SNIA PERSISTENT MEMORY PM SUMMIT

JANUARY 24, 2019 | SANTA CLARA, CA

Performance, Capacity, Persistence – Which one(s)?

Paul Grun
Advanced Technology Development, Cray

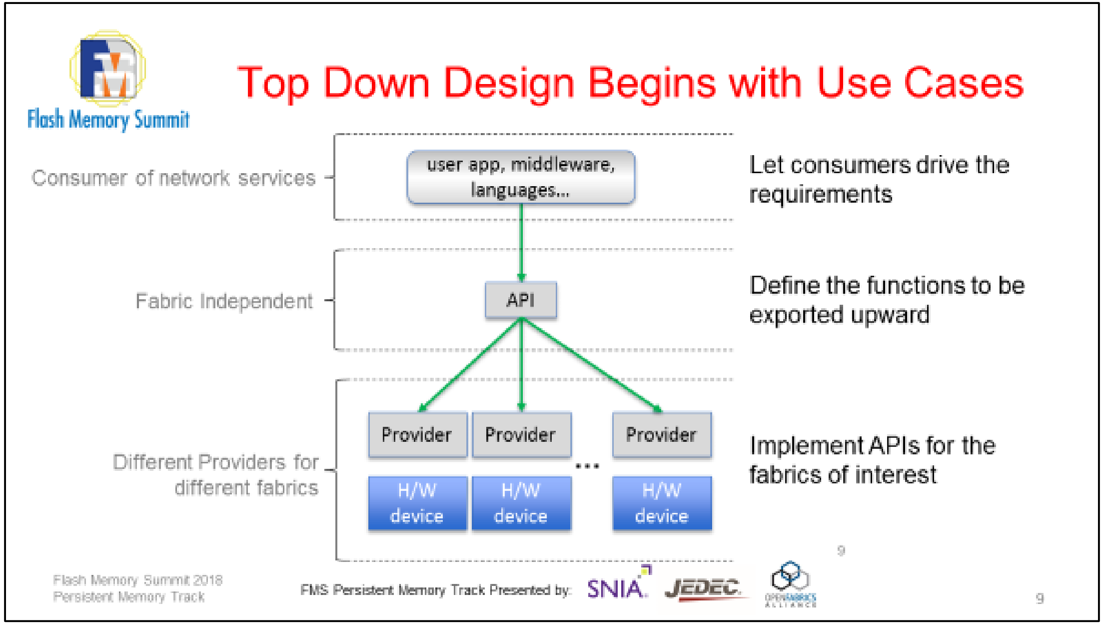
- Many view the emerging PM layer in the memory hierarchy as monolithic, evolving toward Nirvana
  - Nirvana defined as “infinite capacity, infinite bandwidth, zero latency, zero cost”
  - Oh, and “infinite retention”
- The truth is that there will always be tradeoffs
  - Performance vs Capacity vs Cost
  - Local vs Remote
- How to choose the right tradeoffs?

Hmmn. Maybe start at the top?

"The Case for Use Cases"

At FMS 2018, we began to shift the focus onto a discussion of "use cases".

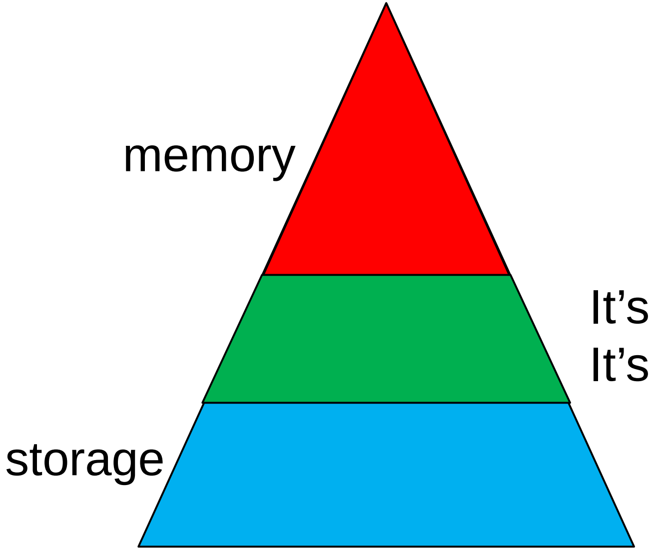
This year's PM Summit continues that trajectory



