KVM on System z:
Channel I/O And How To Virtualize It

Cornelia Huck <cornelia.huck@de.ibm.com>

Agenda

- **Quick history**
- **Basic concepts**
- **Initiating I/O**
- **Linux support for channel I/O**
- **Virtualization support**
- **Virtio-ccw**
- **References**

A Quick History of Channel I/O

- **Initial versions in early IBM mainframes (1950s)**
- **Reference implementation with System/360 in 1963 (SIO style)**
- **START SUBCHANNEL style introduced with 370/XA in 1981**
  - Still in use on today's System z hardware
  - Various enhancements to support new features like 64 bit addressing or high performance ficon

Basic Concepts

- **Channel Subsystem**
  - Provides I/O mechanism
  - Processors dedicated to I/O relieve the main processors
- **Channel Subsystem Image**
  - Comprised of subchannels and channel paths
  - Currently up to 4 images per machine; only one image accessible per logical partition

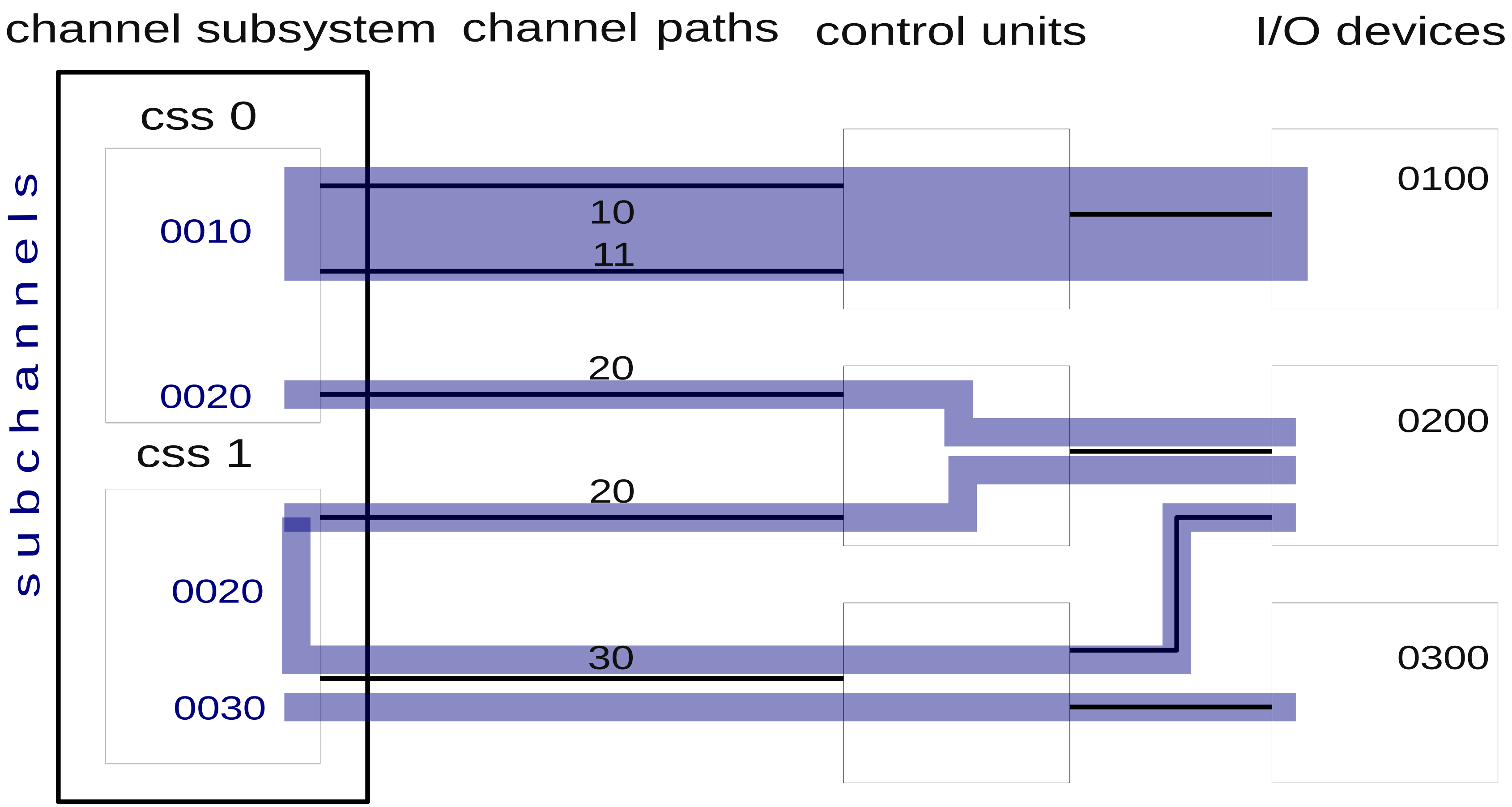
Basic Concepts (2)

