Implement Runtime Environments for HSA using LLVM

Jim Huang（黃敬群）<jserv.tw@gmail.com>

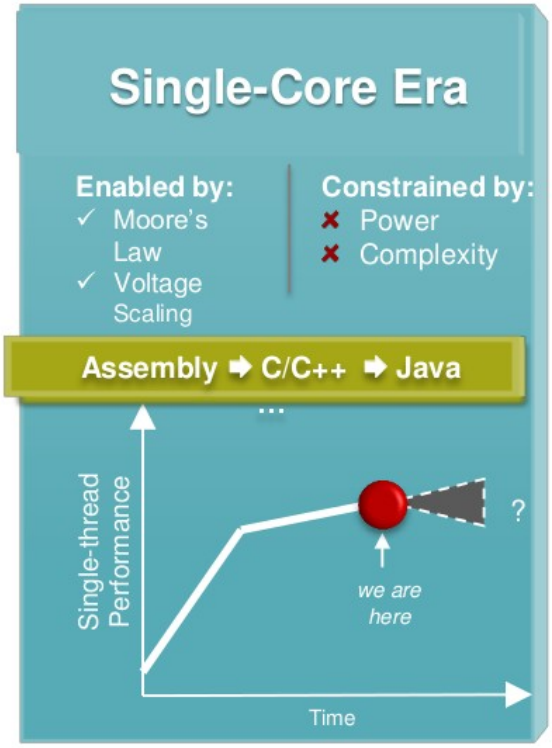
Oct 17, 2013 / ITRI, Taiwan

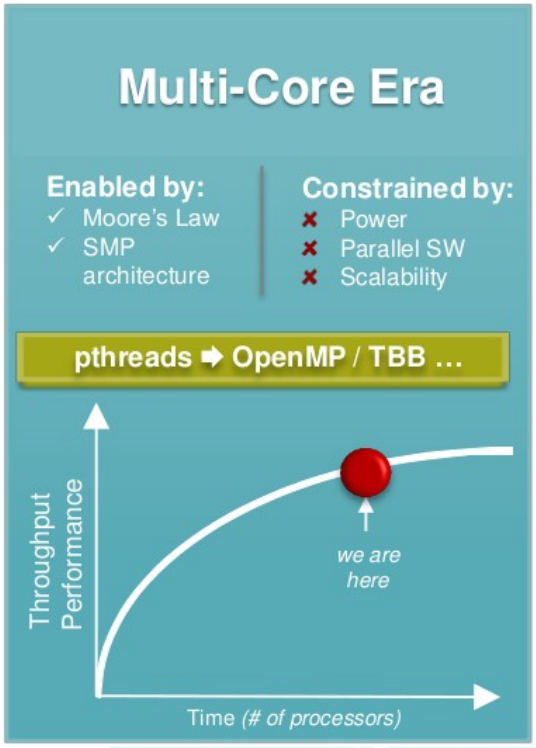
About this presentation

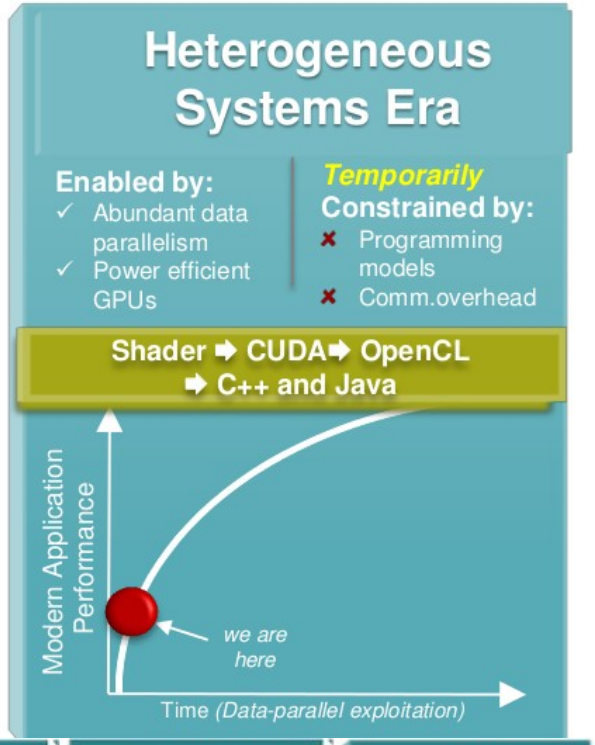
- Transition to heterogeneous
  - mobile phone to data centre
- LLVM and HSA
- HSA driven computing environment

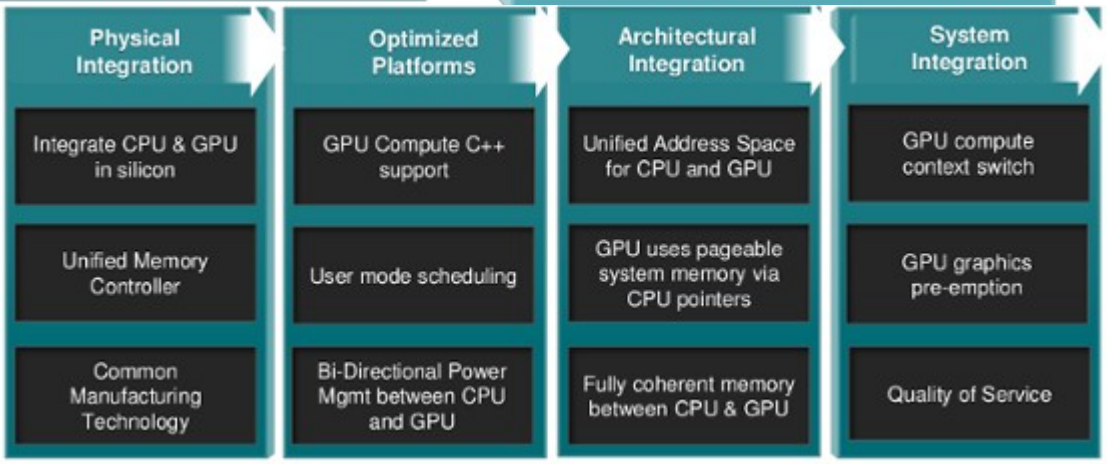
Transition to Heterogeneous Environments

A NEW ERA OF PROCESSOR PERFORMANCE









Herb Sutter's new outlook

http://herbsutter.com/welcome-to-the-jungle/

"In the twilight of Moore's Law, the transitions to multicore processors, GPU computing, and HaaS cloud computing are not separate trends, but aspects of a single trend – mainstream computers from desktops to 'smartphones' are being permanently transformed into heterogeneous supercomputer clusters. Henceforth, a single compute-intensive application will **need to harness different kinds of cores, in immense numbers**, to get its job done."

"The free lunch is over.
Now welcome to the ***hardware jungle***."

Four causes of heterogeneity

- Multiple types of programmable core
  - CPU (lightweight, heavyweight)
  - GPU
  - Accelerators and application-specific
- Interconnect asymmetry
- Memory hierarchies
- Service oriented software demanding

High Performance vs. Embedded

	Embedded	*HPC*
Type of processors	**Heterogeneous**	Homogeneous
Size	Small	**Massive**
Memory	Shared	**Distributed**

but getting closer day by day…

Heterogeneity is mainstream

Quad-core ARM Cortex A9 CPU
Quad-core SGX543MP4+ Imagination GPU

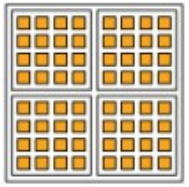
Dual-core ARM 1.4GHz, ARMv7s CPU
Triple-core SGX554MP4 Imagination GPU

Most tablets and smartphones are already powered by heterogeneous processors.

