The gory details of the build service backend

Michael Schröder
Novell, Inc

Novell

Outline

Data management
- storing projects and packages
- the source repository

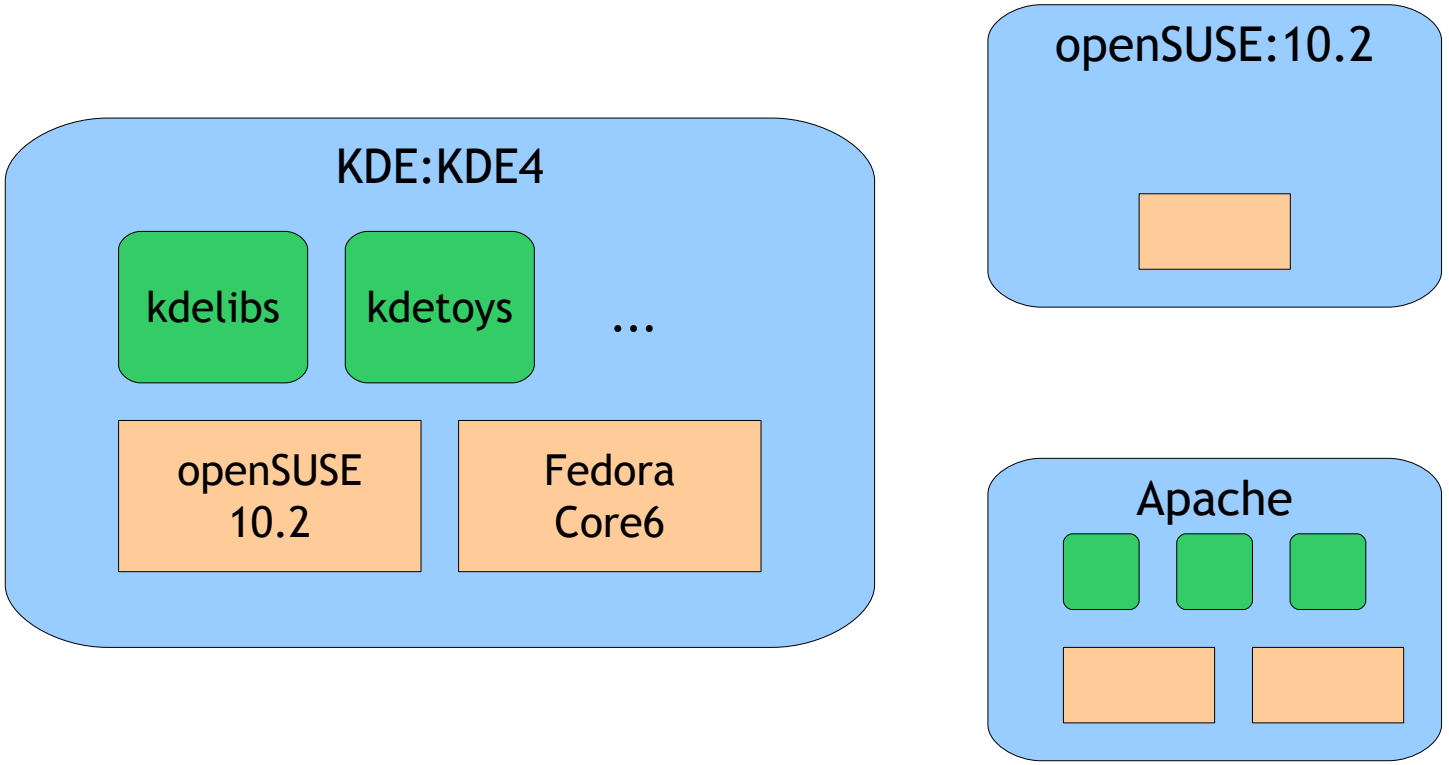
Server framework
- data flow client to the backend
- example: listing package files

Building packages
- from scheduler to build host and back
- scalability issues

Automatic rebuild triggering
- package meta files

Backend basics

Projects, packages and repositories



Novell

How projects are stored

A project consists of

- a project name
- project meta data (summary, description, repositories...)
- project build configuration (setup information, preferred binary packages, rpm macros...)
- multiple packages

```
projects/home:mlschroe.xml
        /home:mlschroe.conf  (if not empty)
        /home:mlschroe.pkg/
```

How packages are stored

A package consists of

- a package name
- package meta data (summary, description, ...)
- package revision log

```
/projects/home:mlschroe.pkg/
    screen.xml
    screen.rev
```

Revision log format (last line is latest revision):

```
12|76|f6d2e7a398fdbd801d7ddc18eaa76e2d|4.0.2|1155842871|mlschroe|
Revision                                version
   Release number                             checkin time
      Source repository identifier                        checkin user
```

The source repository

Manages sources of packages

- allows retrieval of old versions
- switching to new source revisions must be atomic
- identical files in multiple projects and revisions should be shared, i.e. not take extra disk space

Implementation

- files are prefixed with the md5sum value of their content
- a revision is identified by the md5sum over a MD5SUMS file
- nothing gets deleted, just new files added

The source repository (cont.)

Example: the screen package

```
revision:
    screen_4.0.2-4.1.diff.gz
    screen_4.0.2-4.1.dsc
    screen_4.0.2-4.1.spec
    screen_4.0.2.orig.tar.gz

MD5SUMS:
    8f8725fa9b3385042115e84a06866ce6  screen_4.0.2-4.1.diff.gz
    64276af3d6f9c364528fb49223b995a3  screen_4.0.2-4.1.dsc
    47cc233ceb7ba64bf43807978b52c40a  screen_4.0.2-4.1.spec
    ed68ea9b43d9fba0972cb017a24940a1  screen_4.0.2.orig.tar.gz

md5sum MD5SUMS:
    f6d2e7a398fdbd801d7ddc18eaa76e2d  MD5SUMS
```

The source repository (cont.)

Example: the screen package

```
source/screen/
    8f8725fa9b3385042115e84a06866ce6-screen_4.0.2-4.1.diff.gz
    64276af3d6f9c364528fb49223b995a3-screen_4.0.2-4.1.dsc
    47cc233ceb7ba64bf43807978b52c40a-screen_4.0.2-4.1.spec
    ed68ea9b43d9fba0972cb017a24940a1-screen_4.0.2.orig.tar.gz
    ...
    ...
    f6d2e7a398fdbd801d7ddc18eaa76e2d-MD5SUMS
    ...
```

Revision identifier is MD5SUMS' md5sum:

```
f6d2e7a398fdbd801d7ddc18eaa76e2d
```

The source repository (cont.)

