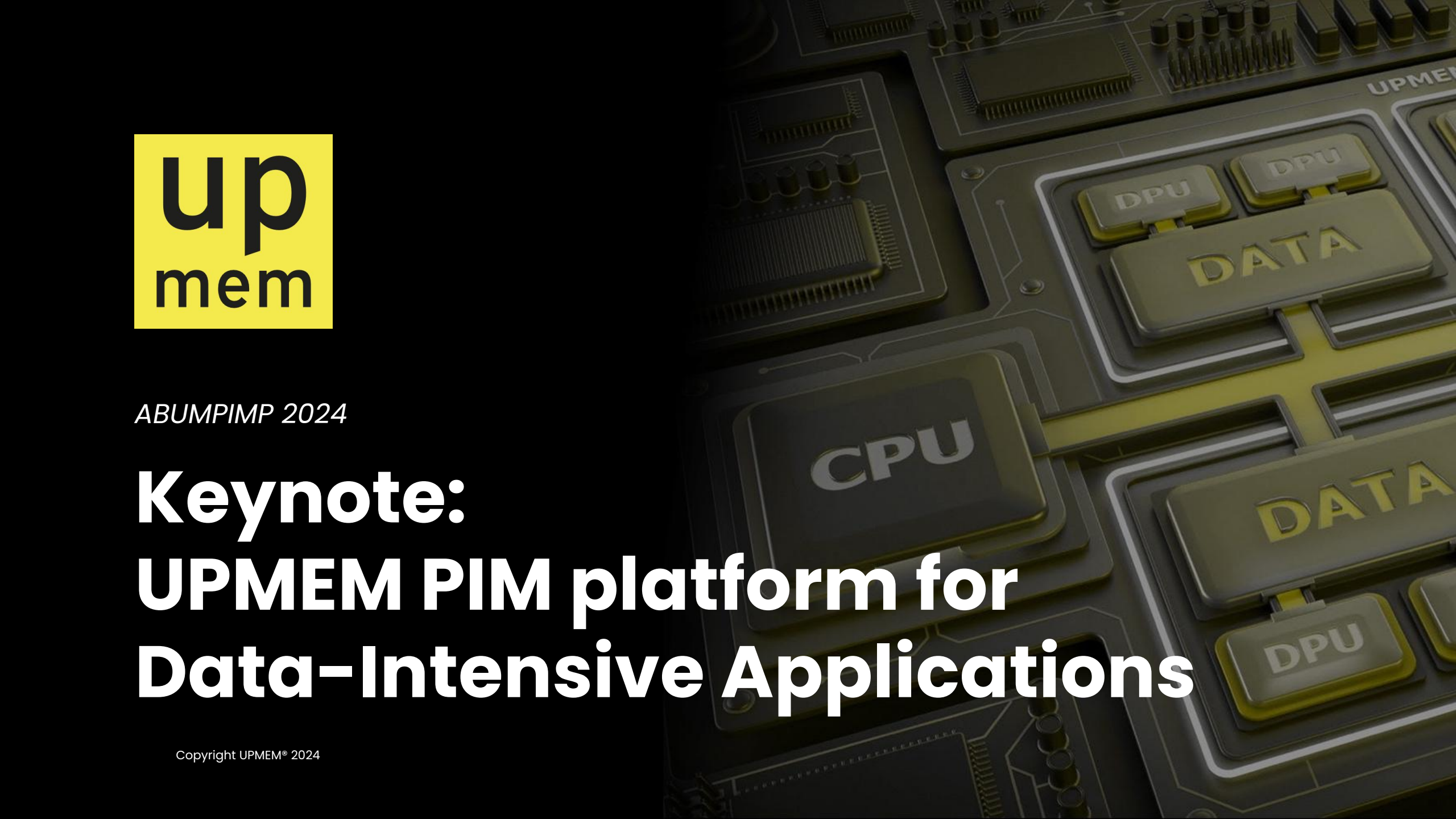
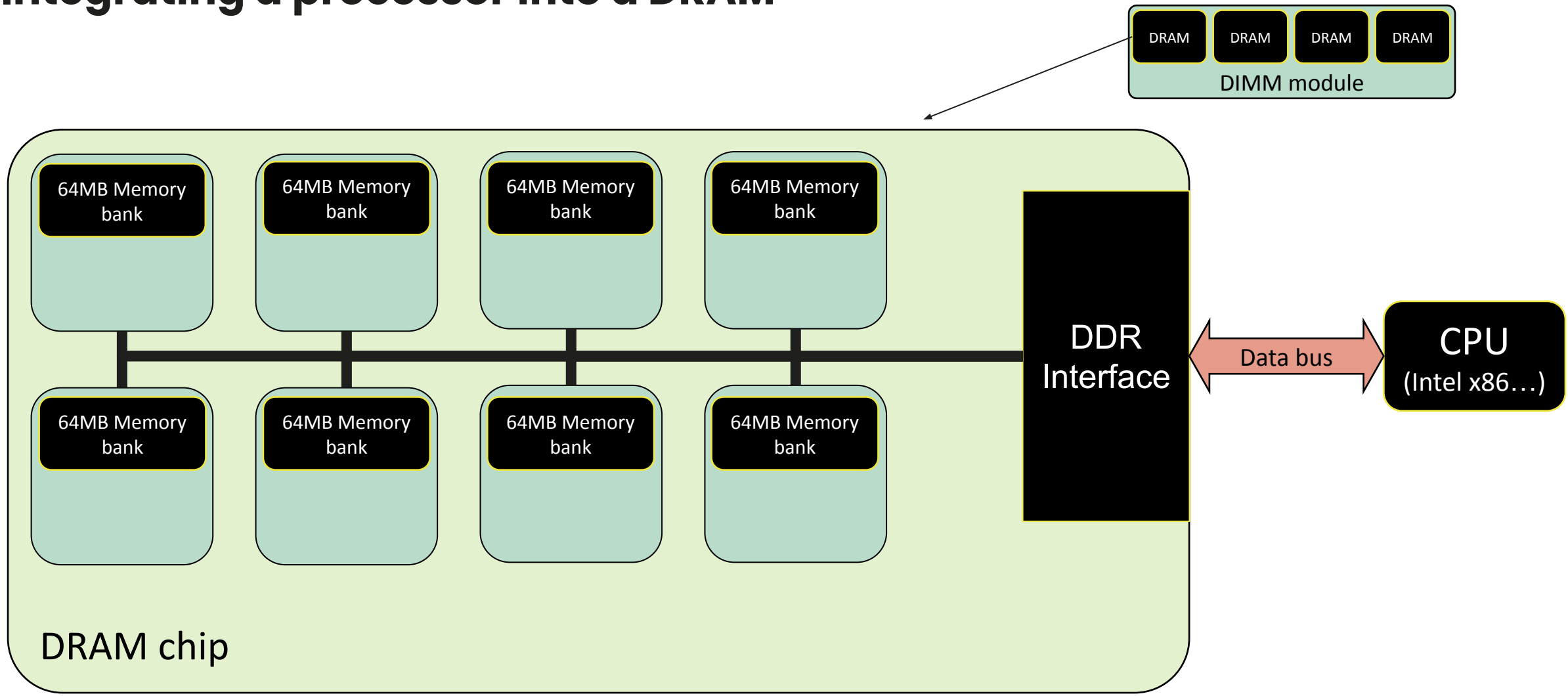


