15th ANNUAL WORKSHOP 2019

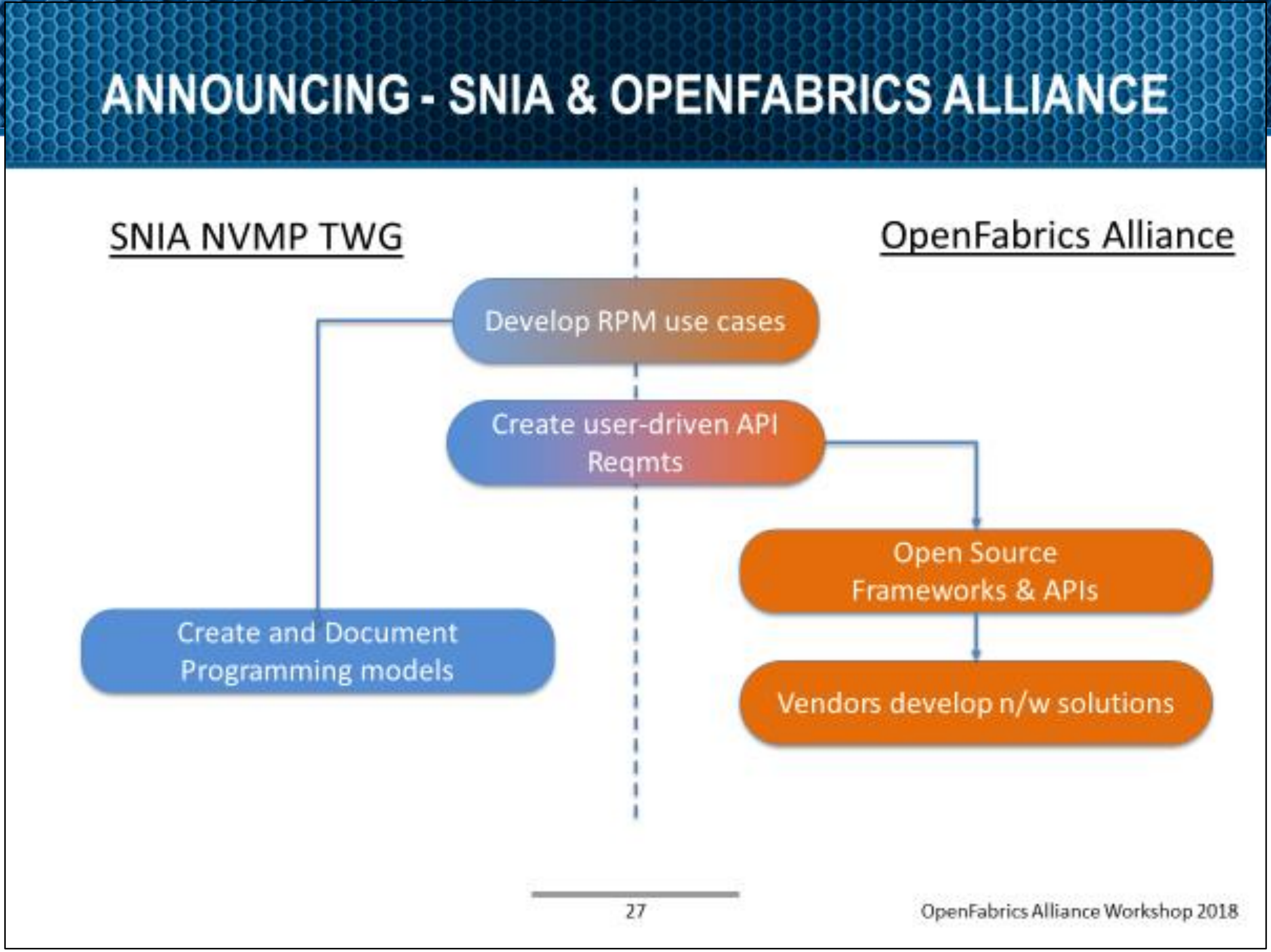
REMOTE PERSISTENT MEMORY

PERFORMANCE, CAPACITY, OR PERSISTENCE?

Paul Grun

Cray Inc

[March 20, 2019]



(from last year's Workshop)

