The Locality Descriptor

A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs

Nandita Vijaykumar

Eiman Ebrahimi, Kevin Hsieh, Phillip B. Gibbons, Onur Mutlu

Carnegie Mellon University

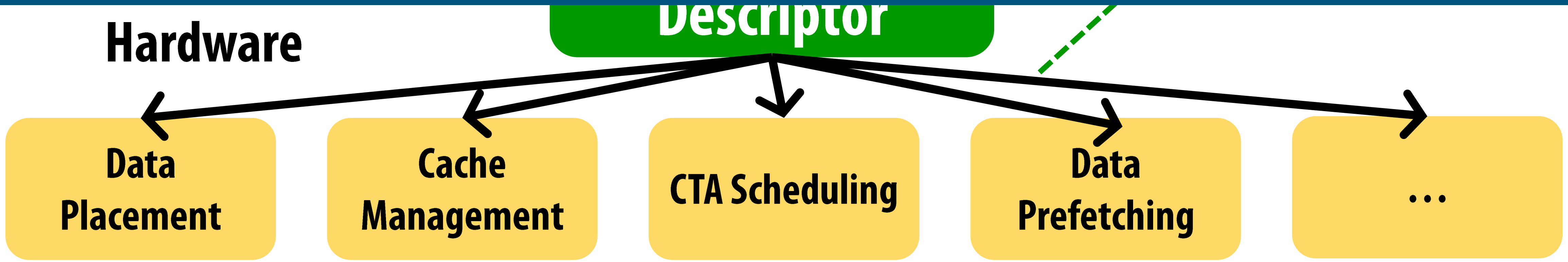
NVIDIA

ETH Zürich

Executive Summary

Exploiting data locality in GPUs is a challenging task

Performance Speedups:
26.6% (up to 46.6%) from <u>cache locality</u>
53.7% (up to 2.8x) from <u>NUMA locality</u>



Outline

Why leveraging data locality is challenging?

Designing the Locality Descriptor

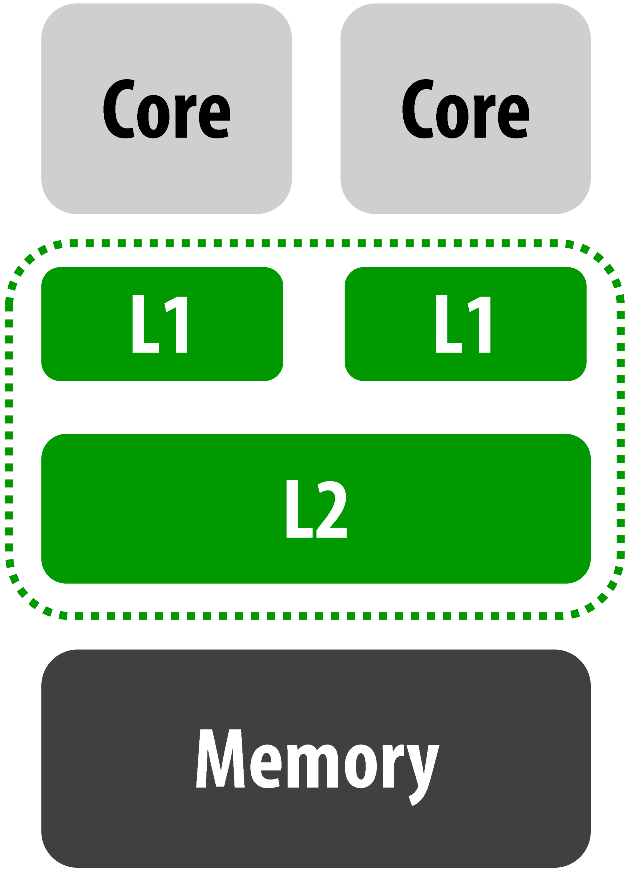
Evaluation

Data locality is critical to GPU performance

Two forms of data locality:

Reuse-based locality (cache locality)

NUMA locality



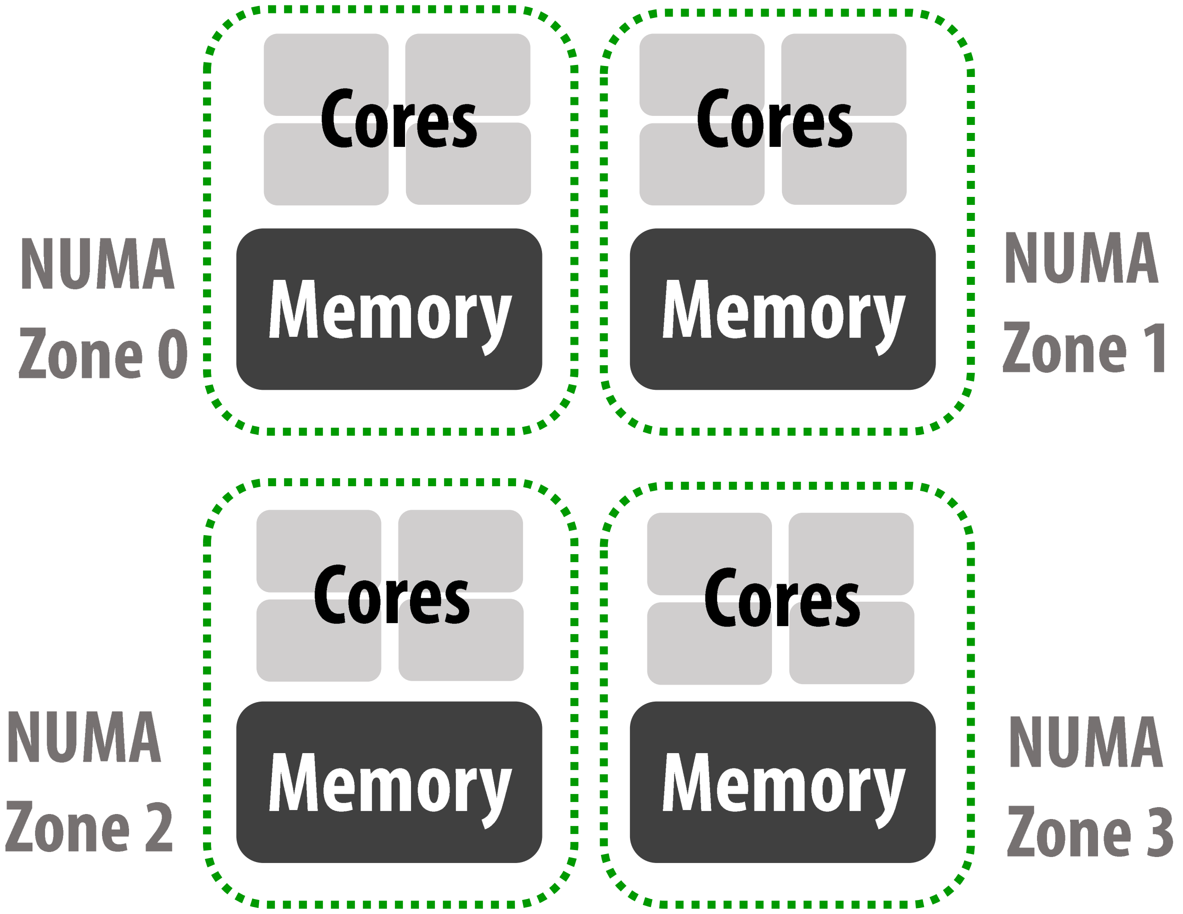
Reuse-based (Cache Locality)

Data locality is critical to GPU performance

Two forms of data locality:

Reuse-based locality (cache locality)

