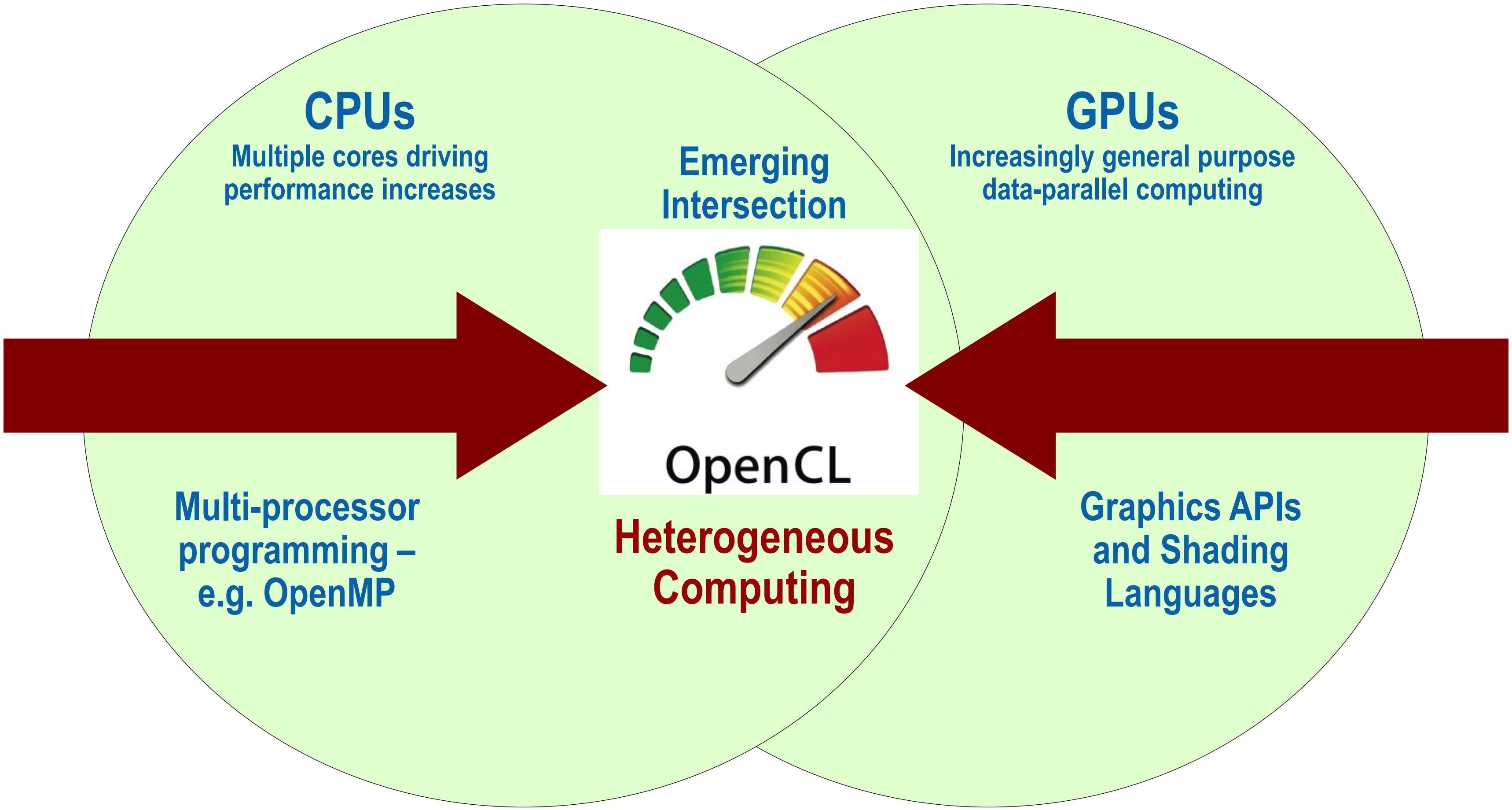
Open 2.0: Unlocking the power of your heterogeneous platform

Tim Mattson
Intel Labs

Industry Standards for Programming Heterogeneous Platforms



OpenCL – Open Computing Language

Open, royalty-free standard for portable, parallel programming of heterogeneous combinations of CPUs, GPUs, and other processors

OpenCL as Parallel Compute Foundation

C++ AMP
Shevlin Park
Uses Clang and LLVM

OpenCL HLM
C++ syntax & compiler extensions

WebCL
JavaScript binding to OpenCL for initiation of OpenCL C kernels

Aparapi
Java language extensions for parallelism

River Trail
Language extensions to JavaScript

PyOpenCL
Python wrapper around OpenCL

Harlan
High level language for GPU programming

OpenCL

OpenCL provides vendor optimized,
cross-platform, cross-vendor
access to heterogeneous compute
resources

Announcing at SC13

OpenCL 2.0 released!

OpenCL

OpenCL 2.0
Significant enhancements to memory and execution models to expose emerging hardware capabilities and provide increased flexibility, functionality and performance to developers

SPIR 1.2 … to be released soon!

SPIR (Standard Parallel Intermediate Representation)
Exploring LLVM-based, low-level Intermediate Representation for IP Protection and as target back-end for alternative high-level languages

Announcing at SC13

OpenCL 2.0 released!

OpenCL 2.0
Significant enhancements to memory and execution models to expose emerging hardware capabilities and provide increased flexibility, functionality and performance to developers

OpenCL

SPIR 1.2 released!

