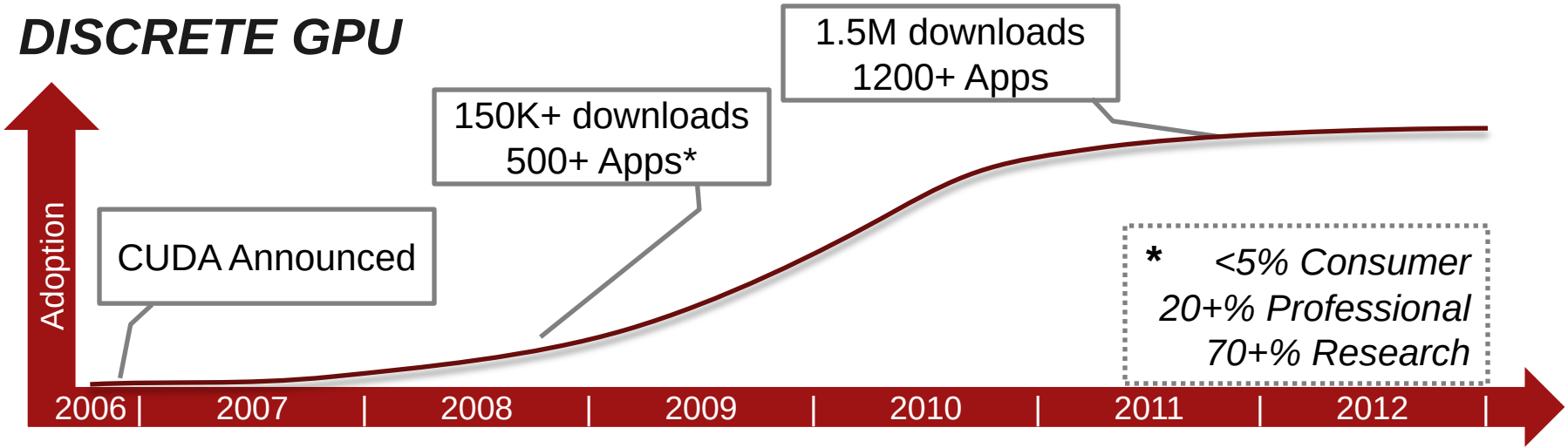
HETEROGENEOUS SYSTEM ARCHITECTURE (HSA) AND THE SOFTWARE ECOSYSTEM

MANJU HEGDE, CORPORATE VP, HETEROGENEOUS SOLUTIONS, AMD

CUDA BRINGS PERFORMANCE TO PRO/RESEARCH ON DISCRETE GPU

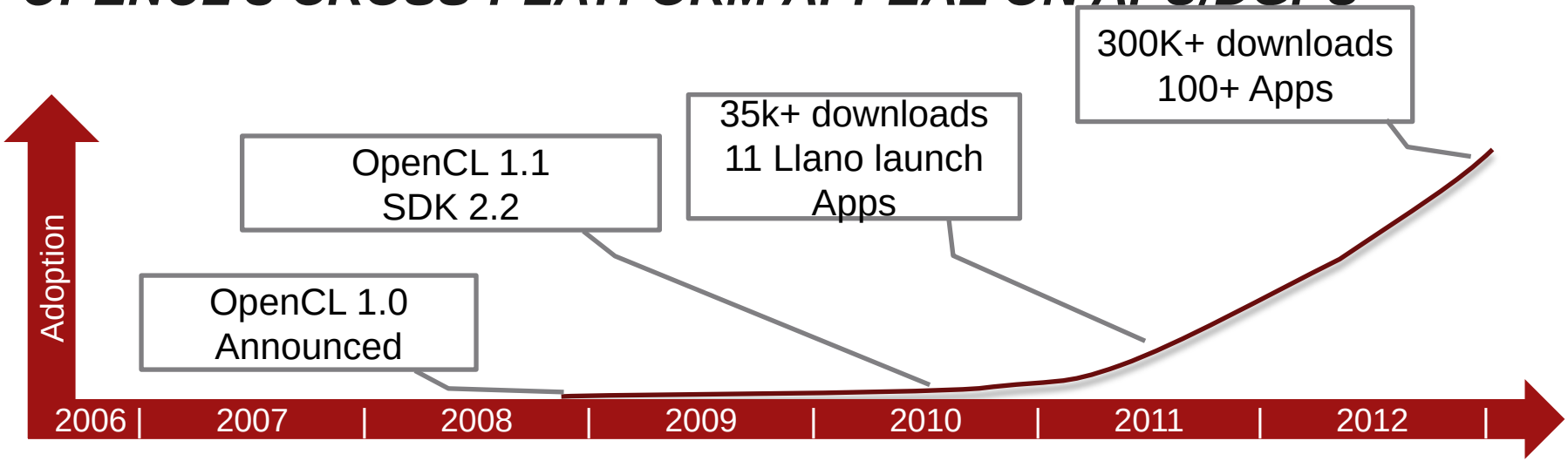


CUDA gave developers access to unprecedented performance

Not easy to use …but enough performance-hungry developers willing to endure pain

Low Consumer space adoption … esp. due to lack of cross-platform

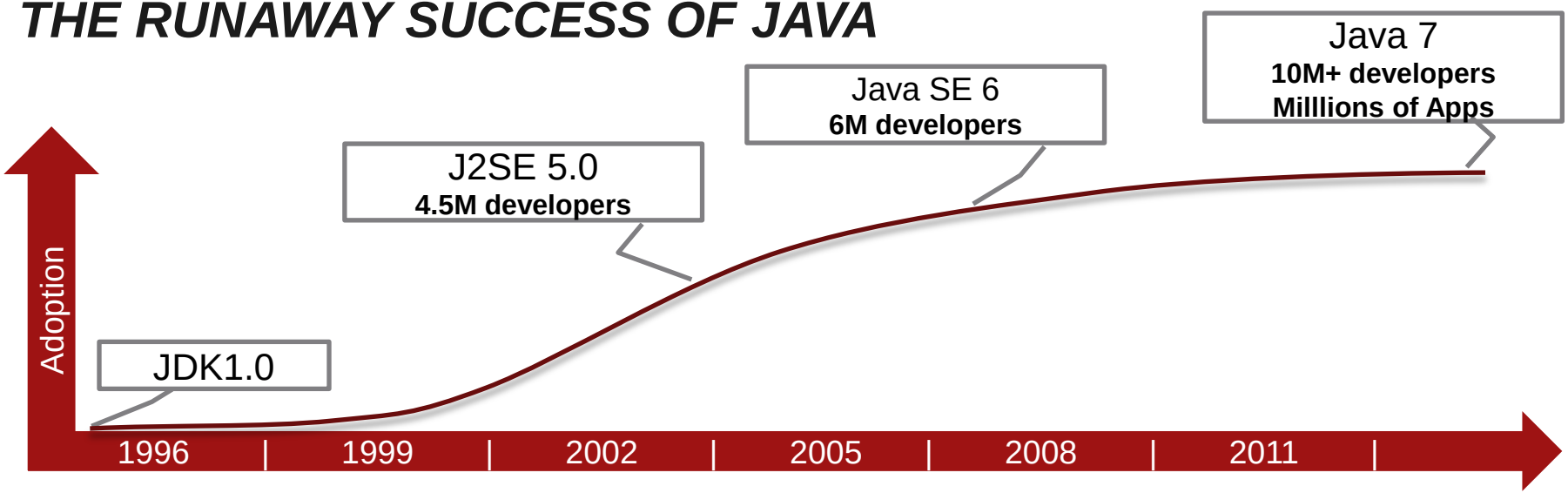
OPENCL'S CROSS-PLATFORM APPEAL ON APU/DGPU



Abundant performance + same complexity as CUDA programming

Cross platform resonates with developers *(needs per-platform optimization)*

THE RUNAWAY SUCCESS OF JAVA



- Easy to program
- Truly cross platform – **W**rite **O**nce **R**un **A**nywhere
- Lack of performance efficiency offset by platform capability

You *can* get developers to change!

(takes time and strategy)

HSA FOUNDATION : DRIVING FUTURE OF HETEROGENEOUS COMPUTING

HSA FOUNDATION

Founders

AMD ARM Imagination MEDIATEK QUALCOMM SAMSUNG Texas Instruments

Promoters

LG Electronics

Supporters

Arteris codeplay FABRIC ENGINE MULTICORE WARE Sandia National Laboratories

Contributors

apical CEVA imp SONICS MARVELL SONY make.believe ST ERICSSON ANALOG DEVICES life.augmented Swarm64 <symbio> tensilica VIVANTE

Academic

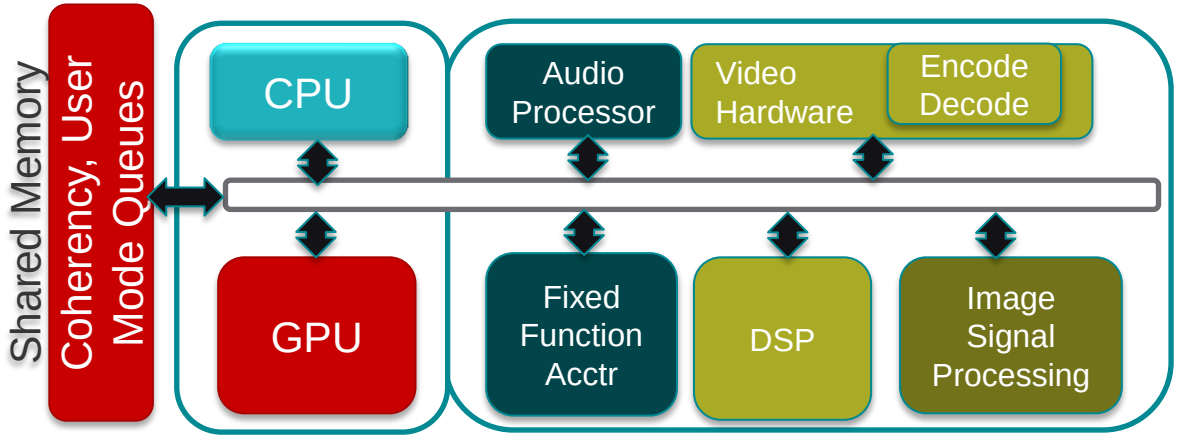
NTHU Programming Language Lab NTHU System Software Lab University of BRISTOL THE UNIVERSITY of EDINBURGH informatics ILLINOIS COMPUTER SCIENCE

GOALS FOR THE HETEROGENEOUS SYSTEM ARCHITECTURE

- Advanced Natural User Interfaces & Presence Capabilities
- Rich Cloud Computing User Experiences
- Perceptual Computing Experiences
- Bring Hollywood Class Realism to Real-time Entertainment

