ONOS

Open Network Operating System

Architecture Update

ONRC October 2014

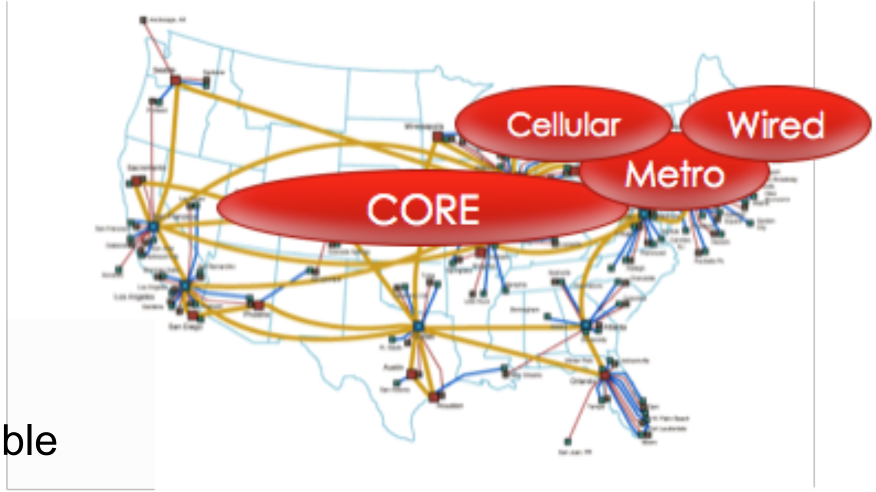
Thomas Vachuska
tom@onlab.us

ONOS:
SDN OS for Service Provider Networks

- Scalability, High Availability & Performance
- Northbound & Southbound Abstractions
- Modularity

Service Provider Networks

- WAN core backbone
  - Multi-Protocol Label Switching (MPLS) with Traffic Engineering (TE)
  - *200-500 routers, 5-10K ports*
- Metro Networks
  - Metro cores for access networks
  - *10-50K routers, 2-3M ports*
- Cellular Access Networks
  - LTE for a metro area
  - *20-100K devices, 100K-100M ports*
- Wired access / aggregation
  - Access network for homes; DSL/Cable
  - *10-50K devices, 100K-1M ports*



Key Performance Requirements

High Throughput:

