

DECADES:  
Deeply-Customized  
Accelerator-Oriented  
Data Supply Systems  
Synthesis

---

Margaret Martonosi  
H. T. Adams '35 Professor of  
Computer Science  
Princeton University



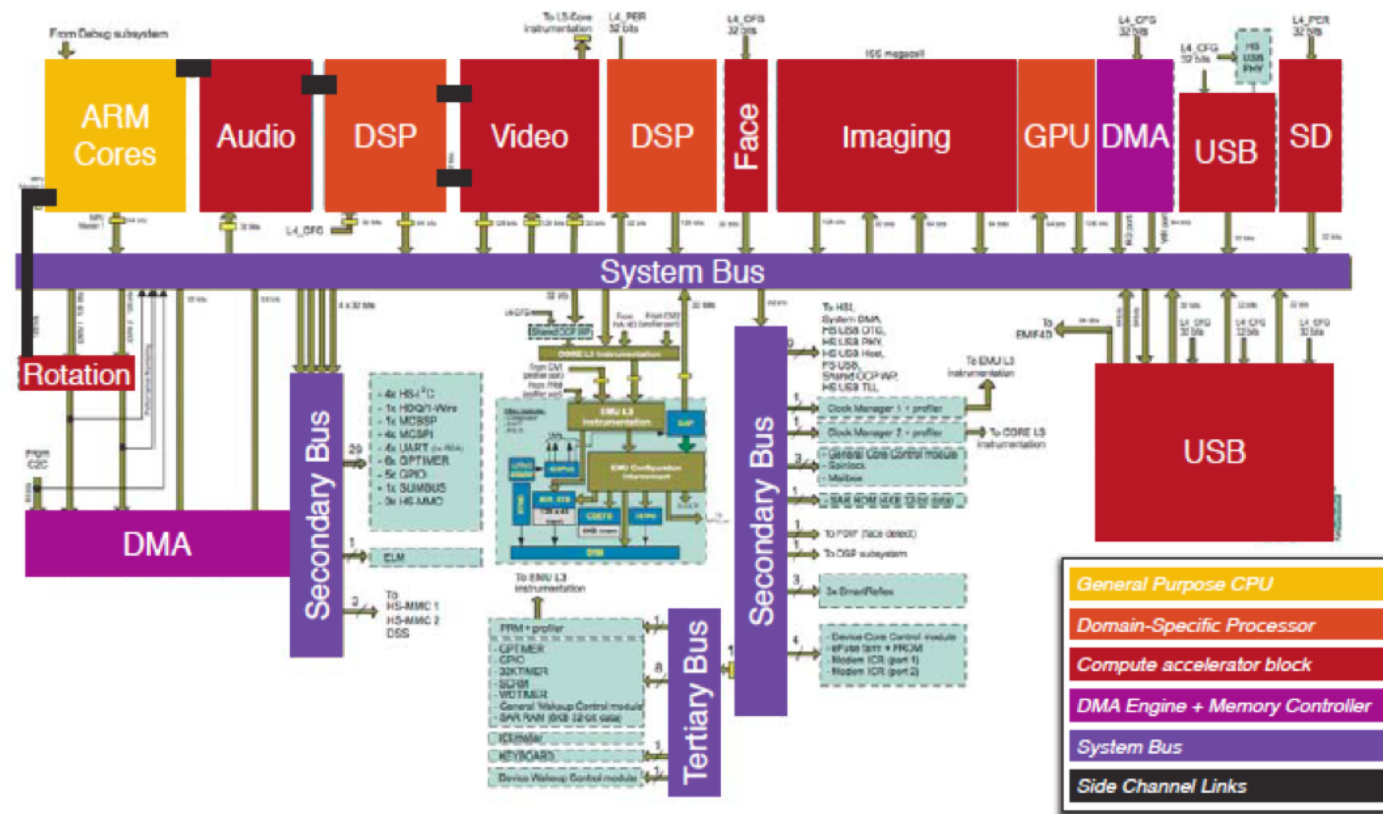
**PRINCETON  
UNIVERSITY**



**COLUMBIA  
UNIVERSITY**

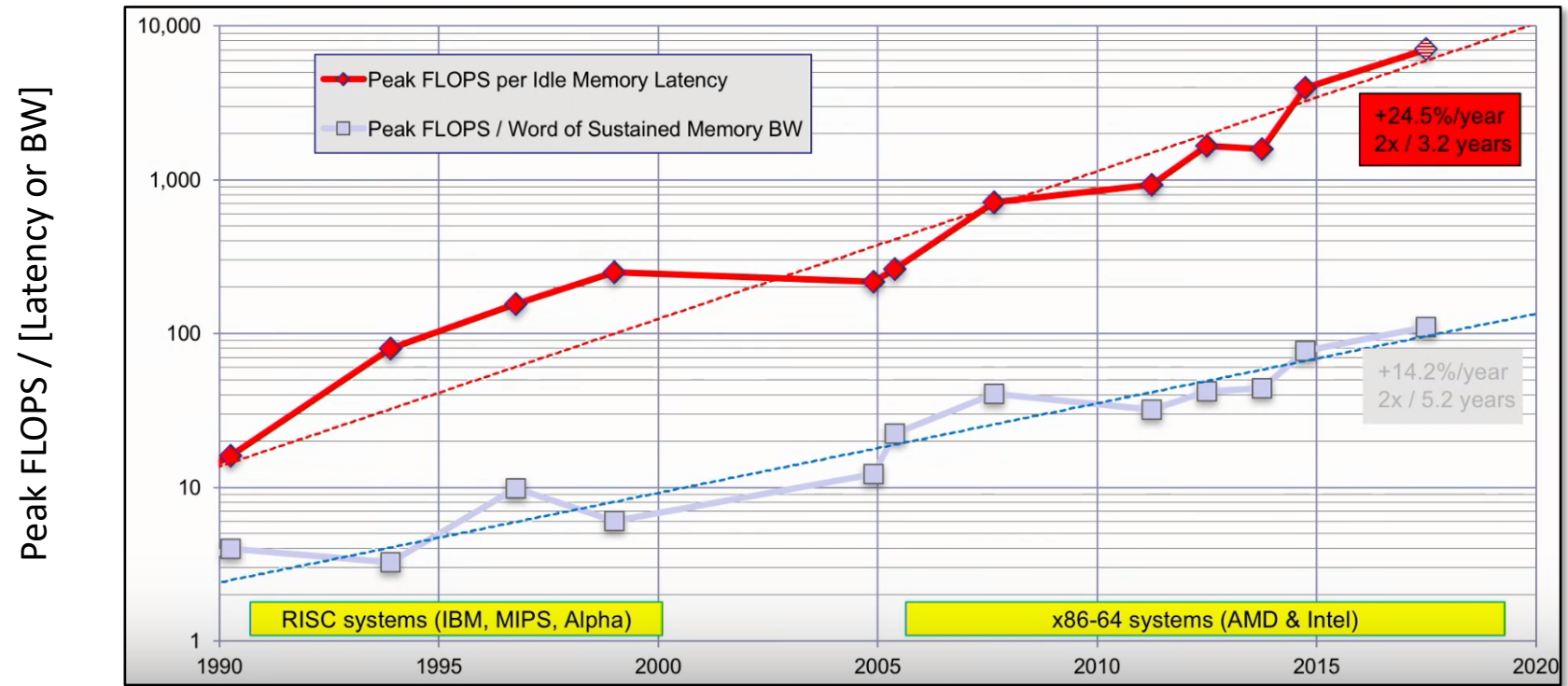
# The Data Supply Challenge

- Modern computer systems are increasingly heterogeneous
- Accelerator-oriented parallelism to meet aggressive performance and power targets
- As accelerators have sped up compute portions, the main challenge is data supply



# Data Supply = Fundamental Bottleneck in Accelerator-Oriented Systems

- **Amdahl's Law:**  
Accelerating compute makes data supply bottlenecks look relatively bigger!
- Key memory/comm bottlenecks lie in supplying specialized accelerators with data
- Different apps -> different data supply needs



John McCalpin, SC'16 Keynote

## Latency and Bandwidth:

- Accelerator often **lacks general-purpose latency-tolerance** mechanisms (e.g., OoO execution, Multithreading)
- Improving Accelerator compute throughput **increases memory bandwidth pressure**

# Our Solution

- Automatically synthesize Data Supply Systems
- Optimizing for Performance and Energy...
- In a full-stack application-specialized way...
- Addressing both latency and bandwidth

# DECADES: A VERTICALLY-INTEGRATED APPROACH

## Language and Compiler Support

- Enhance data locality
- Optimize spatial mapping of threads
- Enable in-memory computing

## Very Coarse-Grained Reconfigurable Tile-Based Architecture

- Coarser than CGRA → VCGRTA
- 3 classes of reconfigurable tiles
- Reconfigurable interconnection network
- Reconfigurable in-memory computing

## Multi-Tiered Demonstration Strategy

- Scalable full-system simulation
- Multi-FPGA emulation infrastructure
- 225-tile DECADES chip prototype

# DECADES: A VERTICALLY-INTEGRATED APPROACH

Language and  
Compiler Support

Lead: Martonosi

- Enhance data locality
- Optimize spatial mapping of threads
- Enable in-memory computing



Very Coarse-Grained  
Reconfigurable  
Tile-Based Architecture

Lead: Carloni

- Coarser than CGRA → VCGRTA
- 3 classes of reconfigurable tiles
- Reconfigurable interconnection network
- Reconfigurable in-memory computing



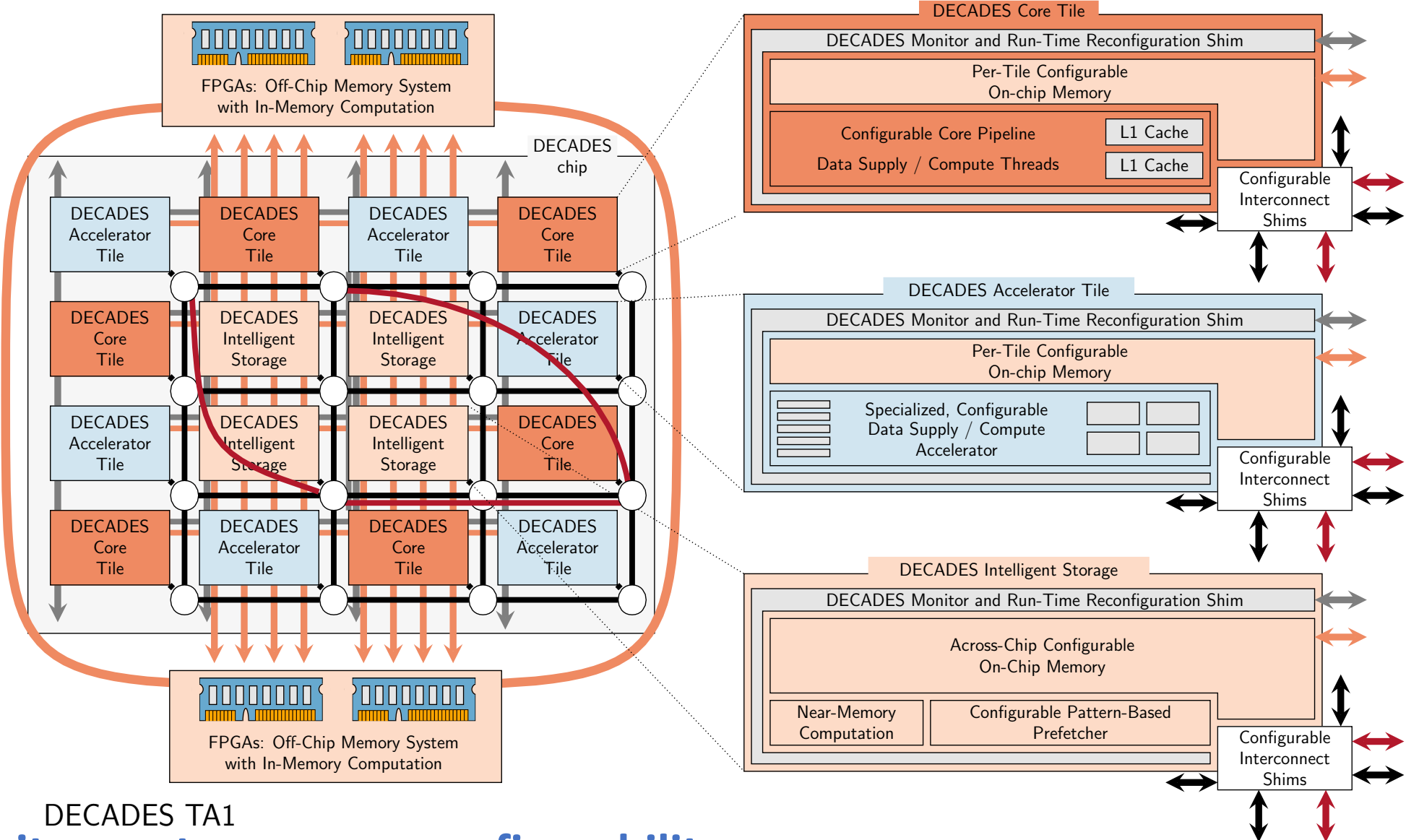
Multi-Tiered  
Demonstration Strategy

Lead: Wentzlaff

- Scalable full-system simulation
- Multi-FPGA emulation infrastructure
- 225-tile DECADES chip prototype



# DECADES PLATFORM ARCHITECTURE

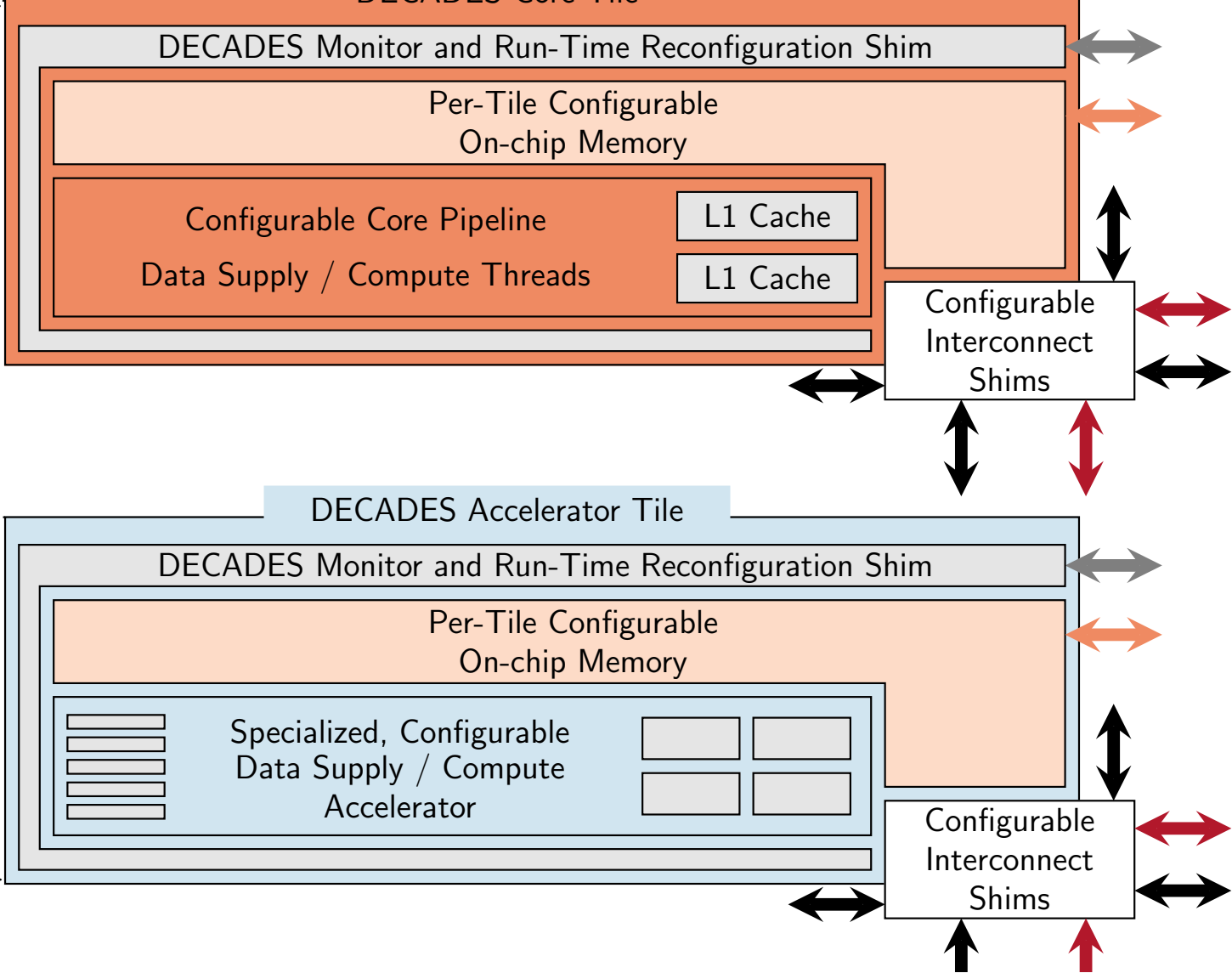


DECADES TA1

**Heterogeneity meets coarse reconfigurability**

# DECADES Core and Accelerator tiles

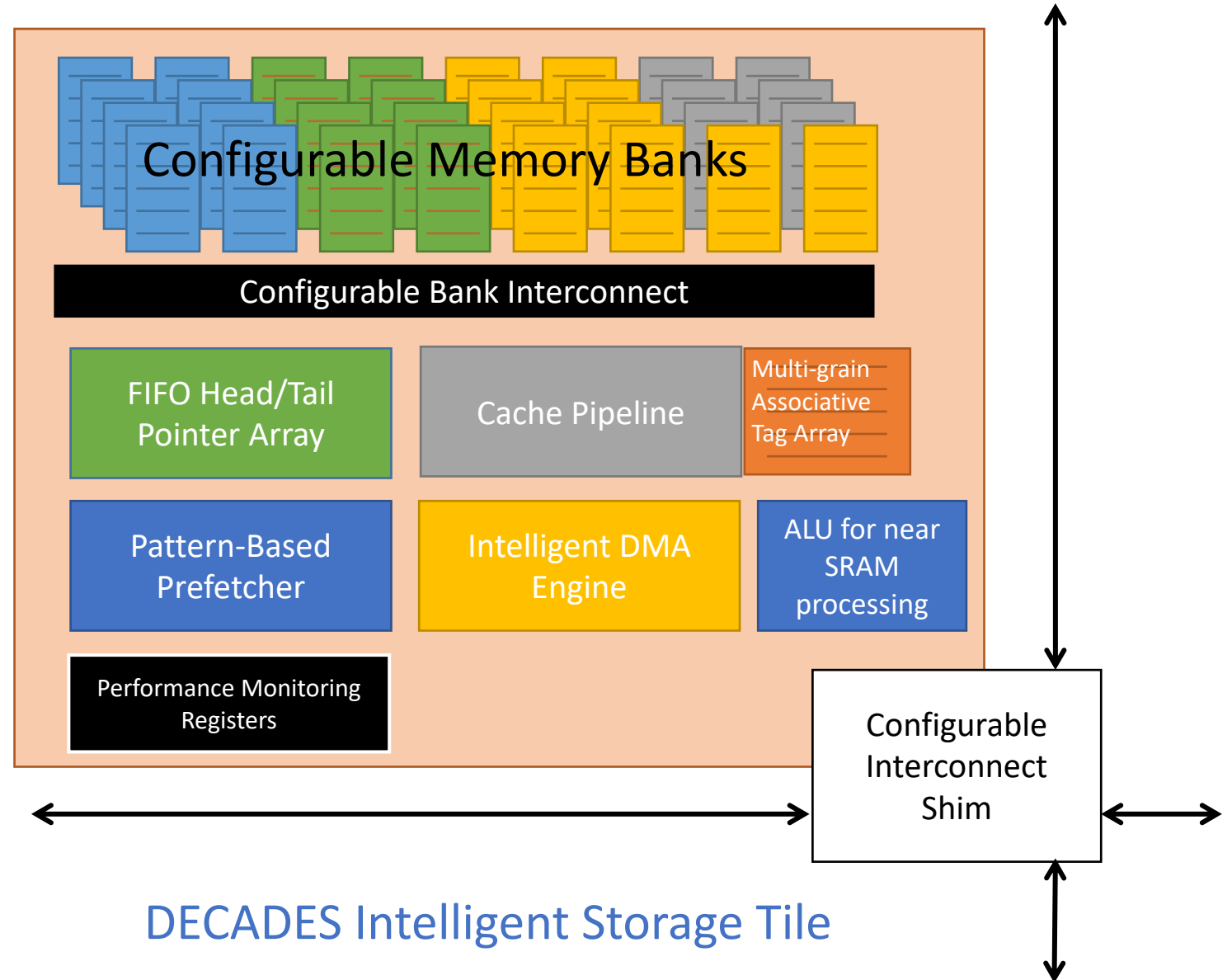
- Computations mapped onto core tiles or available accelerator tiles
  - Each tile is wrapped in monitor/reconfiguration shim
- Dynamic reconfiguration of Supply-Compute decoupling, power-performance tradeoffs, and interconnect





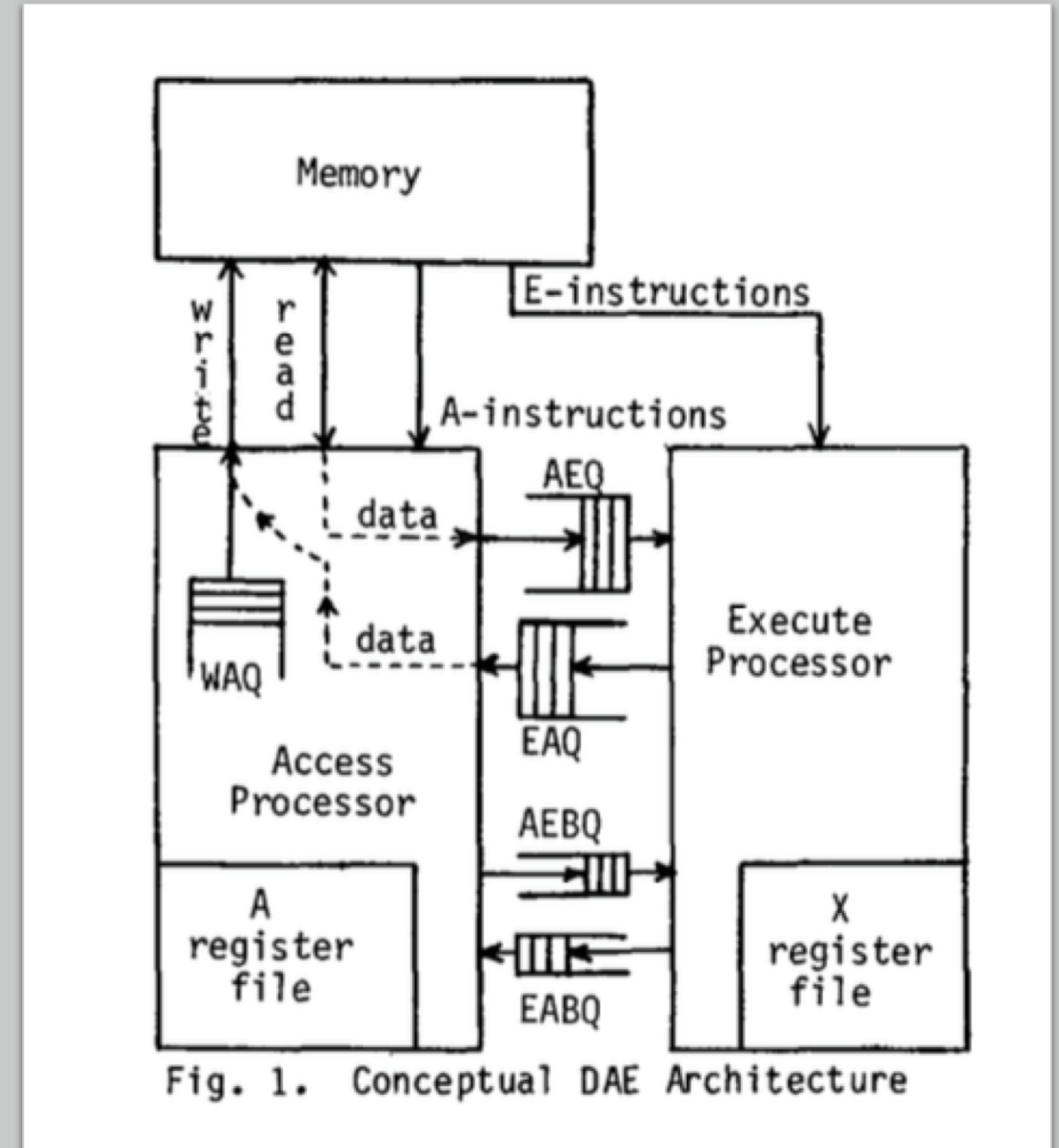
# DECADES Storage Specialization

- Specialization #1:  
Map apps onto mix of compute tiles and intelligent storage (IS) tiles
- Specialization #2:  
Select and configure appropriate storage features within IS
  - Configurable memory banks + address and prefetching features
  - Simple near-SRAM ALU



