

## Links

- <https://gitlab.com/bwidawsk/qemu>
  - cxl-2.0v<latest> branch
- <https://gitlab.com/bwidawsk/linux>
  - cxl-2.0v<latest> branch
- <https://github.com/pmem/ndctl>
  - cxl-2.0v<latest> branch
- <https://www.computeexpresslink.org/download-the-specification>

FOSDEM21 – Emulator devroom

# Compute Express Link in QEMU

Ben Widawsky – Software Engineer



# Introduction

## Problem Statement

It's difficult.



- CXL 2.0 has significant changes.
- CXL 1.1 support isn't even there yet, so there really isn't anything to start with.
- Starting before the spec is even complete.

How to get a Linux device driver upstream, by the time internal and external partners receive silicon?



## End Goal

### T minus 6 months

- Linux CXL 2.0 memory device drivers within 0 days of spec release
  - Validate the spec
  - Enable hardware vendors and validation
- Have some **reusable** infra for regression testing
- Potential use for guests
- Masters of our own destiny.

- In 6 months, have a well tested Linux driver sent on the day the spec is released.
- Whatever environment is used should have some future use.
- Don't be encumbered by the plans/schedules of others.

## Pre-silicon state of the art

- Hardware
  - No 2.0 FPGAs available
  - No 1.1 hardware available
- Prior art
  - NVDIMM faked it all in Linux
  - QEMU CCIX patches



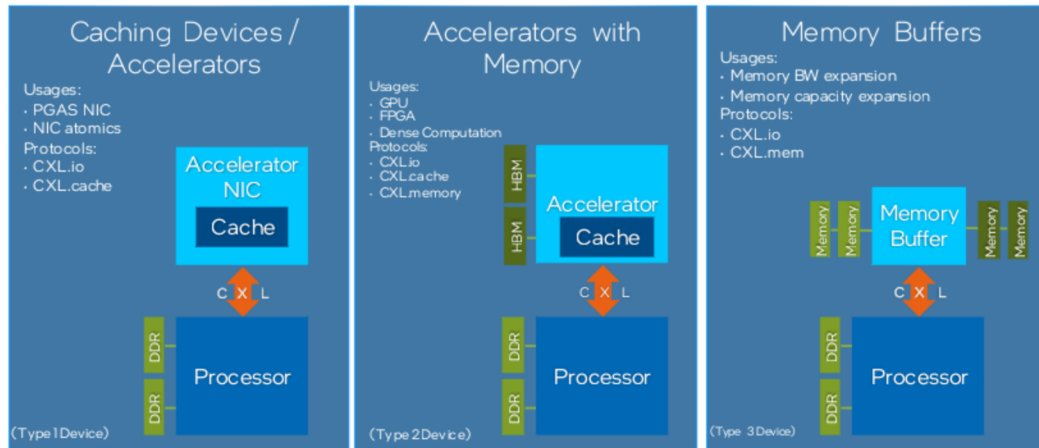
There was no prior art of much value.

# CXL 2.0

## CXL 2.0 High Level

- Open industry standard for high bandwidth, low-latency interconnect
- Connectivity between host processor and accelerators/ memory device/ smart NIC
- Addresses high-performance computational workloads across AI, ML, HPC, and Comms segments
  - Heterogeneous processing: scalar, vector, matrix, spatial architectures spanning CPU, GPU, FPGA
  - Memory device connectivity
  - PCIe PHY completely leveraged with additional latency optimization
  - Dynamic multiplexing of 3 protocols
- Based on PCIe® 5.0 PHY infrastructure
  - Leverages channel, retimers, PHY, Logical, Protocols
  - CXL.io – I/O semantics, like PCIe - mandatory
  - CXL.cache – Caching Semantics – optional
  - CXL.mem – Memory semantics - optional

