Don’t stack your Log on my Log

Jingpei Yang, Ned Plasson, Greg Gillis,

Nisha Talagala, Swaminathan Sundararaman

Oct 5, 2014

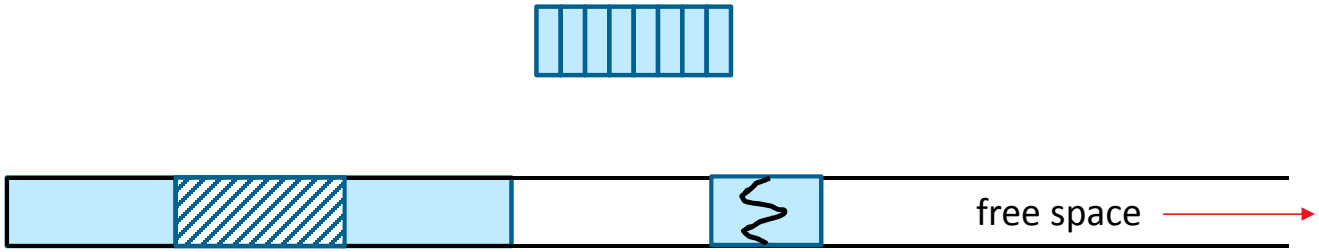
SanDisk®

Outline

- Introduction
- Log-stacking models
- Problems with stacking logs
- Solutions to mitigate log-stacking issues

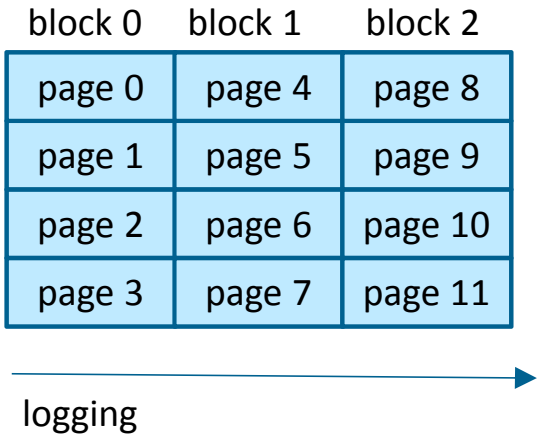
Log-structuring

- Append data at the tail
- Maximized write throughput by aggregating small random writes to large sequential ones
- Easier storage management
  - Free space management
- Fast recovery
  - Good for transactional consistency
  - Low overhead snapshots
- Flexibility on reclaiming space or data chunks
  - Good for database systems that unmodified data could be grouped during GC process



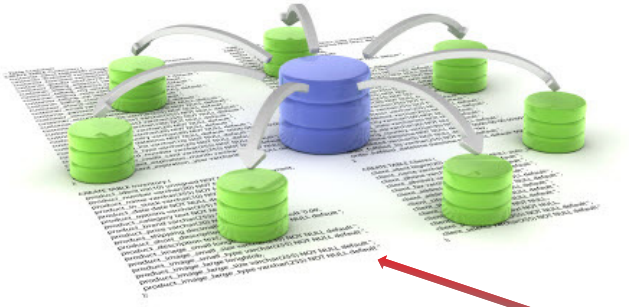
Log-structuring - why on flash?

- Flash does not support in place update
  - Erase is costly
- Asymmetric read/write performance
  - Write is much slower than Read
- Operations are in a unit of page (4KB, 8KB)
  - High overhead for small random I/O
  - e.g. a 512-byte write may consume a 4KB flash page



Log-structure is a better choice for flash memory!

Log-structuring - who?

