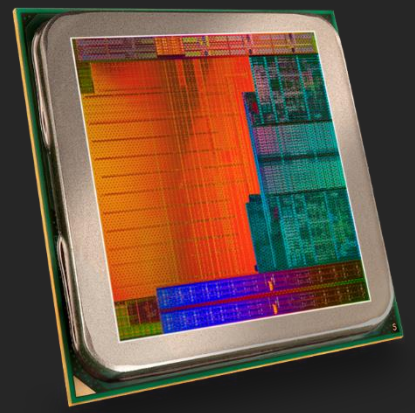
Applying AMD's "Kaveri" APU for Heterogeneous Computing

DAN BOUVIER, BEN SANDER

AUGUST 2014

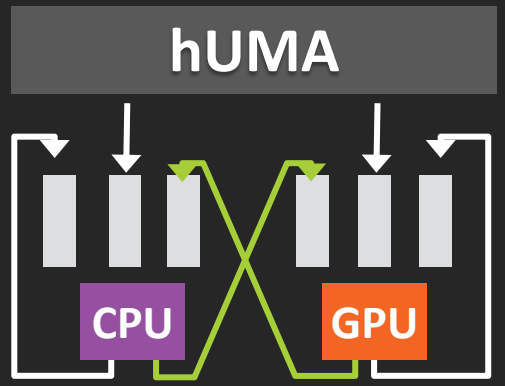
OUR DESIGN CHOICES

REDESIGNED COMPUTE CORES



- All new architecture for 45%[1] more GPU performance

HSA FEATURES TO UNLOCK GFLOPS



- Featuring shared system memory
- Heterogeneous Queuing

ADDED THE LATEST GAMING TECHNOLOGY

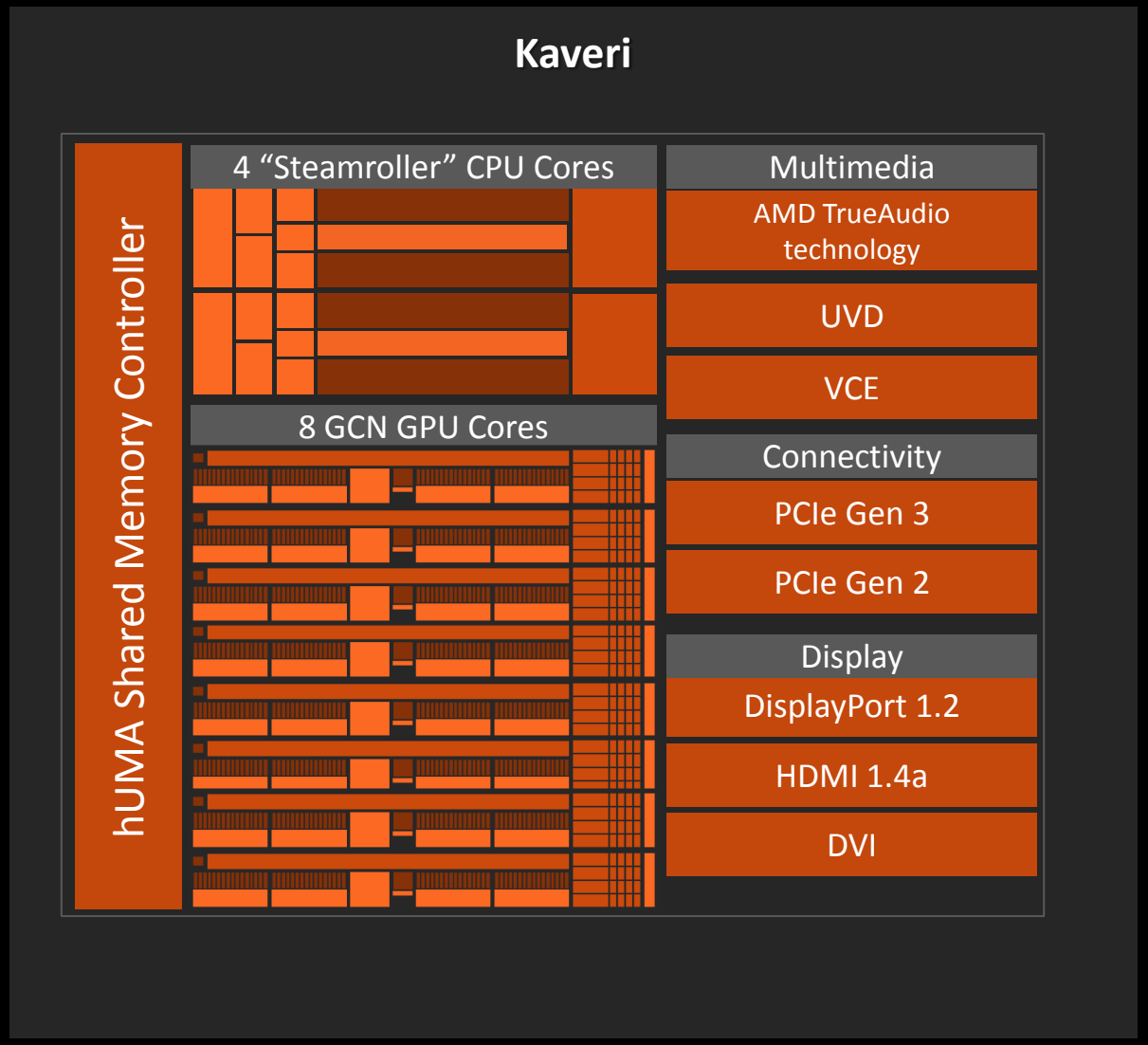
- GCN Architecture
- AMD TrueAudio technology *
- Mantle
- PCI-Express® Gen 3

ENERGY EFFICIENCY

- 95W to 15W solutions featuring configurable TDP

*AMD TrueAudio technology is offered by select AMD Radeon™ R9 and R7 200 Series GPUs and select AMD A-Series APUs and is designed to improve acoustic realism. Requires enabled game or application. Not all audio equipment supports all audio effects; additional audio equipment may be required for some audio effects. Not all products feature all technologies—check with your component or system manufacturer for specific capabilities.

A-SERIES REDEFINES COMPUTE



MAXIMUM COMPUTE PERFORMANCE

- Up to 12 compute cores*
  - 4 "Steamroller" CPU cores
  - 8 GCN GPU cores
  - HSA enabled

ENHANCED USER EXPERIENCES

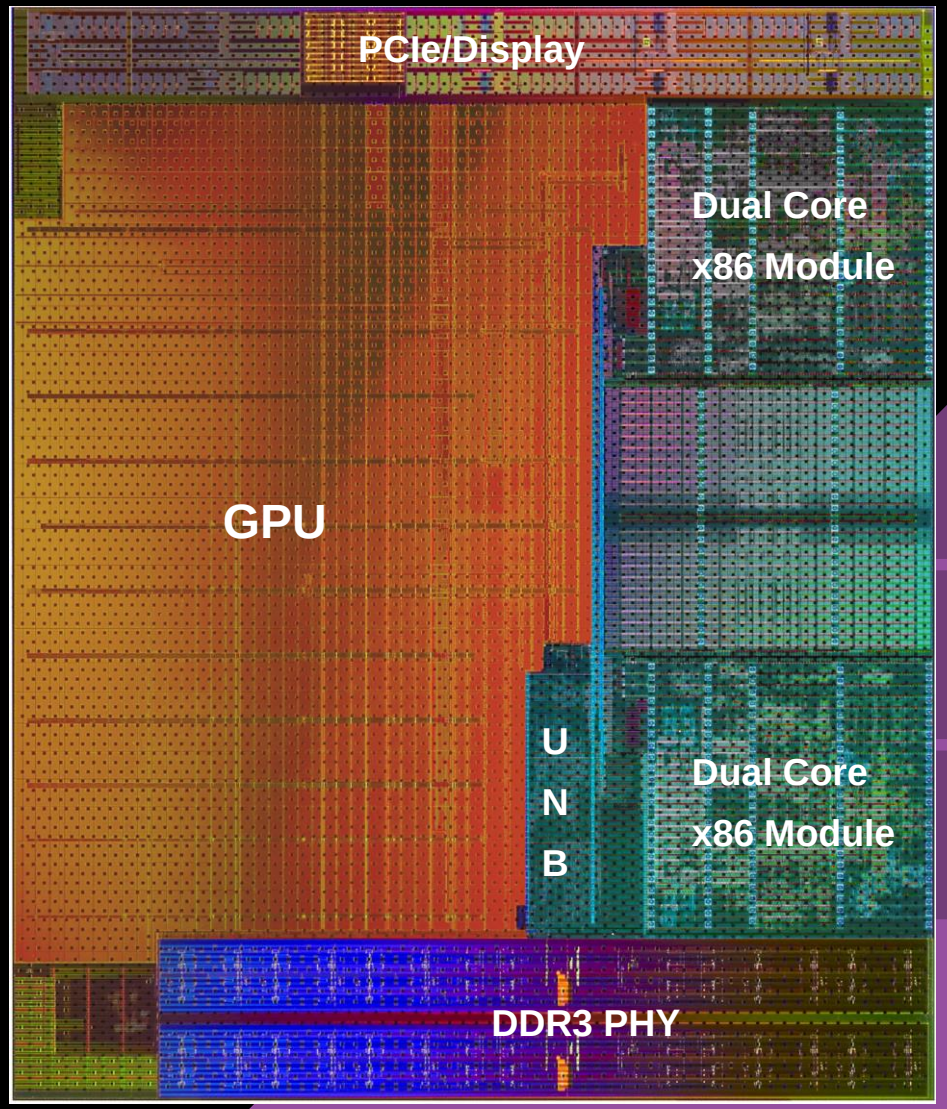
- Video acceleration
- AMD TrueAudio technology
- 4 display heads

HIGH PERFORMANCE CONNECTIVITY

- 128bits DDR3 up to 2133
- PCI-Express® Gen3 x16 for discrete graphics upgrade
- PCI-Express® for direct attach NVMe SSD

* For more information visit amd.com/computecores

“KAVERI”

