China Linux Kernel Conference (October 24th, 2020)

Accelerators For Everyone

Tony Luck. Principal Engineer. Intel Architecture, Graphics, & Software Group

intel®

Tony Luck

- Working on Linux at Intel since 2000
- Specializes in server features
  - RAS
  - RDT
  - Accelerators

The Future (according to experts)

“… domain-specific architectures as the only path forward for improved performance and energy efficiency …”

—Hennessy & Patterson “Computer Architecture, A Quantitative Approach”

Agenda

- General accelerator goodness
- DSA specifics
- Linux driver implementation
- User interface
- Conclusion

Existing accelerators are hard to use

- Suitable only for large tasks
  - Overhead to access eliminates benefit for small tasks
- Typically access physical, not virtual memory addresses
  - Only usable from kernel or driver interfaces
- Limited number of instances
  - Difficult to share between unrelated users

Solution(1): Reducing the overhead - descriptor

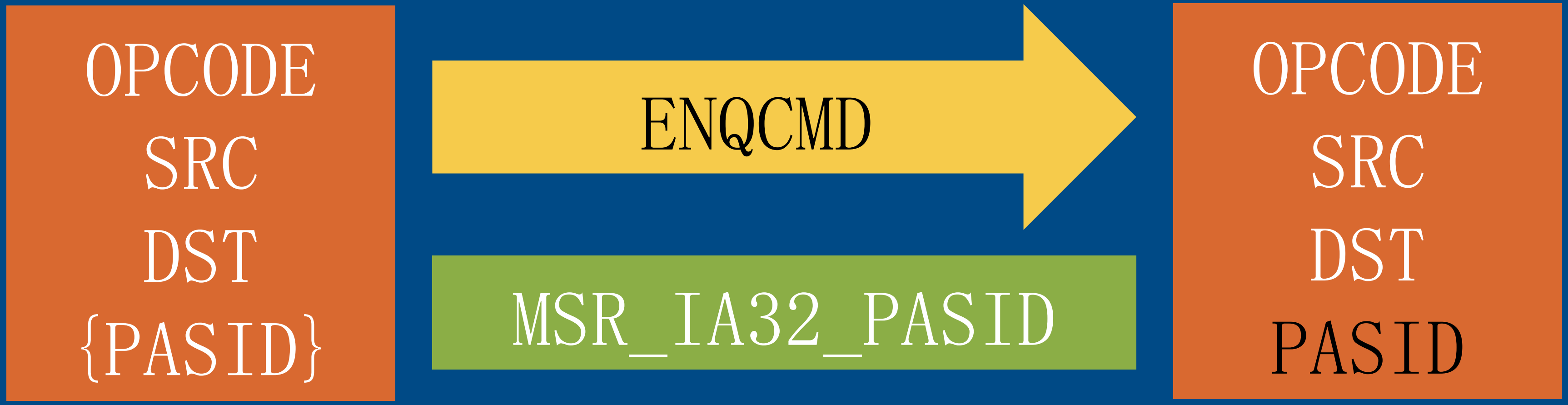
- User composes a “descriptor” in cacheable memory that contains all the information needed for a specific operation
  - “Opcode” = which operation to perform
  - Source and destination addresses for the work
  - Byte count(s) for the size of operands
  - Flags – may modify how the operation is performed and how completion of the operation is indicated
  - Completion record address

Solution(1): Reducing the overhead - submission

- New instructions to add a descriptor to a device work queue using an MMIO “portal” address
  - MOVDIR64B
  - ENQCMD
  - ENQCMDS

Solution(2): Shared virtual memory

- Each descriptor includes virtual addresses of source and destination operands
- Each request is associated with a PASID (Process Address Space IDentifier) that is set up by the operating system and used by the device to request the IOMMU translate virtual addresses to physical



OPCODE
SRC
DST
{PASID}

ENQCMD

MSR_IA32_PASID

OPCODE
SRC
DST
PASID

Solution(3): Sharing between unrelated users

- Devices have a finite amount of storage for queued requests
- In a traditional device all the users need to keep track of how many requests are in the queue
  - Needs locking, or atomic operations on shared variable
- The ENQCMD instruction avoids this by returning status:
  - RFLAGS.ZF=0 Success. The request was queued
  - RFLAGS.ZF=1 Retry/Failure