Complementary Processor Architectures

The CPU 

- Serial workloads and task parallel workloads
- < 10 threads
- 1-4 cores
- Short pipeline, <20 stages
- Low latency
- General purpose
- SIMD engine

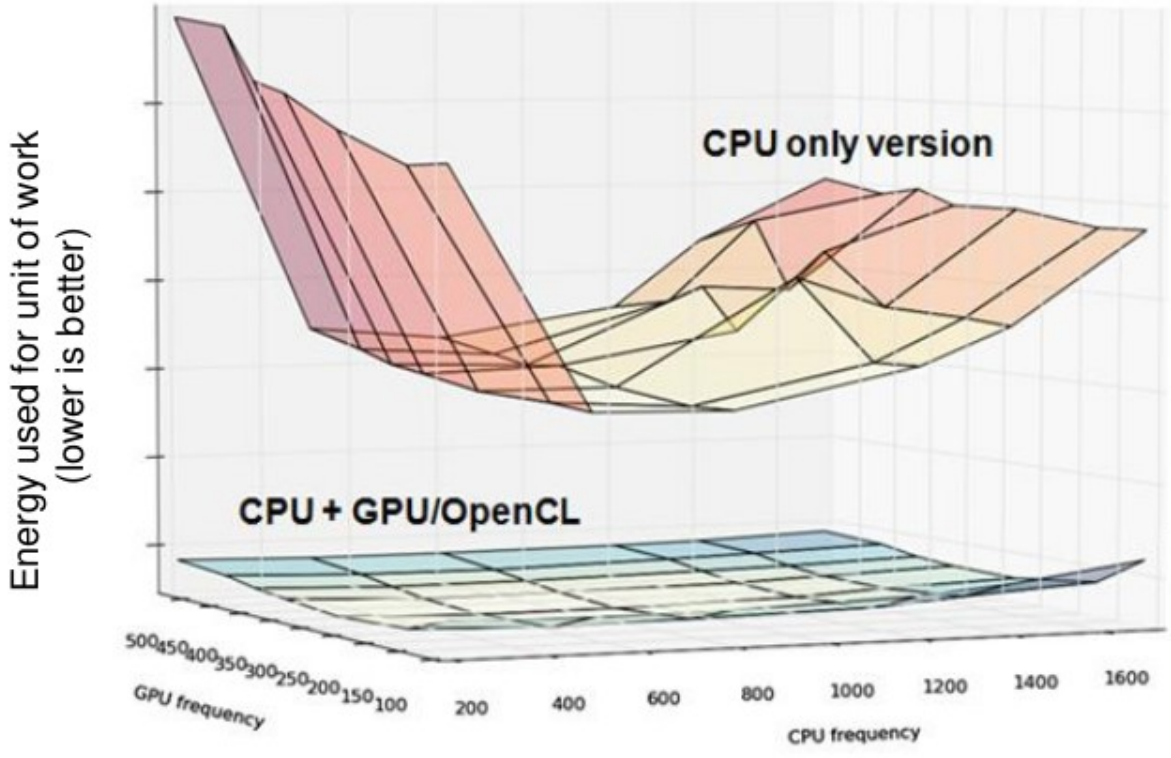
The GPU 

- Data parallel workloads
- 100s-1000s threads
- 1-100s cores
- Long pipeline, >50 stages
- Very high latency
- High throughput
- 2D/3D Graphics
- Stream processing

Source: Heterogeneous Computing in ARM

Computer Vision Based Applications

- Computer Vision entails the acquisition, processing, analysis and understanding of sensor data (images), in order to derive information to enable decisions to be made



In this example:

Consistent 6x speed up

~5x more energy efficiency

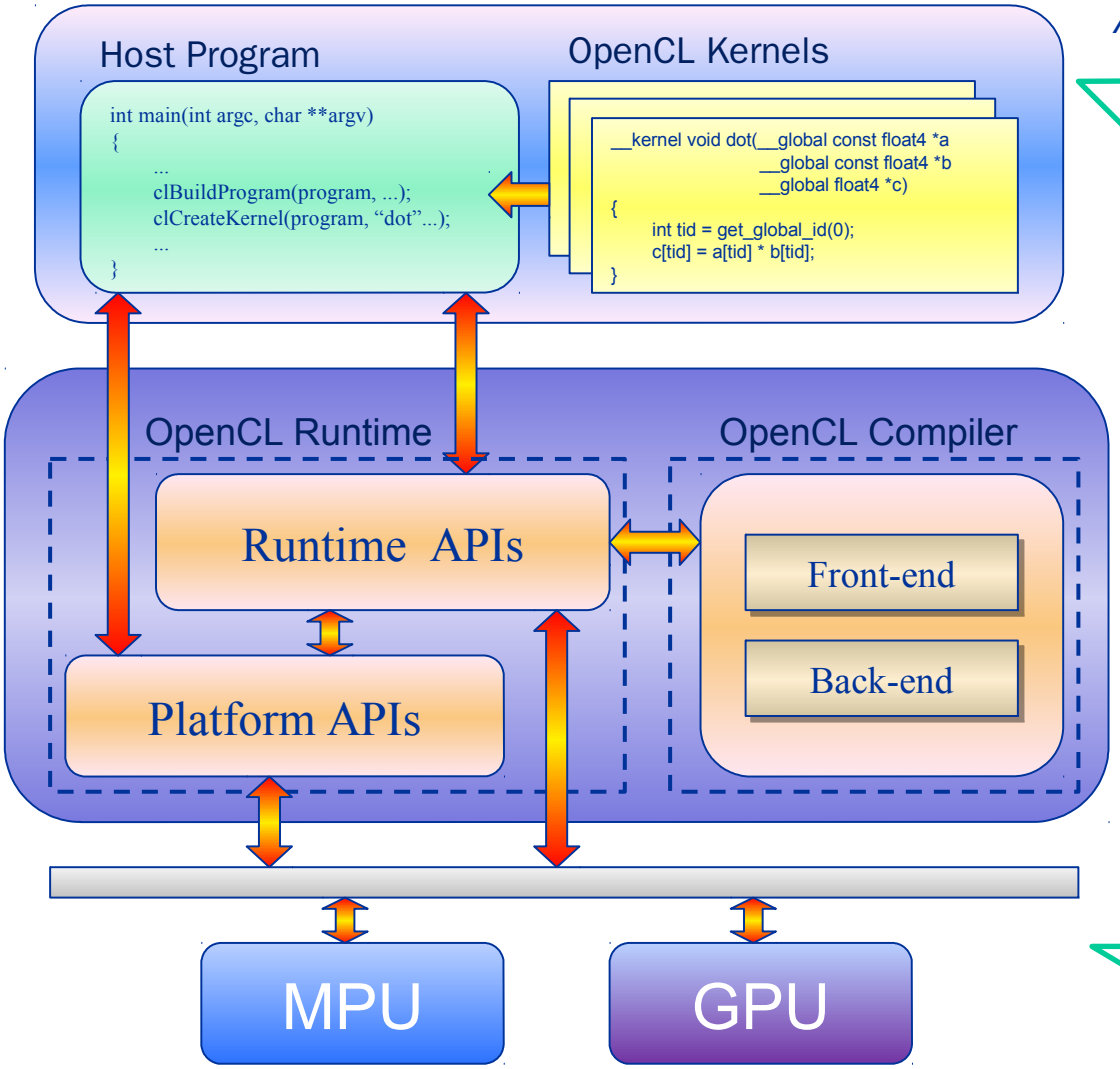
Face detection study on Mali-T604 based silicon

