SNIA DEVELOPER CONFERENCE

SDC 24

BY Developers FOR Developers

September 16-18, 2024
Santa Clara, CA

OFA Sunfish

New Applications for Distributed Storage with SNIA Swordfish®

Nathan Hanford

LLNL-PRES-2000486

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Agenda

- Sunfish Overview
- Sunfish and Agent Communications
- Sunfish and Multiple Agents
- Boundary Component Merging from Multiple Agents
- Status of Sunfish
- Summary and Wrap-up

Sunfish Overview

What is Sunfish?

Composable Disaggregated Infrastructure (CDI)

- CDI enables assigning *pools* of resources to consumers
  - Started with disaggregated storage
  - Moving towards disaggregated memory and disaggregated accelerators
  - Assigned resources may be private to or shared among consumers
- CDI requires one or more interconnecting *fabrics*
  - Disaggregated storage is already supported on several fabrics
    - For example: Infiniband, Ethernet, PCIe and FibreChannel
  - Disaggregated memory requires a memory semantic fabric
- CDI needs to avoid disaggregated management stacks
  - Disaggregated resources come with independent management tools

Sunfish provides a framework for wrangling the multitude of independent management tools behind a single, consistent, standards-based API

Sunfish

- Composable disaggregated infrastructures provide a promising solution to addressing the provisioning and computational efficiency limitations, as well as hardware and operating costs, of integrated, siloed systems. But how do we solve these problems in an open, standards-based way?
- The Sunfish project, a collaboration between DMTF, SNIA, the OFA, and the CXL™ Consortium exists to provide elements of the overall solution, with Redfish® and SNIA Swordfish™ manageability providing the standards-based interface.
- Sunfish is designed to configure fabric interconnects and manage composable disaggregated resources in dynamic High Performance Computing (HPC) infrastructures using client-friendly abstractions.
- This presentation will highlight an open, standards-based approach to composable resource management for disaggregated infrastructures through Sunfish.

HPC Applications

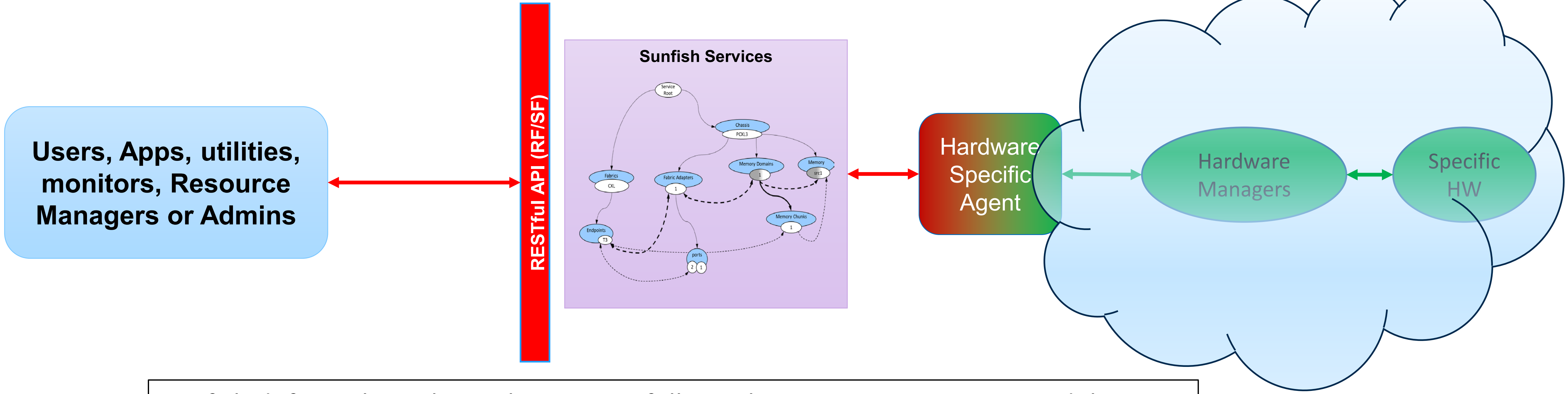
- Integration with Flux as a job manager to allocate disaggregated resources
  - Will allow allocation of fast near-node storage appliances for future HPC systems
  - Could also allow similar integration of Fabric-Attached Memory (FAM)
  - Looking to allocate AI/ML accelerators between HPC clusters using different fabrics
  - Allow users to get stats on how their code actually ran on the hardware/fabric/accelerator

The Sunfish Objective in Visual Form

Sunfish Clients see abstracted Fabric Attached Resource objects

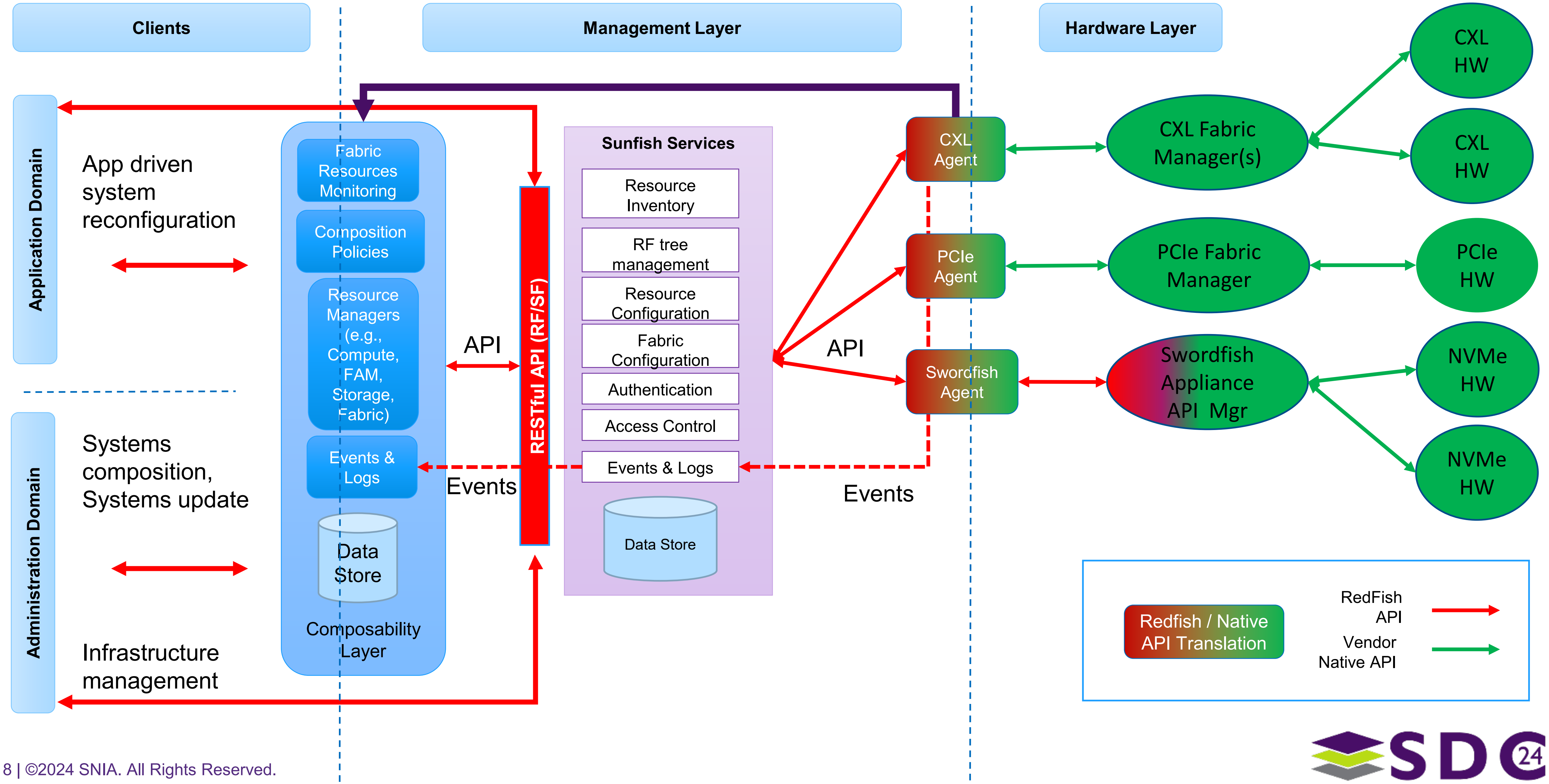
Sunfish Services manages the Redfish models of all resources from multiple hardware Agents

