Exploring CXL Memory: Configuration and Emulation

Fsas Technologies Inc
Yasunori Goto
2024.Oct.28

Table of Contents

Fsas Technologies

Note:

- Though I tried to make sure there is no mistakes in my presentation, there might be mis-understandings or inaccuracies yet
  - If you can find mistakes, please let me know

Introduction

What is CXL?

Fsas Technologies

- CXL is abbreviation of Compute Express Link

It is a new interconnect specification that connects devices such as PCIe

- It is suitable to connect Smart devices like GPGPU, SmartNIC, FPGA, Computational Storage, and so on
  - Offloading processing is necessary due to CPU performance limitations
    - GPGPU、FPGA、Smart NIC must handle the processing instead
  - It allows interactive access between CPUs and CXL devices by additional cache coherency protocol
    - This will enable more efficient data exchange between them

In addition, it is also useful to expand memory (volatile memory and persistent memory)

- Memory capacity needs to be increased
  - While the number of CPU cores has increased, memory capacity has not kept pace
  - Because DDR is a parallel interface, it is difficult to increase the number of CPU pins to connect more memory
  - The CXL specification also allows for increased memory capacity

This specification is expected for next generation servers in the AI era

I talked about its summary in last year's OSSJ. Please check the below slide

https://ossjapan2023.sched.com/event/1TypM/compute-express-linkcxl-the-next-generation-interconnect-overview-and-the-status-of-linux-yasunori-goto-fujitsu-ltd

Expectations for CXL memory

Fsas Technologies

- CXL memory is expected to meet the following requirements

Increase memory capacity

- Many software, such as AI/machine learning, HPC workloads, and DB demand huge amounts of memory
- CXL allows connecting large-capacity memory for such software

Increase memory bandwidth

- Memory bandwidth is sometimes a bottleneck
- CXL offers additional memory bandwidth

More efficient resource utilization

- Users want to make more efficient resource utilization and improved system scalability
- CXL provides a way to create memory pools accessible by multiple hosts

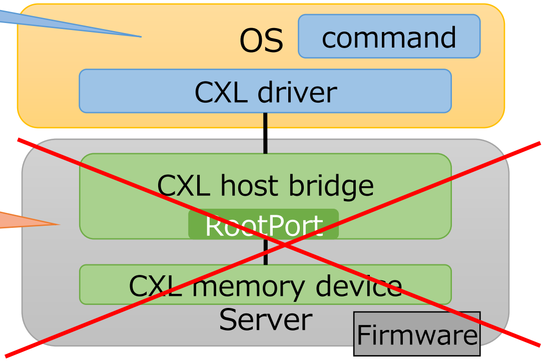
- In addition, software and hardware support for CXL memory devices is more advanced than support for GPGPUs and SmartNICs
  - GPGPU and other devices require vendor-specific implementations
  - However, memory expansion benefits from common development within the CXL specification
  - Therefore, CXL memory is expected to be the first CXL device released

Current problem of CXL

- What is required to use CXL?

Software
- CXL drivers for the Operating System
- OS management command for CXL devices
- An enhanced kernel for CXL memory (desirable)

Hardware
- A CPU with CXL support, including a CXL host bridge
- A CXL-capable platform
  - CXL-capable PCIe connections, a CXL Switch, and CXL-supporting firmware
- Officially released CXL devices



Real CXL environment

- Unfortunately, officially released CXL devices are not yet available
  - Currently, only sample devices are provided, and only a few people can use them
- However, many software developers need to prepare for the release
  - OS drivers/Linux kernel developers,
  - Management software developers
  - Middleware developers
  - Etc.
- Additionally, researchers need to investigate CXL's characteristics

⇨ **CXL hardware emulation environment is crucial for such preparation !!**

QEMU is useful for CXL memory emulation

Fsas Technologies

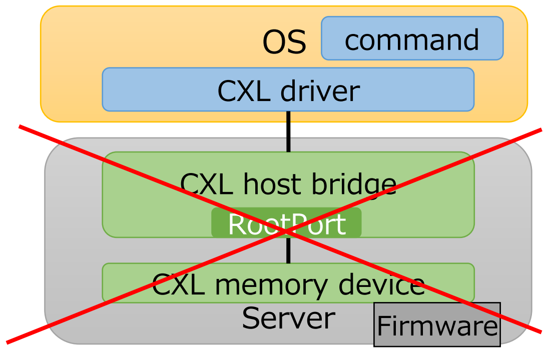
- To emulate CXL memory device, QEMU is useful

What is QEMU?

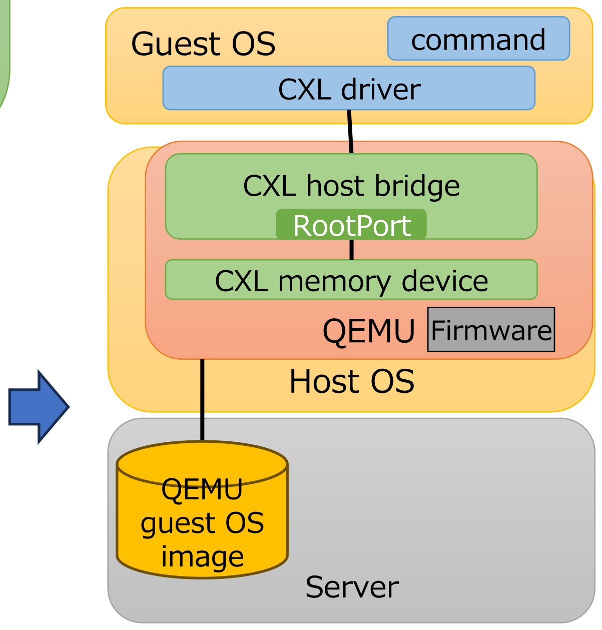
- QEMU is an Open Source machine emulator
  - It can emulate same or different architectures and platforms on your machine
  - It handles memory and IO access and emulates them as needed
- If you use KVM as a virtual machine on Linux, then you are already using a part of QEMU
  - KVM utilizes hardware-assisted virtualization features such as Intel Vt-x
    - It detects and intercepts CPU instructions, I/O access, or memory access that should not be executed on the VM guest
    - QEMU emulates these actions instead

Why is QEMU useful for CXL emulation?

- Many CXL emulation features are already implemented in QEMU
- Its development is very active
- It supports not only simple connection, but also complex CXL memory configurations
- This makes it an excellent environment for software developers and researchers



Real CXL environment



QEMU CXL emulation environment

Problem to use QEMU CXL emulation

Fsas Technologies

The information provided is insufficient for average users

- While there is a document for CXL emulation in QEMU official site, it seems to be written primarily for experts
  - https://www.qemu.org/docs/master/system/devices/cxl.html

Necessary information is scattered across various places

- It requiring extensive searching
  - QEMU site describes only its options, but you need to know how to use cxl command in the guest OS too
  - You may need to examine the source code or git log of related software(QEMU, cxl command, or Kernel) to find the cause of errors

Today I will talk about how to emulate, configure, and use a CXL memory device environment with many tips and my recommendations

Summary of QEMU CXL emulation

Use case of QEMU CXL emulation

Fsas Technologies

What you can do by QEMU CXL emulation

Studying how CXL devices are presented in Linux

Studying the behaviors of new features of Linux kernel for CXL memory

- E.g. memory tiering, Weighted interleave, etc.

