

©Copyright 2026

Naama Amiel

Learn You a gem5:
Development of a Self-Paced Online Course
for the gem5 Computer Architecture Simulator System

Naama Amiel

A Master's Thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2026

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science and Engineering

University of Washington

Abstract

Learn You a gem5:
Development of a Self-Paced Online Course
for the gem5 Computer Architecture Simulator System

Naama Amiel

gem5 is a widely-used computer architecture simulator in academic research, industry work, and computer architecture education. It is currently facing an on-boarding problem for novice researchers, who find that existing materials emphasize the breadth of gem5's features rather than the depth required to get started. Thus, the learning curve becomes too steep for those with an undergraduate-level understanding of computer architecture and no prior simulation experience. *Learn You a gem5* is a free, self-paced online course for learning gem5 targeted at novice researchers. Its design, built from a set of targeted learning objectives, supports student learning using exercises, projects, accessible visual aides, and a jargon-free and familiar instructional tone. *Learn You a gem5* is a living document designed to be revised, edited, and added to as gem5 continues through active development.

TABLE OF CONTENTS

	Page
Chapter 1: Introduction	1
1.1 Computer Architecture Simulation	1
1.2 Teaching Computer Science	3
1.3 Creating an Online gem5 Course	4
Chapter 2: Related Work	5
2.1 The Learning gem5 Project	5
2.2 gem5 Community Support	7
Chapter 3: The <i>Learn You a gem5</i> Project	9
3.1 Project Overview	9
3.2 From Bootcamp to Roadmap	10
3.3 Writing Lessons	15
3.4 Visual Aides	17
Chapter 4: Discussion and Future Work	19
4.1 Lessons Learned	19
4.2 Future Work	20
Bibliography	22
Appendix A: 2024 Davis Bootcamp Prior Knowledge Checklist	24
A.1 Computer Organization	24
A.2 Computer Architecture I	25
A.3 Computer Architecture II	26
A.4 Programming Experience	27
A.5 Software	30
A.6 Miscellaneous/Other	31

Appendix B: 2024 Davis Bootcamp Derived Learning Objectives	33
B.1 Introduction	33
B.2 Using gem5	35
B.3 Developing gem5 Models	39
Appendix C: Revised Learning Objectives	43
C.1 Introduction	43
C.2 Beginner Simulation with gem5	44
C.3 Intermediate gem5 Features	46
C.4 Developing gem5 Models	48
C.5 Advanced gem5 Features	51

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to the Paul G. Allen School of Computer Science and Engineering and especially to her advisor, Dr. Elba Garza, whose mentorship and support made this project possible.

The author is immensely grateful to the Learning gem5 community including Jason Lowe-Power, Mahyar Samani, and Ayrton Chilibeck, who have all provided invaluable feedback, assistance, and advice. Their guidance and contributions helped shape this project into what it is.

DEDICATION

To each and every one of my two-thousand and thirty-six current and former students whose journey of learning I had the privilege of supporting, if only for a quarter or two.

Chapter 1

INTRODUCTION

1.1 Computer Architecture Simulation

1.1.1 The *gem5* Simulator System

The *gem5* Simulator System [2, 11] (“*gem5*”; intentionally written with a lower-case ‘g’) is an open-source, modular computer architecture simulator used in academic and industry research, as well as in teaching. Computer architecture simulation is invaluable in these arenas due to the time and expenses associated with developing hardware; simulation allows for a high-fidelity approximation of prototypes and ideas that would have otherwise taken months or years (and potentially thousands of dollars) to develop. *gem5*’s wide array of supported features (with more in active development) means it is a desirable simulator option for computer architects; as a result, *gem5* has been cited in over six thousand academic papers, including over 500 papers since 2025.

gem5 is distributed as source code on a public GitHub repository; users independently set up their environment, install dependencies, clone the code, and build the system. It is comprised of a Python front-end, a C++ back-end, and several domain-specific languages for specific functions (for example, SLICC is a domain-specific language for defining cache coherence protocols). Thousands of files make up the simulator, with more files created to drive simulation or create components and workloads. At a high level, users configure *gem5* simulations by writing a Python *configuration script* that declares the components of the system, sets parameters such as simulation time, and defines other simulation behavior. To create unique simulation models (*e.g.*, a new cache replacement policy), users define *SimObjects* in new C++ classes. Users may also make use of the *gem5 standard library*, a set of Python classes that define ready-to-use components (for example, an out-of-order processor or a three-level cache hierarchy), and *gem5 resources*, a large repository of workloads, disk images, and other supporting files for simulation.

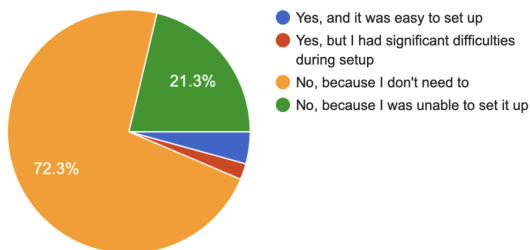


Figure 1.1: “Are you using the provided Docker images as part of your gem5 setup?”

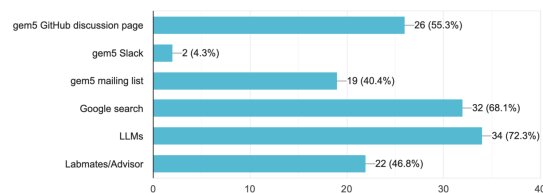


Figure 1.2: “Where/whom do you go to for debugging advice for gem5? (Select all that apply.)”

1.1.2 The On-boarding Problem

gem5’s wide applicability across computer architecture leads to a diverse user base, from seasoned academics and industry professionals to beginning graduate researchers and undergraduate computer science and engineering students. This presents a challenge for developers and documentation writers: who is the “true” target audience for gem5, and how can introductory materials tackle a potential wide gap in prior knowledge?

Along with this breadth of user base, community data shows early challenges in learning gem5. In an anonymous community survey distributed across gem5 communication channels (GitHub discussion page, Slack workspace, mailing list, etc.), respondents identified difficulties in gem5 setup: 21.3% of the 47 respondents could not set up the provided Docker images to run gem5, and another respondent experienced eventual success with significant difficulties along the way (see Figure 1.1). A vast majority (93.6%) of respondents set up their environment on their own, and their most common source of help for debugging gem5 were large-language models (LLMs) (for the latter, see Figure 1.2).

Respondents could optionally leave comments on the survey, many of which echoed difficulty or lack of support:

- “My labmate is unable to setup gem5 on his Mac, and he hasn’t able to fix it.”
- “My whole Linux system used to crash and I had to re-install again and again”
- “Some things are poorly or not at all documented, in which case I try to understand

by reading the source code.”

- “I wish that the information about docker images was easier to find.”
- “The python / C++ split is very difficult to comprehend.”
- “Delete any old unusable documentation. Very confusing.”
- “I find that a lot of the python side of gem5 lack documentation, so crafting your own configuration scripts is quite difficult.”
- “The standard library is so underdocumented. Even setting up a 3 level cache system is a nightmare [...] it’s no wonder that in my field, everyone has left for Champsim.”
- “The documentation does not go into nearly enough detail to do real work with, and all the knowledge is too scattered”

All of these experiences, while not necessarily representative of the breadth of the gem5 user base, reveal a lack of support for getting started with gem5 even among active members of the community who chose to complete this survey. For those just getting started with computer architecture research, this lack of support is even more profound, especially when considering that many novice researchers do not have a strong computer architecture background and may not know the basics of simulation. This becomes a *severe* on-boarding problem for novice researchers that is *not directly addressed* by gem5’s documentation.

1.2 Teaching Computer Science

A successful remedy to this on-boarding problem must be rooted in good pedagogy. Computer science pedagogy describes a set of philosophies, techniques, and strategies for teaching computer science. Two relevant concepts are introduced below.

The Taxonomy of Educational Objectives, more commonly known as Bloom’s Taxonomy [4, 8], is a framework for categorizing learning objectives into tiered cognitive processes. The most revised version of the taxonomy lists these processes as follows (from least to most complex): remembering (factual recall), understanding (comprehension of material), applying (using a procedure in a given situation), analyzing (describing the relationship between constituent parts and the whole), evaluating (assessing based on given standards), and creating (synthesizing information into a new product). Bloom’s Taxonomy provides a

shared language and conceptual tool to describe the breadth and depth of a course’s learning outcomes [8]. It is an integral part of backward design: the process of defining the overall learning outcomes of a course or unit first, then moving backward to creating individual assessments and assignments to support that learning [14].

Pedagogy for massive open online courses (MOOCs) is also informative for this problem. MOOCs are highly available and thus flexible to student needs but often struggle to retain learners [9]. Evidence shows that MOOCs benefit from clearly stated learning outcomes, simple and unambiguous language, and modern up-to-date materials; these elements are inconsistently present in modern MOOCs [5].

