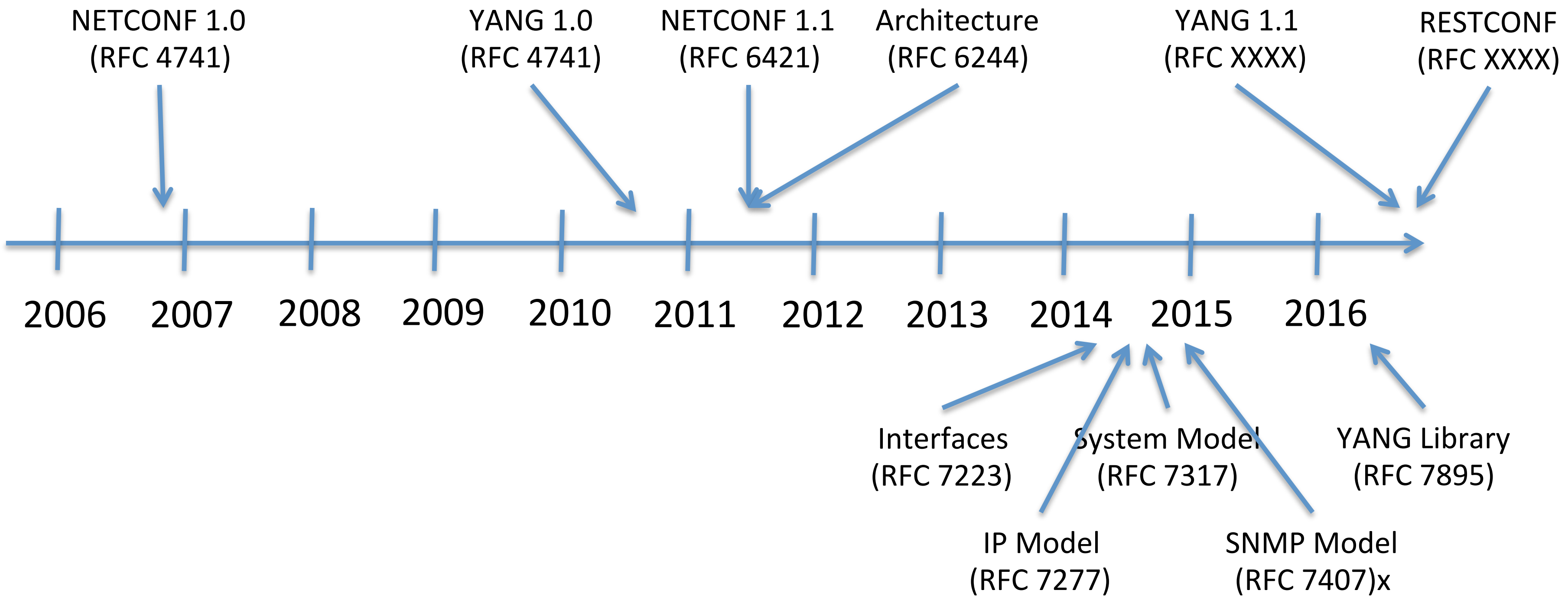
Revised Conceptual Model for YANG Datastores

