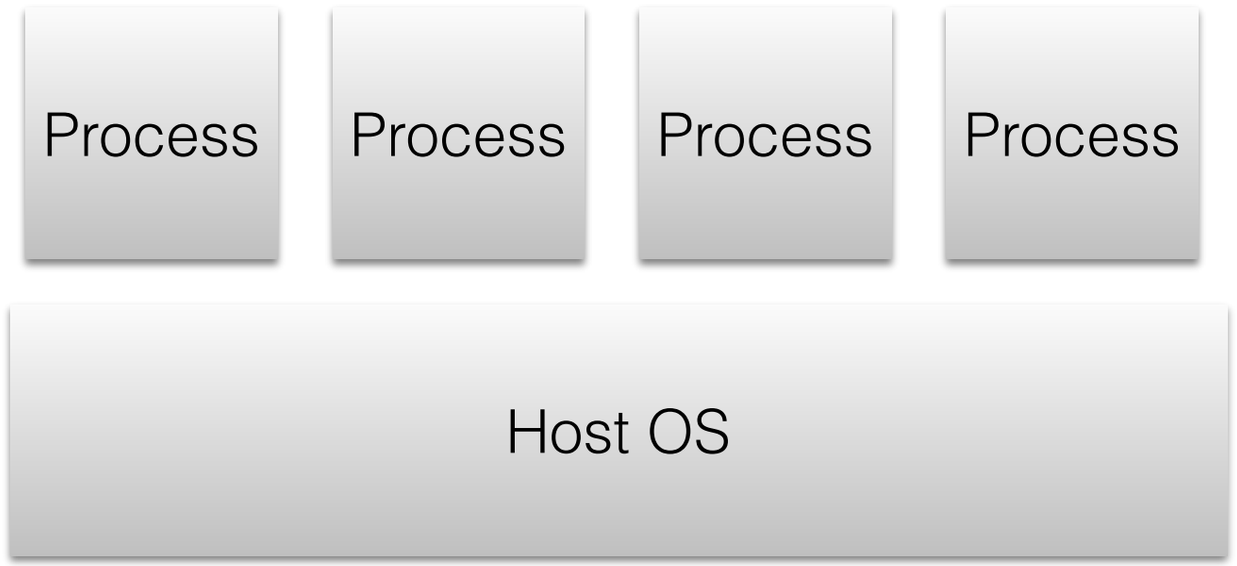
Operating-system-level virtualization and Containers

Kartik Gopalan

Isolation

- Isolation
  - Limiting what/who a process/application can see.
  - Limiting who can see a process/application
- Two extremes
  - Traditional Process
  - System Virtual Machines

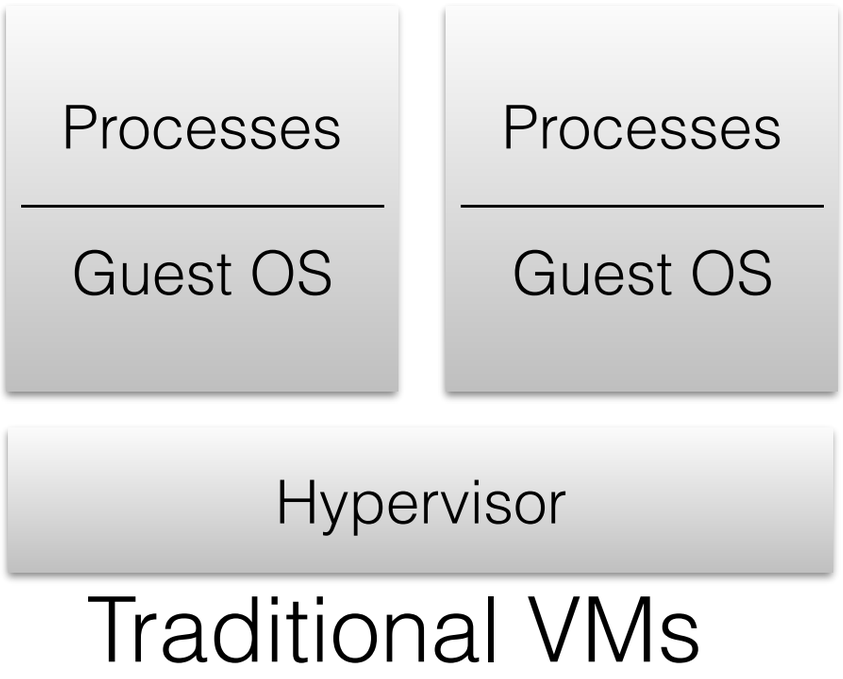
Traditional Processes



Traditional Processes

- Each process gets its own
  - Virtual memory
  - One or more virtual CPUs (threads)
  - Access to OS services via system calls
- All co-located processes can see/share a lot (in an OS-controlled manner)
  - File system, storage, network, and I/O devices
  - Other processes (for Inter-process communication)

