Reuse of requirements and more

An introduction to OSLC Configuration Management

Nick Crossley
Senior Architect

OMG OSLC Summit
La Jolla, December 2015

Abstract

Can you imagine a software team working without a configuration management system? Could your team do any collaboration without versions, streams and baselines? Could you manage reuse of software libraries without SCM? SCM is considered a basic necessity of development. But why wouldn't you want the same capabilities both within each of your tools, and even across all of them?

Starting with the concepts of linked data and Open Services for Lifecycle Collaboration (OSLC), this session explains version and configuration management across the lifecycle.

Speaker

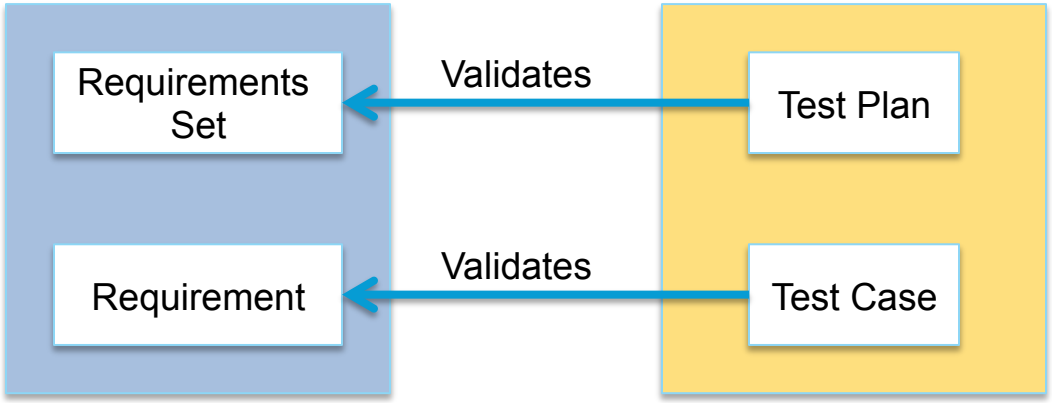
Nick Crossley is the Technical Architect for the IBM Middleware solution for Product Line Engineering, including version and variant management, configuration management, and cross-domain baselining.

Nick is the chair of the OASIS OSLC CCM TC, and also participates in the Core workgroup. Nick has over 40 years of experience with software tools and development.

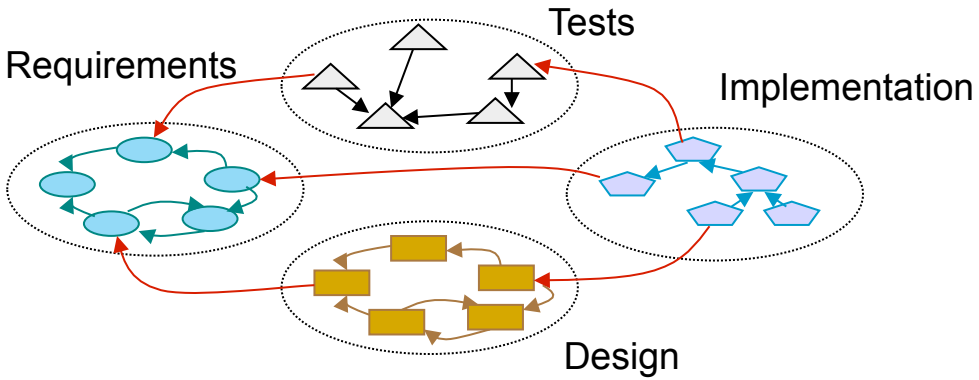
Our flight plan

1. Basic building blocks
2. Using streams and baselines
3. Using global configurations

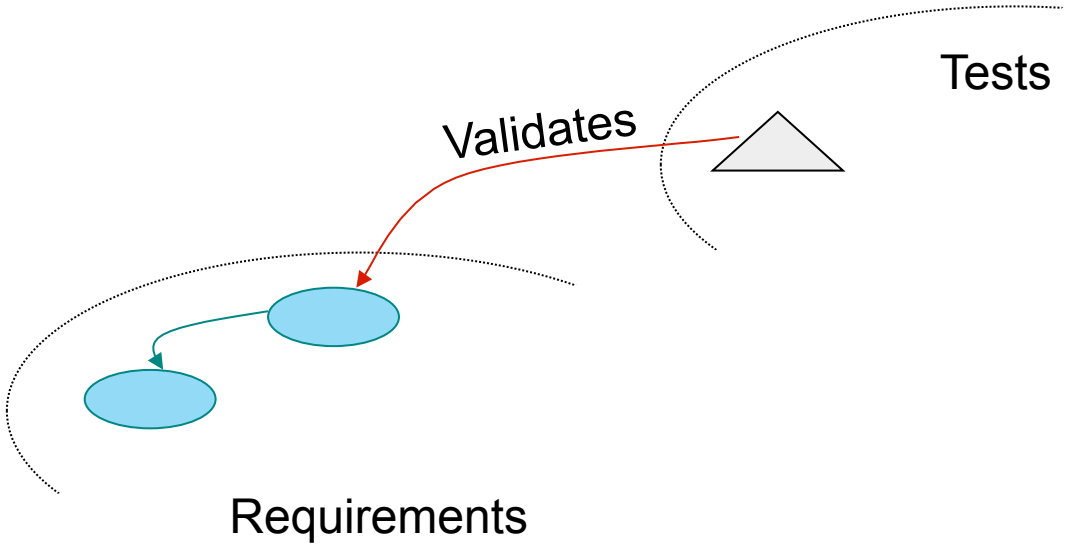
Linked lifecycle data



Linked lifecycle data



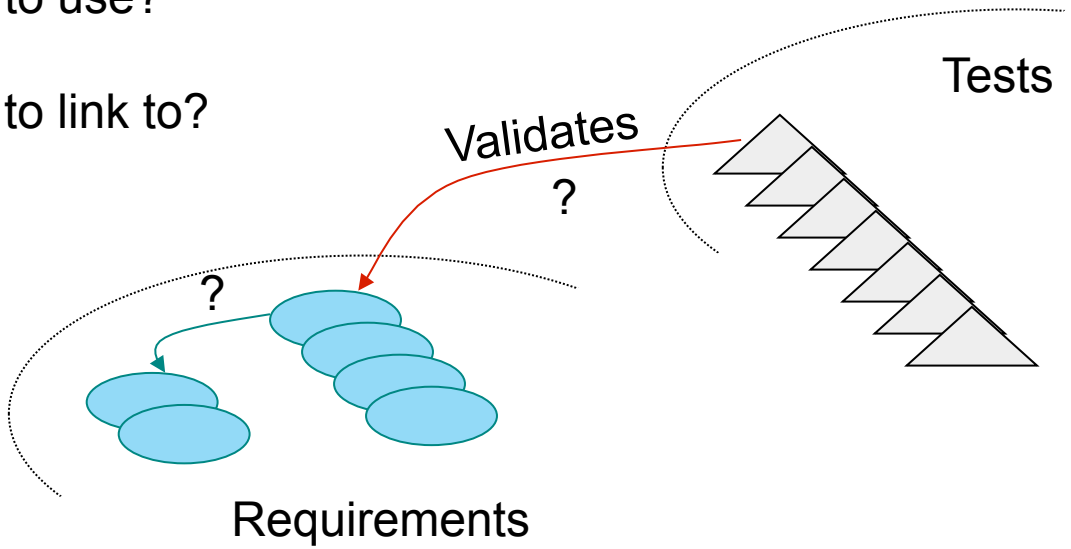
Zooming in …



Artifacts may have many versions

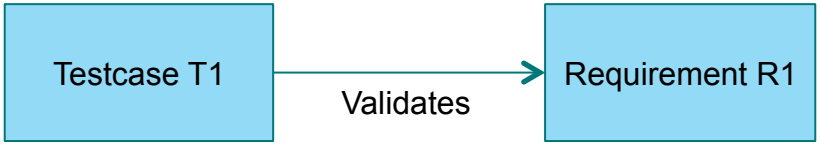
Which version to use?

