PROGRAMMING HETEROGENEOUS SYSTEMS USING HSA

JOHN GLOSSNER, PH.D.

PRESIDENT, HSA FOUNDATION / CEO, GPT

AGENDA

About HSA

Heterogeneous Programming Problem

Programming HSAF Systems – Top Down View

Applications Programming

- Tools / Libraries
- Languages
  - OpenCL, C++17, Cuda, Python

Runtime

- Initiating an HSA Application

System Architecture

- Queues
- Signals

HSAIL

- Portable Heterogeneous Virtual ISA
- Hardware Requirements

Performance Results

- On HSA Hardware

Conclusions

ABOUT HSA

HETEROGENEOUS SYSTEM ARCHITECTURE FOUNDATION

HSA FOUNDATION

Founded in June 2012

Developing a new platform for heterogeneous systems

- Working groups established to define the platform

Specifications

- V1.0 Specifications March 2015
- V1.1 Specifications March 2016
- V1.2 in progress

First compatible hardware

- 4Q 2015 Carrizo
- 4Q 2016 Bristol Ridge
  - 1Q 2017 HP Pavilion laptop

www.hsafoundation.com

HSA – AN OPEN PLATFORM

Publicly Available Specifications

- HSA Programmers Reference Manual
- HSA Platform System Architecture
- HSA Runtime
- HSA Multivendor Specification

Membership Open to Everyone

Royalty Free

- IP, Specifications, and APIs

Open Source

- Tools, Compilers, etc.
- Runtime implementations
- Tests

Abstracting Heterogeneous Programming

- ISA Agnostic
- CPU, GPU, DSP, FPGA, etc.
- Virtual ISA

THE PROBLEM

HETEROGENEOUS APPLICATION DEVELOPMENT

WHAT'S THE PROBLEM?

Heterogeneous processors are widely available

Huge compute capability

- Acceleration Units (GPU, DSP, FPGA)
- CPU Cluster-based computer

Coherency

- Established in high-end
- Migrating to mainstream mobile and consumer

BUT…

Heterogeneous programming models not standardized

Multi-core/device applications difficult to optimize or scale

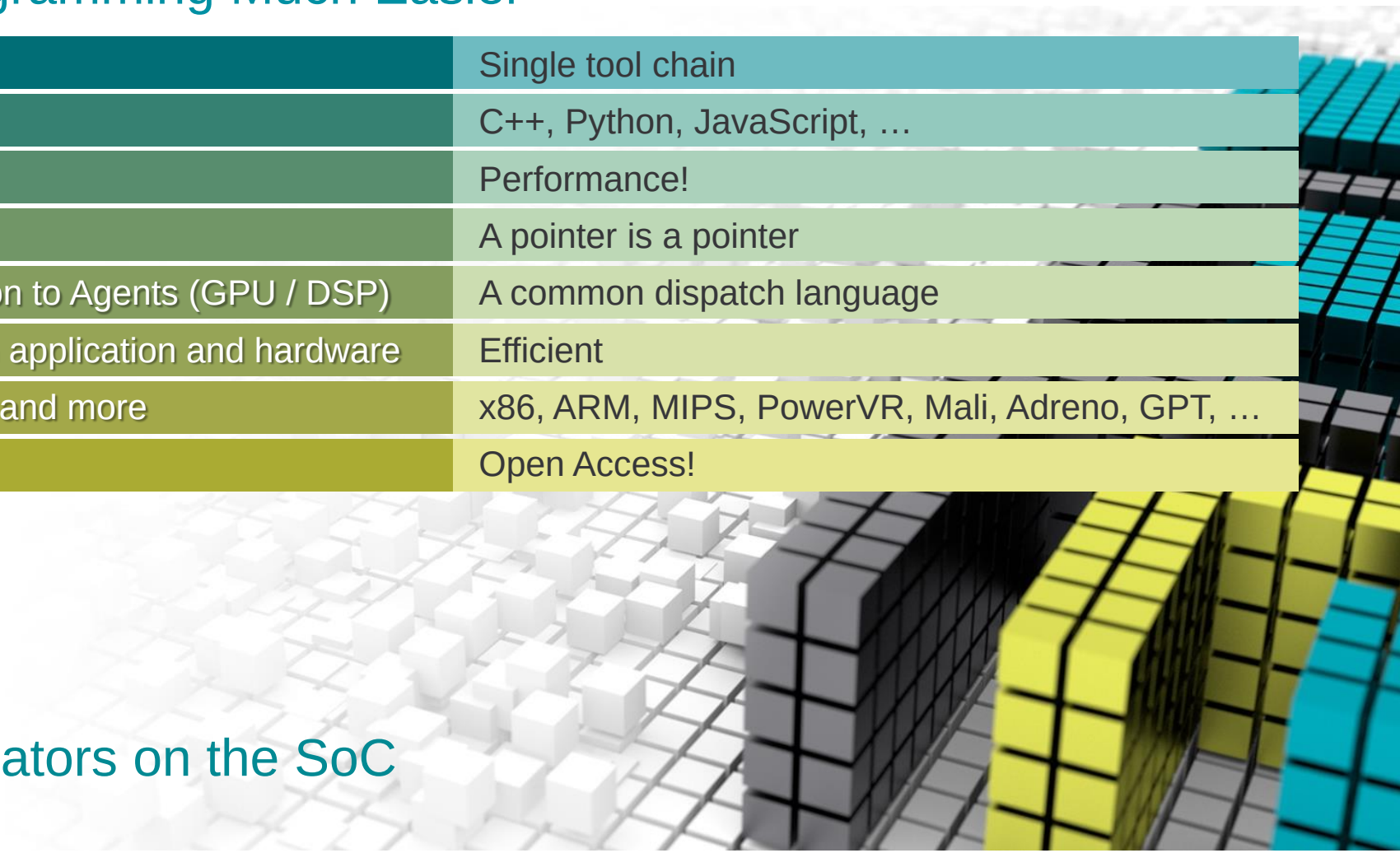
Non-portable application developer ecosystems

HSAF brings compute app abstraction to heterogeneous platforms

THE VISION

Make Heterogeneous Programming Much Easier

1	Single source programming	Single tool chain
2	Any programming language	C++, Python, JavaScript, …
3	Eliminate data copies	Performance!
4	Common address space	A pointer is a pointer
5	Standardized command submission to Agents (GPU / DSP)	A common dispatch language
6	Eliminate software layers between application and hardware	Efficient
7	ISA agnostic for CPU, GPU, DSP, and more	x86, ARM, MIPS, PowerVR, Mali, Adreno, GPT, …
8	Open source software stack	Open Access!



High performance

Low power

Extensible to other accelerators on the SoC

PORTABLE APPLICATIONS PROGRAMMING

FRAMEWORKS, LANGUAGES AND TOOLS

HCC - HETEROGENEOUS COMPUTE COMPILER

C++ & C COMPILER

Compiler Architecture

- Single-source : Host and device code in the same source file
- Fully Open Sourced Compiler using CLANG/LLVM

Expressing Parallelism with HCC

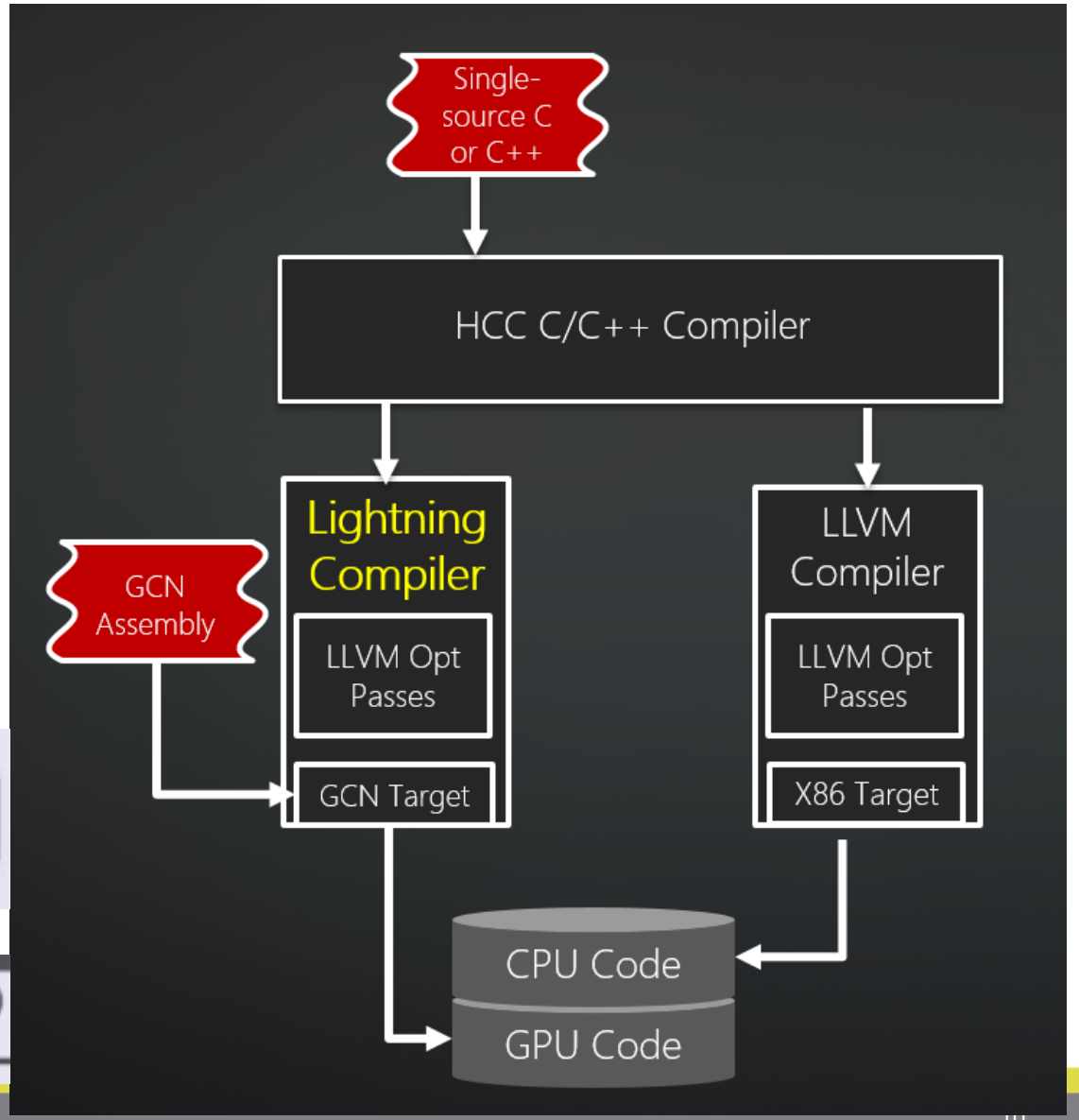
- C++ parallel_for_each + lambda
- C++17 Parallel STL
- OpenMP™ 3.1 C & C++ Support Today for CPU
- C++AMP™ 1.2 standard compatible

