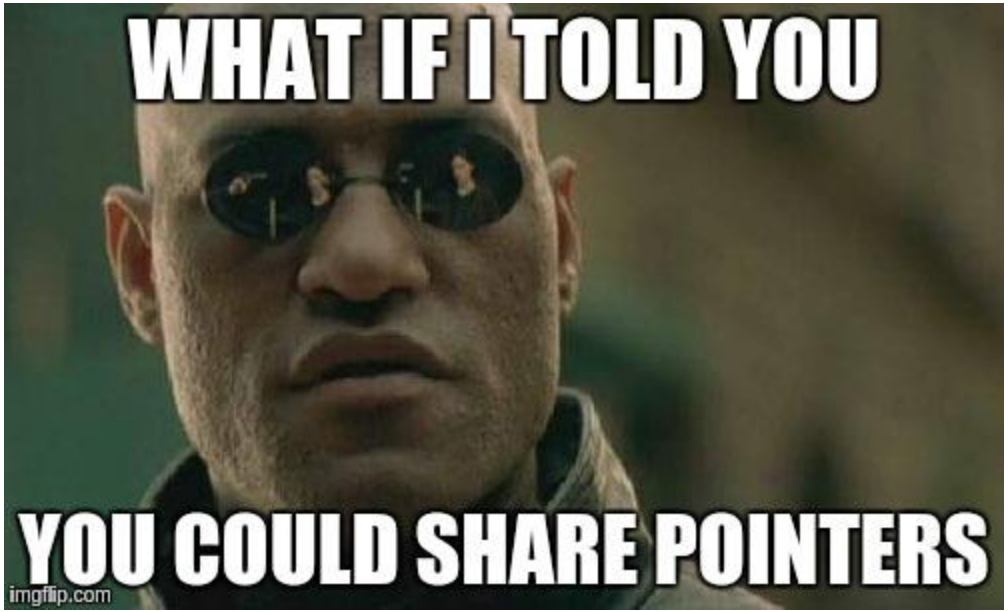
SVM on Intel Graphics

Jesse Barnes
Intel Open Source Technology Center

- What is SVM?
- Discussion of current practices
- SVM OS and driver modifications
- Device options and implications