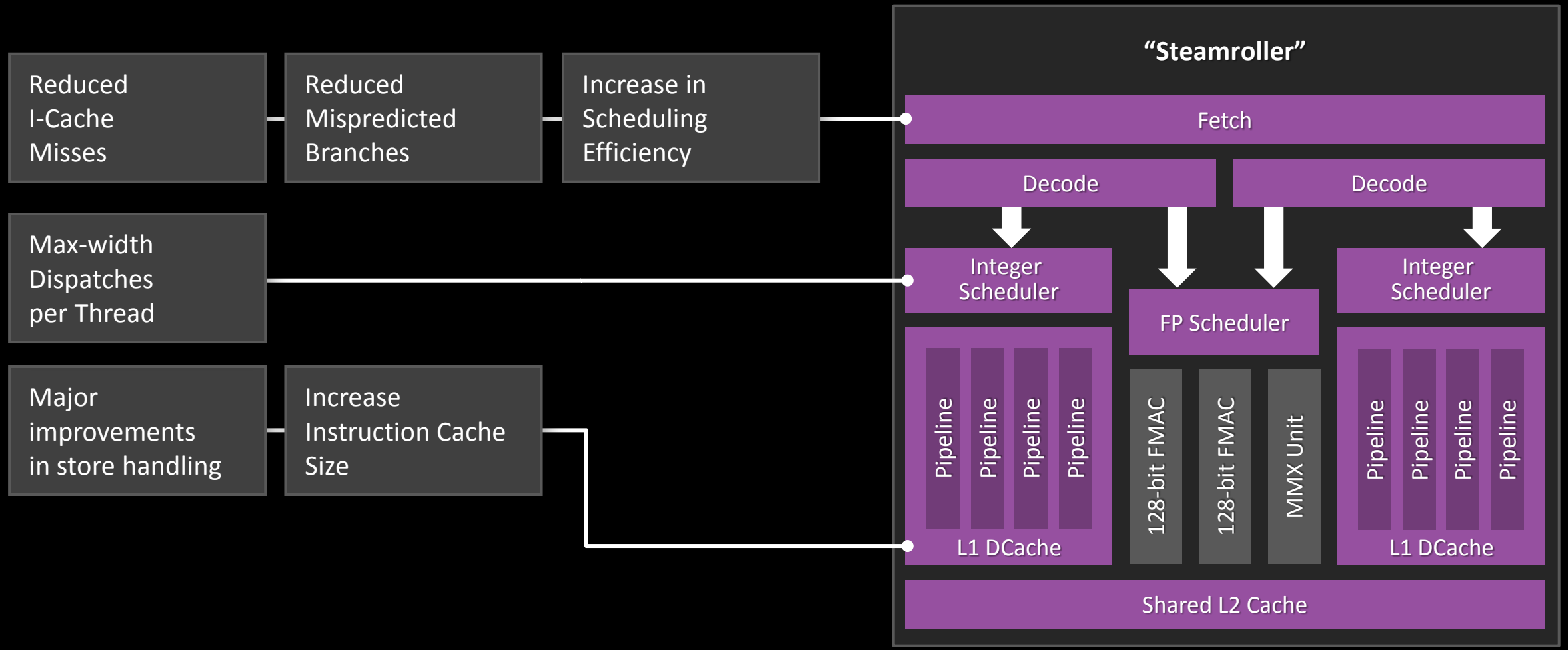


Die Size: 245mm^2

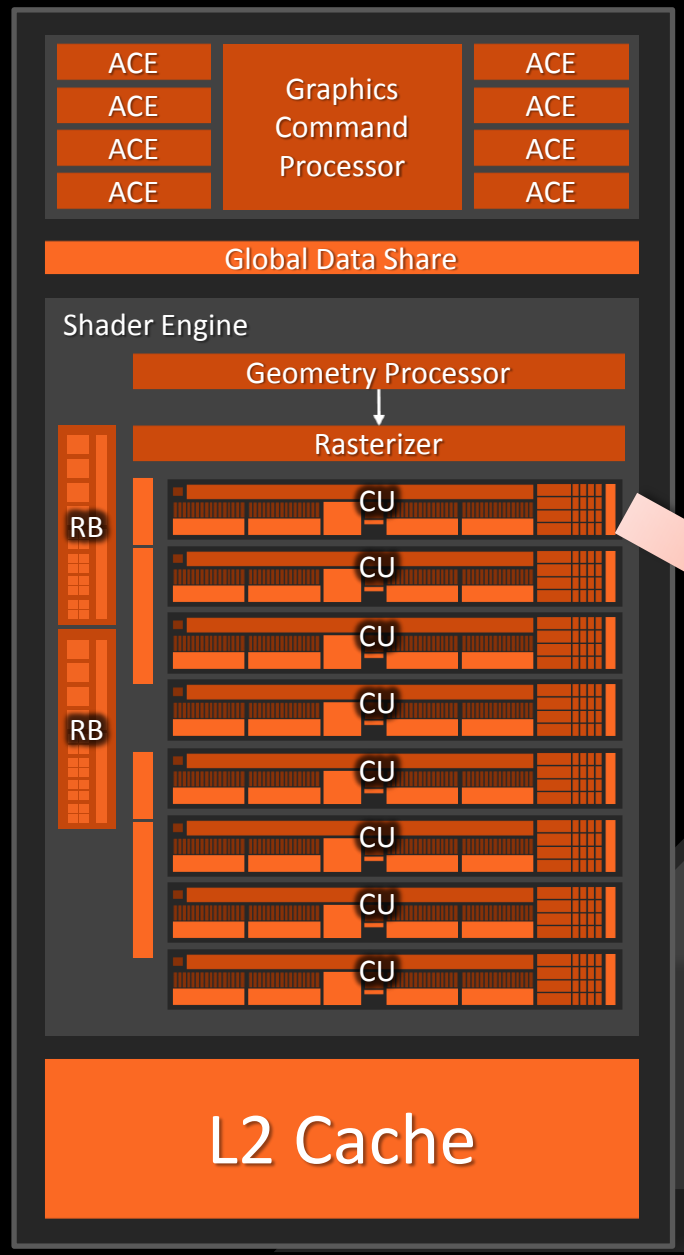
Transistor count: 2.41 Billion

Process: 28nm

IMPROVEMENTS FOR AMD'S DUAL CPU COMPUTE CORES

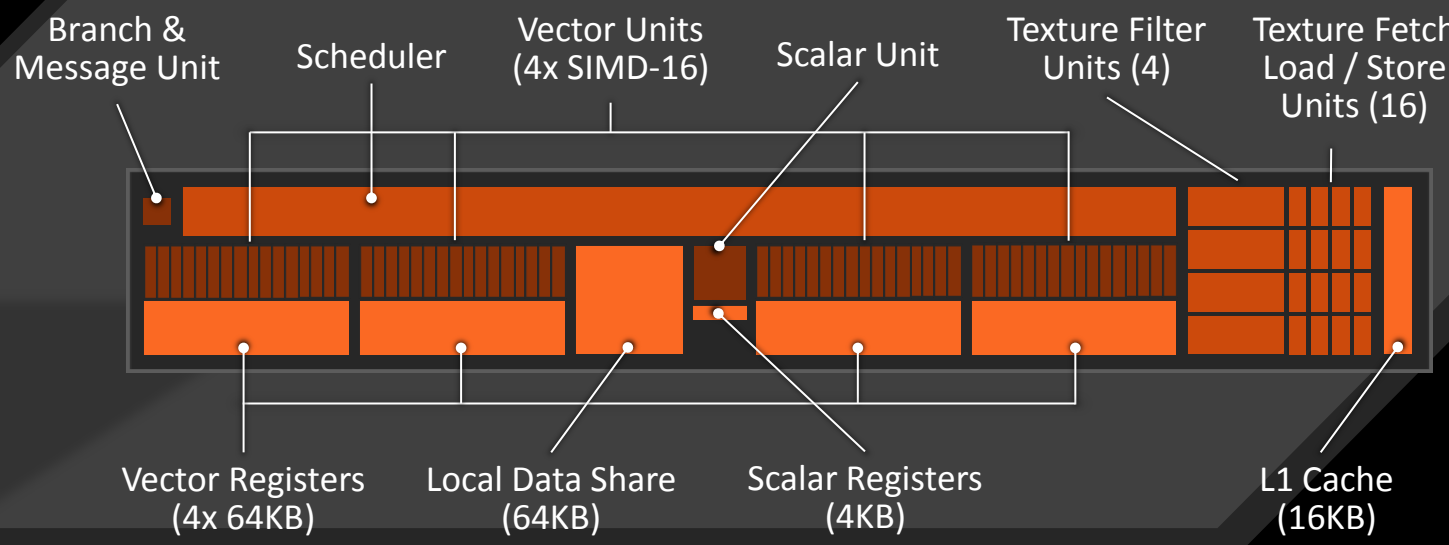


"KAVERI" GPU – GRAPHICS CORE NEXT ARCHITECTURE



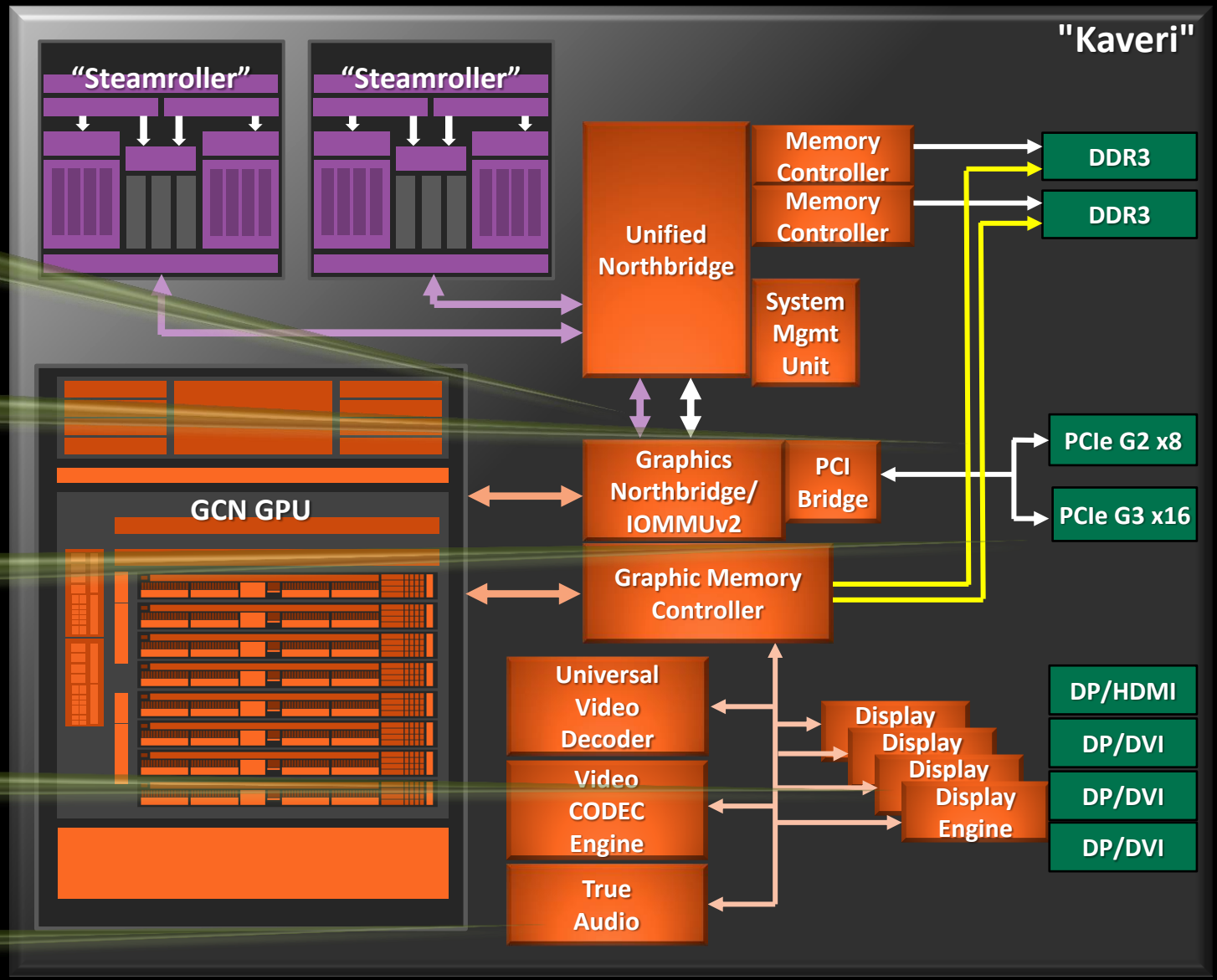
47% of "Kaveri" is dedicated for GPU

- 8 compute units (512 IEEE 2008-compliant shaders)
- Device flat (generic) addressing support
- Masked Quad Sum of Absolute Difference (MQSAD) with 32b accumulation and saturation
- Precision improvement for native LOG/EXP ops to 1ULP



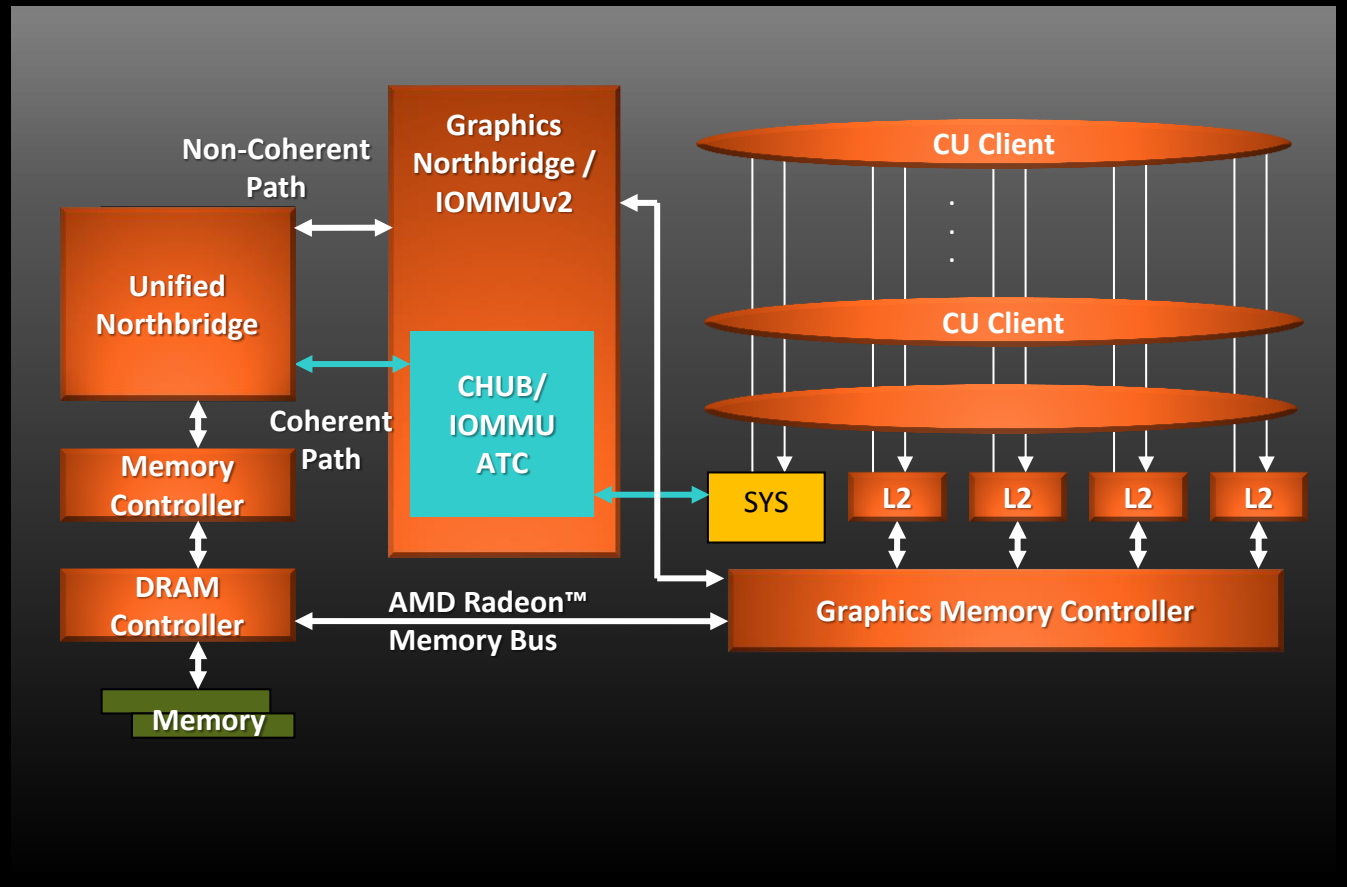
"KAVERI" APU ENHANCEMENTS

- Coherent Hub and IOMMU
  - Dedicated coherent transaction path
- AMD Radeon™ Memory Bus
  - High bandwidth graphics data
- PCI-Express® Gen3 for External Graphics Attach
- Four Display Engines
- AMD TrueAudio Technology



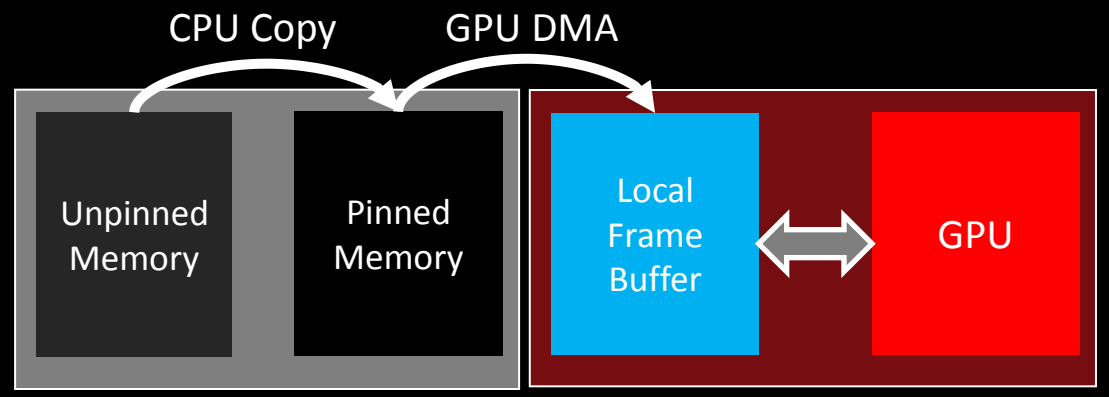
INTRODUCTION OF HARDWARE COHERENCY FOR THE GPU

- Coherent Hub (CHUB)
  - Compute traffic steered to dedicated coherent transaction path
  - Includes IOMMU Address Translation Cache (ATC)
  - Selectively probe CPU caches based on page attribute
- Atomics
  - Single cycle request
  - One at a time (no gathering at the SYS level)
  - All atomics return the original data from DRAM (success of conditional)
  - TYPES Supported:
    Test and OR, Swap, Add, Subtract, AND, OR, XOR, Signed Min, Signed Max, Unsigned Min, Unsigned Max, Clamping Inc, Clamping Dec, Compare and Swap



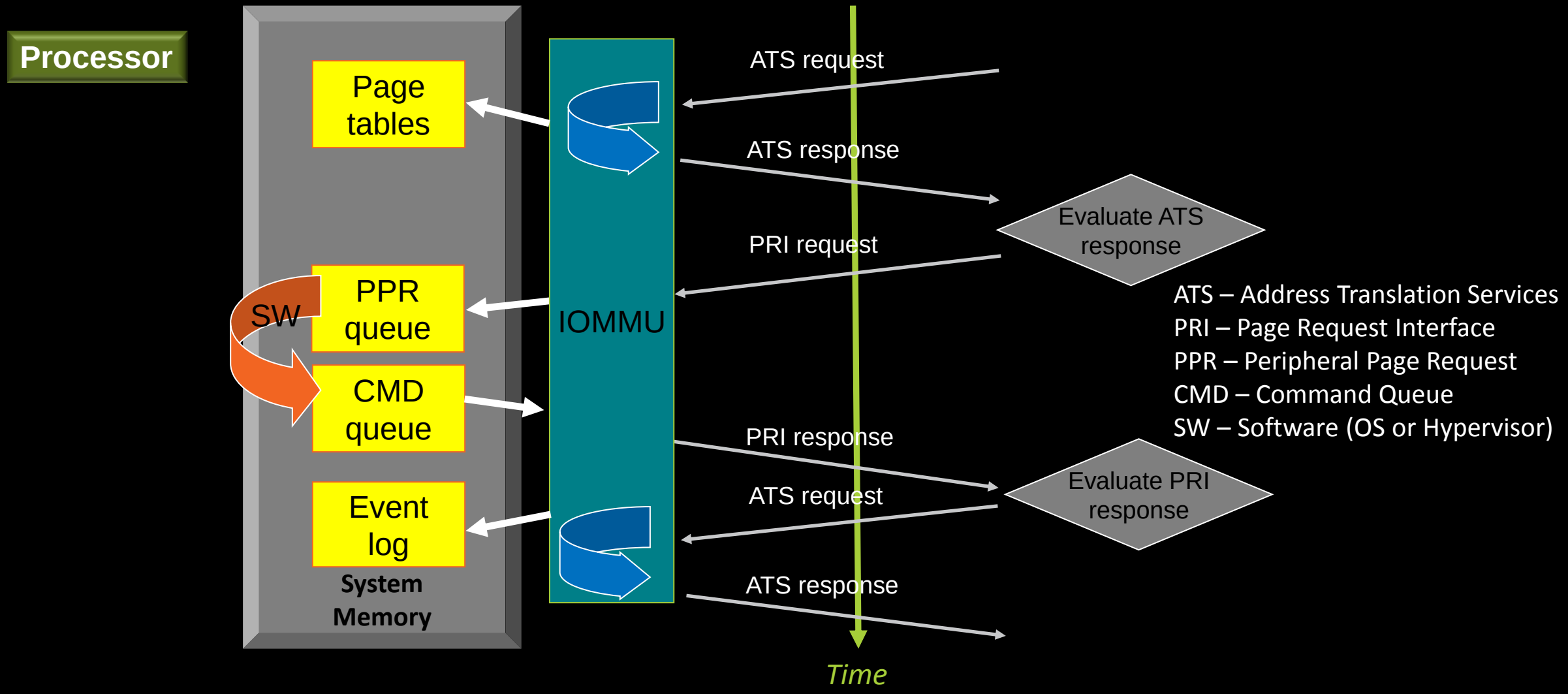
IOMMUv2 – ELIMINATES DOUBLE COPY

- **With traditional memory system**
  - Not all GPU memory is CPU accessible (e.g. local frame buffer memory)
  - Local frame buffer may not be large enough working space
  - Lack of demand-paging support
  - Alignment limitations
  - Data copied from unpinned to pinned regions