Current limitations

- Disjoint view of memory spaces between CPUs and GPUs
- Hard partition between “host” and “devices” in programming models
- Dynamically varying nested parallelism almost impossible to support
- Large overheads in scheduling heterogeneous, parallel tasks

Background Facilities

OpenCL Framework overview



Application

Separate programs into host-side and kernel-side code fragment

OpenCL Framework

Compiler
- compile OpenCL C language just-in-time

Runtime
- allow host program to manipulate context

Platform

MPU : host, kernel program
GPU : kernel program

Source: OpenCL_for_Halifux.pdf, OpenCL overview, Intel Visual Adrenaline

OpenCL Execution Scenario

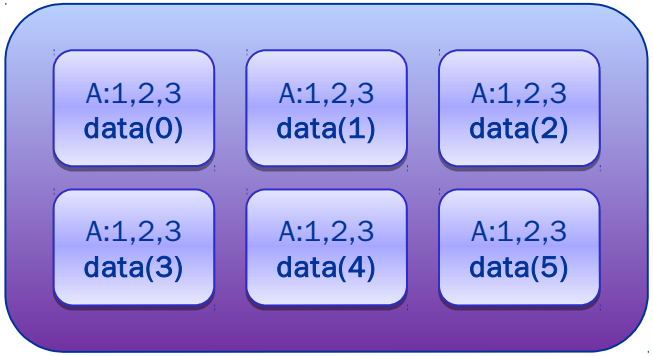
```
kernel
{
   code fragment 1
   code fragment 2
   code fragment 3
}
```

X

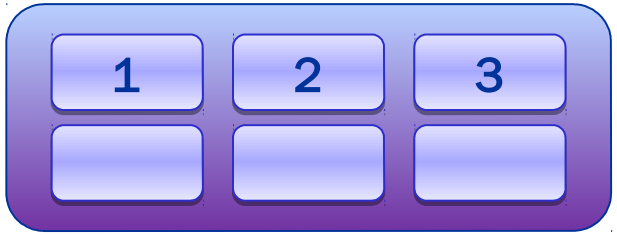
O

```
kernel A
{
   code 1
   code 2
   code 3
}
```

Data-level parallelism



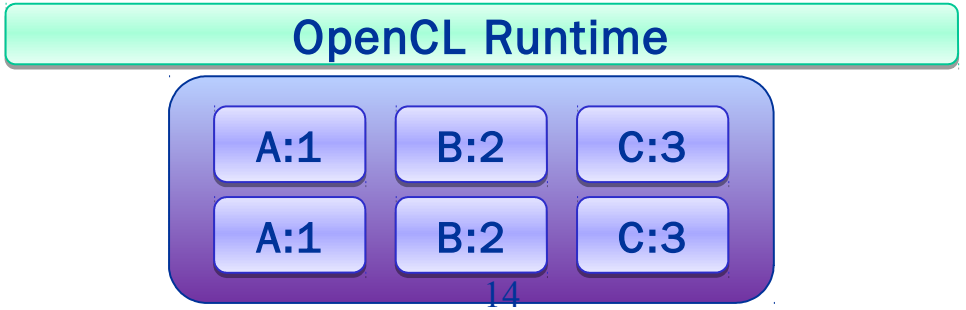
Task-level parallelism




```
kernel A
{
   code 1
}
```

```
kernel B
{
   code 2
}
```

```
kernel C
{
   code 3
}
```

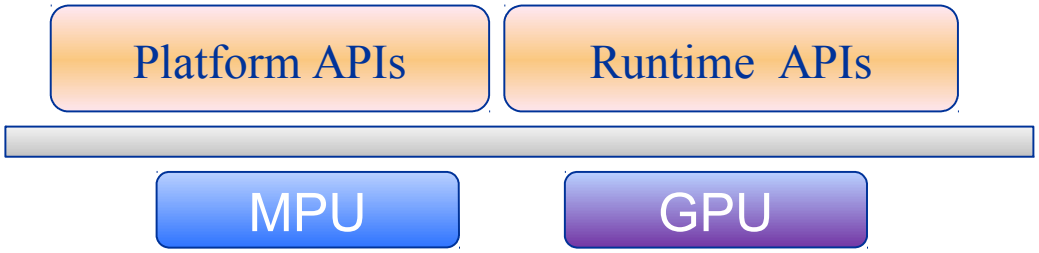


Supporting OpenCL

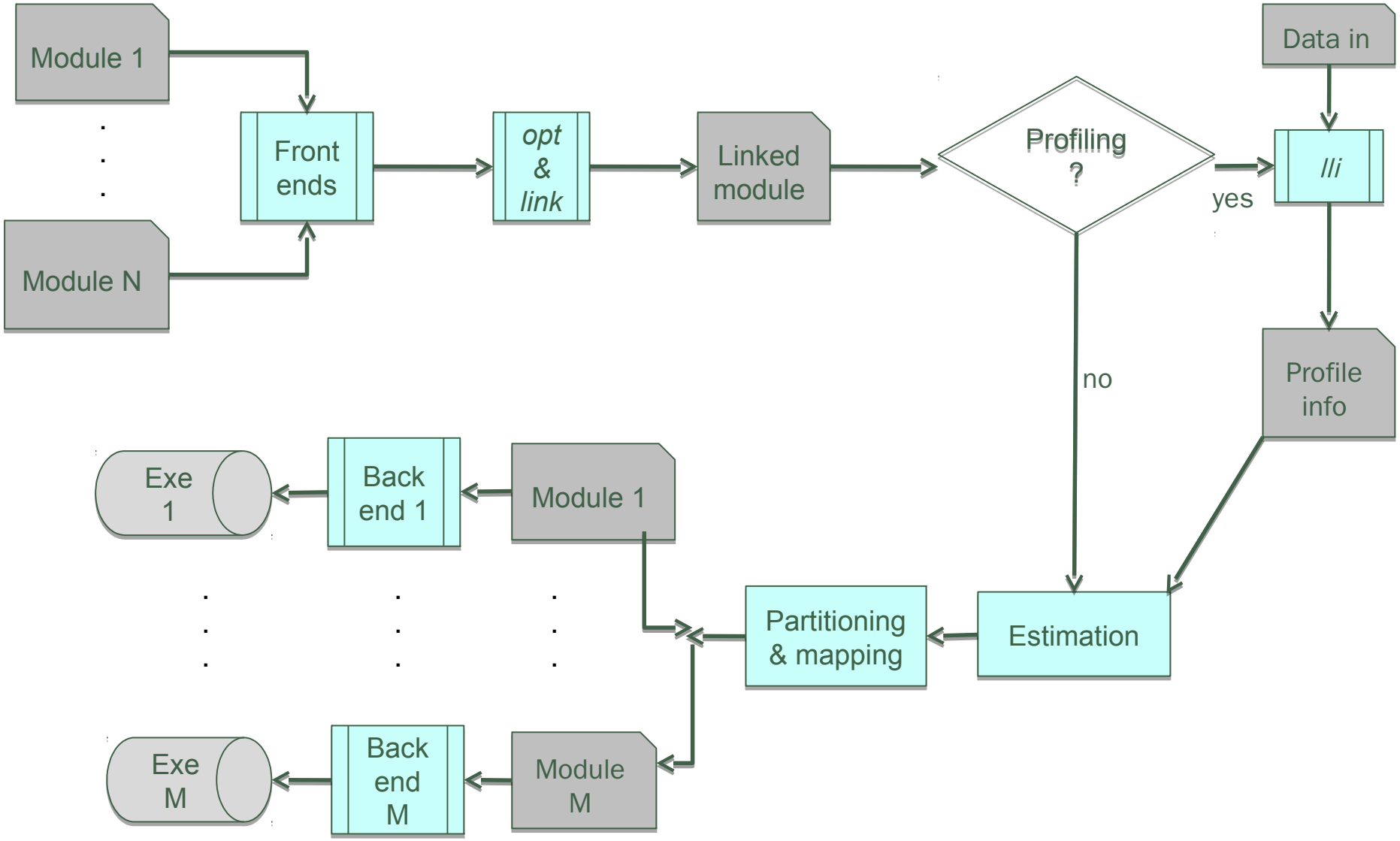
- Syntax parsing by compiler
  - qualifier
  - vector
  - built-in function
  - Optimizations on single core

```
__kernel void add( __global float4 *a,
                   __global float4 *b,
                   __global float4 *c)
{
    int gid = get_global_id(0);
    float4 data = (float4) (1.0, 2.0, 3.0, 4.0);
    c[gid] = a[gid] + b[gid] + data;
}
```

