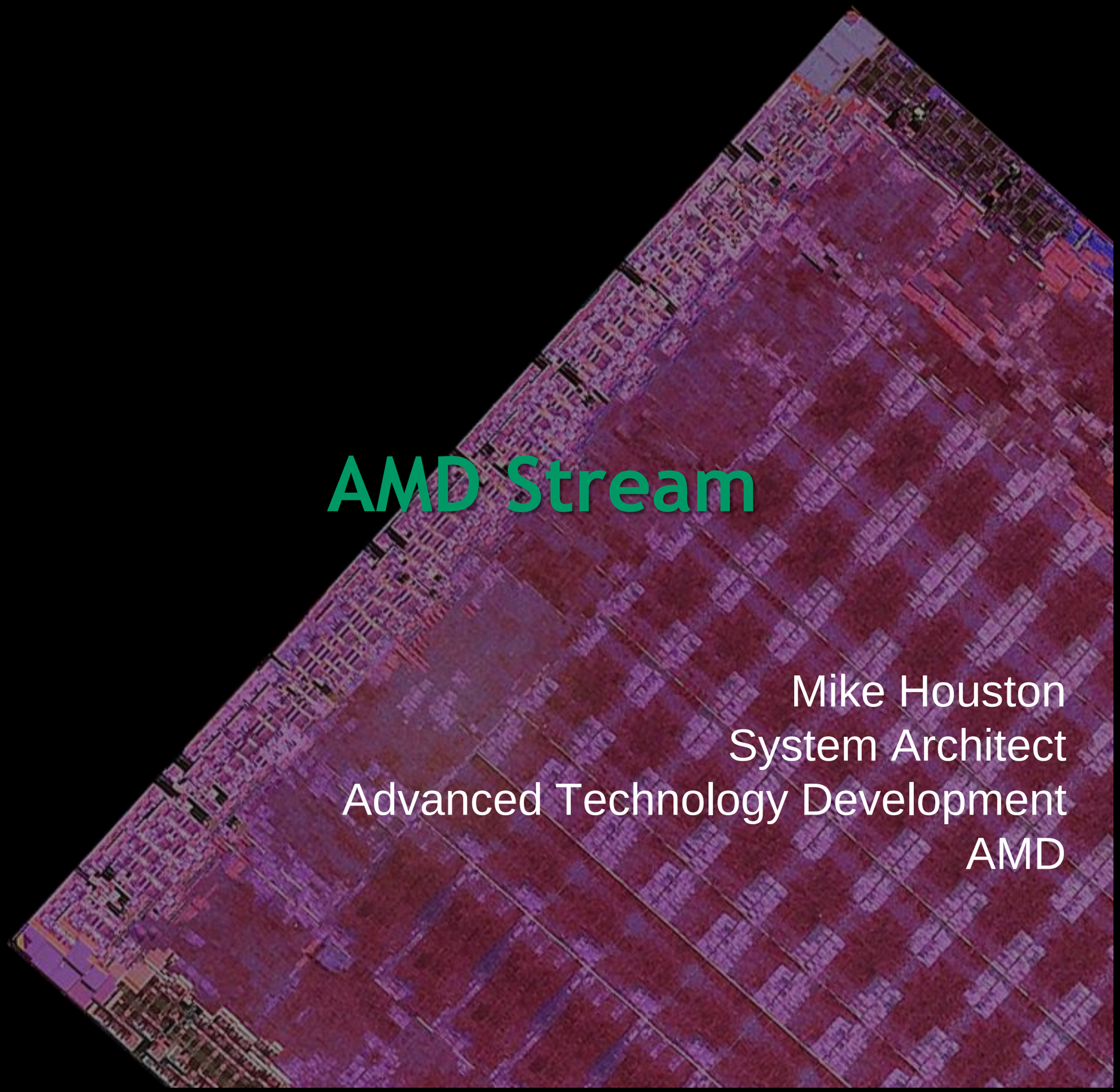


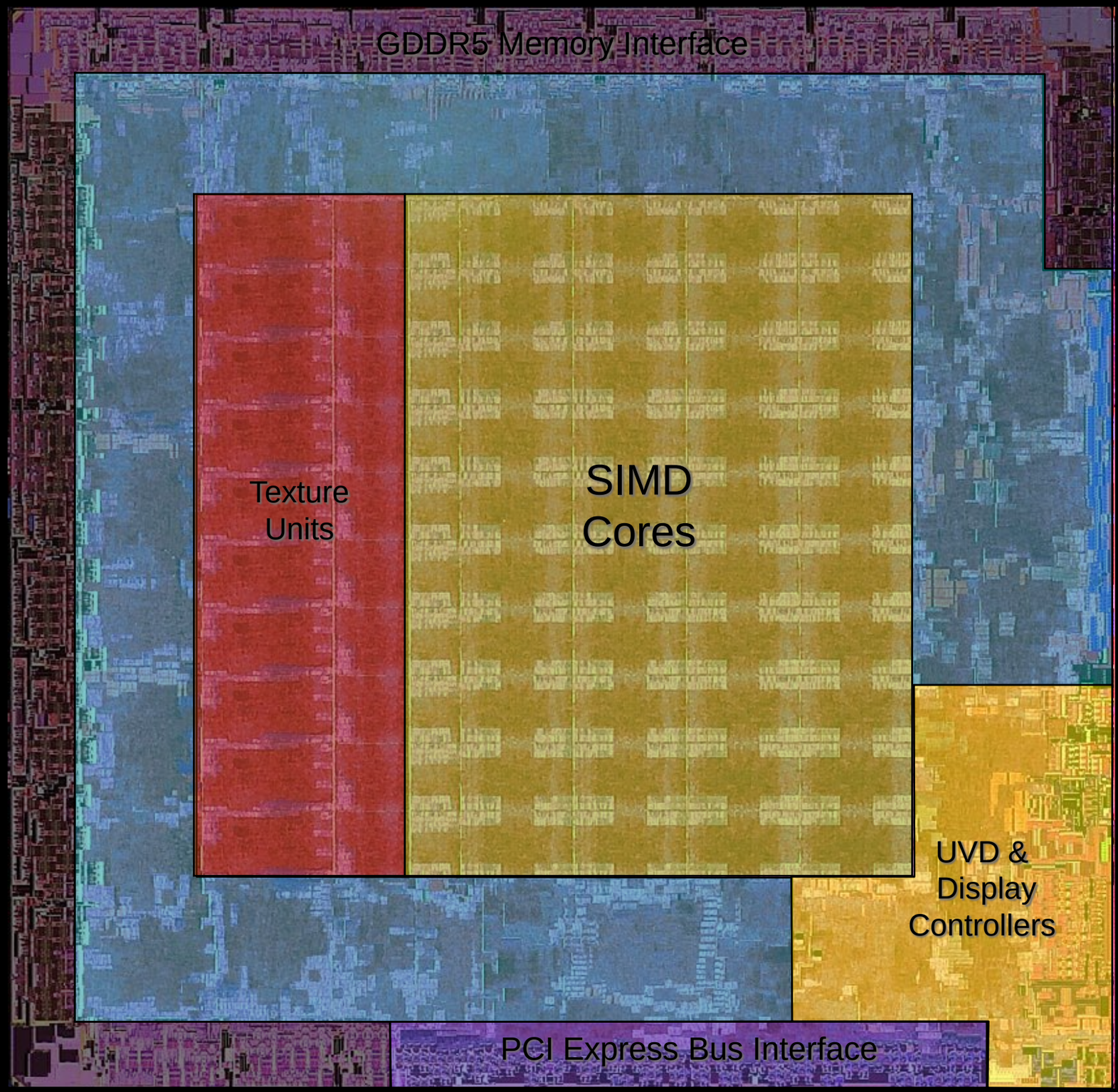
AMD Stream

Mike Houston
System Architect
Advanced Technology Development
AMD

AMD

ATI RADEON PREMIUM GRAPHICS

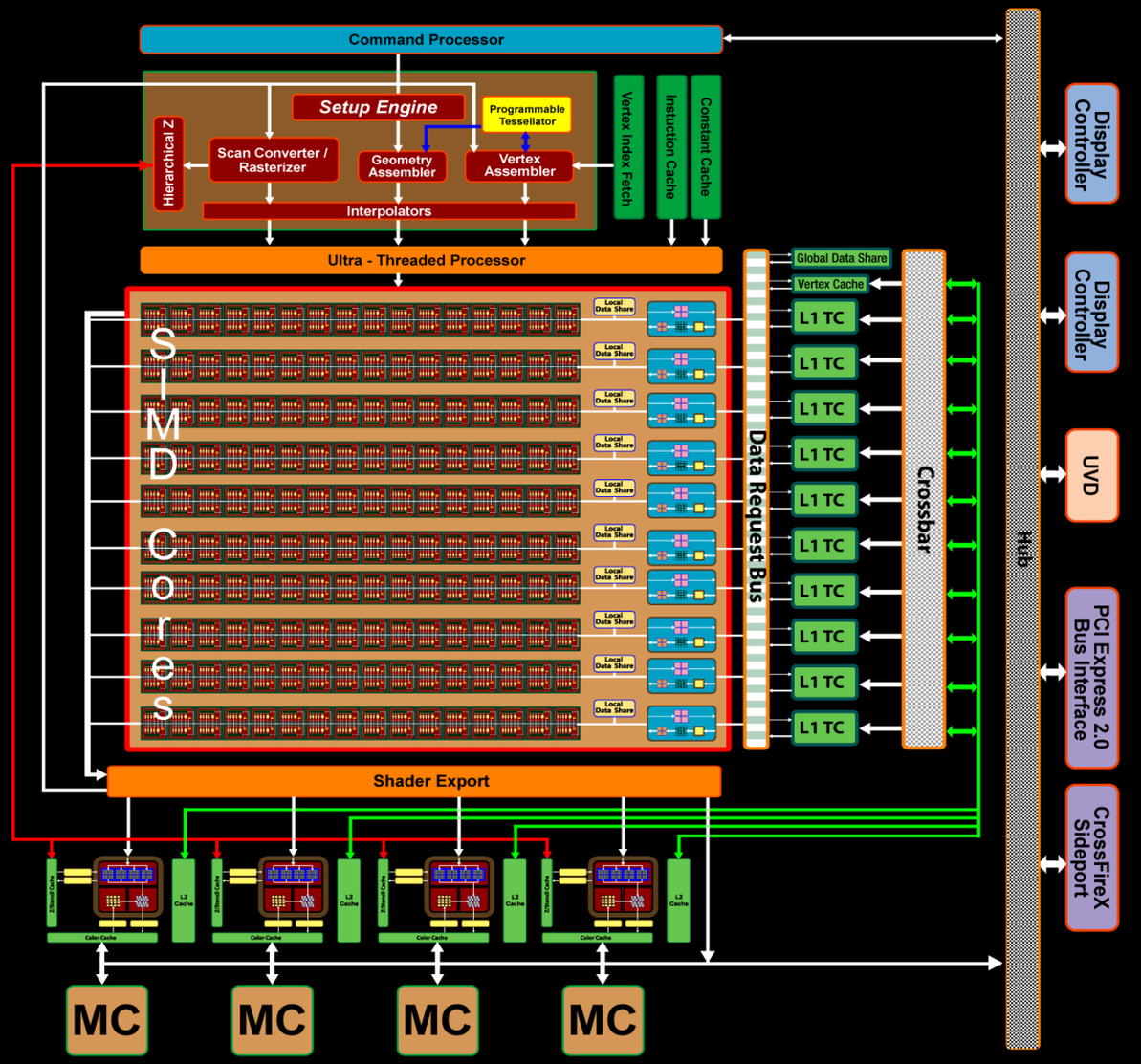
ATI Radeon™ HD 4800 Series Architecture



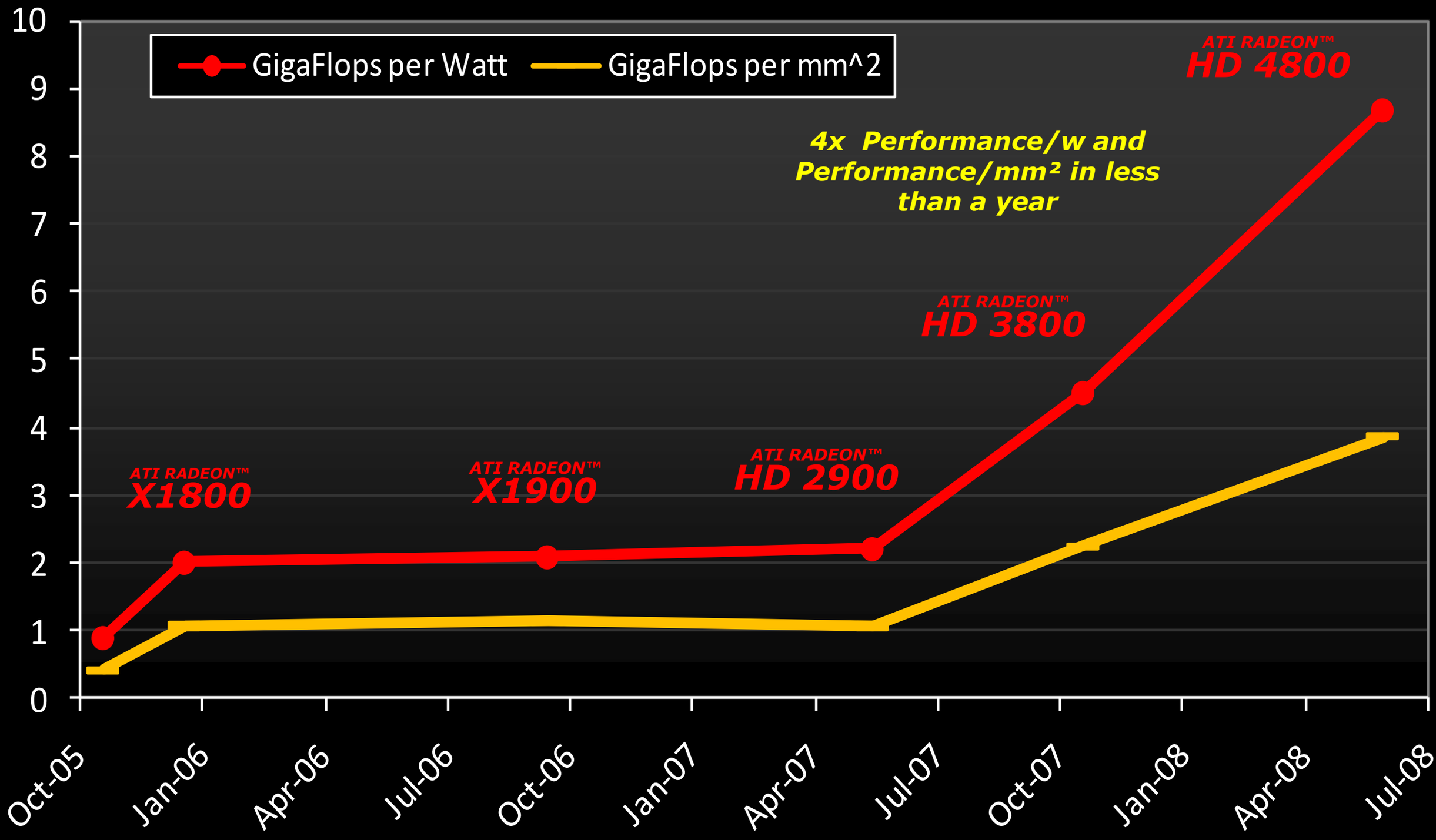
Terascale Graphics Engine

- 800 highly optimized stream processing units
