ORACLE

ORACLE

Siebel CRM Workspaces

Best Practices for Developers

Brian Kelly
Director, Product Management
Siebel CRM

October, 2020

Safe harbor statement

Presenters

Brian Kelly

Director,
Product Management

ORACLE
Siebel

Manasi Bimalkhedkar

Senior Manager,
Software Development

ORACLE
Siebel

Periasamy Rangasamy

Senior Manager,
Software Development

ORACLE
Siebel

Ravi Doraiswamy

Senior Director,
Software Development

ORACLE
Siebel

Outline

- Overview of Workspaces
- Design and Runtime Repositories
- How Does It All Work?
- Best Practices
- Advanced Scenario (Example)

Workspaces Overview

GOALS

- Gain a basic understanding of workspaces
- Learn a lot of terminology

Workspaces Overview

What does a Workspace Do?

- Provides a sandbox in which to develop and test a feature without affecting other users
- Allows for parallel development—many developers can work on an object at once
- Provides a versioning mechanism to track changes
- Handles merging of changes when multiple developers modify an object
- Supplies a flexible, hierarchical framework for building features and releases
- Through related features, such as the Migration Application, provides a mechanism to incrementally update downstream environments

Workspaces Overview

Types of Workspaces

- Developer Workspaces
  - Owned by a single developer
  - Allows developer to build and unit test repository changes
  - Workspace can be “Inspected” by any user to validate
    - Peers
    - Team Lead
    - QA
- Integration Workspaces
  - Parent for one or many Developer Workspaces
  - Multiple layers of integration workspaces are possible (e.g., multiple Features under one Release)
  - Allow for integration testing of multiple developer changes in a sandboxed environment
    - New Feature
    - Release
- MAIN
  - Special case of Integration Branch at root of hierarchy
  - Typically mirrors the current Production configuration

Workspaces Overview

Workspace Inheritance

- Workspaces inherit the object definitions from their ancestor workspaces
  - MAIN—the root—has all object definitions
  - Integration branches have only the changed records (as delivered from developer workspaces)*
  - Developer workspaces have only the records changed in that specific workspace
- When a workspace is delivered to its parent…
  - Original record is maintained as well
  - New record will be created with an increment to the version number

** There are exceptions for workspace-enabled seed data; these will be discussed separately*

Workspaces Overview

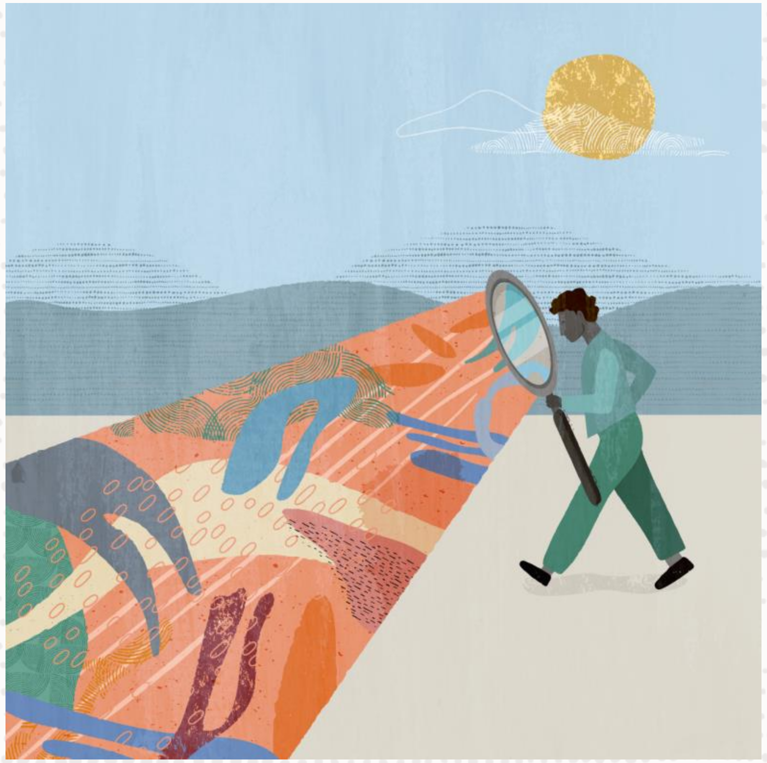
Inspection of Workspaces

- Allows review of configuration changes before delivery
  - Launch client object manager (e.g., Call Center)
  - Open Workspace Dashboard in the client
  - Select the workspace containing the changed objects
  - Navigate to wherever the changes are and test them
- In the background…
  - Application is reading the definition from inspected workspace “on the fly”
  - Merges together content from MAIN with changes in all child, grandchild, etc. workspaces through the one being inspected
  - This hierarchical merging is what allows us to keep only the delta changes in workspaces

Workspaces Overview

Summary

- Workspaces are a way to sandbox configuration changes and allow for parallel development
- There are different types of workspaces, including MAIN, Integration, and Developer
- Workspaces are hierarchical (MAIN→Integration[→Integration]→Developer
- Each workspace contains only the incremental changes from its parent workspace
- Every object has a logical identifier (*WS_SRC_ID*) that links it to all other instances of itself across all versions and workspaces
  - Applies to all objects in the hierarchy—e.g., *S_BUSCOMP*, *S_FIELD*, *S_MVLINK*, etc.
  - For example, change to a Field-level validation rule does not actually effect the BusComp itself
- Every instance of an object can be uniquely identified by *WS_SRC_ID*, *WS_ID*, and *WS_OBJ_VER*
- At runtime, a given workspace can be inspected for test purposes.
- During inspection, The runtime environment will show the net of any changes made to objects from the inspected environment up through MAIN



The Design & Runtime Repositories

GOALS

- Understand the Design Repository
- Understand is the Runtime Repository
- Comprehend the difference

Repository Types
Basic Definitions

- Design Repository (DR)
  - Traditional metadata for configuration Siebel CRM applications
  - Edited through Siebel Tools / Web Tools
  - Contains various object types—Applets, BusComps, Tables, etc.—including child object types
  - Human readable format
- Runtime Repository (RR)
  - Compiled version of each top-level object type
    - E.g., RR definition of an applet contains header (applet) information plus child (web template, control, list column, etc.) in one record
  - One record per language per top-level object that has a UI component
    - E.g., in an ENU/FRA/JPN environment, there will be three compiled definitions for the *Account List Applet*
  - Stored in the actual database in tables *S_RR_xxxxxx* (e.g., *S_RR_APPLET*, *S_RR_BUSCOMP*)
  - Replaces the legacy “SRF” file

Runtime Repository (RR)

Why a Runtime Repository?

