



# HETEROGENEOUS SYSTEM ARCHITECTURE OVERVIEW

VINOD TIPPARAJU

PRINCIPAL MEMBER OF TECHNICAL STAFF,  
AMD AUSTIN.

# AGENDA

---

- ◆ Introduction (terminology)
- ◆ Memory Model & Queues
- ◆ HSAIL
- ◆ Architecture Details

# SOME TERMINOLOGY

---

- ◆ HSA is heterogeneous systems architecture, not just GPUs
- ◆ HSA Component – IP that satisfies architecture requirements and provides identified features
- ◆ SoC – system on Chip, collection of various IPs
  - ◆ E.g. AMD APU (Accelerated Processing Unit) integrates AMD/ARM CPU cores and Graphics IP
  - ◆ It is possible to conceive companies just building parts of the IP
- ◆ HSAIL -- HSA intermediate language very low-level SIMT language
- ◆ HSA Agent – something that can participate in the HSA memory subsystem (i.e. respect page sizes, memory properties, atomics, etc.)

# WHAT IS HSA?

---



## Systems Architecture

- ◆ From a hardware point of view, system architecture requirements necessary
- ◆ Specifies shared memory, cache coherence domains, concept of clocks, context switching, memory based signaling, topology,
- ◆ Rules governing design and agent behavior

## RUNTIME

- ◆ API that wraps the features like user mode queues, clocks, signalling, etc
- ◆ Provides execution control
- ◆ Supports tools

## Programmers Reference (HSAIL)

- ◆ An intermediate representation, very low level.
- ◆ Vendor independence, device compiler optimizations
- ◆ Abstracts HW, or can serve as the lowest level instruction set

## TOOLS

- ◆ Supporting profilers, debuggers and compilers
- ◆ Unique debugging support that greatly simplifies implementing debuggers
- ◆ Excellent profiling support with some user mode access

# TAKING THE HW INTEGRATION TO ITS NATURAL CONCLUSION

---



- ◆ Architectural and System integration
- ◆ Extend architecture to make the component a first class citizen on the SoC
- ◆ Fully-evolved MMU
- ◆ Provide same level of support for tools as CPU
- ◆ Provide context switching, preemption, full-coherence
  - ◆ Helps simulators, migrations, checkpoints, etc
- ◆ Future, other HSA IP

# HIGH LEVEL USAGE SCENARIOS

---

- ◆ Bulk-Synchronous Parallelism -like concurrent computation
  - ◆ Rather large parallel sections followed by synchronization
- ◆ Outstanding support for task-based parallelism
  - ◆ Wavefront is 64 threads
  - ◆ 256 threads sufficient to fully fill the pipeline
  - ◆ Launch in under a microsecond (same with nested launches)
- ◆ Support for execution schedules – excellent compiler target
  - ◆ Architected Queueing Language (AQL), dependencies
- ◆ Advanced language support
  - ◆ Function calls
  - ◆ Virtual functions
  - ◆ Exception handling (throw-catch)

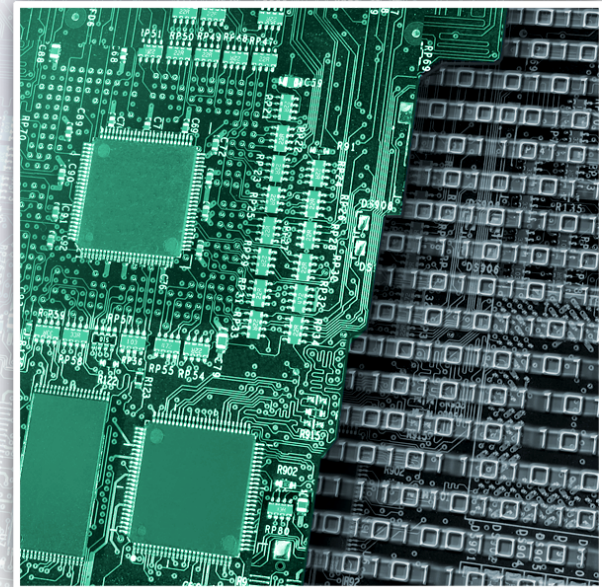


HSA<sup>™</sup>  
FOUNDATION

# MEMORY AND QUEUING MODEL

# HSA MEMORY MODEL

- ◆ Defines visibility ordering between all threads in the HSA System
- ◆ Designed to be compatible with C++11, Java, OpenCL and .NET Memory Models
- ◆ Relaxed consistency memory model for parallel compute performance
- ◆ Visibility controlled by:
  - ◆ Load.Acquire
  - ◆ Store.Release
  - ◆ Barriers





# HSA QUEUING MODEL

---

- ◆ User mode queuing for low latency dispatch
  - ◆ Application dispatches directly
  - ◆ No OS or driver in the dispatch path
- ◆ Architected Queuing Layer
  - ◆ Single compute dispatch path for all hardware
  - ◆ No driver translation, direct to hardware
- ◆ Allows for dispatch to queue from any agent
  - ◆ CPU or GPU
- ◆ GPU self enqueue enables lots of solutions
  - ◆ Recursion
  - ◆ Tree traversal
  - ◆ Wavefront reforming



