2021 OFA Virtual Workshop

Gen-Z Linux Subsystem Update

Jim Hull, Sr. Software Engineer

IntelliProp, Inc.

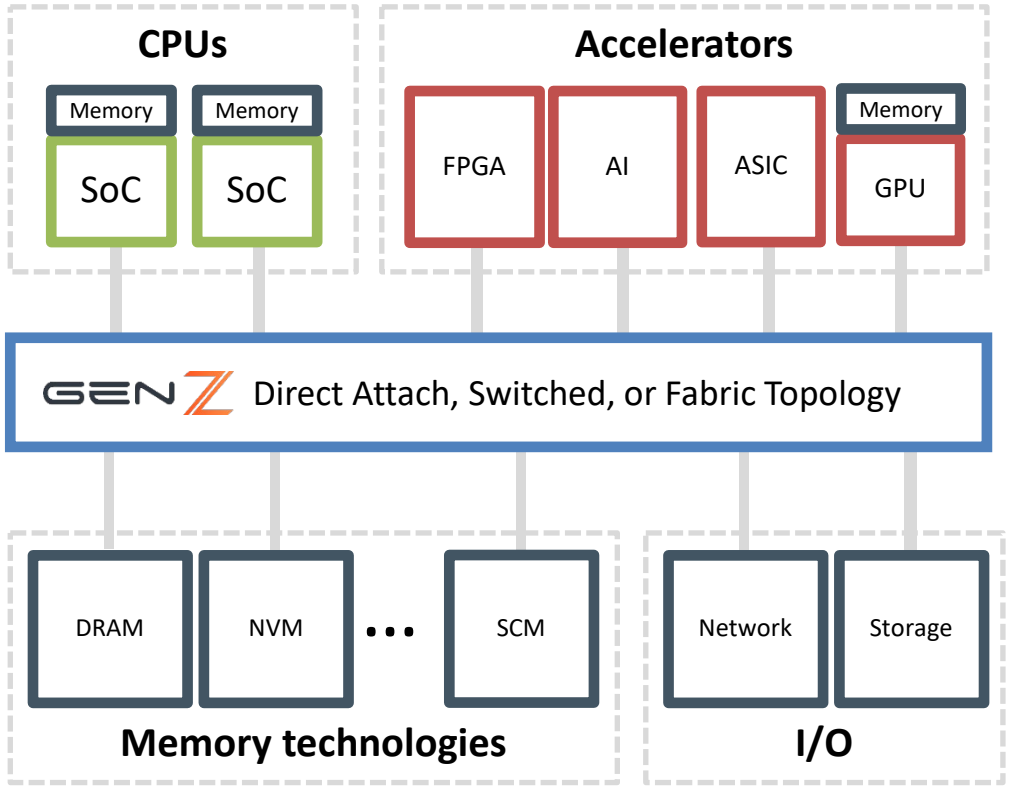
AGENDA

- Gen-Z Introduction
- Gen-Z Linux Subsystem
- Live Demo

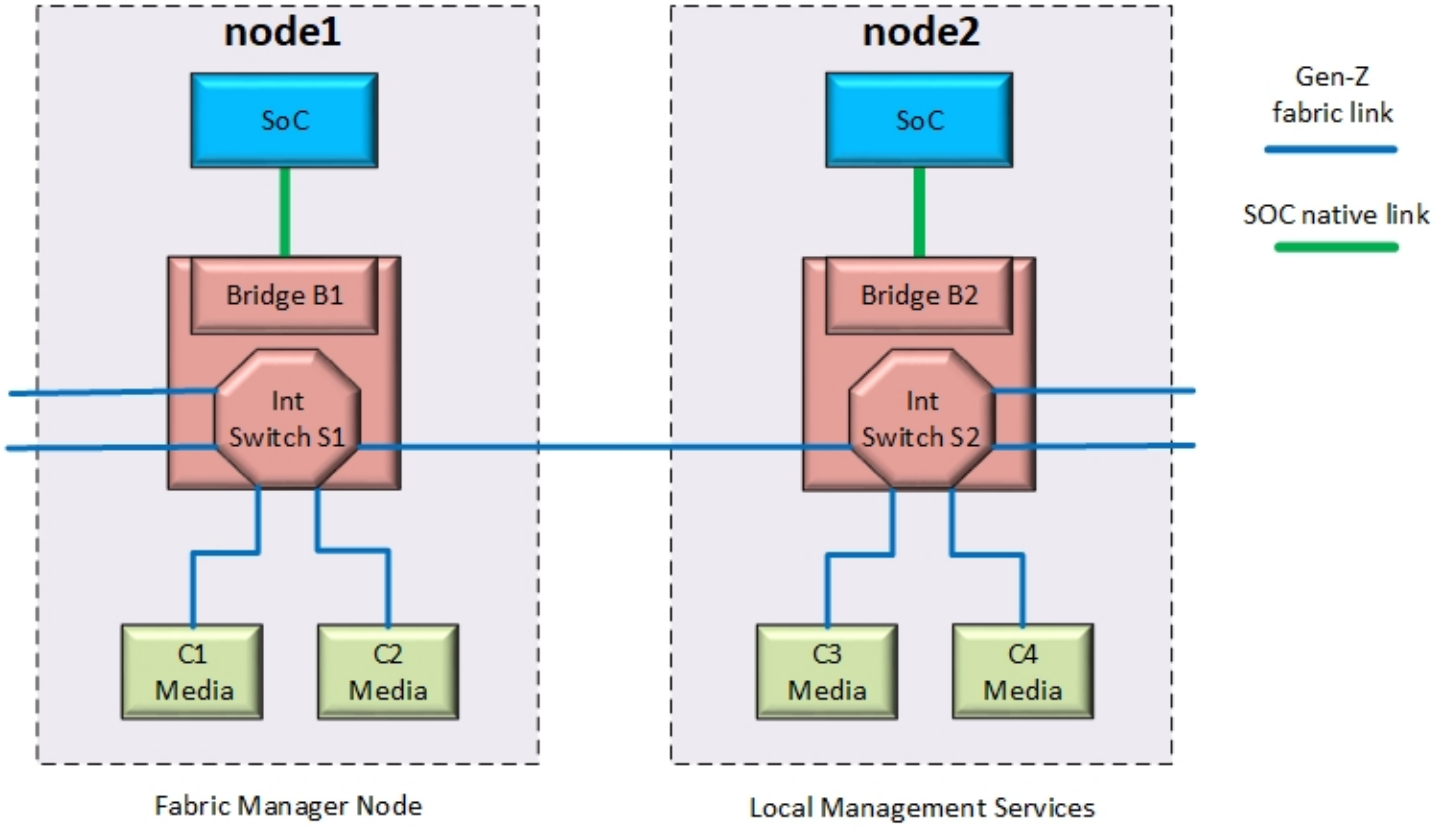
Gen-Z Introduction

GEN-Z, A NEW OPEN INTERCONNECT PROTOCOL

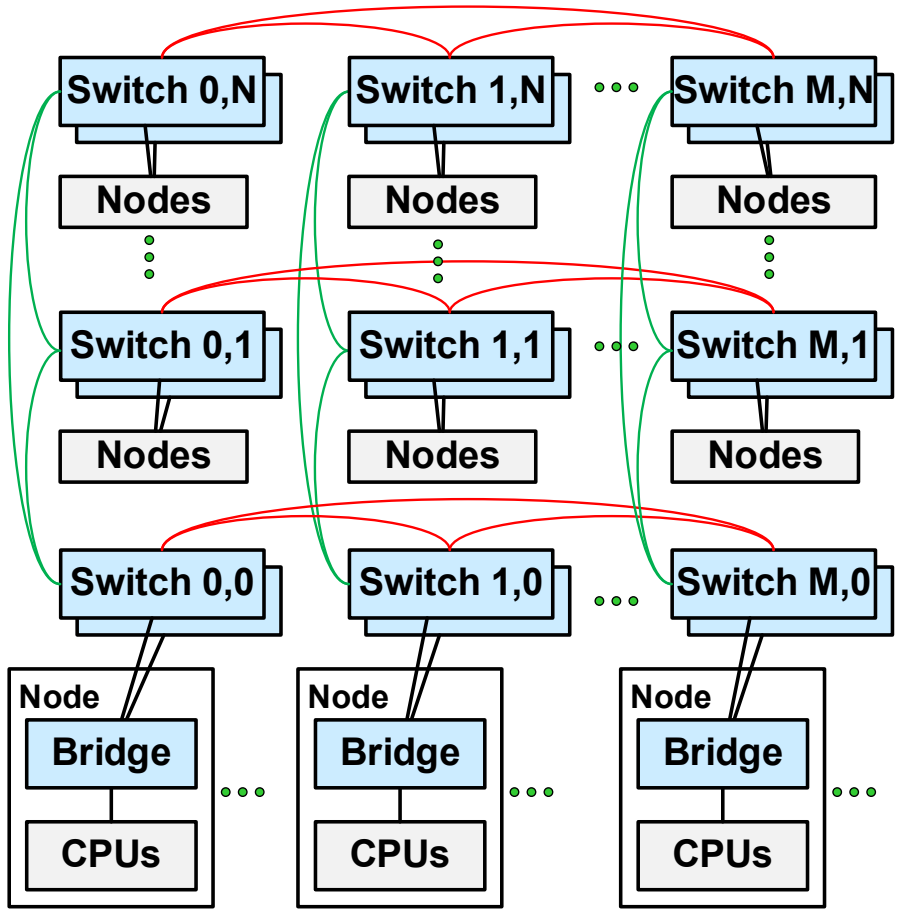
- Open consortium with broad industry support (70+ members)
- Family of Specifications: Core, Physical Layer, Mechanical, Scalable Connectors, Management
- Gen-Z is a memory semantic fabric that scales from 2 to 256M components
- PHY-independent protocol
  - Specific PHY determines latency/bandwidth/reach
  - 32 GT/s PCIe PHY, 25 Gbit and 50 Gbit 802.3 PHYs
- Can support an unmodified OS (e.g. firmware with ACPI support and Logical PCI Devices (LPDs))
- Better to modify OS, e.g., Linux, for full Gen-Z support



EXAMPLE GEN-Z FABRICS



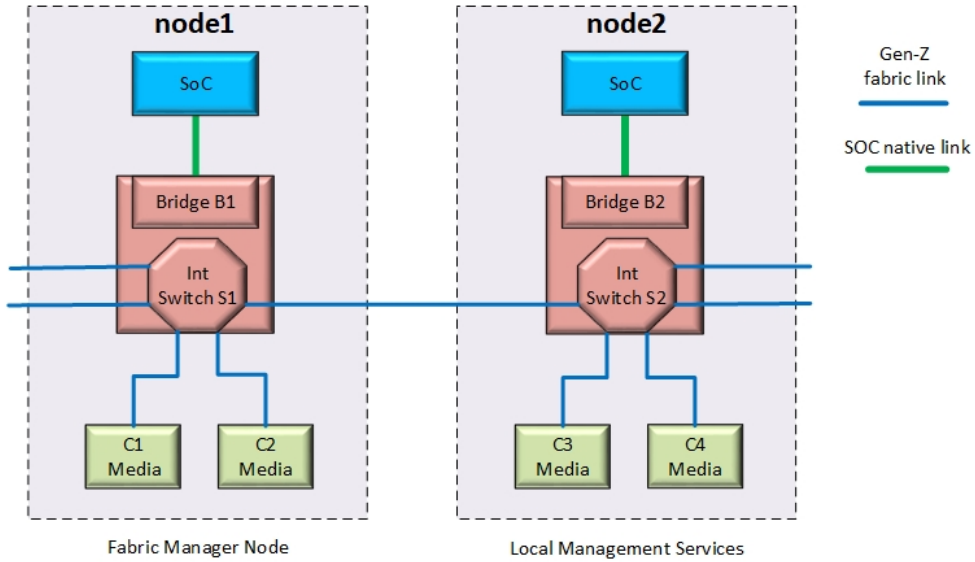
Simple 6 Component Topology



2D HyperX System Topology

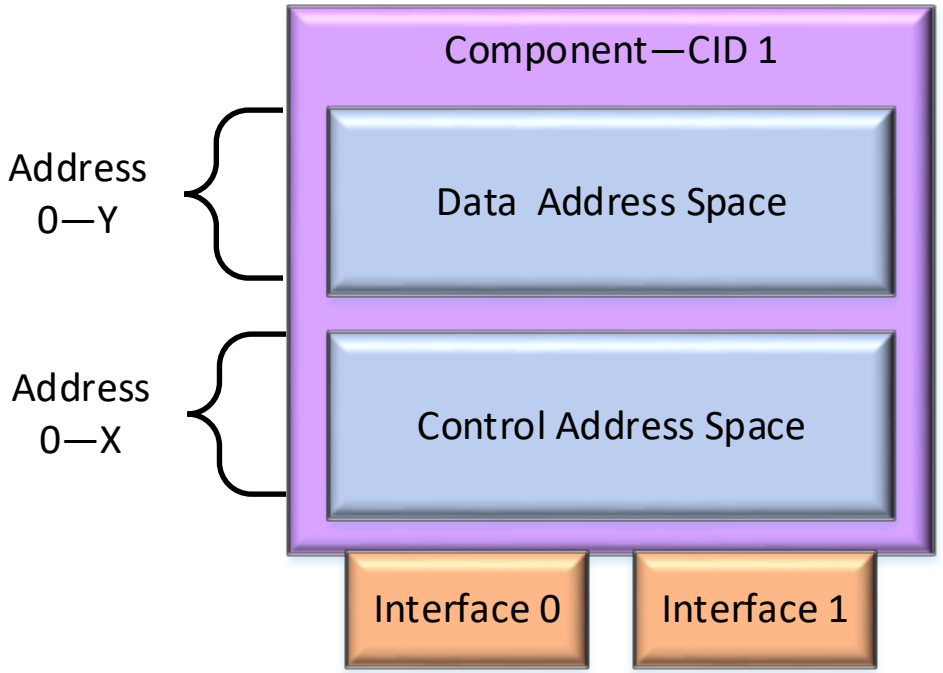
GEN-Z MANAGEMENT SOFTWARE

- Gen-Z fabric spans multiple OS instances
  - No OS instance can assume it “owns” all components on fabric
- Components can be subdivided into resources
  - Example: a big media component split up
- A fabric manager assigns components/resources to each OS according to a “grand plan”
  - Describes components/resources using a DMTF Redfish specification
  - In-band vs out-of-band
  - Programs routing tables, access controls, etc.
- Local Management Services run on each OS instance
  - Consumes Redfish description for its OS instance



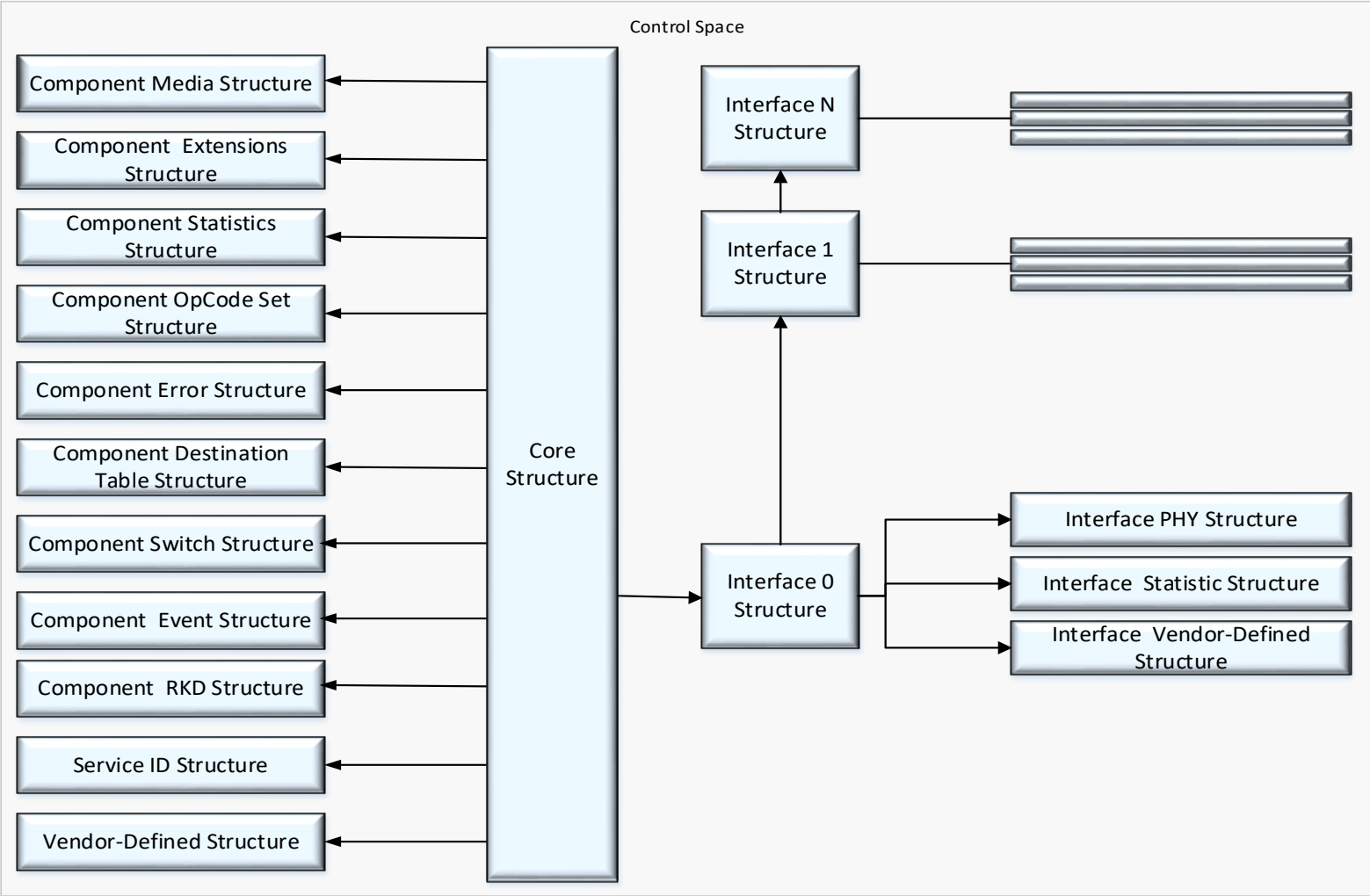
BASIC GEN-Z CONCEPTS

- Basic component roles
  - Requester: initiates packet
  - Responder: responds to request packet and sends acknowledgement (if specified)
  - Switch: routes packets from ingress interface to one or more egress interfaces
- Components have a 28-bit global component ID (GCID) assigned by management software
  - Optional 16-bit subnet ID (SID) plus 12-bit component ID (CID)
- Components have separate control and data space
  - Up to 2^52 bytes of control space for management
  - Up to 2^64 bytes of data space for component specific functionality
- Packets are unordered by default (big difference from PCIe)
- Software-managed coherence

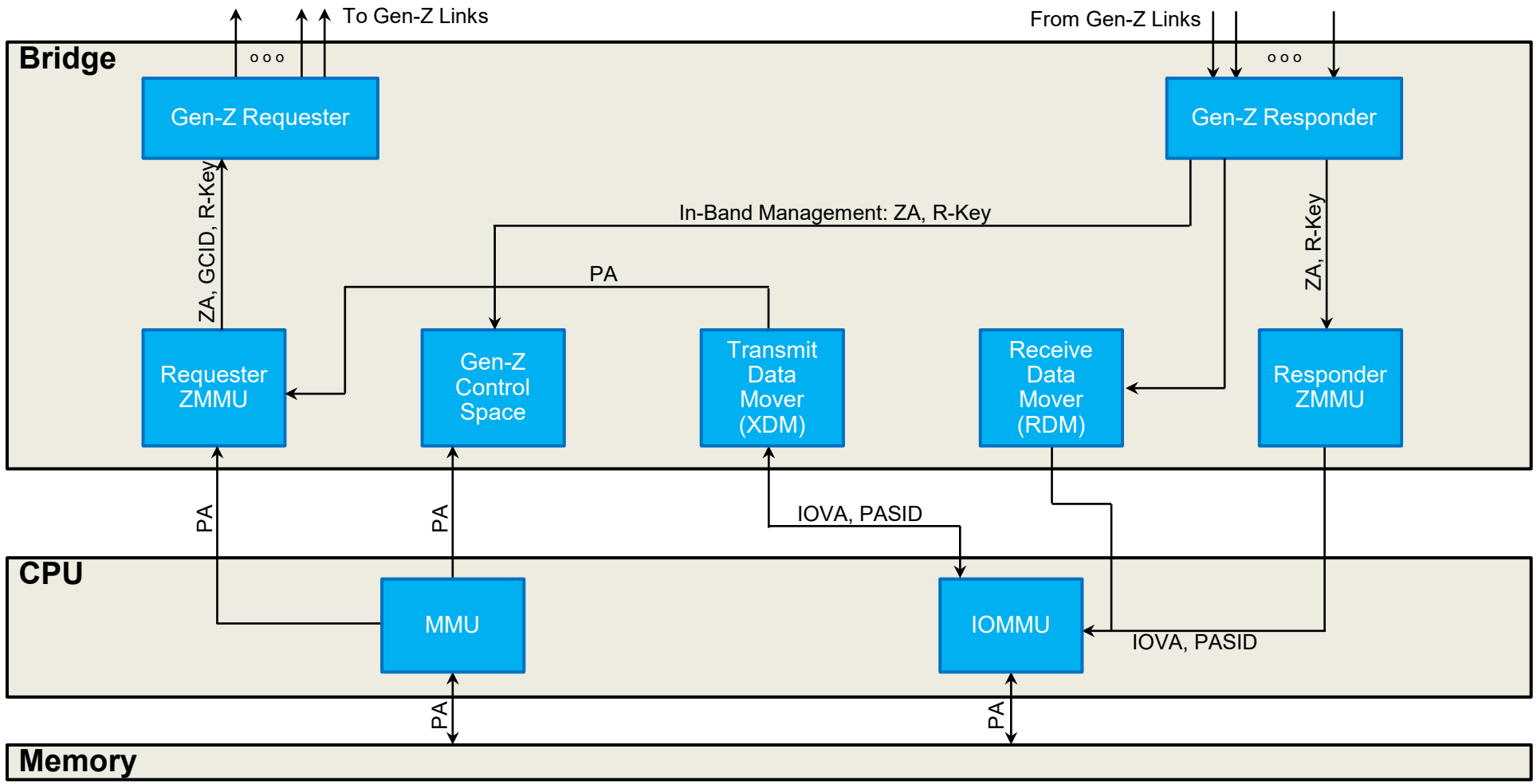


CONTROL SPACE STRUCTURES

- Conceptually similar to PCIe config space, but more elaborate
- Core Structure always at Control Space address 0
- Follow pointers to find other Structures and Tables
- These structures exist independent of in-band vs. out-of-band management



BRIDGE COMPONENT BLOCK DIAGRAM



Gen-Z Linux Subsystem

WHY A GEN-Z SUBSYSTEM?

- Enable native device drivers, exposing the full capabilities of Gen-Z
  - Enables access to Gen-Z advanced features
  - Sharing of fabric resources across Linux instances
  - Common support code not duplicated in every bridge driver
- Enable user space fabric managers and local management services
  - Both in-band and out-of-band fabric managers

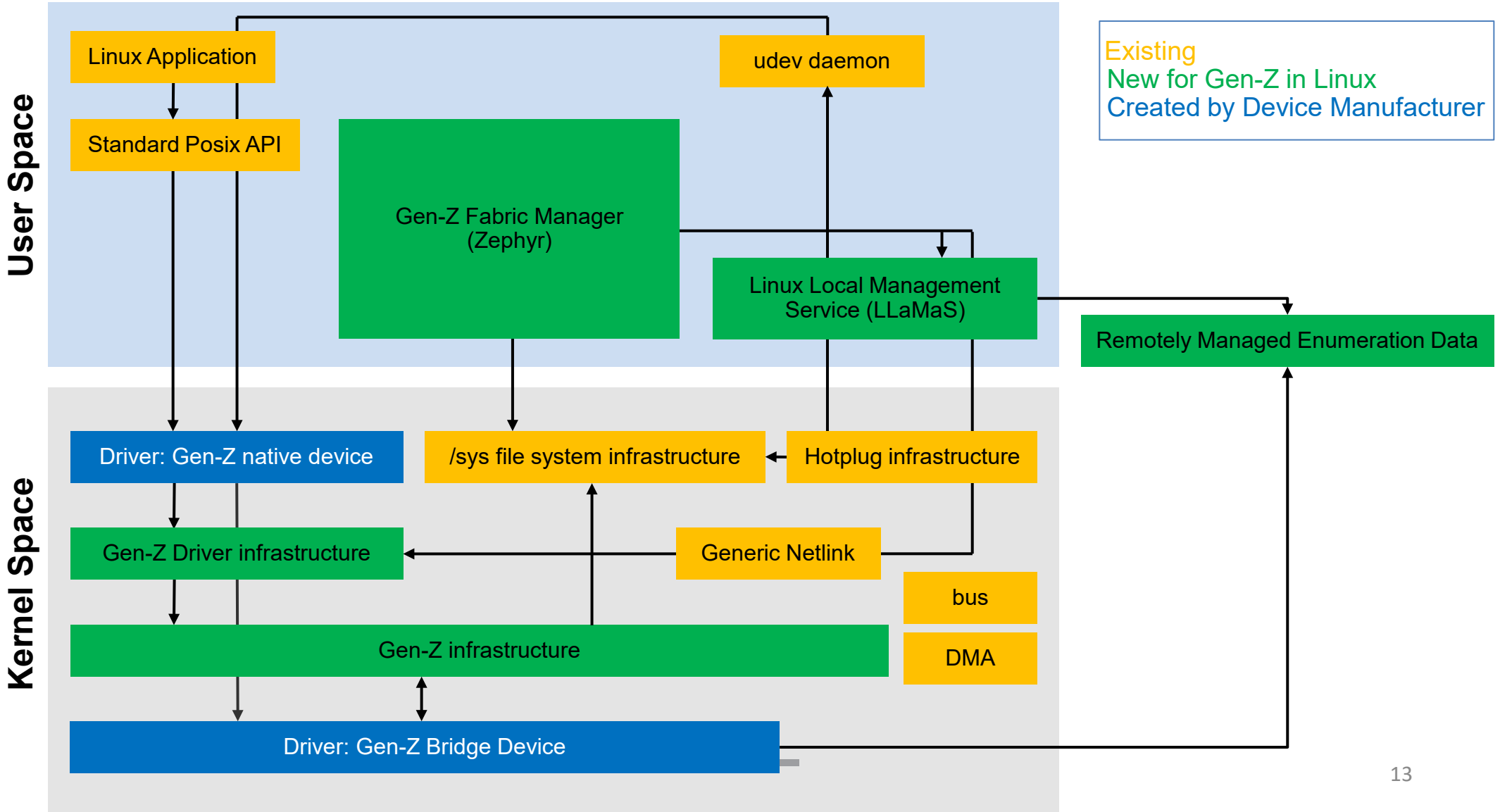
Gen-Z Advanced Features

- Interrupts
- Atomics
- R-Key Update Packets
- Buffer Requests
- Pattern Requests
- Multi-Op Requests
- Precision Time
- Lightweight Notification
- Wake Thread
- Packet Encapsulation
- Transparent Routers
- Strong Ordering Domains

DESIGN CONSIDERATIONS

- Be like PCI, USB and other existing buses when we can
- Policy in user space and mechanism in the kernel
- Use existing kernel services
- Deal with “almost everything is optional in Gen-Z”

GEN-Z SUBSYSTEM BLOCK DIAGRAM



GEN-Z DRIVERS

- **Bridge driver**
  - Is discovered by host bus “native” discovery method (e.g., PCIe, ACPI)
  - Registers with the Gen-Z subsystem
  - Implements a set of APIs to allow subsystem to read/write/mmap control & data space
- **Gen-Z device driver**
  - Modeled after PCI’s interfaces except driver matching is by UUID rather than vendor/device ID
- **Special “generic” Gen-Z device drivers**
  - genz-blk
    - Makes a region of Gen-Z data space visible to host as a DAX-enabled block device
  - genz-mem
    - Makes a region of Gen-Z data space visible to host as system memory
  - genz-enic
    - Creates an emulated Ethernet network on the Gen-Z fabric

DATA MOVERS

- Kernel drivers like a block or eNIC driver would benefit from a generic data mover API
  - Data mover queues can be assigned to other Gen-Z drivers
  - Drivers can use a data mover to generate special Gen-Z packet types like atomics, write message, buffer and pattern requests
  - Allows bridge HW vendors to innovate

- RDMA drivers want to expose the native data mover hardware directly to user space
  - A generic Gen-Z subsystem kernel data mover API is irrelevant
  - Userspace libraries like libfabric can hide HW differences from apps
  - Is a libfabric provider per HW vendor reasonable, or does there need to be some HW standardization?

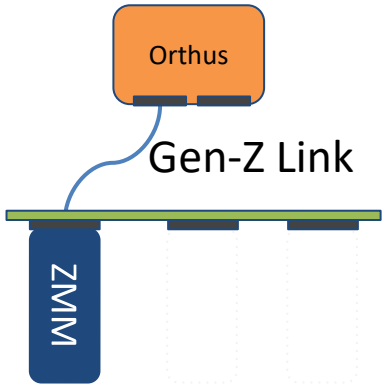
INTERRUPTS AND UNSOLICITED EVENT PACKETS

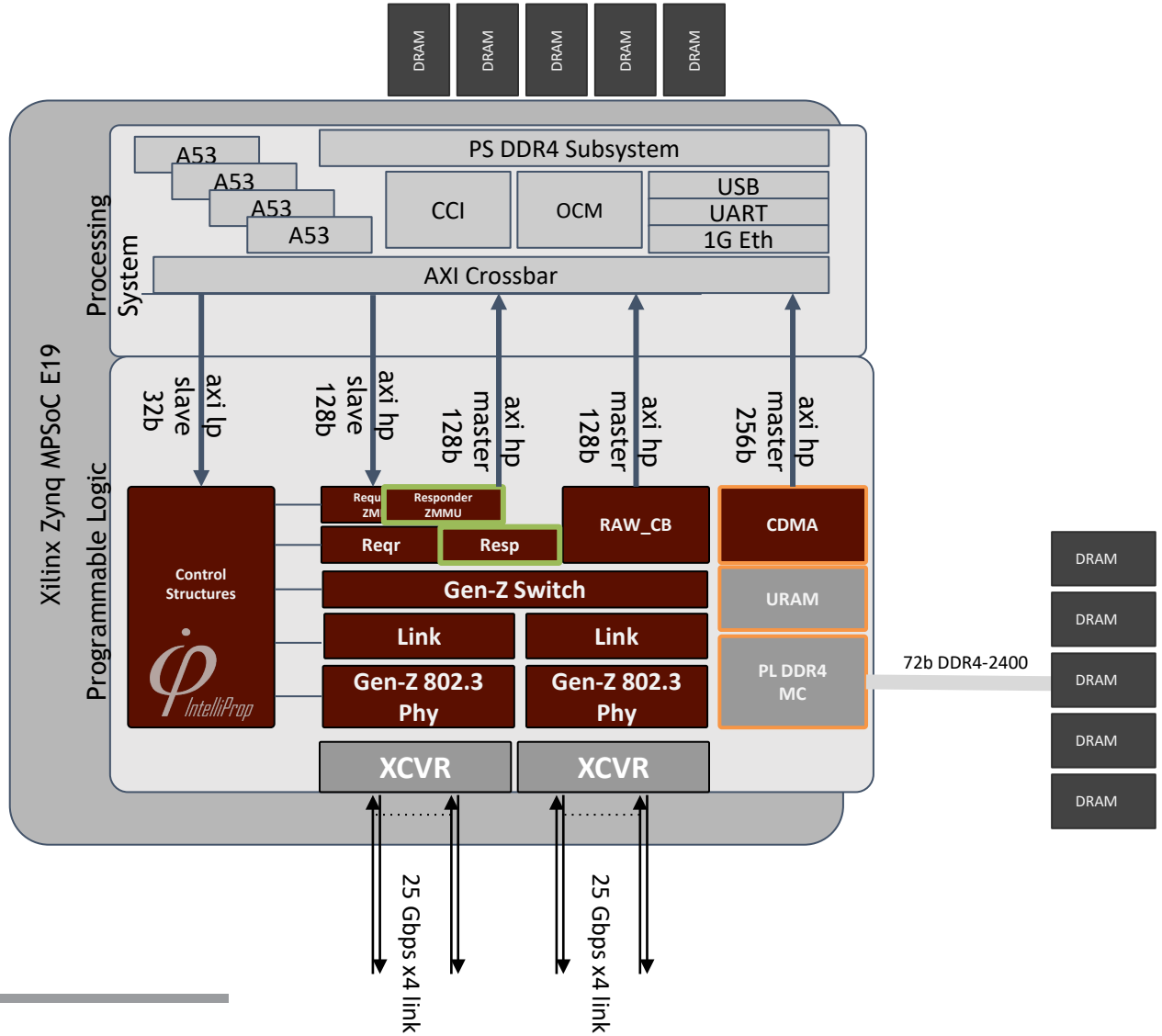
- **Not like PCI's architected MSI/MSI-X interrupts**
- **Gen-Z fabric interrupt sources:**
  - Gen-Z interrupt packets from components
  - UEPs
- **Unsolicited Event Packets (UEP) signal fabric state changes like**
  - Link-up/down
  - Hot add/remove of component
  - Errors
- **UEPs become interrupts from the targeted bridge component**
  - Bridge driver forwards UEP to subsystem
  - Subsystem forwards to zephyr or llamas in userspace

Live Demo

DEMO HW TOPOLOGY

- **Two components directly connected**
  - Gen-Z Bridge (codename Orthus)
  - Gen-Z Memory Module (ZMM)





REFERENCES

- Gen-Z Consortium for specifications: genzconsortium.org
- Gen-Z Linux Subsystem: github.com/linux-genz/linux
- LLaMaS github: github.com/linux-genz/llamas
- Alpaka github: github.com/linux-genz/python3-alpaka

2021 OFA Virtual Workshop

THANK YOU

Jim Hull, Sr. Software Engineer

IntelliProp, Inc.