Developing and testing CXL drivers, commands, and the Linux Kernel

- Actually, my team member found a race condition bug of memory hotplug in the Linux Kernel by the emulation environment

Developing upper layer software for CXL

- Middleware, Orchestrator, and any other new generation software

What you can NOT do by QEMU CXL emulation

Performance evaluation

- It cannot emulate bandwidth and latency of real CXL memory device
- Additionally, memory access speeds in the emulated environment are significantly slower than in a typical KVM guest environment

Expect perfect emulation

- This emulation is still under development
- There are some features that have not been implemented yet
- Additionally, there are some limitations yet
  - For example, a bug currently requires users to fully shut down and start the guest OS instead of using a simple 'reboot' command

What features are currently emulatable?

Fsas Technologies

- Basically, CXL 2.0 spec or later is supported
  - Since making a generic emulation of CXL1.1 was difficult for some reasons, it was skipped
- CXL memory can be emulated
  - Both volatile memory and persistent memory are available
  - Other devices are not emulated yet
- You can configure more complex connections
  - Many memory devices can be connected via CXL switches
- Memory interleave configuration is also available
- Hotplug an entire CXL memory device is available
  - Based on CXL 2.0
  - Today, I will not talk about it due to not having enough time
- Hotplug a part of a CXL memory device might be available
  - Based on the Dynamic Capacity Device(DCD) feature of CXL3.0
  - Unfortunately, I have not tried it yet, so I will not talk about it either

Feature	Summary	Emulation environment	Today's talk
Volatile memory (CXL v1.1)	Volatile memory device of CXLv1.1	Not supported	No
Volatile memory (CXL v2.0 or later)	Volatile memory device	Available	**Yes**
Persistent memory	Persistent memory device	Available	**Yes**
Other devices	Emulation of GPGPU, SmartNIC, etc.	Not implemented	No
CXL Switch	Enable hierarchical connection of devices like PCIe, and allowing connect many devices	Available	**Yes**
CXL Memory interleave	Increase bandwidth by accessing multiple memory device	Available	**Yes**
Memory hotplug (hole device)	Hotplug a memory device unit	Available	No
Memory hotplug (Some part of device)	Hotplug a part of memory device (Dynamic Capacity Device by CXLv3.0)	Might be available (but not sure)	No
Memory Sharing	Share CXL memory devices inter servers	Not implemented	No
Fabric Connection	Enable fabric network connection between hosts and devices like Ethernet	Not implemented	No

Today, I'll talk about the features marked as "Yes" in the table

Required prerequisite knowledge

Fsas Technologies

- To successfully follow this guide, you will need the followings
  - Operation of KVM virtual machine
    - This is useful for preparing your guest image of the emulation environment
  - Download QEMU upstream code, build and start it
    - This is key for creating your emulation environment
  - Download, compile and install the upstream Linux Kernel
    - You need to change the kernel build config options
      - The default kernel provided by distributors may not work well in the CXL emulation environment
    - Build and install it in the guest for the emulation environment

Today, I'll NOT talk about the details of the above.
If you need them, please check them later

  - Basic CXL knowledge is required

→ A summary of the relevant CXL components and a concept will be provided in the next section

Basic information of CXL

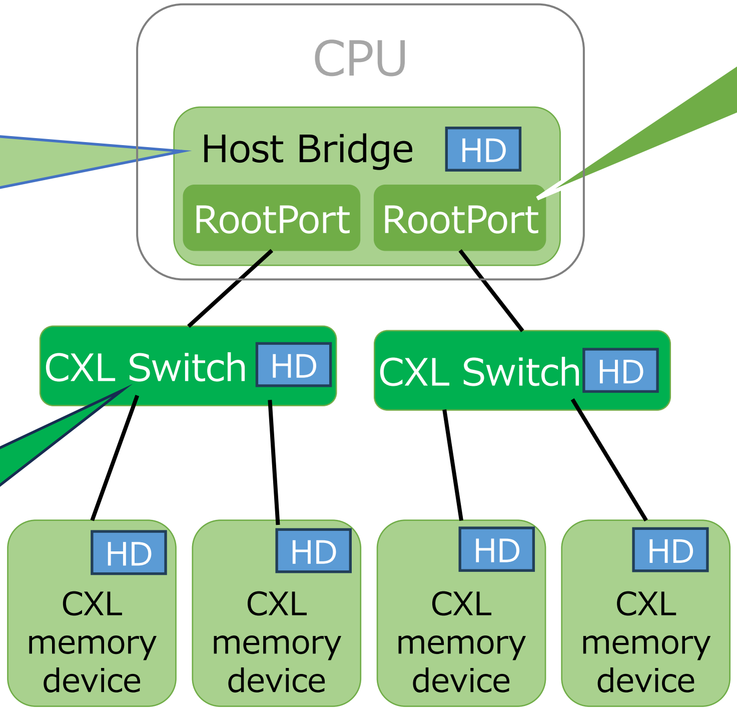
Key hardware components of CXL

Fsas Technologies

- QEMU can emulate the following hardware components
- In other words, you need to specify them to start your emulation environment

Root Port
- Downstream connection port of the Host Bridge
- In the emulation setting, this is used to define how CXL objects are connected

Host bridge
- The root of the connection tree for CXL devices, equivalent to the Host Bridge in PCIe
- Typically integrated into the CPU

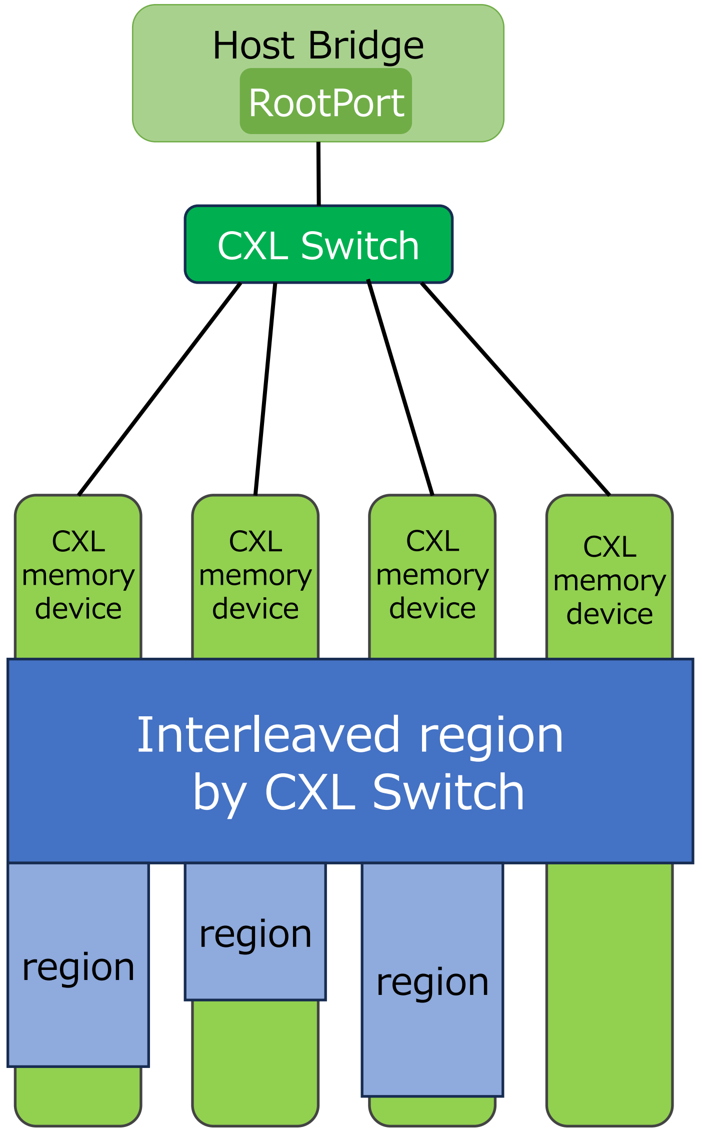


CXL Switch
- Connects an upstream port to multiple downstream ports
- Its connection information is used for emulation