HCC generates both CPU and GPU code

- Traditional CPU programs can be compiled with HCC
- Heterogeneous programs
  - Compiles the host code for CPU
  - Compiles the kernel/parallel region for the GPU

Programmer Optimizable

- Control, pre-fetch, discard data movemer
- Run asynchronous compute kernels
- Access GPU scratchpad memories



“VECTOR ADD”, HCC C++

```
int column = 128;
int row = 256;
// define the compute grid
bounds<2> grid { column, row };

float* a = new float[grid.size()];
float* b = new float[grid.size()];
float* c = new float[grid.size()];

// Using standard C++17 launch syntax
parallel::for_each(par, begin(grid), end(grid),
  [&](index<2> idx) {
    int i = idx[1] * column + idx[0];
    c[i] = a[i] + b[i];
}
```

Single-Heap

Single Source

hcc ISO
Standard C++

CUDA PORTING WITH HCC C++

Challenge:

- Many existing GPU-accelerated apps use proprietary CUDA language and infrastructure

Strategy:

- Make it easy to port from CUDA to a *common* C or C++ programming model
- Common = resulting code runs through either CUDA NVCC or HCC C++ compiler
- Can use best development tools on either Nvidia or AMD platform
- *Provide customers with choice in hardware and development tools*

Implementation

- HIP = "**H**eterogeneous-compute **I**nterface for **P**ortability"
- Header maps hip* calls to CUDA RT or HSA RT
- Strong subset of CUDA RT functionality, focus on most commonly used functions
- Some HCC support to make porting easier
- Can use #ifdef for tricky cases and performance tuning

PORTABLE OPENCL™ (POCL)

OpenCL™ 1.2 and 2.0 API implementation

HSA runtime architecture supported

- HSAIL code from OpenCL™ kernels to finalize through HSA Runtime
- Global/local/private memory
- Barriers

Most of the OpenCL 1.2 kernel builtins

- OpenCL™ 2.0 shared virtual memory (SVM)
- OpenCL™ 2.0 atomics

Performance similar to vendor specific implementations

- Portable HSA (phsa) is a portable HSA implementation for CPU & DSP

PYTHON ON GPU'S

Numba: NumPy aware python compiler

- Open source. Avail on Github
- Sponsored by Continuum Analytics

Direct HSA Support

Automatic Parallelization

- Any universal function
- 2x-200x speedup

```
import math

R_EARTH = 6371.0 # km

@numba.hsa.jit('float64(float64)', device=True)
def deg2rad(deg):
    return math.pi * deg / 180.0

@numba.vectorize('float64(float64, float64, float64, float64)', target='hsa')
def gpu_great_circle_distance(lat1, lng1, lat2, lng2):
    '''Return the great-circle distance in km between (lat1, lng1) and (lat2, lng2)
    on the surface of the Earth.'''
```

RUNTIME

RUNTIME FEATURES

Thin user-mode API

- Interfaces for host to launch compute kernels
- Standard across all vendors
- Does not combine runtime from different vendors

Error Handling

System and Agent information

Signal and Synchronization

Architected Dispatch

Memory Management

RUNTIME PIC GOES HERE

SYSTEMS ARCHITECTURE OVERVIEW

SYS ARCH SPEC OVERVIEW

Requirements Overview

- What is HSA
- Minimum vs complete product
- Programming model
- Compliant systems requirements

2.0 Details

- Shared virtual memory
- Cache coherency
- Flat addressing
- Endianess
- Signaling and synchronization
- Atomic memory operations
- System timestamp
- User mode queueing
- AQL packets
- Agent scheduling
- Kernel dispatch forward progress
- Floating point exceptions
- Kernel agent debug infrastructure
- Platform topology discovery
- Images
- Profiling

Memory Consistency Model

- Definitions
- Execution
- Program order
- Coherent order
- Global dependency order
- Scoped synchronization order
- Happens-before order
- Race-free programs
- Non-sequentially consistent execution
- Races

SYSTEM ARCHITECTURE OVERVIEW

Support Data-Parallel and Task-Parallel Programming

- Multiple instruction sets
- Host CPU's and Kernel agents

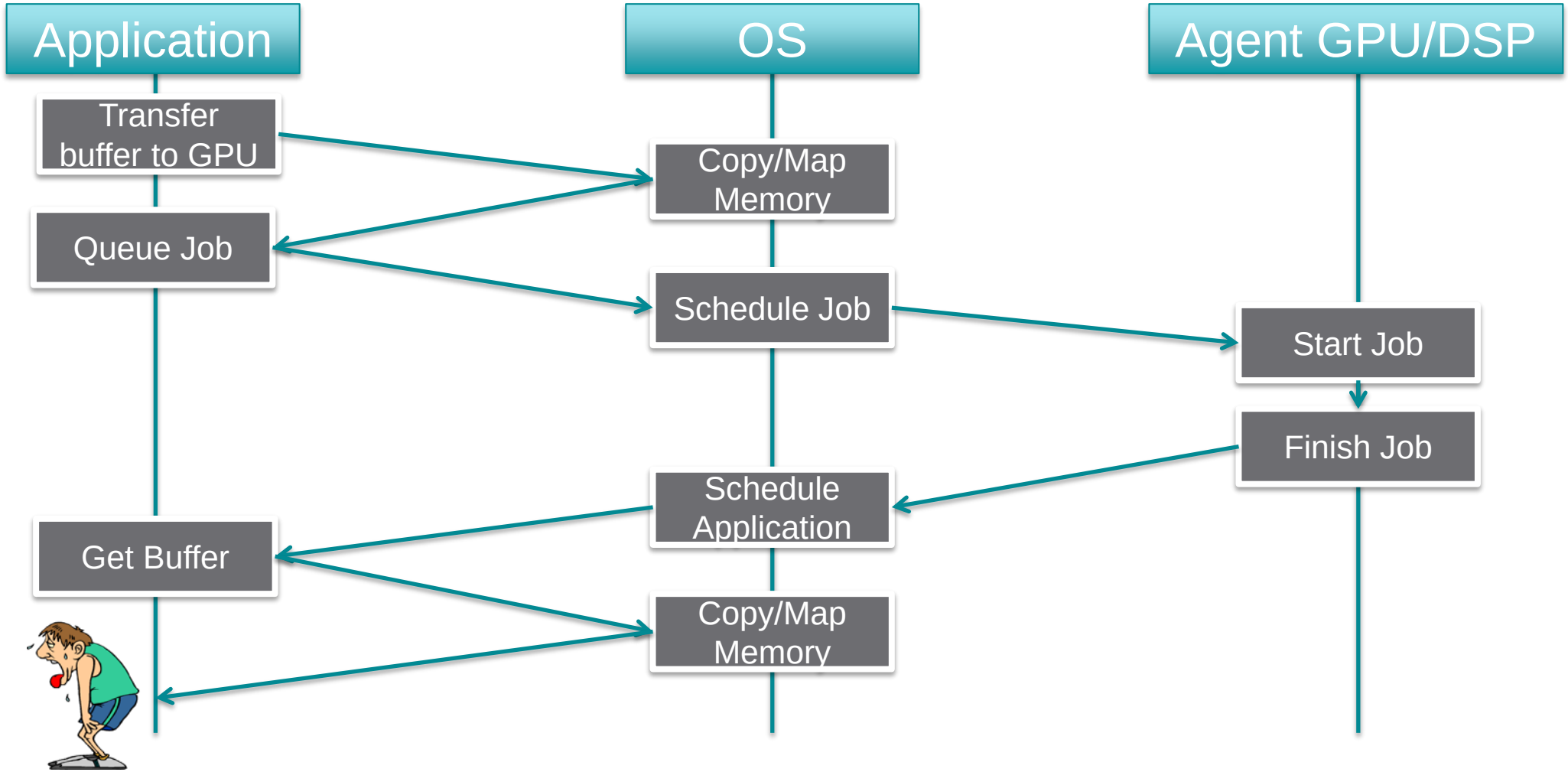
Two Machine Models

- Small Model: 32-bit address space
- Large Model: 64-bit address space
  - optional 32-bit process to run small model code