ABUMPIMP 2024

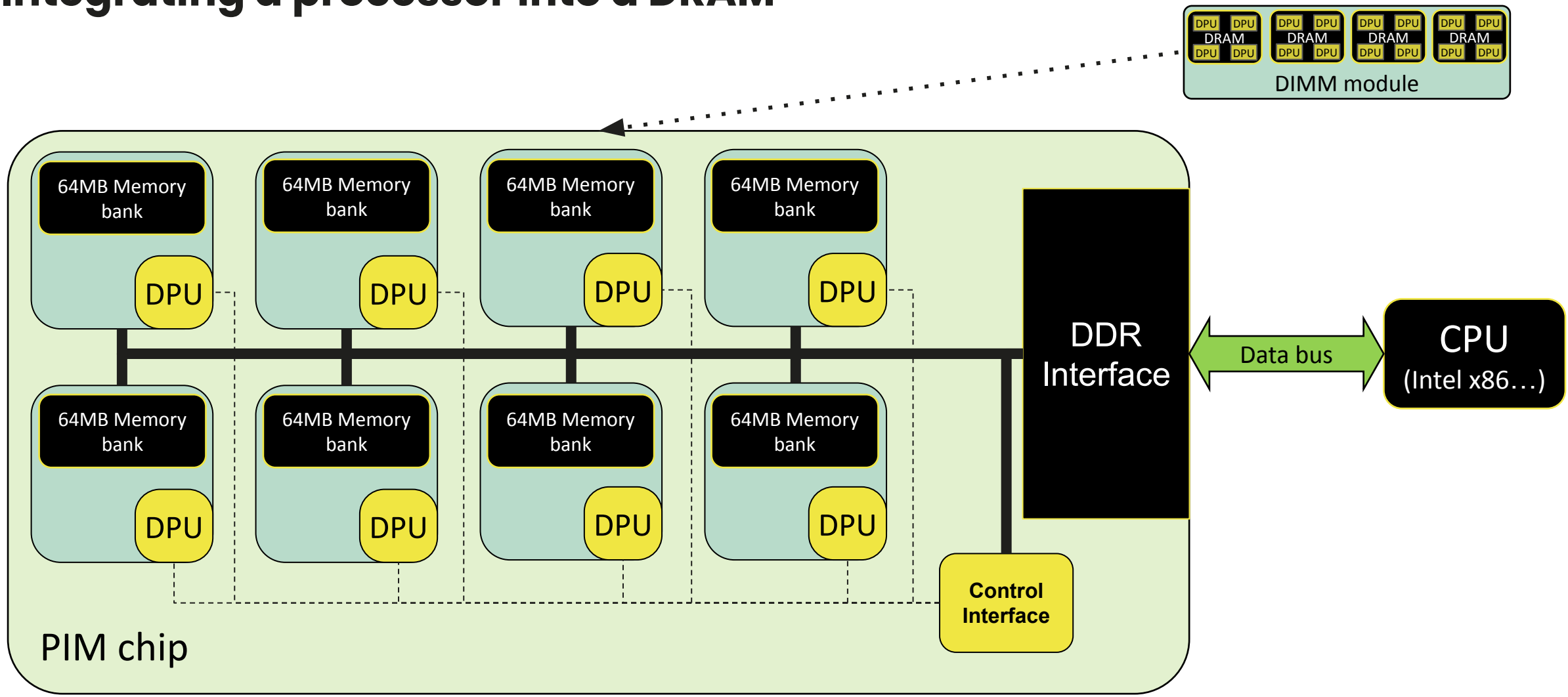
Keynote: UPMEM PIM platform for Data-Intensive Applications

