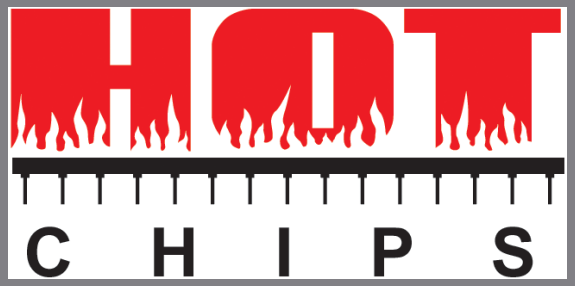


HETEROGENEOUS SYSTEM ARCHITECTURE OVERVIEW

HOT CHIPS TUTORIAL - AUGUST 2013

PHIL ROGERS
HSA FOUNDATION PRESIDENT
AMD CORPORATE FELLOW

HSA FOUNDATION

- Founded in June 2012
- Developing a new platform for heterogeneous systems
- www.hsafoundation.com
- Specifications under development in working groups
- Our first specification, HSA Programmers Reference Manual is already published and available on our web site
- Additional specifications for System Architecture, Runtime Software and Tools are in process

HSA FOUNDATION MEMBERSHIP — AUGUST 2013

Founders	AMD, ARM, Imagination, MediaTek, Qualcomm, Samsung, Texas Instruments
Promoters	LG Electronics
Supporters	Arteris, codeplay, Fabric Engine, MulticoreWare, Sandia National Laboratories
Contributors	Analog Devices, apical, Broadcom, CEVA, Imagination, ITRI, Marvell, Sonics, Sony, ST Ericsson, ST, Swarm64, Symbio, Synopsys, Tensilica, Ubuntu, Vivante
Academic	NTHU Programming Language Lab, NTHU System Software Lab, University of Bristol, University of Edinburgh informatics, Illinois Computer Science, Tsinghua University, Northeastern University, Missouri S&T, Ole Miss
Associates	

SOCS HAVE PROLIFERATED — MAKE THEM BETTER

- SOCs have arrived and are a tremendous advance over previous platforms
- SOCs combine CPU cores, GPU cores and other accelerators, with high bandwidth access to memory
- How do we make them even better?
  - Easier to program
  - Easier to optimize
  - Higher performance
  - Lower power
- HSA unites accelerators architecturally
- Early focus on the GPU compute accelerator, but HSA goes well beyond the GPU

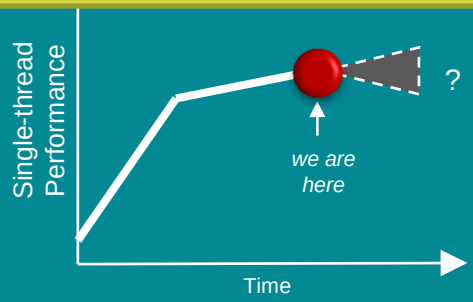
INFLECTIONS IN PROCESSOR DESIGN

Single-Core Era

Enabled by:
- ✓ Moore's Law
- ✓ Voltage Scaling

Constrained by:
- ✘ Power
- ✘ Complexity

Assembly ➡ C/C++ ➡ Java …

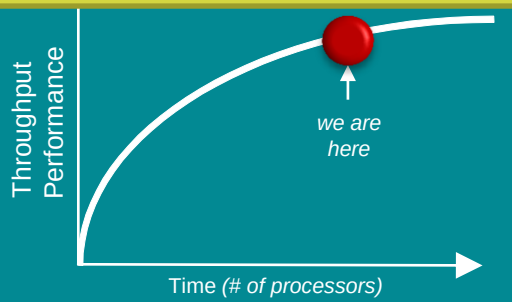


Multi-Core Era

Enabled by:
- ✓ Moore's Law
- ✓ SMP architecture

Constrained by:
- ✘ Power
- ✘ Parallel SW
- ✘ Scalability

pthreads ➡ OpenMP / TBB …

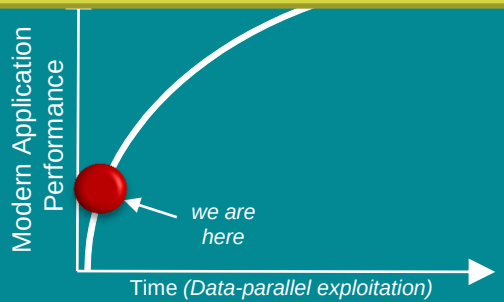


Heterogeneous Systems Era

Enabled by:
- ✓ Abundant data parallelism
- ✓ Power efficient GPUs

Temporarily
Constrained by:
- ✘ Programming models
- ✘ Comm.overhead

Shader ➡ CUDA➡ OpenCL ➡ C++ and Java



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

HSA — AN OPEN PLATFORM

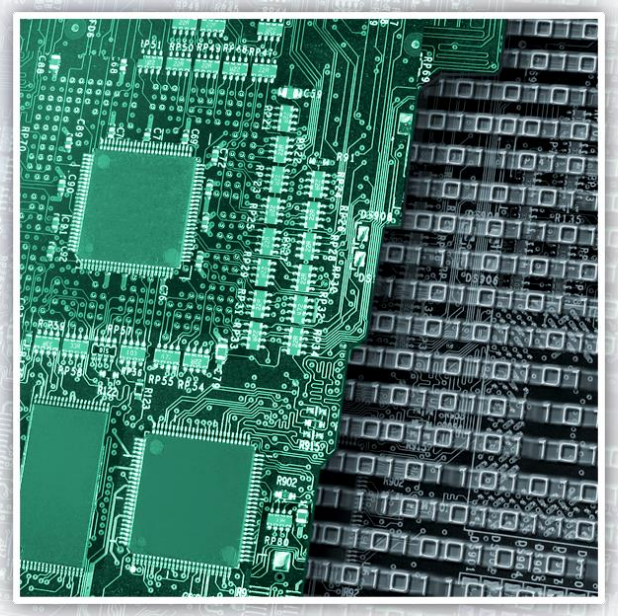
- Open Architecture, membership open to all
  - HSA Programmers Reference Manual
  - HSA System Architecture
  - HSA Runtime
- Delivered via royalty free standards
  - Royalty Free IP, Specifications and APIs
- ISA agnostic for both CPU and GPU
- Membership from all areas of computing
  - Hardware companies
  - Operating Systems
  - Tools and Middleware

