Teaching Microarchitecture Through Metaphors

Julianna Eum
United States Military Academy
West Point, NY 10996
julianna.eum@usma.edu

Simha Sethumadhavan
Computer Architecture and Security Technology Lab
Columbia University
New York, NY 10027
simha@cs.columbia.edu

ABSTRACT
Students traditionally learn microarchitecture by studying textual descriptions with diagrams but few analogies. Several popular textbooks on this topic introduce concepts such as pipelining and caching in the context of simple paper-only architectures. While this instructional style allows important concepts to be covered within a given class period, students have difficulty bridging the gap between what is covered in classes and real-world implementations. Discussing concrete implementations and complications would, however, take too much time.

In this paper, we propose a technique of representing microarchitecture building blocks with animated metaphors to accelerate the process of learning about complex microarchitectures. We represent hardware implementations as road networks that include specific patterns of traffic flow found in microarchitectural behavior. Our experiences indicate an 83% improvement to understanding memory system microarchitecture. We believe the mental models developed by these students will serve them in remembering microarchitectural behavior and extend to learning new microarchitectures more easily.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education]: Curriculum

General Terms
Design, Human Factors

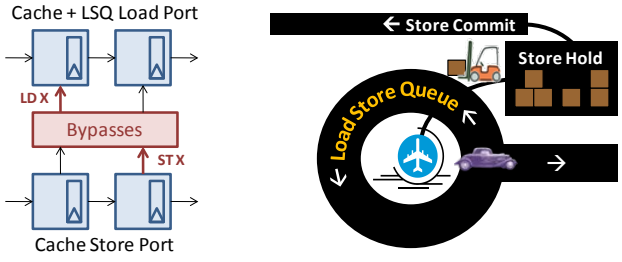
Keywords
Microarchitecture, Pedagogy

1. INTRODUCTION
Single-thread performance continues to be important even as processors become increasingly parallel [8]. To improve single thread performance, computer microarchitects use several techniques to mine parallelism from a stream of instructions. These techniques involve a combination of techniques such as pipelining, caching and speculation. Students well versed in microarchitecture design are invaluable assets for both industry and research.

Processor microarchitecture information is often buried in research papers not easily accessible to most students and instructors. Classroom examples of processor microarchitecture generally remain at the level of simplified, theoretical block diagrams. While these simplified abstractions focus on the core principles, it has been our experience that students struggle to synthesize these concepts. Students struggle when applying these principles in more complex contexts that typically occur in real-world microarchitectural designs. Such design requires concurrent thinking as it primarily deals with multiple interacting events (speculation, pipelining and bookkeeping) that happen in parallel in both space and time. As such, one can view microarchitectures as small scale distributed systems. Most students, however, try to approach real-world microarchitecture problems with a sequential, centralized mindset. We attribute this to a lack of training in concurrent thinking in today's computer science and engineering curricula, and specifically in current computer hardware classes. This results in an incomplete, fuzzy understanding of the problem and an inability to design adequate solutions. We believe the "less is more" mantra leads to misrepresentation of the challenges that face microarchitecture and often disengages students with overly simplified examples.

In this paper, we suggest presenting microarchitecture design through an animated visualization metaphor to enable students to quickly understand complex microarchitectural behavior. The basic idea is to visualize microarchitecture as a road network. Tapping into student experience with roads and interactive displays of traffic patterns allows them to quickly form suitable mental models for even complex microarchitectural behavior. We anticipate this will reduce frustration in learning while providing a metaphor that can serve for long term reference with properly identified similarities and dissimilarities. Additionally, we animated our metaphors to actively engage students during presentations and help them retain the concepts of the overall behavioral model.

In Figure 1, we show an example of a metaphor as applied to store forwarding, a common but complex microarchitecture optimization. In the block diagram (a), we see pipeline steps, registers, and flow paths. This represents the underlying hardware in a relatively direct manner. If data is waiting to be stored in one of the "Cache Store Port" stages, it may be sent through the bypass hardware to the "Load Store Queue (LSQ) Load Ports."