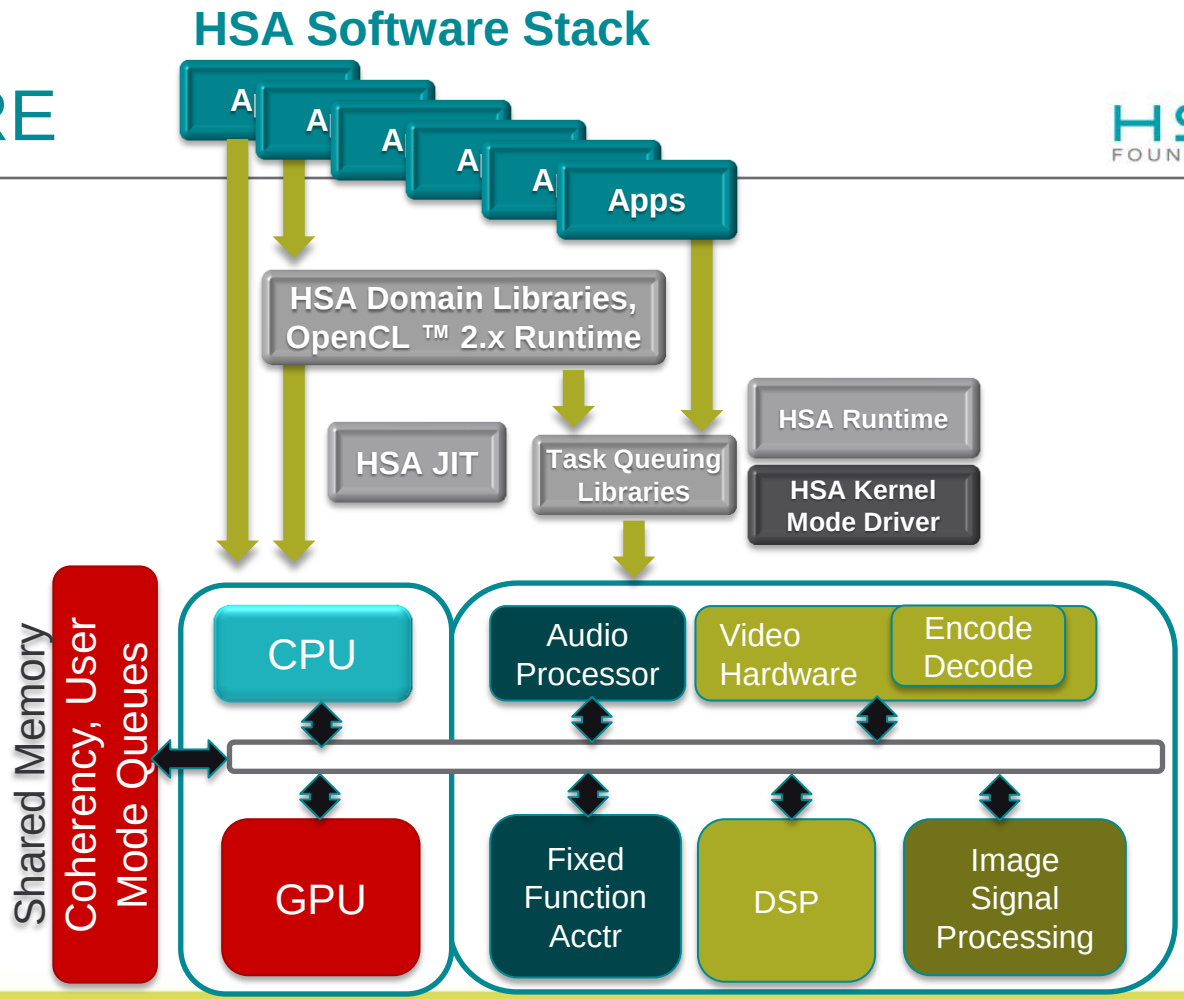
HSA ARCHITECTURE

- **GPU compute C++ support**
- **User Mode Scheduling**
- **Fully coherent memory between CPU & GPU**
- **GPU uses pageable system memory via CPU pointers**
- **GPU graphics pre-emption**
- **GPU compute context switch**



HSA ARCHITECTURE

- GPU compute C++ support
- User Mode Scheduling
- Fully coherent memory between CPU & GPU
- GPU uses pageable system memory via CPU pointers
- GPU graphics pre-emption
- GPU compute context switch

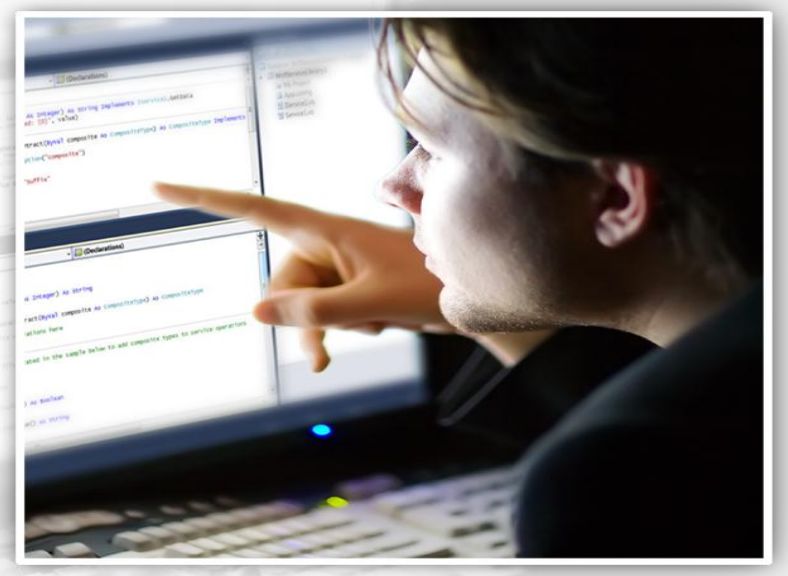


HSA INTERMEDIATE LANGUAGE - HSAIL

- Designed for C99, C++ 2011, Java, Renderscript, OpenCL, C++ AMP
- 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
- Syscall methods
  - GPU code can call directly to system services, IO, printf, etc

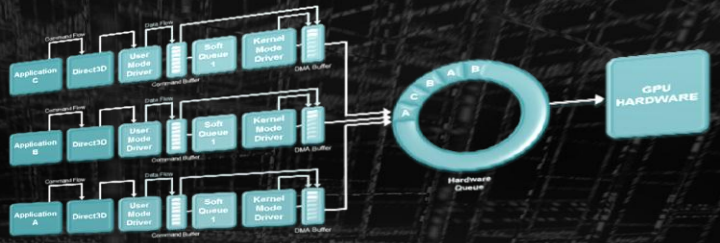
OPENCL™ AND HSA

- HSA is an optimized platform architecture for OpenCL™
  - Not an alternative to OpenCL™
- OpenCL™ on HSA will benefit from
  - Avoidance of wasteful copies
  - Low latency dispatch
  - Improved memory model
  - Pointers shared between CPU and GPU
- HSA also exposes a lower level programming interface, for those that want the ultimate in control and performance
  - Optimized libraries may choose the lower level interface

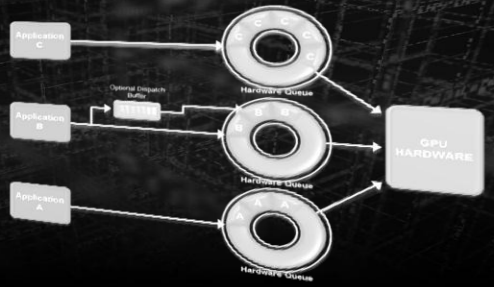


HETEROGENEOUS COMPUTE DISPATCH

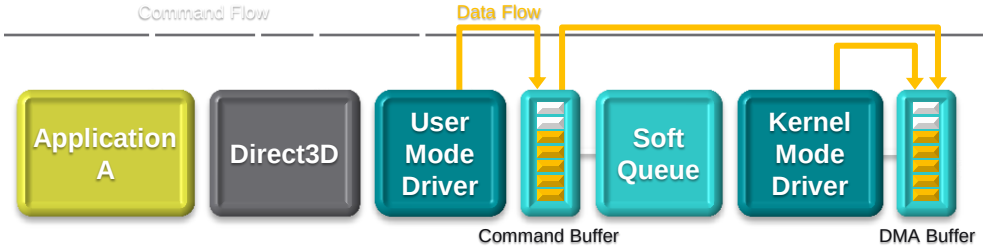
*How compute dispatch operates today in the **driver model***

