Αρχιτεκτονική Υπολογιστών

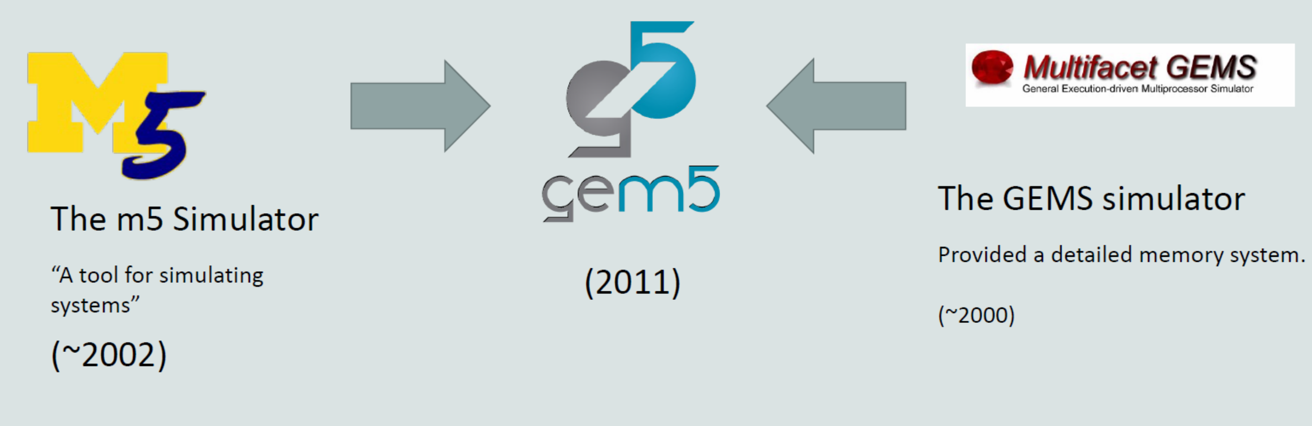
Εισαγωγή στο προσομοιωτή Gem5

Άγγελος Αθανασιάδης

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

Introduction

- A full-system computer architecture simulator
  - Open source tool focused on architectural modeling
  - BSD license
- Encompasses
  - system-level architecture, as well as
  - processor micro-architecture
- The gem5 simulation infrastructure is the merger of
  - The best aspects of the M5 and
  - The best aspects of GEMS



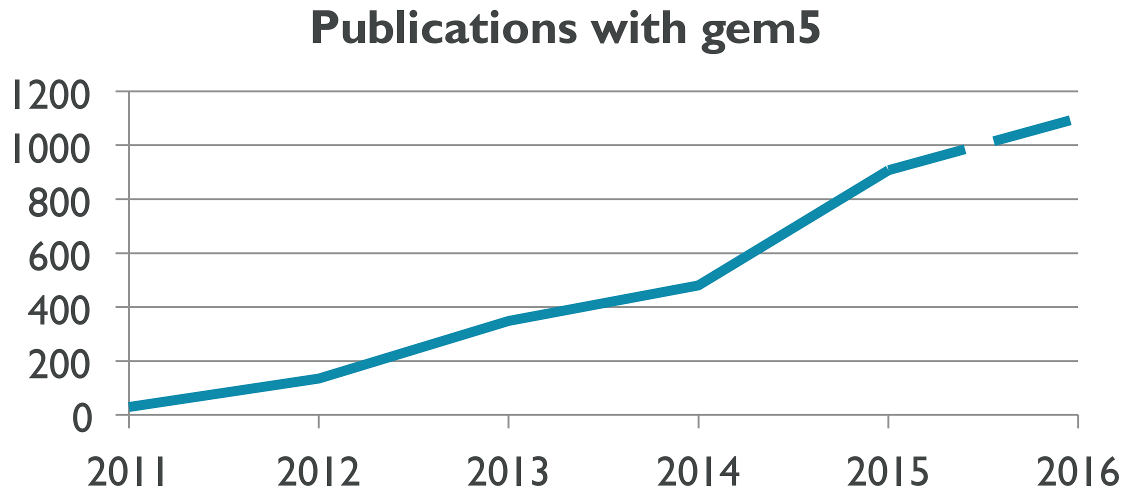
- M5
  - Highly configurable **simulation framework** to support multiple ISAs, and diverse CPU models
  - developed @ The University of Michigan
- GEMS [General Execution-driven Multiprocessor Simulator]
  - detailed and flexible **memory system model**
  - Includes support for **multiple cache coherence protocols** and interconnect models
  - developed @ The University of Wisconsin Madison

