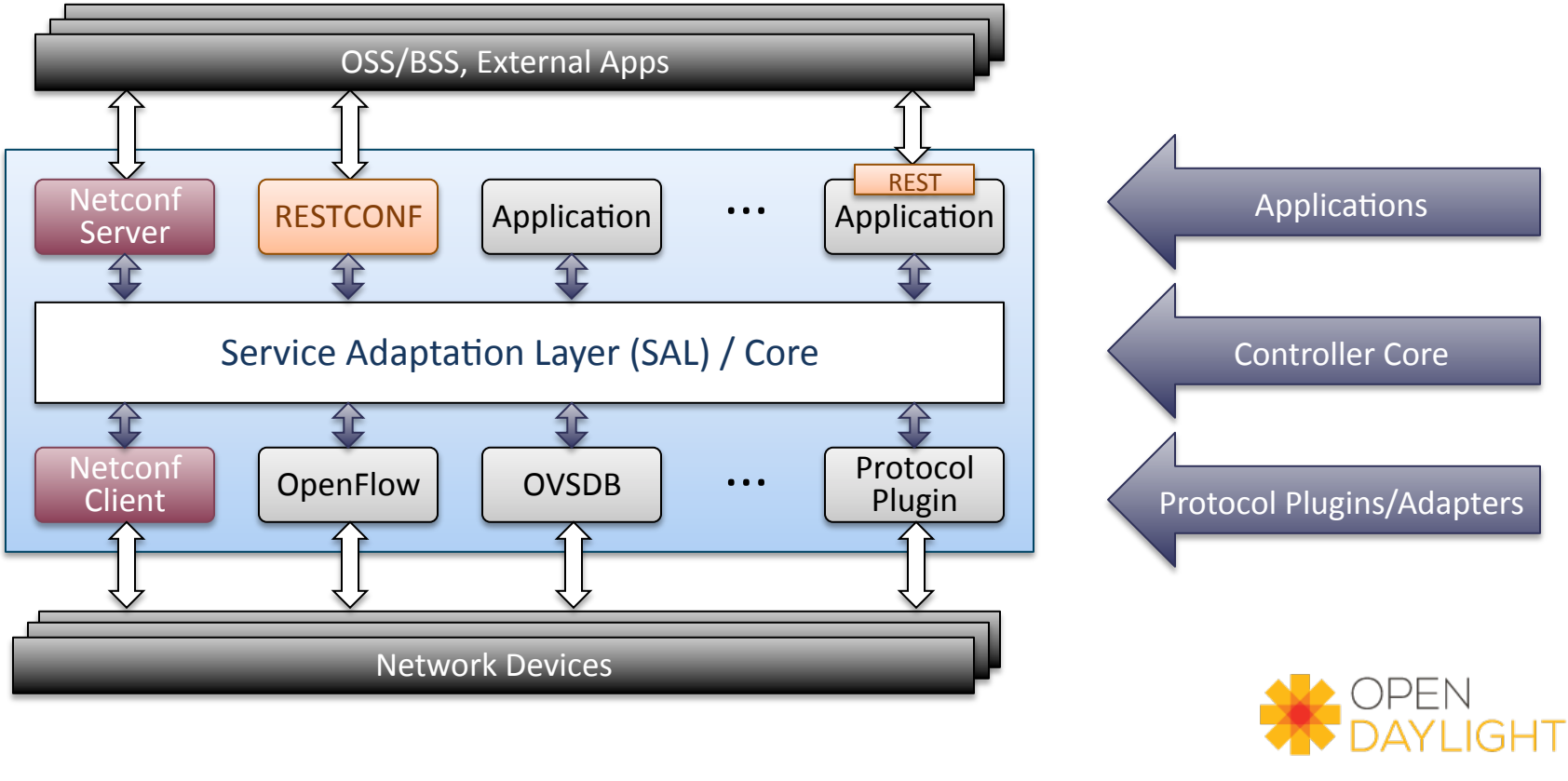
Clustering in OpenDaylight

Colin Dixon
Technical Steering Committee Chair, OpenDaylight
Distinguished Engineer, Brocade

