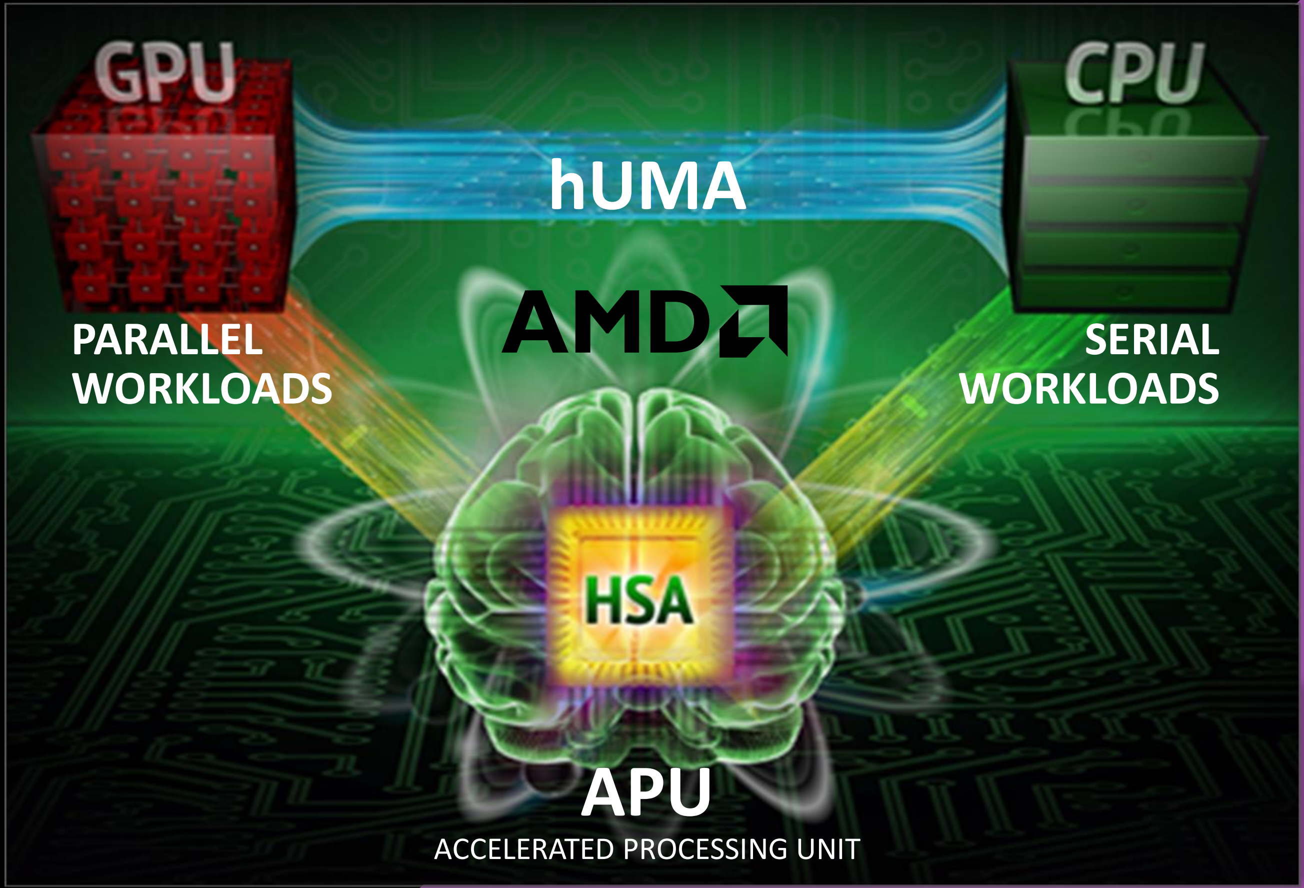
# “KAVERI” AND THE HSA ADVANTAGE

TZACHI COHEN

FEBRUARY 10, 2014

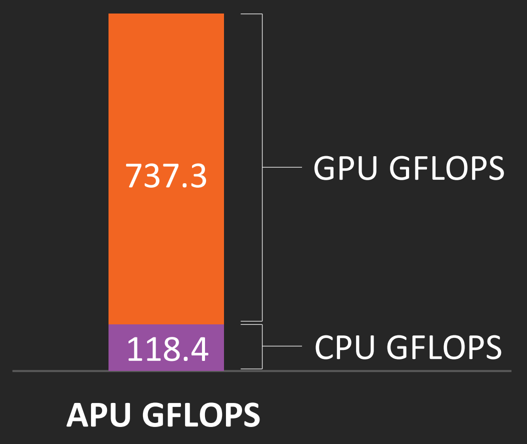
# WHAT IS HSA?




**Processor design** that makes it easy to harness the **entire computing power of an APU** for faster and more power-efficient devices, including personal computers, tablets, smartphones and cloud servers

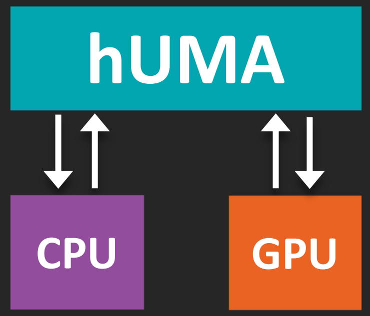
# HSA FEATURES OF "KAVERI"

## UNLOCKING All OF KAVERI'S GFLOPS



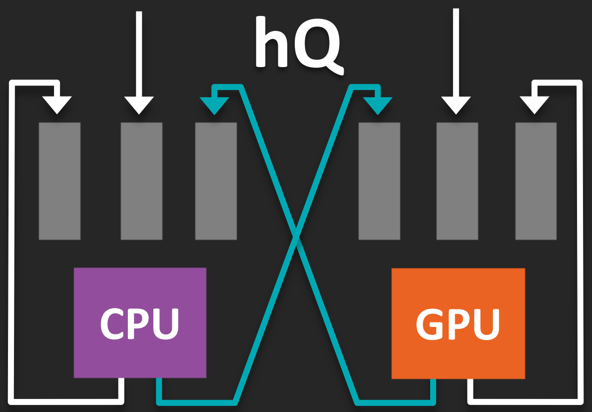

- Access to full potential of Kaveri's APU compute power

## EQUAL ACCESS TO ENTIRE MEMORY




- GPU and CPU have uniform visibility into entire memory space

## ALL-PROCESSORS-EQUAL




- GPU and CPU have equal flexibility to be used to create and dispatch work items

# OPENCL™ 2.0 FEATURES AND HSA

AMD

- Some of the key features of OpenCL 2.0 and their HSA mapping

| OpenCL 2.0 Feature | HSA Mapping |
|---|---|
| Shared Virtual Memory | hUMA |
| Dynamic Parallelism | hQ |
| Pipes | hUMA, hQ |
| C11 Atomics | Platform Atomics |

# HSA ADVANTAGES OF "KAVERI"

- HSA features make "Kaveri" the FIRST full OpenCL 2.0 capable chip
- Ease of programming to use the GPU for compute
- Easy access to up to 12 Compute Cores*
- More applications from ease of use
- Better user experiences

*for more information on Compute Cores please visit AMD.com/computecores

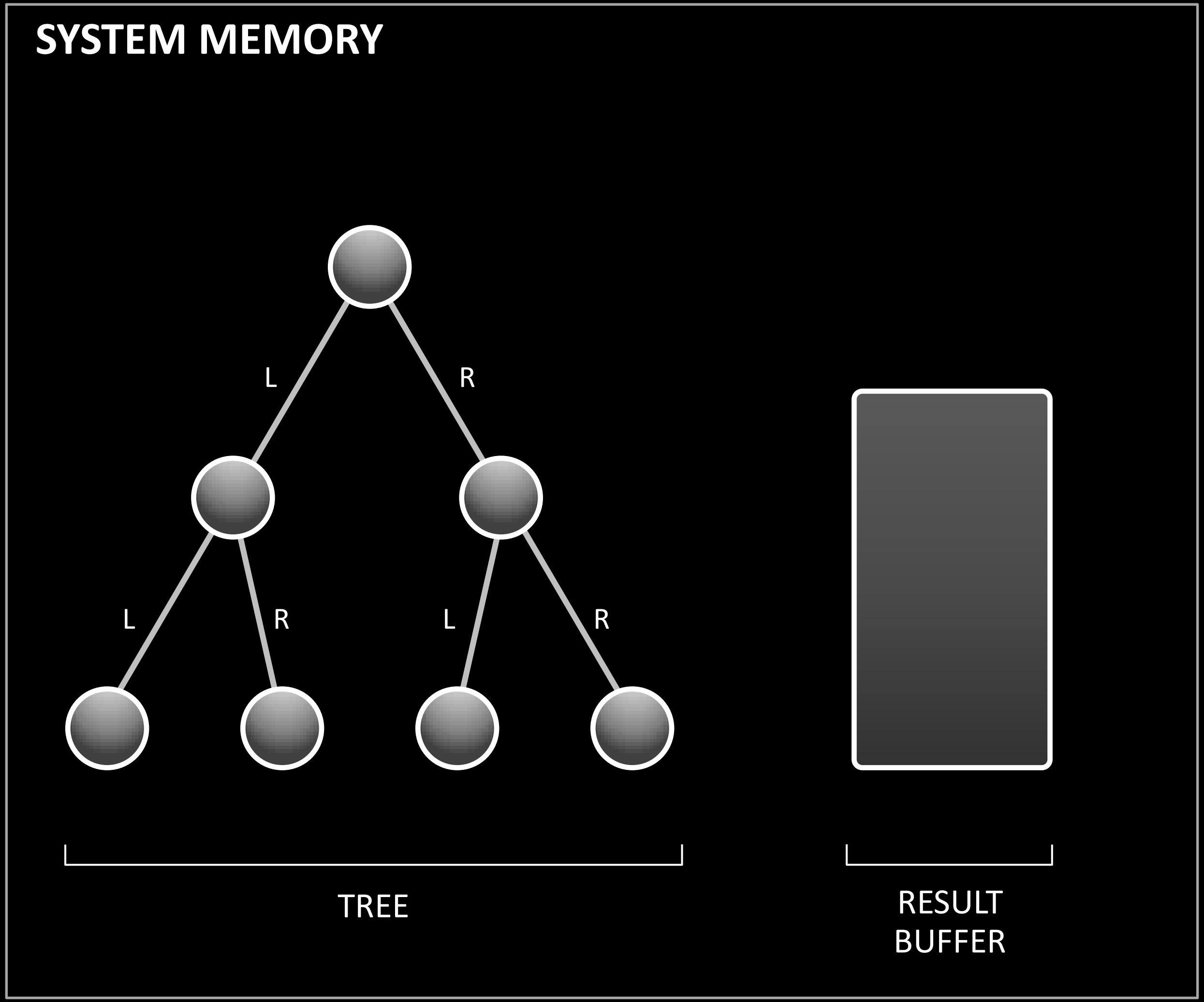
# USE CASES SHOWING HSA ADVANTAGE

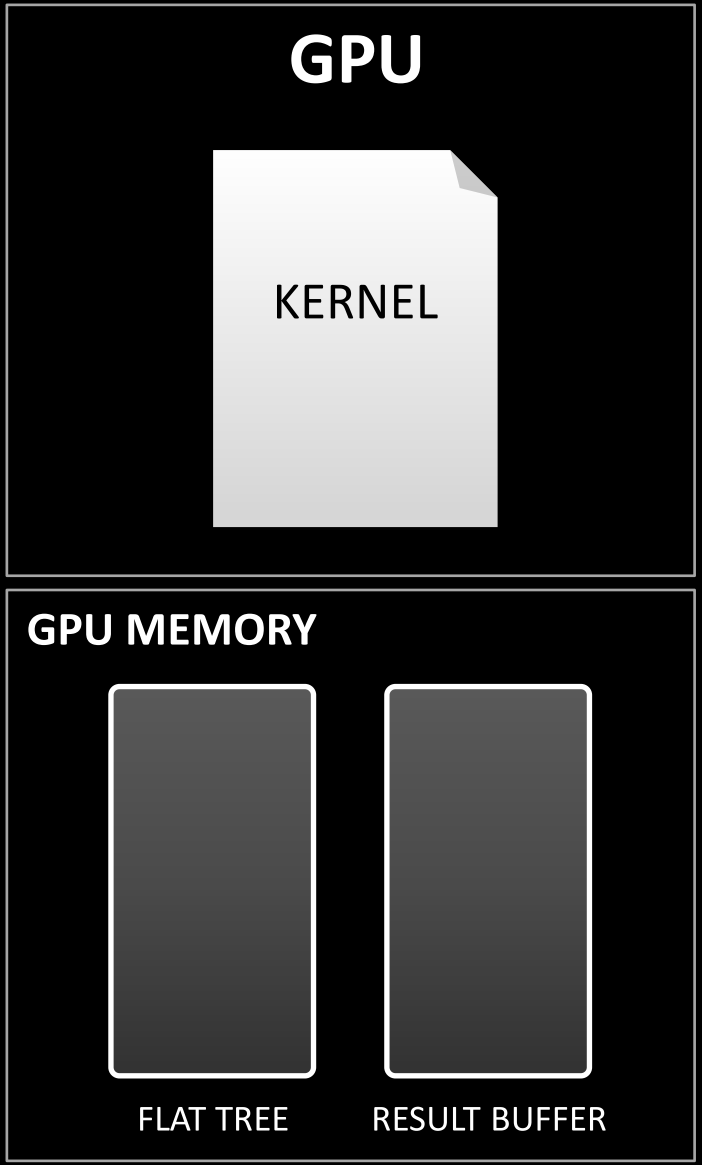
| Programming Technique | Use Case Description | HSA Advantage |
|---|---|---|
| **Data Pointers** | **Binary tree searches**<br>GPU performs searches in a CPU created binary tree | GPU can access existing data structures containing pointers<br>Higher performance through parallel operations |
| **Platform Atomics** | **Binary tree updates**<br>CPU and GPU operating simultaneously on the tree, both doing modifications | CPU and GPU can synchronize using Platform Atomics<br>Higher performance through parallel operations |
| **Large Data Sets** | **Hierarchical data searches**<br>Applications include object recognition, collision detection, global illumination, BVH | GPU can operate on huge models in place<br>Higher performance through parallel operations |
| **CPU Callbacks** | **Middleware user-callbacks**<br>GPU processes work items, some of which require a call to a CPU function to fetch new data | GPU can invoke CPU functions from within a GPU kernel<br>Simpler programming does not require “split kernels”<br>Higher performance through parallel operations |

