A Case for Richer Cross-layer Abstractions: Bridging the Semantic Gap with Expressive Memory

Nandita Vijaykumar

Abhilasha Jain, Diptesh Majumdar, Kevin Hsieh, Gennady Pekhimenko

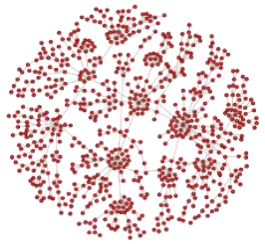
Eiman Ebrahimi, Nastaran Hajinazar, Phillip B. Gibbons, Onur Mutlu

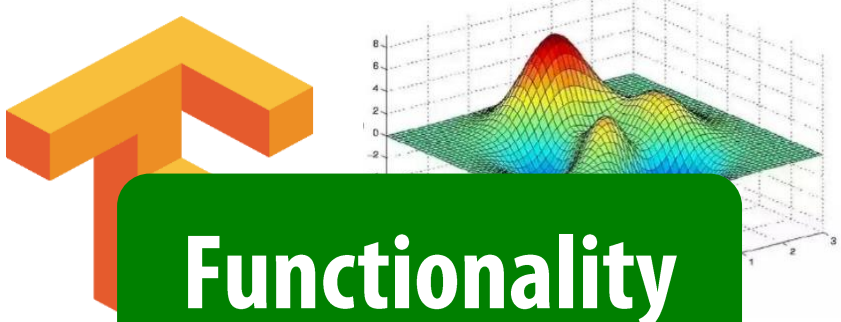
Carnegie Mellon University

UNIVERSITY OF TORONTO

SFU

ETH zürich





Functionality

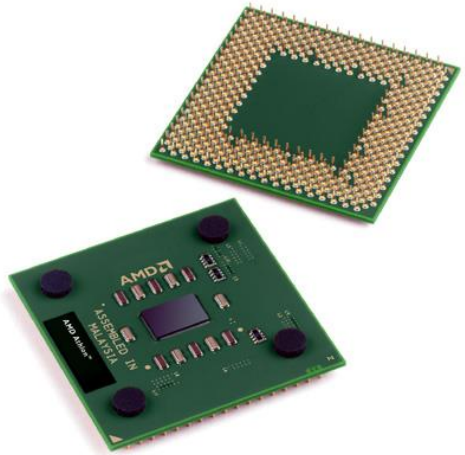
SW Optimization

Software

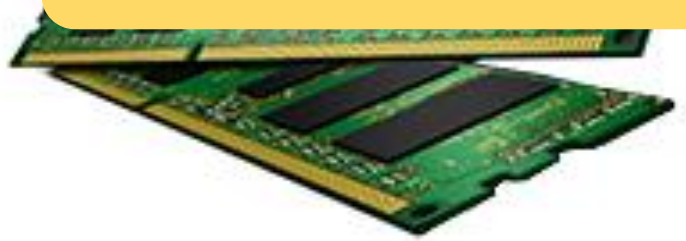
Performance?

HW-SW Interfaces
(ISA & Virtual Memory)

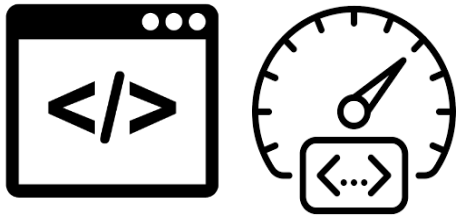
Hardware

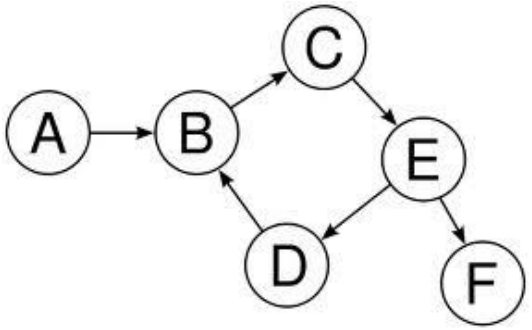


HW Optimization

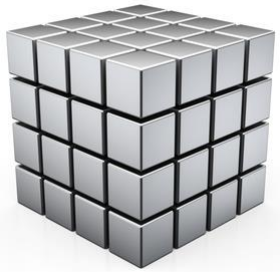


Higher-level information is not visible to HW

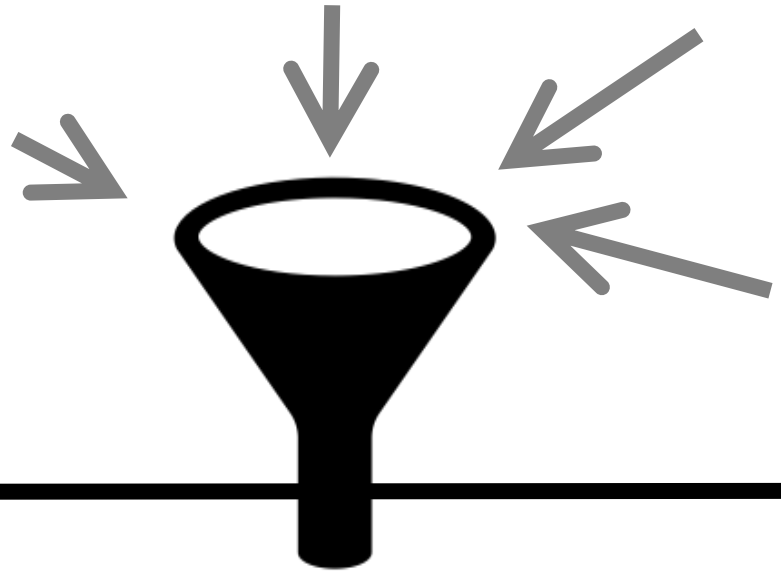




Access Patterns



Data Structures



`Integer` `Float`

Data Type `Char`

Software

Hardware

`100011111...`

`101010011...`

Instructions

Memory Addresses

Software

Performance

Functionality

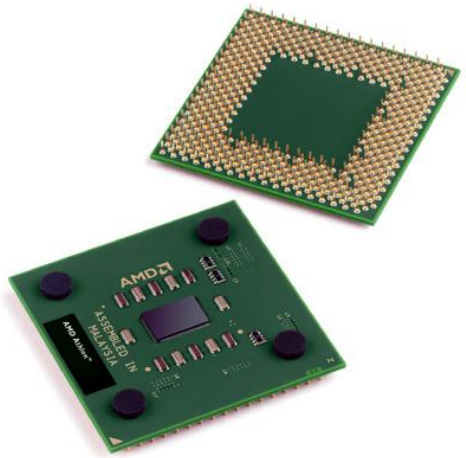
ISA

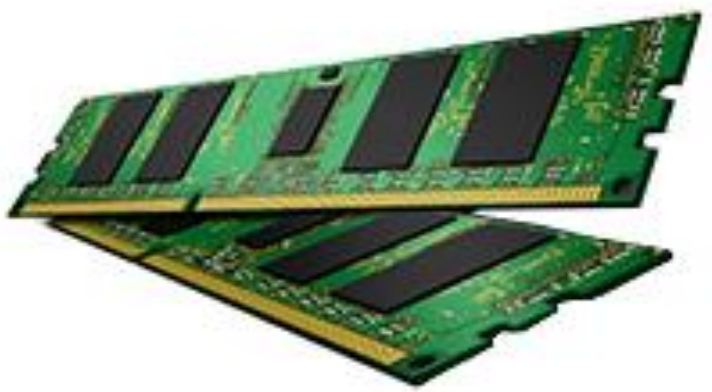
Virtual Memory

Higher-level Program Semantics

Expressive Memory "XMem"

Hardware





Outline

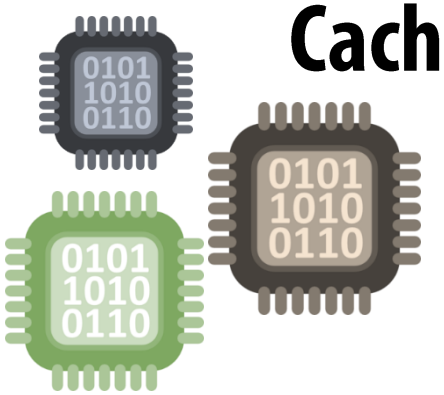
Why do we need a richer cross-layer abstraction?

