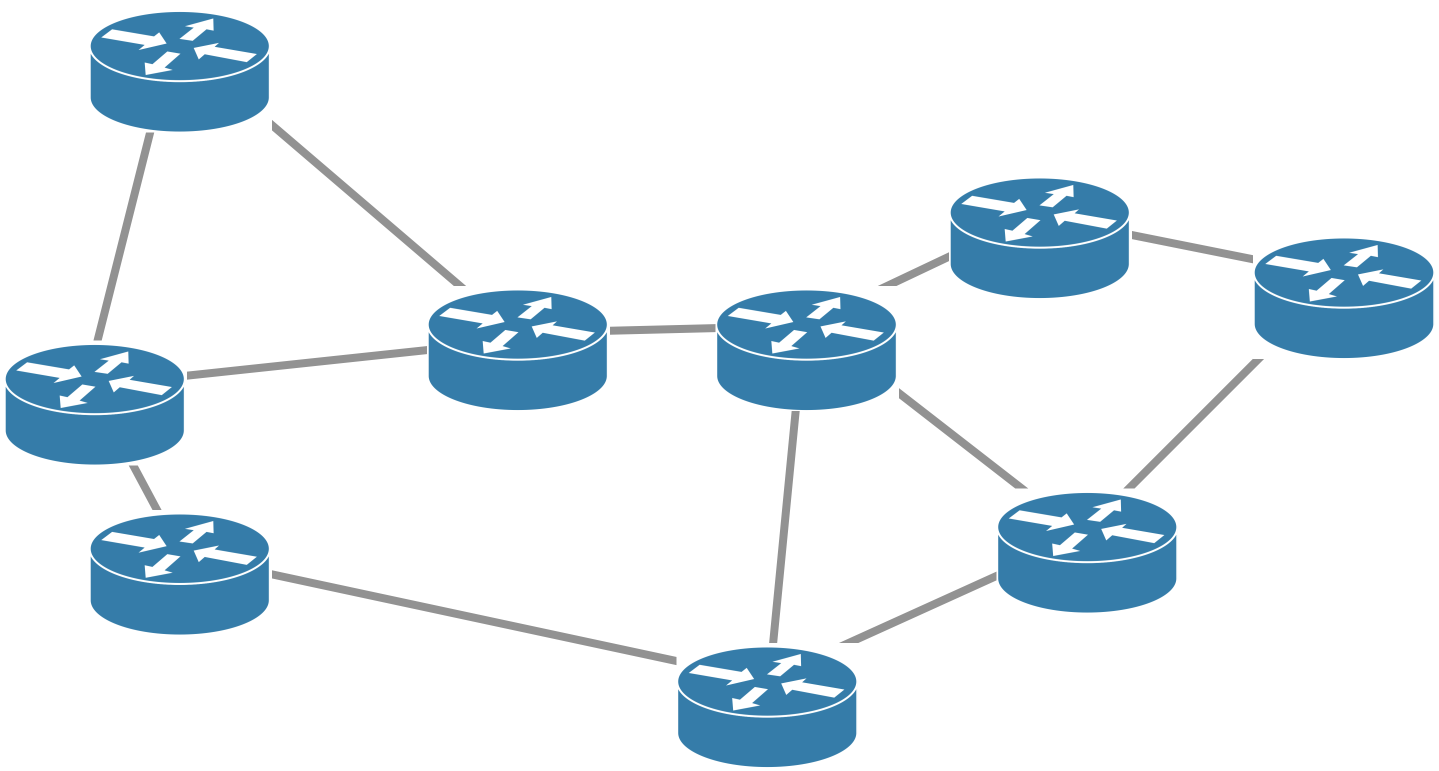
Towards *validated* network configurations with NCGuard



Laurent Vanbever, Grégory Pardoen and Olivier Bonaventure

http://inl.info.ucl.ac.be

WODNAFO'10, Adelaide

Mon 8 Feb 2010

Human factors are responsible for ***50*** to ***80*** percent of network device outages

Juniper Networks, *What's Behind Network Downtime?*, 2008

Configuring networks is like writing a distributed program in assembly language

Sihyung Lee, ICC, 2008

Current approaches could be divided into ***static analysis*** and data mining

Use pattern matching to find *known* misconfigurations

For example, look for typing error in network advertisement

Compare configurations to given specifications

For example, every router must belong to the iBGP full-mesh

Pros and cons

- Effective
- You need to know what a valid network is
- How do you deal with heterogenous languages ?

Current approaches could be divided into static analysis, and ***data mining***

Statistical analysis of configurations

e.g., throw error if an instruction is defined everywhere but on one device

Infer network-specific policies for deviation analysis

Try to understand the meaning of the network

Pros and cons

- Completely independent of *a priori* specifications
- Too verbose. People are flooded with false positives
- How do you deal with heterogenous languages ?

This situation *contrasts* with development in software engineering

Requirements describe precisely systems behavior

lack of equivalence in network configuration

Validation techniques for systematic error detection

currently, devices perform only *syntax* validation

New development schemes improve efficiency

the CLI approach hasn't change very much since ~1990

Our approach: a *high-level* representation with a *validation* and *generation* engine