This is an update on a long-running Work in Progress

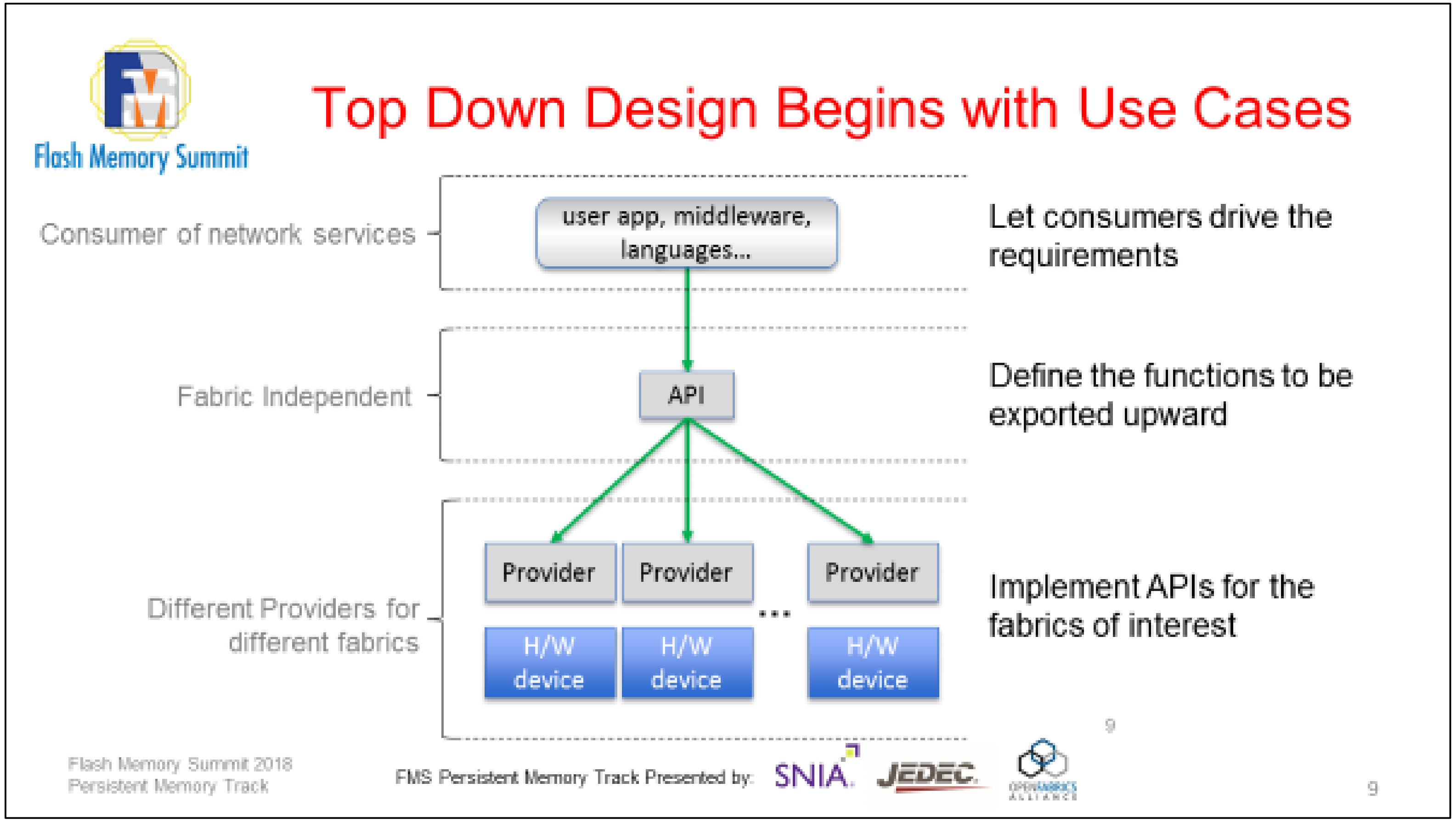
AT LAST YEAR'S WORKSHOP ...

…we began to focus on "use cases"