Sunfish Agents hide the hardware specifics by creating appropriate Redfish models of resources



Sunfish defines the policies that Agents follow when creating resource models so that Clients know how to interpret and manipulate them

The Sunfish Open Fabric Management Framework



Sunfish Hardware Agents

- Many different hardware manufacturers can easily create agents for Sunfish.
- Sunfish is based on Redfish, so agent creation is straightforward for other Redfish-based implementations.
- Our alpha reference agent emulator is available on Github here: https://github.com/OpenFabrics/sunfish_agent_reference

Sunfish Agents and Integration

Swordfish Integration

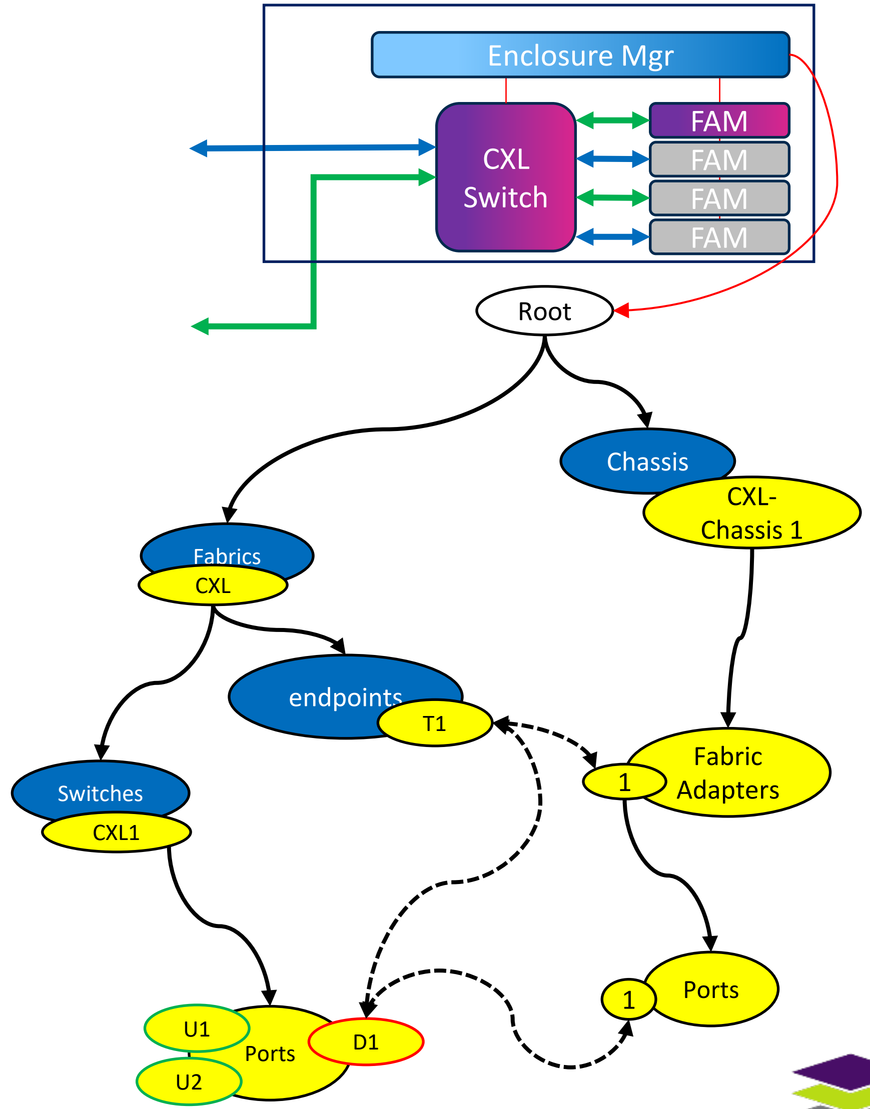
- Sunfish will soon have a Swordfish agent that registers with Sunfish and uploads descriptors.
- The Swordfish Agent will manipulate the resources, and will be a wrapper around the existing Swordfish API Emulator.
- Our alpha Swordfish API Emulator for Integration is here: https://github.com/OpenFabrics/sunfish_agent_reference/tree/rh-cxl-agent

Merging Multiple Agent Resource Trees

- Sunfish enables integration of heterogeneous fabrics with different resource description frameworks.
- The two important problems to solve when working with multiple Agents:
  - Sunfish must detect and resolve Redfish URI namespace conflicts
  - Sunfish must detect and resolve Boundary Component duplicates