High-level representation abstracts useless details

it could be used as a *documented* view of a network

Validation (rules-based) ensures specifications are respected

and that they *will be* respected in the future

Generation produces low-level configurations

that are understandable by the components

NCGuard follows a *top-down* approach

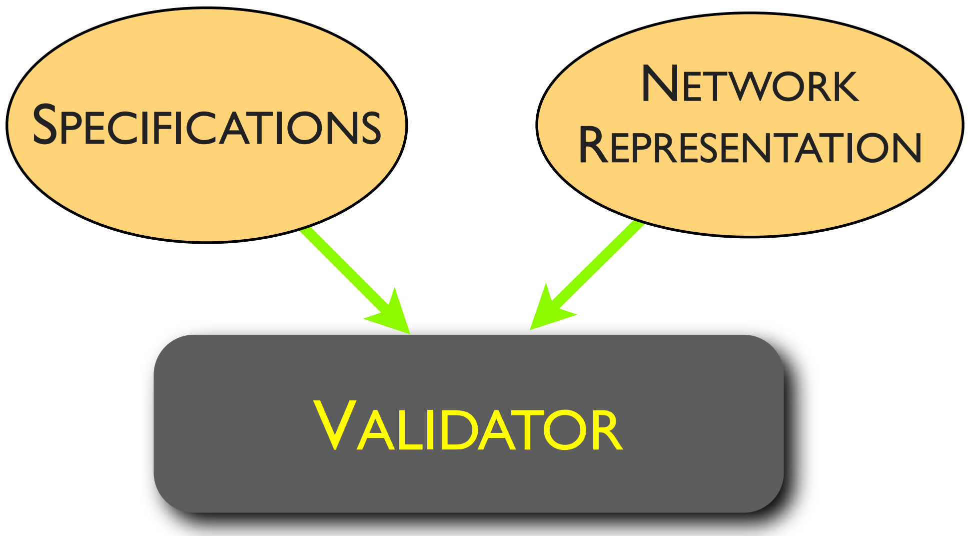
NCGuard follows a *top-down* approach

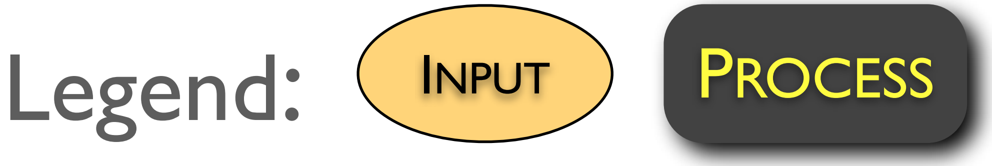
SPECIFICATIONS

NETWORK REPRESENTATION

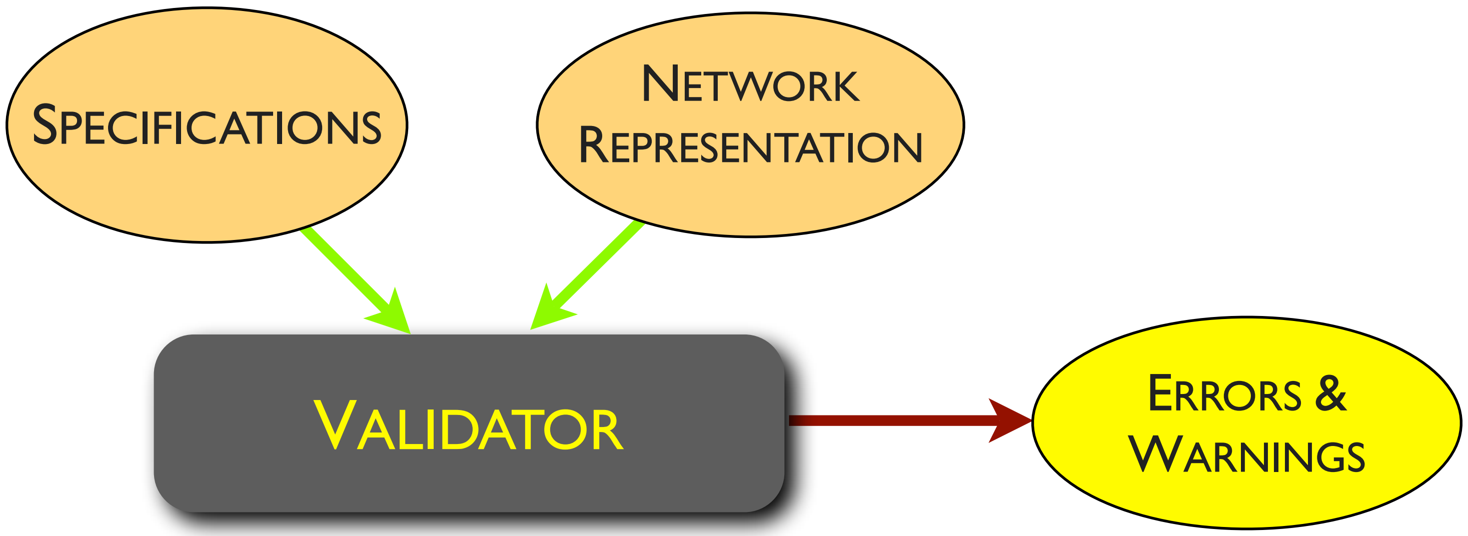
Legend: INPUT

NCGuard follows a *top-down* approach



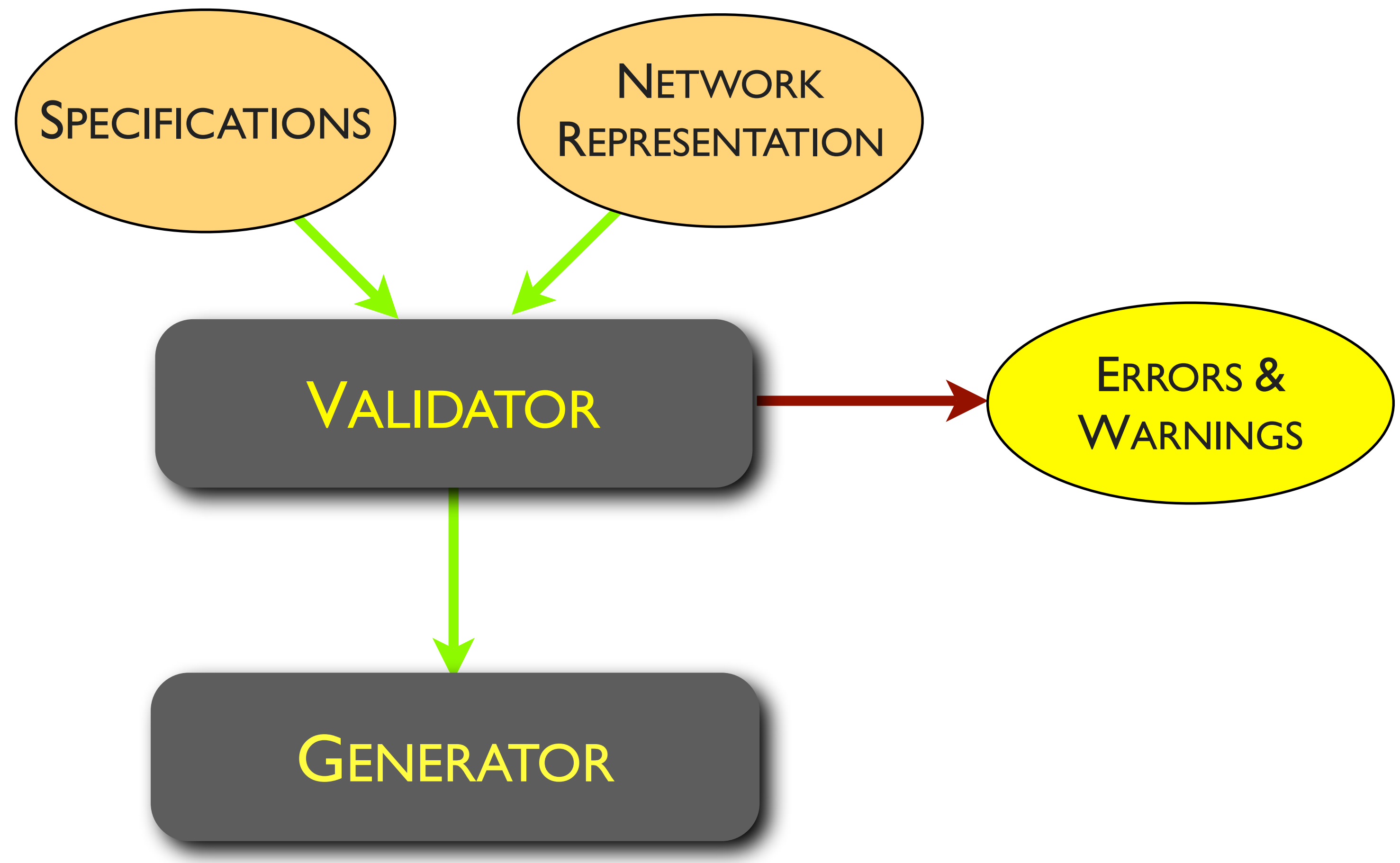


NCGuard follows a *top-down* approach

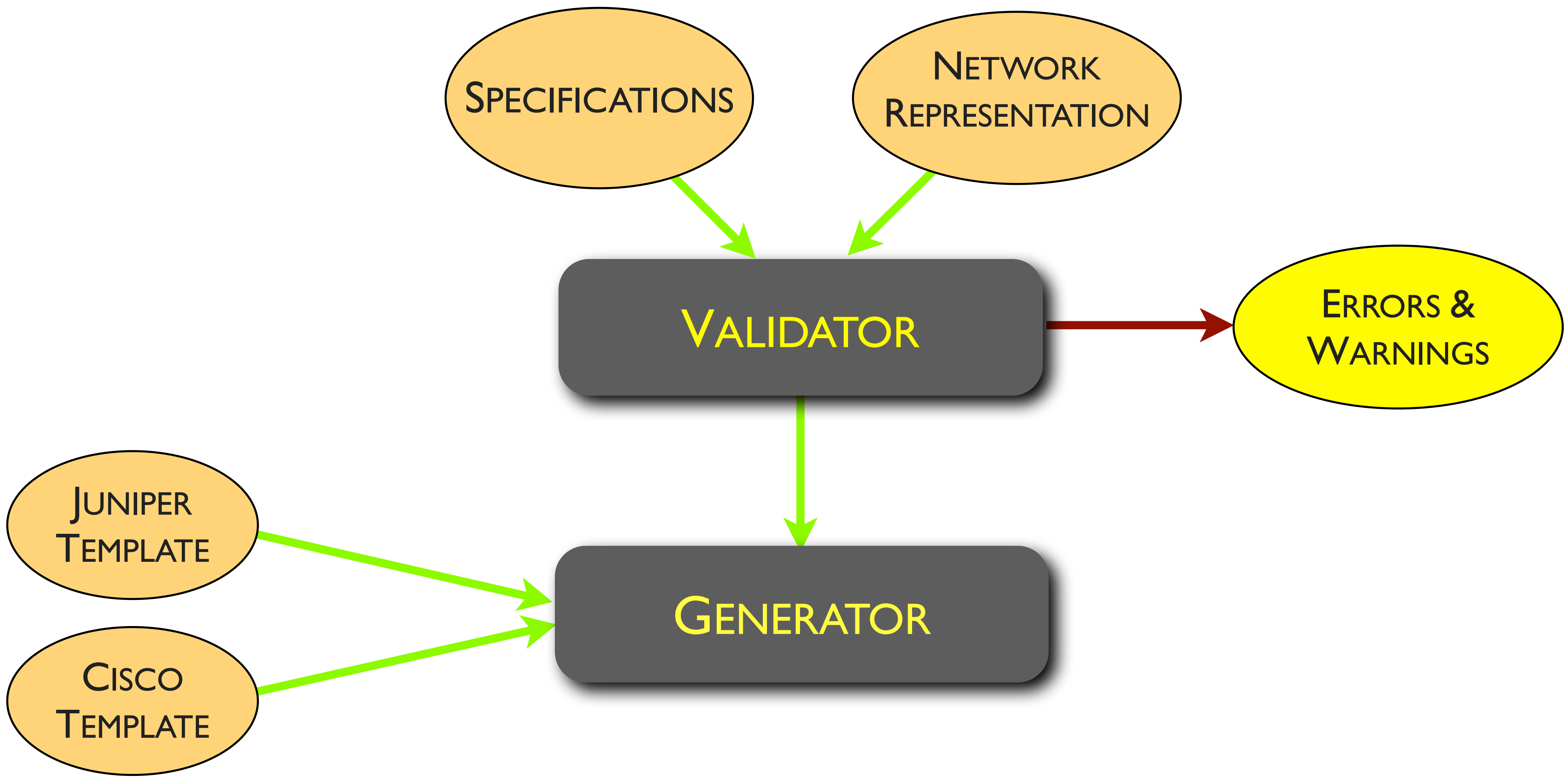


Legend:

NCGuard follows a *top-down* approach

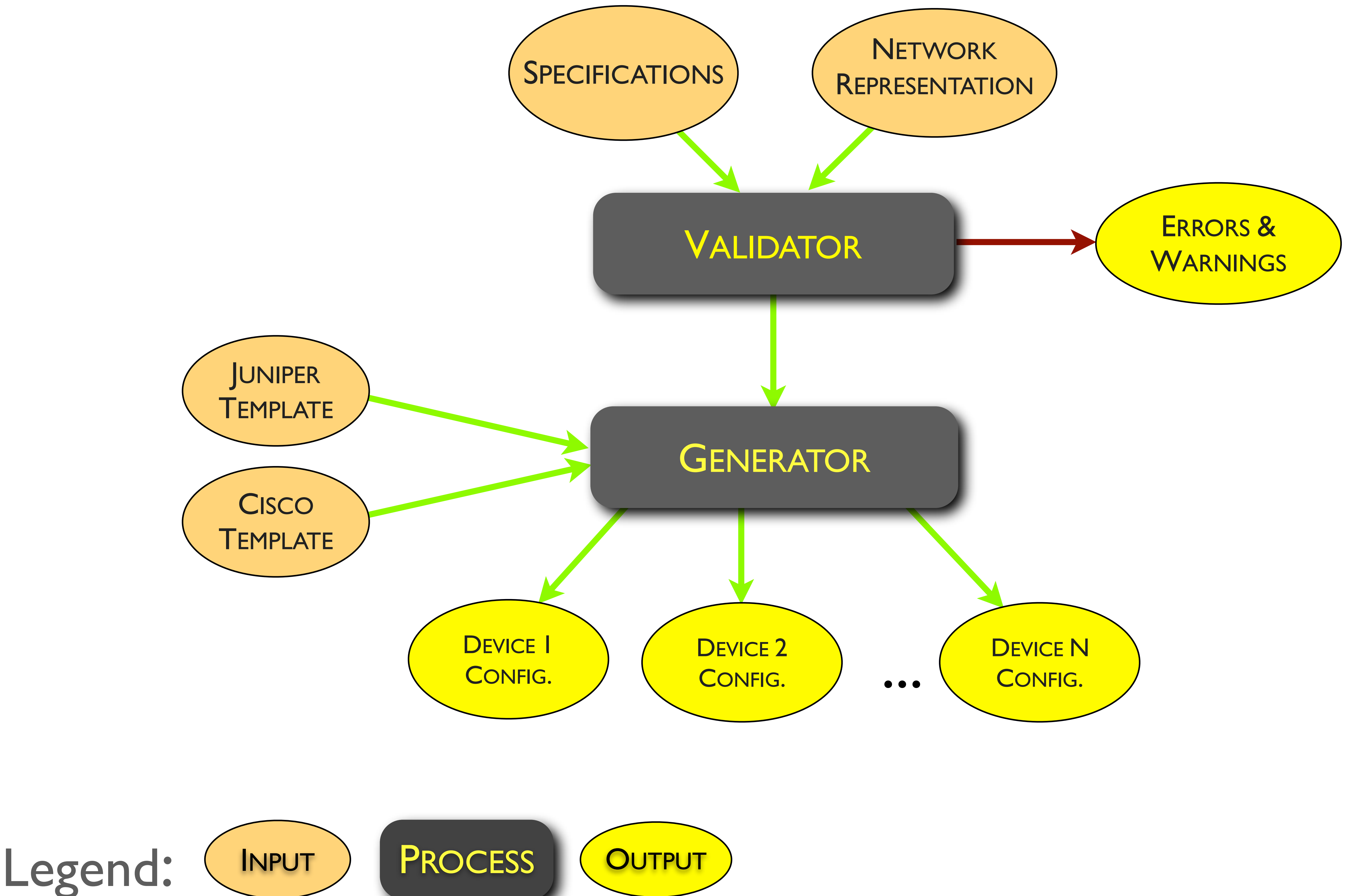


NCGuard follows a *top-down* approach

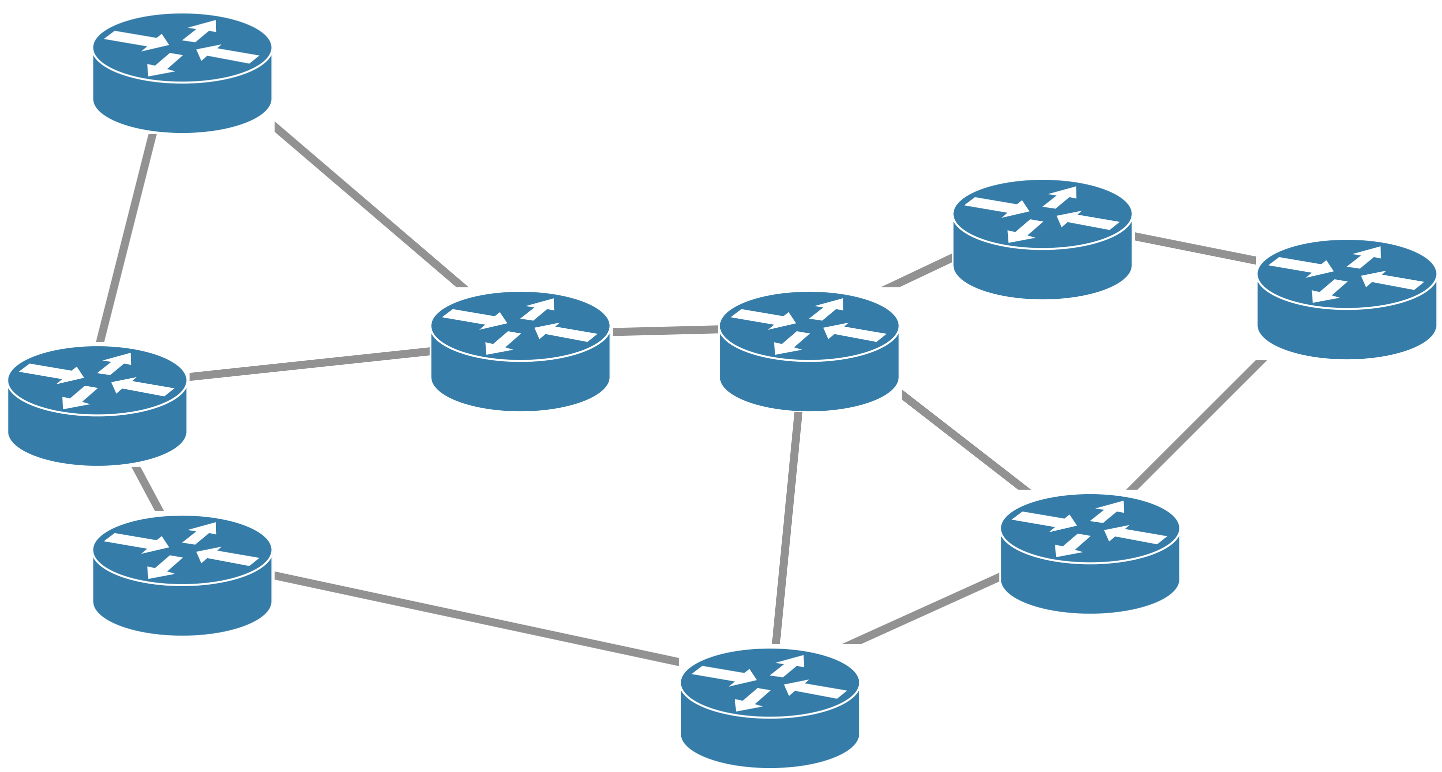


Legend:

NCGuard follows a *top-down* approach



Towards *validated* network configurations



High-level representation

Hide useless details

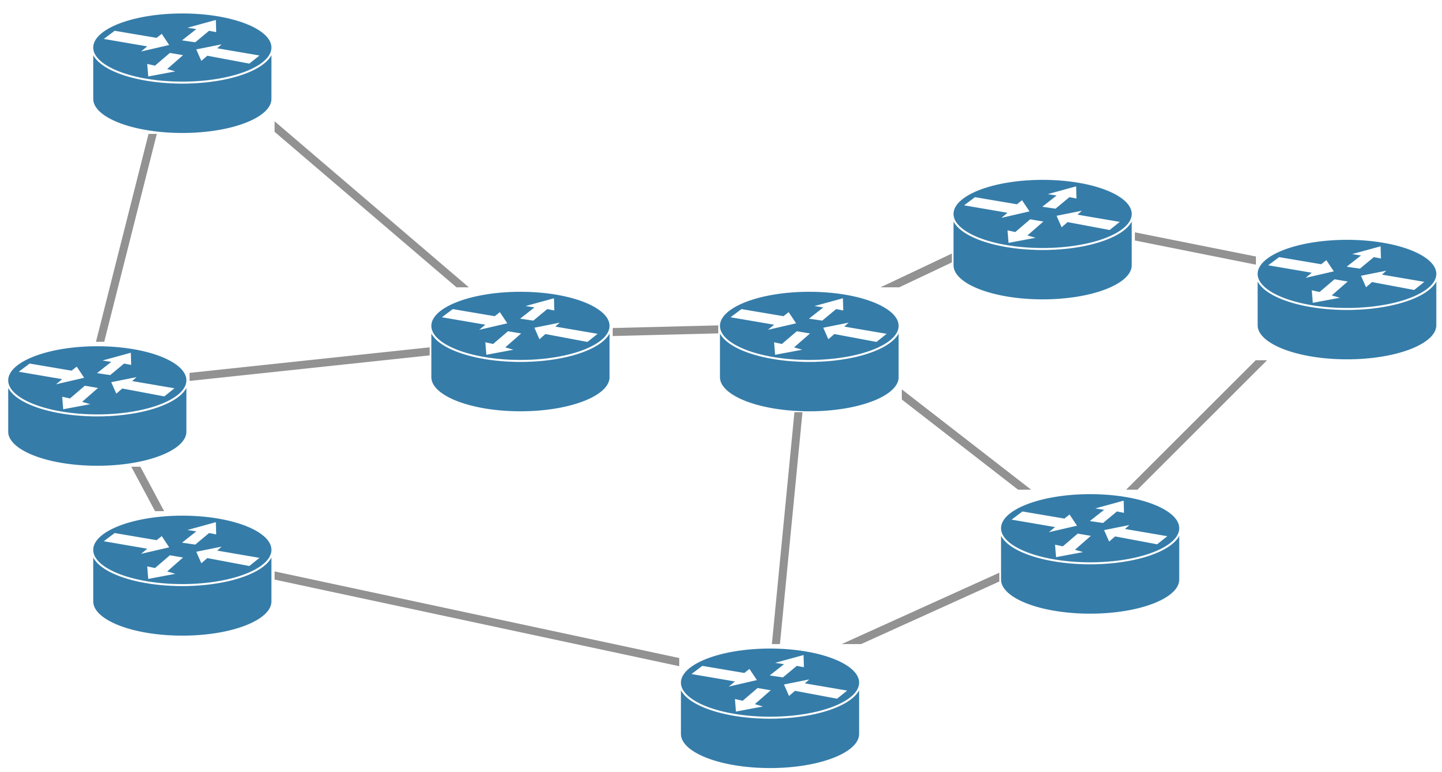
Configuration validation

A *rule-based* approach

Configuration generation

The use of *templates*

Towards *validated* network configurations



High-level representation

Hide useless details

Configuration validation

A *rule-based* approach

Configuration generation

The use of *templates*

High-level representation is a *concise*, and *practical* view of a network

High-level means no more redundancy

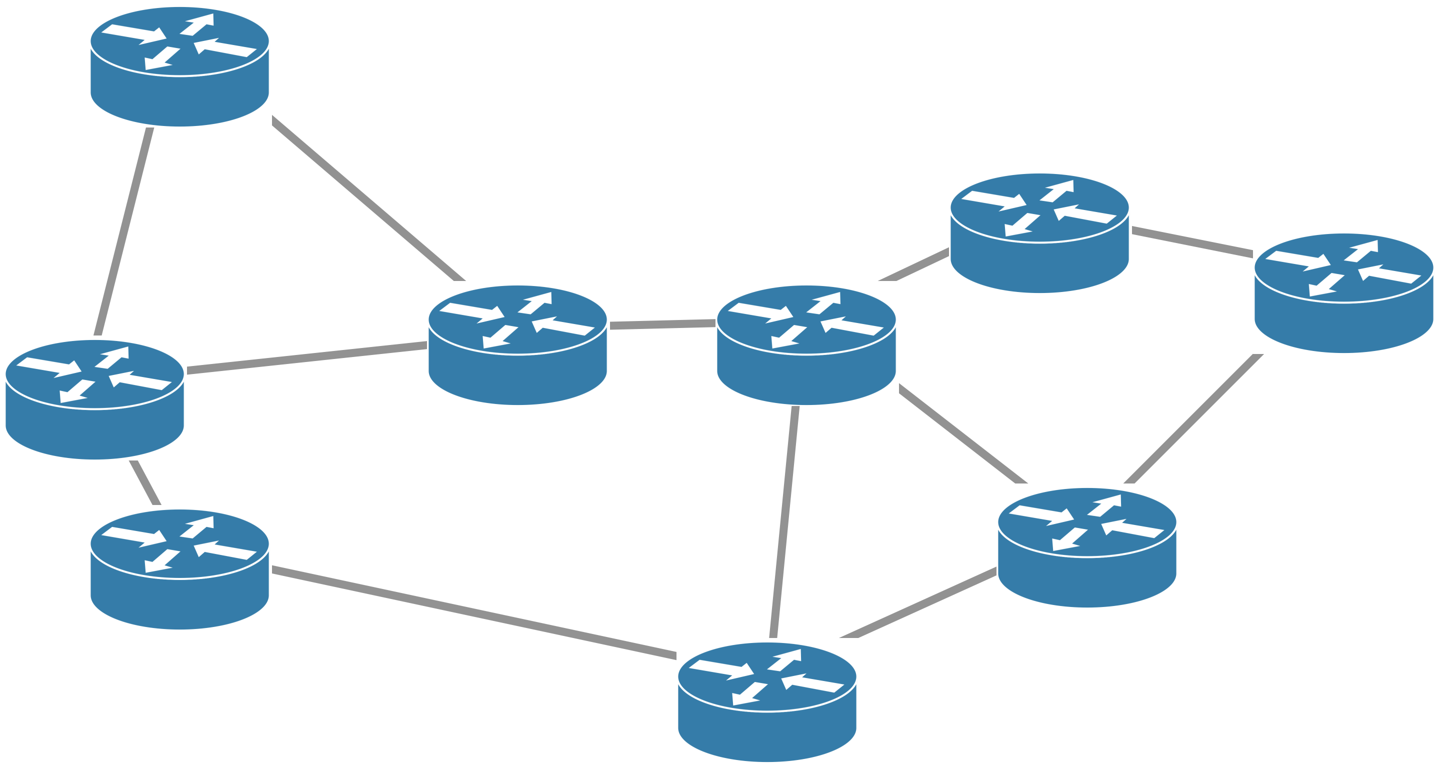
now, you can configure an iBGP full-mesh in a single line

High-level means vendor-independent

no need to bother yourself with language details

```
<node id="NY">
    <characteristics>
        <reference>
            <constructor>juniper</constructor>
   ...
    <rid>64.57.28.242</rid>
    <interfaces>
       <interface id="so-0/0/0">
              <unit number="0">
                  <ip4 mask="31">64.57.28.10</ip4>
```

Towards *validated* network configurations



High-level representation

Hide useless details

Configuration validation

A *rule-based* approach

Configuration generation

The use of *templates*

Validation is performed by using *rules*

A *rule* is a condition that must be met by the *high-level* representation

Many rules follow well-known patterns

- Presence or non-presence
  Each router must have a loopback interface
- Uniqueness
  IP address must be unique
- Symmetry
  MTU must be equal on both sides of a link
- Custom
  Each OSPF area must be connected to the backbone area