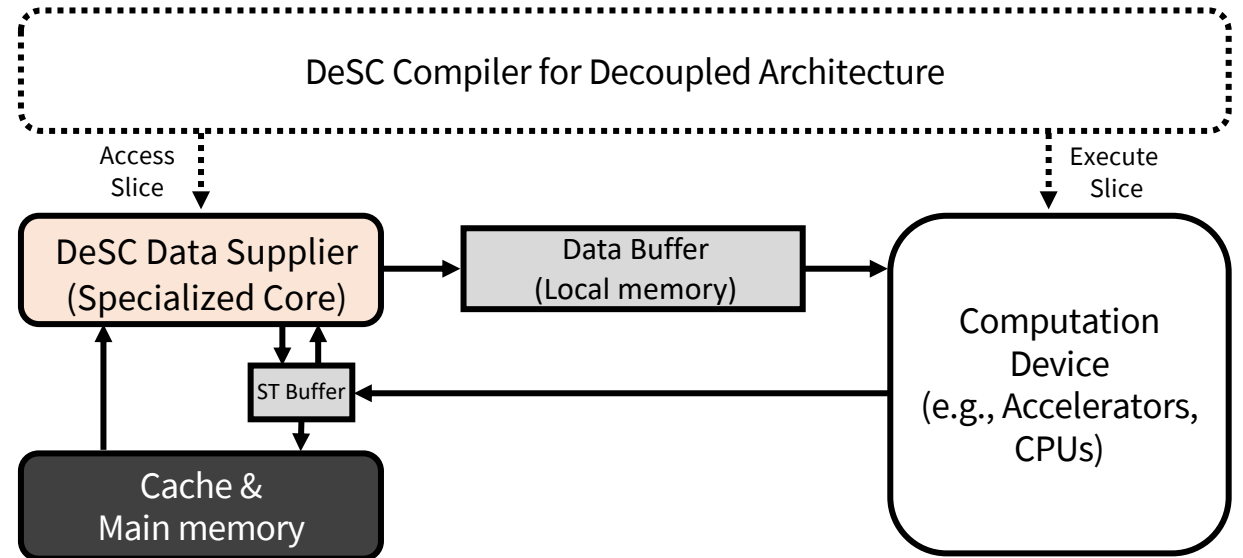
# Improving Latency-Bound Applications with Decoupled Execution

- Roots in seminal 1982 Smith DAE paper: Separately execute memory accesses (supply) from instructions that compute with them (compute)
- **Then:** Latency tolerance “simpler” than out-of-order execution
- **Now:** Fits well with accelerator-oriented design
  - Separate memory supply from accelerator
  - Orthogonal to DOALL parallelism and bandwidth optimizations
  - Automatically identify and slice at compile time



# Our Prior Work: DeSC

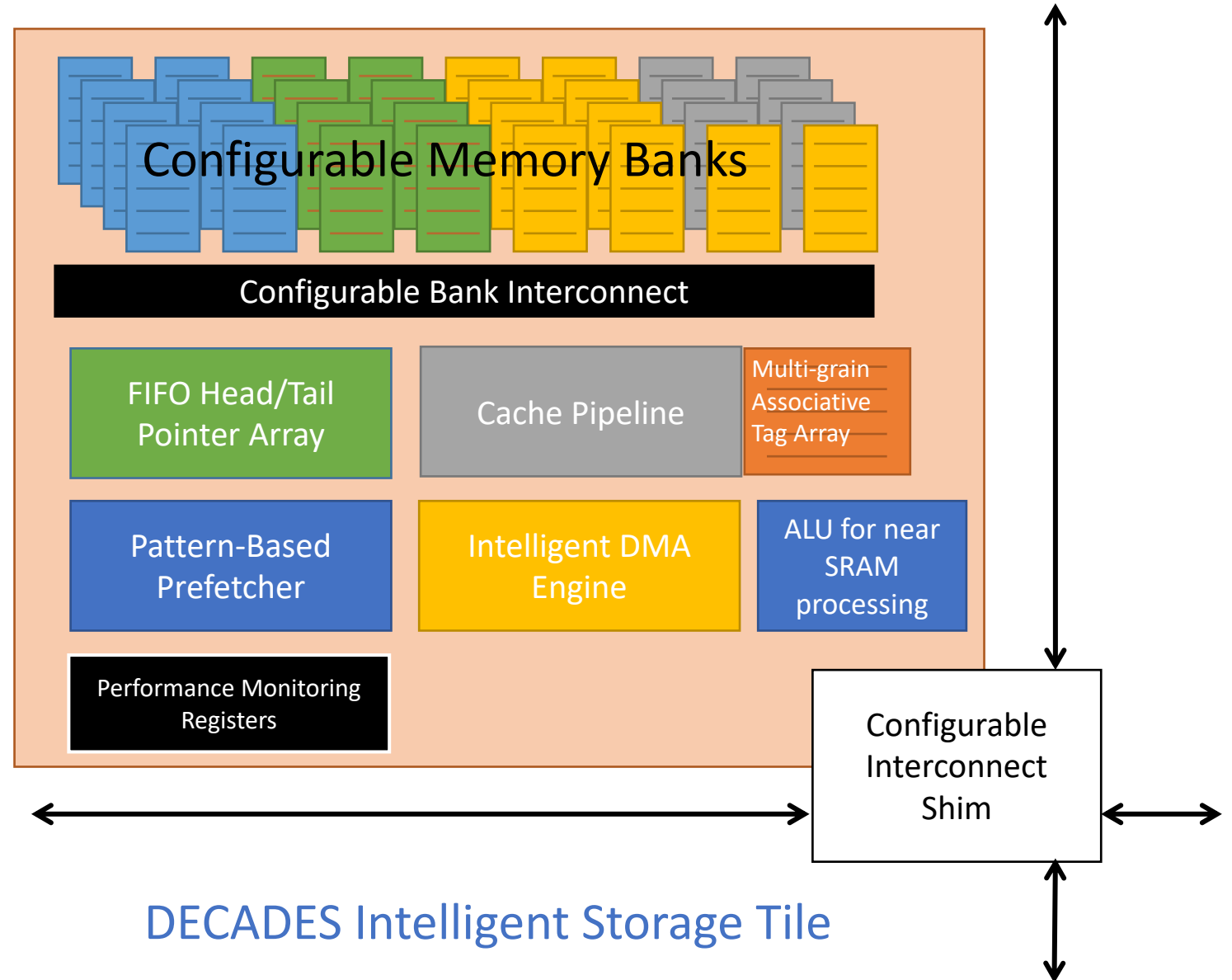
- DeSC: DEcouple data Supply from Compute to provide high latency tolerance without burdening programmers
  - Decouple the target program into two slices with DeSC Compiler Tool



- **Access** Slice -> Specialized DeSC Data Supplier
  - Accesses data from the memory and supplies data to Execute Slice
  - Runahead for latency tolerance
- **Execute** Slice -> Computation device
  - Retrieve data sent from the Access Slice. Performs computation and send result back to Access Slice.
- Performance: 8X better for memory bounded. Further multiplicative speedups with N DOALL pairings of Access and Execute.

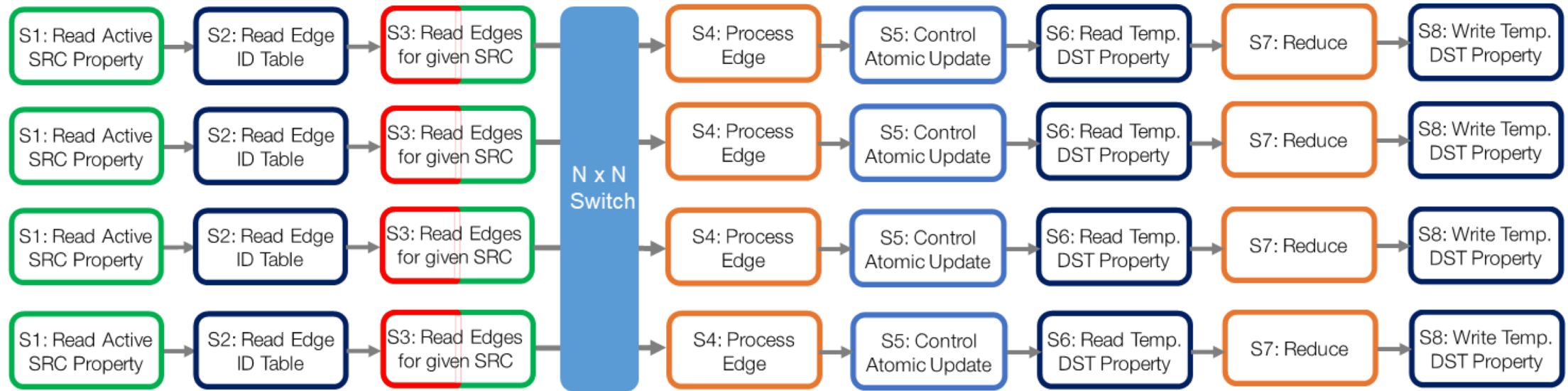
# Improving **Bandwidth-Bound** Applications

- Scratchpad vs. Cache vs. Queue
- Configurable fetch granularity
- Customized tile-to-tile flow
- Application-specific prefetching
- Simple near-SRAM ALU



Application Example:  
Graphicionado on  
DECADES

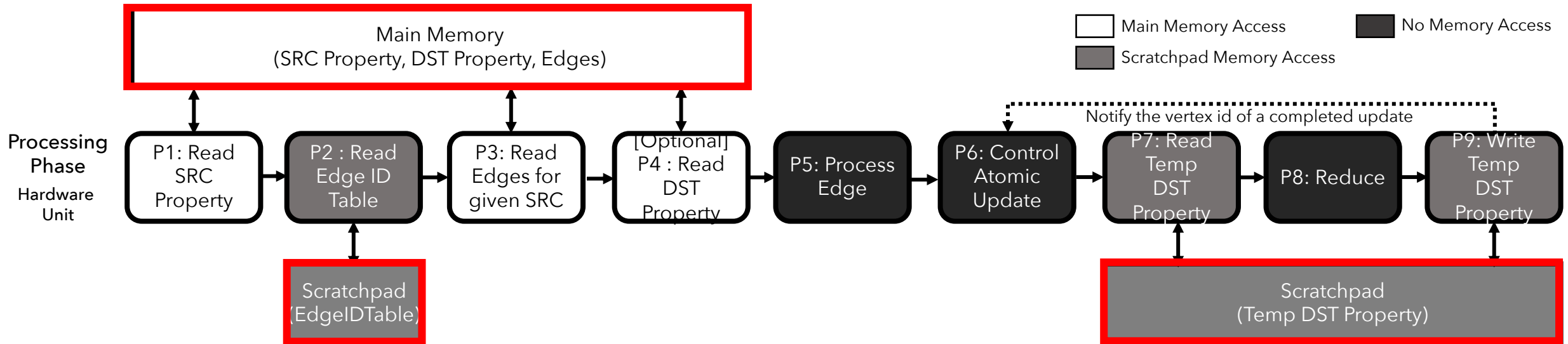
# Our Prior Work: Graphicionado



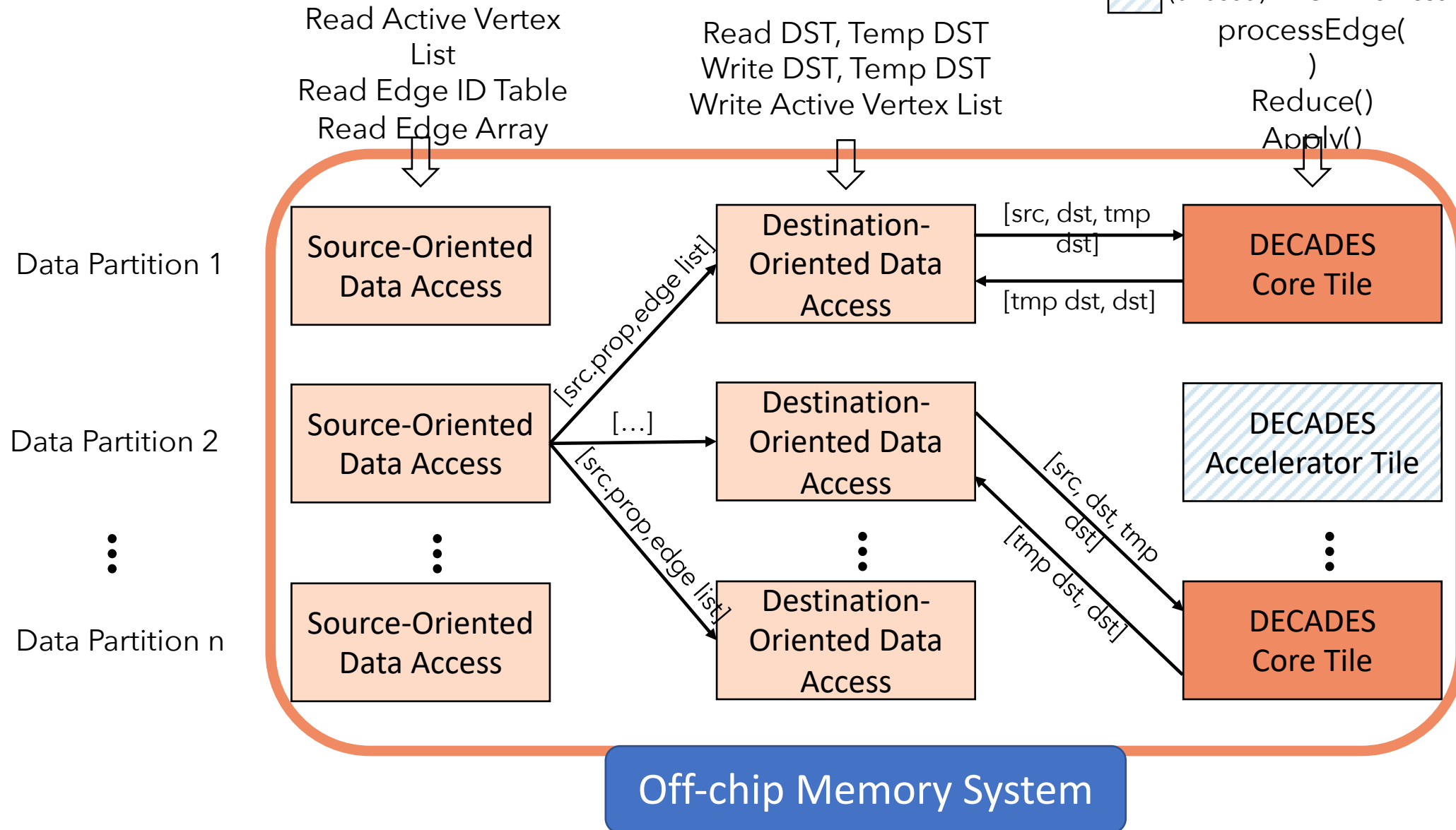
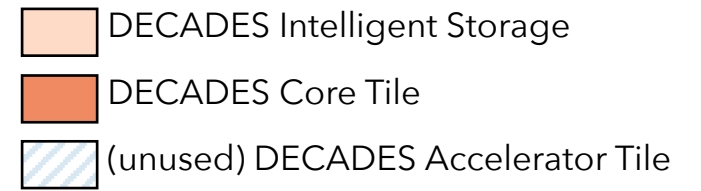
- Application-Specific Memory Hierarchy for **Bandwidth-Bound Graph Analytics**
  - Customized memory hierarchy to minimize off-chip memory access traffic [**3x reduction**]
  - Dataflow pipeline based on high-level abstraction eases the programming and enables hardware reuse for different graph applications
  - Specialized HW accelerator for graph analytics successfully achieves **~3x speedup** and **50x+ energy saving** compared to state-of-the-art software framework on 32-core CPU

[Ham et al., MICRO-49, 2016. IEEE Micro Top Picks Honorable Mention]

# Graphicionado Memory Specialization



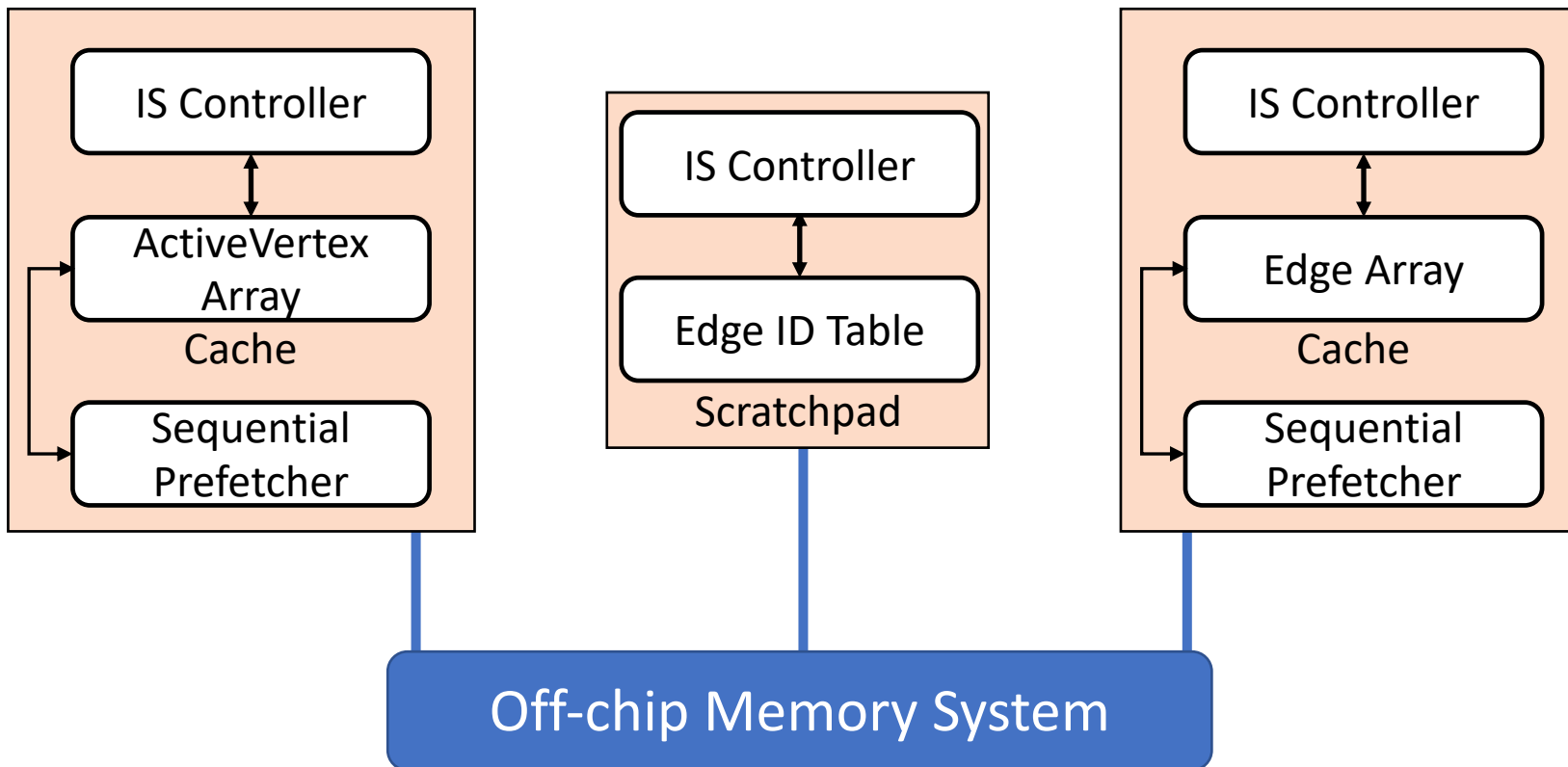
# Mapping Graphicionado to DECADES





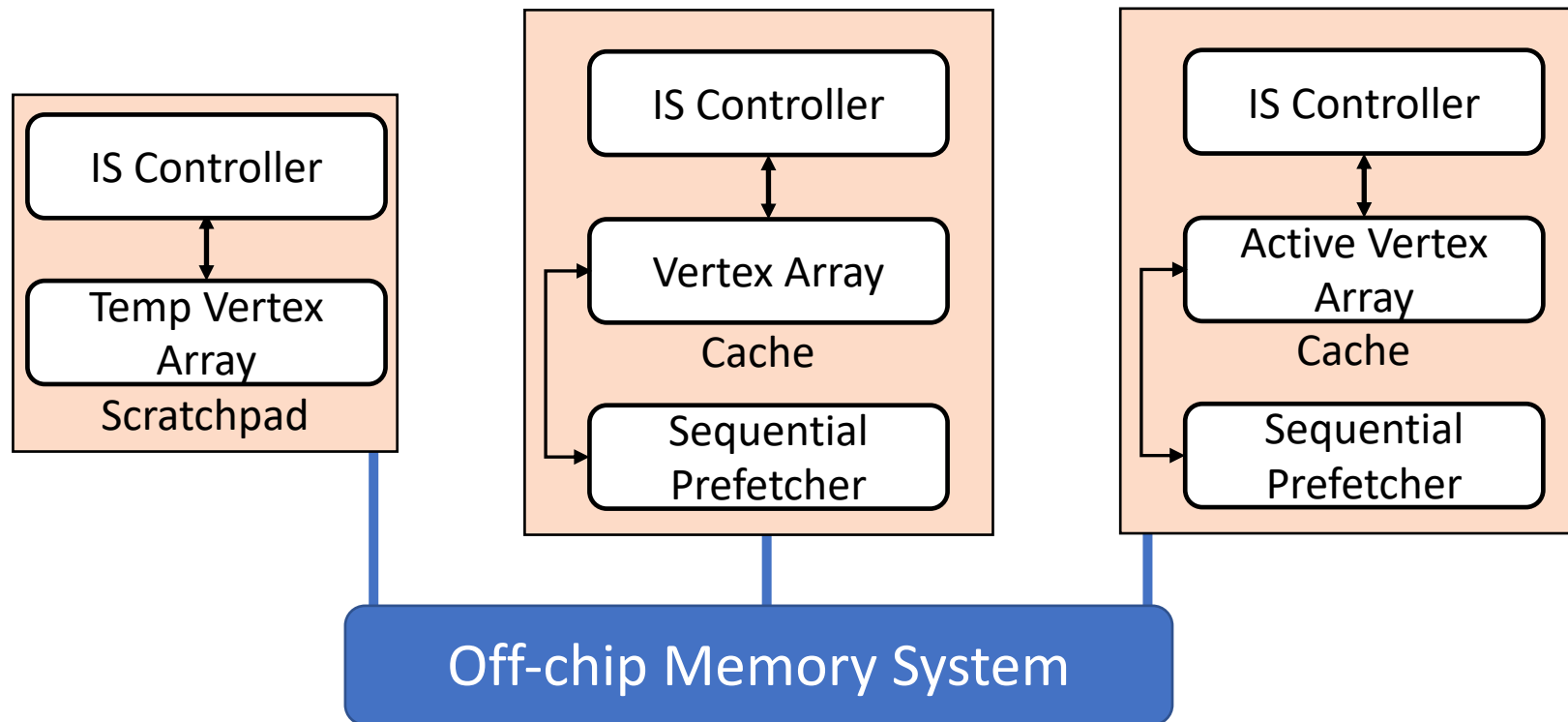
# Source-Oriented Intelligent Storage Tiles