High Level Hardware Architecture

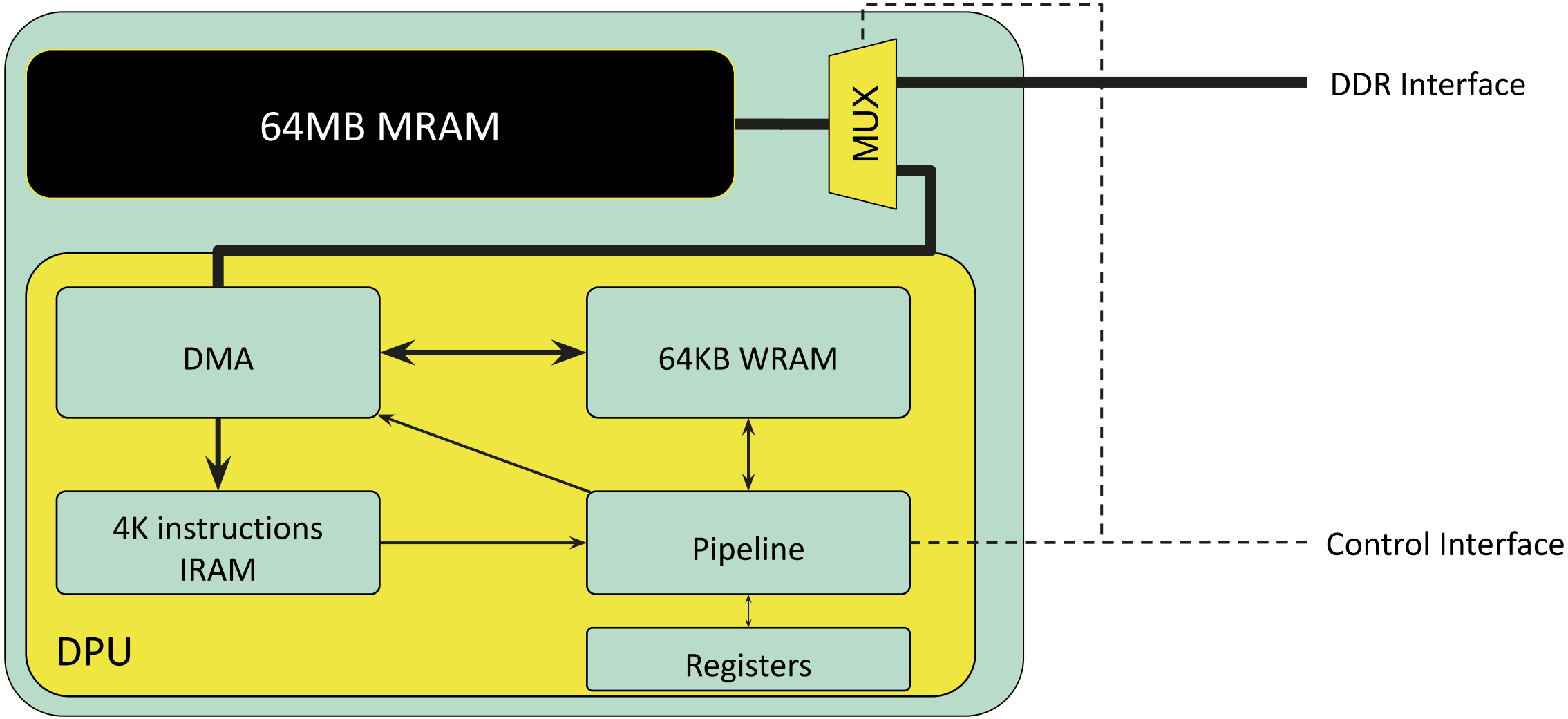
Integrating a processor into a DRAM



Integrating a processor into a DRAM

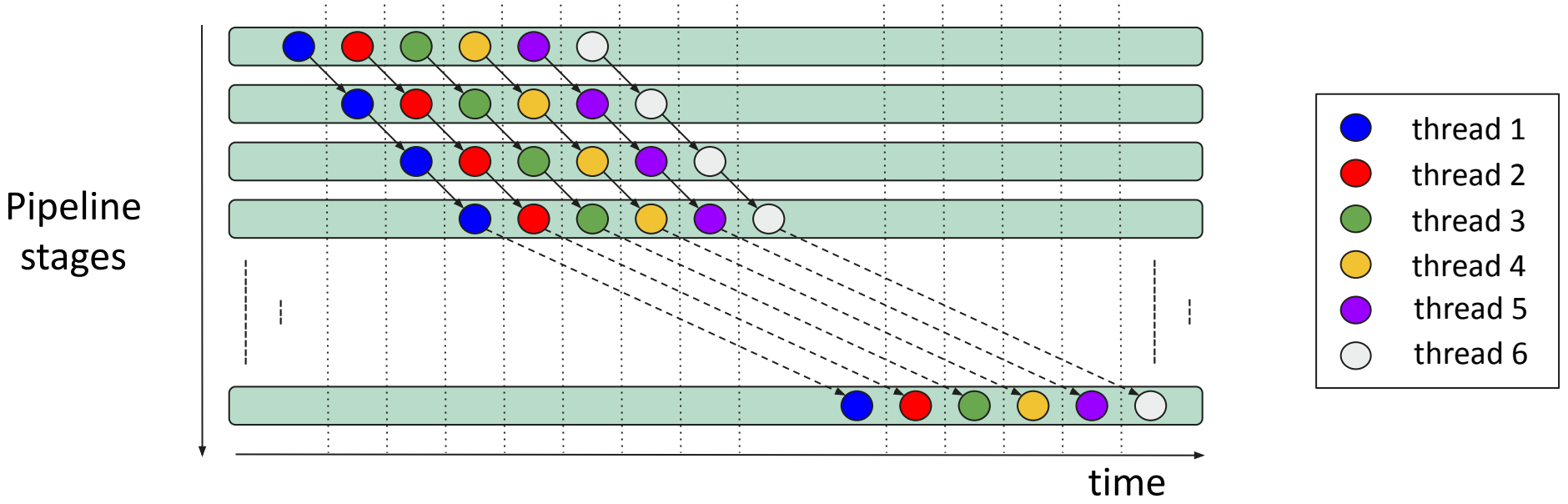


The DPU architecture

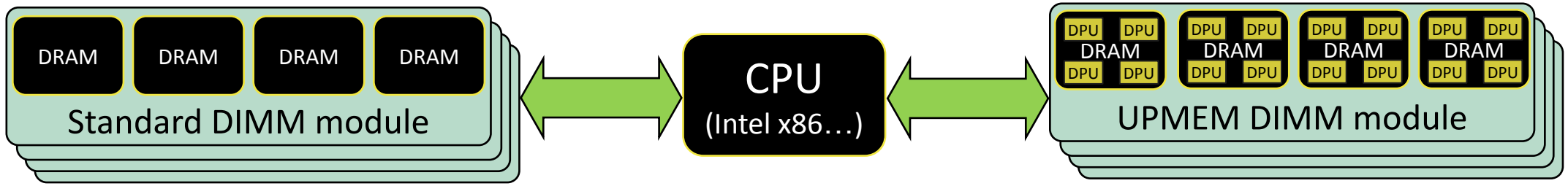


Pipeline properties

- 11-stage pipeline
- 24 hardware threads interleaved
- 32 registers per thread
  - 24 multi-purpose registers
  - 8 constant registers (zero, one, identifier of thread, etc)



