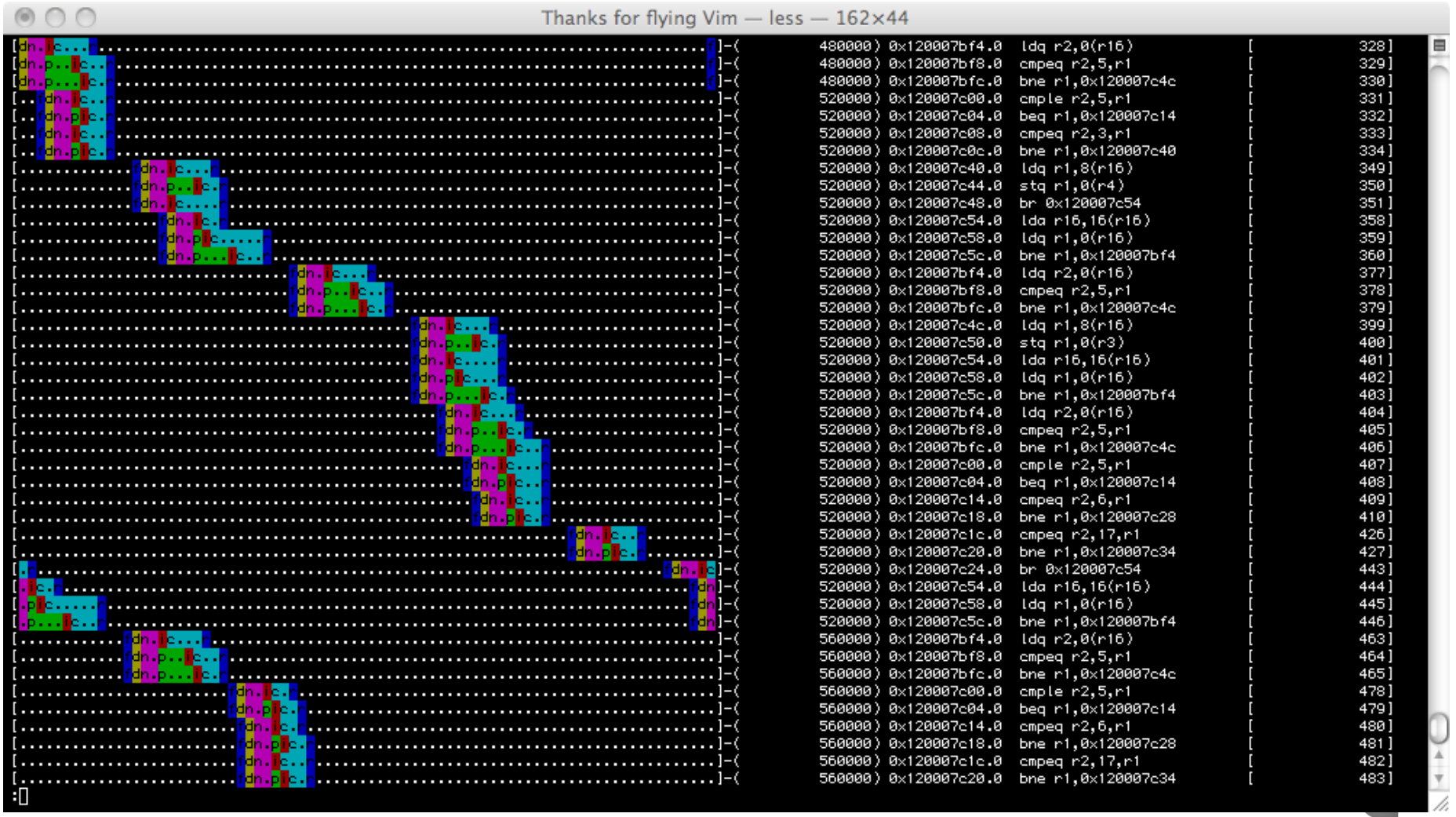
gem5

# Python Debugging

- It is possible to drop into the python interpreter (-i flag)
  - This currently happens after the script file is run
- If you want to do this before objects are instantiated, remove them from script
  - It is possible to drop into the python debugger (--pdb flag)
  - Occurs just before your script is invoked
  - Lets you use the debugger to debug your script code
- Code that enables this stuff is in src/python/m5/main.py
  - At the bottom of the main function
  - Can copy the mechanism directly into your scripts, if in the wrong place for your needs
  - import pdb
  - pdb.set_trace()

