SDC
STORAGE DEVELOPER CONFERENCE
SNIA ■ SANTA CLARA, 2017

SPDK Blobstore: A Look Inside the NVM Optimized Allocator

Paul Luse, Principal Engineer, Intel

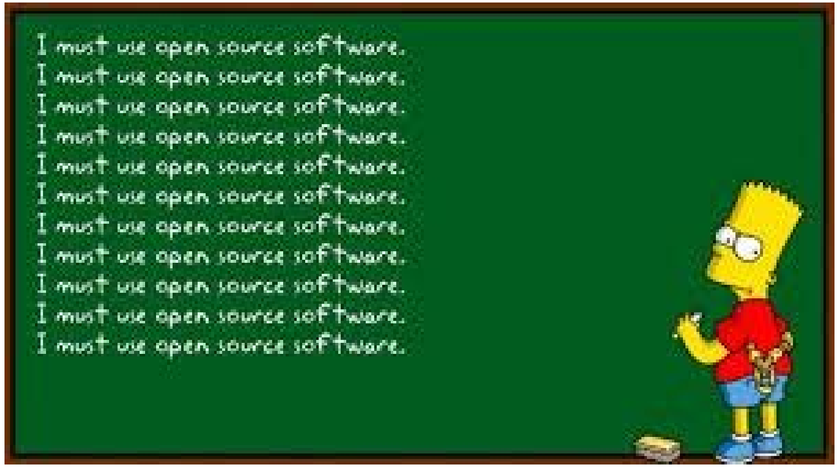
Vishal Verma, Performance Engineer, Intel

Outline

- Storage Performance Development Kit
  - What, Why, How?
- Blobstore Overview
  - Motivation, Design
- A Blobstore Example
  - SPDK Key Elements, Hello Blob Walkthrough
- RocksDB Performance
- Summary

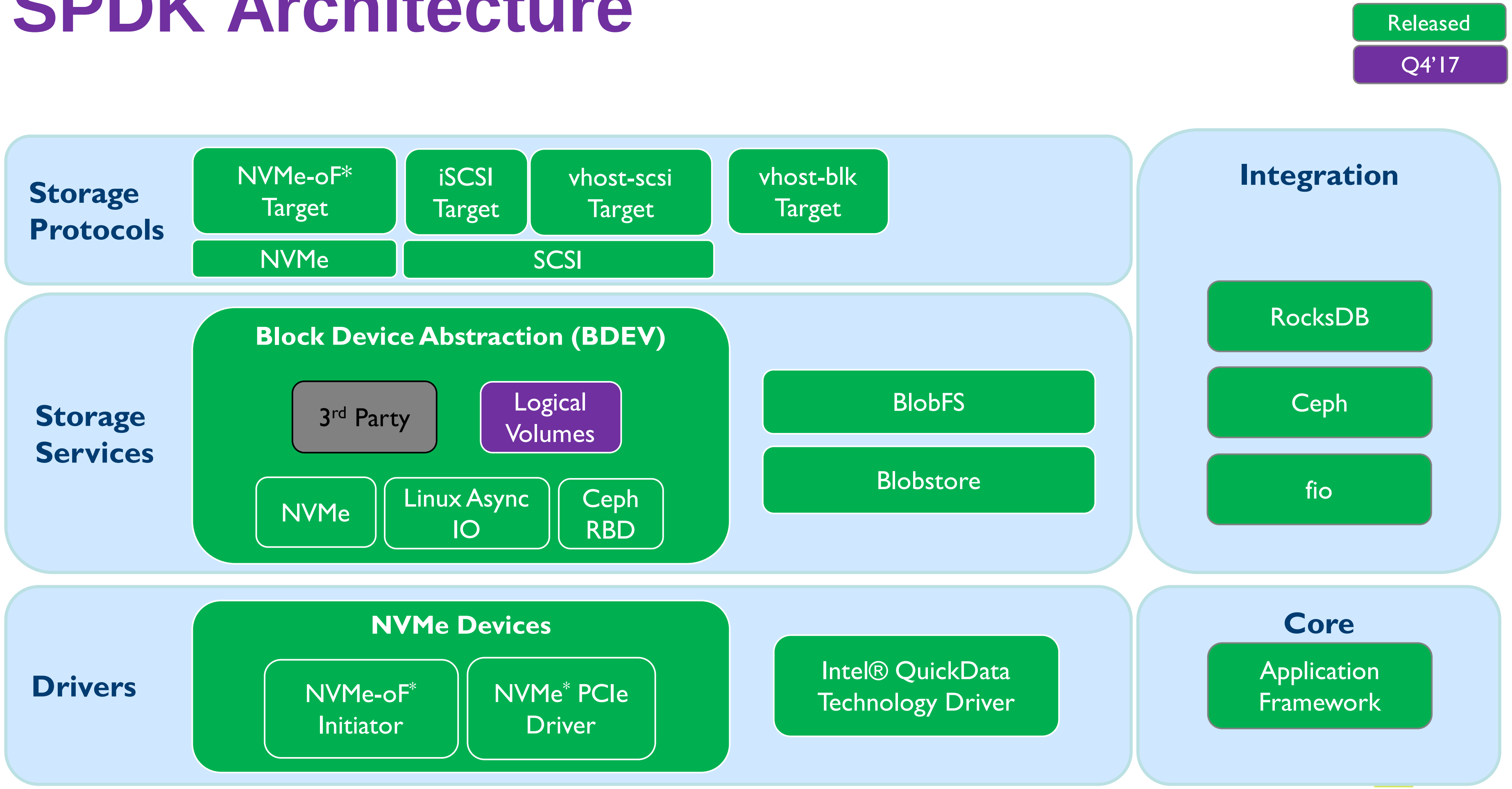
What?
Storage Performance Development Kit

