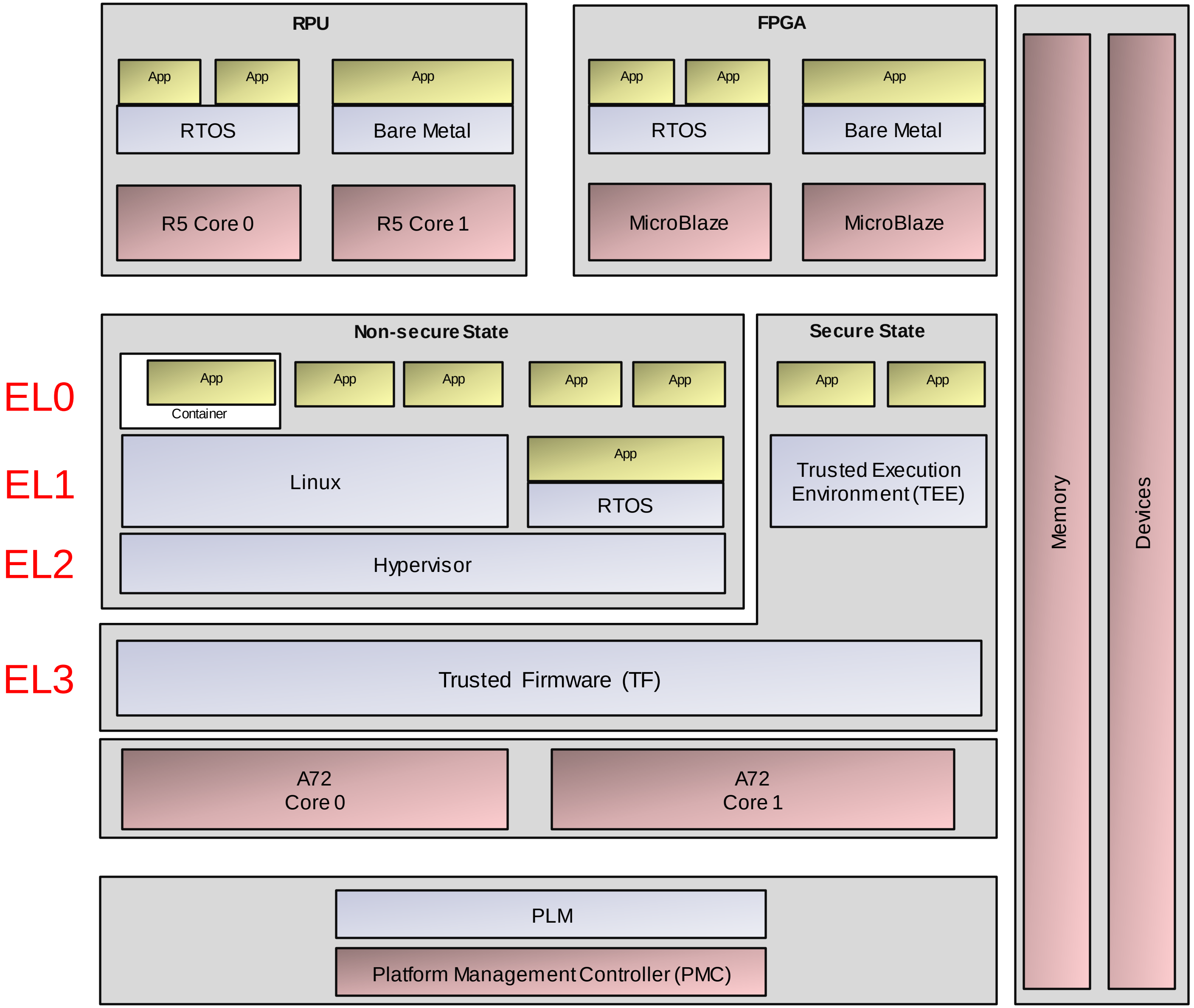
Lopper

Bruce Ashfield & Stefano Stabellini

ELC NA 2022

System Device Tree & Heterogeneous computing

- Heterogeneous System with multiple HW components
  - A72s, R5s, PMC, MicroBlaze clusters
- Multiple Execution Domains with its own address map
  - Each domain with its own operating system
  - Multiple execution levels (EL)
  - Multiple Security Environments
  - E.g.: U-Boot, TF-A, Xen, Linux, OPTEE, Zephyr, baremetal
- The system is divided into domains
  - Each domain is an independent operating environment with CPUs, memory, and devices allocated to it
- System Device Tree: extending DT to describe the full system
  - Including multiple heterogeneous CPU clusters
  - Including multiple domains