SVM defined



Pointer sharing between CPU and GPU

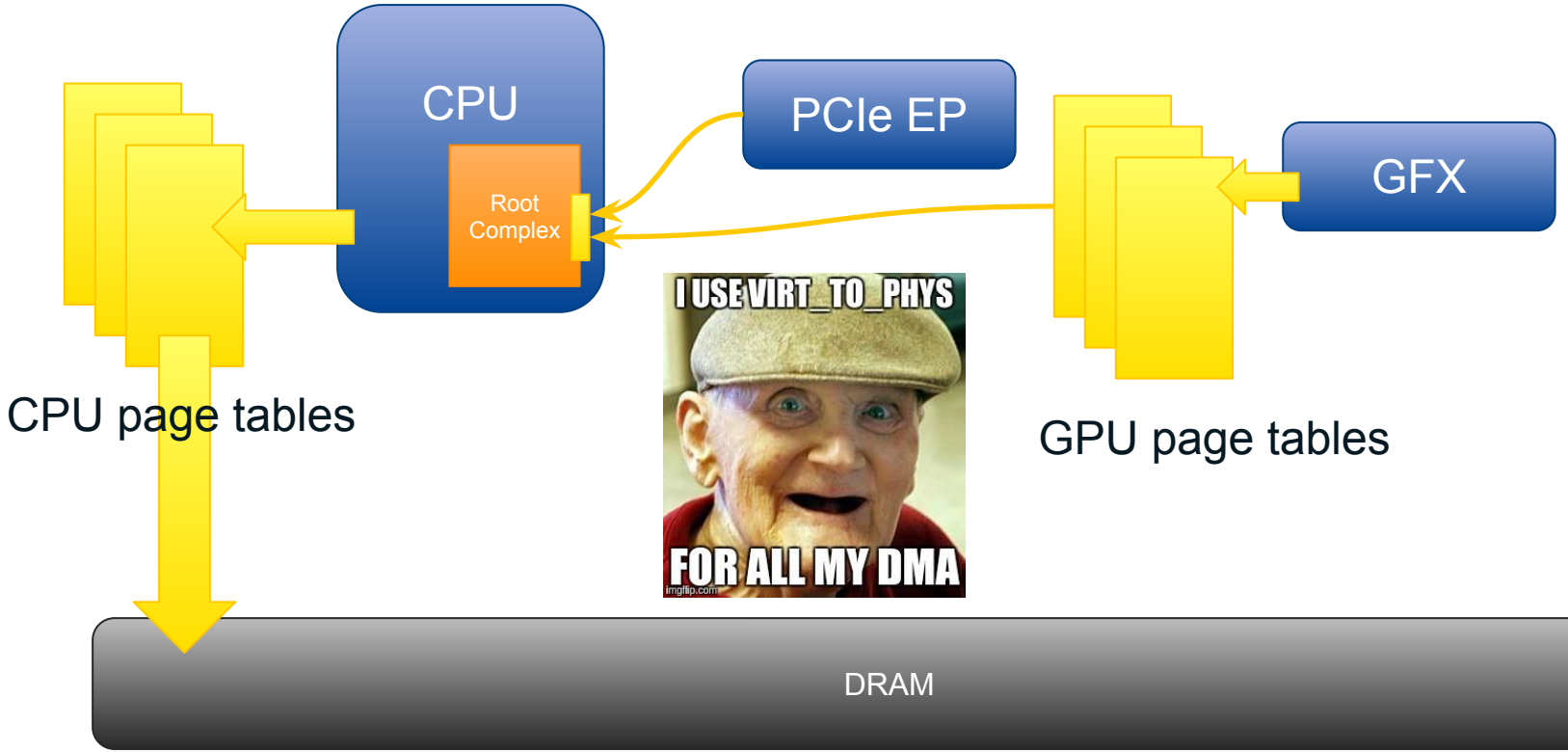
But wait, there's more!

- **Pointer sharing with buffers**
  - Offset in device address space matches offset in process address space
  - Can use a buffer allocation API to manage device page tables
  - Allows OpenCL "fine grained, buffered" model
- **Pointer sharing with a bufferless API**
  - Requires pinning or page fault support
  - Allows OpenCL "fine grained, bufferless" model
  - Requires core OS and driver support
  - Ideal for application programmers
- **Important to be clear when discussing "SVM"**