Rules are implemented by using *three* techniques

Rules are implemented by using *three* techniques

Structural constraints (XML Schema): **Structural rules**

Rules are implemented by using *three* techniques

Structural constraints (XML Schema): **Structural rules**

Queries on the representation (XQuery): **Query rules**

Rules are implemented by using *three* techniques

Structural constraints (XML Schema): **Structural rules**

Queries on the representation (XQuery): **Query rules**

Programming language (Java): **Language rules**

Rules are implemented by using *three* techniques

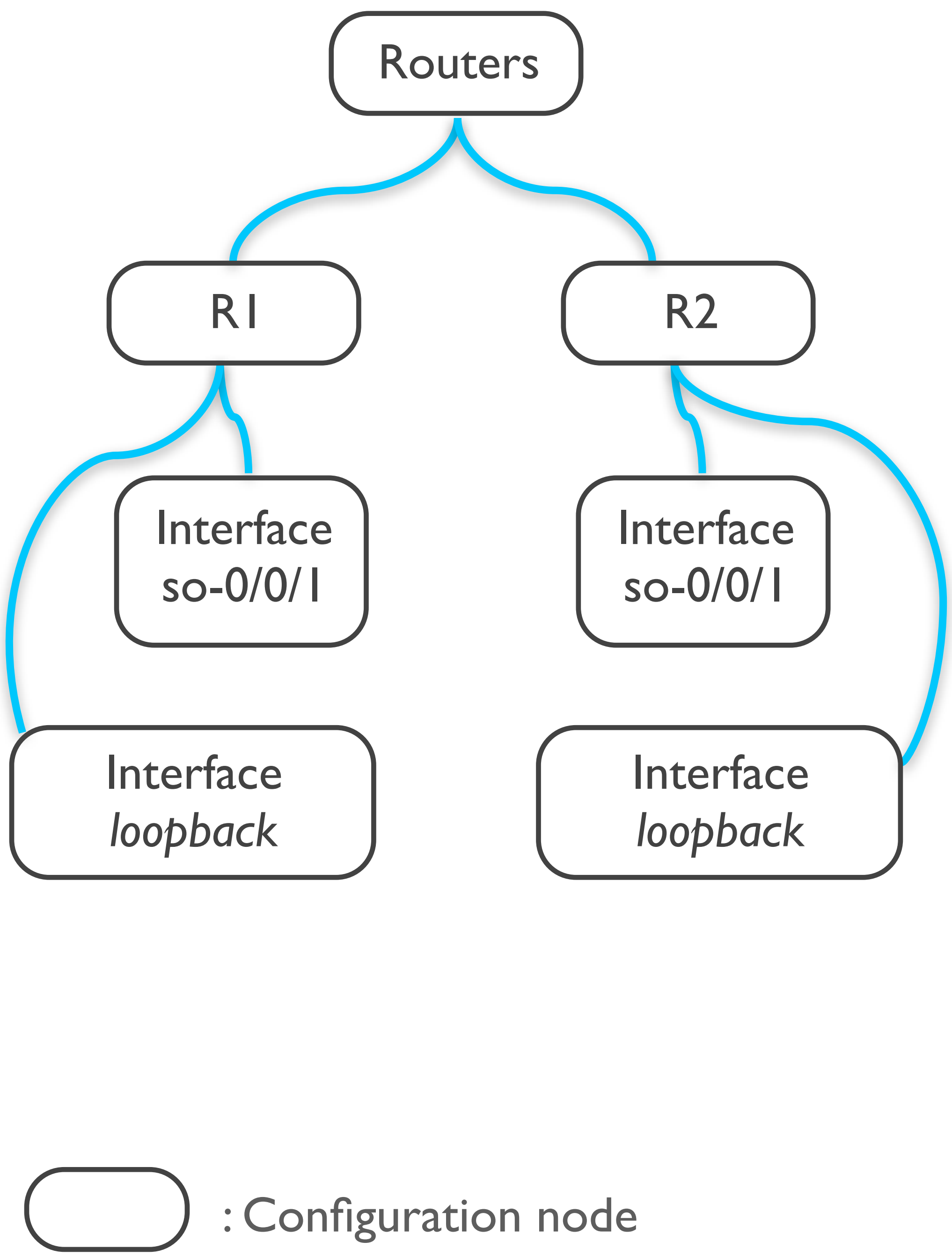
Structural constraints (XML Schema): **Structural rules**

Queries on the representation (XQuery): **Query rules**

Programming language (Java): **Language rules**

	PRESENCE NON-PRESENCE	*UNIQUENESS*	*SYMMETRY*	*CUSTOM*
STRUCTURAL RULES	✓	✓	✓	
QUERY RULES	✓	✓	✓	✓
LANGUAGE RULES				✓

Rules are defined by using a *scope* and a set of *descendants*



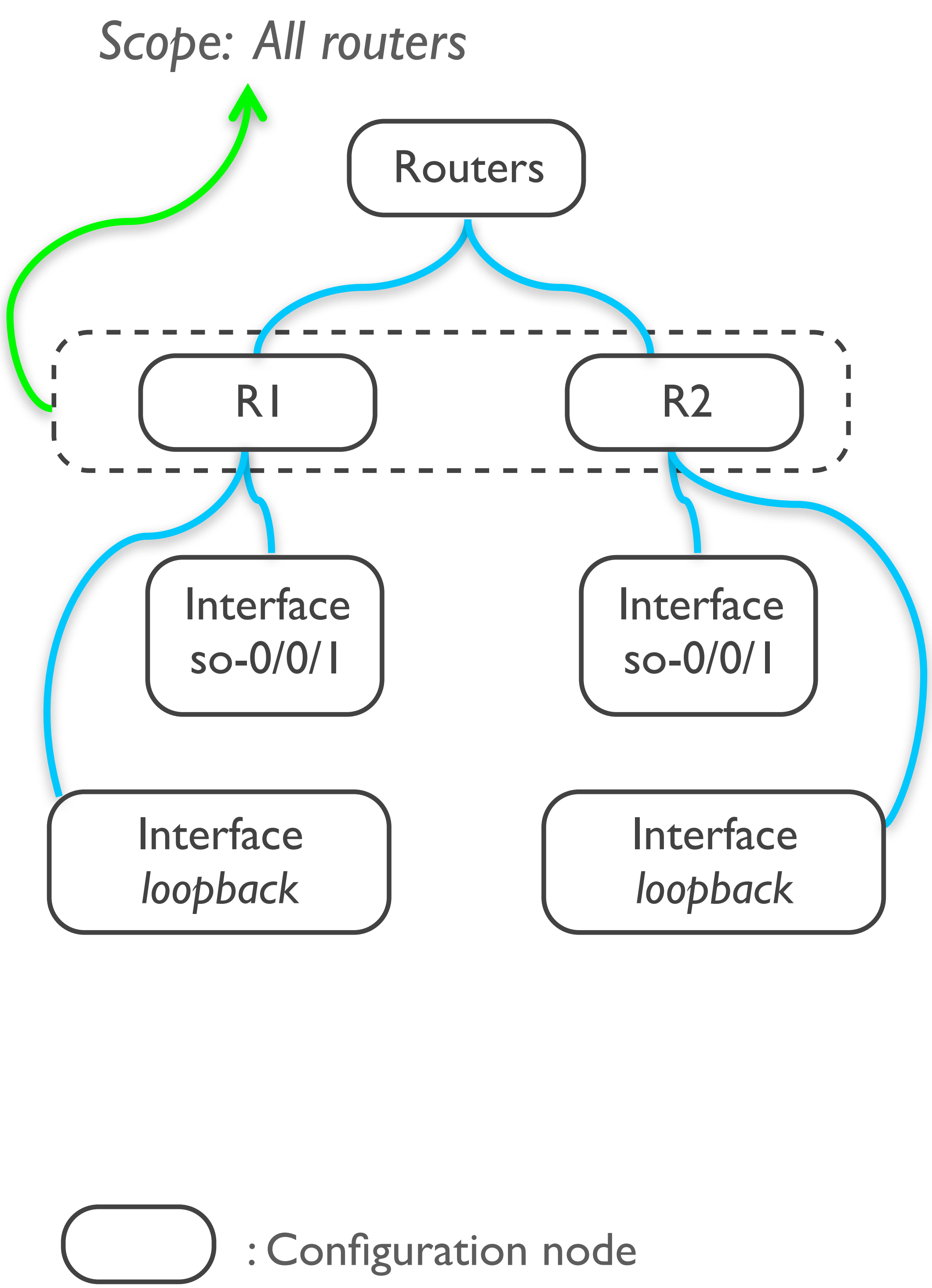
A configuration node is an element of the high-level representation

- A node is composed of attributes

A *scope* is a set of configuration nodes

descendants(x) is a subset of the scope's element *x*

Rules are defined by using a *scope* and a set of *descendants*



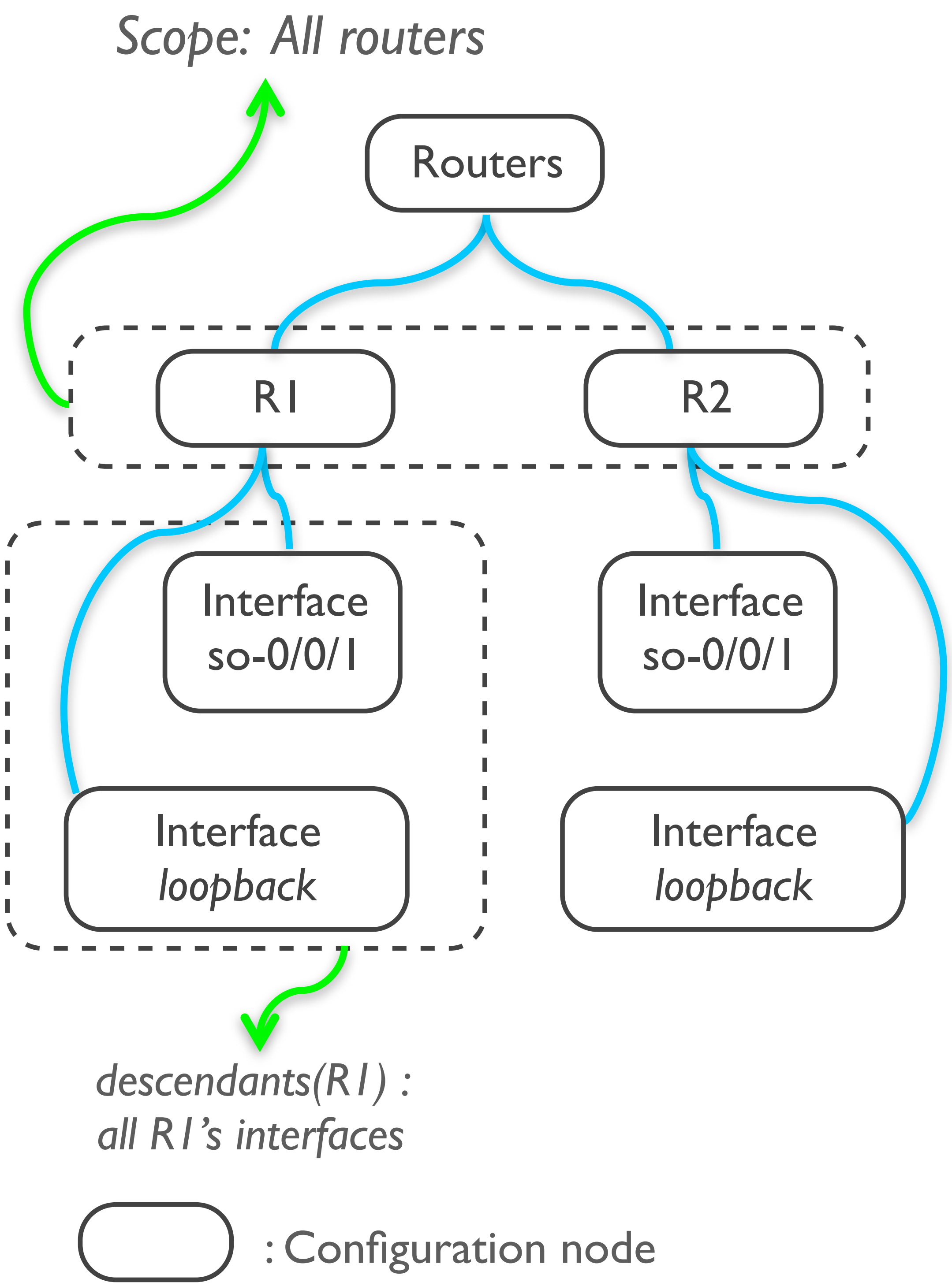
A configuration node is an element of the high-level representation