HD is "HDM decoder"
- Its role is the translation between the "host physical address" and the "device physical address"
  - It is especially important for memory interleave feature
- At least, you need to find the "root decoder", which is included in Host Bridge, to use CXL memory

The concept of Region

- To use CXL memory devices on Linux, you need to know about the “region”
  - A region is an area allocated from a part of CXL memory device, or from multiple memory devices that are configured for interleaving
- In QEMU CXL emulation, you need to configure one or more regions to use the CXL memory devices after your guest OS boot
  - I’ll explain how to configure it in this talk



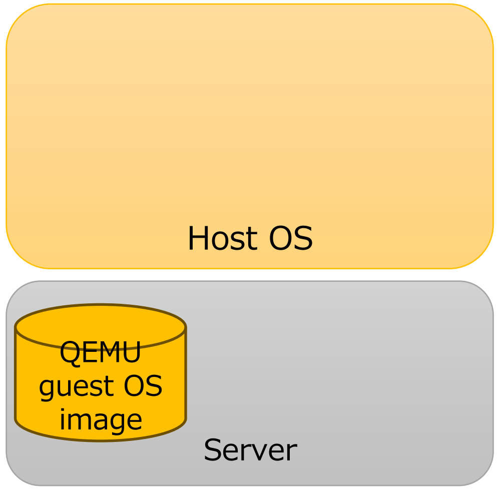
Preparation of emulation environment

Preparation of QEMU guest image (1/3)

- Creating your guest image for QEMU CXL emulation
  - Please prepare a bare metal machine for your QEMU emulation environment

> - If you can use virt-install, cockpit or virt-manager with KVM, it is an easy way to make your guest image
>   - Create one guest machine, and install a Linux OS
>   - Ensure there is enough storage to build kernel on the guest
> - After install the VM guest, I recommend configuring the serial console of the guest
>   - Its information will be displayed on the QEMU console

  - After VM guest has started, please check and record the options specified for the qemu command that is executed for your guest
    - Just enter "ps auxw |grep qemu" to check it
    - Most of these qemu command options are used for emulation environment
      - CPU, DRAM memory, storage, network settings
      - Here is an example from my environment



QEMU CXL emulation environment

```
qemu-system-x86_64 -drive file=<place of guest image>,format=qcow2,index=0,media=disk,id=hd -m 16G,slots=8,maxmem=32G ¥
-machine type=q35,accel=kvm, -smp 16 -enable-kvm -nographic -net nic ¥
-net bridge,br=virbr0 -device e1000,netdev=net0 -netdev user,id=net0,hostfwd=tcp::2222-:22
```

Preparation of QEMU guest image (2/3)

- You need to prepare a new kernel that can work in the CXL emulation environment
  - You need to build and install it on your VM guest
  - At a minimum, the following CONFIG options must be enabled when you build the new kernel. They must be enabled even in a real CXL environment

```
CONFIG_CXL_BUS, CONFIG_CXL_PCI,  CONFIG_CXL_ACPI, CONFIG_CXL_PMEM,
CONFIG_CXL_MEM, CONFIG_CXL_PORT, CONFIG_CXL_REGION
```

  - I recommend enabling the following CONFIG option too. (I'll explain the reason later)

```
CONFIG_MEMORY_HOTPLUG_DEFAULT_ONLINE
```

  - Additionally, the following CONFIG option is essential only for the CXL emulation environment

```
CONFIG_REGION_INVALIDATION_TEST
```

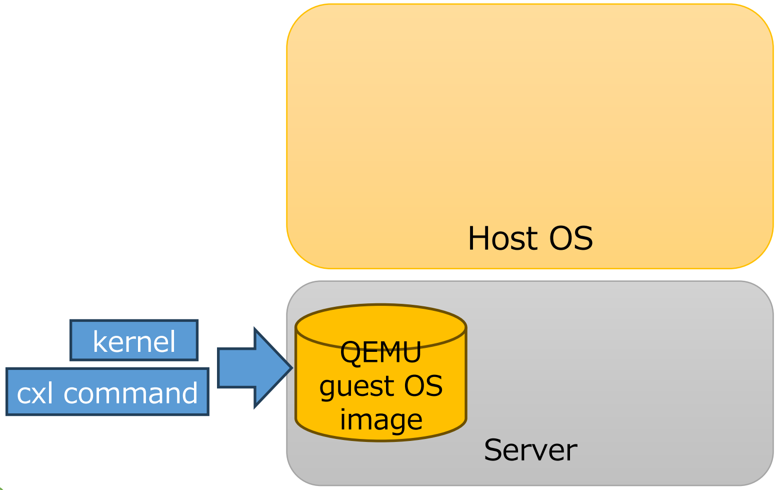
    - If it is not enabled, some operations on the guest OS may not work with errors

      - In some case, the cxl driver needs to use a special CPU instruction for cache write back and invalidation (wbinvd)
      - This is typically only for bare metal x86 environments, but this config option enables it for testing in the CXL emulation environment
      - If you want to more detail, please check the commit d18bc74aced65 in kernel source code

    - Its option is probably disabled in the distributor's kernel
  - Since many developers are enhancing the kernel and cxl driver, using newer kernel is better

Preparation of QEMU guest image (3/3)

- Getting and building the “cxl” command on your guest
  - The cxl command is used for configuring and checking the status of CXL devices
  - A newer version of this command is better for CXL emulation due to bug fixes and new features
    - The newest version is v80, which was released 2024/Oct/8th
    - I recommend downloading and build it from Github
  - Its source code is included ndctl repository, which is the command for persistent memory
    - https://github.com/pmem/ndctl



QEMU CXL emulation environment

- After preparing the kernel and the cxl command in your guest image, you can shutdown the KVM guest OS
  - To start CXL emulation environment, you need to start QEMU with different options compared to a normal KVM setup

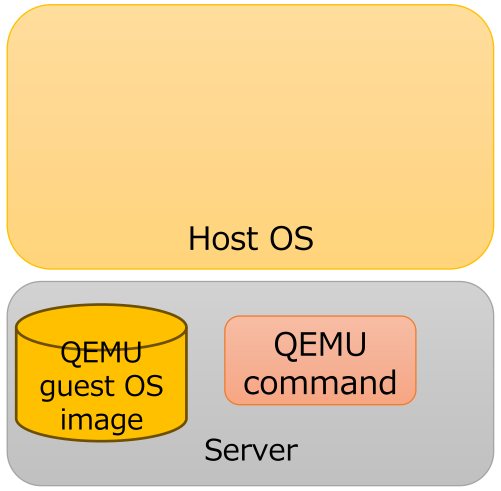
Download and build the newest QEMU

Fsas Technologies

- As mentioned before, CXL emulation development is very active
- Therefore, newer QEMU has newer CXL features and bug fixes
  - Since the version of QEMU included with the distributor is relatively old, I recommend downloading newer QEMU source code from the official site, and building it on your bare metal machine

```
$ git clone https://gitlab.com/qemu-project/qemu.git
```

  - In QEMU, there are no special notice for build unlike the Linux kernel
  - However, it might be better not to install the built QEMU binary to avoid conflicts with the distributor's packages
    - I recommend executing the built QEMU command from your home directory



