CXL.Mem and UMF Programming Workshop

Virtual Workshop

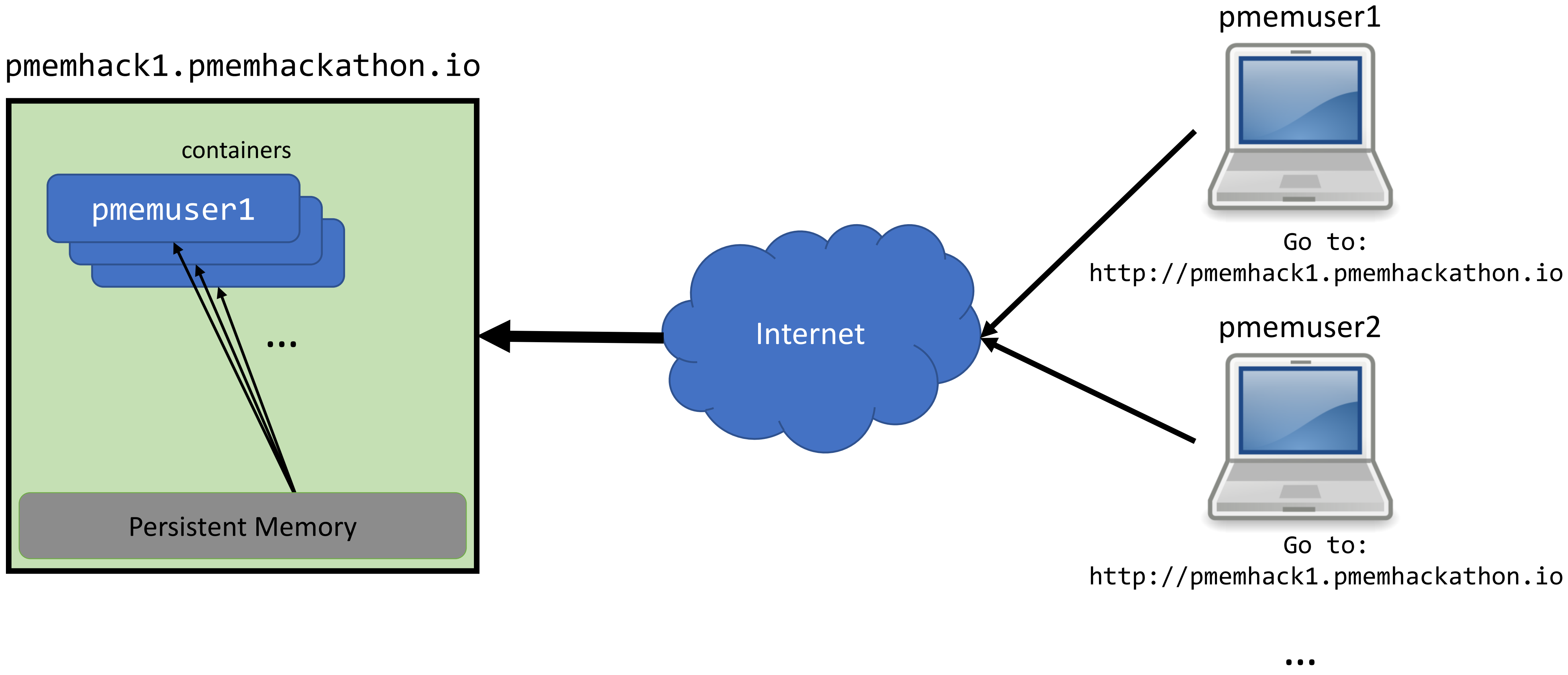
https://github.com/pmemhackathon/hackathon

SNIA

Agenda

- Essential Background Slides, covering:
  - Logistics: how you access persistent memory from your laptop
  - The minimum you need to know about CXL
  - UMF overview