1.3 Creating an Online *gem5* Course

The *Learn You a gem5* project directly tackles *gem5*’s on-boarding problem and aims to break down the common barriers that new users encounter. It is an open-source online course made by and for novice researchers who possess only a modest (upper-level undergraduate) computer architecture background. As such, *Learn You a gem5* focuses not only on the mechanical aspect of *gem5* setup and early simulation, but also on the nature of simulation as a whole, taking care at every step to ensure the student does not feel alienated by new domain-specific terminology. *Learn You a gem5* incorporates known computer science pedagogical techniques with the goal of centering learning rather than tool documentation. It (and its name) is inspired by the free online resource *Learn You a Haskell for Great Good!* [10], which similarly aims to introduce readers to the Haskell programming language in a welcoming and accessible way.

This paper documents the development of the *Learn You a gem5* project. First, existing on-boarding and support solutions are explored; then, the creation process and features of the *Learn You a gem5* online course are described in detail; lastly, lessons from the development process and future directions are considered.

Chapter 2

RELATED WORK

2.1 *The Learning gem5 Project*

gem5’s existing educational resources are concentrated in the *Learning gem5* project, written and maintained by Jason Lowe-Power. These free and publicly-available materials were created with the intention of bridging the gap between new and experienced gem5 users; prior to this work, this gap was anecdotally measured in years [11]. The *Learning gem5* project contains online resources for getting started with gem5, recordings of in-person gem5 bootcamps, and archived presentation materials from gem5 workshops held at conferences over the years.

2.1.1 *Online Resources*

As of the time of writing, the online *Learning gem5* book¹ contains three sections: Getting Started, Modifying/Extending, and Modeling Cache Coherence with Ruby. Chapters within sections are structured as web pages consisting largely of explanatory text and code or output snippets. These online materials address some of the challenges with gem5’s onboarding, but they are (as the section titles above suggest) incomplete and not written as a single cohesive course. Despite the presence of these materials online for a number of years, the learning gap identified in the previous chapter persists. These online materials also serve as the basis for in-person bootcamps.

2.1.2 *In-Person gem5 Bootcamps*

In 2024, two in-person five-day gem5 bootcamps were held: one at UC Davis, CA, USA in late July², and another at the University of Buenos Aires, Buenos Aires, Argentina in

¹https://www.gem5.org/documentation/learning_gem5/introduction/

²<https://www.gem5.org/events/bootcamp-2024>

late November³. Both bootcamps covered similar topics, focusing on the *breadth* of features supported by gem5. As in-person events with limited attendance capacity, these bootcamps are inherently less accessible to the community at large—to mitigate this, archives of both are available on the gem5 YouTube channel⁴.

The Davis bootcamp targeted both junior computer architecture researchers (described as “early graduate students”) and professionals, with junior researchers encouraged to attend the full five days and professionals recommended to attend the final two days only. As such, the bootcamp attempts to walk a difficult balance between addressing the experience and needs of two very different target audiences. While the Davis bootcamp serves as an invaluable repository of gem5 knowledge, it suffers from several pain points. First, it assumes a significant body of prior knowledge from students (including knowledge specific to computer architecture simulation). While a larger base of prior knowledge can create a positive cycle of learning as old knowledge acts as a foundation for new knowledge, gaps in prior knowledge between students may result in growing disparities in student achievement [12]. The larger the set of assumed knowledge, the more likely it is that some students fall through the gaps. Early graduate students who are more likely to have a smaller base of prior knowledge are therefore especially at a disadvantage.

Second, the Davis bootcamp presents content in an ordering that is arguably fragmented. For instance, a single session titled “Running Things in gem5” covers the difference between full-system and syscall emulation mode, the hypercall interface, cross-compilation, and traffic generators. Cognitive Load Theory (CLT) posits that human working memory is limited; thus, educators must be conscious of how many complex new concepts are introduced to students [13, 7]. A lesson such as “Running Things in gem5” may prove to be too cognitively taxing for students for whom every piece of information presented is brand-new. Third, the bootcamp does not include many opportunities for student interactivity other than coding along with the instructor; interactivity in the form of clickers or other methods of fostering student contribution has been shown to improve learning outcomes [1, 3].

³<https://www.gem5.org/events/boot-camp-latin-america-2024>

⁴<https://www.youtube.com/@gem5>

The Buenos Aires bootcamp aims to improve on the Davis bootcamp by narrowing the coverage scope, incorporating out-of-class assignments, and introducing a clicker poll and Q&A system to lessons. This marks a step up in accessibility to the “early graduate students” demographic, but it struggles to maintain a common theme or motivating example across lessons: the theme of the bootcamp, secure memory, appears in early lessons and the pre-reading as well as one later lesson on ports. Incorporating the example into more lessons as a consistent through-line would have aided in the bootcamp’s cohesion. The Buenos Aires bootcamp was the final full in-person gem5 bootcamp held as of this writing.

2.1.3 gem5 Workshops

gem5 workshops have been held yearly at the International Symposium on Computer Architecture (ISCA) and the IEEE International Symposium on High-Performance Computer Architecture (HPCA), most recently at ISCA 2025⁵. These workshops focus on new developments in gem5, community contributions, and research using gem5; as such, while they are an invaluable record of gem5’s evolution, they are not well-suited as an on-boarding tool for novice researchers (and they are not intended to be).

2.2 gem5 Community Support

An active online community to reach out to for help is an invaluable resource when learning a new tool. The gem5 online community is accessible through multiple means; the most active and publicly available location is the GitHub repository’s discussion page⁶, but gem5 also maintains a Slack community and a mailing list. Unfortunately, the activity levels on the discussion page are relatively low; as two respondents to the community survey point out:

There should be some help center or some mail groups where we could post our issues and receive a timely response. The already available ones don’t provide any response.

⁵<https://www.gem5.org/events/isca-2025>

⁶<https://github.com/orgs/gem5/discussions>

gem5 desperately needs something like a subreddit, discord or IRC to reach out to other users with questions. The documentation does not go into nearly enough detail to do real work with, and all the knowledge is too scattered.

It is clear that these community spaces do not fully cater to the needs of all gem5 users. This leaves a more pronounced need for a publicly-available resource that directly addresses common issues and questions encountered by gem5's users, especially novice researchers learning the tool for the first time.

Overall, the existing resources for gem5 education, while useful in their own right, do not adequately address the needs of novice researchers and do not consider pedagogical best practices in their construction. This leaves a need for an accessible resource dedicated to further flattening the learning curve for those who need to construct gem5 knowledge from the ground up instead of transferring relevant knowledge from other simulators or research experience.

Chapter 3

THE *LEARN YOU A GEM5* PROJECT

3.1 *Project Overview*

The *Learn You a gem5* project was initiated in response to gem5’s on-boarding problem and the gaps created by previous education efforts, especially in-person bootcamps. It is a publicly-available, free, self-paced online course for getting started with gem5, taking students from their first configuration script to their first custom simulation model. *Learn You a gem5* directly targets the demographic of novice researchers, assuming as little prior knowledge as possible and connecting students to short summaries and external resources for any knowledge they may be lacking. The course also includes accessible visual aides to support student understanding and explicit opportunities for coding exercises and projects. It is written using plain, direct, familiar language to remove barriers to understanding.

The course is organized as a series of lessons written in the Markdown language. Each lesson is accompanied by one or more workspace files ranging from fill-in-the-blank configuration scripts to skeleton files for writing new SimObjects. Solutions are provided for all exercises. The course is divided into five units:

1. **Introduction:** This unit is dedicated to motivating the need for computer architecture simulation, familiarizing students with gem5, and setting students up to simulate with pre-written configuration scripts.
2. **Beginning Simulation:** This unit is dedicated to “toggling” the parameters of a simulated system. Students transition from using and manipulating pre-written standard library components to writing their own standard library components by wrapping existing C++ models in Python classes.
3. **Intermediate Features:** Now that students have a strong foundation in the basics of gem5 simulation, this unit introduces more complex features of simulation such as

traffic generation, hypercalls, and full-system vs. syscall emulation mode.

4. **Developing gem5 Models:** This unit focuses on the transition between using gem5’s pre-written C++ models to writing new models for different simulation goals. Due to the increasing complexity of material in this unit, it is divided into three tracks depending on the student’s research focus:

- *Processors Track:* Students explore gem5 models through the lens of branch prediction, write their own branch predictor SimObject, and learn the fundamentals of making changes to ISAs in gem5.
- *Caches Track*:* Students explore gem5 models through the lens of cache replacement policies, write their own replacement policy SimObject, and learn the fundamentals of cache coherence using Ruby caches.
- *Memory Track*:* Students explore gem5 models through the memory system and write a gem5 SimObject that interacts with ports.