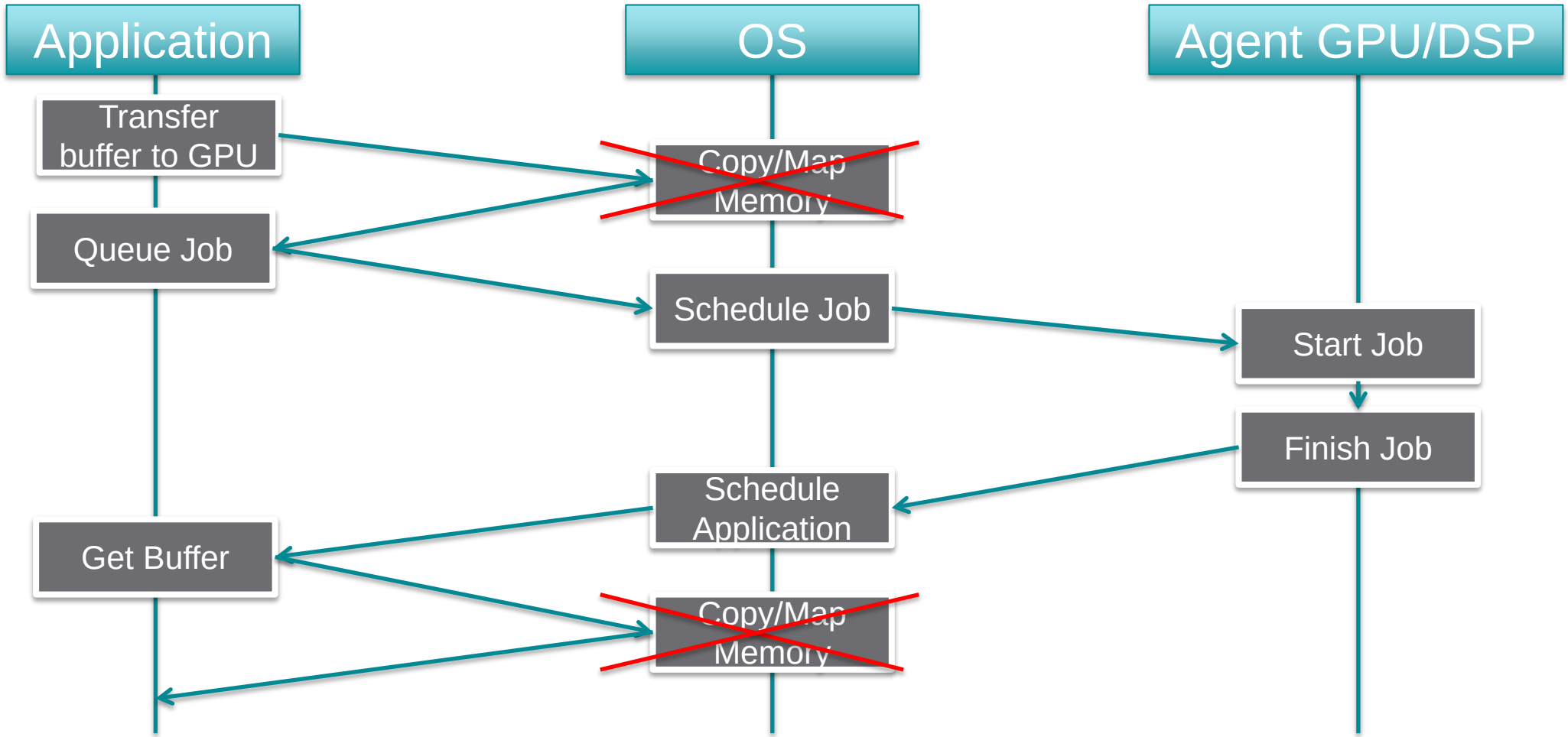
HSA Compliant System Meets:

- Queuing model
- Memory model
- Quality of service
- ISA for parallel processing
- Standardized interfaces, processes, communication protocols, and memory models

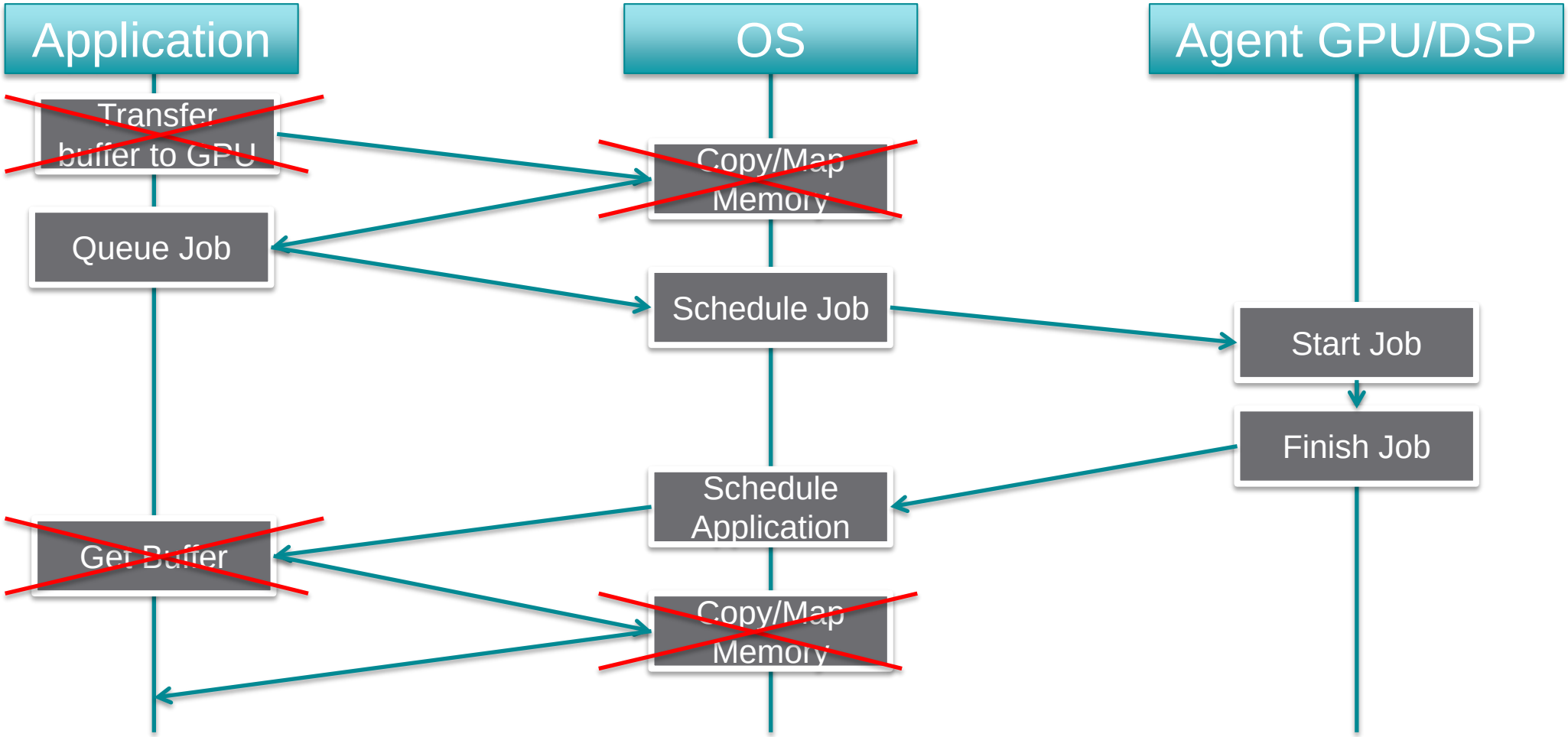
MOTIVATION (TODAY'S PICTURE)



WITH SHARED VIRTUAL MEMORY



WITH COHERENT CACHE MEMORY



SIGNALS

HSA agents support signaling

- creation/destruction using runtime APIs

Any Agent can access signals

- Wake up agents waiting upon the object
- Query/Wait for current object
- Allows conditions

Hardware-assisted signaling and synchronization primitives

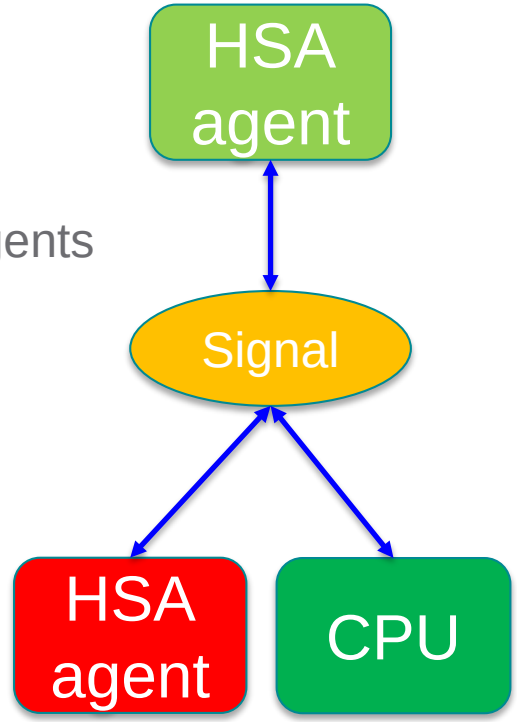
- Memory semantics, equivalent to platform atomics
  - e.g. 32bit or 64bit value, content updated atomically
  - wait on value by HSA agents and AQL packets
- Synchronizes execution between threads on HSA agents and host CPU

