Dynamic Configuration

background

SDN and Dynamic Control

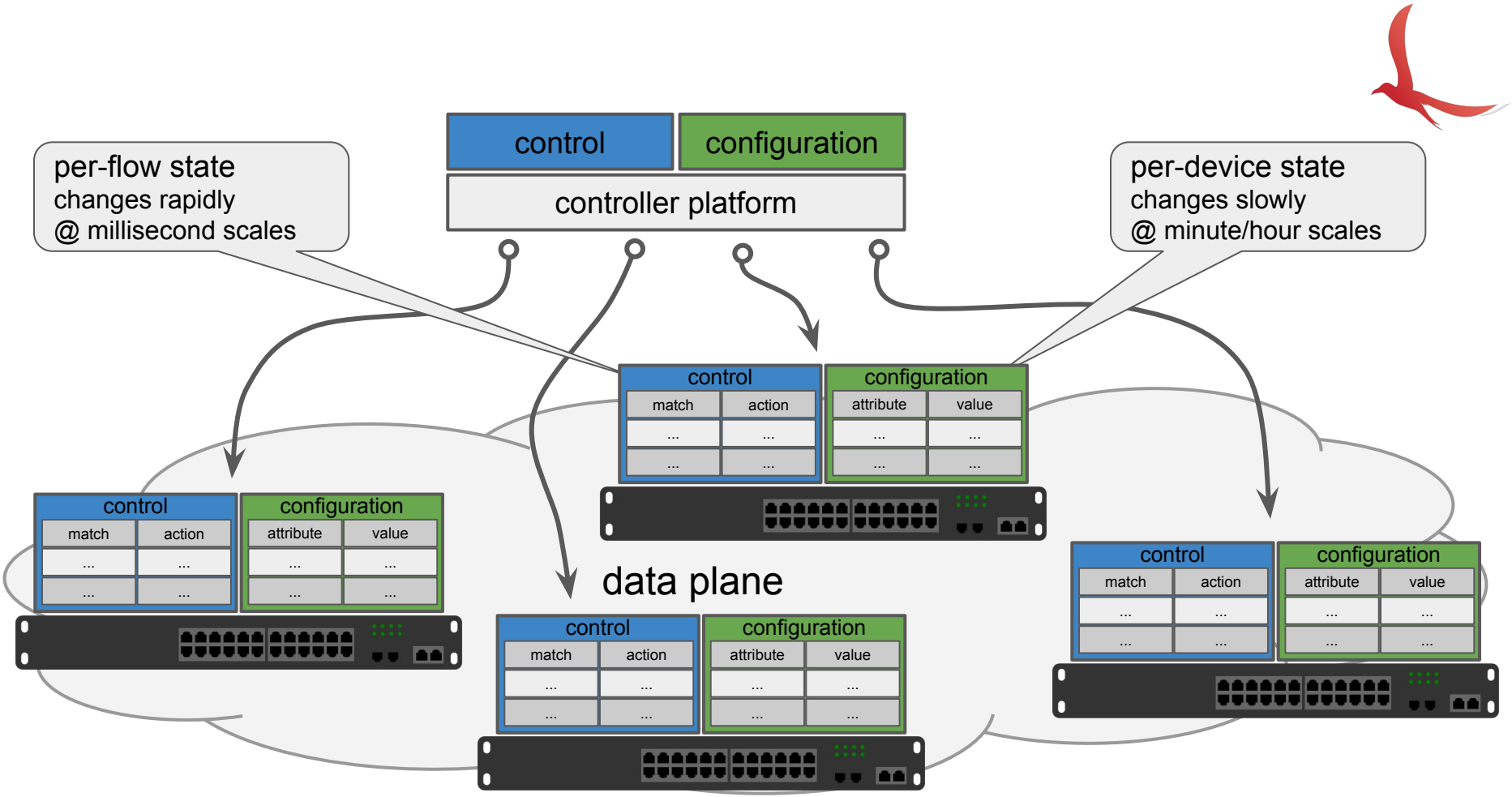
- Dynamic control over forwarding plane behaviour
  - from a logically centralized vantage point

Configuration and Management

- Industry has been doing this for quite a long time
  - ...from centralized vantage points
  - and for very large networks
- So what is different?

Increased demands

- Dynamic control increased demands on performance and reliability
  - from human time-scales to machine time-scales
  - reduced tolerance for control or management plane failures
- Performance and scalability are important for configuration as well, but...
  - configuration & management requires ~1,000s / day
  - control requires ~1,000s, if not ~1,000,000s / second
  - control requires low latencies