Tracks marked with an ‘’ are outlined but not implemented as of the writing of this document.*

5. **Advanced gem5 Features:** This unit focuses on features of gem5 that are not fundamental to simulation but are nevertheless commonly used by researchers; this includes acceleration techniques (fast-forwarding, checkpointing, and sampling), concurrent simulation, and more.

Learn You a gem5 was written with pedagogical frameworks in mind and thus was created using backward design [14]. The design and implementation process for *Learn You a gem5* is detailed below.

3.2 From Bootcamp to Roadmap

Due to its expansive coverage, the 2024 Davis bootcamp was selected as the foundation for the new online course. The Davis bootcamp was composed from the efforts of many graduate students and other gem5 collaborators, and as a result did not have a centralized

gem5 includes support for configuring cache coherence protocols. Students are expected to be able to:

- Explain the Single-Writer Multiple-Reader (SWMR) invariant,
- Explain how cache coherence protocols such as MSI, MESI, and MOESI address the issue of cache coherence,
- Explain what directories are in the context of cache coherence,
- Define false sharing,
- Explain how the CHI (Coherent Hub Interface) protocol works.

Figure 3.1: Cache coherence prior knowledge requirements for the 2024 Davis bootcamp.

source of development resources. Thus, the first step towards *Learn You a gem5* was to decompose the Davis bootcamp materials into their educational foundations.

3.2.1 Prior Knowledge

An issue with the Davis bootcamp (as identified above) is the large and largely undefined set of assumed student prior knowledge. To make this implicit requirement explicit, a list of all prior knowledge topics was compiled. Information was categorized as assumed prior knowledge if it was mentioned without explanation (for example, usage of acronyms like “ROI” to mean “region of interest”, mentions of tools such as QEMU, employment of techniques such as power modeling, and applications of concepts such as cache hierarchies) or required for engagement for the bootcamp (for example, Python and C++ programming or knowledge of the command line, git, and Visual Studio Code). These prior knowledge topics were then categorized according to topic and proficiency. The complete list of prior knowledge topics as derived from the Davis bootcamp is provided in Appendix A; Figure 3.1 depicts the “Cache Coherence” subcategory under the “Computer Architecture II” prior knowledge category.

Even on its own, this list is a valuable addition to future learning materials: educators can quickly gather which topics they are able to add without expanding the list, and students can feel confident that they meet the requisite knowledge to succeed or seek out resources

Computer Organization

	Measurement Terminology	Cache Organization	Virtual Memory	Multiprocessing/threading
Standard Library		Parameters configured		
gem5 Resources				
Running in gem5		Discussion of miss rate		
Cores				
Cache hierarchies				
Memory				
Full system				
Accelerating Simulation				
Sampling				
Modeling Power				
Multisim				

Figure 3.2: Knowledge dependencies of computer organization topics in the “Using gem5” unit of the 2024 Davis bootcamp.

Programming Experience

	Python Programming	The Command Line	Makefiles	C++ Programming	GDB
SimObjects					
Debugging gem5					
Event driven simulation					
Ports					
Modeling cores					
Modeling cache coherence					
CHI protocol					
Network with Garnet					

Figure 3.3: Knowledge dependencies of programming topics in the “Developing gem5 models” unit of the 2024 Davis bootcamp.

to fill in the gaps prior to continuing with the course. However, a secondary goal of creating this list was to shrink it as much as possible for *Learn You a gem5*. Assuming a smaller set of prior knowledge topics allows the course to be consumed by a larger number of novice researchers; the goal is to turn away as few learners as possible at the outset. However, trimming this list is not a simple matter of selecting topics and removing them from the bootcamp; prior knowledge requirements build on one another, and early lessons with a knowledge assumption form the foundation for future lessons.

3.2.2 Knowledge Dependencies

To untangle this web of prior knowledge requirements, knowledge dependency charts were created. These charts express, for each subcategory of prior knowledge topics, how strongly each lesson in the bootcamp relies on the subcategory. Figures 3.2 and 3.3 depict two such dependency charts. Darker cell shading corresponds to stronger dependency. These charts allow for quick visual identification of the prior knowledge topics that are the most weakly connected to the bootcamp topics, thus marking them for removal in future iterations. A column with little to no shading indicates a prior knowledge subcategory that can be removed entirely or explained at a high level as part of the lesson itself.

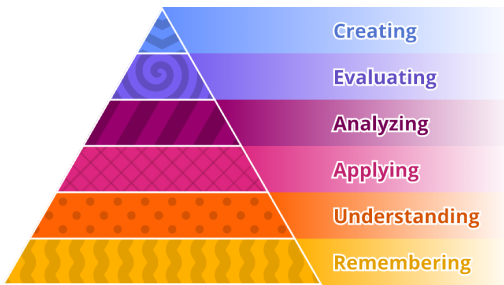


Figure 3.4: The Taxonomy of Educational Objectives (Bloom's Taxonomy).

Debug	Create	Build	Test	Evaluate
Modify	Manage	Code	Predict	Measure
Assemble	Produce		Assess	Determine
Diagram			Interpret	Apply
Use	Modify		Demonstrate	Complete
			Contrast	Compare
Discuss			Describe	
Explain	Run		Trace	List
			Locate	Identify

Figure 3.5: Measurable verbs used in learning objectives by taxonomy level.

3.2.3 Learning Objectives

Learning objectives concentrate the educational goals of the bootcamp into clear, actionable, and measurable targets for student learning. They also act as a “north star” for implementing future versions of materials, as adjusting a learning objective signals which corresponding course adjustments must be made. Learning objectives for each lesson were written separately and ordered by descending priority; the action verbs in each objectives were categorized according to their level in Bloom's taxonomy. This allowed for a large-scale evaluation of the course's teaching priorities and level of cognitive demand. A full list of derived learning objectives is provided in Appendix B.

3.2.4 Revising and Resequencing

The derived educational scaffolds from the Davis bootcamp materials served as a critical point for feedback and reflection. Stakeholders in the *Learn You a gem5* project and the *Learning gem5* project as a whole were consulted for their critiques and insight into the development of the bootcamp, which allowed for an overhaul of the learning objectives and prior knowledge list ahead of *Learn You a gem5* development. This overhaul focused on restructuring fragmented sections, eliminating superfluous learning objectives, and structuring lessons such that learning objectives slowly ascend Bloom's Taxonomy.

As an illustrative example, consider the “Running Things in gem5” section from the Davis bootcamp. The derived learning objectives for this section were as follows:

- Describe the benefits and drawbacks of Syscall Emulation (SE) mode, including what it can and cannot simulate.
- Change exit event behavior by writing a custom event handler and adding it to a Simulation object.
- Modify C++ workloads by adding m5ops annotations.
- Build the m5ops library and link it to a workload.
- Test the performance of a board in simulation using standard library traffic generators.
- Create a new traffic generator by extending the AbstractGenerator class.
- Interpret statistics for simulations that use traffic generators.
- Describe the default behavior of common exit events.
- List the 3 versions of m5ops and the 2 ways they can be added to a simulation.
- Explain the purpose of cross-compilation and the difference between static and dynamic compilation.
- Contrast the behavior of linear cores and random cores.

Through conversations with stakeholders, this lesson was identified to be highly fragmented between three major topics: simulation modes (FS vs. SE), traffic generators, and interpreting simulation statistics. Due to developments in gem5's hypercall interface since the bootcamp, the low-level m5ops objective significantly grew in importance, necessitating a lesson of its own. On the other hand, the cross-compilation objective was relegated to a prerequisite piece of knowledge that could be explained as a short aside during a relevant lesson. While the lesson fragmentation was evident from the list of objectives, conversations about the priorities and intent behind the objectives and recent gem5 changes revealed less obvious directions for improvement. The objectives were trimmed, broken down, and revised into four lessons:

Traffic Generation

- Test the performance of a memory system in-simulation using standard library traffic generators.
- Create a new traffic generator by extending the AbstractGenerator class.

- Interpret statistics for simulations that use traffic generators.

Simulation Modes

- Assess whether SE or FS mode is more appropriate for a particular simulation goal.
- Contrast the benefits and drawbacks of Syscall Emulation (SE) and Full-System (FS) simulation modes.
- Change whether a simulation uses SE or FS mode.

Exit Events

- Explain what an exit event is in gem5.
- Change exit event behavior by writing a custom event handler and adding it to a Simulation object.
- Describe the default behavior of common exit events.

Communication From the Guest Application to the Simulator

- Modify C++ workloads by adding m5ops annotations.
- Build the m5ops library and link it to a workload.
- List the 3 versions of m5ops and the 2 ways they can be added to a simulation.