- Intelligent Storage configured with Cache, Scratchpad, and Prefetcher
- Tiles programmed with bulk access directives, destination tile(s)



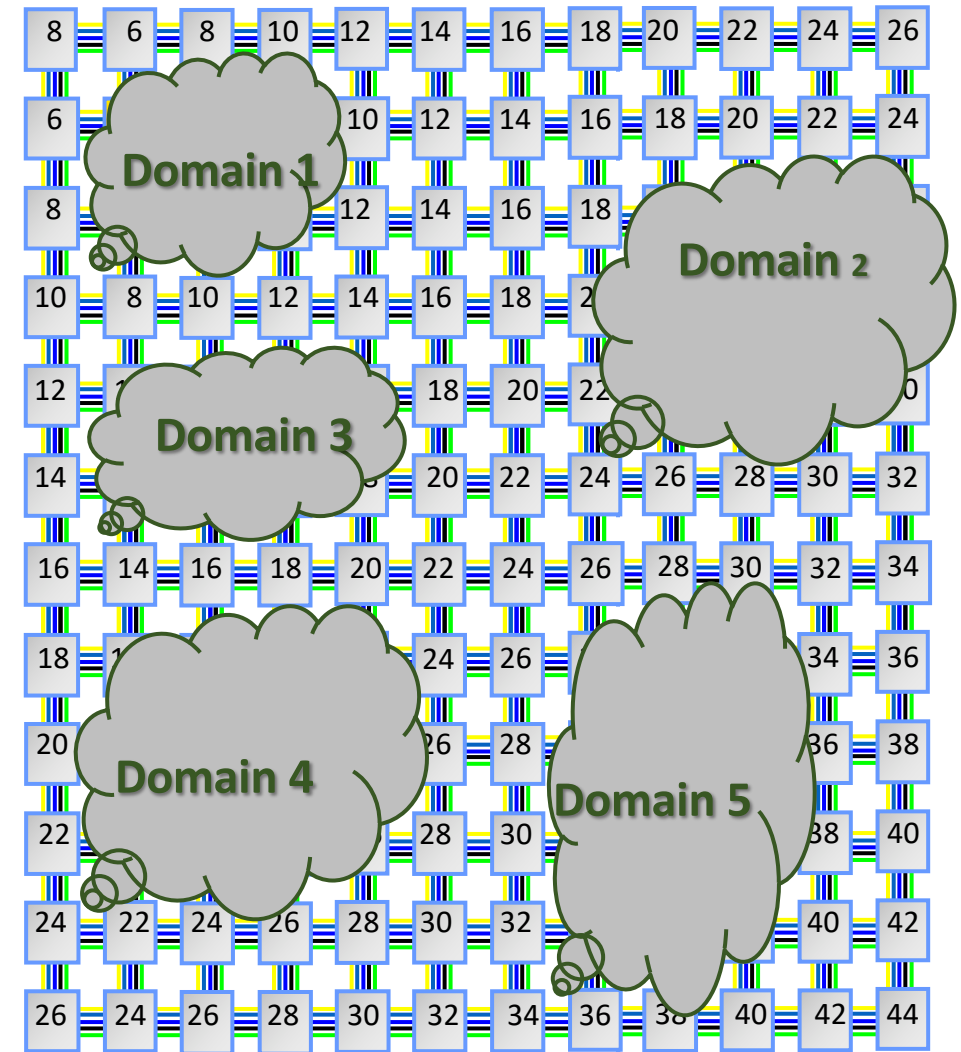
# Destination-Oriented Intelligent Storage Tiles

- Intelligent Storage configured with Cache, Scratchpad, and Prefetcher
- Tiles programmed with bulk access directives, destination tile(s)



# Other DECADES optimizations

- Coherence domain restrictions [Fu et al. MICRO 2015]
- Automatic consistency/coherence protocol optimizations for heterogeneous hardware.
- Per-tile power gating, clock gating, and V/f scaling.
- Turn on/off NoC planes at compile-time or runtime.
- Streamline tile-to-tile data flow
- Novel consistency and transaction models

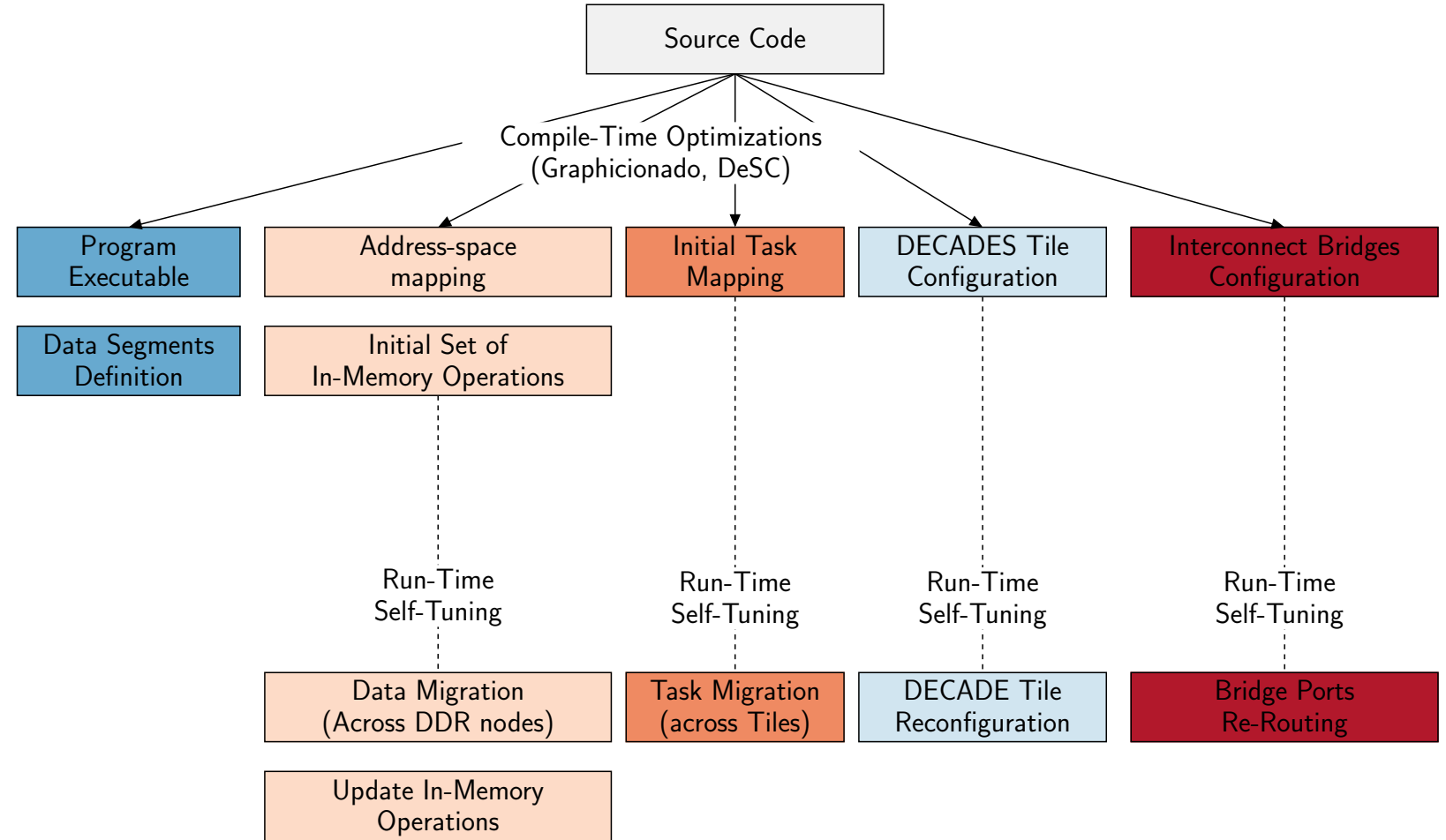


Coherence Domain Restriction.  
[Fu et al. MICRO 2015]

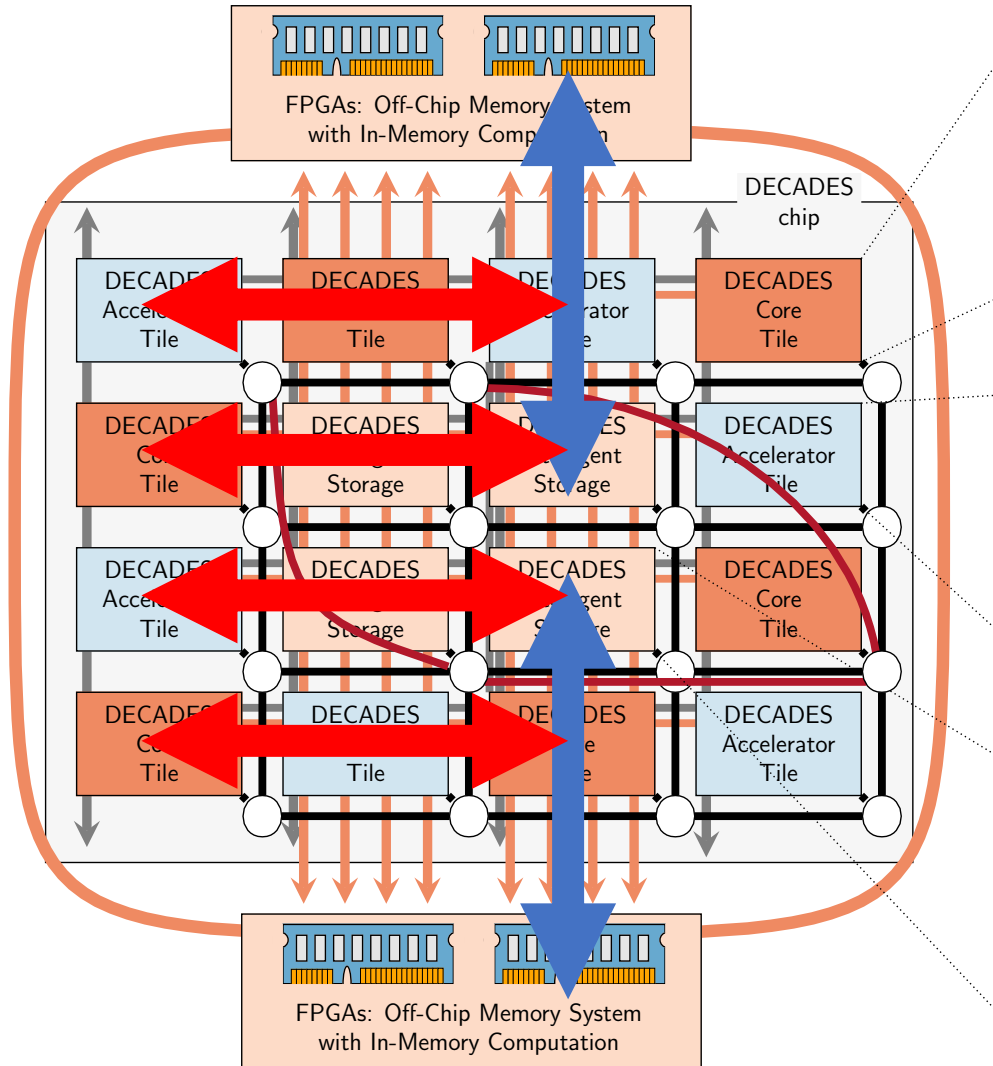
# DECADES Language, Compiler & Runtime System

# LANGUAGE, COMPILER & RUNTIME SYSTEM

- Bandwidth optimizations through cache optimizations and locality/granularity tailoring
- Latency tolerance through decoupling
- Build on DeSC LLVM compiler infrastructure



# Combined Latency and Bandwidth Approaches



**Decouple Supply-Compute for Latency Tolerance**

**Tailor Parallelism, Granularity, and Storage Structures for Bandwidth Optimization**

**Multiplicative Speedup**

Simulation  
Emulation  
Chip Design

# Evaluation Plans

## Design Tradeoffs & Lightweight Simulation

- LLVM IR -> Dependence graph
- Resource limits -> Timing, Power, Area

## QEMU and FPGA Emulation

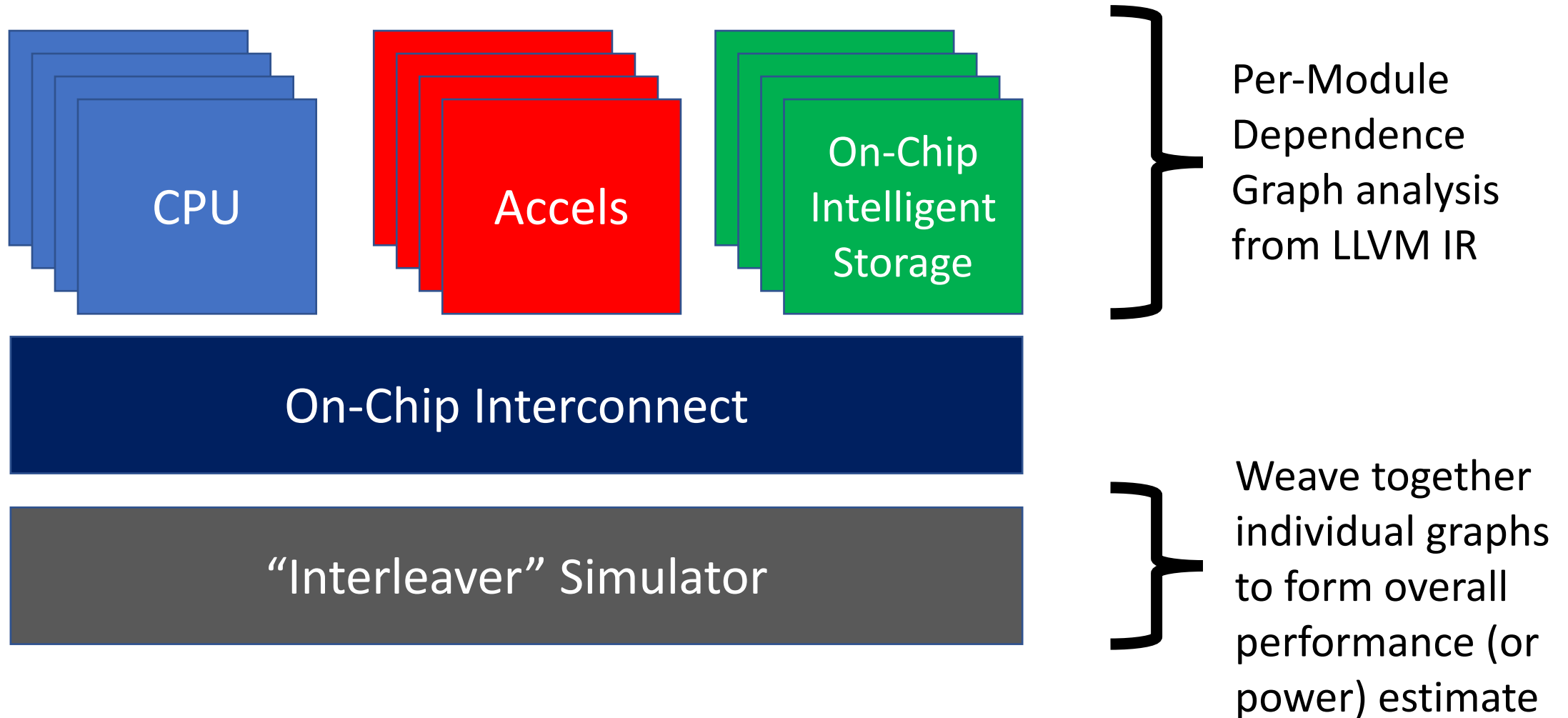
- Build on Embedded Scalable Platforms work [Carlioni, DAC 2016]
- Direct maps to FPGAs

## Chip Prototyping

- Phase 2: Test Chip
- Phase 3: DECADES Prototype

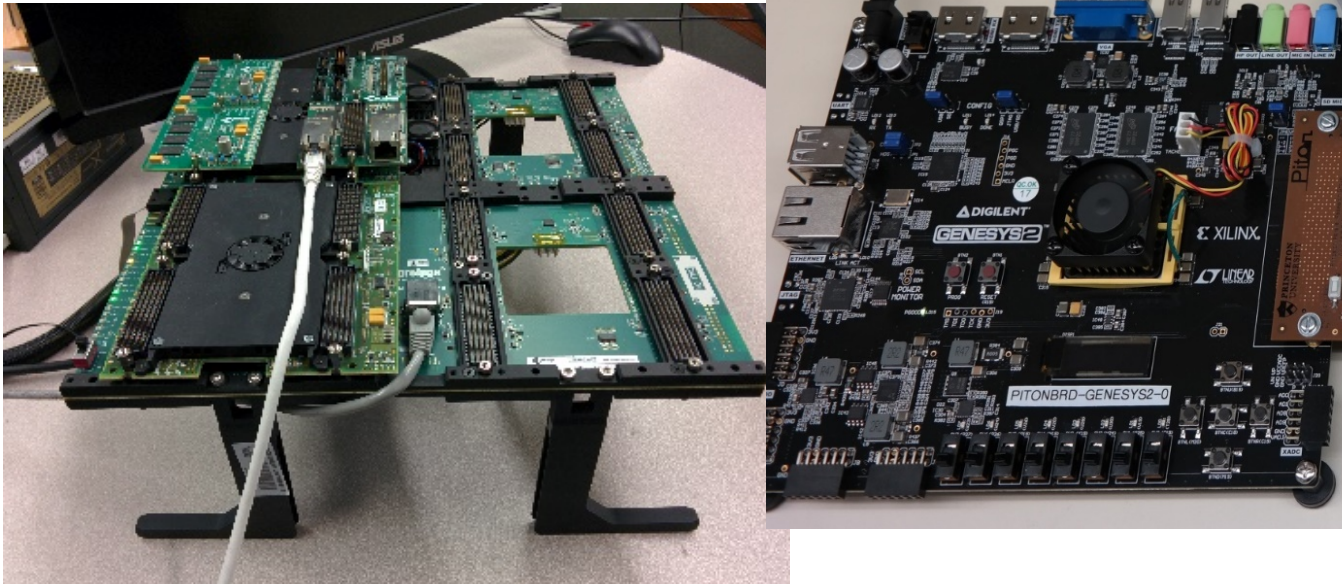


# High-level Simulator Approach



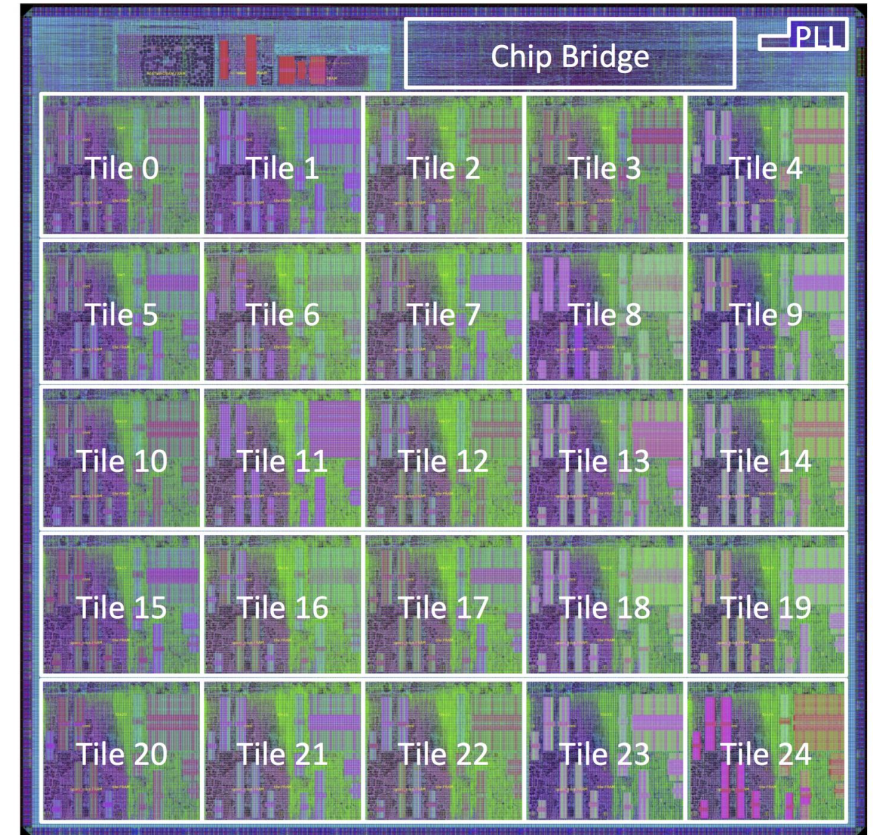
# Emulation & Prototyping

- Take DECADES architecture to FPGA
- Continued design refinement throughout program