# O3 Pipeline Viewer

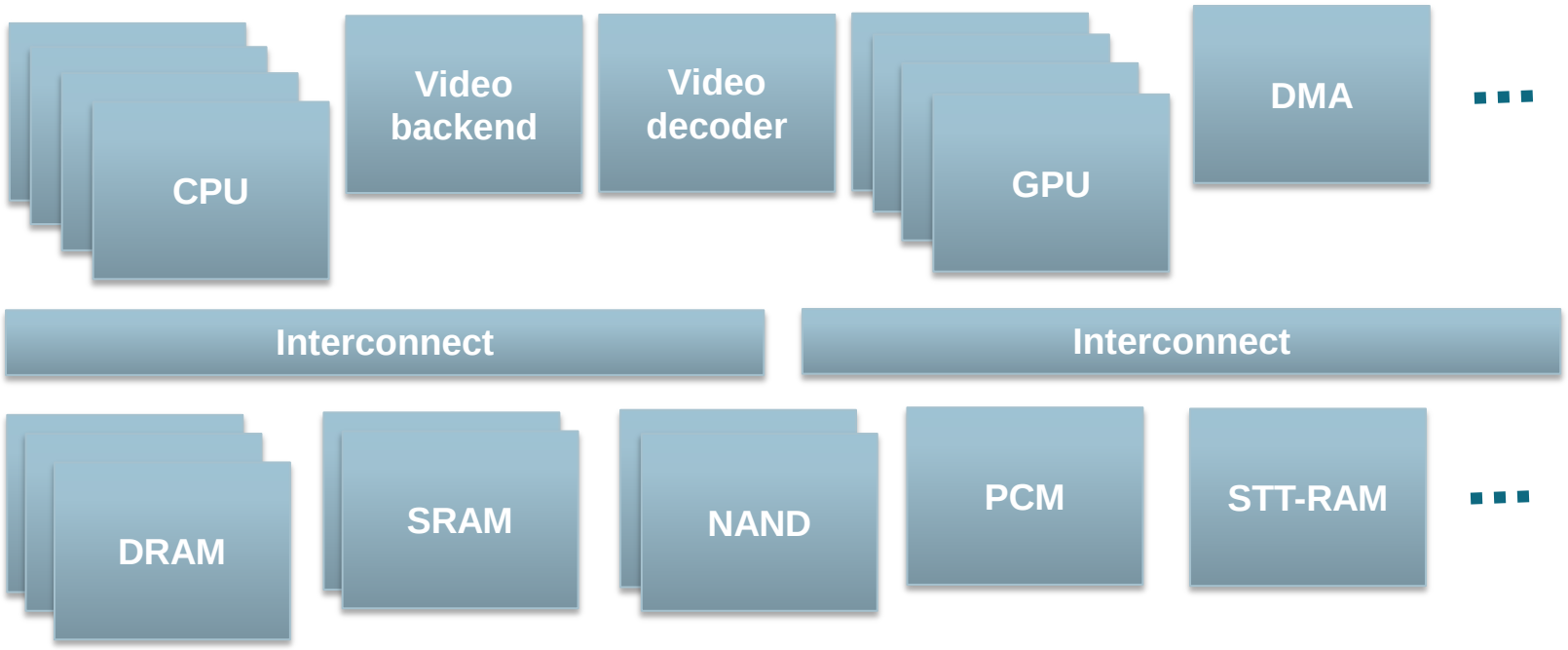
Use --debug-flags=O3PipeView and util/o3-pipeview.py



# MEMORY SYSTEM

# 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

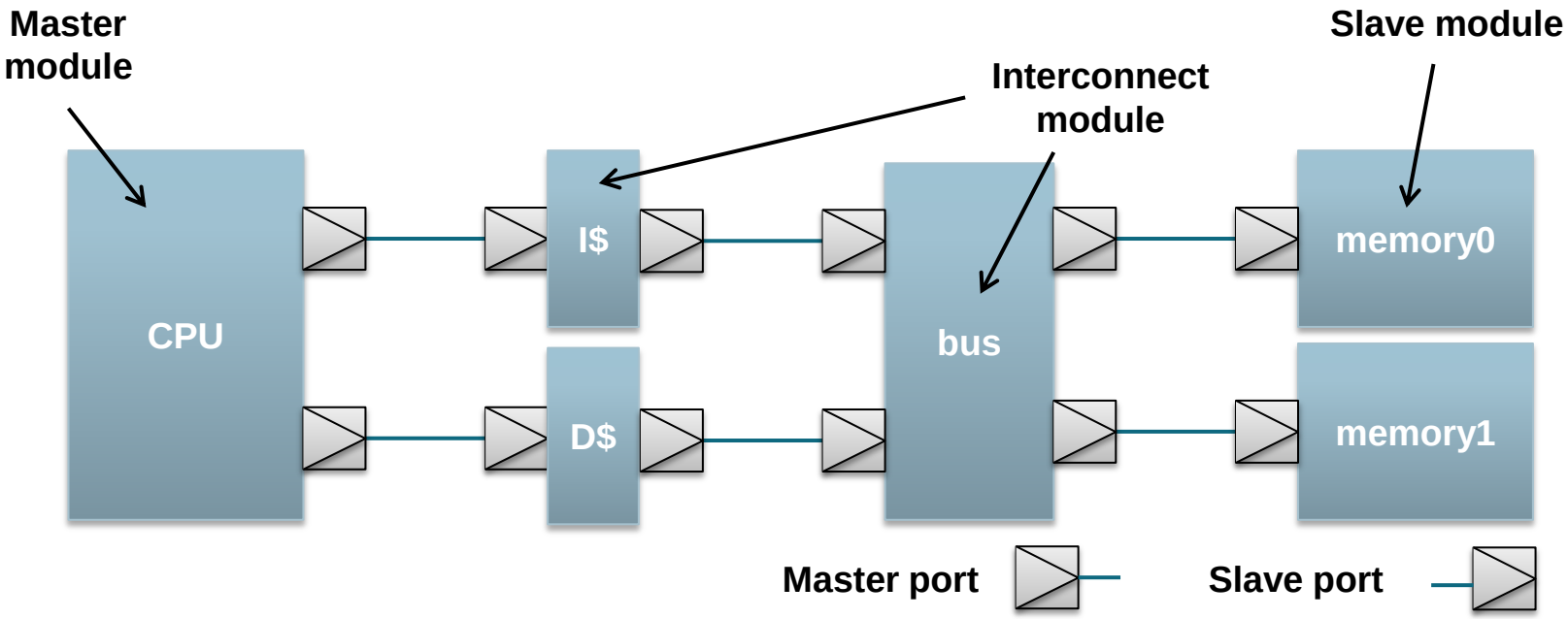


# Goals, contd.

- Two worlds...
  - Computation-centric simulation
    - e.g. SimpleScalar, Simics, 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 striking a 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)

