Hi-DMM: High-Performance Dynamic Memory Management in High-Level Synthesis

- Tingyuan Liang, Jieru Zhao, Liang Feng, Sharad Sinha, and Wei Zhang

- Hong Kong University of Science and Technology (HKUST)
- Indian Institute of Technology Goa (IIT Goa)

Outline

- Motivation
- Overview of Hi-DMM
- Implementation of Software
- Implementation of Hardware
- Evaluation of Hi-DMM
- Open-Source Hi-DMM Platform
- Conclusion

Outline

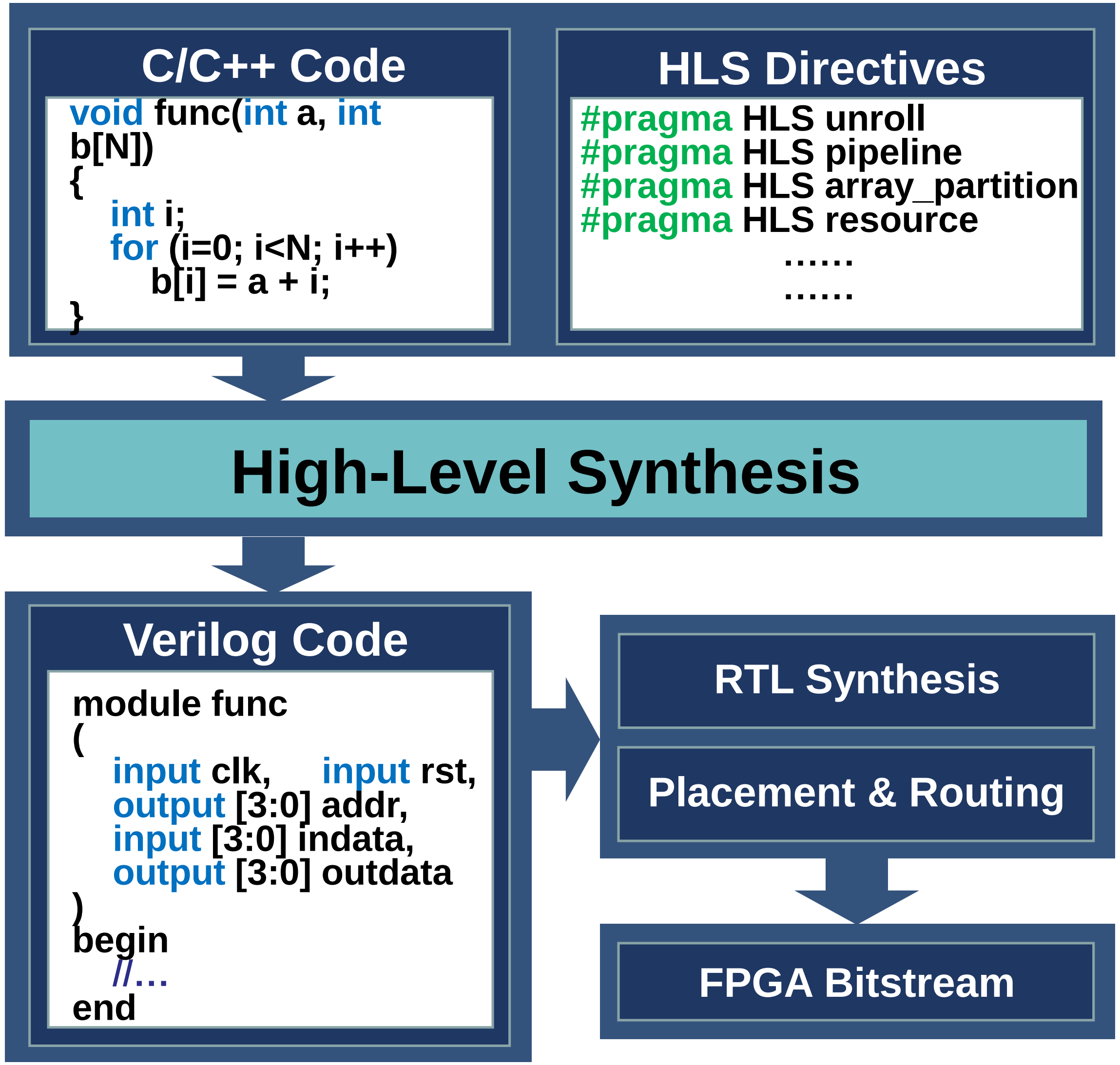
- Motivation
- Overview of Hi-DMM
- Implementation of Software
- Implementation of Hardware
- Evaluation of Hi-DMM
- Open-Source Hi-DMM Platform
- Conclusion

Motivation: High-Level Synthesis

1. Describe Hardware at High Level
   e.g. from C/C++ to Verilog

2. Fast Development of FPGA design

3. Friendly to Complex Applications



Motivation: Dynamic Memory Management

1. Feature of High-Level Language

e.g. malloc(), free(), new, delete

2. Flexible and Efficient

make full use of memory

3. Unsupported by current HLS

if DMM in HLS realized,

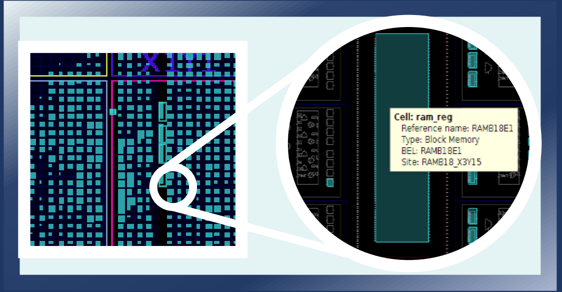
the utilization of BRAMs will be raised

C/C++ Code with DMM

```
int *a = (int *) malloc(n * sizeof(int));
…
free(a);
```

HLS Failure

Block-RAM on FPGA



High Performance

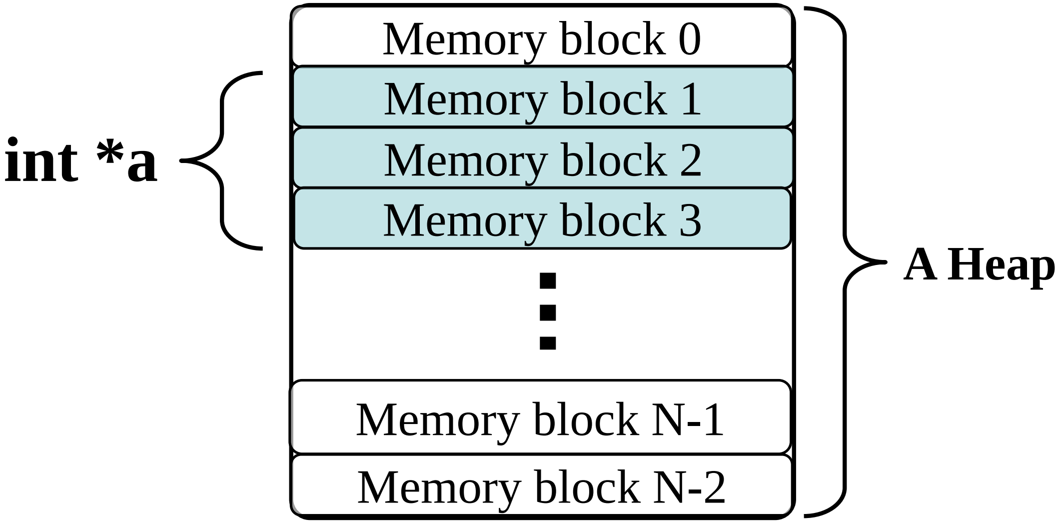
Resource Constraint

Motivation: Challenges for DMM in HLS

```
int *a = (int *) malloc (n * sizeof(int));
```

Where to store dynamic data on FPGA?

Heaps (Block-RAMs on FPGA)



Motivation: Challenges for DMM in HLS

```
int *a = (int *) malloc (n * sizeof(int));
```

Who will manage the dynamic memory on FPGA?

- Record which blocks are used.
- Allocate memory according to required size

Hardware Allocator

1. **Low Latency**
2. **High Utilization of BRAM**
3. **Large Management Capability with Low Overhead of Area**

Motivation: Challenges for DMM in HLS

```
int *a = (int *) malloc (n * sizeof(int));
```

Who will manage the dynamic memory on FPGA?

Hardware Allocator

1. **Low Latency**
2. **High Utilization of BRAM**
3. **Large Management Capability with Low Overhead of Area**

SysAlloc / DMM-HLS

Hundreds of cycles?

Fixed-size Allocation?

DOMMU

AMMU

High overhead?

Motivation: Challenges for DMM in HLS

int *a = (int *) malloc (n * sizeof(int));

Who will manage the dynamic memory on FPGA?

Hardware Allocator

Hi-DMM Allocators

1. Low Latency
2. High Utilization of BRAM
3. Large Management Capability with Low Overhead of Area

Motivation: Challenges for DMM in HLS

```
int *a = (int *) malloc (n * sizeof(int));
```

How can it **be synthesized** into a **high-performance** design with **HLS features**?

Source Code Compiler

1. **Source Code Transformation**
2. **Resource Mapping**
3. **Adaption to HLS Directives**

Previous Works

Automatic Transformation? ✗
Resource Mapping? ✗
HLS Directive Compatibility? ✗

Motivation: Challenges for DMM in HLS

```
int *a = (int *) malloc (n * sizeof(int));
```

How can it **be synthesized** into a **high-performance** design with **HLS features**?

Source Code Compiler

1. Source Code Transformation
3. Adaption to HLS Directives

Hi-DMM Compiler ✓

Motivation: Hi-DMM

```
int *a = (int *) malloc (n * sizeof(int));
```

Who will manage the dynamic memory on FPGA?

How can it **be synthesized** into a **high-performance** design with **HLS features**?

Hardware Allocator

Hi-DMM Allocators ✓

Source Code Compiler

Hi-DMM Compiler ✓

Hi-DMM

Outline

- Motivation
- Overview of Hi-DMM
- Implementation of Software
- Implementation of Hardware
- Evaluation of Hi-DMM
- Open-Source Hi-DMM Platform
- Conclusion

Overview: Workflow of Hi-DMM

Compilation-time

Source Code of Accelerator with DMM

```
int *a = (int *) malloc(n * sizeof(int));
…
free(a);
```

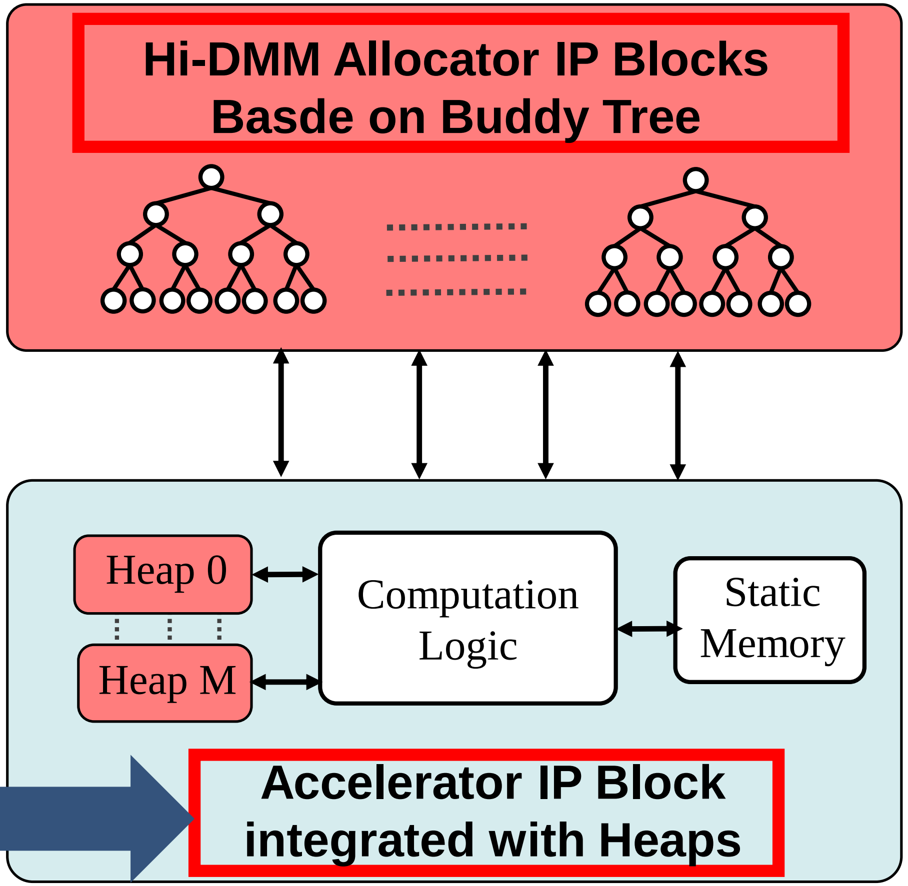
Hi-DMM Compiler

Source Code Compatible with HLS

```
int offset_a = Hi_malloc(n, allocator1);
…
Hi_free(offset_a);
```

High-Level Synthesis

Run-time



Overview: Workflow of Hi-DMM

Compilation-time

Source Code of Accelerator with DMM

```
int *a = (int *) malloc(n * sizeof(int));
…
free(a);
```

Hi-DMM Compiler

Source Code Compatible with HLS

```
int offset_a = Hi_malloc(n, allocator1);
…
Hi_free(offset_a);
```

High-Level Synthesis

Run-time

Un-synthesizable ✗

Based on Clang

Synthesizable ✓

Accelerator IP Block integrated with Heaps

Static Memory

Overview: Workflow of Hi-DMM

Compilation-time

Source Code of Accelerator with DMM

```
int *a = (int *) malloc(n * sizeof(int));
...
```

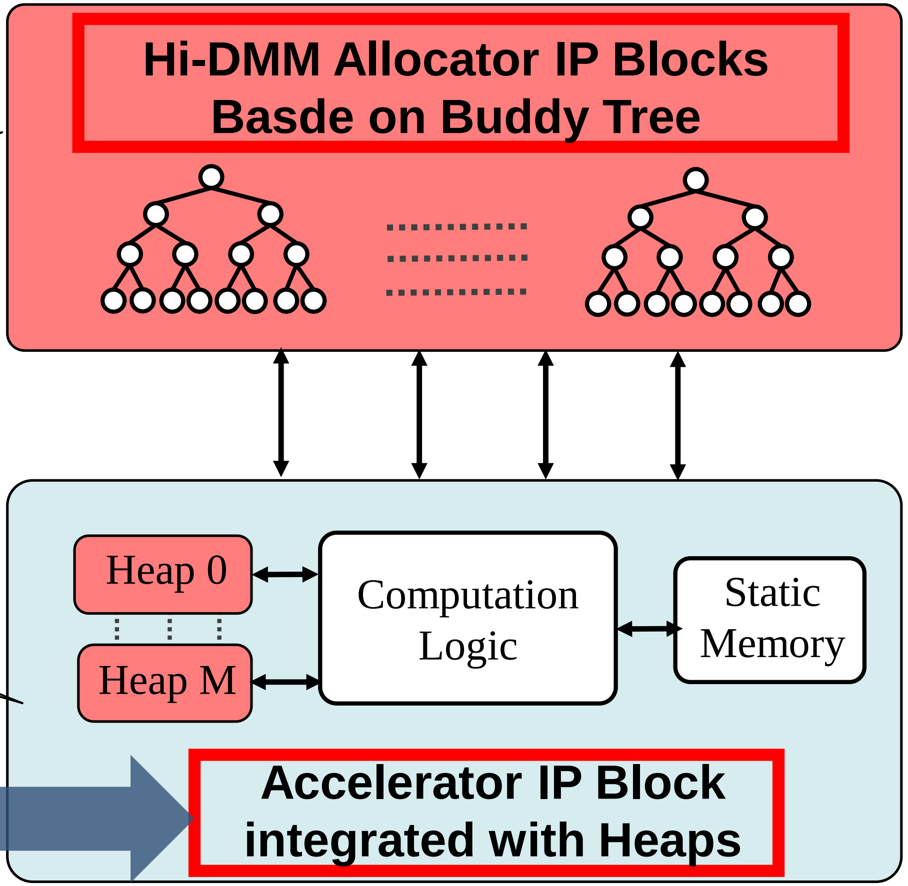
Source Code Compatible w...

```
int offset_a = Hi_malloc(n, allocator1);
...
Hi_free(offset_a);
```