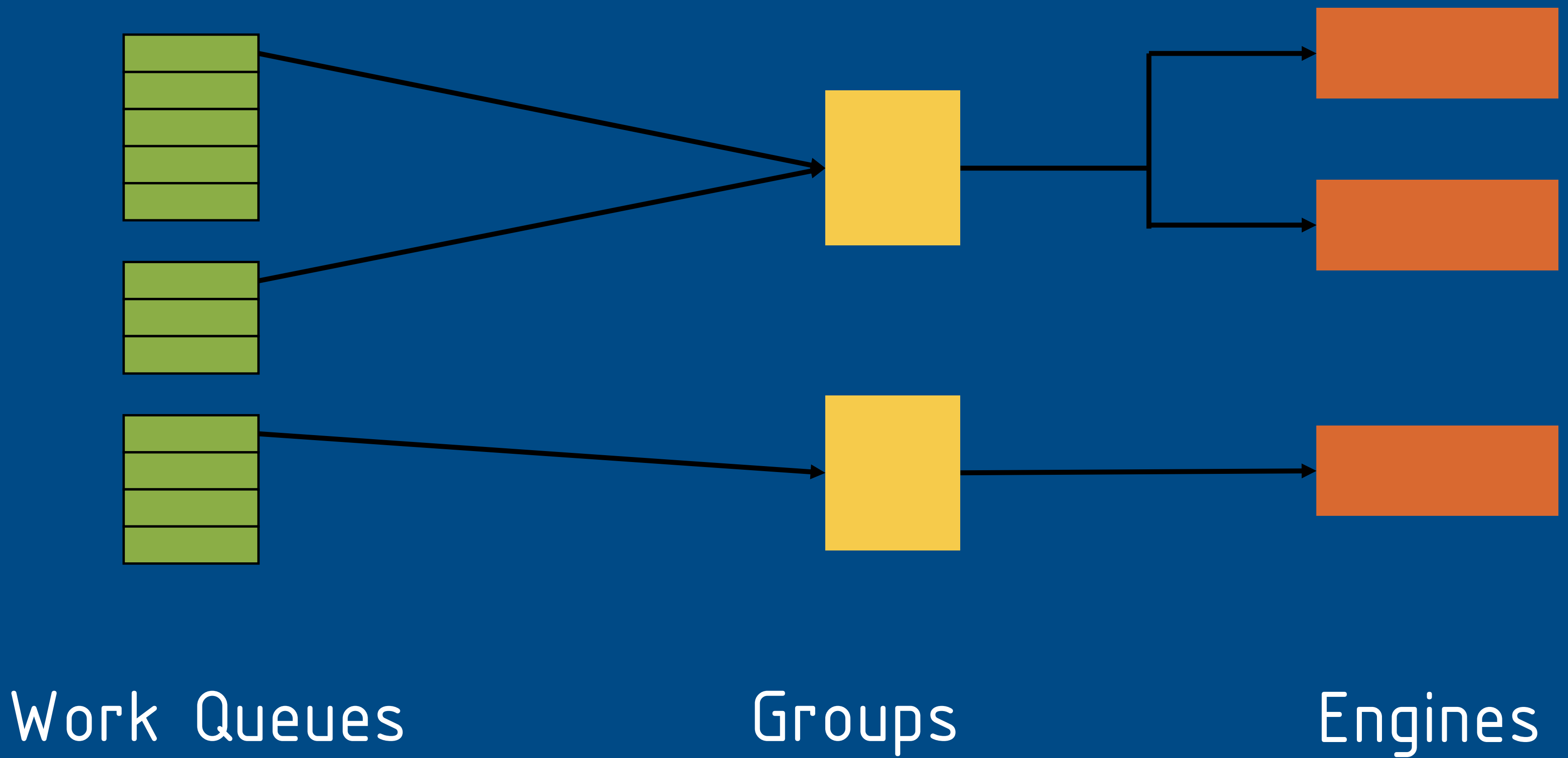
Intel®Data Streaming Accelerator

A high-performance data copy and transformation accelerator

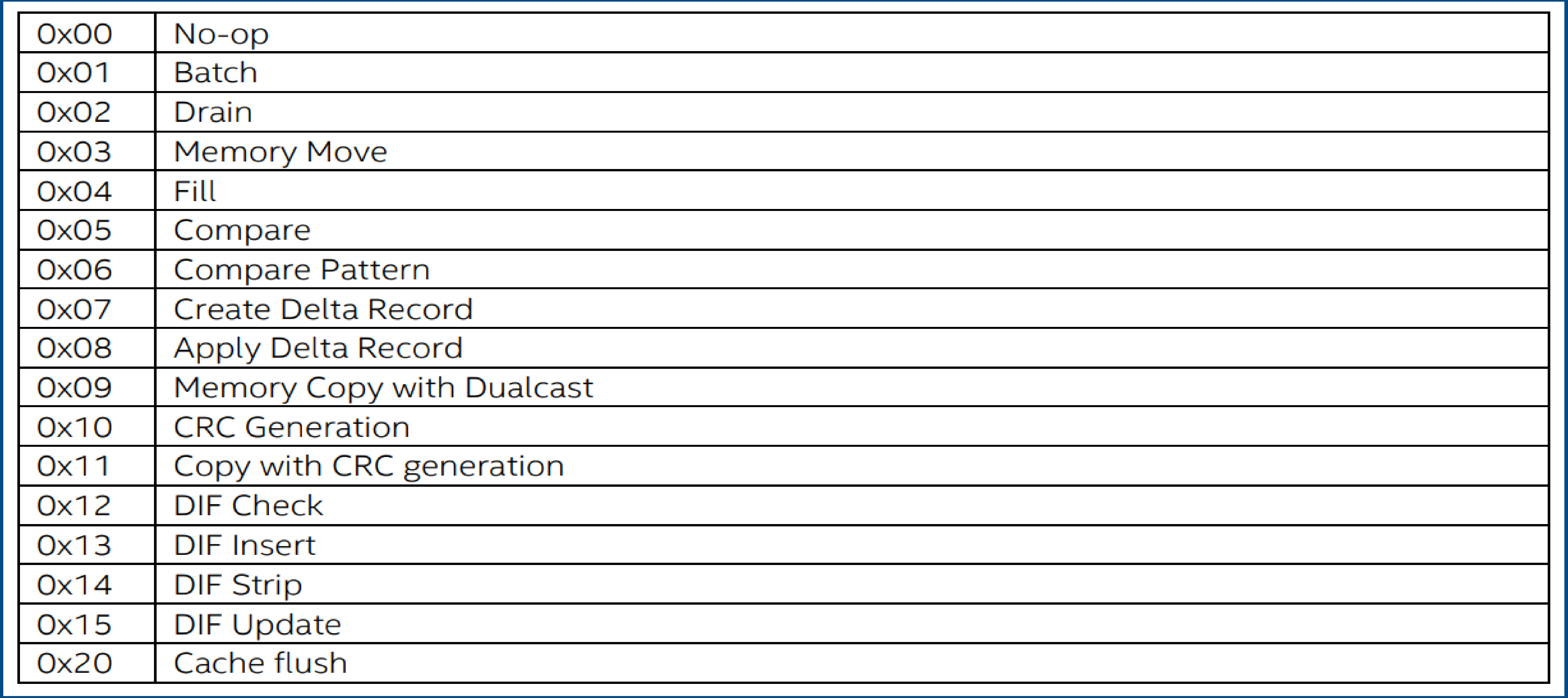
DSA – Use cases

- Data center CPU cores spend a lot of cycles on trivial “copy” and “clear” page operations
  - OS must clear pages before re-using them
  - VMM must clear pages before assigning to a guest
  - OS migrates pages between NUMA nodes when scheduler rebalances load
  - Page de-duplication
  - “I/O” to persistent memory (e.g. 3D-Xpoint™)
  - Network solutions like DPDK. Storage solutions like SPDK

DSA Configuration (high level view)