- **Subchannel**
  - Logical communication path to and from device
  - Collects status for I/O, connections and device
  - Organized into up to four subchannel sets of up to 64k subchannels (per channel subsystem image)
- **Channel Path**
  - Corresponds to machine ↔ control unit connection
  - Shared between subchannels (up to 8 channel paths per subchannel)
  - Up to 255 channel paths per channel subsystem image

Basic Concepts (3)

- **Control Unit**
  - Accepts a set of channel commands
  - May be integrated with the I/O device
  - Self-descriptive (e.g. SenseID channel command)
  - Responsible for translating between channel commands and device-specific actions

Basic Concepts (4)



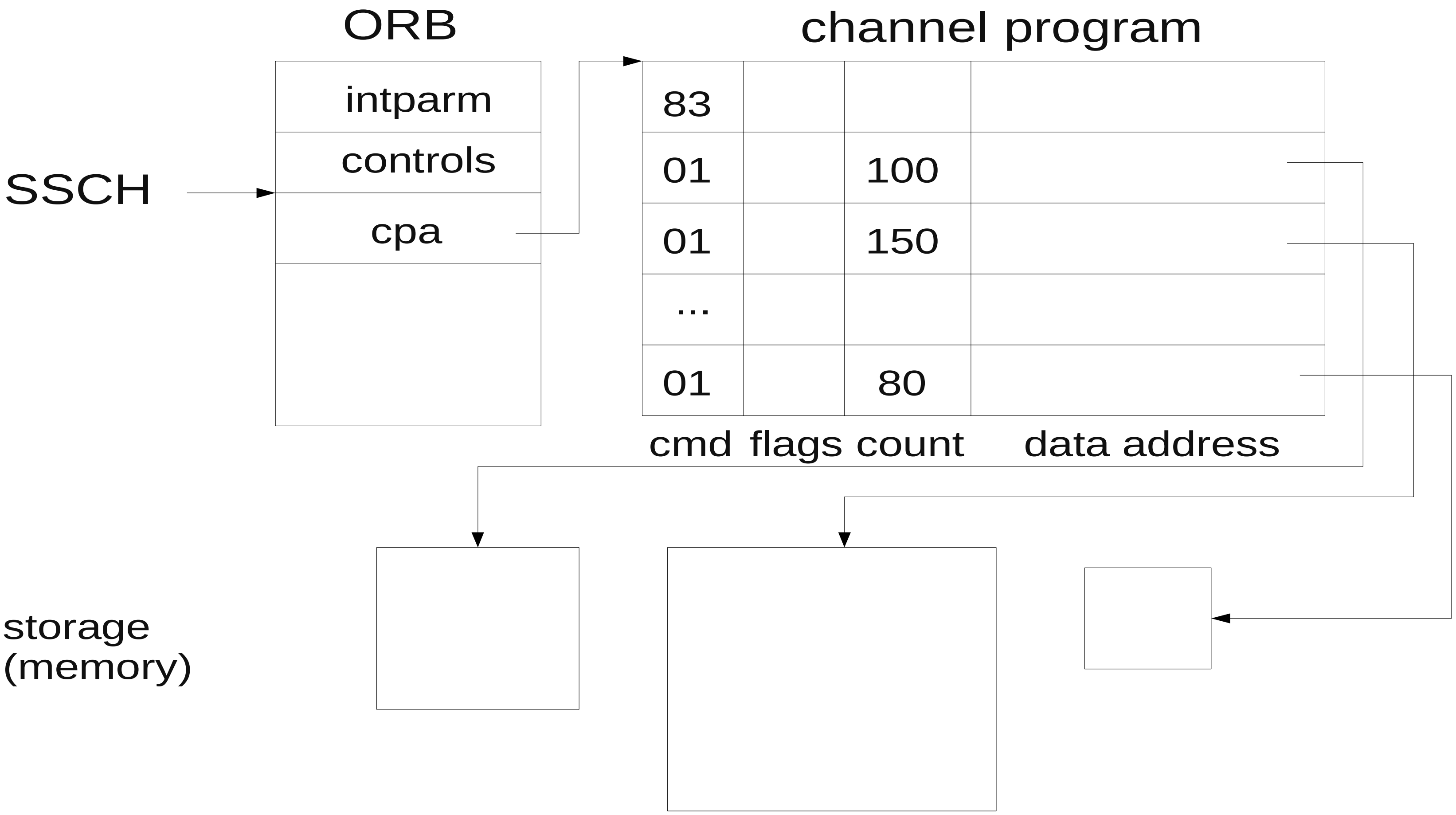
Initiating I/O

- **Start Subchannel (ssch)**
  - Provide a channel program and parameters to the channel subsystem
  - Channel program is performed asynchronously by the channel subsystem
  - Upon conclusion, error or caller's request, the subchannel is made status pending and an I/O interrupt is generated

Initiating I/O (2)

- **Channel programs**
  - Consist of channel command words (ccws)
  - Each ccw refers a specific command (e.g. read, write) and may refer to a memory area
  - Multiple ccws may be chained (e.g. multiple reads) and started by a single ssch
  - Running channel programs may be modified in-flight
  - Special features: TIC (GOTO equivalent), suspend marker, program controlled interrupts

Initiating I/O (3)