Designing Expressive Memory (XMem)

Evaluation (with a focus on one use case)

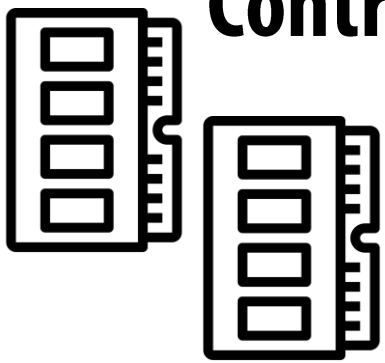
Performance optimization in hardware

What we do today: We design hardware to infer and predict program behavior to optimize for performance

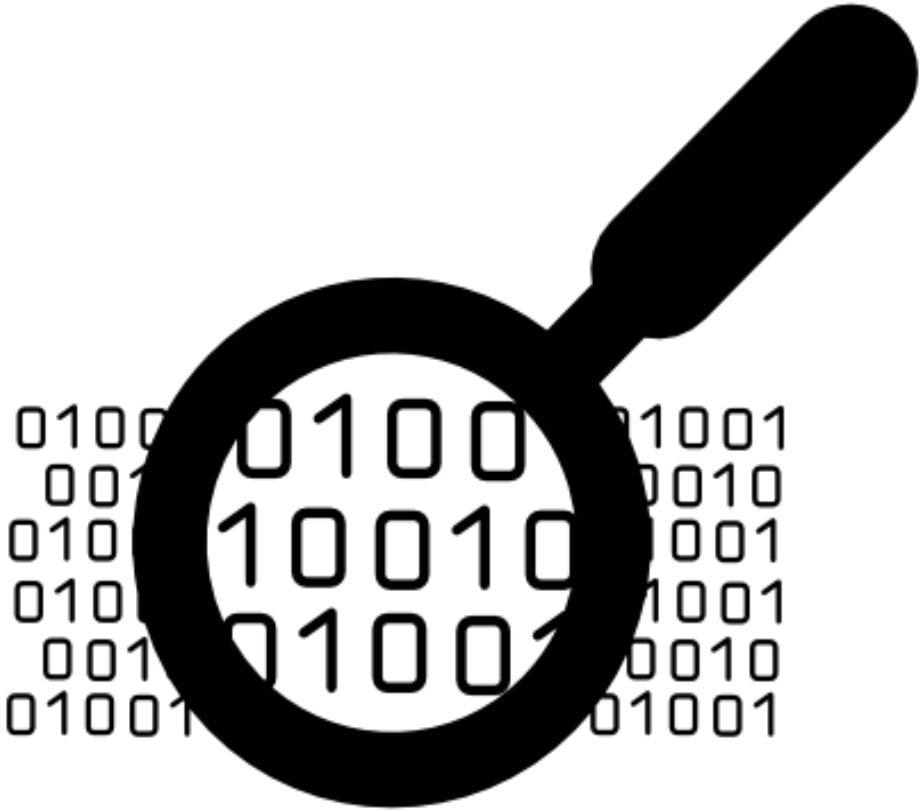


Caches

Prefetchers

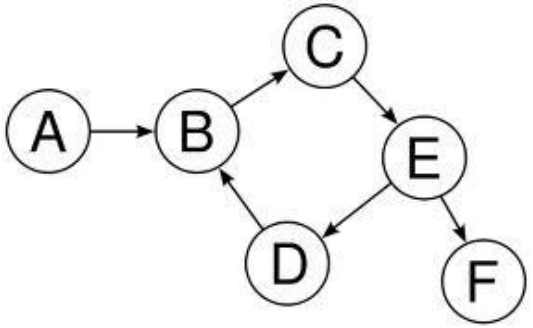


Memory Controllers



With a richer abstraction: SW can provide program information can significantly help hardware

Data Structures



Access Patterns

Integer

Float

Data Type/Layout

Char

Software

Hardware

Data Placement

Prefetcher

Data Compression

Benefits of a richer abstraction:

Express:

Data structures

Access semantics

Data types

Working set

Reuse

Access frequency

…..

Optimizations:

Cache Management

Data Placement in DRAM

Data Compression

Approximation

DRAM Cache Management

NVM Management

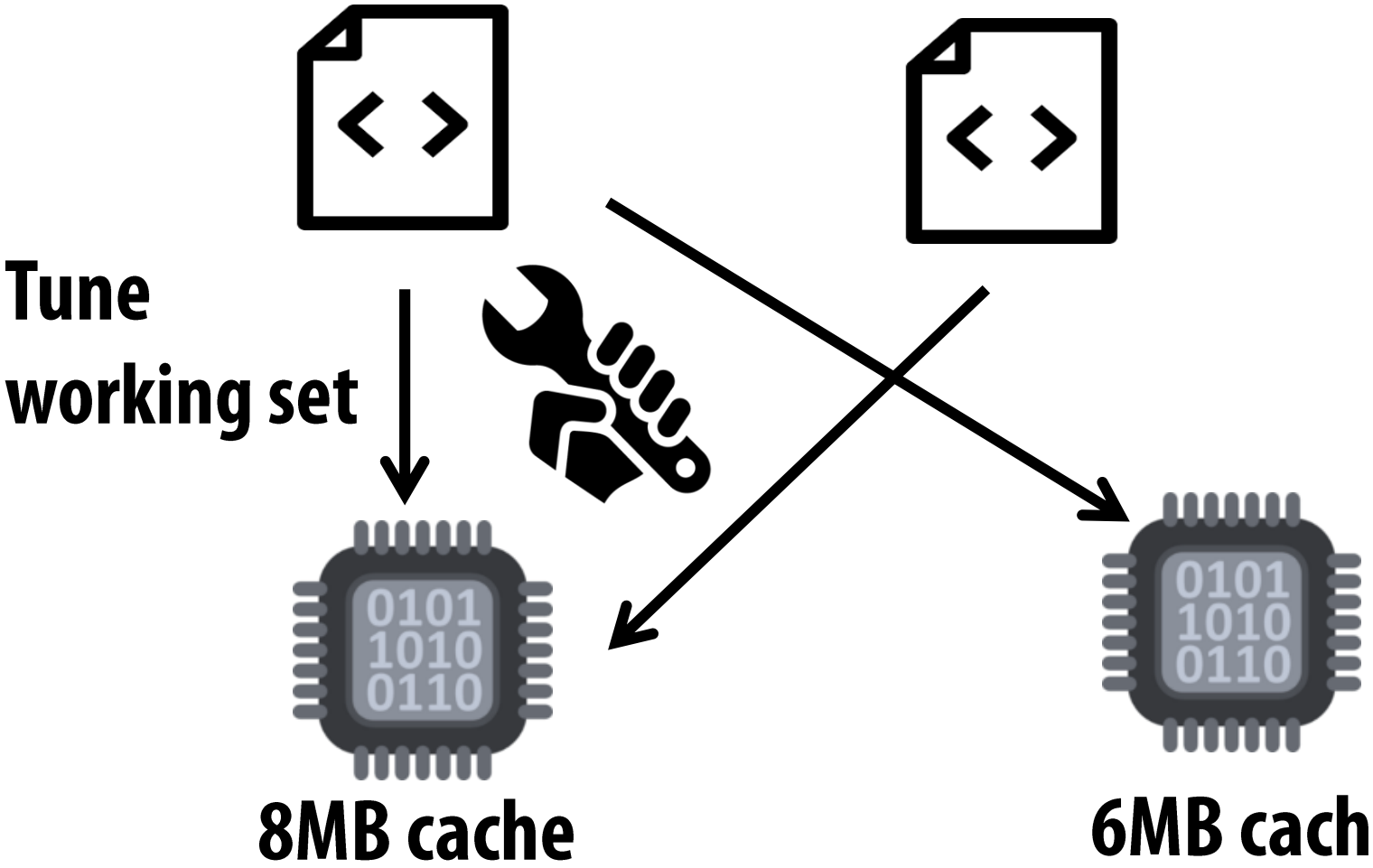
NUCA/NUMA Optimizations

….

