STORAGE DEVELOPER CONFERENCE
SDC 22
Fremont, CA
September 12-15, 2022
BY Developers FOR Developers
A SNIA Event

# What's Faster Than a Cheetah and More Flexible Than a Cirque du Soleil Performer?

# The new SPDK Accelerator Framework!

Paul Luse

# Notices and Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing on certain dates using certain configurations and may not reflect all publicly available updates. Reach out to Intel for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

# Agenda

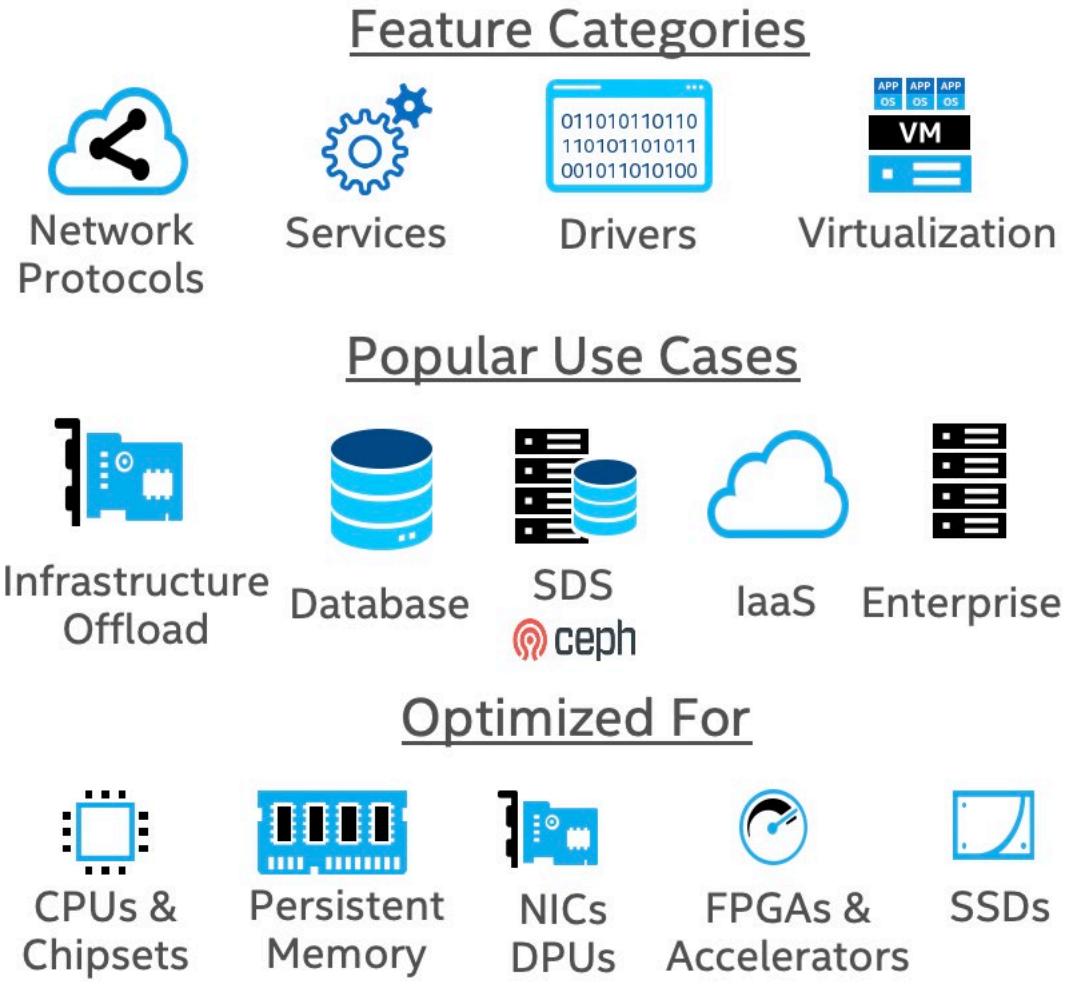
- What is SPDK?
- High Level Architecture
- The Original Copy Engine
- New Requirements
- The Accelerator Framework
- Quick Deep Dive: The Intel® Data Streaming Accelerator Module

# SPDK Overview

- Open-Source Project for developing bleeding edge storage solutions
- Vibrant, Multi-vendor, Global Community
  - 25+ companies contributed over last 12 months
- Libraries, Drivers, Applications and Tools
- Unmatched Speed and Efficiency
  - 10M+ NVMe IO/core/sec
- Rich Feature Set
- Broad adoption in Cloud & Enterprise
- Fully open development model & BSD Licensed