NUMA locality



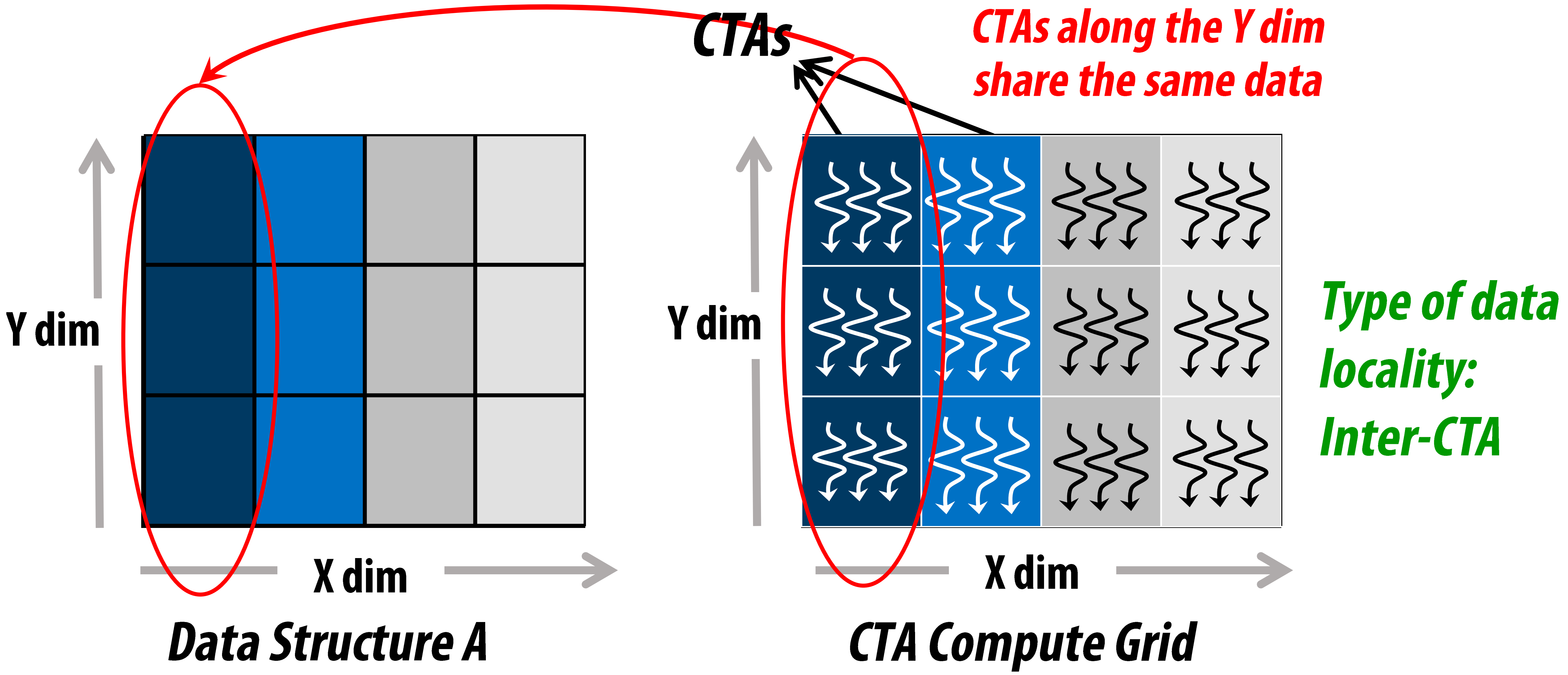
NUMA Locality

The GPU execution and programming models are designed to explicitly express parallelism…

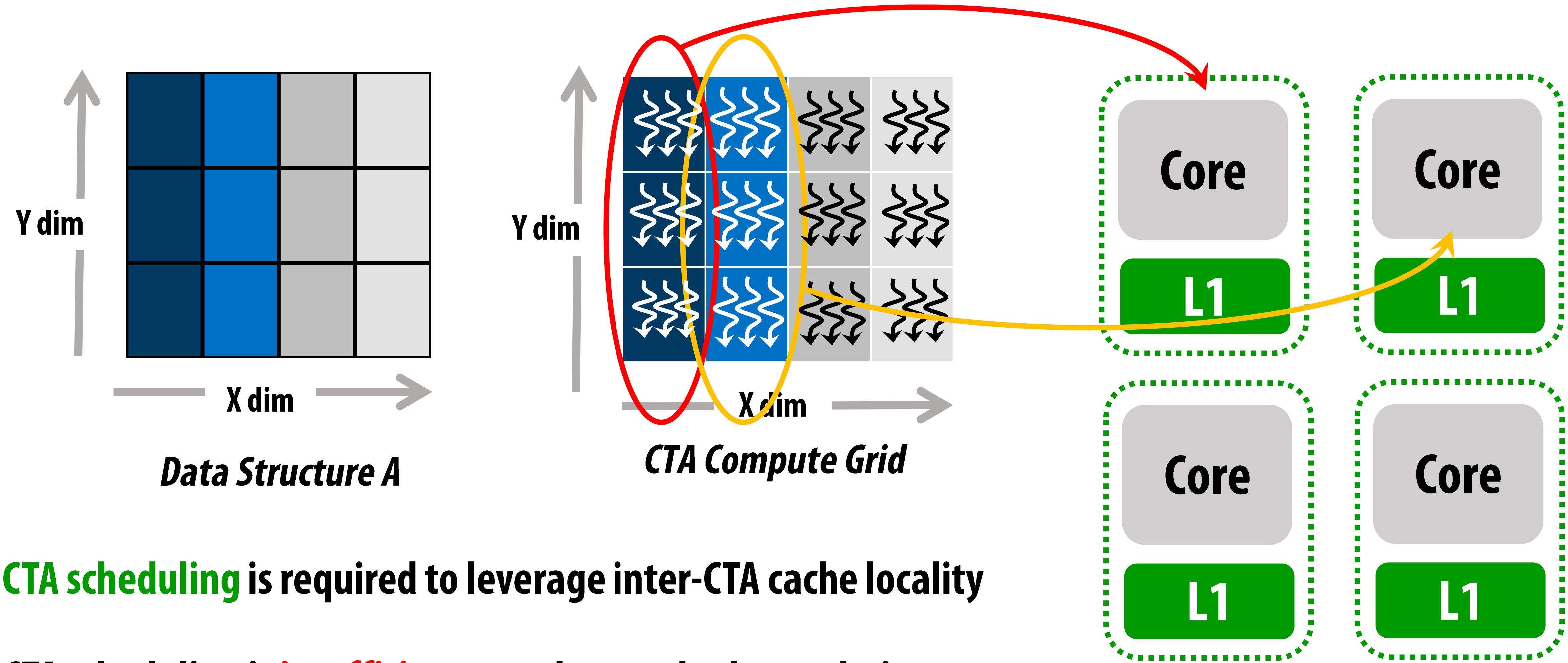
But there is no explicit way to express data locality

Exploiting data locality in GPUs is a challenging and elusive feat

A case study in leveraging data locality: `Histo`



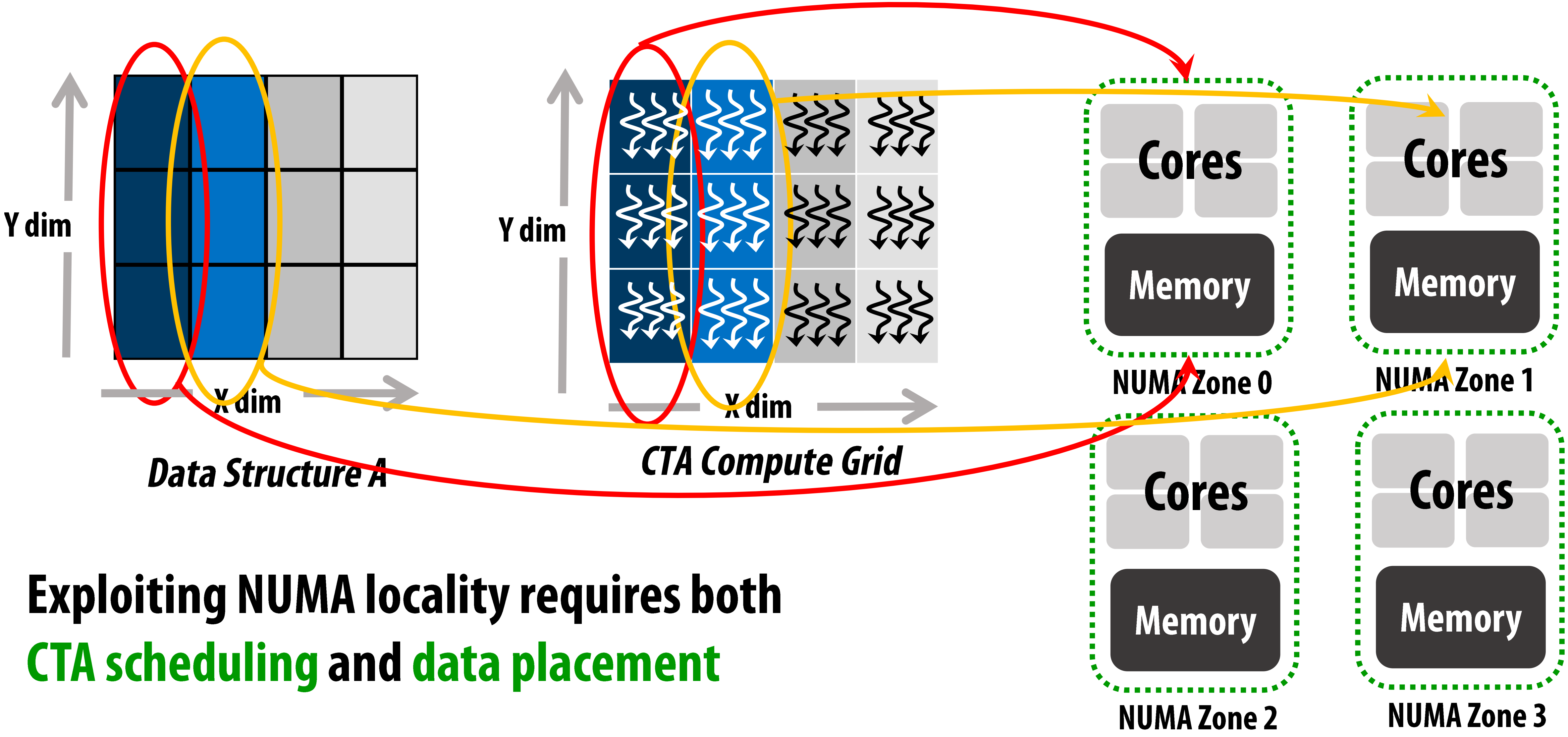
Leveraging cache locality



CTA scheduling is required to leverage inter-CTA cache locality

CTA scheduling is insufficient: we also need other techniques

Leveraging NUMA locality



Exploiting NUMA locality requires both CTA scheduling and data placement

Today, leveraging data locality is challenging

As a programmer:

- **No easy access** to architectural techniques – CTA scheduling, cache management, data placement, etc.
- Even when using work-arounds, optimization is **tedious** and **not portable**

As the architect:

- **Key program semantics** are not available to the hardware

Where to place data?