"Mr. Chairman, I'd like to revise and extend my remarks"

The Familiar Memory Hierarchy

It's clear that Persistent Memory isn't exactly memory, and it's not precisely storage



It's a dessert topping!
It's a floor wax! *

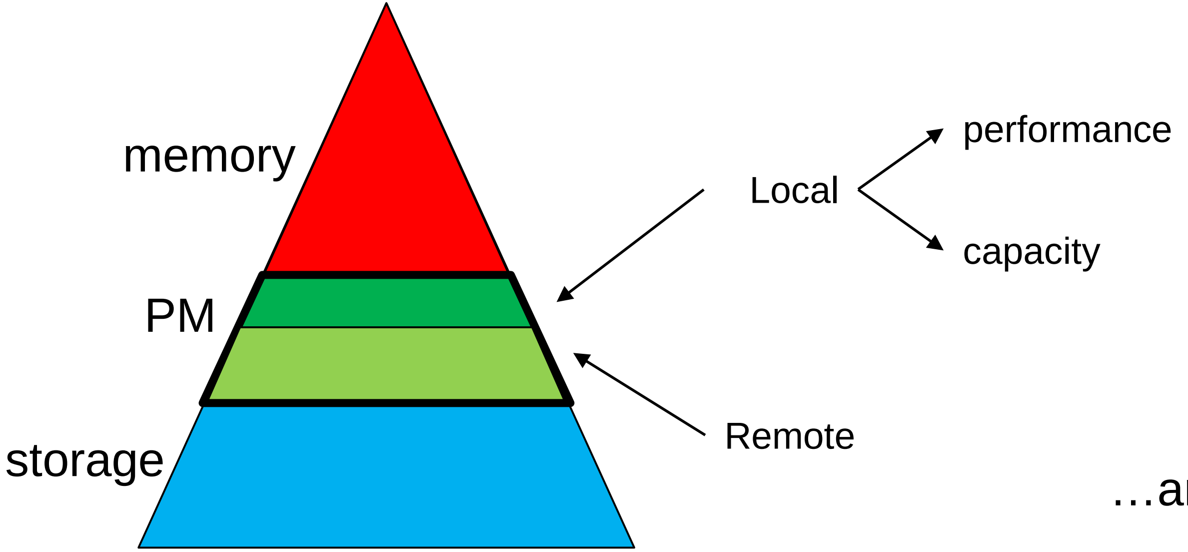
…so how do we characterize it?
What role does it fill, exactly?

* With thanks to SNL, 1/10/76

The Familiar Memory Hierarchy…

… with a wrinkle

Turns out that this new layer isn't monolithic…



…and there are tradeoffs within the sublayers

Choices!

Selecting the right technology depends on understanding (at least):

- The key system design objectives
  - Scalability? In which dimension? Single server? Cluster?
- Application requirements
  - Is data being shared among threads or nodes?
  - Are there application performance or capacity requirements?

Resolving the tradeoffs among PM solutions depends on System Objectives and Application Requirements

Possible (likely?) Targets for PM

- Database Applications
  - A modifiable, an in-memory database that survives power cycles
- Data Analytics
  - Create a persistent database once, run new queries repeatedly
- Graph Analytics
  - Operate on larger graphs than would fit in local memory
- Commercial Applications
  - Enable collaboration on large scale projects
- HPC Applications
  - Scalability, parallel applications

Could use some help here

Possible System Objectives

- Data Availability/Protection
  - Replicate local cache to RPM to achieve high availability
- Local System Performance
  - Eliminate disk accesses e.g. to stored databases
- Scale Out Architectures
  - Scale out distributed databases, analytics applications, HPC parallel applications
- Scale Up Architectures
  - In-memory databases that exceed local DRAM capacity
- Disaggregated System Architectures
  - Compute capacity scales independently of memory capacity
- Shared Data
  - Support simultaneous data access to large teams
- Improved Uptime, Fast Restart
  - Quick server recovery following power cycle
  - Checkpoint restart
- ~~Improved Disk Storage Performance~~

revised and extended from Flash Memory Summit 2018

A topic for a storage forum, not a PM Summit.
We're talking about memory reads and writes.
For disk replacement, swap SSDs for HDDs