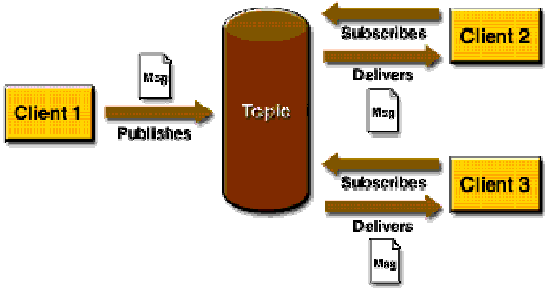


database
e.g. change and access logging records

flash aware file systems, FTL
e.g. data log, metadata log

logging

logging system, pub/sub
everything is logged



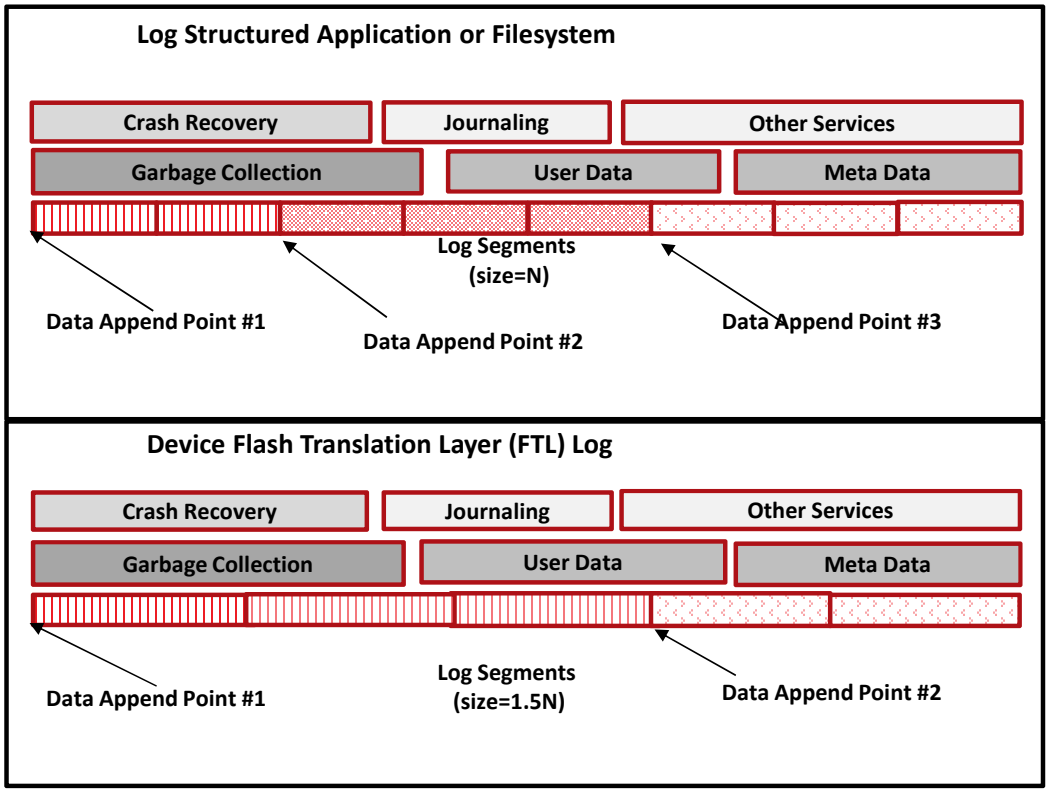
Are more logs better?

- Individual logging is optimized for performance
- Individual logging provides better isolation
- Multiple logs within one application/layer is even better
  - Hot/cold data separation
  - Easy metadata management
- ***Multiple layers of logs?***

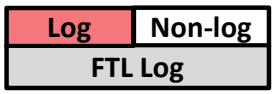
Outline

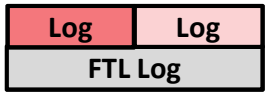
- Introduction
- **Log-stacking models**
- Problems with stacking logs
- Solutions to mitigate log-stacking issues

