# COEN-4730 Computer Architecture
# Info on GEM5, McPAT, and More

Cristinel Ababei
Dept. of Electrical and Computer Engineering
Marquette University



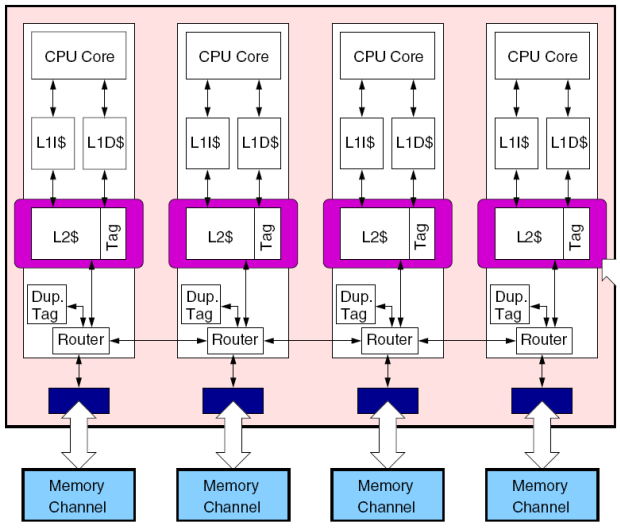
## Resources

- **GEM5 Resources**
  - http://www.m5sim.org/Tutorials
  - http://www.m5sim.org/Documentation
  - http://www.m5sim.org/wiki/index.php/Tutorials
  - http://www.gem5.org/dist/tutorials/isca_pres_2011.pdf
  - http://gem5.org/dist/tutorials/hipeac2012/gem5_hipeac.pdf
  - http://www.m5sim.org/Ruby
- **GEMS Resources**
  - http://research.cs.wisc.edu/gems/
  - http://lists.cs.wisc.edu/mailman/listinfo/gems-users
- **You can find (some) answers at:**
  - http://blog.gmane.org/gmane.comp.emulators.m5.users
  - http://www.mail-archive.com/gem5-users@gem5.org/
  - http://www.m5sim.org/Frequently_Asked_Questions
  - http://qa.gem5.org/
- **Check periodically the status matrix of GEM5:**
  - http://www.m5sim.org/Status_Matrix

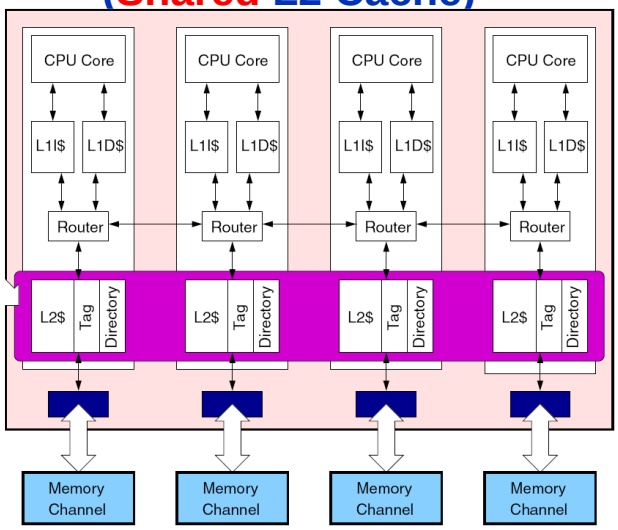
- **Refresher**



# CMP Cache Organizations (Private L2 Cache)



# CMP Cache Organizations (Shared L2 Cache)




5

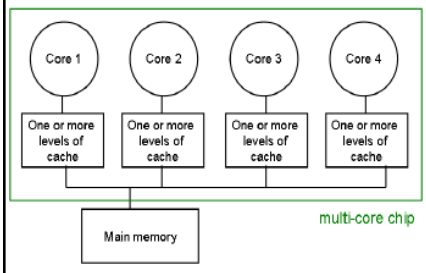
# CMP Cache Coherence

1. **Snoop based:**
   - **All caches on the bus snoop the bus to determine if they have a copy of the block of data that is requested on the bus. Multiple copies of a data block can be read without any coherence problems; however, a processor must have exclusive access (either invalidate or update other copies) to the bus in order to write.**
   - **Enough for small-scale CMPs with bus interconnection**
2. **Directory based**
   - **the data being shared is tracked in a common directory that maintains the coherence between caches. When a cache line is changed the directory either updates or invalidates the other caches with that cache line.**
   - **Necessary for many-core CMPs with such interconnection as mesh**

