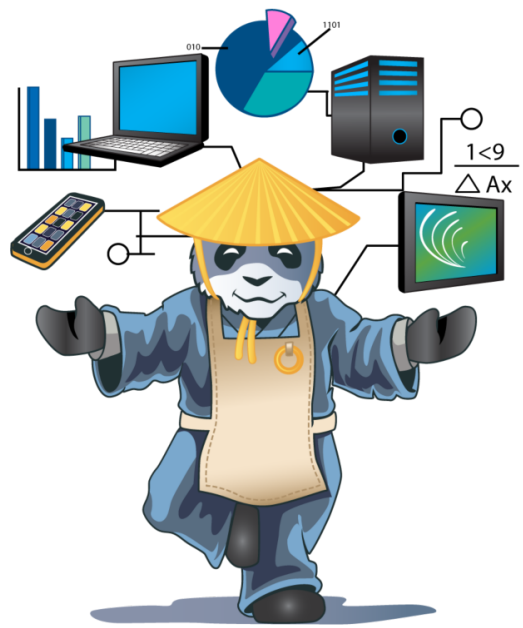
Towards a configurable and slimmer x86 hypervisor

Liu Wei

Budapest – July 11-13, 2017



Xen Project

Current state of affairs

- PV mode: no hardware extension needed, used in legacy systems, useful in certain cases like running unikernel and nested-virt without vVMX or vSVM
- HVM mode: needs hardware support and QEMU for emulation, has become the mainstream Xen VM mode
- PVH mode: essentially HVM without QEMU, under development

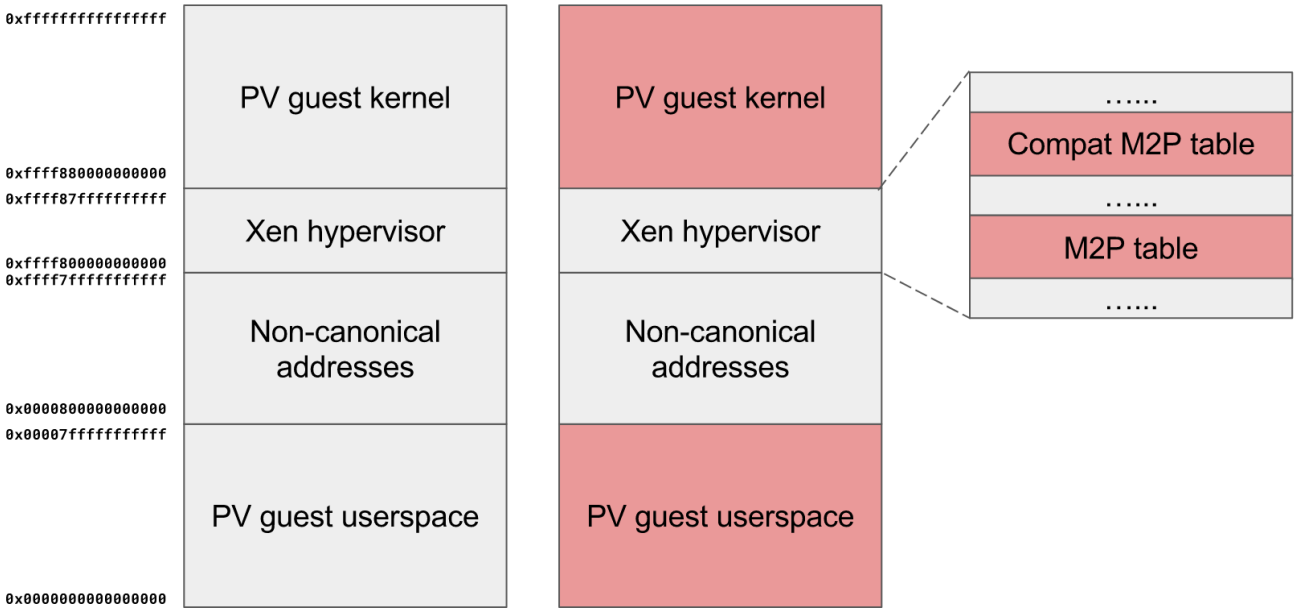
Two (big) projects

- Splitting PV and HVM code: Refactor x86 hypervisor code. Make guest supporting code configurable via Kconfig
- PV ABI in PVH container: Implement a PV ABI shim. Use it to translate PV hypercalls into PVH ones when necessary