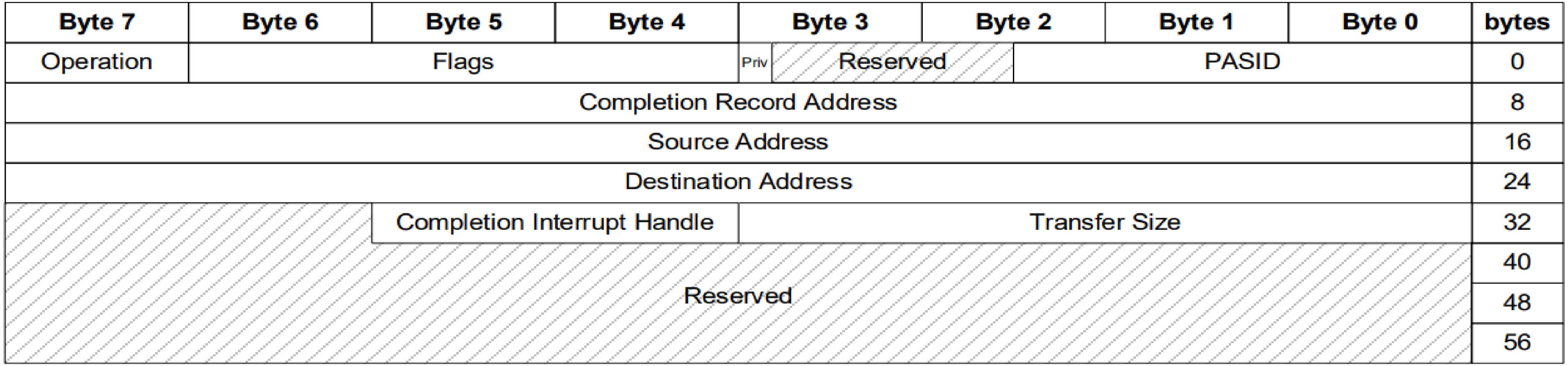
DSA Operation Types



0x00	No-op
0x01	Batch
0x02	Drain
0x03	Memory Move
0x04	Fill
0x05	Compare
0x06	Compare Pattern
0x07	Create Delta Record
0x08	Apply Delta Record
0x09	Memory Copy with Dualcast
0x10	CRC Generation
0x11	Copy with CRC generation
0x12	DIF Check
0x13	DIF Insert
0x14	DIF Strip
0x15	DIF Update
0x20	Cache flush

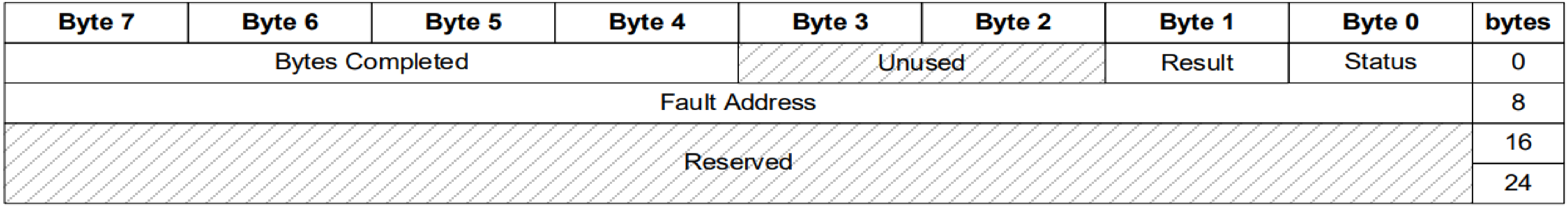
DSA Example Descriptor and Completion

Memory Move Descriptor



<table>
<tr><th colspan="2">Byte 7</th><th colspan="2">Byte 6</th><th colspan="2">Byte 5</th><th colspan="2">Byte 4</th><th colspan="2">Byte 3</th><th colspan="2">Byte 2</th><th colspan="2">Byte 1</th><th colspan="2">Byte 0</th><th>bytes</th></tr>
<tr><td colspan="2">Operation</td><td colspan="6">Flags</td><td>Priv</td><td colspan="2">Reserved</td><td colspan="5">PASID</td><td>0</td></tr>
<tr><td colspan="16">Completion Record Address</td><td>8</td></tr>
<tr><td colspan="16">Source Address</td><td>16</td></tr>
<tr><td colspan="16">Destination Address</td><td>24</td></tr>
<tr><td colspan="4"></td><td colspan="4">Completion Interrupt Handle</td><td colspan="8">Transfer Size</td><td>32</td></tr>
<tr><td colspan="16" rowspan="3">Reserved</td><td>40</td></tr>
<tr><td>48</td></tr>
<tr><td>56</td></tr>
</table>

Memory Move Completion Record



<table>
<tr><th>Byte 7</th><th>Byte 6</th><th>Byte 5</th><th>Byte 4</th><th>Byte 3</th><th>Byte 2</th><th>Byte 1</th><th>Byte 0</th><th>bytes</th></tr>
<tr><td colspan="4">Bytes Completed</td><td colspan="2">Unused</td><td>Result</td><td>Status</td><td>0</td></tr>
<tr><td colspan="8">Fault Address</td><td>8</td></tr>
<tr><td colspan="8" rowspan="2">Reserved</td><td>16</td></tr>
<tr><td>24</td></tr>
</table>

