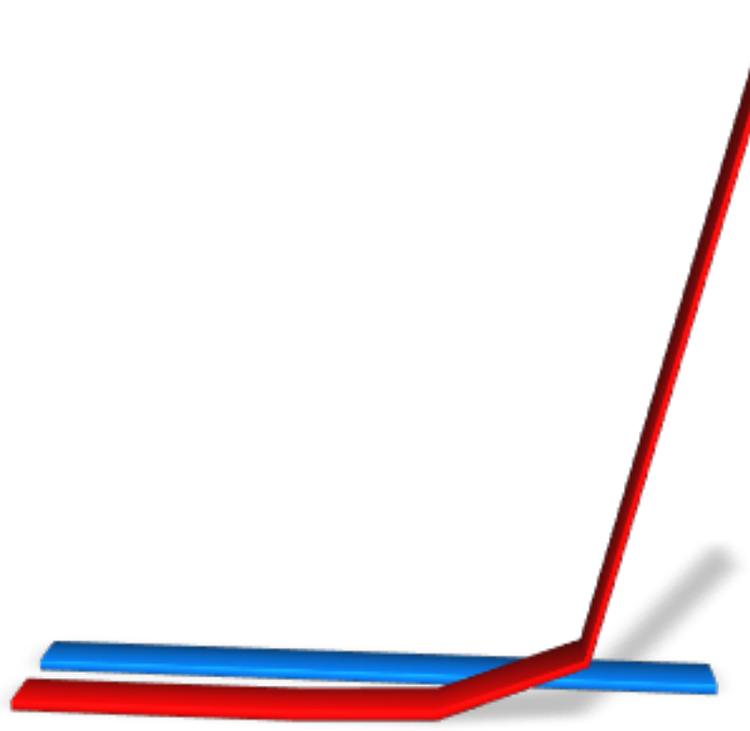
Stories, Not Words: Abstract Datatype Instruction Sets

Martha Kim
Columbia University

Workshop on New Directions in Computer Architecture
6/5/2011

The Utilization Wall

- Exponential decrease in percentage of transistors that can be operated at full frequency.



Moore's Law (manufacturable transistors)

Power budget (operable transistors)

- In 45nm TSMC process, 7% of 300mm ^{2}die can operate at full frequency
- In 32nm, 3.5%

Goulding et al. Conservation cores: Reducing the energy of mature computations. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 205–218, Pittsburgh, Pennsylvania, March 2010.

Specialization Is a Promising Approach

- R. Hameed et al., “Understanding sources of inefficiency in general-purpose chips,” ISCA '10
- G. Venkatesh et al., “Conservation cores: reducing the energy of mature computations,” ASPLOS '10
- J. Kelm, D. Johnson, W. Tuohy, S. Lumetta, and S. Patel, “Cohesion: a hybrid memory model for accelerators,” ISCA '10
- H. Franke et al., “Introduction to the wire-speed processor and architecture,” IBM Journal of Research and Development, vol. 54, no. 1, pp. 3:1–3:11, 2010.
- V. Govindaraju, C. Ho, and K. Sankaralingam, “Dynamically Specialized Datapaths for energy efficient computing,” HPCA ’11
- M. Lyons, M. Hempstead, G. Wei, and D. Brooks, “The Accelerator Store framework for high-performance, low-power accelerator-based systems,” Computer Architecture Letters, vol. 9, no. 2, pp. 53–56, 2010.
- C. Cascaval, S. Chatterjee, H. Franke, K. Gildea, and P. Pattnaik, “A taxonomy of accelerator architectures and their programming models,” IBM Journal of Research and Development, vol. 54, no. 5, p. 5, 2010.
- R. Hou et al., “Efficient data streaming with on-chip accelerators: Opportunities and challenges,” HPCA ’11
- N. Goulding et al., “GreenDroid: A Mobile Application Processor for Silicon’s Dark Future,” Hotchips ‘10.

