VMs, Unikernels and Containers: Experiences on the Performance of Virtualization Technologies

Felipe Huici, Filipe Manco, Jose Mendes, Simon Kuenzer
NEC Europe Ltd. (Heidelberg)

In the Beginning…

In the Beginning…

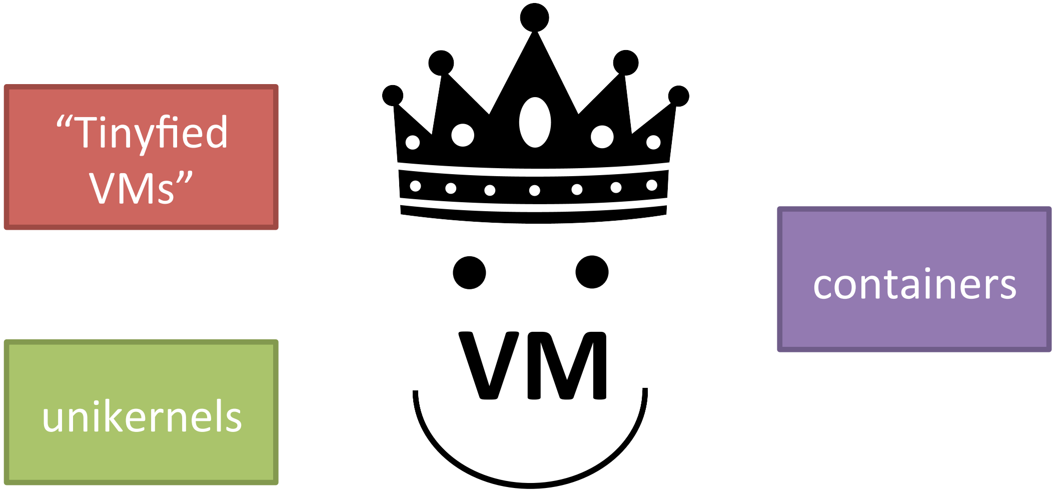
“Tinyfied VMs”

In the Beginning…

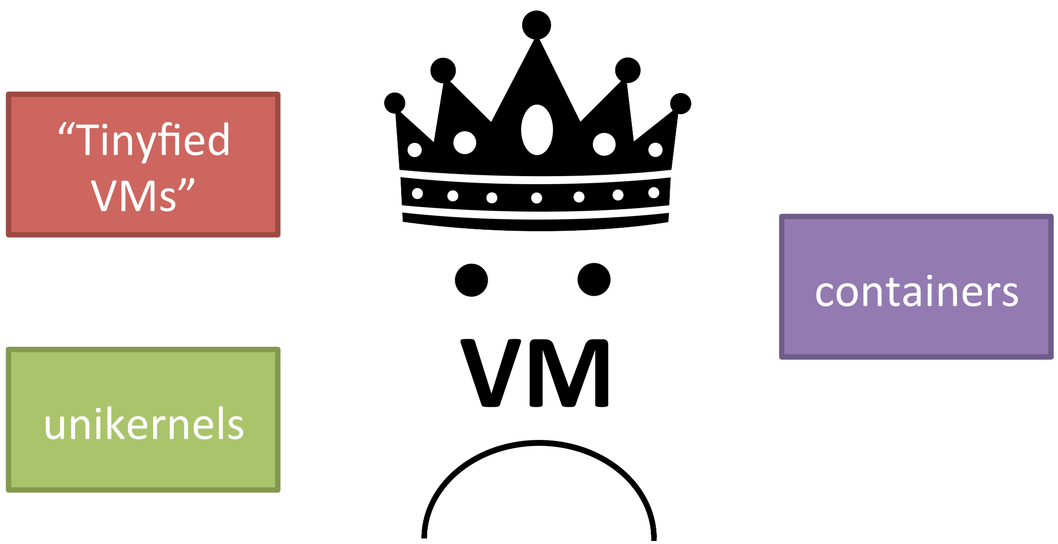
“Tinyfied VMs”

unikernels

In the Beginning…



In the Beginning…



Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput

Virt. Technology Benchmarking

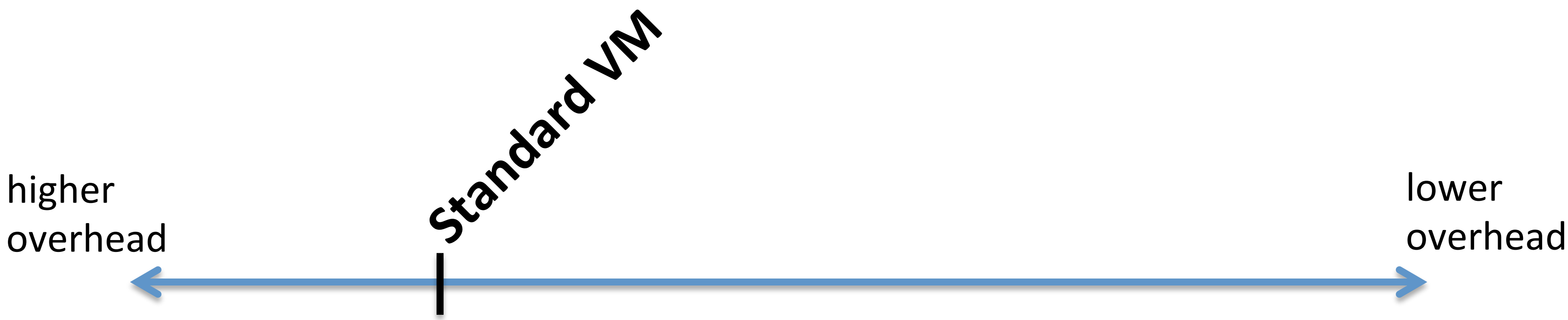
- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput

higher overhead

lower overhead

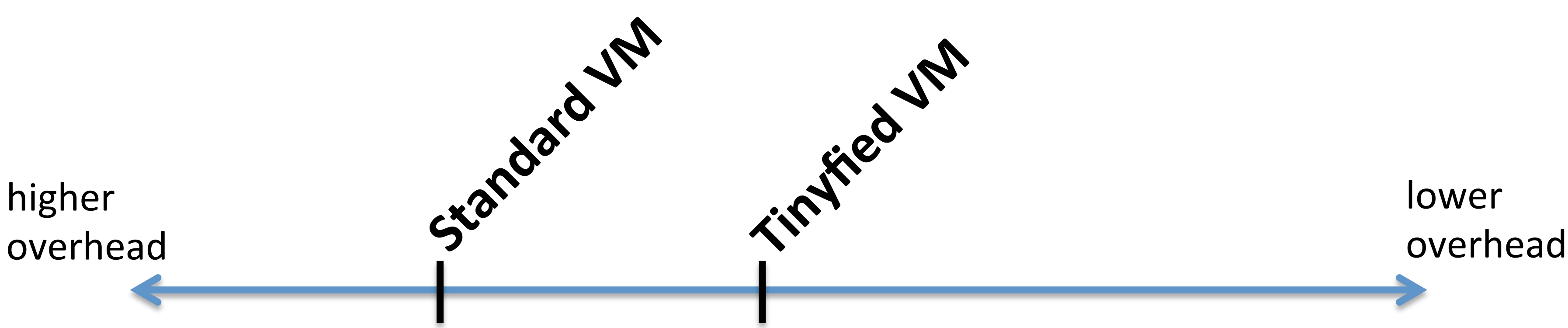
Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput



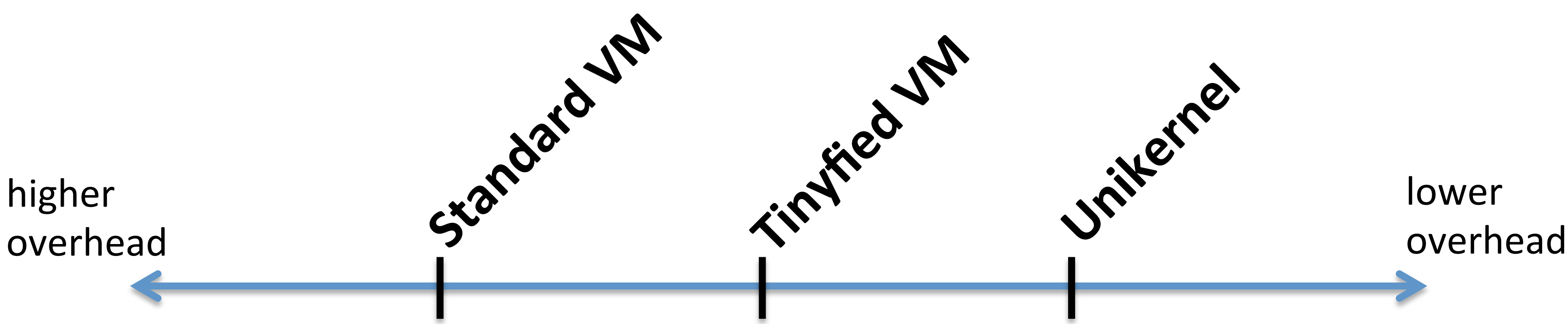
Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput



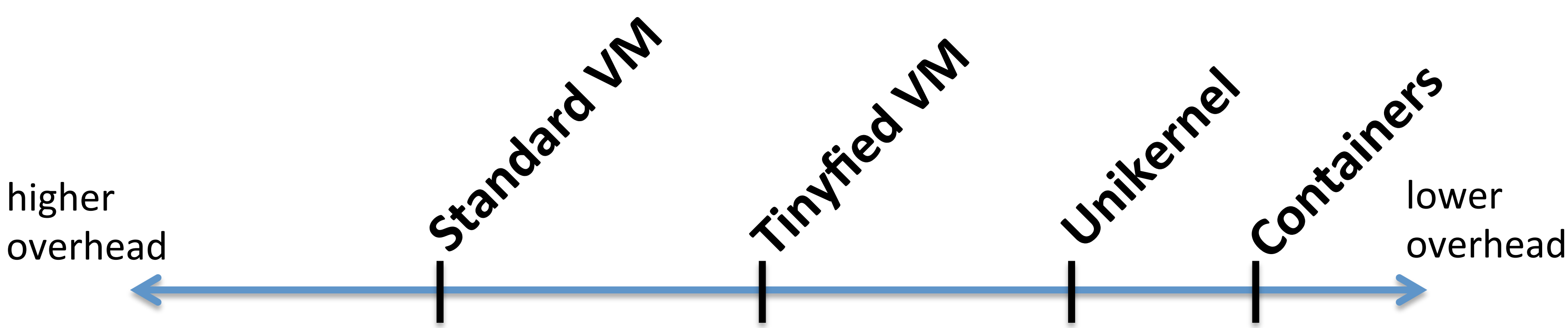
Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput



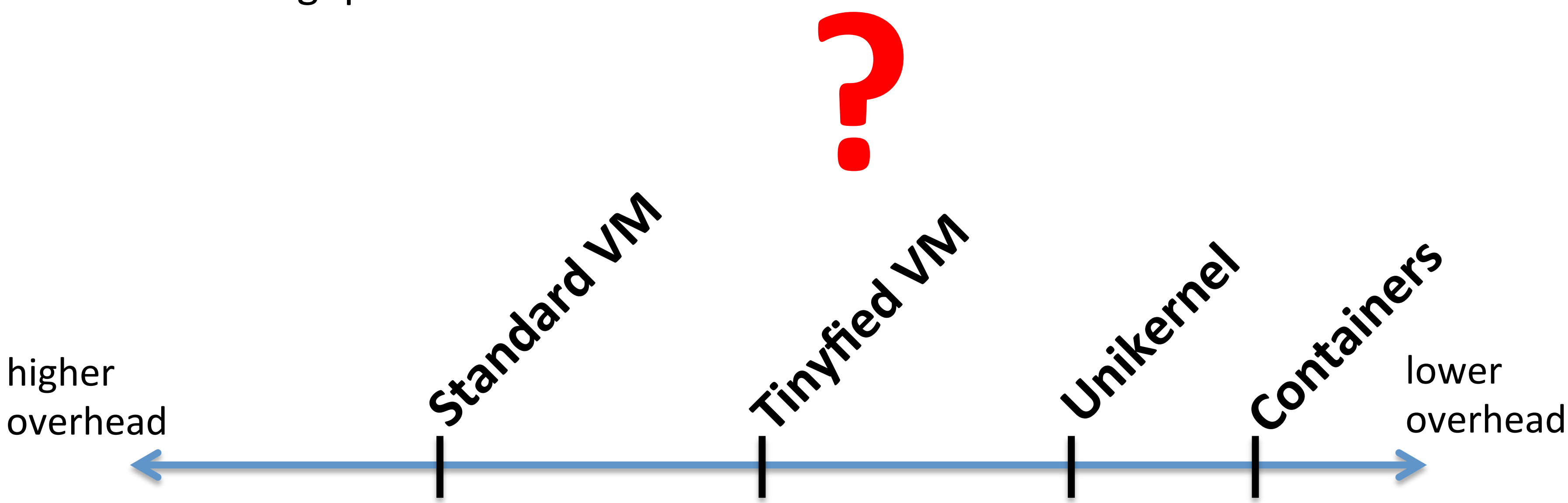
Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput



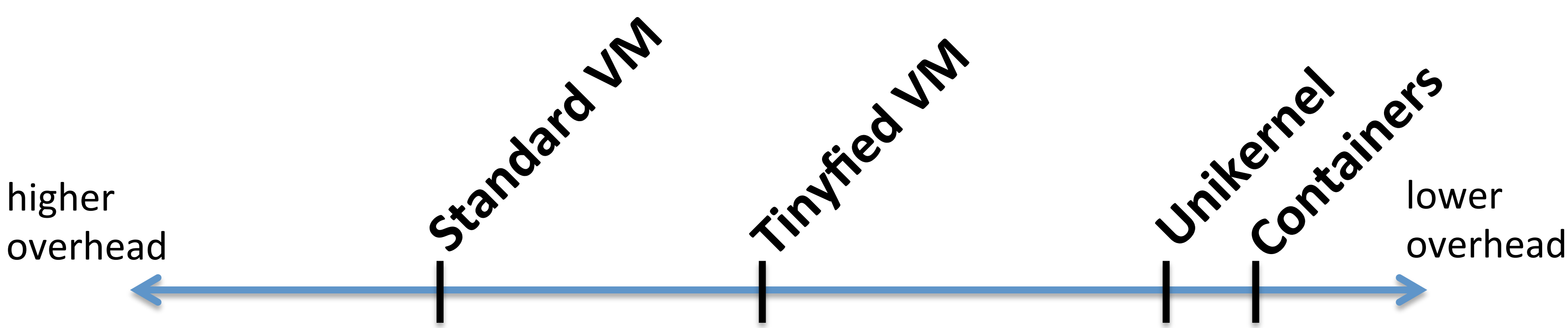
Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput



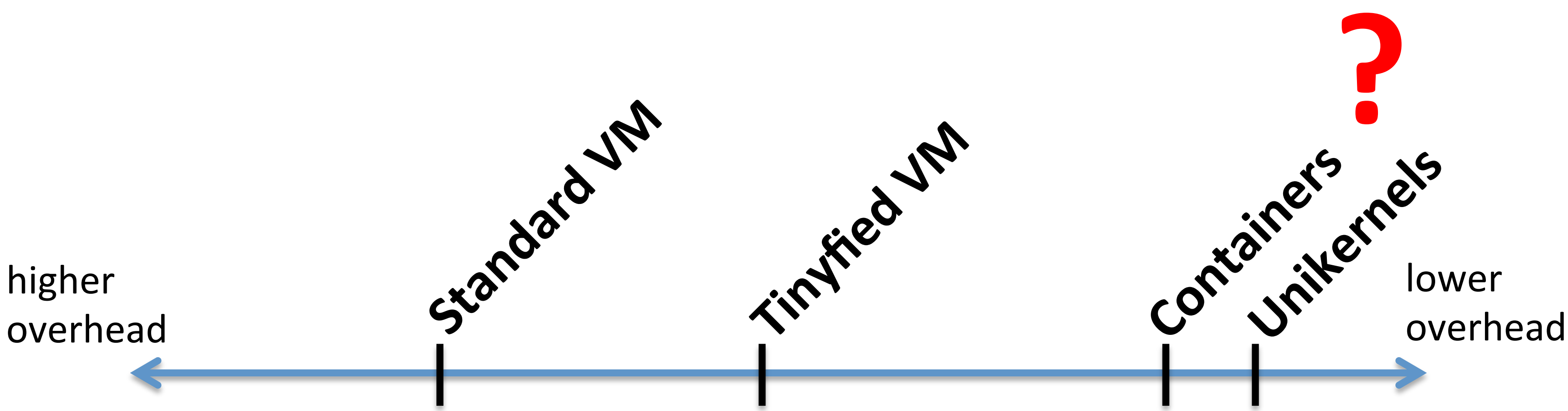
Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput



Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput



Virtualization Technology Benchmarking

- Metrics:
  - VM image and memory consumption: ls, top, xl
  - VM creation time: SYN flood + RST detection
  - Throughput: iperf, guest to host (TCP traffic)
  - RTT: ping flood
- VM-based tests run on both Xen and KVM
- Hardware: x86_64 server with an Intel Xeon E5-1630 v3 3.7GHz CPU (4 cores), 32GB RAM.

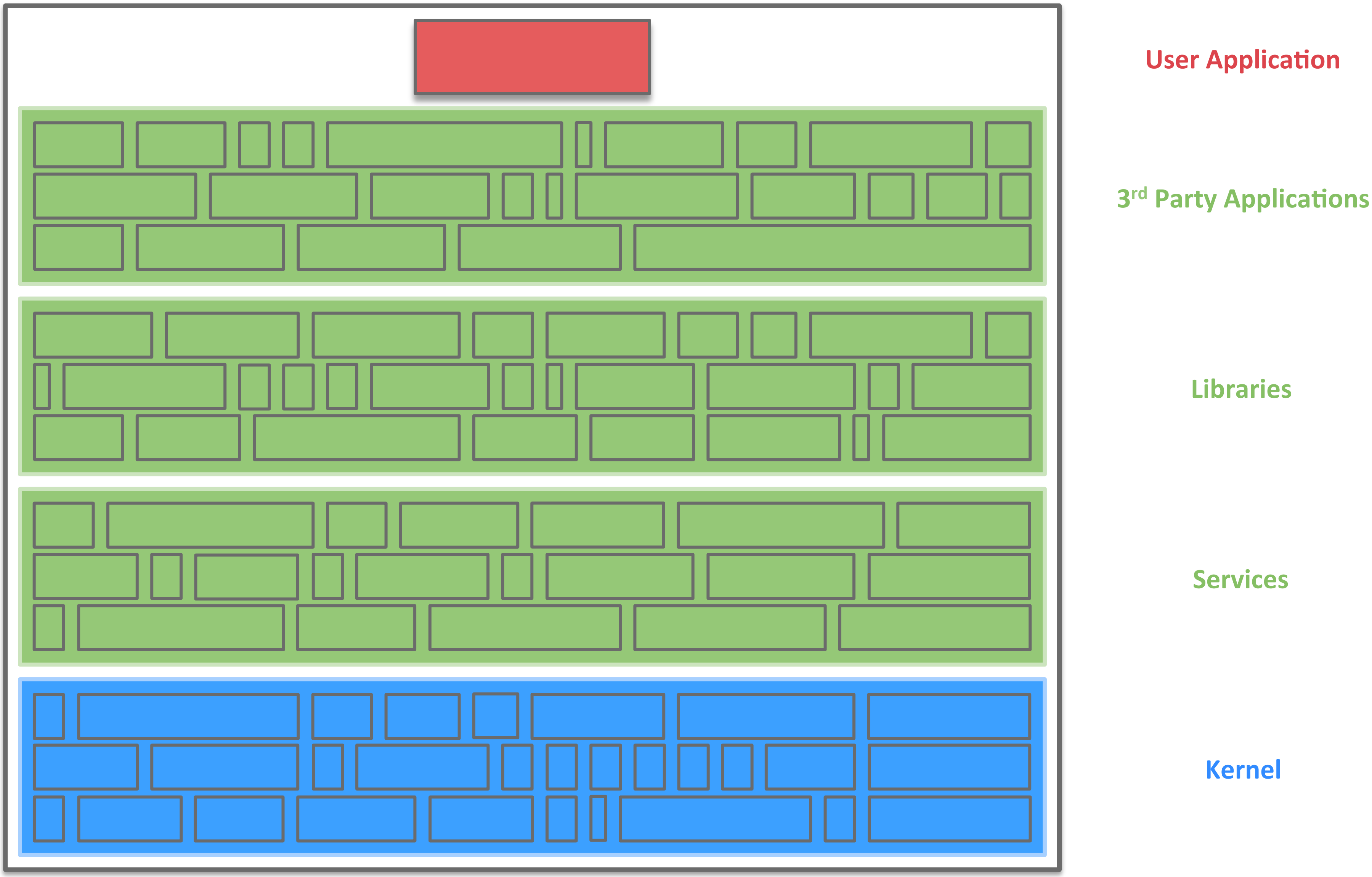
Virtualization Technologies

- “Standard” VM
  - Standard Debian-based Linux VM
- “Tinyfied” VM
  - Tinyx, based on Linux kernel/busybox
- Unikernel
  - On Xen: MiniOS + miniperf
  - On KVM: OSv + iperf
- Containers
  - Docker

