XILINX

System Device Tree & Lopper

Stefano Stabellini
Tomas Evensen
Bruce Ashfield
LVC20

XILINX

Problem Statement

- Allocation and configuration of HW resources are complex
  - Typically done today in an ad-hoc way depending SoC and OS/Firmware
  - Especially tricky to define shared resources, such as OpenAMP/virtIO buffers

- Industry standards and common tools are needed to simplify configuration
  - Define resource allocation to Domains in one place, including shared buffers
  - Allows common flows for any SoC and Operating system/Firmware
  - Simplifies integration of OSes, Hypervisors and Firmware from different places and vendors

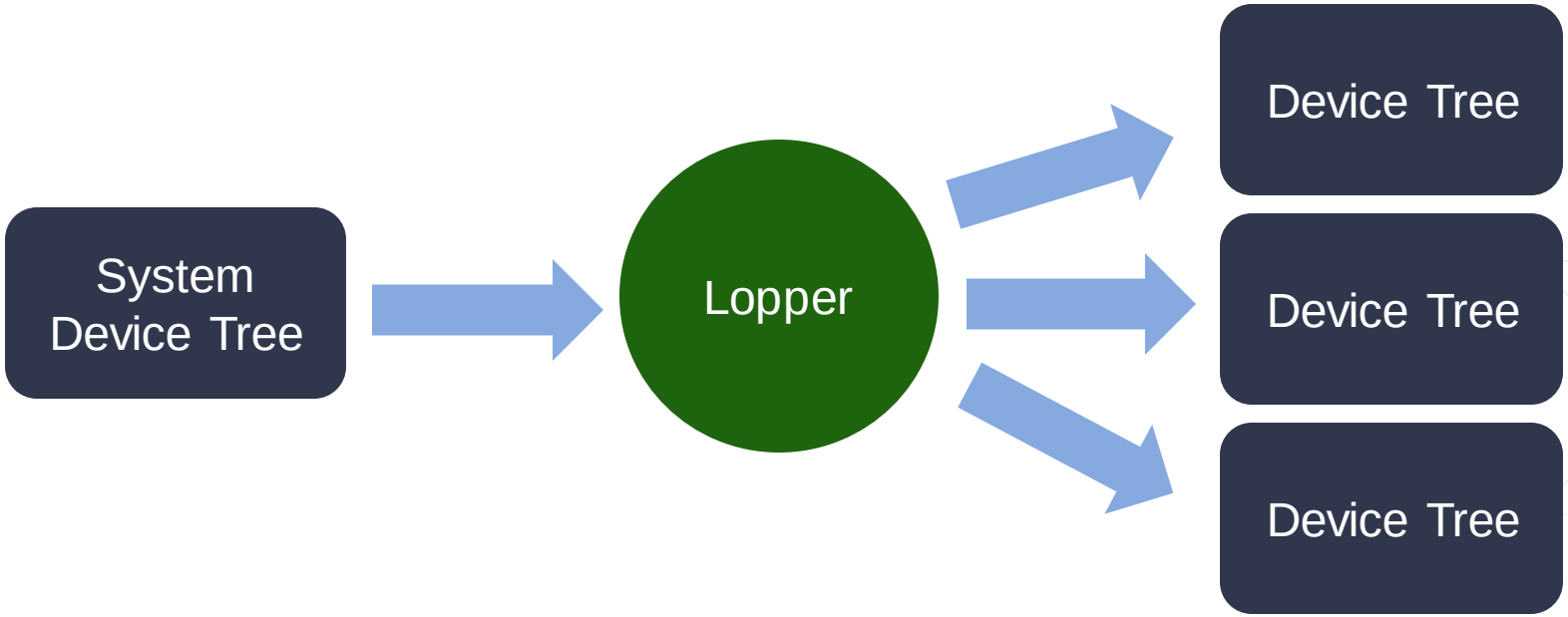
- Verification is critical at integration time and/or runtime
  - Make sure that different Domains are not erroneously using the same resources