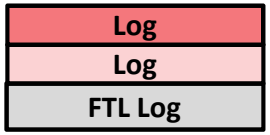
Log-on-log models



Log-on-log models





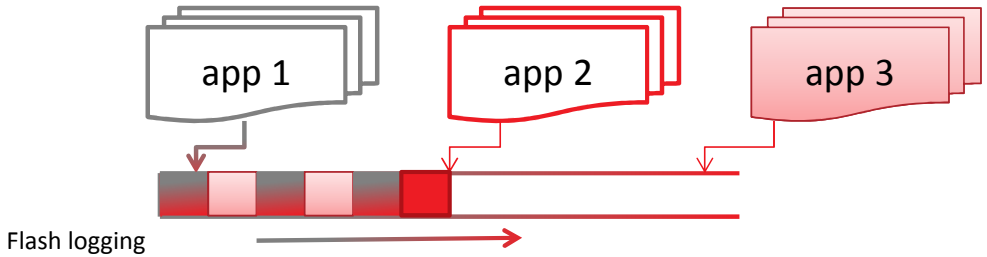


Outline

- Introduction
- Log-stacking models
- **Problems with stacking logs**
- Solutions to mitigate log-stacking issues

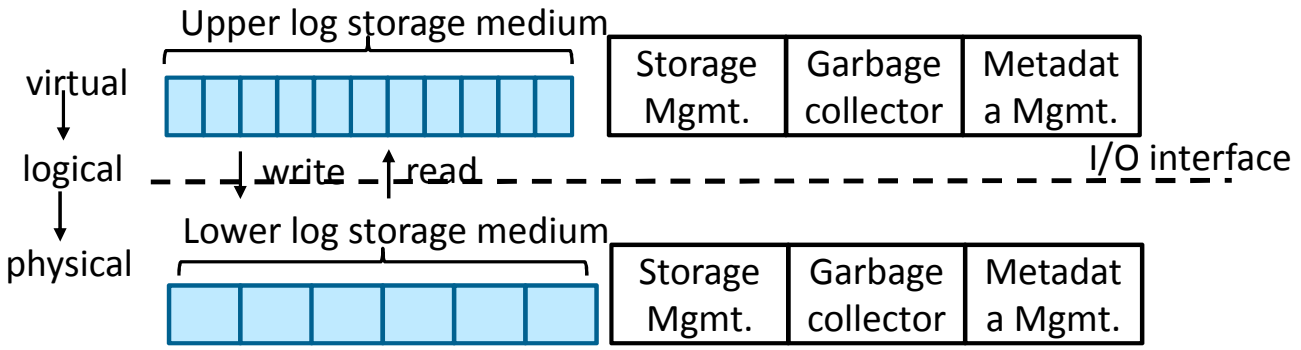
Problems with stacking logs

- Increased write pressure
- Destroyed sequentiality
- Duplicated log capabilities
- Lack of coordination
  - Application logs are not FTL aware
  - FTL log is not application aware



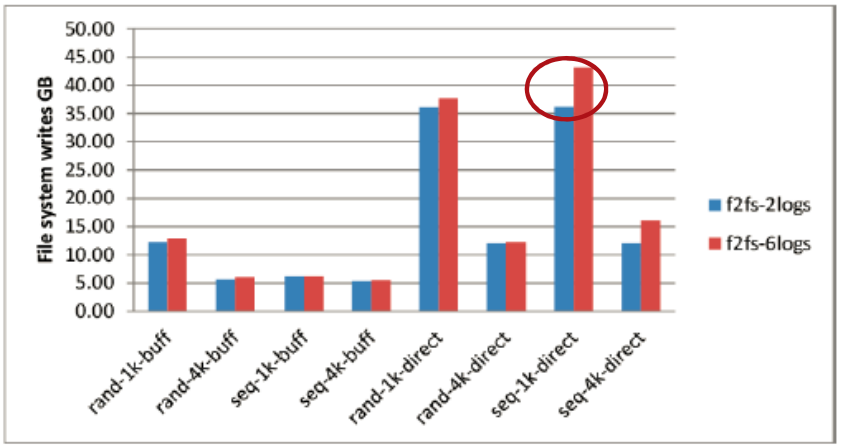
Experimental methodology

- A flash-aware file system – F2FS
  - Up to 6 logs
  - On top of a single log (one append point) SSD
- A log-on-log simulator
  - Mimic a file system to device storage system
  - Data placing, storage management, garbage collector