Where did it come from

Ref: http://gem5.org/dist/tutorials/hipeac2012/gem5_hipeac.pdf

Users and contributors

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



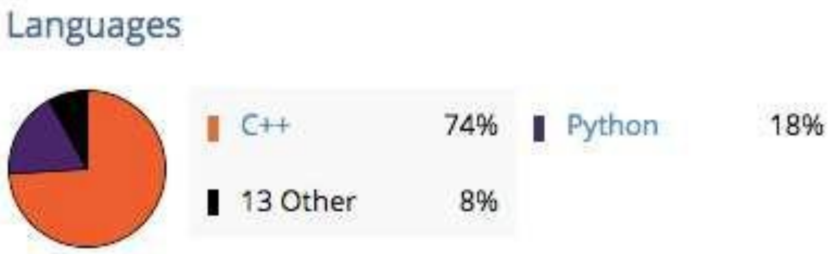
In a Nutshell, gem5...

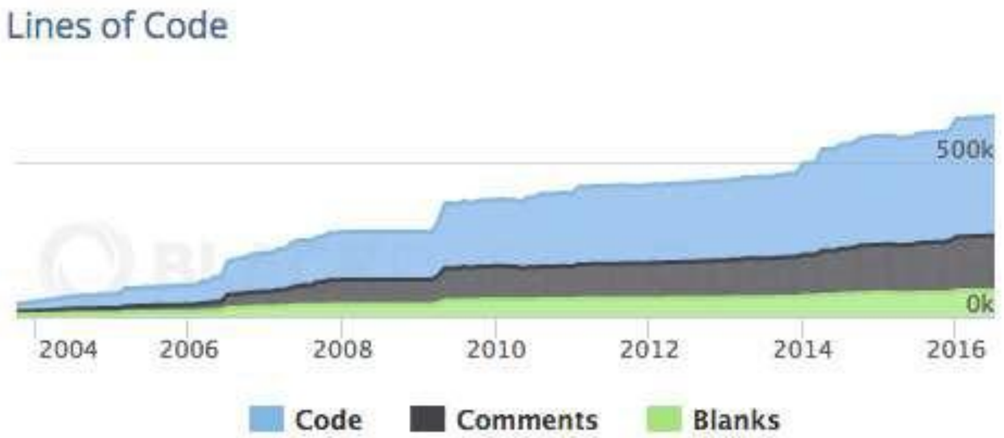
… has had 11,558 commits made by 193 contributors representing 386,321 lines of code

… 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

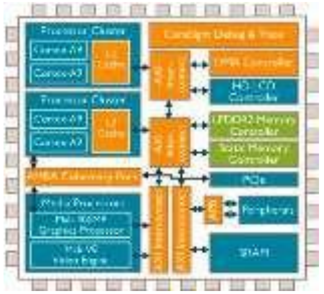




ARM

Why gem5?

