2025 OFA Webinar Series

Exploring OFA Sunfish

Brian Pan

Russ Herrell

Independent OFA/Sunfish Contributor

Meet the Presenters: Brian Pan and Russ Herrell

Brian Pan - Founder & Chief Executive Officer at H3 Platform
Brian Pan is the Founder and Chief Executive Officer of H3 Platform, a company specializing in innovative PCIe switch boxes for the HPC industry. Brian's background demonstrates expertise in sales leadership and business development. Prior to H3 Platform, Brian served as Sales Director at Qnap, contributing to the company's retail operations. Before Qnap, Brian held the position of Sales Director at Zyxel Communications Corp, a provider of broadband access solutions. Brian holds a Master of Business Administration from National Chengchi University and a Bachelor's degree from National Cheng Kung University.

Russ Herrell - Distinguished Technologist, Retired from Hewlett Packard Enterprise (HPE)
Russ recently retired from Hewlett Packard Labs of Hewlett Packard Enterprise, having worked for Bill and Dave's companies in Fort Collins, Colorado for his entire career. He has designed 3D graphics engines, management systems for large SMP business and technical computers, and specialized in complete hardware/software solution architectures. Russ continues to work with the Open Fabric Alliance and DMTF organizations on Sunfish, an open standard for management of disaggregated fabric resources