One-to-one and one-to-many signaling

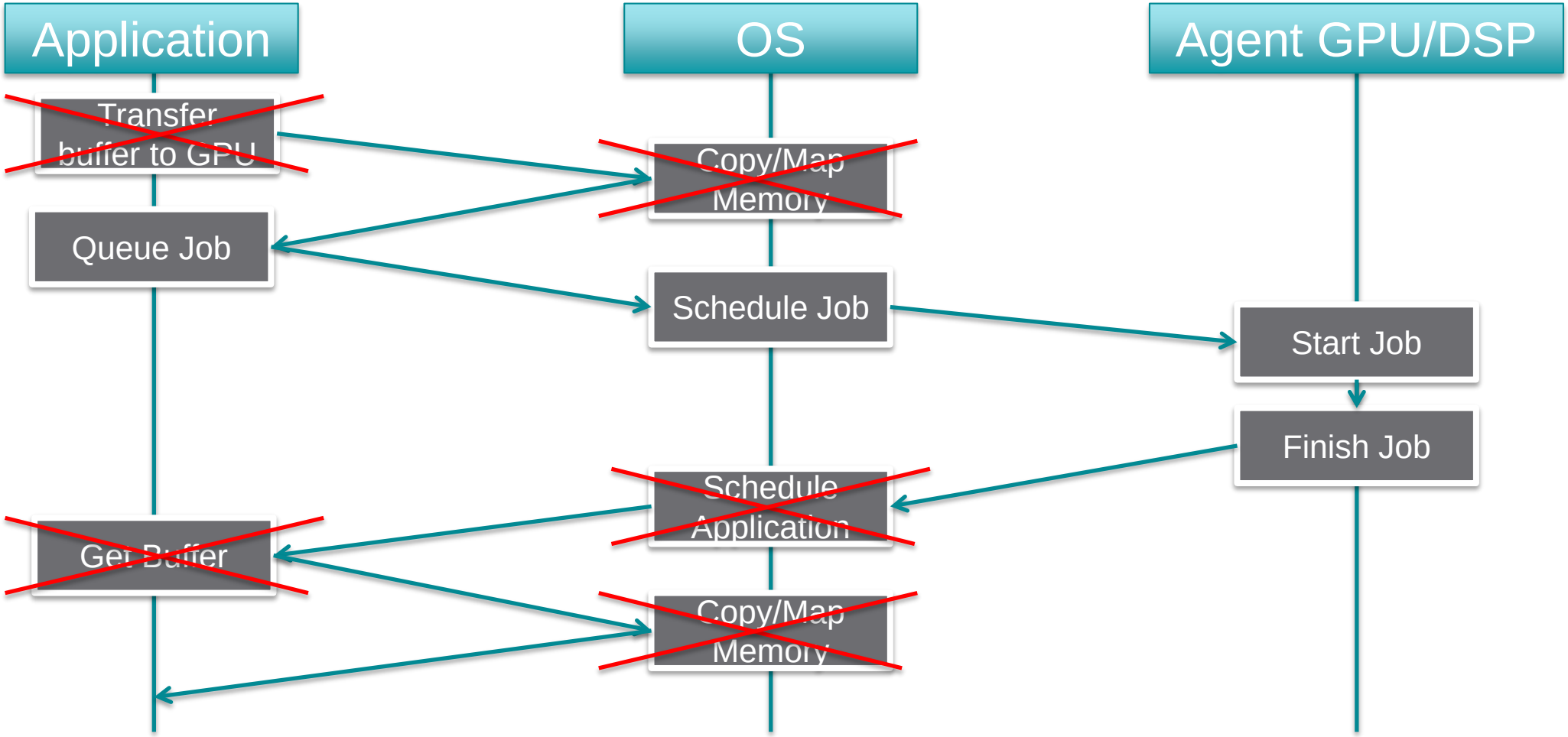
- System Software, runtime & application SW use infrastructure to build higher-level synchronization
  - mutexes, semaphores, …

Advantages

- Asynchronous events between agents
  - Doesn't require CPU
- Common idiom for work offload
- Low power waiting



WITH SIGNALING



HSA QUEUING MODEL

User mode queuing

- Low latency dispatch
- Application dispatches directly
- No OS or driver required
- “Unlimited” # of queues per process

Architected Queuing Layer (AQL)

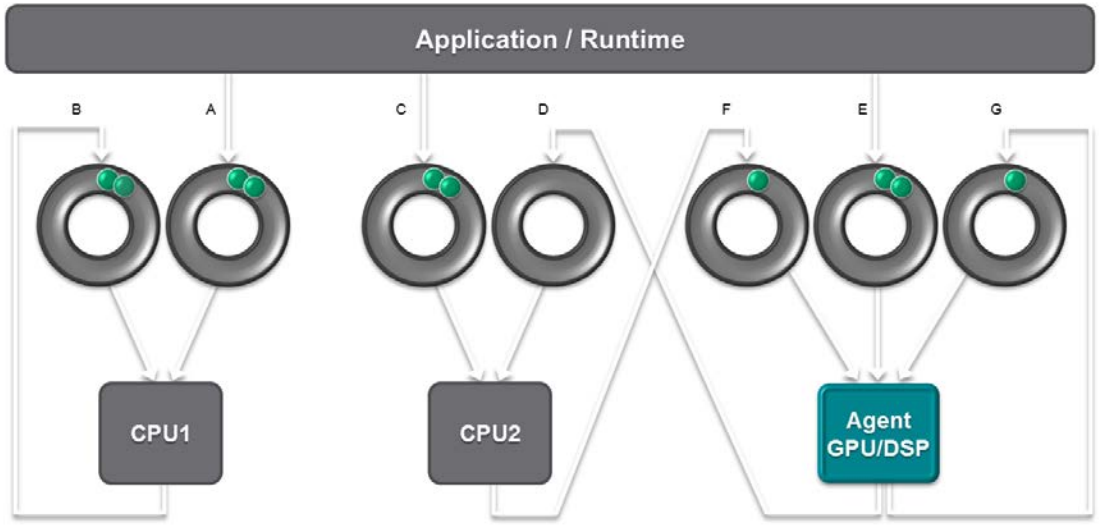
- Single compute dispatch path for all hardware
- No driver translation, direct to hardware
- Standard across vendors!
- Guaranteed backward compatibility

Allows for dispatch to queue from any agent

- CPU or GPU or DSP or FPGA, etc.