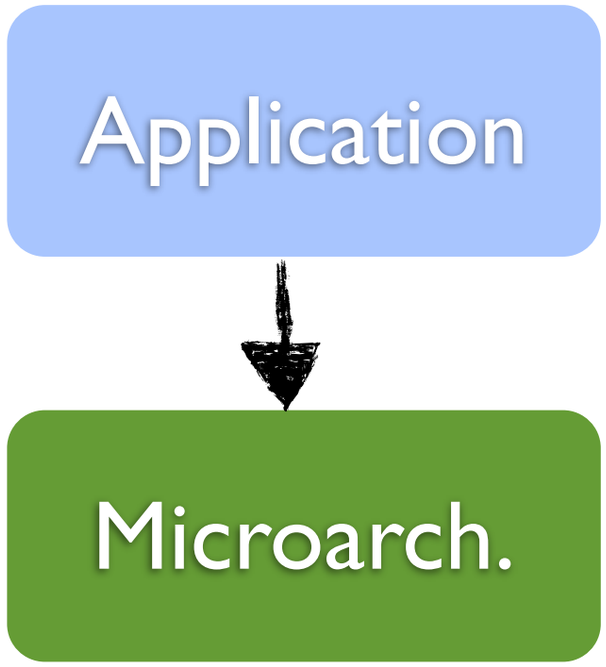
An Ideal Accelerator System

- High Performance
- Low Energy
- Easy to Program
- Software Portability

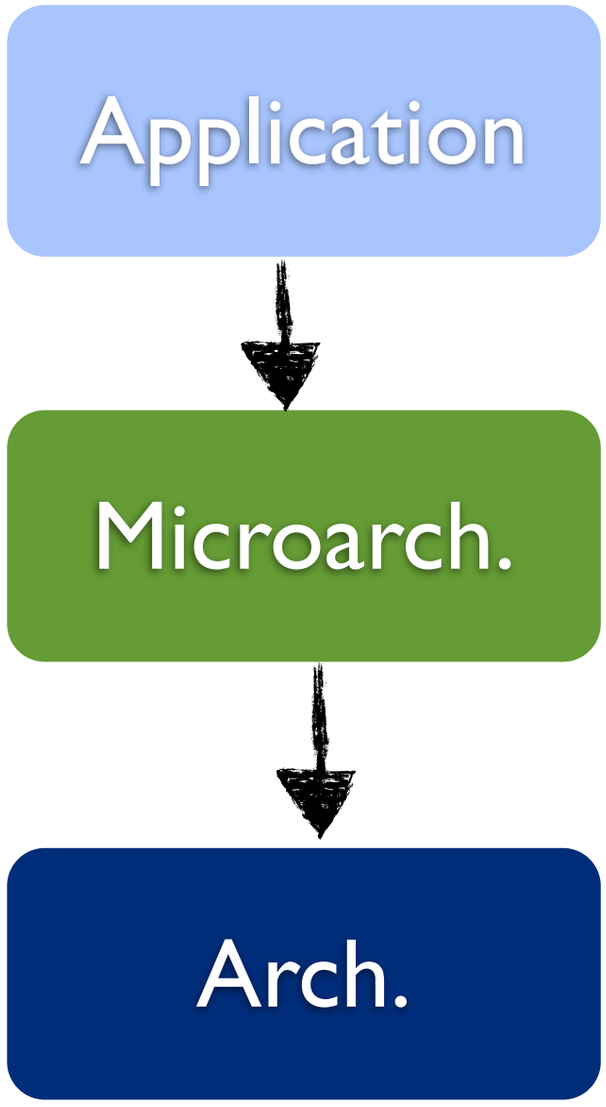
Accelerator Design Processes

Application

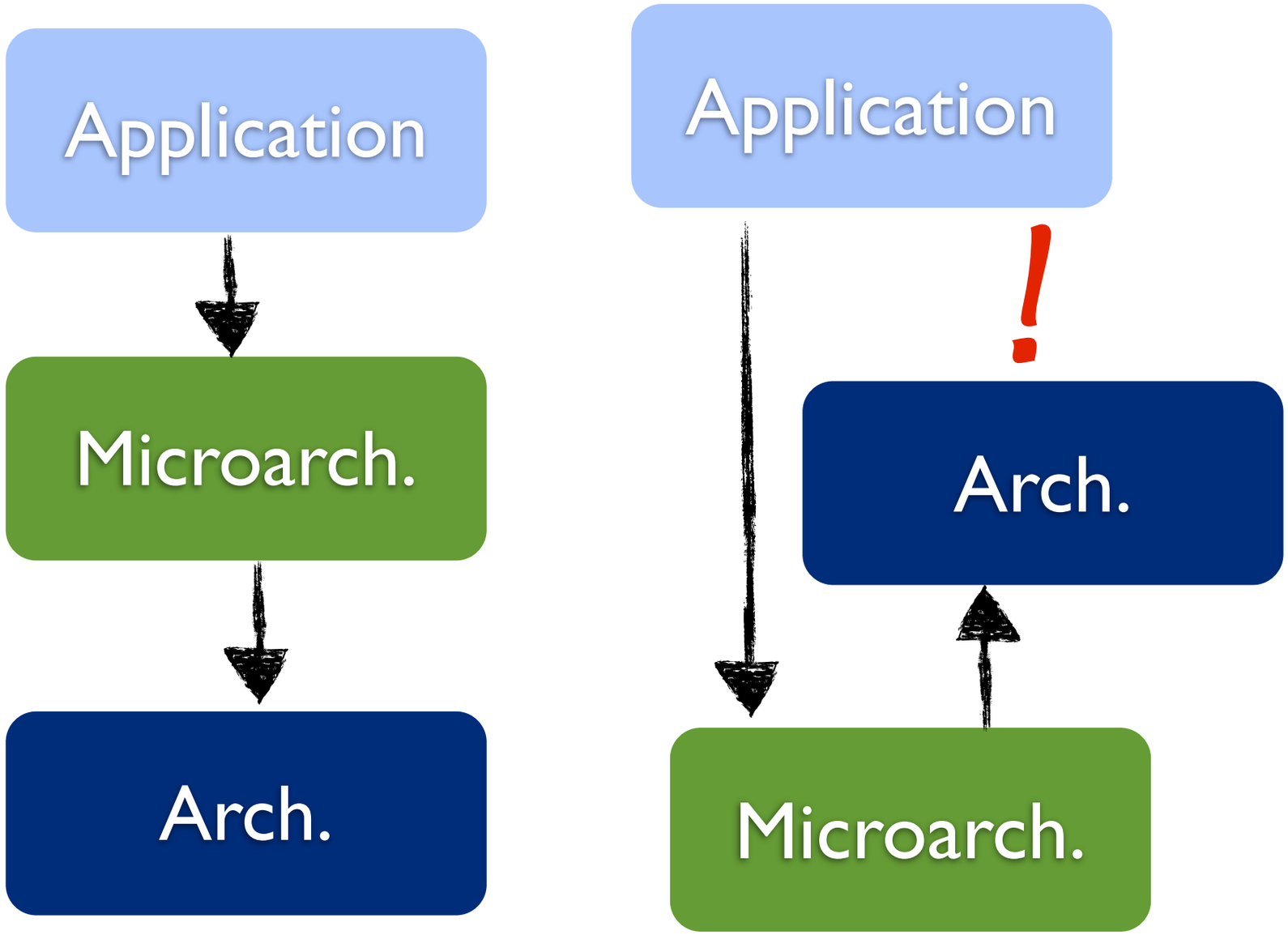
Accelerator Design Processes



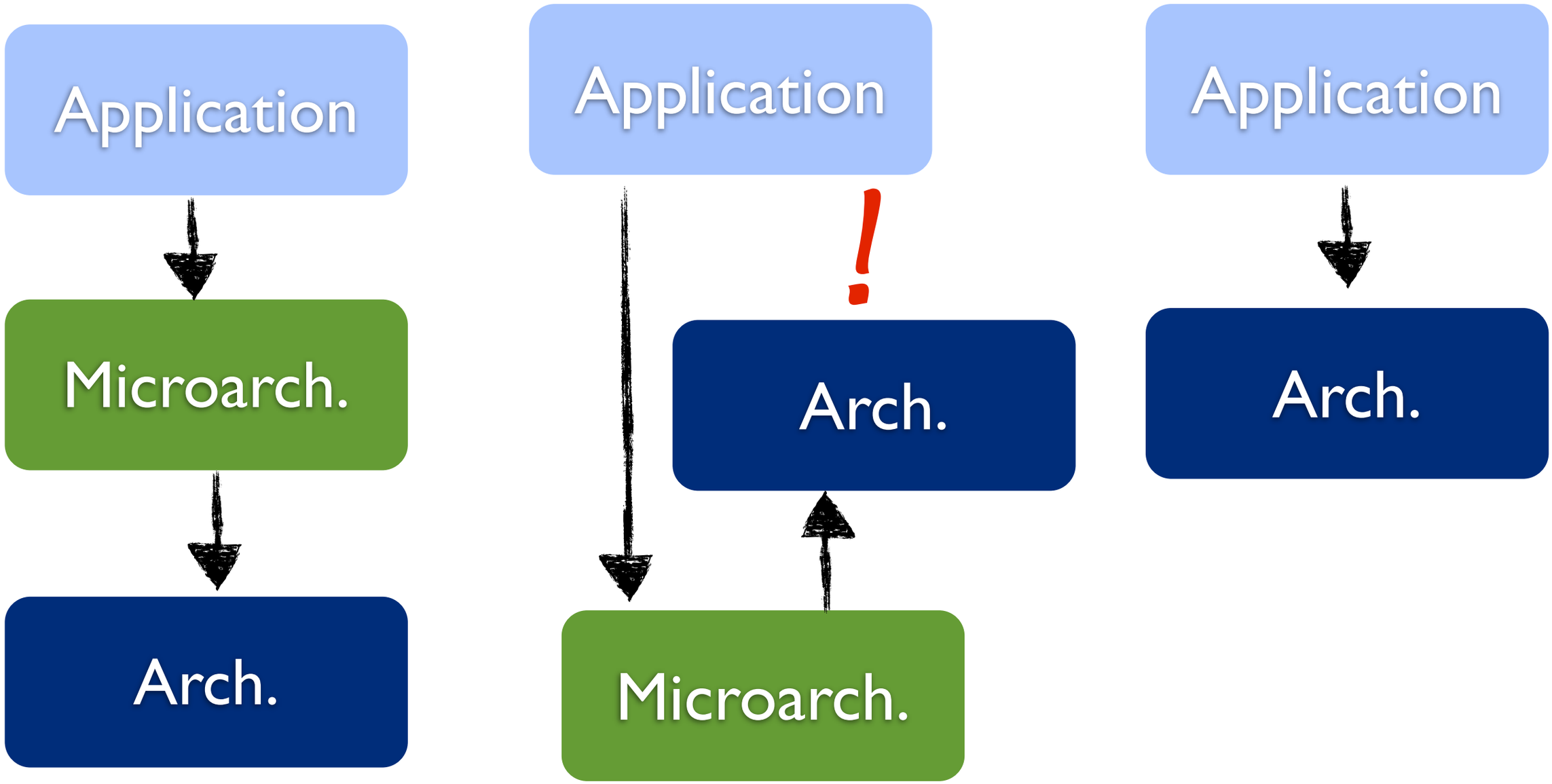
Accelerator Design Processes