- A node is composed of attributes

A *scope* is a set of configuration nodes

descendants(x) is a subset of the scope's element *x*

Rules are defined by using a *scope* and a set of *descendants*



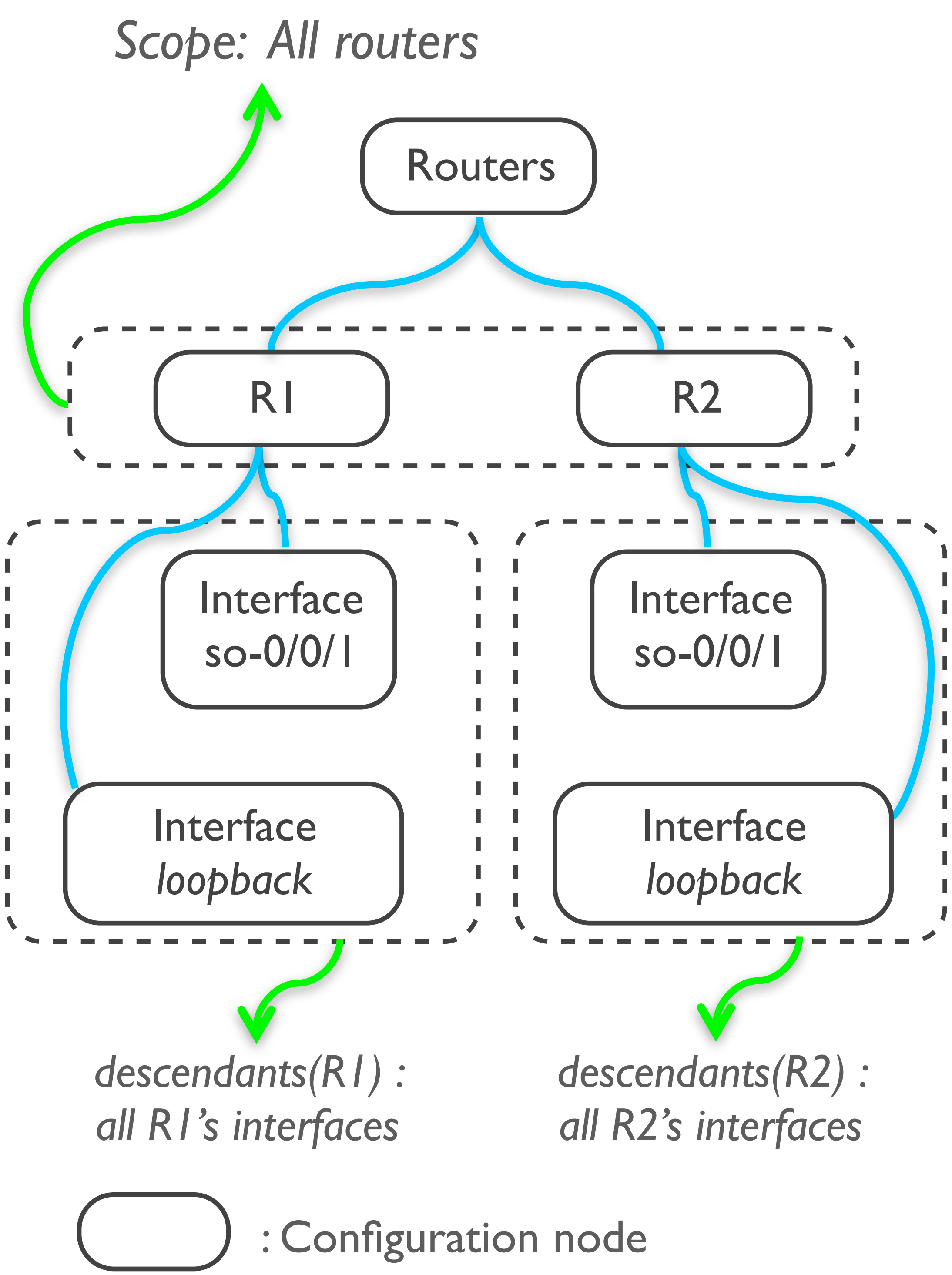
A configuration node is an element of the high-level representation

- A node is composed of attributes

A *scope* is a set of configuration nodes

descendants(x) is a subset of the scope's element *x*

Rules are defined by using a *scope* and a set of *descendants*



A configuration node is an element of the high-level representation

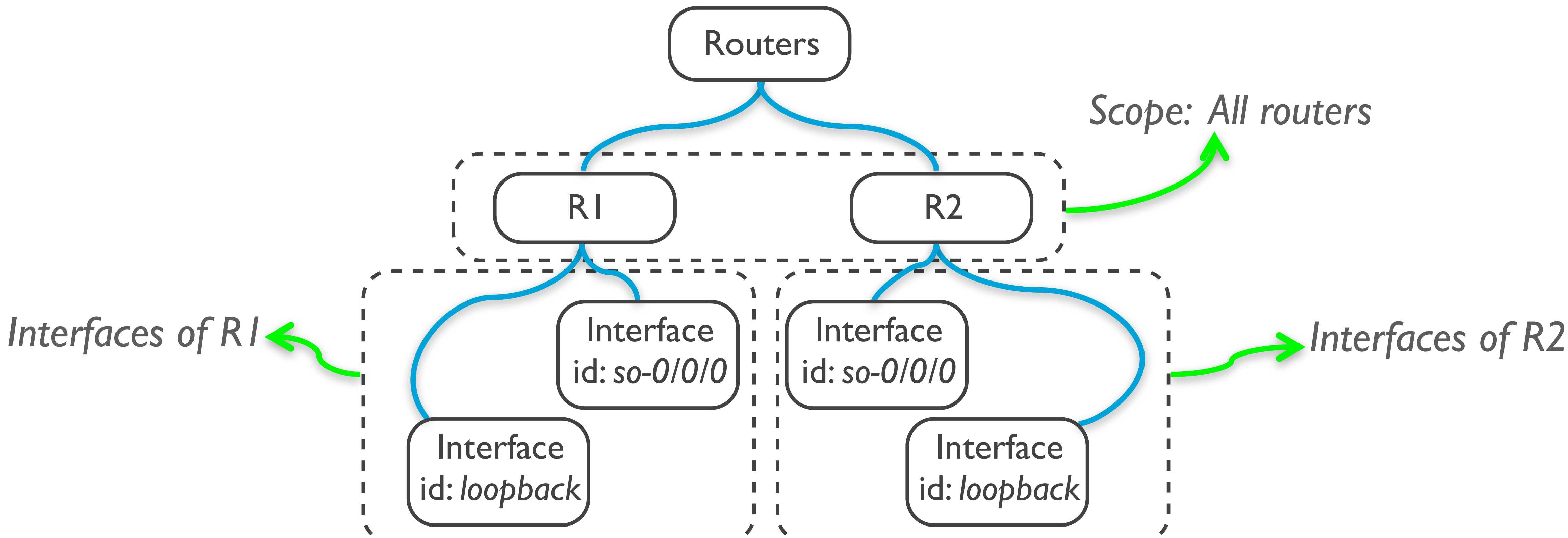
- A node is composed of attributes

A *scope* is a set of configuration nodes

descendants(x) is a subset of the scope's element *x*

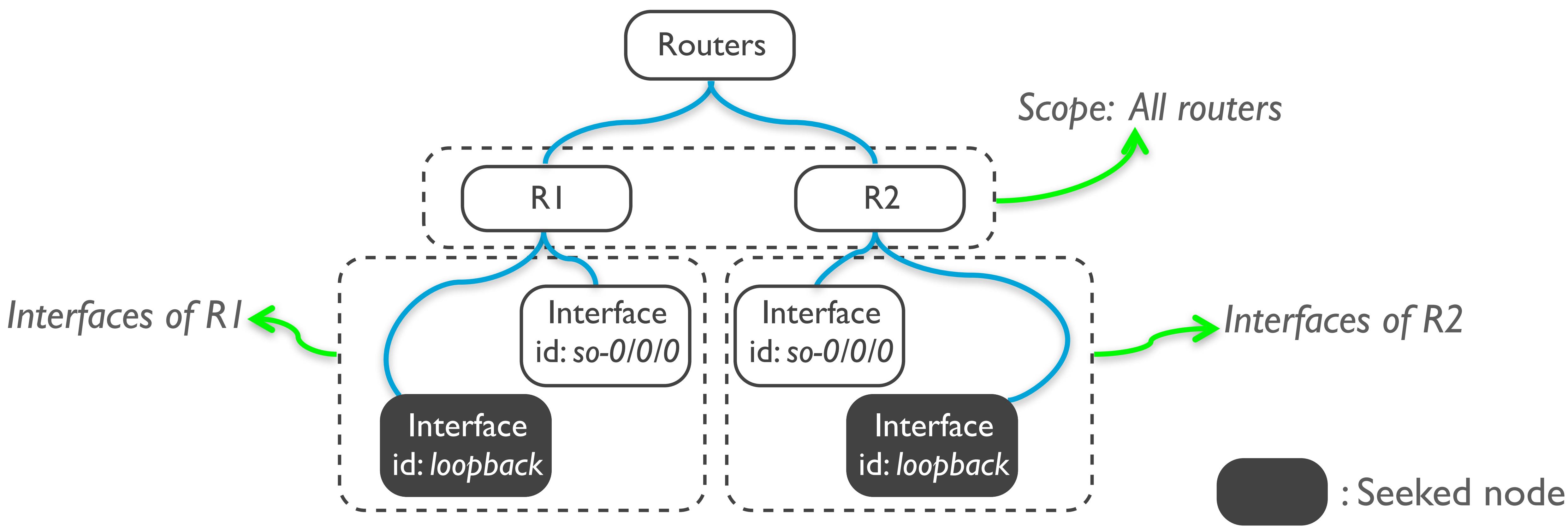
Presence rules check whether nodes are in the representation

Each router ***must*** have a loopback interface



Presence rules check whether nodes are in the representation

Each router ***must*** have a loopback interface



Presence rules check whether nodes are in the representation

There is at least one configuration node respecting a given condition in each *descendants* set.

$$\forall x \in \textsc{scope}\ \exists y \in \texttt{descendants}(x)\ :\ C_{\mathrm{presence}}(T, y)$$

Each router **must** have a loopback interface

$$\forall x \in \textsc{routers}\ \exists y \in \texttt{interfaces}(x)\ : y.id = loopback$$

```
<rule id="LOOPBACK_INTERFACE_ON_EACH_NODE" type="presence">
<presence>
        <scope>ALL_NODES</scope>
        <descendants>interfaces/interface</descendants>
        <condition>@id='loopback'</condition>
</presence>
</rule>
```

Presence rules check whether nodes are in the representation

There is at least one configuration node respecting a given condition in each *descendants* set.

$$\forall x \in \textsc{scope}\ \exists y \in \mathtt{descendants}(x)\ :\ C_{\text{presence}}(T, y)$$

Each router **must** have a loopback interface

$$\forall x \in \textsc{routers}\ \exists y \in \mathtt{interfaces}(x)\ : y.id = loopback$$

```
<rule id="LOOPBACK_INTERFACE_ON_EACH_NODE" type="presence">
<presence>
        <scope>ALL_NODES</scope>
        <descendants>interfaces/interface</descendants>
        <condition>@id='loopback'</condition>
</presence>
</rule>
```

Presence rules check whether nodes are in the representation

There is at least one configuration node respecting a given condition in each *descendants* set.

$$\forall x \in \textsc{scope}\ \exists y \in \texttt{descendants}(x)\ :\ C_{\text{presence}}(T, y)$$

Each router **must** have a loopback interface

$$\boxed{\forall x \in \textsc{routers}}\ \exists y \in \texttt{interfaces}(x)\ : y.id = loopback$$

```
<rule id="LOOPBACK_INTERFACE_ON_EACH_NODE" type="presence">
<presence>
       <scope>ALL_NODES</scope>
       <descendants>interfaces/interface</descendants>
       <condition>@id='loopback'</condition>
</presence>
</rule>
```

Presence rules check whether nodes are in the representation

There is at least one configuration node respecting a given condition in each *descendants* set.

$$\forall x \in \textsc{scope}\ \exists y \in \texttt{descendants}(x)\ :\ C_{\text{presence}}(T, y)$$

Each router **must** have a loopback interface

$$\forall x \in \textsc{routers}\ \boxed{\exists y \in \texttt{interfaces}(x)} : y.id = loopback$$

```
<rule id="LOOPBACK_INTERFACE_ON_EACH_NODE" type="presence">
<presence>
        <scope>ALL_NODES</scope>
        <descendants>interfaces/interface</descendants>
        <condition>@id='loopback'</condition>
</presence>
</rule>
```