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, etc

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

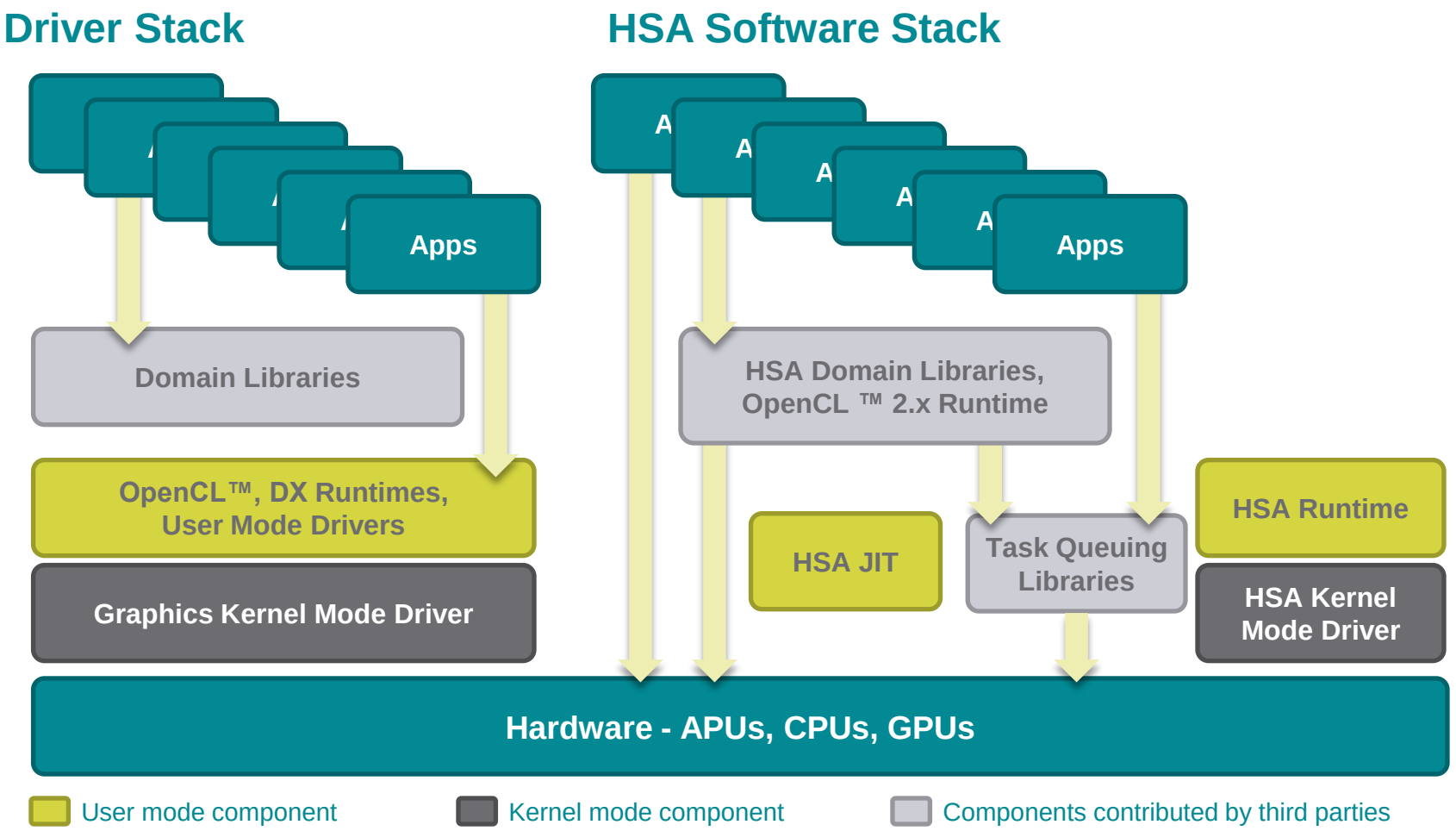


HSA QUEUING MODEL

- 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

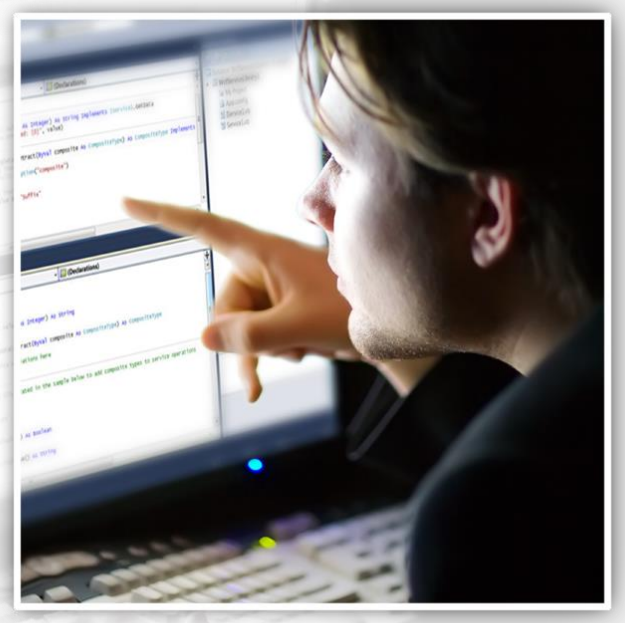
HSA SOFTWARE

TITLE



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
- OpenCL™ 2.0 shows considerable alignment with HSA
  - Many HSA member companies are also active with Khronos in the OpenCL™ working group



BOLT — PARALLEL PRIMITIVES LIBRARY FOR HSA

- Easily leverage the inherent power efficiency of GPU computing
  - Common routines such as scan, sort, reduce, transform
  - More advanced routines like heterogeneous pipelines
  - Bolt library works with OpenCL and C++ AMP
- Enjoy the unique advantages of the HSA platform
  - Move the computation not the data
- Finally a single source code base for the CPU and GPU!
  - Developers can focus on core algorithms
- Bolt version 1.0 for OpenCL and C++ AMP is available now at https://github.com/HSA-Libraries/Bolt

HSA OPEN SOURCE SOFTWARE

- HSA will feature an open source linux execution and compilation stack
  - Allows a single shared implementation for many components
  - Enables university research and collaboration in all areas
  - Because it's the right thing to do

Component Name	IHV or Common	Rationale
HSA Bolt Library	Common	Enable understanding and debug
HSAIL Code Generator	Common	Enable research
LLVM Contributions	Common	Industry and academic collaboration
HSAIL Assembler	Common	Enable understanding and debug
HSA Runtime	Common	Standardize on a single runtime
HSA Finalizer	IHV	Enable research and debug
HSA Kernel Driver	IHV	For inclusion in linux distros

ACCELERATING JAVA

GOING BEYOND NATIVE LANGUAGES

JAVA ENABLEMENT BY APARAPI

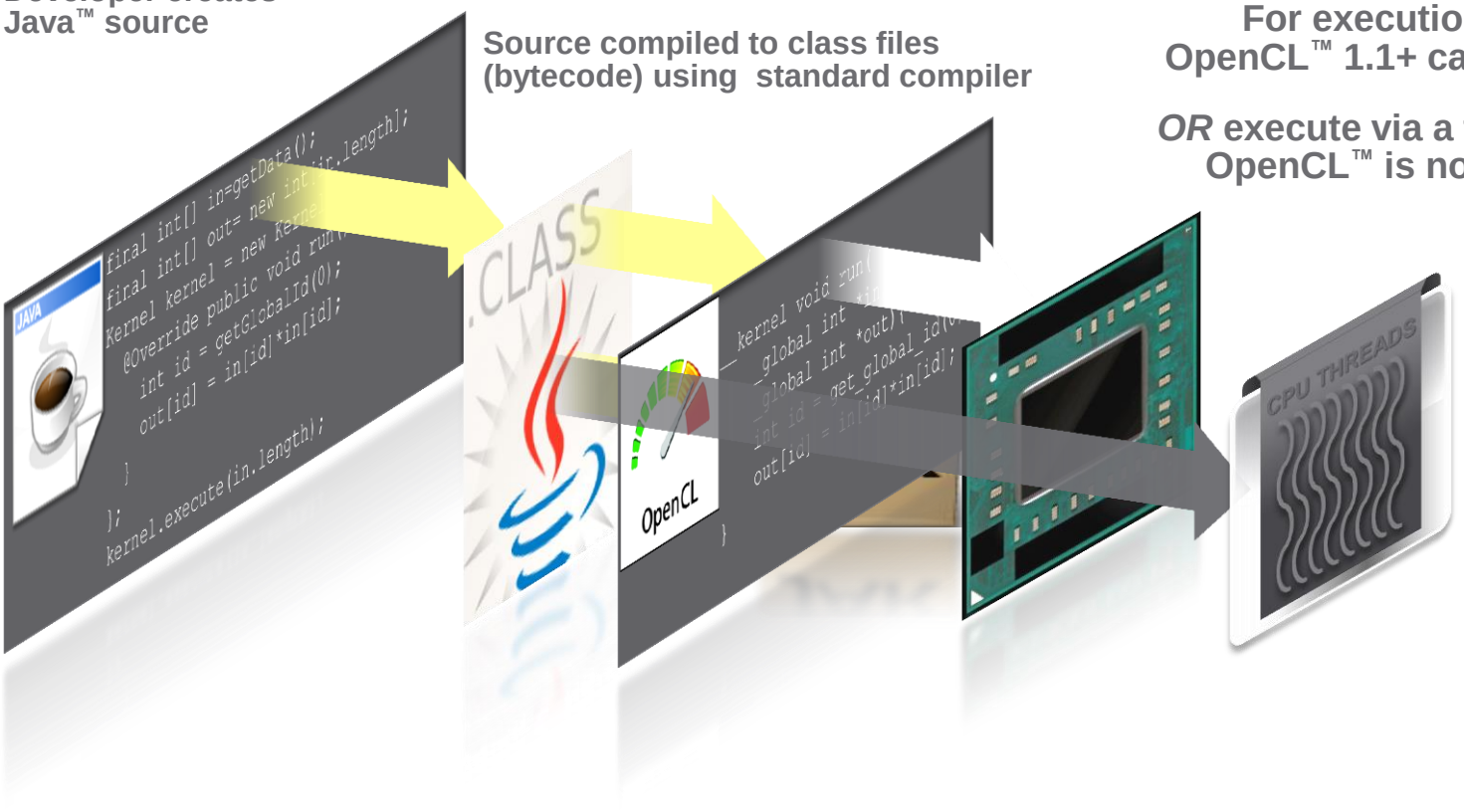
Aparapi = Runtime capable of converting Java™ bytecode to OpenCL™