SVM PCIe and VT-d extensions

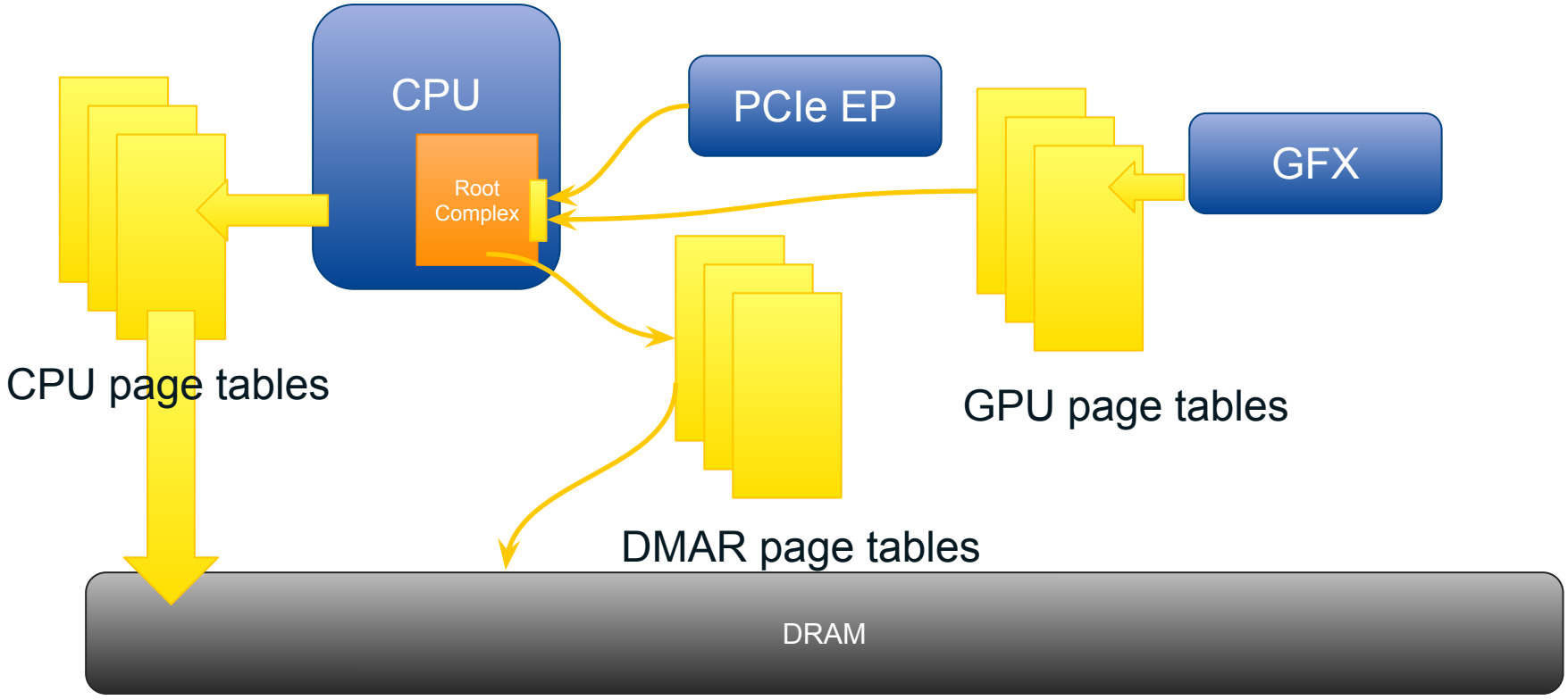
- **ATS – Address Translation Services**
  - Basic IOMMU support
- **PASID – Process Address Space ID**
  - Tells IOMMU which page tables to use, equivalent to the ASID on the CPU side
- **PRI – Page Request Interface**
  - Allows functions to raise page faults to the IOMMU
- **VT-d SVM**
  - Extends root complex IOMMU to comprehend x86 page table formats