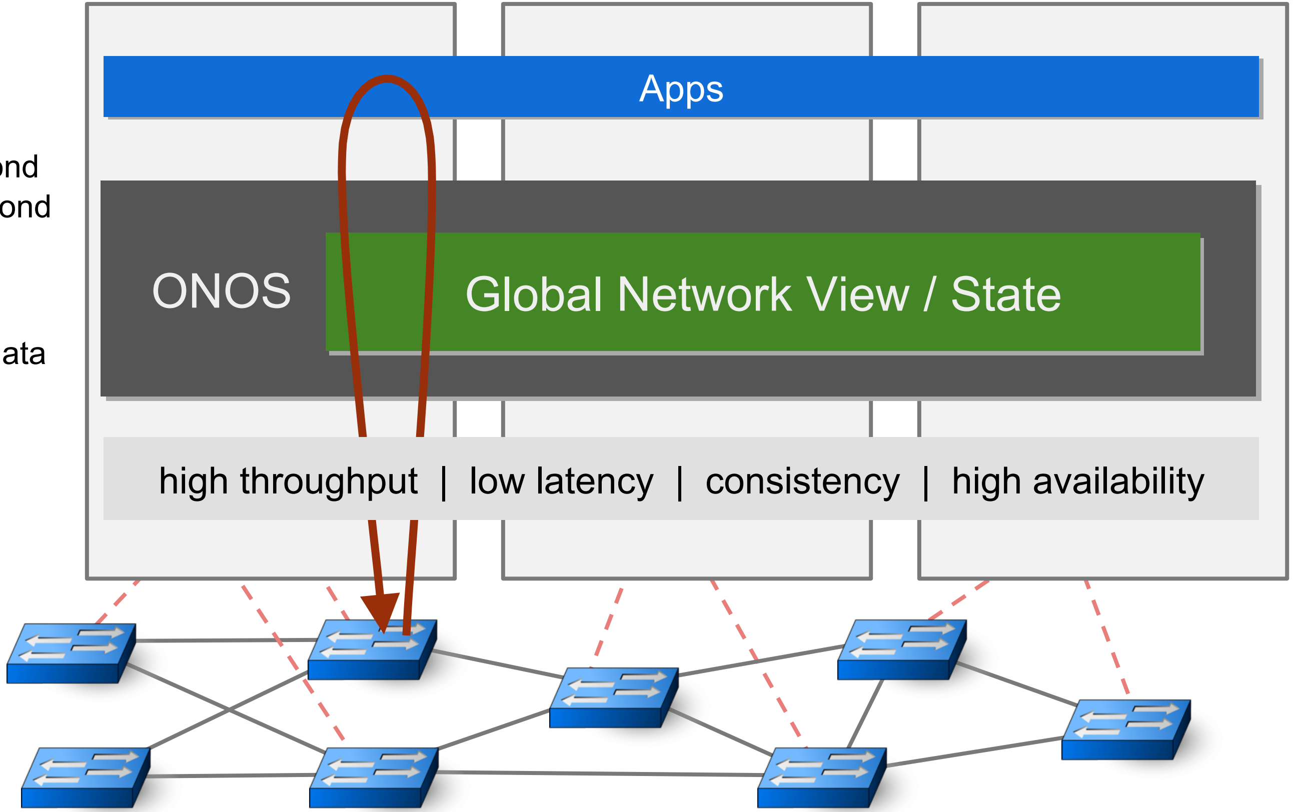
~500K-1M paths setups / second

~3-6M network state ops / second