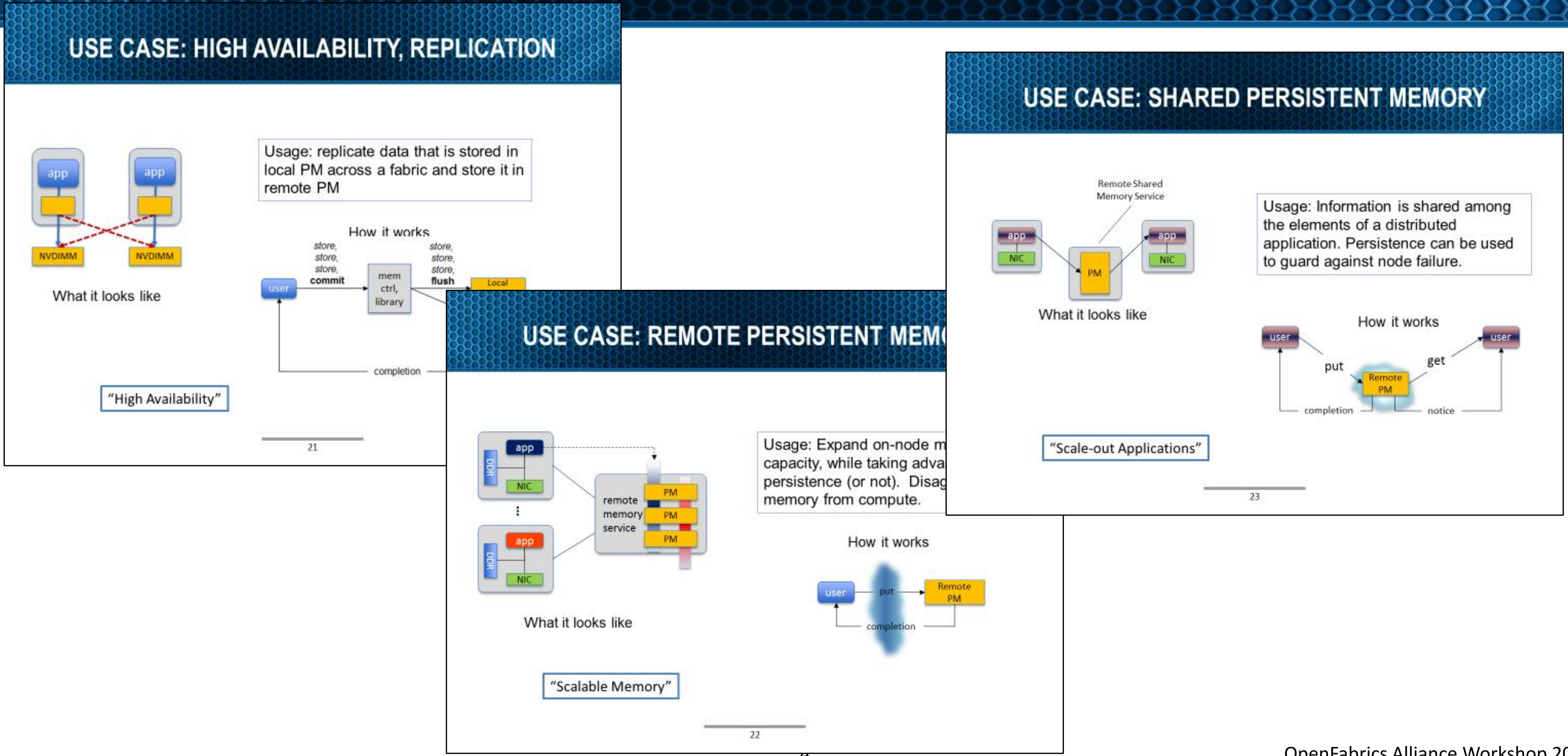
The discussion continued at FMS 2018

And continues again today

Objective is to define a set of use cases and requirements that can be used to drive an API definition



THREE CATEGORIES OF USE CASES WERE DESCRIBED



FLASH MEMORY SUMMIT 2018

Flash Memory Summit

A Multi-dimensional Problem

To craft a network solution, and particularly to optimize the network software stack, there are number of factors to consider:

- Consumer considerations
  - For what purpose is the consumer storing/accessing persistent data remotely?
  - Under what conditions are data shared?
  - What is the security model?
- System objectives
  - For any given system, what are its design objectives? Performance? Scalability? High Availability?
  - What type of service is being offered? Object store? Pools of Memory?

Flash Memory Summit 2018
Persistent Memory Track

FMS Persistent Memory Track Presented by: SNIA. JEDEC. OPENFABRICS ALLIANCE

12

We began to describe Consumer Requirements and System Objectives that will impact the network architecture needed to support RPM

PERSISTENT MEMORY SUMMIT 2019

PERSISTENT MEMORY PM SUMMIT
JANUARY 24, 2019 | SANTA CLARA, CA

- 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?

2

We began to discover that use cases imply certain system characteristics

Persistence is only one

MOVING THE BALL A LITTLE FURTHER DOWNFIELD

- **Objective for this session – integrate the various characteristics of RPM into a discussion of the use cases already presented**

- **Ultimate goal – Propose an API that meets the requirements described by the set of use cases**