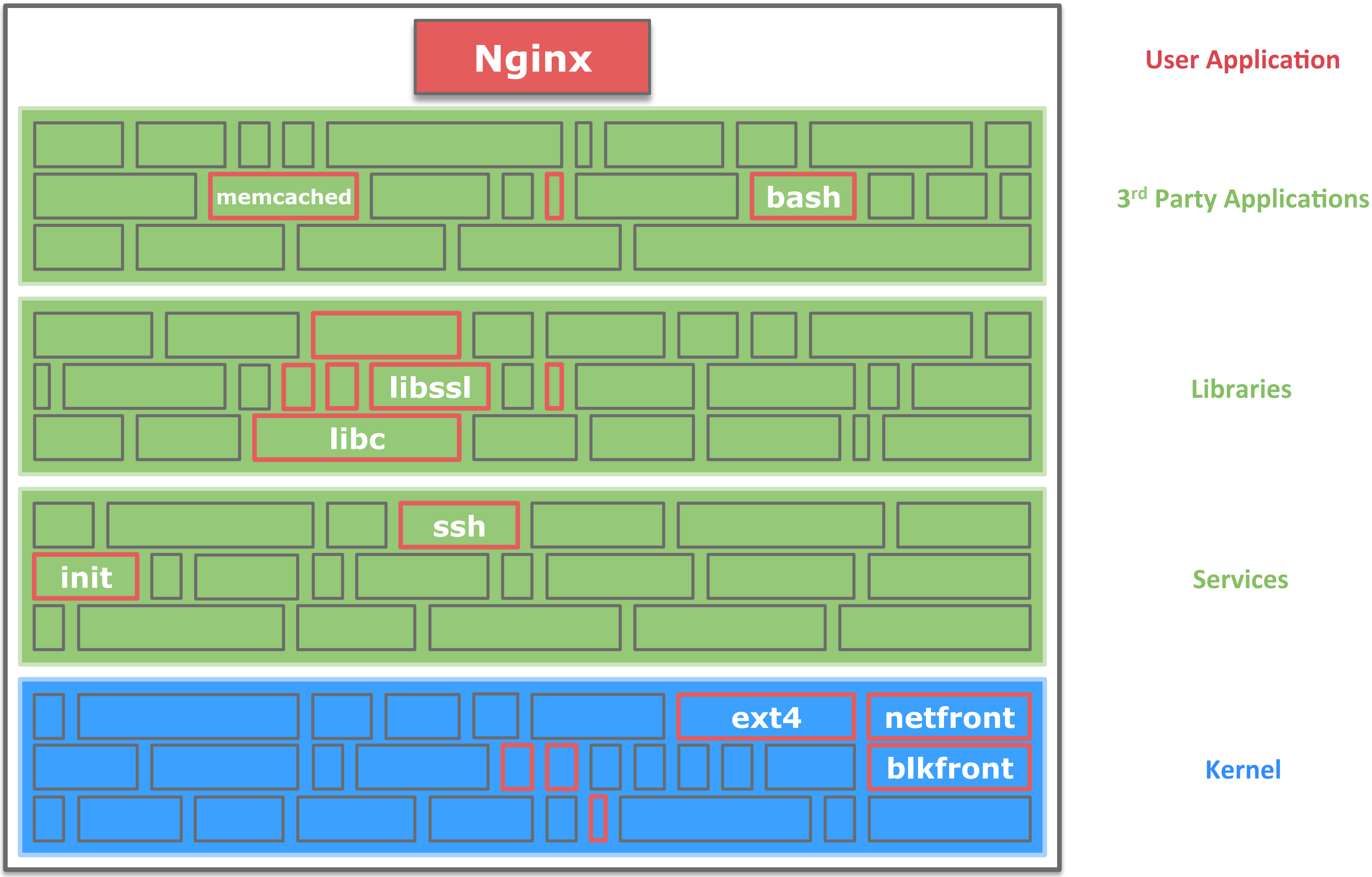
Virtualization Technologies

- “Standard” VM
  - Standard Debian-based Linux VM
- “Tinyfied” VM
  - Tinyx, based on Linux kernel/busybox
- Unikernel
  - On Xen: MiniOS + miniperf
  - On KVM: OSv + iperf
- Containers
  - Docker

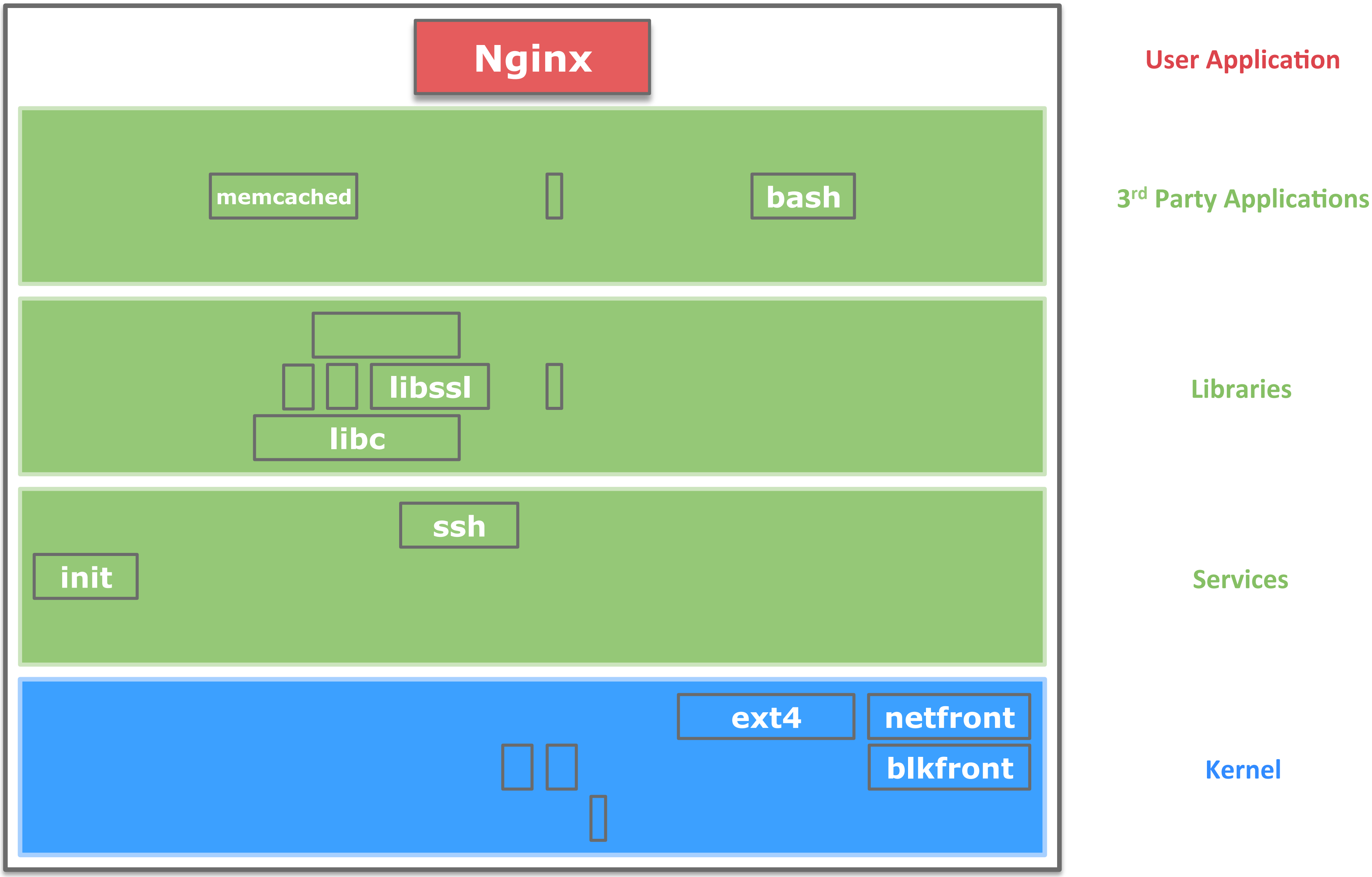
Standard VM: Application on Top of Distro