# Data Pointers

# DATA POINTERS

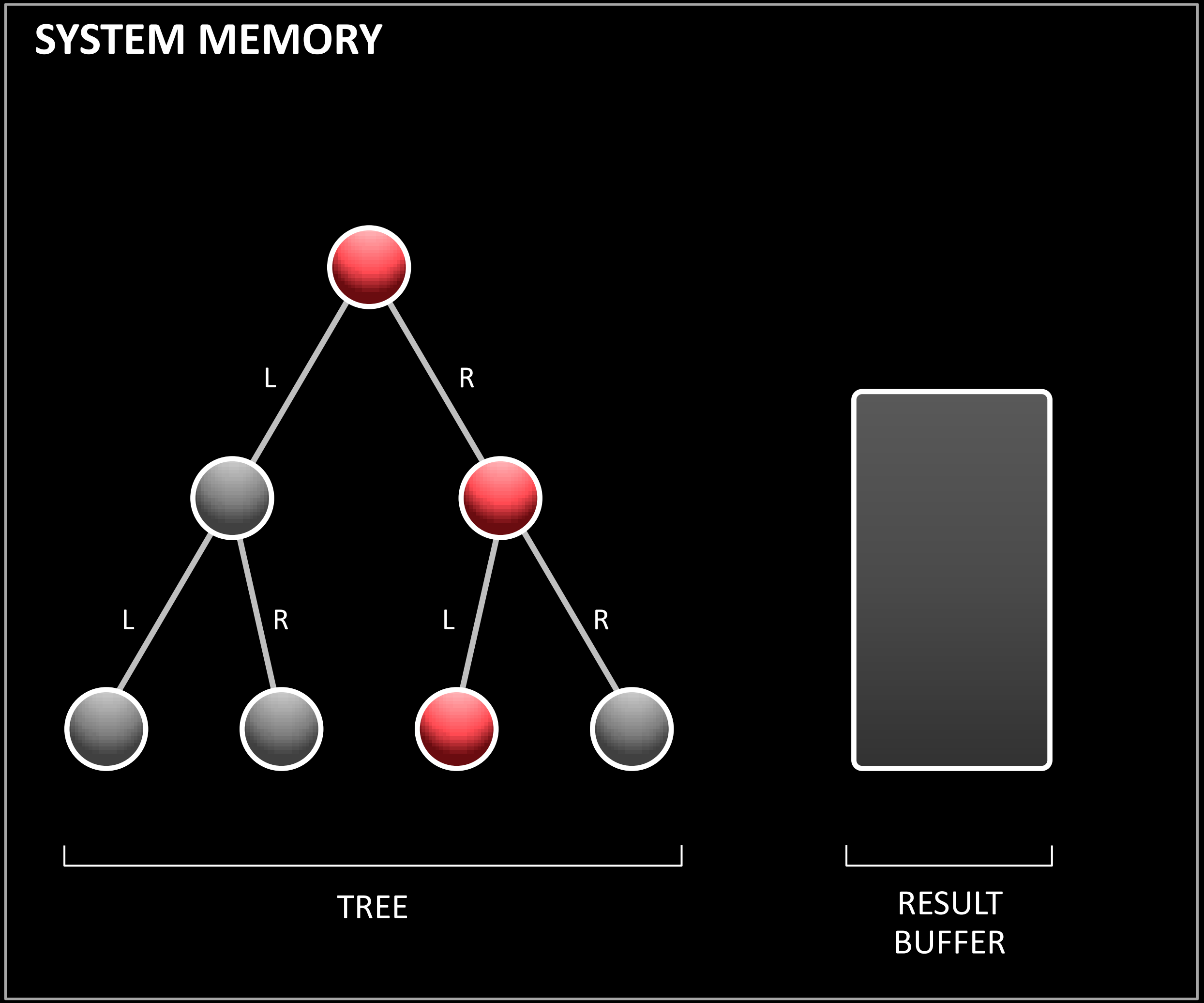
Legacy

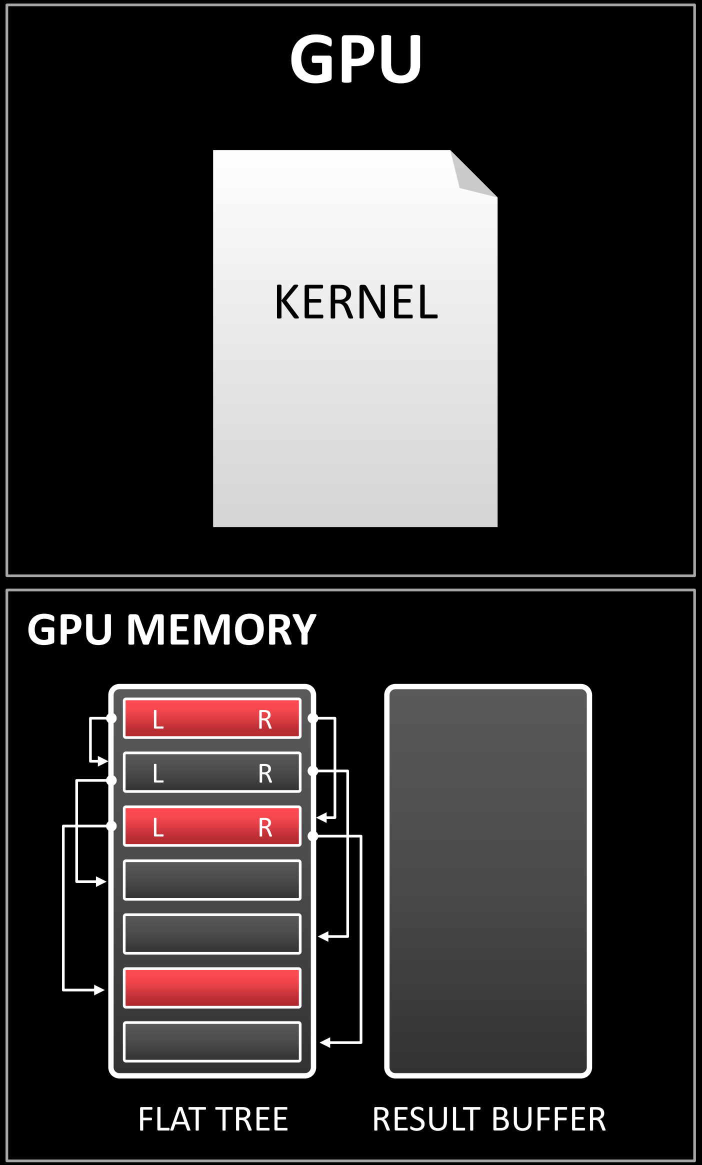


# DATA POINTERS

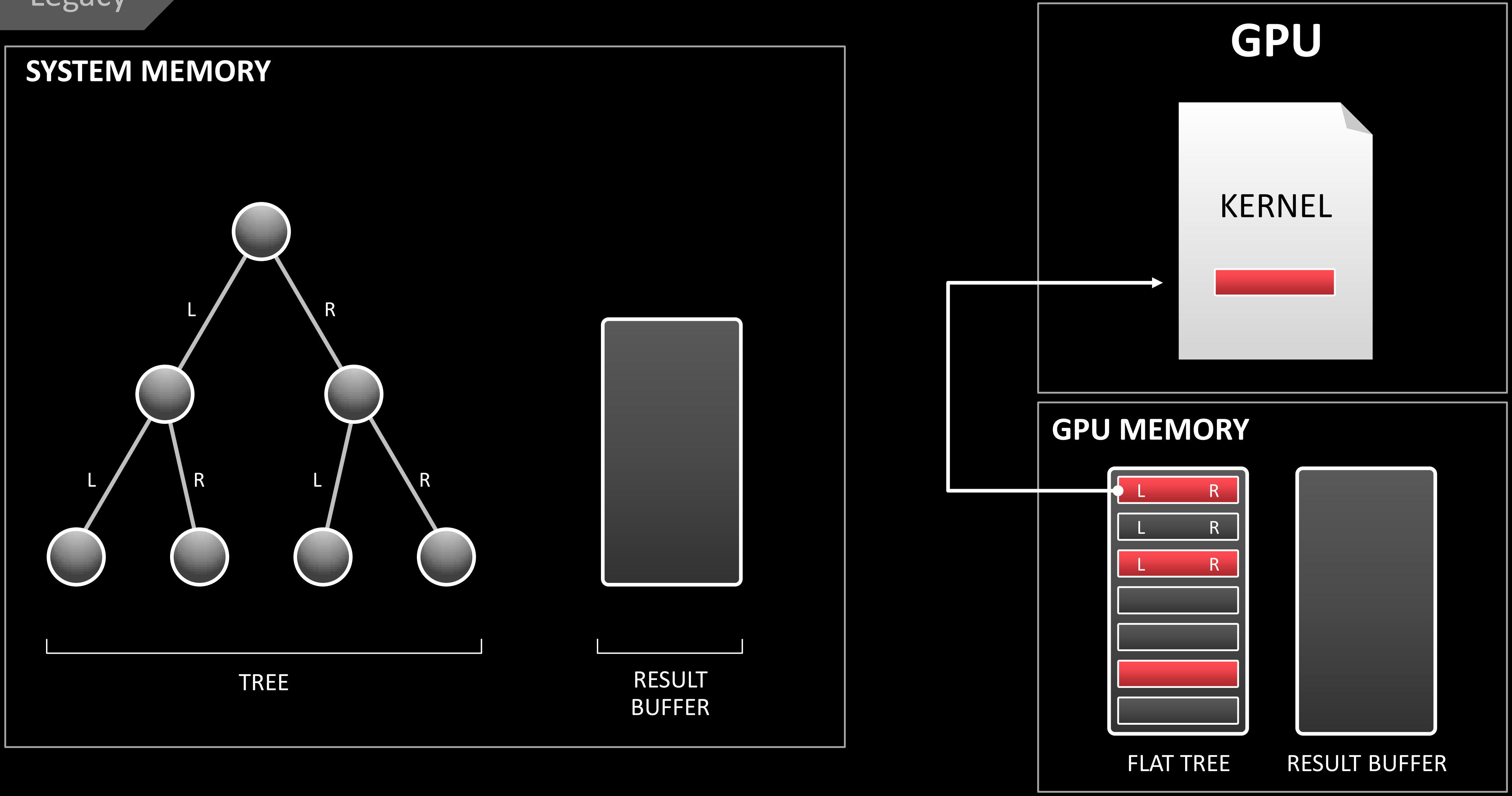
Legacy



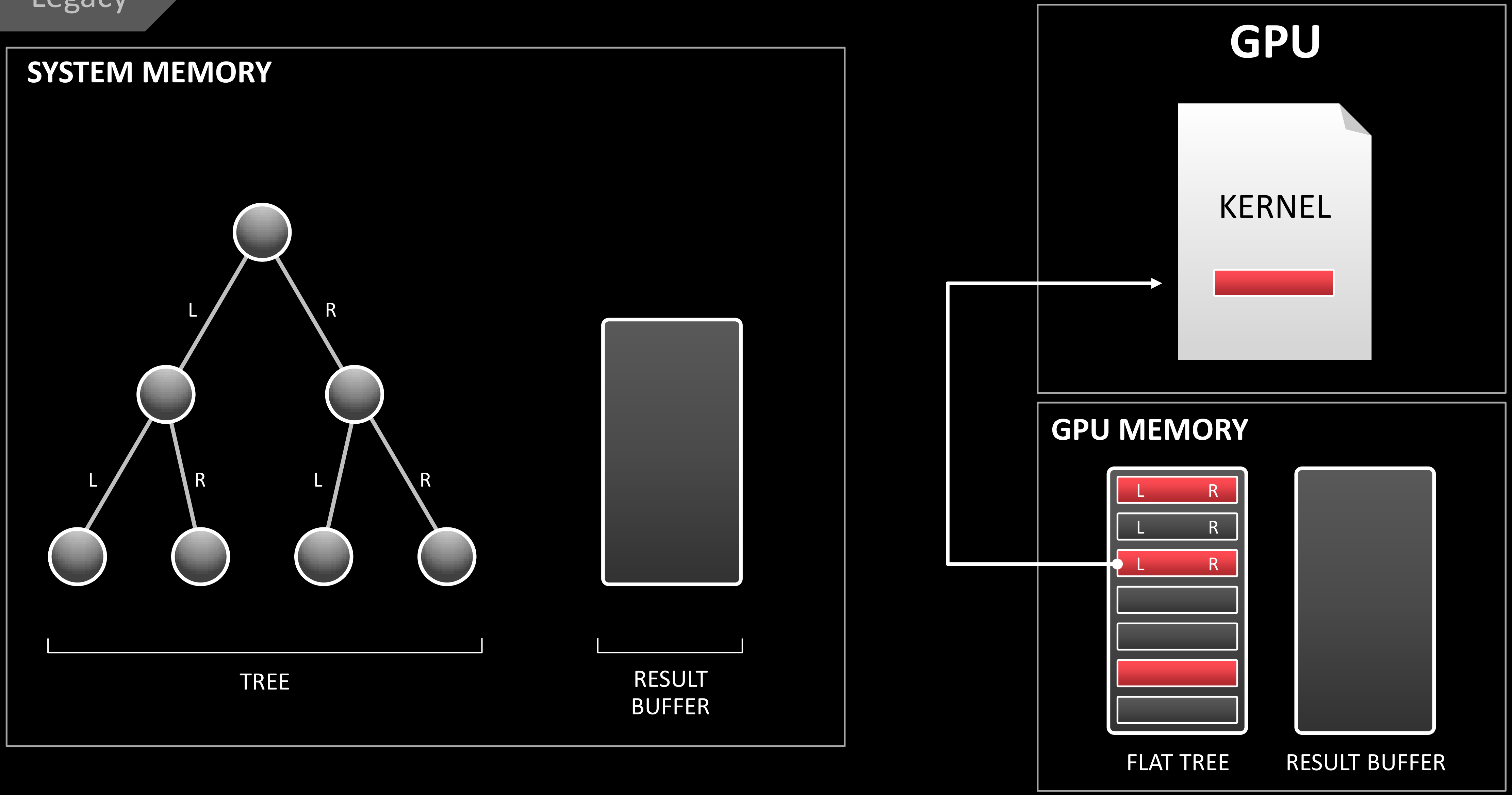
# DATA POINTERS

Legacy



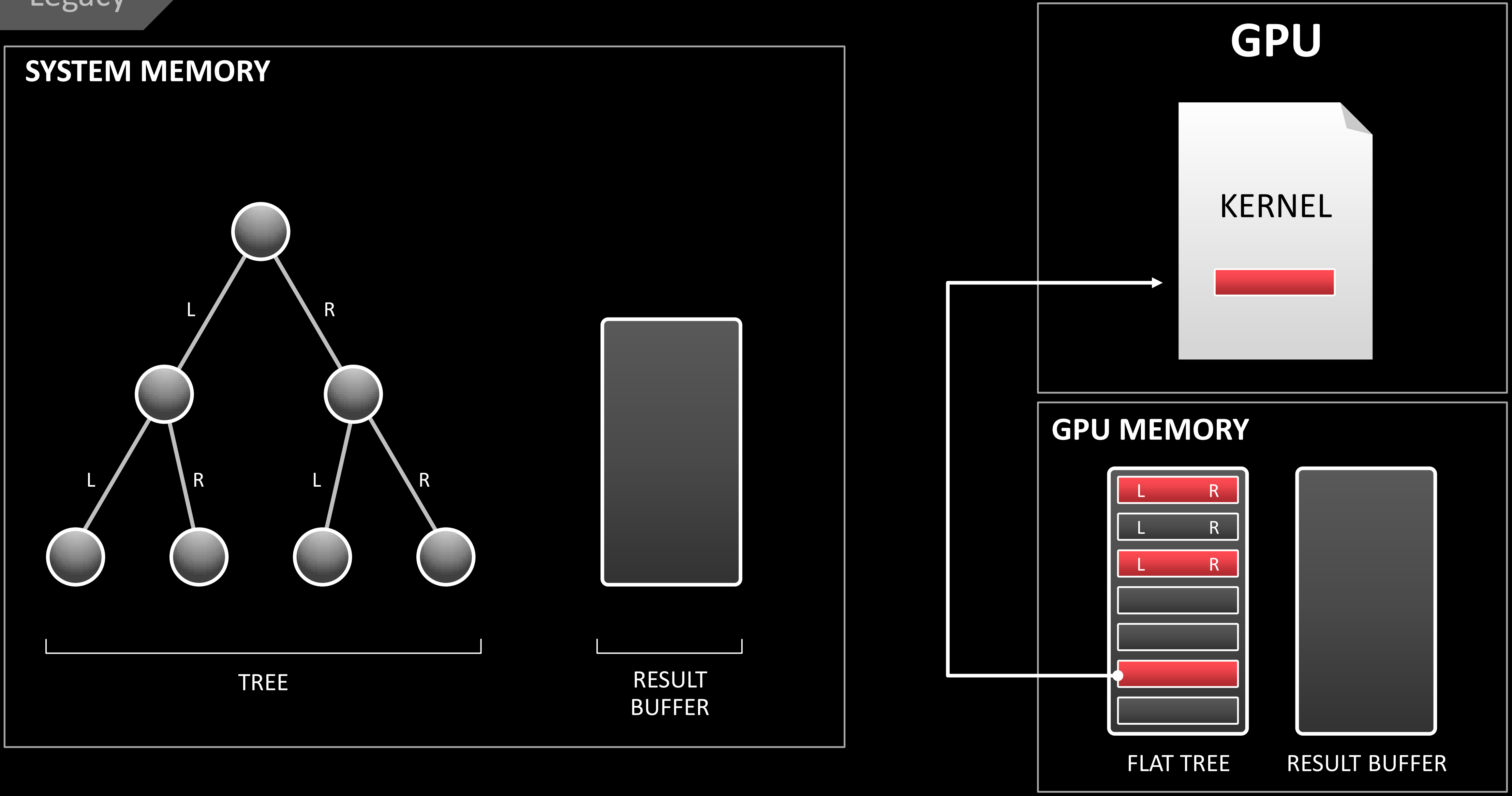
# DATA POINTERS

Legacy



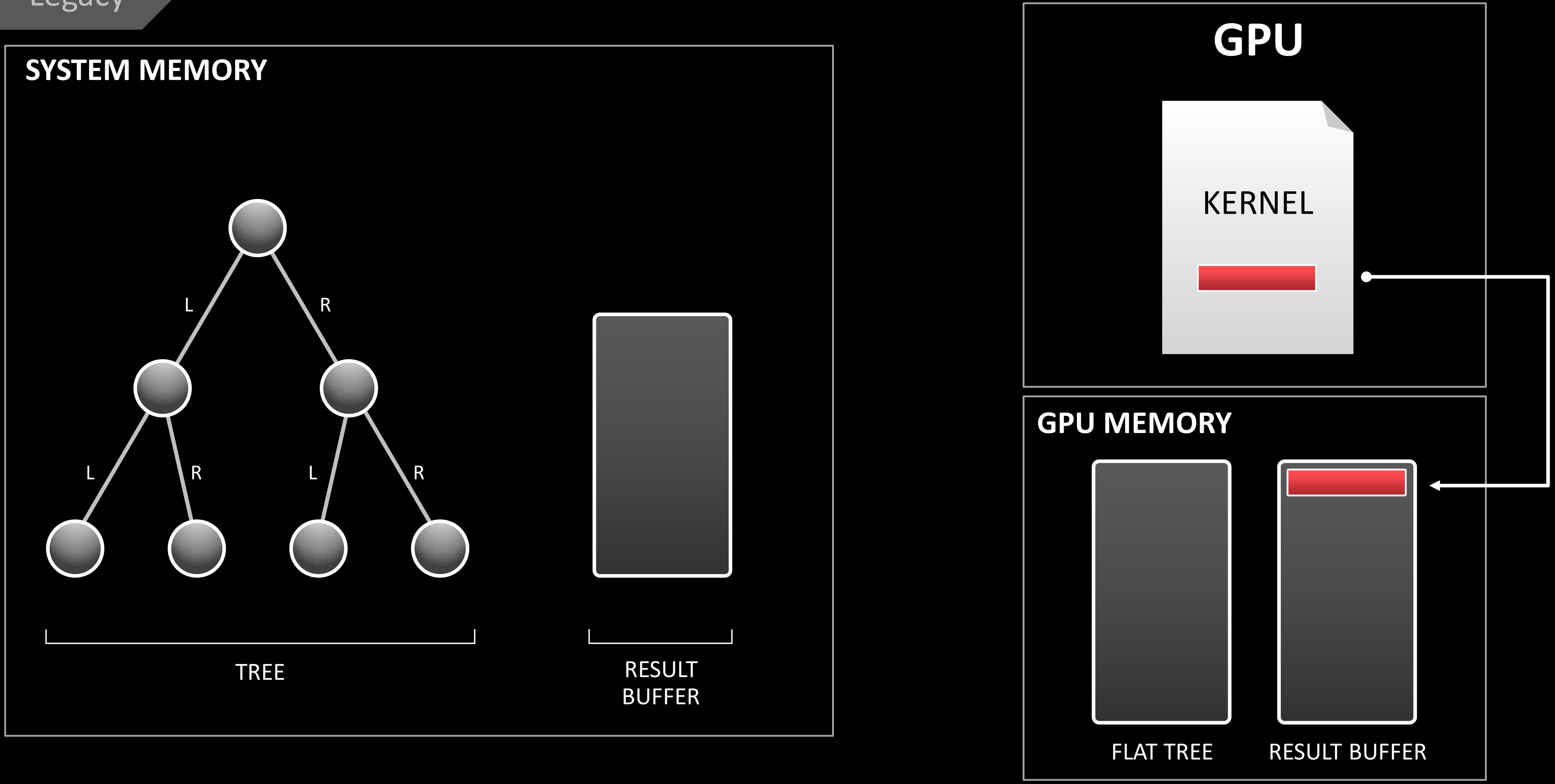
# DATA POINTERS

Legacy



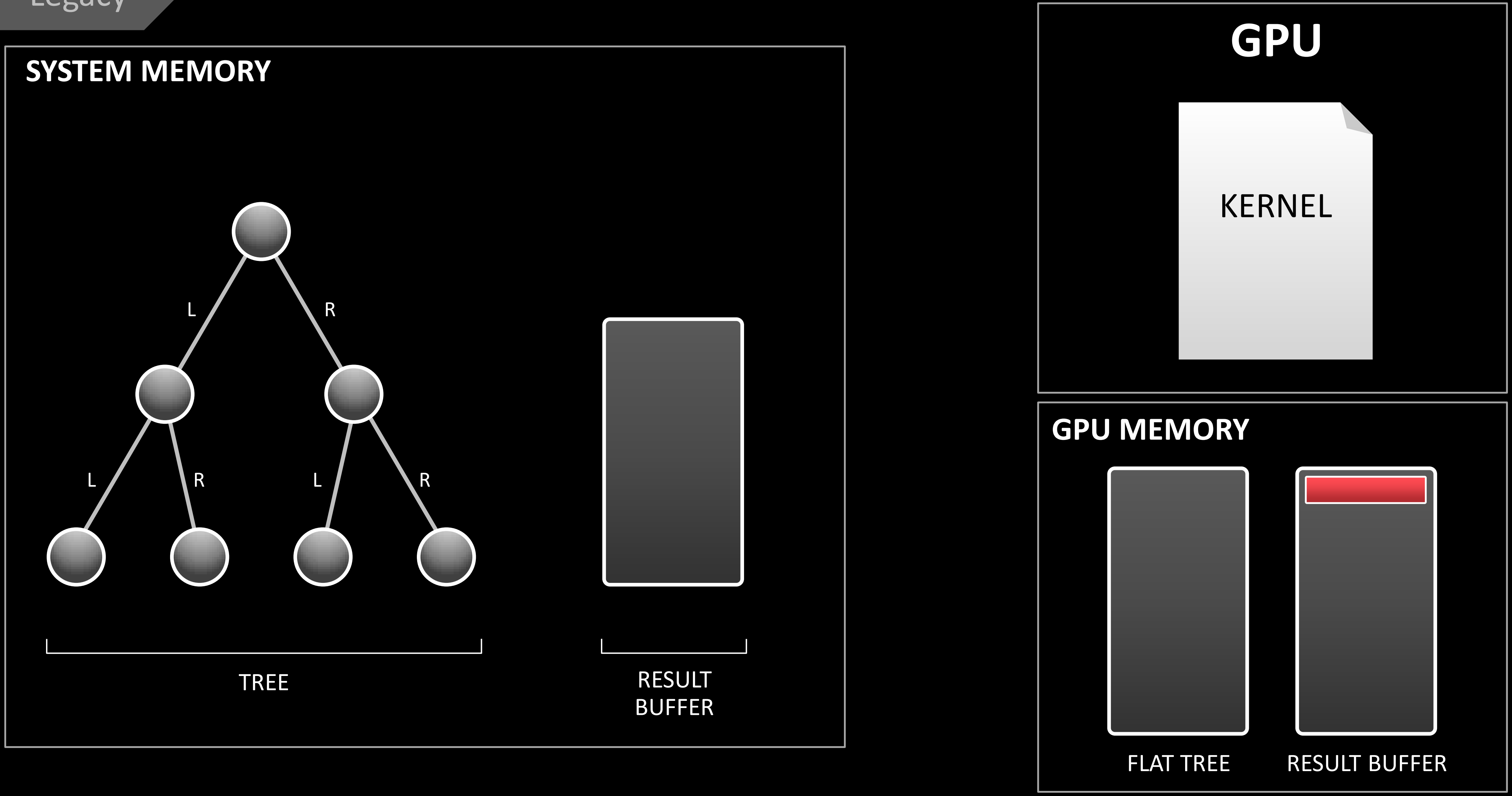
# DATA POINTERS

Legacy



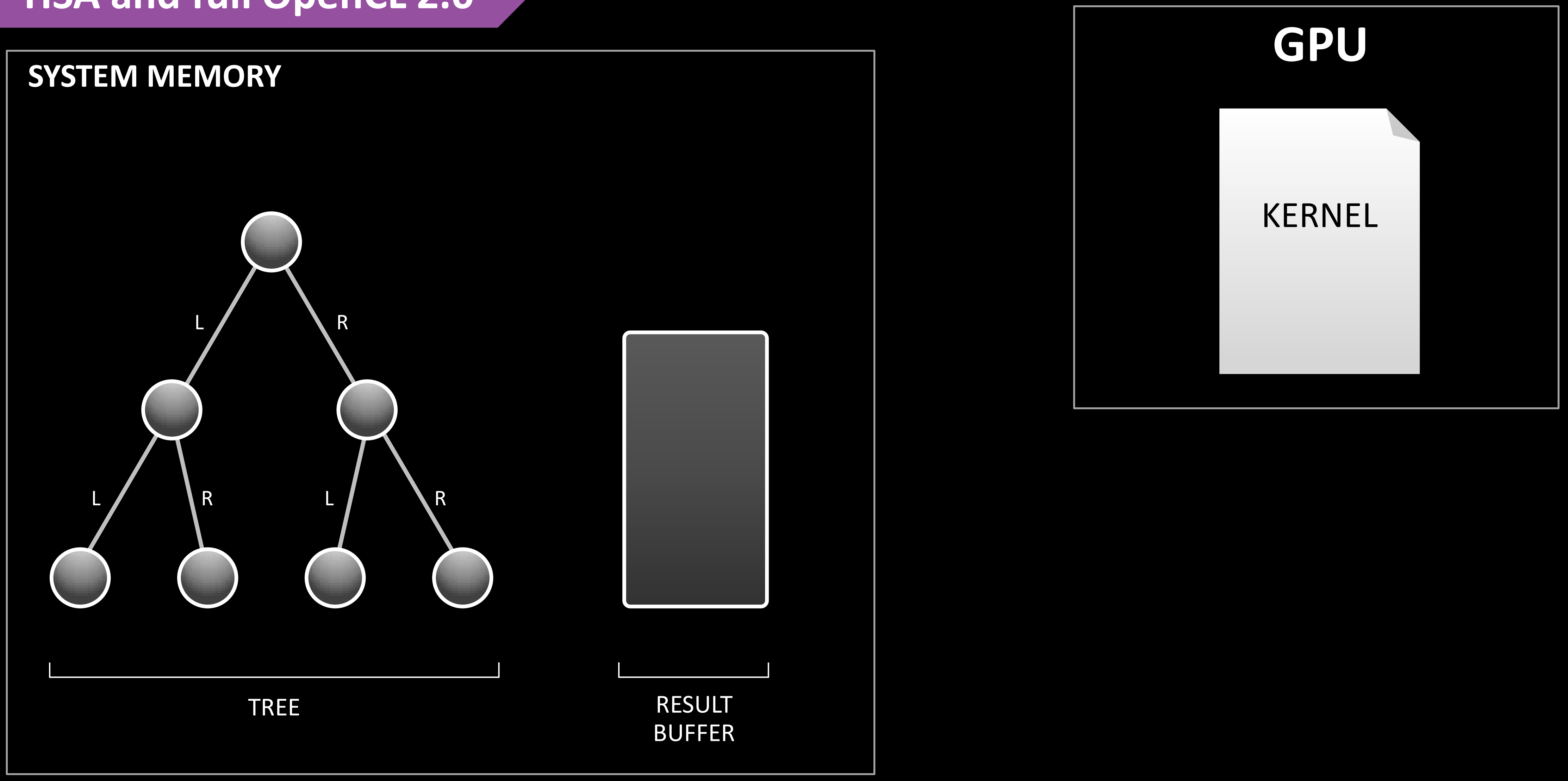
# DATA POINTERS

## Legacy



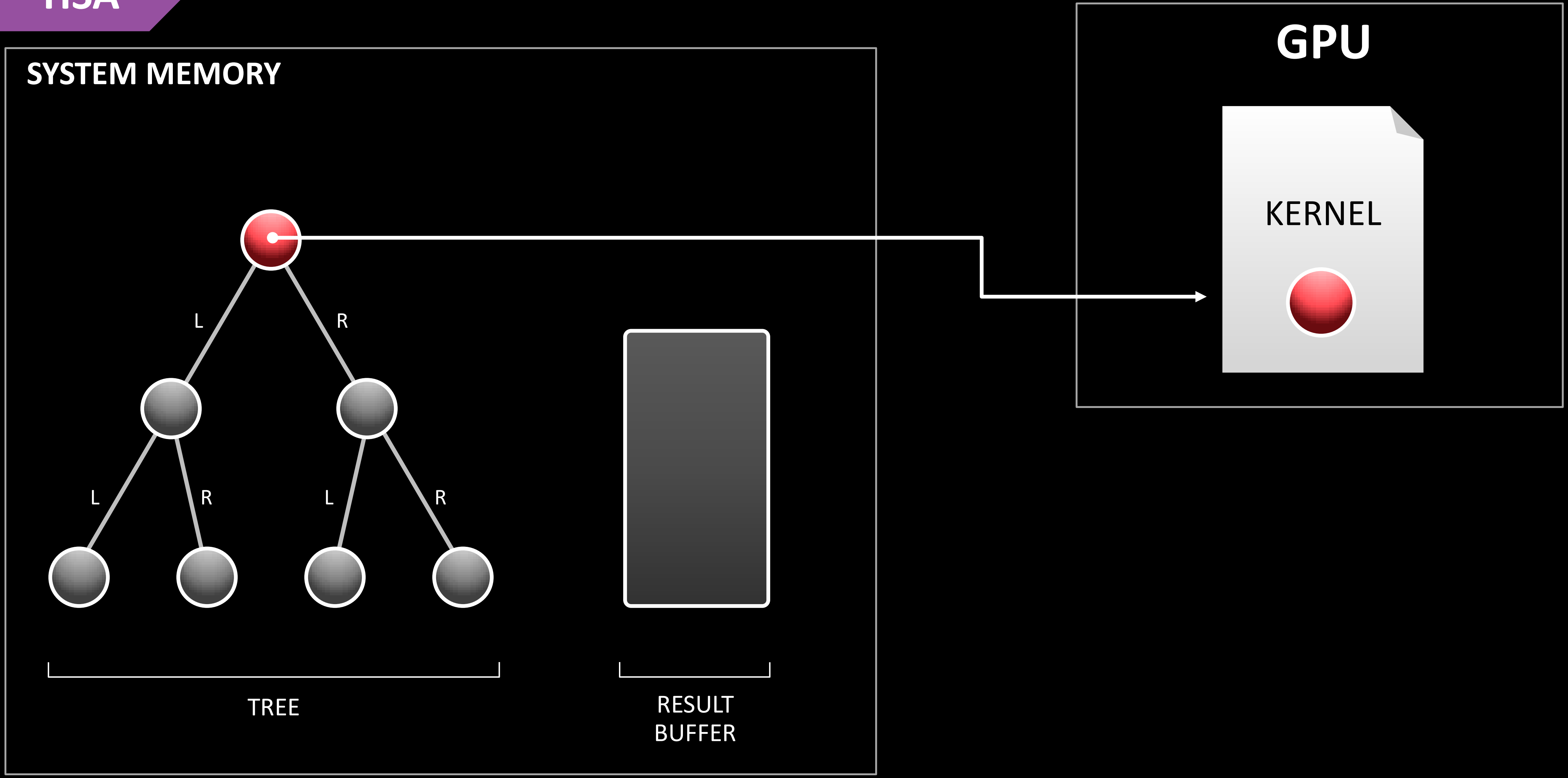
# DATA POINTERS

## HSA and full OpenCL 2.0



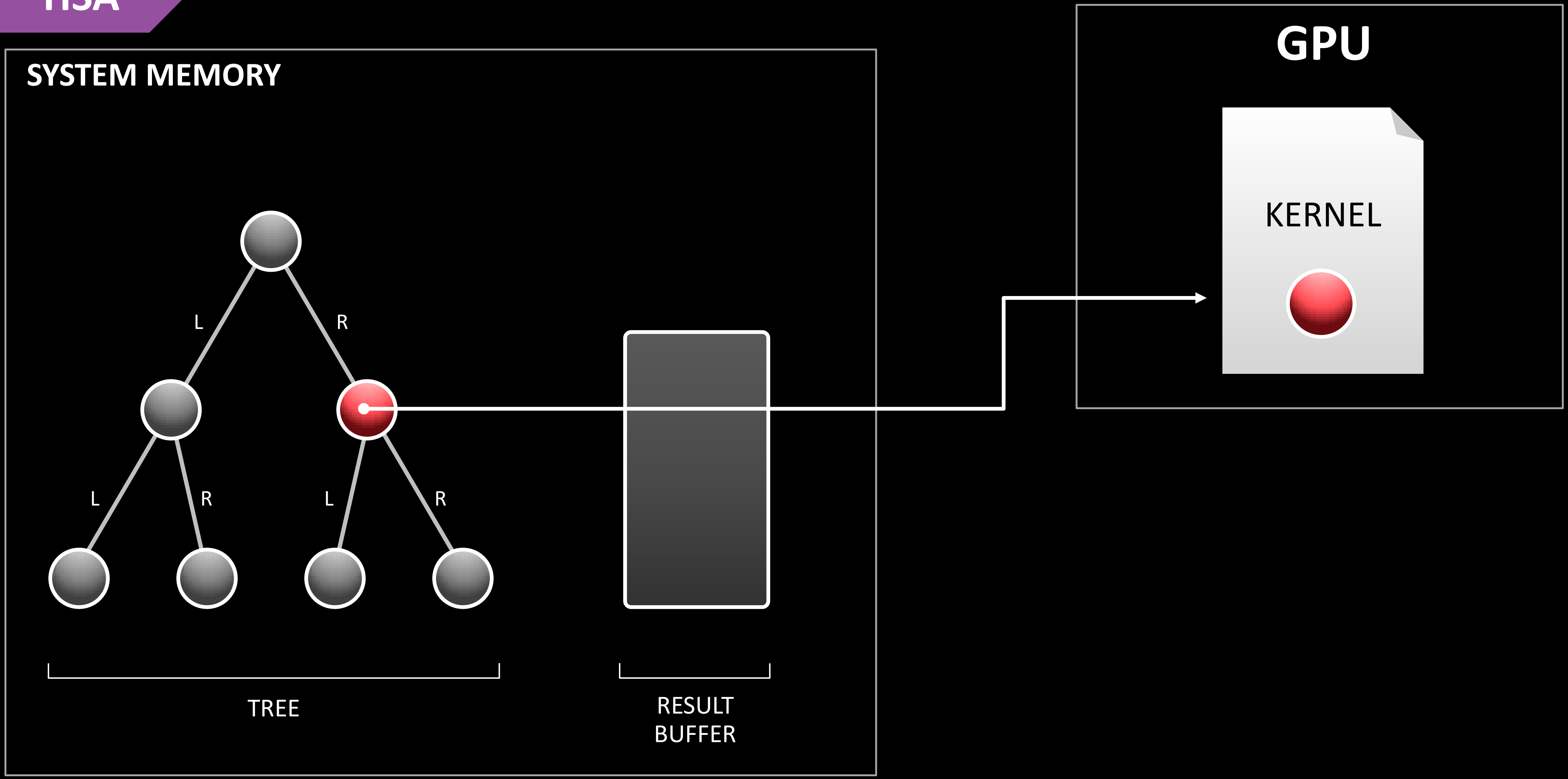
# DATA POINTERS

## HSA



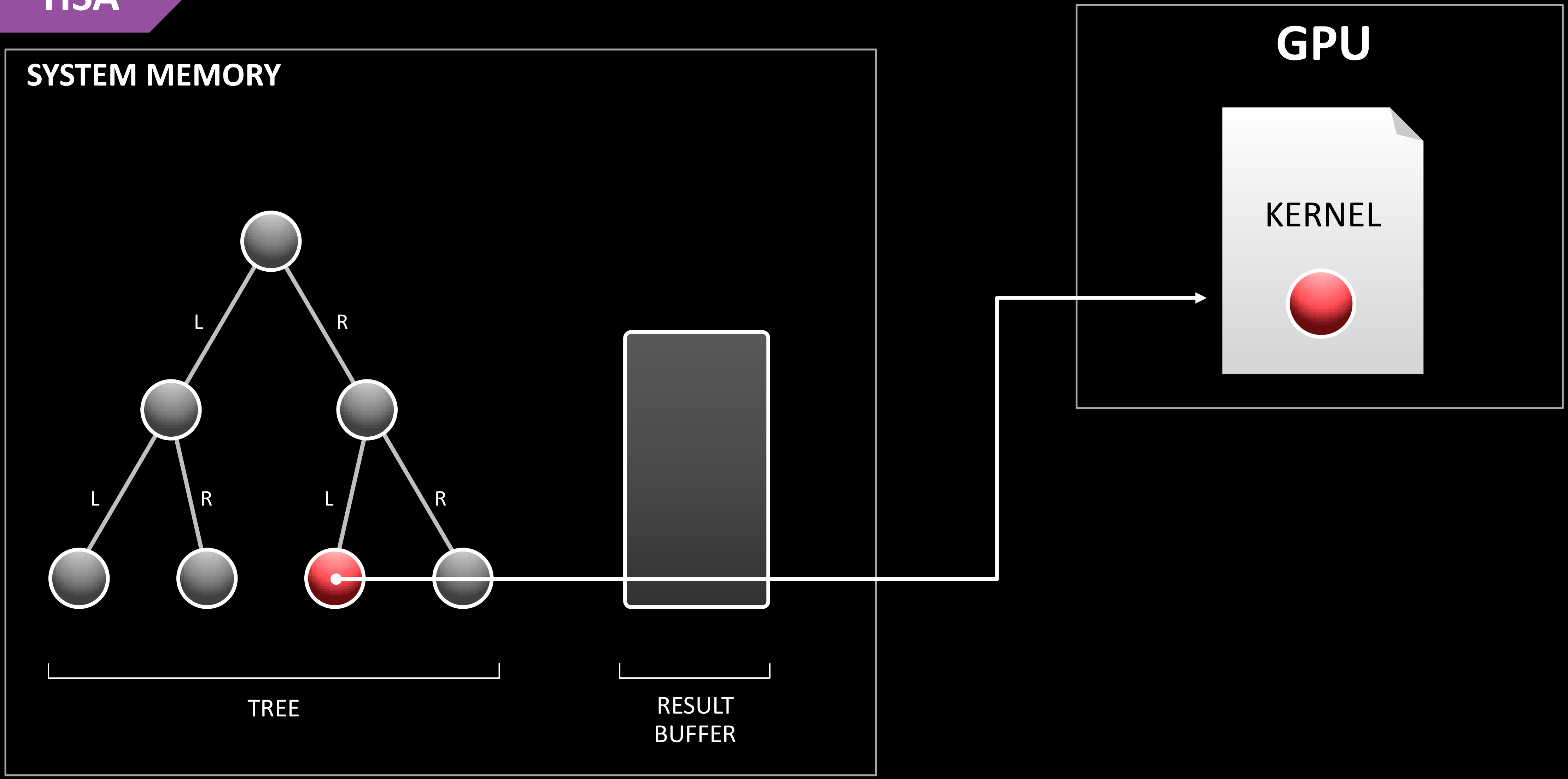
# DATA POINTERS

**HSA**



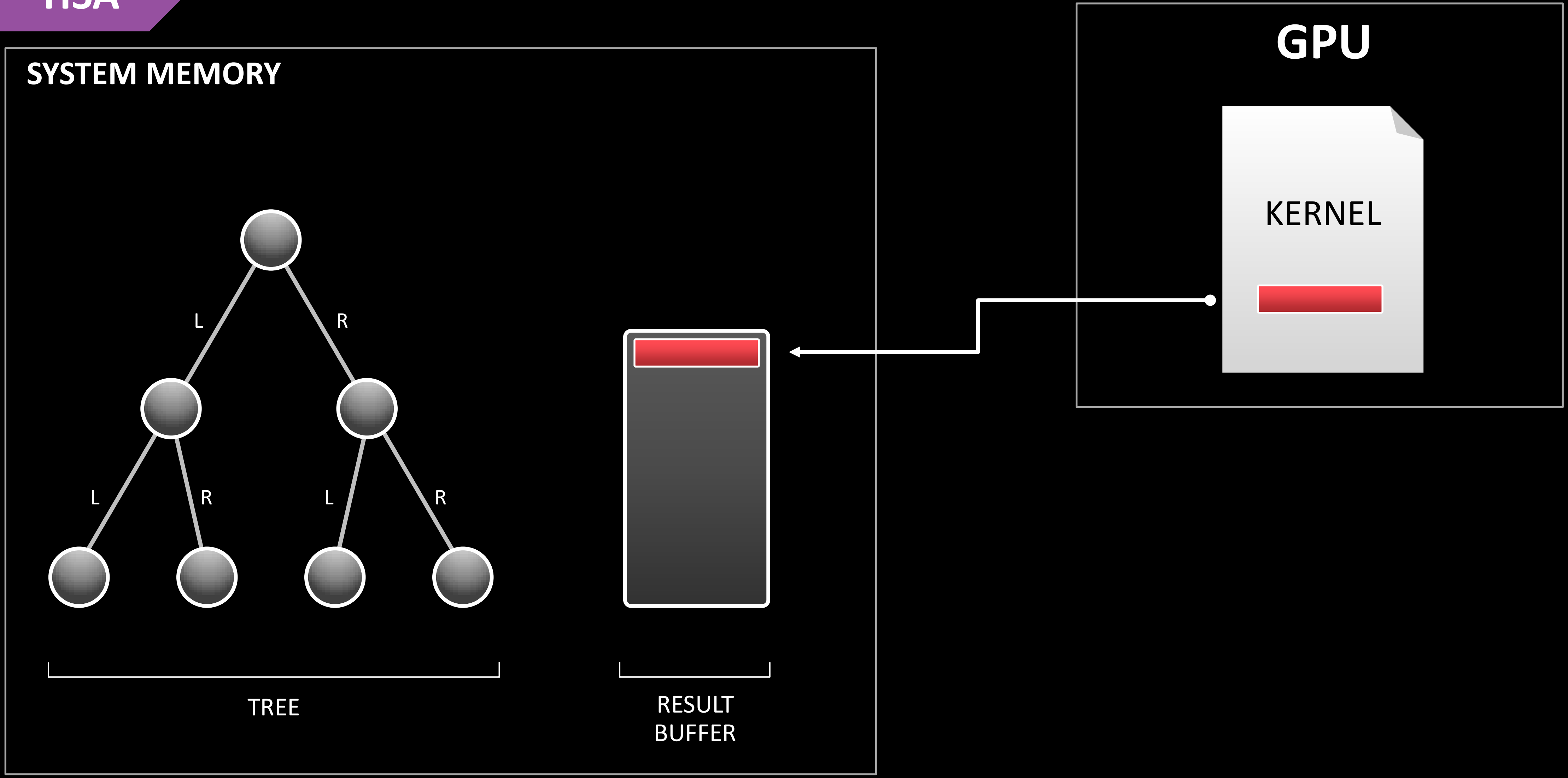
# DATA POINTERS

## HSA



# DATA POINTERS

**HSA**



# DATA POINTERS - CODE COMPLEXITY

AMD

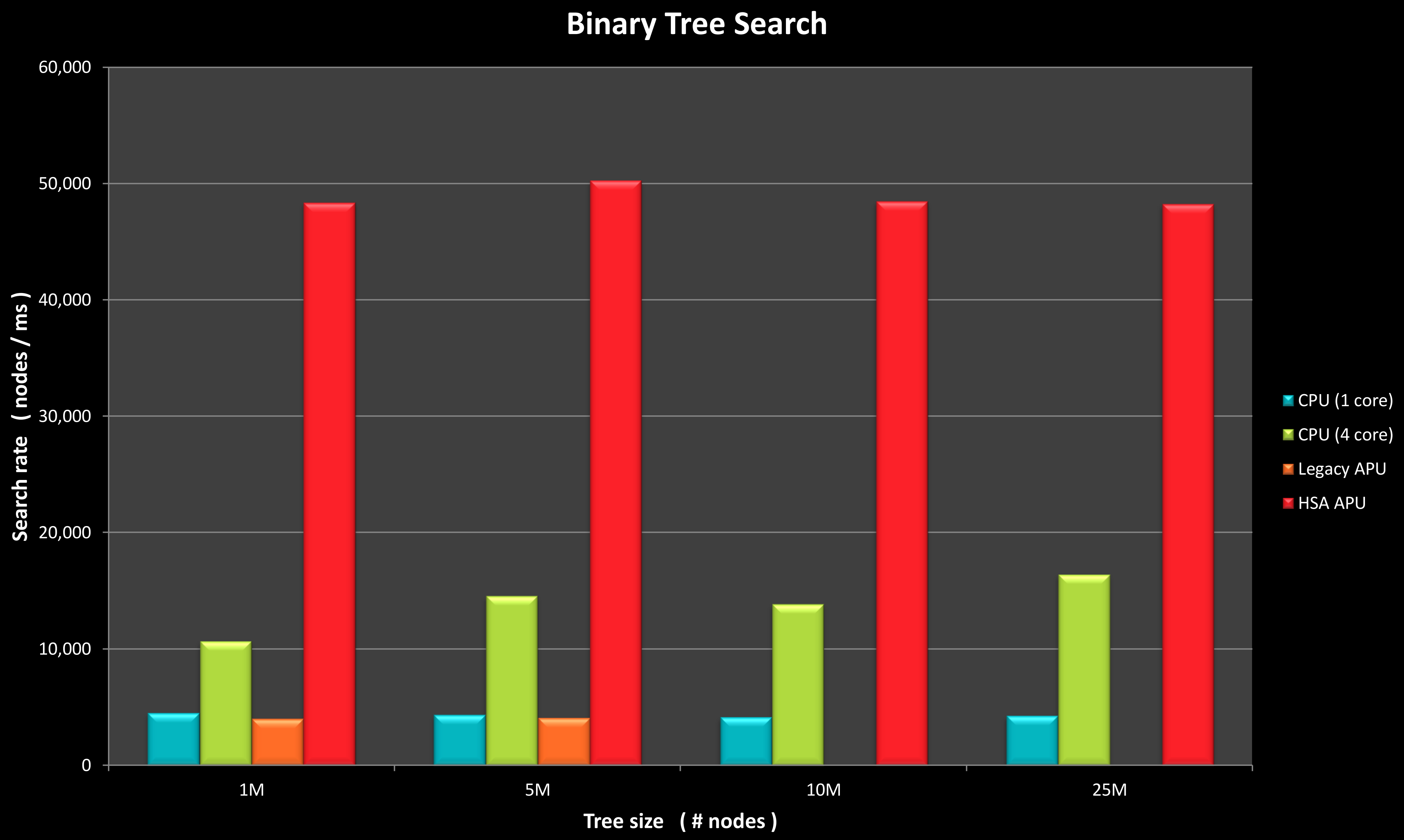
HSA

```
static void run_hsa_path()
{
    /* Allocation and initialization */
    tree = (node *) clSVMAlloc(context, CL_MEM_READ_ONLY,
            num_nodes * sizeof(node), 0);
    initialize_nodes(tree, num_nodes);
    root = construct_BST(num_nodes, tree);

    search_keys = (int *) clSVMAlloc(context, CL_MEM_READ_ONLY,
            num_search_keys * sizeof(int), 0);
    initialize_search_keys(search_keys, num_search_keys, sort_input);

    found_key_nodes = (node **) clSVMAlloc(context, CL_MEM_WRITE_ONLY,
            num_search_keys * sizeof(node *), 0);
    memset(found_key_nodes, 0, num_search_keys * sizeof(node *));

    /* GPU work enqueue */
    clSetKernelArgSVMPointer(search_kernel, 0, root);
    clSetKernelArgSVMPointer(search_kernel, 1, search_keys);
    clSetKernelArgSVMPointer(search_kernel, 2, &num_search_keys);
    clSetKernelArgSVMPointer(search_kernel, 3, found_key_nodes);

    clEnqueueNDRangeKernel(queue, search_kernel, 1, NULL,
            &num_search_keys, &preferredLocalSize, 0, NULL, &kernel_event);

    clFinish(queue);

    /* Cleanup */
    clSVMFree(context, tree);
    clSVMFree(context, found_key_nodes);
    clSVMFree(context, search_keys);
}
```

Legacy

```
static void run_ocl_path()
{
    /* Allocation and initialization */
    tree = (node *) malloc(num_nodes * sizeof(node));
    initialize_nodes(tree, num_nodes);
    root = construct_BST(num_nodes, tree);

    search_keys = (int *) malloc(num_search_keys * sizeof(int));
    initialize_search_keys(search_keys, num_search_keys, sort_input);

    found_keys = (int *) malloc(num_search_keys * sizeof(int));
    memset(found_keys, 0, num_search_keys * sizeof(int));

    ocl_tree = (ocl_node *) malloc(num_nodes * sizeof(ocl_node));

    cl_mem cl_ocl_tree = clCreateBuffer(context, CL_MEM_READ_ONLY,
            num_nodes * sizeof(ocl_node), NULL, &status);