**https://spdk.io**

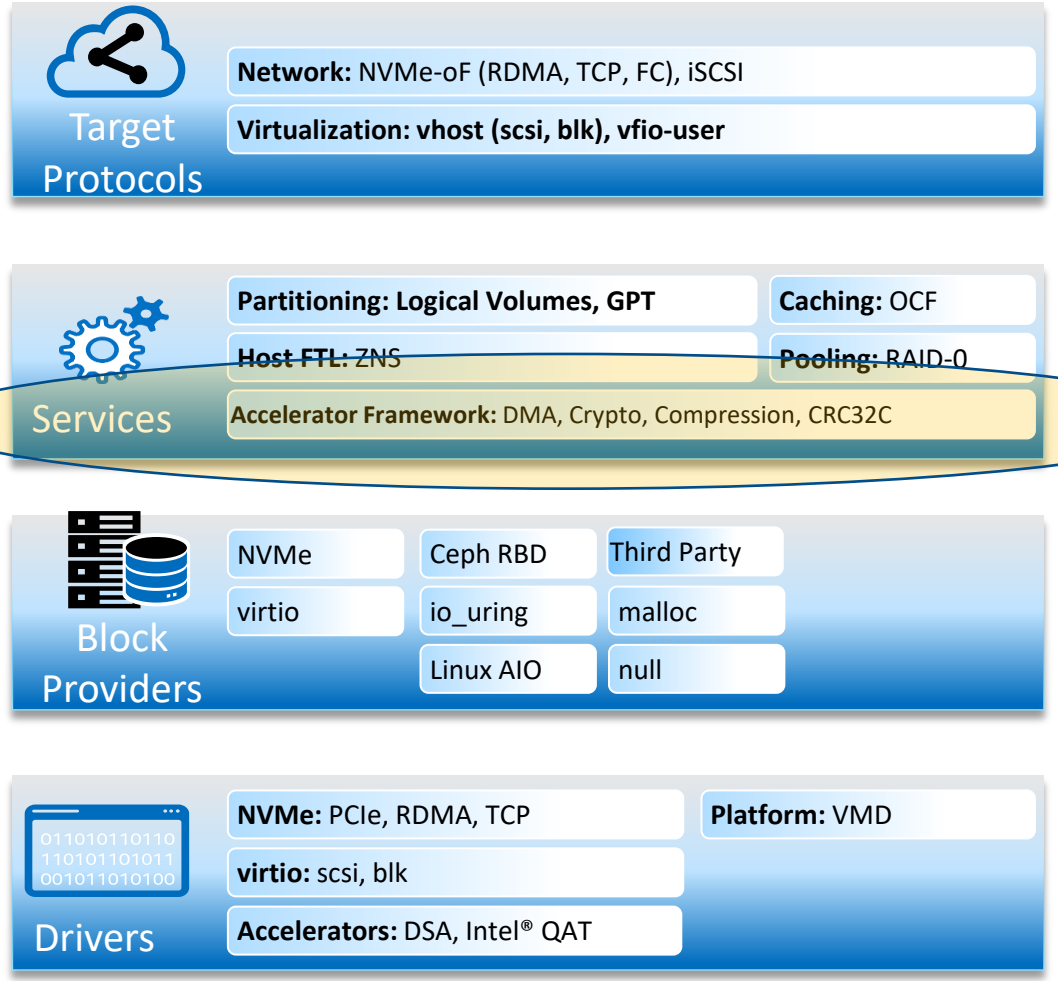


# Storage Performance Development Kit Architecture

Plug-in Based Framework For all kinds of HW and/or SW Accelerator Modules

## Architecture




## Integrations

**Orchestration**
Cinder
Kubernetes

**Database**
RocksDB

**SDS**
DAOS, Ceph, Linstor

**Virtualization**
Qemu