HSA<sup>™</sup>  
FOUNDATION

HSAIL

# HSA INTERMEDIATE LAYER — HSAIL

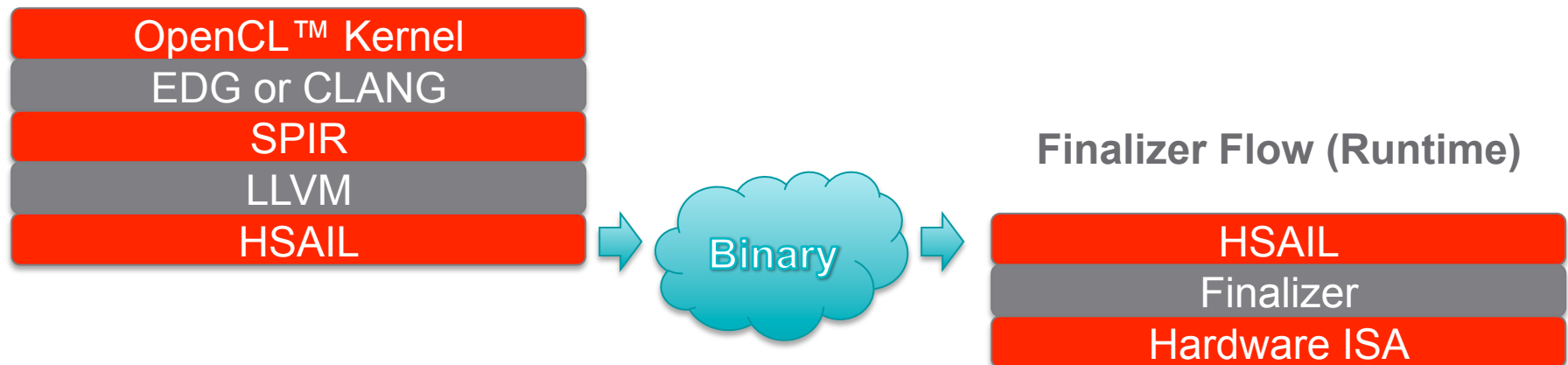
- ◆ HSAIL is a virtual ISA for parallel programs
  - ◆ Finalized to ISA by a JIT compiler or “Finalizer”
  - ◆ ISA independent by design for CPU & GPU
- ◆ Explicitly parallel
  - ◆ Designed for data parallel programming
- ◆ Support for exceptions, virtual functions, and other high level language features
- ◆ Lower level than OpenCL SPIR
  - ◆ Fits naturally in the OpenCL compilation stack
- ◆ Suitable to support additional high level languages and programming models:
  - ◆ Java, C++, OpenMP, Fortran etc



# WHAT IS HSAIL?

- ◆ HSAIL is the intermediate language for parallel compute in HSA
  - ◆ Generated by a high level compiler (LLVM, gcc, Java VM, etc)
  - ◆ Low-level IR, close to machine ISA level
  - ◆ Compiled down to target ISA by an IHV “Finalizer”
  - ◆ Finalizer may execute at run time, install time, or build time
- ◆ Example: OpenCL™ Compilation Stack using HSAIL

## High-Level Compiler Flow (Developer)



# KEY HSAIL FEATURES

---

- ◆ Parallel
- ◆ Shared virtual memory
- ◆ Portable across vendors in HSA Foundation
- ◆ Stable across multiple product generations
- ◆ Consistent numerical results (IEEE-754 with defined min accuracy)
- ◆ Fast, robust, simple finalization step (no monthly updates)
- ◆ Good performance (little need to write in ISA)
- ◆ Supports all of OpenCL<sup>™</sup> and C++ AMP<sup>™</sup>
- ◆ Support Java, C++, and other languages as well

# SIMT EXECUTION MODEL

---

- ◆ HSAIL Presents a “SIMT” execution model to the programmer
  - ◆ “Single Instruction, Multiple Thread”
  - ◆ Programmer writes program for a single thread of execution
  - ◆ Each work-item appears to have its own program counter
  - ◆ Branch instructions look natural
- ◆ Hardware Implementation
  - ◆ Most hardware uses SIMD (Single-Instruction Multiple Data) vectors for efficiency
  - ◆ Actually one program counter for the entire SIMD instruction
  - ◆ Branches implemented with predication
- ◆ SIMT Advantages
  - ◆ Easier to program (branch code in particular)
  - ◆ Natural path for mainstream programming models
  - ◆ Scales across a wide variety of hardware (programmer doesn’t see vector width)
  - ◆ Cross-lane operations available for those who want peak performance



HSA<sup>TM</sup>  
FOUNDATION

# ARCHITECTURE DETAILS – WALK THROUGH OF FEATURES AND BENEFITS

# HIGH LEVEL FEATURES OF HSA

---

- ◆ Features currently being defined in the HSA Working Groups\*\*
  - ◆ Unified addressing across all processors
  - ◆ Operation into pageable system memory
  - ◆ Full memory coherency
  - ◆ User mode dispatch
  - ◆ Architected queuing language
  - ◆ High level language support for GPU compute processors
  - ◆ Preemption and context switching

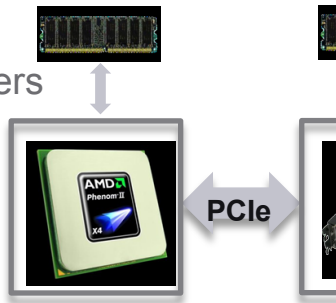
\*\* All features subject to change, pending completion and ratification of specifications in the HSA Working Groups



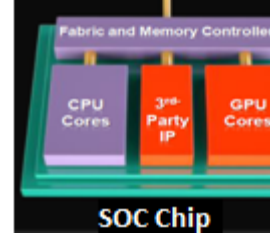
# STATE OF GPU COMPUTING

- GPUs are fast and power efficient : high compute density per-mm and per-watt
- But: Can be hard to program

