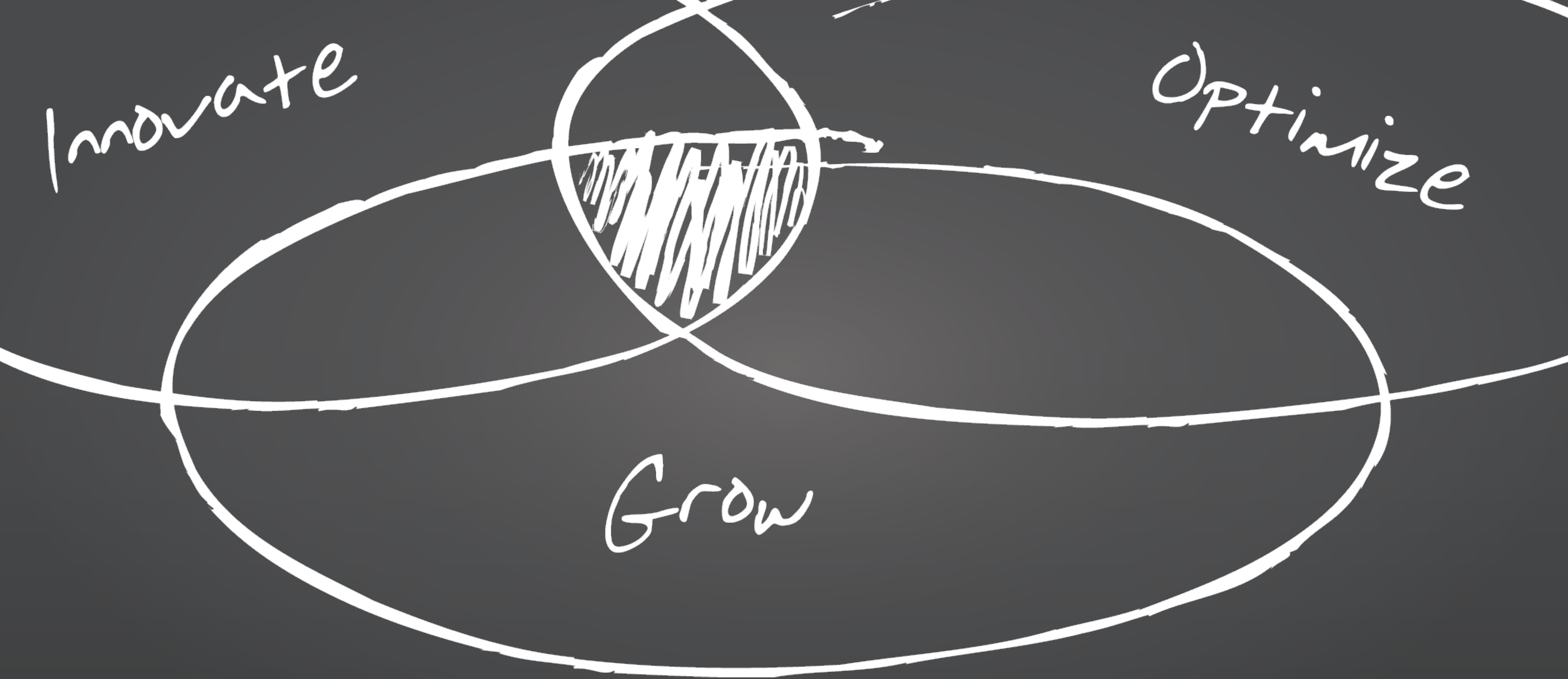
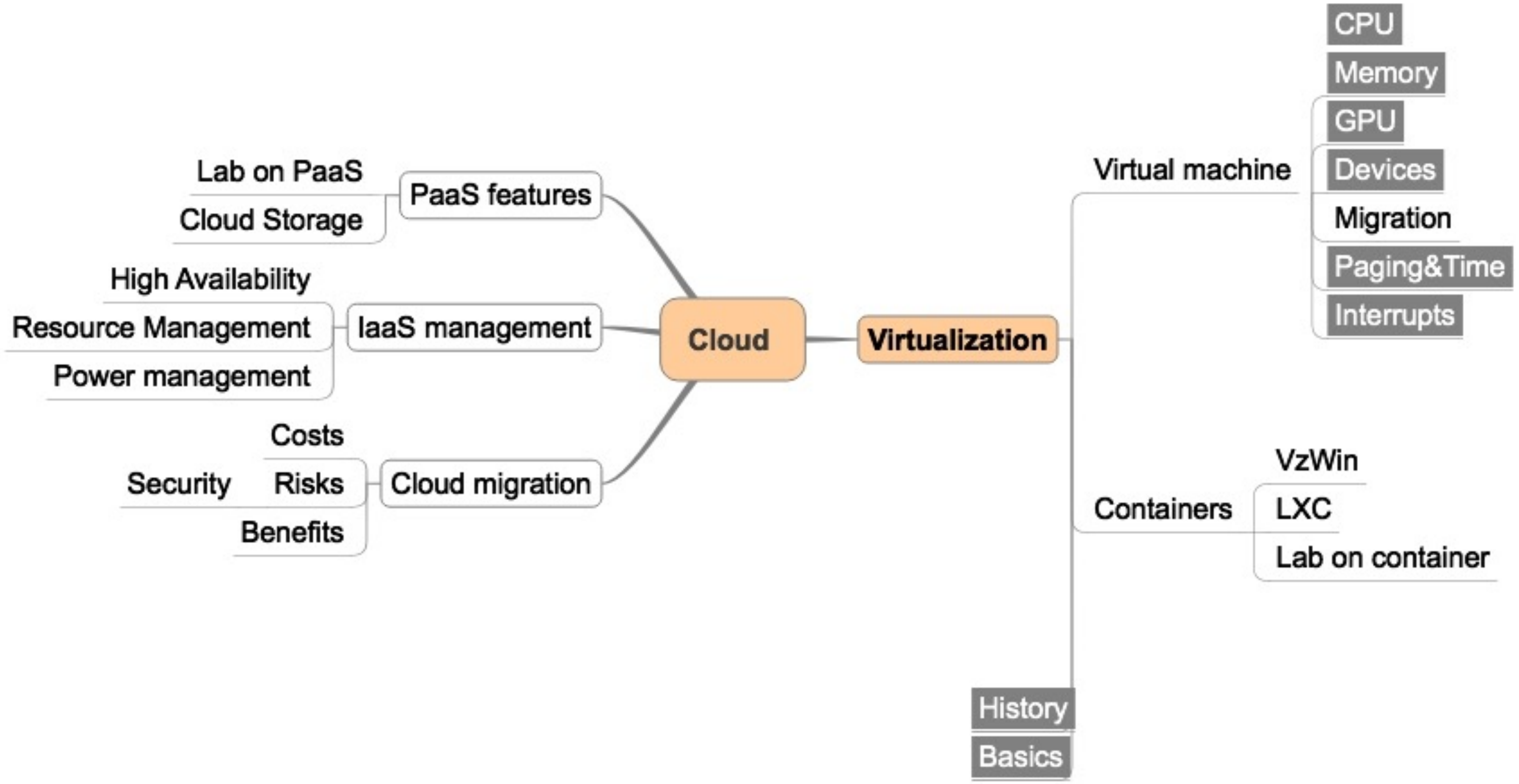