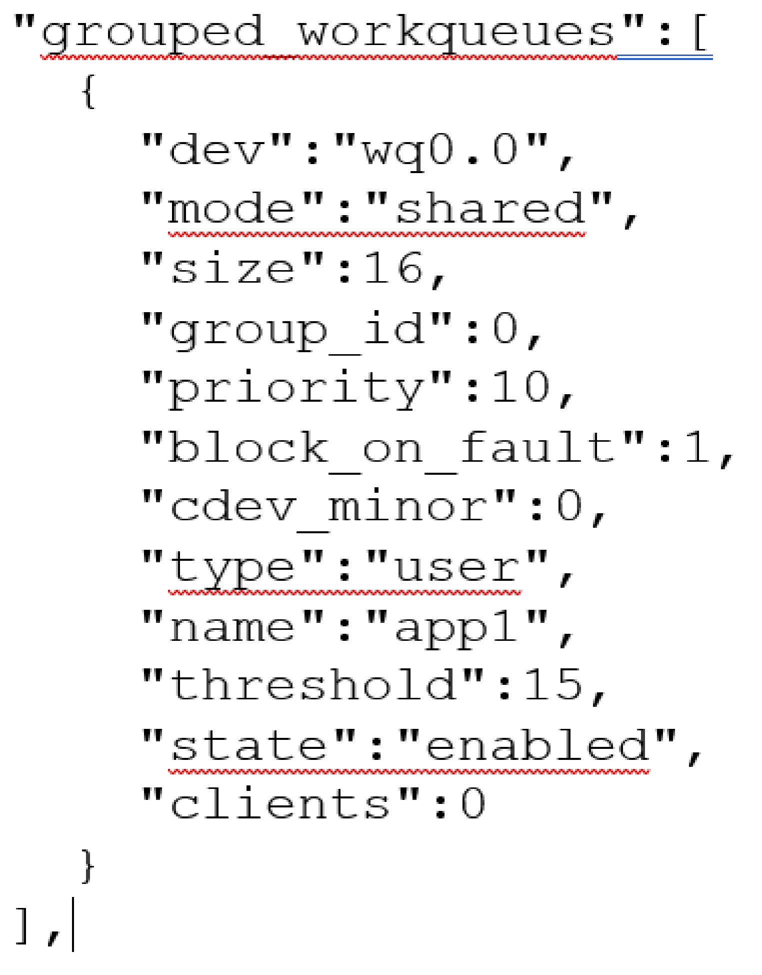
Linux driver implementation

Configuration tool

- There will be multiple DSA devices in a system
  - Up to 8 work queues per device
- Previous picture didn't include all the details
- Some use cases might involve run-time reconfiguration

- Solution: Driver provides interface in /sys to configure
- User space tool: accel-config
  - https://github.com/intel/idxd-config

Example accel-config “json” output



```
"grouped_workqueues":[
  {
    "dev":"wq0.0",
    "mode":"shared",
    "size":16,
    "group_id":0,
    "priority":10,
    "block_on_fault":1,
    "cdev_minor":0,
    "type":"user",
    "name":"app1",
    "threshold":15,
    "state":"enabled",
    "clients":0
  }
],
```

Driver status

- Basic driver with minimal functionality in v5.6
- Kernel support for PASID in v5.10
- Support for shared work queues and shared virtual memory v5.11
  - Merge through dmaengine tree
  - git://git.kernel.org/pub/scm/linux/kernel/git/vkoul/dmaengine.git
- In progress – virtualization support

- Code is in drivers/dma/idxd/*

User interface

In-kernel users

- Copy operations may use existing dma kernel interfaces
- Or can request direct access to work queues from the driver
  - iadx_request_available_wqs()

Applications

- Currently driver provides character device interface for each work queue
- Applications open the device
- Use mmap(2) to get access to the portal to submit work
- Evaluating the “uacce” Linux interface to see if it can be extended

Conclusion

- Accelerator devices are coming soon
- Low latency programming model
- Accessible from:
  - Bare metal OS
  - Applications
  - Guest OS running on VMM