How to retrieve a file from the repository:

Task: read file `screen_4.0.2-4.1.spec` from source revision `f6d2e7a398fdbd801d7ddc18eaa76e2d` package `screen`:

- enter directory `source/screen`
- read `f6d2e7a398fdbd801d7ddc18eaa76e2d-MD5SUMS`
- find md5sum of file `screen_4.0.2-4.1.spec`: `47cc233ceb7ba64bf43807978b52c40a`
- read file `47cc233ceb7ba64bf43807978b52c40a-screen_4.0.2-4.1.spec`

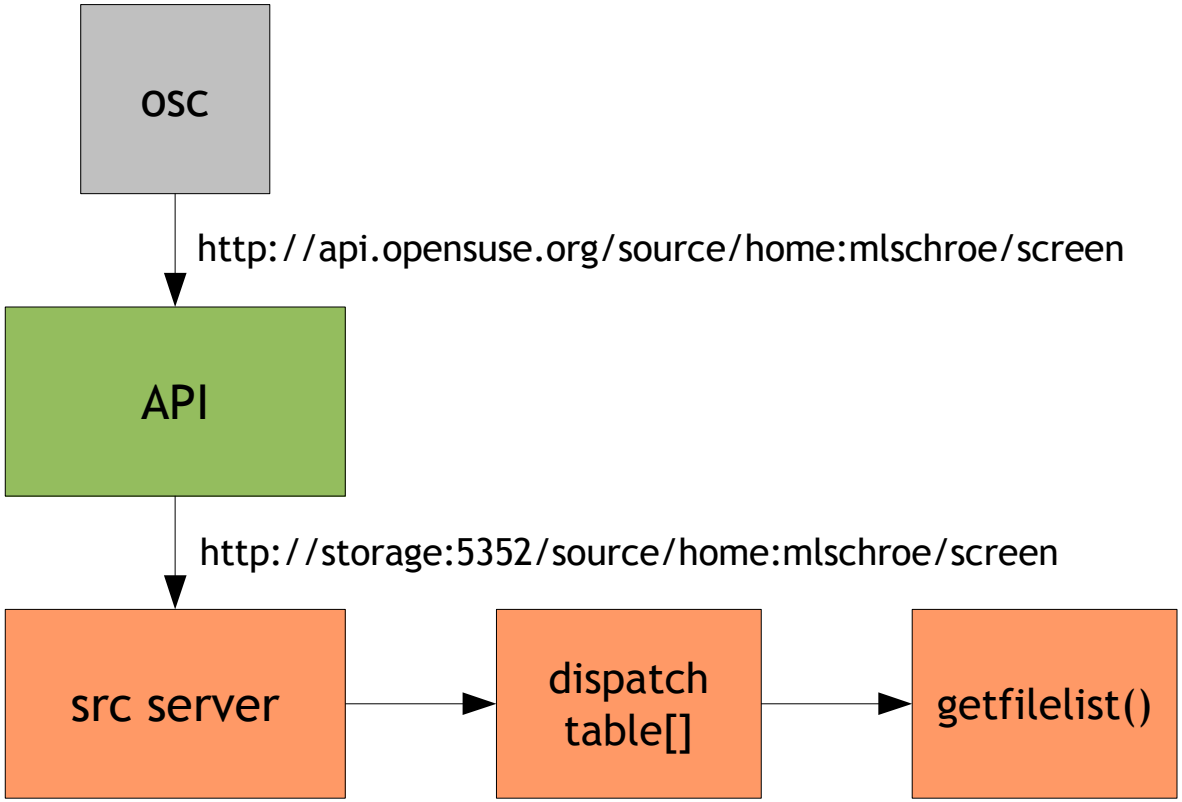
Server Framework

Flexible, extensible http server framework

- written for the build server, perl5
- supports file/cpio streaming
- automatic parameter type checking
- forks a process for every request, but requests taking long time can be passed to a special process
  - used for AJAX updates of build status and for logfile streaming
  - if load gets too high, return “retry after *n* seconds” error

Server Framework (cont.)

Example: getting a source listing



Server Framework (cont.)

Requests are dispatched via a dispatch array:

```
my $dispatches = [ ...
    '/source/$project/$package rev?' => \&getfilelist,
... ];


sub getfilelist {
    my ($cgi, $projid, $packid) = @_;
    my $rev = getrev($projid, $packid, $cgi->{'rev'});
    my $files = lsrep($projid, $packid, $rev->{'srcmd5'});
    my $dir = {'name' => $packid, 'rev' => $rev->{'rev'}};
    ...
    $dir->{'entry'} = \@res;
    return ($dir, $BSXML::dir);
}
```

Server Framework (cont.)