# Ports, Masters and Slaves

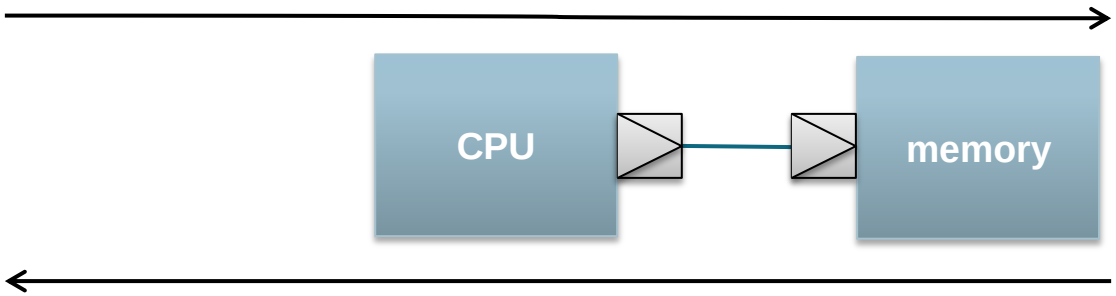
- MemObjects are connected through master and slave ports
- A master module has at least one master port, a slave module at least one slave port, and an interconnect module at least one of each
  - A master port always connects to a slave port
  - Similar to TLM-2 notation



# 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 behind 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 stack
    - Enables adding arbitrary information along packet path

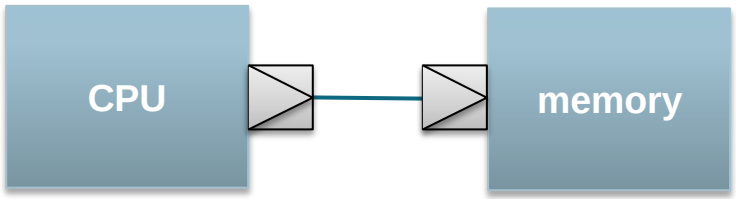
# Functional transport interface

- 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)
{
    ...
```

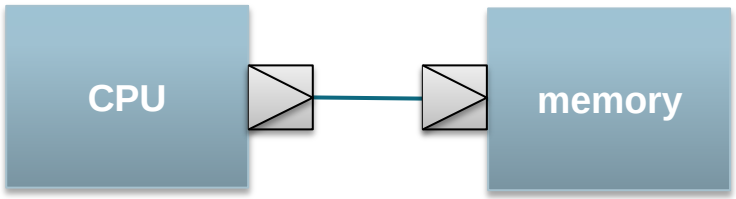


# Atomic transport interface

- 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;
}
```

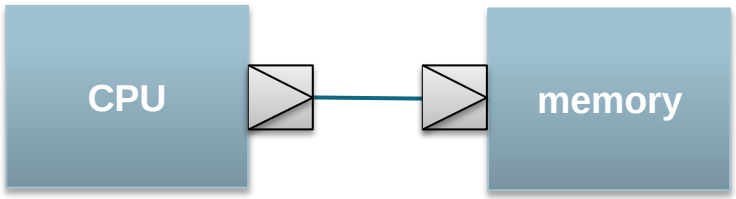


# Timing transport interface

- On a master port we try to send a request packet using sendTiming
- This in turn calls recvTiming on the connected slave port
- For a specific slave port we implement the desired functionality by overloading recvTiming
  - 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 sendRetry to alert the master port to try again

```
bool success = masterPort.sendTiming(pkt);
if (success) {
   // request packet is sent
   ...
} else {
   // failed, will get
   // retry from slave port
   ...
}
```

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

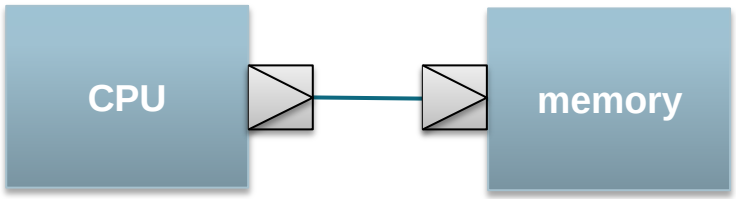


# Timing transport interface (cont'd)

- 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 sendRetry to alert the slave port to try again

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

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



# Ruby for Networks and Coherence

- As an alternative to the conventional 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)
- Runs with Alpha and X86
  - Limited support for functional accesses

# Caches

- Single cache model with several components:
  - Cache: request processing, miss handling, coherence
  - Tags: data storage and replacement (LRU, IIC, 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

# Buses & Bridges

- Create rich system interconnect topologies using a simple bus model and bus bridge
- Buses 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

# Memory

- All memories in the system inherit from AbstractMemory
  - Encapsulates basic “memory behaviour”:
    - Has an address range with a start and size
    - Can perform a zero-time functional access and normal access
- SimpleMemory
  - Multi-port memory controller
  - Fixed-latency memory
  - Fixed bandwidth
- SimpleDRAM
  - Multi-port memory controller
  - Designed to replicate real DRAM controller and DRAM
  - Models read/write queues, scheduling, address mapping, page policy
  - Uses real device parameters
    - tCRD, tCL, tRP etc
  - gem5 ships with DDR3, LPDDR2/3 and WideIO models