Which version to link to?



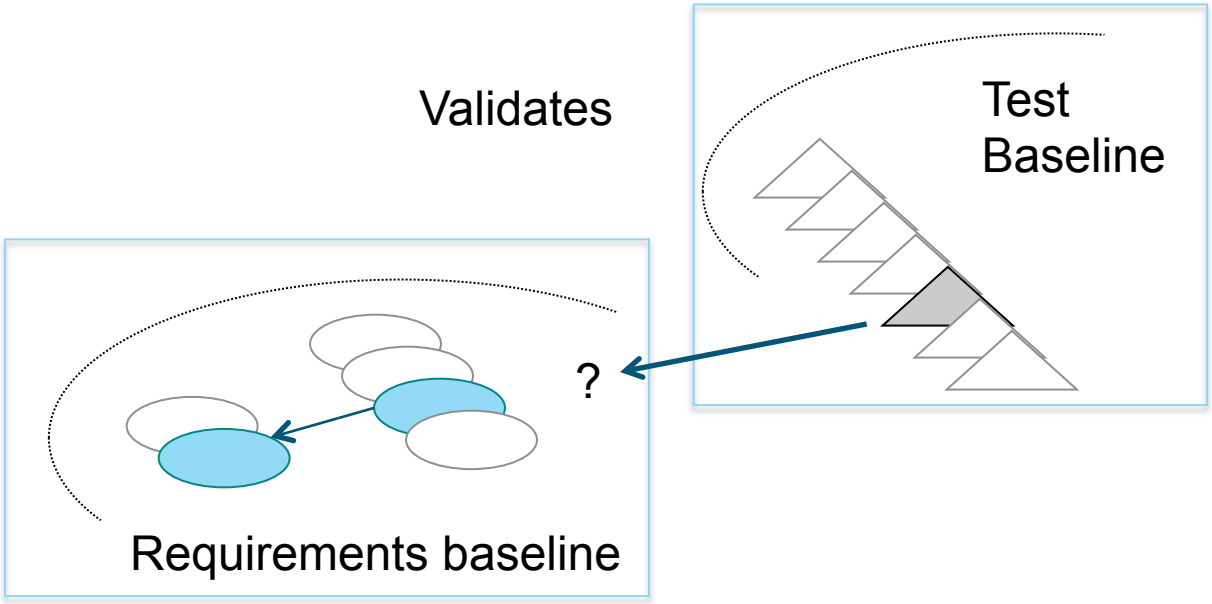
Linking Model – basics

- A link has a name/type, a starting artifact and a target artifact; this implies a link has a specific direction
- Links are a visible, queriable, reportable aspect of versions of artifacts, and changes to links are an important part of the history of an artifact
- OSLC linking model allows a lot of flexibility in the storage of links
- For simplicity, IBM decided to go with a model where links are stored (or *behave as if* they are stored) in one of the two artifacts being linked – normally the 'from' end of the link
  - With RDF, the physical location of the storage really does not matter much; you can put a triple anywhere, and it can make a statement about any subject resource
  - However, most outgoing links are an important part of the semantic state of an artifact, and so should be included in the version history, and in things like digital signatures of the artifact.

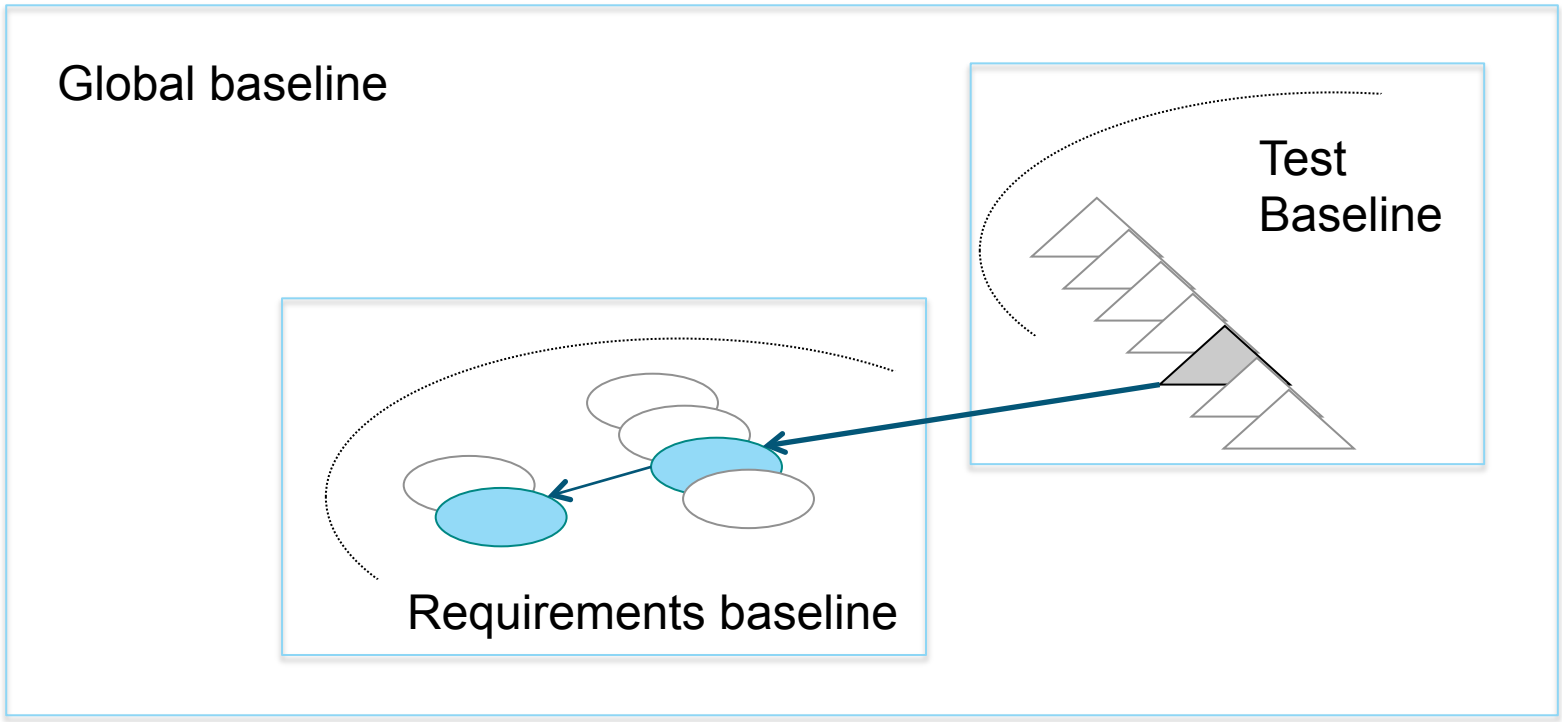


9

Each tool uses streams and baselines to select the right artifact versions

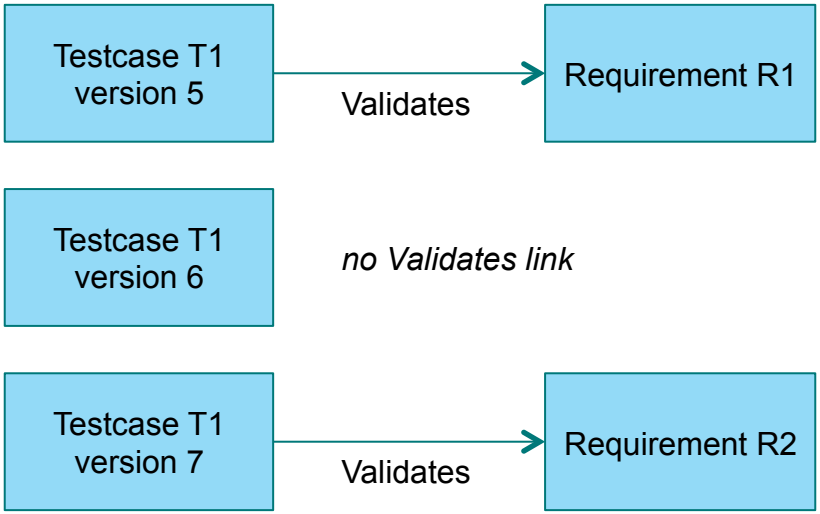


A global configuration provides context to resolve links to the right artifact versions