Device Trees and System Device Trees

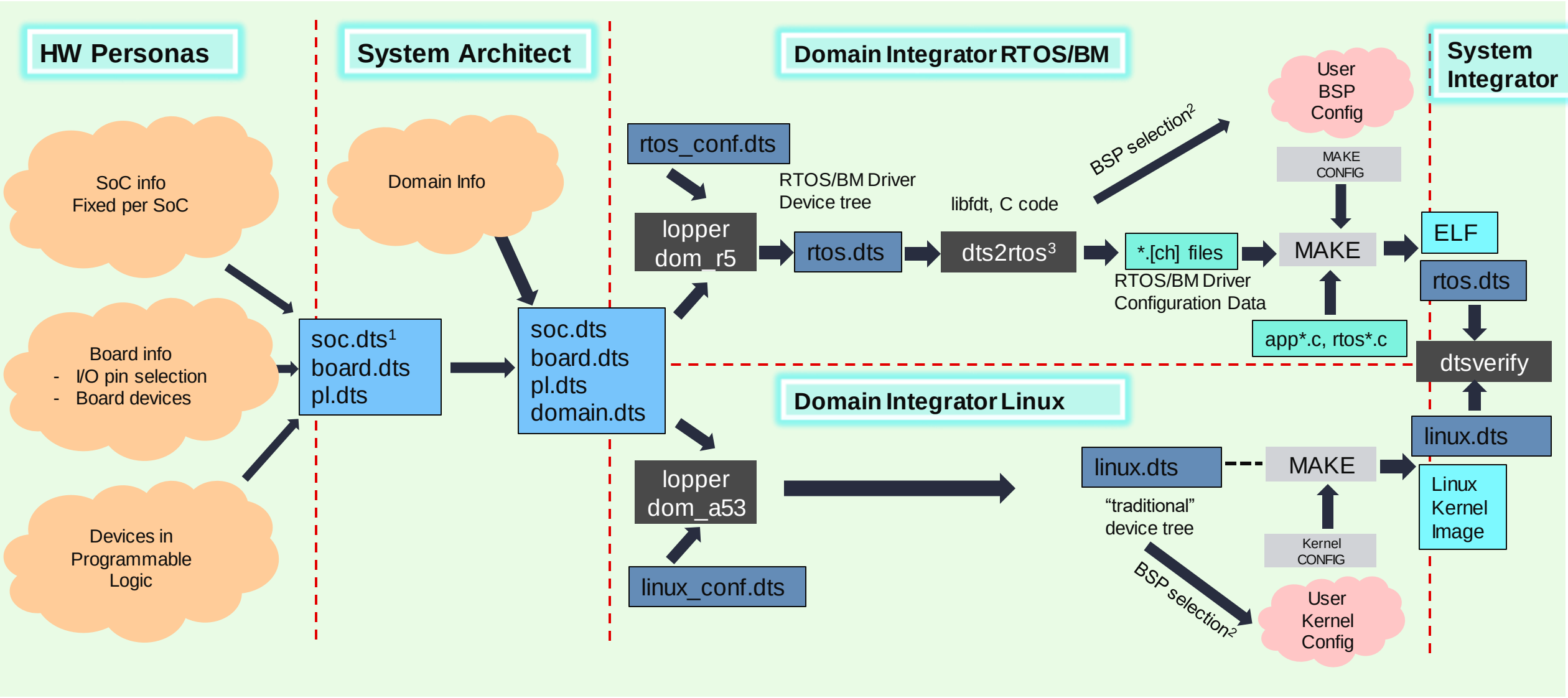
- **Device Trees describes HW nodes and topologies**
  - ***Traditional Device Trees are only describing the world seen from one Address Space***
- **Additional system level Device Tree information is proposed**
  - A **System Device Tree (S-DT)** describes all HW that later can be divided into different partitions
- System DT additions include two parts:
  1. Hardware *Description*
  - Describing multiple cpu clusters and corresponding views of their address spaces
  - Enabling source-to-source translations by adding options to keep labels and comments
  2. AMP *Configuration* information
  - Resource allocation using *domains* and *resource groups*
  - Specification of shared resources, such as pages for virtio buffers
  - Aligned with hypervisor information (e.g. Xen Dom0-less configuration)