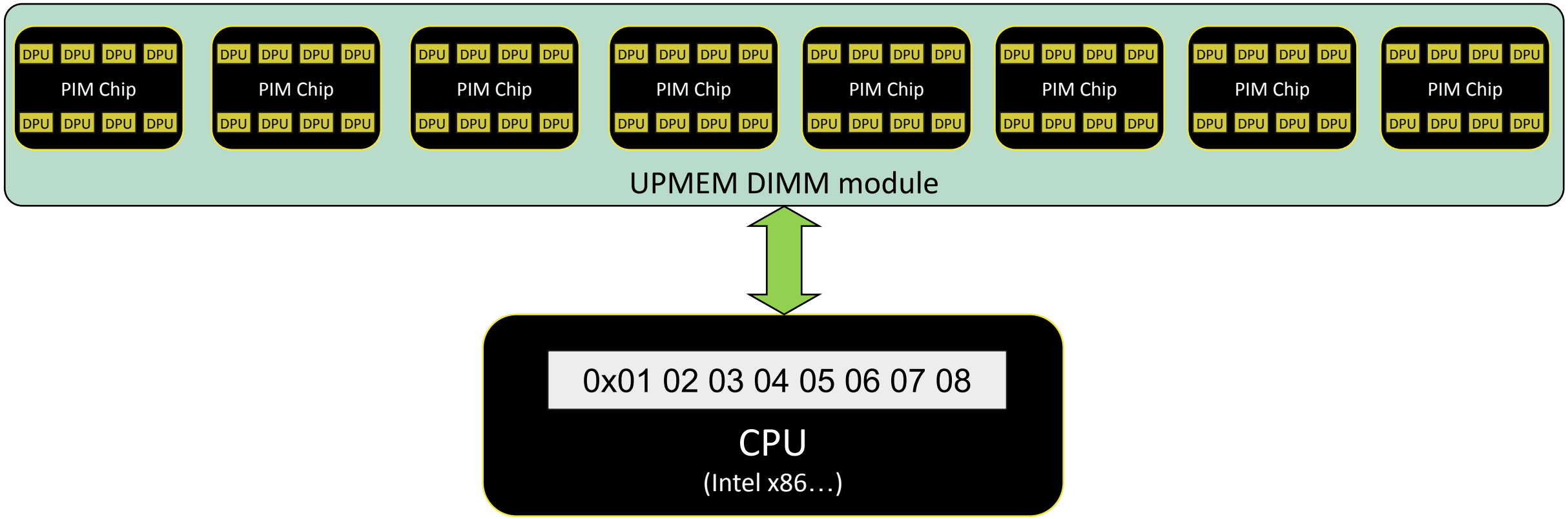
System overview



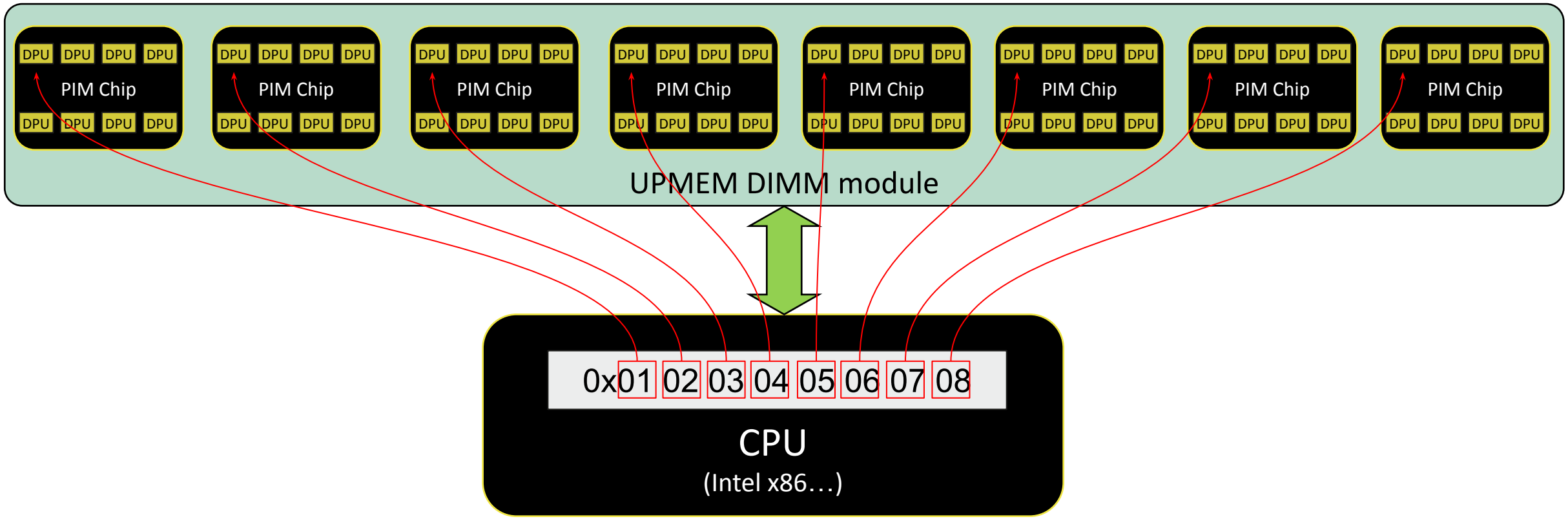
Typical workflow:

1. Init DPUs
2. Loop
   a. Copy data to DPU(s)
   b. Execute program on DPU(s)
   c. Copy data from DPU(s)

CPU/DPUs communication



CPU/DPUs communication



Programming the UPMEM PIM

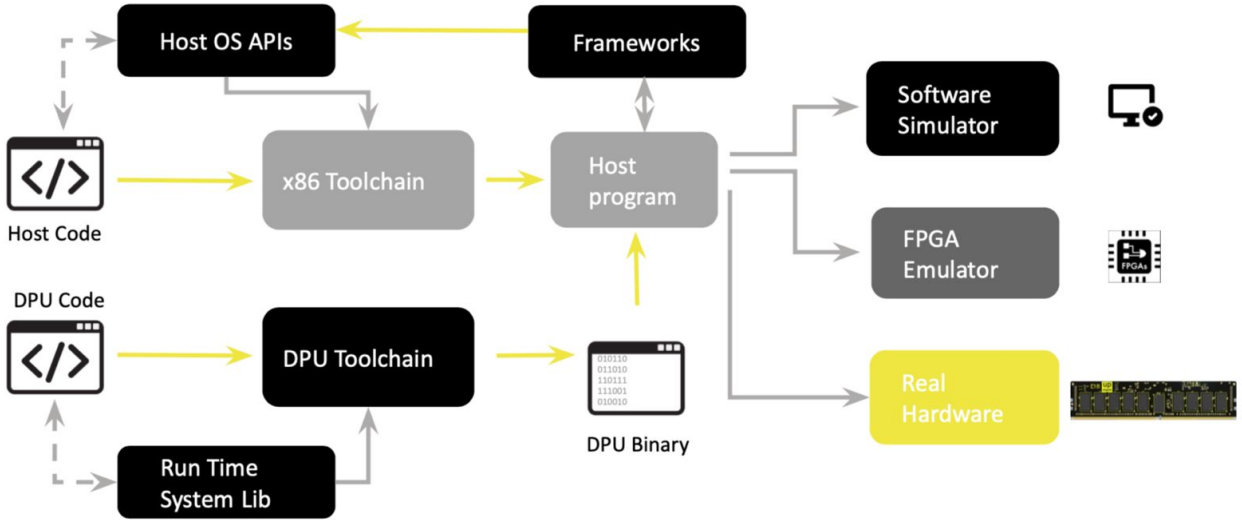
UPMEM SDK & Application Design Flow

- **CPU program**
  - compiled using the x86 toolchain
  - uses UPMEM's SDK host library
  - Allocate DPUs, load DPU program, boot DPUs, copy, asynchronous orchestration etc.

- **DPU program**
  - Written in C, subset of C library available
  - usually common to all DPUs (not mandatory)
  - compiled using dpu-clang (based on LLVM 12)
  - uses the DPU libraries for primitives like WRAM/MRAM management, thread synchronization (mutex etc.), perf counters etc.

- **Profiling & debugging tools**
  - DPU: dpu-lldb, dpu-trace, etc.
  - Host: dpu-profiling tool based on Linux perf

- **System**
  - Linux driver for x86 servers
  - Server BIOS binaries



The DPU ISA

- Proprietary RISC triadic instruction set
- **No FPU** : FP using a software library (IEEE-754)
  - ~200 cycles for a 32x32 float multiplication
  - consider quantization for FP applications
- No vectorized instructions (except cmpb4)
- 8x8 multiplication instruction, up to 32 ops for 32x32
- **Rich set of conditions for jump (examples next slide)**
- Assembly code analysis & optimization can be useful

- Operators (non-exhaustive):
  - Arithmetic: ADD, SUB, AND, OR, XOR
  - Loads/Stores: SD, SW, SH, SB, LD, LW, LH, LB
  - Shift/Rotate: LSL, LSR, ROL, ROR
  - Count bits: CLZ, CLO, CLS, CAO
  - Multiplication/Division: MUL_STEP, DIV_STEP, MUL
  - DMA loads/stores: SDMA, LDMA, LDMAI