- Software Building Blocks
- Open Source
- BSD Licensed
- Userspace and Polled Mode


I must use open source software.
I must use open source software.
I must use open source software.
I must use open source software.
I must use open source software.
I must use open source software.
I must use open source software.
I must use open source software.
I must use open source software.
I must use open source software.
I must use open source software.

http://spdk.io

SPDK Architecture



Why? Efficiency & Performance

- Up to 10X MORE IOPS/core for NVMeoF* vs Linux kernel
- Up to 8X MORE IOPS/core for NVMe vs Linux kernel
- Up to 50% BETTER tail latency for some RocksDB workloads
- More EFFICIENT use of development resources

* Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance

How? SPDK Community

- **Github :** https://github.com/spdk/spdk
- **Trello** : https://trello.com/spdk
- **GerritHub** : https://review.gerrithub.io/#/q/project:spdk/spdk+status:open
- **IRC:** https://freenode.net/ we're on #spdk
- **Home Page:** http://www.spdk.io/

1st SPDK Hackathon!! Nov 6-8 2017, Phoenix

Blobstore - Motivation

- SPDK came on the scene and enabled applications that consumed block to realize incredible gains
- But what about applications that don’t consume blocks?

Blobstore Design – Design Goals

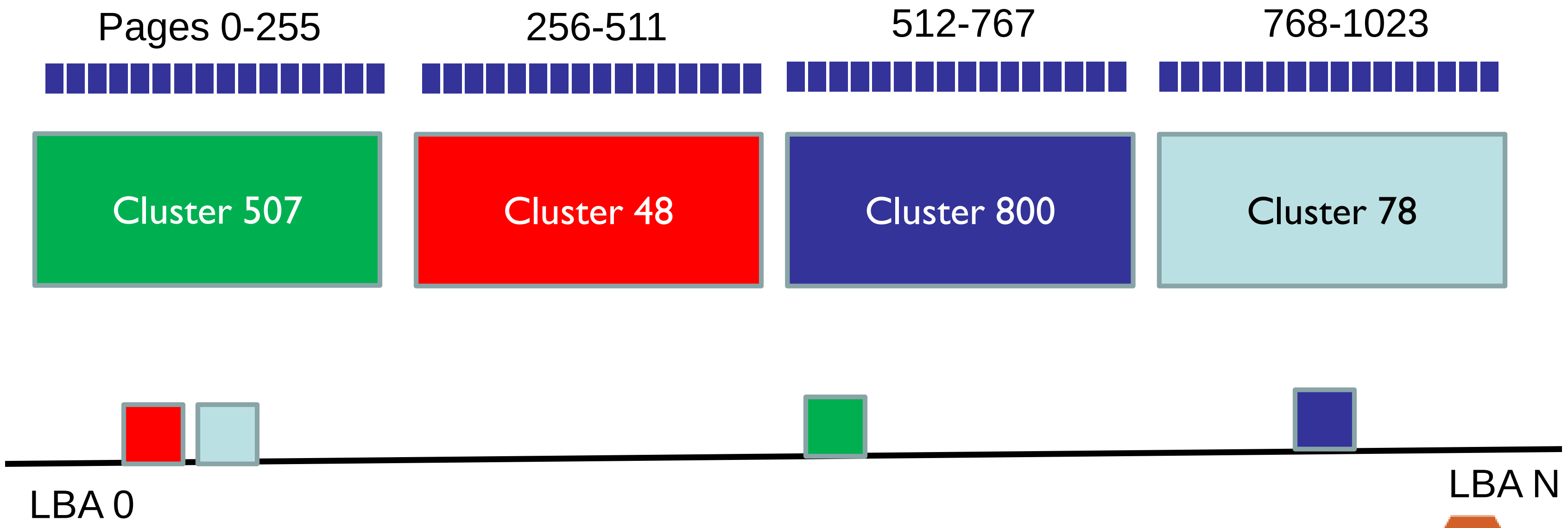
- Minimalistic for targeted storage use cases like RocksDB & Logical Volumes
- Deliver only the basics to enable another class of application
- Design for fast storage media

Blobstore Design – High Level

- Application interacts with chunks of data called blobs
  - Mutable array of pages of data, accessible via ID
- Asynchronous
  - No blocking, queuing or waiting
- Fully parallel
  - No locks in IO path