Note the more descriptive lesson names: with more specificity in learning objectives, each lesson develops a more focused purpose. While not every lesson was decomposed into others like “Running Things in gem5” was, all lessons were re-evaluated for their place on Bloom’s Taxonomy, and individual objectives were re-sequenced accordingly. This point in development also marked the creation of the new unit breakdowns as described in section 3.1. A full list of revised learning objectives is provided in Appendix C, and an overview of the revisions is represented in Figure 3.6; overall, lessons became shorter (and thus more focused), and learning objectives more slowly ascend the levels of Bloom’s taxonomy.

3.3 Writing Lessons

Once the revised learning objectives were in place, the development of individual lessons could begin in earnest. Many of the Davis materials (formatted as presentation slides) were

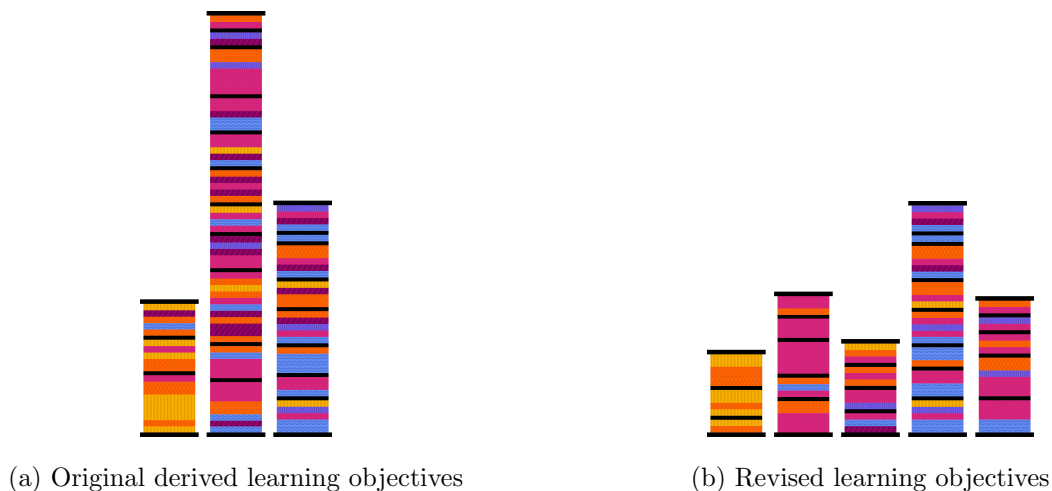


Figure 3.6: Comparison of the Bloom’s Taxonomy breakdown and lesson organization of learning objectives. Individual bars represent separate units. Each colored line represents a single objective; black lines represent a lesson boundary.

reused and incorporated into the new text-based lessons. Converting a set of slides into a lesson involved significant “translation” work to add relevant context and simplify the language being used. All references to external tools (for example, other simulators) were accompanied with relevant links to additional information, and short summaries on weakly-related prior knowledge topics were added as visually-distinct “tips”. Longer primers on foundational topics students were less likely to know (*e.g.*, cache coherence) were added as collapsible “tips”, allowing experienced students to skip them while keeping novice students on track.

An important focus for lesson development was introducing more opportunities for student engagement; this was achieved in a lightweight fashion by taking advantage of disclosure elements that are hidden by default. A typical student exercise begins with a visible demonstration of a gem5 feature—for example, parameterizing a C++ model by wrapping it in a Python class. The student is then directed to attempt a similar task on their own—to continue the example, parameterizing a different model. The answer (usually consisting of a code snippet and an explanation, if needed) is hidden behind a closed disclosure element. Students can therefore progress through the exercise at their own pace, including scrolling

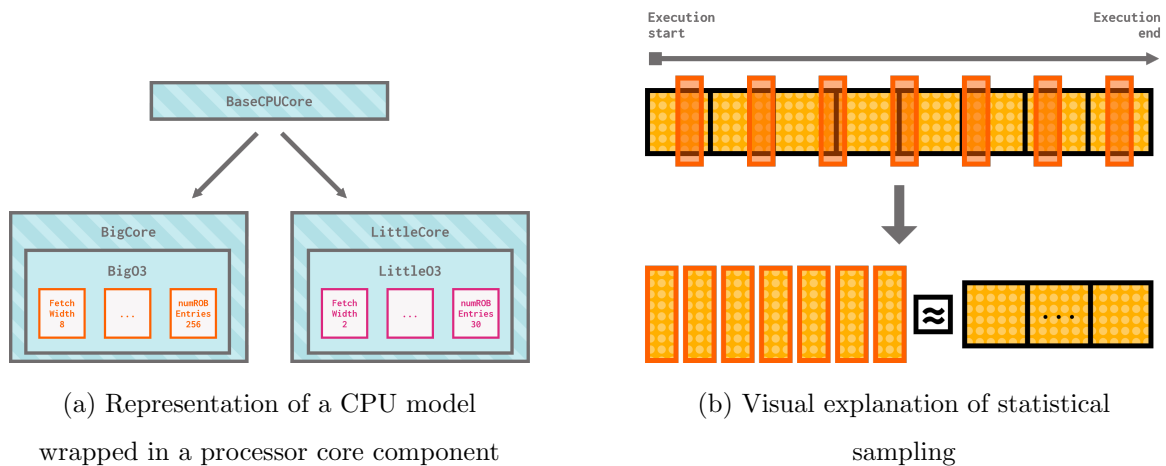


Figure 3.7: Visual aides to support student exercises (3.7a) and complicated simulation procedures (3.7b).

further in the lesson without accidentally revealing the correct answer. When the student is ready, they may click on the disclosure element to reveal the answer.

The fourth unit of the course, which is focused on developing gem5 models, is split into three tracks—a divergence from the original bootcamp. These tracks cover similar high-level learning objectives through the lens of the area of computer architecture most relevant to the student’s research. Students may choose to complete one track in order to advance to the advanced topics unit, or they may complete more than one track to gain a more complete understanding of gem5’s capabilities. Each track is *project-based* and culminates with the student creating their own gem5 model with limited scaffolding from the lesson itself. Useful hints and suggested workflows, as well as a sample solution, are available under closed-by-default disclosure elements.

3.4 Visual Aides

Text-based lessons are complemented with visual aides. Alongside newly-created visuals, the existing visuals from the Davis bootcamp were re-created to improve readability, remove extraneous information, and adhere to a consistent visual style. This visual style is colorblind-friendly, distinguishing emphasized details by more than just color and using an

accessible color palette; this palette is pulled from the IBM Design Library¹. Additionally, all figures were verified for clarity using Color Oracle, a free color-blindness simulator [6]. All visual aides include appropriate alternative text (alt text), a descriptive summary communicating the meaning of an image, for students who rely on screen readers. Visuals were created using a simple vector editor and following simple design principles that can be easily reproduced by future contributors.

Figure 3.7 presents a sample of the visual aides created for *Learn You a gem5*. Visual aides assist in breaking down student exercises into digestible pieces: for example, gem5’s composition of C++ models and Python components is a common pain point for students. By visually representing the models and components as students create them, this confusion can be alleviated. Visual aides are also used to represent abstract or complicated procedures in a more concrete way: for example, the sampling lesson discusses targeted and statistical sampling methods, which students are unlikely to have encountered before. Representing these procedures visually fulfills the high-level understanding students need to make intelligent use of these tools. Overall, visual aides break up the textual monotony of lessons and provide an alternative mode of learning for visually-oriented students.

¹The palette, named ”Color Blind Safe” is now archived.

Chapter 4

DISCUSSION AND FUTURE WORK

4.1 Lessons Learned

4.1.1 Learn as You Teach

The author created *Learn You a gem5* while learning about computer architecture simulation and gem5 for the first time. Perhaps counter-intuitively, this lack of experience helped the development process immensely. First, the author was situated firmly within the target demographic for the eventual course; this provided her with firsthand knowledge of the prior experience, perspectives, and challenges faced by her audience. Second, the author learned about gem5 through the existing learning materials, allowing her to identify and address pain points and confusing explanations much more directly. Any specialized gem5 knowledge required to bring the course to fruition that the author was not able to derive on her own was supplied by knowledgeable collaborators. This resulted in a course that is well-informed and friendly to novice researchers.

4.1.2 Minimize the Set of Assumptions

The target audience for *Learn You a gem5* brings a smaller set of prior knowledge and skills. This is not a deficiency on the part of the student, and is crucial not to treat as such. Learning gem5 is a daunting task grounded in a complex and broad field; novice learners simply require more context to situate their understanding in. As such, explaining references that may seem obvious to the computer architecture research community, or removing them entirely if appropriate, allows students to feel as though the course is meeting them where they are at. In areas where building on prior knowledge is unavoidable, directing students to where they can independently fill in the gaps conveys trust that students are capable of success with gem5.