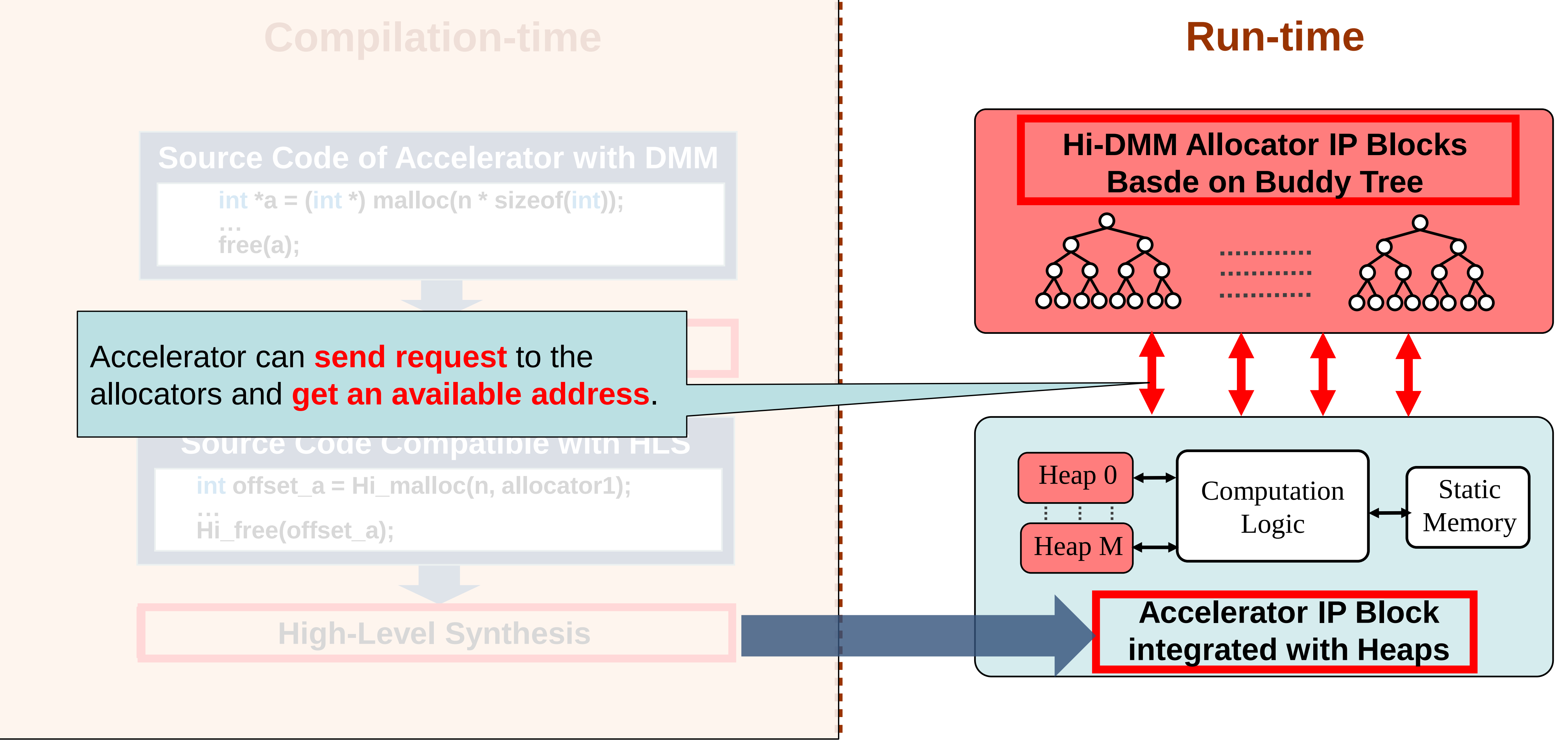
High-Level Synthesis

Both of the accelerator and the allocators are **described in C** and synthesized by Vivado HLS.

Run-time



Overview: Workflow of Hi-DMM



Overview: Workflow of Hi-DMM

Compilation-time

Source Code of Accelerator with DMM

```
int *a = (int *) malloc(n * sizeof(int));
…
free(a);
```

↓

Hi-DMM Compiler

↓

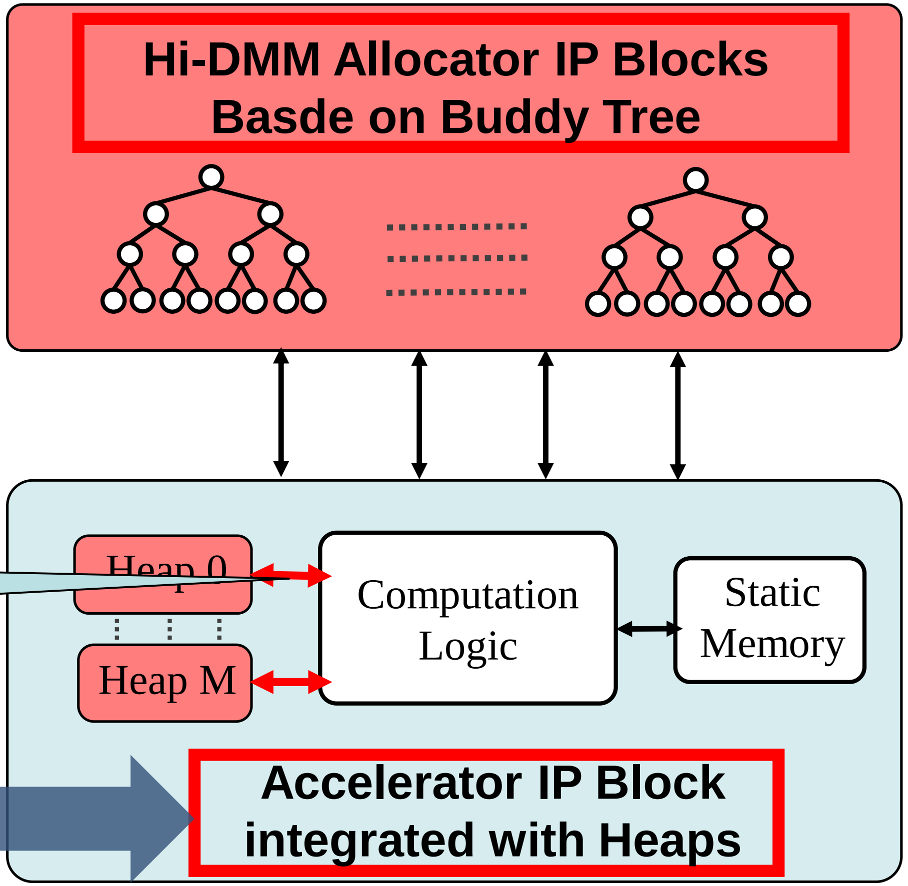
Source Code Compatible with HLS

Then the accelerator can **access the heaps** with the address directly.

↓

High-Level Synthesis

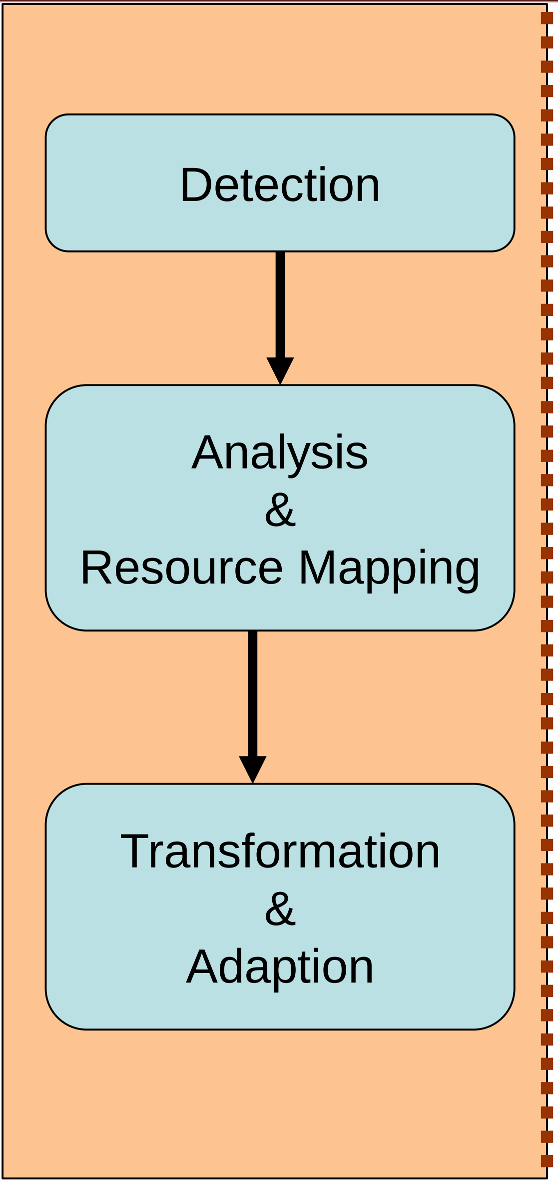
Run-time



Outline

- Motivation
- Overview of Hi-DMM
- Implementation of Software
- Implementation of Hardware
- Evaluation of Hi-DMM
- Open-Source Hi-DMM Platform
- Conclusion

Software: Hi-DMM Compiler



Software: Hi-DMM Compiler

Detection

Analysis & Resource Mapping

Transformation & Adaption

Definition of Pointers

```
int *a, *b, c;
ap_unint<13> *e, f, *g;
```

```
float *x, *y, z;
User_Struct *h, *k;
```

1. Name
2. Type
3. Width

Allocation Function Call

```
a = (int*) malloc(n*sizeof(int));
e = (ap_unint<13> *) malloc(100*sizeof(ap_unint<13> ));
h = (User_Struct *) malloc(sizeof(User_Struct ));
```

4. Allocation Requester
5. Granularity

Access to Pointers

```
a[i] = b[j] + c;
e = g;
```

```
h->next = k;
h->val = 123;
```

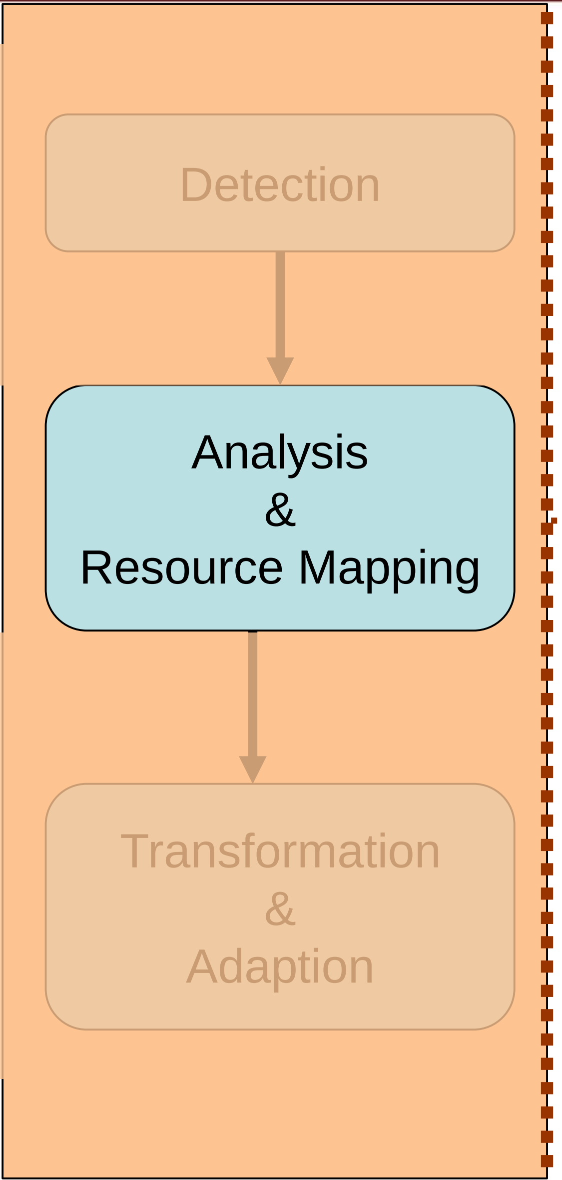
6. Type of Access
7. Dependencies

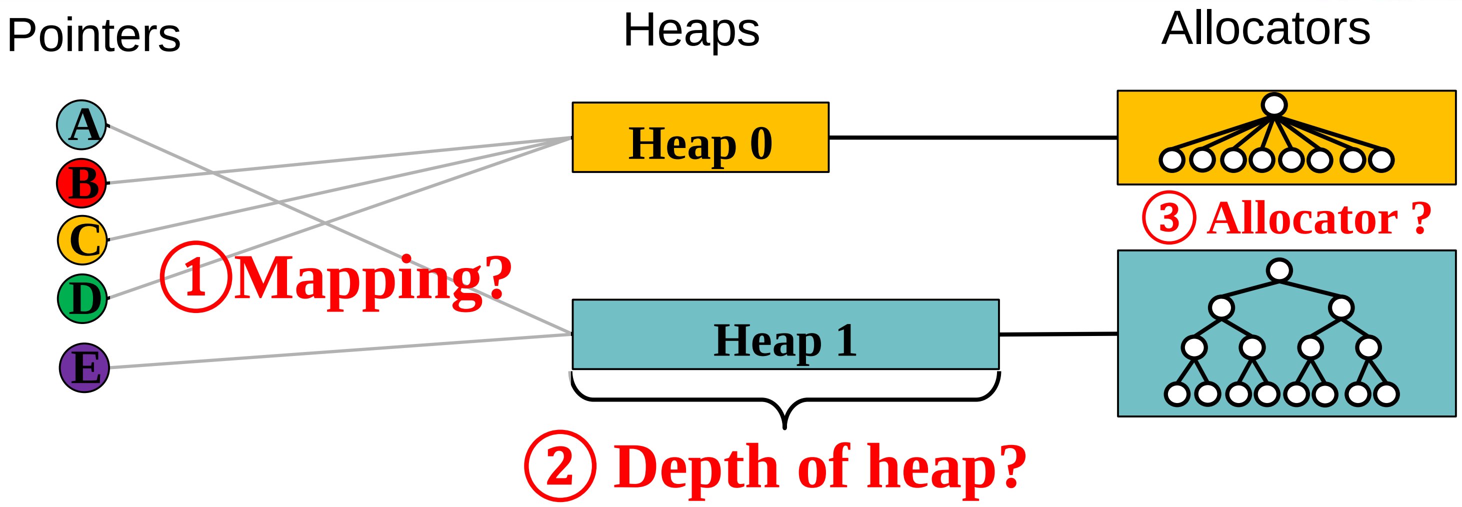
HLS Directives

```
#pragma HLS array_partition variable=xxx factor=xxx
#pragma HLS unroll factor=xxx
```

