

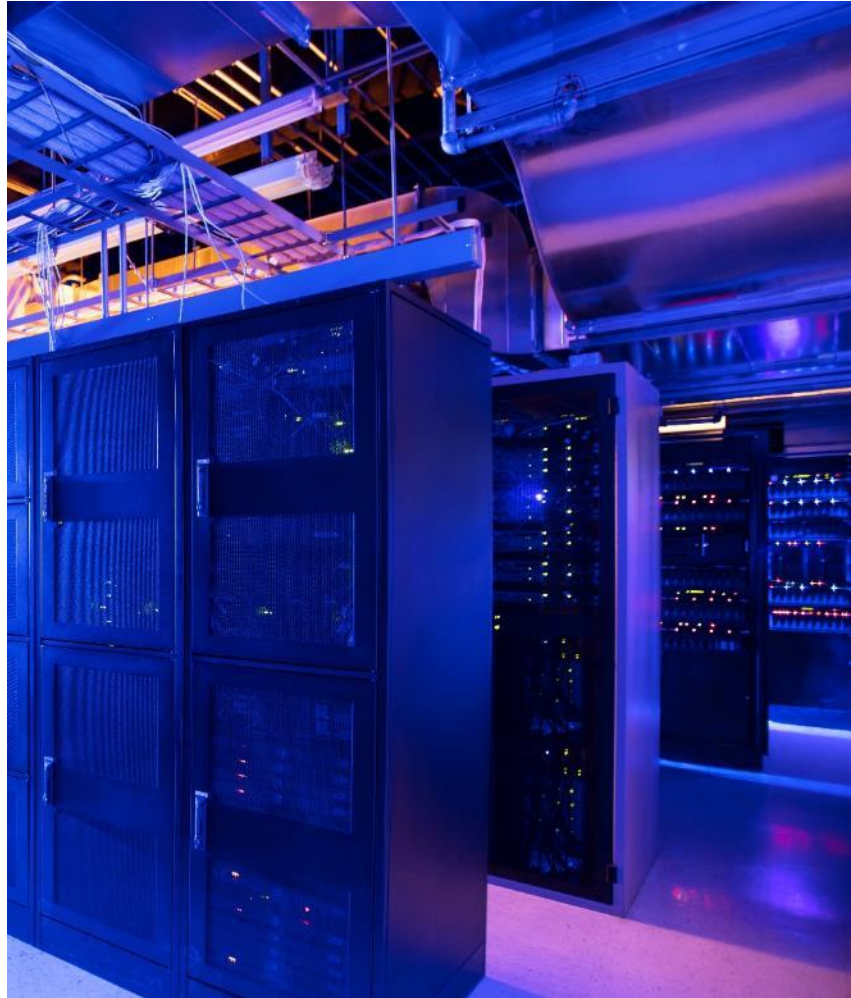


SQUEEZING COMPRESSION INTO SPDK

Paul Luse, Jim Harris

AGENDA

- High Level Architecture
- DPDK Library Overview
- Crypto Bdev Module
- Compression Bdev Module
- Introducing “reduce”



HIGH LEVEL ARCHITECTURE

Storage
Protocols



Storage
Services



Drivers



In Progress

DPDK LIBRARIES

Core and feature libs

Core libraries

Core functions such as memory management, software rings, timers, bus/device mgmt, etc.

Packet classification

Software libraries for hash/exact match, LPM, ACL etc.

Accelerated SW libraries

Common functions such as IP fragmentation, reassembly, reordering etc.

Stats

Libraries for collecting and reporting statistics.

QoS

Libraries for QoS scheduling and metering /policing

Packet Framework

Libraries for creating complex pipelines in software.