4.1.3 *Be Plain, Direct, and Familiar*

Learn You a gem5 takes a very direct and familiar instructional tone, intentionally using as little domain-specific jargon as possible and addressing the learner directly in the second person:

“gem5 might be your first time exploring hardware simulation (if that’s true, don’t worry! You’re going to be alright). Naturally, you might be wondering why researchers want to simulate hardware at all.” (Unit I, Section 02)

Thus, the course communicates with the student much like an instructor or teaching assistant would, as opposed to behaving like a piece of documentation. The course reassures students along the way, provides curious learners with avenues to explore the codebase or visit other sources of documentation, and acknowledges that the learning process is gradual. gem5 is already well-documented; the value provided by *Learn You a gem5* lies in its ability to connect with novice learners and support them on their journey.

4.1.4 *Access First*

A primary goal of *Learn You a gem5* was accessibility, and this goal was established very early on. While retrofitting existing materials to meet accessibility best practices is time-consuming, creating materials with accessibility in mind requires a relatively small amount of additional work during development. Development of *Learn You a gem5* was enriched by thinking about the experiences of future students who may consume the course using a screen reader or other accessibility aides, influencing the file and distribution format as well as the color language for the course.

4.2 **Future Work**

4.2.1 *Remaining Project Tracks: Caching and Memory*

While students who select the processors track in Unit IV are able to experience a complete path through the course, the caching and memory tracks are not yet implemented. The lessons in the processors track outline the structure that the caching and memory tracks

should follow, including the sequencing of information, reusable visual aides, and avenues to introduce the project topic before the independent project lesson. The caching project is intended to support students through writing their own cache replacement policy, and the memory project should familiarize students with the way memory requests and ports function in gem5.

4.2.2 Conversion Into an In-Person Format

While in-person bootcamps have not been held in a few years, *Learn You a gem5* provides a foundation for transforming the text-based lessons into bootcamp presentation slides and exercises. Should an opportunity arise to once again hold an in-person gem5 bootcamp, it is recommended to convert text lessons into individual presentations, incorporate tools for student engagement such as clicker polls for smaller lesson exercises, and encourage students to work on the project tracks collaboratively depending on their research interests. This new version of the bootcamp would be directed towards beginners only; professionals may choose to attend workshops or separate events.

4.2.3 Building Advanced Topics and Continuing Maintenance

gem5 currently supports a vast array of advanced simulation features that are not foundational to getting started with simulation but are nonetheless valuable. This includes simulating GPUs, building custom disk images, and using gem5's power models. *Learn You a gem5* is written such that these future lessons (or lesson sequences) can be added to Unit V as an advanced topic. Since gem5 is in active development, some features might be added, changed, or deprecated over time; the structure of the course supports seamless lesson additions in Unit III (intermediate topics) and Unit V (advanced topics), depending on the difficulty and necessity of the feature for beginners. If a feature is changed or deprecated, the learning objectives for each lesson can be used as a guide to determine which lesson(s) need to be modified. *Learn You a gem5* is built to be a living document. It is the author's hope that this resource continues to be maintained as gem5 evolves.

BIBLIOGRAPHY

- [1] Gary Beauchamp and Steve Kennewell. Interactivity in the classroom and its impact on learning. *Computers & Education*, 54(3):759–766, 2010. Learning in Digital Worlds: Selected Contributions from the CAL 09 Conference.
- [2] Nathan L. Binkert, Bradford M. Beckmann, Gabriel Black, Steven K. Reinhardt, Ali G. Saidi, Arkaprava Basu, Joel Hestness, Derek Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib Bin Altaf, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, 2011.
- [3] Lorena Blasco-Arcas, Isabel Buil, Blanca Hernández-Ortega, and F. Javier Sese. Using clickers in class. the role of interactivity, active collaborative learning and engagement in learning performance. *Computers & Education*, 62:102–110, 2013.
- [4] B.S. Bloom, D.R. Krathwohl, and B.B. Masia. *Taxonomy of Educational Objectives: The Classification of Educational Goals. Affective domain. Handbook II*. Number v. 1 in *Taxonomy of Educational Objectives: The Classification of Educational Goals*. Longman Group, 1964.
- [5] Ebru Burcu, Nivedita Viswanathan, and Anum Masood. Teachers’ framework in assessing quality of moocs. In *2025 IEEE Digital Education and MOOCS Conference (DEMOcon)*, pages 1–6, 2025.
- [6] Bernhard Jenny and Nathaniel Vaughn Kelso. Color design for the color vision impaired. page 61–67, Sep. 2007.
- [7] Sam Koudsia and Michael Kirchner. Reducing cognitive overload for students in higher education: A course design case study. *Journal of Higher Education Theory and Practice*, 24(10):97–124, 2024.
- [8] David R Krathwohl. A revision of bloom’s taxonomy: An overview. *Theory into practice*, 41(4):212–218, 2002.
- [9] Johannes Krugel and Peter Hubwieser. Web-based learning in computer science: Insights into progress and problems of learners in moocs. In *Non-Formal and Informal Science Learning in the ICT Era*, pages 51–79. Springer, 2020.
- [10] Miran Lipovača. Learn you a haskell for great good!: A community version.

- [11] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Am-slinger, Matteo Andreozzi, Adrià Arnejach, Nils Asmussen, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jer'onimo Cas-trill'on, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Marjan Fariborz, Amin Farmahini Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Bagus Hanindhito, Andreas Hansson, Swap-nil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kanno, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Tiago M"uck, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc S. Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, An-dreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and 'Eder F. Zu-lian. The gem5 simulator: Version 20.0+. *CoRR*, abs/2007.03152, 2020.
- [12] Kathryn S. McCarthy and Danielle S. McNamara. The multidimensional knowledge in text comprehension framework. *Educational Psychologist*, 56(3):196–214, 2021.
- [13] John Sweller, Jeroen J. G. van Merrienboer, and Fred G. W. C. Paas. Cognitive architecture and instructional design. *Educational Psychology Review*, 10(3):251–296, 1998.
- [14] Grant Wiggins. Understanding by design. *Association for Supervision and Curriculum Development*, 2005.

Appendix A

2024 DAVIS BOOTCAMP PRIOR KNOWLEDGE CHECKLIST**A.1 Computer Organization***A.1.1 Measurement Terminology*

gem5 allows for collection of statistics which require understanding of common measurement terms. Students are expected to be able to:

- Use IEC and SI prefixes (for example, KiB for kibi-bytes and KB for kilobytes, GHz for gigahertz).
- Describe what latency and bandwidth are.

A.1.2 Cache Organization

Caches are one of the key components that can be simulated in gem5. Cache parameters are configured during some demonstrations. Students are expected to be able to:

- Define cache size, block size, and cache associativity,
- Describe common cache statistics such as miss rate,
- Describe the management bits stored in a cache,
- Describe cache write policies (write back, write through, write allocate).

A.1.3 Virtual Memory Basics

gem5 can simulate the operating system, including virtual memory translations. Students are expected to be able to:

- Define the terms page table and TLB (Translation Lookaside Buffer),
- Explain what a page table walk is.

A.1.4 Multiprocessing and Multithreading Basics

gem5 can simulate multiprocessing/multithreading, and a discussion of multithreading and multiprocessing in order to speed up gem5 simulations is included in the bootcamp. Students are expected to be able to:

- Explain the difference between multithreading and multiprocessing,
- Define a context switch.

A.2 Computer Architecture I

A.2.1 Cache Hierarchy

gem5 can simulate a variety of cache hierarchies, and different hierarchies are used in code examples throughout the bootcamp. Students are expected to be able to:

- Define L1, L2, and L3 caches,
- Explain what an I-cache, D-cache, and a unified cache are,
- Explain the difference between a private cache and a shared cache.

A.2.2 Processors/CPU's

CPUs (Central Processing Units AKA processors) and cores are a key simulation component in gem5. Processors are configured and simulated in many code demonstrations. Students are expected to be able to:

- Explain the difference between a core and a CPU,
- Describe a CPU in terms of its clock speed,
- Explain the difference between a single-cycle and a pipelined CPU,
- Explain how a pipelined CPU works,
- Define out-of-order execution and related terminology like ROBs (re-order buffers) and LSQs (load-store queues),
- Explain common CPU metrics such as IPC (Instructions Per Cycle).

A.2.3 ISAs

gem5 supports multiple ISAs (Instruction Set Architectures) and different ISAs are used across code demonstrations. Students are expected to be able to:

- List common ISAs (x86, ARM, RISC-V, MIPS, SPARC),
- Describe at a high level how an instruction is decoded by reading different bit fields of the instruction (ex: the opcode).