Open Source Tool (Lopper) to Manipulate S-DT

- Built on top of existing Device Tree tooling and libraries (DTC, libfdt)
- Manipulates S-DTs and DTS by
  - Prunes System Device Trees to traditional/partitional Device Trees
    - Removes nodes not used, resolves address mapping issues
    - Other transformation (clocks, phandles, …)
    - Use *domain* configuration to select what to be kept/pruned
- Additional features
  - Can be run multiple times, potentially by different personas, supporting different flows
  - Keeps as much as possible of original information, including labels and comments
  - Supports YAML as alternative input/output
    - AMP configuration can be provided in YAML


System
Device Tree
Lopper
Device Tree
Device Tree
Device Tree

System Device Tree Data Flow (RTOS & Linux)



Standardized Data files | Open source tool | New open source code | RTOS specific data

1) .dts files before lopper is a System Device Tree
2) The .dts can be used to select drivers
3) RTOS specific transformations

XILINX.

Specification

XILINX

Latest Update

- Multi-View Multi-OS hardware description
  - *cpus,cluster*
  - *address-map*
  - *indirect-bus*
- AMP Configuration
  - execution domains (*openamp,domain*)
  - unique access to hardware resource (*access*)
- Lopper upstream at devicetree.org

```
cpus_r5: cpus-cluster@0 {
        compatible = "cpus,cluster";

        address-map =
                <0xf1000000 &amba 0xf1000000 0xeb00000>,
                 <0xf9000000 &amba_rpu 0xf9000000 0x10000>,
                 <0x0 &memory 0x0 0x80000000>,
                 <0x0 &tcm 0xFFE90000 0x10000>;

        cpu@0 {
        };

        cpu@1 {
        };
};


domains {
        domain@1 {
                compatible = "openamp,domain-v1";

                cpus = <&cpus_r5 0x3 0x80000001>;
                memory = <0x0 0x100000 0x0 0x400000>;

                access = <&can0 0x80000000>;
        };
```

Bus-Firewalls in System Device Tree: Goals

- describe bus firewall controllers
- describe which device is protected by which controller
- describe a new ID space to configure the firewalls
- describe domains-based firewall configurations in System Device Tree

Bus-Firewall Controllers

- a node describing a firewall controller
  - there can be multiple controllers in a system
- a #firewall-cells property to define firewall-specific extra information
  - #firewall-cells can be 0
- a firewall property under each device node linking to the appropriate firewall controller
  - it links to the firewall controller protecting accesses *to the device MMIO regions*

Example

```
amba_xppu: indirect-bus@1 {
        compatible = "indirect-bus";
        #address-cells = <0x2>;
        #size-cells = <0x2>;

        lpd_xppu: xppu@ff990000 {
            compatible = "xlnx,xppu"
            #firewall-cells = <0x0>;
            reg = <0x0 0xff990000 0x0 0x1000>;
        };
        pmc_xppu: xppu@f1310000 {
            compatible = "xlnx,xppu"
            #firewall-cells = <0x0>;
            reg = <0x0 0xf1310000 0x0 0x1000>;
        };
    };

    amba {
        ethernet0: ethernet@ff0c0000 {
            bus-master-id = <&lpd_xppu 0x234 &pmc_xppu 0x234>;
            firewall-0 = <&lpd_xppu>;
        };

        can0: can@ff060000 {
            firewall-0 = <&lpd_xppu>;
        };

        mmc0: sdhci@f1050000 {
            bus-master-id = <&lpd_xppu 0x243 &pmc_xppu 0x243>;
            firewall-0 = <&pmc_xppu>;
        };

        serial0: serial@ff000000 {
            firewall-0 = <&lpd_xppu>;
        };
    };
```