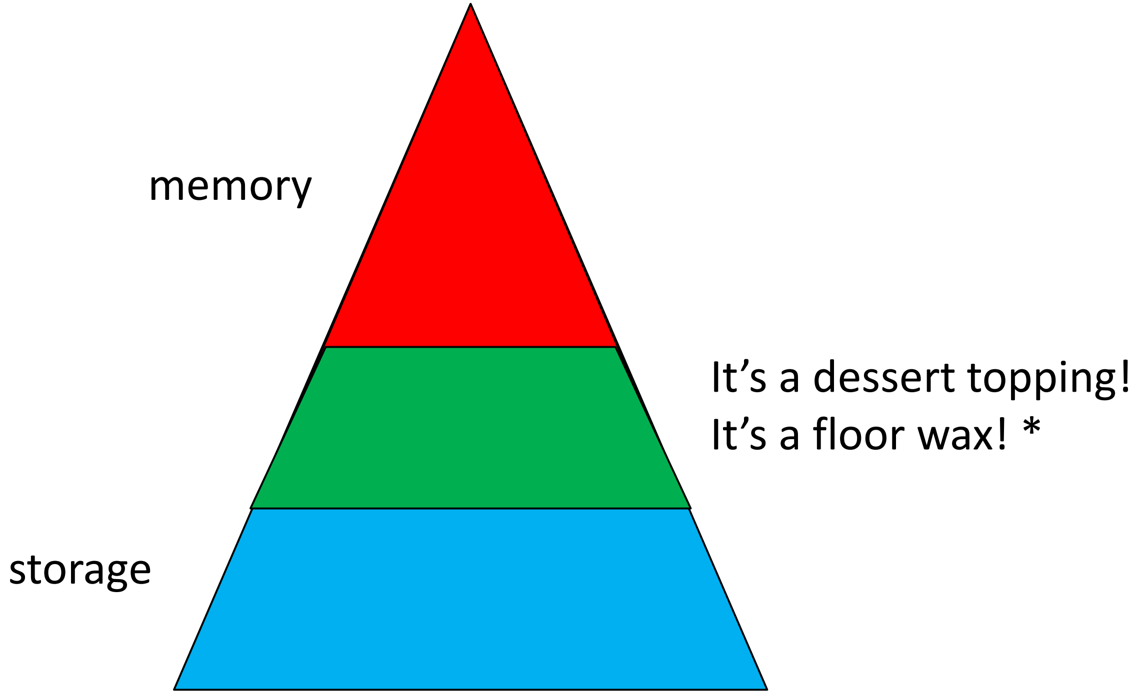
INTRODUCING THE ‘CHARACTERISTICS’ VARIABLE

- **Many view the emerging Persistent Memory 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?**

Assertion – understanding these characteristics, and how they map onto different use cases, will guide the development of networks to support RPM. (Which is our ultimate goal.)

Our objective today is to take a refined look at the emerging list of use cases and try to understand which characteristics matter most

THE FAMILIAR MEMORY HIERARCHY



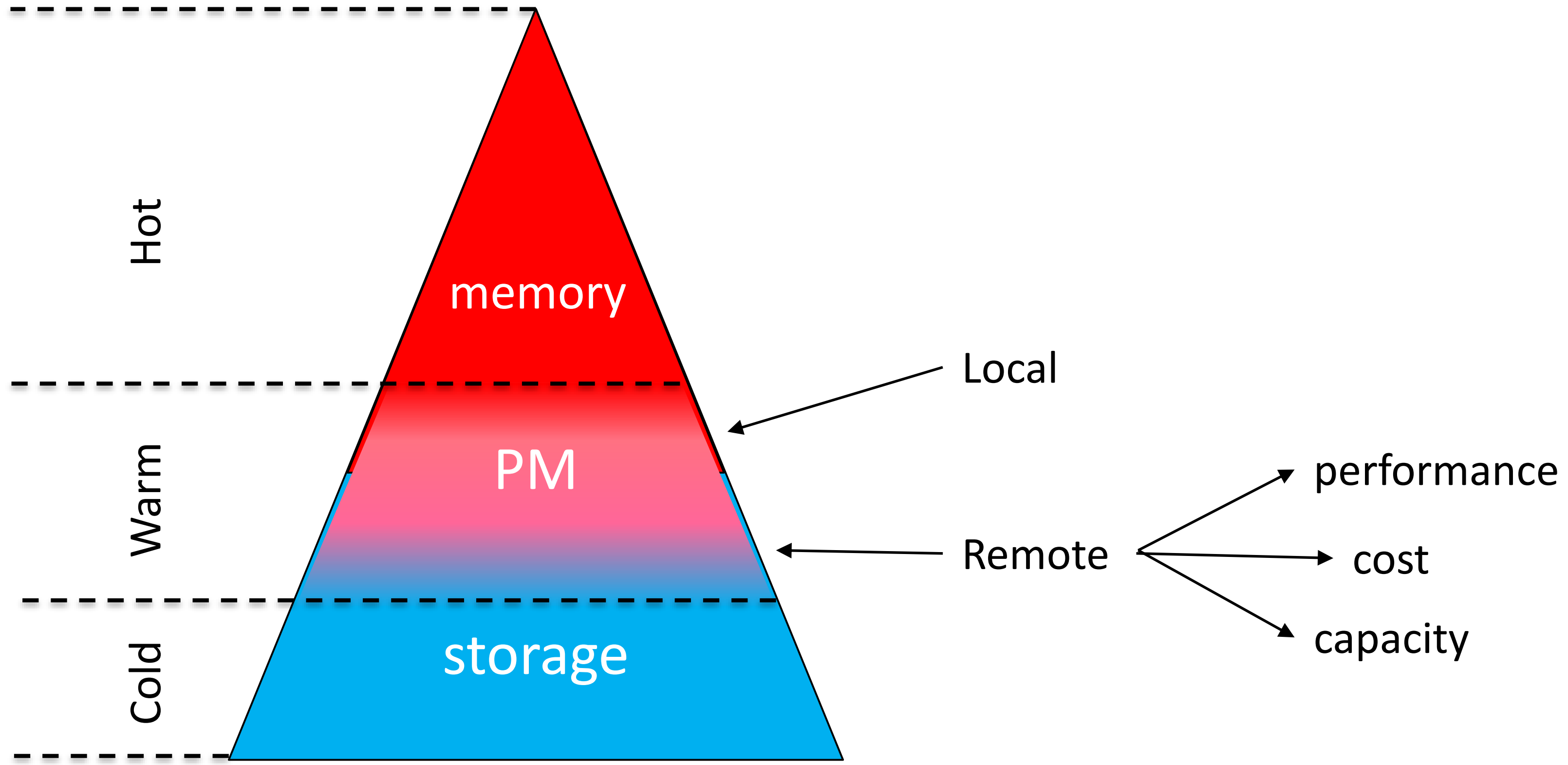
It's clear that Persistent Memory isn't exactly memory, and it's not precisely storage…