Developer creates Java™ source

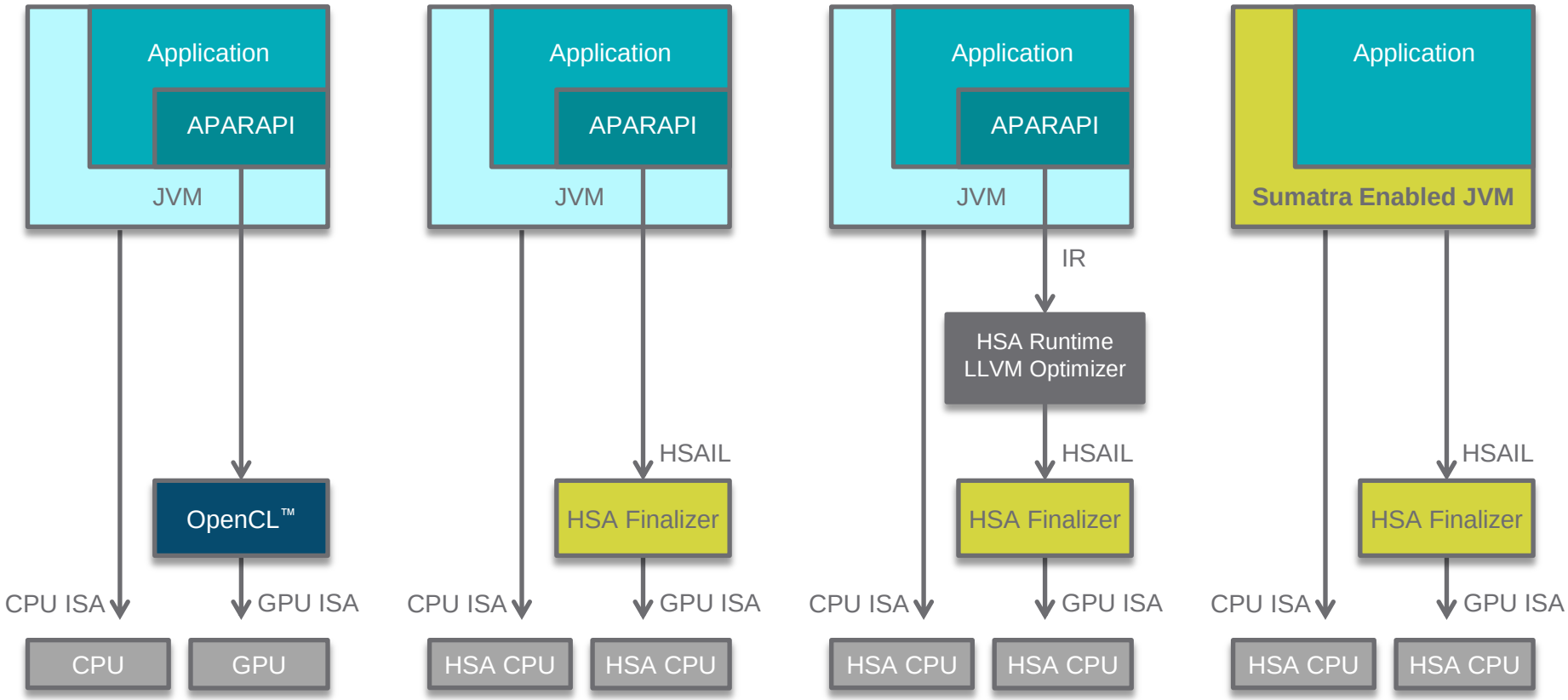
Source compiled to class files (bytecode) using standard compiler

For execution on any OpenCL™ 1.1+ capable device

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

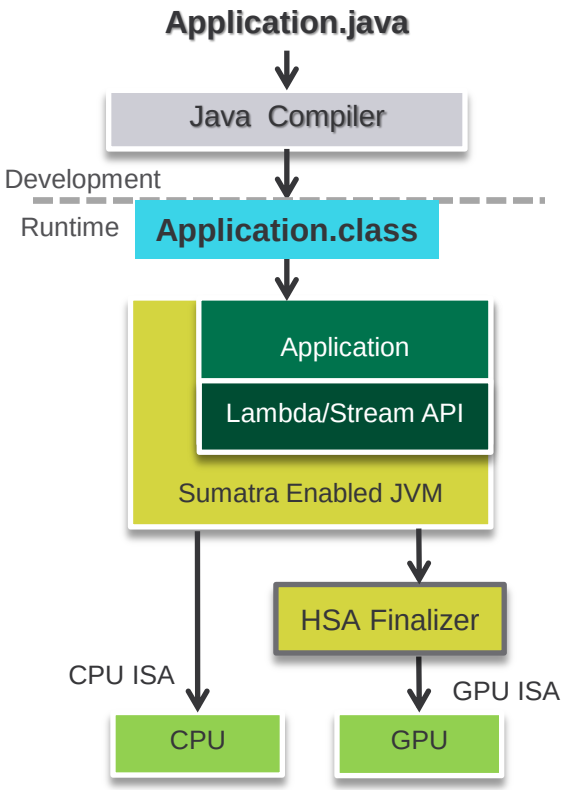


JAVA HETEROGENEOUS ENABLEMENT ROADMAP



SUMATRA PROJECT OVERVIEW

- AMD/Oracle sponsored Open Source (OpenJDK) project
- Targeted at Java 9 (2015 release)
- Allows developers to efficiently represent data parallel algorithms in Java
- Sumatra 'repurposes' Java 8's multi-core Stream/Lambda API's to enable both CPU or GPU computing
- At runtime, Sumatra enabled Java Virtual Machine (JVM) will dispatch 'selected' constructs to available HSA enabled devices
- Developers of Java libraries are already refactoring their library code to use these same constructs
  - So developers using existing libraries should see GPU acceleration without **any** code changes
  - http://openjdk.java.net/projects/sumatra/
  - https://wikis.oracle.com/display/HotSpotInternals/Sumatra
  - http://mail.openjdk.java.net/pipermail/sumatra-dev/