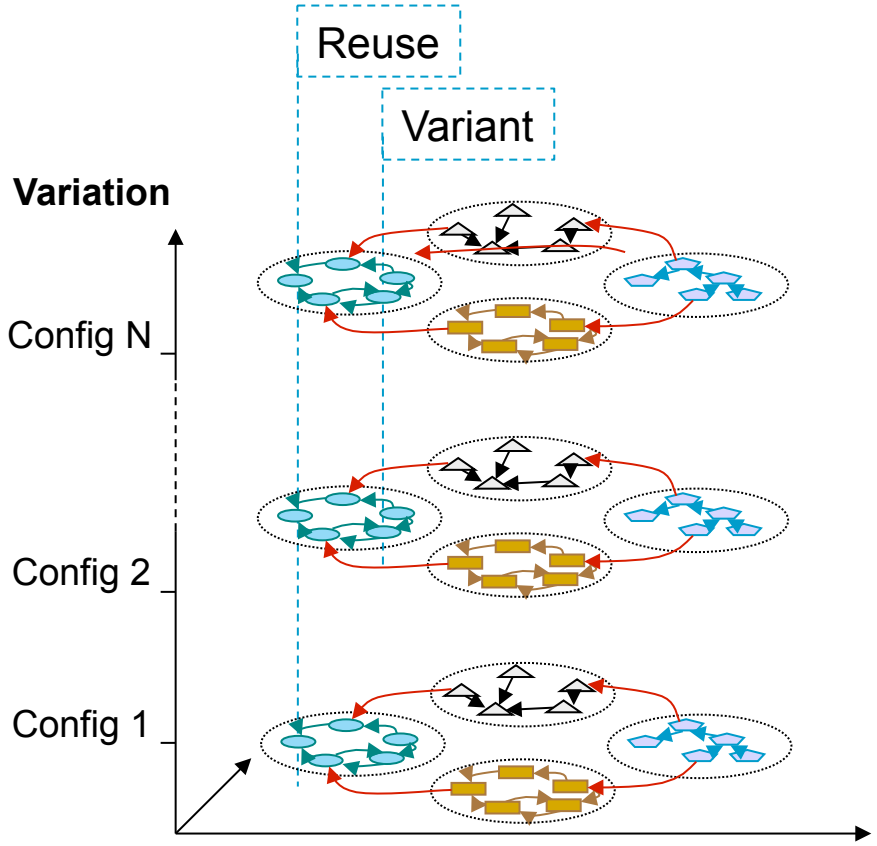


Linking Model – configuration determines version of target

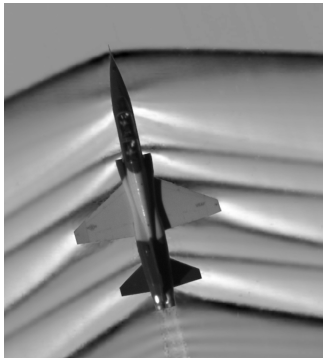
- For linking between versioned resources in a configuration-aware world, we did not want the target version of a link predetermined by the version of the source – we wanted the configuration used to determine the right version of the target
  - avoids the need to update the source every time a minor change is made to the target
  - allows a single source to link to multiple variants of the target, depending on configuration context
- But links are an important part of the state (version) of the source, and may vary between versions of the same artifact



Work in the context of a global stream or baseline

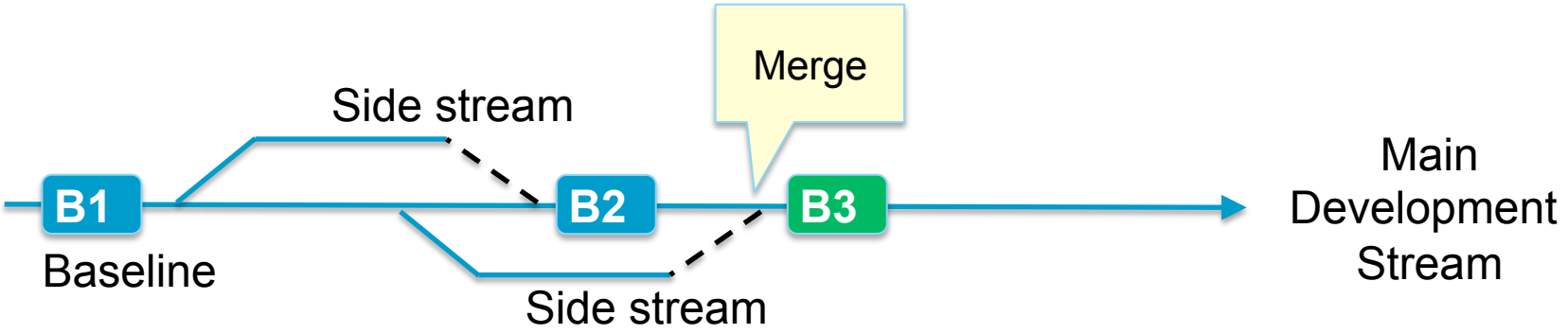


Our flight plan



1. Basic building blocks
2. Using streams and baselines
3. Using global configurations

Streams and baselines



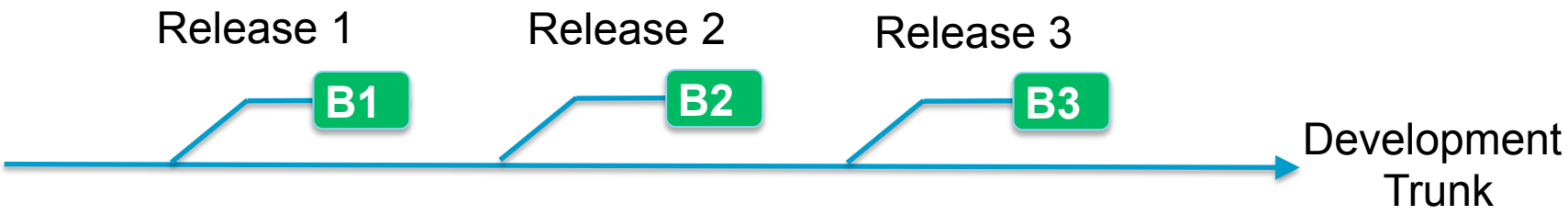
- Streams are where work is done and changes are made
- Baselines are immutable (read-only) records of the state of a stream at some point
- Support various branching strategies
  - Both ***within*** and ***across*** the lifecycle!