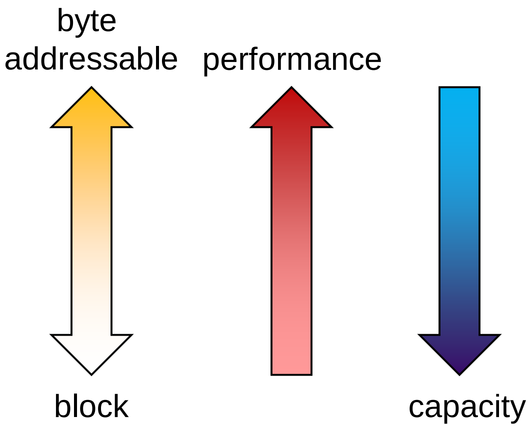
Some Consumer* Considerations

* consumer of memory services

- Application Objectives
  - Performance vs capacity?
- Sharing Models
  - Shared data vs unshared data?
  - A shared service vs a dedicated service?
- Memory Model
  - Flat address space vs object stores?
- Characteristic Traffic Patterns, Traffic Engineering Requirements
  - Small byte operations vs bulk data transfer?
- Ordering Semantics, Atomicity
- …

Nonvolatile Memory Tradeoffs

- **Technology**
  - DRAM “replacements”
    - STT-MRAM, NRAM, PCM …
  - Gap fillers (between DRAM and Flash)
    - 3DXP, Crossbar ReRAM (resistive RAM)
  - Capacity Devices
    - NAND Flash
- **Form Factors**
  - NVDIMM-N, NVDIMM-P, PCIe
- **Locality**
  - Local versus Remote



Not All Applications Value Persistence

- Persistence is valuable for:
  - High Availability applications where maintaining state between power cycles is crucial
  - Reducing or eliminating the need to access slower media, e.g. HDDs
  - Data protection and preservation
- Persistence not required, but nice to have:
  - Certain applications, such as analytics, that require establishing a database. Build the database once, run multiple queries against it
  - Collaborative workspaces

If the app doesn't need persistence, then the so-called convergence of storage and memory is uninteresting

First Order Tradeoff: Local vs Remote

- Some requirements are met by siting persistent memory devices on the local compute node
  - Capacity-based applications
  - Some High Availability usages
  - Replacement of local storage for performance reasons
- Others are only achieved by distributing persistent memory
  - Compute/memory disaggregation
    - independent scaling of compute and memory
  - Shared resource / shared data
  - Team collaboration

Use Cases – Local PM

- Data Availability/Protection
  - Replicate local cache to RPM to achieve high availability
- Local System Performance
  - Eliminate disk accesses e.g. to stored databases
- Scale Out Architectures
  - Scale out distributed databases, analytics applications, HPC parallel applications
- Scale Up Architectures
  - Scale up databases that exceed local memory capacity
- Disaggregated System Architectures
  - Compute capacity scales independently of memory capacity
- Shared Data
  - Support simultaneous data access to large teams
- Improved Uptime, Fast Restart
  - Quick server recovery following power cycle
  - Checkpoint restart

Tradeoffs - Local PM

	Persistence	Performance	Capacity
Performance	√√√	√√	√
Scale Up	√	√√√	√√√
Fast Restart	√√√	√	√