HSA
FOUNDATION
EXAMPLE WORKLOADS

HAAR FACE DETECTION

CORNERSTONE TECHNOLOGY FOR COMPUTERVISION

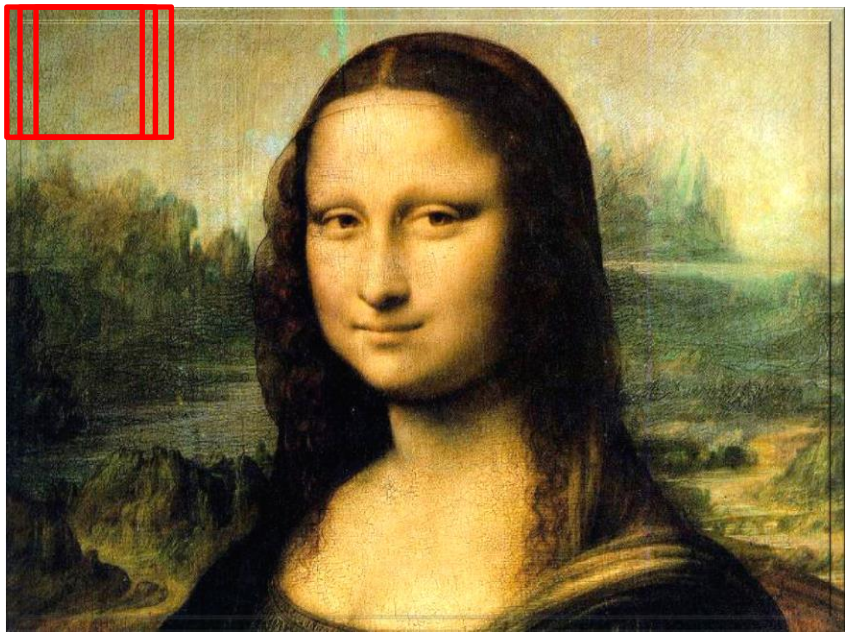
LOOKING FOR FACES IN ALL THE RIGHT PLACES

Quick HD Calculations

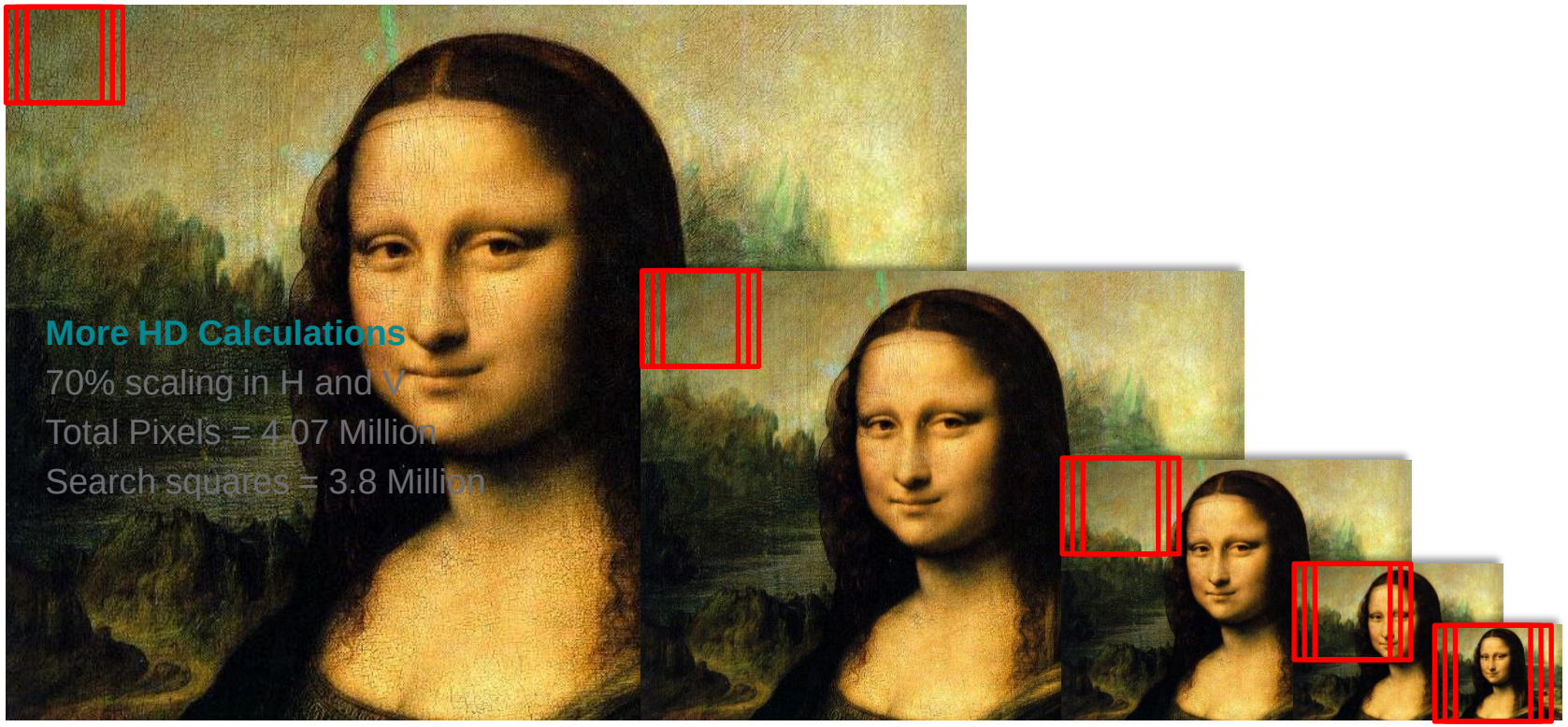
Search square = 21 x 21

Pixels = 1920 x 1080 = 2,073,600

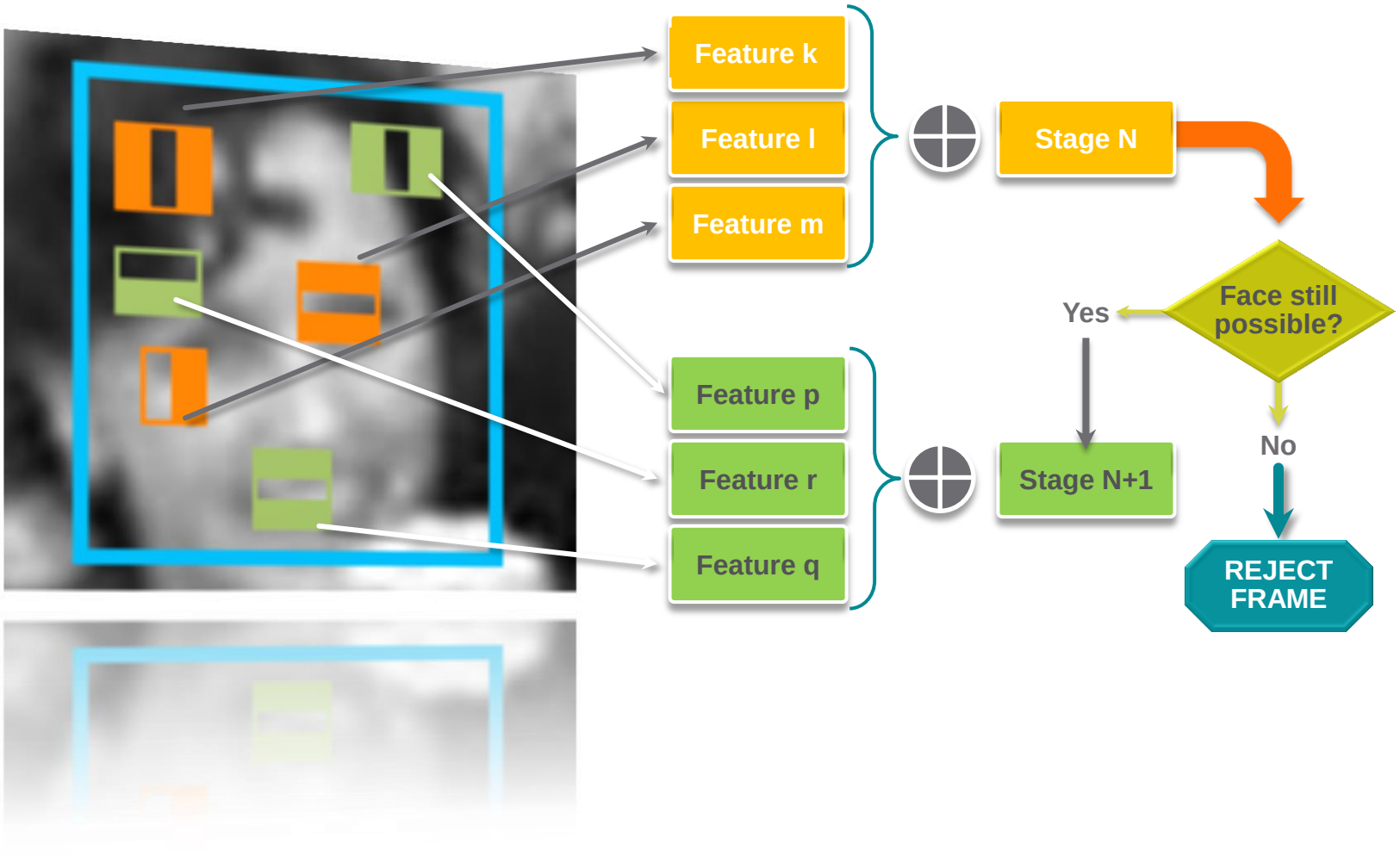
Search squares = 1900 x 1060 = ~2 Million