**Analyzers**
VTune™

## Tools

**Benchmarking**
fio
SPDK perf

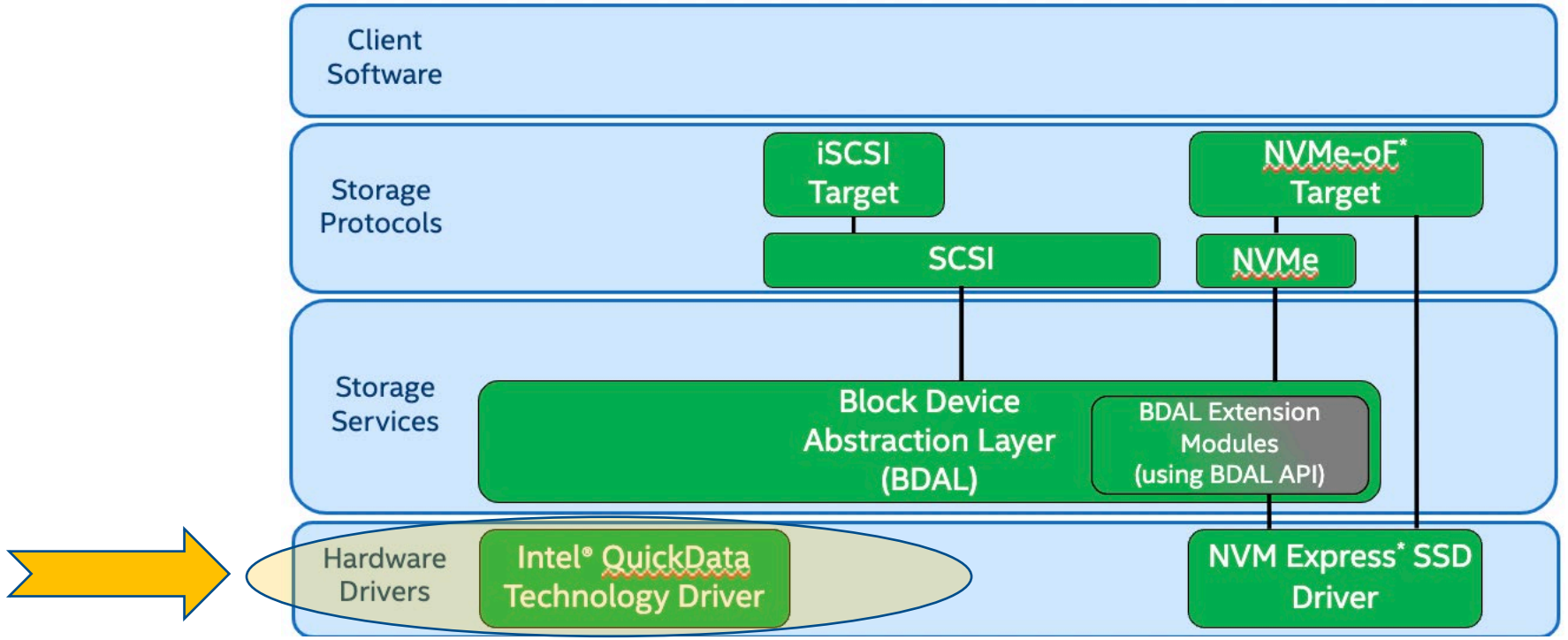
**Management**
nvme-cli
spdk-cli

**Diagnostic**
spdk_top

# The Original Copy Engine - Blast from the Past!

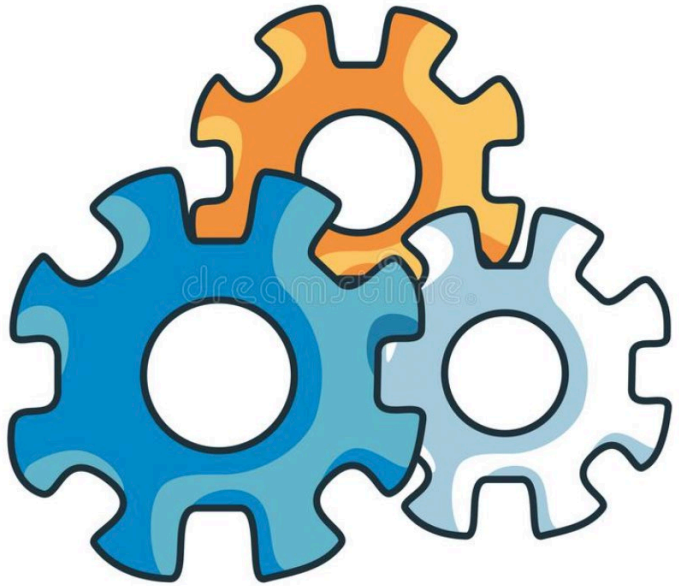
**Where is the Copy Engine in this architecture diagram from 2016?**



