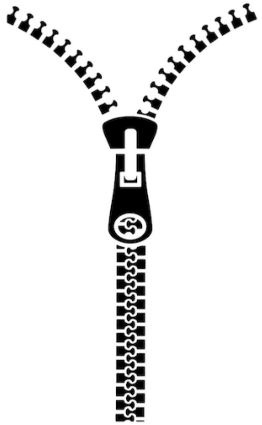
Zipper: Latency-Tolerant Optimizations for High-Performance Buses

Shibo Chen[†] Hailun Zhang[‡] Todd Austin[†]

[†] University of Michigan – Ann Arbor
[‡] University of Wisconsin – Madison

ASIA SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE



Compute Offload Overhead

Compute Process

Memory Access

Host-Accelerator Communication

CPU/Host

Shared Memory

Accelerator

Compute Offload Overhead

Compute Process

Memory Access

Host-Accelerator Communication

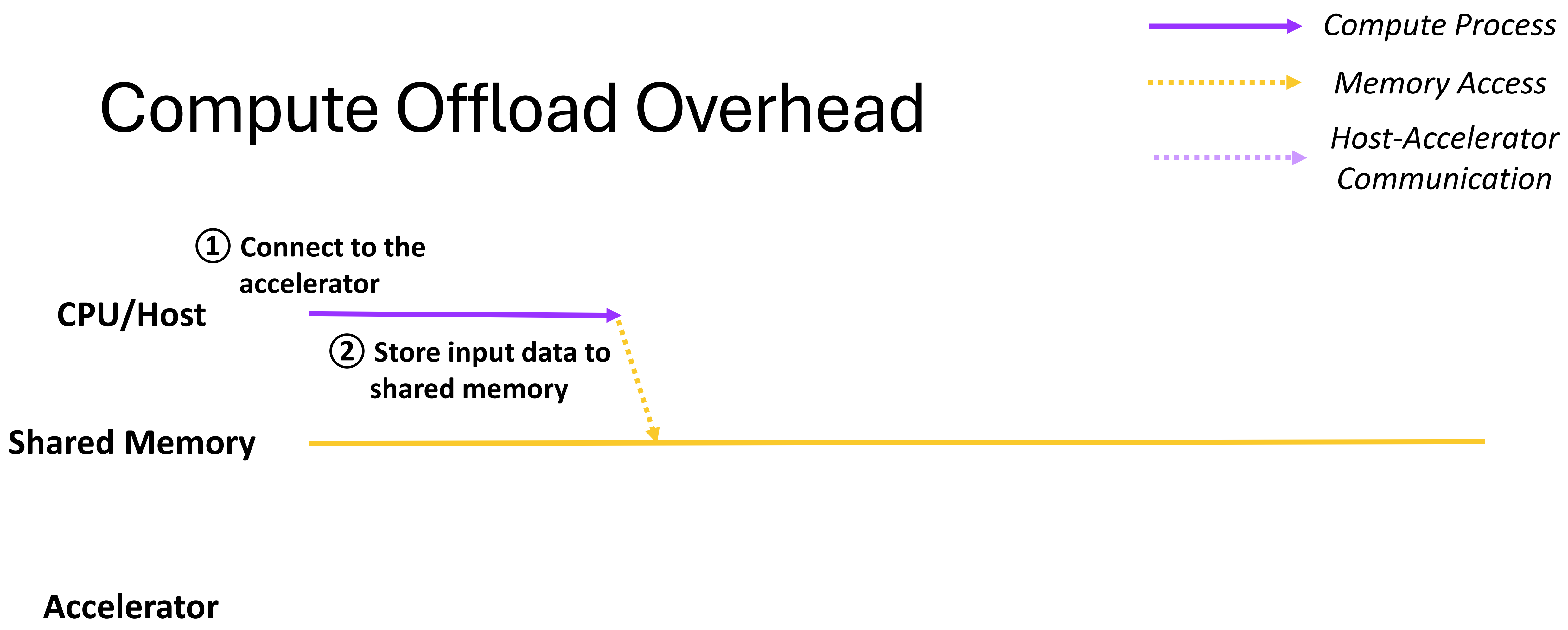
① **Connect to the accelerator**

CPU/Host

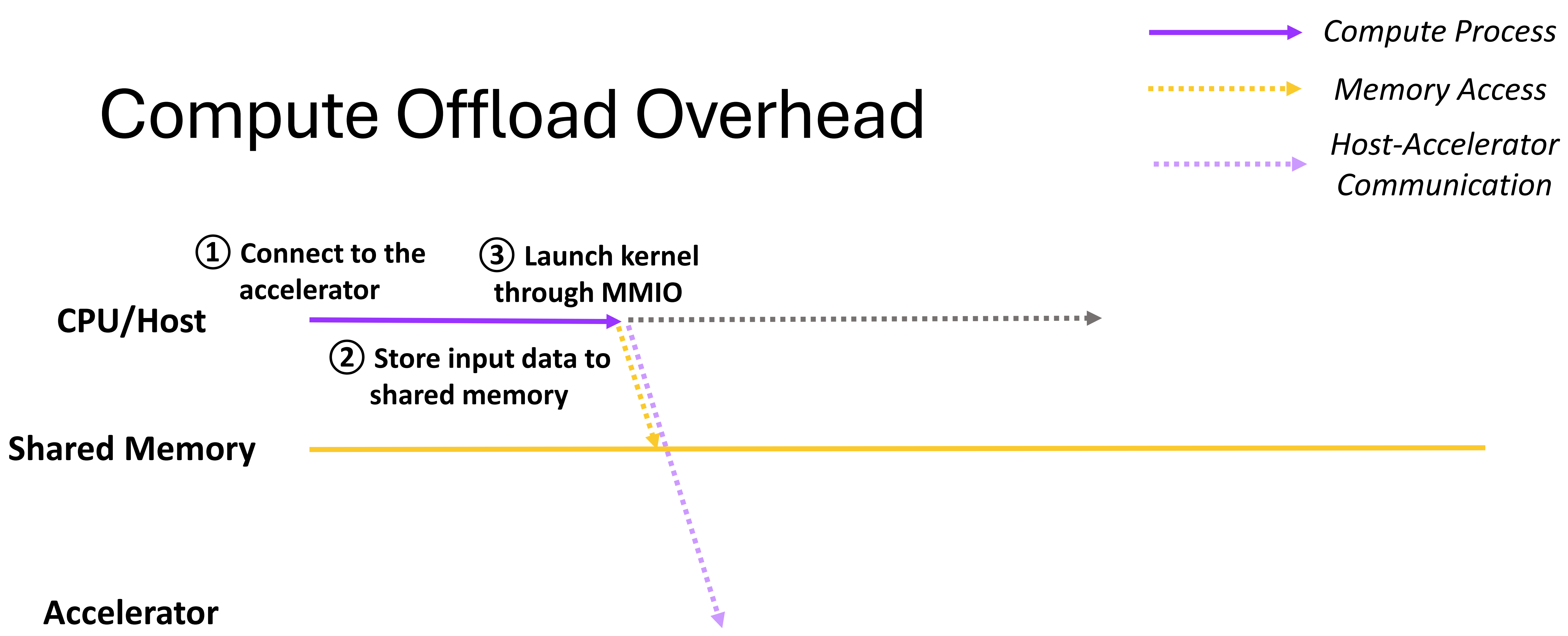
Shared Memory

Accelerator

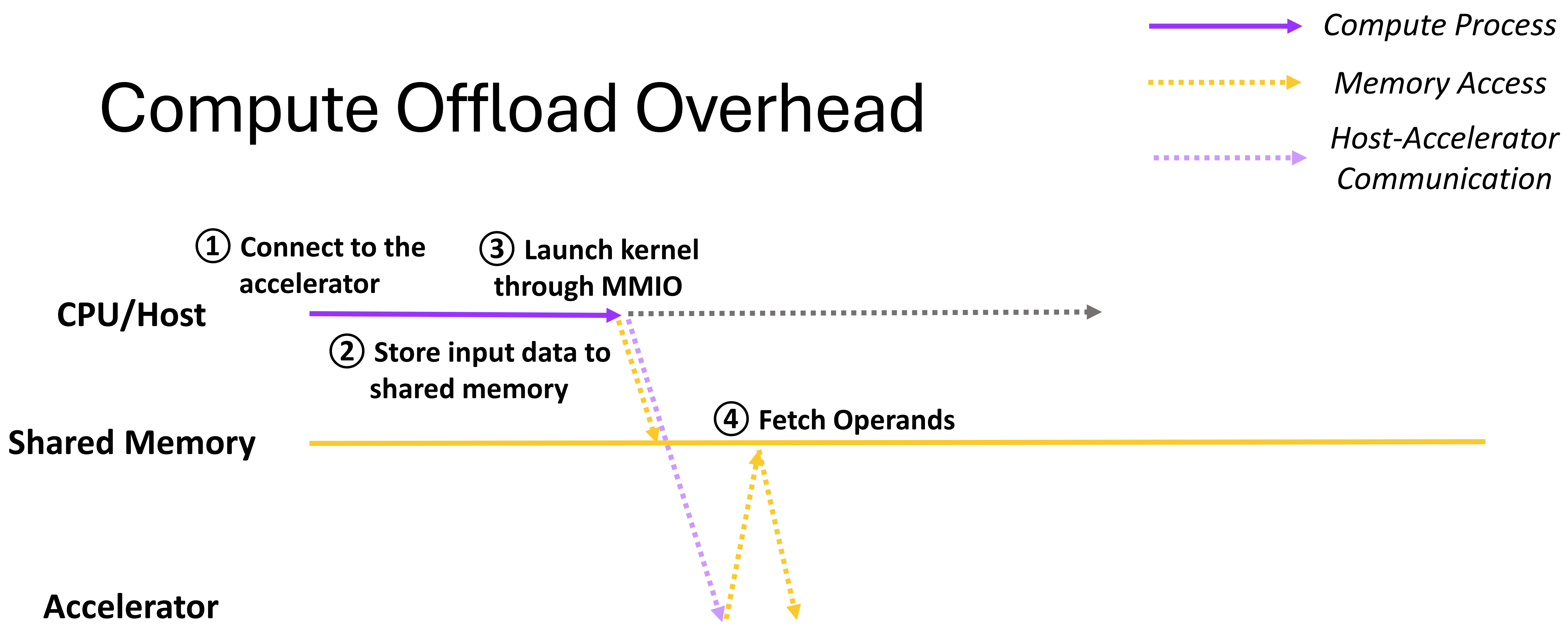
Compute Offload Overhead



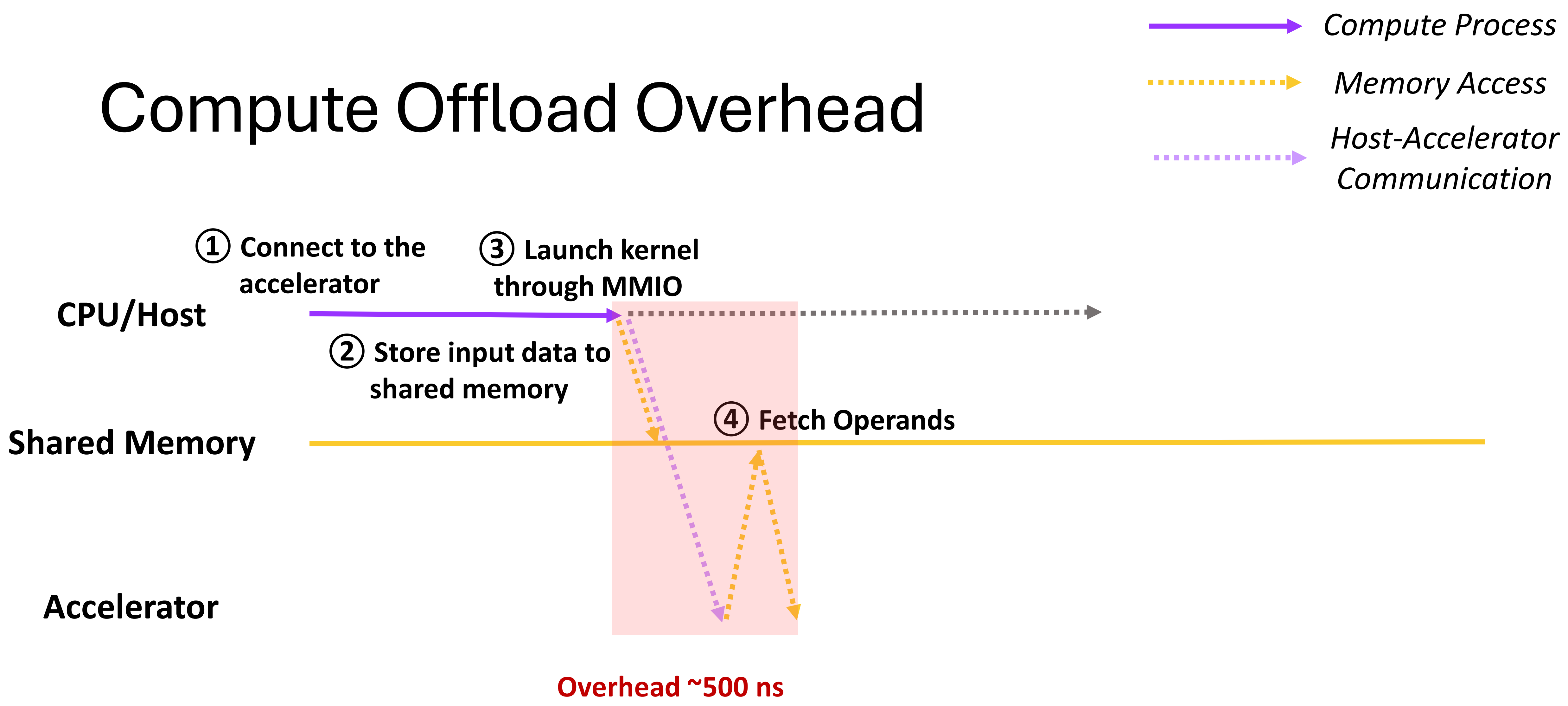
Compute Offload Overhead



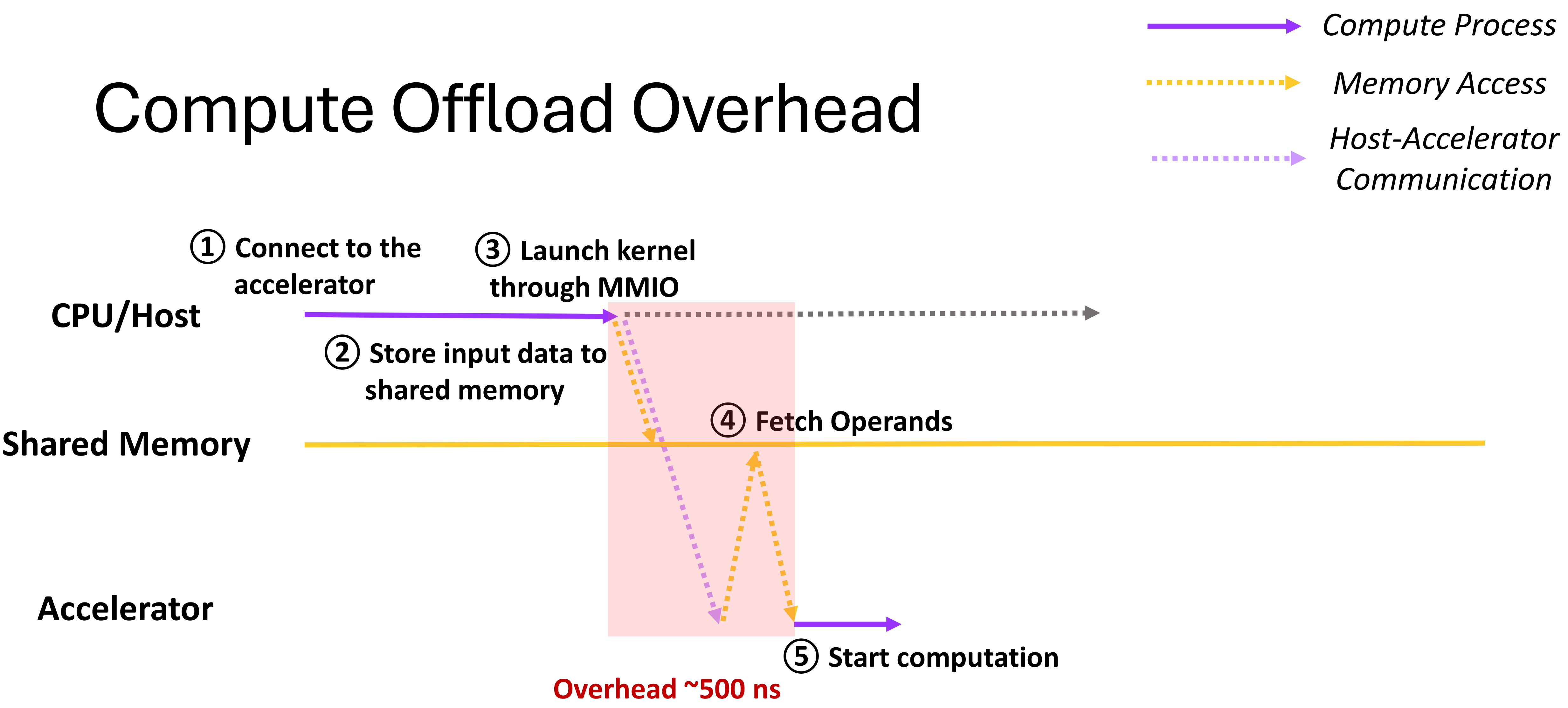
Compute Offload Overhead



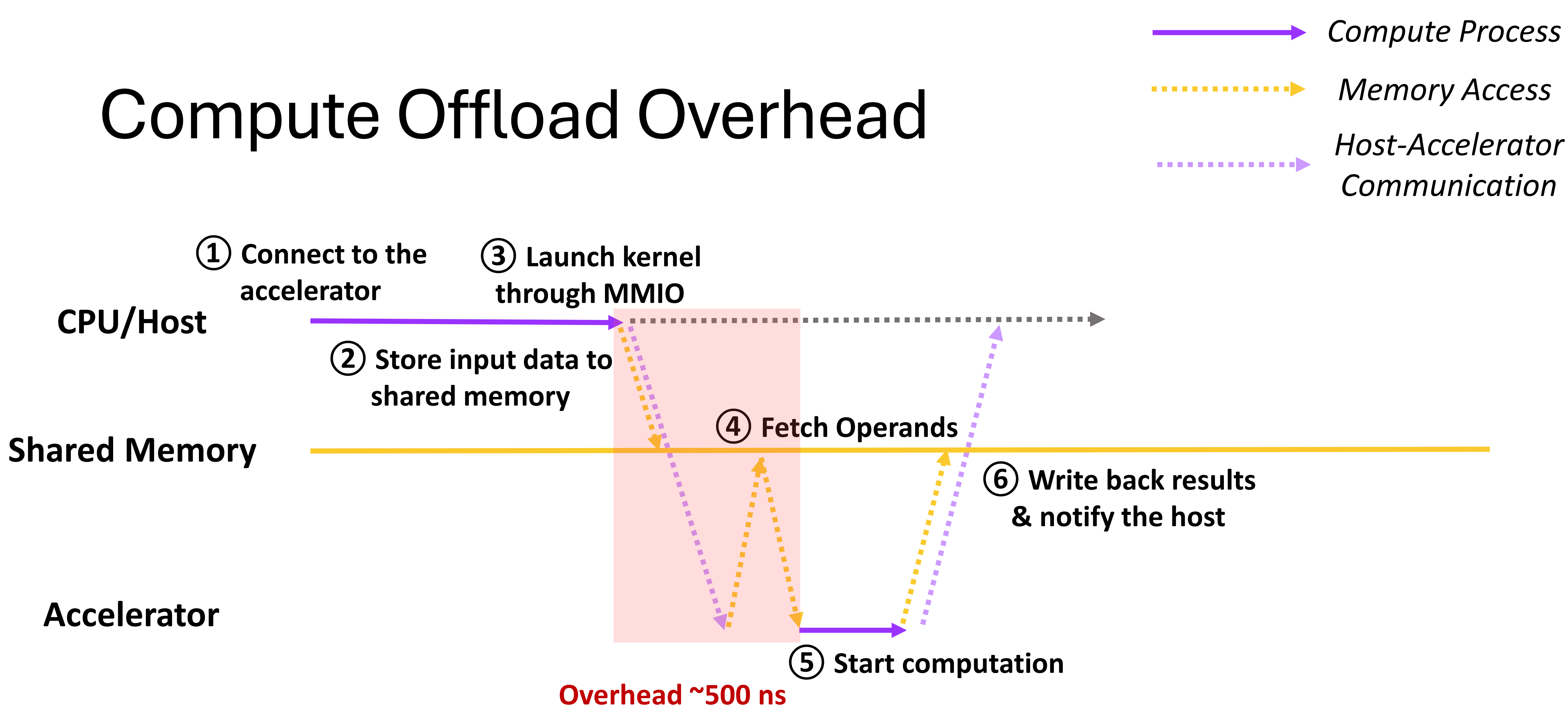
Compute Offload Overhead



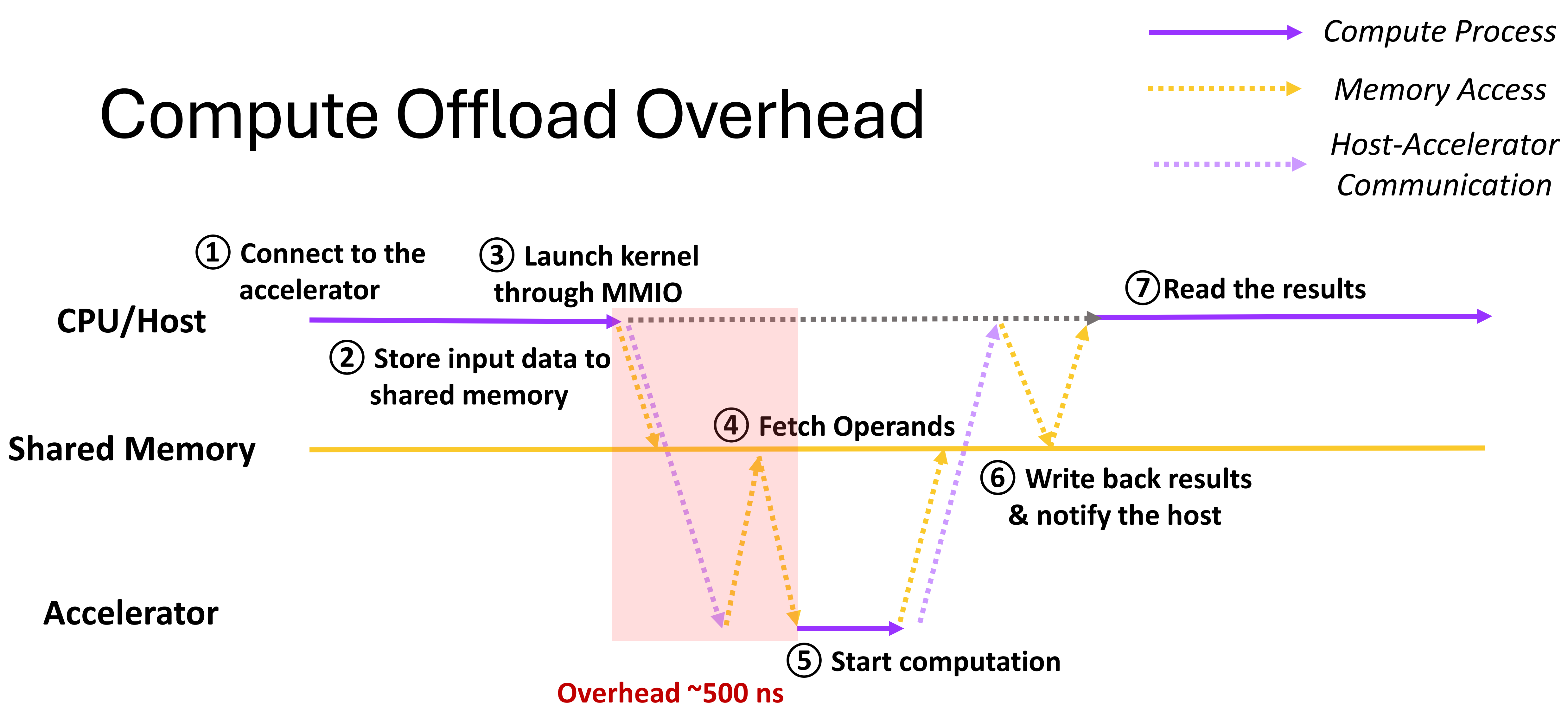
Compute Offload Overhead



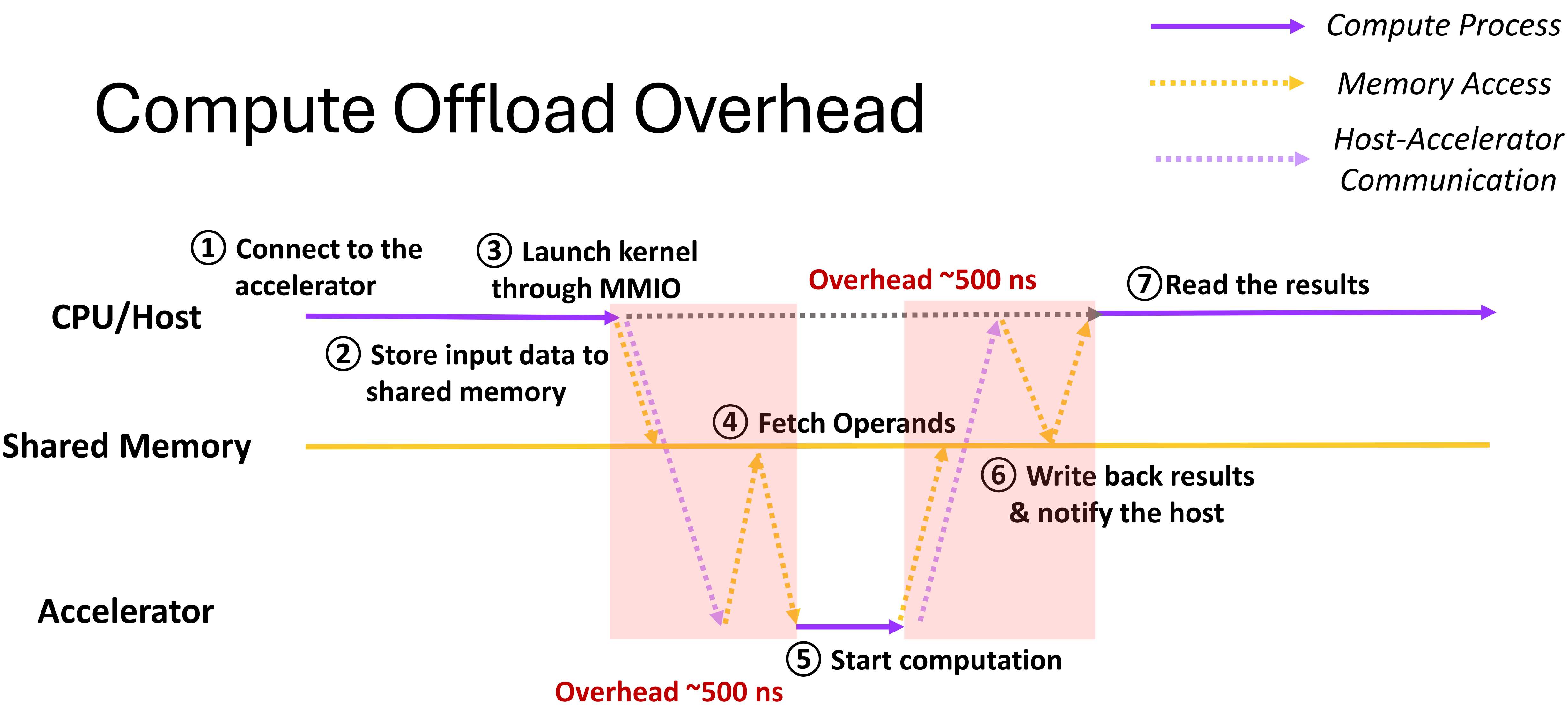
Compute Offload Overhead



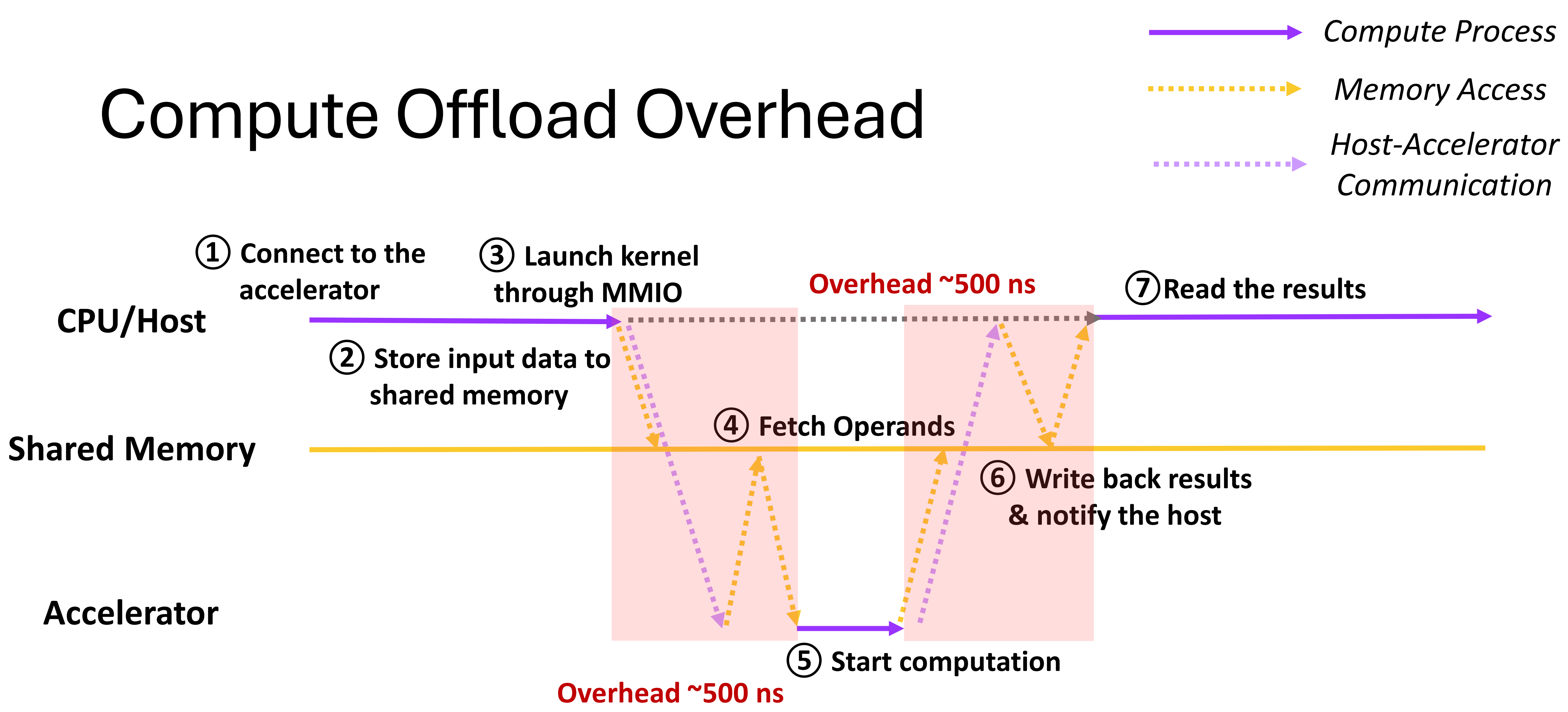
Compute Offload Overhead



Compute Offload Overhead



Compute Offload Overhead



~1000ns Overhead for Round Trip Latency

Equation for Computing Offload Trade-offs