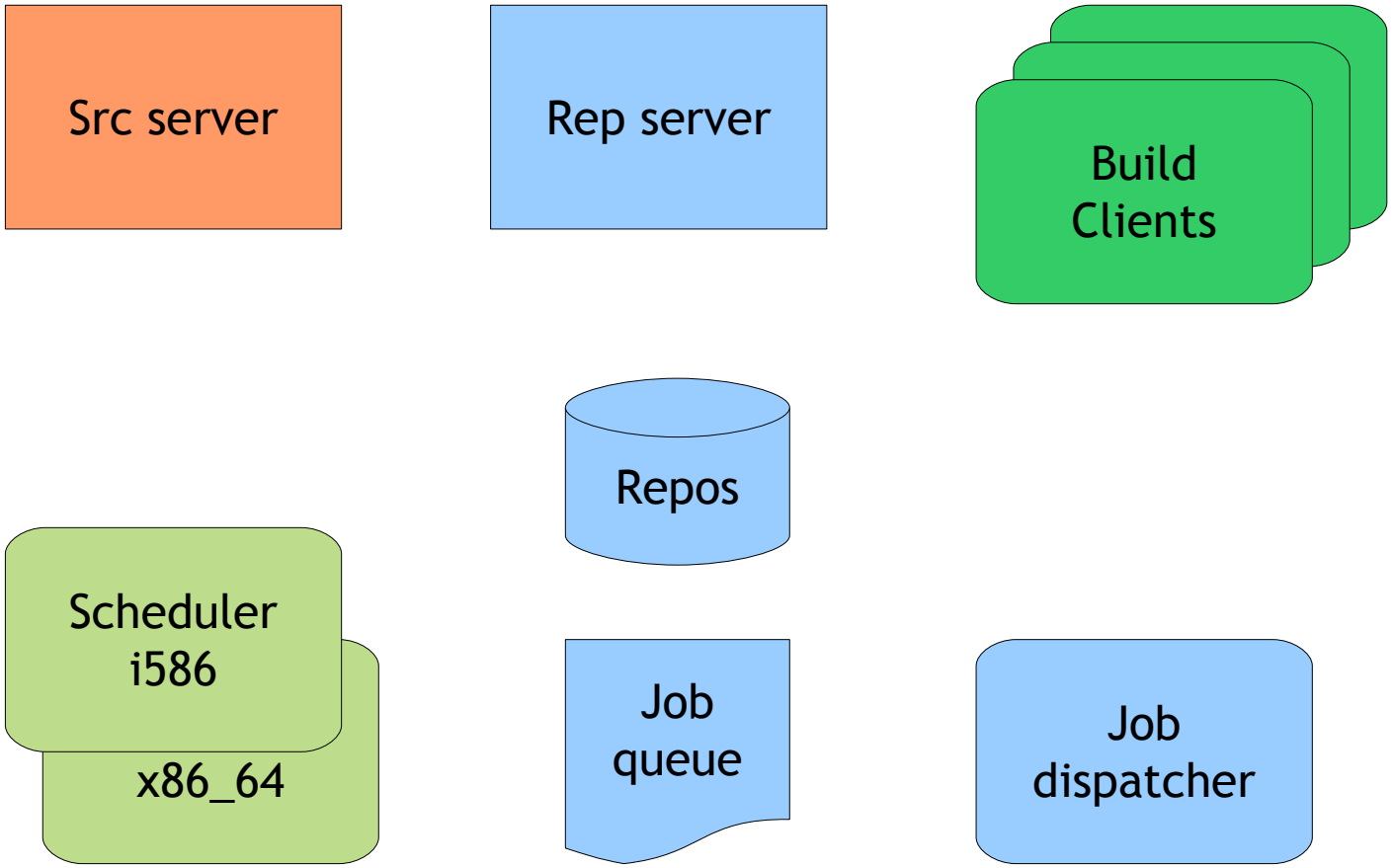
Conversion to XML is done via the XML::Structured module

```
$BSXML::dir = [
  'directory' =>
    'name',
    'rev',
    [ $BSXML::entry ],
];


$BSXML::entry = [
  'entry' =>
    'name',
    'size',
];

$dirxml = XMLout($BSXML::dir, $dir);
$dir = XMLin($BSXML::dir, $dirxml);
```

```
<directory name="screen" rev="12">
  <entry name="screen_4.0.2-4.1.d..."
         size="33462" />
  <entry name="screen_4.0.2-4.1.dsc"
         size="624" />
  <entry name="screen_4.0.2-4.1.spec"
         size="1611" />
  <entry name="screen_4.0.2.orig..."
         size="840519" />
</directory>
```