Logistics: The *webhackathon* Tool



Webhackathon Basics

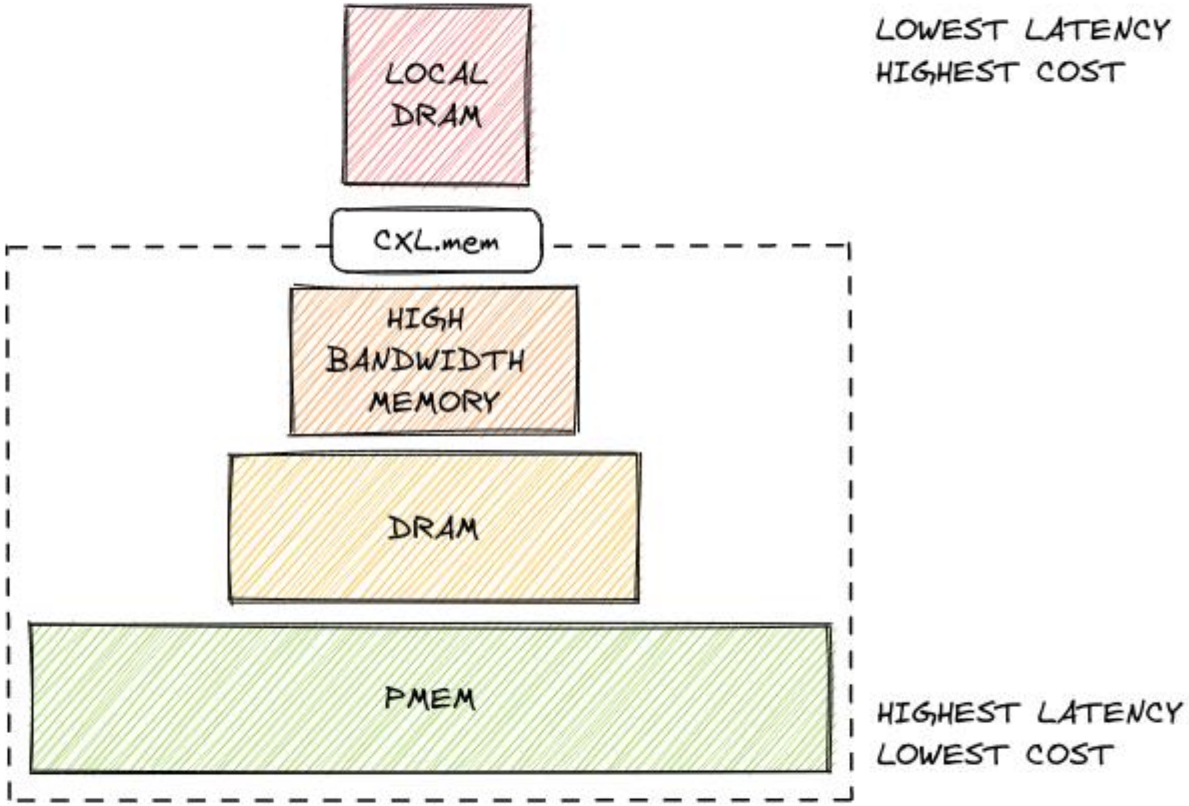
- List of examples presented on main page
  - First four recommended to provide essential background
    - We will walk through some of these together
  - Pick examples that are interesting to you (task, language, etc)
  - Use them as a starting point for your own code
- Menu provides:
  - Access to these background slides
  - Browse your copy of the repo (to download something you want to keep)
  - Browser-based shell window for your container (for users who need it)
- Everything you do runs in your own container on the server
  - With your own copy of the hackathon repo
- We're all friends here: please no denial-of-service attacks on server!

