STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers
Virtual Conference
September 28-29, 2021

Computational Storage Architecture Simplification and Evolution

Jason Molgaard, Sr. Principal Storage Solutions Architect, Arm

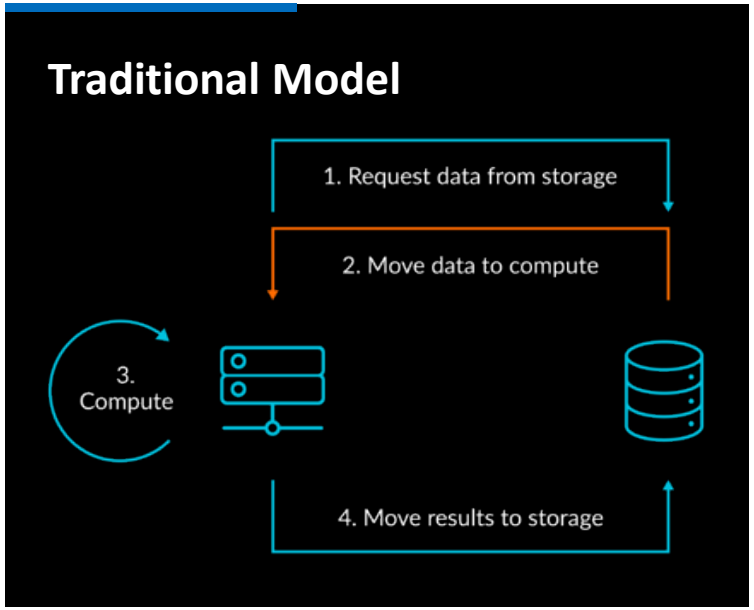
Agenda

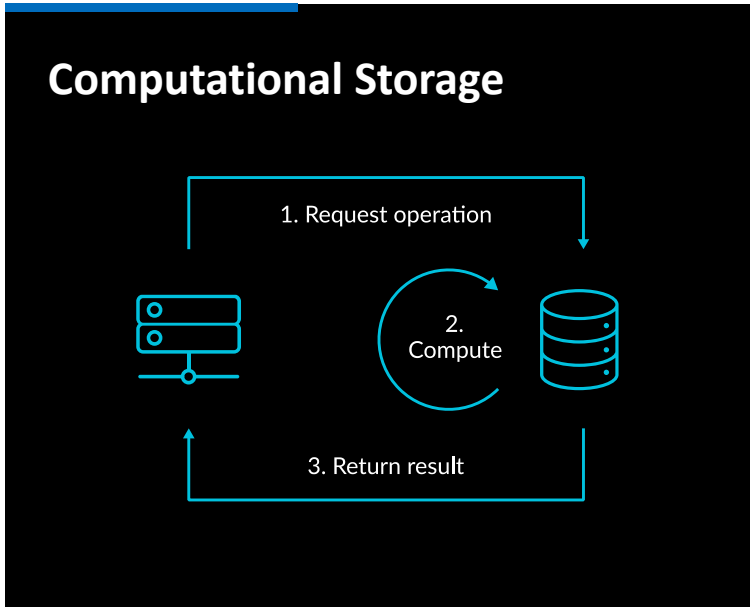
- Computational Storage Overview
- Controller Hardware Architecture
- Controller Software Architecture
- Combining CXL and Computational Storage
- Conclusions