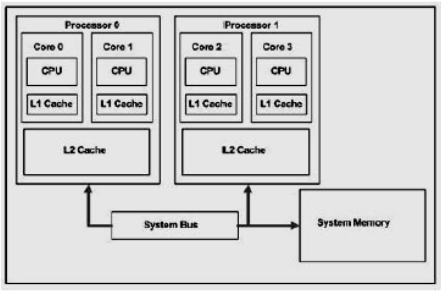
6

# Multicore vs. Multiprocessor

Multicore




single physical processor contains the 4 Logic core




Dual –core dual-processor system

7

- GEM5 = M5 + GEMS (i.e., mainly Ruby part of it)

8

## GEM5 = M5 + GEMS (i.e., Ruby only)

- **GEM5**
  - **Supports both functional and timing simulation**
  - **Has two simulation modes: full-system (FS) and syscall emulation (SE)**
  - **Supports multiple ISAs**
    - » ALPHA: well-developed to support both FS and SE modes
  - **It models**
    - » Processor Cores + Memory Hierarchy + I/O Systems
  - **Written by using C++, Python & Swig, and totally open-source**
- **More things about M5**
  - **http://www.m5sim.org/wiki/index.php/Main_Page**
  - **The most important document: http://www.m5sim.org/wiki/index.php/Tutorials**

9

## Capabilities

- **Execution modes**: System-call Emulation (SE) & Full-System (FS)
- **ISAs**: Alpha, ARM, MIPS, Power, SPARC, x86
- **CPU models**: AtomicSimple, TimingSimple, InOrder, and O3
- **Cache coherence protocols**: broadcast-based, directories, etc.
- **Interconnection networks**: Simple & Garnet (Princeton, MIT)
- **Devices**: NICs, IDE controller, etc.
- **Multiple systems**: communicate over TCP/IP

10

## 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

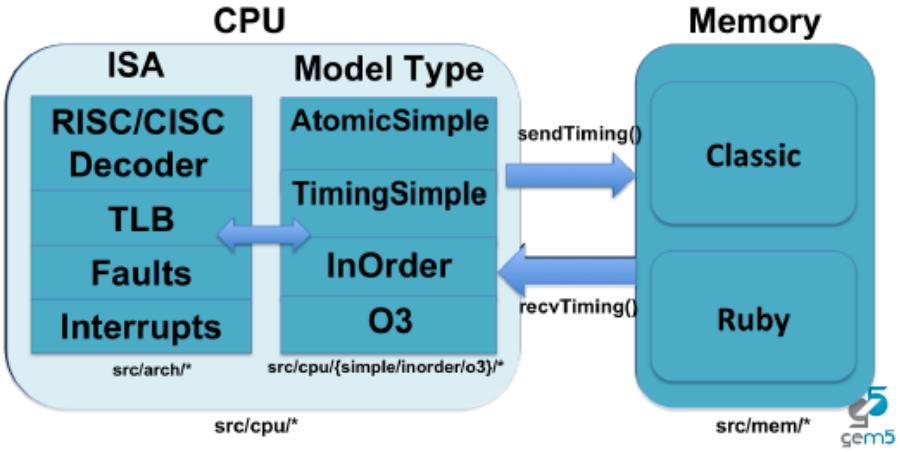
11

## CPU Models Overview

- **Supported CPU Models**
  - **AtomicSimpleCPU**
  - **TimingSimpleCPU**
  - **InOrderCPU**
  - **O3CPU**
- **CPU Model Internals**
  - **Parameters**
  - **Time Buffers**
  - **Key Interfaces**

## CPU Models - System Level View

CPU Models are designed to be "hot pluggable" with arbitrary ISAs and Memory Systems

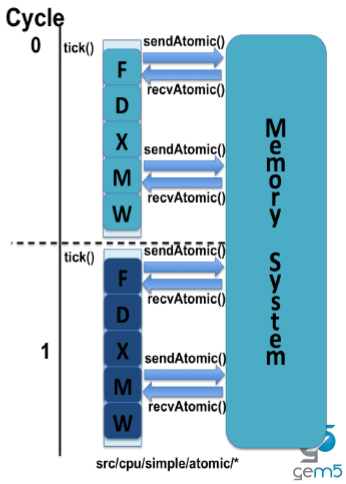



## Supported CPU Models

- **Simple CPUs**
  - Models Single-Thread 1 CPI Machine
  - Two Types: AtomicSimpleCPU and 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
- **Detailed CPUs**
  - Parameterizable Pipeline Models w/SMT support
  - Two Types: InOrderCPU and O3CPU
  - "Execute in Execute", detailed modeling
  - Slower than SimpleCPUs: 200K instructions per second on the "twolf" benchmark
    - » 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.