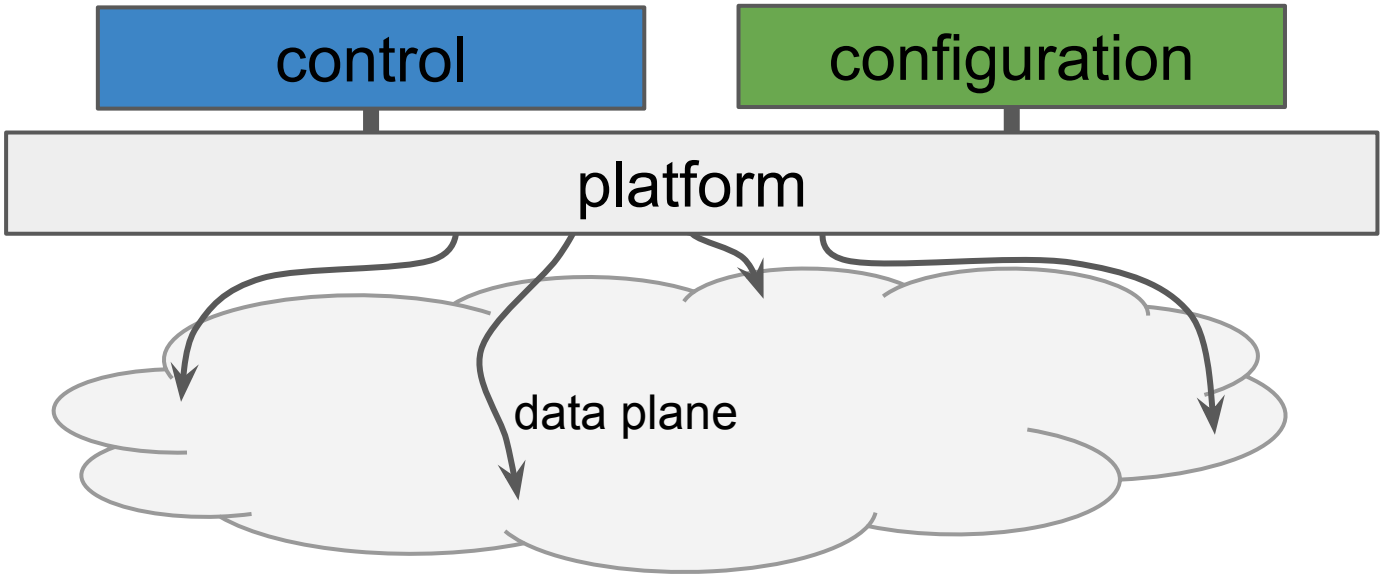

control
configuration
controller platform
per-flow state
changes rapidly
@ millisecond scales
per-device state
changes slowly
@ minute/hour scales
control
configuration
match
action
attribute
value
...
data plane

Configuration still critical

- Dynamic (re)configuration continues to be critical
  - networks still need to be managed and configured
  - if nothing else, configured to be controlled
- Configuration even more important in brown-fields
  - devices may expose only limited control capabilities
- Operators want to create & sell customized services
  - do this with agility and minimal human intervention
  - create automated ways to instantiate such network services
  - services comprise *both* configuration & control

Control and Configuration

- Operators need a resilient and scalable platform capable of *both* control and configuration



approaches

API-driven approach

- Control abstractions and APIs are semi-fixed
  - borne out of following the OpenFlow standard
- Configuration abstractions are harder to fix
  - though standards exist, vendors want to expose unique features

API-driven approach

- Application developers presented with a firm surface
- Applications are not tied to a closed set of protocols
  - adapters can use YANG/NETCONF, SNMP, REST, or any other
- Helps application portability and stable evolution
  - burden of adaptation falls on southbound adapters
- Limits access to new or differentiating device features
  - unless API is sufficiently open-ended

Model-driven approach

- Model defines abstraction
  - YANG source file is the canonical representation
  - APIs and other code are derived from the model
- Model also defines the data exchange format(s)
  - XML or JSON schema are derived from the model

Model-driven approach

- Code-generation avoids manual boilerplate code
  - consideration must be given to versioning
  - impact of a model change on the (re)generated API
- Applications have access to nuanced features
  - not limited by a fixed API
- Applications presented with a fluid surface
  - application must be model-aware, i.e. know model semantics
  - impact on application portability