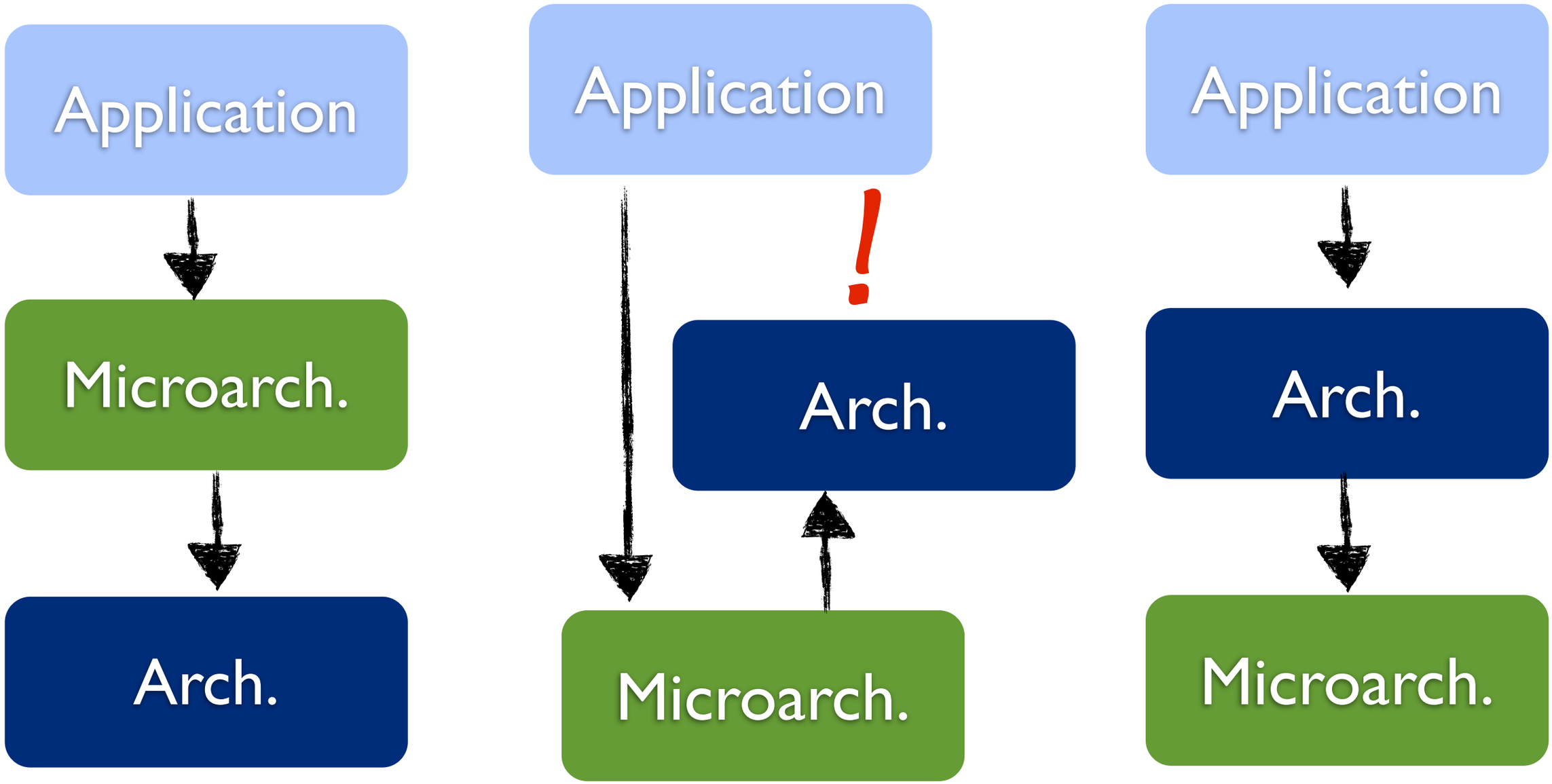
Accelerator Design Processes



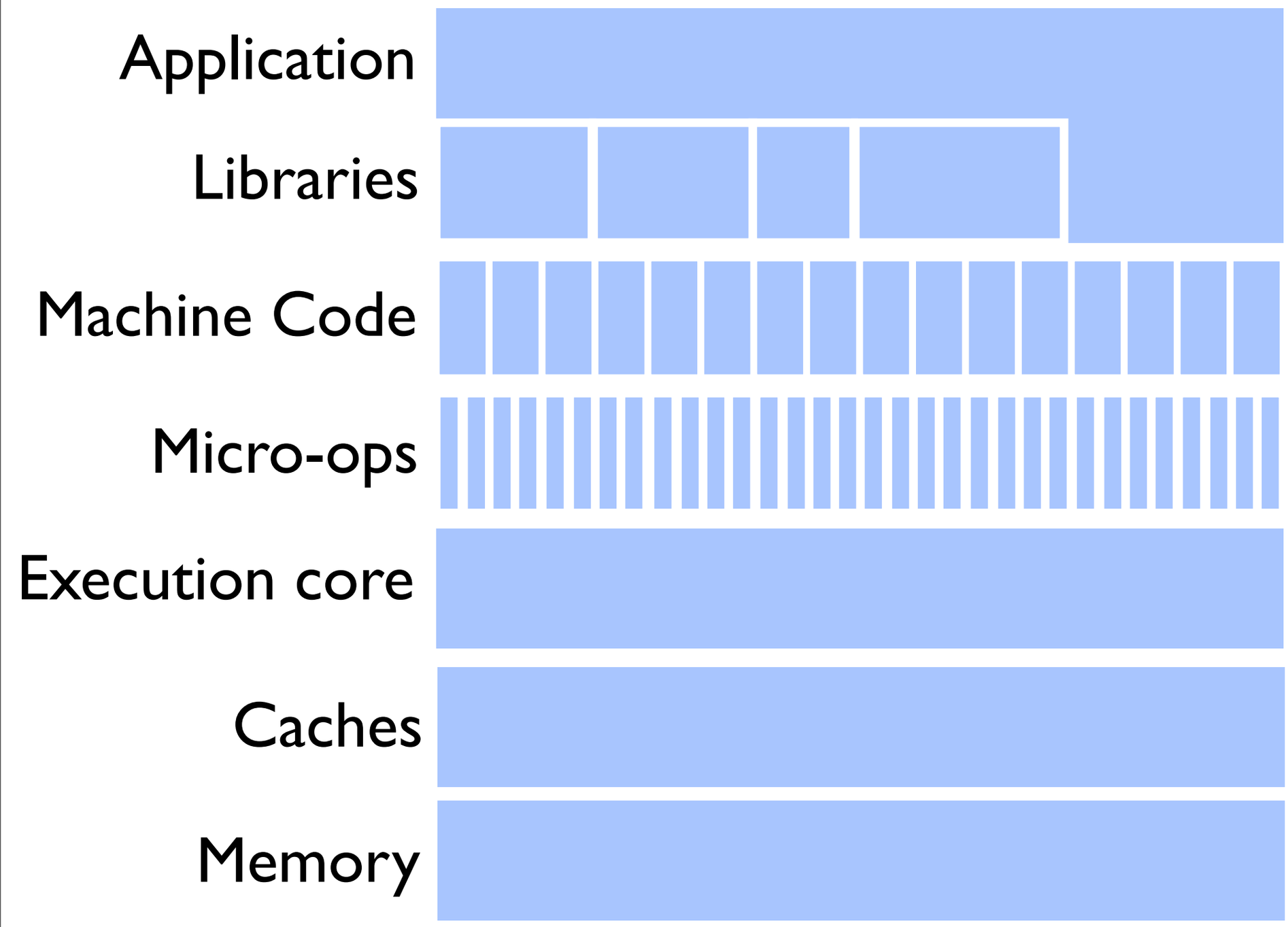
Accelerator Design Processes



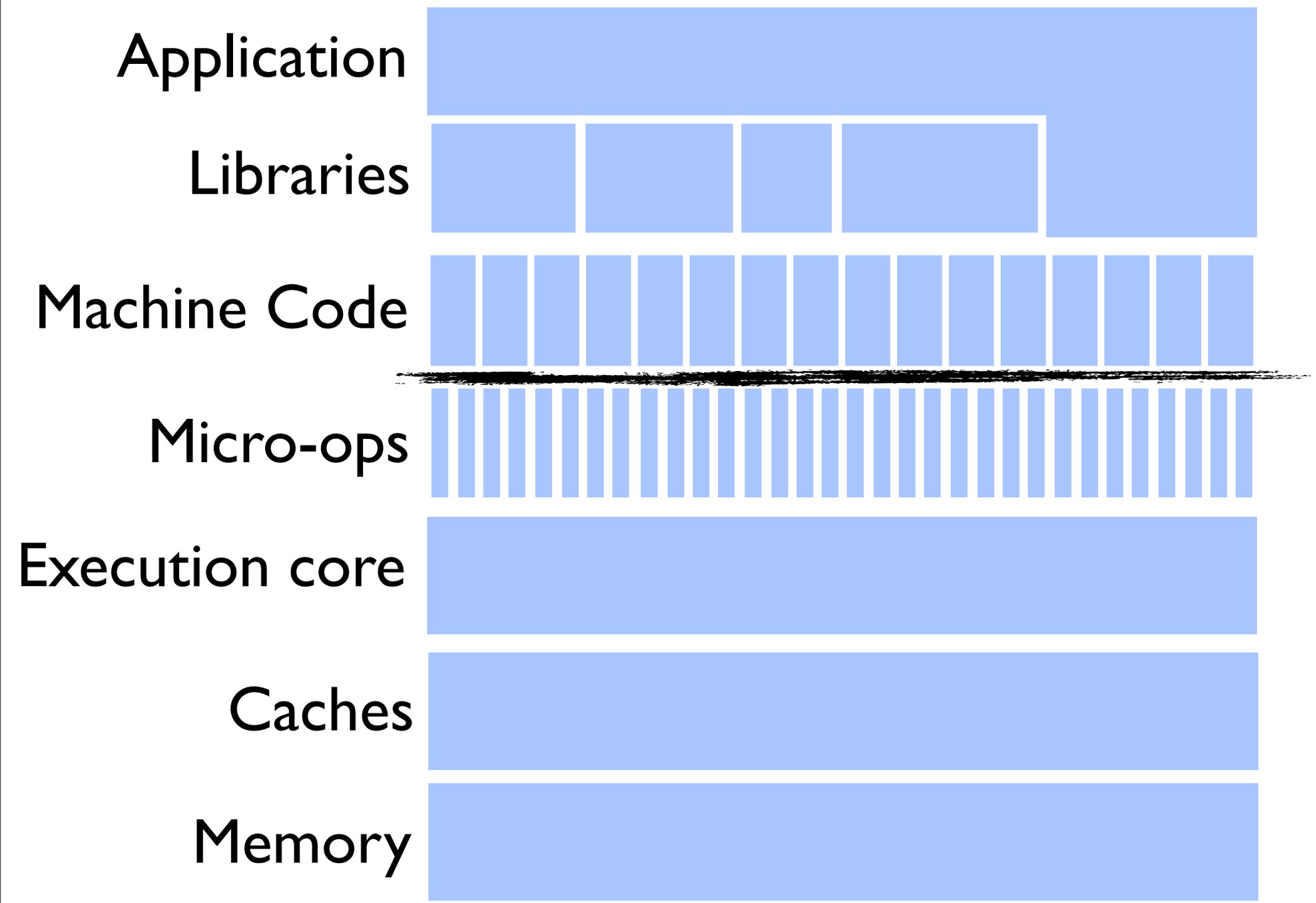
Accelerator Design Processes



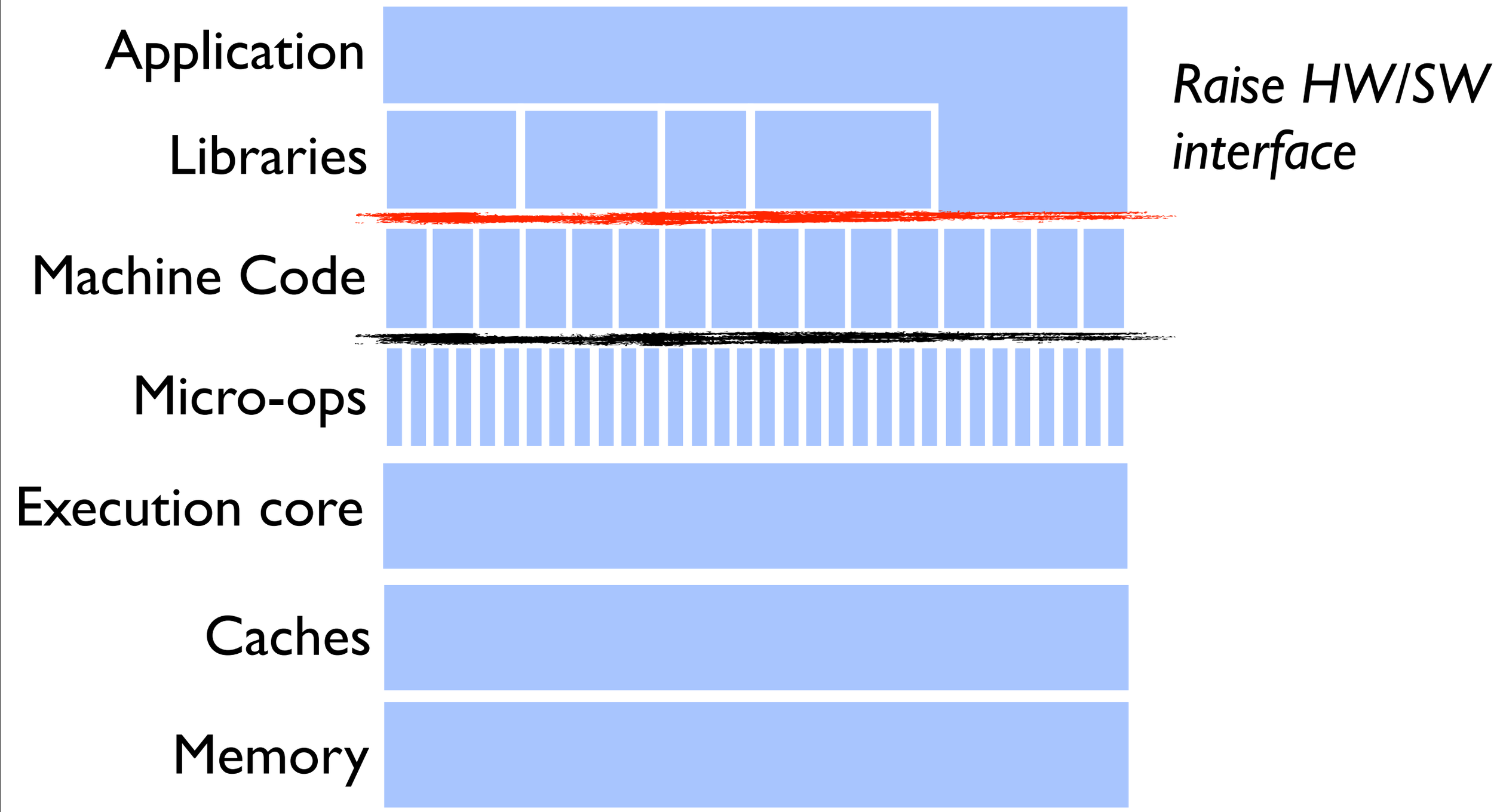
Extending Software Abstractions to Hardware



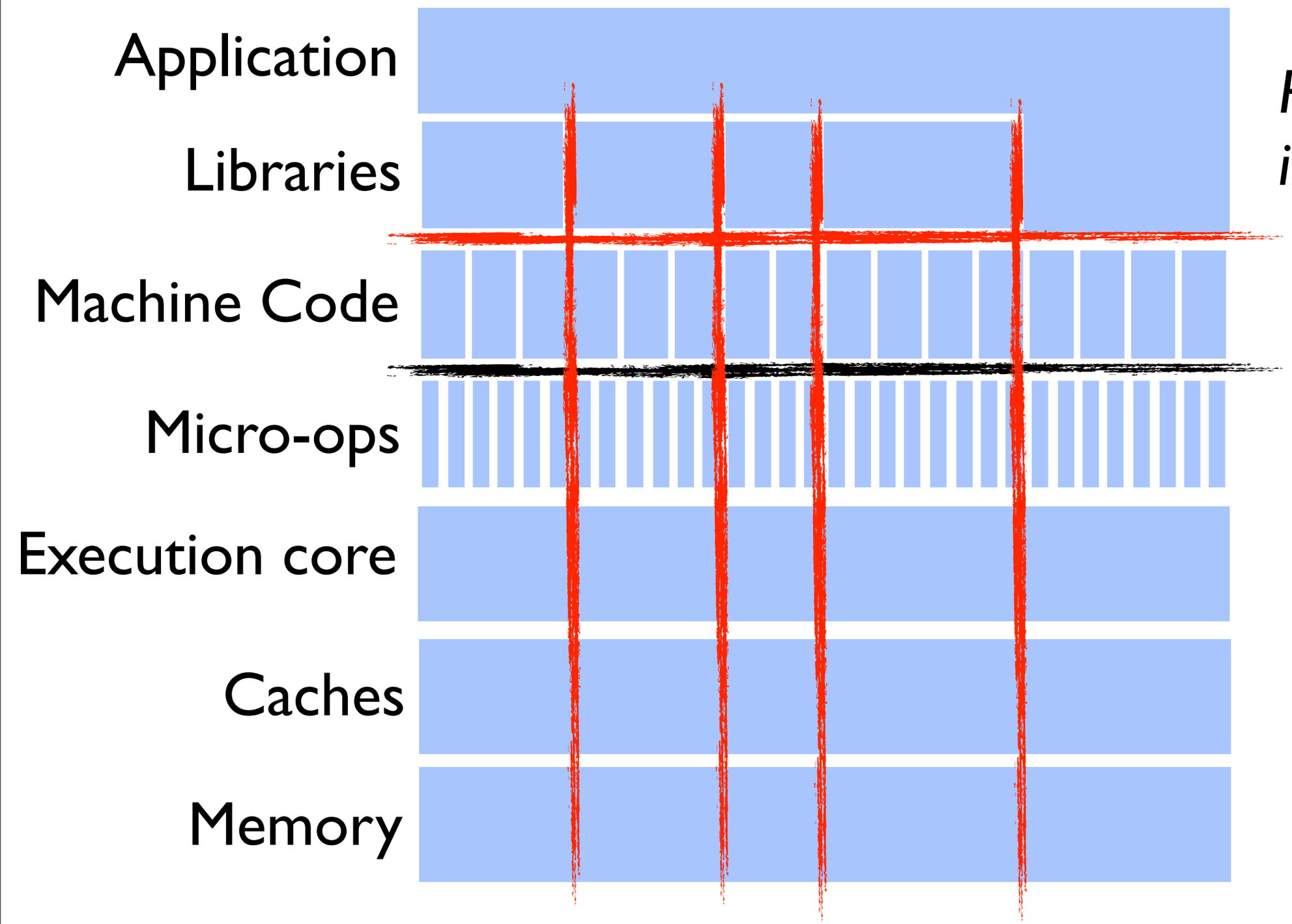
Extending Software Abstractions to Hardware