Ratio of Raw Time Saved Over Offload Overhead $\left(\frac{\mathrm{P}}{\mathrm{O}}\right)$:

Equation for Computing Offload Trade-offs

Ratio of Raw Time Saved Over Offload Overhead $\left(\frac{P}{O}\right)$:

$$\frac{P}{O} = \frac{\quad}{\quad}$$

Equation for Computing Offload Trade-offs

Ratio of Raw Time Saved Over Offload Overhead $\left(\frac{\mathrm{P}}{\mathrm{O}}\right)$:

$$\frac{\mathrm{P}}{\mathrm{O}} = \frac{(Execution\_Time_{CPU} - Execution\_Time_{accelerator})}{}$$

Equation for Computing Offload Trade-offs

Ratio of Raw Time Saved Over Offload Overhead $\left(\frac{\mathrm{P}}{\mathrm{O}}\right)$:

$$\frac{\mathrm{P}}{\mathrm{O}} = \frac{(Execution\_Time_{CPU} - Execution\_Time_{accelerator})}{Communication\_Latency}$$

Equation for Computing Offload Trade-offs

Ratio of Raw Time Saved Over Offload Overhead $\left(\frac{\mathrm{P}}{\mathrm{O}}\right)$:

$$\frac{\mathrm{P}}{\mathrm{O}} = \frac{(Execution\_Time_{CPU} - Execution\_Time_{accelerator})}{Communication\_Latency}$$

$$= \frac{(T_{cpu} - T_{acc})}{T_{Lat}}$$

Equation for Computing Offload Trade-offs

Ratio of Raw Time Saved Over Offload Overhead $\left(\frac{P}{O}\right)$:

$$\frac{P}{O} = \frac{(Execution\_Time_{CPU} - Execution\_Time_{accelerator})}{Communication\_Latency}$$

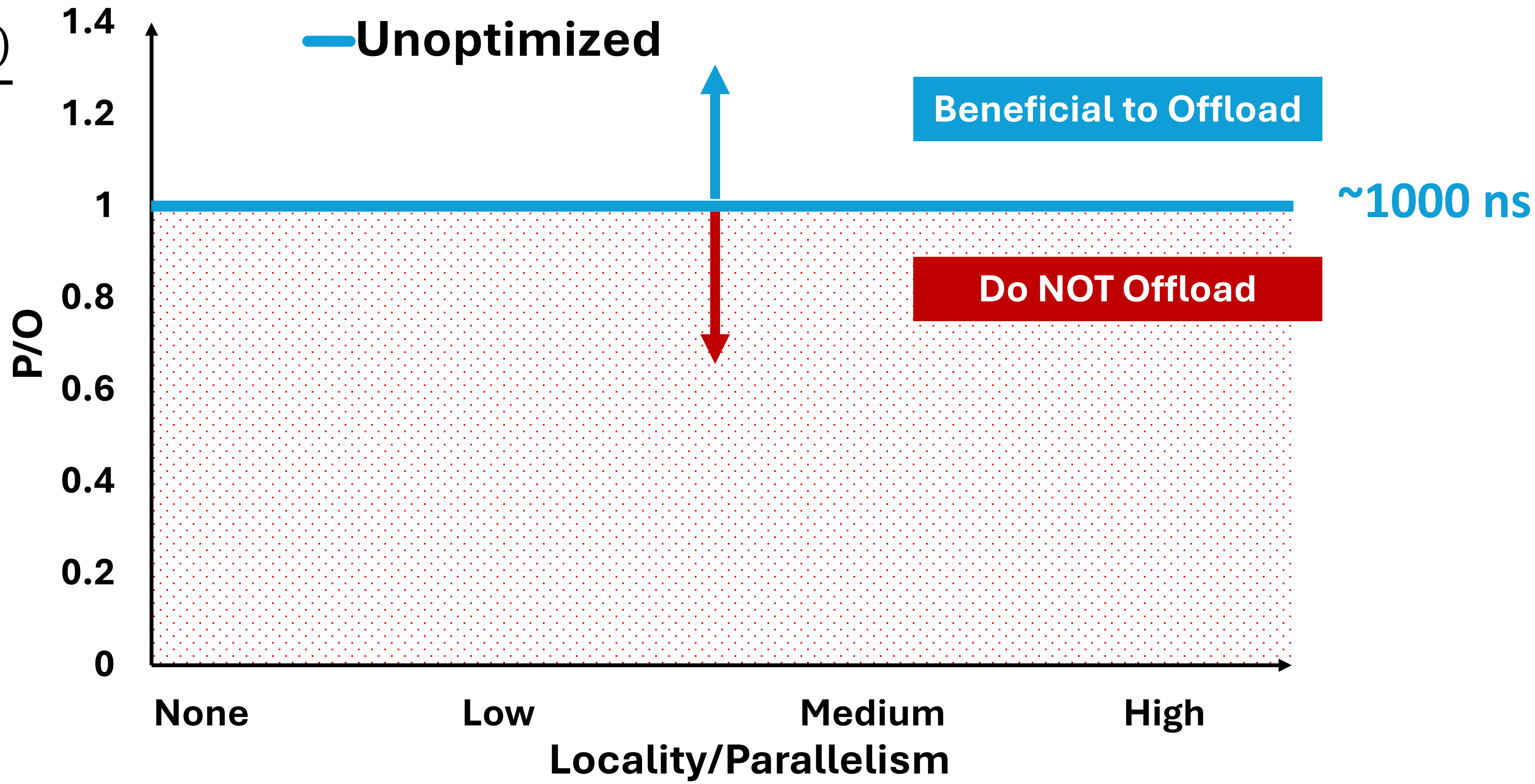
$$= \frac{(T_{cpu} - T_{acc})}{T_{Lat}}$$