A.3 Computer Architecture II

A.3.1 Memory

Memory is a key simulation component in gem5. Memory is described and configured in many code demonstrations. Students are expected to be able to:

- Explain what SRAM and DRAM are,
- Describe how memory is configured (channel/rank/bank/row/column bits),
- List common memory devices ((LP)DDR3/4/5),
- Name media-specific operations for memory (activation, pre-charge, refresh, low-power modes, etc.),
- Name scheduling algorithms for memory read and write requests.

A.3.2 Cache Coherence

gem5 includes support for configuring cache coherence protocols. Students are expected to be able to:

- Explain the Single-Writer Multiple-Reader (SWMR) invariant,
- Explain how cache coherence protocols such as MSI, MESI, and MOESI address the issue of cache coherence,
- Explain what directories are in the context of cache coherence,
- Define false sharing,
- Explain how the CHI (Coherent Hub Interface) protocol works.

A.3.3 Advanced Cache Terminology

Some caching parameters in gem5 are not covered in early computer architecture classes. Students are expected to be able to:

- Explain what banked caches are,
- Explain what cacheclusivity is,
- Describe what a miss status handler register (MSHR) does,
- Explain what prefetching is.

A.3.4 Interconnects and Networks

gem5 allows for creation of custom on-chip network topologies. Students are expected to be able to:

- Define and contrast different types of topologies (buses, crossbars, rings, mesh, torus, etc.),
- Explain what internal and external links are,
- Explain what routers are and name common routing algorithms,
- Explain what virtual channels and networks are,
- Explain what flits and flow control are.

A.4 Programming Experience

A.4.1 Python Programming

The interface to gem5 is Python scripts. Students will be writing Python configuration scripts in order to run gem5 simulations. Students are expected to be able to:

- Explain what a Python interpreter is,
- Use import statements,
- Code with variables,
- Instantiate objects and call object member functions,
- Call functions.

If students choose not to attend the Python background session during the first day of the bootcamp, students are additionally expected to be familiar with:

- Print statements,
- Primitives (integers, floats, strings, booleans),
- Collections (lists, sets, and dictionaries),
- List comprehension (including nested list comprehension),
- Control flow (if-statements, for-loops, while-loops),
- Functions (defining and calling, parameters, return values, typing hints),
- Generators,
- Object-Oriented Design (OOD)
- Classes and objects,
- Member variables and functions,
- Constructors and construction/instantiation,
- Inheritance (base class, derived class) and overriding,
- Abstract classes and methods,
- Encapsulation.

A.4.2 The Command Line

gem5 and related utilities are run from the command line. Students are expected to be able to:

- Run command-line utilities,
- Use common bash commands such as `grep`, `cd`, `touch`, `cp`, etc.,
- Set environment variables.

A.4.3 Makefiles

Pre-made makefiles are used for demonstrations throughout the bootcamp. Students are expected to be able to:

- Read makefiles and describe how they are used for compilation,
- Modify makefiles to change compilation options.

A.4.4 C++ Programming

The backend of gem5 is written in C++. To develop new gem5 models, C++ programming proficiency is required. Students are expected to be able to:

- Write header (.hh) files,
- Use include guards,
- Use include statements,
- Explain and use namespaces,
- Use macros and static inline functions,
- Use intermediate programming constructs (primitives, functions, control flow),
- Describe what structs are,
- Use the const keyword,
- Use standard library data structures (ex: vectors and queues),
- Use OOD in C++
- Write classes and constructors,
- Use initialization lists,
- Code with inheritance,
- Inherit from abstract classes and override virtual functions.
- Print to the console (cout and cerr),
- Program with pointers and references,
- Use assert and static assert statements.

A.4.5 GDB (GNU Debugger)

GDB is used to examine a program while it runs. It is used during a bootcamp session to trace the execution of gem5. Students are expected to be able to:

- Run GDB on an executable,
- Use GDB commands,
- Explain what a breakpoint is.

A.5 Software

A.5.1 Git/GitHub

gem5 is distributed as source code and acquired by cloning a git repository, and the boot-camp uses GitHub codespaces. Students are expected to be able to:

- Set up git on their computers,
- Clone a repository (repo) using git,
- Define the concept of a branch within git,
- Define the concept of a tag within git,
- Create a GitHub account.

A.5.2 SCons

SCons is a tool used to build gem5. Students are expected to be able to:

- Use SCons using the command line.

A.5.3 Packer

Packer is used to automate disk image creation during one demonstration. Students are expected to be able to:

- State what Packer is used for.

A.5.4 QEMU

QEMU is referenced across multiple sessions and used to automate disk image creation during one demonstration. Students are expected to be able to:

- Explain what QEMU is used for.
- Use QEMU commands.

A.5.5 Ubuntu

Ubuntu is a Linux distribution used as the operating system across several demonstrations. Students are expected to be able to:

- Explain what Ubuntu is.
- Recognize what Ubuntu autoinstall is.

A.5.6 Visual Studio Code (VSCode)

Resources refer to Visual Studio Code as the default IDE (Integrated Development Environment) for use in the bootcamp, and all demonstrations are done in Visual Studio Code. Students are encouraged to use Visual Studio Code, though this is not required.

A.6 Miscellaneous/Other

A.6.1 Operating Systems Basics

gem5 can simulate operating systems (OS). Students are expected to be able to:

- Describe the role of the operating system,
- Explain what an interrupt is,
- Explain what a system call (syscall) is,
- List common Linux system calls,
- Explain the concept of deadlock.

A.6.2 Virtual Machines

Required to understand how the gem5 simulator emulates hardware. Students are expected to be able to:

- Explain what a virtual machine is,
- Define the concept of a virtualizer/hypervisor/VMM,
- Define what a guest OS and a host OS are.

A.6.3 Power Modeling

The bootcamp covers gem5’s power modeling capabilities with a short demonstration. Students are expected to be able to:

- Explain what “activation count” power modeling is.
- Identify what McPAT, Wattch, and CACTI are.

A.6.4 Field-Specific Acronyms/Initialisms

Presented in no particular order, this section serves as a reference for acronyms/initialisms that may be spoken without definition. Students don’t need to have a deep understanding of these, but they might find it helpful to look them up if they haven’t heard of them before.

- ASIC (Application-Specific Integrated Circuit)
- GABPS (Graph Algorithm Platform Benchmark Suite)
- RTL (Register Transfer Level)
- ROI (Region Of Interest)
- DIMM (Dual In-Line Memory Module)
- HBM (High-Bandwidth Memory)
- NAS, NPB (NASA Advanced Supercomputing, NAS Parallel Benchmarks)
- IS (Integer Sort), EP (Embarrassingly Parallel)
- BP (Branch Prediction)
- FCFS (First Come First Served)
- FIFO (First In First Out)
- DMA (Direct Memory Access)

Appendix B

2024 DAVIS BOOTCAMP DERIVED LEARNING OBJECTIVES

B.1 Introduction

B.1.1 Overall Objectives

Students will be able to...

- Download and build gem5.
- Explain how gem5 simulates components and the passage of time.
- Create and run simple Python configuration scripts using demo components.

B.1.2 Per-Topic Objectives

Background on Simulation

Students will be able to...

- Describe the purpose and goals of the gem5 simulator.
- Explain why simulation is a useful tool for computer architecture research.
- Describe what a discrete event simulator is and how it simulates the passage of time.
- Identify the predecessor simulators to gem5.
- List gem5's two main abstractions.
- Describe the interaction between gem5's C++ models with its Python frontend.
- Explain the difference between extending and specializing a model.
- Discuss the differences between cycle-level and cycle-accurate simulation.
- Contrast the tradeoffs of functional, instrumentation-based, trace-based, execution-driven, and full-system simulation.

Getting Started with gem5

Students will be able to...

- Download and build the gem5 source code.
- Run pre-existing gem5 Python scripts.
- Locate and interpret the output of gem5 simulator runs.
- Identify important directories in the gem5 code for source code, resources, and boot-camp materials.

Background on Python and gem5

Students will be able to...

- Explain how a Python configuration script is used in conjunction with the gem5 binary.
- Produce correct Python programs that utilize intermediate Python features:
 - Integer and float operations, String printing, and boolean comparisons,
 - Collections including lists, sets, and dictionaries,
 - List comprehension, including nested list comprehension,
 - Control structures, including if-statements, for and while loops,
 - Functions, including parameters, return values, and type hinting,
 - Import statements,
 - Generators,
 - Object-oriented programming, including inheritance, overriding and abstract classes.
- Describe what SimObjects represent within gem5 and which Python constructs and principles are used to implement them.
- Modify a SimObject by extending an existing abstract class to add new functionality.
- Identify the special rules for SimObjects in gem5 that do not apply to standard Python objects.

B.2 Using gem5

B.2.1 Overall Objectives

Students will be able to...