## AtomicSimpleCPU `src/cpu/simple/atomic/*.hh,cc`

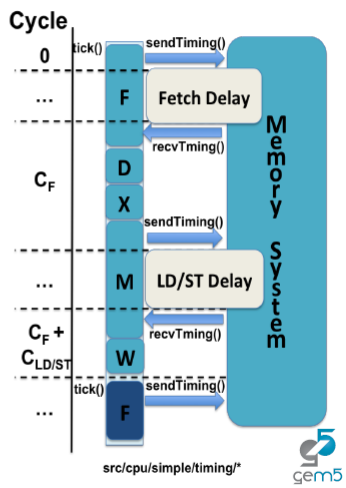
- On every CPU "tick()", perform all necessary operations for an instruction
- Memory accesses are atomic
- Fastest functional simulation






## TimingSimpleCPU `src/cpu/simple/timing/*.hh,cc`

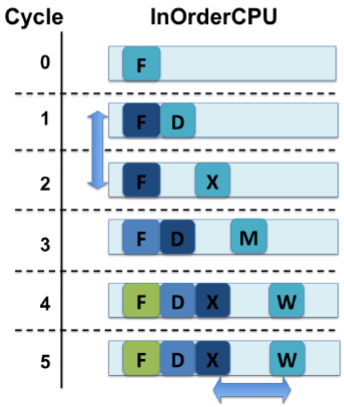
- Memory accesses use timing path
- CPU waits until memory access returns
- Fast, provides some level of timing



## InOrder CPU Model

`src/cpu/inorder/*.hh,cc`

- Detailed in-order CPU
- *InOrder* is a new feature to the gem5 Simulator
  - Default 5-stage pipeline
    - Fetch, Decode, Execute, Memory, Writeback






## InOrder CPU Model

`src/cpu/inorder/*.hh,cc`

- Detailed in-order CPU
  - Default 5-stage pipeline
    - Fetch, Decode, Execute, Memory, Writeback
  - Key Resources
    - CacheUnit, ExecutionUnit, BranchPredictor, etc.
  - Key Parameters
    - Pipeline Stages, Hardware Threads
- Implementation: Customizable Set of Pipeline Components
  - 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

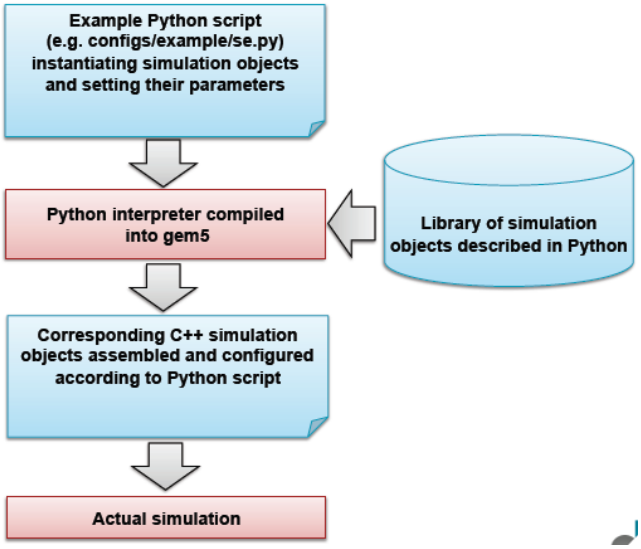
## O3 CPU Model

`src/cpu/o3/*.hh,cc`

- Detailed out-of-order CPU
  - Default 7-stage pipeline
    - Fetch, Decode, Rename, IEW,Commit
    - IEW <=>Issue, Execute, and Writeback
    - Model varying amount of pipeline stages by changing delays between pipeline stages (e.g. fetchToDecodeDelay)
  - Key Resources
    - Physical Register (PR) File, IQ, LSQ, ROB, Functional Unit (FU) Pool
  - Key Parameters
    - Interstage pipeline delays, Hardware threads, IQ/LSQ/ROB/PR entries, FU Delays
  - Other Key Features
    - Support for CISC decoding (e.g. x86)
    - Renaming with a Physical Register (PR) File
    - Functional units with varying latencies
    - Branch Prediction
    - Memory dependence prediction




