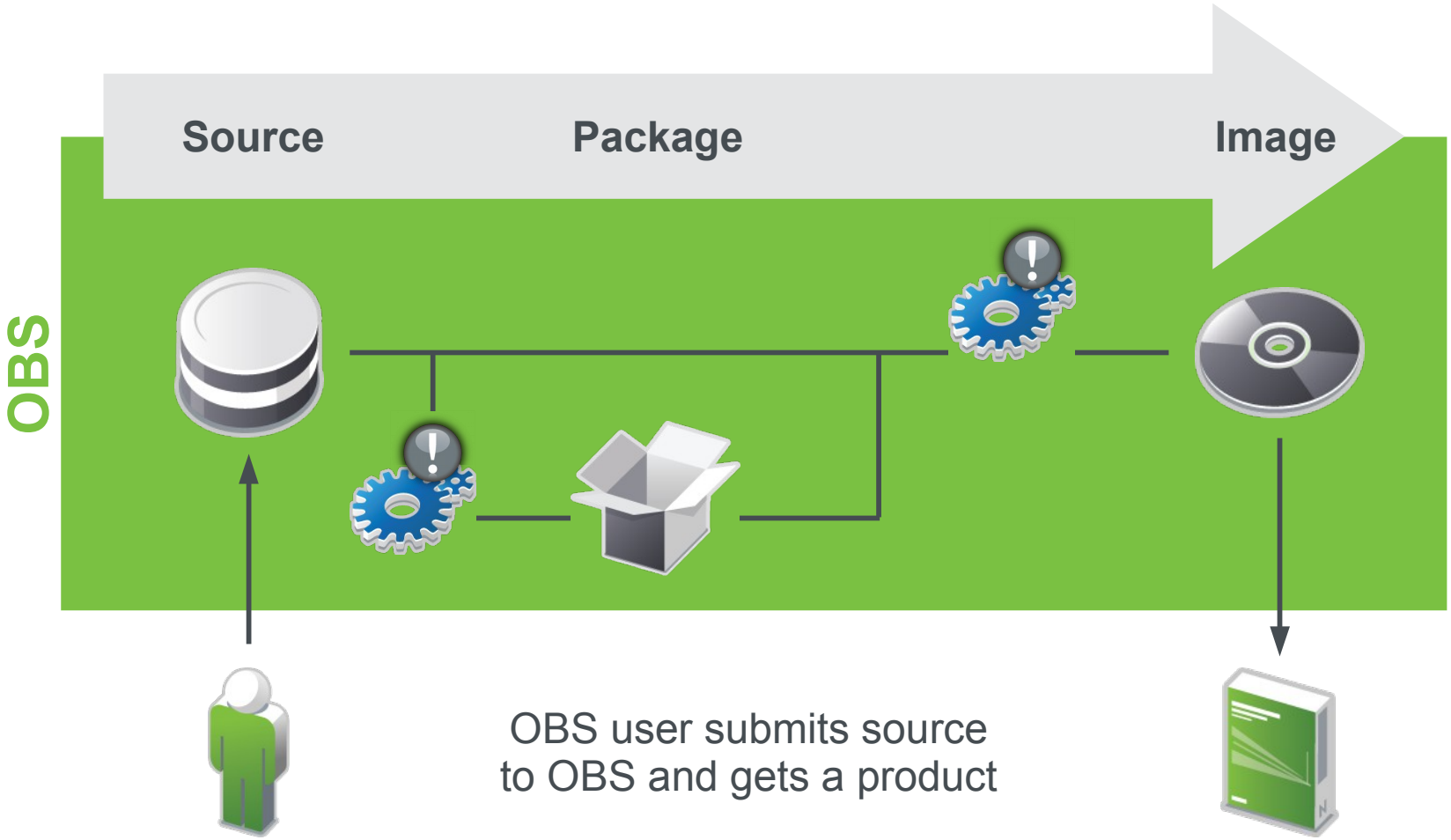
Open Build Service

From Holism to Reductionism

What is Open Build Service?

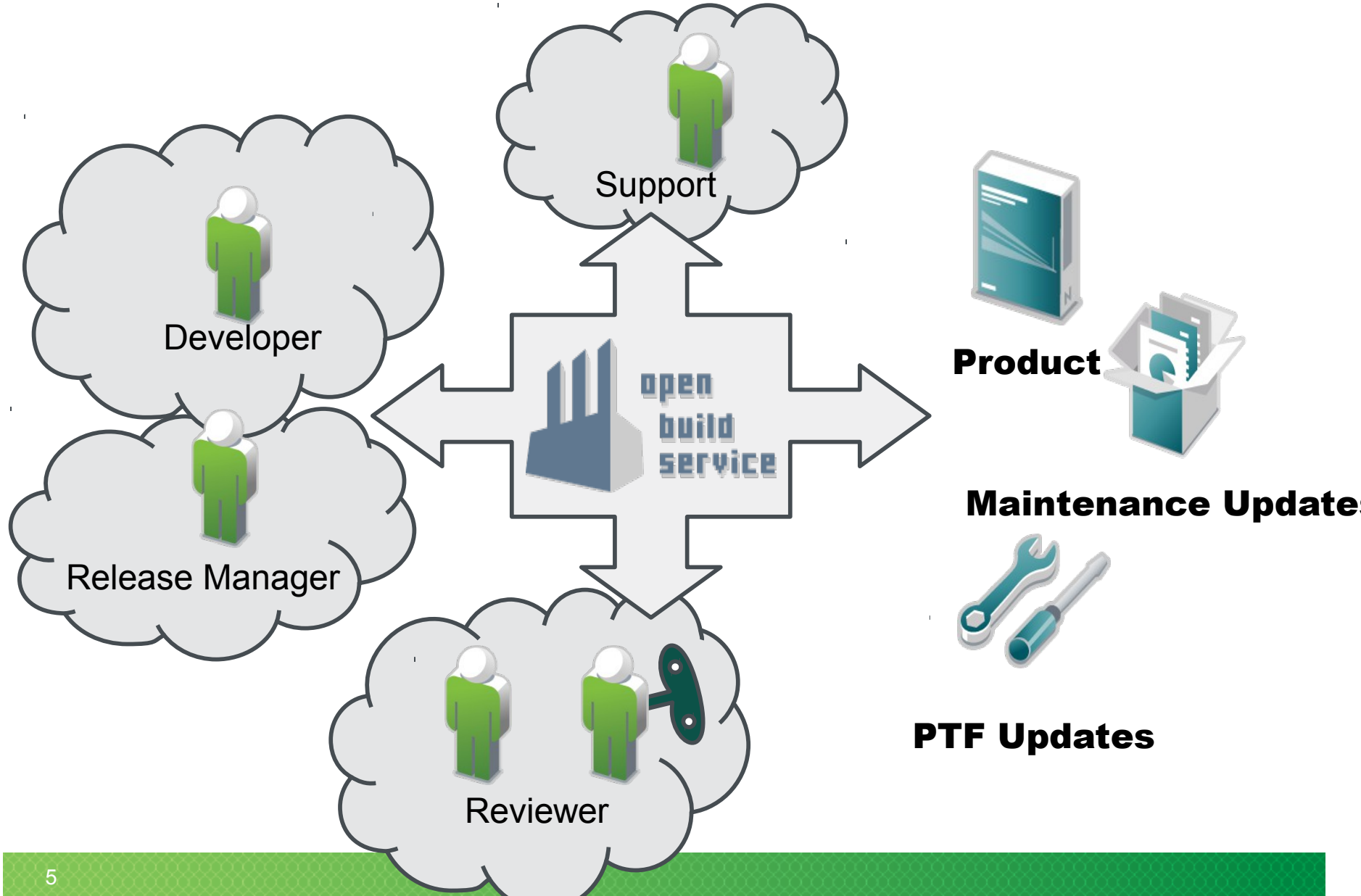
What is the Open Build Service(OBS)?



What Can OBS Create?

