

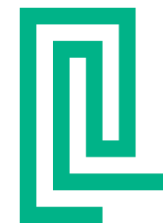


Hewlett Packard
Enterprise

The OpenFAM API: a programming model for disaggregated persistent memory

Kimberly Keeton, Sharad Singhal, Michael Raymond
 {kimberly.keeton,sharad.singhal,mraymond}@hpe.com

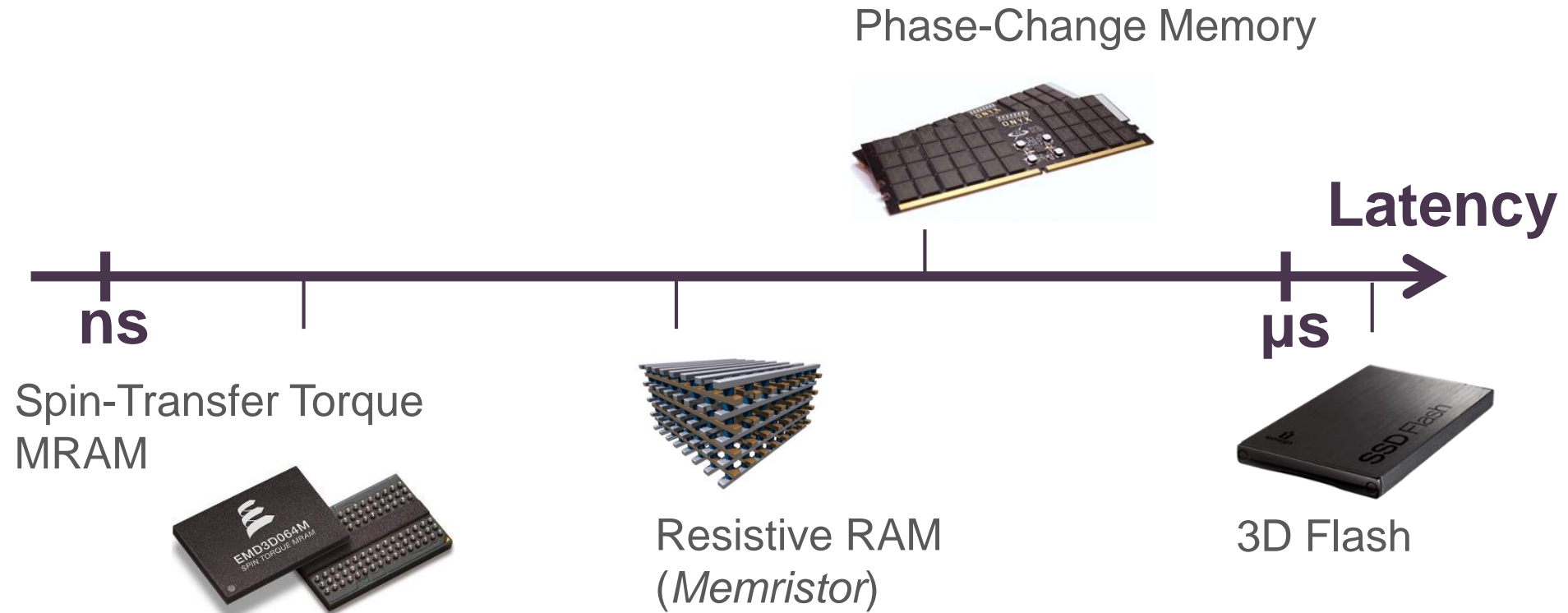
OpenSHMEM 2018



Outline

- Motivation: persistent disaggregated memory
- OpenFAM: programming model for fabric-attached memory
 - Overview and concepts
 - API methods
 - Topics for discussion
- Summary

Non-Volatile Memory (NVM)



- Persistently stores data
- Access latencies comparable to DRAM
- Byte addressable (load/store) rather than block addressable (read/write)
- More energy efficient and denser than DRAM

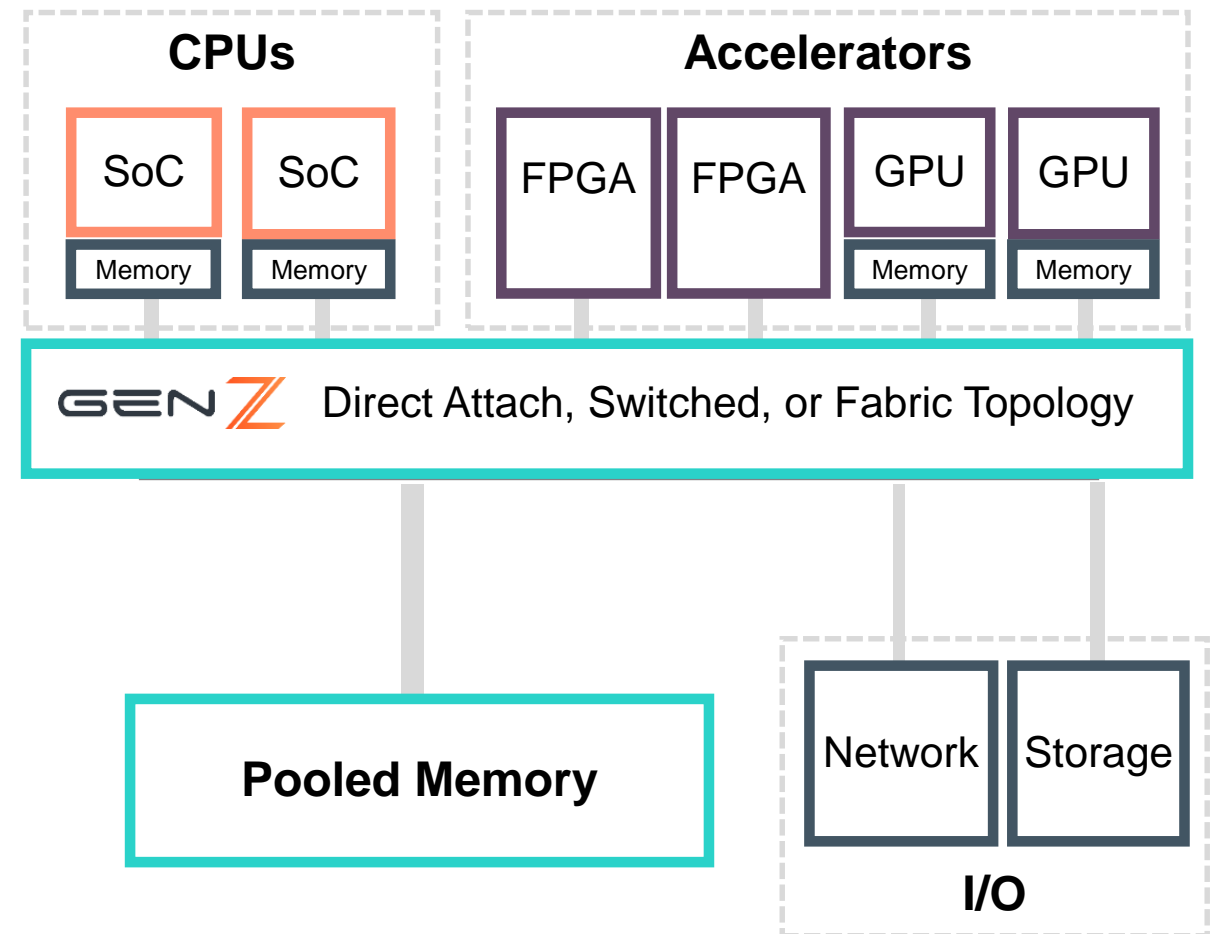
Haris Volos, et al. "Aerie: Flexible File-System Interfaces to Storage-Class Memory," *Proc. EuroSys 2014*.