    cl_mem cl_search_keys = clCreateBuffer(context, CL_MEM_READ_ONLY,
            num_search_keys * sizeof(int), NULL, &status);
    cl_mem cl_found_nodes_id = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
            num_search_keys * sizeof(int), NULL, &status);

    /* The tree is converted to its array form */
    int root_id;
    initialize_ocl_nodes(ocl_tree, num_nodes);
    convert_tree_to_array(root, ocl_tree, &root_id);

    /* Copy the tree and search keys array to the GPU */
    clEnqueueWriteBuffer(queue, cl_ocl_tree, CL_TRUE, 0,
            num_nodes * sizeof(ocl_node), ocl_tree, 0, NULL, NULL);

    clEnqueueWriteBuffer(queue, cl_search_keys, CL_TRUE, 0,
            num_search_keys * sizeof(int), search_keys, 0, NULL, NULL);

    /* GPU work enqueue */
    clSetKernelArg(search_kernel, 0, sizeof(cl_ocl_tree), &cl_ocl_tree);
    clSetKernelArg(search_kernel, 1, sizeof(cl_int), &root_id);
    clSetKernelArg(search_kernel, 2, sizeof(cl_search_keys), &cl_search_keys);
    clSetKernelArg(search_kernel, 3, sizeof(cl_int), &num_search_keys);
    clSetKernelArg(search_kernel, 4, sizeof(cl_found_nodes_id), &cl_found_nodes_id);

    clEnqueueNDRangeKernel(queue, search_kernel, 1, NULL,
            &num_search_keys, &preferredLocalSize, 0, NULL, NULL);

    clFinish(queue);

    /* Copy the results back from the GPU */
    clEnqueueReadBuffer(queue, cl_found_nodes_id, CL_TRUE, 0,
            num_search_keys * sizeof(int), found_keys, 0, NULL, NULL);

    /* Cleanup */
    free(ocl_tree);
    free(tree);
    free(found_keys);
    free(search_keys);

    clReleaseMemObject(cl_ocl_tree);
    clReleaseMemObject(cl_search_keys);
    clReleaseMemObject(cl_found_nodes_id);
}


static void initialize_ocl_nodes(ocl_node *ocl_tree, long long int num_nodes)
{
    for (int i = 0; i < num_nodes; i++) {
        ocl_tree[i].left = -1;
        ocl_tree[i].right = -1;
    }
}


static void convert_tree_to_array(node *root, ocl_node *ocl_tree, int *root_id)
{
    node **tree_queue;
    node *tmp;

    tree_queue = (node **)calloc(num_nodes, sizeof(node *));

    long long int front = 0;
    long long int rear  = 0;

    tree_queue[rear] = root;
    ocl_tree[rear].value = root->value;
    rear++;

    *root_id = 0;

    while (front != rear) {
        tmp = tree_queue[front];
        if (!tmp)
            break;

        if (tmp->left) {
            tree_queue[rear] = tmp->left;
            ocl_tree[rear].value = tmp->left->value;
            ocl_tree[front].left = (int)rear;
            rear++;
        }

        if (tmp->right) {
            tree_queue[rear] = tmp->right;
            ocl_tree[rear].value = tmp->right->value;
            ocl_tree[front].right = (int)rear;
            rear++;
        }

        front++;
    }

    if (tree_queue)
        free(tree_queue);
}
```