System Device Tree

- Device Tree expresses HW information relevant to an operating environment
  - Used by U-Boot, Linux, Xen, TF-A and others

- **System Device Tree** is an extension to Device Tree to describe the full system
  - It describes multiple CPU clusters with different address maps
    - Both A72s and R5s
  - It describes multiple domains and the resources allocated to each

- A domain could be:
  - an heterogenous computing unit, e.g. Zephyr on R5s
  - an operating environment at a specific execution level, e.g. OPTEE
  - a virtual machine, e.g. a Xen domain

Lopper: an introduction

- Lopper
  - Is a framework for manipulating System Device Trees and transforming information
  - Original goal / concept was to produce standard devices trees to support existing platforms/OSs
    - Produces any number of outputs: device trees, generated code, custom, etc
  - Flexible development / runtime workflow integration
  - Data driven
- A few details:
  - OpenSource, BSD-3 License
    - **https://github.com/devicetree-org/lopper, https://pypi.org/project/lopper/**
    - Written in python, using pluggable backends (libfdt,dtlib) for device tree manipulations
    - Additional logic components in the future
  - Works with dts, dtb and yaml inputs
  - Supports unit operations (lops) and more complex python assist modules
    - Depending on the task, both can be used
  - Flexible output / input is provided via python assists
  - Performs validation and consistency checking