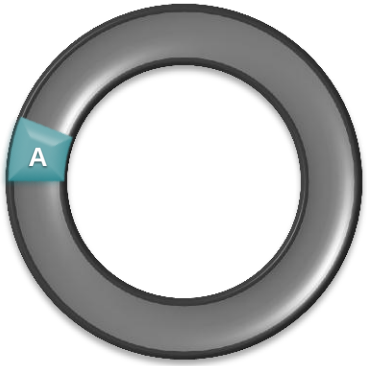


*How compute dispatch improves **under HSA***



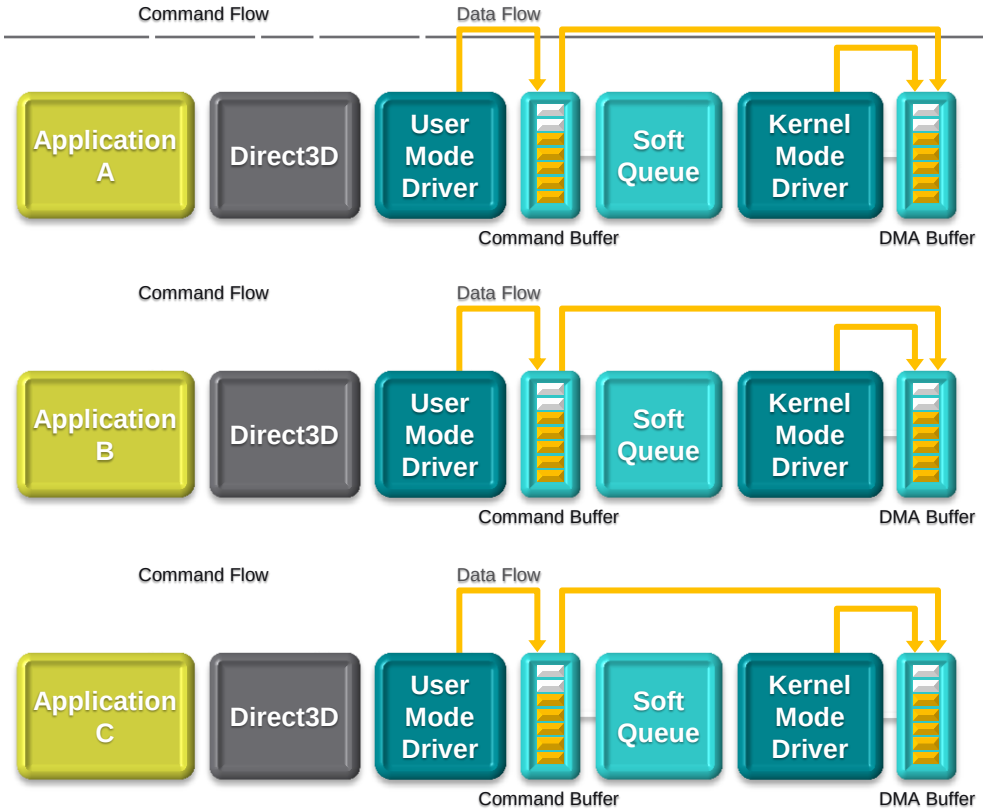
TODAY'S COMMAND AND DISPATCH FLOW

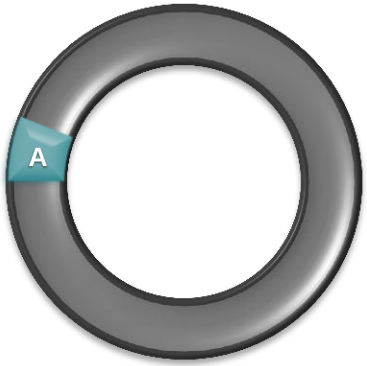


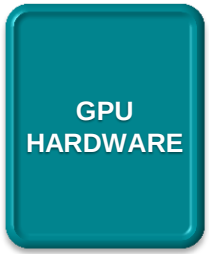


GPU HARDWARE

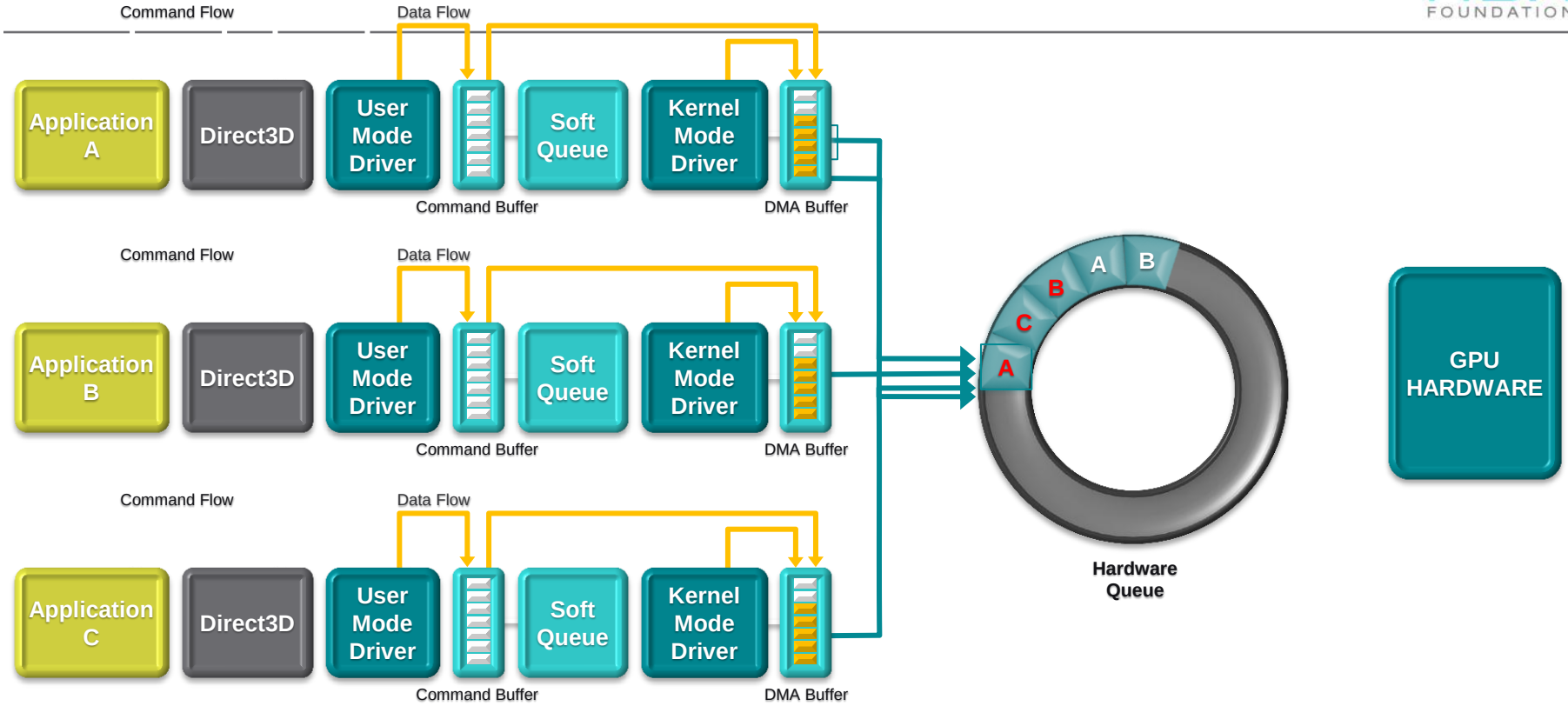
TODAY'S COMMAND AND DISPATCH FLOW



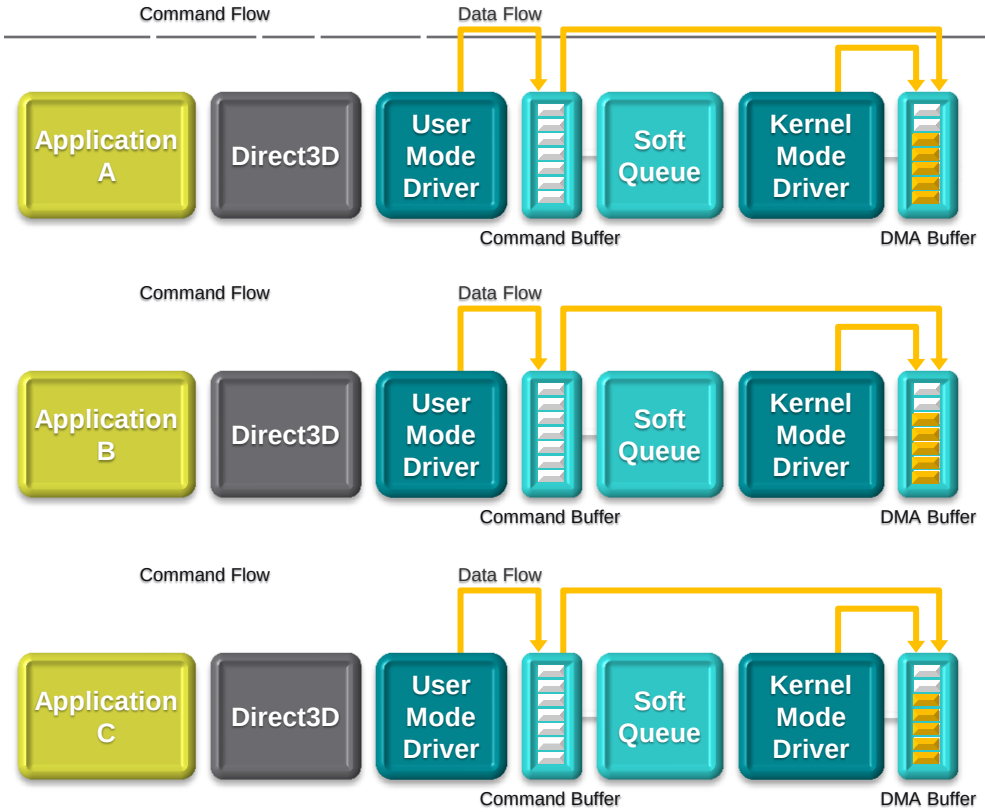


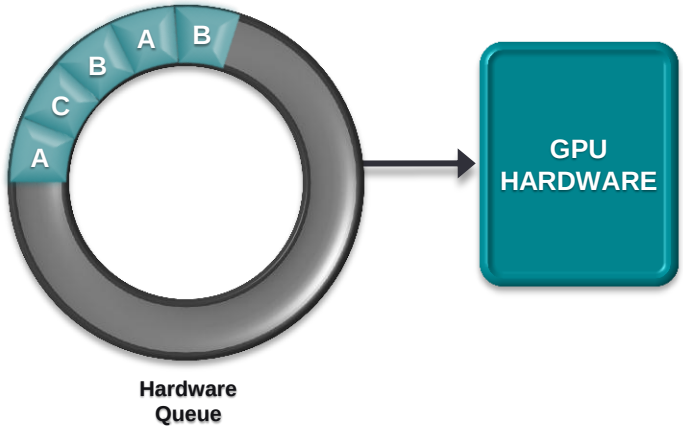