System Virtual Machines



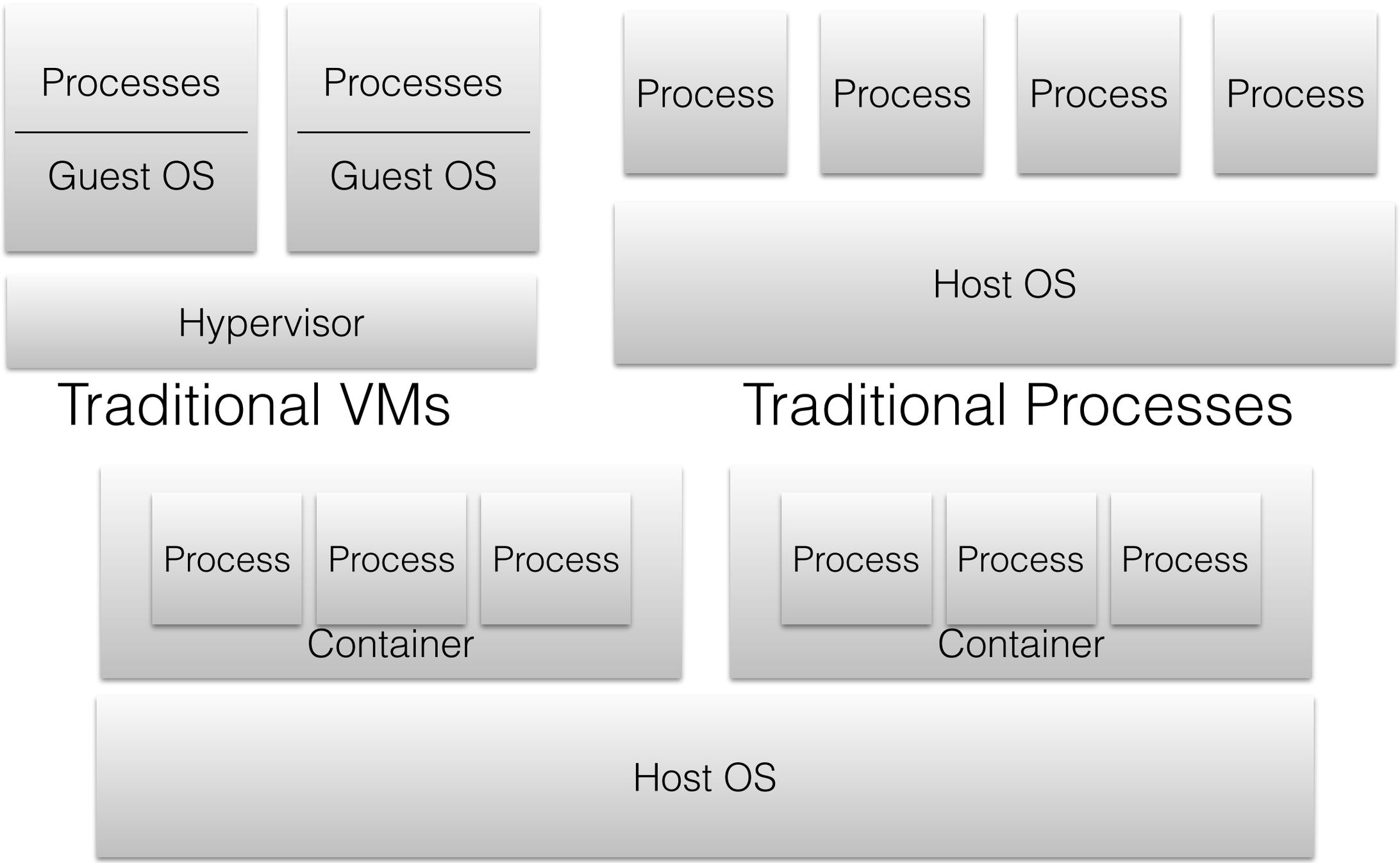
Traditional VMs

- Co-related processes grouped into VMs
- Each VM has its own
    - Guest OS
    - Guest physical memory (“virtualized” view of memory seen by guest OS)
    - One or more virtual CPUs
    - Virtual I/O devices: virtual disk, virtual network
- Ideally: Co-located VMs don’t see/share ANYTHING

What level to isolate?

- Processes share too much
  - Great performance but not isolated enough
- System VMs are too heavy
  - Great Isolation but too heavy due to separate guest OS per VM
- Operating-system-level virtualization
  - Multiple isolated user-spaces
  - Share one kernel.
  - Native performance

Containers



Containers group traditional processes and restrict what resources they can see.

Chroot

- An early precursor to modern namespaces
- Change root directory for the calling process and its children to a given path
- `$ chroot NEWROOT`

OR

- `$ chroot(path)`
- “This call changes an ingredient in the pathname resolution process and does nothing else.” — man chroot
- Not secure. Lots of ways to escape chroot jail.

FreeBSD Jails

- Builds upon chroot to compartmentalize files and other resources
- Jails protects rest of the system from the jailed process
  - Not the other way around!
- Virtualized resources
  - file system,
  - the set of users, including own root account.
  - networking subsystem
- Again: Jail escapes were possible!

Linux Namespaces

- “A **namespace** wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource.” - from “$ man namespaces”

- PID Namespace
    - Limit the set of processes that can be see each other.

- IPC namespace
    - Limit the set of processes which are allowed to communicate with each other

- Filesystem namespace
    - Limit which part of filesystem is seen by a process group

- Network namespace
    - Unique IP address host name, domain name, etc for a group of processes

- User Namespace
    - User and Group IDs

Cgroups (Control Groups)

- Beancounter
  - performs resource accounting for groups of processes
- Allows administrator to set soft/hard limits on usage of memory, network bandwidth, CPU etc.
- Typically used alongside with Linux namespaces

Single System Image

- Extend the notion of namespaces to multiple physical machines
- Multiple machines look like one (or more) namespace(s)
    - PID namespace
    - IPC Namespace
    - Filesystem namespace
- Process migration
    - Allows moving processes from one machine to another without changing its namespace.
- Examples: MOSIX, OpenSSI, Kerrighed