RunX

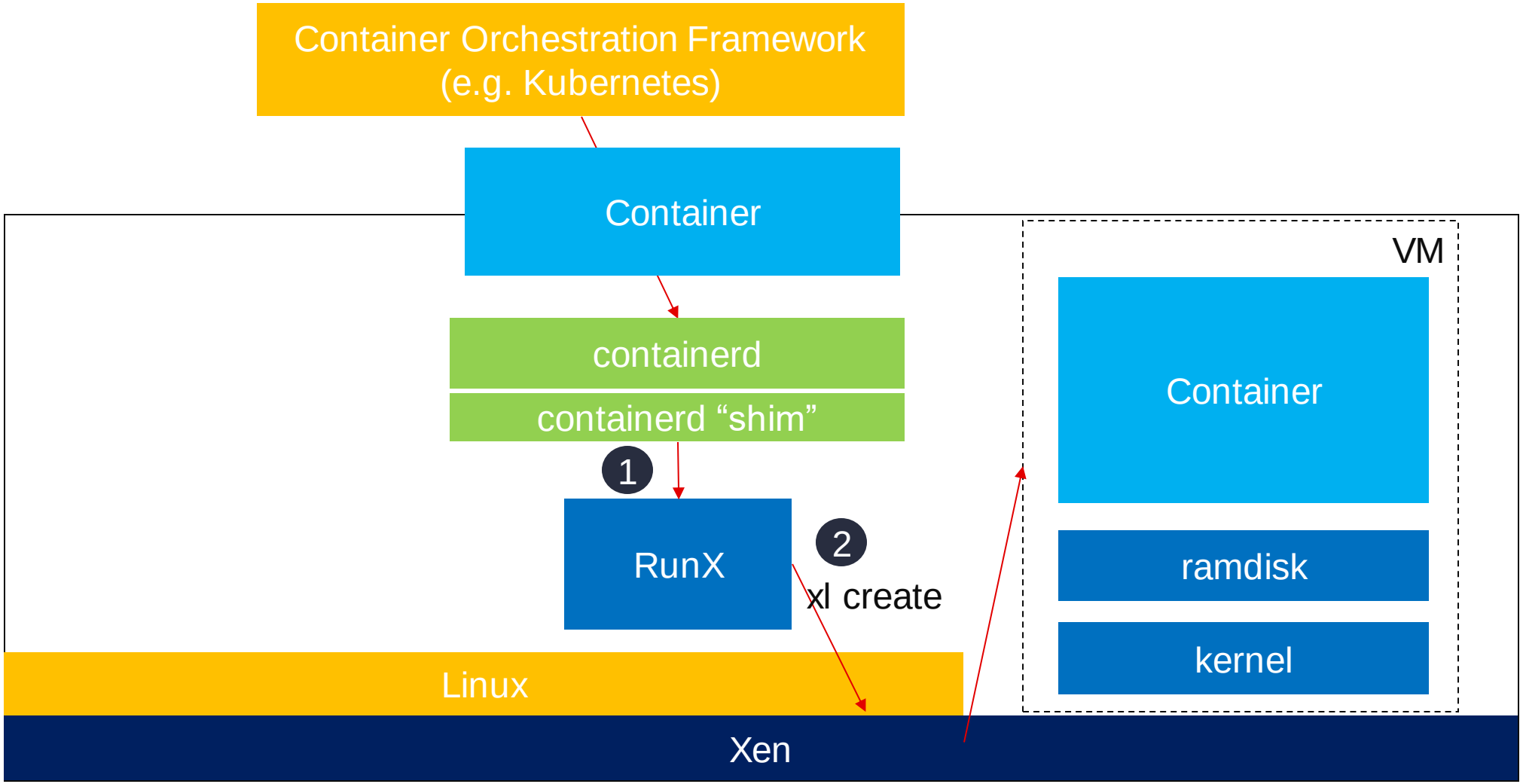
Stefano Stabellini, Xilinx
Bruce Ashfield, Xilinx
Rob Woolley, WindRiver

XILINX

Introducing RunX

- https://github.com/lf-edge/runx
- A **new** OCI-compatible containers runtime to start containers as Xen VMs
- Written for Embedded
  - Very simple
  - Minimal overhead
  - Real-Time support
  - Accelerators support
  - Secure by Default
- New project started under the Linux Foundation Edge (LF-Edge) umbrella
  - Early collaboration with Zededa
  - Permissive license (Apache v2)
  - Open to contributions from the start
  - All development using a public mailing list: https://lists.lfedge.org/g/eve-runx

Introducing RunX



XILINX.