Optimizing for performance in software

What we do today: Use platform-specific optimizations to tune SW

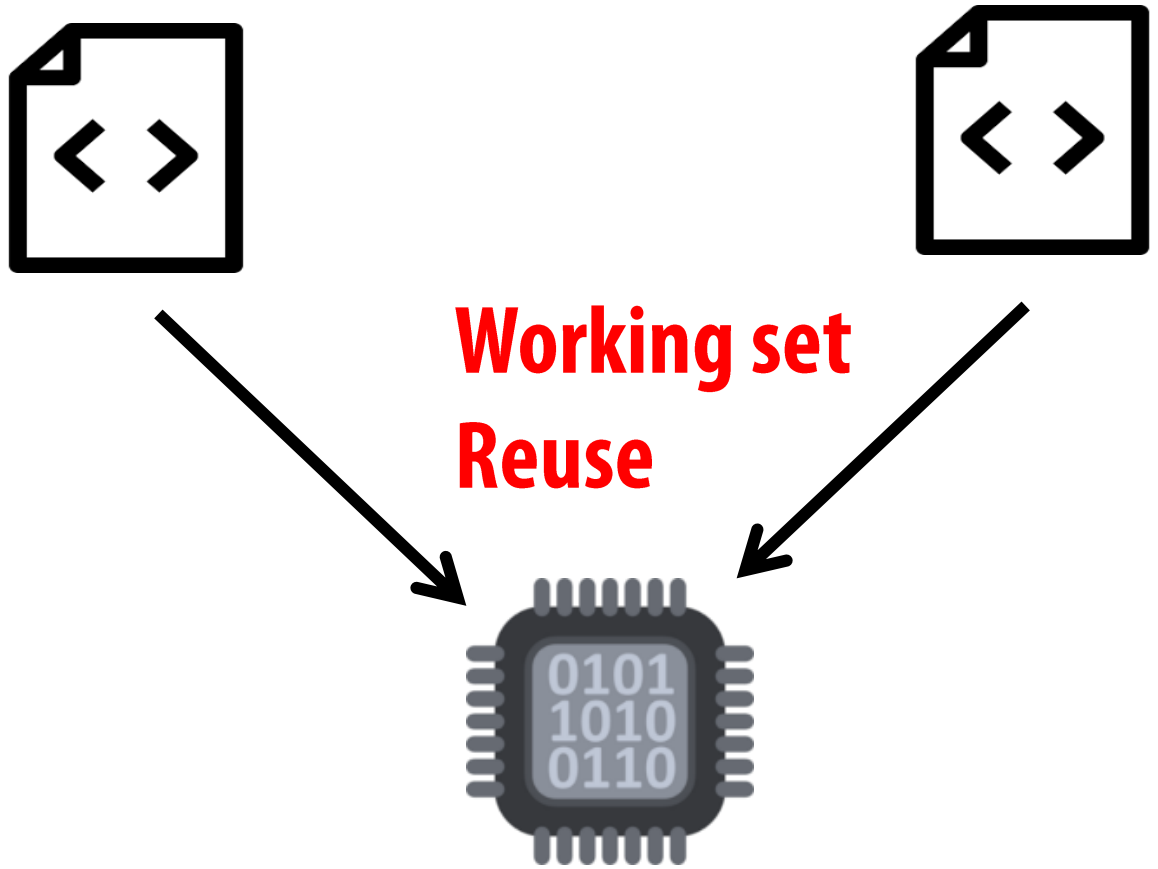
Example: SW-based cache optimizations



SW optimizations make assumptions regarding HW resources

Significant portability and programmability challenges

With a richer interface: HW can alleviate burden on SW



SW only expresses program information

System/HW handles optimizing for specific system details
e.g. exact cache size, memory organization, NUMA organization

Benefits of a richer abstraction:

Express:

Data structures

Access semantics

Data types

Working set

Reuse

Access frequency

…..

Optimizations:

Cache Management

Data Placement in DRAM

Data Compression

Approximation

DRAM Cache Management

NVM Management

NUCA/NUMA Optimizations

….

HW optimizations

✓Performance

SW optimizations

✓Programmability

✓Portability

SW information in HW is proven to be useful, but..

Lots of research on hints/directives to hardware and HW-SW co-designs

Cache hints, Prefetcher hints, Annotations for data placement, . . .

Downsides:

Not scalable – can't add new instructions for each optimization

Not portable – make assumptions about underlying resources

These downsides significantly limit adoption of otherwise useful approaches

Our Goal:

Design a rich, general, unifying abstraction between HW and SW for performance

Outline

Why do we need a richer cross-layer abstraction?

Designing Expressive Memory (XMem)

Evaluation (with a focus on one use case)