Mixed approach

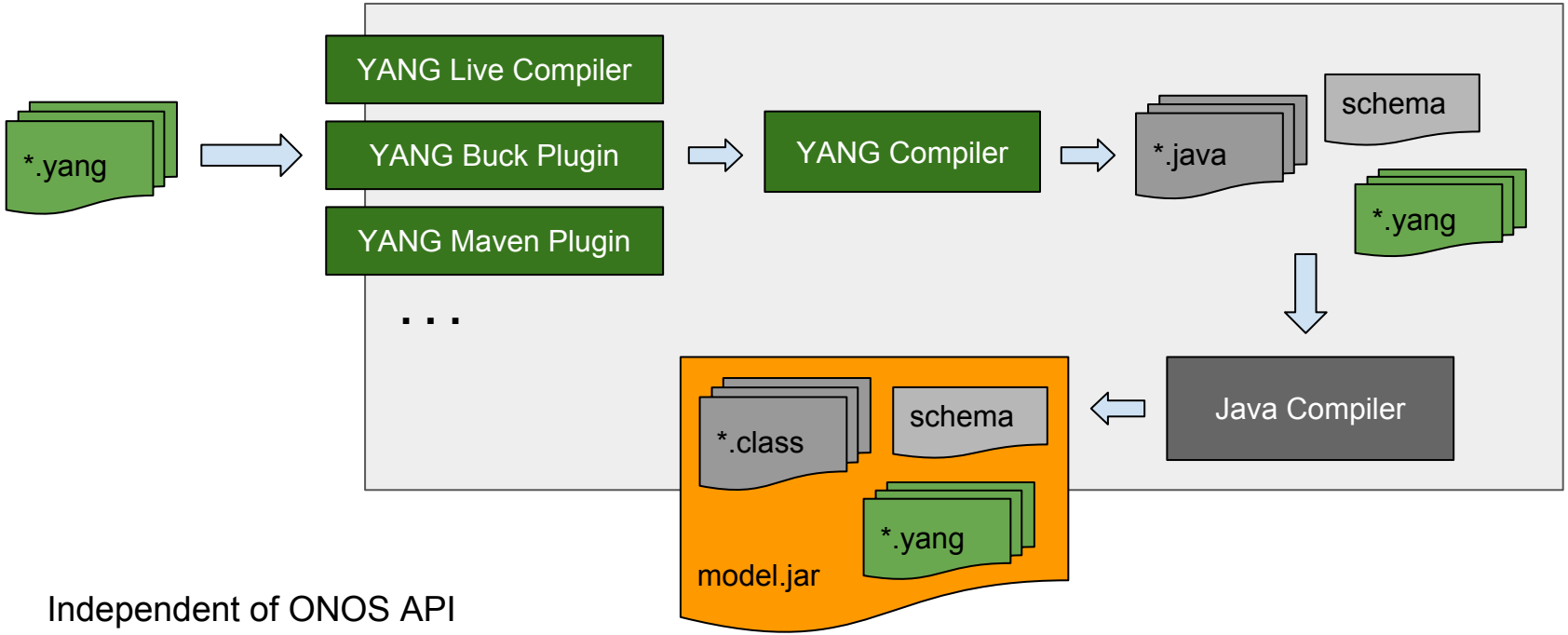
- There is merit in a blend of approaches
  - model-driven approach, YANG specifically, very important
  - emphasis on *standard* models especially OpenConfig & IETF
- Avoid dependence on specific technology or protocol
  - provide support for OpenFlow, YANG/NETCONF, etc.
  - but don’t limit the platform solely to either
  - provide APIs whenever possible and appropriate
  - expose pass-through merely as a fallback option
- Exploit technology, do not fall victim to it

ONOS YANG tools

YANG Tool Chain

- Few options for open-source YANG tools for Java
  - did not meet ONOS needs
  - either did not support required language features
  - or were strongly tied to their own platform (ODL YANG tools)
- ONOS community built standalone YANG tools
  - independent of ONOS platform in any way
  - initial availability as part of the *Kingfisher* release
- YANG Parser, Compiler, Code-Generator & Runtime
  - artifacts usable outside the context of ONOS, or even OSGi
  - include build plugins for Maven, Buck & shortly also for Bazel
  - support model-agnostic & model-specific data representation

YANG Tool Chain



- ✓ Independent of ONOS API
- ✓ Supports model-agnostic data traversal
- ✓ Generates schema for run-time validation and encoding/decoding
- ✓ Generates model-specific rich data types