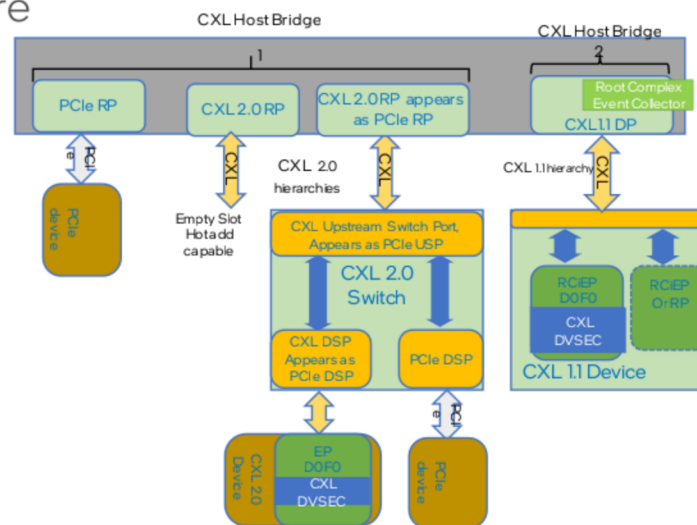
# CXL 2.0 Usages

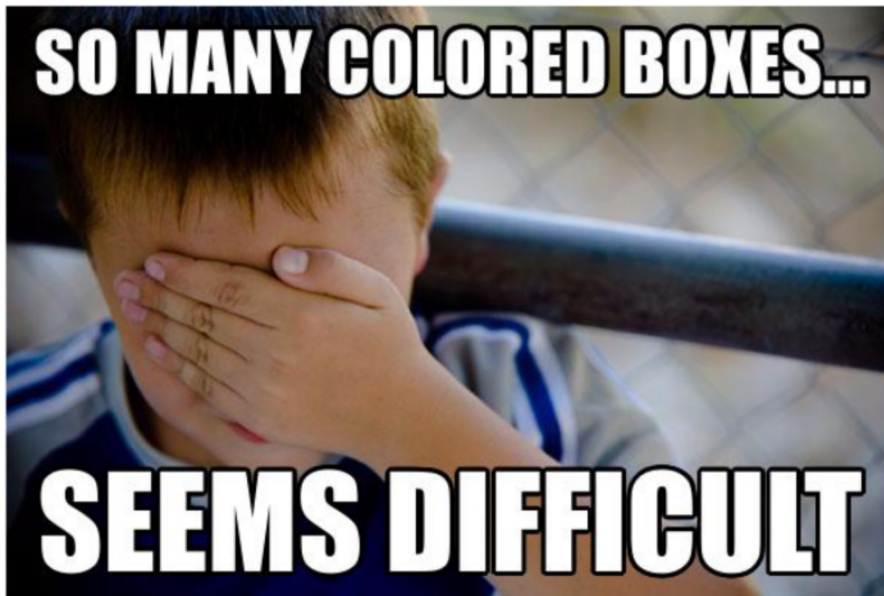


## CXL 2.0 Architecture

### ▪ CXL 2.0 hierarchy appears like PCIe hierarchy

- Legacy PCI SW and CXL SW sees a RP or DSP with Endpoints below
- CXL link/interface errors are signaled to RP, not RCEC
- Port Control Override registers prevent legacy PCIe software from unintentionally resetting the device and the link





# QEMU

## Implementing CXL

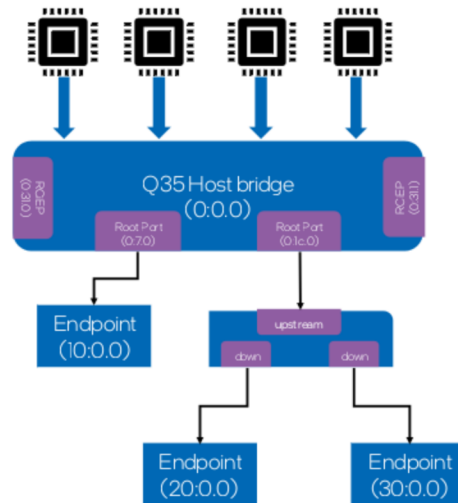
QEMU is a good for CXL enabling because it's open source and very featureful.  
CXL is an industry wide standard. A lot of different parties want lots of different things with different priorities.  
Making a useful platform for everyone will increase adoption and make it more useful to everyone.