$\frac{P}{O} > 1$: Beneficial to offload

$\frac{P}{O}$ <=1: Not beneficial to offload

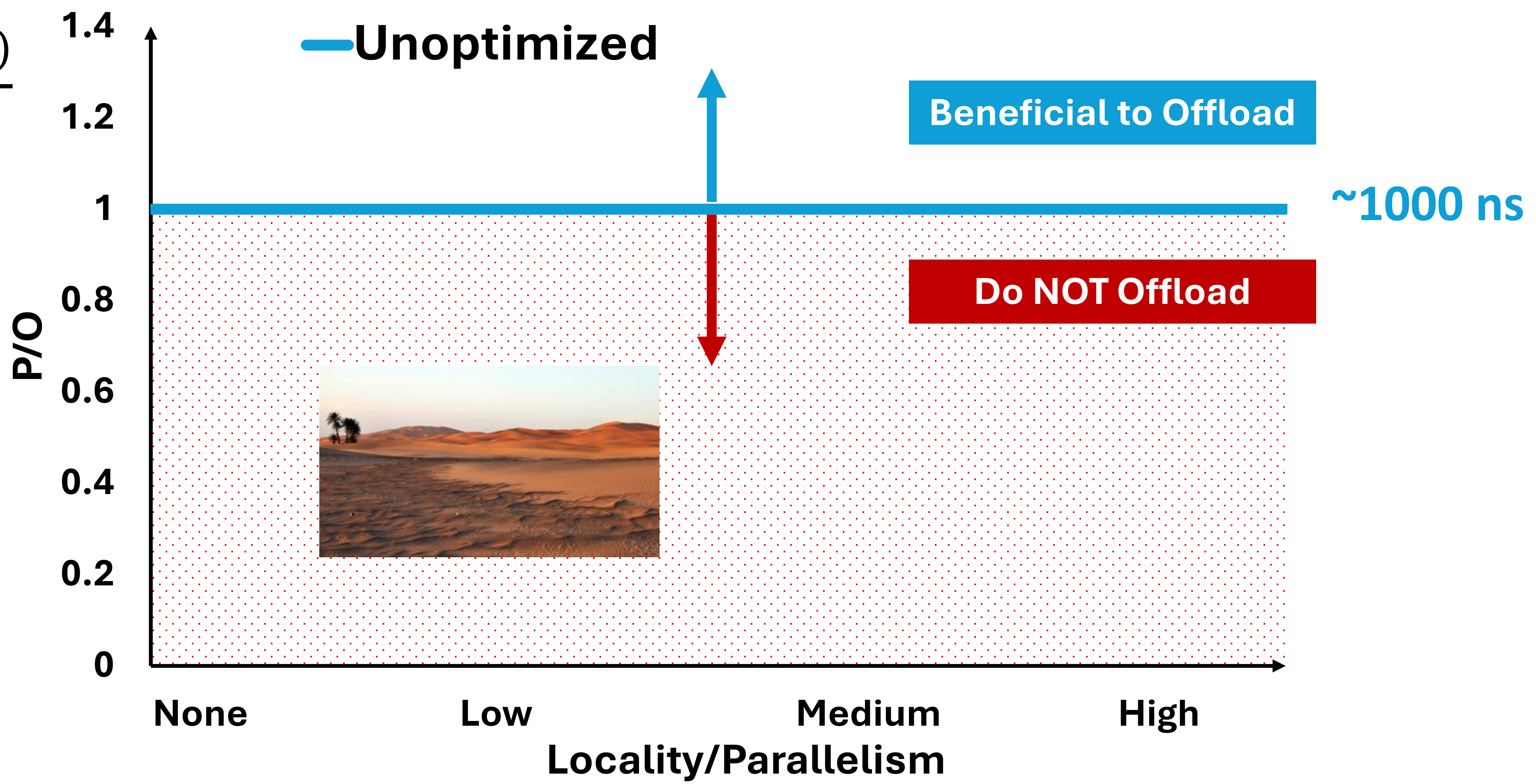
The Death Zone of Compute Offload

$$\frac{P}{O} = \frac{(T_{cpu} - T_{acc})}{T_{Lat}}$$



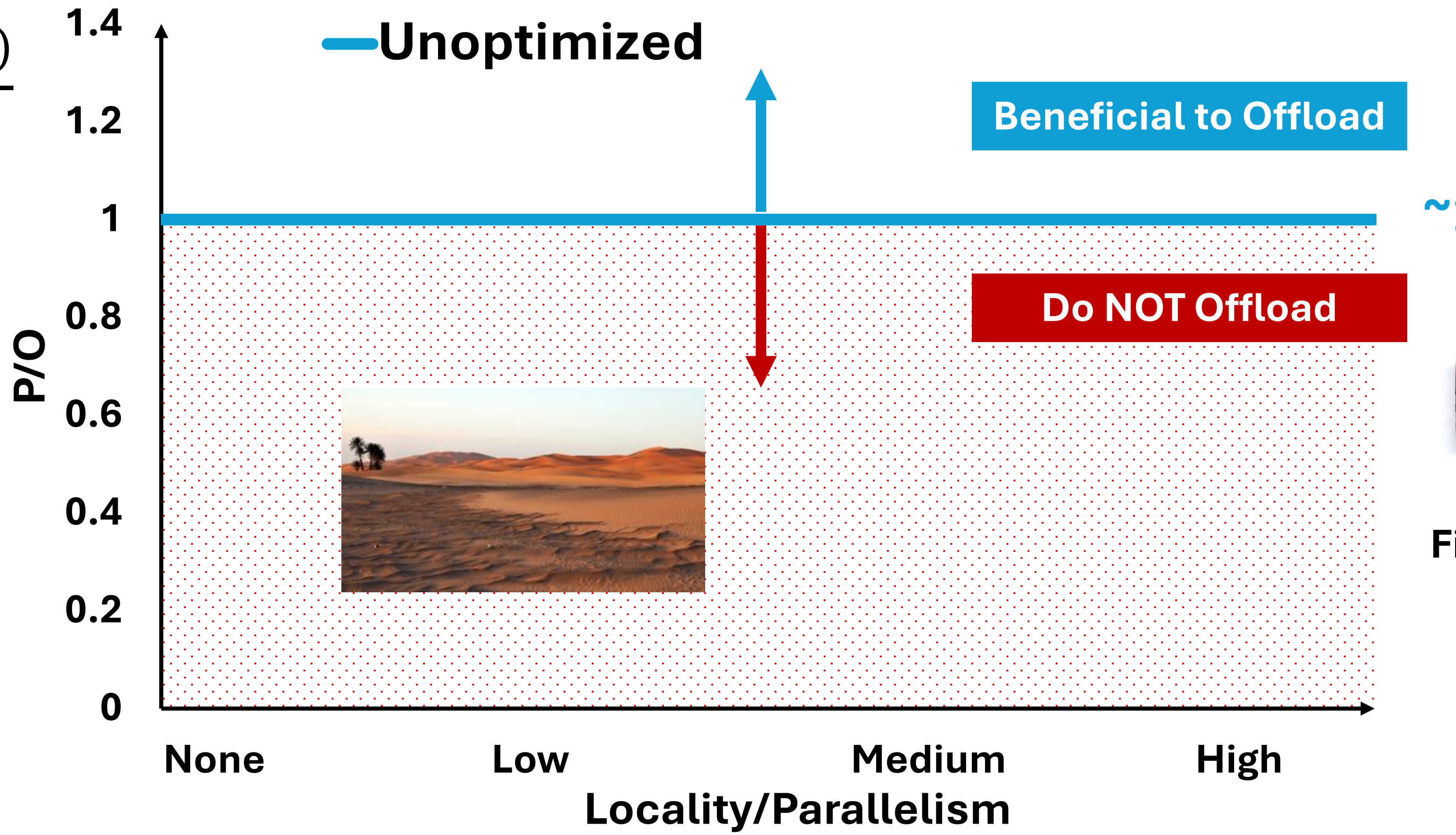
The Death Zone of Compute Offload

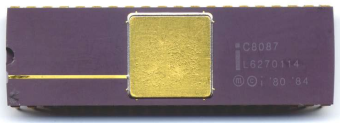
$$\frac{P}{O} = \frac{(T_{cpu} - T_{acc})}{T_{Lat}}$$



The Death Zone of Compute Offload

$$\frac{P}{O} = \frac{(T_{cpu} - T_{acc})}{T_{Lat}}$$

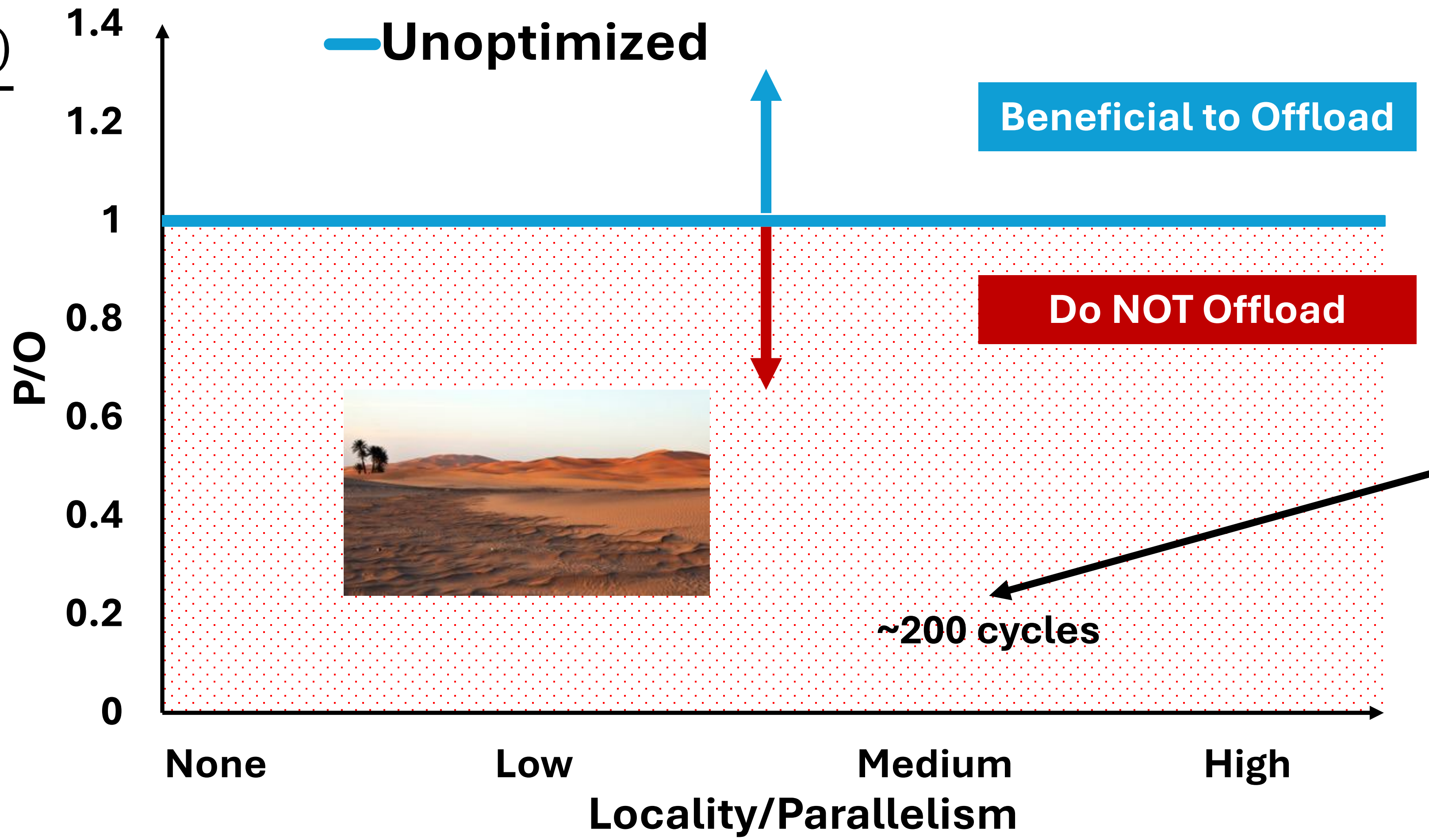


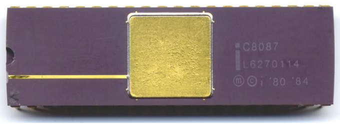


Intel 8087
First Floating Point CoProcessor

The Death Zone of Compute Offload

$$\frac{P}{O} = \frac{(T_{cpu} - T_{acc})}{T_{Lat}}$$

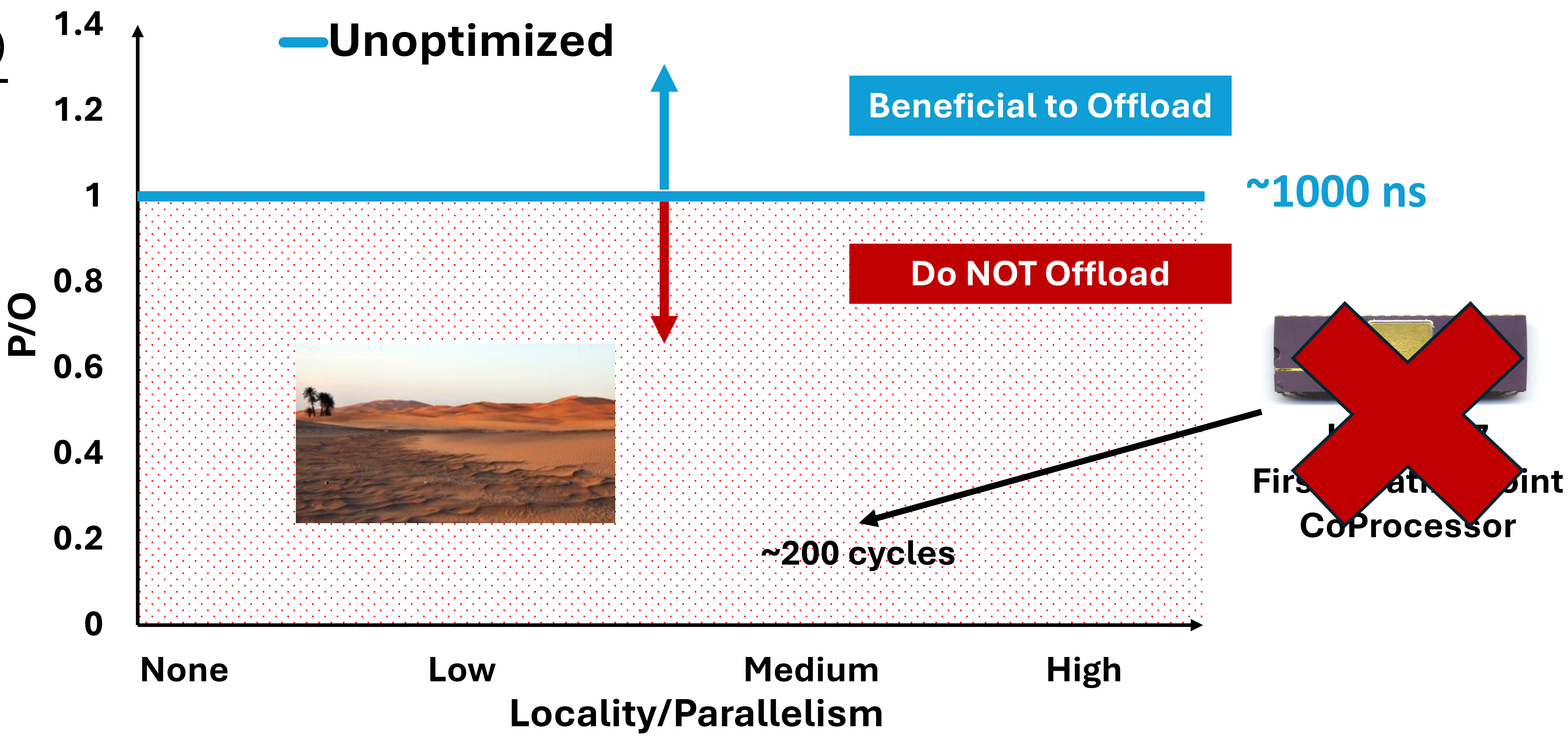




Intel 8087
First Floating Point CoProcessor

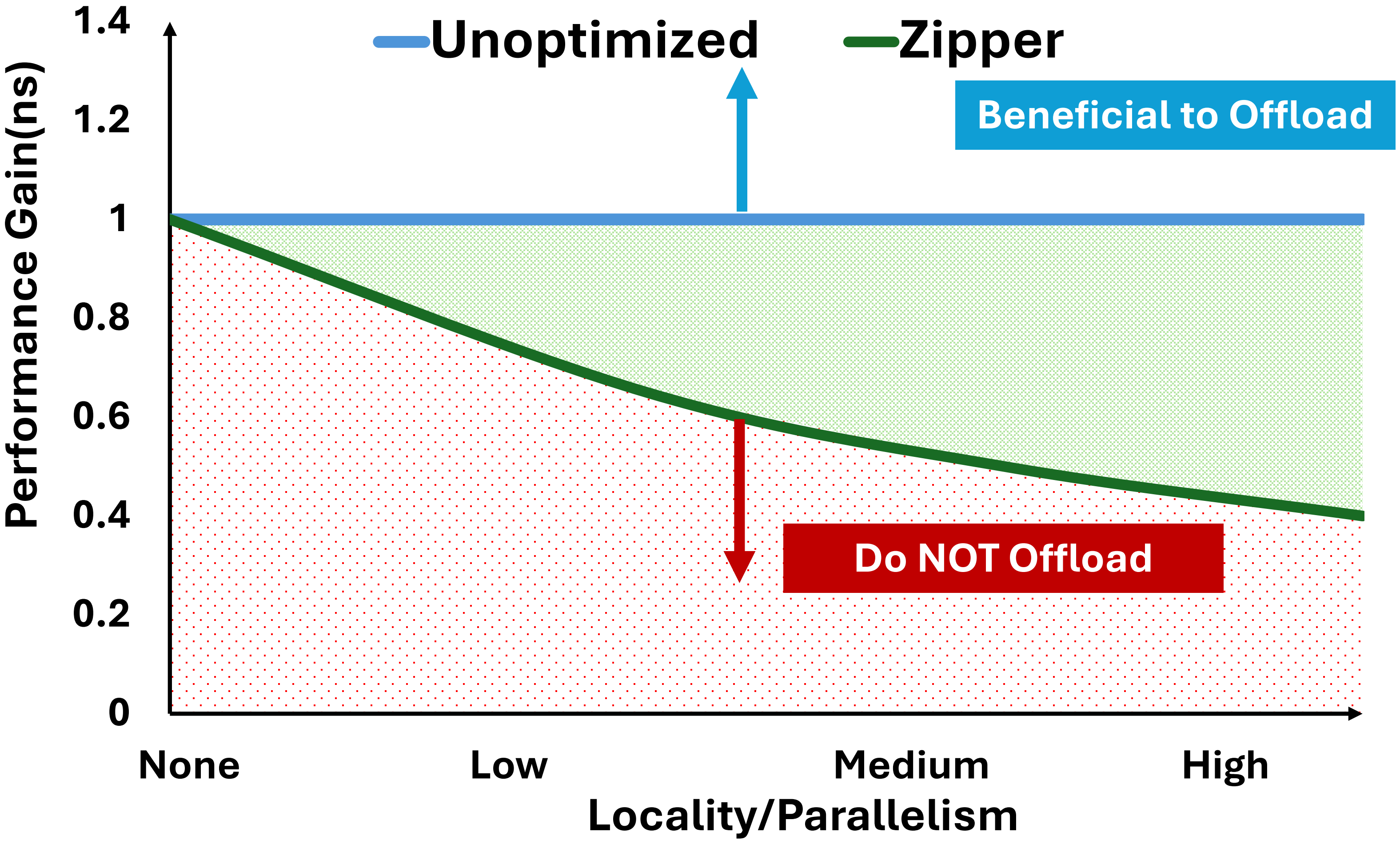
The Death Zone of Compute Offload

$$\frac{P}{O} = \frac{(T_{cpu} - T_{acc})}{T_{Lat}}$$



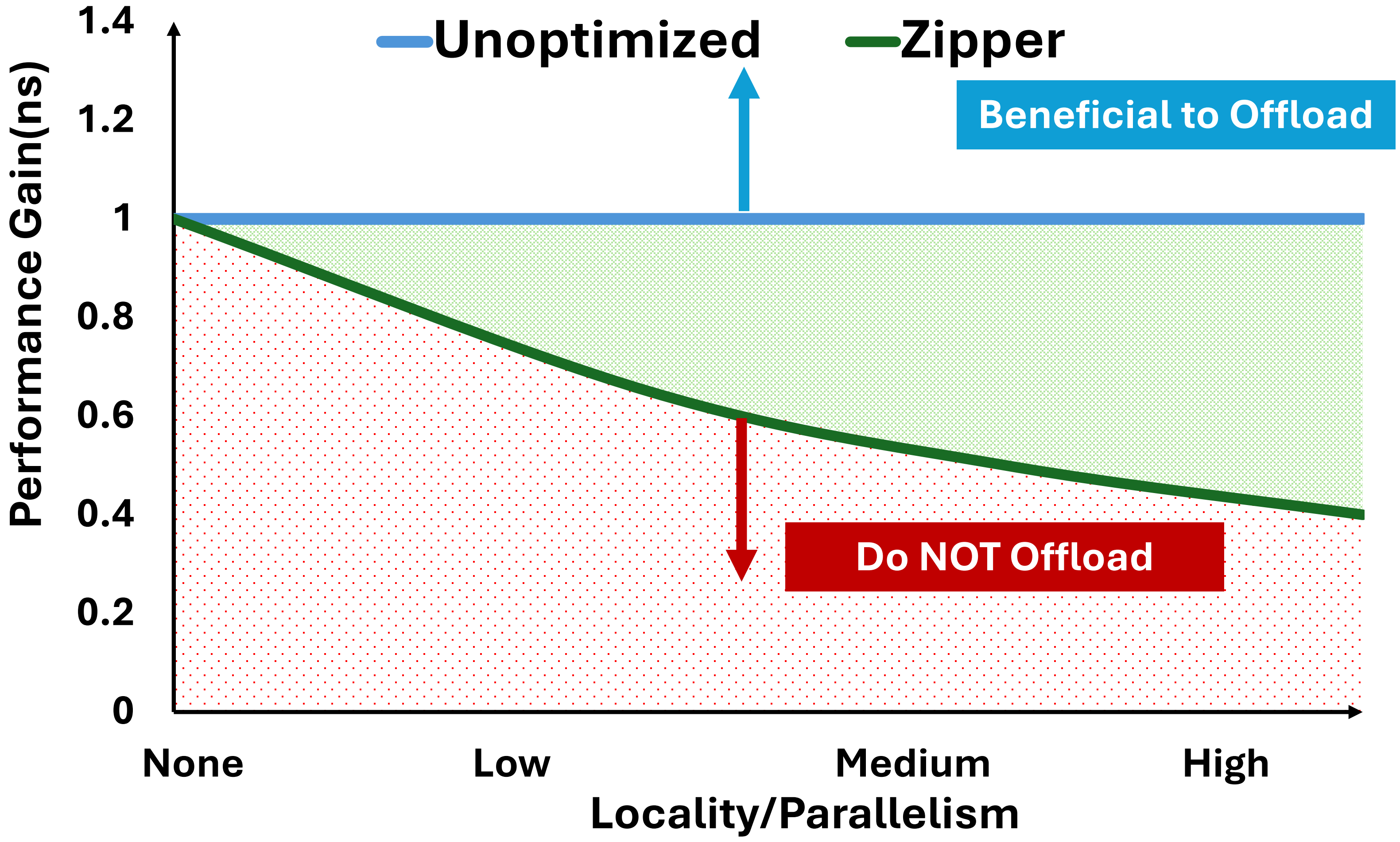
More Forgiving Trade-Offs with Bus Optimizations

$$\frac{P}{O} = \frac{(T_{cpu} - T_{acc})}{T_{Lat}}$$



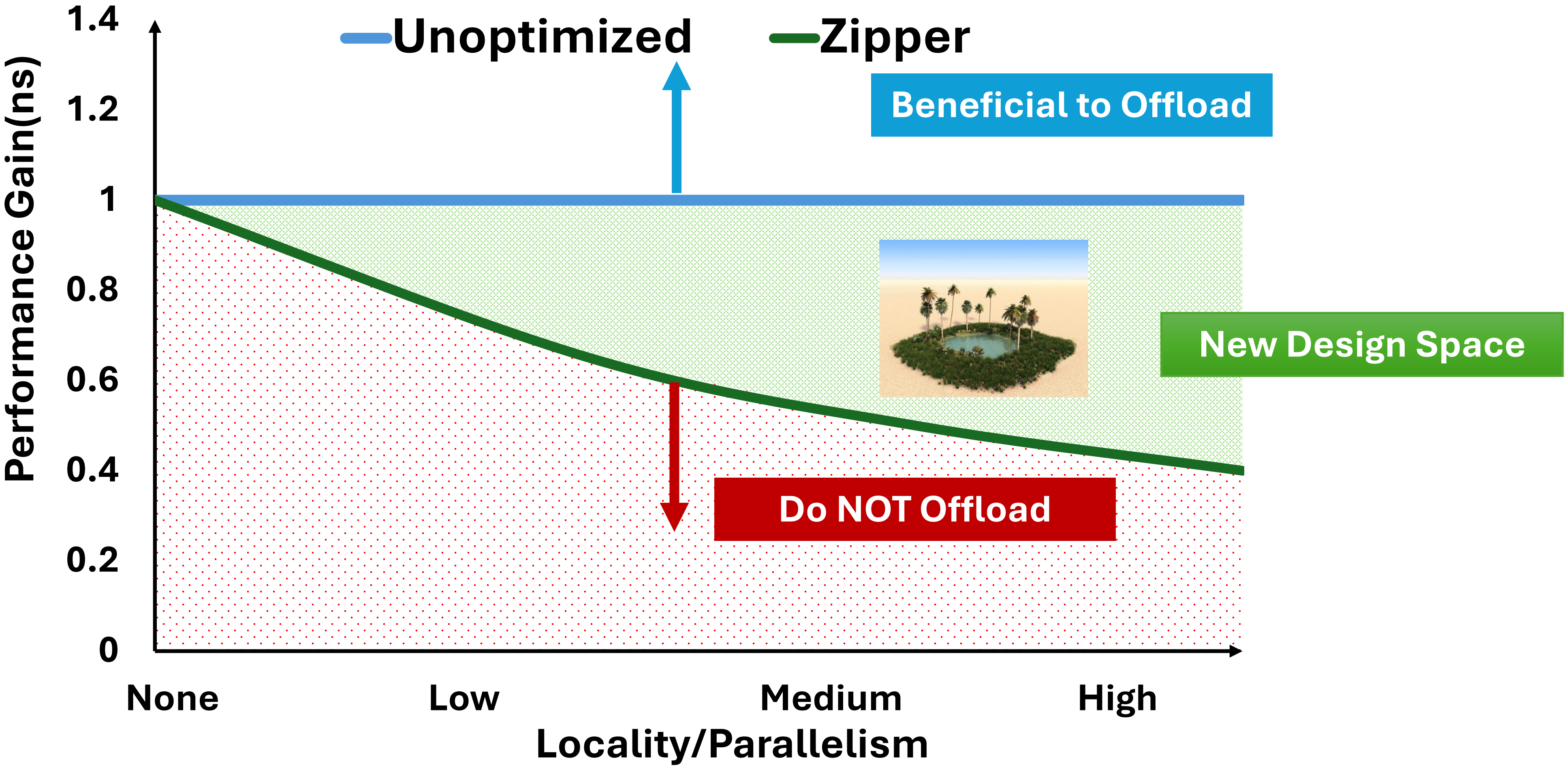
More Forgiving Trade-Offs with Bus Optimizations

$$\frac{P}{O} = \frac{(T_{cpu} - T_{acc})}{T_{Lat}} \downarrow$$



More Forgiving Trade-Offs with Bus Optimizations

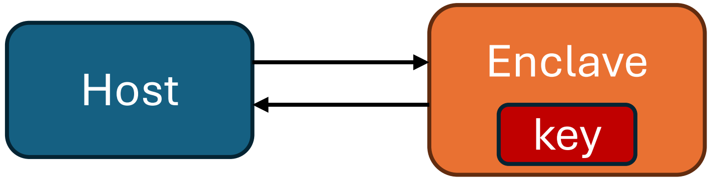
$$\frac{P}{O} = \frac{(T_{cpu} - T_{acc})}{T_{Lat} \downarrow}$$