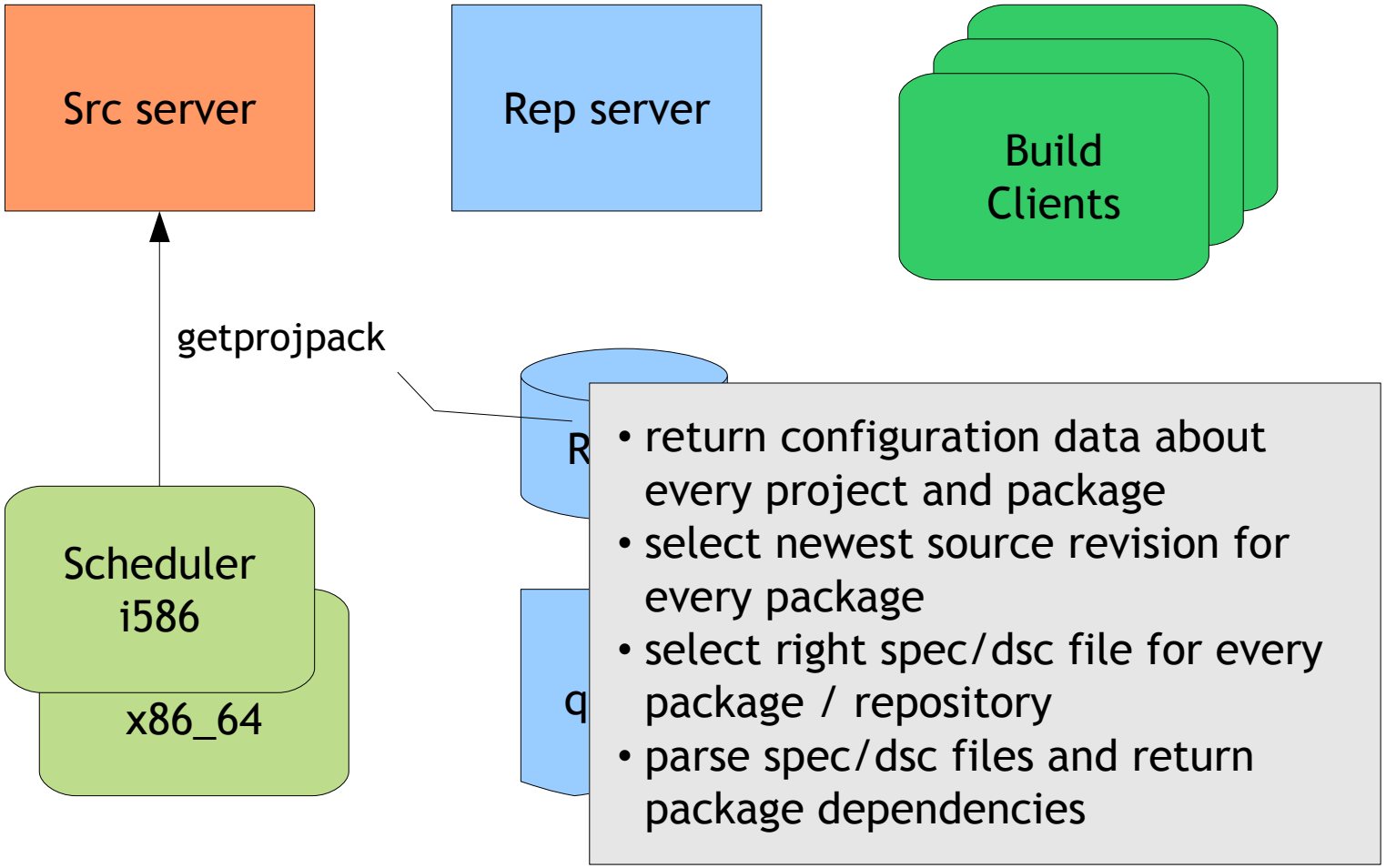

Novell

Backend Architecture



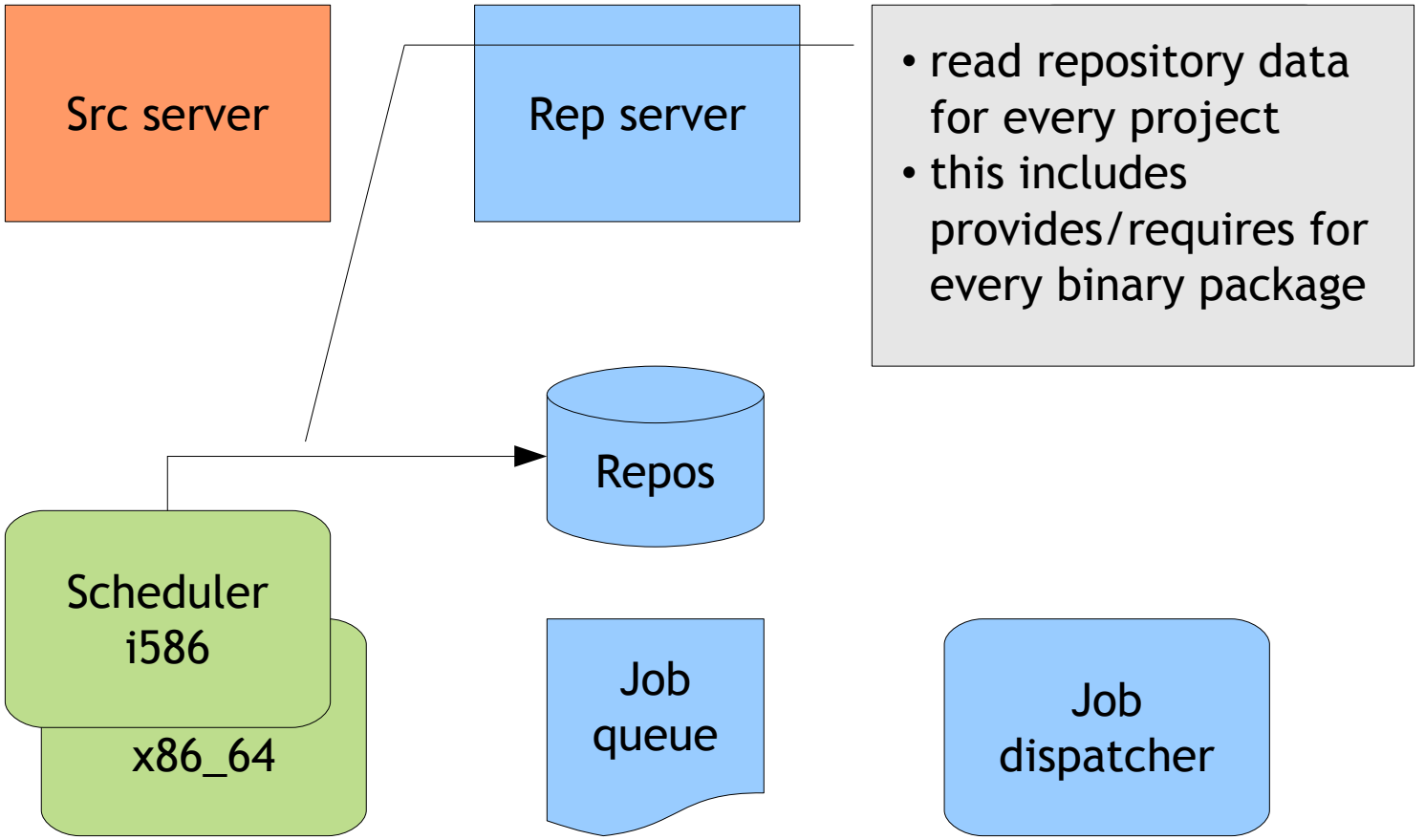
Novell

Backend Architecture

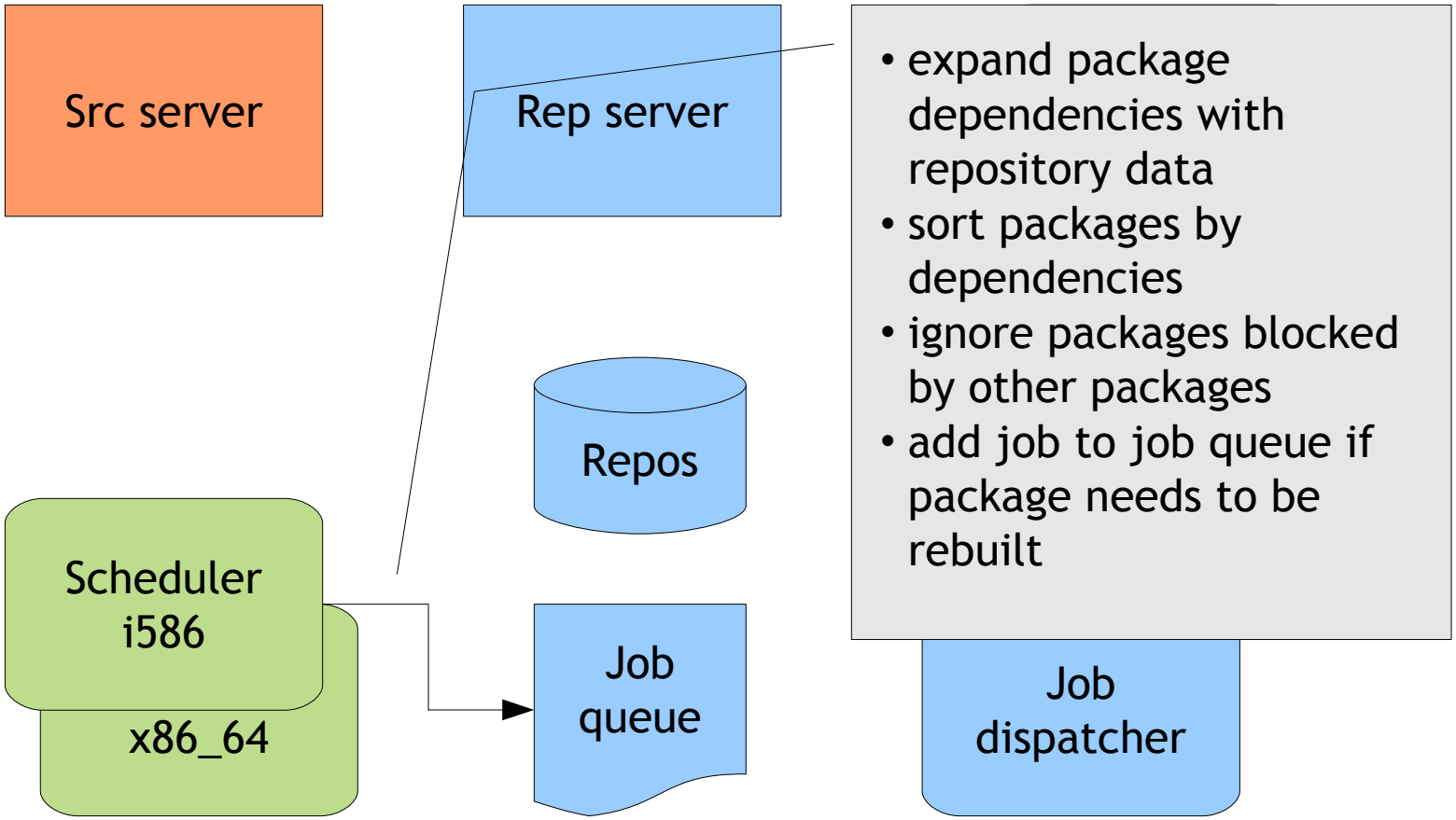