Which CTAs to schedule together?

To make things worse:

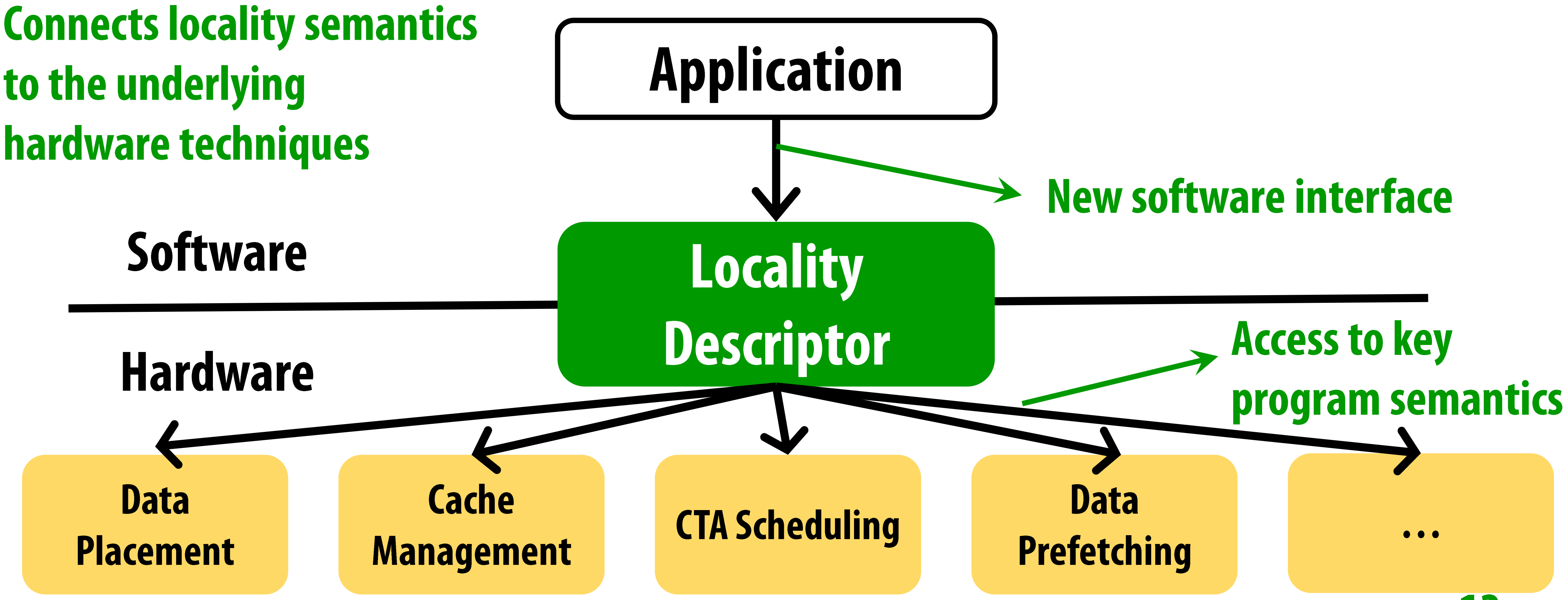
There are many different locality types: Inter-CTA, inter-warp, intra-thread, …

Each type requires a different set of architectural techniques:

- **Inter-CTA locality requires CTA scheduling + prefetching**
- **Intra-thread locality requires cache management**
- **…**