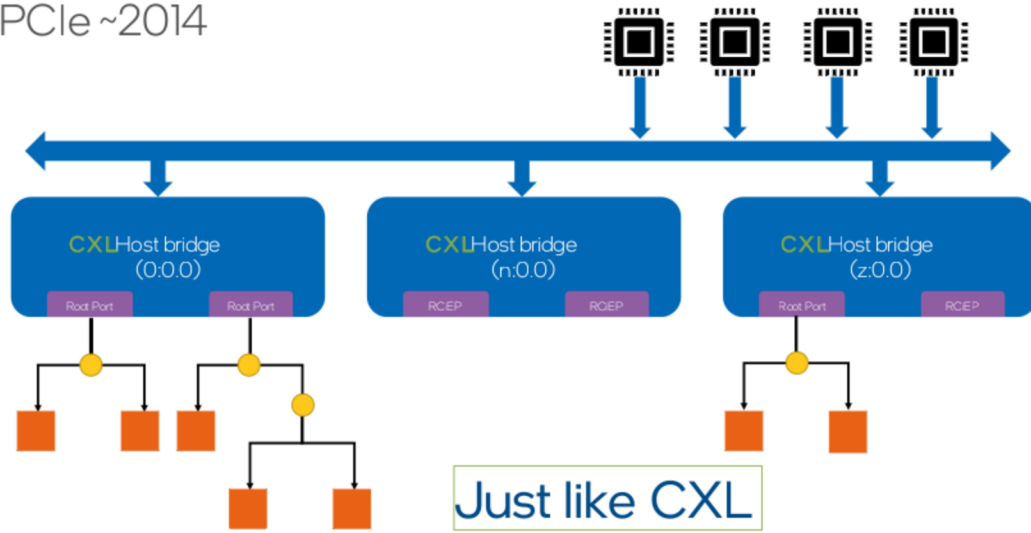
# PCIe in QEMU

## What we all know and love

- Single root complex
  - Endpoints
  - Root ports
  - Switches
- All traffic is funneled to the single host bridge
  - QPI/UPI (not modeled)



PCIe ~2014



This is what a modern platform actually looks like.

CXL is building on top of a modern platform.

Is this going to work with Q35?

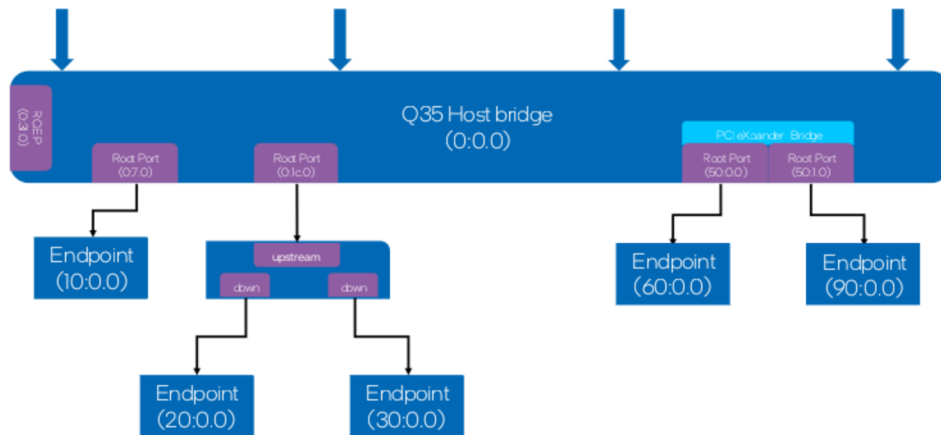
## Options

- Use a single host bridge
  - Fine for basic driver bring-up
  - Limited for interleave scenarios
  - Risk?
- Replace Q35 with something new
  - A lot of work for not much gain (for CXL)
  - Support Burden
- PXB
  - Good compromise

Single host bridge limits testable use-cases and risks adding bugs to core functionality. We need something that understands multiple host bridges, and a new thing would work.

PXB solves the issue in a perfectly acceptable way.

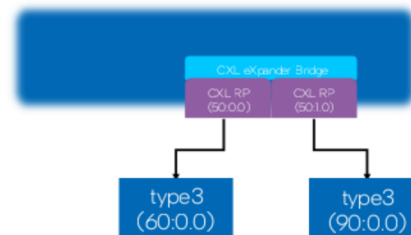
## PCI eXpander Bridge(PXB)



A PXB kind of looks like a PCI-PCI bridge architecturally.  
This allows us to keep existing PCIe mechanisms while introducing distinct CXL topologies (including interleave across PXBs)

## CXL Components

- CXL Type 3 device
  - hw/mem/cxl\_type3.c
- CXL Root Port
  - hw/pci-bridge/cxl\_root\_port.c
- CXL PXB
  - hw/pci-bridge/pci\_expander\_bridge.c



Now we can get to emulating the different components...