RunX: implementation choices

- Easy to Build: minimal build dependencies
  - gcc, make, go
  - cross-compiles
  - no Xen dependecy at build-time
- Easy to Run: minimal runtime dependencies
  - (in addition to Xen,) bash, jq, socat, daemonize
  - no ties between Xen and RunX versions

RunX: implementation choices

- No in-guest agents: minimal runtime overhead
  - Provides a minimal Linux kernel and Busybox-based ramdisk for booting regular containers as VMs
  - Pristine container environment
- Tiny Micro-VMs optimized for embedded
  - A minimal environment
  - No device emulation
  - No in-guest firmware or bootloaders
- OCI Runtime Spec compliant
  - Developed together with ContainerD
  - Should work with any container engines

RunX (Cross)Build

- Cross-build requirements:
  - cross-compilation toolchain
    - e.g. Linaro: https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu
  - golang compiler (soon to be removed)
    - distro golang package works

```
$ export ARCH=aarch64
$ export GOROOT=/usr/lib/go-1.10
$ export CROSS_COMPILE=/path/to/aarch64-linux-gnu-
$ ./build.sh
```

RunX Runtime

- Copy runX and /usr/share/runX to target
- Enable it in containerd's config.toml, containerd <= 1.2.9:

```
[plugins.linux]
     runtime="/usr/sbin/runX"
```

- containerd >= 1.4.0:

```
ctr run --runc-binary=/usr/sbin/runX ...
```

Yocto + RunX

- Why use Yocto / OE to build / deploy RunX ?
  - Leverage the Yocto / OE core values
    - configurability, licence management, cross build, fine grained image composition, performance tuning, etc.
  - Transition path from development to production
  - Active community and integration with BSPs (e.g. xilinx, rpi)
  - Multiconfig builds
    - Build host + guests + containers + firmware in a single platform

- The build of RunX within Yocto is an integration
  - Development and build directly with upstream projects is always possible

RunX: oe-core + BSP + meta-virtualization

- Layers define the platform and software stack possibilities
  - Distro, configuration, package recipes and image recipes: customize and specify the details
- For RunX:
  - oe-core: base support
    - toolchain, base packages, image construction, etc
  - meta-virtualization: container runtimes + support
    - RunX, Xen and supporting components (containerd, cni, ...)
    - Xen host reference image recipe, dom0
    - u-boot, initrd and image build
  - BSP layers provide hardware support
    - Kernel, bootloader, firmware, tightly coupled userspace packages

Yocto + RunX: simplified build steps (xilinx-zcu102)

- Use master branches (there are no stable/released variants yet)

```
$ git clone -b master http://git.yoctoproject.org/git/poky
$ git clone -b master http://git.openembedded.org/meta-openembedded
$ git clone -b master https://git.yoctoproject.org/git/meta-virtualization
$ git clone -b master https://github.com/Xilinx/meta-xilinx.git

$ . ./oe-init-build-env zcu102-zynqmp

$ bitbake-layers add-layer $(readlink -f $PWD/../meta-openembedded/meta-oe)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-openembedded/meta-filesystems)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-openembedded/meta-python)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-openembedded/meta-networking)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-virtualization)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-xilinx/meta-xilinx-bsp)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-xilinx/meta-xilinx-contrib)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-xilinx/meta-xilinx-standalone)
```

Yocto + RunX: Simplified setup

- Local build setup (will eventually be in a xen distro config)

```
$ cat <<EOF >> conf/local.conf
MACHINE ??= "zcu102-zynqmp"
DISTRO = "poky"
BBMULTICONFIG ?= "pmu"
do_image[mcdepends] = "multiconfig::pmu:pmu-firmware:do_deploy"

IMAGE_FSTYPES += "tar.gz cpio.gz.u-boot jffs2"
DISTRO_FEATURES_append=" xen virtualization vmsep"
IMAGE_INSTALL_append = " busybox xen-tools zlib-dev runx"
IMAGE_INSTALL_append += " virtual/containerd virtual/runc"
ASSUME_PROVIDED += "iasl-native"
PACKAGECONFIG_remove_pn-xen += " sdl"
PREFERRED_PROVIDER_qemu-native = "xilinx-qemu-native"
PREFERRED_PROVIDER_nativesdk-qemu = "nativesdk-qemu-xilinx"
BUILDHISTORY_FEATURES ?= "image package sdk"
QB_DEFAULT_KERNEL="none"
QB_MEM = "-m 4096"
EOF

$ cat << EOF > conf/multiconfig/pmu.conf
MACHINE="microblaze-pmu"
DISTRO="xilinx-standalone"
TMPDIR="${TOPDIR}/pmutmp"
EOF
```

Yocto + RunX: build steps

- Note: upstream submission and merging is in progress

```
# download and extract pmu files (optional: only if not using multiconfig):

# Download: https://www.xilinx.com/member/forms/download/xef.html?filename=xilinx-zcu102-v2020.1-final.bsp&akdm=1

$ tar -O -xf xilinx-zcu102-v2020.1-final.bsp xilinx-zcu102-2020.1/pre-built/linux/images/pmu_rom_qemu_sha3.elf >
pmu-rom.elf
$ tar -O -xf xilinx-zcu102-v2020.1-final.bsp xilinx-zcu102-2020.1/pre-built/linux/images/pmufw.elf > pmufw.elf
$ tar -O -xf xilinx-zcu102-v2020.1-final.bsp xilinx-zcu102-2020.1/pre-built/linux/images/system.dtb > system.dtb

# copy files to deploy dir:

$ cp pmufw.elf build/tmp/deploy/images/zcu102-zynqmp/pmu-zcu102-zynqmp.bin
$ cp pmufw.elf build/tmp/deploy/images/zcu102-zynqmp/pmu-zcu102-zynqmp.elf
$ cp pmu-rom.elf $BUILDDIR/tmp/deploy/images/zcu102-zynqmp/pmu-rom.elf
$ cp system.dtb $BUILDDIR/tmp/deploy/images/zcu102-zynqmp/system.dtb

# build the image(s)

$ bitbake core-image-minimal
$ bitbake xen-image-minimal
```