High Volume:

~500GB-1TB of network state data

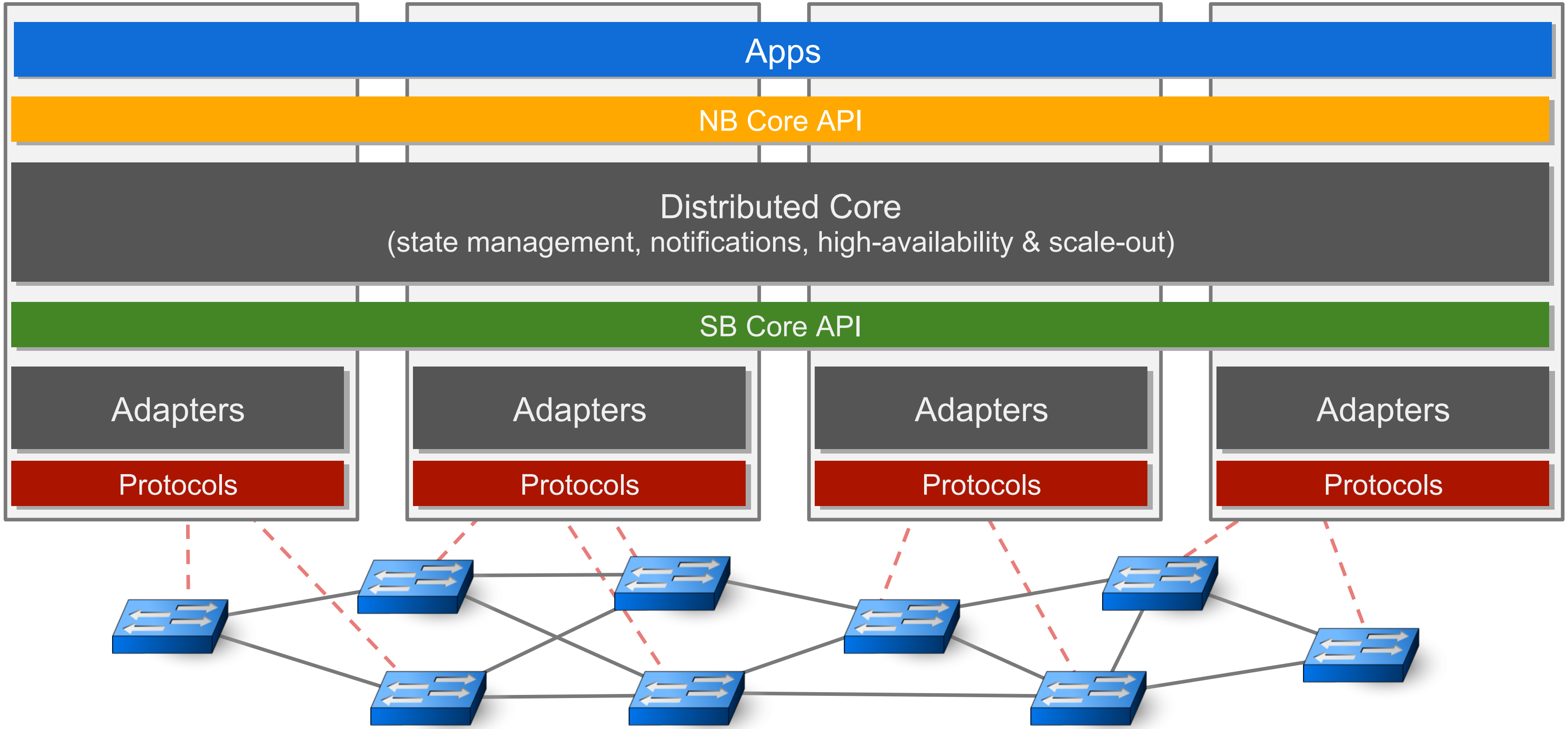
Difficult challenge!