- Runtime implementation
  - handle multi-core issues
  - Co-work with device vendor



LLVM-based compilation toolchain

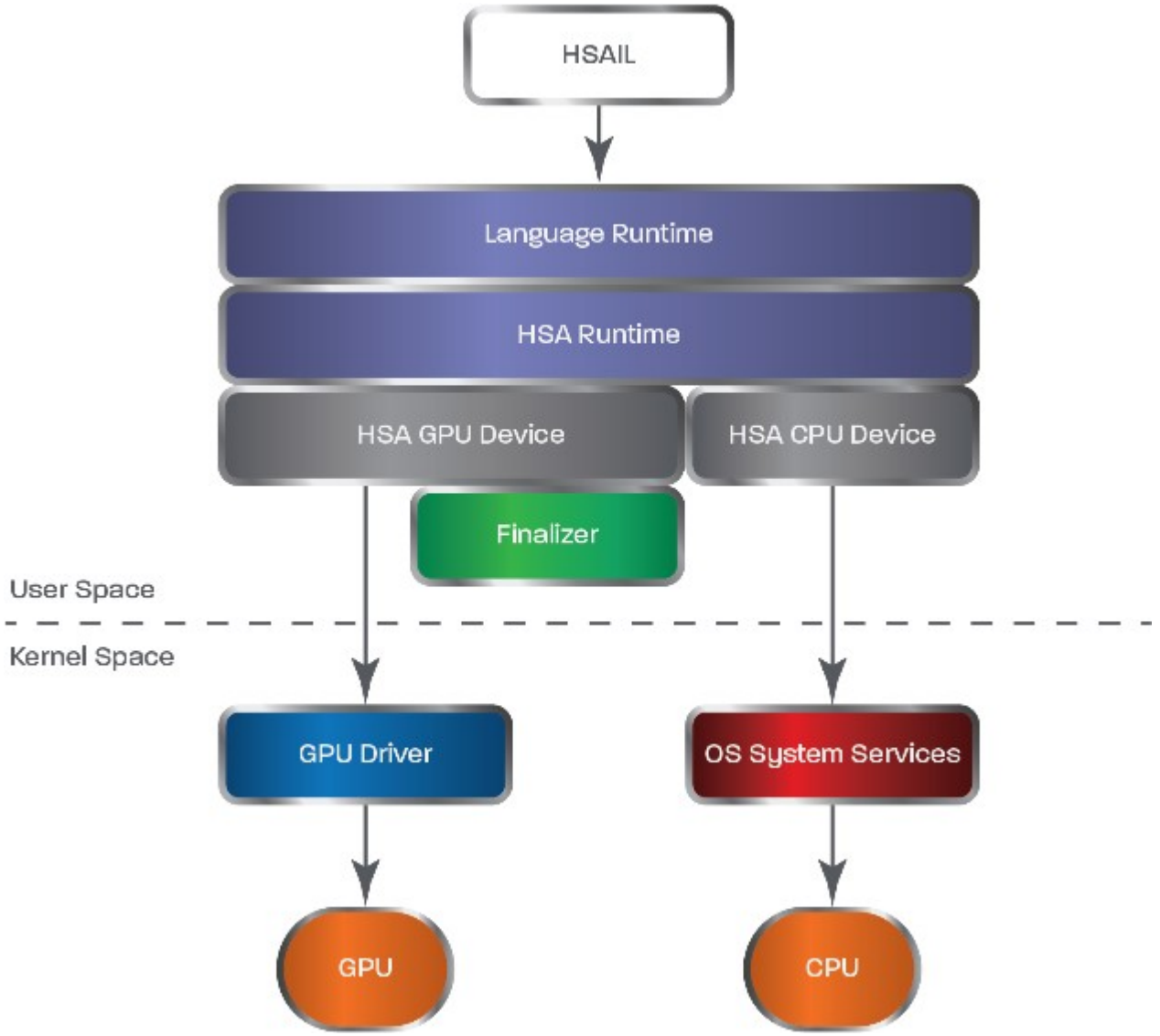


HSA: Heterogeneous System Architecture

HSA overview

- *open* architecture specification
  - HSAIL virtual (parallel) instruction set
  - HSA memory model
  - HSA dispatcher and run-time
- Provides an optimized platform architecture for heterogeneous programming models such as OpenCL, C++AMP, et al