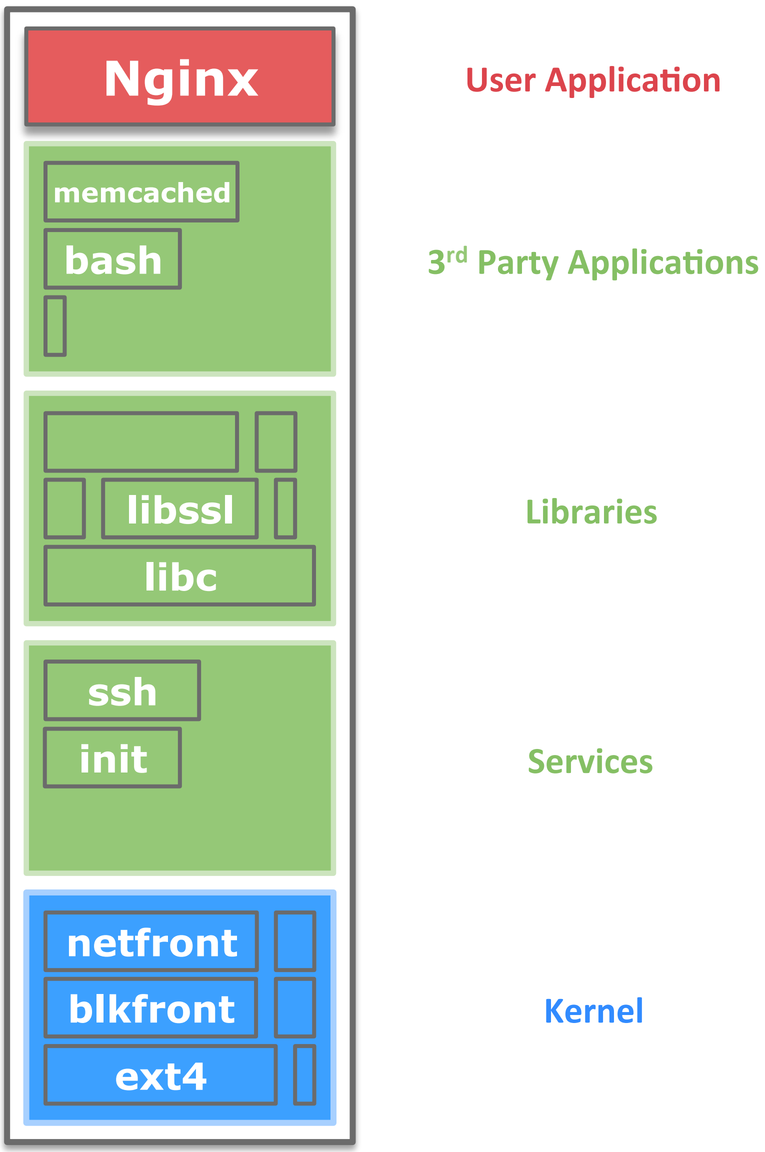
Most of the VM not Used…



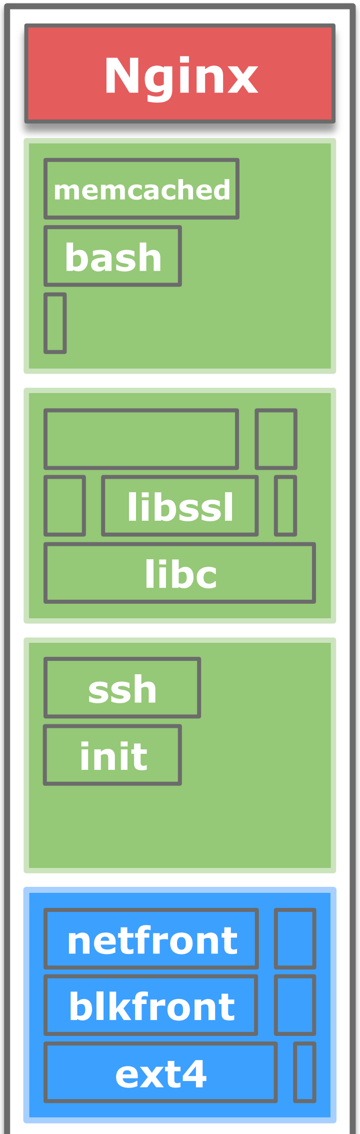
Tinyx: Keep Only What's Needed

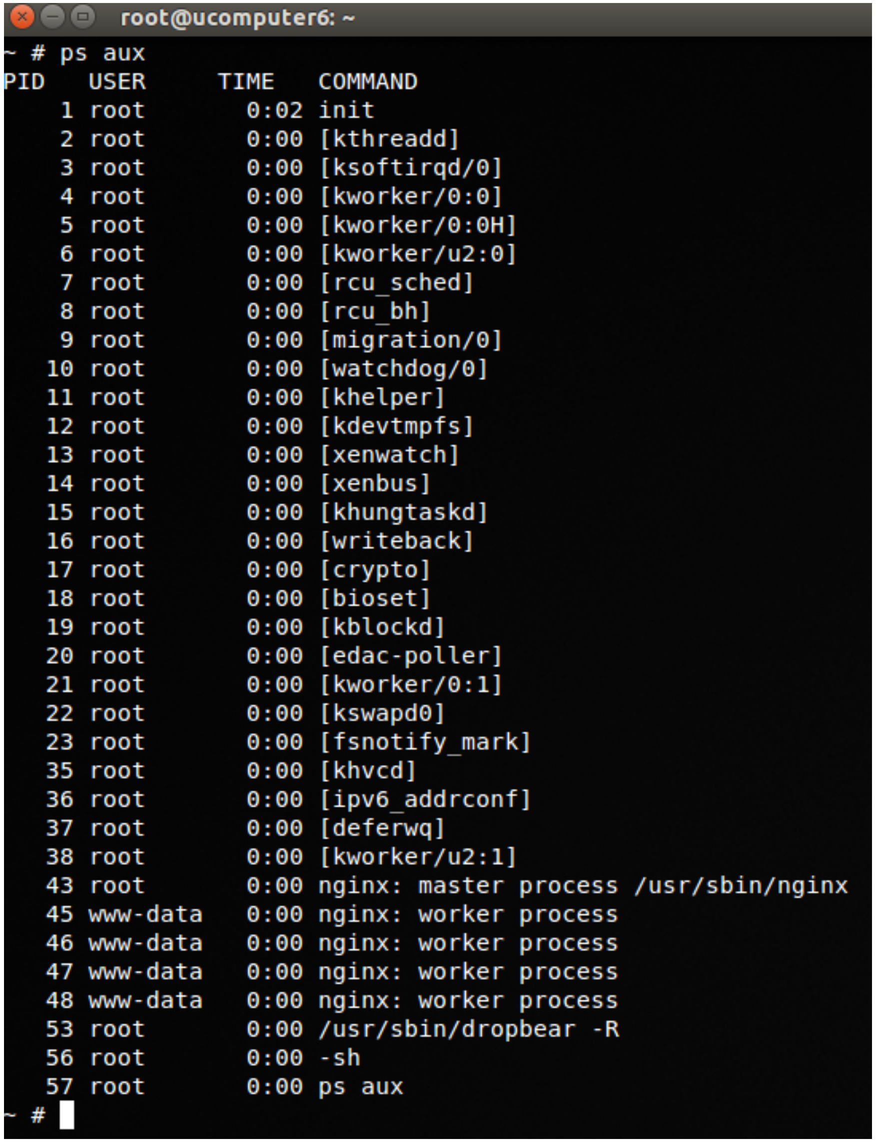


Tinyx: Taylor-made Distro



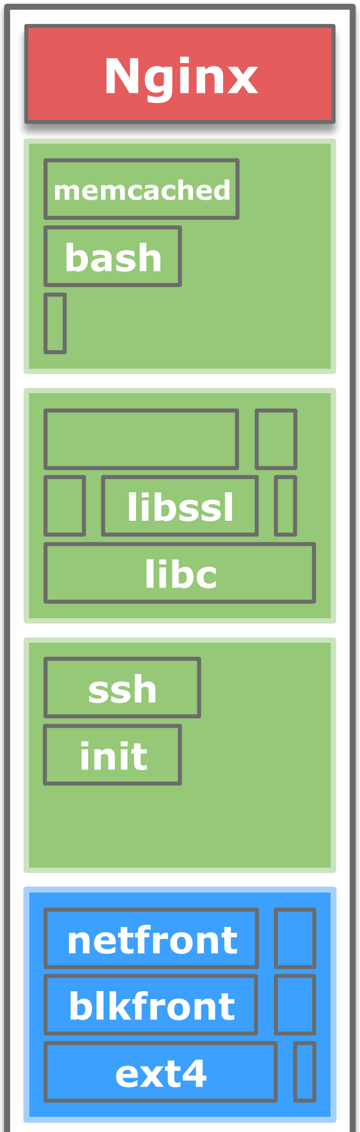
Tinyx: Taylor-made Distro






```
~ # ps aux
PID   USER     TIME   COMMAND
    1 root       0:02 init
    2 root       0:00 [kthreadd]
    3 root       0:00 [ksoftirqd/0]
    4 root       0:00 [kworker/0:0]
    5 root       0:00 [kworker/0:0H]
    6 root       0:00 [kworker/u2:0]
    7 root       0:00 [rcu_sched]
    8 root       0:00 [rcu_bh]
    9 root       0:00 [migration/0]
   10 root       0:00 [watchdog/0]
   11 root       0:00 [khelper]
   12 root       0:00 [kdevtmpfs]
   13 root       0:00 [xenwatch]
   14 root       0:00 [xenbus]
   15 root       0:00 [khungtaskd]
   16 root       0:00 [writeback]
   17 root       0:00 [crypto]
   18 root       0:00 [bioset]
   19 root       0:00 [kblockd]
   20 root       0:00 [edac-poller]
   21 root       0:00 [kworker/0:1]
   22 root       0:00 [kswapd0]
   23 root       0:00 [fsnotify_mark]
   35 root       0:00 [khvcd]
   36 root       0:00 [ipv6_addrconf]
   37 root       0:00 [deferwq]
   38 root       0:00 [kworker/u2:1]
   43 root       0:00 nginx: master process /usr/sbin/nginx
   45 www-data   0:00 nginx: worker process
   46 www-data   0:00 nginx: worker process
   47 www-data   0:00 nginx: worker process
   48 www-data   0:00 nginx: worker process
   53 root       0:00 /usr/sbin/dropbear -R
   56 root       0:00 -sh
   57 root       0:00 ps aux
~ #
```