Key design goals

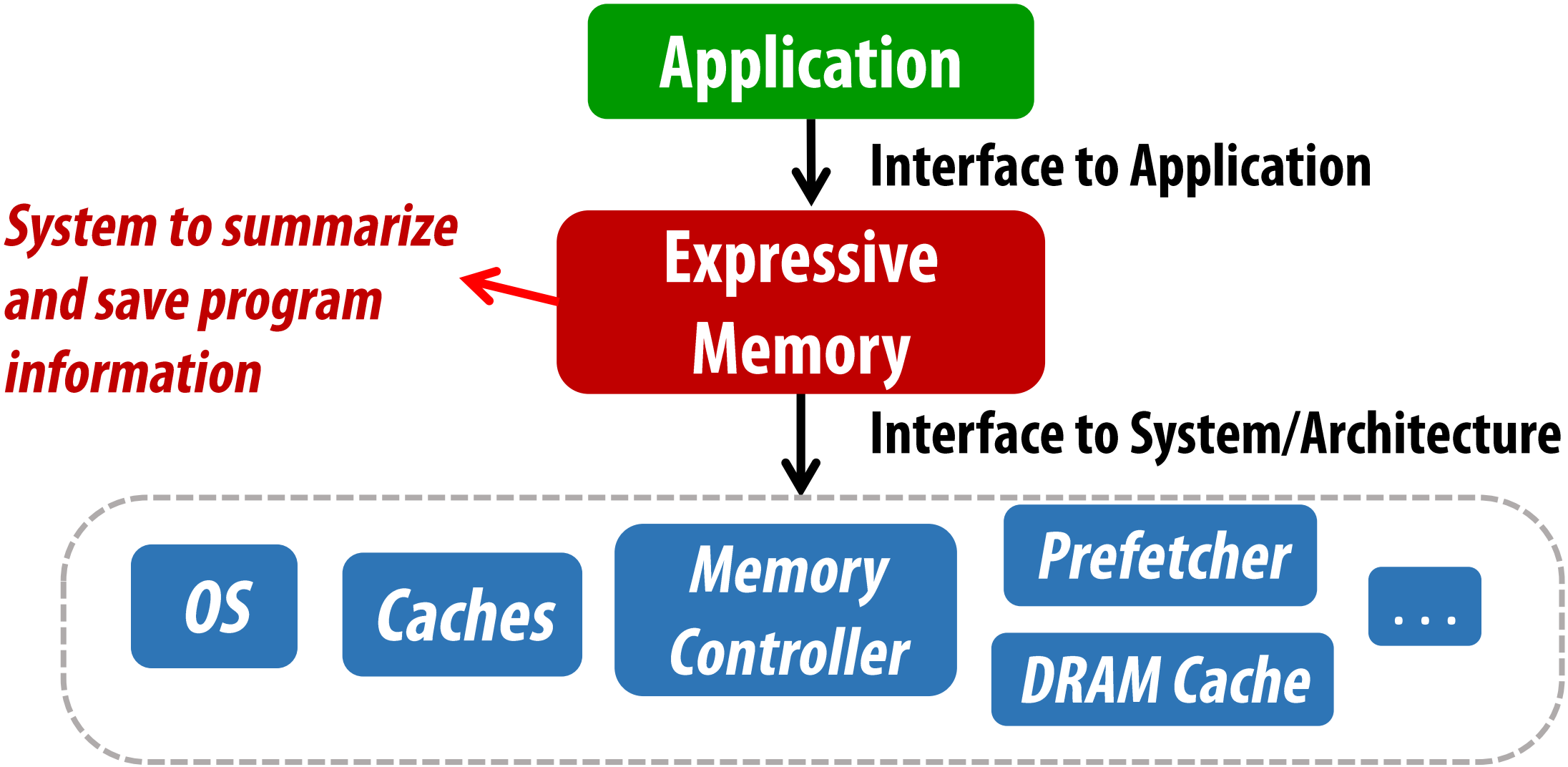
Supplemental and hint-based only

General and extensible

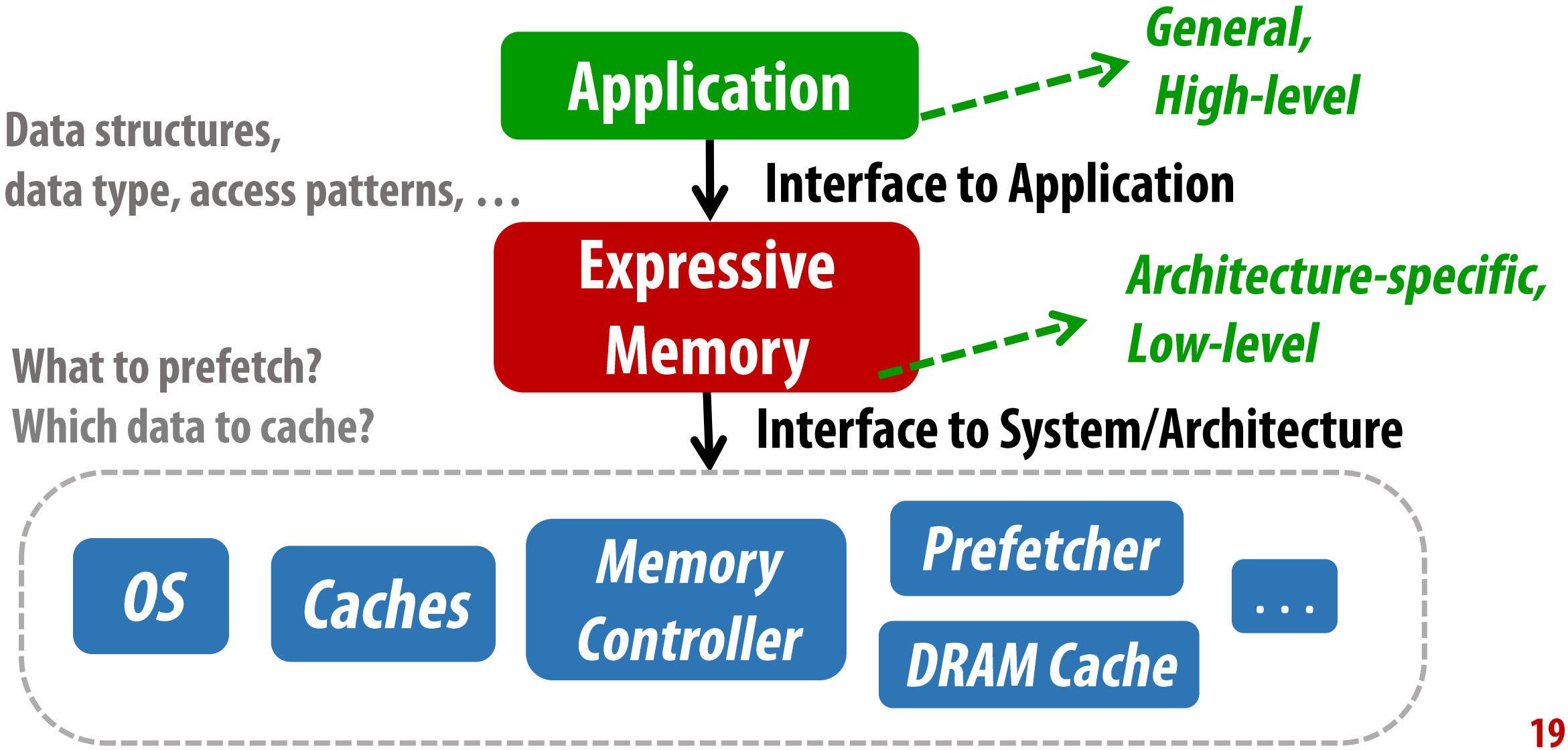
Architecture-agnostic

Low overhead

An Overview of Expressive Memory



Challenge 1: Generality and architecture-agnosticism

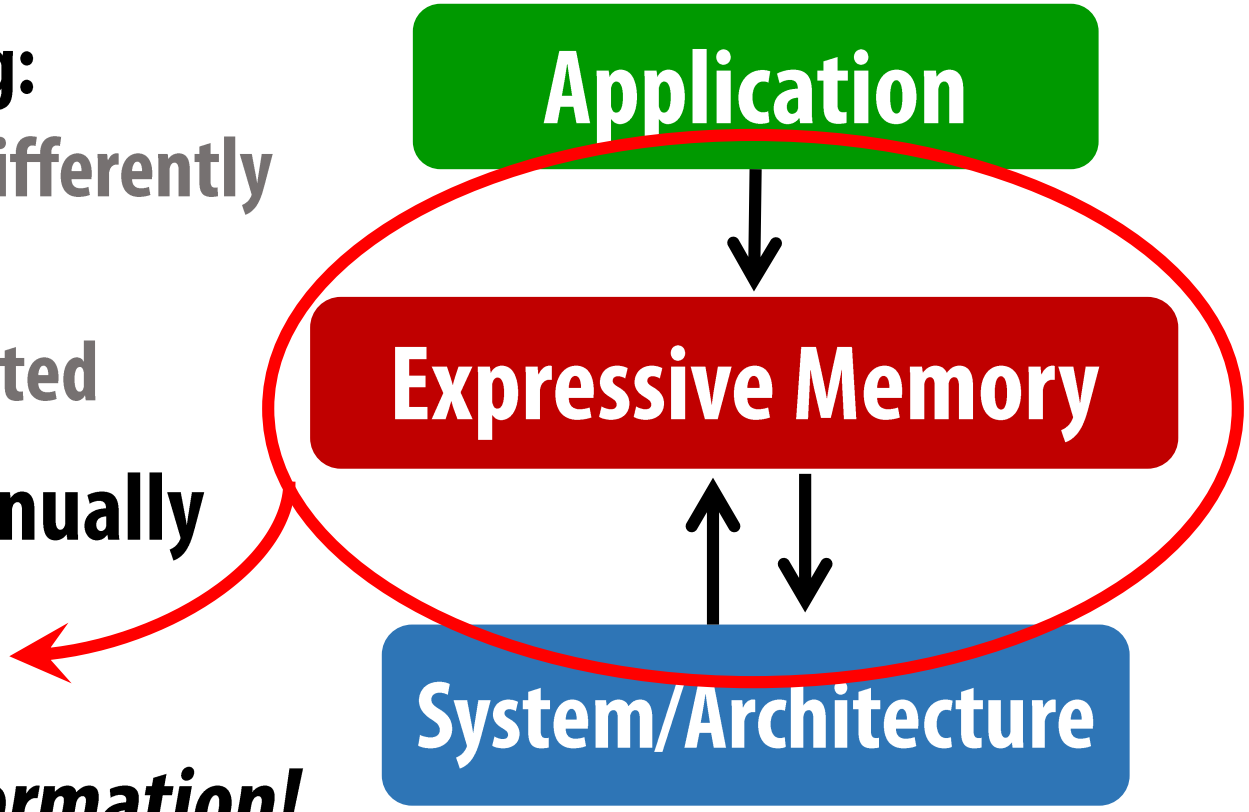


Challenge 2: Tracking changing program properties with low overhead

Program behavior keeps changing:

- Data structures are accessed differently in different phases
- New data structures are allocated

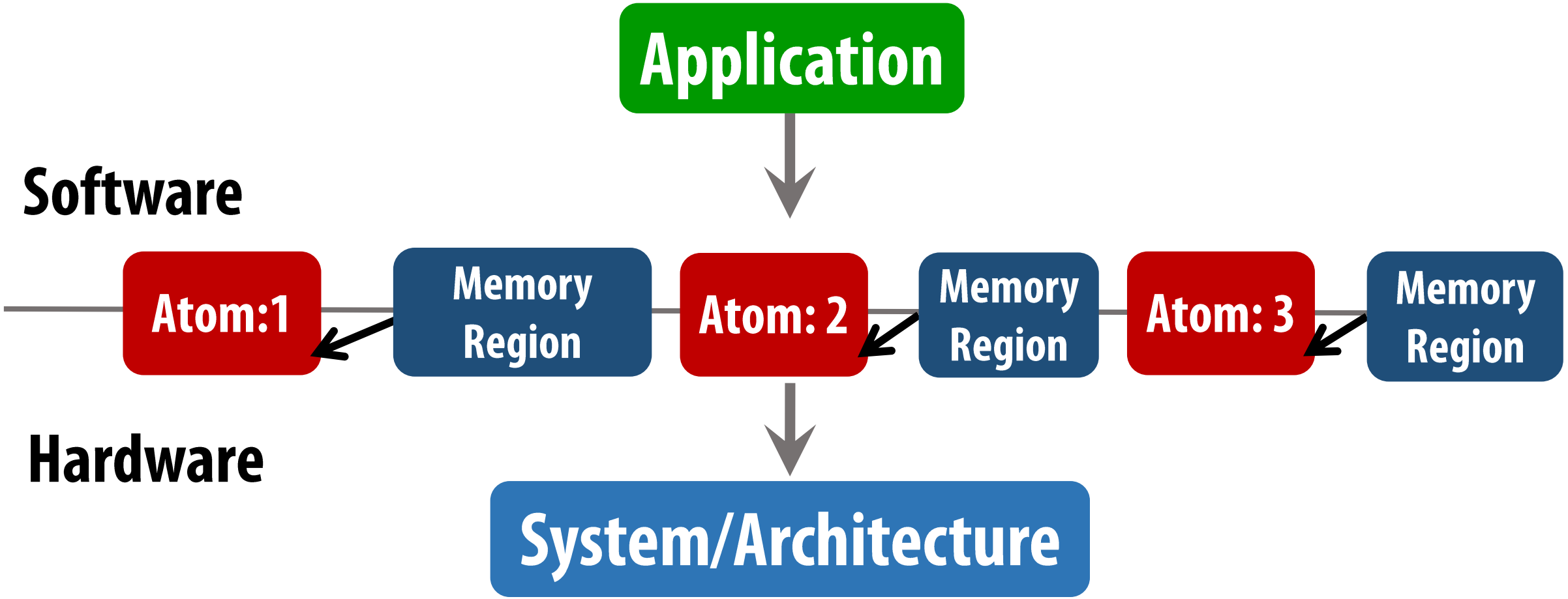
Dynamic interface that continually tracks program behavior



We want to convey lots of information!