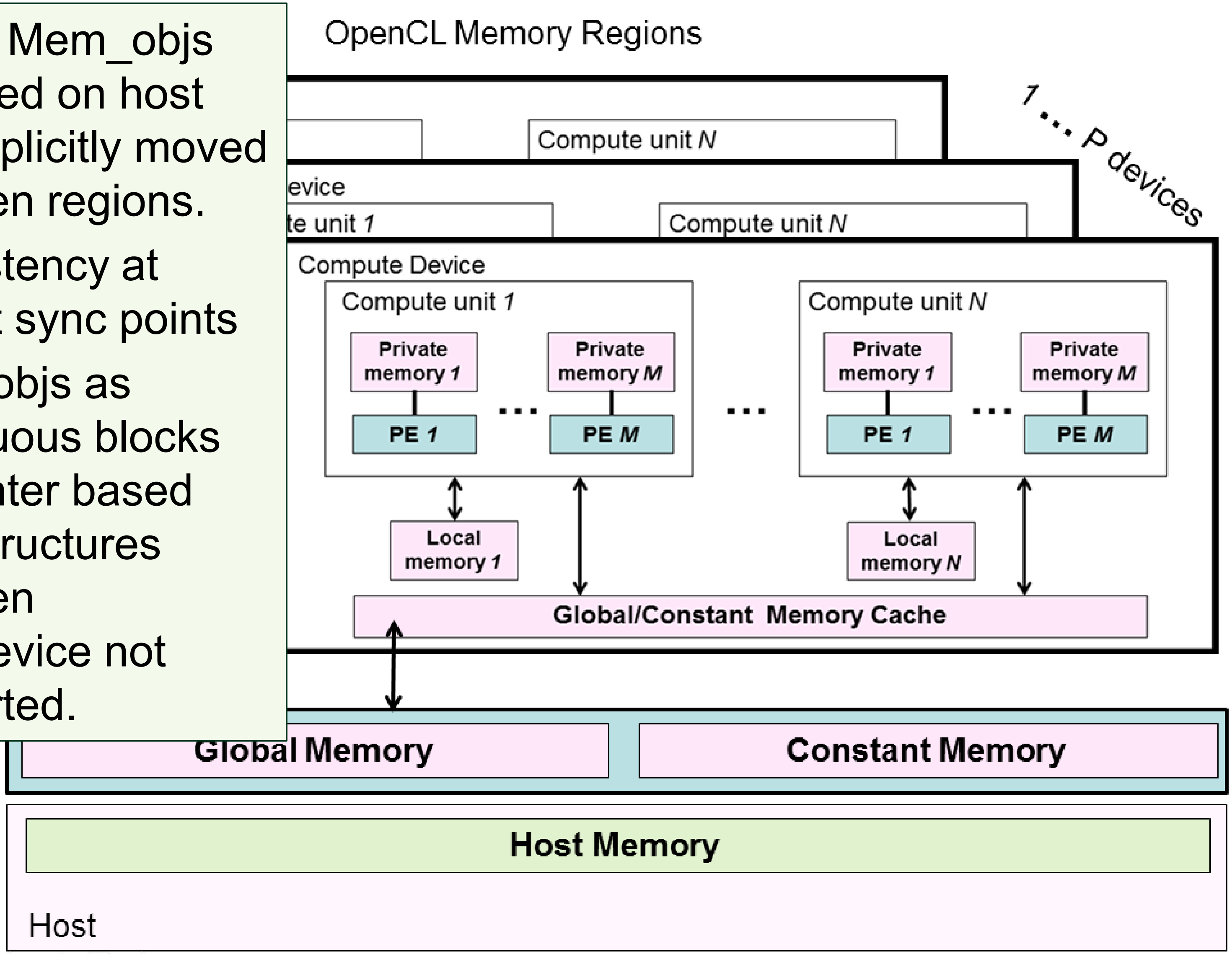
SPIR (Standard Parallel Intermediate Representation)
Exploring LLVM-based, low-level Intermediate Representation for IP Protection and as target back-end for alternative high-level languages

Goals

- Ease of Use
- Performance Improvements
- Enable New Programming Patterns
- Well-defined Execution & Memory Model
- Improve OpenCL / OpenGL sharing

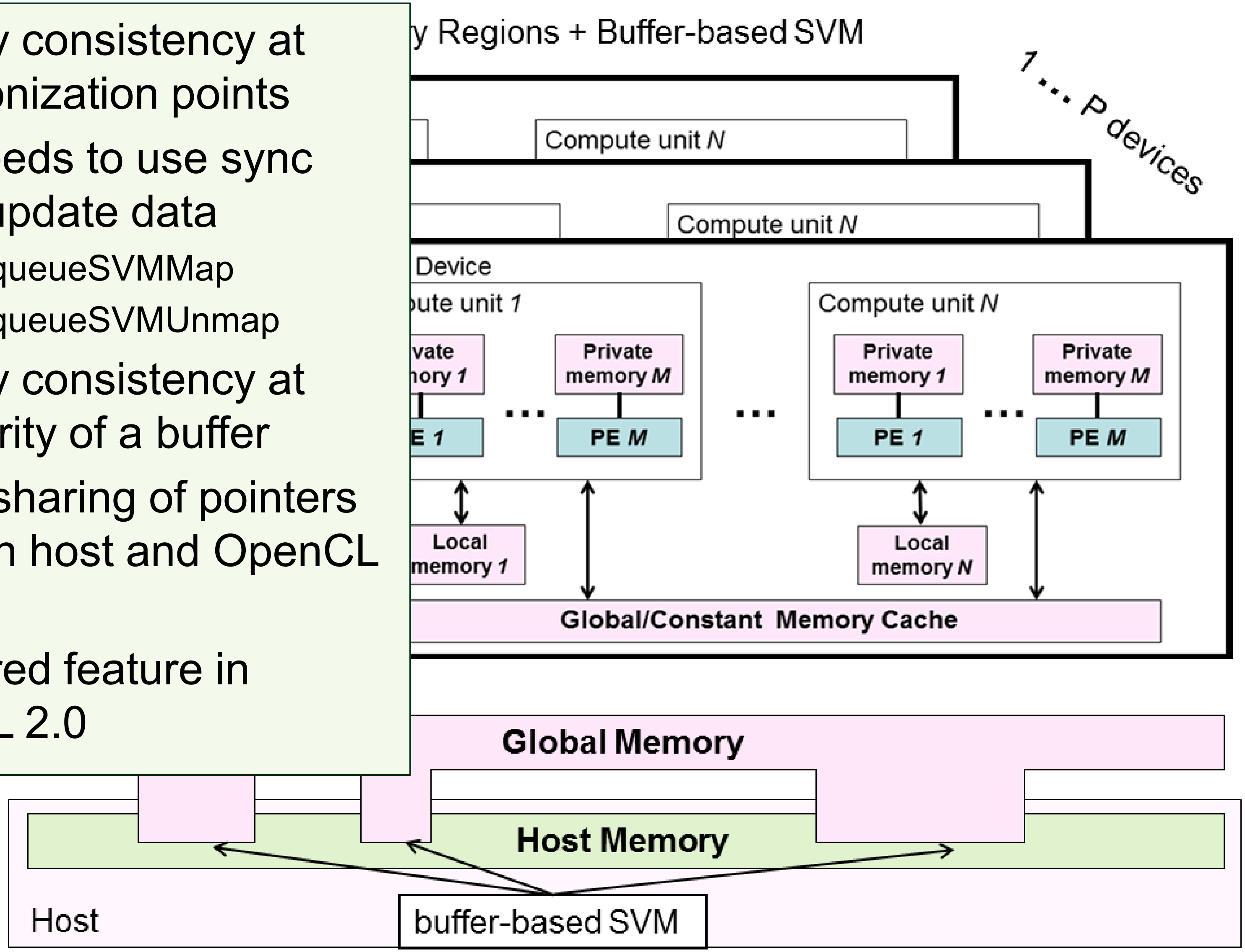
OpenCL 1.X memory Regions

- Global Mem_objs allocated on host and explicitly moved between regions.
- Consistency at explicit sync points
- Mem_objs as contiguous blocks … pointer based data structures between host/device not supported.