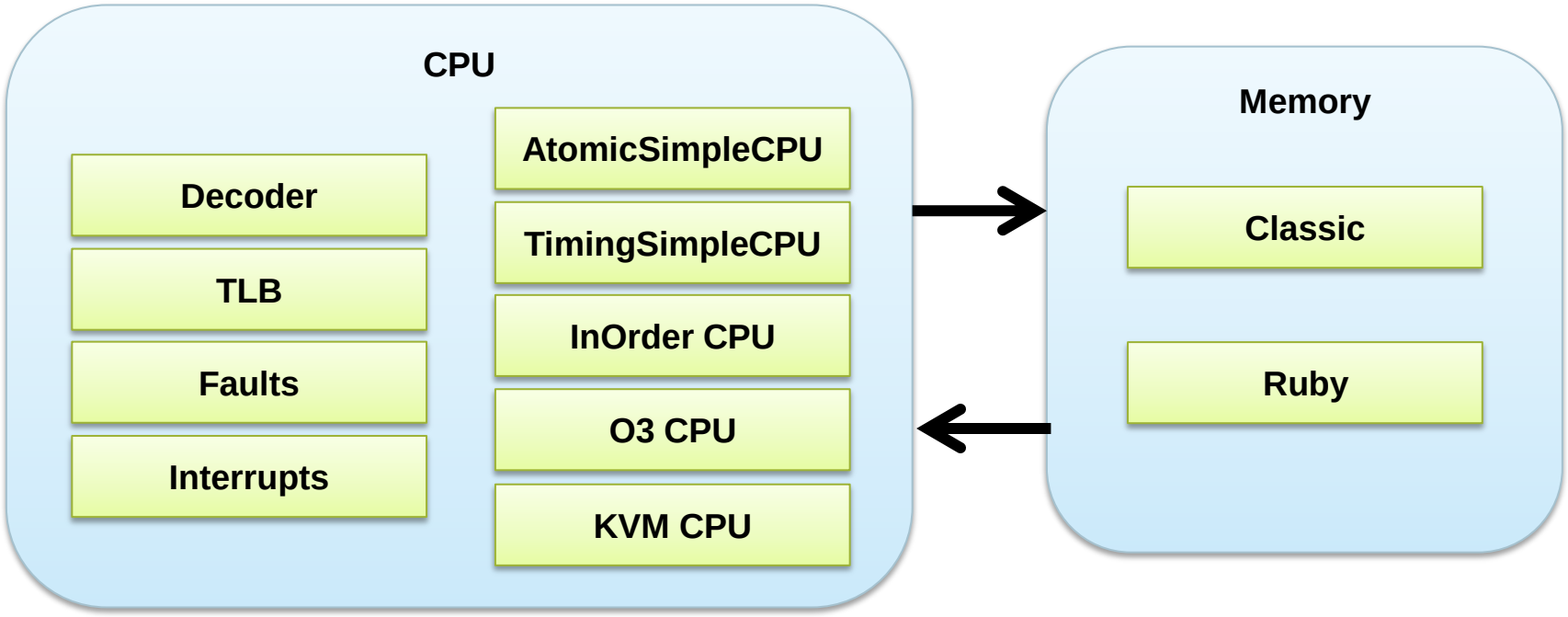
# Instantiating and Connecting Objects

```
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
…
```

# CPU MODELS

# CPU Models – System Level View

- CPU Models are design to be “hot pluggable” with arbitrary ISA and memory systems