## CXL utilities

### hw/cxl/\*

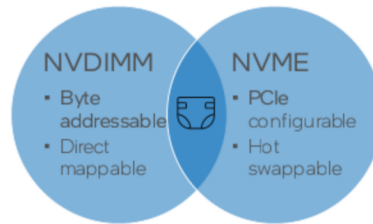
- `cxl-component-utils.c`
  - DVSEC helpers
  - .cache/mem MMIO handler
- `cxl-device-utils.c`
  - Device MMIO
  - Memory device MMIO
  - Mailbox MMIO
- `cxl-mailbox-utils.c`
  - Mailbox command implementations
- `pci_expander_bridge.c`
  - New CXL PxBisms
  - MMIO for host bridge
  - Windows (\*not spec'd)
- `cxl_root_port.c`
  - PCIe root port setup
  - DVSEC initialization

So many components and utility libraries just for a simple "Hello World"

## CXL Type 3 Device (Memory Device)

"PCI and NVDIMM had a coherent byte addressable baby..."

- Ben Widawsky



## CXL Type 3 Device

- NVDIMM & PCI had a baby...
- Inherits from both interfaces
- Mailbox handling

```
static const TypeInfo ct3d_info = {  
    .name = TYPE_CXL_TYPE3_DEV,  
    .parent = TYPE_PCI_DEVICE,  
    .class_init = ct3_class_init,  
    .instance_size = sizeof(CXLType3Dev),  
    .instance_init = ct3_instance_init,  
    .instance_finalize = ct3_finalize,  
    .interfaces = (InterfaceInfo[]) {  
        { TYPE_MEMORY_DEVICE },  
        { INTERFACE_CXL_DEVICE },  
        { INTERFACE_PCIE_DEVICE },  
        {}  
    },  
};
```

The code very clearly indicates it's PCI + a memory device.  
Specifically (not exclusive to a type 3 device), mailbox handling sets it apart.



## ACPI

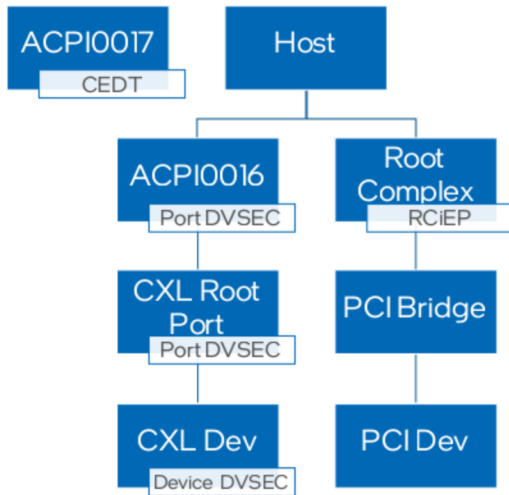
- ACPI0016
  - Per host bridge
  - \_OSC
  - More in the future?
- ACPI0017
  - singleton
  - CEDT
    - CHBS
    - More in the future?
- hw/acpi/cxl.c

CXL requires ACPI per the spec to be useful.  
QEMU provides everything we need to make it work.

# Linux

## Driver Design

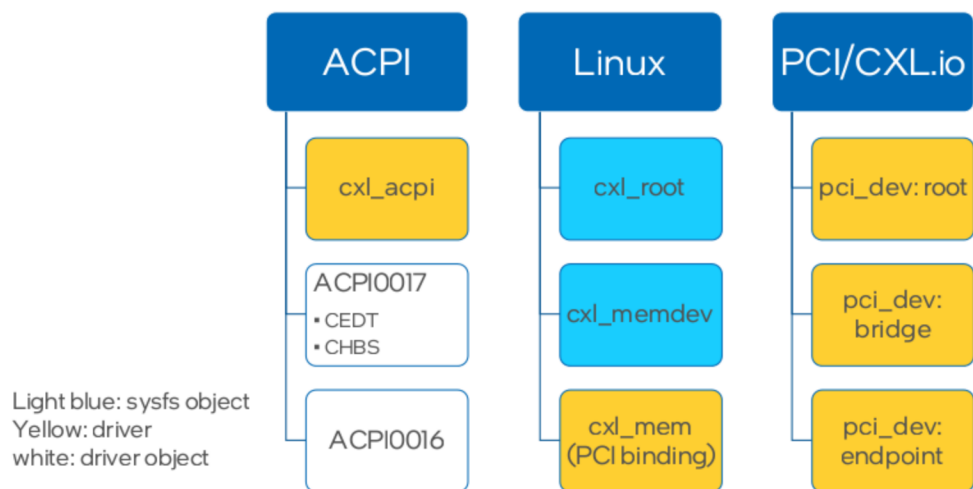
## CXL Type-3 vs PCI Enumeration



- A driver named "cxl\_acpi" attaches to the ACPI0017 instance and is thus informed of the presence of the CEDT.
- Typical ACPI PCI enumeration runs and attaches ACPI0016 as a "companion device" to the corresponding root complex
- PCI Discovery finds CXL Dev and binds a driver named "cxl\_mem" by Class Code.
- HDM decoder validation and programming is coordinated via the cxl\_acpi\_driver that is aware of the ACPI0016 objects that are participating in an interleave set.