Computational Storage Overview

Computational Storage

Generating insight where data is stored





Energy efficiency

Low latency

Security

Data-centric workloads

Benefits of Moving Compute to Storage

Bandwidth

Power

Cost

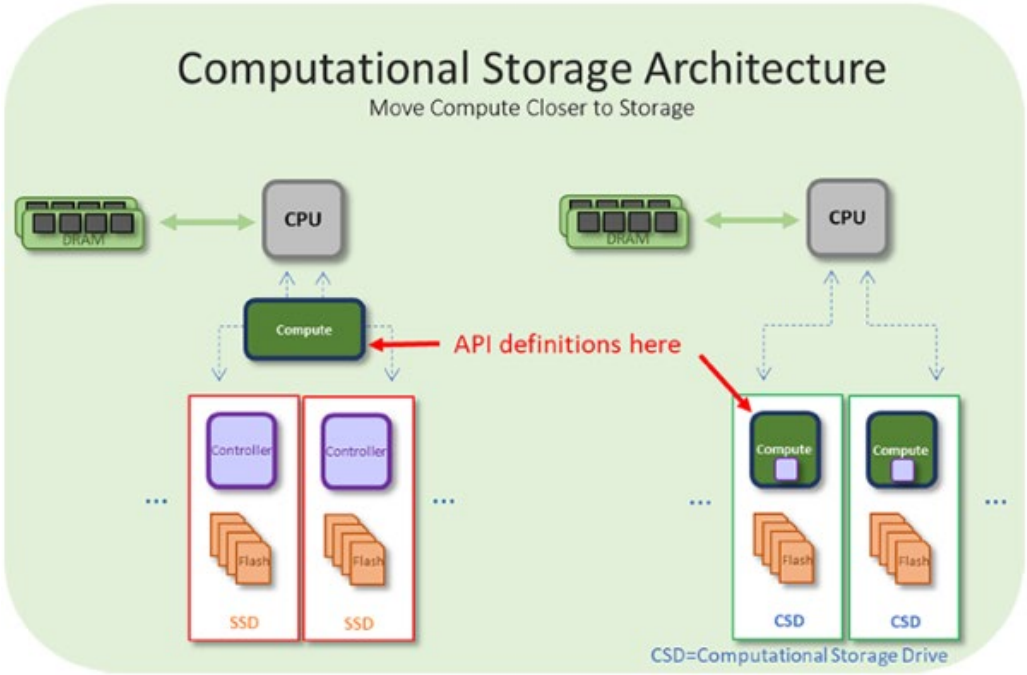
Latency

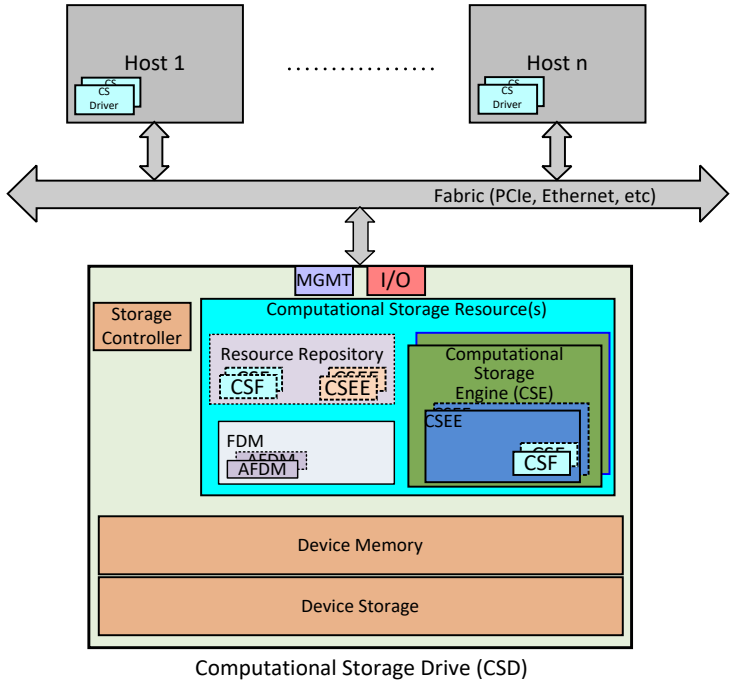
Reliability

Security

SNIA Computational Storage TWG

- CSD Standard required for adoption
  - CSD purchasers require multi-sourcing
- Arm is an active member participant
- Draft architecture (0.8r0) available
- Draft API (0.5r0) available





Computational Storage Drive (CSD)

SNIA | COMPUTATIONAL STORAGE

51 Participating Companies - 261 Member Representatives

SNIA COMPUTATIONAL STORAGE

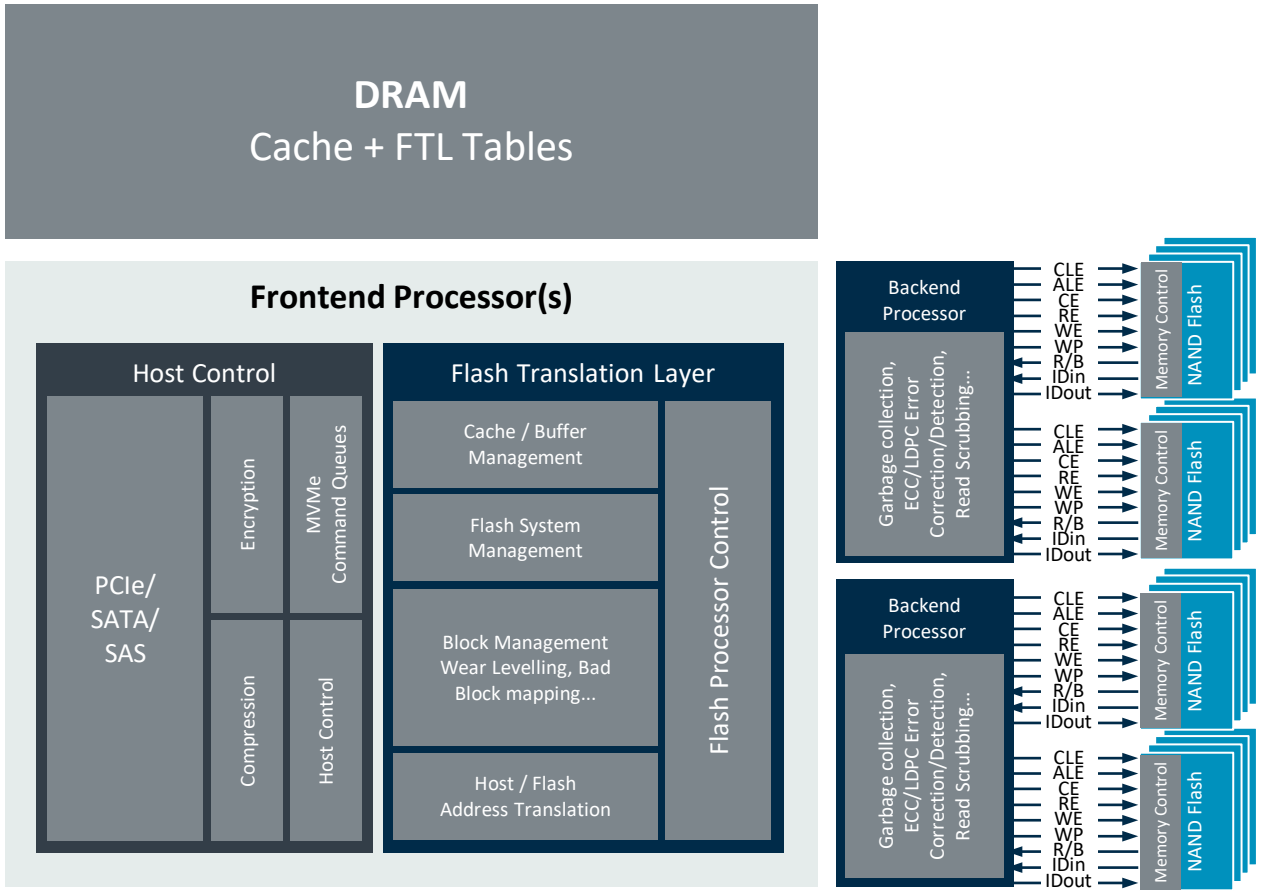
STORAGE DEVELOPER CONFERENCE SDC 21

Controller Hardware Architecture

Compute in SSD Controllers

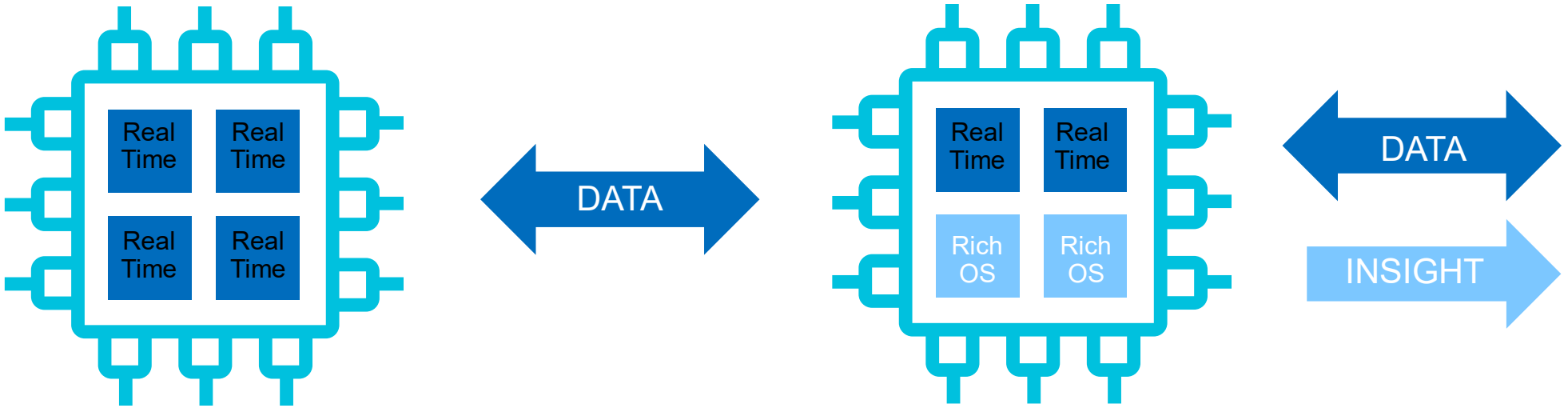
- Frontend: Host I/F + FTL
  - Arm Cortex-R or Cortex-A series
- Backend: Flash management
  - Cortex-R or Cortex-M series
- System IP
  - CoreSight, GIC, Interconnect, etc
- Hardware Engines:
  - Encryption, LDPC, Compression…
- DRAM ~1GB per 1TB of flash
- Storage: 256GB to 64TB… flash
- Interfaces: PCIe/SATA/SAS…