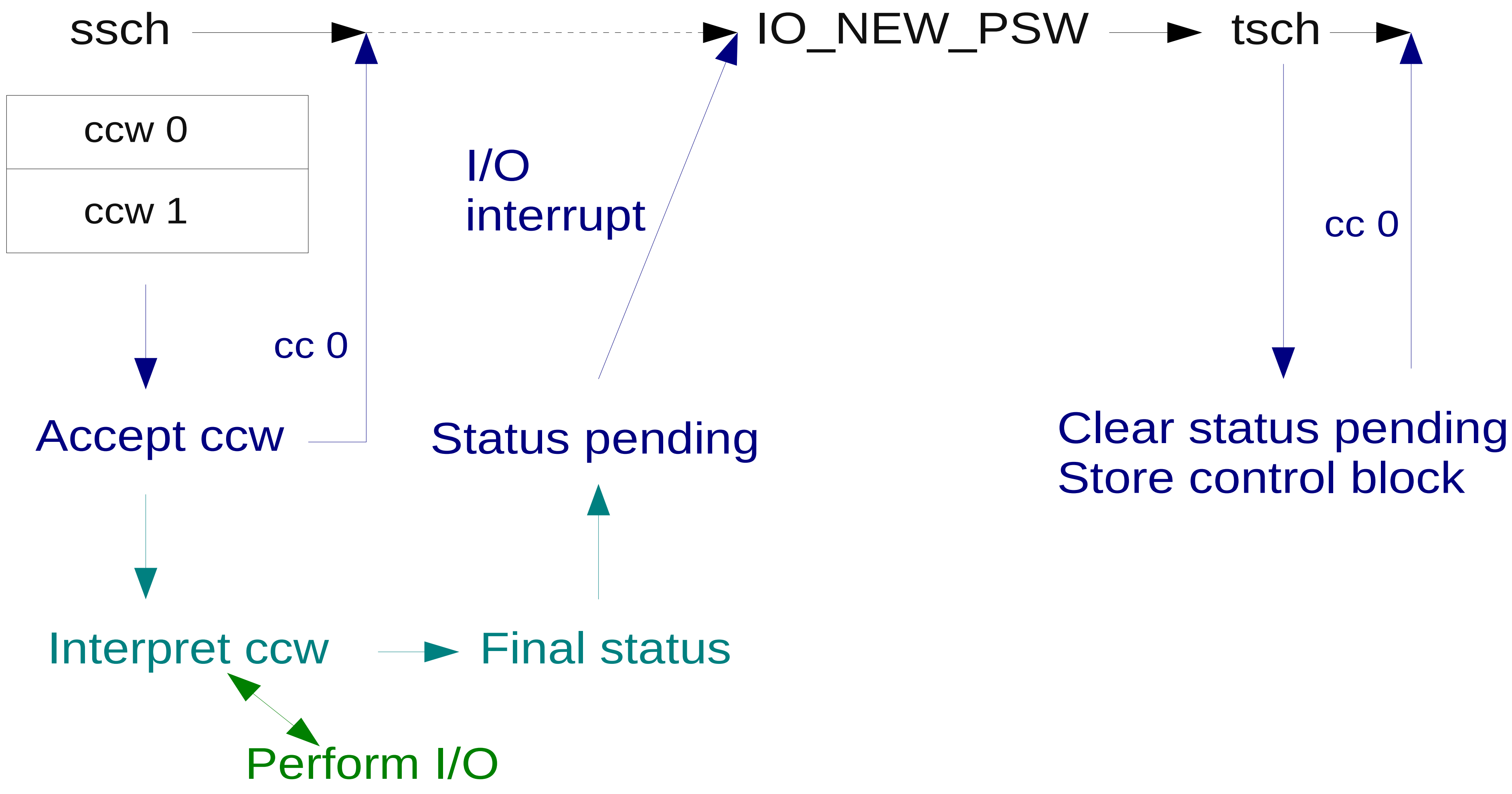


cmd	flags	count	data address
83			
01		100	
01		150	
...			
01		80	

Initiating I/O (4)

- **I/O Interrupts**
  - Floating interrupt – may occur on any CPU
  - Made pending when a subchannel becomes status pending, delivered via PSW swap
  - Carries payload designating the subchannel, written into CPU's lowcore
  - Pending but not delivered I/O interrupts may be removed by I/O instructions (TPI – test pending interruption, TSCH – test subchannel)
  - Usually triggers a TSCH by the program to collect subchannel status

Initiating I/O (5)



Linux Support for Channel I/O

- **Common I/O Layer**
  - Provides wrapper around low-level channel I/O
  - Handles basic channel I/O and I/O interrupts
- **CCW device drivers**
  - Support for various devices and control units
  - Channel commands specific to device types
  - Examples: dasd (disks), channel attached tapes

Linux Support for Channel I/O (2)

- **Example of a guest running under z/VM:**

```
[root@r1760001 ~]# lscss
