OPENFABRICS ALLIANCE

14th ANNUAL WORKSHOP 2018

# PERSISTENT MEMORY OVER FABRICS

## BEYOND HIGH AVAILABILITY


Paul Grun


April 10, 2018

CRAY THE SUPERCOMPUTER COMPANY

SNIA SSSI | SOLID STATE STORAGE

# REMOTE PERSISTENT MEMORY

- **Remote Persistent Memory is something different**

- **It might prove to be a transformative technology**
- **It's unlikely to take the industry by storm**
- **It's going to take some work**

**Objective for today is to establish the basis for a detailed exploration of Remote Persistent Memory**

- **Significant work already done on RPM for High Availability**
  - Well-defined use case
  - Requirements are pretty well understood
- **But not too much work on other use cases … yet**
  - Use cases not as well defined
  - Why?

**Which came first,**

**the technology chicken?**

**or the application egg?**

# SCOPE FOR AN RPM DISCUSSION

- **Locality**
  - A PM device accessed over a network
  - ~~A local PM device attached to an I/O bus or a memory channel~~

- **Access Method**
  - Persistent Memory as a target of memory operations (hence, ‘memory’)
  - ~~Persistent Memory as a target of I/O operations e.g. NVMe~~

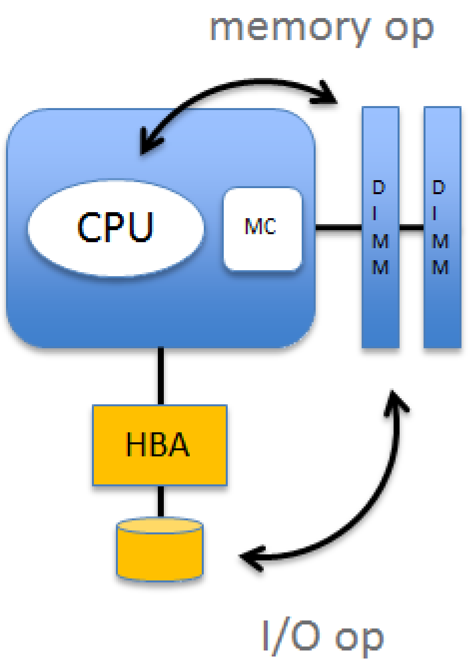
- **Implementation**
  - Byte-addressable non-volatile memory, or
  - Block-based NVM devices

Note the distinction between *access method* and *implementation*

# ACCESS METHOD - MEMORY VS I/O

**Memory operations**

- Data is moved between a CPU register and a memory location
- Memory location is identified by a real or virtual memory address
- Fast and synchronous - no CPU stalls




**I/O**

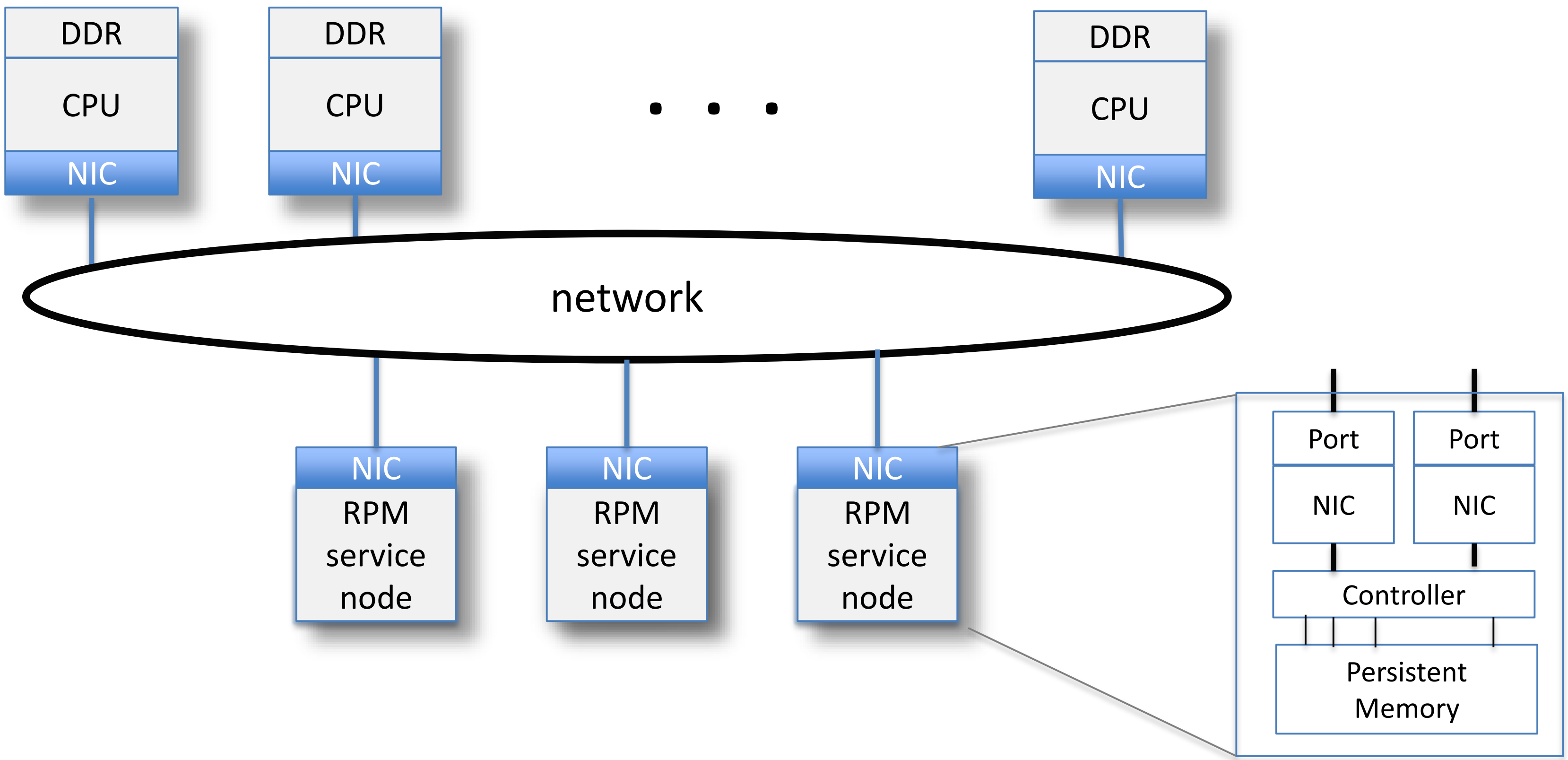
- An extent (block) of data is transferred between memory and a storage device
- The block is identified by an abstract, protocol-specific identifier (e.g. an LBA)
- Uses asynchronous I/O techniques