Extending Software Abstractions to Hardware



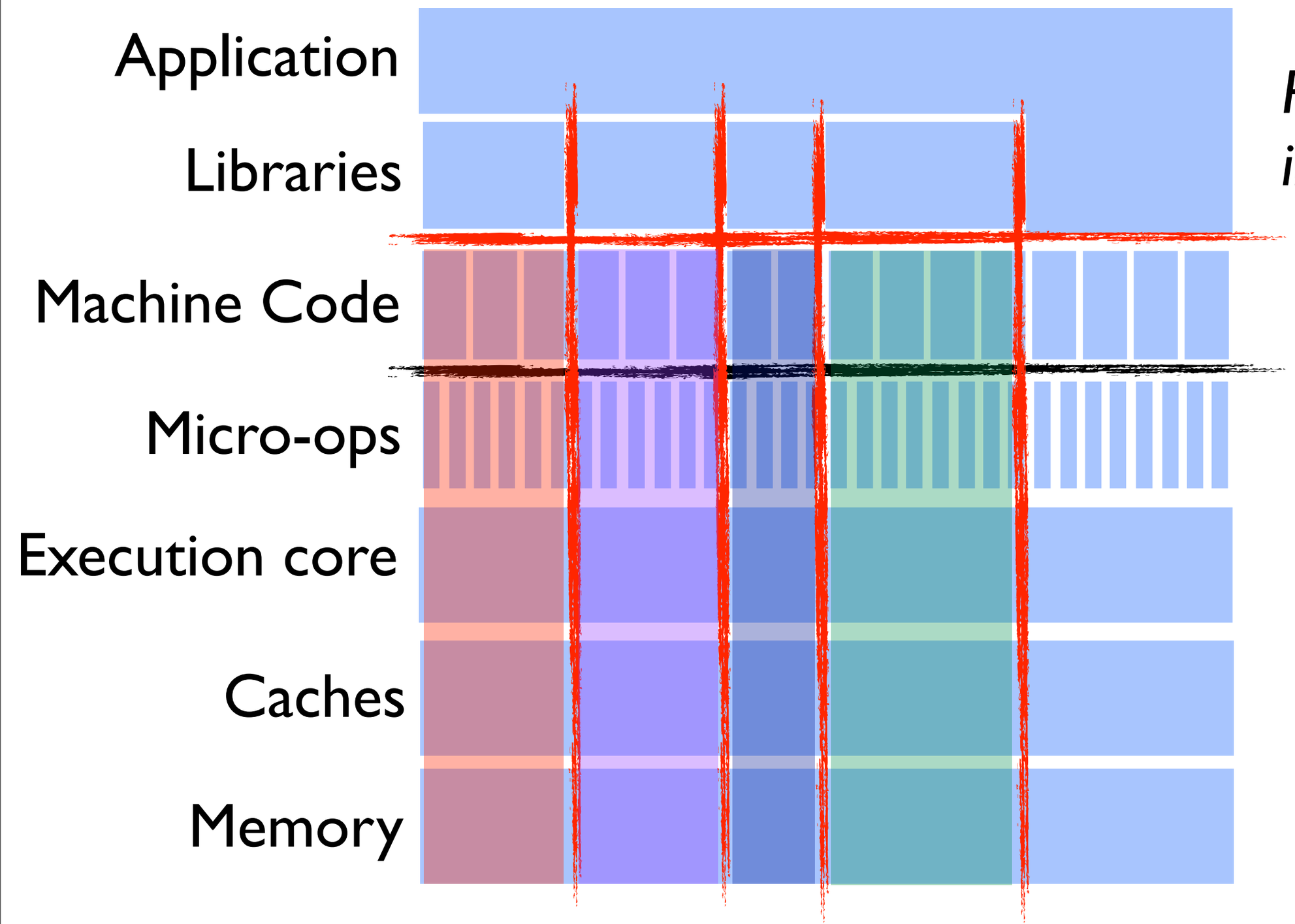
Extending Software Abstractions to Hardware



Raise HW/SW interface

Extend *interfaces from libraries to hardware*

Extending Software Abstractions to Hardware



Raise HW/SW interface

Extend *interfaces from libraries to hardware*

Exploit *interfaces with specialized hardware*

Abstract Datatype Processing

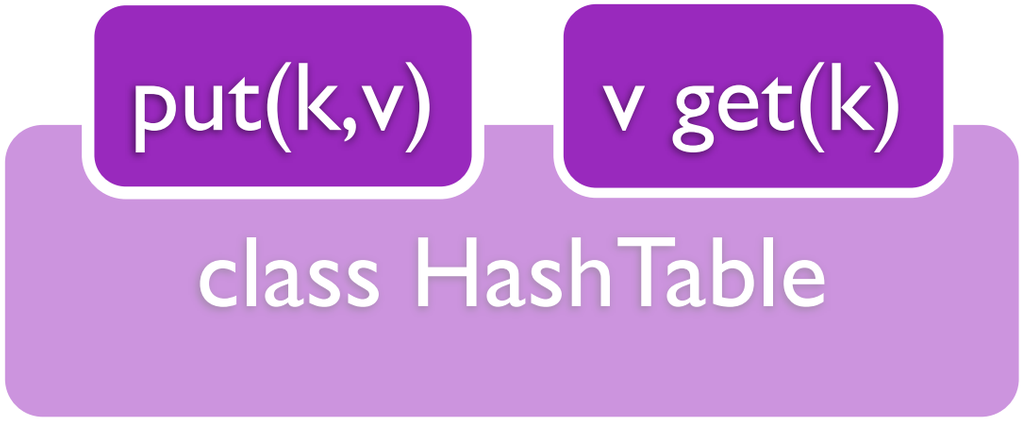
SW

Arch

UArch

Abstract Datatype Processing

SW



Arch

UArch

Abstract Datatype Processing

SW

put(k,v)

v get(k)