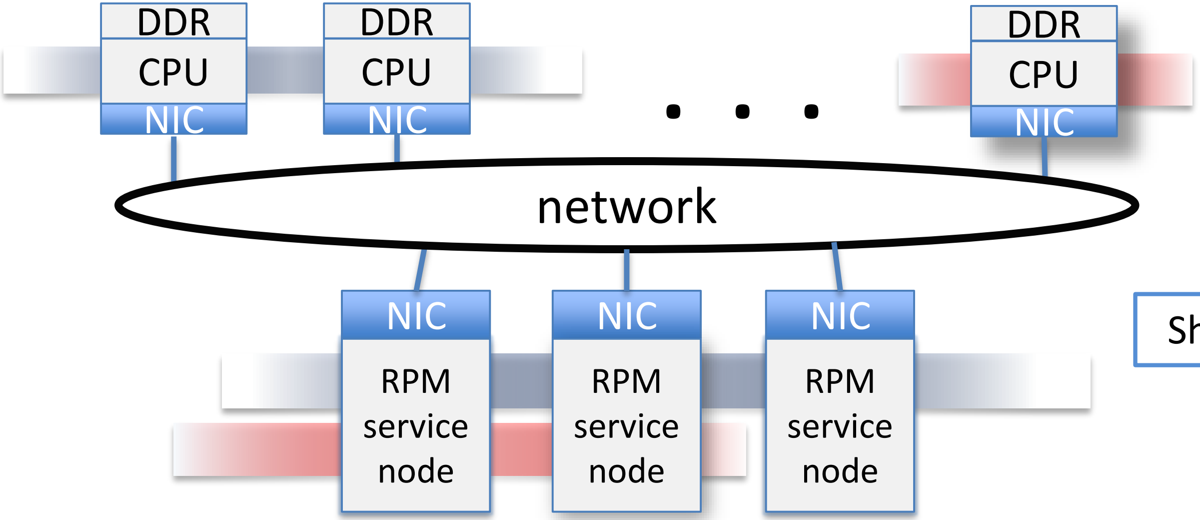
√√√ Required

√√ Desirable

√ unimportant

All are debatable.
The point is to make tradeoffs based on your use case.

Remote PM – System, Memory Model



Shared or unshared resource

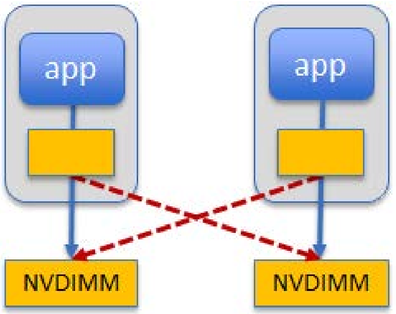
Organized into pools, accessed as memory

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

Use Cases – Remote PM

- Data Availability/Protection
  - Replicate local cache to RPM to achieve high availability
- Local System Performance
  - Eliminate disk accesses e.g. to stored databases
- Scale Out Architectures
  - Scale out distributed databases, analytics applications, HPC parallel applications
- Scale Up Architectures
  - Scale up databases that exceed local memory capacity
- Disaggregated System Architectures
  - Compute capacity scales independently of memory capacity
- Shared Data
  - Support simultaneous data access to large teams
- Improved Uptime, Fast Restart
  - Quick server recovery following power cycle
  - Checkpoint restart