HSA Runtime Stack



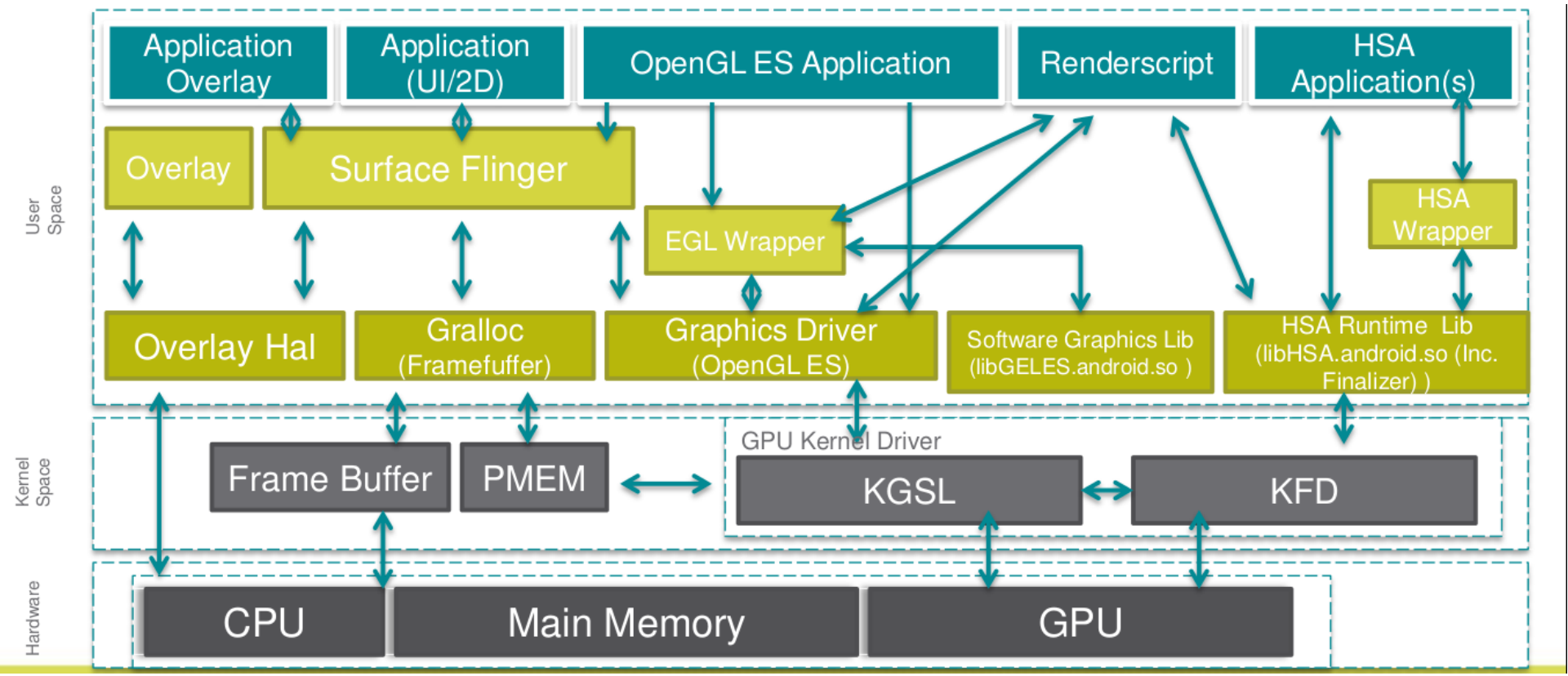
Heterogeneous Programming

- Unified virtual address space for all cores
  - CPU and GPU
  - distributed arrays
- Hardware queues per code with lightweight user mode task dispatch
  - Enables GPU context switching, preemption, efficient heterogeneous scheduling
- First class barrier objects
  - Aids parallel program composability

HSA Intermediate Layer (HSAIL)

- Virtual ISA for parallel programs
- Similar to LLVM IR and OpenCL SPIR
- *Finalised* to specific ISA by a JIT compiler
- Make late decisions on which core should run a task
- Features:
  - Explicitly parallel
  - Support for exceptions, virtual functions and other high-level features
  - syscall methods (I/O, printf etc.)
  - Debugging support

HSA stack for Android (conceptual)



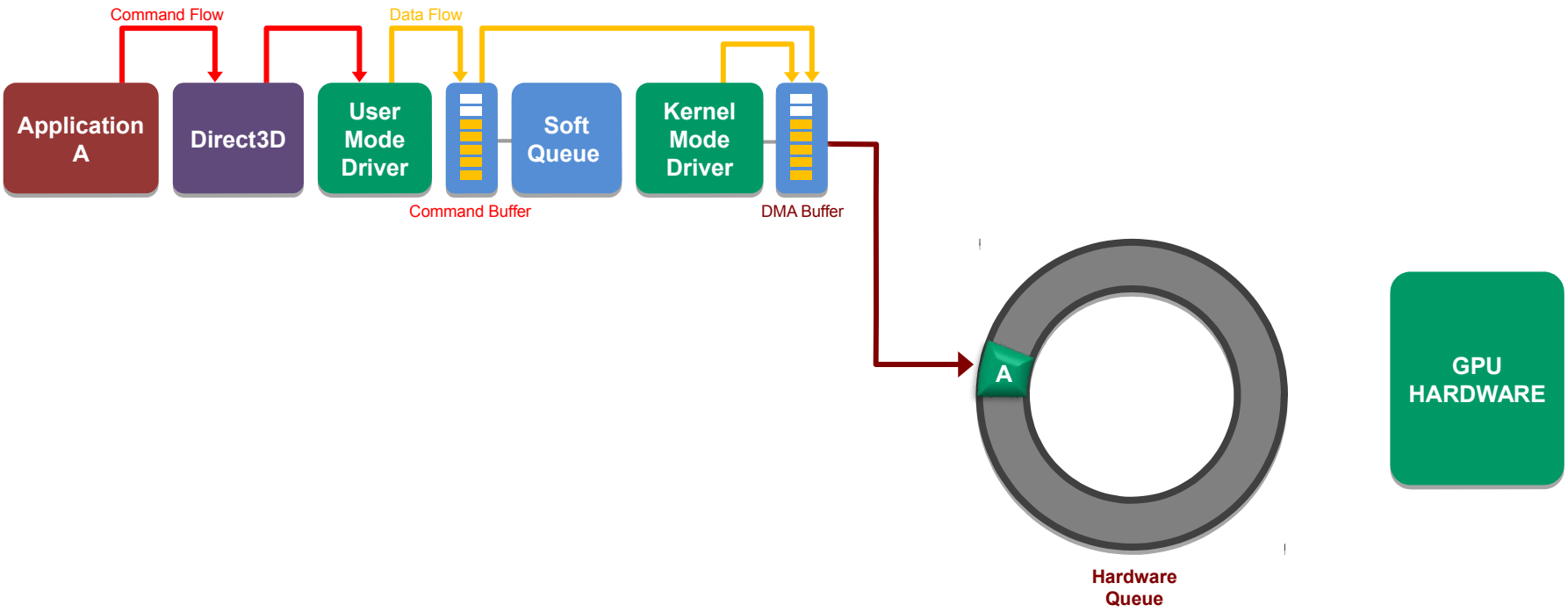
HSA memory model

- Compatible with C++11, OpenCL, Java and .NET memory models
- Relaxed consistency
- Designed to support both managed language (such as Java) and unmanaged languages (such as C)
- Will make it much easier to develop 3rd party compilers for a wide range of heterogeneous products
  - E.g. Fortran, C++, C++AMP, Java et al