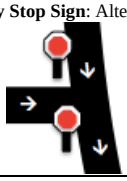
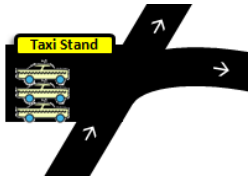
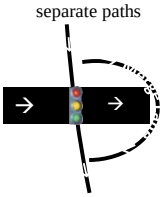
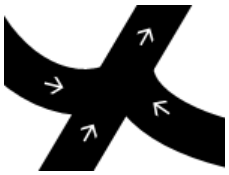
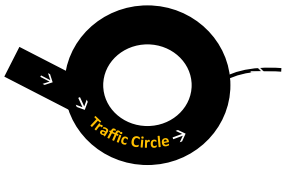


(a) Block Diagram **(b) Metaphorical Behavioral Model**

Figure 1. Store Forwarding Mechanism. The subcomponent hardware is represented by a block diagram in (a) and a traffic metaphor in (b). The metaphor provides an intuitive visualization of the component behavior.

SIGCSE'14, Date, Location, USA.

Table 1. Traffic Flow Metaphor Building Blocks

Parking Lot: Queues and buffers 	**Stoplight**: Event driven hold 	**All-Way Stop Sign**: Alternating flow 	**Highway**: Longer distance wires and networks 
Branch: Signal splitting 	**Merge**: Convergence of information from separate paths 	**Join**: All inputs may proceed 	**Traffic Circle**: Unknown access time 

This action is represented in the metaphor (b) as moving crates from the "Store Hold" yard to a car waiting in the LSQ traffic circle. The yard is adjacent to an airfield where new crates (stores) may arrive. These stores have not yet have been committed and are waiting to be cached in the warehouses. This metaphor explains the activity of store forwarding as well as the reason it is done. Visualizing store forwarding as a logistical activity clarifies the concept.

This paper is organized as follows: Section 2 describes our method for making metaphors out of microarchitectural building blocks. Section 3 applies this method to a processor's primary memory component. We provide initial evaluation of the metaphor method's efficacy and our plan for evaluation in Section 4. Section 5 describes related work in visualization, metaphors, and instructional games. We present final remarks in Section 6.

2. METAPHORS

In general, metaphors provide a way to represent new or unclear concepts with familiar objects and actions. Choosing an appropriate metaphor makes or breaks the success of this process. As such, the metaphor of city traffic as computer traffic presents as a time-tested comparison. Urban Planner Forrest Warthman presented detailed analogies covering comparisons between roads, buildings, doors, and even telephone wires to hardware components in computer architecture [14]. This metaphor has been featured in media, notably in the 1995 movie "Hackers," in which Artem Visual Effects created an urban landscape dataspace of information skyscrapers and computer circuitry [5].

Building on these metaphors, we suggest that instructions and data moving through processors may be represented as vehicles through city streets. Traffic control devices and signals direct the movement of the vehicles in a similar way to arbiters. The temporary storage aspect of a vehicle in a parking lot naturally translates to memory holding mechanisms such as queues and buffers. With many more analogies possible, we believe this particular metaphor has enough depth to support representing processor behaviors. We have laid out the basic parts for such representations in the following section, and include both reasons for the analogies as well as limitations in comparisons. Because the focus is data flow, it is important to remind students that not all elements within a component will have a direct analogy but rather provide a way to easily think through behavior.

2.1 Traffic Flow Building Blocks

Vehicle traffic within cities can take many different forms depending upon city planning, traffic laws, and safety calculations. However, many aspects of traffic flow carry enough universality to be easily recognizable. We propose the following general component analogies, shown in Table 1.

2.1.1 Parking Lots

Parking lots best represent queues and buffers, in which instructions must wait for data accesses to complete before they can execute. Within this general analogy, the specifics of how each parking lot behaves can be tailored to match the represented component. A FIFO may be represented by considering a meter-style method of ticketing. A queue that can be accessed at any point may be depicted by a cell-phone waiting lot (the vehicle will be notified when its components are ready).

2.1.2 Stoplights

Stoplights represent the ability to hold an instruction or data until an event occurs. The event could be the arrival of other instructions or data in the form of vehicles piling up at the stoplight, thus representing a pressure-sensitive stoplight. The event could also be dependent upon time factors, representing a timing programmed stoplight. The stoplight may also be able to control multiple vehicles proceeding forward if the hardware arbiter behaves this way.

2.1.3 All-Way Stop Signs

These represent any flow control where instructions or data will take turns to move forward along what will generally be the same road. All-way stop signs differ from stoplights in that the stoplight analogy is dependent upon a time or condition driven event and can allow varying behavior at the intersection while the signs always dictate the same alternating right-of-way behavior. The signs provide an ideal analogy for memory bank accesses, where data from different locations will need to return along the same path but cannot do so simultaneously. They may be used in conjunction with joins.

