Unified Memory Framework

Unified API for diverse memory technologies

OCP GLOBAL SUMMIT | OCT 15-17, 2024
SAN JOSE, CA

Server: Composable Memory Systems (CMS)

SERVER

Unified Memory Framework

Igor Chorazewicz

OPEN PLATINUM™

Agenda

- Heterogenous memory systems challenges
- Solving the challenges using UMF
- UMF architecture overview
- Status and plans
- Summary and Call to Action

Heterogenous memory systems

- Increased demand for data processing leads to memory subsystems of modern server platforms becoming heterogeneous
- A single application can leverage multiple types of memory:
  - Local DRAM
  - HBM
  - CXL-attached memory
  - GPU memory
- Utilizing heterogenous memory requires:
  - **A way to discover available memory resources**
  - **Deciding where to place the data and how to migrate it between memory types**
  - **Interacting with different APIs for allocation & data migration**

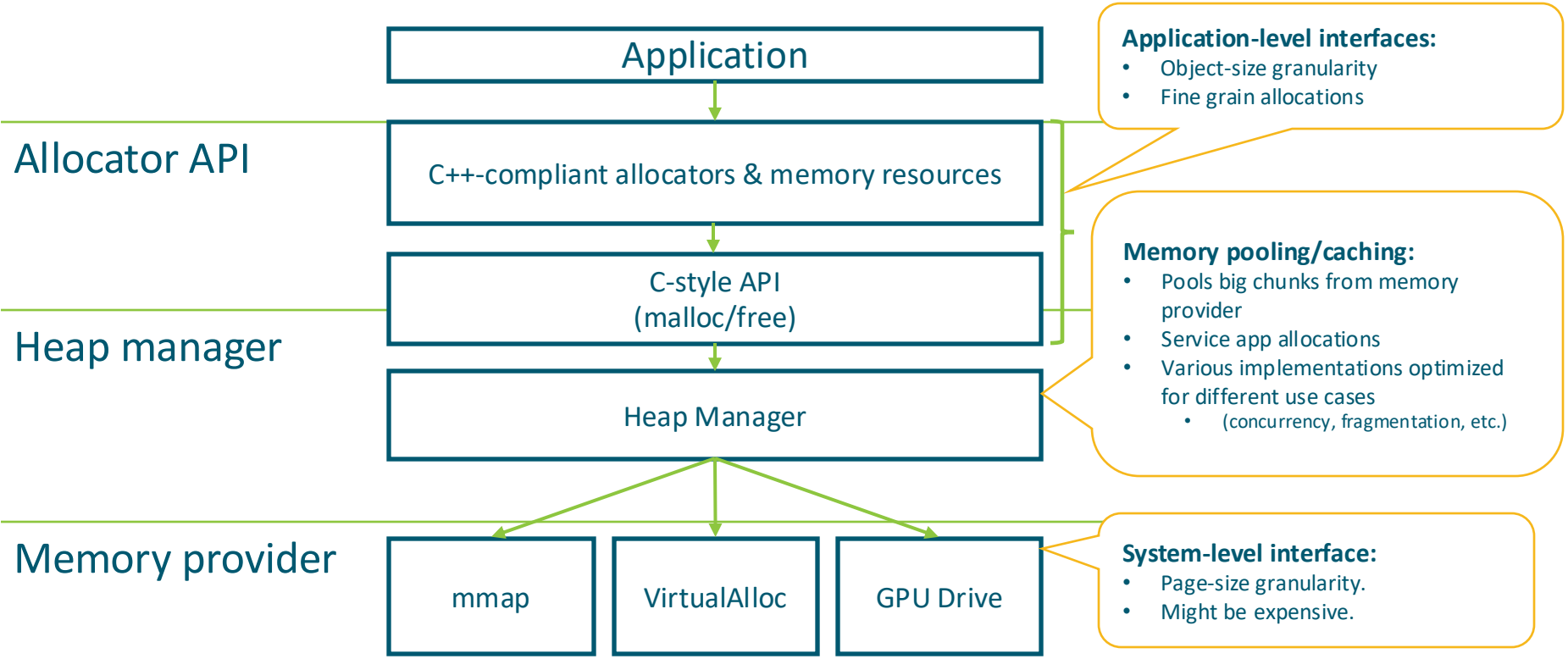
Unified Memory Framework (UMF)

Goal: Unify path for heterogeneous memory allocations and resource discovery among higher-level runtimes (SYCL, OpenMP, Unified Runtime, MPI, oneCCL, etc.) and external libs/applications.