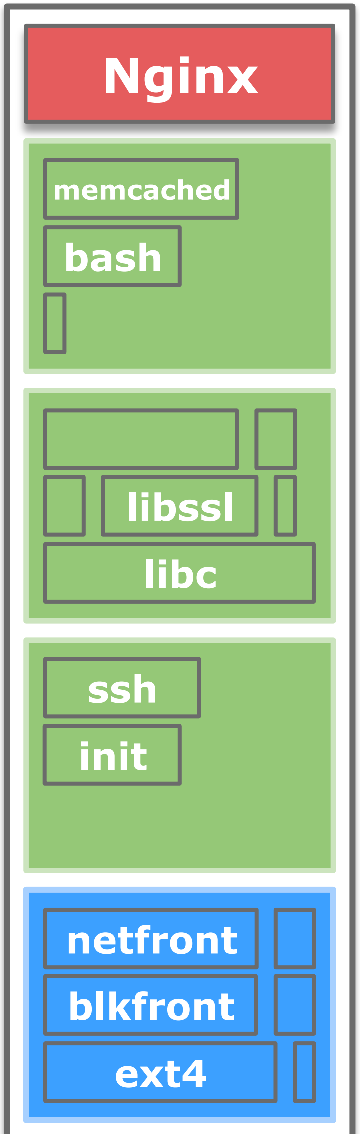
Tinyx: Taylor-made Distro



root@ucomputer6: ~

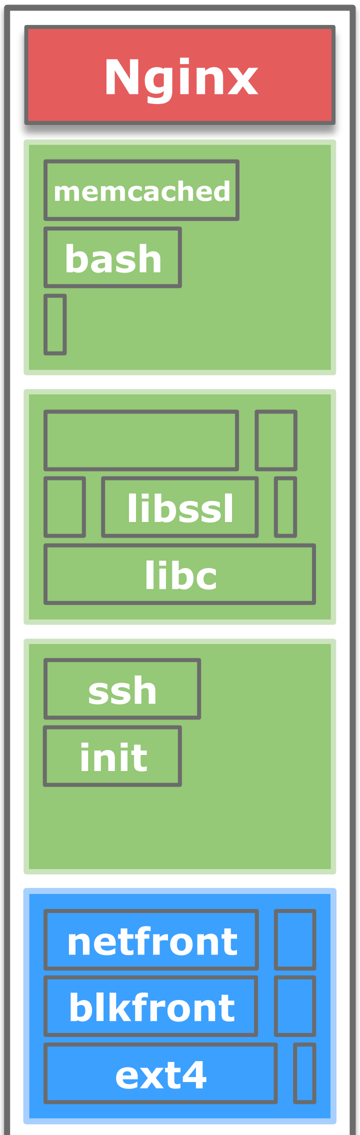
```
~ # ps aux
PID   USER      TIME   COMMAND
    1 root       0:02 init
    2 root       0:00 [kthreadd]
    3 root       0:00 [ksoftirqd/0]
    4 root       0:00 [kworker/0:0]
    5 root       0:00 [kworker/0:0H]
    6 root       0:00 [kworker/u2:0]
    7 root       0:00 [rcu_sched]
    8 root       0:00 [rcu_bh]
    9 root       0:00 [migration/0]
   10 root       0:00 [watchdog/0]
   11 root       0:00 [khelper]
   12 root       0:00 [kdevtmpfs]
   13 root       0:00 [xenwatch]
   14 root       0:00 [xenbus]
   15 root       0:00 [khungtaskd]
   16 root       0:00 [writeback]
   17 root       0:00 [crypto]
   18 root       0:00 [bioset]
   19 root       0:00 [kblockd]
   20 root       0:00 [edac-poller]
   21 root       0:00 [kworker/0:1]
   22 root       0:00 [kswapd0]
   23 root       0:00 [fsnotify_mark]
   35 root       0:00 [khvcd]
   36 root       0:00 [ipv6_addrconf]
   37 root       0:00 [deferwq]
   38 root       0:00 [kworker/u2:1]
   43 root       0:00 nginx: master process /usr/sbin/nginx
   45 www-data   0:00 nginx: worker process
   46 www-data   0:00 nginx: worker process
   47 www-data   0:00 nginx: worker process
   48 www-data   0:00 nginx: worker process
   53 root       0:00 /usr/sbin/dropbear -R
   56 root       0:00 -sh
   57 root       0:00 ps aux
~ #
```

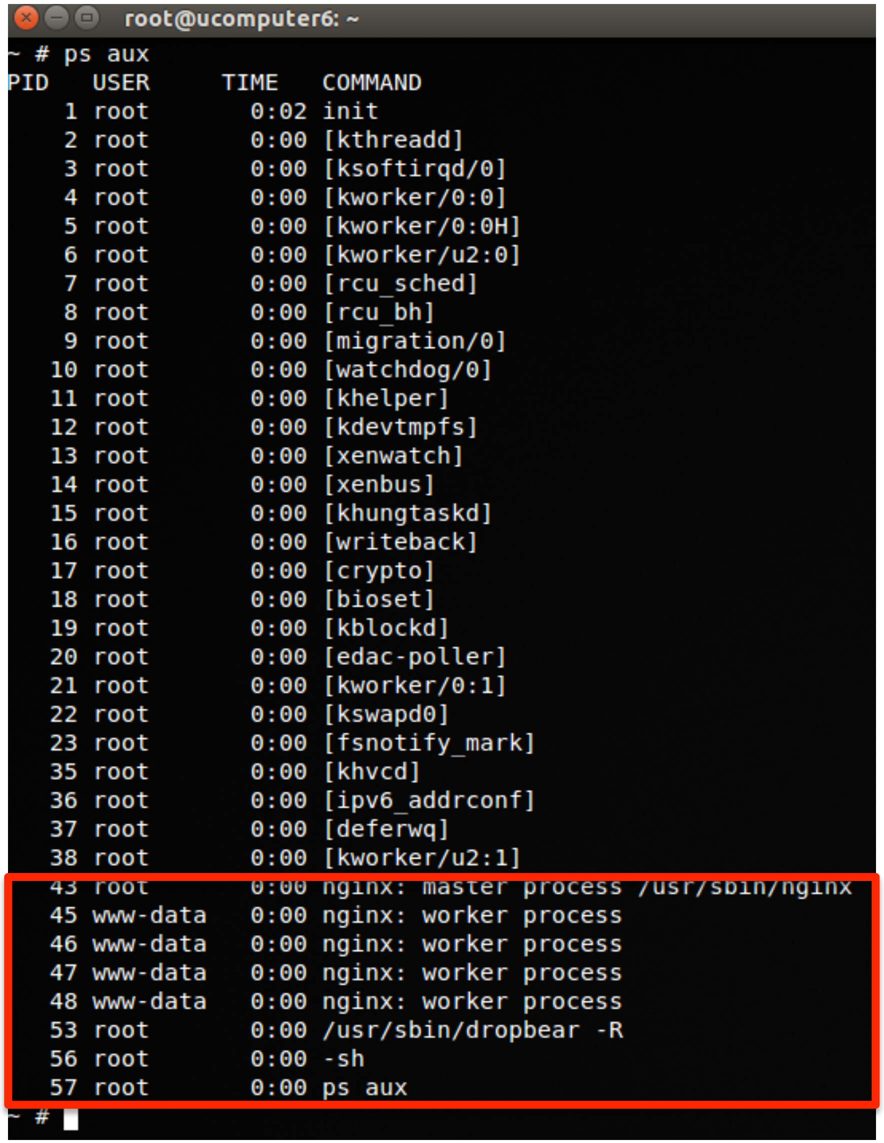
Tinyx: Taylor-made Distro




```
root@ucomputer6: ~
~ # ps aux
PID   USER      TIME   COMMAND
    1 root      0:02 init
    2 root      0:00 [kthreadd]
    3 root      0:00 [ksoftirqd/0]
    4 root      0:00 [kworker/0:0]
    5 root      0:00 [kworker/0:0H]
    6 root      0:00 [kworker/u2:0]
    7 root      0:00 [rcu_sched]
    8 root      0:00 [rcu_bh]
    9 root      0:00 [migration/0]
   10 root      0:00 [watchdog/0]
   11 root      0:00 [khelper]
   12 root      0:00 [kdevtmpfs]
   13 root      0:00 [xenwatch]
   14 root      0:00 [xenbus]
   15 root      0:00 [khungtaskd]
   16 root      0:00 [writeback]
   17 root      0:00 [crypto]
   18 root      0:00 [bioset]
   19 root      0:00 [kblockd]
   20 root      0:00 [edac-poller]
   21 root      0:00 [kworker/0:1]
   22 root      0:00 [kswapd0]
   23 root      0:00 [fsnotify_mark]
   35 root      0:00 [khvcd]
   36 root      0:00 [ipv6_addrconf]
   37 root      0:00 [deferwq]
   38 root      0:00 [kworker/u2:1]
   43 root      0:00 nginx: master process /usr/sbin/nginx
   45 www-data  0:00 nginx: worker process
   46 www-data  0:00 nginx: worker process
   47 www-data  0:00 nginx: worker process
   48 www-data  0:00 nginx: worker process
   53 root      0:00 /usr/sbin/dropbear -R
   56 root      0:00 -sh
   57 root      0:00 ps aux
~ #
```

Tinyx: Taylor-made Distro






```
~ # ps aux
PID   USER     TIME   COMMAND
    1 root       0:02 init
    2 root       0:00 [kthreadd]
    3 root       0:00 [ksoftirqd/0]
    4 root       0:00 [kworker/0:0]
    5 root       0:00 [kworker/0:0H]
    6 root       0:00 [kworker/u2:0]
    7 root       0:00 [rcu_sched]
    8 root       0:00 [rcu_bh]
    9 root       0:00 [migration/0]
   10 root       0:00 [watchdog/0]
   11 root       0:00 [khelper]
   12 root       0:00 [kdevtmpfs]
   13 root       0:00 [xenwatch]
   14 root       0:00 [xenbus]
   15 root       0:00 [khungtaskd]
   16 root       0:00 [writeback]
   17 root       0:00 [crypto]
   18 root       0:00 [bioset]
   19 root       0:00 [kblockd]
   20 root       0:00 [edac-poller]
   21 root       0:00 [kworker/0:1]
   22 root       0:00 [kswapd0]
   23 root       0:00 [fsnotify_mark]
   35 root       0:00 [khvcd]
   36 root       0:00 [ipv6_addrconf]
   37 root       0:00 [deferwq]
   38 root       0:00 [kworker/u2:1]
   43 root       0:00 nginx: master process /usr/sbin/nginx
   45 www-data   0:00 nginx: worker process
   46 www-data   0:00 nginx: worker process
   47 www-data   0:00 nginx: worker process
   48 www-data   0:00 nginx: worker process
   53 root       0:00 /usr/sbin/dropbear -R
   56 root       0:00 -sh
   57 root       0:00 ps aux
~ #
```