The DPU ISA

16bit-integer multiplication

```
uint32_t mult(uint16_t a, uint16_t b) {
    return a * b;
}
```

```
mult:                                          // @mult
        move r2, r0
        mul_ul_ul r0, r1, r2, small, .LBB0_2
        mul_uh_ul r3, r1, r2
        lsl_add r0, r0, r3, 8
        mul_uh_ul r3, r2, r1
        lsl_add r0, r0, r3, 8
        mul_uh_uh r1, r1, r2
        lsl_add r0, r0, r1, 16
.LBB0_2:
        jump r23
```

```
int sum(const int* data, int size)
{
    int val = 0;
    for(int i=size; i!= 0; i--)
        val += data[i];

    return val;
}
```

```
sum:
        move r2, r0
        move r0, 0
        jeq r1, 0, .LBB0_2
.LBB0_1:
        lsl_add r3, r2, r1, 2
        lw r3, r3, 0
        add r0, r3, r0
        add r1, r1, -1, nz, .LBB0_1
```

Reverse Loop

The DPU ISA

compiler pessimization

```
int euclidian_distance(
    size_t n,
    int8_t a[static n],
    int8_t b[static n])
{
    int dist = 0;
    for (int i = 0; i < n; ++i) {
        int16_t diff = a[i] - b[i];
        dist += diff * diff;
    }
    return dist;
}
```



```
euclidian_distance:
        sd r22, 16, d22
        add r22, r22, 24
        sd r22, -16, d14
        sd r22, -24, d16
        move r14, r2
        move r16, r1
        move r15, 0
        move r17, r0, z, .LBB1_2
.LBB1_1:
        lbs r0, r16, 0
        lbs r1, r14, 0
        sub r0, r0, r1
        move r1, r0
        call r23, __mulsi3
        add r15, r0, r15
        add r14, r14, 1
        add r16, r16, 1
        add r17, r17, -1, nz, .LBB1_1
.LBB1_2:
        move r0, r15
        ld d16, r22, -24
        ld d14, r22, -16
        ld d22, r22, -8
        jump r23
```

The DPU ISA

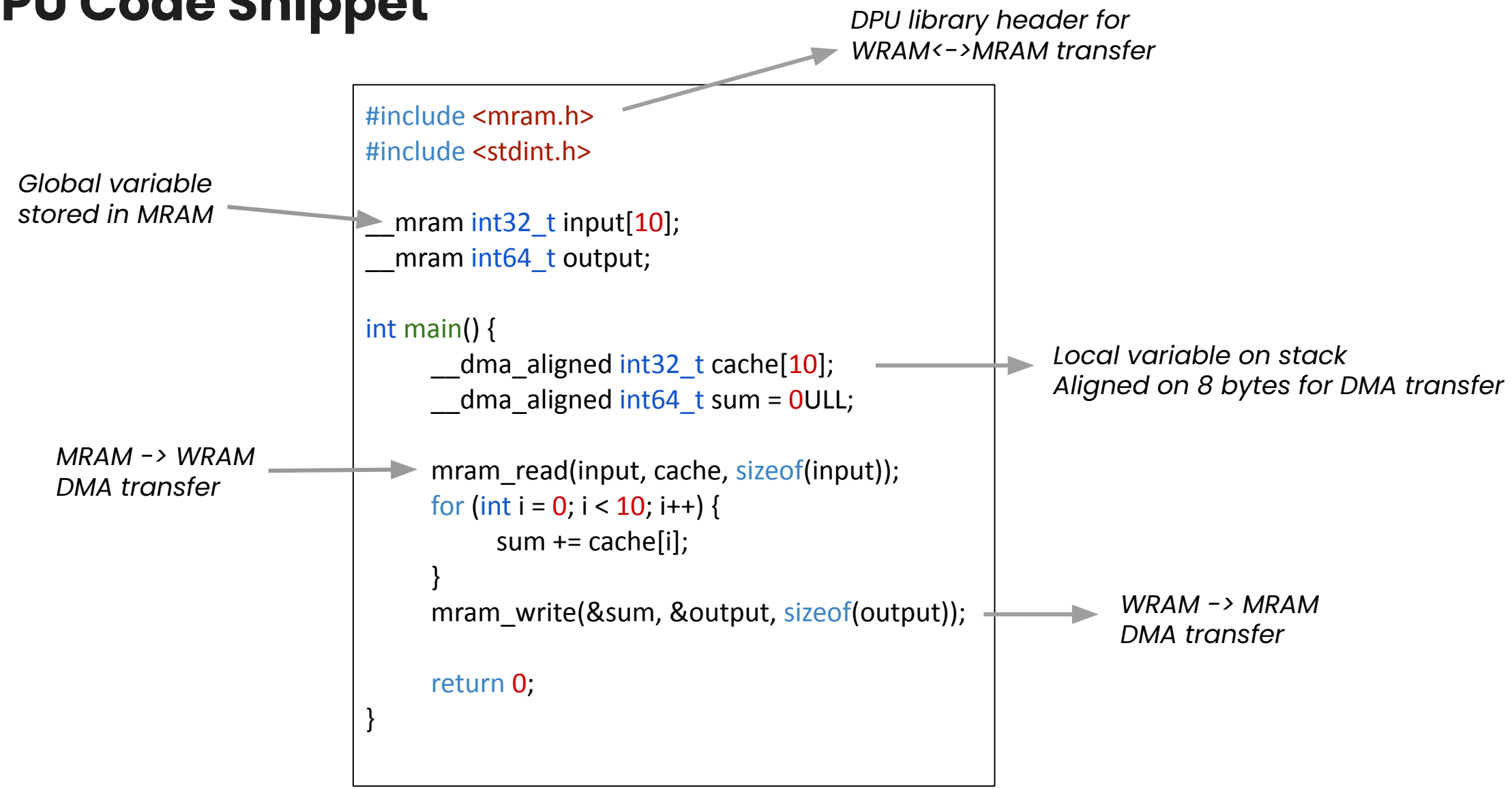
workaround

```
int euclidian_distance(
    size_t n,
    int8_t a[static n],
    int8_t b[static n])
{
    int dist = 0;
    for (int i = 0; i < n; ++i) {
        volatile int16_t diff = a[i] - b[i];
        dist += diff * diff;
    }
    return dist;
}
```



```
euclidian_distance:                    // @eu
        move r3, r0, z, .LBB0_1
        move r0, 0, true, .LBB0_4
.LBB0_6:                               //   i
        add r0, r4, r0
        add r2, r2, 1
        add r1, r1, 1
        add r3, r3, -1, z, .LBB0_2
.LBB0_4:                               // =>T
        lbs r4, r1, 0
        lbs r5, r2, 0
        sub r4, r4, r5
        sh r22, 0, r4
        lhs r5, r22, 0
        lhs r6, r22, 0
        mul_ul_ul r4, r6, r5, small, .LBB0_6
        mul_sh_ul r7, r6, r5
        lsl_add r4, r4, r7, 8
        mul_sh_ul r7, r5, r6
        lsl_add r4, r4, r7, 8
        mul_sh_sh r5, r6, r5
        lsl_add r4, r4, r5, 16, true, .LBB0_6
.LBB0_1:
        move r0, 0
.LBB0_2:
        jump r23
```