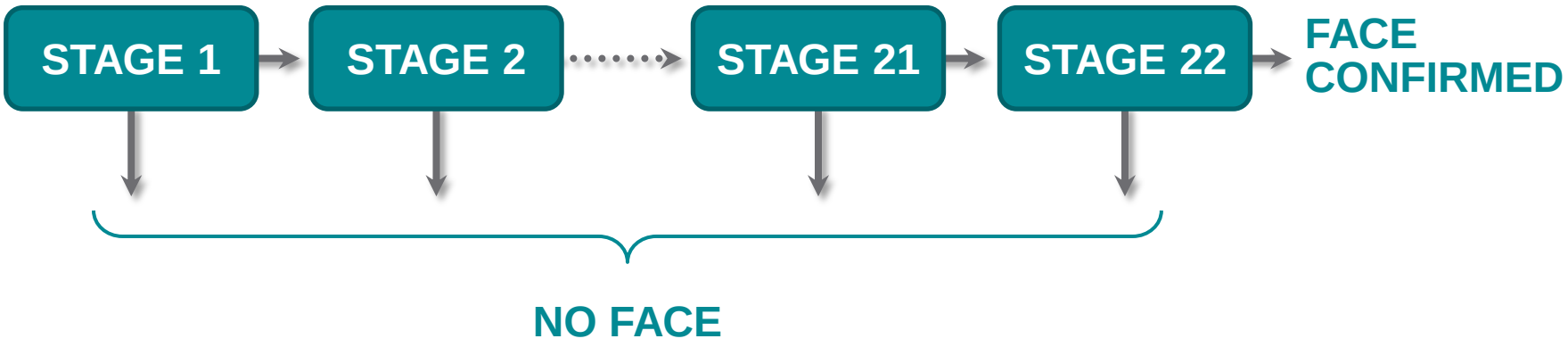
LOOKING FOR DIFFERENT SIZE FACES — BY SCALING THE VIDEO FRAME



HAAR CASCADE STAGES



22 CASCADE STAGES, EARLY OUT BETWEEN EACH



Final HD Calculations

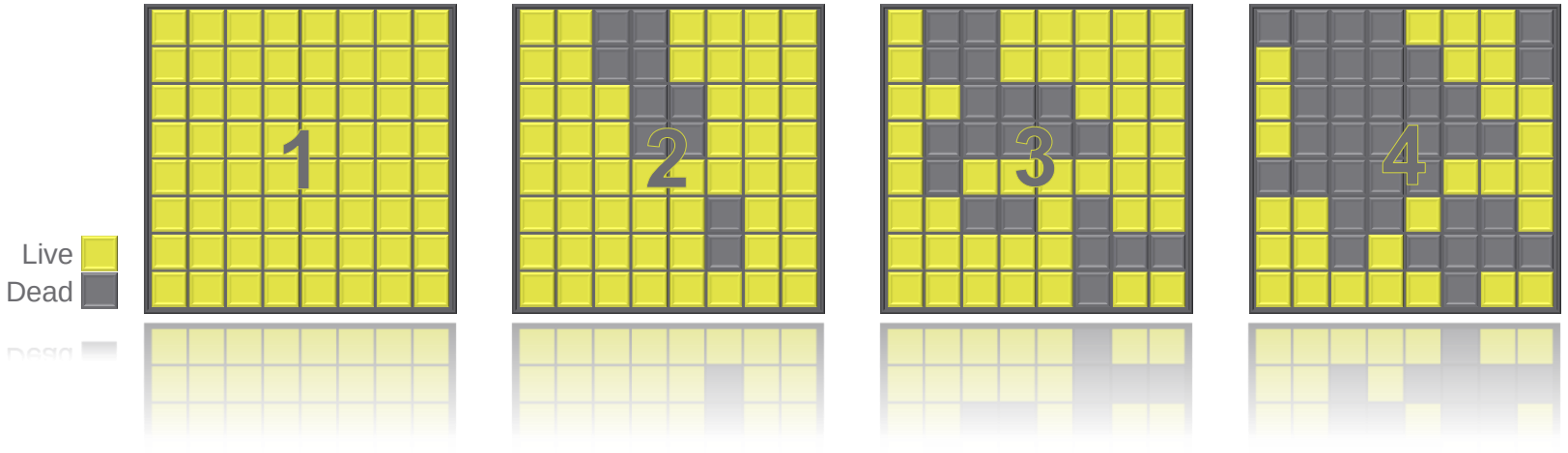
Search squares = 3.8 million
Average features per square = 124
Calculations per feature = 100
Calculations per frame = 47 GCalcs