## Sample Run – Behind the scenes



# What output is generated?

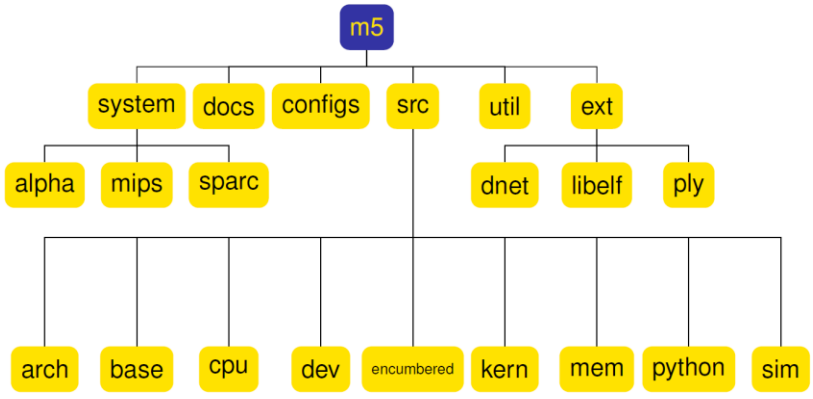
- Files describing the configuration
  - config.ini – ini formatted file that has all the objects and their parameters
  - config.json – json formatted file which is easy to parse for input into other simulators (e.g. power)
- Statistics
  - stats.txt – You've seen several examples of this
- Checkpoints
  - 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

- **Read more at:**
  - http://gem5.org/Configuration_/_Simulation_Scripts

21

# Introduction to M5

- **M5's Source Tree Structure**

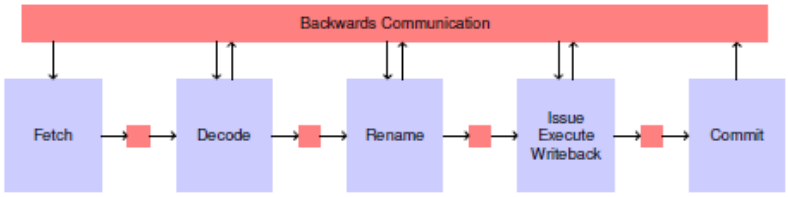



22

# Introduction to M5

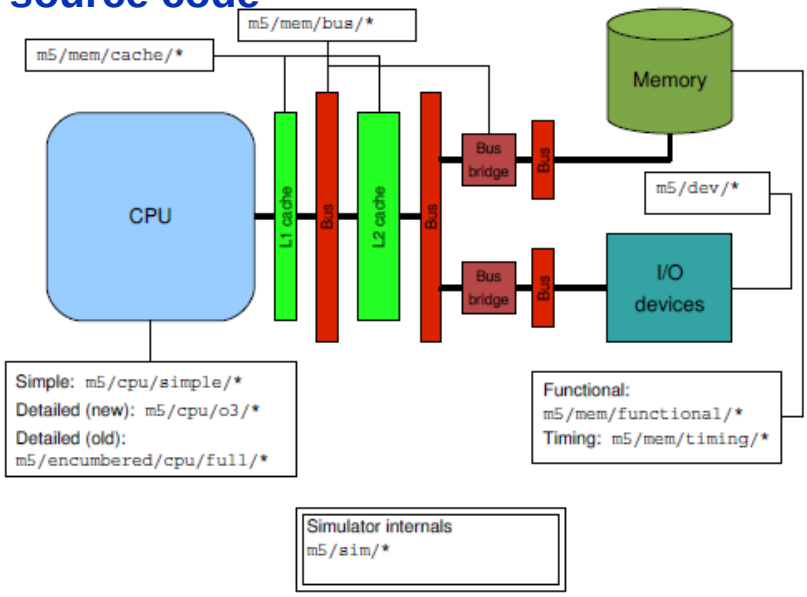
- **CPU Modeled by M5**
  - **SimpleCPU**
  - **TimingCPU**
  - **O3CPU**