Let’s agree: PM refers to accesses to a non-volatile memory device using memory semantics

# AGENDA

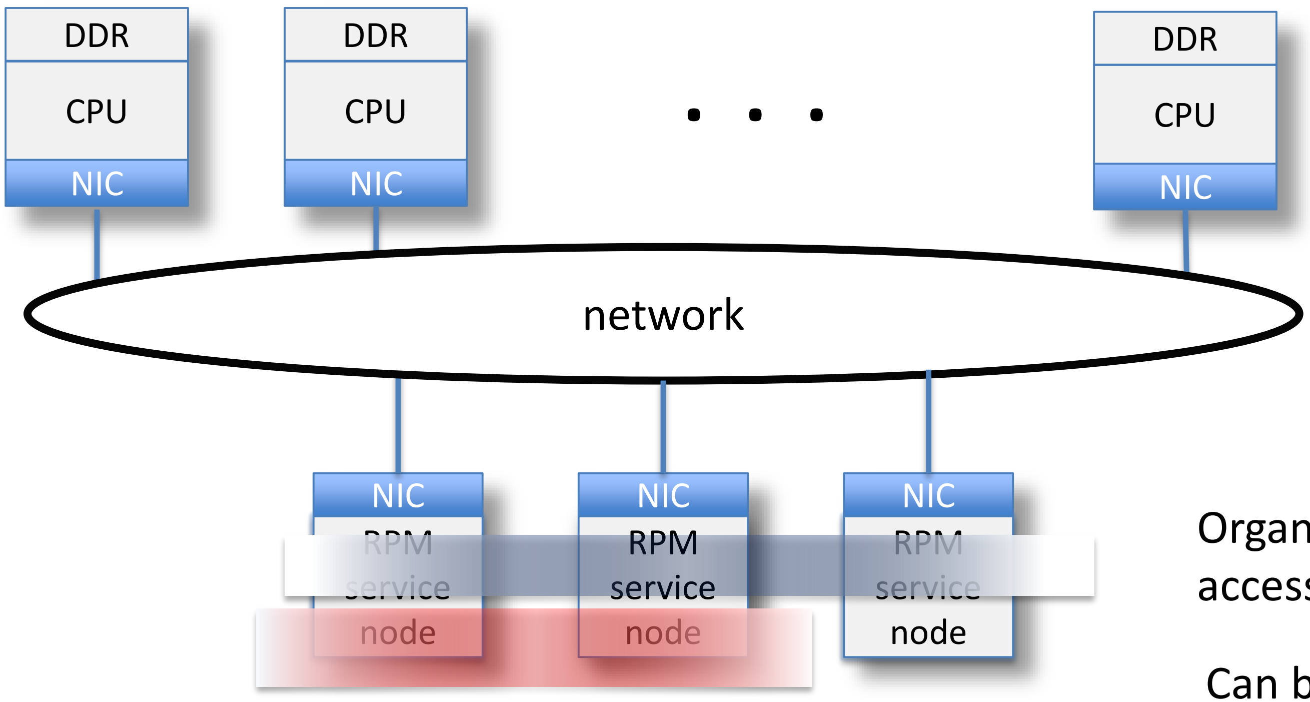
- **System and Memory Models**
- **An Example Application**
- **A Multi-dimensional Problem, Some Factors to Consider**
- **Various Use Cases for Remote Persistent Memory**
- **What's Next**