The total virtualization

Tools and migration

Lecture 9

Course overview



Contents

- Suspend/resume
  - Snapshotting
  - Live Migration case
- Architecture of the overall solution

Migration is a key step from virtualization to clouds. Tools are the great example of user experience

Suspend/resume

Suspend/resume(1)

- Complete async device request
- Guest memory
  - Store bitmap (?)
  - Flush memory
- Flush all vmm caches
- Store virtualized env state
- Store(serialize) vmm state

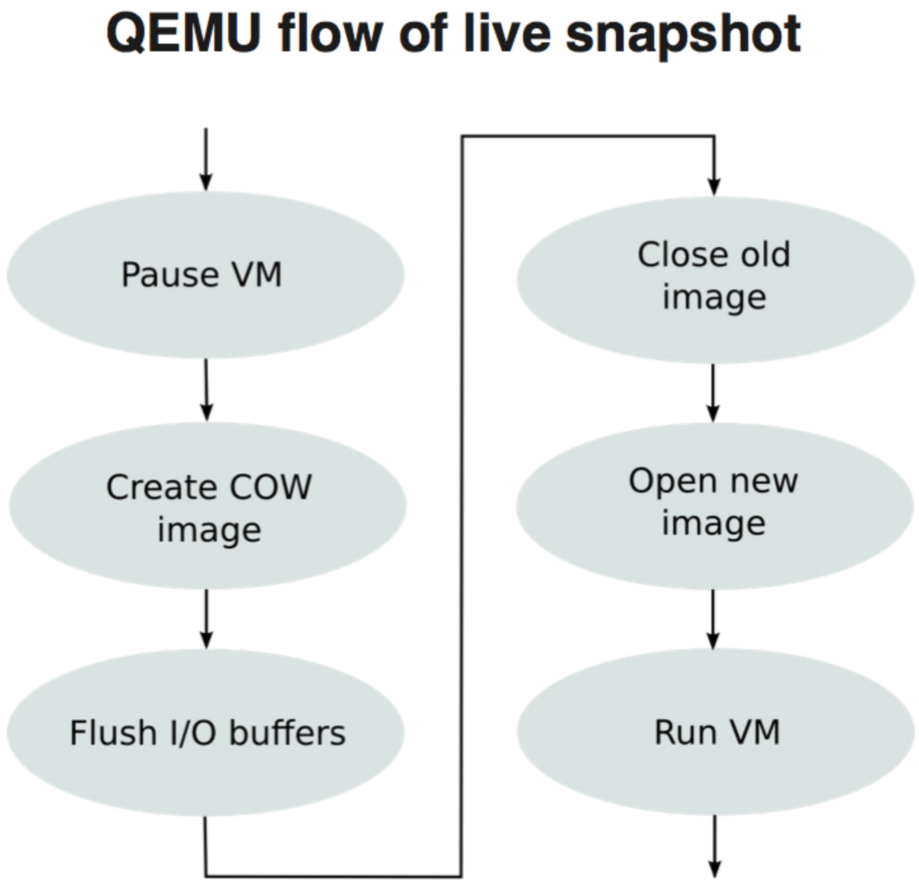
Suspend/resume (2)

- Guest memory
  - Restore active pages/all/nothing
- Restore vmm state
  - Start threads, timers, queues…
- Restore virtualized env state
  - Put guest state to fit ACPI-S3/S4

Snapshoting

What is the difference from suspend/resume?

Snapshoting: live snapshoting



Snapshoting: all about disk delta

Snapshoting: all about disk delta

- Full copy
- COW
- BTRFS

Live Migration

Live migration

Live migration

Live Migration aims

- ✓ Decrease downtime
- ✓ Decrease migration time
- ✓ Save performance after migration and during migration
- ✓ Save existing network connections

Live Migration: memory migration (1)

Push phase The source VM continues running while certain pages are pushed across the network to the new destination. To ensure consistency, pages modified during this process must be re-sent.

Stop-and-copy phase The source VM is stopped, pages are copied across to the destination VM, then the new VM is started.

Pull phase The new VM executes and, if it accesses a page that has not yet been copied, this page is faulted in (“pulled”) across the network from the source VM

Live Migration: memory migration (2)

- ✓ Stop-and-copy (pre-copy)
  - Simple, high down-time and migration time
- ✓ Demand-migration (post-copy)
  - Low downtime, high migration time, bad performance
- ✓ Working-set-copy (pre-copy)
  - Kinda optimal ?

on: common algorithm

Stage 0: Pre-Migration
- Active VM on Host A
- Alteranative physical host may be preselected to migration
- Block devices mirrored and free resources maintaines

Stage 1: Reservation
- Initialize a container on the target host

Stage 2: Iterative Pre-copy
- Enable shadow paging
- Copy dirty pages in succcessive rounds

Stage 3: Stop and copy
- Suspend VM on host A
- Generate ARP to redirect traffic to Host B

Stage 4: Commitment
- VM state on Host A is released

Stage 5: Activation
- VM starts on Host B
- Connects to local devices

VM running normally on Host A

Overhead due to copying

Downtime VM Out of Service

VM running normally on Host B

Live Migration: challenges

- Moment X: no more hopes
- Memory hacks:
  - Compressing deltas?
  - Merging disk and memory deltas

Live migration that never ends

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *ptr;
    size_t sz = 1024 * 1024 * 1024 / sizeof(int) / 2; // 512MB
    ptr = (int *) calloc(sz, sizeof(int));


    for (int i = 0;; i = i + 1) {
        if (i > sz) {
            i = 0;
            printf("One more iteration done.\n");
        }
        ptr[i] = 1;

    }
    return EXIT_SUCCESS;
}
```

The app inside guest ruins migration

credit to Fanis Kalimullin and Dilyara Altynbaeva

Live migration: code of migration cycle in qemu

```c
while (s->state == MIGRATION_STATUS_ACTIVE ||
       s->state == MIGRATION_STATUS_POSTCOPY_ACTIVE) {
    int64_t current_time;

    if (urgent || !qemu_file_rate_limit(s->to_dst_file)) {
        MigIterateState iter_state = migration_iteration_run(s);
        if (iter_state == MIG_ITERATE_SKIP) {
            continue;
        } else if (iter_state == MIG_ITERATE_BREAK) {
            break;
        }
    }