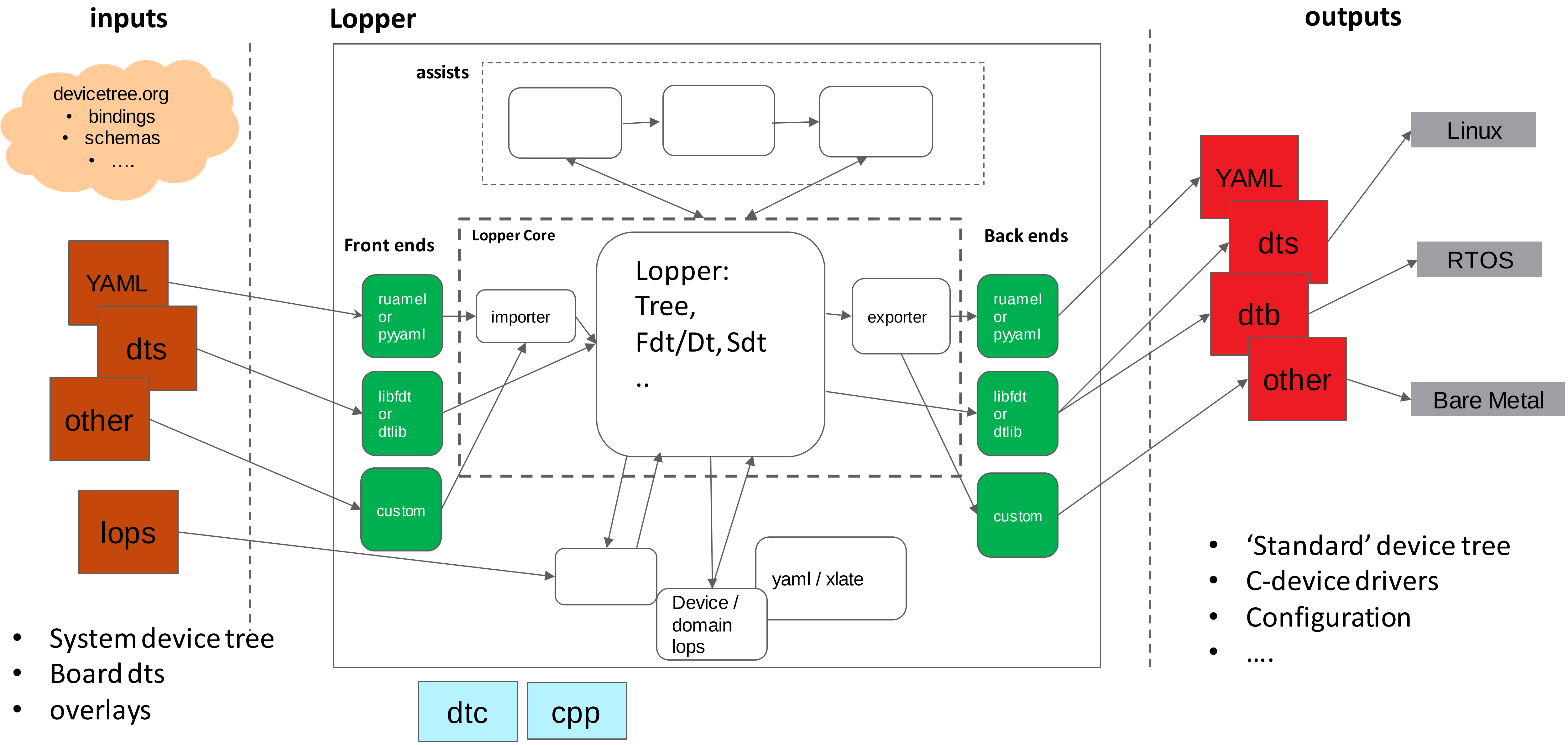
Lopper: a framework

- Common base for tooling inquiring or manipulating device trees
- Built-in:
  - Core tree manipulation, merging
  - Input and output: dts / dtb / yaml
  - Parsing: libfdt or dtlib
  - Assist / lop management
  - Device tree sanity and consistency checking
  - Tree manipulation library / routines

Lopper: a framework

- Optional / plugin:
  - Assists provide Logic / semantics / context awareness
  - Front ends
    - domains / protection, yaml expansion
  - Backends / Assists
    - System Device tree pruning
    - RTOS/Bare Metal backend creates #defines into #include files (early availability)
    - OpenAMP plugin that generates device specific DT for shared pages/IRQs
    - Security / partitioning backend for subsystem and firewall information (TBD)
    - Default Domain specification plugin creates a YAML file that can later be edited (under development)
    - Verification assist that compares DTs from different domains to make sure they are compatible (TBD)
    - Sub device tree / overlay extraction
  - Detailed tree analysis / modification
  - Source validation and expansion
  - ReST API: an HTTP server to support GUIs and other tooling

Lopper: components



- System device tree
- Board dts
- overlays

- 'Standard' device tree
- C-device drivers
- Configuration
- ….

Xen Partial Device Trees

- Similar to device tree overlays but older

- Everything under the top-level *passthrough* node is copied to the guest device tree

- Xen reads partial device trees today to:
  - configure guest device assignment
  - describe passthrough hardware to the guest

- Starting from a copy of the host device tree node and editing it
  - Removing unwanted properties
  - Adding Xen specific properties

```
passthrough {
		compatible = "simple-bus";
		ranges;
		#address-cells = <0x2>;
		#size-cells = <0x2>;

		timer@ff110000 {
				compatible = "cdns,ttc";
				status = "okay";
				interrupt-parent = <0xfde8>;
				interrupts = <0x0 0x24 0x4 0x0 0x25 0x4 0x0 0x26 0x4>;
				reg = <0x0 0xff110000 0x0 0x1000>;
				timer-width = <0x20>;
				xen,force-assign-without-iommu = <1>;
				xen,reg = <0x0 0xff110000 0x0 0x1000 0x0 0xff110000>;
				xen,path = "/axi/timer@ff110000";
		};
};
```