Device APIs

ETHDEV

CRYPTODEV

EVENTDEV

SECURITY

COMPRESSDEV

BBDEV

Device PMDs

PMDs for physical and virtual Ethernet devices

PMDs for HW and SW crypto accelerators

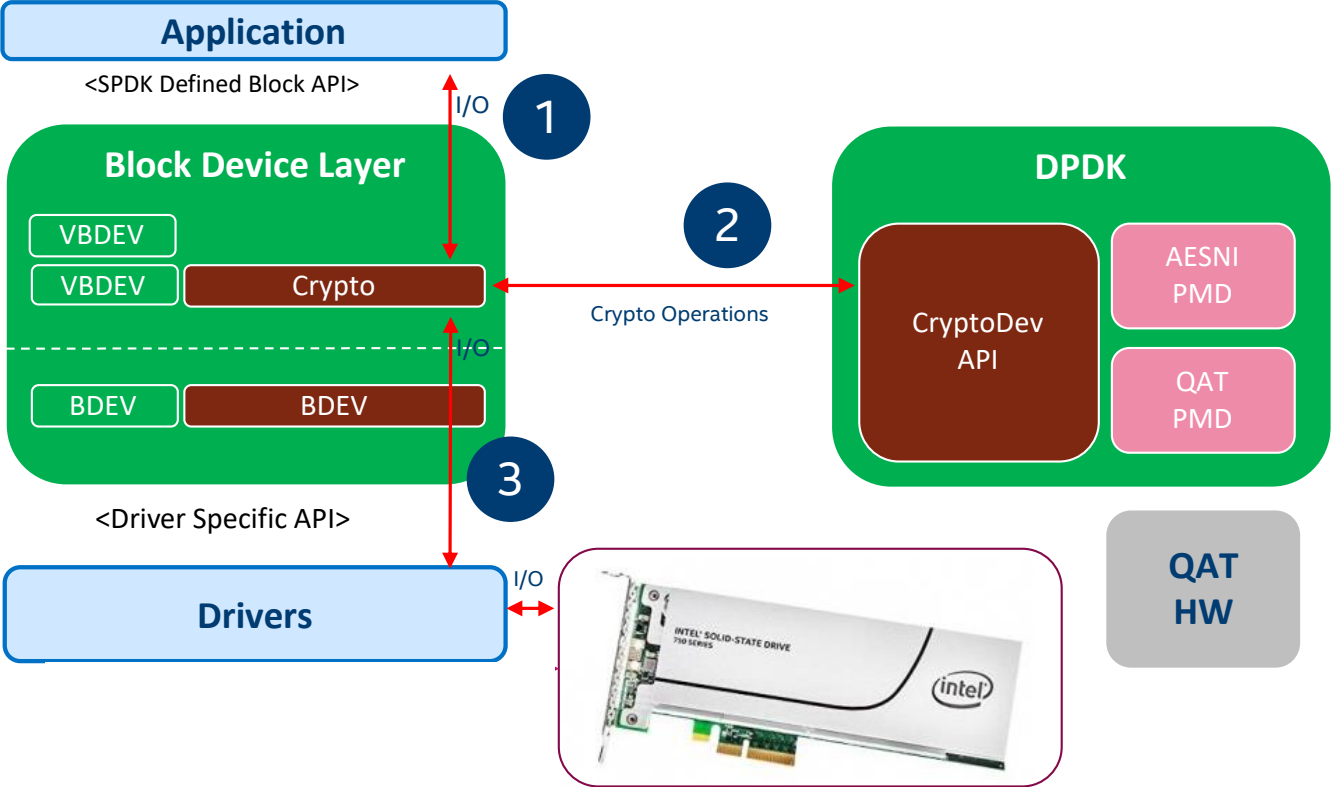
Event-driven PMDs

Hardware acceleration APIs for security protocols

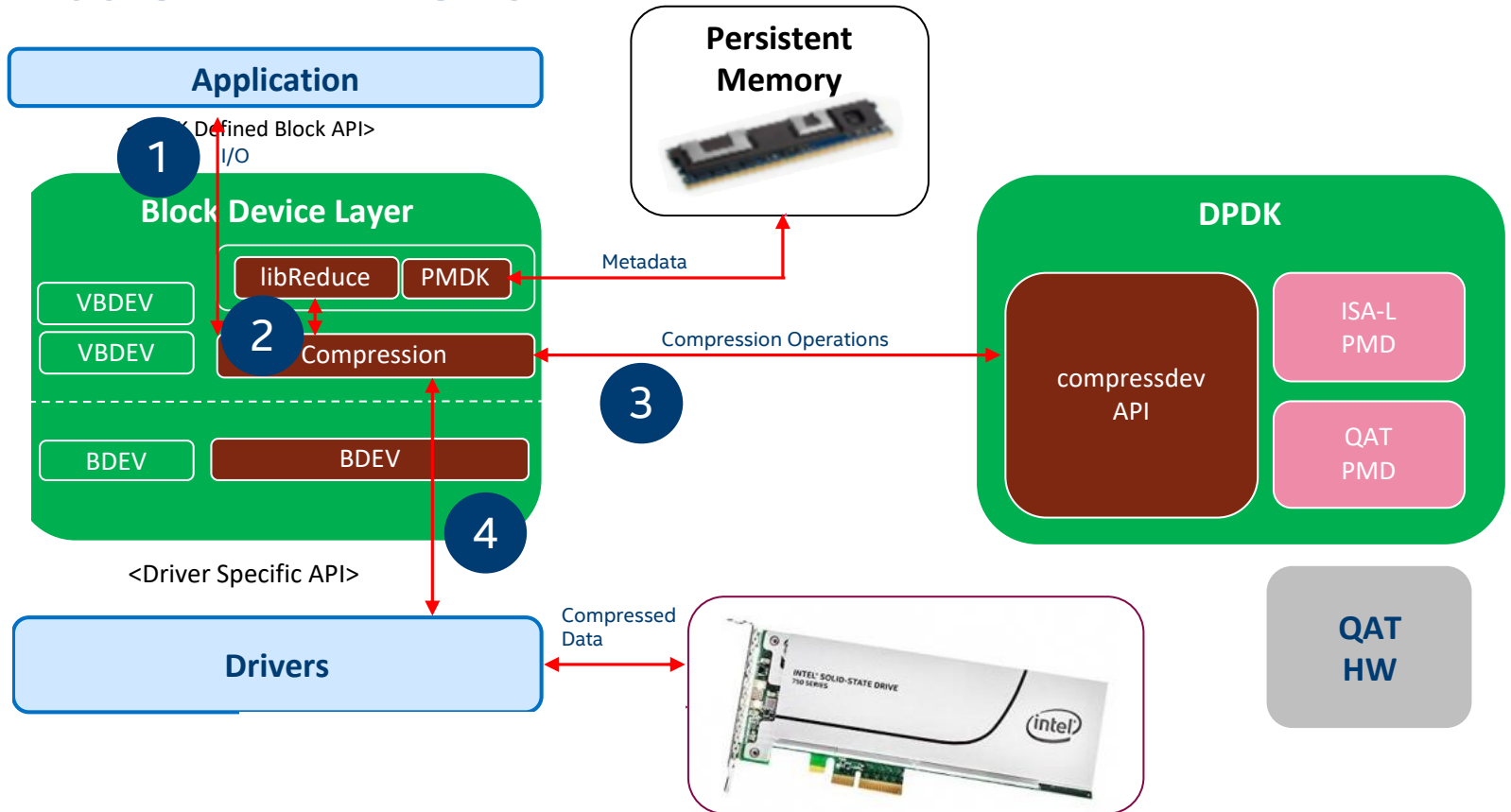
PMDs for HW and SW compression accelerators