8. Directive for Pointers
9. Directive involving Pointers

Software: Hi-DMM Compiler



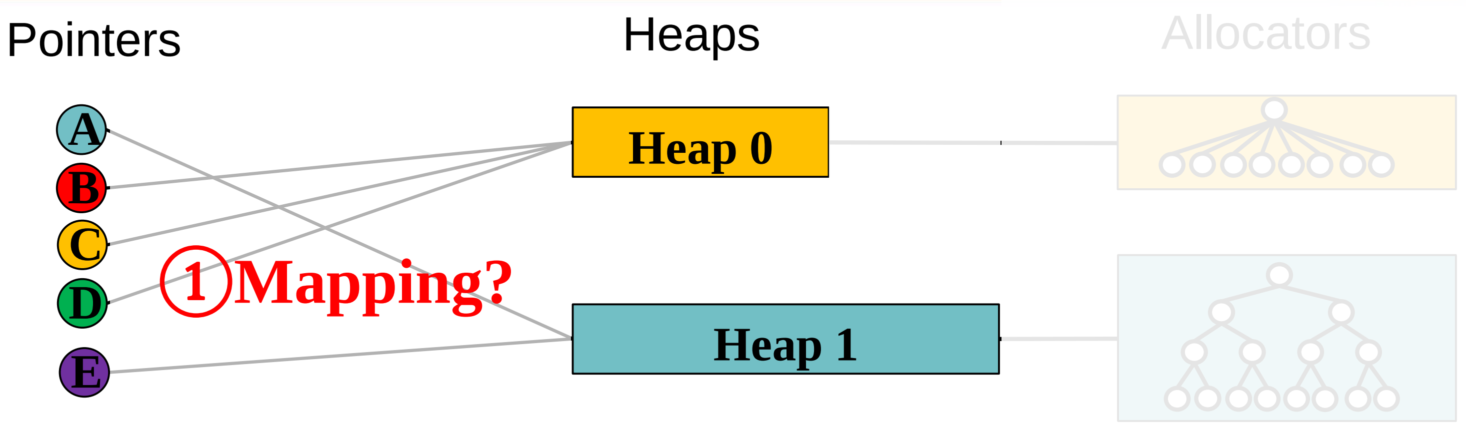


Software: Hi-DMM Compiler

Detection

Analysis & Resource Mapping

Transformation & Adaption

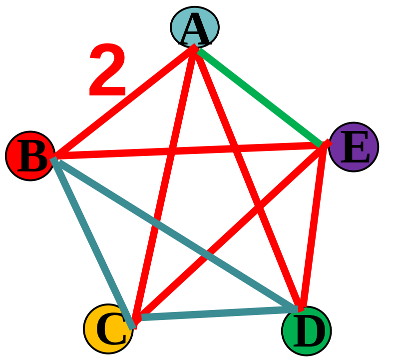


1. Map Pointers to heaps

Co-operations between pointers:

A[i] = B[j] + C[i];
…
D[k] = A[i] + B[m];
…
2 co-operations between A and B

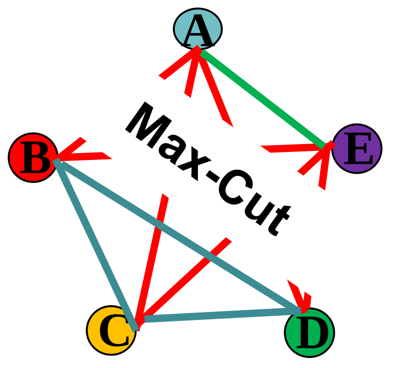
Fully-Connected Graph



Karger Algorithm

Max-Cuts

Separated Sub-graphs

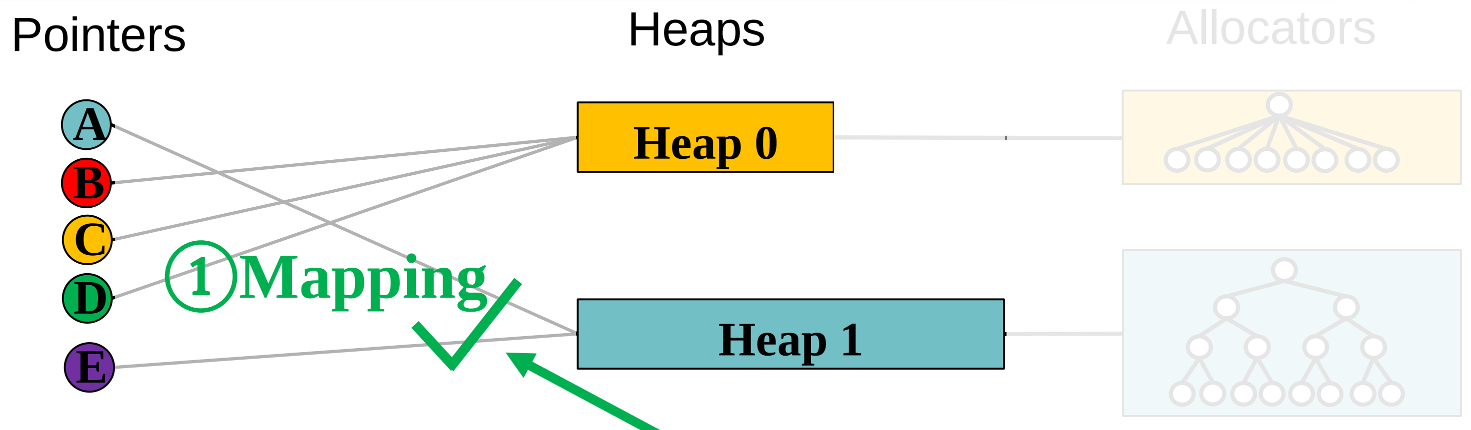


Software: Hi-DMM Compiler

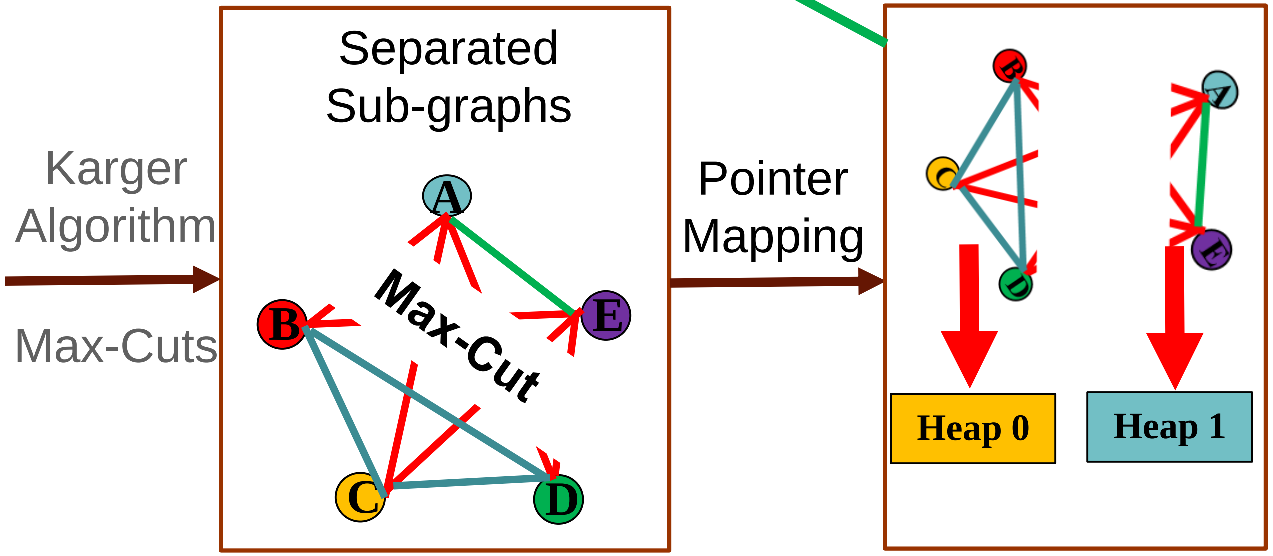
Detection

Analysis & Resource Mapping

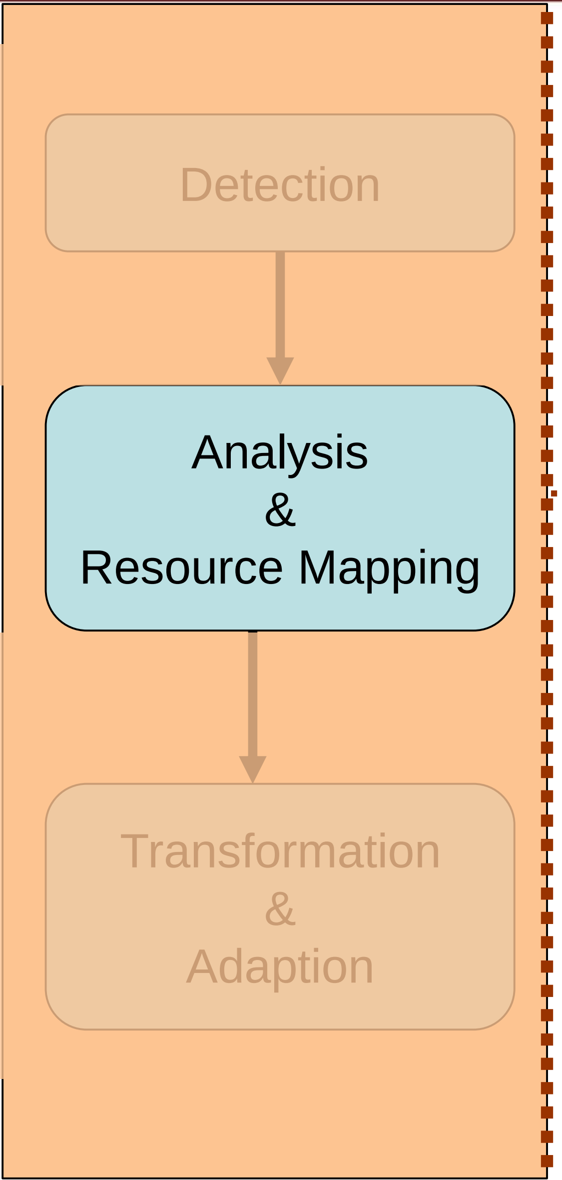
Transformation & Adaption

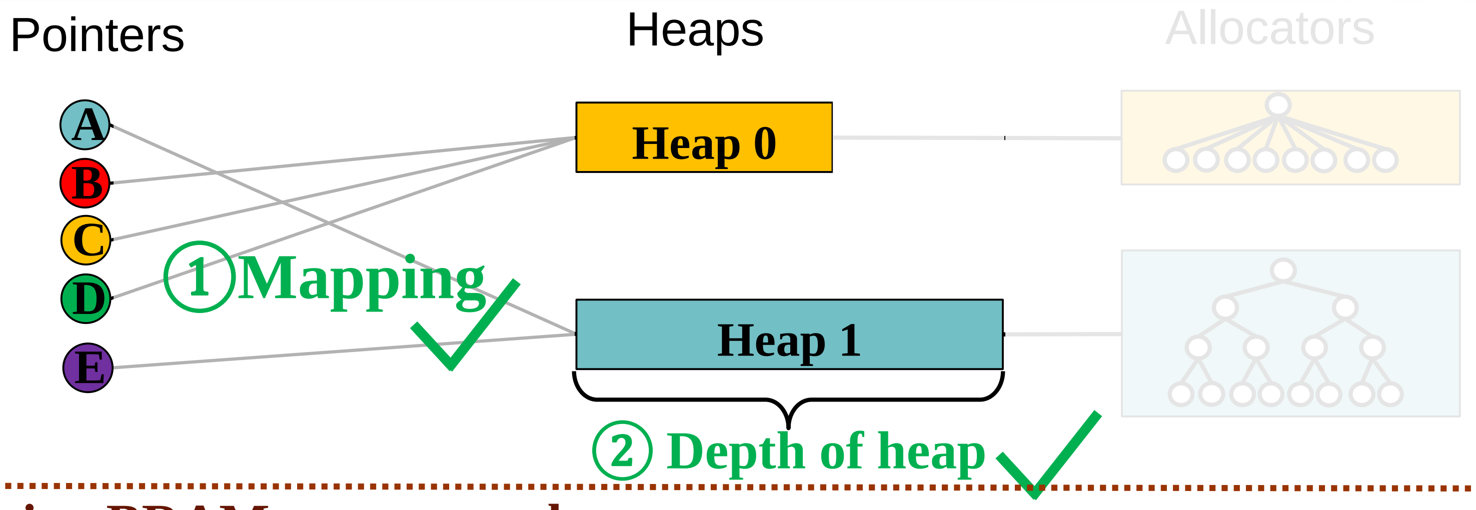


1. **Map Pointers to heaps**

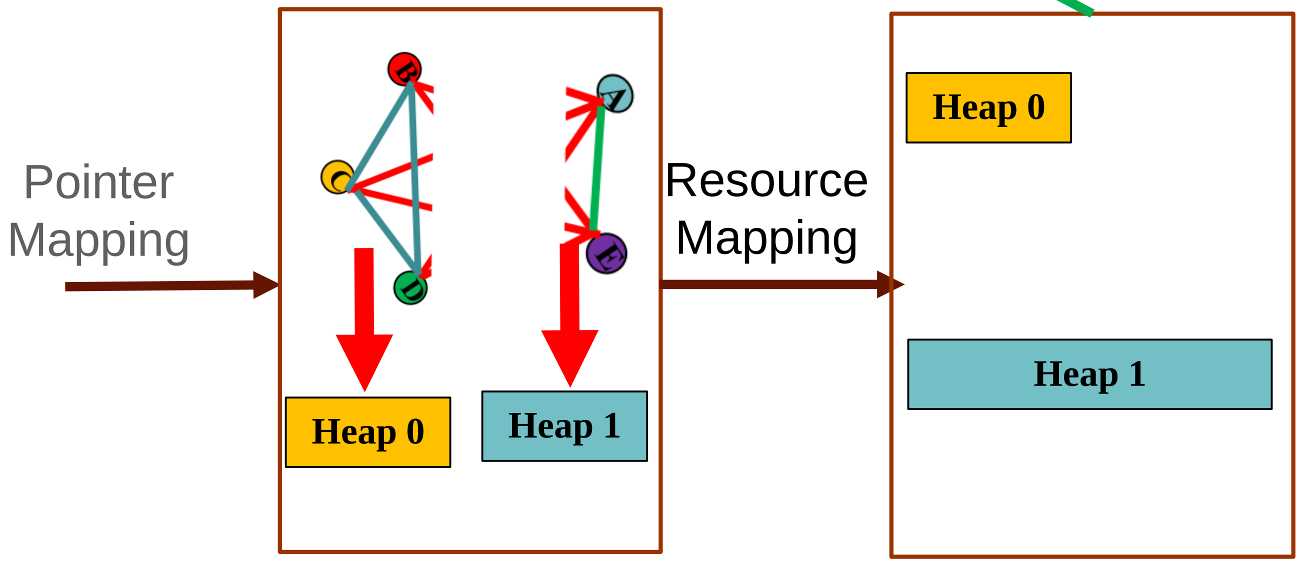


Software: Hi-DMM Compiler

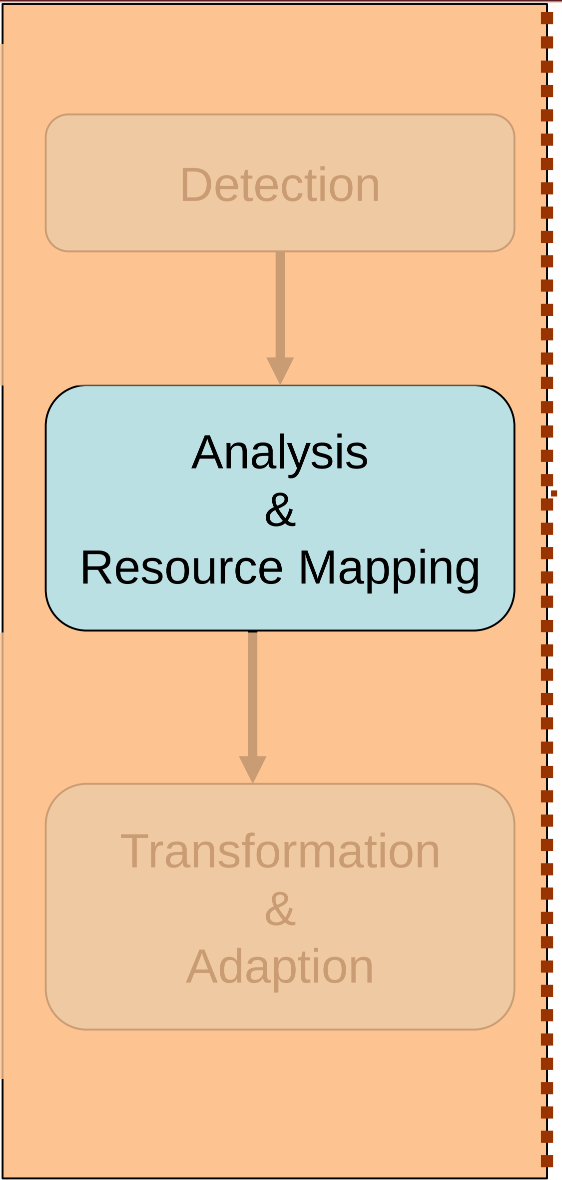


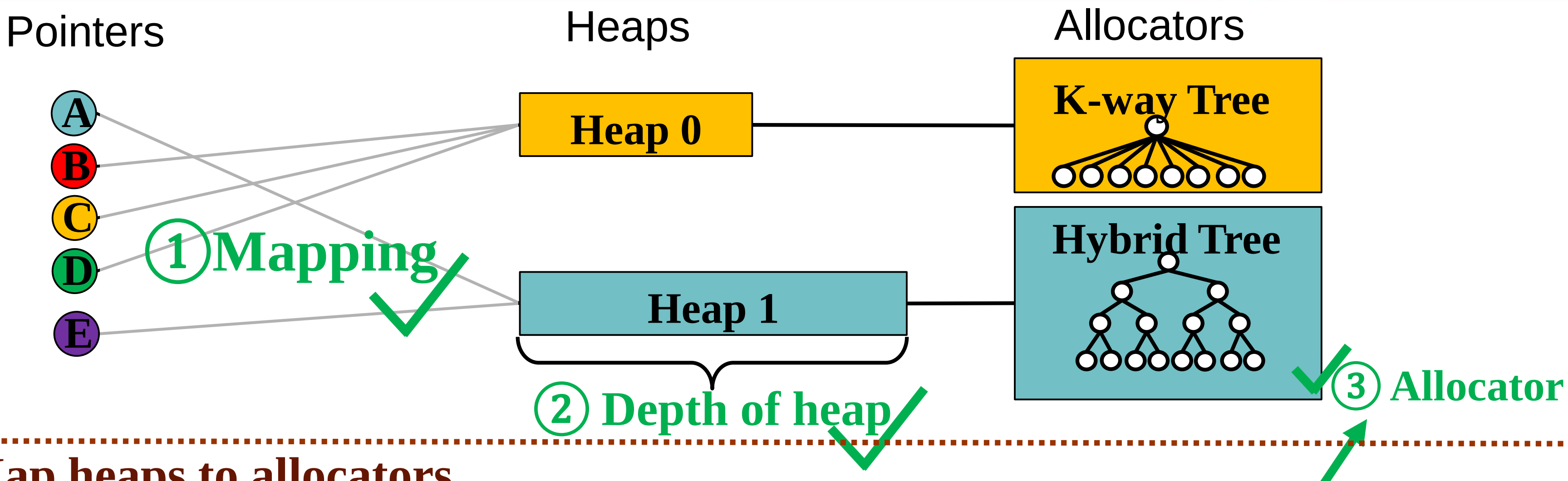


2. Assign BRAM resource to heaps

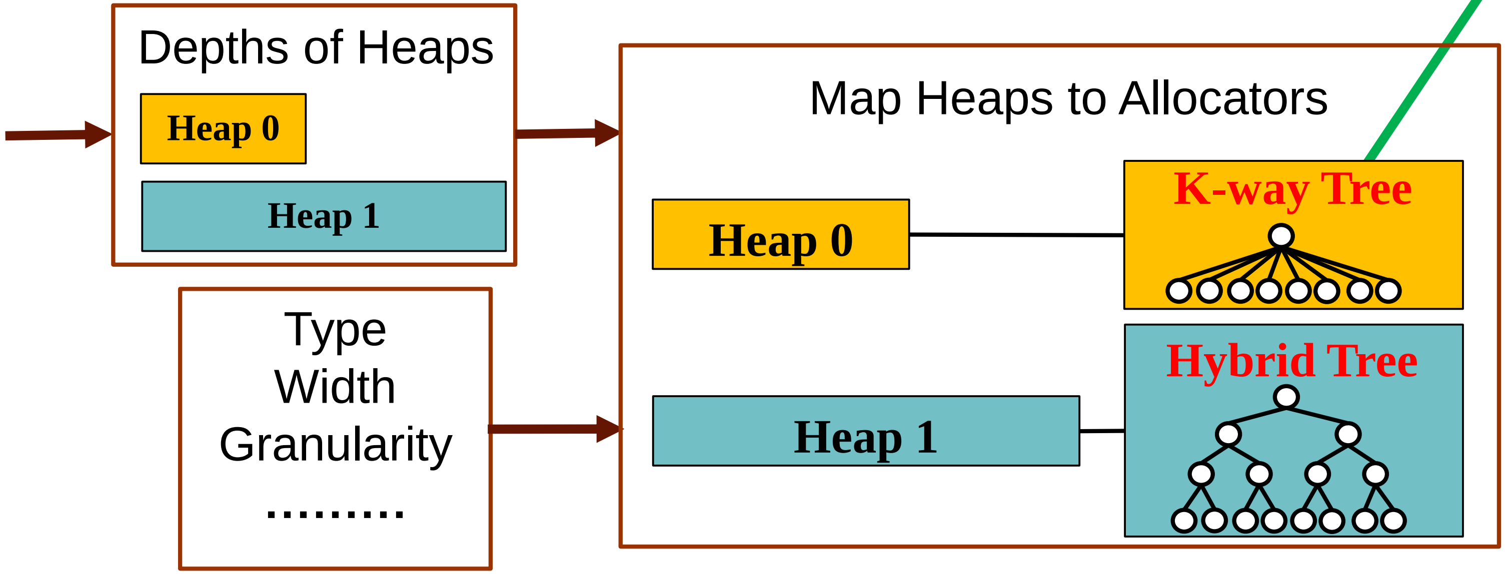


Software: Hi-DMM Compiler





3. Map heaps to allocators



Software: Hi-DMM Compiler

Detection

↓

Analysis
&
Resource Mapping

↓

**Transformation
&
Adaption**

1. Transformation

Definition of DMM Heaps

```
int Hi_DMM_Heap_0[8192];
ap_uint<7> Hi_DMM_Heap_1[2048];
#pragma HLS array_partition variable=Hi_DMM_Heap_1 cyclic  factor=4
```

DMM Interface

```
void TOP(hidmm_alloc_port *Hi_DMM_allocator_0)
{
#pragma HLS interface ap_hs port = Hi_DMM_allocator_0
   ….
}
```

Synthesizable ✓

Function Calls

```
offset_local_dis = HLS_malloc<8192>(n, Hi_DMM_allocator_1_Super_HTA8k;
```

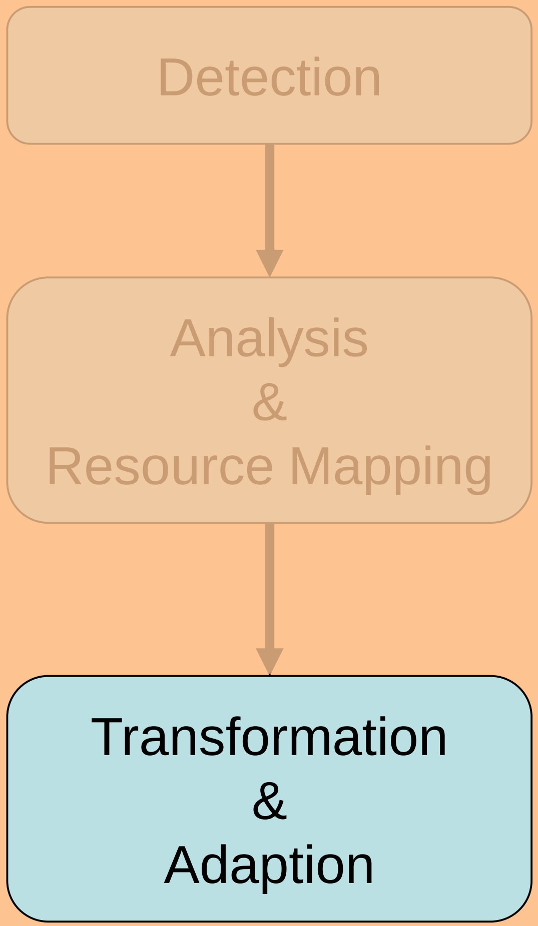
Accesses to “Struct” Pointers

```
//head->VAL = data[0];
head[OFFSET_LIST_VAL] = data[0];
```

Assignment of Pointers (from one to another one)

```
// now = tail;
offset_now = offset_tail;
```

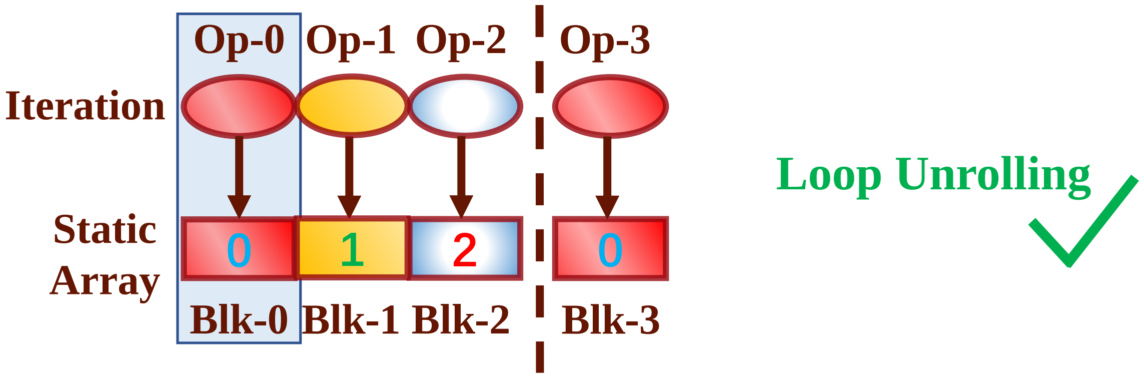
Software: Hi-DMM Compiler



2. Adaption to HLS Directives

e.g. Loop Transformation for Loop Unrolling

Example without DMM: Operations mapped to corresponding partitions



Software: Hi-DMM Compiler

Detection

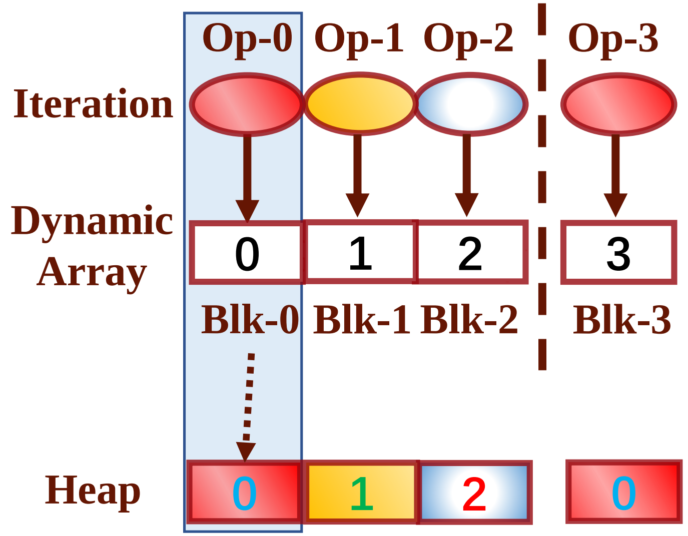
Analysis & Resource Mapping

Transformation & Adaption

2. Adaption to HLS Directives

e.g. Loop Transformation for Loop Unrolling

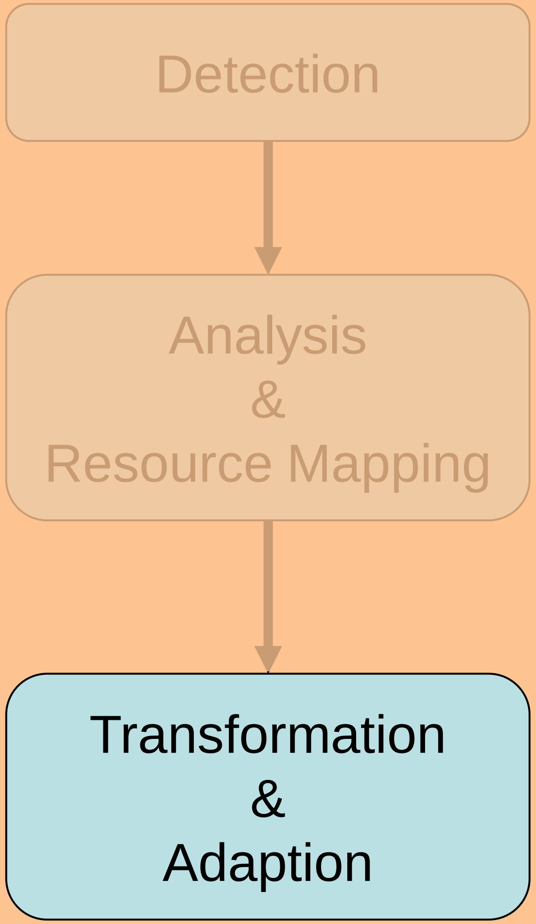
Example with DMM: Operations mapped to all partitions



Loop Unrolling ✗

Location of dynamic array is unknown.

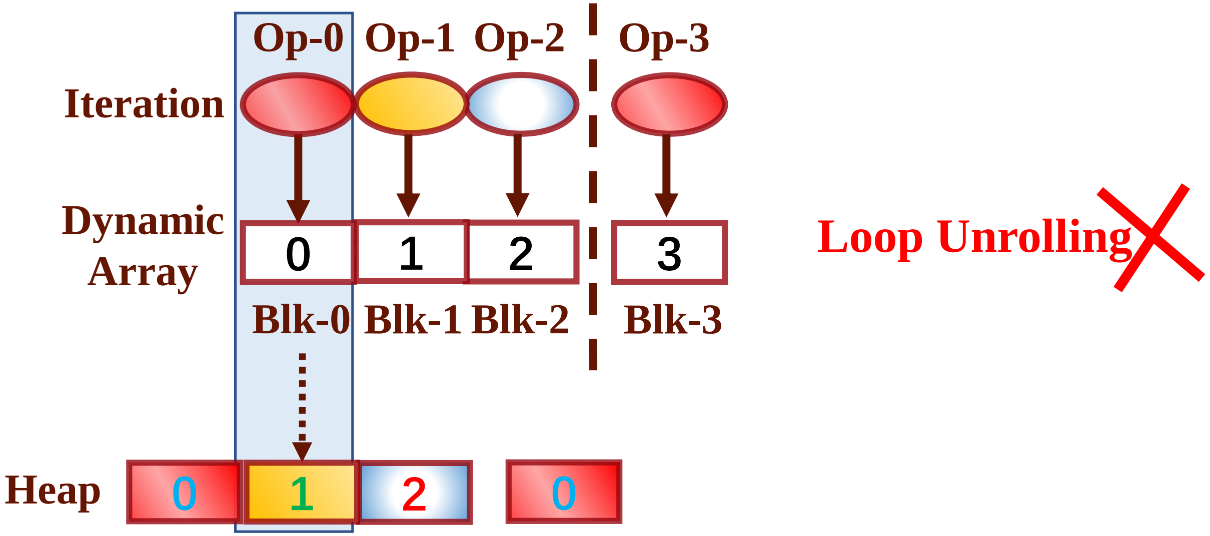
Software: Hi-DMM Compiler



2. Adaption to HLS Directives

e.g. Loop Transformation for Loop Unrolling

Example with DMM: Operations mapped to all partitions



Location of dynamic array is unknown.

