

Umpire: Next-Generation Memory and Resource Management

CoE Performance Portability Meeting

David Beckingsale & Rich Hornung

August 22nd 2017



Upcoming hardware platforms have complex sets of memory and execution resources

- Technology specific APIs force application developers to commit to one implementation
- For programming models, this can be mitigated with approaches like RAJA & Kokkos
- We need a similar approach for memory and execution resources

- Umpire API will be driven by application needs and library requirements

Umpire Motivation

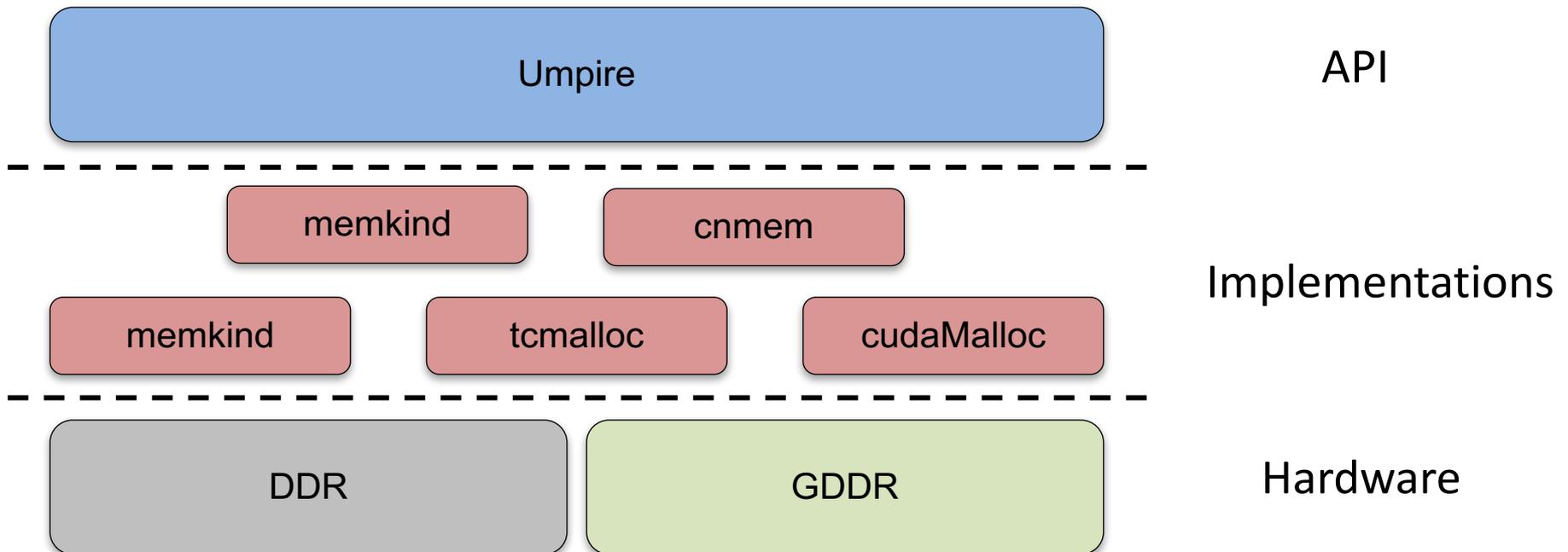
- How can applications and libraries co-ordinate using limited memory resources?
- How can we support flexible allocation strategies for different allocation types (e.g. temporary arrays)?
- How can data be moved between places in the memory hierarchy?