- Multi-FPGA emulation infrastructure

- Prototype chip to de-risk architecture



- Recent 25-core manycore system built by our team

# Technology Transfer Plans

- **Outputs:**

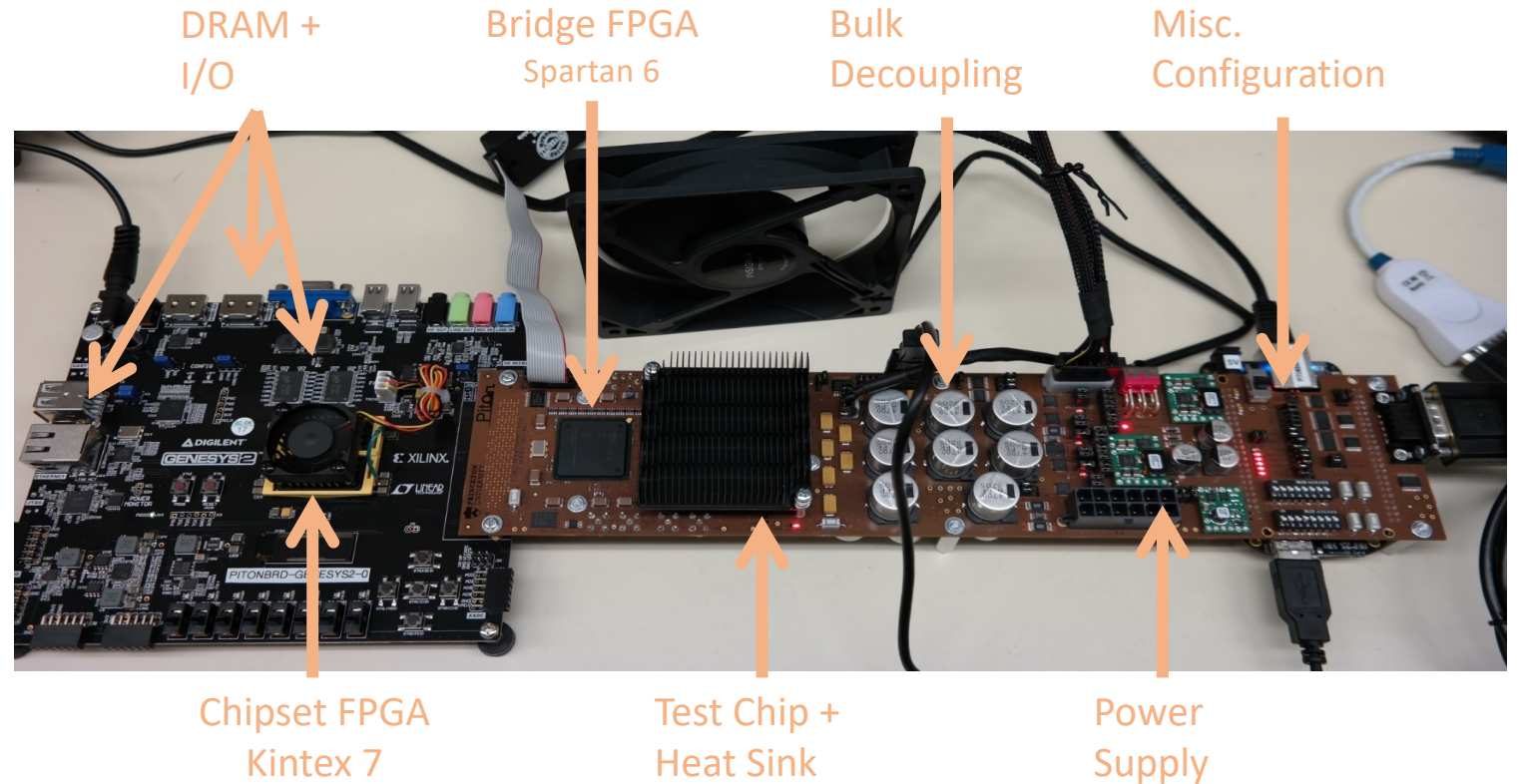
- Software ecosystem
- Chip design
- FPGA emulation system

- **Technology transfer plans:**

- Release of software, hardware, and data where possible
- Commercialization and licensing

- **Leverage extensive past experience:**

- Widely-used open-source software (Wattch, \*Check tools, scalable QEMU)
- Patents licensed to major companies (Power-efficient ALUs)
- Technology transferred from academia to startups (Tilera)
- Open-Source Hardware (OpenPiton)



# Summary & Impact

## Language/Compiler/Runtime:

- Latency: >4X per thread performance benefits from memory data supply decoupling
- Bandwidth: Granularity management and Multiplicative outer-loop parallelism up to bandwidth limit
- Total of 50X over single-thread from software

## Configurable Hardware Platform:




- Hardware speedups from accelerators for address calculation, memory fetch, or compute
- Fine-grained, low-overhead measurements drive adaptation and module depowering
- 10-20X multiplicative power/performance benefits

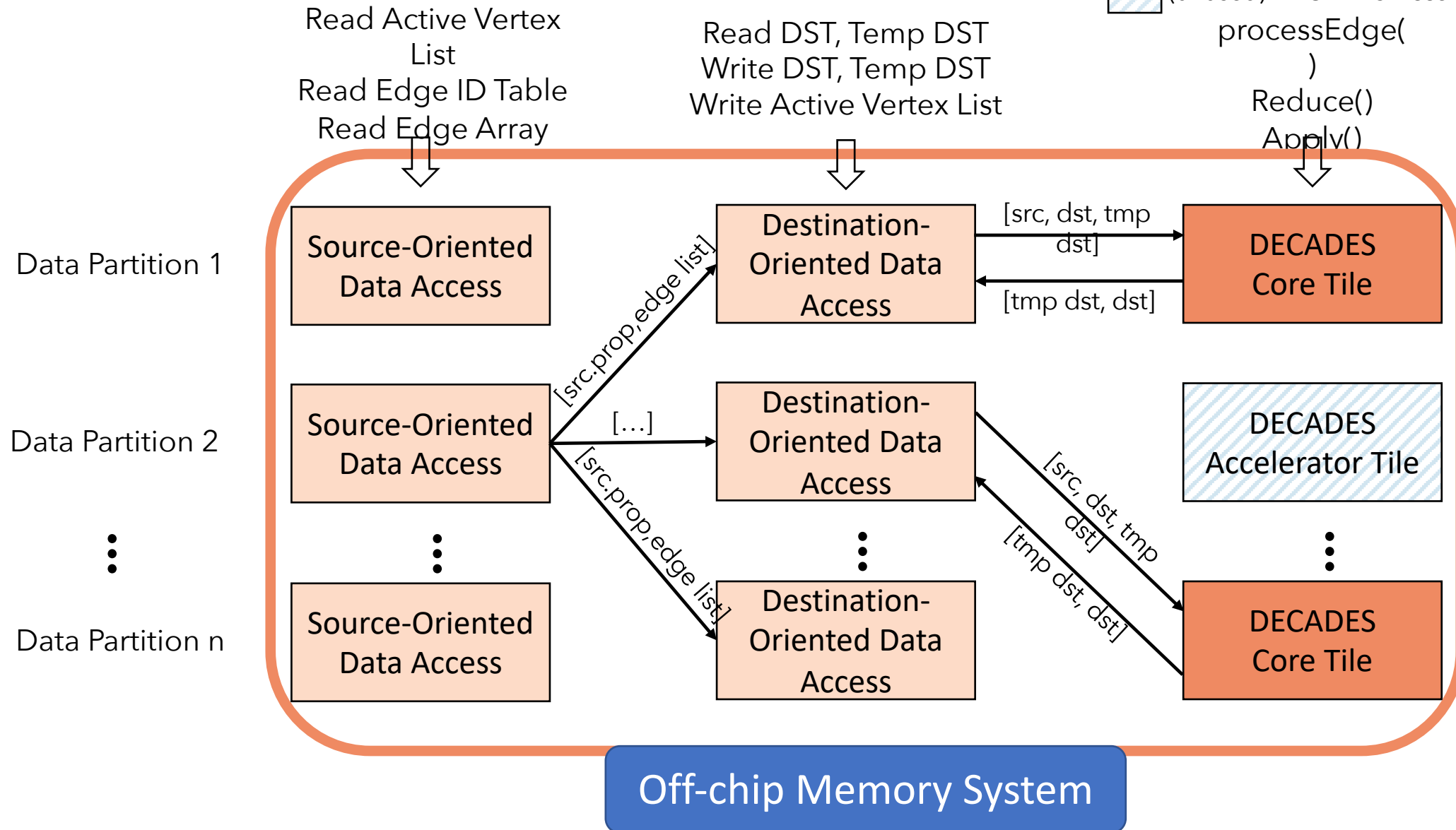


Private Talk  
Starts Here...



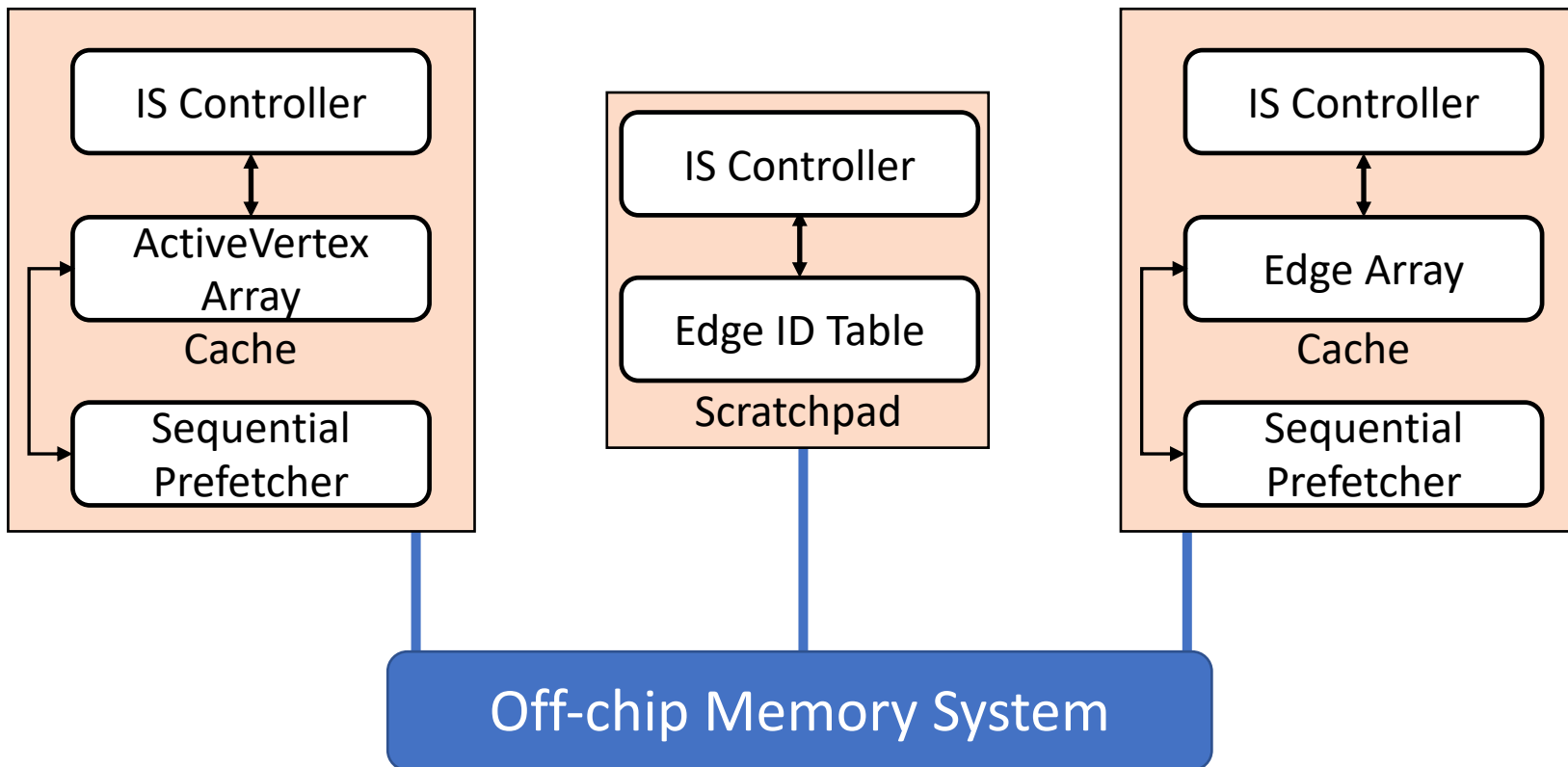
# Mapping Graphicionado to DECADES

-  DECADES Intelligent Storage
-  DECADES Core Tile
-  (unused) DECADES Accelerator Tile



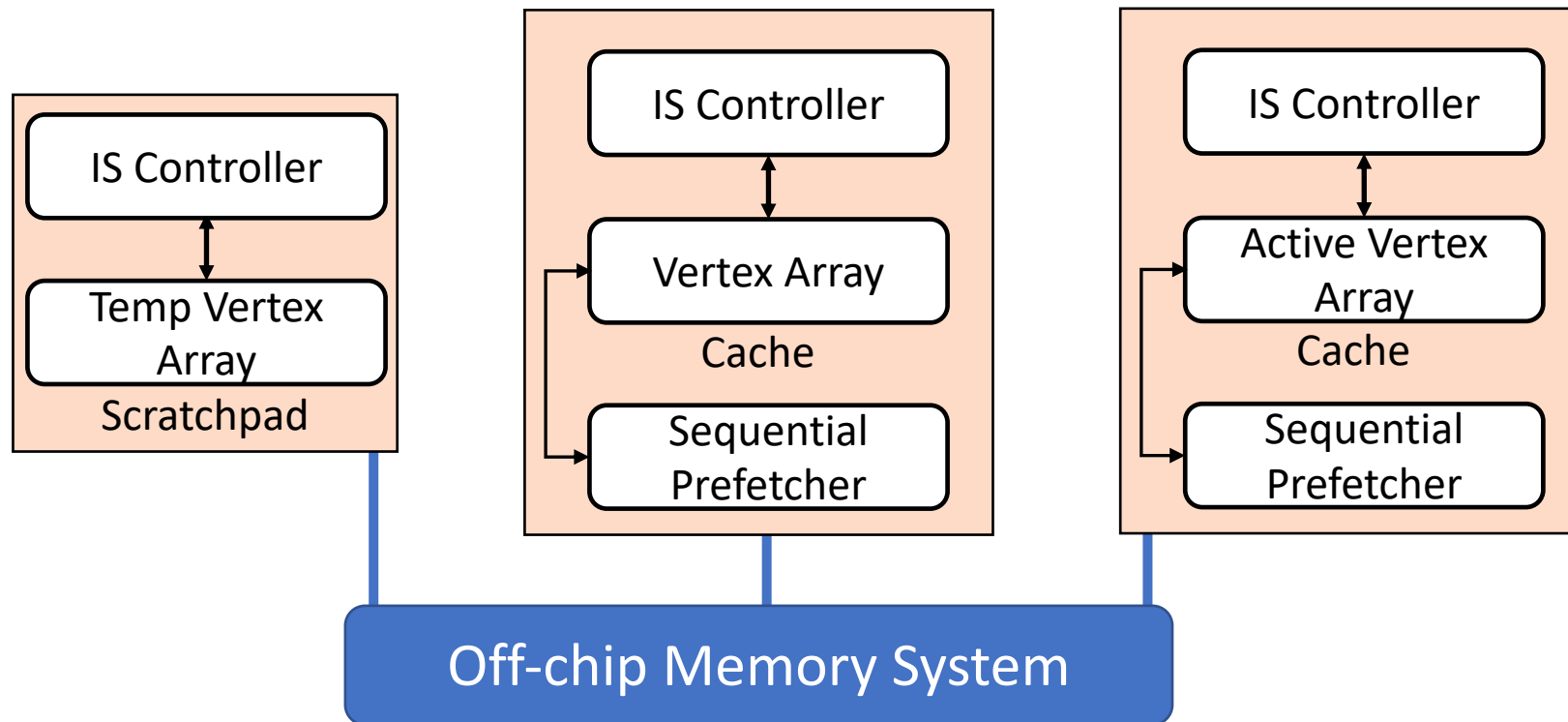
# Source-Oriented Intelligent Storage Tiles

- Intelligent Storage configured with Cache, Scratchpad, and Prefetcher
- Tiles programmed with bulk access directives, destination tile(s)



# Destination-Oriented Intelligent Storage Tiles

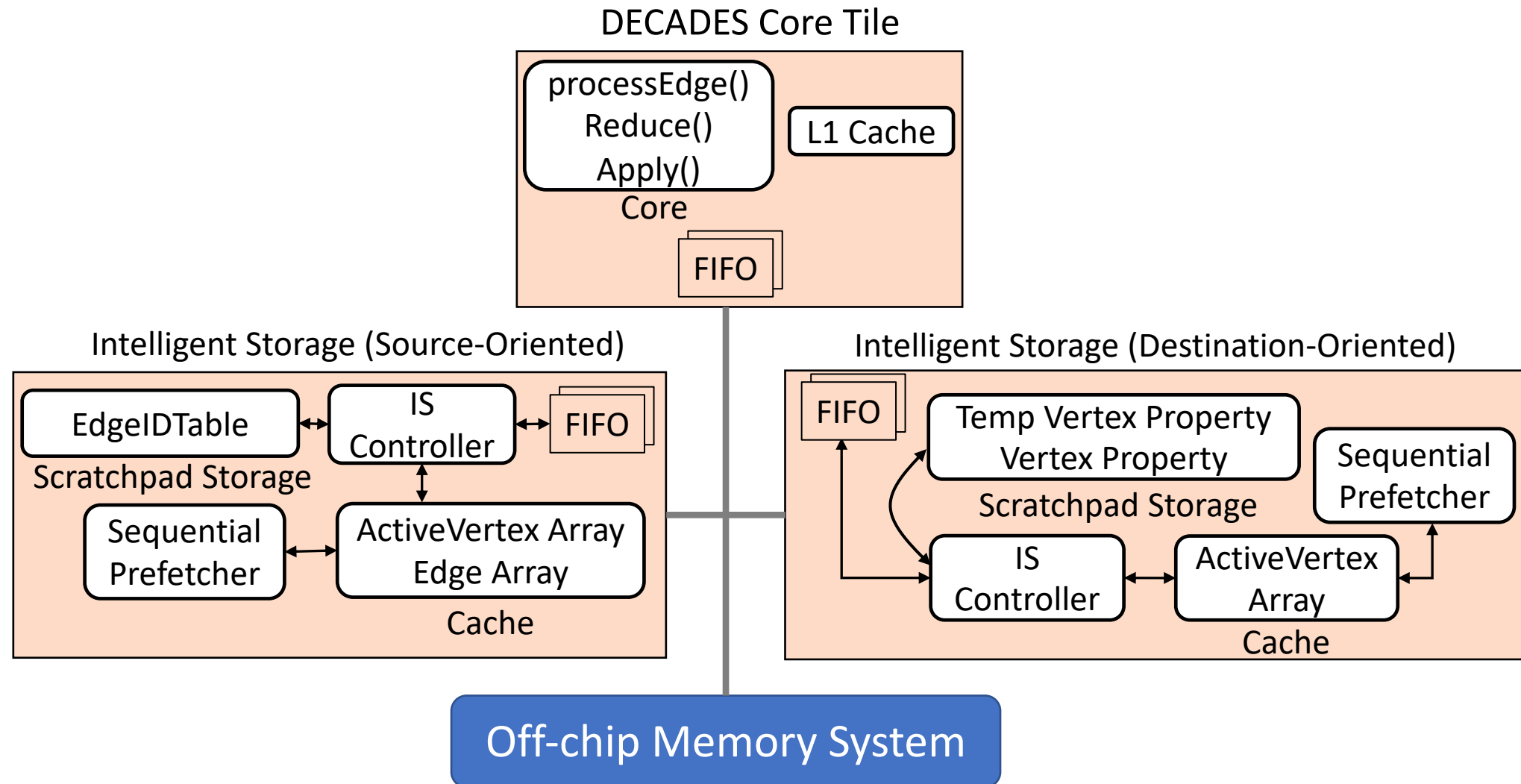
- Intelligent Storage configured with Cache, Scratchpad, and Prefetcher
- Tiles programmed with bulk access directives, destination tile(s)






# Tile Configurations for Graphicionado

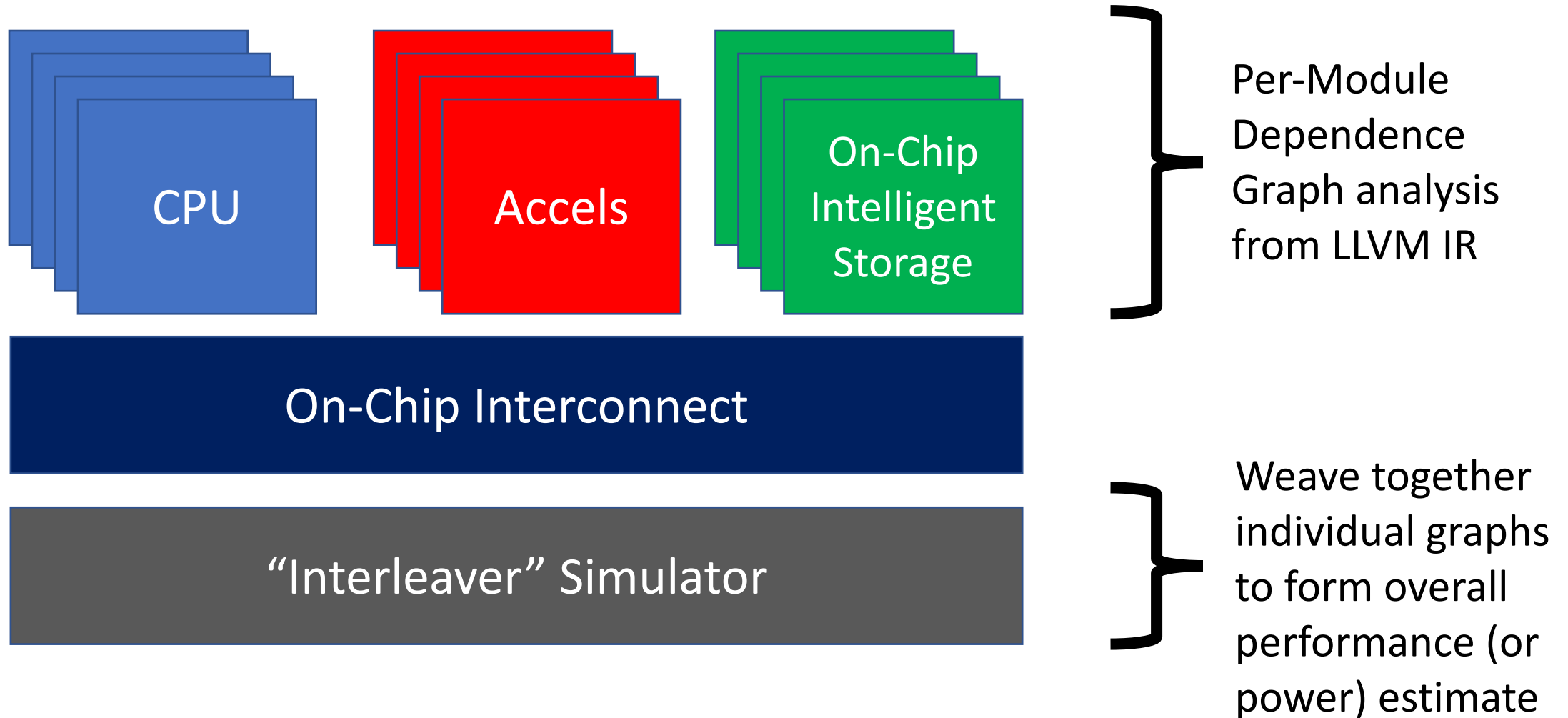
- Intelligent Storage configured with Cache, Scratchpad and Prefetcher



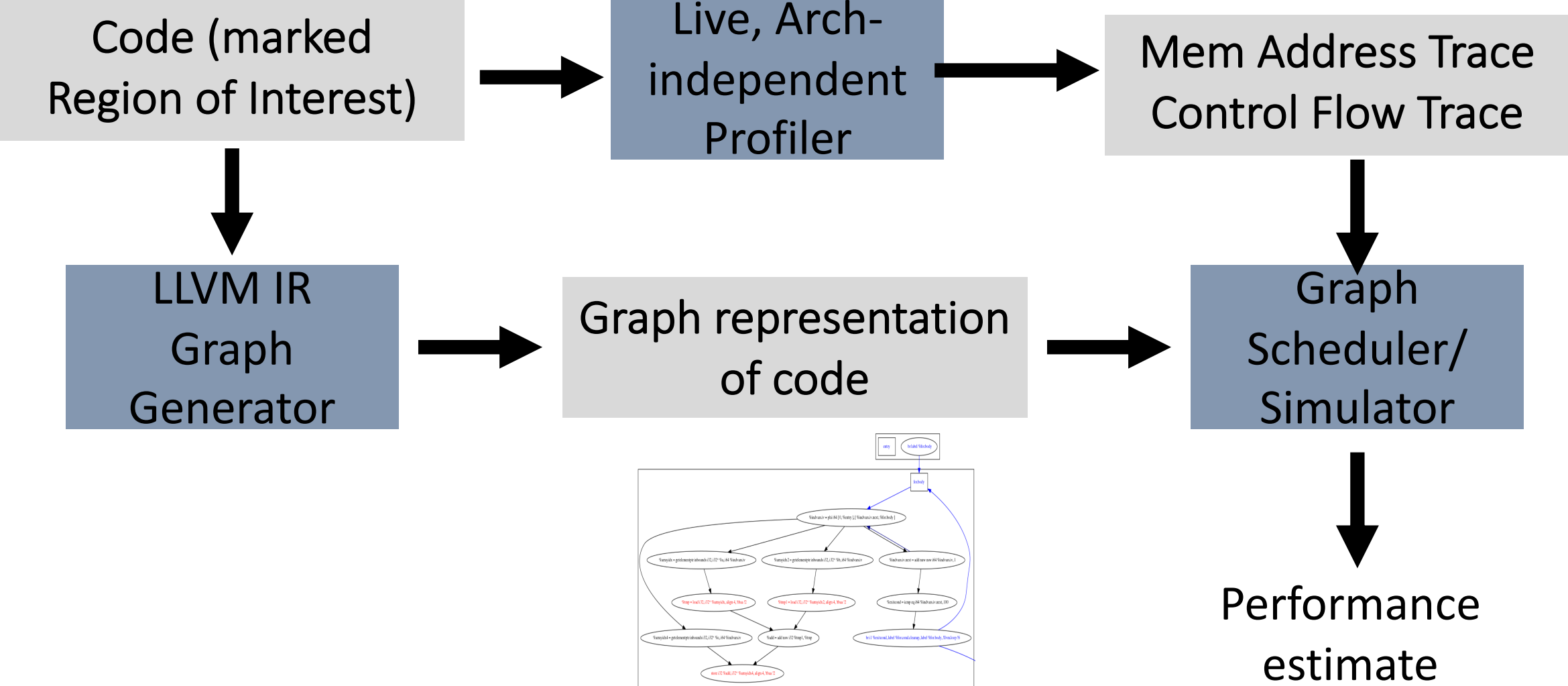
Evaluation  
Systems  
Details...