- Keep only the necessary bits and pieces
  - Specialized kernel build containing only the necessary modules
  - Root filesystem populated with only necessary services, libraries and 3rd party applications

Virtualization Technologies

- “Standard” VM
  - Standard Debian-based Linux VM
- “Tinyfied” VM
  - Tinyx, based on Linux kernel/busybox
- Unikernel
  - On Xen: MiniOS + miniperf
  - On KVM: OSv + iperf
- Containers
  - Docker

Virtualization Technologies

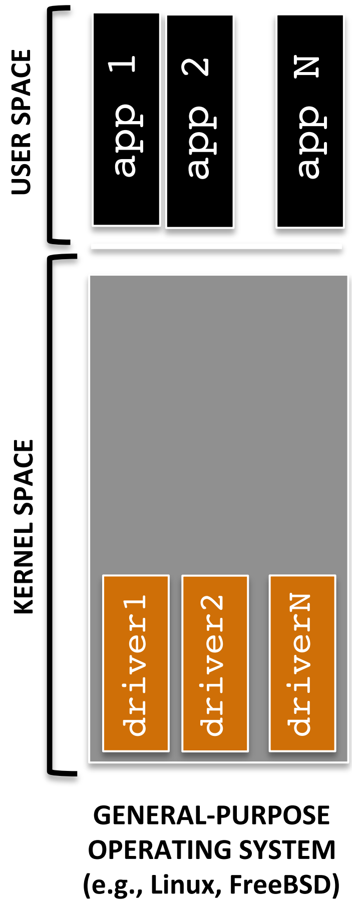
- “Standard” VM
  - Standard Debian-based Linux VM
- “Tinyfied” VM
  - Tinyx, based on Linux kernel/busybox
- Unikernel
  - On Xen: MiniOS + miniperf
  - On KVM: OSv + iperf
- Containers
  - Docker

What's a Unikernel?

- Specialized VM: single application + minimalistic OS
- Single address space, co-operative scheduler so *low* overheads

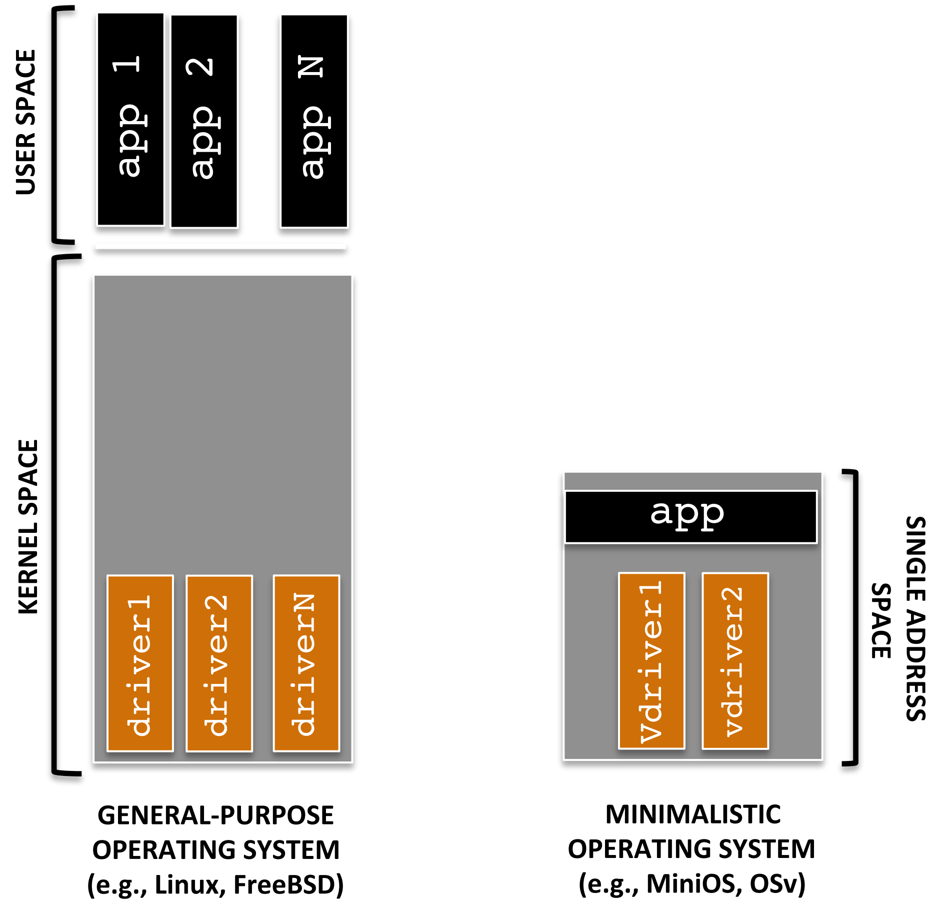
What’s a Unikernel?

- Specialized VM: single application + minimalistic OS
- Single address space, co-operative scheduler so *low* overheads

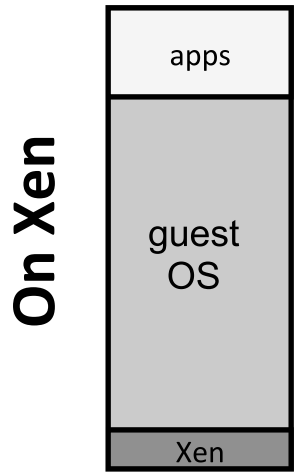


What's a Unikernel?

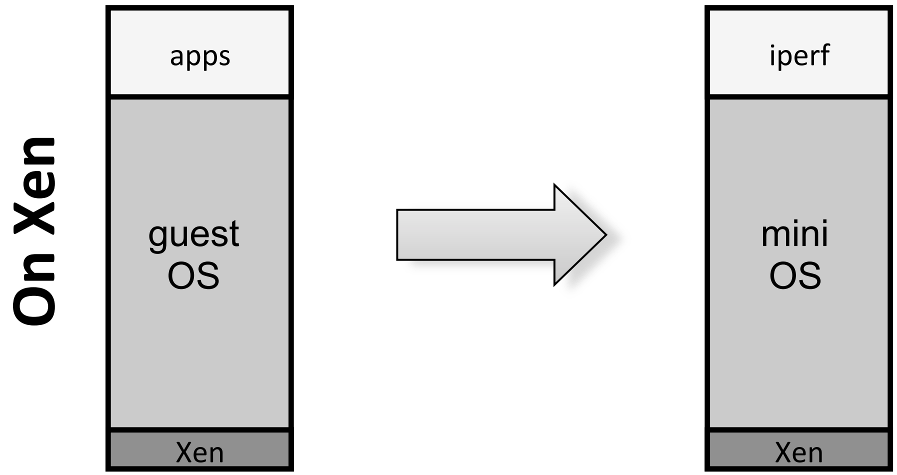
- Specialized VM: single application + minimalistic OS
- Single address space, co-operative scheduler so *low* overheads