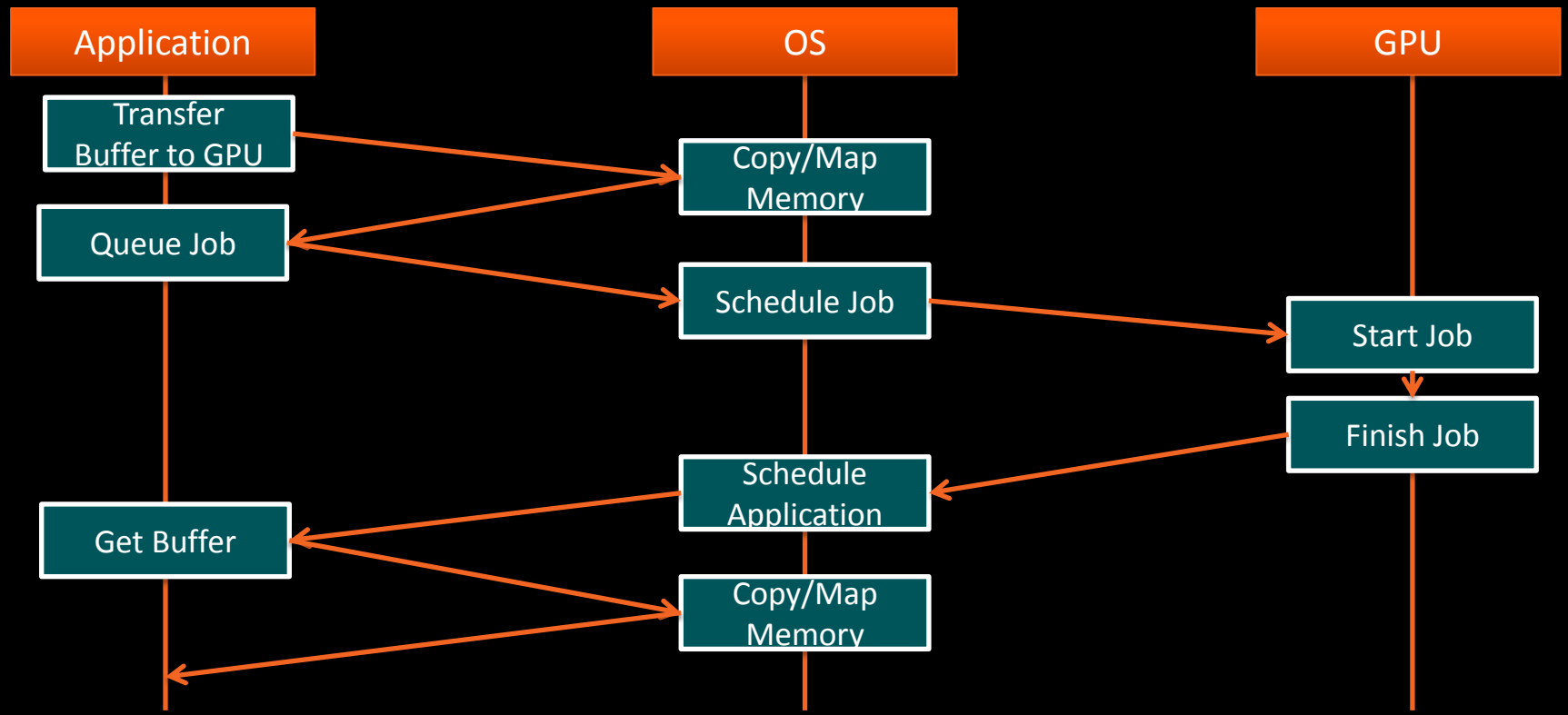
- **With IOMMUv2**
  - Eliminate CPU & DMA copy operations (in both directions!!)
  - GPU operates on unpinned region directly

IOMMUV2 PERIPHERAL PAGE FAULTS



QUEUING (TODAY'S PICTURE)

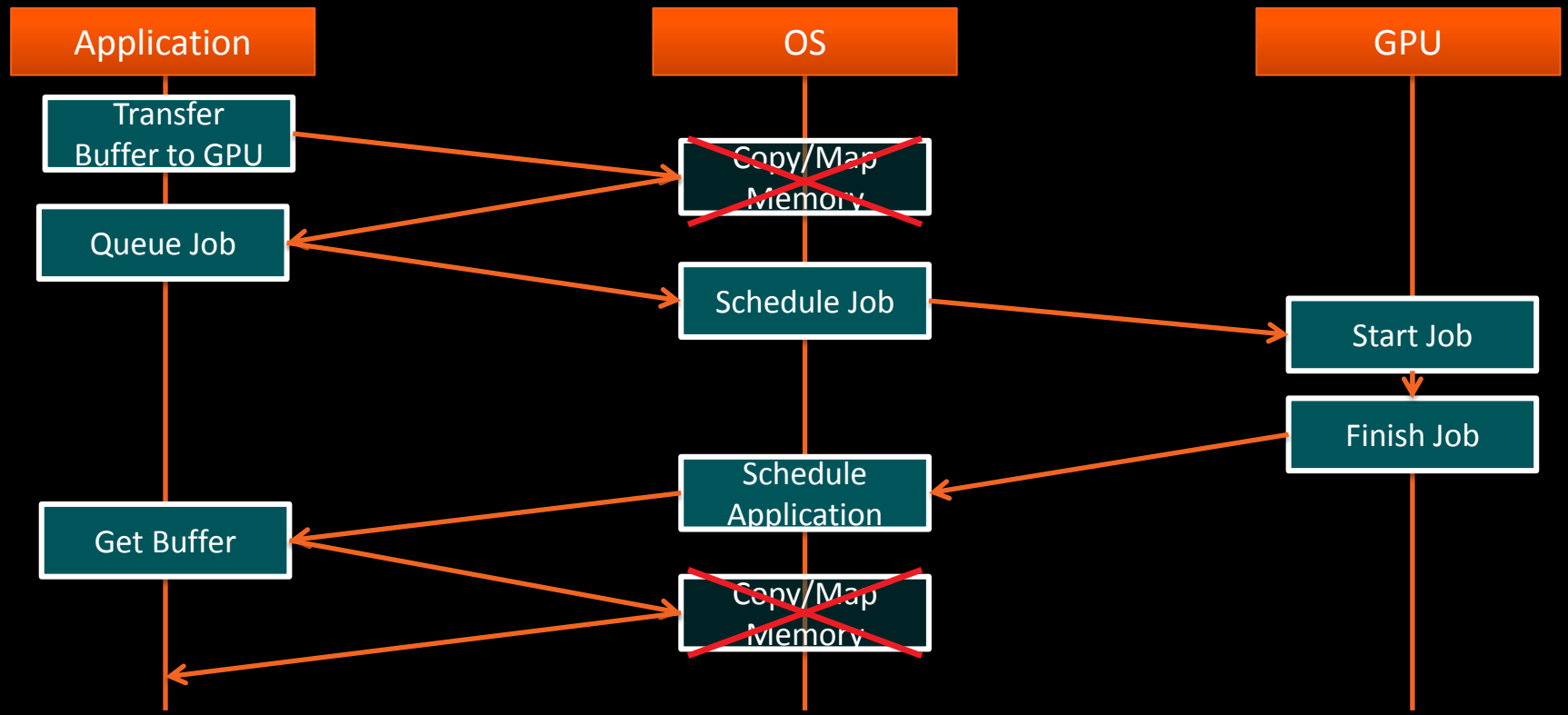
- High overhead to pass work to/from GPU