Gen-Z: open systems interconnect standard

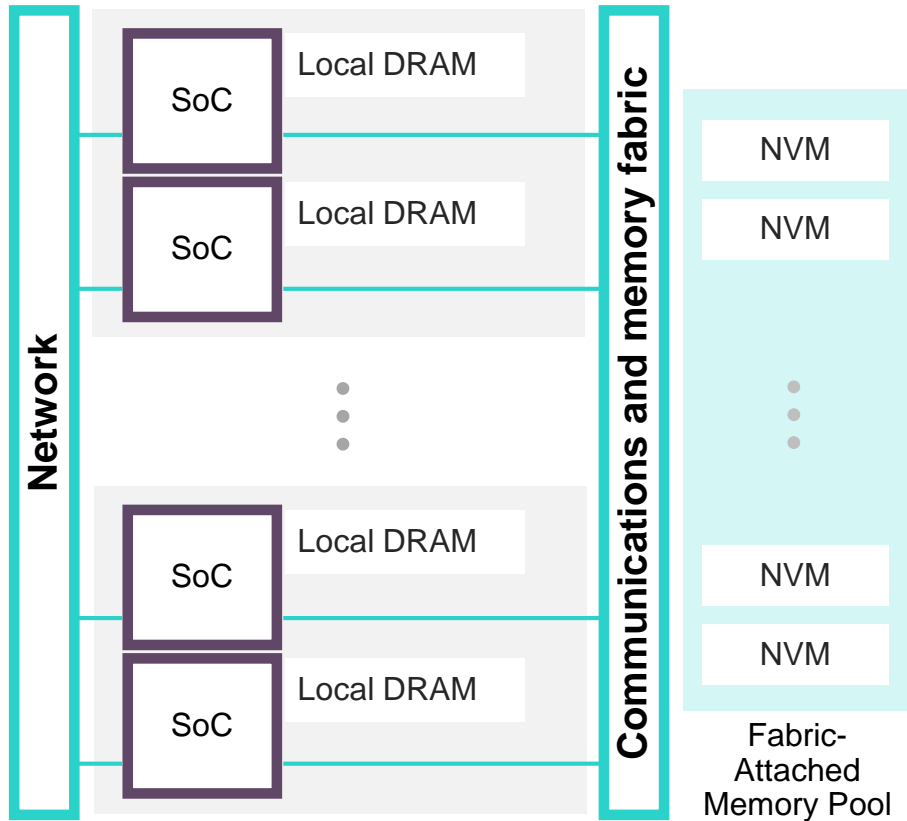


<http://www.genzconsortium.org>

- Open standard for memory-semantic interconnect
- Members: 50+ companies covering SoC, memory, I/O, networking, mechanical, system software, etc.
- Motivation
 - Emergence of low-latency storage class memory
 - Demand for large capacity, rack-scale resource pools and multi-node architectures
- Memory semantics
 - All communication as memory operations (load/store, put/get, atomics)
- High performance
 - Tens to hundreds GB/s bandwidth
 - Sub-microsecond load-to-use memory latency
- *Spec available for public download*



Fabric-attached (persistent) memory



- **Converging memory and storage**
 - Byte-addressable NVM replaces hard drives and SSDs
- **Resource disaggregation leads to high capacity shared memory pool**
 - Fabric-attached memory pool is accessible by all compute resources
 - Low diameter networks provide near-uniform low latency
- **Local volatile memory provides lower latency, high performance tier**
- **Distributed heterogeneous compute resources**
- **Software**
 - Memory-speed persistence
 - Direct, unmediated access to all fabric-attached NVM across the memory fabric
 - Non-coherent accesses between compute nodes
 - Operating system instance per compute node
- **Enables Memory-Driven Computing**

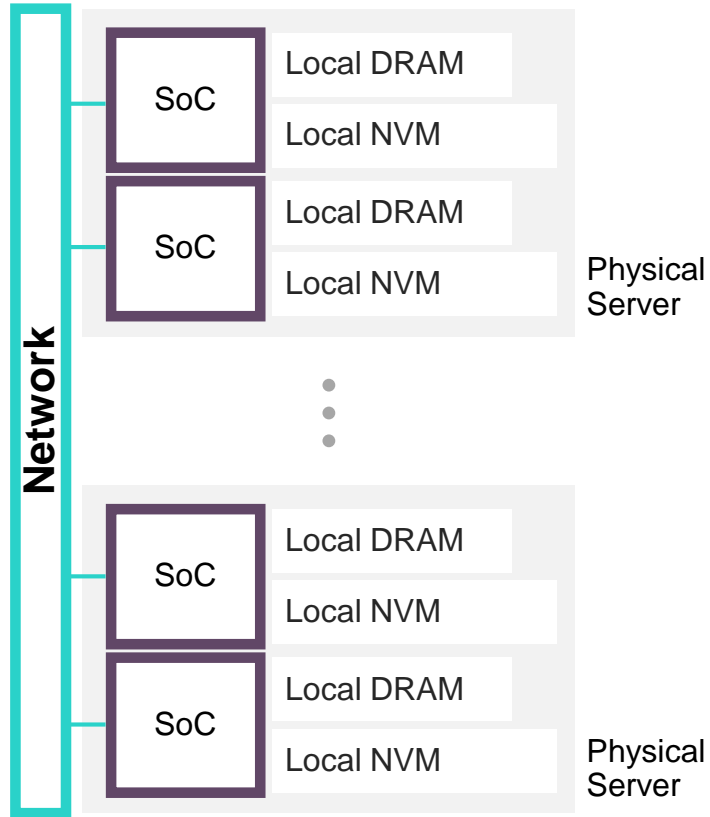
HPE introduces the world's largest single-memory computer

