UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Simulation with gem5

RISC-V: 1-DAY COURSE FROM THEORY TO INDUSTRY

Speaker: Adrià Armejach (adria.armejach@upc.edu)
Special thanks to Nitish Arya

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA
Plan de Recuperación, Transformación y Resiliencia
España | digital 2026
Barcelona Supercomputing Center Centro Nacional de Supercomputación

Logistics

Repository

- Link to the repository at the bottom of each slide
- Helper repository with relevant information:
    - PDF with the slides
    - README file with all the commands we will be using during the hands-on later
        - Useful to copy&paste
    - Helper script(s)

Preparing the docker container

- Hands-on with gem5
  - Preparing and starting the docker container (online)

```
$ docker run -it --rm registry.gitlab.bsc.es/aarmejac/gem5-handson
```

```
root@5a52b432kje7o:~#
```

  - Preparing and starting the docker container (USB Key)

```
$ docker load < gem5-handson.tar.gz
$ docker run -it --rm registry.gitlab.bsc.es/aarmejac/gem5-handson
```

```
root@5a52b432kje7o:~#
```

https://gitlab.bsc.es/aarmejac/gem5-handson

Brief Introduction to gem5

gem5 history

- First there was M5

Created at Michigan by students of Steve Reinhardt, principally Nate Binkert .

“A tool for simulating systems”

- Then came GEMS

Multifacet GEMS
General Execution-driven Multiprocessor Simulator

Created at Wisconsin by students of Mark Hill and David Wood
Detailed memory system

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

gem5 is open source - resources

- gem5 code: https://github.com/gem5/gem5
- gem5 website: https://www.gem5.org
- gem5 YouTube: https://youtube.com/@gem5
- gem5 Slack: https://gem5-workspace.slack.com/join/shared_invite/zt-2e2nfln38-xslkN1aRmofRlAHOIkZaEA

gem5-20+: A new era in computer architecture simulation

Abdul Mutaal Ahmad
Adrian Herrera
Adrien Pesle
Adrià Armejach
Akash Bagdia
Alec Roelke
Alexandru Dutu
Ali Jafri
Ali Saidi
Amin Farmahini
Anders Handler
Andrea Mondelli
Andrea Pellegrini
Andreas Hansson
Andreas Sandberg
Andrew Bardsley
Andrew Lukefahr
Andrew Schultz
Andriani Mappoura
Ani Udipi
Anis Peysieux
Anouk Van Laer
Arthur Perais
Ashkan Tousi
Austin Harris
Avishai Tvila
Ayaz Akram
Bagus Hanindhito
Benjamin Nash
Bertrand Marquis
Binh Pham
Bjoern A. Zeeb
Blake Hechtman
Bobby R. Bruce
[illegible]
[illegible]
[illegible]
[illegible]
Brandon Potter
Brian Grayson
Cagdas Dirik
Chander Sudanthi
Chen Zou
Chris Adeniyi-Jones
Chris Emmons
Christian Menard
Christoph Pfister
Christopher Torng
Chuan Zhu
Chun-Chen Hsu
Ciro Santilli
Clint Smullen
Curtis Dunham
Dam Sunwoo
Dan Gibson
Daniel Carvalho
Daniel Johnson
Daniel Sanchez
David Guillen-Fandos
David Hashe
David Oehmke
[illegible]
[illegible]
[illegible]
Djordje Kovacevic
Dongxue Zhang
Doğukan Korkmaztürk
Dylan Johnson
Earl Ou
Edmund Grimley Evans
Emilio Castillo
Erfan Azarkhish
Eric Van Hensbergen
Erik Hallnor
Erik Tomusk
Faissal Sleiman
Fernando Endo
Gabe Black
Gabe Loh
Gabor Dozsa
Gedare Bloom
Gene WU
Gene Wu
Geoffrey Blake
Georg Kotheimer
Giacomo Gabrielli
[illegible]
Glenn [illegible]
Hamid Reza Khaleghzadeh
Hanhwi Jang
Hoa Nguyen
Hongil Yoon
Hsuan Hsu
Hussein Elnawawy
Ian Jiang
IanJiangICT
Ilias Vougioukas
Isaac Richter
Isaac Sánchez Barrera
Ivan Pizarro
Jack Whitham
Jairo Balart
Jakub Jermar
James Clarkson
Jan-Peter Larsson
Jason Lowe-Power
Javier Bueno Hedo
Javier Cano-Cano
Javier Setoain
Jayneel Gandhi
[illegible]
[illegible]
[illegible]
Jiuyue Ma
Joe Gross
Joel Hestness
John Alsop
John Kalamatianos
Jordi Vaquero
Jose Marinho
Jui-min Lee
Kanishk Sugand
Karthik Sangaiah
Ke Meng
Kevin Brodsky
Kevin Lim
Khalique
Koan-Sin Tan
Korey Sewell
Krishnendra Nathella
Lena Olson
Lisa Hsu
Lluc Alvarez
Lluís Vilanova
Mahyar Samani
Malek Musleh
Marc Mari Barcelo
[illegible]
[illegible]
[illegible]
Marjan Fariborz
Matt DeVuyst
Matt Evans
Matt Horsnell
Matt Poremba
Matt Sinclair
Matteo Andreozzi
Matteo M. Fusi
Matthew Poremba
Matthias Hille
Matthias Jung
Maurice Becker
Maxime Martinasso
Maximilian Stein
Maximilien Breughe
Michael Adler
Michael LeBeane
Michael Levenhagen
Michiel Van Tol
Miguel Serrano
Mike Upton
[illegible]iles Kaufmann
[illegible]
[illegible]
[illegible]
Mohammad Alian
Monir Mozumder
Moyang Wang
Mrinmoy Ghosh
Nathan Binkert
Nathanael Premillieu
Nayan Deshmukh
Neha Agarwal
Nicholas Lindsay
Nicolas Derumigny
Nicolas Zea
Nikos Nikoleris
Nils Asmussen
Nuwan Jayasena
Ola Jeppsson
Omar Naji
Pablo Prieto
Palle Lyckegaard
Pau Cabre
Paul Rosenfeld
Peter Enns
Pin-Yen Lin
[illegible]
[illegible]
[illegible] Petrakis
Pouya Fotouhi
Prakash Ramrakhyani
Pritha Ghoshal
Radhika Jagtap
Rahul Thakur
Reiley Jeapaul
Rekai Gonzalez-Alberquilla
Rene de Jong
Ricardo Alves
Richard D. Strong
Richard Strong
Rico Amslinger
Riken Gohil
Rizwana Begum
Robert Kovacsics
Robert Scheffel
Rohit Kurup
Ron Dreslinski
Ruben Ayrapetyan
Rune Holm
Ruslan Bukin
Rutuja Oza
Ryan G[illegible]
Samuel [illegible]
Sandip[illegible]
Santi G[illegible]
Sascha Bischoff
Sean McGoogan
Sean Wilson
Sergei Trofimov
Severin Wischmann
Shawn Rosti
Sherif Elhabbal
Siddhesh Poyarekar
Somayeh Sardashti
Sooraj Puthoor
Sophiane Senni
Soumyaroop Roy
Srikant Bharadwaj
Stan Czerniawski
Stanislaw Czerniawski
Stephan Diestelhorst
Stephen Hines
Steve Raasch
Steve Reinhardt
Stian Hvatum
[illegible]
[illegible]
[illegible]
[illegible]
Tao Zhang
Thomas Grass
Tiago Mück
Tim Harris
Timothy Hayes
Timothy M. Jones
Tom Jablin
Tommaso Marinelli
Tony Gutierrez
Trivikram Reddy
Tuan Ta
Tushar Krishna
Umesh Bhaskar
Uri Wiener
Victor Garcia
Vilas Sridharan
Vince Weaver
Vincentius Robby
Wade Walker
Weiping Liao
Wendy Elsasser
William Wang
Willy Wolff
Xiangyu Dong
Xianwei Zhang
Xiaoyu Ma
[illegible] Ouyang
[illegible] Eckert
[illegible]
[illegible]
[illegible]Wang
[illegible]
[illegible]odama
[illegible]heng
[illegible]Vang
[illegible]
[illegible]
[illegible]lian

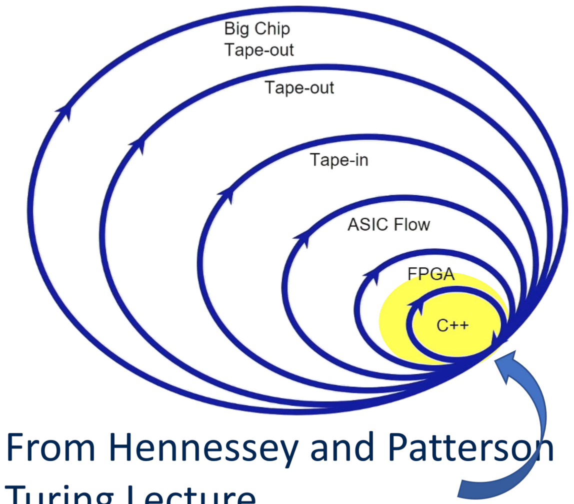
Your name here!

https://gitlab.bsc.es/aarmejac/gem5-handson

Why simulation?

