SSD Architecture for IO Determinism

Tim Canepa
CTO – Stealth Startup

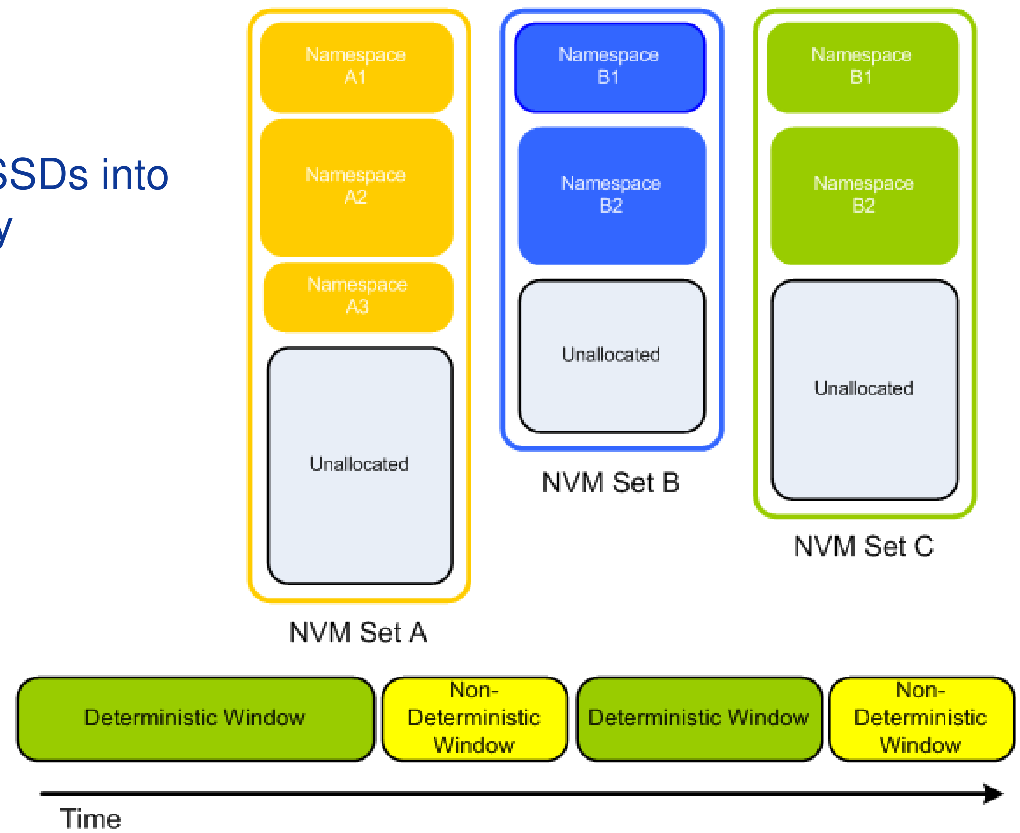
Topics

- Review of IO Determinism
- Architecting SSDs for IO Determinism
- Conclusions

* Disclaimer – I could talk about this topic all day, but gien the limited time, I'll just hit the highlights

Review of IO Determinism

- Goals
  - Allow applications to partition SSDs into regions with predictable latency
- Features
  - Configurable Sets
    - QoS Isolated
    - Attributes (endurance)
    - Multiple Namespaces
  - Predictable Latency Modes
    - Deterministic IO Windows