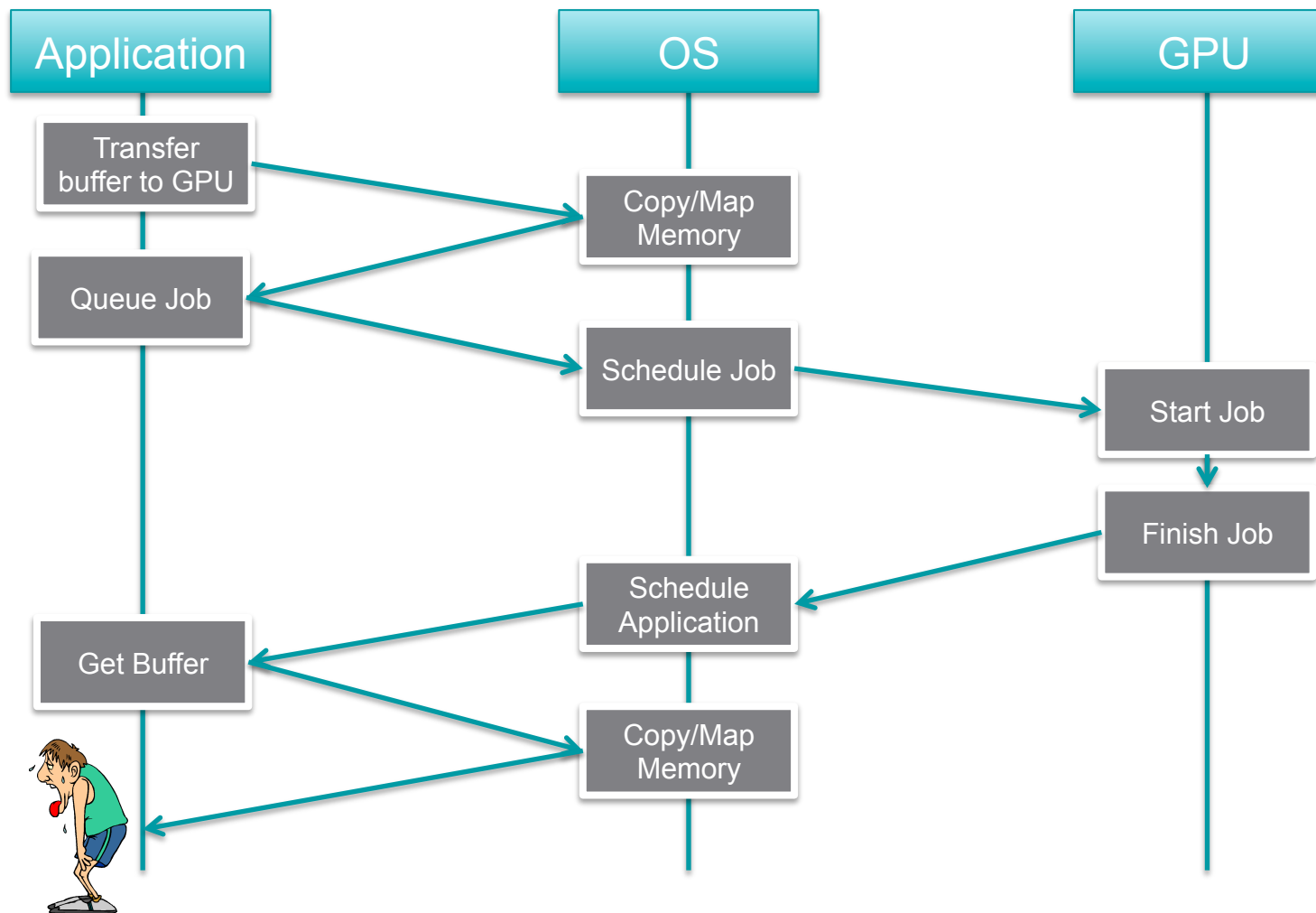
## Today's Challenges

- ◆ Separate address spaces
    - ◆ Copies
    - ◆ Can't share pointers
- 
- ◆ New language required for compute kernel
    - ◆ EX: OpenCL™ runtime API
    - ◆ Compute kernel compiled separately than host code

## Emerging Solution

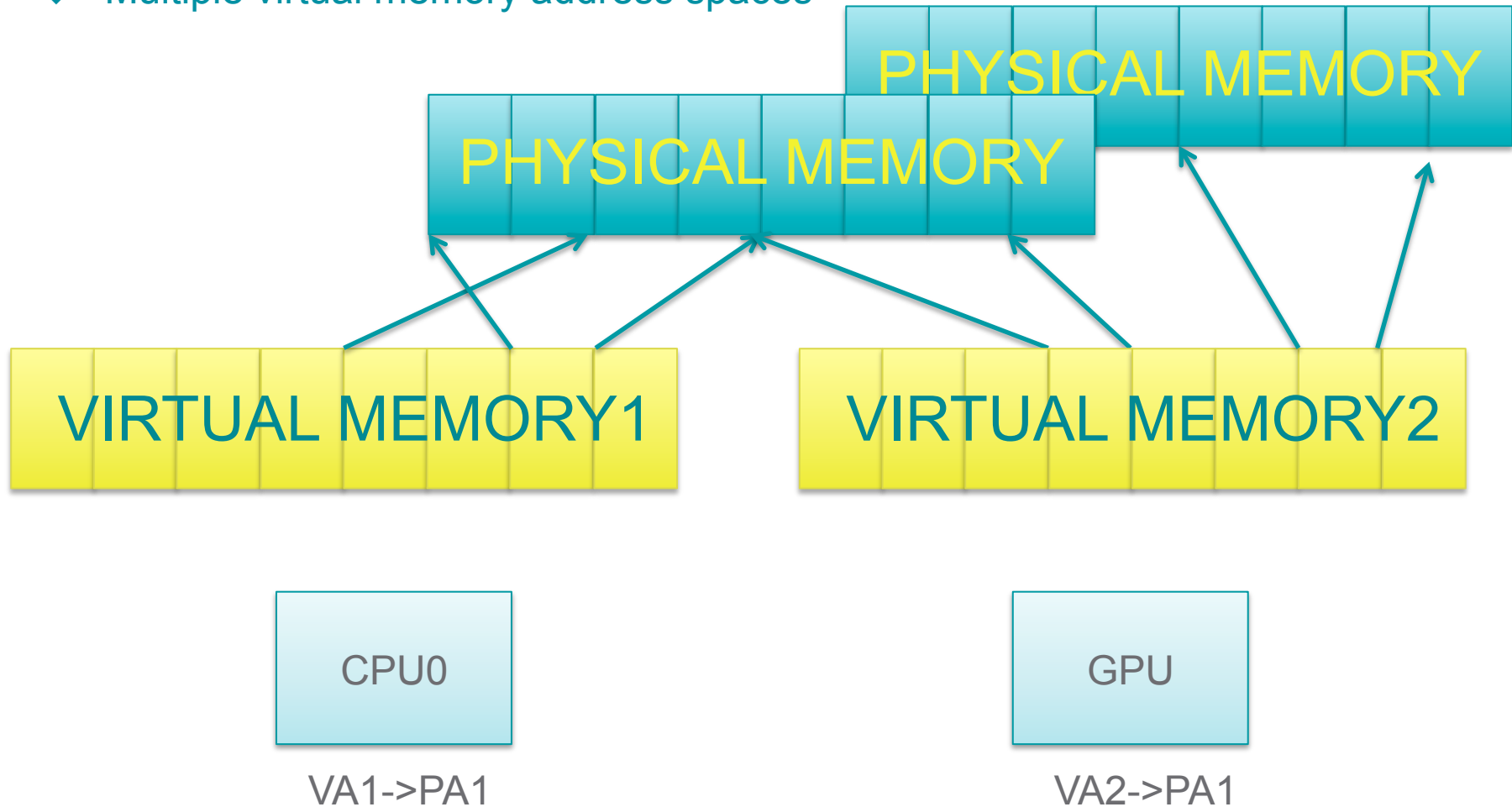
- ◆ HSA Hardware
    - ◆ Single address space
    - ◆ Coherent
    - ◆ Virtual address space
    - ◆ Fast access from all components
    - ◆ Can share pointers
- 
- ◆ Bring GPU computing to existing, popular, programming models
    - ◆ Single-source, fully supported by compiler
    - ◆ HSAIL compiler IR (Cross-platform!)

