VirtIO on Xen hypervisor (Arm)

Oleksandr Tyshchenko, Lead Software Engineer
EPAM Systems Inc.

Linaro Connect
VIRTUAL • SPRING 2021

<epam>

About me

Oleksandr Tyshchenko

- Software developer at EPAM Systems Inc., Kyiv, Ukraine
- Xen project contributor
- Focusing on the following topics:
  - Renesas R-Car Gen3 platform support and maintenance
  - IOMMU subsystem
  - VirtIO support
  - Power management

Agenda

- Why VirtIO?
- A bit of VirtIO history on Xen
- Current VirtIO activities on Xen
- What is the IOREQ Server?
- The idea to reuse IOREQ for virtio-mmio
- VirtIO requires more than just IOREQ, what else?
- VirtIO backend problem on Xen
- The first virtio-mmio backend named “virtio-disk”
- Outstanding VirtIO questions on Xen
- virtio-pci support is planned as well
- “bufioreq” for VirtIO notification doorbell
- Future work
- Demo “Unmodified Guest runs on virtio-blk”

Why VirtIO?

VirtIO topic needs no real introduction.

- Flexible, generic, well-documented and standardized cross-hypervisor solution for I/O virtualization in the Automotive domain
- The support is available in Linux, Android, there are a lot of existing VirtIO drivers which could be just reused, also there is work in progress for the Automotive domain
- Has common core transports and data structures that significantly reduce work to implement each new virtual device driver
- Device drivers for paravirtualization do not need to be maintained uniquely for different hypervisors
- Guest VMs could be moved among different hypervisor environments without further modification/adaptation

Current spec “Virtual I/O Device (VIRTIO) Version 1.1”
https://docs.oasis-open.org/virtio/virtio/v1.1/cs01/virtio-v1.1-cs01.html

A bit of VirtIO history on Xen

VirtIO has never been fully supported on mainline Xen.
The prototype was implemented as the project for Google Summer of Code 2011 (now outdated):
https://wiki.xenproject.org/wiki/Virtio_On_Xen
Periodical discussions at the design sessions:
https://lists.xenproject.org/archives/html/xen-devel/2019-07/msg01746.html

What does Xen use instead?
Xen uses its own solution for the device paravirtualization so far (Xen's PV split device-driver mode) which is based on Xen specific transport:

- event channels for inter-domain communication
- grants for memory sharing
- xenbus for configuration negotiation

There a lot of Xen PV frontends, which have been in mainline Linux for years: block, network, console, display, sound, input, etc. PV backends are also available, some of them are kernel drivers, other are user-space programs (Qemu or standalone applications).

Current VirtIO activities on Xen

There is an increasing interest in VirtIO on Xen these days.

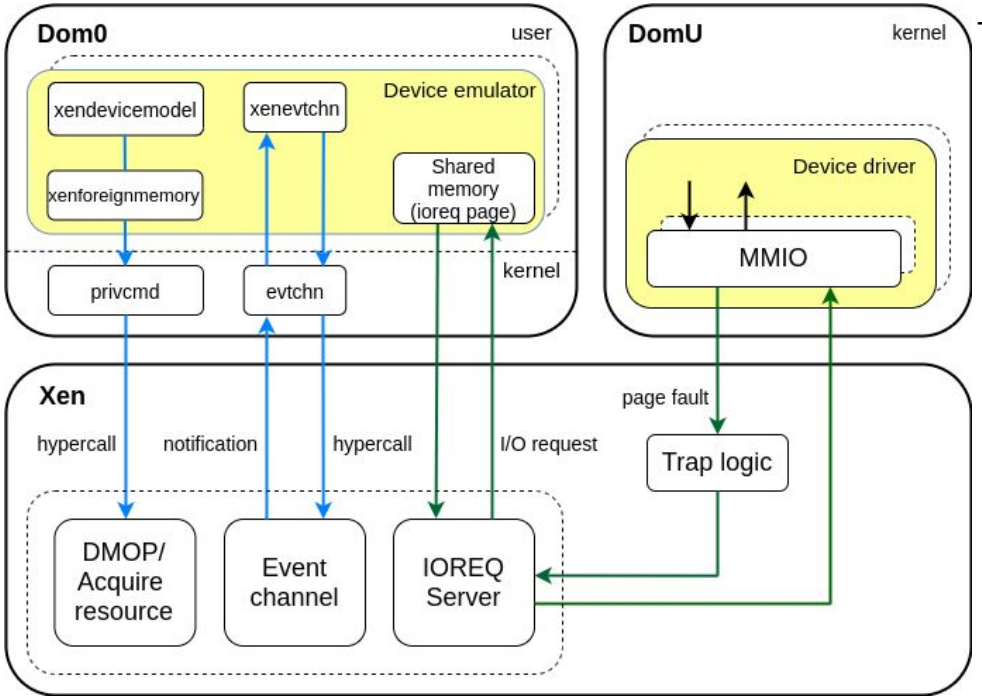
There are at least two related activities:

- Xen Arm community's "VirtIO using IOREQ Server feature": https://lists.xenproject.org/archives/html/xen-devel/2020-07/msg00825.html
  - Based on Xen's ability to trap, decode and emulate accesses to memory
  - Doesn't require any modifications to existing VirtIO infrastructure
  - Doesn't assume isolation: frontend memory buffers are accessible to backends
- Open XT's "Argo HMX transport for VirtIO" https://openxt.atlassian.net/wiki/spaces/DC/pages/1696169985/VirtIO-Argo+Development+Phase+1
  - Based on Hypervisor-Mediated data eXchange
  - Requires new virtio-argo transport device driver
  - Assumes isolation (no shared memory, hypervisor performs data movement)

The IOREQ enabling work is an area of interest for the Linaro's project Stratos to create VirtIO aware Xen model for QEMU for testing additional devices.
https://projects.linaro.org/browse/STR-19

What is the IOREQ Server?

The interface between device emulator and Xen is called an IOREQ Server.
The IOREQ is the connecting link between device emulator and the Guest.



The IOREQ feature:

- allows for several device emulator to be attached to the same Guest, each emulating different sets of virtual HW.
- allows for a relatively easy and simple implementation of the necessary "hookups" between your device emulator and the Guest.
- is used by QEMU and the Xen reference implementation DEMU, and can be used for MMIO as well as PIO assesses.
- is well-suited to implement virtio-mmio... https://xcp-ng.org/blog/2020/06/03/device-emulation-in-the-xen-hypervisor/

The idea to reuse IOREQ for virtio-mmio

The idea belongs to Julien Grall (Xen Arm maintainer) and initially implemented in his PoC "xen/arm: Add support for Guest IO forwarding to a device emulator".

Key notes:

- Guest MMIO assesses are forwarded to the virtio-mmio backend using the IOREQ feature.
- The backend is an userspace application which runs in privileged domain and emulates virtual device(s) for the Guest(s).
- Up to 8 various backends can be run simultaneously (multiple IOREQ Servers).
- This approach doesn't require any modifications at the Guest side.
- The IOREQ feature can be reused for implementing the virtio-pci transport later on.

Initially x86's features the IOREQ/DM have been recently ported on Arm and upstreamed!
[PATCH V6 00/24] IOREQ feature (+ virtio-mmio) on Arm
https://lore.kernel.org/xen-devel/1611884932-1851-1-git-send-email-olekstysh@gmail.com/
https://lore.kernel.org/xen-devel/1611938365-19059-1-git-send-email-olekstysh@gmail.com/

Thanks to the Linaro Community who helped to review and test the patch series!

VirtIO requires more than just IOREQ, what else?

Yes, the IOREQ is a key component, but what we also need are:
(Available in mainline Xen)

- **Memory sharing:**
  The backend needs to access the Guest memory. Xen provides functionality for privileged foreign mapping “xenforeignmemory” which allows mapping Guest memory in its address space.
- **Guest notification:**
  The frontend notifies the backend using MMIO write. The backend in turn needs to notify the frontend using an SPI. Xen provides mechanism for device model operation hypercalls “xendevicemodel”.
  New DMOP “xendevicemodel_set_irq_level” was introduced to serve that purpose.
  Proposed interface allows emulator to set the logical level of a one of a domain's IRQ lines.

VirtIO requires more than just IOREQ, what else?