- New SIMD core layout
- Optimized fetch units
- New cache design
- New memory architecture

	ATI Radeon™ HD 3870	ATI Radeon™ HD 4870	Difference
Die Size	190 mm^2	260 mm^2	1.4x
Memory	72 GB/sec	115 GB/sec	1.6x
Fetch units	16	40	2.5x
Shader	320	800	2.5x

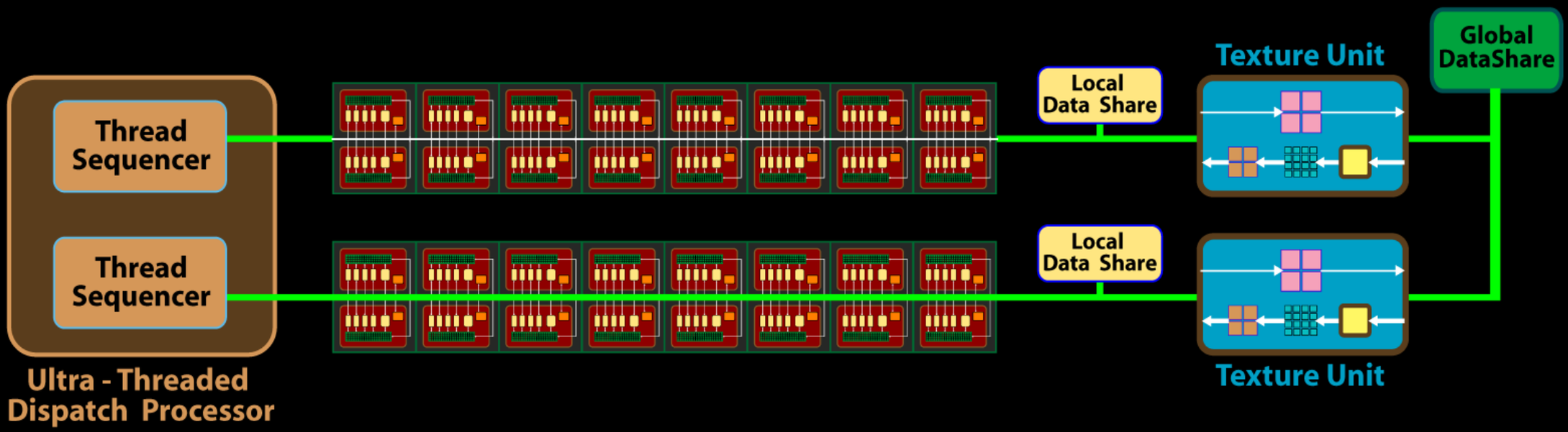


Design Efficiency



SIMD Cores

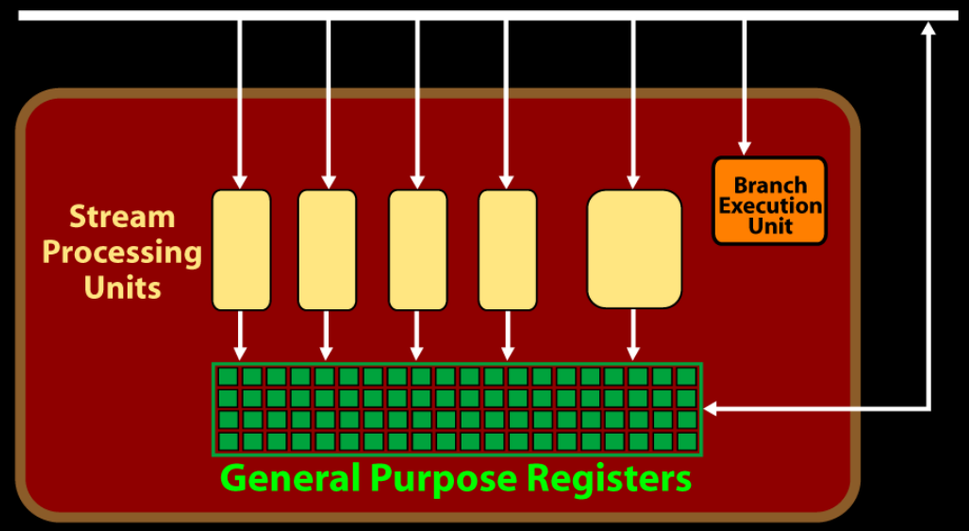
- Each core:
  - Includes 80 scalar stream processing units in total + 16KB Local Data Share
  - Has its own control logic and runs from a shared set of threads
  - Has 4 dedicated texture units + L1 cache
  - Communicates with other SIMD cores via 16KB global data share
- New design allows texture fetch capability to scale with shader power, maintaining 4:1 ALU:TEX ratio