Yocto + RunX: build artifacts

- Outputs: build/tmp/deploy/images/$machine
  - Deploy directory is setup as tftp target on boot via runqemu by default

```
% xen-image-minimal*
  76M Oct 12 21:06 xen-image-minimal-zcu102-zynqmp-20201013040413.rootfs.cpio.gz.u-boot
  76M Oct 12 21:06 xen-image-minimal-zcu102-zynqmp-20201013040413.rootfs.tar.gz
  722M Oct 12 21:06 xen-image-minimal-zcu102-zynqmp-20201013040413.rootfs.wic.qemu-sd
  101M Oct 12 21:09 xen-image-minimal-zcu102-zynqmp-20201013040921.rootfs.jffs2

% Image*
  17M Sep 18 07:45 Image--5.4+git0+22b71b4162-r0-zcu102-zynqmp-20200918143300.bin
  56M Sep 16 14:16 image.ub

% u-boot*
  878K Sep 16 12:41 u-boot-zcu102-zynqmp-v2020.01-xilinx-v2020.1+gitAUTOINC+86c84c0d0f-r0.bin
  942K Sep 16 12:41 u-boot-zcu102-zynqmp-v2020.01-xilinx-v2020.1+gitAUTOINC+86c84c0d0f-r0.elf
```

Yocto + RunX: launch

- Some on target configuration/interaction is required (automated in the future)
- qemu example build (h/w boot differs)

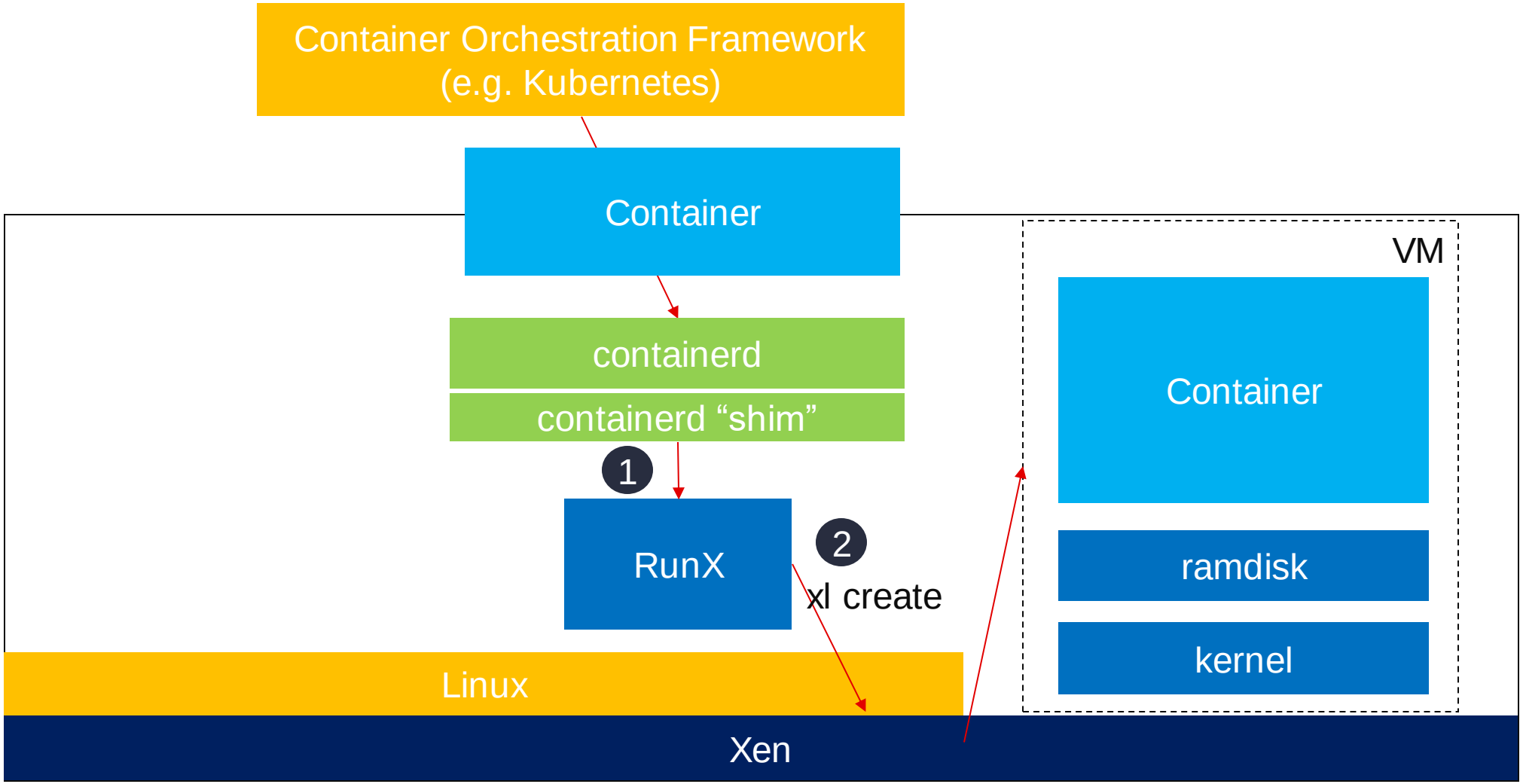
```
# core-image-minimal as a sanity test. Works out of deployed artifacts by default:

$ runqemu core-image-minimal slirp nographic

# xen-minimal: requires manual u-boot config, or boot.scr support from image builder
(https://gitlab.com/ViryaOS/imagebuilder)

$ runqemu xen-image-minimal nographic slirp
```