Novell

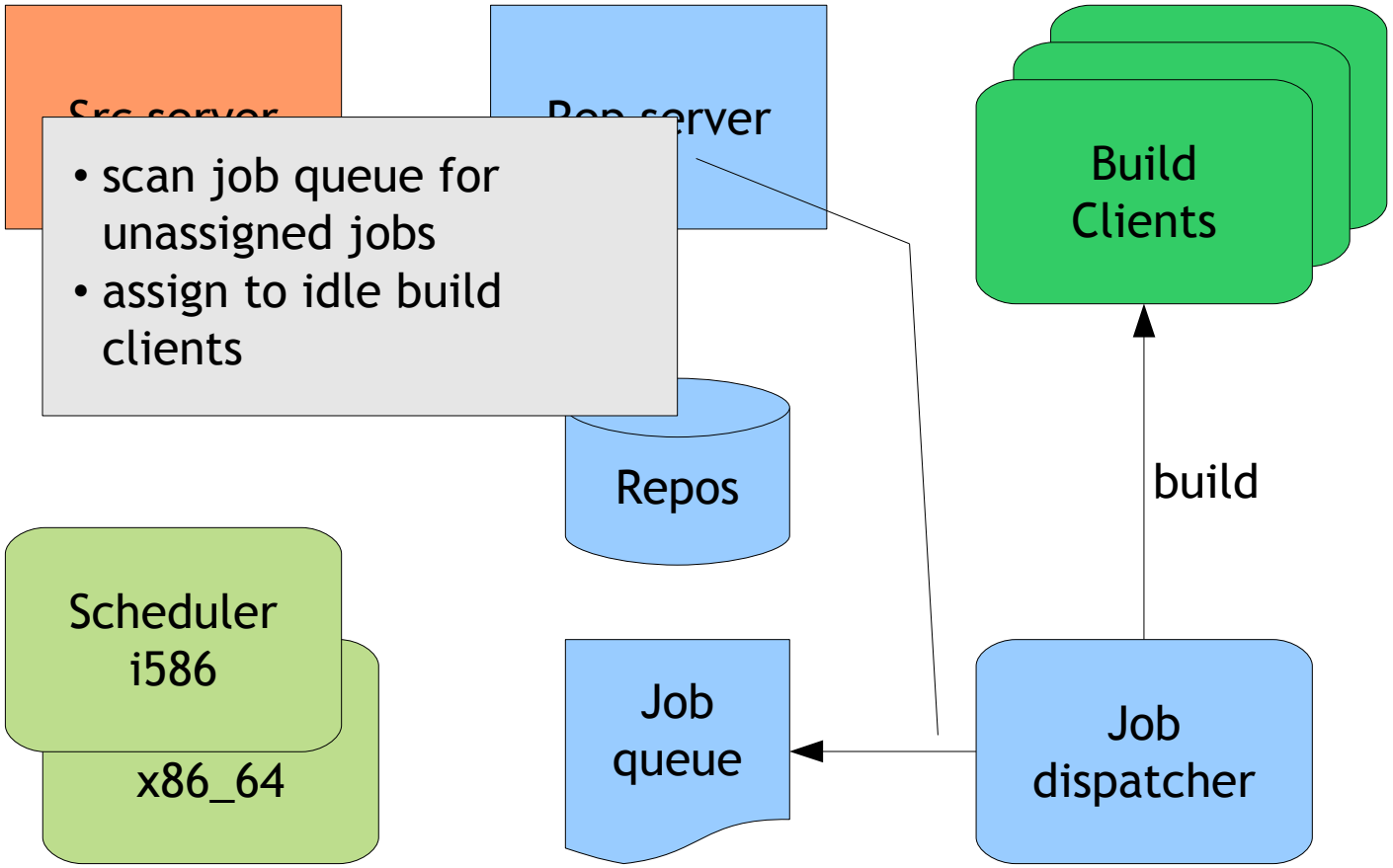
Backend Architecture



Backend Architecture

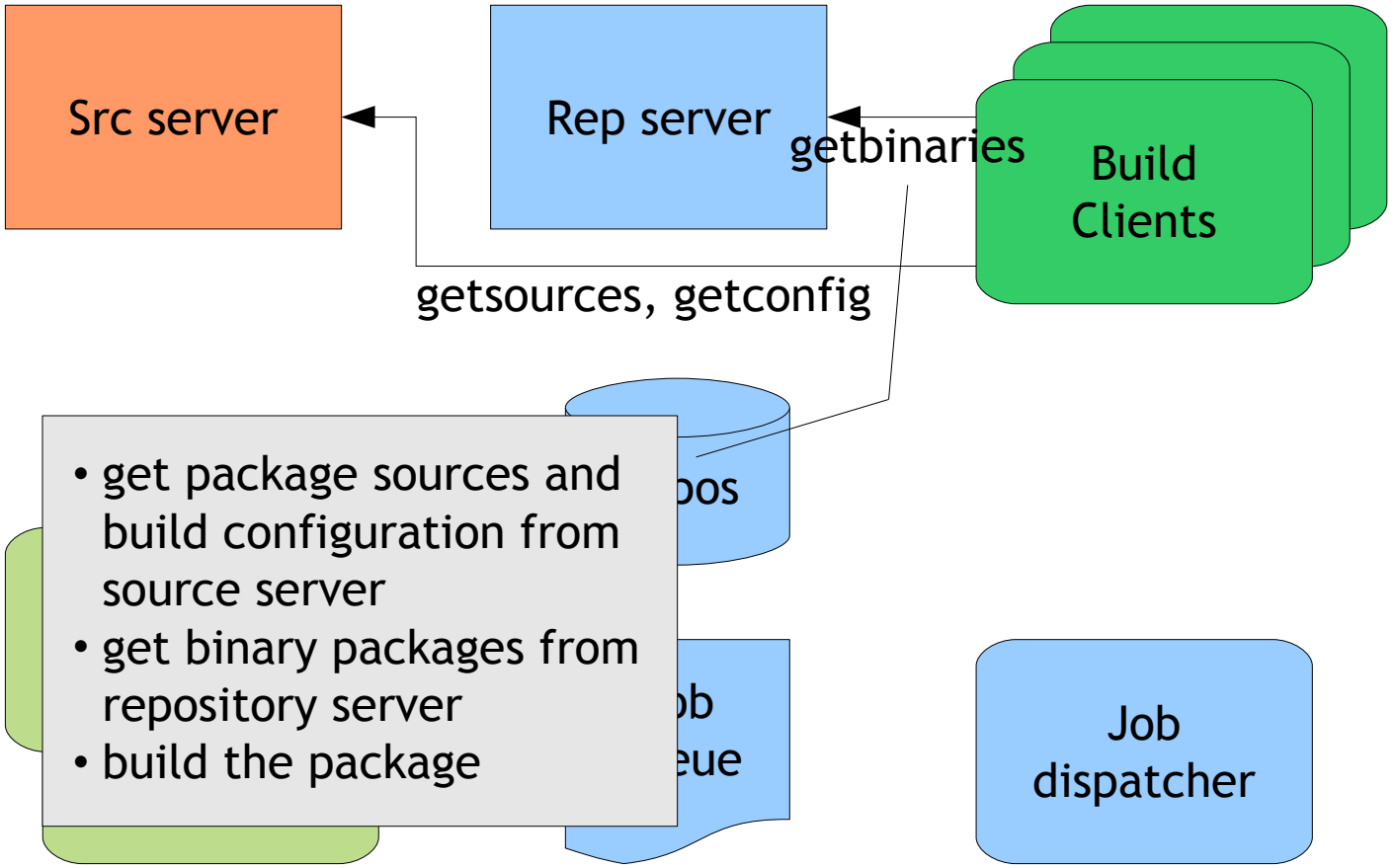


Backend Architecture