1 - Metadata footprint increase

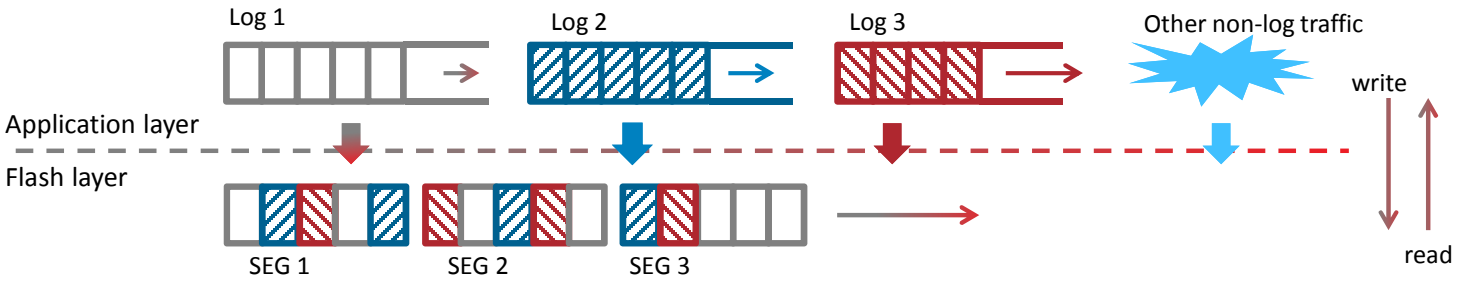
- Each layer/log has its own metadata
- On-media metadata is increased – resulting in more writes to the device
- In-memory metadata is increased – higher memory consumption



Increased metadata results in reduced storage for user data and additional writes which reduces endurance.

2 - Fragmented logs

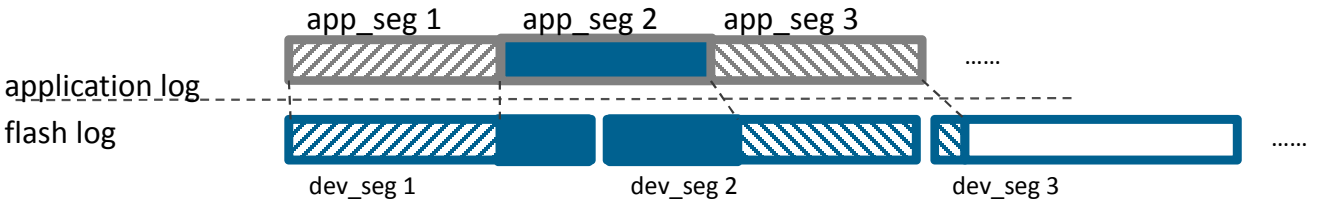
- Mixed workload from logs and other traffic destroys sequentiality
  - Each log writes sequentially, but the device gets mixed workload, most likely to be random
  - Underlying flash-based SSD also writes its own metadata



Multiple logs on a shared device results in random traffic seen by underlying device.

2 - Fragmented logs (cont.)

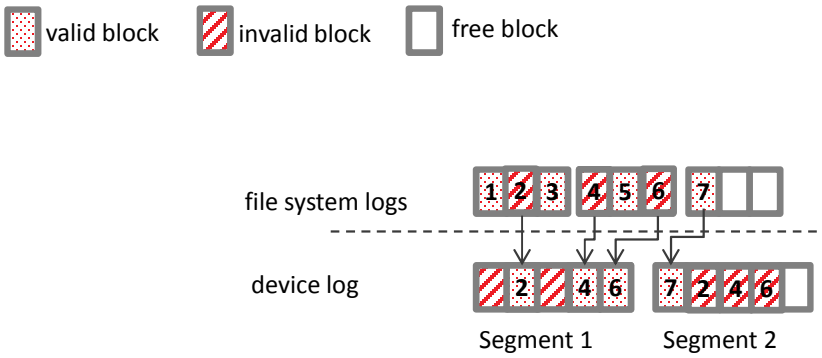
- Unaligned segment size
  - Garbage collecting one upper segment results in data invalidation across multiple segments in device log
  - Matching segment sizes does not prevent data fragmentation



Unaligned segment size results in fragmented device space caused by garbage collection.