class HashTable

Arch

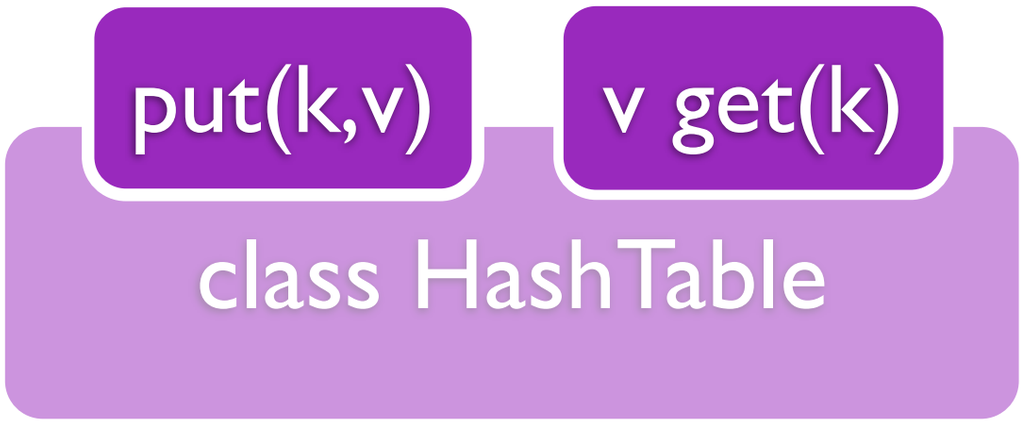
put $h, $k, $v

get $h, $k, $v

UArch

Abstract Datatype Processing

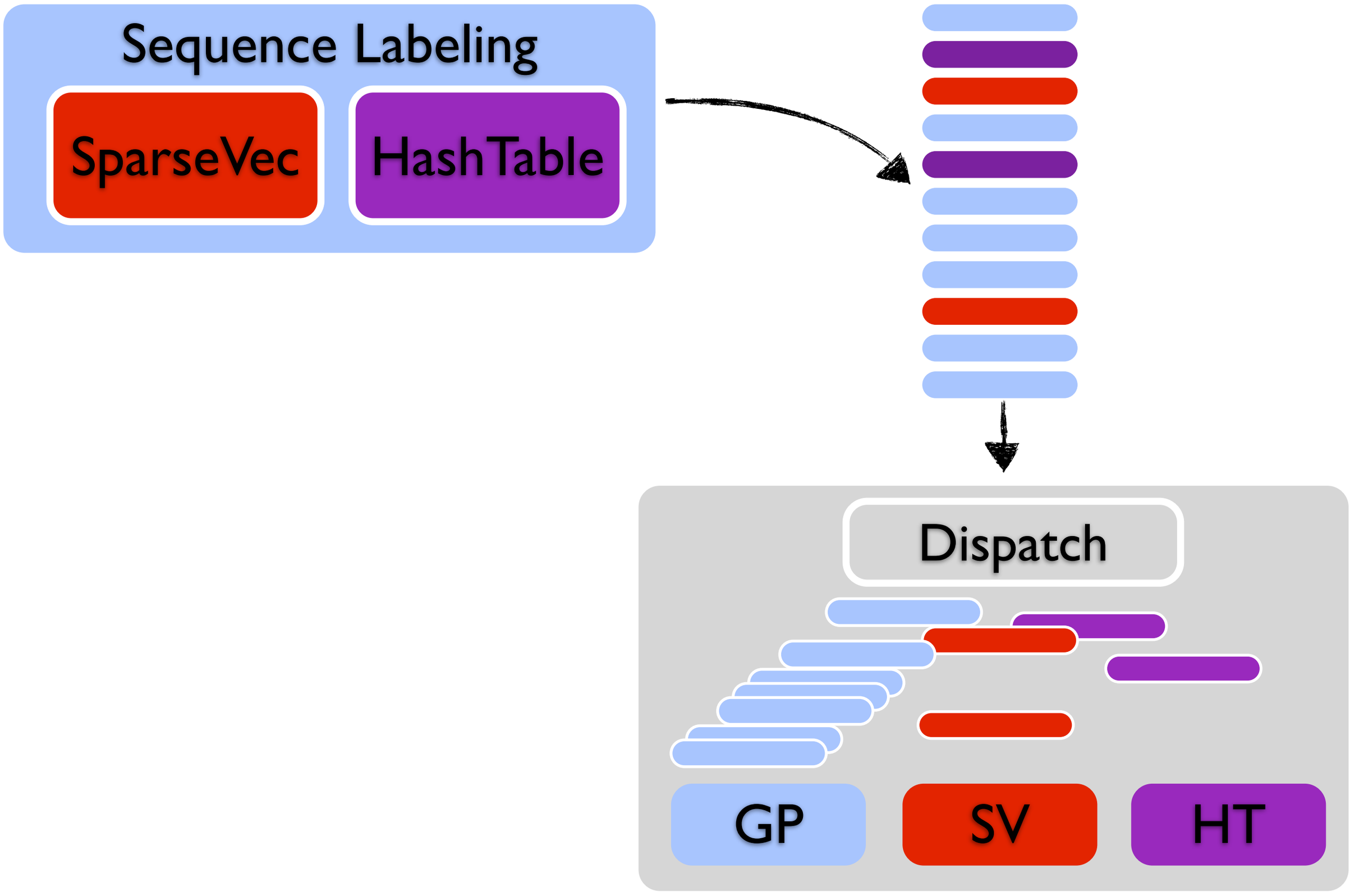
SW



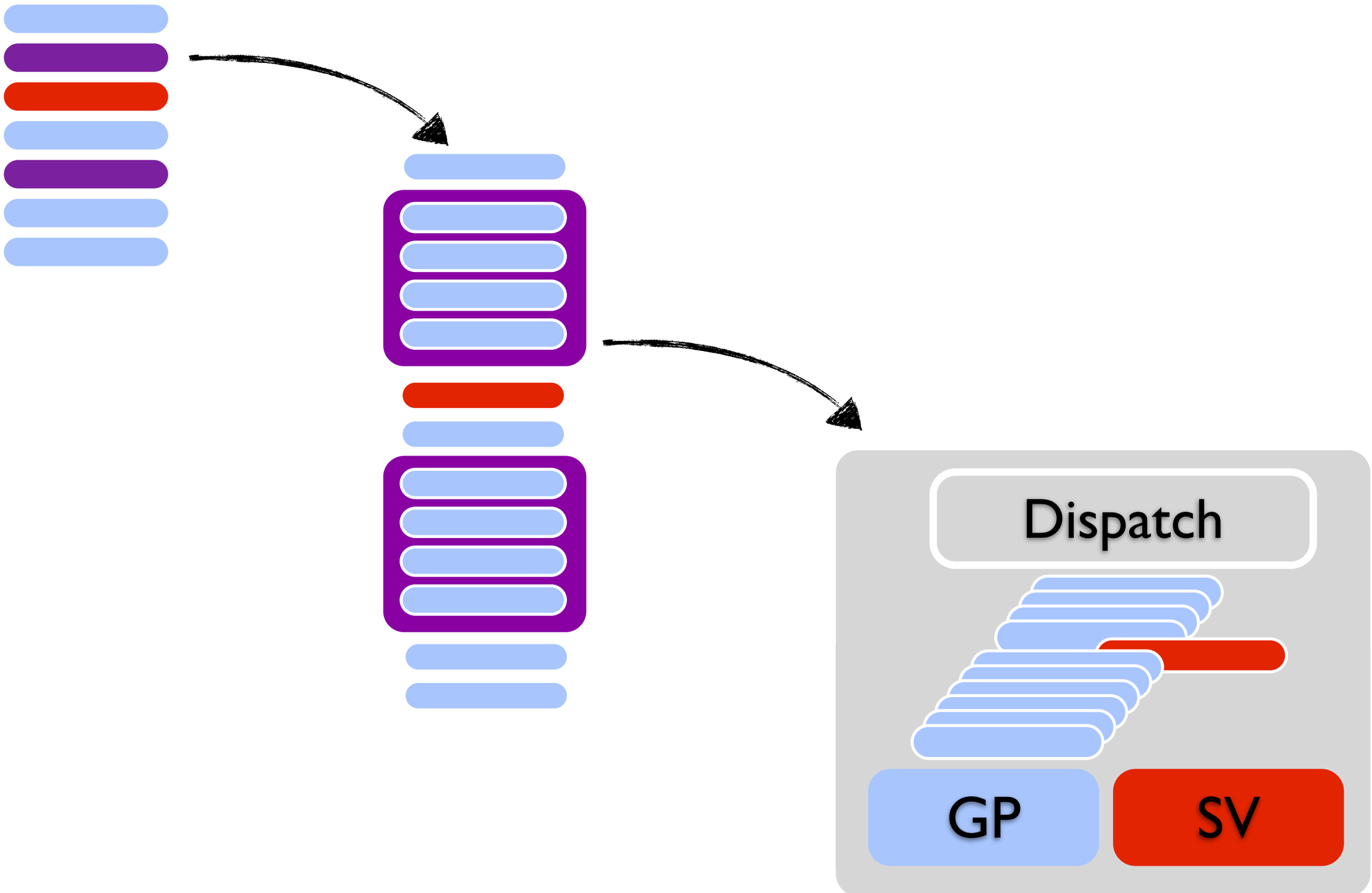
Arch

UArch

Compilation & Execution



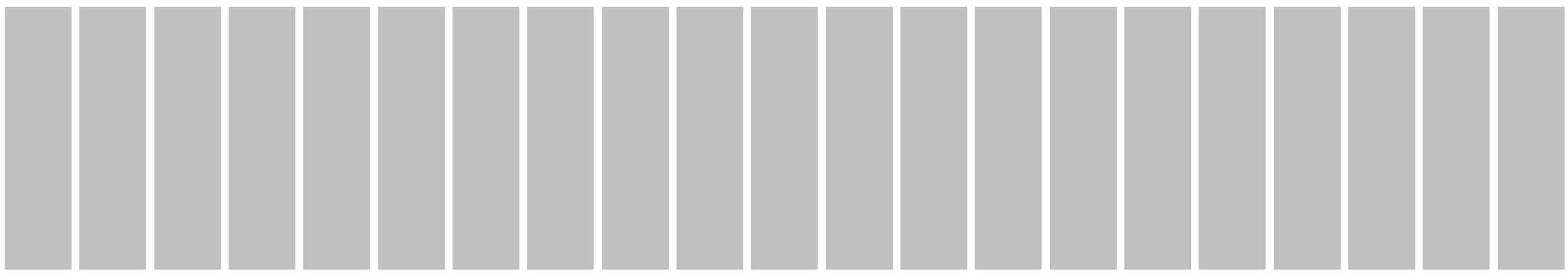
The Software Fallback



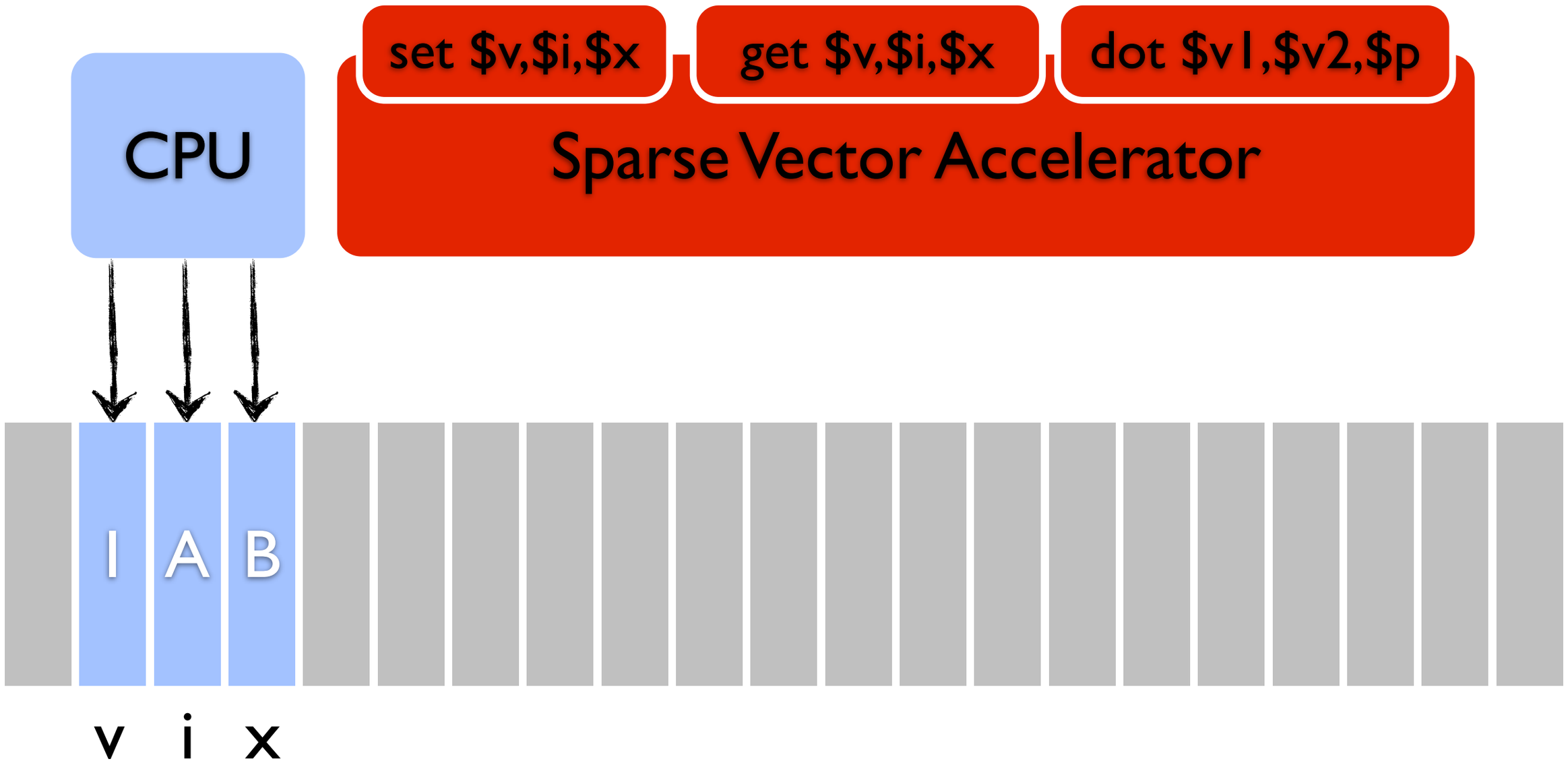
An Ideal Accelerator System

- High Performance
- Low Energy
- Easy Use - align hardware interfaces with those software is already using
- Portability - software fallback plan

