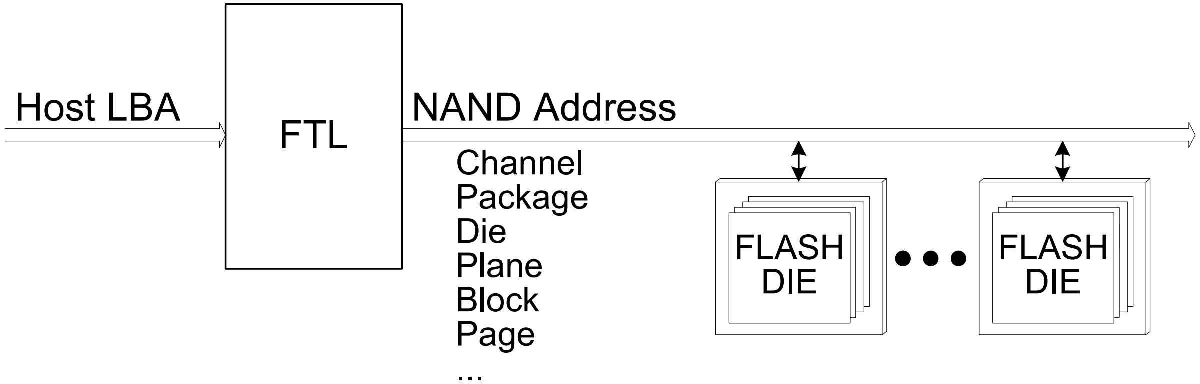
Anatomy of a high-performance, hierarchical FTL Architecture

Tim Canepa
Flash Components Division
Seagate

Introduction

- Flash Translation Layers (FTLs)
  - Provide the **dynamic** mapping from Host Logical Block Addresses (LBAs) to addresses in flash
  - In a desired granularity (e.g., block, page, …)



Some FTL Aspects

- Mapping of LBA (algorithm, granularity, …)
- Recycling (garbage collection)
- Wear-Leveling (Block Picking, Data Placement)
- Bad-Block Handling
- Performance / Overhead
- Checkpoint and Recovery Algorithms

Some FTL performance factors

- Read/Write overhead to use/maintain FTL
  - Flash bandwidth consumption, incl. write amplification
- CPU processing overhead
- Wake-up time (what must be loaded at power-on)
  - How quickly after power-on is user data accessible?
- Unsafe shutdown recovery time
  - How long to restore the FTL after a power fail?

FTL Holy Grail

No questions about swallows or coconuts

- Minimize
  - CPU overhead
  - FTL related WA
  - Data loss
  - Latency
  - Recovery time
- Maximize
  - Concurrent access to FTL structures
- Optimize
  - Garbage collection & wear leveling to maximize performance and drive life
- Maintain data coherency & integrity

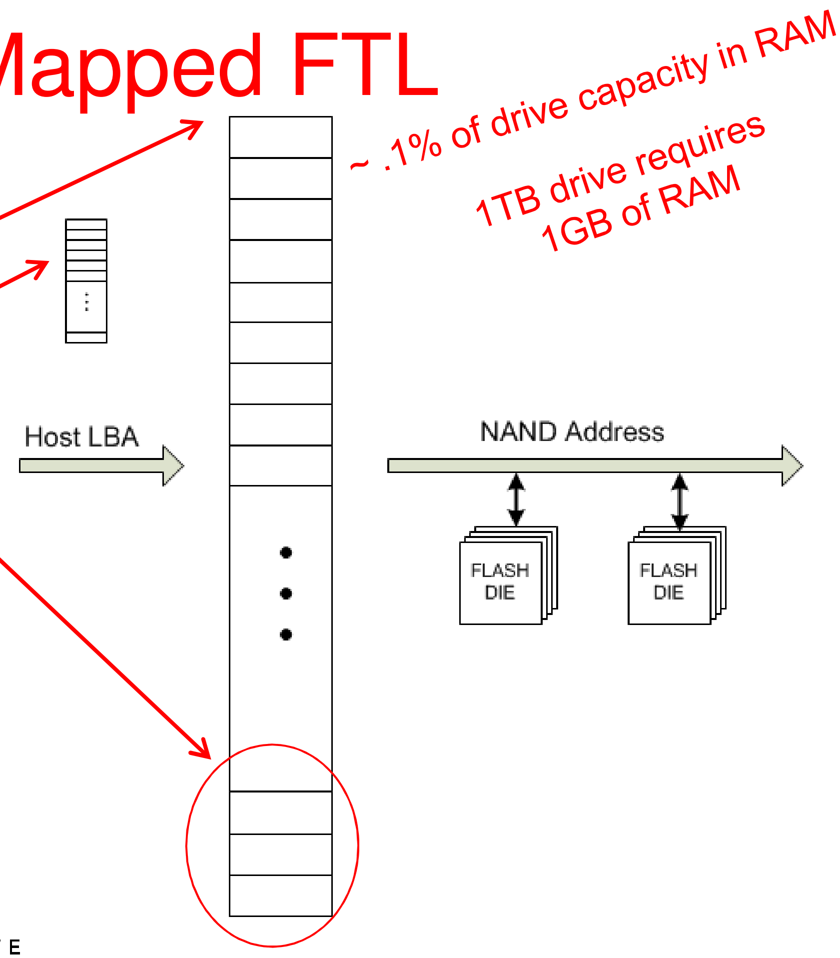
Doing all that in the smallest hardware/resource footprint possible