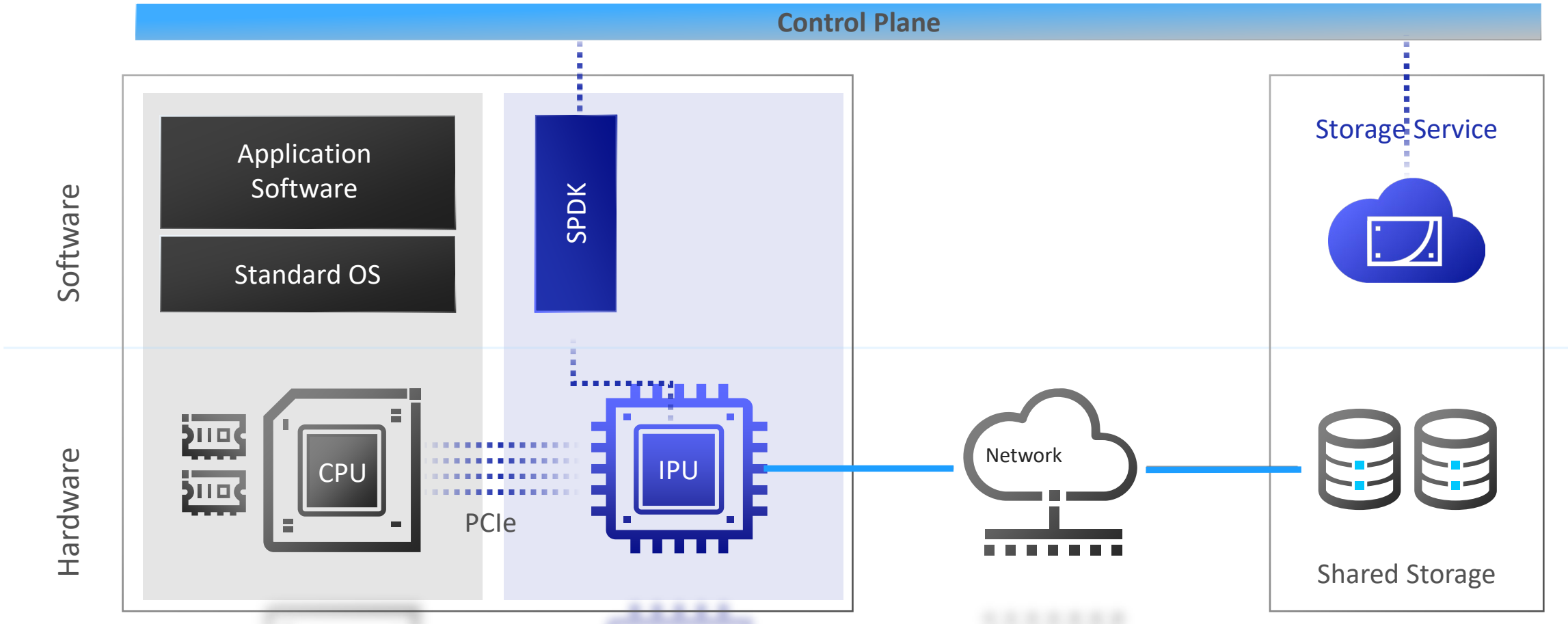
# Requirements That Drove The Accelerator Framework

- Build on what the Copy Engine was developed for; to provide a generic set of APIs that can map to either hardware offload or software depending on configuration.
- The coming of new HW offload engines (ie the Intel ® Data Streaming Accelerator aka DSA) that do operations like copy, fill, CRC32C, and more.
- The introduction of virtual block devices for crypto and compression that directly used DPDK interfaces from the block device (bdev) layer
- The Infrastructure Processing Unit (IPU) is likely to introduce new offloads



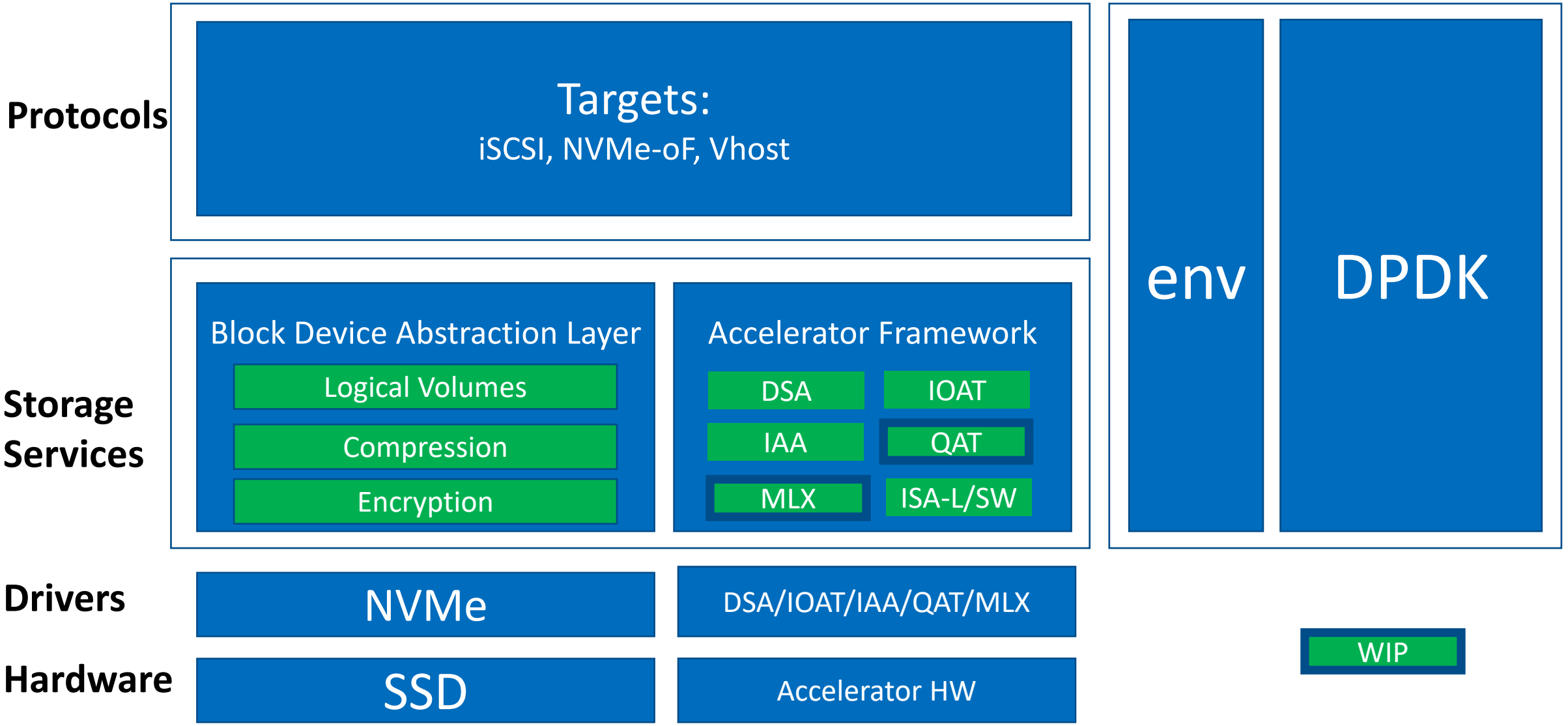
# SPDK and Infrastructure Processing Units