# DATA POINTERS - PERFORMANCE




Measured in AMD labs Jan 1-3 on system shown in back up slide

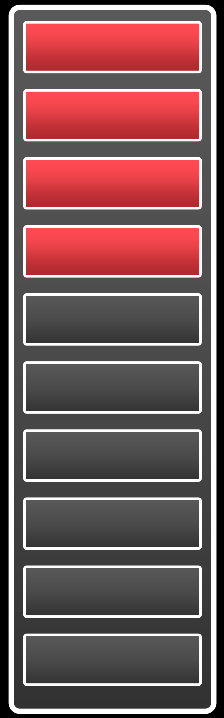
# Platform Atomics

# PLATFORM ATOMICS

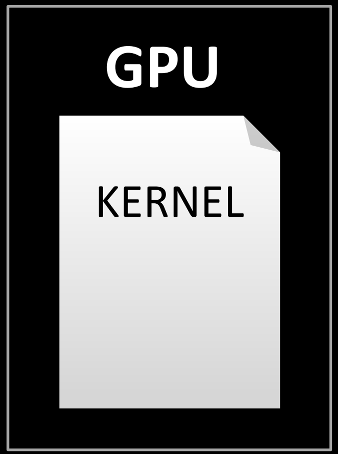
Legacy

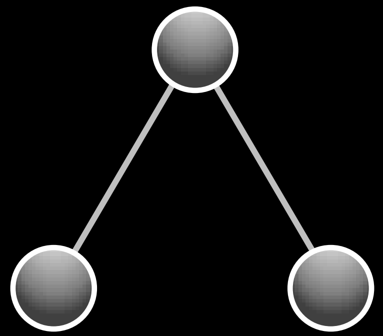
Only GPU can work on input array

**Concurrent processing not possible**



INPUT BUFFER



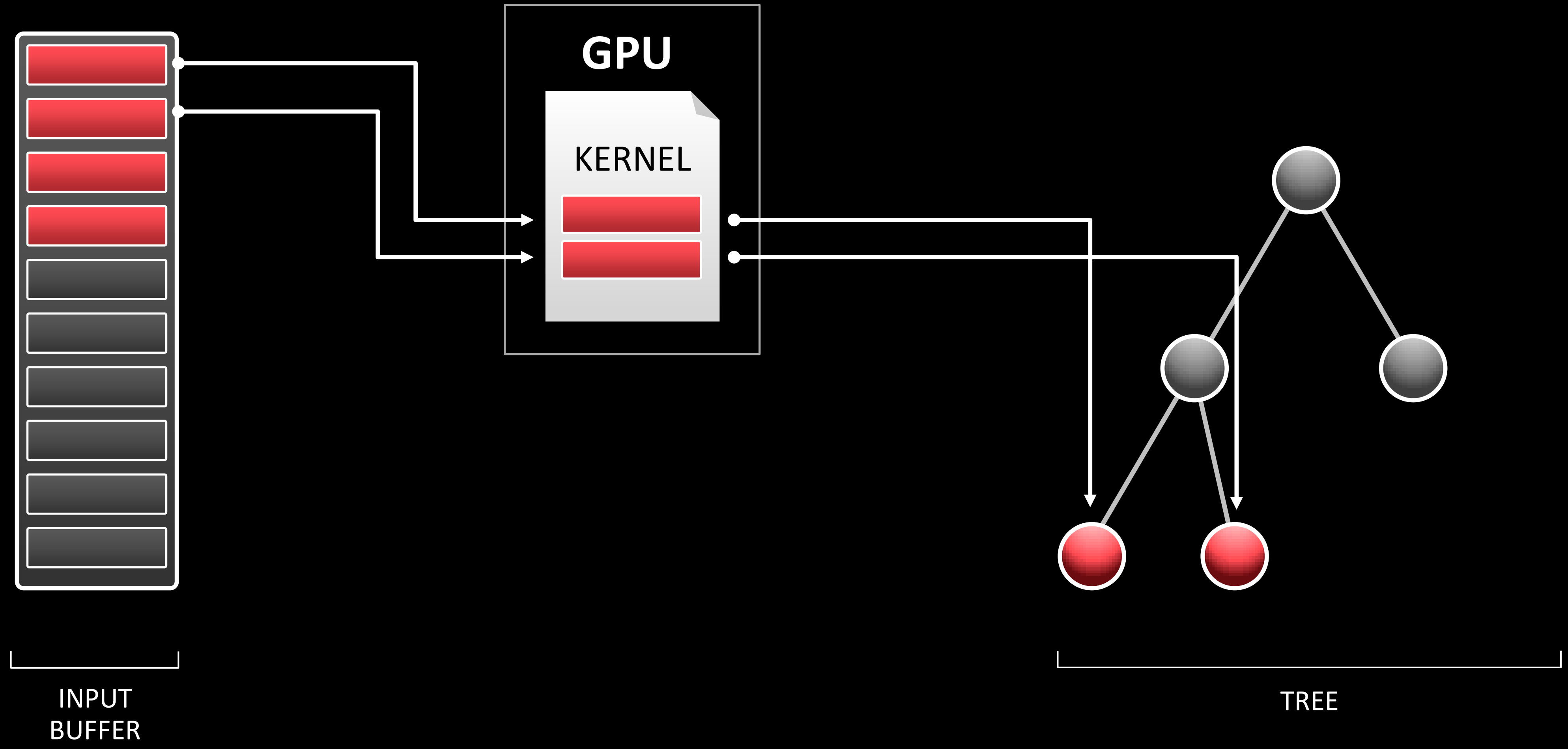
TREE

# PLATFORM ATOMICS

Legacy

Only GPU can work on input array

**Concurrent processing not possible**

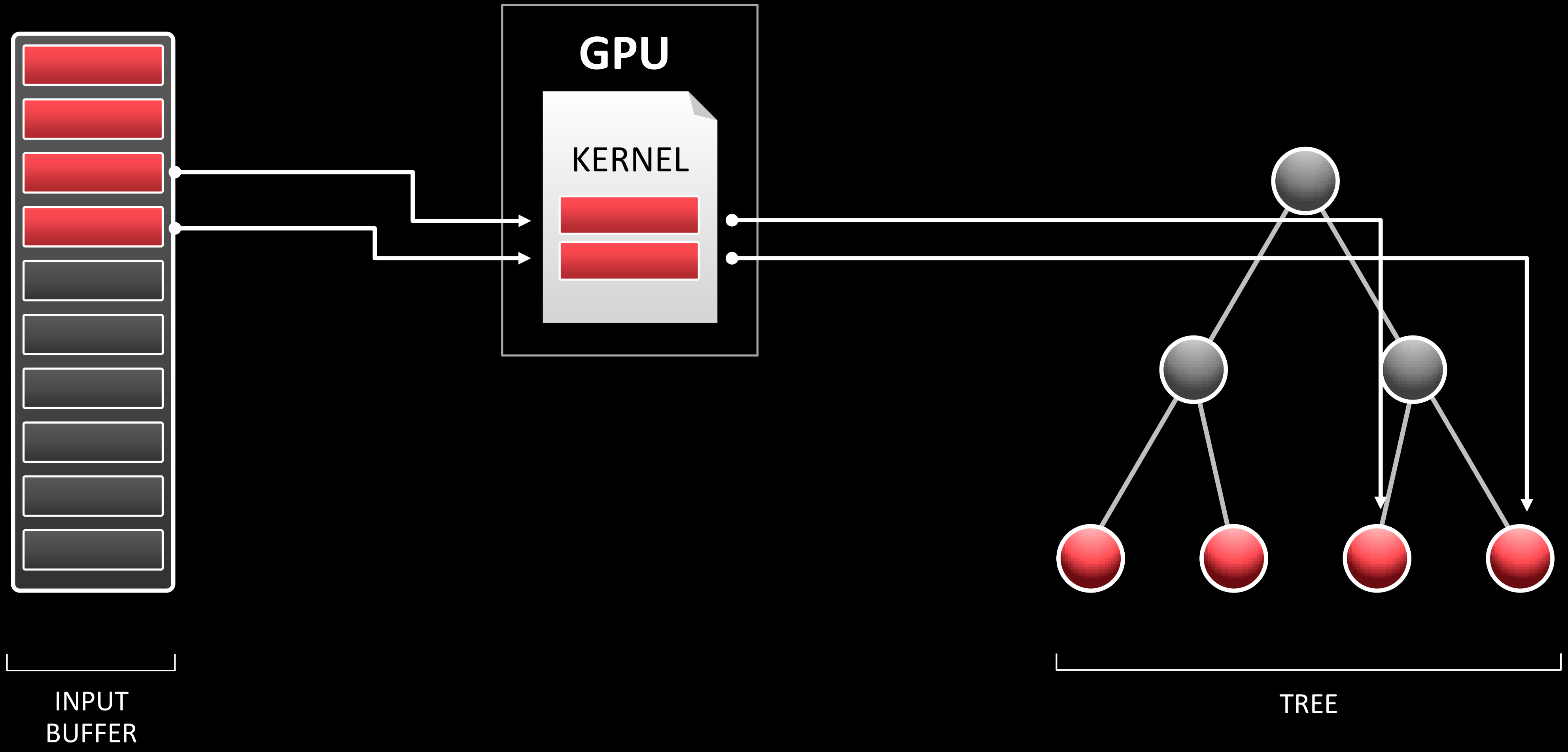


# PLATFORM ATOMICS

Legacy

Only GPU can work on input array

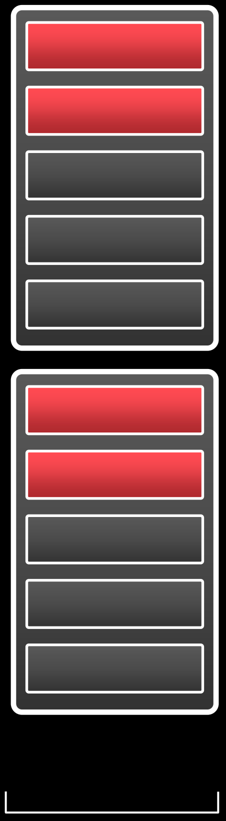
**Concurrent processing not possible**



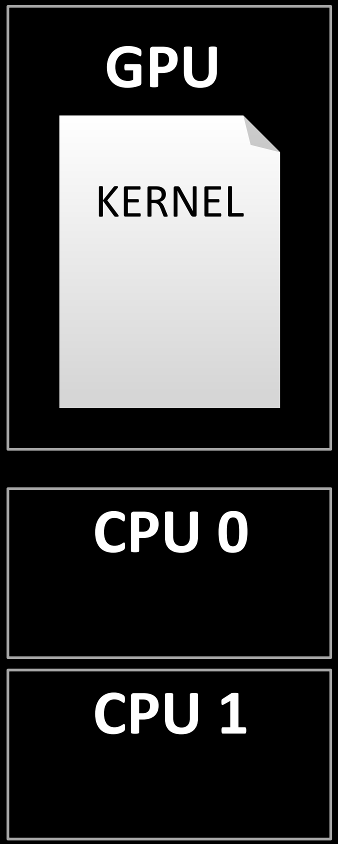
# PLATFORM ATOMICS

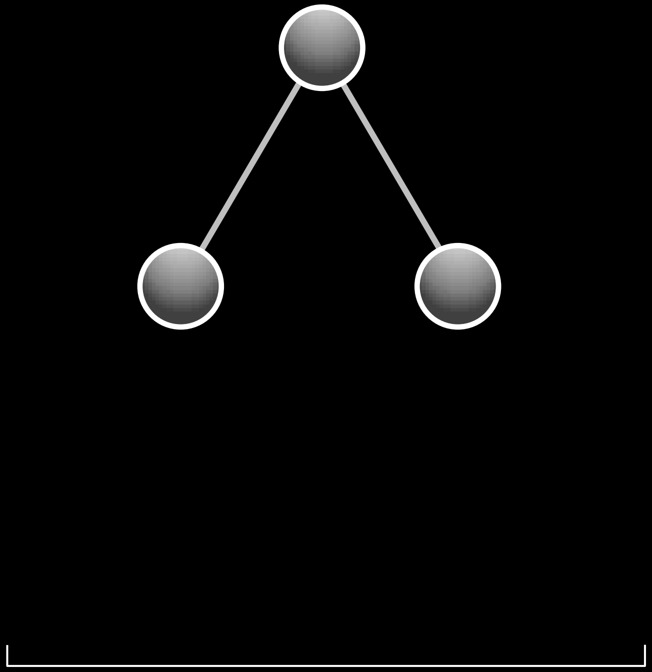
## HSA and full OpenCL 2.0

Both
CPU+GPU
operating on
same data
structure
**concurrently**



INPUT
BUFFER



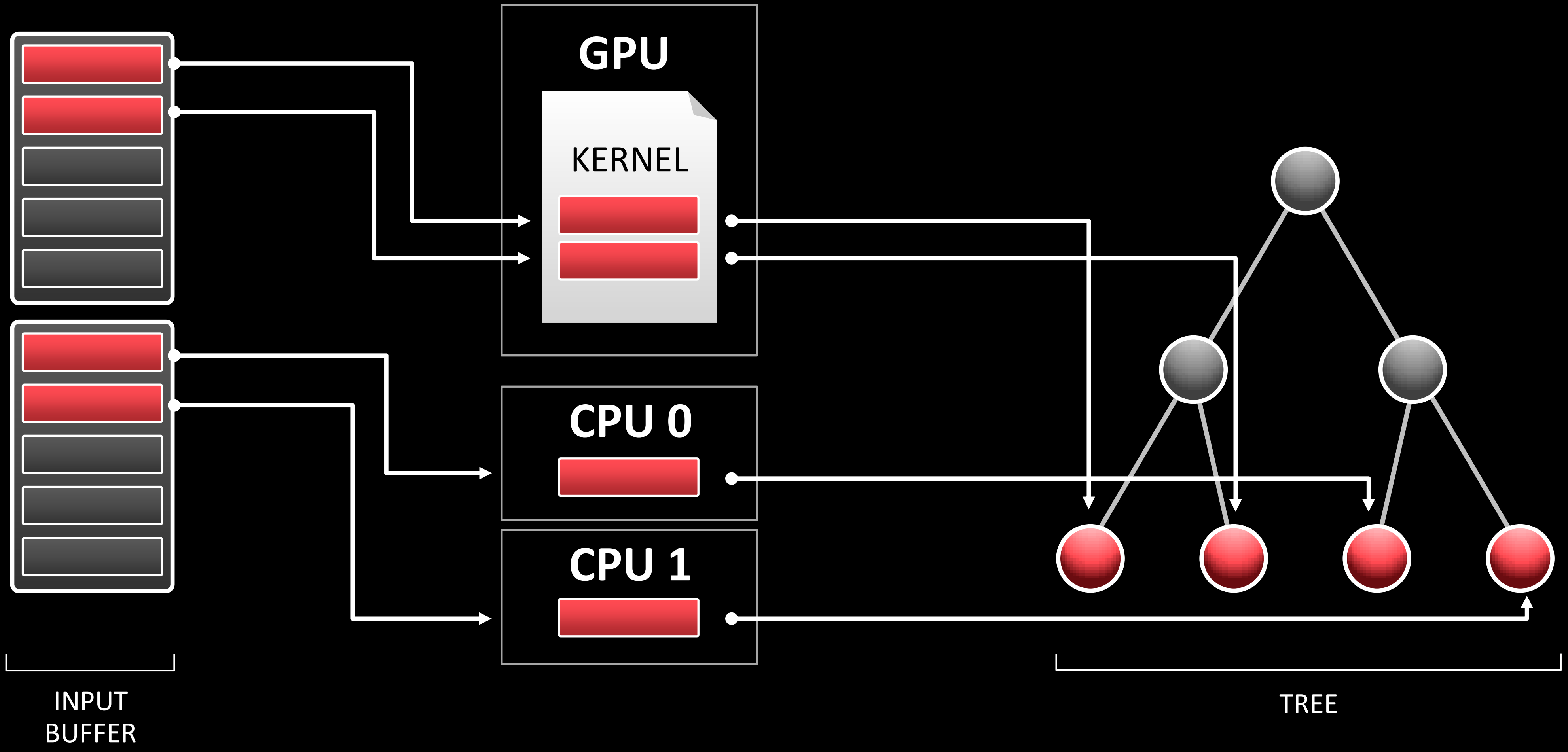



TREE

# PLATFORM ATOMICS

## HSA and full OpenCL 2.0

Both
CPU+GPU
operating on
same data
structure
**concurrently**



# Large
# Data Sets

# PROCESSING LARGE DATA SETS

AMD

**SYSTEM MEMORY**

**GPU**

The CPU creates a large data structure in System Memory. Computations using the data are offloaded to the GPU.

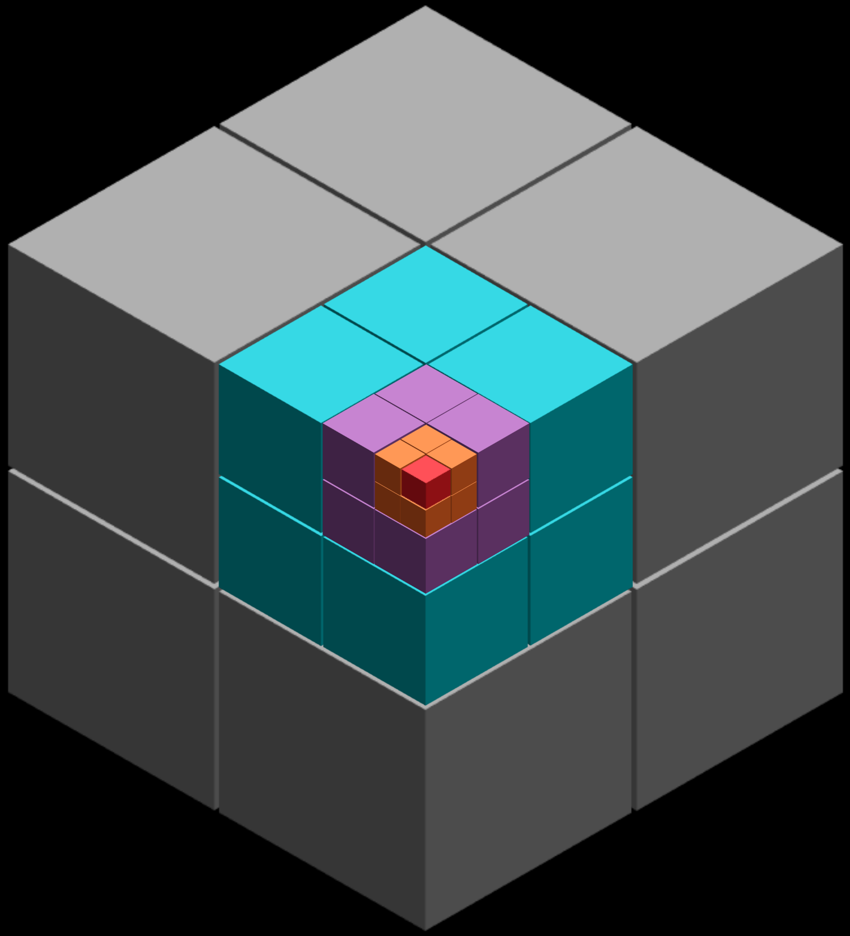
# PROCESSING LARGE DATA SETS

Level 1 Level 2 Level 3 Level 4 Level 5

AMD

**SYSTEM MEMORY**

Large 3D spatial data structure



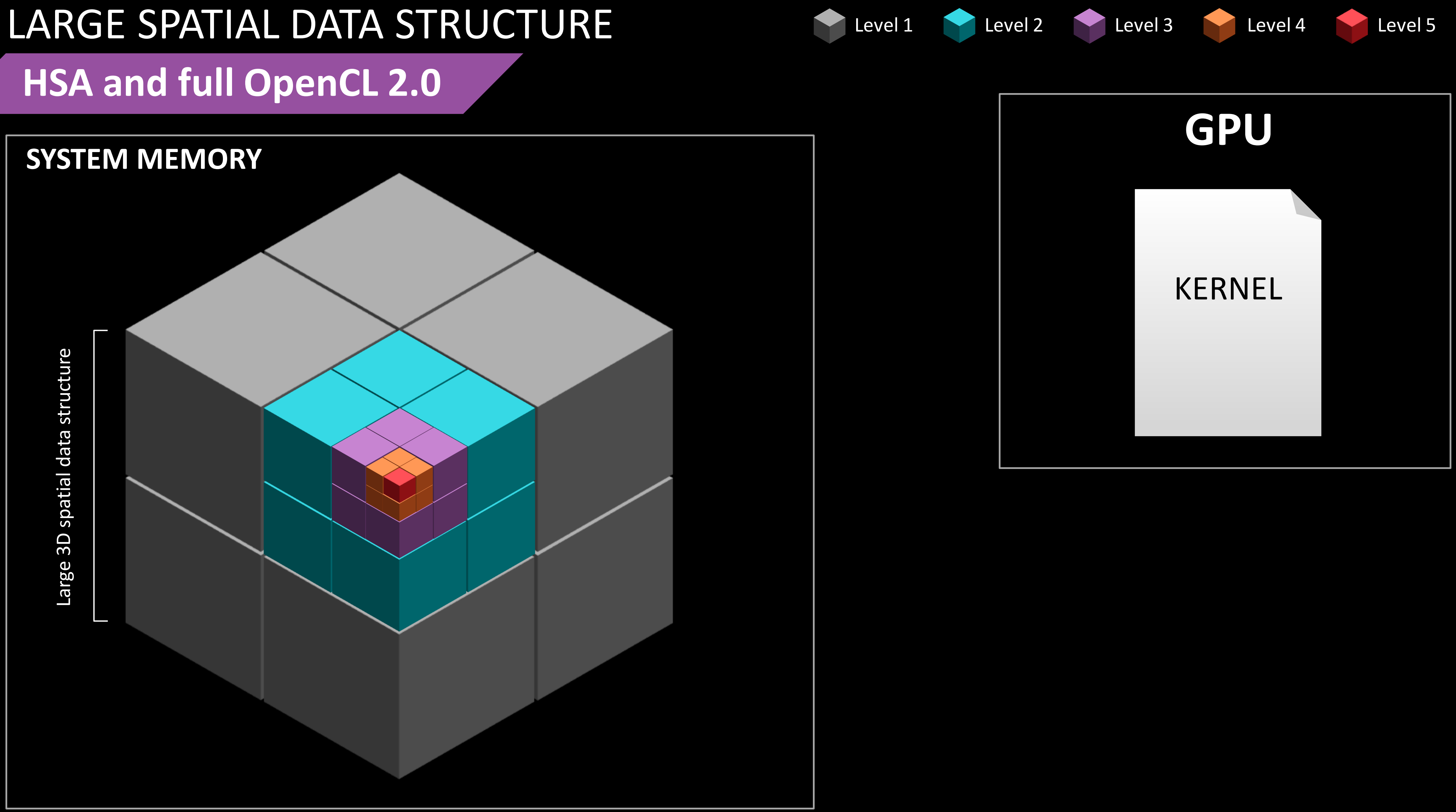
**GPU**

The CPU creates a large data structure in System Memory. Computations using the data are offloaded to the GPU.