3 - Decoupled segment cleaning

- Without TRIM, data has different ‘validation’ view on each log layer
- Data could be moved multiple times across log layers



Uncoordinated garbage collection on each log destroys the sequentiality of data that the application log intends to maintain, and leads to more flash writes.

Even with TRIM support, the sequentiality could hardly be guaranteed in the lower flash log.

Fragmented logs + decoupled cleaning

- A combined effect on the overall WA - TCWA
- Higher log ratio results in higher device WA

60GB random writes, 4K direct IO

random dist	log #	fsys W GB	device W GB	erase cnt	GC data GB	GC WA
zipf:0.8	2	120.55	221.02	370	98.28	1.82
zipf:0.8	6	121.08	267.88	473	144.58	2.19
zipf:1.1	2	122.05	222.78	374	98.52	1.81
zipf:1.1	6	122.53	277.10	493	152.35	2.24
uniform	2	96.32	137.94	188	39.96	1.41
uniform	6	96.42	141.25	195	43.15	1.45

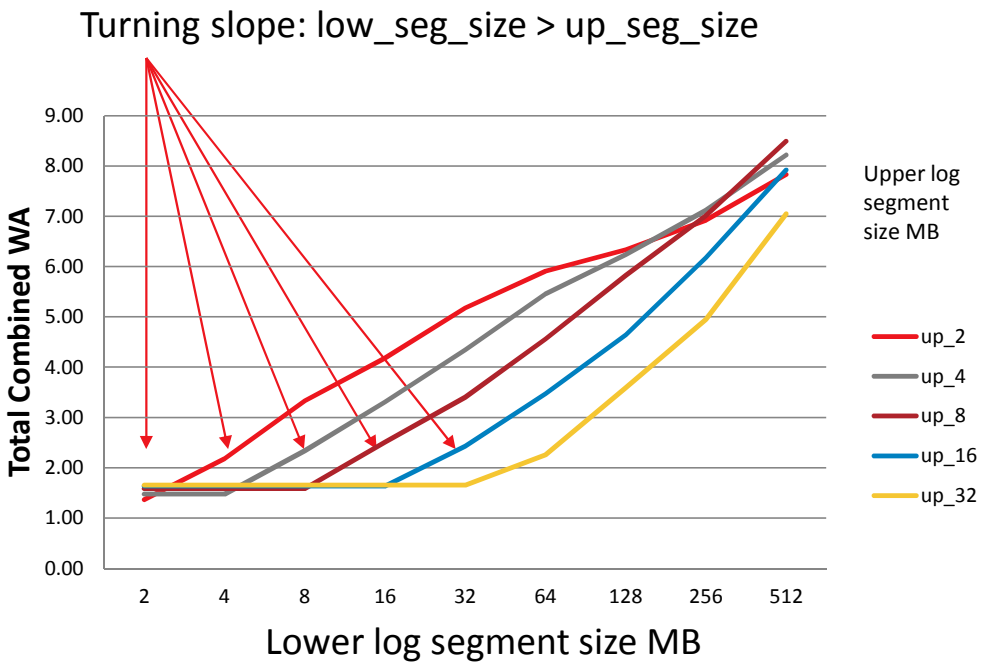
< 1% 4 - 32% 3 - 24%

More upper log results in higher device WA, and hence higher TCWA

Outline

- Introduction
- Log-stacking models
- Problems with stacking logs
- **Solutions to mitigate log-stacking issues**