    /*
     * Try to detect any kind of failures, and see whether we
     * should stop the migration now.
     */
    thr_error = migration_detect_error(s);
    if (thr_error == MIG_THR_ERR_FATAL) {
        /* Stop migration */
        break;
    } else if (thr_error == MIG_THR_ERR_RECOVERED) {
        /*
         * Just recovered from a e.g. network failure, reset all
         * the local variables. This is important to avoid
         * breaking transferred_bytes and bandwidth calculation
         */
        s->iteration_start_time = qemu_clock_get_ms(QEMU_CLOCK_REALTIME);
        s->iteration_initial_bytes = 0;
    }

    current_time = qemu_clock_get_ms(QEMU_CLOCK_REALTIME);

    migration_update_counters(s, current_time);

    urgent = false;
```

goes to ram_save_iterate

can't detect migration

Live migration: code of ram_save_iterate

```
while ((ret = qemu_file_rate_limit(f)) == 0 ||
        !QSIMPLEQ_EMPTY(&rs->src_page_requests)) {
    int pages;

    if (qemu_file_get_error(f)) {
        break;
    }

    pages = ram_find_and_save_block(rs, false);
    /* no more pages to sent */
    if (pages == 0) {
        done = 1;
        break;
    }
    rs->iterations++;

    /* we want to check in the 1st loop, just in case it was the 1st time
       and we had to sync the dirty bitmap.
       qemu_get_clock_ns() is a bit expensive, so we only check each some
       iterations
    */
    if ((i & 63) == 0) {
        uint64_t t1 = (qemu_clock_get_ns(QEMU_CLOCK_REALTIME) - t0) / 1000000;
        if (t1 > MAX_WAIT) {
            trace_ram_save_iterate_big_wait(t1, i);
            break;
        }
    }
    i++;
}
```

while loop will not break here

Migration: incompatible processors

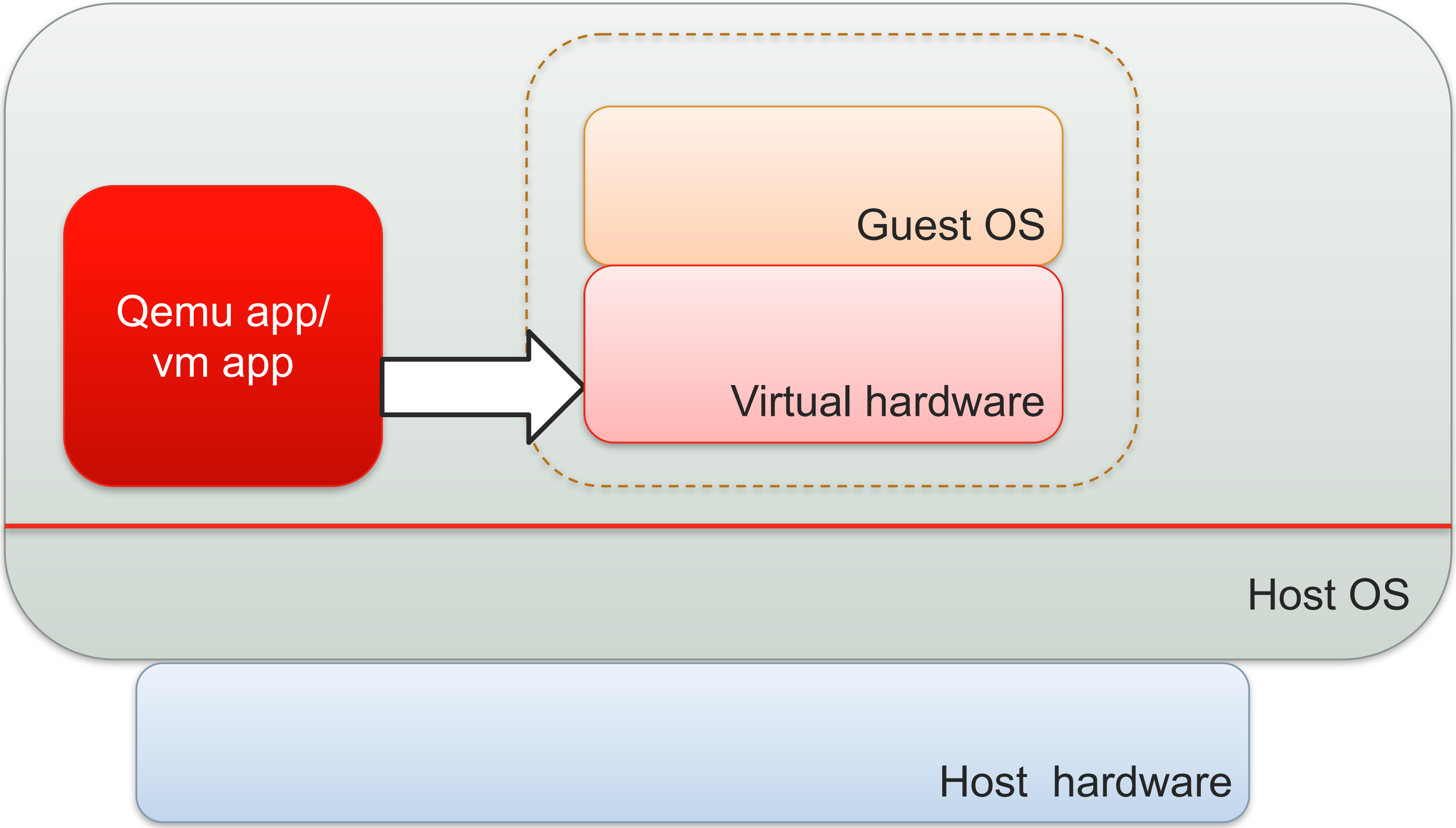
```
intel64@intel64-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Cor
stepping        : 7
cpu MHz         : 2455.716
cache size      : 6144 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 syscall nx rdtscp lm constant_tsc rep_good n
opl pni monitor ssse3 lahf_lm
bogomips        : 4911.43
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
```

```
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm c
onstant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf
 eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm p
cid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand lah
f_lm ida arat epb xsaveopt pln pts dtherm tpr_shadow vnmi flexpriority ept vpid
fsgsbase smep erms
bogomips        : 4190.56
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