Calculation Rate

30 frames/sec = 1.4TCalcs/second
60 frames/sec = 2.8TCalcs/second

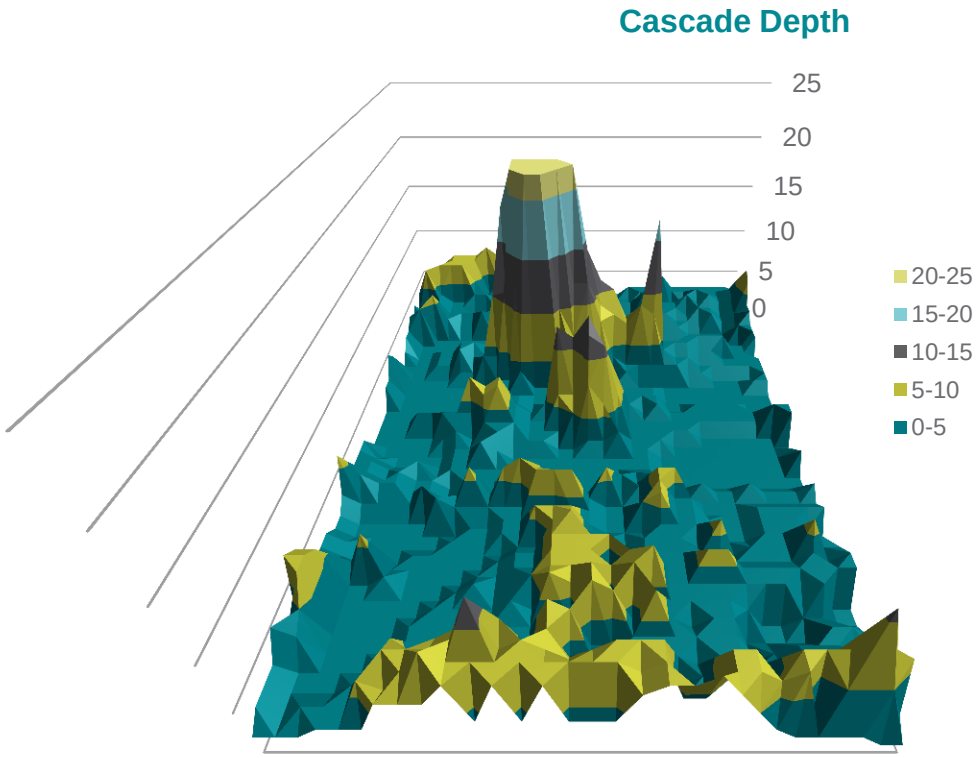
… and this only gets front-facing faces

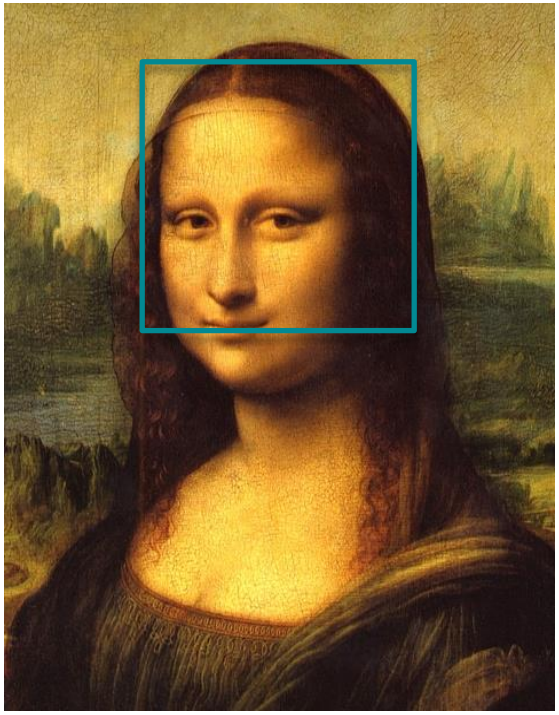
UNBALANCING DUE TO EARLY EXITS



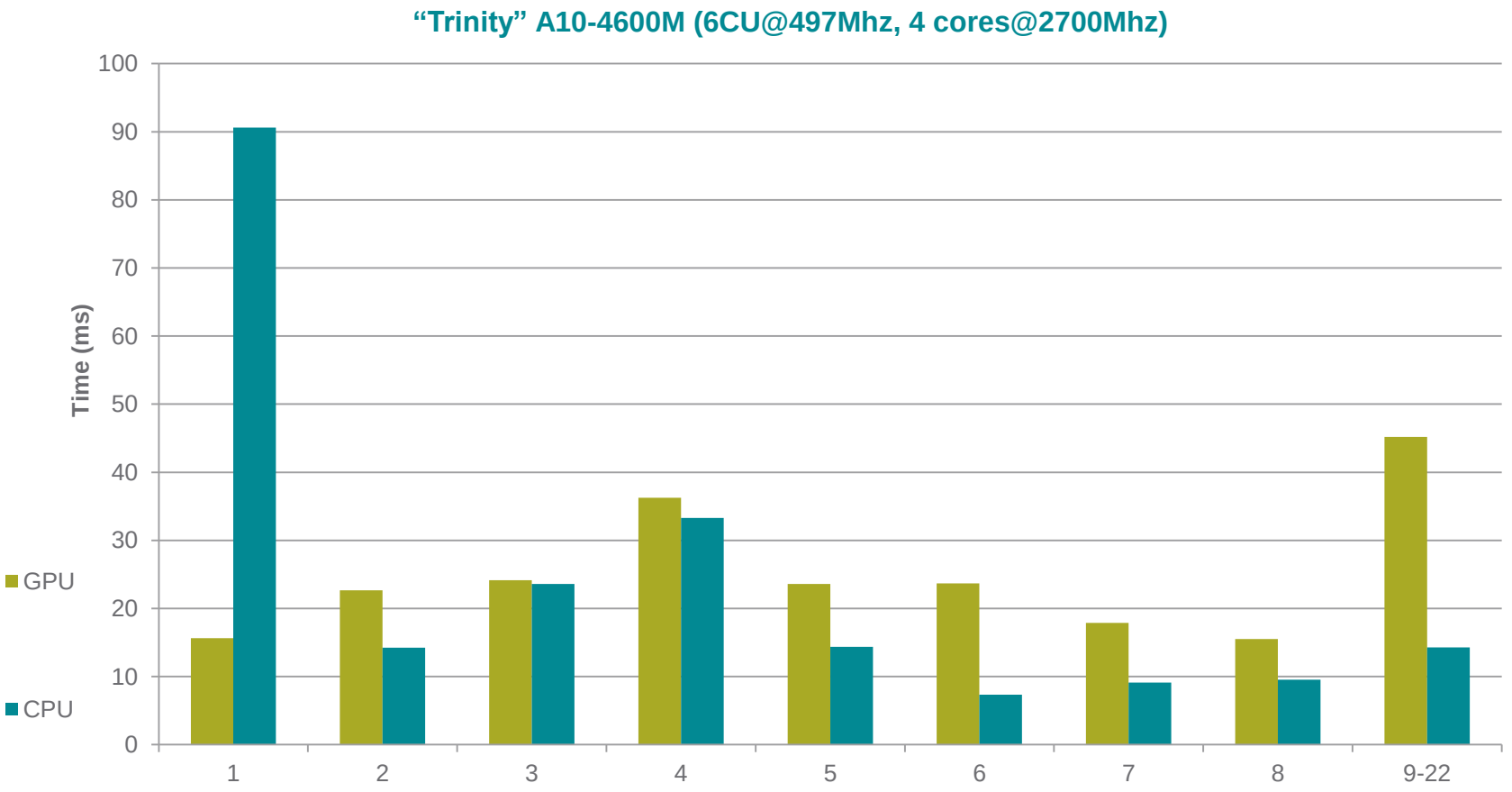
- When running on the GPU, we run each search rectangle on a separate work item
- Early out algorithms, like HAAR, exhibit divergence between work items
  - Some work items exit early
  - Their neighbors continue
  - SIMD packing suffers as a result