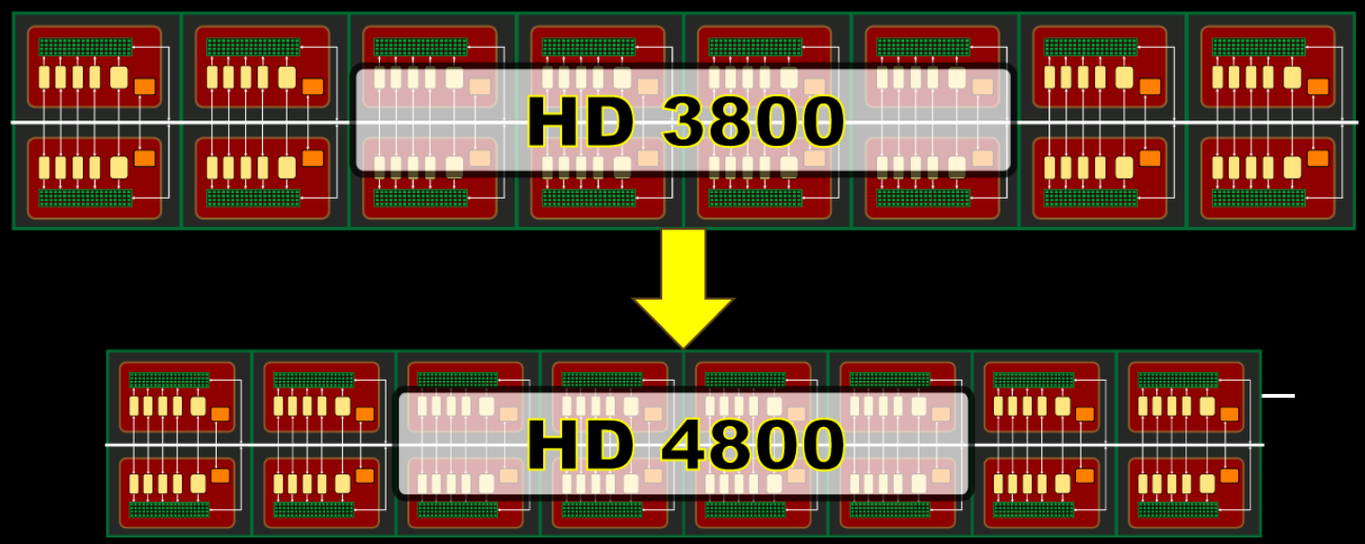


Stream Processing Units

- 40% increase in performance per mm^{2*}
- More aggressive clock gating for improved Performance per Watt *
- Fast double precision processing (240 GigaFLOPS)
- Integer bit shift operations for all units

  (12.5x improvement *)

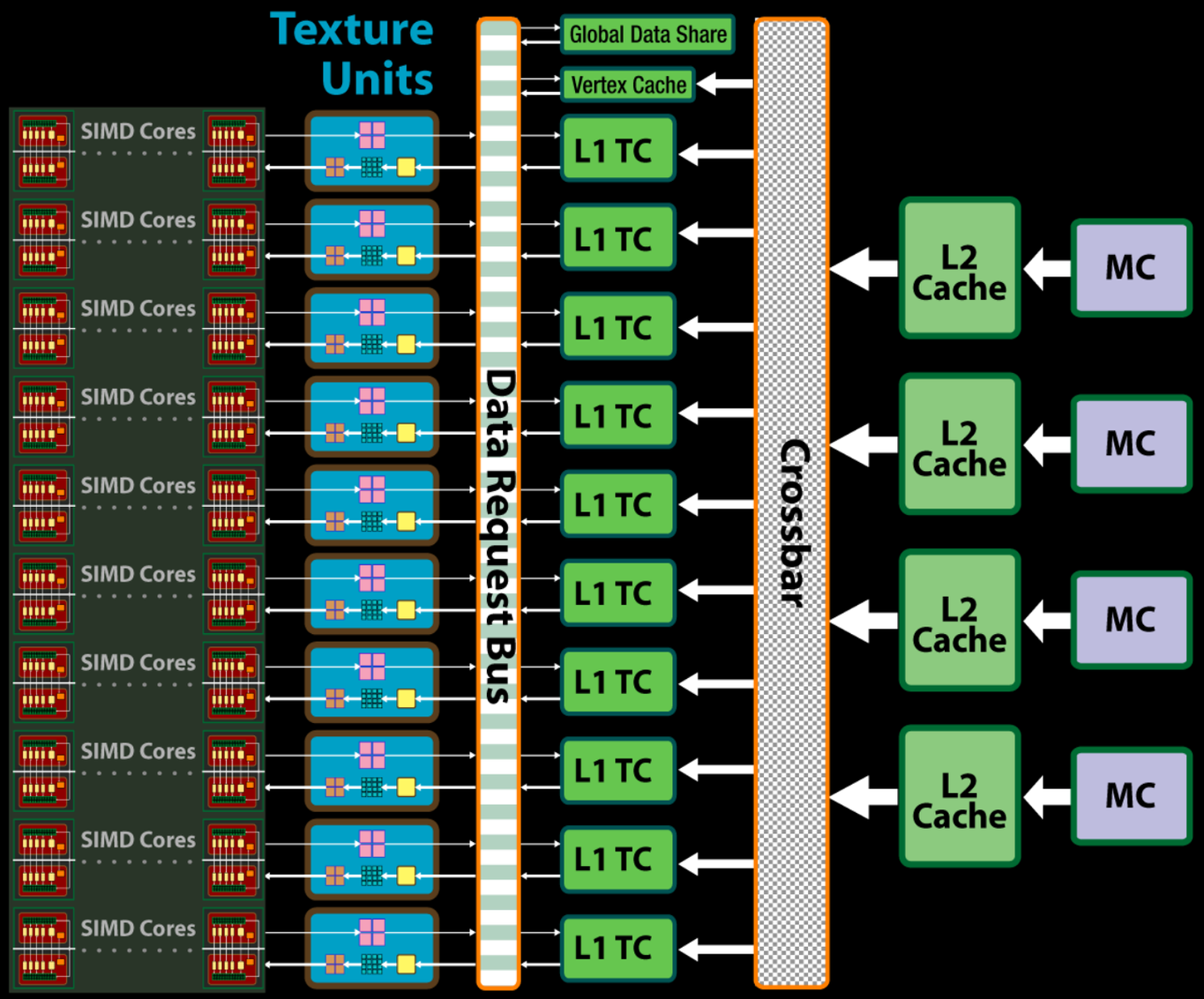




* Internal AMD test results comparing ATI Radeon™ HD 4800 series and ATI Radeon™ HD 3800 series