2.1.4 Highways

A highway may be any major road but most often implies a high-speed, multi-lane thoroughfare that spans long distances. In contrast to standard roads that represent on-chip networks, highways best represent inter-component wires.

2.1.5 Branches

A road branch causes two roads to proceed from one road. In the analogy, splitting a signal causes the original signal to proceed on both wires. This dissimilarity to the road traffic metaphor can be sidestepped by considering a duplicate copy of the same command to either not be a problem, or considering the branch itself to provide an opportunity for the original signal to hire an agent to do the same job on a different path.

2.1.6 Merges

Merges occur when multiple input roads converge to a single road. Unlike the join described in the next section, merges include selecting the data in transit. Two merging instructions or data become one element. In our metaphor, a duplicate copy provides any data received to the original instruction at a merge. The original instruction then performs the management duty of an abstracted multiplexer.

2.1.7 Joins

When multiple roads combine to form a single road, their convergence requires flow control as noted in Section 2.1.3. A join represents two or more roads that converge without data selection. In a join, each instruction (represented by a car) will continue independently.

2.1.8 Traffic Circles

Traffic circles allow a vehicle to continue circling for as long as necessary. This correlates with non-uniform access times for hardware (request-return time). The total time for this type of access cannot be known nor avoided. In real life, one would wish to exit the traffic circle after less than a full rotation. For the processor metaphor, vehicles that continue to circle represent accesses that take longer than a cycle to complete. These signals will circle at a high enough speed that subsequent vehicles do not feel safe to merge into traffic, thus allowing only one instruction to be in the circle at a time. This causes backpressure if it takes longer than one cycle for the instruction to complete access.

2.2 Developing Hardware Metaphors

While the building blocks provide a good starting point, a simple one-for-one translation of hardware via the described analogies does not effectively communicate a design. For example, in the case of a MIPS floating-point unit, all of the queues, registers, and buffers will become parking lots while all of the addressing, memory and functional units will become traffic circles [7]. This provides minimal benefit beyond the block diagram itself because this is a high enough level of abstraction that thinking through component behavior is a simple task that does not require an aid.

For more complex hardware, effective metaphors can be developed using a three-step process:

(1) Use the building blocks to form a basic roadmap.

(2) Develop a storyline to describe the represented behaviors. Complete and connect the roadmap to fit this story.

(3) Animate the actions on the roadmap.

The story helps provide a reason for movement as well as clues to the way the instructions and data will move through the hardware. In addition to a storyline, the metaphorical model needs to include animation so that students may clearly see how the traffic is moving and what goes on at each step. Static models provide an indication of how the movement would work, but are too abstract for students to grasp easily. The animations will follow the actions being taught and, in so doing, train the students in how to see the visualization. In this way, a metaphorical story model of the component behavior enables those unfamiliar with the component to draw simple conclusions as to behavior not explicitly defined.

3. APPLYING METAPHORS

As we mentioned in the introduction, microarchitecture experience is valuable in industry and research, and also provides an opportunity to introduce students to principles of distributed processing in complex systems. In this section, we provide an overview of a complicated and important microarchitectural block – the primary data memory subsystem – to illustrate some of the difficulties in teaching real-world microarchitecture in a class period and how this complex concept can be explained with metaphors.

We aim to make this material accessible to undergraduate students in a junior or senior level computer architecture class. Currently, most architecture classes we are aware of (at this level) teach high-level microarchitectural constructs at a simplified level, such as single pipelines with a limited number of interlocks and without any speculation support [7,10]. As the example in the next section illustrates, real-world cases have a number of interacting pipelines with complex flow control scenarios, bookkeeping structures and severe power and area constraints.

3.1 Case Study: A Primary Memory System

Our example processor is composed of "tiles" that provide specific processor functions connected through a scalable on-chip network. This model is representative of many current and proposed processors [4,9,12]. On such a substrate, the memory instructions can execute on any of the tiles, and can load from or store to cache memory available within a tile. Because of the distributed nature of the execution, loads and stores may execute out of program order; to guarantee correct results, the primary memory system must enforce read-after-write and write-after-write dependencies to the same memory address failing which a data hazard may occur. The Load-Store-Queue (LSQ) provides these functions by buffering all executed loads and stores in a bookkeeping structure. When read-after-write hazards are detected by the LSQ (by examining the addresses of the buffered loads and stores in the LSQ), any dependent read (load) that is returned prior to its matching store is re-executed. This invalidates any work completed prior to hazard detection. Regular read-after-write dependencies are enforced by supplying the value of the load from the in-flight store buffered in the LSQ. This process is called store forwarding. In certain circumstances, the load value may have to be obtained by the LSQ and the Level-1 Data cache (L1D). To prevent write-after-write hazards, the LSQ buffers all stores and commits them to cache memory in program order.