The Locality Descriptor

A hardware-software abstraction to express and exploit data locality

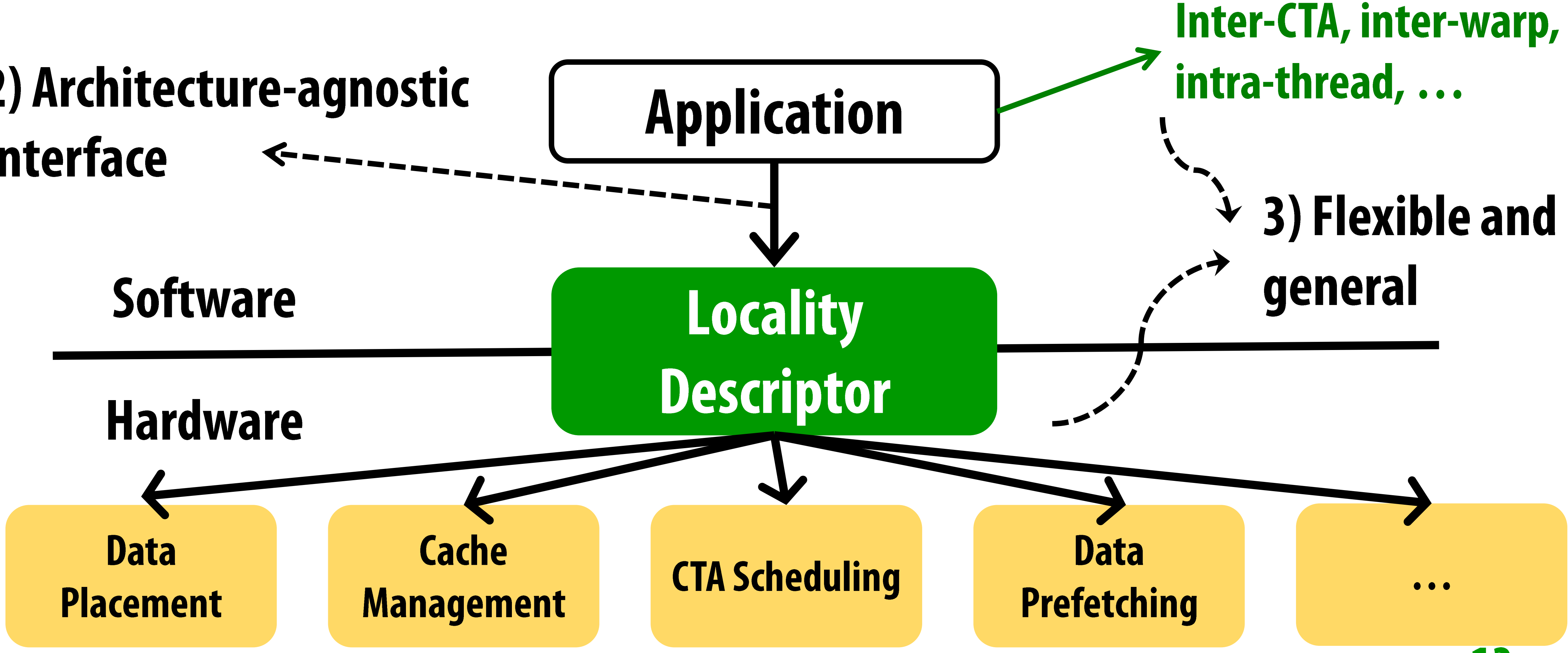


Goals in designing the Locality Descriptor

1) Supplemental and hint-based

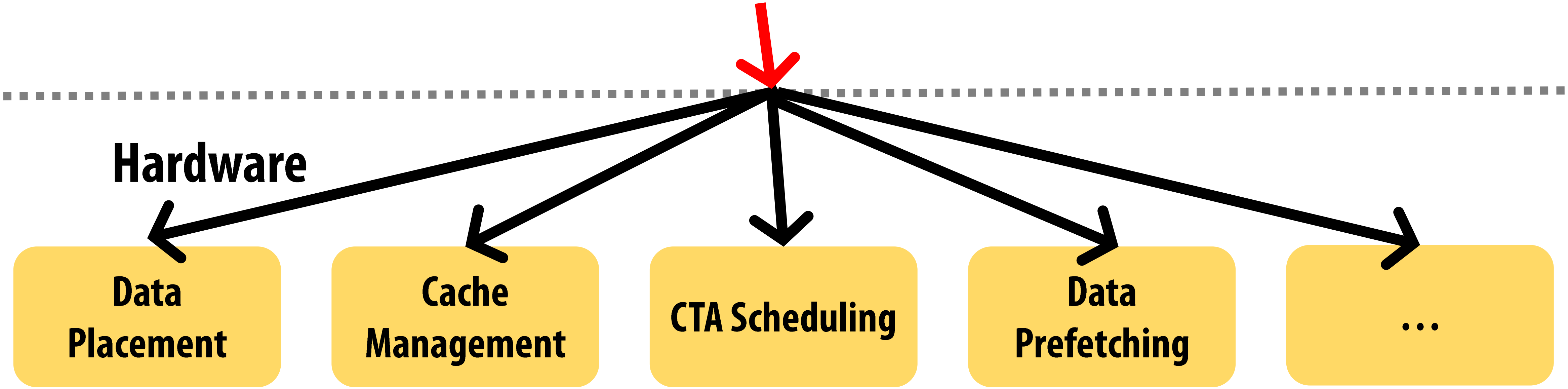
2) Architecture-agnostic interface

3) Flexible and general



Designing the Locality Descriptor

```
LocalityDescriptor ldesc(X; Y, Z);
```



An Overview: The components of the Locality Descriptor

1) Data Structure

```
LocalityDescriptor ldesc(A, len,
                   INTER-THREAD,
                           tile,
                           loc);
```

3) Tile Semantics

2) Locality Type

4) Locality Semantics

Outline

Why leveraging data locality is challenging?

Designing the Locality Descriptor

Evaluation

1. How to choose the basis of the abstraction?

Key Idea: Use the data structure as the basis to describe data locality

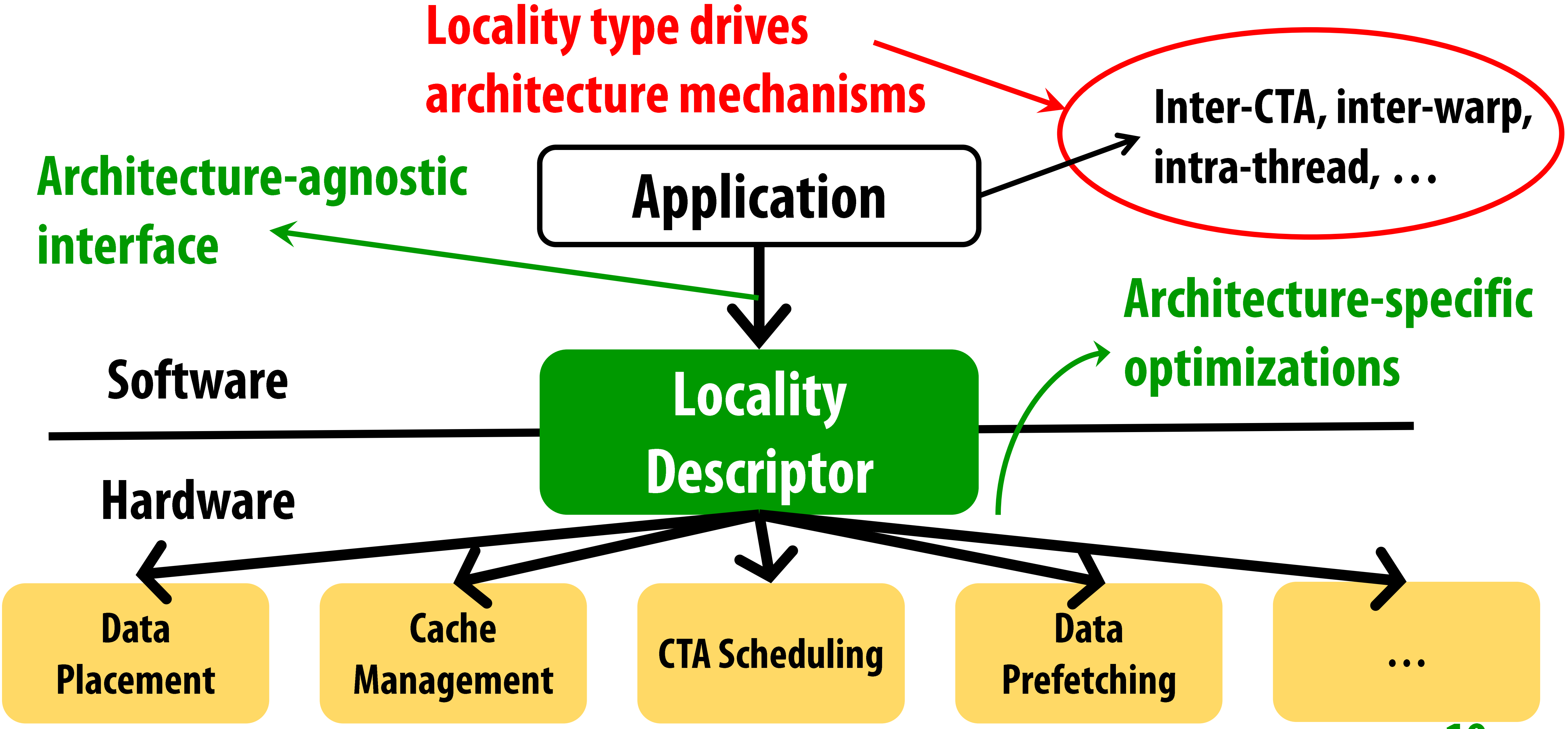
- Architecture-agnostic
- Each data structure is accessed the same way by all threads

A new instance is required for each important data structure

```
LocalityDescriptor ldesc(A);
```

→ *Data Structure*

2. How to communicate with hardware?



2. How to communicate with hardware?

Key Idea: Use locality type to drive underlying architectural techniques

Origin of locality (or locality type) causes the challenges in exploiting it

E.g.:

Inter-CTA locality requires CTA scheduling as reuse is across threads
Intra-thread locality requires cache management to avoid thrashing

Locality type is application-specific and known to the programmer

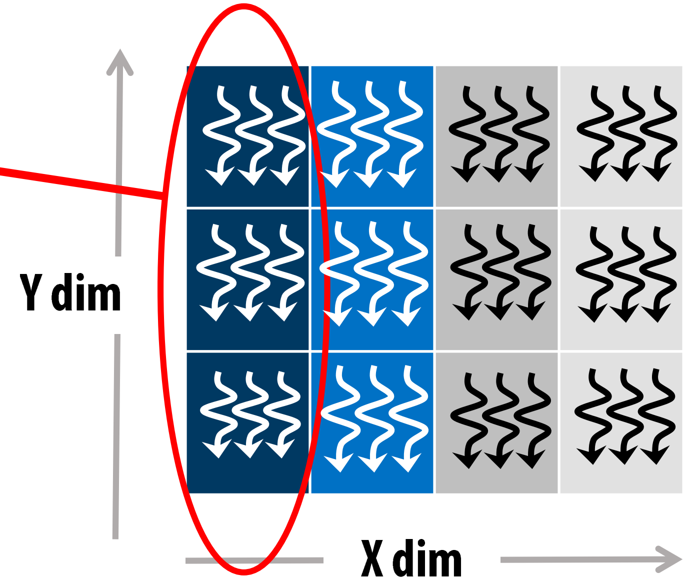
2. How to communicate with hardware?

Key Idea: Use <u>locality type</u> to drive underlying architectural techniques

Three fundamental types:

- INTER-THREAD
- INTRA-THREAD
- NO-REUSE

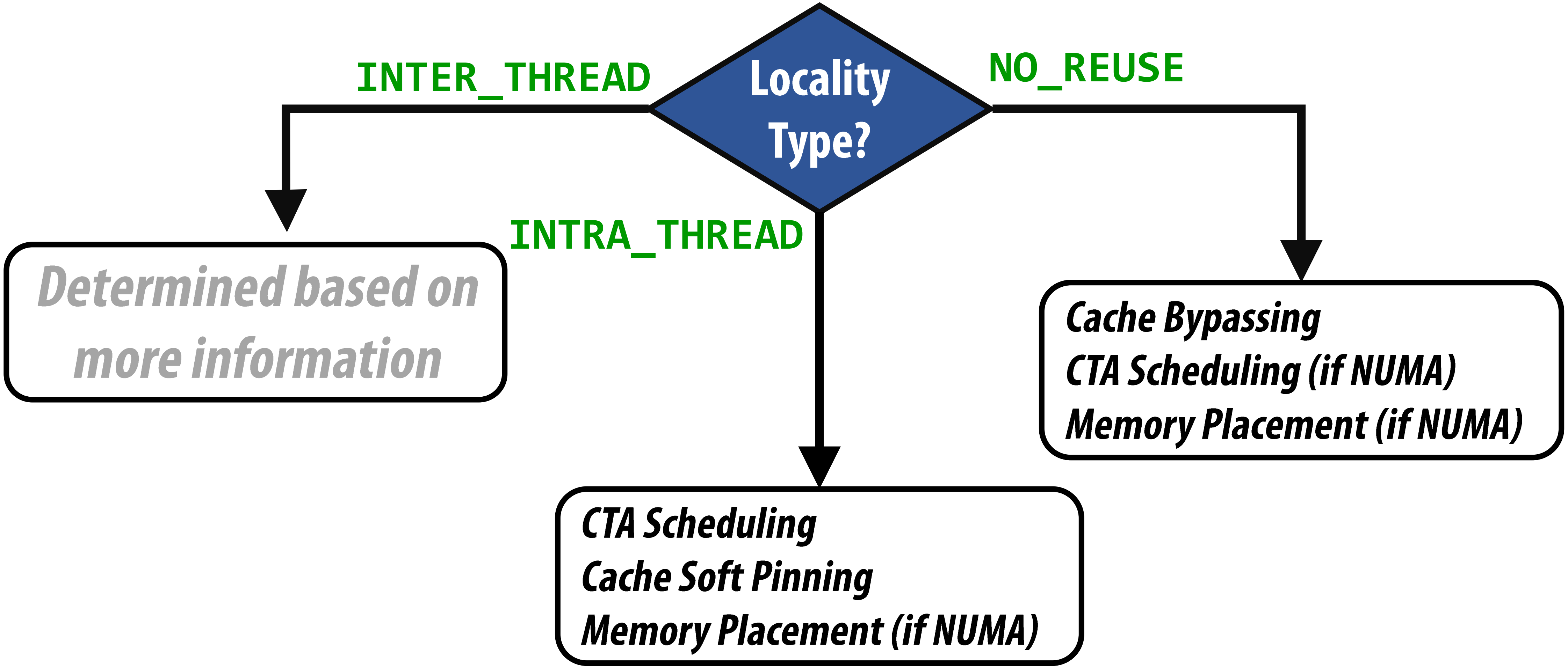
INTER-THREAD



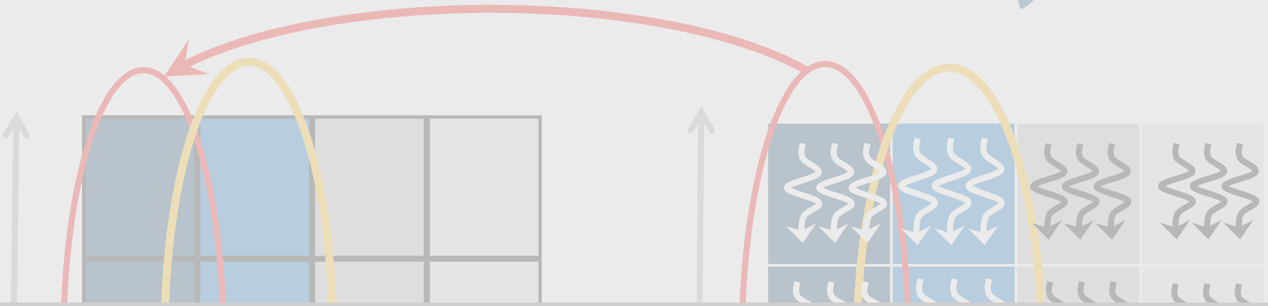
CTA Compute Grid

```
LocalityDescriptor ldesc(A);INTER-THREAD);
```

Driving underlying architectural techniques



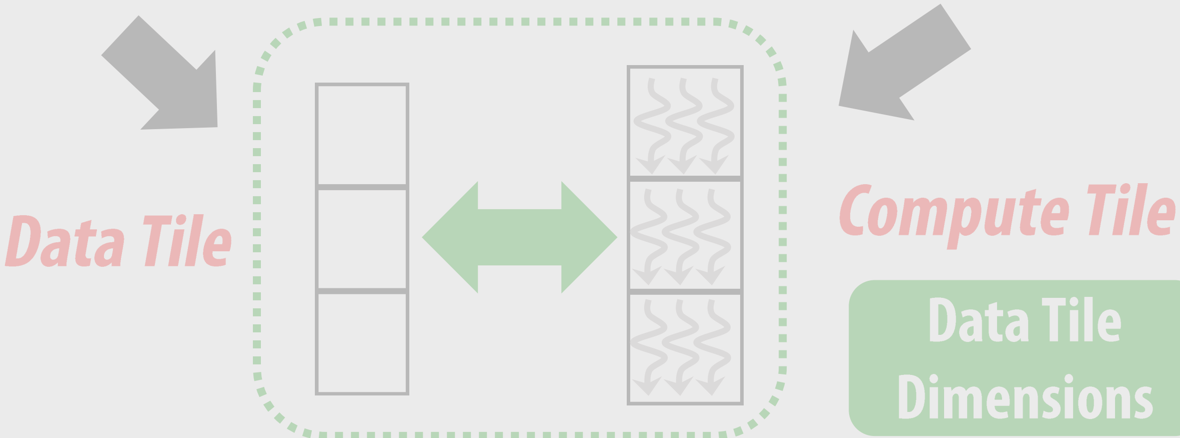
3. How to describe locality?



Key Idea: Partition the data structure and compute grid into tiles

Basic unit of locality:

```
LocalityDescriptor ldesc(A, INTER-THREAD);tile);
```




```
tile((X_tile, Y_len, 1),
     (1, GridSize.y, 1),
     (1, 0, 0));
```

Data Tile Dimensions

Compute Tile Dimensions

Compute-Data Map

Additional features of the Locality Descriptor

Locality type insufficient to inform underlying architectural techniques

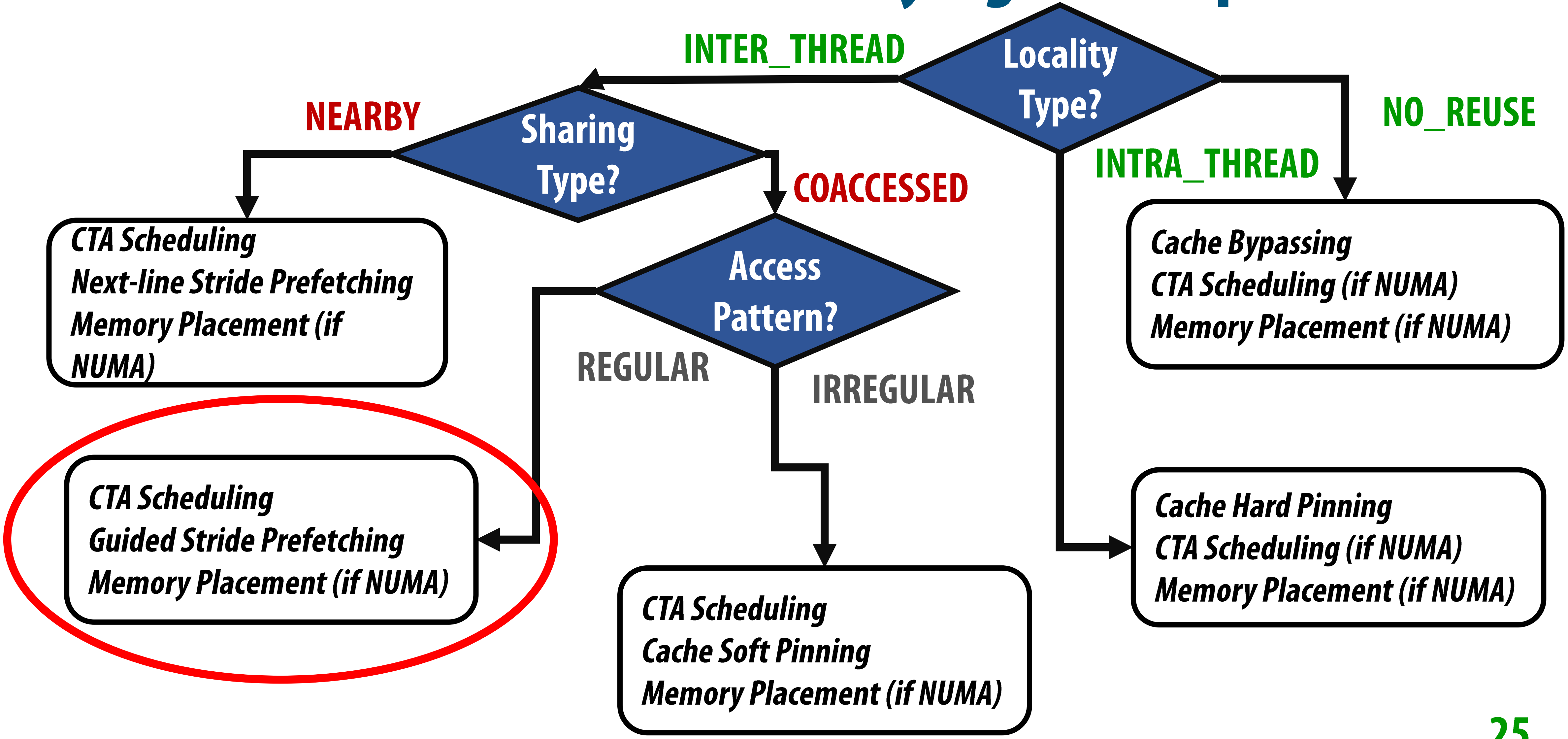
(INTER-THREAD, INTRA-THREAD, NO-REUSE)