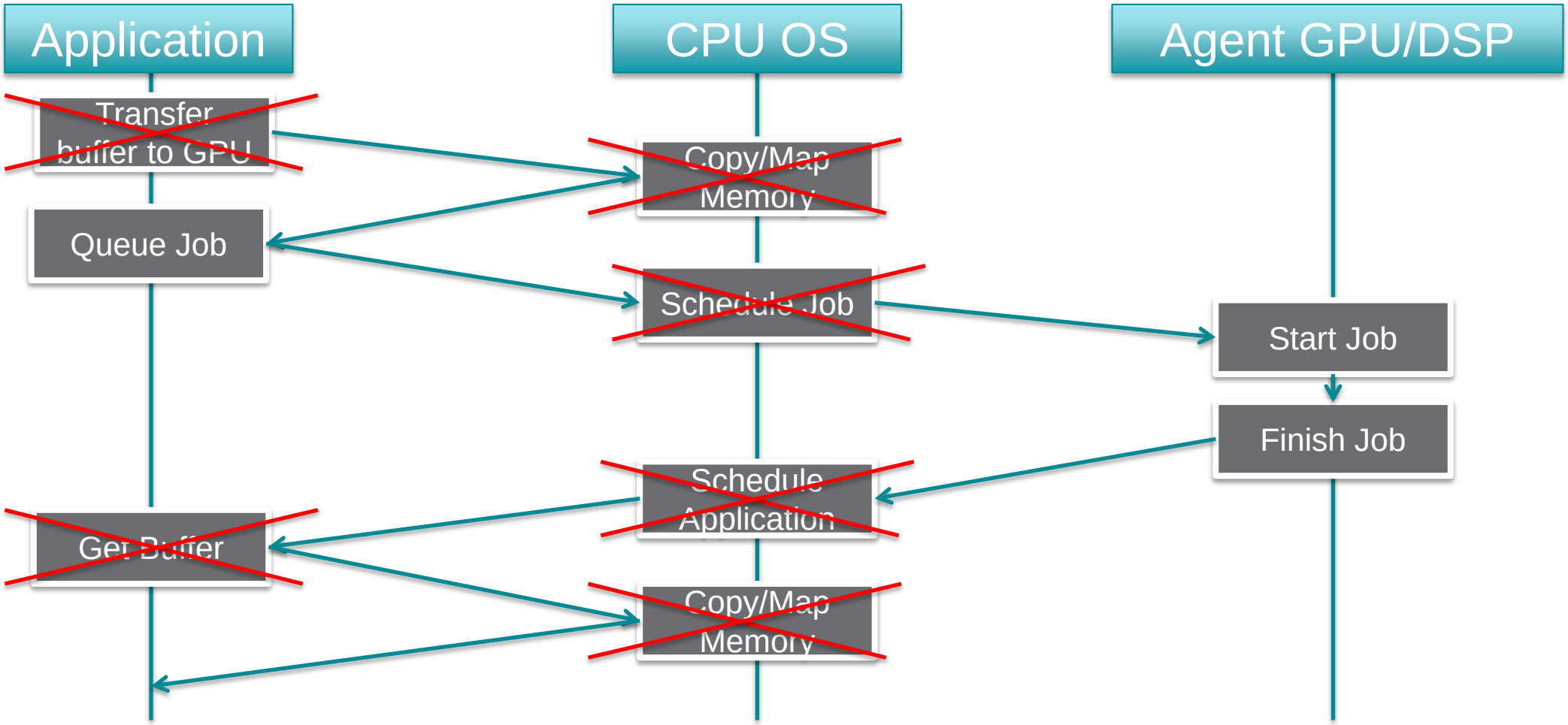
Agent self enqueue enables

- Recursion, Tree traversal, Wavefront reforming

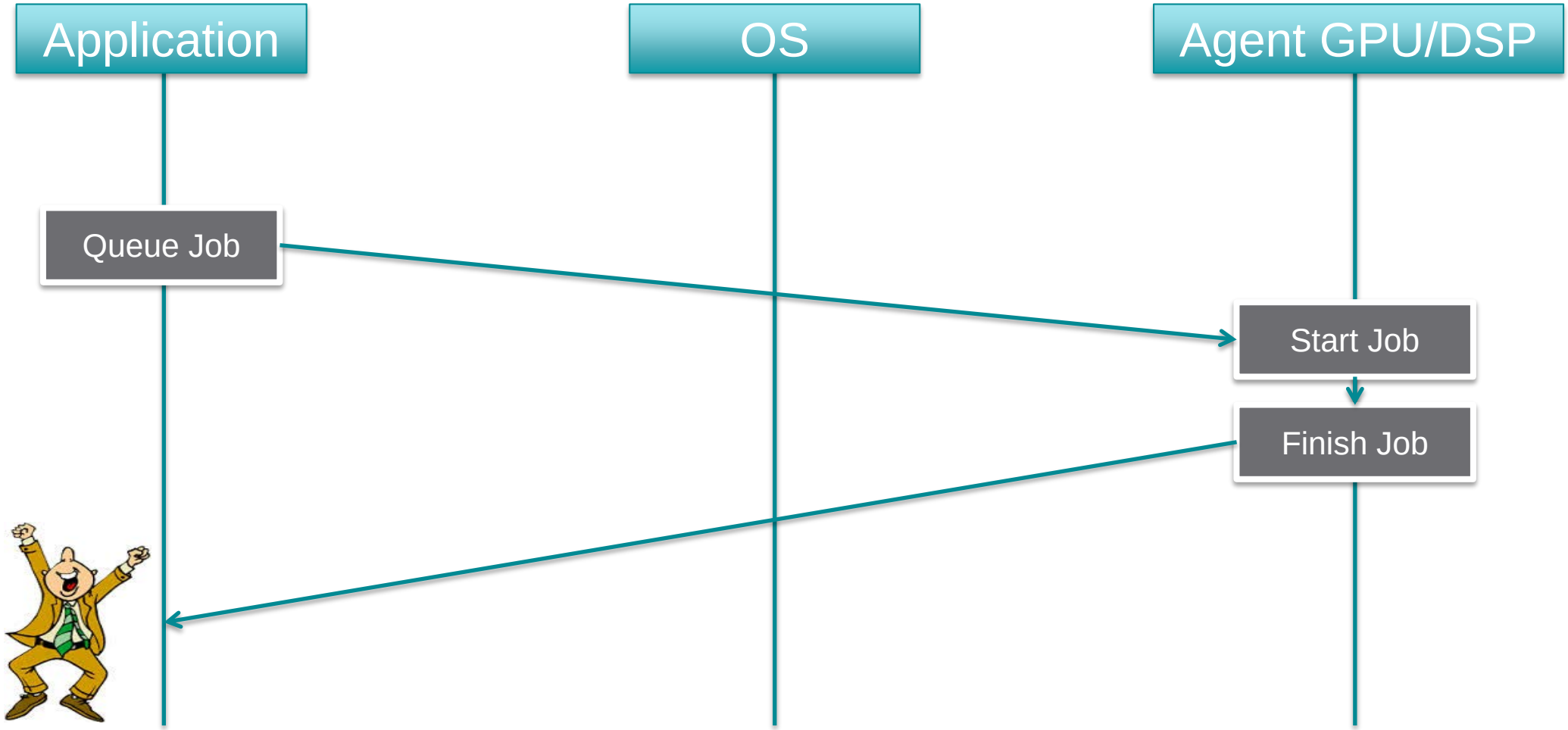


Benefits from shared virtual memory, platform atomics and coherency

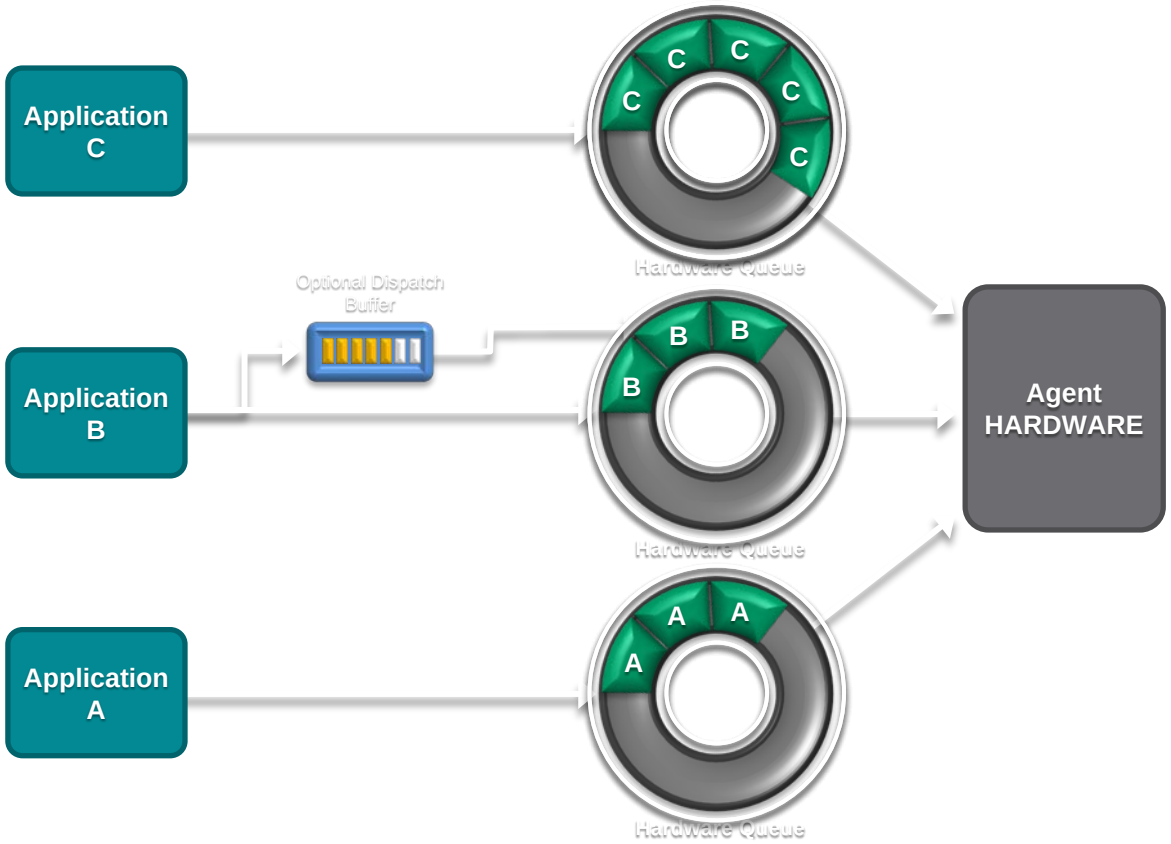
WITH USER MODE QUEUING



FINAL PICTURE: SVM + CACHE COHERENCY + SIGNALS + USER MODE QUEUES



HSA COMMAND AND DISPATCH FLOW

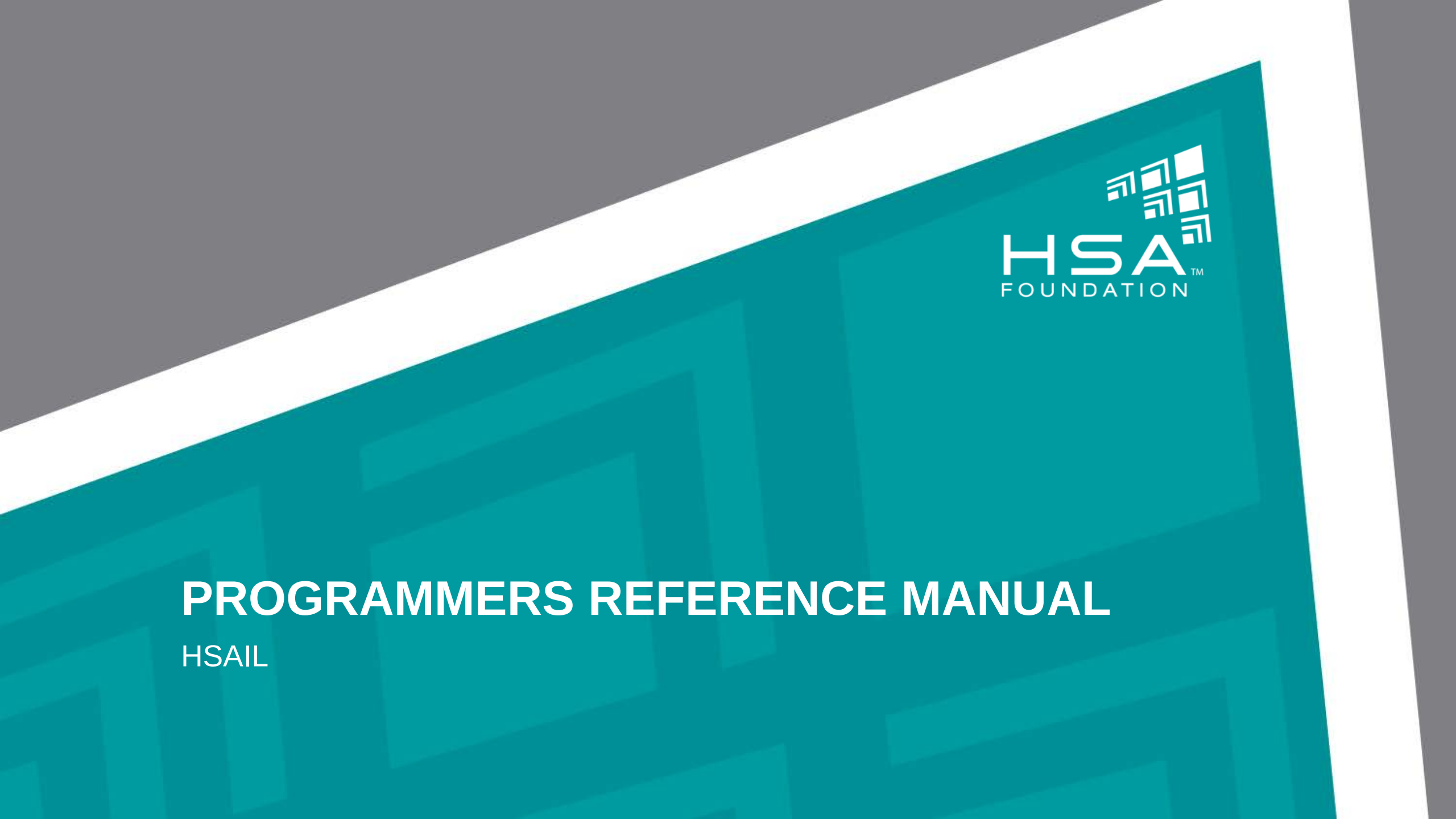


SW view:

- User-mode dispatches to HW
- No KMD overhead
- Low dispatch times
- CPU & GPU dispatch APIs

HW view:

- HW / microcode controlled
- HW scheduling
- Architected Queuing Language (AQL)
- HW-managed protection



PROGRAMMERS REFERENCE MANUAL

HSAIL

THE PORTABILITY CHALLENGE

CPU ISAs – Backwards Compatible

- ISA innovations added incrementally (ie NEON, AVX, etc)
  - ISA retains backwards-compatibility with previous generation
- HSA instruction-set architectures: ARM, MIPS, and x86

Kernel Agent ISAs – No Backwards Compatibility

- GPU, DSP, DNN, Image Signal Processor, Custom Accelerators, etc.
- Massive diversity of architectures in the market
  - Each vendor has own ISA - and often several in market at same time