A speculative structure called the Dependence Predictor (DPR) is used to predict if pending, unexecuted stores precede a matching load. When the dependence predictor predicts a hit, the load request will wait at the LSQ until all prior stores have arrived. Once all stores have arrived, it will then be reinjected into processing pipelines to begin the process of accessing the L1D and LSQ again, as the newly stored data is now accessible through those components. If a load request misses in the L1D, LSQ, and the DPR, the load request will go into a missed load pipeline and wait for the data to be sent from the Level 2 Cache off-tile. Once it has replaced a line in the L1D, the load request will be notified to replay as if it were a new load. This mechanism simplifies control logic significantly.

Any stores that hit in the L1D will update and stay, while stores that miss will be forwarded to the Level 2 Cache. Because of the tiled microarchitecture, memory accesses that go off-tile will have to hop through the on-processor and on-chip networks (OPN and OCN). Each movement of a load or store request will take at least one clock cycle to complete, so shorter distances will equal faster replies. All ingress and egress points can have backpressure and all these processing functions are carried out by shared hardware bookkeeping structures; correct sharing requires structural hazards to be handled correctly.

Table 2. Metaphorical Equivalent of Primary Memory System Operation

CONDITIONS	ACTIONS	EXPLANATION
L1D Hit LSQ Miss DPR Miss	Step 1. Cache & LSQ access Step 2. Load reply	After steps explain this is anticipated as the common case and has a short, two cycle time for completion.
L1D Hit LSQ Hit DPR Miss	Step 1. Cache & LSQ access Step 2. L1D & LSQ Hit with Store Forwarding Step 3. Merge Step 4. Load Reply	Prior to steps, ask how the employee at the airport could get the newer components if they are sitting in the Store Hold yard. After answers, explain Store Forwarding and how multiple forklift trips may be needed if components were sent in different crates.
L1D Miss LSQ Miss DPR Miss	Step 1. Cache & LSQ access Step 2. Load Miss & Cache Fill Request Step 3. Data Return & Alert Miss Registers Step 4. Cache & LSQ access Step 5. Load Reply	Explain that the Request and Fill Processing stations streamline processing (merging in these slots is possible; the merge process will be discussed in more detail in the next action). Discuss how the RFID signaling system keeps loads and stores in the correct order.
Store	Step 1. Drop at LSQ; Notify other Facilities Step 2. Continue until Yard is Full Step 3. Commit oldest; Hits at Local Facility Step 4. Commit next oldest; Miss = Send to Warehouse	Discuss memory hierarchy store policies. Our example processor follows a Write-Back policy for store hits and a Write-No Allocate policy for store misses. Explain that crate/pallet tags are sent from the yard's scanner to a network (Data Status Network or DSN) that includes package facilities in the same region (processor). Components (Bytes) that go into the same crate (cache line) do so at the merge processing yard. Since the yard cannot hold multiple sets of crates to merge, a crate would be sent forward when a different crate needs to enter the merge yard.
Dependent Load followed by Store	Step 1. Cache & LSQ access Step 2. Load Reply Step 3. Store Arrival Step 4. Dependence Discovered Step 5. Recovery Initiated	This scenario should be avoided if at all possible; anything that was done with the elements inside the crate will have to be redone after the car returns to pick up the correct elements.
DPR Hit	Step 1. Cache & LSQ access Step 2. Load Wait in LSQ for all prior Stores Step 3. All Stores complete Step 4. Cache & LSQ access Step 5. Load Reply	The load is informed that the needed components are on an incoming flight. To save the company from excess costs, the load requestors wait in the cell phone lot. After all prior stores have arrived, some of which may be allocated to the L1D, requestors are notified to return. Since they do not know where the most updated components are until they check face to face, they repeat their L1D and LSQ access.
Load/Store in Secondary Memory System	Step 1. Access Network for Address Translation Step 2. Hop to Destination Memory Tile Step 3. Pick up or Drop off Package Step 4. Return Hop	Before steps, verify that students know what 'hop' means in terms of traffic flow and time to complete. Here, the data is shown as returning via the same On-Chip Network (OCN) line to correlate with the primary memory system requests previously shown. Briefly explain that it could have returned on another OCN road if the dispatcher (control system) knew that a different package facility needed the retrieved components. Ask the students why this would be better.