OpenCL 2.0: coarse grained SVM

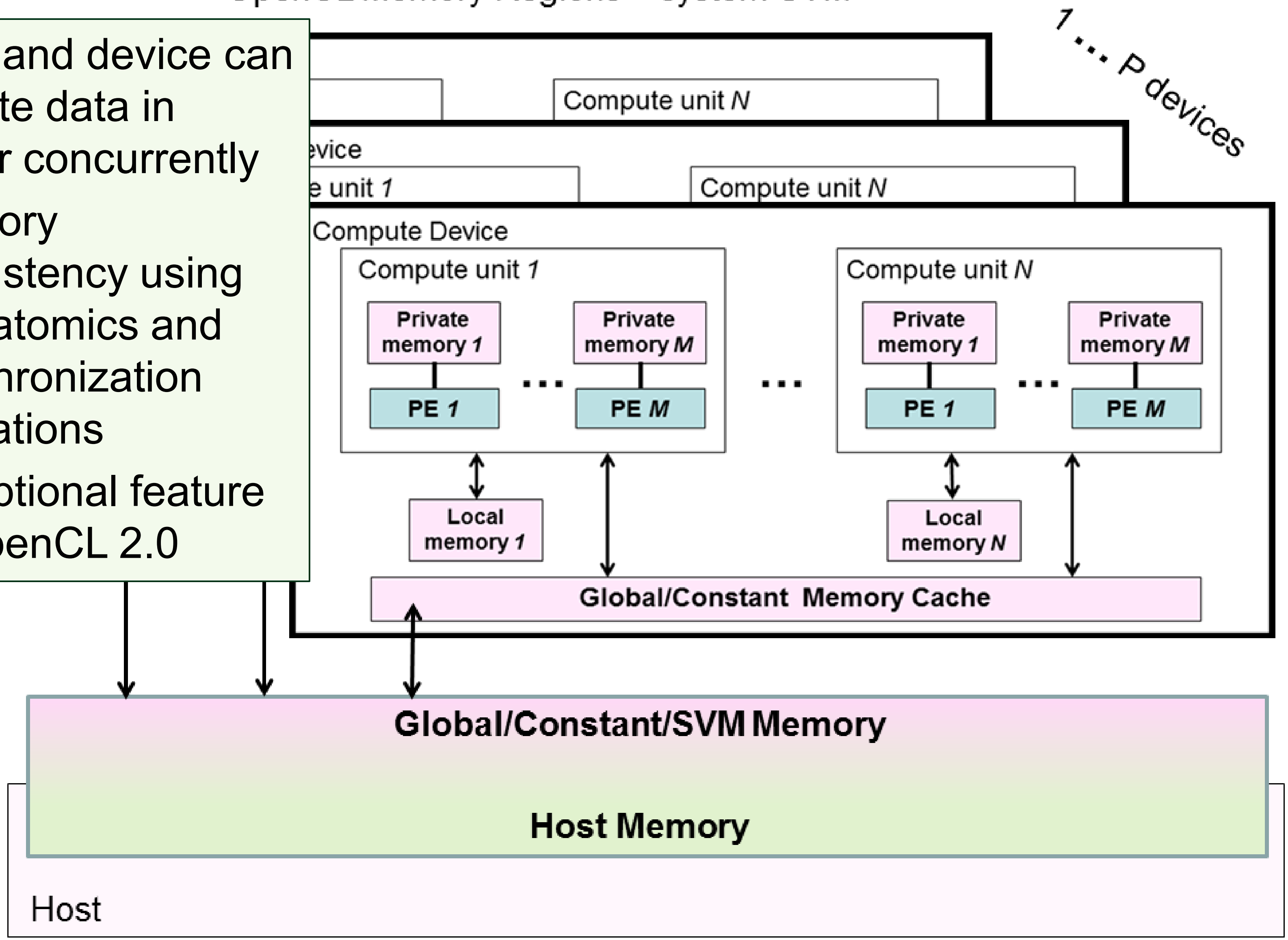
- Memory consistency at synchronization points
- Host needs to use sync API to update data
  - clEnqueueSVMMap
  - clEnqueueSVMUnmap
- Memory consistency at granularity of a buffer
- Allows sharing of pointers between host and OpenCL device
- A required feature in OpenCL 2.0



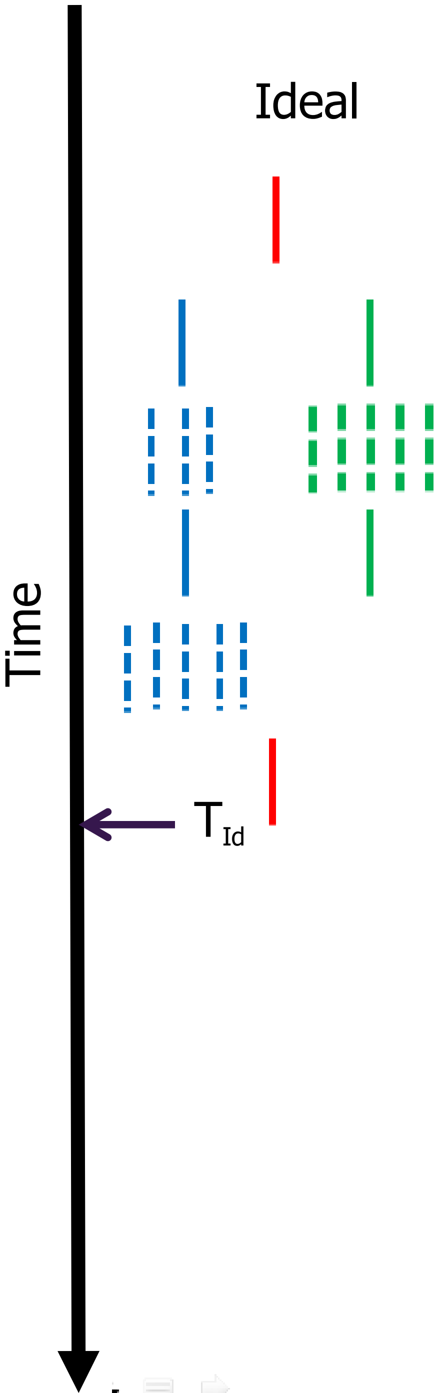
OpenCL 2.0: fine grained/System SVM

- Host and device can update data in buffer concurrently
- Memory consistency using C11 atomics and synchronization operations
- An optional feature in OpenCL 2.0