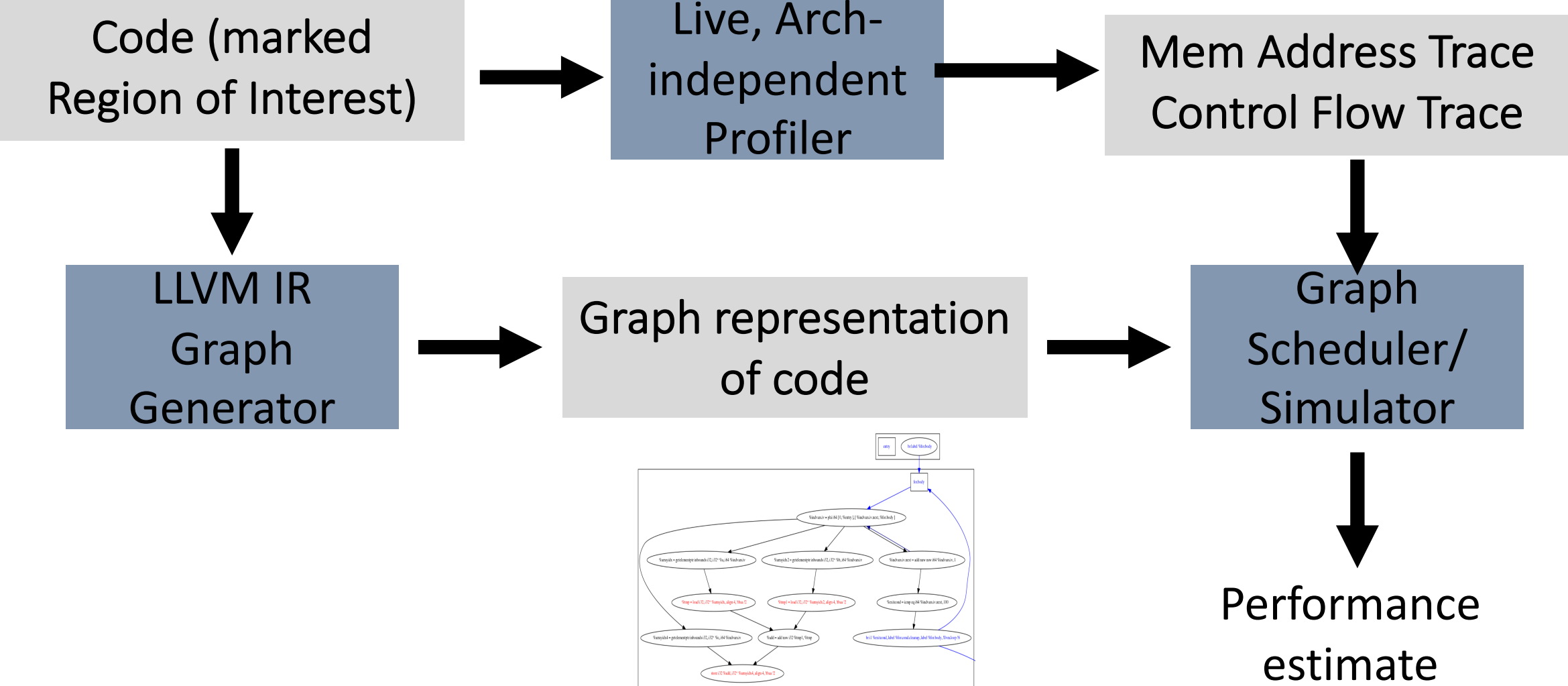
# High-level Simulator Approach



# Per-Module Graph-Based Performance Analysis

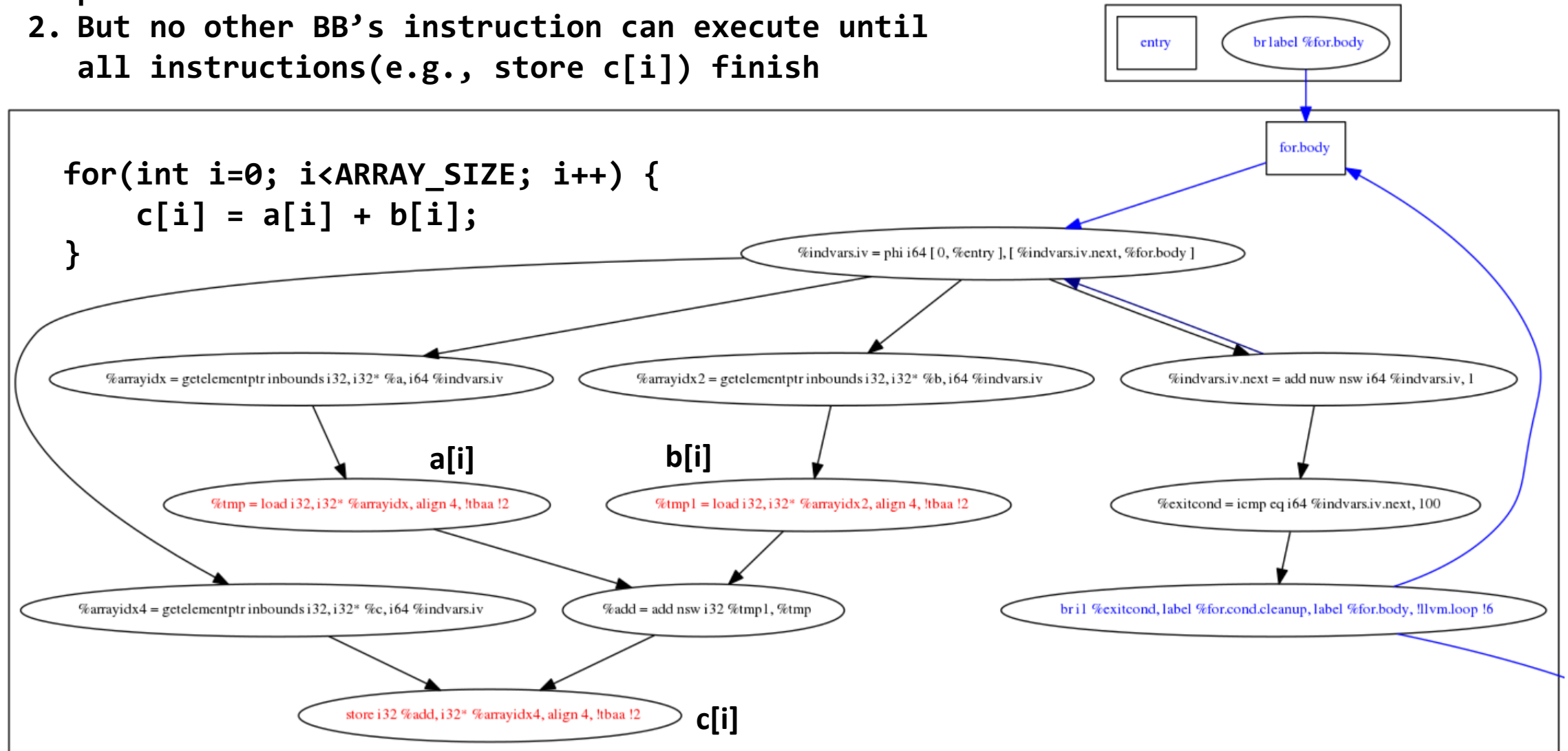


# Per-Module Graph-Based Performance Analysis



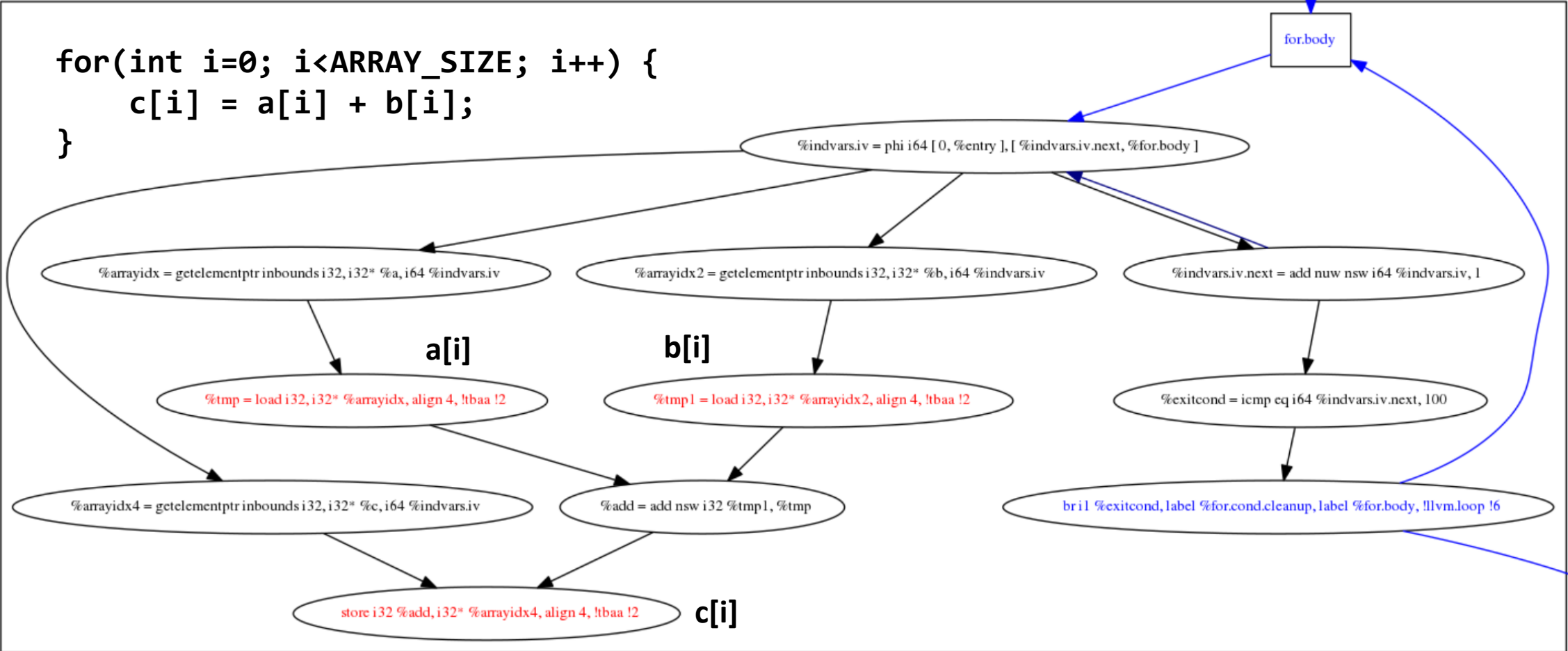
# Model 1: ILP within a basic block only

1. Instructions within a basic block can execute in parallel
2. But no other BB's instruction can execute until all instructions(e.g., store c[i]) finish



# Model 2: Non-speculative Execution

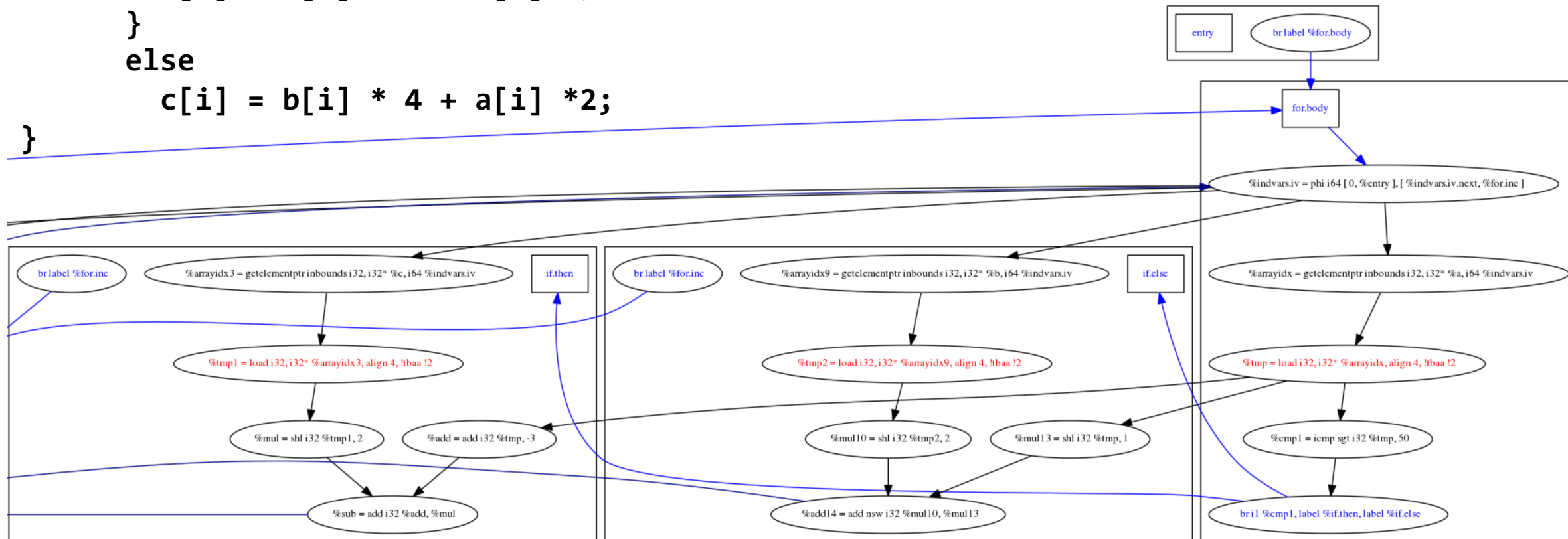
- 1. Instructions within a basic block can execute in parallel
- 2. After branch outcome is obtained (i.e., br), next basic block can immediately start execution
- 3. (Or if the loop is known to be fully parallel)



# Model 3: Speculative Access

1. Instructions within a basic block can execute in parallel
2. Even before the branch outcome is obtained, it speculatively processes the next (potential) basic block as long as there's no data dependency

```
for(int i=0; i<ARRAY_SIZE; i++) {  
    if(a[i]>50) {  
        b[i] = c[i] * 4 + a[i]-3;  
    }  
    else  
        c[i] = b[i] * 4 + a[i] *2;  
}
```





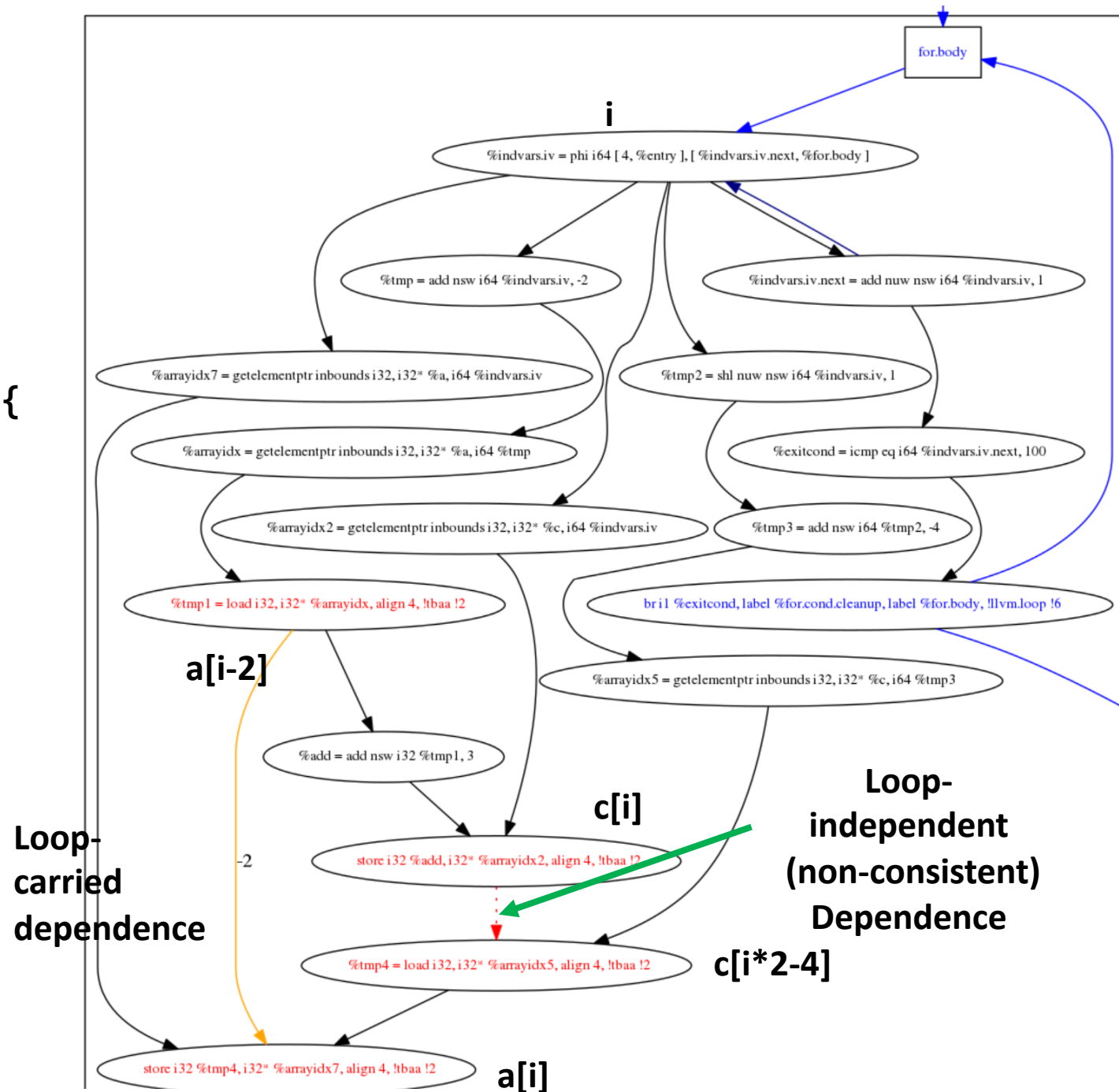
# Model 4: Memory Dependence

```
for(int i=2; i<ARRAY_SIZE; i++) {
    c[i] = a[i-2] + 3;
    a[i] = c[i*2-4];
}
```


01. Do not process load (with potential alias) until previous store is finished

02. Do not process load until previous store address is ready (do not wait until value)

03. Speculatively process load until even under the presence of aliasing store

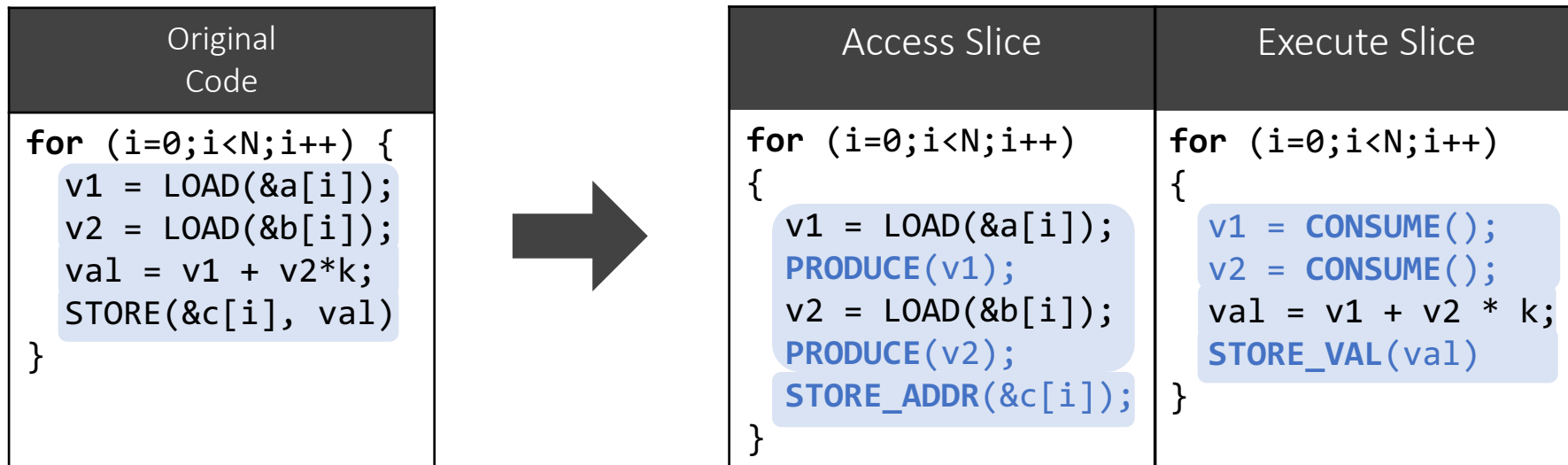


Language,  
Compiler,  
Runtime  
Details



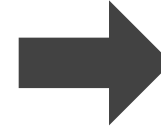
# Decoupling Data Supply from Computation

- Access Slice
  - Performs LOAD and supplies data to Execute Slice with PRODUCE instruction.
  - Computes address for STORE and updates it with STORE\_ADDR.
- Execute Slice
  - Retrieves data sent from Access Slice with CONSUME instruction.
  - Computes value for STORE and updates it with STORE\_VAL.
- Performance: 8X better from memory boundedness. Further multiplicative speedups with N DOALL pairings of Access and Execute.



