FlashArray//m NVRAM

FlashArray//m and NVRAM; how do these relate to Pure Storage? Flash storage has to balance two competing goals:

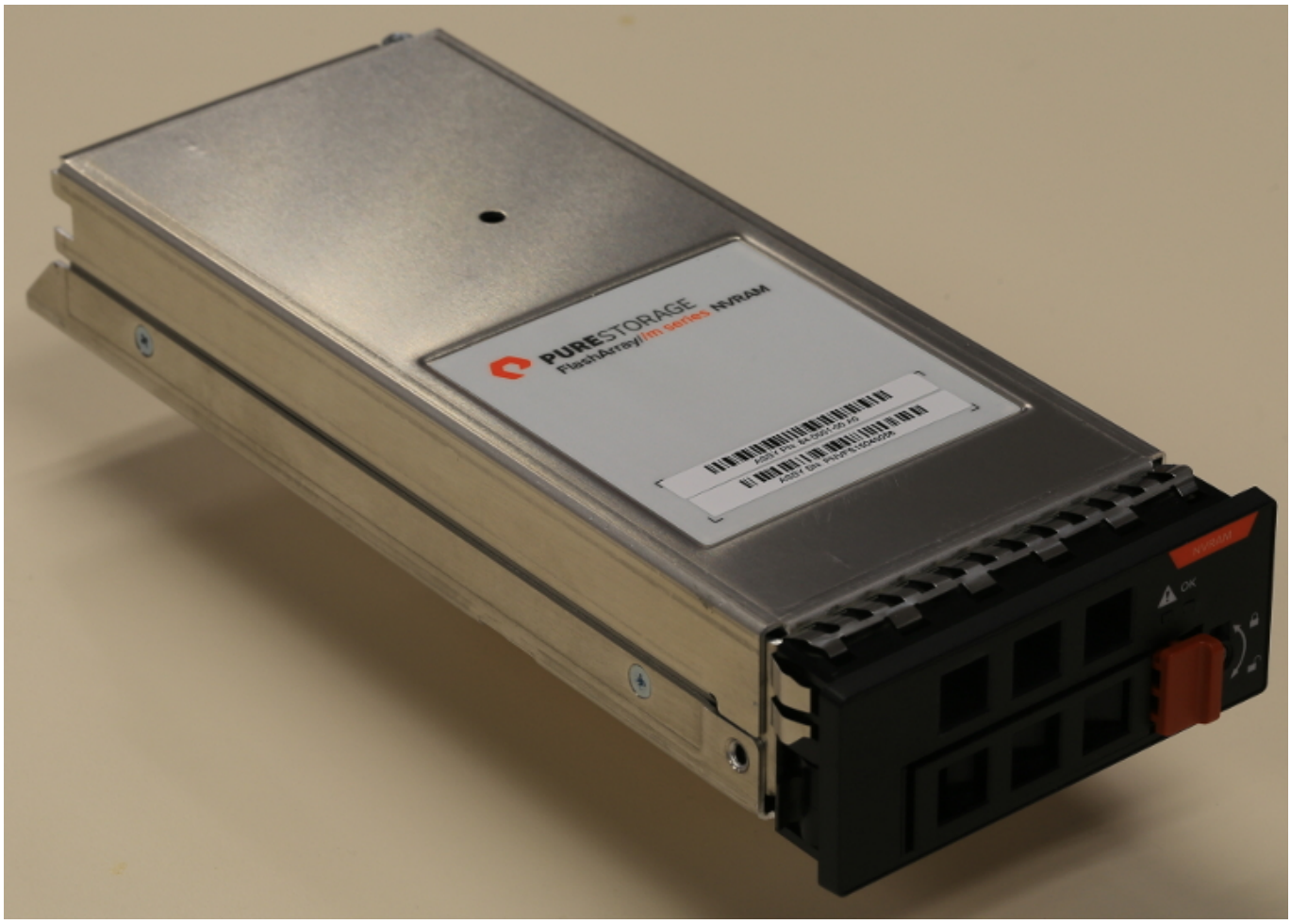
1. We want to acknowledge completion of incoming IO as soon as we can, since low latency is the key to improving application performance. And if your application performance doesn't improve, what's the point of flash storage?
2. We want to wait as long as we can before writing data to flash, so that we can batch up multiple writes and run sophisticated data reduction. We want to write in friendly, large chunks, aligned to the flash's natural erase blocks, and we want to write as little as possible. Using less flash and making it last longer makes the storage system more efficient in power, space and cost.

Most storage systems solve this problem with some form of non-volatile memory, or NVRAM. The array immediately writes incoming data to the NVRAM and acknowledges the IO back to the initiator, since the data is securely persisted. The data is only freed from NVRAM after it has been written to flash, which happens once the array has had time to compress, deduplicate and batch the data up.

One common approach is using battery-backed DRAM in the storage controller, or by using a UPS to provide backup power to the entire storage controller. At Pure, we used a slightly different approach, with flash-based NVRAM modules located in the storage shelf and shared between the redundant controllers. This allows for Pure's "stateless controller" architecture. If a controller is replaced, there is no data in NVRAM or anywhere else to be destaged or mirrored.

Replacing controllers is quick and easy with FlashArray//m, so we knew stateless controllers become even more important. If you can slide a new controller into the chassis in seconds, you don't want a complex data migration process to get in the way. At the same time, we knew that as we continued to improve controller performance, the SAS connection and flash backend of our FA-400 NVRAMs would become a bottleneck.

Enter the FlashArray//m NVRAM, custom hardware designed by Pure and the world's first dual-connected PCI Express storage backed by DRAM:



With dedicated slots in the FlashArray//m chassis, each NVRAM is simultaneously connected to both controllers via PCI Express, with extremely low latency and more than 3 GB/sec of throughput. And the FlashArray//m chassis can hold four of these modules!

Inside the NVRAM, data is written to DDR3 DRAM, which means we have unlimited endurance as well as the performance to keep up with our PCIe interface. Of course, we need the data to stick around, so we use supercapacitors to store energy to write the RAM contents into flash if the power fails. Since we only write to the flash on power failure, endurance is not a concern. Everything, including the DRAM, flash and supercapacitors is included in the NVRAM module, so there are no battery packs, cables, UPSes or any other complexity to deal with.

Finally, the controllers talk to the FlashArray//m NVRAM using NVM Express (or NVMe for short), a standard interface for PCIe-attached storage. NVMe was designed for high performance as well as scalability on multicore systems, and both the NVMe protocol and software stack are slimmer than traditional SCSI, leading to lower latency. (Currently, we only use NVMe for the NVRAM modules, but the FlashArray//m midplane is wired for PCIe, so in the future we'll be able to get the same advantages of NVMe for our flash modules as NVMe SSDs become available.)

All of this sounds great, but how does the FlashArray//m NVRAM module actually perform? I grabbed some real hardware and compared the new NVRAM module to the SAS NVRAM used in FA-400. I used a synthetic load of 100% 24 KB writes (which is representative of the workload of a Pure NVRAM, although these numbers come from driving the NVRAM harder than a full system does):

	SAS NVRAM	//m NVRAM
IO latency	170 µsec	50 µsec
Max IOPS	27K	143K
Max throughput	649 MB/sec	3432 MB/sec

We can be very confident that NVRAM performance won't be a bottleneck for FlashArray//m for a long time.