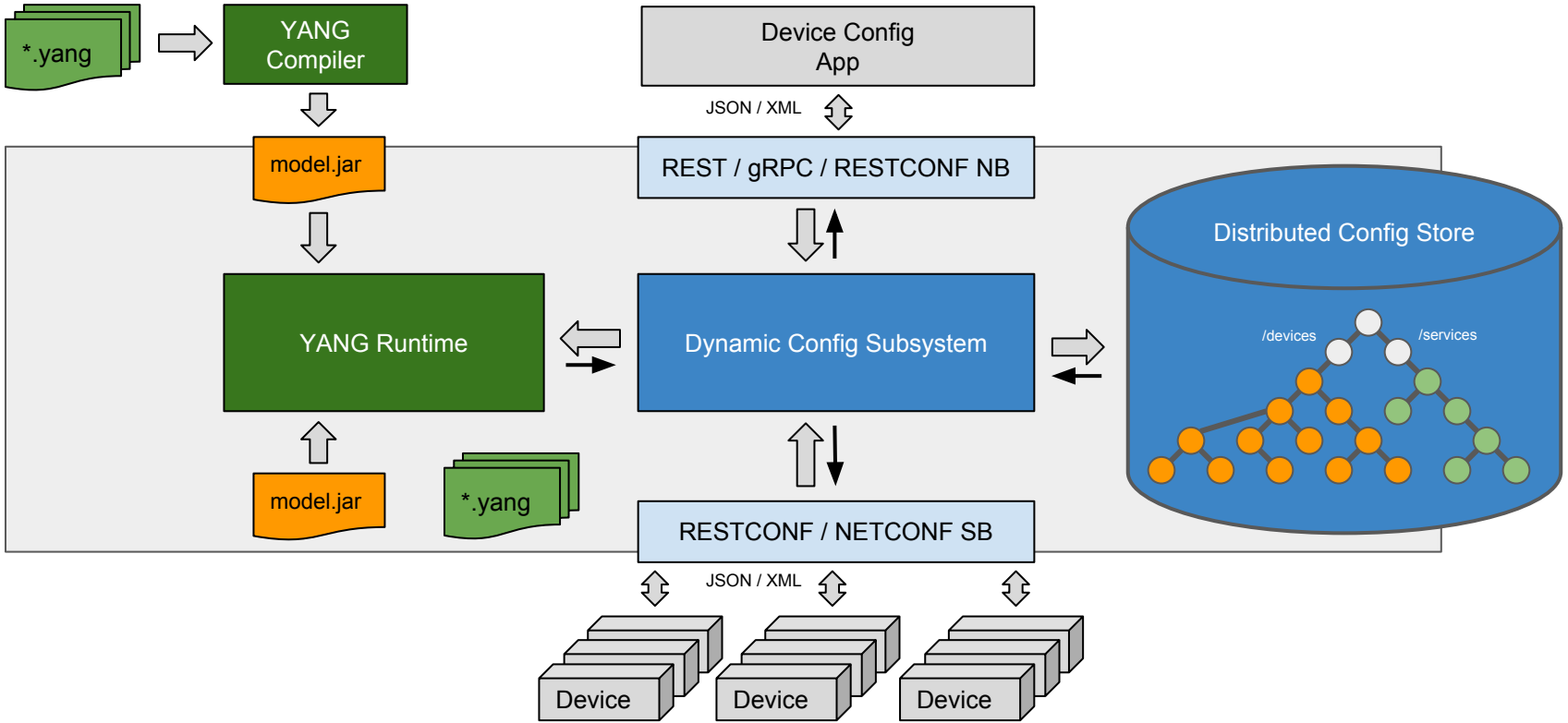
Models as ONOS extensions

- Compiled YANG models shipped as ONOS extensions
  - models compiled off-line via Maven, Buck or
  - models compiled on-line via YANG live compile feature
- JAR files suitable for both development and runtime
  - models can be downloaded from Maven central or from ONOS
  - models compiled on-the-fly can be downloaded from ONOS
- Number of standard model suites included
  - OpenConfig
  - Open ROADM
  - IETF (subset)

Major System Components

- YANG Compiler
  - processes YANG models to understand structure of data
  - generates model APIs and code that carries and conveys data
- YANG Runtime
  - transforms data between external and internal representations
- Protocol Adapters
  - ingest & emit data using various protocols, NETCONF, gRPC
- Information Store
  - persist and distribute data throughout the cluster of nodes
  - retain NB-to-SB edicts and SB-to-NB operational state

Major System Components



Design Considerations

- Framework components must be model-agnostic
  - e.g. runtime, protocol adapters, information store
  - must be capable dealing with any potential model
  - cannot be linked with the model-specific objects