# Decoupling Data Supply from Computation

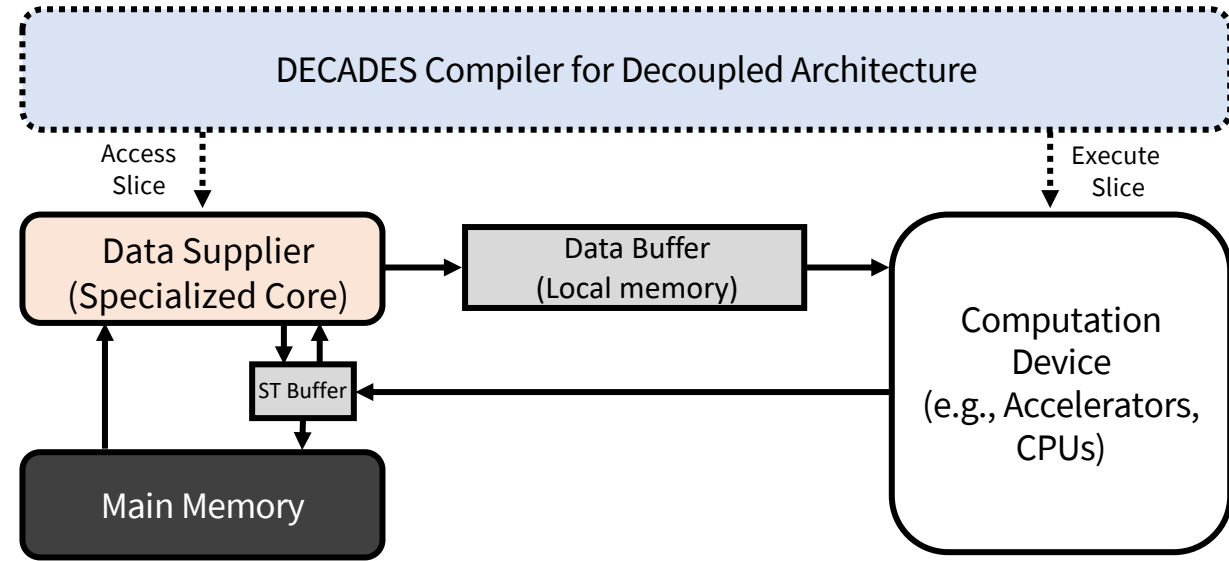
Original Code
<pre>for (i=0;i&lt;N;i++) {     v1 = LOAD(&amp;a[i]);     v2 = LOAD(&amp;b[i]);     val = v1 + v2*k;     STORE(&amp;c[i], val) }</pre>



Access Slice	Execute Slice
<pre>for (i=0;i&lt;N;i++) {     v1 = LOAD(&amp;a[i]);     PRODUCE(v1);     v2 = LOAD(&amp;b[i]);     PRODUCE(v2);     STORE_ADDR(&amp;c[i]); }</pre>	<pre>for (i=0;i&lt;N;i++) {     v1 = CONSUME();     v2 = CONSUME();     val = v1 + v2 * k;     STORE_VAL(val) }</pre>

- DeSC Compiler slices a program into two parts: **access slice** and **execute slice**
  - Inspired by seminal work: [James Smith, Decoupled Access/Execute Architecture \(DAE\), ISCA'82](#)
- **Access Slice**
  - Performs **LOAD** and supplies data to Execute Slice with **PRODUCE** instruction.
  - Computes **address for STORE** and updates it with **STORE\_ADDR**.
- **Execute Slice**
  - Retrieves data sent from Access Slice with **CONSUME** instruction.
  - Computes **value for STORE** and updates it with **STORE\_VAL**.
- Access Slice can **run ahead** of Execute Slice

# ISA Additions for Memory Decoupling



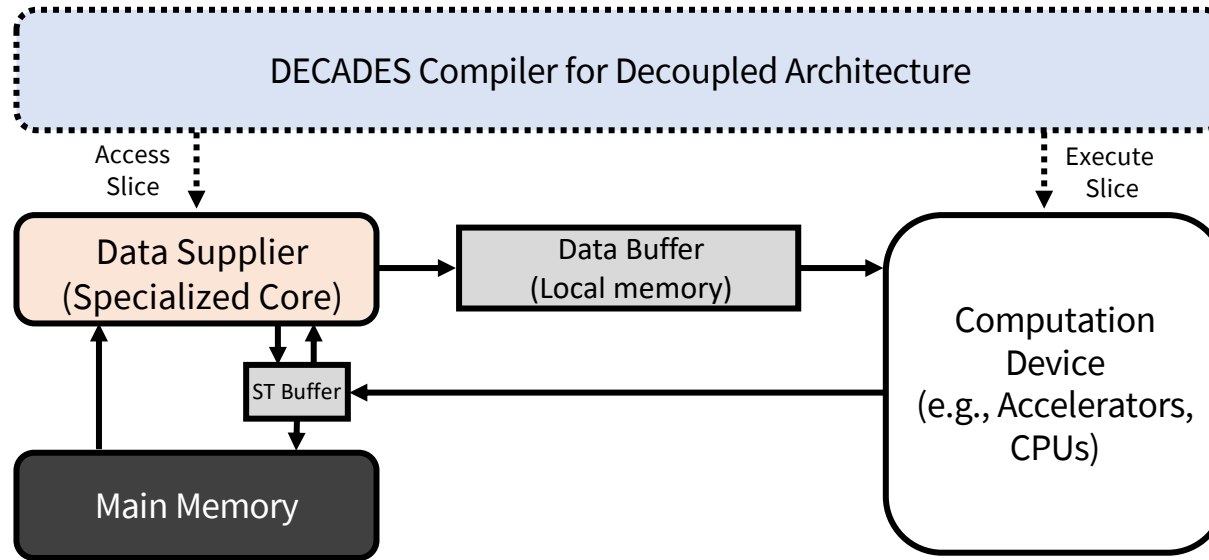
## Data Supplier Special Instructions

- **PRODUCE**  
Inserts data to the Data Buffer
- **STORE\_ADDR**  
Updates Store Address in ST Buffer

## Computation Device Special Instructions

- **CONSUME**  
Retrieves data from the Data Buffer
- **STORE\_VAL**  
Updates Store Value in ST Buffer

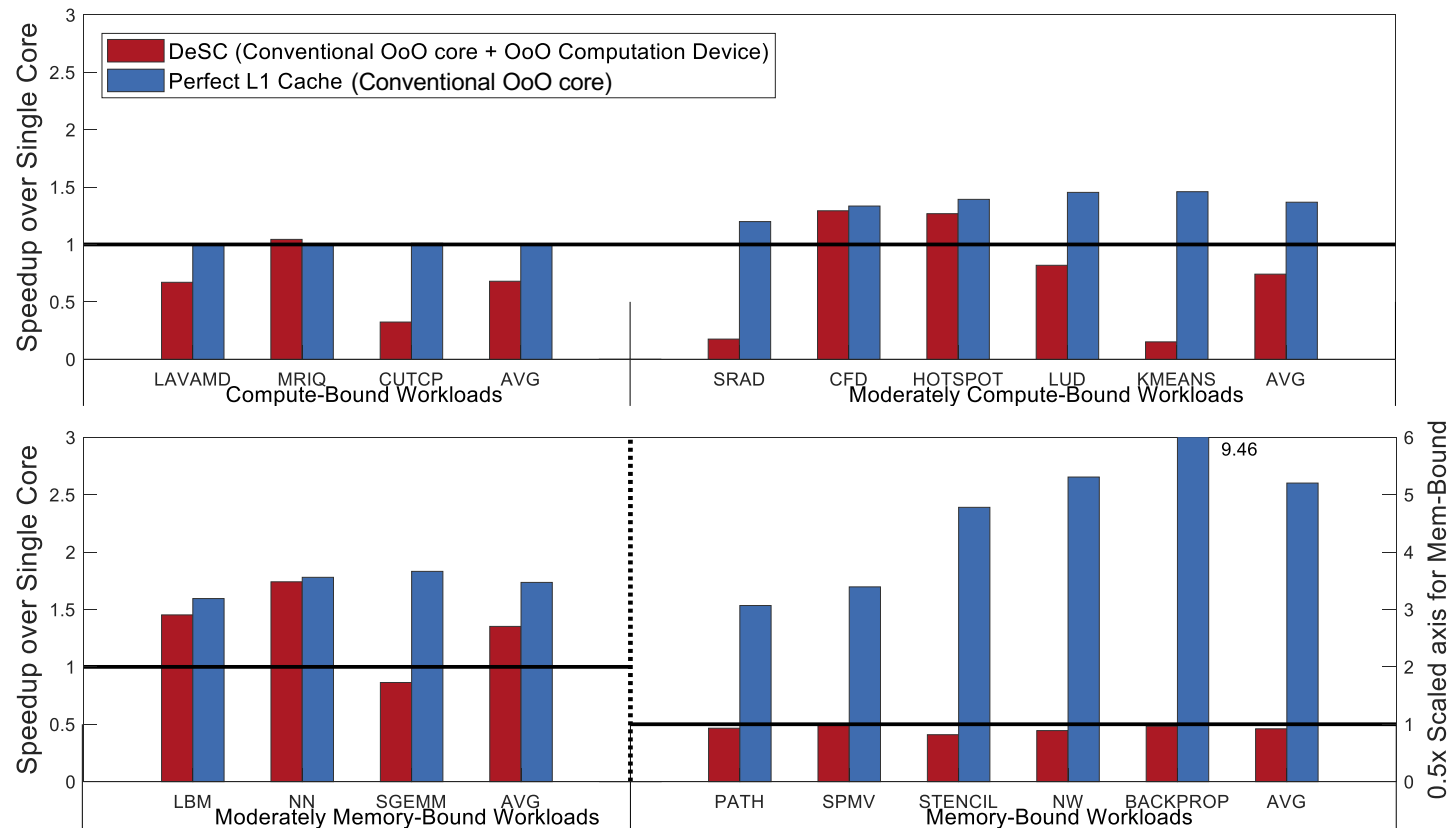
# Decoupled Execution: Key Ideas



1. DECADES compiler automatically generates code for decoupled Data Supply and thus does not require programmer input for communication management
2. DECADES specializes general-purpose core for data supply task for higher performance
3. DECADES Data Supplier hardware can be used to supply data for different types of computation devices

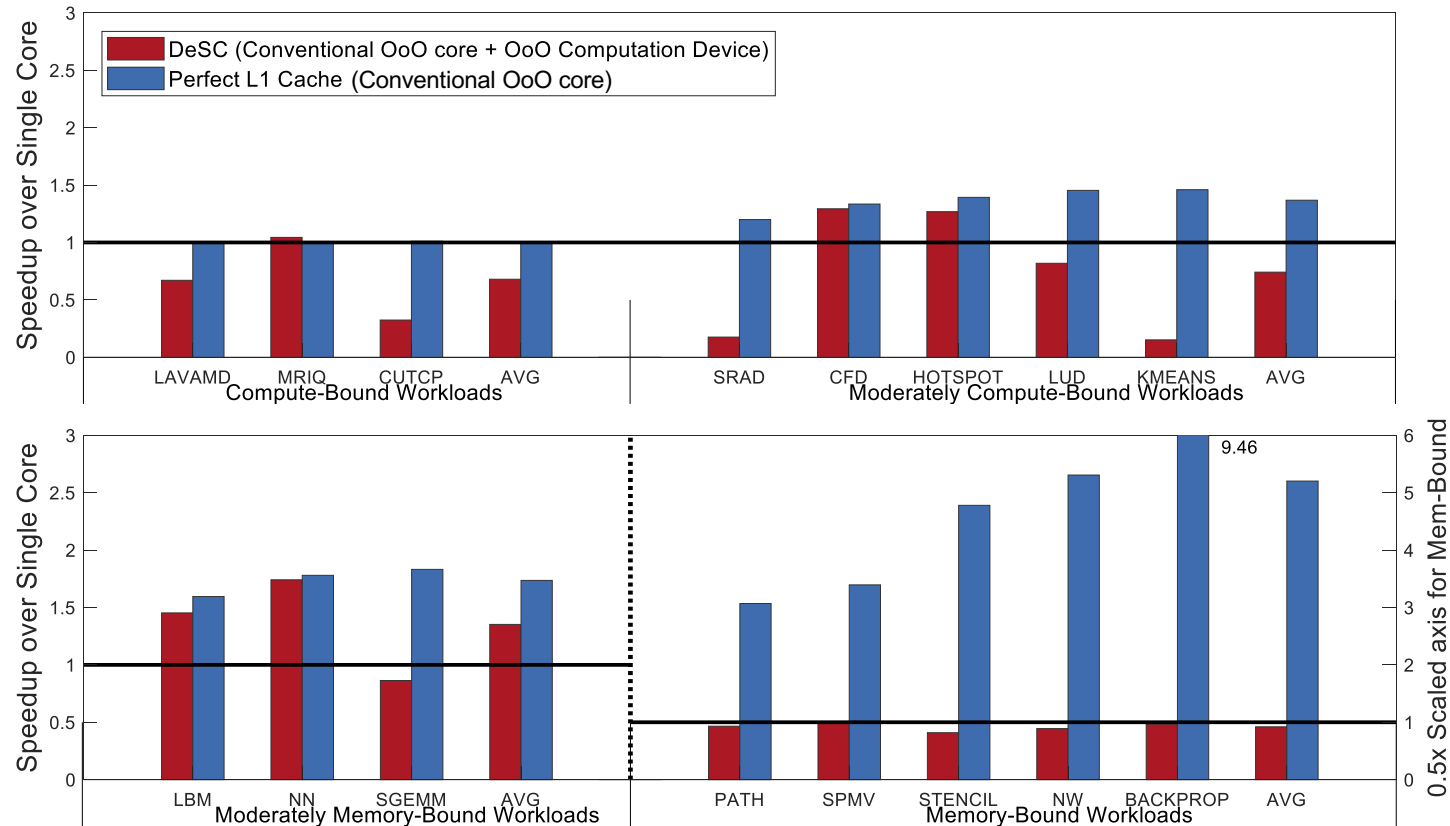
# Why Decoupled Optimizations?

- Use of conventional OoO core as a data supplier in DAE **often fails to improve performance**
- Performance is **often worse** than for a single core
- **Significantly worse** than **perfect latency tolerance** (single core w/ perfect L1 cache)



# Why Decoupled Optimizations?

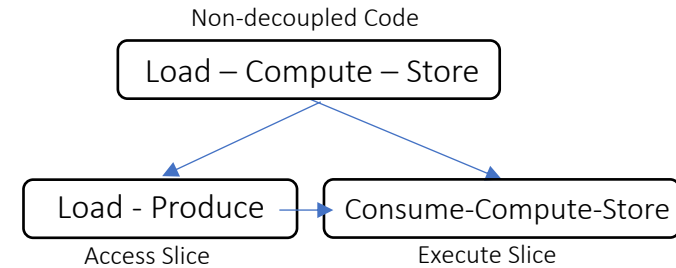
- What are main problems?
  1. **Inefficiency in OoO core:** Later instructions cannot commit if long latency load is blocking the head of the ROB
  2. **Loss of Decoupling Events:** Data Supplier depends on computation device and has to stall





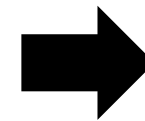
# Opportunities in Decoupled Access Slice

- **Opportunity:** Most of LOADs in a Decoupled Access Slice have a single PRODUCE instruction as its only dependent
- **Conventional Architecture:** Results of LOAD instructions are used for computation
- **Decoupled Architecture:** Results of most LOAD instructions are only used in the Execute Slice
  - No other dependent except for the immediate PRODUCE
- **Terminal Load:** Loads whose fetched value is only used for the following PRODUCE
  - Compiler converts such loads to LOAD\_PRODUCE which has no dependent



```
Example Access Slice
for (i=0;i<N;i++)
{
  idx = LOAD(&a[i]);
  tmp = LOAD(&b[idx]);
  PRODUCE(tmp);
}
```

Code before marking Terminal Loads



```
Example Access Slice
for (i=0;i<N;i++)
{
  idx = LOAD(&a[i]);
  LOAD_PRODUCE(&b[idx]);
}
```

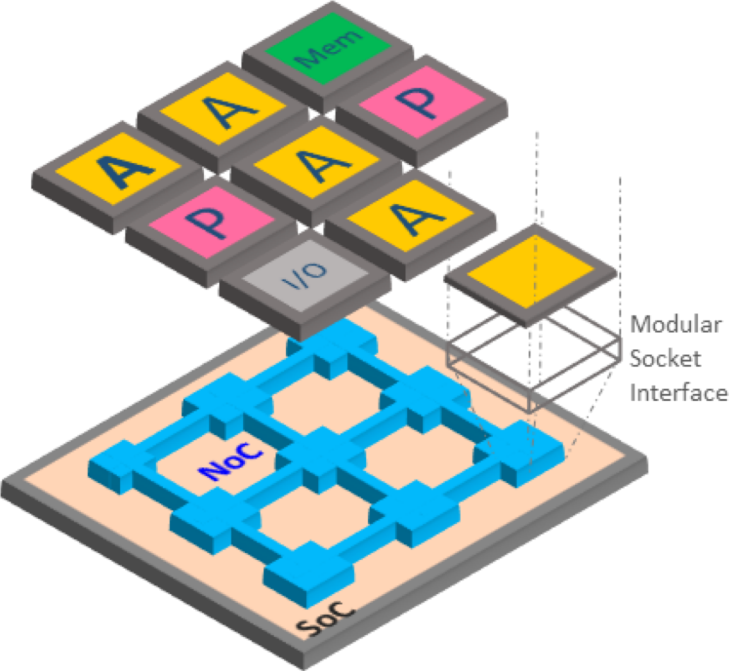
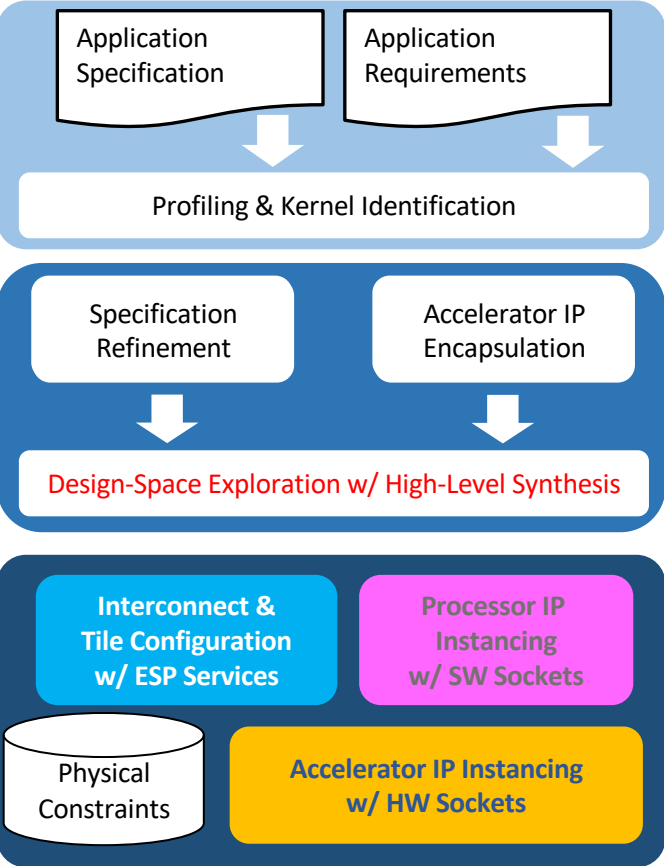
Code after marking Terminal Loads

Prior Work:  
ESP, Carloni

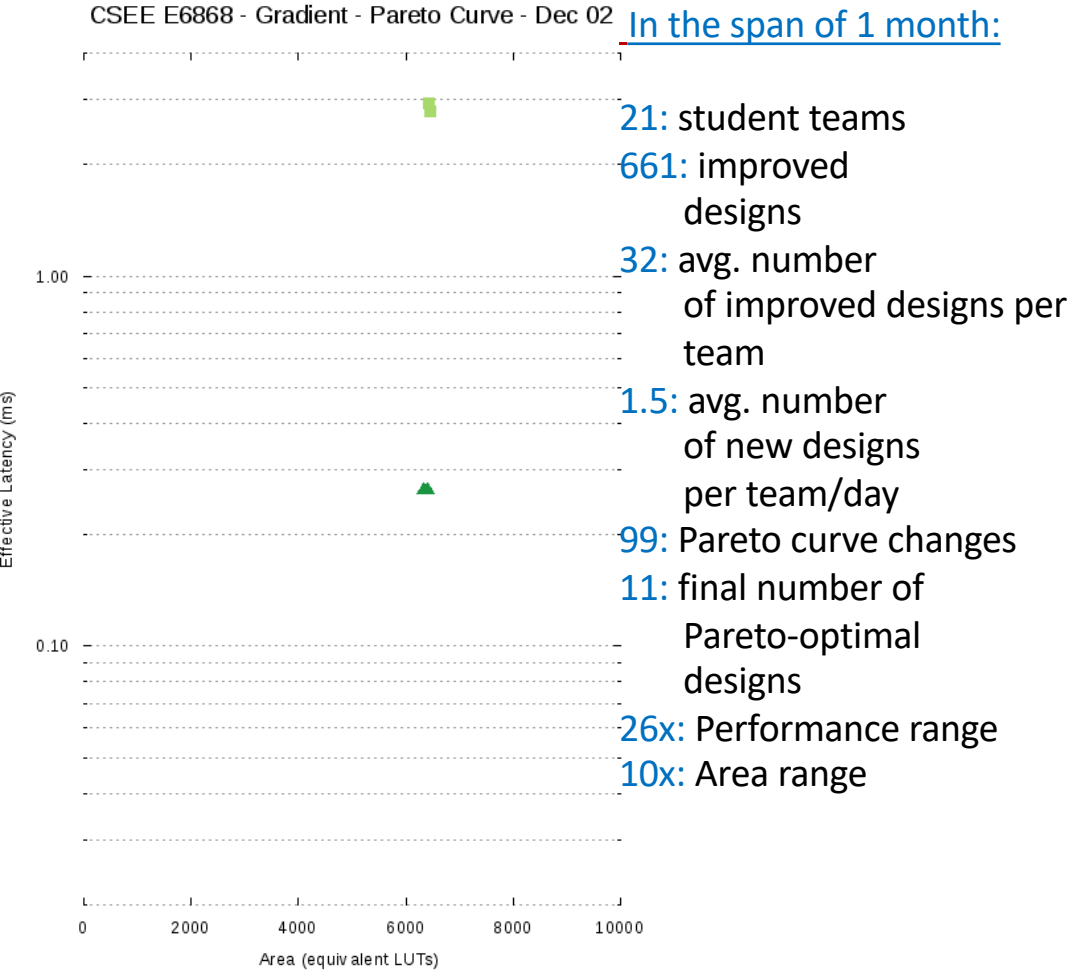


# PRIOR WORK: EMBEDDED SCALABLE PLATFORMS

- Flexible Tile-Based Architecture
- System-Level Design Methodology



- SoC Design Productivity



[Carloni, DAC 2016]

# Coherence Models for Loosely-Coupled Accelerators

- **Most accelerators embrace the shared-memory paradigm...**
- **...but implement different levels of cache coherence. We classified them as:**

## 1. **Non-Coherent DMA**

- Data must be flushed to memory
- Large data sets are accessed faster and without polluting caches

## 2. **Fully-Coherent Load/Store**

- A private cache is required to handle coherence transparently
- Preferred for frequent interleaving of processor and accelerator execution