- Runtime Repository allows us to test in parallel (SRF did not)
  - Many definitions of an object may exist in various workspaces
  - Inspect / Open allows selection of which definition to use
- Why compile an RR object?
  - Performance dictates we have a compiled object
  - Example: Applet object relies on ~30 tables; RR definition has one record / object / language
- Allows for Versioning (SRF did not)
  - Each RR object is tagged with a version (*VERSION_NUM*)
  - At runtime, the version context for the OM selects the correct version from the versions available in that workspace
- Object managers always use RR definitions
  - Exception: During *Inspect*, DR definitions compiled on the fly from selected workspace

Runtime Repository (RR)

Miscellaneous Information

- Delivery into an Integration Branch or MAIN generates a Runtime Repository record
  - As noted, “Inspect” on a workspace allows reading from the DR tables without delivery
- Only Runtime Repository records are migrated downstream (QA/Test/Prod)
  - Minimizes migration time
  - Ensures that what is in QA/Test/Prod is what is in Development (unlike SRF)
- Non-development (QA/Test/Prod) are <u>RR-only</u>
  - Contrast: DR environments actually contain DR + RR
- RR environments do not have a workspace hierarchy
  - Only workspace is MAIN
  - Multiple versions of MAIN may exist

Runtime Repository
Versioning

- Each *S_RR_** record has a *VERSION_NUM*
- Object Manager selects correct record at runtime
  - Object Manager defaults to latest version of MAIN
  - Can be overridden using Repository Rollback
- Example
  - Full Migration creates *Account* BusComp in *S_RR_BUSCOMP*
    - *VERSION_NUM*: 0
  - Later migration (say five migrations later) updates the *Account* BusComp
    - *VERSION_NUM*: 5
    - Both records have the same *WS_SRC_ID*
  - Object Manager will normally select the latest version (5)
  - Repository Rollback to Version 4 would pick up the version 0 record
    - Most recent version that is four or less

How Does All This Work?

GOALS

- Learn Underlying Data Model
- Experience behind-the-scenes of Object Life Cycle

Workspaces Overview

Critical Workspace-related Columns

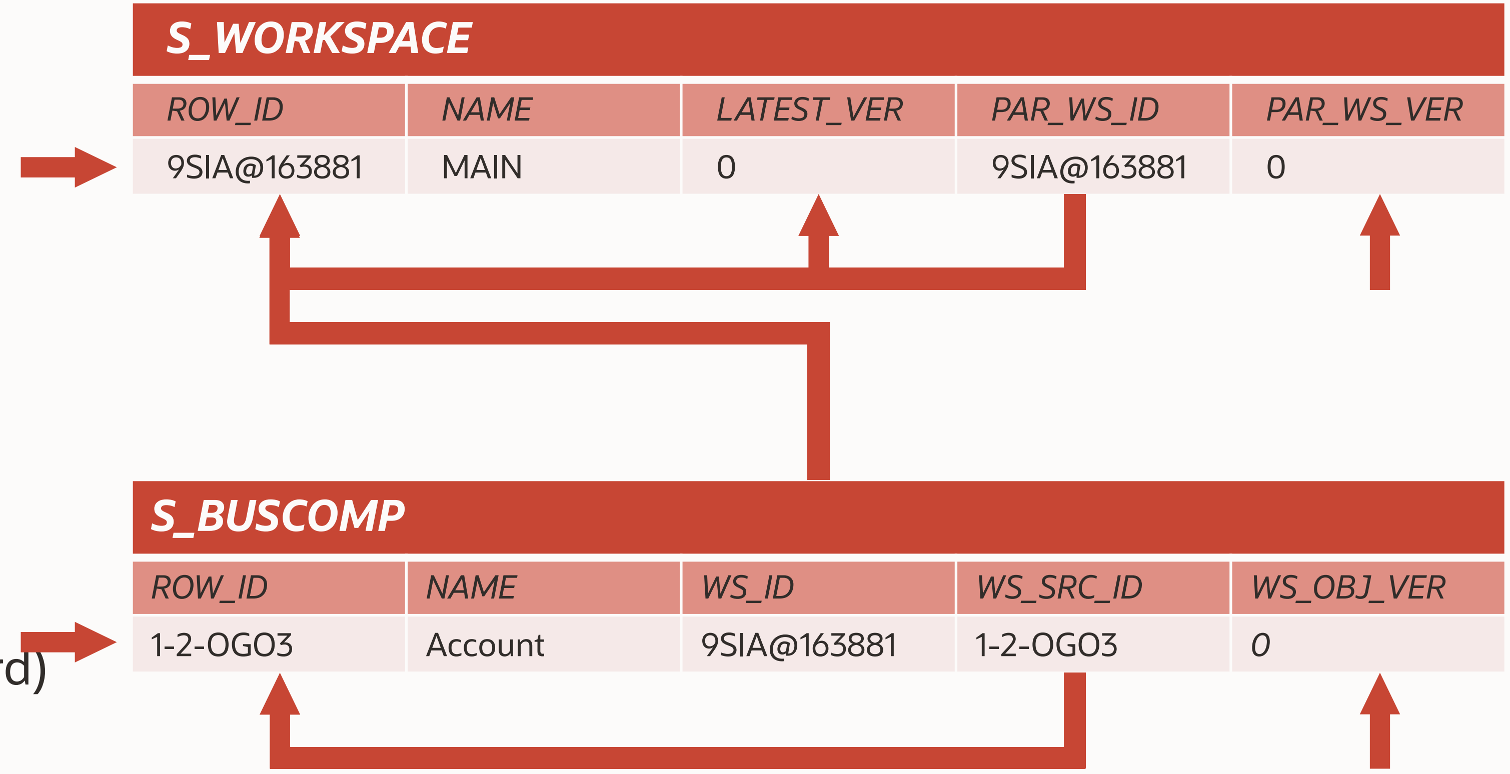
Column	Purpose
WS_ID	The *ROW_ID* of a particular workspace as defined in the table *S_WORKSPACE.* *S_WORKSPACE* is hierarchical, and therefore has: • *PAR_WS_ID*—a FK to itself identifying the parent workspace • *PAR_WS_VER*—the version number of the parent when the workspace was created
WS_SRC_ID	A **logical identifier** for a particular object across all workspaces and versions. Example: The *Account* BusComp will have the <u>same</u> *WS_SRC_ID* in every workspace across every instance
WS_OBJ_VER	Represents the version of an object within a given workspace. Incremented every time that object is modified <u>within that workspace</u>

Workspaces Overview

Workspace Life Cycle—Example

Initial State

- *S_WORKSPACE*
  - Single record (MAIN)
  - *PAR_WS_ID* = *ROW_ID* (root node)
  - *LATEST_VER*
  - *PAR_WS_VER*
- *S_BUSCOMP*
  - Single record (*Account* BusComp)
  - *WS_ID* (object is in MAIN)
  - *WS_SRC_ID* = *ROW_ID* (original record)
  - *WS_OBJ_VER* (zeroth in workspace)



S_WORKSPACE

ROW_ID	NAME	LATEST_VER	PAR_WS_ID	PAR_WS_VER
9SIA@163881	MAIN	0	9SIA@163881	0

S_BUSCOMP

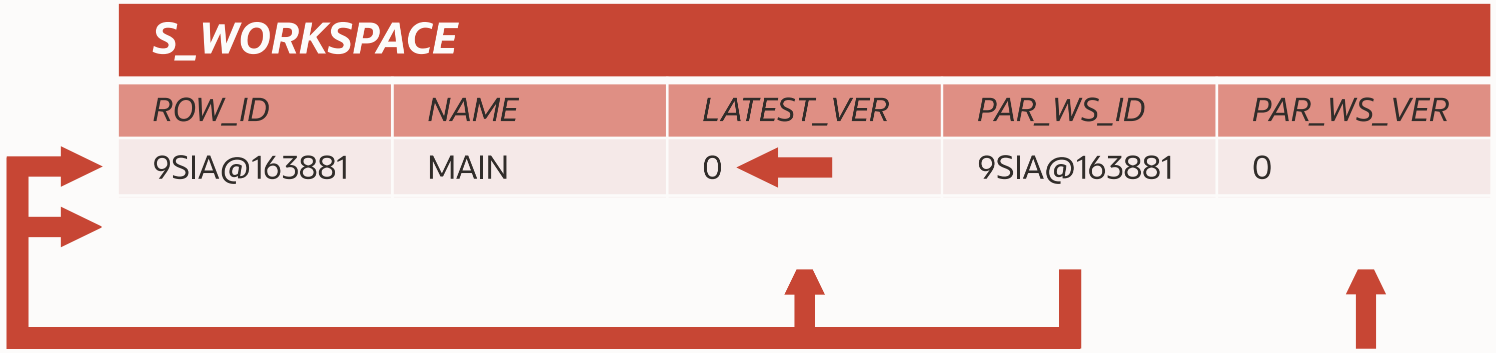
ROW_ID	NAME	WS_ID	WS_SRC_ID	WS_OBJ_VER
1-2-OGO3	Account	9SIA@163881	1-2-OGO3	0