- Need a tool to evaluate systems that don't exist (yet)
  - Performance, power, energy, etc.
- Very costly to actually make the hardware
- Computer systems are complex
  - Not easy to be accurate without a full system view
- Simulation can be parameterized
  - Design-space exploration of parameters
  - Sensitivity analysis

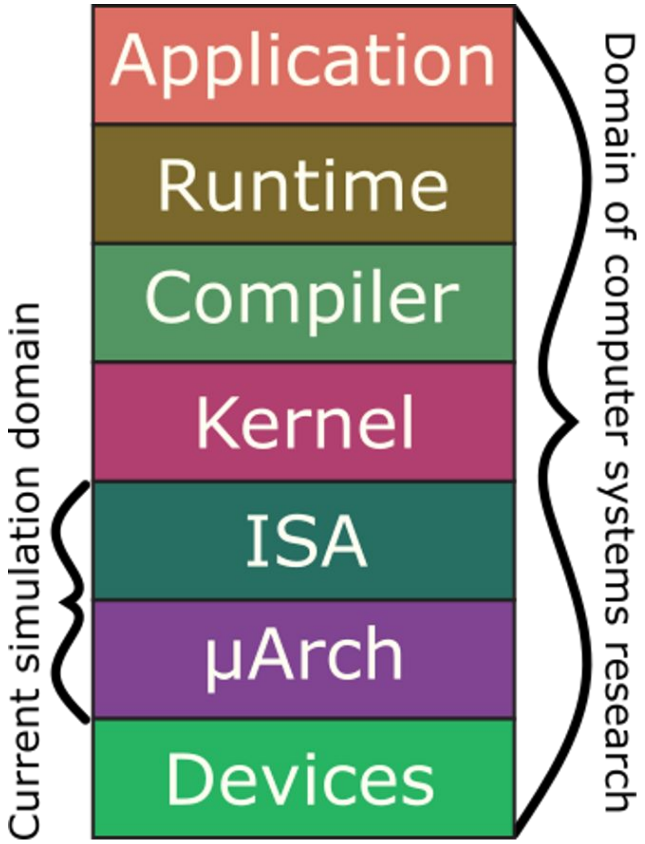
Agile Hardware Dev. Methodology



From Hennessey and Patterson Turing Lecture

gem5 goals

- Anyone (including non-architects) can download and use gem5
- Used for cross-stack research:
  - Change kernel, change runtime, change hardware, all in concert
  - Run full ML stacks or other emerging apps
- You can help the community!
  - 100s of contributors & 1000s(?) of users
- Aim to meet the needs of
  - Academic community
  - Industry research and development
  - Classroom use


Application
Runtime
Compiler
Kernel
ISA
µArch
Devices
Current simulation domain
Domain of computer systems research

https://gitlab.bsc.es/aarmejac/gem5-handson

Kinds of simulation

- Functional simulation
- Instrumentation-based
- Trace-based
- Execution-driven
- Full system

Kinds of simulation (gem5)

Execution driven

- Functional and timing simulation is combined
- Real instruction flow is simulated
- gem5 in Syscall Emulation mode, and many others

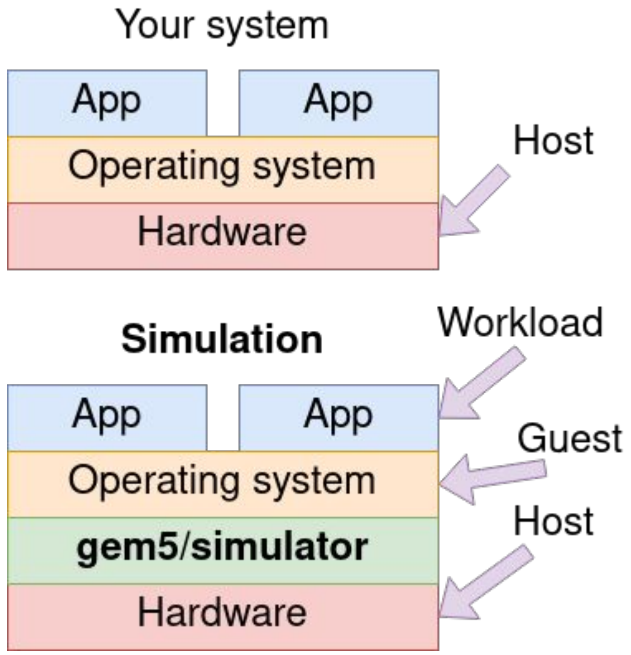
Full system

- Components modeled with enough fidelity to boot unmodified Linux
- Simulated applications run on top of this simulated Linux OS
- gem5 in Full System mode, and others

Nomenclature (gem5)

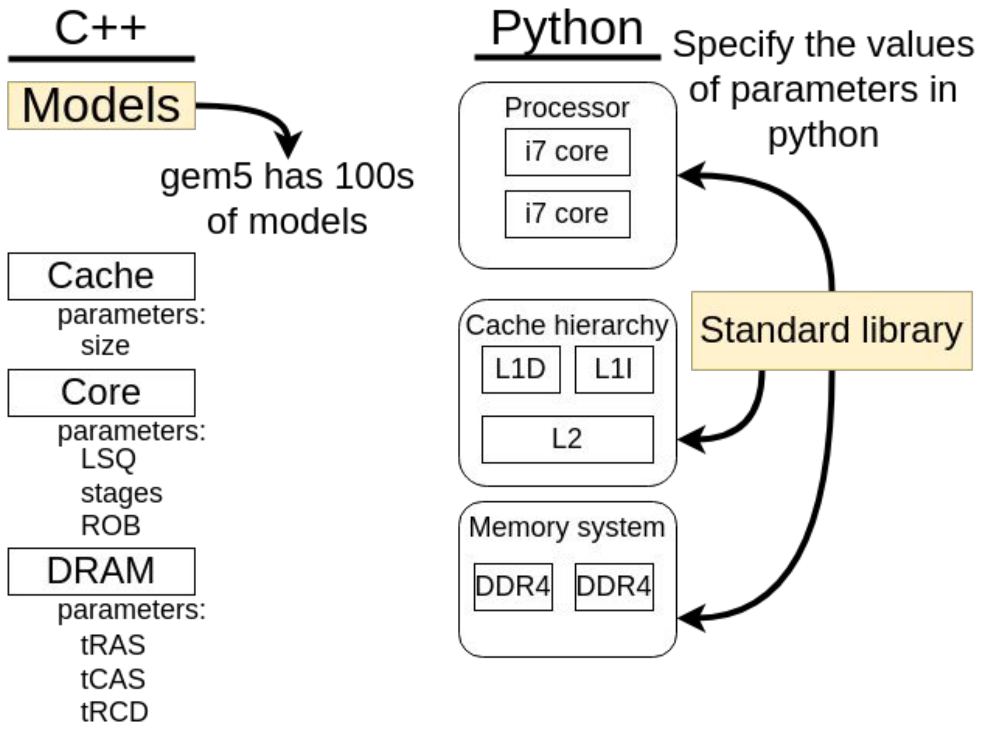
- **Host**: The actual hardware you are using
- **Simulator:** Runs on the host
  - Exposes hardware to the guest
- **Guest:** Code running on simulated hardware
  - OS running on gem5 is guest OS
  - gem5 is simulating hardware

- **gem5/simulator code:** Runs natively
  - executes/emulates the guest code
- **Guest's code:** (or benchmark, **Workload**, etc.)
  - Runs on gem5, not on the host



gem5 organization and software architecture

gem5 software architecture



Python

Control the simulation

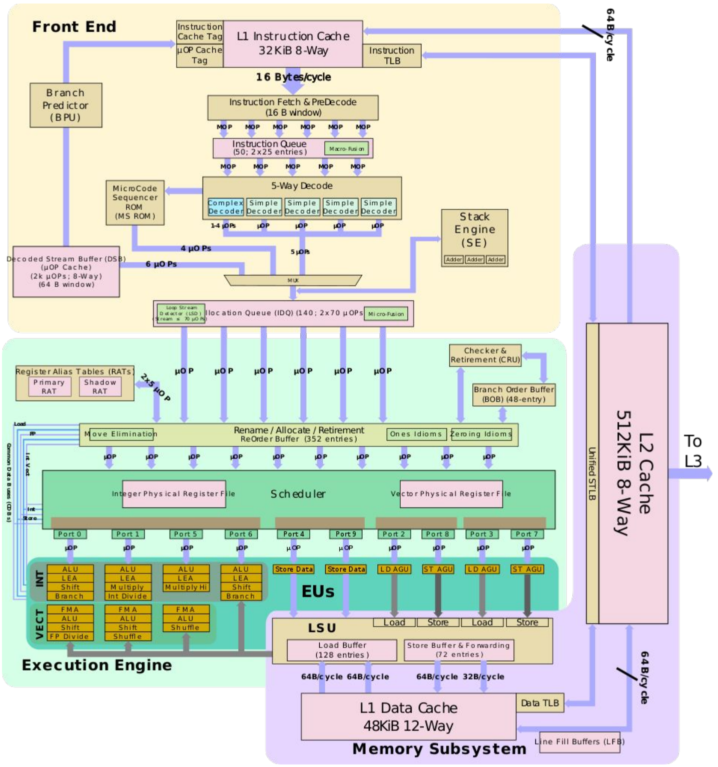
```
sim.start()
sim.dump_stats()
sim.stop_at_inst(...)
```

Develop experiments with the key parameters

https://gitlab.bsc.es/aarmejac/gem5-handson

gem5 architecture: SimObject

- **Model** - this is the C++ code in **src/**
  - does the timing simulation
- **Parameters** - python code in **src/**
  - Expose parameters for each SimObject
- **Instance or Configuration** - python code
  - A particular choice for the parameters
  - May connect multiple SimObjects
  - Examples
    - In the gem5 standard library
    - In **configs/**