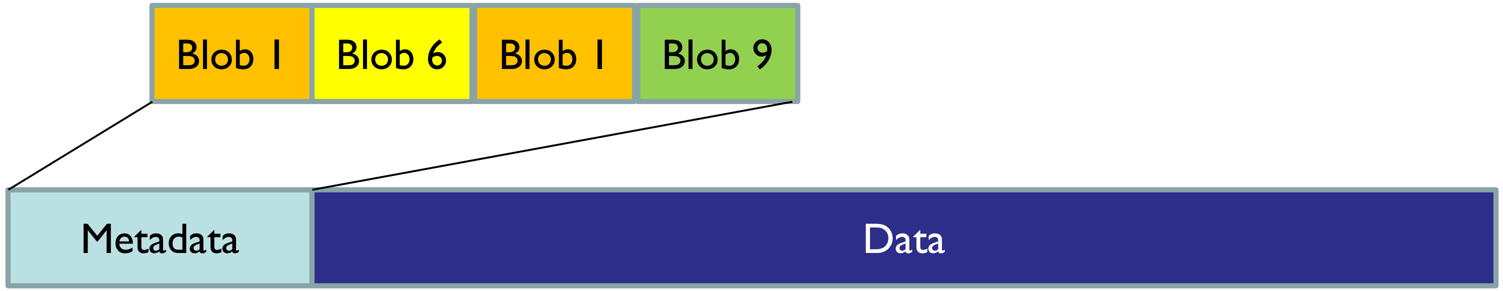
Blobstore Design - Layout

A blob is an array of pages organized as an ordered list of clusters



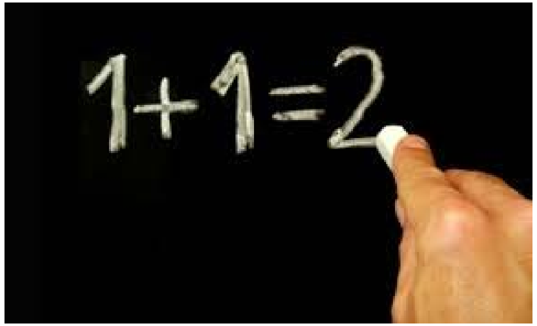
Blobstore Design - Metadata

- Stored in pages in reserved region of disk
- Not shared between blobs
- One blob may have multiple metadata pages



Blobstore Design - API

- Open, read, write, close, delete, resize, sync
- Asynchronous, callback driven
- Read/write in pages, allocate in clusters
- Data is <u>direct</u>
- Metadata is <u>cached</u>
- Minimal support for xattrs



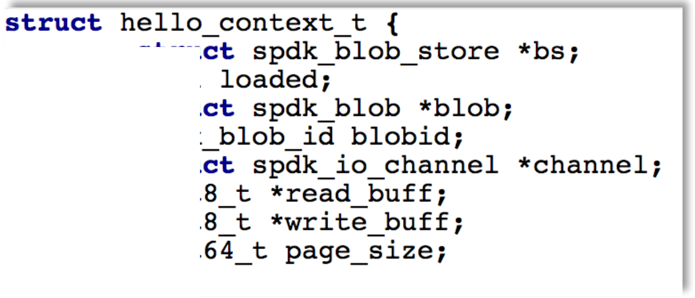
Hello Blob Walkthrough

```
#include "spdk/stdinc.h"

#include "spdk/bdev.h"
#include "spdk/env.h"
#include "spdk/event.h"
#include "spdk/blob_bdev.h"
#include "spdk/blob.h"
#include "spdk/log.h"
```