Workspaces Overview

Workspace Life Cycle—Example

Create Developer Workspace

- *S_WORKSPACE*
  - New record for new Workspace
  - *PAR_WS_ID* = *ROW_ID* of MAIN
  - *LATEST_VER*
  - *PAR_WS_VER*
- *S_BUSCOMP*
  - No changes



S_WORKSPACE

ROW_ID	*NAME*	*LATEST_VER*	*PAR_WS_ID*	*PAR_WS_VER*
9SIA@163881	MAIN	0	9SIA@163881	0

S_BUSCOMP

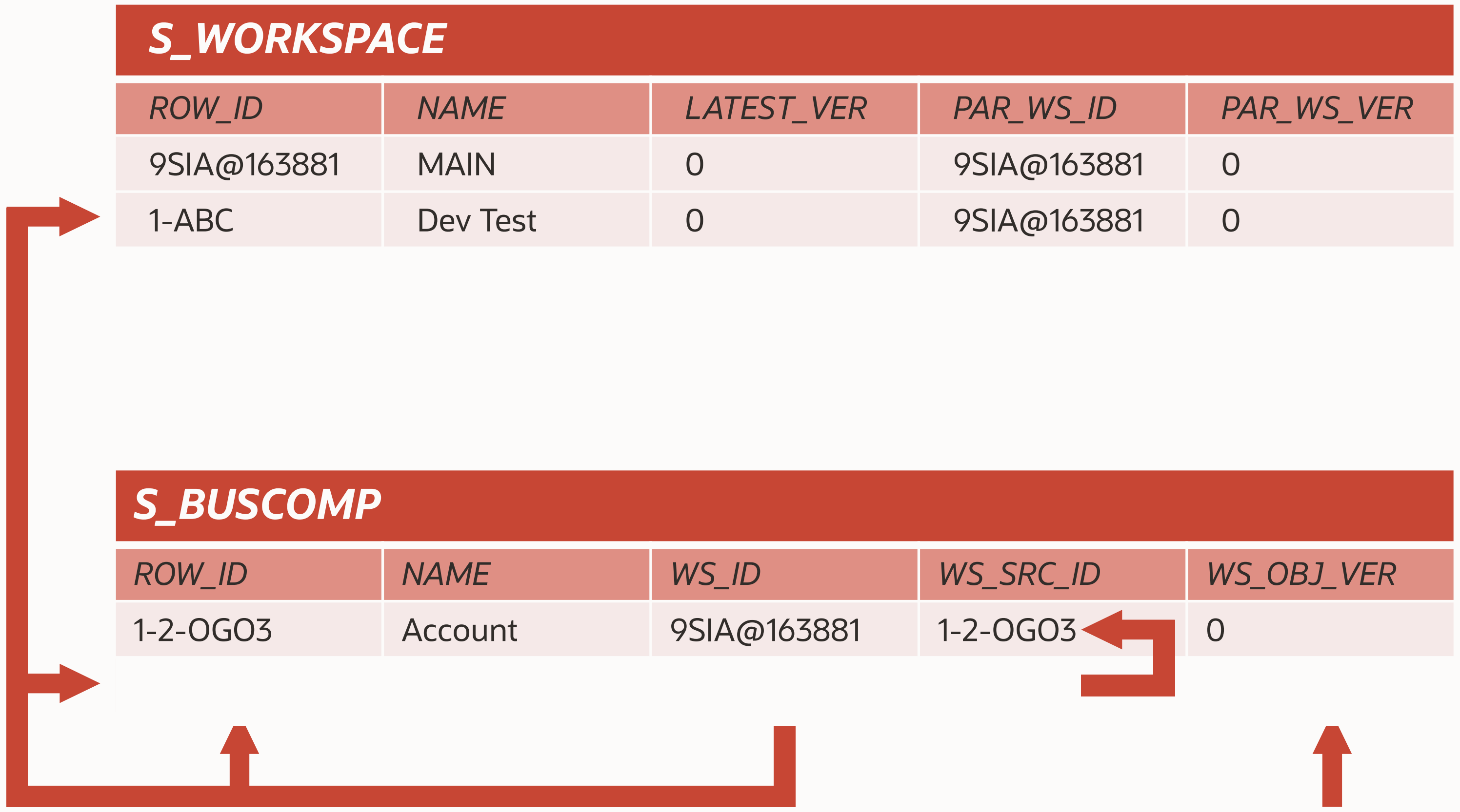
ROW_ID	*NAME*	*WS_ID*	*WS_SRC_ID*	*WS_OBJ_VER*
1-2-OGO3	Account	9SIA@163881	1-2-OGO3	*0*

Workspaces Overview

Workspace Cycle—Example

Change *Account* BusComp

- *S_WORKSPACE*
  - No Changes
- *S_BUSCOMP*
  - New record with new *ROW_ID*
  - *Workspace Id* refers to the Developer's Workspace
  - *Workspace Source Id* points to base *Account* BusComp record—indicates same logical object
  - *Workspace Object Version*—each change in a Workspace, starting with 100,001 (for first level workspaces—200,000 for second level, etc.)



S_WORKSPACE

ROW_ID	*NAME*	*LATEST_VER*	*PAR_WS_ID*	*PAR_WS_VER*
9SIA@163881	MAIN	0	9SIA@163881	0
1-ABC	Dev Test	0	9SIA@163881	0

S_BUSCOMP

ROW_ID	*NAME*	*WS_ID*	*WS_SRC_ID*	*WS_OBJ_VER*
1-2-OGO3	Account	9SIA@163881	1-2-OGO3	0

Workspaces Overview

Workspace Life Cycle—Example

Checkpoint Workspace

- *S_WORKSPACE*
  - Updates Workspace's Latest Version
- *S_BUSCOMP*
  - No changes