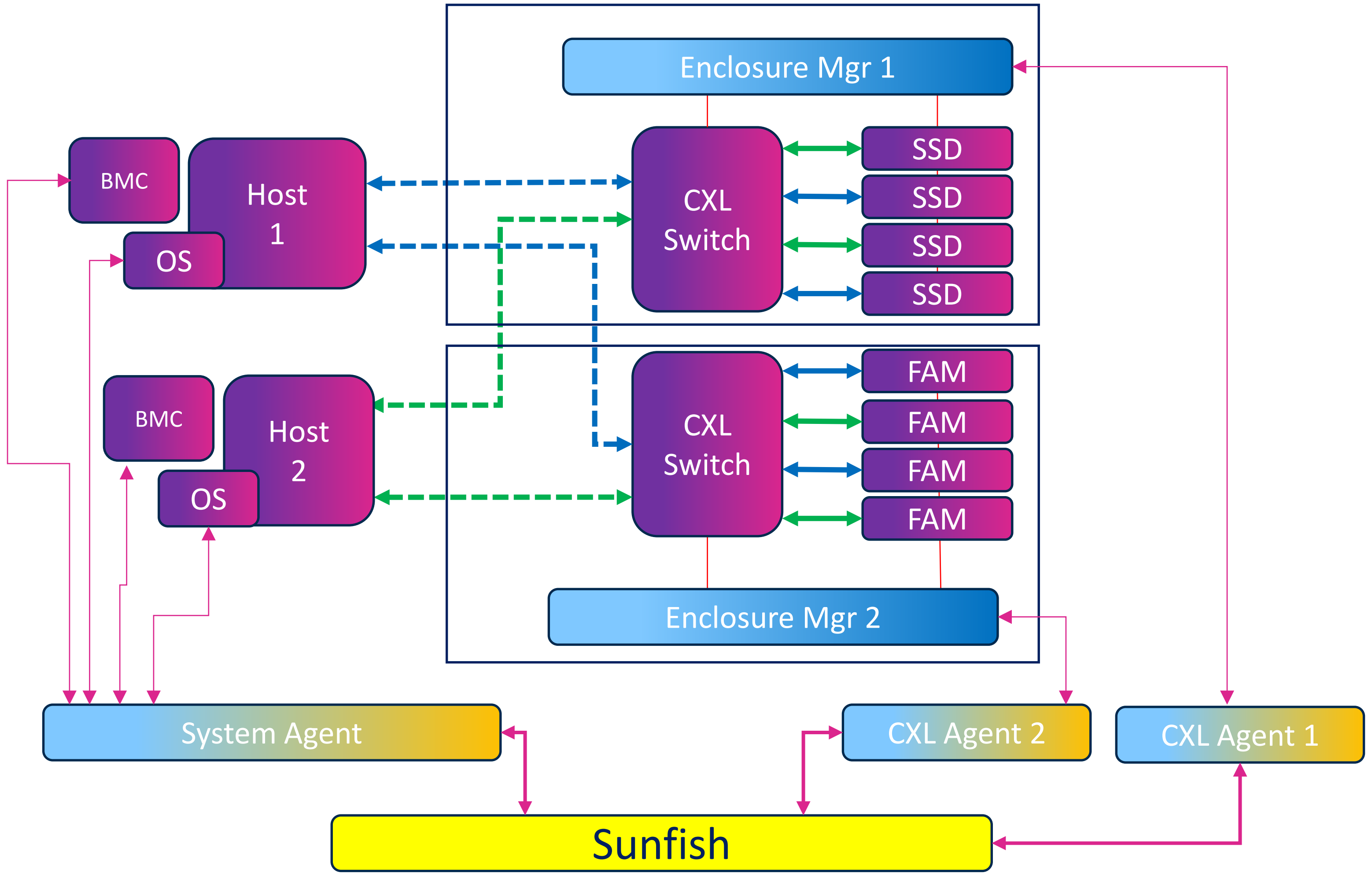
A Simplified Redfish Model of CXL FAM Appliance

- The Enclosure Manager and/or its Sunfish CXL Agent create a Redfish model of the resources within the Appliance
- This model is created within its own Redfish URI namespace
- The Enclosure Manager/Agent recognizes ONLY this URI namespace
- Multiple enclosures may all use the same local URI naming conventions
  - E.g., every instance of such an enclosure may name its enclosure's fabric "/redfish/v1/Fabrics/CXL"