- Demonstrated on out-of-order pipeline ...
  - Red is a time buffer






# Overview of M5 with references to source code

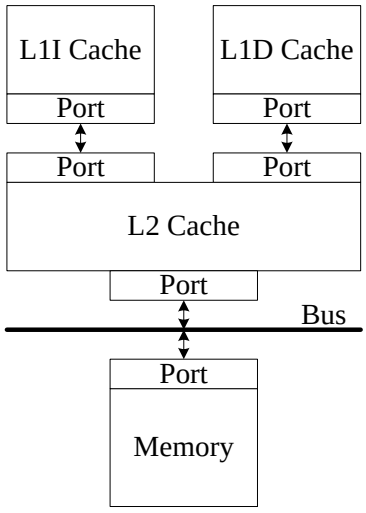


## Inside Gem5: Memory Model

- General Memory System
  - Ports
  - Packets
  - Requests
  - Atomic/Timing/Functional accesses
- Two memory system models
  1. Classic
  2. Ruby

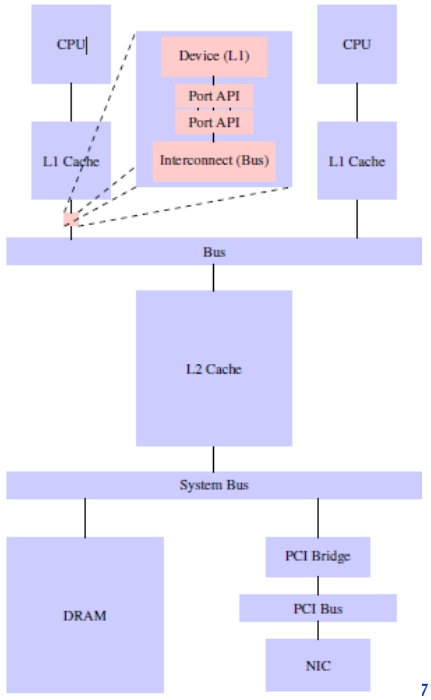
## 1: Classic Memory System

- Classic Memory Hierarchy Modeled by M5, the first of the two memory models supported by GEM5
- Read more at:
  - http://www.m5sim.org/Classic_Memory_System
- In classic memory system coherence is a MOESI snooping protocol
- You do not need to run GEM5 with –ruby option in this case