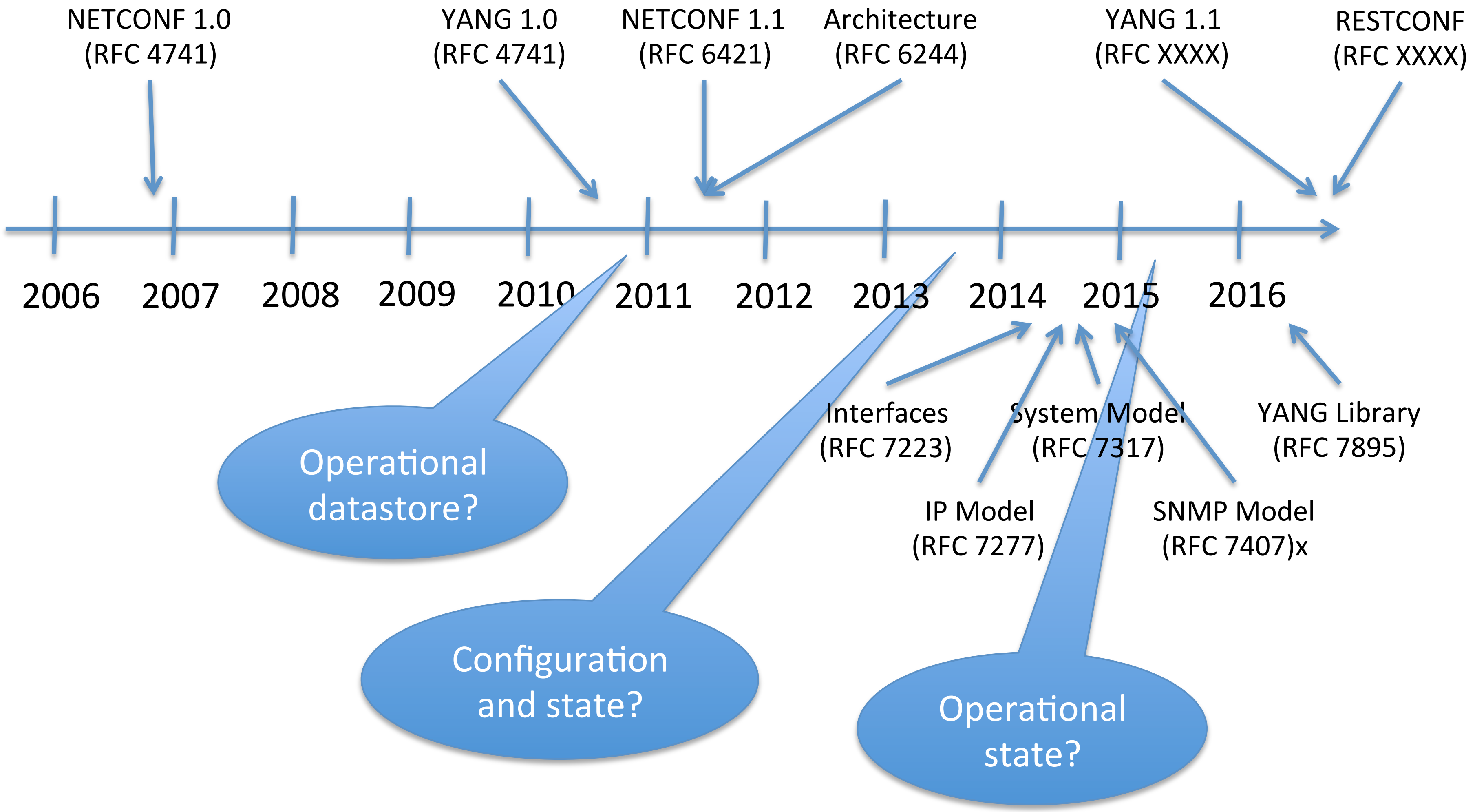
draft-schoenw-netmod-revised-datastores-01

Jürgen Schönwälder
Jacobs University Bremen

NETCONF/RESTCONF/YANG Timeline



NETCONF/RESTCONF/YANG Timeline



Interfaces Data Model

- Some design challenges (2013/2014):
  - Configuration of interfaces currently not present
  - Interfaces created dynamically by the system
  - Auto-configuration mechanisms (mtu = auto)
- Resulted in the /foo and /foo-state approach
  - Configuration goes into /interfaces
  - Operational state goes into /interfaces-state

I2RS and OPSTATE Discussions

- I2RS discussions (2014-2016)
  - Ephemeral configuration datastore
  - Pane of glass model

- OPSTATE discussions (2015-2016)
  - Intended configuration
  - Applied configuration
  - Single tree vs. multiple trees vs. multiple datastores
  - Separation of state and statistics

Complicated Discussions…

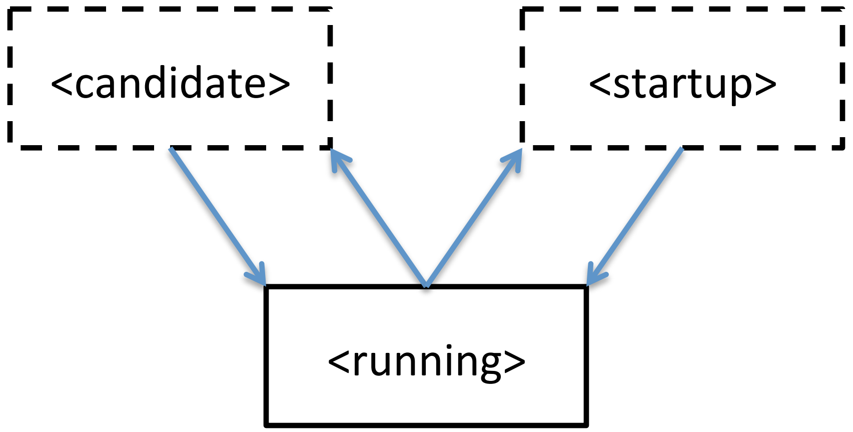
- Terminology is often confusing
  - People use the same words but with different semantics
  - Even carefully worded requirements can be interpreted differently
  - Careful wording sometimes leads to statements that are true in multiple world models
  - Example: *What is 'configuration'?*