Presence rules check whether nodes are in the representation

There is at least one configuration node respecting a given condition in each *descendants* set.

$$\forall x \in \textsc{scope}\ \exists y \in \texttt{descendants}(x)\ :\ C_{\text{presence}}(T, y)$$

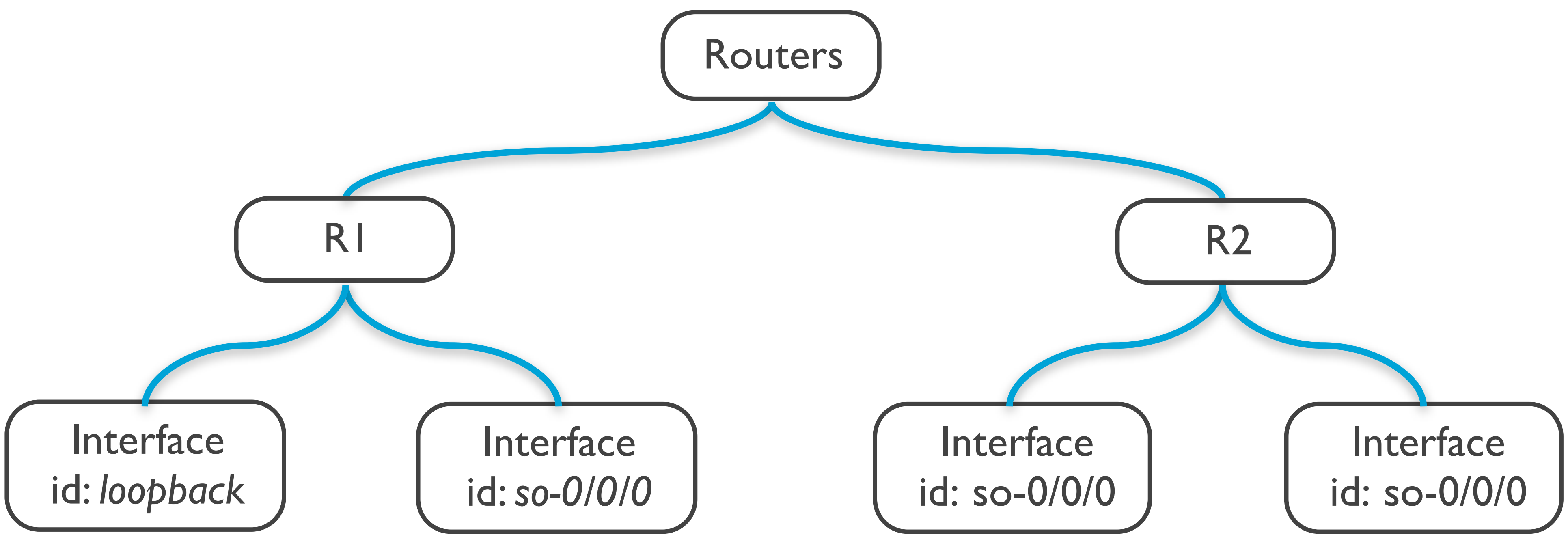
Each router **must** have a loopback interface

$$\forall x \in \textsc{routers}\ \exists y \in \texttt{interfaces}(x)\ :\boxed{y.id = loopback}$$

```
<rule id="LOOPBACK_INTERFACE_ON_EACH_NODE" type="presence">
<presence>
        <scope>ALL_NODES</scope>
        <descendants>interfaces/interface</descendants>
        <condition>@id='loopback'</condition>
</presence>
</rule>
```

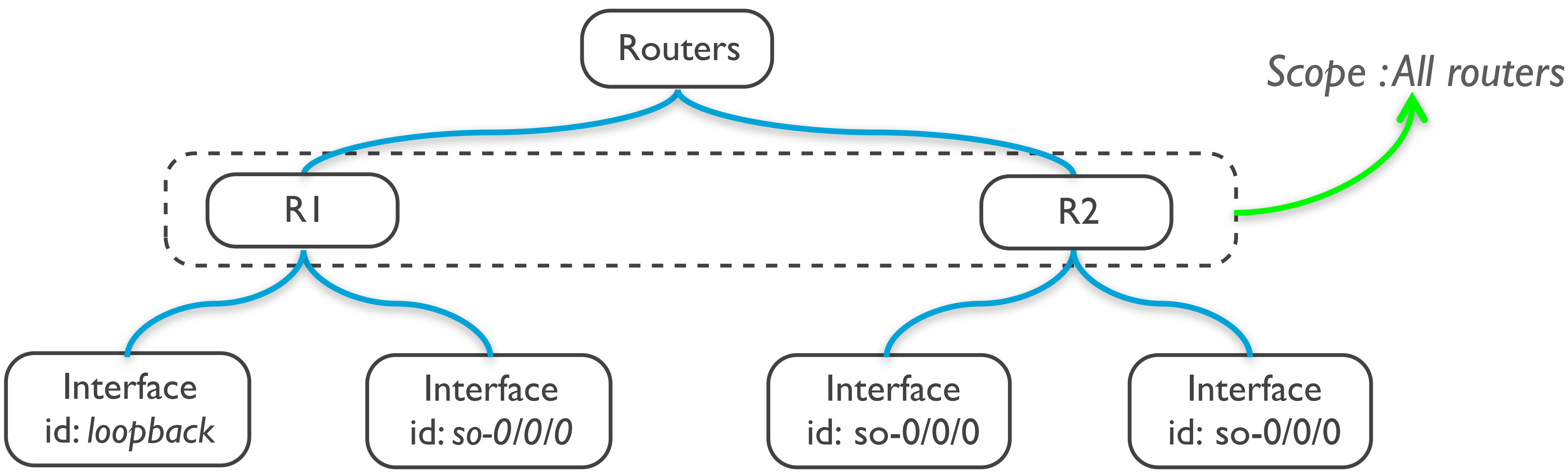
Uniqueness rules verify the cardinality of a field among a set of nodes

Routers interfaces identifiers ***must*** be unique



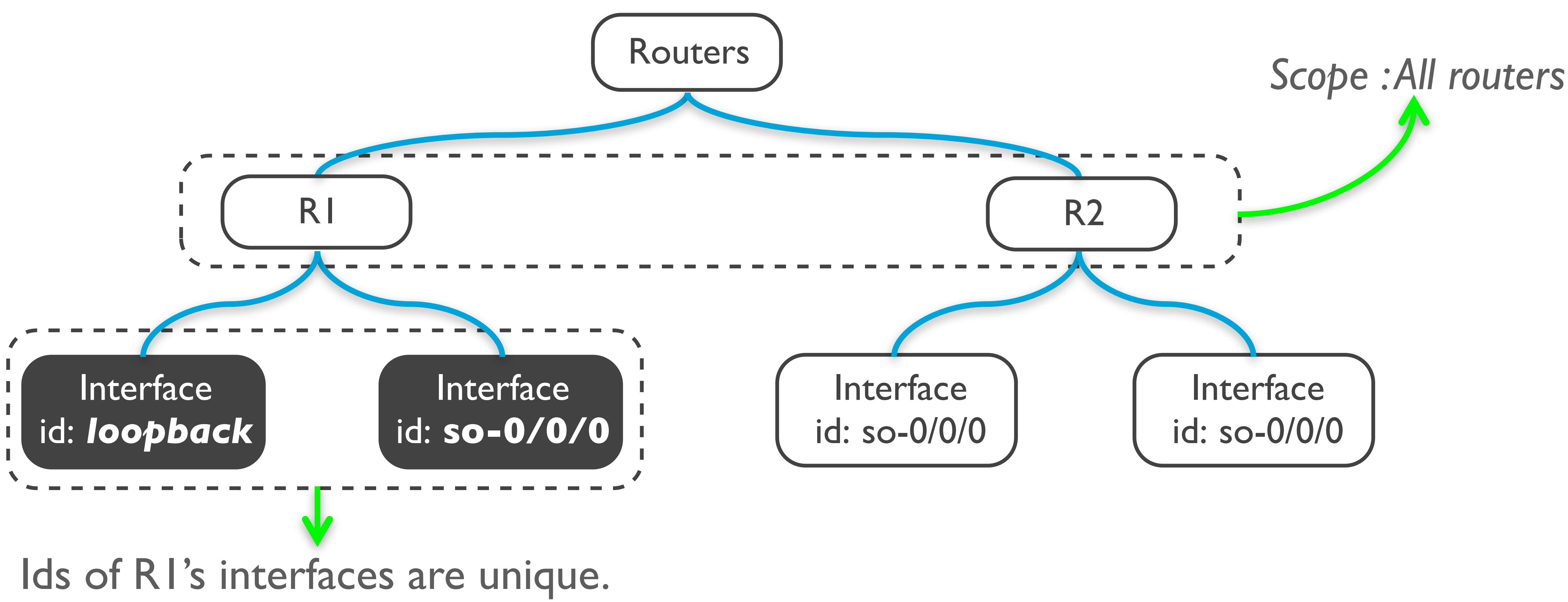
Uniqueness rules verify the cardinality of a field among a set of nodes

Routers interfaces identifiers ***must*** be unique



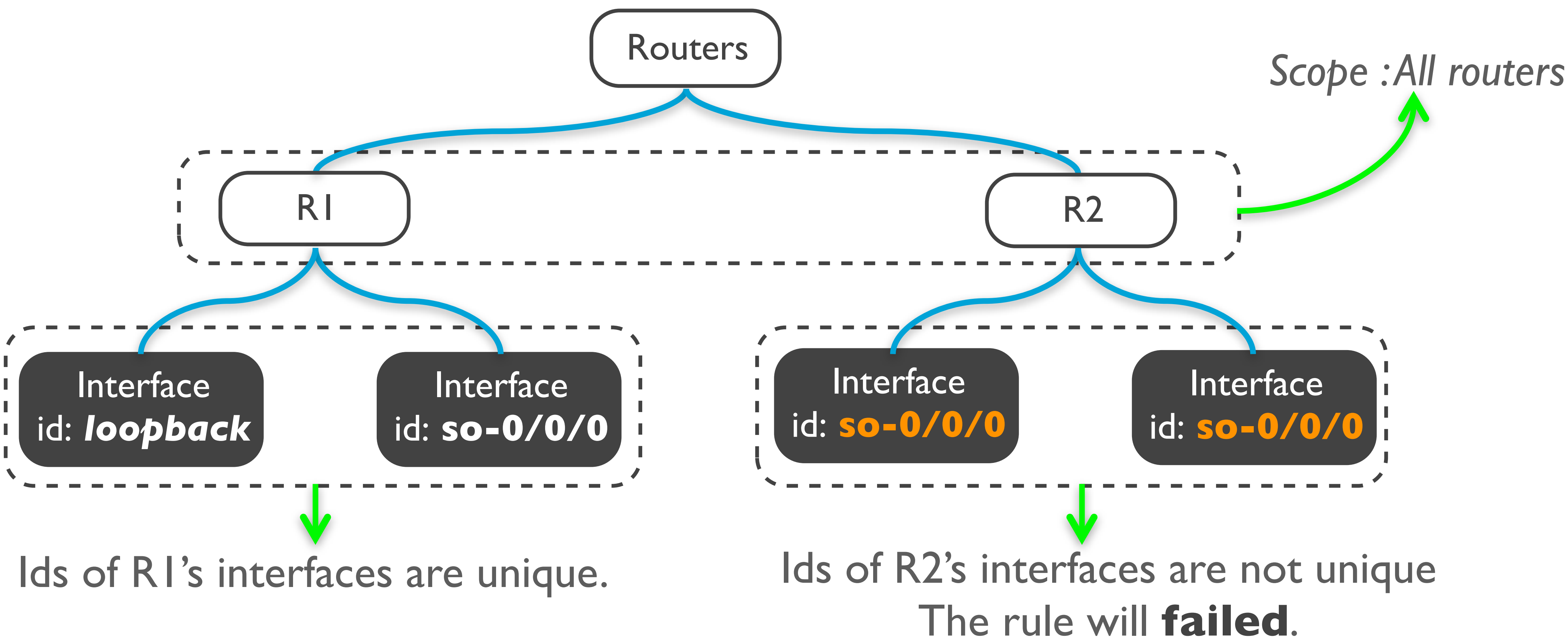
Uniqueness rules verify the cardinality of a field among a set of nodes