- Package repositories
  - Add-on packages
  - Entire distributions
  - Variations of packages or entire products
- Installable Products
- Appliances
- Maintenance updates

OBS Inside of SUSE



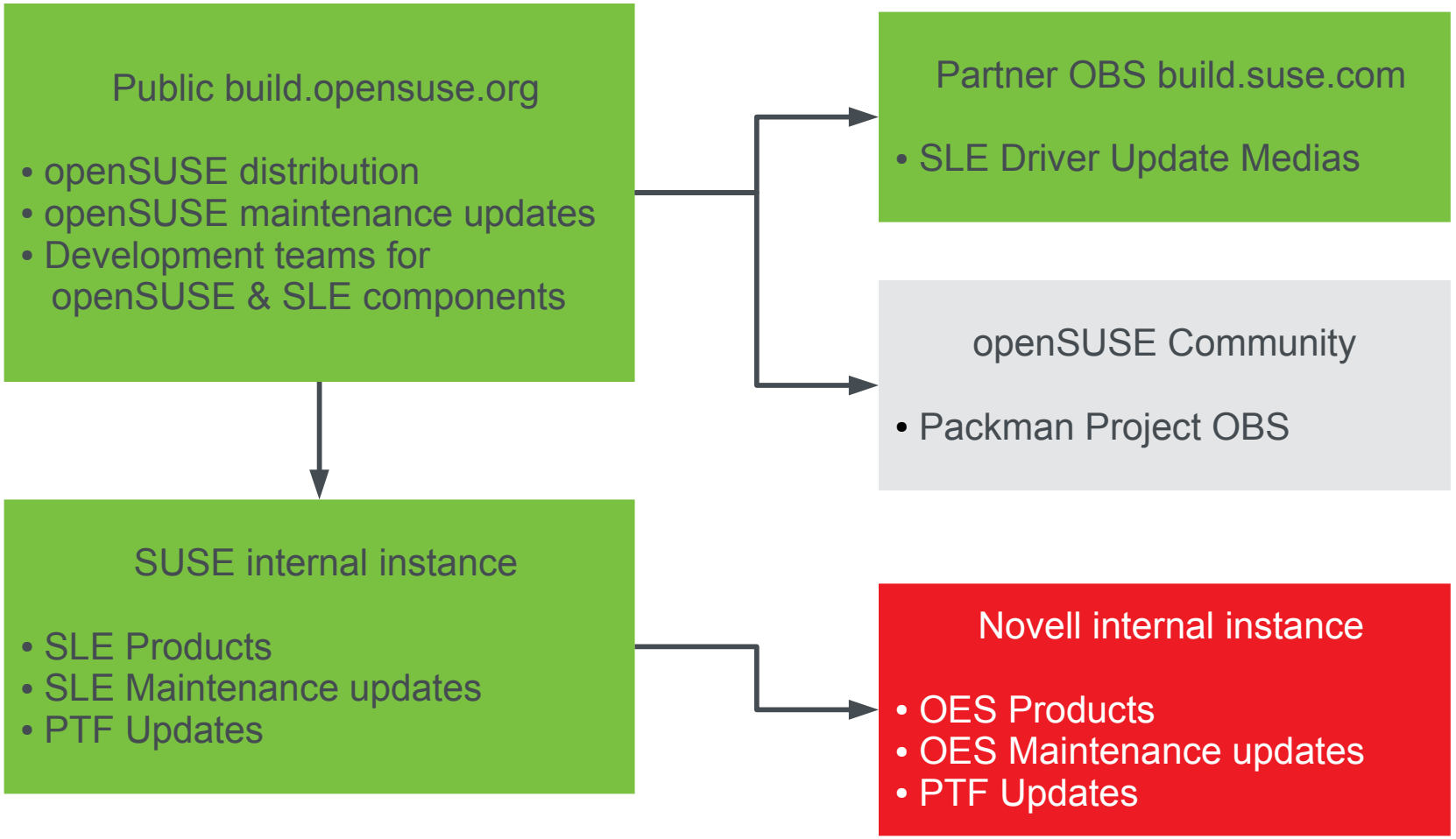
What is Supported by OBS?

- Build formats
  - rpm (spec)
  - deb (dsc)
  - kiwi (product & appliances)
  - Debian Livebuild
  - ArchLinux
- Build process features
  - Build in chroot, lxc, XEN or KVM (experimental: cloud)
  - Architectures: ia32, ia64, x86_64, ppc*, hppa, mips, m68k, s390*, various Arm architectures
    - Qemu can be used to emulate not existing hardware
  - Repositories: rpm-md, yast, apt, maintenance channels

Faces of the Build Service

- Build Software Packages
  - Always clean (aka reproducable) build from one source
  - Supports SUSE®, Fedora, Mandriva, Debian, Ubuntu, … package building
- Build Products based on packages
  - Respins of official openSUSE or SLE medias
  - Build Add-On medias
  - Build Live ISOs, OEM image, USB, XEN, ... media
- Make development workflows transparent
  - Submissions to distributions
  - Run maintenance updates

Where is OBS Used at SUSE®?



Where Else is it Used?

Various large vendors use it for creating their products (Intel, Dell, Cray, many more)

Tizen Project

Used to build the core system

ISVs

Private instances for own add-on products

Build.opensuse.org for open source components

Researchers

Universities

Company research labs

More than 100 other OBS instances connect to build.opensuse.org permanently

Differentiators to Other Build Systems

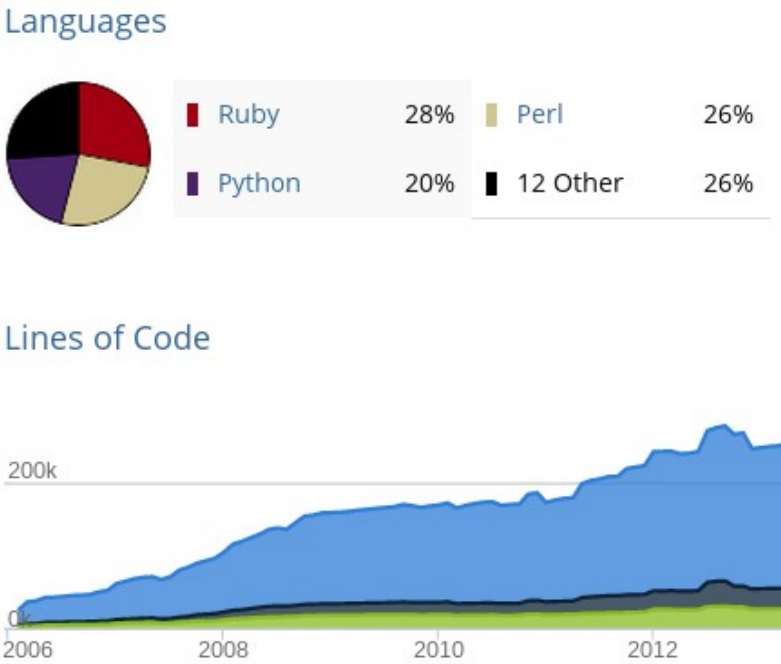
- Supports transient and reproducible builds
- Covers entire distribution handling process
  - From source to images
  - Integrated workflows
  - Partly integrated QA
- Available as open source in an installable form
- Fast prototyping support
- Integrated support for hardware emulated builds
- Stable and open API

OBS History

- Created in 2005 as a rewrite of SUSE's internal autobuild system
  - Goals: transparency, flexibility, openness
  - First presented at FOSDEM 2006
- Current Release: OBS-2.5
  - Further architectures (aarch64, ppc64le)
  - Integrated notification system
  - Support for maintenance updates
- Upcoming OBS 2.6:
  - SLE 12 support
  - Binary tracking support

Development

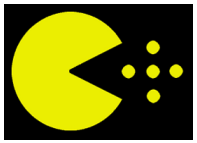
- Licensed under GPL
  - https://github.com/openSUSE/open-build-service/
- Lines of Code: > 200000
  - Perl/Python/Ruby