Software: Hi-DMM Compiler

Detection

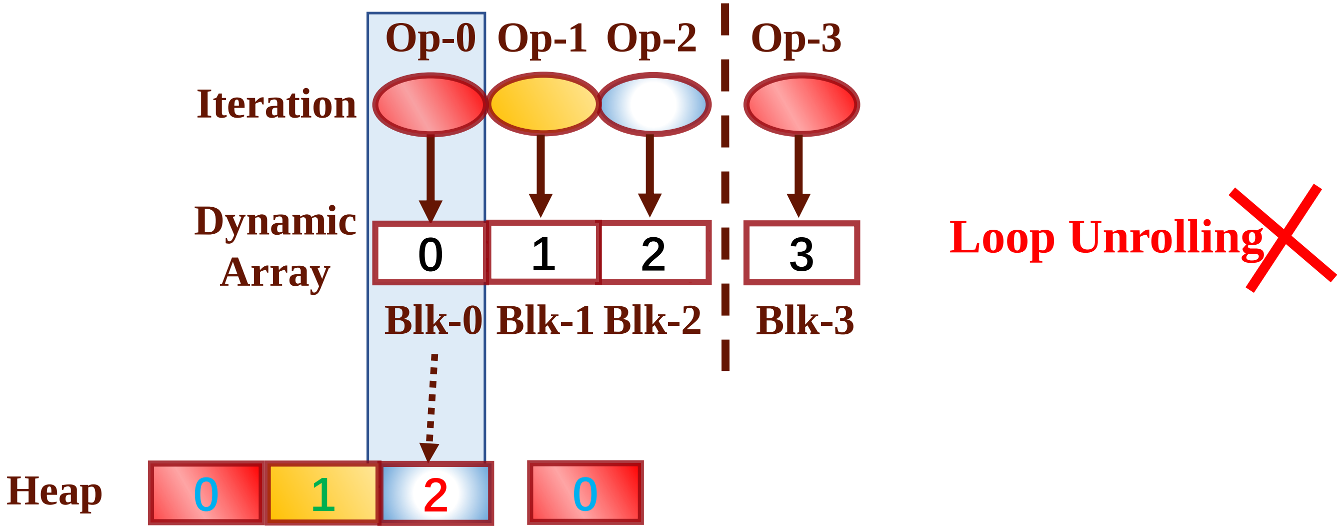
Analysis & Resource Mapping

Transformation & Adaption

2. Adaption to HLS Directives

e.g. Loop Transformation for Loop Unrolling

Example with DMM: Operations mapped to all partitions



Location of dynamic array is unknown.

Software: Hi-DMM Compiler

Detection

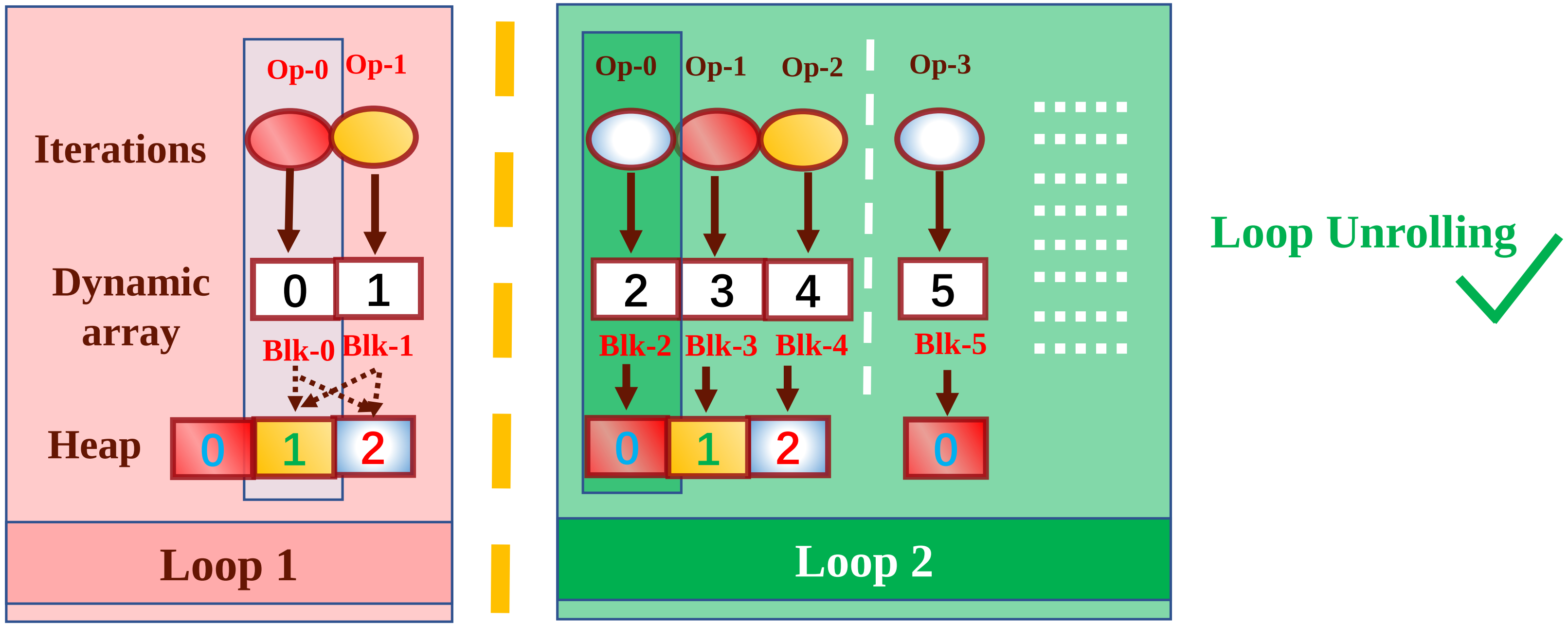
Analysis & Resource Mapping

Transformation & Adaption

2. Adaption to HLS Directives

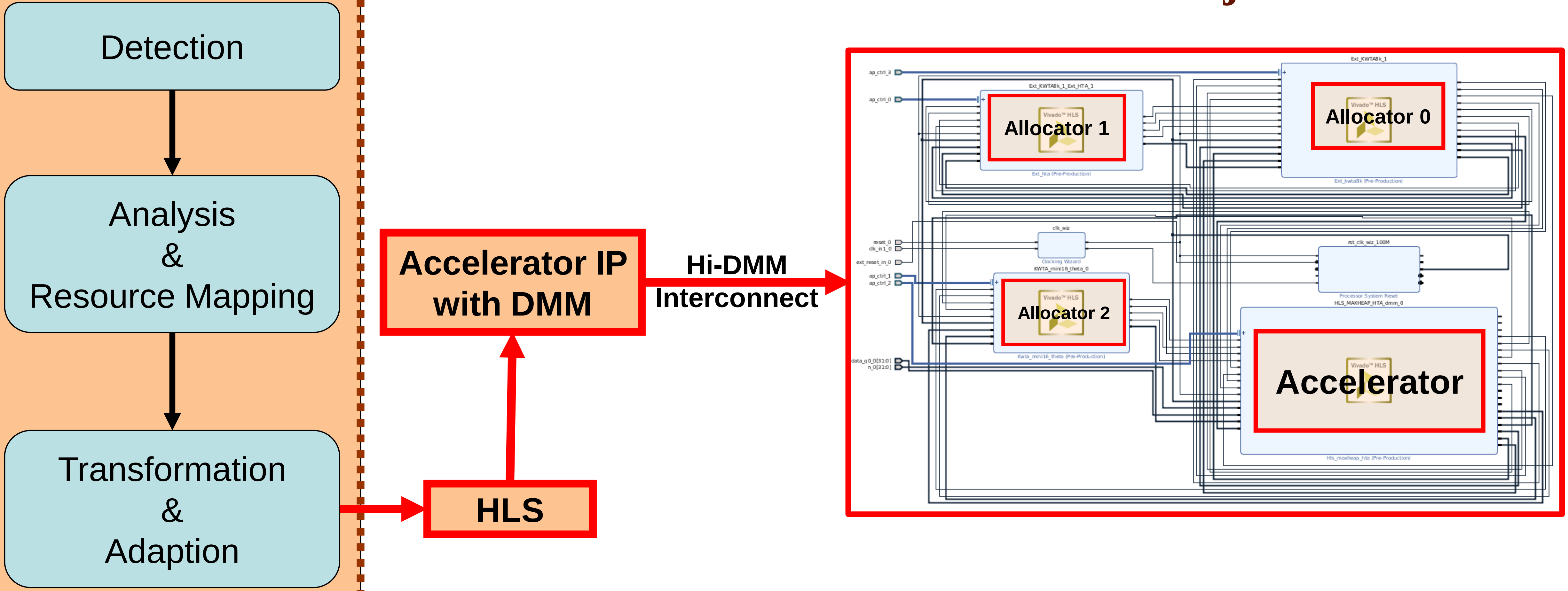
e.g. Loop Transformation for Loop Unrolling

Solution: Loop Splitting



Software: Hi-DMM Compiler

Generated Vivado Project



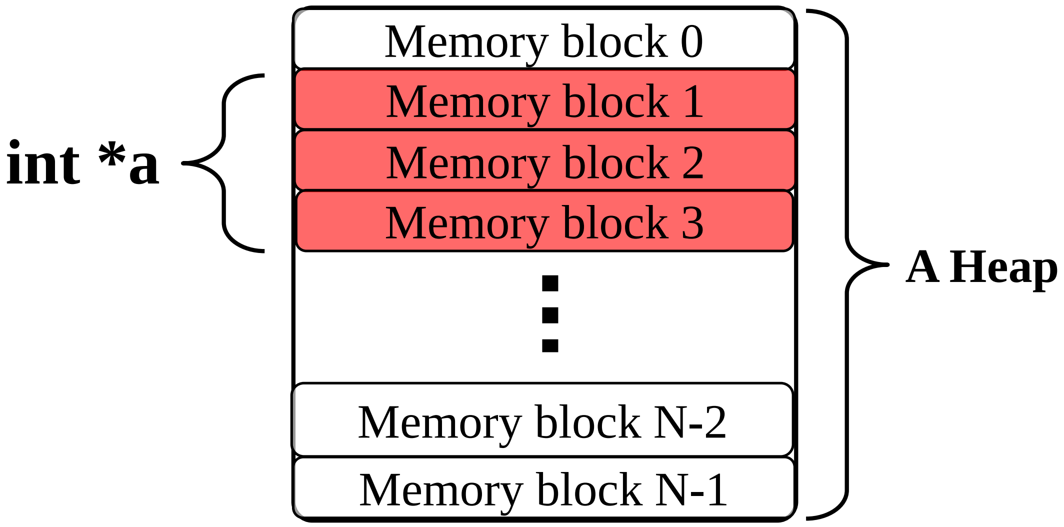
Outline

- Motivation
- Overview of Hi-DMM
- Implementation of Software
- Implementation of Hardware
- Evaluation of Hi-DMM
- Open-Source Hi-DMM Platform
- Conclusion

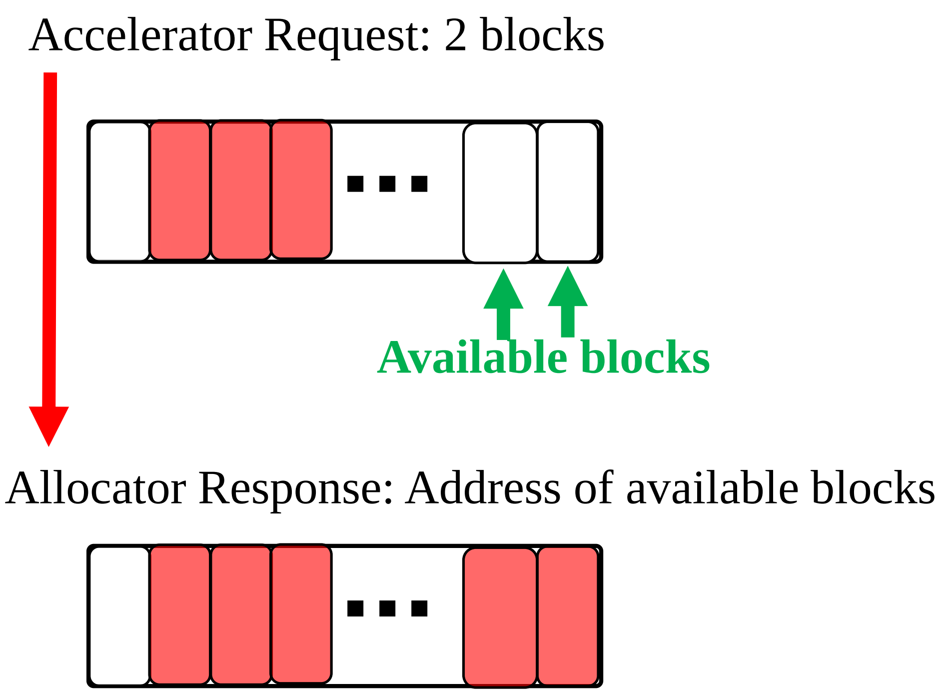
Hardware: Hi-DMM Allocators

What do allocators do?

1. Record which blocks in the heap are used



2. Allocate memory according to the size



Hardware: Hi-DMM Allocators

Compare with Previous Works

Previous Mechanism	Allocation Latency	Resource Cost	Memory Efficiency
Buddy Tree	medium	high	high
Fixed-Size Blocks	high	low	low
Free List	high	medium	high

Hi-DMM allocators are based on buddy tree.

