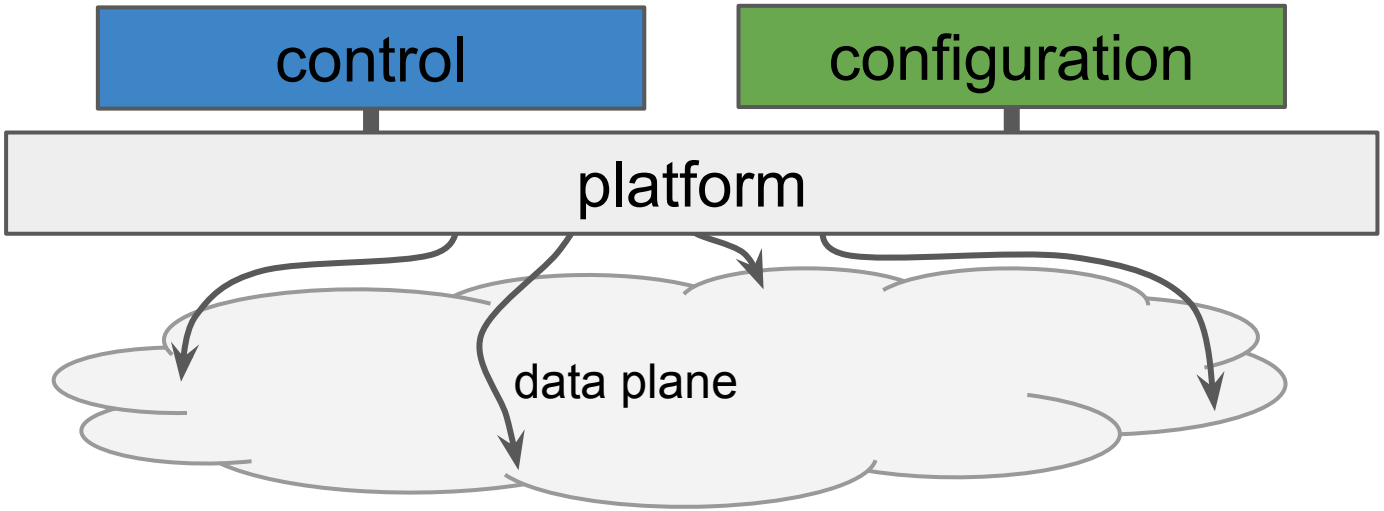
Dynamic Configuration

Brian O'Connor (ONF)
Open Networking Korea
23 April 2018

Control and Configuration

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



ONOS initially focused on control, but we have been improving the configuration subsystem over the past several releases.

Approaches to building APIs

API-driven

- Abstraction is specified via an explicit, programmatic API

Model-driven

- Abstraction is modelled using a domain specific language
- Programmatic API is model-agnostic

ONOS uses both types. For configuration, **Behaviours** are the API-driven approach, and **Dynamic Configuration** is the model-driven approach.