Yocto + RunX: runtime steps

- u-boot prompt (or automatically): load and exec boot configuration

```
% cat boot.source
tftpb 0x80000 Image;
tftpb 0x1280000 zynqmp-zcu102-rev1.0.dtb; tftpb 0x1400000 xen.ub; tftpb 0x9000000 xen-image-minimal-zcu102-zynqmp.cpio.gz.u-boot

fdt addr 0x1280000 ; fdt resize ; fdt set /chosen \#address-cells <1> ; fdt set /chosen \#size-cells <1>
fdt set /chosen xen,xen-bootargs "console=dtuart dtuart=serial0 dom0_mem=1G"

fdt mknod /chosen module@0 ; fdt set /chosen/module@0 compatible "xen,linux-zimage" "xen,multiboot-module"
fdt set /chosen/module@0 reg <0x80000 0x109aa00> ; fdt set /chosen/module@0 bootargs "root=/dev/ram earlyprintk=serial,ttyPS0
console=ttyPS0,115200n8 earlycon=xenboot clk_ignore_unused"

fdt mknod /chosen module@1 ; fdt set /chosen/module@1 compatible "xen,linux-initrd" "xen,multiboot-module"
fdt set /chosen/module@1 reg <0x9000000 0x241a84d>

bootm 0x1400000 0x9000000 0x1280000
```