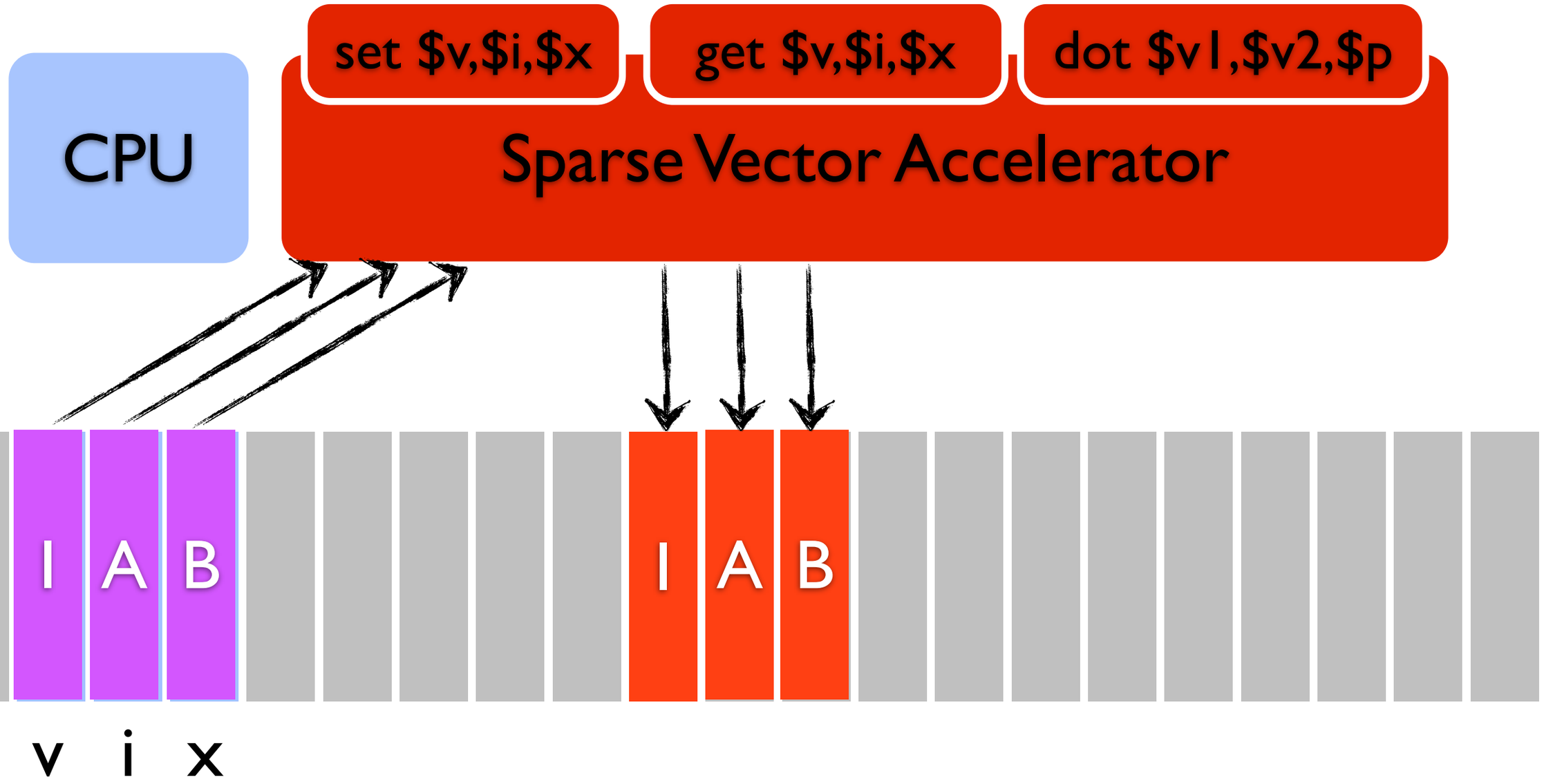
Enforcing Data Encapsulation



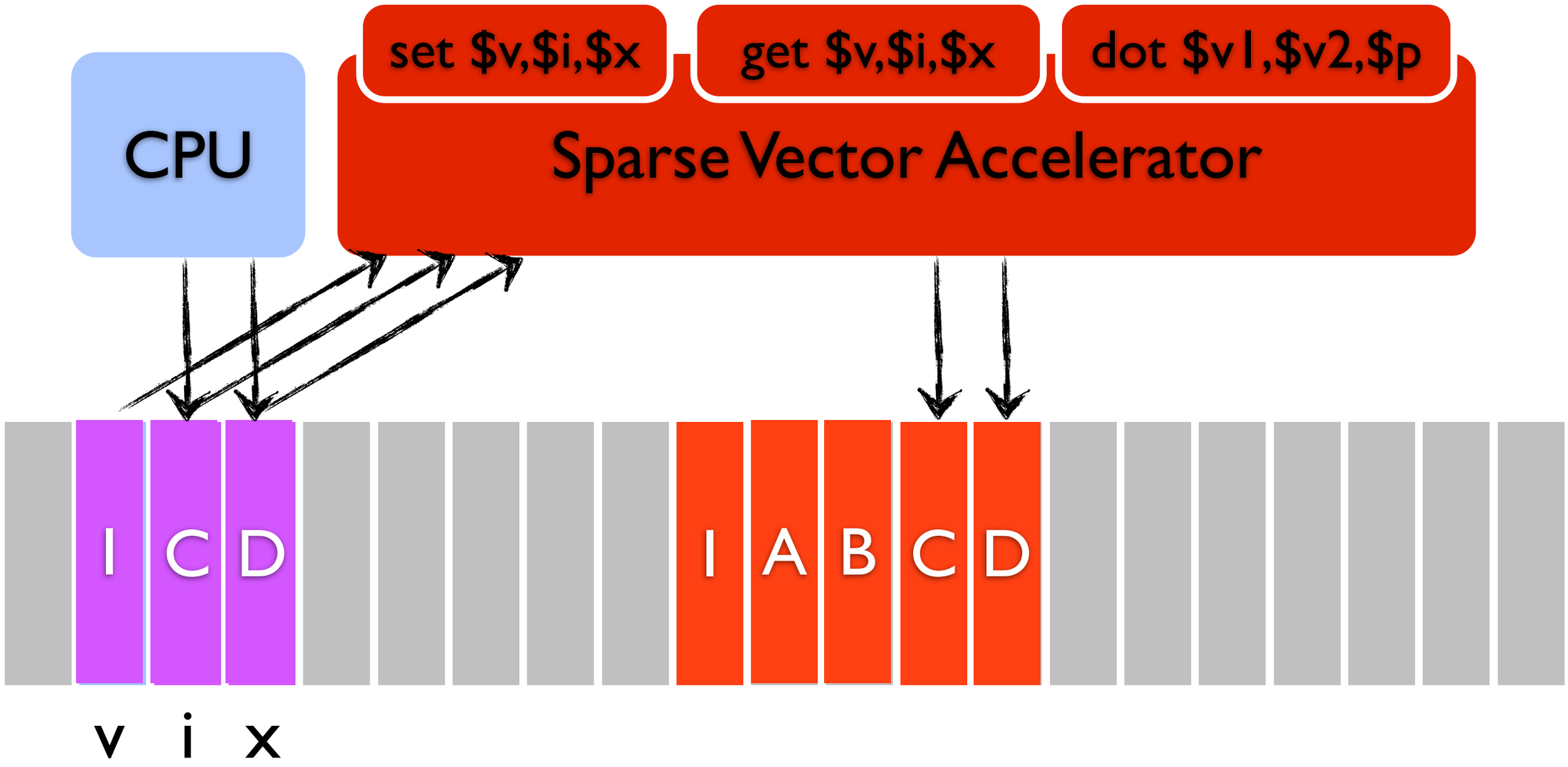
Enforcing Data Encapsulation



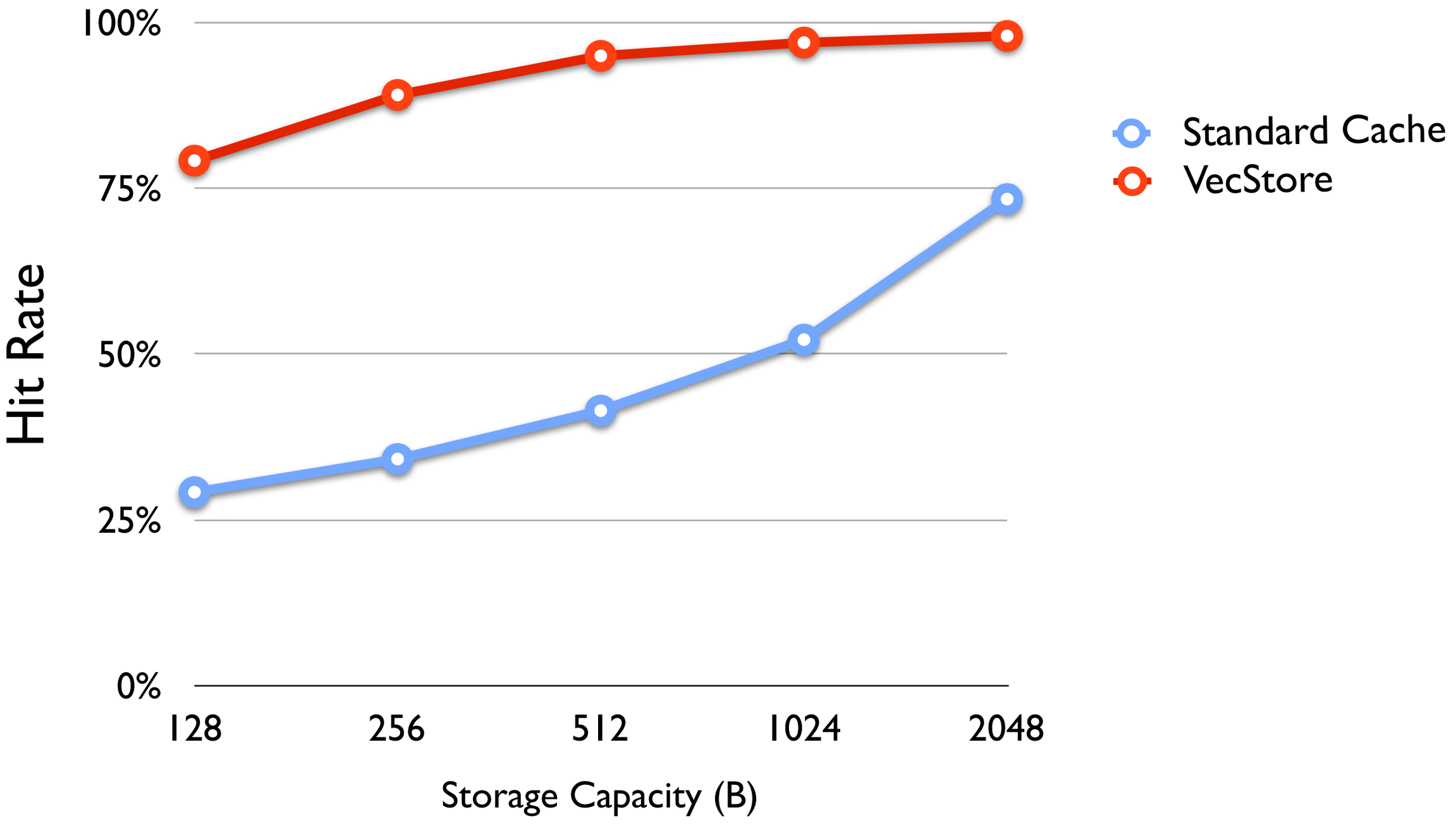
Enforcing Data Encapsulation



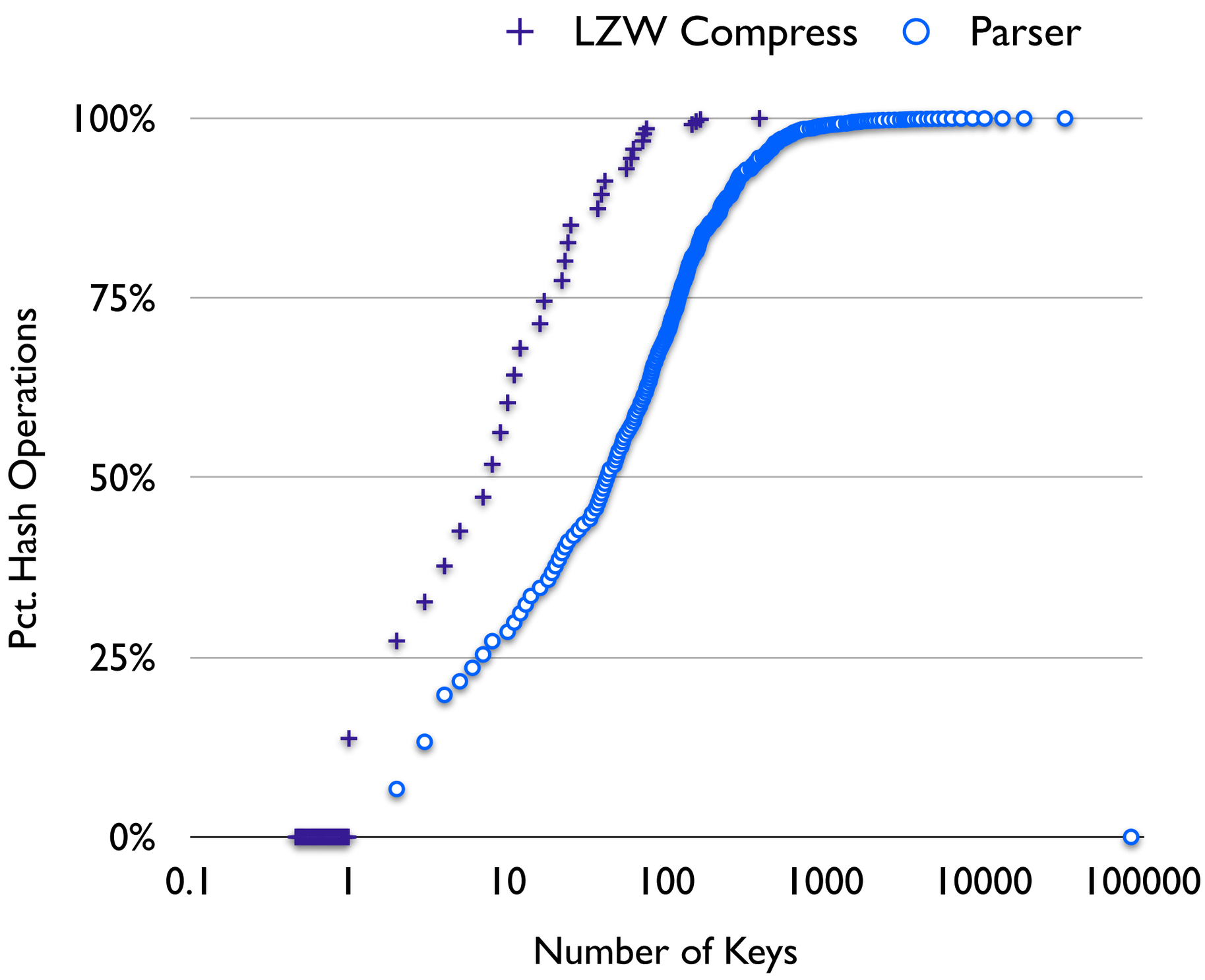
Enforcing Data Encapsulation



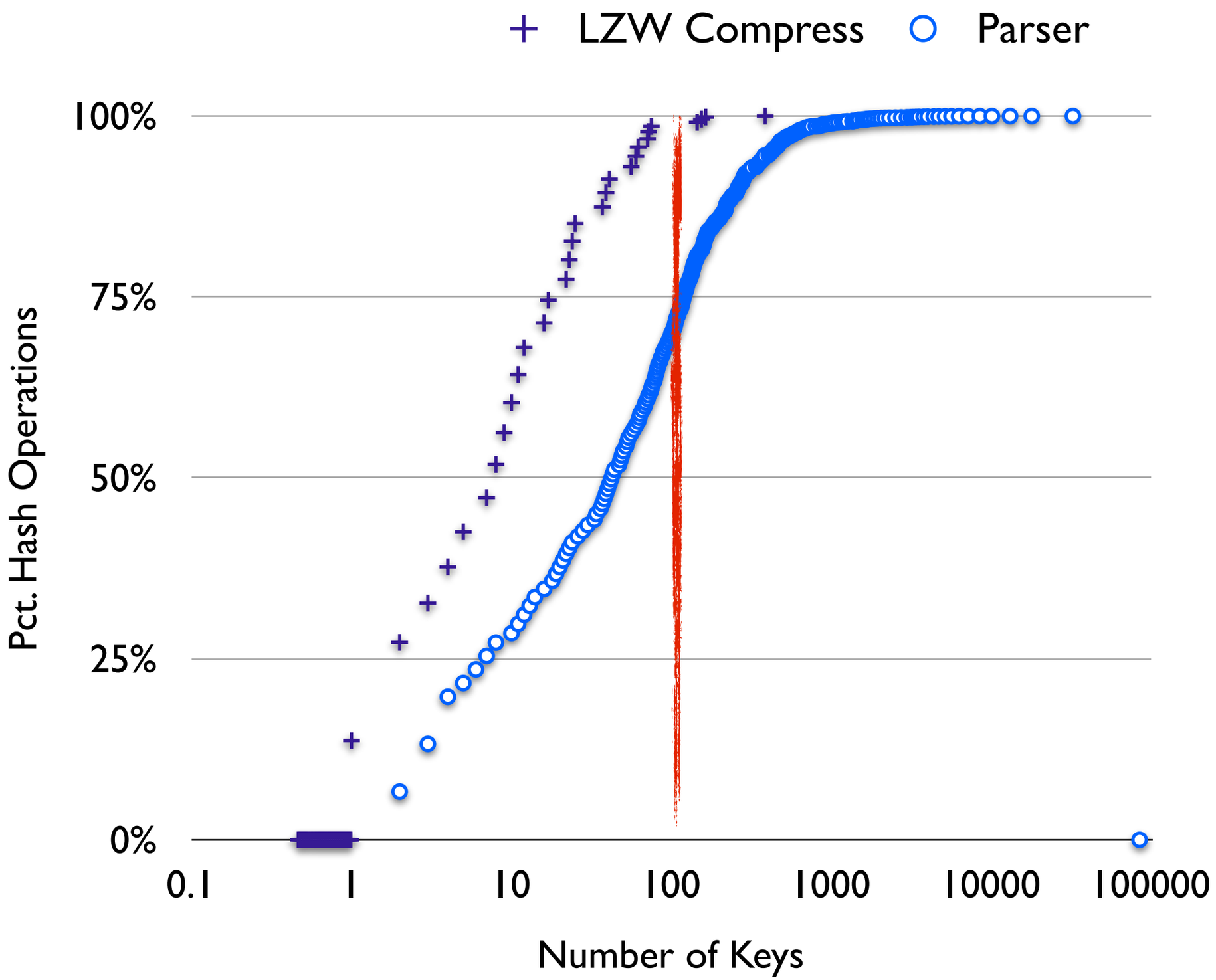
Specialized Caching for Sparse Vectors



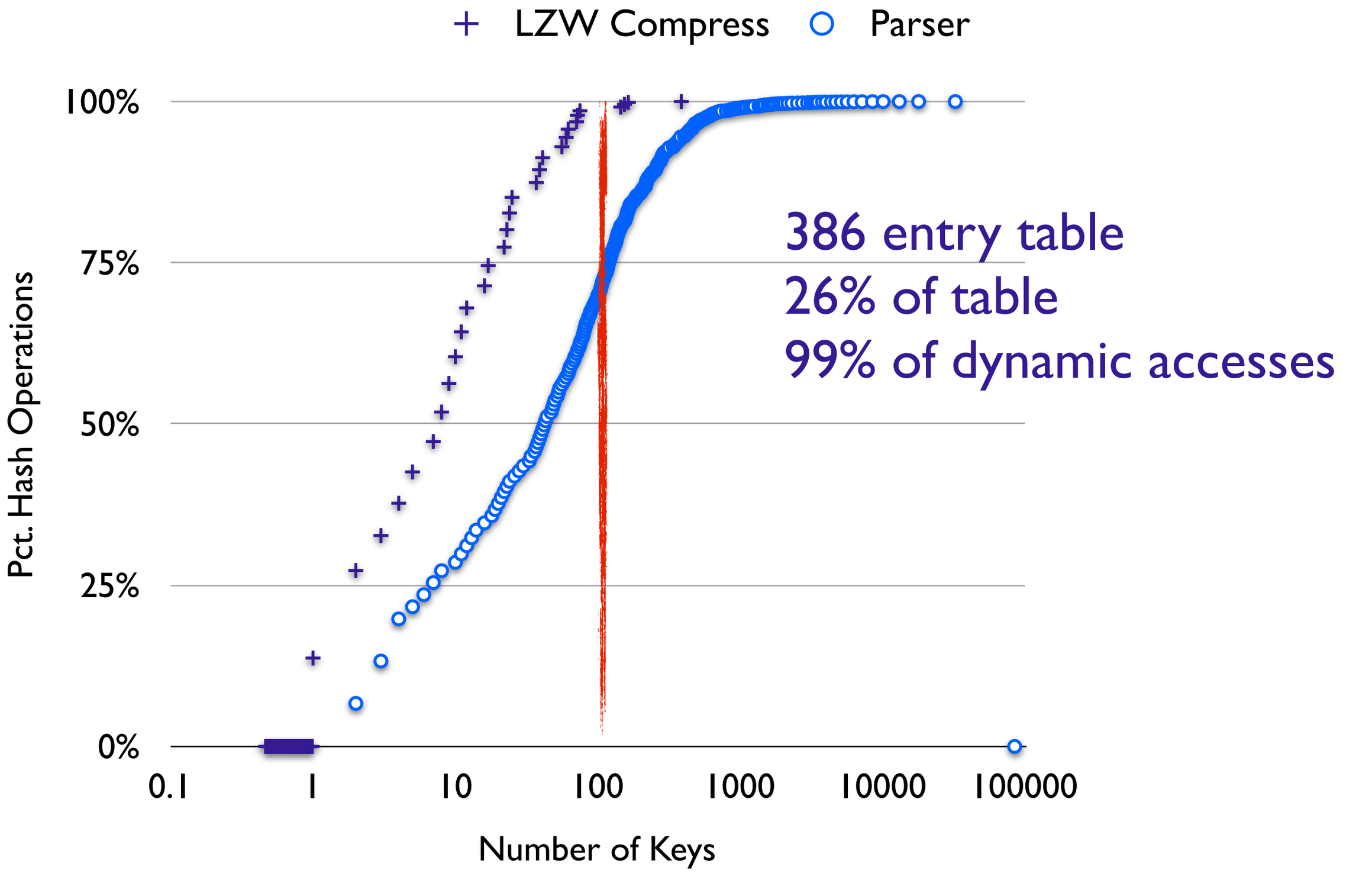
Key Reuse in Hash Tables



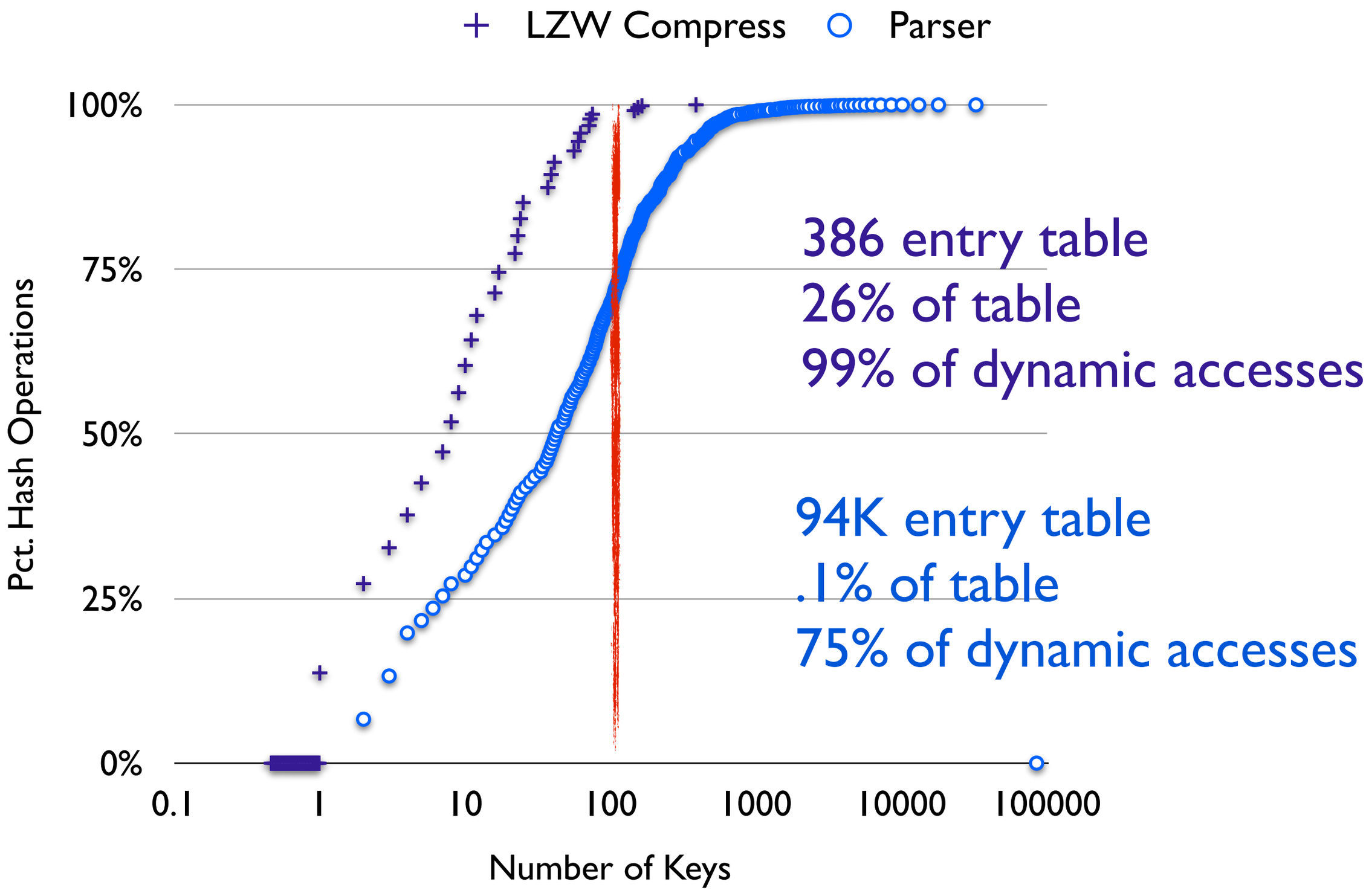
Key Reuse in Hash Tables



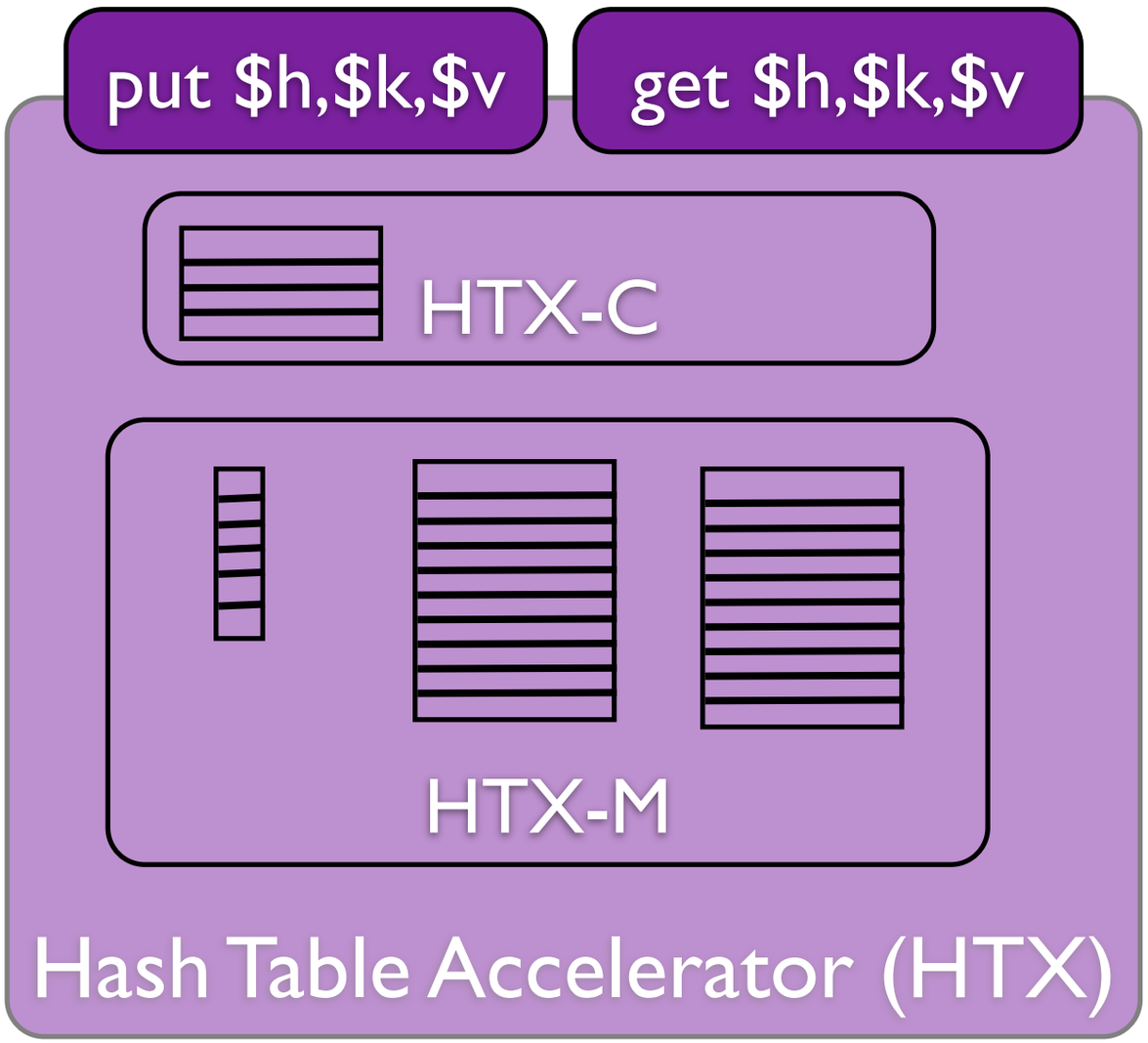
Key Reuse in Hash Tables