Apps
ONOS
Global Network View / State
high throughput | low latency | consistency | high availability

Why Operating System?

- Provides useful services to applications
  - e.g. maintains connection persistence
- Provides framework for driving devices via arbitrary protocols
- Arbitrates shared network resources
- Provides abstractions to simplify resource sharing
  - application intent, network graph & device abstractions
- Isolates and protects resources, tenants & users
  - resource virtualization
- Comes with an SDK
  - APIs & docs, debugging, emulation, monitoring

Distributed Architecture



Apps

NB Core API

Distributed Core

(state management, notifications, high-availability & scale-out)

SB Core API

Adapters

Adapters

Adapters

Adapters

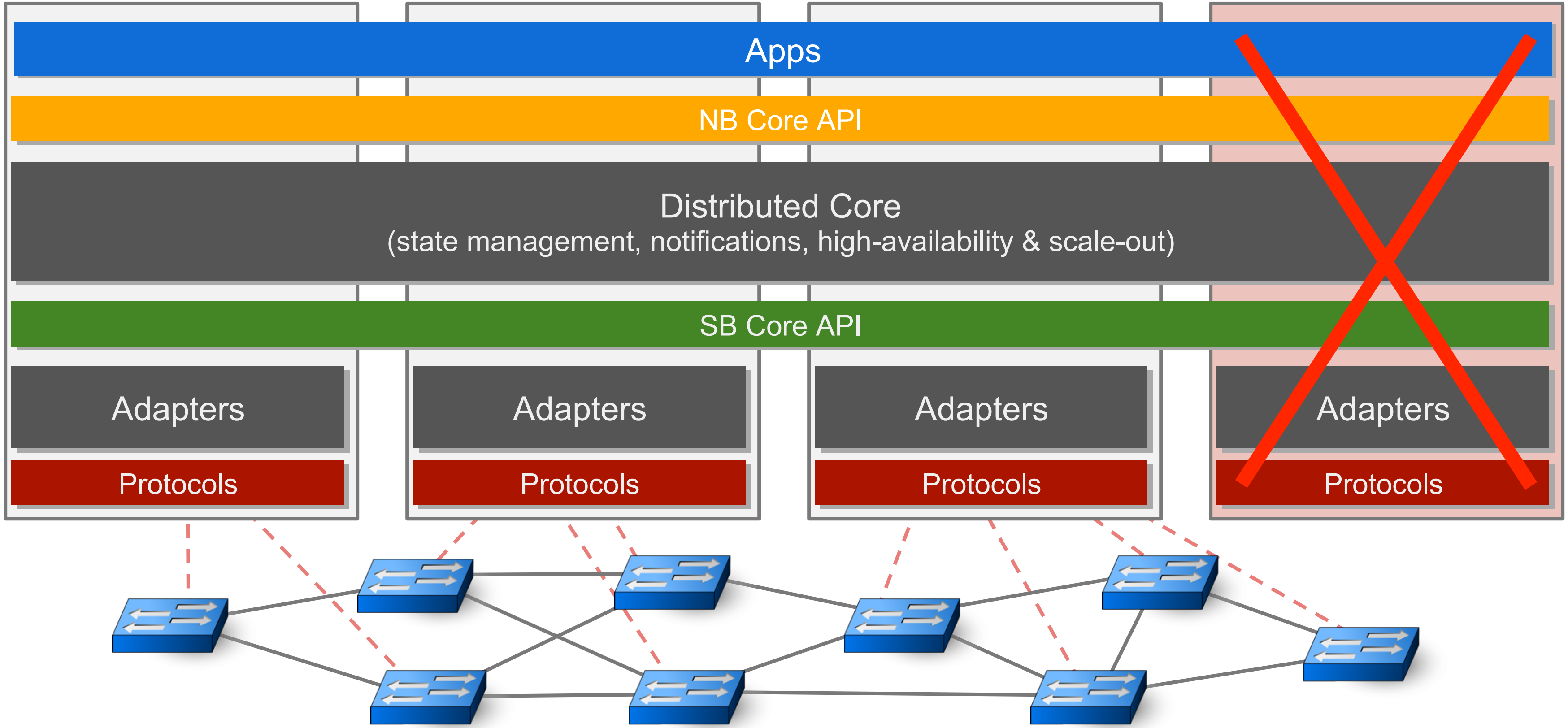
Protocols

Protocols

Protocols

Protocols

Distributed Architecture



Apps

NB Core API

Distributed Core

(state management, notifications, high-availability & scale-out)

SB Core API

Adapters

Protocols

Adapters

Protocols

Adapters

Protocols

Adapters

Protocols

ONOS Evolution

- Written in Java
- First prototype
  - basic functionality, OpenFlow 1.0
  - scale-out, high-availability, northbound graph abstraction
- Second prototype
  - performance, scale improvements over first generation
- Both
  - prototype quality code
  - OpenFlow as the only southbound protocol
  - relied heavily on open-source off-the-shelf components

ONOS November Release

- Many improvements to distributed core
  - revamped NB & SB interfaces
  - revamped distributed state management
- New abstraction & API
  - application intents
- New & pluggable southbound
  - OpenFlow 1.3 support
  - plugin architecture for legacy protocols
- Improved GUI & CLI
- Modularity
  - revamped code-base for modularity
  - built atop OSGi container - Apache Karaf

ONOS November Release

Northbound Abstraction:

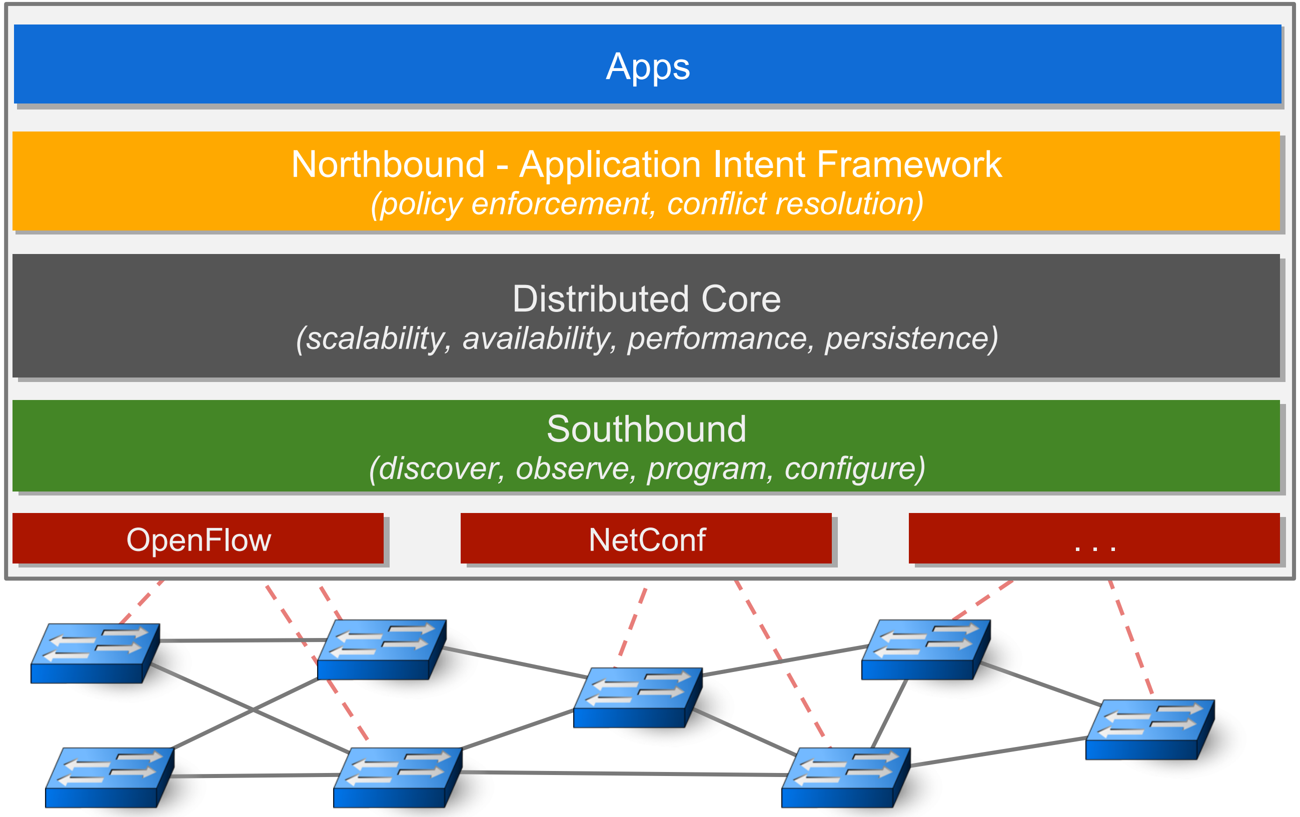
- network graph
- application intents

Core:

- distributed
- protocol independent

Southbound Abstraction:

- generalized OpenFlow
- pluggable & extensible



Apps

Northbound - Application Intent Framework

(policy enforcement, conflict resolution)

Distributed Core

(scalability, availability, performance, persistence)

Southbound

(discover, observe, program, configure)

OpenFlow

NetConf

. . .

Application Intent Framework

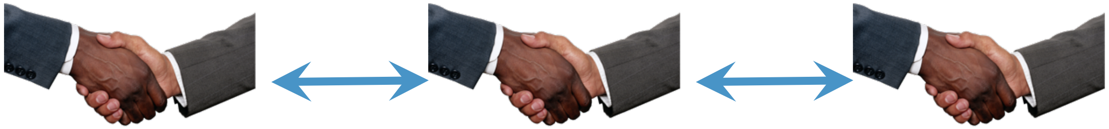
- Application specifies high-level intents; not low-level rules
  - focus on *what* should be done, rather than *how* it should be done
- Intents are compiled into actionable objectives which are installed into the environment
  - e.g. *HostToHostIntent* compiles into two *PathIntents*
- Resources required by objectives are then monitored
  - e.g. link vanishes, capacity or lambda becomes available
- Intent subsystem reacts by recompiling intent and re-installing revised objectives

Distributed Core

- Distributed state management framework
  - built for high-availability and scale-out
- Different types of state require different types of synchronization
  - fully replicated
  - master / slave replicated
  - partitioned / distributed
- Novel topology replication technique
  - *logical* clock in each instance timestamps events observed in underlying network
  - *logical* timestamps ensure state evolves in consistent and *ordered* fashion
  - allows rapid convergence without complex coordination
  - applications receive notifications about topology changes

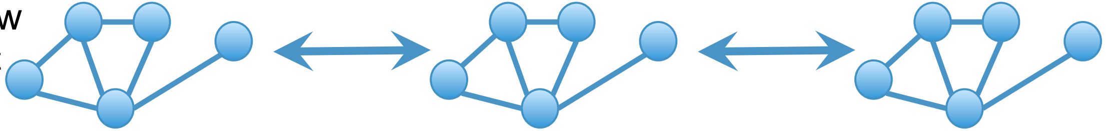
Distributed Core

Application Intents
- immutable
- durable & replicated



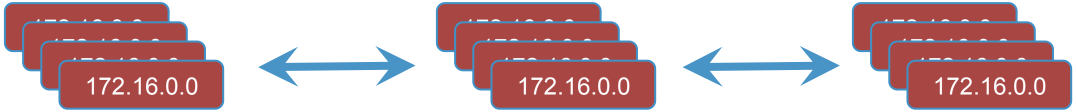
3-way replication
- H/A execution via distributed queues