Good, old days



Generally devices used physical addrs

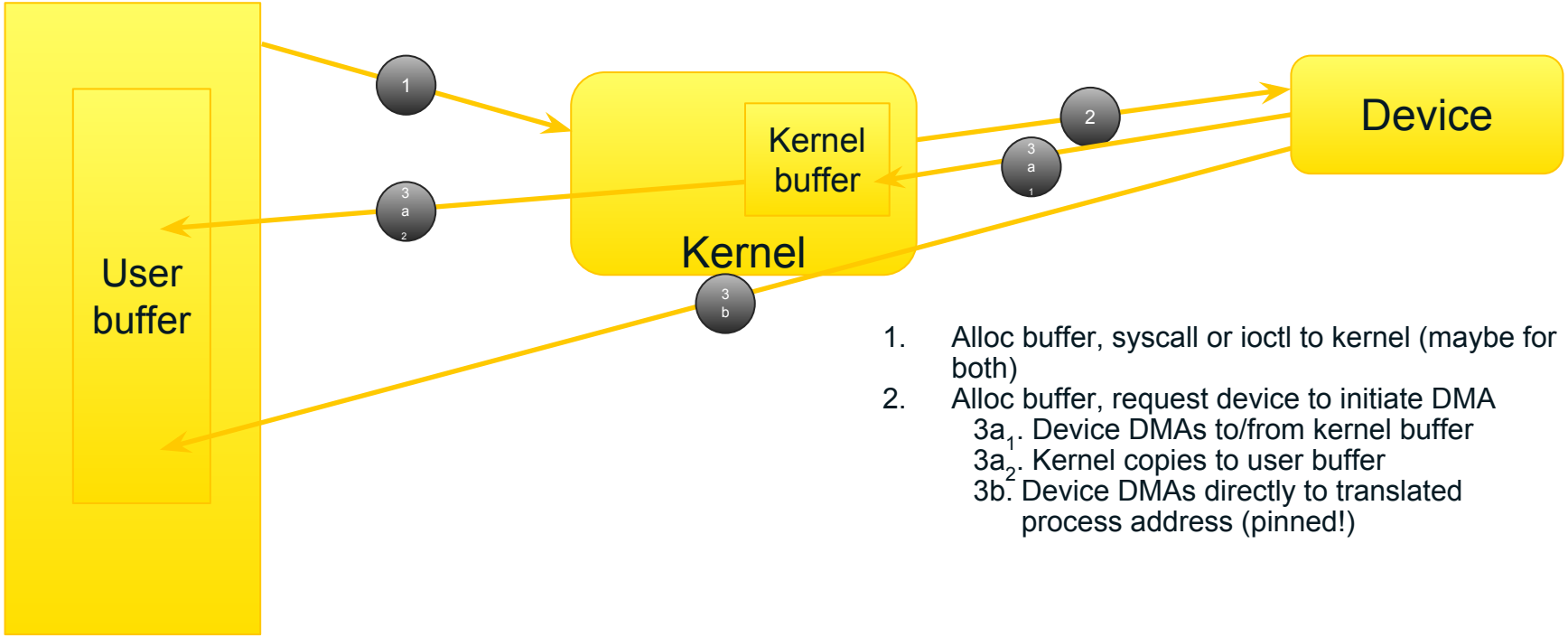
VT-d (and big servers) make it more complicated...



DMA address is indirect

Current interfaces (sans softpin)

- **Buffer alloc**
- **Buffer map – allow direct CPU access to buffer**
- **Buffer read/write – just like read/write on I/O buffers**
- **Buffer share – create handle for inter-process sharing**
- **Buffer query – check status of buffer**
- **Exec buffer – execute code, pass in whole buffer list**
  - Synchronizes with all of the above