Umpire Goals

- Umpire is a resource management library that provides a unified high-level API for discovery, provision, and management of memory on next-generation hardware architectures
- Decouple resource allocation from specific memory **spaces**, memory **allocators** and memory **operations**
- Provide introspection capability for these allocations, allowing applications and libraries make decisions based on allocation properties

Umpire will leverage Third-Party Libraries

- Don't reinvent the wheel
- Provide a unified, high-level and application-focused API for projects like tcmalloc, jemalloc, memkind, SICM

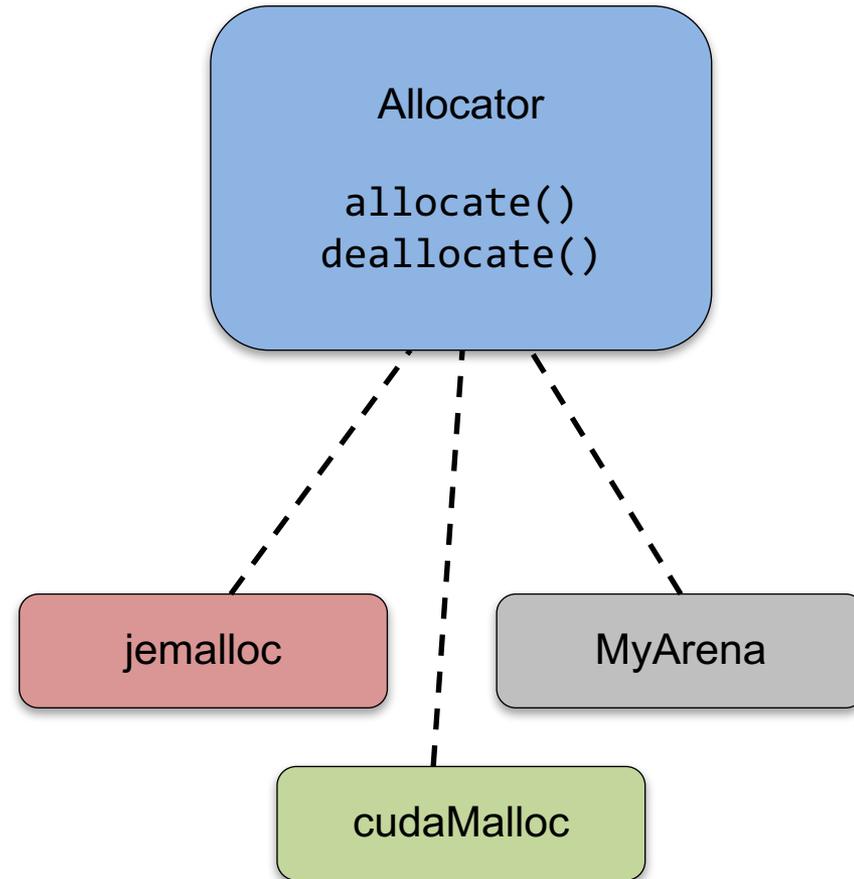


Umpire Concepts

- **Spaces** abstract a memory location, providing an interface to inspect properties and to allocate/free via a strategy.
- **Allocator** is a lightweight interface for making and querying memory allocations
- **AllocationStrategy** decouple allocations from the area they are made in, allowing for complex allocation mechanisms.
- **Operations** allow allocations to be moved from one space to another. These operations will be specialized based on the source and destination.

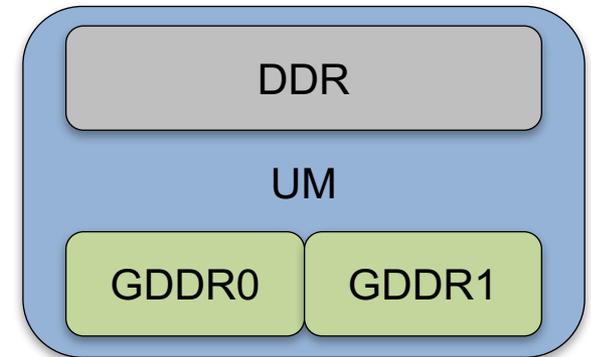
Allocators & AllocationStrategies

- Umpire user interface will be based around Allocators
- Allocator object hides specific implementation behind a unified interface
- Allows accessing a particular memory space the same way as a complex slab allocator
- Allocators are accessed by querying a central ResourceManager



Spaces

- Spaces are created based on accessibility of different memory resources
- For example, on a typical CPU-GPU node:
 - 1 area for the DRAM
 - 1 area per GDRAM (device memory)
 - 1 area for “unified memory”
- Note that although these spaces overlap, they are still separately identified
- Once a space is constructed, it will be tied to a “system” allocator