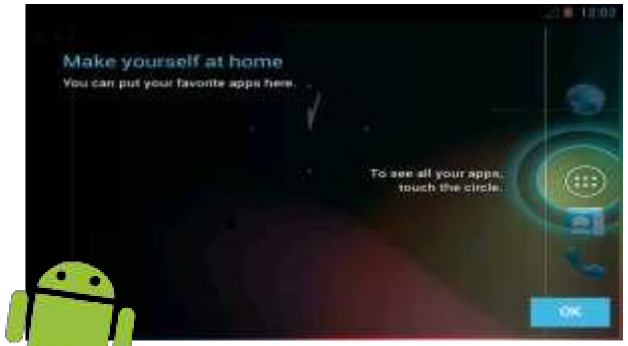
- Runs real workloads
  - Analyze workloads that customers use and care about
  - … including complex workloads such as Android
- Comprehensive model library
  - Memory and I/O devices
  - Full OS
  - Clients and servers
- Rapid *early* prototyping
  - New ideas can be tested quickly
  - System-level 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!*



AMBA Digital Highway from ARM

Ubuntu (Linux 6.x)

Android

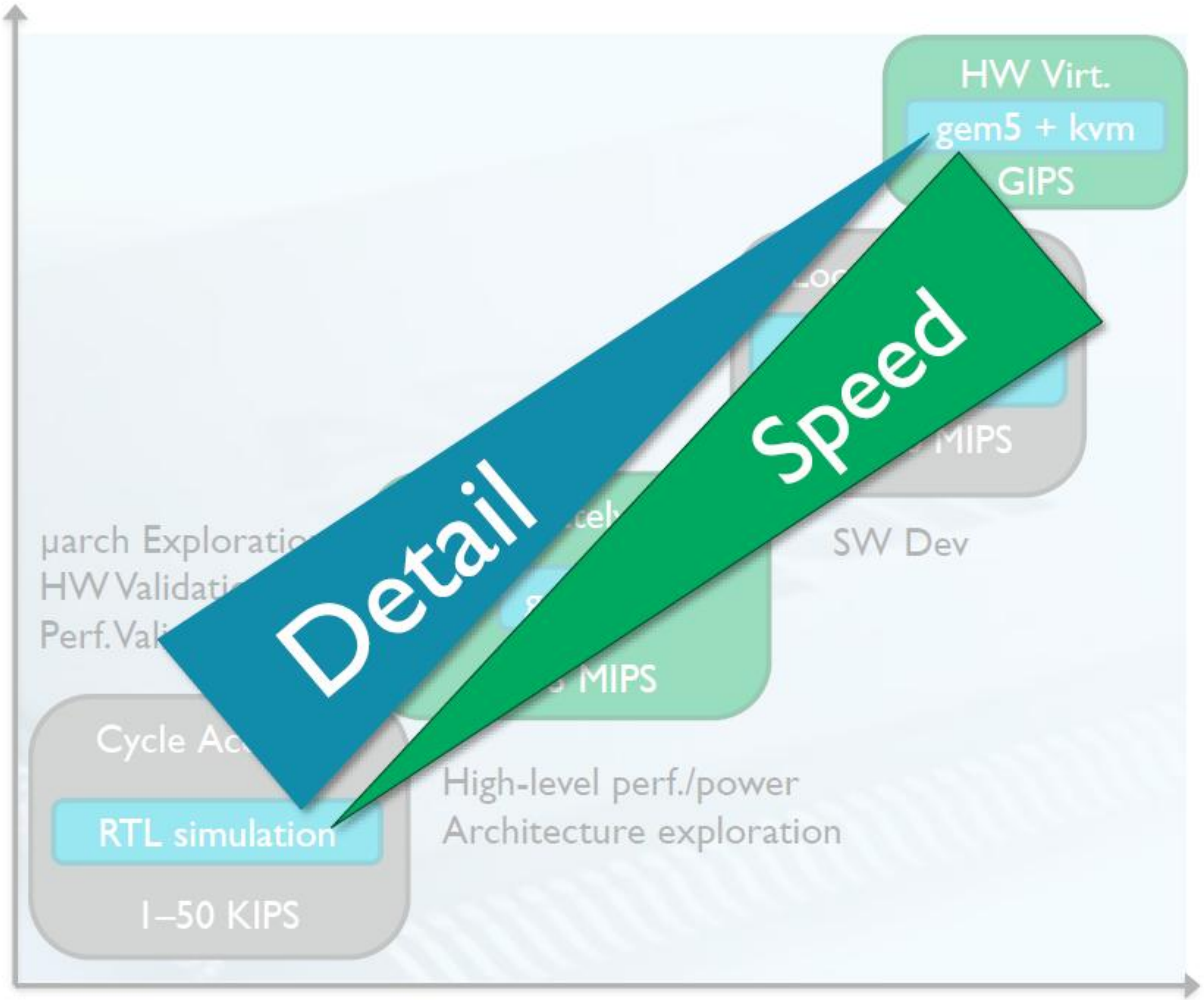


android

ARM

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

Prerequisites

- Operating system:
  - OSX, Linux (recommended)
  - Limited support for Windows 10 with a Linux environment
- Software:
  - git
  - Python (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 -j4
```

- Guest architecture
- Several architectures in the source tree.
- Most common ones are:
  - **ARM**
  - **RISC-V**
  - **X86**

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

Modes

- gem5 has two fundamental modes
- Full system (FS)
  - For **booting operating systems (OSs)**
  - Including devices
  - Interrupts, exceptions, privileged instructions
- Syscall emulation (SE)