Routers interfaces identifiers ***must*** be unique



Uniqueness rules verify the cardinality of a field among a set of nodes

Routers interfaces identifiers ***must*** be unique



Uniqueness rules verify the cardinality of a field among a set of nodes

Check if there is no two configuration nodes with identical value of *field*

$$\forall x \in \textsc{scope}\ \forall y \in \mathtt{d}(x)\ : \neg(\exists z_{\neq y} \in \mathtt{d}(x) : y.field = z.field)$$

Uniqueness of routers interfaces identifiers

$$\forall x \in \textsc{routers}\ \forall y \in \mathtt{interfaces}(x)\ : \neg(\exists z_{\neq y} \in \mathtt{interfaces}(x) : y.id = z.id)$$

```
<rule id="UNIQUENESS_INTERFACE_ID" type="uniqueness">
<uniqueness>
        <scope>ALL_NODES</scope>
        <descendants>interfaces/interface</descendants>
        <field>@id</field>
</uniqueness>
</rule>
```

Uniqueness rules verify the cardinality of a field among a set of nodes

Check if there is no two configuration nodes with identical value of *field*

$$\forall x \in \textsc{scope}\, \forall y \in \mathtt{d}(x)\ :\neg(\exists z_{\neq y} \in \mathtt{d}(x) : y.field = z.field)$$

Uniqueness of routers interfaces identifiers

$$\forall x \in \textsc{routers}\, \forall y \in \mathtt{interfaces}(x)\ :\neg(\exists z_{\neq y} \in \mathtt{interfaces}(x) : y.id = z.id)$$

```
<rule id="UNIQUENESS_INTERFACE_ID" type="uniqueness">
<uniqueness>
        <scope>ALL_NODES</scope>
        <descendants>interfaces/interface</descendants>
        <field>@id</field>
</uniqueness>
</rule>
```

Uniqueness rules verify the cardinality of a field among a set of nodes

Check if there is no two configuration nodes with identical value of *field*

$$\forall x \in \textsc{scope}\, \forall y \in \mathtt{d}(x)\ : \neg(\exists z_{\neq y} \in \mathtt{d}(x) : y.field = z.field)$$

Uniqueness of routers interfaces identifiers

$$\forall x \in \textsc{routers}\, \forall y \in \mathtt{interfaces}(x)\ : \neg(\exists z_{\neq y} \in \mathtt{interfaces}(x) : y.id = z.id)$$

```
<rule id="UNIQUENESS_INTERFACE_ID" type="uniqueness">
<uniqueness>
        <scope>ALL_NODES</scope>
        <descendants>interfaces/interface</descendants>
        <field>@id</field>
</uniqueness>
</rule>
```

Uniqueness rules verify the cardinality of a field among a set of nodes

Check if there is no two configuration nodes with identical value of *field*

$$\forall x \in \textsc{scope}\, \forall y \in \mathtt{d}(x) \; : \neg(\exists z_{\neq y} \in \mathtt{d}(x) : y.field = z.field)$$

Uniqueness of routers interfaces identifiers

$$\forall x \in \textsc{routers}\, \boxed{\forall y \in \mathtt{interfaces}(x)} : \neg(\exists z_{\neq y} \in \mathtt{interfaces}(x) : y.id = z.id)$$

```
<rule id="UNIQUENESS_INTERFACE_ID" type="uniqueness">
<uniqueness>
        <scope>ALL_NODES</scope>
        <descendants>interfaces/interface</descendants>
        <field>@id</field>
</uniqueness>
</rule>
```

Uniqueness rules verify the cardinality of a field among a set of nodes

Check if there is no two configuration nodes with identical value of *field*

$$\forall x \in \textsc{scope}\, \forall y \in \mathtt{d}(x)\ : \neg(\exists z_{\neq y} \in \mathtt{d}(x) : y.field = z.field)$$

Uniqueness of routers interfaces identifiers

$$\forall x \in \textsc{routers}\, \forall y \in \mathtt{interfaces}(x)\ : \boxed{\neg(\exists z_{\neq y} \in \mathtt{interfaces}(x) : y.id = z.id)}$$

```
<rule id="UNIQUENESS_INTERFACE_ID" type="uniqueness">
<uniqueness>
        <scope>ALL_NODES</scope>
        <descendants>interfaces/interface</descendants>
        <field>@id</field>
</uniqueness>
</rule>
```

Symmetry rules verify the equality of a field among a set of nodes

Such rules can be checked **implicitly** by the high-level representation

Symmetry rules verify the equality of a field among a set of nodes

Such rules can be checked **implicitly** by the high-level representation

MTU ***must*** be equal on both ends of a link

Symmetry rules verify the equality of a field among a set of nodes

Such rules can be checked **implicitly** by the high-level representation

MTU ***must*** be equal on both ends of a link

Automatically checked by modeling it once at the link level, instead of twice at the interfaces level

Hypothesis: duplication phase is correct

Custom rules check advanced conditions

They are expressed in a query or programming language

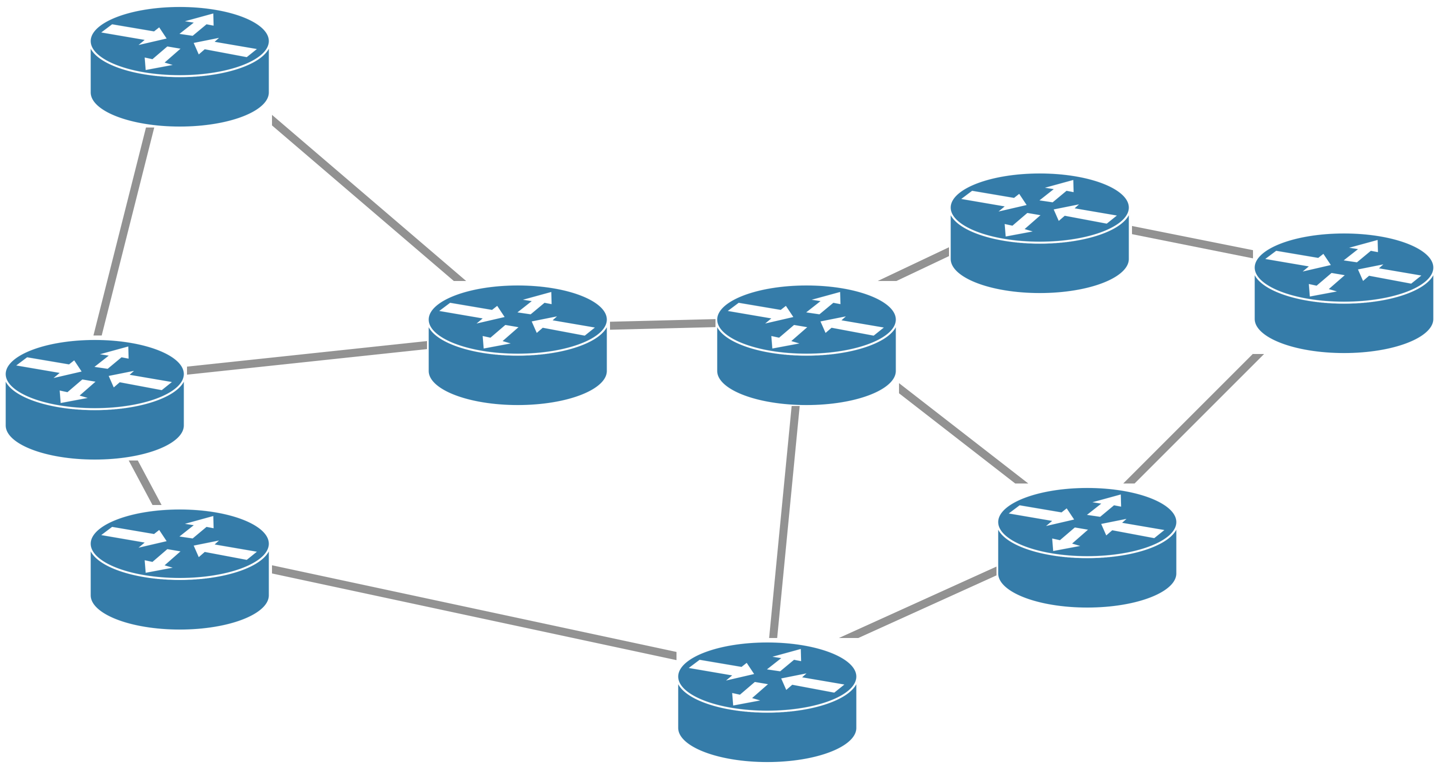
Example: All OSPFs areas must be connected to the backbone

```
<rule id="ALL_AREAS_CONNECTED_TO_BACKBONE_AREA" type="custom">
    <custom>
        <xquery>
            for $area in /domain/ospf/areas/area[@id!="0.0.0.0"]
            let $nodes := $area/nodes/node
            where count(/domain/ospf/areas/area) > 1
            and not(some $y in $nodes satisfies /domain/ospf/areas/
                    area[@id="0.0.0.0"]/nodes/node[@id=$y/@id])
           return
               <result><area id="{$area/@id}"/></result>
        </xquery>
    </custom>
</rule>
```

Over 100 rules were written for a network composed of 9 routers

Type	Total
Presence, non-presence	97
Uniqueness	20
Symmetry	10
Custom	9
Total	136