26

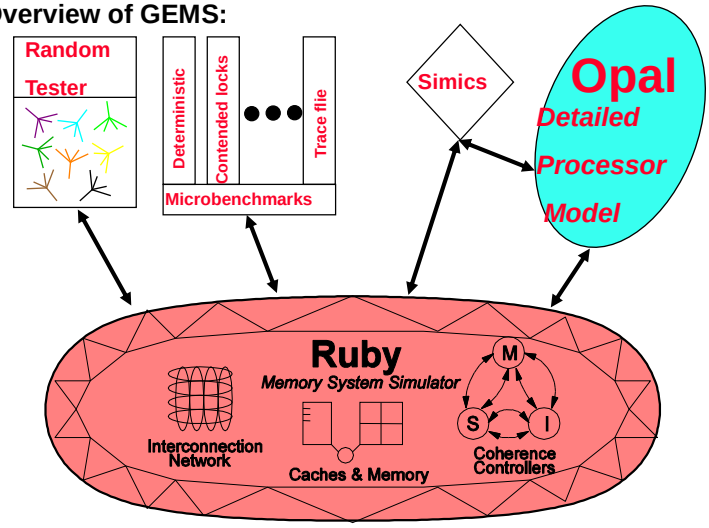
# Example of possible system hierarchy



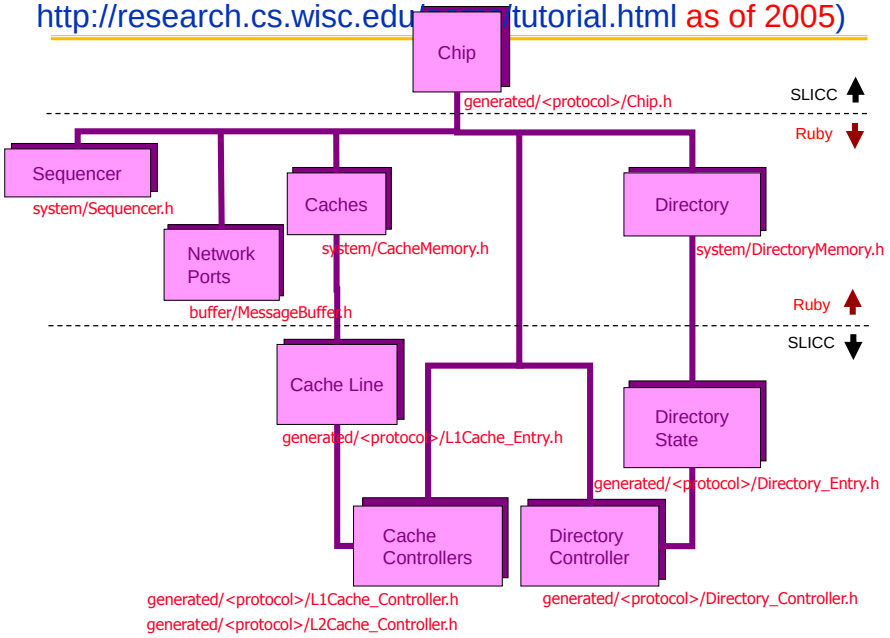

7

# 2: RUBY memory model

- GEMS includes RUBY, the second of the two memory models supported by GEM5
- An Overview of GEMS:



## Ruby Software Structure (within GEMS software framework: http://research.cs.wisc.edu/.../tutorial.html as of 2005)




## Ruby Memory Model

- **Flexible Memory System**
  - **Rich configuration - Just run it**
    - » **Simulate combinations of caches, coherence, interconnect, etc...**
  - **Rapid prototyping - Just create it**
    - » **Domain-Specific Language (SLICC) for coherence protocols**
    - » **Modular components**
- **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)**
  - **Memory (DDR2)**

## Ruby Memory Model

- **Can build many different memory systems**
  - **CMPs, SMPs, SCMPs**
  - **1/2/3 level caches**
  - **Pt2Pt/Torus/Mesh Topologies**
  - **MESI/MOESI coherence**
- **Each component is individually configurable**
  - **Build heterogeneous cache architectures (new)**
  - **Adjust cache sizes, bandwidth, link latencies, etc...**

## Configuration Examples

1. 8 core CMP, 2-Level, MESI protocol, 32K L1s, 8MB 8-banked L2s, crossbar interconnect
   - `scons build/ALPHA_FS/gem5.opt PROTOCOL=MESI_CMP_directory RUBY=True`
   - `./build/ALPHA_FS/gem5.opt configs/example/ruby_fs.py -n 8 --l1i_size=32kB --l1d_size=32kB --l2_size=8MB --num-l2caches=8 --topology=Crossbar --timing`
2. 64 socket SMP, 2-Level on-chip Caches, MOESI protocol, 32K L1s, 8MB L2 per chip, mesh interconnect
   - `scons build/ALPHA_FS/gem5.opt PROTOCOL=MOESI_CMP_directory RUBY=True`
   - `./build/ALPHA_FS/m5.opt configs/example/ruby_fs.py -n 64 --l1i_size=32kB --l1d_size=32kB --l2_size=512MB --num-l2caches=64 --topology=Mesh --timing`

- Many other configuration options
- Protocols only work with specific architectures (see wiki)

## Ruby Memory Model

- Domain-Specific Language
  - Syntatically similar to C/C++
  - Like HDLs, constrains operations to be hardware-like (e.g., no loops)
- Two generation targets
  - C++ for simulation
    - » Coherence controller object
  - HTML for documentation
    - » Table-driven specification (State x Event -> Actions & next state)

## 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
- **Read more at:**
  - http://www.m5sim.org/Ruby

34

## Introduction to Ruby

- **Essential Components in Ruby**
  - **(1) Caches & (2) Memory**
  - **(3) Coherence Protocols**
- The following cache coherence protocols are supported:
  - **MI_example**: example protocol, 1-level cache.
  - **MESI_CMP_directory**: **single chip, 2-level caches, strictly-inclusive hierarchy.**
  - **MOESI_CMP_directory**: **multiple chips, 2-level caches, non-inclusive (neither strictly inclusive nor exclusive) hierarchy.**
  - **MOESI_CMP_token**: 2-level caches. TODO.
  - **MOESI_hammer**: single chip, 2-level private caches, strictly-exclusive hierarchy.
  - **Network_test**: dummy protocol to operate the network tester, 1-level cache.