Bus Master IDs

- Bus mastering devices are identified by bus-firewalls using IDs. Their transactions are marked with a device ID. These IDs are used to configure bus-firewalls.
- We shall call these IDs "Bus Master IDs"
- We shall advertise them on device tree using a new property: bus-master-id

```
dev0: device@0 {
        bus-master-id = <&lpd_xppu 0x212>;
```

System Device Tree Hardware Description Example

```
cpus_r5: cpus-cluster@0 {
        #address-cells = <0x1>;
        #size-cells = <0x0>;
        #cpus-mask-cells = <0x1>;
        compatible = "cpus,cluster";

        bus-master-id = <&lpd_xppu 0x0 &pmc_xppu 0x0 &lpd_xppu 0x1 &pmc_xppu 0x1>;
    };

    amba {
        ethernet0: ethernet@ff0c0000 {
            bus-master-id = <&lpd_xppu 0x234 &pmc_xppu 0x234>;
            firewall-0 = <&lpd_xppu>;
        };

        can0: can@ff060000 {
            firewall-0 = <&lpd_xppu>;
        };

        mmc0: sdhci@f1050000 {
            bus-master-id = <&lpd_xppu 0x243 &pmc_xppu 0x243>;
            firewall-0 = <&pmc_xppu>;
        };

        serial0: serial@ff000000 {
            firewall-0 = <&lpd_xppu>;
        };
    };
```

Configuring Bus-Firewalls

- Domains are a natural place to describe the desired firewall configuration
  - they already specify device assignments
  - let's add protection to the assignments
- Introducing:
  - *resource-group* nodes to specify shared resources
  - *firewallconf* and *firewallconf-default* properties to specify the bus-firewall configuration

resource-group

- A way to group together resources shared across multiple domains
- Instead of access, we are using a new property *include* to link to a resource-group

```
domains {
            #address-cells = <0x1>;
            #size-cells = <0x1>;

            resource_group_1: resource_group_1 {
                compatible = "openamp,resource-group-v1";
                access = <&ethernet 0x0>, <&serial0 0x0>;
            };

            domain@0 {
                compatible = "openamp,domain-v1";
                access = <&mmc0 0x0>;
                include = <&resource_group_1>;
            };

            domain@1 {
                compatible = "openamp,domain-v1";
                access = <&can0 0x0>;
                include = <&resource_group_1>;
            };
```

firewallconf property

- *firewallconf = <&domain0 block 0>;*
- it applies to all address ranges in the resource-group or domain it appears in
- the phandle is a link to a node of a bus mastering device
  - lopper will retrieve the bus-master-ids of the linked node for the relevant controllers
  - it is also possible to link to a domain in which case lopper will use:
    - the bus-master-id of every device in the domain access list
    - the bus-master-id of the CPU cluster of the domain
- allow/block/block-desirable
  - whether the bus-master-id range is allowed or blocked access
  - block-desirable means "block it if you can"
- priority: priority of the rule in case of block-desirable is specified, otherwise unused

Example

"Block all bus-master-ids of domain0 from accessing 0x0-0x100000 and the MMIO region of can0."

```
domain1: domain@1 {
    compatible = "openamp,domain-v1";
    memory = <0x0 0x100000>;
    access = <&can0 0x0>;
    firewallconf = <&domain0 block 0>;
};
```

firewallconf-default

- firewallconf-default applies to all bus-master-ids except for the ones listed in the firewallconf properties

```
"Block all bus-master-ids except for the ones of domain@0 and domain@1"


firewallconf-default = <block-desirable 8>,
firewallconf = <&domain@0 allow 0>, <&domain@1 allow 0>;
```

Example

"Two domains, blocking each other's access to their resources."

```
domains {
    #address-cells = <0x1>;
    #size-cells = <0x1>;

    domain0: domain@0 {
        compatible = "openamp,domain-v1";
        memory = <0x100000 0x100000>;
        access = <&mmc0 0x0>;
        firewallconf-default = <allow 0>;
        firewallconf = <&domain1 block 0>;
    };

    domain1: domain@1 {
        compatible = "openamp,domain-v1";
        memory = <0x0 0x100000>;
        access = <&can0 0x0>;
        firewallconf-default = <allow 0>;
        firewallconf = <&domain0 block-desirable 5>;
    };
```