Streams and baselines

Requirements …
Requirements and tests …
Requirements, designs, tests and code …

Stabilization streams

Requirements …
Requirements and tests …
Requirements, designs, tests and code …

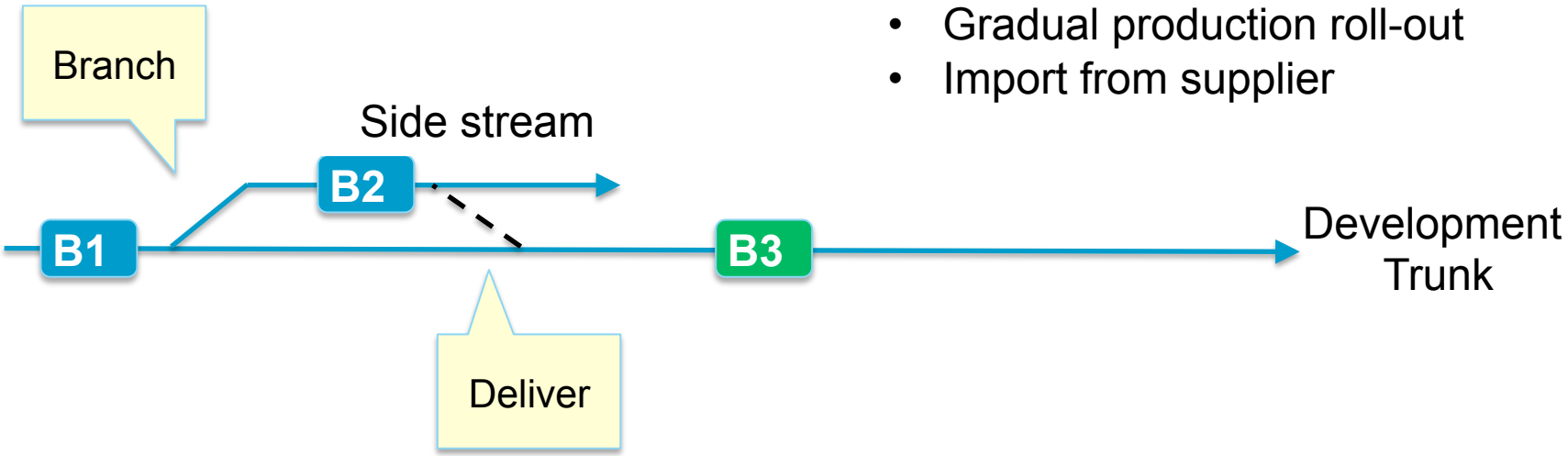


Branching and delivering – using side streams

Requirements …
Requirements and tests …
Requirements, designs, tests and code …

Some reasons for side stream

- Spike experiment
- A/B Test
- Gradual production roll-out
- Import from supplier

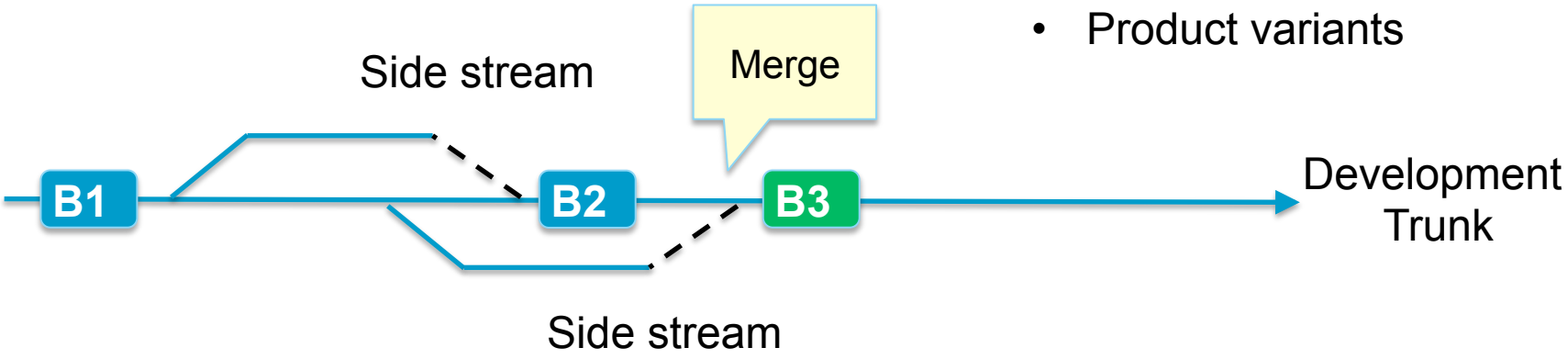


Parallel development and merging

Requirements …
Requirements and tests …
Requirements, designs, tests and code …

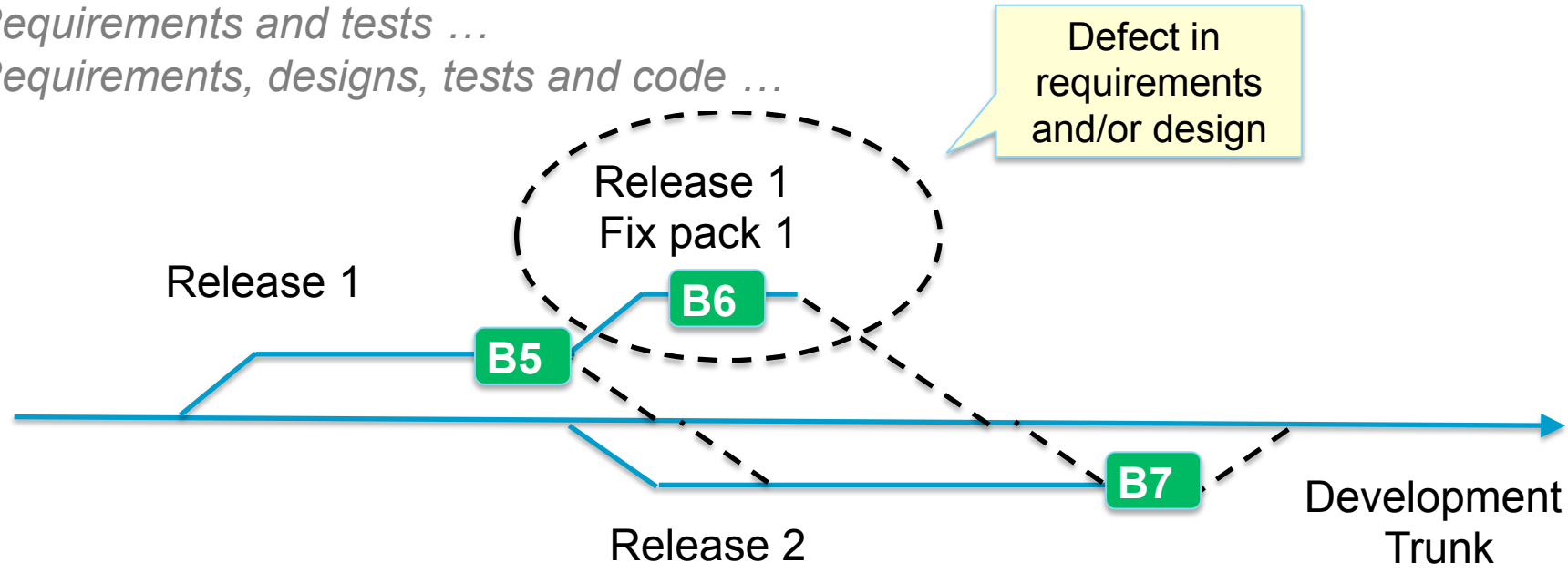
Some reasons for parallel development

- Shared artifacts
- Overlapping releases
- Product variants



Fix pack

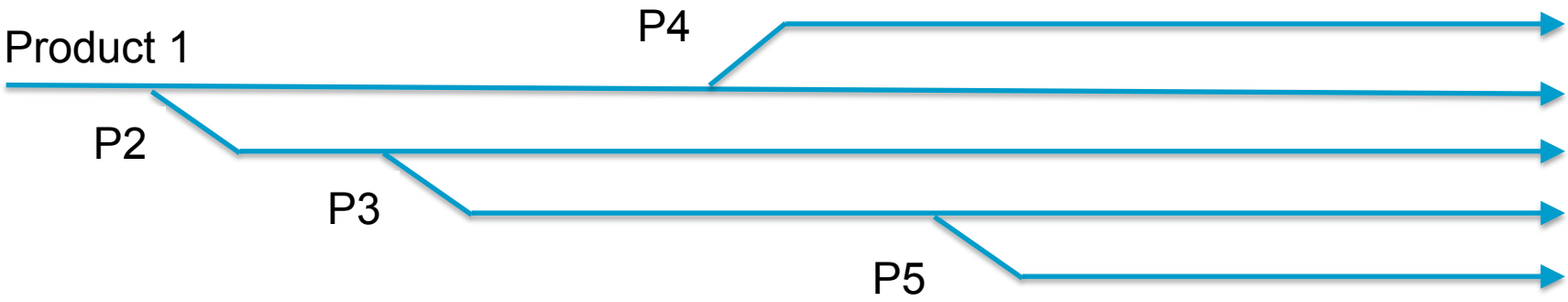
Requirements …
Requirements and tests …
Requirements, designs, tests and code …