OpenCL Memory Regions + system SVM

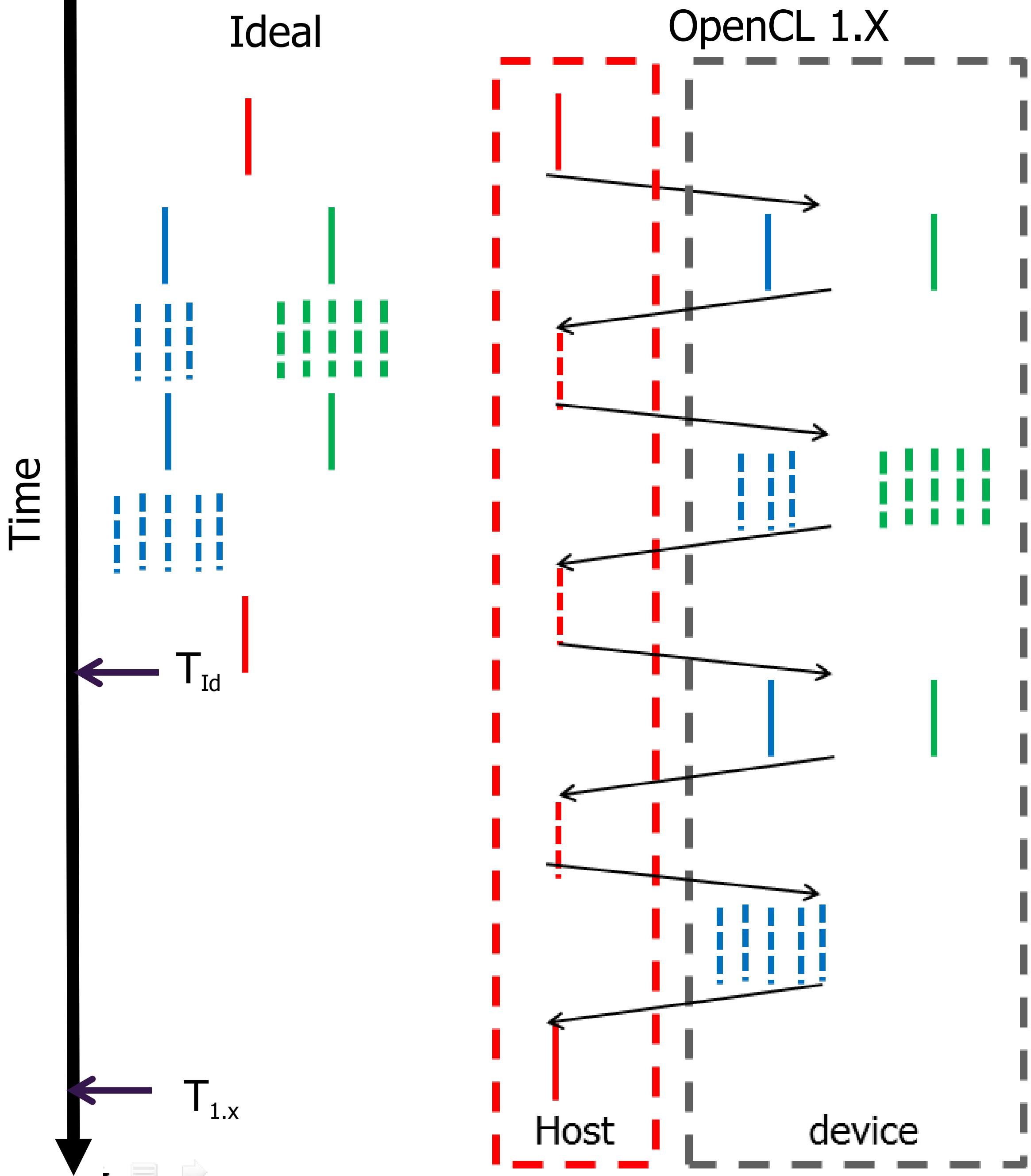


Nested Parallelism



Consider an algorithm as a task graph where the task structure is determined at runtime based on the input data.

