1. Device IO
   - CPU
   - Device
     - Device mem
     - poll
   - interrupts register

2. Direct Memory Access (DMA)
   - use CPU cycles when data moved
     - large data
     - scattered data

3. Mix interrupts with polling
   - network: interrupt -> interrupt disable
     - poll for timeout/ until no data
     - reenable interrupts

4. How should IO be virtualized?
   - guest can't program sensitive device registers
   - device I/O on behalf of guest must be
     - verified by the hypervisor - addresses part
     - of guest domain

5. Approach 1: data copy
   - DMA to/from trusted memory
   - programmed by dom 0
   - don't copy, validate guest memory

6. Validation of guest memory
   - in DMA descriptor
   - don't copy, validate guest memory

7. Approach 2: pin pages inside hypervisor
   - don't copy, validate guest memory

8. Approach 3: pin pages inside hypervisor
   - upon DMA request validation
     - pin page inside hypervisor's own
     - page tables, cannot be reallocated
     - to anyone else until DMA complete
7. Need for hardware support
   - Validation per DMA data transfer
   - 30% of wire network throughput

8a. Translation: useful when device can only access part of host memory (e.g., 32-bit device versus 64-bit CPU)

8b. Isolation: TDMMU has a page table. Entries are only mapped/unmapped by privileged entity (hypervisor) after validation.

9. Data movement with TDMMU & address space isolation

10. Single Root I/O Virtualization (SR-IOV)
    - Expose single device as multiple devices (PF/VF/UNIC)
    - Dedicated DMA data movement registers per VF
    - Sensitive registers programmed by privileged entity
    - Direct data to the right VF in hardware