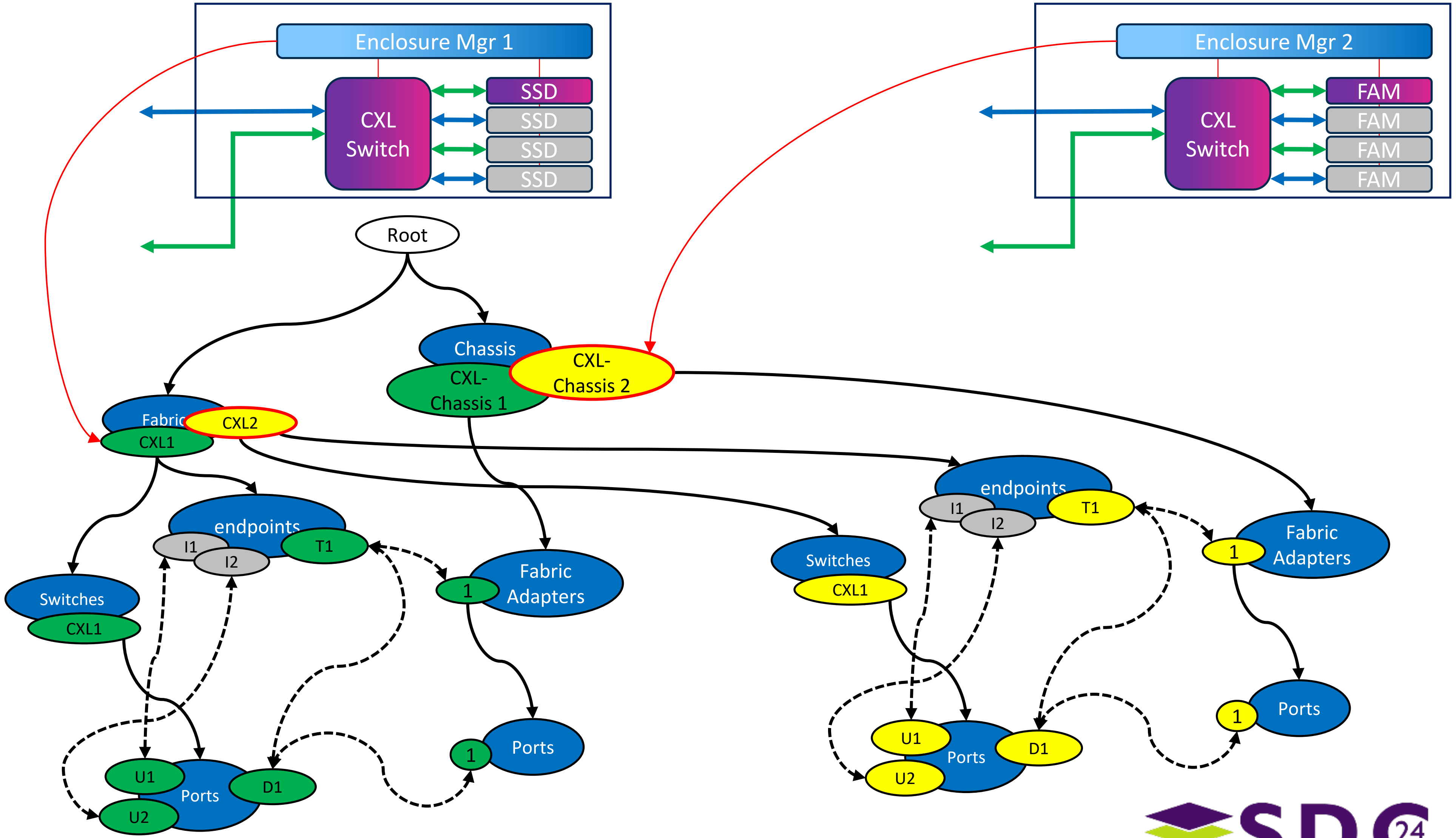
Managing Multiple Agents

- Sunfish Aggregates resources from multiple hardware managers
- Hosts have BMCs and OS
- Storage and FAM Appliances have Enclosure Managers
- Each of these hardware manager / Agent stacks has their own management domain and Redfish URI namespace
- In this topology, each CXL Enclosure is an independent CXL Fabric
  - No Switch-to-Switch links
  - Single level switch topology
- Sunfish Service eliminates duplicate URIs that may occur because the two Enclosure Managers may both use the same Redfish URIs
- Sunfish Service also needs to merge the Redfish URI namespaces of the System Agent and the two CXL Agents
- Dotted links are Boundary Links that cross hardware manager domains
  - E.g. links between a Host and an enclosure switch



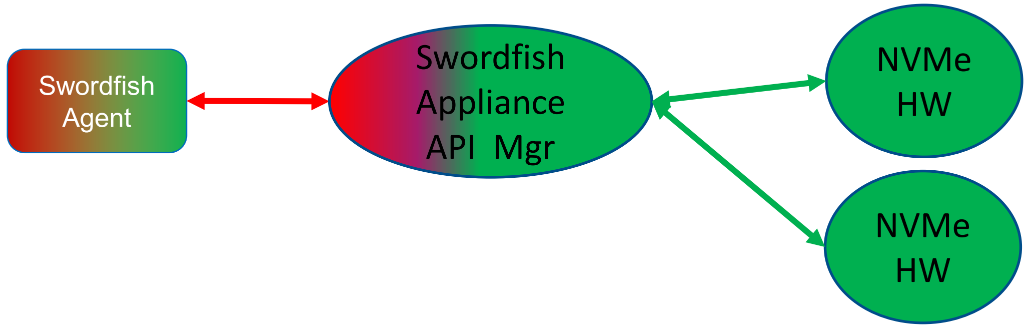
A Merged Redfish Model of 2 Fabric Appliances

- Merging multiple Redfish URI namespaces is relatively easy if the actual hardware / logical instances are independent
- Sunfish keeps track of all URIs it translates from a given Agent's URI namespace to the URI used in the Sunfish URI namespace
- Sunfish must validate that the Fabrics and Chassis involved are completely independent
- Every physical device on every fabric needs to have a hardware ID that is unique within the scope of the Sunfish Services
- Sunfish keeps a list of all hardware IDs and searches for duplicates as each new resource is inserted into the database



Storage Capabilities in Progress

- Management of NVMe native devices along with NVMe appliances through Sunfish's Swordfish Agent
- Aggregation of multiple Swordfish targets
- Adaptation of SNIA Swordfish compliance testing
- Our video tutorial is coming soon here: https://www.openfabrics.org/snia-and-the-ofmf/

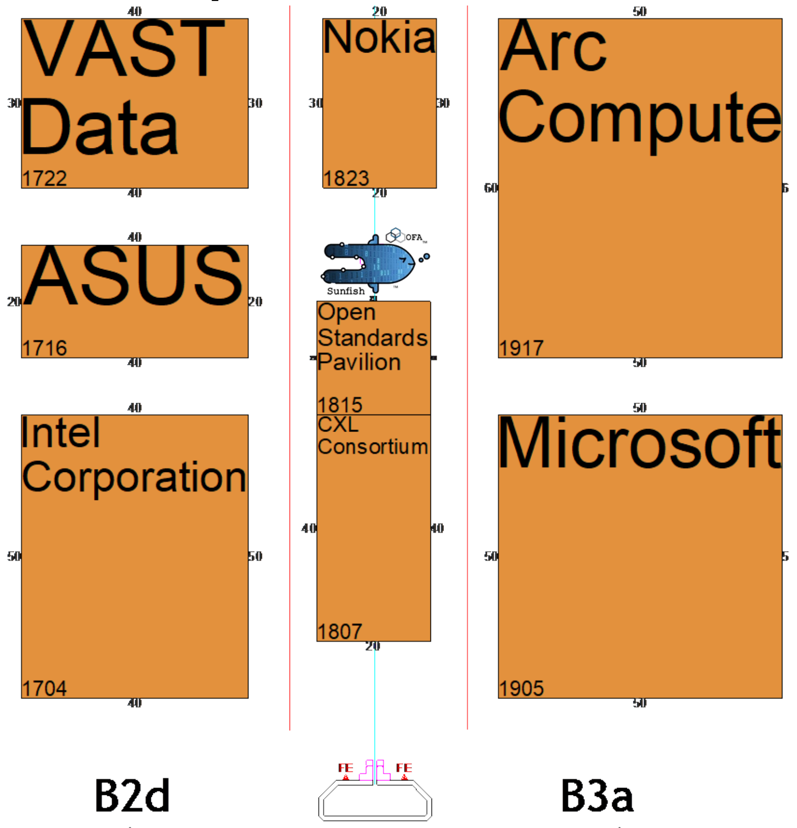


Sunfish Status

- New Releases Planned
  - Customization plugin
  - Redis storage backend
- Soliciting new use cases and hardware
  - Started working on the Flux client side
- Working on client-side resource blocks and resource managers
- Creating a reference CXL Agent

Find us at SC!

- We will be holding a Sunfish BoF at SC this year:
  - Tentatively scheduled for Thu Nov 21, 12:15pm-1:15pm in Room B214
- We will be at the Open Standards Pavilion (Booth 1815)



Summary

- Sunfish allows allocating multiple different Redfish-derived hardware agents from different vendors in the same Sunfish instance.
- Sunfish can also merge different fabrics via its service layer.
- Sunfish is currently targeting Kubernetes via several operators and the Flux scheduling framework for HPC.
- We will have a BoF and several demos during SC.