S_WORKSPACE

ROW_ID	NAME	LATEST_VER	PAR_WS_ID	PAR_WS_VER
9SIA@163881	MAIN	0	9SIA@163881	0
1-ABC	Dev Test	1	9SIA@163881	0

S_BUSCOMP

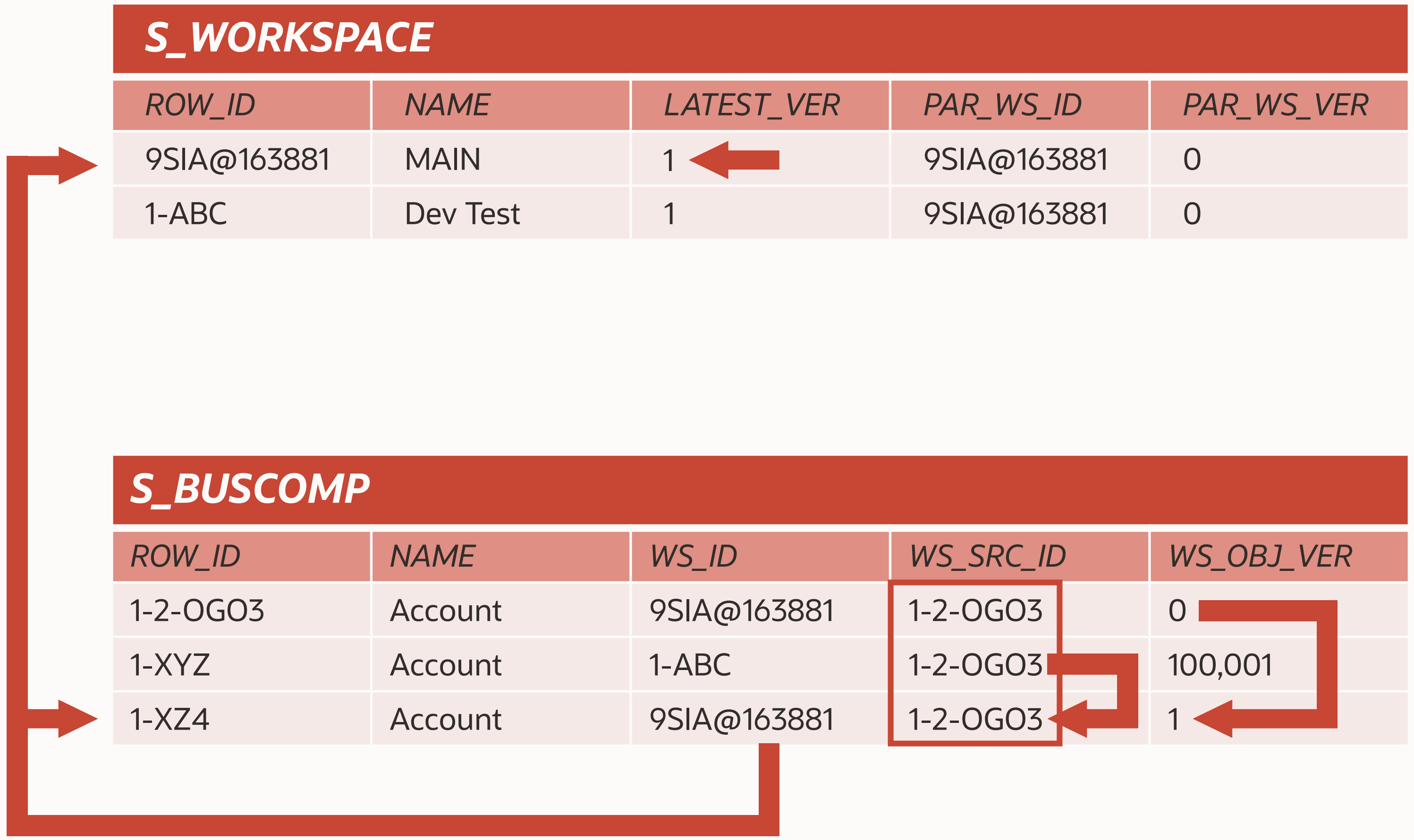
ROW_ID	NAME	WS_ID	WS_SRC_ID	WS_OBJ_VER
1-2-OGO3	Account	9SIA@163881	1-2-OGO3	0
1-XYZ	Account	1-ABC	1-2-OGO3	100,001

Workspaces Overview

Workspace Life Cycle—Example

Deliver Workspace

- *S_WORKSPACE*
  - MAIN updated with new Version
- *S_BUSCOMP*
  - New record with new *ROW_ID*
  - *Workspace Id* set to MAIN
  - *Workspace Source Id* copied from Workspace
    - Note that all match!
  - *Workspace Object Version* is one higher than previous version in same Workspace



S_WORKSPACE

ROW_ID	*NAME*	*LATEST_VER*	*PAR_WS_ID*	*PAR_WS_VER*
9SIA@163881	MAIN	1	9SIA@163881	0
1-ABC	Dev Test	1	9SIA@163881	0

S_BUSCOMP

ROW_ID	*NAME*	*WS_ID*	*WS_SRC_ID*	*WS_OBJ_VER*
1-2-OGO3	Account	9SIA@163881	1-2-OGO3	0
1-XYZ	Account	1-ABC	1-2-OGO3	100,001
1-XZ4	Account	9SIA@163881	1-2-OGO3	1