Front End
L1 Instruction Cache 32KiB 8-Way
Instruction TLB
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode
Instruction Queue
5-Way Decode
MicroCode Sequencer ROM (MS ROM)
Decoded Stream Buffer (DSB)
Stack Engine (SE)
Allocation Queue (IDQ)
Register Alias Tables (RATs)
Rename / Allocate / Retirement ReOrder Buffer (352 entries)
Scheduler
Integer Physical Register File
Vector Physical Register File
EUs
LSU
Load Buffer (128 entries)
Store Buffer & Forwarding (72 entries)
Execution Engine
L1 Data Cache 48KiB 12-Way
Data TLB
Memory Subsystem
L2 Cache 512KiB 8-Way
To L3

https://gitlab.bsc.es/aarmejac/gem5-handson

Instantiate a SimObject (L1I cache) and connect with cpu

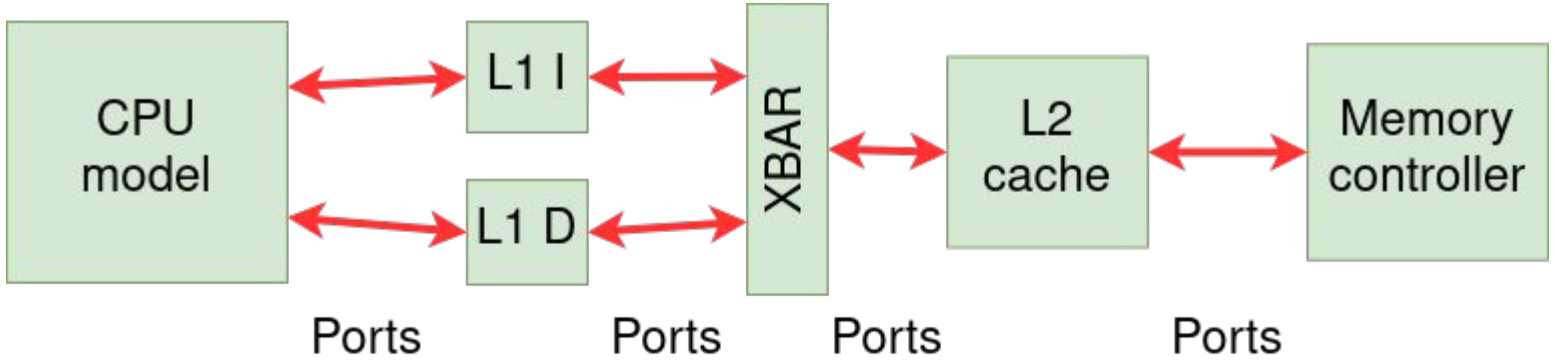
```python
from m5.objects import Cache
from abc import ABC

class L1Cache(type(Cache), type(ABC)):
    """Simple L1 Cache with default values"""

    def __init__(self):
        # Here we set/override the default values for the cache.
        self.assoc = 8
        self.tag_latency = 1
        self.data_latency = 1
        self.response_latency = 1
        self.mshrs = 16
        self.tgts_per_mshr = 20
        self.writeback_clean = True
        super().__init__()
```

```python
class L1ICache(L1Cache):
    """Simple L1 instruction cache with default values
    """

    def __init__(self):
        # Set the size
        self.size = "32kB"
        super().__init__()

    # This is the implementation needed for
    # the L1ICache to connect to the CPU.
    def connectCPU(self, cpu):
        """Connect this cache's port to a CPU icache port
        """
        self.cpu_side = cpu.icache_port
```

https://gitlab.bsc.es/aarmejac/gem5-handson

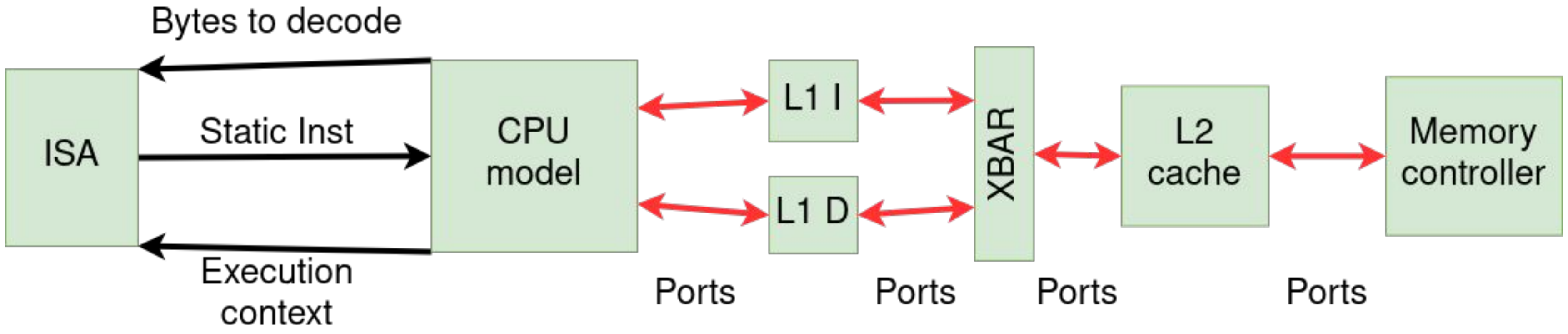
gem5's main abstractions: Memory Requests

- **Ports** allow you to send requests and receive responses - *unidirectional*
- Anything with a Request port can be connected to any Response port



gem5's main abstractions: ISA vs CPU model

- ISA and CPU models are orthogonal
  - Any ISA should work with any CPU model
- "Execution Context" is the interface
- gem5 supports multiple ISAs: x86, Arm, RISC-V



https://gitlab.bsc.es/aarmejac/gem5-handson

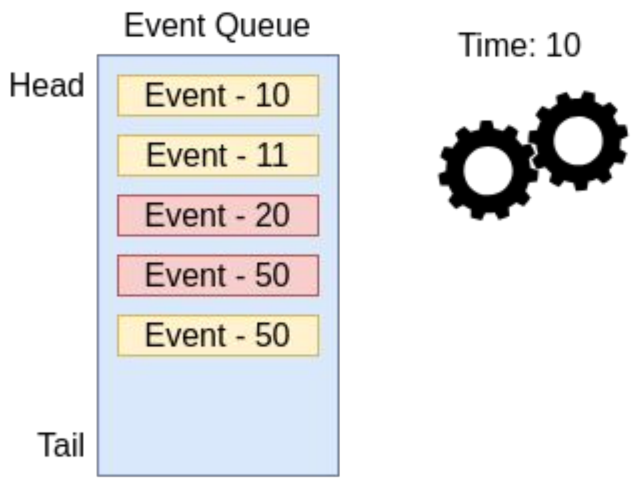
How does gem5 simulate?

gem5 architecture: Simulation

gem5 is a ***discrete event simulator***

At each timestep:

1. Event at the head is dequeued

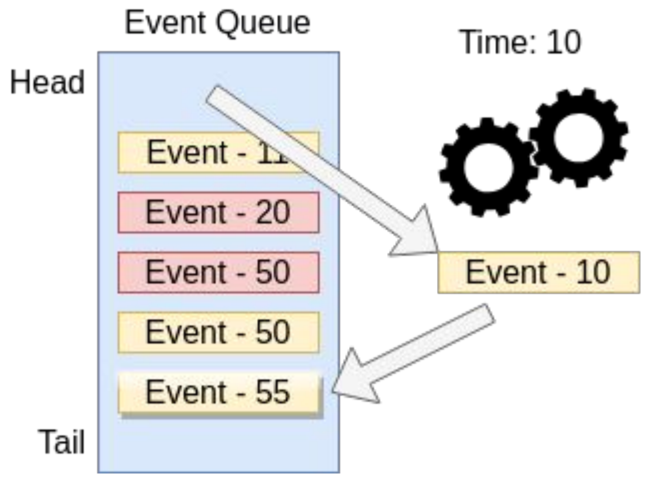


gem5 architecture: Simulation

gem5 is a ***discrete event simulator***

At each timestep:

1. Event at the head is dequeued
2. The event is processed



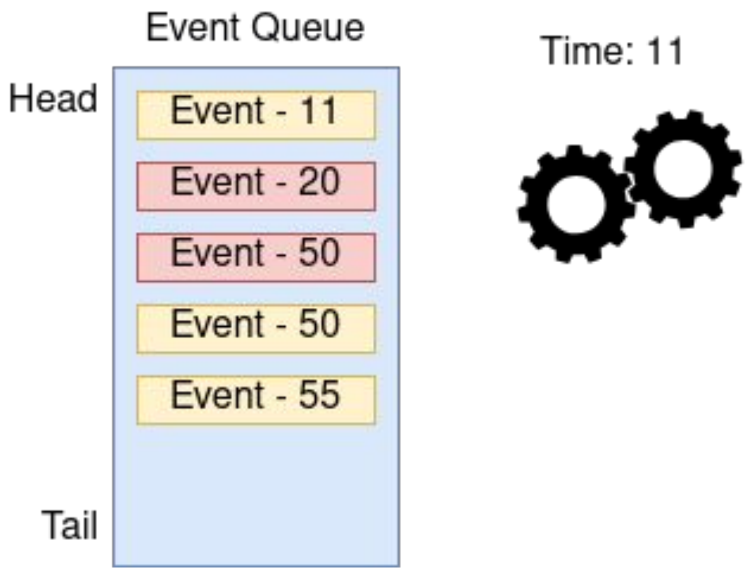
gem5 architecture: Simulation

gem5 is a ***discrete event simulator***

At each timestep:

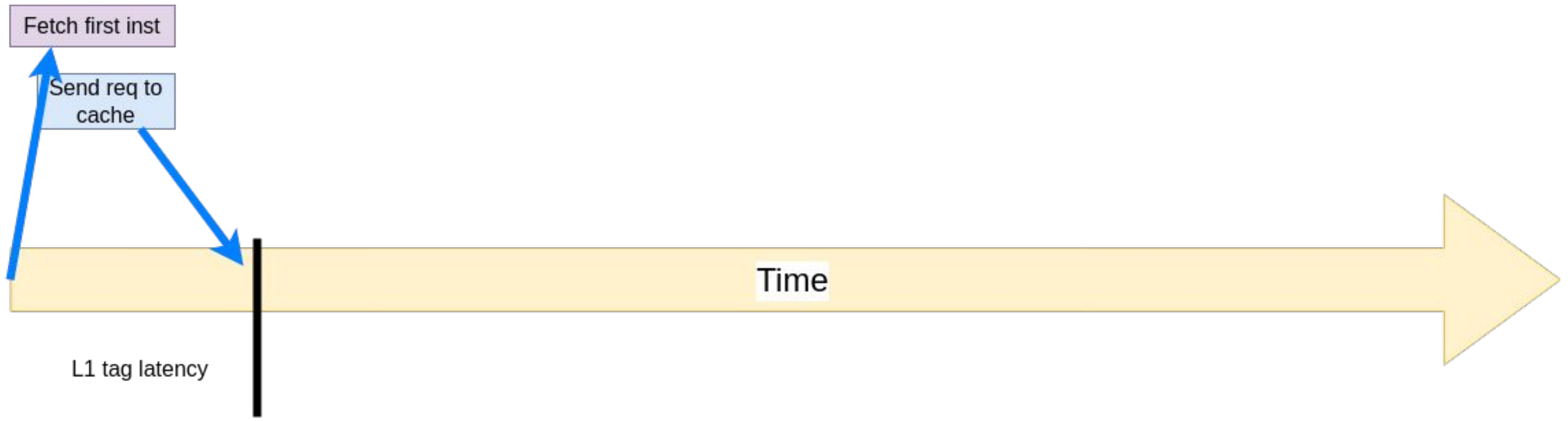
1. Event at the head is dequeued
2. The event is executed
3. New events may be scheduled

All SimObjects can add events into the event queue



https://gitlab.bsc.es/aarmejac/gem5-handson

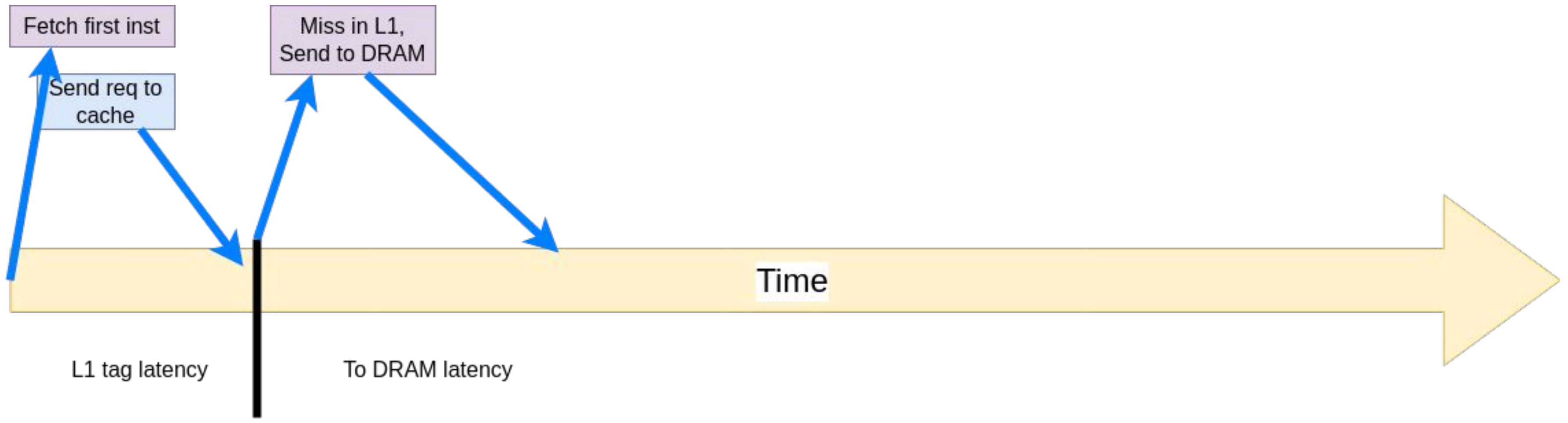
Discrete event simulation example



- To model things that take time, schedule the next event in the future
  - latency of current event
- Can call functions instead of scheduling events, but they occur at the same “time”

https://gitlab.bsc.es/aarmejac/gem5-handson

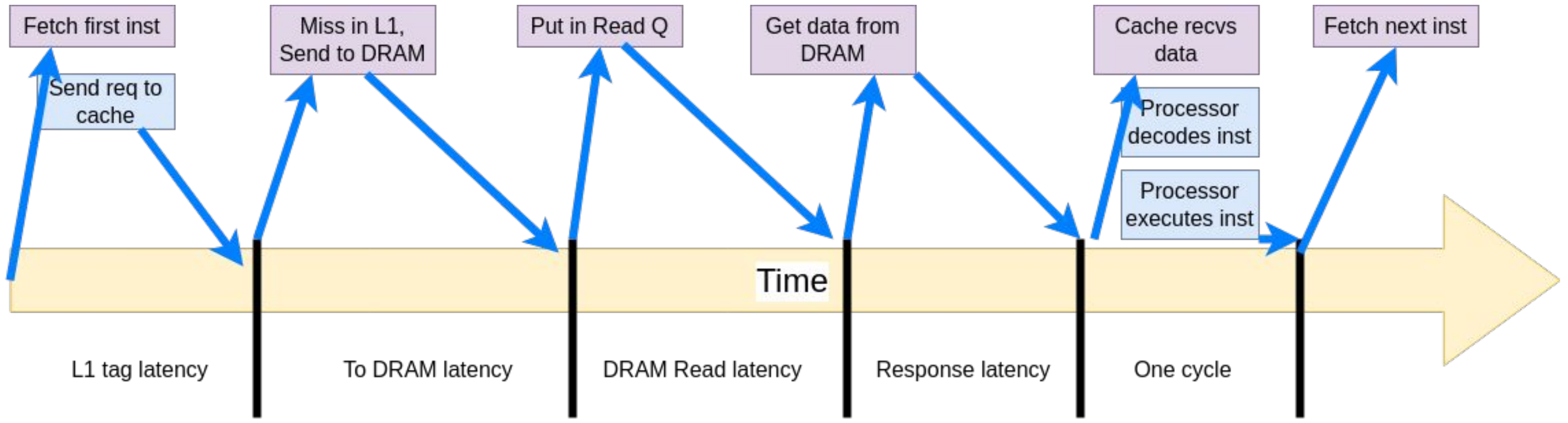
Discrete event simulation example



- To model things that take time, schedule the next event in the future
  - latency of current event
- Can call functions instead of scheduling events, but they occur at the same “time”

https://gitlab.bsc.es/aarmejac/gem5-handson

Discrete event simulation example



- To model things that take time, schedule the next event in the future
    - latency of current event
- Can call functions instead of scheduling events, but they occur at the same “time”

https://gitlab.bsc.es/aarmejac/gem5-handson

Using the gem5 standard library to instantiate our system

gem5's Standard Library

- Purpose: to provide a set of predefined components that can be used to build a simulation - does the majority of the work for you.
- Due to its modular object-oriented design
  - gem5 can be thought of as a set of components (SimObjects) that can be plugged together to form a simulation
  - **Processor:** have one or more *cores (SimObjects)* which are of some CPU model (c++ code)
  - **Cache hierarchy:** a set of caches (SimObjects) that can be connected to a processor and memory system
  - **Memory system:** a set of memory controllers and memory devices that can be connected to the cache hierarchy
  - **Board:** The backbone of the system. You plug components into the board.

gem5 standard library components: SimpleBoard

- Will use the SimpleBoard
  - Can run any ISA in Syscall Emulation mode
  - Defines all the necessary SimObjects
- Requires other components from the standard library
  - **processor** - which can have one or more cores
  - **cache_hierarchy** - defines all the levels between the processor and memory
  - **memory** - which will define memory controllers and interfaces (DDR, HBM, …)

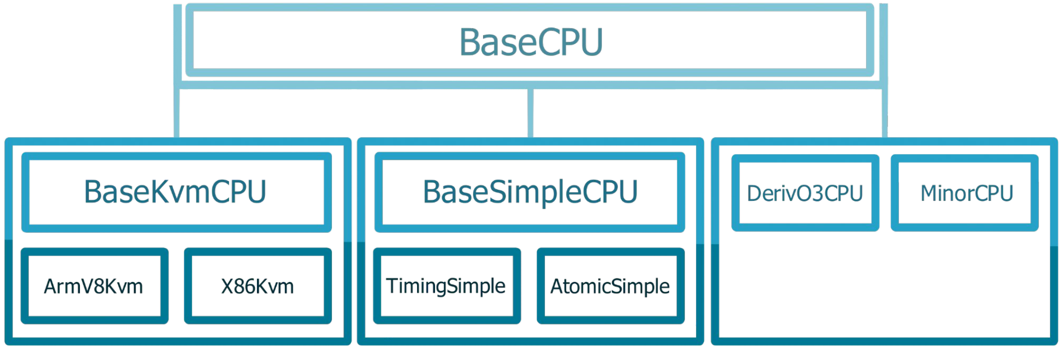
```
from
gem5.components.board.simple_board
import SimpleBoard

# Setup the board.

board = SimpleBoard(

clk_freq="1GHz",

processor=processor,

memory=memory,

cache_hierarchy=cache_hierarchy,

)
```