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

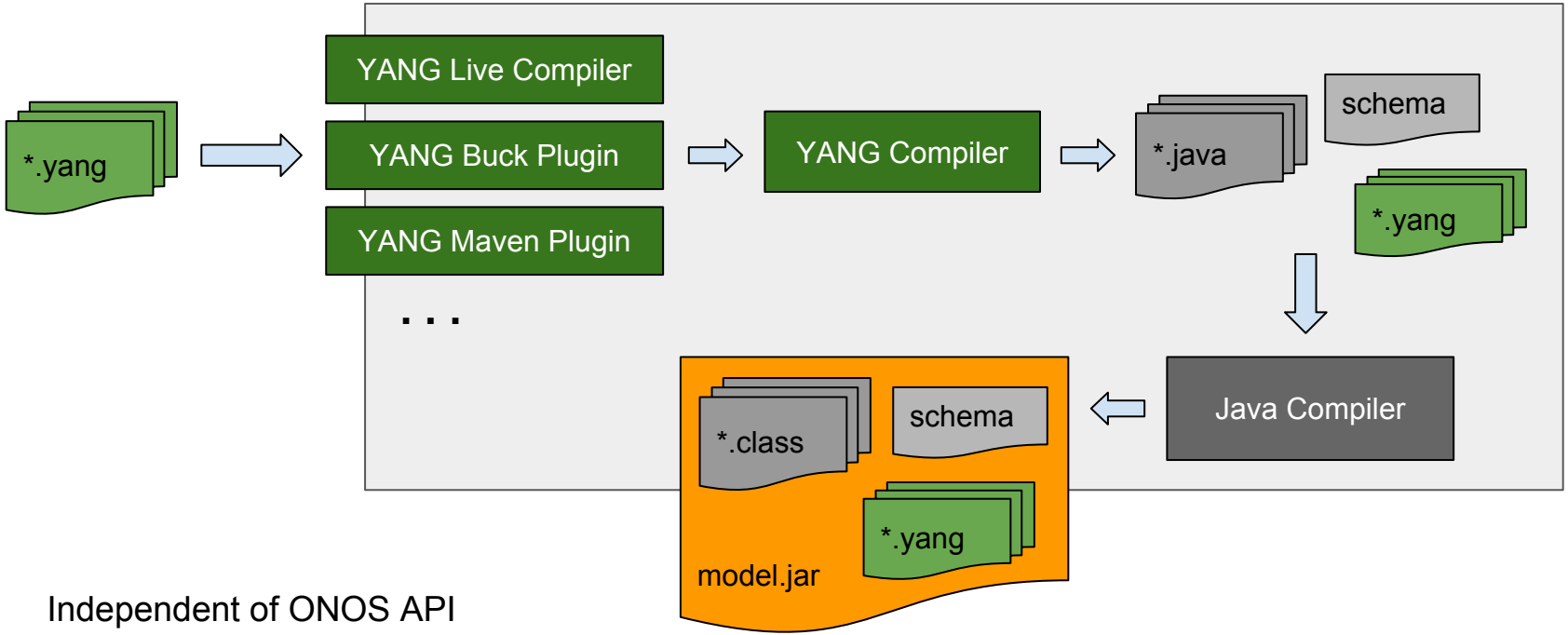
Mixed approach

- 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

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

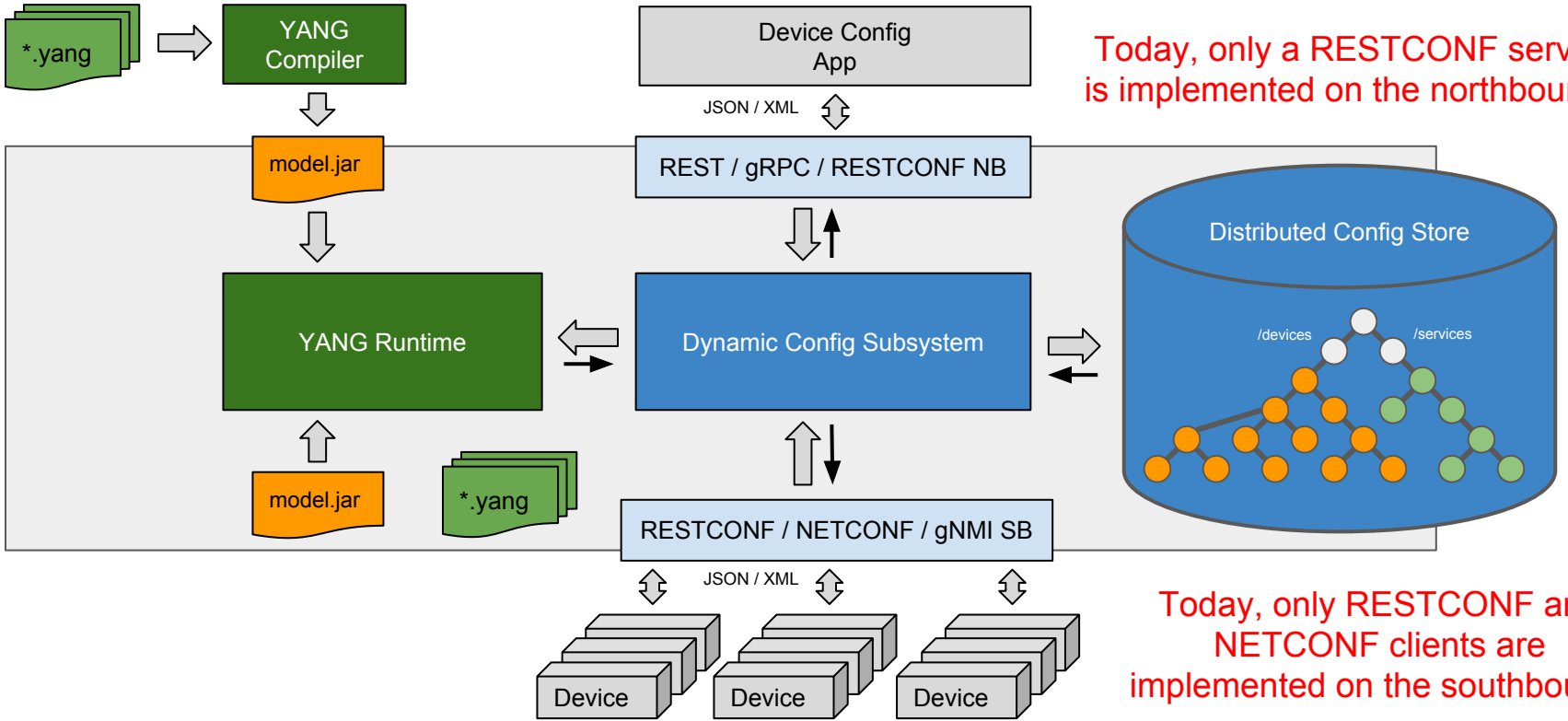
Models as ONOS extensions