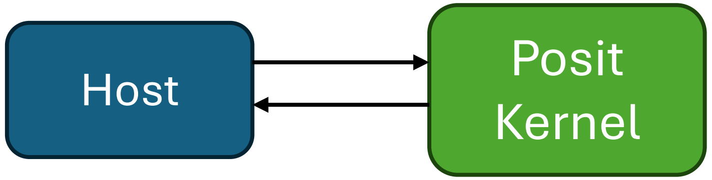
Case Studies

Case Study #1: Sequestered Encryption Enclave + VIP-Bench

- Support RISC-like instructions
- Compute on encrypted operands
- Running privacy-focused algorithms



- Case Study #2: Posit Hardware Kernel + NAS Parallel Benchmark
  - Posit is an alternative to IEEE 754 Floating Point
  - Support arithmetic operations
  - Running scientific applications



Exploitable Opportunities Exist

*Within an 8-Request Window:

- Temporal Locality:
  - Greater than 50% of input operands are from the results of the past 7 requests
- Request-level Parallelism:
  - On average, 5 requests can be executed in parallel
- Traffic Reduction:
  - Less than 22% of the accelerator results need to be sent back to the host
- Device-level Parallelism:
  - On average, greater than100 ms between request issue and result use.

*Based on the two case studies covered in the talk

Challenges

Analyzing Dependencies Between Two ISAs

- Compiler modifications not easy for regular developers

Communicating Locality and Parallelism Information

- Generic communication semantics do not capture this information

Minimal Hardware Modifications

- Intrusive ones are costly and prone to bugs and errors

Different Communication Protocols/APIs to Support

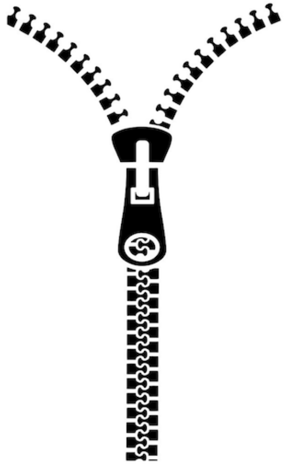
Zipper Overview

Zipper is a set of flexible and reconfigurable **software-hardware optimizations** that tolerate the communication latency for latency-sensitive applications.

Our FPGA-based evaluation shows Zipper provides a significant performance boost while

- Needs **NO** compiler modifications -- only C++ libraries
- Captures **more than 90% of the locality** and enables parallelism
- Has **low** hardware overhead and **NO** intrusive modifications
- Is **agnostic** to underlying bus APIs/semantics

Zipper Overview



Zipper is a set of flexible and reconfigurable **software-hardware optimizations** that <u>tolerate the communication latency</u> for latency-sens applications.

Our FPGA-based evaluation shows Zipper provides a significant performance boost while

- Needs **NO** compiler modifications -- only C++ libraries
- Captures **more than 90% of the locality** and enables parallelism
- Has **low** hardware overhead and **NO** intrusive modifications
- Is **agnostic** to underlying bus APIs/semantics

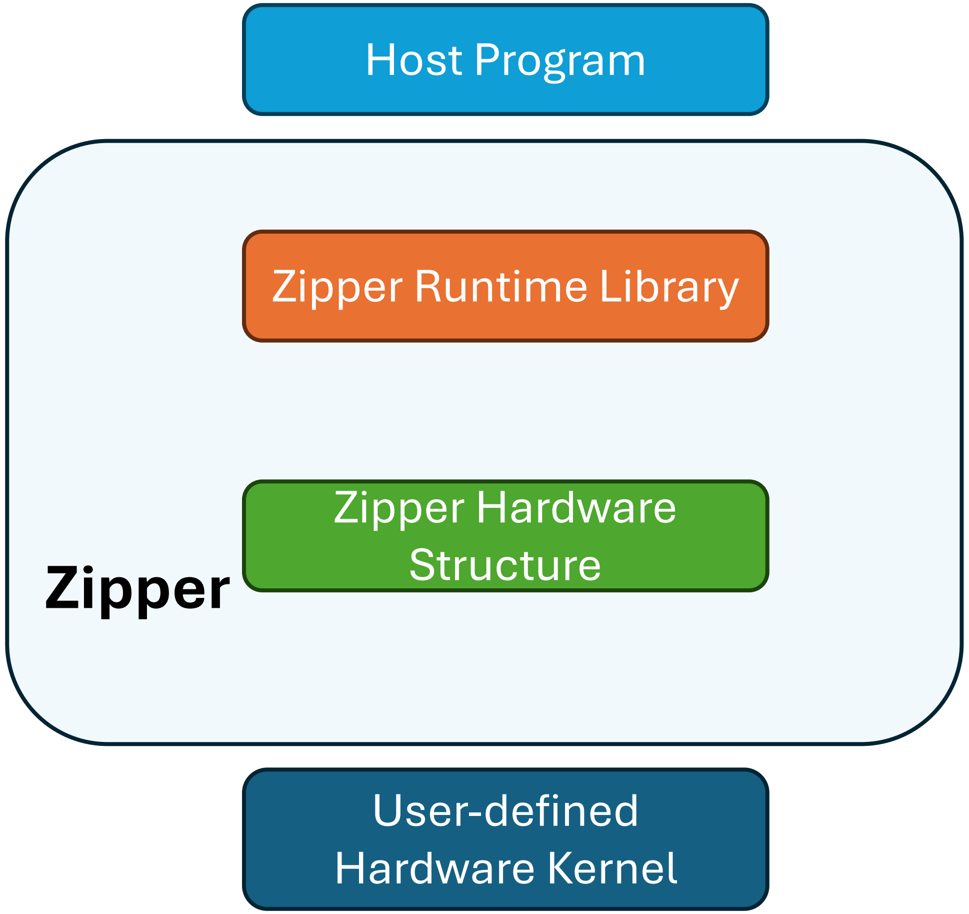
Zipper Overview

Software Runtime Library:

- Detects dependencies between accelerator requests and between the host and the accelerator request.
- Manages shared memory.
- Sends requests to the accelerator & fetches results back to the host.

Hardware Structure:

- Schedules request issuing
- Buffers recent results for locality
- Fetches input or forwards results



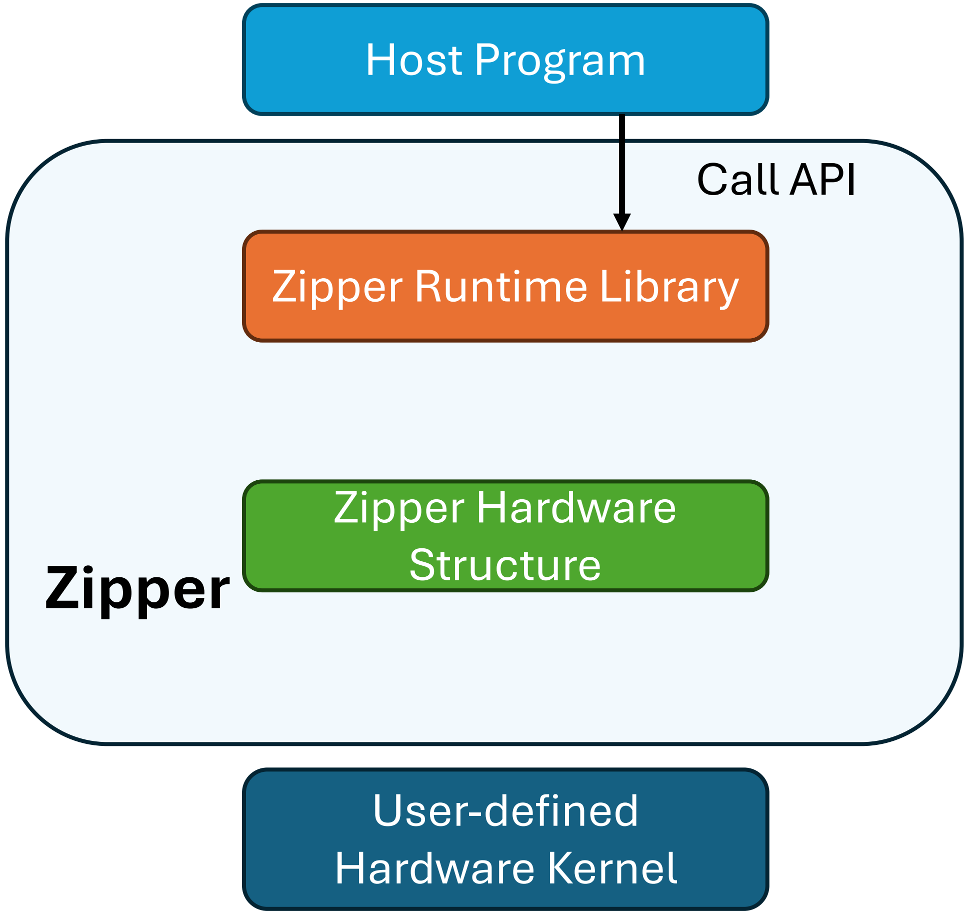
Zipper Overview

Software Runtime Library:

- Detects dependencies between accelerator requests and between the host and the accelerator request.
- Manages shared memory.
- Sends requests to the accelerator & fetches results back to the host.

Hardware Structure:

- Schedules request issuing
- Buffers recent results for locality
- Fetches input or forwards results



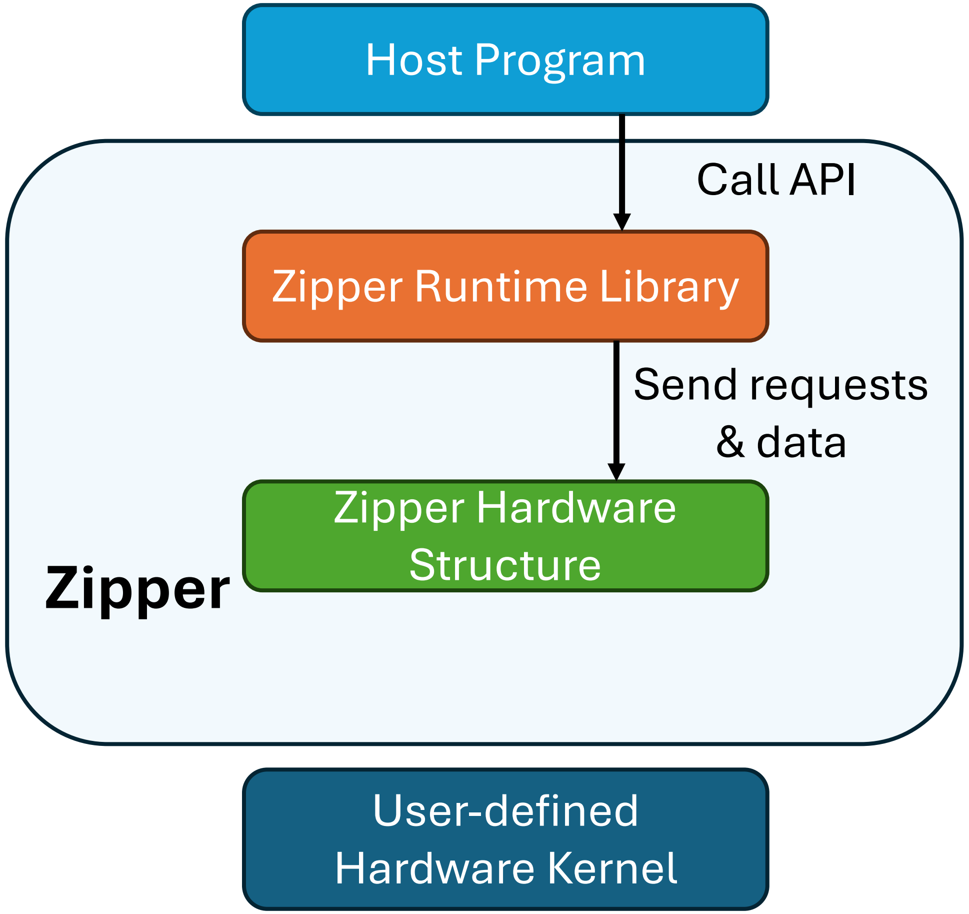
Zipper Overview

Software Runtime Library:

- Detects dependencies between accelerator requests and between the host and the accelerator request.
- Manages shared memory.
- Sends requests to the accelerator & fetches results back to the host.

Hardware Structure:

- Schedules request issuing
- Buffers recent results for locality
- Fetches input or forwards results



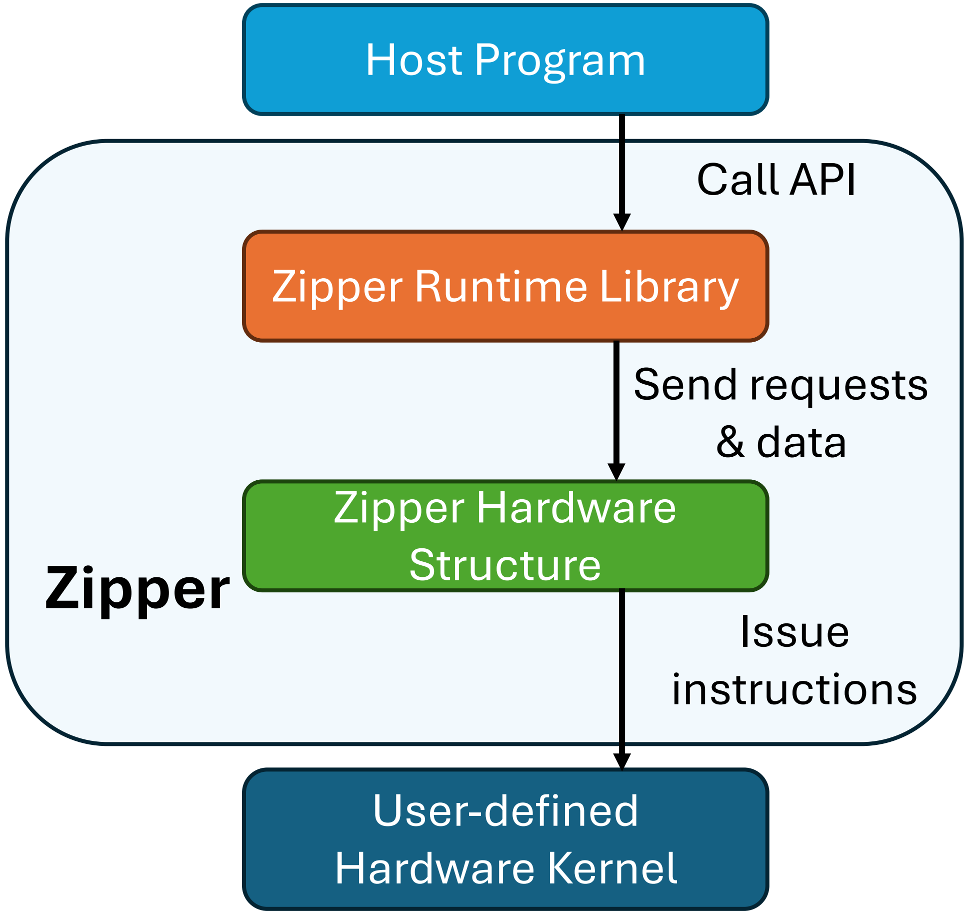
Zipper Overview

Software Runtime Library:

- Detects dependencies between accelerator requests and between the host and the accelerator request.
- Manages shared memory.
- Sends requests to the accelerator & fetches results back to the host.

Hardware Structure:

- Schedules request issuing
- Buffers recent results for locality
- Fetches input or forwards results



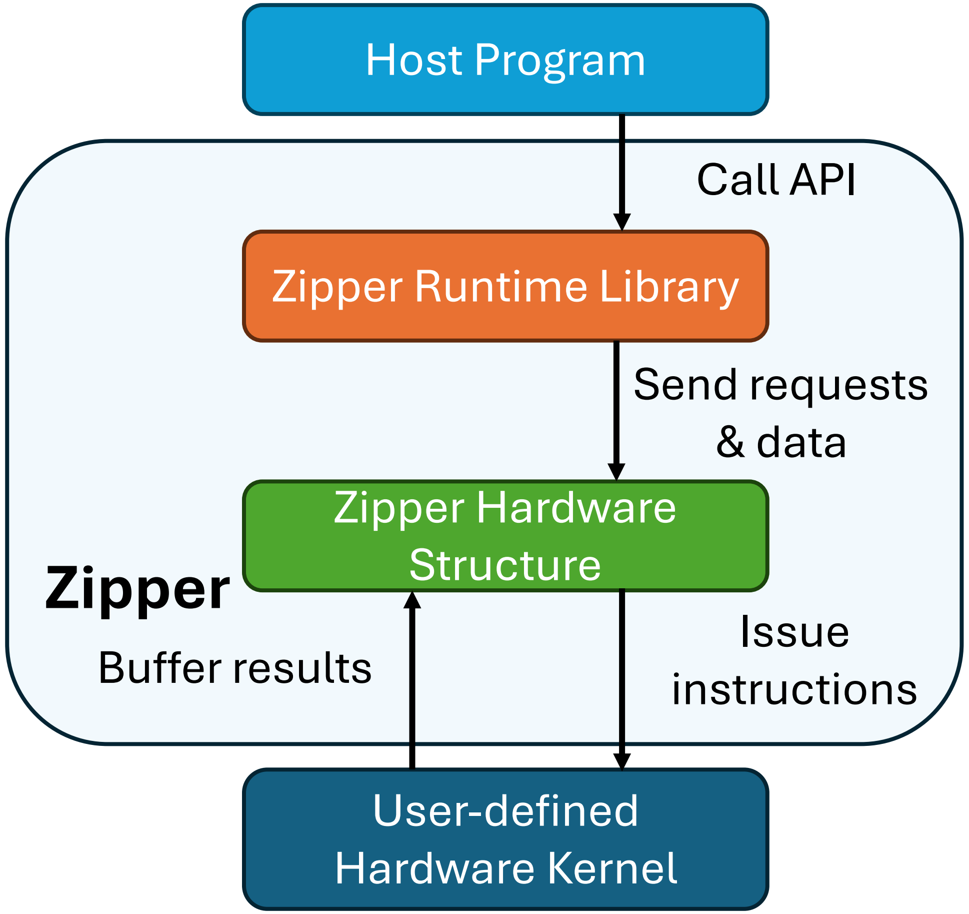
Zipper Overview

Software Runtime Library:

- Detects dependencies between accelerator requests and between the host and the accelerator request.
- Manages shared memory.
- Sends requests to the accelerator & fetches results back to the host.