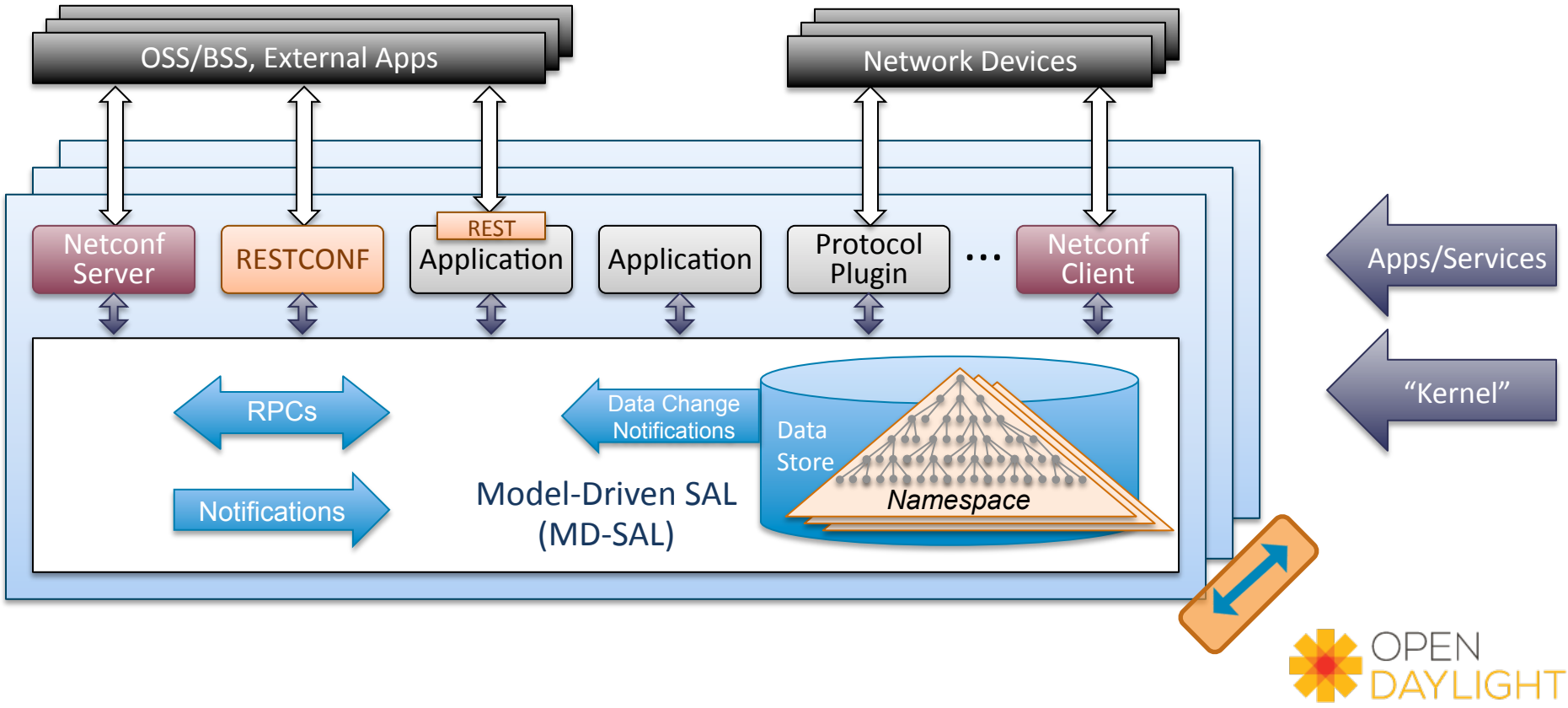
Borrowed ideas and content from Jan Medved, Moiz Raja, and Tom Pantelis