Yes, the IOREQ is a key component, but what we also need are:
(Not available in mainline Xen yet)

- **Guest device-tree updates:**
  Allocate a pair “IRQ and memory range” and insert resulting virtio-mmio node for each VirtIO device in the device-tree. Xen provides Guest DT generation.
  https://elixir.bootlin.com/linux/v5.11/source/Documentation/devicetree/bindings/virtio/mmio.txt
- **Backend configuration:**
  Read VirtIO device options from the configuration file and pass them to the backend. Xen provides filesystem-like database “xenstore” and bus abstraction “xenbus”.

The nearest plan is to upstream these missing virtio-mmio bits to Xen toolstack (prototype works):
[PATCH V4 23/24] libxl: Introduce basic virtio-mmio support on Arm
https://lore.kernel.org/xen-devel/1610488352-18494-24-git-send-email-olekstysh@gmail.com/
[PATCH V4 24/24] [RFC] libxl: Add support for virtio-disk configuration
https://lore.kernel.org/xen-devel/1610488352-18494-25-git-send-email-olekstysh@gmail.com/

VirtIO backend problem on Xen

The main problem is the absence of “ready-to-use” and “out-of-Qemu” VirtIO backends.

There are multiple implementations of VirtIO backends, but they can’t be easily reused on Xen:

- Mostly the backend is tightly coupled with a hypervisor it was designed for.
- Mostly the backend is hosted in QEMU, an additional effort is required to pull them out.

The aim of our PoC is to create backend as a standalone application and outline the hypervisor specific parts.

Ideally the VirtIO backend should be hypervisor agnostic (with a minimal shim dealing with hypervisor specifics).

This is also an area of interest for the Linaro’s project Stratos to create a common VirtIO library for use by programs implementing the backend:
https://projects.linaro.org/browse/STR-7

The first virtio-mmio backend named “virtio-disk”

1. IOREQ Server maintenance:
   register, set state, map I/O range, etc

2. New DMOP to send notification to the Guest:
   xendevicemodel_set_irq_level

1. Map shared IOREQ page(s) at initialization

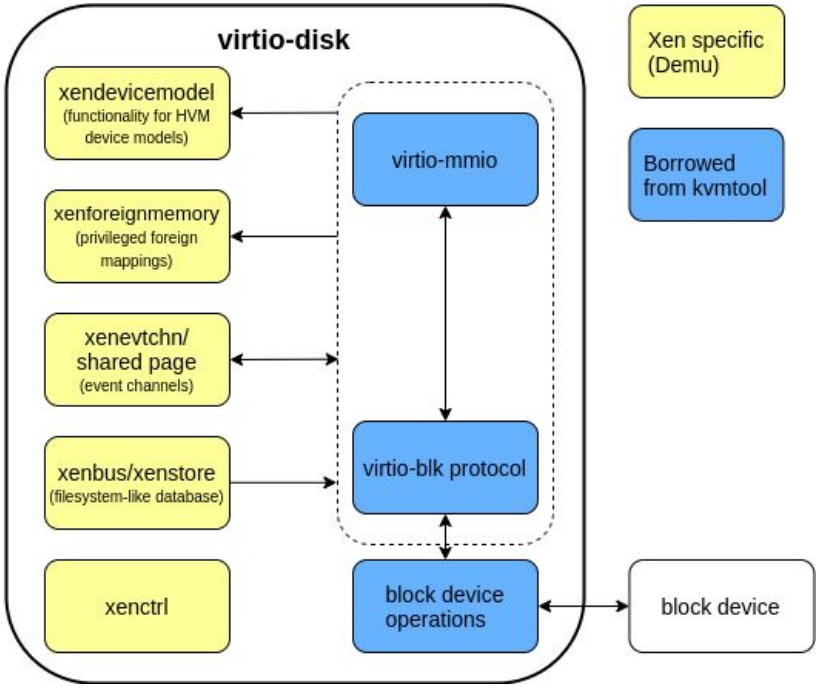
2. Map/unmap Guest memory at runtime

Communication channel with Xen composed of a shared page and an event channel:
get notifications when Guest accessing target MMIO region. I/O request via shared IOREQ page

1. Read configuration from Xenstore:
- disk filename, readonly property
- virtio-mmio IRQ and base address

2. Get notification when Guest creating/destroying

Retrieve domain's info (the number of vCPUs)*



Based on demu: https://xenbits.xen.org/gitweb/?p=people/pauldu/demu.git
and kvmtool: https://git.kernel.org/pub/scm/linux/kernel/git/will/kvmtool.git
Available at: https://github.com/xen-troops/virtio-disk/commits/ioreq_ml2

Outstanding VirtIO questions on Xen

Backend is able to access all Guest memory.