Texture Units

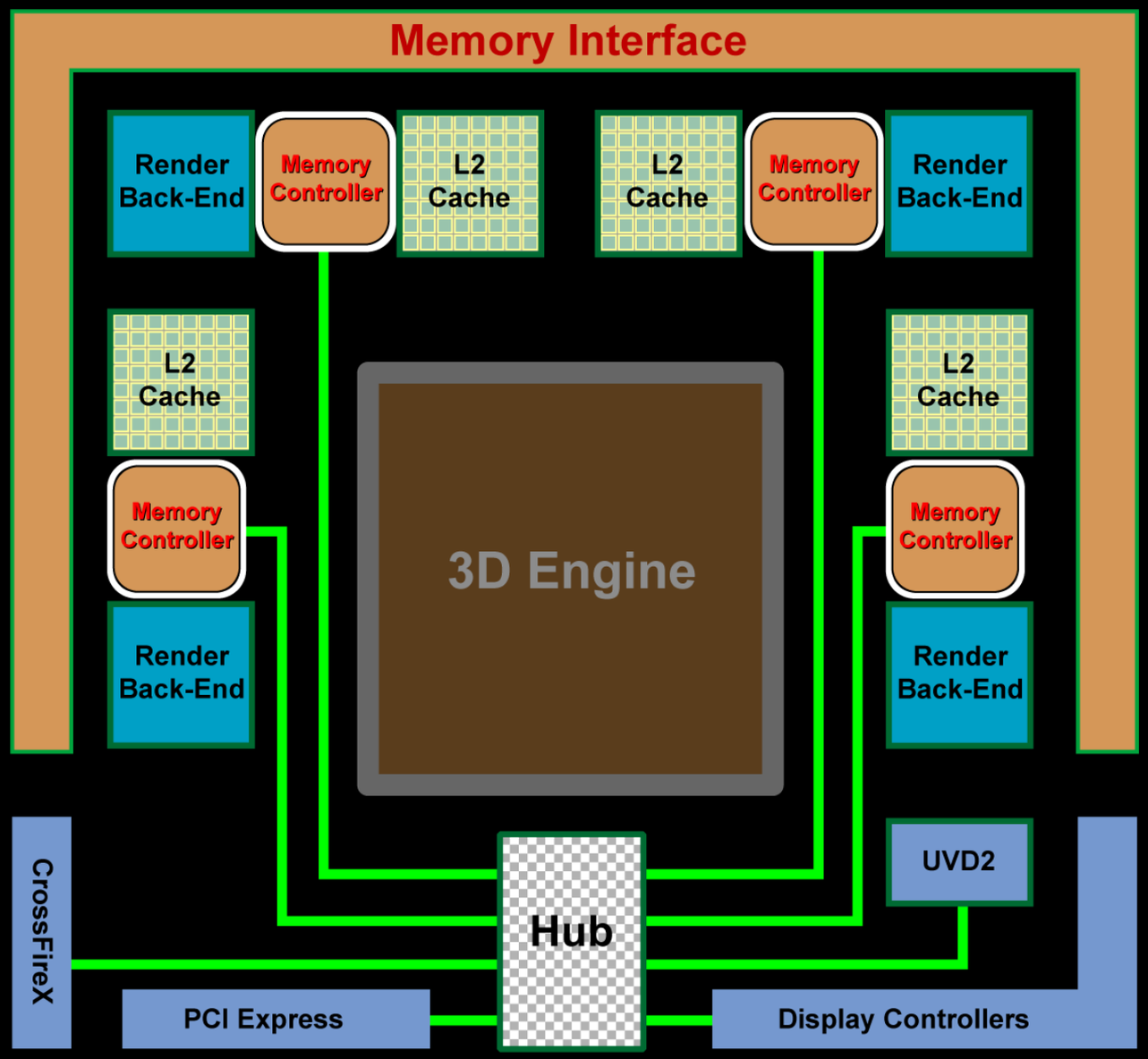
- New cache design
  - L2s aligned with memory channels
  - L1s store unique data per SIMD
    - 2.5x increase aggregate L1*
  - Separate vertex cache
  - Increased bandwidth
    - Up to 480 GB/sec of L1 texture fetch bandwidth
    - Up to 384 GB/sec between L1 & L2