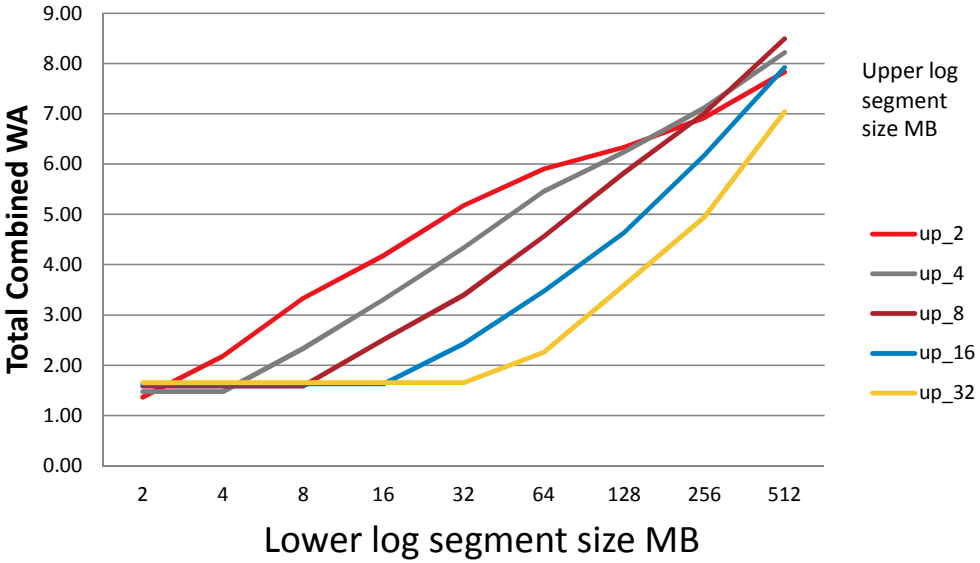
Methods for log coordination



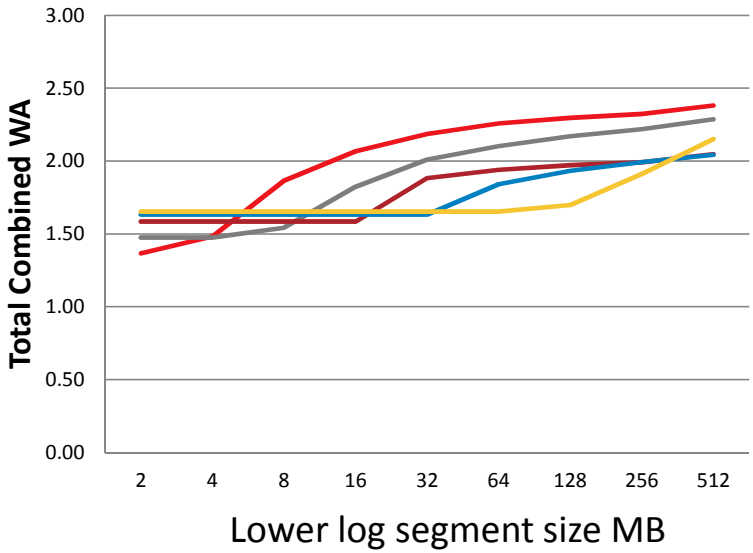
Tpce - like: upper/lower size ratio 90%

- Smaller flash segment size is better in terms of WA
- Total WA increases as lower log segment size increases
- Turning slope when low_seg_size > up_seg_size

Methods for log coordination (cont.)



upper/lower size ratio 90%



upper/lower size ratio 70%

Total WA is a combined effect of capacity ratio, segment size ratio, GC frequency, and etc.