…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

KEY DRIVERS

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

Eventual API proposal should reflect a combination of *Use Cases* and *App Requirements*

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?

EXAMPLE TARGETS FOR PM

Database Applications

- A modifiable, 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
- Checkpointing

persistence, capacity

capacity, density, performance

persistence, capacity, cost

USE CASES, SO FAR - WIP

- Data Availability/Protection
  - Replicate local cache to RPM to achieve data availability
- Improved Uptime, Fast Restart
  - Quick server recovery following power cycle
  - Checkpoint restart
- Local System Performance
  - Eliminate disk accesses e.g. to stored databases
- Scale Up Architectures
  - In-memory databases that exceed local DRAM capacity
- Scale Out Architectures
  - Distributed databases, analytics applications, HPC parallel applications
- Disaggregated System Architectures
  - Compute capacity scales independently of memory capacity
- Shared Data
  - Support simultaneous data access from multiple processes
  - A central shared repository for a distributed team collaborating on a large artifact
- ~~Improved Disk Storage Performance~~

- First developed at last year's RPM Think Tank,
- Revised and extended at Flash Memory Summit 2018,
- And again at the PM Summit 2019

SOME APPLICATION CHARACTERISTICS

- 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
  - Small byte operations vs bulk data transfer?
- Ordering Semantics, Atomicity
- …

These aren’t exactly “Use Cases”, but will clearly impact the API design

PERSISTENCE? NOT ALWAYS REQUIRED

- 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
- Other characteristics may prove to be more valuable than persistence

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

FOR EXAMPLE…

- Performance
  - Persistence often comes at the cost of performance (but not always)
- Cost
  - If you can accept a lower level of performance, or you do not care much about byte addressability, there may be lower cost options available
- Capacity
  - To achieve higher capacity, you might wish to use a different technology, sacrificing e.g. byte addressability for higher capacity

1ST ORDER TRADEOFF: LOCAL VS REMOTE

- Some requirements are met by siting persistent memory devices on the local compute node
  - Capacity-based applications
  - Some data protection 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
  - Distributed/Scale-out applications