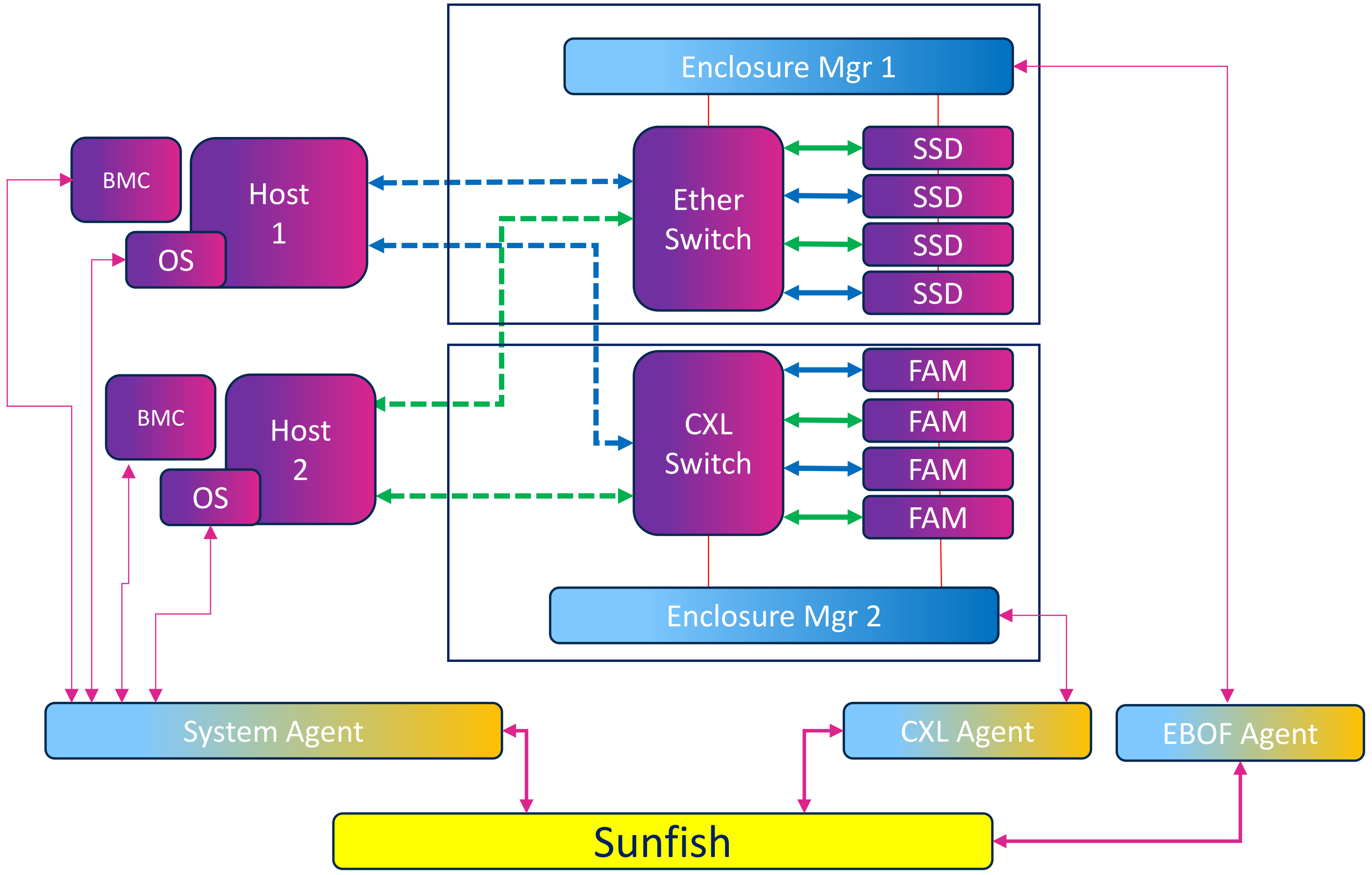
Questions?

Read the docs here!
https://github.com/OpenFabrics/sunfish_docs

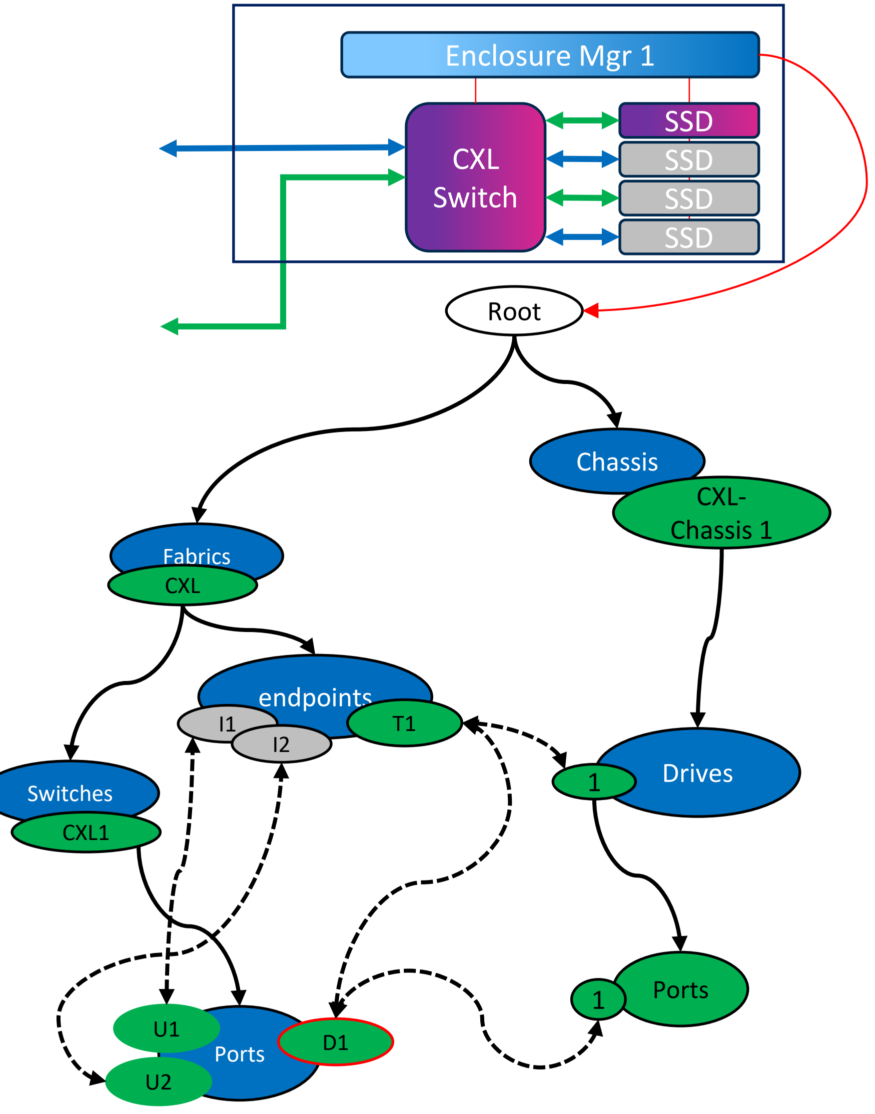
Join our development here!
https://www.openfabrics.org/openfabrics-management-framework/