Towards *validated* network configurations



High-level representation

Hide useless details

Configuration validation

A *rule-based* approach

Configuration generation

The use of *templates*

Configurations are automatically produced based on the high-level representation

We use intermediate representations

it represents the high-level configuration of one device

Templates translate them into configuration files

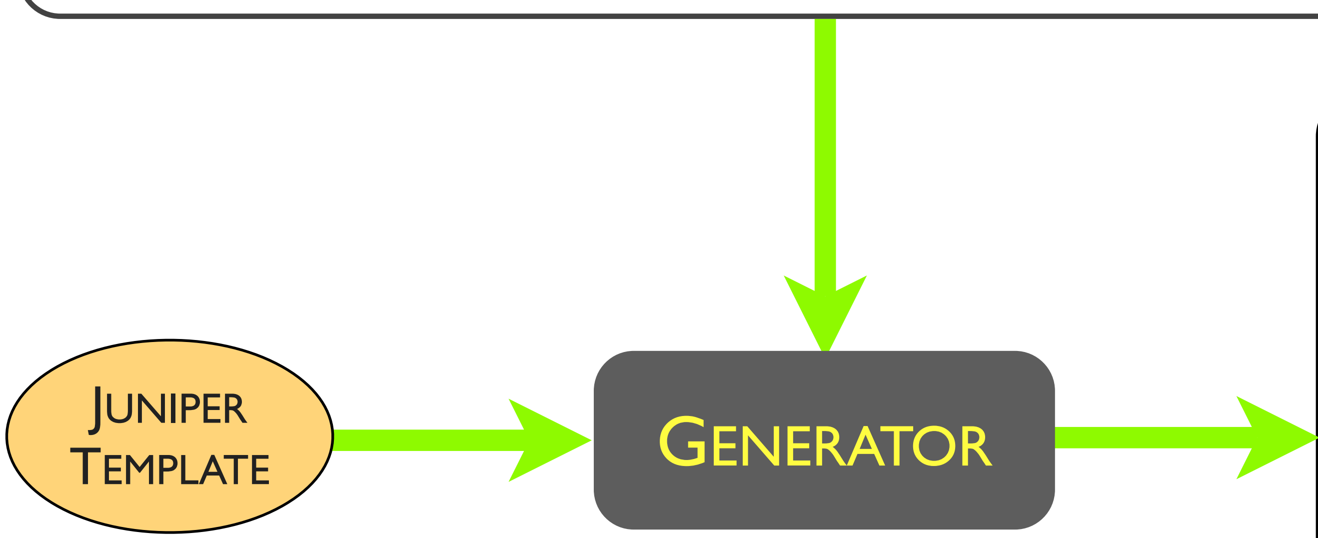
templates are vendor-specific

To support a new vendor, add a new template

we have a template for Juniper and Cisco configurations

Low-level configurations are automatically generated

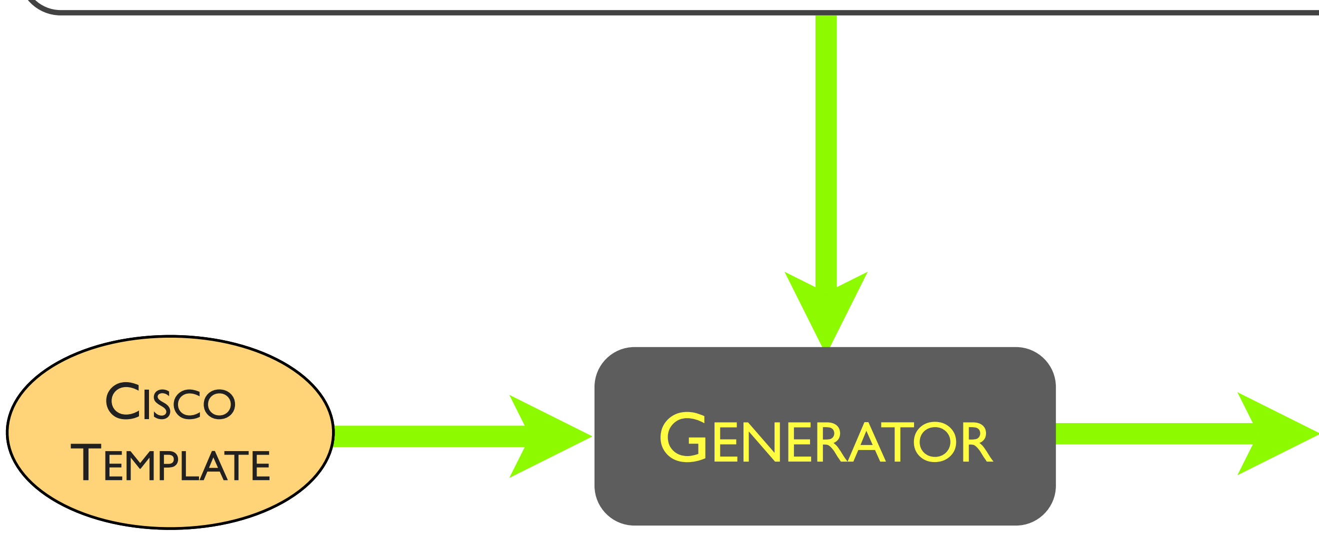
```
<node id="SALT">
   <interfaces>
      <interface id="lo0">
         <unit number="0">
            <ip4>198.32.8.200</ip4>
            <ip6>2001:468:16::1</ip6>
         </unit>
      </interface>
   </interfaces>
</node>
```




```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 198.32.8.200/32;
            }
            family inet6 {
                address 2001:468:16::1/128;
            }
        }
    }
```

Low-level configurations are automatically generated

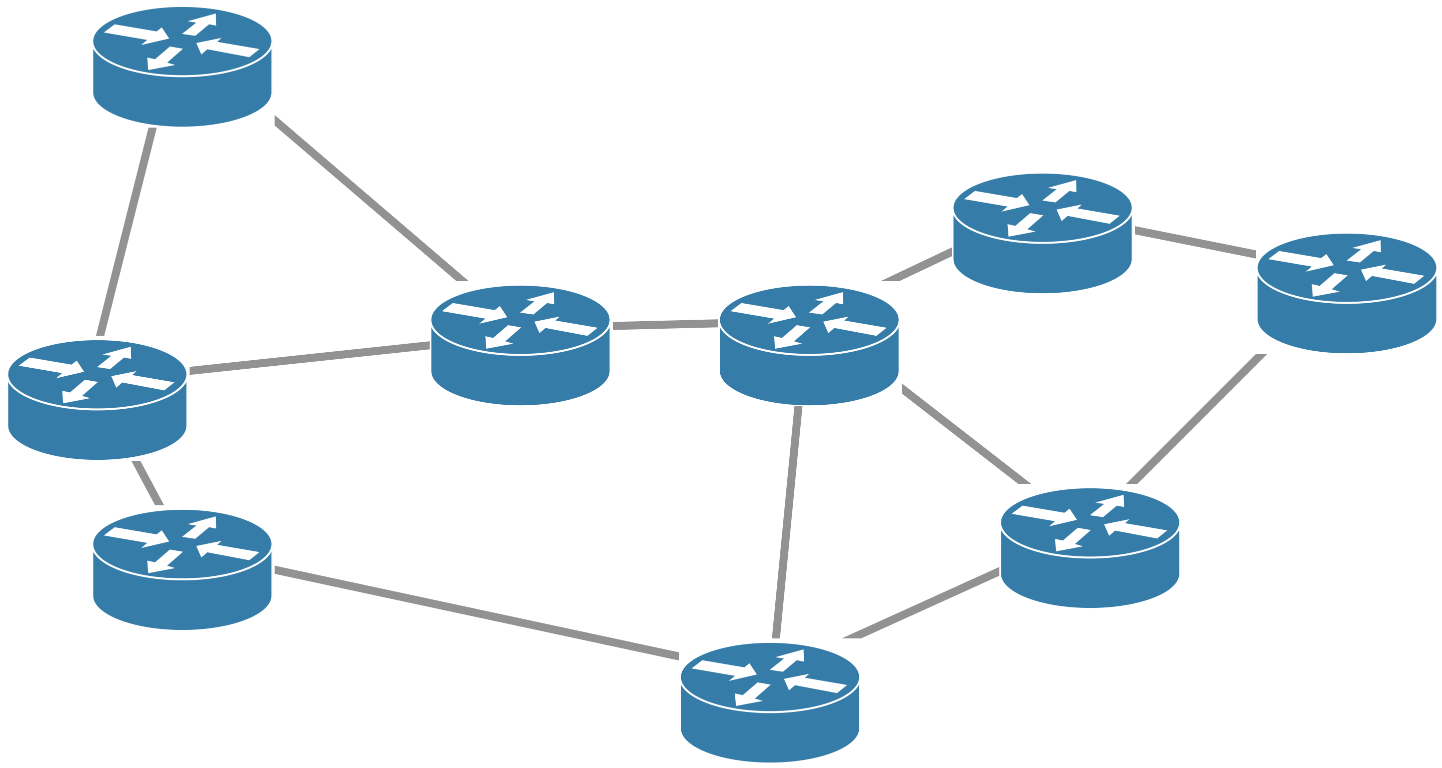
```
<node id="SALT">
   <interfaces>
      <interface id="lo0">
         <unit number="0">
            <ip4>198.32.8.200</ip4>
            <ip6>2001:468:16::1</ip6>
         </unit>
      </interface>
   </interfaces>
</node>
```




```
interface Loopback0
   ip address 198.32.8.200/32;
   ipv6 address 2001:468:16::1/128;
!
```

Demonstration

Towards *validated* network configurations



High-level representation

Hide useless details

Configuration validation

A *rule-based* approach

Configuration generation

The use of *templates*

Producing validated network configurations is *possible*

Use high-level representations

suppress redundancy, hide useless details

Validate the representation

really easy to add rules (most are a few lines length)

Generate low-level configurations automatically

flexibility is kept by letting you modify the templates

What is next ?

Improve the high-level representation ?

XML may not be the most appropriate...

An open-source library of validation rules ?

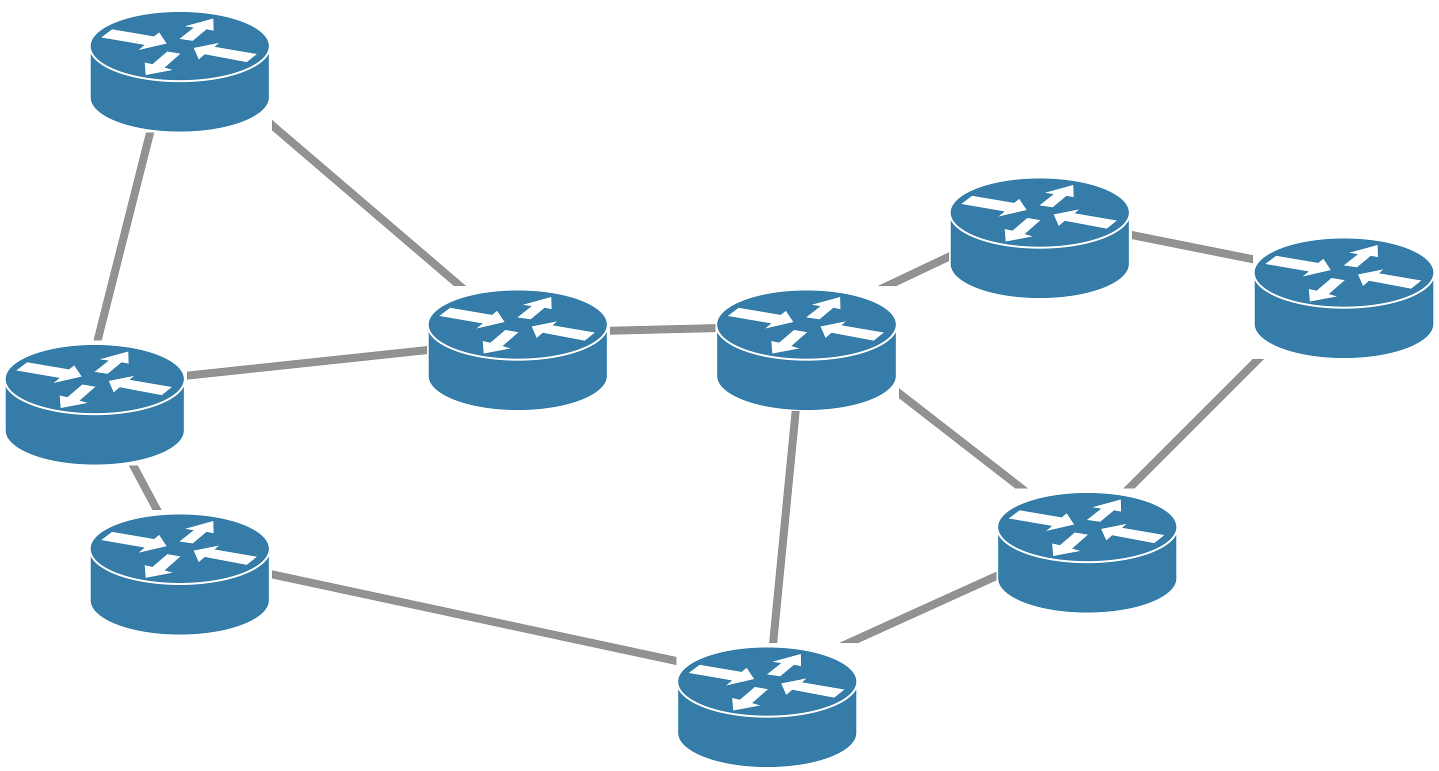
e.g., rules checking the BCP of OSPF, BGP, etc.

How do we validate dynamic properties ?

Can we deploy generated configurations automatically ?

and, if possible, *without* traffic disruption

Towards *validated* network configurations



Laurent Vanbever

http://inl.info.ucl.ac.be/lvanbeve

Thank you for your attention

Any questions ?