# GENERAL SYSTEM VIEW




Think of Remote Persistent Memory as a *service* located on a network

# MEMORY MODEL




Organized into pools, accessed as memory

Can be configured as a flat address space, or as object storage

Or both

# RPM FOR GRAPH ANALYTICS

- **Why?**
  - Operate on larger graphs than would fit in local memory
    - Solve Petabyte-sized graph problems on 1,000 node systems
    - As opposed to 10,000 nodes
  - Persist data structures between program executions
    - Run multiple query jobs sequentially and potentially in parallel
  - Use existing programming models and languages
  - Make better use of available DRAM for algorithms, not just holding data
- **Alternatives**
  - Limit the size of graphs one can study to what fits in memory
  - Use out-of-core methods which store graph data structures on disk
    - Apply traditional HPC graph algorithms, but only read in portions of the graph at a time
  - Store graphs in large NoSQL database, write new algorithms

# EXAMPLE - INTRUSION DETECTION

- **Demonstrates key differentiators of RPM**
  - Small word random access
  - Very large data structures sparsely accessed over an extended time
- **After an initial alert the time and data intensive work begins**
  - Verify a compromise
  - Determine the scope
- **Typically done by examining comms logs**
  - Netflow is the most popular format
  - Contact chaining of compromised computers with potential victims

# PYTHON COLLECTIONS OVER RPM

- **Created Python modules that implement various collection modules**
  - TCP sockets for transport
  - Pickle objects for serialization
  - Key-Value servers run on compute nodes
- **Maintained Python API**
  - Requires a name and a meta server for each collection
- **Anyone who can use a Python dictionary can use these modules**
  - Maintaining the collections API and functionality was of primary concern
  - Performance was secondary – but still good!

# MORE POSSIBLE APPLICATION TARGETS

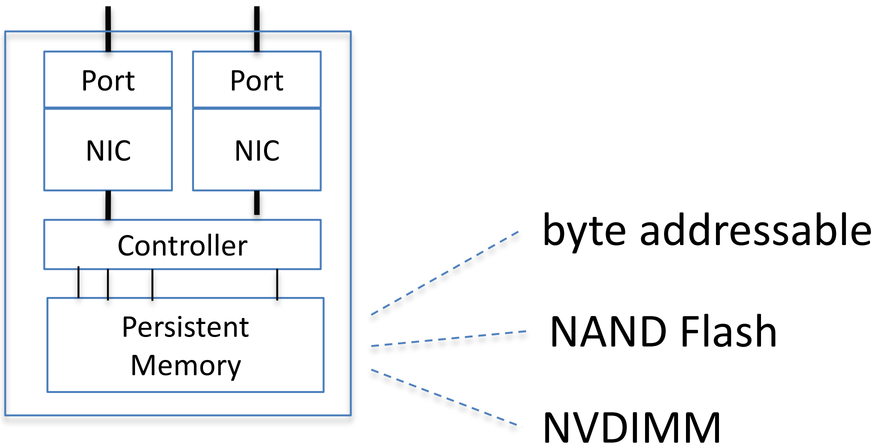
- **Scale up Databases**
  - Operate on datasets larger than would fit into traditional memory
  - Persist data structures between program executions
  - Avoid disk accesses
- **Scale out (distributed) Databases**
  - Simple methods for creating a common data store shared among instances
  - Persist data structures
  - Avoid disk accesses
- **Graph Analytics**
  - Mentioned previously
- **HPC Applications**
  - GLOBALMEM symmetric heap
  - Asymmetric RPM regions

# SOME KEY FACTORS

- Target application?
  - Analytics
  - HPC
  - DB apps
- System objective?
  - scale out/up?
  - server disaggregation?
  - persistence?
- Resource allocation?
  - Job launch?
  - At runtime?
- Programming environment?
  - SHMEM
  - Python
  - Java
  - Chapel, UPC
- NVM device implementation?
  - Block-based vs byte addressable device
  - Driven by access patterns and economics
- Memory model?
  - flat memory model
  - object-based model