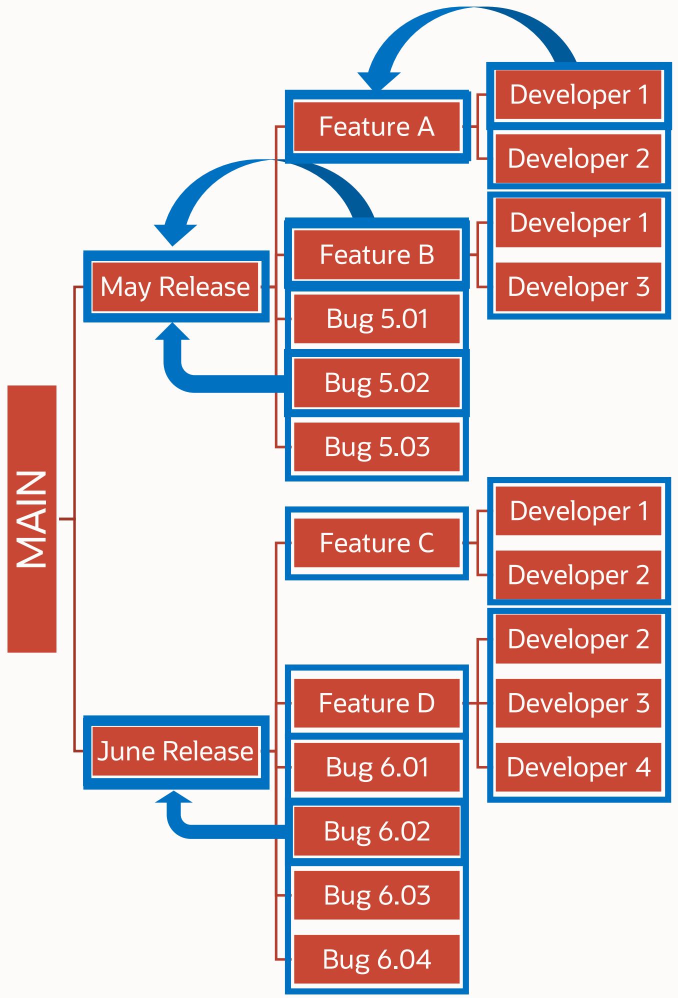
Best Practices

GOALS

- Understand (basic) recommended workspace structure

Workspaces Overview

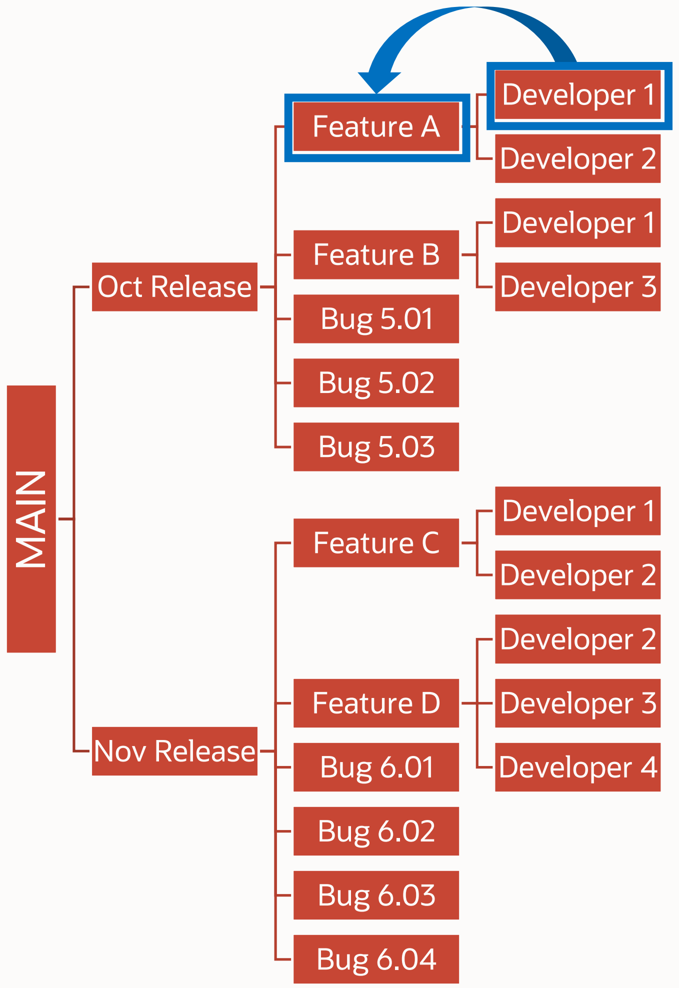
Basic Structure (Example Only)



- Two releases in parallel (May and June)
- Each release has its own features
- Each release also has bug fixes planned (and each has its own workspace)
- Each feature has developers working on it with each developer having his or her part in a unique Developer Workspace
- As developers finish their work on a bug or partial feature, they deliver to the parent branch for integration testing
- When entire feature is ready, deliver to release level

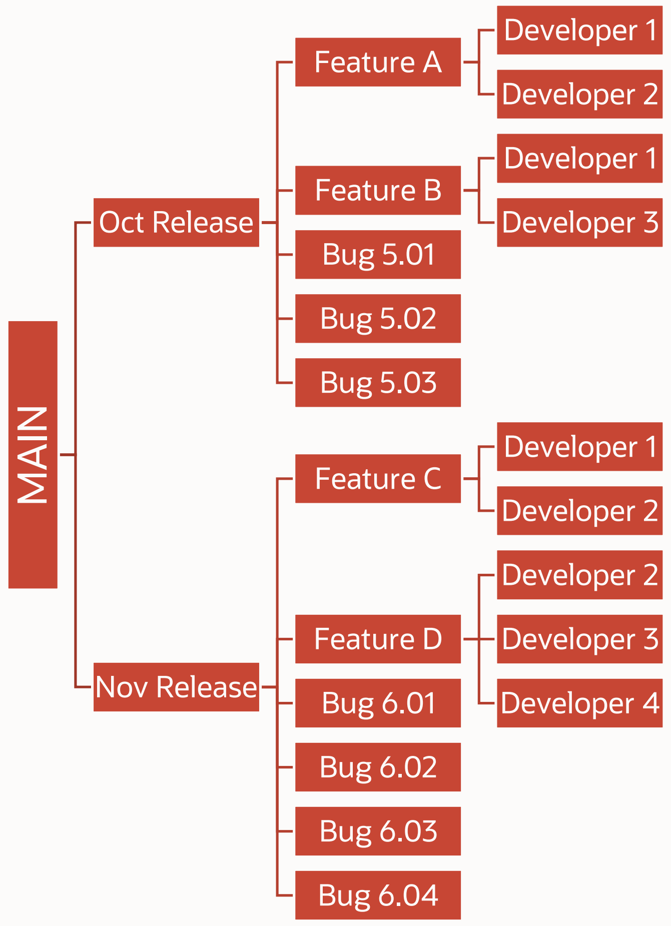
Workspace Best Practices

Change in Approach



- Development environment not just for developers
- QA Team embedded with Development Team
  - Pre-validates changes in Dev Workspace
  - Delivery allowed only after pre-validation
- Product Owner validates features in Development
- Allows for…
  - …early detection of defects
  - …better quality features (identify design gaps, allow for agile changes)
- Downstream environments more stable
- Downstream Test Environments?
  - User Acceptance Testing
  - Integration Testing
  - Pre-Production

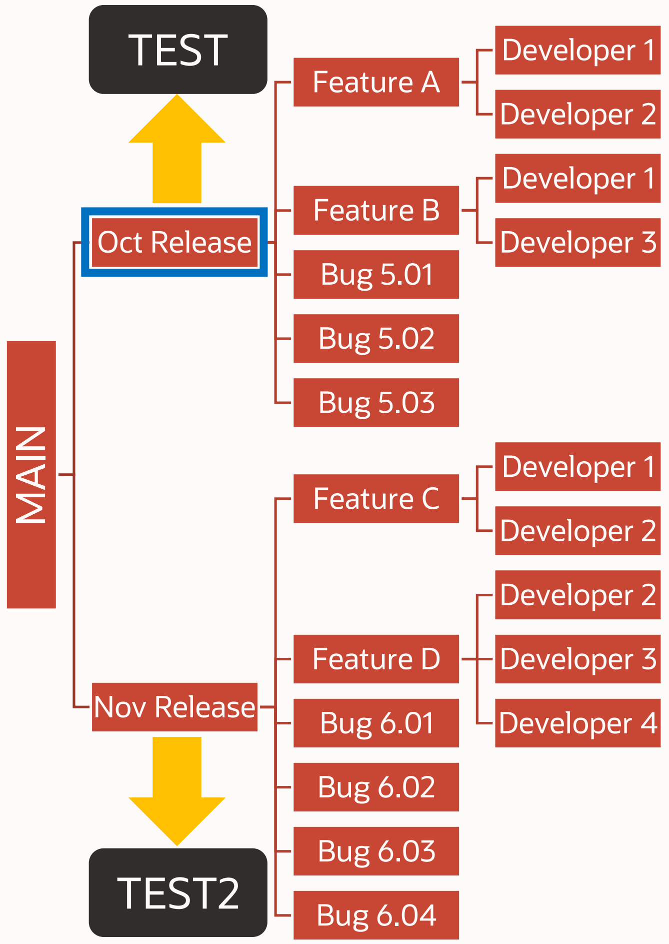
Workspace Best Practices
Mapping Downstream Environments



- Various Test and Production (RR) environments
  - Populated by Integration Branches in DR
  - RR → RR not possible (e.g., no “Test → Prod”)
- Generally…
  - Integration Branches → Test
  - MAIN → Production
  - (Optionally) MAIN child for Pre-Production