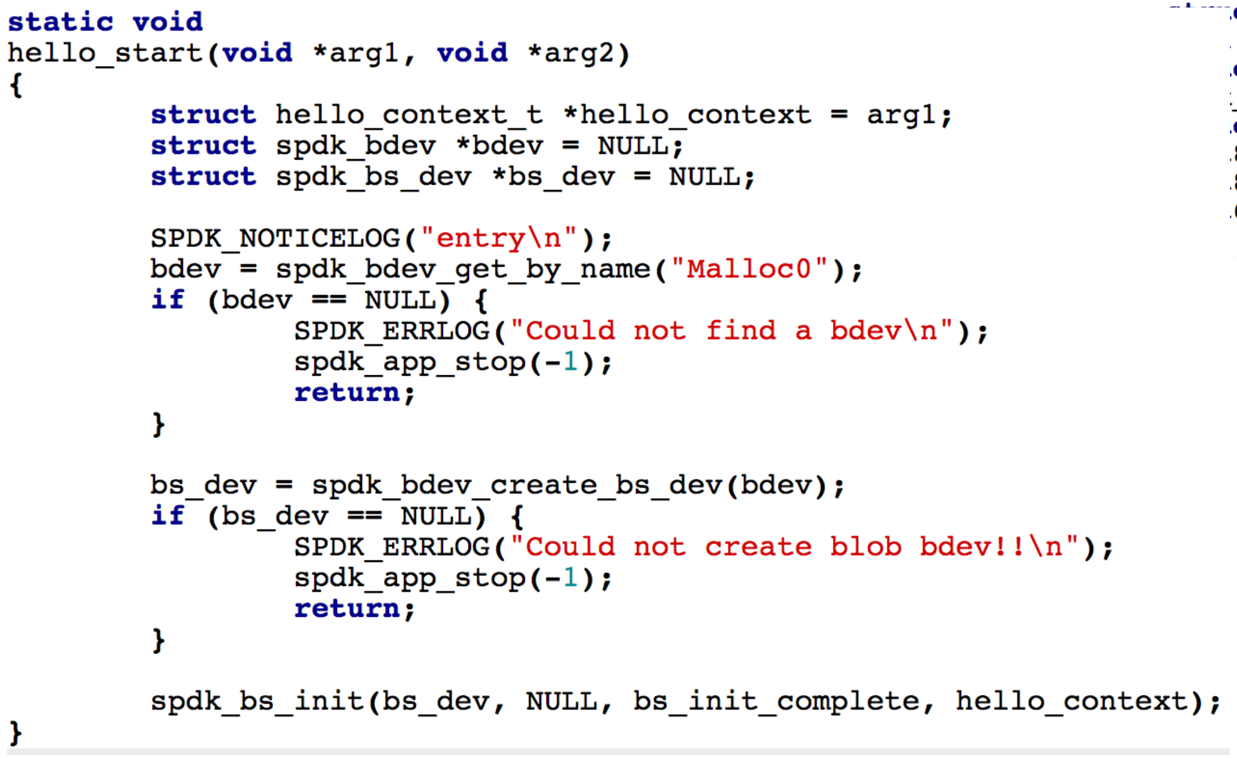
```
int
main(int argc, char **argv)
{
        struct spdk_app_opts opts = {};
        int rc = 0;
        struct hello_context_t *hello_context = NULL;

        SPDK_NOTICELOG("entry\n");
        spdk_app_opts_init(&opts);
        opts.name = "hello_blob";
        opts.config_file = "hello_blob.conf";

        hello_context = calloc(1, sizeof(struct hello_context_t));
        if (hello_context != NULL) {
                rc = spdk_app_start(&opts, hello_start, hello_context, NULL);
                if (rc) {
                        SPDK_ERRLOG("Something went wrong!\n");
                        if (hello_context->loaded == true) {
                                unload_bs(hello_context);
                        }
                } else {
                        SPDK_NOTICELOG("SUCCCESS!\n");
                }
                hello_cleanup(hello_context);
        } else {
                SPDK_ERRLOG("Could not alloc hello_context struct!!\n");
                rc = -ENOMEM;
        }

        spdk_app_fini();
        return rc;
}
```

2017 Storage Developer Conference. © Intel Corporation. All Rights Reserved.

Hello Blob Walkthrough



```
struct hello_context_t {
                     ct spdk_blob_store *bs;
                      loaded;
                     ct spdk_blob *blob;
                     _blob_id blobid;
                     ct spdk_io_channel *channel;
                     8_t *read_buff;
                     8_t *write_buff;
                     64_t page_size;
```



```
static void
hello_start(void *arg1, void *arg2)
{
        struct hello_context_t *hello_context = arg1;
        struct spdk_bdev *bdev = NULL;
        struct spdk_bs_dev *bs_dev = NULL;

        SPDK_NOTICELOG("entry\n");
        bdev = spdk_bdev_get_by_name("Malloc0");
        if (bdev == NULL) {
                SPDK_ERRLOG("Could not find a bdev\n");
                spdk_app_stop(-1);
                return;
        }

        bs_dev = spdk_bdev_create_bs_dev(bdev);
        if (bs_dev == NULL) {
                SPDK_ERRLOG("Could not create blob bdev!!\n");
                spdk_app_stop(-1);
                return;
        }

        spdk_bs_init(bs_dev, NULL, bs_init_complete, hello_context);
}
```