Backup Slides

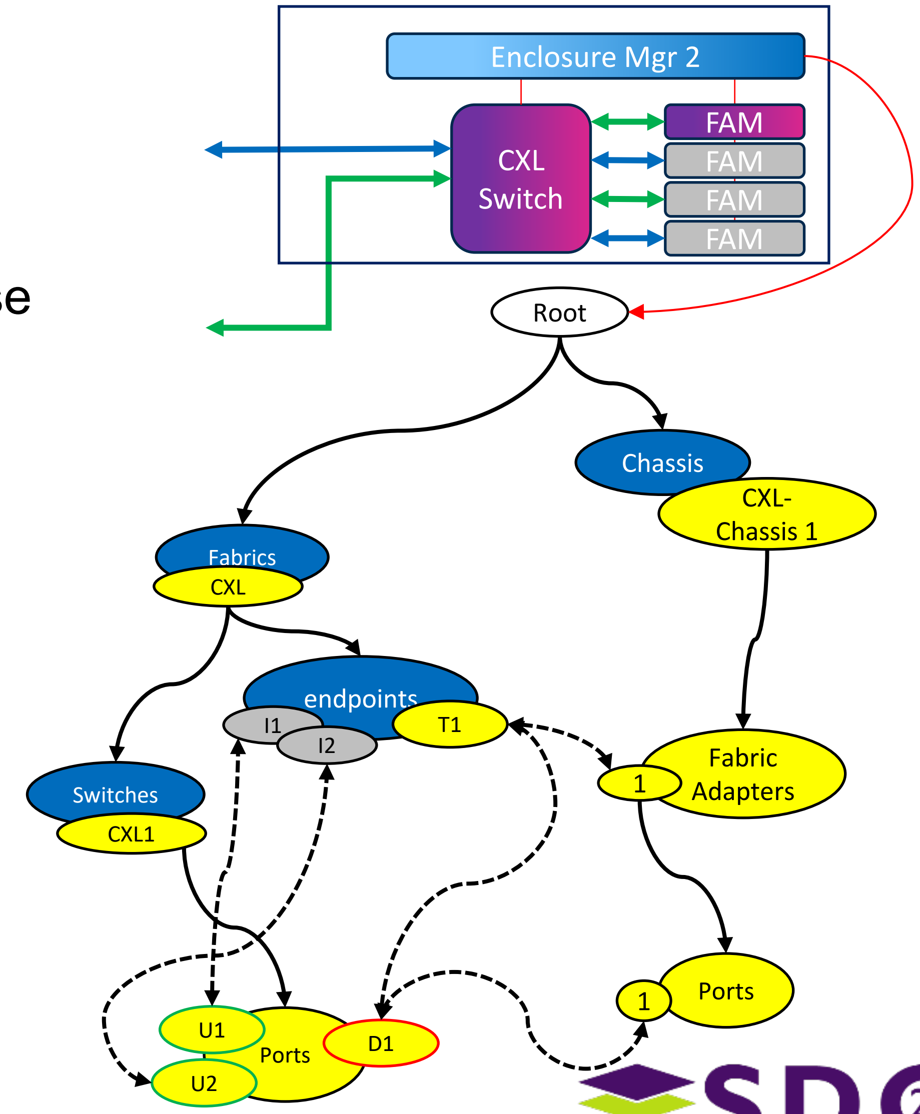
Demo


Enclosure Mgr 1
BMC
Host 1
OS
Ether Switch
SSD
SSD
SSD
SSD
BMC
Host 2
OS
CXL Switch
FAM
FAM
FAM
FAM
Enclosure Mgr 2
System Agent
CXL Agent
EBOF Agent
Sunfish

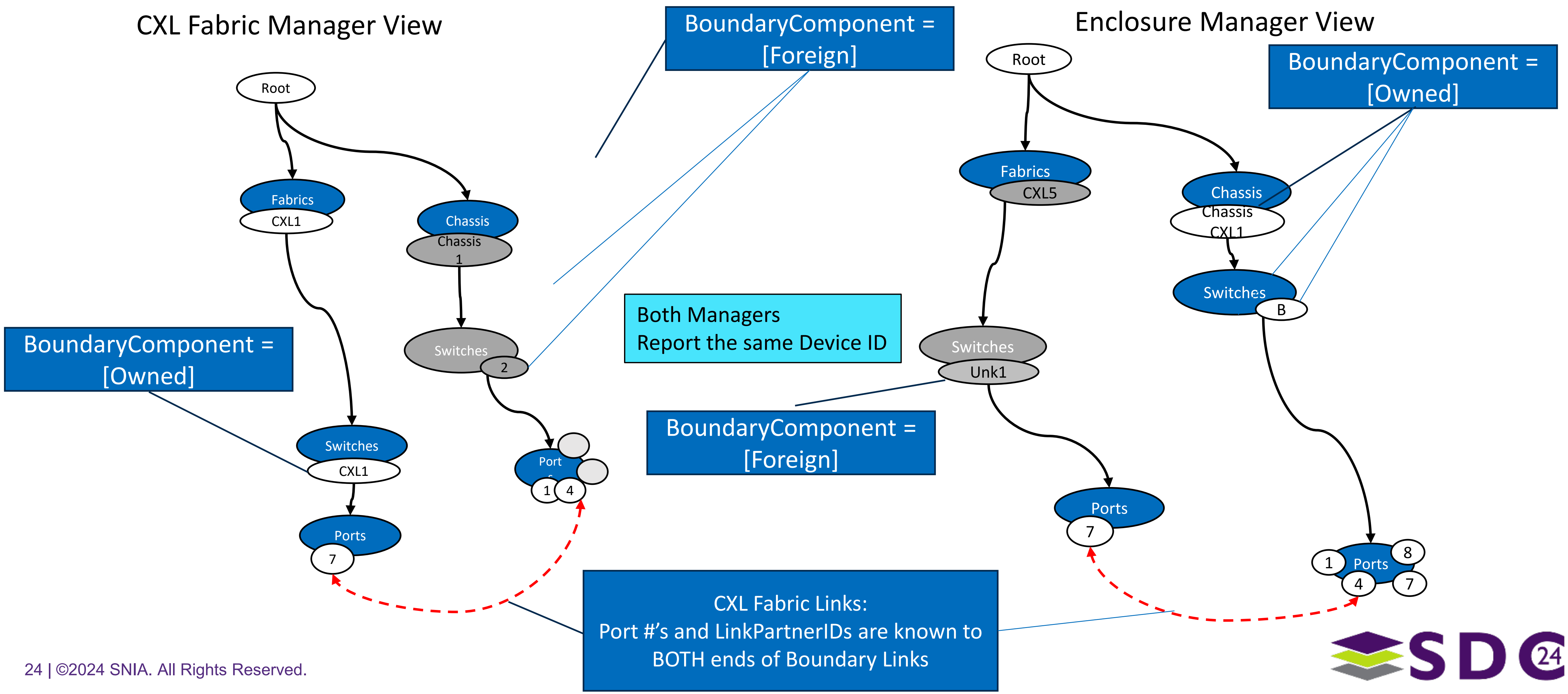
A Simplified Redfish Model of Multiple CXL Appliances