Workspace Best Practices

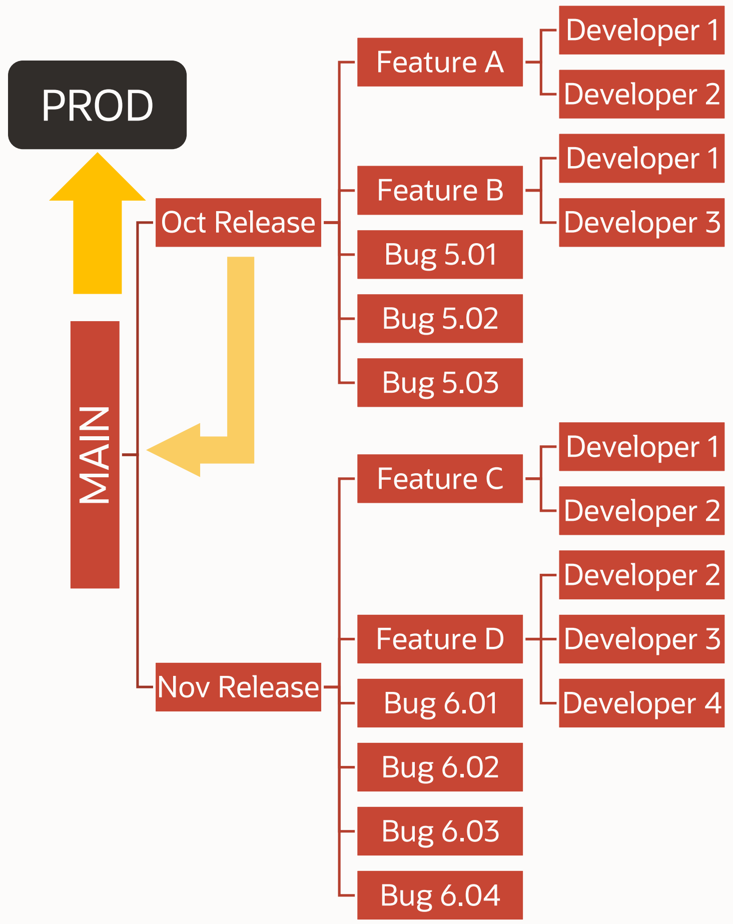
Mapping Downstream Environments



- Test environment reflects next release
  - Populate as soon as previous release GA
  - Repeatedly populate as changes delivered
- Multiple environments
  - November mapped to second Test
  - Like October, can be constantly updated
  - Options
    - Multiple Repositories in destination
    - Virtual Machines to make environments "throwaway"

Workspace Best Practices

Ready for Production!!!

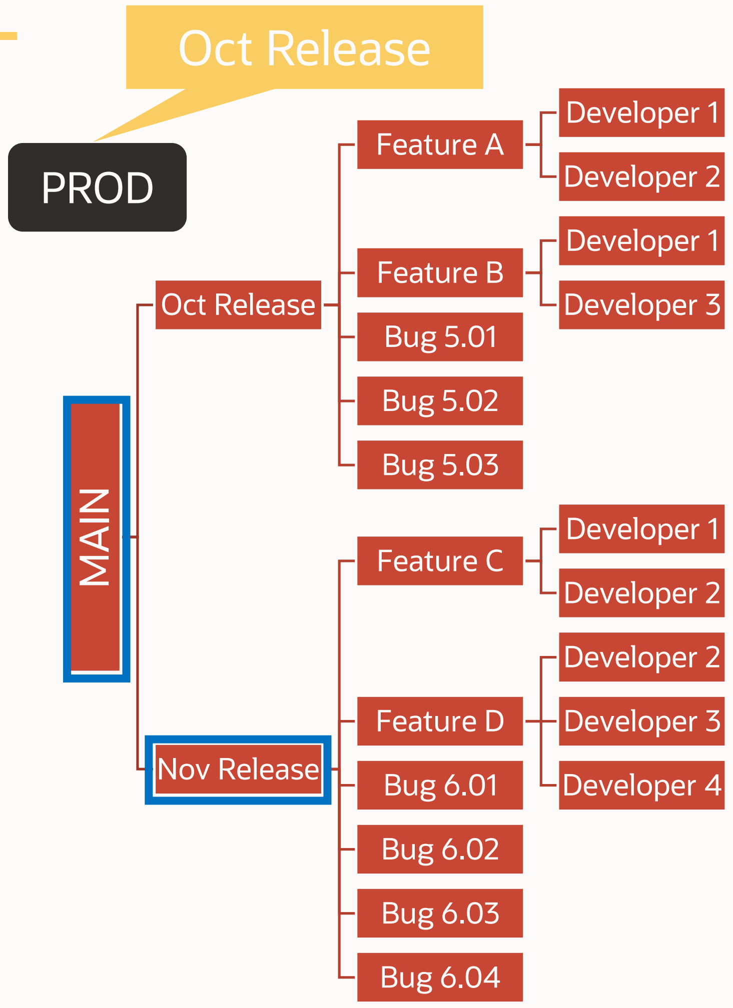


Two step process

- Deliver Release to MAIN
  - “Oct Release” known to be good
  - MAIN now equals “Oct Release”
- *Immediately* migrate MAIN to Production
  - Production now matches “Oct Release”