Hardware Structure:

- Schedules request issuing
- Buffers recent results for locality
- Fetches input or forwards results



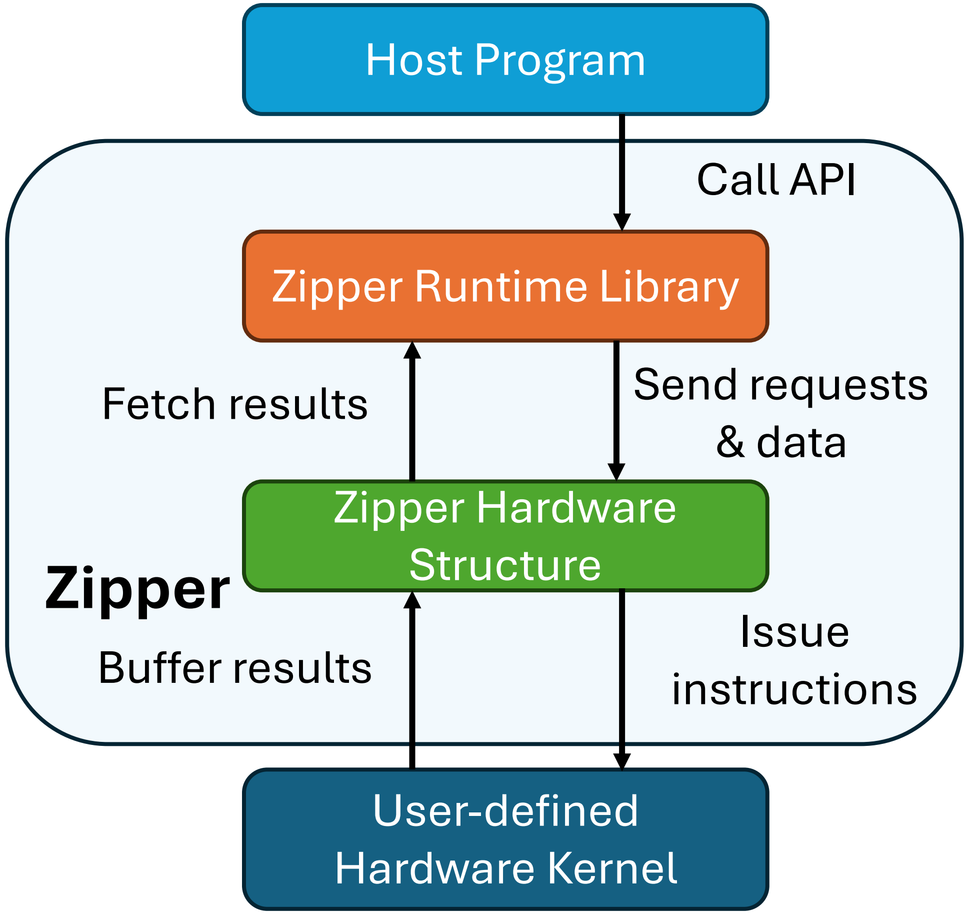
Zipper Overview

Software Runtime Library:

- Detects dependencies between accelerator requests and between the host and the accelerator request.
- Manages shared memory.
- Sends requests to the accelerator & fetches results back to the host.

Hardware Structure:

- Schedules request issuing
- Buffers recent results for locality
- Fetches input or forwards results



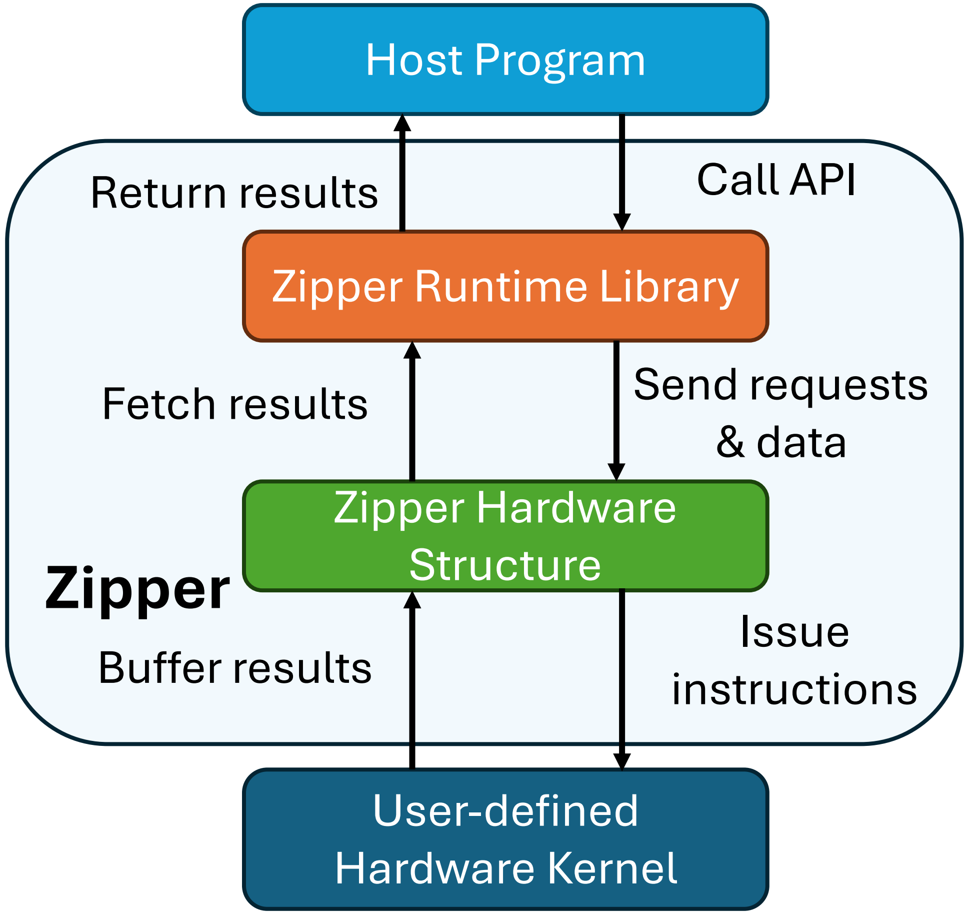
Zipper Overview

Software Runtime Library:

- Detects dependencies between accelerator requests and between the host and the accelerator request.
- Manages shared memory.
- Sends requests to the accelerator & fetches results back to the host.

Hardware Structure:

- Schedules request issuing
- Buffers recent results for locality
- Fetches input or forwards results

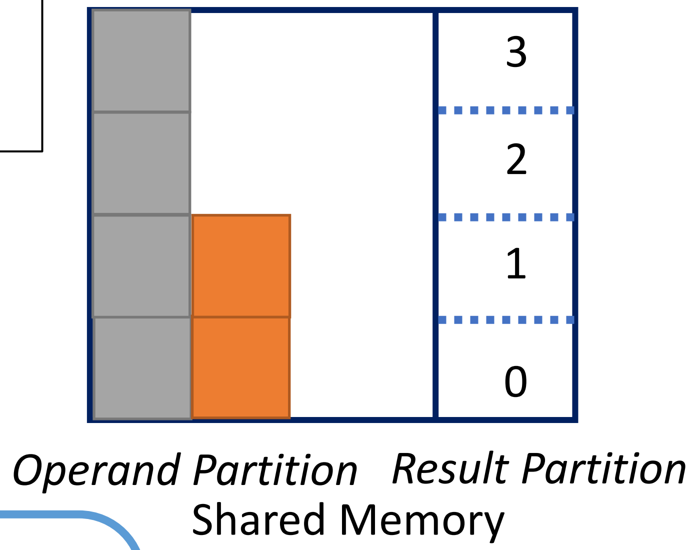


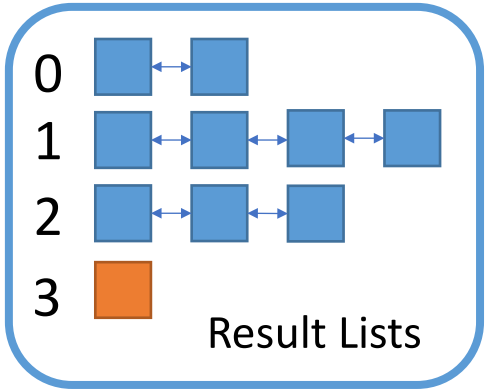
Zipper Runtime Library

Three data structures:

- Overloaded data types: track results' status, location, etc.
- Shared Memory: Separate into operand partition and result partition.
- Result list: track objects that share the same results.

```
Data_type
{ Accl_val_t val;
    bool valid;
    bool inAccl;
    int location; }
```

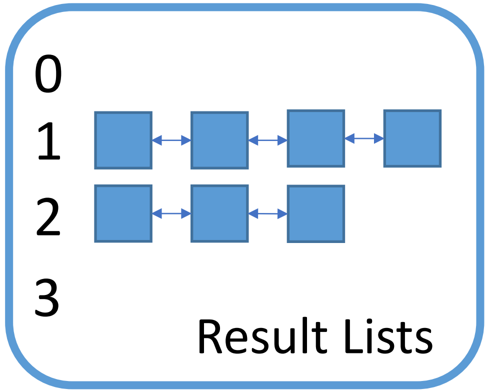


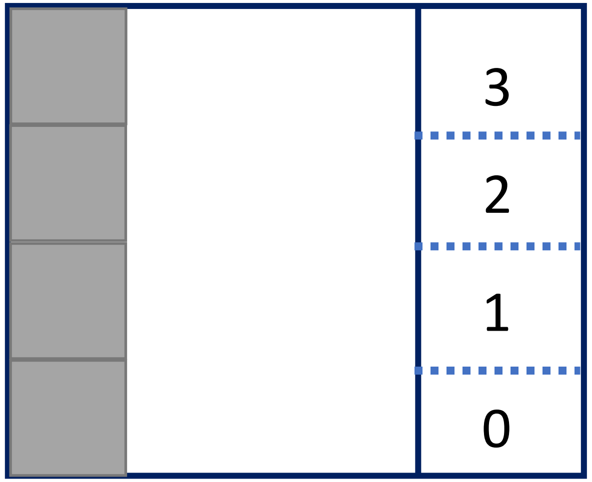


Zipper Runtime Library Example(1/2)

Example Code Snippet

```
int m, n, i;
Accl_t a = m ⊗ n;
Accl_t b = a ⊗ i;
```





Operand Partition *Result Partition*
Shared Memory

```
a{
  Accl_val_t val;
  bool valid;
  bool inAccl;
  int location;
}
```

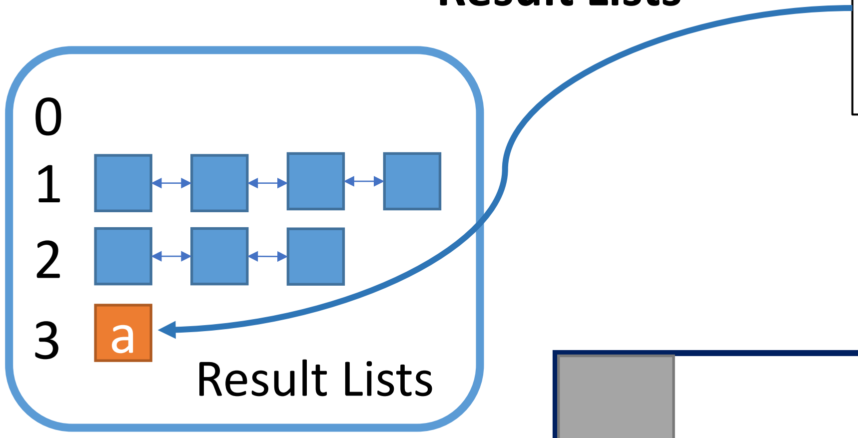
Data Bus

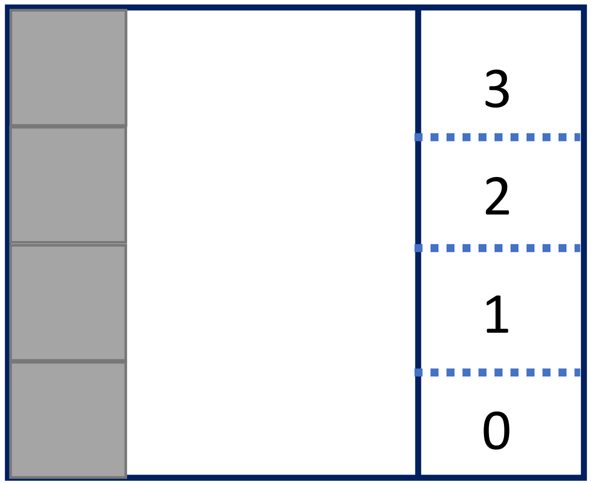
Zipper Runtime Library Example(1/2)

Example Code Snippet

```
int m, n, i;
Accl_t a = m ⊗ n;
Accl_t b = a ⊗ i;
```

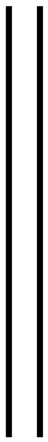
1 Register a in Result Lists





Operand Partition *Result Partition*
Shared Memory

```
a{
  Accl_val_t val;
  bool valid;
  bool inAccl;
  int location;
}
```

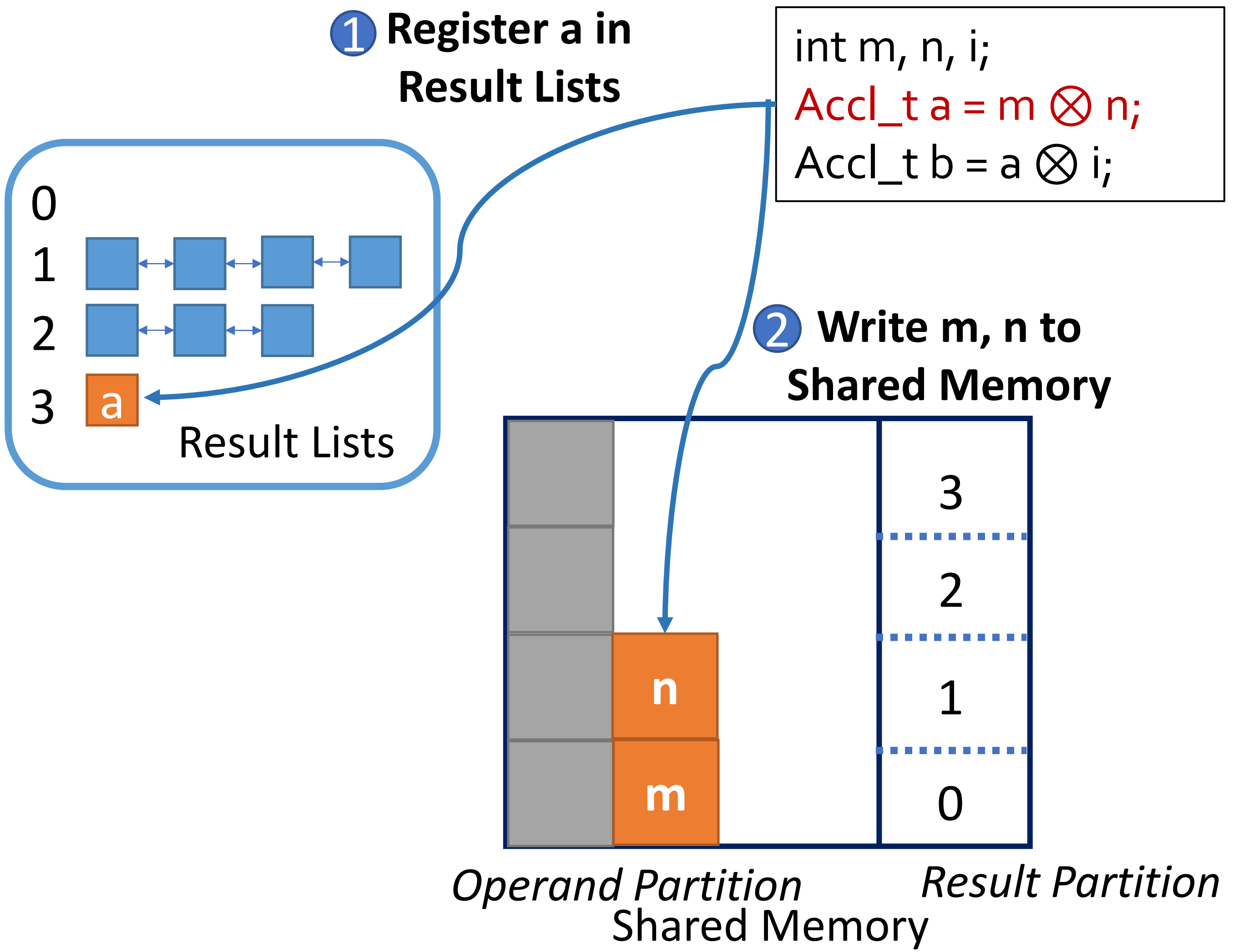


Data Bus

Zipper Runtime Library Example(1/2)

Example Code Snippet

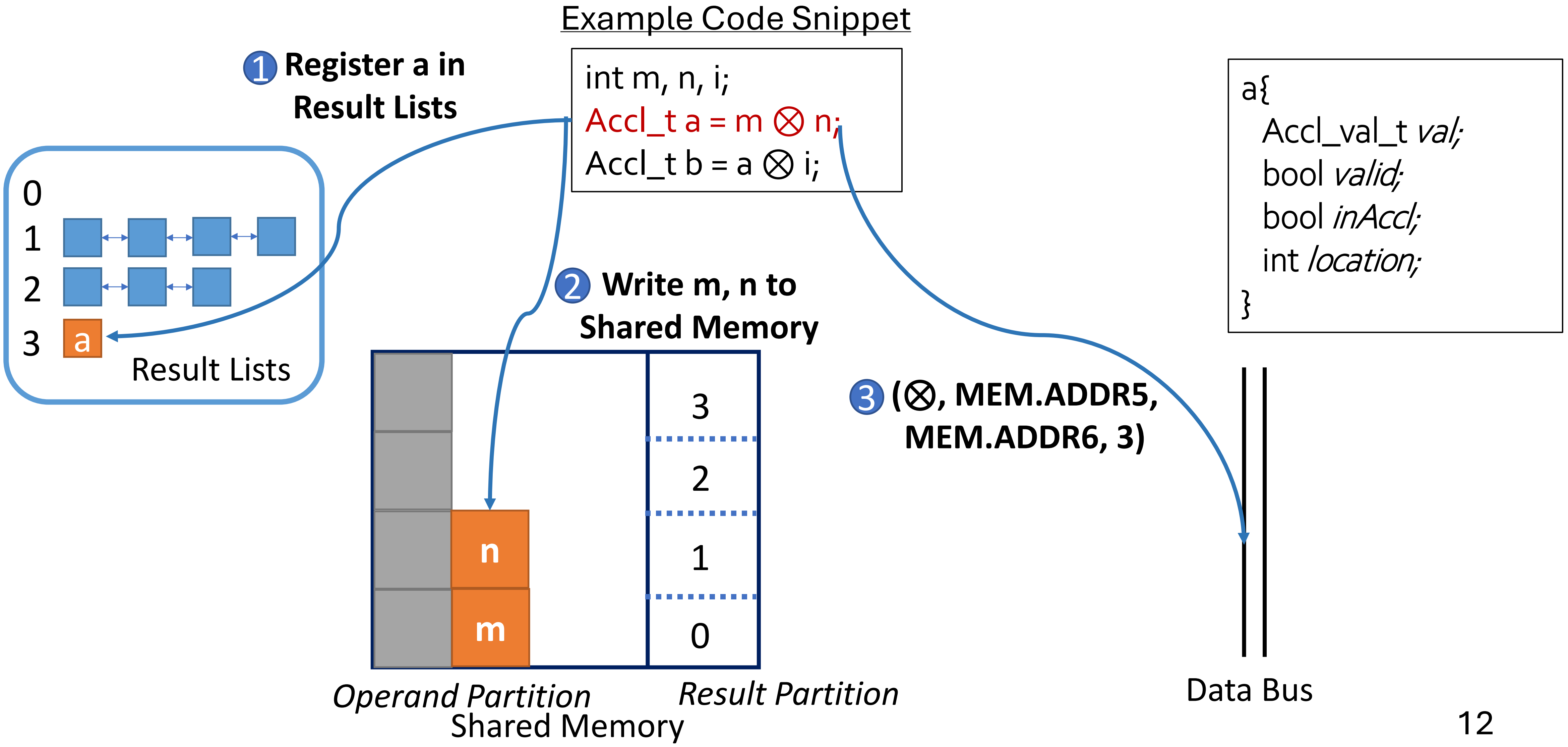
```
int m, n, i;
Accl_t a = m ⊗ n;
Accl_t b = a ⊗ i;
```