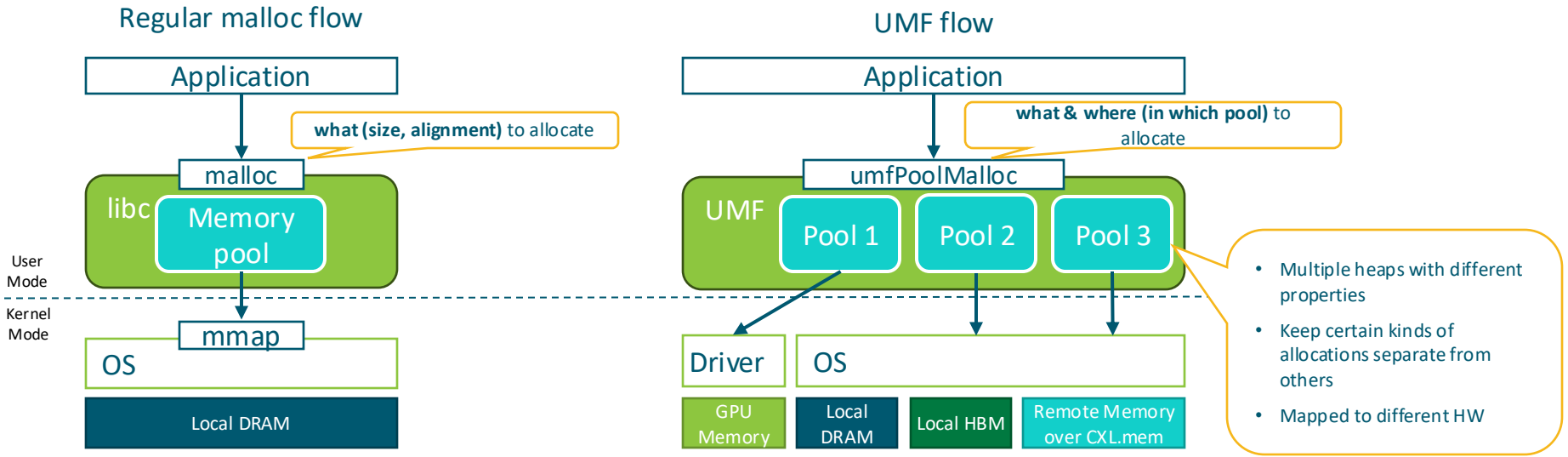
What it is:

- A single project to accumulate technologies related to memory management.
- Flexible mix-and-match API allows tuning for a particular use case.
- Complement (not compete with) OS capabilities.
  - OS - page-size granularity; Applications – object-level abstraction.

Common Memory Allocation Structure



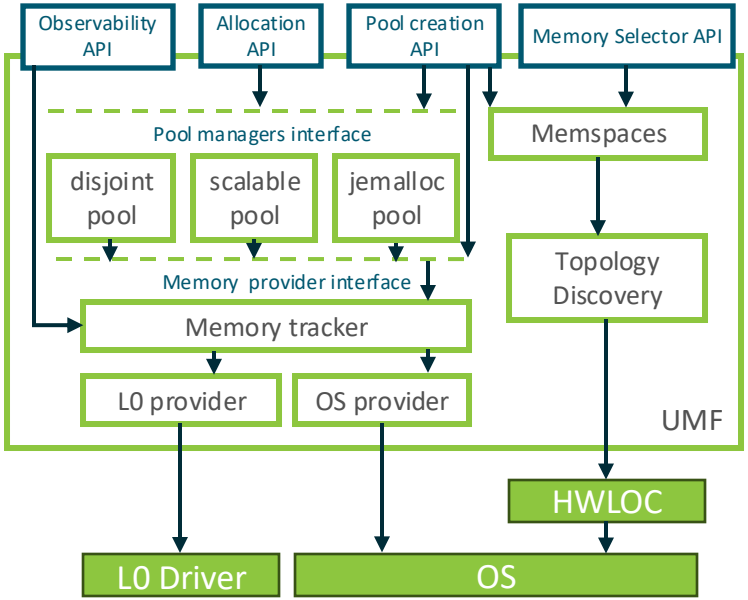
UMF: High-Level Idea



- Expose different kinds of memory as pools/heaps with different properties and behavior. For example:
  - Pool 1 resides on GPU.
  - Pool 2 relies on OS memory tiering - do the same as regular malloc.
  - Pool 3 is bound to DRAM & CXL.mem (allows OS to migrate pages between DRAM and CXL.mem but prohibits migration to HBM). Heap manager can do page monitoring (like Linux DAMON) and make advice to OS (madvise).

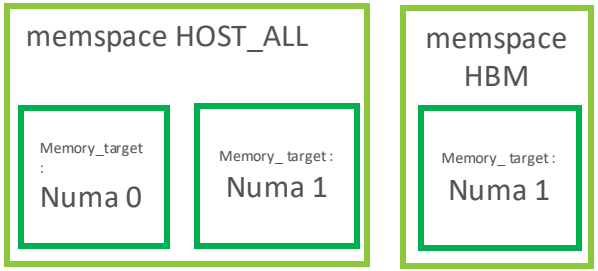
UMF Architecture

- UMF is a framework to build allocators and organize memory pools.
- Pool is a combination of pool manager and memory provider.
  - Memory provider does actual memory (coarse-grain) allocations.
  - Heap manager manages the pool and services fine-grain malloc/free request.
- UMF defines heap manager and memory provider interfaces.
  - Provides implementations (disjoint pool, scalable pool, OS provider) of heap managers and memory providers.
  - Heap managers and Memory provider implementations are static libraries that can be linked on demand.
  - External heap managers and memory providers are allowed.
  - Users can choose existing ones or provide their own.