  - For running individual applications, or set of applications on MP
  - Models user-visible ISA plus common system calls
  - System calls emulated, typ. by calling host OS
  - Simplified address translation model, no scheduling

Capabilities

- **Execution modes**: System-call Emulation (SE) & Full-System (FS)
- **ISAs**: Alpha, **ARM**, **RISC-V**, 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

Flexibility

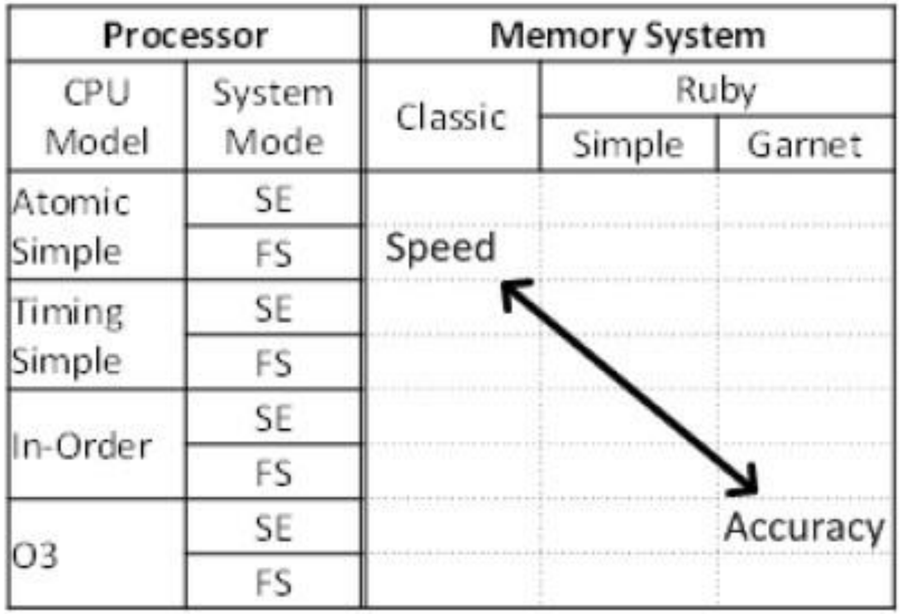


Figure 1: Speed vs. Accuracy Spectrum.

- Classic (from M5): Fast and configurable memory system model.
- Ruby (from GEMS) : framework/infrastructure to model variety of cache-coherent memory system.

Source: The gem5 Simulator, May 2011 issue of ACM SIGARCH Computer Architecture News

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`

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

Running an example script

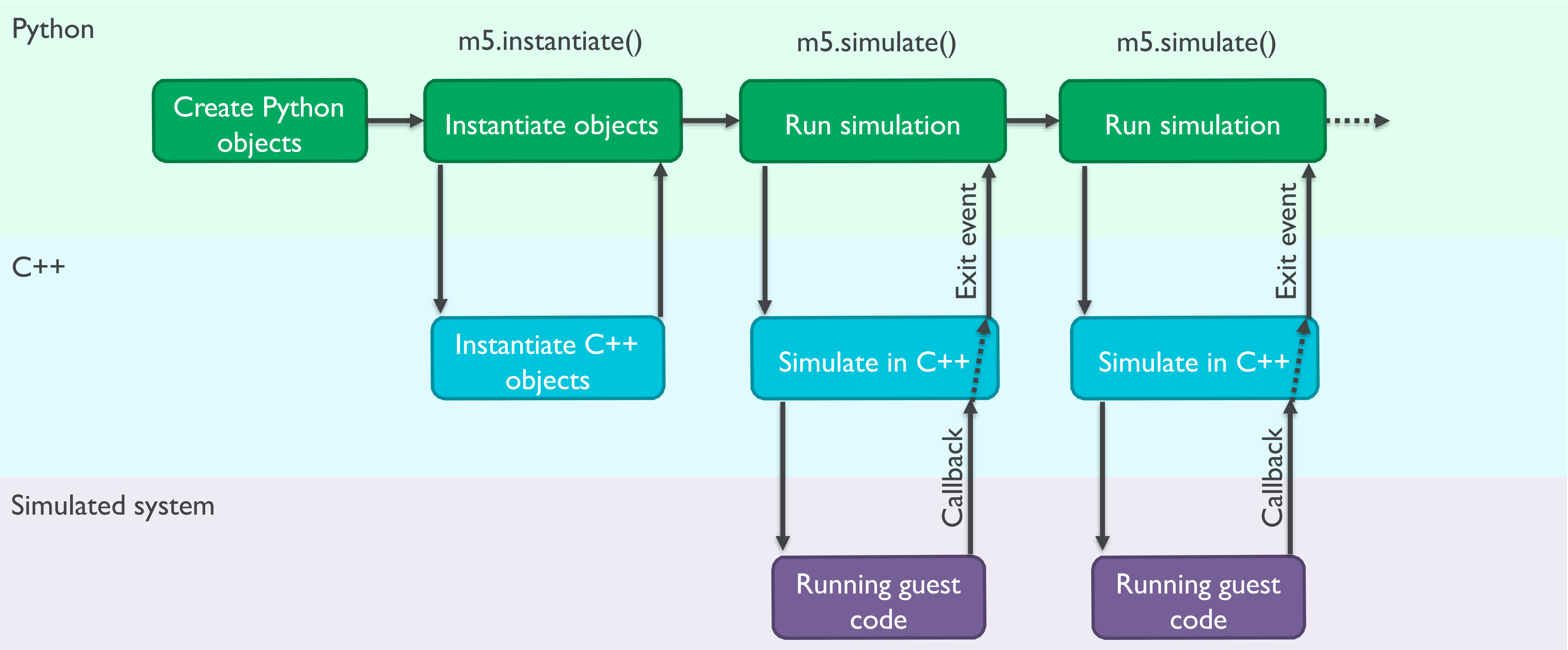
```
$ build/ARM/gem5.opt configs/example/arm/fs_bigLITTLE.py \
    --kernel path/to/vmlinux \
    --cpu-type atomic \
    --dtb $PWD/system/arm/dt/armv8_gem5_v1_big_little_1_1.dtb \
    --disk your_disk_image.img
```

- 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

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

Control flow



ARM

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

ARM

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

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

Understanding gem5 output

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

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

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

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

stats.txt

```
---------- Begin Simulation Statistics -------

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

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

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

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 `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`

ARM

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                    0xe353001e
  50000:     system.cpu:  Decode:  Decoded cmps instruction:
  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"`

ARM

Ευχαριστώ για την προσοχή σας!
απορίες?

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