The prototype contains 160 terabytes of fabric-attached memory

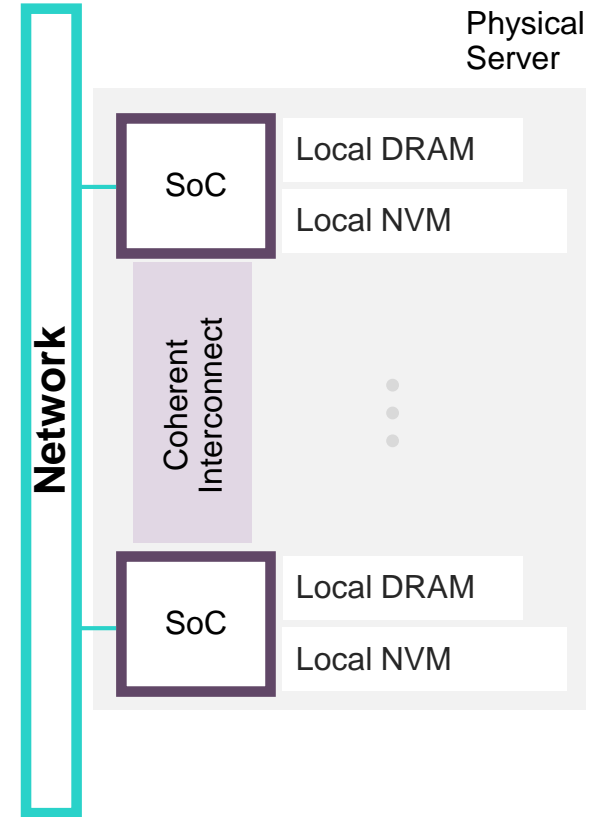
- 160 TB of fabric-attached, shared memory spread across 40 physical nodes, interconnected using a high-performance fabric protocol
- An optimized Linux-based operating system running on ThunderX2, Cavium's flagship second generation dual socket capable ARMv8, a workload optimized System on a Chip
- Photonics/Optical communication links, including the new X1 photonics module
- Software programming tools designed to take advantage of abundant of persistent memory