TODAY'S COMMAND AND DISPATCH FLOW

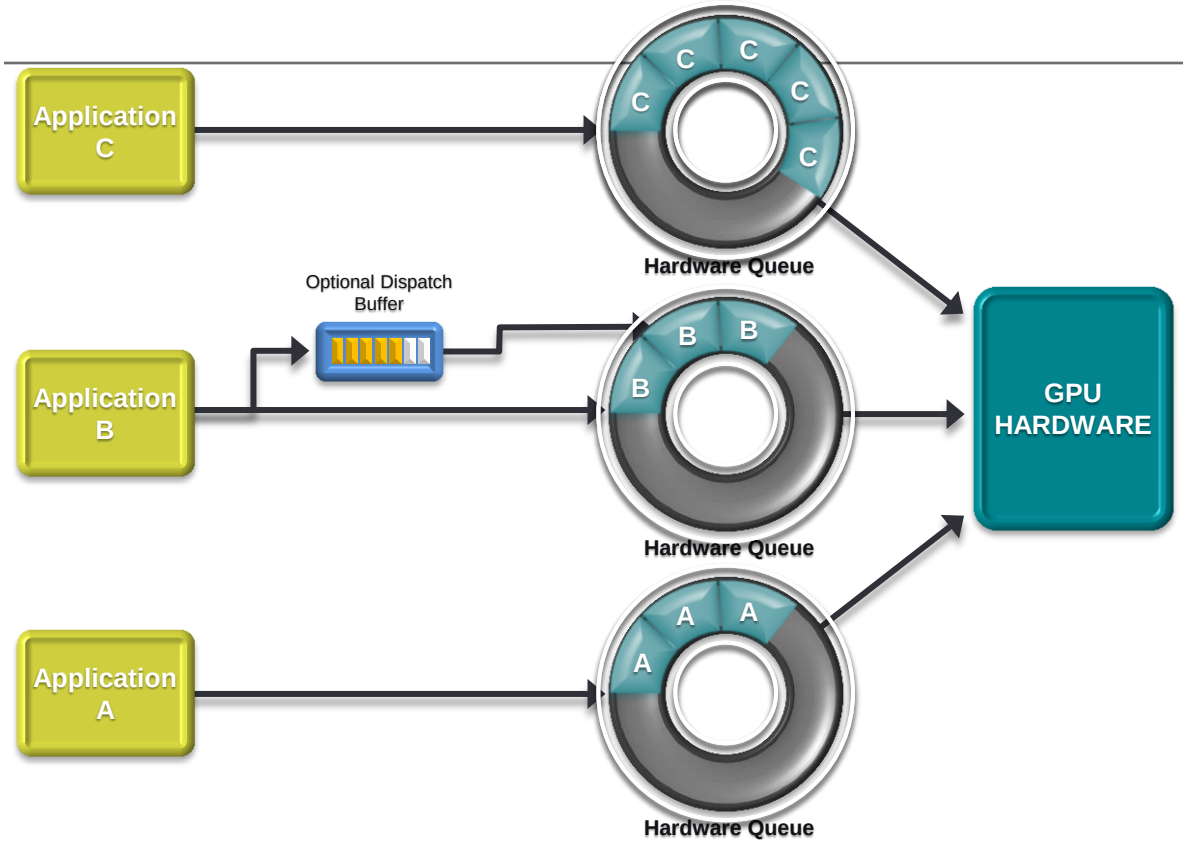


TODAY'S COMMAND AND DISPATCH FLOW





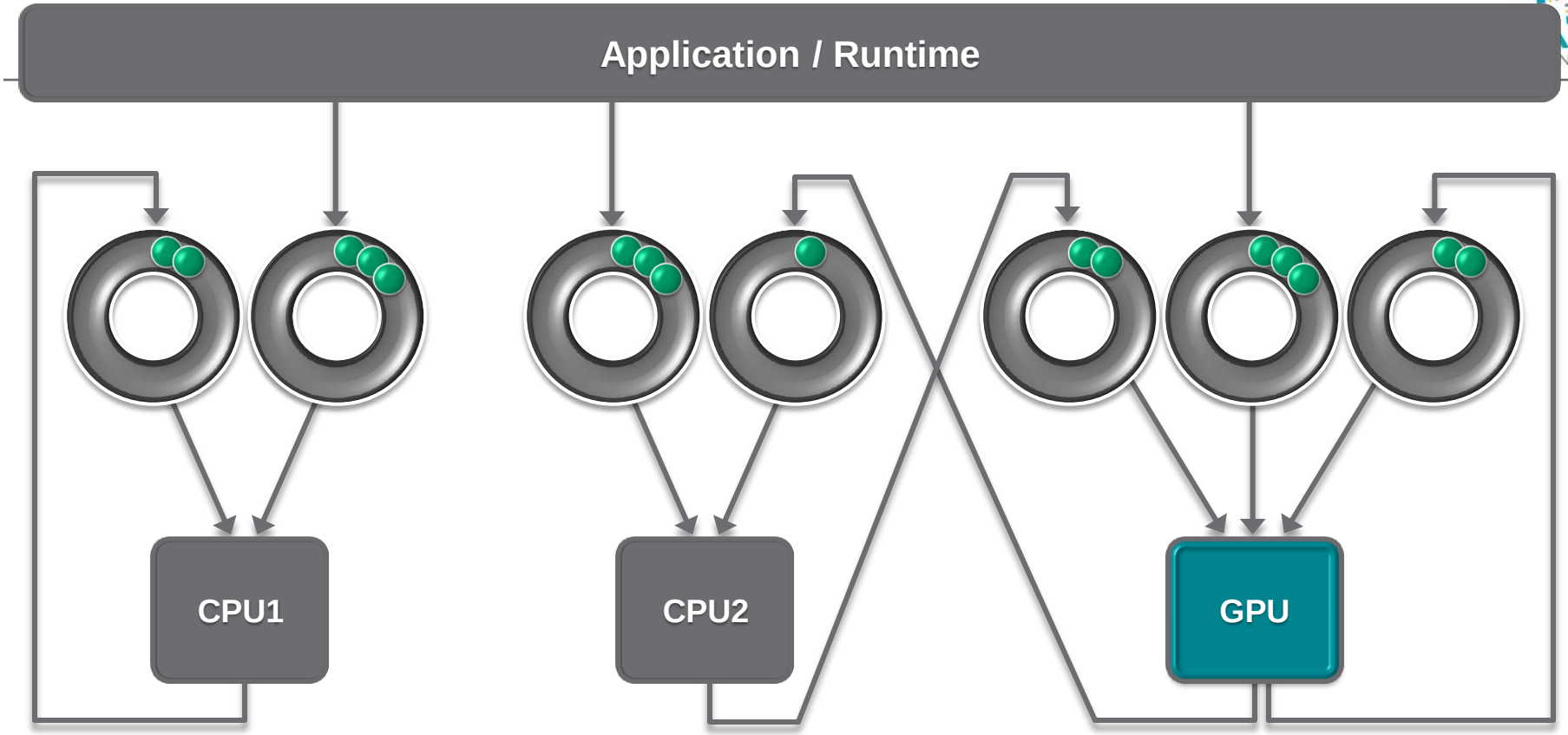
HSA COMMAND AND DISPATCH FLOW

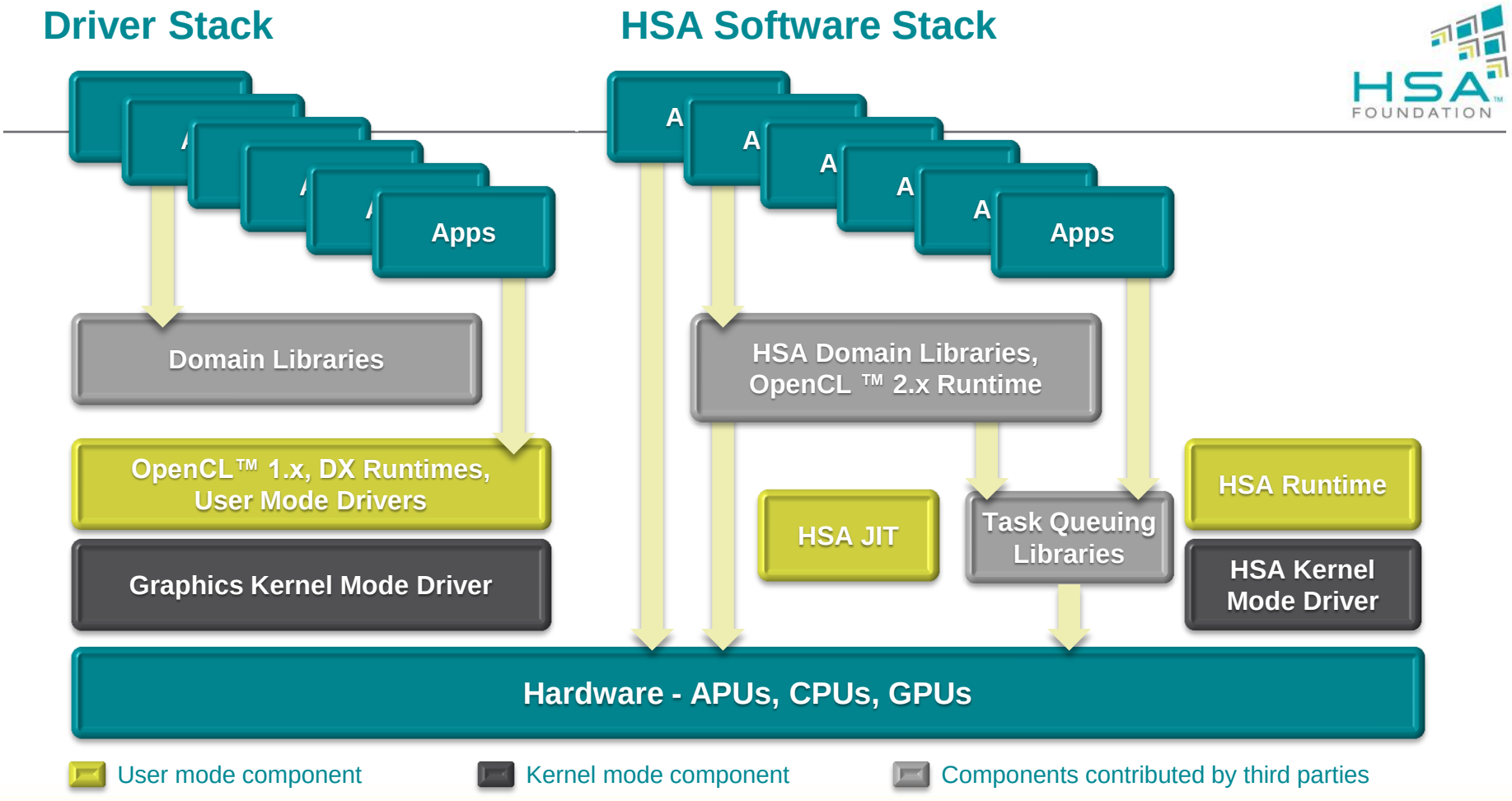


- Application codes to the hardware
- User mode queuing
- Hardware scheduling
- Low dispatch times

- No APIs
- No Soft Queues
- No User Mode Drivers
- No Kernel Mode Transitions
- No Overhead!