CXL.mem

CXL

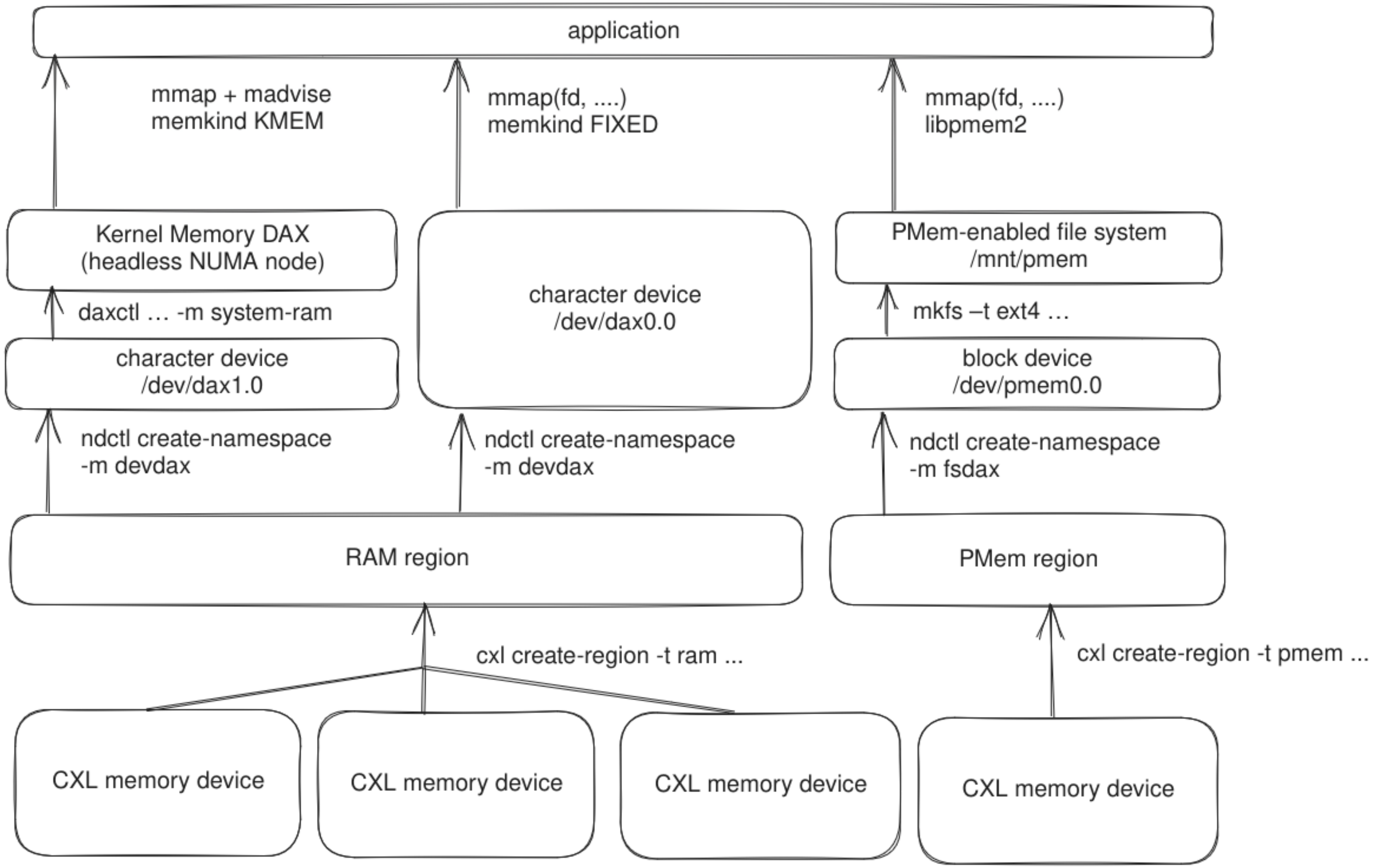
- Interconnect standard built on top of PCIe
- Facilitates cache-coherent memory access between CPUs and supporting PCIe-attached devices (pure memory devices but also accelerators) – CXL.cache and CXL.mem
- Supports memory pooling and sharing
- Memory connected through CXL can be exposed similarly as Pmem