Potentially very high storage/communication overhead at run time

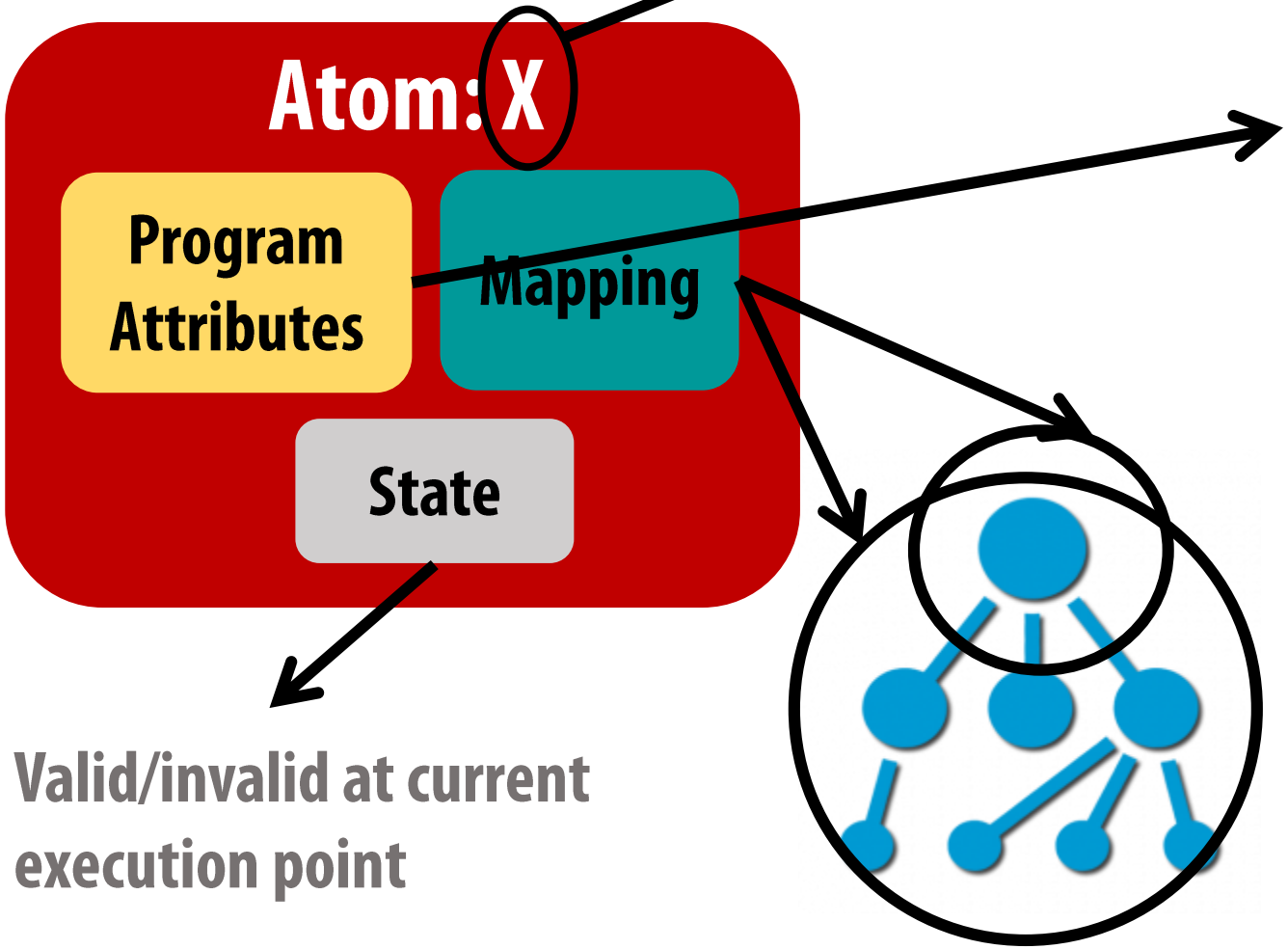
A new HW-SW abstraction



The Atom: A closer look

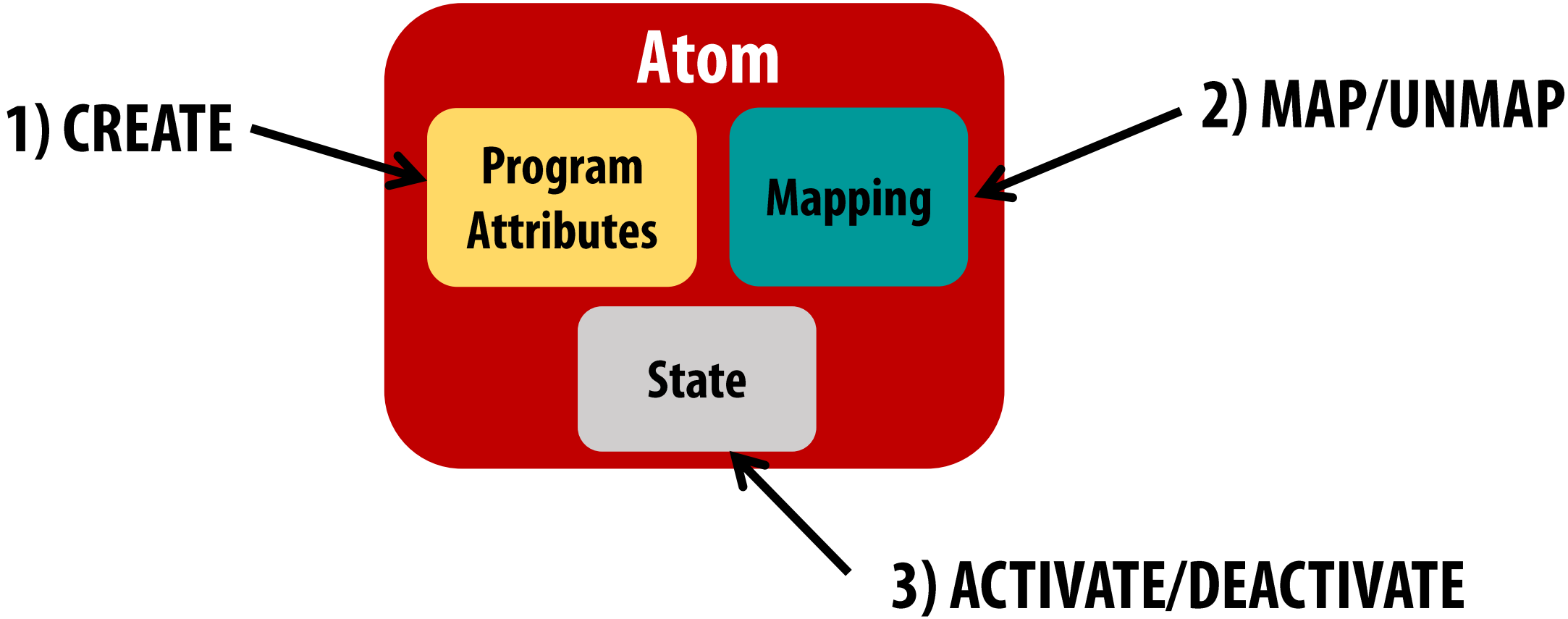
Unique Atom ID

A hardware-software abstraction to convey program semantics

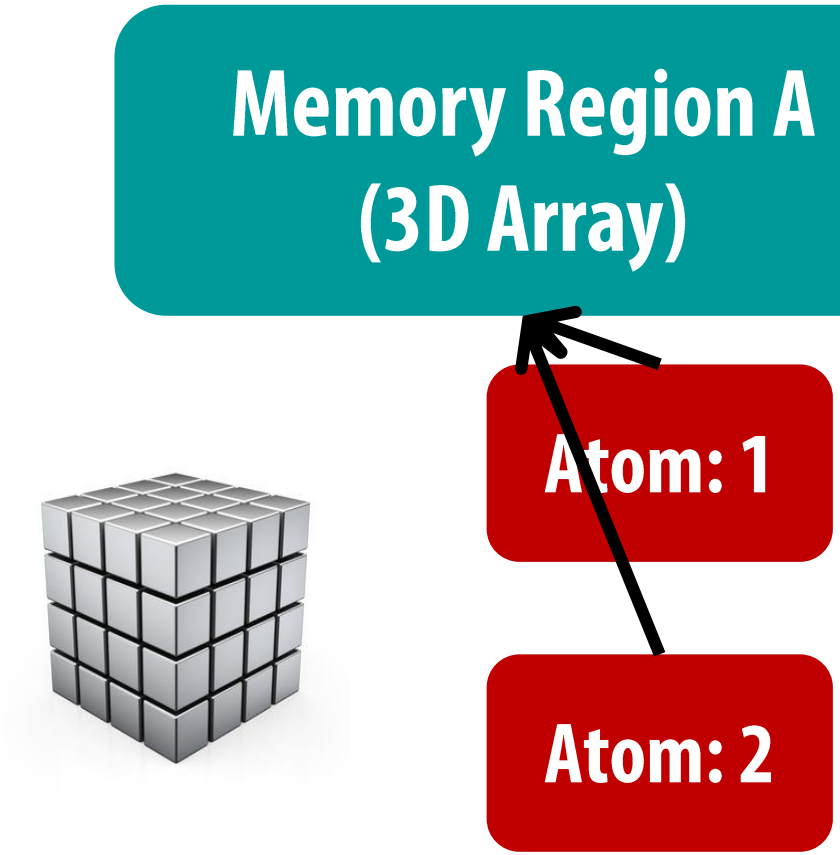


1) **Data Value Properties:**
   - INT, FLOAT, CHAR,…
   - COMPRESSIBLE, APPROXIMABLE
2) **Access Properties:**
   - Read-Write Characteristics
   - Access Pattern
   - Access Intensity (“Hotness”)
3) **Data Locality:**
   - Working Set
   - Reuse
4) ….

The three Atom operators



Using Atoms to express program semantics




```
A = malloc ( size );
Atom1 = CreateAtom( "INT", "Regular", …);
MapAtom( Atom1, A, size );