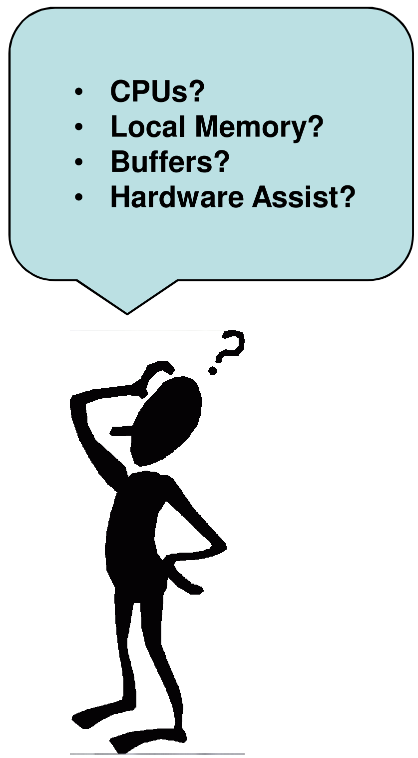


Other Efforts addressing IO Determinism

- NVMe ABO
  - Negotiation between the SSD and Host on when GC and other background operations can be performed
- Open Channel
  - Thin SSD controller with FTL running on the host

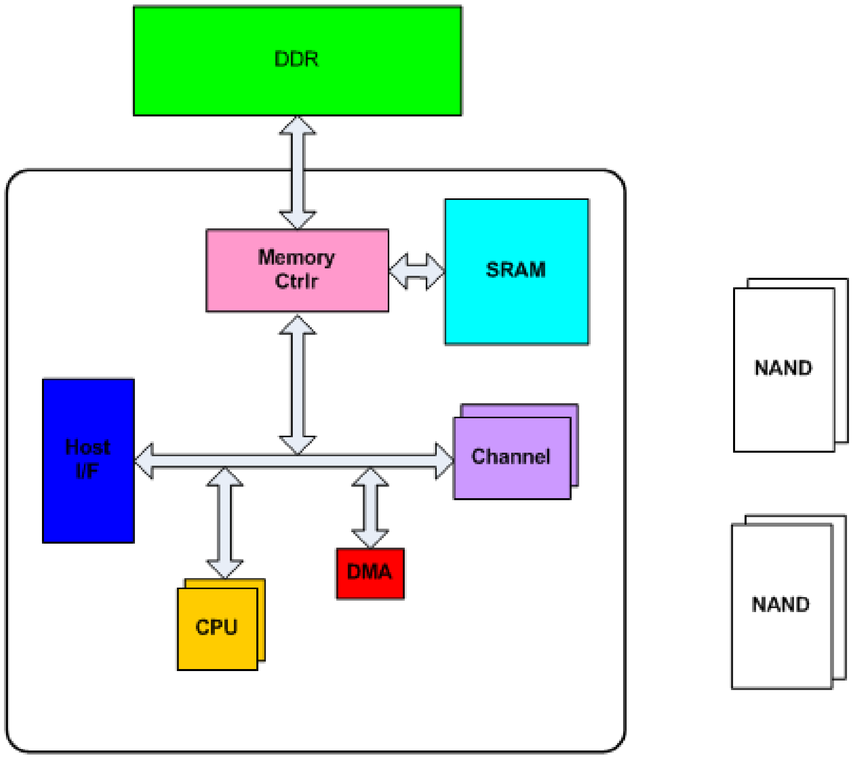
So how does IO Determinism impact SSD Controller and FTL Architecture?

- Separating Flash die into groups is the easy part. But in order to make them truly separate, you need to isolate the other associated controller resources!