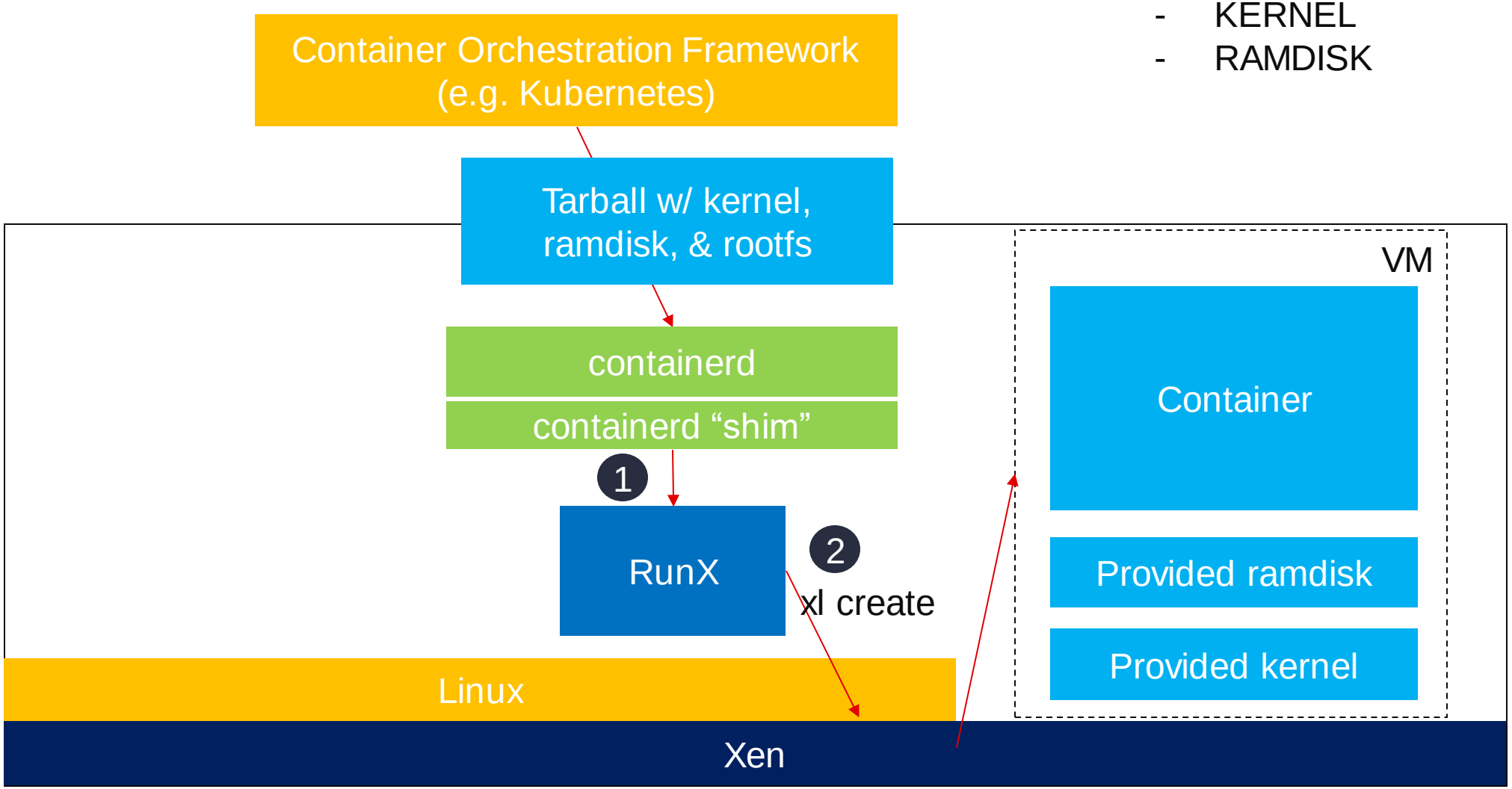
RunX: Traditional Containers



XILINX.

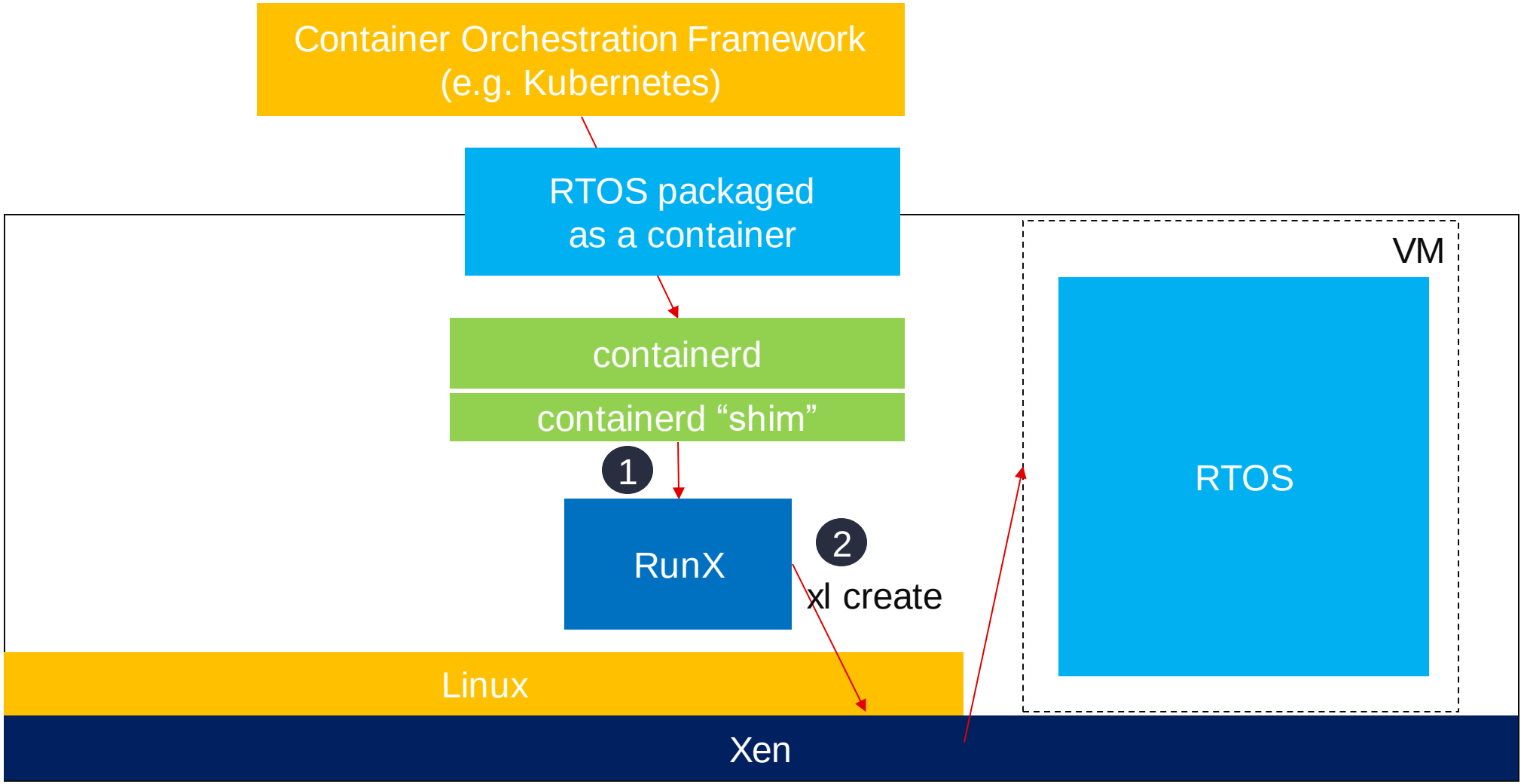
RunX: Containers with a Kernel

OCI Image Spec Extensions:
- KERNEL
- RAMDISK



XILINX.