Typical FTL Components

- Map
  - A means to convert LBAs to NAND addresses
- Meta-data
  - Generally stored in-band with user data, such as storing the LBA with the data itself
  - LBA stored with data acts as a “reverse map” – used for recovery
- Logs/Journals
  - Used in some FTLs to record changes to the map

Traditional Flat-Mapped FTL

- Single level mapping scheme
- One large table contains logical to physical mapping
- Journals to track updates
- Periodic flushing of mapping table
- Data and FTL structures written to flash in same stream
- Various mechanisms employed to maintain coherency between FTL structures and Data



Traditional Flat-Map FTL looks good on paper, but…

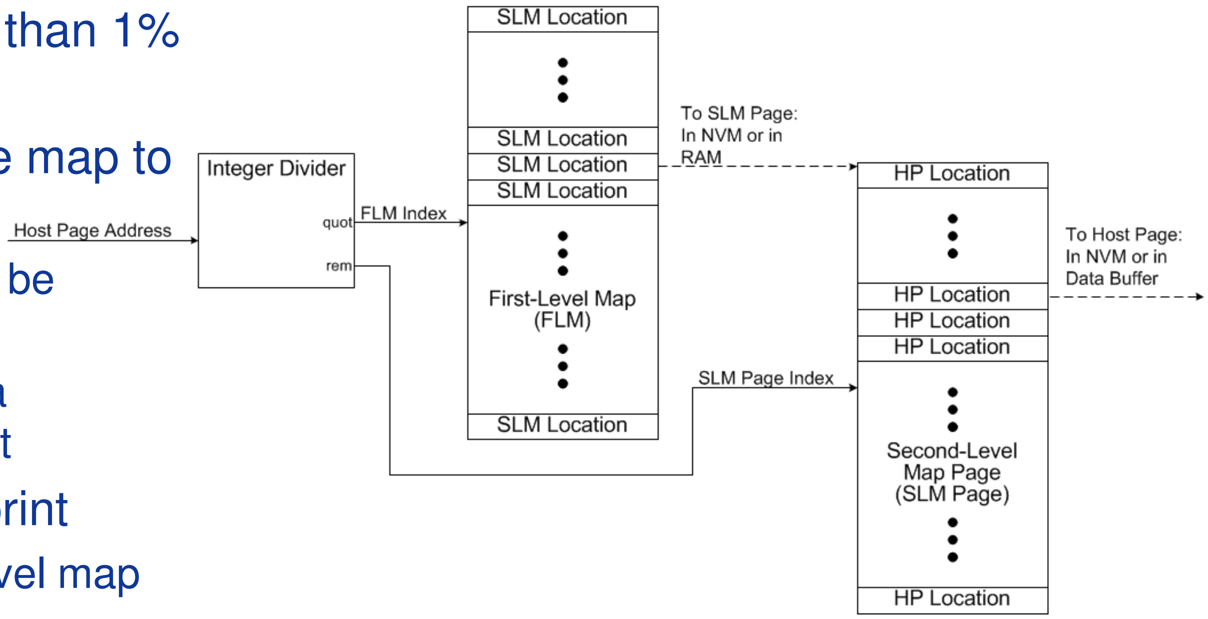
- Needs lots of RAM to hold map
  - RAM requirements scale with drive capacity
- On power up, entire map must be recovered before drive is operational
  - Recovery increases with drive capacity
  - Journals must be replayed before map can be used
  - Supercaps may be necessary to save FTL structures during unexpected power loss
- Map sub-partitioning and various journaling schemes help, but…
  - All exhibit some form of pathological behavior
- **And**… as you start to sub-partition the flat-map, you are in essence making a "poor man's" hierarchical FTL