QUEUING WITH HSA ON KAVERI

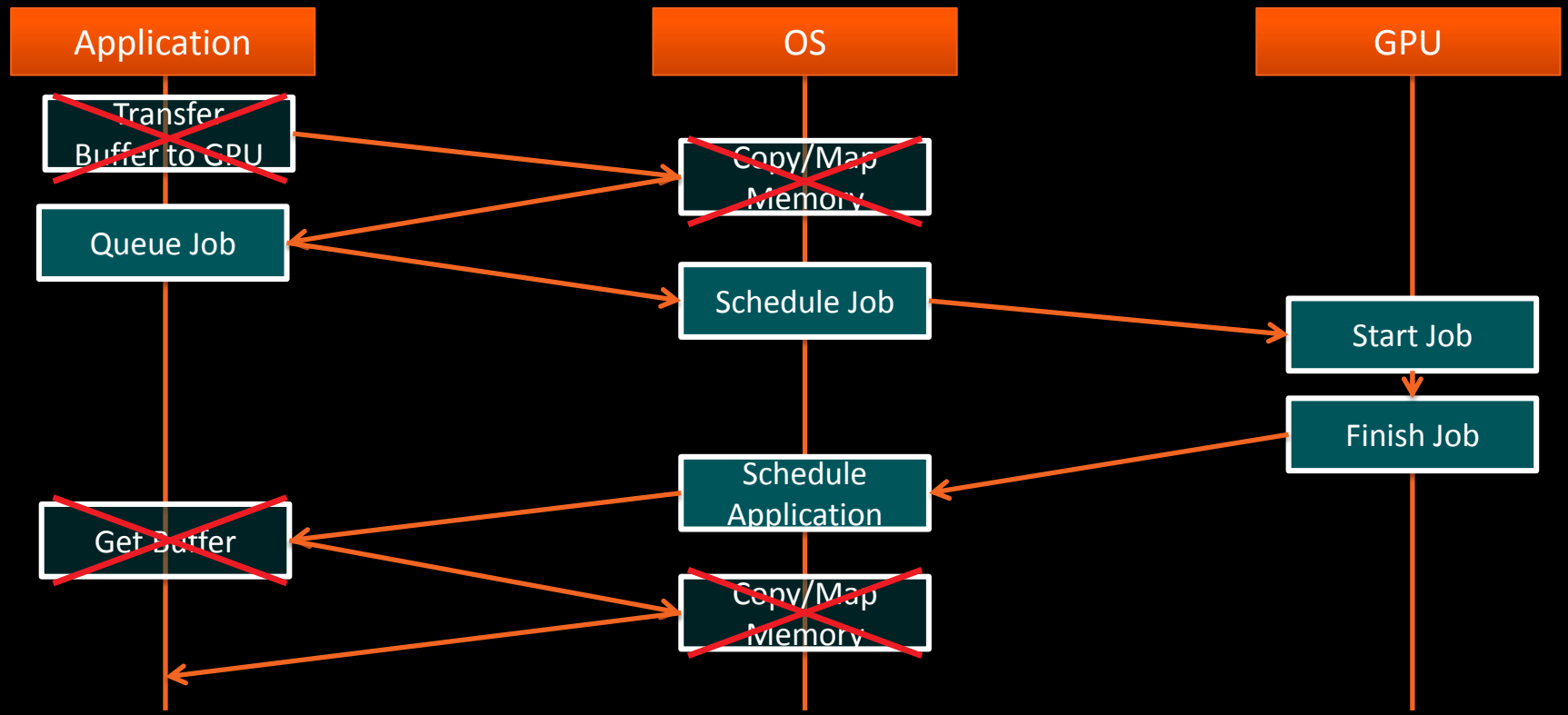
AMD

- Shared Virtual Memory



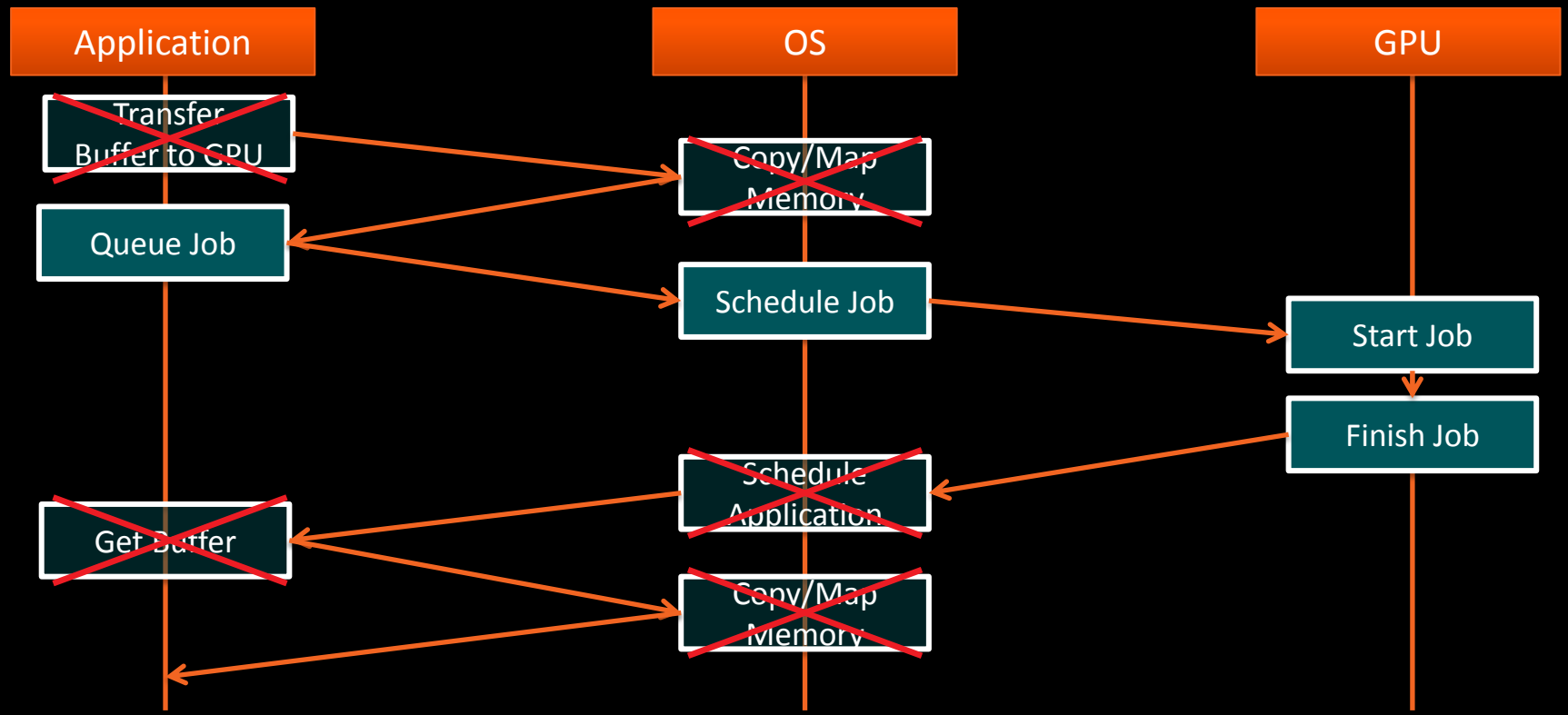
QUEUING WITH HSA ON KAVERI

- Shared Virtual Memory
- System Coherency



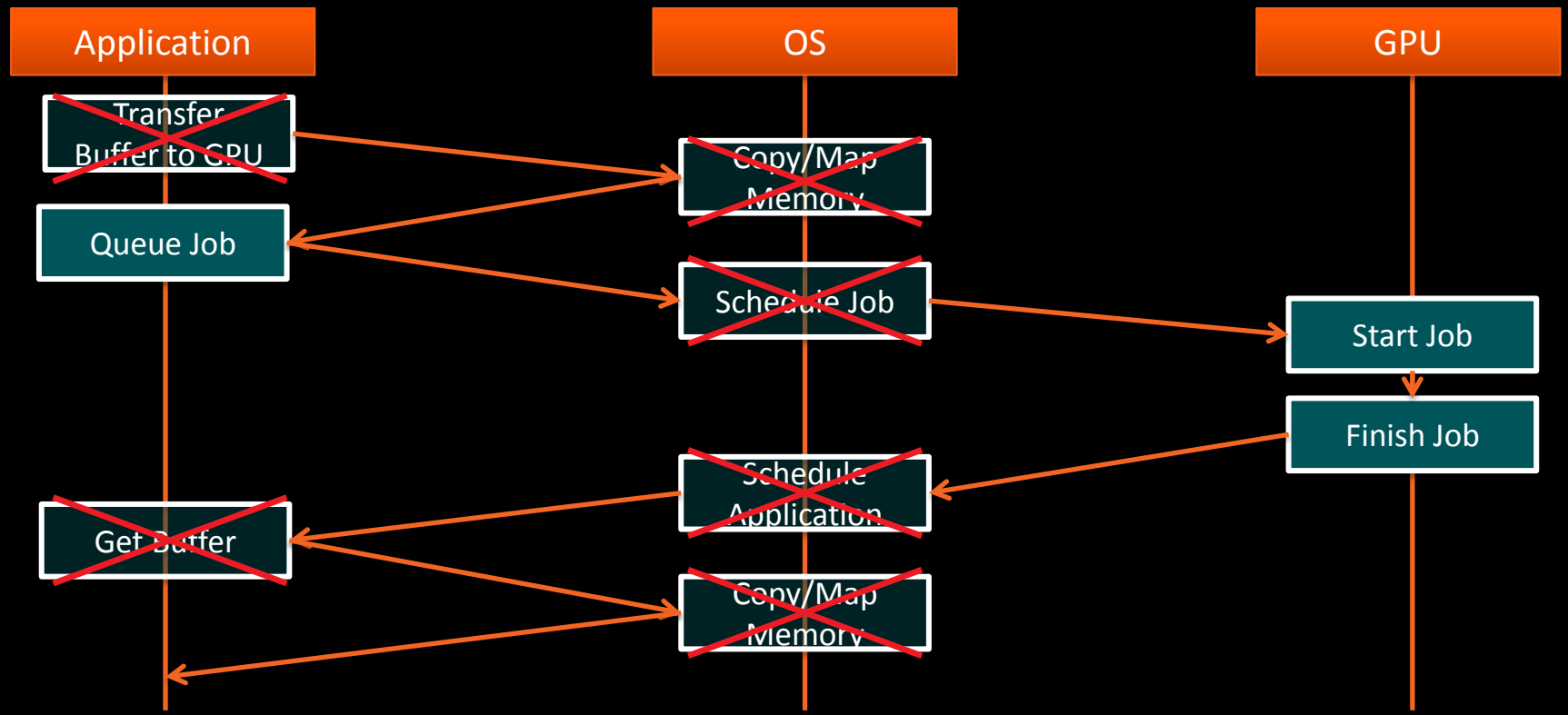
QUEUING WITH HSA ON KAVERI

- Shared Virtual Memory
- System Coherency
- Signaling



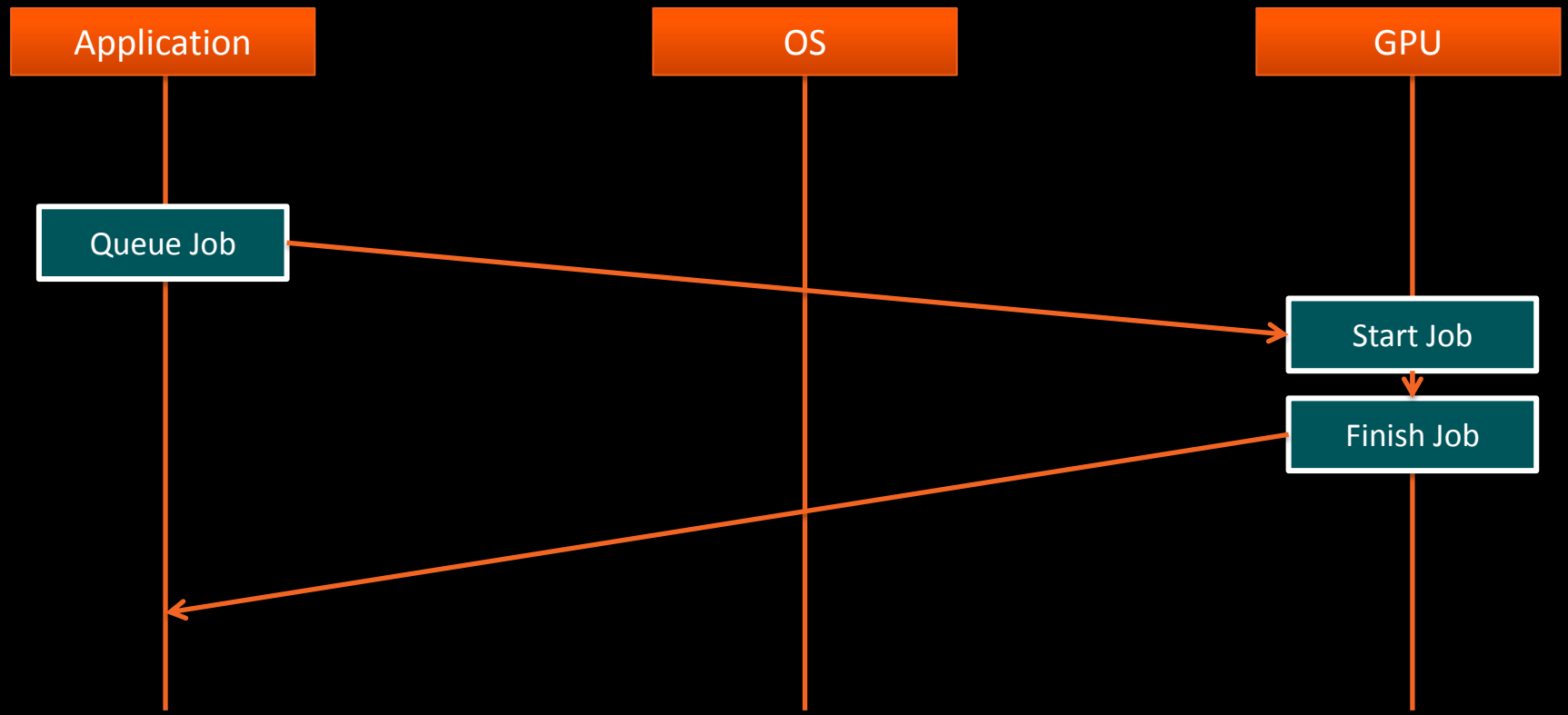
QUEUING WITH HSA ON KAVERI

- Shared Virtual Memory
- System Coherency
- Signaling
- User Mode Queuing



QUEUING WITH HSA ON KAVERI

- Shared Virtual Memory
- System Coherency
- Signaling
- User Mode Queuing



HSA IN A NUTSHELL

AMD

HSA Hardware Building Blocks

- Shared Virtual Memory
  - Single address space
  - Coherent
  - Pageable
  - Fast access from all components
  - Can share pointers
- Architected User-Level Queues
- Signals

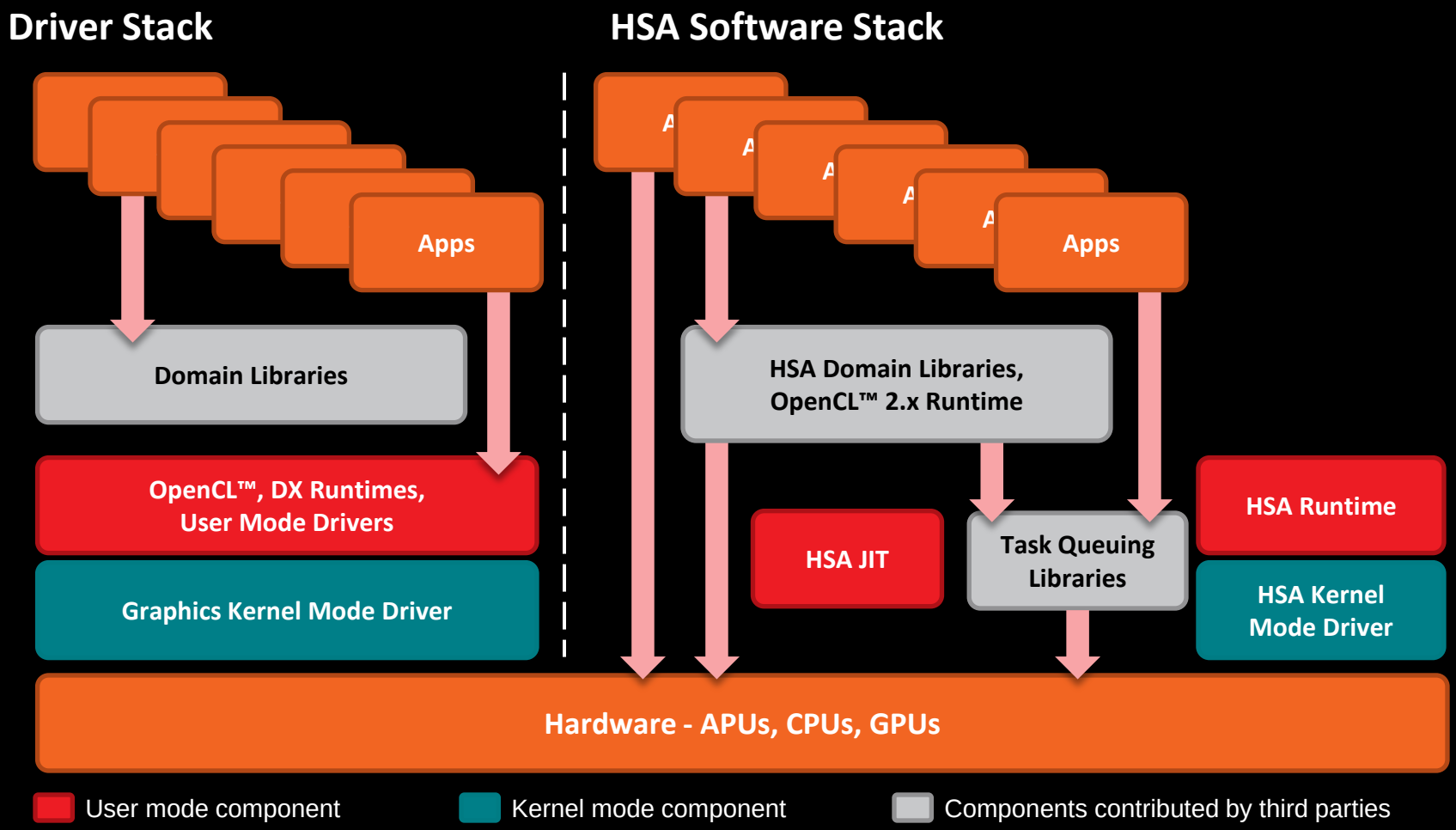
Provide industry-standard, architected requirements for how devices share memory and communicate with each other

HSA Software Building Blocks

- HSAIL
  - Portable, parallel, compiler IR
- HSA Runtime
  - Create queues
  - Allocate memory
  - Device discovery
- Reference High-level Compiler
  - CLANG/LLVM
  - Generate HSAIL
  - OpenCL, C++AMP

Provide industry-standard compiler IR and runtime to enable existing programming languages to target the GPU