35

## Introduction to Ruby

- **Essential Components in Ruby**
  - **(4) Interconnection Networks**
    - » **Either be automatically generated by default**
      - Intra-chip network: Single on-chip switch
      - Inter-chip network: 4 included
    - » **Or be customized by users**
      - Defined in *_FILE_SPECIFIED.txt under the directory "$GEMS_ROOT_DIR/ruby/network/simple/Network_Files "

36

## 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)

37

## 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

38

## Detailed Component Simulation: Caches

- Set-Associative Caches
- Each CacheMemory object represents one *bank* of cache
- Configurable bit select for indexing
- Modular replacement policy
  - Tree-based pseudo-LRU
  - LRU
- See `src/mem/ruby/system/CacheMemory.hh`

39

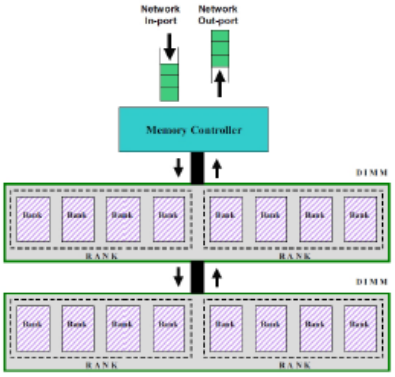
## Memory

- 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 is currently the only subclass
  - Multi-port memory controller
  - Fixed-latency memory (possibly with a variance)
  - Infinite throughput

40

- **Read more at:**
  - http://www.m5sim.org/Coherence-Protocol-Independent_Memory_Components

## Detailed Component Simulation: Memory



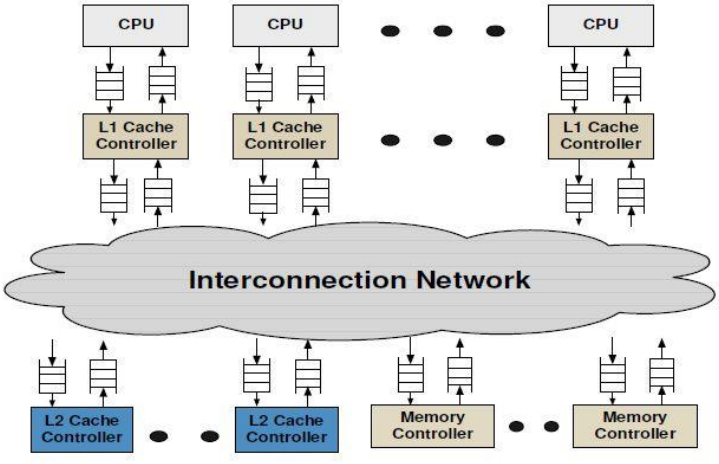

- Memory controller models a **single channel** DDR2 controller
- Implements closed-page policy
- Can configure ranks, tCAS, refresh, etc..
- See `src/mem/ruby/system/MemoryController.hh`