https://gitlab.bsc.es/aarmejac/gem5-handson

Processor: Using the BaseCPUProcessor

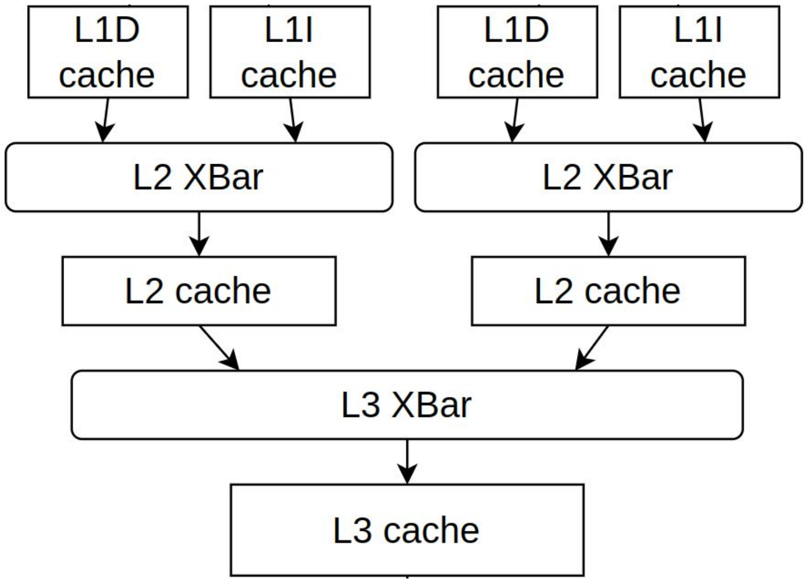
```
from gem5.components.processors.base_cpu_processor import BaseCPUProcessor
```

- Expects a list of cores
- Cores can be of any available CPU model
- Can mix different core models



https://gitlab.bsc.es/aarmejac/gem5-handson

Cache hierarchy



- Different components available
  - Private L1 caches
  - Private L1 and private L2 cache
  - Private L1, private L2 cache, shared L3 cache

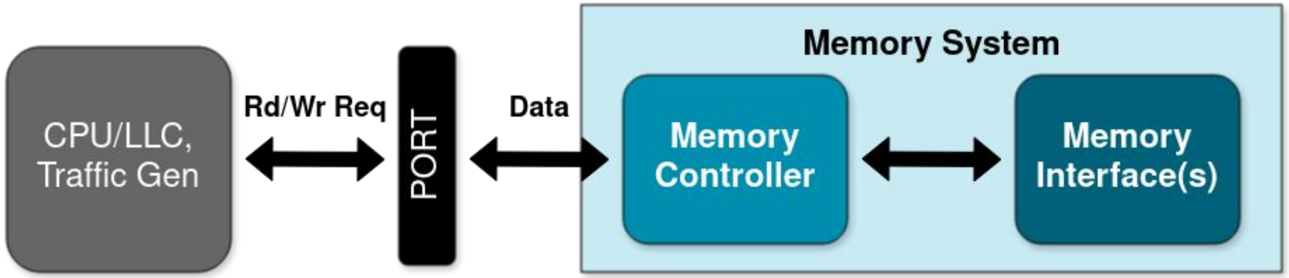
```
from gem5.components.cachehierarchies.classic.private_l1_private_l2_cache_hierarchy
import (

PrivateL1PrivateL2CacheHierarchy,

)
```

https://gitlab.bsc.es/aarmejac/gem5-handson

Modeling memory

gem5's memory system consists of two main components:

1. Memory Controller - responsible for scheduling and issuing read/write requests
2. Memory Interface(s) - implements the architecture and timing parameters



Modeling memory

- **SimpleMemory()** allows the user to not worry about timing parameters and instead just give the desired latency and bandwidth
- **ChanneledMemory()** encompasses a whole memory system (both the controller and the interface)

Component for a single-channel memory system using a DDR31600 DIMM

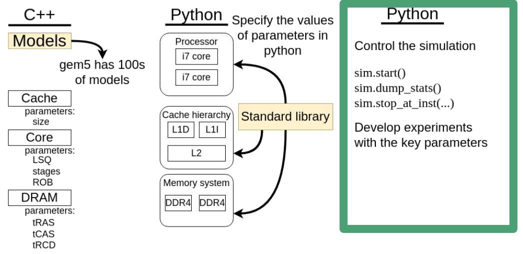
```
from gem5.components.memory import SingleChannelDDR3_1600
```

Plug components into the board

- Boards aggregate components
- Once we have instantiated required objects, we can pass them to a board for simulation
- **SimpleBoard** can run any ISA in Syscall Emulation (SE) mode

```
from
gem5.components.board.simple_board
import SimpleBoard

# Setup the board.

board = SimpleBoard(

clk_freq="1GHz",

processor=processor,

memory=memory,

cache_hierarchy=cache_hierarchy,

)
```

Set up the workload and simulation



- Now we set the workload which we need to run on the simulated system
- We pass the whole board to the simulator object and start the simulation
- This completes the required configuration for running an SE mode simulation

```
board.set_se_binary_workload(
BinaryResource(local_path=f"/usr/local/bin/{args.binary}",id=args.binary),
arguments=args.arguments)

simulator = Simulator(board=board, full_system=False)

print("Beginning simulation!")

simulator.run()
```

https://gitlab.bsc.es/aarmejac/gem5-handson

Invoking gem5 to perform a simulation

Performing a simulation

- The most common format to start a gem5 simulation is with the below command

```
gem5.binary [gem5_options] config.py [config_file_options]
```

- An example command:

```
gem5.opt -d /tmp/first_gem5_run configs/example.py --cores=4
```

- There are a lot of *gem5_options* available, one of which is to specify an output directory where all the simulation data will be stored (-d)
    - The default output directory is **m5out/**

gem5's output

In **m5out/** you will see:

- **stats.txt** - The statistics from the simulation
  - In text format
- **config.{ini/json}** - The configuration file used in structured format
  - It contains each and every connection, component and parameter
- **config*.{pdf/svg}** - A visualization of the configuration for the system and caches

Hands-on with gem5

Starting the docker container

- Hands-on with gem5
  - Preparing and starting the docker container (online)

```
$ docker run -it --rm registry.gitlab.bsc.es/aarmejac/gem5-handson
```

```
root@5a52b432kje7o:~#
```

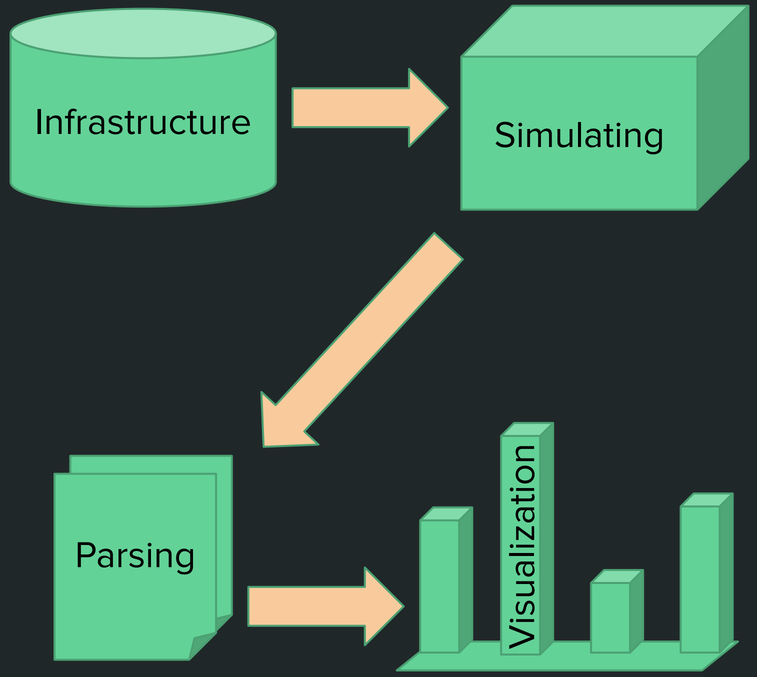
  - Preparing and starting the docker container (USB Key)

```
$ docker load < gem5-handson.tar.gz
$ docker run -it --rm registry.gitlab.bsc.es/aarmejac/gem5-handson
```

```
root@5a52b432kje7o:~#
```

https://gitlab.bsc.es/aarmejac/gem5-handson

Our Setup



1 - Infrastructure

Infrastructure

- A **Docker container** that includes:
  - Pre-built gem5 binary for RISC-V
  - Static application binaries that we will use for Syscall Emulation simulations
  - A configuration script to instantiate the system we want to simulate

- A **gitlab repository** that we will now clone, contains:
  - Slide deck
  - README with commands for hands-on
  - Parsing script we will use during the hands-on

https://gitlab.bsc.es/aarmejac/gem5-handson

Running the container

- Create a directory to store the output of simulations on your host
- Run the docker container, this will drop us inside the container