Reuse: Branch from closest

Requirements …
Requirements and tests …
Requirements, designs, tests and code …

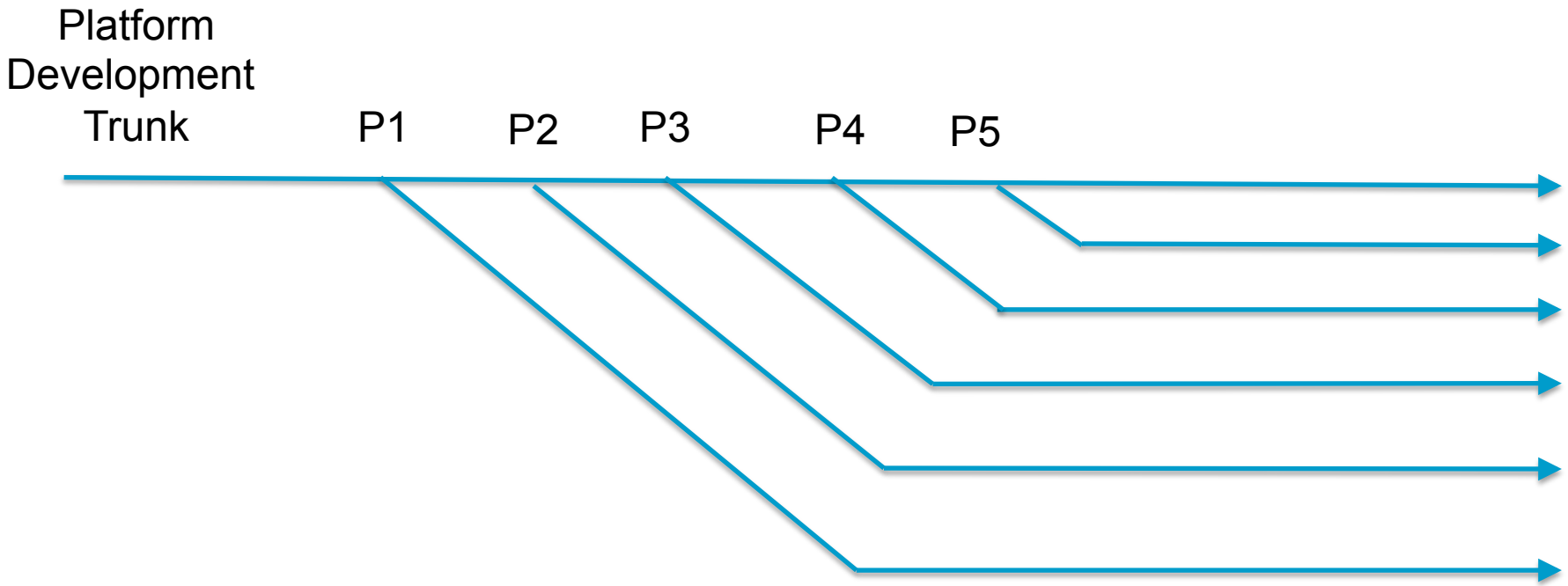
Each branch is a new product



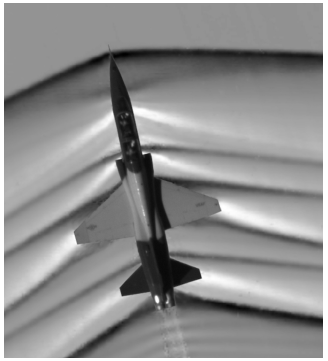
Reuse: Branch from base

Requirements …
Requirements and tests …
Requirements, designs, tests and code …

Derive new product from base stream



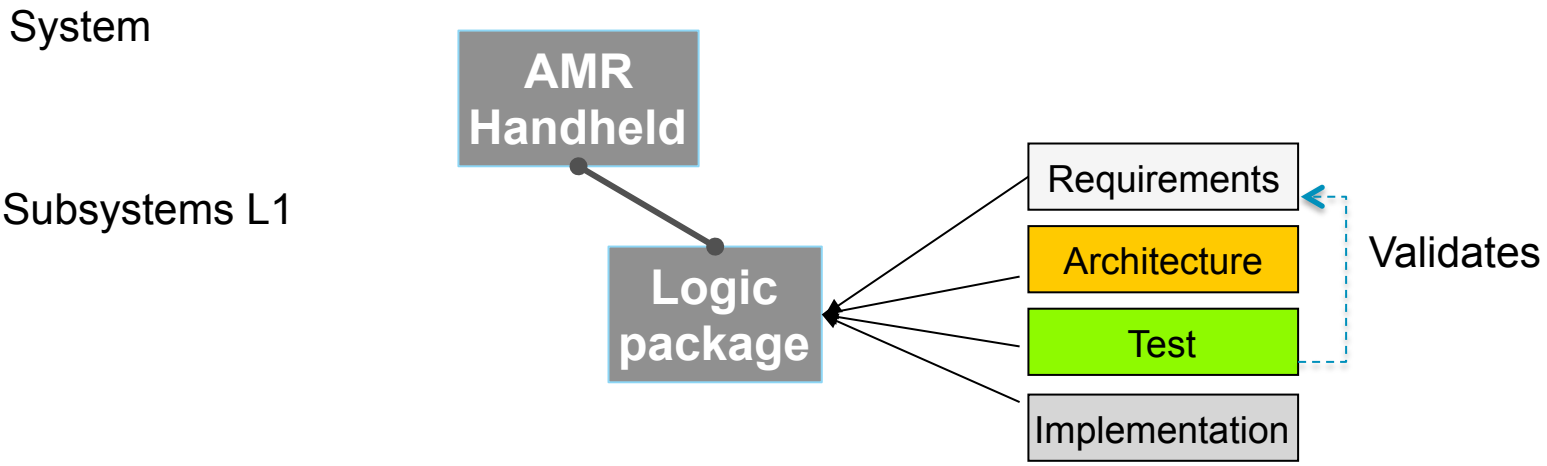
Our flight plan



1. Basic building blocks
2. Using streams and baselines
3. Using global configurations

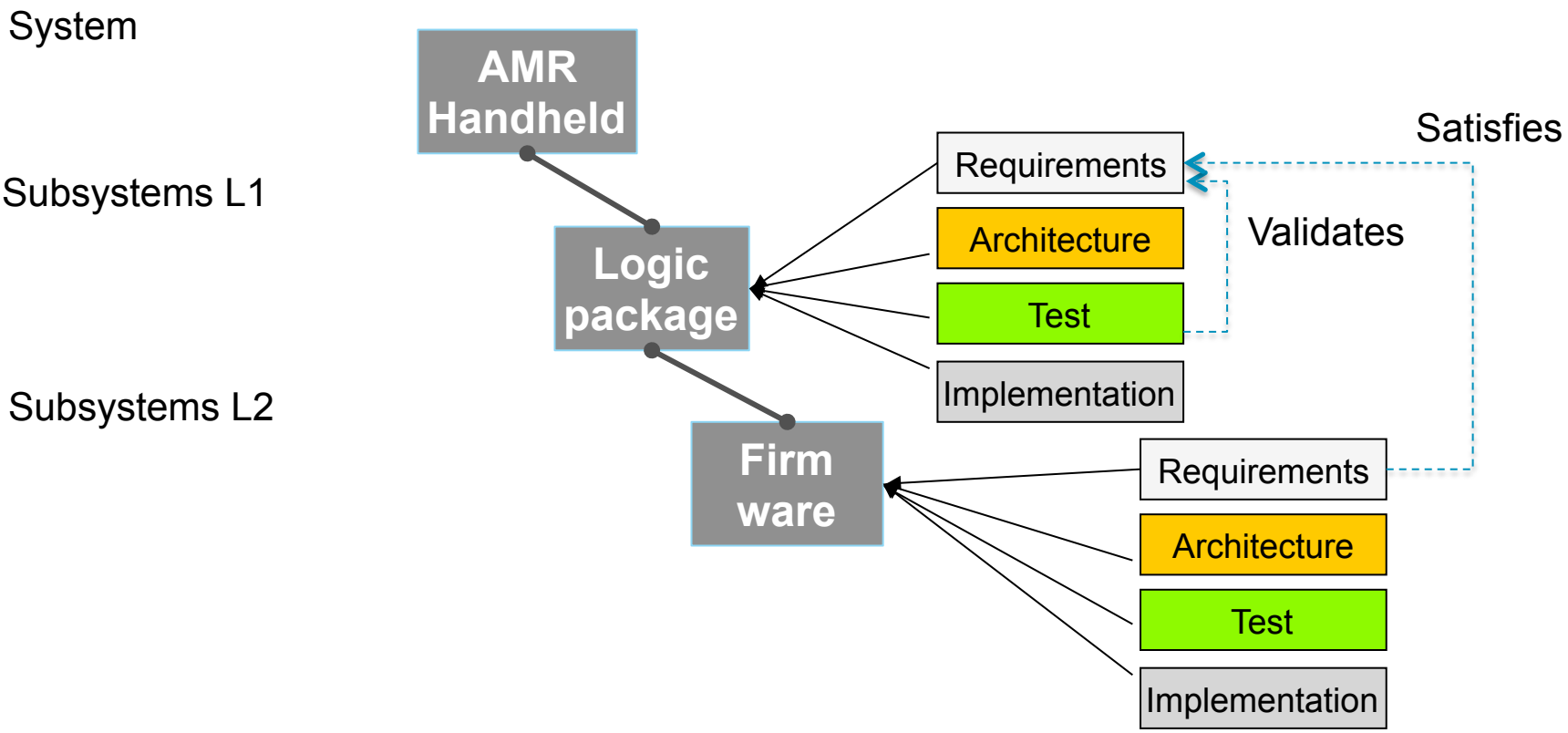
Complex products in a configuration hierarchy