gem5

41

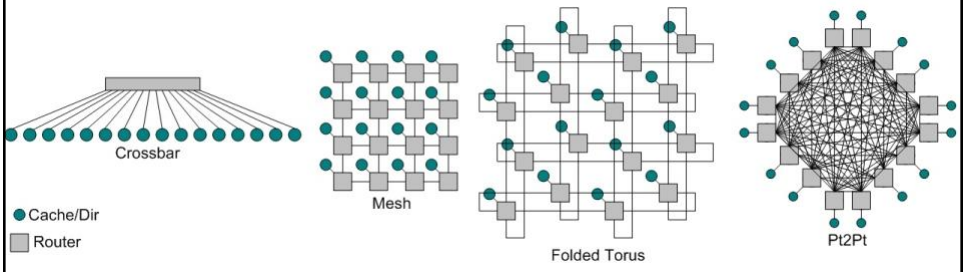
## Interconnection Network

- **Read more at:**
  - http://www.m5sim.org/Interconnection_Network




42

# Network Topologies



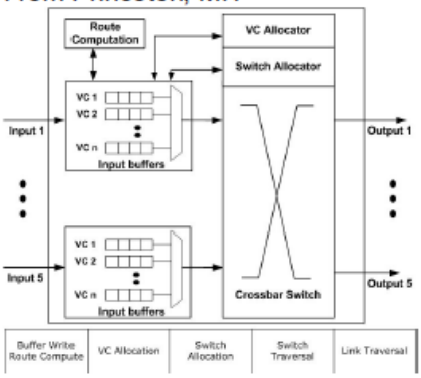



# Topology Parameters

- **Link latency**
  - » **Auto-generated**
    - ON_CHIP_LINK_LATENCY
    - NETWORK_LINK_LATENCY
  - » **Customized**
    - 'link_latency:'
- **Link bandwidth**
  - » **Auto-generated**
    - On-chip = 10 x g_endpoint_bandwidth
    - Off-chip = g_endpoint_bandwidth
  - » **Customized**
    - Individual link bandwidth = 'bw_multiplier:' x g_endpoint_bandwidth
- **Buffer size**
  - » **Infinite by default**
  - » **Customized network supports finite buffering**
    - Prevent 2D-mesh network deadlock through e-cube restrictive routing
    - 'link_weight'
- **Perfect switch bandwidth**

## Detailed Component Simulation: Network

- Simple Network
  - Idealized routers - fixed latency, no internal resources
  - **Does** model link bandwidth
- Garnet Network
  - Detailed routers - both fixed and flexible pipeline model
  - From Princeton, MIT




45

## Run PARSEC 2.1 under FS mode

- PARSEC (from Texas) related:
  - Download Parsec benchmarks from here:
    - » http://www.cs.utexas.edu/~cart/parsec_m5/
  - If you generate running scripts .rcS using the writescripts.pl, you should change the script or remove from each generated .rcS script the line containing switchcpu
  - Read here for more on those m5ops:
    - » http://www.m5sim.org/M5ops
- http://www.cs.utexas.edu/~parsec_m5/
- http://www.m5sim.org/PARSEC_benchmarks

# Run SPLASH2 under FS mode

- Preparation: put your code into the image

```
sudo mount –o loop,offset=32256 linux-latest.img /mnt
sudo mkdir –p /mnt/benchmark/mybench
sudo cp FFT /mnt/benchmark/mybench
sudo umount /mnt
```

- Run

```
scons build/ALPHA/gem5.opt
./build/ALPHA/gem5.opt configs/example/fs.py

m5term 3456
./FFT -t
```

# Run SPLASH2 under FS mode

- more convenient way?

```
vi configs/common/Benchmarks.py
+ 'fft':          [SysConfig('fft.rcS', '512MB')],

vi configs/boot/ffs.rcS
+ #!/bin/sh
+ cd benchmarks/mybench
+ echo "Running FFT now…"
+ ./FFT –t –p1
+ /sbin/m5 exit
```

- Run

```
scons build/ALPHA/gem5.opt
./build/ALPHA/gem5.opt configs/example/fs.py –n 1 –b fft

cat m5out/system.terminal
```

## Do not forget…

- If you use Ruby (Caches and Garnet networks-on-chip such as Mesh or Crossbar) models you must work with ALPHA and X86; because only those are supported now?
- If you use Ruby memory model (especially if utilized with any of the Garnet networks), it is slower than the Classic memory model supported by GEM5
  1. Classic: http://www.m5sim.org/Classic_Memory_System
  2. Ruby: http://www.m5sim.org/Ruby
- To read about interconnects and their .py scripts:
  - http://www.m5sim.org/Interconnection_Network

49

## Do not forget…

- Mesh network:
  - This topology requires the number of directories to be equal to the number of CPUs.
  - It can be invoked from command line by --topology=Mesh.
  - The number of routers/switches is equal to the number of CPUs in the system.
  - The number of rows in the mesh has to be specified by --mesh-rows.
  - Each router/switch is connected to one L1, one L2 (if present), and one Directory.

50

## Do not forget…

- **How many CPUs can M5 run?**
- There is no inherent limit in M5 (other than simulation speed). In SE mode there are no obstacles to simulating as many CPUs as you want. However, in FS mode, the real-world Alpha platform we model (Tsunami) only supports up to 4 processors. To get around this limit, we defined and implemented a variant of the Tsunami platform (which we call BigTsunami) that can take up to 64 processors. Note that BigTsunami does not correspond to any real system. BigTsunami support is included in the standard M5 Alpha build, **but booting with more than 4 CPUs requires modifications to the PAL code and kernel as well**. Take a look at the Download page for our Linux patches and modified PAL code. Note that even with the BigTsunami changes, simulating 64 processors will be quite slow, and the Linux scheduler doesn't seem particularly good at scheduling a large number of processors.
- Prebuilt kernel and PAL binaries can be found at: http://www.cs.utexas.edu/~cart/parsec_m5/