SSD SoC Functionality:



Enabling Rich Operating System Workloads On-Drive

Generating insight where data is stored



Storage controllers traditionally run real-time workloads to store and access data

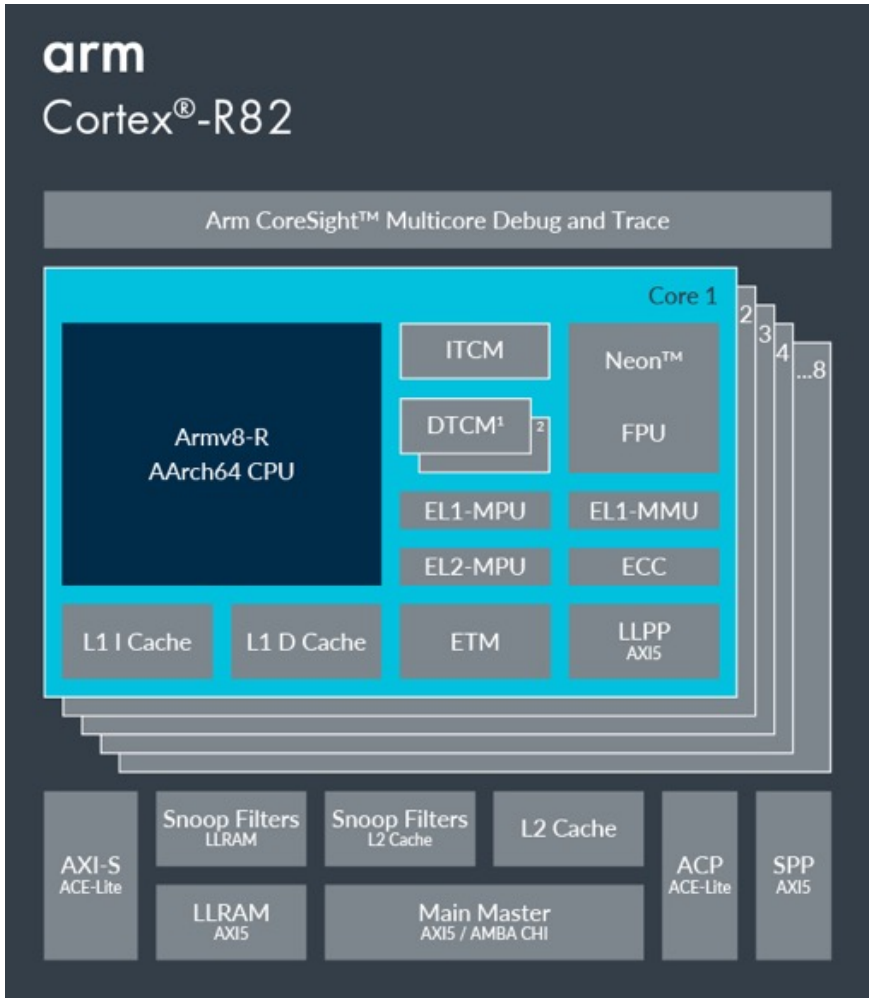
A Memory Management Unit enables eBPF or Rich Operating System workloads to generate insight from the data

Applications Processors: Low-cost, Low-power Compute

- Linux requires a processor with a Memory Management Unit
  - Memory Management Unit (MMU) to virtualize memory
  - Cortex-A series have 21 processors available + strong roadmap
  - Cortex-R82 provides real-time plus MMU capability
  - **Meeting every performance point at the lowest possible power**
- Some eSSD controllers already use application processors
  - Other controllers, using real-time processors, can easily add them
- Arm Neon enables high-performance ML as standard
  - Single Instruction Multiple Data (SIMD) greatly accelerates ML
- ML processors, FPGAs, or dedicated hardware easily integrated