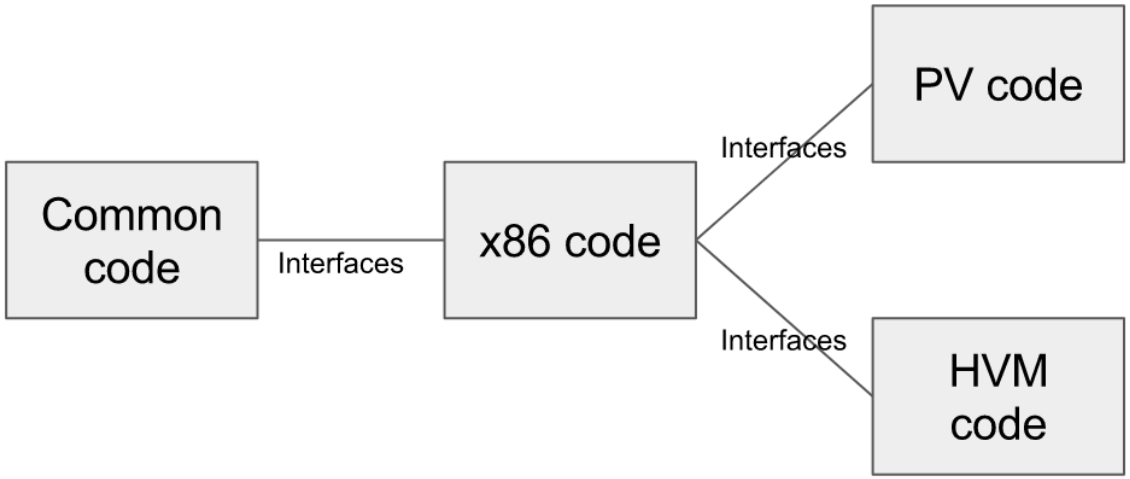
Why splitting PV and HVM code?

- Users can pick and choose the guest interfaces
  - Smaller binary, smaller attack surface
  - Reclaim precious address space if PV is disabled, to let Xen support >16TB host memory more easily
- Improve x86 hypervisor code base
- *NOT* intending to kill PV in the hypervisor

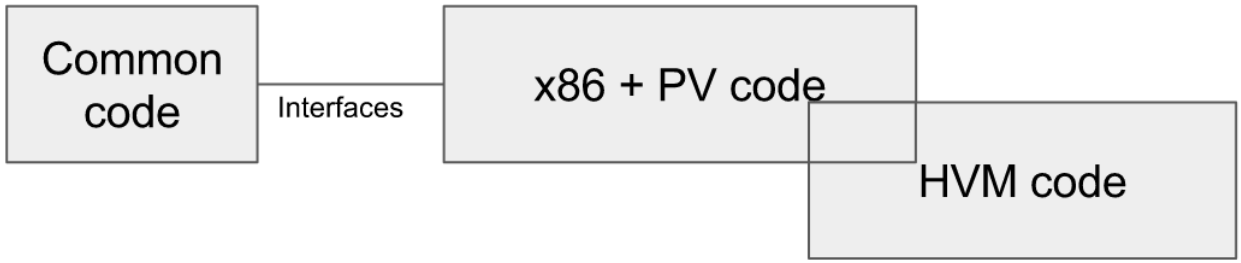
Xen x86 PV memory layout



The conceptual map of code



The reality



Current code

```
do_foo(...)
{
    /* ... */

    if (hvm) {
        do_foo_hvm();
        return;
    }

    /* lots of code to do foo for pv */

    return;
}
```

Current code

```
do_bar(...)
{
    /* ... */

    if (hvm) { /* Some code */ };

    if (pv) { /* Some code */ };

    /* lots of code for common case */

    if (hvm) { /* Some code */ };

    if (pv) { /* Some code */ };

    return;
}
```

Future code

```
do_baz(...)
{
    /* code for common case */

    if (hvm)
        do_baz_hvm();

    if (pv)
        do_baz_pv();

    return;
}
```

Game plan for splitting PV and HVM code

- Identify all the components that need refactoring
  - Dom0 builder
  - Domain handling code
  - Trap handling code
  - Memory management code
  - Guest memory accessor
  - ...