# MOTIVATION (TODAY'S PICTURE)



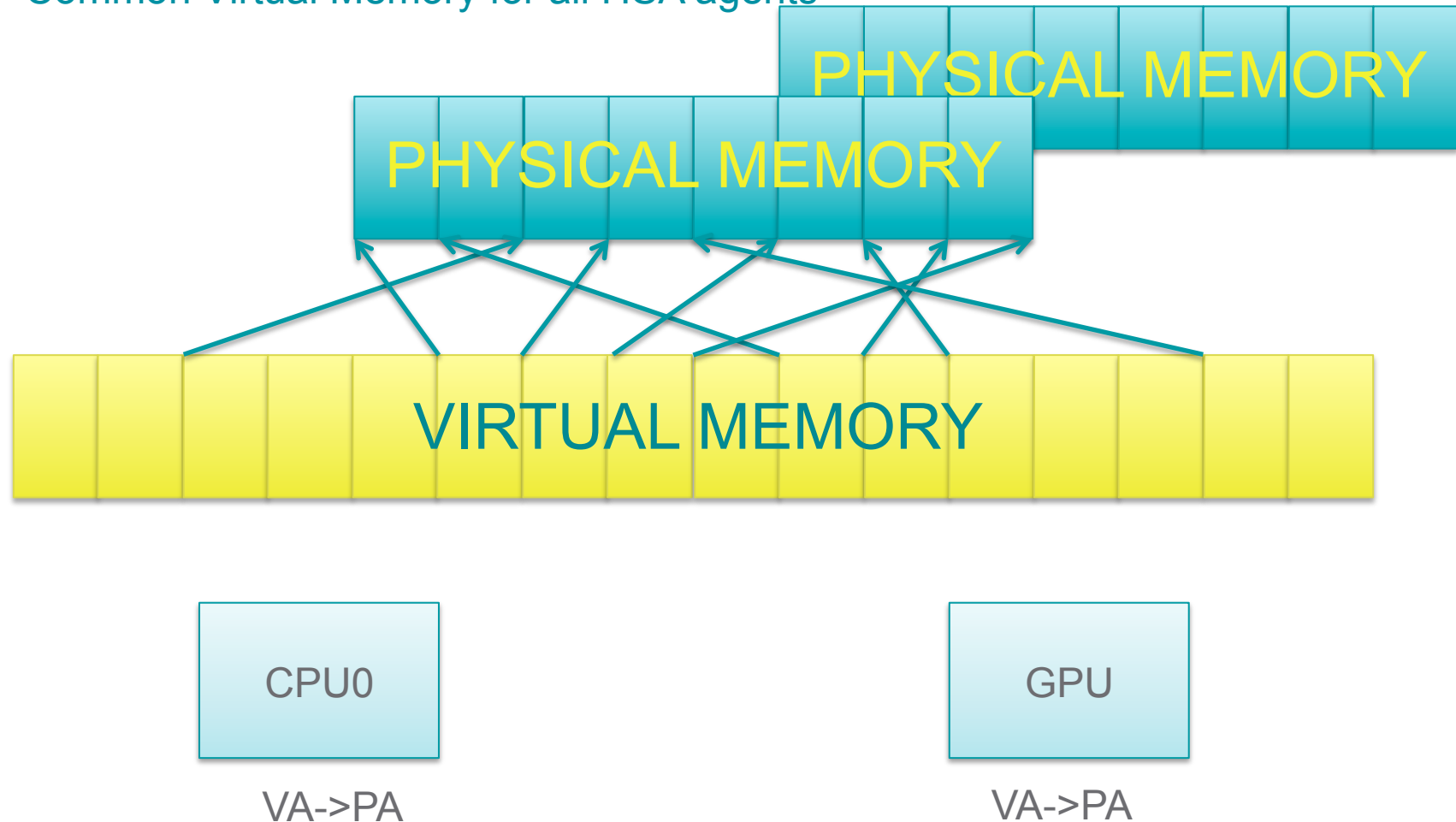
# SHARED VIRTUAL MEMORY (TODAY)