Heterogenous memory hierarchy



System topology with CXL

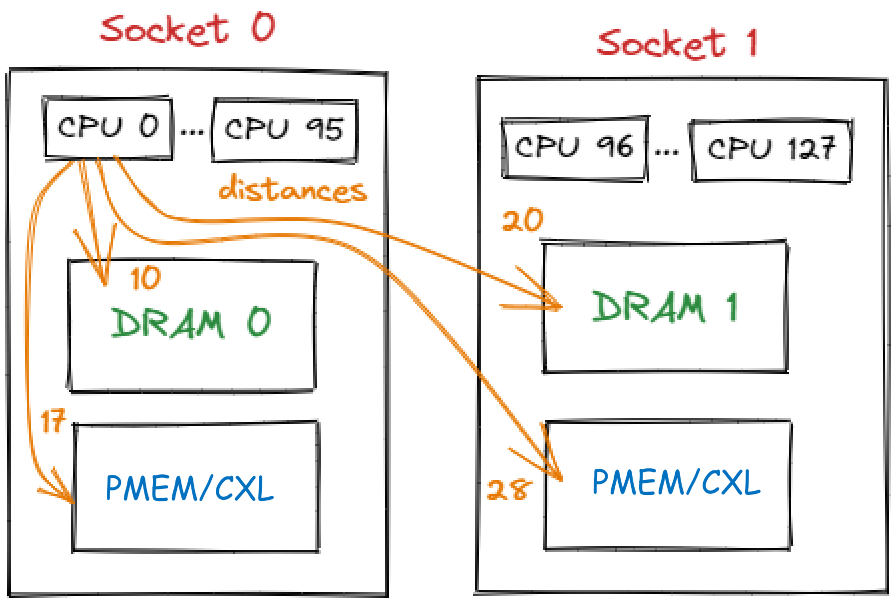
- CXL Type 3 (memory) devices can provide both volatile or persistent capacity
- Transitioning from PMem to CXL is straightforward for most use-cases



CXL Software ecosystem

CXL Memory Configuration	Administrative steps	Use cases	Programming model (same as PMem)
Default Global volatile memory (system ram as NUMA)	None.	Adding more volatile memory capacity, potentially with software tiering.	Unmodified apps: Traditional memory management, OS-managed NUMA locality. Modified apps: Speciality NUMA allocators (e.g., `libnuma`, `memkind`). All apps: Direct use of `mmap`/`mbind`.
Volatile devdax	Reconfiguring namespace to devdax.	Adding new isolated memory capacity, manual tiering.	Speciality allocators capable of operating on raw memory ranges (e.g., `memkind`), manual use of `mmap`.
Volatile use of fsdax	Configuring pmem region and fsdax namespace.	Named volatile regions of volatile memory using file system to control access.	Speciality allocators capable of managing pools on top of file systems (e.g., `memkind`). **Note** For new software, a better alternative may be using tmpfs bound to a system-ram NUMA node. It's likely to be faster and less error-prone.
Persistent fsdax	Configuring pmem region and fsdax namespace.	Existing PMem-aware or storage-based software that uses regular files.	SNIA Persistent Memory Programming Model. Unmodified apps just work. New ones can still use PMDK.
Persistent devdax	Configuring pmem region and devdax namespace.	Custom software requiring full control of memory.	Raw access through `mmap`, can flush using CPU instructions. Apps can use PMDK.

NUMA nodes



Socket 0
Socket 1
CPU 0
...
CPU 95
CPU 96
...
CPU 127
distances
20
10
DRAM 0
DRAM 1
17
PMEM/CXL
28
PMEM/CXL