Global configurations provide context for links



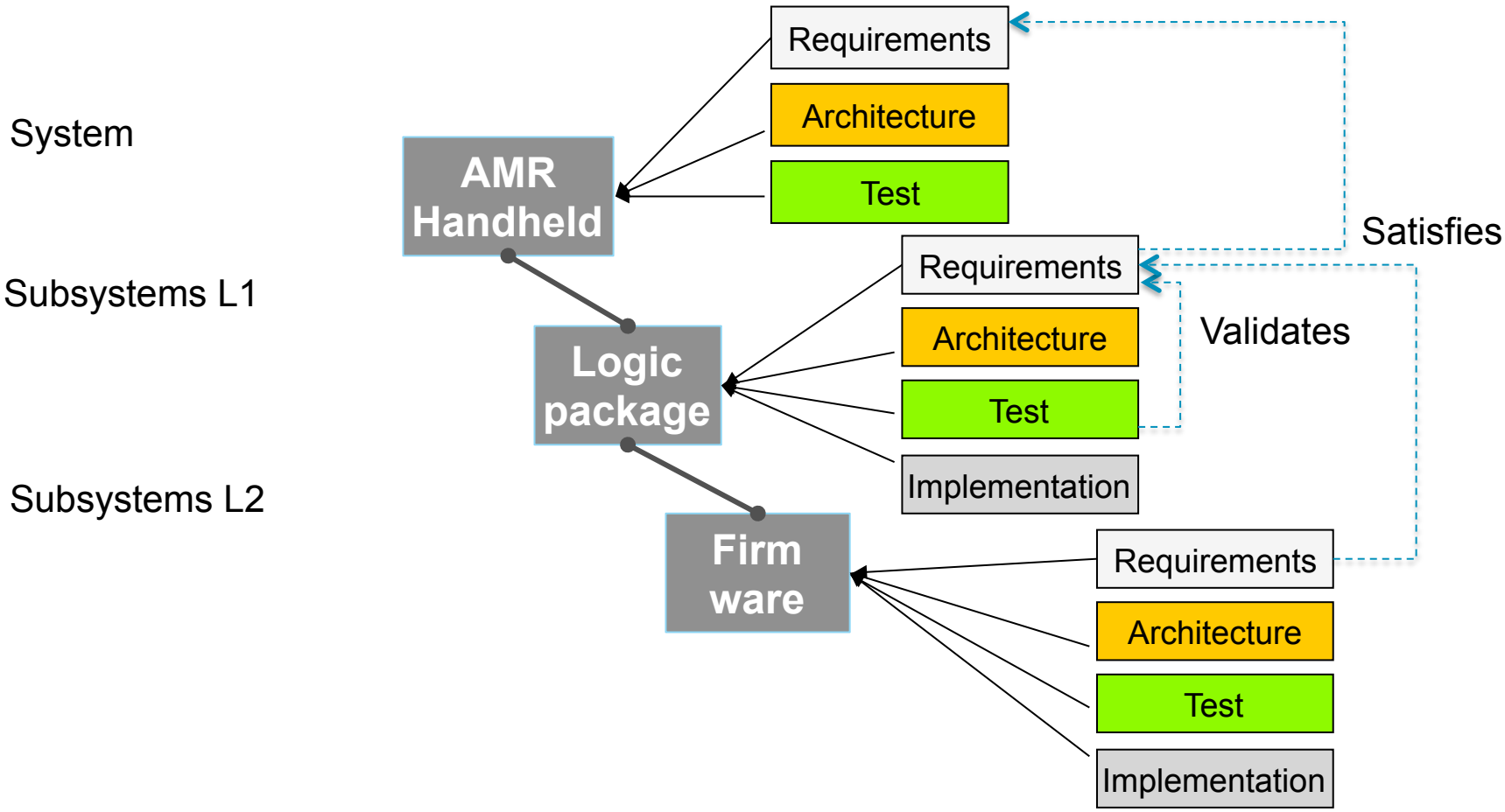
Complex products in a configuration hierarchy

Global configurations provide context for links



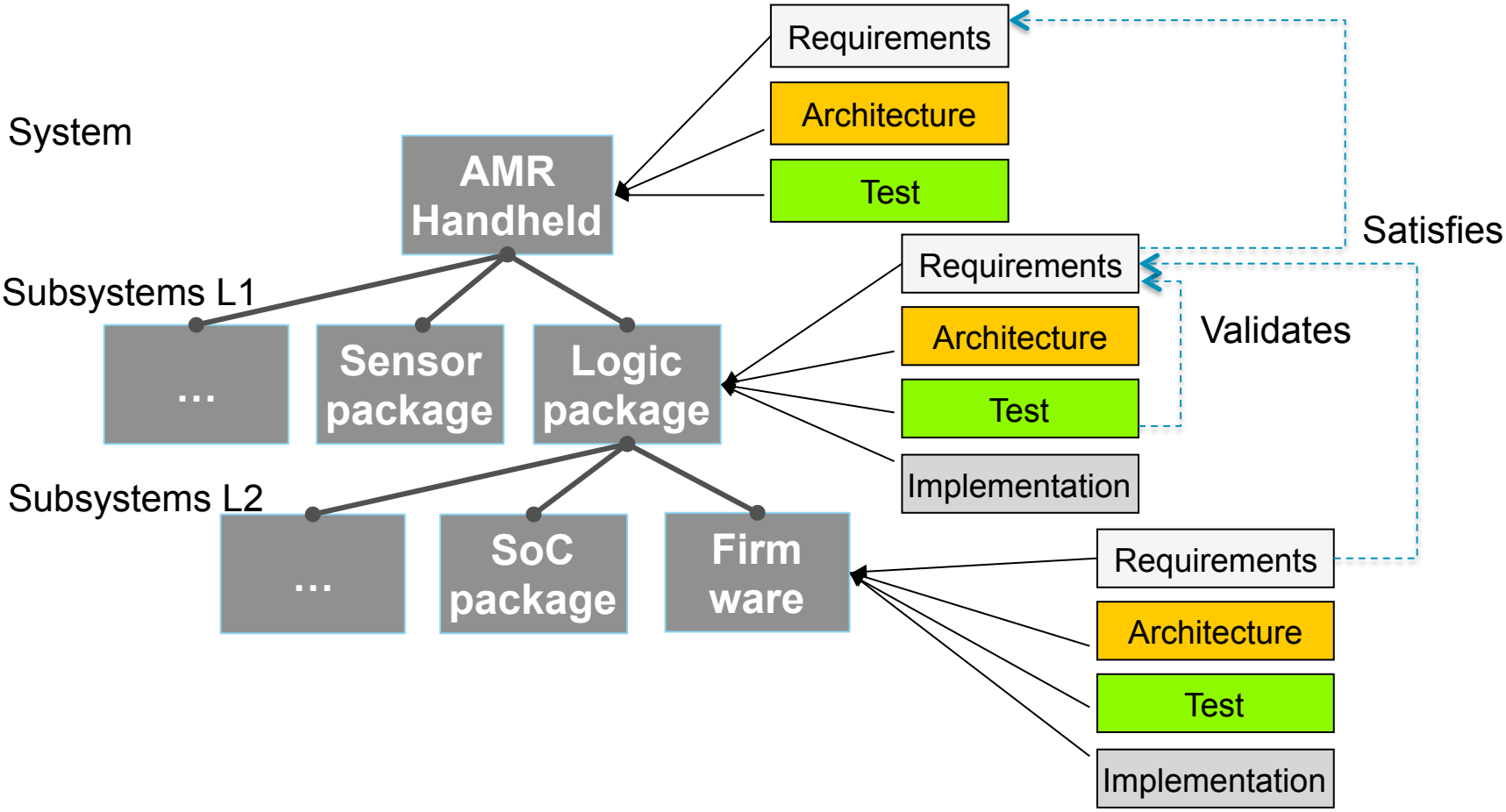
Complex products in a configuration hierarchy