Xen Partial Device Trees

- Xen specific properties and quirks:
  - **xen,reg** to specify memory mappings
  - **xen,path** points to the corresponding host device tree node (used for IOMMU configurations)
  - **xen,force-assign-without-iommu** when IOMMU configuration is not necessary
  - interrupt-parents = <**0xfde8**> to point to the virtual interrupt controller of the guest
  - no **iommus**, because the IOMMU is used by Xen and not exposed to the guest
  - **xen,passthrough;** in the corresponding host device tree node to mark it for assignment

```
timer@ff110000 {
          compatible = "cdns,ttc";
          status = "okay";
          interrupt-parent = <0xfde8>;
          interrupts = <0x0 0x24 0x4 0x0 0x25 0x4 0x0 0x26 0x4>;
          reg = <0x0 0xff110000 0x0 0x1000>;
          timer-width = <0x20>;
          xen,force-assign-without-iommu = <1>;
          xen,reg = <0x0 0xff110000 0x0 0x1000 0x0 0xff110000>;
          xen,path = "/axi/timer@ff110000";
};
```

Xen Partial Device Trees: the problems

- Xen specific changes are not simple, but they could be automated more easily

- Xen properties aside, how to generate the partial device tree?
  - Which properties to remove compared to the host device tree node?
  - How to solve clock dependencies? Do we need to include the clock controller too?
  - What about power domains and reset lines?

- Generic problem: affects device assignment on any hypervisor and heterogeneous domains too

Xen Partial Device Trees: a more complex example

```
zynqmp-firmware {
            compatible = "xlnx,zynqmp-firmware", "xlnx,zynqmp";
            method = "smc";
            #power-domain-cells = <0x1>;
            phandle = <0x1>;

            clock-controller {
                        u-boot,dm-pre-reloc;
                        #clock-cells = <0x1>;
                        compatible = "xlnx,zynqmp-clk";
                        clocks = <0x6 0x7 0x8 0x9 0xa>;
                        clock-names = "pss_ref_clk", "video_clk",
                                    "pss_alt_ref_clk",
                                    "aux_ref_clk", "gt_crx_ref_clk";
                        phandle = <0x3>;
            };
};

pss_ref_clk {
            compatible = "fixed-clock";
            #clock-cells = <0x0>;
            clock-frequency = <0x1fc9350>;
            phandle = <0x6>;
};
```

```
video_clk {
            compatible = "fixed-clock";
            #clock-cells = <0x0>;
            clock-frequency = <0x1fc9f08>;
            phandle = <0x7>;
};

pss_alt_ref_clk {
            compatible = "fixed-clock";
            #clock-cells = <0x0>;
            clock-frequency = <0x0>;
            phandle = <0x8>;
};


[…]

mmc@ff170000 {
            compatible = "xlnx,zynqmp-8.9a", "arasan,sdhci-8.9a";
            status = "okay";
            interrupt-parent = <0xfde8>;
            interrupts = <0x0 0x31 0x4>;
            reg = <0x0 0xff170000 0x0 0x1000>;
            xlnx,device_id = <0x1>;
            clock-names = "clk_xin", "clk_ahb";
            #clock-cells = <0x1>;
            clock-output-names = "clk_out_sd1", "clk_in_sd1";
            clocks = <0x3 0x37 0x3 0x1f>;
            phandle = <0x37>;
            xen,reg = <0x0 0xff170000 0x0 0x1000 0x0 0xff170000>;
            xen,path = "/axi/mmc@ff170000";
};
```