Memory-Driven Computing in context

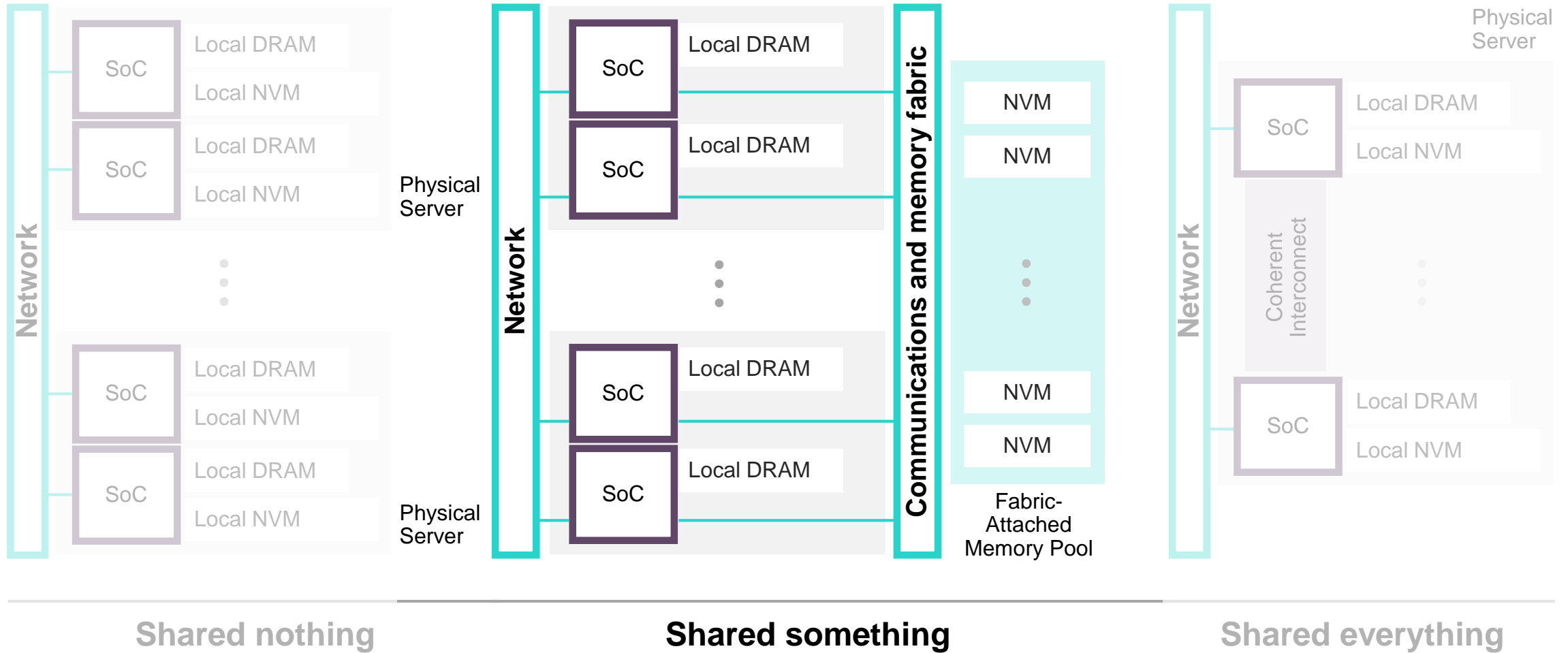


Shared nothing

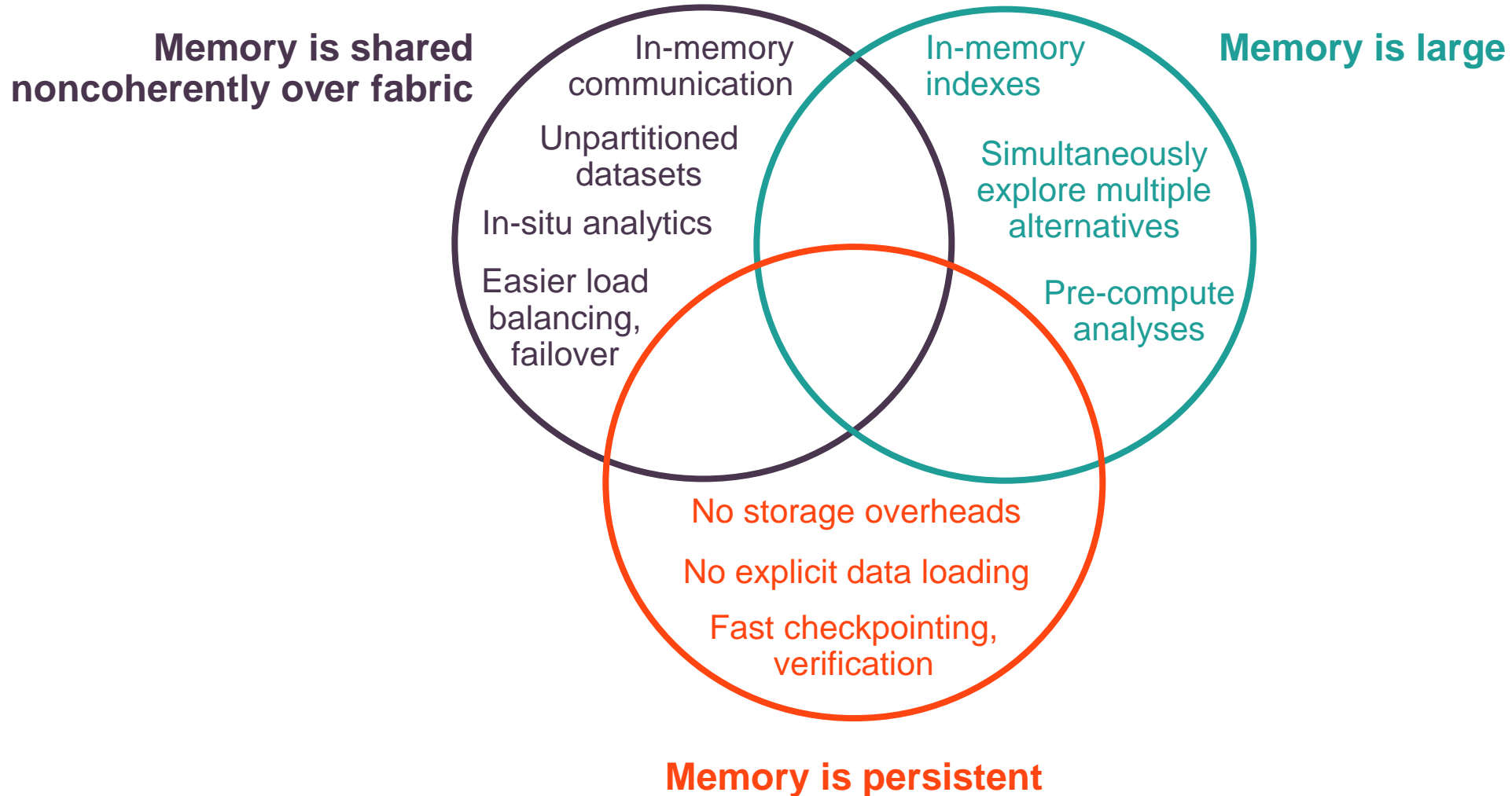


Shared everything

Memory-Driven Computing in context



Memory-Driven Computing benefits applications

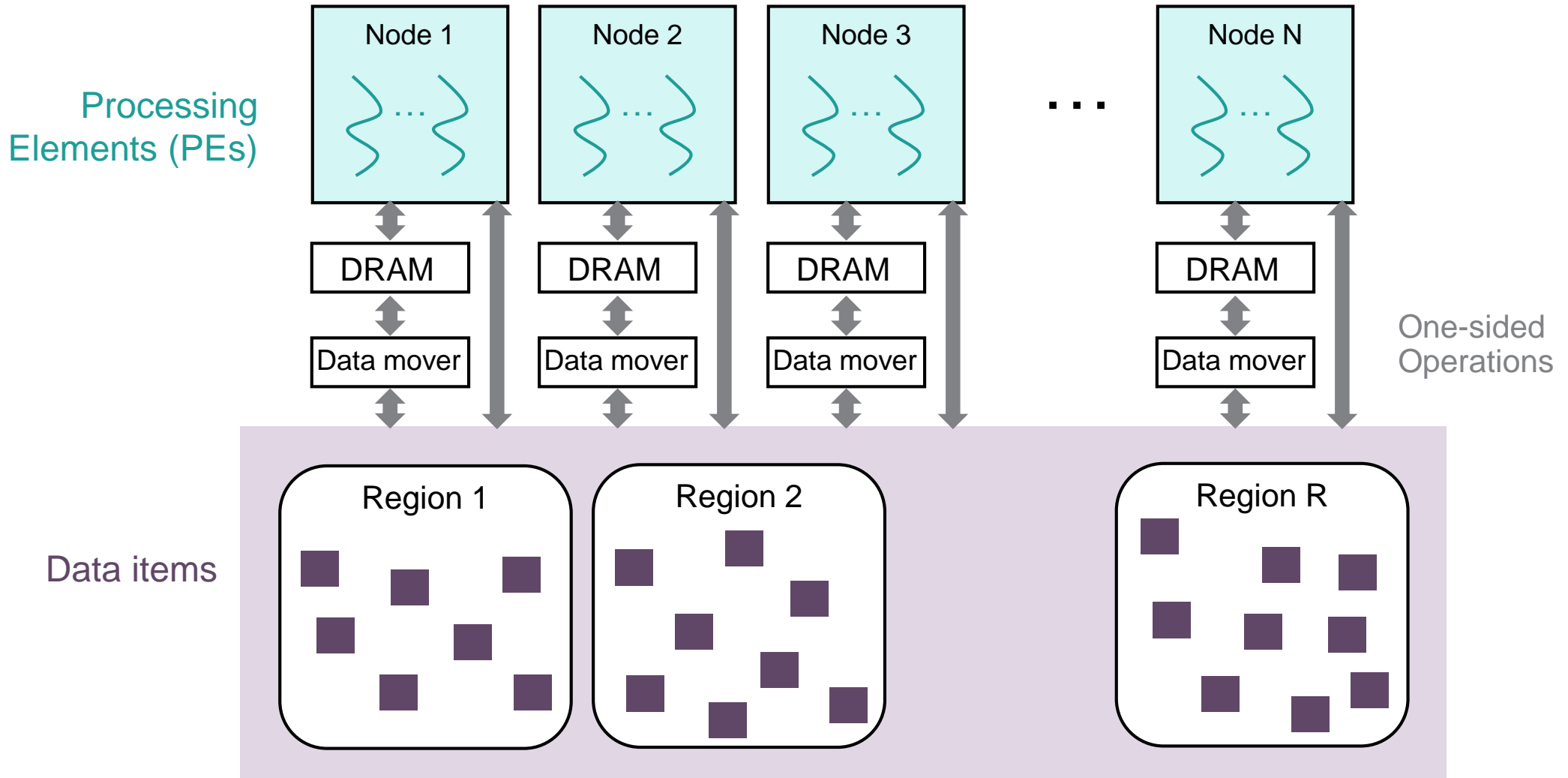


OpenFAM: programming model for fabric-attached memory

- Inspired by
 - OpenSHMEM (<http://openshmem.org>): open source partitioned global address space (PGAS) library with one-sided communication, atomic and collective operations
 - HPE's work on The Machine program (<https://www.labs.hpe.com/the-machine>)
- Differences between OpenFAM and OpenSHMEM environments:
 - Focus is on fabric-attached memory (FAM) instead of remote processing element (PE) memory
 - We assume that node memory is treated as private and that FAM is shared
 - FAM may be persistent, so data can live beyond a single program invocation
 - FAM may be larger than what a processor's physical address bits can address
 - FAM-resident data may be shared across nodes with independent operating system instances
 - Memory semantic fabrics permit load/store access to FAM-resident data

OpenFAM concepts

Compute Nodes + Locally-Attached Memories (LAMs)



Regions vs. data items

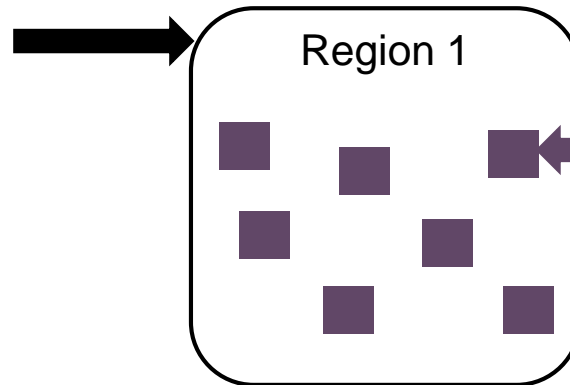
- Regions define sections of FAM with specific characteristics (e.g., persistence, redundancy)
- Useful to permit multiple regions associated with a given job to accommodate different data needs. Ex:
 - No redundancy for communication or scratch space
 - Redundancy for computation results
- Named regions of FAM enable sharing between PEs of a given job and also between jobs of a workflow (for persistent data)
- Region forms basis for heap allocator in memory management routines
 - Data items are allocated using heap allocator