Global configurations provide context for links



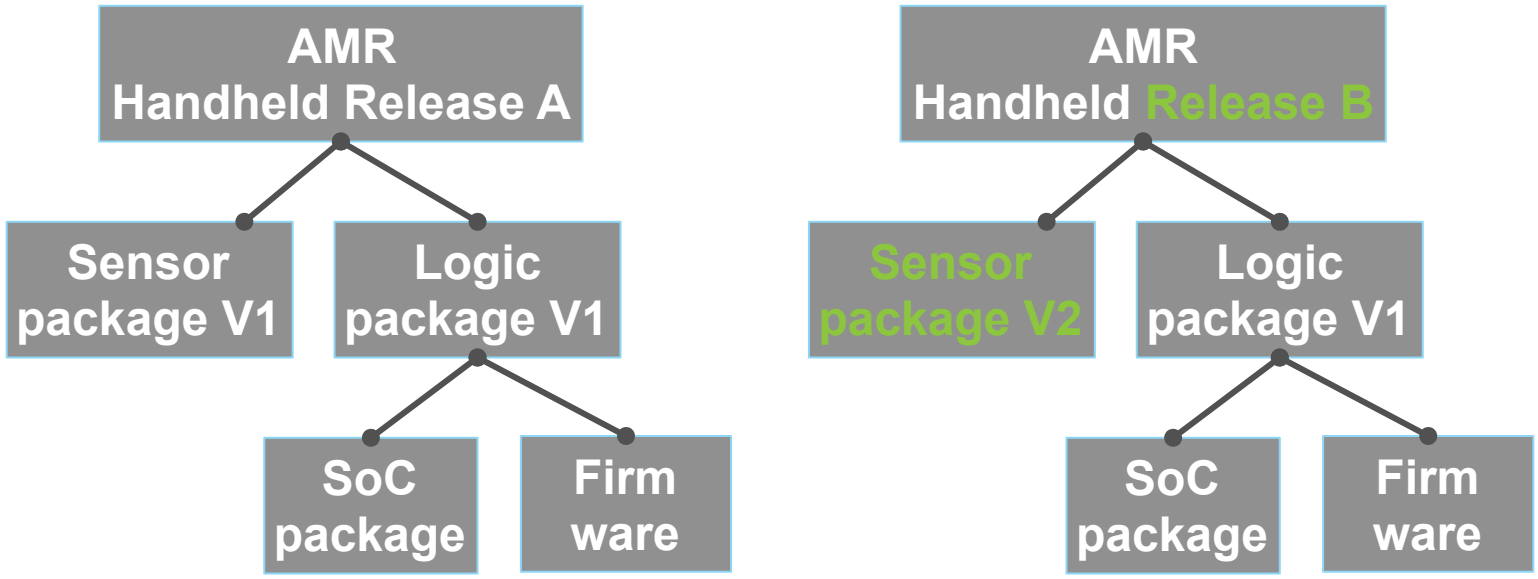
Complex products in a configuration hierarchy

Global configurations provide context for links



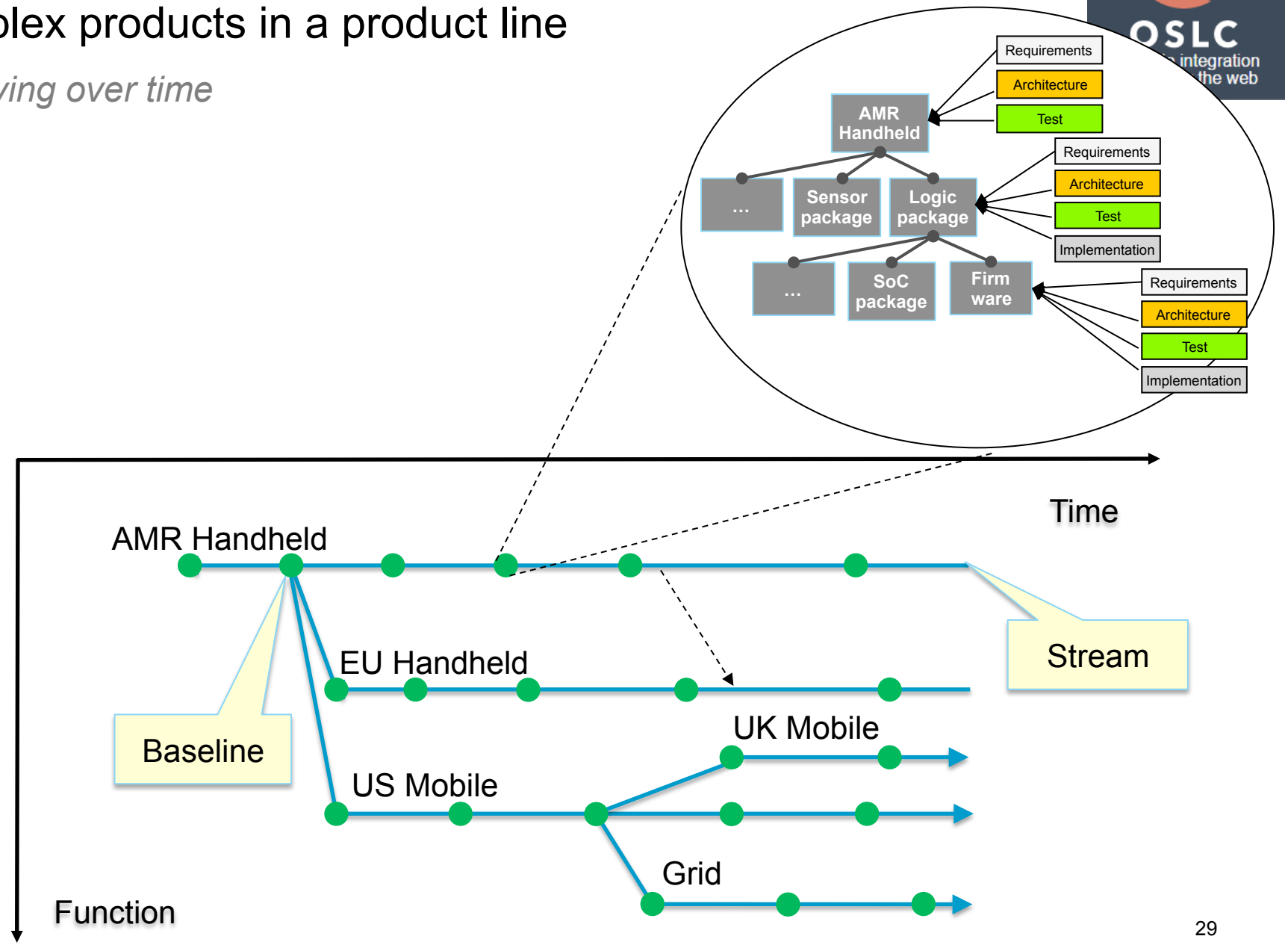
Complex products and reuse (over time, in multiple variants)