* Comparing ATI Radeon™ HD 4800 series and ATI Radeon™ HD 3800 series

Memory Controller Architecture

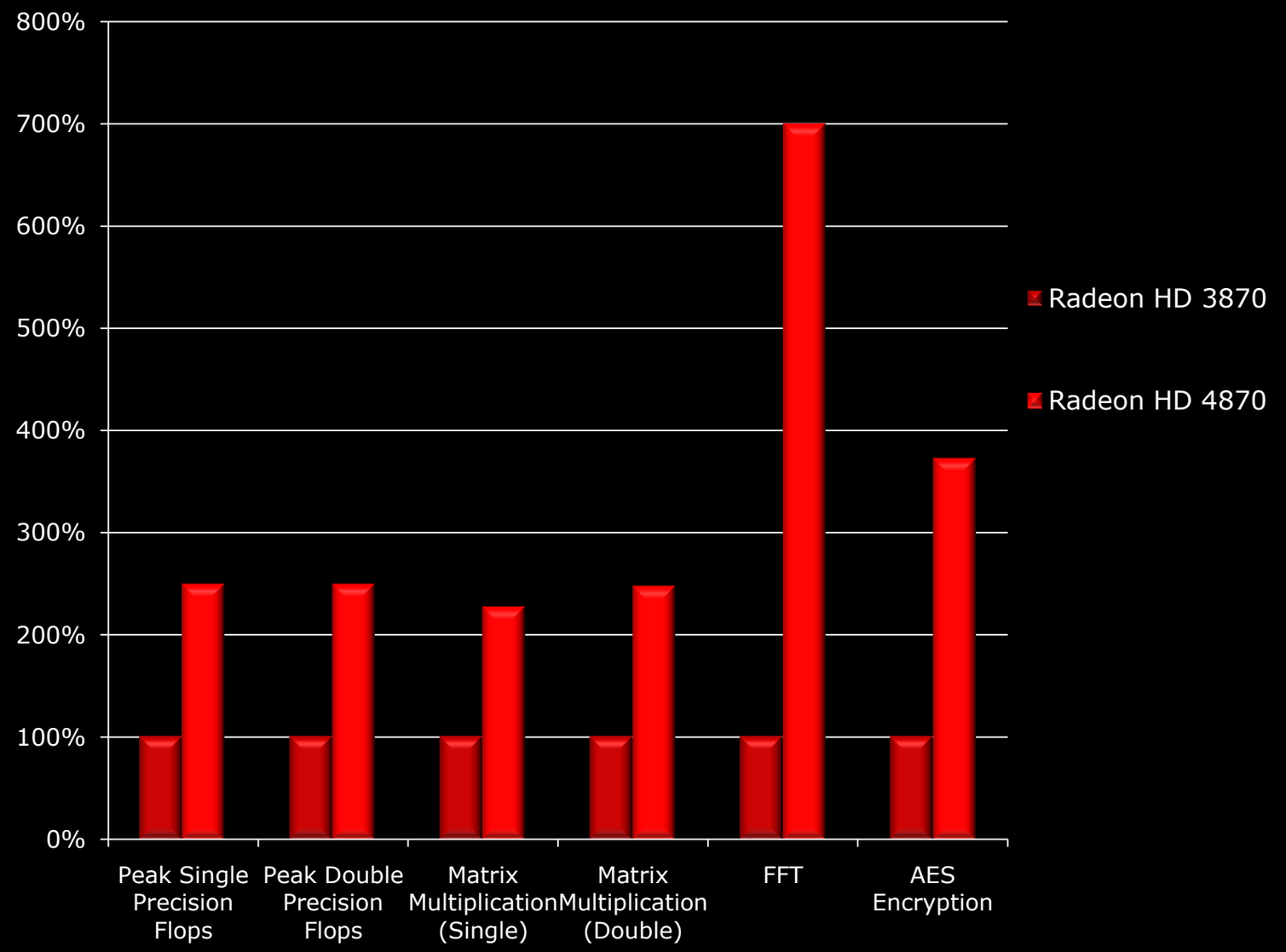
- New distributed design with hub
- Controllers distributed around periphery of chip, adjacent to primary bandwidth consumers
- Memory tiling & 256-bit interface allows reduced latency, silicon area, and power consumption
- Hub handles relatively low bandwidth traffic
  - PCI Express, CrossFireX interconnect, UVD2, display controllers, intercommunication)



ATI Radeon™ HD 4800 Series Stream Architecture

- Several enhancements done for stream computing
  - Fast compute vector
  - Local and Global data shares
  - Fast Integer Processing
  - Fast Gather/Scatter

- Significant increases in performance on many important stream processing workloads



Internal AMD testing, CAL SDK version 1.1, Intel QX6800 CPU, Catalyst version 8.5

ATI Radeon™ HD 4870 Computation Highlights

>100 GB/s memory bandwidth
- 256b GDDR5 interface

Targeted for handling thousands of simultaneous lightweight threads

800 (160x5) stream processors
- 640 (160x4) basic units (FMAC, ADD/SUB, etc.)
  - ~1.2 TFlops theoretical peak
- 160 enhanced transcendental units (adds COS, LOG, EXP, RSQ, etc.)
- Support for INT/UINT in all units (ADD/SUB, AND, XOR, NOT, OR, etc.)
- 64-bit double precision FP support
  - 1/5 single precision rate (~250GFlops theoretical performance)

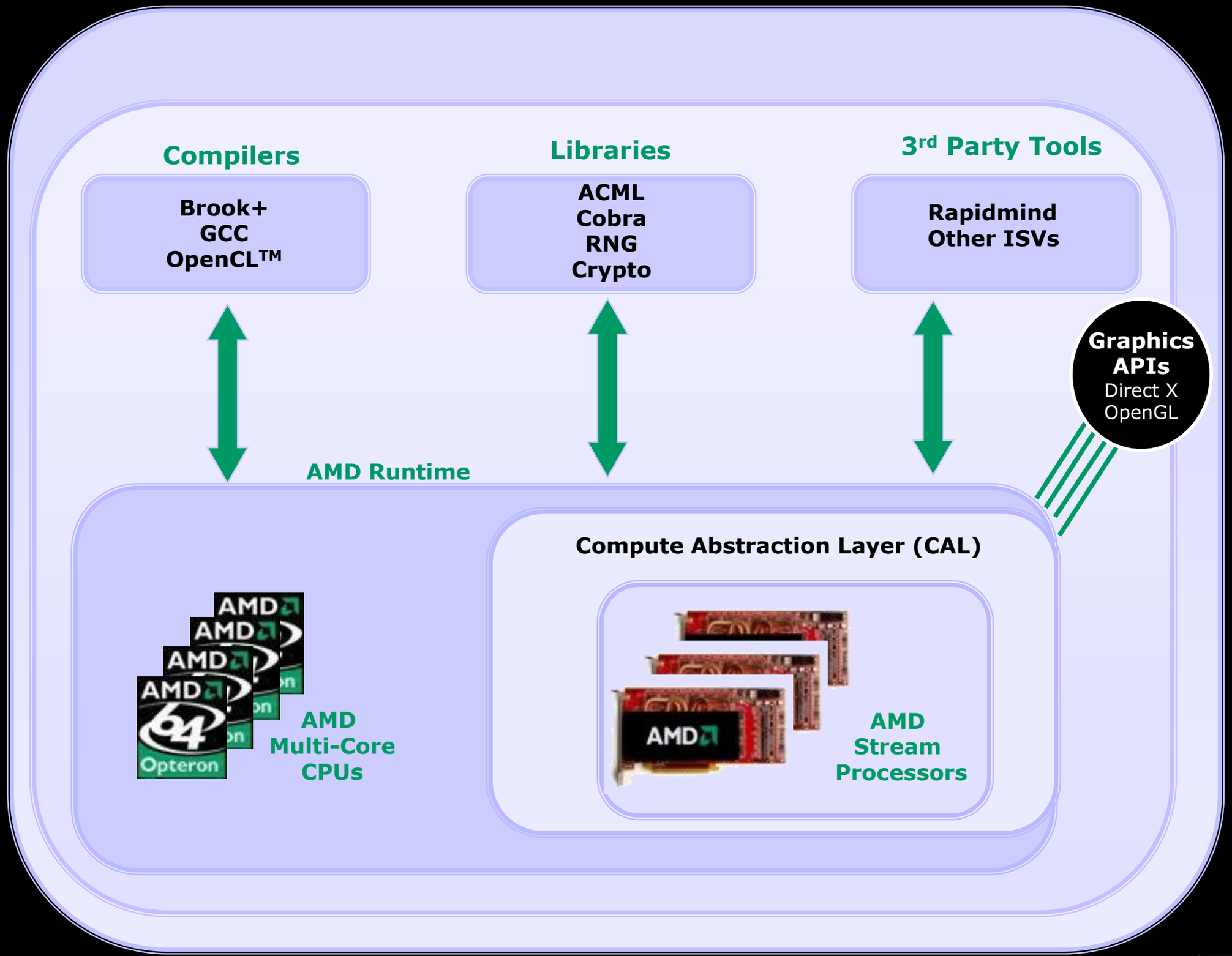
4 SIMDs -> 10 SIMDs
- 2.5X peak performance increase over ATI Radeon™ 3870
- ~1.2 TFlops FP32 theoretical peak
- ~250 GFlops FP64 theoretical peak