Multi-Protocol SDN Controller Architecture

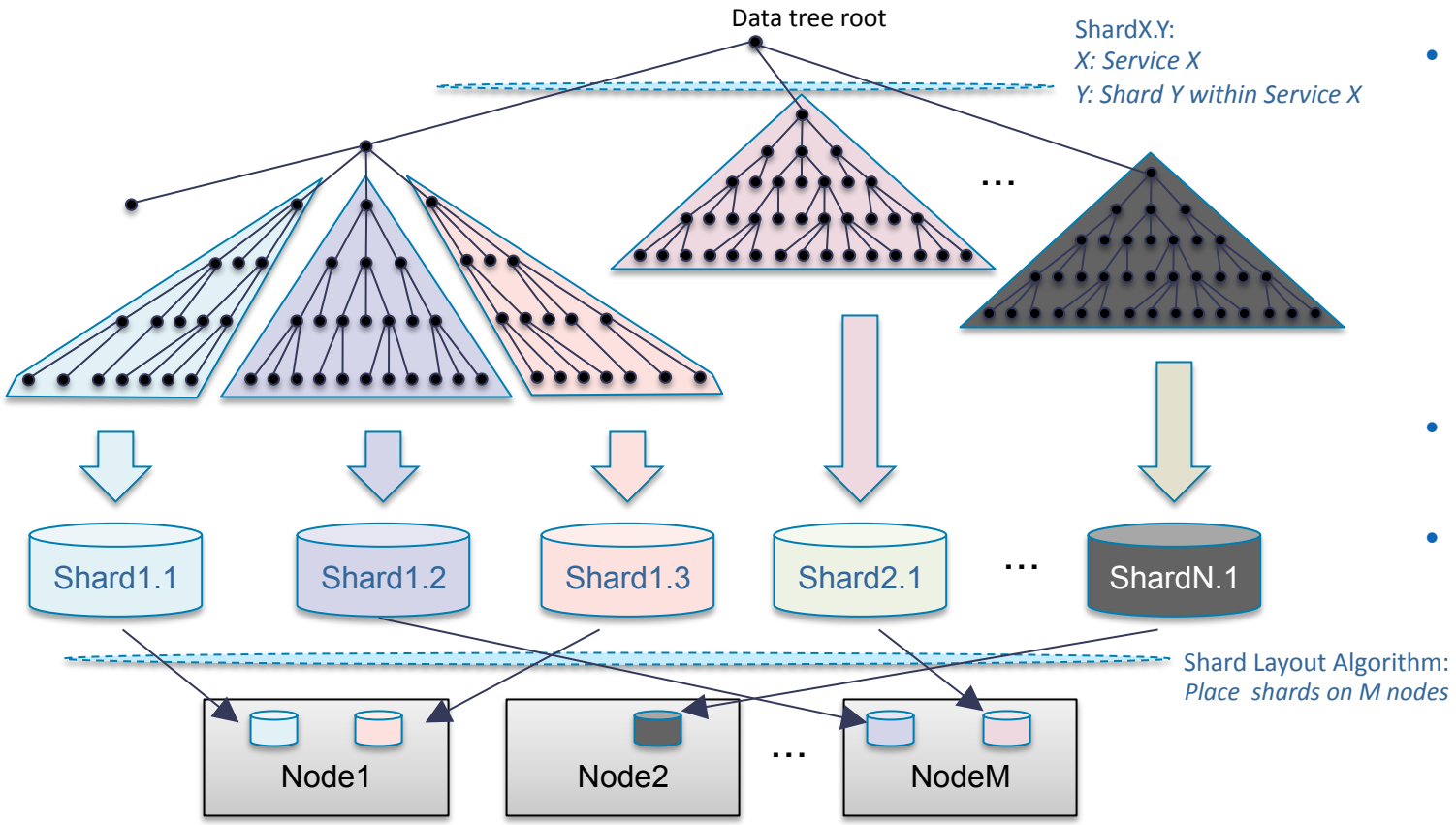


OPEN DAYLIGHT

Software Architecture



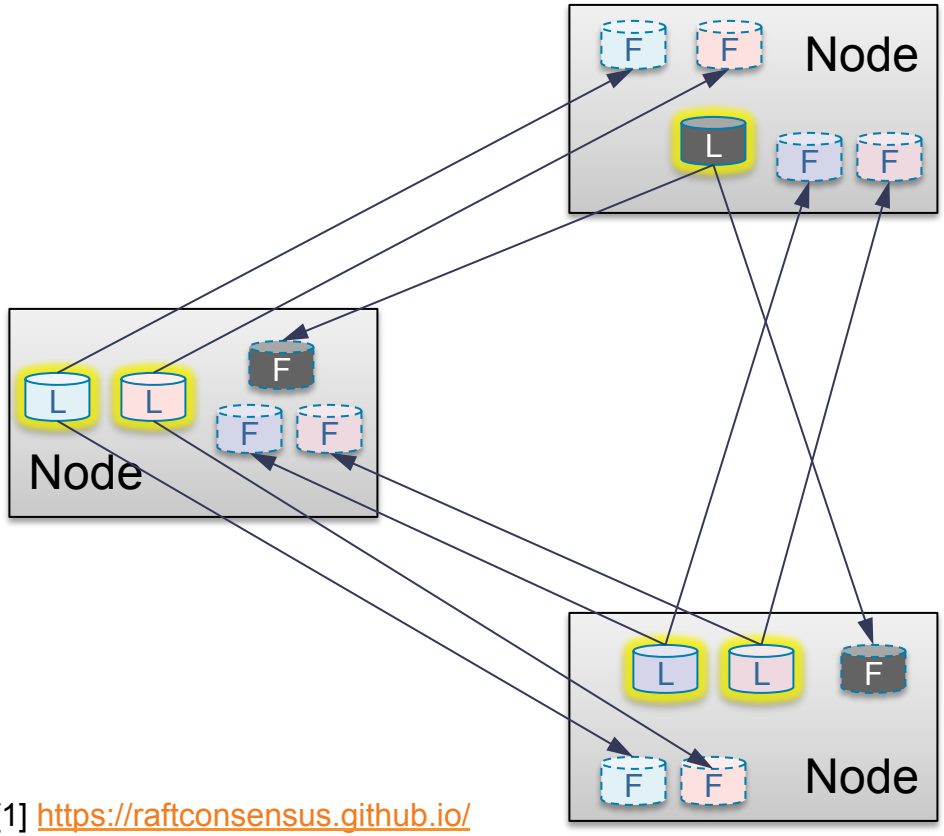
Data Store Sharding



- Select data subtrees
  - Currently, can only pick a subtree directly under the root
  - Working on subtrees at arbitrary levels
- Map subtrees onto shards
- Map shards onto nodes

OPEN DAYLIGHT

Shard Replication



- Replication using RAFT [1]
  - Provides **strong consistency**
  - Transactional data access
    - snapshot reads
    - snapshot writes
    - read/write transactions
  - Tolerates f failures with 2f+1 nodes
    - 3 nodes => 1 failure, 5 => 2, etc.
- Leaders handle all writes
  - Send to followers before committing
- Leaders distributed to spread load

[1] https://raftconsensus.github.io/

OPEN DAYLIGHT

Strong Consistency

- Serializability
  - Everyone always reads the most recent write. Loosely “everyone is at the same point on the same timeline.”
- Causal Consistency
  - Loosely “you won’t see the result of any event without seeing everything that could have caused that event.” [Typically in the confines of reads/writes to a single datastore.]
- Eventual Consistency
  - Loosely “everyone will eventually see the same events in some order, but not the necessarily the same order.” [Eventual in the same way that your kid will “eventually” clean their room.]