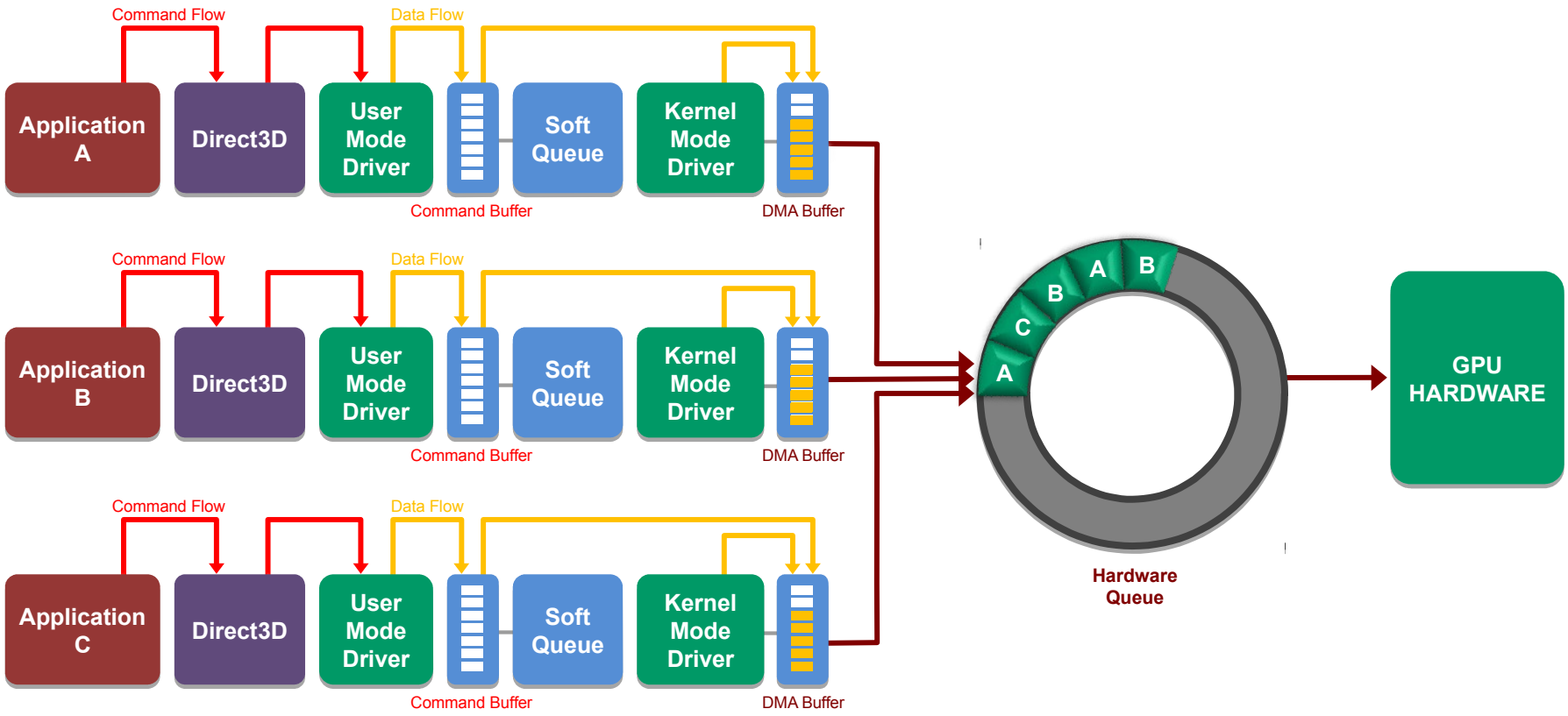
HSA dispatch

- HSA designed to enable heterogeneous task queuing
  - A work queue per core (CPU, GPU, …)
  - Distribution of work into queues
  - Load balancing by work stealing
- Any core can schedule work for any other, including itself
- Significant reduction in overhead of scheduling work for a core