Operations

- Operations allow data movement between spaces
- The resource manager will handle all data movement, so the user only needs to provide the source pointer and destination space

```
void* moved = rm.move(ptr, new_space);
```

- Higher-level capabilities around data movement like caching allocations in different locations will be handled by other libraries/applications (e.g. CHAI)

Umpire will co-ordinate with other projects

RAJA

- Lightweight portability layer for loops (“on-node” programming model)
- Gives context for CHAI data copies

CHAI

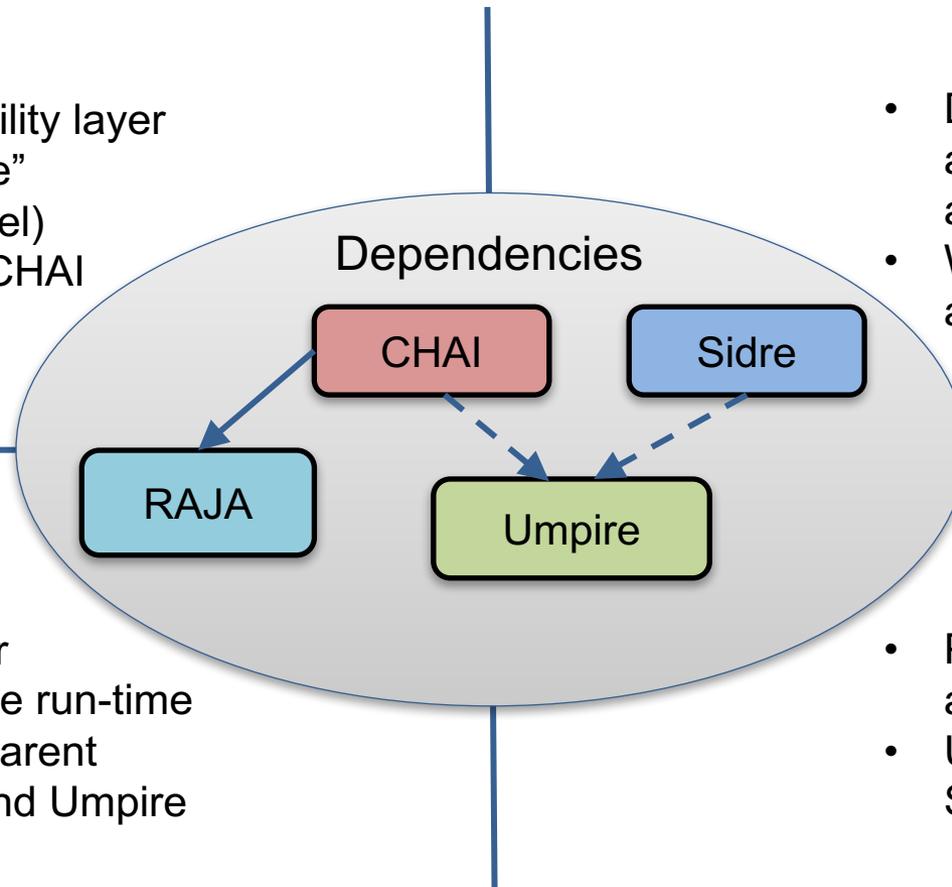
- Lightweight pointer abstraction to make run-time data copies transparent
- Requires RAJA (and Umpire in future)

Sidre

- Data description and access for sharing across apps and tools
- Will require Umpire for allocations (future)

Umpire

- Portable memory allocation and query API
- Underpins CHAI and Sidre (future)