Why strong consistency matters

- A flapping port generates events:
  - port up, port down, port up, port down, port up, port down, …
  - Is the port up or down?
  - If they're not ordered, you have no idea.

  - …sure, but these are events from a single device, so you can keep them ordered easily

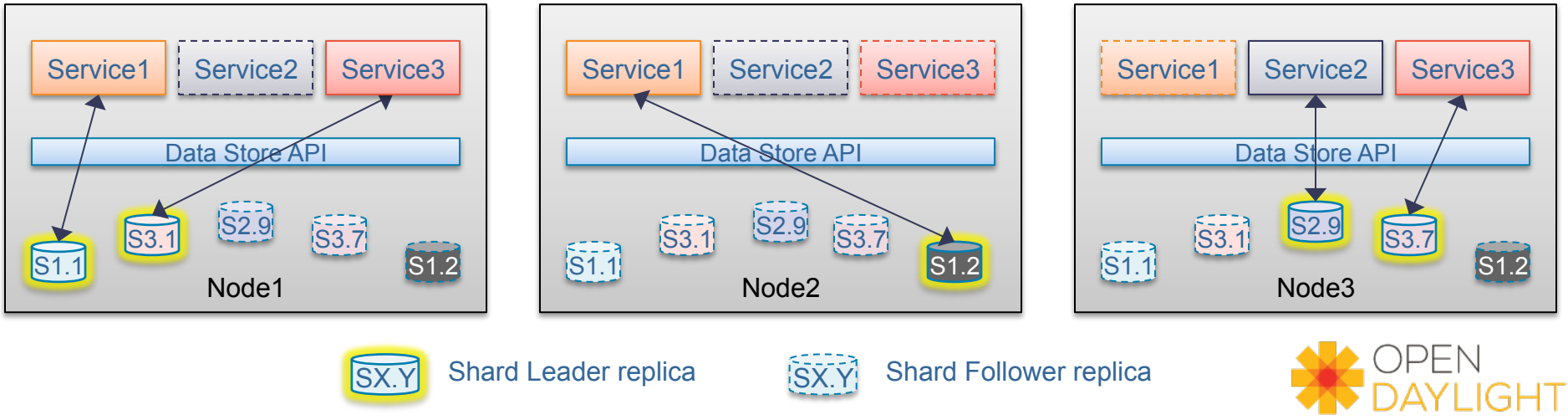
- More complex examples
  - switches (or ports on those switches) attached to different nodes go up and down
  - If you don't have ordering, different nodes will now come to different conclusions about reachability, paths, etc.

Why everyone doesn't use it

- Strong consistency can't work in the face of partitions
  - If you're network splits in two, either:
    - one side stops
    - you lose strong consistency
- Strong consistency requires coordination
  - Effectively, you need some single entity to order everything
  - This has performance costs, i.e., a single strongly consistent data store is limited by the performance of a single node
- Question is: do we start with strong and relax it or start weak and strengthen it?
  - OpenDaylight has started strong