Users

Distribution development, Maintenance Updates

Open Source Communities

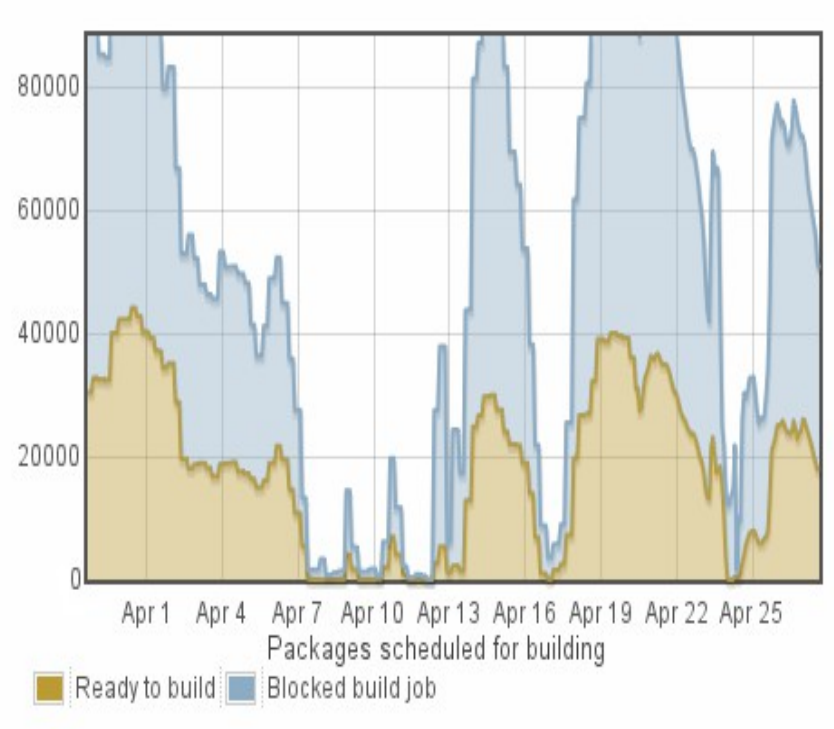



Add-Ons: Driver Developer and ISVs

Researchers/Universities
Administration Teams

Numbers (from build.opensuse.org)

- Confirmed Users: >37000
- Package builds per day: > 75000
  - Build farm: 70 hosts, 440 workers
- Storage:
  - Sources: 8 TBytes
  - Binaries: 19 TBytes



Support

- Community
  - opensuse-buildservice@opensuse.org
  - Irc: #opensuse-buildservice on freenode
- Professional
  - http://www.open-build-service.org/contact/
  - B1 Systems (L3 backing by SUSE)

Server Structure

Parts of OBS

A Server which

- Stores all sources in revision control system
- Stores all binaries
- Calculates the need of rebuilds
- Acts as a database for all kind of queries