Game plan for splitting PV and HVM code

- Coarse-grained refactoring – mostly for PV code
  - Move code around
  - Split code into manageable trunks
  - Do some basic cleanups:
    - Use better function names
    - Use better coding style

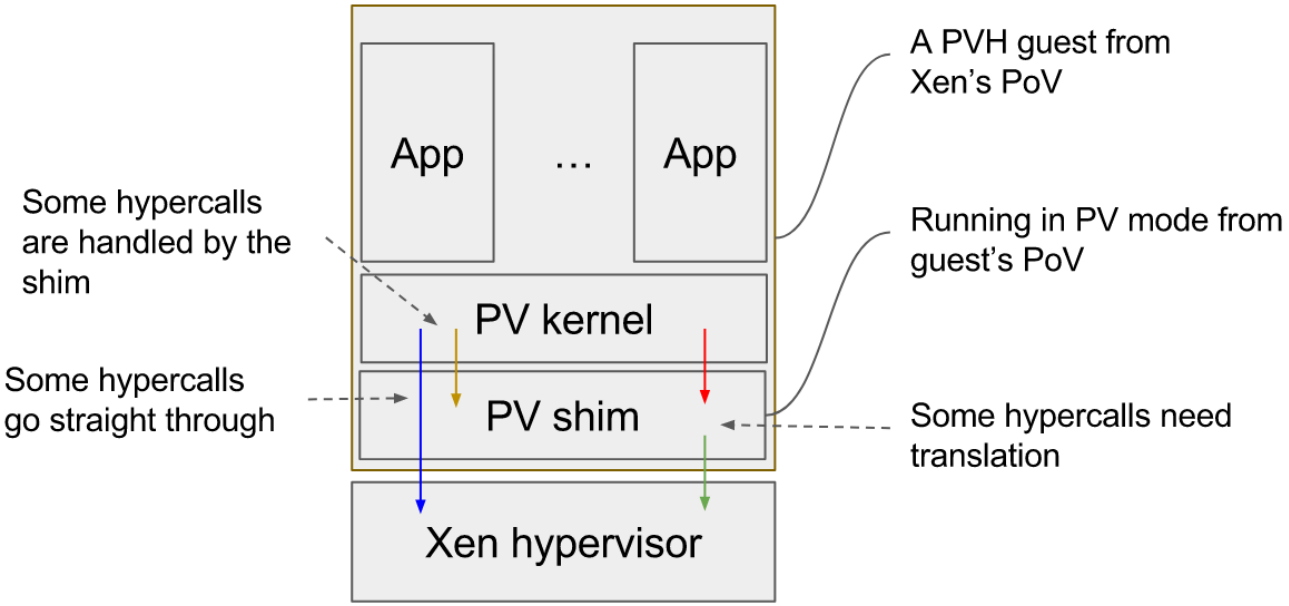
Game plan for splitting PV and HVM code

- Fine-grained refactoring – for both PV and HVM code
  - Abstract out a set of guest interfaces
  - Adjust internal interfaces between components if necessary
  - Fix x86 common code
  - Make PV and HVM configurable

Why PV ABI in PVH container?

- Continue to support PV in a more secure manner
- Have more than 128GB worth of 32bit PV guests

PV ABI in PVH container



Game plan for PV ABI in PVH container

- Build the PV shim – essentially a stripped-down Xen hypervisor
  - Go through all PV hypercall handlers, categorize them into the aforementeioned groups
  - Further refactor PV guest supporting code: provide the "real PV" handlers and "PV shim" handlers while sharing as much code as possible
  - Change the build system to pull in the right objects

Game plan for PV ABI in PVH container

- Adjust Xen toolstack
  - Construct a PVH guest while using the PV shim as "firmware"
- Further improvements (open questions at the moment)
  - Provide mechanism to parse guest kernel inside the container, something like pvgrub
  - Provide mechanism to pass-through PCI devices (if that's still relevant)

Current status

- Doing coarse-grained refactoring:
  - Dom0 builder (done)
  - Domain handling code (done)
  - Trap handling code (done)
  - Memory management code (doing)
  - Guest memory accessor
  - ...
- ETA: Some point in the future :)

Q&A