Nested Parallelism

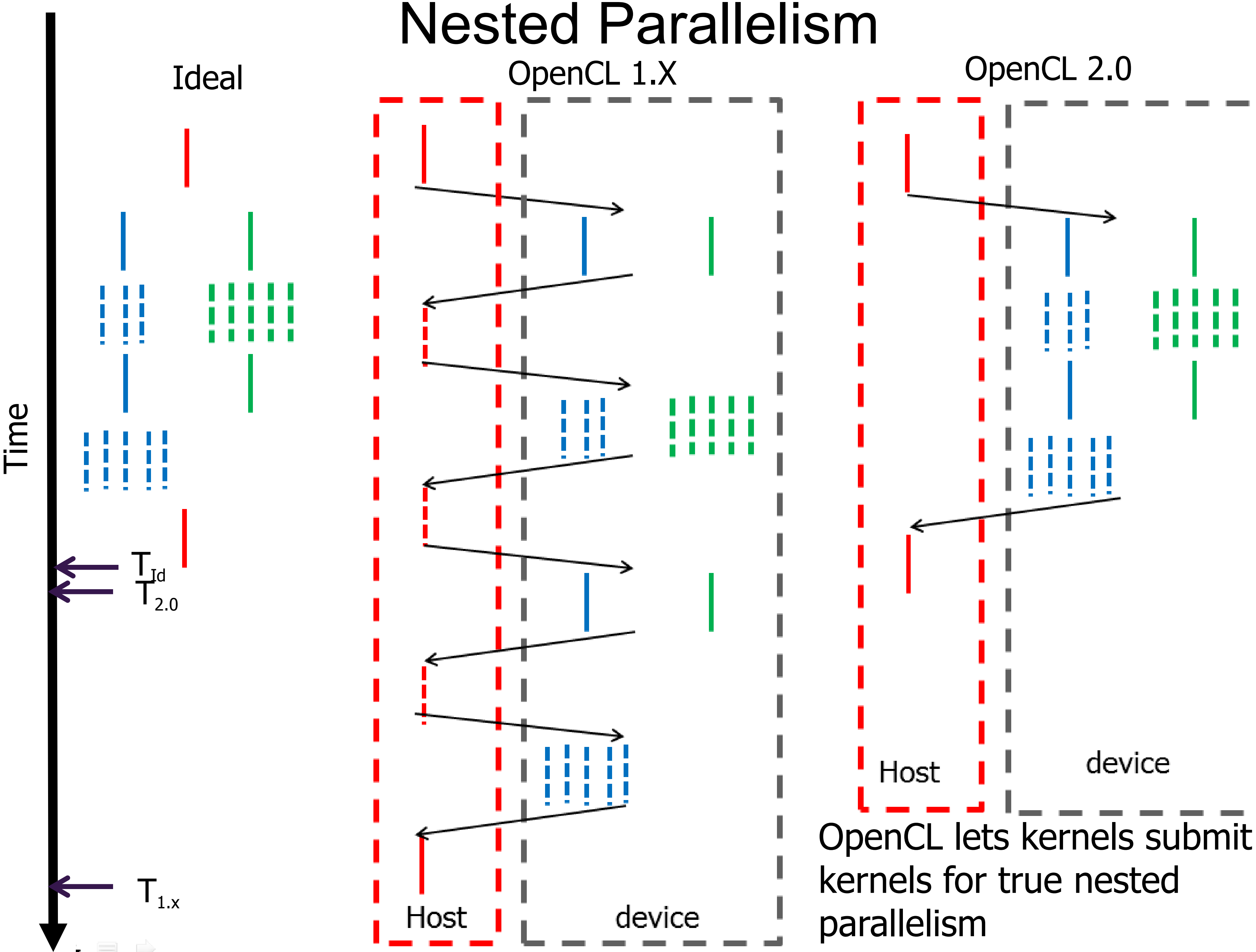


With OpenCL 1.X only the host can submit kernels for execution.

So after each task ends, it must copy data back to the host so the host knows which kernels to submit in the next phase.

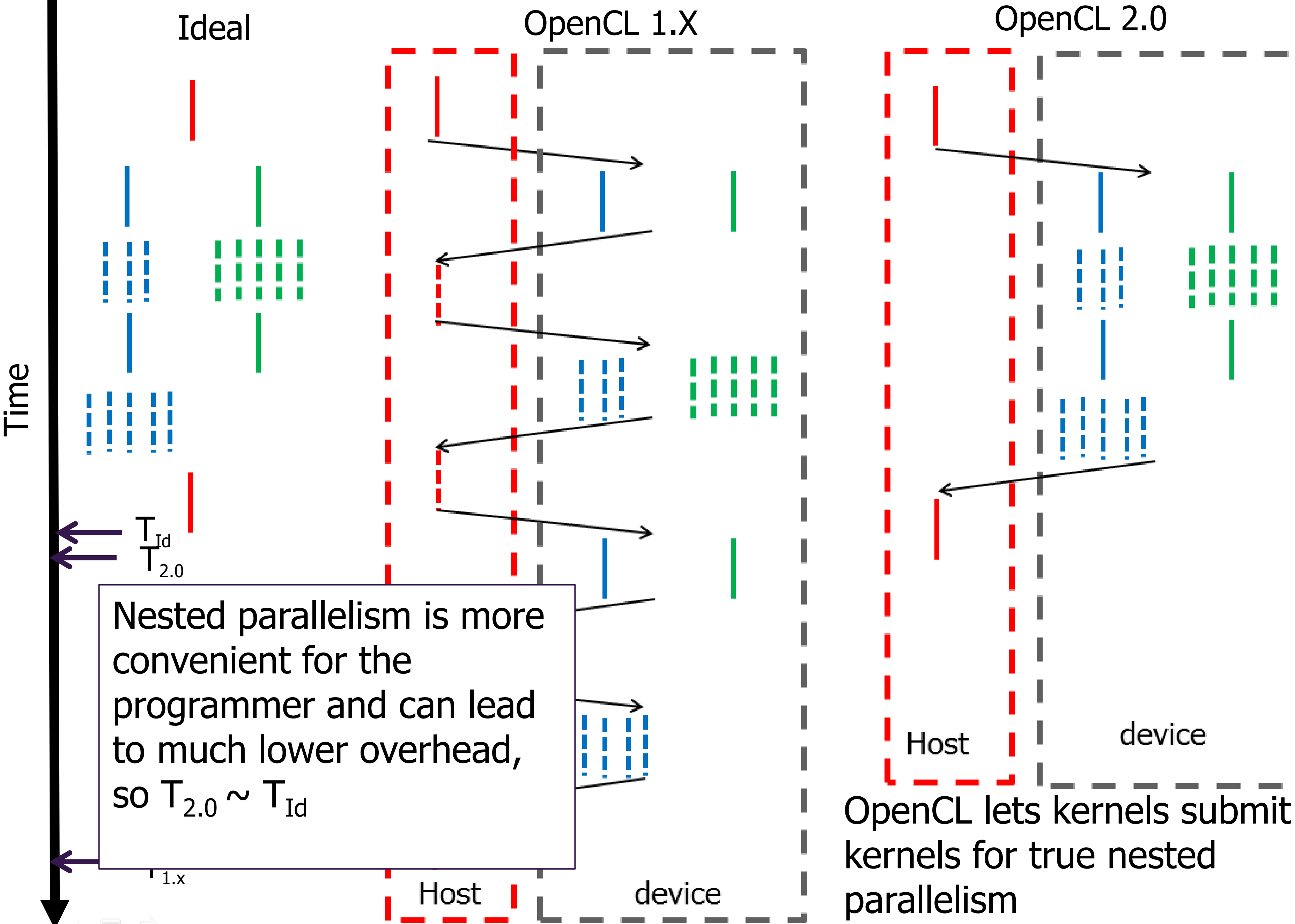
This requires extra code (the red dotted lines) and overhead resulting in $T_{1.x} >> T_{Id}$

Nested Parallelism



OpenCL lets kernels submit kernels for true nested parallelism

Nested Parallelism



Nested parallelism is more convenient for the programmer and can lead to much lower overhead, so $T_{2.0} \sim T_{Id}$

OpenCL lets kernels submit kernels for true nested parallelism

Nested Parallelism

- Use clang Blocks to describe kernel to queue

```
kernel void my_func(global int *a, global int *b)
{
    …
    void (^my_block_A)(void) =
              ^{
                    size_t id = get_global_id(0);
                    b[id] += a[id];
               };

    enqueue_kernel(get_default_queue(),
                      CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                        ndrange_1D(…),
                        my_block_A);
}
```

Generic Address Space

- OpenCL 2.0 no longer requires an address space qualifier for arguments to a function that are a pointer to a type
  - Except for kernel functions
- Generic address space assumed if no address space is specified
- Makes it really easy to write functions without having to worry about which address space arguments point to

```
void
my_func (int *ptr, …)
{
      …
      foo(ptr, …);
      …
}

kernel void
foo(global int *g_ptr,
      local int *l_ptr, …)
{
      …
      my_func(g_ptr, …);
      my_func(l_ptr, …);
}
```

Other OpenCL 2.0 Features

- What made it in
  - Memory model based on C'11 … includes atomics, and memory orders
  - Pipe memory objects to support pipeline algorithms.
  - Flexible work-group sizes
  - Expanded set of work-group functions (collective operations across work-items in a single work-group).
    - `broadcast, reduction, vote (any & all), prefix sum`
  - … and much more
- But we still lack …
  - Support for a C++ kernel programming language.
  - Ability to write a wide range of algorithms that require concurrency guarantees (e.g. try writing a spin lock in OpenCL).