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.f5f0 0.0.0000  1732/01 1731/01 yes  80  80  ff   76000000 00000000
0.0.f5f1 0.0.0001  1732/01 1731/01 yes  80  80  ff   76000000 00000000
0.0.f5f2 0.0.0002  1732/01 1731/01 yes  80  80  ff   76000000 00000000
0.0.3800 0.0.0003  3390/0c 3990/e9 yes  fc  f0  ff   30313233 3c3d0000
0.0.3801 0.0.0004  3390/0c 3990/e9 yes  fc  f0  ff   30313233 3c3d0000
0.0.3802 0.0.0005  3390/0c 3990/e9 yes  fc  f0  ff   30313233 3c3d0000
0.0.0191 0.0.0006  3390/0c 3990/e9      fc  f0  ff   30313233 3c3d0000
0.0.0009 0.0.0007  0000/00 3215/00 yes  80  80  ff   01000000 00000000
0.0.000c 0.0.000e  0000/00 2540/00      80  80  ff   01000000 00000000
0.0.000d 0.0.000f  0000/00 2540/00      80  80  ff   01000000 00000000
0.0.000e 0.0.0010  0000/00 1403/00      80  80  ff   01000000 00000000
0.0.0190 0.0.0011  3390/0c 3990/e9      fc  f0  ff   30313233 3c3d0000
0.0.019d 0.0.0012  3390/0c 3990/e9      fc  f0  ff   30313233 3c3d0000
0.0.019e 0.0.0013  3390/0c 3990/e9      fc  f0  ff   30313233 3c3d0000
0.0.0592 0.0.0014  3390/0c 3990/e9      fc  f0  ff   30313233 3c3d0000
```