- Modify existing gem5 components (cores, caches, and memory) to achieve desired functionality using object-oriented design practices such as extension and instantiation.
- Create gem5 simulations with customized exit events and annotated workloads.
- Manage existing gem5 code such as the standard library and the resource repository.
- Demonstrate use of simulation acceleration techniques such as KVM fast-forwarding, checkpointing, and sampling.

B.2.2 Per-Topic Objectives

gem5's standard library

Students will be able to...

- Assemble a board for simulation by “plugging in” standard library components and setting a workload.
- Modify parameters of standard library components to achieve desired behavior.
- Create a new simulation component by extending an existing standard library component.
- Describe the functionality the gem5 standard library provides to users.
- Explain the difference between models and components.

The following objectives were skipped in the video as the presenter moved to the next example:

- Contrast Syscall Emulation (SE) and Full-System (FS) simulation modes.
- Demonstrate setting a full-system workload on a board.
- Demonstrate using a simulator and exit events to run a simulation.

Gem5 resources

Students will be able to...

- Demonstrate use of the `obtain_resource` function to download a `gem5` resource.
- Demonstrate running binaries, workloads, and suites in `gem5`.
- Interpret the JSON format of a `gem5` resource file.
- Create a local `gem5` resource and run a simulation using the resource.
- Explain what `gem5` resources are available and what categories they can fall into.

Running things in gem5

Students will be able to...

- Describe the benefits and drawbacks of Syscall Emulation (SE) mode, including what it can and cannot simulate.
- Change exit event behavior by writing a custom event handler and adding it to a `Simulation` object.
- Modify C++ workloads by adding `m5ops` annotations.
- Build the `m5ops` library and link it to a workload.
- Test the performance of a board in simulation using standard library traffic generators.
- Create a new traffic generator by extending the `AbstractGenerator` class.
- Interpret statistics for simulations that use traffic generators.
- Describe the default behavior of common exit events.
- List the 3 versions of `m5ops` and the 2 ways they can be added to a simulation.
- Explain the purpose of cross-compilation and the difference between static and dynamic compilation.
- Contrast the behavior of linear cores and random cores.

Modeling cores in gem5

Students will be able to . . .

- Contrast the simulation behavior of atomic, functional, and timing memory accesses.
- Compare the features and advantages of simple, KVM, O3, and minor CPUs.
- Modify a board by changing the CPU type.
- Evaluate the simulation performance of different gem5 CPU types.
- Modify a CPU's parameters by extending an existing CPU model.

Modeling caches in gem5

Students will be able to . . .

- Contrast the advantages and disadvantages of using classic and Ruby caches in gem5.
- Construct a classic cache hierarchy in gem5 using caches and crossbars.
- Interpret the unique simulation statistics provided by Ruby caches.
- List the Ruby cache components.

Modeling memory in gem5

Students will be able to . . .

- Explain what memory controllers and memory interfaces do in gem5.
- Modify a memory system by instantiating an instance of ChanneledMemory with custom parameters.
- Compare the performance of the SimpleMemory and ChanneledMemory standard library components.
- Modify a memory system by adding a CommMonitor.
- Explain the purpose of address interleaving.

Full system simulation

Students will be able to...

- Create a disk image using Packer and QEMU given the gem5 resources script.
- Modify and build a base disk image to add a custom benchmark.
- List the components simulated by full-system mode.
- Demonstrate running a full-system workload in gem5 using a pre-existing workload resource.
- Demonstrate use of m5term to view gem5's simulation output in the terminal.

Accelerating simulation

Students will be able to...

- Set up KVM fast-forwarding.
- Produce a simulation checkpoint.
- Modify a Simulation object to restore/resume from a checkpoint.
- Contrast the benefits and downsides of using KVM fast-forwarding and checkpointing for accelerating simulation.
- Interpret statistics from an accelerated simulation run.

Sampled simulation with gem5

Students will be able to...

- Demonstrate analyzing a workload with a SimPoint probe, creating checkpoints based on SimPoint analysis, and running SimPoint checkpoints.
- Demonstrate running a simulation with a SMARTS generator.
- Contrast the benefits and drawbacks of targeted and statistical sampling methods.
- Demonstrate running ELFies using gem5.
- Predict overall simulation performance based on SimPoint representative region performance and SMARTS areas of detailed simulation.

- Explain broadly how targeted sampling techniques determine representative regions of a program.
- Explain broadly how statistical sampling techniques determine when to simulate with higher fidelity.

Power modeling

Students will be able to...

- Modify a SimObject by adding a MathExprPowerModel to its instantiation.
- Assess the appropriateness and accuracy of gem5 power models.

Multisim

Students will be able to...

- Demonstrate running multiple simulations using a MultiSim config file.
- Explain why using MultiSim to create parallel instances of gem5 simulations is preferable over multiprocessing via script.

B.3 Developing gem5 Models

B.3.1 Overall Objectives

Students will be able to...

- Create gem5 SimObjects with custom parameters, ports, and events.
- Debug gem5 code using C++ and gem5 utilities.
- Code according to gem5's conventions and best practices.
- Explain how ISAs are implemented independently from CPUs in gem5.
- Explain how Ruby cache coherency protocols are implemented in gem5.

B.3.2 Per-Topic Objectives

SimObject intro

Students will be able to...

- Create a simple SimObject (with a constructor and parameters) from scratch by creating the appropriate Python, C++, and SConscript files.
- Build gem5 using scons and Kconfig.
- Apply gem5 code conventions such as naming conventions, include guards, and best practices when creating new SimObjects.
- Determine which build-time configuration and compiler configuration are appropriate for a research goal.
- Trace m5's `initiate()` function to follow SimObject creation.

Debugging and debug flags

Students will be able to...

- Create a new debugging flag and enable/disable it using a configuration script and a command line option.
- Use `DPRINTF()` statements to print debugging information in the presence of a flag.
- Use C++ `assert` statements and gem5 `fatal` and `panic` statements to verify assumptions and catch user and developer mistakes.

Event-driven simulation

- Create a new event for a SimObject and schedule it in-simulation.
- Incorporate a SimObject as a parameter to another SimObject.
- Create a compound flag that enables/disables multiple other flags.
- Explain at a high level how gem5's event-driven simulation is implemented as a series of events and callbacks.

Modeling ports

Students will be able to...

- Create CPU- and memory-side ports for a SimObject and facilitate packet sending and retries between them and their peers.
- Simulate port request/response processing and retry latency using events.
- Measure custom statistics about SimObjects using gem5 statistics.
- Use sender state to store additional information about packets.
- Explain at a high level how ports (RequestPort and ResponsePort) and packets facilitate communication in gem5.

Modeling cores

Students will be able to...

- Explain at a high level how gem5 implements ISA and CPU independence using StaticInst, DynamicInst, and ExecContext.
- Describe the instruction decoding process in the ISA parser.
- Modify the ISA decoder to add a custom instruction.
- Trace the execution path of a single instruction from the decoding of the instruction object through the instruction's execution.

Modeling cache coherence (CA2)

Students will be able to...

- Debug a cache coherence protocol using protocol trace.
- Diagram cache coherence protocols (such as MSI) as tables of states, events, and actions.
- Complete a partially-filled SLICC specification of a given cache coherence protocol.
- Explain how the elements of a cache state machine (cache memory, message buffers, state, events, in ports, actions, and transitions) interact with each other.

- Explain what the four types of Ruby components (controller models, controller topology, interconnect model, interface) do.

Using the CHI protocol (CA2)

Students will be able to...

- Create a cache hierarchy using gem5's implementation of the CHI protocol and the gem5 standard library.

Modeling the on-chip network with Garnet (CA2)

Students will be able to...

- Implement a custom network topology for a Ruby cache system.
- Modify a simple network to use Garnet.
- Contrast the simulation capabilities of simple and Garnet networks.
- Test the performance of a network topology and make appropriate changes based on the results.

Appendix C

REVISED LEARNING OBJECTIVES

C.1 Introduction

C.1.1 Overall Objectives

Students will be able to...

- Download, build, and run gem5.
- Explain what the gem5 simulator is and what it is built to do.
- Explain why simulation is a useful tool for computer architecture research.

C.1.2 Per-Topic Objectives

Simulation Background

Students will be able to...

- Explain why simulation is a useful tool for computer architecture research.
- Define functional, instrumentation-based, trace-based, execution-driven, and full-system simulation.

gem5 Introduction

Students will be able to...

- Describe the purpose and goals of the gem5 simulator.
- Explain how gem5 differs from cycle-accurate RTL simulators.
- Describe the roles of gem5's C++ backend and Python frontend.
- Identify the predecessor simulators to gem5.

Getting Started with gem5

Students will be able to...

- Download and build the gem5 source code.
- Run pre-existing gem5 Python scripts.
- Explain how a Python configuration script is used in conjunction with the gem5 binary.
- Locate the output of gem5 simulator runs.
- Identify important directories in the gem5 code for source code, resources, and boot-camp materials.