Announcing at SC13

OpenCL 2.0 released!

OpenCL 2.0

Significant enhancements to memory and execution models to expose emerging hardware capabilities and provide increased flexibility, functionality and performance to developers

OpenCL

SPIR 1.2 … to be released soon!

SPIR (Standard Parallel Intermediate Representation)

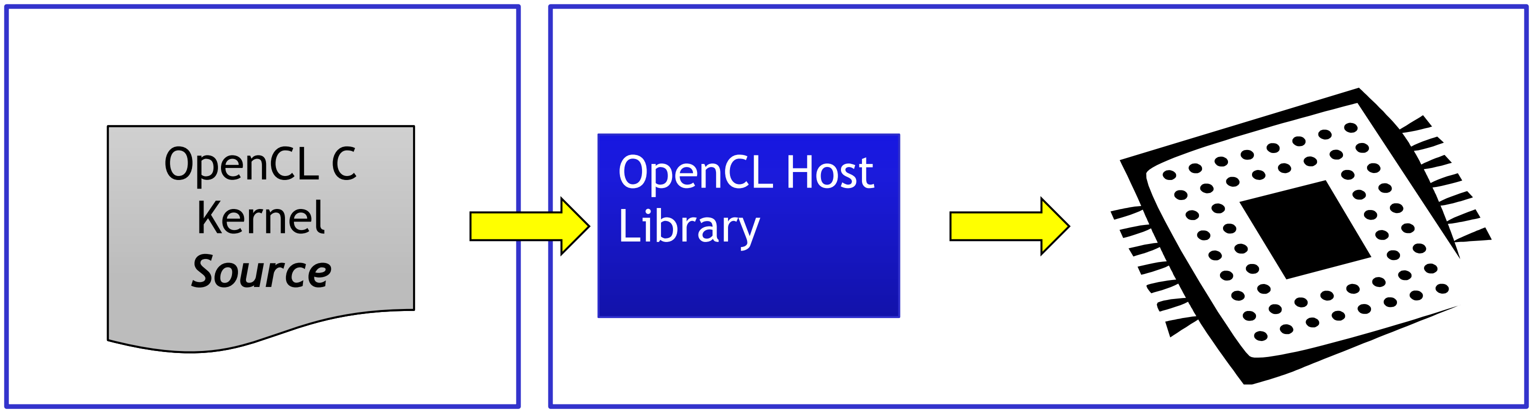
Exploring LLVM-based, low-level Intermediate Representation for IP Protection and as target back-end for alternative high-level languages

SPIR Market Goals

- **Standard Portable Intermediate Representation**
- **Enhance ISV experience**
  - Avoid IP exposure: Don't ship source
  - Manage device/driver/vendor proliferation
  - Avoid market lag
- **Support 3rd party compilers**
  - Just need new front end
  - Long term: C++, domain specific, …
- **User choice**
  - Retarget a shipped application to new devices, new vendors

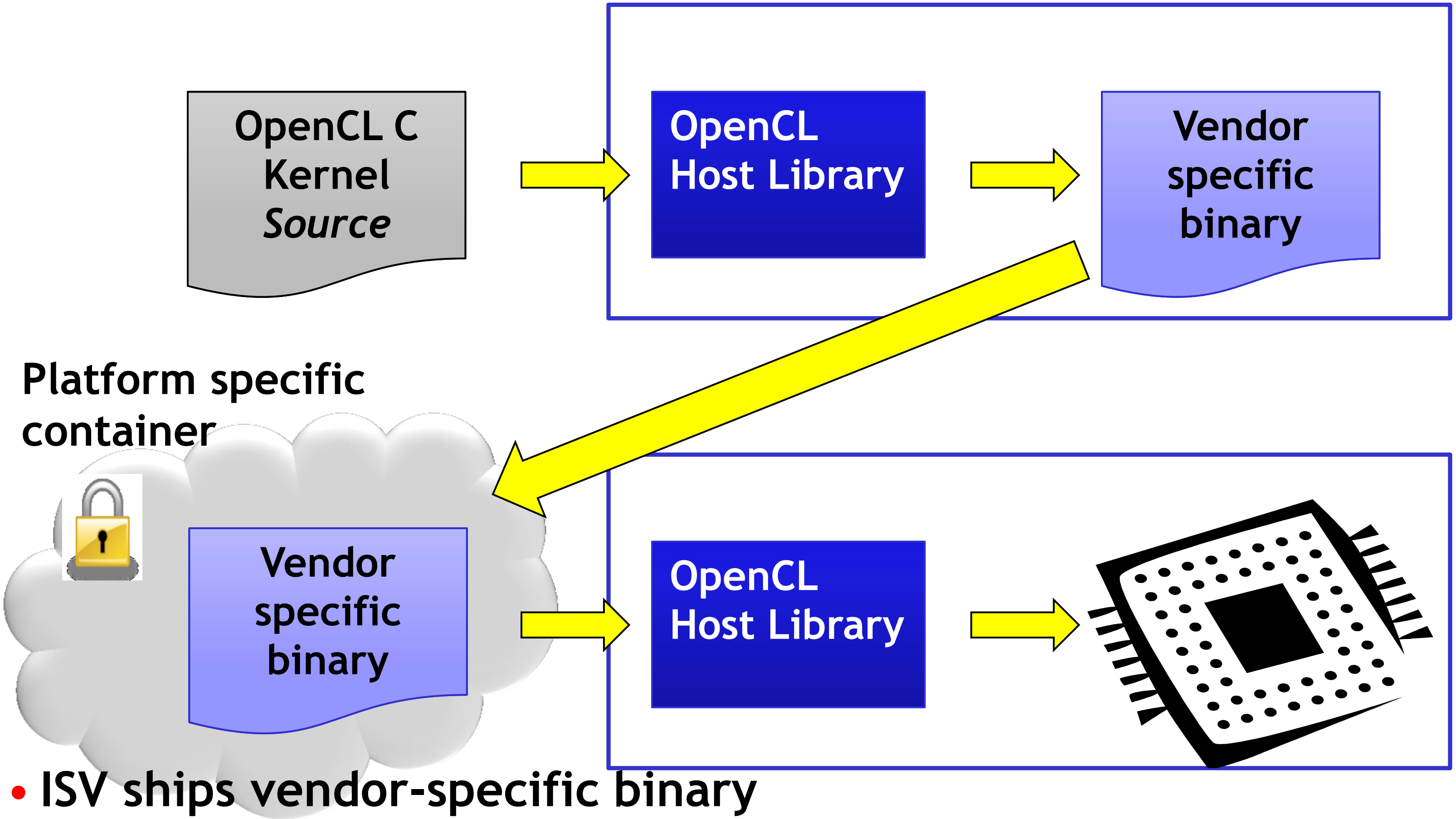
Opportunity to unleash industry innovation:
Domain Specific Languages, C++ Compilers ….