*For more info see https://ipdk.io/*

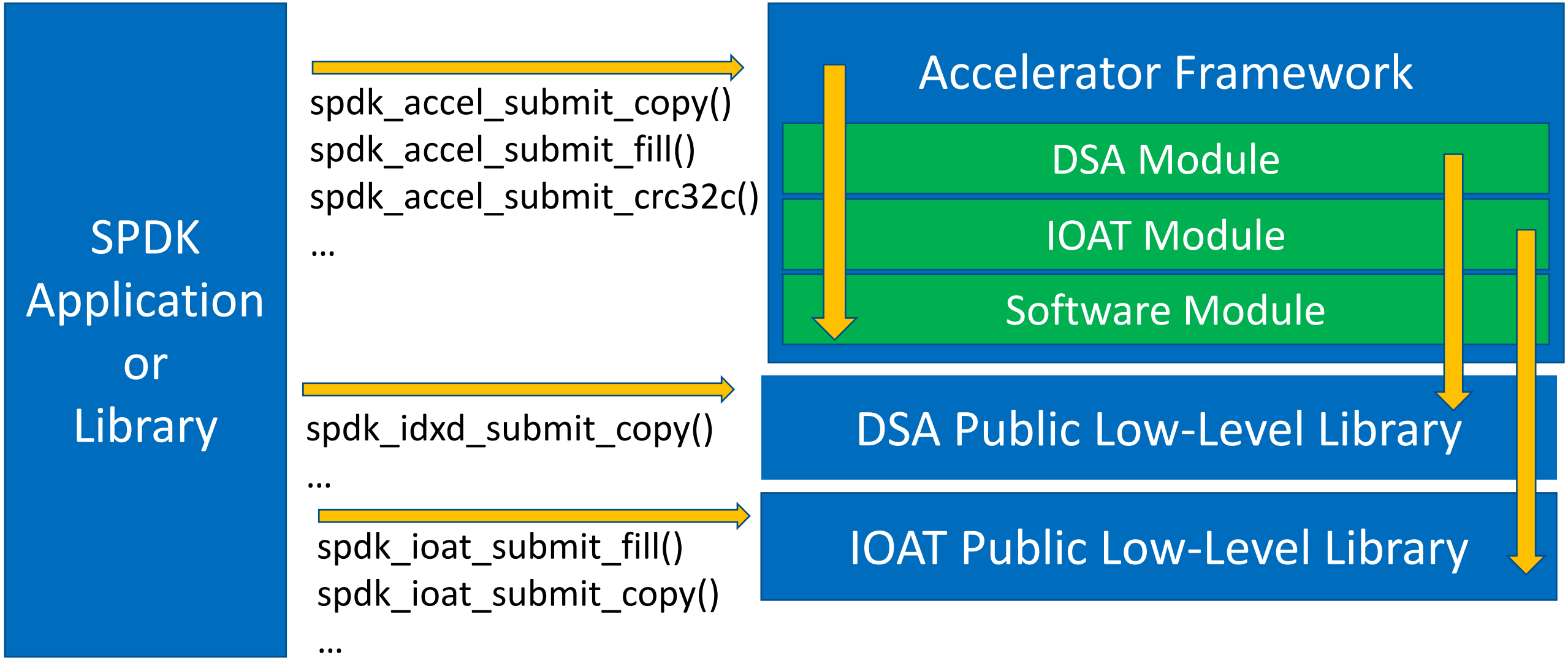
 

# The SPDK Accelerator Framework



# SPDK Accelerator Framework & Libraries

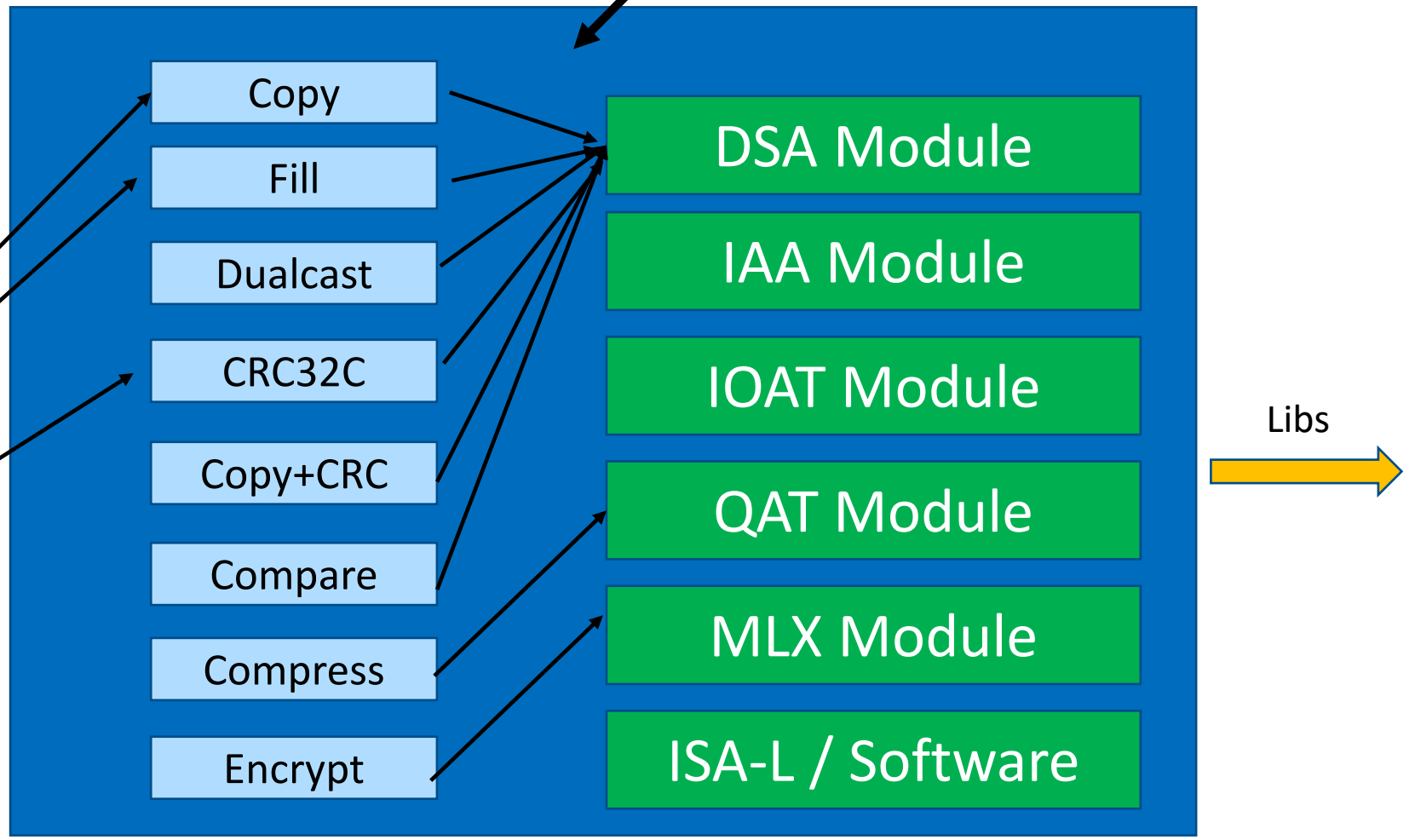


# SPDK Accelerator Framework

*Op code to engine mappings are init'd*
*Based on priorities but can be overridden*
*Via RPC or API*

spdk_accel_submit_copy()
spdk_accel_submit_fill()
spdk_accel_submit_crc32c()
…



# Engine/Module Discovery

*Via C API*

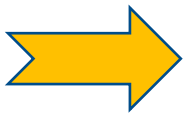
Retrieve the name of any engine given the opcode using:

spdk_accel_get_opc_engine_name()

*Via RPC*

Get a complete list of mappings using:

accel_get_engine_info



Example request:

```
{
  "jsonrpc": "2.0",
  "method": "accel_get_engine_info",
  "id": 1
}
```

Example response:

```
[
  {
    "engine": "software",
    "supported ops": [
      "copy",
      "fill",
      "dualcast",
      "compare",
      "crc32c",
      "copy_crc32c",
      "compress",
      "decompress"
    ]
  },
  {
    "engine": "dsa",
    "supported ops": [
      "copy",
      "fill",
      "dualcast",
      "compare",
      "crc32c",
      "copy_crc32c"
    ]
  }
]
```

For more info: https://spdk.io/doc/jsonrpc.html

# Engine/Module Assignment

*Via C API*

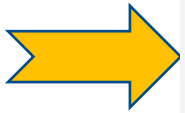
Assign an opcode to a specific engine using:

spdk_accel_assign_opc()