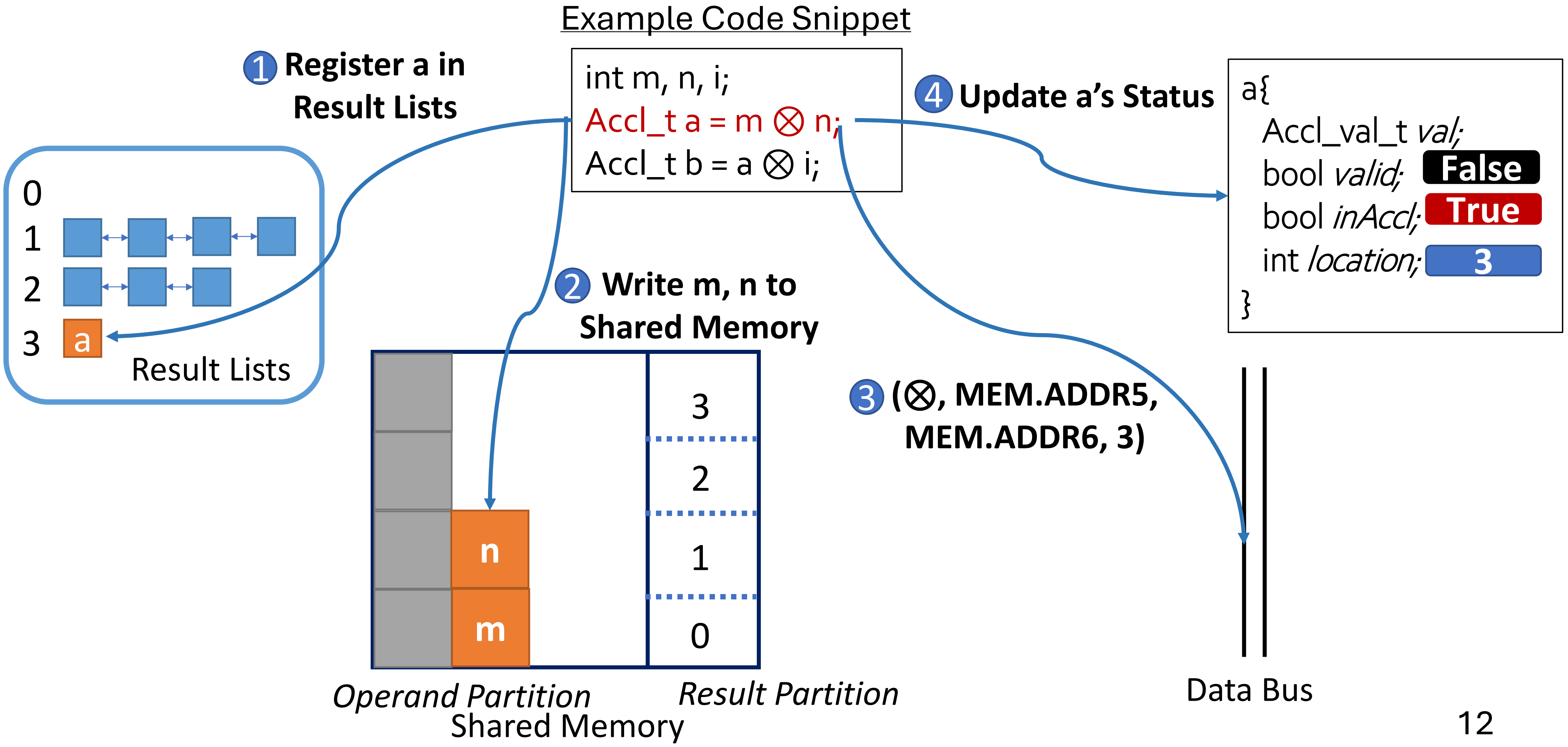

```
a{
  Accl_val_t val;
  bool valid;
  bool inAccl;
  int location;
}
```

Data Bus

Zipper Runtime Library Example(1/2)



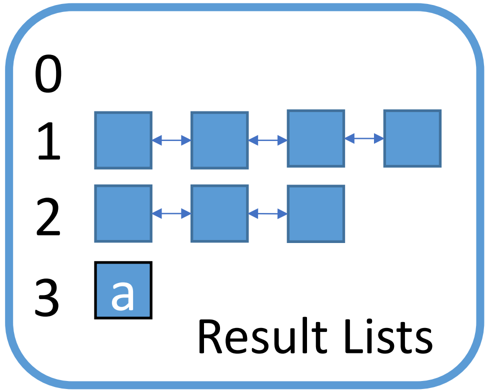
Zipper Runtime Library Example(1/2)

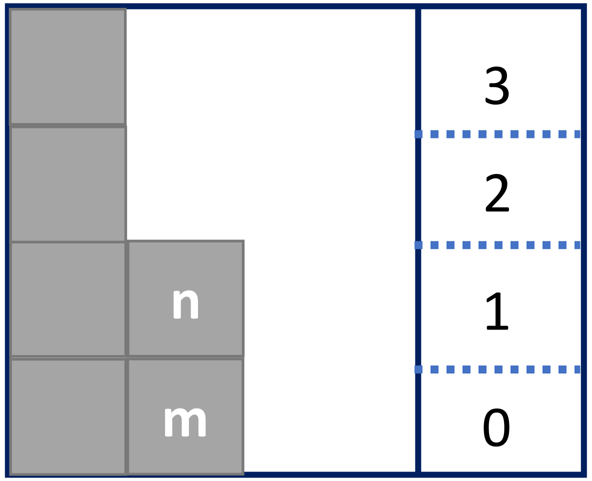


Zipper Runtime Library Example(2/2)

<u>Example Code Snippet</u>

```
int m, n, i;
Accl_t a = m ⊗ n;
Accl_t b = a ⊗ i;
```






```
b{
  Accl_val_t val;
  bool valid;
  bool inAccl;
  int location;
}
```

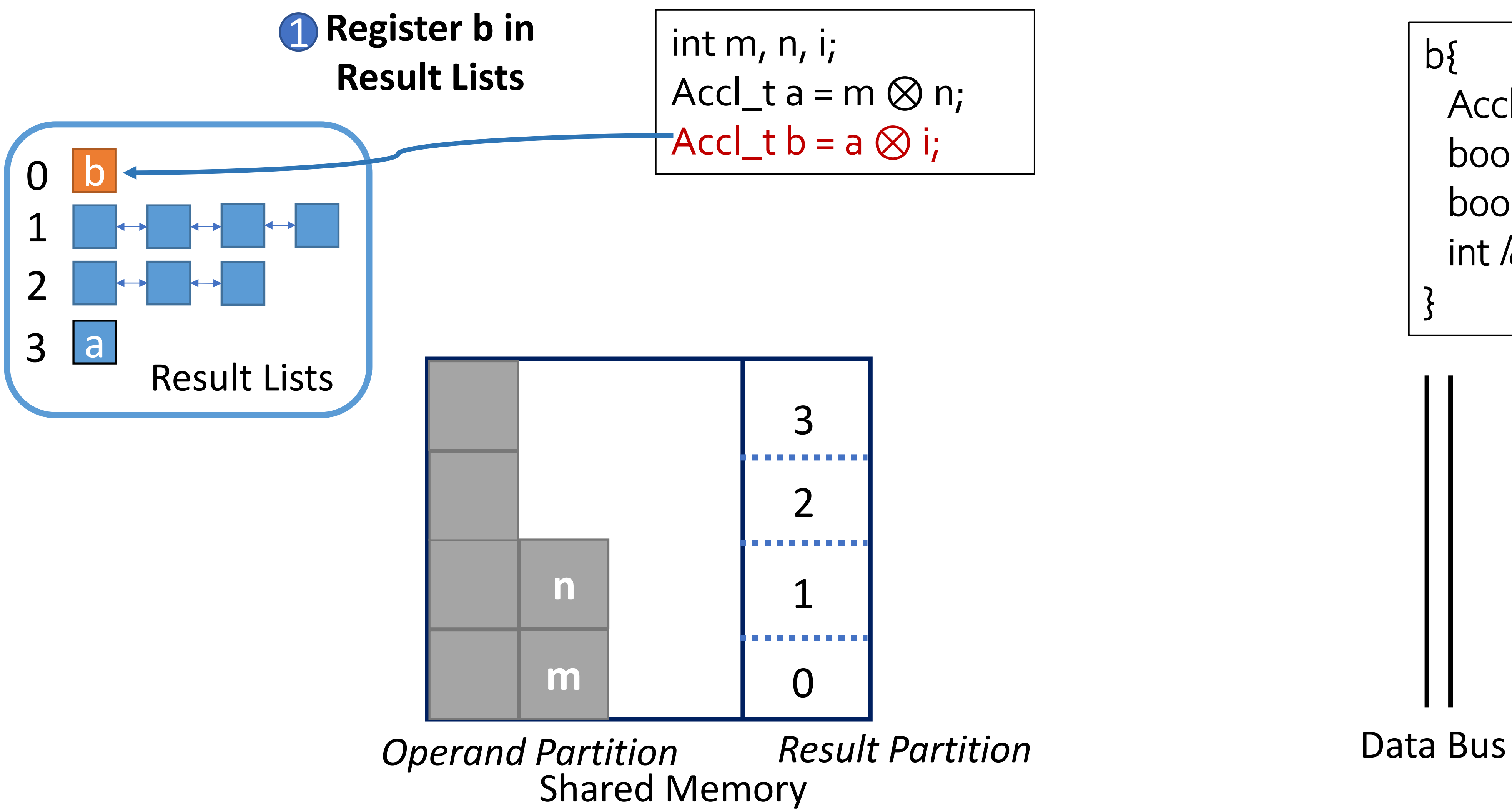
Data Bus

Zipper Runtime Library Example(2/2)

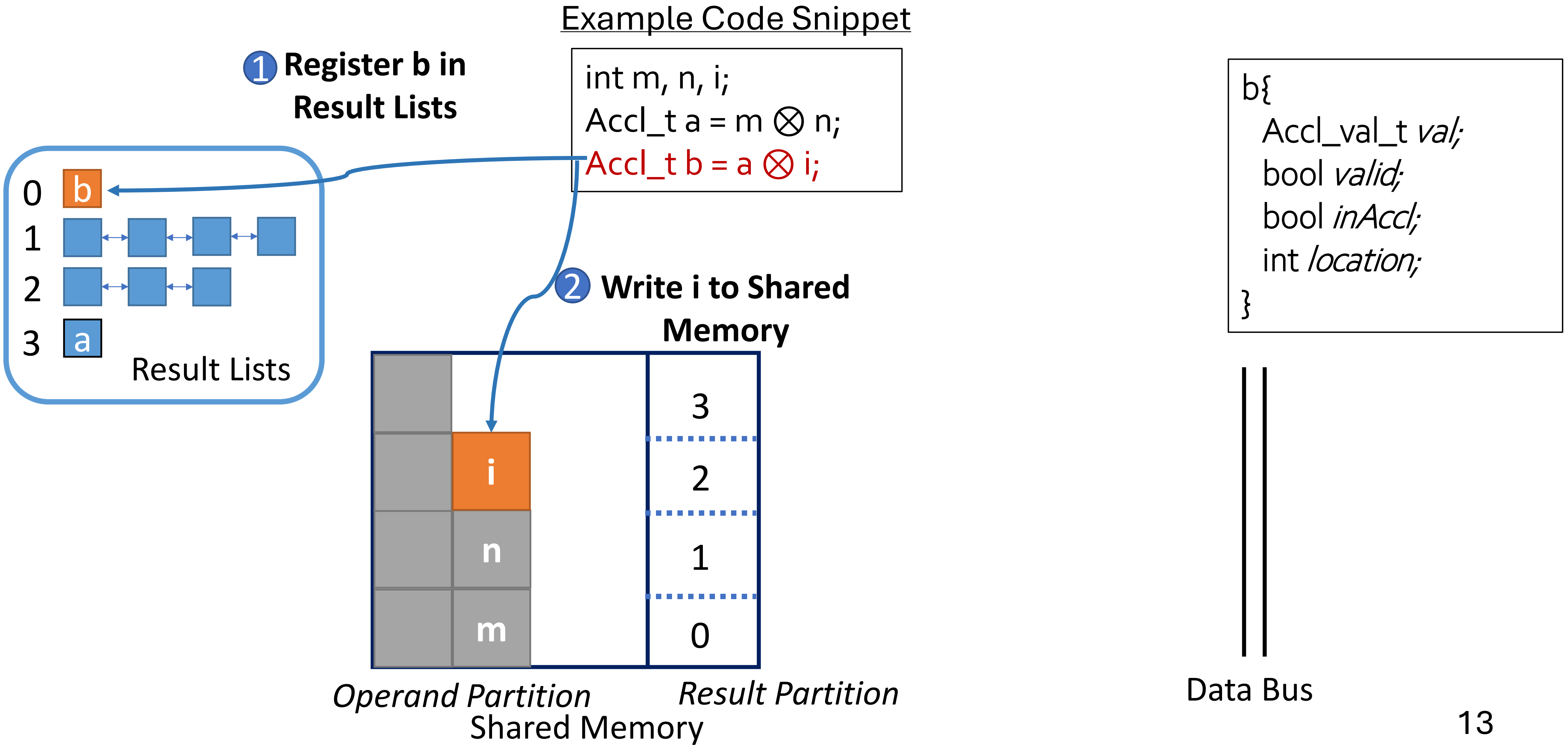
Example Code Snippet

```
int m, n, i;
Accl_t a = m ⊗ n;
Accl_t b = a ⊗ i;
```

```
b{
  Accl_val_t val;
  bool valid;
  bool inAccl;
  int location;
}
```



Zipper Runtime Library Example(2/2)

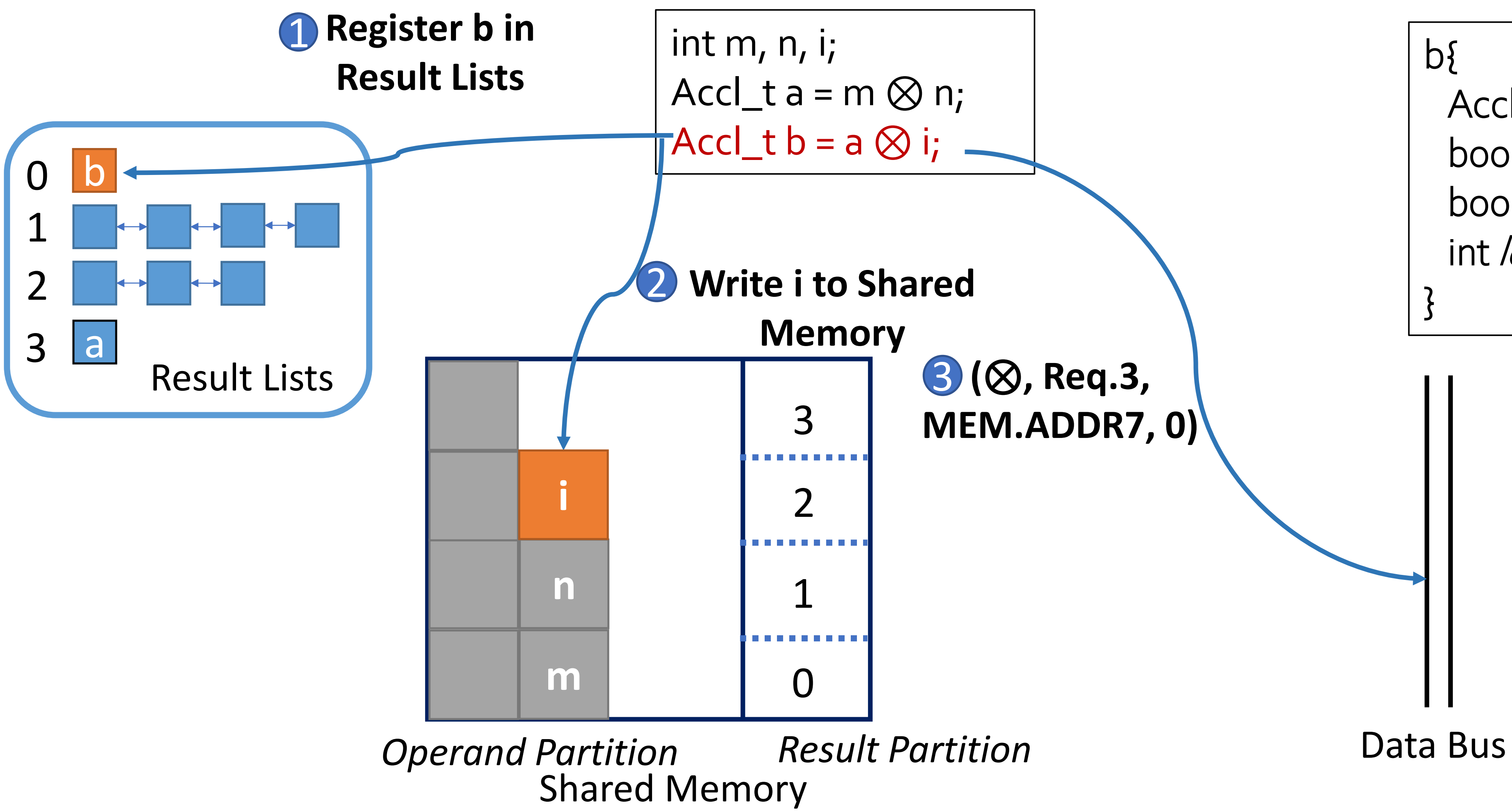


Zipper Runtime Library Example(2/2)

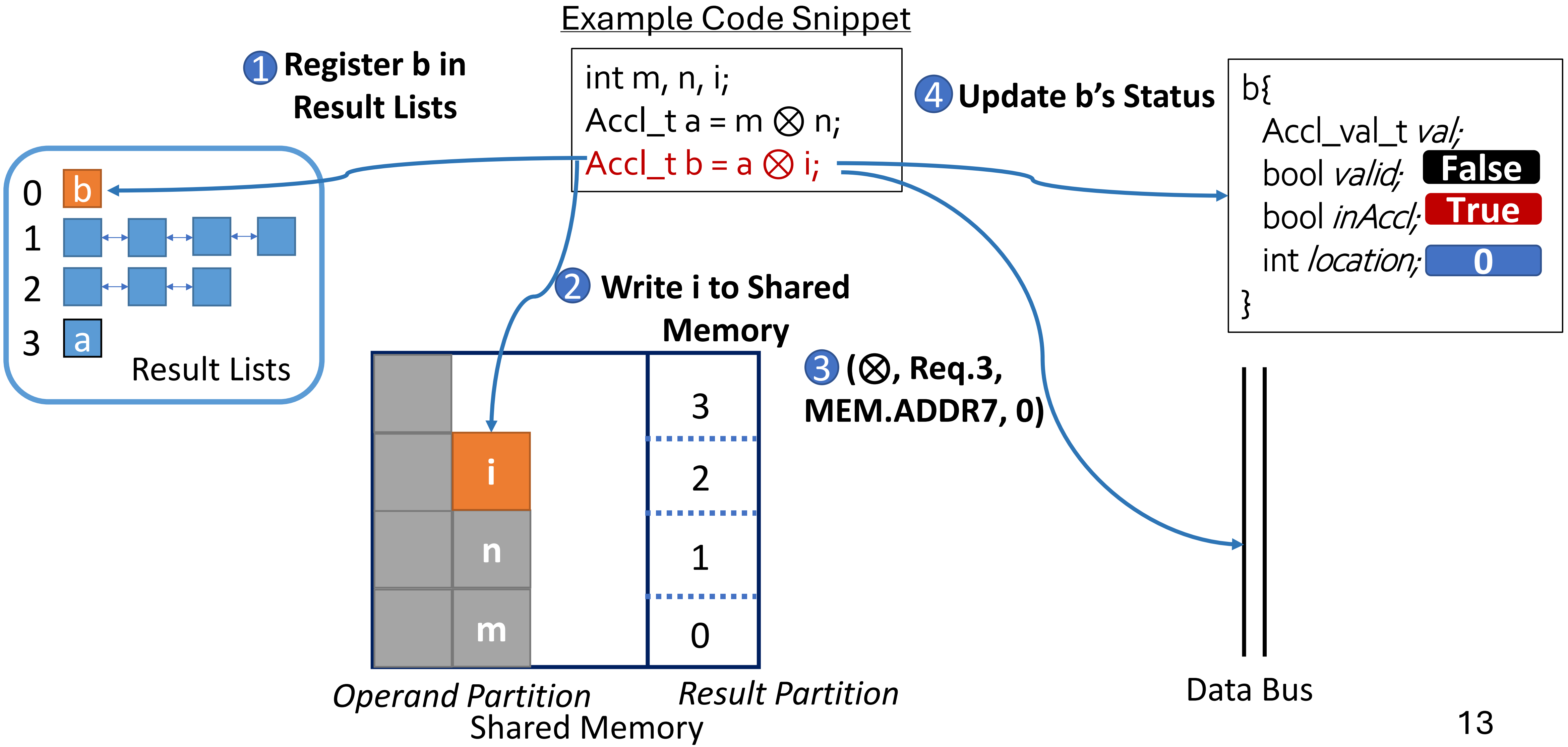
Example Code Snippet

```
int m, n, i;
Accl_t a = m ⊗ n;
Accl_t b = a ⊗ i;
```