COMMAND AND DISPATCH CPU <-> GPU





ACCELERATED WORKLOADS

CLIENT AND SERVER EXAMPLES

HAAR Face Detection

CORNERSTONE TECHNOLOGY FOR COMPUTERVISION

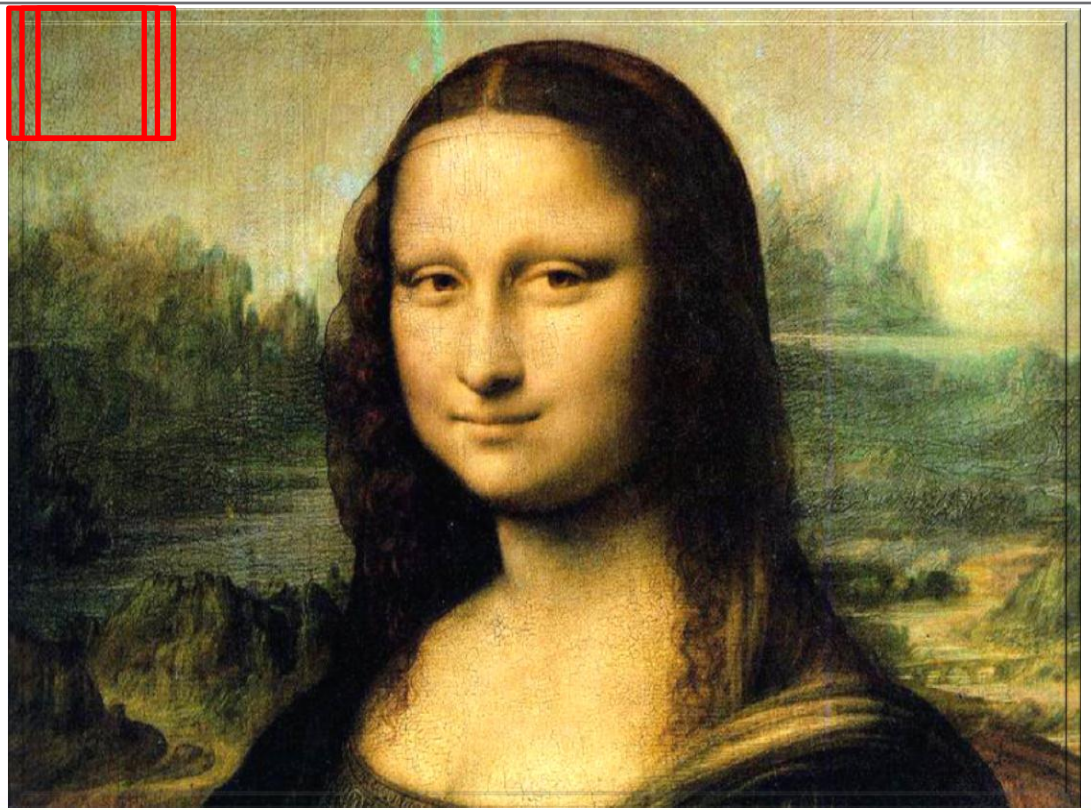
LOOKING FOR FACES IN ALL THE RIGHT PLACES

Quick HD Calculations

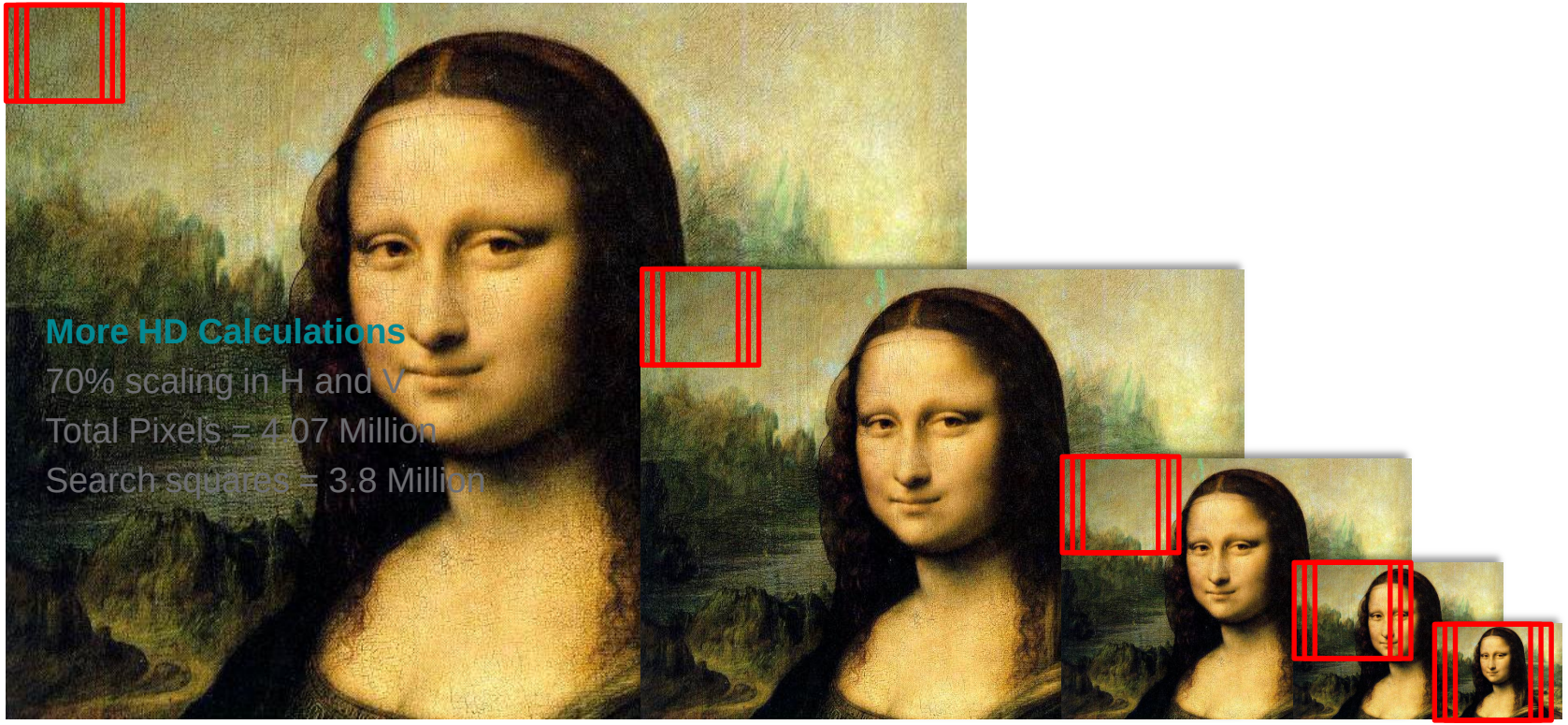
Search square = 21 x 21

Pixels = 1920 x 1080 = 2,073,600

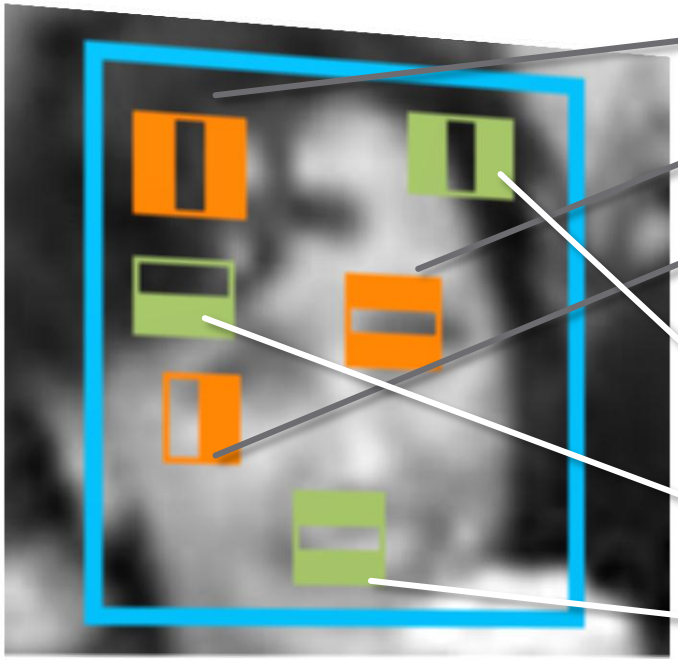
Search squares = 1900 x 1060 = ~2 Million



LOOKING FOR DIFFERENT SIZE FACES – BY SCALING THE VIDEO FRAME



HAAR CASCADE STAGES



Feature k

Feature l

Feature m

Stage N

Face still possible?