Making an efficient FTL (both cost & performance)

- Break the relationship between FTL ram requirements and drive capacity
- Optimize how often you have to read/write FTL structures
- Optimize CPU processing overhead
- Fast recover time after power loss means you can't afford to recover ALL your FTL from flash

Hierarchical FTL Anatomy 101: Heart (Map Hierarchy & Behavior)

- Two level map
  - First level map is less than 1% the size of total map.
- Design to allow parts of the map to be fetched from flash
  - Allows portions of map to be cached on chip
  - Allows you to operate in a constrained RAM footprint
- Decide the FTL RAM footprint
  - Any part of the second level map can be absent from RAM



Hierarchical FTL Anatomy 102: Constitution (define your recovery time)

- How long to get back on your feet?
- How much data/journals can you afford to crawl through?
- How much map/journals/data to recover to make the map consistent & operational?

Example

1 TB Drive

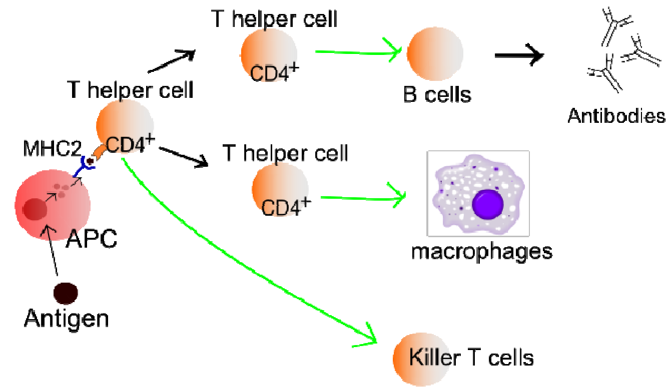
- 32 x 32GB DIE x 8 channels ~ 3GBps
- 200ms recovery time = 600MB of data max + processing time

Need to recover

- 10MB of First Level map
- Some number of data pages/journals to replay against map. Say 10,000 data pages. ~ 40MB
- Another 10,000 SLM updates ~ 10MB

Hierarchical FTL Anatomy 203: Immune system (Define your checkpoint scheme)

- How often do you want to flush the First Level Map?
- How long do you let dirty portions of the map stay in memory before they are written to flash?
- Don't let stuff get too stale
- Striking the right balance keeps recovery time and FTL related write amplification balanced



Hierarchical FTL Anatomy 204: Endurance (Make it fast and efficient)

- Aim small, miss small
  - Make your Map access fast & coherent
- Logging and FTL flushing
  - Minimize log/FTL redundancy
- Garbage collection & Block picking
  - Free space tracking must persist across power failures
  - Data placement decisions make a difference (blue bin vs. yellow bin)

Nothing is free

- The SSD must be designed from the ground up to support a hierarchical FTL
- Free space tracking complexities
- Recycling complexities
- Increase in FTL mapping structure sizes

Benefits of a Hierarchical FTL

- Can live in a variety of memory footprints
  - From small 100s of K to ~ .1% (or less) of drive capacity
- Deterministic recovery time, regardless of capacity
- Deterministic FTL Write Amplification
  - WA for each performance corner is constrained
- Cost / Performance / Data loss tradeoffs become flexible

Thank You!   Questions?

Visit Seagate Booth #505

Learn how Seagate accelerates storage with one of the broadest SSD and Flash portfolios in the market

www.seagate.com/flash