Factors that may, or may not, have an impact on API and fabric design

# PM DEVICE IMPLEMENTATION

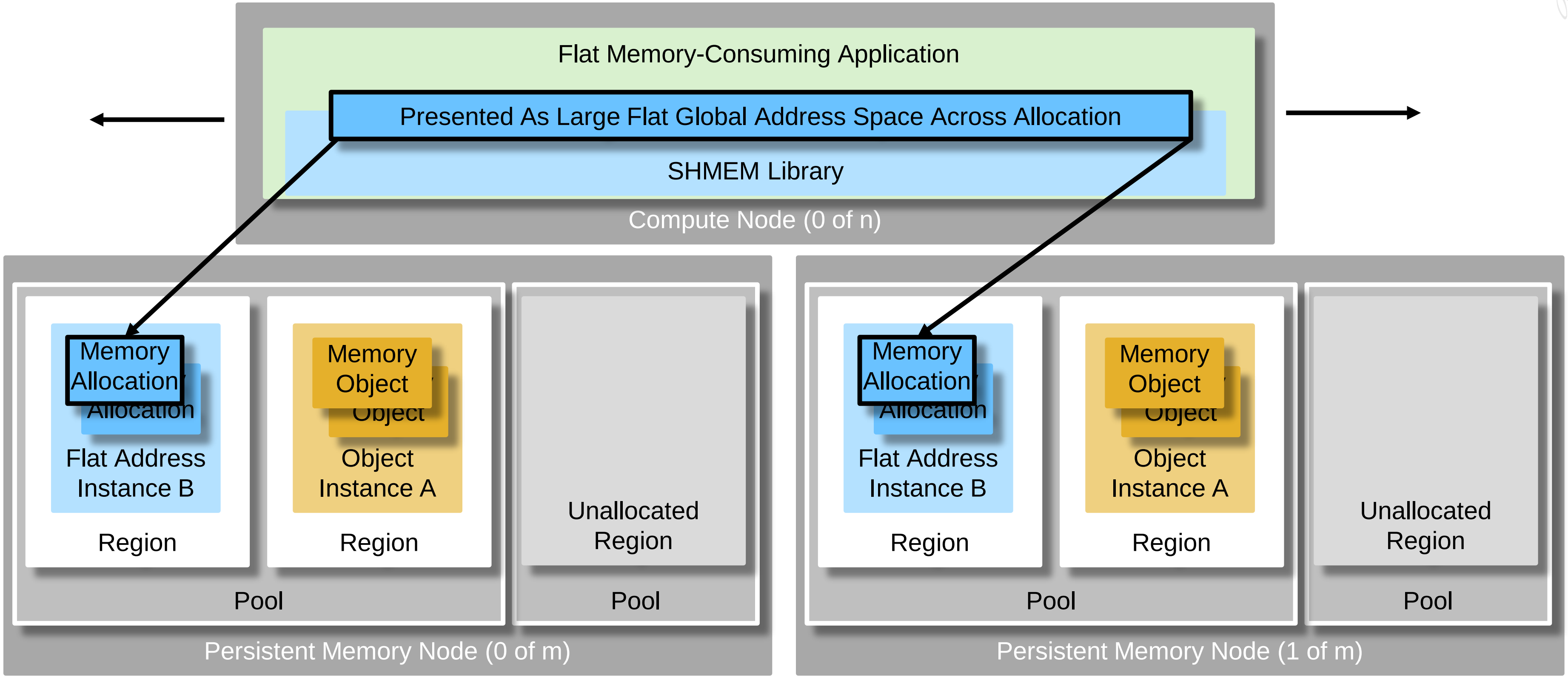



Regardless of the technology on which RPM is implemented, it is still accessed as remote memory

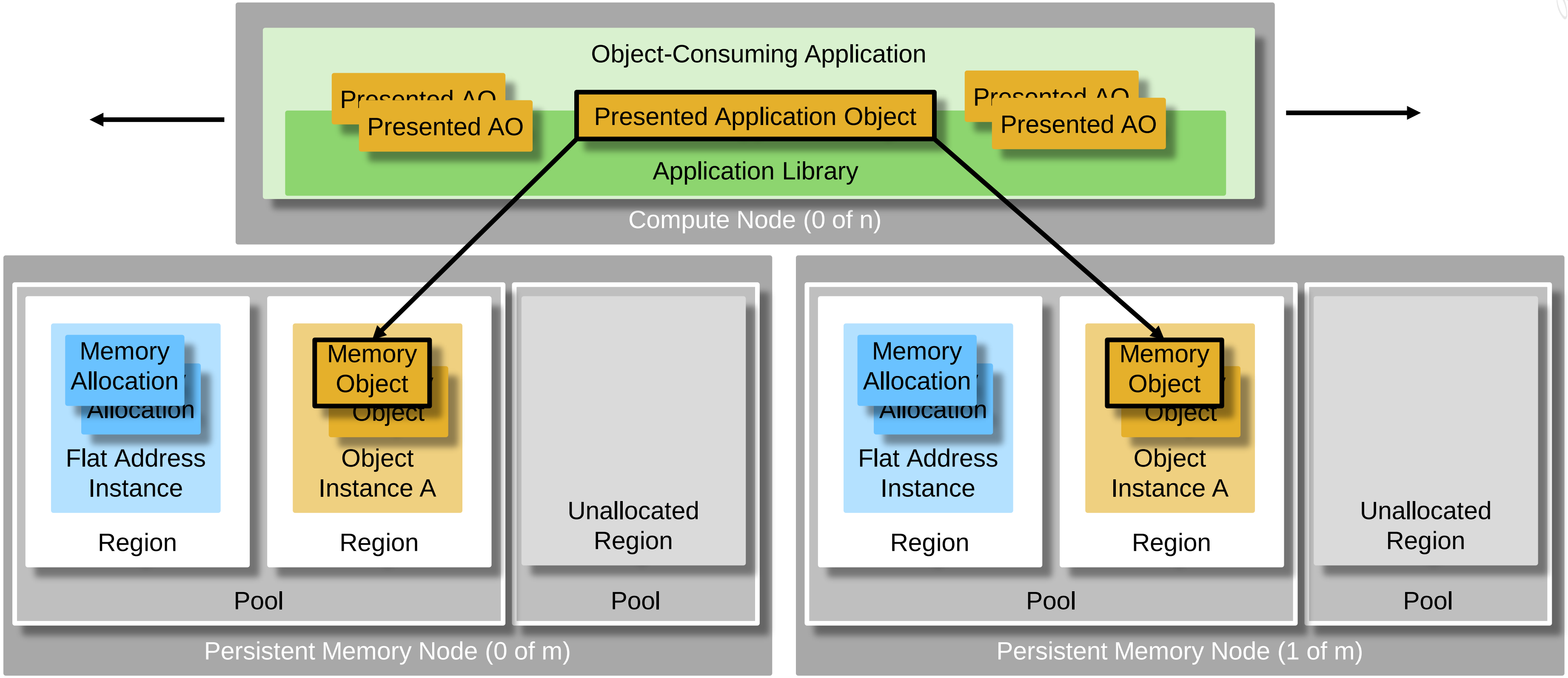
The technology choice is driven by application access patterns and by economics