Service/Application Model

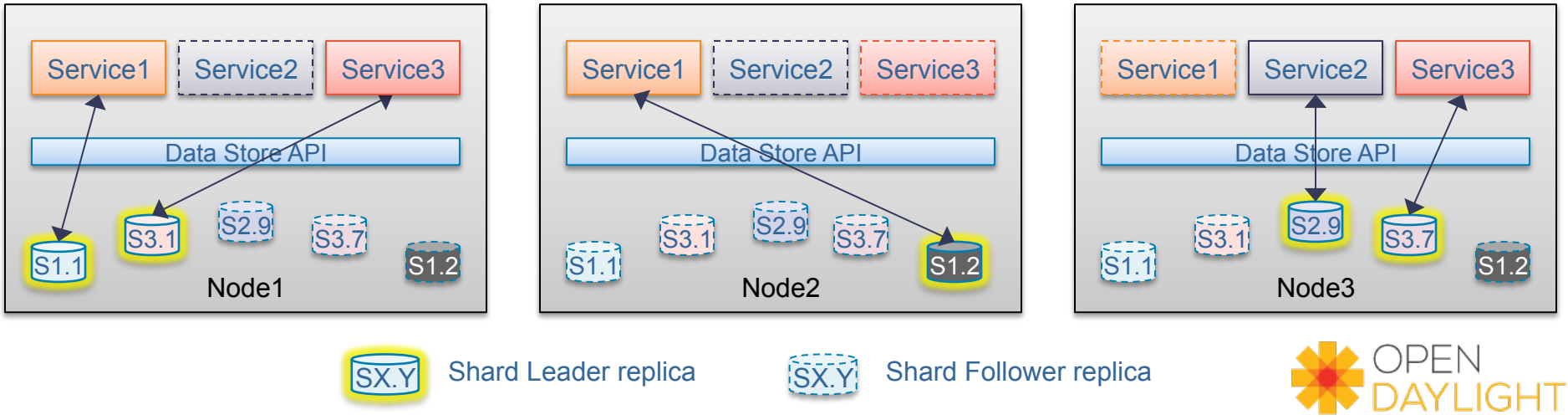
- Logically, each service or application (**code**) has a primary subtree (**YANG model**) and shard it is associated with (**data**)
- One instance of the **code** is co-located with each replica of the data
- All instances are stateless, all state is stored in the data store
- The instance that is co-located with the shard leader handles writes



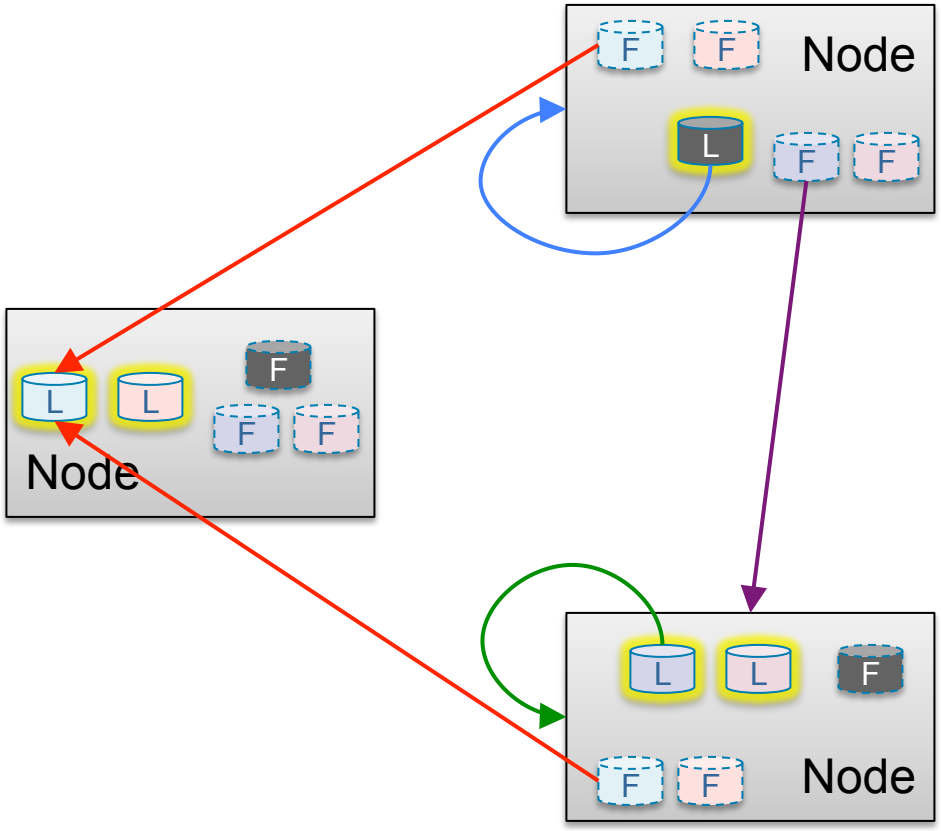
OPEN DAYLIGHT

Service/Application Model (cont'd)