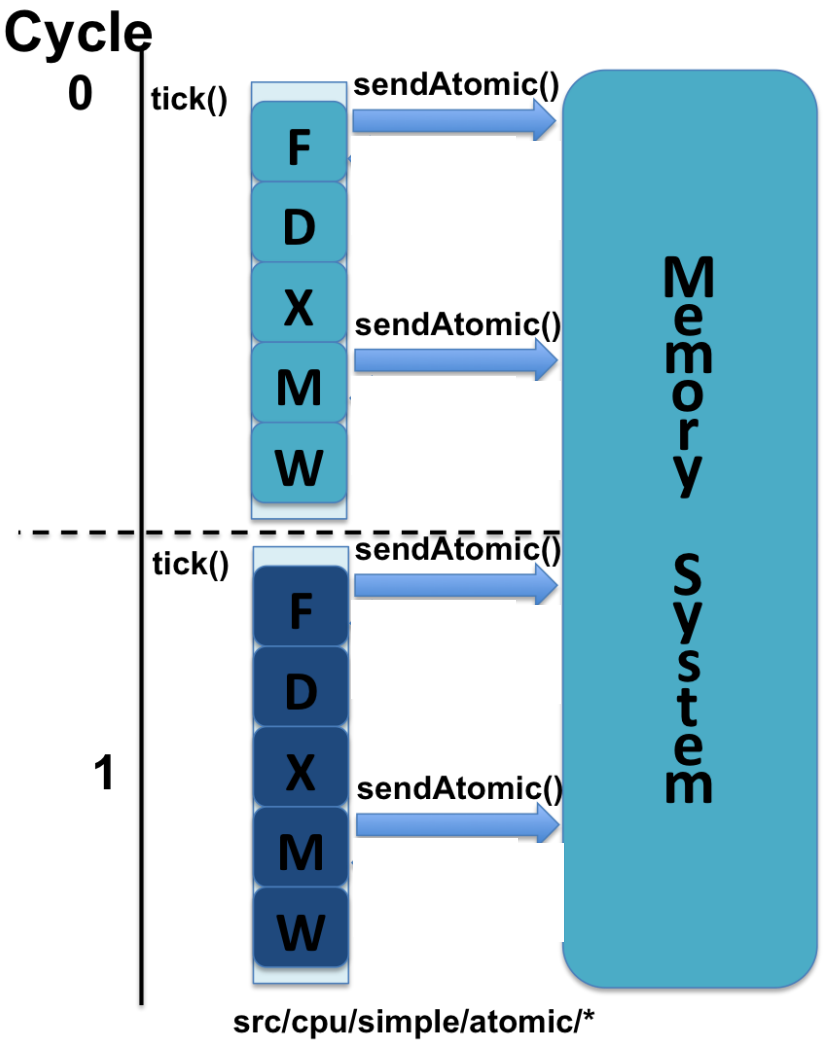


# Simple CPU Models (1)

- Models Single-Thread 1 CPI Machine
- Two Types:
  - AtomicSimpleCPU
  - TimingSimpleCPU
- Common Uses:
  - Fast, Functional Simulation
    - 2.9 million and 1.2 million instructions per second on the twolf benchmark
  - Warming Up Caches
  - Studies that do not require detailed CPU modeling

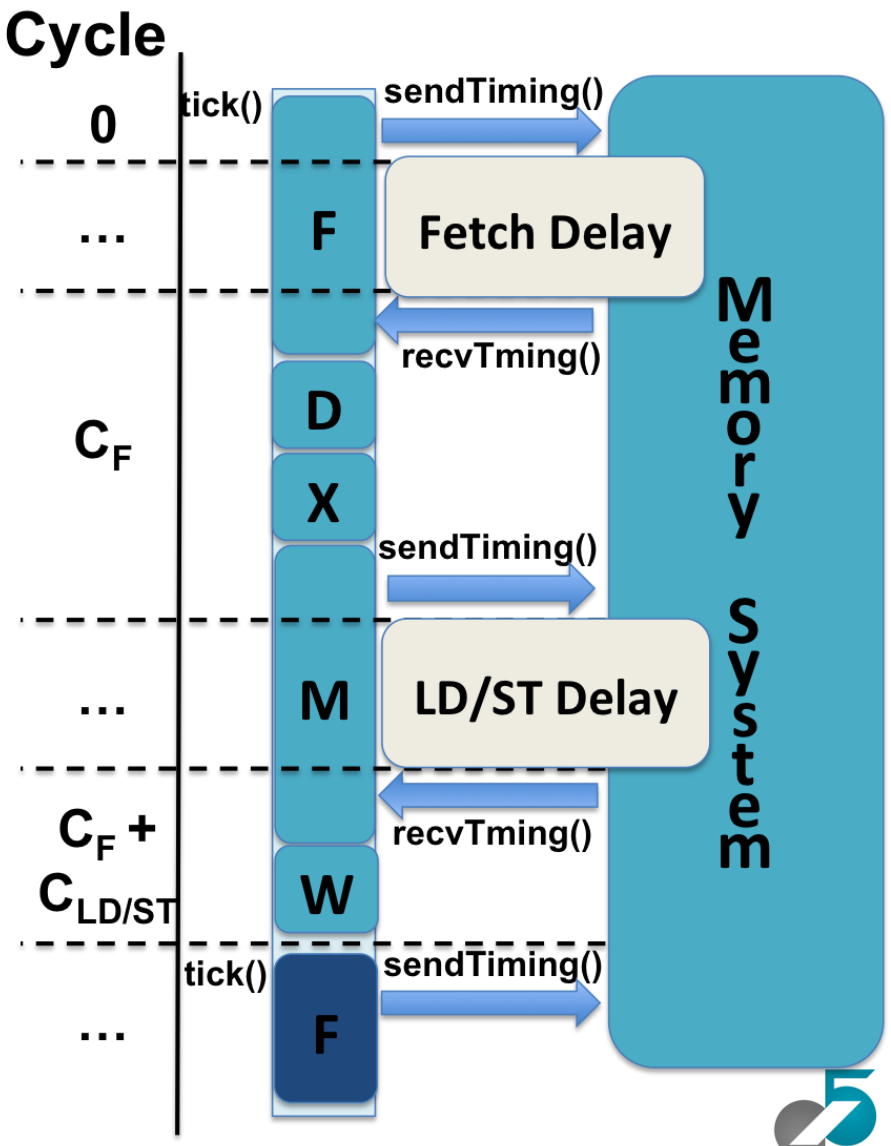
# Atomic Simple CPU

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



# Timing Simple CPU

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



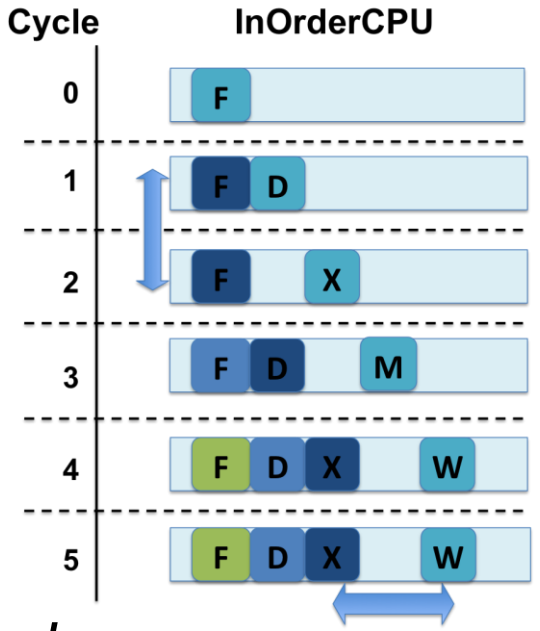

src/cpu/simple/timing/*

# Detailed CPU Models

- Parameterizable Pipeline Models w/SMT support
- Two Types
  - InOrderCPU
  - O3CPU
- “Execute in Execute”, detailed modeling
  - Roughly an order-of-magnitude slower
    - ~200K instructions per second on twolf
  - 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
- Both only support some architecutres
  - See Status Matrix on gem5.org for up-to-date info