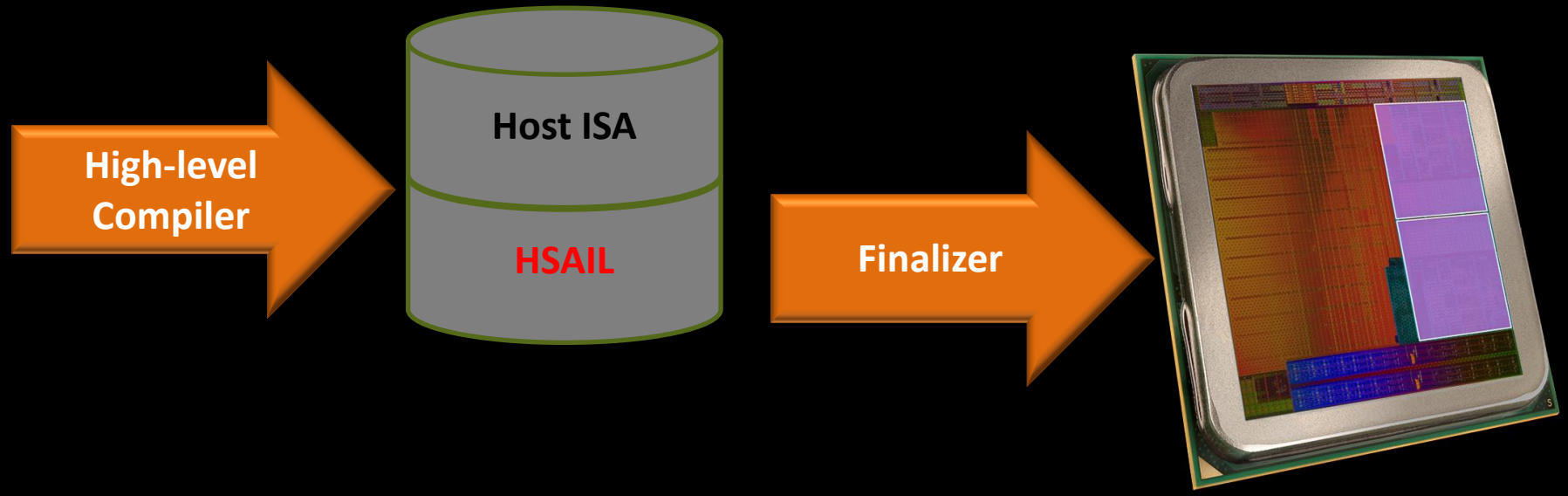
EVOLUTION OF THE SOFTWARE STACK



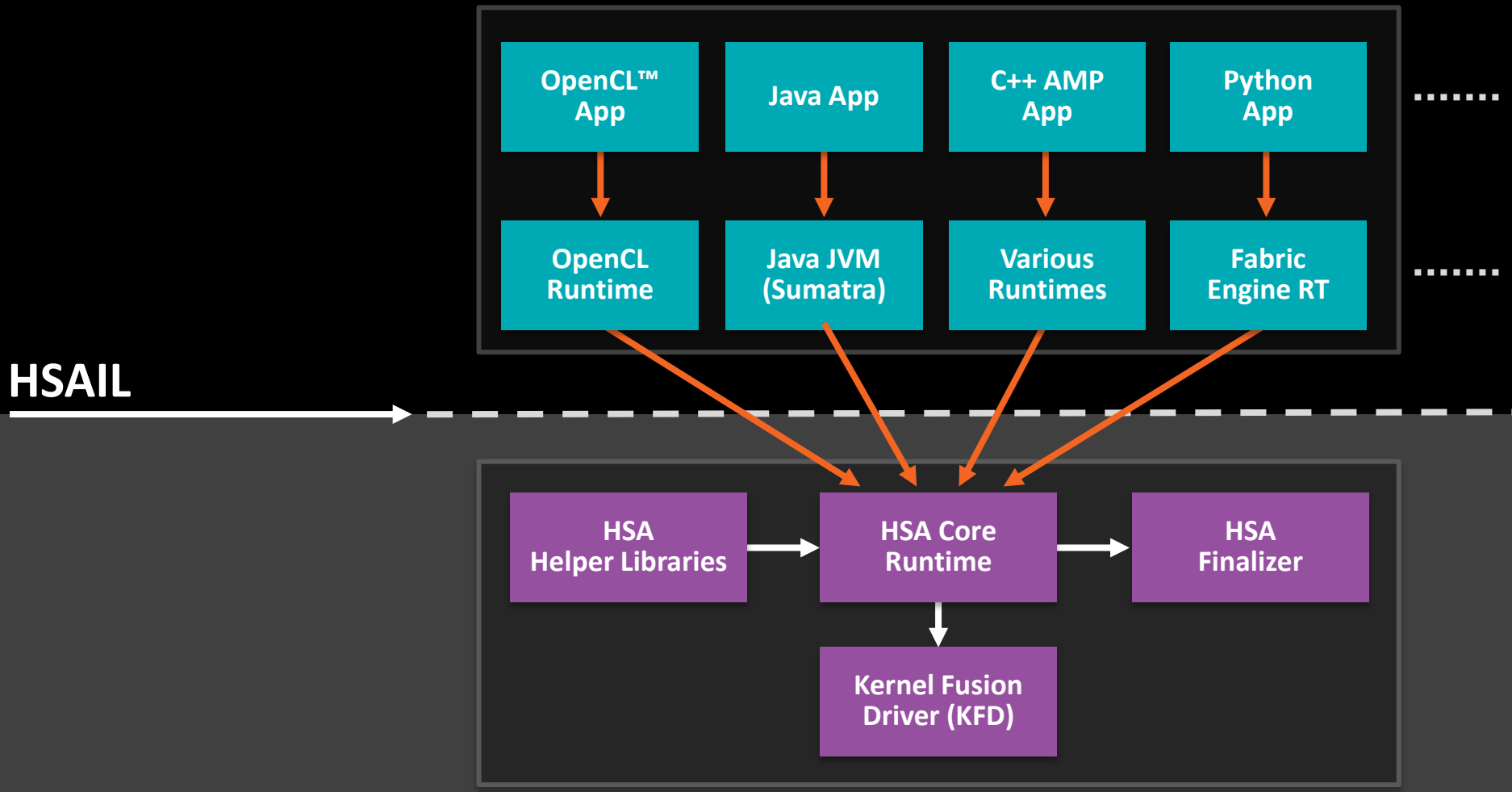
WHAT IS HSAIL?

- Intermediate language for parallel compute in HSA
- Generated by a “High-level Compiler” (GCC, LLVM, Java VM, etc)
- Expresses parallel regions of code
- Portable across vendors and stable across product generations
- *Goal: Bring parallel acceleration to mainstream programming languages (OpenMP, C++AMP, Java)*

```
main() {
…

#pragma omp parallel for
for (int i=0;i<N; i++) {
}


…
}
```



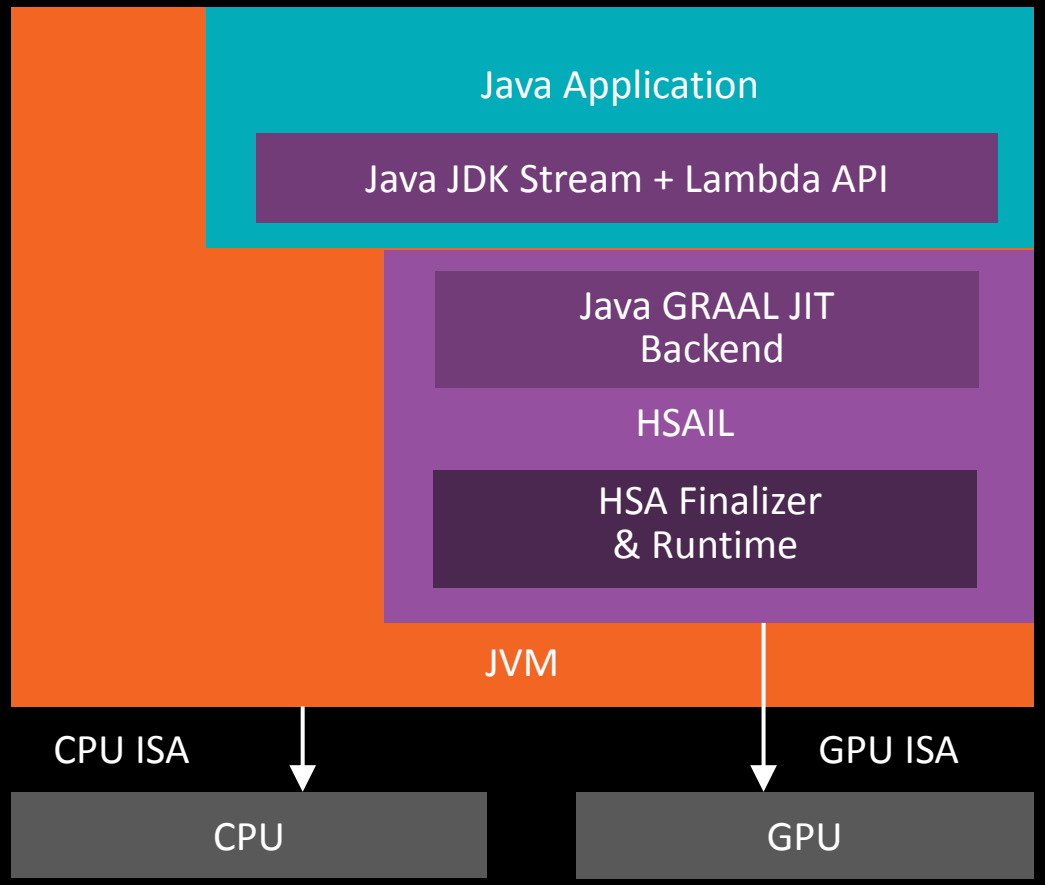
PROGRAMMING LANGUAGES PROLIFERATING ON HSA



HSA ENABLEMENT OF JAVA

JAVA 9 – HSA-ENABLED JAVA (SUMATRA)

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



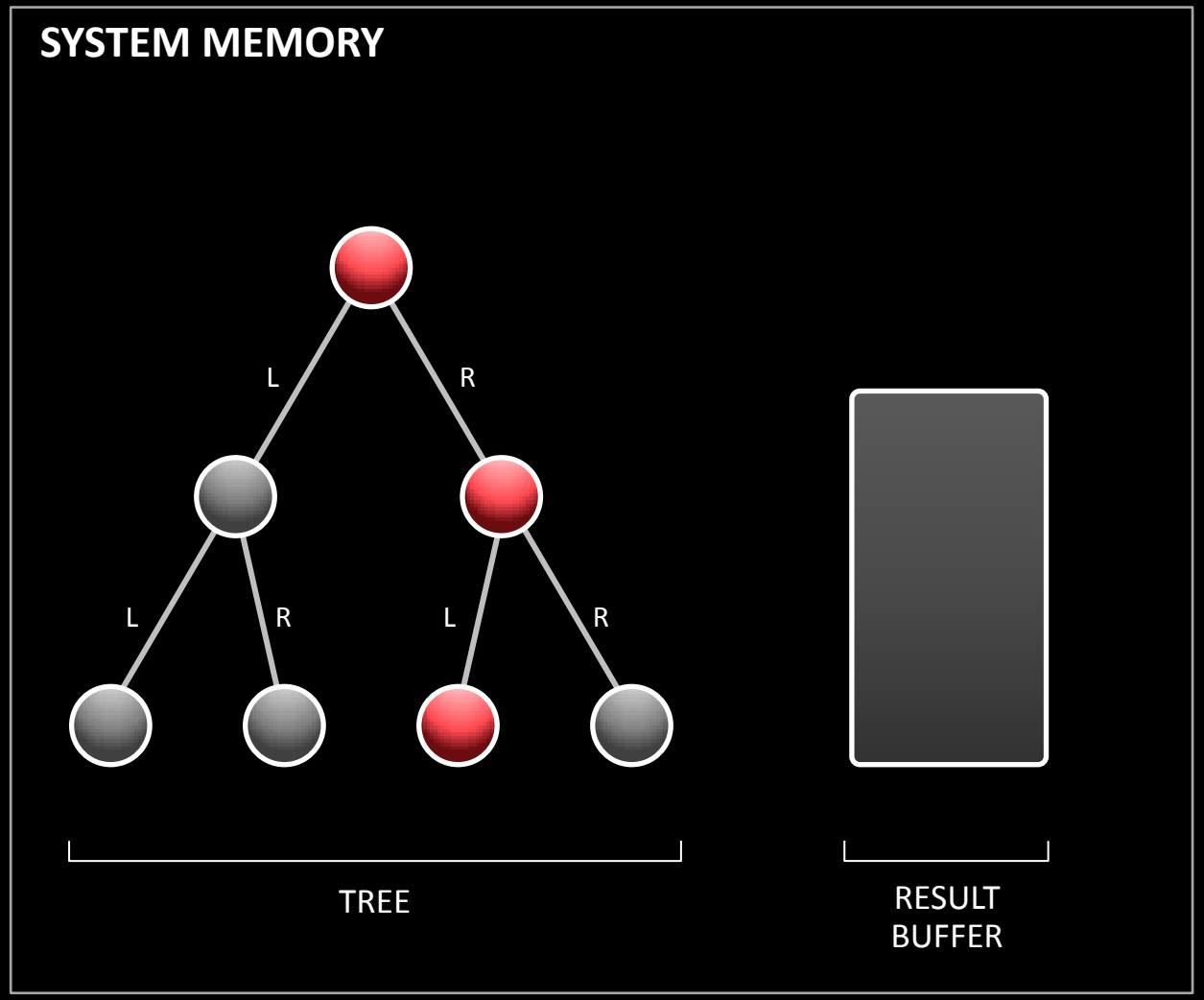
USE CASES SHOWING HSA ADVANTAGE ON KAVERI

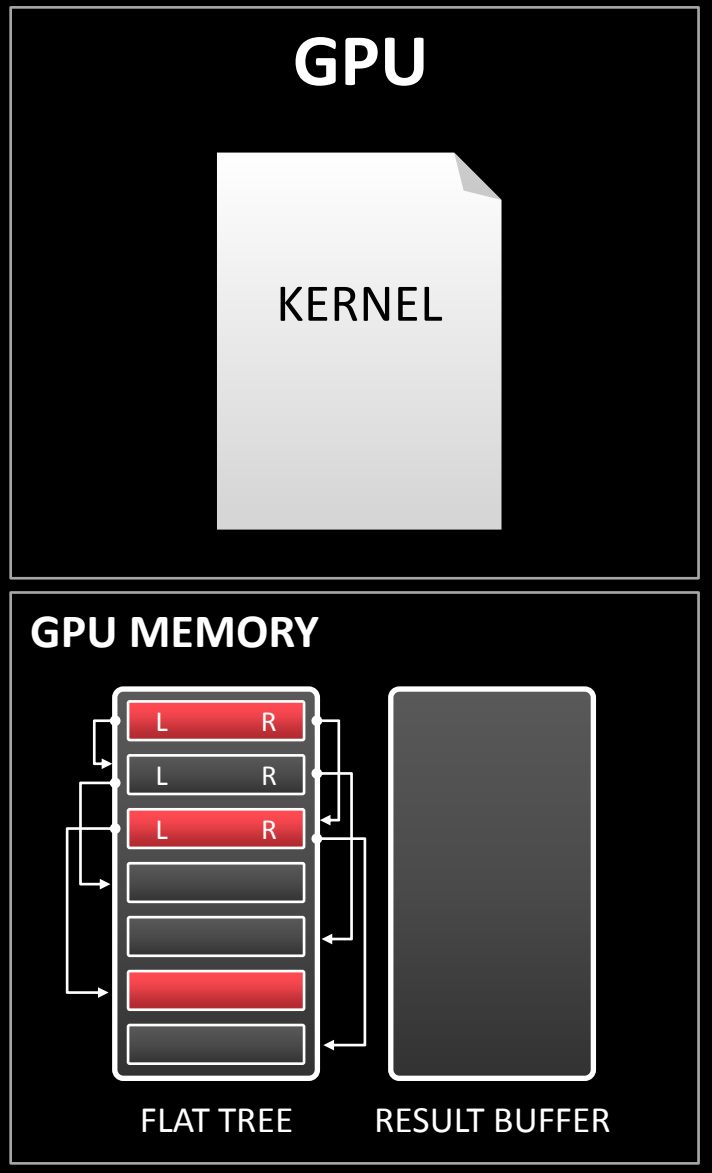
Programming Technique	Use Case Description	HSA Advantage
Data Pointers	**Binary tree searches** GPU performs searches in a CPU-created binary tree	GPU can access existing data structures containing pointers Higher performance through parallel operations
Platform Atomics	**Binary tree updates** CPU and GPU operating simultaneously on the tree, both doing modifications	CPU and GPU can synchronize using Platform Atomics Higher performance through parallel operations
Large Data Sets	**Hierarchical data searches** Applications include object recognition, collision detection, global illumination, BVH	GPU can operate on huge models in place Higher performance through parallel operations
CPU Callbacks	**Middleware user-callbacks** 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 Simpler programming does not require “split kernels” Higher performance through parallel operations