There are a few possible solutions how to restrict a backend:

- **Memory sharing using Xen's grant-table:**
  - Requires modifications to the VirtIO spec (grants instead of raw addresses in vring)
  - May slow the adoption of Xen in some areas
- **Virtual IOMMU (VirtIO or Xen PV IOMMU):**
  - Requires an implementation of Virtual IOMMU backend in Xen
- **The proposal based on pre-shared memory "Swiotlb bounce buffer":** https://lwn.net/Articles/818793/
  - Requires modifications to the VirtIO infrastructure
  - Involves expensive memory copy

Unfortunately, all these solutions will affect performance if we require to map/unmap at every request...

Xen community is not the single hypervisor community interested in creating "less privileged" backend.
This is also an area of interest for the Linaro's project Stratos to solve this for all hypervisors.
https://projects.linaro.org/browse/STR-25

Outstanding VirtIO questions on Xen

Accessing Guest memory via “foreing mapping”

The memory management between KVM and Xen is quite different:

- In the case of KVM, the Guest memory is effectively the memory from the userspace and then shared with the Guest. It is managed the same way as "normal" userspace memory.
- In the case of Xen, the backend has to issue hypercalls in order to map/unmap Guest memory, so we technically steal a physical page from backend domain memory in order to map the Guest page in backend address space.
  We don’t provide any facilities for Linux to reclaim the page if it needs it before the backend actually unmaps the Guest page.

A lot of “foreing mapping” could lead to the memory exhaustion in backend domain (XSA-300).
https://xenbits.xen.org/xsa/advisory-300.html

Outstanding VirtIO questions on Xen

Accessing Guest memory via “foreing mapping” (continue)

There are two possible approaches how to map Guest memory:

- **Map all Guest memory at boot (static):**
  - Better performance
  - Big memory overhead in backend domain (depends on the Guest memory size)
  - Need to remap Guest memory when the Guest memory layout is changed
- **Map/unmap Guest memory at runtime (dynamic):**
  - Small memory overhead in backend domain
  - No need to care for the Guest memory layout changes
  - Worse performance than with static approach

Dynamic approach is currently used in the backend, the static approach has been checked as well.

Julien has an idea how to handle “foreing mapping” the same way as Linux handles userspace memory.

virtio-pci transport is planned as well

The virtio-pci is more complex than virtio-mmio, so why we want/need to support it?

- We shouldn't tie Xen to any of the VirtIO protocols.
- virtio-pci is more interesting from the Xen on x86 PoV (virtio-mmio can't be easily reused on x86).
- virtio-pci provides flexibility: virtio-mmio is a good fit when we know all devices at boot and for fastboot requirements. For the hotplug purposes the virtio-pci is a better fit.
- With virtio-pci we get a bunch of things "for free" - enumeration and the MSI support. MSI implementation is useful to improve performance (support for ITS).

There is an interesting proposal to enhance virtio-mmio spec by supporting MSI:
https://lwn.net/Articles/812055/

The good sign for virtio-pci is that PCI passthrough/vPCI on Arm is actively developed these days.

The main differences in comparison with virtio-mmio from Xen PoV:

- for virtio-pci we need to forward Guest PCI config space accesses to the backend (support for vPCI).
- for virtio-pci we don't need to allocate memory/interrupts in the Xen toolstack.

“bufioreq” for VirtIO notification doorbell

The current way to handle Guest MMIO access (using default non-buffered I/O mode):

- Pause the vCPU
- Forward the access to the backend
- Schedule the backend domain
- Wait for the access to be handled by the backend
- Unpause the vCPU

The sequence is going to be fairly expensive on Xen.
It might be possible to optimize the ACK and avoid waiting for the backend to handle the access using the buffered I/O mode provided by IOREQ.

Future work

What's next?