- 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
- As of Nightingale release, 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

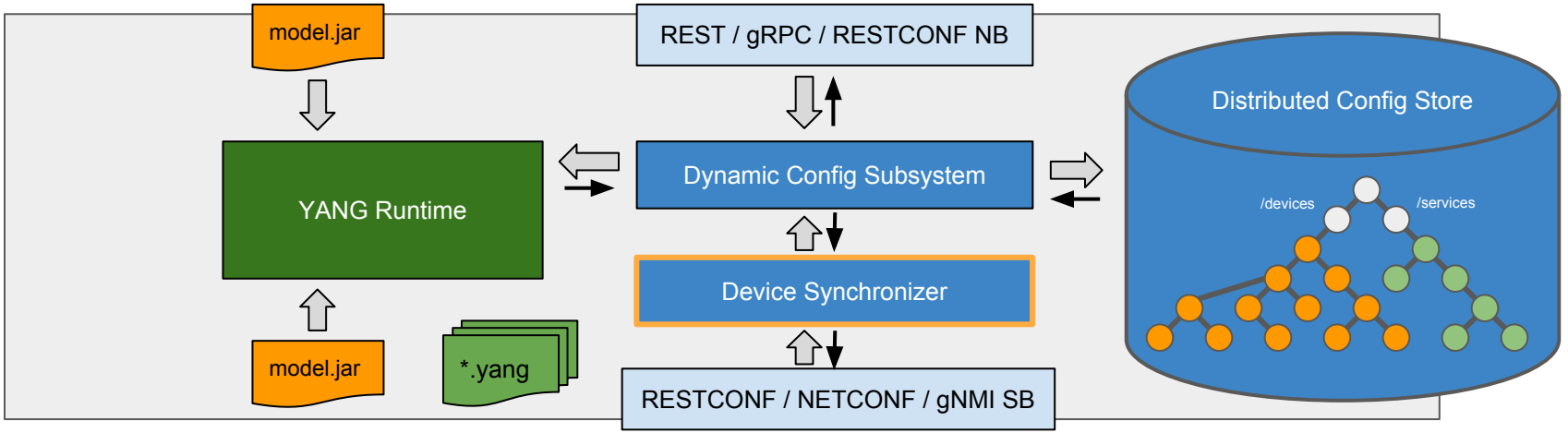


Today, only a RESTCONF server is implemented on the northbound.

Today, only RESTCONF and NETCONF clients are implemented on the southbound.

