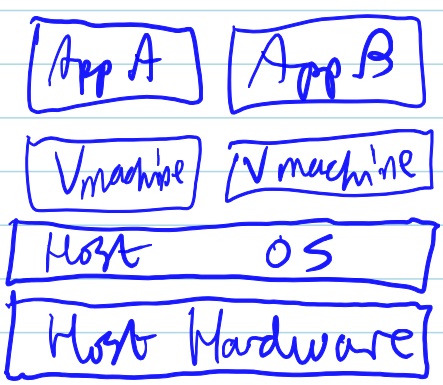
Virtualisation

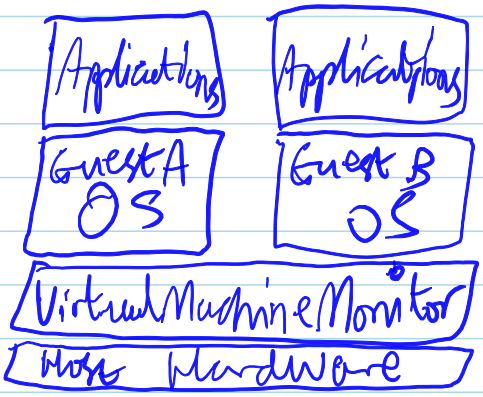
- Portability: one program should always feel like it has whole memspace available to it
- Resource Mgmt.
  - allowing multiple processes to run at once
  - Heap starts at bottom, stack at top
- SECURITY: memory protection
- SAFETY:
  - one process crashing SHOULDN'T imply other one crashes
- How to achieve virtualisation:
  - FIDELITY
  - PERFORMANCE — no penalties
  - SAFETY

- Virtualisation Technologies:
  - CPU
  - VM, obviously
  - STORAGE virtualisation
    - partitioning or merging (large/small) storage space
  - Virtual Machines (e.g Java)
  - System Virtualisation

(!) Isolate hardware from software using it
- VM: amount, layout
- Storage: position, size, location of virtual disk
- JVMach...: abstract ALL hardware
  - instruction encoding
  - registers
  - bytecode: NOTHING known about hardware
    - REALLY portable (on any machine can run)

- System: abstracting away all complexities
  - I/O devices
  - Memory
- OS + Virtualisation
  - isolates application from HW but still closely integrated
  - INSTALLING → OS → creates STATE
    - application on
  - Moving installed application between systems is HARD
    - Paths will differ
    - Hardware VERY different
    - but still POSSIBLE
  - Same with installed OS too
  - moving a running application is almost IMPOSSIBLE
- Process Virtualisation:
  - run process under control of layer of software
- System Virtualisation
  - run SYSTEM under control of layer of software

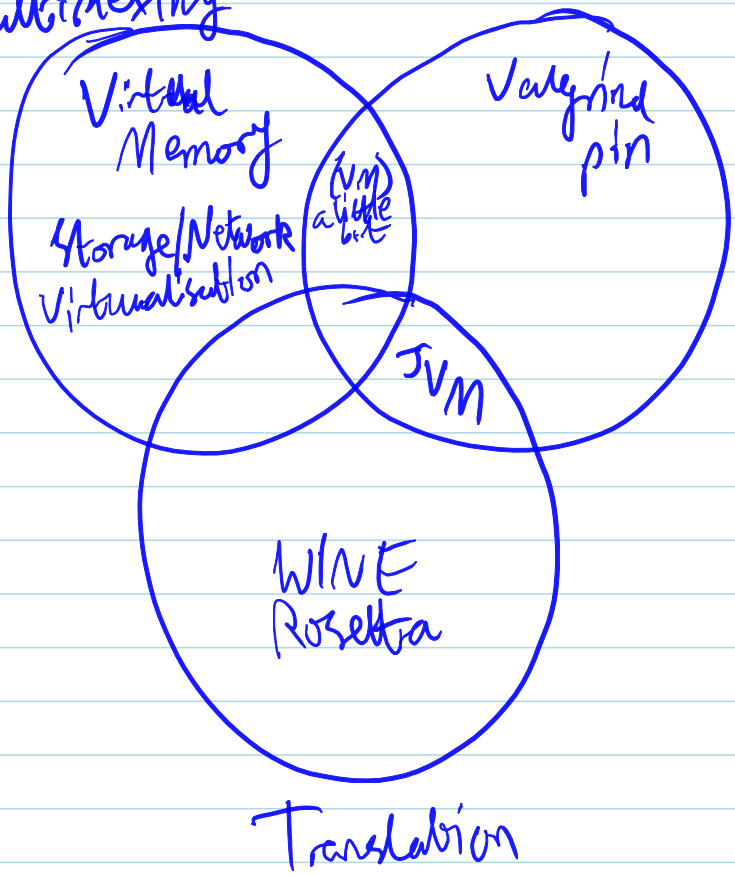




resource mgmt:
every process has its own unique memory space

- Virtualisation able to:
  - Change level of abstraction
  - Multiplex/demultiplex resources
  - Translate between equivalent facilities
- JVM:
  - Interprets then compiles "byte code" files
  - "Write once, run anywhere"
- Rosetta:
  - Translates stuff on the fly
- pin
  - annotate Intel binary
- valgrind
  - check memory references and dynamic allocation

multiplexing



- JVM, rosetta: can do all 3 BUT don't have all 3 as main purposes