- Byte addressable NVM devices
  - $$
  - Big capacity
- NAND Flash
  - Existing analytics frameworks stream data sequentially
  - A flash-based system could be cost effective
- NVDIMM
  - capacity limited

# Flat Memory Model, Object Store



# Object Store, Flat Memory Model

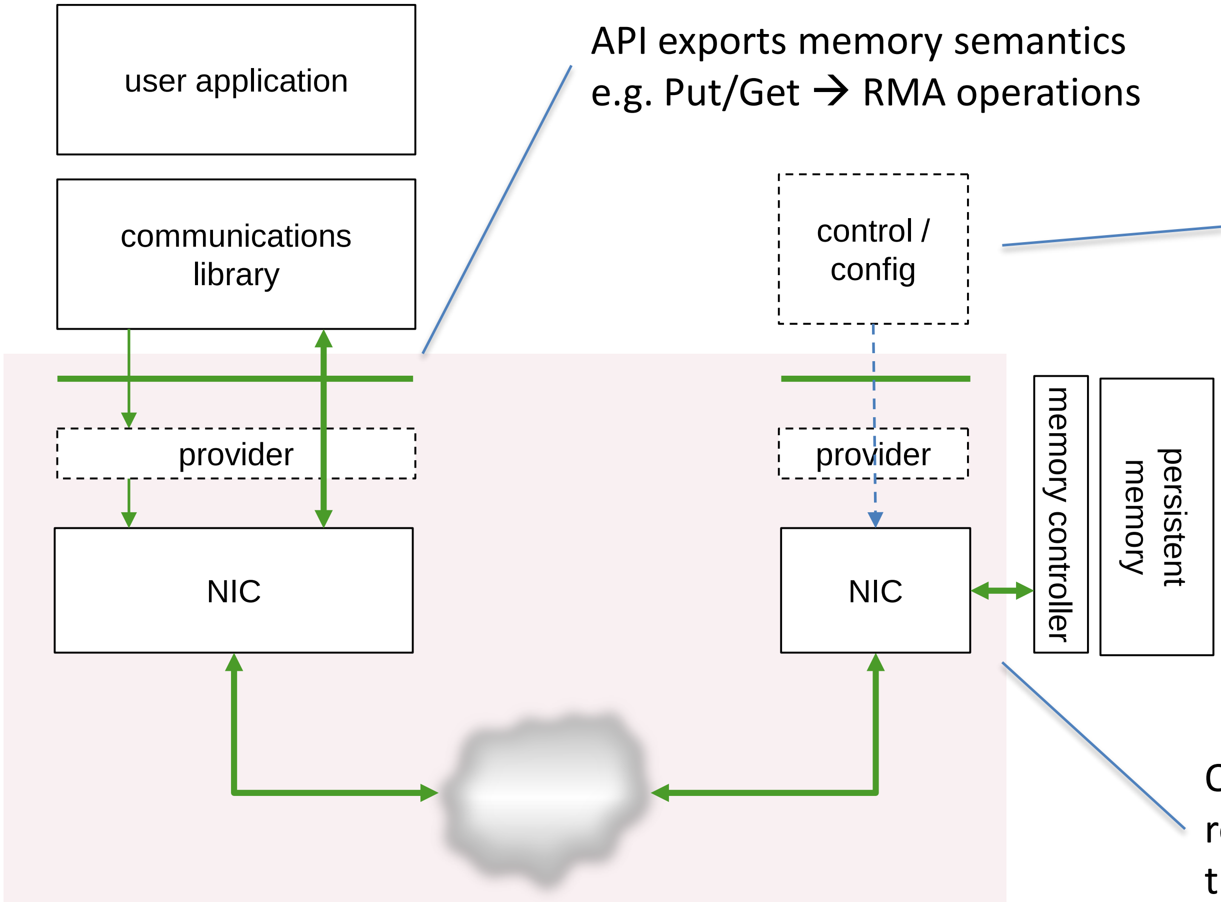


# FLAT MEMORY VS OBJECT STORE

- **Flat Memory**
  - Simple, well-understood memory model
  - Difficult to share among uncoordinated apps
- **Object Store**
  - Sharing of memory-resident data across uncoordinated processes, languages
  - Named data persists across jobs, program executions, and time
  - Complex big data analyses across larger datasets
  - Database-like features
  - Security, resiliency
  - Good for high value data