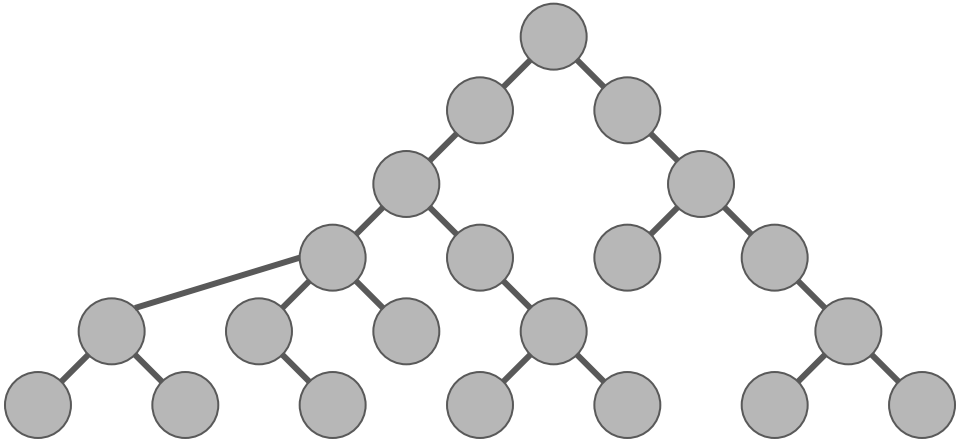
Device Synchronizer

- Initially, dynamic config system was limited to write-only
- Provides a way to get current device state and reconcile with desired state
- Basic implementation of synchronizer provided in Nightingale release