Descriptors

- Descriptors are opaque read-only data structures that uniquely identify a location in FAM
- Descriptors are portable across OS instances
 - Use base + offset addressing
 - Can be freely copied and shared across processing nodes by the program

Fam_RegionDescriptor

- includes redundancy level

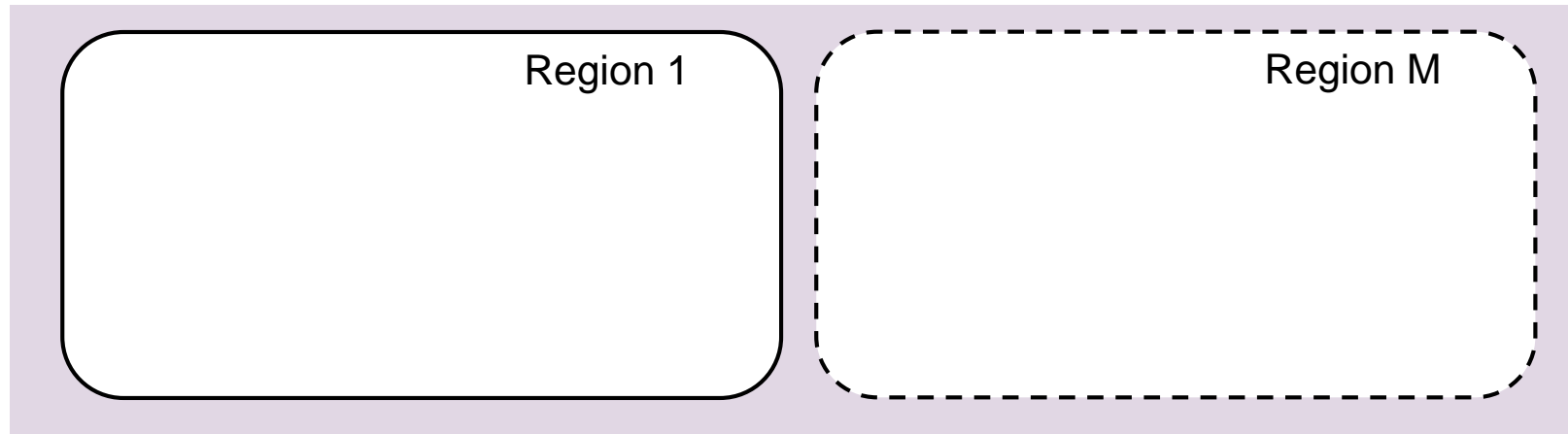


Fam_Descriptor

- includes region ID, offset

Allocation (region management)

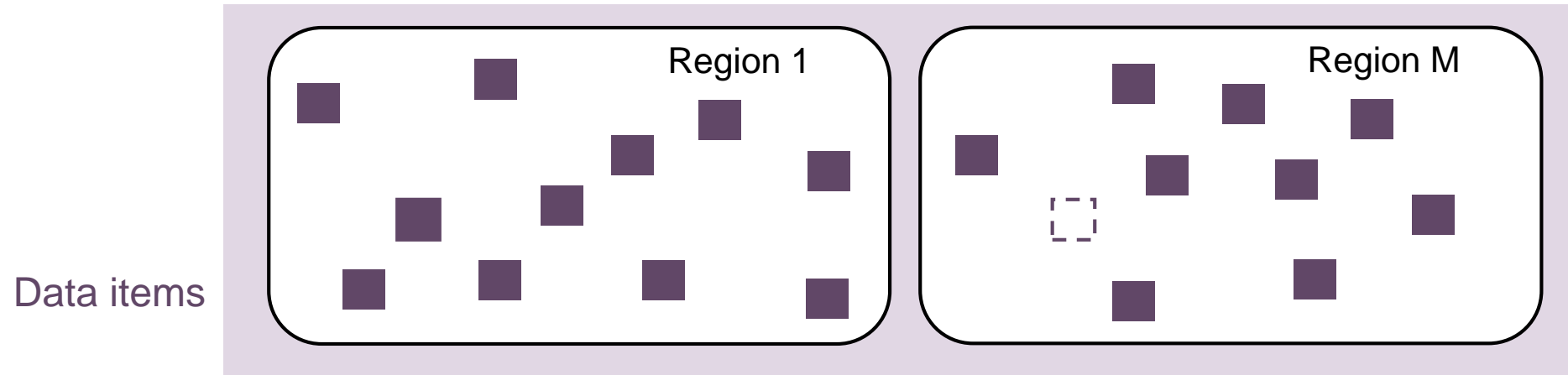
- **fam_create_region**: allocates space for region, with associated options (size, redundancy, persistence, permissions, name)
- **fam_destroy_region**: used to indicate that user is done with allocated region
 - Triggers delayed reclamation to accommodate ongoing users
- **fam_resize_region**: change size of a region allocation



Global Shared Non-volatile Memory (aka Fabric-Attached Memory (FAM))

Allocation (data item / heap management)