Scratch-pad memories
- 16KB per SIMD (LDS)
- 16KB across SIMDs (GDS)

Synchronization capabilities

Compute Shader
- Launch work without rasterization
- "Linear" scheduling
- Faster thread launch

Stream Computing SDK



What is Brook+?

Brook is an extension to the C-language for stream programming originally developed by Stanford University

Brook+ is an implementation by AMD of the Brook GPU spec on AMD's compute abstraction layer with some enhancements

Simple example - sum 2 arrays

```
float a[Y][X];
float b[Y][X];
float c[Y][X];

for(int i=0; i<Y; i++)
{
  for(int j=0; j<X; j++)
  {
     c[i][j] = a[i][j] + b[i][j];
  }
}
```

Simple example - sum 2 arrays

```
kernel void sum(float a<>, float b<>, out float c<>)
{
    c = a + b;
}

int main(int argc, char** argv)
{
    int i, j;
    float a<10, 10>;
    float b<10, 10>;
    float c<10, 10>;

    float input_a[10][10];
    float input_b[10][10];
    float input_c[10][10];

    for(i=0; i<10; i++) {
        for(j=0; j<10; j++) {
            input_a[i][j] = (float) i;
            input_b[i][j] = (float) j;
        }
    }

    streamRead(a, input_a);
    streamRead(b, input_b);

    sum(a, b, c);

    streamWrite(c, input_c);
    ...
}
```

Kernels – Program functions that operate on streams

Streams – collection of data elements of the same type which can be operated on in parallel.

Brook+ memory access functions

Brook+ kernels

```
kernel void sum(float a<>, float b<>, out float c<>)
{
    c = a + b;
}
```

Standard Streams - implicit and predictable access pattern

```
kernel void sum(float a[], float b[], out float c<>)
{
    int idx = indexof(c);
    c = a[idx] + b[idx];
}
```

Gather Streams - dynamic read access pattern

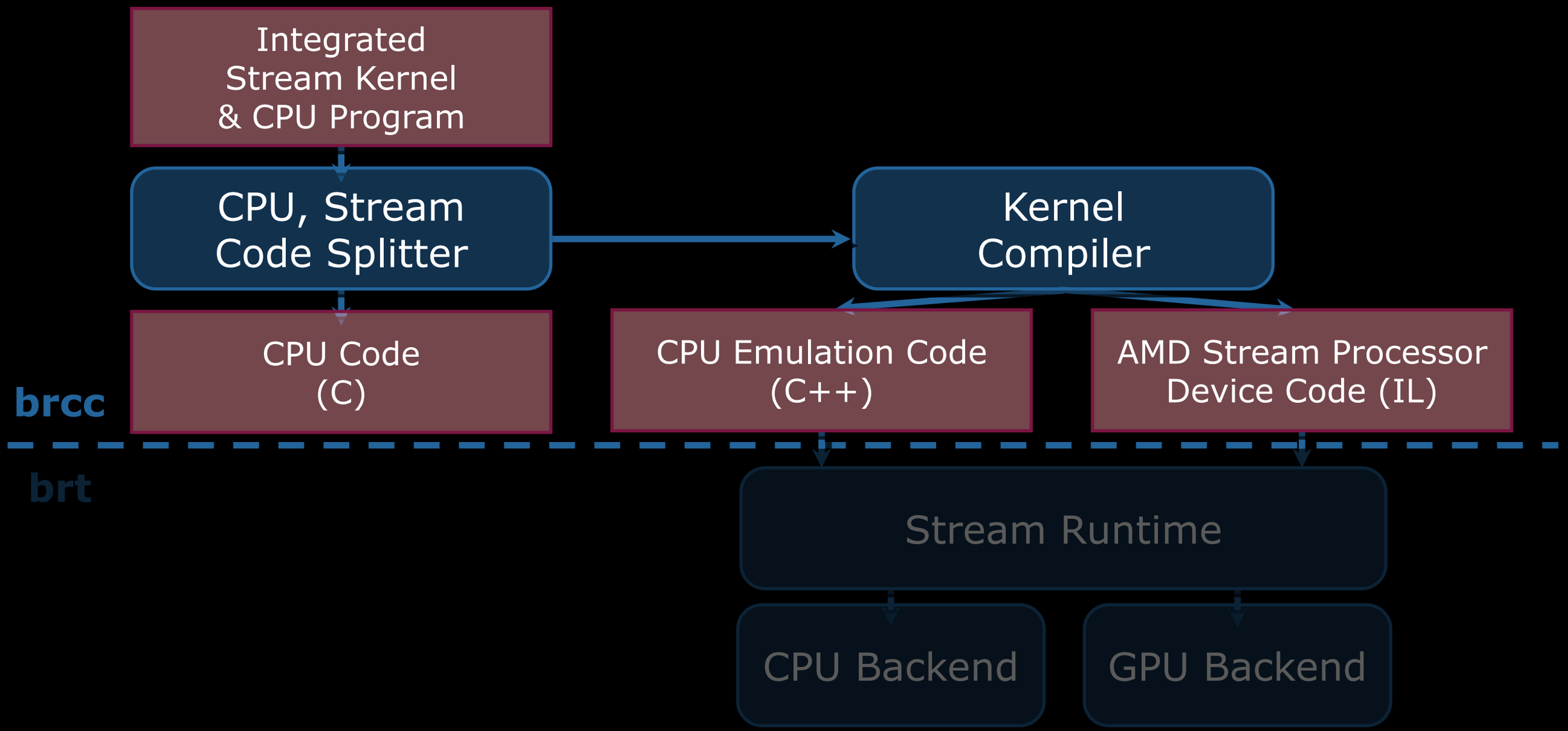
```
kernel void sum(float a<>, float b<>, out float c[])
{
    int idx = indexof(c);
    c[idx] = a + b;
}
```