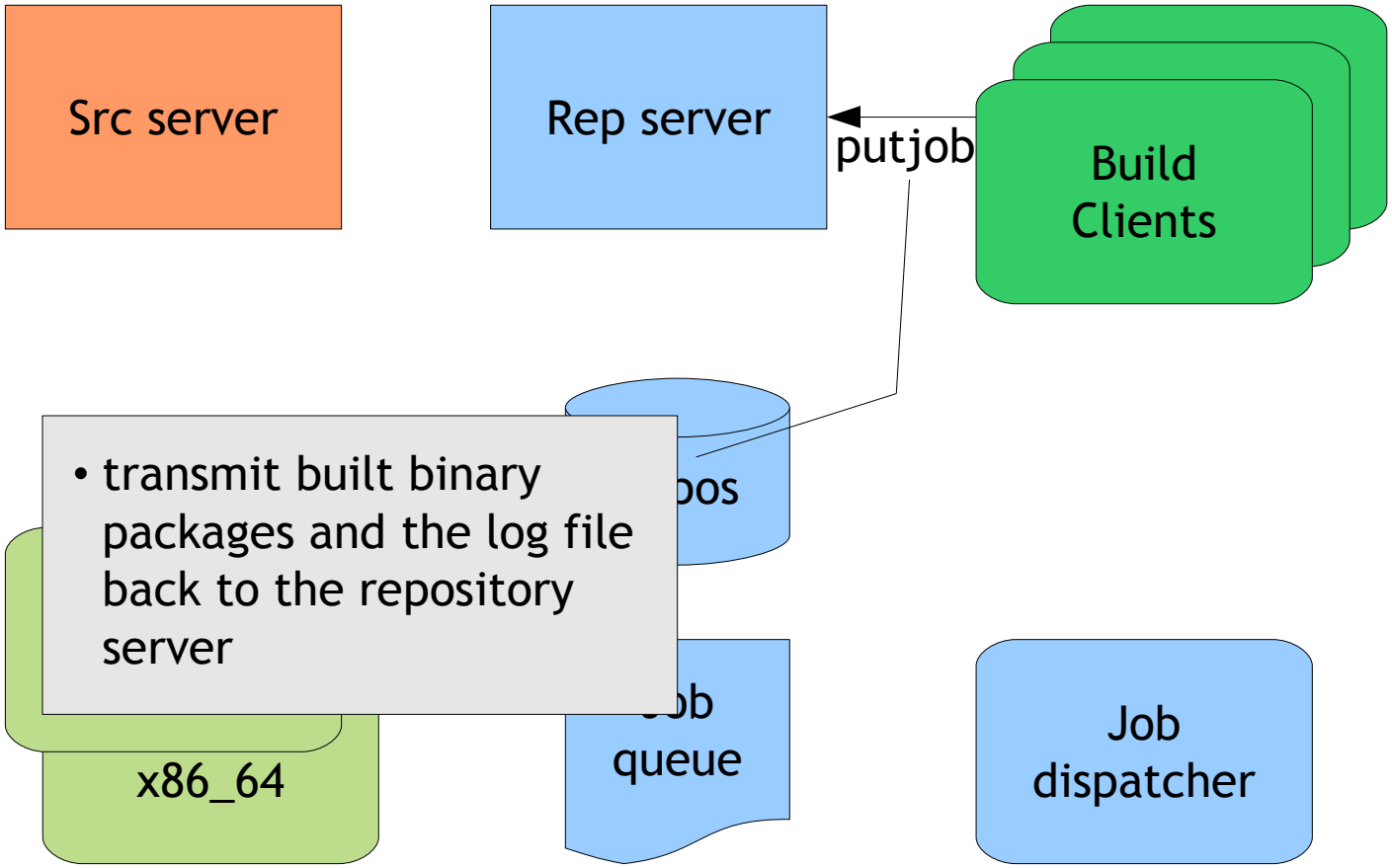
Novell

Backend Architecture

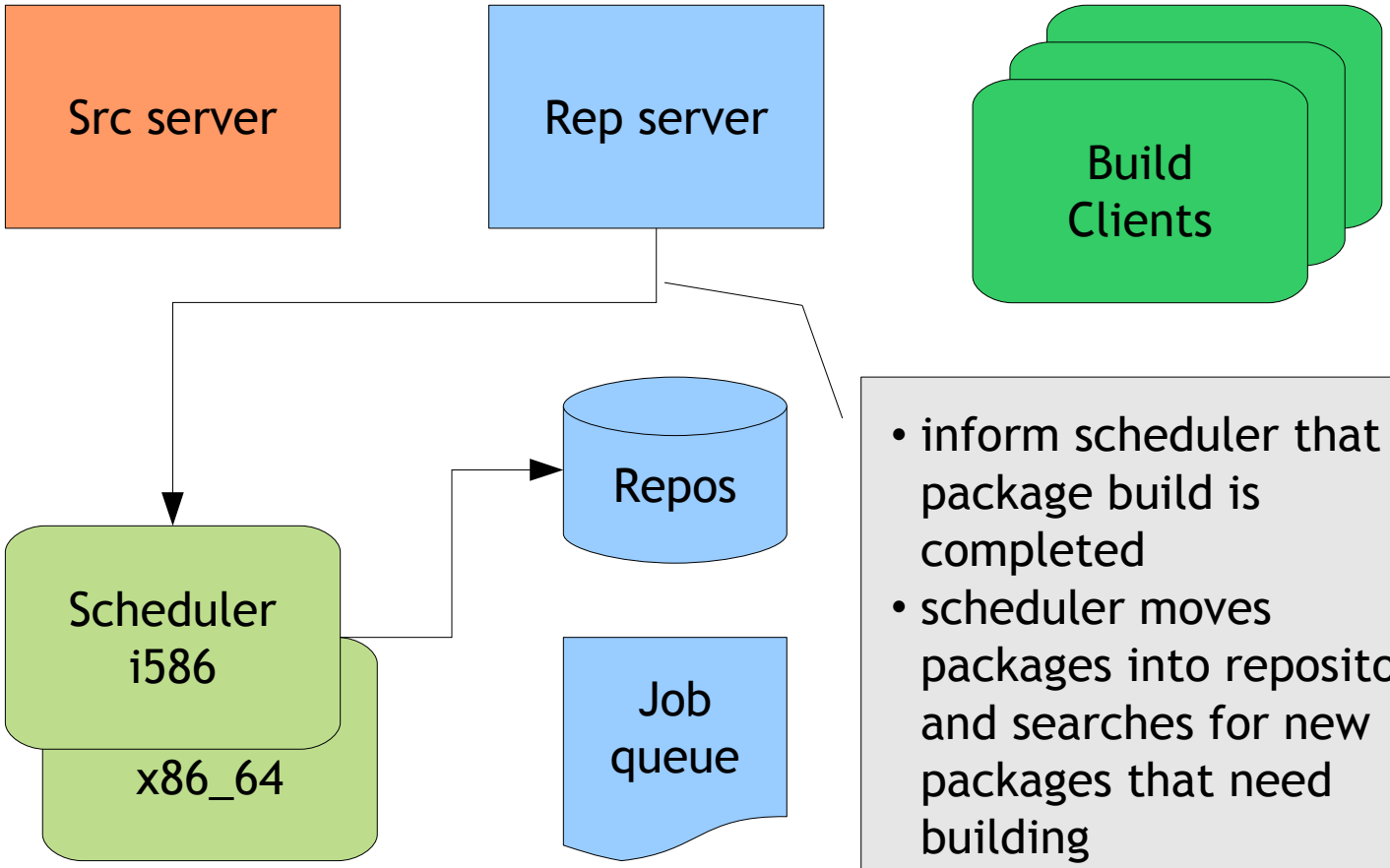


Novell

Backend Architecture



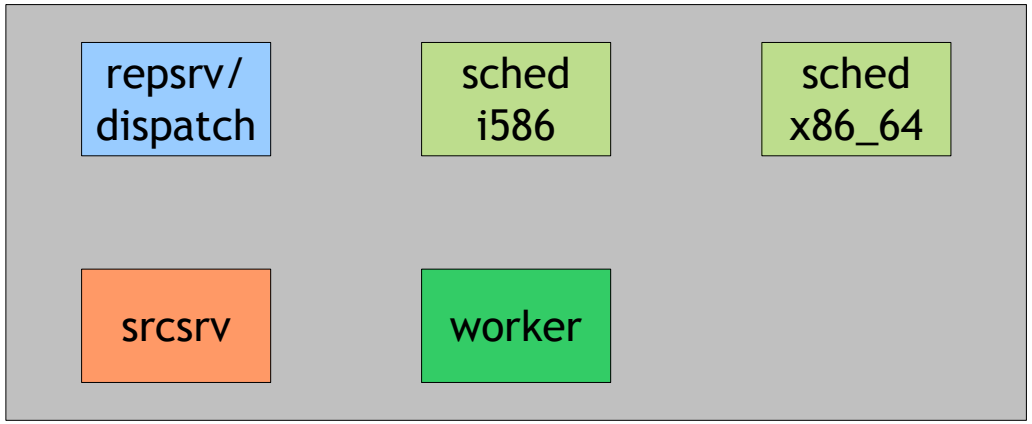
Backend Architecture



- inform scheduler that package build is completed
- scheduler moves packages into repository and searches for new packages that need building