Compare HSA and Legacy methods

# LARGE SPATIAL DATA STRUCTURE

## HSA and full OpenCL 2.0

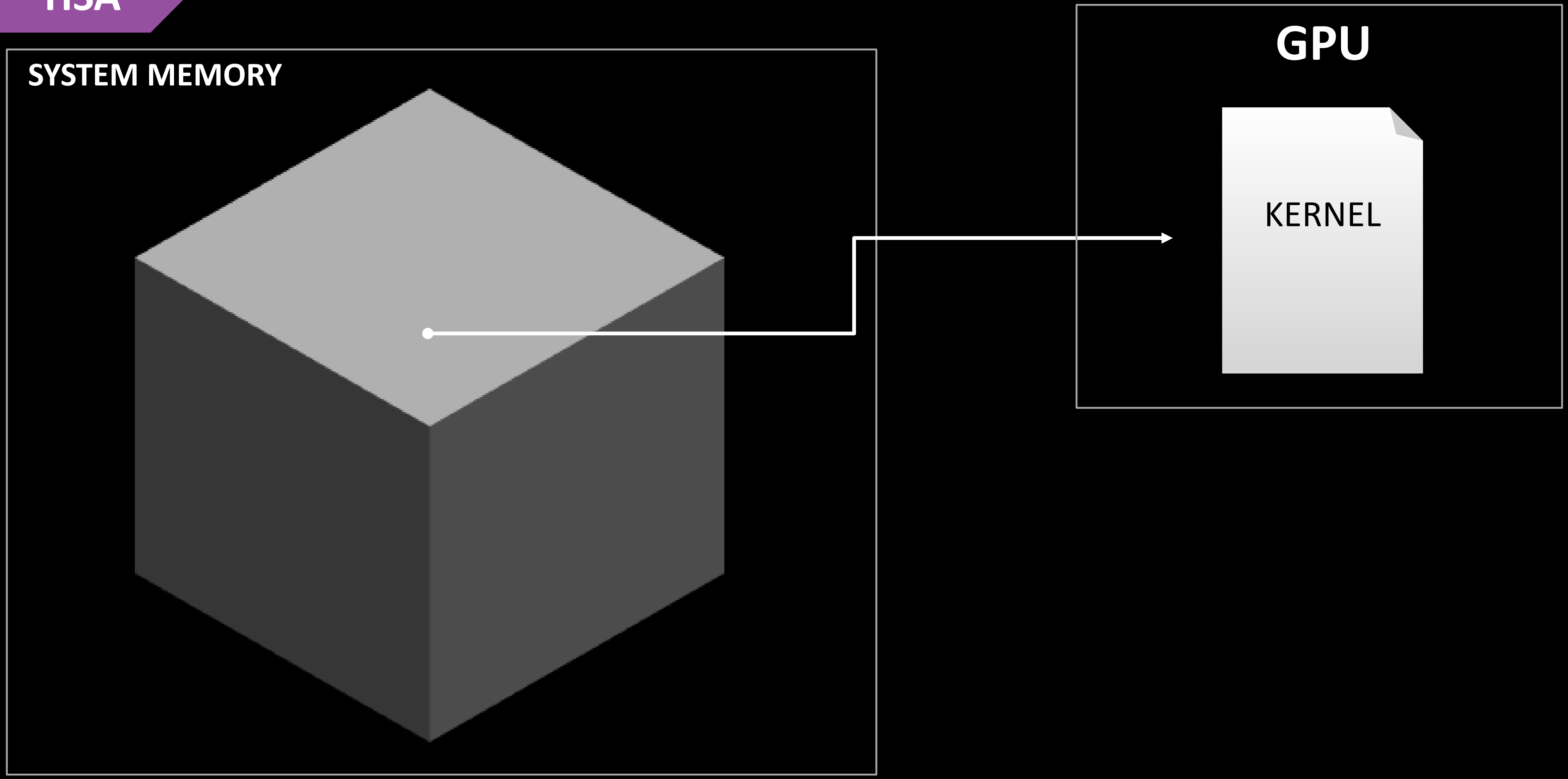


# GPU CAN TRAVERSE ENTIRE HIERARCHY

Level 1 Level 2 Level 3 Level 4 Level 5

AMD

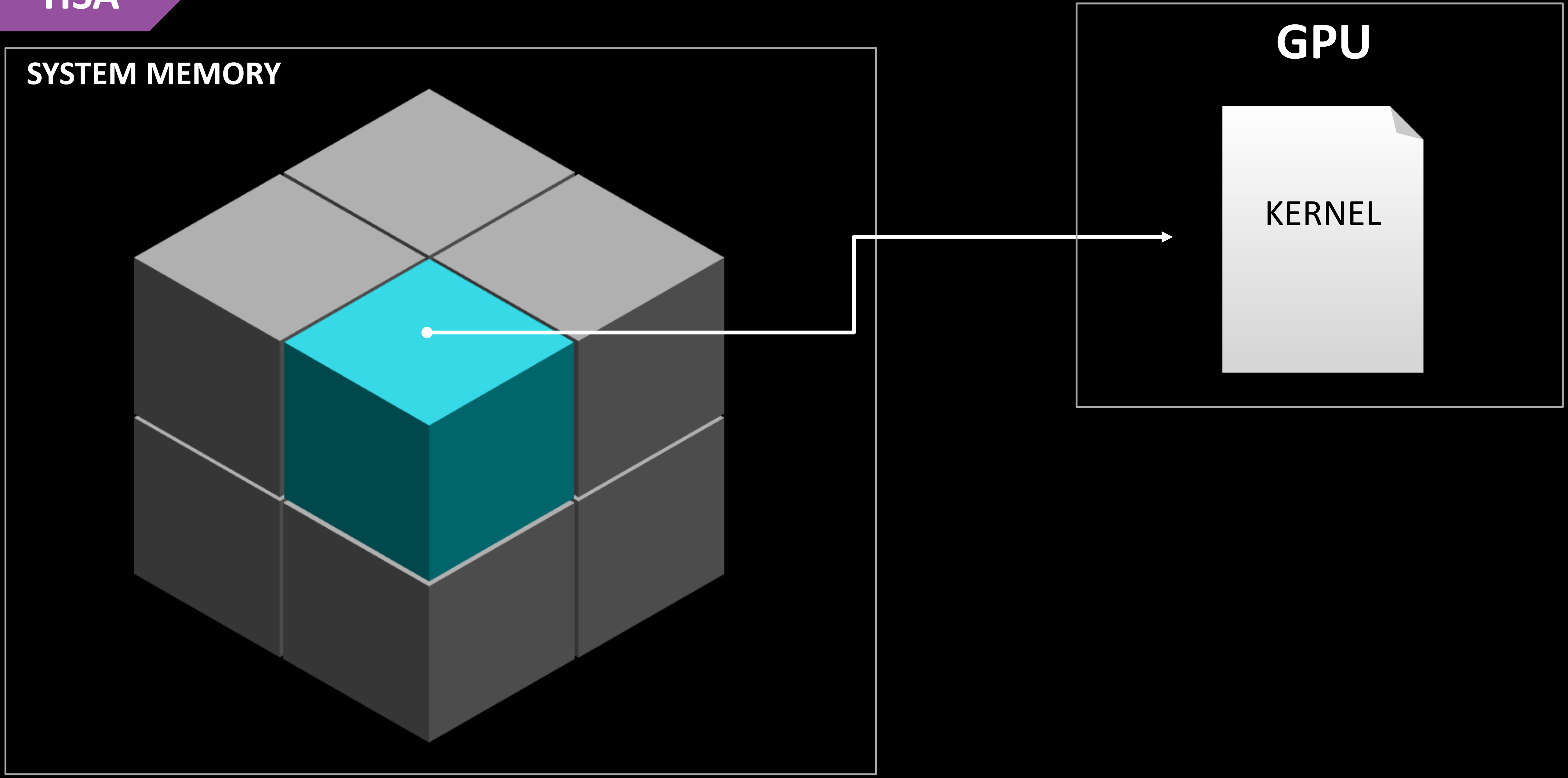
HSA



# GPU CAN TRAVERSE ENTIRE HIERARCHY

Level 1 Level 2 Level 3 Level 4 Level 5

HSA

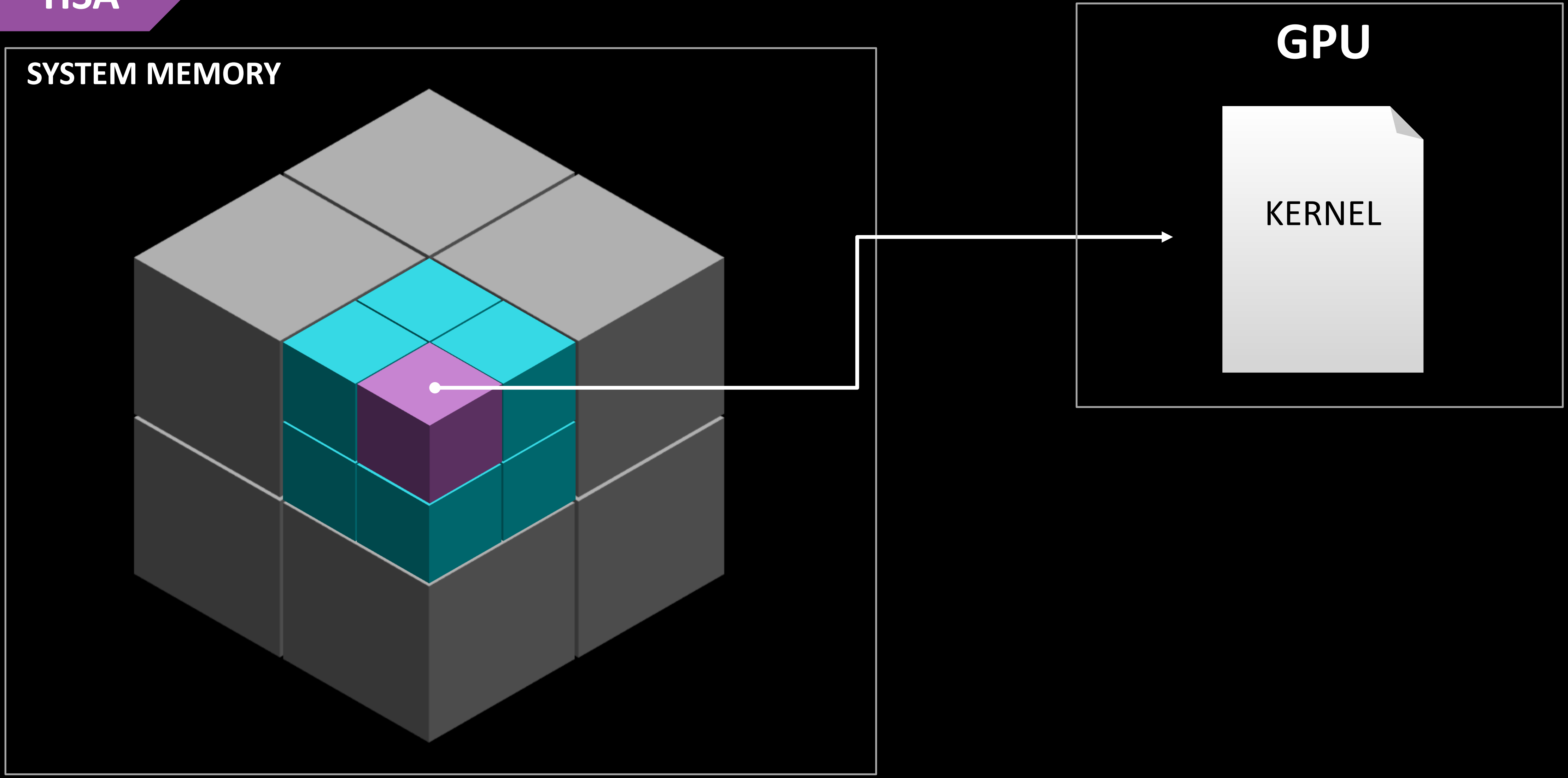


# GPU CAN TRAVERSE ENTIRE HIERARCHY

Level 1 Level 2 Level 3 Level 4 Level 5

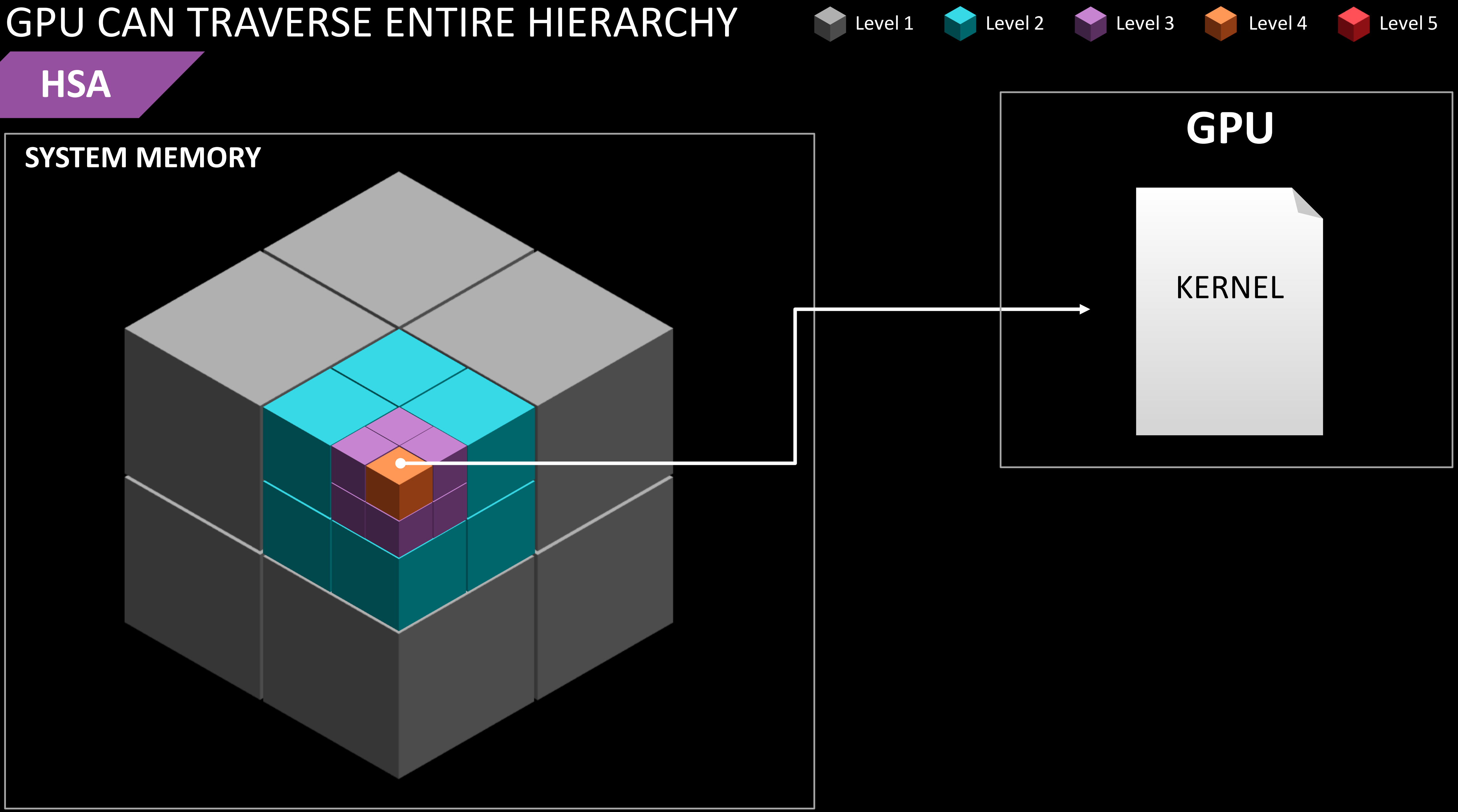
AMD

HSA



# GPU CAN TRAVERSE ENTIRE HIERARCHY

HSA

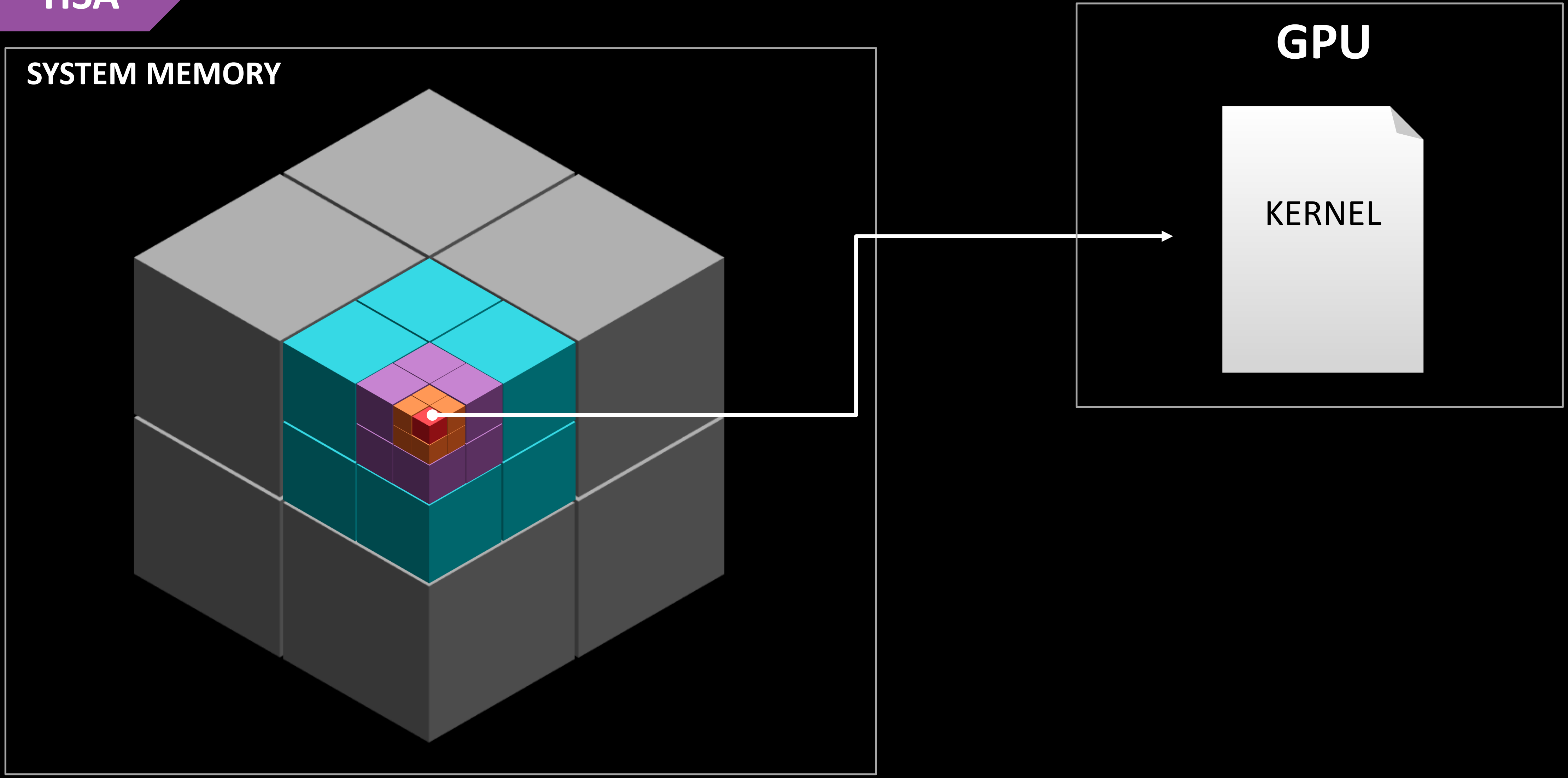


# GPU CAN TRAVERSE ENTIRE HIERARCHY

Level 1 Level 2 Level 3 Level 4 Level 5

AMD

**HSA**



# LEGACY ACCESS USING GPU MEMORY

Legacy

**SYSTEM MEMORY**

**GPU**

**GPU MEMORY**

GPU Memory is smaller

Have to copy and process in chunks

# LEGACY ACCESS TO LARGE STRUCTURES

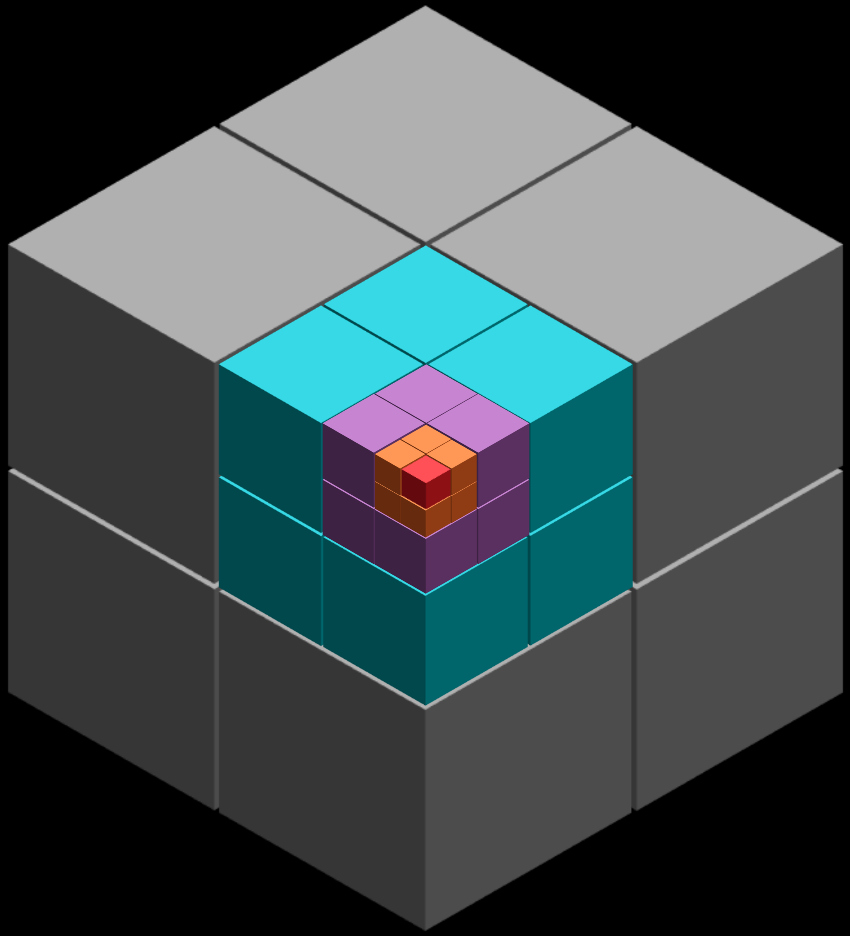
Level 1 Level 2 Level 3 Level 4 Level 5

AMD

Legacy

**SYSTEM MEMORY**

Large 3D spatial data structure



**GPU**

**GPU MEMORY**

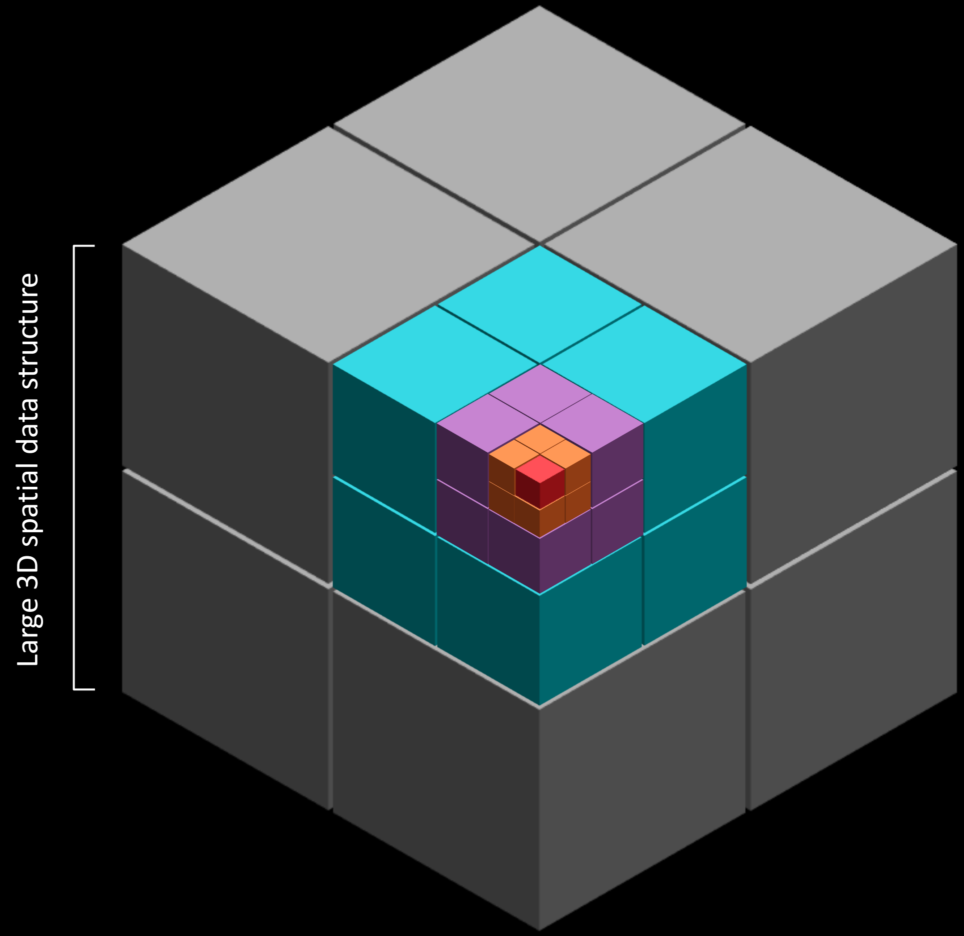
# COPY ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