PMDs for HW and SW wireless accelerators

CRYPTO BDEV MODULE



COMPRESSION BDEV MODULE



LIBREDUCE OVERVIEW

Block device for backing I/O units

- Typically thin-provisioned SPDK logical volume

Persistent memory file for mapping metadata

- Uses PMDK directly for persistent memory access

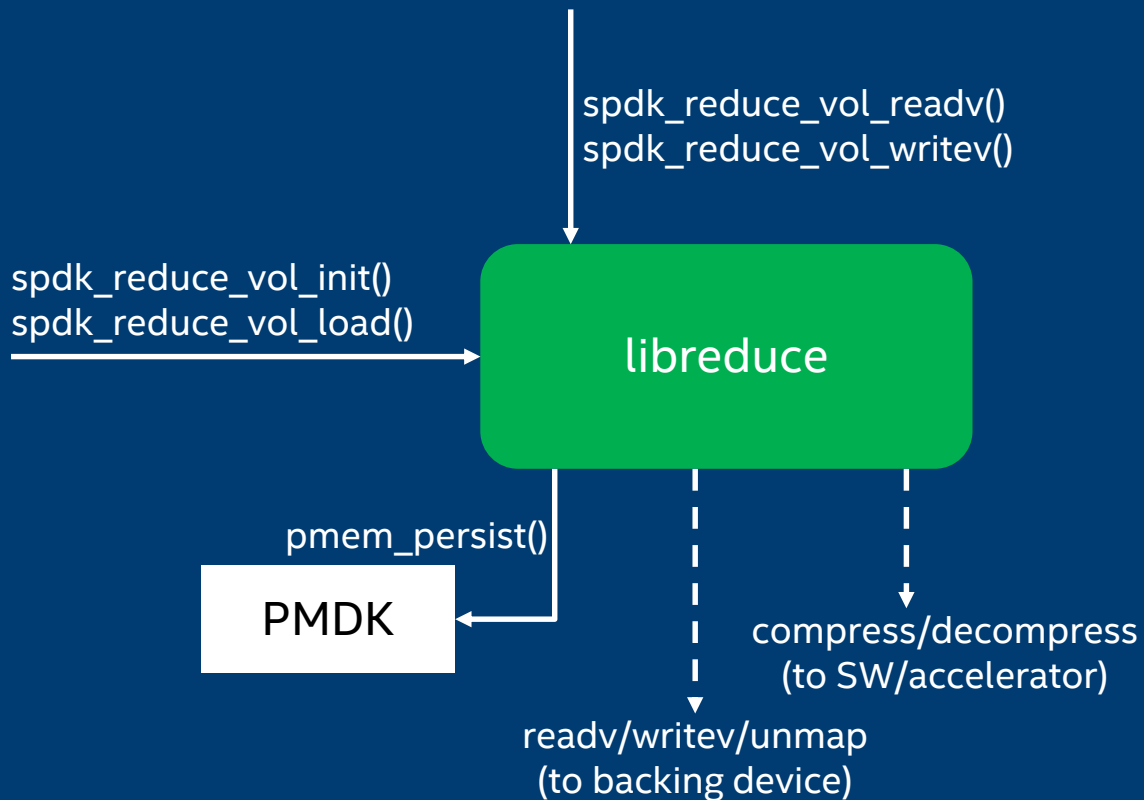
Metadata on block device

- Libreduce parameters
- Path to persistent memory file

Metadata algorithm only!

- Uses standard compression algorithms

INTEGRATION



Independent from SPDK framework and bdev layer

Caller ensures I/Os do not cross chunk boundary

Single-threaded (per compression volume)