# InOrder CPU Model

- Default 5-stage pipeline
  - Fetch, Decode, Execute, Memory, Writeback
- Key Resources
  - Cache, Execution, BranchPredictor, etc.
  - Pipeline stages




- Pipeline stages interact with *Resource Pool*
- Pipeline defined through *Instruction Schedules*
  - Each instruction type defines what resources they need in a particular stage
  - If an instruction can't complete all it's resource requests in one stage, it blocks the pipeline

# Out-of-Order (O3) CPU Model

- Default 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

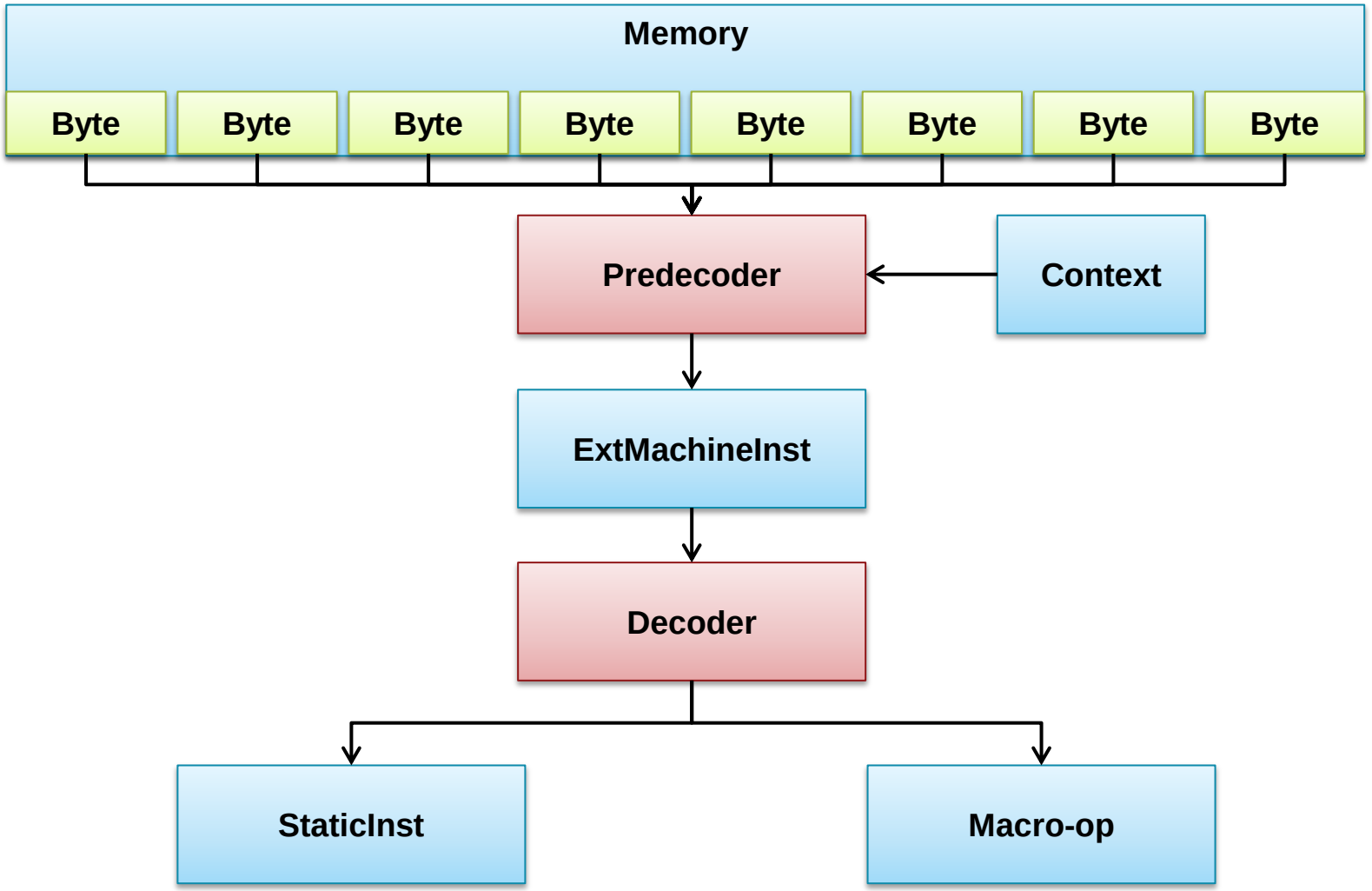
# KVM CPU Model

- Use the host CPU to execute guest instructions natively
- KVM is used to setup a virtual machine
  - gem5 guest memory mapped to allow virtualized CPU to use it
- Massive speedup, even compared to atomic CPU
  - Can interact with simulated system!
- One caveat:
  - Simulation ISA must match host ISA
  - Currently working on ARM ISA