*Via RPC*

Assign an opcode to a specific engine using:

accel_assign_opc



Example request:

```
{
  "jsonrpc": "2.0",
  "method": "accel_assign_opc",
  "id": 1,
  "params": {
    "opanme": "copy",
    "engine": "software"
  }
}
```

Example response:

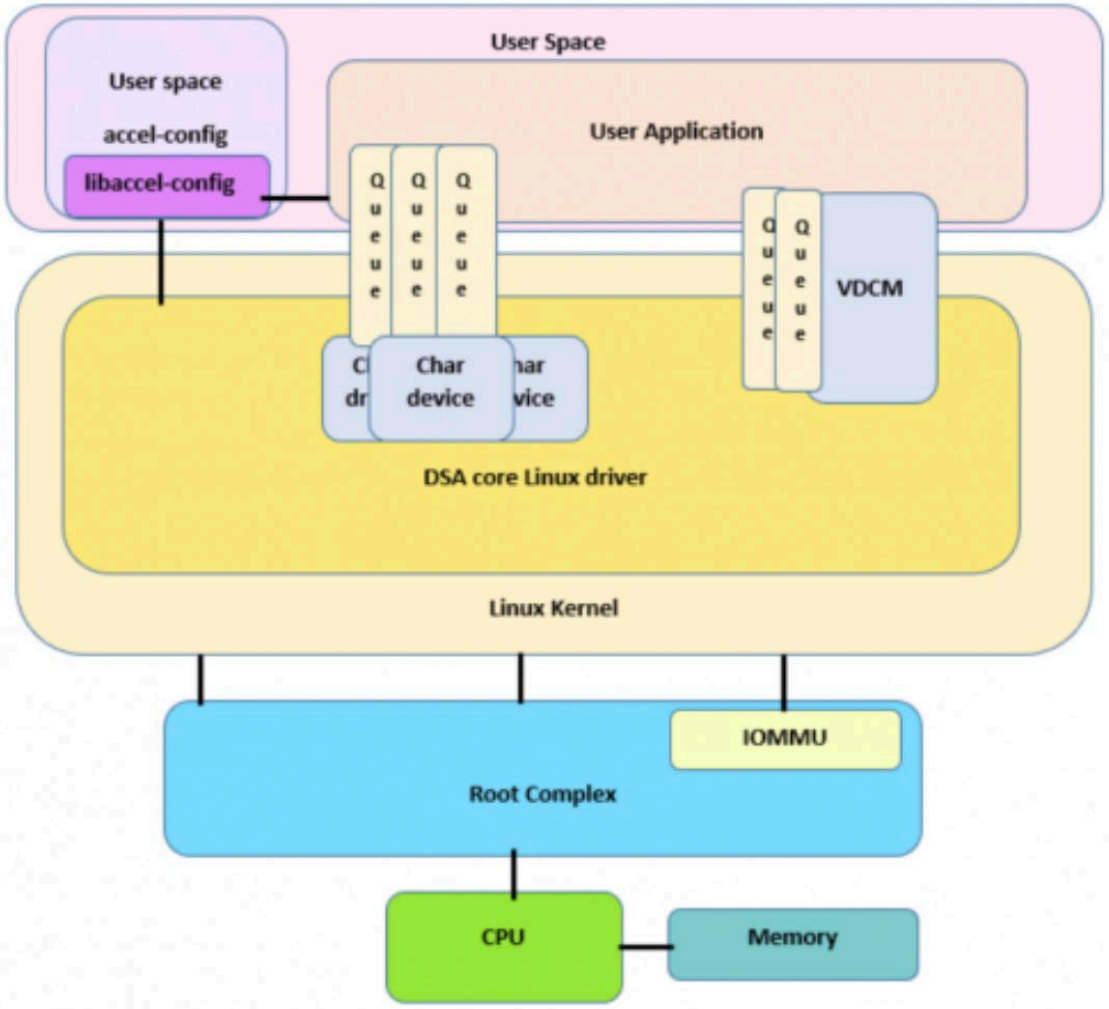
```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": true
}
```

For more info: https://spdk.io/doc/jsonrpc.html

# General (not SPDK specific) Intel® DSA Overview

*Key Point:*
*Hardware designed for both kernel mode and user space use cases*




See https://01.org/blogs/2019/introducing-intel-data-streaming-accelerator for more info

# The SPDK Implementation of Intel® DSA: Two Modes

## Default Mode: SPDK User space driver

- SPDK controls DSA via UIO or VFIO and is responsible for complete initialization and control of the DSA device.  Best for use cases where the SPDK based application wants complete control with no kernel dependencies.