- Reusable components and sub-systems
  - New Sensor Package in Release B (V1 → V2)
  - Reused Logic Package (V1 → V1)



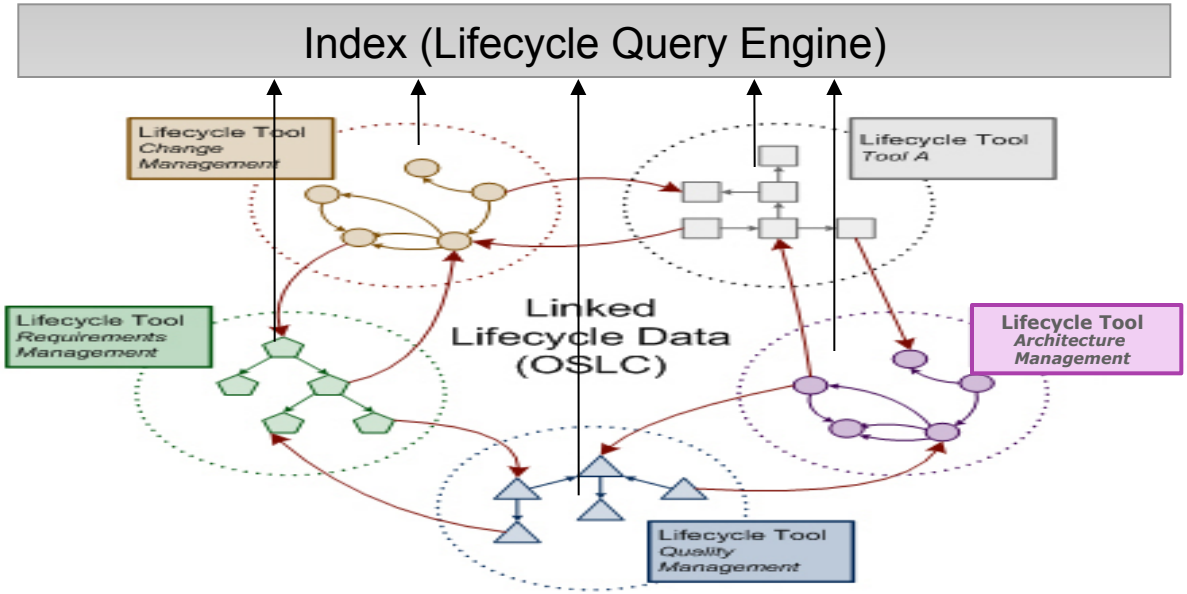
Complex products in a product line

Evolving over time



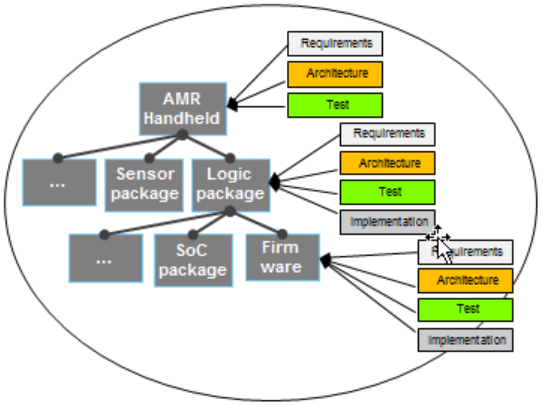
Index of Linked Lifecycle Data

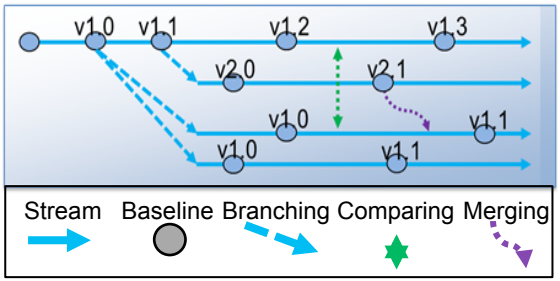
- An index of Linked Data is created from domain tools that allows for cross-domain Lifecycle Queries – including links between resources



Gain engineering efficiencies with OSLC Configuration Management

- Stream-based development – more than versioned documents
  - Parallel, insulated work in streams and baselines
  - Compare / Merge / Branch
- Reusable components – not just software source code
  - Across the development lifecycle
  - Including traceability relationships
- Reuse in hierarchical configurations
  - Defining systems and subsystems
  - Enabling reuse at any level
- An open, federated approach
  - Multiple tools & vendors participating in configurations





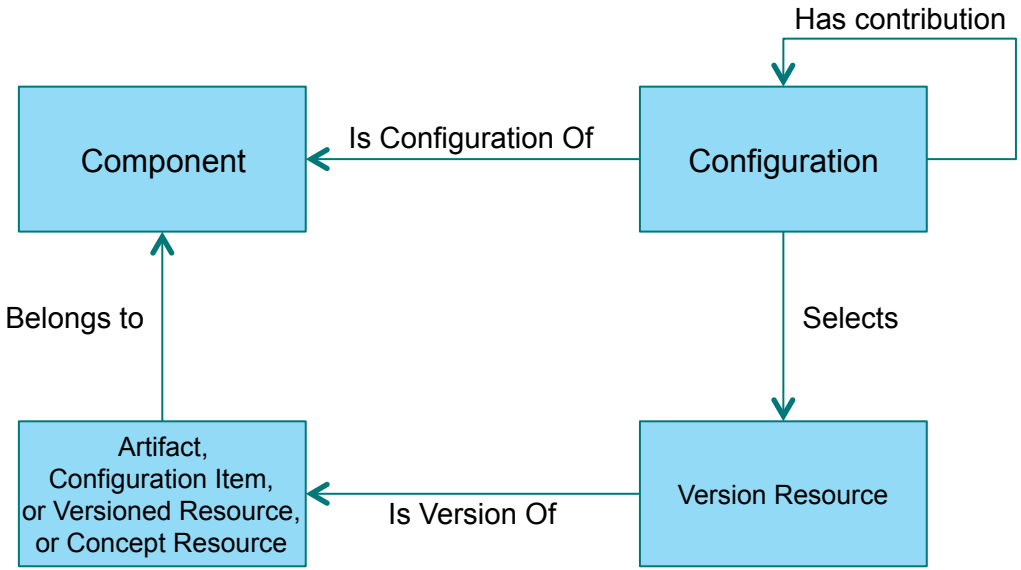
New capabilities automate what previously had to be done manually (and with copies)

OSLC Configuration Management

- Draft standard at:
  https://tools.oasis-open.org/version-control/browse/wsvn/oslc-ccm/trunk/specs/config-mgt/oslc-config-mgt.html
- See also some videos of IBM's implementation at:
  https://www.youtube.com/playlist?list=PLZGO0qYNSD4V6xyq6nmZgD8jg7Y9iDpGM
- To participate, join OASIS at https://www.oasis-open.org/join,
  and contact ncrossley@us.ibm.com

Configuration Information Model

- This picture is a simplification to show the overall idea behind OSLC Configuration Management – there is indirection in some of these links, and some additional resources for specific purposes; the exact relationship names may be found in the spec
- Artifacts that are versioned belong to some component
- Configurations are configurations of some component
- Configurations select a version of each of a set of versioned resources from the component
- Configurations form a contribution hierarchy; global configurations collect contributions from multiple tools/domains



OSLC
Lifecycle integration
inspired by the web