AMD

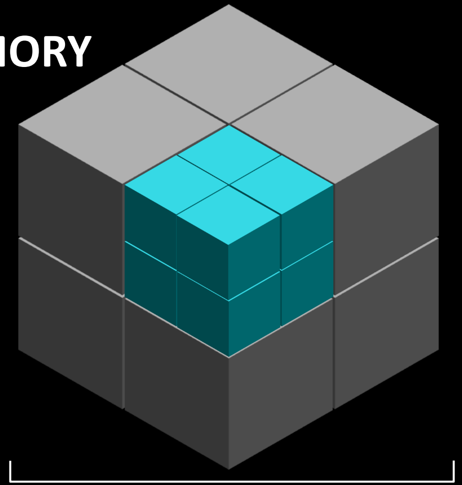
Legacy

## SYSTEM MEMORY




## GPU

## GPU MEMORY

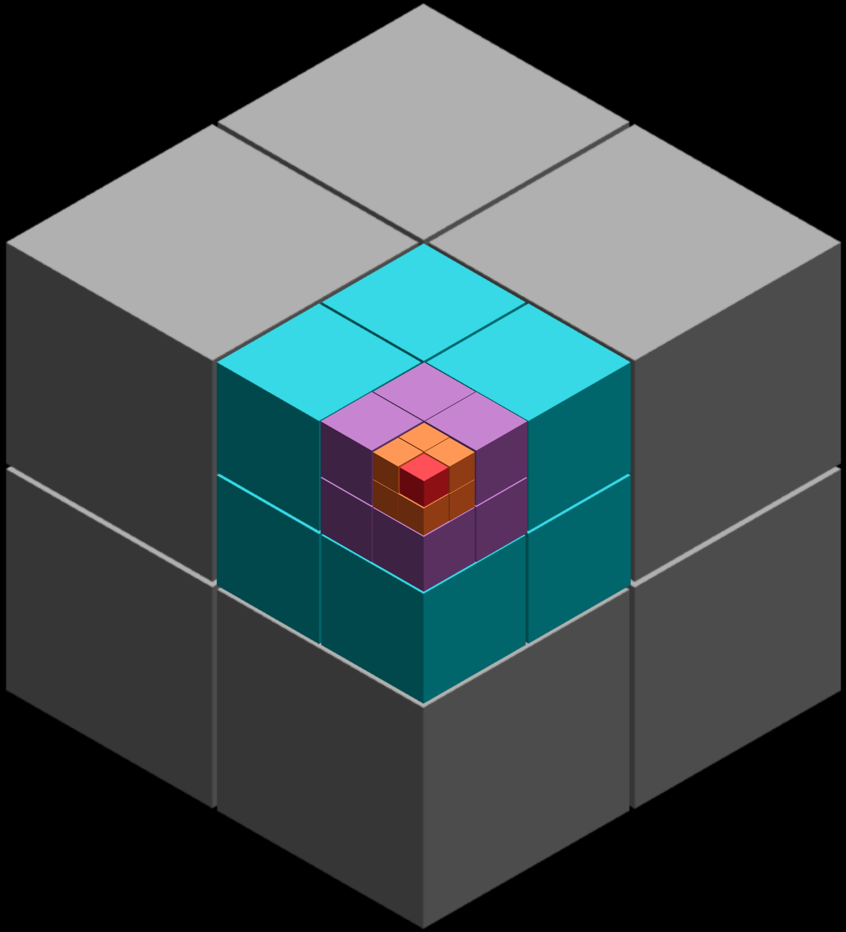


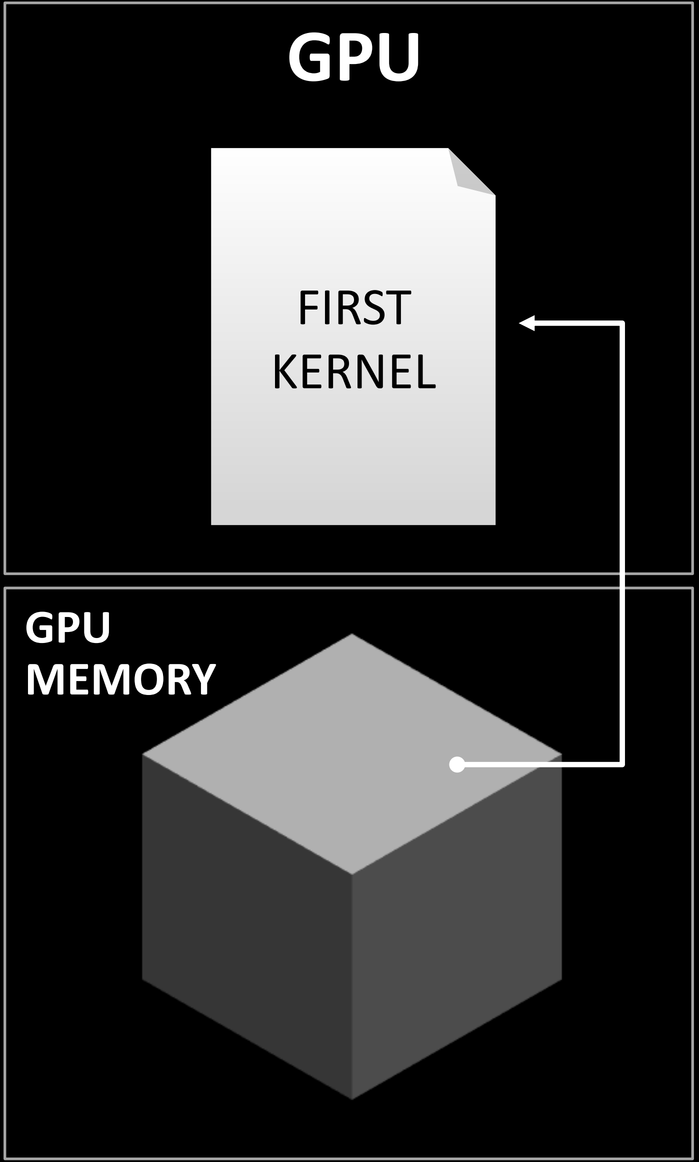
# PROCESS ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

Legacy

**SYSTEM MEMORY**

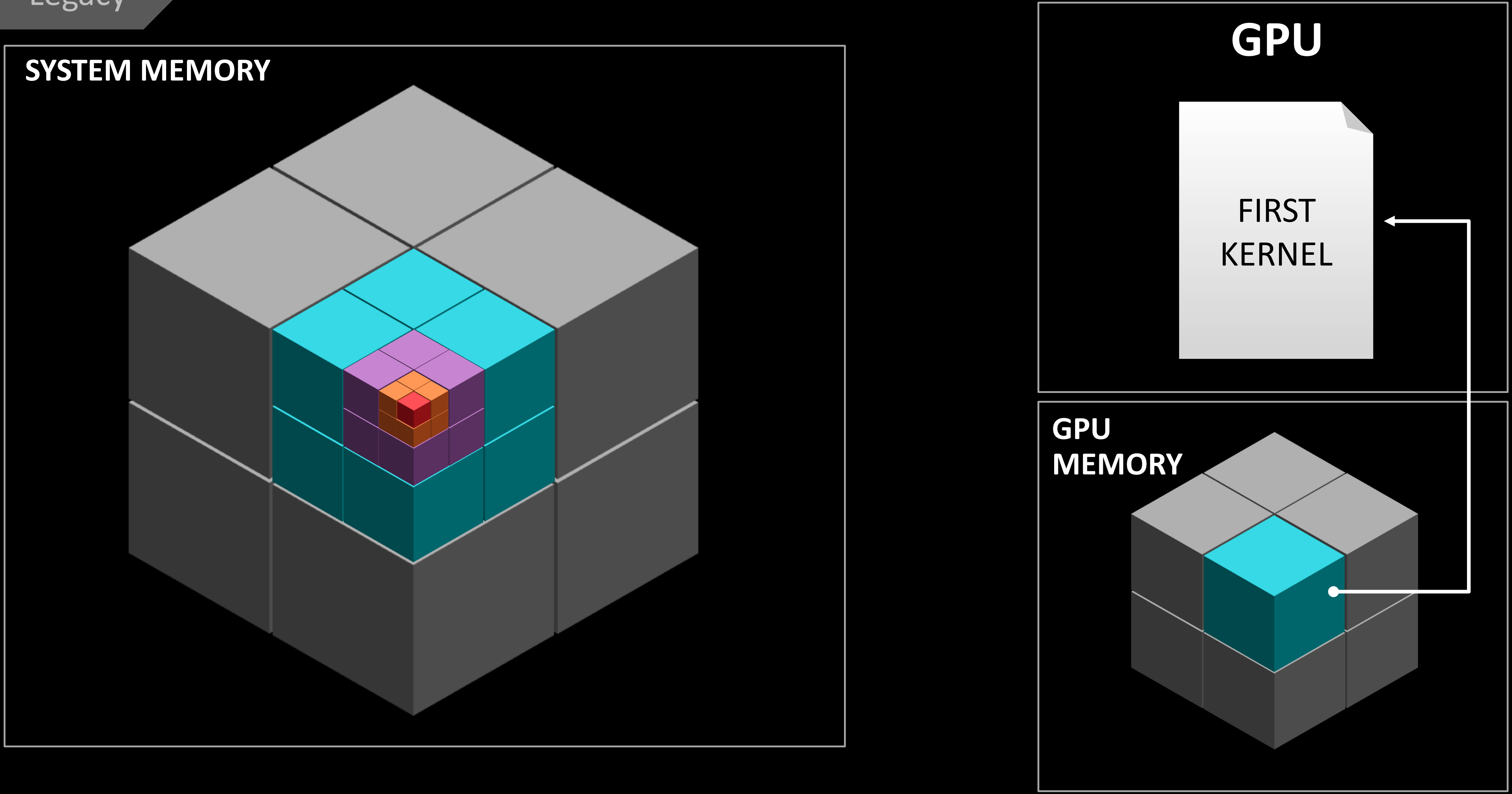




# PROCESS ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

Legacy



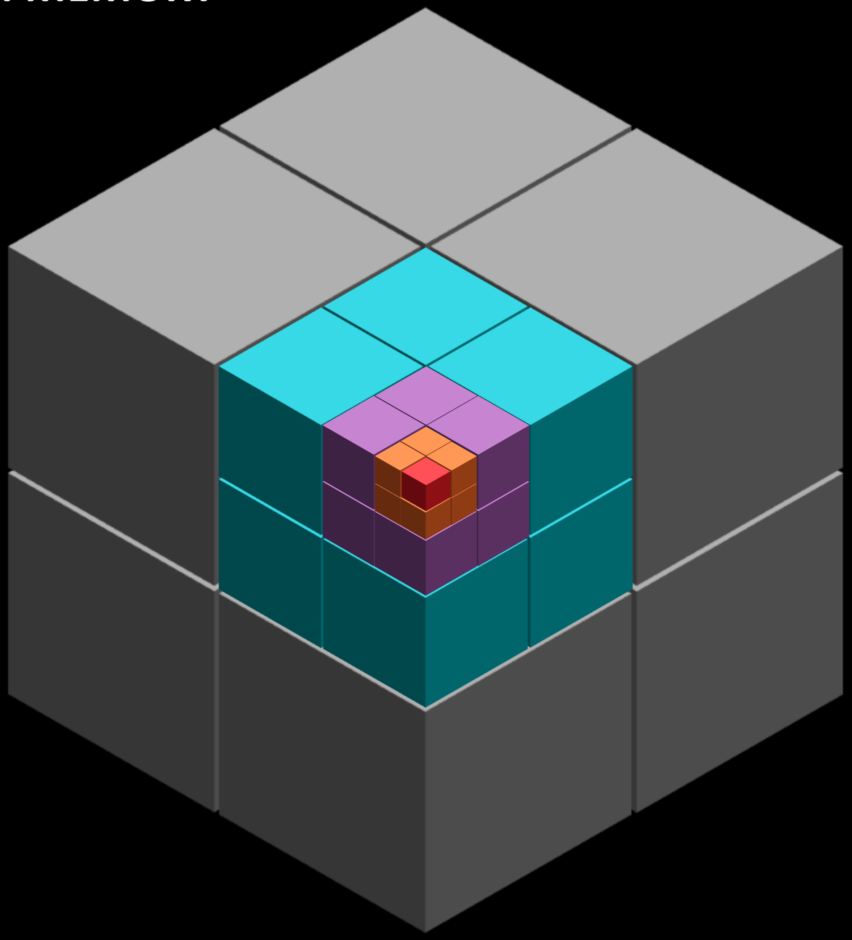
# PROCESS ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

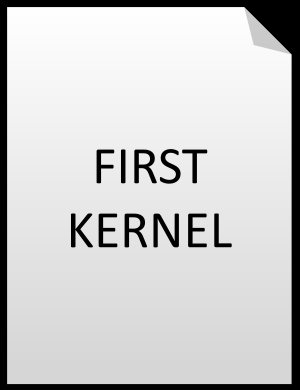
AMD

Legacy

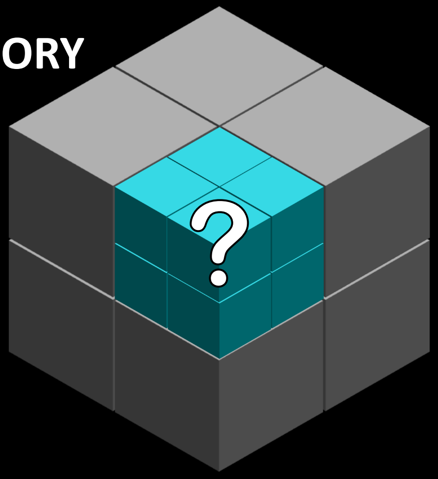
**SYSTEM MEMORY**



**GPU**




**GPU MEMORY**



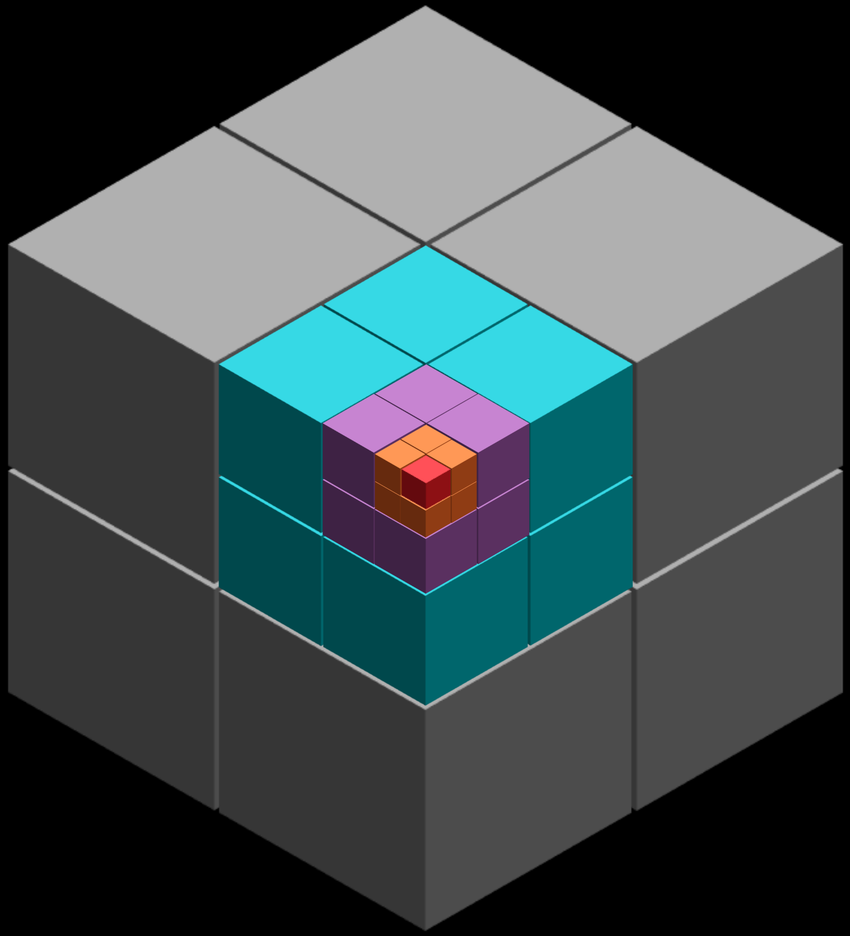
# COPY ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