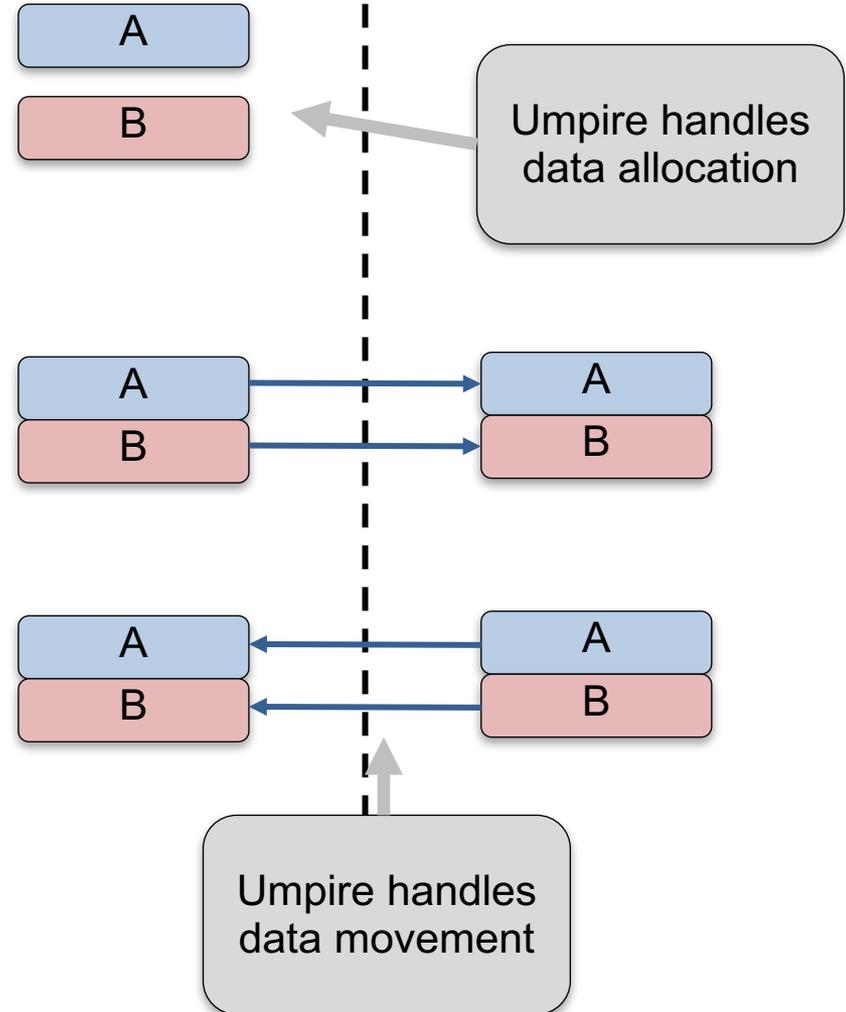


Chai

CPU

GPU

```
chai::ManagedArray<float> a(100);  
chai::ManagedArray<float> b(100);  
  
// init data on host  
  
const float x = 1.0;  
  
forall<cuda_exec>(0, 100, [=] (int i) {  
    a[i] = a[i]*x + b[i];  
});  
  
forall<seq_exec>(0, 100, [=] (int i) {  
    std::cout << "a[i] = " << a[i];  
    std::cout << std::endl;  
});
```



Initial Use Cases

- Flexible pools for temporary allocation of GPU data
 - See Brian Ryujin's talk
- Passing Allocators from application through to library so that library data allocated in the same place
- CHAI in use in multiple LLNL application
 - See Adam Kunen's talk

Current Status

- Initial implementation supporting Allocators for CPU and GPU and simple arena allocation
- Release process underway, will host on GitHub
- We are interested in collaborations at all levels of the memory software stack



**Lawrence Livermore
National Laboratory**

Allocator Concepts

- In Umpire, the Allocator concept is a stateless object that handles allocations at the system level, and is the lowest-level component in the Umpire system.
- The required interface is modeled on that of the C++17 allocator concept
- We currently have wrappers around `std::malloc`, `cudaMalloc`, `cudaMallocManaged` that support this interface

Allocation Introspection

- Umpire will support introspection of allocations and resources, allowing applications and libraries to dynamically adjust their behavior
- Where (what space) is this pointer?
- How much memory is left in this space?