Data Pointers

DATA POINTERS

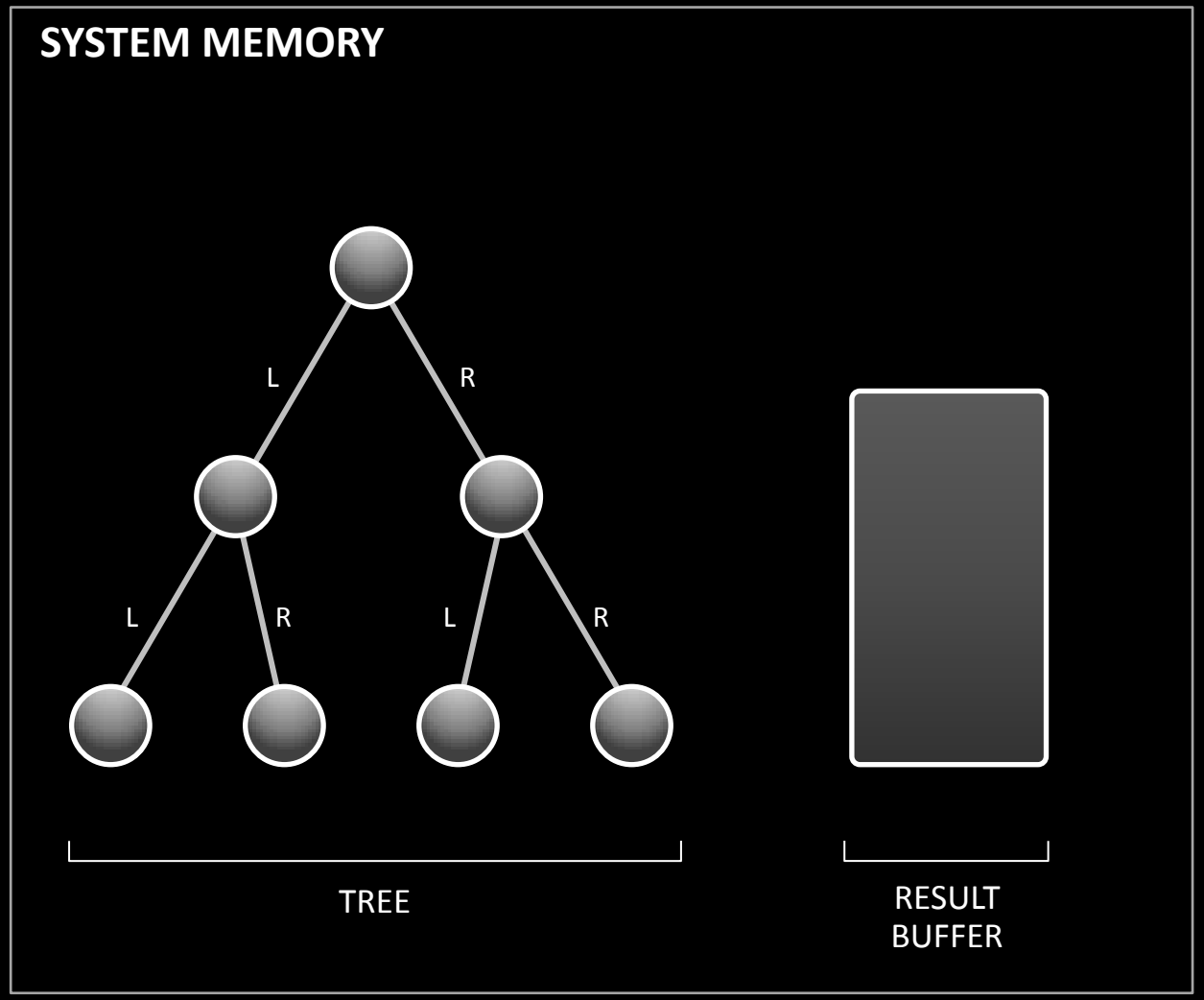
Legacy

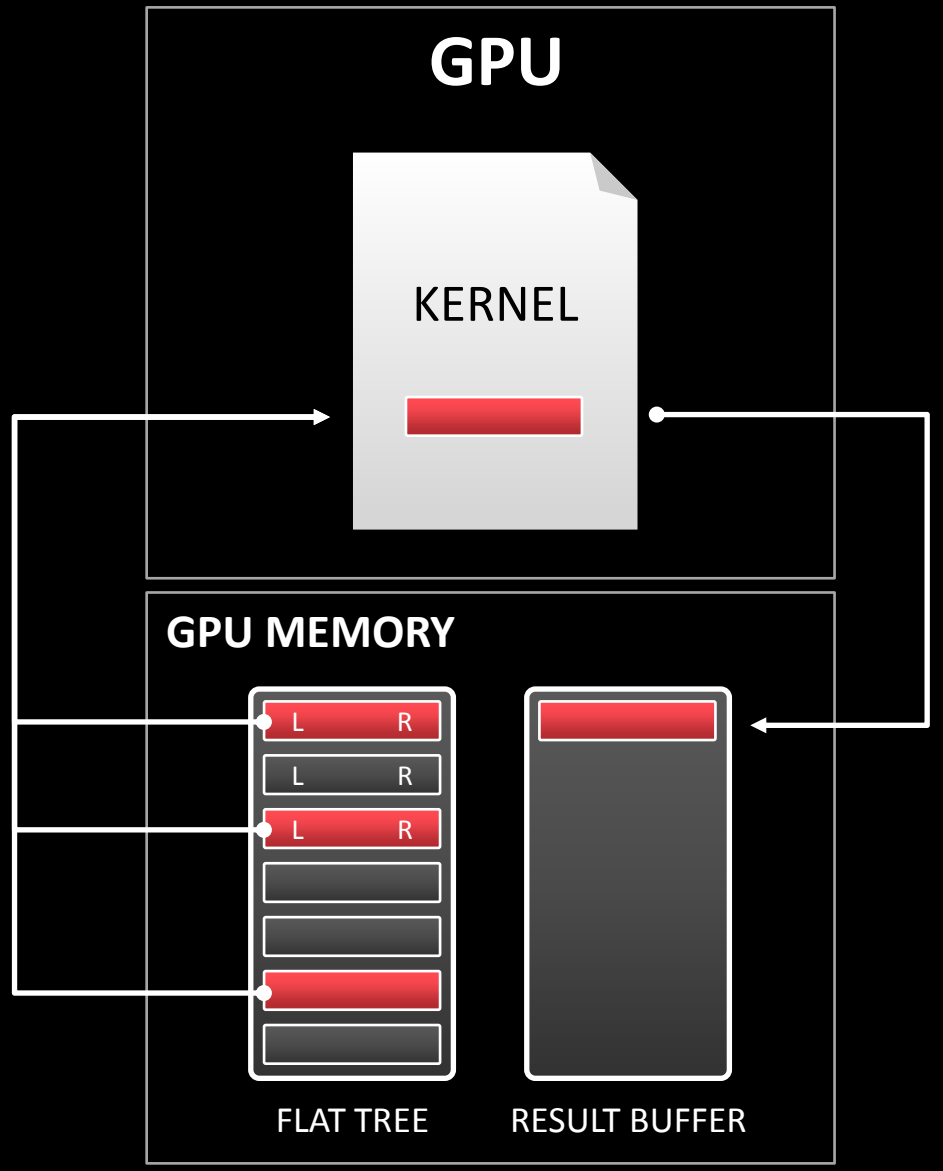




DATA POINTERS

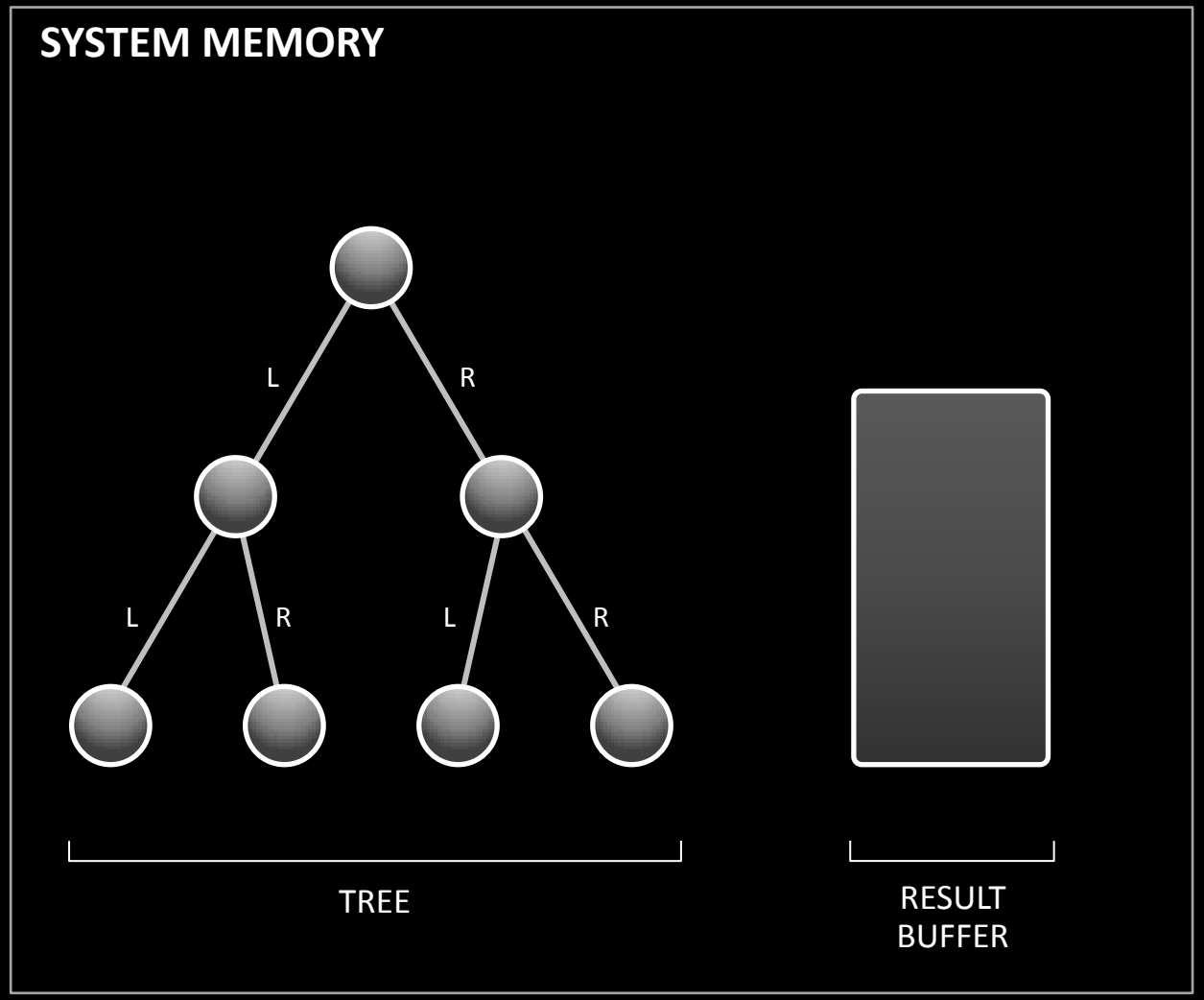
Legacy

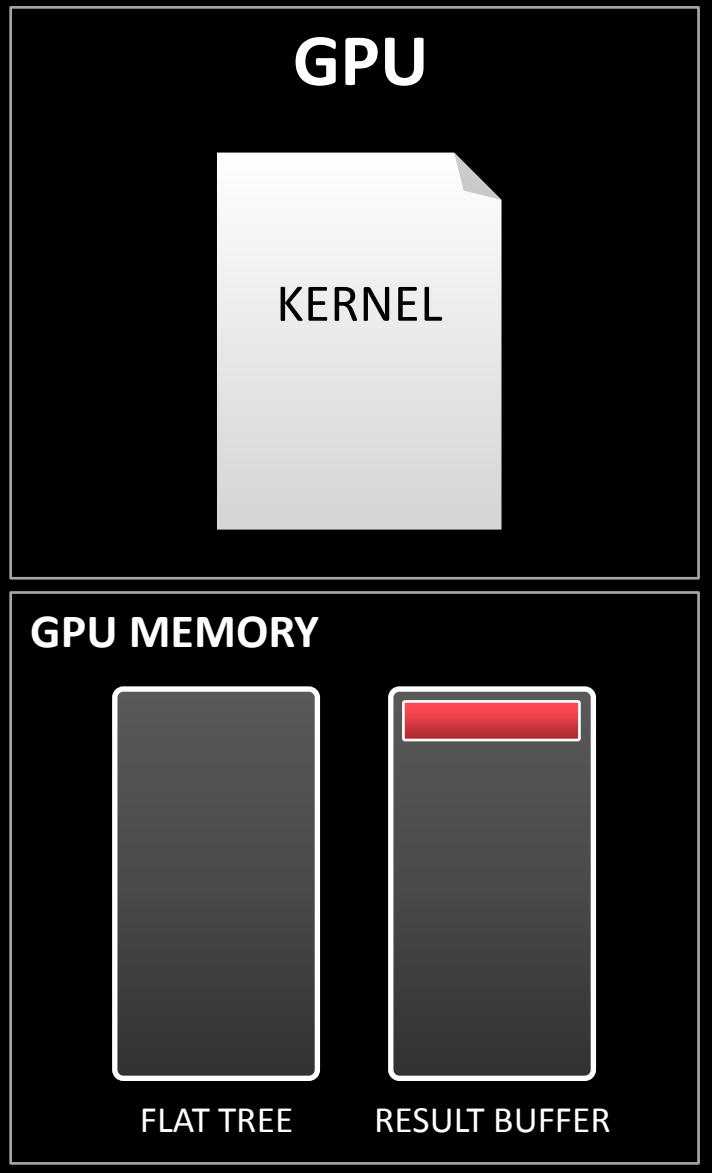




DATA POINTERS

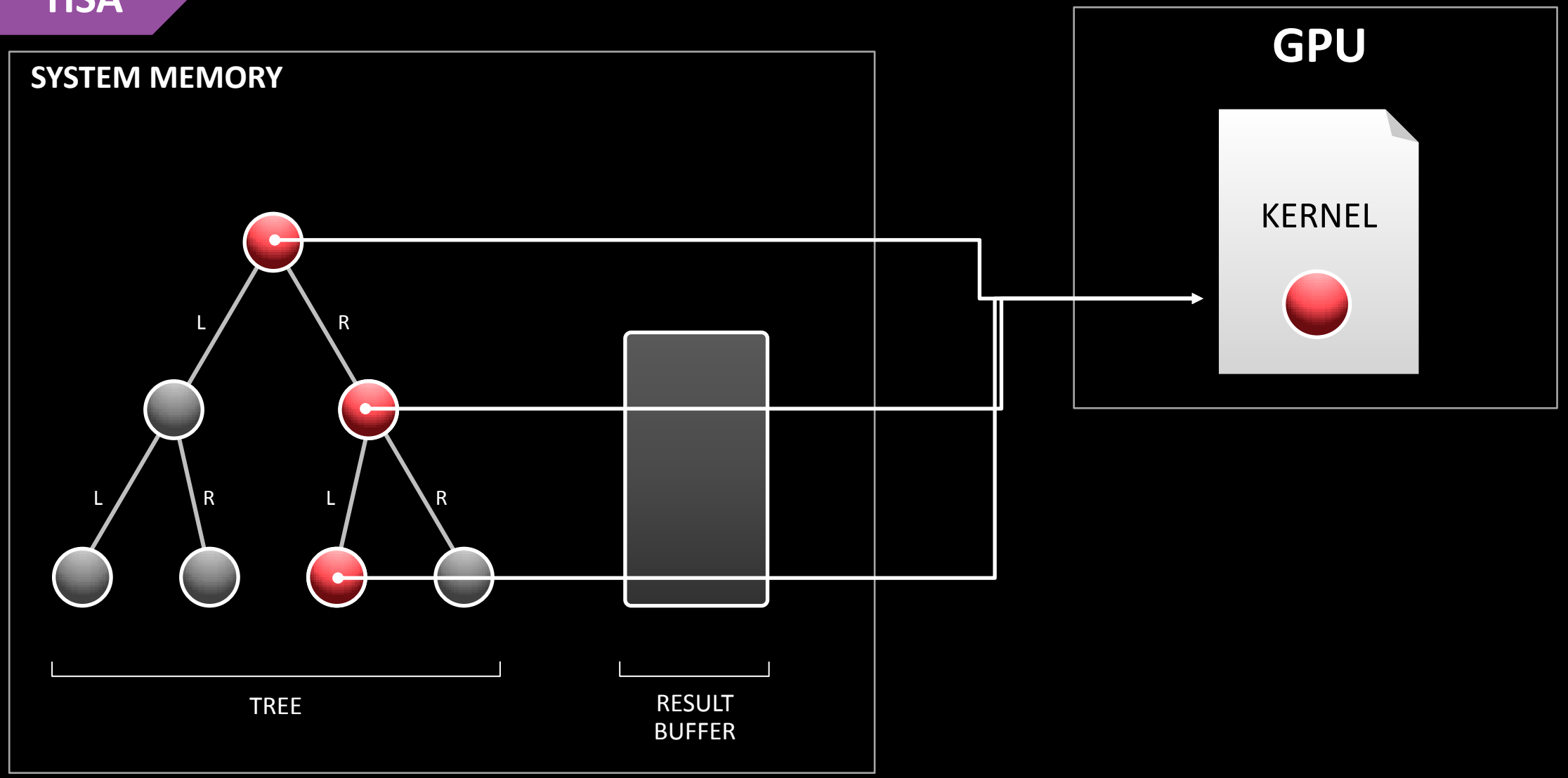
Legacy





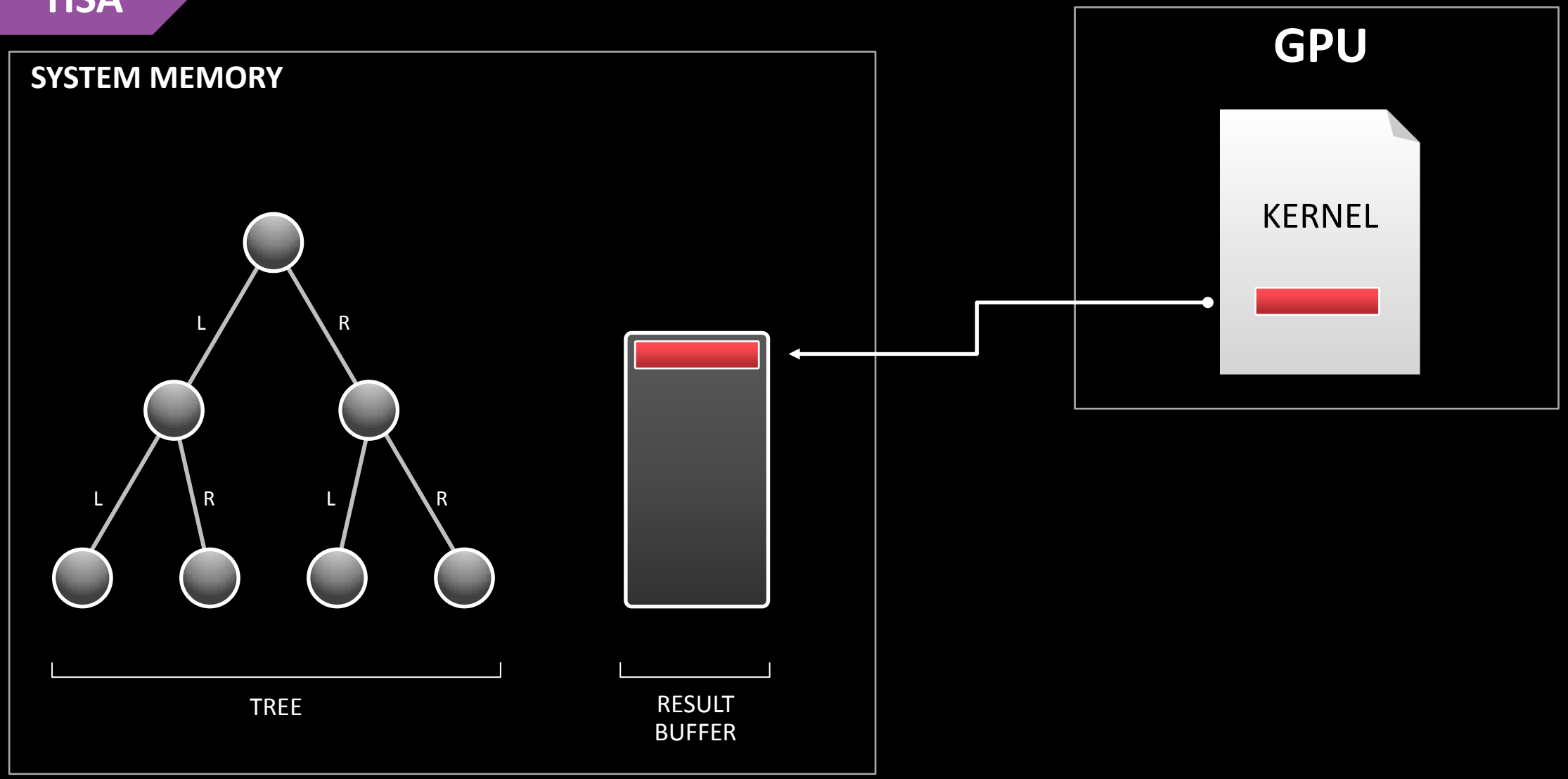
DATA POINTERS

HSA



DATA POINTERS

HSA



DATA POINTERS - CODE COMPLEXITY

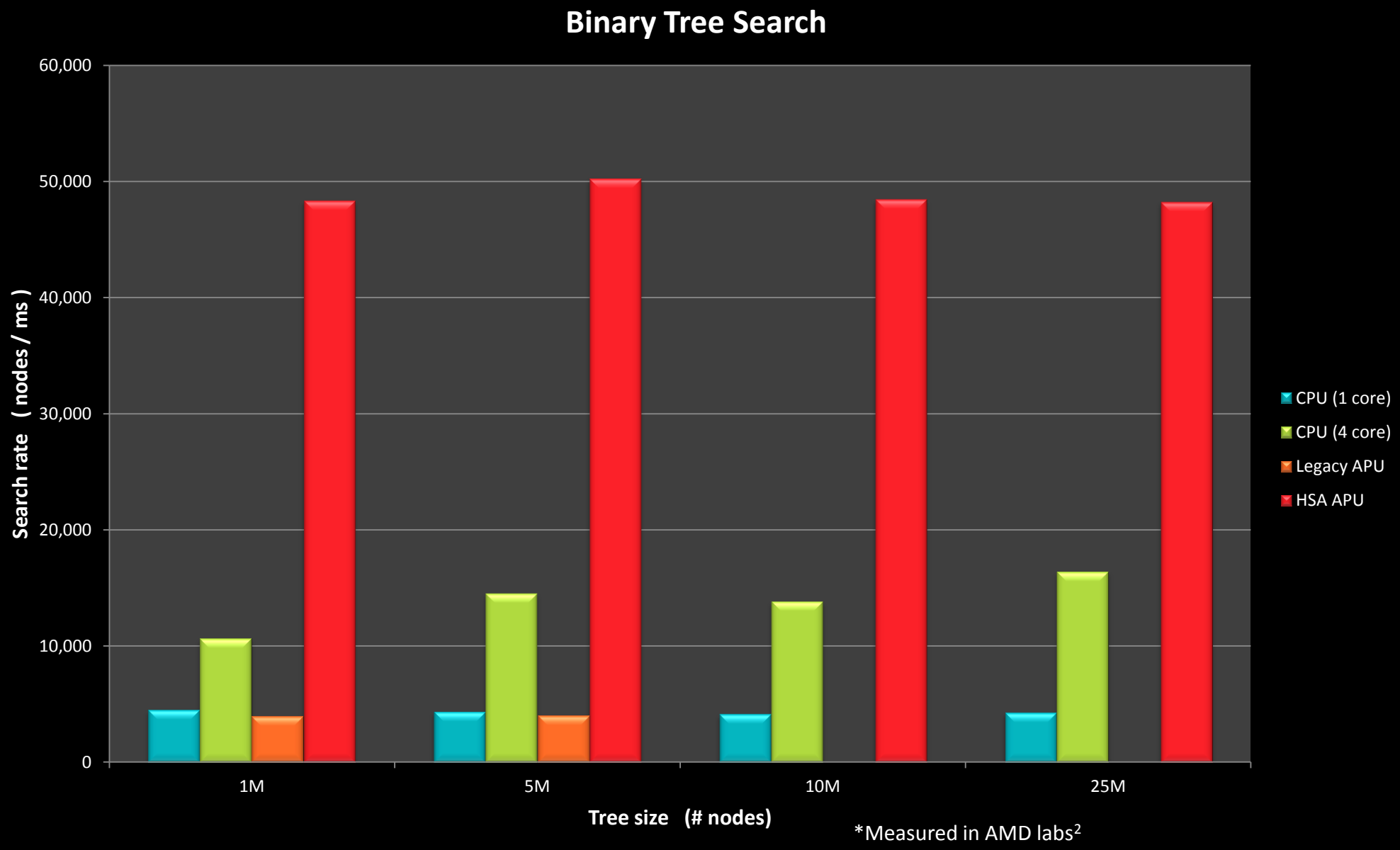
HSA

Legacy

```
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);
}
```

```
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



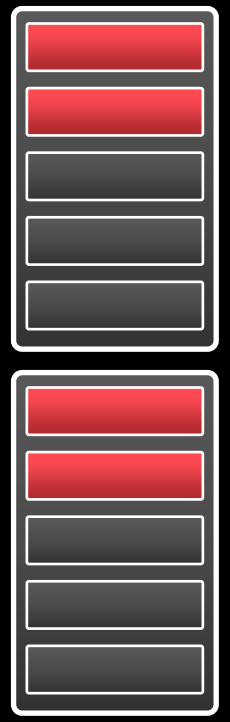
*Measured in AMD labs[2]

Platform Atomics

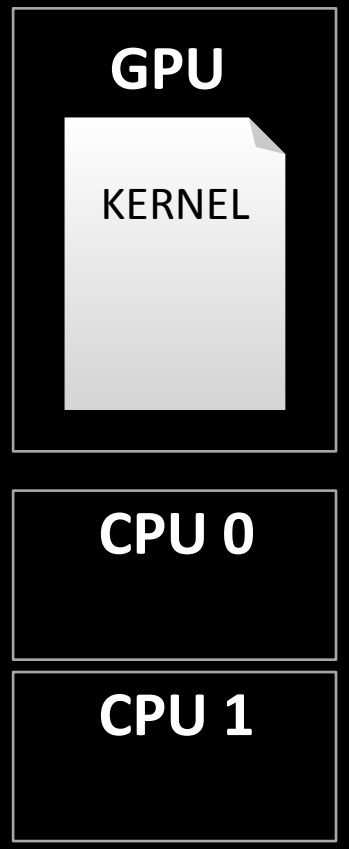
PLATFORM ATOMICS

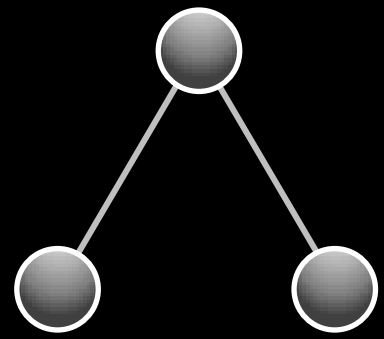
HSA and full OpenCL 2.0

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



INPUT BUFFER



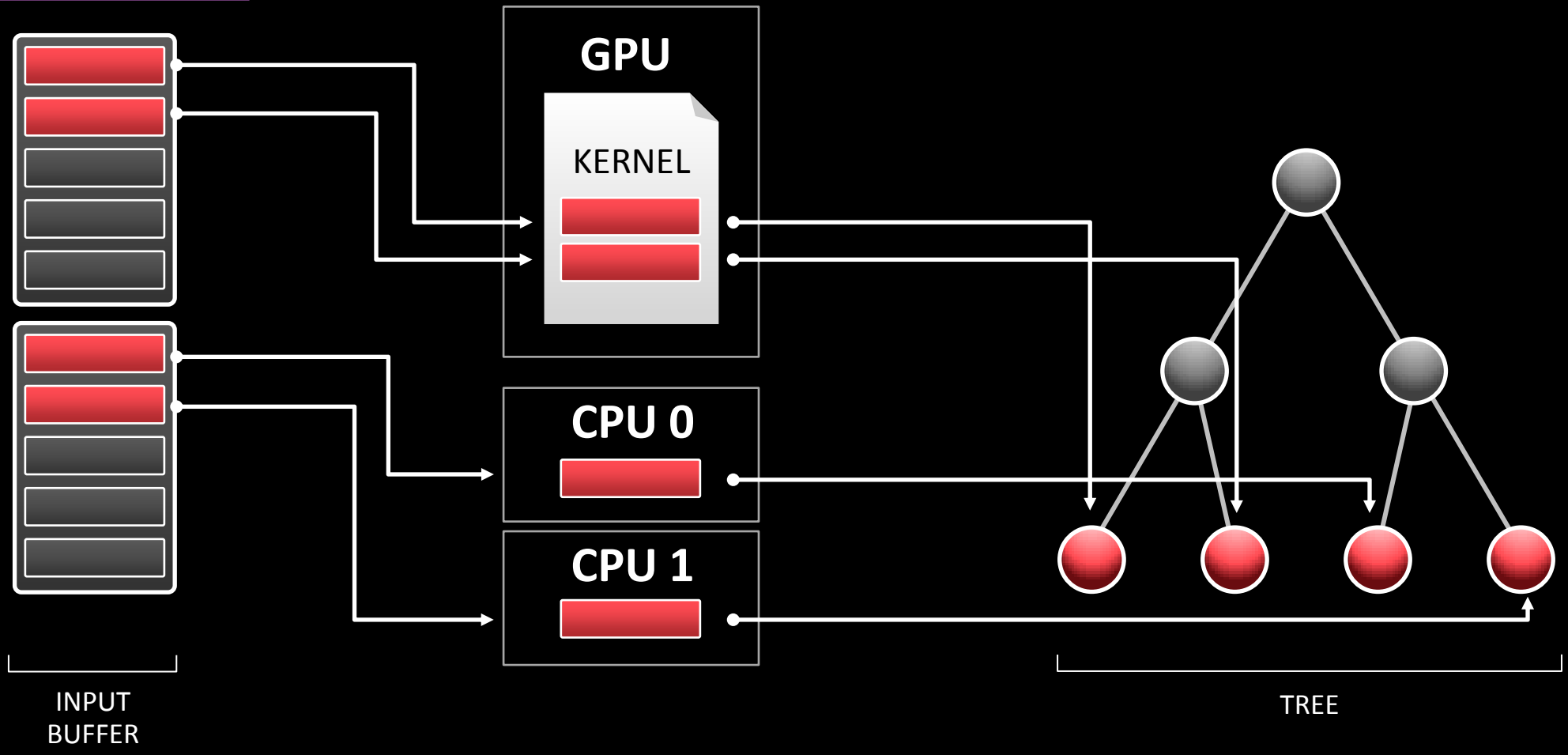


TREE

PLATFORM ATOMICS

HSA and full OpenCL 2.0

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



AMD'S UNIFIED SDK

- 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

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++

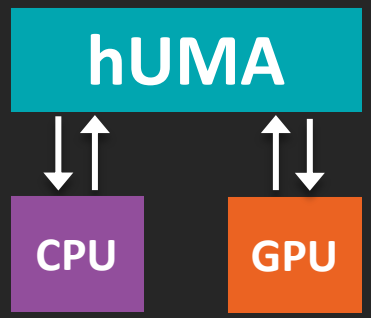
Aparapi

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

KAVERI OPENS THE GATES TO PERFORMANCE

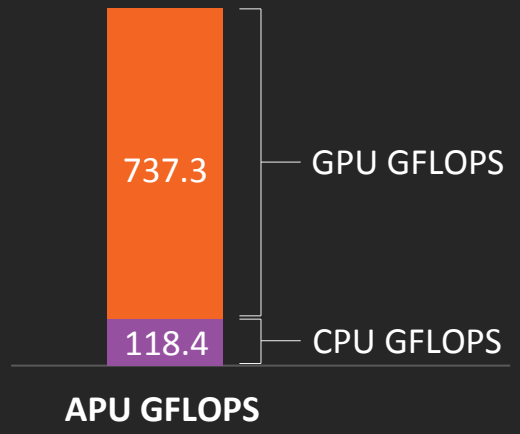
AMD

EQUAL ACCESS TO ENTIRE MEMORY