- What if we have two of these enclosures?
- We have two Redfish roots, with two possibly conflicting URI namespaces.
- Sunfish Framework defines how to handle this



Managing CXL Boundary Components



BoundaryComponent = [Foreign]

BoundaryComponent = [Owned]

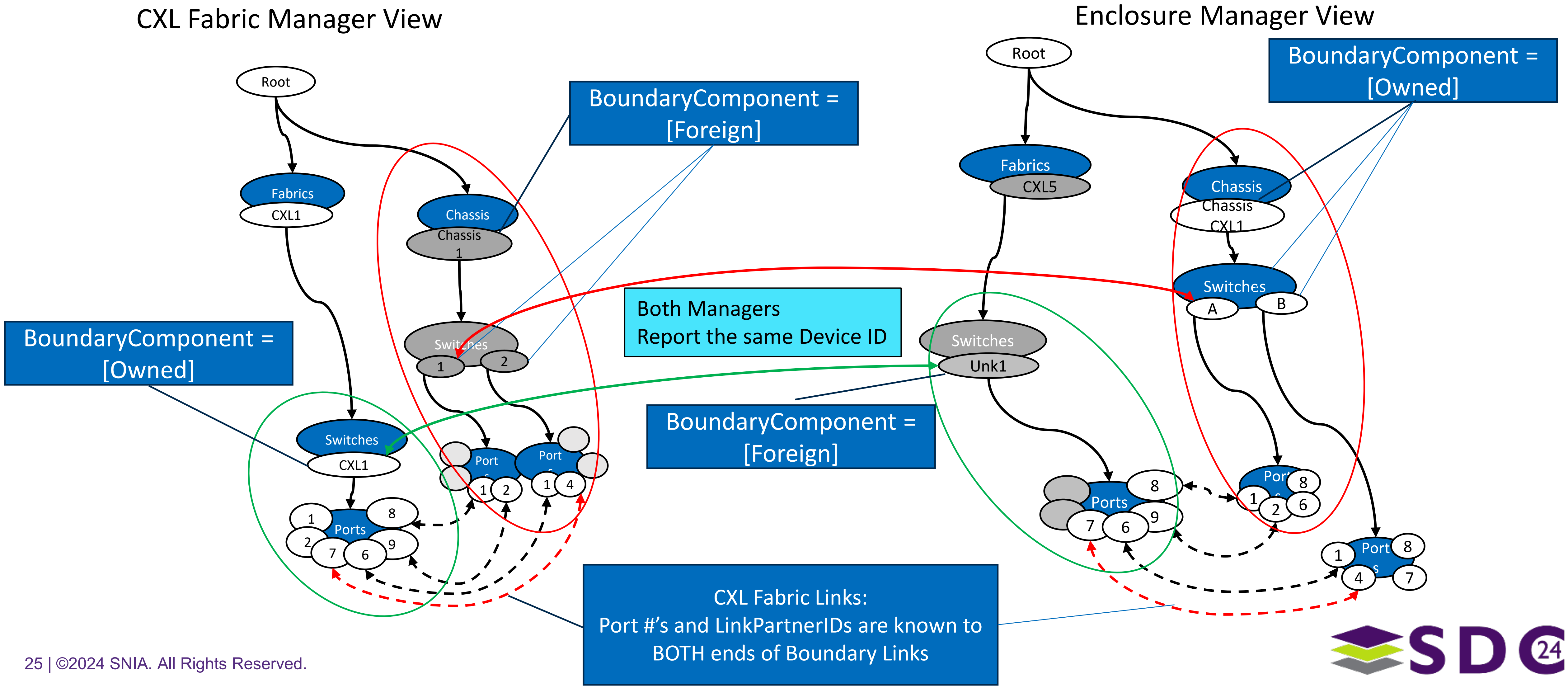
BoundaryComponent = [Owned]