CXL and PCI enumeration are distinct. CXL can be thought of as an overlay to PCIe that is accomplished via DVSEC and MMIO.

# CXL.mem Objects



## CXL.mem to libnvdimm objects (future)



1. One NVDIMM bus for all CXL.mem persistent memory
  - cxl\_acpi is the root driver for x86 and ACPI ARM, other archs / non-ACPI ARM may provide a different root description for CXL.mem resources
2. A mapping is a "window + port + dev" tuple
3. cxl\_acpi attaches to the root of the HDM decoder hierarchy and interfaces with the CXL.mem core for HDM topology reconfiguration.
4. libnvdimm and ndctl translate existing NVDIMM provisioning flows to CXL operations

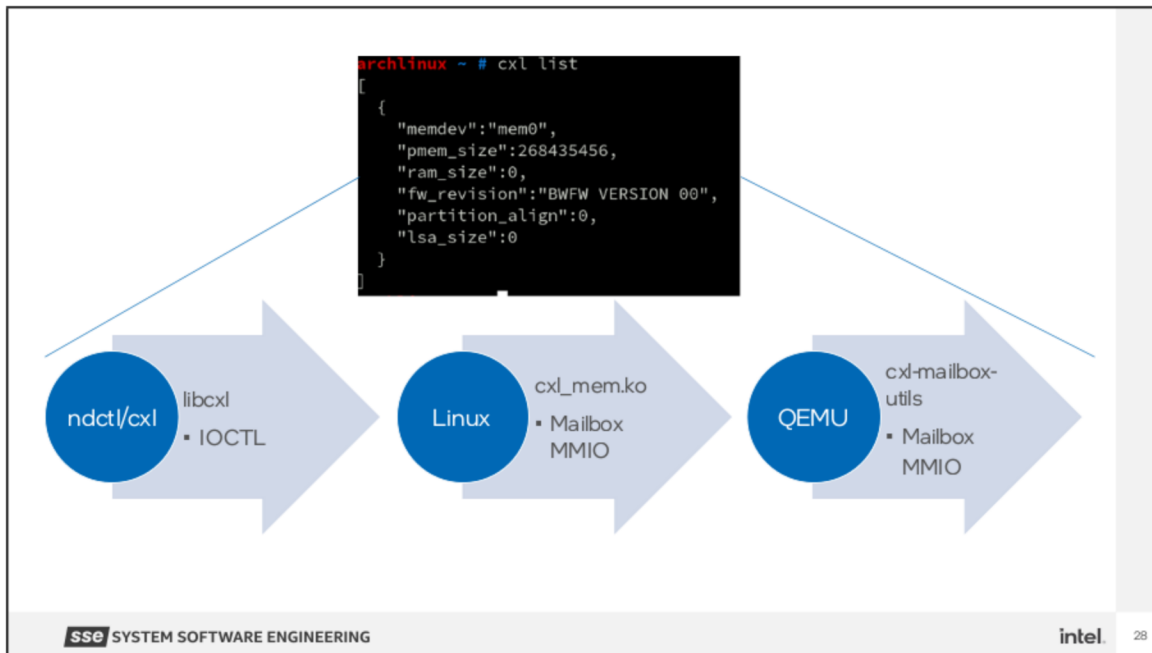
## Interfacing with CXL

- Sysfs
  - /sys/bus/cxl/
- IOCTL
  - QUERY
  - SEND
    - Managed command set
    - RAW escape command
- Future
  - Error reporting

## Putting it all together

- QEMU Emulation
  - Memory region
  - MMIO
  - Mailboxes
- Kernel driver
  - Documentation!
- Tools
- 2.0 spec is available

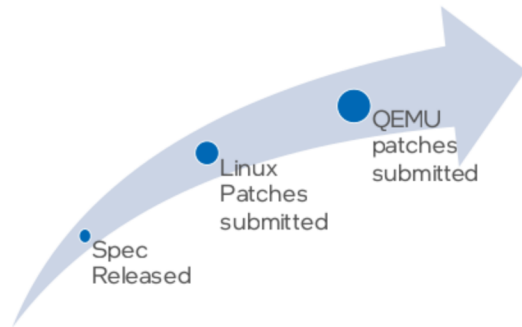






# Outcome

- November 10th 2020



**PATCHBOMB ALL THE THINGS**



## Results

- QEMU v2 patches sent
  - Minimal review
- Linux driver v3 sent
  - Moderate review
- Nothing upstream
- Still no hardware to test on
- Still no other viable pre-silicon platform to test on.
- Made our deadline!
- This talk!!!