Global Network View
- eventually consistent
- fully replicated



Optimistic Replication
- gossip based
- anti-entropy
- partial ordering

Flow Table Entries
- strongly consistent
- partitioned



Master/Backup Replication

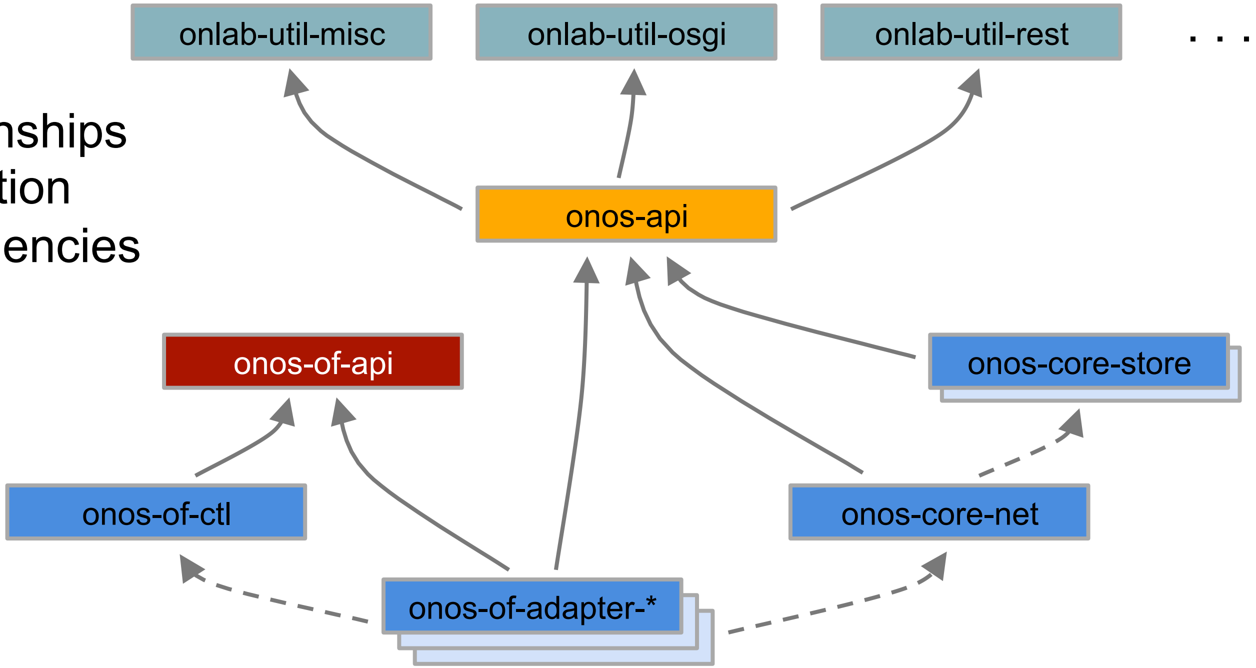
- Distribution & replication methods optimized for the type of state
- Based on size and read/write access patterns

Modularity Objectives

- Increase architectural coherence, testability and maintainability
  - establish tiers with crisply defined boundaries and responsibilities
  - setup code-base to follow and enforce the tiers
- Facilitate extensibility and customization by partners and users
  - unit of replacement is a module
- Avoid speciation of the ONOS code-base
  - APIs setup to encourage extensibility and community code contributions
- Preempt code entanglement, i.e. cyclic dependencies
  - reasonably small modules serve as firewalls against cycles
- Facilitate pluggable southbound

ONOS Modules

- Well-defined relationships
- Basis for customization
- Avoid cyclic dependencies



What’s coming on December 5th?

- ONOS with all its key features
  - scalability, high-availability, performance
  - northbound abstractions (application intents)
  - southbound abstractions (OpenFlow adapters)
  - modular code-base
- Open source
  - ONOS code-base on GitHub
  - documentation & infrastructure processes to engage the community
- Use-case demonstrations
  - SDN-IP, Packet-Optical
- Sample applications
  - reactive forwarding, mobility, proxy arp

Check out posters -- learn details from ONOS architects and developers