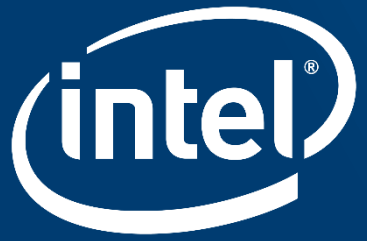


# IDEAL/OPTIMAL MEMORY MANAGEMENT FOR FUTURE VMS

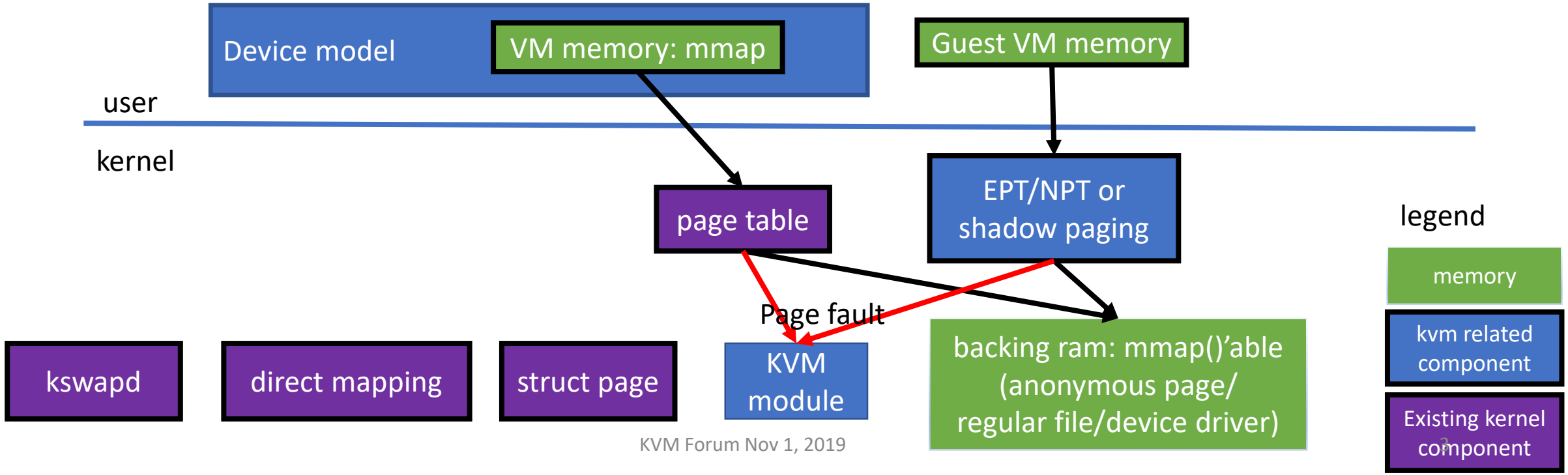
Isaku Yamahata, Intel

# Legal Disclaimer

- Intel provides these materials as-is, with no express or implied warranties.
- All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at http://intel.com.
- Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.
- Intel and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
- © Intel Corporation

# The existing guest VM memory management

- KVM re-use the existing Linux infrastructure to integrate well with Linux memory management
- They are built around page fault on files as memory
- support any generation of hardware(CPU with/without EPT/NPT)