Client tools communication via the API

- osc CLI

Optimized for handling sources

Run local builds in same way as on server

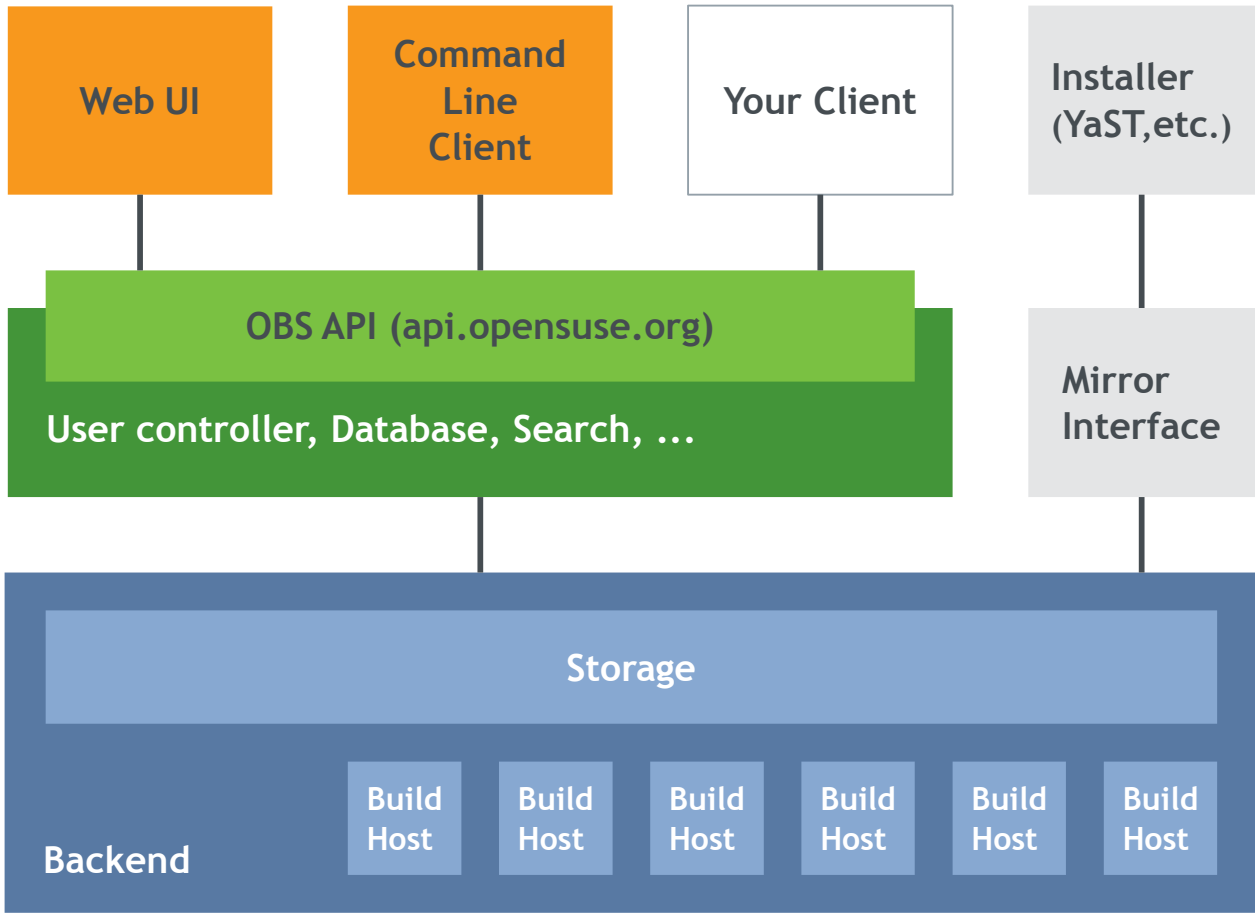
- WEBUI

Optimized to see current status

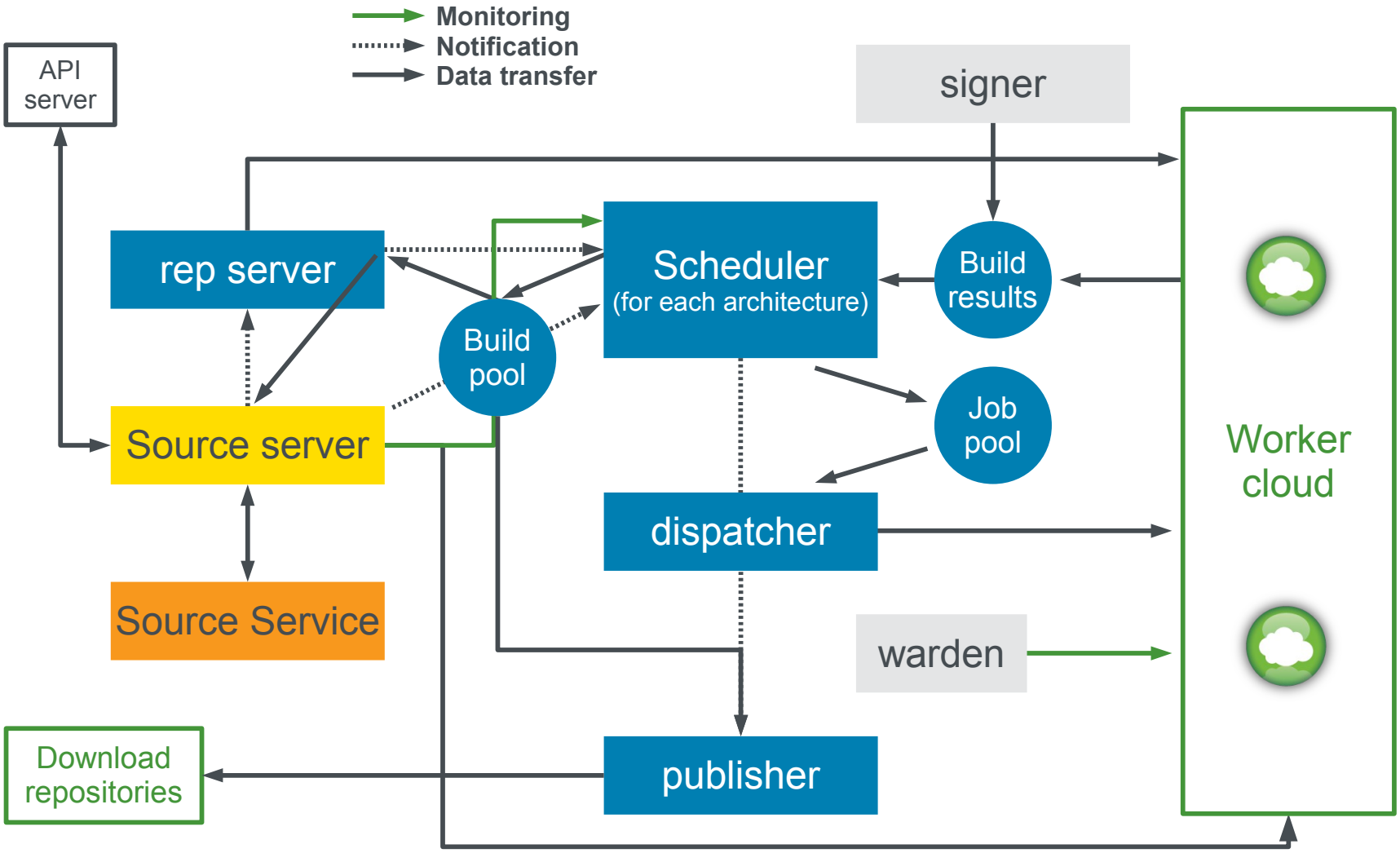
Manage requests

- Various other clients

Server Components



Backend Components



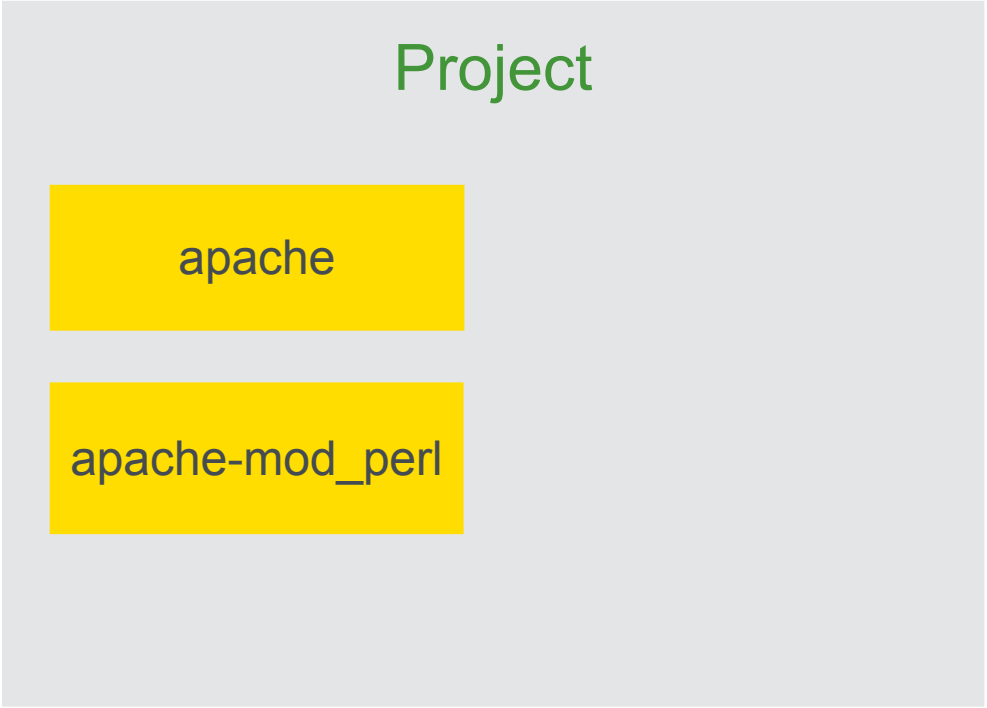
Building Scenarios

Own Set of Packages

- Create project

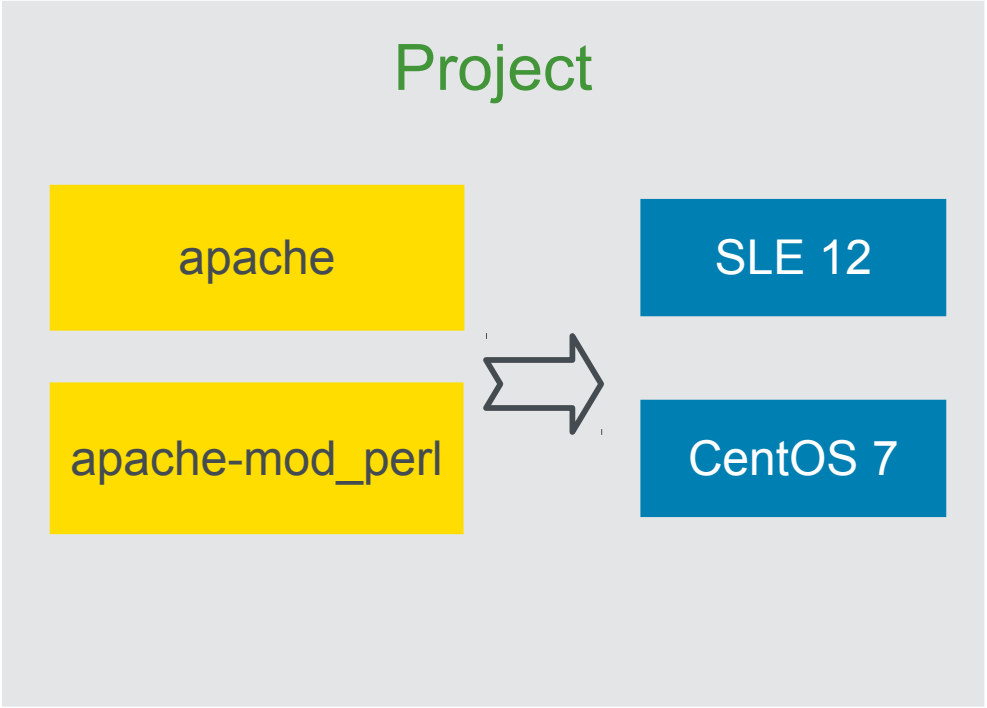
Own Set of Packages

- Create project
- Add package sources



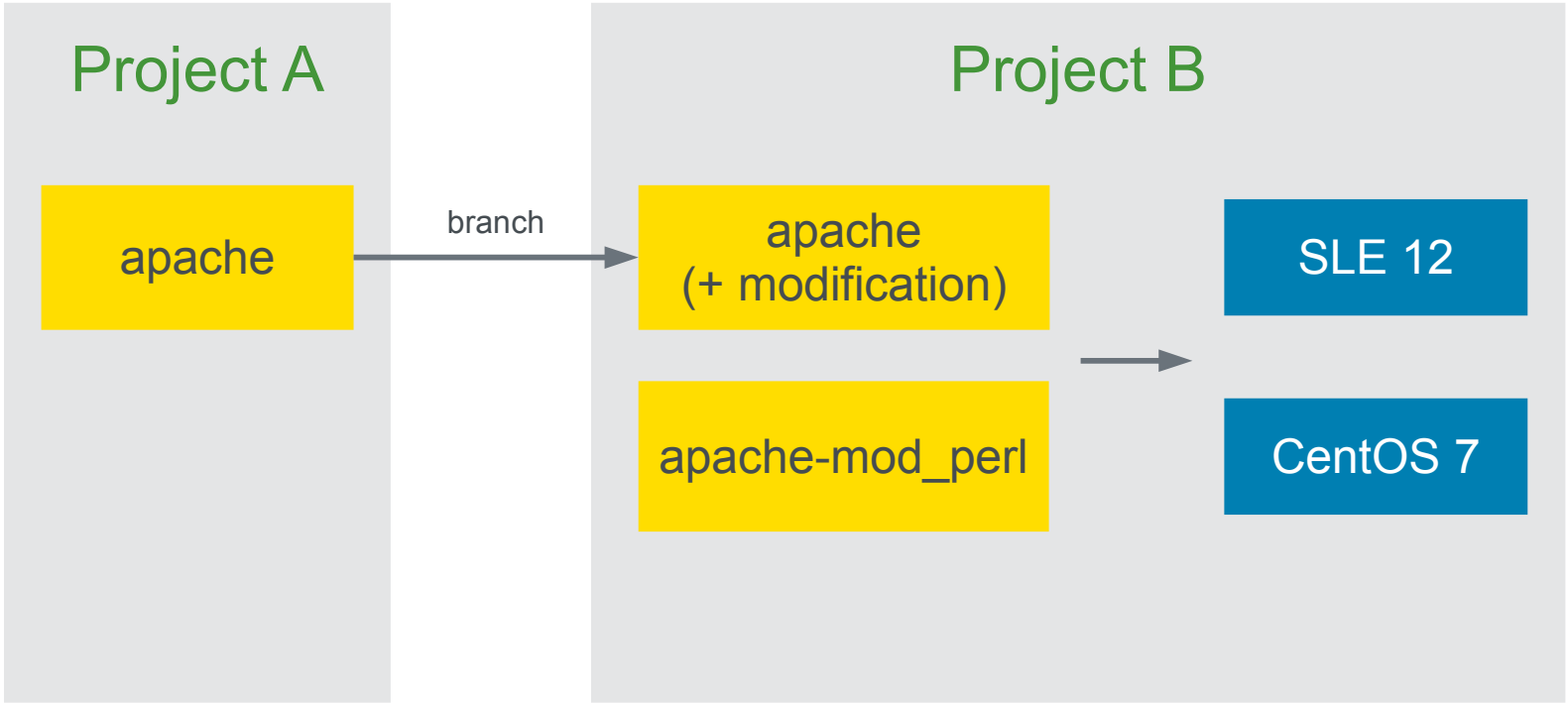
Own Set of Packages

- Create project
- Add packages
- Add targets



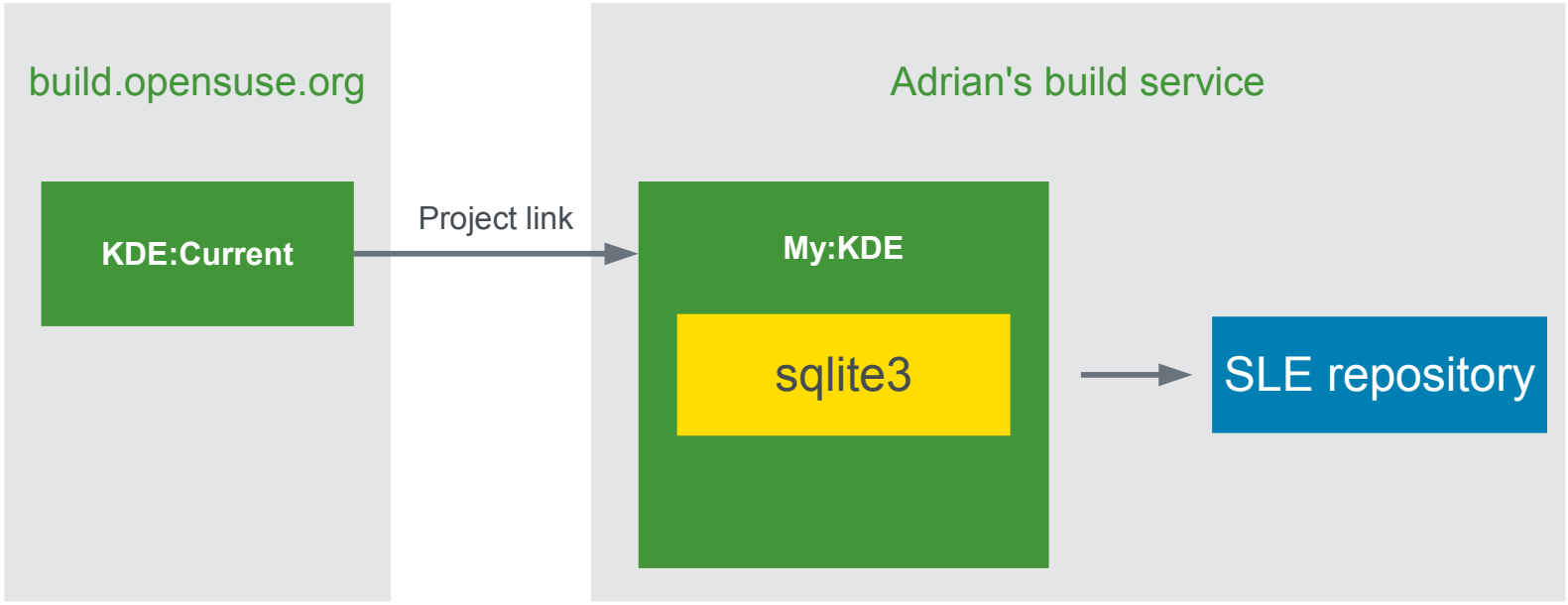
A Variation of a Package

- Branch a package. Can be from a remote instance.



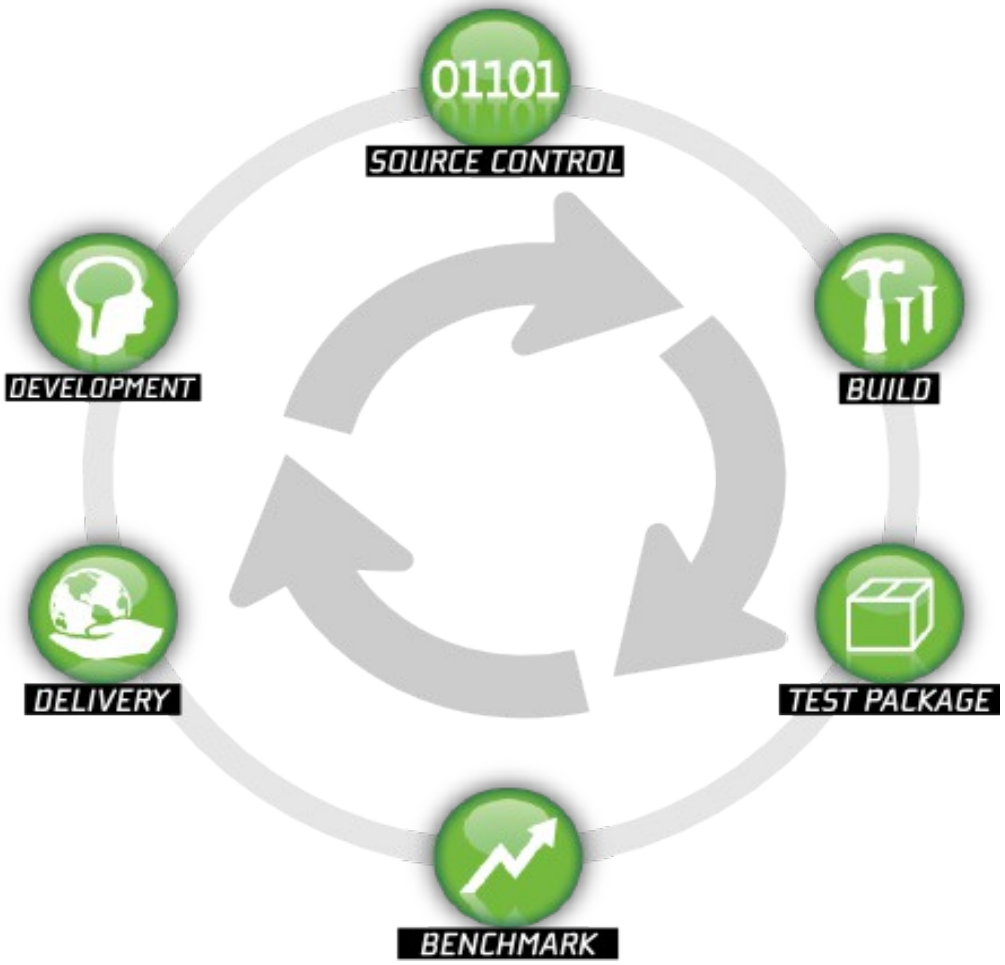
Rebuild a Part of Entire Project

- Link a project
- replace a package source
- Configure repository to rebuild all packages affected by this package



How to Do Cont. Delivery with OBS

Continuous Delivery is Expected



Source Update Variant #1

- Write own script and submit it frequently

```
# bash my_source_update_script.sh
# osc addremove
# osc ci -m “update sources”
```

Source Update Variant #2

- Define tar ball updates via source service

One time

- Write spec file

osc add git://....

Update it via

osc service remoterun <PROJECT> <PACKAGE>

or simple HTTP api call

Integrate External Systems

Avoid usage of your credentials:

- Create an authorization token

  # osc token –create [<PROJECT> <PACKAGE>]

=> Update sources via api call using the token

  # osc token –trigger <TOKEN>

=> Used in OBS support of github.com

Unit Tests

- Package specific test cases in %check

For Benchmarking

- Ensure to build on a reliable host via build constraints
- Use build times as high level benchmark
- Export benchmark details as build results

Benchmarking

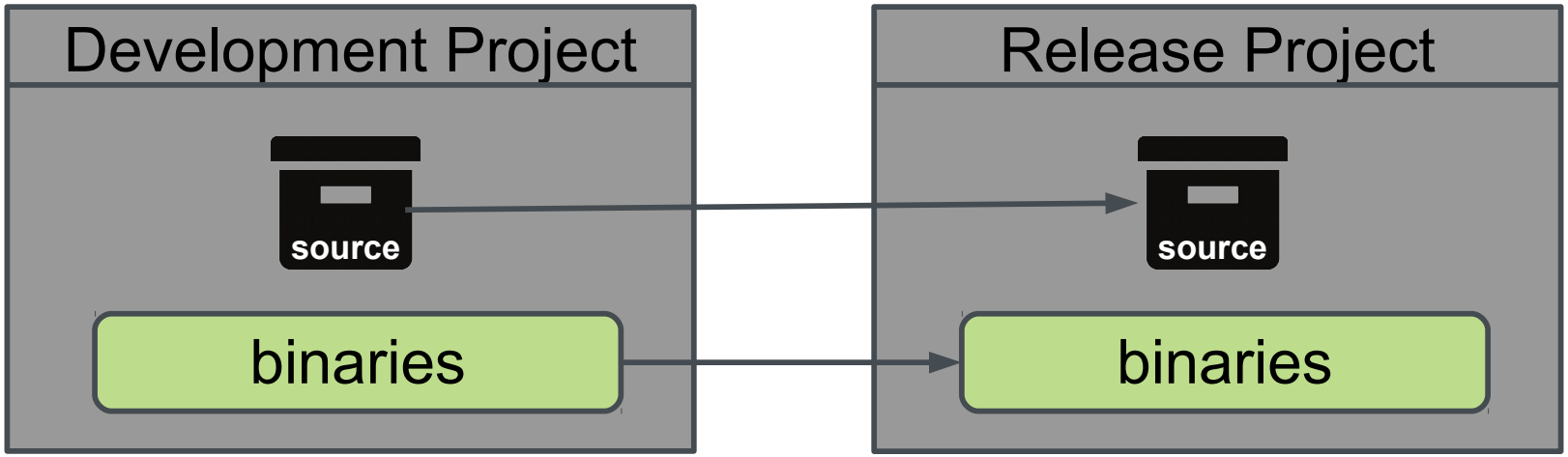
- Ensure to build on a reliable host via build constraints
- Export benchmark results as extra files
- Use build times as benchmarks

How to Deliver

- Successfully tested builds can be released to another repository:

Define releasetarget in repository

osc release <PROJECT>



How We Test atm

- Unit tests
  - Package specific test cases in %check
- Integration tests
  - Packages with BuildRequires to be tested packages
- Functional tests
  - openqa.opensuse.org: Used for automatic installation run for openSUSE installation medias
  - Appliances to test applications

Current Problems

- Unit test short comings
  - QA checks increases build time
  - Failed QA check drops build package
- Integration tests inside of OBS are not tracked
  - Extra test package results are not connected
- Functional tests with OBS lacks integration
  - External QA instances like openQA can not deliver results
  - Appliances can not be executed (easily)

Cool Projects

SLE 12 Community Project

openSUSE:CPE:SLE-12

- Collects add-on packages for SLE 12
- No conflicts with official SLE 12 packages
- Sources come from openSUSE stream

Ports

openSUSE:Factory:ARM
openSUSE:Factory:PowerPC

- Rebuilds everything for unofficial architectures
- Include sometimes small modifications, before merged back

MS-Windows

windows:mingw:win32
windows:mingw:win64

- Cross-Building OSS software for MS-Windows

openSUSE Maintenance

openSUSE:Maintenance

- Shows past and upcoming maintenance updates
- You can use the OBS search to find updates for given bugzilla or CVE entries.

openSUSE Staging

openSUSE:Factory:Staging:*

- Interesting example how to setup a QA setup before releasing
- Allows openSUSE:Factory to become a rolling release

ISV Area

isv:*

- Used by ISV's to ship their open source variant
- For example:

Dell & Microsoft (drivers)

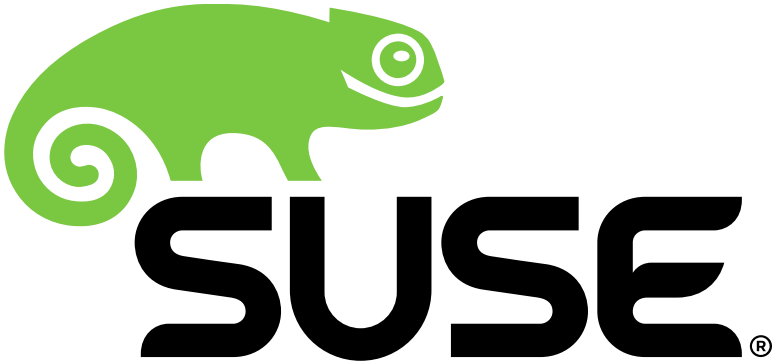
SUSE:Cloud and Owncloud (cloud solutions)

Spb (Brazilian government software)

- Lot's of ISV's still elsewhere, eg. server:OX

Plenty of Optional Repos

- Apache
- Application:Geo
- Graphics
- KDE & GNOME
- devel:languages: (ruby, perl, …)
- Education
- Emulators
- Games
- Kernel:stable


SUSE®
We adapt. You succeed.

Demonstration

Getting Started

- Http://build.opensuse.org
- How to get an account
- Starting home project
- Overview of “projects”, “packages”, etc.

Use the OBS Work-Flows

Request System

Every project in OBS has a configured list of maintainers. Others need to use requests to apply changes there.