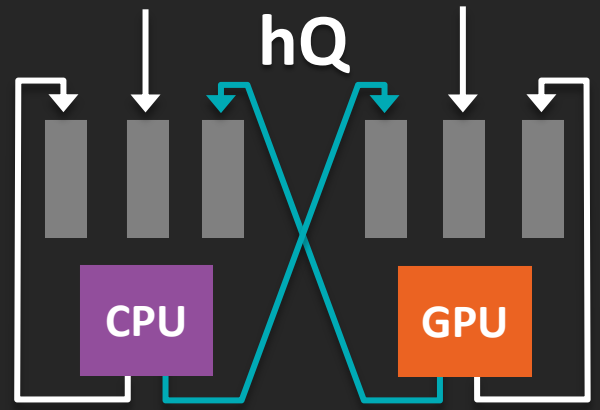
- GPU and CPU have uniform visibility into entire memory space

UNLOCKING APU GFLOPS



- Access to full potential of Kaveri APU compute power

ALL-PROCESSORS-EQUAL



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

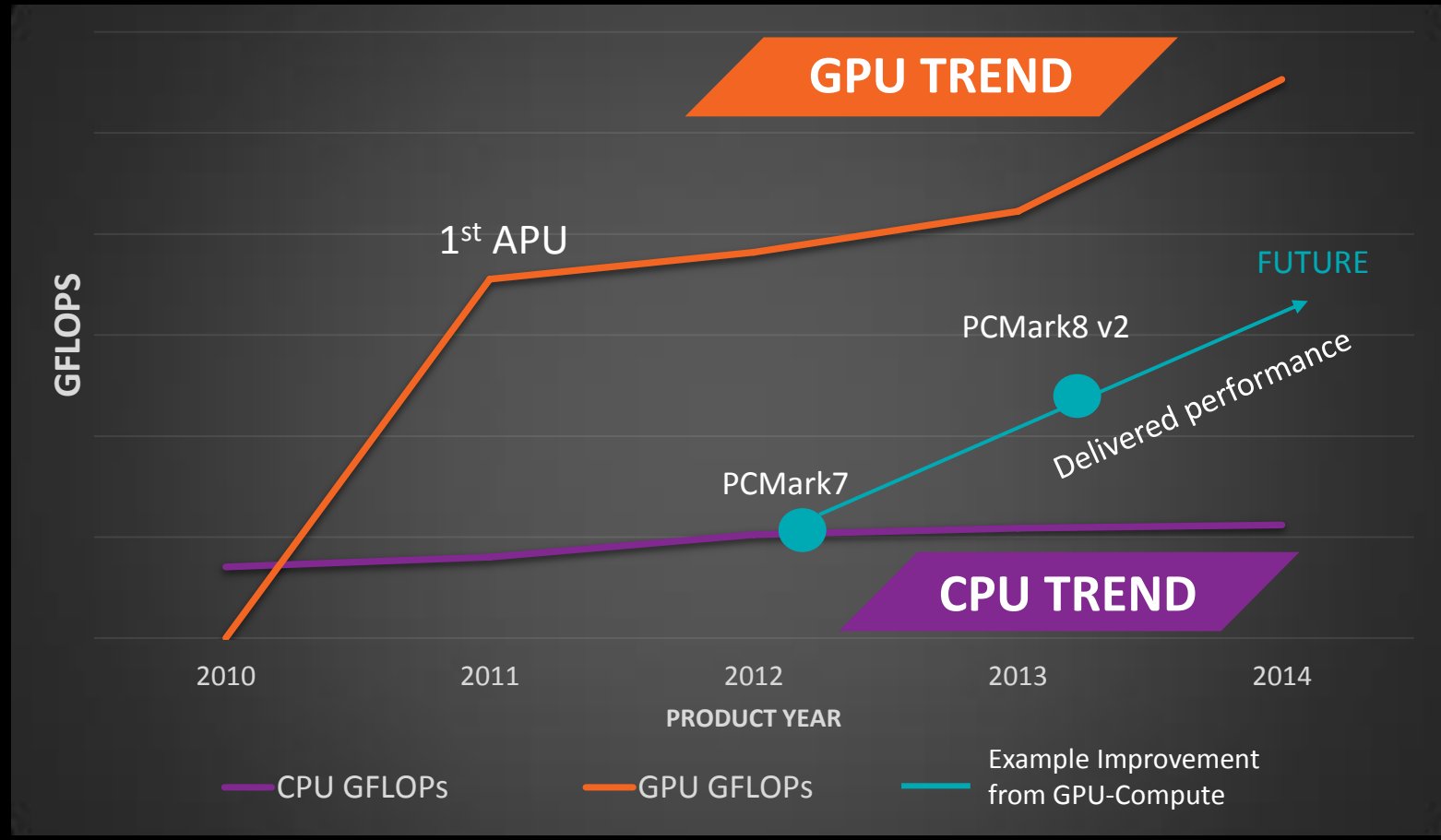
END RESULT: HSA RESULTS IN MORE ENERGY EFFICIENT COMPUTATION

AMD

What does this mean for power?

- Many important workloads execute many times more efficiently using GPU compute resources than CPU only
  - E.g. video indexing, natural human interfaces, pattern recognition
- For the same power, much better performance → finish early (computation, web page render, display update) and go to sleep

COMPUTE CAPACITY Trend in PCs



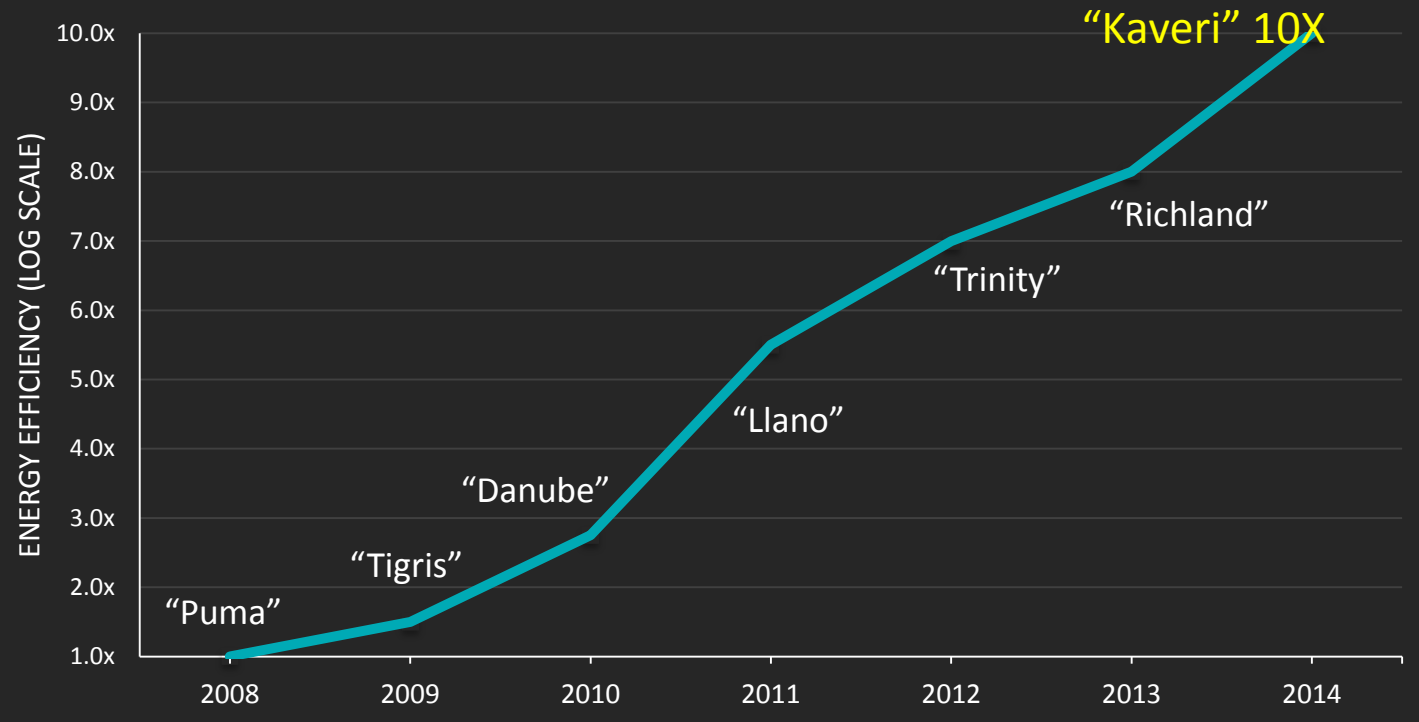
Source: AMD Internal data

AMD POWER EFFICIENCY IMPROVEMENTS

Power Reductions Over Time

- Over the last 6 years (2008-2014), AMD achieved a 10x improvement in platform energy efficiency*
- Enabled by:
  - Intelligent dynamic power management
  - Further integration of system components (NorthBridge, GPU, SouthBridge)
  - Silicon power optimizations
  - Process scaling improvements

Typical-Use Energy Efficiency

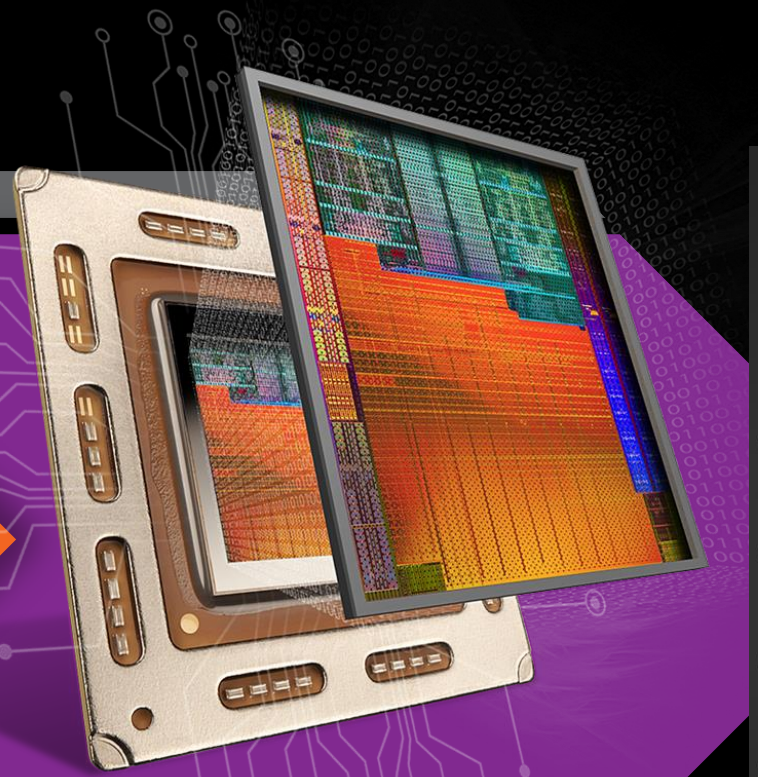


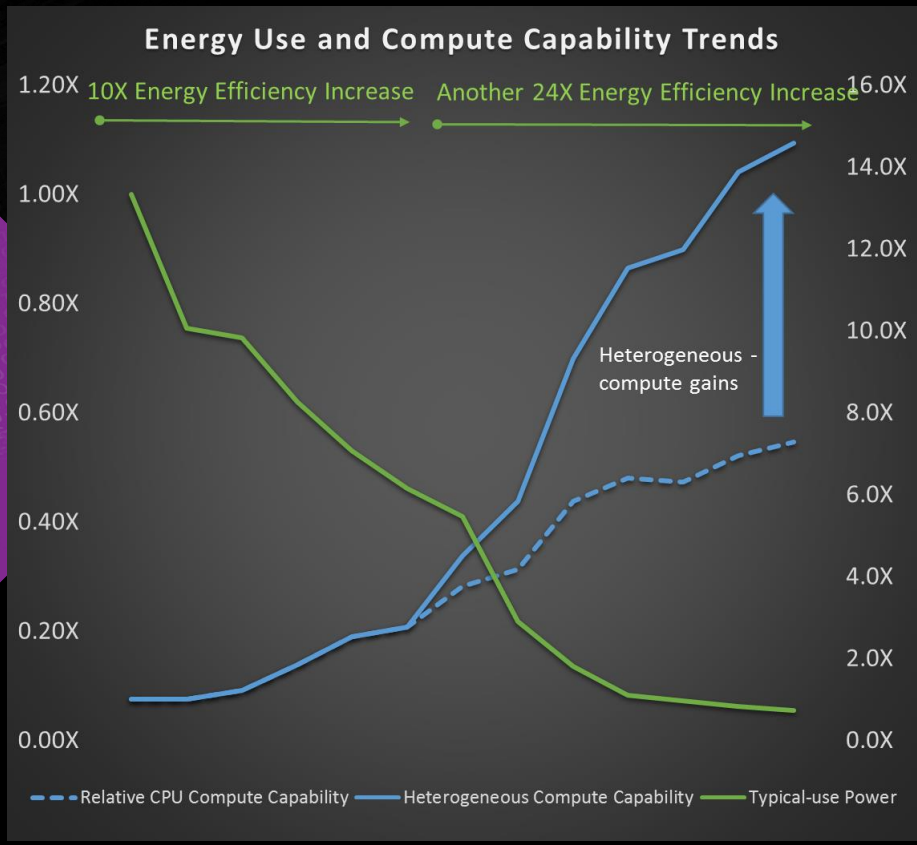
Energy use drops ⬇ While performance increases ⬆ = Increased efficiency ✚