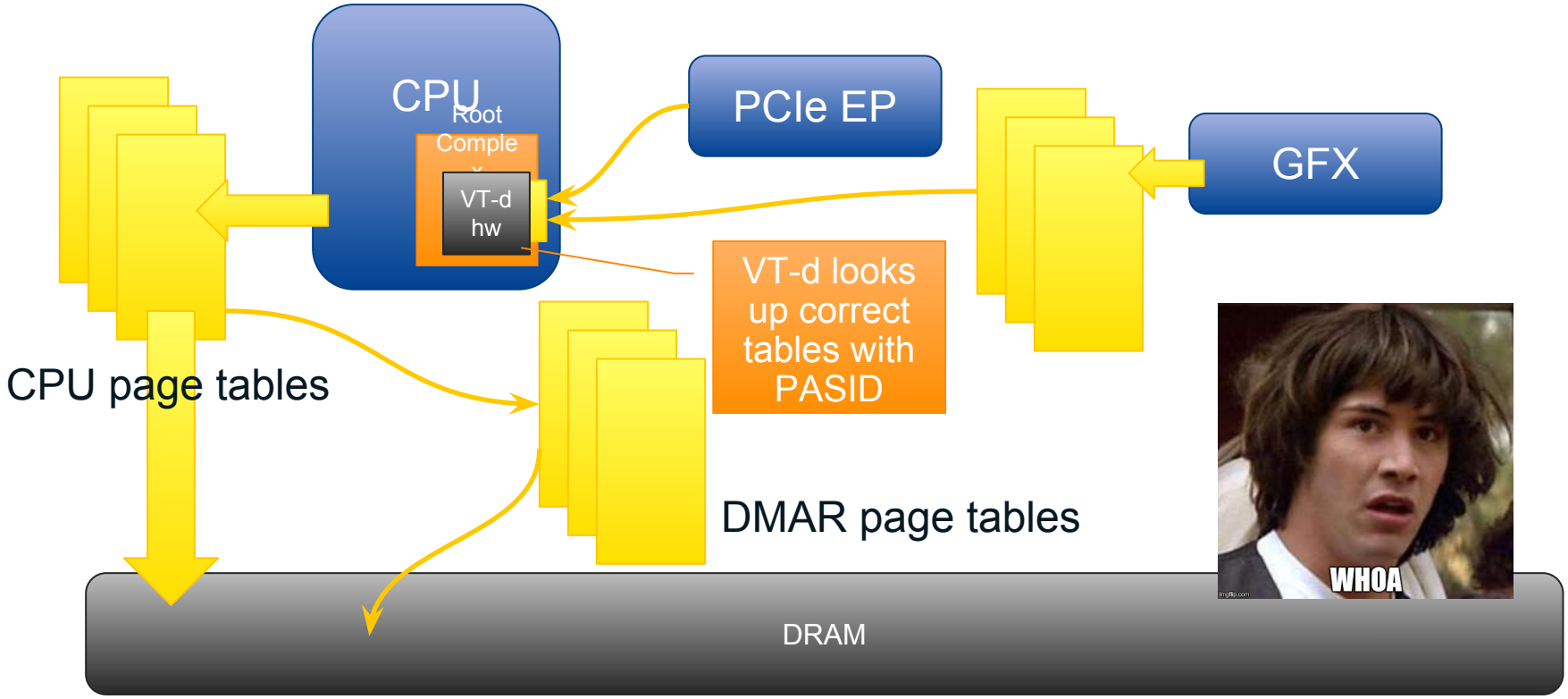


1. Alloc buffer, syscall or ioctl to kernel (maybe for both)
2. Alloc buffer, request device to initiate DMA
   - $3a_1$. Device DMAs to/from kernel buffer
   - $3a_2$. Kernel copies to user buffer
   - 3b. Device DMAs directly to translated process address (pinned!)