  - hence must deal with data in a neutral format, i.e. raw tree
- Applications should be model-aware
  - e.g. applications that implement network services
  - must understand the semantics of the model to operate
  - raw tree representation not expressive or application friendly
  - hence should deal with data in domain-friendly model objects

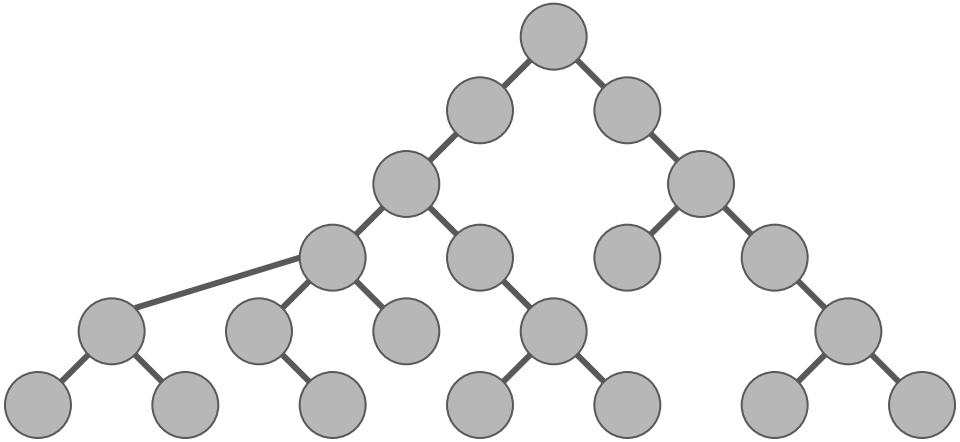
information store

Approaches to distributed stores

- Each ONOS subsystem has its own store
  - allows choice of distributed primitives, e.g. maps, sets, queues
  - allows choice of consistency strategy, e.g. strong, eventual
  - approach used by ONOS core and apps alike
  - offers very high performance
- Dynamic configuration store
  - allows configuration information modelled via YANG to be persisted without apps having to author their own stores
  - makes it easier to develop apps & services
  - may not support extremely high performance

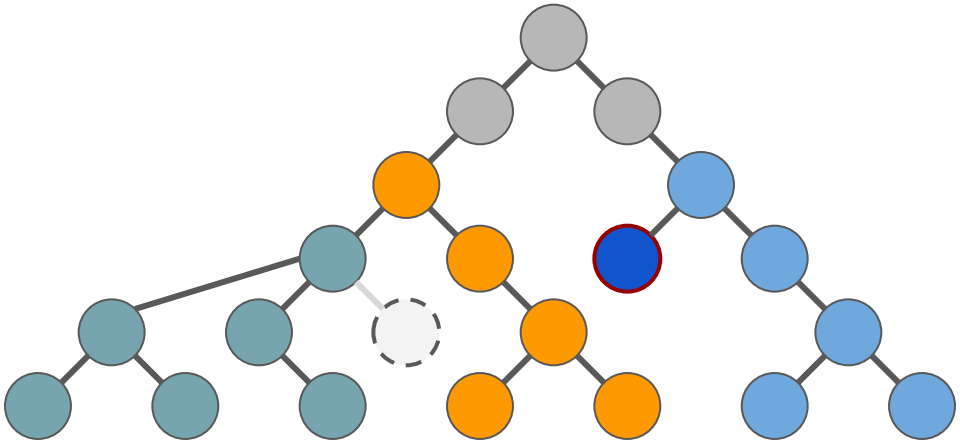
Information Store as a Tree

- YANG describes a logical tree structure
  - larger constructs built from smaller ones in a hierarchy
- Using tree structure to hold instance data is natural
  - individual data elements held in data nodes comprising the tree