Today's Command and Dispatch Flow

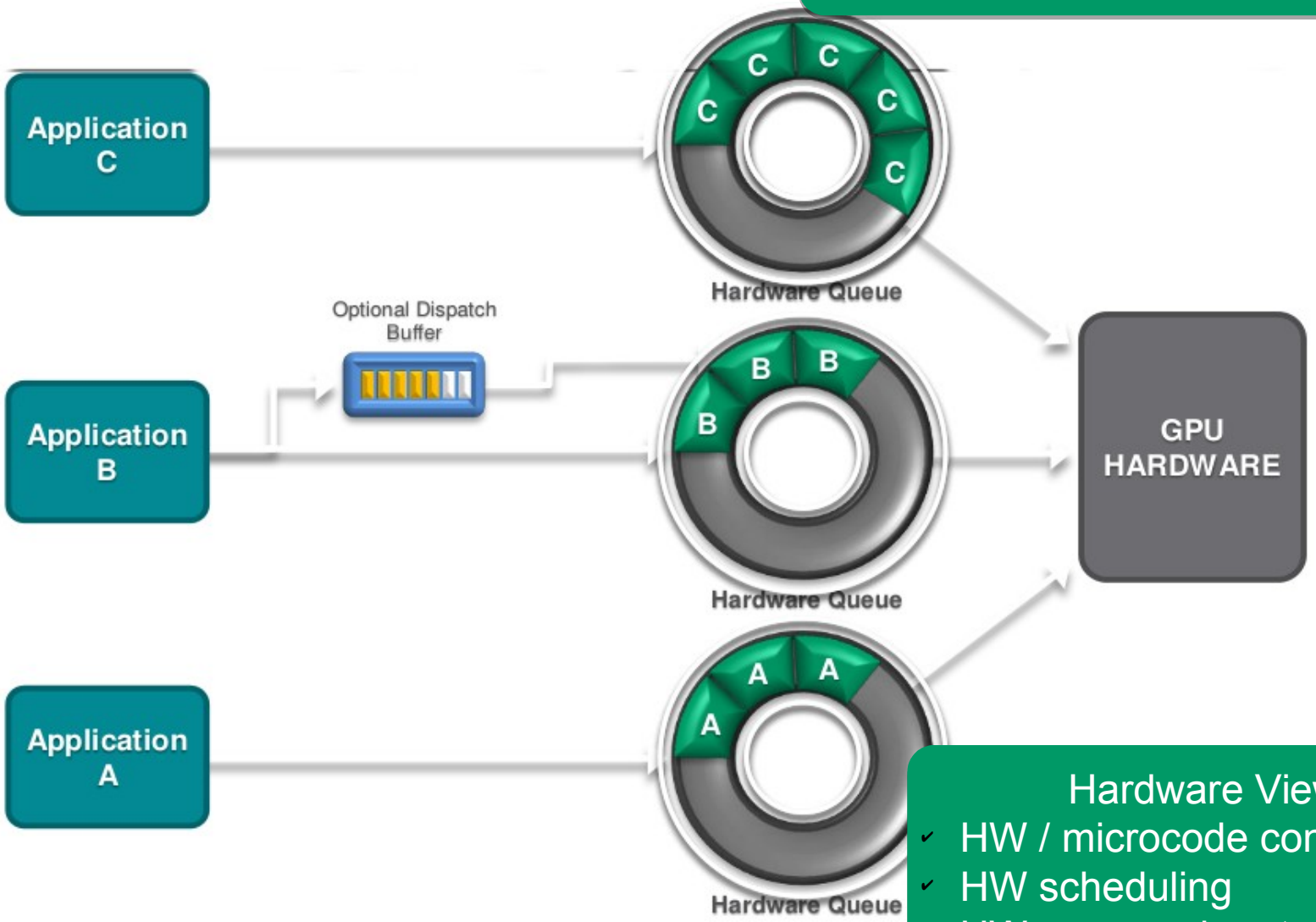


Today's Command and Dispatch Flow



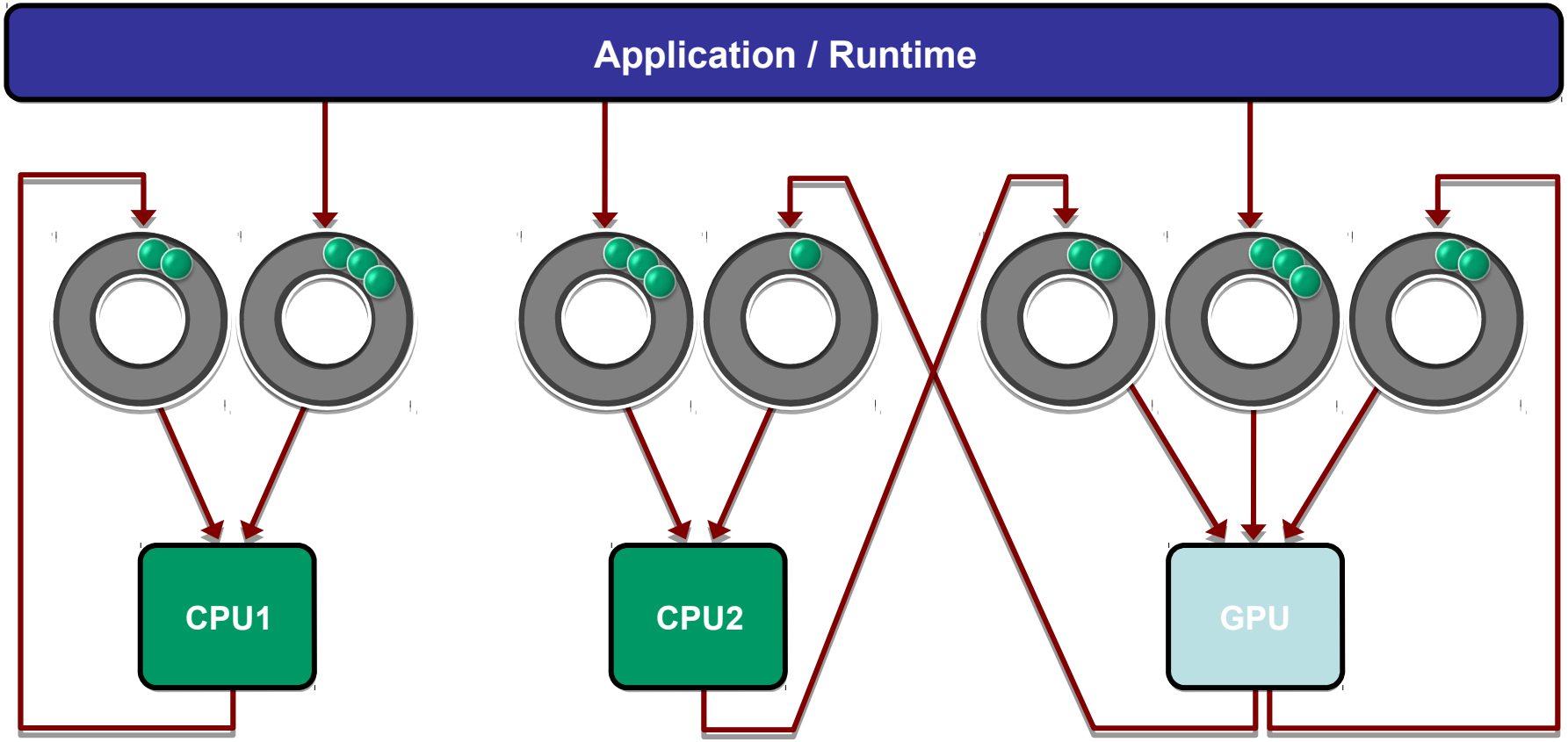
HSA Dispatch Flow

Software View
- User-mode dispatch to hardware
- No kernel mode driver overhead
- Low dispatch latency



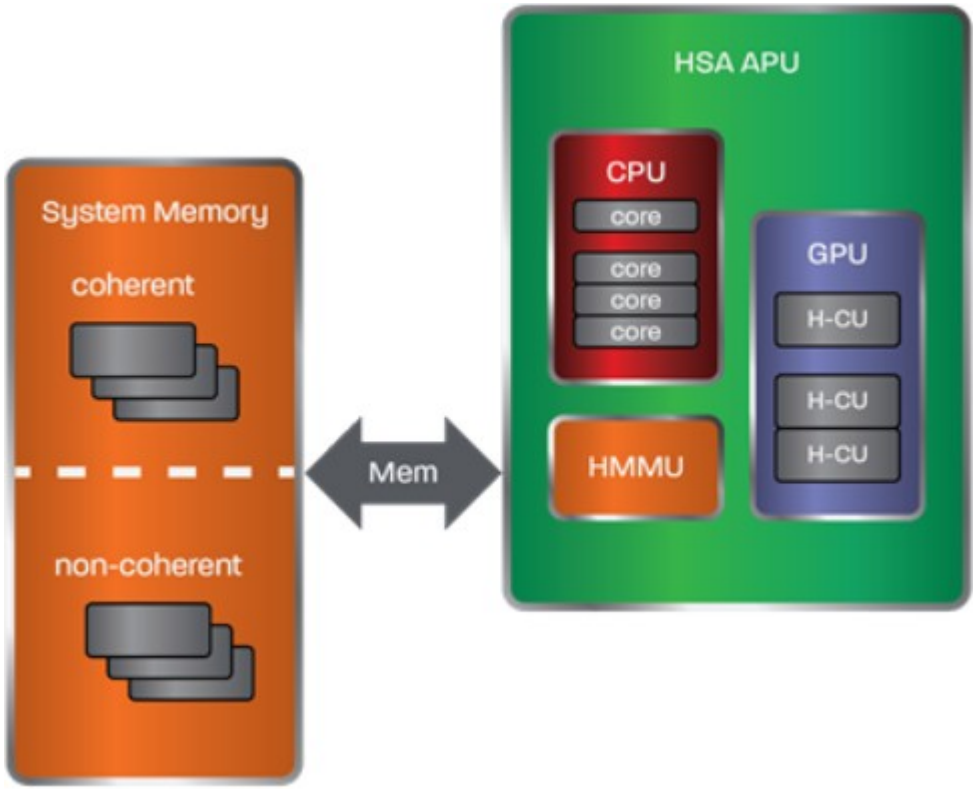
Hardware View
- HW / microcode controlled
- HW scheduling
- HW-managed protection

HSA enabled dispatch

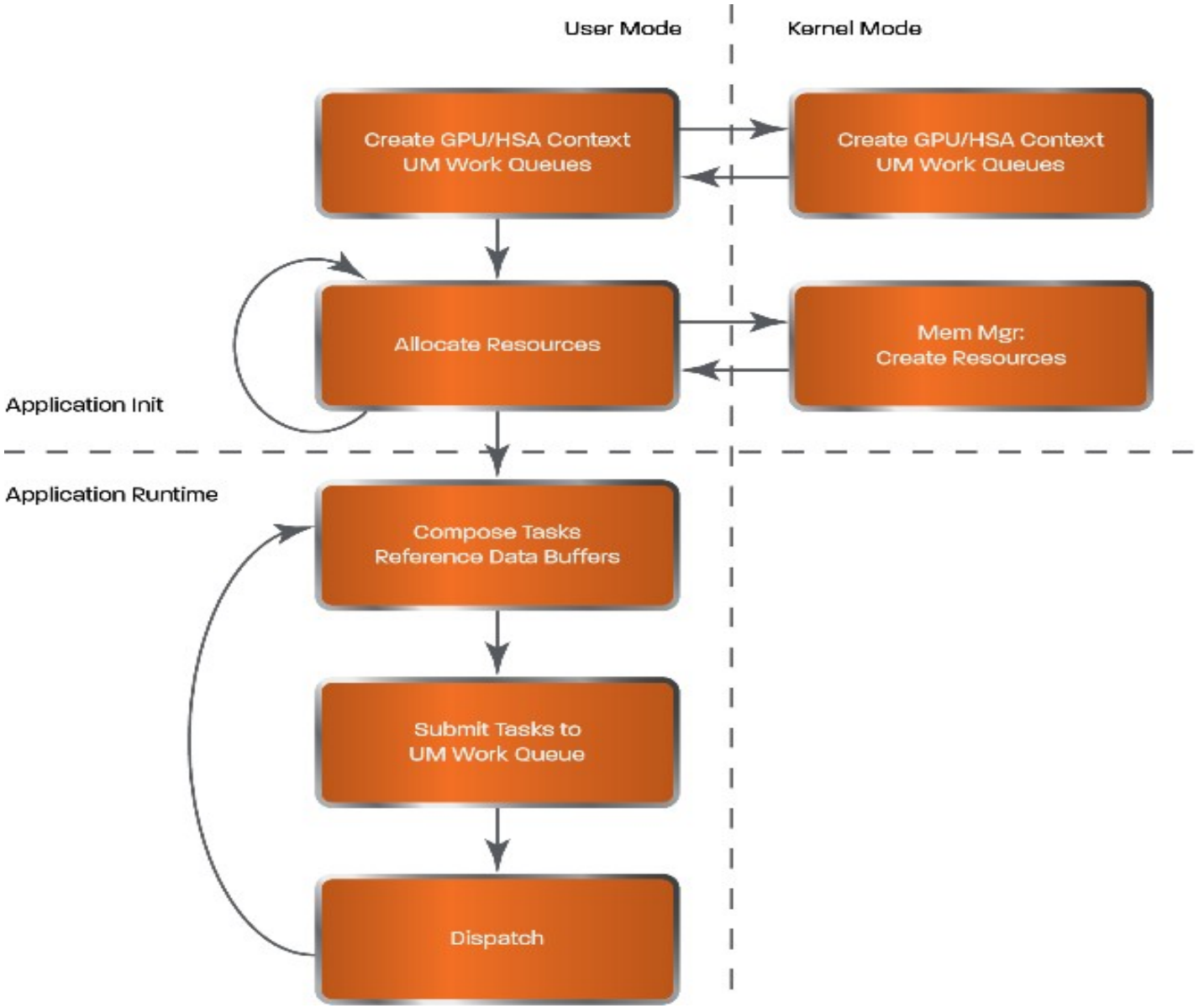


HSA Memory Model

- compatible with C++11/Java/.NET memory models
- Relaxed consistency memory model
- Loads and stores can be re-ordered by the finalizer