Non-SPIR Source Compilation Flow



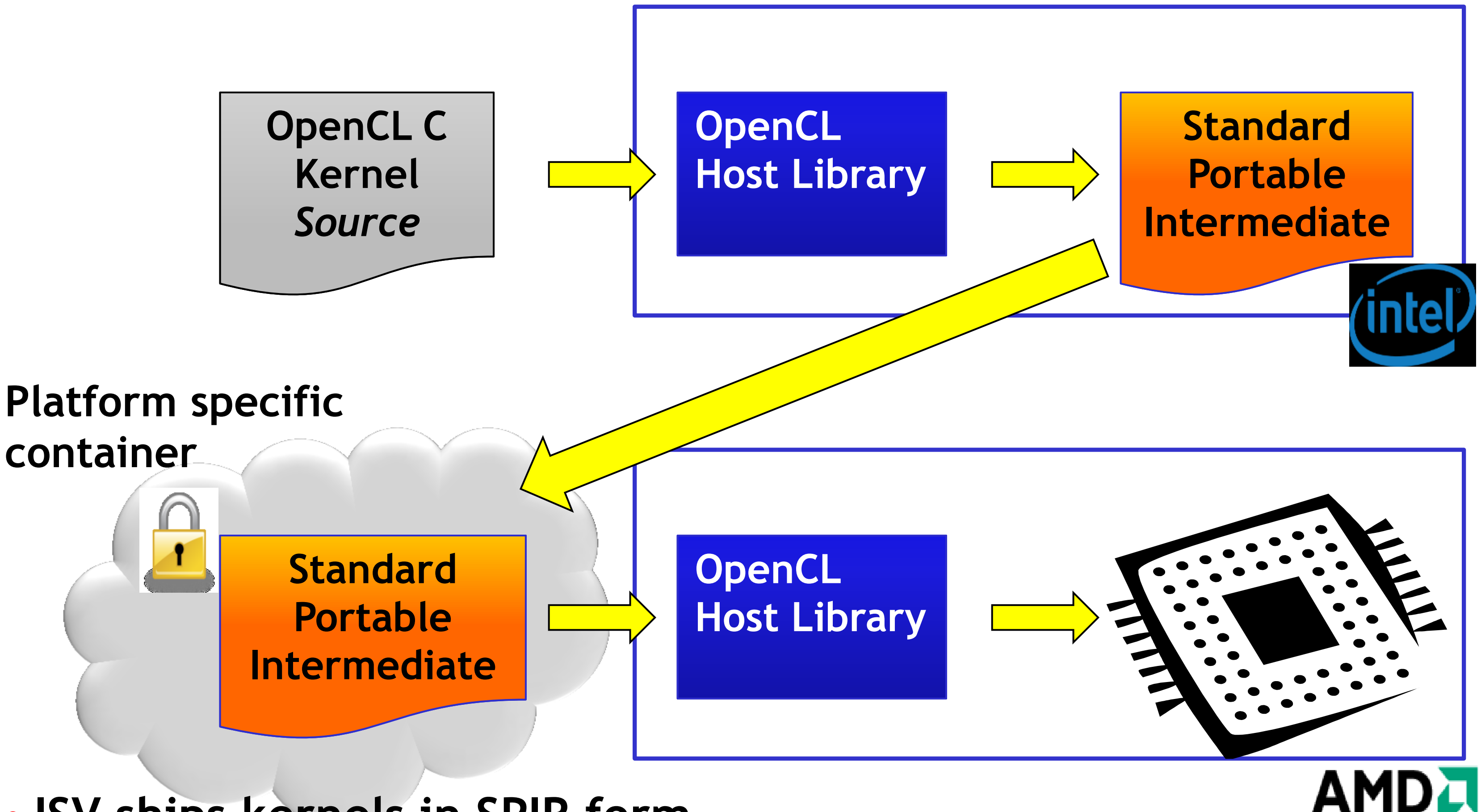
- Supports only OpenCL C
- ISV ships their kernel source
  - Exposes IP

SPIR Binary compilation flow



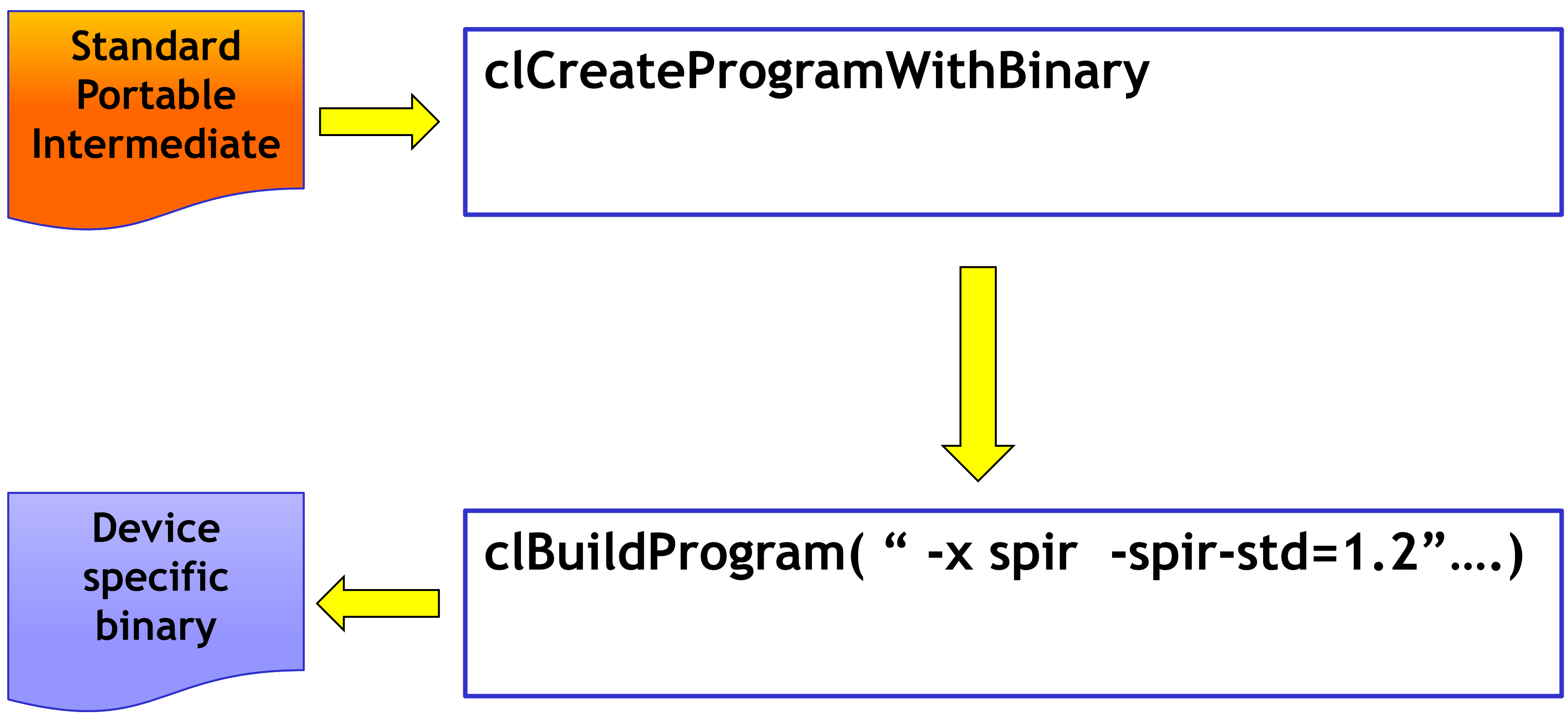
- ISV ships vendor-specific binary
  - Proliferation: devices, driver revisions, vendors
  - Market-lagging: target shipped products