CASCADE DEPTH ANALYSIS



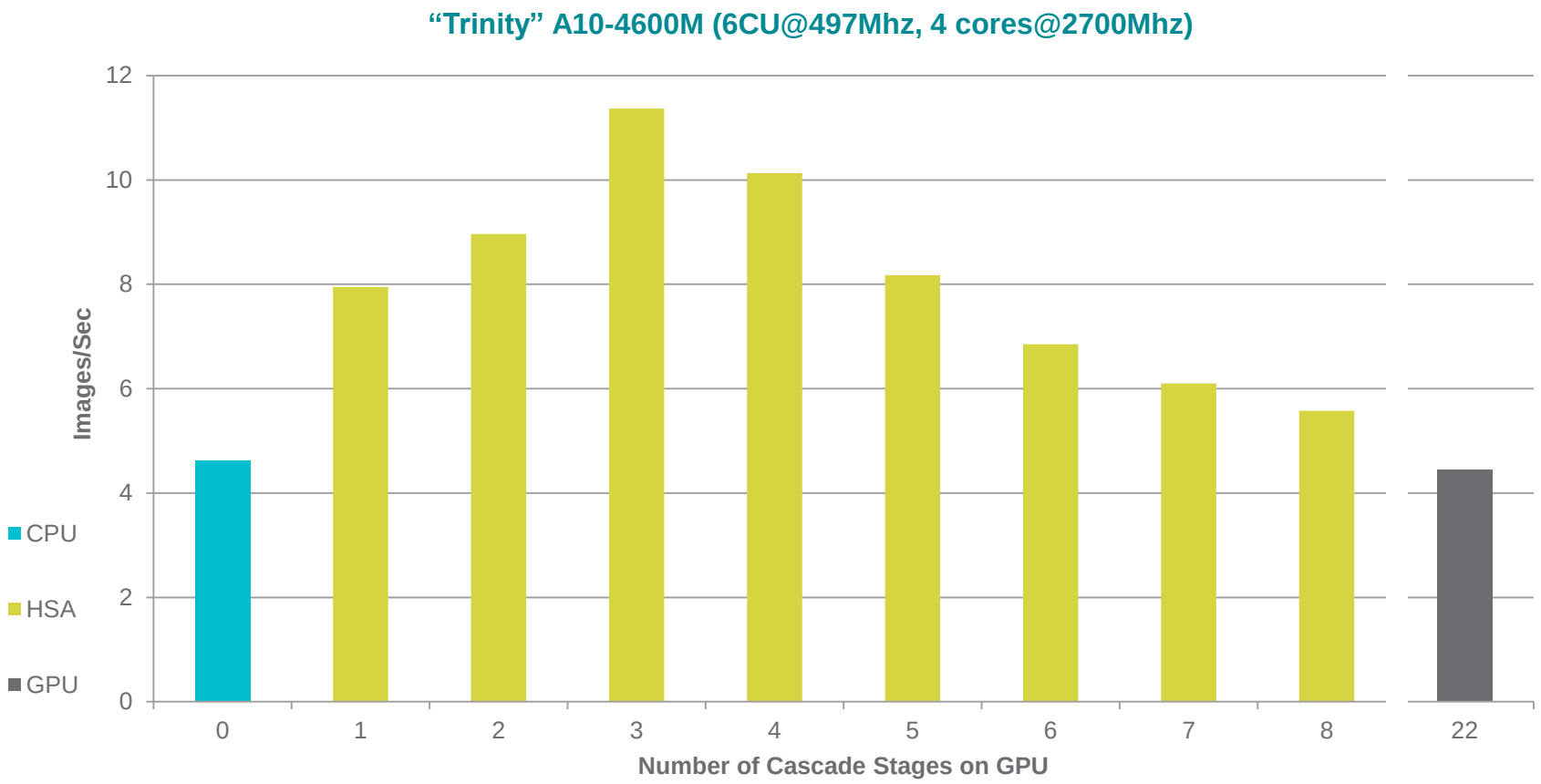


PROCESSING TIME/STAGE



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)

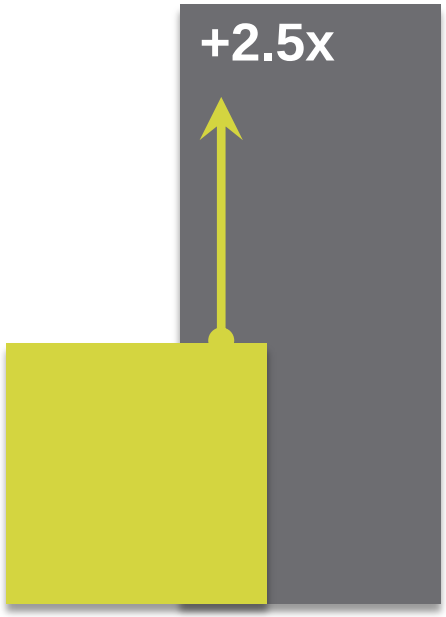
PERFORMANCE CPU-VS-GPU



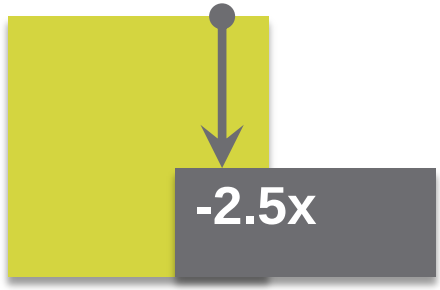
AMD A10 4600M APU with Radeon™ HD Graphics; CPU: 4 cores @ 2.3 MHz (turbo 3.2 GHz); GPU: AMD Radeon HD 7660G, 6 compute units, 685MHz; 4GB RAM; Windows 7 (64-bit); OpenCL™ 1.1 (873.1)

HAAR SOLUTION — RUN DIFFERENT CASCADES ON GPU AND CPU

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



INCREASED PERFORMANCE



DECREASED ENERGY PER FRAME

ACCELERATING SUFFIX ARRAY CONSTRUCTION

CLOUD SERVER WORKLOAD

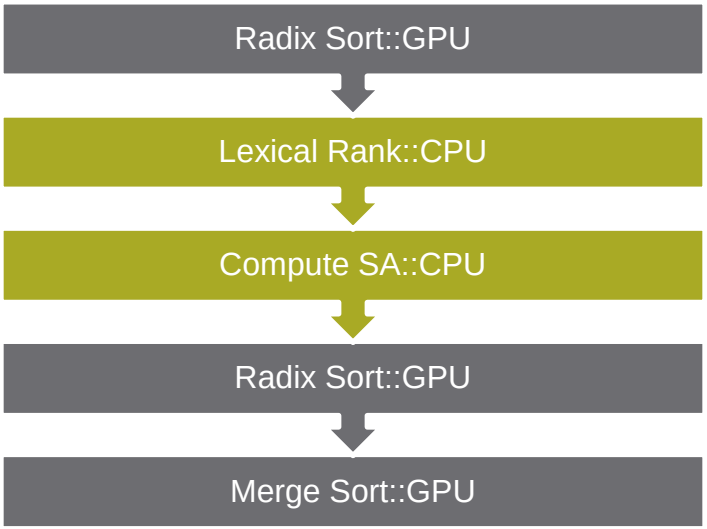
SUFFIX ARRAYS

- 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 T
- Suffix Arrays are used to accelerate in-memory cloud workloads
  - Full text index search
  - Lossless data compression
  - Bio-informatics

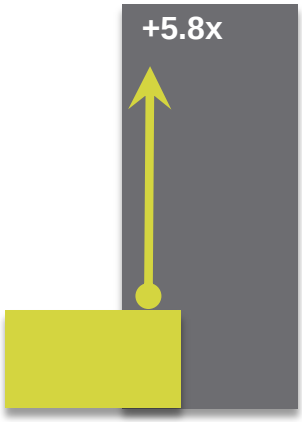
ACCELERATED SUFFIX ARRAY CONSTRUCTION ON HSA