www.arm.com/products/silicon-ip-cpu

Cortex-R82: Enabling Next-Generation Storage

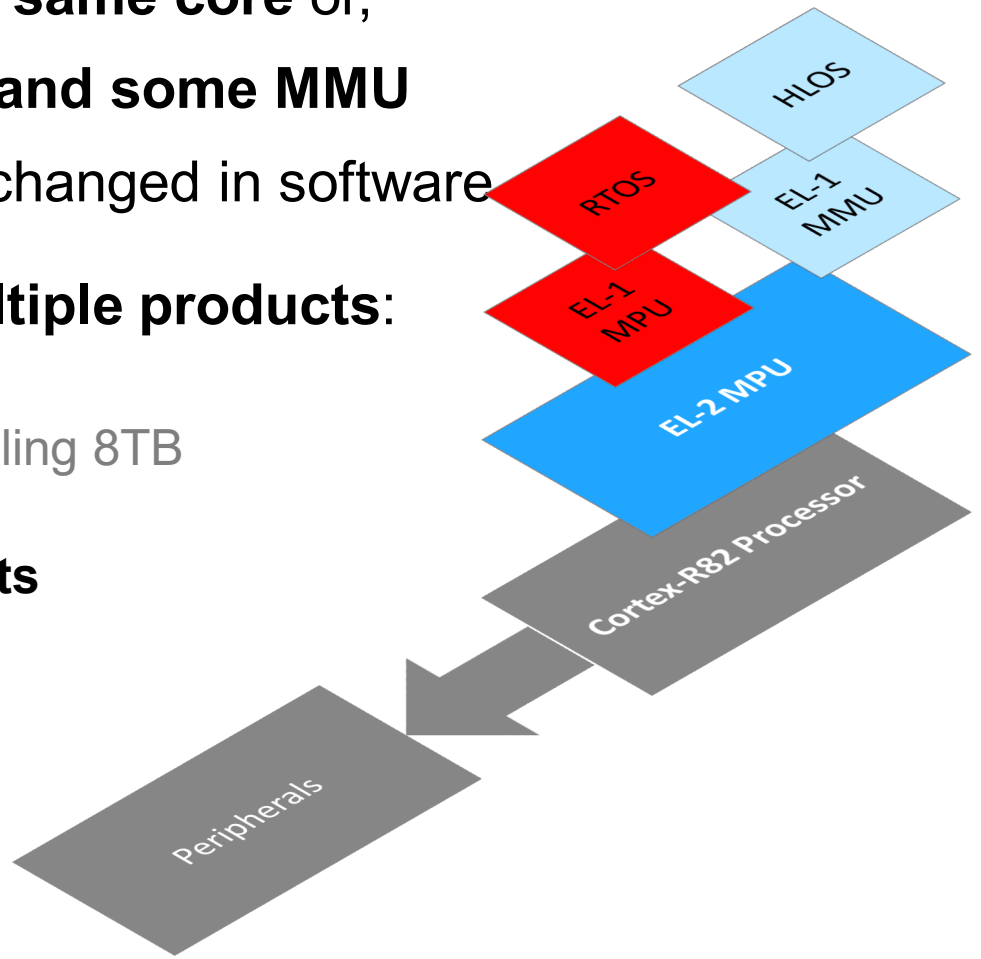


- First 64-bit Cortex-R series processor
  - Large address map for high-capacity
- Hard real-time needed for controller FTL
  - Specialized hard real time features: lowest latencies and consistent performance
- Memory Management Unit
  - Enabling Linux (or other HLOS) support
- Shared coherent memory across the system
  - Between clusters and even across CXL/CCIX
- Advanced Machine Learning support with Neon

Cortex-R82: Real-time MPU + Linux MMU at EL-1

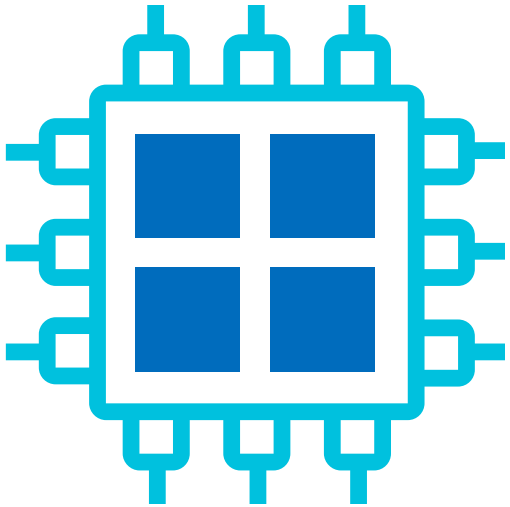
- Separate virtual machines for bare metal/RTOS and MMU contexts
- Enables real-time and Linux to co-exist on **same core** or,
- **Some cores in cluster can be real-time and some MMU**
- Balance of real-time / Linux cores can be changed in software
- Enables a **single controller ASIC for multiple products**:
  1. All real-time eSSD handling 16TB
  2. Half Linux processing / half real-time handling 8TB

Reducing the number of high cost mask-sets

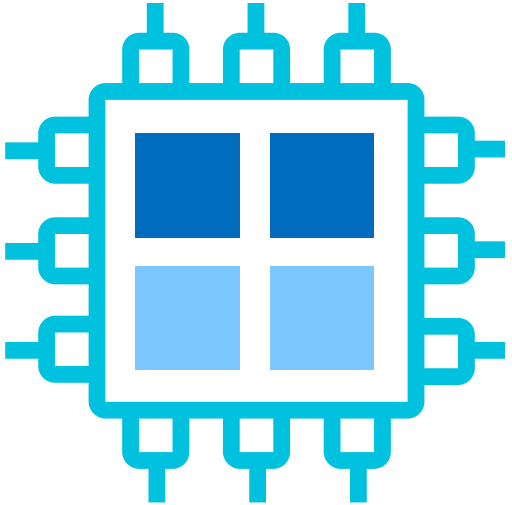


Designing Flexible Controllers with Cortex-R82

One storage controller tape-out for both pure storage and computational storage



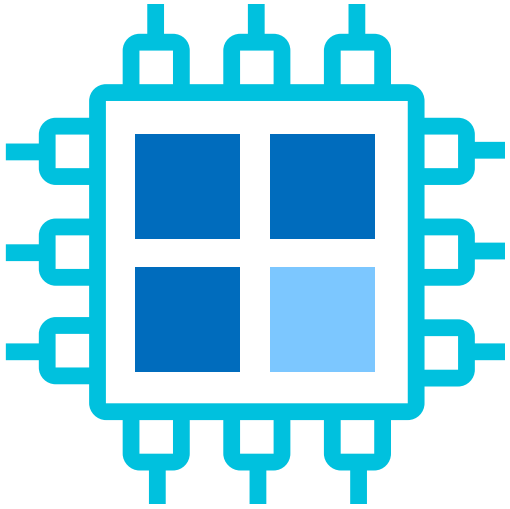
Classic Enterprise Drive



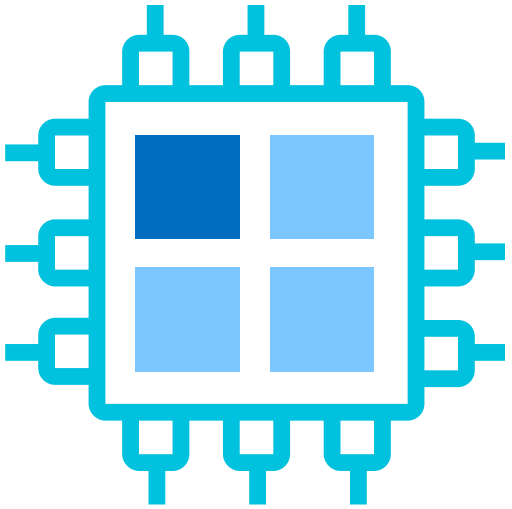
Computational Storage Drive

Flexibility to Change the Balance of Workload

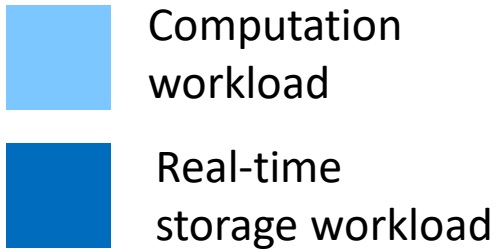
Dynamically adjusting the workload balance on the controller based on external demands