- Unknown protocol requirements
  - YANG models are used to drive proprietary protocols that have only partially known properties
  - Example: *Our protocol does not have datastores.*

Solution Design Objectives

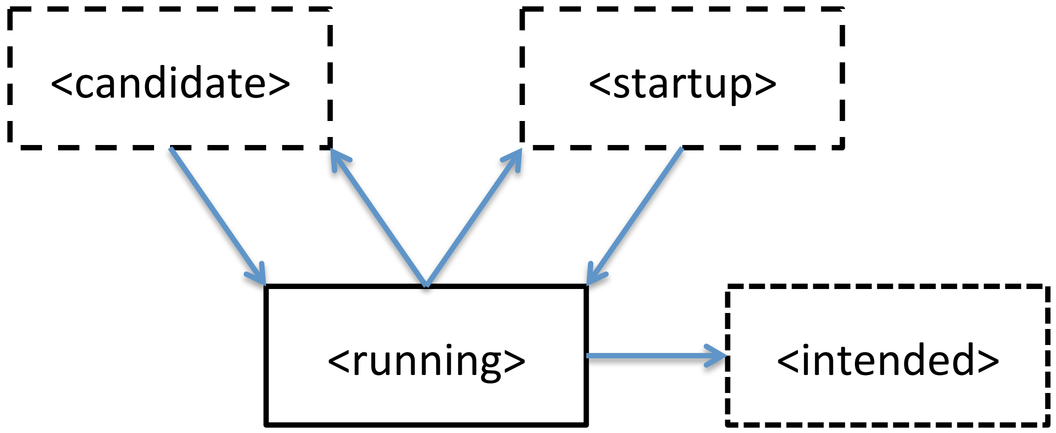
- Configuration **datastores** are a fundamental **architectural concept** of NETCONF and YANG
- We need to accommodate requirements in a way that is largely an **extension** of what we have today and thus is **backwards compatible**
- We need a **conceptual model** that enables **modular implementations** (not every implementation will support everything)
- An **architectural model** is needed to ensure multiple protocols **expose data consistently**

Original NETCONF Datastores



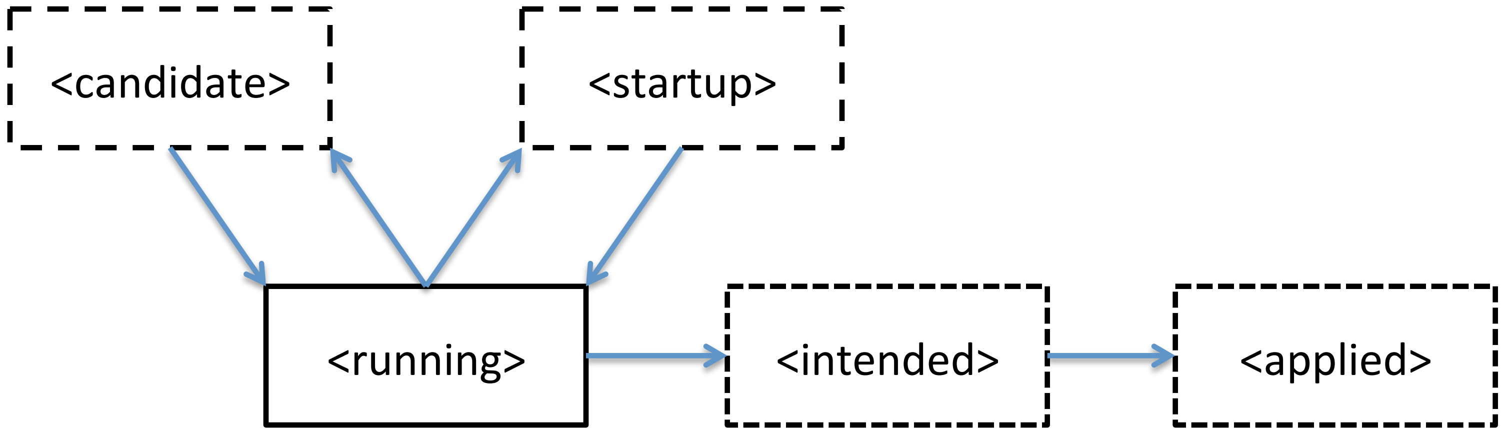
- Mandatory <running> holds optionally editable **eventually persistent configuration** (config true)
- Optional <candidate> holds editable and typically ephemeral **configuration** (config true)
- Optional <startup> holds copyable **persistent configuration** (config true)
- Configuration is used in a narrow sense; it does **not include soft state** obtained from the interaction with other systems