- Compatibility via APIs (OpenGL, DirectX, OpenCV)

HSA INTERMEDIATE LAYER — HSAIL

Virtual ISA for parallel programs

- Finalized to native ISA by a compiler
  - Dynamic or Offline
- ISA independent by design

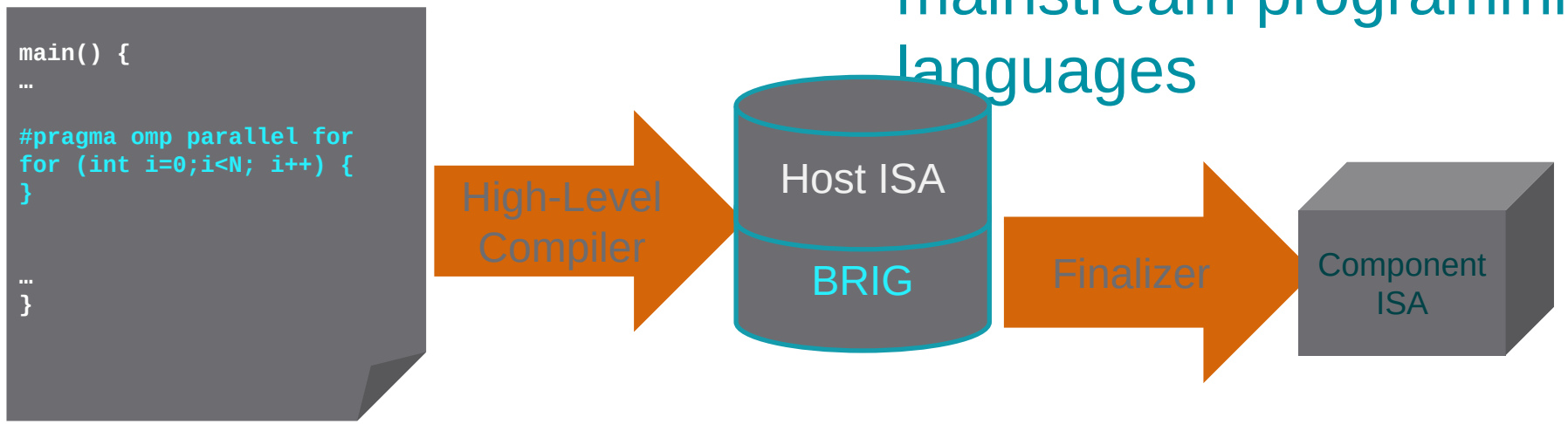
Explicitly parallel

- Designed for data parallel programming

Multiple HLL Support

- Exceptions, virtual functions, etc.
- Java, C++, OpenMP, C++, Python, etc

Brings parallel acceleration to mainstream programming languages



HSAIL FEATURES

A Virtual Explicitly Parallel ISA

- ~135 Opcodes
- RISC Register-based Load/Store
- Arithmetic
  - IEEE 754 Floating Point including 16-bit
  - Integer (32/64-bit)
  - DSP fixed point
  - Packed / SIMD
    - f16x2, f16x4, f16x8, f32x2, f32x4, f64x2
    - signed/unsigned 8x4, 8x8, 8x16, 16x2, 16x4, 16x8, 32x2, 32x4, 64x2
- Branches & Function Calls
- Atomic Operations

Wavefronts

- 1, 2, 4, 8, 16, 32, or 64 SIMD lanes
- Lanes can be active or inactive

Memory

- Shared Virtual Memory

Exceptions

```
ld_global_u64     $d0, [$d6 + 120] ; $d0= load($d6+120)
add_u64           $d1, $d0, 24     ; $d1= $d2+24
```