Yes

No

Feature p

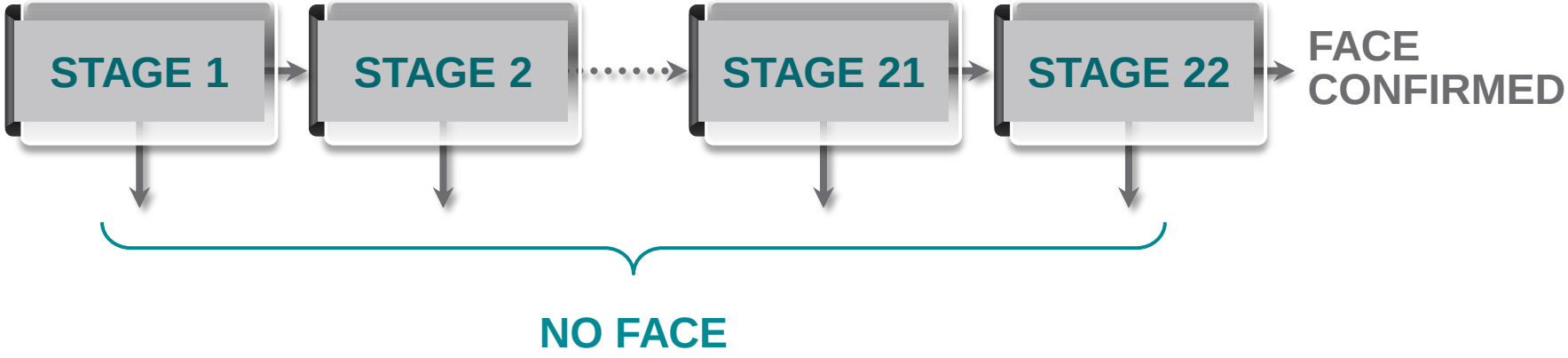
Feature r

Feature q

Stage N+1

REJECT FRAME

22 CASCADE STAGES, EARLY OUT BETWEEN EACH



Final HD Calculations

Search squares = 3.8 million

Average features per square = 124

Calculations per feature = 100

Calculations per frame = 47 GCalcs

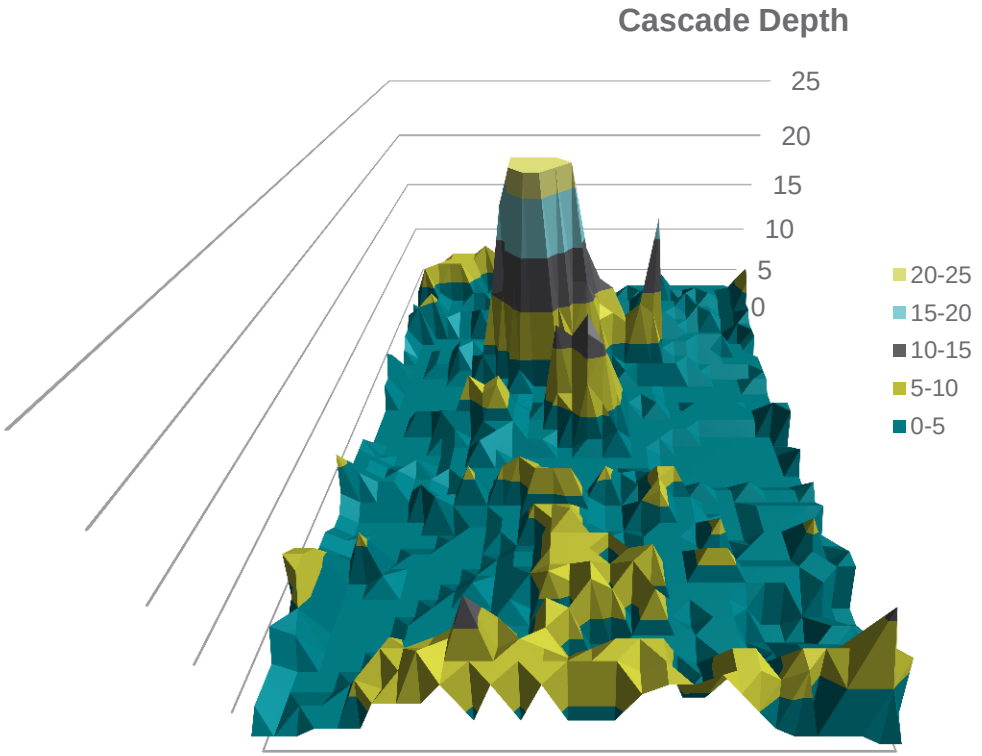
Calculation Rate

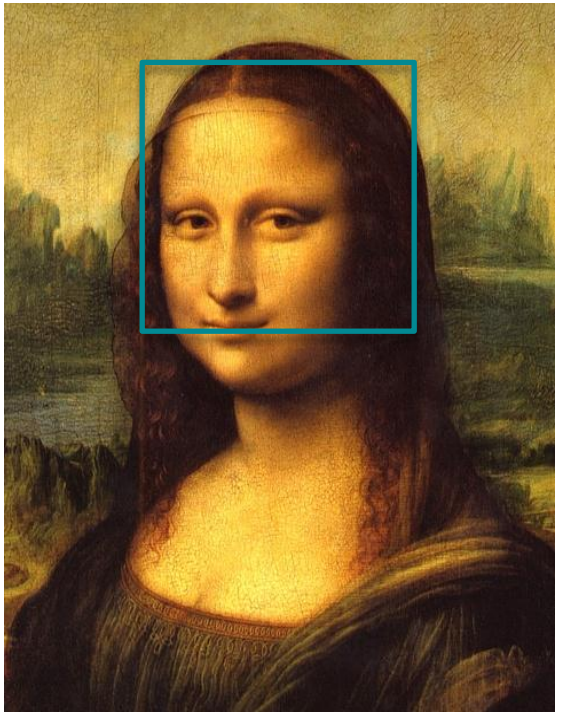
30 frames/sec = 1.4TCalcs/second

60 frames/sec = 2.8TCalcs/second

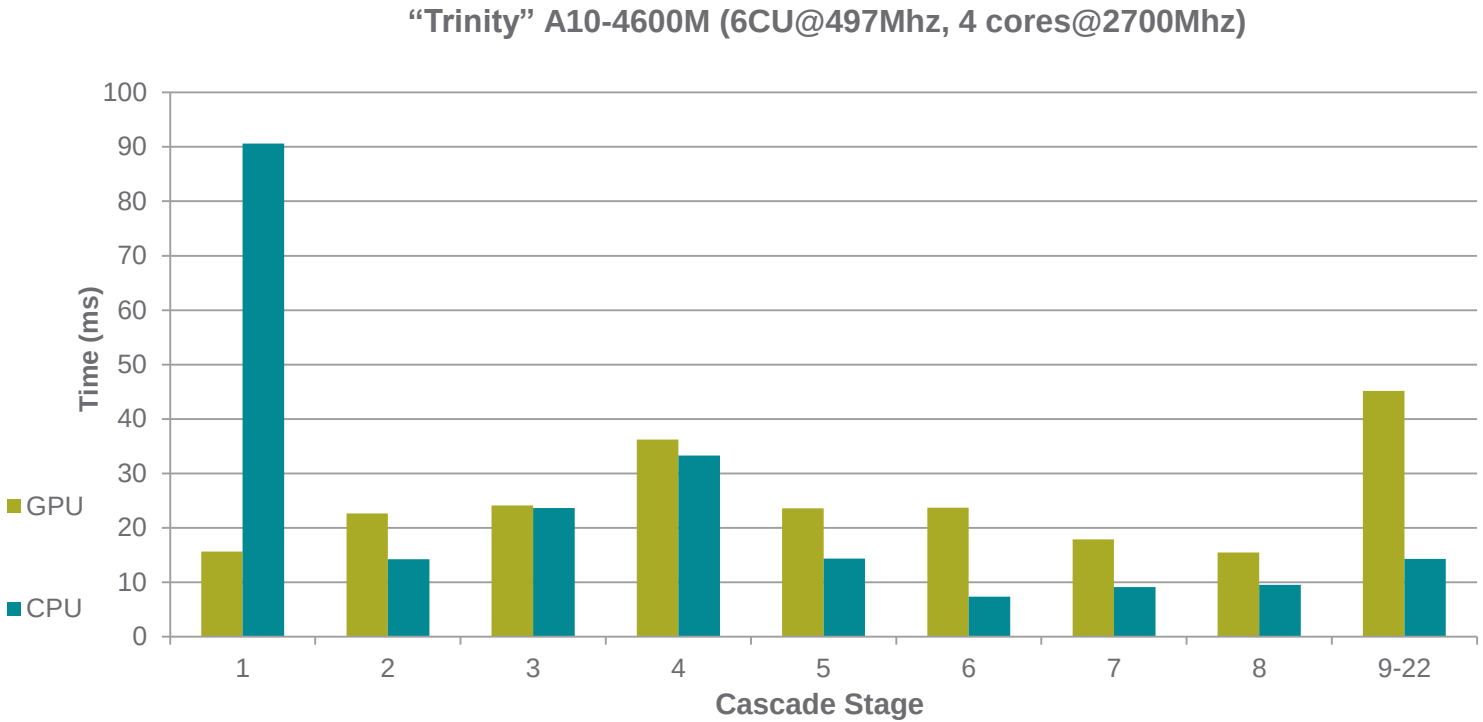
…and this only gets front-facing faces

CASCADE DEPTH ANALYSIS



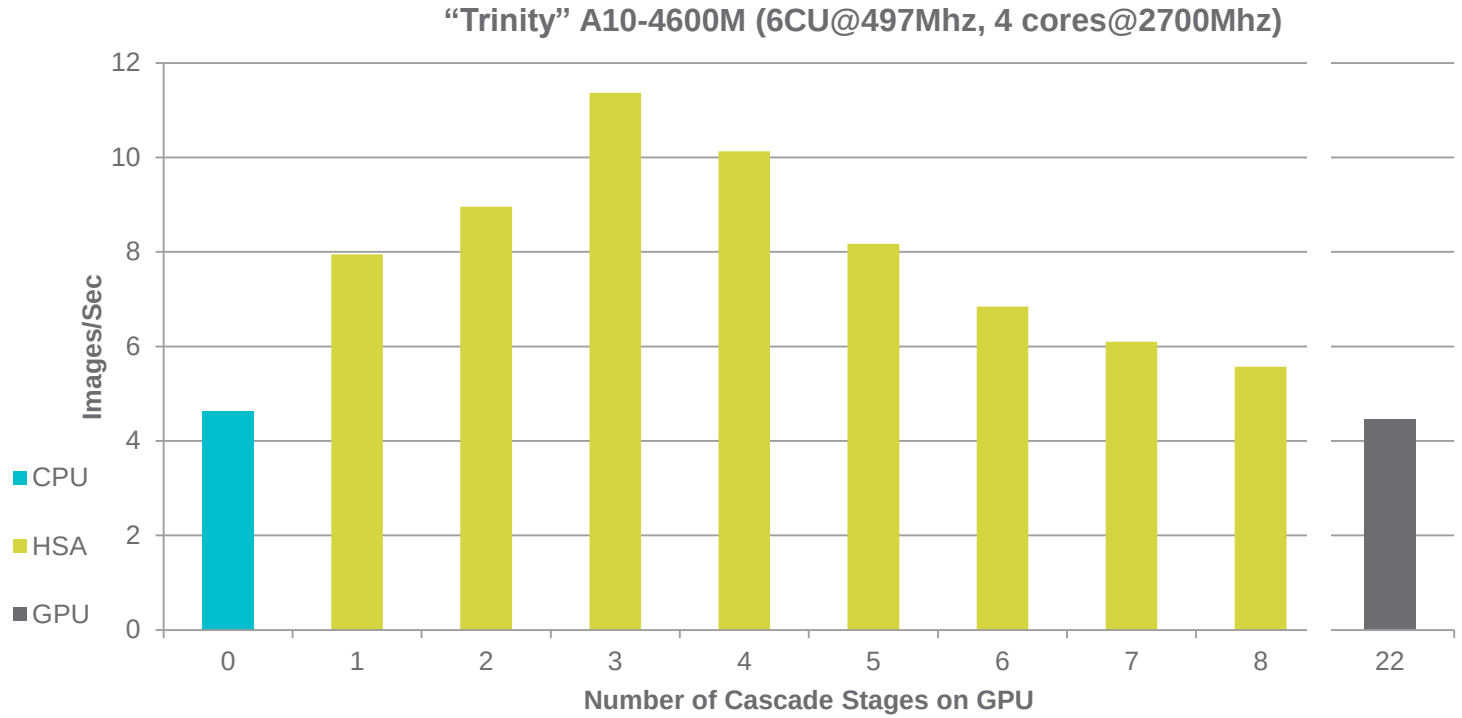


PROCESSING TIME/STAGE



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; Windows 7 (64-bit); OpenCL™ 1.1 (873.1)

PERFORMANCE CPU-VS-GPU



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; Windows 7 (64-bit); OpenCL™ 1.1 (873.1)

HAAR SOLUTION – RUN DIFFERENT CASCADES ON GPU AND CPU

By seamlessly sharing data between CPU and GPU, HSA allows the right processor to handle its appropriate workload