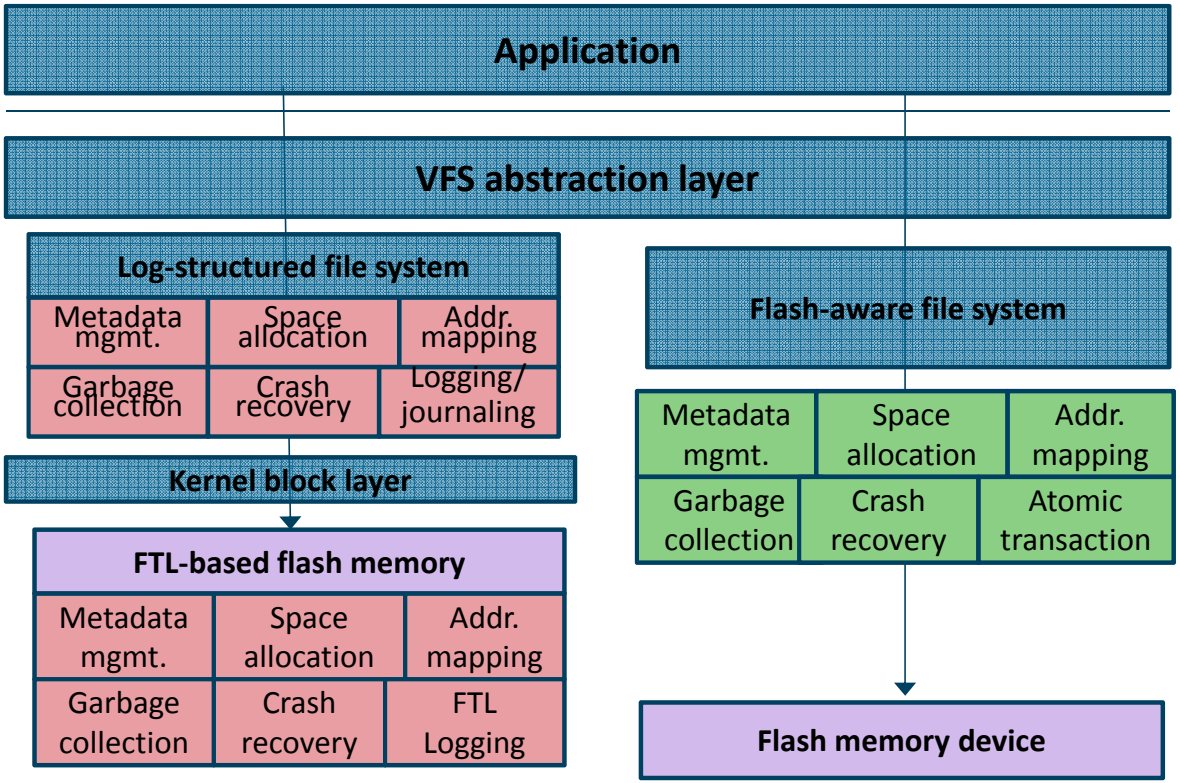
Methods for log coordination (cont.)

- Size coordination to reduce Total Combined WA - TCWA
  - TCWA = (upper log WA) * (lower log WA)
  - Capacity ratio
  - Segment size ratio
  - Tradeoff the space management and GC efficiency by adjusting upper log segment size
- Coordinated GC activity
  - Postpone the device log GC while the upper GC is active
  - Avoid/minimize the same data to be moved multiple times
  - Start multiple upper logs GC simultaneously

Collapsing logs

- Stacking logs is not always optimal for flash-based system
  - Log coordination is less possible with multiple layers be involved
- Two approaches to collapsing logs
  - NVMFS (formerly called DirectFS): sparse space, atomic writes, PTRIM
  - Object-based flash: breaking the standard block interface

Collapsing logs (cont.)



- Eliminate redundant log layers
  - Remove similar log functionalities
- Break the block interface
- Keep the log capabilities in only one layer
  - File system layer with a lightweight flash device design
  - FTL layer with a log-less file system design

Conclusion

- Log structured system is designed to provide high throughput
  - Combining small random requests to large sequential ones
- Multiple logs/append points in different system layers tends to provide better data and space management
  - hot/cold separation
- Stacking logs on another log achieves sub-optimal performance
  - Lack of coordination on each layer mixes the traffic
- Log-on-log can perform better with improved coordination
  - Size coordination, GC coordination
  - File system support
- Collapsing logs – breaking the block interface
  - NVMFS: sparse space, atomic writes, PTRIM
  - Object-based flash storage

Thank You

SanDisk®