Conceptual <intended> Datastore



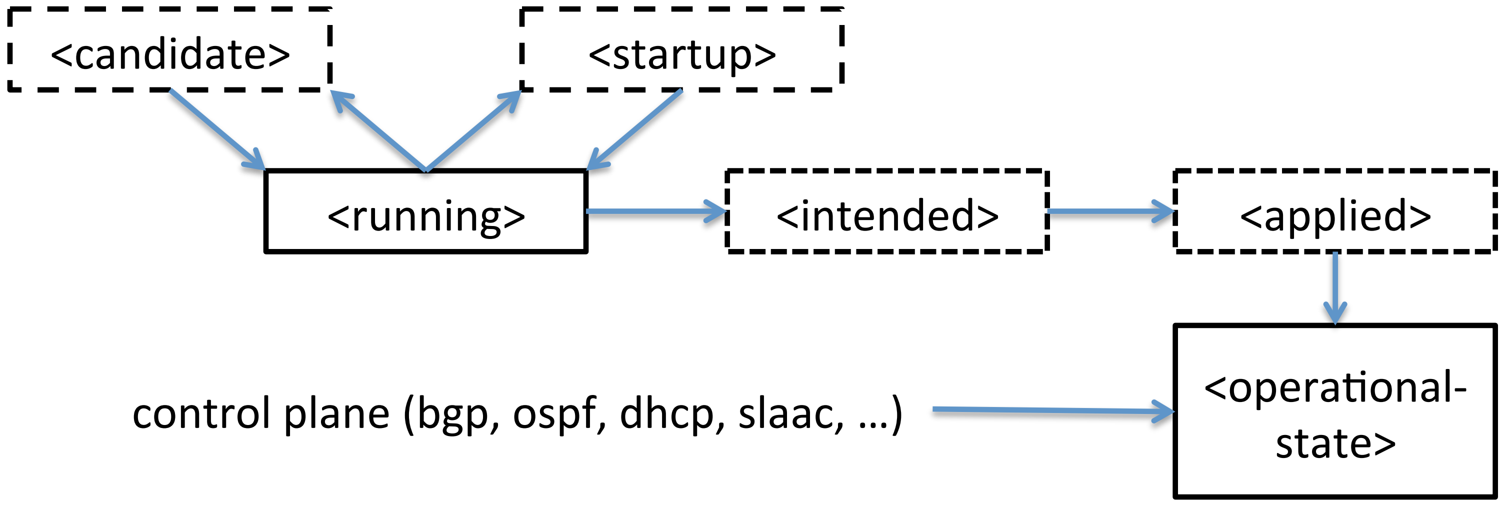
- The <intended> **conceptual datastore** usually holds the same data as the <running> datastore
- Exceptions are situations where configuration data held in <running> is marked to be inactive ("commented out")
- The content of the <intended> datastore is subject to YANG validation rules

Conceptual <applied> Datastore



- The <applied> **conceptual datastore** usually holds the same data of the <intended> datastore
- Differences may exist due to **missing resources** (e.g., a linecard) or if if a configuration change **takes time** to be applied
- Hence, differences between <intended> and <applied> may be short- or long-lived (<applied> may never converge to <intended>)

Operational-state Datastore



- The <operational-state> datastore holds all operational state
- It is read-only and ephemeral; it exposes the ground truth about what the system is doing
- It may expose config true objects as read-only objects
- The operational state includes information obtained from control plane mechanisms as well as statistics generated by the system
- Metadata may expose the origin of data in <operational-state>

Impact on Data Models and Protocols

- A single tree can be used to model configuration and state of a resource (no need for /foo and /foo-state anymore)
- The proposed conceptual datastores expose the differences between <running>, <intended>, <applied>, and <operational-state>
- Not all implementations may expose these conceptual datastores
- It is desirable to provide protocol operations that make it efficient to retrieve the differences between conceptual datastores
- It is desirable to provide notifications that can signal changes of differences
- The <get> operation is considered harmful (since it assumes configuration datastore content and operational state can be represented as a single tree) and it should be deprecated
- For additional details, see the discussion in the Internet-Draft