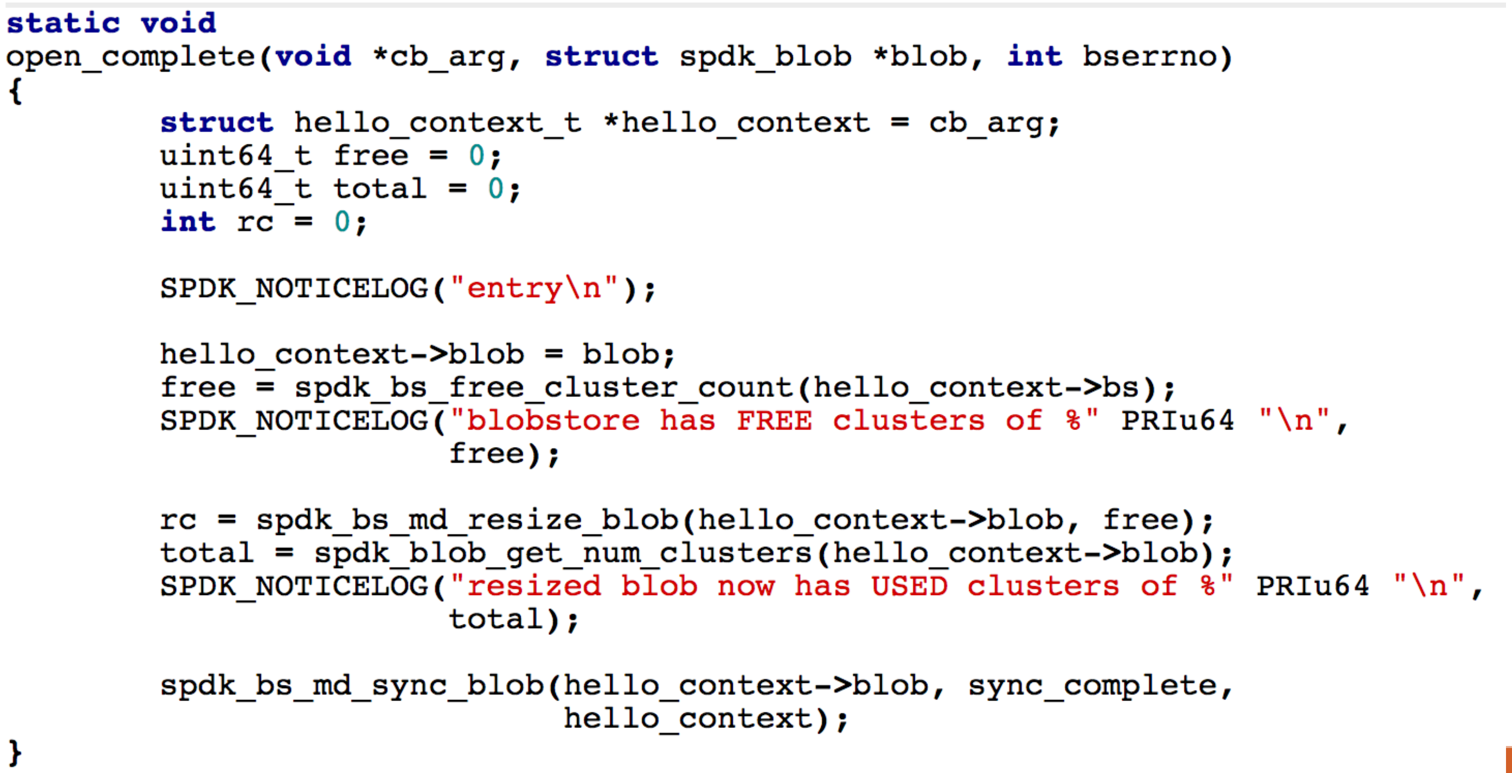
Hello Blob Walkthrough

```
static void
bs_init_complete(void *cb_arg, struct spdk_blob_store *bs,
                 int bserrno)
{
        struct hello_context_t *hello_context = cb_arg;

        SPDK_NOTICELOG("entry\n");
        if (bserrno) {
                SPDK_ERRLOG("Error %d init'ing the blobstore\n", bserrno);
                spdk_app_stop(bserrno);
                return;
        }
        hello_context->loaded = true;
        hello_context->bs = bs;
        SPDK_NOTICELOG("blobstore: %p\n", hello_context->bs);
        hello_context->page_size = spdk_bs_get_page_size(hello_context->bs);
        create_blob(hello_context);
}
```

Hello Blob Walkthrough

```
static void
blob_create_complete(void *arg1, spdk_blob_id blobid, int bserrno)
{
        struct hello_context_t *hello_context = arg1;

        SPDK_NOTICELOG("entry\n");
        if (bserrno) {
                SPDK_ERRLOG("Error %d blob create callback\n", bserrno);
                spdk_app_stop(bserrno);
                return;
        }

        hello_context->blobid = blobid;
        SPDK_NOTICELOG("new blob id %" PRIu64 "\n", hello_context->blobid);

        spdk_bs_md_open_blob(hello_context->bs, hello_context->blobid,
                             open_complete, hello_context);
}

static void
create_blob(struct hello_context_t *hello_context)
{
        SPDK_NOTICELOG("entry\n");
        spdk_bs_md_create_blob(hello_context->bs, blob_create_complete,
                               hello_context);
}
```

Hello Blob Walkthrough



```
static void
open_complete(void *cb_arg, struct spdk_blob *blob, int bserrno)
{
        struct hello_context_t *hello_context = cb_arg;
        uint64_t free = 0;
        uint64_t total = 0;
        int rc = 0;

        SPDK_NOTICELOG("entry\n");

        hello_context->blob = blob;
        free = spdk_bs_free_cluster_count(hello_context->bs);
        SPDK_NOTICELOG("blobstore has FREE clusters of %" PRIu64 "\n",
                       free);

        rc = spdk_bs_md_resize_blob(hello_context->blob, free);
        total = spdk_blob_get_num_clusters(hello_context->blob);
        SPDK_NOTICELOG("resized blob now has USED clusters of %" PRIu64 "\n",
                       total);

        spdk_bs_md_sync_blob(hello_context->blob, sync_complete,
                             hello_context);
}
```

Hello Blob Walkthrough

```
static void
blob_write(struct hello_context_t *hello_context)
{
        SPDK_NOTICELOG("entry\n");

        hello_context->write_buff = spdk_dma_malloc(hello_context->page_size,
                                    0x1000, NULL);
        memset(hello_context->write_buff, 0x5a, hello_context->page_size);

        hello_context->channel = spdk_bs_alloc_io_channel(hello_context->bs);

        spdk_bs_io_write_blob(hello_context->blob, hello_context->channel,
                              hello_context->write_buff,
                              0, 1, write_complete, hello_context);
}

static void
sync_complete(void *arg1, int bserrno)
{
        struct hello_context_t *hello_context = arg1;

        SPDK_NOTICELOG("entry\n");
        blob_write(hello_context);
}
```