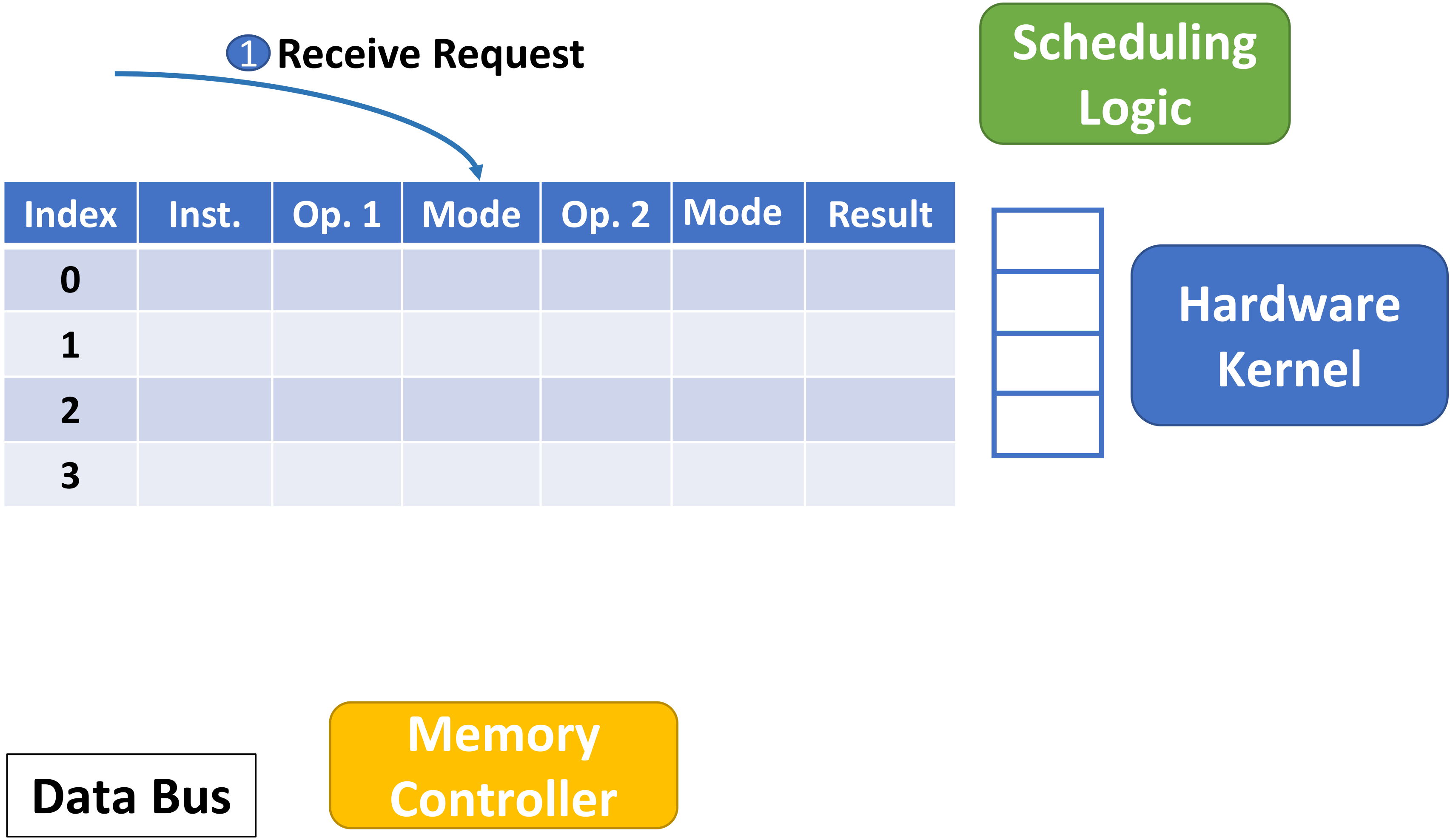
```
b{
  Accl_val_t val;
  bool valid;
  bool inAccl;
  int location;
}
```



Zipper Runtime Library Example(2/2)



Zipper Hardware Structure



Index	Inst.	Op. 1	Mode	Op. 2	Mode	Result
0						
1						
2						
3						

Zipper Hardware Structure

① **Receive Request**

Scheduling Logic

Index	Inst.	Op. 1	Mode	Op. 2	Mode	Result
0						
1						
2						
3	⊗	*Addr5*	MEM	*Addr6*	MEM	

Hardware Kernel

Data Bus

Memory Controller

Zipper Hardware Structure

1 Receive Request

Scheduling Logic

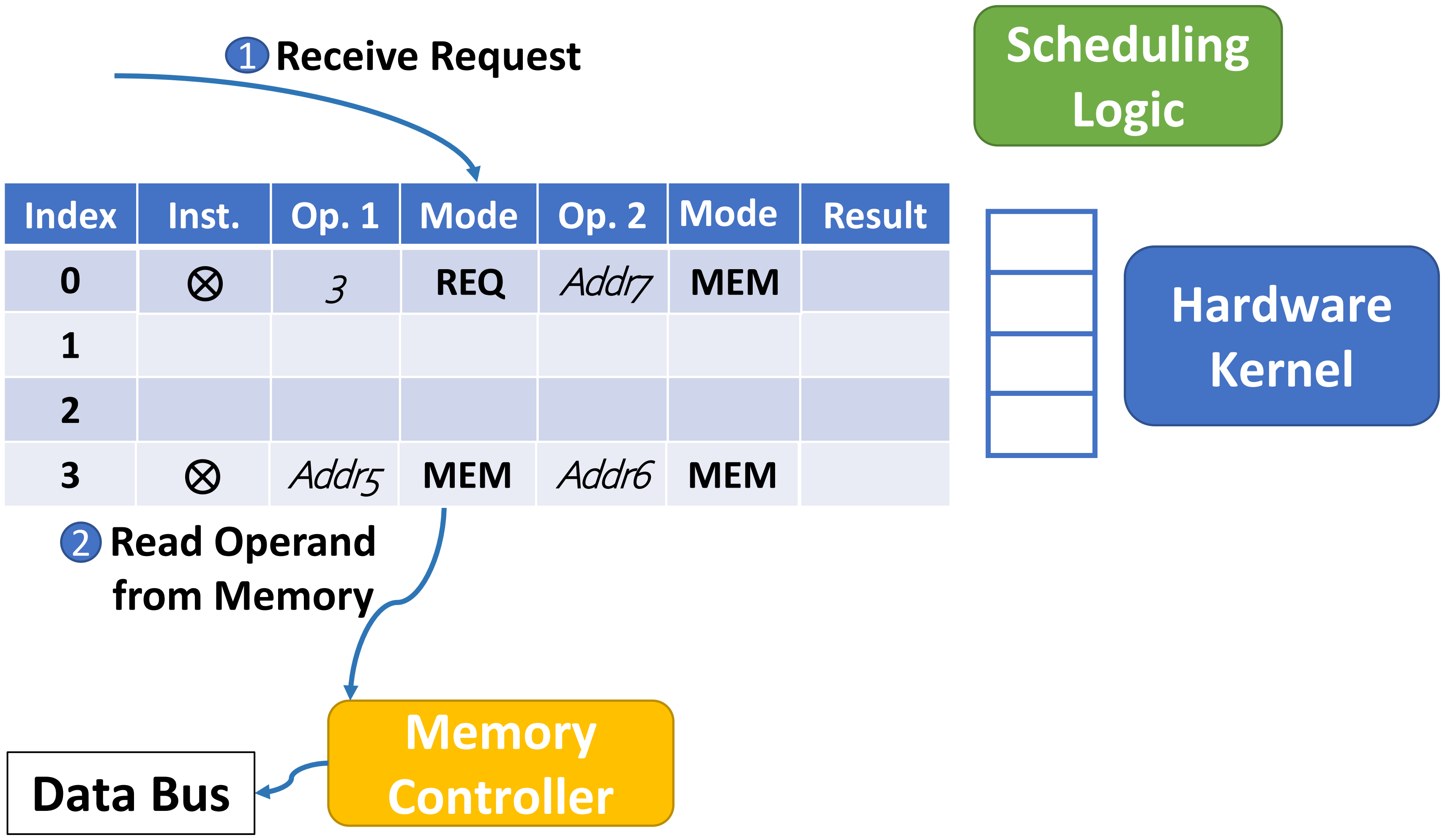
Index	Inst.	Op. 1	Mode	Op. 2	Mode	Result
0	⊗	*3*	REQ	*Addr7*	MEM	
1						
2						
3	⊗	*Addr5*	MEM	*Addr6*	MEM	

Hardware Kernel

Data Bus

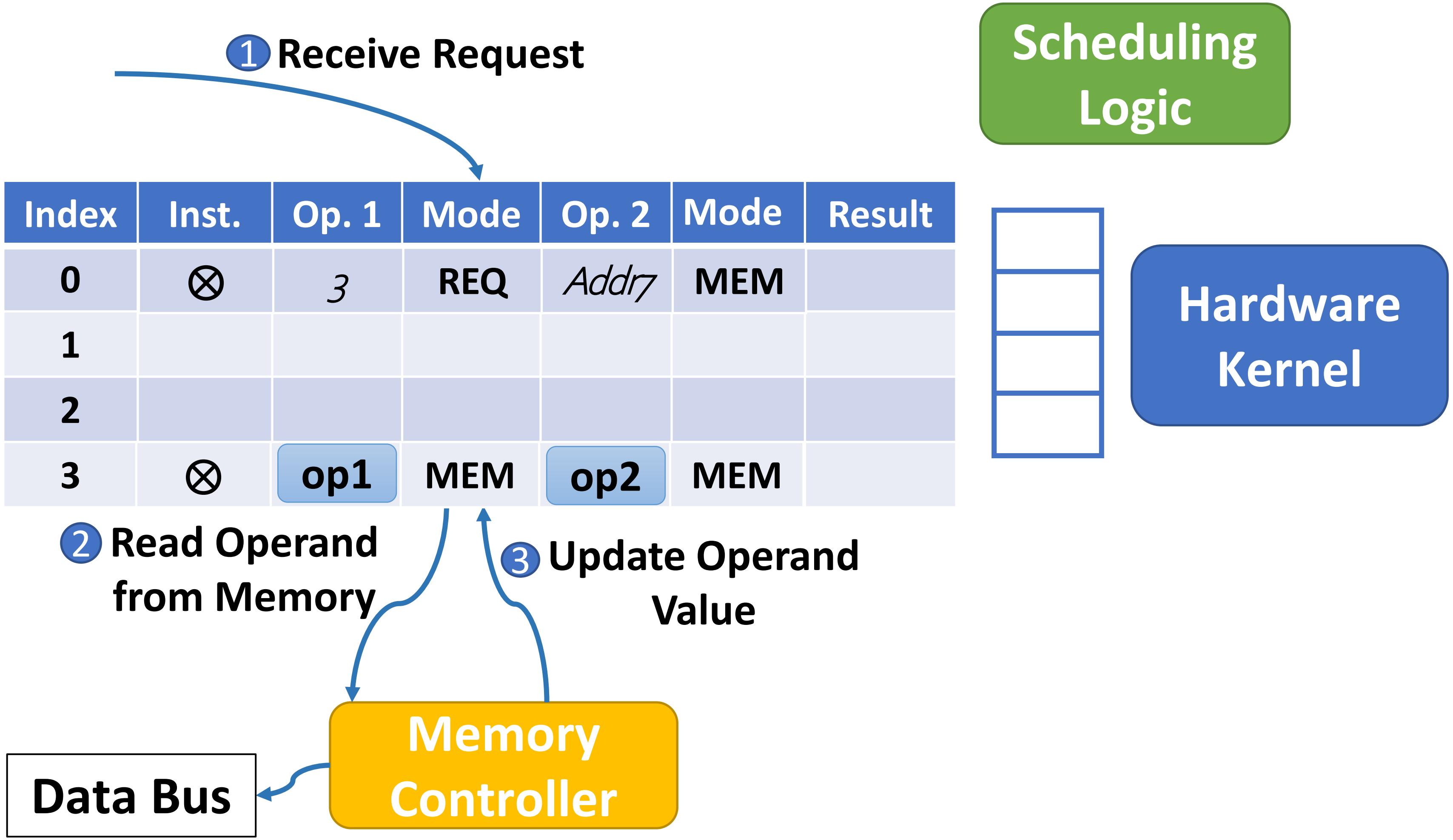
Memory Controller

Zipper Hardware Structure



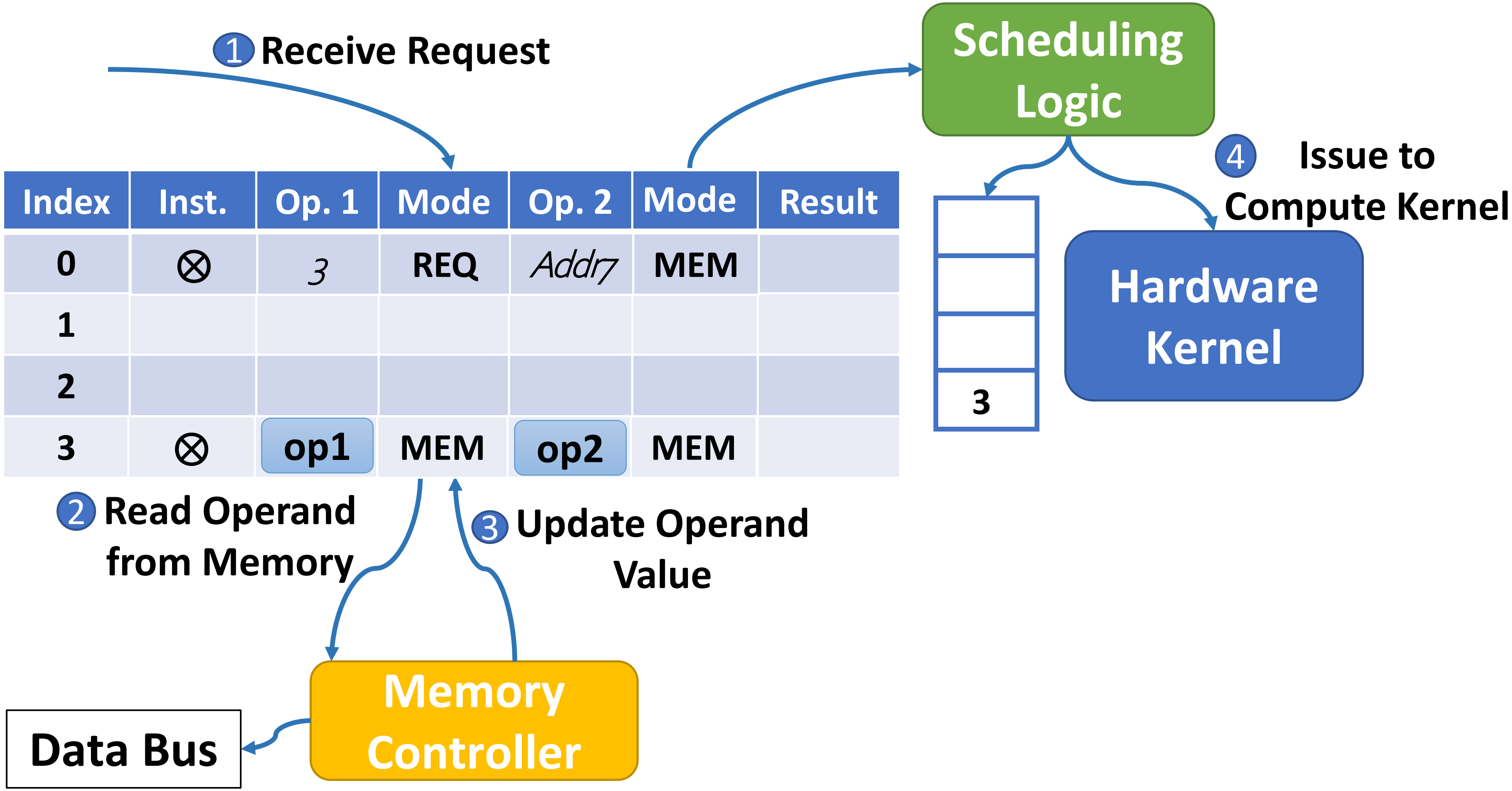
Index	Inst.	Op. 1	Mode	Op. 2	Mode	Result
0	⊗	*3*	REQ	*Addr7*	MEM	
1						
2						
3	⊗	*Addr5*	MEM	*Addr6*	MEM	

Zipper Hardware Structure

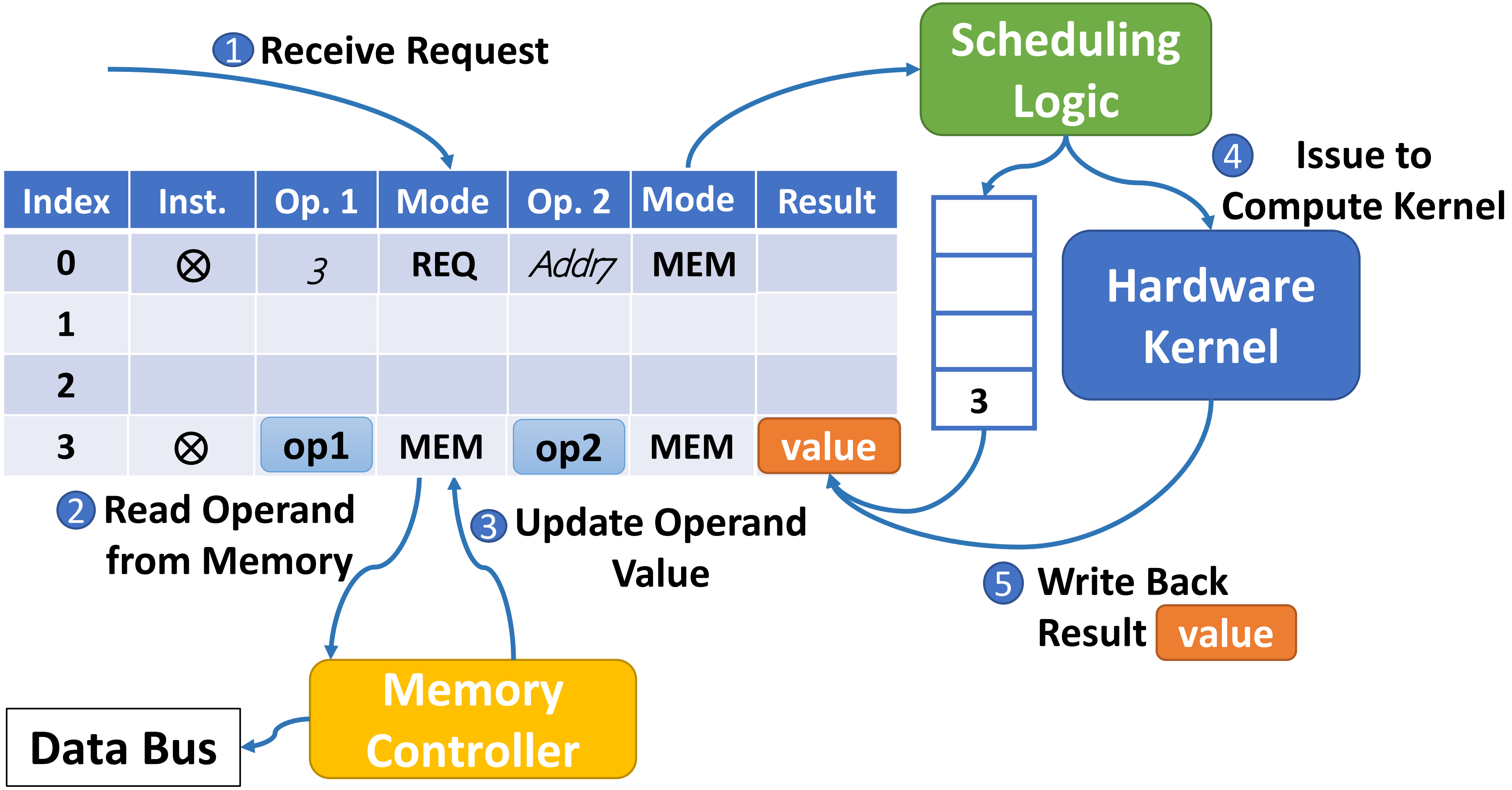


Index	Inst.	Op. 1	Mode	Op. 2	Mode	Result
0	⊗	*3*	REQ	*Addr7*	MEM	
1						
2						
3	⊗	op1	MEM	op2	MEM	