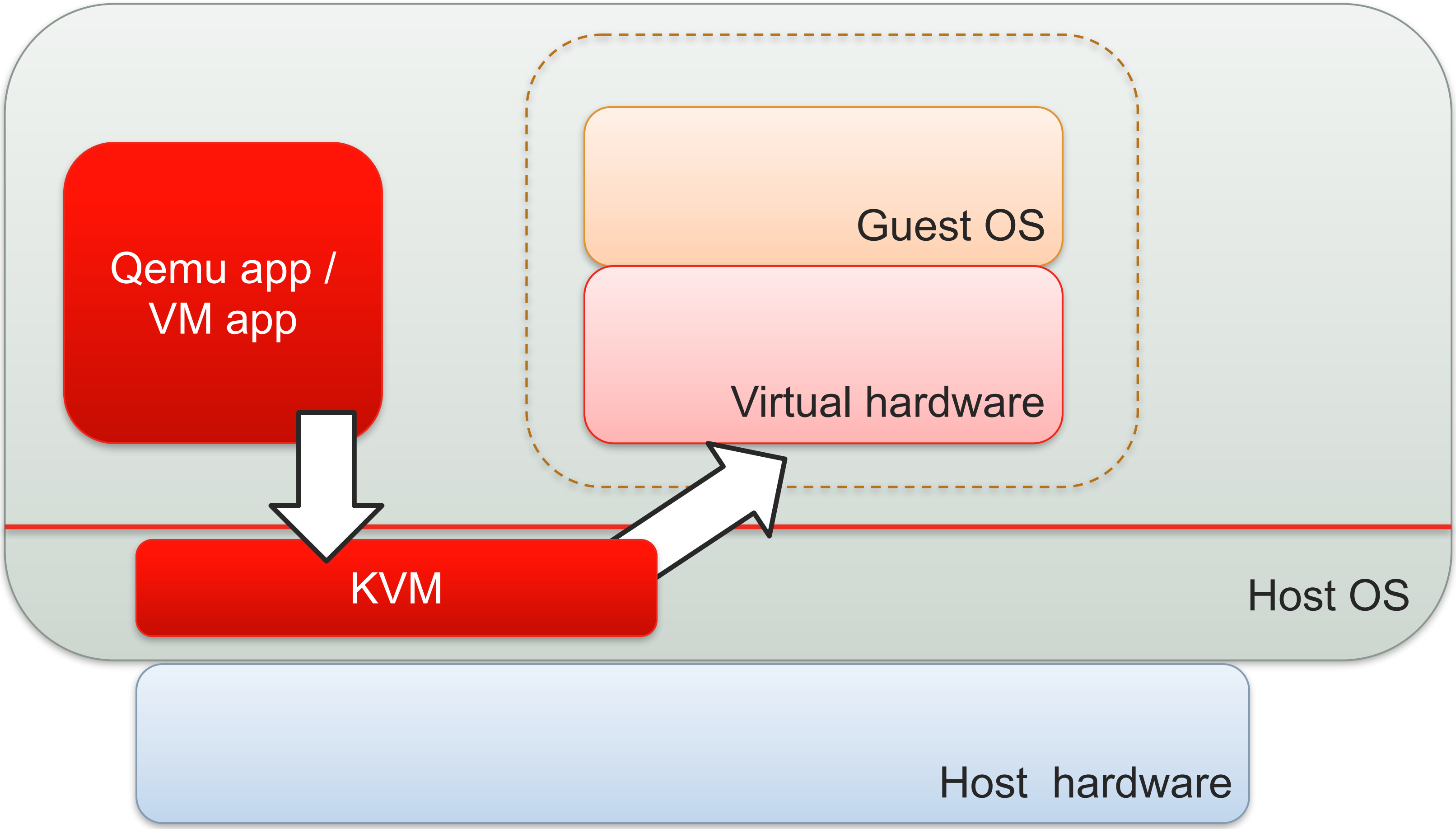
Virtual machine components

- ✓ VMM creates a virtual hardware
- ✓ De-privileged guest OS runs on a virtual hardware
- ✓ Hypervisor shares resources between few virtual hardware
- ✓ Some UI processes the user’s commands and pass them to some service that interacts with VMM/hypervisor
- ✓ Paravirtualization module runs in the guest OS