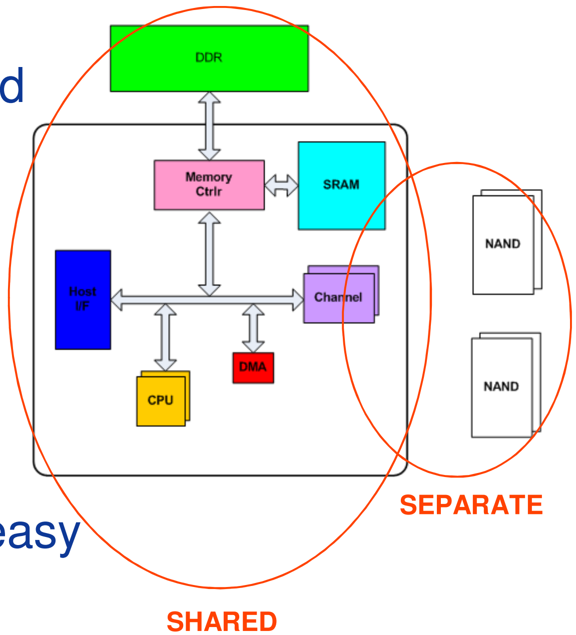
Basic Anatomy of a SSD Controller

- CPUs for
  - Host I/F
  - FTL
  - Flash Ctrl
- Local CPU memory
- Internal Buffer
- DMAs
- HW Assists
- DDR Controller
- ECC Engines
- Host I/F



IO Determinism allows drives to be configured into multiple sub-drives

- And… configurability has its challenges
- As stated earlier, it's not just Flash and IO channels that are being divided up
- Other internal resources and algorithms are impacted
  - CPU
  - Buffering
  - FTL Management
  - Write data processing for multiple streams
  - Data protection strategy
  - Wear leveling and wear out
  - Tracking multiple Write & GC data streams
- And lets not forget managing small die groups isn't easy
  - A sea of blocks and dies becomes a pond

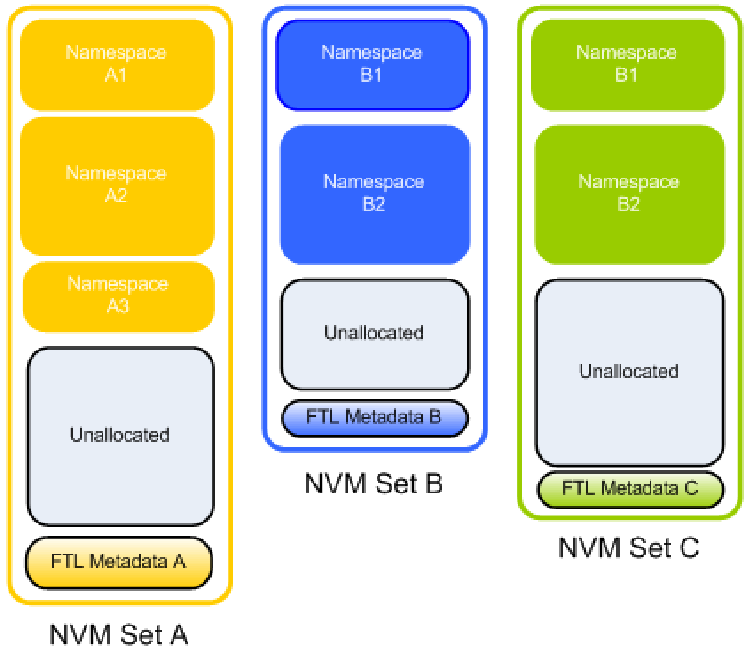


The subtle things in SSDs that create challenges

- Resource consumption on a SSD isn't just limited to die, channel and host bandwidth
- Buffer footprints, bandwidth and tenure vary by workload and WA
- Write data sources often have disparate velocities
- CPU utilization also varies by workload and WA
- Flash programming models need to match host read BW requirements
- Nothing is done in fractions. Whole blocks, whole pages, whole buffers. The more you slice up the drive, the more you lose to fragmentation. Death by fractions, or lack thereof