SPIR flow

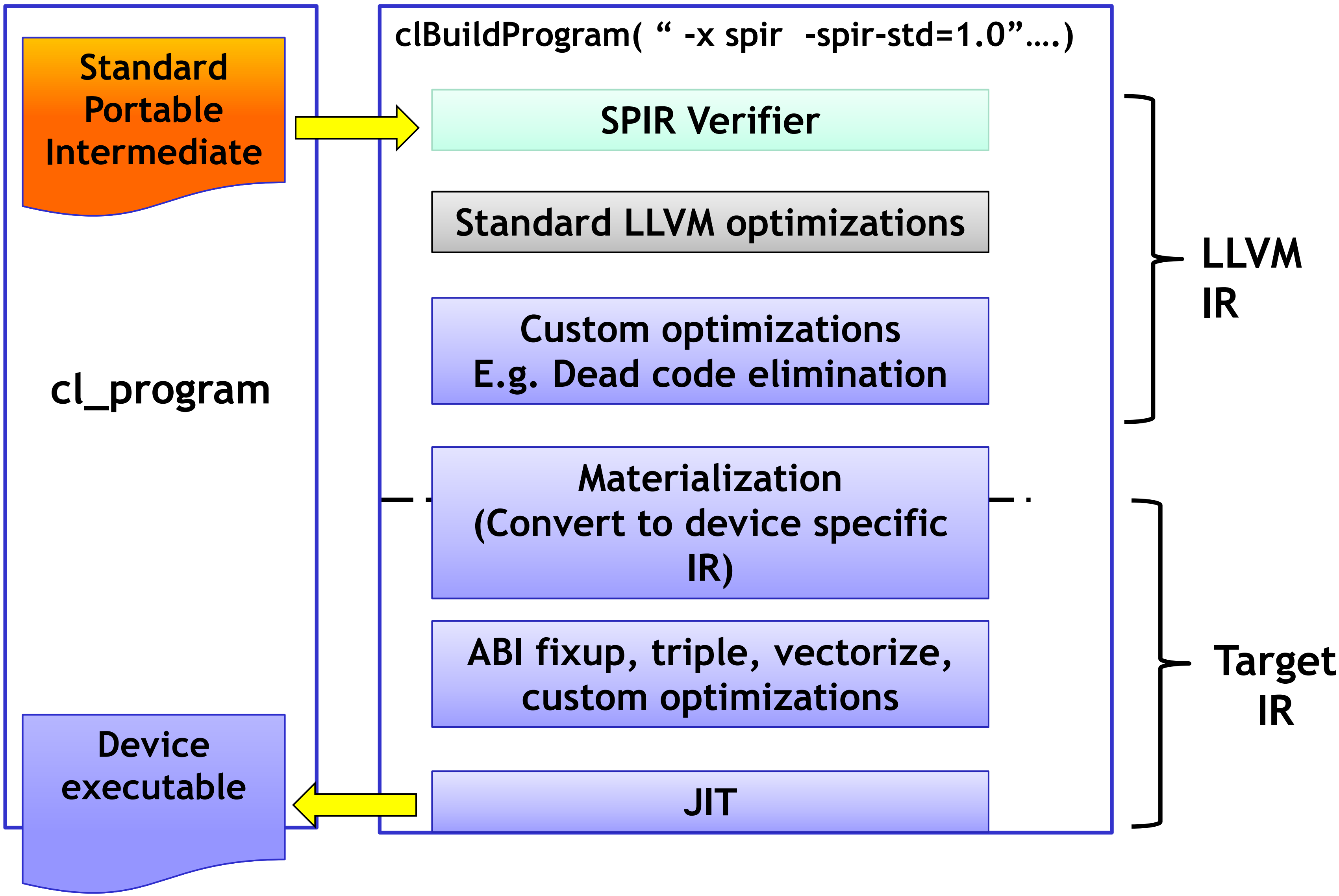


- ISV ships kernels in SPIR form
- Customer runs application on platform of their choice

Sample SPIR Consumption Flow



Sample SPIR flow: Room for optimizations



OpenCL SPIR Status

- **OpenCL 1.2 Extension standardizes an API for reading SPIR files**
  - cl_khr_spir
- **Final specification … to be released soon!**
  - Supports newer OpenCL 1.2 features
    - MSAA, Depth, depth stencil images
- **Pushing patches into open source Clang to GENERATE SPIR…**
  - Clang 3.2 trunk will be able to generate SPIR
    - 32-bit separate from 64-bit, Little endian only
    - Many low level changes: e.g. encode kernel arg info
- **Using LLVM 3.2 to CONSUME SPIR**
  - Generate optimized LLVM IR
  - Convert to target IR and JIT to executable

Summary

- OpenCL is evolving rapidly to match HW innovation in heterogeneous platforms.
- We are announcing at SC13:
  - OpenCL 2.0
  - SPIR (Standard Portable Intermediate Representation).
- You can learn more at our web site:

  www.khronos.org/opencl/

Visit us on the SC13 exhibit floor

Booths 126, 2713

ALL PROGRAMMABLE™
at Alpha Data
Booth 4237

Booth 613

Booth 3725

Booth 1113

OpenCL

Booth 4137

4332

Booth 2718

Booth 3109

Booth 3141

Booths 2501, 2701