# Call to Action

## Missing Pieces

- Linux
  - DOE
    - CDAT
  - DPA mapping (WIP)
  - Interleave
    - Provisioning
    - Recognition
  - Hotplug
    - Hot add
    - Managed remove
  - Asynchronous mailbox
- Userspace
  - Testing
- QEMU
  - Better tests
  - DOE
    - CDAT
  - Upstream/Downstream Ports
  - Interleave Support
    - Host bridge
    - switch
  - More firmware commands
  - Hotplug support
  - Error testing
  - Interrupt support
  - -----
  - Make Q35 CXL capable
  - CXL type 1 and 2 devices
  - CXL 1.1

Everything below dashed line is not of importance to my team.  
Interleave and CDAT are probably the next things we really need.

## Making it useful

- `qemu-system-x86_64`
  - `-machine q35,<cxl>`
  - `-object memory-backend-file,id=cxl-mem1,share,mem-path=cxl-type3,size=512M`
  - `-device pxb-cxl,id=cxl.0,bus=pcie.0,bus_nr=52,uid=0,len-window-base=1>window-base[0]=0x4c0000000,memdev[0]=cxl-mem1`
  - `-device cxl-rp,id=rp0,bus=cxl.0,addr=0.0,chassis=0,slot=0`
  - `-device cxl-type3,bus=rp0,memdev=cxl-mem1,id=cxl-pmem0,size=256M`
- CXL utilities part of `ndctl` being developed
  - <https://github.com/pmem/ndctl/tree/cxl-2.0v1>

## Thanks

- Mahesh Natu – CXL architectural diagrams and high level
- Dr. Debendra Das Sharma – CXL usages
- Dan Williams – Linux diagrams



36

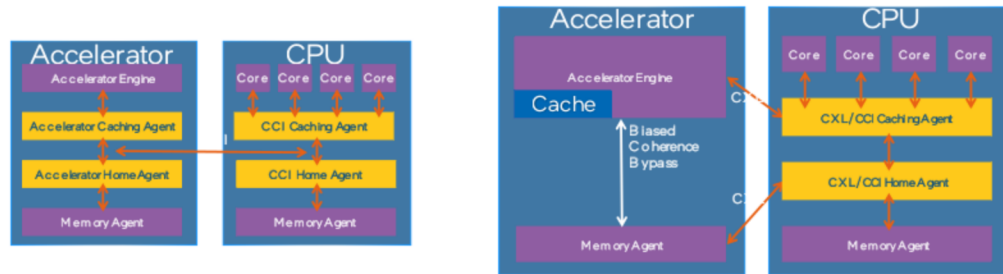
Well, thanks for your ears and eyes. I hope you got something out of it, and I look forward to any feedback people might have.





# Backup

## Asymmetric Protocol => Reduced Complexity



## CXL 1.1 -> CXL 2.0

Feature	Description
CXL PCIe End-Point	CXL device to be discovered as PCIe Endpoint Support of CXL 1.1 devices directly connected to Root-Port or Downstream Switch Port
Switching	Single level of switching with multiple Virtual Hierarchies (cascaded possible in a single hierarchy) CXL Memory Fan-Out & Pooling with Interleaving CXL Cache is direct routed between CPU and device with a single caching device within a hierarchy. Downstream port must be capable of being PCIe.
Resource Pooling	Memory Pooling for Type3 device – Multiple Logical Device (MLD), a single device to be pooled across 16 Virtual Hierarchies.
CXL cache CXL mem enhancements	Persistence (Global Persistence Flush), Managed Hot-Plug, Function Level Reset Scope Clarification, Enhanced FLR for CXL Cache/Mem, Memory Error Reporting and QoS Telemetry
Security	Authentication and Encryption – CXL.I/O uses PCIe IDE, CXL defines similar capability for CXL.Mem
Software Infrastructure/ API	ACPI & UEFI ECNs to cover notification and management of CXL Ports and devices CXL Switch API for a multi-host or memory pooled CXL switch configuration and management