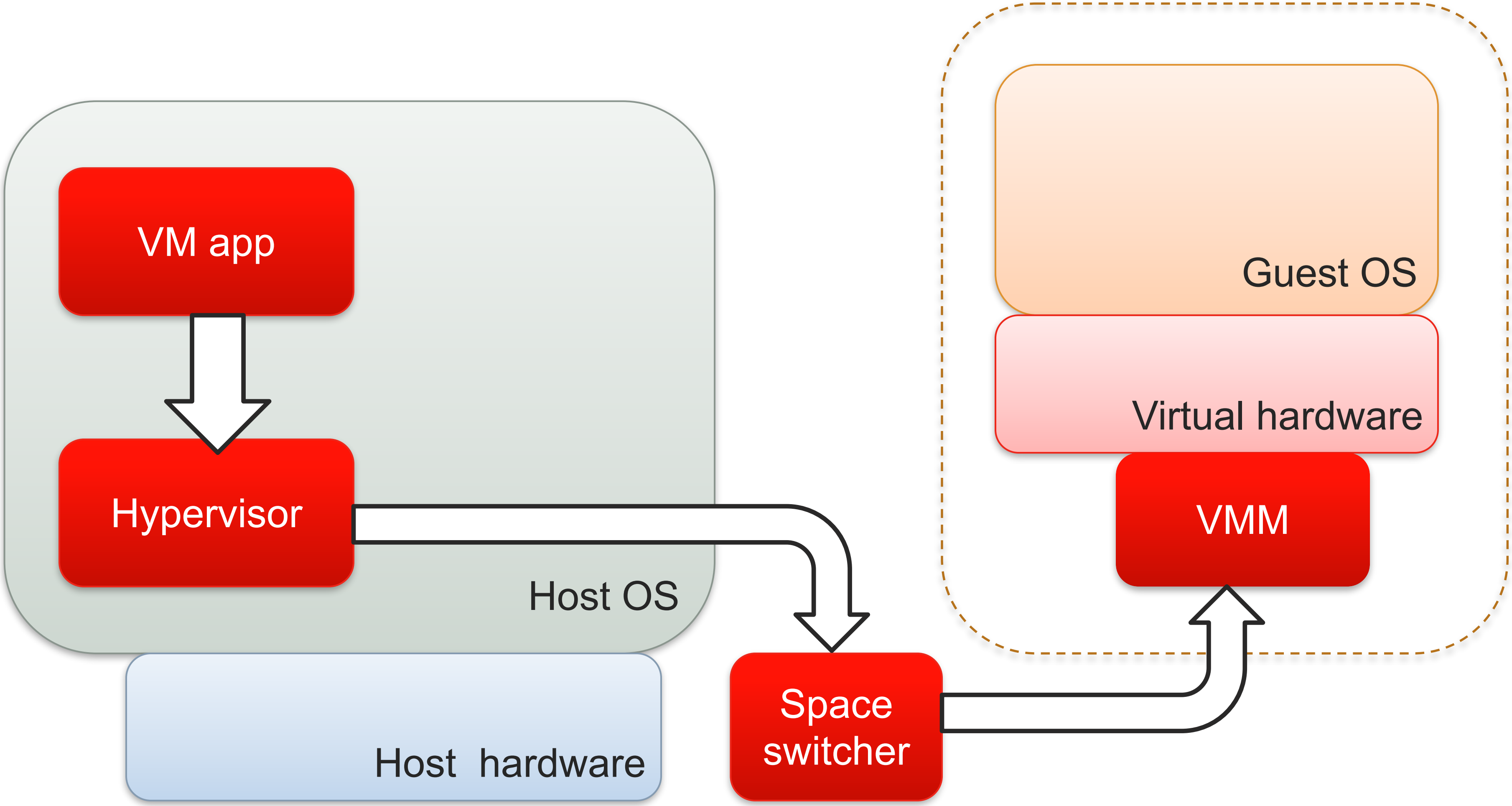
Architecture: qemu



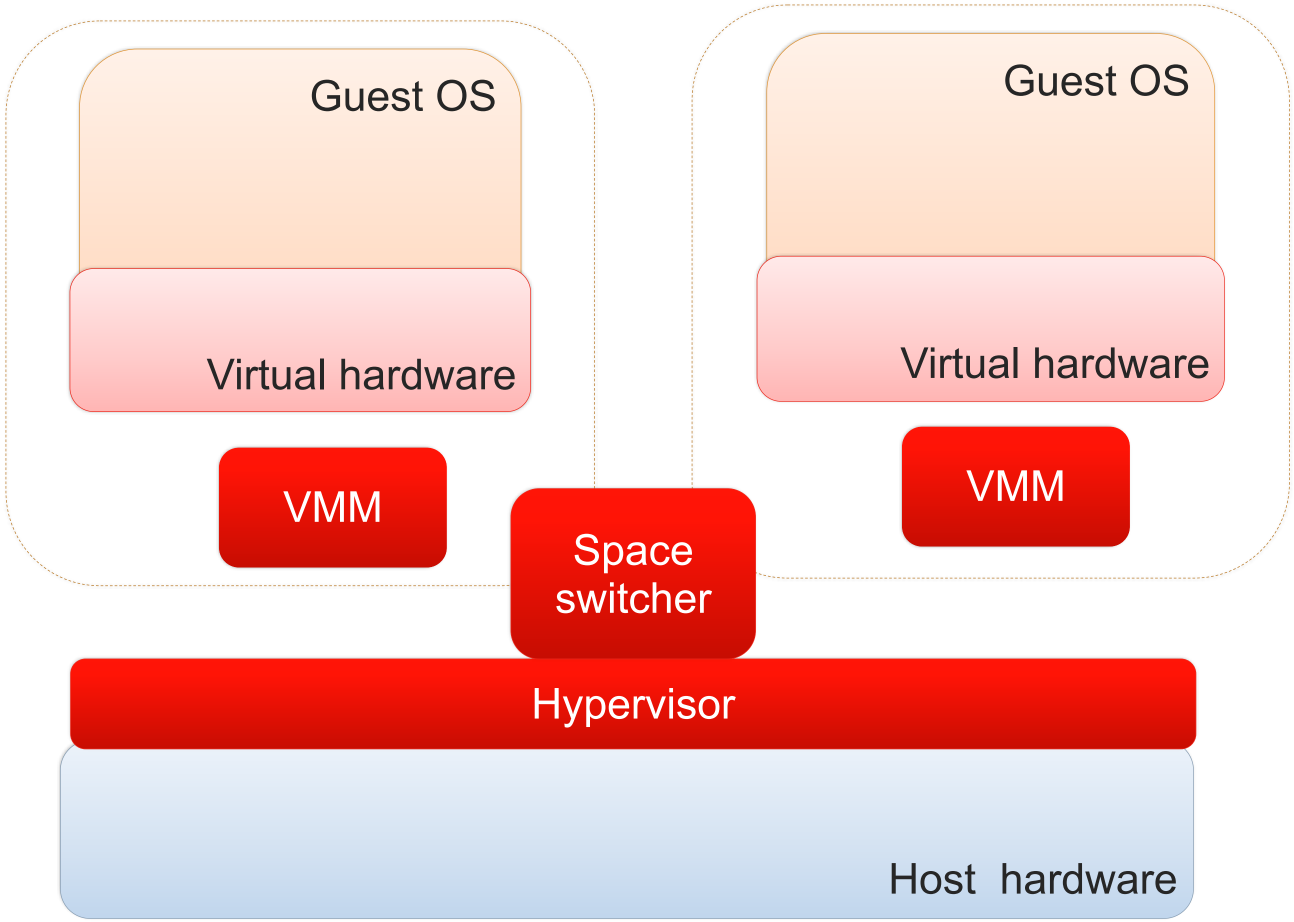
Architecture: qemu + kvm



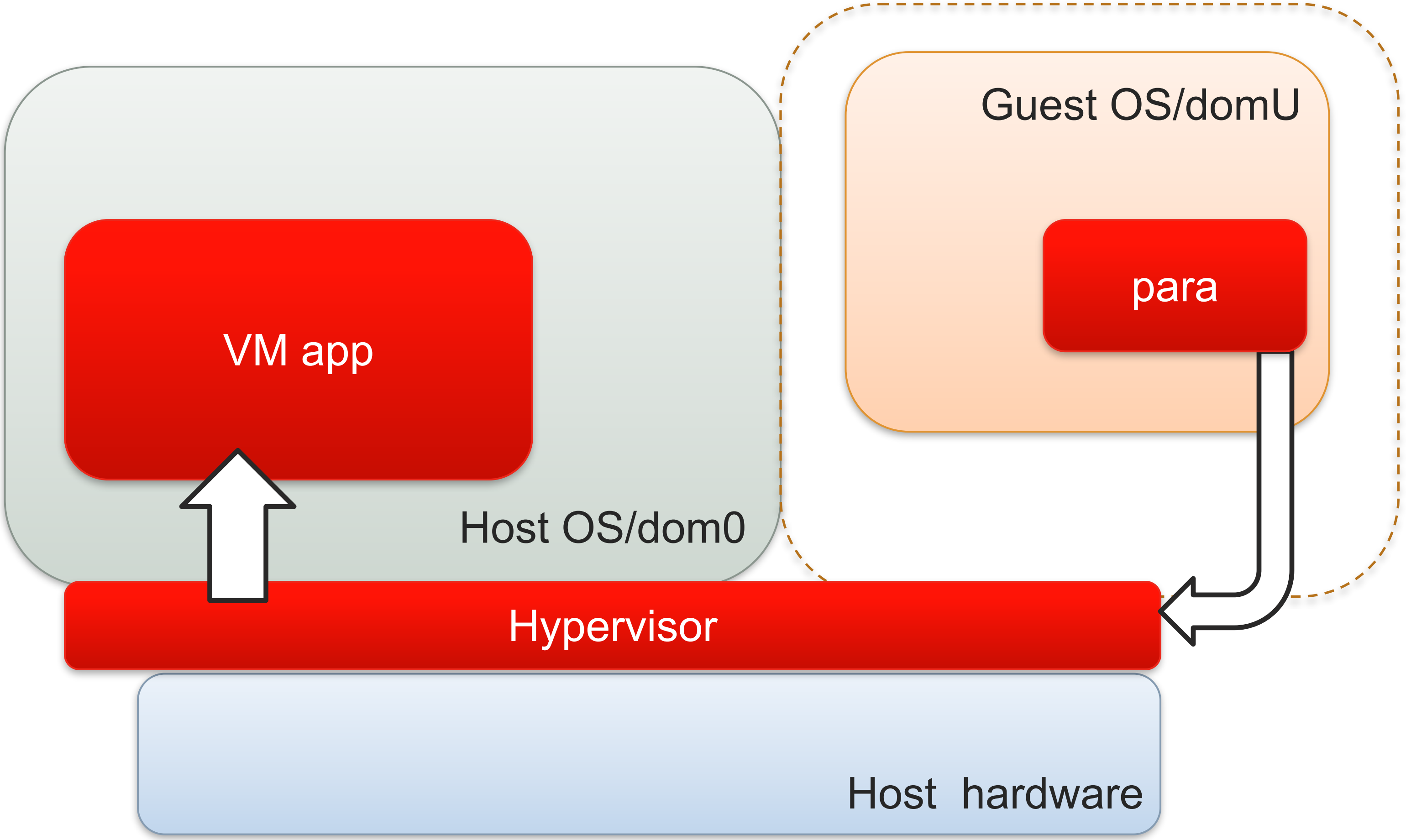
Architecture: Paralles, VMware, VBox



Architecture: ESXi (bare-metal)



Architecture: Xen, HyperV



Quality attribute for virtual machines?

Architecture criteria (QA)

- Performance
  - Context switching (guest vs vmm vs host): which of contexts could be merged, how often context switching occur
  - Access to devices
  - Access to privilege CPU extensions