INCREASED PERFORMANCE

DECREASED ENERGY PER FRAME

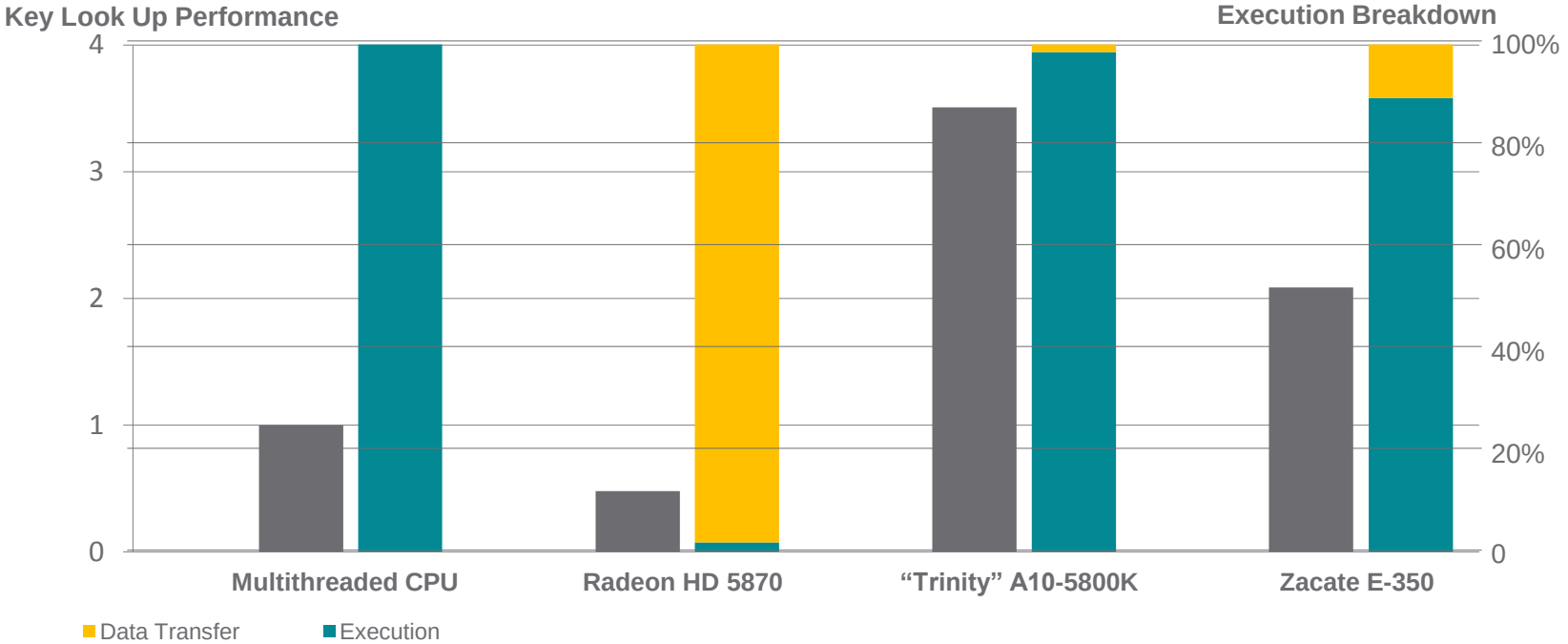
ACCELERATING MEMCACHED

CLOUD SERVER WORKLOAD

MEMCACHED

- A Distributed Memory Object Caching System Used in Cloud Servers
- Generally used for short-term storage and caching, handling requests that would otherwise require database or file system accesses
- Used by Facebook, YouTube, Twitter, Wikipedia, Flickr, and others
- Effectively a large distributed hash table
  - Responds to store and get requests received over the network
  - Conceptually:
    - store(key, object)
    - object = get(key)

OFFLOADING MEMCACHED KEY LOOKUP TO THE GPU



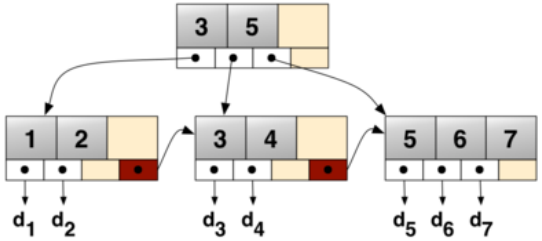
T. H. Hetherington, T. G. Rogers, L. Hsu, M. O'Connor, and T. M. Aamodt, "Characterizing and Evaluating a Key-Value Store Application on Heterogeneous CPU-GPU Systems," *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2012)*, April 2012.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6189209

ACCELERATING B+TREE SEARCHES

CLOUD SERVER WORKLOAD

B+TREE SEARCHES

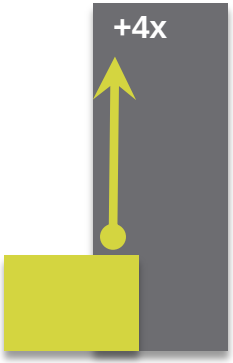
- B+Trees are a fundamental data structure
  - Used to reduce memory & disk access to locate a key
  - Can support index- and range-based queries
  - Can be updated efficiently
- B+Trees are used by enterprise DB applications
  - SQL: SQLite, MySQL, Oracle, and others
  - No-SQL: Apache CouchDB, Tokyo Cabinet, and others
    - Audio search, video copy detection



A simple B+Tree linking the keys 1-7. The linked list (red) allows rapid in-order traversal.

PARALLEL B+TREE SEARCHES ON HSA

By efficiently sharing data between CPU and GPU, HSA increases performance versus Multi Threaded CPU, *even for tree structures that reside in host memory.*



INCREASED PERFORMANCE

With HSA, DB can be larger than GPU memory, and can be shared.

- HSA lets us move compute to data
  - Parallel search can move to GPU
  - Sequential updates can remain on CPU

Platform	Size < 1.5 GB	Size 1.5-2.7 GB	Size > 2.7 GB
dGPU (memory size = 3GB)	✓	✓	✗
HSA	✓	✓	✓

M. Daga, and M. Nutter, "Exploiting Coarse-Grained Parallelism in B+Tree Searches on an APU", Accepted at "*Second Workshop on Irregular Applications: Algorithms and Architectures*, (IA3)" November 2012.

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

ACCELERATING JAVA

GOING BEYOND NATIVE LANGUAGES

JAVA ENABLEMENT BY APARAPI

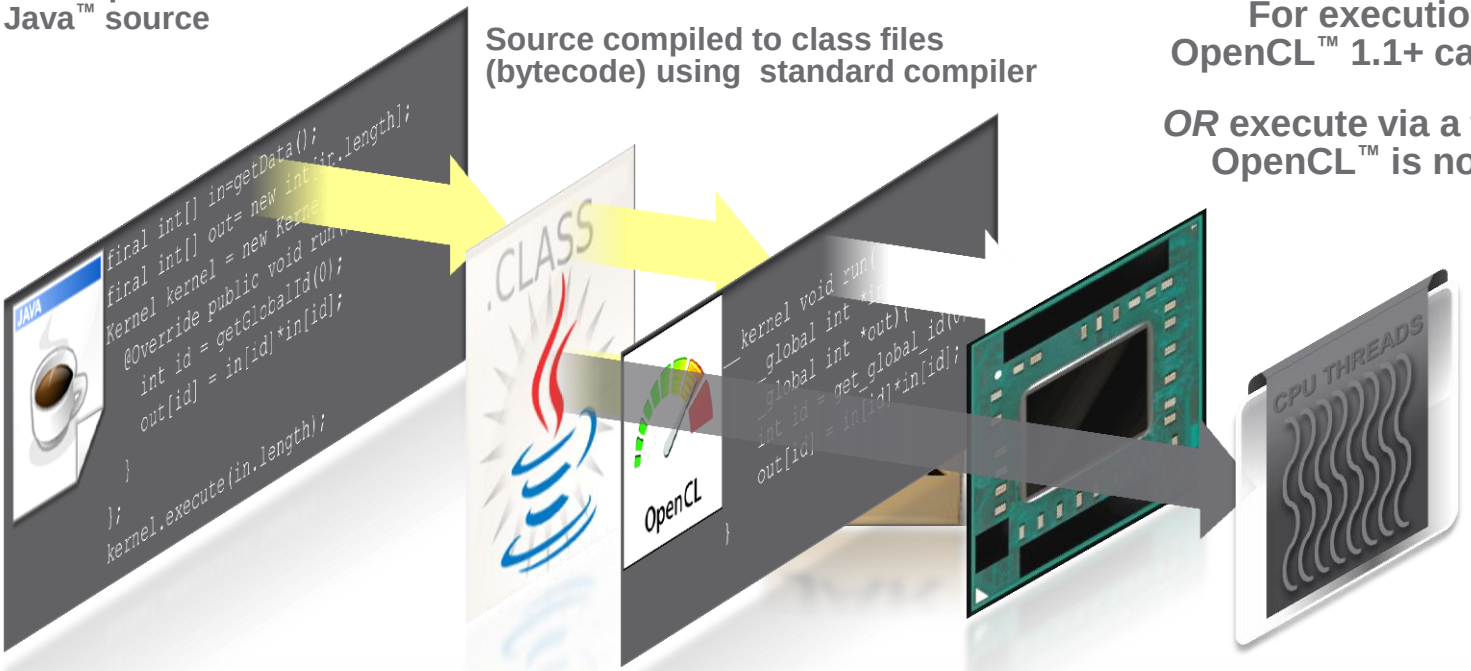
Aparapi = Runtime capable of converting Java™ bytecode to OpenCL™

Developer creates Java™ source

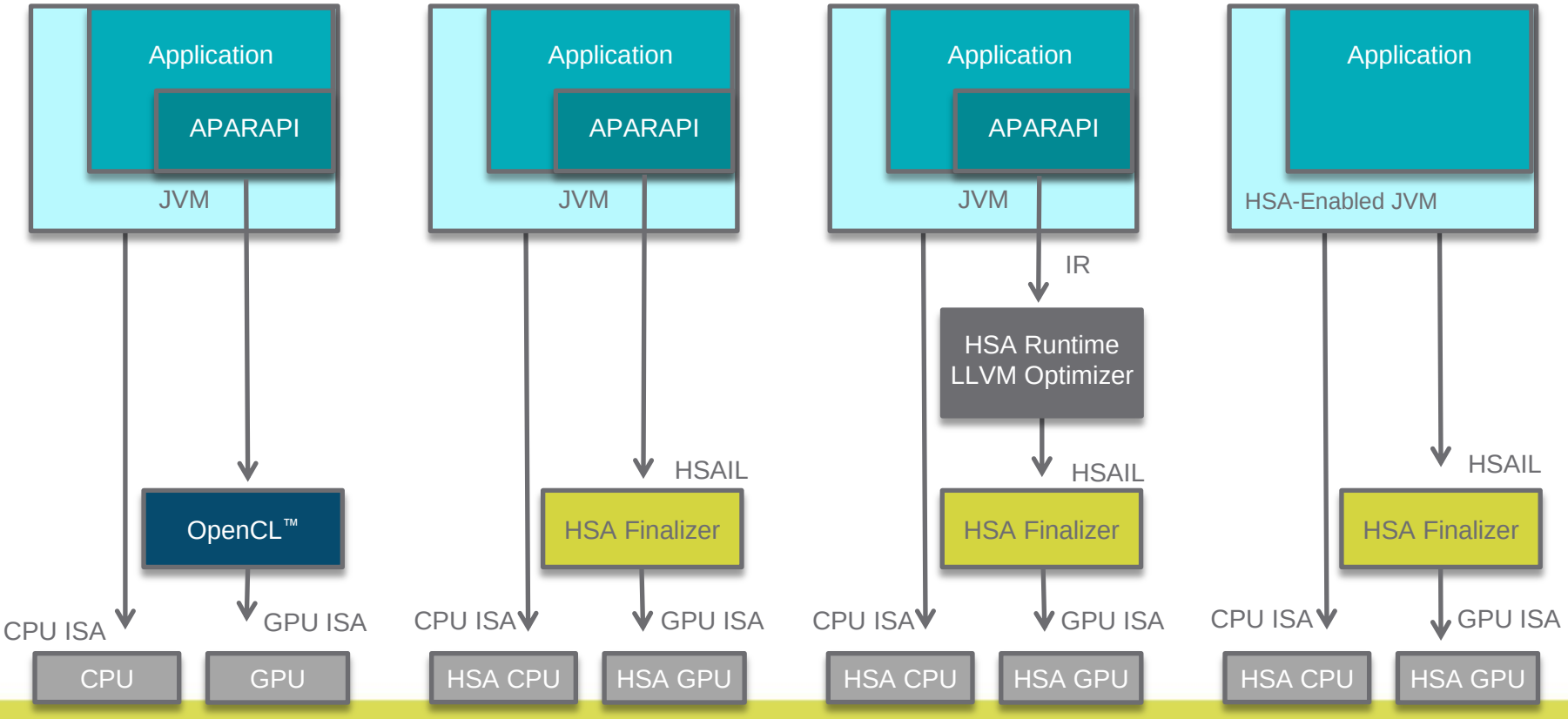
Source compiled to class files (bytecode) using standard compiler