- Upstream missing virtio-mmio bits to the Xen toolstack
- Various IOREQ optimizations ("bufioreq" for VirtIO notification doorbell, etc)
- Handle foreing mapping (Guest memory) the same way as Linux deals with "normal" userspace memory
- virtio-pci implementation (once vPCI is fully supported on Arm)

Demo “Unmodified Guest runs on virtio-blk”

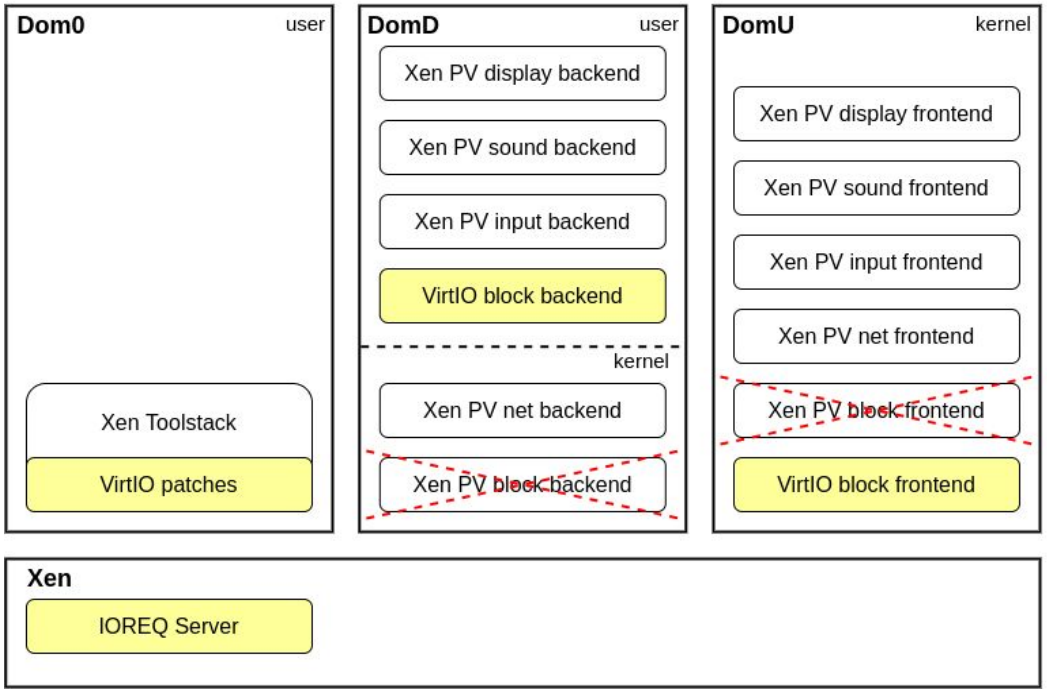
Board:
Renesas R-Car Starter Kit Premier board with Kingfisher Infotainment Board
https://elinux.org/R-Car/Boards/H3SK
https://elinux.org/R-Car/Boards/Kingfisher

Product:
Xen-troops’ Prod-devel with VirtIO pull request
https://github.com/xen-troops/meta-xt-prod-devel

System: Xen + Domains
Xen: Xen version 4.15-unstable (was built with CONFIG_IOREQ_SERVER=y and CONFIG_EXPERT=y)
Domain-0: Linux version 4.14.75-ltsi-yocto-tiny
DomD/DomU: Linux version 4.14.75-ltsi-yocto-standard

```
root@generic-armv8-xt-dom0:~# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0   256     4      r-----    15.4
DomD                                         1  1995     4      -b----    47.8
DomU                                         2  1535     4      -b----     6.8
```

Demo “Unmodified Guest runs on virtio-blk”



Involved system components:

Xen:
No extra patches! Build with "CONFIG_EXPERT = y" and "CONFIG_IOREQ_SERVER = y"

Dom0:
1. Two additional patches are needed for Xen Toolstack:
- enable virtio-mmio support
- enable "virtio-disk" backend support
2. "virtio-disk" backend configuration in domu.cfg