LAYOUTS

Persistent Memory

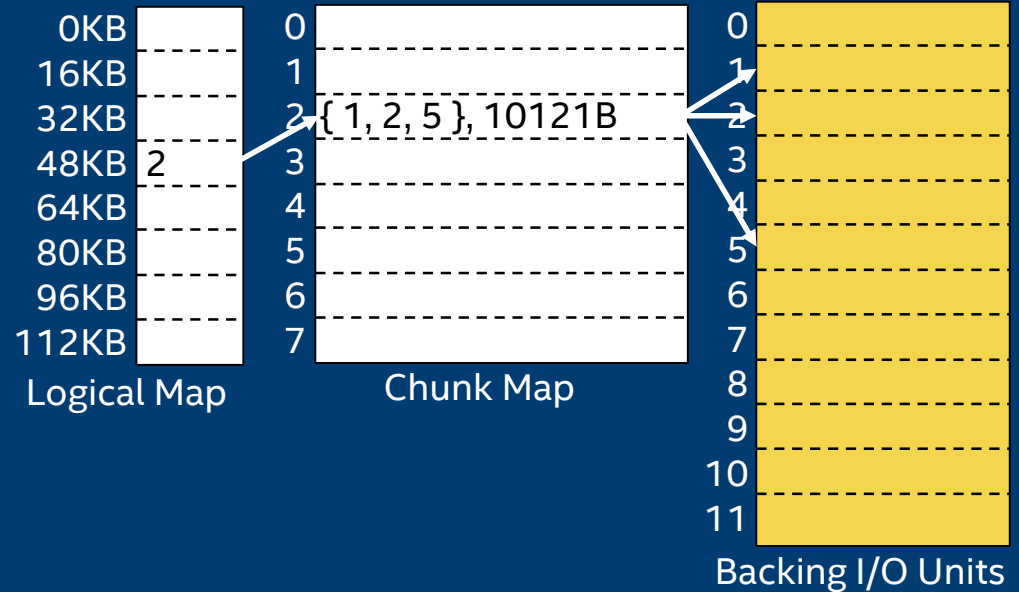
SSD

Backing Device

- Split into I/O units

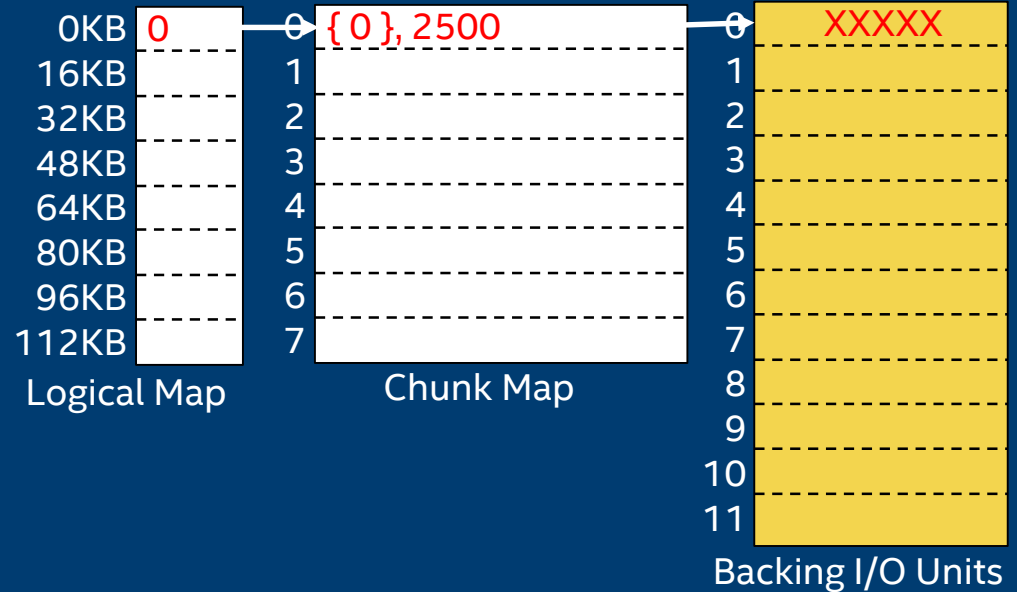
Persistent Memory File

- Metadata based on chunks
- Chunk map contains chunk entries
- Chunk entry maps a logical chunk to its I/O units on disk
- Logical map contains logical map entries
- Logical map entries map a logical offset to its chunk entry