PERFORMANCE RESULTS

Python Geographic Locality

What is the distance from a set of points to a target point

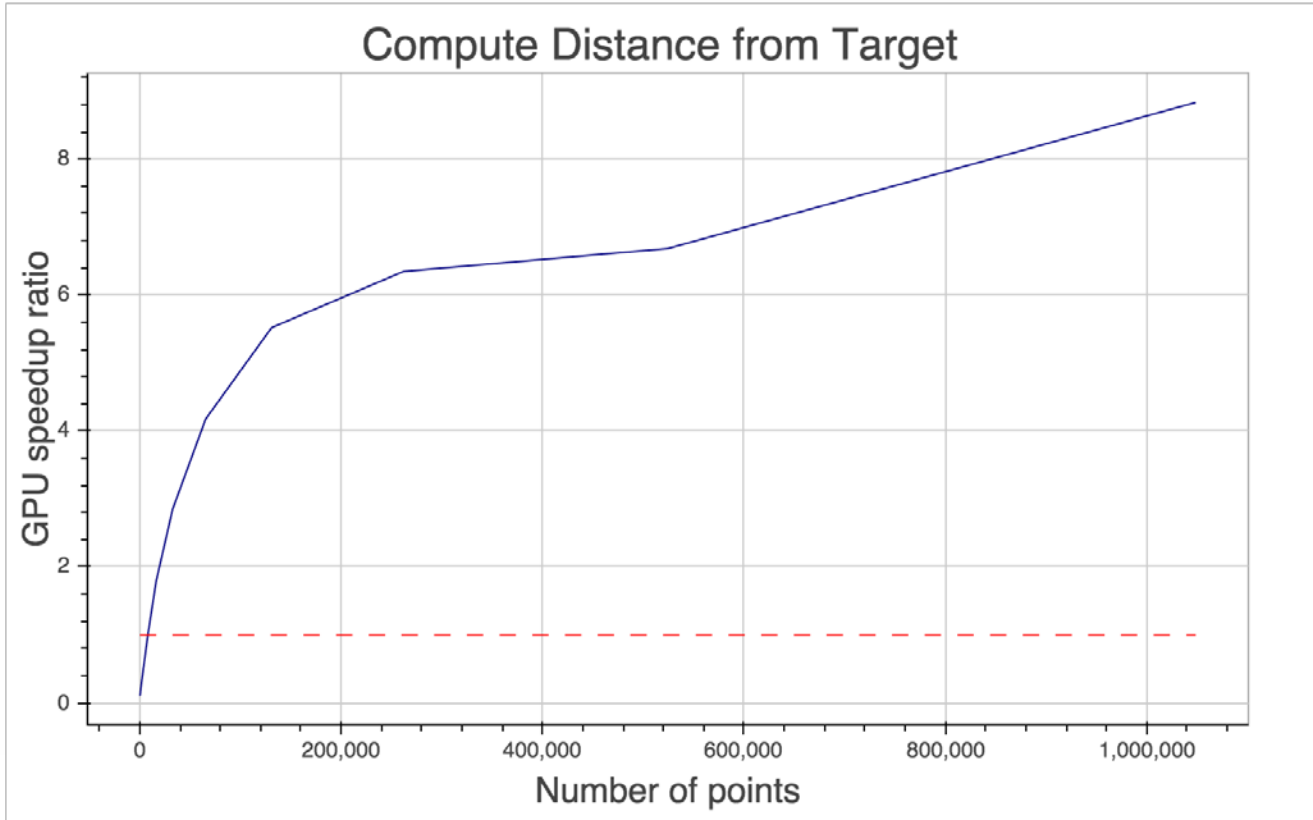
- How many points are within a specified range

Numba can auto-parallelize user universal functions for HSA

- Ufunc's broadcast operation over elements of a NumPy array
- ZERO HSA developer knowledge required

1M Points

- >8X speedup



https://github.com/ContinuumIO/Numba-HSA-Webinar

OPENCL: FIR & AES

FIR is a memory-intensive streaming workload

AES is a compute-intensive streaming workload

CL12 – cl_mem buffer

- Copy to/from the device

CL20 – SVM buffer – Coarse Grain Sync

- Copy to/from SVM
- Data copy cannot be avoided, since the space for SVM is limited

HSA – Unified Memory Space – Fine Grained Sync

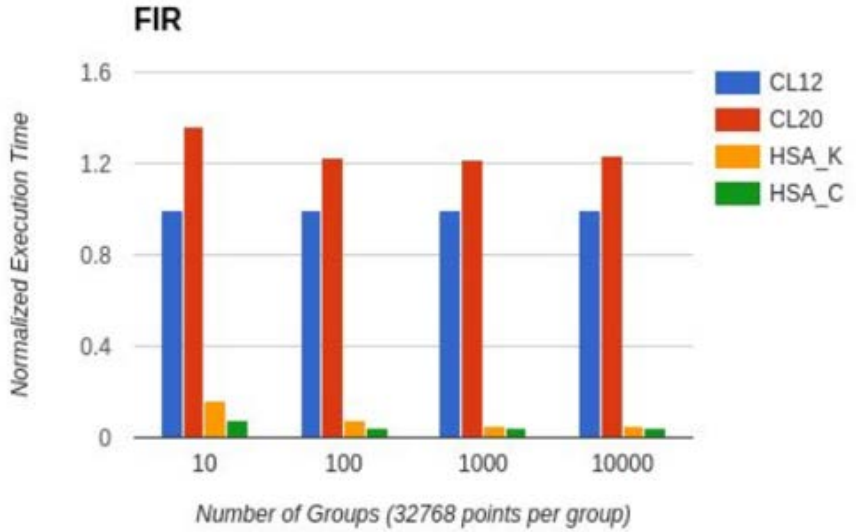
- Regular pointer
- No explicit copy

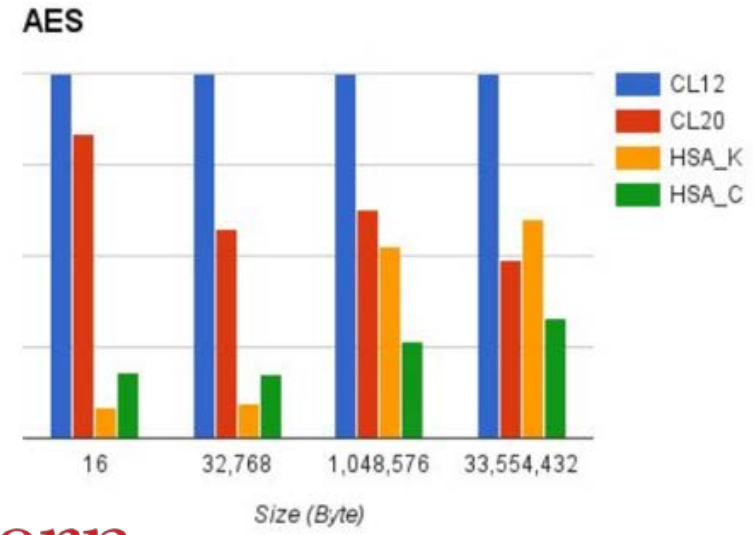
Results

- HSA compute abstraction
- NO performance penalty

Not all algorithms run faster

- Measured on Kaveri (A pre-HSA 1.0 device)
- Limited Coherent throughput





Northeastern

Saoni Mukherjee, Yifan Sun, Paul Blinzer, Amir Kavyan Ziabari, David Kaeli,*A Comprehensive Performance Analysis of HSA and OpenCL 2.0,* **Proceedings of the 2016 International Symposium on Program Analysis and System Software,** April 2016, to appear.

BLACK-SCHOLES

C++ on HSA

- Matches or outperforms OpenCL

Course Grained SVM

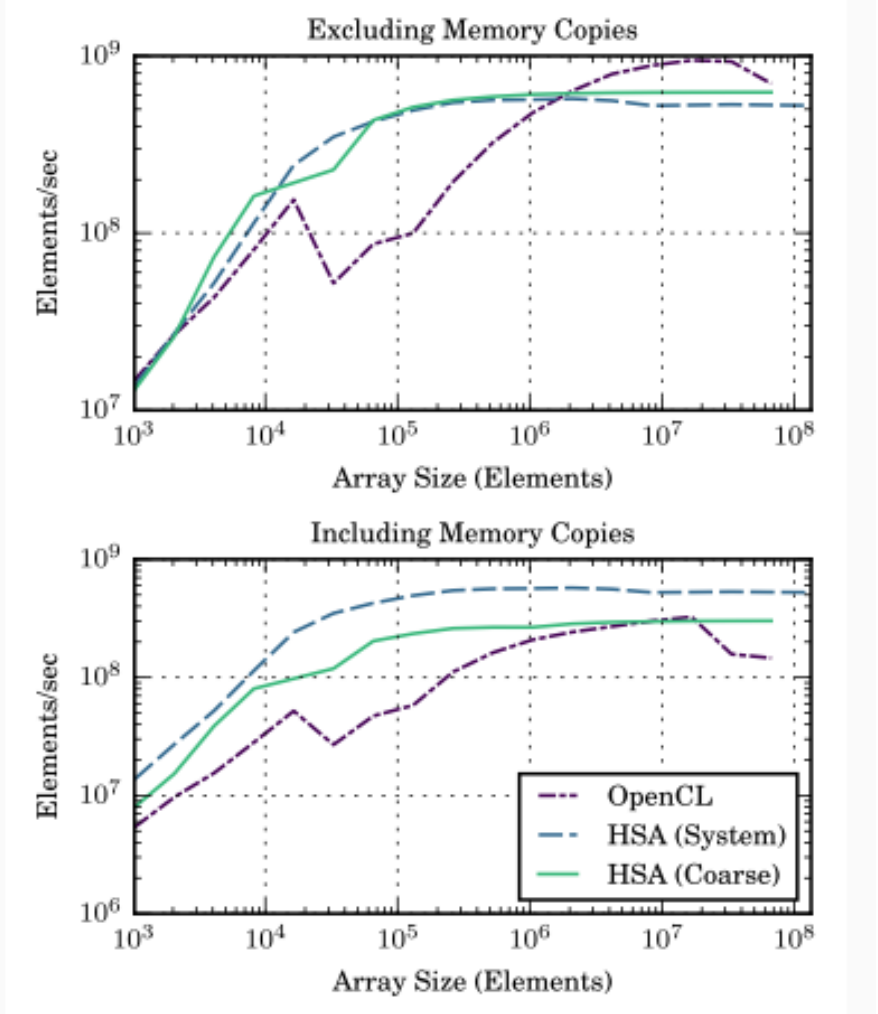
- Matches OpenCL buffers for bandwidth
- More predictable performance