- ◆ Multiple virtual memory address spaces



# SHARED VIRTUAL MEMORY (HSA)

- ◆ Common Virtual Memory for all HSA agents



# SHARED VIRTUAL MEMORY

---

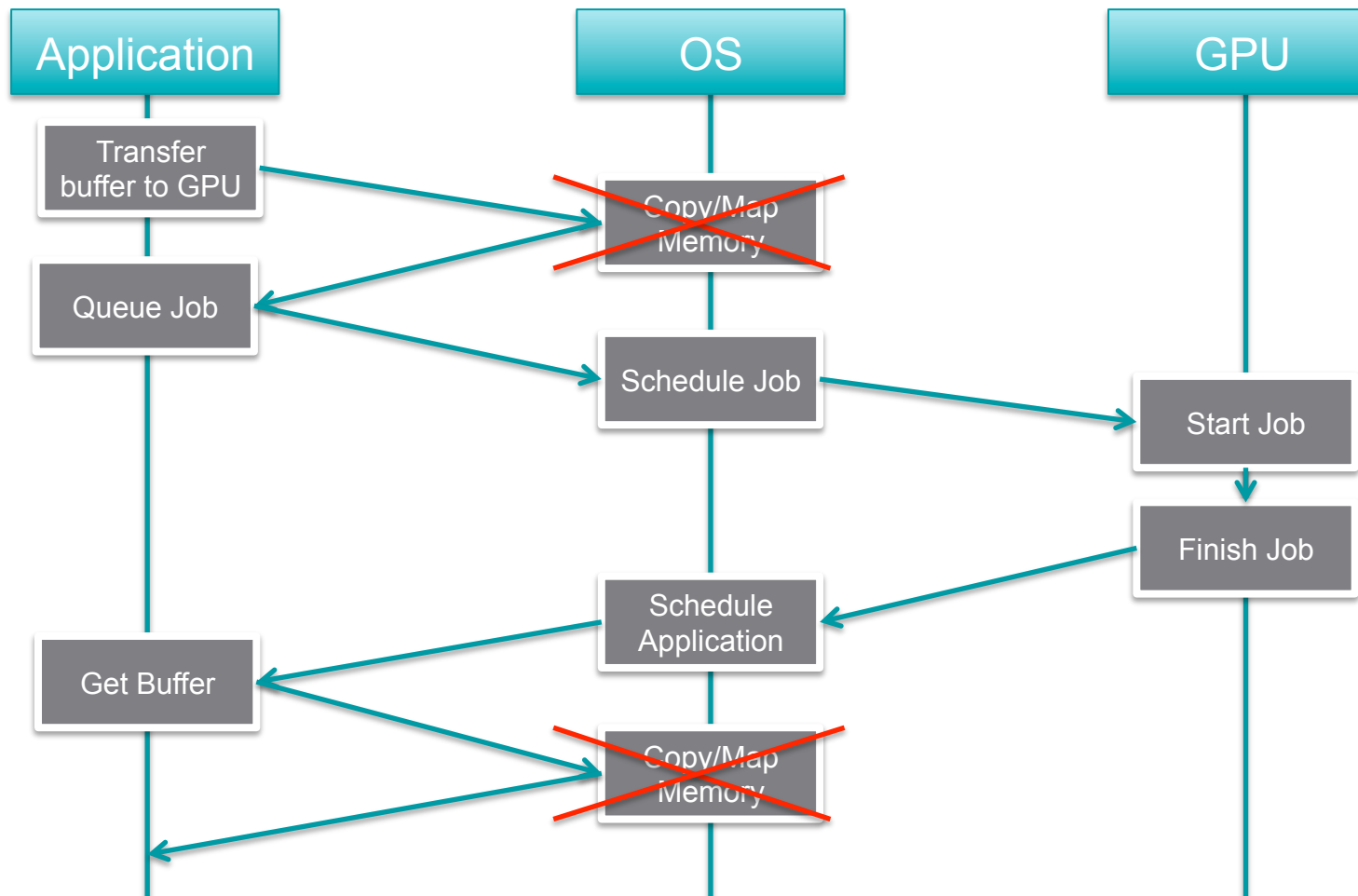
## ◆ Advantages

- ◆ No mapping tricks, no copying back-and-forth between different PA addresses
- ◆ Send pointers (not data) back and forth between HSA agents.

## ◆ Implications

- ◆ Common Page Tables (and common interpretation of architectural semantics such as shareability, protection, etc).
- ◆ Common mechanisms for address translation (and servicing address translation faults)
- ◆ Concept of a process address space ID (PASID) to allow multiple, per process virtual address spaces within the system.

# GETTING THERE ...



# SHARED VIRTUAL MEMORY

---

## ◆ Specifics

- ◆ Minimum supported VA width is 48b for 64b systems, and 32b for 32b systems.
- ◆ HSA agents may reserve VA ranges for internal use via system software.
- ◆ All HSA agents other than the host unit must use the lowest privilege level
- ◆ If present, read/write access flags for page tables must be maintained by all agents.
- ◆ Read/write permissions apply to all HSA agents, equally.



HSA<sup>™</sup>  
FOUNDATION

# CACHE COHERENCY



# CACHE COHERENCY DOMAINS (1/2)

---

- ◆ *Data accesses to global memory segment from all HSA Agents shall be coherent without the need for explicit cache maintenance.*