High-level API: memspaces

- Memspace is an abstraction over memory resources: it's a collection of memory targets.
- Memspace can be used as a means of discovery or for pool creation
- Memory target represents a single memory source (numa node, memory-mapped file, etc.) and can have certain properties (e.g. latency, bandwidth, capacity)
- UMF exposes predefined memspaces (HOST_ALL, HBM, LOWEST_LATENCY, etc.)



Basic Example

Pool creation flow

```
// Create memory pool of HBM memory from predefined memspace
umf_memory_pool_handle_t hbmPool = NULL;
umf_memspace_handle_t MEMSPACE_HBW = umfMemspaceHighestBandwidthGet();
umfPoolCreateFromMemspace(MEMSPACE_HBW, NULL, &hbmPool);

// Create memory pool on top of the highest capacity memory
umf_memory_pool_handle_t highCapPool = NULL;
umf_memspace_handle_t MEMSPACE_HIGH_CAP = umfMemspaceHighestCapacityGet();
umfPoolCreateFromMemspace(MEMSPACE_HIGH_CAP, NULL, &highCapPool);
```

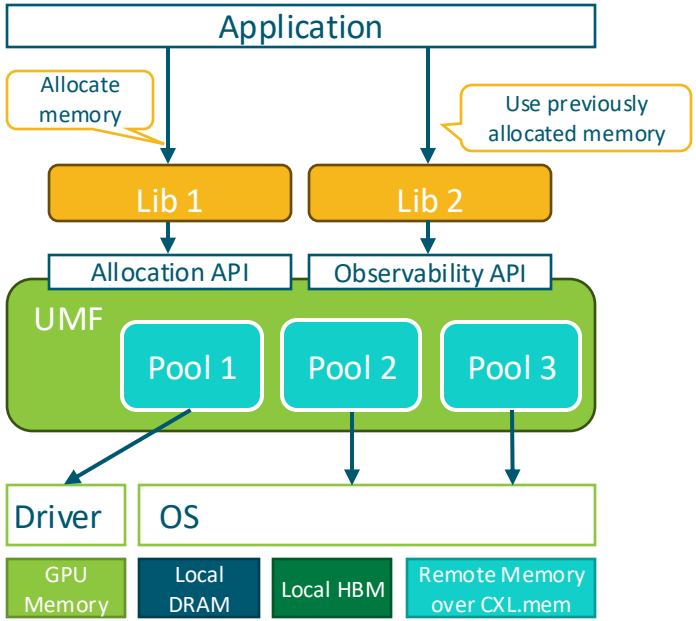
malloc/free flow

```
// Allocate HBM memory from the pool
void* ptr1 = umfPoolMalloc(hbmPool, 1024);

// Allocate memory from the highest capacity pool
void* ptr2 = umfPoolMalloc(highCapPool, 1024);

umfFree(ptr1); // Pool is found automatically
umfFree(ptr2); // Pool is found automatically
```

UMF: Interop capabilities

Memory is a key for efficient interoperability

- Modern applications are complex.
  - Multiple libraries/runtimes might be used by a single application.
  - Memory allocated by one library might be used by another library.
- UMF aggregates data about allocations.
  - Can provide memory properties of allocated regions.
- **Example:** Memory allocated by OpenMP/SYCL is used by MPI for scale-out. UMF can tell:
  - Whether it is OS-managed or GPU driver-managed memory.
  - Which NUMA node is used.
  - MPI can get IPC handle to map memory to another process.



Current Status and Plans for 2024

- First release as internal component of oneAPI 2025.0
- Open-source repo is created for open development.
- Key stakeholders:
  - **Unified Runtime:** USM memory pooling (used by SYCL and OpenMP offload).
  - **Intel MPI:** interop with SYCL and OpenMP based on Observability & IPC API.
  - **oneCCL:** memory pooling for big allocations and IPC functionality.
  - **libiomp:** build OpenMP 6.0 support on top of UMF.
  - **CAL:** malloc/free intercept based on UMF

Summary and Call to Action

- UMF unifies interfaces to work with memory hierarchies.
- UMF improves efficiency by code/technology reuse.
  - Set of building blocks to adapt to particular needs.
- UMF handles interop between runtimes by aggregating data about all allocations.
- Try out UMF when dealing with heterogenous memory or building a custom memory allocator
- Where to find additional information
  - https://oneapi-src.github.io/unified-memory-framework/introduction.html
  - https://github.com/oneapi-src/unified-memory-framework

Thank you!