Expect both to be implemented – APIs and networks should support either

# FLAT MEMORY TARGET NODE

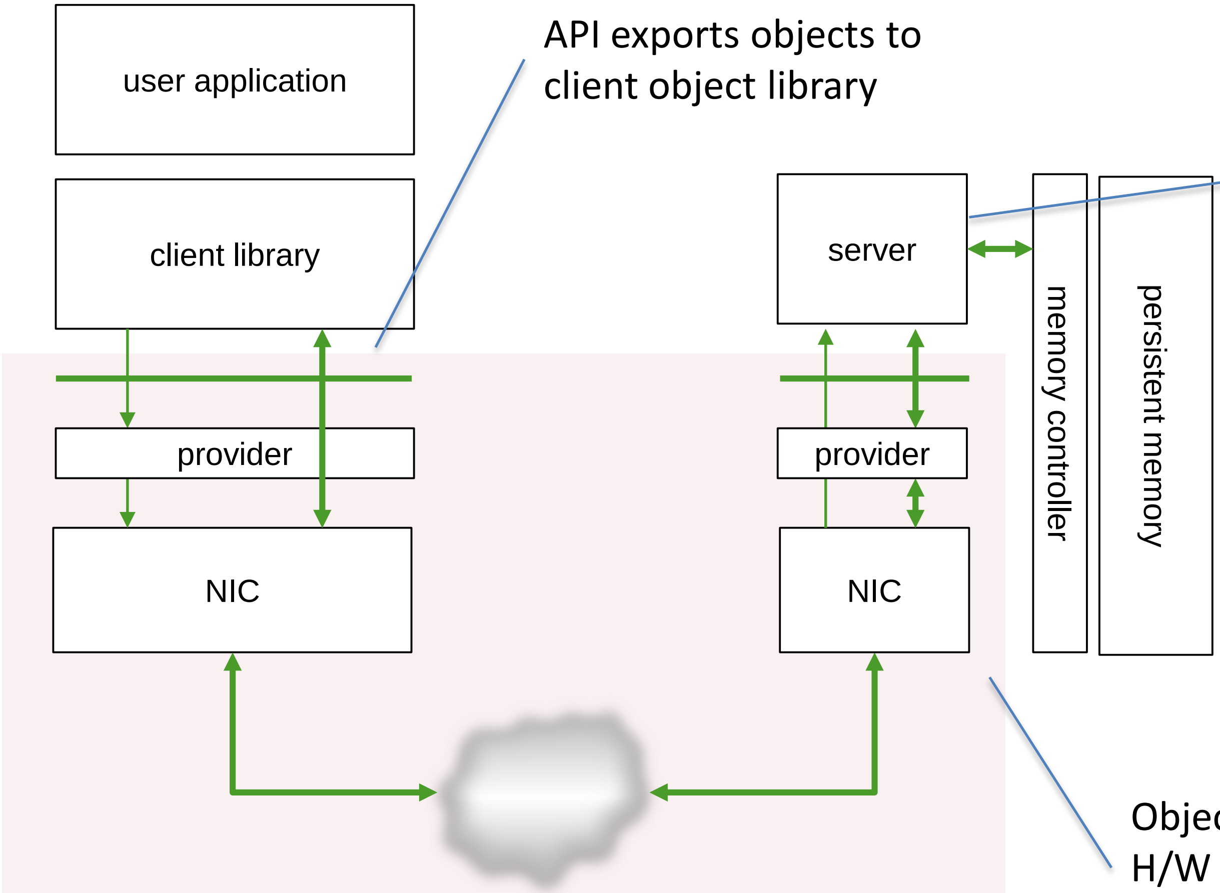



API exports memory semantics
e.g. Put/Get → RMA operations

Device resource allocation, operation setup

Optimized path for memory reads/writes. No software in the access path

# OBJECT STORE TARGET NODE




API exports objects to client object library

Provides object services
- journaling
- sharding/resiliency
- allocation
- data aging
- object versions

Object Transport
H/W accelerated translation of objects to memory addresses

# SYSTEM OBJECTIVES

- **High availability**
  - Replicate local cache to RPM to achieve High Availability

- **Scale out**
  - Scale out distributed database or analytics applications

  → shared Remote Persistent Memory

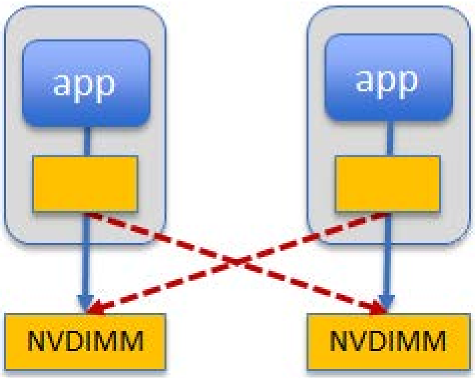
- **Scale up**
  - Scale up databases that exceed local memory capacity

  → unshared Remote Persistent Memory

- **Disaggregation / independent scaling of memory and compute**
  - Applications that scale linearly with memory footprint