Workspace Best Practices

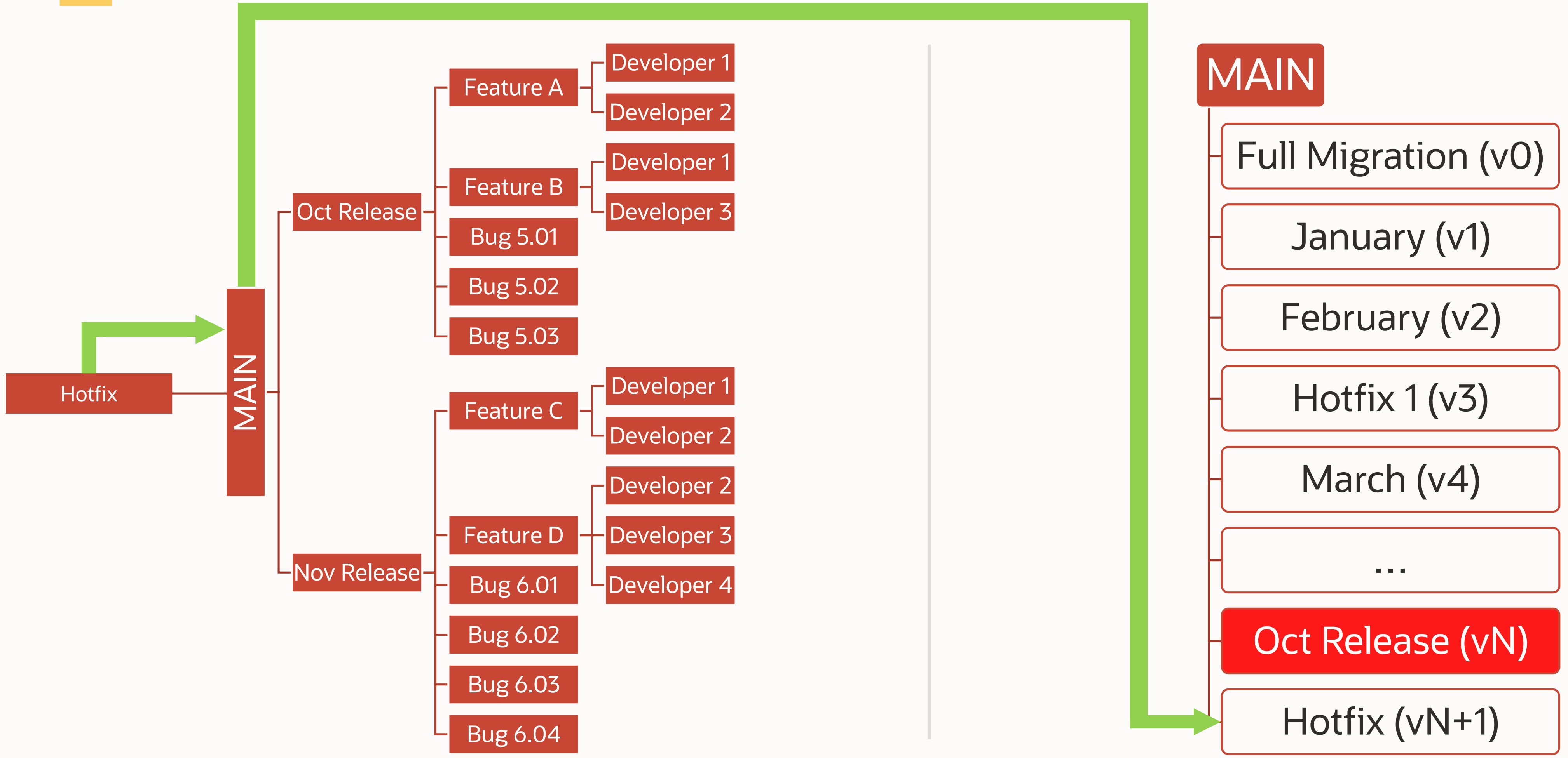
Why Promote to Production Immediately



- Consider Production after “Oct Release”
  - Matches October release
- What if hotfix needed before “Nov Release”?
  - “Nov Release” not ready—cannot promote
  - Must build release directly off MAIN
  - Ergo, MAIN must always match Production