For execution on any OpenCL™ 1.1+ capable device

***OR* execute via a thread pool if OpenCL™ is not available**



JAVA AND APARAPI HSA ENABLEMENT ROADMAP



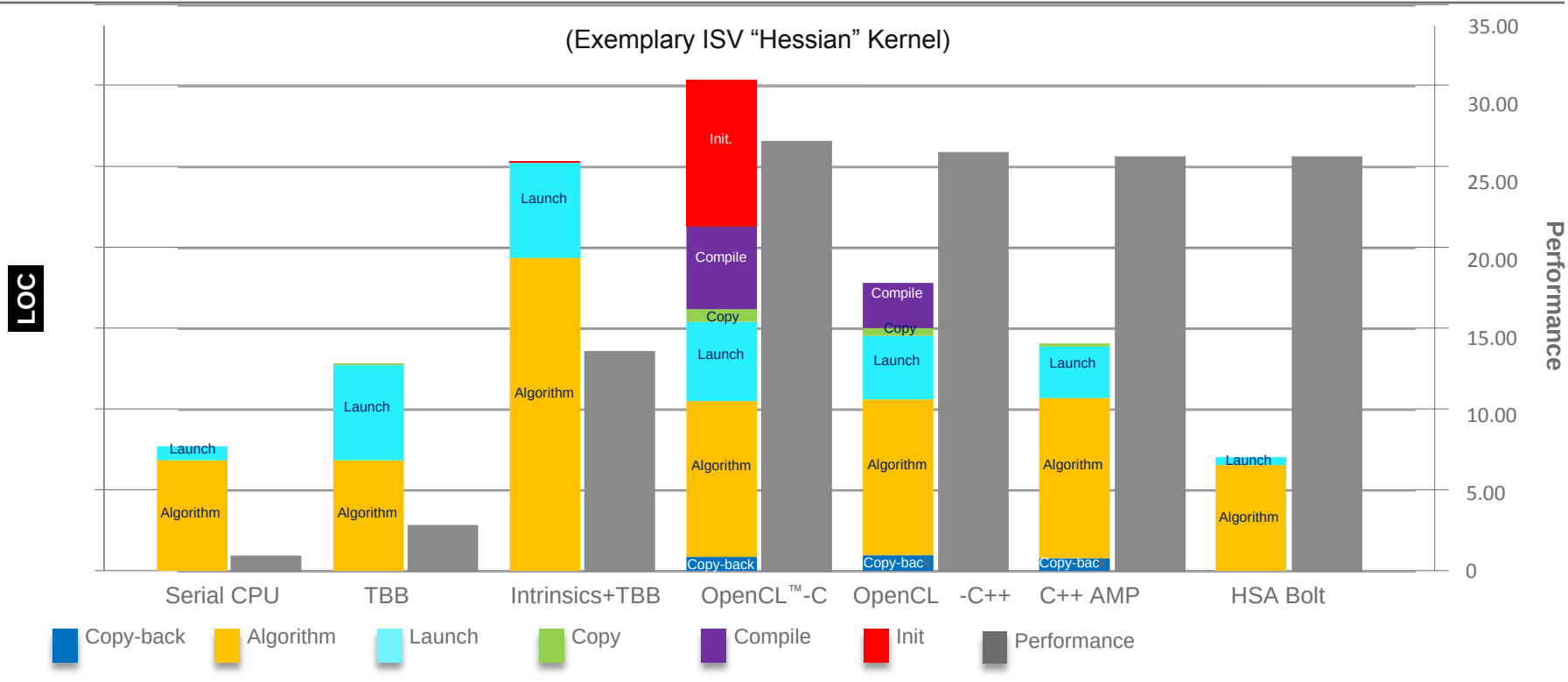
EASE OF PROGRAMMING

CODE COMPLEXITY VS. PERFORMANCE

BOLT

- Optimized template library routines for common GPU functions
  - For OpenCL™ and C++ AMP, across multiple platforms
- Programming model interface similar to multicore Task Parallel Runtimes (TBB, ConCRT)
- CPU performance as good or better than multicore Task Parallel Runtimes
- Excellent performance and power efficiency on HSA Devices
- For many applications, single source code base for both CPU and GPU !
- Leverage robust Visual Studio C++AMP debug solution

LINES-OF-CODE AND PERFORMANCE FOR DIFFERENT PROGRAMMING MODELS

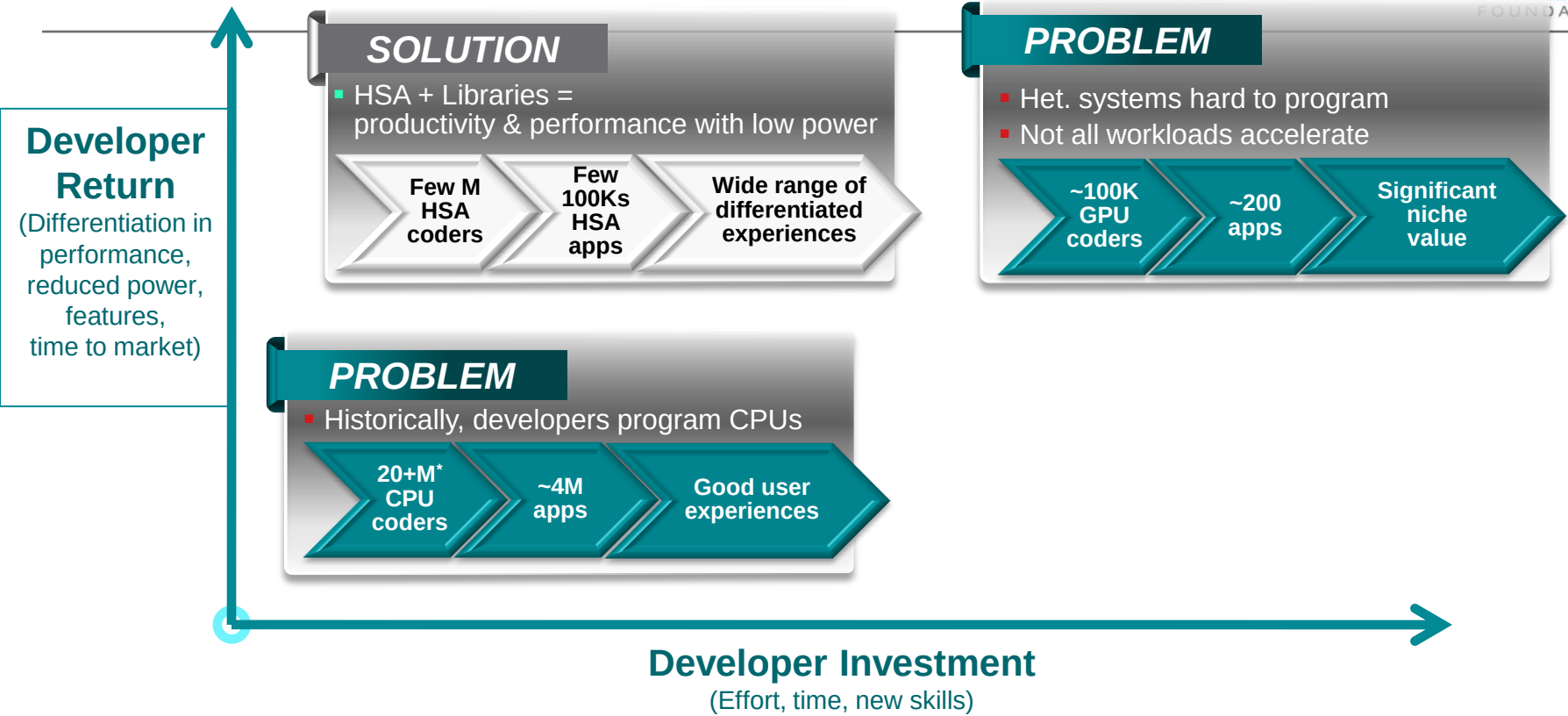


AMD A10-5800K APU with Radeon™ HD Graphics – CPU: 4 cores, 3800MHz (4200MHz Turbo); GPU: AMD Radeon HD 7660D, 6 compute units, 800MHz; 4GB RAM.
Software – Windows 7 Professional SP1 (64-bit OS); AMD OpenCL™ 1.2 AMD-APP (937.2); Microsoft Visual Studio 11 Beta

RESEARCH TOPICS IN HSA

Category	Description	Comments
Languages/Compilers	Higher-level languages. GPU languages are primitive today. OpenCL is a good expert tool. Look into domain specific languages (graphics, math). Ex: HSA could have a database accelerator component	
	Split compilation model – high level compliers & low level compilers and how to make them work well together	
	How to run best on a device with multi ISA's	
Software Run-Time	Classic load balancing. Look for new ways to partition algorithms automatically in the runtime. Simultaneous running of multiple kernels or multiple applications. Quality of service & virtualization. Scheduling for complex status graphs and scheduling dynamic parallelism	
System Architecture	• Bandwidth/memory arch (balancing BW with compute) • Load balancing • Memory configurations: Stack memory devices will eventually appear and systems will change around idea of bandwidth. Shared memory stacks – what are the implications? • TCU/LCU ratios	
Hardware	Logical split between split function hardware. • Applying HSA to non-GPU devices (DSPs, FPGAs, etc.) • Heterogeneous conformance optimization - how to run a program that runs well on all different HSA platforms and hardware	
	Memory system design: low cost support for coherency and would give programmers a way to optimize their use of coherence	
	Security: looking into securing systems	
	Efficient synchronization primitives	
	3D graphics pipes – integration with HSA	

THE HSA OPPORTUNITY



*IDC