RunX: Baremetal and RTOS Containers

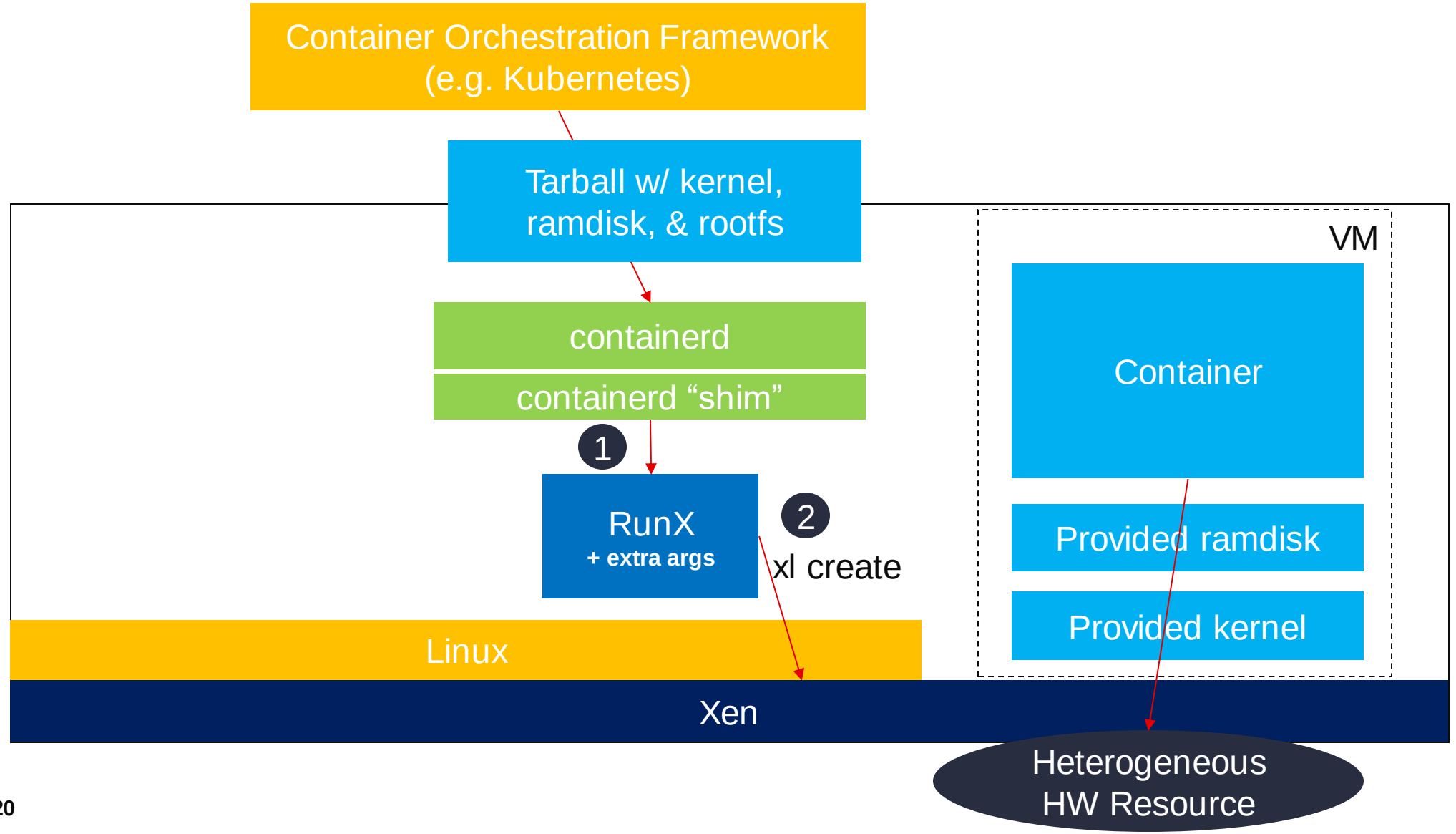


XILINX.

RunX: Containers with a Kernel

- Support containers that come with their own Kernel and/or Ramdisk
  - a specific version of the Linux kernel
  - a specific kernel configuration
  - LinuxRT
- Non-Linux OSes
  - RTOSes
  - Baremetal applications
  - VxWorks
- Kernel and Ramdisk are advertised using new OCI Image flags
  - TBD; currently Implemented using Environmental Variables
    - *RUNX_KERNEL*
    - *RUNX_RAMDISK*
  - Work with CNCF to standardize the new labels