  - Access to host OS paging, memory management structures, timing
- Deployment
  - Minimal privilege required
- Density
  - Access to host OS paging, memory management structures, timing
  - Access to privilege CPU extensions

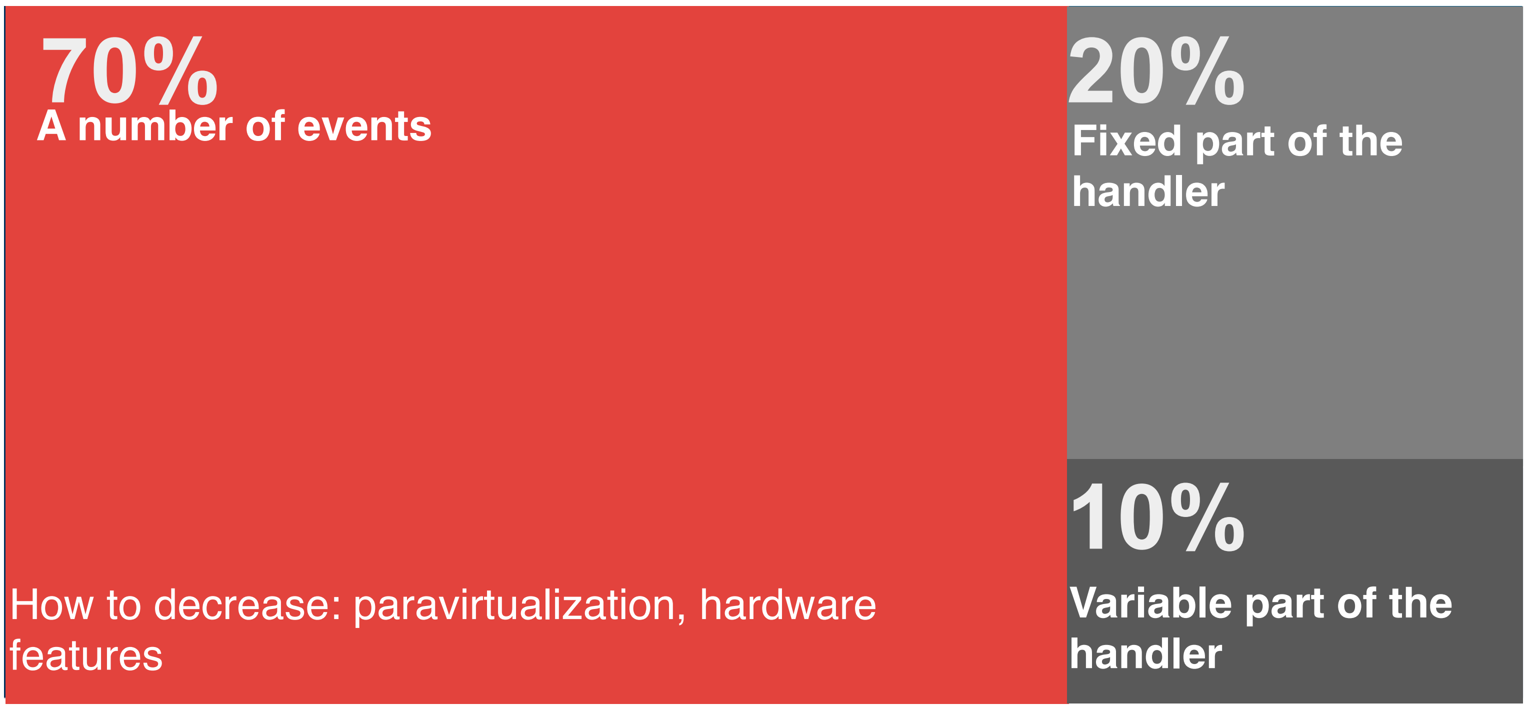
Architecture criteria (2)

- Portability
  - Ease of porting to a new CPU arch
  - Ease of porting to another OS
- Maintainability
  - Ease of debugging and re-using 3rd party tools
  - Ease of classifying different kinds of failures

