2. Virtualization
   - time sharing
   - resource sharing
   - limited direct
   - execution
   - Isolation
   - Performance

3. Isolation why?
   - process can be buggy
   - crash
   - infinite loop
   - process can be malicious
   - take over other processes/users

4. System Resources
   - CPU, mem, disk, net
   - access only after
     vetting, but process directly
     executed
   - Get hardware support
     - privilege levels
       - traps

5. trap table - how data structure
   - OS sets up trap
     table at boot time

6. trap process
   - process
   - OS
   - Kernel Mode
     - Privileged operations
       - traps
     - Kernel Mode
     - Trap
     - entry
     - result
     - address of fun/ handler

7. system calls
   - library to do req assembly

8. Basic limited direct exe.
   - OS (kernel mode)
   - process (user mode)

   Boot time
   - set up all trap
     handlers & table
   - Run time
     - initialize text, data, stack w/ args
     - save regs
     - return from trap

9. sharing between
   - processes
     - cooperative approach:
       - wait for sys call
       - yield()
     - non-cooperative approach
       - Timer Interrupt
         - hardware support once again

10. Timer Interrupt
    - start timer
      interrupt &
      set up handler
    - whenever
      choose to
      move to
      different
      process
    - save reg to kernel
      stack
    - move to kernel
      mode
    - trap to handler
1. Virtualizing memory

- Basic model
  - OS
  - user 1
  - user 2
- Address space
- Virtual addresses
- Actually located at different locations

2. Dynamic relocation

- OS
- code
- stack
- User abstraction
  - "Fixed" addressing
  - Instructions regardless of location

3. Hardware translation

- Base & bounds
- Load register addr
- Virtual to phy addr
  - If < free + vaddr < bound
  - Load from base + vaddr
- Else fault -> trap
- Memory Management Unit (MMU)

4. Primitives for hardware

- Address translation
  - Privilege modes
  - Base & bound registers
  - Check vaddr within bound
  - Privileged ins to update base/bounds ⇒ "effective"
  - Privileged ins to register exception handlers
  - Raise exceptions on invalid addresses
  - OS must start managing memory

5. Segmentation

- Empty mostly addr spaces
- Generalize to many base and bounds

6. Hardware for Segmentation

- Tell hardware which segment?
- Top & bot of addr
- Memory protection bits
- Read? Write? Execute?
- Problems
  - Context switch
  - Save & restore segment descriptor
  - Non-compact memory layouts

7. Paging

- Use fixed size chunks of many
- Memory

8. Translation

- Vaddr offset
- Per process data structure
22. Translation
   - Check valid, protection, PTE valid.
   - Else trap!
   - Minor fault (retrieve shared)
   - Major fault (retrieve from disk)
   - Invalid fault (vaddr not valid)

24. Address translation cache (TLB)
   - Efficient memory, small → cache PTEs
   - First check TLB → hit → use (protection etc.)
   - Miss → walk page table

26. TLB entry
   - Up# → pf# → protection bits etc.
   - Fully associative cache

27. TLB + context switches
   - Per process data structure of page-tables
     - Option 1: TLB flush
     - Option 2: tagged TLB
     - Use address space ID

29. TLBs make paging feasible
   - Physically indexed vs. virtually-indexed caching
3) System virtualization

1) OS assumes it is the most privileged entity.
2) OS virtualization: why?
   - time sharing uses
   - unit of software/app deployment
   - any software & config can be run
3) Actually property of both OS & hardware architecture (together provide Isolation)

3.1) Theorem (basic result)

\[ \text{(unmodified guest OS) \rightarrow efficiency virtualized if } S_2 \cup S_3 \leq S_1 \] (full virtualization)

\[ \rightarrow \text{"trap and emulate" virtualization} \]

\[ \rightarrow \text{sufficient condition, unmodified guest OS} \]

\[ \rightarrow \text{workaround tech} \]

1) Dynamic recompilation - replace critical instructions at run-time.
2) Cache emulation code
   - assist through hardware
3) Paravirtualization - post before running

3.2) Virtual machine monitor (VMM)

\[ (\text{VMM}) \rightarrow 1974 \]

\[ \rightarrow \text{virtualize all system resources (processor, memory, disk, network, I/O, ...)} \]

3) Requirements (Tyck & Goldberg)

1) Equivalence
   - same as direct execution on underlying real machine
2) Resource control/safety
   - VMM must have complete control of resources

3.3) ISA classification

1) Privileged instructions
   - must trap on user mode
2) Control sensitive
   - can change processor mode or "whether something will trap in 1st place"
   - (e.g., change memory bounds)

3) Behaviour sensitive
   - effect of instruction depends on mode or bounds registers

3.4) Containers

1) Kernel namespaces
   - (access isolation, networking, processes, file systems)
2) Control groups (process management, memory, block)

3) Layering
   - copy on write
   - union filesystem

\[ \text{CPU memory block} \]
1. CPU → ring 1 or (else trap ring 0)

2. I/O virtualisation
   - Ring buffers
   - producer-consumer
descriptor data separation / zero copy

3. Memory
   - guest uses manageable
unified virtual address
   - and effect of flush on xen

4. Management
   - separate policy from xen