Explaining this particular microarchitecture within a portion of one class period proves too much for even graduate students to fully grasp through text and diagrams alone. By having students participate in a behavioral visualization animation, students will have a chance to focus on the important takeaways from the study instead of getting bogged down in design complexity. Students should be able to walk away from the exercise with a model of how real-world microarchitecture behavior flows.

3.2 Metaphor for Primary Memory System

Following the steps in Section 2.2, we first constructed a roadmap that included representations for the main elements with enough sub-elements to support a basic set of memory system actions. We next formulated a story where an external organization needs to retrieve items from a logistical management company. Finally, we animated the actions to fit the steps listed in Table 2. We believe this story and the actions we have included adequately enable students to form a behavioral model.

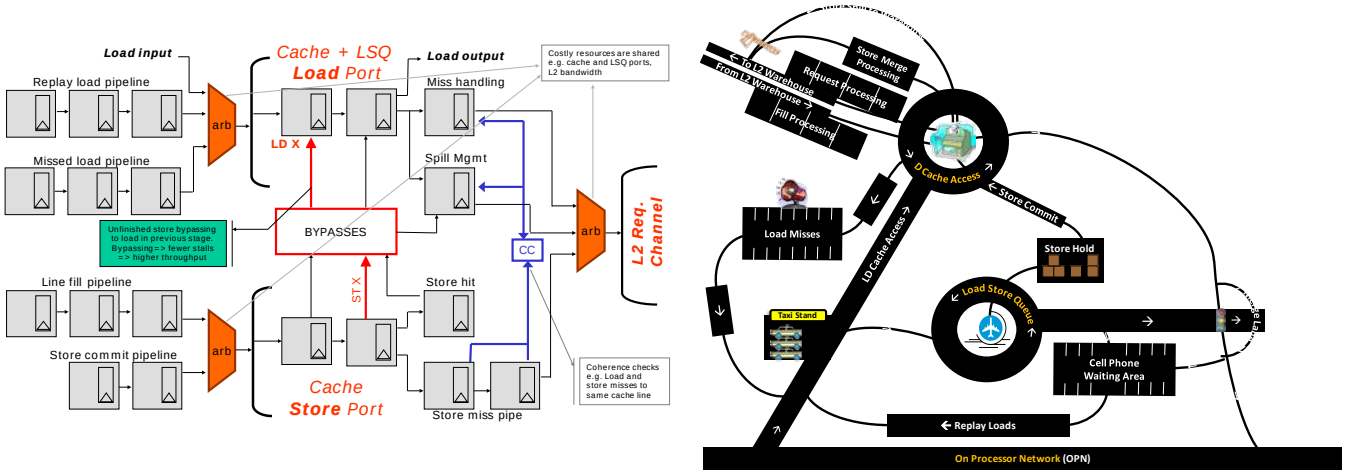
The metaphorical story begins with a mission to retrieve a package of multiple components as fast as possible. The scenario is that the organization cannot wait any longer than necessary for the package to be delivered and has one company car and two people to put towards rapid retrieval. As shown in Figure 2, the package might be at the package processing center (L1D), stored in the package center's warehouses (L2 cache via the OCN), or it could still be at the airfield (LSQ).

4. Evaluation

Our experiences in presenting the data memory metaphor without animation showed limited value. We used a previous version of Figure 2b in class and saw some student interest. The version we presented lacked the unifying storyline for actions and animation. Students were given a short amount of time to consider the correlation between the hardware and the metaphor, but did not further participate. Because of these limitations, the takeaway value for students was unclear and therefore short-lived.

We presented the material described in this paper to two classes. The senior and graduate level computer architecture class covered typical graduate level content and used the Hennessy and Patterson Computer Architecture book as its text. We presented our material during the memory systems lesson block, so it had maximum correlation to course content. The junior level computer systems fundamentals class was mandatory for all computer science and electrical engineering undergraduates and covered basic digital logic design and computer organization. The junior level course included four lessons on pipelining and caching; our goal was to show how these two concepts fit together in a real world context.