RunX: Device Assignment



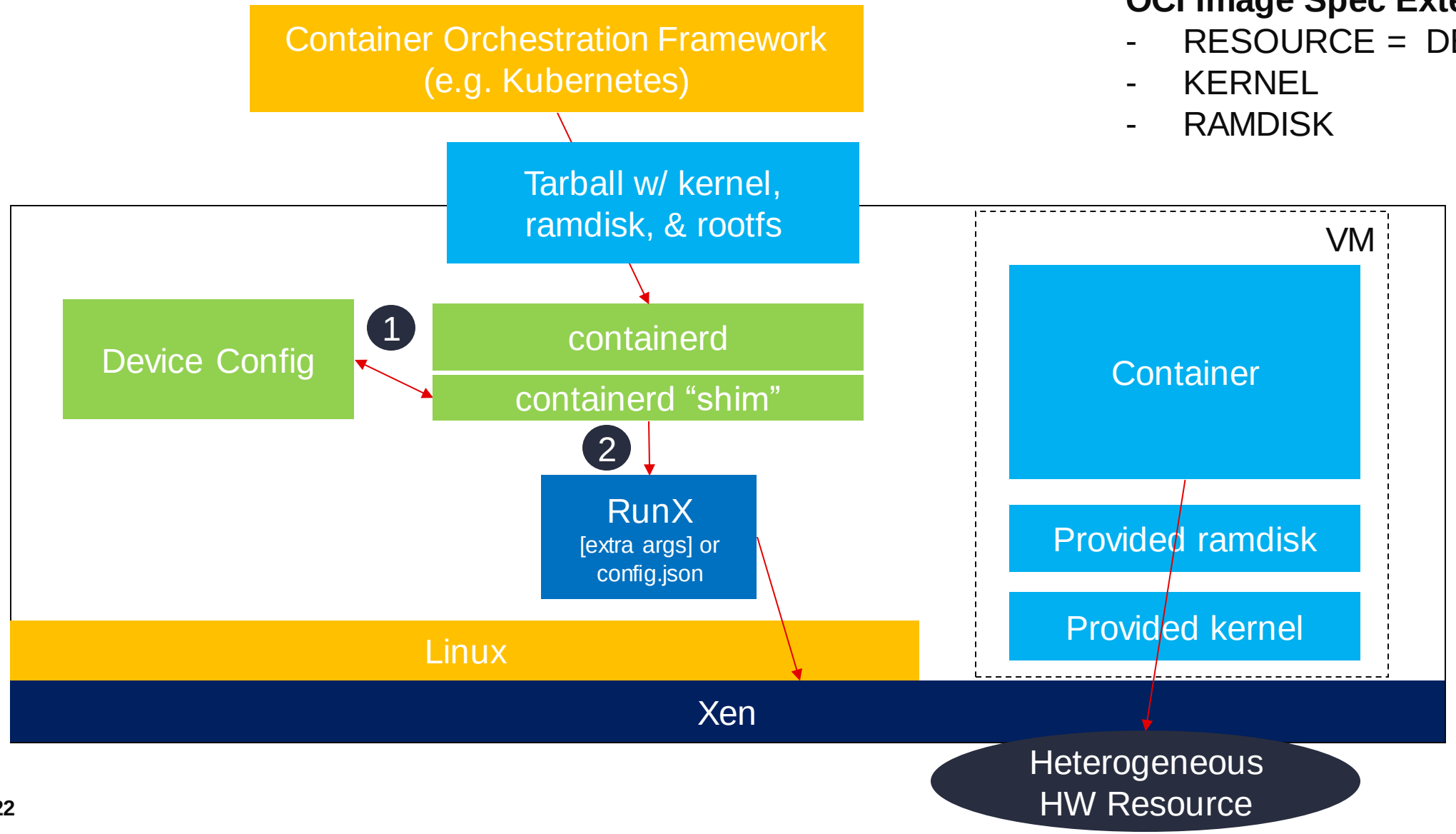
XILINX.

Device Assignment

- Device Assignment support via XLCONF
  - Appends configuration options to the xl config file
  - It can be used for anything from device assignment to changing vcpus and memory configurations
  - It can be used to set real-time configurations
  - It is set by the user/admin (not by the container)

Vision: Accelerators & FPGAs

OCI Image Spec Extensions:
- RESOURCE = DEVICE_DATA
- KERNEL
- RAMDISK



XILINX.