DomD:
"virtio-disk" backend and it's systemd service

DomU:
No changes! Make sure that VirtIO configs are enabled in Linux defconfig

Dom0: launches VMs, handles VirtIO configuration

domu.cfg:

```
...
extra = "root=/dev/vda rw rootwait ..."
vdisk = [ 'backend=DomD, disks=rw:/dev/mmcblk1p3' ]
virtio = 1
```

domu.dts:

```
...
    virtio@2000000 {
            compatible = "virtio,mmio";
            reg = < 0x00 0x2000000 0x00 0x200 >;
            interrupts = < 0x00 0x01 0xf01 >;
            interrupt-parent = < 0xfde8 >;
            dma-coherent;
    };
```

xen/include/public/arch-arm.h:

```
...
/* VirtIO MMIO definitions */
#define GUEST_VIRTIO_MMIO_BASE  xen_mk_ullong(0x02000000)
#define GUEST_VIRTIO_MMIO_SIZE  xen_mk_ullong(0x200)
#define GUEST_VIRTIO_MMIO_SPI   33
```

root@generic-armv8-xt-dom0:~# xenstore-ls -f | grep virtio_disk

```
...
/local/domain/2/device/virtio_disk = ""
/local/domain/2/device/virtio_disk/0 = ""
/local/domain/2/device/virtio_disk/0/backend =
   "/local/domain/1/backend/virtio_disk/2/0"
/local/domain/2/device/virtio_disk/0/backend-id = "1"
/local/domain/2/device/virtio_disk/0/state = "1"
/local/domain/2/device/virtio_disk/0/0 = ""
/local/domain/2/device/virtio_disk/0/0/filename = "/dev/mmcblk1p3"
/local/domain/2/device/virtio_disk/0/0/readonly = "0"
/local/domain/2/device/virtio_disk/0/0/base = "33554432"
/local/domain/2/device/virtio_disk/0/0/irq = "33"
```

DomD: runs virtio-disk backend (owns block HW)

root@h3ulcb-4x2g-kf-xt-domd:~# dmesg | grep mmcblk1

...
[2.213704] mmcblk1: mmc1:aaaa SL16G 14.8 GiB
[2.225682] mmcblk1: p1 p2 p3

root@h3ulcb-4x2g-kf-xt-domd:~# journalctl -u virtio-disk

...
Jan 31 08:59:54 h3ulcb-4x2g-kf-xt-domd systemd[1]: Starting virtio-disk backend...
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: main: read frontend domid 2
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: Info: connected to dom2
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: >XENSTORE_ATTACHED
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: domid = 2
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: filename[0] = /dev/mmcblk1p3
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: readonly[0] = 0
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: base[0] = 0x2000000
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: irq[0] = 33
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: >XENCTRL_OPEN
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: >XENEVTCHN_OPEN
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: >XENFOREIGNMEMORY_OPEN
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: >XENDEVICEMODEL_OPEN
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_initialize: 4 vCPU(s)
Jan 31 09:00:03 h3ulcb-4x2g-kf-xt-domd virtio-disk[3371]: demu_seq_next: >SERVER_REGISTERED

DomU: runs virtio-disk frontend (rootfs on vda)

root@h3ulcb-4x2g-kf-xt-domu:~# dmesg | grep vda

```
[    0.000000] Kernel command line: root=/dev/vda rw rootwait console=hvc0 cma=256M@1-2G pvrsrvkm.DriverMode=1
[    0.590065] EXT4-fs (vda): mounted filesystem with ordered data mode. Opts: (null)
[    1.403703] EXT4-fs (vda): re-mounted. Opts: (null)
```