Architecture analysis: performance

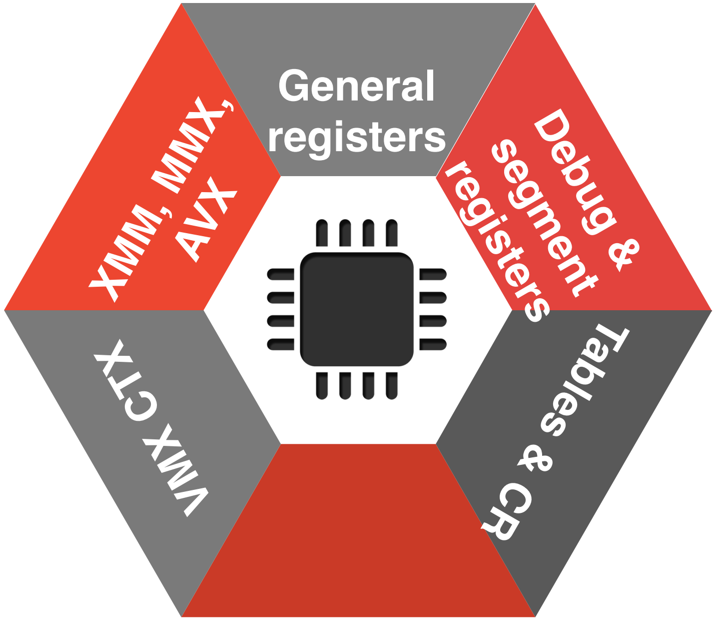
Architecture analysis: performance

How to decrease: Decrease context size



How to decrease: caches, lazy writes, groupings, heuristics

Context



General
RAX - RDI, R8-R15, flags, RIP

Control regs
CR3, CR4, CR2, CR0, CR8, pahging reload, IDT, GDT, etc

Debug and segment registers
DR0..DR7, common segment regs, FS/GS base

Media registers
FPU, XMM, MMX, AVX registers.

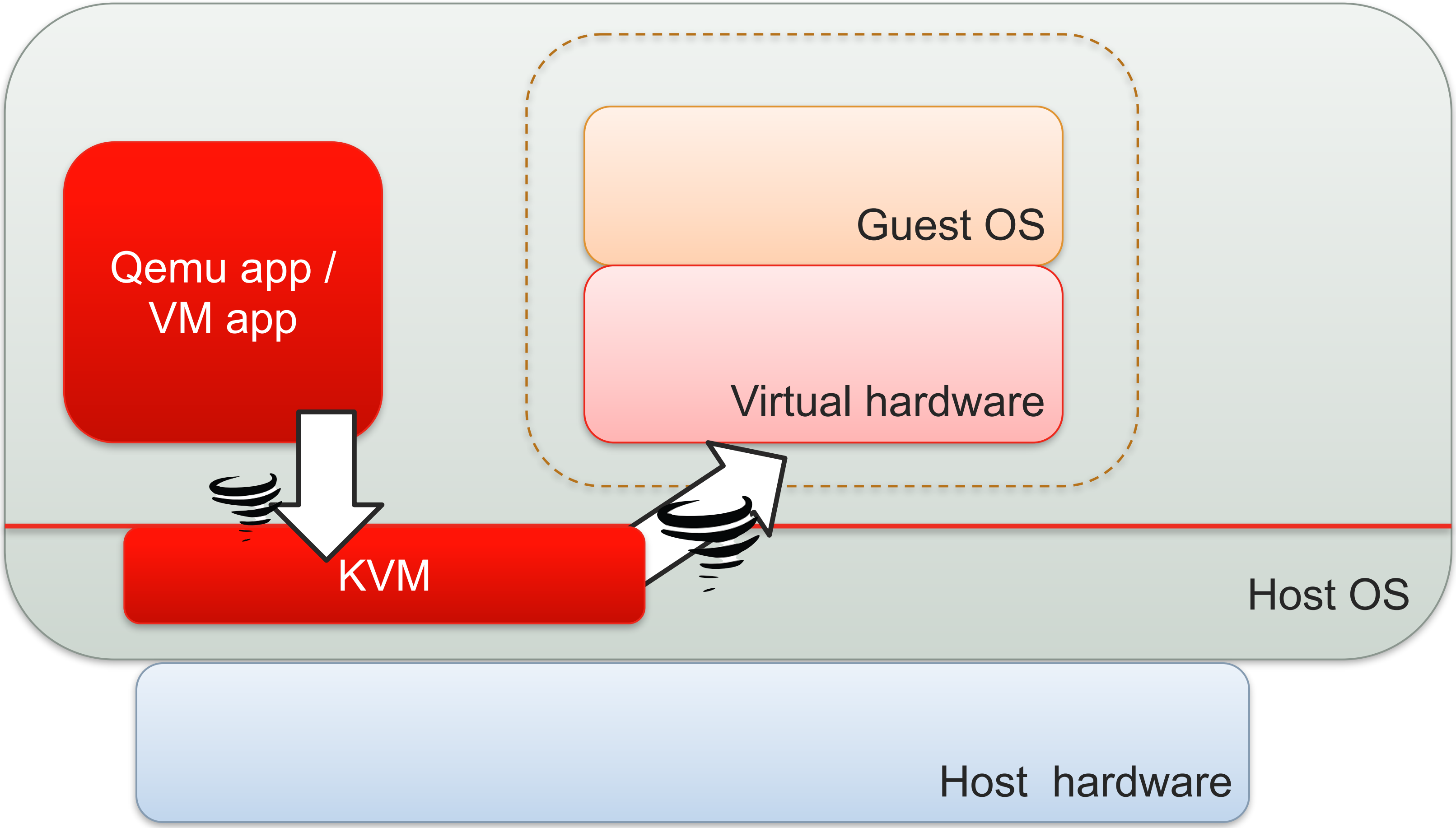
VMX ctx
VMCS reload. A few instructions but heavy ones