# Emerging new class of memory and use case

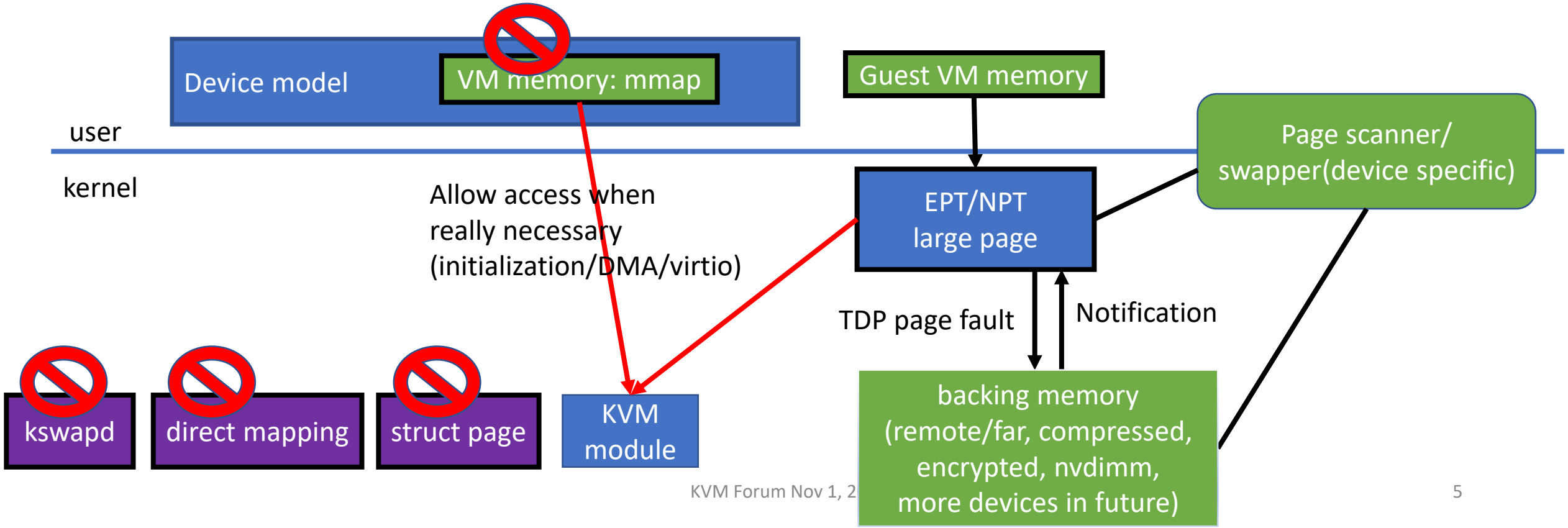
- Large page is suitable for performance/management
- Far/remote memory(compression, software-defined, nvdimm)
  - Software defined far memory: https://dl.acm.org/citation.cfm?id=3304053
- Encrypted memory(MKTME, SGX, SME, SEV)
- Isolation from host kernel/user/VMM/guest
  - Session: *Enhancing KVM for Guest Protection and Security* - Jun Nakajima
  - Friday, November 1 • 16:15 - 16:45
- fast reboot

Things have changed a lot. It's time to revise it

# requirements for new class of memory

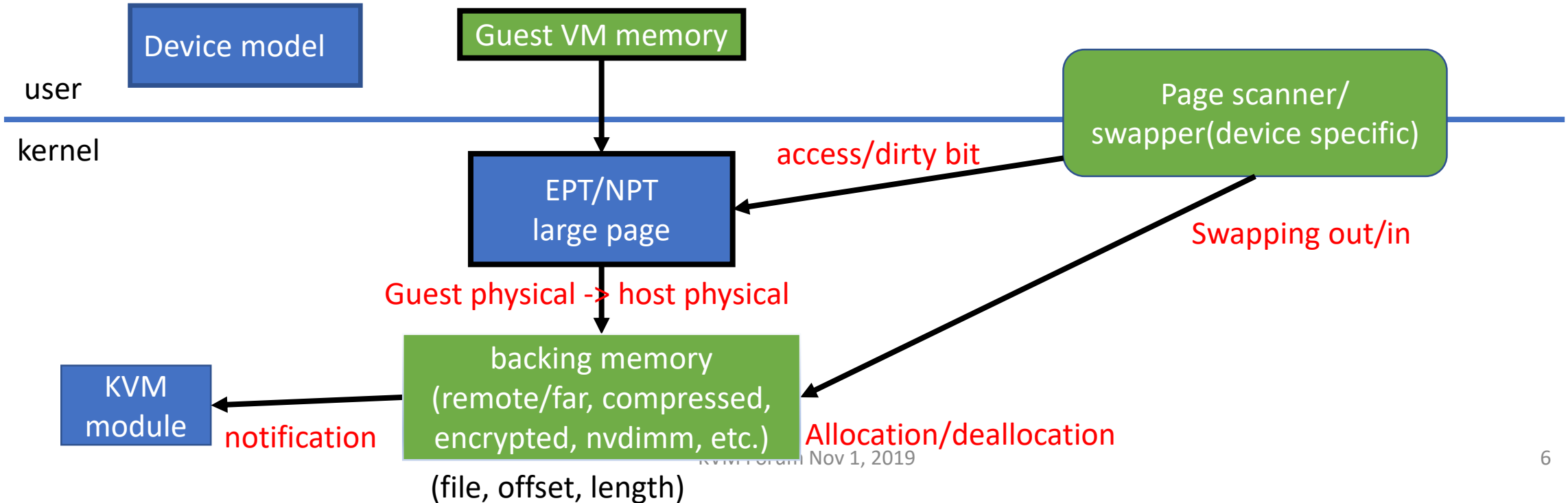
- No mmap/kernel direct map/struct page
- No kswapd. Allow page-reclaim specific to backend. (kernel or user)
- Only EPT/NPT. Taking advantage of large page