# ThreadContexts

- Interface for accessing total architectural state of a single thread
  - PC, register values, etc.
- Used to obtain pointers to key classes
  - CPU, process, system, ITB, DTB, etc.
- Abstract base class
  - Each CPU model must implement its own derived ThreadContext

# Instruction Decoding



# 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

- Dynamic version of StaticInst
  - Used to hold extra information detailed CPU models
  - BaseDynInst
    - Holds PC, Results, Branch Prediction Status
    - Interface for TLB translations
- Specialized versions for detailed CPU models

# ISA Description Language

- Custom domain-specific language
- Defines decoding & behavior of ISA
- Generates C++ code
  - Scads of StaticInst subclasses
  - decodeInst() function
    - Maps machine instruction to StaticInst instance
  - Multiple scads of execute() methods
    - Cross-product of CPU models and StaticInst subclasses

# COMMON TASKS

# Common Tasks

- Adding a statistic
- Parameters and SimObject
- Creating an SimObject
  - Configuration
  - Initialization
  - Serialization
  - Events
- Instrumenting a benchmark

# Adding a statistic

- Add a statistic to the atomic CPU model
  - Track number of instruction committed in user mode
- Number of statistics classes in gem5
  - Scalar, Average, Vector, Formula, Histogram, Distribution, Vector Dist
- We'll choose a Scalar and a Formula
  - Count number of instructions in user mode
  - Formula to print percentage out of total

# Add Stats to src/cpu/simple/base.hh

```
// statistics
   virtual void regStats();
   virtual void resetStats();

   // number of simulated instructions
   …
   Stats::Scalar numInsts;
   …
   Stats::Scalar numOps;
   …
   Stats::Scalar numUserInsts;
   Stats::Formula percentUserInsts;
```

- **Controls registering the statistics when the simulation starts.**
- **All stats must be registered in regStats() as they can't be dynamically added during the running simulation.**
- **resetStats() is called when the stats are zerod; You normally don't need to do anything for this.**

- **numUserInsts will contain count of instructions executed in user mode**
- **percentUserInsts will be numUserInsts/numInsts**

# Add Stats to src/cpu/simple/base.cc

```
numInsts
    .name(name() + ".committedInsts")
    .desc("Number of instructions committed")
    ;

numUserInsts
    .name(name() + ".committedUserInsts")
    .desc("Number of instructions committed"
          " while in use code")
    ;

percentUserInsts
    .name(name() + ".percentUserInsts")
    .desc("Percent of total of instructions"
          " committed while in use code")
    ;
...
idleFraction = constant(1.0) - notIdleFraction;
percentUserInsts = numUserInsts/numInsts;
```

- **Give the stats we created in the header file a name and a description**
- **Other stat types (e.g. vector) need a length here**

- **Formulas will be evaluated when statistics are output**

# Accumulate numUserInsts

```
void countInst()
  {
     if (!curStaticInst->isMicroop() || curStaticInst->isLastMicroop()) {
        numInst++;
        numInsts++;
        if (TheISA::inUserMode(tc))
           numUserInsts++;
     }
  …
  }
```

# Look at the results

**Command Line:**

[/work/gem5] ./build/ARM/gem5.opt configs/example/fs.py --script=./configs/boot/halt.rcS
gem5 Simulator System.  http://gem5.org
…
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
…
Exiting @ tick 2332316587000because m5_exit instruction encountered

**Stats:**

[/work/gem5]  grep Insts m5out/stats.txt
system.cpu.committedInsts  59262896  # Number of instructions committed
system.cpu.committedUserInsts  6426560  # Number of instructions committed while in user code
system.cpu.percentUserInsts  0.108442  # Percent of instructions committed  while in user code

# Parameters and SimObjects

- Parameters to SimObjects are synthesized from Python structures that represent them
  - This example is from src/dev/arm/Realview.py

```
class Pl011(Uart):
    type = 'Pl011'
    gic = Param.Gic(Parent.any, "Gic to use for interrupting")
    int_num = Param.UInt32("Interrupt number that connects to GIC")
    end_on_eot = Param.Bool(False, "End the simulation when a EOT is received")
    int_delay = Param.Latency("100ns", "Time between action and interrupt generation")
```

**Python class name**

**Python base class**

**C++ class**

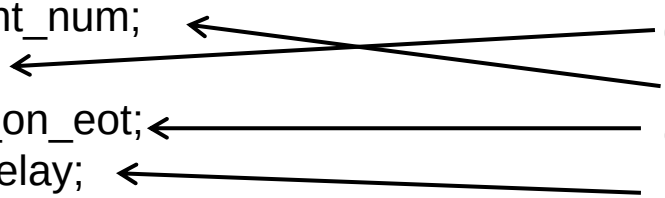
**Parameter type** **Parameter default** **Parameter Description**

**Parameter name**

# 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__
```

```
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), control(0x300), fbrd(0), ibrd(0), lcrh(0), ifls(0x12), imsc(0),
      rawInt(0), maskInt(0), intNum(p->int_num), gic(p->gic),
      endOnEOT(p->end_on_eot), intDelay(p->int_delay), intEvent(this)
{
    pioSize = 0xfff;
}
```

You can also access parameters through params() accessor on SimObject incase you have parameters that aren't stored in a SimObject directly.

# Creating a SimObject

- Derive Python class from Python SimObject
  - Defines 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 SConscript
- 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 function for the object
    - Look at the bottom of an existing object for info
- Recompile

# SimObject Initialization

- SimObjects go through a sequence of initialization

1. C++ object construction
   - Other SimObjects in the system may not be constructed yet
2. SimObject::init()
   - Called on every object before the first simulated cycle
   - Useful place to put initialization that requires other SimObjects
3. SimObject::initState()
   - Called on every SimObject when **not** restoring from a checkpoint
4. SimObject::loadState()
   - Called on every SimObject when restoring from a checkpoint
   - By default the implementation calls SimObject::unserialize()

# 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<ClassName, &ClassName::timerHappend> timerEvent;
```