Scatter Stream - dynamic write access pattern

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
+	+	+	+	+	+	+	+
b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]
=	=	=	=	=	=	=	=
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]

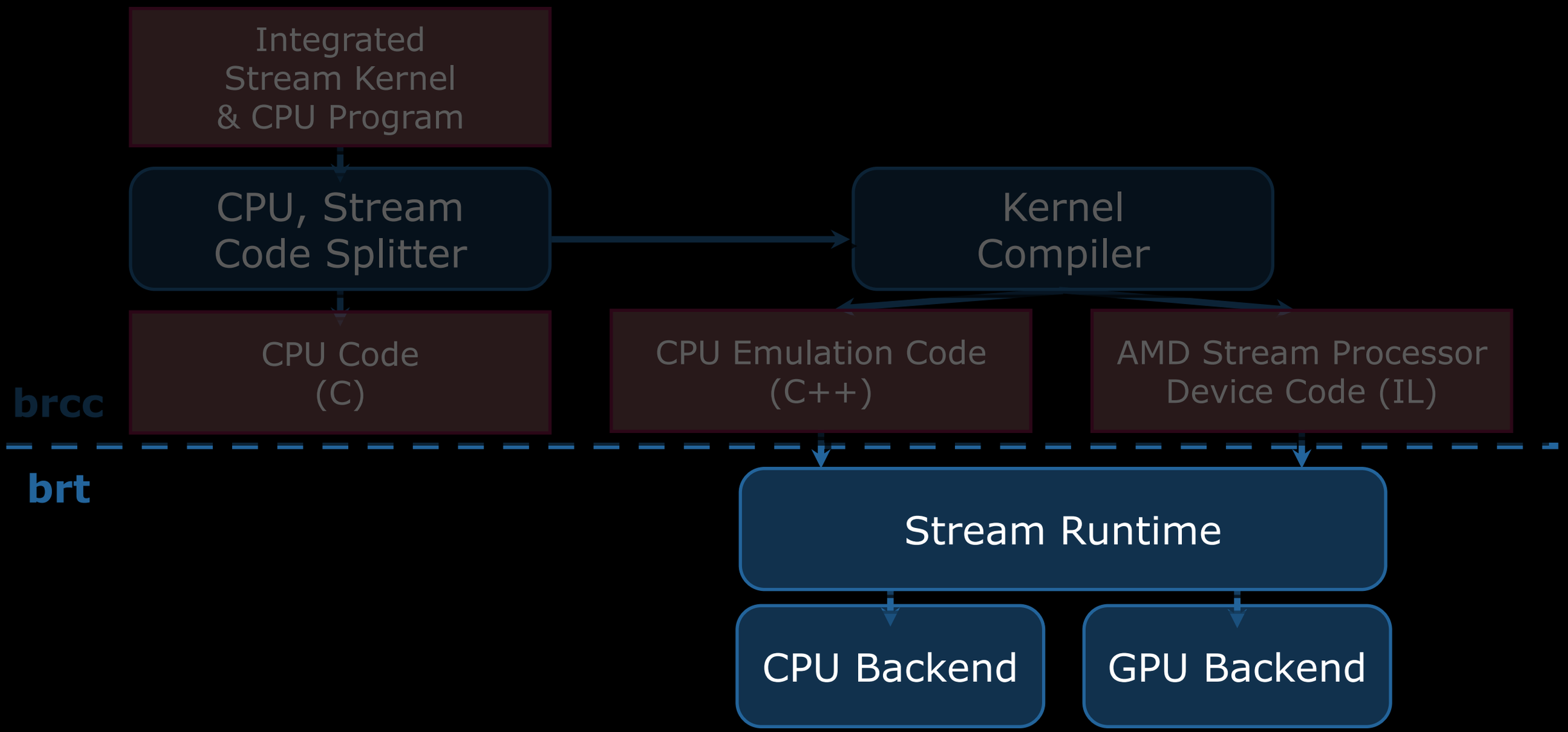
Brook+ compiler

Converts Brook+ files into C++ code. Kernels, written in C, are compiled to AMD's IL code for the GPU or C code for the CPU.



Brook+ runtime

IL code is executed on the GPU. The backend is written in CAL.



Brook+ features

Brook+ is an extension to the Brook for GPUs source code.

Features of Brook for GPUs relevant to modern graphics hardware are maintained.

Kernels are compiled to AMD's IL

Runtime uses CAL to execute on AMD GPUs

- CAL runtime generates ASIC specific ISA dynamically

Double precision

Integer support

Scatter (mem-export)

Asynchronous CPU->GPU transfers (GPU->CPU still synchronous)

OpenCL

KHRONOS GROUP

CPUs
Multiple cores driving performance increases

Multi-processor programming – e.g. OpenMP

Emerging Intersection

OpenCL
Heterogenous Computing

GPUs
Increasingly general purpose data-parallel computing
Improving numerical precision

Graphics APIs and Shading Languages

OpenCL – Open Computing Language
Open, royalty-free standard for portable, parallel programming of heterogeneous parallel computing CPUs, GPUs, and other processors

AMD ATI RADEON PREMIUM GRAPHICS

OpenCL

- Use all computational resources in system
  Program GPUs, CPUs, and other processors as peers
  Support both data- and task- parallel compute models

- Efficient C-based parallel programming model
  Abstract the specifics of underlying hardware

- Abstraction is low-level, high-performance but device-portable
  Approachable – but primarily targeted at expert developers
  Ecosystem foundation – no middleware or "convenience" functions

- Implementable on a range of embedded, desktop, and server systems
  HPC, desktop, and handheld profiles in one specification

- Large industry effort:
  3DLABS, Activision Blizzard, AMD, Apple, ARM, Barco, Broadcom, Codeplay, Electronic Arts, Ericsson, Freescale, HI, IBM, Intel, Imagination Technologies, Kestrel Institute, Motorola, Movidia, Nokia, NVIDIA, QNX, RapidMind, Samsung, Seaweed, Takumi, Texas Instruments and Umeå University

For more information

AMD Stream Computing SDK
http://ati.amd.com/technology/streamcomputing/

OpenCL
http://www.khronos.org/opencl/

Disclaimer and Attribution

DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION