Putting p2 into practice: Releasing independent products from a shared code base

Robert Mischke
German Aerospace Center (DLR)
EclipseCon Europe 2012

DLR

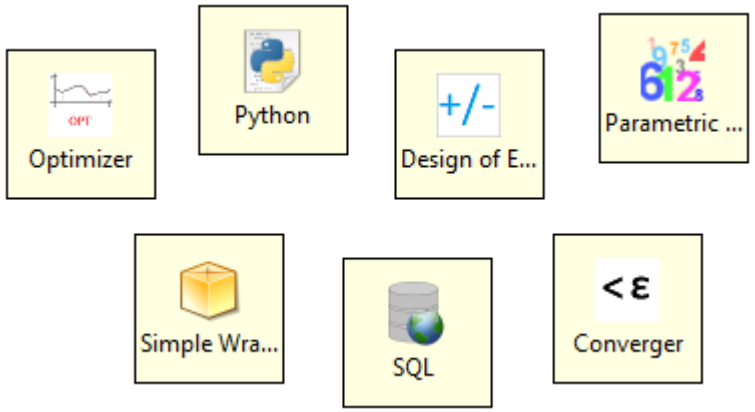
Outline

- Introduction
- Important terms
- The multi-product p2 approach
- Developer roles/scenarios
- The Maven/Tycho setup
- Conclusion

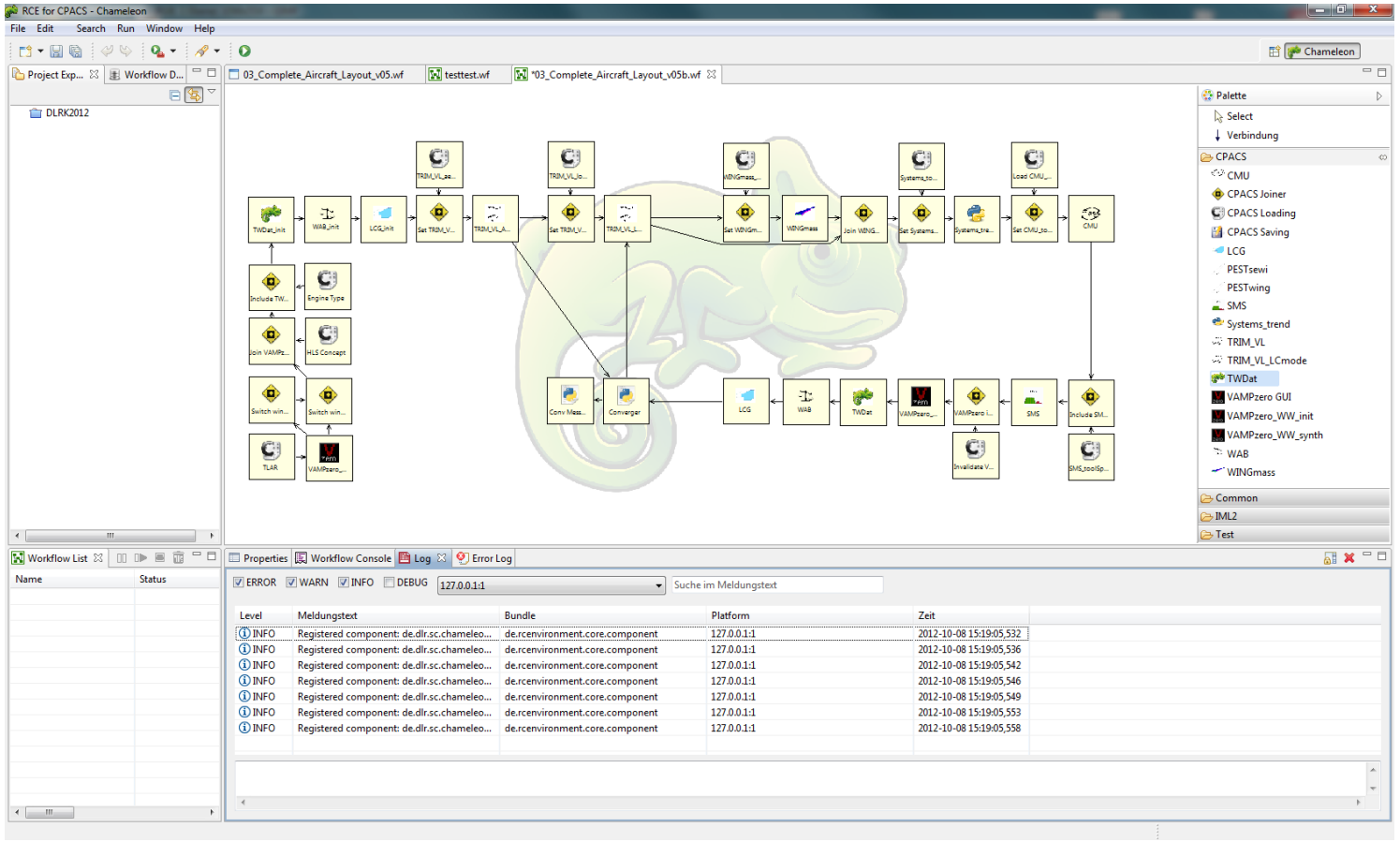
Introduction – Motivation & Requirements

- Example project: The “Remote Component Environment” (RCE)
  - Distributed, workflow-driven integration environment
  - Open Source (EPL)
  - Based on Equinox/RCP
  - Research and simulation applications

- Common framework and UI
- GEF-based workflow editor
- Workflow components
  - Common
  - Project-specific



Introduction – Motivation & Requirements (2)



Introduction – Motivation & Requirements (3)

- Very project-specific requirements
- A release may contain…
  - …standard components
  - …custom components
    - (may evolve into standard components!)
  - …large binaries
    - component-driven (usually platform-specific)
    - embedded JVM
  - …custom libraries
  - …custom external features
  - …custom branding
  - …custom configuration, documentation, …

Introduction – Motivation & Requirements (4)

- Organisational requirements
  - Custom release cycles
  - Compatibility within work groups
    - Choice of framework version
  - Project life cycle: Feature releases stop…
    - …but may resume later (follow-up projects!)
    - Maintenance fixes
    - Long-term build stability

Introduction – Motivation & Requirements (5)

- Consequence: Independent **distributions**
  - Basically: software product line approach
    - Custom combination of elements
    - Custom choice of framework version
    - (…and other parts; more on that later)
  - Released independently
    - Custom versioning
  - Results in one or more (Eclipse) products

- **How to build – and manage – this with Maven / Tycho / p2?**

Introduction – Recap of important terms

- p2 repository
- Eclipse product
- Tycho
- Target platform

Holding OSGi Artifacts

- JARs with meta data
- Two basic approaches:
  - Install/deploy into Maven repo(s)
    - Versioned on artifact level
    - „Big box of everything“; stateful
    - Actively assemble p2 repositories when needed
  - Generate p2 repository on every build
    - Atomic blocks of related artifacts
    - Less disk space efficient
    - Deployed to web server (rsync)
    - Immediately usable as target platform

p2 repository strategy – Initial decisions

- Followed here: p2 repository approach
  - Used in all build stages
  - Stored locally or on web server
  - All content provided by plugin or feature
- Fundamental questions:
  - How many repositories?
  - What goes into which one?
  - Versioning and snapshots?
  - Multiple products/distributions?

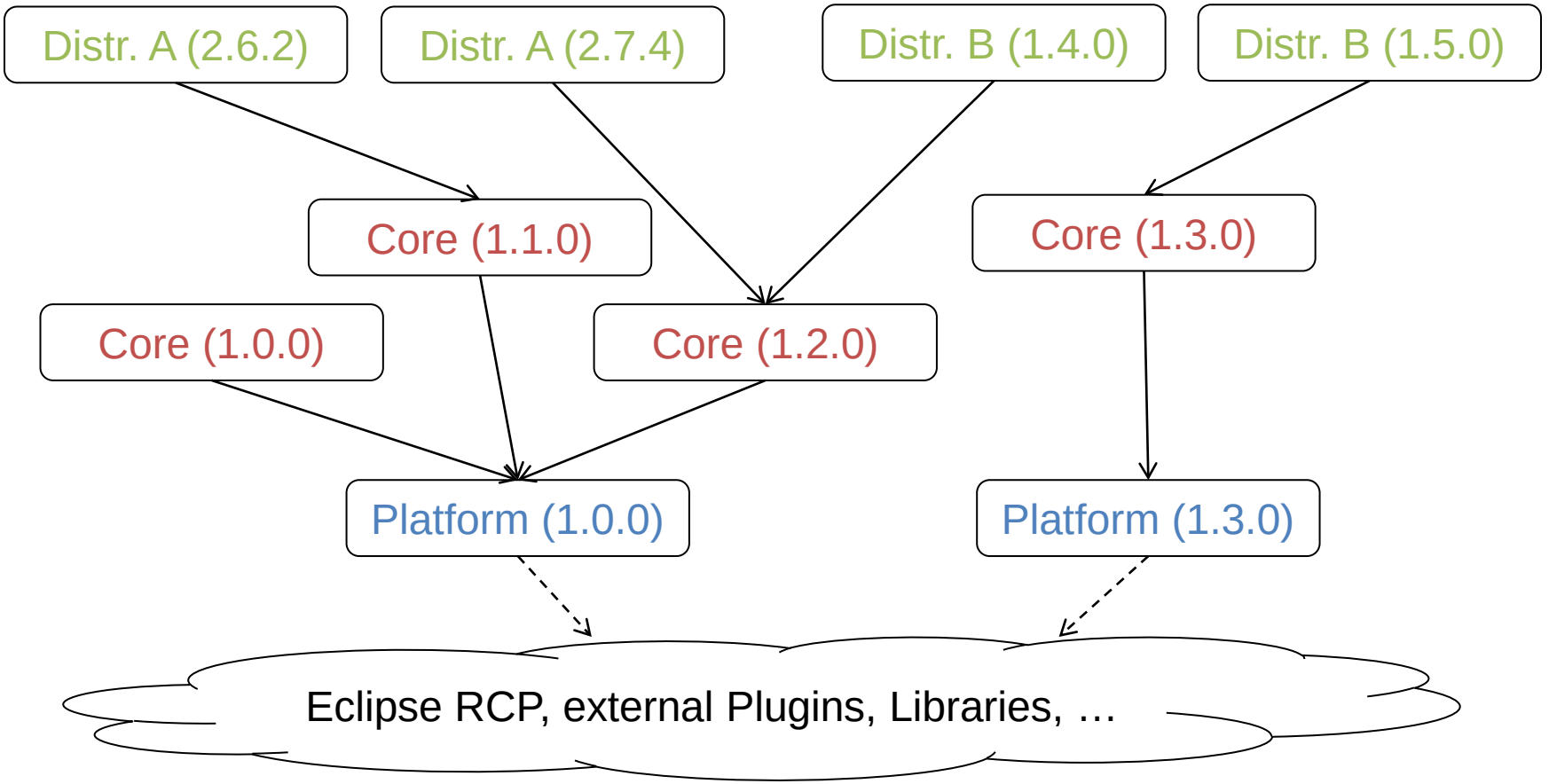
p2 repository strategy – Basic layout

- Our approach:
  - **Three layers** of repositories
    - Distribution
    - Core
    - Platform
  - Each repository is **self-contained**
    - Important decision!
    - Disk space vs. maintainability
    - Higher-level repositories **can** cherry-pick elements
  - **Versioning** and snapshot provisioning **on each layer**

p2 repository strategy – The layers

- Platform repository
  - All external resources
    - Libraries
    - External plugins and features
- Core repository
  - Common/shared elements
    - Includes common files and branding
- Distribution repositories
  - Distribution-specific elements
    - Includes custom files and branding
  - (Location of product definitions)

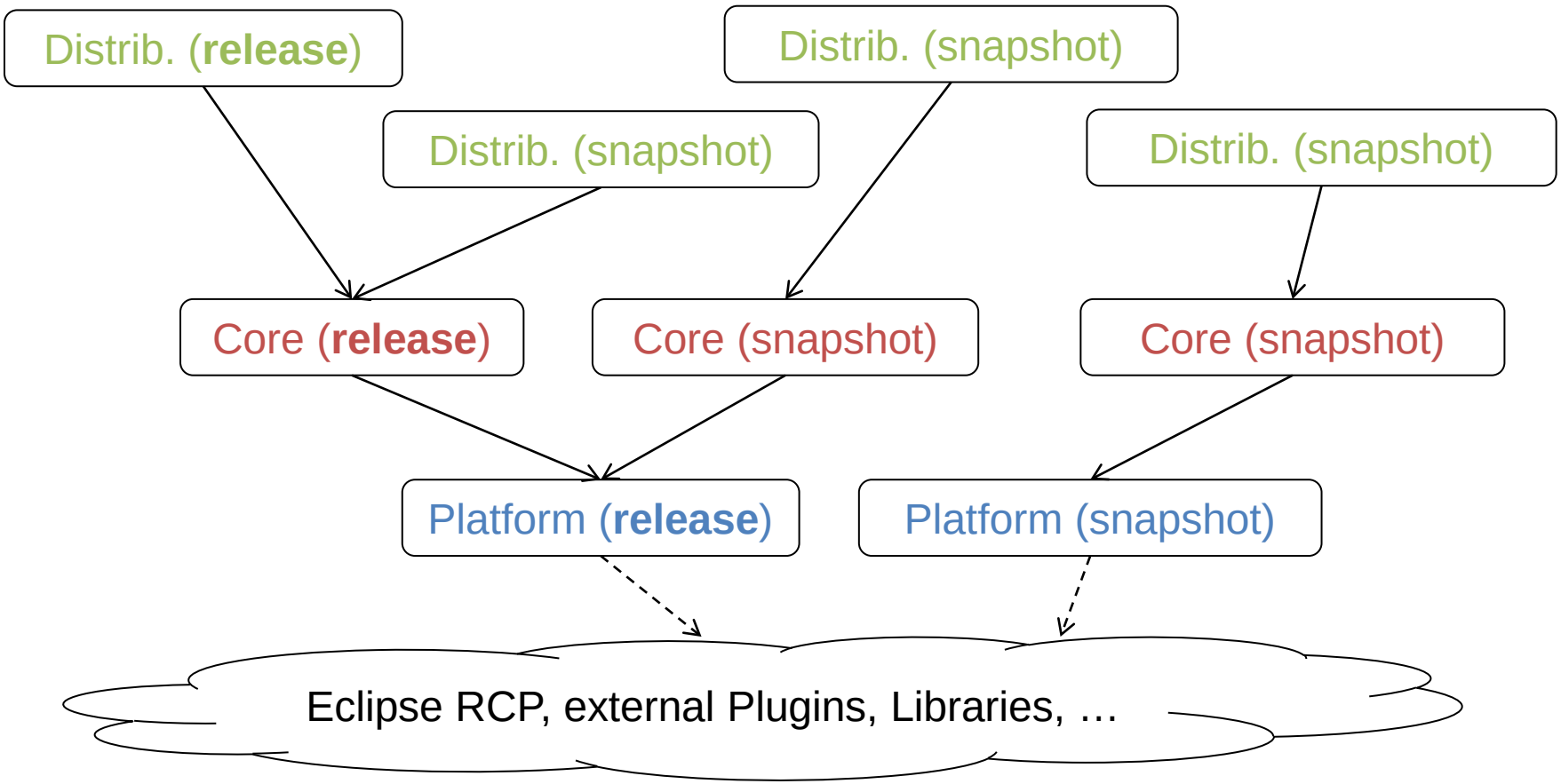
p2 repository strategy – Basic layout



p2 repository strategy – The platform repository

- Why a separate platform repository?
  - Encapsulate „messy“ details
    - Source repository URLs
    - Version incompatibilities
    - Library classpaths
  - Provide „clean“ OSGi to upper layers
  - Separation of roles
    - Platform maintainer
  - Reduce update size
    - Changes less frequently than core repository content
    - Release/update only when necessary

p2 repository strategy – Snapshot options



p2 repository strategy – Snapshot options (2)

- Very flexible
  - All variants have use cases
- Small downside: snapshots are slightly ambiguous
  - „Distribution snapshot“: using release or snapshot core?
  - „Core snapshot“: using release or snapshot platform?
- Not much of a problem in practice

Developer roles and target platforms

- Each self-contained repository is a complete target platform!

- Distribution developer
  - Set fitting core repository as TP
  - Check out distribution code
  - Start developing
  - (Need an unreleased core feature? Set core snapshot as TP!)

- Core developer
  - Set fitting platform repository as TP
  - Check out core code
  - Start developing

Developer roles and target platforms (2)

- (Mixed core/distribution developer?)
  - Set platform repository as TP
  - Check out distribution(s) **and** core code
  - Start developing

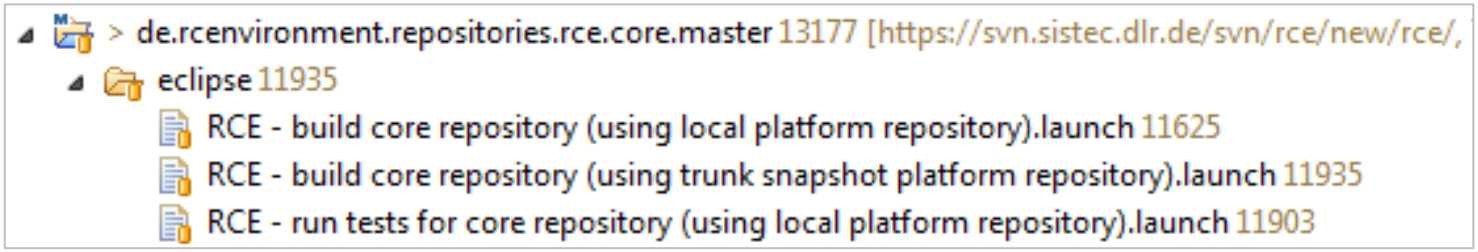
- Special role: Platform repository maintainer
  - Works on Maven (instead of PDE) level
- Goal: modify platform and verify changes
  - Check out platform and core/distribution code
  - Modify platform setup
  - Build local snapshot of platform repo
    - Set as TP, confirm that core/distribution works

p2 repository strategy – Local/remote building

- Each repository can be built
  - …as a local snapshot
  - …as a deployed snapshot (from CI)
  - …as a deployed release
- Practical questions:
  - How to consume repositories (as TP)?
  - How to build them?
- Consumption
  - Provide **.target** files for TP options

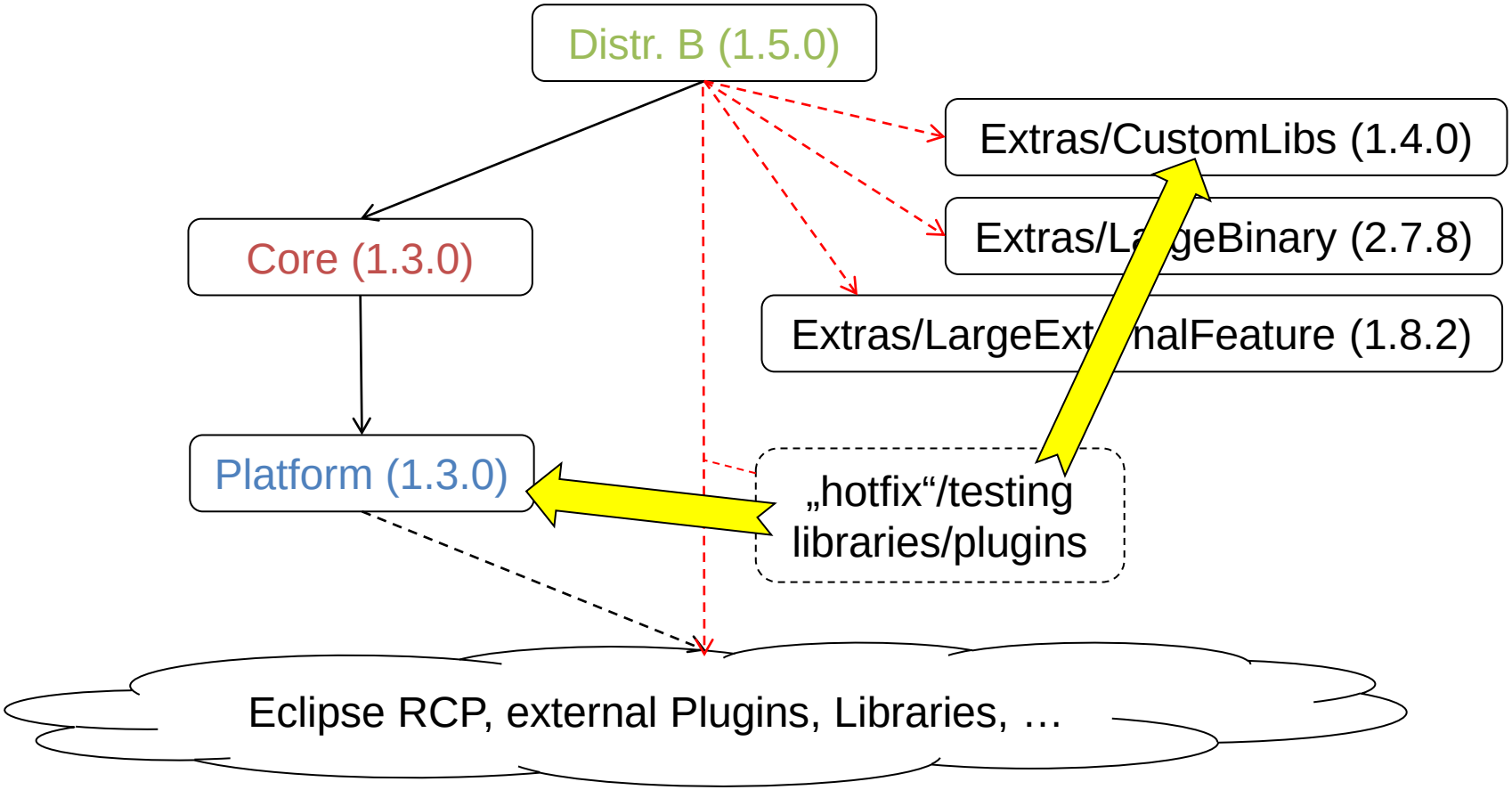
p2 repository strategy – Local/remote building (2)

- Recommended build setup:
  - Provide **.launch files** for local snapshots



  - „Snapshot“ Jenkins job(s)
    - Deploys to web server
    - Example: http://<server>/eclipse/rce/**core/snapshots/trunk**
  - „Versioned release“ Jenkins job
    - Parametrized: Builds from release branch/tag
    - Example: http://<server>/eclipse/rce/**core/releases/2.3/2.3.6**

p2 repository strategy – The exceptions



The Maven/Tycho build – Overview

- Foundation: Tycho repository and product builds
- Special aspects
  - Handling local snapshots / deployed snapshots / deployed releases
  - Build stability: avoiding static p2 URLs
  - Building the platform repository
- Also:
  - The final product build
  - Providing end-user updates

The Maven/Tycho build – URLs and versioning

- Split „input“ repository URL into **root** and **version** part
  - Example **root**: “http://<server>/eclipse/rce/platform/releases/”
  - Example **version**: “2.3/2.3.1”
- Define **version** part as Maven property (tagged in SCM)
- Define **root** URL outside of tagged code
  - Can be adapted if URL has to change

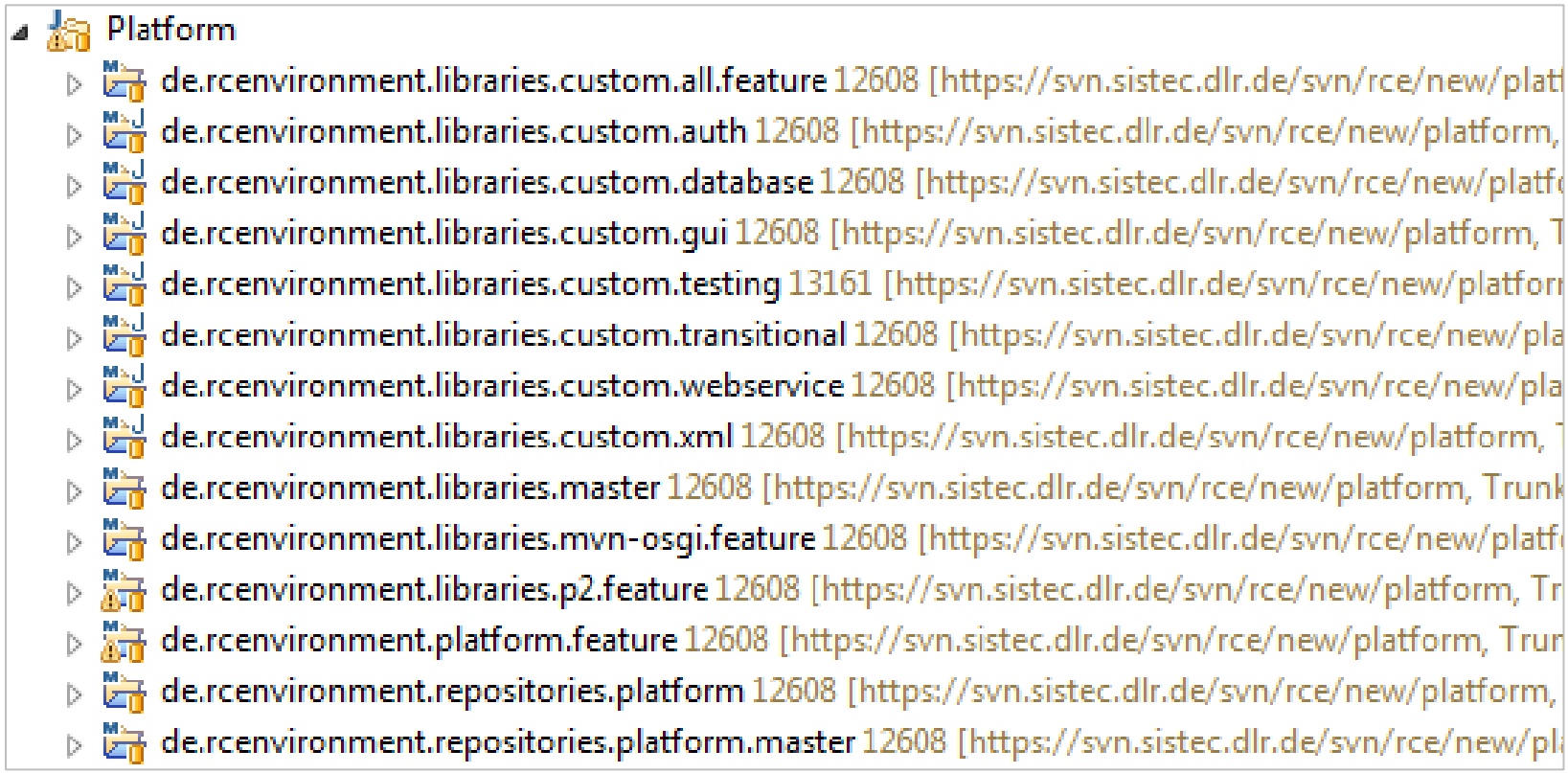
The Maven/Tycho build – URLs and versioning (2)

- Target platform URL for build defined by three Maven profiles:
  - Root URL part given?
    - Building against release; use **root**+**version** URL
  - Full URL given?
    - Building against snapshot; use URL as-is
  - No URL given?
    - Local build: use relative file URL
      - file:/${basedir}/../<repository project>/target/repository/
- (Same principle used for „extras“ repositories on „exceptions“ slide)

The Maven/Tycho build – The platform repository

- 3 types of sources:
  - Bundles from p2 repository
  - JARs with manifest from Maven repo
  - Plain Maven JARs
    - wrap into bundles…
    - …or convert using BND
- Integrating them is not trivial
  - …but usually a one-time effort
- Problem: Getting OSGi bundles from Maven repo to resolve via TP
  - Example: „PAX Logging“ providing commons-logging
  - Solution: Create „stub“ for TP resolution; not in final repo

The Maven/Tycho build – The platform repository



The Maven/Tycho build – Creating the products

- Recommended setup:
  - One project for the distribution repository
    - category.xml
    - product definition(s)
    - POM with tycho-p2-director-plugin instructions
  - One „Maven master“ project
    - invokes plugin, feature and repository modules

The Maven/Tycho build – Creating the products (2)

- de.rcenvironment.repositories.rce.main 13177 [https://svn.sistec.dlr.de/svn/rce/new/rce/, Trunk:
  - resources 11508
  - target
  - .project 11622
  - category.xml 13177
  - pom.xml 13177
  - productParent.pom 13177
  - rce_default.p2.inf 12581
  - rce_default.product 13177
- de.rcenvironment.repositories.rce.main.master 13177 [https://svn.sistec.dlr.de/svn/rce/new/rce/,
  - eclipse 11626
  - .project 11622
  - pom.xml 13177

The Maven/Tycho build – Creating the products (3)

- Possible: More than one product per distribution repo!
  - Useful for distribution **variants**
    - (minimal/full, with/without JVM, …)
  - Can be built together

- Distributions vs. product variants
  - Guideline: Products always released/versioned together?

The Maven/Tycho build – End-user updates

- Use distribution repositories as „software sites“
  - Also for providing optional features
    - Include in repository, but not in product
- Register repository URL(s) in product-specific p2.inf
  - p2 touchpoint instructions
  - Example: rce_default.p2.inf
- Tip: Use symlinks („/latest“) to limit automatic upgrades
  - Change the URL via touchpoint if desired

Conclusion – Main benefits

- Building products locally: huge benefit to developers/maintainers
  - Rapidly verify deployed products
  - Eliminates round-trips to CI server
- Self-contained platform repository
  - Network-independent
  - Fast!

Conclusion – Main benefits (2)

- Layers of self-contained bundles
  - Easy versioning and snapshots
  - Separation of roles
  - Developers can get started quickly
- Long-term build stability
  - No URLs in tagged files
  - Archived repositories

Conclusion – Summary

- Switching to p2/Tycho setup: overall positive experience
- Expect learning and setup time!
  - Probably useful: designated release engineer
- Chosen approach works well
  - Distribution maintainers quite independent
  - Low overhead for new distributions
  - Only drawback: disk usage; outweighted by benefits
- Overall: Recommended!

Questions?

Contact

- RCE Project Page:
  http://software.dlr.de/p/rcenvironment/

- Email:
  robert.mischke@dlr.de