2025 OFA Webinar Series

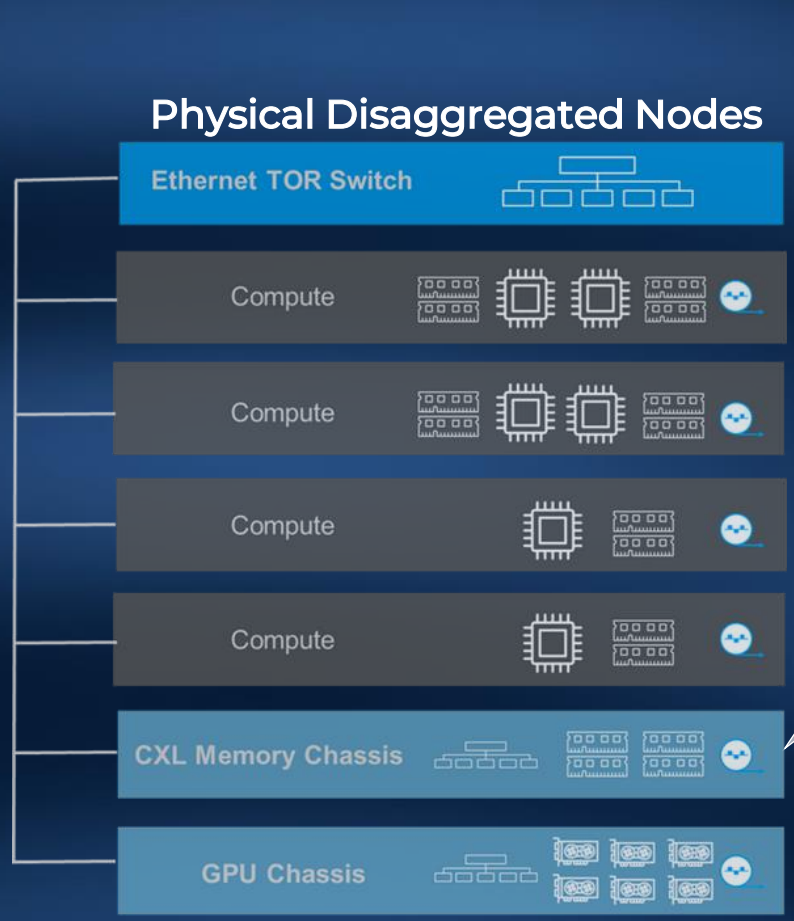
Sunfish for CXL Fabric Management

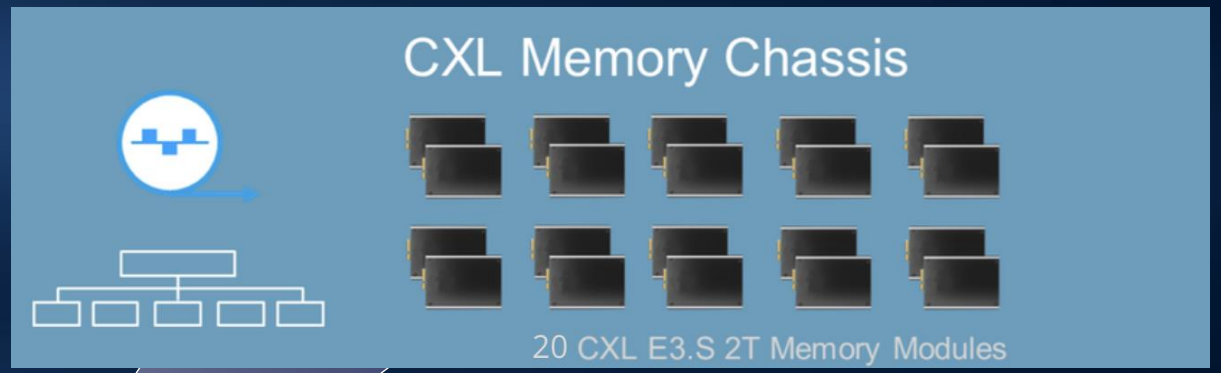
Brian Pan

CEO

Physical Cluster Architecture

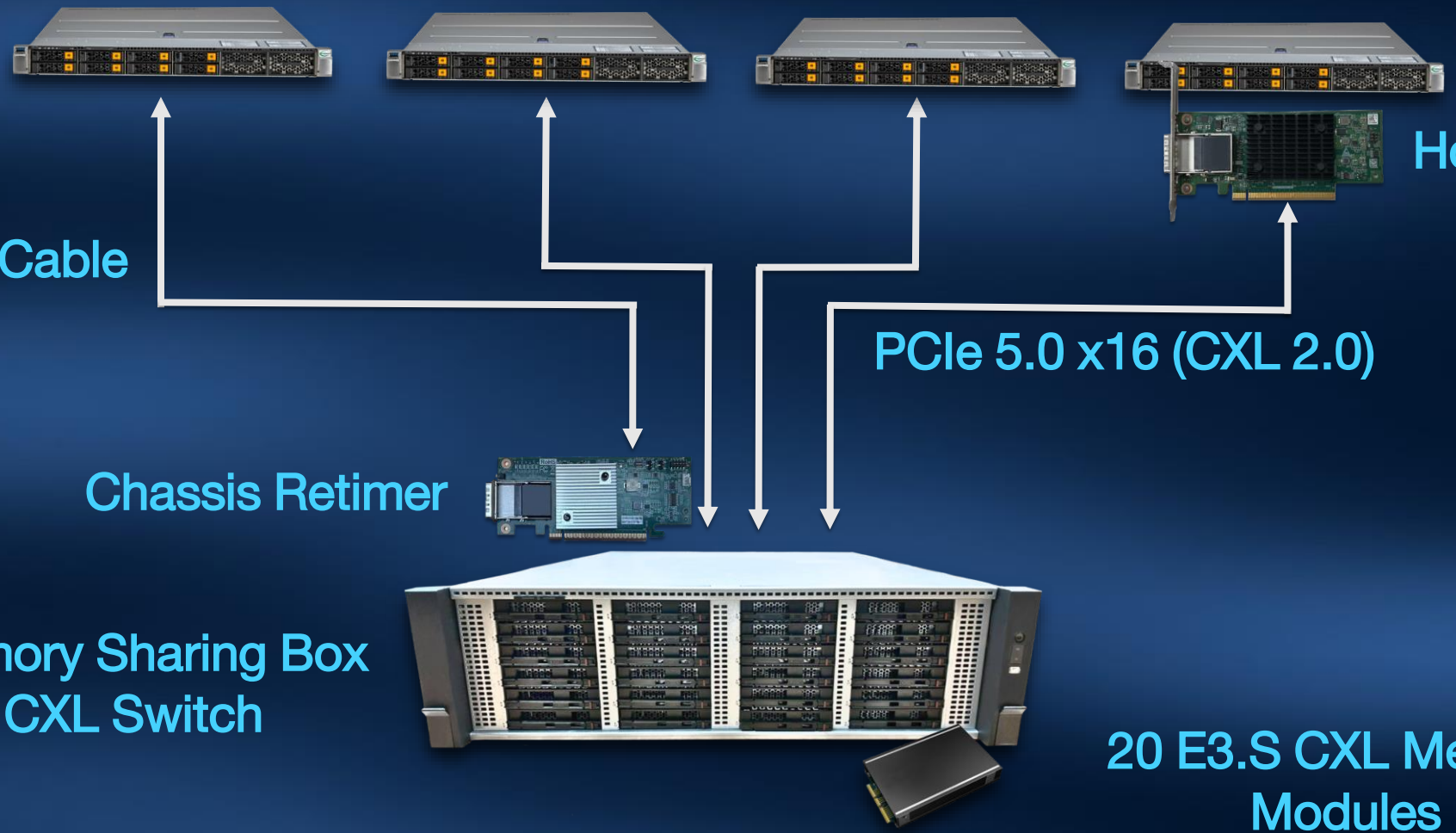
System Architecture of CXL Deployment





Fabric management
+
CXL switch
+
E3.S CXL Memory

Physical Cluster Architecture



Logic Architecture of the Sunfish Management

The Sunfish Objective in Visual Form

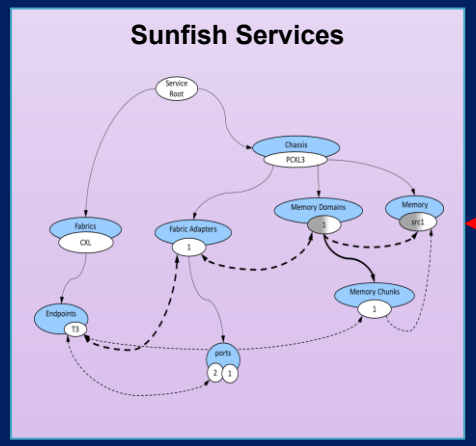
Sunfish Clients see abstracted Fabric Attached Resource objects

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

Users, Apps, utilities, monitors, Resource Managers or Admins

RESTful API (RF/SF)



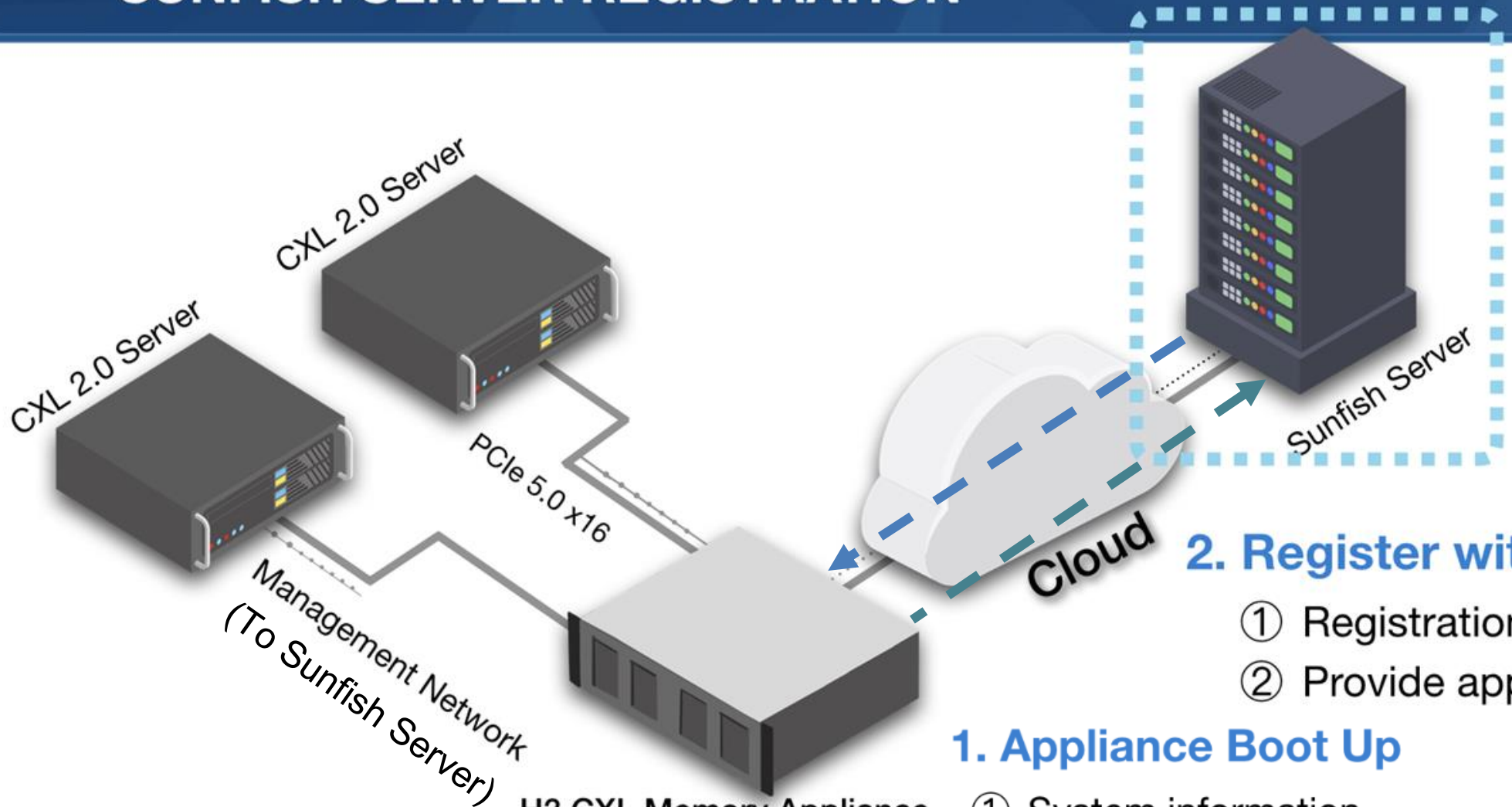
Hardware Specific Agent

CXL Switch Managers- Sunfish

H3 CXL memory appliance

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

SUNFISH SERVER REGISTRATION



2. Register with Sunfish Server

① Registration to Sunfish server

② Provide appliance information

1. Appliance Boot Up

① System information

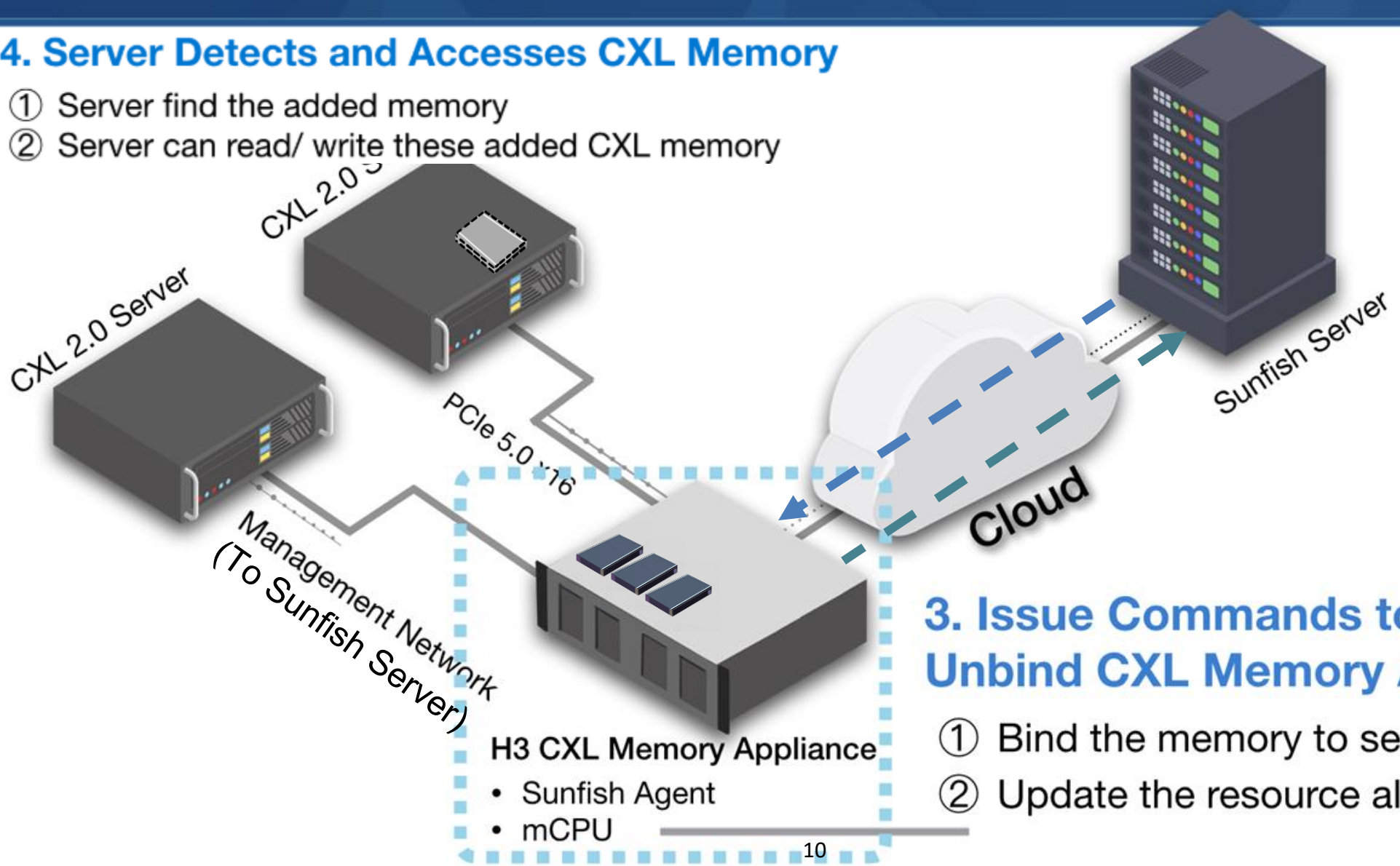
② CXL memory modules in the appliance

③ CXL topology and port configuration

CXL MEMORY BINDING

4. Server Detects and Accesses CXL Memory

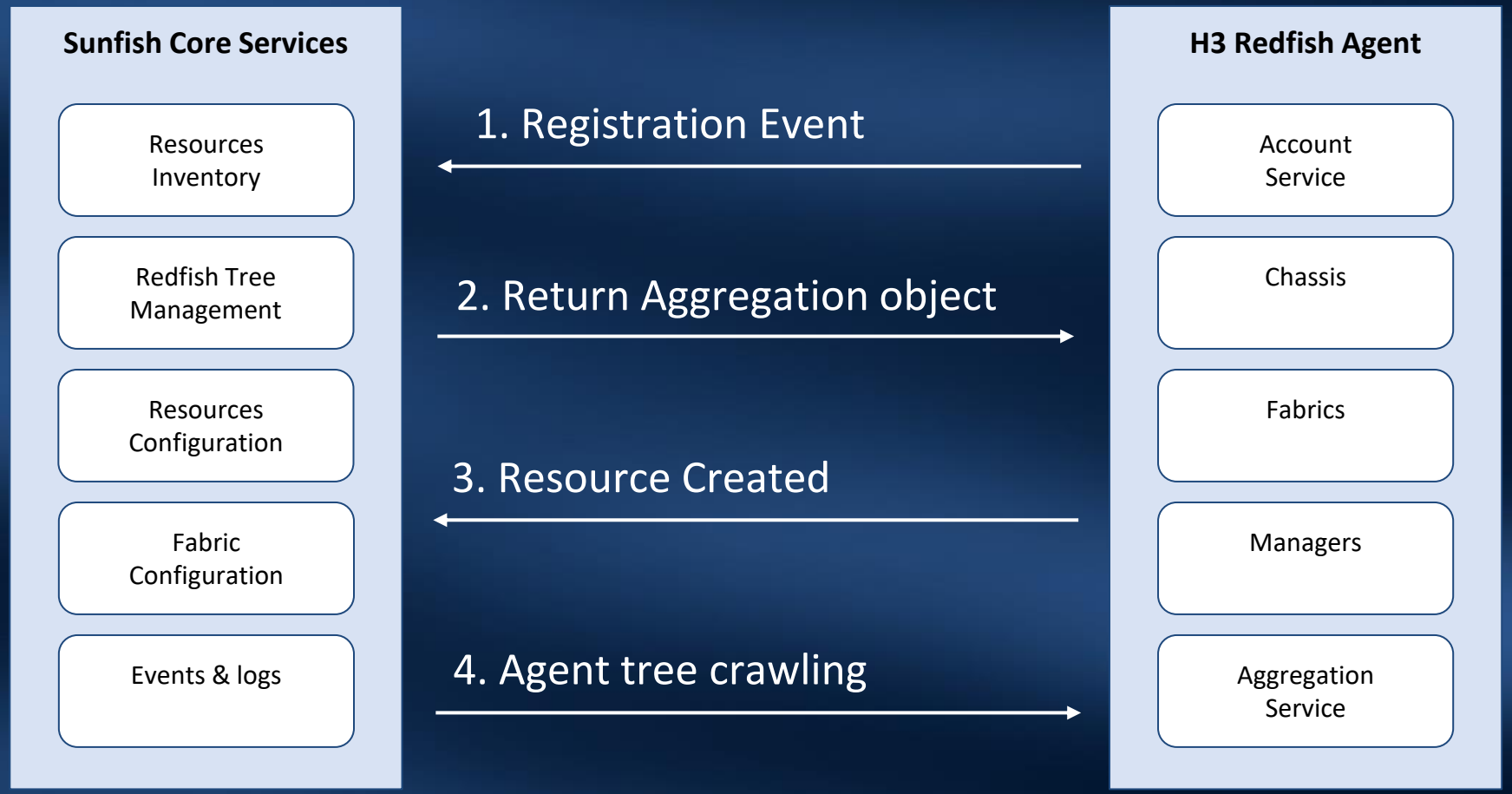
① Server find the added memory
② Server can read/ write these added CXL memory



3. Issue Commands to Bind & Unbind CXL Memory Appliance

① Bind the memory to server
② Update the resource allocation

Registration of CXL Memory Appliance to Sunfish



2025 OFA Webinar Series

THANK YOU

Brian Pan

H3 Platform

2025 OFA Webinar Series

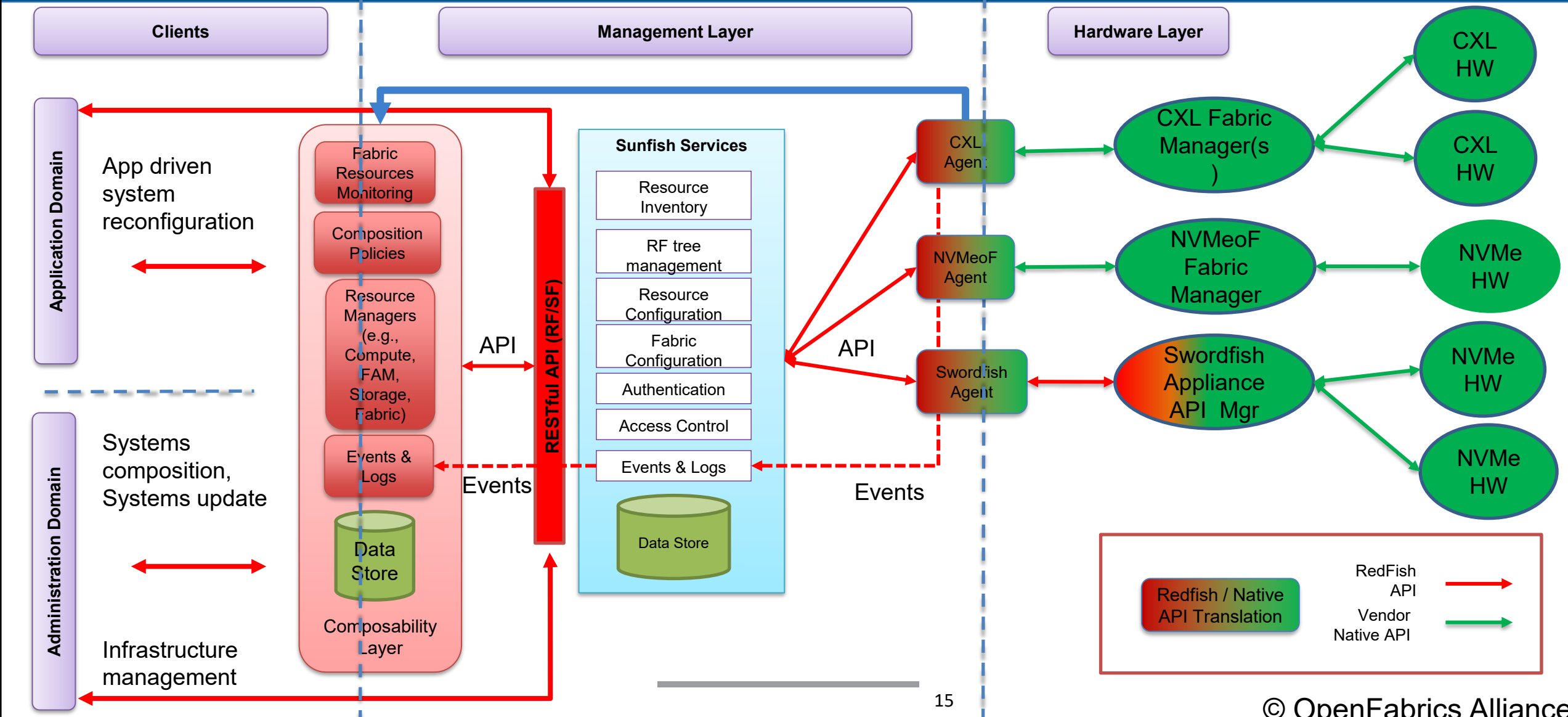
Sunfish for CXL Fabric Management

Russ Herrell

Independent OFA/Sunfish Contributor

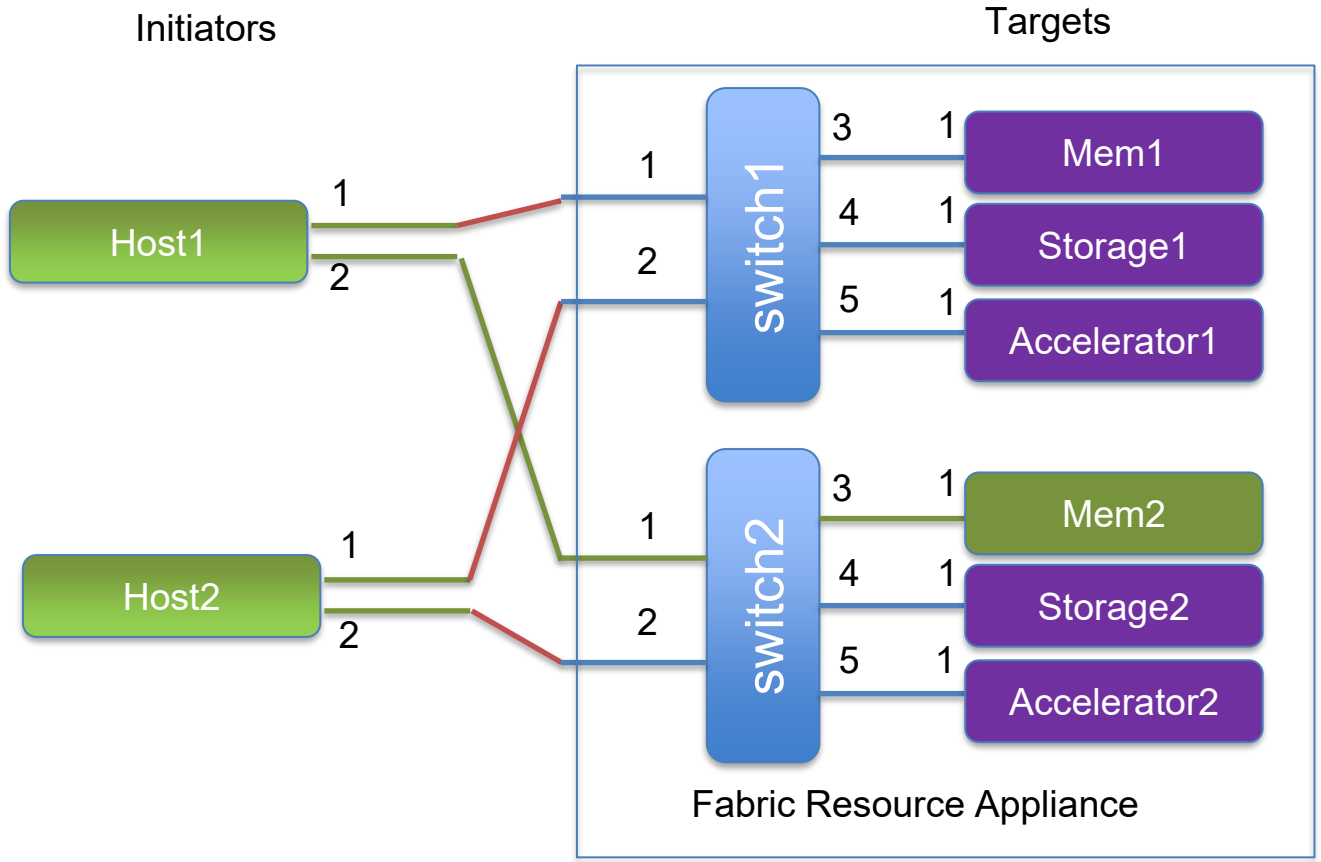
Sunfish Overview

The Sunfish Open Fabric Management Framework



Redfish Fabric Modeling Basics

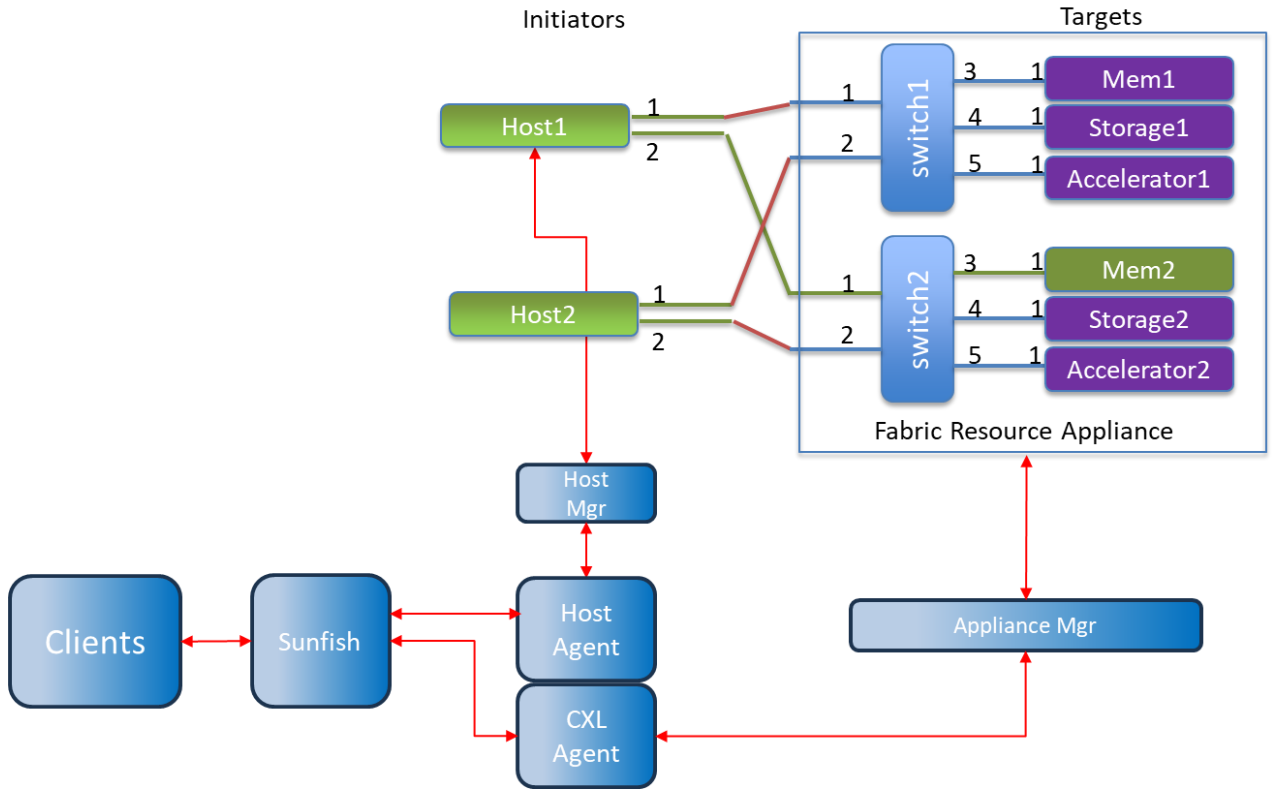
- **Redfish Fabric Concepts:**
  - Initiators issue fabric requests
  - Targets satisfy these requests
  - Switches route these requests and their responses between the Initiators and Targets
  - Physical Links connecting components are modeled as links between Ports on the components
  - The “Fabric” consists of Initiator and Target *Endpoints* connected physically to *Ports* on the *Switches* doing the routing
  - The physical topology is given by the physical connectivity of all Ports
  - Managing the Switches is essential to managing the “Fabric”
  - Redfish enables traffic between a specific Initiator *Endpoint* and a specific Target *Endpoint* by creating a *Zone* containing them both and declaring a *Connection* between them



Redfish focuses on modeling of hardware

Sunfish Fabric Modeling Basics

- **Sunfish Fabric Concepts:**
  - Agents or the hardware managers give each Redfish object they intend to manage a Redfish namespace URI and Id of the Agent's choosing.
  - All future conversations (requests, responses, events) between an Agent and the Sunfish Server must use the Agents' URIs
  - Sunfish Agents on the same fabric shall use the same Fabric object name and UUID when registering and uploading their Redfish objects to the Sunfish Server
  - Sunfish will 'share' a common Fabric object with multiple Agents
  - Sunfish assumes every Chassis and Systems object has one, and only one primary Agent that 'owns' the object
  - Sunfish Agents are fabric specific
  - Sunfish Agents know when a Port they manage is linked to a Port they do not manage

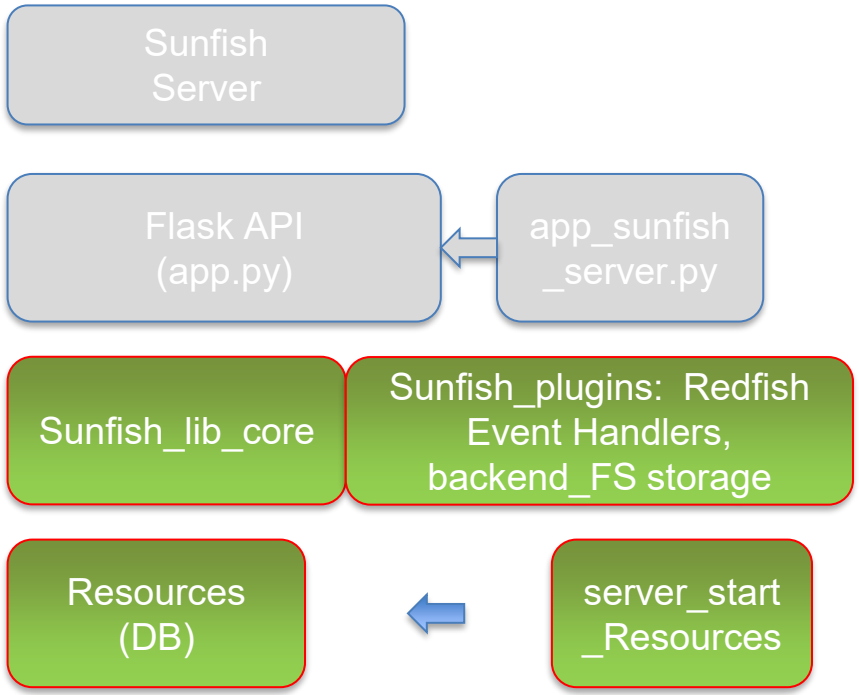


Sunfish focuses on management of hardware

Sunfish Reference Code Notes

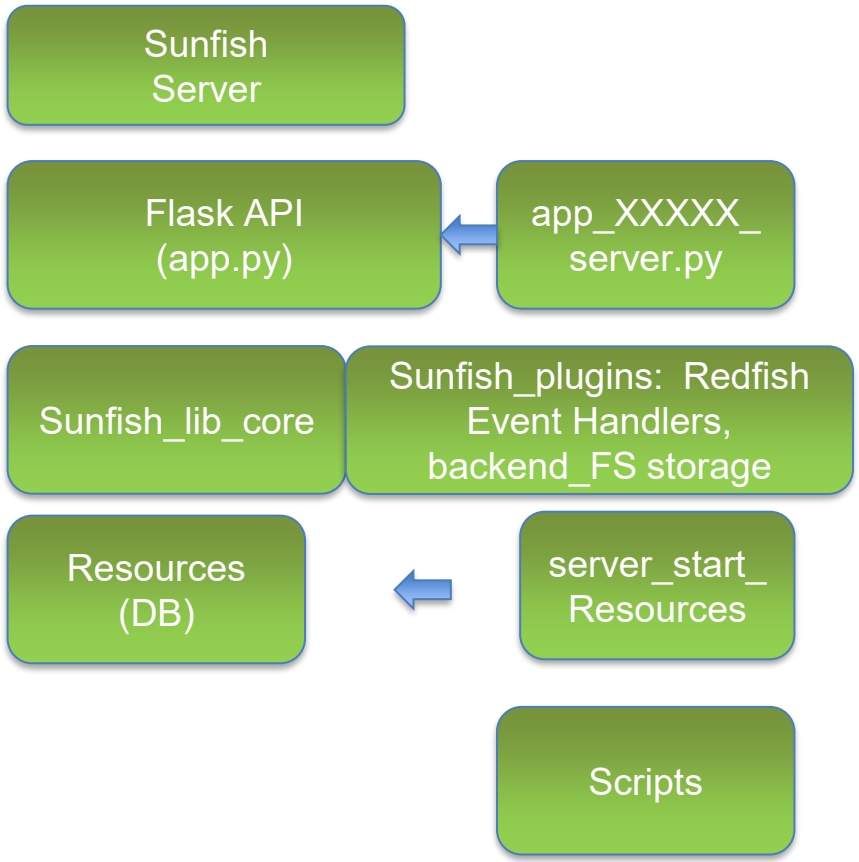
Sunfish Server, Agent, and Lib

- **sunfish_library_reference**
  - The Sunfish_library_reference contains several core functions designed to sort and execute requests (GET, POST, PATCH, DELETE) against various Redfish objects (Switches, Ports, Chassis, MemoryDomains, etc.), read and write the Sunfish resource storage, and check for the need to forward such requests to the owning Agent.
  - Reading and Writing the resource storage requires a minimum set of Resource objects, so there is a required Resources directory as part of the actual repo.
  - These same core functions are needed by most Agents to manage the Agent's local Redfish data base. Therefor, these core routines are built into a Class library and distributed via Wheel.
  - The sunfish_library_reference repo can be downloaded from github, built, and tested (somewhat) standalone.



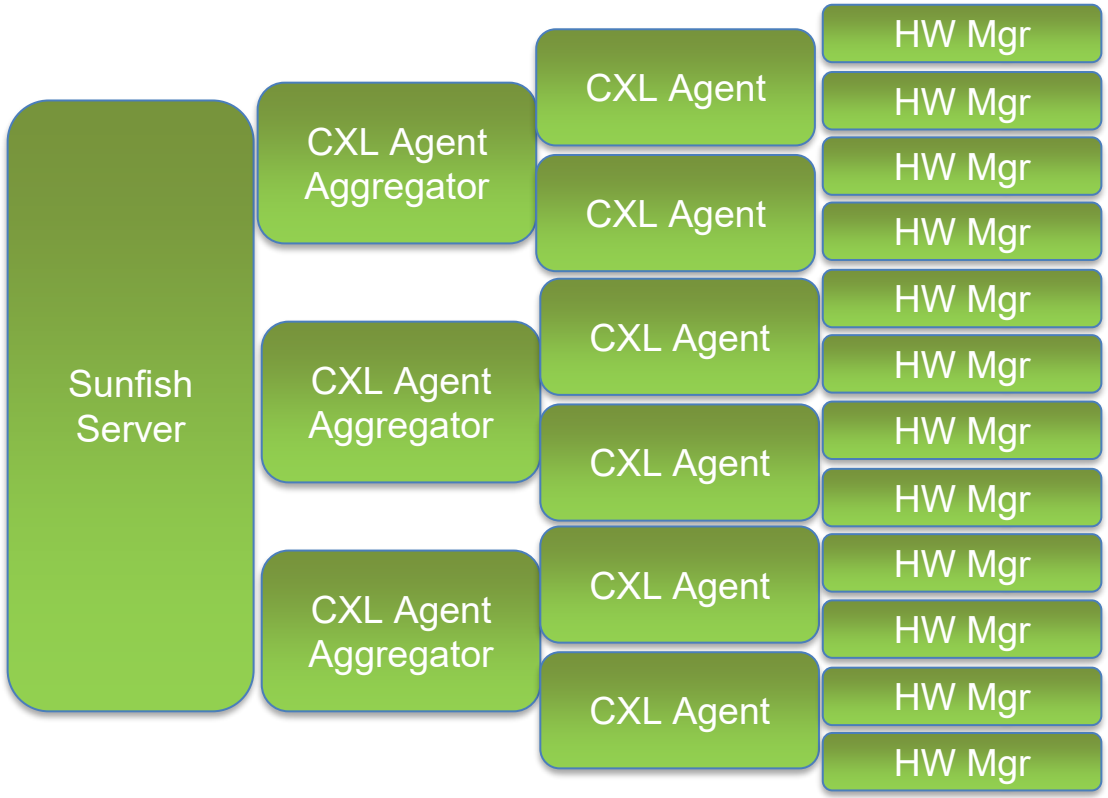
Sunfish Server, Agent, and Lib

- **sunfish_server_reference**
  - The Sunfish_server_reference contains these key pieces:
    - Several versions of an app.py file to be run via Flask
      - The main differences between them are in the conf{} configuration structure
    - A requirements.txt file to be used to create the proper virtual environment
    - A Resources directory which would be the default minimum Redfish Resources to allow the core-lib library to initialize
    - A Scripts directory which contains several useful scripts for cloning the correct repos, setting up useful mockup Resources, launching multiple instances of the different server flavors (Sunfish Service, Agent 1, Agent 2, NVMe_Agent)
    - A mockups directory which contains a few useful mockup fabric topology starting Resources that will replace the default minimum Resources
  - Copy the appropriate app_x_server.py file to the app.py file, then set up which mockups to use within the server to set the flavor of the server functionality that appears when this server instance is launched. Example mockups available:
    - A basic Sunfish Server
    - A single NVMeoF Agent
    - Two CXL Agents (Fabric Manager and Appliance Manager)



Sunfish Server, Agent, and Lib

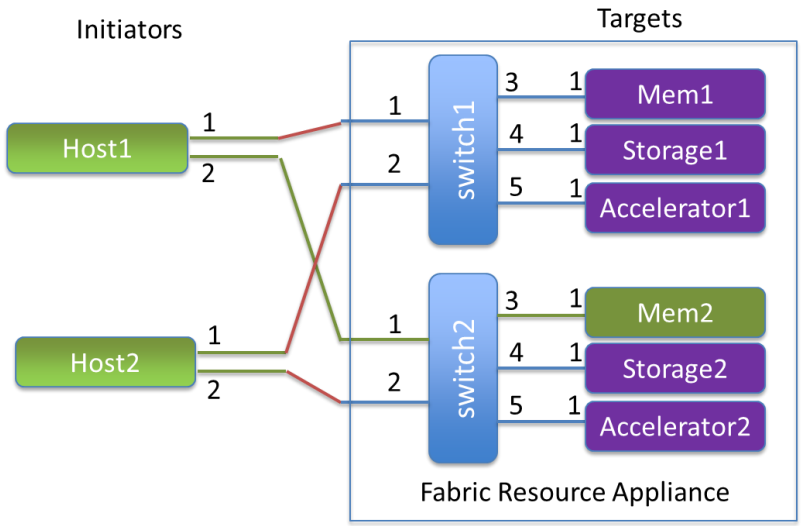
- **Sunfish hierarchical aggregation**
  - Since the sunfish_server_reference serves as both an Agent frontend as well as an aggregation function which uploads Redfish Resource trees, the potential is there to have multiple levels of Agents aggregating resources from very large-scale installations.
  - Large scale applications would be more appropriately implemented in some language other than Python

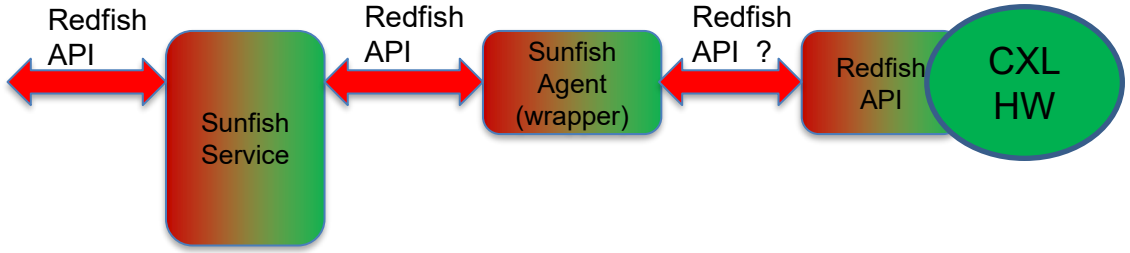


Sunfish Use Cases

Sunfish and a single Resource Appliance

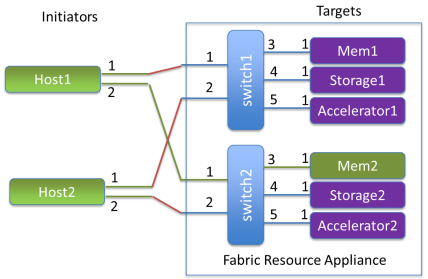
- **Sunfish Concepts:**
  - Fabric Resource Appliance has a local hardware manager
  - Local hardware manager may or may not have a Redfish API
  - Sunfish Agent could be quite simple if the hardware manager implements a Sunfish compliant Redfish API
  - Sunfish Service will upload and maintain accurate Redfish resource models for the Switches, Endpoint details and any component state made available for Endpoints within the appliance.
  - Target Endpoint components will likely be modeled to enough detail to support basic resource allocations to the Hosts.
  - NOTE: Hosts are probably only described as associated Endpoints and Switch Ports. Hosts are not likely to have associated Redfish Systems models, as the appliance hardware manager does not own the Hosts.
  - Details about which Host is attached to which Switch Port and thus is which Endpoint must be given by another entity, for example a Host manager.
- **If the hardware manager has a Redfish API, and only knows about the Targets, what is the Sunfish Value-add?**
  - Practically nothing in this degenerate case.
  - However, just add a couple more appliances….

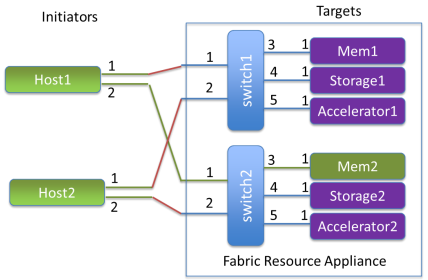


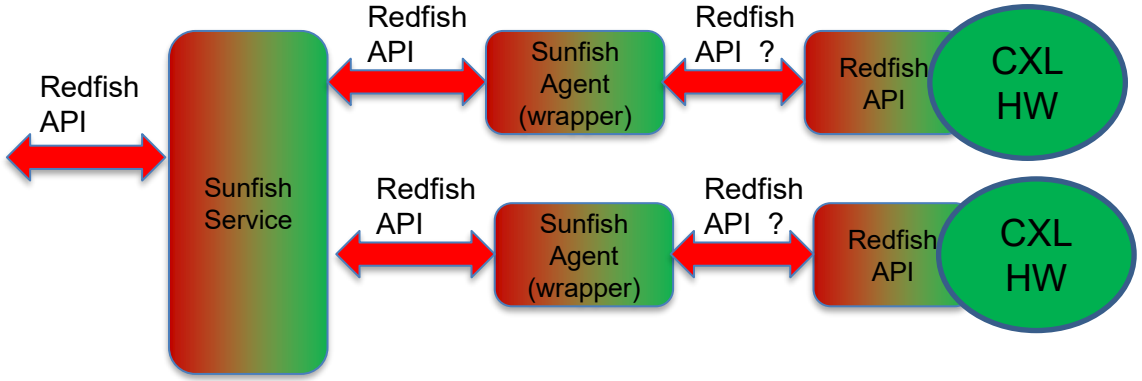


Sunfish and Multiple Resource Appliances

- **Multiple Appliances, topology 1:**
  - Two or more independent appliances – two or more independent Fabrics
  - With no connectivity between Hosts on Appliance A and Targets on Appliance B (and vice versa), the Sunfish Service will simply treat all resources of each hardware manager as isolated to their respective Fabrics.
  - Every isolated Fabric must have their own Redfish namespace!
  - It is possible that every appliance manager wishes to call its Fabric and its Chassis objects the same Redfish URIs and Ids, which means any entity trying to aggregate the resources of all the appliances into one Redfish service root (API namespace) needs to disambiguate the resulting name conflicts.
    - For example, every CXL fabric object would be identified as "/redfish/v1/Fabrics/CXL"
  - In such a topology, Sunfish Service provides a simple aggregation function that renames all namespace conflicts so that Sunfish clients only see one Redfish namespace, the Sunfish namespace.
  - Sunfish Service translates any renamed URIs back to the original URI given by the Agent that uploaded the object when forwarding client Redfish commands to the owning Agent.
  - Sunfish also renames (translates) all *navigation links* to a renamed object when they are encountered in other objects within the same Fabric namespace (same Agent upload).

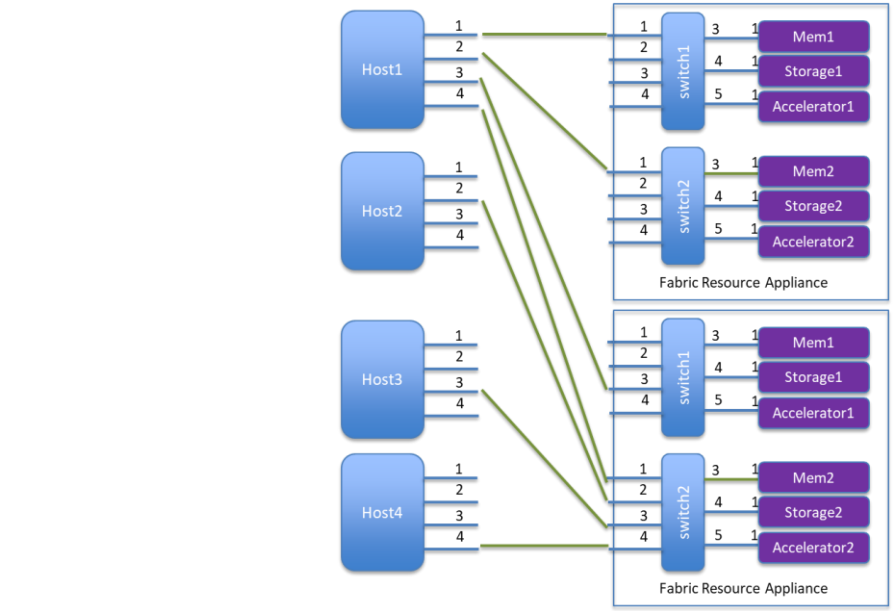


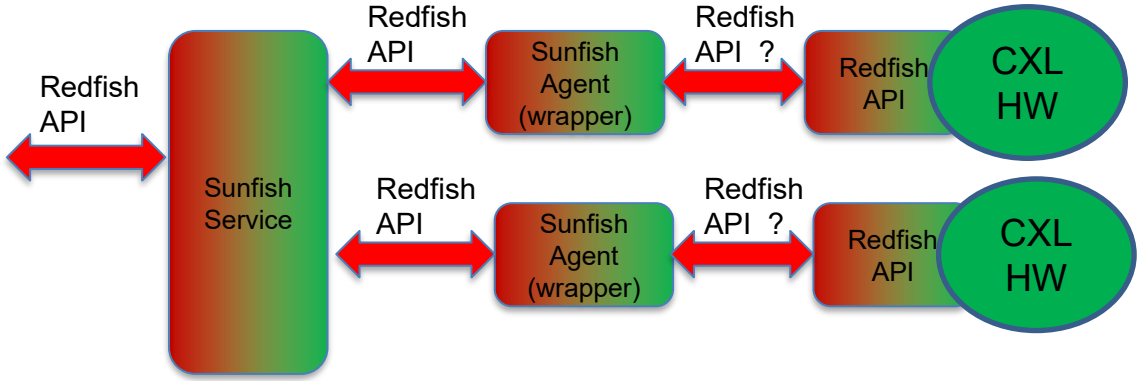




Sunfish and Multiple Resource Appliances

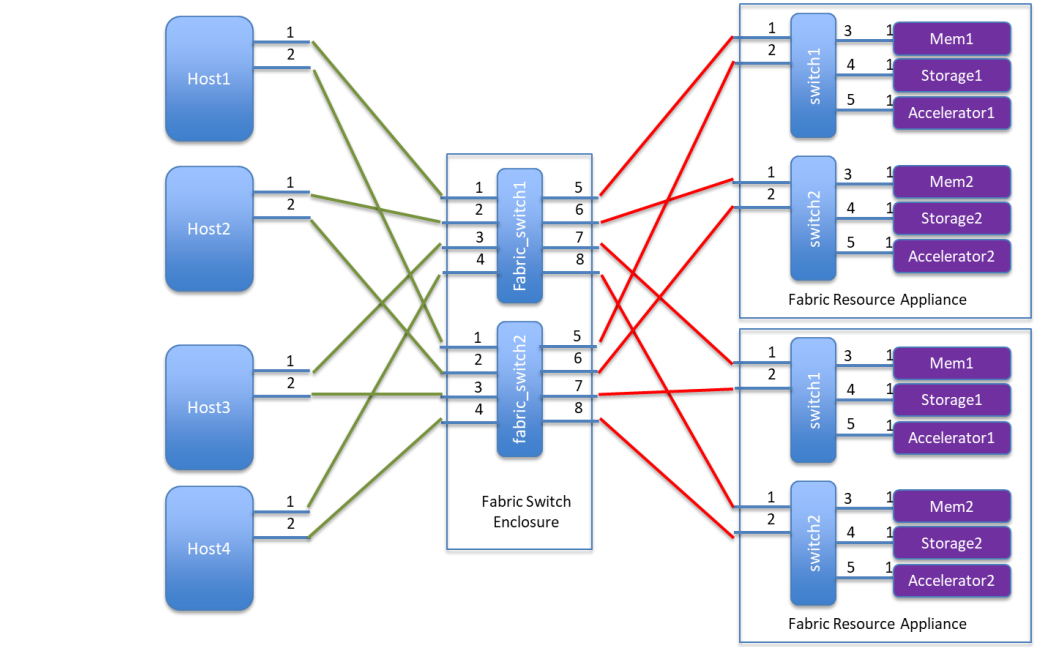
- **Multiple Appliances, topology 2:**
  - Two or more appliances with physical links to every Host
  - With potential connectivity between any Host and any Endpoint on any appliance Switch, this is a single fabric with a single fabric namespace
  - Every Endpoint needs to be a subordinate of the same Redfish Fabric model, even if different appliance managers use different fabric names. This is the opposite problem from independent fabrics.
  - In such a topology, Sunfish requires the fabric Admin to force all Sunfish Agents to upload the identical /redfish/v1/Fabrics/x object, in which the Redfish property "UUID" is identical across all Agents needing to be in the common Fabric and its namespace. If both the URI and the UUID of Fabric object match a pre-existing object in the Sunfish namespace, Sunfish will 'share' that Fabric object among all Agents attempting to upload another instance.
  - Other than a Fabric object, Sunfish Service still renames any object URIs that conflict among multiple Agent namespaces. For example, all Agents may assign Endpoint URIs and Ids sequentially starting at 1.
  - Sunfish still also renames (translates) all *navigation links* to a renamed object when they are encountered in other objects within the same Fabric namespace (same Agent upload).

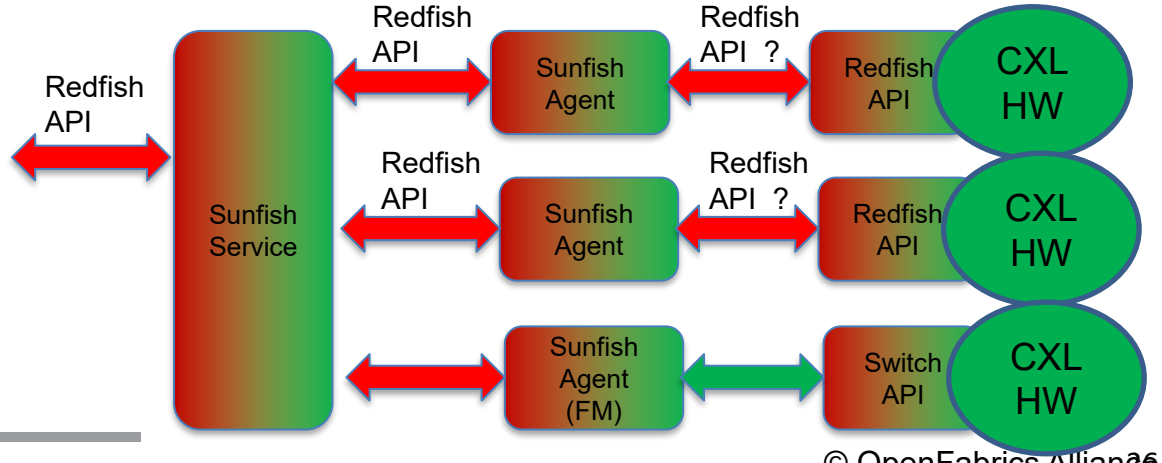




Sunfish and Multiple Switch Layers

- **Multiple Switch Layers:**
  - All to All switched topology
  - With potential connectivity between any Host and any Endpoint on any appliance Switch, this is a single fabric with a single fabric namespace
  - The same problems of creating a single Redfish namespace for all entities on the Fabric as discussed on previous slides exist in multi-layer switch topologies.
  - However, the new Switch to Switch links (in red, sometimes called ‘fabric links’ vs ‘endpoint links’) introduce several new problems.
  - Example: Redfish Connections and Zones are used to define which Endpoints may exchange traffic with which other Endpoints, but they do not contain any methods to denote which Switch to Switch links may be permitted or prohibited in the path. The Sunfish team is addressing this limitation with the Redfish team.
  - Example: Switch to Switch links may associate two Ports being managed by different Agents / hardware managers. Sunfish calls these Switch to Switch links that cross manager boundaries, “Boundary Links”. **Sunfish library event handler code addresses this issue.**

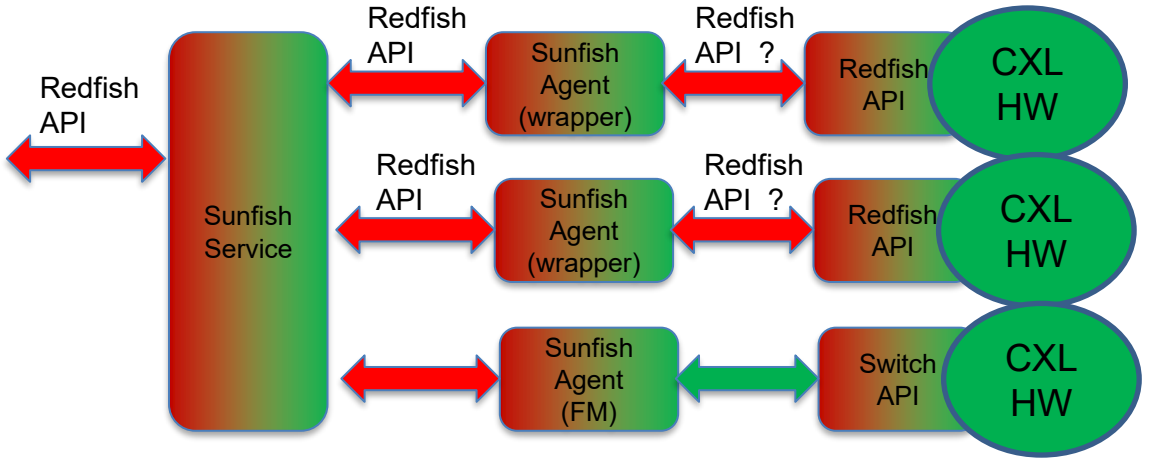




Summary

Summary

- **What is available from Sunfish Repo:**
  - Sunfish_library_reference has core routines to manipulate Redfish fabric objects and manage a Redfish database
  - Sunfish_server_reference has routines and scripts that enable users to develop API servers, configured as one Sunfish Service fed by some number of Agents using mocked up Resources.
- **Why use Sunfish?**
  - Scaling
    - When dealing with numerous hardware managers, Sunfish architecture keeps the workload per Agent to a manageable level.
    - Sunfish handles multi-Agent namespace collisions so clients are not exposed to the bookkeeping gymnastics that are required in the administration of installations of even modest size.
  - Portability across vendors, across multiple fabric configurations and across hardware revisions
    - Redfish-based configuration and control scripts can isolate testing and operating scripts from hardware changes.



Sunfish documentation: https://openfabrics.github.io/sunfish_docs/Sunfish%20Doc.html.
Sunfish GitHub repository: https://github.com/OpenFabrics/sunfish_library_reference
https://github.com/OpenFabrics/sunfish_library_reference

2025 OFA Webinar Series

THANK YOU

Russ Herrell

Individual Contributor

OFA Sunfish Project

Q & A

2025 OFA Webinar Series

THANK YOU