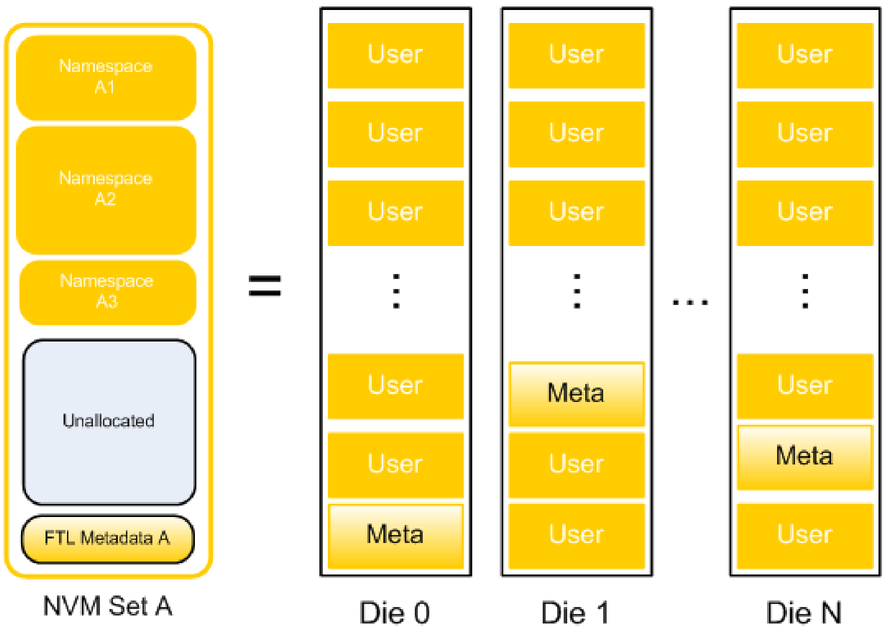
FTL & FTL Metadata Changes

- Gotta Split
  - To enforce isolation, FTL Metadata must reside in the same die group as its data
- Can't afford to store the Metatdata in separate die
  - In a drive with 32 die, using one die for Metadata consume 3.125% of capacity (1/32)



FTL & FTL Metadata changes (continued)

- Think of each FTL journal as a stream
  - Metadata stream can't be combined with user data – must reside in its own blocks
  - Why? Velocity & Garbage Collection disparity between User data and FTL Metadata
  - Like oil and water, they do not mix



Handling Multiple Write Streams

- Each Set is essentially a separate write stream
- How data is striped across die within a stream matters
  - At least for read performance and write accumulation buffers
- Why?
  - Read performance necessitates striping sequential data across die
  - But the larger the stripe, the larger the write accumulation buffer needs to be
  - And, each stream needs one – which can substantially increase write buffer requirements

Composition of write performance is more complicated that you might think

- Why do writes slow down?
  - Simple answer is buffer scarcity
  - No place to put the data means you have to wait
  - But buffers are expensive, consuming BW, Power and Die area
- Sharing write buffers between sets makes things complicated
  - Tenure and velocity disparity between streams impacts determinism
- Outer codes can help, but they are not a panacea
  - Outer codes can be used to recover from program failures, allowing buffers to be freed before programming completes
  - But, it complicates program failure recovery…

Data Protection & Related Flash Utilization

- Blocks fail, dies fail, read errors happen
  - In an enterprise environment, data loss isn't general acceptable unless there's a redundant copy of data
- Outer codes and Sets
  - Traditional Outer code overhead is too high for sets
  - Means no block/die failure protection
  - Stronger ECC (inner codes) may be required to prevent read errors
  - Can still used for FTL Metadata
  - Temporary Outer codes on data still needed for write buffer tenure reduction
  - Weaker Outer codes may be alternative, but don't protect against die failure

Buffer Overhead Increase per Set

- Major contributors
  - Host write data accumulation
  - GC write data accumulation
  - Outer code Die/Program failure protection accumulation
- Minor contributors
  - Statistical variability in GC
  - Per set data structure overhead
- All that buffer management increases CPU overhead
- How much things increase LARGELY depends on striping scheme

Conclusions

- Implementing IO Determinism is more complicated than simply dividing up die and channel resources among applications
- FTL Metadata requires repartitioning
- Buffer requirements can increase substantially
- Data protection schemes may need to change
- Flash overhead (reducing OP) increases
- CPU overhead increases – requiring more processing power