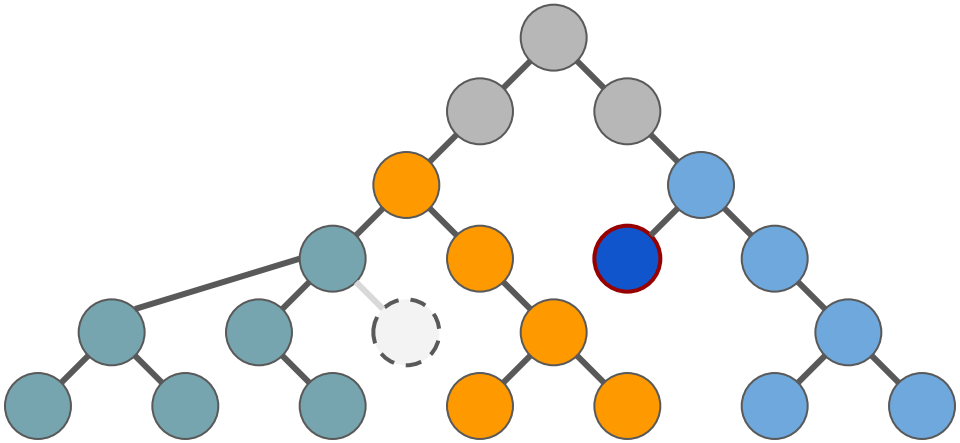
Information Store as a Tree

- 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



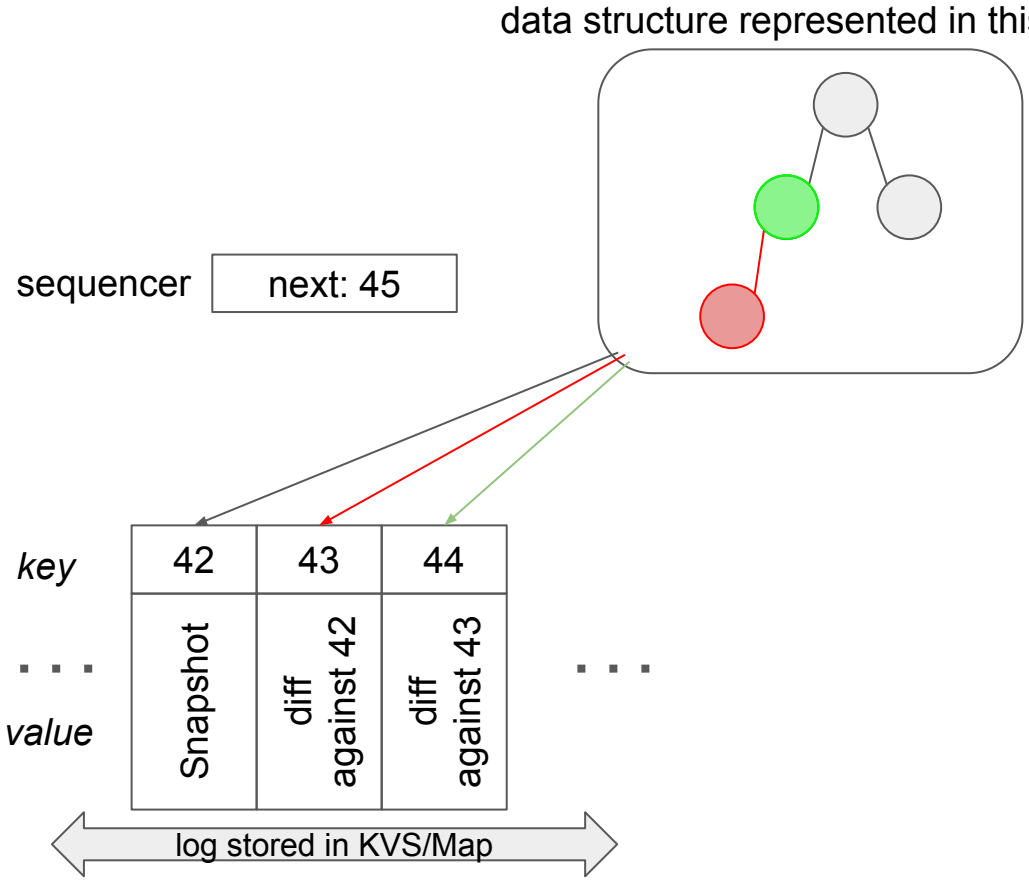
Information Store as a Tree

- ONOS Dynamic Configuration Store 1.0
  - today implemented as a fully-expanded tree
  - holds both configuration data and operational state
  - holds both service and device configurations
- 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 performance penalty

Proposed Improvements for DCS 2.0

- Support for batch operations as first class API
- Move to diff-oriented updated pattern (vs. existing snapshot-oriented one)
  - Better suited for JSON Patch (RFC6902 and RFC7386) and gNMI updates
  - Backed by tree store as data structure over distributed log
- Decoupling store implementation from ONOS Yang Tools binary objects
- Better scalability while preserving significant flexibility

Tree on Distributed Log



Two distributed primitives required:

- Sequencer (e.g., AtomicInteger)
- Key-Value Store [KVS] (e.g., EC or consistent Map)

Reader:

1. listen to KVS event
2. apply change set against local copy

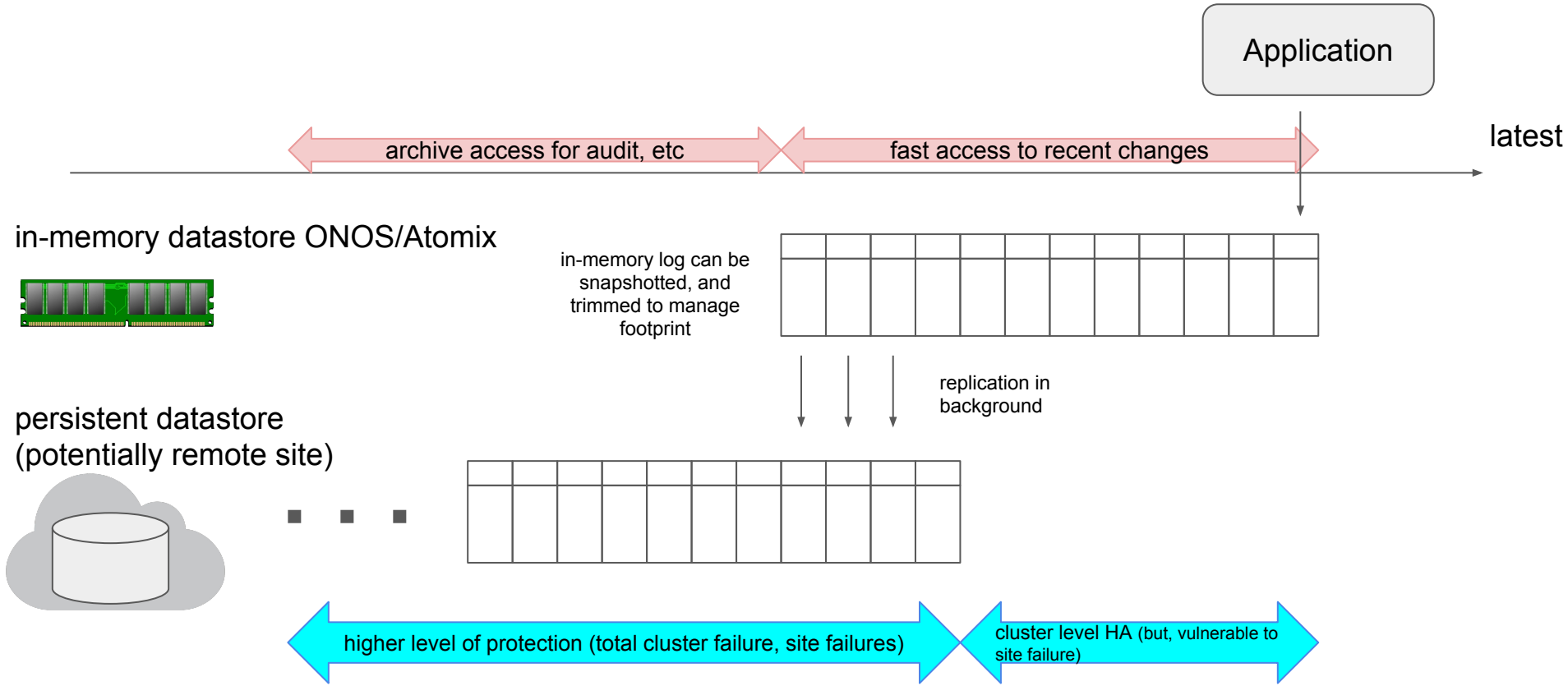
Writer:

1. increment index on sequencer
2. replay up to allocated index -1
3. test applicability of transaction
4. upon success, write entry to log on KVS

Benefit of Log-Based Tree

- Batch updates build on KVS batch update support (via transaction log)
- Notification frequency and size reduced to actual changes
- Configuration history is trivial to achieve via log and can be tuned

Option for Log Replication



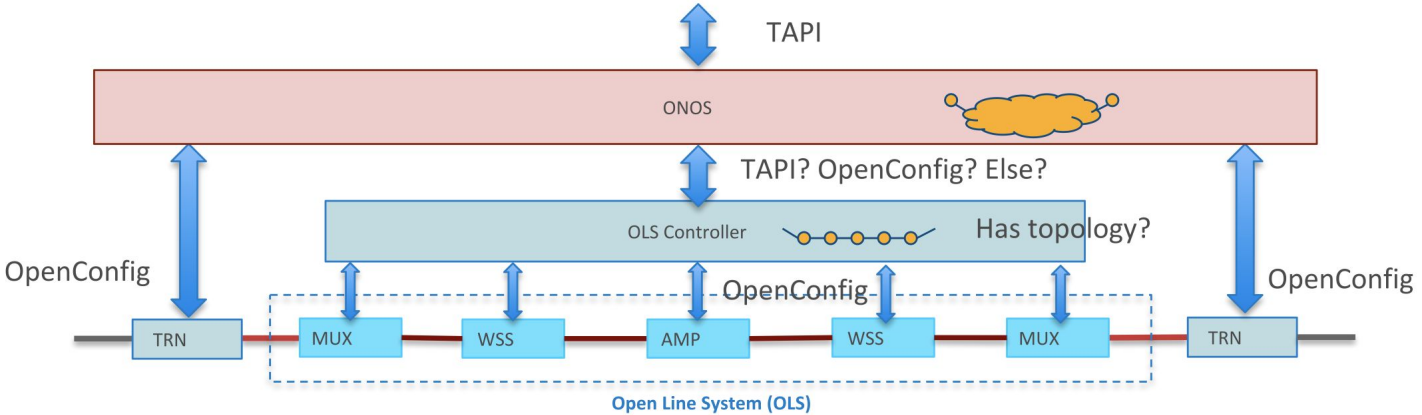
Why JSON as storage format?

- More concise then XML
  - smaller memory footprint
- Schema-less
  - Decouple storage system from any domain knowledge
- Easily printable for debugging, logging
- Better chance of leveraging existing libraries
  - RFC6902, etc.