ActivateAtom(Atom1);
….
….
Atom2 = CreateAtom( "INT", "Irregular", …);
UnMapAtom( Atom1, A, size);
MapAtom( Atom2, A, size );
ActivateAtom(Atom2);
```

Attributes cannot be changed

Implementing the Atom

Compile Time (CREATE)

Load Time (CREATE)

Run Time (MAP and ACTIVATE)

Compile Time (CREATE)

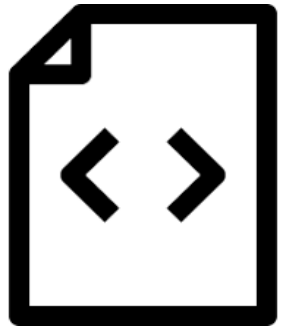
```
A = malloc ( size );
Atom1 = CreateAtom( "INT", "Regular", …);
```

Atom: 1

Program Attributes 1

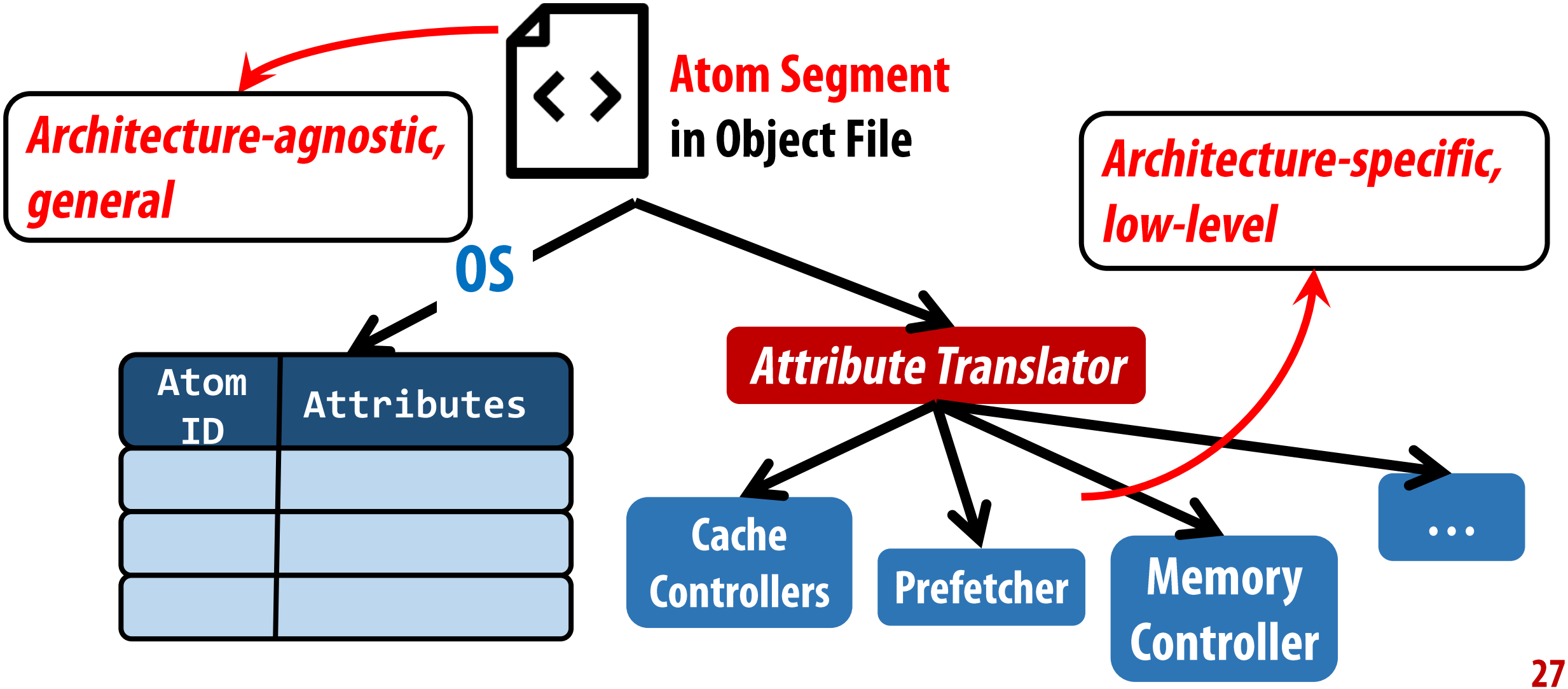
High overhead operations are handled at compile time

```
Atom2 = CreateAtom( "INT", "Irregular", …);
UnMapAtom( Atom1, A, size);
MapAtom( Atom2, A, size );
ActivateAtom(Atom2);
```



Atom Segment in Object File

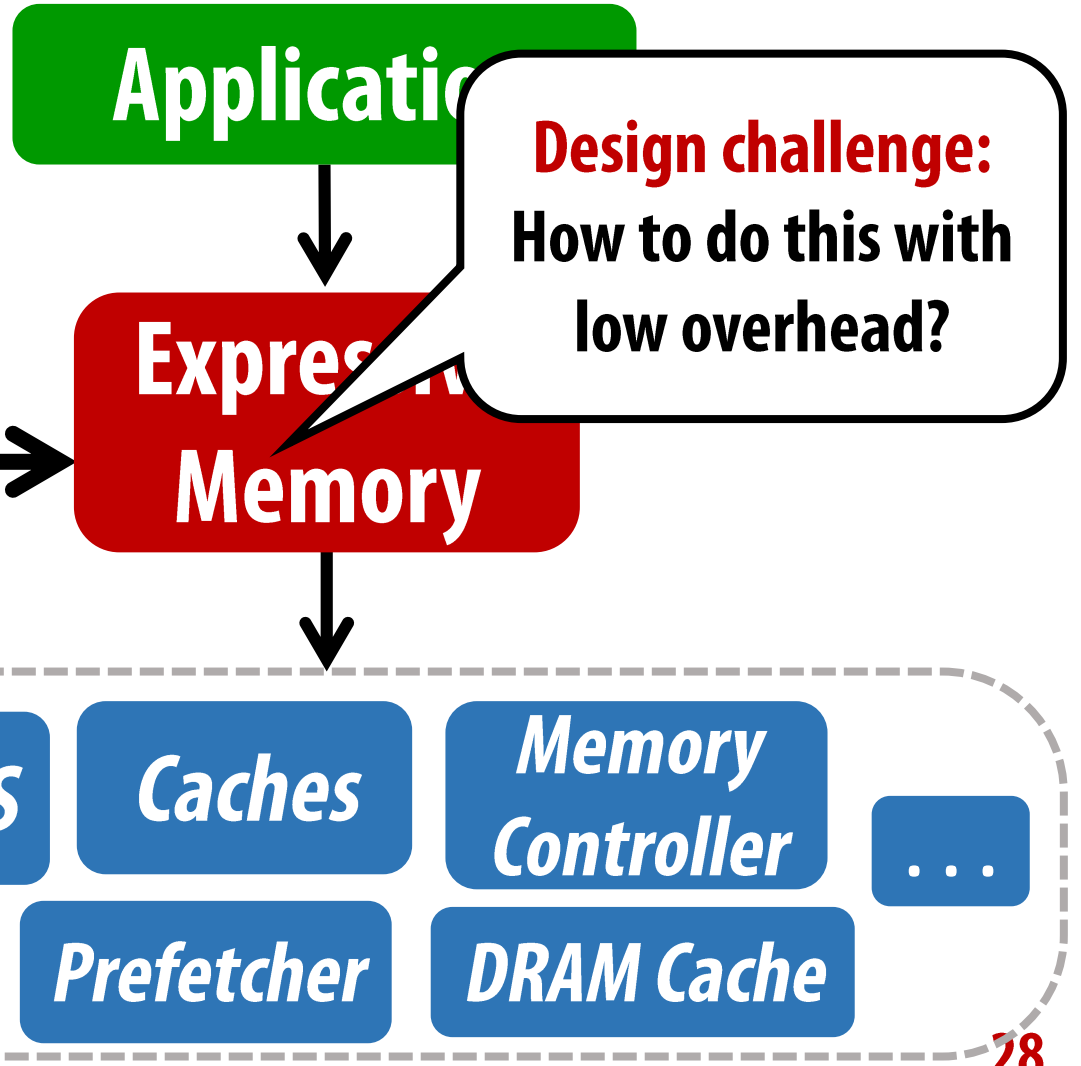
Load Time (CREATE)