Login to server…

`http://pmemhack1.pmemhackathon.io`

Click [Request Access] to get a login

`Workshop ID:cxl`

Unified Memory Framework

Authors: Sergei Vinogradov, Igor Chorazewicz, Piotr Balcer, Rafal Rudnicki

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

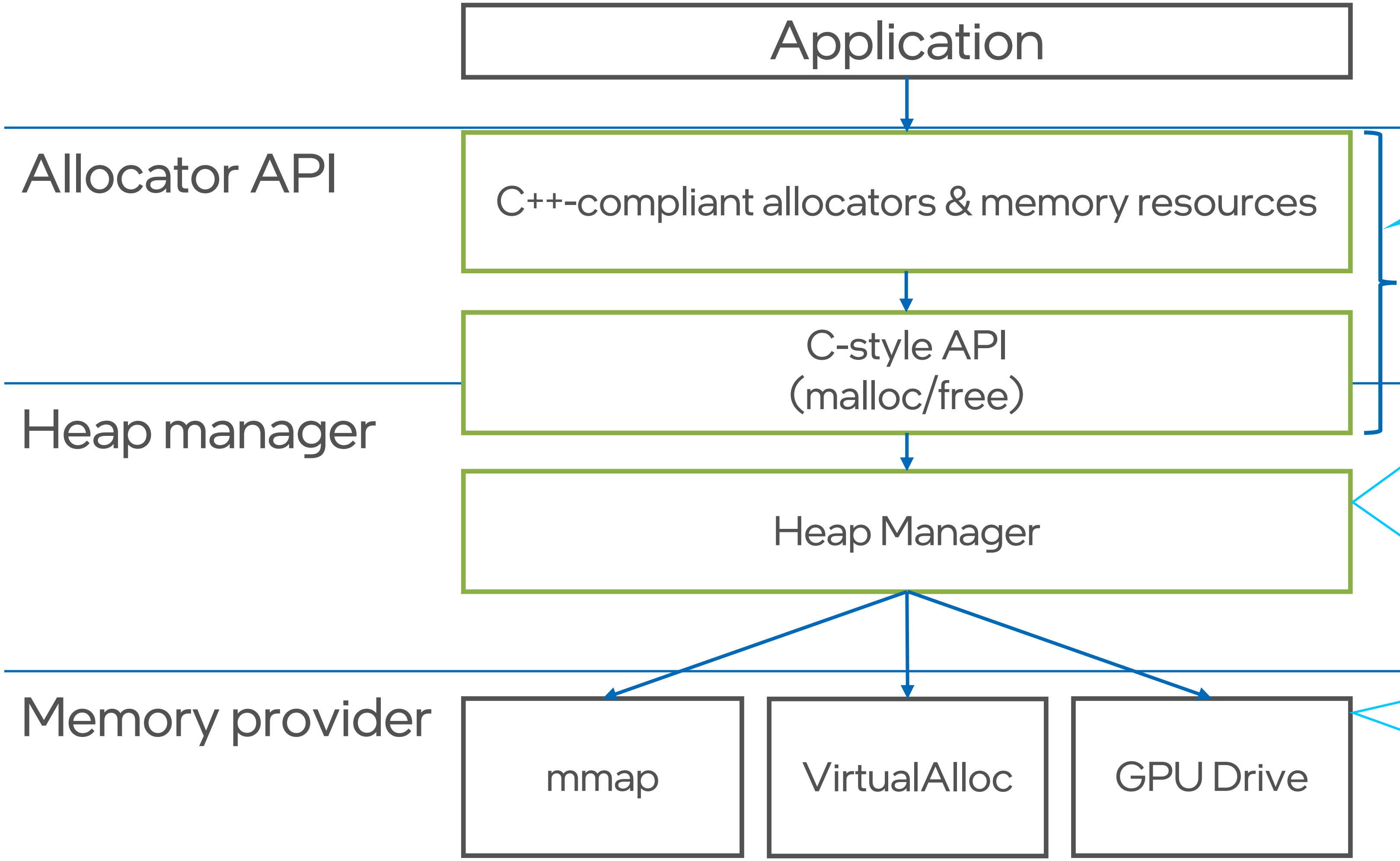
Unified Memory Framework (UMF)