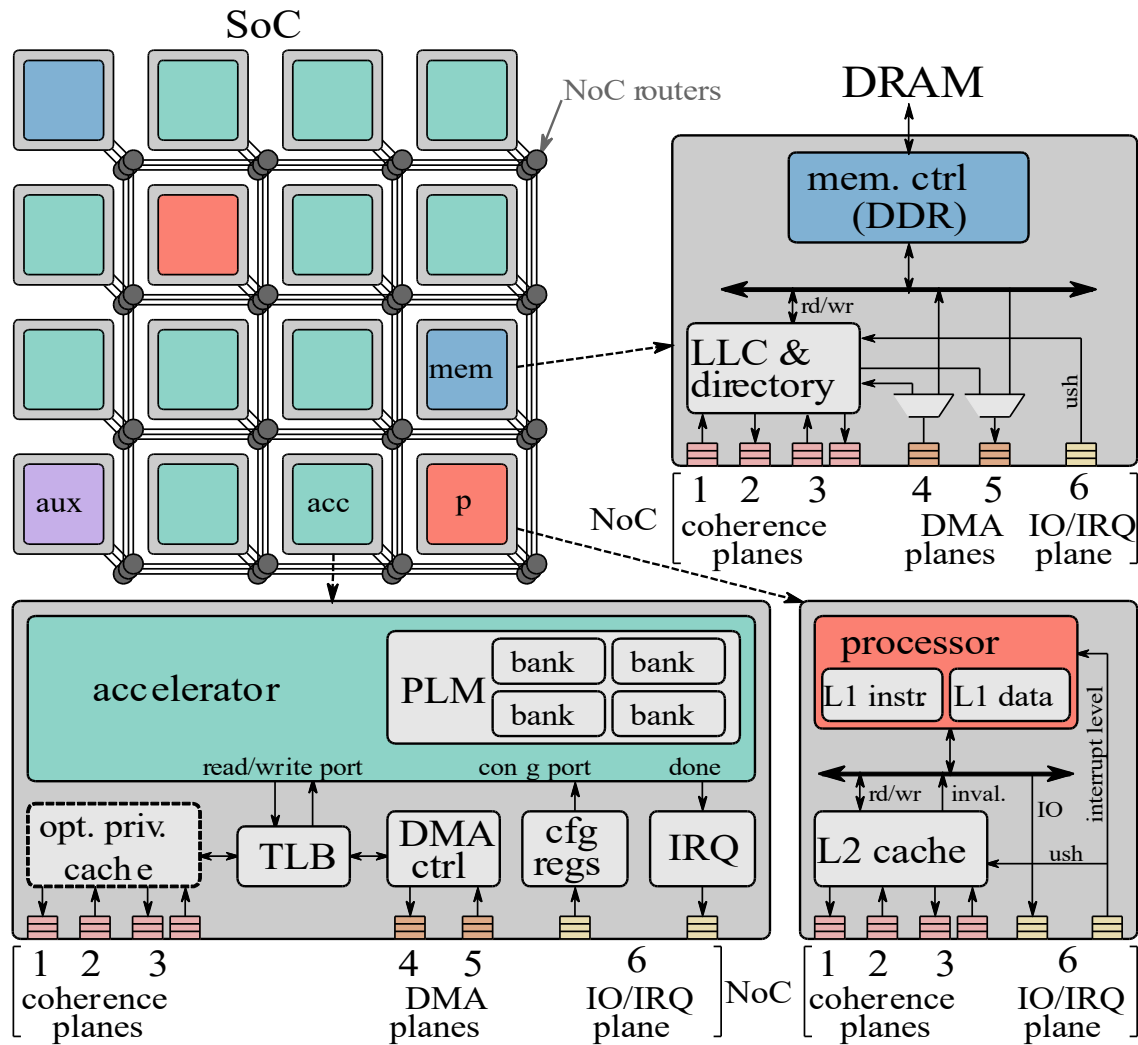
## 3. **Last Level Cache (LLC)-Coherent DMA (new)**

- Data are flushed to LLC, thus reducing accesses to external memory
- Medium-sized data sets are accessed faster

## 4. **Coherent DMA (new on NoC)**

- No flush required thanks to recalls handled by the LLC on an NoC, or snooping on a bus
- Almost as fast as LLC-coherent DMA
- Better than LLC-coherent DMA for frequent interleaving of processor and accelerator execution

# Reconfigurable Coherence for Accelerators in ESP



- **First NoC-based system enabling the four models of coherence for accelerators to coexist and operate simultaneously through run-time selection in the same SoC**
  - **Design based on ESP Platform Services**
- **Extension of the MESI directory-based protocol to integrate LLC-coherent accelerators into an SoC**
  - **The design leverages the tile-based architecture of ESP over a packet-switched NoC to guarantee scalability and modularity**

# Heterogeneous Coherence Implementation

Check and Update SoC Configuration

Accelerator sort <input type="checkbox"/> Cache	Accelerator spmv <input type="checkbox"/> Cache	Accelerator fft2d <input type="checkbox"/> Cache	Accelerator fft1d <input type="checkbox"/> Cache
Accelerator fft1d <input type="checkbox"/> Cache	Memory & Debug <input type="checkbox"/> Cache	Processor <input checked="" type="checkbox"/> Cache	Accelerator sort <input type="checkbox"/> Cache
Accelerator fft2d <input type="checkbox"/> Cache	Processor <input checked="" type="checkbox"/> Cache	Memory <input type="checkbox"/> Cache	Accelerator spmv <input checked="" type="checkbox"/> Cache
Accelerator spmv <input type="checkbox"/> Cache	Accelerator sort <input type="checkbox"/> Cache	Accelerator fft1d <input checked="" type="checkbox"/> Cache	Accelerator fft2d <input type="checkbox"/> Cache

- **The CAD Infrastructure of ESP:**
  - direct instantiation of heterogeneous configurable components from predesigned libraries
  - Fully automated flow from the GUI to bitstream for FPGAs
- **Support for atomic test-and-set and compare-and-swap operations over the NoC enable:**
  - multi-processor and multi-accelerator applications on top of Linux SMP

# Heterogeneous Coherence: Experimental Setup

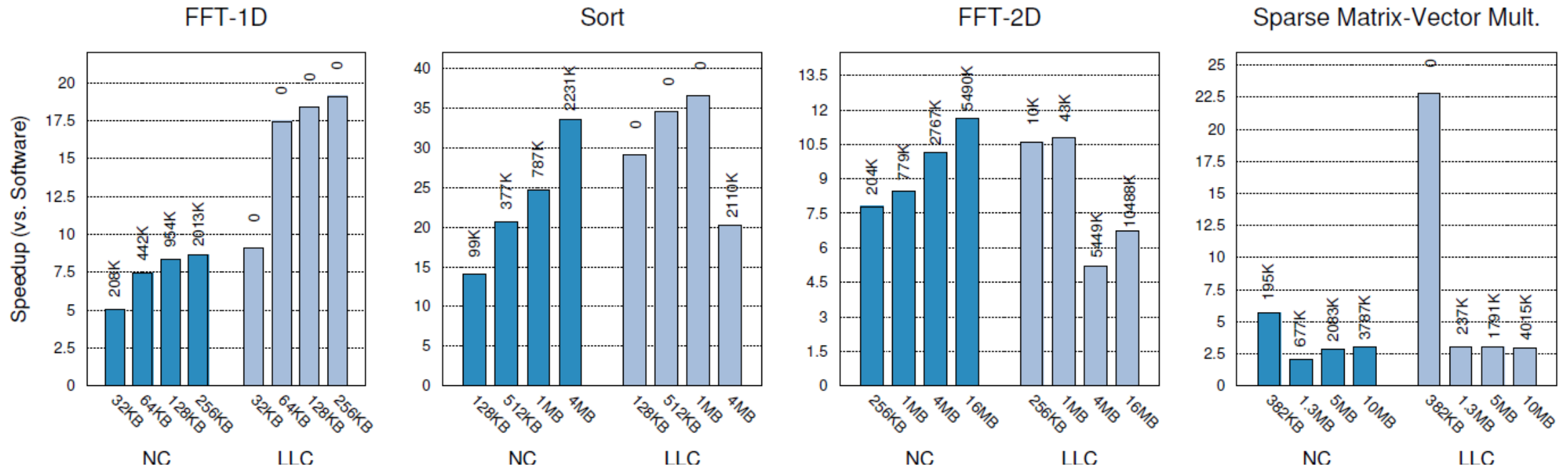
CHARACTERIZATION OF THE TARGET ACCELERATORS.

Accelerator	Memory Footprint	PLM (kB)	FPGA Resources		
			LUT	FF	BRAM
FFT 1D	32kB - 256kB	40	7,537	4,310	10
Sort	128kB - 4MB	24	36,868	31,300	6
FFT 2D	256kB - 16MB	128	3,965	2,190	48
SPMV	25kB - 10MB	12	8,136	4,476	24

*“The ability to have perfectly balanced accelerator stages is highly dependent on the specific memory access patterns, as well as on the system interconnect and the memory hierarchy, including the selected cache-coherence model”*

- **FFT1D**
  - streaming memory access
- **Sort**
  - no temporal locality, but in-place (i.e. in the PLM) data processing
- **FFT2D**
  - Streaming read accesses and single-word write accesses.
- **SPMV**
  - asymmetric data reuse with irregular access pattern
  - low compute-to-memory ratio

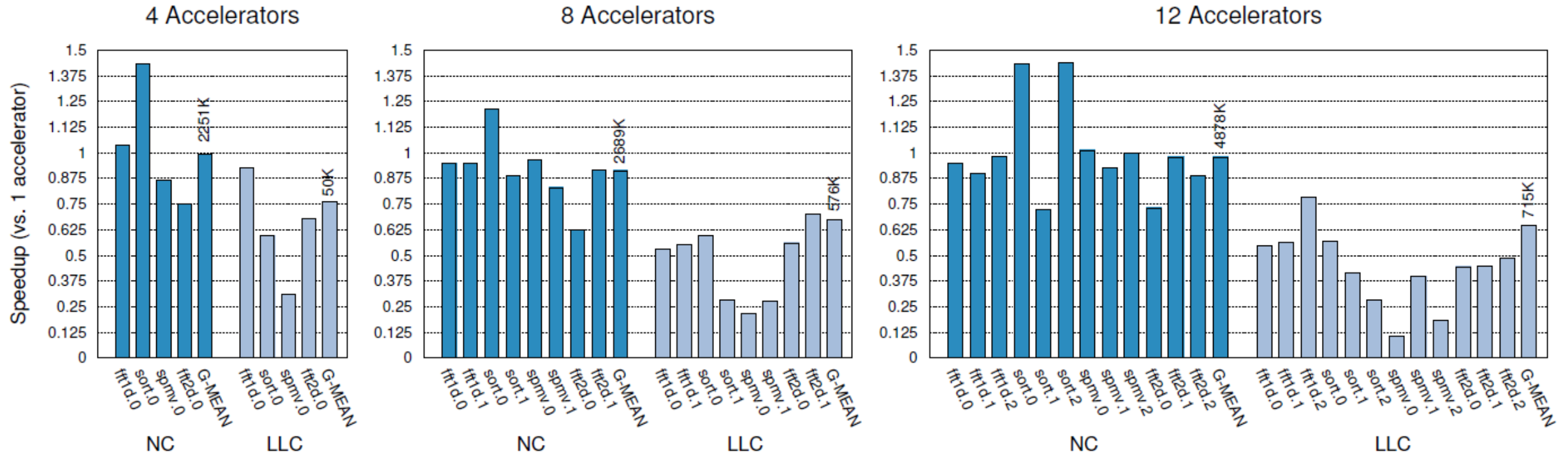
# Non-Coherent vs. LLC-Coherent (single accelerator)



- Compared to non-coherent accelerators, the relative speedup of LLC-coherent accelerators ranges between 0.5x and 4x
  - the memory access count, instead, ranges from 0 to at most 2x (in worst-case scenario)
- Confirmation of the benefits of runtime model selection based on footprint

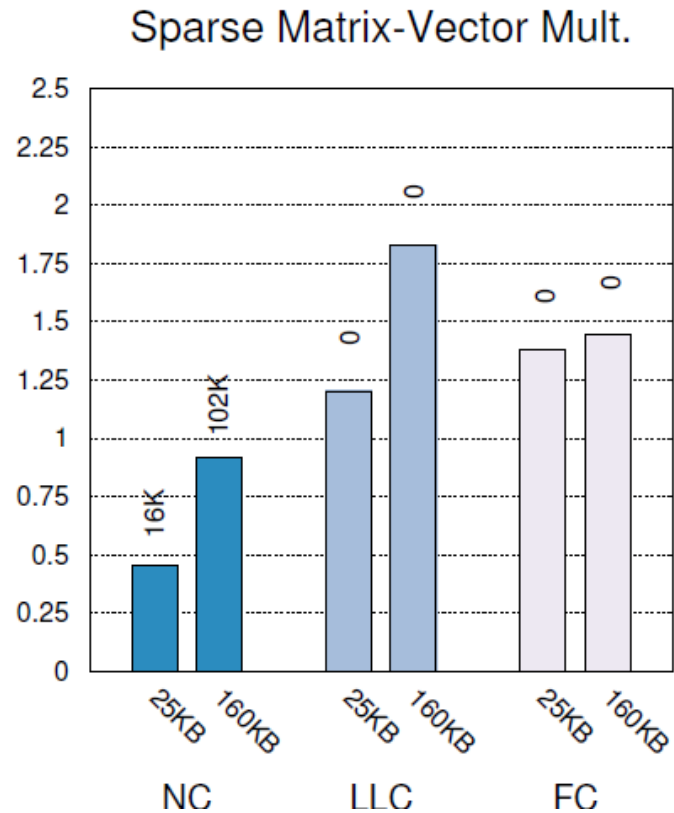
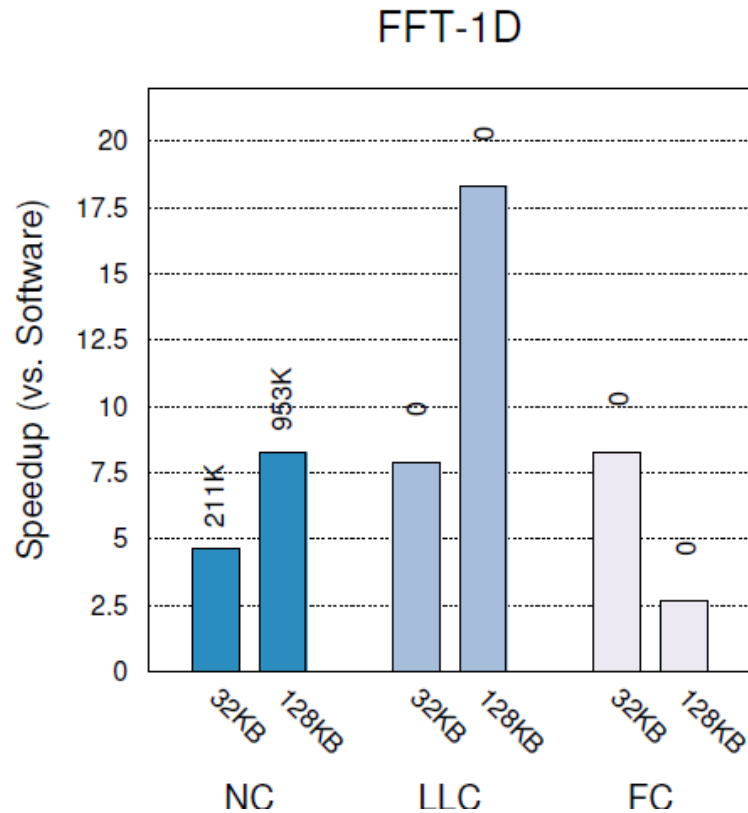


# Non-Coherent vs. LLC-Coherent (multiple accelerators)



- The average performance degraded by up to 38% and 10% for LLC-coherent and non-coherent accelerators, respectively
- Confirmation that the choice of the cache-coherence model should be based on the ratio between the size of the aggregate memory footprint of all running accelerators and the capacity of the LLC

# Cache-Coherence Models for Tiny Workloads

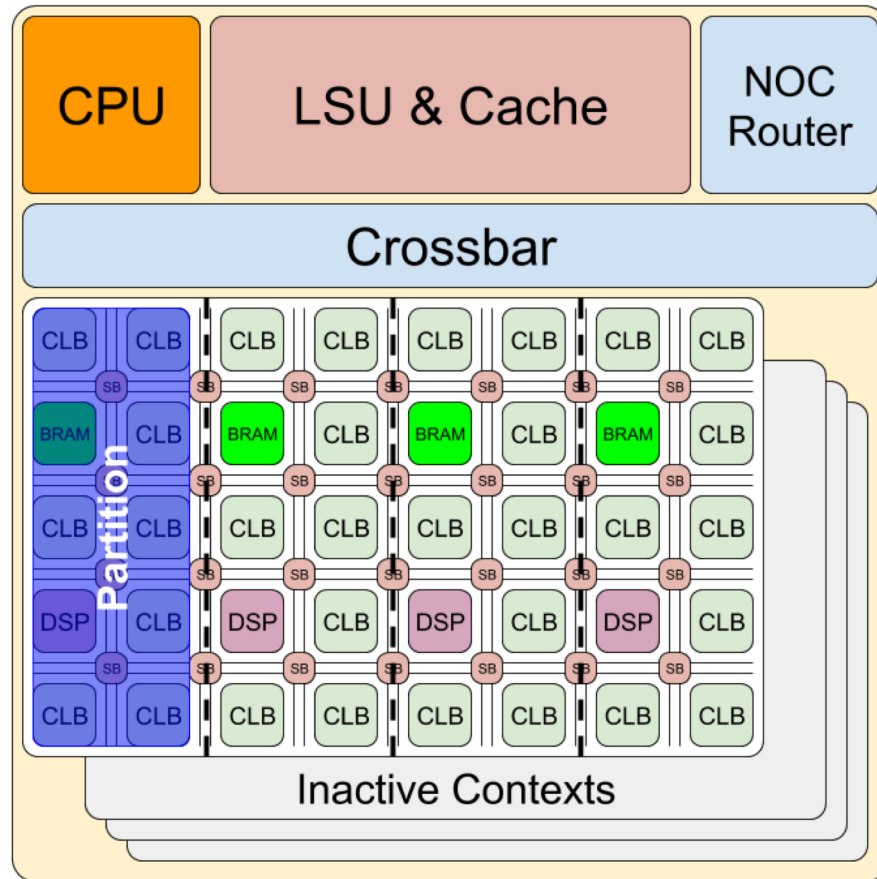


- **The fully-coherent model can have similar or better performance than the other two models, but only for the smallest datasets**
  - like the LLC-coherent model it also reduces or removes memory accesses
  - additionally, it does not require the flushing of the processors' caches, which could disrupt the work of other components of the SOC

Fast-  
Configurable  
LUT-based  
Tiles

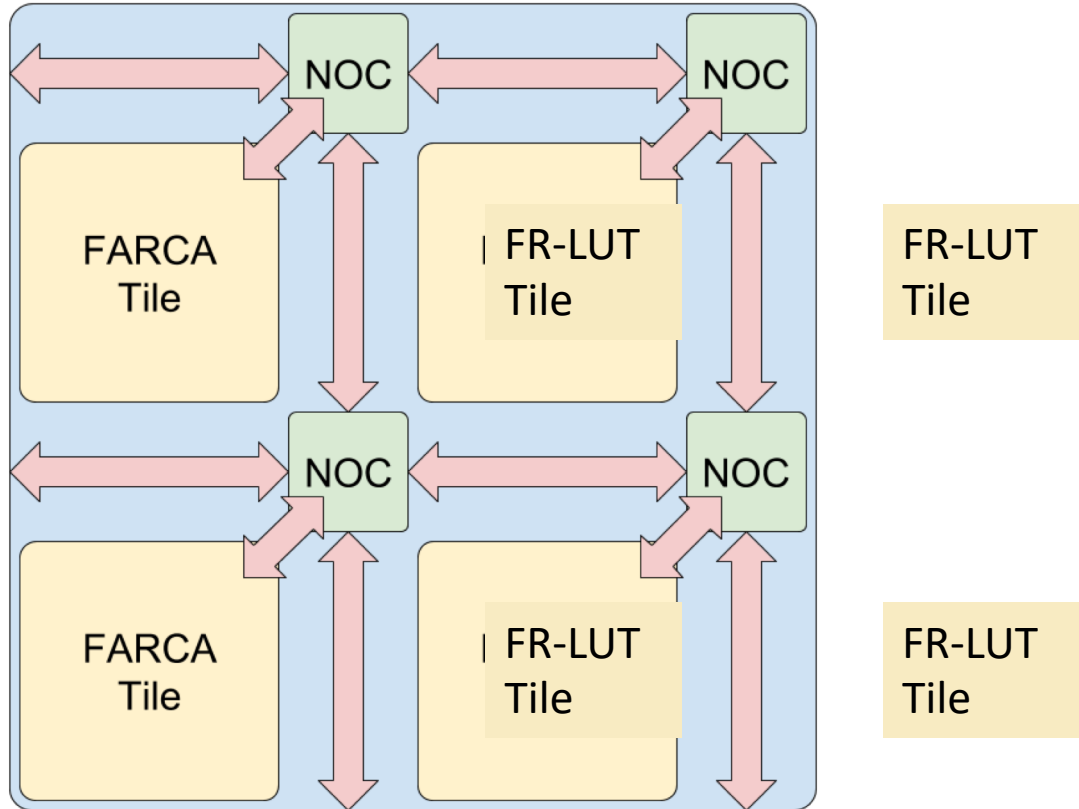
---

# DECADES Fast Reconfigurable LUT Tile



- CPU
  - Kernel scheduling
  - Codes that can't benefit from static acceleration
- Memory
  - Shared between CPU & reconfigurable array (RA)
- Reconfigurable Array
  - Partitionable
  - Runtime-configurable multi-context
  - BRAM & arithmetic block

# Array of FR-LUT Tiles




- NOC
  - Communication between tiles (CPUs)
  - Memory system messages
- Configurable memory system
- Integrates into DECADES tile array

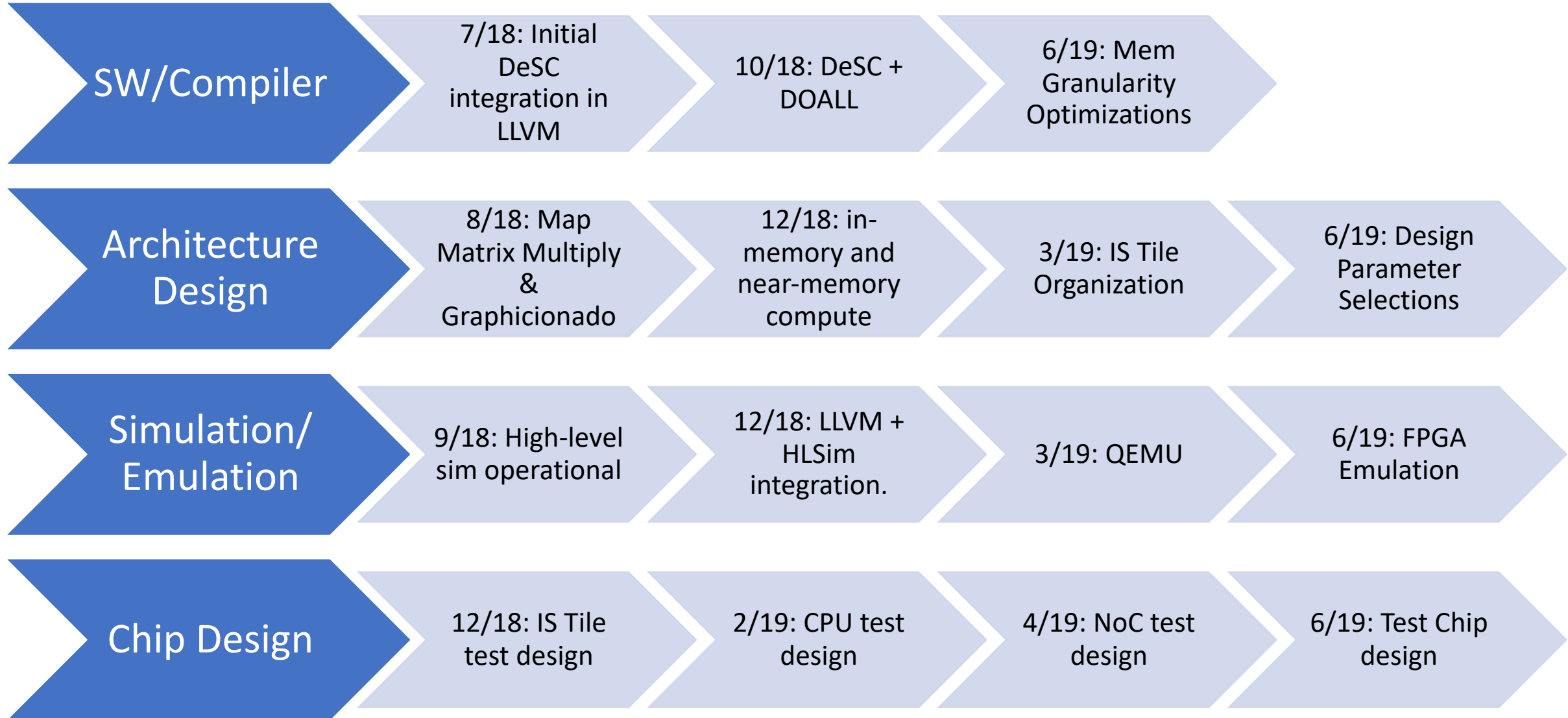
# DECADES Chip Progress

- Begun legal enablement of silicon technology acquisition (MOSIS, GF, Invecas)
- Brought Semiconductor Physical Design Kit (PDK) of Global Foundries GF 14LPP (14nm) in house at Princeton
  - Currently setting up design kit
- Working to acquire Hardware IP primitives needed
- Very early development of chip tool flow

Timeline and  
other issues...