Run Time (MAP and ACTIVATE)

```
A = malloc ( size );
Atom1 = CreateAtom( "INT", "Regular", ...);
MapAtom( Atom1, A, size );
ActivateAtom(Atom1);
....
....
Atom2 = CreateAtom( "INT", "Irregular", ...);
UnMapAtom( Atom1, A, size);
MapAtom( Atom2, A, size );
ActivateAtom(Atom2);
```

New insts in ISA



Outline

Why do we need a richer cross-layer abstraction?

Designing Expressive Memory (XMem)

Evaluation (with a focus on one use case)

A fresh approach to traditional optimizations

Express:

- Data structures
- Access semantics
- Data types
- Working set
- Reuse
- Access frequency
- …..

Optimizations:

- Cache Management
- Data Placement in DRAM
- Data Compression
- Approximation
- DRAM Cache Management
- NVM Management
- NUCA/NUMA Optimizations
- ….

HW optimizations

✓Performance

SW optimizations

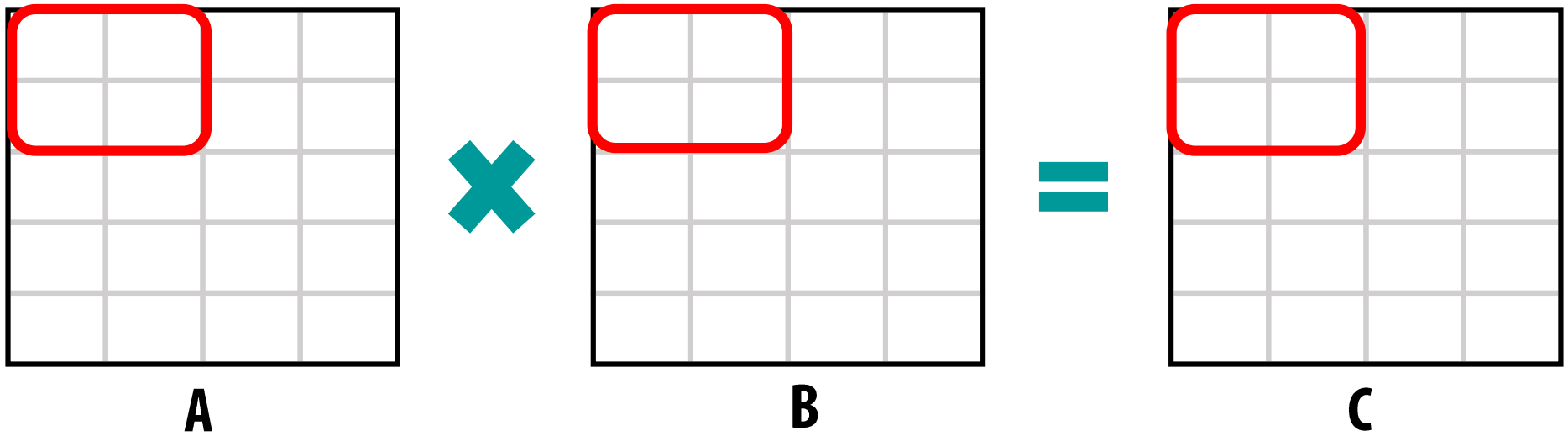
✓Programmability

✓Portability

Use Case 1: Improving portability of SW cache optimization

SW-based cache optimizations try to fit the working set in the cache

Examples: hash-join partitioning, cache tiling, stencil pipelining



Methodology (Use Case 1)

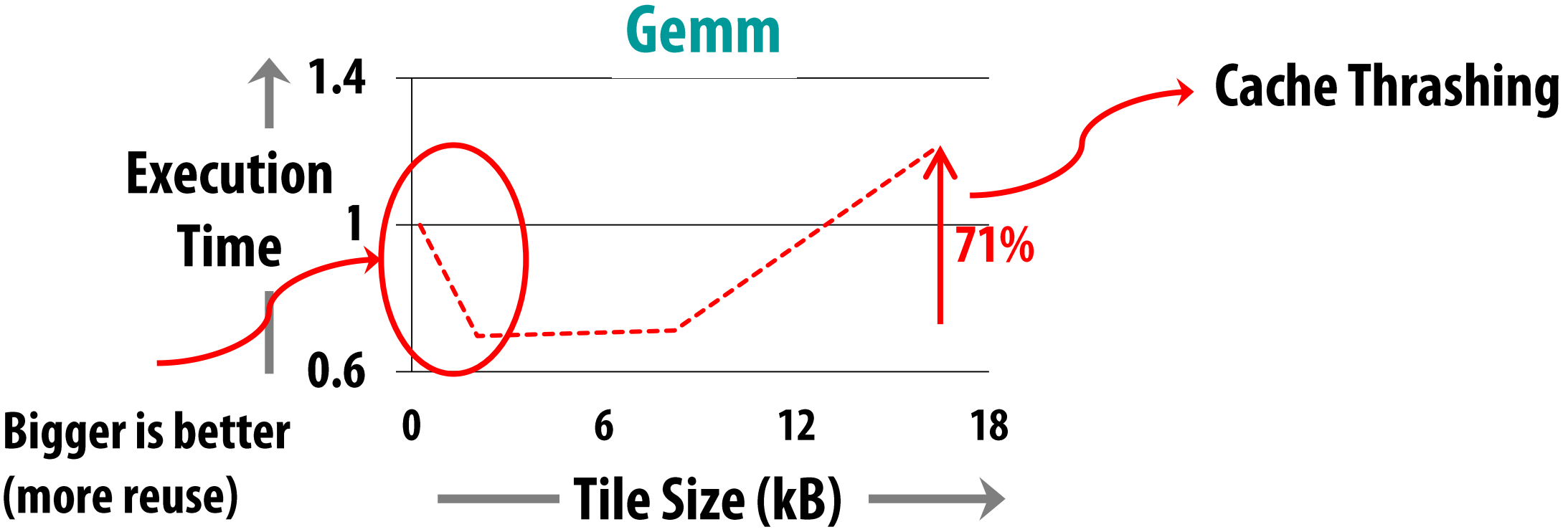
Evaluation Infrastructure: Zsim, DRAMSim2

Workloads: Polybench

System Parameters:

- **Core:** 3.6 GHz, Westmere-like OOO, 4-wide issue, 128-entry ROB
- **L1 Cache:** 32KB Inst and 32KB Data, 8 ways, 4 cycles, LRU
- **L2 Cache:** 128KB private per core, 8 ways, 8 cycles, DRRIP
- **L3 Cache:** 8MB (1MB/core, partitioned), 16 ways, 27 cycles, DRRIP
- **Prefetcher:** Multi-stride prefetcher at L3, 16 strides
- **Memory:** DRAM DDR3-1066, 2 channels, 1 rank/channel, 8 banks/rank, 17GB/s (2.1GB/s/core), FR-FCFS, open-row policy