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

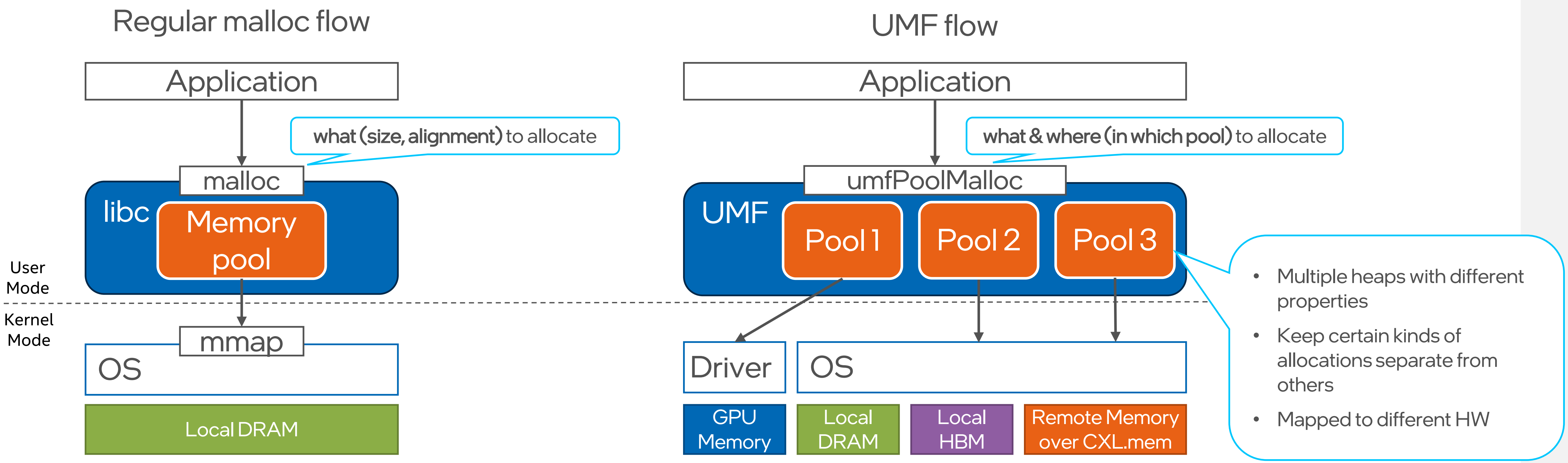


Application-level interfaces:
- Object-size granularity
- Fine grain allocations

Memory pooling/caching:
- Pools big chunks from memory provider
- Service app allocations
- Various implementations optimized for different use cases
  - (concurrency, fragmentation, etc.)

System-level interface:
- Page-size granularity.
- Might be expensive.

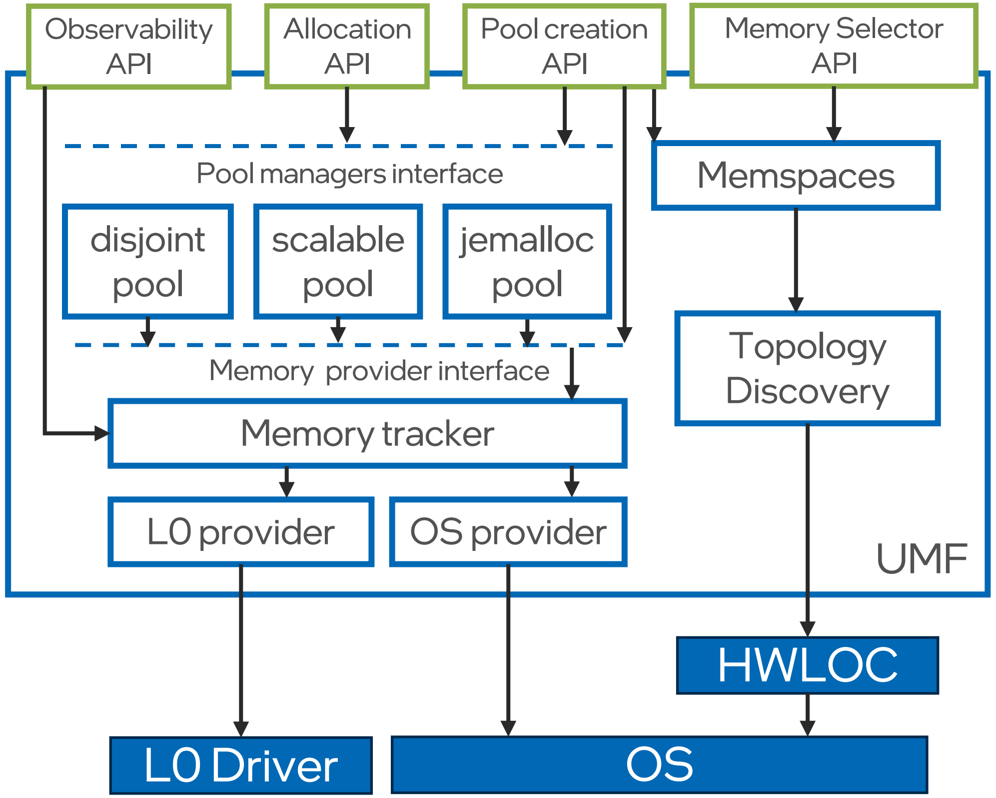
UMF: High-Level Idea



- 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).