MSR
Many-many of them for different technologies

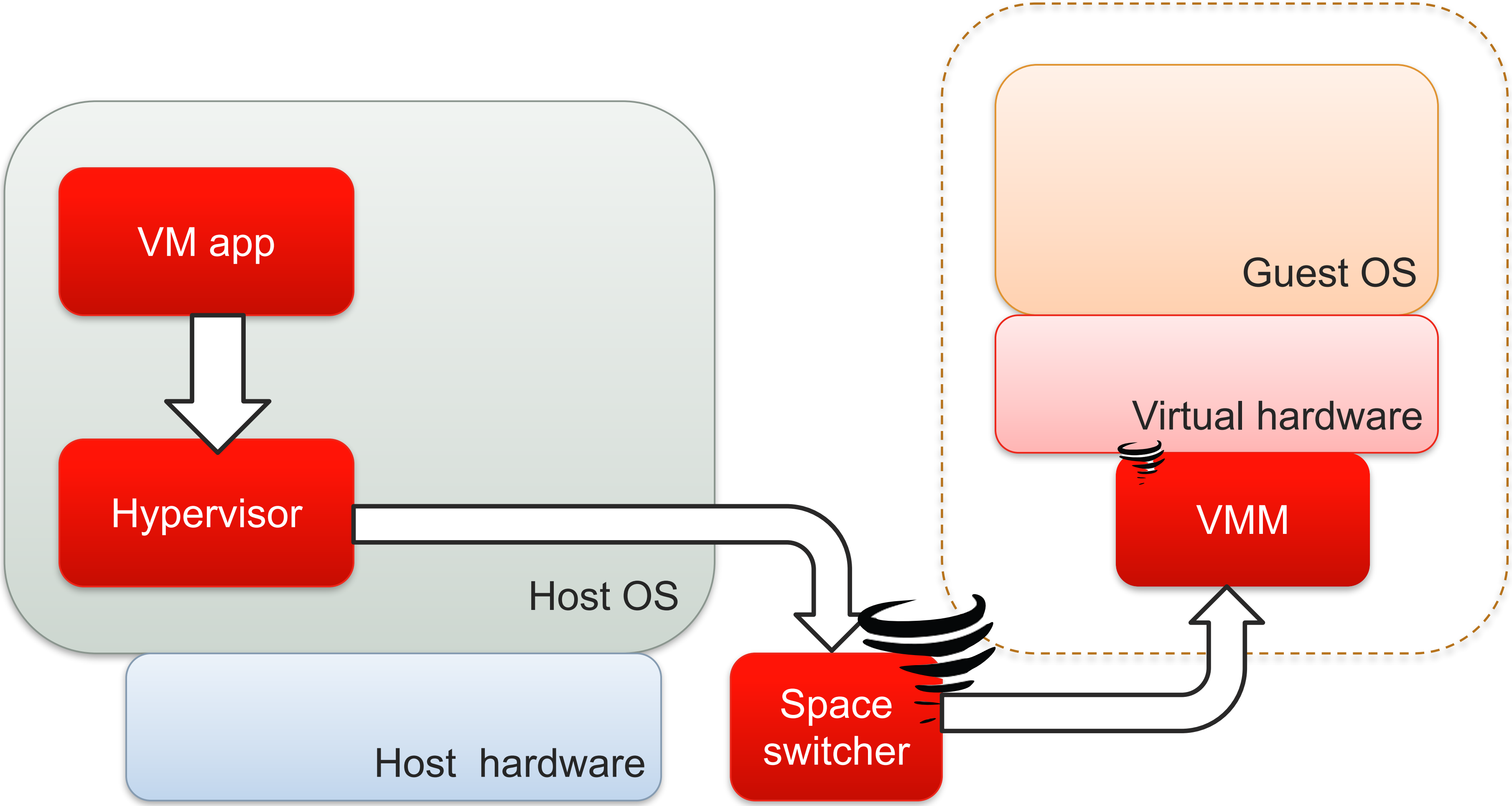
Full context switch

Just don’t do it!

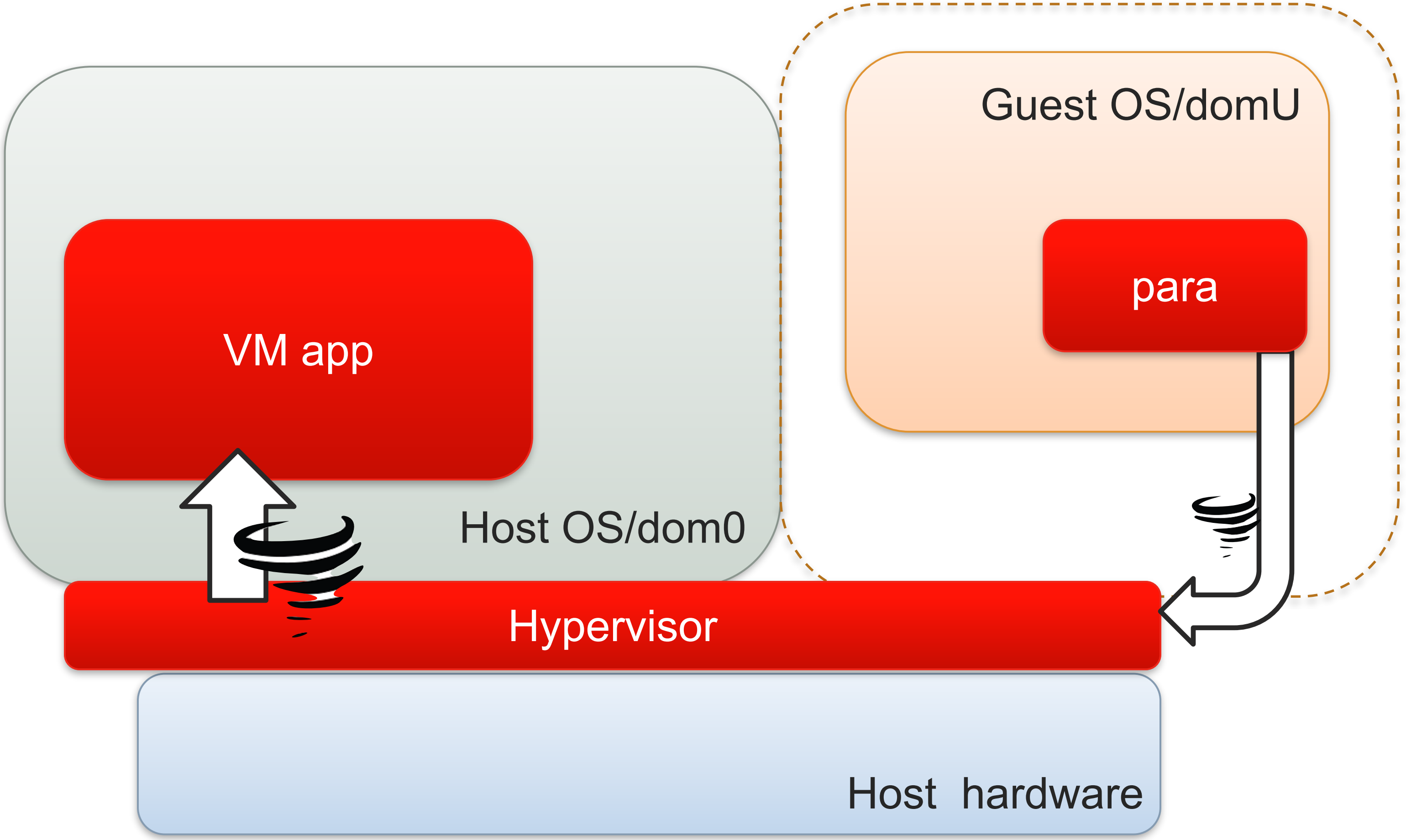
Context: qemu + kvm



Context: Paralles, VMware, VBox



Context: Xen, HyperV



Architecture analysis

Architecture

- ✓ QEMU is easy to deploy/debug and it's portable, but not that fast
- ✓ QEMU+KVM has good performance and high density, but on some workloads may be slow because of context switching. Deploy is not that ease. Not very portable.
- ✓ HyperV/Xen has very high performance and very high density, but they are not portable (between OS), complicated to debug, manual deploy is almost impossible
- ✓ PD/VMware/VBox are portable (between host OS), has good performance but they have density and debug problem (because of isolated own context)

Conclusions

Each architecture has scenario with limited performance. Live migration is must-have for a server virtualization product, though the stability of the feature is not the best one.

Questions

Projects

Project. Live migration (task for 2)

Perform live migration between 2 your notebooks. Make the load hard enough to fail the migration