=> Decouple guest VM mgmt from host Linux memory mgmt

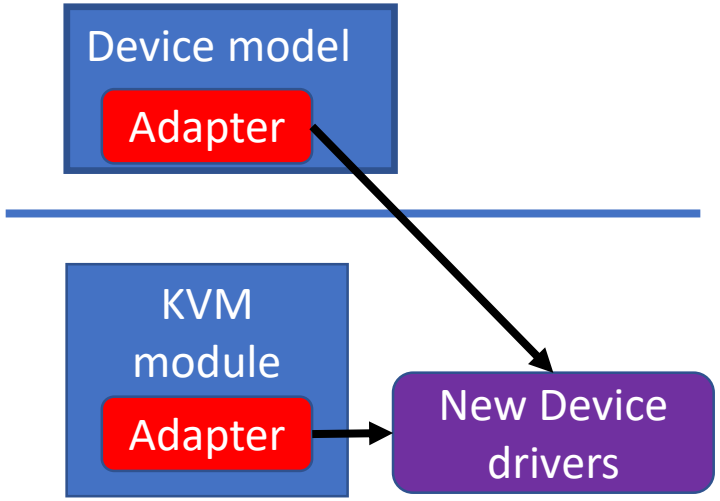


# What operations are needed?

- allocation/deallocation
- Guest physical address -> host physical address conversion
- Access/dirty bit logging
- Swapping out/in

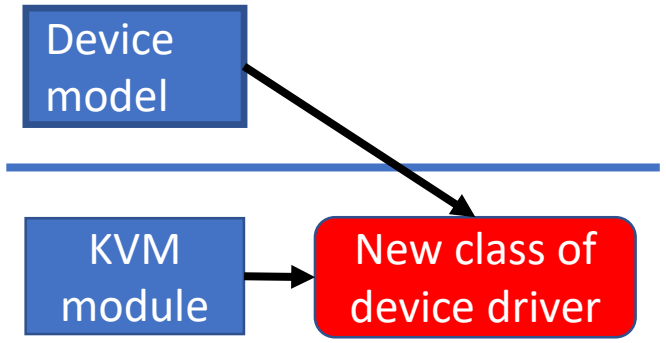


# Options for implementation

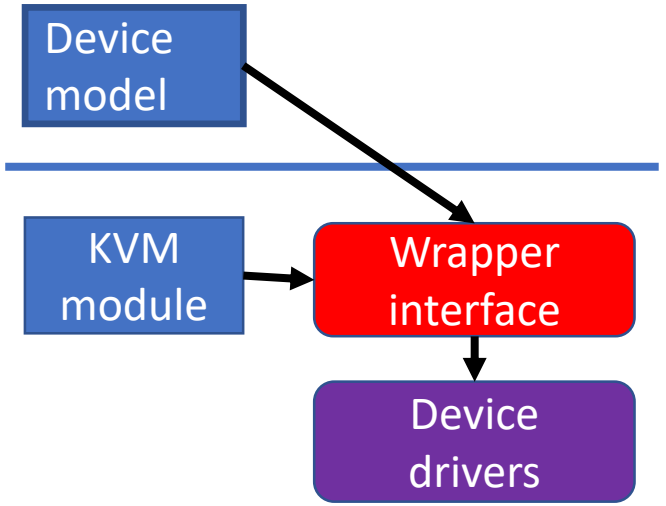



Adds logic to kvm/device model for each backing device drivers

Happening now




Define a class of device driver and port device driver to it

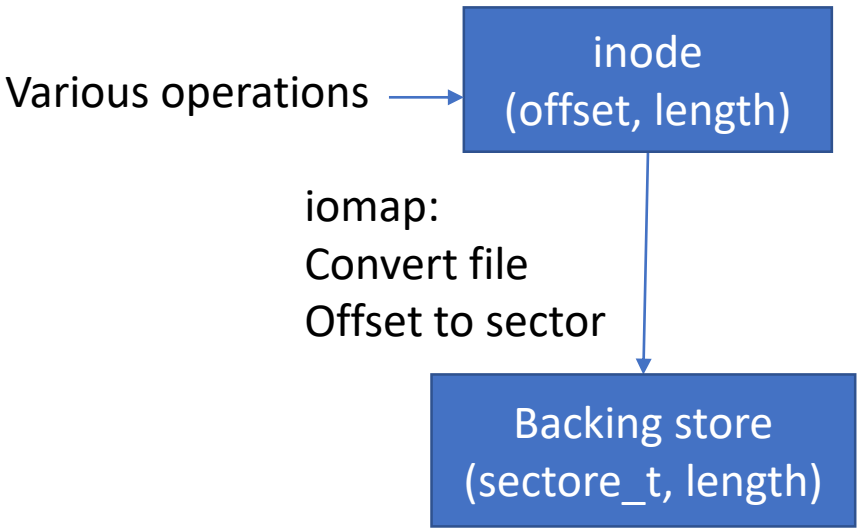



Define a wrapper class as unified interface and write adapter for each device drivers