Virtualization Support

- **SIE: Virtualization instruction on s390**
- **I/O instructions get SIE exits**
  - Instruction intercept for most I/O instructions
  - Additionally I/O intercept for SSCH
    - Currently not used by KVM
  - Special intercepts for passthrough of real channel devices

Virtualization Support (2)

- **Handling I/O**
  - Perform path-related operations
  - Interpret channel programs
    - Doing this for arbitrary channel programs is the most complex part!
  - Actually do I/O
    - Either on virtual backend (virtio, …)
    - Or on real (passthrough) I/O device
  - Keep subchannel control blocks up to date

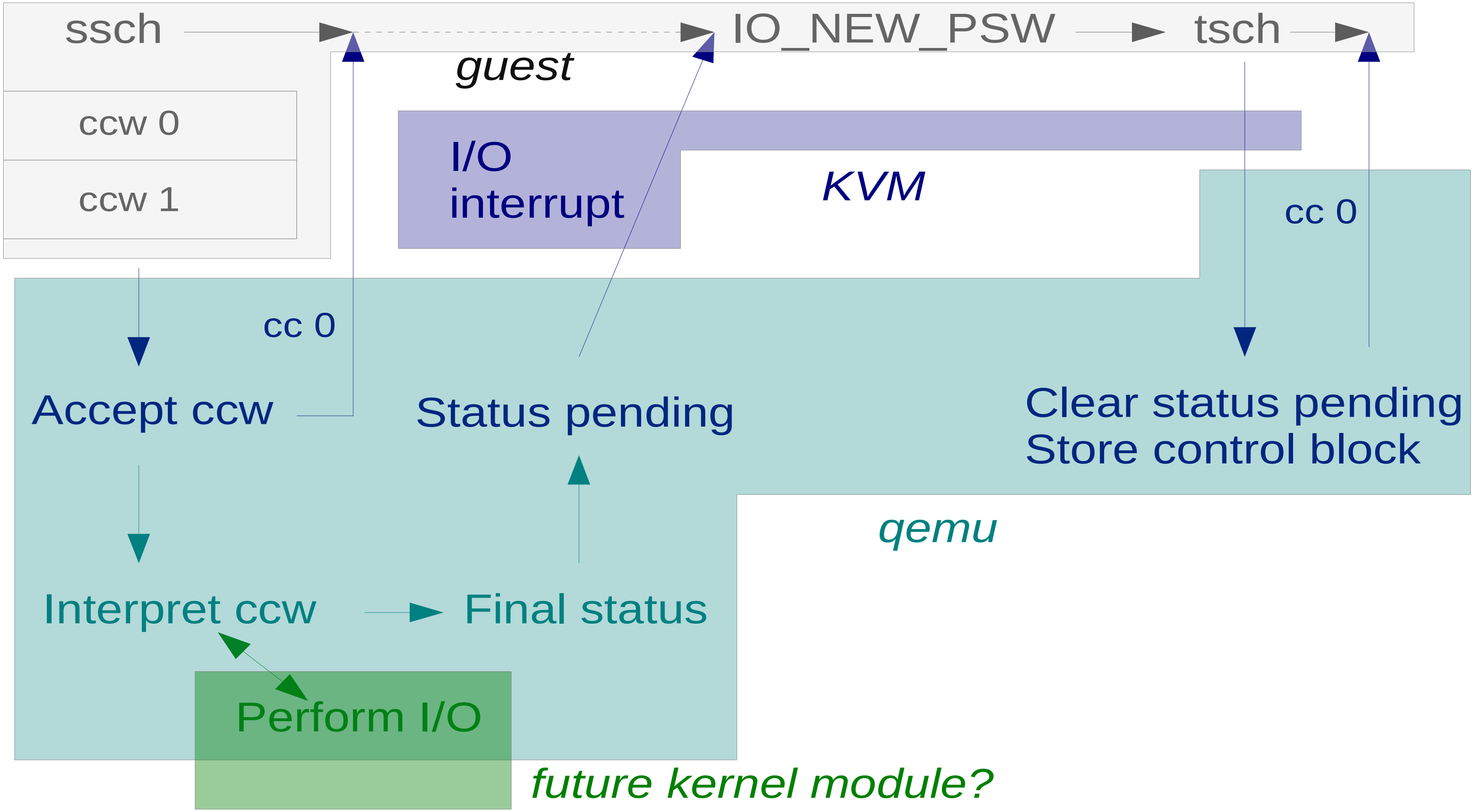
Virtualization Support (3)