# CACHE COHERENCY DOMAINS (2/2)

---

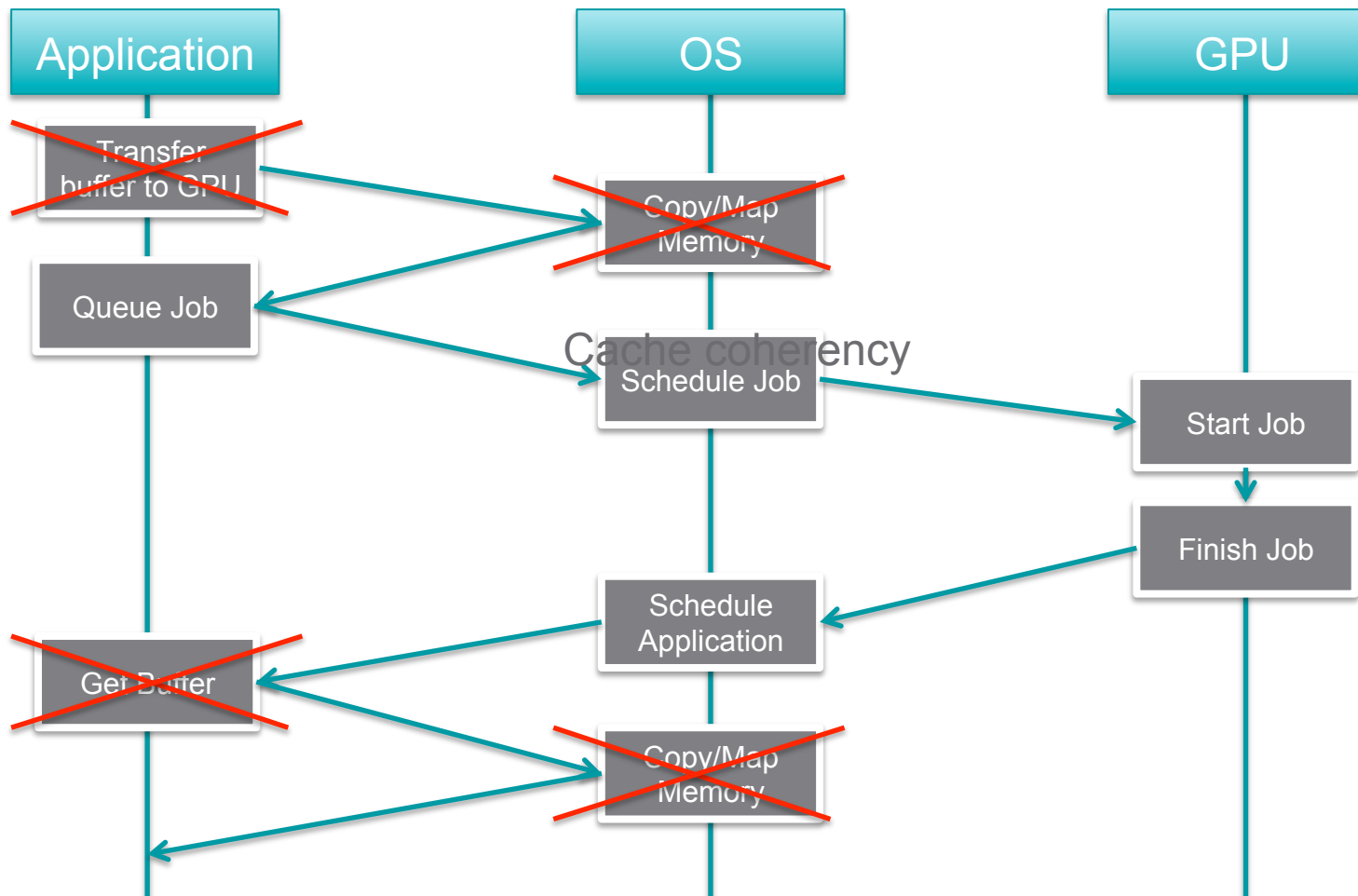
## ◆ Advantages

- ◆ Composability
- ◆ Reduced SW complexity when communicating between agents
- ◆ Lower barrier to entry when porting software

## ◆ Implications

- ◆ Hardware coherency support between all HSA agents
- ◆ Can take many forms
  - ◆ Stand alone Snoop Filters / Directories
  - ◆ Combined L3/Filters
  - ◆ Snoop-based systems (no filter)
  - ◆ Etc ...

# GETTING CLOSER ...





HSA<sup>™</sup>  
FOUNDATION

# SIGNALING

## SIGNALING (1/2)

---

- ◆ HSA agents support the ability to use signaling objects
  - ◆ All creation/destruction signaling objects occurs via HSA runtime APIs
    - ◆ Object creation/destruction
  - ◆ From an HSA Agent you can directly accessing signaling objects.
    - ◆ Signaling a signal object (this will wake up HSA agents waiting upon the object)
    - ◆ Query current object
    - ◆ Wait on the current object (various conditions supported).

# SIGNALING (2/2)

---

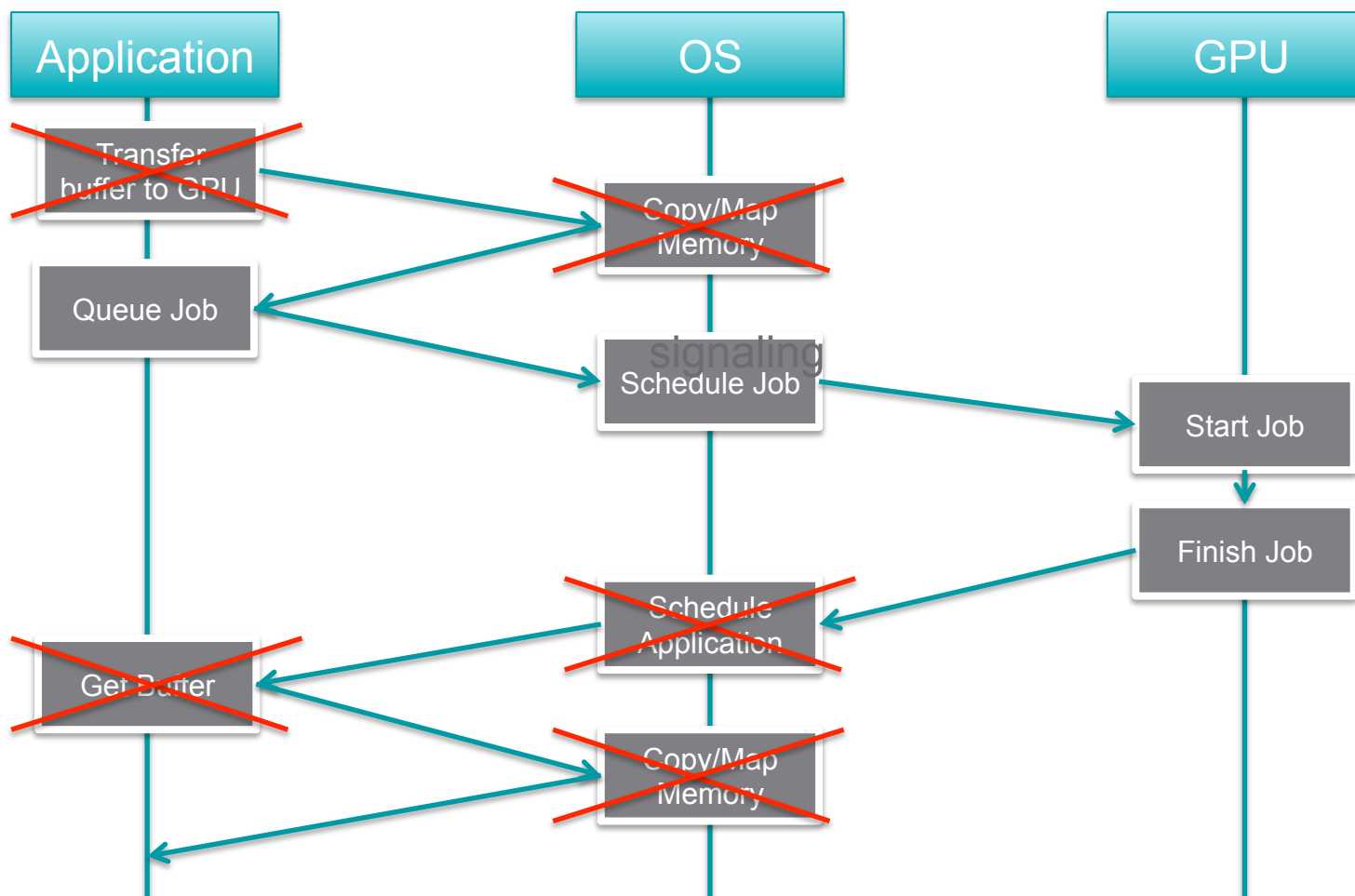
## ◆ Advantages

- ◆ Enables asynchronous interrupts between HSA agents, without involving the kernel
- ◆ Common idiom for work offload
- ◆ Low power waiting

## ◆ Implications

- ◆ Runtime support required
- ◆ Commonly implemented on top of cache coherency flows

# ALMOST THERE...





HSA<sup>™</sup>  
FOUNDATION

# USER MODE QUEUEING



# USER MODE QUEUEING (1/3)

---

## ◆ User mode Queueing

- ◆ Enables user space applications to directly, without OS intervention, enqueue jobs (“Dispatch Packets”) for HSA agents.
  - ◆ Dispatch packet is a job of work
- ◆ Support for multiple queues per PASID
- ◆ Multiple threads/agents within a PASID may enqueue Packets in the same Queue.
- ◆ Dependency mechanisms created for ensuring ordering between packets.