For both classes, we explained memory management behavior in terms of the logistics scenario, and related the behaviors to the traditional block diagram hardware design at the end of the presentation. 83% of students said the metaphor helped improve



(a) Block Diagram **(b) Metaphorical Behavioral Model**

Figure 2. Primary Memory System Representations. The abstracted block diagram (a) shows the pipeline stages for a primary memory tile within a heavily partitioned parallel processor. The metaphorical visualization of the system (b) enables students to think naturally through each cycle for a given set of actions.

their understanding of memory system microarchitecture. Figure 3 shows a breakdown of the responses to this question from both classes. For the senior and graduate level class, we also surveyed the students on their understanding of memory systems prior to the lecture. Using the answer scale as weights to assign a number to their responses, the computer architecture class self reported at 275.3 understanding prior to and 385.8 understanding after the lecture (on a scale where 500 shows self-reported perfect understanding).

In summary, 74% of the juniors and 93% of the seniors and graduate level students to whom we presented the material found it fairly to extremely helpful. The ability of the material to tie together multiple concepts through an intuitive look at specific real world hardware may help students frame not only memory system behavior but also how microarchitecture fits into their disciplines. While more evaluation needs to be done, we believe these results indicate our proposed animated scenario-driven metaphor corrected for the initial issues and has high potential to clearly communicate a lasting model.

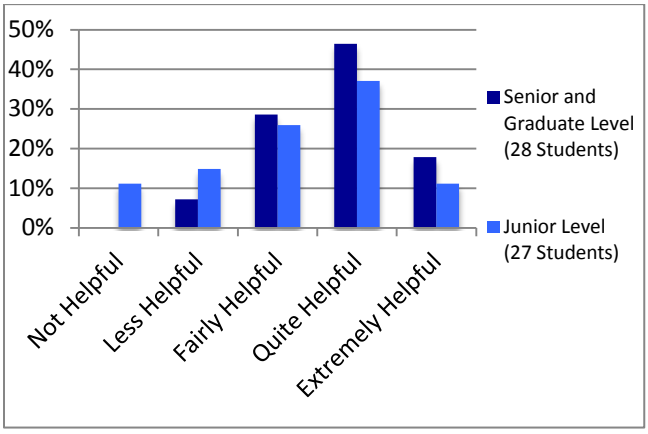


Figure 3. Class Survey Responses. The graph shows the answers two classes gave for the question of how helpful the metaphorical visualization was to understanding memory system behavior.

5. RELATED WORK

While much work has been done on visualization of scientific data and software, we are not aware of any similar work in metaphorical visualization for computer architecture instruction. The three fields of visualization research, computer metaphors as models, and instructional games contribute to the ideas we have presented. We will briefly outline related work in these areas.

Visualization research: Research in this field shows that graphical representations clearly and efficiently convey complex ideas [13]. Users may benefit from visualization through exploratory discovery, analytical decision-making, or descriptive explanations [2]. Because of this potential, we believe that visualization beyond standard block diagrams will prove to be useful in helping students reach the decision making stage about the value of the underlying concepts. By capitalizing on pre-existing mental models of road networks and traffic flow, the instructional visualization proposed in this paper can move students efficiently from unknown scanning to an analytical appreciation of the microarchitecture represented.

Computer Metaphors as Models: Students begin applying metaphors to computer architecture early in their education. Many basic computer systems classes use the classic laundry example of pipelining the tasks of washing, drying, folding, and storing [10]. While this provides a quick visualization, it supports only a weak mental model of the process. A good metaphor, in contrast, provides a strong mental model. In this context, we can define a model as explicit, comprehensive, valid and differentiated from the open-ended, incomplete and inconsistently valid qualities of a metaphor [3]. While a metaphor itself is not a model, it can provide an accessible starting point for discovering and building an underlying model [1]. A well-defined metaphor that highlights similarities as well as dissimilarities can succeed in enabling a student to form a mental model of the subject material [16]. One of the closest examples of this paper's proposed method lies in a 1996 paper proposing the InHouse system, visualizing a hotel with rooms containing several suitcases each containing pieces correlated to a system with multiple processors running processes each composed of statements [6]. The Austrian team designed this metaphor for a parallel system monitoring and visualization user interface, and not for teaching concepts. By introducing a

robust microarchitecture metaphor suitable for enabling students to develop behavioral models, we believe our contribution uniquely advances computer architecture education.