- **Interception requests for injecting I/O interrupts**
  - Drop VCPU out of SIE when I/O interrupts enabled
  - Further interception requests for control register 6 (interruption subclasses)
- **I/O interrupts may be cleared by tsch/tpi**
- **Hypervisor needs to keep track of interrupt payload (subchannel ID, interruption parameter)**

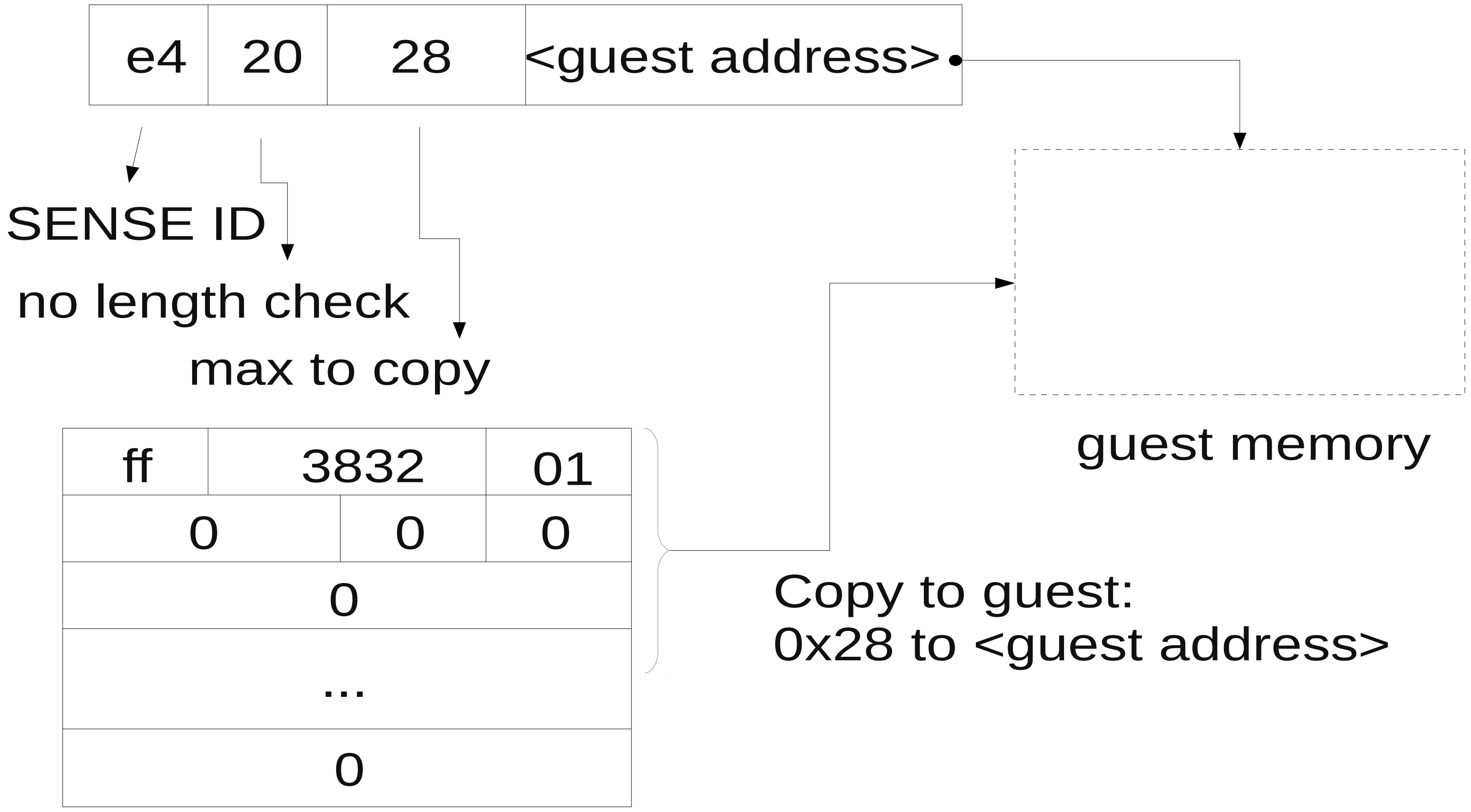
Virtualization Support (4)