  → unshared Remote Persistent Memory

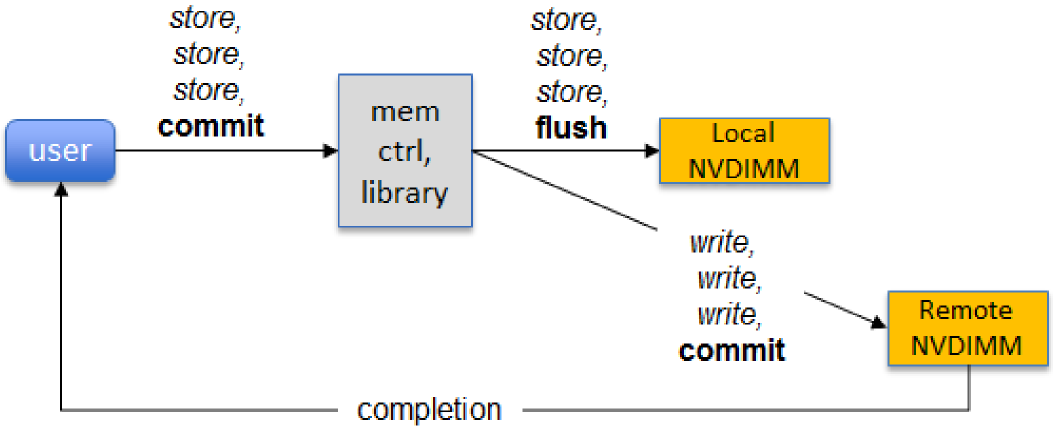
# USE CASE: HIGH AVAILABILITY, REPLICATION




What it looks like

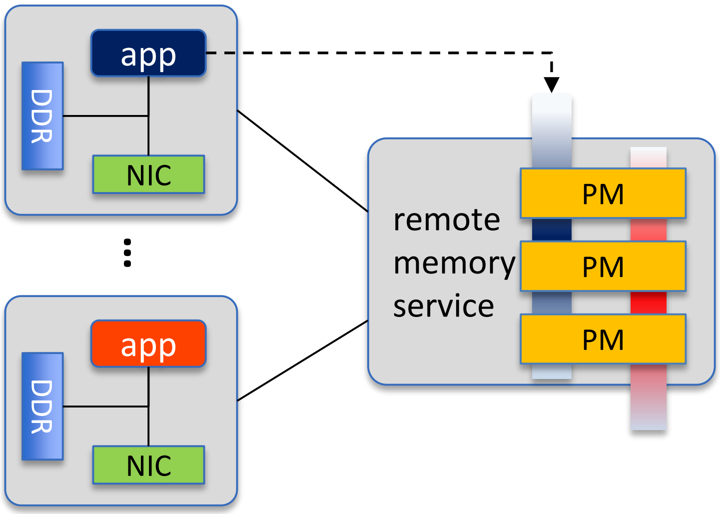
Usage: replicate data that is stored in local PM across a fabric and store it in remote PM

How it works




"High Availability"

# USE CASE: REMOTE PERSISTENT MEMORY

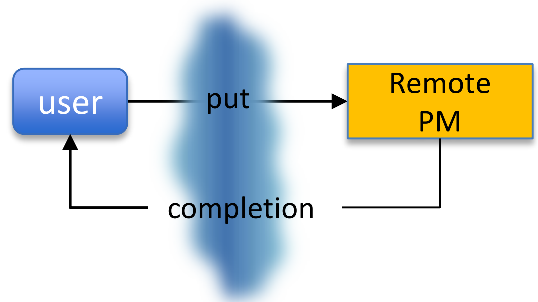



What it looks like

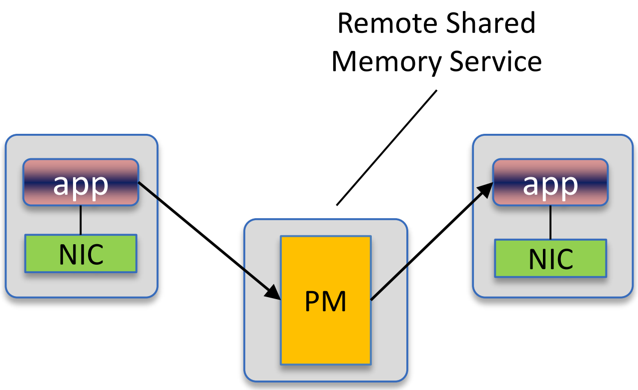
“Scalable Memory”

Usage: Expand on-node memory capacity, while taking advantage of persistence (or not). Disaggregate memory from compute.

How it works



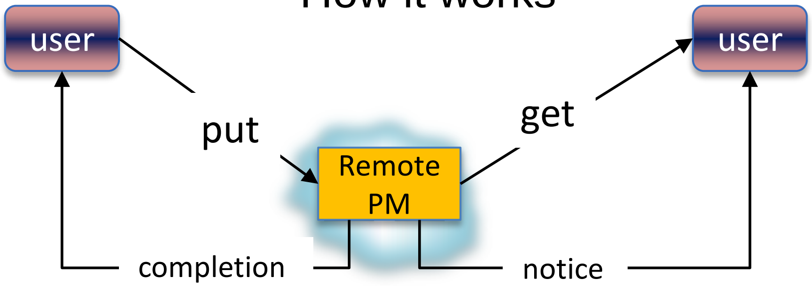
# USE CASE: SHARED PERSISTENT MEMORY




What it looks like

Usage: Information is shared among the elements of a distributed application. Persistence can be used to guard against node failure.