Hello Blob Walkthrough

```
static void
read_blob(struct hello_context_t *hello_context)
{
        SPDK_NOTICELOG("entry\n");

        hello_context->read_buff = spdk_dma_malloc(hello_context->page_size,
                                   0x1000, NULL);
        spdk_bs_io_read_blob(hello_context->blob, hello_context->channel,
                             hello_context->read_buff, 0, 1, read_complete,
                             hello_context);
}

static void
write_complete(void *arg1, int bserrno)
{
        struct hello_context_t *hello_context = arg1;

        SPDK_NOTICELOG("entry\n");
        read_blob(hello_context);
}
```

Hello Blob Walkthrough

```
static void
read_complete(void *arg1, int bserrno)
{
        struct hello_context_t *hello_context = arg1;
        int match_res = -1;

        SPDK_NOTICELOG("entry\n");
        match_res = memcmp(hello_context->write_buff, hello_context->read_buff,
                           hello_context->page_size);
        if (match_res) {
                SPDK_ERRLOG("Error in read completion, buffers don't match\n");
                spdk_app_stop(-1);
                return;
        } else {
                SPDK_NOTICELOG("read SUCCESS and data matches!\n");
        }

        spdk_bs_md_close_blob(&hello_context->blob, delete_blob,
                              hello_context);
}
```

Hello Blob Walkthrough

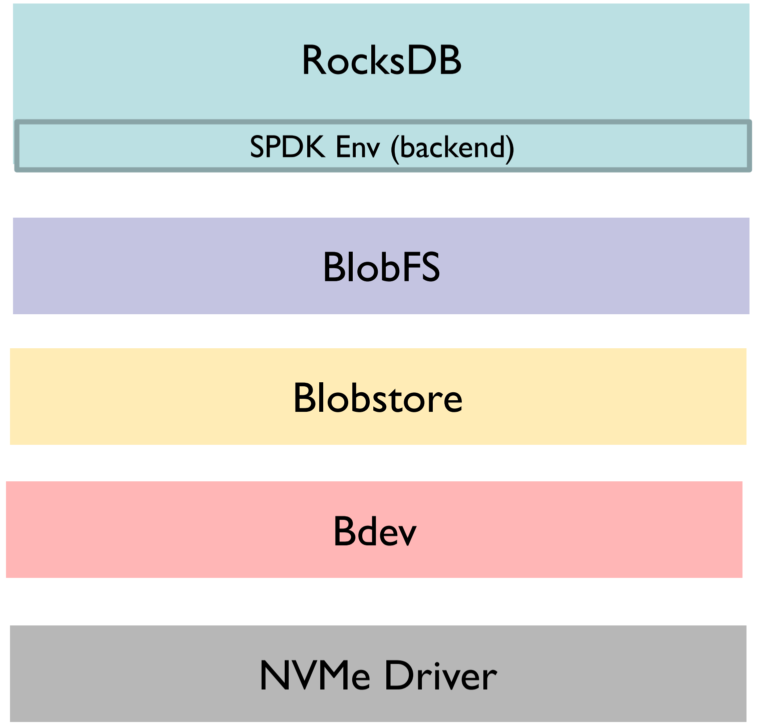
```
static void
delete_complete(void *arg1, int bserrno)
{
        struct hello_context_t *hello_context = arg1;

        SPDK_NOTICELOG("entry\n");
        unload_bs(hello_context);
}

static void
delete_blob(void *arg1, int bserrno)
{
        struct hello_context_t *hello_context = arg1;

        SPDK_NOTICELOG("entry\n");
        spdk_bs_md_delete_blob(hello_context->bs, hello_context->blobid,
                               delete_complete, hello_context);
}
```

2017 Storage Developer Conference. © Intel Corporation. All Rights Reserved.

Hello Blob Walkthrough

```
static void
unload_complete(void *cb_arg, int bserrno)
{
        struct hello_context_t *hello_context = cb_arg;

        SPDK_NOTICELOG("entry\n");
        hello_context->loaded = false;
        spdk_app_stop(0);
}

static void
unload_bs(struct hello_context_t *hello_context)
{
        spdk_bs_free_io_channel(hello_context->channel);
        spdk_bs_unload(hello_context->bs, unload_complete, hello_context);
}
```

SPDK and RocksDB



2017 Storage Developer Conference. © Intel Corporation. All Rights Reserved.

Performance Comparisons (SPDK vs. Linux Kernel)

System, Dataset & Workload Configuration

System Configuration	OS Configuration	SSD Details
Processor: Intel(R) Xeon(R) CPU E5-2618L v4 @ 2.20GHz **Total Physical CPU** (HT disabled): 20 **Total Memory**: 64GB	**Distro:** Ubuntu 16.04.1 LTS **Kernel:** 4.12.4 (built) **Arch:** x86_64 Intel ® Optane™ SSD DC P4800X	**SSD:** 1x Intel ® Optane™ SSD DC P4800X 375GB

SPDK Tuning parameters	Linux Kernel 4.12.0 Tuning parameters
Cache Size: 30GB	Page Cache Size: 30GB (using cgroups)
Cache_buffer_size: 64KB	Bytes per sync: 64KB XFS filesystem, agcount=32, mount with discard