Proposed Mechanism	Allocation Latency	Resource Cost	Memory Efficiency
Hi-DMM Allocators	low	low	high

Hardware: Hi-DMM Allocators

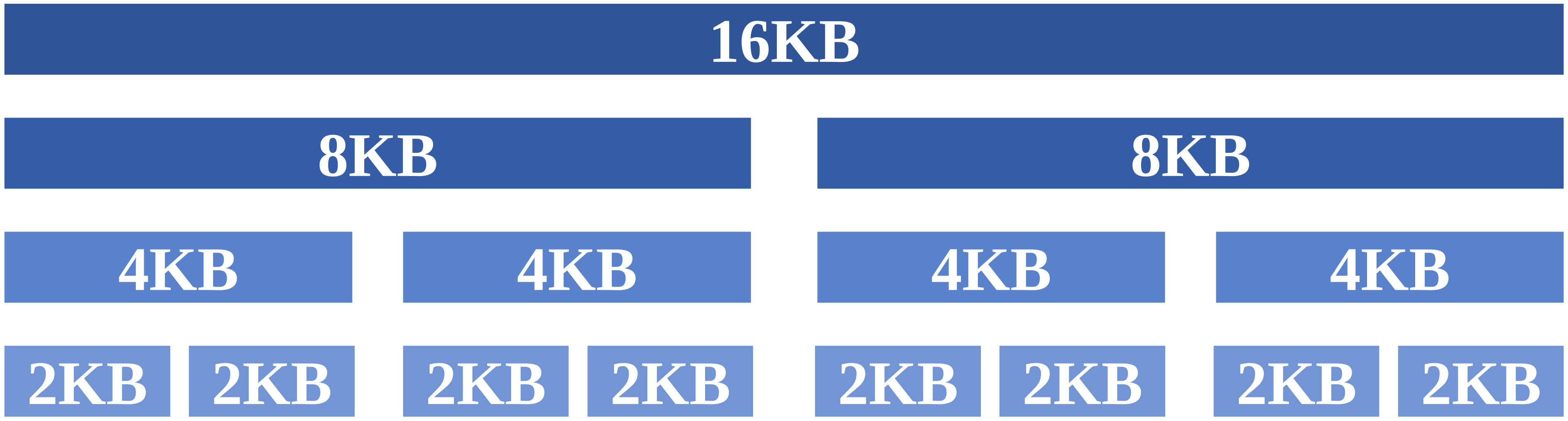
Compare among Hi-DMM Allocators

Hi-DMM Mechanism	Allocation Latency	Resource Cost	Highlights
Fast Buddy Tree	Very low (~10 cycles)	Low	Fast allocator for small heap
Pre-Allocation Tree	Very low (~5 cycles)	Low	Pre-allocate a block for next request
Hybrid Tree	Low (~20 cycles)	Very low	Very large management capability
K-Way Tree	Extremely low (~1 cycles)	Extremely Low	Fixed-size allocator for user-defined struct

Meet the requirements of various applications

Hardware: Buddy Tree

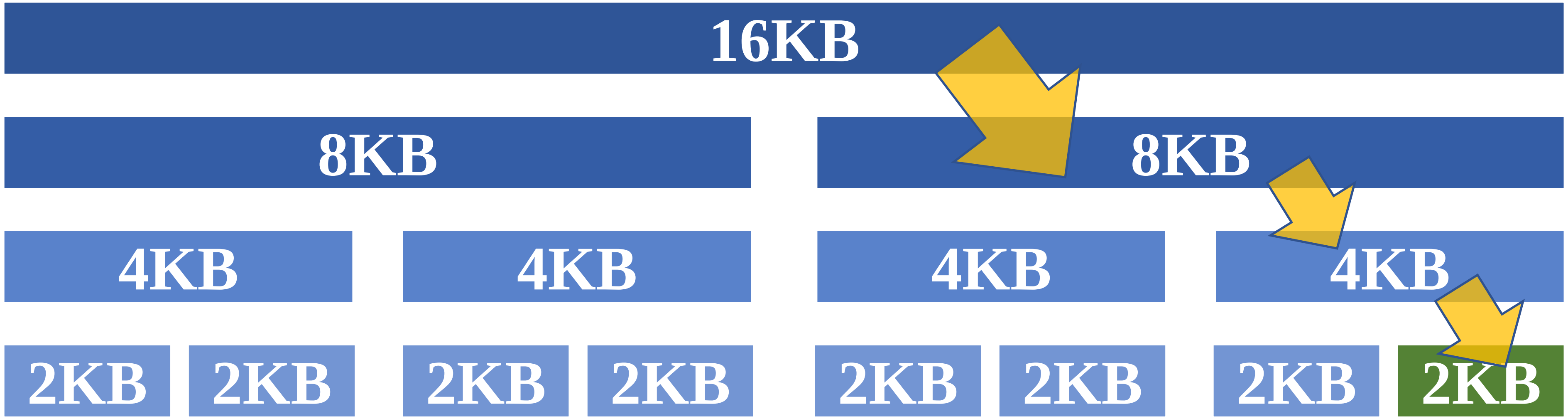
- **Conventional Buddy Tree Allocation:**
  - Splits the entire space of heap repetitively in half to find an available memory block best fitting the size of request.



Hardware: Buddy Tree

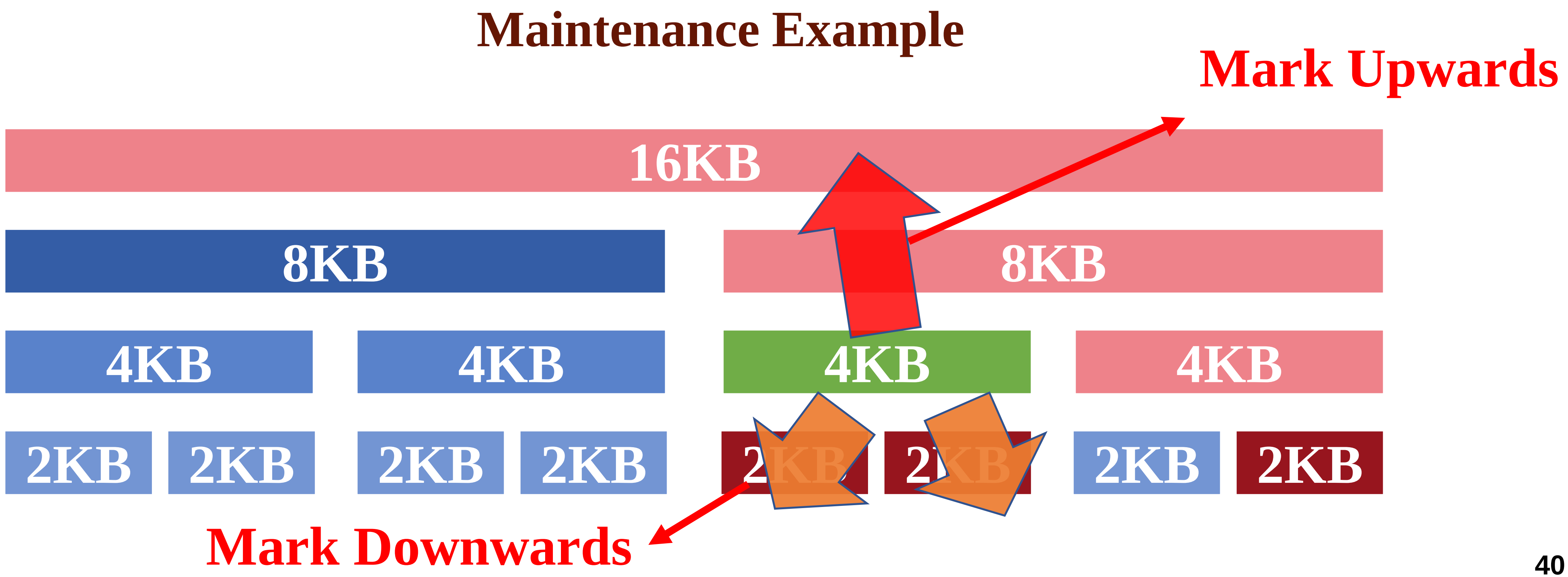
- **Conventional Buddy Tree Allocation:**

Allocation Example



Hardware: Buddy Tree

- **Conventional Buddy Tree Allocation:**



Hardware: Hi-DMM Allocators

- **Fast Buddy Tree Allocator (FBTA):**

1. Allocation *without searching* layer by layer, based on *bit operation*

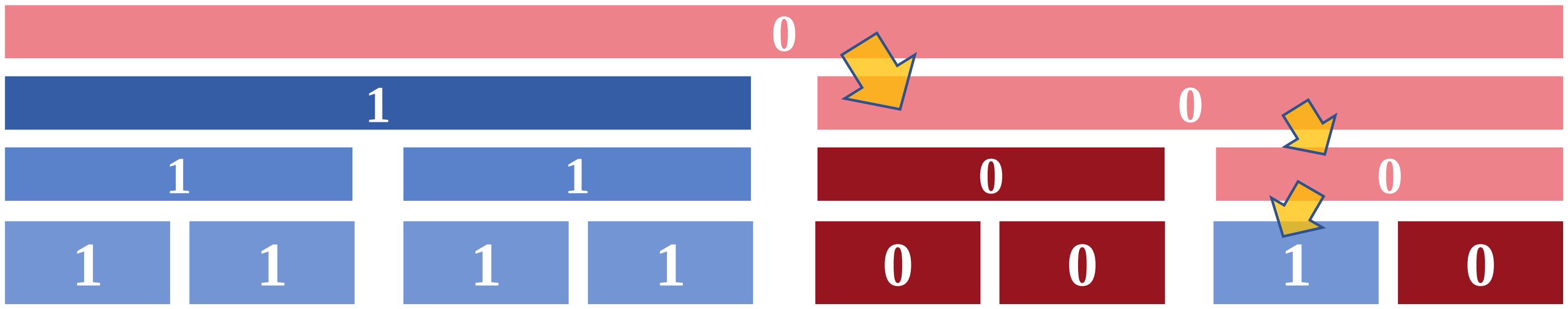
Bit-Vector (BV)

0

1 0

1 1 0 0

1 1 1 1 0 0 1 0



Hardware: Hi-DMM Allocators

- **Fast Buddy Tree Allocator (FBTA):**

  1. Allocation *without searching* layer by layer, based on *bit operation*

BV = 1 1 1 1 0 0 1 0

How to find the lowest set (i.e. 1) bit in BV?

- BV = 0 0 0 0 1 1 1 0 ***(two's complement)***

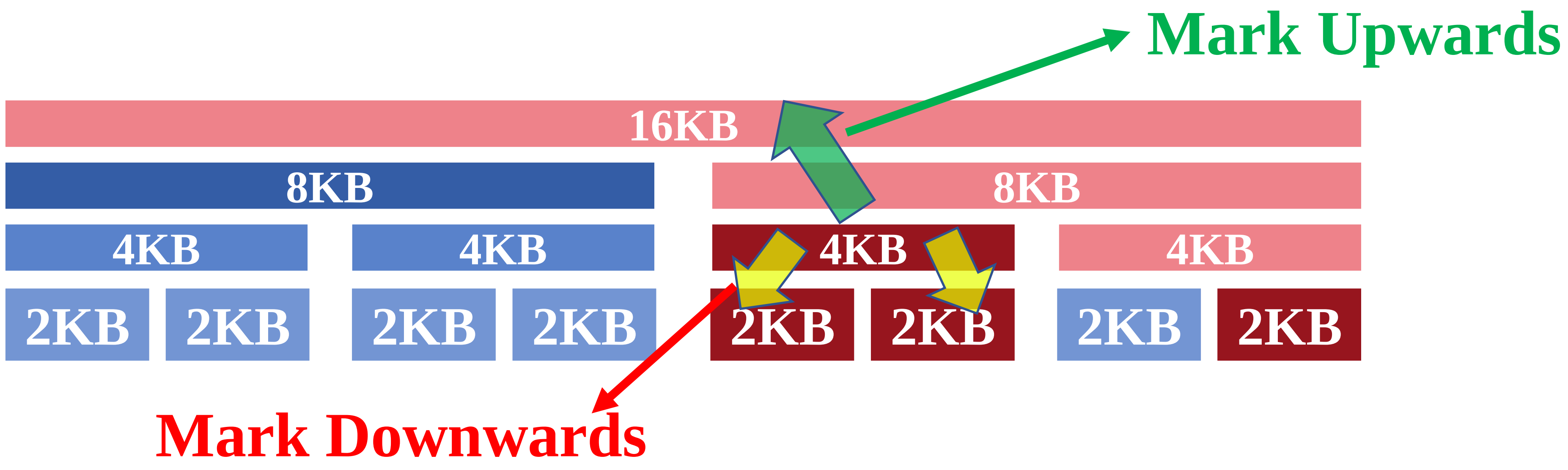
(-BV) &BV = 0 0 0 0 0 0 1 0 → **Index**

Log2 MUX

Hardware: Hi-DMM Allocators

- **Fast Buddy Tree Allocator (FBTA):**

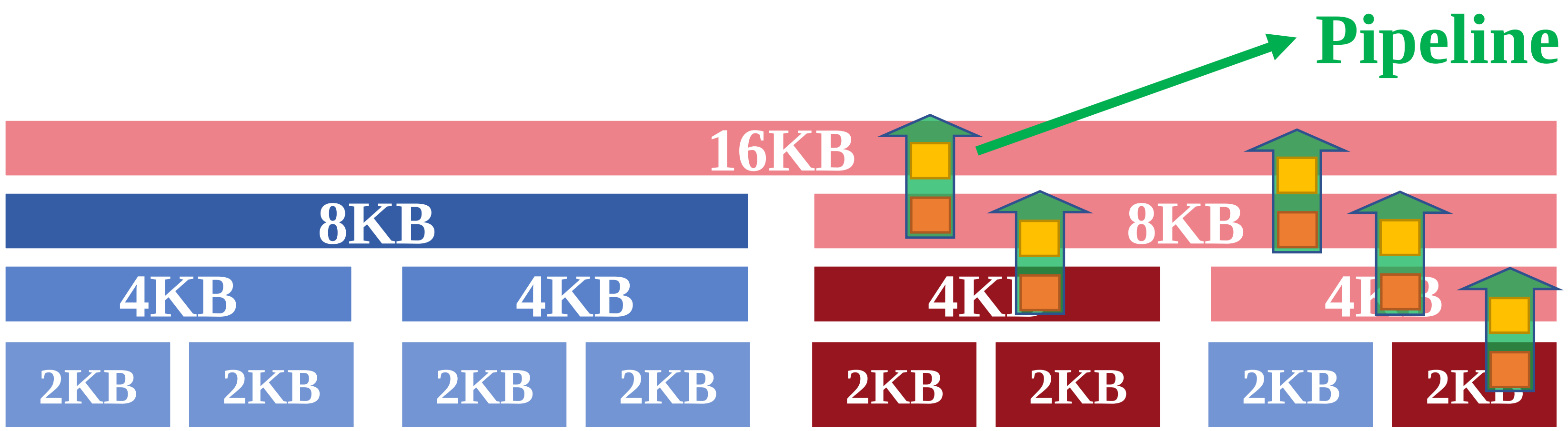
  2. Maintenance *parallelized*



Hardware: Hi-DMM Allocators

- **Fast Buddy Tree Allocator (FBTA):**

    2. Maintenance ***parallelized***



Hardware: Hi-DMM Allocators

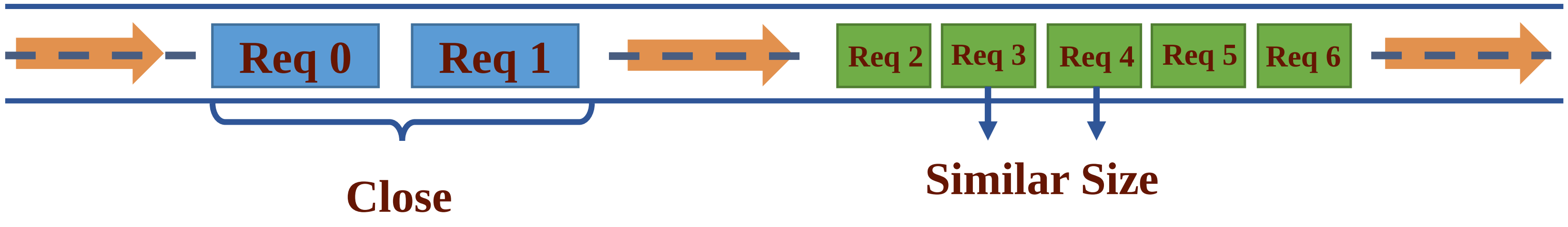
- **Pre-Allocation Tree Allocator (PATA):**