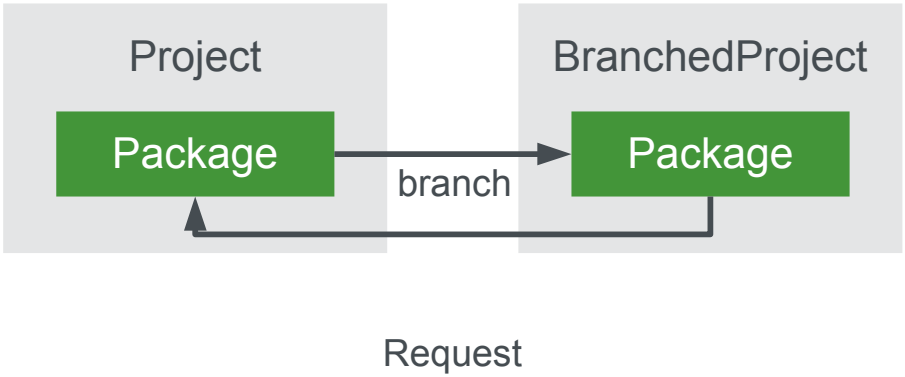
Requests can be used to

- Modify sources
- Add or remove packages
- Request write permissions
- Run an official maintenance update

Change Source in a Package

A typical simple work-flow for modifying a different package is:

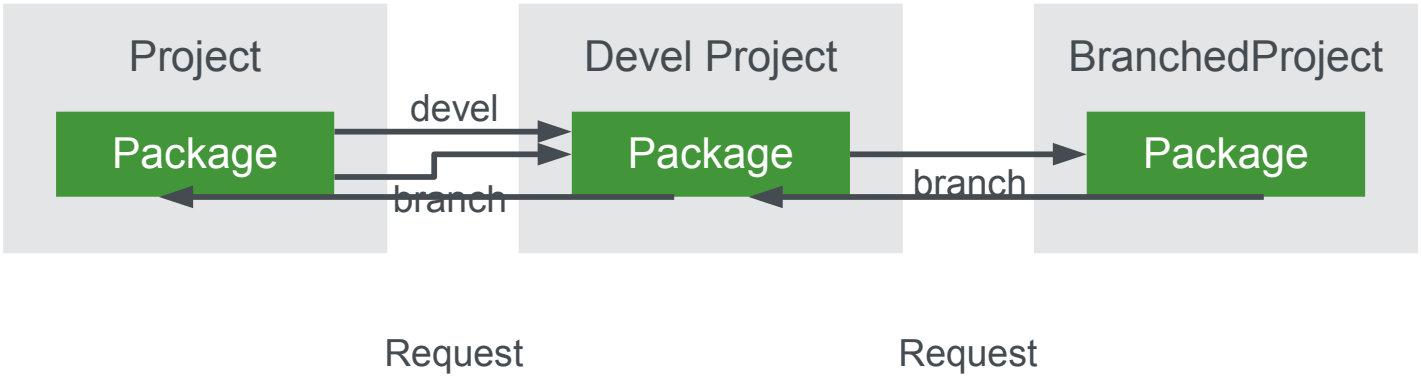
- Find the package
- Branch the package
- Modify the package
- Let OBS build it
- Test it
- Request the merge
- Modification gets accepted
- Usually the branched project gets removed automatically



Handle Large Projects

Large projects like openSUSE:Factory with many contributors have projects where software stacks get managed:

- A package may define another source instance which should be used as stage area (devel project).
- Modifications can be tested there before breaking the large project.
- The maintainers can manage themselves in the devel project
- Submissions can be coordinated



Use the Review System

Modifications may need to be reviewed by multiple groups.
A request stays in the state “review” until all reviews got accepted.
The request gets declined when a reviewer declines it.

- Reviewers can be added manually to any request
- Default reviewers can configured in a project or package
- Reviewers can be

Single person

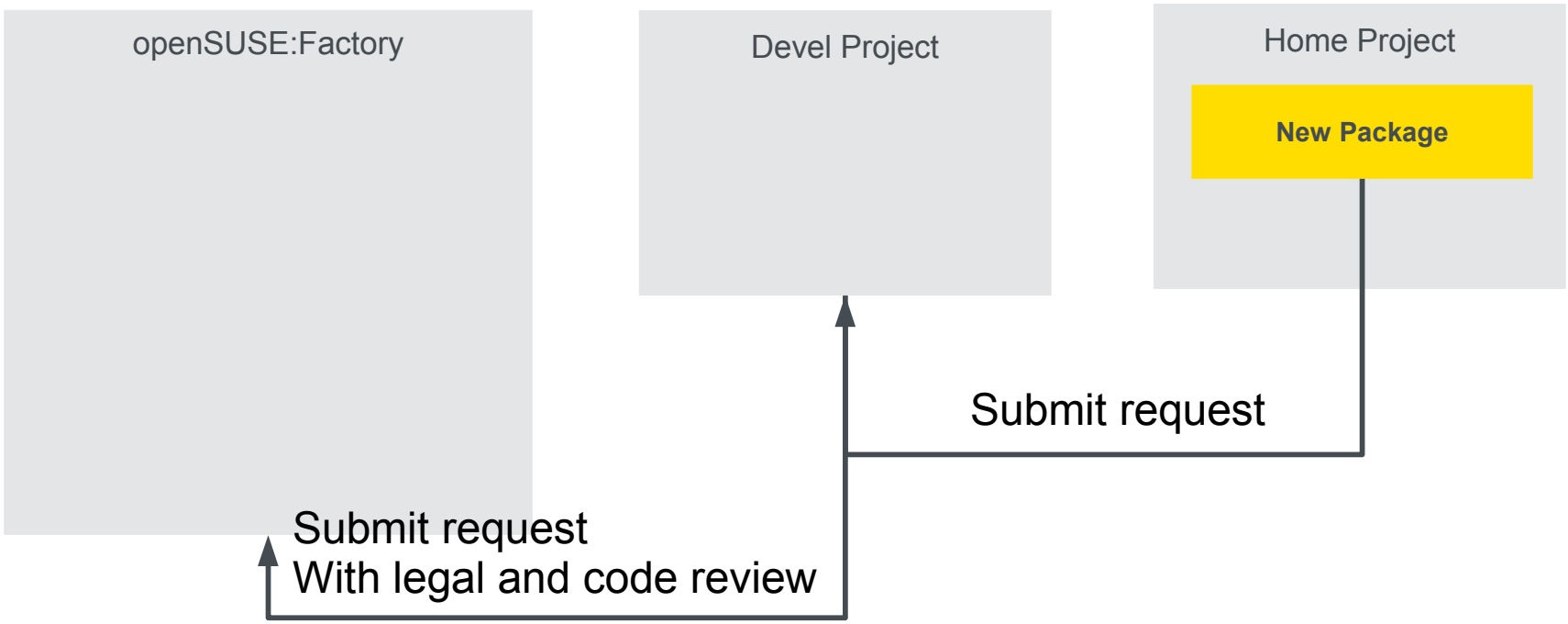
An external script to apply standard checks

A group

Maintainers of another project or package

Example:

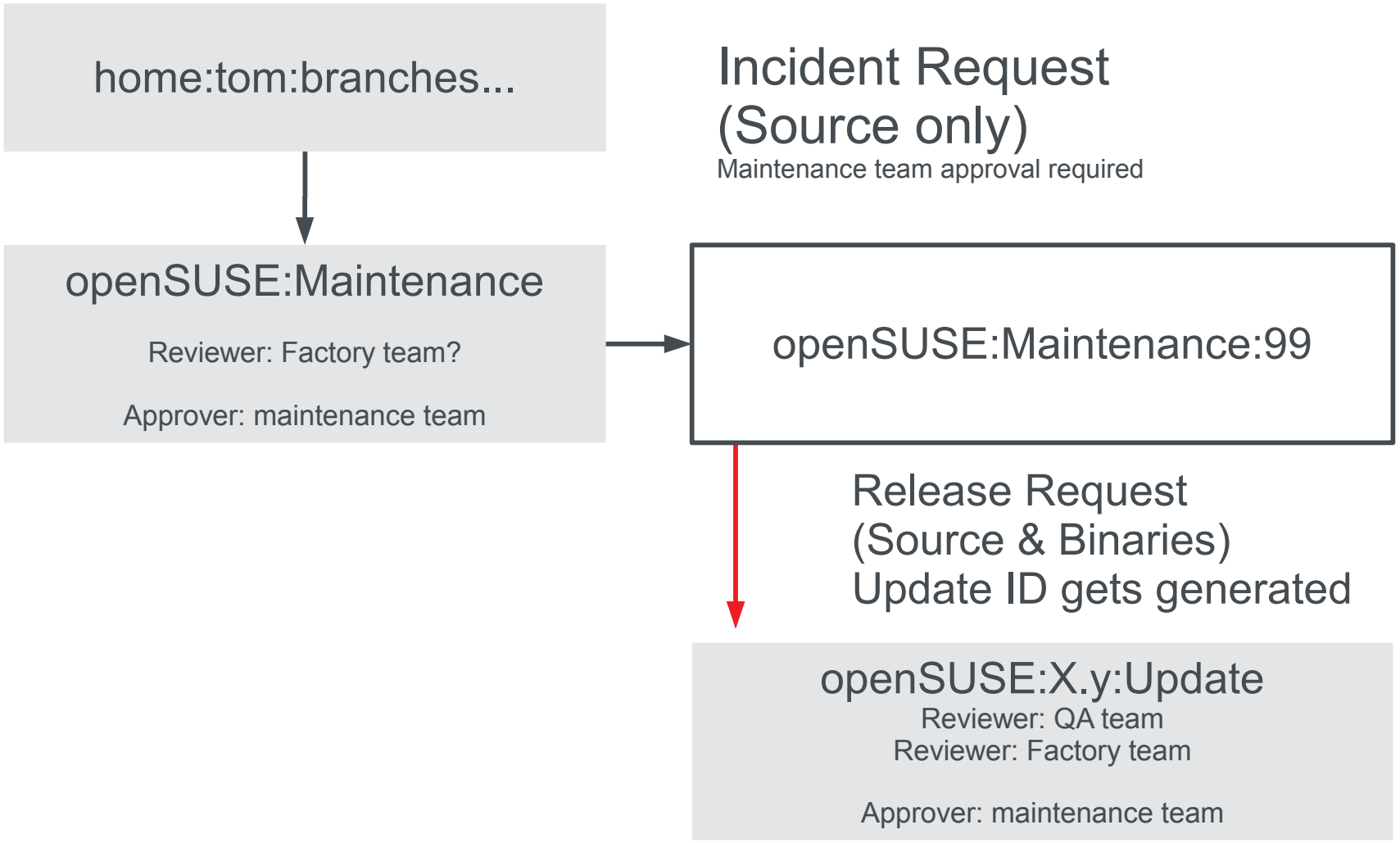
Add New Package to openSUSE®



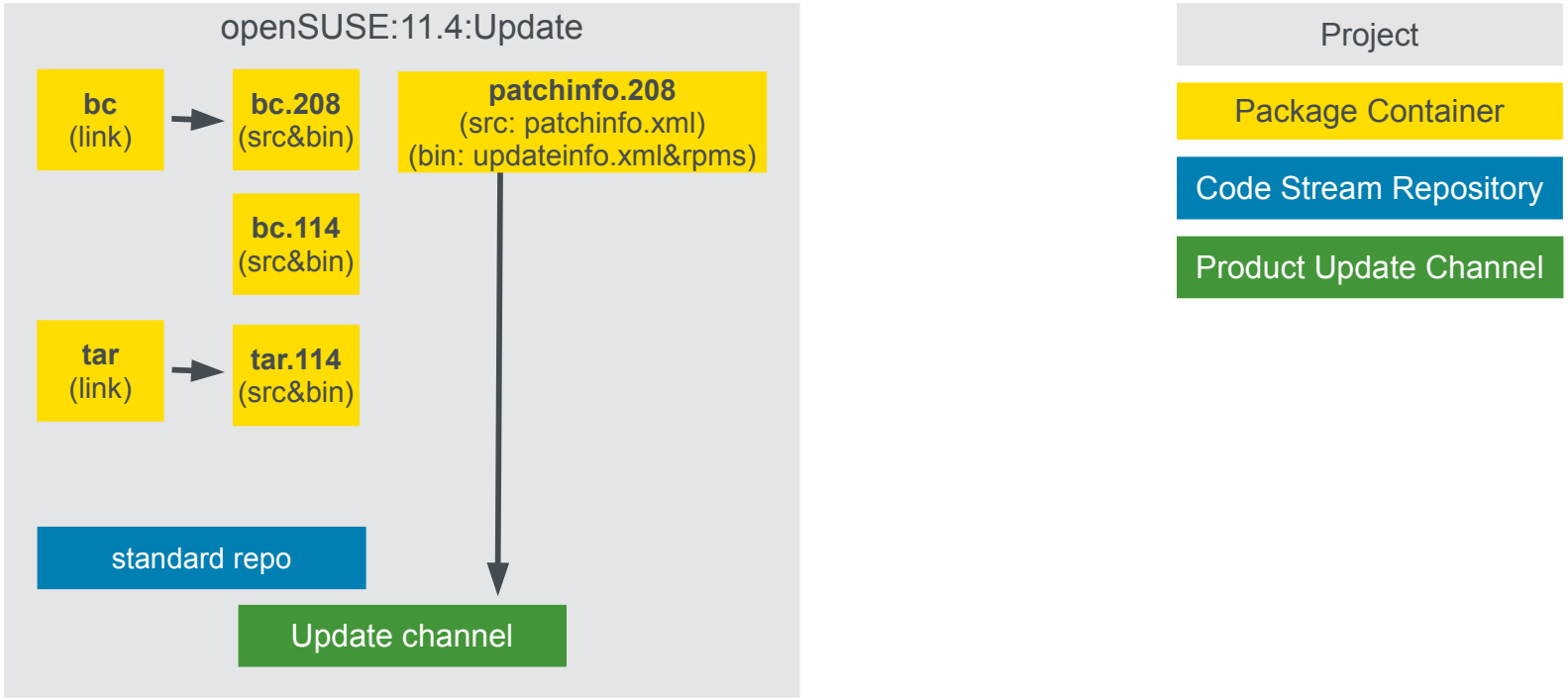
openSUSE:Factory has the policy that every package must be tested in a devel project before submitting to Factory.

Example:

Make an Official Update



openSUSE Update Project Layout



114 & 208 are incident numbers.
114: one incident for bc & tar package
208: one incident for bc package

Build Products Automatically

Build an Appliance with Kiwi

- The simplest product is a live appliance
- Add kiwi images repository
- Build happens by adding a .kiwi file as source
  - Can be taken from example build.opensuse.org
  - Can be taken from Studio or kiwi templates, but requires following modifications:
    - Repositories must be referenced via obs://$PROJECT/$REPOSITORY

Build an Add-on Product

- Add kiwi images repository
- Create a “_product” package
- Supply a product description

Xml files as specified here:

All required _product:* packages get generated automatically and build started

Automate Source Processing

Source Service System

- The source service system allows to
  - Enforce source processing on server side
  - Run locally
    - By default on any action
    - Optionally, eg. For doing a version update explicit in a transparent way
- Can be used for example to
  - Import sources in a documented and reproducable way
  - Generate build descriptions
  - Run validation checks
  - Modify sources to apply standards and policies (eg. Copyright header)

Import Sources

Download a file, esp. useful for tar balls to track the origin:

- URL specified in service
- URL specified in spec file, validates that remote file has same content

Checkout from git/svn/cvs/..:

- Checkout from given URL via “tar_scm” service
- Compress tar ball via “compress” service
- Update spec/dsc file to match file name via “set_version”

One API call is enough after wards to update the sources and rebuild package.

Validate Sources

To validate sources use project wide source services:

- Validate tar balls via “download_files” service
- Check policies eg via the SUSE® “source_validator” service
- Give the spec files a common format via “format_spec_file” service

Write Your Own Services

A source service is

- A rpm package called “obs-service-$NAME” which provides
  - /usr/lib/obs/service/$NAME
    executable which gets started
  - /usr/lib/obs/service/$NAME.service
    XML file which describes this service, the required and optional parameters

Future

SCM Integration

- OBS will be able to host git trees
  - Tar balls will be created during build time only
  - Reducing storage and bandwidth needs

QA Support

Integrate with existing QA frameworks (not re-invent them)

- OBS handle build and QA results independent
- Offer interface to request and submit test results

- Auto-apply of test cases affected by component
- Auto-branch and test runs for github.com pull requests

- Multi-instance network setup test cases