Data Flow of HSA



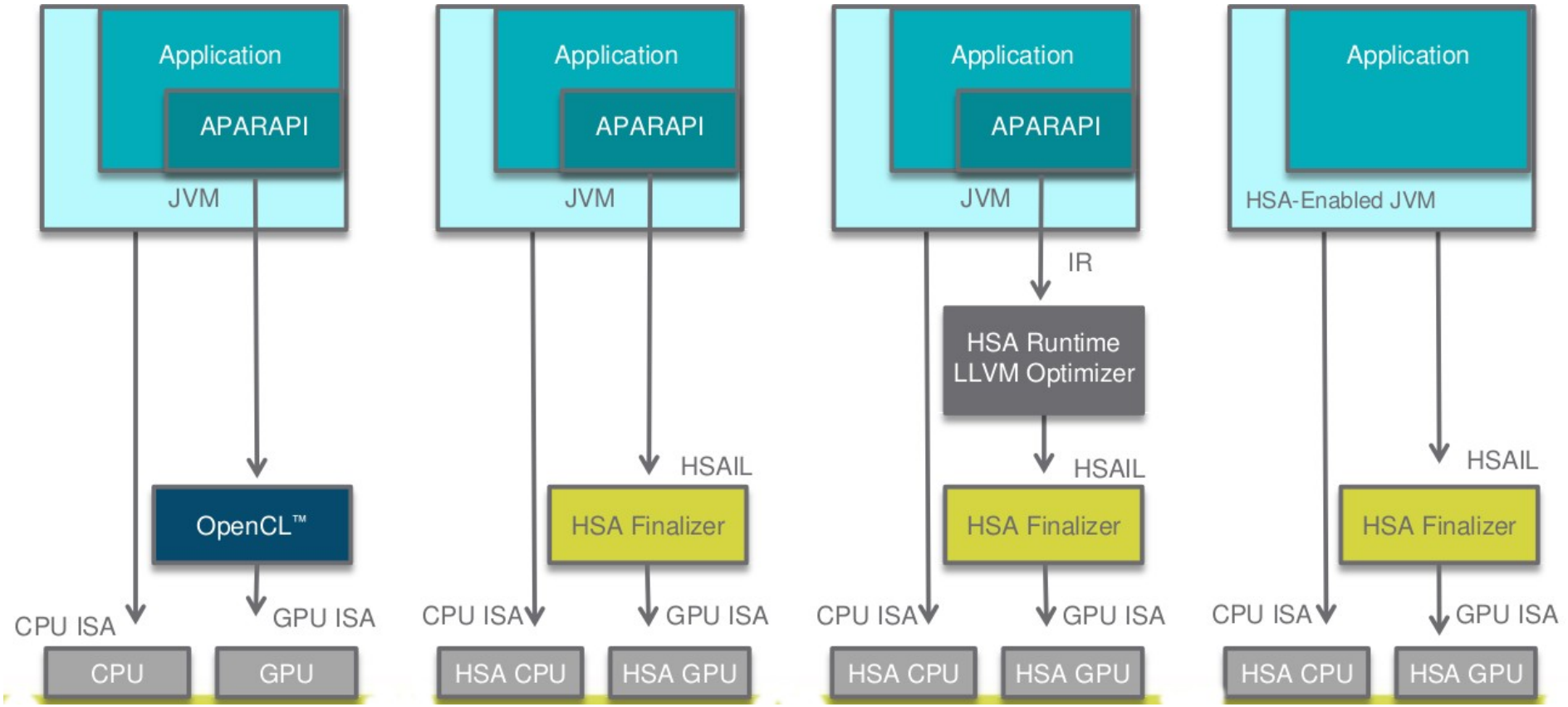
HSAIL

- intermediate language for parallel compute in HSA
  - Generated by a high level compiler (LLVM, gcc, Java VM, etc)
  - Compiled down to GPU ISA or parallel ISAFinalizer
  - Finalizer may execute at run time, install time or build time
- low level instruction set designed for parallel compute in a shared virtual memory environment.
- designed for fast compile time, moving most optimizations to HL compiler
  - Limited register set avoids full register allocation in finalizer

GPU based Languages

- Dynamic Language for exploration of heterogeneous parallel runtimes
- LLVM-based compilation
  - Java, Scala, JavaScript, OpenMP
- Project Sumatra:GPU Acceleration for Java in OpenJDK
- ThorScript

Accelerating Java



Source: HSA, Phil Rogers

Open Source software stack for HSA

A Linux execution and compilation stack is open-sourced by AMD

- Jump start the ecosystem
- Allow a single shared implementation where appropriate
- Enable university research in all areas

Component Name	**Purpose**
HSA Bolt Library	Enable understanding and debug
OpenCL HSAIL Code Generator	Enable research
LLVM Contributions	Industry and academic collaboration
HSA Assembler	Enable understanding and debug
HSA Runtime	Standardize on a single runtime
HSA Finalizer	Enable research and debug
HSA Kernel Driver	For inclusion in Linux distros

Open Source Tools

- Hosted at GitHub: https://github.com/HSAFoundation
- HSAIL-Tools: Assembler/Disassembler
- Instruction Set Simulator
- HSA ISS Loader Library for Java and C++ for creation and dispatch HSAIL kernel

HSAIL example

```
ld_kernarg_u64 $d0, [%_out];
ld_kernarg_u64 $d1, [%_in];

@block0:
workitemabsid_u32 $s2, 0;
cvt_s64_s32 $d2, $s2;
mad_u64 $d3, $d2, 8, $d1;
ld_global_u64 $d3, [$d3];   //pointer
ld_global_f32 $s0, [$d3+0]; // x
ld_global_f32 $s1, [$d3+4]; // y
ld_global_f32 $s2, [$d3+8]; // z
mul_f32 $s0, $s0, $s0;       // x*x
mul_f32 $s1, $s1, $s1;       // y*y
add_f32 $s0, $s0, $s1; // x*x + y*y
mul_f32 $s2, $s2, $s2;      // z*z
add_f32 $s0, $s0, $s2;// x*x + y*y + z*z
sqrt_f32 $s0, $s0;
mad_u64 $d4, $d2, 4, $d0;
st_global_f32 $s0, [$d4];
ret;
```

```
sqrt((float
  (i*i +
   (i+1)*(i+1) +
   (i+2)*(i+2)))
```

Conclusions

- Heterogeneity is an increasingly important trend
- The market is finally starting to create and adopt the necessary open standards
- HSA should enable much more dynamically heterogeneous nested parallel programs and programming models

Reference

- Heterogeneous Computing in ARM (2013)
- Project Sumatra: GPU Acceleration for Java in OpenJDK
  - http://openjdk.java.net/projects/sumatra/
- HETEROGENEOUS SYSTEM ARCHITECTURE, Phil Rogers