Dataset:

242GB (250 million uncompressed records)

- ❐ Dataset size kept higher (4:1) than main memory size
- ❐ Fills ~70% of disk capacity

Key_size: 16 Bytes

Value_size: 1000 Bytes

Db_bench:

Part of RocksDB: https://github.com/spdk/rocksdb

Test Duration: 5 minutes

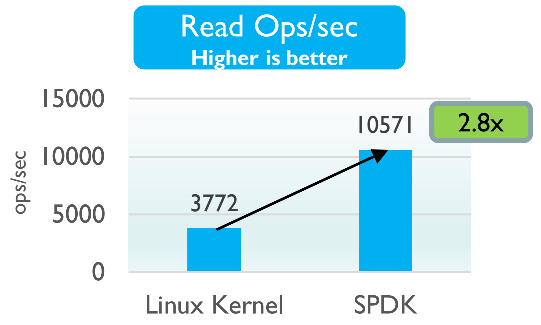
No. of Runs for each test: 3

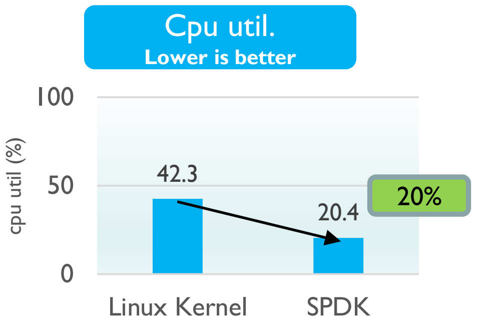
Compression: None

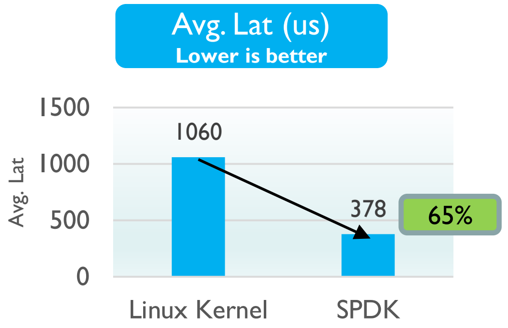
Workloads:

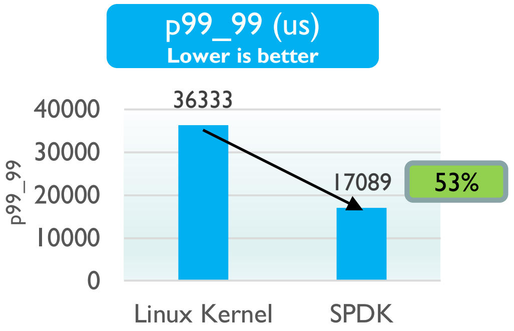
- ❐ ReadWrite
- ❐ Fillsync
- ❐ Overwrite

Performance & Latency Workload # 1: Readwrite



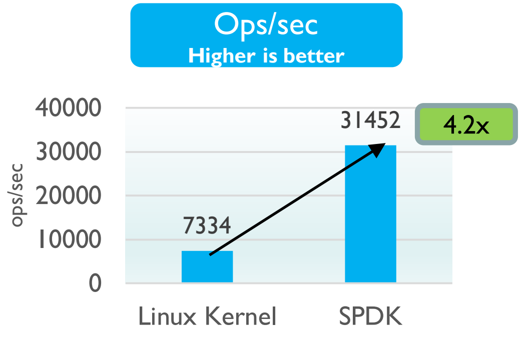


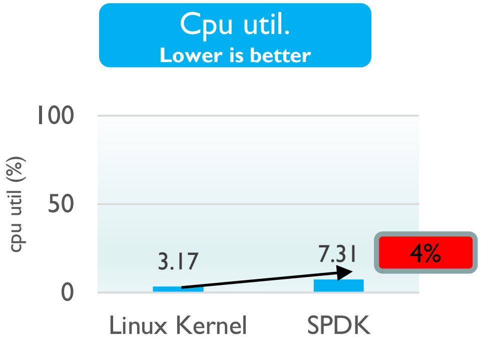


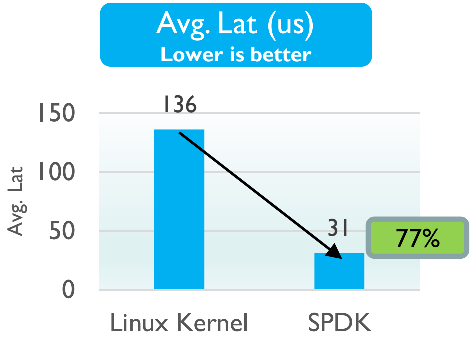


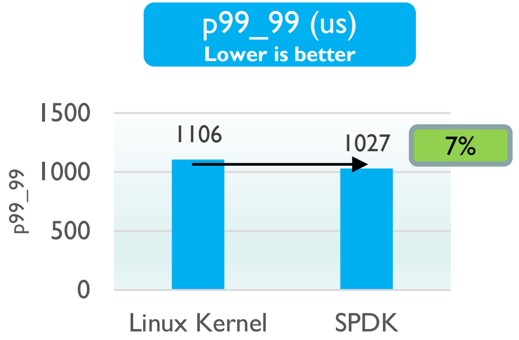
- 90% Reads 10% Writes
- Up to 2.8x performance improvement
- Up to 53% improvement in tail latency
- Up to 20% improvement in CPU utilization