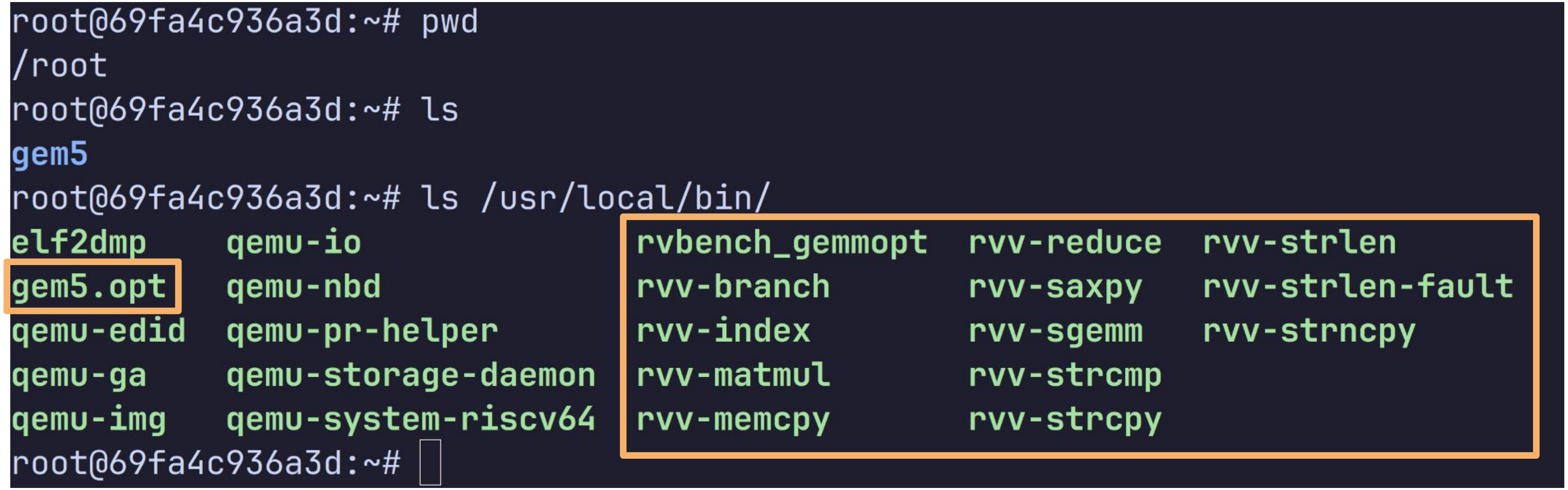
```
$ mkdir riscv_handson

$ cd riscv_handson

$ docker run -it --rm registry.gitlab.bsc.es/aarmejac/gem5-handson

root@650acbc0e4d3:~#
```

host container

https://gitlab.bsc.es/aarmejac/gem5-handson

Exploring the container

- Once inside the container we can explore the file system



```
root@69fa4c936a3d:~# pwd
/root
root@69fa4c936a3d:~# ls
gem5
root@69fa4c936a3d:~# ls /usr/local/bin/
elf2dmp     qemu-io                 rvbench_gemmopt  rvv-reduce  rvv-strlen
gem5.opt    qemu-nbd                rvv-branch       rvv-saxpy   rvv-strlen-fault
qemu-edid   qemu-pr-helper          rvv-index        rvv-sgemm   rvv-strncpy
qemu-ga     qemu-storage-daemon     rvv-matmul       rvv-strcmp
qemu-img    qemu-system-riscv64     rvv-memcpy       rvv-strcpy
root@69fa4c936a3d:~#
```

https://gitlab.bsc.es/aarmejac/gem5-handson

Cloning the helper repository

- We will clone the gitlab repository **inside** the docker container

```
root@650acbc0e4d3:~# git clone https://gitlab.bsc.es/aarmejac/gem5-handson.git

root@650acbc0e4d3:~# ls gem5-handson

README.md         demo-Dockerfile       parse_gem5_stats.py
```

host container

https://gitlab.bsc.es/aarmejac/gem5-handson

2 - Simulating with gem5

Inspecting gem5 source code

- **build** - compiled code and ISA specific gem5 binary
- **configs** - pre-written configuration files to instantiate Systems-on-Chip for simulation
- **src** - gem5 source code with the different models

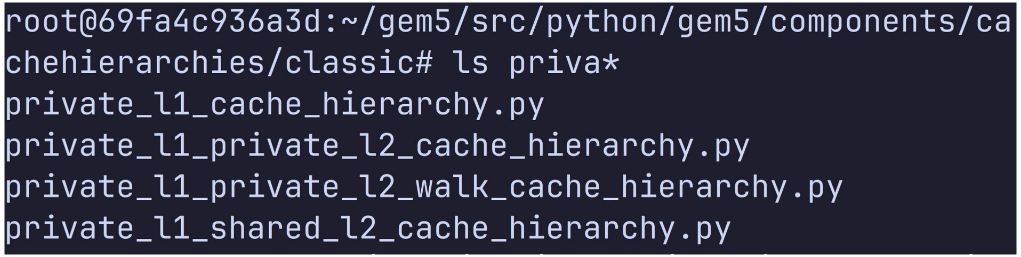
```
gem5/  (some files are omitted for brevity)
├── build
├── build_opts
├── build_tools
├── configs
├── ext
├── include
├── src
├── tests
├── util
├── SConstruct
└── TESTING.md
```

https://gitlab.bsc.es/aarmejac/gem5-handson

The gem5 standard library

- We have a lot of components on a System-on-Chip (core, caches, memory, ...)
  - And gem5 has multiple models for each component!
- It can be challenging to configure/instantiate a System-on-Chip
  - Solution: gem5 standard library

```
gem5/src/python/gem5/components/
├── boards
├── cachehierarchies
├── devices
├── __init__.py
├── memory
├── prefetch
└── processors
```



```
root@69fa4c936a3d:~/gem5/src/python/gem5/components/ca
chehierarchies/classic# ls priva*
private_l1_cache_hierarchy.py
private_l1_private_l2_cache_hierarchy.py
private_l1_private_l2_walk_cache_hierarchy.py
private_l1_shared_l2_cache_hierarchy.py
```

https://gitlab.bsc.es/aarmejac/gem5-handson

System-on-Chip instantiation

```
root@650acbc0e4d3:~# vim /root/gem5/configs/example/riscv-se.py
```

```
gem5/src/python/gem5/components/
├── boards
├── cachehierarchies
├── devices
├── __init__.py
├── memory
├── prefetch
└── processors
```

```
# Setup the board.
board = SimpleBoard(
    clk_freq="1GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)
```

```
cache_hierarchy = PrivateL1PrivateL2CacheHierarchy(
    l1d_size="32KiB", l1i_size="32KiB", l2_size="512KiB"
)
```

```
processor = BaseCPUProcessor(
    cores=create_cores(args.cores, args.vlen, args.elen)
)
```

```
# Setup the system memory.
memory = SingleChannelDDR3_1600()
```

https://gitlab.bsc.es/aarmejac/gem5-handson

Inspecting the benchmark

A bit about vectorized code

- If a vector register is wider you can do more work per instruction
- As you increase vector width the number of instructions you need to execute is lower
- Wider vectors have a hardware cost, it is not for free!
- In RISC-V the size of the vector in bits is called VLEN.

https://gitlab.bsc.es/aarmejac/gem5-handson

Blocked GEMM (Micro-kernel)

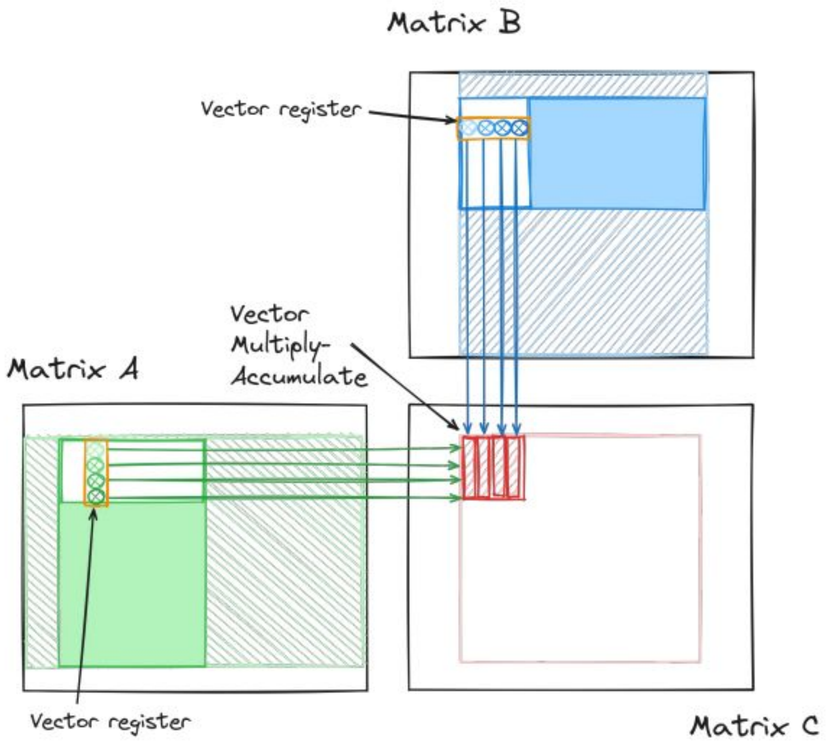


Image source: https://www.irit.fr/~Thomas.Carle/wp-content/uploads/sites/32/2024/05/A-predictable-SIMD-library-for-GEMM-routines.pdf