DPU Code Snippet




```
#include <mram.h>
#include <stdint.h>

__mram int32_t input[10];
__mram int64_t output;

int main() {
        __dma_aligned int32_t cache[10];
        __dma_aligned int64_t sum = 0ULL;

        mram_read(input, cache, sizeof(input));
        for (int i = 0; i < 10; i++) {
                sum += cache[i];
        }
        mram_write(&sum, &output, sizeof(output));

        return 0;
}
```

Host Code Snippet

Include C Host library header

```
#include <dpu.h>

int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

int main() {
        struct dpu_set_t dpu_set, dpu;
        DPU_ASSERT(dpu_alloc(1, "", dpu_set));
        DPU_ASSERT(dpu_load(dpu_set, "dpu_binary", NULL));

        DPU_FOREACH(dpu_set, dpu) {
                DPU_ASSERT(dpu_copy_to(dpu, "input", 0, array, sizeof(array)));
        }

        DPU_ASSERT(dpu_launch(set, DPU_SYNCHRONOUS));

        int64_t sum;
        DPU_FOREACH(dpu_set, dpu) {
                DPU_ASSERT(dpu_copy_from(dpu, "output", 0, &sum, sizeof(sum)));
        }
        return 0;
}
```

Allocate 1 DPU

Load DPU program

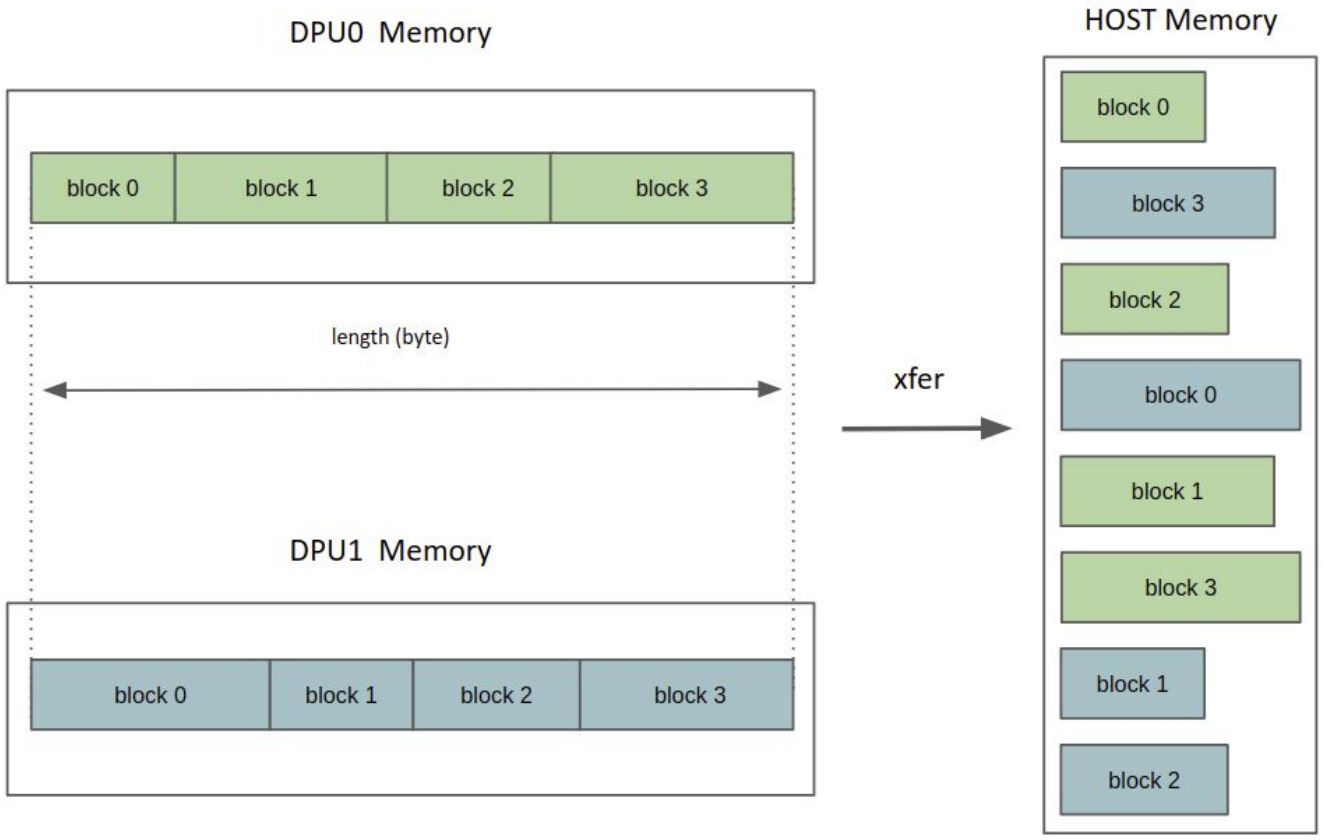
Copy the array content to the DPU variable named "input"

Boot the DPU program and wait for it to finish

Copy the DPU variable named "output" to the host variable sum

New features of the SDK

Scatter/Gather Transfers



(and vice versa)

functional API

```
struct sg_block_info {
    uint8_t *addr;
    uint32_t length;
};
```

```
bool get_block(
  struct sg_block_info *out,
  uint32_t dpu_index,
  uint32_t block_index,
  void *args
);
```


up mem

Transfers Optimization

- Gather Data Sampling side-channel vulnerability discovered last summer.
- Microcode mitigation caused AVX512 Gather instructions to get much slower.