How it works




"Scale-out Applications"

# FACTORS AFFECTING THE API/NETWORK

- **Object Store**
  - Export object semantics to the consumer
  - Intelligence in the target node to manage object features
- **Flat Memory**
  - Export memory semantics to the consumer
  - Simple target node designs
  - Address translation features
- **PM Technology**
  - Block oriented devices may require intelligence for byte level access
  - Byte oriented devices may require more sophisticated network protocols
- **Resource Allocation**
  - Resource allocation to applications whether scale-out or scale-up

# STEPS FORWARD – A LOT TO THINK ABOUT

- **Remote PM for High Availability has been discussed extensively**
  - A set of fabric features to support HA has been explored, and is in process

- **Getting beyond HA**
  - Begin by understanding the relevant use cases
    - Even those that don't yet exist
  - Understand the access patterns and value propositions associated with those use cases
  - Use those to develop "application centric requirements" to drive API design
  - Develop the necessary APIs

# DRIVING ADOPTION OF RPM

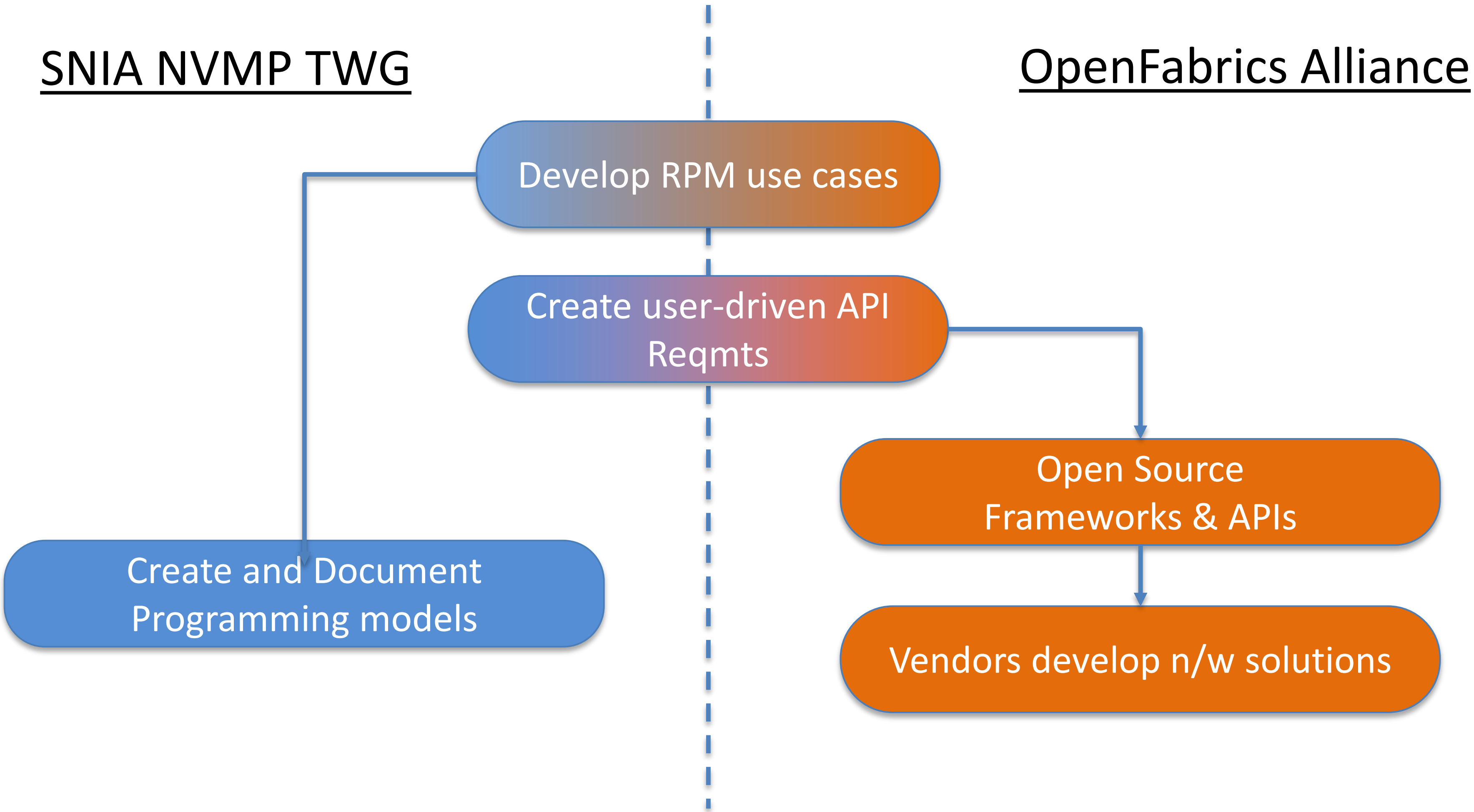
Adoption of Remote Persistent Memory requires:

- A common understanding among application developers of the behaviors that are required to reliably access Remote Persistent Memory, } SNIA

- The means for an application to implement those required behaviors } OFA

This is going to take some serious effort

# ANNOUNCING - SNIA & OPENFABRICS ALLIANCE



14th ANNUAL WORKSHOP 2018

# THANK YOU

Paul Grun

Cray Inc