Simulating - use case: VLEN sizing

Perform 3 simulations providing different VLENs (vector register lengths)

vlen	128	256	512

```
root@650acbc0e4d3:~# gem5.opt -d /root/results/vlen-128
/root/gem5/configs/example/riscv-se.py rvbench_gemmopt 128 -v 128

root@650acbc0e4d3:~# gem5.opt -d /root/results/vlen-256
/root/gem5/configs/example/riscv-se.py rvbench_gemmopt 128 -v 256

root@650acbc0e4d3:~# gem5.opt -d /root/results/vlen-512
/root/gem5/configs/example/riscv-se.py rvbench_gemmopt 128 -v 512
```

host	container

Simulating - Check the output directories

- Check the output directories
- There will be a non-empty stats.txt for every successful simulation

```
root@650acbc0e4d3:~# ls /root/results

vlen-128   vlen-256   vlen-512

root@650acbc0e4d3:~# ls /root/results/vlen-128

citations.bib  config.ini  config.json  cpt.604173000
stats.txt
```

host | container

https://gitlab.bsc.es/aarmejac/gem5-handson

Inspecting the stats file

- At the end of the simulation a stats file is generated, containing the relevant information to extract conclusions from the execution.
- Inspect stats file

```
root@650acbc0e4d3:~# grep -m1 commitStats0.numInsts
/root/results/vlen-128/stats.txt

board.processor.cores.core.commitStats0.numInsts     4453140
# Number of instructions committed (thread level) (Count)
```

host container

https://gitlab.bsc.es/aarmejac/gem5-handson

3 - Parsing

Parsing

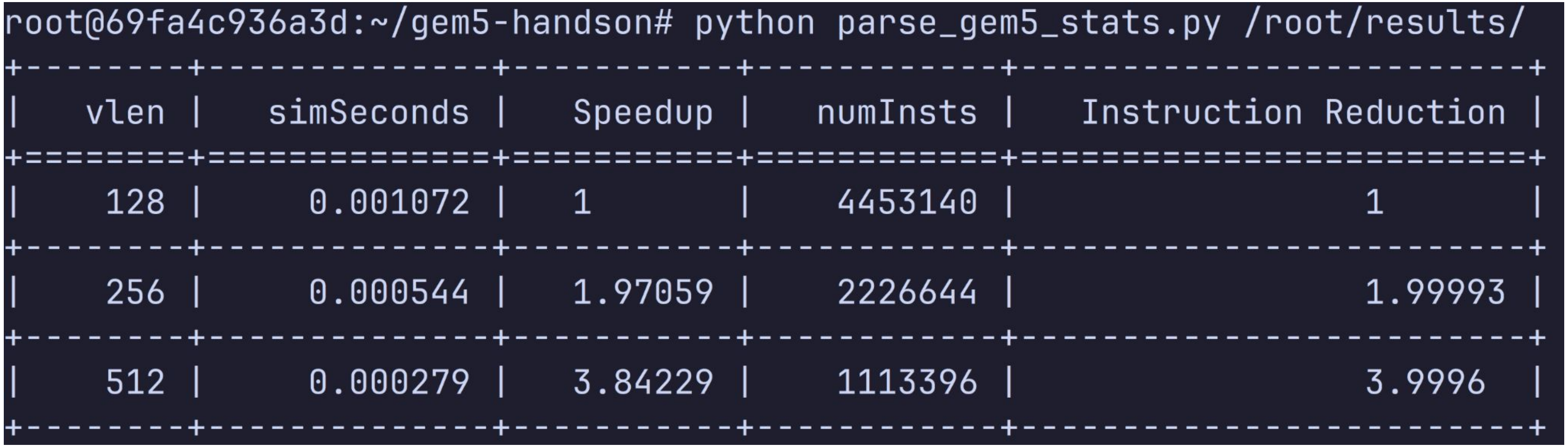
- The script to parse statistics is in the gitlab repository
    - which we already cloned: `gem5-handson`

```
root@5a52b12bae2d:~# cd gem5-handson

root@5a52b12bae2d:~/gem5-handson# ls

README.md     demo-Dockerfile    parse_gem5_stats.py

root@5a52b12bae2d:~/gem5-handson# apt-get install python3-tabulate

root@5a52b12bae2d:~/gem5-handson# python parse_gem5_stats.py /root/results
```

host container

https://gitlab.bsc.es/aarmejac/gem5-handson

Parsed results



```
root@69fa4c936a3d:~/gem5-handson# python parse_gem5_stats.py /root/results/
+--------+--------------+-----------+------------+-----------------------+
|   vlen |   simSeconds |   Speedup |   numInsts |   Instruction Reduction |
+========+==============+===========+============+=======================+
|    128 |     0.001072 |   1       |    4453140 |                     1 |
+--------+--------------+-----------+------------+-----------------------+
|    256 |     0.000544 |   1.97059 |    2226644 |               1.99993 |
+--------+--------------+-----------+------------+-----------------------+
|    512 |     0.000279 |   3.84229 |    1113396 |               3.9996  |
+--------+--------------+-----------+------------+-----------------------+
```

- If you just want a machine to run this type of GEMMs…
  - Would you implement a VLEN of 512 bits or a VLEN of 128 bits?

4 - Visualization for performance understanding (if time permits)

Gem5 trace generation infrastructure

- Tracing can be enabled by passing debug flags to the gem5 binary:

```
gem5.opt --debug-flags=Fetch --debug-file=trace.out.gz
```

- Generates a file in the output directory named trace.out.gz
    - Which contains the trace generated by the Fetch flag
- gem5 comes with a large number of debug flags
    - You can also add your own

Generating traces to visualize the pipeline

- The O3PipeView debug flag enables visualization of the execution pipeline

```
root@5a52b12bae2d:~# gem5.opt --debug-start=187742000 --debug-end=195000000
--debug-flags=O3PipeView,O3CPUAll --debug-file=o3pipeview.gz -d /root/results/vlen-512-trace
/root/gem5/configs/example/riscv-se.py rvbench_gemmopt 128 -v 512

root@5a52b12bae2d:~# zcat /root/results/vlen-512-trace/o3pipeview.gz | grep O3Pipe | head -n 4

O3PipeView:fetch:81000:0x00010632:0:8:c_li a3, 0

O3PipeView:decode:0

O3PipeView:rename:0

O3PipeView:dispatch:0
```

https://gitlab.bsc.es/aarmejac/gem5-handson

Konata pipeline viewer

- Konata is an instruction pipeline visualizer for gem5-O3PipeView formats

Getting Konata

- Download a pre-built binary from here (unzip and binary is inside)
    - Direct download links in the README

Using Konata

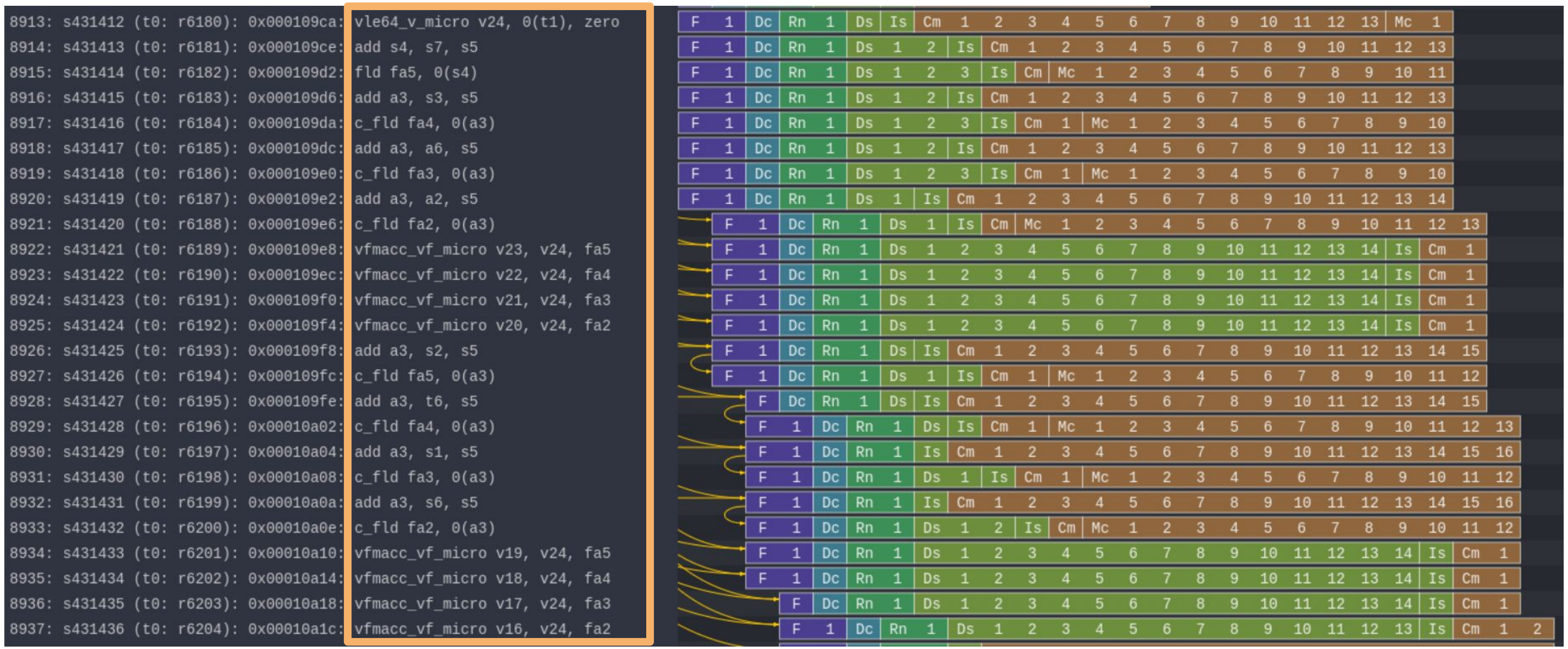
- Copy the results directory from container to host
- Launch Konata and load the trace file from menu or drag&drop

```
$ docker cp container_name:/root/results ./

$ ls results

vlen-128      vlen-256      vlen-512      vlen-512-trace

$ konata          # this opens a konata window
```