Latency and Throughput

Architecture	Latency	Throughput (CPI)
Skylake	30	9.75

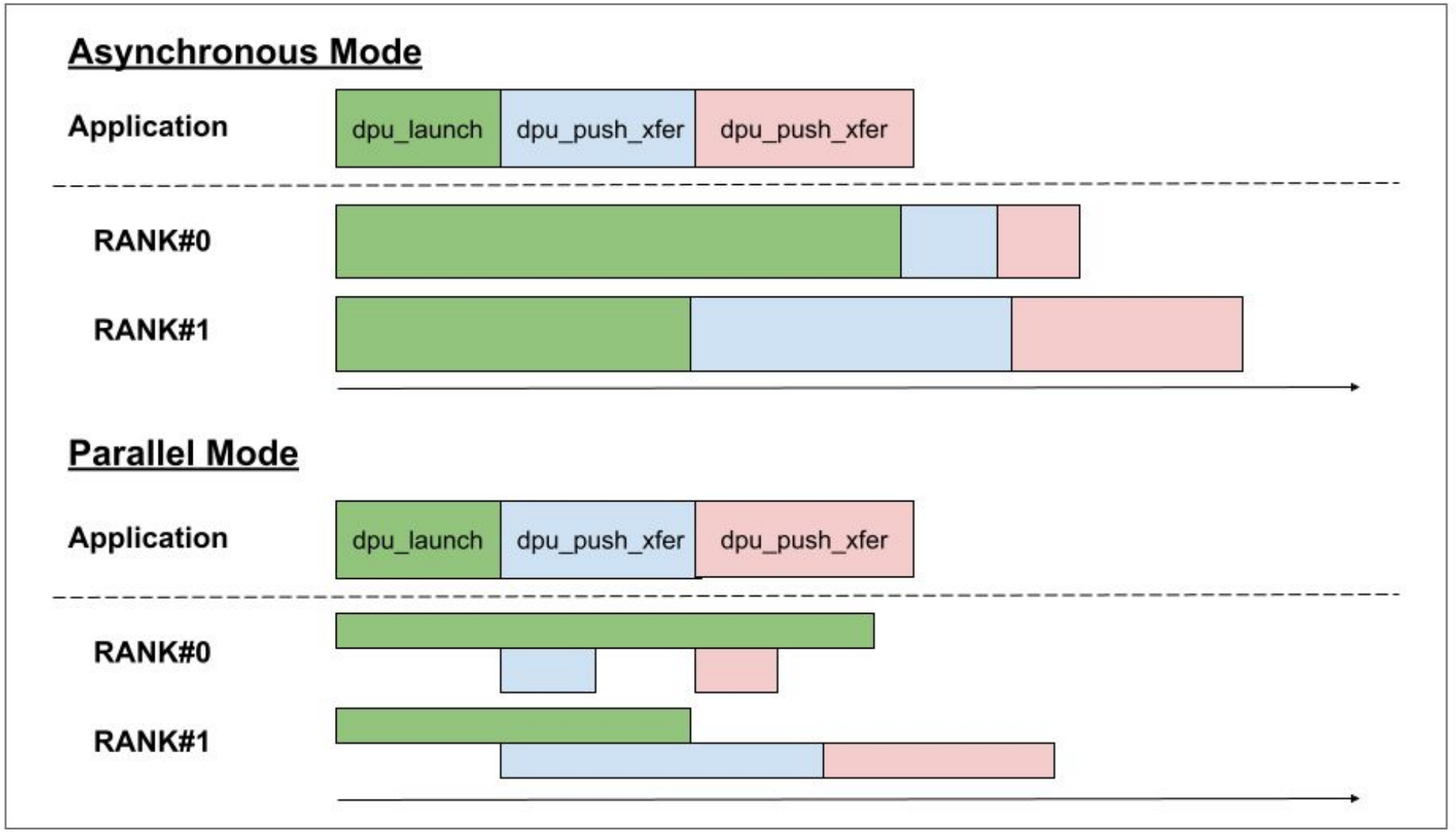
- Gather instruction used in the critical loop of host <-> DPU MRAM transfers.
- Customer benchmarks registered a x2~3 performance degradation.
- Debian 10 servers not affected. Ubuntu 20/22 affected.
- Solution offered by Dr. Manuel Penschuck from Frankfurt University.

- modified SDK (with further improvements): github.com/manpen/upmem-libdpu
- benchmarks: github.com/manpen/dpu-communication-bench

substitute gather with load-permute

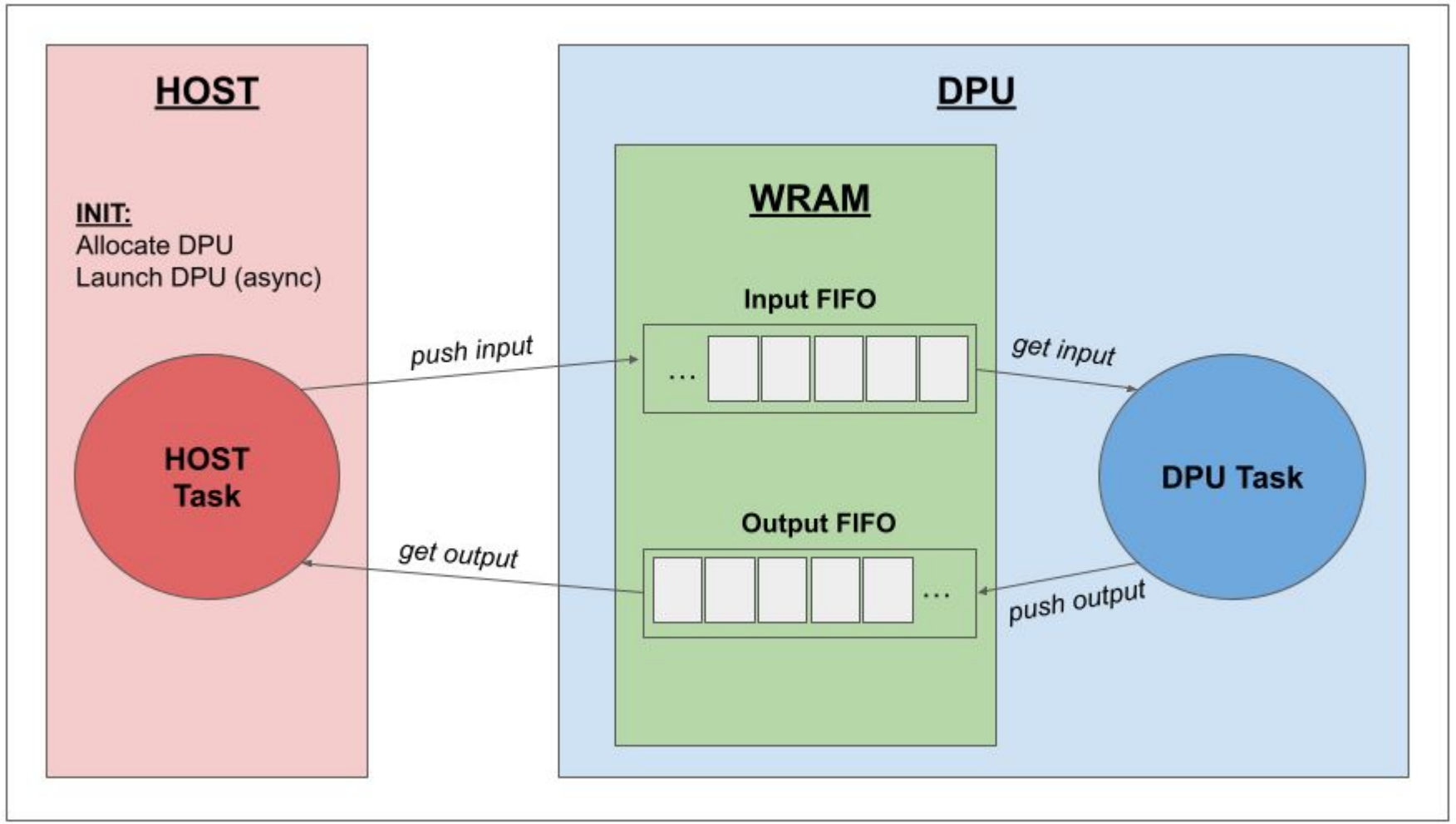
```
__m512i gathered = _mm512_i32gather_epi32(vindex, input, 1);
__m512i load = _mm512_loadu_si512(input);
__m512i gathered = _mm512_permutexvar_epi32(vindex, load);
```