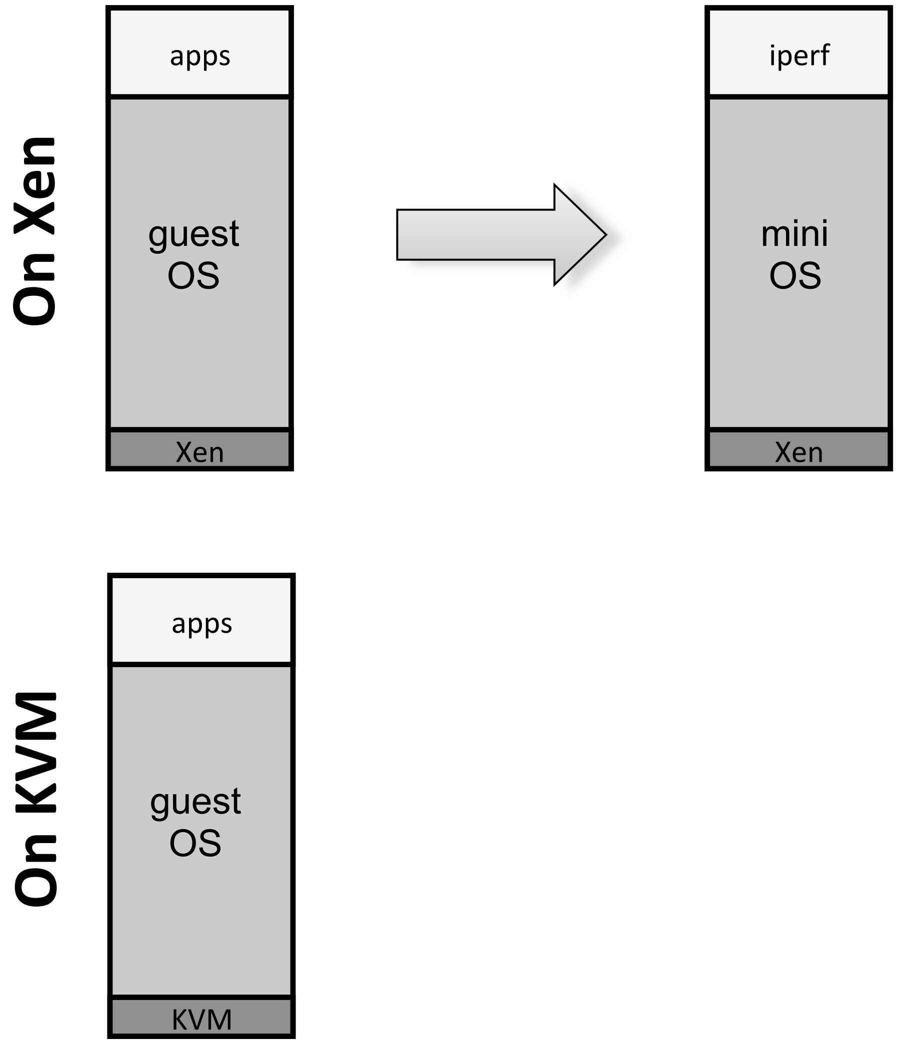
Unikernels for Benchmarking



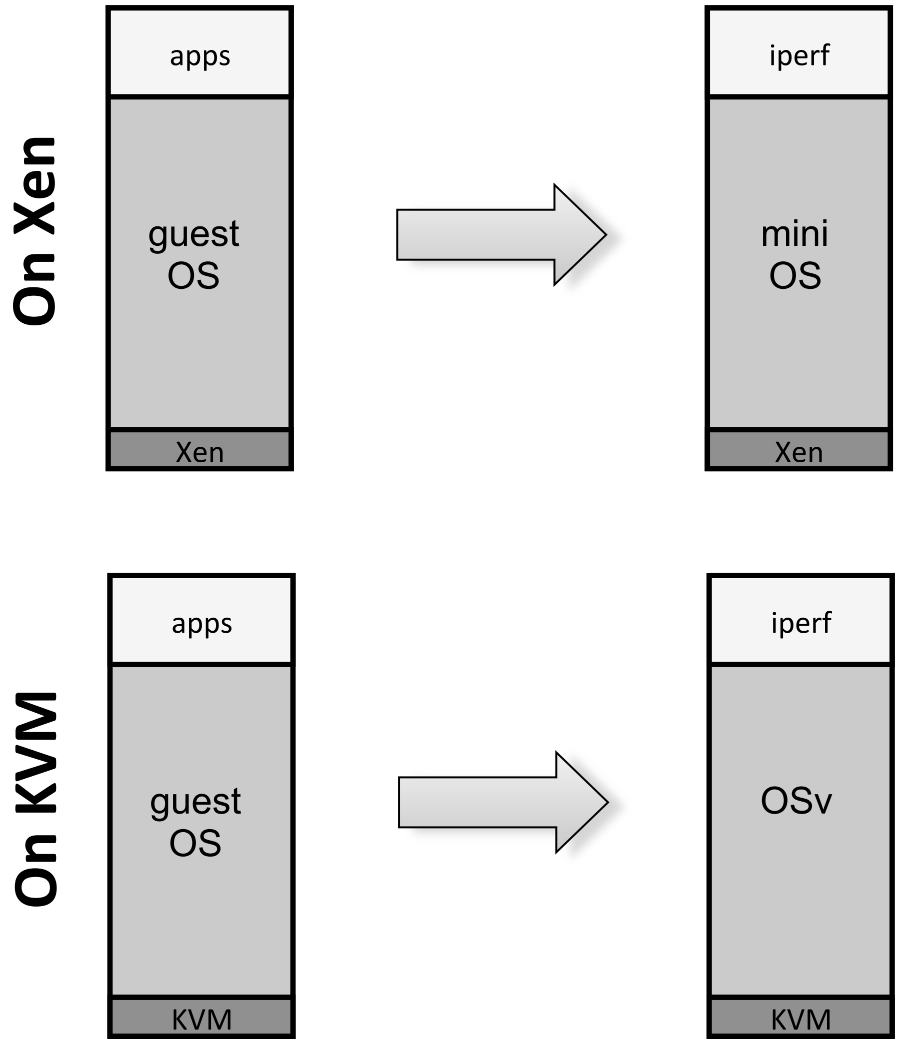
Unikernels for Benchmarking



Unikernels for Benchmarking



Unikernels for Benchmarking

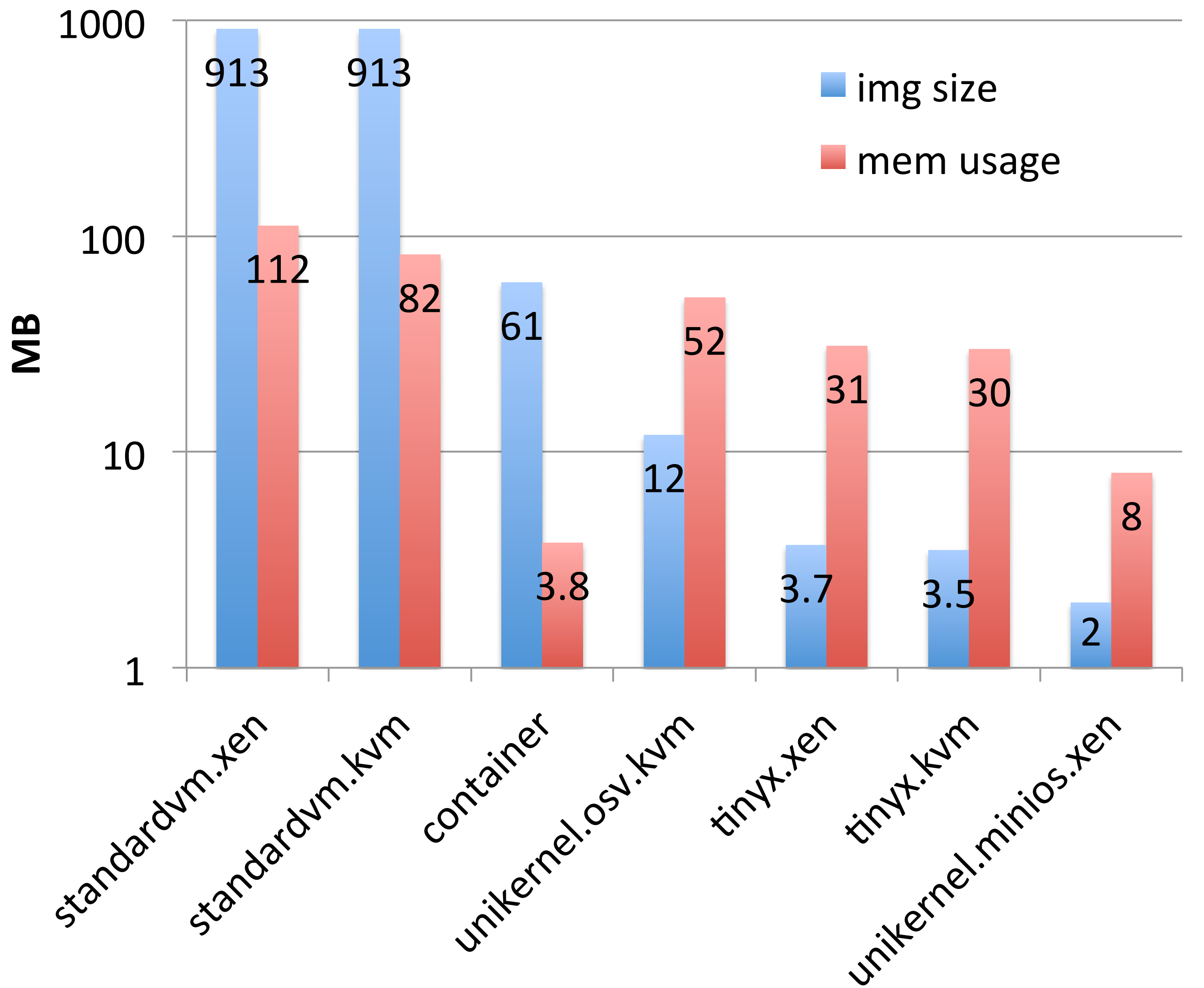


Nota Bene…

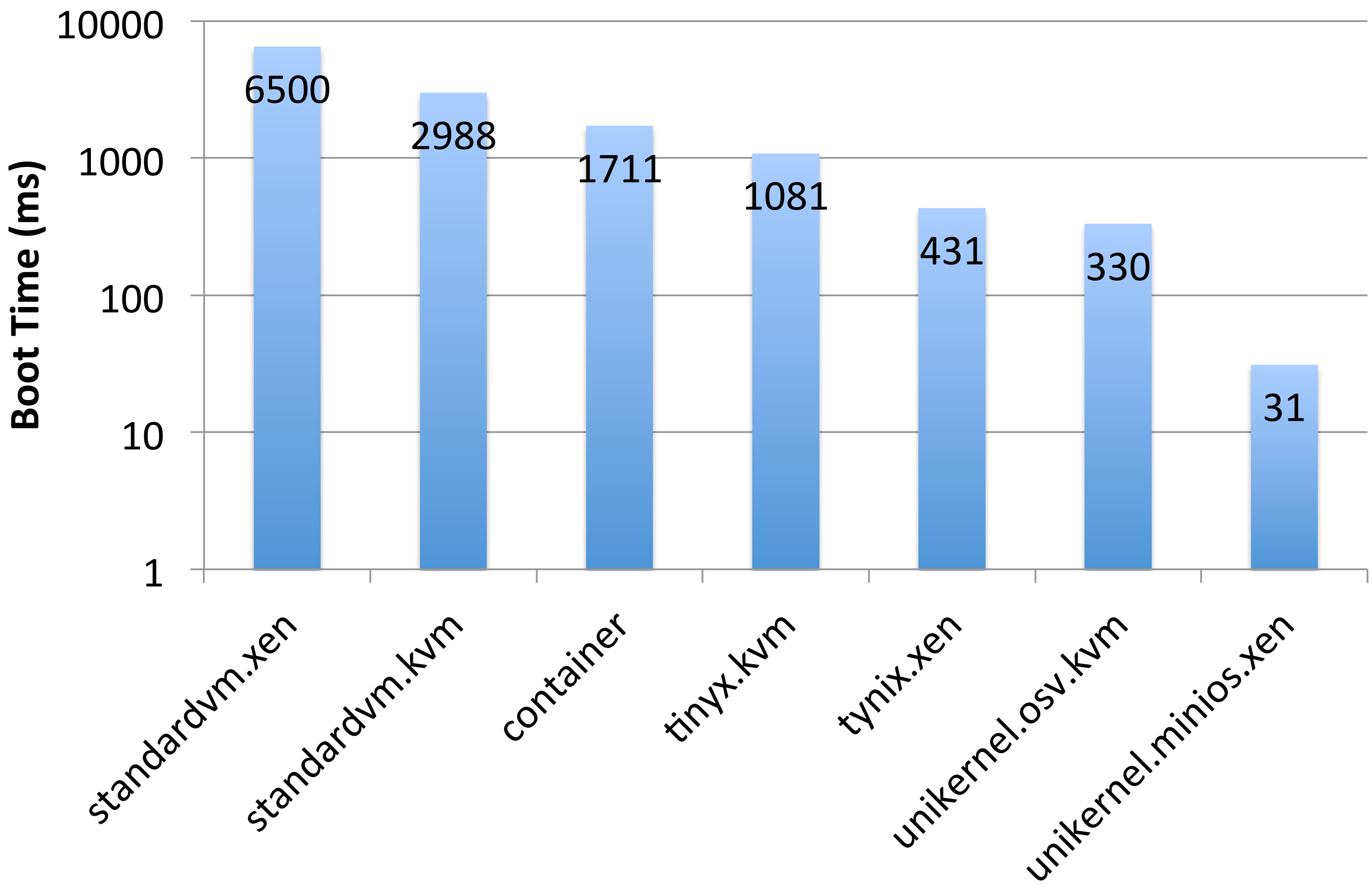
- Our unikernel numbers include optimizations to the underlying virtualization platforms (Xen, KVM)
  - Toolstacks
  - Back-end stores
  - Hotplug scripts
  - Network drivers (on Xen Tx)
- No time to go over these…