Needless to say, this is our focus at the moment - RPM

Local may be synchronous, Remote is almost certain to be asynchronous

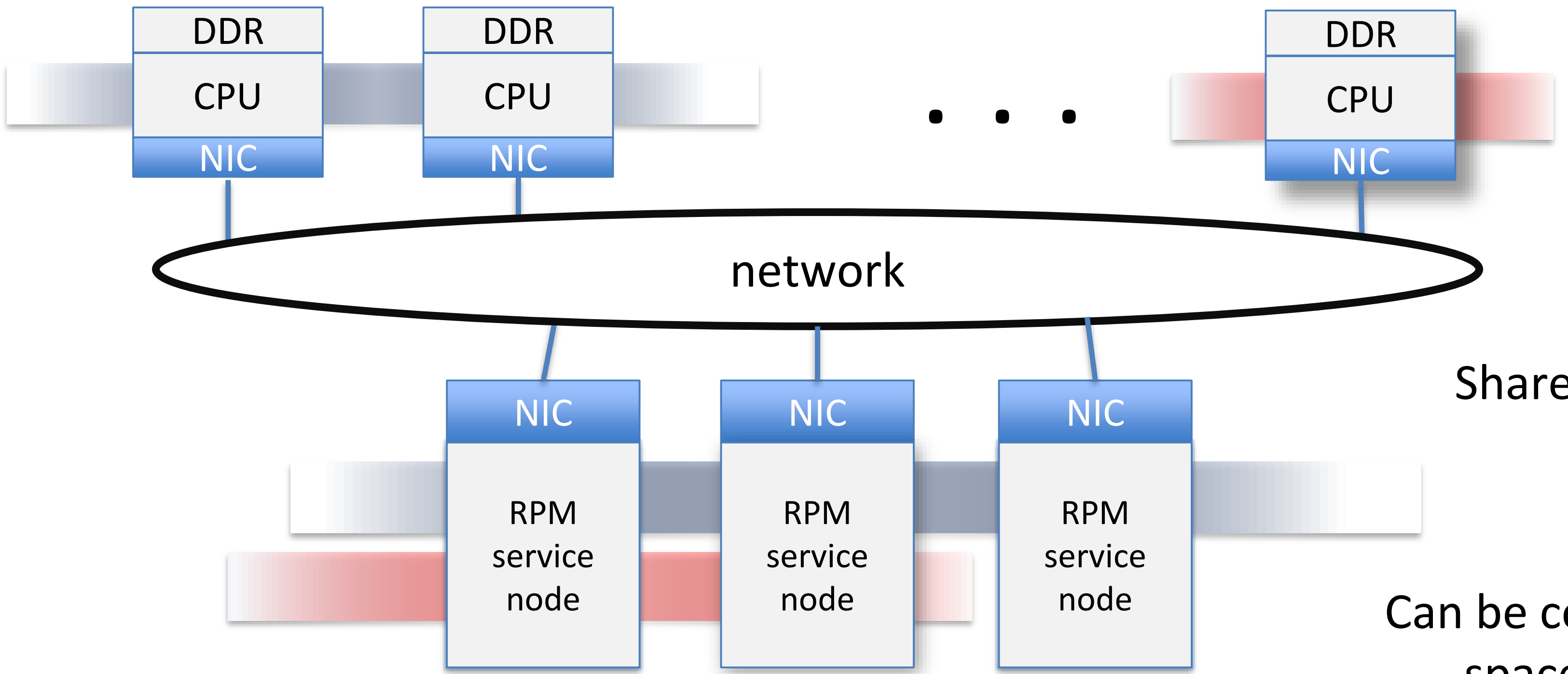
USE CASES – LOCAL PM

- Data Availability/Protection
  - Replicate local cache to RPM to achieve h
- Local System Performance
  - Eliminate disk accesses
- Scale Out Architectures
  - Scale out distributed databases, analytics
- 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

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

these need to be refined and developed in much more detail

REMOTE PM – SYSTEM AND MEMORY MODELS



Shared or unshared resource

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

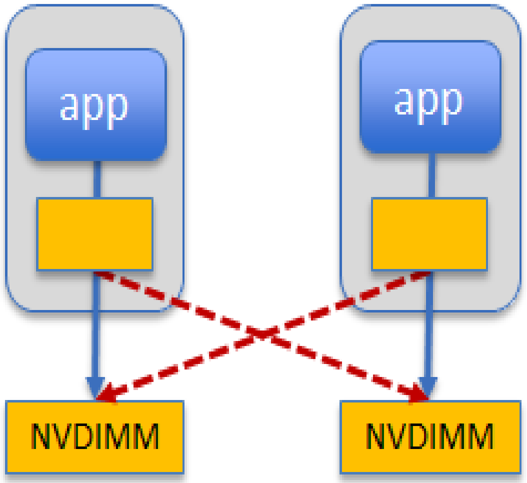
Organized into pools, accessed as memory

All will have a significant impact on the API

USE CASES – REMOTE PM

- Data Availability/Protection
  - Replicate local cache to RPM to achieve high availability
- Improved Uptime, Fast Restart
  - Quick server recovery following power cycle
  - Checkpoint restart
- 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 from multiple processes
  - A central shared repository for a distributed team collaborating on a large artifact