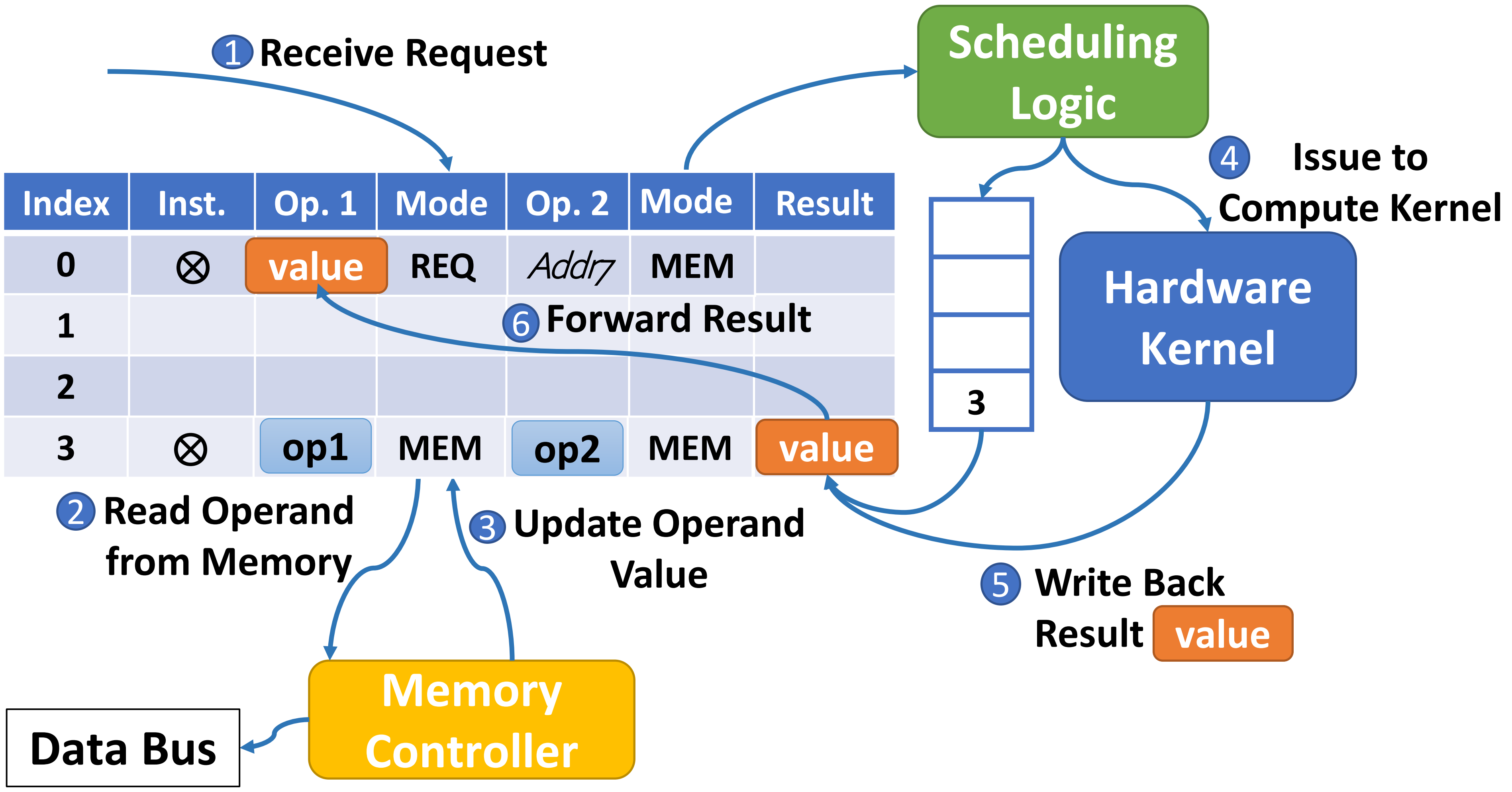
Zipper Hardware Structure



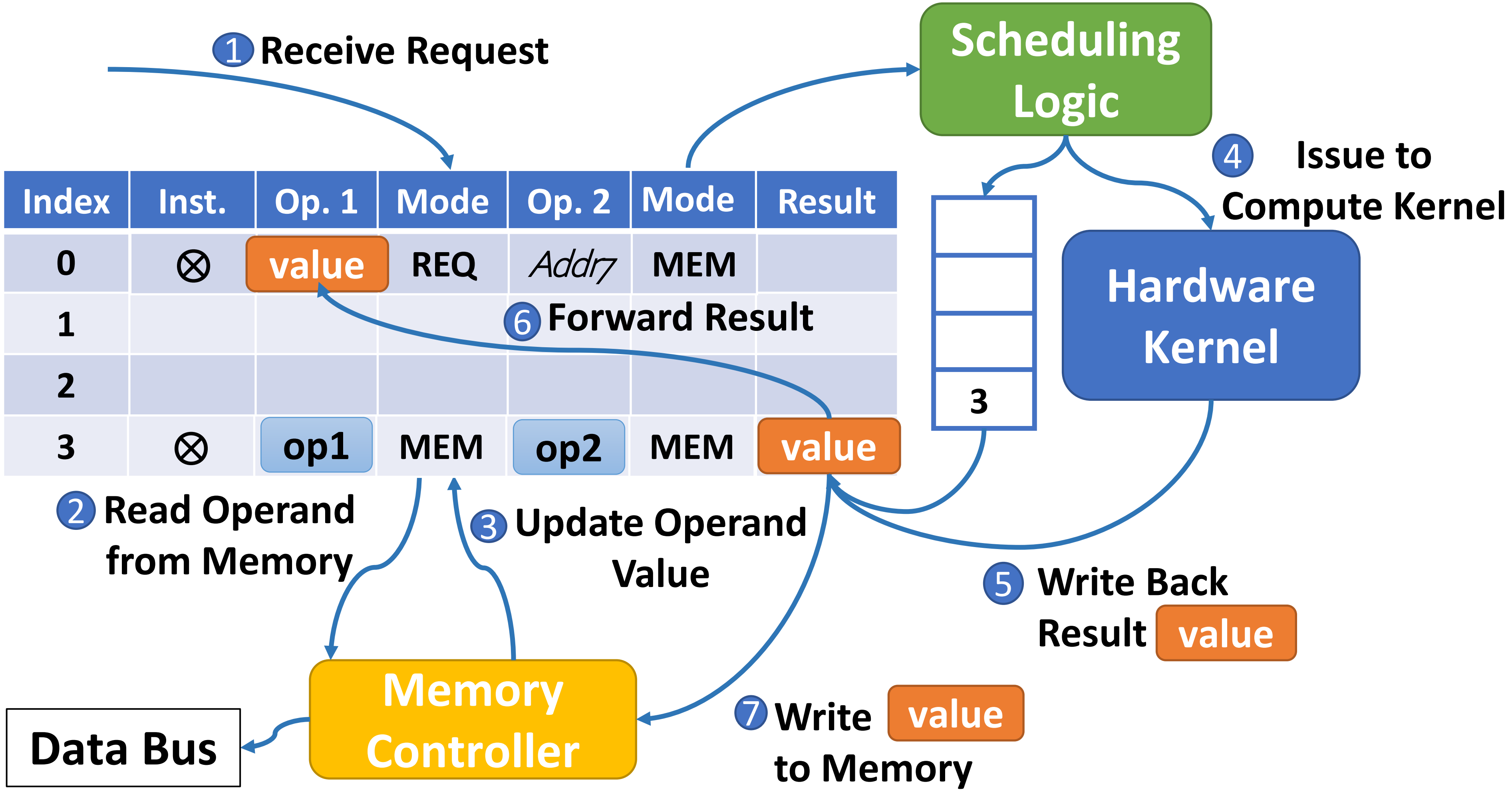
Zipper Hardware Structure



Zipper Hardware Structure



Zipper Hardware Structure



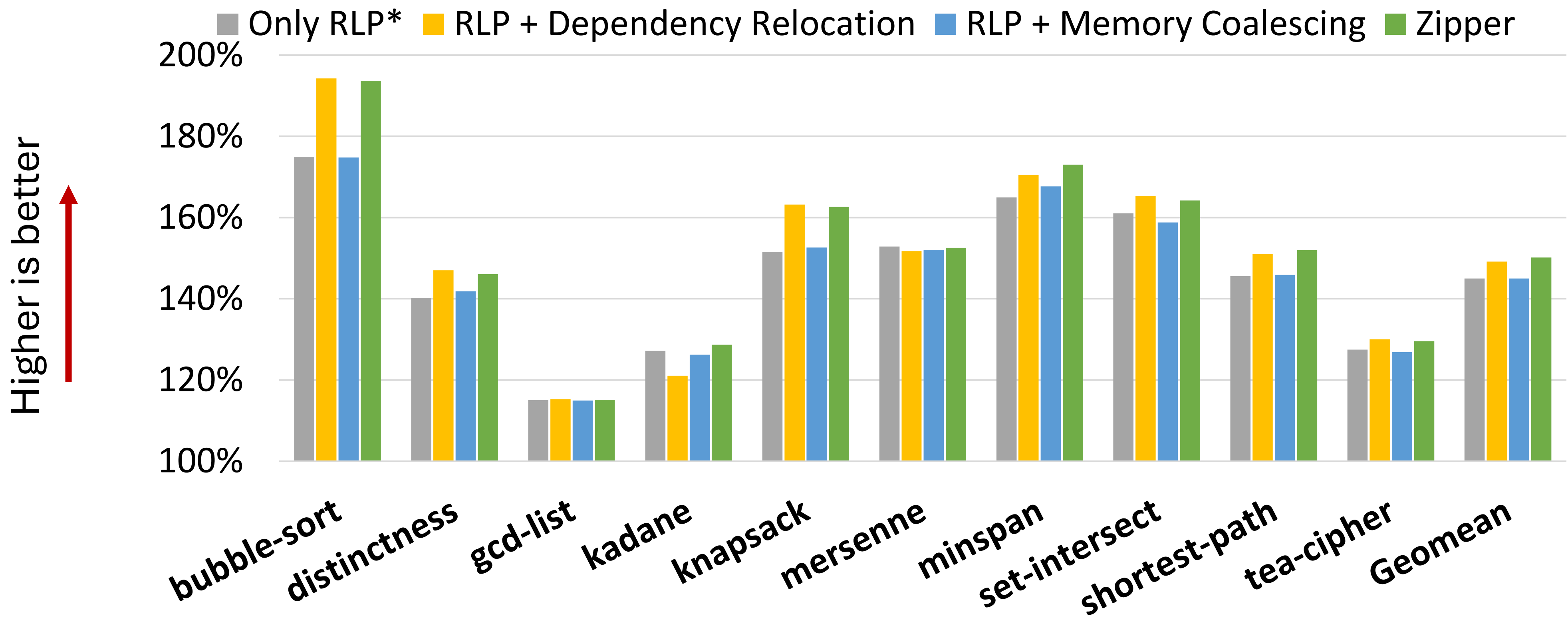
FPGA-Based Evaluation

Experiment Setup

Platform Name	Intel HARP V2
Host CPU	Intel Xeon CPUs (E5-2699v4)
Host Frequency	2.2GHz
FPGA Type	Arria10 GX1150
Interconnect	Intel QuickPath Interconnect (QPI)
Bus Interface	Core Cache Interface(CCI-P)

A Photo of Intel HARP V1

Performance Improvements with Low Area Overhead (1)

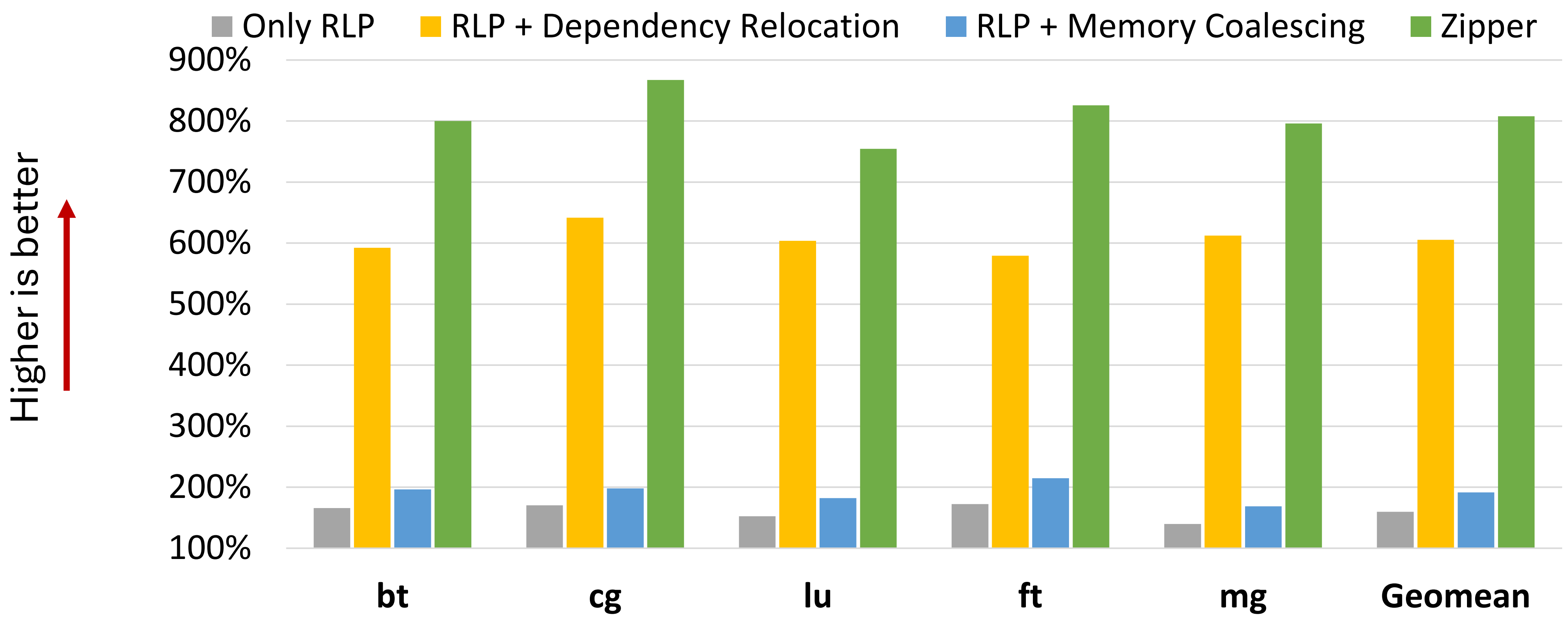


VIP-Bench + Sequestered Encryption Enclave

1.5x Speedup with 0.9% Adaptive Logic Module overhead

*RLP = request-level parallelism

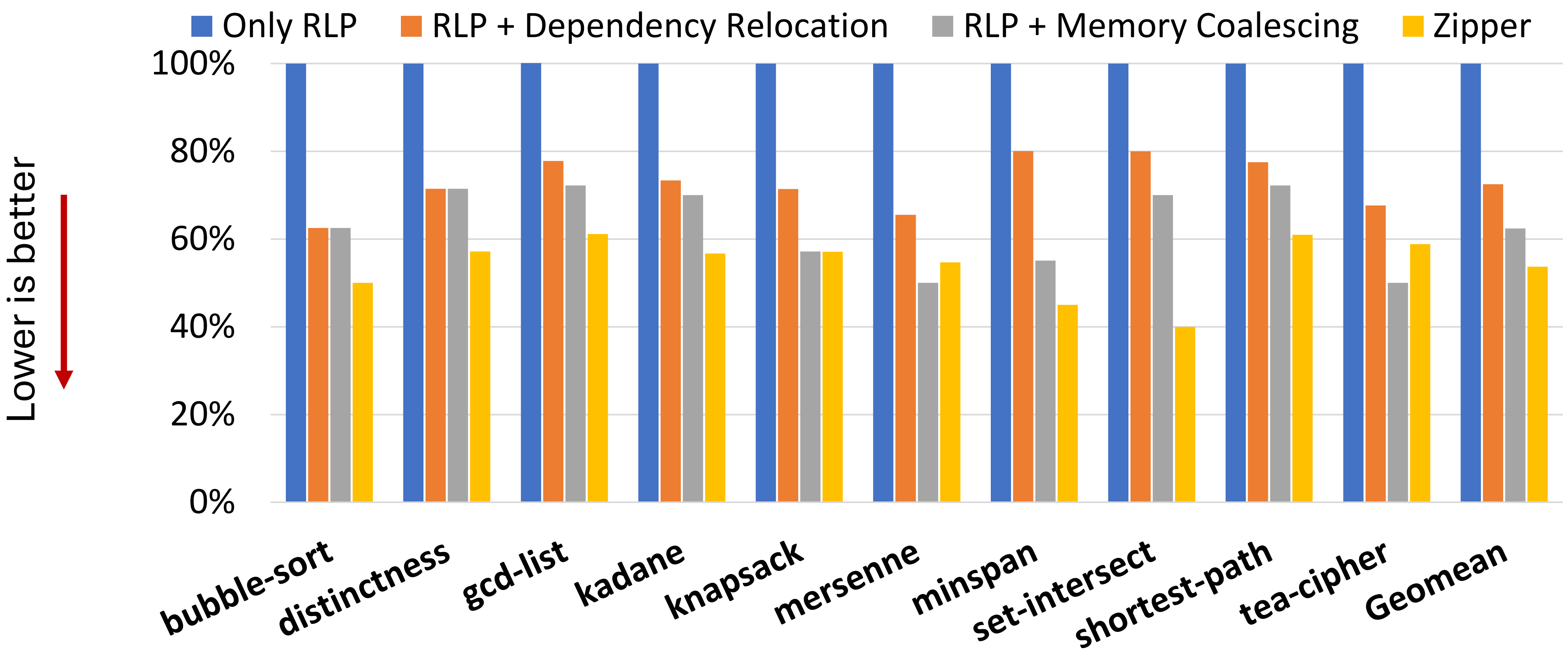
Performance Improvements with Low Area Overhead (2)



NAS Parallel Benchmark + Posit Hardware Kernel

8x Speedup with 4.3% Adaptive Logic Module overhead

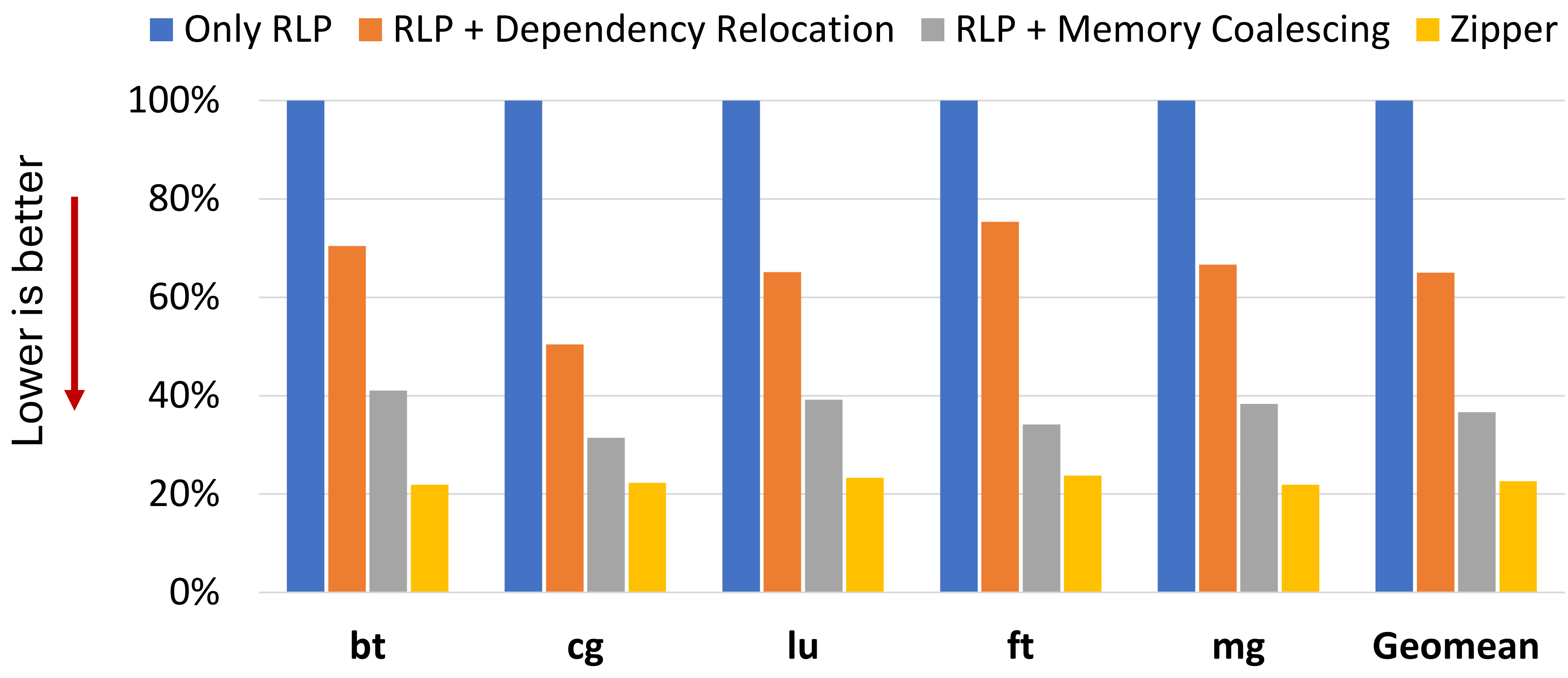
Zipper Improves Performance by Reducing Memory Traffic (1)



VIP-Bench + Sequestered Encryption Enclave

Zipper reduces 46% of bus transactions

Zipper Improves Performance by Reducing Memory Traffic (2)



NAS Parallel Benchmark + Posit Hardware Kernel

Zipper reduces 77% of bus transactions

Conclusions & Looking Ahead

- Communication latency is not getting any lower
- However, they can be tolerated and hidden…
- Zipper achieves, even without any drastic and intrusive changes:
  - On average, 1.5-8X speed-up with <5% area overhead.
  - No compiler changes or intrusive changes to the hardware kernel.
  - Portable to all buses, APIs, and operating systems.
- Zipper is open-sourced @ https://github.com/zipper-bus-optimizations

Questions?