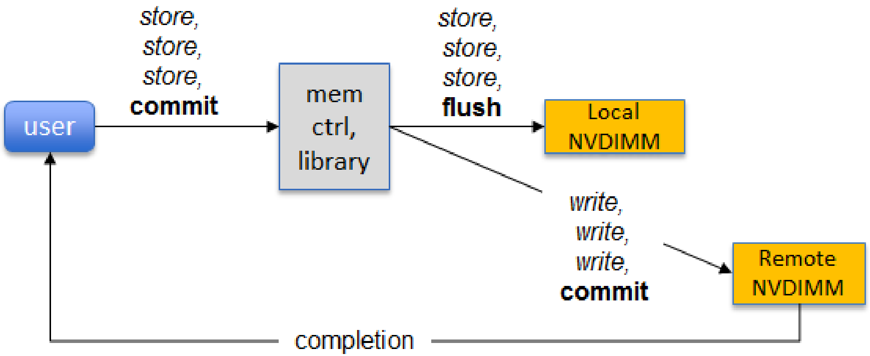
Data Protection Use Case



What it looks like

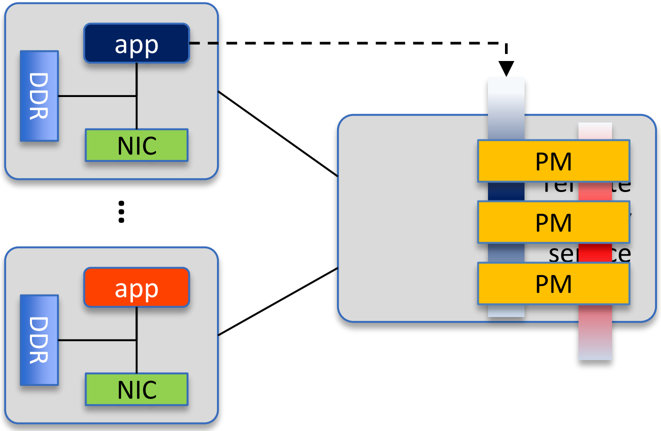
Usage: replicate data that is stored in local PM across a fabric and store it in remote PM

How it works



"High Availability"

Scale Out Use Case

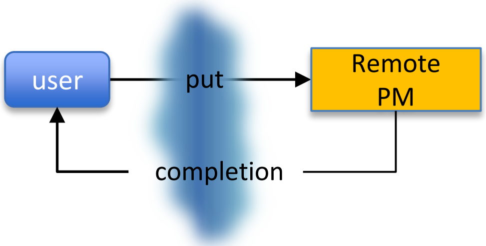


What it looks like

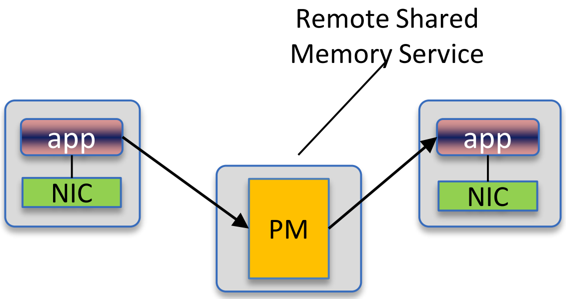
“Scalable Memory”

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

How it works



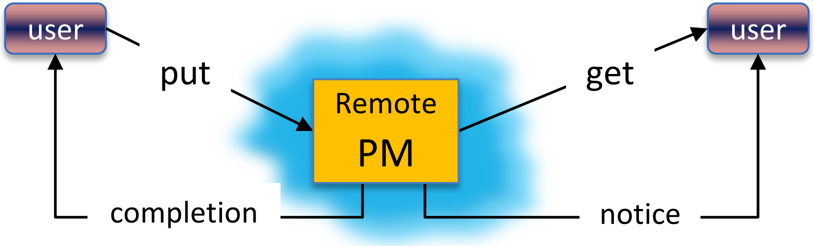
Shared Data Use Case



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"

Tradeoffs - Remote PM

	Persistence	Performance	Capacity
Data Availability	√√√	√√	√
Scale Out	√	√√√	√√√
Disaggregation	√√	√√√	√√√
Shared Data	√√	√√	√√√
Checkpoint	√√√	√√	√√

√√√ Required

√√ Desirable

√ unimportant

All are debatable.
The point is to make tradeoffs based on your use case.

A Few Interesting Apps for RPM

- High Availability
  - (Almost) simultaneous writes to local memory and remote PM
  - For data recovery and failover with little to no work loss
- HPC Checkpoint/Restart
  - Application pauses to enable rapid copy of relevant state to a checkpoint
- Distributed collaboration
  - A central shared repository for a distributed team collaborating on a large artifact
- Machine learning, Sensor data ingest and analysis
  - Ingest of large datasets
  - Data analysis accomplished by distributed threads - short random reads

Some Challenges with RPM

- NUMA, by definition
  - Probably okay, just be aware of it
- Generally requires asynchronous operation
  - Including delayed completions
- Networks introduce unavoidable latencies
  - As long as the application can tolerate it
- Transaction model will often favor pull vs push operations
  - not necessarily native to the way application writers think

Net-net, probably can’t treat remote and local PM exactly the same.
Not quite transparent, but close.

Pithy Commentary

- Understand the use case(s) first
- Consider all the attributes of PM, beyond persistence
  - Think about Cost, Performance & Capacity
- Consider the chicken and the egg
  - PM as an accelerator or existing application models,
  - PM as an enabler of new application models