Information Store as a Tree

- Adjustable to model augmentations & deviations
  - new nodes can be introduced, some can be removed
- Can be logically extended to aggregate information
  - many devices, many services under a unified tree structure

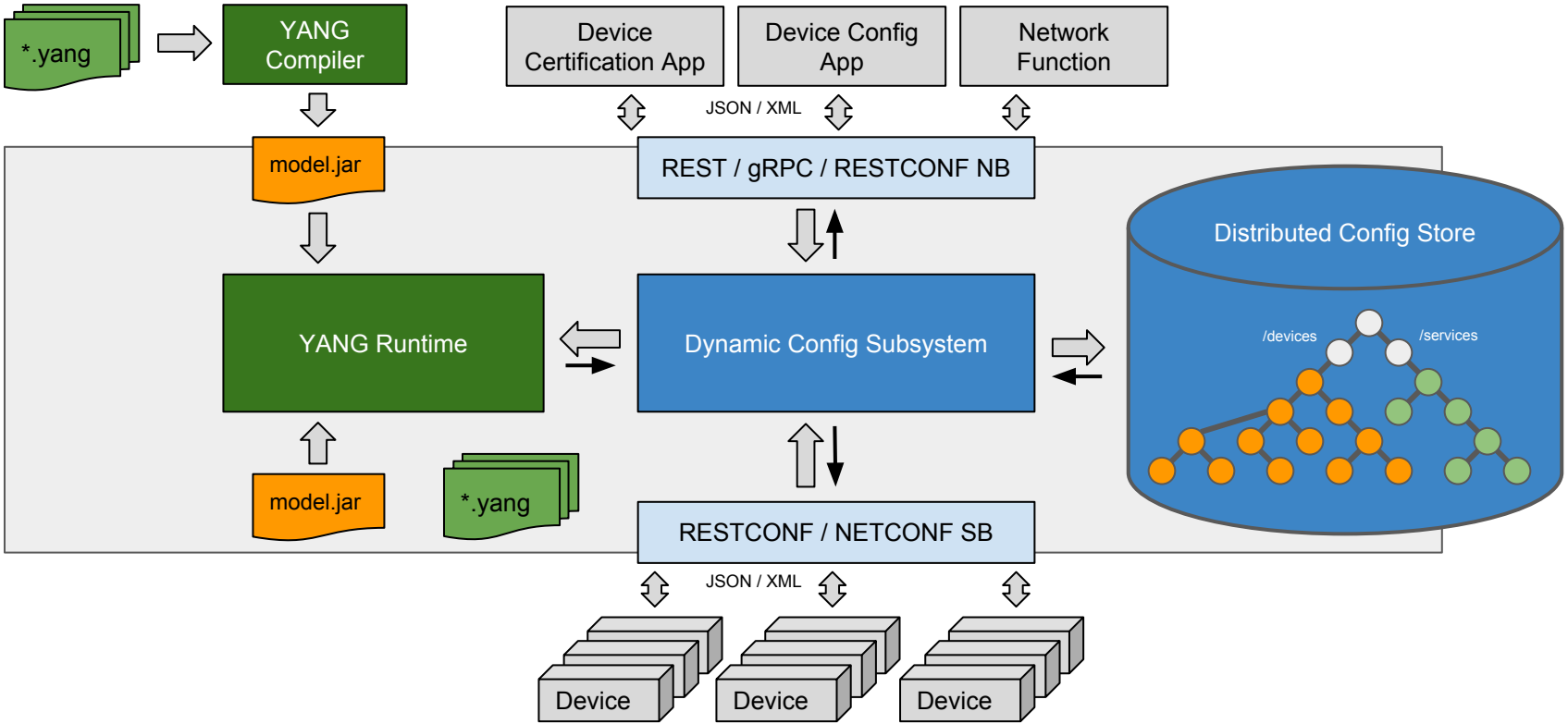


Information Store as a Tree

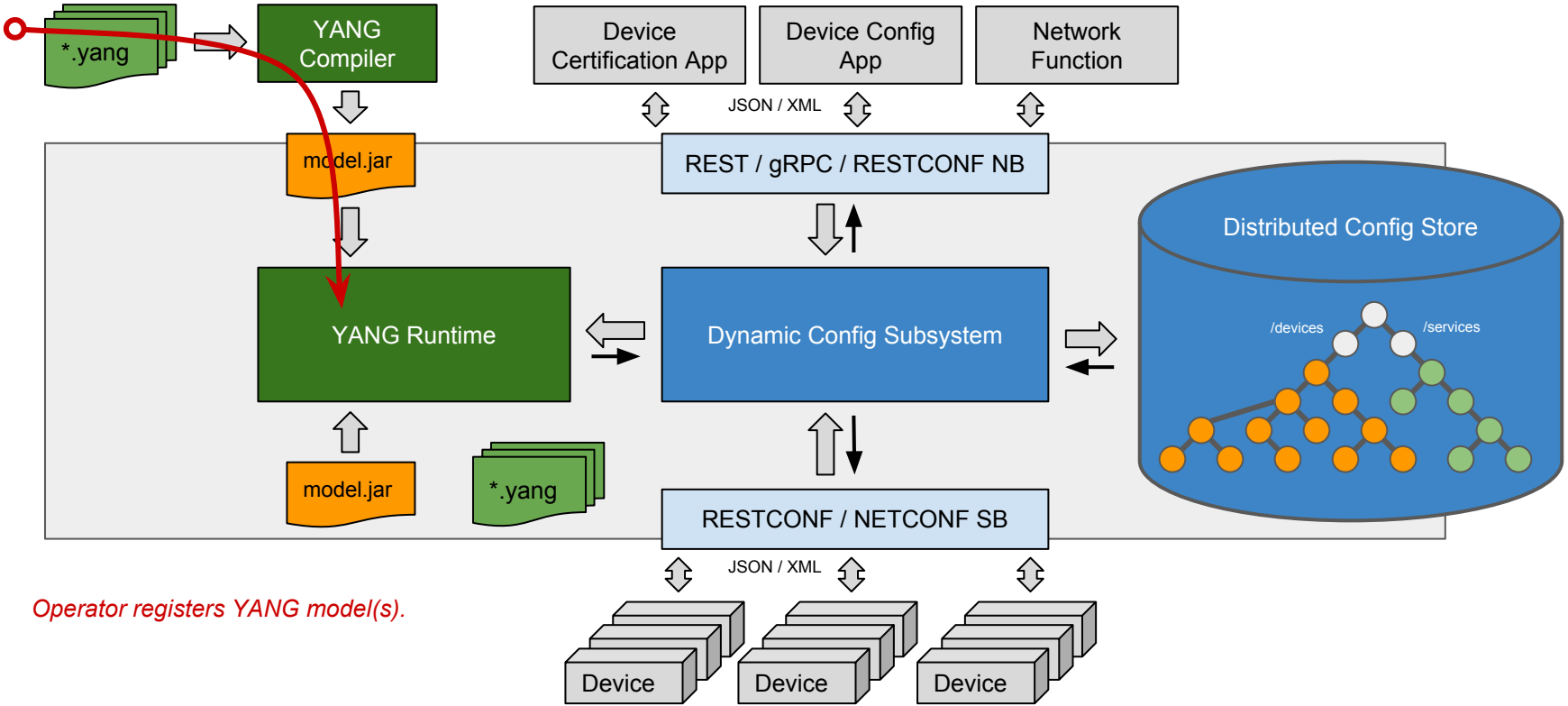
- Scalability challenge for large networks
  - requires partitioning and extensive optimizations to scale
  - partitions replicated to maintain performance & high-availability
  - addressing meta-information is disproportionately sized
  - high flexibility carries a fairly heavy tax
- ONOS Dynamic Configuration Store
  - today implemented as a fully-expanded tree
  - holds both configuration data and operational state
  - holds both service and device configurations
- Considering alternate approach for the long-term
  - easier to scale, but still offer significant flexibility