Performance & Latency Workload # 2: Fillsync





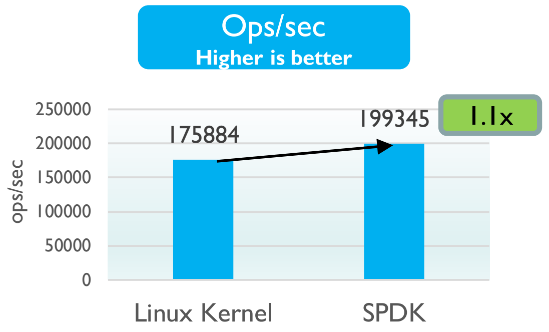


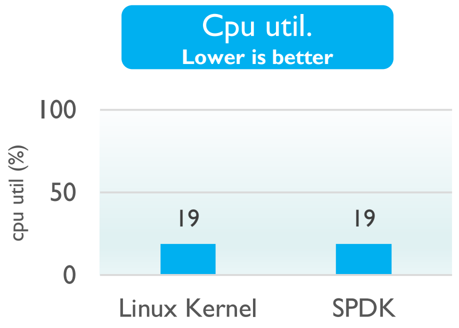


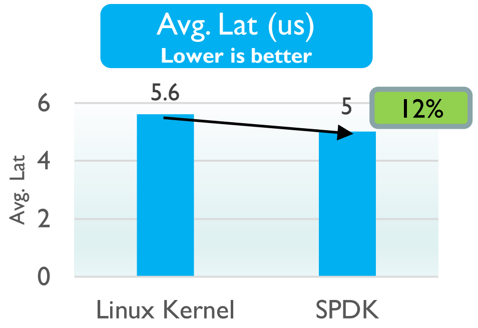
- Fillsync writes values in random key order in **sync** mode
- Up to 4.2x performance improvement
- Up to 77% improvement in average latency

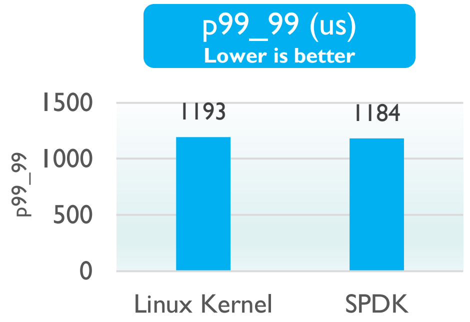
SDC 17

Performance & Latency Workload # 3: Overwrite









- Updates values in random key order in async mode
- Workload comprised of large block I/Os to the disk
- Compaction & flush activity happening in background so not much potential of performance improvement

Summary

- ❐ SPDK allows storage applications to take full advantage of today’s fastest CPUs and SSDs
- ❐ New features and functions are always coming
- ❐ SPDK is an open source community that’s growing strong!

For more info: http://www.spdk.io/ or catch us on IRC!

STORAGE DEVELOPER CONFERENCE

SNIA SANTA CLARA, 2017

Backup

Benchmark Configuration

Db_bench: From RocksDB v5.4.5

Workload	db_bench parameters
Overwrite	overwrite, threads=1, disable_wal=1
ReadWrite	readwhilewriting, threads=4, disable_wal=1
Insert	fillseq, threads=1, disable_wal=1
Fillsync	fillsync, threads=1, disable_wal=0, sync=1
Common parameters	--disable_seek_compaction=1,--mmap_read=0, --statistics=1, --histogram=1, --key_size=16, --value_size=1000, --cache_size=10737418240 , --block_size=4096, -, bloom_bits=10, --open_files=500000, --verify_checksum=1, --db=/mnt/rocksdb, --sync=0, --compression_type=none, --stats_interval=1000000, --compression_ratio=1, --disable_data_sync=0, -,target_file_size_base=67108864, --max_write_buffer_number=3, --max_bytes_for_level_multiplier=10, --max_background_compactions=10, --num_levels=6, --delete_obsolete_files_period_micros=3000000, --max_grandparent_overlap_factor=10, --stats_per_interval=1, --max_bytes_for_level_base=10485760, --stats_dump_period_sec=60

The Problem: Software is becoming the bottleneck

	HDD	SATA NAND SSD	NVMe* NAND SSD	Intel® Optane™ SSD
Latency	>2ms	<100µs	<100µs	<10µs
I/O Performance	<500 IO/s	>25,000 IO/s	>400,000 IO/s	>500,000 IO/s

The Opportunity: SPDK unlocks new media potential

SPDK Updates: 17.07 Release (Aug 2017)

32 Unique Contributors!

Userspace vhost-blk Target

- Vhost-scsi target extended to also support vhost-blk

GPT Partition table support

- Exports partitions as SPDK bdevs

Build system improvements

- Added configure script which simplifies build time configuration

Improvements to existing features

- API cleanup and simplification