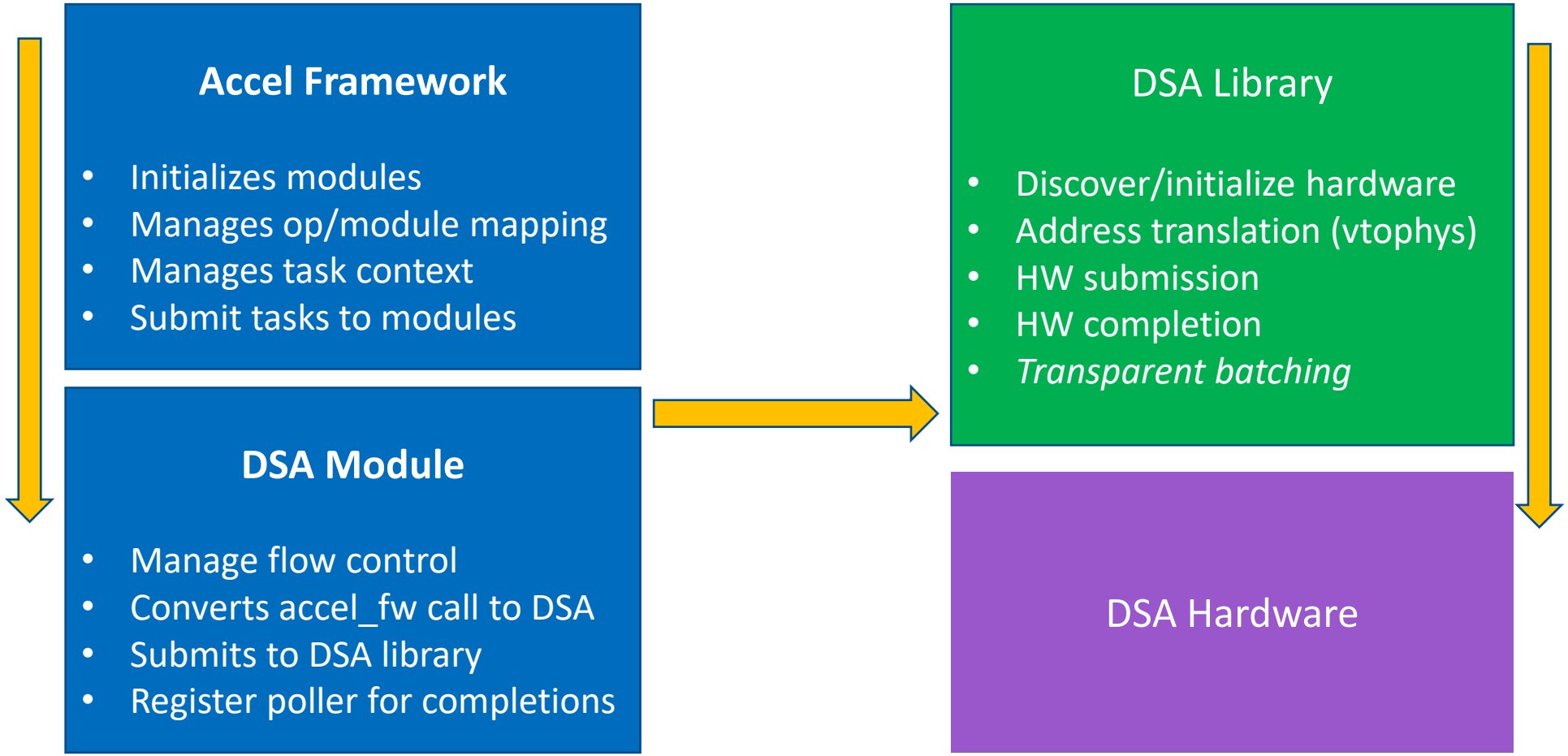
## Alternate Mode: Shared Kernel mode driver

- SPDK relies in the Linux kernel driver to initialize the hardware and uses a tool called accel_config (not from SPDK) to create work queue(s) and configure the hardware.  SPDK is responsible for submitting and completing operations (entire IO path).  Best for cases where multiple software applications want to share the DSA hardware (k8s, etc)

*Note: No performance difference have been seen between these modes*

# The SPDK Implementation of Intel® DSA: R&Rs



# Intel® DSA Transparent Batching

**Goal: Make the most use of the hardware, it's lightning quick!**

DSA has a batch operation that allows for the submission of a set of other commands

Although it's possible to expose this to the SPDK based application, we found few application use cases that could take advantage of it with any quantifiable gains.

As DSA is so fast, we found that it's optimal to "batch up" single operations as they come in and then submit them as a single chain when either (a) the batch is full or (b) the poller runs

| | |
|---|---|
| 1 | copy |
| 2 | copy |
| 3 | fill |
| 4 | dualcast |
| 5 | crc32c |
| 6 | copy |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |
| 14 | |
| 15 | |
| 16 | |

Example:
*This batch is partial and contains unrelated operations. If the poller runs before all 16 slots are full they will be submitted as a batch. If the app is fast enough to fill it before the next poller runs it will submit at that time.*

# What Can You Do?

Get Involved!

http://spdk.io

# Please take a moment to rate this session.

Your feedback is important to us.