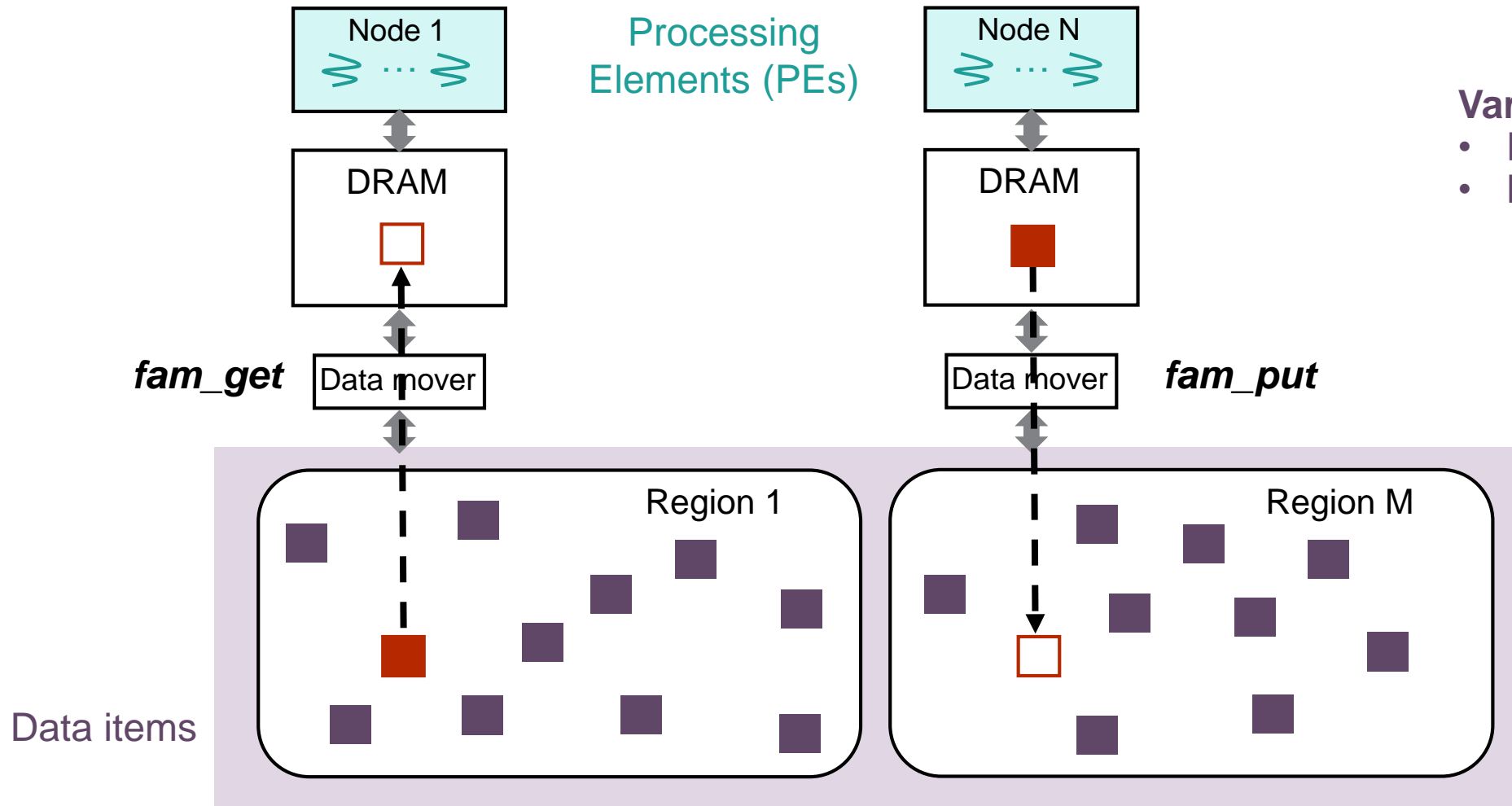
- **fam_allocate**: allocates space for data item within a region, with associated options (size, permissions, name)
- **fam_deallocate**: used to indicate that user is done with allocated data item
 - Triggers delayed reclamation to accommodate ongoing users
- **fam_change_permissions**: change permissions of a data item or region in FAM



Global Shared Non-volatile Memory (aka Fabric-Attached Memory (FAM))

Data path (get / put)

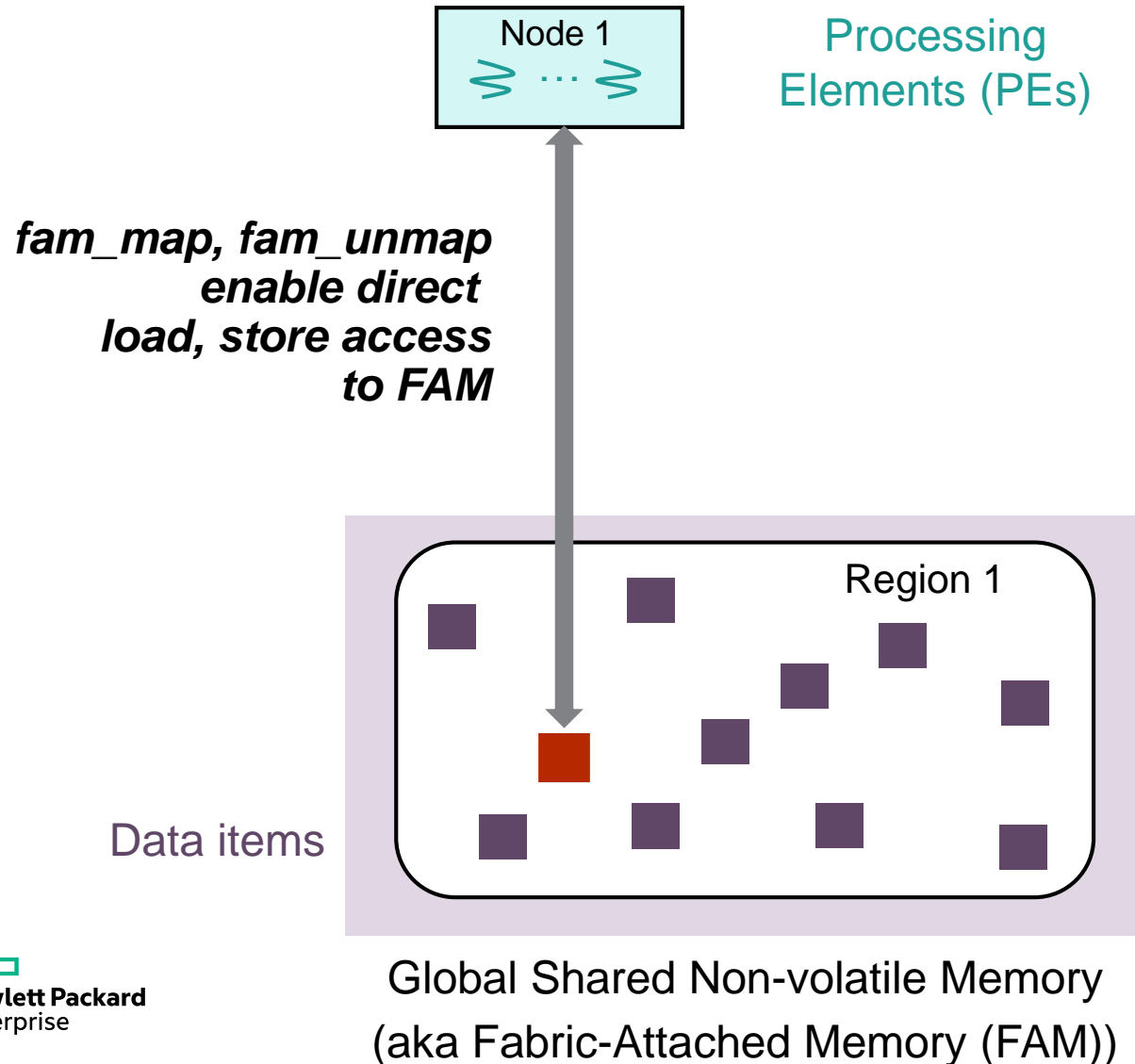
Compute Nodes + Locally-Attached Memories (LAMs)