- **Current status for KVM and qemu:**
  - Support for I/O interrupts and related I/O instructions (tsch, tpi) in KVM
  - Support for I/O instructions on virtual subchannels in qemu (virtual css)
  - virtio-ccw support in qemu
- **Possible future enhancements**
  - Support advanced I/O functionality (IDALs, …)
  - Support for adapter (thin) interrupts
  - Support for passthrough of real channel I/O devices

Virtualization Support (5)



Virtualization support (6)



e4	20	28	<guest address>

SENSE ID

no length check

max to copy

guest memory

ff	3832		01
0		0	0
0			
...			
0			

SENSE ID data

Copy to guest:
0x28 to <guest address>

Virtio-ccw

- **Virtio transport based upon channel I/O**
- **Fully virtual channel devices used as virtio bridge devices**
  - Virtual channel subsystem image 0xfe
  - Virtual channel path type 0x32 (only to satisfy architecture)
  - Virtual control unit type 0x3832
    - Virtio device type used as control unit model

Virtio-ccw (2)

- **Virtio-related operations implemented via channel commands**
  - Setup virtual queues, get and set features, read and write configuration...
  - Guest → host notification via diagnose (hypercall)
  - Host → guest notification via I/O interrupts and indicator bits
- **Documented in virtio spec**

Virtio-ccw (3)

- **Example of a guest running under qemu with virtio-ccw:**

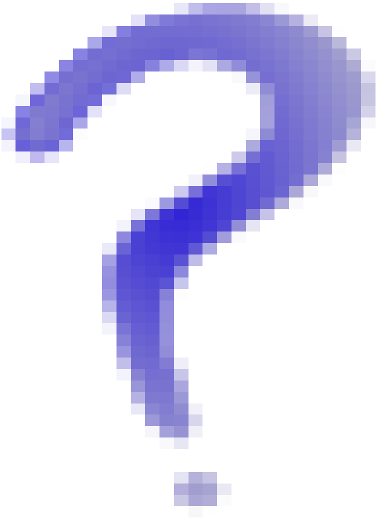
```
[root@localhost ~]# lscss
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.0000 0.0.0000  0000/00 3832/01 yes  80  80  ff   00000000 00000000
0.0.0815 0.0.0001  0000/00 3832/02 yes  80  80  ff   00000000 00000000
0.0.0002 0.0.0002  0000/00 3832/03 yes  80  80  ff   00000000 00000000
0.1.abcd 0.1.0000  0000/00 3832/05 yes  80  80  ff   00000000 00000000

[root@localhost ~]# lschp
CHPID  Vary  Cfg.  Type  Cmg  Shared  PCHID
============================================
0.00   1     -     32    -    0       -
```

References

- **IBM publications**
  - z/Architecture Principles of Operation (SA22-7832), chapter 13 ff.
  - Common I/O-Device Commands and Self-Description (SA22-7204)
- **Virtio spec**
  - See https://github.com/rustyrussell/virtio-spec

Questions?



Legal Statement

This work represents the view of the author and does not necessarily represent the view of IBM.

IBM, IBM(logo), z/Architecture, zSeries, Enterprise Systems Architecture/390, ESA/390, Enterprise Systems Architecture/370, ESA/370 and System/360 are trademarks and/or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.