QEMU CXL emulation environment

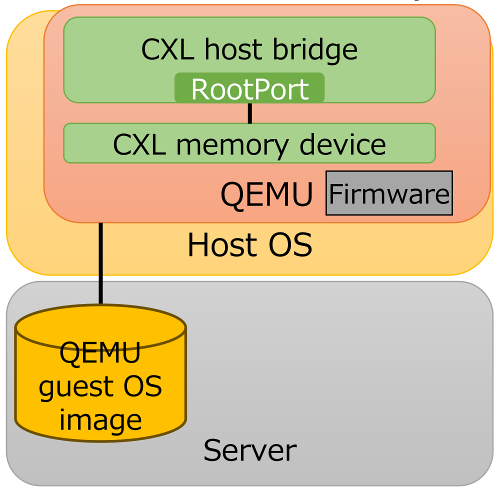
How to start QEMU CXL emulation environment

How to Start QEMU CXL Emulation

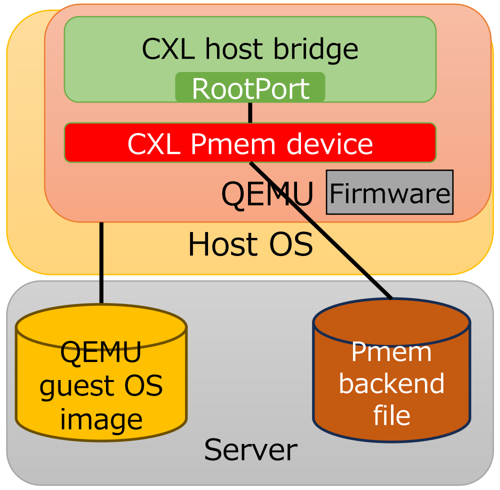
Fsas Technologies

- I'll show you three examples to start QEMU

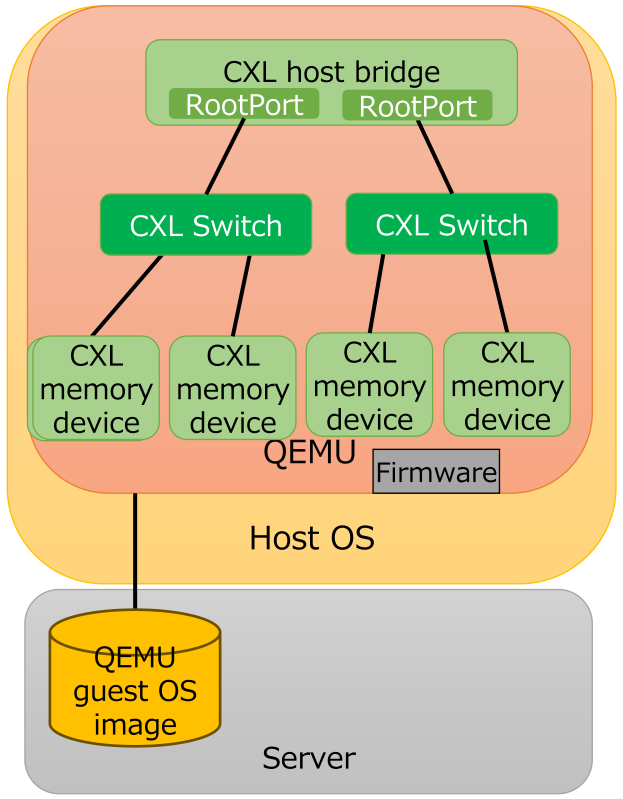
1) Simple connection of CXL volatile memory



2) CXL Persistent memory



3) Complex connection
- Two Switches
- Four CXL memory devices



Simple connection of CXL volatile memory (1/5)

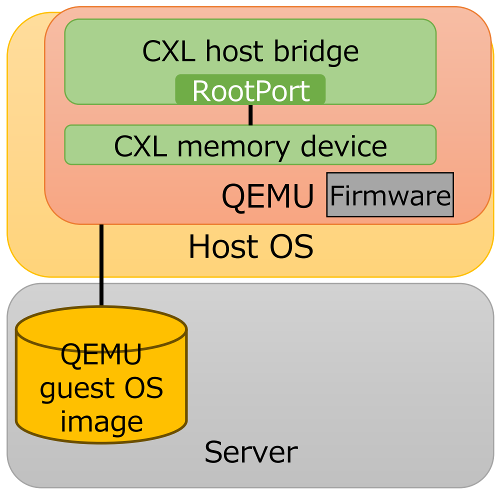
- QEMU defines various devices in its command options, and it is same for CXL emulated devices
- To specify them easily, I recommend creating a simple shell script to start it
  - Here is an example of a simple connection of CXL memory setup, as shown in the right figure
    - One Host Bridge with one root port, and one CXL volatile memory

The text in black font can remain the same as when you executed your VM guest

```
$ cat run_cxl_emu.sh
sudo /<qemu built directory>/qemu-system-x86_64 -drive file=<prepared your guest disk
image>,format=qcow2,index=0,media=disk,id=hd ¥
-m 16G,slots=8,maxmem=32G -machine type=q35,accel=tcg,cxl=on -smp 16 ~~-enable-kvm~~ -
nographic -net nic -net bridge,br=virbr0 -device e1000,netdev=net0 ¥
-netdev user,id=net0,hostfwd=tcp::2222-:22 ¥
-object memory-backend-ram,id=vmem0,share=on,size=256M ¥
-device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.1 ¥
-device cxl-rp,port=0,bus=cxl.1,id=root_port13,chassis=0,slot=2 ¥
-device cxl-type3,bus=root_port13,volatile-memdev=vmem0,id=cxl-vmem0 ¥
-M cxl-fmw.0.targets.0=cxl.1,cxl-fmw.0.size=4G
```

The text in red font indicates what you need to add or modify for emulation

Since they are many new options, I'll describe the meaning of each option from the next page



QEMU CXL emulation environment

Simple connection of CXL volatile memory (2/5)

- These two lines are used to create CXL type3 memory device

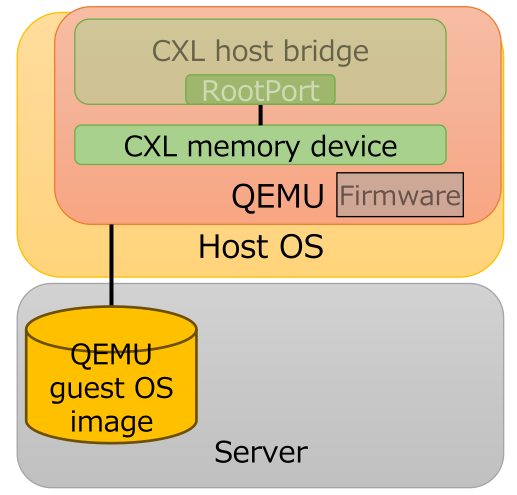
Definition of backend device for memory emulation

- "memory-backend-ram" means that you use DDR DRAM of the bare metal server as the backend for the emulated device
- Size is allocation size, and it also means emulated memory size
- Id is a unique name for this backend (id=mem0)

```
nographic              bridge,br=virbr0 -device e1000,netdev=net0 ¥
-netdev     ,id=net0,hostfwd=tcp::2222-:22 ¥
-object memory-backend-ram,id=vmem0,share=on,size=256M ¥
-device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.1 ¥
-device cxl-rp,port=0,bus=cxl.1,id=root_port13,chassis=0,slot=2 ¥
-device cxl-type3,bus=root_port13,volatile-memdev=vmem0,id=cxl-vmem0 ¥
                                                 ize=4G
```