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

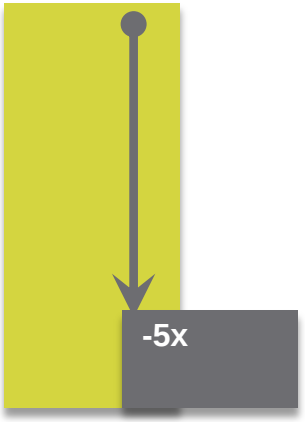
Skew Algorithm for Compute SA



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



INCREASED PERFORMANCE



DECREASED ENERGY

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

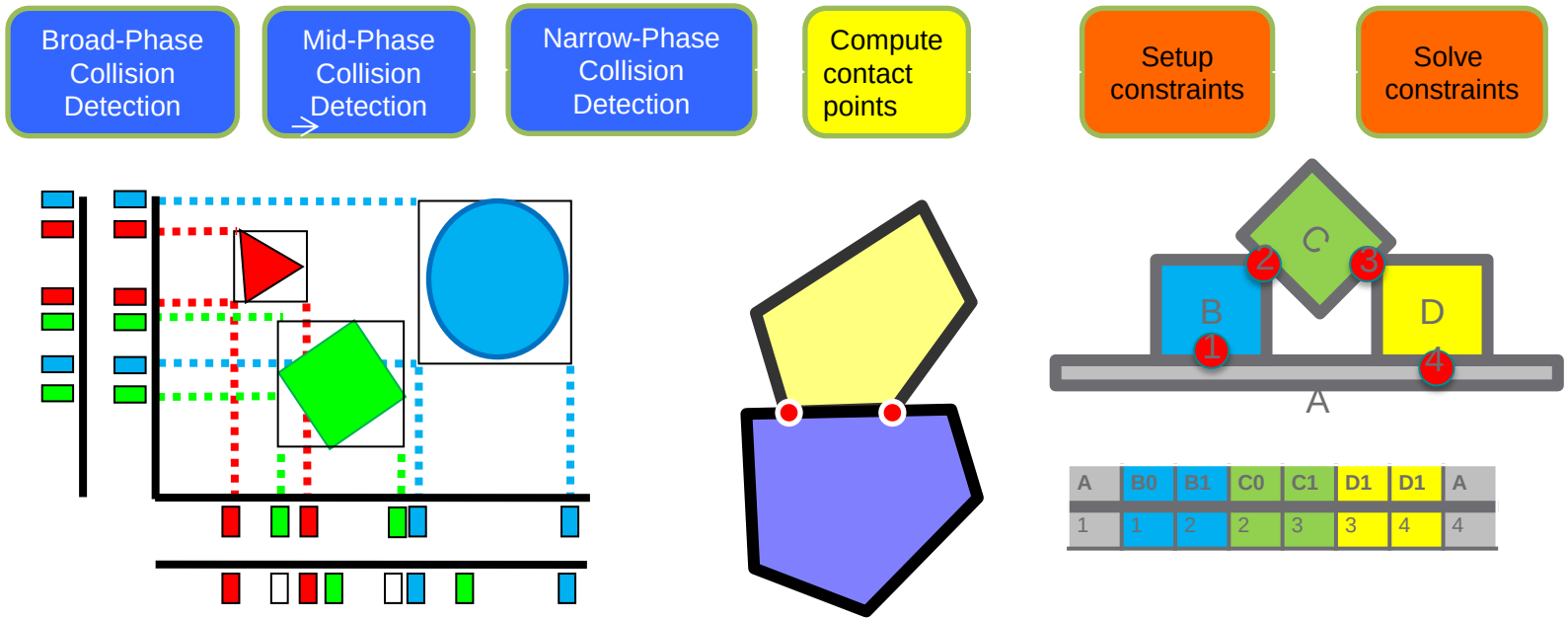
GAMEPLAY RIGID BODY PHYSICS

RIGID BODY PHYSICS SIMULATION

- Rigid-Body Physics Simulation is:
  - A way to animate and interact with objects, widely used in games and movie production
  - Used to drive game play and for visual effects (eye candy)
- Physics Simulation is used in many of today's software:
  - Middleware Physics engines such as Bullet, Havok, PhysX
  - Games ranging from Angry Birds and Cut the Rope to Tomb Raider and Crysis 3
  - 3D authoring tools such as Autodesk Maya, Unity 3D, Houdini, Cinema 4D, Lightwave
  - Industrial applications such as Siemens NX8 Mechatronics Concept Design
  - Medical applications such as surgery trainers
  - Robotics simulation
- ***But GPU-accelerated rigid-body physics is not used in game play — only in effects***

RIGID BODY PHYSICS — ALGORITHM

- Find potential interacting object “pairs” using bounding shape approximations.
- Perform full overlap testing between potentially interacting pairs
- Compute exact contact information for a various shape types
- Compute constraint forces for natural motion and stable stacking



RIGID BODY PHYSICS — CHALLENGES & SOLUTIONS

Implementation Challenges

- Game engine and Physics engine need to interact synchronously during simulation
- The set of pairs can be huge and changes from frame to frame
  - Thousands to Millions for any given frame
- Narrow-phase algorithms cause thread divergence

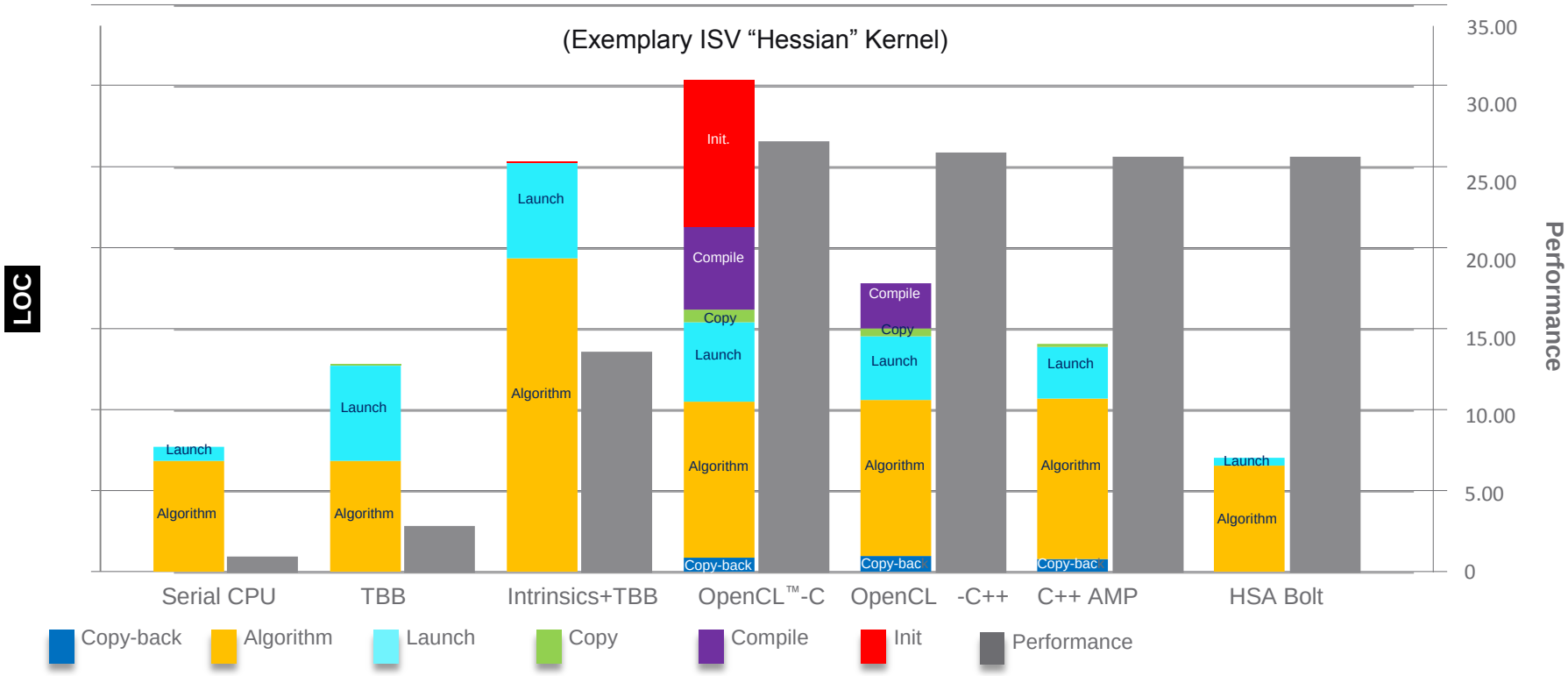
Benefits of HSA

- Fast CPU round-trips
  - User mode dispatch
  - Unified Addressing,
  - Pageable memory,
  - Coherency
- Supports as large a pair list as CPU
  - Entire memory space
  - Dynamic memory allocation
- Improved handling of divergence
  - GPU enqueue

EASE OF PROGRAMMING

CODE COMPLEXITY VS. PERFORMANCE

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

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, in the cloud

HSA™ FOUNDATION

THANK YOU