WRITE 4KB AT OFFSET 0KB

- Logical Map: Lookup 0KB => empty
- Allocate chunk entry in chunk map => 0
- Compress chunk data
 - 4KB user data + 12KB zeroes
 - Compresses to 2500 bytes
- Allocate 1 backing I/O unit => 0
- Write compressed data to SSD
- Write and persist chunk entry
- Write and persist logical map entry



WRITE 16KB AT OFFSET 64KB

Logical Map: Lookup 16KB => empty

Allocate chunk entry in chunk map => 1

Compress chunk data

- 16KB user data
- Compresses to 14000 bytes

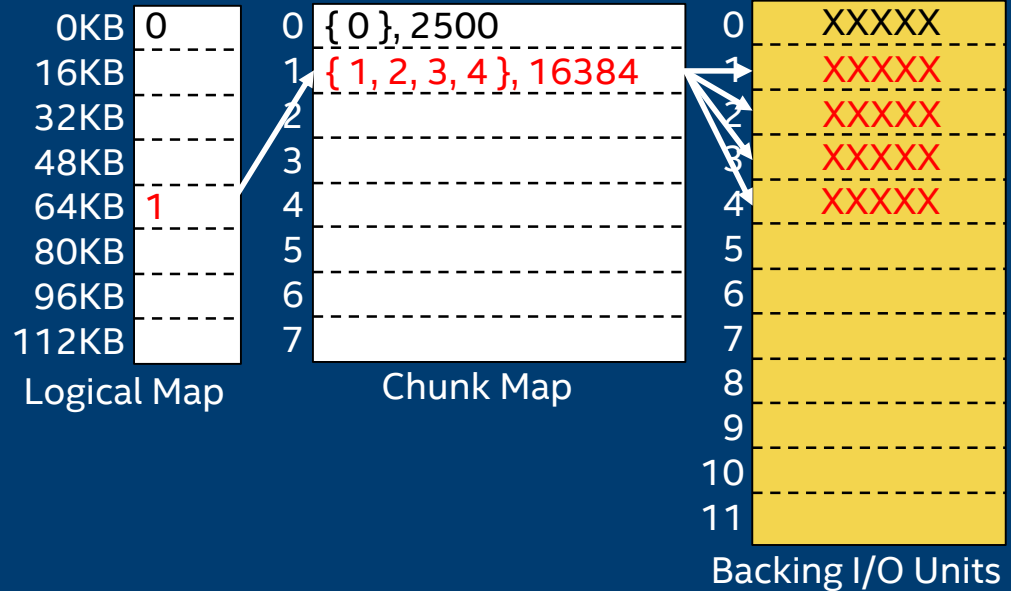
Allocate 4 backing I/O unit => 1, 2, 3, 4