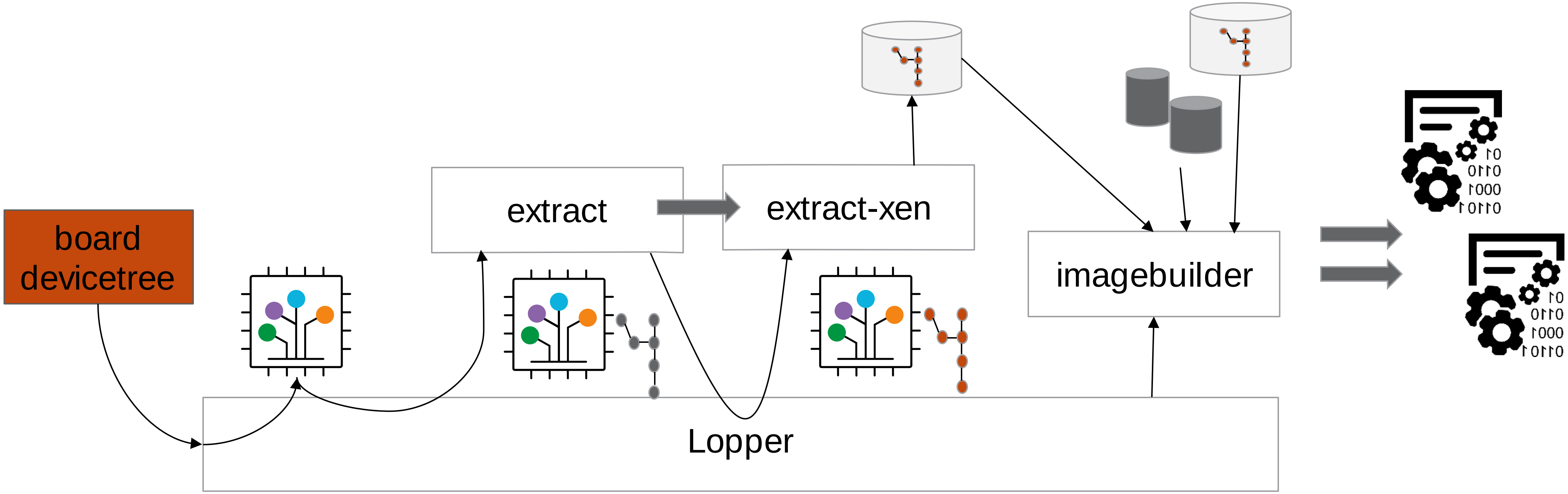
The Solution with Lopper

- Leverage core Lopper functionality + assists
  - Read the device tree -> LopperTree
  - Analyze the devices
  - Manipulate the tree
  - Output dts/dtbs for passthrough devices
  - Trigger image generation (optional)

- Requirements:
  - No hardcoded Xen knowledge
    - Inputs + devicetree properties as triggers
  - Split functionality into generic / resuable components
  - Use Lopper Library routines where possible
  - Data driven + command line options for flexibility
  - Works for any device that can be passed through

Implementation

- A pipeline of assists to extract device tree nodes and their dependencies
  - extract: Generates a partial / extracted device tree starting from a target node
  - extract-xen: Converts generic extracted tree to a Xen understood format
  - Imagebuilder: Takes extracted dtbs and generates boot artifacts



Implementation: Assists

```
Usage: extract -t <target node> [OPTION]
     -t       target node (full path)
     -i       include node if found in extracted node paths
     -p       permissive matching on target node (regex)
     -v       enable verbose debug/processing
     -x       exclude nodes or properties matching regex
     -o       output file for extracted device tree
```

```
Usage: extract-xen -t <target node> [OPTION]
     -t       target node (full path)
     -p       permissive matching on target node (regex)
     -v       enable verbose debug/processing
     -o       output file for extracted device tree
```

```
Usage: image-builder [--uboot] -o <output dir> --imagebuilder <path to imagebuilder>

   wrapper around imagebuilder (https://gitlab.com/xen-project/imagebuilder)

     --uboot  execute imagebuilder’s ”uboot-script-gen”, with the
              options: -t tftp -c ./config, and the supplied output directory
     -i       path to imagebuilder clone
     -v       enable verbose debug/processing
     -o       output directory for files
```

Demo

Lopper and Yocto

- Recipe is part of the meta-virtualization layer

- Current use cases:
  - Configuration generation from device tree
  - Modification of qemu device trees for Xen boot support
    - See: qemuboot-xen-dtb.bbclass

- Future:
  - Map devices to kernel configuration
  - Boot artifacts generation
  - …

Future Work

- Runtime dynamic device tree extraction -> overlay
  - Containers
  - Partitioning
- Validation and smart comparisions
- Common library / routines:
  - Code / driver generation
  - Device analysis and export
- Replace scripts / custom tools and solutions
- Generic image configuration and generation wrapper
- Improved schema integration
- Extend Yocto Project integration

Questions?

AMD