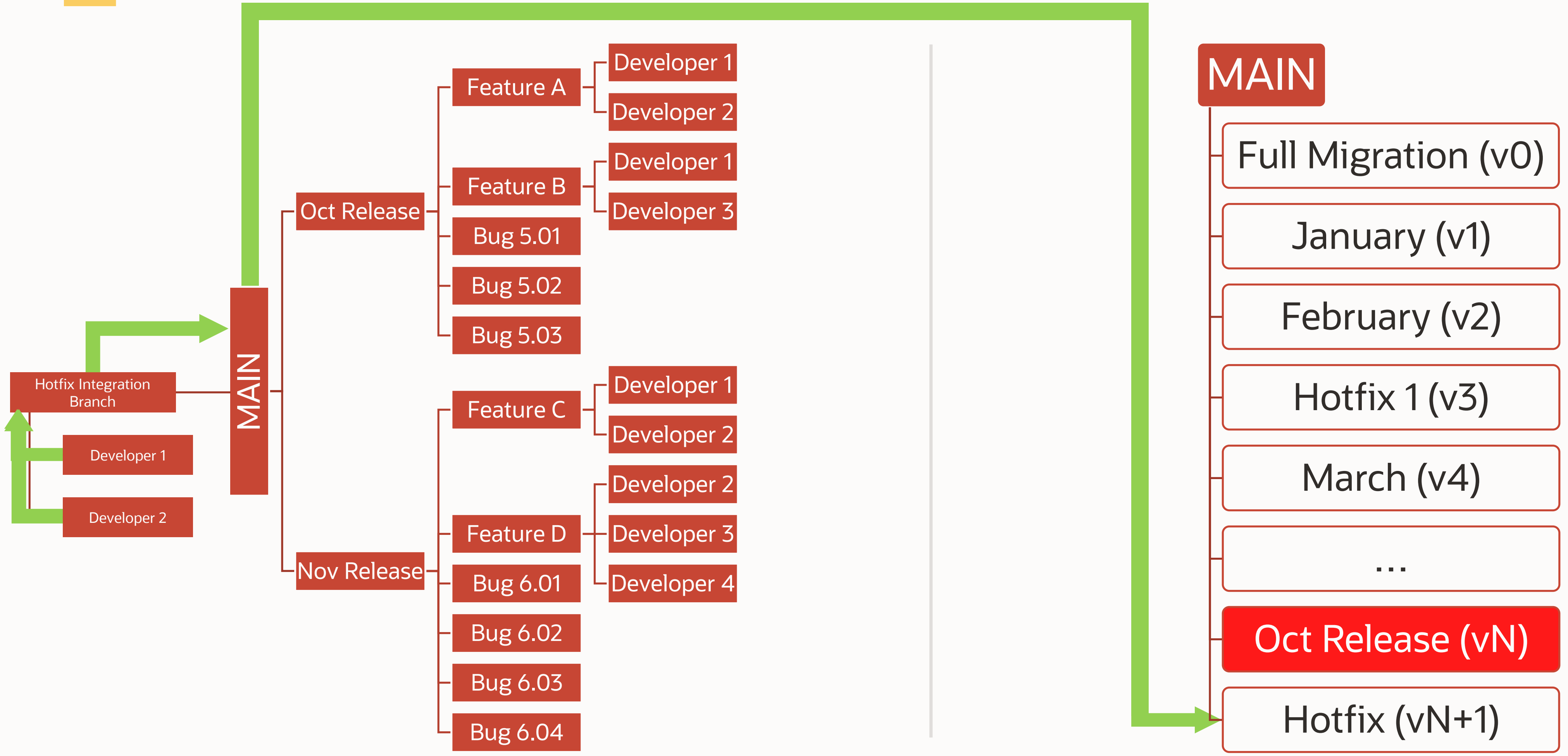
Production Hotfix Needed

Simple Case—Resolve off MAIN



Production Hotfix Needed

Advanced Case—Resolve off MAIN



Advanced Options Example
Pre-Production Scenario

GOALS

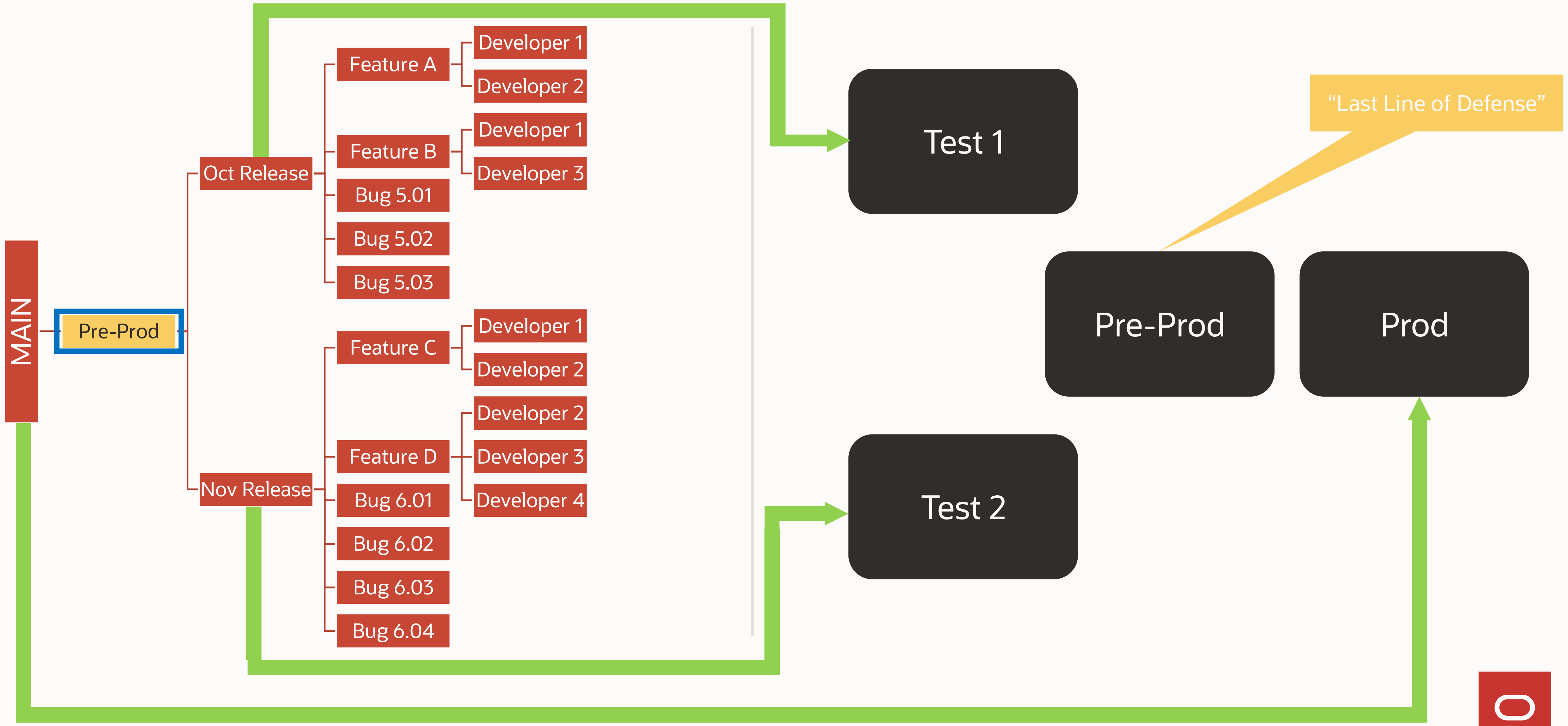
- Consider (one) option for variation

Workspaces Management—Advanced
Pre-production Option

- Repository, seed, and manifest updates…
  - …thoroughly tested in Test
  - …assumed to be correct
- Pre-Production final failsafe
  - Typically a clone of Production
  - Final opportunity to catch a configuration or other issue

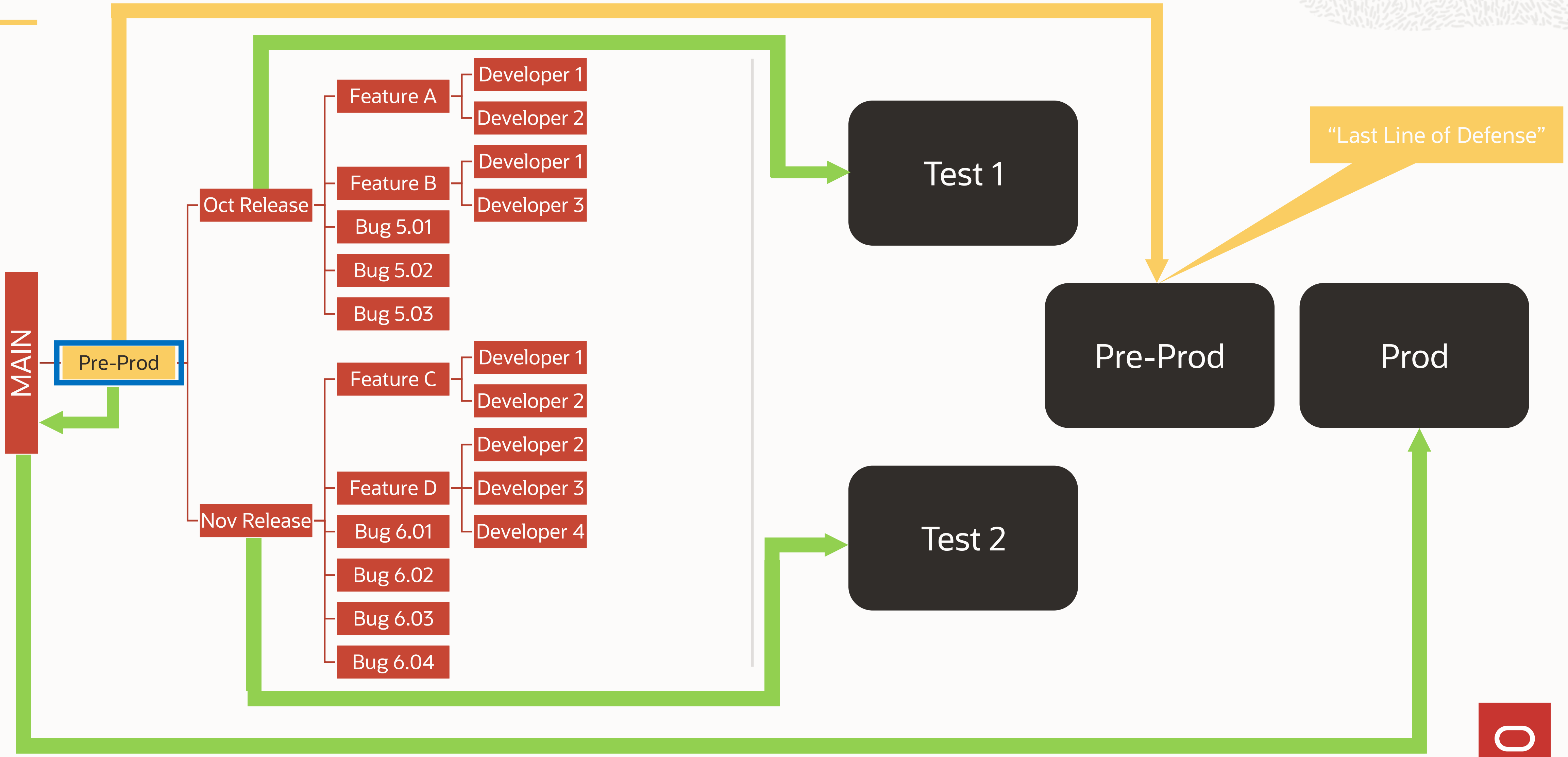
Workspace Best Practices

Introduction of Pre-Production Branch



Workspace Best Practices

Last Line of Defense



Key Takeaways

1 Workspaces is a sophisticated parallel development and sandboxing solution for Developers and agile teams

2 Flexible, hierarchical framework for building features and multiple releases in parallel, with versioning mechanism and governance

3 Improve Developer productivity with enhanced application development UX

Thanks!

Brian Kelly

brian.kelly@oracle.com

blogs.oracle.com/siebelcrm

ORACLE

Stay Connected

blogs.oracle.com/siebelcrm

Take the Siebel CRM Innovation Survey

Let us help you kickstart your Siebel CRM transformation

https://go.oracle.com/siebelcrm-innovation

Useful Resources

- Siebel CRM Blog
- Siebel CRM YouTube
- Siebel CRM Sales Team 🖂
- Siebel CRM ACS Services 🖂
- Oracle Support Value
- Partner Spotlights

- Siebel CRM Learning Subscription (Free content, click Preview)
- Siebel CRM Bookshelf
- Siebel CRM Github
- Siebel CRM Advisor Webcasts
- My Oracle Support Community

- Siebel CRM Statement of Direction
- Siebel CRM Release Updates
- Siebel CRM Premier Support
- Datasheets – Features by Release
- Siebel CRM Ideas (Collaboration)

- Siebel CRM Customer Connect CAB portal
- LinkedIn Customer Connect
- Newsletter Email Distribution list (Customer) & (Partner)
- Virtual CAB replays