Example

"Two domains sharing two key resources. The two resources are accessible by the two domains using them, but everybody else is prevented from accessing them. The two domains also want to protect themselves against foreign access but at a lower priority."

```
domains {
    #address-cells = <0x1>;
    #size-cells = <0x1>;

    resource_group_1: resource_group_1 {
        compatible = "openamp,group-v1";
        access = <&ethernet 0x0>, <&serial0 0x0>;
        firewallconf-default = <block 0>;
    };

    domain0: domain@0 {
        compatible = "openamp,domain-v1";
        memory = <0x100000 0x100000>;
        access = <&mmc0 0x0>;
        include = <&resource_group_1>;
        firewallconf-default = <block-desirable 8>;
    };

    domain1: domain@1 {
        compatible = "openamp,domain-v1";
        memory = <0x0 0x100000>;
        access = <&can0 0x0>;
        include = <&resource_group_1>;
        firewallconf-default = <block-desirable 8>;
    };
```

Example

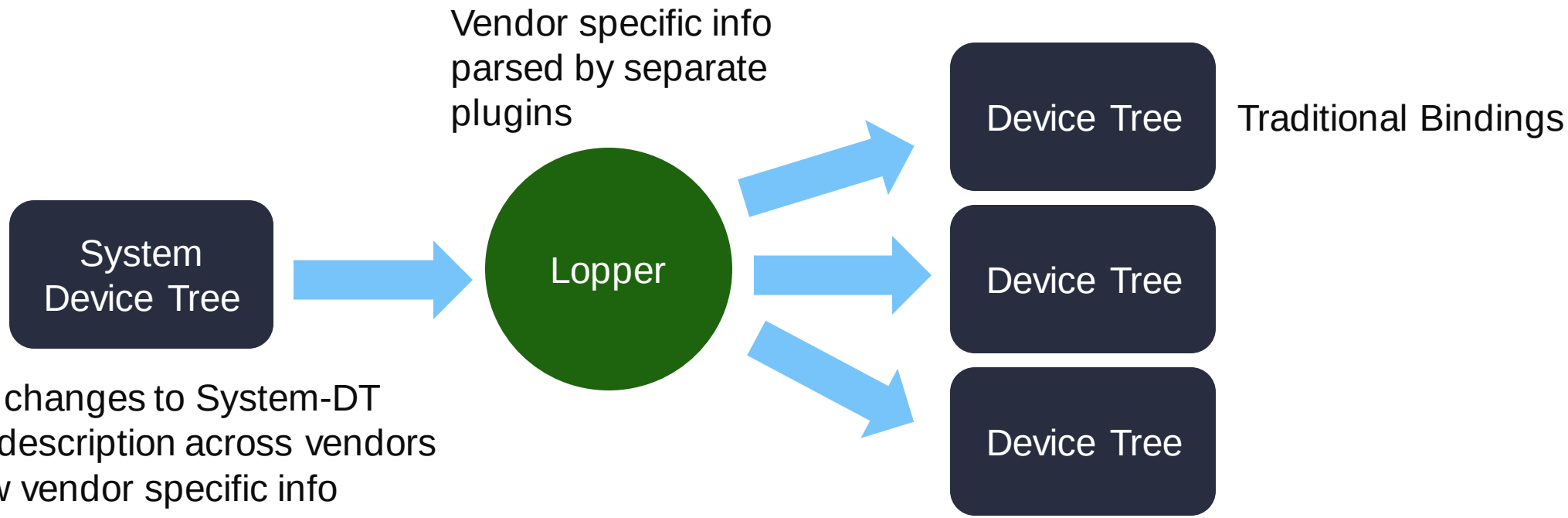
"Three domains with one privileged over the others."

```
        domains {
            #address-cells = <0x1>;
            #size-cells = <0x1>;

            domain0: domain@0 {
                compatible = "openamp,domain-v1";
                memory = <0x100000 0x100000>;
                access = <&mmc0 0x0>;
                firewallconf-default = <block 0>;
            };

            domain1: domain@1 {
                compatible = "openamp,domain-v1";
                memory = <0x0 0x100000>;
                access = <&can0 0x0>;
                firewallconf-default = <block-desirable 8>;
                firewallconf = <&domain0 allow 0>;
            };

            domain2: domain@2 {
                compatible = "openamp,domain-v1";
                memory = <0x0 0x100000>;
                access = <&can0 0x0>;
                firewallconf-default = <block-desirable 8>;
                firewallconf = <&domain0 allow 0>;
            };
```