Key Reuse in Hash Tables

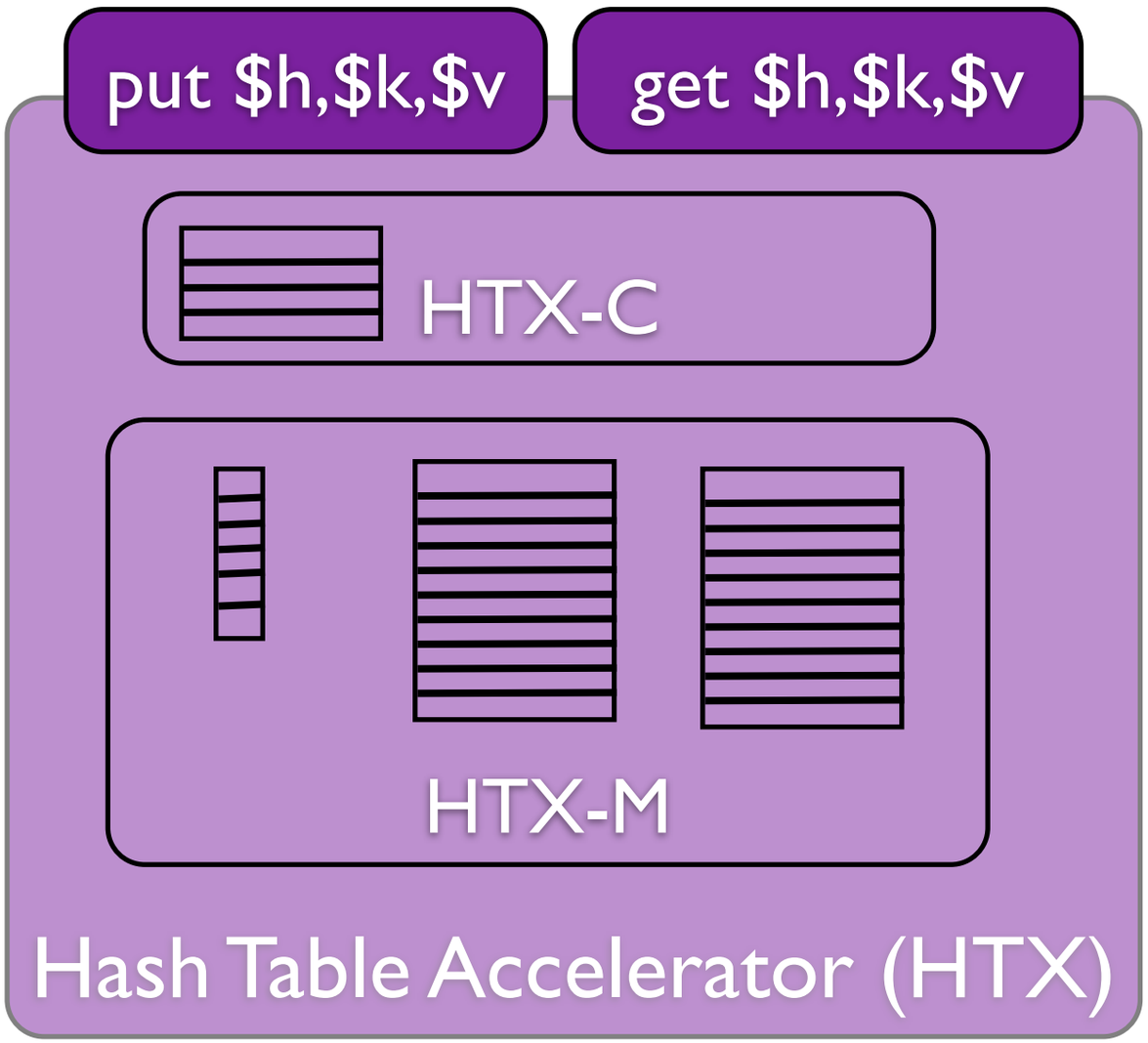


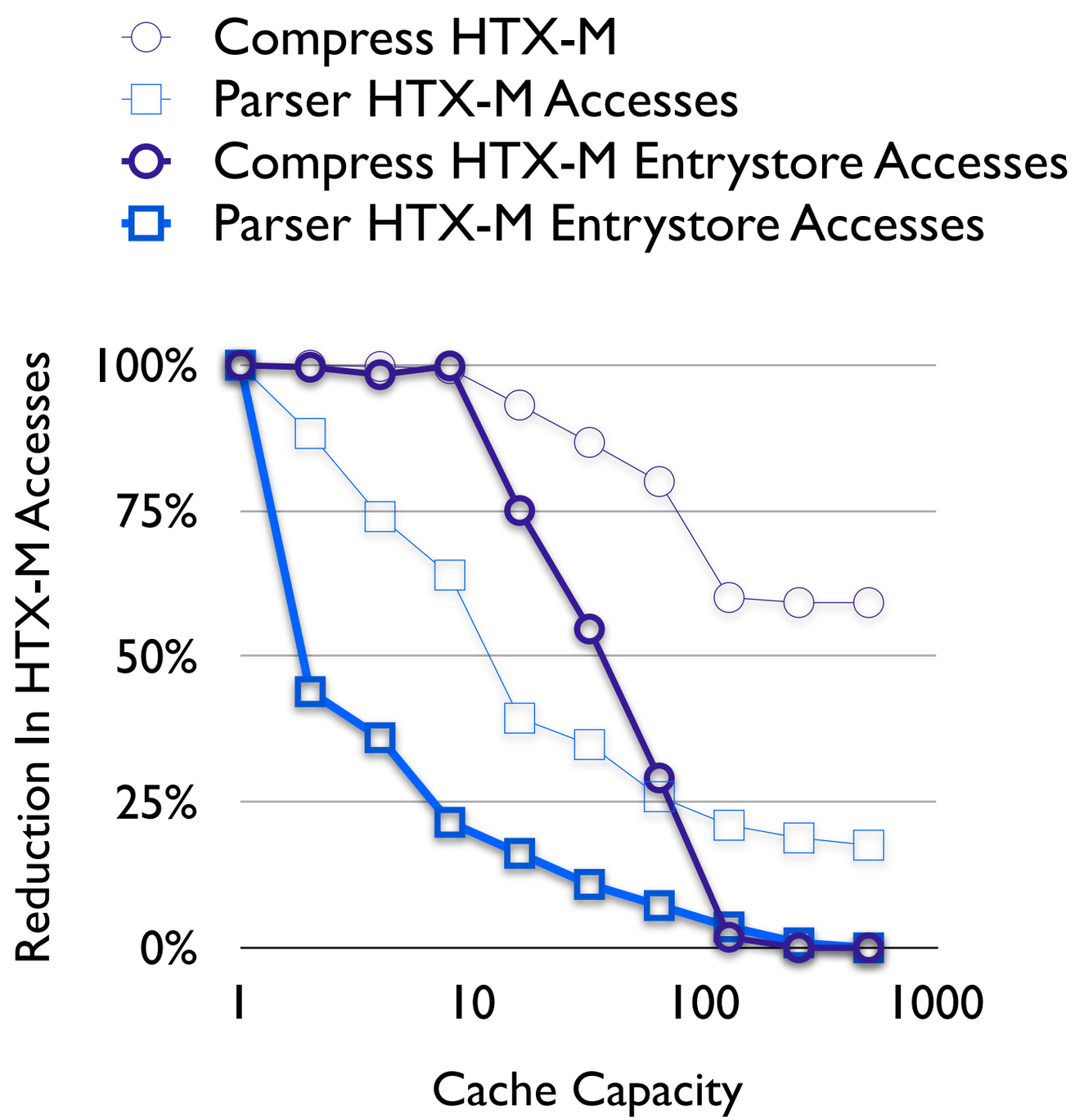
Exploiting Key Reuse



- Compress HTX-M
- Parser HTX-M Accesses
- Compress HTX-M Entrystore Accesses
- Parser HTX-M Entrystore Accesses

Exploiting Key Reuse





Summary

Extend software's encapsulated datatypes into hardware accelerators

Natural alignment with standard software engineering

Accelerator utility on all applications that use a particular type

A software fallback that ensures portability

Aggressive optimization of computation and data movement

Research Challenges

- What are the appropriate types to target? What is the lower bound in complexity? Is there a max number of types a hardware system can support?
- How do I implment polymorphism efficiently? (e.g., priority queue with arbitrary types and user-defined sort function)
- How do I optimized enforcement of data encapsulation? (copy-on-read is conservative)
- Can the execution model support parallel execution?
- What is type-specific coherence like? Simpler? Uglier?
- What is the appropriate system-level resource allocation between general and specialized? Between different types?

Thank You