Based on FBTA but can *pre-allocate before the request*

Locality of Allocation

Temporal: **An allocation request is usually followed closely by another one.**

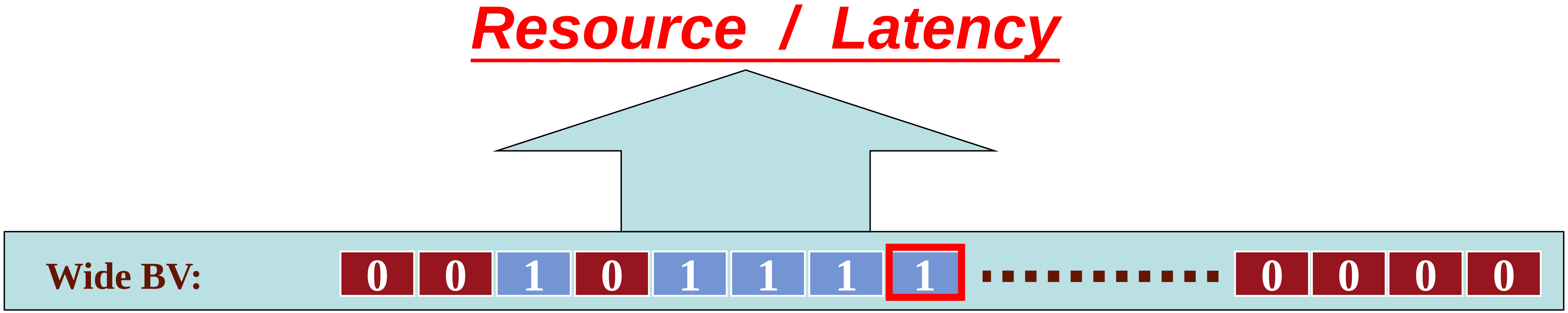
Spatial: **Those allocation requests close to each other usually ask for similar size.**



Hardware: Hi-DMM Allocators

- **Hybrid Tree Allocator (HTA):**

Based on FBTA but can manage those *wide bit-vectors with thousands of bits.*

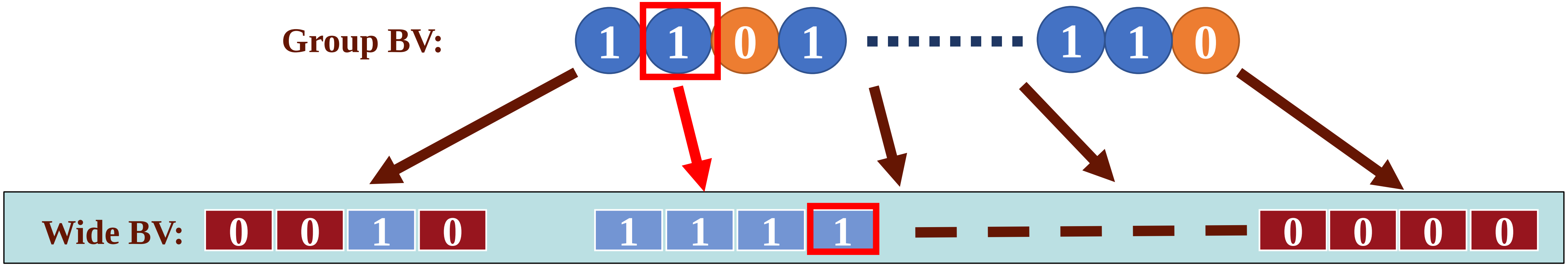


Hardware: Hi-DMM Allocators

- **Hybrid Tree Allocator (HTA):**

Based on FBTA but can manage those *wide bit-vectors with thousands of bits.*

Solution: *use BV to mange a wide BV*



Hardware: Hi-DMM Allocators

- **K-Way Tree Allocator (KWTA):**

Manage fine-grained fixed-size user-defined struct variables

Scenario: Dynamic Data Structure

Characteristics:

1. Fixed-size:

tree_node

queue_element

2. Frequent (De-)Allocation Operations

insert_tree_node()

queue_***pop***()

Waste of Resource

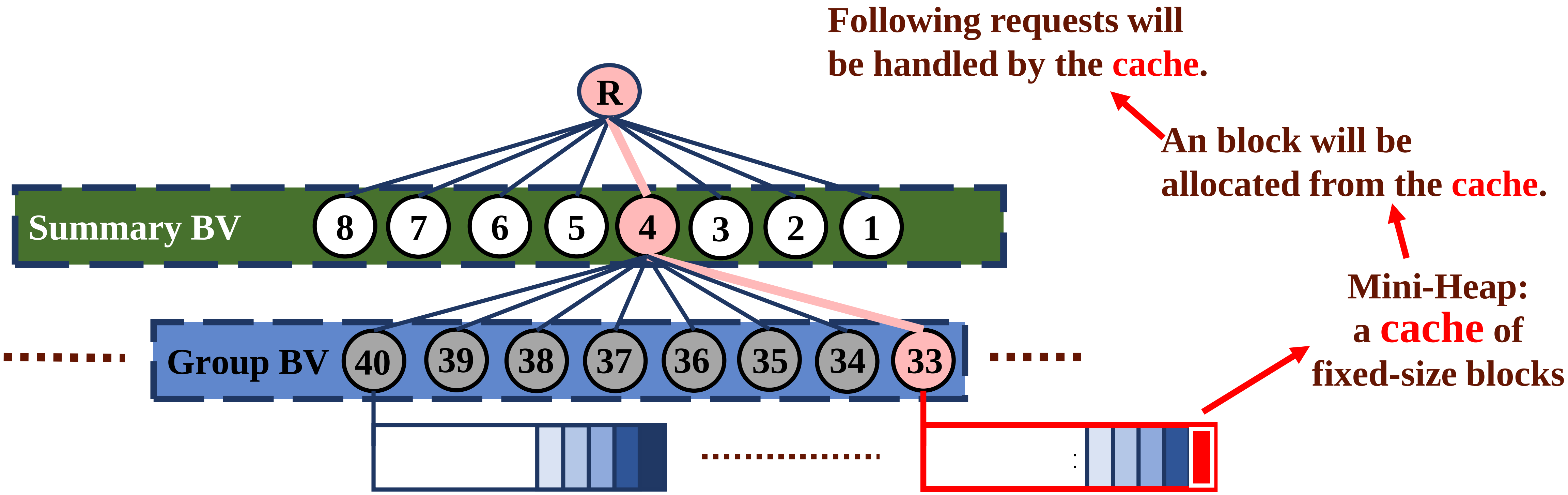
Buddy Tree?

Further Lower Latency Required

Hardware: Hi-DMM Allocators

- **K-Way Tree Allocator (KWTA):**

Manage user-defined struct variables with extremely low latency



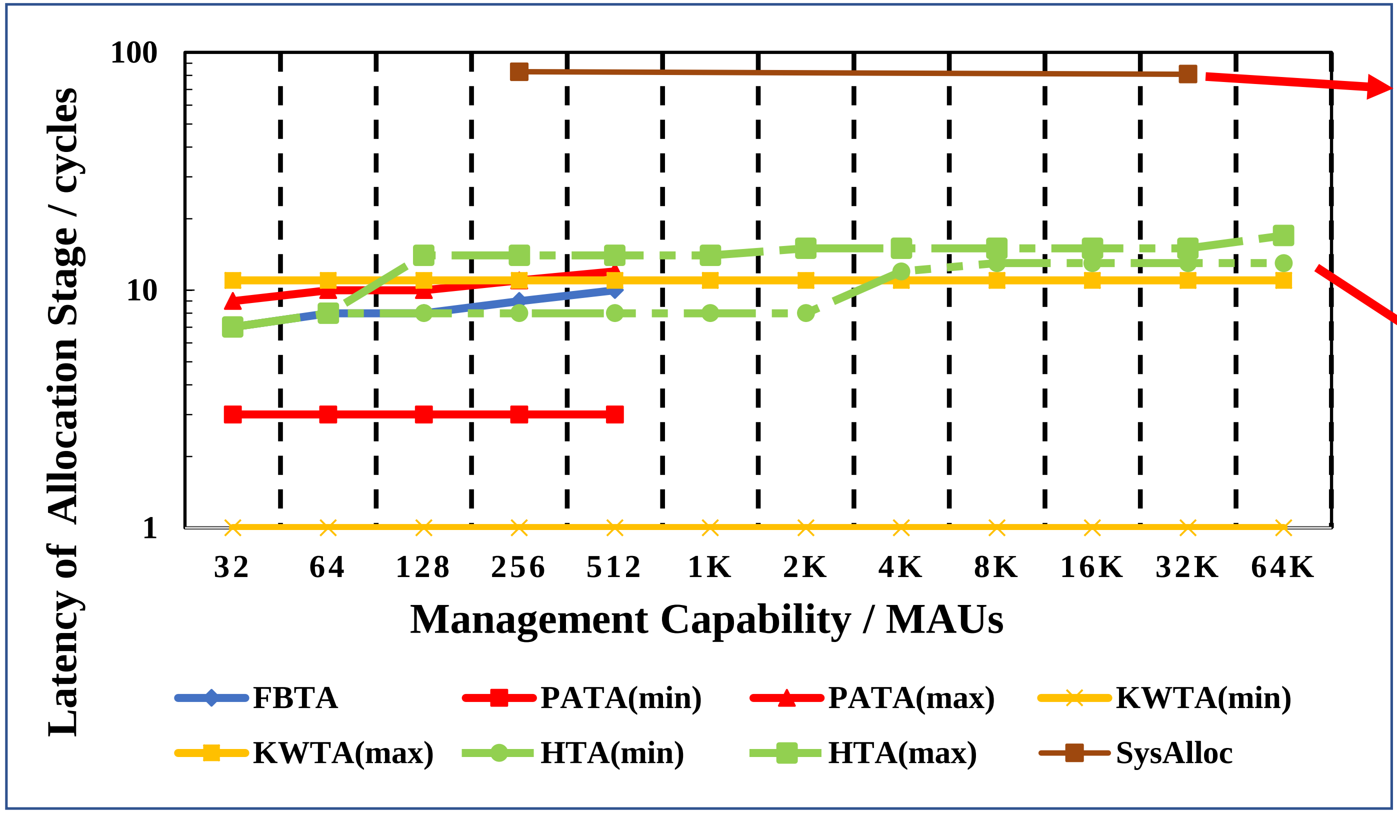
Outline

- Motivation
- Overview of Hi-DMM
- Implementation of Software
- Implementation of Hardware
- Evaluation of Hi-DMM
- Open-Source Hi-DMM Platform
- Conclusion

Evaluation: Allocator Performance

@ 100MHz with Zynq-7020

Allocation Latency



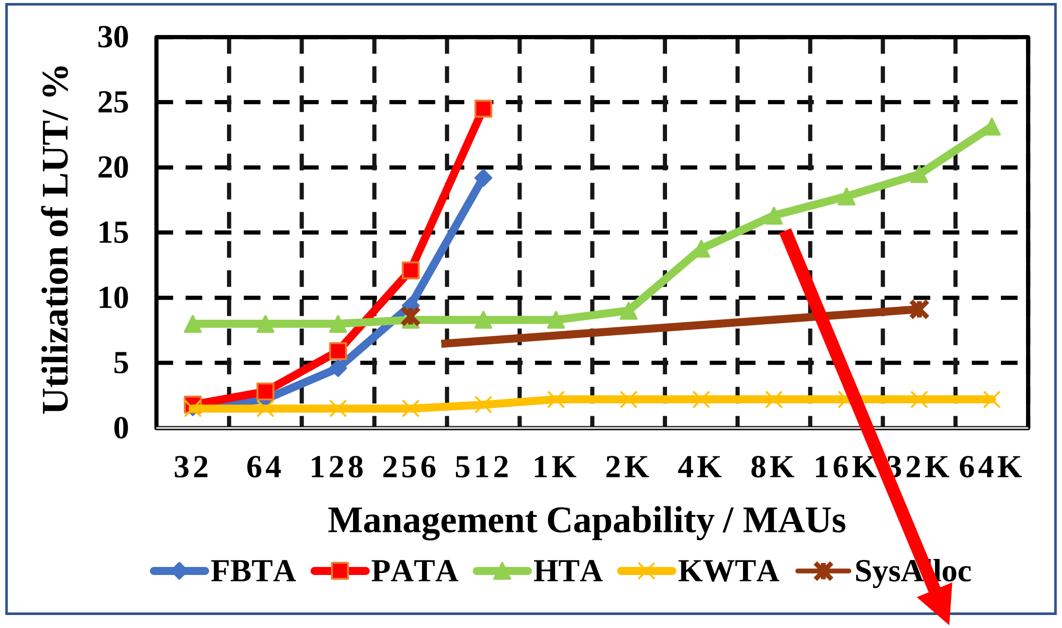
Request handled by SysAlloc (Hundreds of cycles)

Request handled by Hi-DMM (Tens of cycles)