System Device Tree & RemoteProc

- Under discussion
- System Device Tree Goals for RemoteProc:
  - RemoteProc information in a single place
  - Minimalistic: provide as little information as possible to enable it
  - Reduce vendor-specific details
  - Allow current bindings in traditional device tree -- no changes required to existing device trees
- **Vision:**
  - Generic way of describing RemoteProc bindings in device tree

OpenAMP RemoteProc Flow with System-DT



- Minimal changes to System-DT
- Shared description across vendors
- Only few vendor specific info

OpenAMP RemoteProc Flow with System-DT

- System Device Tree RemoteProc
  - Use common System Device Tree concepts
  - Vendor-specific info reduced to one 32-bit field
- Lopper's plugins generates today's RemoteProc bindings

```
resource_group: resource_group@0 {
        compatible = "openamp,remoteproc-v1", "openamp,group-v1";
        memory = <0x0 0x3ed40000 0x0 0x4000
                  0x0 0x3ed44000 0x0 0x4000
                  0x0 0x3ed48000 0x0 0x100000
                  0x0 0x3ed00000 0x0 0x40000>;
        access = <&tcm 0x0>;
};

domain@1 {
        access = <&ipi_mailbox_rpu0 0x13>;
        /* 0x1: master */
        include = <&resource_group 0x1>;
};
```

Current State of RemoteProc bindings

- Existing RemoteProc bindings are fragmented across vendors
  - Many have similar concepts but subtly different
- For System Device Tree, no changes are required
  - Lopper's plugins deal with vendors differences; one plugin per vendor
  - Decouple System Device Tree work from RemoteProc bindings work
- Aligning RemoteProc bindings across vendors in the long term would:
  - make the implementation easier in Lopper; potentially remove the need for plugins
  - improve both the bindings and drivers in Linux and RTOSes

Lopper

Lopper: Overview

- https://github.com/devicetree-org/lopper
- Lopper takes an input device tree (normally a system device tree),
- applies operations to that tree, and outputs one or more modified/processed trees.
- Operations may be small transformations, inline python, or full python assists (for input and/or output)
- Modular, and has very few dependencies
- Constructs a normalized tree from inputs (dts, dtb or yaml) and abstracts plugins from the details of reading writing dts/fdt/dtbs/yaml

Lopper: Use Cases

- System device tree to multi-os standarized trees (zero touch for consumer)
- Extracting / Summarizing / Querying information from a device tree
- Removing / Adding / Modifying nodes or properties
- Creating multiple target device trees from a single input
- Validating and checking inputs (partitioning, domains, memory, etc)
- Sanitizing / Cleaning output trees
- Generating custom outputs from device tree (build info, firmware)
- Generating device driver / code
- ….

Lopper: Operations (lops) Examples

- Trivial: remove an invalid property

```
> ./lopper.py -f -i linaro/lops-simple.dts linaro/input-tree.dts linaro/output-tree.dts

> grep special-tag linaro/input-tree.dts
special-tag = <0x2>;

> grep special-tag linaro/output-tree.dts
```

```
/dts-v1/;

/ {
        compatible = "system-device-tree-v1";
        lops {
                // compatible = "system-device-tree-v1,lop";
                lop_1 {
                        compatible = "system-device-tree-v1,lop,modify";
                        modify = "/cpus/:special-tag:";
                };
        };
};
```