- Entity Ownership Service allows for related tasks to be co-located
  - e.g., the tasks related to a given OpenFlow switch should happen where it's connected
  - also handles HA/failover, automatic election of a new entity owner
- RPCs and Notifications are directed to the entity owner
  - New cluster-aware data change listeners provide integration into the data store



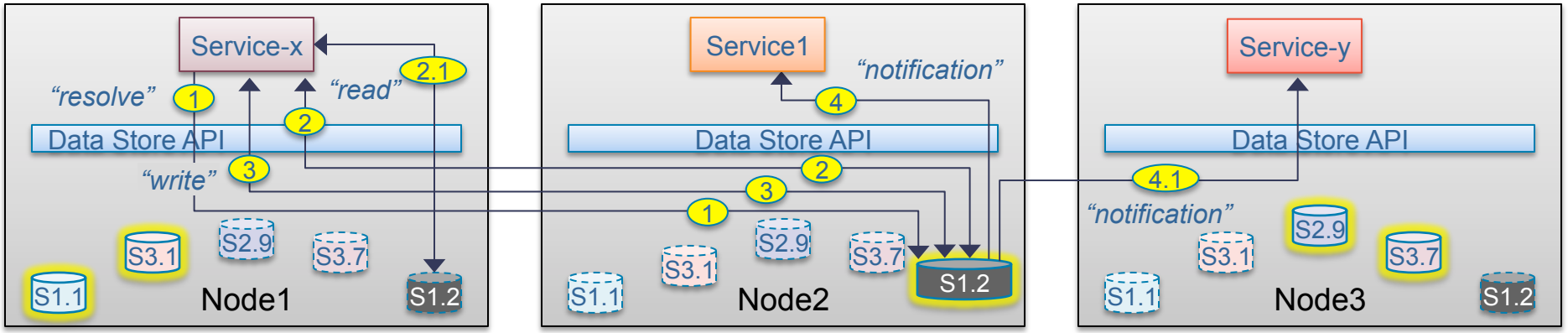
Handling RPCs and Notifications in a Cluster



- Data Change Notifications
  - Flexibly delivered to shard leader, or any subset of nodes
- YANG-modeled Notifications
  - Delivered to the node on which they were generated
  - Typically guided to entity owner
- Global RPCs
  - Delivered to node where called
- Routed RPCs
  - Delivered to the node which registered to handle them

OPEN DAYLIGHT

Service/Shard Interactions



1. Service-x "resolves" a read or write to a subtree/shard
2. Reads are sent to the leader
    1. Working on allowing for local reads
3. Writes are send to the leader to be ordered
4. Notifications changed data are sent to the shard leader
    1. and to anyone registered for remote notification

Shard Leader replica

Shard Follower replica

Major additions in Beryllium

- Entity Ownership Service
  - **`EntityOwnershipService`**
- Clustered Data Change Listeners
  - **`ClusteredDataChangeListener`** and **`ClusteredDataTreeChangeListener`**

- Singificant application/plugin adoption
  - OVSDB
  - OpenFlow
  - NETCONF
  - Neutron
  - Etc.

Work in progress

- Dynamic, multi-level sharding
  - Multi-level, e.g., OpenFlow should be able to say it's subtrees start at the "switch" node
  - Dynamic, e.g., an OpenFlow subtree should be moved if the connection moves
- Improve performance, scale, stability, etc. as always
  - Faster, but "stale", reads from local replica vs. always reading from leader
  - Pipelined transactions for better cluster write throughput
- Whatever else you're interested in helping with

Longer term things

- Helper code for building common app patterns
  - run once in the cluster and fail-over if that node goes down
  - run everywhere and handle things
- Different consistency models
  - Giving people the knob is easy
  - Dealing with the ramifications is hard
- Federated/hierarchical clustering