In addition, we also have Locality Semantics to include:

- Sharing Type
- Access Pattern

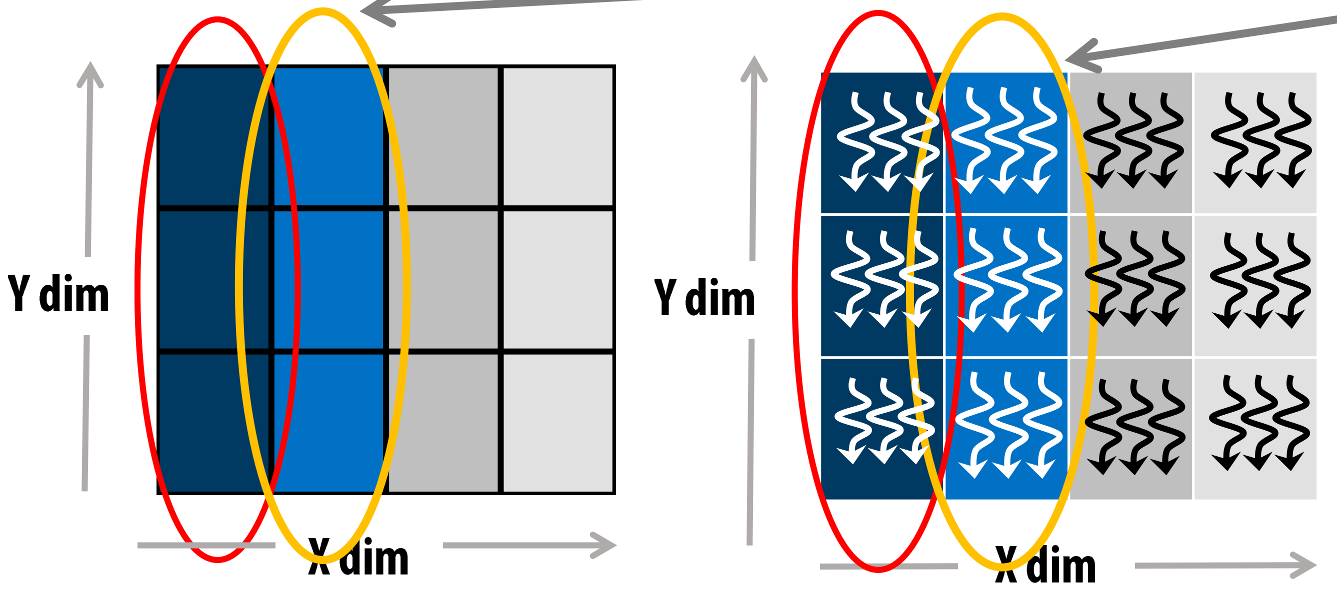
```
(COACCESSED, REGULAR, X_len)

LocalityDescriptor ldesc(A, INTER-THREAD, tile);loc);
```

A decision tree to drive underlying techniques



Leveraging the Locality Descriptor

```
LocalityDescriptor ldesc(A, INTER-THREAD, tile, loc);
```



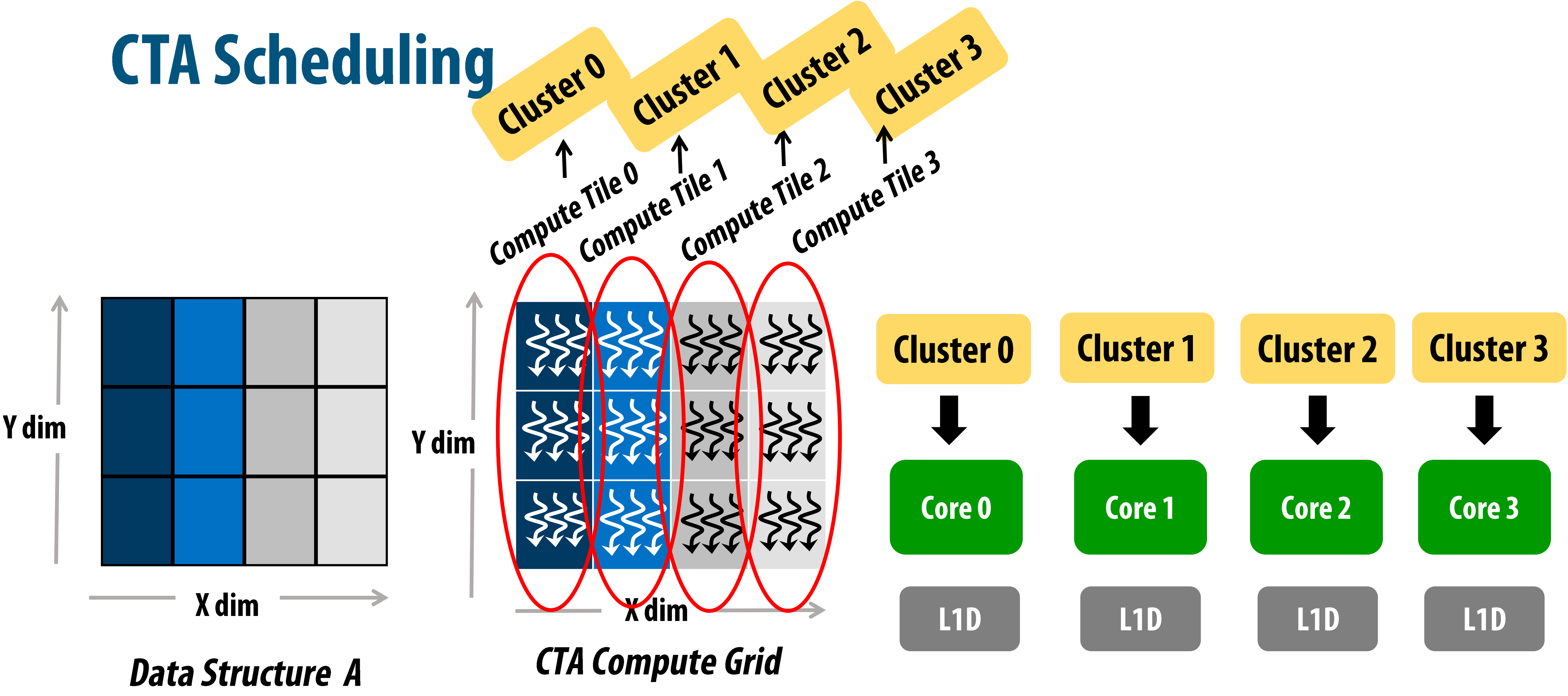
Data Structure A

CTA Compute Grid

Architectural techniques:

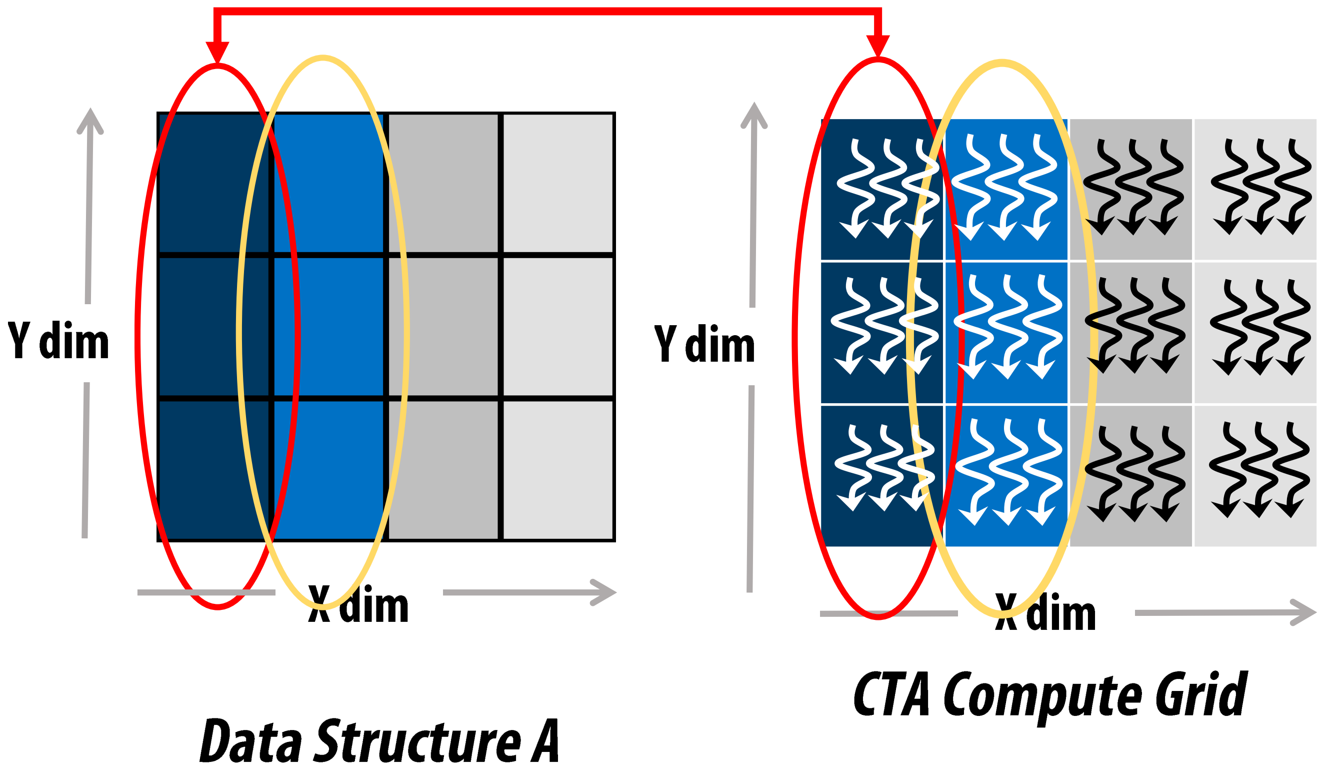
1) CTA Scheduling
2) Prefetching
3) Data Placement

CTA Scheduling



Leveraging the Locality Descriptor

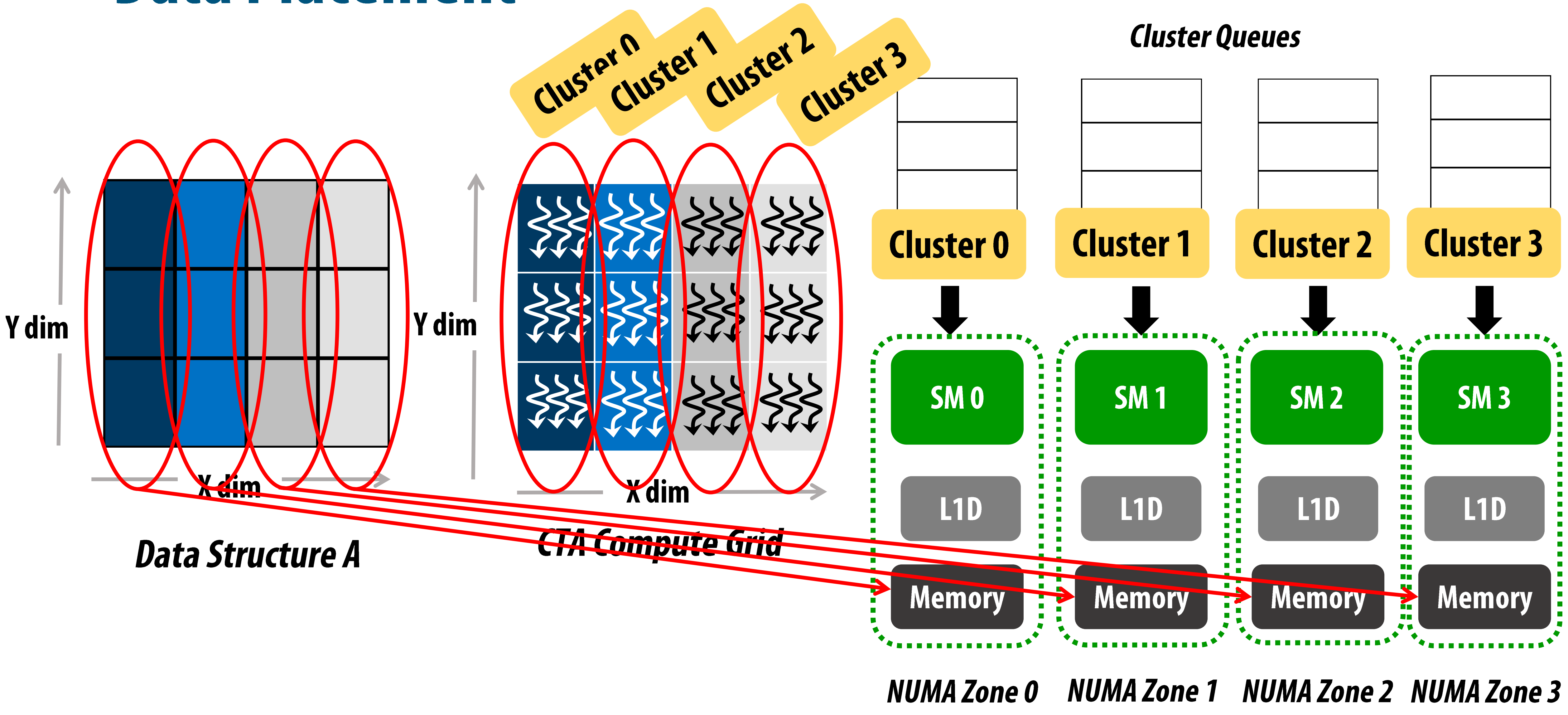
```
LocalityDescriptor ldesc(A, INTER-THREAD, tile, loc);
```



Architectural techniques:

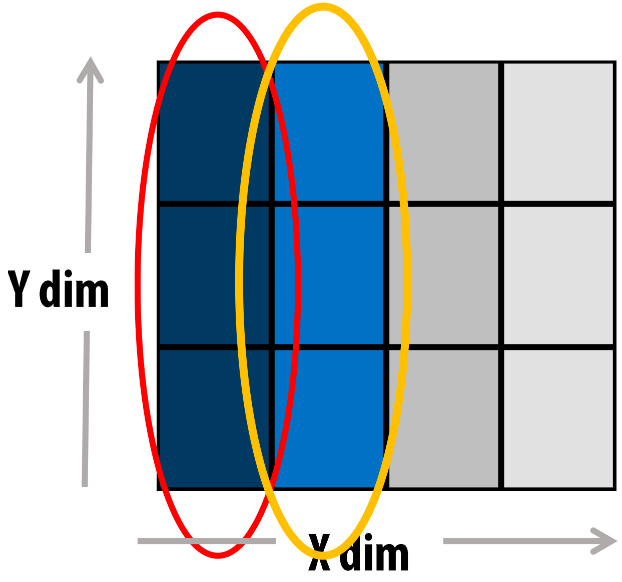
1) CTA Scheduling
2) Prefetching
3) Data Placement

Data Placement

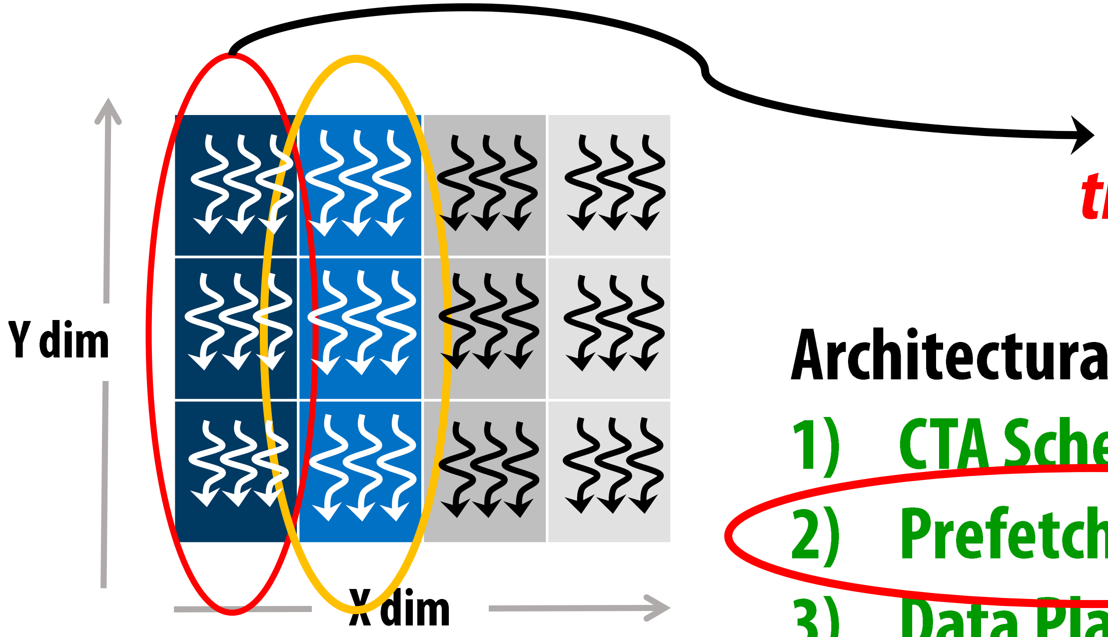


Leveraging the Locality Descriptor

```
LocalityDescriptor ldesc(A, INTER-THREAD, tile, loc);
```



Data Structure A



CTA Compute Grid

All threads stall because they wait on the same data

Architectural techniques:

1) CTA Scheduling
2) Prefetching
3) Data Placement

Outline

Why leveraging data locality is challenging?