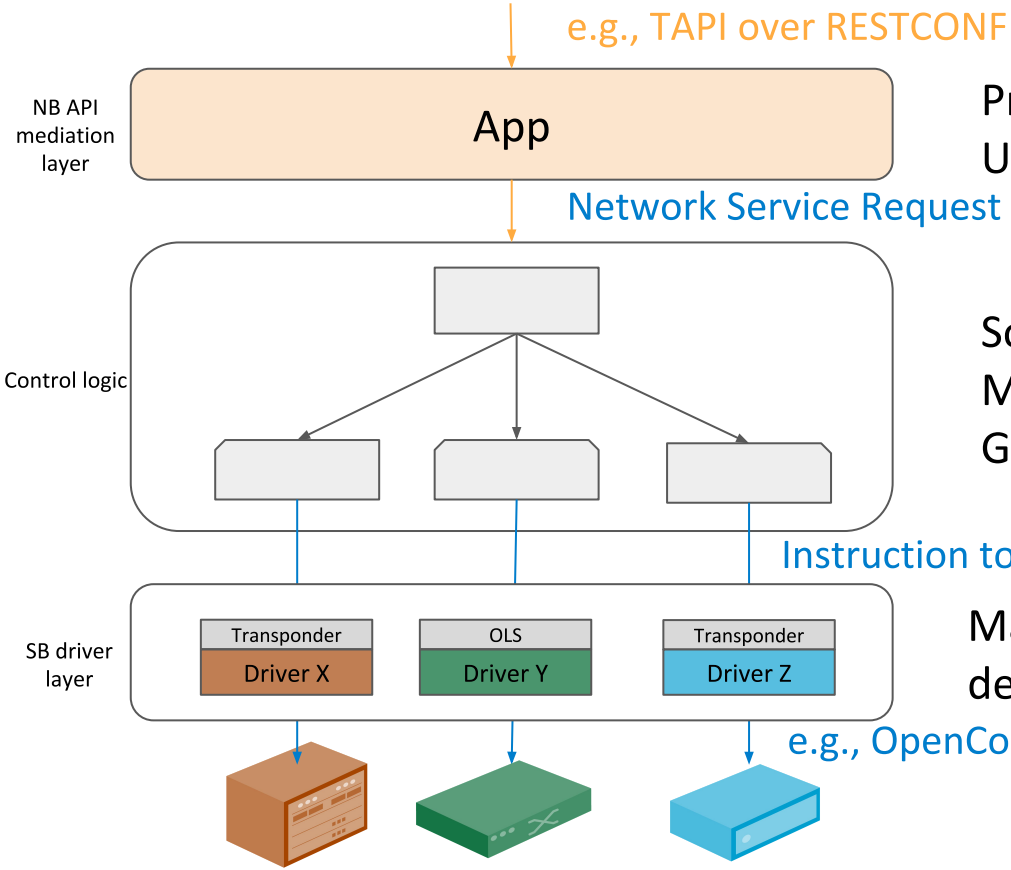
Note: nothing prevents us from replacing them with binary equivalent in the future (e.g. protobuf-defined)

ODTN Use Case

- Open and Disaggregated Transport Networks
  - Using open and common data models for both devices and services
  - Drive disaggregation of optical networks
- Bring ecosystem together to:
  - Build reference implementation using open source and open standards
  - Do lab and field trials



ODTN High Level Design



Protocol handling
Understand semantics of request

Solves constrained path
Manage resources
Generate device control and configuration

Map imposed semantics to commands that device understands

ODTN Requirements for Config

Initial goal is basic zero-touch provisioning functionality

- Obtain the current state from the device
- Configure the device - reconciling desired vs. current state (via Device Synchronizer)
- Rollback configuration on failure
  - Single update to single devices
  - Multiple updates to single device
  - Multiple updates to multiple devices

This use case/demo is currently driving Dynamic Config work and is targeting July 2018 for a demo.

roadmap

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; gNMI
- Device Synchronizer
  - reconciliation between intended & actual state
- Dynamic Configuration Store 2.0
  - support for explicit transactions / batches
  - Moved to log-based tree structure
- Configuration-based intents
  - incorporate configuration activities into the intent subsystem
  - mixing of control & configuration actions

More Information

https://wiki.onosproject.org/display/ONOS/Yang+Tools
https://wiki.onosproject.org/display/ONOS/Dynamic+Configuration+Subsystem

Dynamic Configuration Brigade

Email: brigade-dynconfig@onosproject.org

Bug and feature tracker:

https://jira.onosproject.org/secure/RapidBoard.jspa?rapidView=28&view=detail

Areas where you can help

- Lab setups and devices for testing
- DCS2.0 development
- gNMI support

Software Defined Transformation of Service Provider Networks

Join the journey @ onosproject.org