AMD

Legacy

**SYSTEM MEMORY**

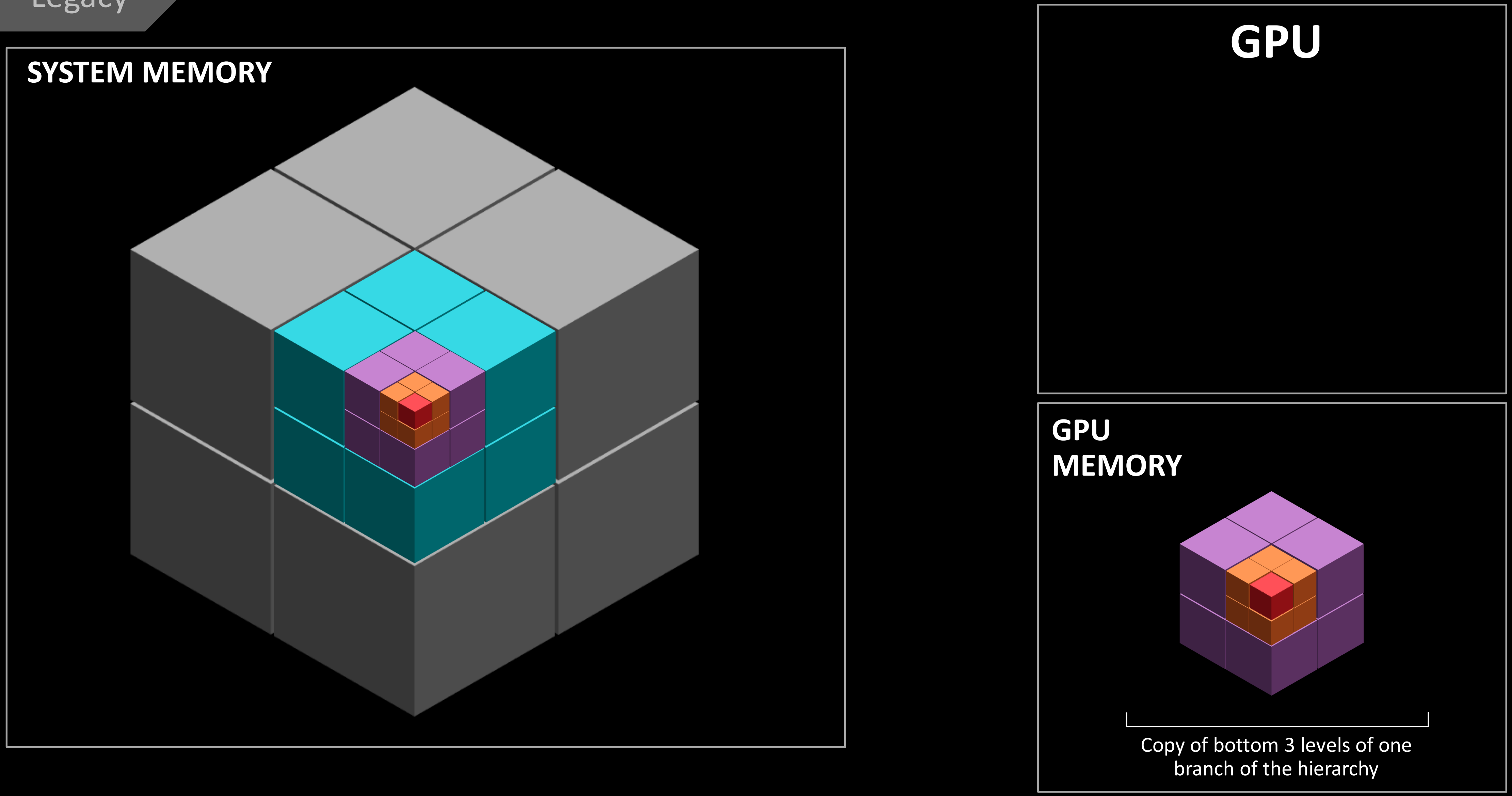


**GPU**

**GPU MEMORY**

# COPY ONE CHUNK AT A TIME


Legacy

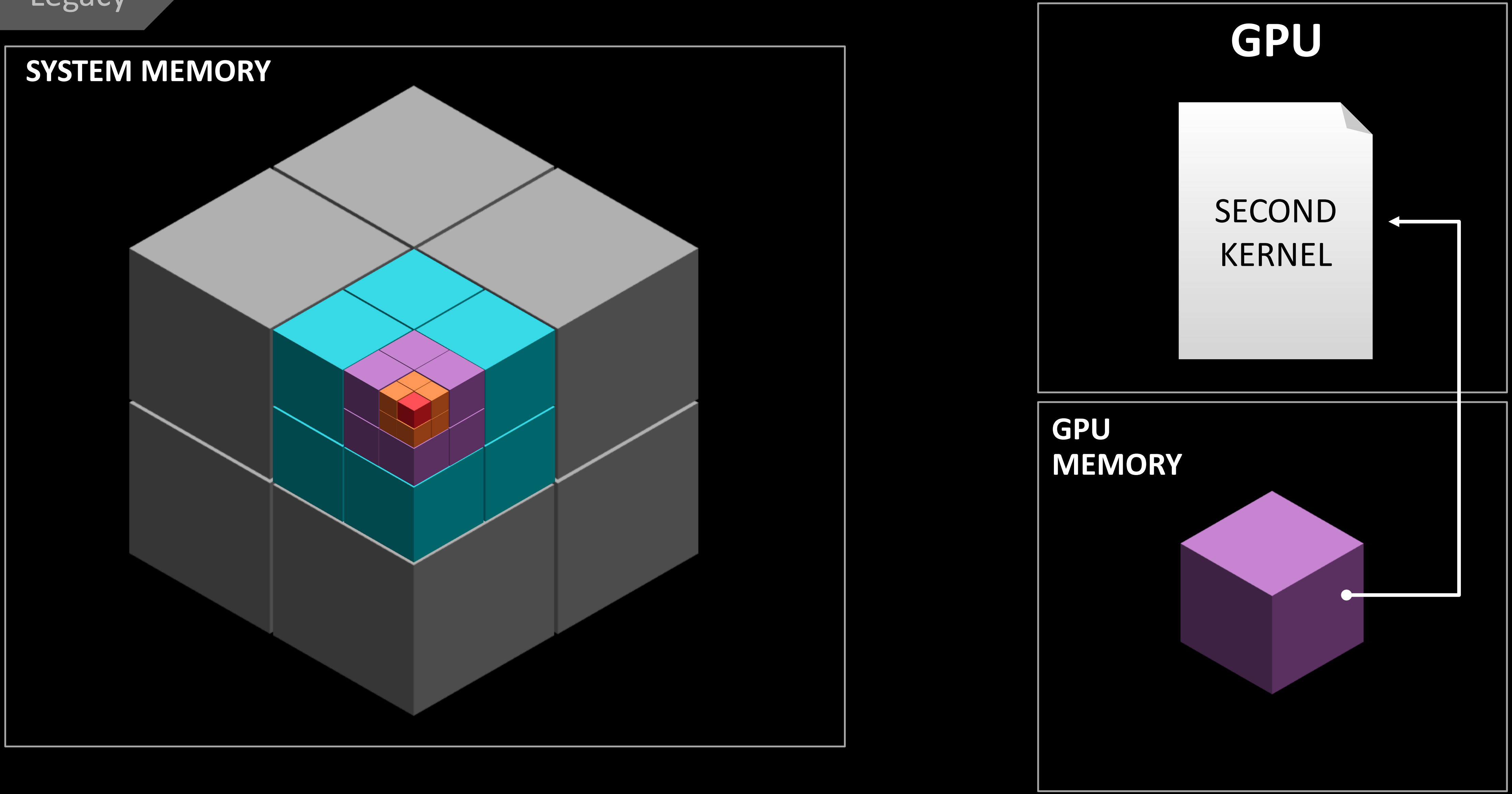


# PROCESS ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

AMD

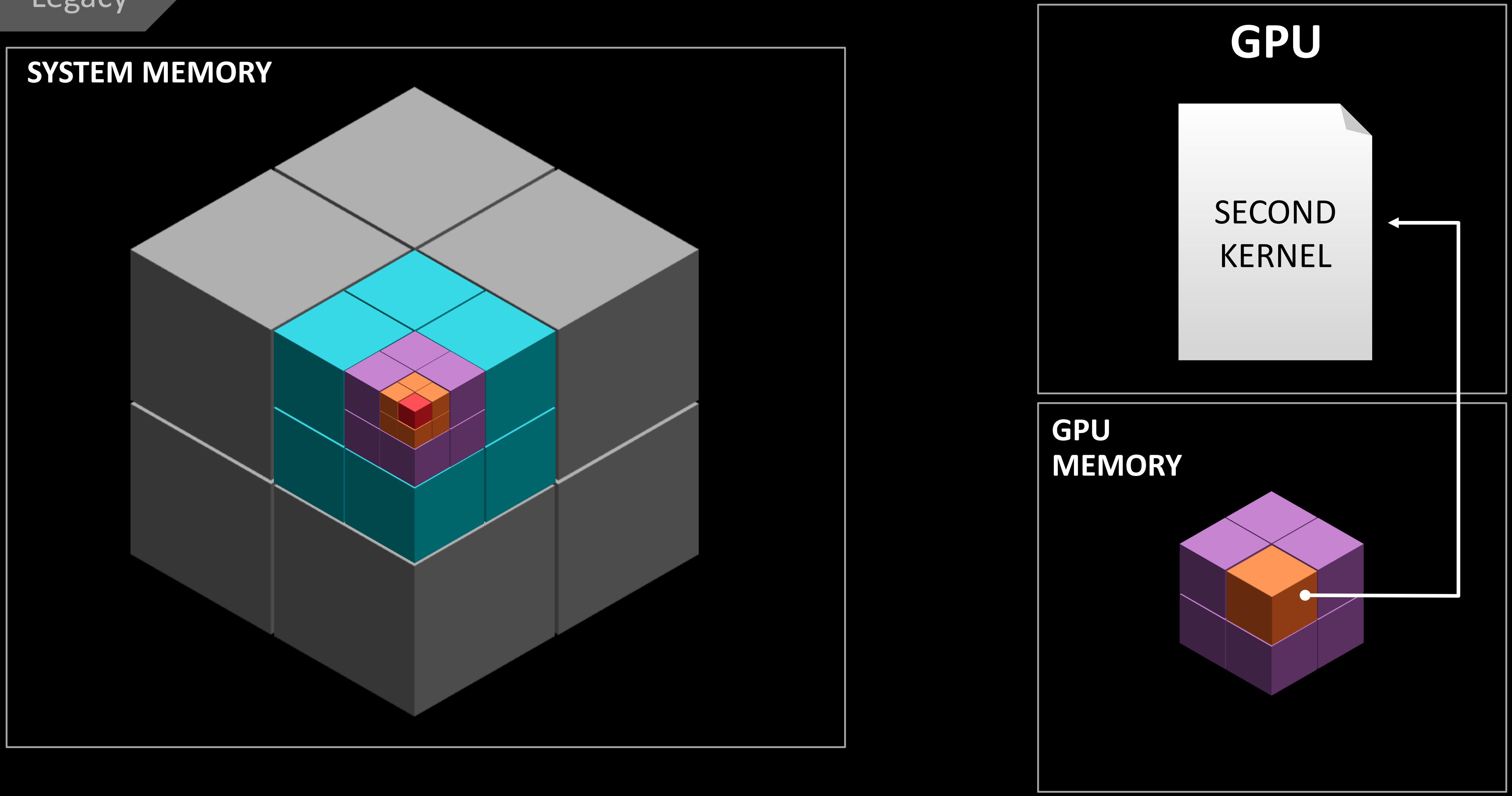
Legacy



# PROCESS ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

Legacy

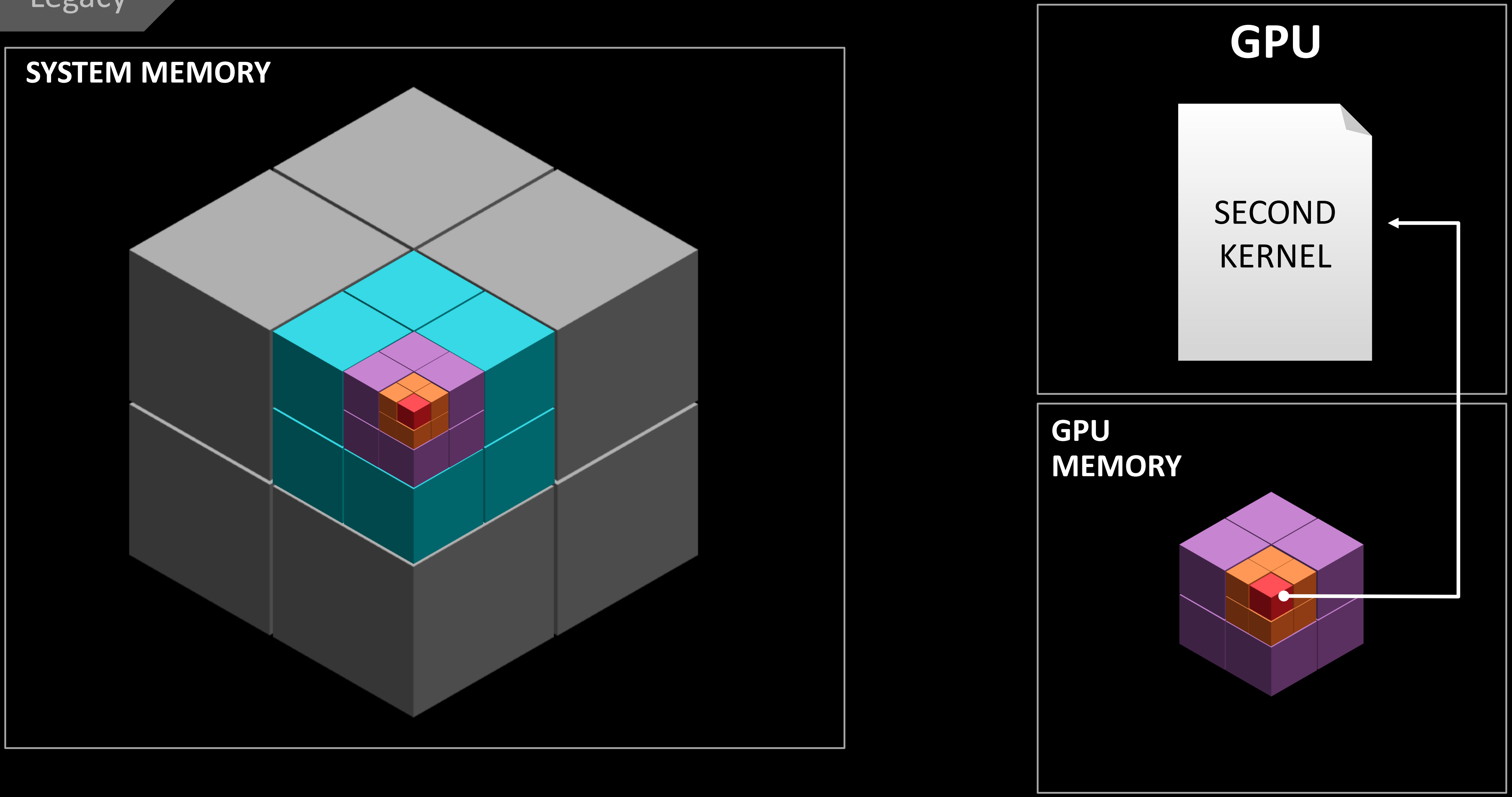


# PROCESS ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

AMD

Legacy



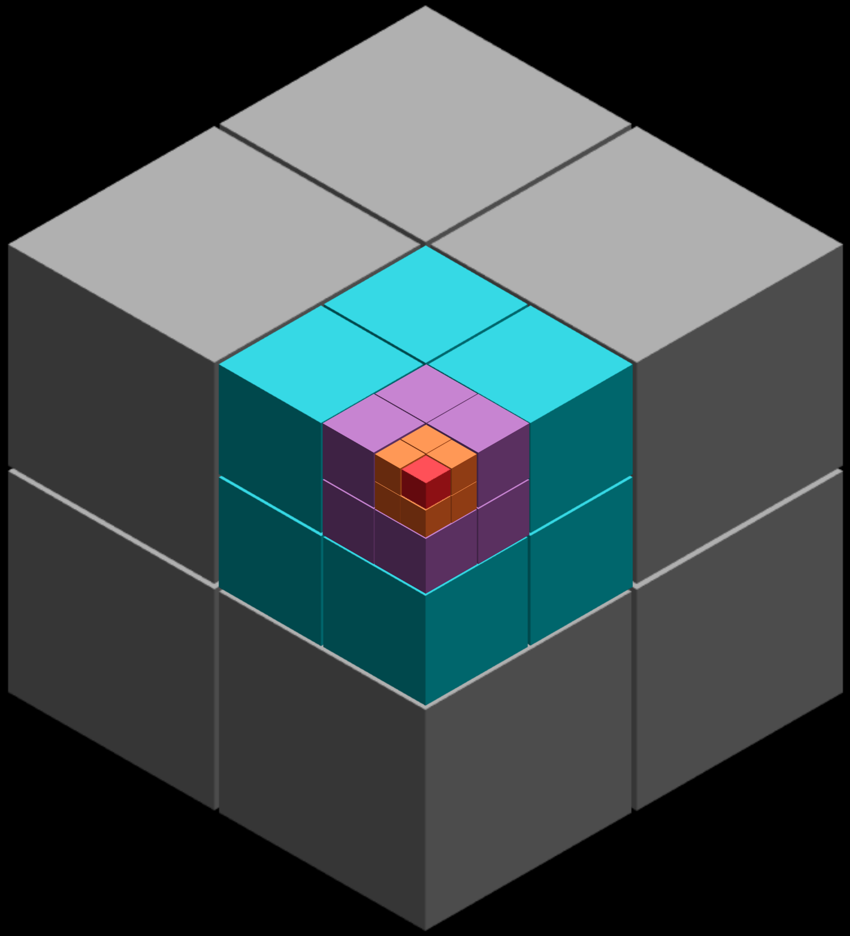
# COPY ONE CHUNK AT A TIME

Level 1 | Level 2 | Level 3 | Level 4 | Level 5

AMD

Legacy

**SYSTEM MEMORY**

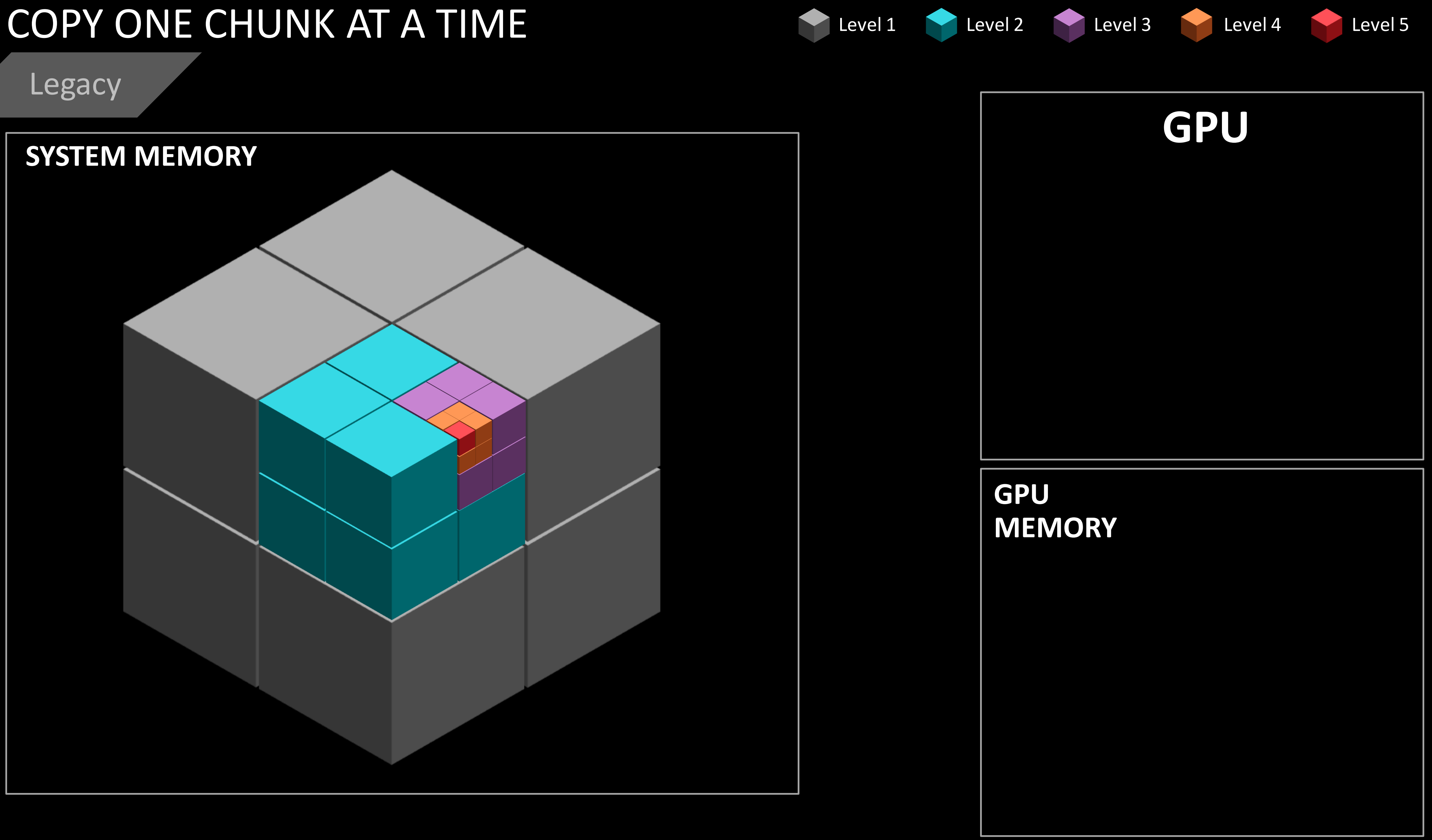


**GPU**

**GPU MEMORY**

# COPY ONE CHUNK AT A TIME

Legacy

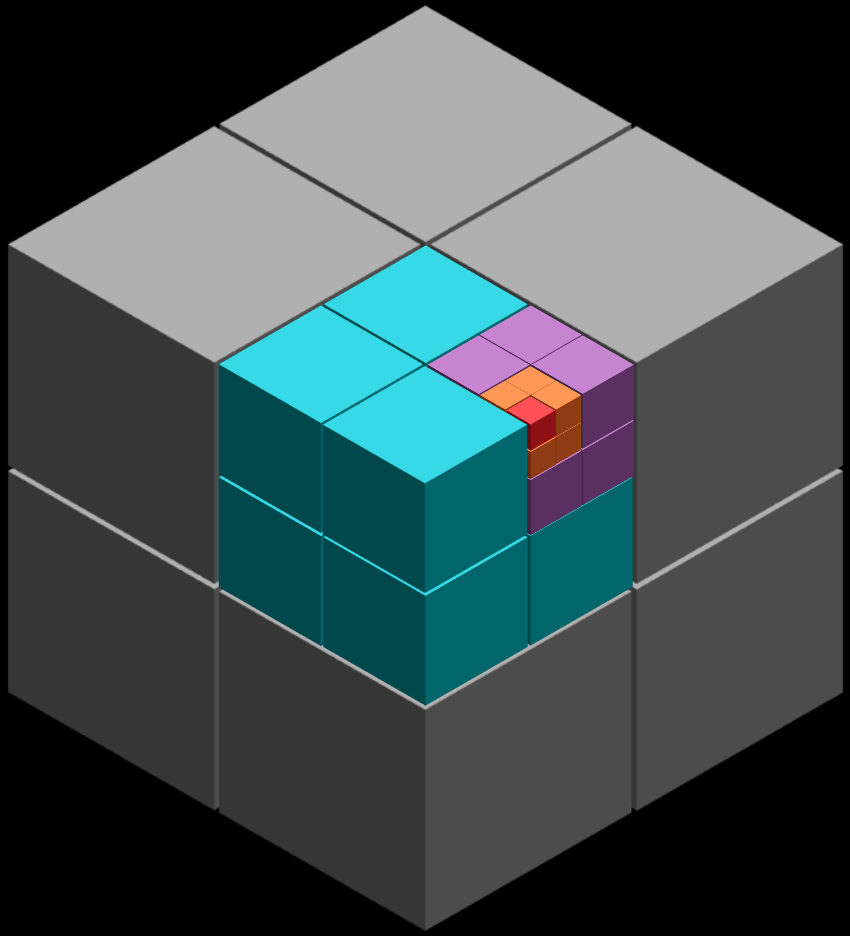


# COPY ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

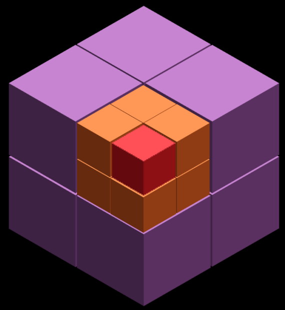
Legacy

SYSTEM MEMORY



GPU

GPU MEMORY



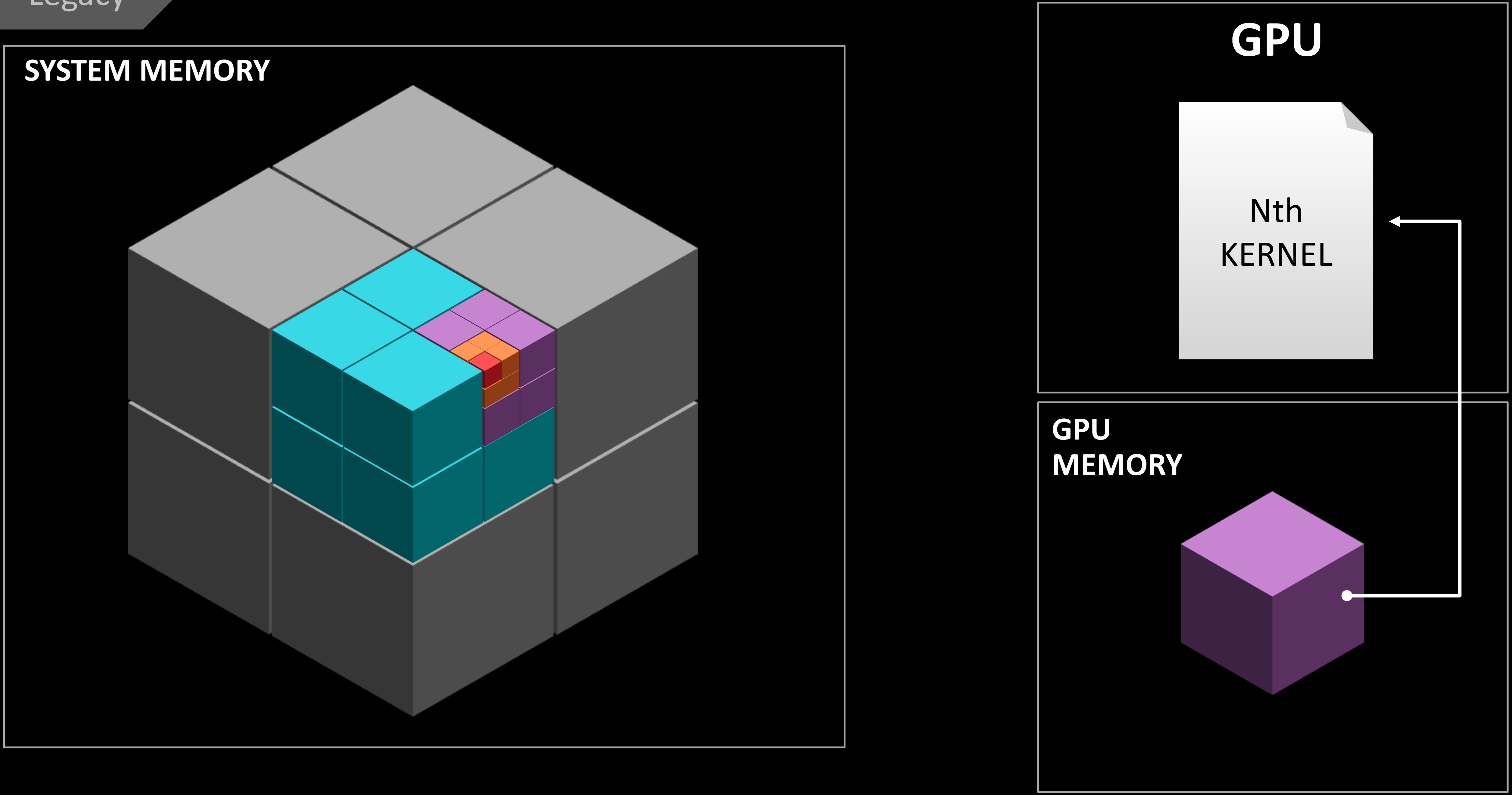
Copy of bottom 3 levels of a different branch of the hierarchy