Write uncompressed data to SSD

- 14000 bytes requires 4 4KB I/O units

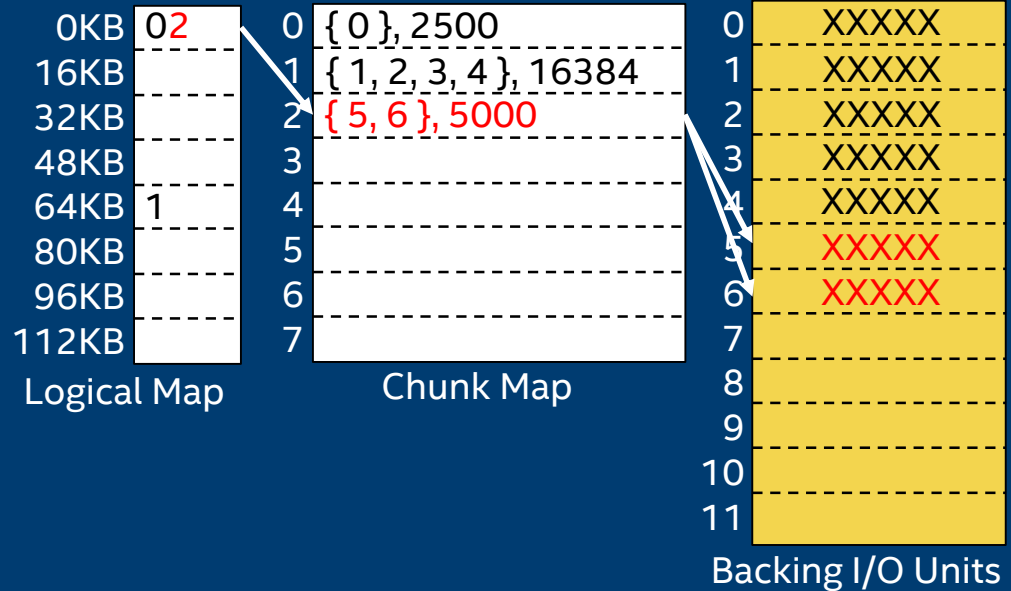
Write and persist chunk entry

Write and persist logical map entry



WRITE 4KB AT OFFSET 4KB

- Logical Map: Lookup 0KB => 0
- Read I/O unit 0
- Decompress 2500B => 16KB
- Merge incoming 4KB
- Allocate chunk entry in chunk map => 2
- Compress chunk data => 5000B
- Allocate 2 backing I/O unit => 5, 6
- Write compressed data to SSD
- Write and persist chunk entry
- Write and persist logical map entry
- Release old chunk entry and I/O units



TRIM 16KB AT OFFSET 64KB

Logical Map: Lookup 64KB => 1

Clear and persist logical map entry

Release old chunk entry and I/O units

0KB	2
16KB	
32KB	
48KB	
64KB	1
80KB	
96KB	
112KB	

Logical Map

0	
1	{ 1, 2, 3, 4 }, 16384
2	{ 5, 6 }, 5000
3	
4	
5	
6	
7	

Chunk Map

0	
1	XXXXXX
2	XXXXXX
3	XXXXXX
4	XXXXXX
5	XXXXXX
6	XXXXXX
7	
8	
9	
10	
11	

Backing I/O Units

READ 4KB AT OFFSET 4KB

Persistent Memory

SSD

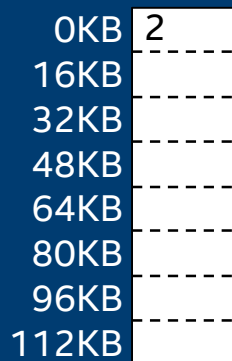
Logical Map: Lookup 0KB => 2

Read I/O units 5 and 6

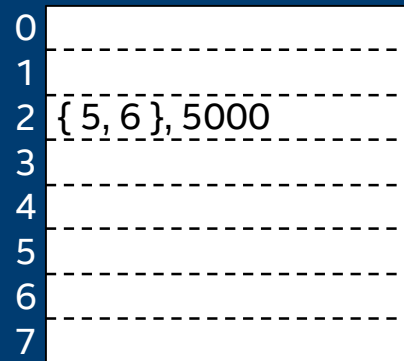
Decompress 5000B => 16KB

Target user buffer for 4KB

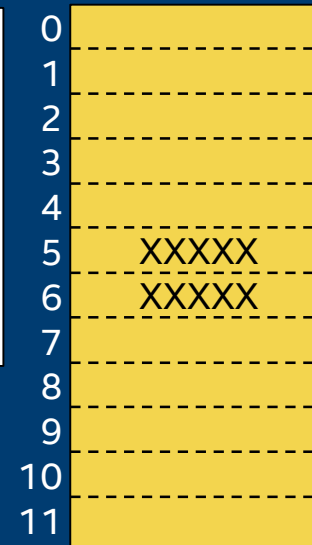
Bit bucket for remaining 12KB



Logical Map



Chunk Map



Backing I/O Units

NEXT STEPS

Master branch has working init/load and read/write path

- Tested with PMDK backed by block device (not persistent memory)

`spdk_reduce_vol_unmap`

- With sub-chunk allocation masks

I/O path optimizations

Additional on-disk metadata parameters (i.e. compression algorithm)