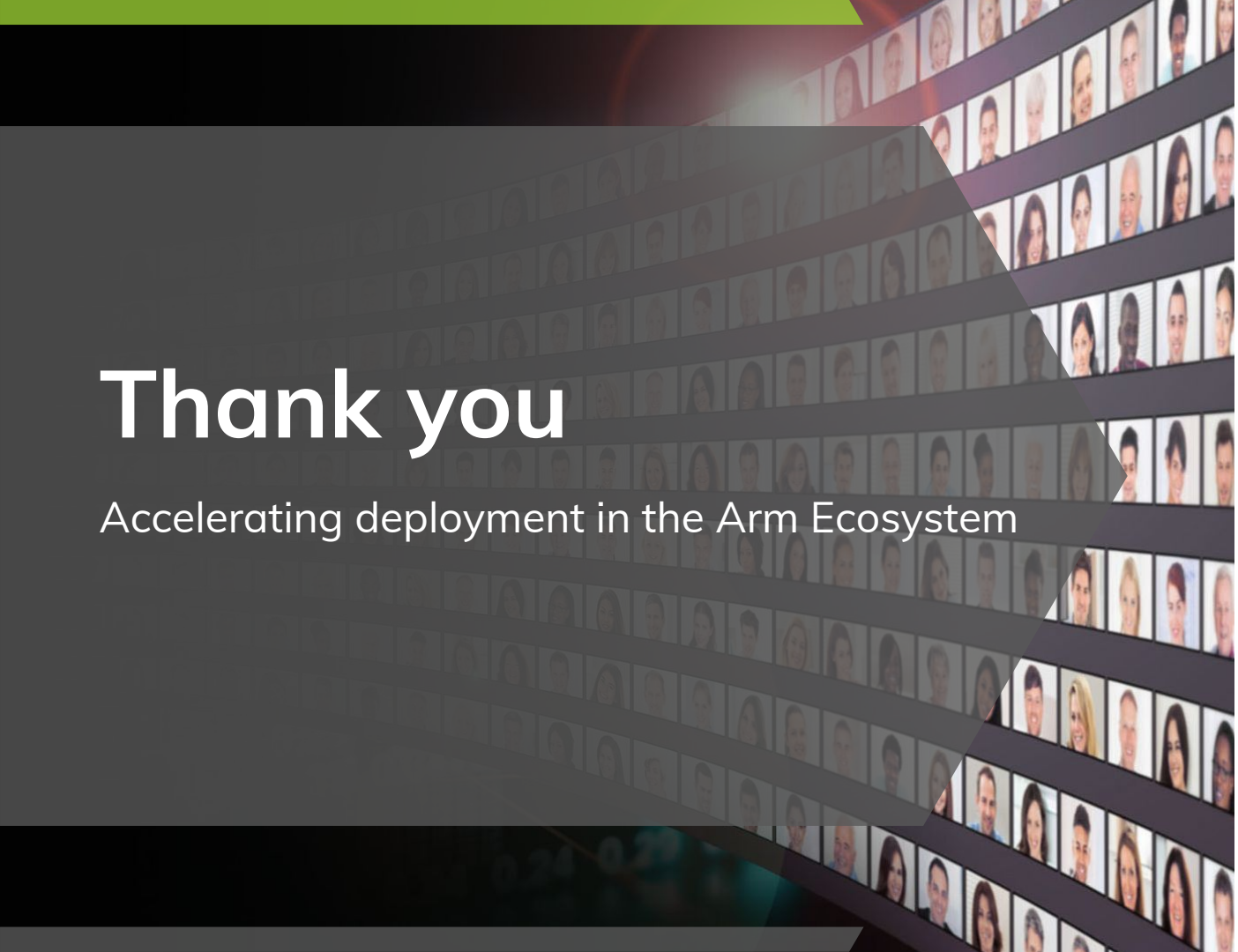
root@h3ulcb-4x2g-kf-xt-domu:~# ls -l /sys/block/vda/device/

```
total 0
drwxr-xr-x 3 root root    0 Jan 31 17:01 block
-r--r--r-- 1 root root 4096 Jan 31 17:14 device
lrwxrwxrwx 1 root root    0 Jan 31 17:14 driver -> ../../../../bus/virtio/drivers/virtio_blk
-r--r--r-- 1 root root 4096 Jan 31 17:14 features
-r--r--r-- 1 root root 4096 Jan 31 17:14 modalias
drwxr-xr-x 2 root root    0 Jan 31 17:14 power
-r--r--r-- 1 root root 4096 Jan 31 17:14 status
lrwxrwxrwx 1 root root    0 Jan 31 17:01 subsystem -> ../../../../bus/virtio
-rw-r--r-- 1 root root 4096 Jan 31 17:01 uevent
-r--r--r-- 1 root root 4096 Jan 31 17:14 vendor
```

VirtIO PoC works fine with suitable performance...

For block sizes > 4K (64K, 256K, 512K, 1M, ...) the bandwidth is *not* lower than with traditional Xen PV block driver.

Questions?



Thank you

Accelerating deployment in the Arm Ecosystem

Linaro Connect
VIRTUAL • SPRING 2021