Definition of CXL memory device

- "cxl-type3" means CXL memory device
- "bus=" specifies which port is used for connection of this device
- "volatile-memdev=vmem0" specifies the backend ID
- "id" is a unique name for this CXL memory (=cxl-vmem0)



QEMU CXL emulation environment

Simple connection of CXL volatile memory (3/5)

- Definition of CXL Host bridge and root port

Definition of the Host Bridge

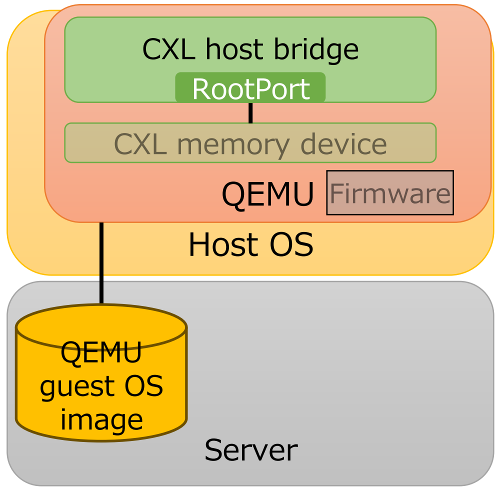
- In this example, the bus number of the Host Bridge is 12 (pbc-cxl.bus_nr=12)
- It is connected under PCIe bus 0
- “cxl.1” is identifier of this host brige

- “cxl=on” must be specified to use CXL emulation

```
$
su                                                     ve file=<gue
in
-m 16G,slo                m=32G -machine type=q35,accel=tcg,cxl=on -smp 16 ~~-enable-kvm~~ -
nographic         ic -net bridge,br=virbr0 -device e1000,netdev=net0 ¥
-netdev       ,id=net0,hostfwd=tcp::2222-:22 ¥
-object memory-backend-ram,id=vmem0,share=on,size=256M ¥
-device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.1 ¥
-device cxl-rp,port=0,bus=cxl.1,id=root_port13,chassis=0,slot=2 ¥
-device cxl-type3,       root_port13,volatile-memdev=vmem0,id=cxl-vmem0 ¥
-M cxl-fmw.0.target                0,size=4G
```

Definition of the root port of the Host bridge

- “cxl-rp.port=0” specifies the port number
- “bus=cxl.1” indicates that this port belongs to the “cxl.1” Host Bridge
- The bus number of devices under this root port becomes 13 by “id=root_port13”
- The (slot, chassis) pair is mandatory and must be unique for each



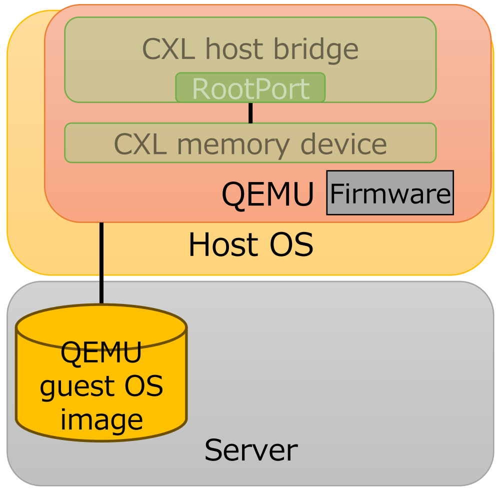
QEMU CXL emulation environment

Simple connection of CXL volatile memory (4/5)

- This line is for firmware emulation, and reserves guest physical address space for CXL

- “cxl-fmw” is to emulate CXL Fixed Memory Windows(CFMWS), which shows physical address map of CXL memory devices from firmware to OS
  - In this example, “targets.0” is used for the CXL root port (=“bus=cxl.1”)
    - “cxl-fmw.0.size” must be specified with a size larger than the total size of CXL devices that belong to this table(=“4G”)

```
-device cx                , bus=cxl.1, id=root_port13, chassis=0, slot=2 ¥
-device cxl-type3, bus=root_port13, volatile-memdev=vmem0, id=cxl-vmem0 ¥
-M cxl-fmw.0.targets.0=cxl.1, cxl-fmw.0.size=4G
```



QEMU CXL emulation environment

Simple connection of CXL volatile memory (5/5)

Fsas Technologies

- You need to disable the hardware assist for virtualization
  - If not, it can cause unpredictable issues in the emulation
    - In our case, an illegal instruction error occurred in the emulation environment until we disabled it
  - Since CXL memory is interleaved at granularities as fine as 64 bytes, the emulator needs to translate read and write access at this size
  - However, hardware assist can detect only the page size of the architecture
    - It is 4 KB or more

Important!!

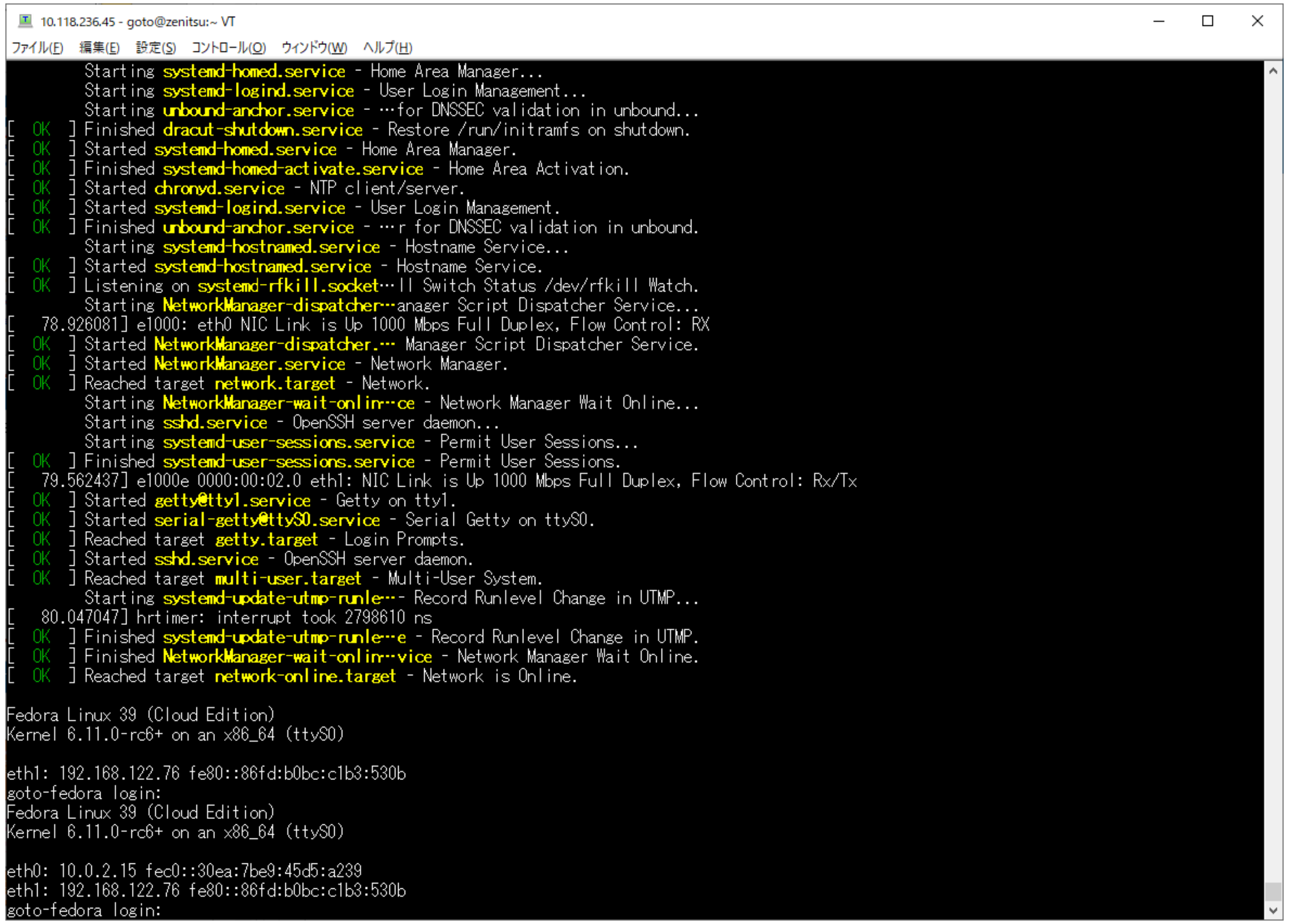
```
$ cat run_cxl_emu.sh
sudo /<qemu built directory>/qemu-system-x86_64 -drive file=<guest disk
image>,format=qcow2,index=0,media=disk,id=hd ¥
-m 16G,slots=8,maxmem=32G -machine type=q35,accel=tcg,cxl=on -smp 16 ~~-enable-kvm~~ -
nographic -net nic -net bridge,br=virbr0 -device e1000,netdev=net0 ¥
-netdev user,id=net0,hostfwd=tcp::2222-:22
-object memory-
-device pxb-cxl
-device cxl-rp,
-device cxl-type3,bus=root_port13,volatile-memdev=vmem0,id=cxl-vmem0 ¥
-M cxl-fmw.0.targets.0=cxl.1,cxl-fmw.0.size=4G
```