RESULTS

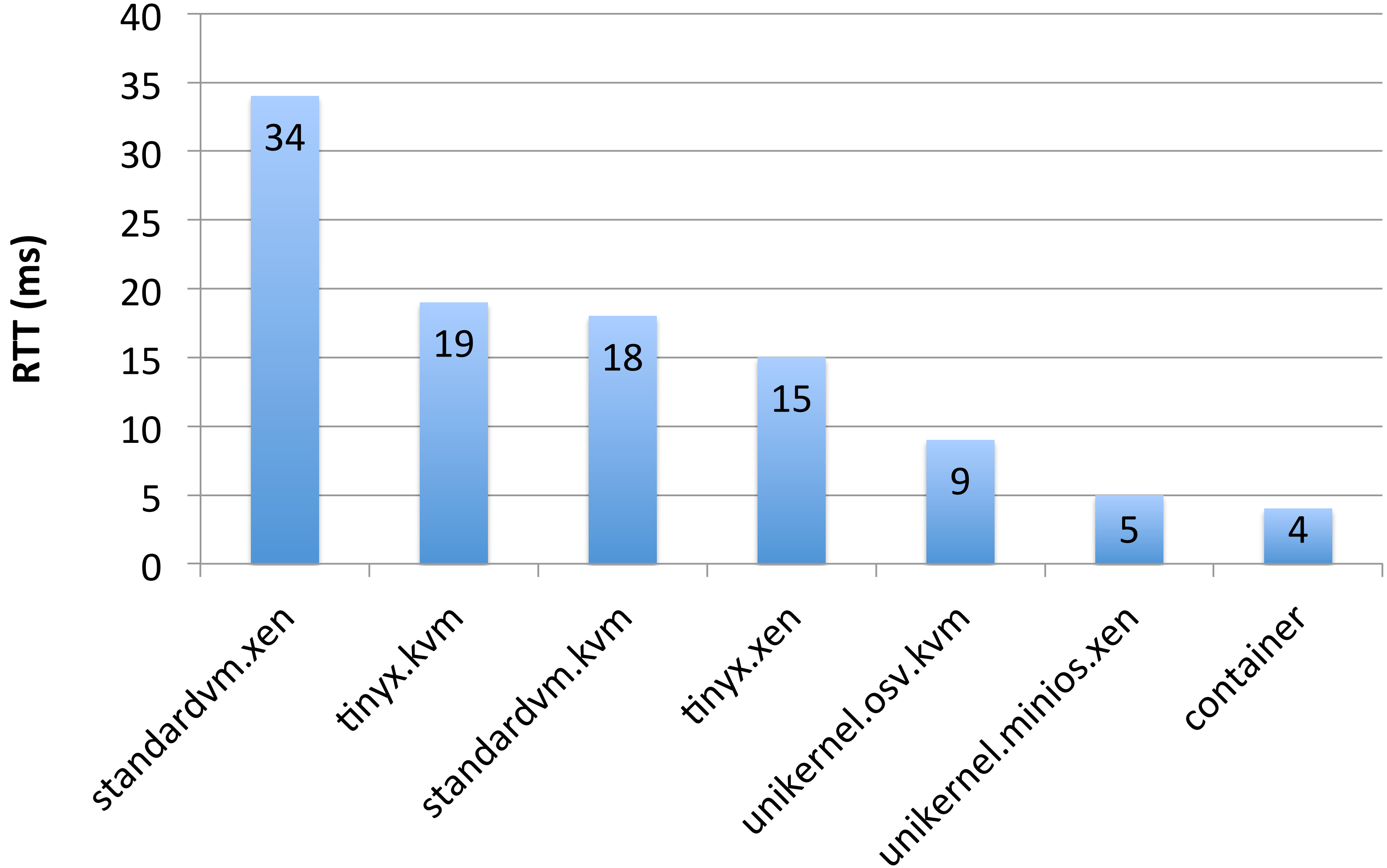
Image Size, Memory Usage (log scale)



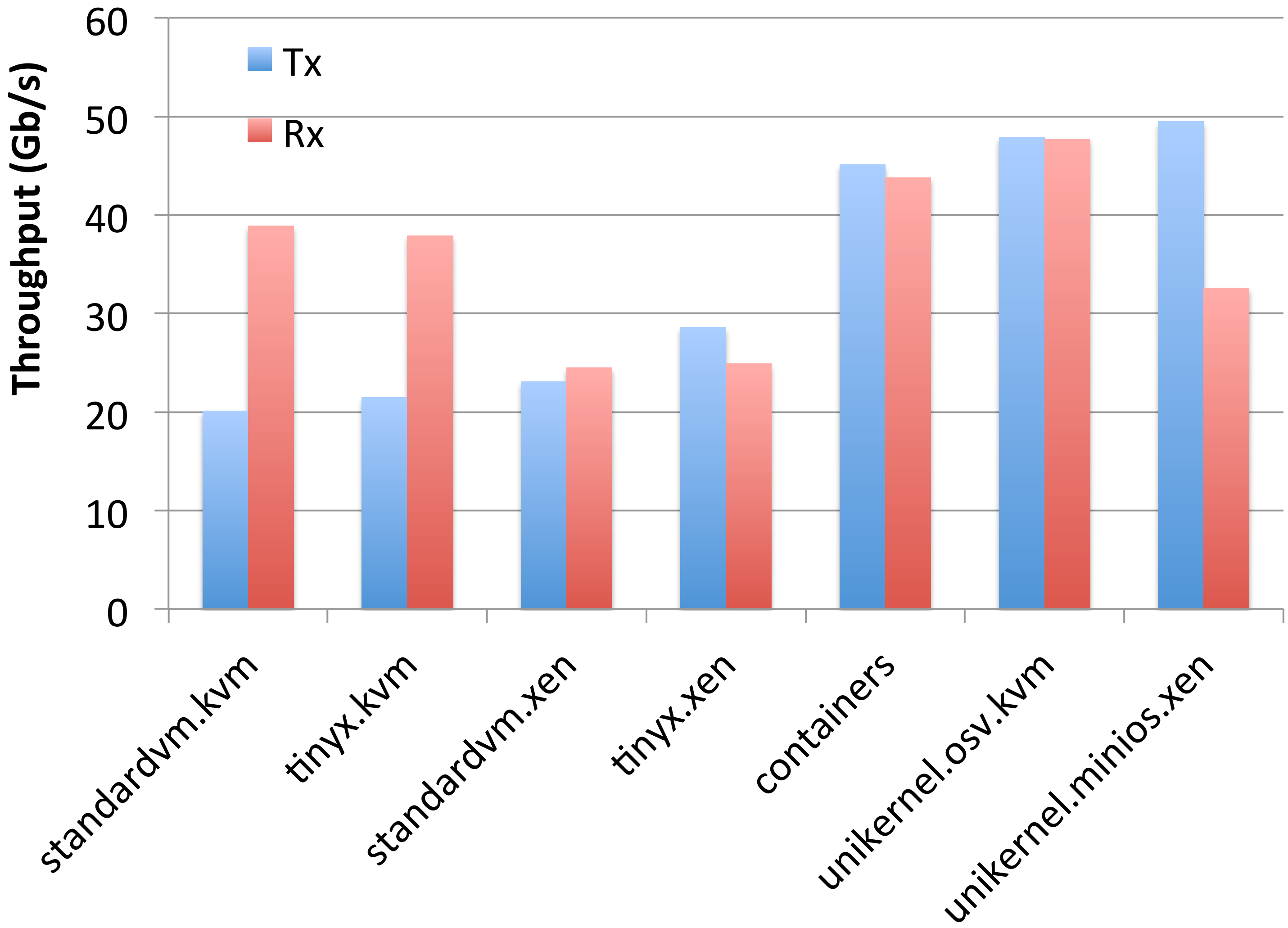
Boot Times (log scale)



RTT


RTT (ms)
40
35
30
25
20
15
10
5
0
34
19
18
15
9
5
4
standardvm.xen
tinyx.kvm
standardvm.kvm
tinyx.xen
unikernel.osv.kvm
unikernel.minios.xen
container

Throughput



Throughput (Gb/s)
60
50
40
30
20
10
0
Tx
Rx
standardvm.kvm
tinyx.kvm
standardvm.xen
tinyx.xen
containers
unikernel.osv.kvm
unikernel.minios.xen

Conclusions

- Common lore: VMs provide good isolation but are heavyweight
  - Results with standard VMs confirm this
- Containers provide lighter-weight virtualization
  - But tinyfied VMs and especially unikernels yield comparable performance

Conclusions

- Common lore: VMs provide good isolation but are heavyweight
  – Results with standard VMs confirm this
- Containers provide lighter-weight virtualization
  – But tinyfied VMs and especially unikernels yield comparable performance

Potential Contributions to draft-natarajan-nfvrg-containers-for-nfv-01

2.1.1 Challenges

- VNF provisioning time
- Runtime performance (throughput, scaling up/down)

3. Benefits of Containers

- Service agility vs VMs
- Containers have better runtime performance
- Auto-scaling of VNFs
- Cross-VNF compatibility: container unikernel/minimalistic distro
- Overall performance: VMs -25% throughput vs containers

5. Conclusion

- Containers have significant advantages vs hypervisor-based solutions