Storage Balance



Computation Balance



Controller Software Architecture

Accelerating Innovation for Storage Developers

Cloud-native software technologies speed development and accelerate innovation

The ability to run Linux on a storage device opens up a range of software tools and technologies to developers

Arm enjoys full support for many Linux distributions and open-source software technologies

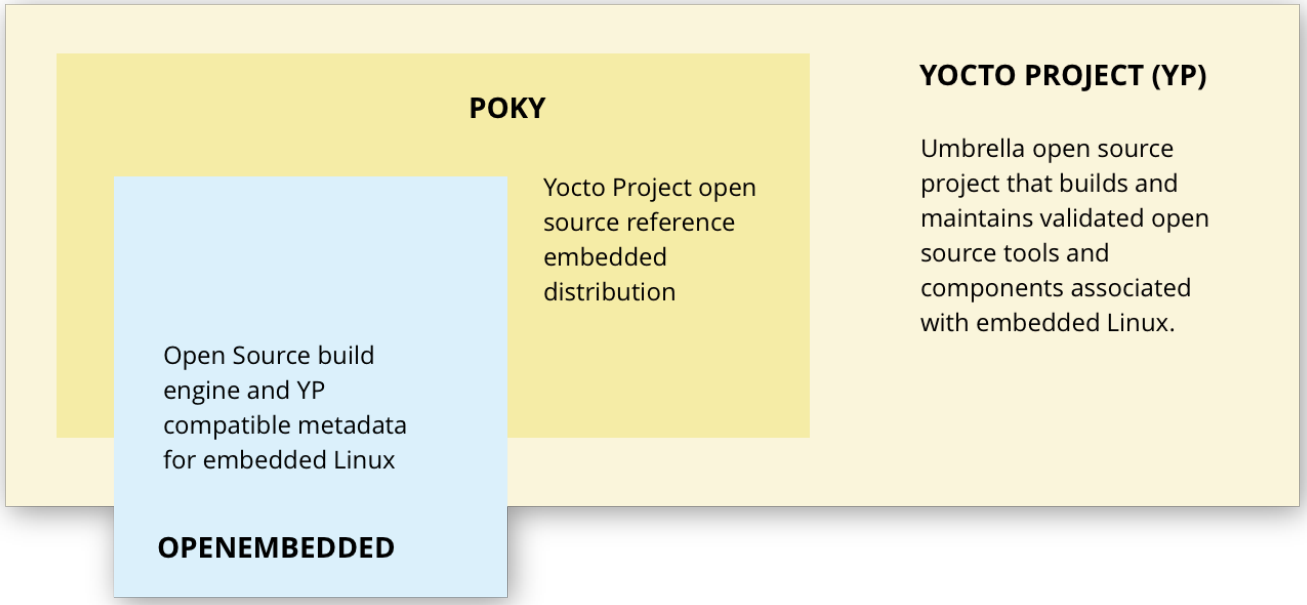
Familiarity with these tools and technologies will accelerate innovation, development and deployment

The Yocto Project

Not a distro! Helps developers build embedded distributions easily

Utilizes a layer model for collaboration and code reuse

Modularity enables optimization of speed, footprint and memory utilization



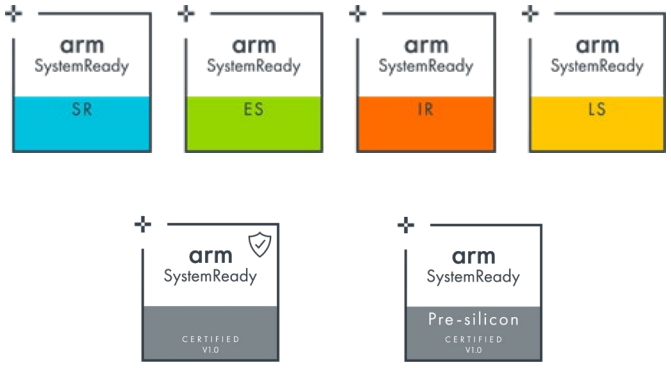
Official documentation at yoctoproject.org

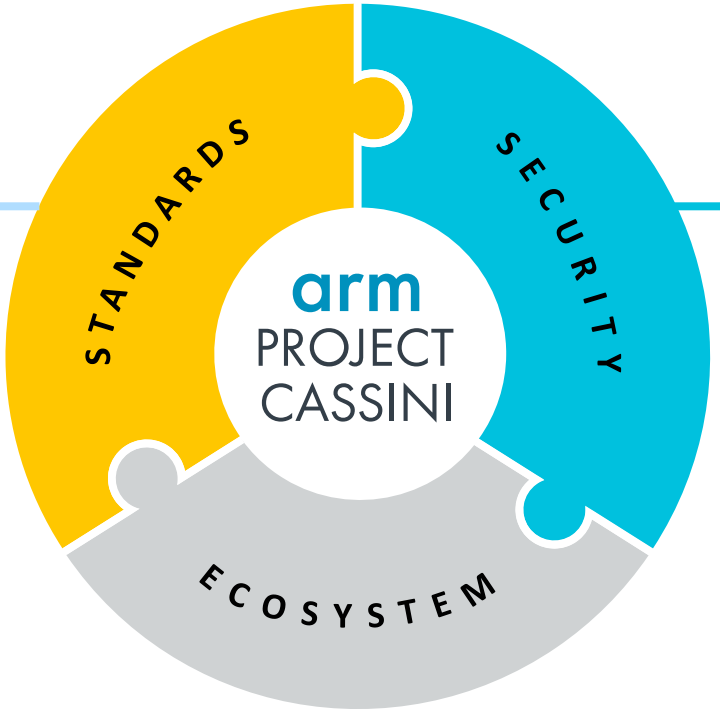
Project Cassini

Ensuring a cloud-native experience across a diverse and secure edge ecosystem

arm SystemReady

- Hardware, firmware specifications
- Certification program





psacertified™

PARSEC

- Security Certification program
- Open API for cross-platform security services

Cloud Native Stacks

- Edge Solution Reference Implementations

Unifying Lower Software Stacks Across Devices

Reducing fragmentation, providing host-device design-patterns

Builds on SystemReady

- Profiles for all device types: embedded to server-class
- Standard device management software (ACPI, device-tree, UEFI, LinuxBoot)

Software “design patterns” for devices

- Software layers above discovery and management to interoperate with host
- Conforming to emerging standard host-accelerator interfaces defined by standards bodies
- Interface specific device functionality allowing differentiation for vendors

Cassini leverages SystemReady compliance for easy deployment of OS and application software