- "accel=tcg" should be specified instead of "accel=kvm"
- "-enable-kvm" should be removed

- Though this is not written in QEMU CXL public web site, but this patch commit says more detail reason
  https://lore.kernel.org/all/20240219173153.12114-3-Jonathan.Cameron@huawei.com/T/

After boot up

- Your terminal, which is running QEMU, displays the OS console as shown below
  - You can use this console (or login from another terminal via ssh)


10.118.236.45 - goto@zenitsu:~ VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

```
         Starting systemd-homed.service - Home Area Manager...
         Starting systemd-logind.service - User Login Management...
         Starting unbound-anchor.service - …for DNSSEC validation in unbound...
[  OK  ] Finished dracut-shutdown.service - Restore /run/initramfs on shutdown.
[  OK  ] Started systemd-homed.service - Home Area Manager.
[  OK  ] Finished systemd-homed-activate.service - Home Area Activation.
[  OK  ] Started chronyd.service - NTP client/server.
[  OK  ] Started systemd-logind.service - User Login Management.
[  OK  ] Finished unbound-anchor.service - …r for DNSSEC validation in unbound.
         Starting systemd-hostnamed.service - Hostname Service...
[  OK  ] Started systemd-hostnamed.service - Hostname Service.
[  OK  ] Listening on systemd-rfkill.socket…ll Switch Status /dev/rfkill Watch.
         Starting NetworkManager-dispatcher…anager Script Dispatcher Service...
[   78.926081] e1000: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[  OK  ] Started NetworkManager-dispatcher.… Manager Script Dispatcher Service.
[  OK  ] Started NetworkManager.service - Network Manager.
[  OK  ] Reached target network.target - Network.
         Starting NetworkManager-wait-onlin…ce - Network Manager Wait Online...
         Starting sshd.service - OpenSSH server daemon...
         Starting systemd-user-sessions.service - Permit User Sessions...
[  OK  ] Finished systemd-user-sessions.service - Permit User Sessions.
[   79.562437] e1000e 0000:00:02.0 eth1: NIC Link is Up 1000 Mbps Full Duplex, Flow Control: Rx/Tx
[  OK  ] Started getty@tty1.service - Getty on tty1.
[  OK  ] Started serial-getty@ttyS0.service - Serial Getty on ttyS0.
[  OK  ] Reached target getty.target - Login Prompts.
[  OK  ] Started sshd.service - OpenSSH server daemon.
[  OK  ] Reached target multi-user.target - Multi-User System.
         Starting systemd-update-utmp-runle… - Record Runlevel Change in UTMP...
[   80.047047] hrtimer: interrupt took 2798610 ns
[  OK  ] Finished systemd-update-utmp-runle…e - Record Runlevel Change in UTMP.
[  OK  ] Finished NetworkManager-wait-onlin…vice - Network Manager Wait Online.
[  OK  ] Reached target network-online.target - Network is Online.

Fedora Linux 39 (Cloud Edition)
Kernel 6.11.0-rc6+ on an x86_64 (ttyS0)

eth1: 192.168.122.76 fe80::86fd:b0bc:c1b3:530b
goto-fedora login:
Fedora Linux 39 (Cloud Edition)
Kernel 6.11.0-rc6+ on an x86_64 (ttyS0)

eth0: 10.0.2.15 fec0::30ea:7be9:45d5:a239
eth1: 192.168.122.76 fe80::86fd:b0bc:c1b3:530b
goto-fedora login:
```

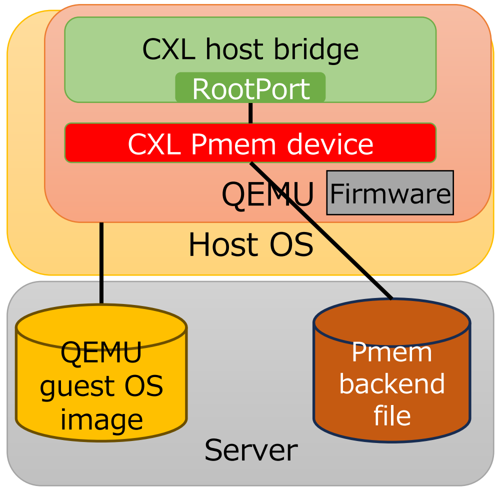
Example of CXL persistent memory

- CXL persistent memory emulation

- A file is better for the backend than RAM to maintain data persistency
- If the file already exists, its size must match the size specified in this line
  - If specified size does not match the actual file size, QEMU will fail to start
- “shared=on” is recommended to save your data as persistent memory
  - If it is “off”, then the written data will not be applied to the file
- In this example, the file size must be 256M

```
nographic ¥
-net nic -net            virbr0 -device e1000,netdev=net0 -netdev
user,id=net0,      wd=tcp::2222-:22 ¥
-object memory-backend-file,id=cxl-mem1,share=on,mem-path=/tmp/cxltest.raw,size=256M ¥
-object memory-backend-file,id=cxl-lsa1,share=on,mem-path=/tmp/lsa.raw,size=2M ¥
-device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.1 ¥
-device cxl-rp,p      bus=cxl.1,id=root_port13,chassis=0,slot=2 ¥
-device cxl-typ          port13,persistent-memdev=pmem0,id=cxl-pmem0 ¥
-M cxl-fmw.0.t            xl-fmw.0.size=4G
```

- Persistent memory has a “Label Storage Area”(=cxl-lsa1), which stores the settings when the internal area of the device is divided for use
- To emulate it, you also need to add its definition with a backend file