*Typical-use Energy Efficiency as defined by taking the ratio of compute capability as measured by common performance measures such as SpecIntRate, PassMark and PCMark, divided by typical energy use as defined by E_{TEC} (Typical Energy Consumption for notebook computers) as specified in Energy Star Program Requirements Rev 6.0 10/2013

HSA A KEY ENABLER FOR AN ENERGY EFFICIENT ROADMAP

- Power efficient APUs
- Heterogeneous compute with energy efficient accelerators
- Smart power management
- Integration and miniaturization





SUMMARY

With HSA features, Kaveri is an optimized platform for Heterogeneous Computing

- HSA features make Kaveri the FIRST full OpenCL 2.0 capable chip
  - Fine Grained SVM (hUMA)
  - C11 Atomics (Platform Atomics)
  - Dynamic Parallelism (hQ)
  - Pipes (hUMA, hQ)

ENDNOTES

[1]Testing by AMD Performance labs using 3DMark Sky Diver as of July 3, 2014. AMD A10-7850K with 2x8GB DDR3-2133 memory, 512GB SSD, Windows 8.1, Driver 14.20.1004 Beta 11 scored 5523. AMD A10-6800K with 2x8GB DDR3-2133 memory, 512GB SSD, Windows 8.1, Driver 14.20.1004 Beta 11 scored 3796.

[2] System configuration “Kaveri” A10- 95W TDP, CPU Speed 3.7GHz/4.0 GHz, GPU speed 720MHz, Memory 2x4GB DDR3-1600, Disk HDD, Video Driver 13.35/HAS Beta 2.2, test dates January 1-3, 2014, Microsoft Windows 8.1 (64-bit)

DISCLAIMER & ATTRIBUTION

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.

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