configuration scenario

Configuration of Devices

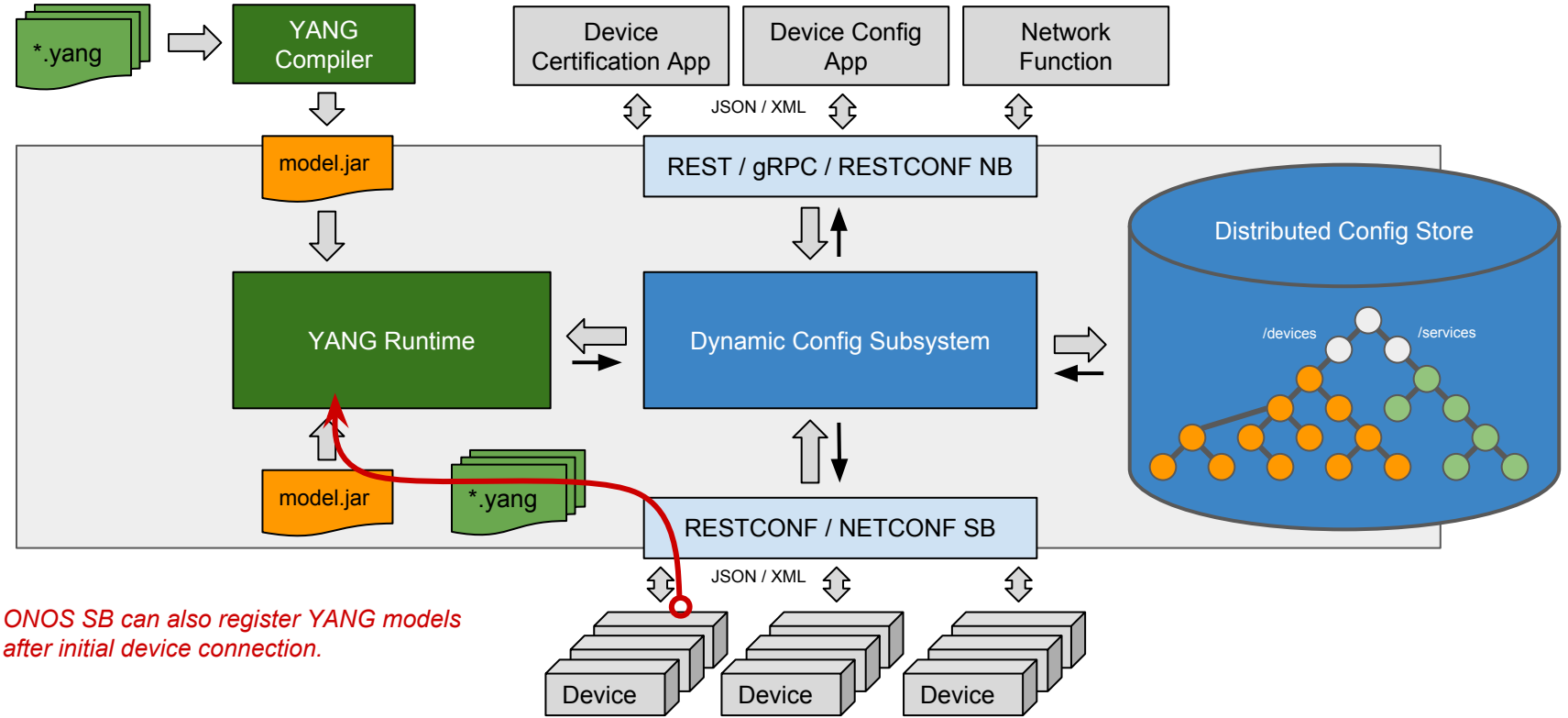


Configuration of Devices



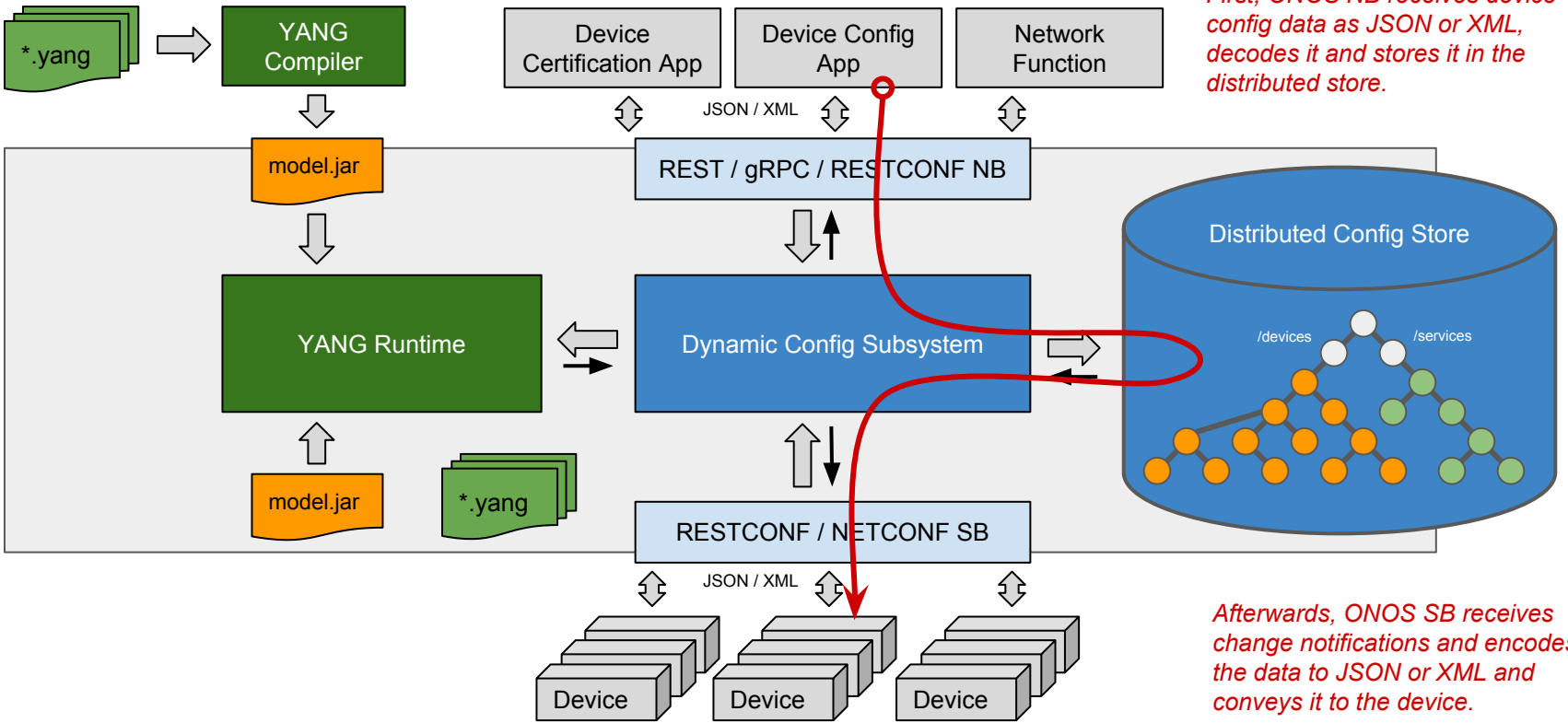
Operator registers YANG model(s).

Configuration of Devices



ONOS SB can also register YANG models after initial device connection.