Instructional Games: While most games in computer education focus on creation and coding, disciplines that use games purely to educate have seen great success. A few examples for computer education do exist, and one recently implemented game at New York University focused on system administration. Their game introduced a "firefighting" scenario where students as system administrators attempt to keep advertisement serving traffic, and thus revenue, high [15]. Students found the game enjoyable and learned how to work through a distributed system to learn about scalability and distributed algorithms. However, this approaches the concept of visualization from a software-based-only perspective, visible in the game allowing students to simply purchase new computers if they required more processing power. We believe that combining the enjoyment factor of a quiz game with a solid hardware metaphor will maximize participation and long-term benefits for students.

6. CONCLUSION AND FUTURE WORK

At the end of the nineteenth century, German chemist August Kekulé spoke and wrote of how he discovered the ring structure of the benzene molecule [11]. His discovery was historically attributed to imagining a snake eating its own tail while he was dreaming, but further study into the circumstances of relating the account and improved translation show conscious mental imagery. He first began to train his *mental eye* in visualizing atom groupings while developing the structure theory, and was later able to use this mental model to "see" benzene's structure.

In this paper, we have illustrated such a visualization in allowing students to "see" microarchitectural behavior. Students who are able to think through the examples should gain a deeper understanding than they would otherwise glean from only a high abstraction level block diagram. We believe the traffic metaphor will provide students with a working mental model for heightened and expedited understanding throughout their studies.

While we have described the process of creating these in enough detail to be reproduced, we do understand that teachers have limited time to develop new lesson material. It took us approximately seven hours to complete the primary memory system roadmap and story, and over thirty hours to identify and animate key steps for seven actions. To facilitate other teachers in using this material, we offer to release the templates. In doing so, we appreciate and welcome evaluation input to ensure this tool may benefit as many students as possible.

7. REFERENCES

[1] Max Black. More about metaphor. *Dialectica*, 31(3-4):431–457, 1977.

[2] D.M. Butler et al. Visualization reference models. In G.M. Nielson and D. Bergeron, editors, *Proceedings, IEEE Conference on Visualization*, pages 337–342, October 1993.

[3] John M. Carroll and Robert L. Mack. Metaphor, computing systems, and active learning. *International Journal of Man-Machine Studies*, 22(1):39 – 57, 1985.

[4] Peter Clarke. Tilera launches many-core 64-bit processor. Webpage, June 24 2011. Available at http://www.eetimes.com/electronics-news/4217220/Tilera-launches-many-core-processor.

[5] Martin Dodge. Artistic representations of cyberspace. Webpage, February 2007. Available at http://personalpages.manchester.ac.uk/staff/m.dodge/cybergeography/atlas/artistic.html.

[6] G. Haring, G. Kotsis, and S. Musil. Inhouse-a user-oriented monitoring approach. In *Parallel and Distributed Processing, 1996. PDP '96. Proceedings of the Fourth Euromicro Workshop on*, pages 478 –485, Jan 1996.

[7] John L. Hennessy and David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[8] Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *Computer*, 41:33–38, 2008.

[9] Min Li et al. Functional partitioning to optimize end-to-end performance on many-core architectures. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–12, Washington, DC, USA, 2010.

[10] David A. Patterson and John L. Hennessy. *Computer Organization and Design, Fourth Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[11] Albert Rothenberg. Creative cognitive processes in kekulé's discovery of the structure of the benzene molecule. *The American Journal of Psychology*, 108(3):pp. 419–438, Autumn 1995.

[12] Karthikeyan Sankaralingam et al. Distributed microarchitectural protocols in the trips prototype processor. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 480–491, Washington, DC, USA, 2006.

[13] Edward R. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press, 1997.

[14] Forrest Warthman. Cities and computers: Their architecture. YouTube, Feb 9 2011. Available at http://www.youtube.com/watch?v=NHI63efXoko.

[15] Joel Wein et al. Virtualized games for teaching about distributed systems. In *Proceedings of the 40th ACM technical symposium on Computer science education*, SIGCSE '09, pages 246–250, New York, NY, USA, 2009.

[16] Lucy Anne Wozny. The application of metaphor, analogy, and conceptual models in computer systems. *Interacting with Computers*, 1(3):273 – 283, 1989