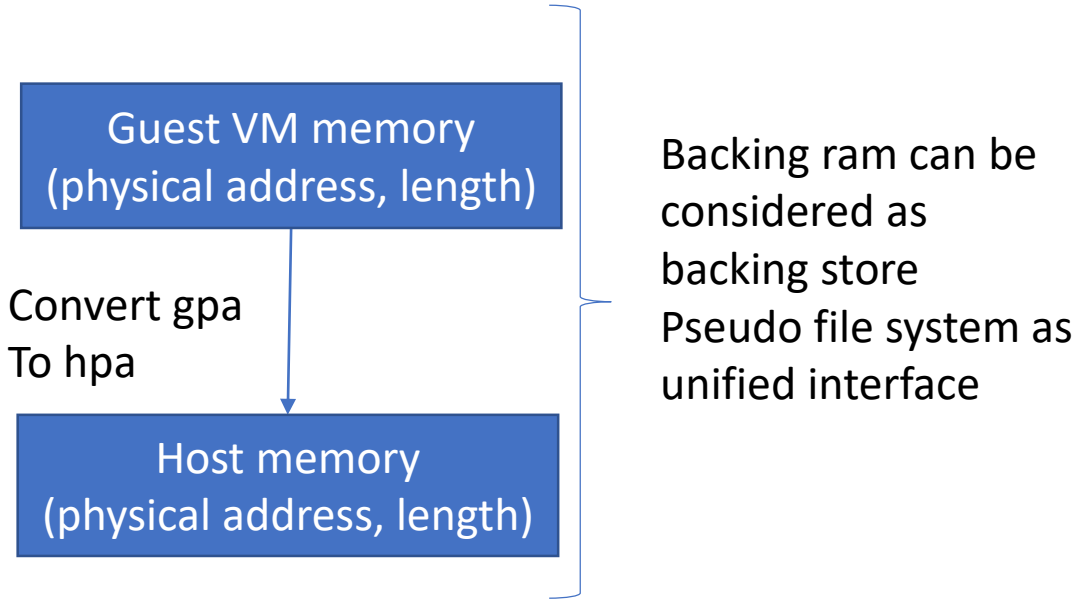
new

# Pseudo file system as unified interface

- memory as backing store
  - sector_t as pfn: Similar to DAX
  - Various operations defined(fallocate, punch hole, seal, fanotify)