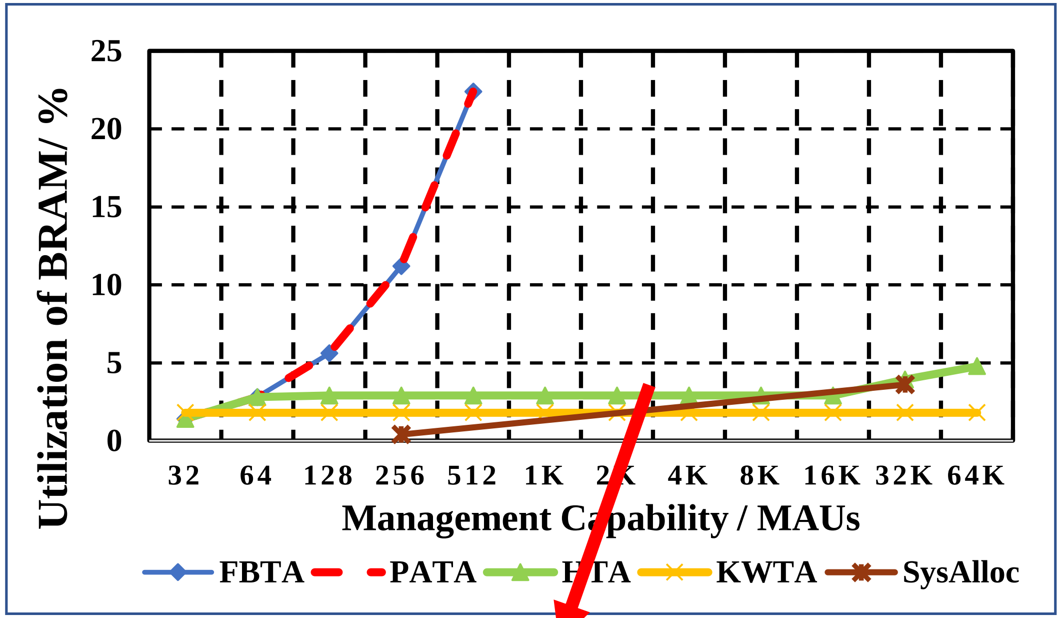
Evaluation: Allocator Resource

@ 100MHz with Zynq-7020

LUT Usage



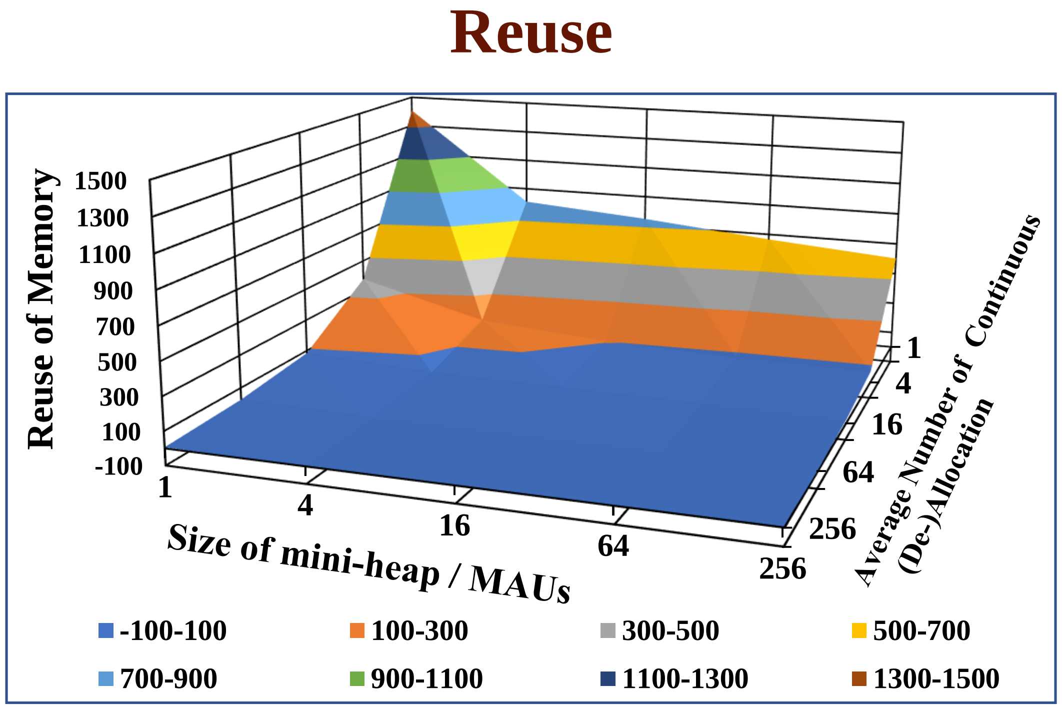
BRAM Usage

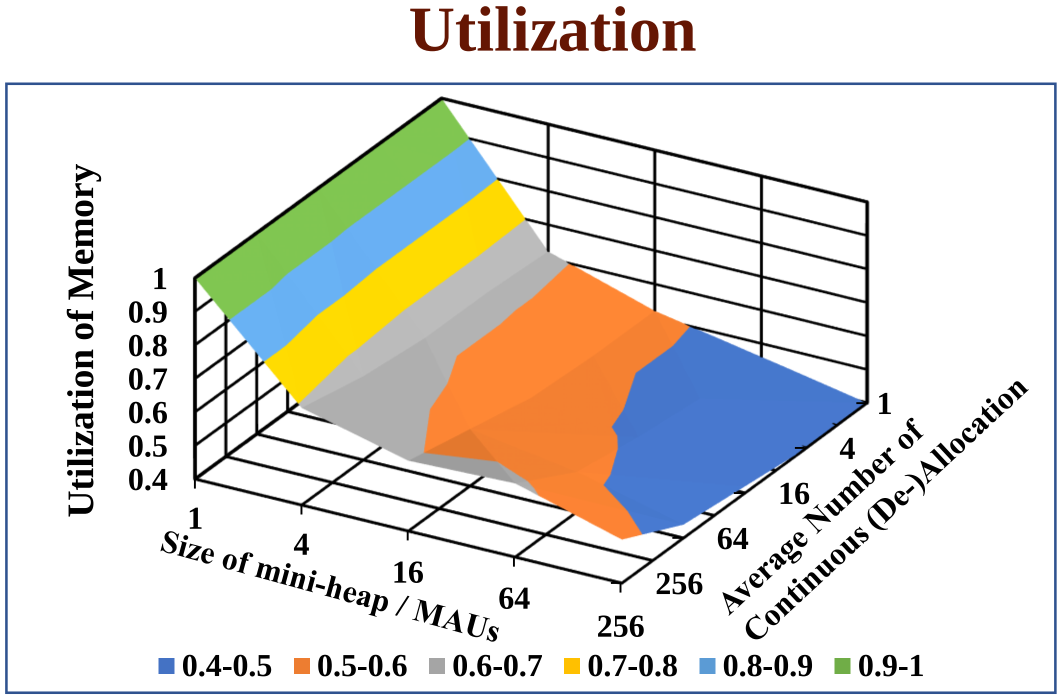


Resource cost by HTA is much lower than FBTA.

Evaluation: Memory Efficiency of KWTA

@ 100MHz with Zynq-7020

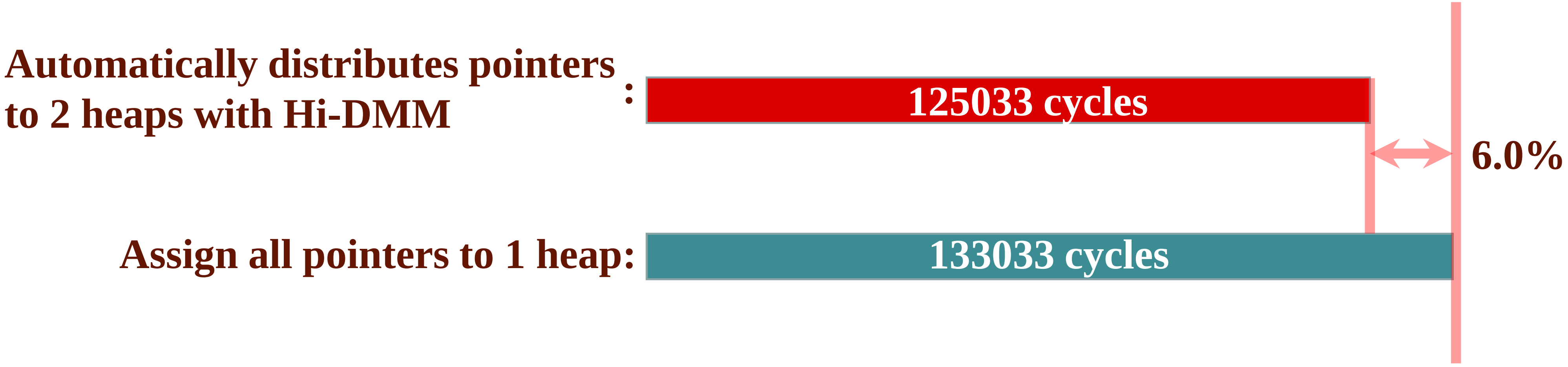




Evaluation: Source Code Optimization

Pointer Mapping

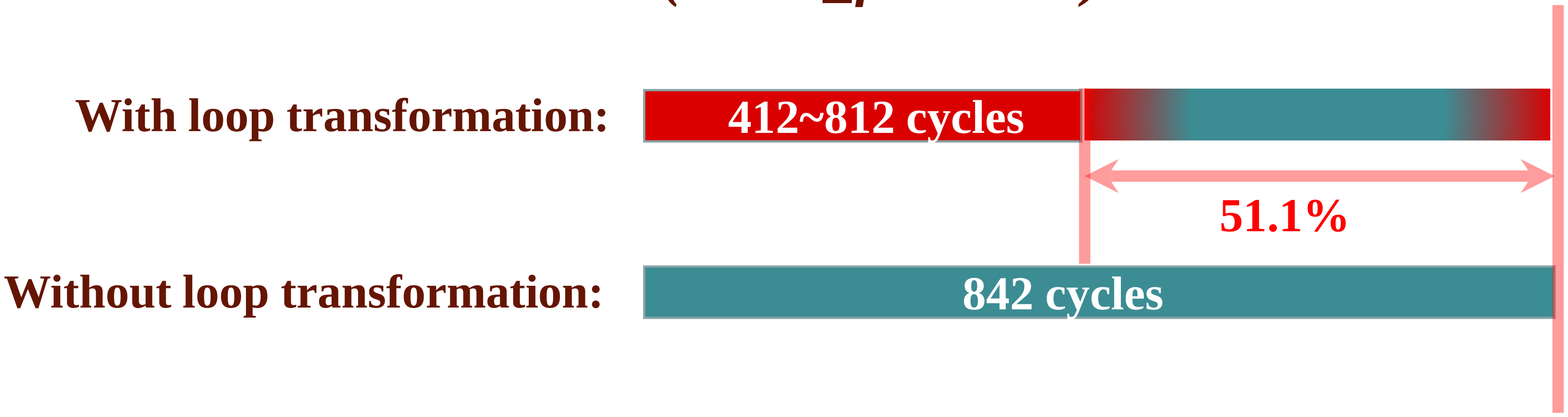
Calculation with Multiple Matrices: ABCD + EF



Evaluation: Source Code Optimization

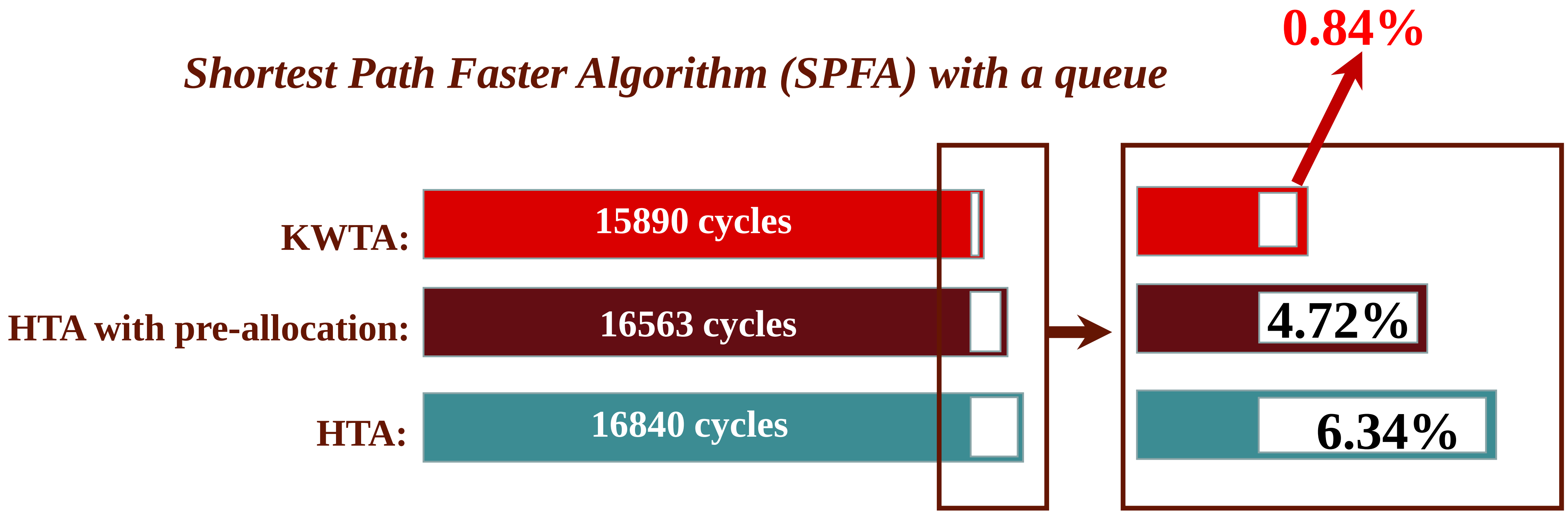
Loop Transformation

Reduction Operation based on Dynamic Arrays
(unroll_factor = 4)



Evaluation: Source Code Optimization

Allocation Selection

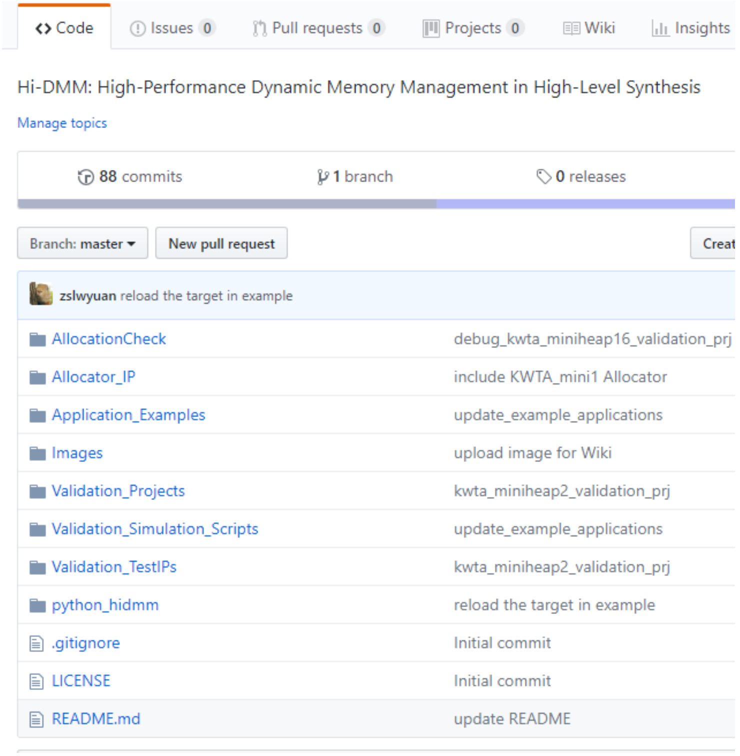


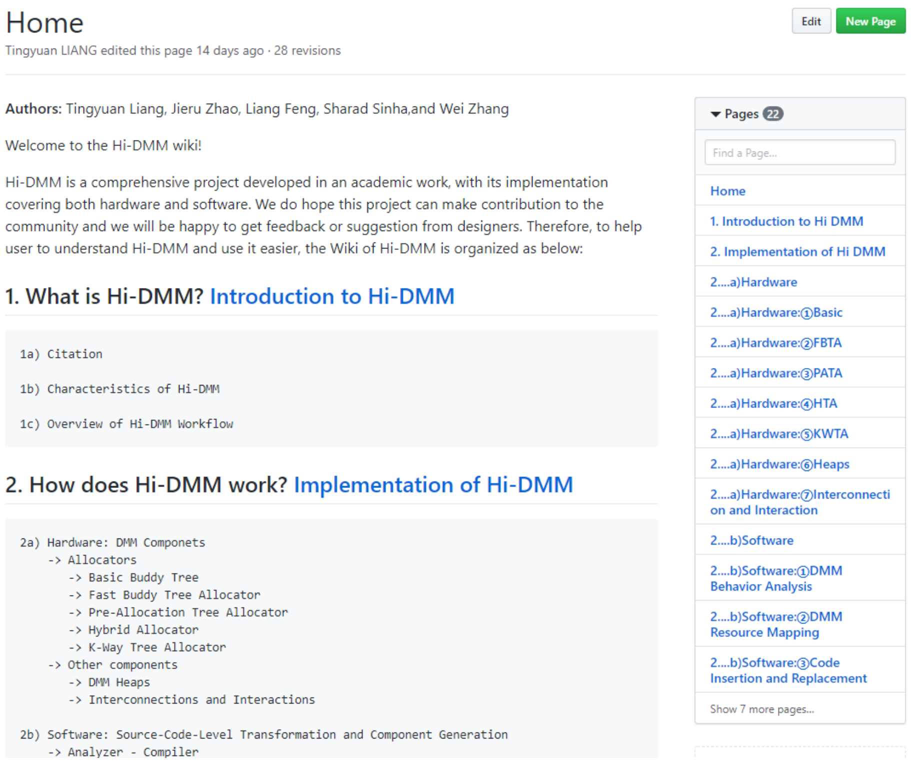
Outline

- Motivation
- Overview of Hi-DMM
- Implementation of Software
- Implementation of Hardware
- Evaluation of Hi-DMM
- Open-Source Hi-DMM Platform
- Conclusion

Hi-DMM is open to the community

https://github.com/zslwyuan/Hi-DMM





Outline

- Motivation
- Overview of Hi-DMM
- Implementation of Software
- Implementation of Hardware
- Evaluation of Hi-DMM
- Open-Source Hi-DMM Platform
- Conclusion

Conclusion

– Software: Hi-DMM Compiler
  - Automatic Transformation
  - HLS-Friendly
  - Couple with commercial tools (Vivado / Vivado HLS)

– Hardware: Hi-DMM Allocator
  - High Performance
  - HLS-Friendly
  - Adaptive to various applications

– Future Works
  - Consider more DMM characteristics
  - Further improve the performance of allocators

Thanks!