Lopper: Operations (lops) Examples

- Advanced: tag and count nodes, print summary

```
lop_17_1 {
      compatible = "system-device-tree-v1,lop,select-v1";
      // clear any old selections
      select_1;
      select_2 = "/cpus/.*:compatible:.*arm,cortex-a72.*";
      // if the path specifier isn't used, we operate on previously selected nodes.
      select_3 = ":cpu-idle-states:3";
};
lop_17_2 {
      compatible = "system-device-tree-v1,lop,modify";
      // this will use the selected nodes above, to add a property
      modify = ":cputag:priority";
};
lop_17_3 {
      compatible = "system-device-tree-v1,lop,code-v1";
      code = "
          print( 'cpus with idle-state 3:' )
          for s in tree.__selected__:
              print( '  cpu tag in (%s): %s' % (s.abs_path, s['cputag'].value[0] ) )";
};
lop_17_4 {
      compatible = "system-device-tree-v1,lop,select-v1";
      select_1;
      select_2 = "/cpus/.*:compatible:.*arm,cortex-a72.*";
      // since there's a path specifier, this is an OR operation
      select_3 = "/amba/.*:phy-handle:0xa";
};
```

Lopper: Operations (lops) Examples

```
lop_17_5 {
compatible = "system-device-tree-v1,lop,code-v1";
code = "
    print( 'selected nodes (cortex a72 or phy-handle: 0xa):' )
    for s in tree.__selected__:
        if re.search( 'amba.*', s.abs_path ):
            print( '  %s {' % s )
            for p in s:
                print( '    %s' % p )
            print( '}' )
        else:
            print( '  %s' % s.abs_path )";
```

```
> ./lopper.py -f -i linaro/lops-count.dts linaro/input-tree.dts linaro/output-tree.dts
cpus with idle-state 3:
  cpu tag in (/cpus/cpu@1): priority
selected nodes (cortex a72 or phy-handle: 0xa):
  /cpus/cpu@0
  /cpus/cpu@1
  /amba/ethernet@ff0d0000 {
    compatible = "cdns,versal-gem";
    phy-handle = <0xa>;
    status = "okay";
    reg = <0x0 0xff0d0000 0x0 0x1000>;
    interrupts = <0x0 0x3a 0x4 0x0 0x3a 0x4>;
    clock-names = "pclk","hclk","tx_clk","rx_clk","tsu_clk";
    ….
}
```

Lopper: Assist Example

- Assists can leverage all of python's capabilities
  - Implement complex logic, modify the tree, produce output, etc
- See OpenAmp and Baremetal assiss in lopper repository:
  - https://github.com/devicetree-org/lopper/blob/master/assists/openamp.py
  - https://github.com/devicetree-org/lopper/blob/master/assists/baremetal_bspconfig_xlnx.py
- Example: 'grep' is a python assist
  - assists/grep.py
  - Implements the is_compat() function as described in Lopper Documentation
    - Indicates for which modules, or nodes its entry function should be executed
  - When matching nodes are found, the routine is called
  - Searches for a pattern and outputs matches

Lopper: Assist Example (grep)

```
...

def is_compat( node, compat_string_to_test ):
    if re.search( "module,grep", compat_string_to_test):
        return grep
    return ""

# tgt_node: is the matching node
# sdt: is the system device tree
def grep( tgt_node, sdt, options ):
    try:
        verbose = options['verbose']
    except:
        verbose = 0

    try:
        args = options['args']
    except:
        args = []

    if verbose:
        print( "[INFO]: cb: grep( %s, %s, %s, %s )" % (tgt_node, sdt, verbose, args))

...
```

Lopper: Assist Example (grep)

```
> ./lopper.py -f linaro/input-tree.dts -- grep compatible.* "/cpus.*"
/cpus: compatible = "cpus,cluster";
/cpus/cpu@0: compatible = "arm,cortex-a72","arm,armv8";
/cpus/cpu@1: compatible = "arm,cortex-a72","arm,armv8";
/cpus/idle-states/cpu-sleep-0: compatible = "arm,idle-state";
```

Adaptable.
Intelligent.