QEMU CXL emulation environment

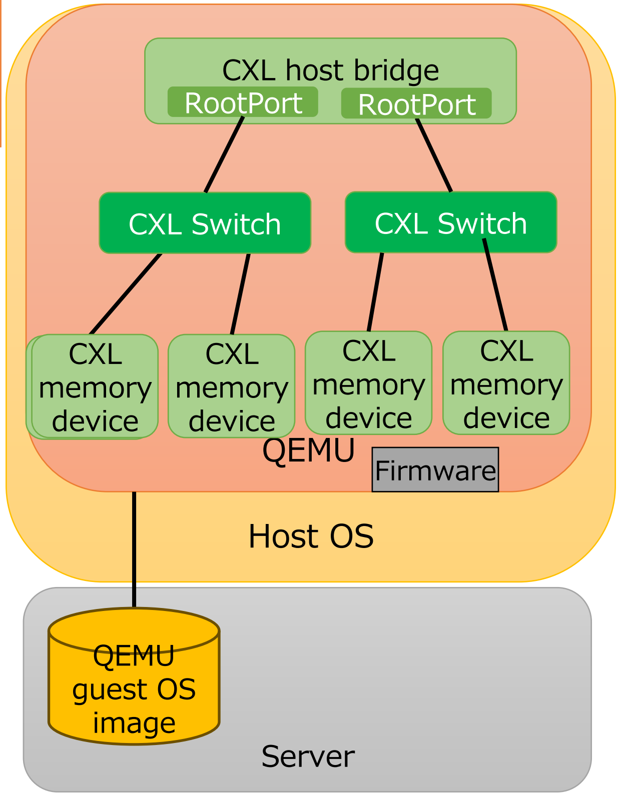
Example of CXL Switch emulation

Fsas Technologies

- You can create a more complex environment
  - Here is an example where four CXL memory devices are connected via two CXL switches

- A CXL switch definition consists of a pair of an upstream port and multiple downstream ports
- In this example, two pairs of switches are defined
- Two memory devices are connected to each switch

```
-object memory-backend-ram,id=cxl-mem2,share=on,size=
-object memory-backend-ram,id=cxl-mem3,share=on,siz
-device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.1 ¥
-device cxl-rp,port=0,bus=cxl.1,id=root_port0,chassis=0,slot=0 ¥
-device cxl-rp,port=1,bus=cxl.1,id=root_port1,chassis=0,slot=1 ¥
-device cxl-upstream,bus=root_port0,id=us0 ¥
-device cxl-upstream,bus=root_port1,id=us1 ¥
-device cxl-downstream,port=0,bus=us1,id=swport0,chassis=0,slot=4 ¥
-device cxl-type3,bus=swport0,volatile-memdev=cxl-mem0,id=cxl-vmem0 ¥
-device cxl-downstream,port=1,bus=us1,id=swport1,chassis=0,slot=5 ¥
-device cxl-type3,bus=swport1,volatile-memdev=cxl-mem1,id=cxl-vmem1 ¥
-device cxl-downstream,port=2,bus=us0,id=swport2,chassis=0,slot=7 ¥
-device cxl-type3,bus=swport2,volatile-memdev=cxl-mem2,id=cxl-vmem2 ¥
-device cxl-downstream,port=3,bus=us0,id=swport3,chassis=0,slot=6 ¥
-device cxl-type3,bus=swport3,volatile-memdev=cxl-mem3,id=cxl-vmem3 ¥
-M cxl-fmw.0.targets.0=cxl.1,cxl-fmw.0.size=32G
```



Operations on the guest OS side after boot

Summary of OS side operations to use CXL memory devices

Fsas Technologies

- Confirmation of CXL memory devices
  - How to confirm the presence of CXL memory
- Operations about regions
  - To use CXL memory devices, you need to configure one or more regions from the guest OS side
  - I'll show you three examples of creating a region, like the examples of QEMU options
    1. Simple connection of CXL volatile memory
    2. CXL Persistent memory
    3. Complex connection

Confirm the emulated memory device and related component

- Simple example
  - One volatile memory device is attached directly to the host bridge

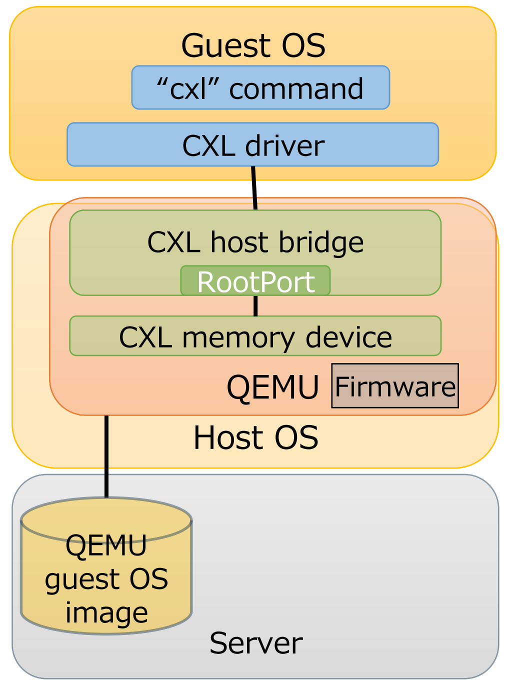
```
# ./cxl list -DMu
[
  {
    "memdevs":[
      {
        "memdev":"mem0",
        "ram_size":"256.00 MiB (268.44 MB)",
        "ram_qos_class":0,
        "serial":"0",
        "host":"0000:0d:00.0"
      }
    ]
  },
  {
    "root decoders":[
      {
        "decoder":"decoder0.0",
        "resource":"0xa90000000",
        "size":"4.00 GiB (4.29 GB)",
        "interleave_ways":1,
        "max_available_extent":"4.00 GiB (4.29 GB)",
        "pmem_capable":true,
        "volatile_capable":true,
        "accelmem_capable":true,
        "qos_class":0,
        "nr_targets":1
      }
    ]
  }
```

device name

root decoder name

- -M lists all memory devices
- -D lists all decoders
- -u shows some data in human readable format (in this case, memory size)

- You can find the name of a memory device and root decoder in this example
- These name need to be specified to created a region



QEMU CXL emulation environment

How to create a region (1/2)

- Simple example
  - To create a region, you need to specify the root decoder, memory device name, and memory usage type as volatile memory

```
# cxl create-region -d decoder0.0 -m mem0 -t ram
{
  "region":"region0",
  "resource":"0xa90000000",
  "size":"256.00 MiB (268.44 MB)",
  "type":"ram",
  "interleave_ways":1,
  "interleave_granularity":256,
  "decode_state":"commit",
  "mappings":[
    {
      "position":0,
      "memdev":"mem0",
      "decoder":"decoder2.0"
    }
  ]
}
cxl region: cmd_create_region: created 1 region
```

- “-t ram” means that this region is used as volatile memory
- OS and user applications can use CXL emulated memory after this command completion

- However, if the kernel of the guest OS is not compiled with the CONFIG_MEMORY_HOTPLUG_DEFAULT_ONLINE option, memory hot-add operations are required after the region creation yet
  - This requires knowledge of memory hotplug
  - Its operation is a bit troublesome for the first trial of CXL emulation
  - This is why I recommend that its option

How to create a region (2/2)

- You can confirm that new memory is added and a node is created by using the free command and the numactl command
  - This is an example one 8GB CXL memory region is created in an environment with two 8GB DDR DRAM NUMA nodes

```
$ free
              total        used        free      shared  buff/cache   available
Mem:       16392992      617092    15845680        8956      217916    15775900
Swap:       8388604           0     8388604

$ numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 7950 MB
node 0 free: 7691 MB
node 1 cpus: 8 9 10 11 12 13 14 15
node 1 size: 8058 MB
node 1 free: 7795 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

Create region

```
$ free
              total        used        free      shared  buff/cache   available
Mem:       23360288      626196    22833044        8956      227316    22734092
Swap:       8388604           0     8388604