- Variants:**
- **Blocking**
 - **Non-blocking**

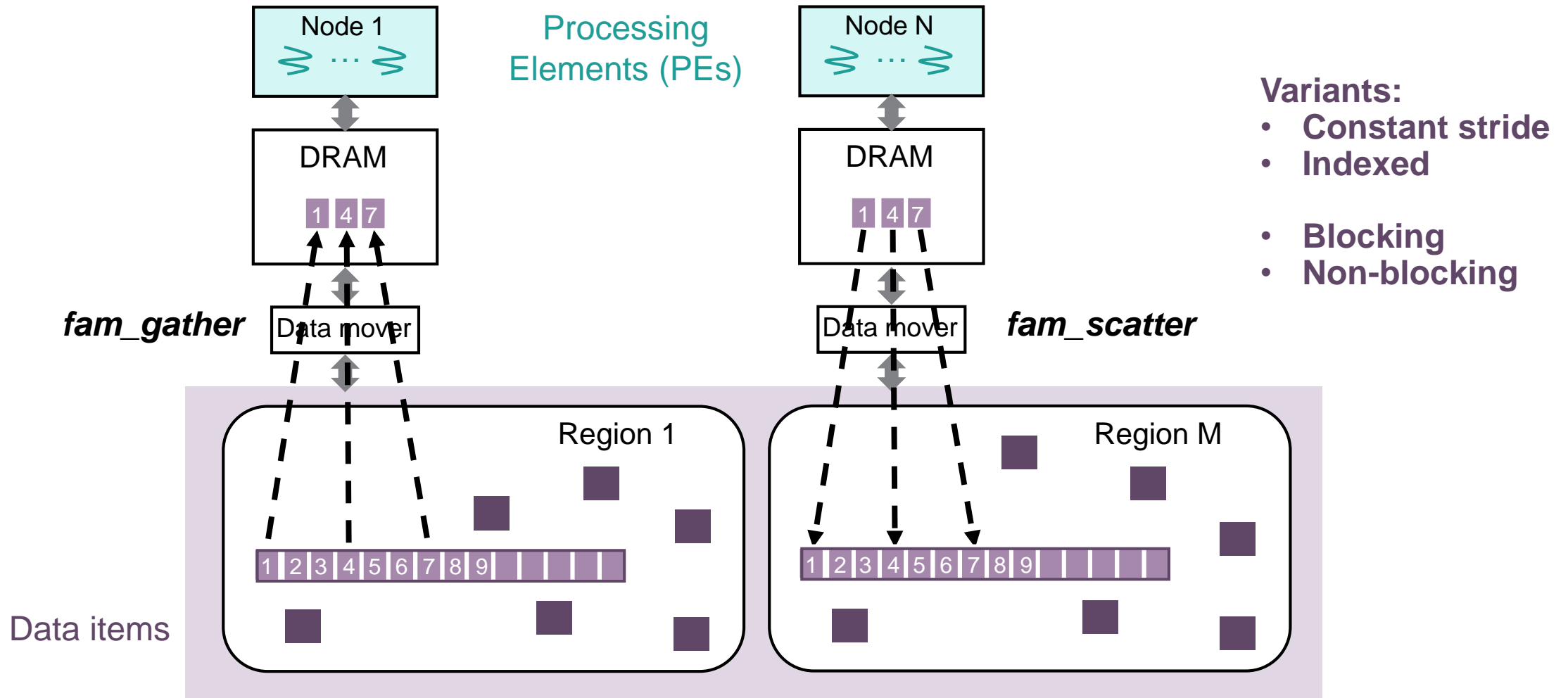
Data path (direct access)

Compute Nodes + Locally-Attached Memories (LAMs)



Data path (gather / scatter)

Compute Nodes + Locally-Attached Memories (LAMs)

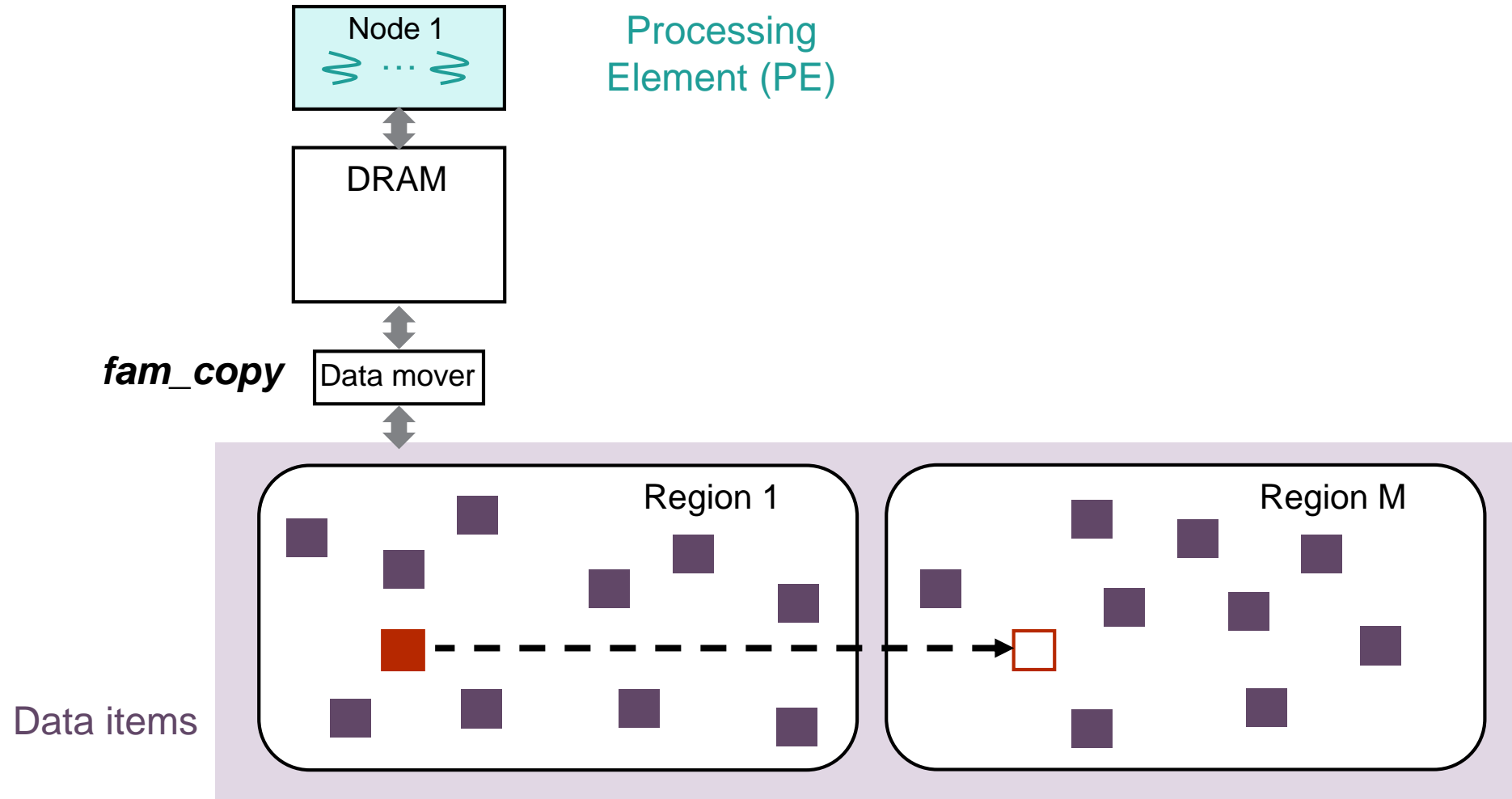


- Variants:**
- Constant stride
 - Indexed
 - Blocking
 - Non-blocking

Global Shared Non-volatile Memory (aka Fabric-Attached Memory (FAM))

Data management (copy)