# PROCESS ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

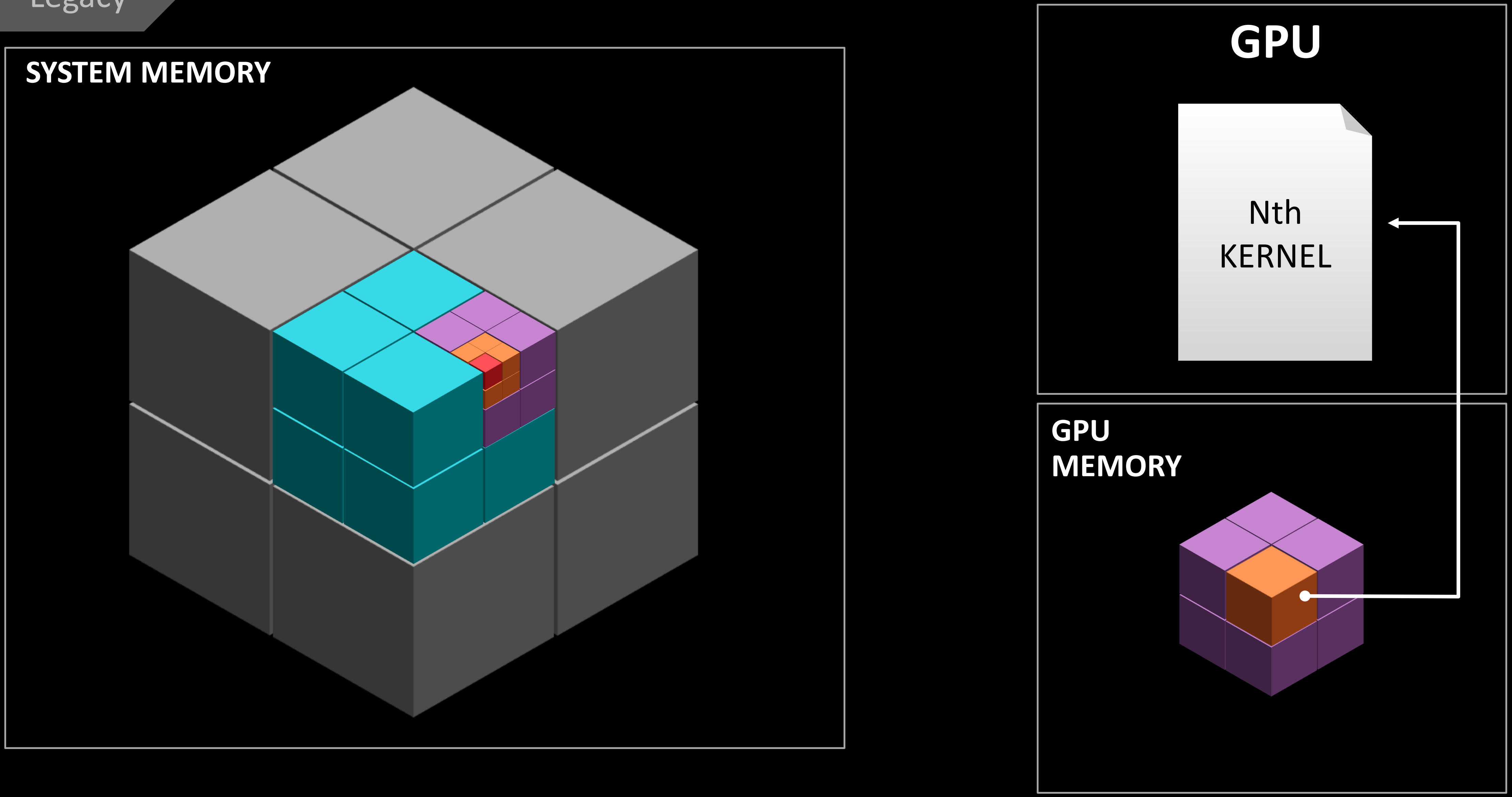
AMD

Legacy



# PROCESS ONE CHUNK AT A TIME


Legacy

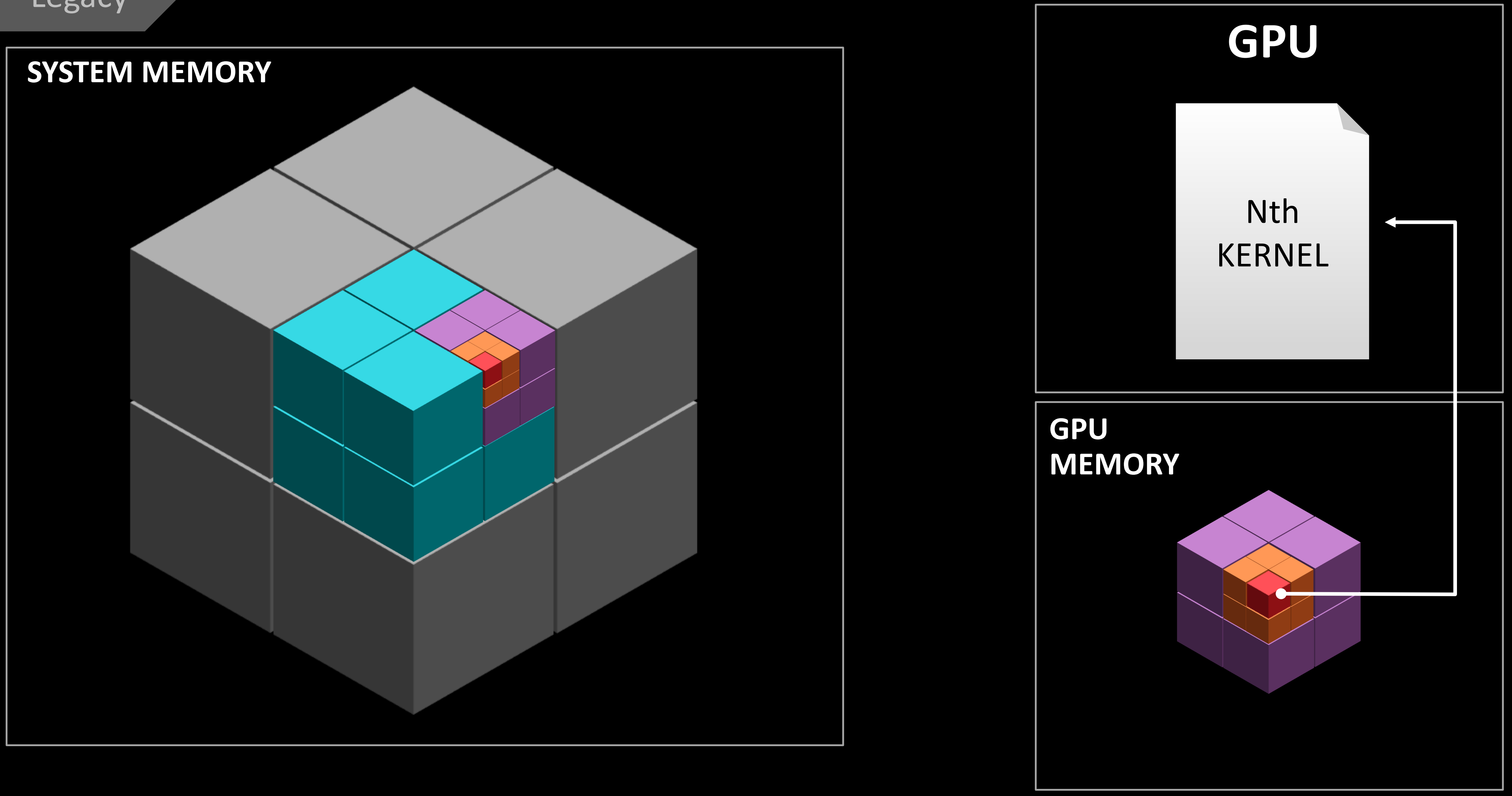


# PROCESS ONE CHUNK AT A TIME

Level 1 Level 2 Level 3 Level 4 Level 5

AMD

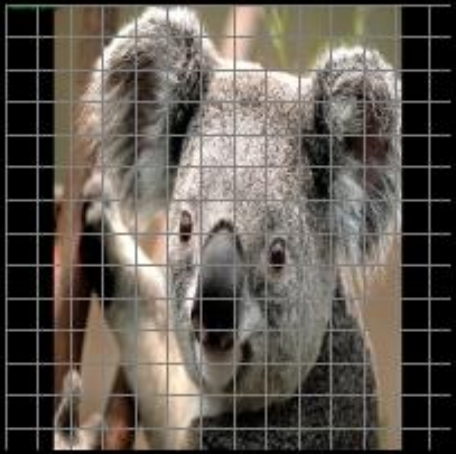
Legacy



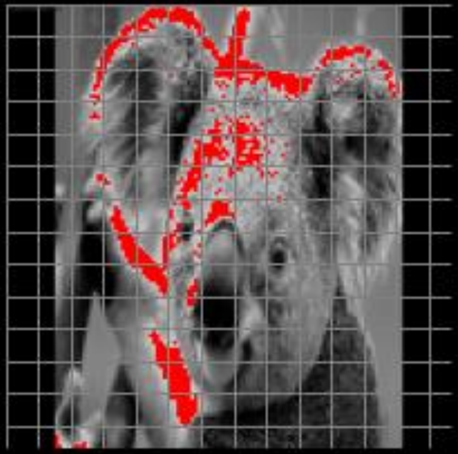
# Callbacks

# CALLBACKS
## COMMON SITUATION IN HC

- Parallel processing algorithm with branches
- A seldom taken branch requires new data from the CPU
- On legacy systems, the algorithm must be split:
  - Process Kernel 1 on GPU
  - Check for CPU callbacks and if any, process on CPU
  - Process Kernel 2 on GPU



Input Image

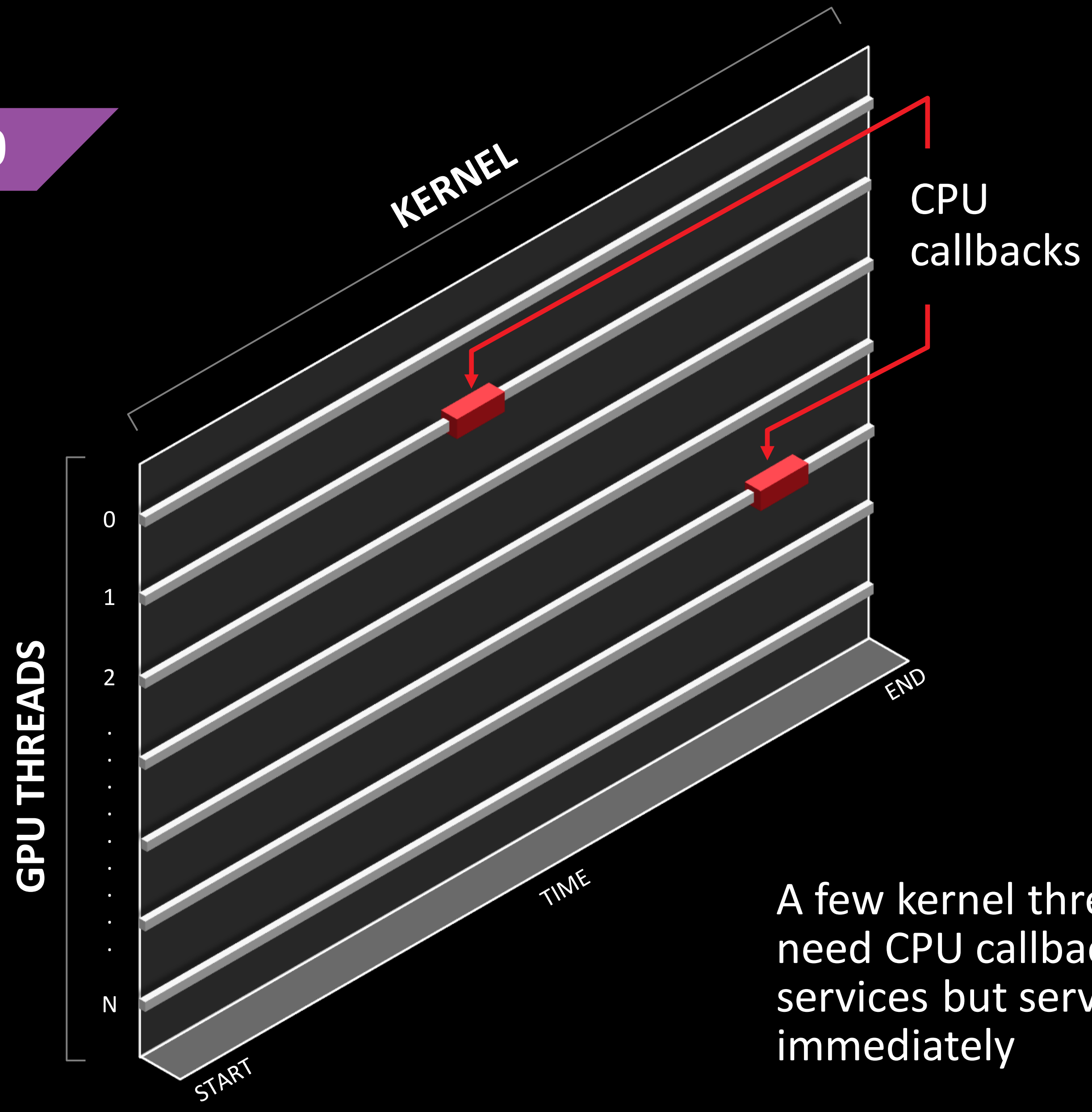


Output Image

- Example algorithm from Image Processing
  - Perform a filter
  - Calculate average LUMA in each tile
  - Compare LUMA against threshold and call CPU callback if exceeded (rare)
  - Perform special processing on tiles with callbackx\s

# CALLBACKS

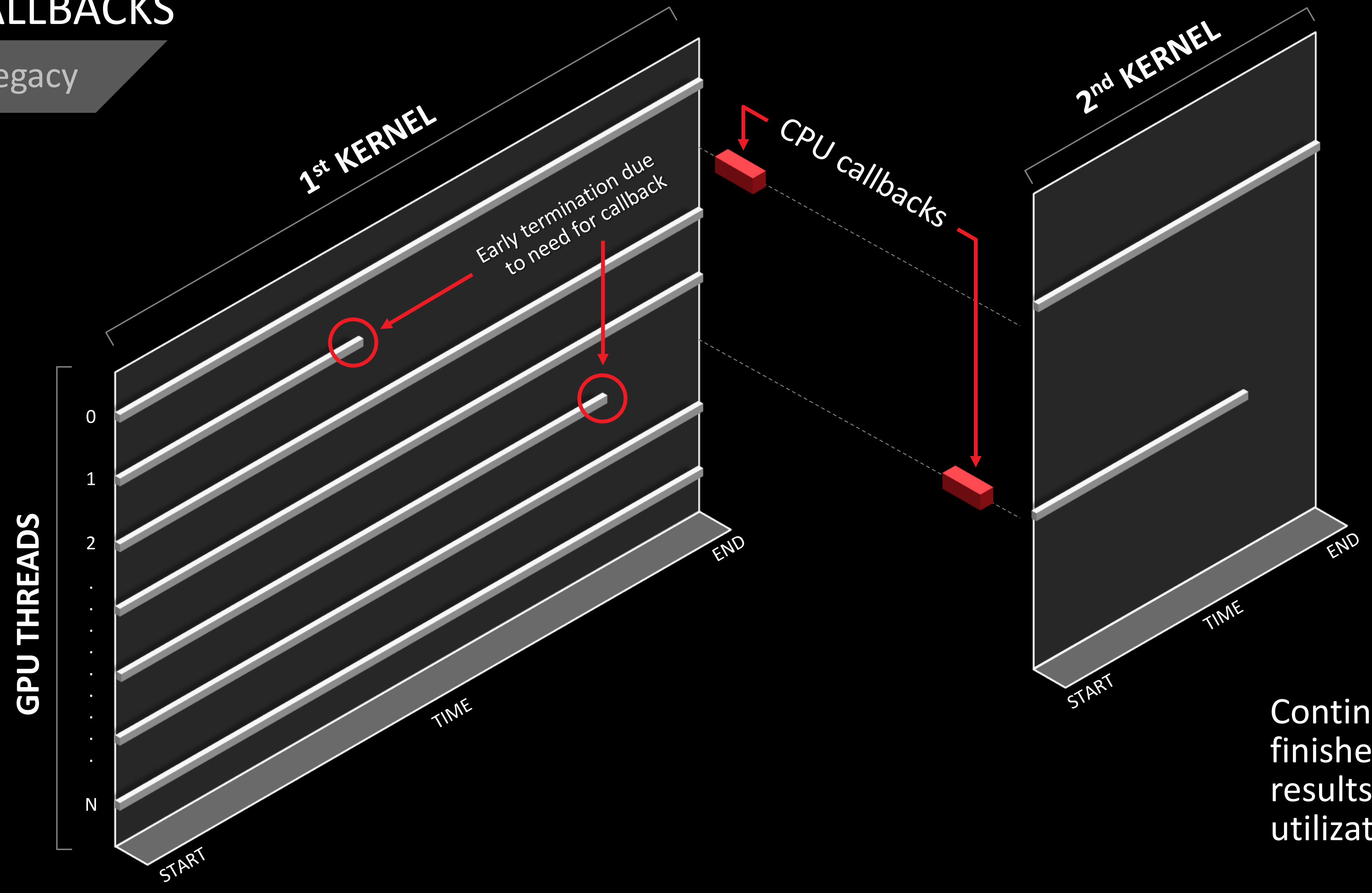
## HSA and full OpenCL 2.0




A few kernel threads need CPU callback services but serviced immediately

# CALLBACKS

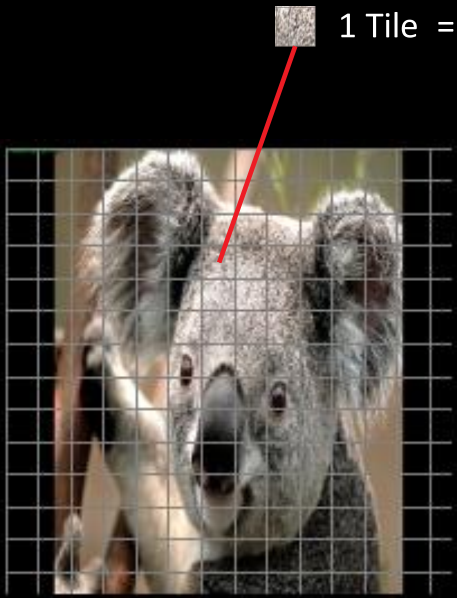
Legacy



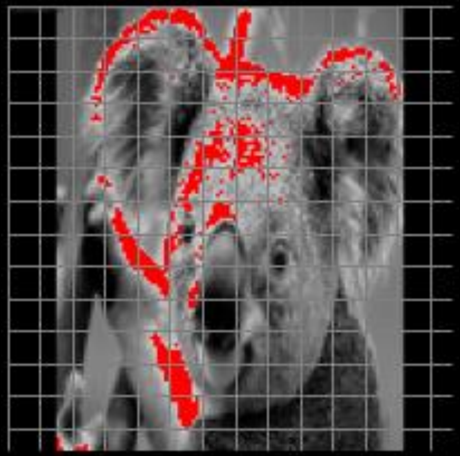

Continuation kernel finishes up kernel works results in poor GPU utilization

**GPU to CPU callbacks use Shared Virtual Memory (SVM) Semaphores, implemented using Platform Atomic Compare-and-Swap.**

1 Tile = 1 OpenCL Work Item



Input Image



Output Image

## GPU

- Work items compute average RGB value of all the pixels in a tile
- Work items also compute average Luma from the average RGB
- If average Luma > threshold, workgroup invokes CPU CALLBACK
- In parallel with callback, compute data used to saturate LUMA

## CPU

- For selected tiles, update average Luma value (set to RED)

## GPU

- Work items apply the Luma value to all pixels in the tile

# AMD HETEROGENEOUS COMPUTING SOLUTIONS OVERVIEW

DEVELOPER TOOLS

PROGRAMMING LANGUAGES

OPTIMIZED LIBRARIES

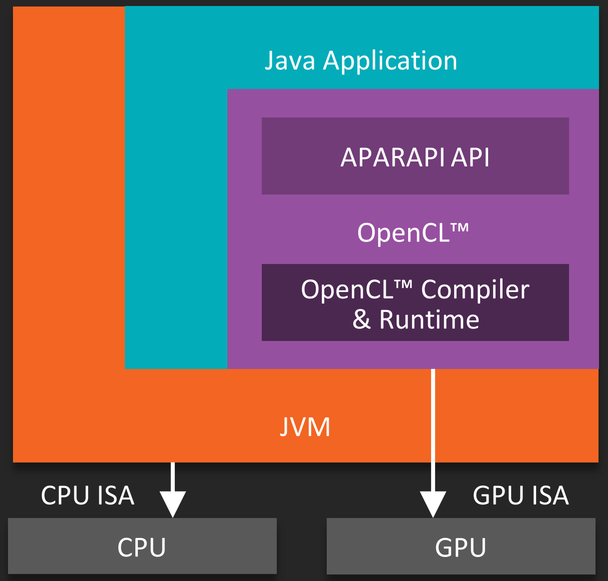
***Unified SDKs***

***Web Resources and Developer Forums***

# HSA ENABLEMENT OF JAVA

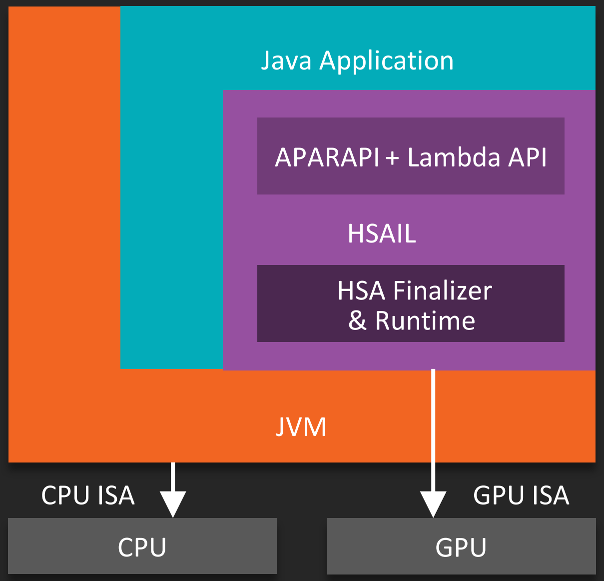
## JAVA 7 – OpenCL™ ENABLED APARAPI

- AMD initiated Open Source project
- Program only in Java
  - Accelerated by OpenCL™
- Active community captured mindshare



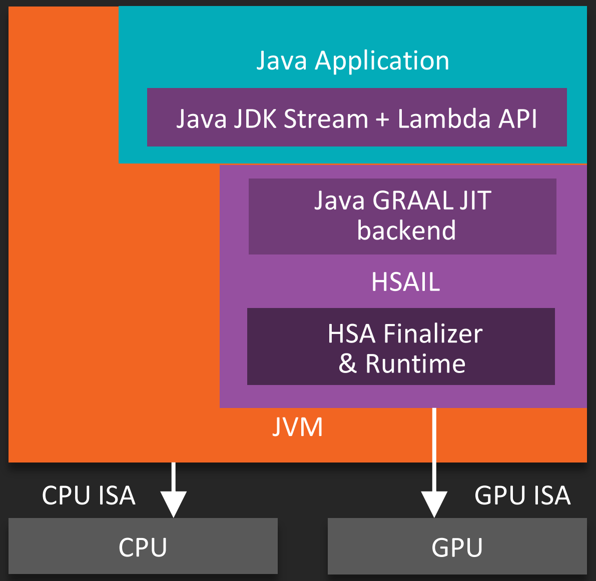

## JAVA 8 – HSA ENABLED APARAPI

- Java 8 adds Stream, Lambda APIs
  - CPU Multicore Parallelism
- APARAPI on HSA accelerates Lambdas
  - Parallel acceleration on HSA APU




## JAVA 9 – HSA ENABLED JAVA (SUMATRA)

- Adds native APU acceleration to Java Virtual Machine (JVM)
- Developer uses Lambda, Stream API
- JVM generates HSAIL automatically



# AMD'S UNIFIED SDK

AMD

- Access to AMD APU and GPU programmable components
- Component installer - choose just what you need
- Initial release includes:
- APP SDK v2.9
- Media SDK 1.0

**AMD Unified SDK**

## APP SDK 2.9

- Web-based sample browser
- Supports programming standards:  OpenCL™, C++ AMP
- Code samples for accelerated open source libraries:
  - OpenCV, OpenNI, Bolt, Aparapi
- OpenCL™ source editing plug-in for visual studio
- Now supports Cmake

## MEDIA SDK 1.0

- GPU accelerated video pre/post processing library
- Leverage AMD's media encode/decode acceleration blocks
- Library for low latency video encoding
- Supports both Windows Store and Classic desktop

# AMD CODE XL V1.3

AMD

- AMD's comprehensive heterogeneous developer tool suite including:
  - CPU and GPU Profiling
  - GPU kernel Debugging
  - GPU kernel analysis

- New features in version 1.3:
  - Supports Java, the world's most popular programming language
  - Integrated static kernel analysis
  - Remote debugging/profiling
  - Supports latest AMD APU and GPU products

## CPU PROFILER

- Time-based profiling
- Analyze call-chain relationships
- Java profiling with inline function support
- Cache-line utilization profiling
- Supports latest AMD processors

## GPU PROFILER

- OpenCL Application Trace
- Profile OpenCL kernels
- Timeline visualization of GPU counter data
- Kernel Occupancy Viewer
- Remote GPU Profiling

## GPU DEBUGGER

- Real-time OpenCL kernel debugging with stepping and variable display
- OpenCL and OpenGL API Statistics
- Object visualization
- Remote GPU debugging

## STATIC KERNEL ANALYZER

- Compile, analyze and disassemble OpenCL Kernels
- View kernel compilation errors/warnings
- Estimate kernel performance
- View generated ISA code
- View registers

# ACCELERATED OPEN SOURCE LIBRARIES

AMD

## OpenCV

- Most popular computer vision library
- Now with many OpenCL™ accelerated functions

## Bolt

- C++ template library
- Provides GPU off-load for common data-parallel algorithms
- Now with cross-OS support and improved performance/functionality

## clMath

- AMD released APPML as open source to create clMath
- Accelerated BLAS and FFT libraries
- Accessible from Fortran, C and C++

## Aparapi

- OpenCL accelerated Java 7
- Java APIs for data parallel algorithms (no need to learn OpenCL)

# "KAVERI" AND HSA HAVE ARRIVED

## THE REVOLUTION HAS STARTED!

# Questions?

# DISCLAIMER & ATTRIBUTION

AMD

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

OpenCL™ is a trademark of Apple Inc. which is licensed to the Khronos organization. Linux™ is the trademark of Linus Torvalds. Microsoft™ and Windows™ are the trademarks of Microsoft Corp. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**ATTRIBUTION**

# Backup

# MICRO-BENCHMARK SYSTEM CONFIGURATION

AMD

| Details | AMD “Kaveri” APU |
| --- | --- |
| Operating System | Microsoft® Windows 8.1® (64-bit)<br>Single Language |
| Processor | "Kaveri" A10 – 95W<br>AMD Engineering Sample<br>ZD376091I4468_40/37/18/07_130F |
| CPU speed (base/boost) | 3.7 GHz / 4.0 GHz |
| GPU speed | 720 MHz |
| Memory | 2x4GB DDR3-1600 |
| Disk | HDD |
| Video Driver | 13.35 / HSA Beta 2.2 |
| Test Dates | January 1-3, 2014 |