Arm SystemReady Bands Applied to Device Types

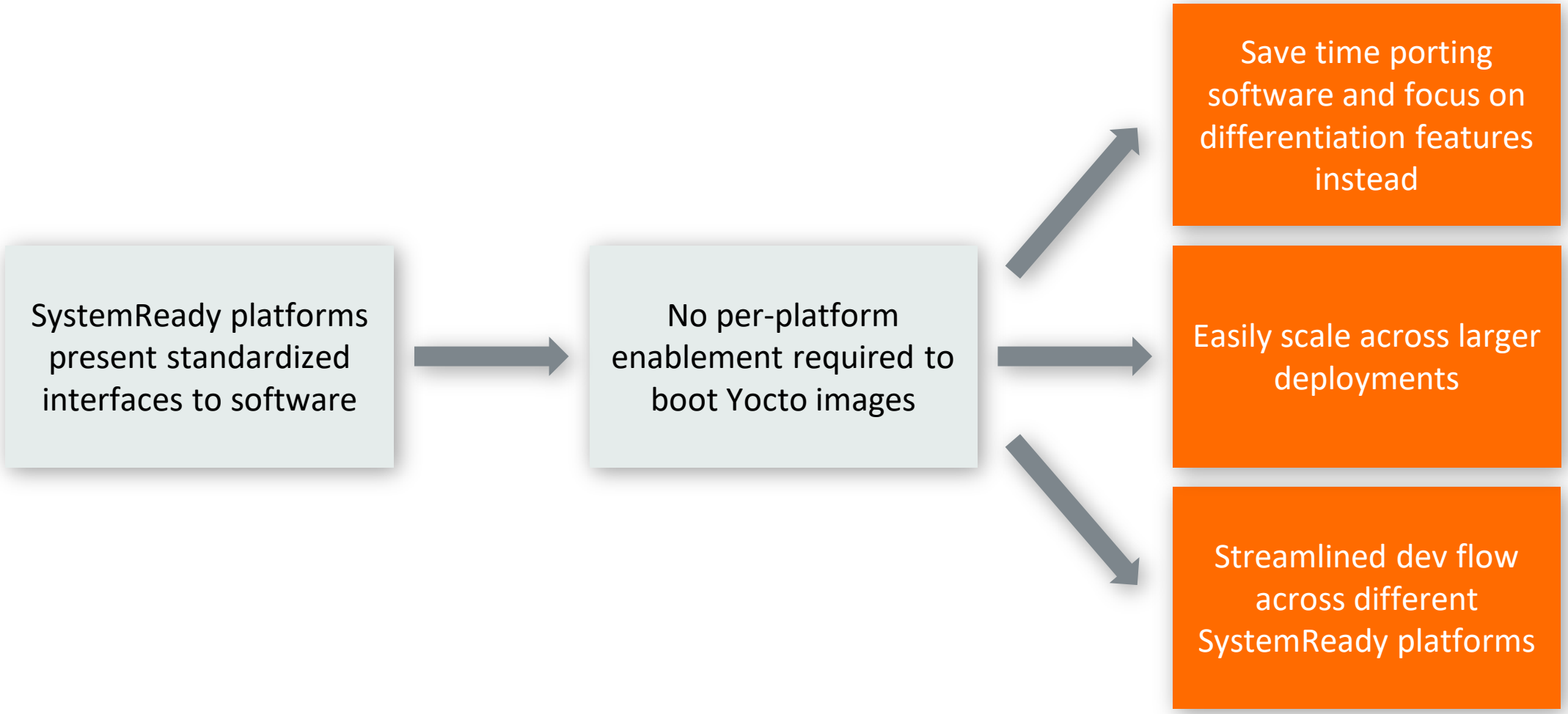
SystemReady LS	SystemReady IR	SystemReady ES	SystemReady SR
DPU or Server CXL Accelerators	Edge CXL Accelerators	Computational Storage Drives	DPU or Server CXL Accelerators
OVS, DPDK, SPDK	5G Data Filtering and Aggregation, Security	Filtering, Search	OVS, DPDK, SPDK
K3S, K8S, OpenUCX	K3S	K3S, ArmNN	K3S, K8S, OpenUCX
Linux, Windows, VMware ESXi	Linux, Windows IoT	Linux, Windows IoT, VMware ESXi	Linux, Windows, VMware ESXi
ACPI + SMBIOS	UEFI + Device Tree	UEFI + ACPI + SMBIOS	UEFI + ACPI + SMBIOS
SR Recipe: ✓ BSA ✓ SBSA ✓ LBBR	SR Recipe: ✓ BSA ✓ EBBR	SR Recipe: ✓ BSA ✓ SBBR	SR Recipe: ✓ BSA ✓ SBSA ✓ SBBR
64b Arm	32b or 64b	64b Arm	64b Arm

Device Management Functionality (OOB, Redfish e.g.)

LEGEND

- Customer Function
- Applications
- Middleware
- OS
- Firmware
- SR Recipe
- Arm HW IP