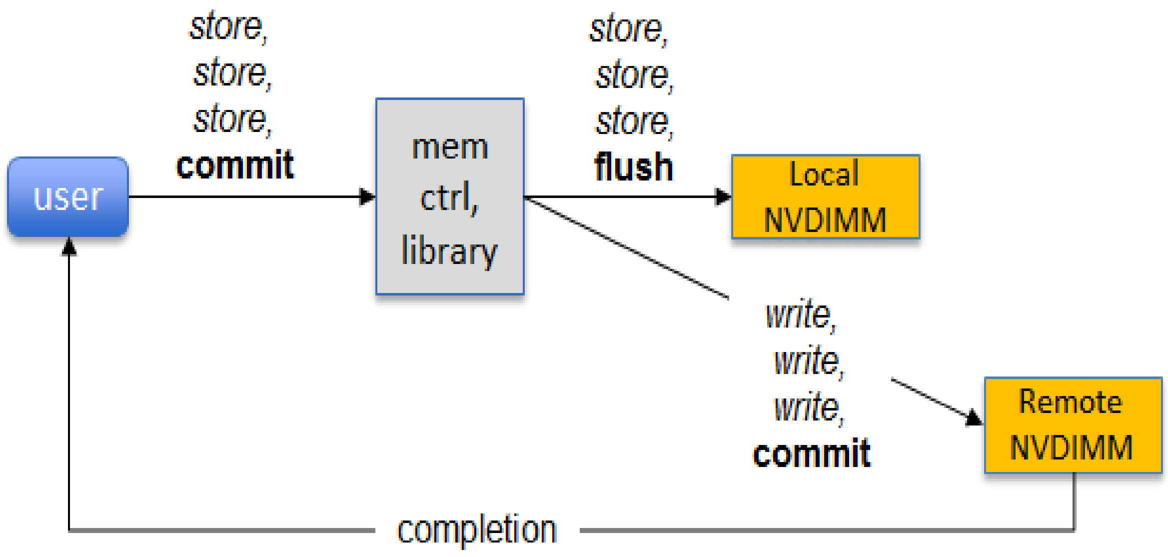
DATA PROTECTION USE CASE



What it looks like

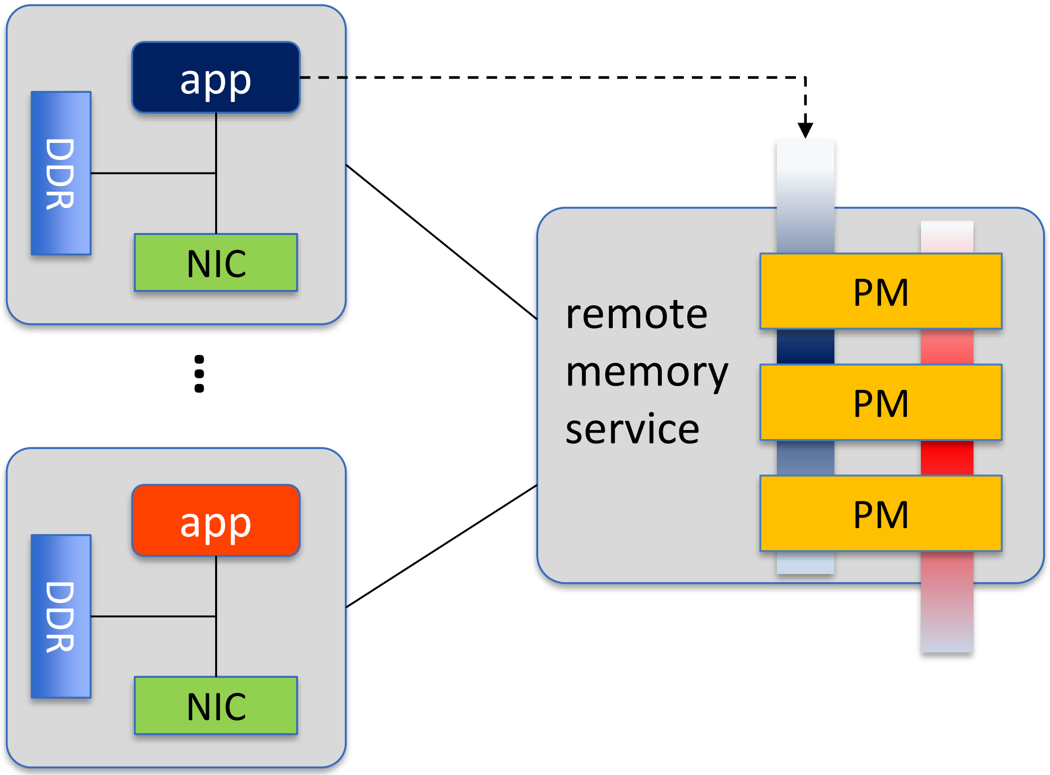
	Persistence	Performance	Capacity
Data Availability	√√√	√√	√
Checkpoint	√√√	√√	√√

How it works

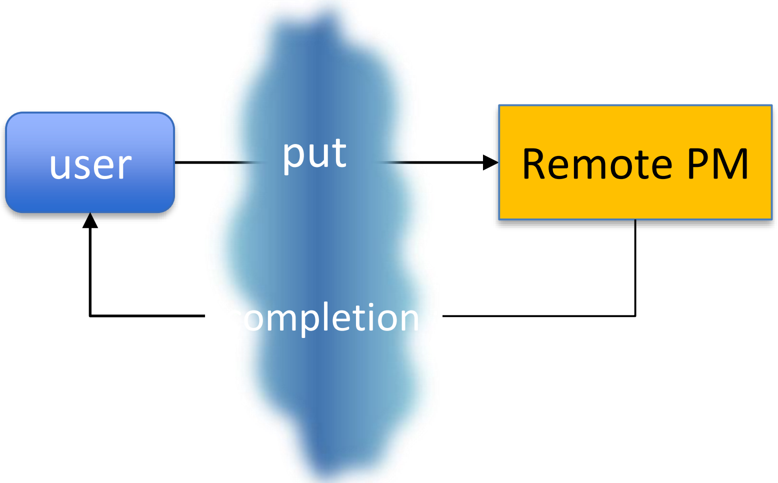


Usage: replicate data that is stored in local PM across a fabric and store it in remote PM