# USER MODE QUEUEING (2/3)

---

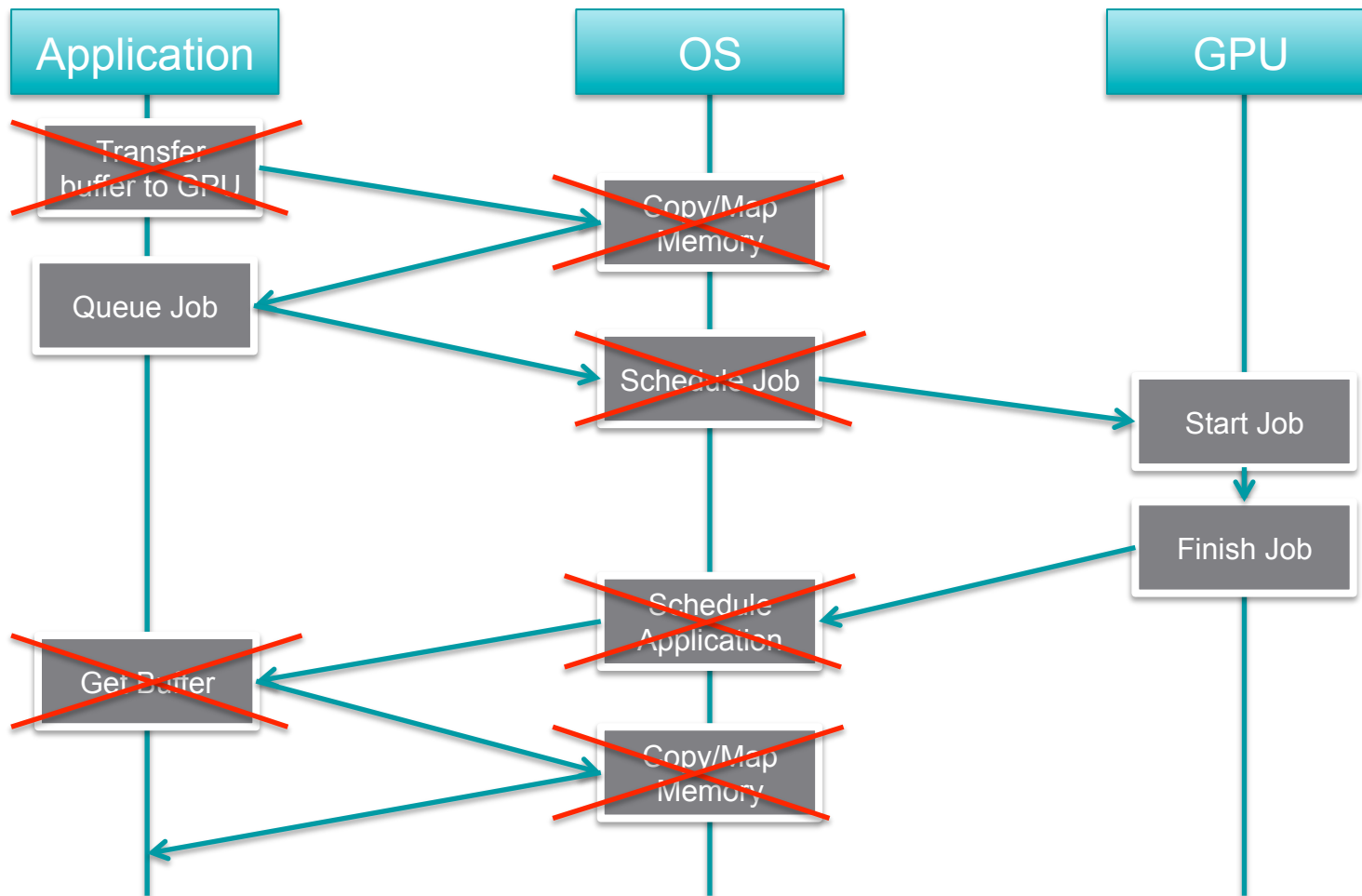
## ◆ Advantages

- ◆ Avoid involving the kernel/driver when dispatching work for an Agent.
- ◆ Lower latency job dispatch enables finer granularity of offload
- ◆ Standard memory protection mechanisms may be used to protect communication with the consuming agent.