WRAM Parallel Access



upmem

WRAM Parallel Access



WRAM Parallel Access

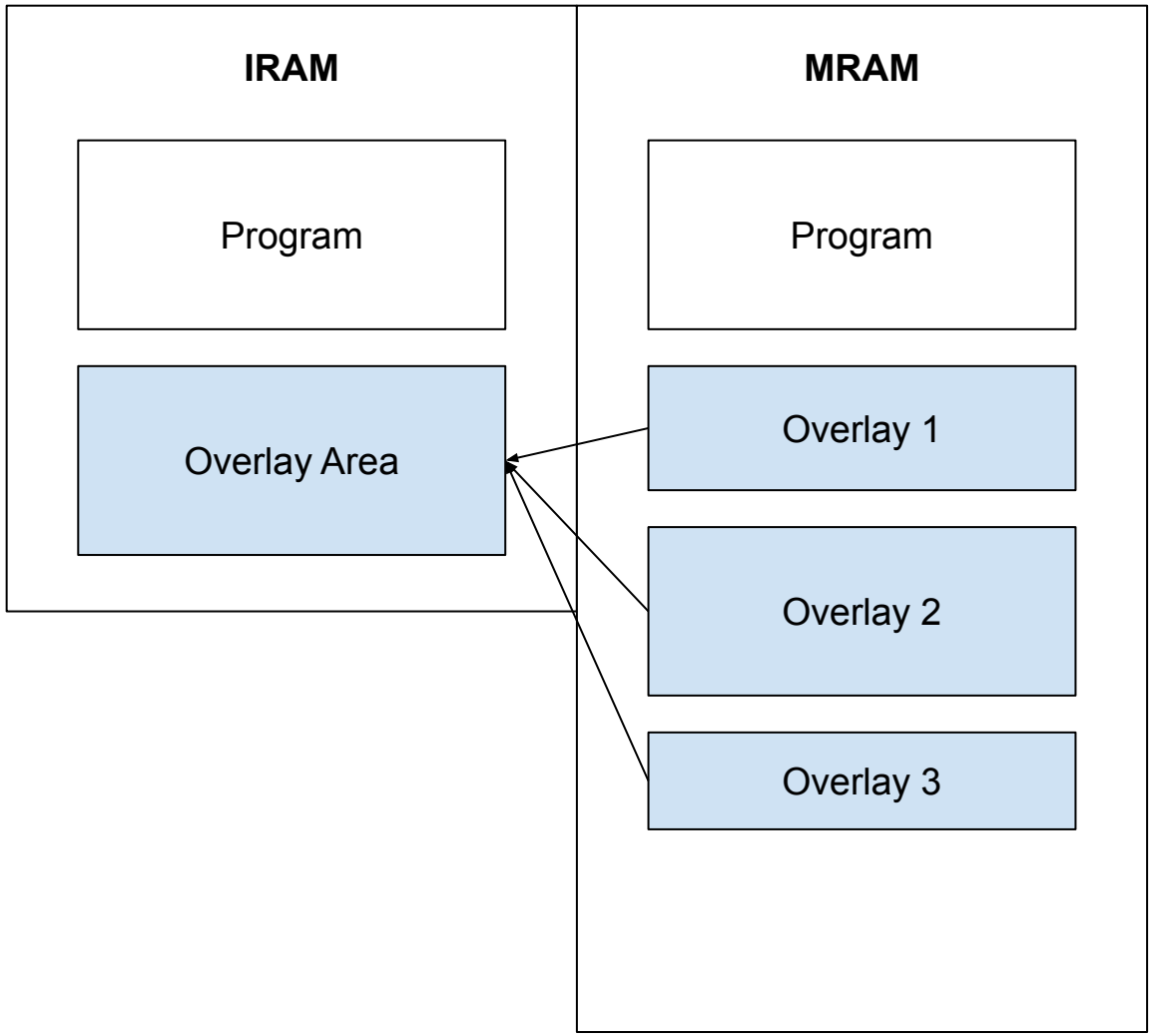
DPU

```
[...]

__host volatile uint64_t active = 1;
BARRIER_INIT(barrier, NR_TASKLETS);
INPUT_FIFO_INIT(input_fifo, INPUT_FIFO_PTR_SIZE
/*=5*/, INPUT_FIFO_DATA_SIZE /*=8*/);
OUTPUT_FIFO_INIT(output_fifo, OUTPUT_FIFO_PTR_SIZE
/*=5*/, OUTPUT_FIFO_DATA_SIZE /*=16*/);

void values_sum_compute(uint8_t* input, void* ctx);
void values_sum_reduce(uint8_t* input, uint8_t*
output, void* ctx);

int main() {
  process_inputs_all_tasklets(
    &input_fifo,
    &output_fifo,
    values_sum_compute,
    values_sum_reduce,
    sum_tasklets,
    &barrier,
    &active
  );

  return 0;
}
```

HOST

```
[...]

int main() {
  [...]

  dpu_launch(dpu_set, DPU_ASYNCHRONOUS);

  for(...){
    dpu_fifo_push_xfer(dpu_set, &input_link,
                       DPU_XFER_NO_RESET);
    dpu_fifo_push_xfer(dpu_set, &output_link,
                       DPU_XFER_NO_RESET);
    dpu_callback(dpu_set, my_callback, &cb_args,
                 DPU_CALLBACK_PARALLEL);
  }

  uint64_t active = 0;
  dpu_prepare_xfer(dpu_set, &active);
  dpu_push_xfer(dpu_set, DPU_XFER_TO_DPU, "active",
                0, sizeof(uint64_t),
                DPU_XFER_PARALLEL));

  dpu_sync(dpu_set);

  [...]
}
```

Functions Overlays (upcoming)




```c
void __function_group(0) function1() {
    printf("this is function 1\n");
}

void __function_group(1) function2() {
    printf("this is function 2\n");
}

BARRIER_INIT(main_barrier, NR_TASKLETS);

int main() {
    SYNC_AND_SWITCH_FUNCTION_GROUP(0,
      main_barrier);
    function1();
    SYNC_AND_SWITCH_FUNCTION_GROUP(1,
      main_barrier);
    ASSERT_FUNCTION_GROUP_IS_LOADED(1);
    function2();
    return 0;
}
```

Useful links

- Website
- Resource page
- Github
- SDK

Thank you

Sylvan Brocard, Technical leader on the SDK and applications on PIM.
sbrocard@upmem.com