  - Scheduling them is easy too:

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

# Checkpointing SimObject State

- If you have state that needs to be saved when a checkpoint is created you need to serialize or marshal that data
- When a checkpoint happens SimObject::drain() is called
  - Objects need to return if they're OK to drain or not
  - Should always be OK in atomic mode
  - In timing mode you stop issuing transactions and complete outstanding
- When every object is ok to checkpoint SimObject::serialize()
  - Save necessary state (not parameters you get from config system)
  - SERIALIZE_*() macros help
- To restore the state SimObject::loadState() is called
  - This calls SimObject::unserialize() by default
  - UNSERIALIZE_*() macros

# Checkpointing Timers and Objects

- Checkpointing events, objects are slightly more difficult
  - To checkpoint an object you can use (UN)SERIALIZE_OBJPTR()
    - Save object name
  - To save an event you need to check if it's scheduled

```
bool is_in_event = timerEvent.scheduled();
SERIALIZE_SCALAR(is_in_event);

Tick event_time;
if (is_in_event){
    event_time = timerEvent.when();
    SERIALIZE_SCALAR(event_time);
}
```

# Instrumenting a Benchmark

- You can add instructions that tell simulator to take action inside the binary
  - We went through some examples with checkpointing and stats reset

- Other options are
  - m5_initparam() – get integer passed on command line –initparam=
  - m5_reset_stats(initial_delay, repeat) – reset the stats to 0
  - m5_dump_stats(initial_delay, repeat) – dump stats to text file
  - m5_work_begin(work_id, thread_id)  -- begin a item sample
  - m5_work_end (work_id, thread_id)  -- end a item sample
    - Average time complete work_ids will be printed in stats file

# CONFIGURATION

# Simulator Configuration

- Config files that come with gem5 are meant to be examples
  - Certainly not meant to expose every parameter you would want to change
- Configuration files are Python
  - You can programmatically create objects
  - Put them into a hierarchy
  - gem5 will instantiate all the Python SimObjects you create and attach them together
- Good news is you can do anything you want for configuration
  - Possibly also bad news

# SimObject Parameters

- Parameters can be
  - Scalars – Param.Unsigned(5), Param.Float(5.0)
  - Arrays -- VectorParam.Unsigned([1,1,2,3])
  - SimObjects – Param.PhysicalMemory(…)
  - Arrays of SimObjects – VectorParam.PhysicalMemory(Parent.any)
  - Range – Param.Range(AddrRange(0,Addr.max))
- Some are converted from strings:
  - Latency – Param.Latency('15ns')
  - Frequency – Param.Frequency('100MHz')
- Others are converted to bytes
  - MemorySize – Param.MemorySize('1GB')
- Few more complex types:
  - Time – Param.Time('Mon Mar 25 09:00:00 CST 2012')
  - Ethernet Address – Param.EthernetAddr("90:00:AC:42:45:00")



gem5

# A Simple Example

```
import m5
from m5.objects import *
```

```
class MyCache(BaseCache):
    assoc = 2
    block_size = 64
    latency = '1ns'
    mshrs = 10
    tgts_per_mshr = 5
```

```
class MyL1Cache(MyCache):
    is_top_level = True
```

```
cpu = TimingSimpleCPU(cpu_id=0)
cpu.addTwoLevelCacheHierarchy(MyL1Cache(size = '128kB'),
                              MyL1Cache(size = '256kB'),
                              MyCache(size = '2MB', latency='10ns'))
```

```
system = System(cpu = cpu,
                physmem =SimpleMemory(),
                membus = Bus())
```



gem5

# A Simple Example Part 2

```
root.system.cpu.workload = LiveProcess(cmd = 'hello', executable = binpath('hello'))
```

```
system.system_port = system.membus.slave
system.physmem.port = system.membus.master
```

```
# create the interrupt controller
cpu.createInterruptController()
cpu.connectAllPorts(system.membus)
cpu.clock = '2GHz'
```

```
root = Root(full_system=False, system = system)
```

```
# instantiate configuration
m5.instantiate()
```

```
# simulate until program terminates
exit_event = m5.simulate(m5.MaxTick)
```

# Two Classes of Configuration

- Python files in the src directory are “built” into the executable
  - If you change one of these you need to recompile
  - Or, set the M5_OVERRIDE_PY_SOURCE env variable to True
- Other python files aren’t built into the binary
  - They can be changed and no recompiling is needed

# CONCLUSION

# Summary

- Basics of using gem5
  - High-level features
  - Running simulations
  - Debugging them
- Under the hood
  - Memory system
  - CPU Models
- Common Tasks
  - Adding a statistics
  - SimObject Parameters
  - Creating a SimObject
  - Instrumenting a Benchmark

# Keep in Touch

- Please check out the website:
  - Subscribe to the mailing lists
    - gem5-users – Questions about using/running gem5
    - gem5-dev – Questions about modifying the simulator
  - Submit a patch to our ReviewBoard
    - http://reviews.gem5.org
  - Read & Contribute to the wiki
    - http://www.gem5.org
- We hope you found this tutorial and will find gem5 useful
- We'd love to work with you to make gem5 more useful to the community
- Thank you