Combining SystemReady and Yocto



Leverage Arm Cloud Native Software Ecosystem

Drone.io

Travis CI

GitLab

Jenkins

GitHub Actions

Azure Pipelines

AWS CodePipeline

CI/CD

WORKLOADS

LANGUAGE & LIBRARY

CONTAINERS & VIRTUALIZATION

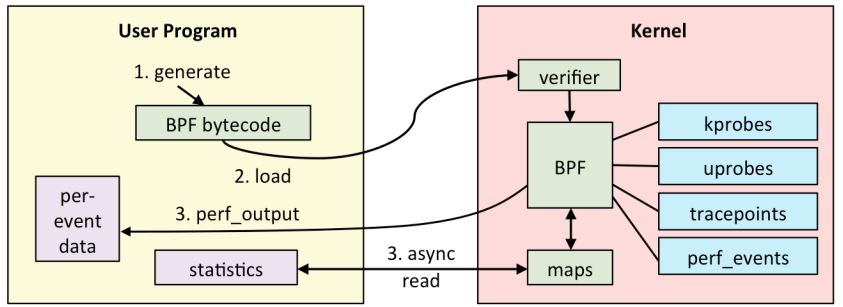
OPERATING SYSTEM

NETWORKING

eBPF

Significant interest in eBPF

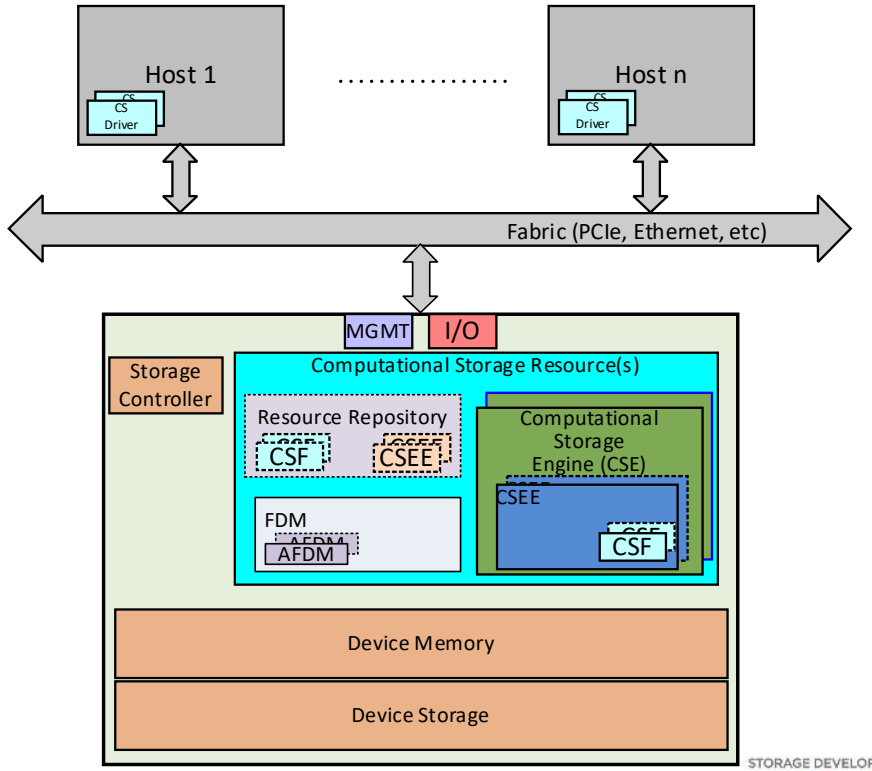
- Runs on Arm cores
  - Often in kernel space, but can run in user space as well
  - Potential performance benefits to running in kernel space
- Challenges with eBPF
  - Code is less flexible (than C, for example), especially in kernel space
  - Security is a key concern
- Arm actively optimizing eBPF on Arm based CSDs



Combining CXL and Computational Storage

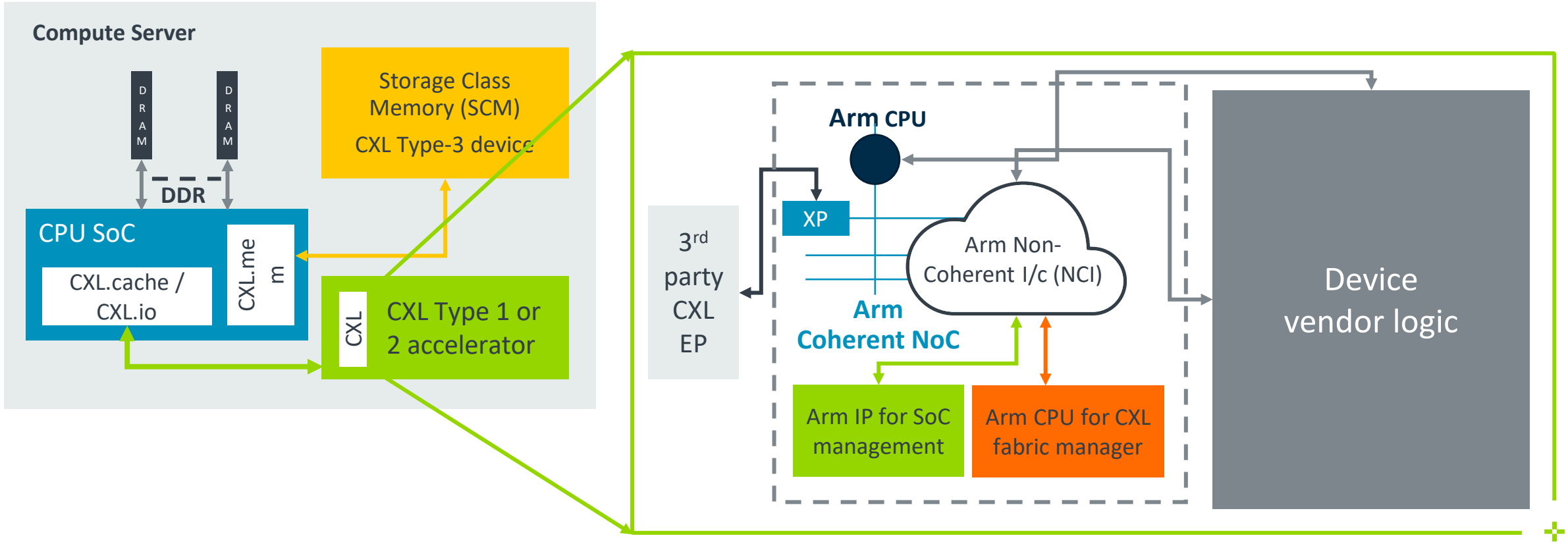
A CSD is an Accelerator (with storage)

- CXL designed to interface to accelerators
- CXL gaining momentum as an interface for coherently connected devices
  - Memory disaggregation with CXL receiving a lot of interest and attention
- Persistent memory blurring the line between memory and storage
- CXL is a natural technology to enhance CSD capabilities
  - CXL will accelerate deployment of CSDs at scale