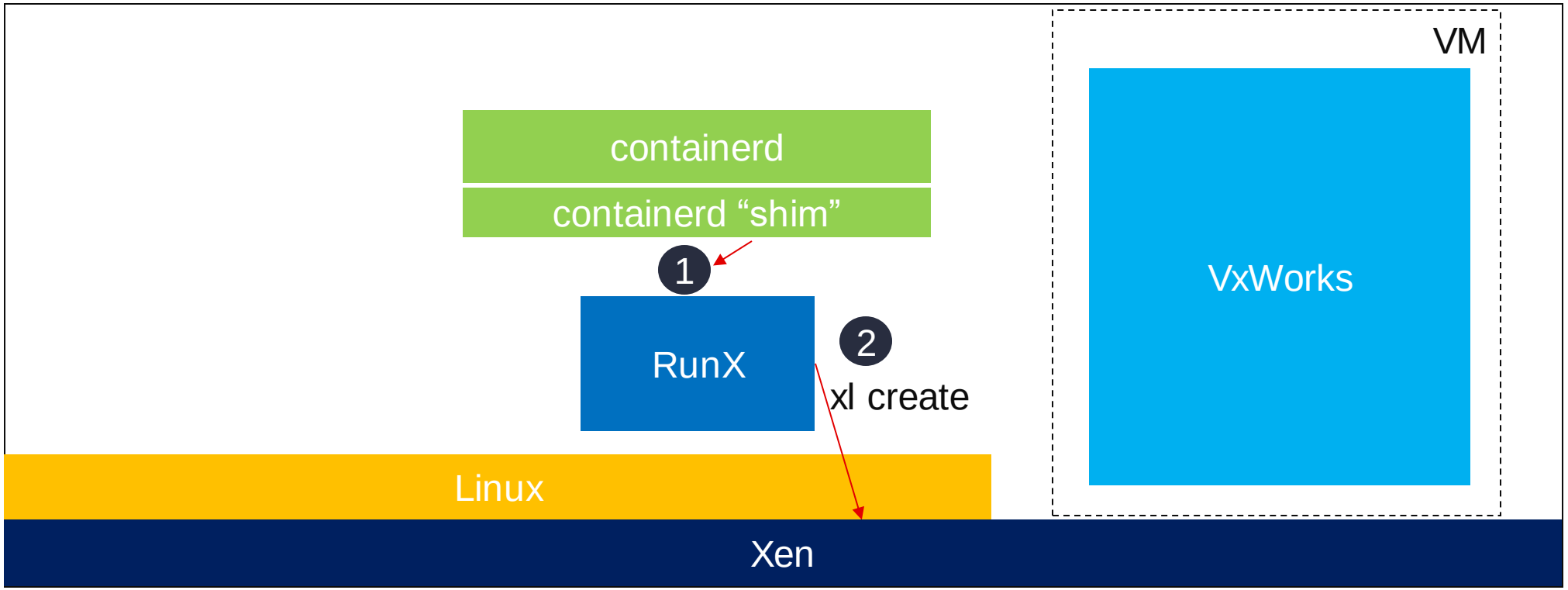
Vision: Accelerators & FPGAs

1. Containers come with their own accelerator's binaries and data
   - FPGA bitstreams
   - Co-Processor Kernels
   - AIE Kernels
2. ContainerD calls to a service to program the accelerators
3. RunX assigns the accelerator's resources to the VM

Demo

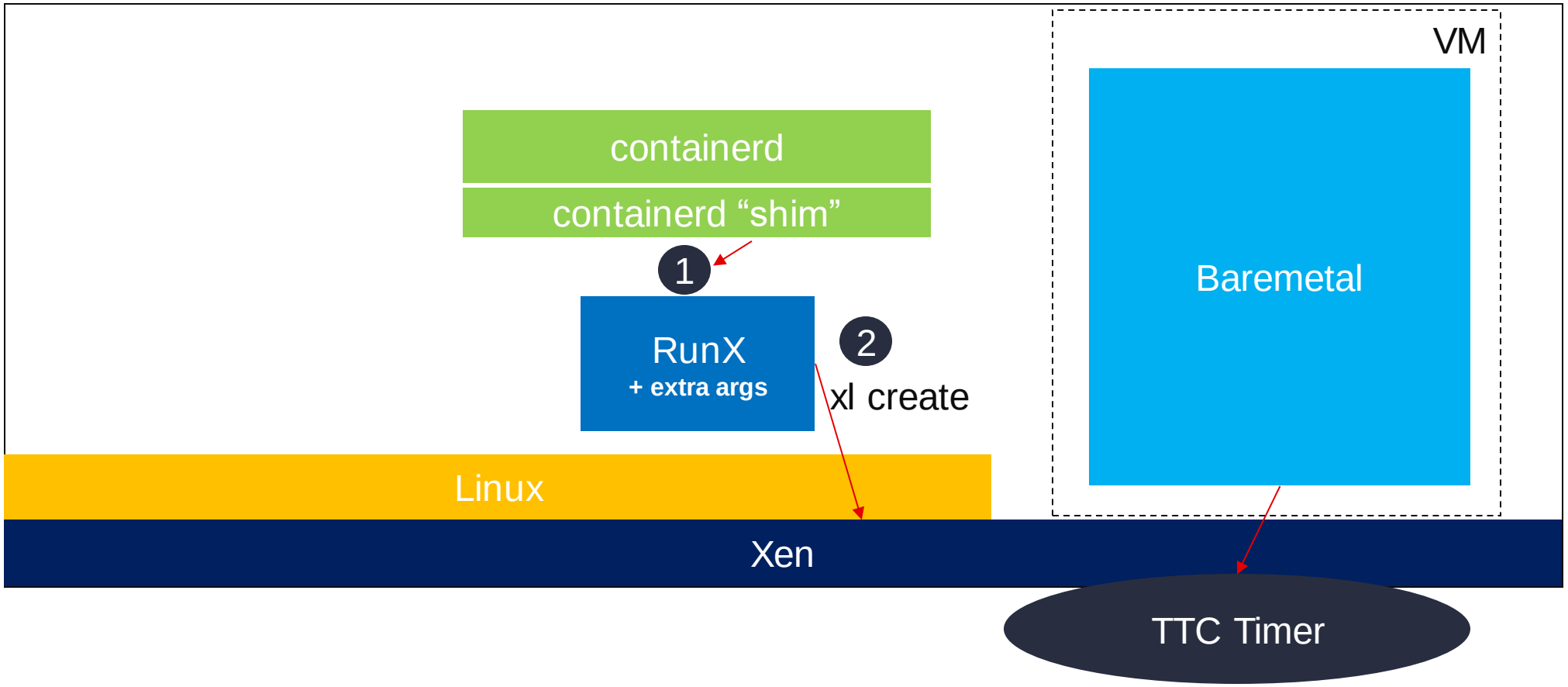
RunX / VxWorks Demo

Deploy VxWorks as a container with RunX



RunX Device Assignment Demo

Baremetal Container with access to the physical TTC timer



XILINX.

Thank You