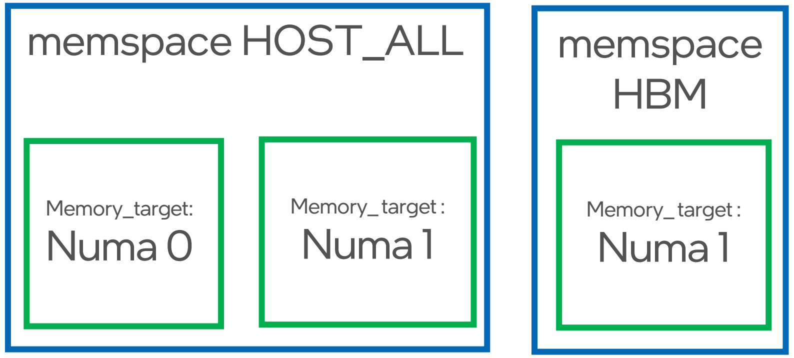
UMF Architecture

- 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);
```

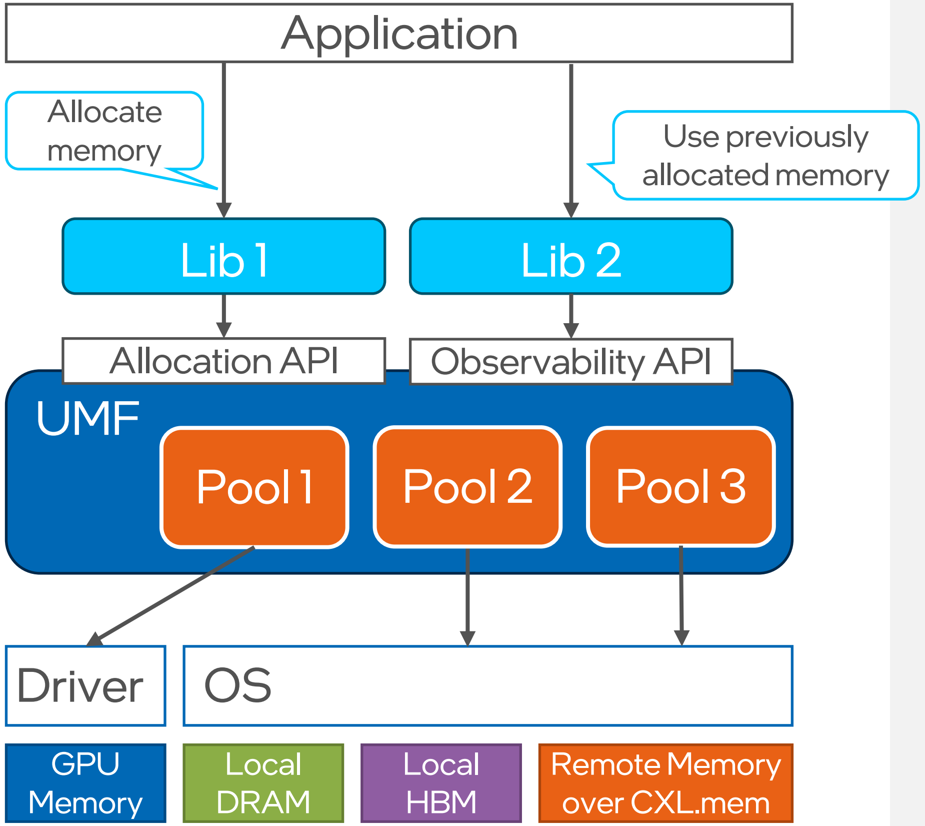
malloc/free flow

```
// 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 in 2024Q3.
- 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

- 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.

- **Call to action:**
  - Try out UMF when dealing with heterogenous memory or building a custom memory allocator
- **Extra resources:**
  - https://oneapi-src.github.io/unified-memory-framework/introduction.html