Fine Grained SVM

- Faster kernel dispatch
- Larger allocations
- Shared data structure

Results

- HSA compute abstraction
- NO performance penalty



Source: Ralph Potter – Codeplay. Presentation made to SG14 C++ Workgroup

HSA PRODUCTS

HP PAVILION - 15Z

15.6" diagonal FHD IPS UWVA BrightView WLED-backlit (1920 x 1080) Touch Display

- Model V1M95AV_1

AMD Quad-Core A12-9700P (Bristol Ridge)

- 2.5 GHz, up to 3.4 GHz,
- 2 MB cache
- AMD Radeon™ R7 Graphics

16GB DDR4-2133 SDRAM (2 x 8GB)

2TB 5400 rpm SATA

- Or 256GB SSD

HP Wide Vision HD Webcam with Dual Digital Microphone

Intel® 802.11ac (1x1) Wi-Fi® and Bluetooth® 4.2 Combo

Full-size island-style backlit keyboard

SuperMulti DVD burner

Runs RoCm

$500 base price

$880 as configured

CONCLUSIONS

V1.2 SPECIFICATIONS IN PROGRESS

Coarse grain memory

- Improved Data Interop
- Fixed function accelerators (e.g. FPGA)
- Local device memory

Architecture

- Architected Debug
- BRIG, new linking formats

Programming Models

- Fully formalized memory model
- HSAIL Parallel loops
- Flexible API and access semantics

SUMMARY

2012 goal of changing chip H/W architecture achieved

- Cache coherent shared virtual memory

2014-2015 S/W architecture to support H/W

- March 2015 V1.0 specs
- Programmed in any language (C++, Python, OpenCL)

2015-2016

- May 2016 v1.1 specs (backward compatible)
- Multivendor support
- Wider range of processors

2016 Initial H/W platforms arriving

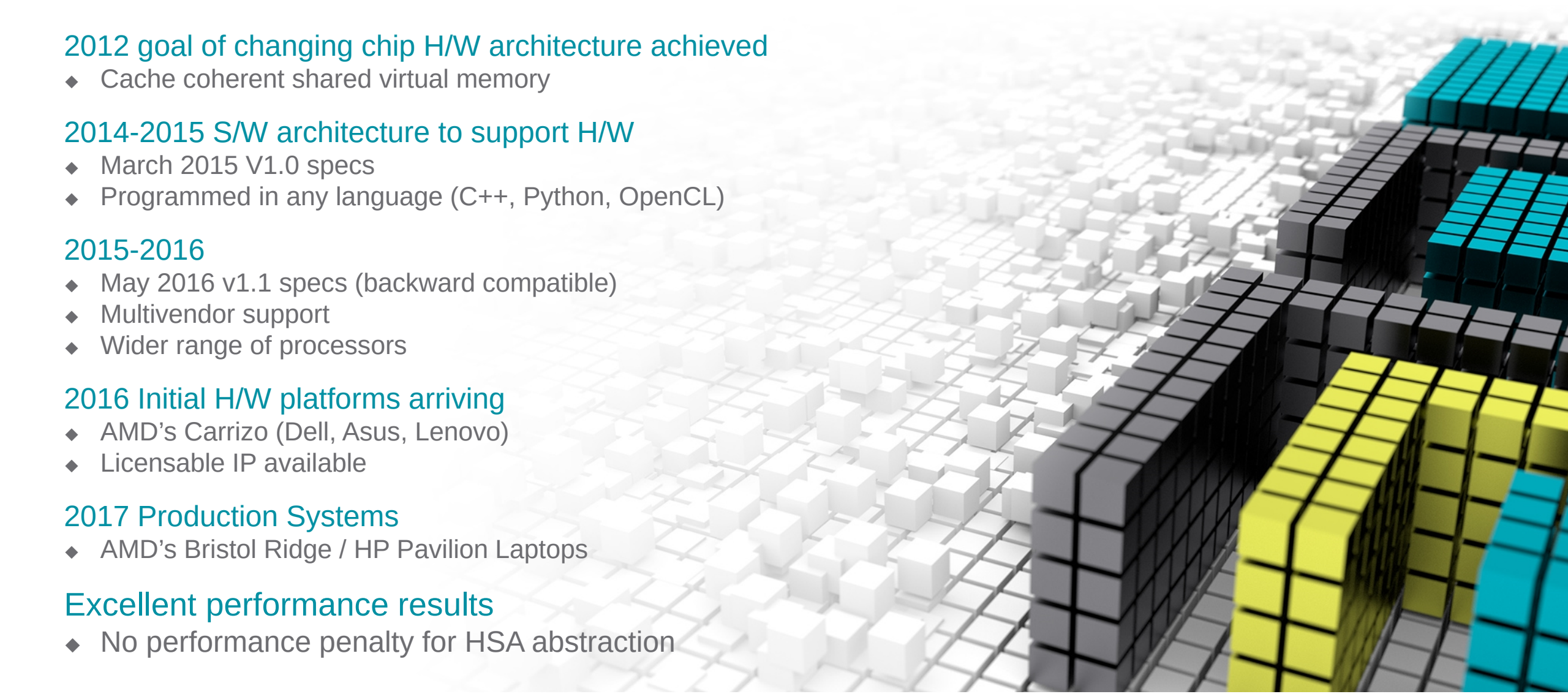
- AMD's Carrizo (Dell, Asus, Lenovo)
- Licensable IP available

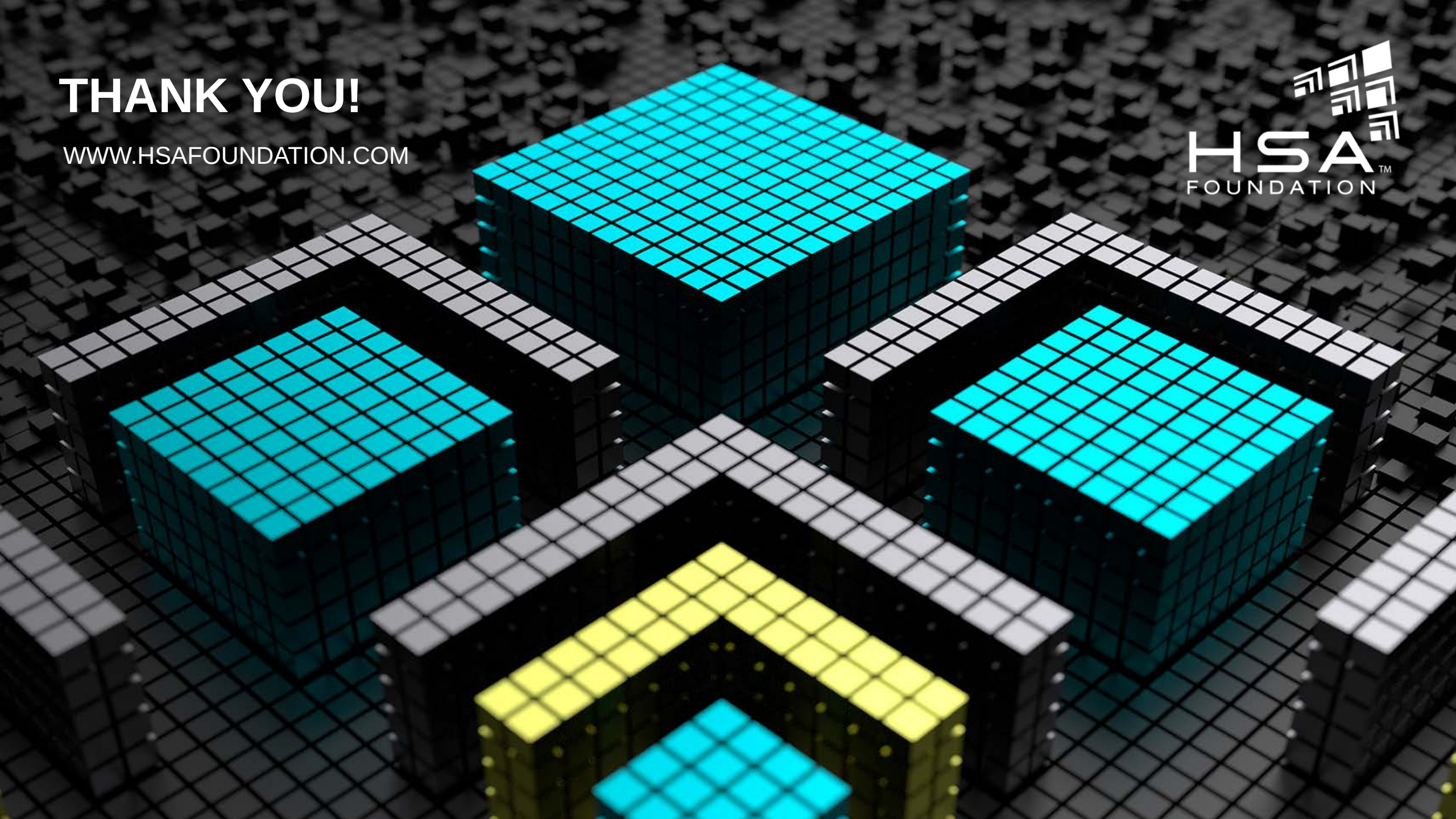
2017 Production Systems

- AMD's Bristol Ridge / HP Pavilion Laptops

Excellent performance results

- No performance penalty for HSA abstraction




THANK YOU!
WWW.HSAFOUNDATION.COM
HSA™
FOUNDATION