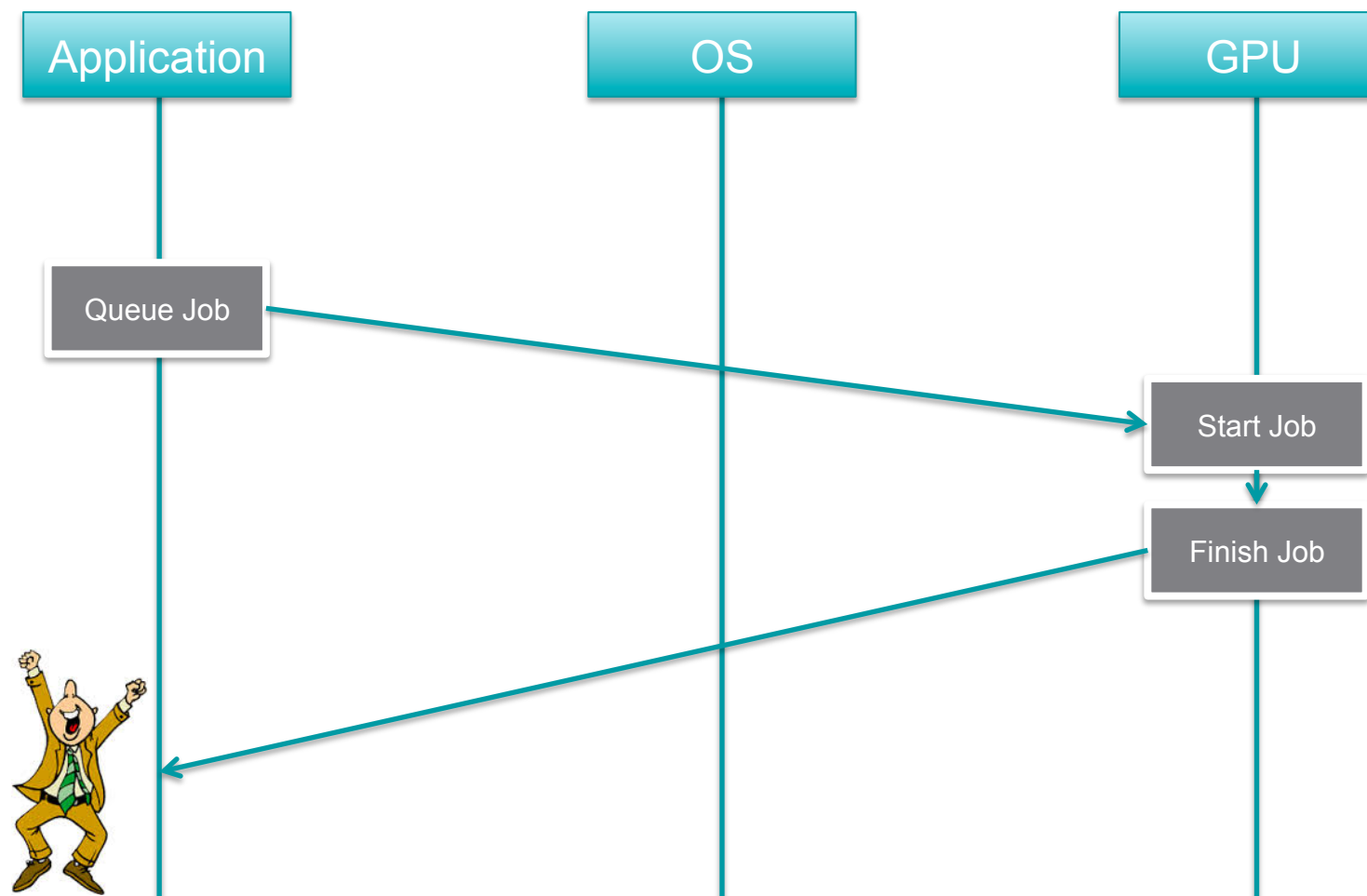
## ◆ Implications

- ◆ Packet formats/fields are *Architected* – standard across vendors!
  - ◆ Guaranteed backward compatibility
- ◆ Packets are enqueued/dequeued via an Architected protocol (all via memory accesses and signalling)
- ◆ More on this later.....

# SUCCESS!



# SUCCESS!





HSA<sup>™</sup>  
FOUNDATION

# ACCELERATING SUFFIX ARRAY CONSTRUCTION

CLOUD SERVER WORKLOAD

# SUFFIX ARRAYS

---

- ◆ Suffix Arrays are a fundamental data structure
  - ◆ Designed for efficient searching of a large text
    - ◆ Quickly locate every occurrence of a substring  $S$  in a text  $\mathcal{T}$
- ◆ Suffix Arrays are used to accelerate in-memory cloud workloads
  - ◆ Full text index search
  - ◆ Lossless data compression
  - ◆ Bio-informatics

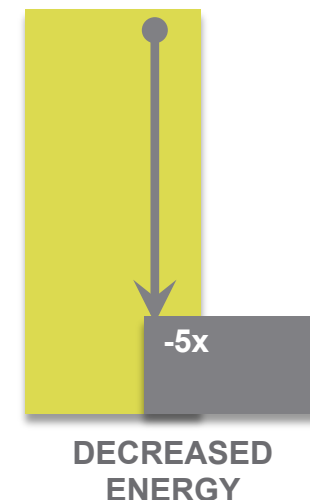
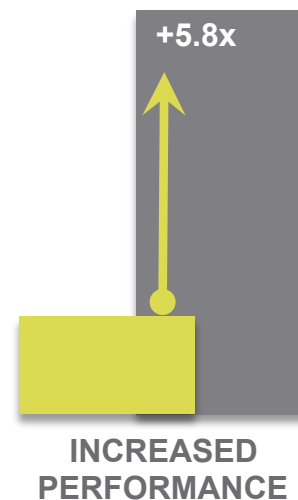
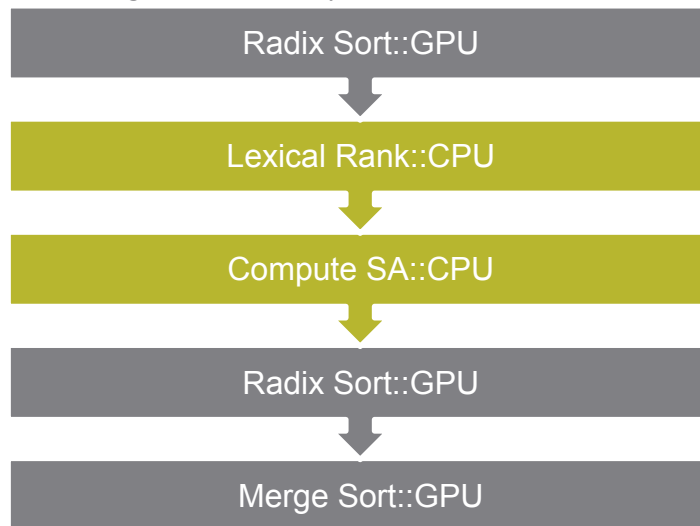
# ACCELERATED SUFFIX ARRAY CONSTRUCTION ON HSA



By efficiently sharing data between CPU and GPU, HSA lets us move compute to data without penalty of intermediate copies.

By offloading data parallel computations to GPU, HSA increases performance and reduces energy for Suffix Array Construction versus Single Threaded CPU.

*Skew Algorithm for Compute SA*



M. Deo, "Parallel Suffix Array Construction and Least Common Prefix for the GPU", Submitted to "Principles and Practice of Parallel Programming, (PPoPP'13)" February 2013. AMD A10 4600M APU with Radeon™ HD Graphics; CPU: 4 cores @ 2.3 MHz (turbo 3.2 GHz); GPU: AMD Radeon HD 7660G, 6 compute units, 685MHz; 4GB RAM

# THE HSA FUTURE

---

Architected heterogeneous processing on the SOC

Programming of accelerators becomes **much easier**

Accelerated software that runs across multiple hardware vendors

Scalability from smart phones to super computers on a common architecture

GPU acceleration of parallel processing is the initial target, with DSPs and other accelerators coming to the HSA system architecture model

Heterogeneous software ecosystem evolves at a much faster pace

Lower power, more capable devices in your hand, on the wall or in a super computer.



# BACKUP

# SEGMENTS