Correctly sizing the working set is critical



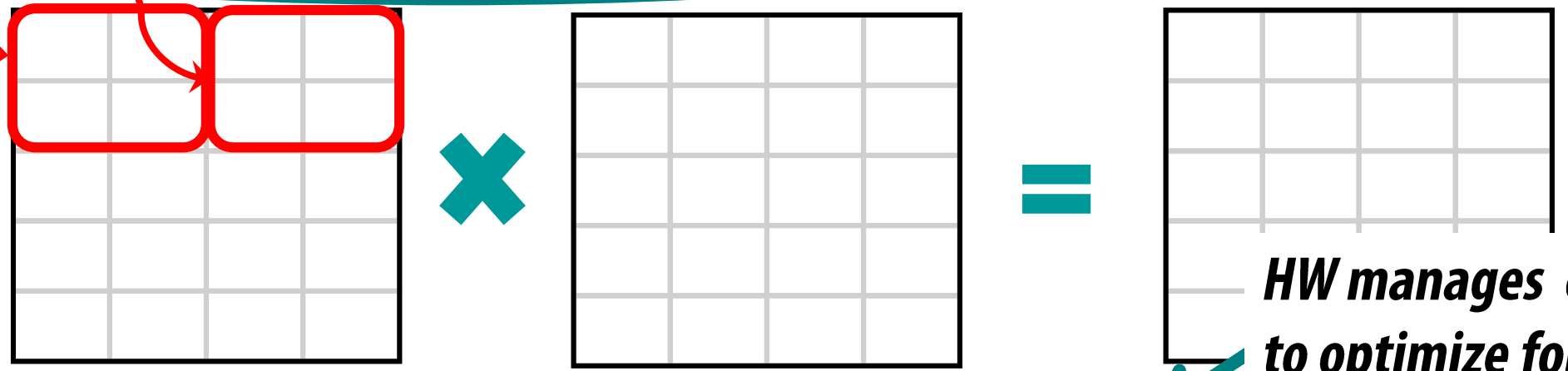
Optimal tile size depends on available cache space:

This causes portability and programmability challenges

Leveraging Expressive Memory for cache tiling

Map tile to an atom, specifying high reuse and tile size

SW expresses program-level semantic information

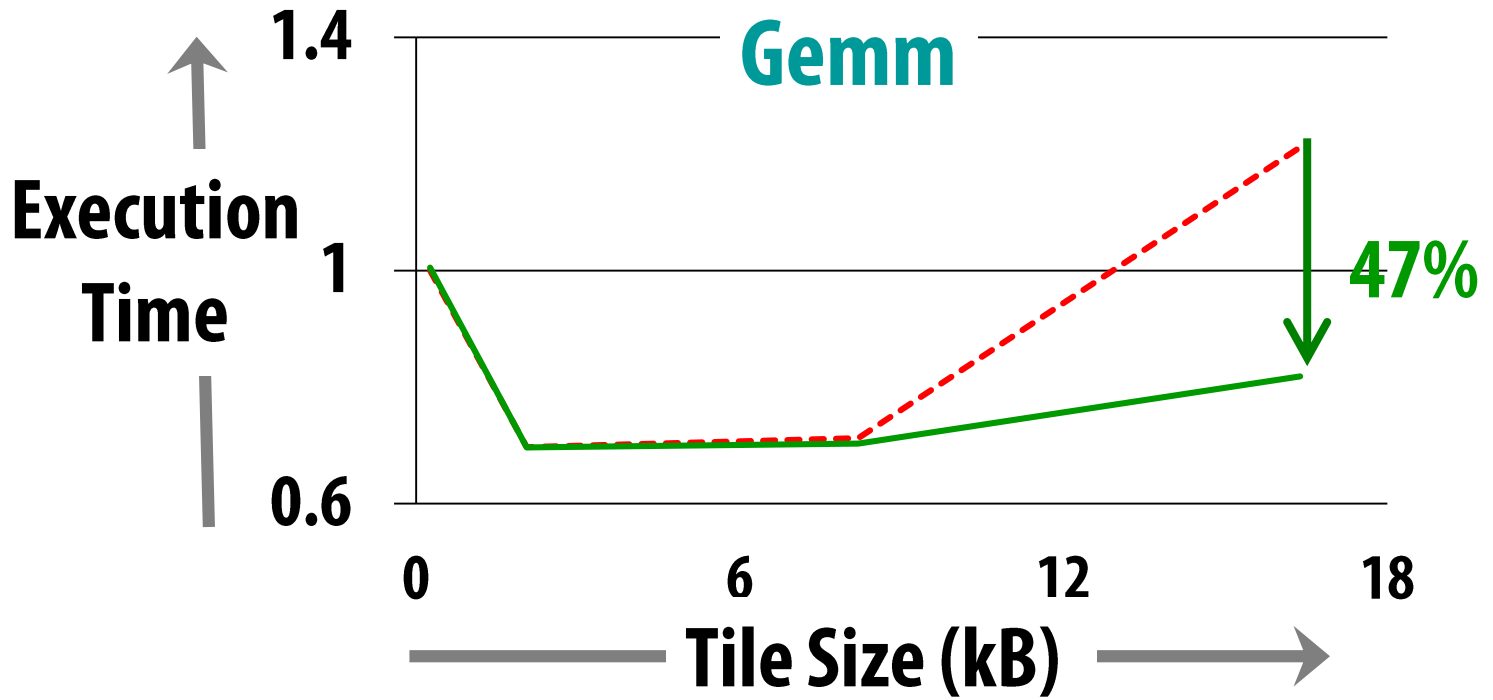


HW manages cache space to optimize for performance

If tile size < available cache space: default policy

If tile size > available cache space: pin a part of the tile, prefetch the rest (avoid thrashing)

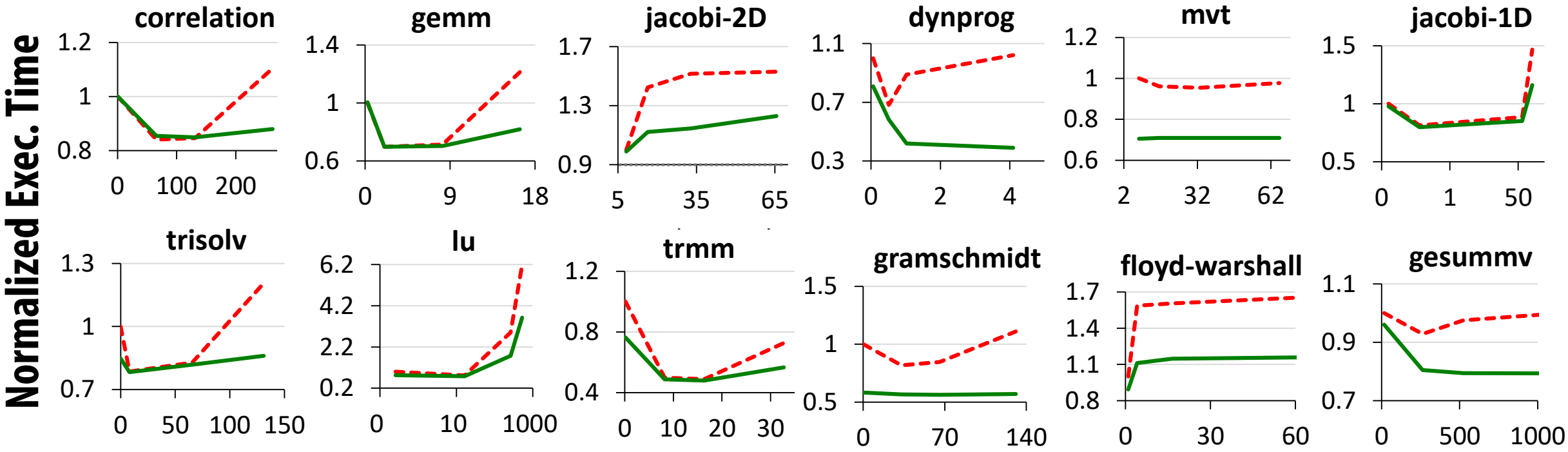
Cache tiling with Expressive Memory



Knowledge of locality semantics enables more intelligent cache management

Improves portability and programmability

Results across more workloads



More in the paper

Use Case 2: Leveraging data structure semantics to enable more intelligent OS-based page placement

More details on the implementation

Overhead analysis

Other use cases of XMem

Conclusion

General and architecture-agnostic interface to SW to express program semantics

Software

ISA

Virtual Memory

Higher-level Program Semantics

Expressive Memory "XMem"

Hardware

Key program information to aid system/HW components in optimization

A Case for Richer Cross-layer Abstractions: Bridging the Semantic Gap with Expressive Memory

Nandita Vijaykumar

Abhilasha Jain, Diptesh Majumdar, Kevin Hsieh, Gennady Pekhimenko

Eiman Ebrahimi, Nastaran Hajinazar, Phillip B. Gibbons, Onur Mutlu

Carnegie Mellon University

UNIVERSITY OF TORONTO

SFU

ETH zürich