The Locality Descriptor

Evaluation

Methodology

Evaluation Infrastructure: GPGPUSim v3.2.2

Workloads: Parboil, Rodinia, CUDA SDK, Polybench

System Parameters:

Shader Core: 1.4 GHz; GTO scheduler [50]; 2 schedulers per SM, Round-robin CTA scheduler

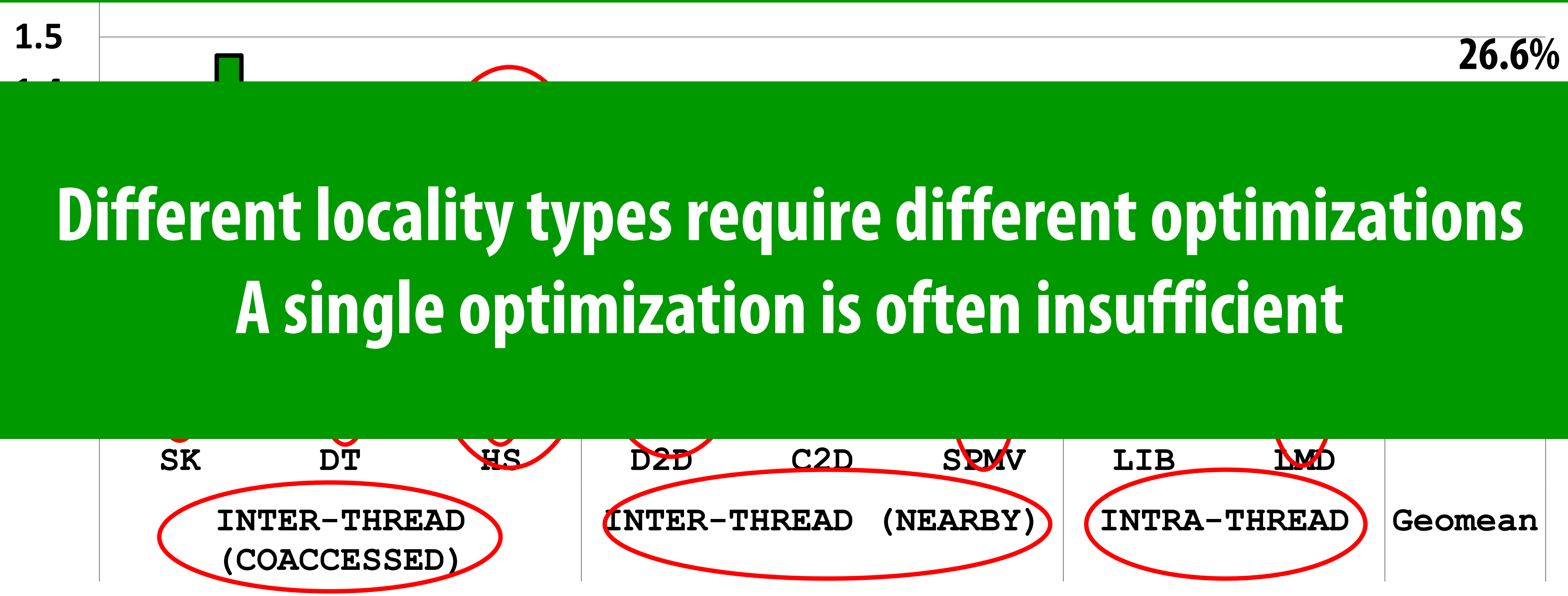
SM Resources Registers: 32768; Scratchpad: 48KB, L1: 32KB, 4 ways

Memory Model: FR-FCFS scheduling [59, 60], 16 banks/channel

Single Chip System: 15 SMs; 6 memory channels; L2: 768KB, 16 ways

Multi-Chip System: 4 NUMA zones, 64 SMs (16 per zone); 32 memory channels; L2: 4MB, 16 ways; Inter-GPM Interconnect: 192 GB/s; DRAM Bandwidth: 768 GB/s (192 GB/s per module)

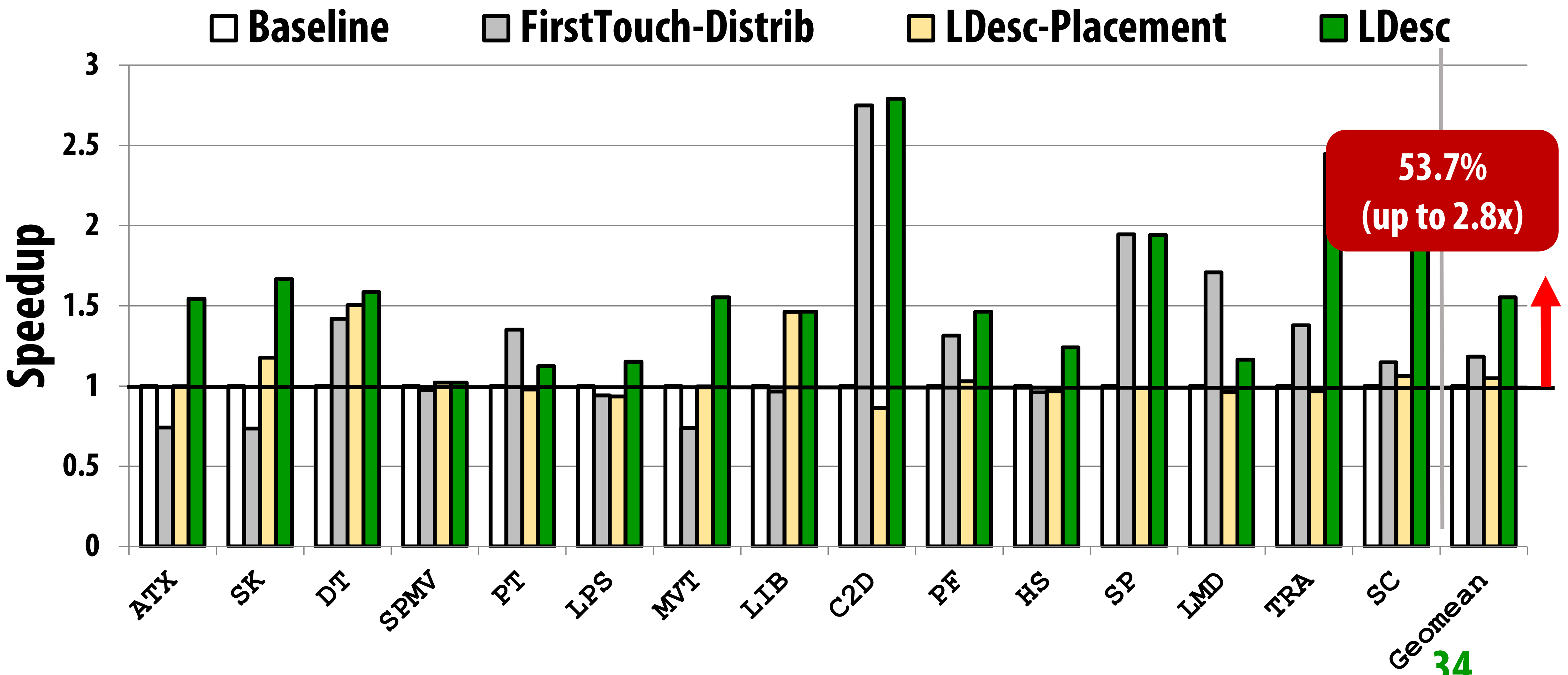
Locality descriptors are an effective means to leverage cache locality



Different locality types require different optimizations

A single optimization is often insufficient

Performance Impact: Leveraging NUMA Locality



Conclusion

Problem:

- GPU programming models have no explicit abstraction to express data locality
- Leveraging data locality is a challenging task, as a result

Our Proposal: The Locality Descriptor

- A HW-SW abstraction to explicitly express data locality
- A architecture-agnostic and flexible SW interface to express data locality
- Enables HW to leverage key program semantics to optimize locality

Key Results:

- 26.6% (up to 46.6%) performance speed up from leveraging cache locality
- 53.7% (up to 2.8x) performance speed up from leveraging NUMA locality

The Locality Descriptor

A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs

Nandita Vijaykumar

Eiman Ebrahimi, Kevin Hsieh, Phillip B. Gibbons, Onur Mutlu

Carnegie Mellon University

NVIDIA

ETH Zürich