$ numactl -H
available: 3 nodes (0-2)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 7950 MB
node 0 free: 7722 MB
node 1 cpus: 8 9 10 11 12 13 14 15
node 1 size: 8058 MB
node 1 free: 7783 MB
node 2 cpus:
node 2 size: 8064 MB
node 2 free: 8064 MB
node distances:
node   0   1   2
  0:  10  20  20
  1:  20  10  20
  2:  20  20  10
```

The total memory size is increased

CXL memory is treated as a CPU less NUMA node

Persistent memory example

```
# cxl list -Mu
{
  "memdev":"mem0",
  "pmem_size":"2.00 GiB (2.15 GB)",
  "serial":"0",
  "host":"0000:0d:00.0"
}

# ndctl zero-labels nmem0
Zeroed 1 nmem

# cxl create-region -d decoder0.0 -m mem0 -t pmem -s 256M
{
  "region":"region0",
  "resource":"0x1290000000",
  "size":"256.00 MiB (268.44 MB)",
  "type":"pmem",
  "interleave_ways":1,
  "interleave_granularity":256,
  "decode_state":"commit",
  "mappings":[
    {
      "position":0,
      "memdev":"mem0",
      "decoder":"decoder2.0"
    }
  ],
  "qos_class_mismatch":true
}
cxl region: cmd_create_region: created 1 region
```

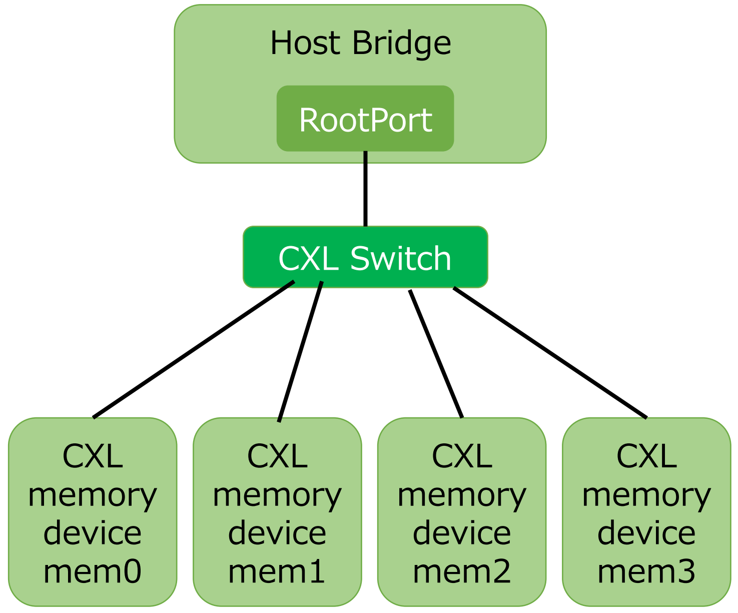
- Currently, you may need to clear “Label Storage Area” first
  - Because the CXL driver's LSA support is inadequate, the namespace creation operation described below may not work unless this operation is performed
  - To clear it, the “ndctl zero-labels” command can be used as a workaround

Note: “cxl zero-labels” cannot be used for it due to the intended design

- For persistent memory, you need to specify “pmem” for the –type option of create-region
- If you specify the size option, you can use a portion of the device
  - In this example, a 256MB size region is created in the 2GB memory device
  - The storage-like nature of persistent memory makes this feature useful in cases where you need different users to access different parts
- After the region creation, you can create namespace in the region by the ndctl command, similar to Non Volatile DIMM

```
# ndctl create-namespace --region region0 -m fsdax
{
  "dev":"namespace0.0",
  "mode":"fsdax",
  "map":"dev",
  "size":"250.00 MiB (262.14 MB)",
  "uuid":"2ae2ae84-f04a-408c-b79d-14ec97fefa7a",
  "sector_size":512,
  "align":2097152,
  "blockdev":"pmem0"
}
```

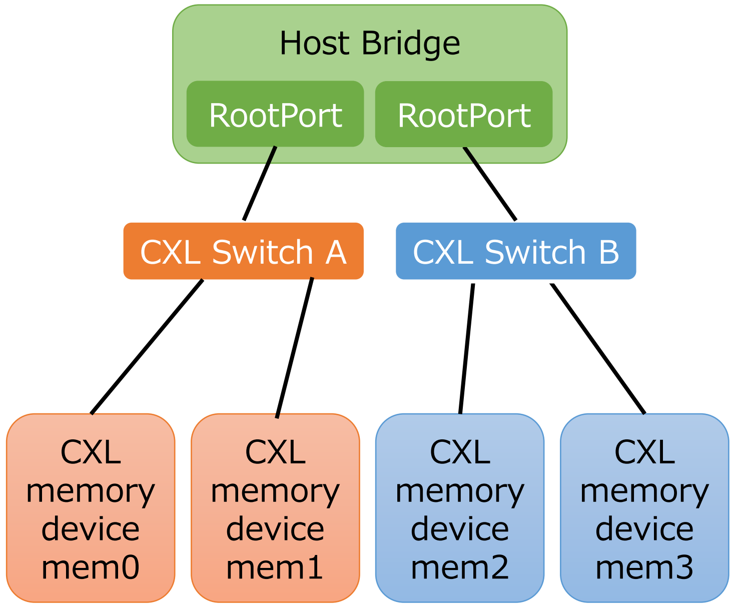
Interleaved region example (1/2)



```
# cxl create-region -d decoder0.0 -m mem0 mem1 mem2 mem3 -t ram
```

- When you want to create interleaved region, you need to specify all of the member of memory devices
  - If all the memory devices are connected under a CXL switch, it is easy
    - You just need to specify all the name of the devices

Interleaved region example (2/2)



OK
```
# cxl create-region -d decoder0.0 -m mem0 mem2 mem1 mem3 -t ram
```

OK
```
# cxl create-region -d decoder0.0 -m mem1 mem3 mem2 mem2 -t ram
```

CXL switch order is alternating

NG
```
# cxl create-region -d decoder0.0 -m mem0 mem1 mem2 mem3 -t ram
```

NG
```
# cxl create-region -d decoder0.0 -m mem3 mem2 mem0 mem1 -t ram
```

CXL switch order is NOT alternating

- However, if you want to create interleave region with multiple switches, it is currently a bit more difficult
- Please note that you must specify the correct order of memory devices
  - If you selected a memory device under one switch as the first specified device, you need to specify a memory device under another CXL switch next
    - In other words, the order of CXL switch must alternate
  - The order of memory device under a switch can be whichever comes first
- To confirm these connection, the following command will be useful
  - It can display the topology of all of the memory device and switches in JSON format

```
# cxl list -M -P -D -T -E -u
```

conclusion

Conclusion

Fsas Technologies

- Introduced how to use QEMU CXL emulator
  - To make it easier for everyone to use, I showed many tips and my recommendations
  - I hope this is good start for you to use it

- There are more features in QEMU CXL emulation, such as hotplug, that I couldn't introduce today
  - Please refer to the official information and other guides too
    - QEMU CXL site: https://www.qemu.org/docs/master/system/devices/cxl.html
    - The man pages of cxl commands
    - Blog posts about CXL: https://stevescargall.com/categories/cxl/
    - Talk with each OSS community

- I really hope that this will increase the number of software engineers interested in CXL
  - Please try the CXL emulation on your machine

Fsas Technologies

Bandwidth of CXL memory emulation

Fsas Technologies

- Sample value of an environment in my team
  - CXL memory emulation (accel=tcg)
    - 9.1MB/s
  - NVDIMM emulation (accel=kvm)
    - 6006MB/s