SCALE OUT USE CASE



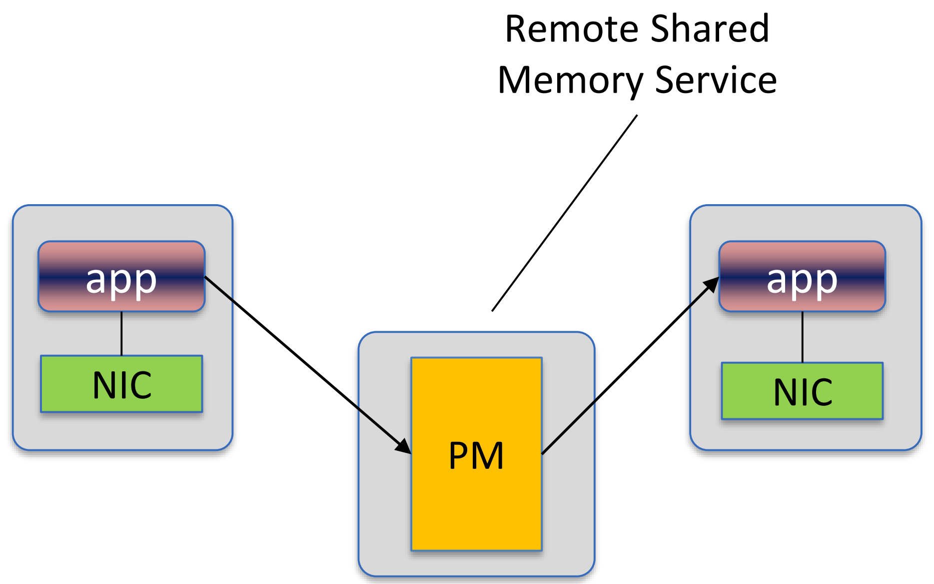
	Persistence	Performance	Capacity
Scale Out	√	√√√	√√√
Disaggregation	√√	√√√	√√√



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

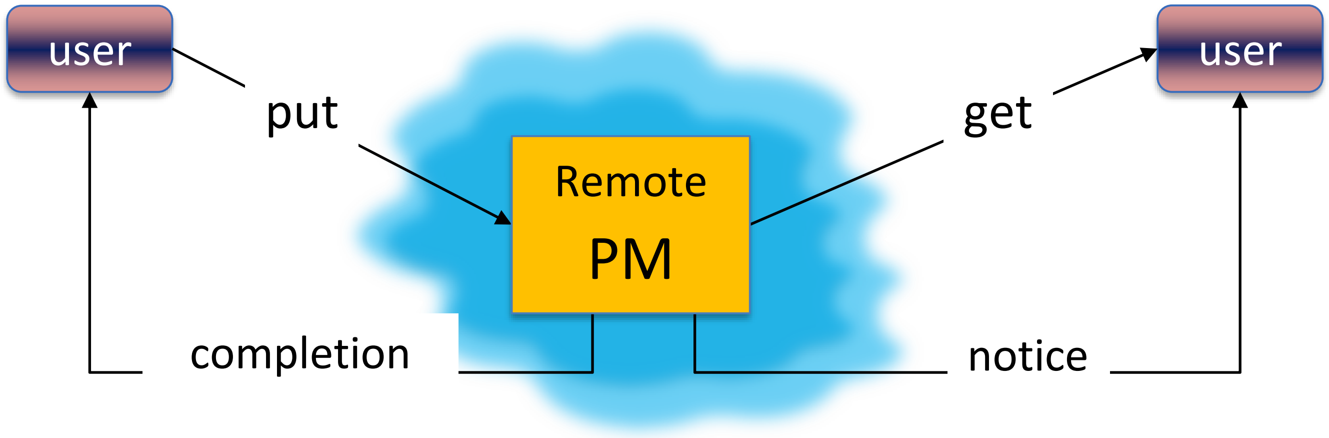
"Scalable Memory"

SHARED DATA USE CASE



	Persistence	Performance	Capacity
Shared Data	√√	√√	√√√

How it works



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

SOME PRELIMINARY OBSERVATIONS

- **Non-persistent use case don’t require flush semantics**
- **RPM is NUMA**
- **APIs for local vs remote PM are likely to be different, because of asynchronicity**
- **Capacity use cases likely have different access patterns than e.g. performance use cases**
  - large reads / writes vs byte level accesses
- **For persistence use cases, some should be ‘automatic’ e.g. Data Protection, others should be ‘on-demand’**
- **Distinguish between the access method that the client sees vs the technology that is implemented on the RPM node**
  - They are very different things
- **Consider the chicken and the egg – it’s hard to predict what will be needed for new application models**
  - PM as an accelerator for existing application models, or
  - PM as an enabler of new application models

NEXT STEPS

1. Commit the foregoing to text, allowing us to dig into the details
2. Begin thinking about what this implies for the API

15th ANNUAL WORKSHOP 2019

THANK YOU