Both Managers
Report the same Device ID

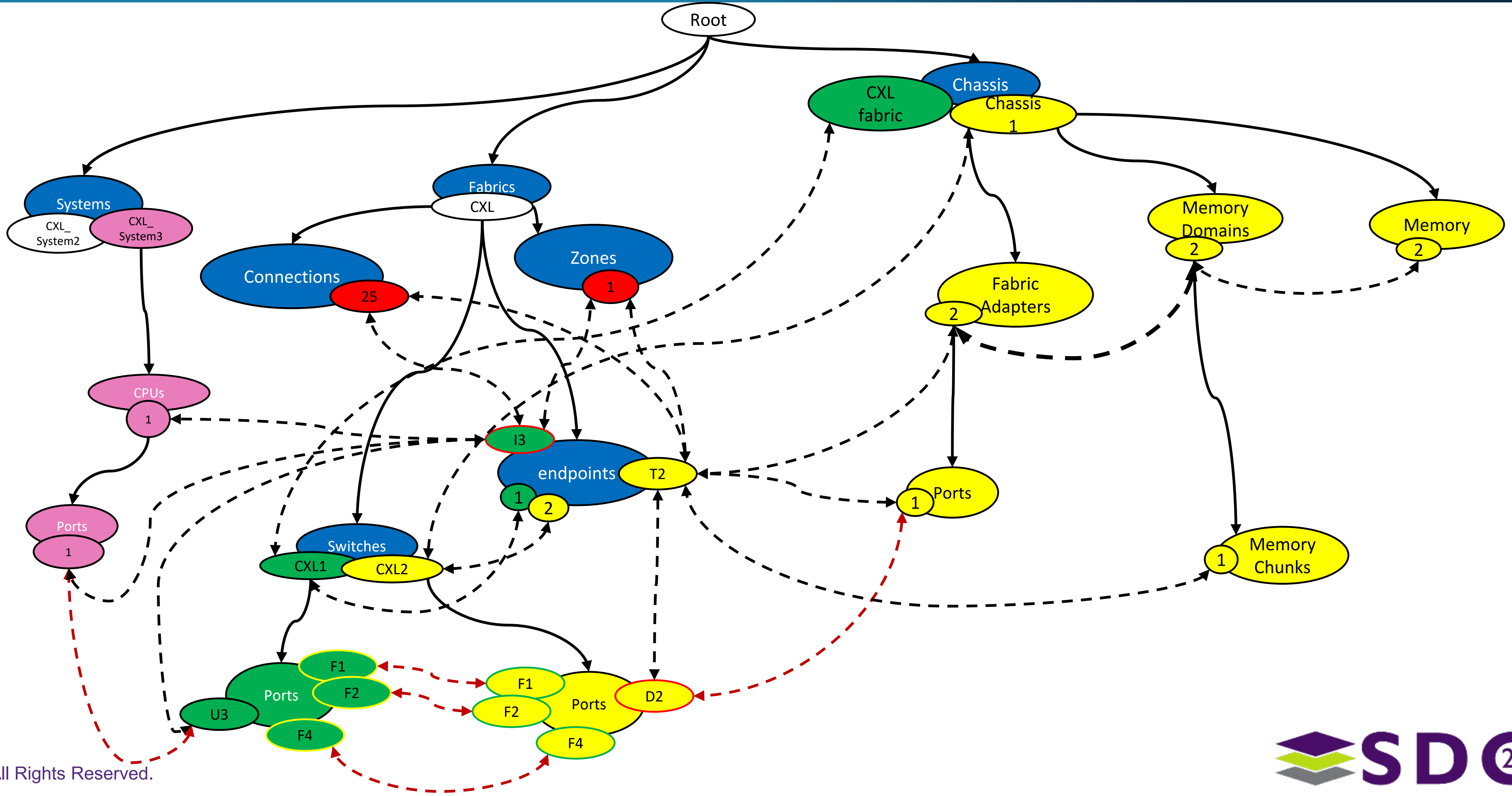
BoundaryComponent = [Foreign]

CXL Fabric Links:
Port #'s and LinkPartnerIDs are known to BOTH ends of Boundary Links

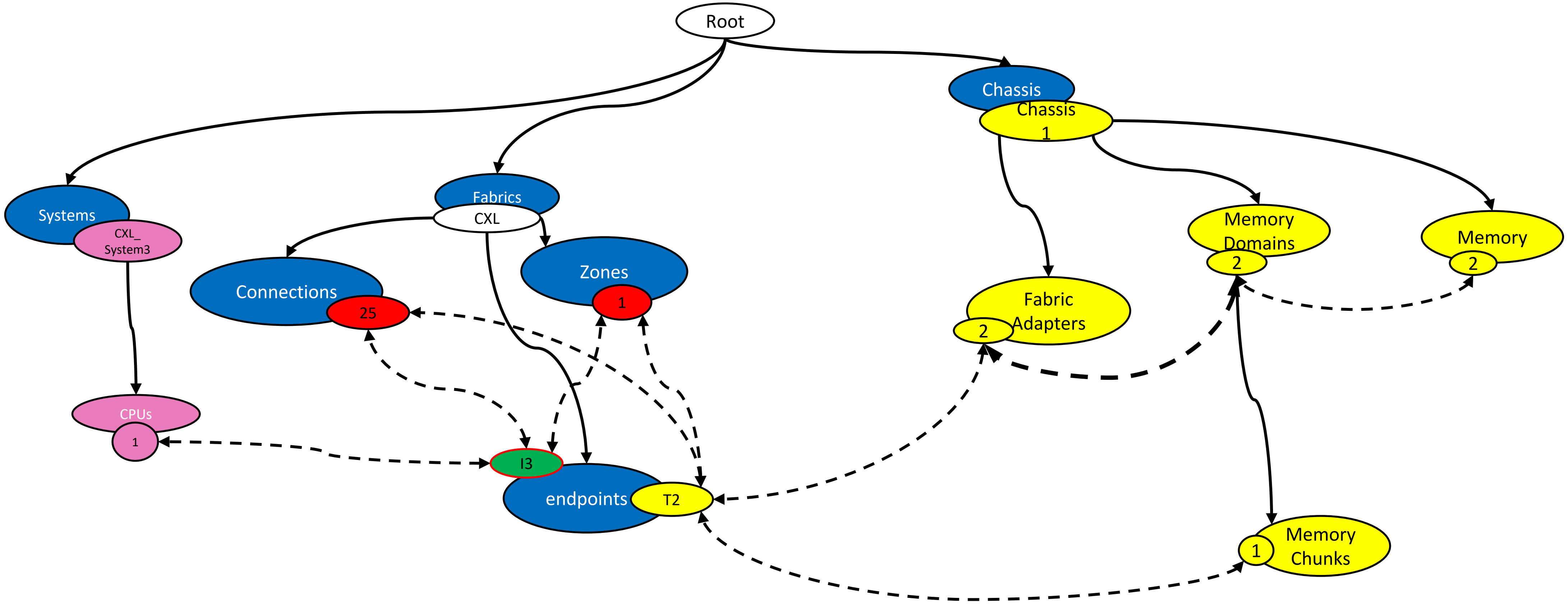
Managing CXL Boundary Components


CXL Fabric Manager View
Root
Fabrics
CXL1
Chassis
Chassis 1
Switches
1
2
Port
1
2
4
Switches
CXL1
Ports
1
2
7
6
9
8
BoundaryComponent = [Foreign]
BoundaryComponent = [Owned]
Both Managers Report the same Device ID
BoundaryComponent = [Foreign]
CXL Fabric Links: Port #'s and LinkPartnerIDs are known to BOTH ends of Boundary Links
Enclosure Manager View
Root
BoundaryComponent = [Owned]
Fabrics
CXL5
Chassis
Chassis CXL1
Switches
A
B
Switches
Unk1
Ports
8
9
7
6
Ports
8
1
2
6
Ports
1
8
4
7

Merging Multiple Agent Resource Trees



Merging Multiple Agent Resource Trees


Root
Chassis
Chassis 1
Systems
CXL_ System3
Fabrics
CXL
Connections
25
Zones
1
Memory Domains
2
Memory
2
Fabric Adapters
2
CPUs
1
I3
endpoints
T2
Memory Chunks
1