host

Konata viewer - Opening the trace

- When the Konata window opens, chose the trace to load from the File option in the top-left



```
8913: s431412 (t0: r6180): 0x000109ca: vle64_v_micro v24, 0(t1), zero
8914: s431413 (t0: r6181): 0x000109ce: add s4, s7, s5
8915: s431414 (t0: r6182): 0x000109d2: fld fa5, 0(s4)
8916: s431415 (t0: r6183): 0x000109d6: add a3, s3, s5
8917: s431416 (t0: r6184): 0x000109da: c_fld fa4, 0(a3)
8918: s431417 (t0: r6185): 0x000109dc: add a3, a6, s5
8919: s431418 (t0: r6186): 0x000109e0: c_fld fa3, 0(a3)
8920: s431419 (t0: r6187): 0x000109e2: add a3, a2, s5
8921: s431420 (t0: r6188): 0x000109e6: c_fld fa2, 0(a3)
8922: s431421 (t0: r6189): 0x000109e8: vfmacc_vf_micro v23, v24, fa5
8923: s431422 (t0: r6190): 0x000109ec: vfmacc_vf_micro v22, v24, fa4
8924: s431423 (t0: r6191): 0x000109f0: vfmacc_vf_micro v21, v24, fa3
8925: s431424 (t0: r6192): 0x000109f4: vfmacc_vf_micro v20, v24, fa2
8926: s431425 (t0: r6193): 0x000109f8: add a3, s2, s5
8927: s431426 (t0: r6194): 0x000109fc: c_fld fa5, 0(a3)
8928: s431427 (t0: r6195): 0x000109fe: add a3, t6, s5
8929: s431428 (t0: r6196): 0x00010a02: c_fld fa4, 0(a3)
8930: s431429 (t0: r6197): 0x00010a04: add a3, s1, s5
8931: s431430 (t0: r6198): 0x00010a08: c_fld fa3, 0(a3)
8932: s431431 (t0: r6199): 0x00010a0a: add a3, s6, s5
8933: s431432 (t0: r6200): 0x00010a0e: c_fld fa2, 0(a3)
8934: s431433 (t0: r6201): 0x00010a10: vfmacc_vf_micro v19, v24, fa5
8935: s431434 (t0: r6202): 0x00010a14: vfmacc_vf_micro v18, v24, fa4
8936: s431435 (t0: r6203): 0x00010a18: vfmacc_vf_micro v17, v24, fa3
8937: s431436 (t0: r6204): 0x00010a1c: vfmacc_vf_micro v16, v24, fa2
```

https://gitlab.bsc.es/aarmejac/gem5-handson

Wrap up

Wrap-up

- gem5 is a detailed full-stack computer architecture simulator
  - Led by academia with a lot of recent contributions from industry
- RISC-V support in gem5 is now very stable
  - Last year we managed to upstream the RISC-V vector ISA support
  - Many bugfixes and new features added since!
  - gem5 v25.0 just released
- The gem5 standard library is a great starting point to perform out-of-the-box simulations
- Repository with materials and container are public
  - The container will be available for download for at least another week

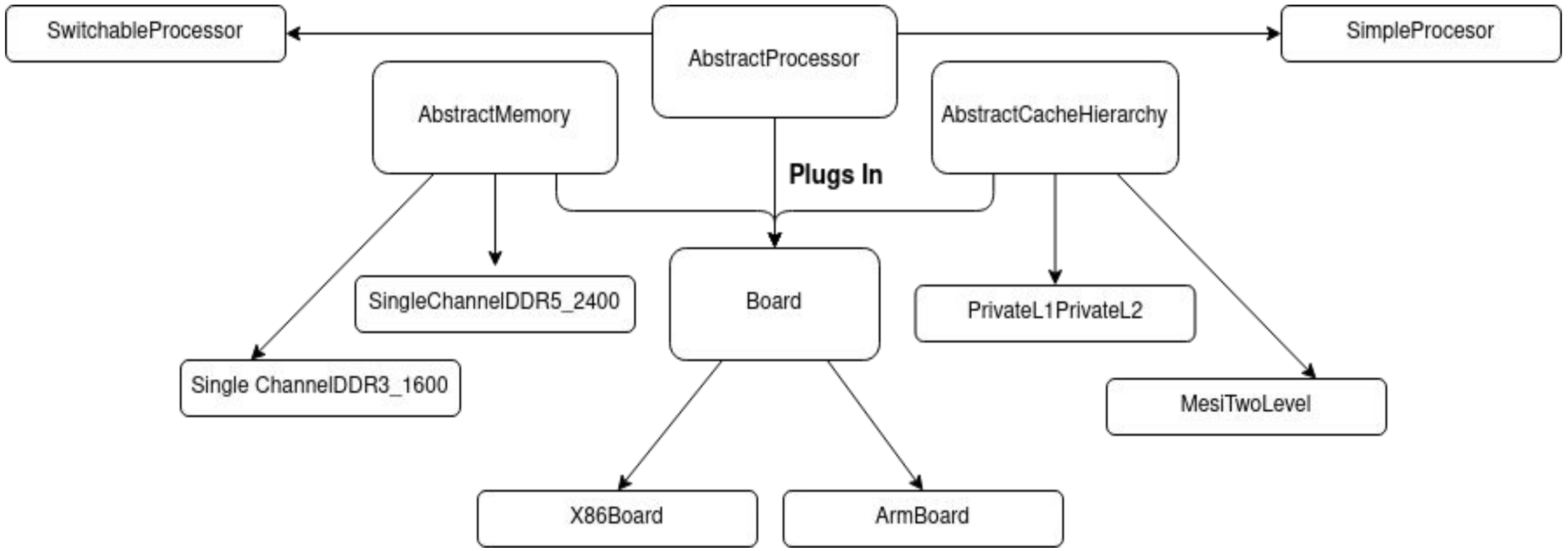
https://gitlab.bsc.es/aarmejac/gem5-handson

Backup

Kinds of simulation (details)

- Functional simulation - referred as emulation
  - Executes programs correctly. Usually no timing information
  - Used to validate correctness of compilers, etc.
  - RISCV Spike, QEMU, RARS, and gem5 ‘atomic’ mode
- Trace based
  - Generate addresses/events and re-execute
  - Can be fast (no need to do functional simulation). Reuse traces
  - If execution depends on timing, this will not work!
  - “Specialized” simulators for single aspect (e.g. just cache hit/miss)
- Instrumentation based
  - Often binary translation. Runs on actual hardware with callbacks
  - Like trace based. Not flexible for new ISAs.
  - DynamoRIO (RISC-V, Arm), PIN (x86), NVBit (nvidia)

The metaphor: Plugging components together into a board



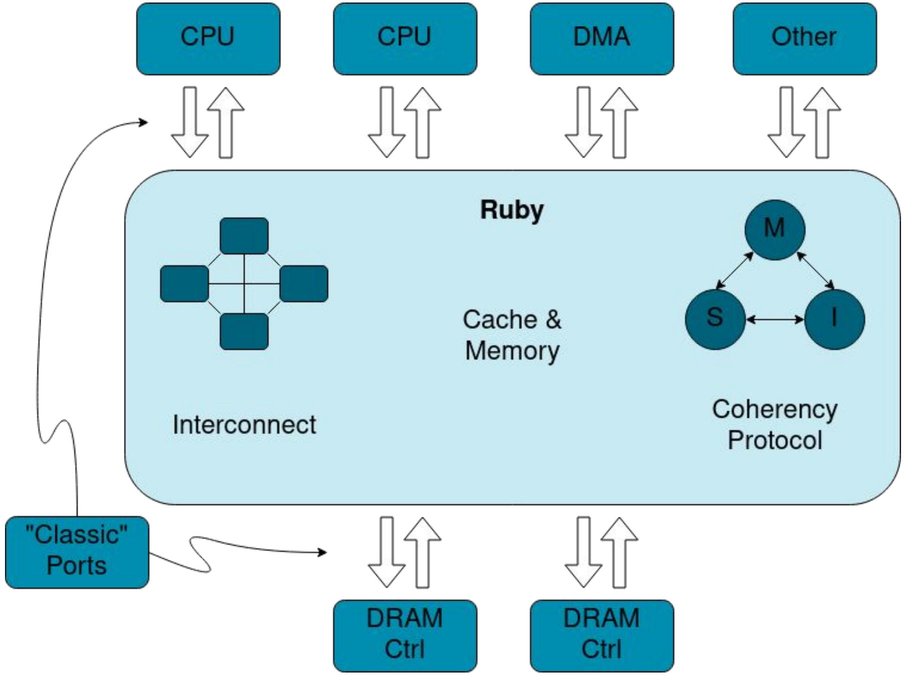
https://gitlab.bsc.es/aarmejac/gem5-handson

Modeling caches

Two types of cache models in gem5:

1. **Classic Cache**: Simplified, faster and less flexible.

2. **Ruby**: Models cache coherence in detail
   a. Coherence controller
   b. Caches + Interface
   c. Interconnect



https://gitlab.bsc.es/aarmejac/gem5-handson

Overview

gem5 compiled binary (gem5.opt)

- Can be thought of as a C++ program that interprets a Python script (Config file)
- Contains the object code of the c++ models
- Gives the script the **m5** module which provides the interface between the configuration script and the gem5 simulator

Simulation configuration (config.py)

- Instantiation of SimObjects
- Sets the parameters for each SimObject
- Specifies the connections between SimObjects
    - hierarchy of caches, core clusters, etc.
- stdlib provides components that wrap models into a standard API

https://gitlab.bsc.es/aarmejac/gem5-handson

Gem5 trace generation infra

gem5 DPRINTF

- We can generate traces for any interaction happening in gem5 between any component
- This is done by inserting DPRINTF statements in the source code

gem5/src/cpu/o3/iew.cc

```
    for (; inst_num < insts_to_execute;
          ++inst_num) {

        DPRINTF(IEW, "Execute: Executing instructions from IQ.\n");

        DynInstPtr inst = instQueue.getInstToExecute();

        DPRINTF(IEW, "Execute: Processing PC %s, [tid:%i] [sn:%llu].\n",
                inst->pcState(), inst->threadNumber,inst->seqNum);
```

https://gitlab.bsc.es/aarmejac/gem5-handson