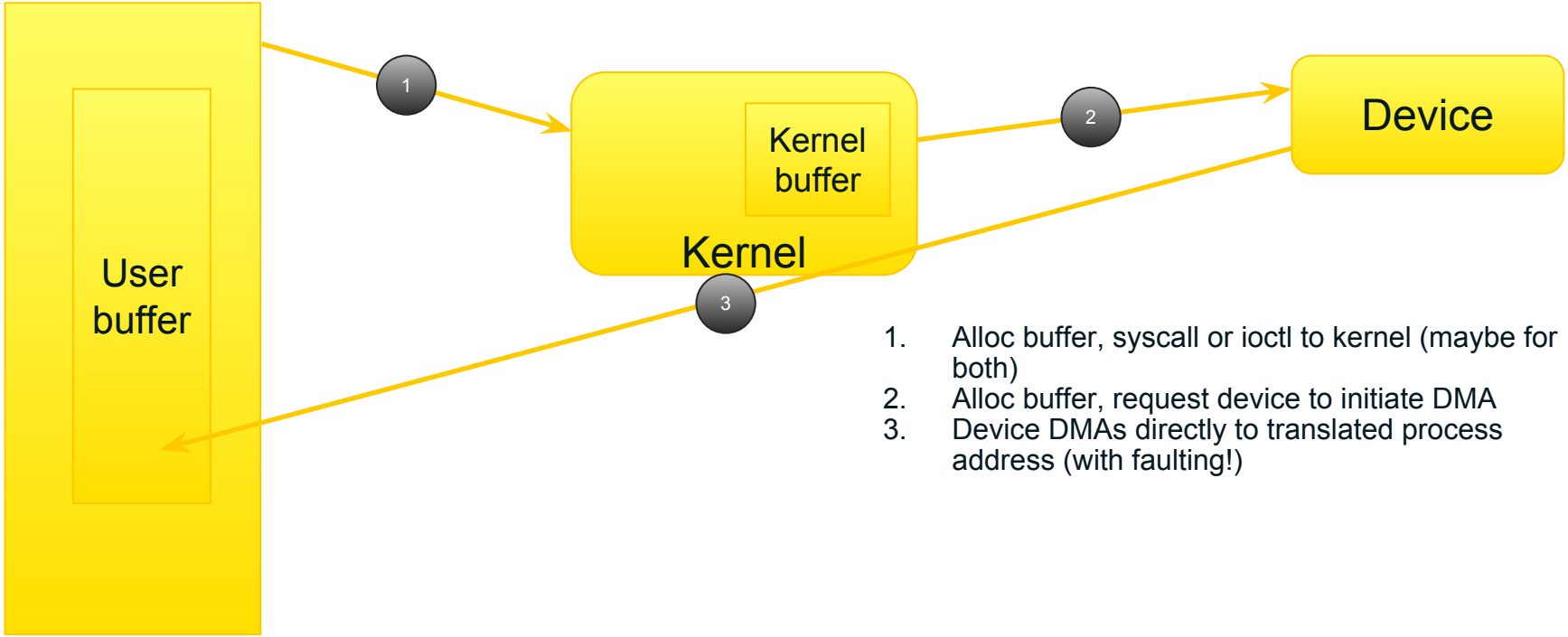
Process address space

SVM changes

Possible SVM model



Potential to share page tables



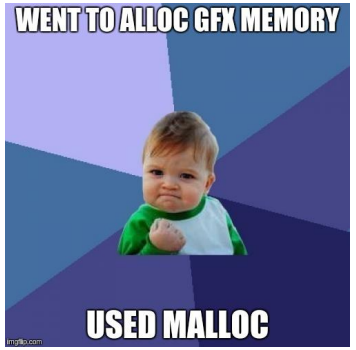
1. Alloc buffer, syscall or ioctl to kernel (maybe for both)
2. Alloc buffer, request device to initiate DMA
3. Device DMAs directly to translated process address (with faulting!)

Process address space

Driver implications

- **Must alloc/track PASID**
  - Either linked to process or device specific context struct
- **Optionally design new APIs**
  - Potentially just “execute starting at this address” or “write to this address”
- **Device<->CPU synchronization is flexible**
  - PCIe atomic ops
  - Memory polling
  - Interrupts passed from device to process through driver specific mechanism

Possible SVM driver interfaces



- **Malloc, mmap, etc – normal libc interfaces for memory management**
- **Context create ioctl takes a flag to indicate you want an SVM context**
  - Can mix & match SVM and non-SVM execution
- **Single interface for submission: i915_exec_mm ioctl**
  - struct drm_i915_exec_mm { batch_ptr; ctx_id; ring, flags; fence; deps; }
- **Synchronization through interrupt forwarding**
  - Command buffer contains interrupt command, driver maps that back

SVM for devices

SVM device options

- **Adding PASID support**
  - Can get you a shared address space on supported platforms
  - Application to device interaction still potentially complex – need to manage pinning, potentially include buffer alloc APIs
- **Adding page faults**
  - Allows for bufferless APIs – simple malloc and use of pointers across CPU/device boundaries
  - Major and minor faults can be handled
  - What to do while servicing the fault?

Context handling

- **Wait for fault handling**
  - Simple, but potentially poor device utilization, depending on use model
- **Restart/abort on fault**
  - Simply re-submit work after fault is handled, starting from the top
  - Also simple to implement, but potentially even worse utilization than waiting
- **Context switch on fault**
  - Save device context on fault, switch to new context like on CPU
  - Potentially very complex for device designers
  - Added complexity for drivers


Q & A