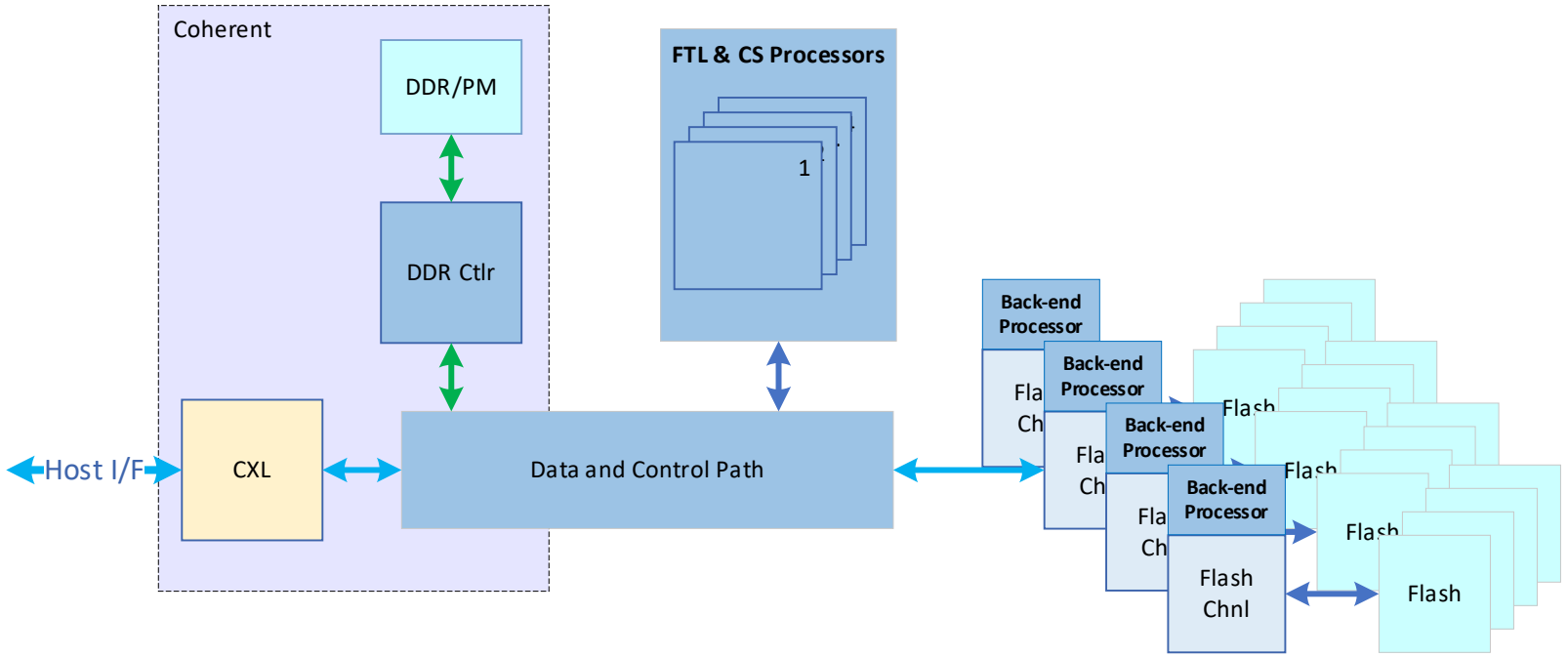
Computational Storage Drive (CSD)

CXL CSD Accelerator



Evolution of CSD to CXL

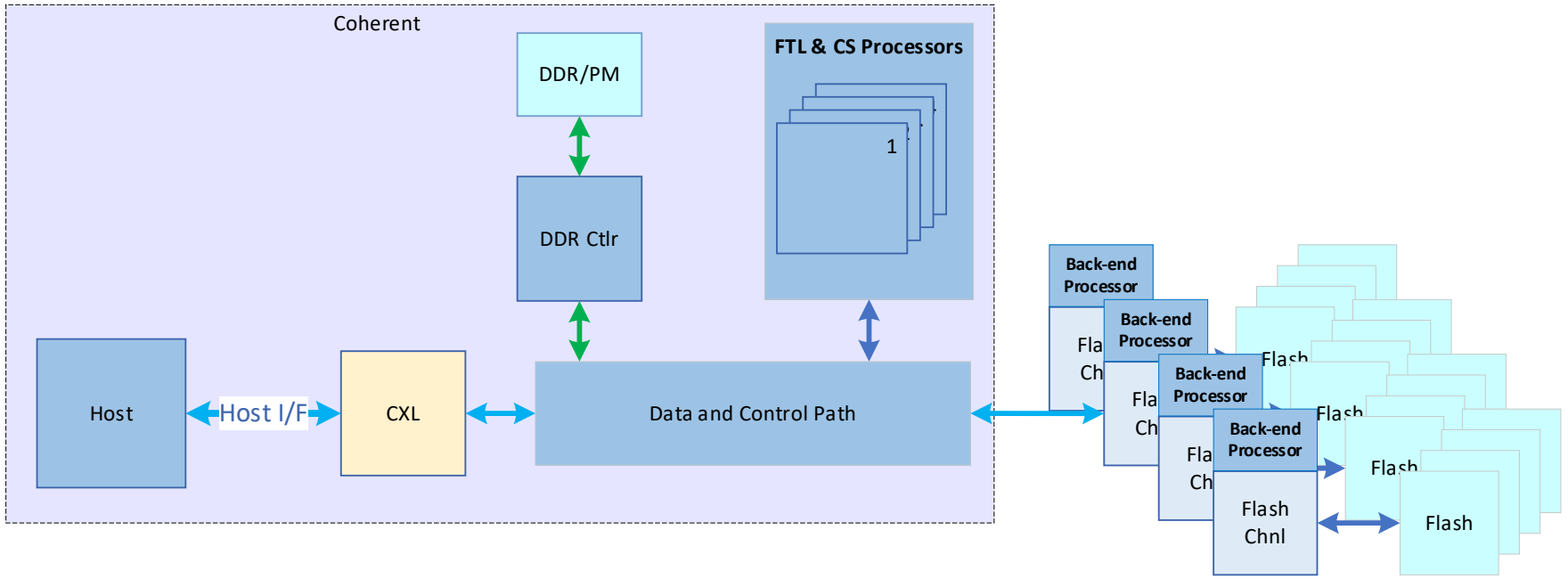
Step 1



- Type 3 CXL device
- Allows the host and drive to share DDR

Evolution of CSD to CXL

Step 2



- Type 2 CXL device with memory
- All the capabilities of step 1
- Coherent interaction with Compute

Combining CXL and Computational Storage

A CSD is effectively an accelerator closely connected to storage

- CXL is designed for connecting to accelerators
- Type 3 initially and evolve to type 2

Dispatch or migrate workloads to the CSD

- Improved efficiency downloading workloads directly to the device
- Device compute and host can run the same OS and software stack
- Migrate workloads coherently from one Arm device to another
  - Run workloads where the most benefit will be realized

Leverage CXL and CSD Technologies

- Split workloads between CSD and host CPUs
  - Split FTL/ZNS is a current example
- NVMe should run over CXL.io today
  - Could be adapted to leverage CXL or migrate to CXL byte addressing

Conclusion

Conclusion

- Computational storage is happening now
  - Many benefits to Computational Storage
  - SNIA specs are progressing
- CSD hardware and software require flexibility
  - Controller designs need to enable multiple products/workloads
  - Software stacks need to be easy to develop and leverage industry standards
  - Dynamically balancing real-time and Linux capabilities
- CSDs can leverage the CXL protocol
  - CSD is an accelerator
- Call to Action
  - Lets talk about your CSD or CXL challenge

Please take a moment to rate this session.

Your feedback is important to us.