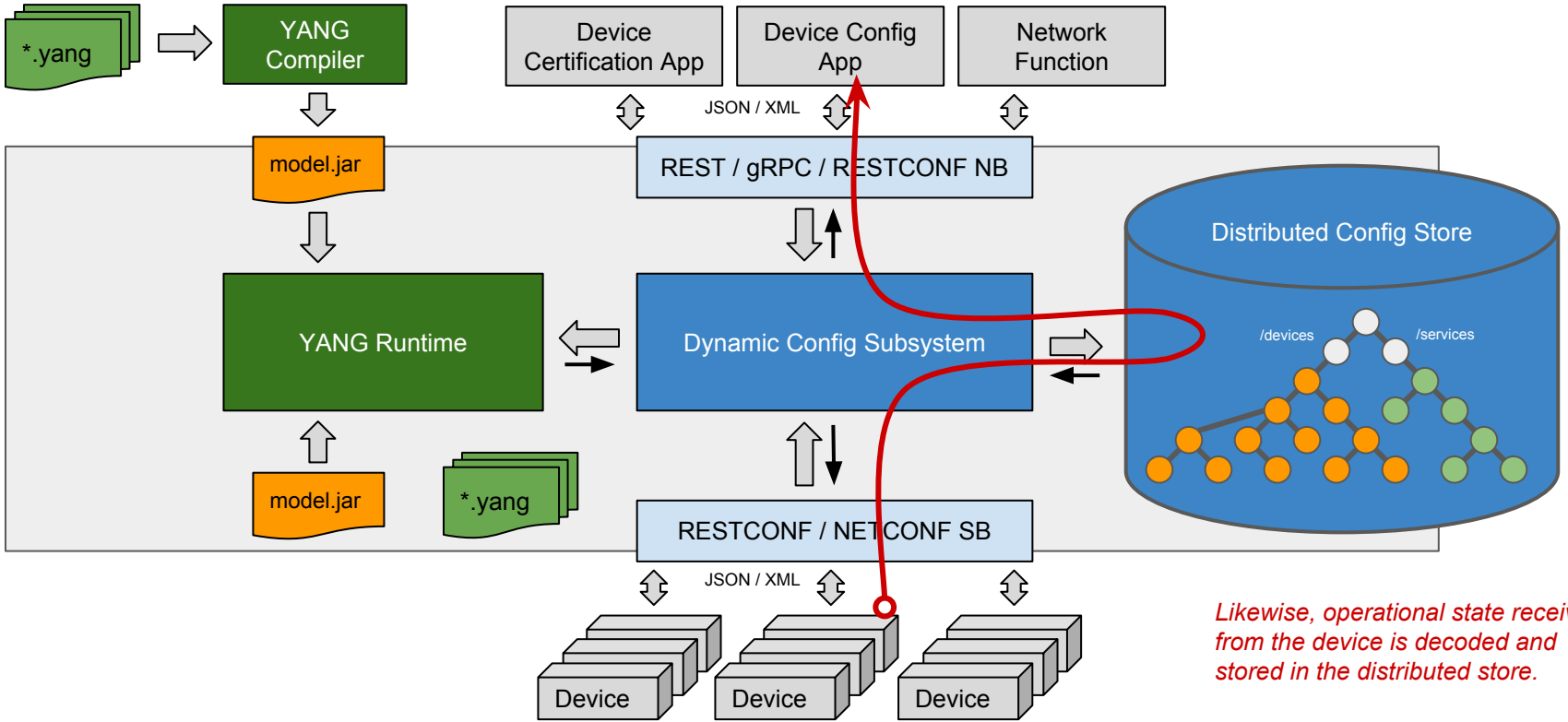
Configuration of Devices



First, ONOS NB receives device config data as JSON or XML, decodes it and stores it in the distributed store.

Afterwards, ONOS SB receives change notifications and encodes the data to JSON or XML and conveys it to the device.

Configuration of Devices



Likewise, operational state received from the device is decoded and stored in the distributed store.

roadmap & summary

ONOS YANG Tools features

- YANG Tools (Compiler & Runtime)
  - ONOS independent, Maven & Buck build plugins, live compilation
  - encode/decode between external and internal representations
  - transform model-agnostic tree to model-specific object structure
  - YANG 1.0 language support
  - support for OpenConfig models, YANG live compilation and YANG RPC
- Protocol Adapters
  - northbound RESTCONF, southbound RESTCONF (client)
  - southbound NETCONF (client)
- Distributed Dynamic Configuration Store subsystem
  - initial implementation, unified configuration tree

ONOS YANG Tools roadmap

- YANG Tools (Compiler & Runtime)
  - YANG 1.1 language support
- Protocol Adapters
  - RESTCONF notification (e.g. YANG-push) support
- Device Synchronizer
  - reconciliation between intended & actual state
- Distributed Dynamic Configuration Store subsystem
  - support for explicit transactions
  - 2-phase commit across multiple partitions, optimized sharding
- Configuration-based intents
  - incorporate configuration activities into the intent subsystem
  - mixing of control & configuration actions

Summary

- Control and configuration are both essential
  - operators need a platform capable of both
  - managing network elements and network services alike
- Framework design choices make a difference
  - impact on flexibility, ease-of-use
  - impact on high-availability, scalability and performance
- ONOS is a platform that supports both
  - built with stringent needs of dynamic control from the start
  - extended to provide fluidity required by dynamic configuration

Software Defined Transformation of Service Provider Networks

Join the journey @ onosproject.org