C.2 Beginner Simulation with gem5*C.2.1 Overall Objectives*

Students will be able to...

- Construct a Python configuration script using standard library components and gem5 resources.
- Modify existing gem5 components (cores, caches, and memory) in Python using object-oriented design practices such as extension and instantiation.
- Navigate and interpret the output of gem5 simulations.

*C.2.2 Per-Topic Objectives**Interpreting Simulation Output*

Students will be able to...

- Navigate to the simulation output of simulator runs.
- Examine simulator output in stats.txt to identify desired statistics.

The gem5 Standard Library

Students will be able to . . .

- Construct a board for simulation by “plugging in” standard library components and setting a workload.
- Modify parameters of standard library components to achieve desired behavior.
- Derive a new simulation component by extending an existing standard library component in Python.
- Describe the functionality the gem5 standard library provides to users.
- Explain the difference between models and components.

gem5 Resources

Students will be able to . . .

- Use a variety of existing gem5 resources, including binaries, workloads, and suites.
- Create a local gem5 resource and use it in a simulation.
- Explain what gem5 resources are available and what categories they fall into.

Modeling Cores in gem5

Students will be able to . . .

- Compare the features and advantages of simple, KVM, O3, and minor CPUs.
- Modify a board by changing the CPU type.
- Modify a CPU’s parameters by extending an existing CPU model in Python.
- Contrast the simulation behavior of atomic, functional, and timing memory accesses.
- Compare the simulation performance of different gem5 CPU types.

Modeling Caches in gem5

Students will be able to...

- Contrast the advantages and disadvantages of using classic and Ruby caches in gem5.
- Construct a classic cache hierarchy in gem5 using caches and crossbars.
- Interpret the unique simulation statistics provided by Ruby caches.

Modeling Memory in gem5

Students will be able to...

- Explain what memory controllers and memory interfaces do in gem5.
- Modify a memory system by instantiating an instance of ChanneledMemory with custom parameters.
- Compare the performance of the SimpleMemory and ChanneledMemory standard library components.

C.3 Intermediate gem5 Features*C.3.1 Overall Objectives*

Students will be able to...

- Determine appropriate simulation uses for Syscall Emulation (SE) and Full-System (FS) mode.
- Facilitate communication between the guest application and the simulator using customized exit events and m5ops annotations.
- Create and use traffic generators in place of processor models to simulate memory accesses.

*C.3.2 Per-Topic Objectives**Traffic Generation*

Students will be able to...

- Test the performance of a memory system in-simulation using standard library traffic generators.
- Create a new traffic generator by extending the AbstractGenerator class.
- Interpret statistics for simulations that use traffic generators.

Simulation Modes

Students will be able to . . .

- Assess whether SE or FS mode is more appropriate for a particular simulation goal.
- Contrast the benefits and drawbacks of Syscall Emulation (SE) and Full-System (FS) simulation modes.
- Change whether a simulation uses SE or FS mode.

Exit Events

Students will be able to . . .

- Explain what an exit event is in gem5.
- Change exit event behavior by writing a custom event handler and adding it to a Simulation object.
- Describe the default behavior of common exit events.

Communication From the Guest Application to the Simulator

Students will be able to . . .

- Modify C++ workloads by adding m5ops annotations.
- Build the m5ops library and link it to a workload.
- List the 3 versions of m5ops and the 2 ways they can be added to a simulation.

C.4 Developing gem5 Models

C.4.1 Overall Objectives

Students will be able to...

- Create gem5 SimObjects with custom parameters, ports, and events.
- Debug gem5 code using C++ and gem5 utilities.
- Code according to gem5's conventions and best practices.
- Develop gem5 simulations without relying on standard library components.
- Modify an ISA implementation in gem5.
- Implement Ruby cache coherency protocols in gem5.

C.4.2 Per-Topic Objectives

SimObjects in gem5

Students will be able to...

- Create a simple SimObject (with a constructor and parameters) from scratch by creating the appropriate Python, C++, and SConscript files.
- Build gem5 using SCons and Kconfig.
- Apply gem5 code conventions such as naming conventions, include guards, and best practices when creating new SimObjects.
- Determine which build-time configuration and compiler configuration are appropriate for a research goal.
- Trace m5's `initiate()` function to follow SimObject creation.

Debugging and Debug Flags

Students will be able to...

- Create a new debugging flag and enable/disable it using a configuration script and a command line option.

- Create a compound flag that enables/disables multiple other flags.
- Use `DPRINTF()` statements to print debugging information in the presence of a flag.
- Use C++ `assert` statements and `gem5 fatal` and `panic` statements to verify assumptions and catch user and developer mistakes.

Event-Driven Simulation

Students will be able to . . .

- Explain at a high level how `gem5`'s event-driven simulation is implemented as a series of events and callbacks.
- Create a new event for a `SimObject` and schedule it in-simulation.
- Incorporate a `SimObject` as a parameter to another `SimObject`.

Modeling Ports

Students will be able to . . .

- Create CPU- and memory-side ports for a `SimObject` and facilitate packet sending and retries between them and their peers.
- Simulate port request/response processing and retry latency using events.
- Measure custom statistics about `SimObjects` using `gem5` statistics.
- Use sender state to store additional information about packets.
- Explain at a high level how ports (`RequestPort` and `ResponsePort`) and packets facilitate communication in `gem5`.

ISA Implementation in gem5

Students will be able to . . .

- Trace the execution path of a single instruction from the decoding of the instruction object through the instruction's execution.
- Modify the ISA decoder to add a custom instruction.

- Explain at a high level how gem5 implements ISA and CPU independence using StaticInst, DynamicInst, and ExecContext.
- Describe the instruction decoding process in the ISA parser.

Modeling Cache Coherence with Ruby and SLICC

Students will be able to...

- Debug a cache coherence protocol using protocol trace.
- Diagram cache coherence protocols as tables of states, events, and actions.
- Complete a partially-filled SLICC specification of a given cache coherence protocol.
- Explain how the elements of a cache state machine (cache memory, message buffers, state, events, in ports, actions, and transitions) interact with each other.
- Explain what the four types of Ruby components (controller models, controller topology, interconnect model, interface) do.

Using the CHI Protocol

Students will be able to...

- Create a cache hierarchy using gem5's implementation of the CHI protocol and the gem5 standard library.

Modeling the On-Chip Network with Garnet

Students will be able to...

- Implement a custom network topology for a Ruby cache system.
- Modify a simple network to use Garnet.
- Contrast the simulation capabilities of simple and Garnet networks.
- Test the performance of a network topology and make appropriate changes based on the results.

C.5 Advanced gem5 Features

C.5.1 Overall Objectives

Students will be able to...

- Incorporate acceleration and sampling methods into simulations as appropriate.
- Modify and use customized disk images to meet a simulation goal.
- Use multisim to run multiple simulations in parallel.
- Incorporate power models into gem5 simulations.

C.5.2 Per-Topic Objectives

Accelerating Simulation

Students will be able to...

- Set up KVM fast-forwarding.
- Produce a simulation checkpoint.
- Modify a Simulation object to restore/resume from a checkpoint.
- Contrast the benefits and downsides of using KVM fast-forwarding and checkpointing for accelerating simulation.
- Interpret statistics from an accelerated simulation run.

Sampled Simulation

Students will be able to...

- Demonstrate analyzing a workload with a SimPoint probe, creating checkpoints based on SimPoint analysis, and running SimPoint checkpoints.
- Demonstrate running a simulation with a SMARTS generator.
- Contrast the benefits and drawbacks of targeted and statistical sampling methods.
- Predict overall system performance based on the results of sampled simulation.

- Explain broadly how targeted sampling techniques determine representative regions of a program.
- Explain broadly how statistical sampling techniques determine when to simulate with higher fidelity.

Using and Modifying Disk Images

Students will be able to...

- Use disk image resources in gem5 simulations.
- Explain how disk images are created for use in gem5 simulations.
- Modify and build a base disk image to specialize it for research.

Power Modeling

Students will be able to...

- Modify a SimObject by adding a MathExprPowerModel to its instantiation.
- Assess the appropriateness and accuracy of gem5 power models.

Multisim

Students will be able to...

- Use a MultiSim config file to run multiple simulations in parallel.
- Explain why using MultiSim to create parallel instances of gem5 simulations is preferable over multiprocessing via script.

VITA

Naama Amiel is a computer science educator specializing in computer organization, operating systems, and computer architecture. She earned a Bachelor of Science in Computer Science from the University of Washington, Seattle in June 2025, where she graduated *magna cum laude* and was honored with the Bob Bandes Memorial Award for Excellence in Teaching. She earned her Master of Science from the University of Washington in June 2026.