# Phase 1 Technical Milestones and Plans





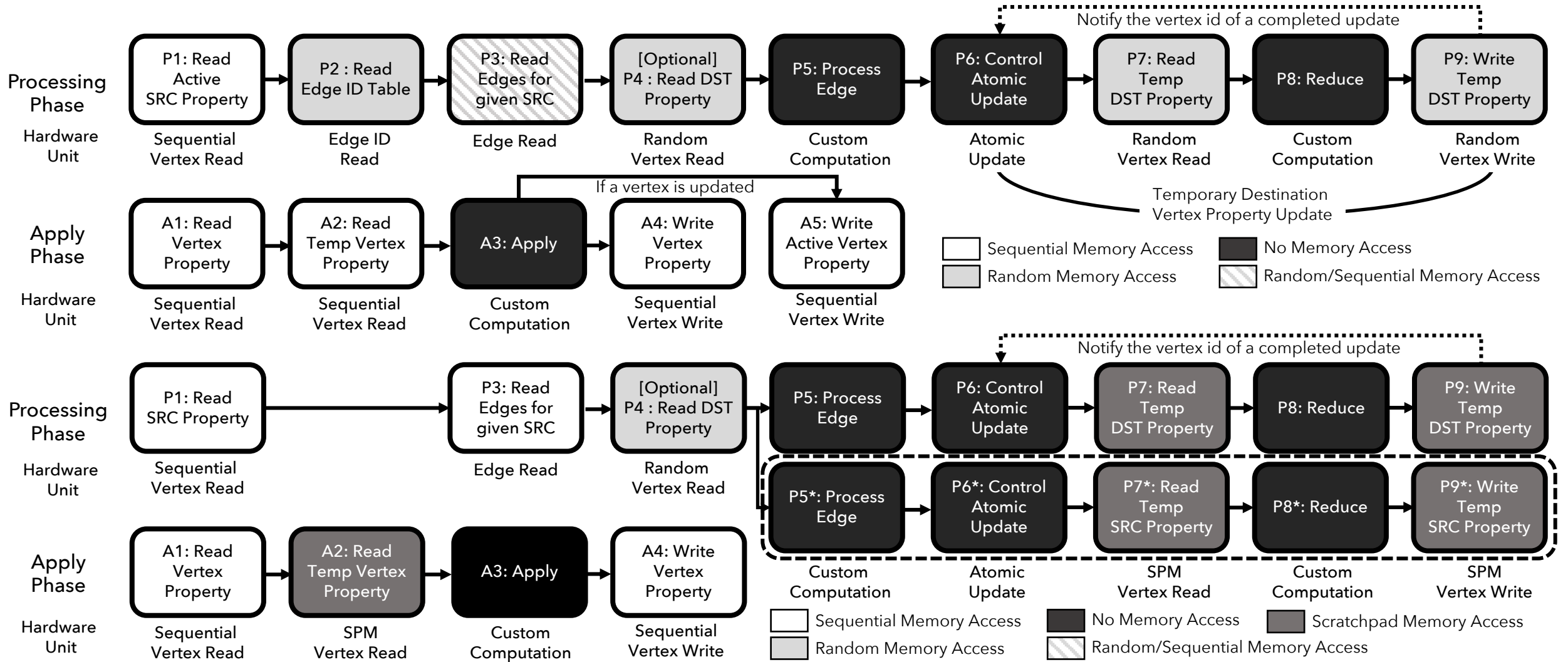
# Questions

- Demonstrate with DW (SPARC) core, but ideas apply to other ISA/Cores too
- Use existing compute accelerator designs wherever possible
  - Eg Luca Carloni Columbia CS class
- Design or synthesize data supply accelerators, using either CPU or IS tile
- Considering use of GPU tiles for some apps
  - Largely orthogonal to our primary ideas about custom data supply and interconnect
- OP + Accels speak AXI + ethernet, UARTS, etc
- FPGA Hardware at Columbia vs. FPGA resources in cloud

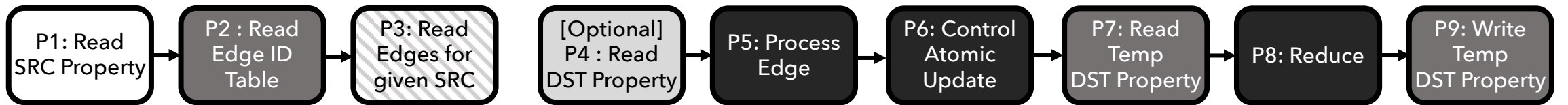
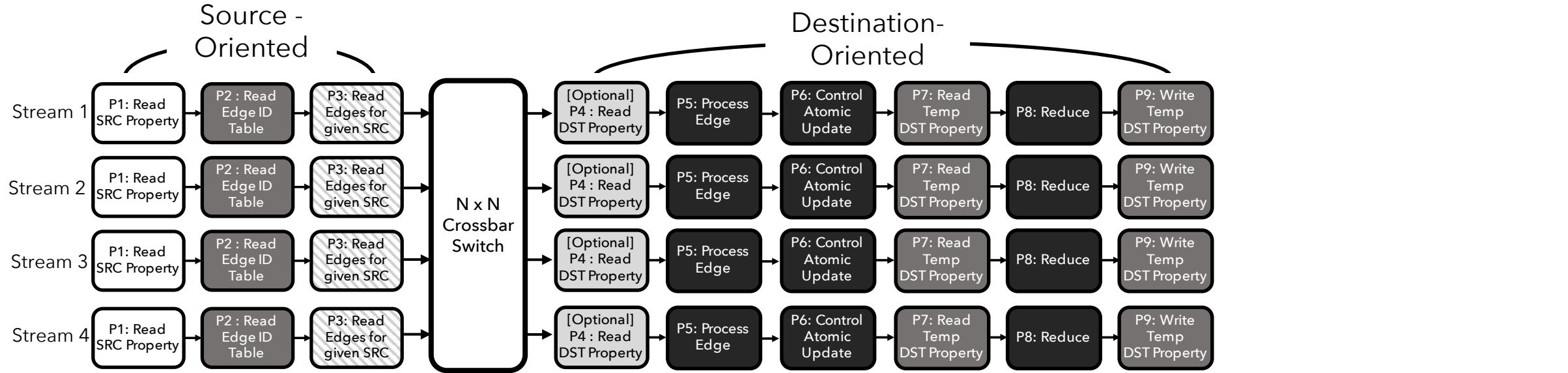
Backup slides



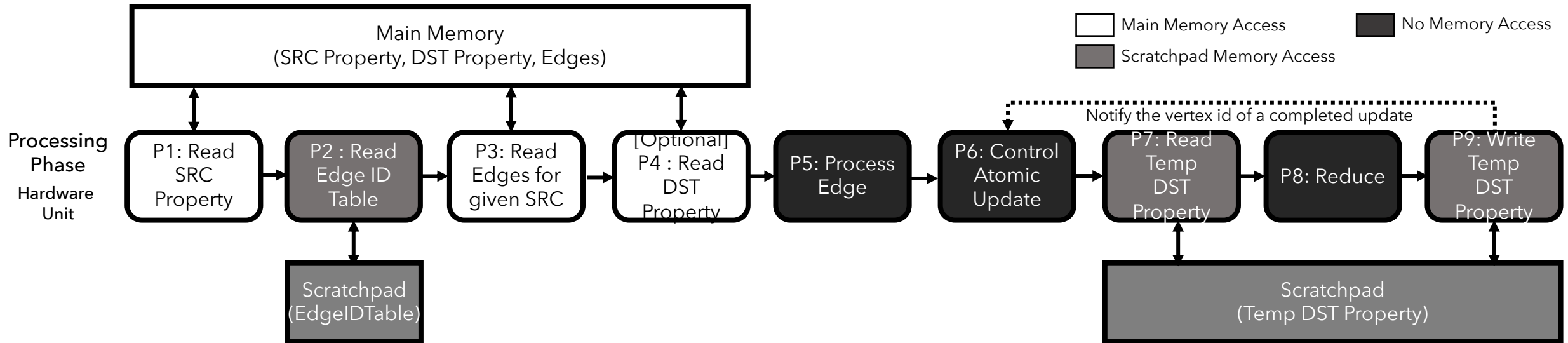
# Graphicionado



# Graphicionado

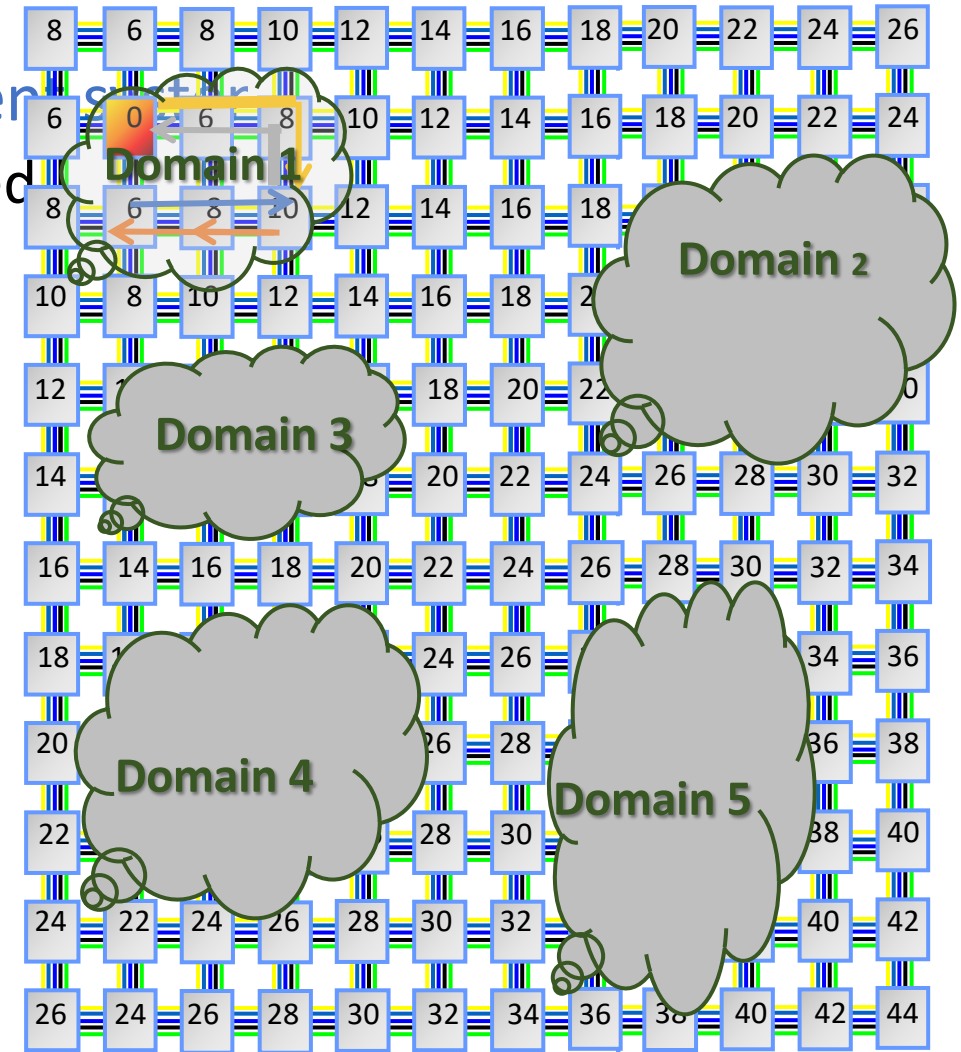


# Graphicionado



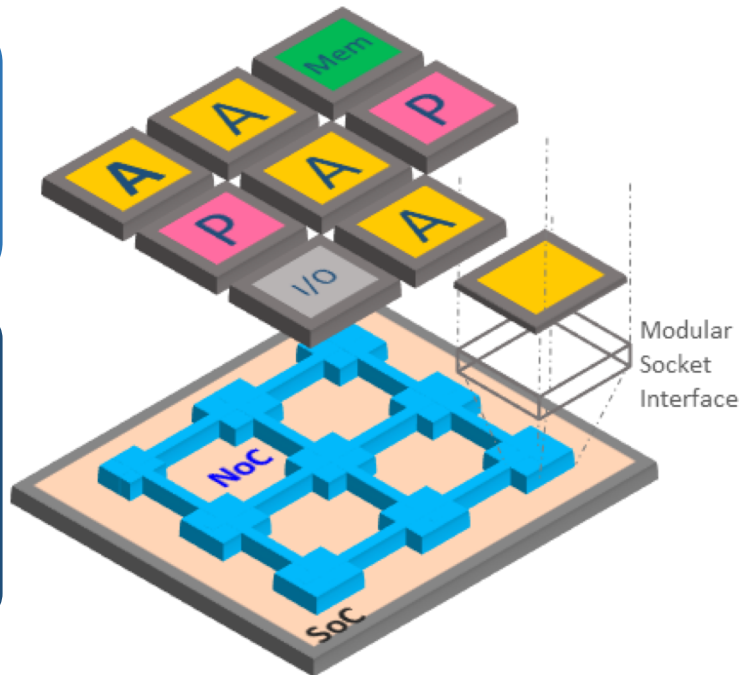
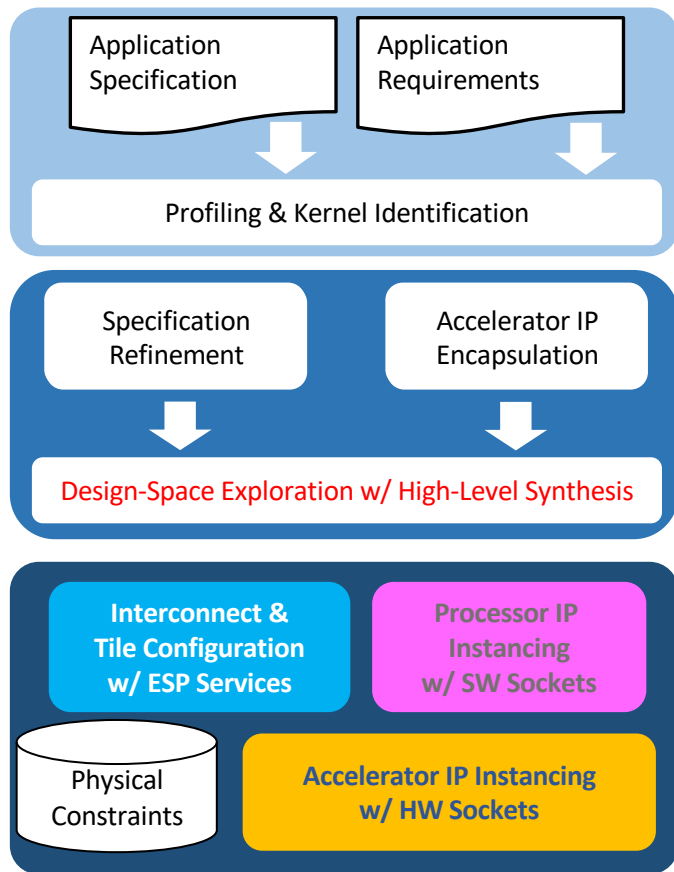
# PRIOR WORK: Coherence Domain Restriction Flexible Memory

- Flexible memory system on top of cache coherence system
  - Enables the exact minimal communication needed
  - Build incoherent coherent domains
- Restriction on application- or page-level
  - Improves performance
    - Shorter network on-chip distances
    - Less interfering memory coherence traffic
  - Reduces energy
    - Fewer on-chip network links need to be transited
    - Less area dedicated to tracking cache line sharers
  - Reduces area
    - Track fewer sharers on large configurations

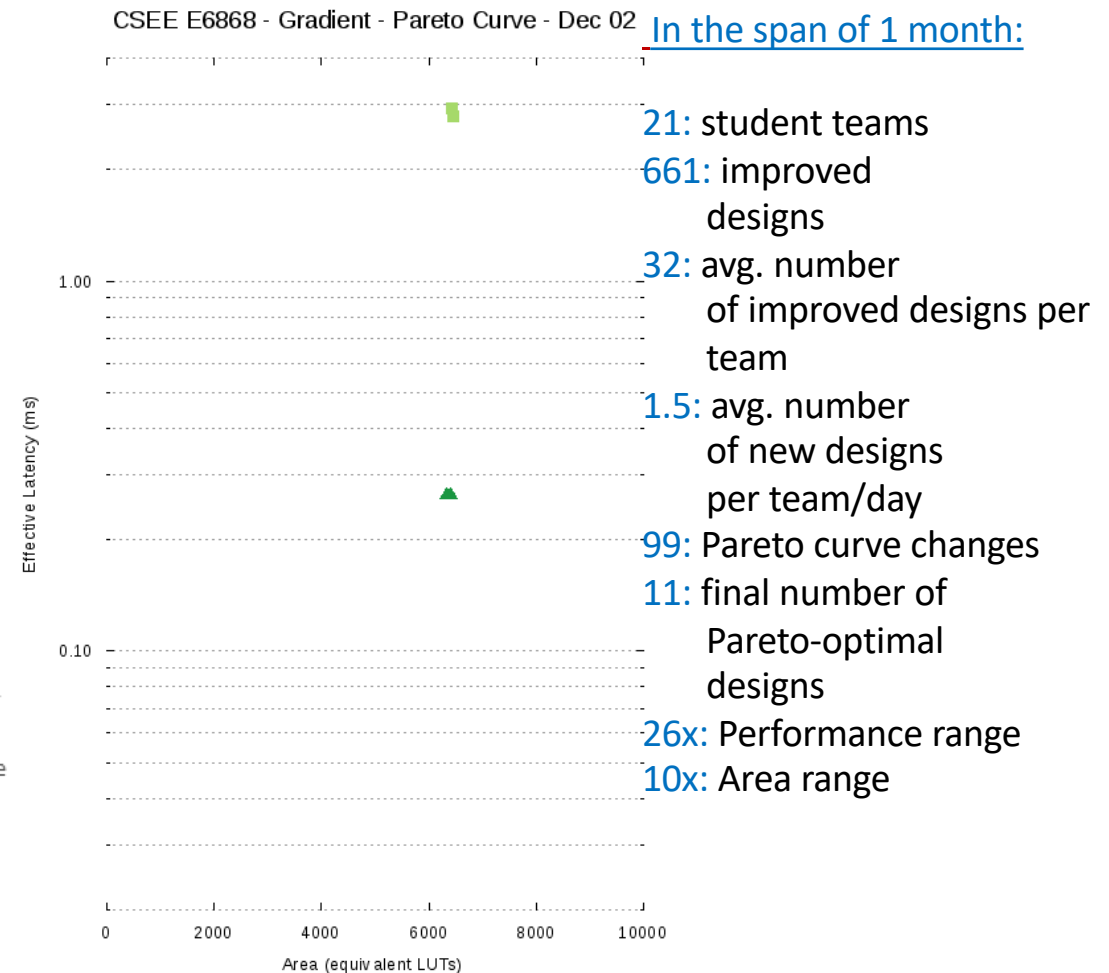


# PRIOR WORK: EMBEDDED SCALABLE PLATFORMS

- Flexible Tile-Based Architecture
- System-Level Design Methodology



- SoC Design Productivity

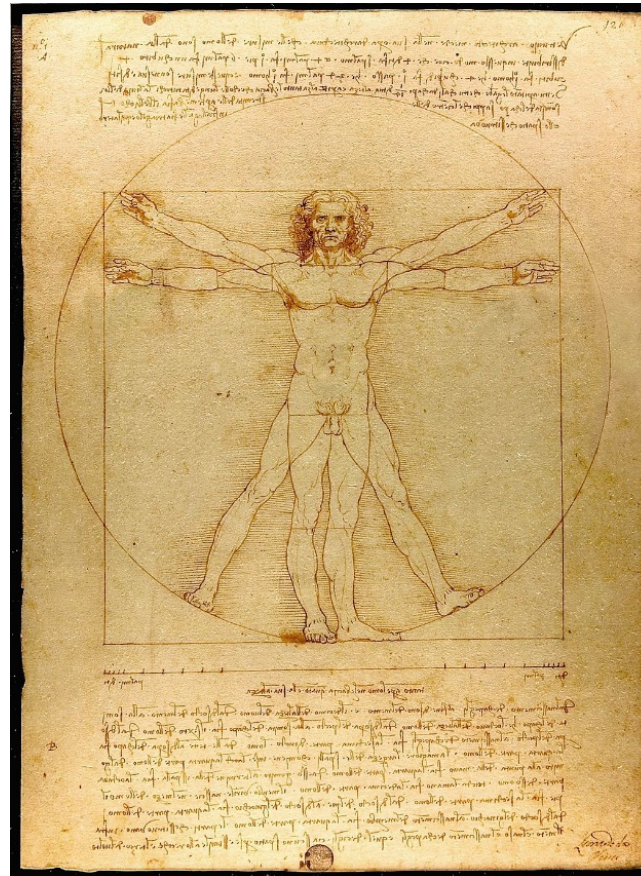


[Carloni, DAC 2016]

# TOWARDS A COMPUTER Design Renaissance

- The end of silicon dimensional scaling and the rise of heterogeneous reconfigurable computing bring an opportunity for a Computer Design Renaissance

- ...by supporting the creativity of application developers to realize innovative architectures, chips, systems and products



- ... through richly reconfigurable substrates and intelligent compilation and mapping