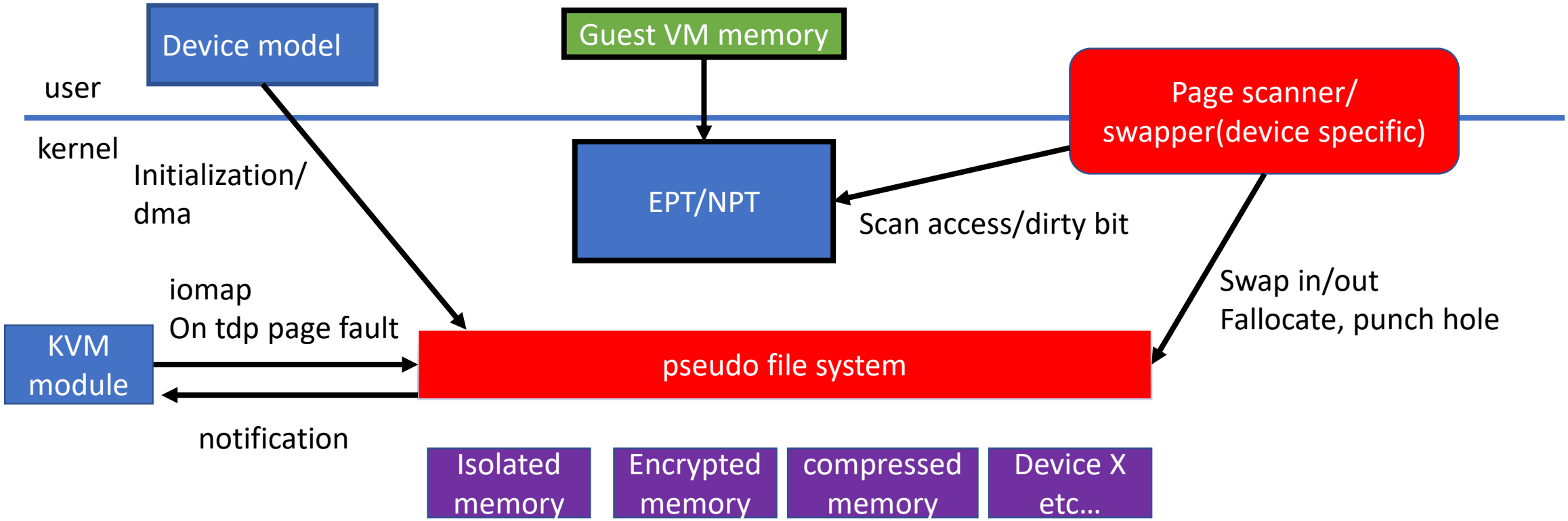



Regular file on local file system



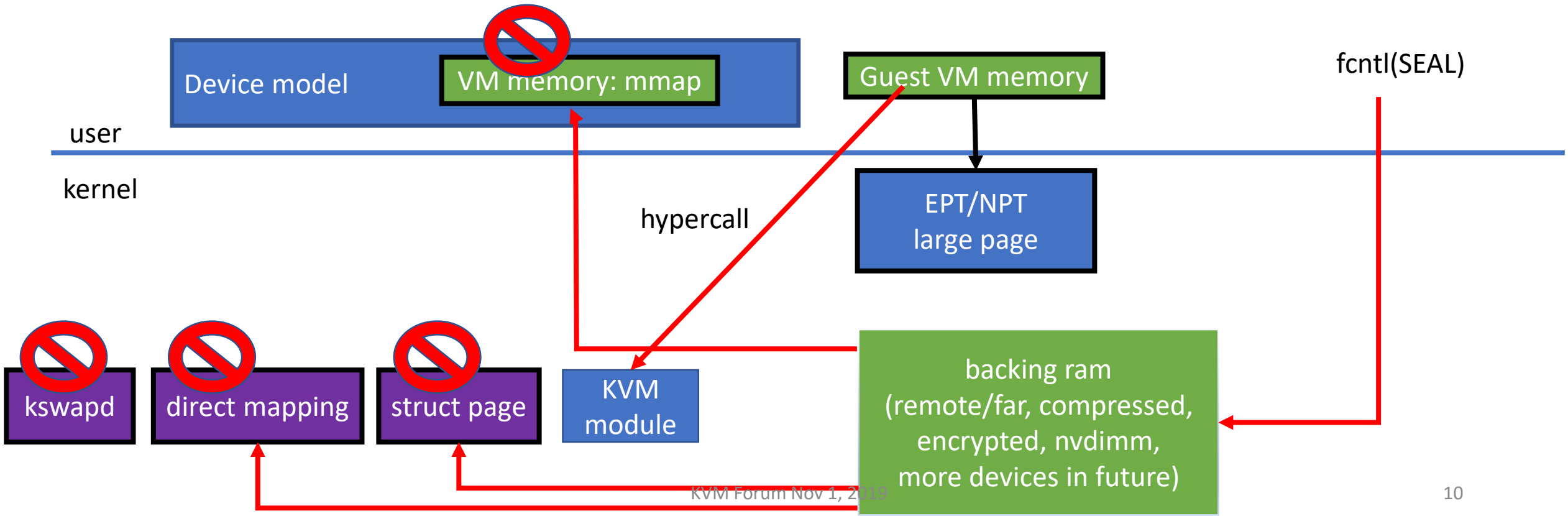

Guest VM memory conversion

# Pseudo file system for kvm



# Memory isolation

- After initialization is done,
- Remove the region from kernel mapping(directmap)
- Optional: no struct page by hot-unplug
- Ensure that no user space mapping (or error)



# PoC: current status

- Pseudo file system
- Adapter to reserved memory
- Modification to kvm kernel module not to use page fault.(WIP)

# Future work

- Discussion/Post patches
- Benchmark
- Exercise more backing memory technology
  - Encrypted memory
  - Memory compression
  - Live migration(precopy, postcopy)
  - virtio
- Preparing for memory isolation

# Summary

- Need to revise guest memory management
- New unified interface for it as pseudo file system

Next step

- Discussion to align with other activities to make progress.