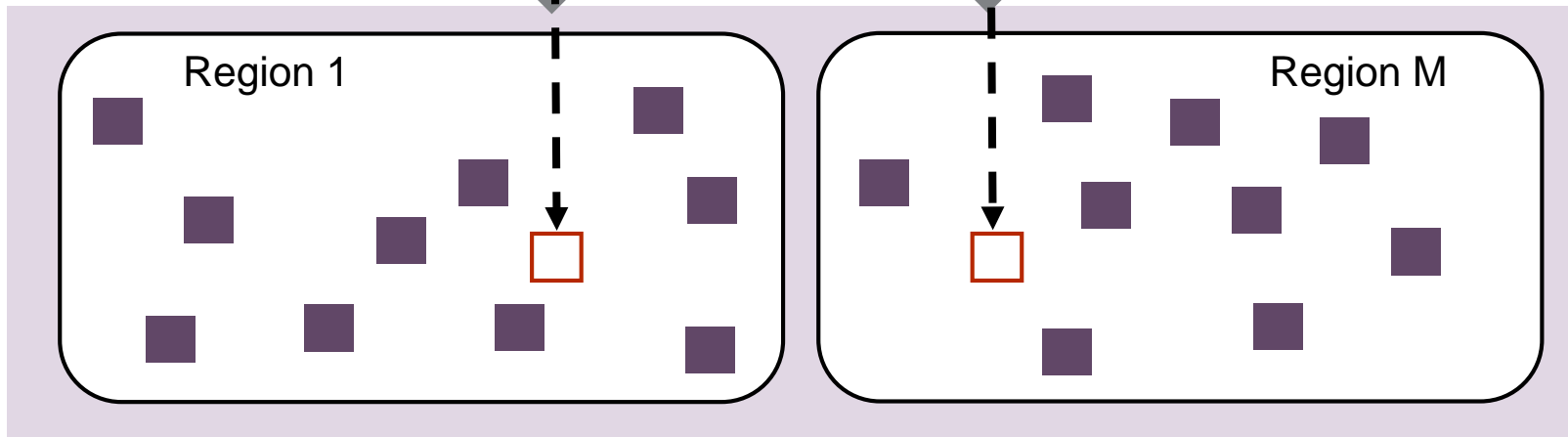
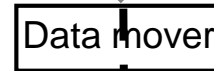
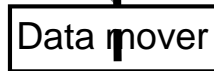
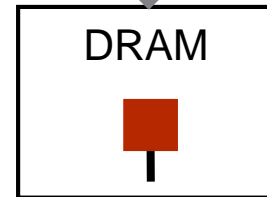
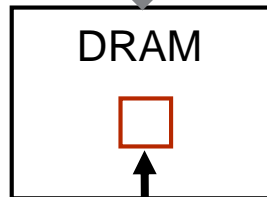
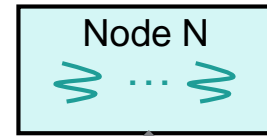
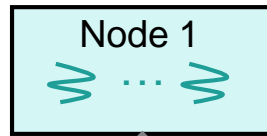
Compute Node + Locally-Attached Memory (LAM)



Atomics

Compute Nodes + Locally-Attached Memories (LAMs)

Processing Elements (PEs)



Data items

Global Shared Non-volatile Memory (aka Fabric-Attached Memory (FAM))

OpenFAM fetching routines:

Fetch, swap, add, subtract, min, max: signed and unsigned 32b and 64b integer, float, double

Compare-and-swap: signed and unsigned 32b and 64b integer, 128b integer

And, or, xor: 32b and 64b unsigned integer

OpenFAM non-fetching routines:

Set, add, subtract, min, max: signed and unsigned 32b and 64b integer, float, double

And, or, xor: 32b and 64b unsigned integer

Memory ordering (fence)

Pre-fence
fabric memory
operations



Post-fence
fabric memory
operations

Memory fence

- **fam_fence** ensures that FAM operations issued by the calling thread before the fence are completed before FAM operations issued after the fence are dispatched (non-blocking)
- **fam_quiet** waits for completion of all outstanding requests by the calling thread to global fabric-attached memory (puts, atomics, stores) (blocking)

Alternatives for handling OpenFAM call failures

Failure-reporting (app control)

- OpenFAM failures report their errors
 - Application can determine how to handle, based on severity of error, application logic, etc.
- Advantages:
 - Permits applications to implement their own redundancy (e.g., app-directed replication)
 - Stronger guarantees about semantics may be possible (e.g., all-or-nothing completion), albeit at greater overheads
- Disadvantages: Performance limitations
 - Attributable failures can only be delivered for blocking calls
 - All-or-nothing completion implementations must enforce stricter data update mechanisms (e.g., begin/end transaction, logging or non-in-place updates)

Fail-fast (performance)

- OpenFAM failures cause application to be killed
 - Similar to OpenSHMEM semantics
 - No guaranteed atomicity of operations; partial completions may occur
 - Applications should be structured to perform appropriate application-specific recovery if needed
- Advantages:
 - Potential for improved performance (non-blocking operations possible)
 - Guarantees better matched to direct access (load/store) mode
- Disadvantages:
 - Fabric or FAM failures may result in corrupted persistent data

Areas for discussion

- Access control
- Memory ordering
- Scratch regions
- PE synchronization
- Error reporting and handling
- Data item resizing
- Multi-item gather/scatter
- Data item snapshots
- Scalable metadata service

Access control

- Goal: accesses to FAM-resident data are permitted only if requesting user has sufficient rights
- Challenge: verification must be done by trusted entity like the OS, which can be costly
- Potential approach: fam_open() and fam_close()
 - fam_open(): performs mandatory name-to-descriptor translation and permissions check before data item/region access, and returns Fam_Descriptor
 - Fam_(Region_)Descriptor treated as capability: identifies data item (region) and access rights granted
- Added benefits:
 - Explicitly tracking open data items (regions) could simplify delayed reclamation
 - Explicit close operation provides opportunity for asynchronous error notification

Error handling and reporting

- Goal: provide richer error notification than fail-fast model for non-blocking calls
- Potential approaches:
 - Delay asynchronous error notification until subsequent `fam_quiet()` call
 - Delay asynchronous error notification until `fam_close()` call
 - Explicit non-blocking calls: modify all non-blocking calls to return handles
 - Application can use handles to check call's progress and completion status (success vs. failure)
 - To simplify checking status for multiple outstanding requests, API could also include calls for coalescing handles

Summary

- OpenFAM API is a programming model for disaggregated persistent fabric-attached memory
 - Focus on fabric-attached memory (FAM) instead of remote processing element (PE) memory
 - Introduce naming, permissions and failure handling model to accommodate persistent FAM-resident data
 - Employ two-level allocation scheme to manage large FAM capacities
 - Use opaque descriptors for portable references to FAM-resident data across OS instances
 - Provide map/unmap API to leverage memory semantic fabric load/store access capabilities

- Status
 - “Ported” example applications to validate OpenFAM API
 - Ex: sparse matrix-vector multiplication, large-scale graph inference, PageRank pipeline, sort, random access
 - Implementation is in progress

- We’re interested in your feedback
 - Draft of OpenFAM API spec available for review: <https://github.com/OpenFAM/API>
 - Email us at openfam@groups.ext.hpe.com