Scalability

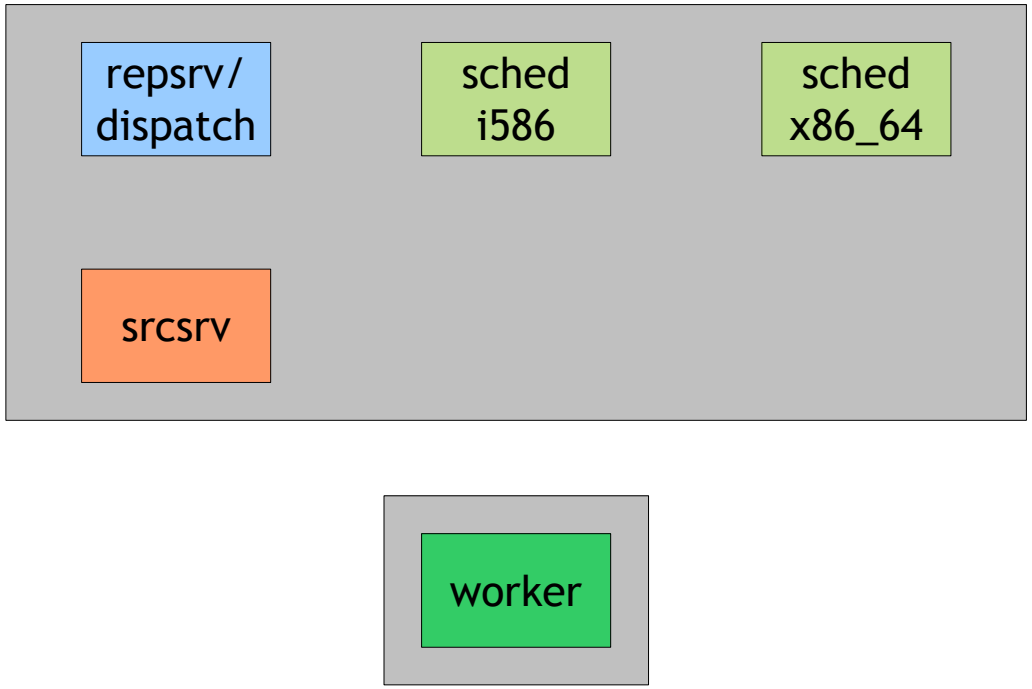
Simplest configuration: everything on one host



Novell

Scalability

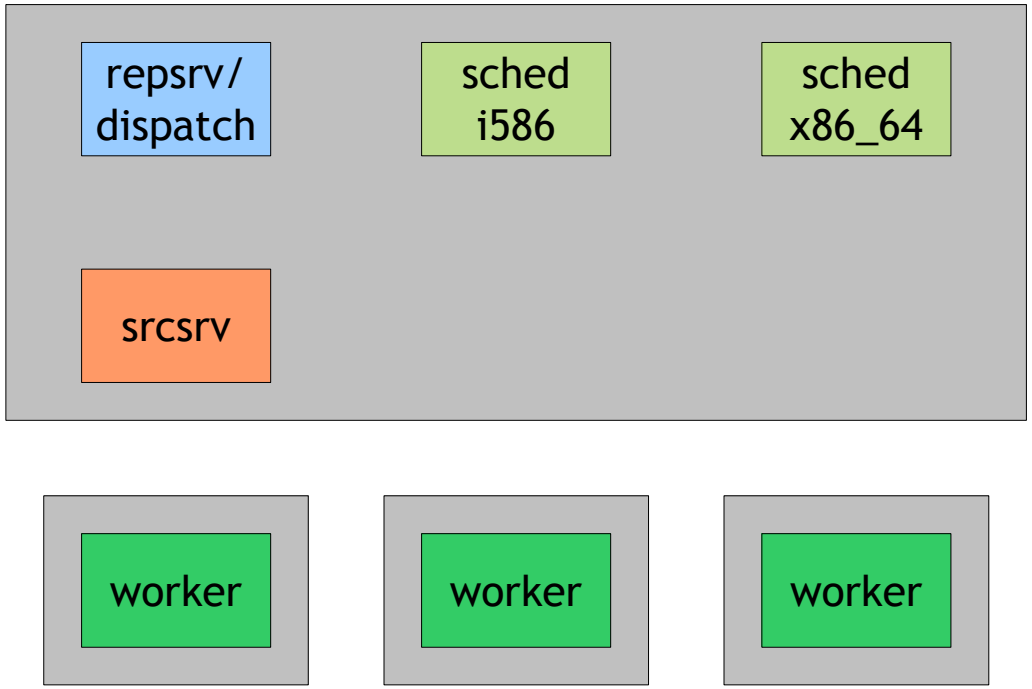
Put build client on extra host



Novell

Scalability

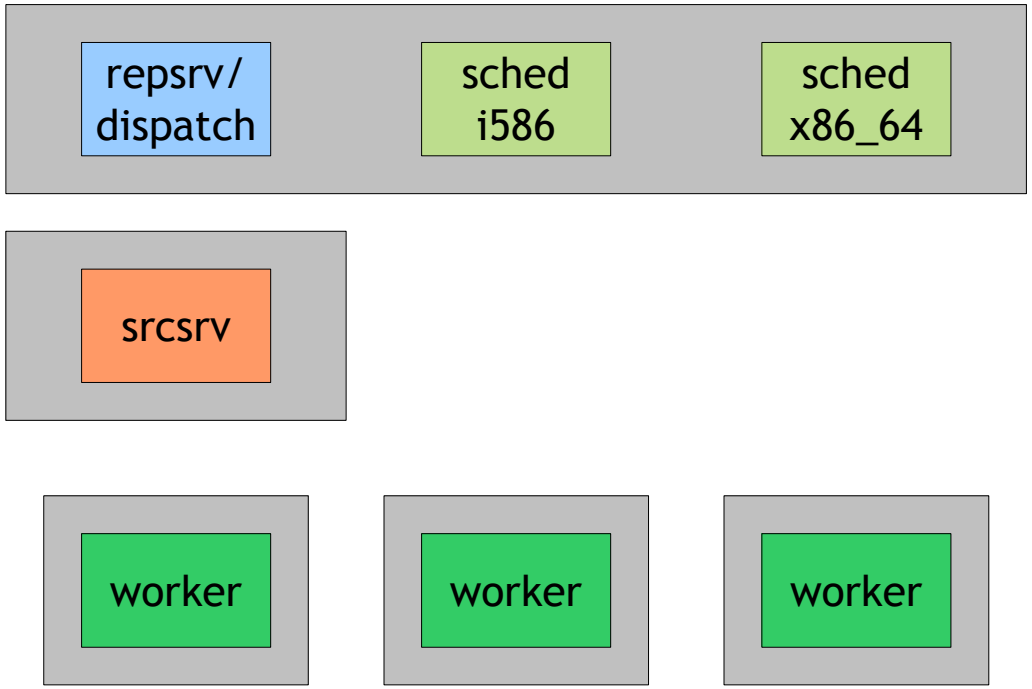
Add more clients



Novell

Scalability

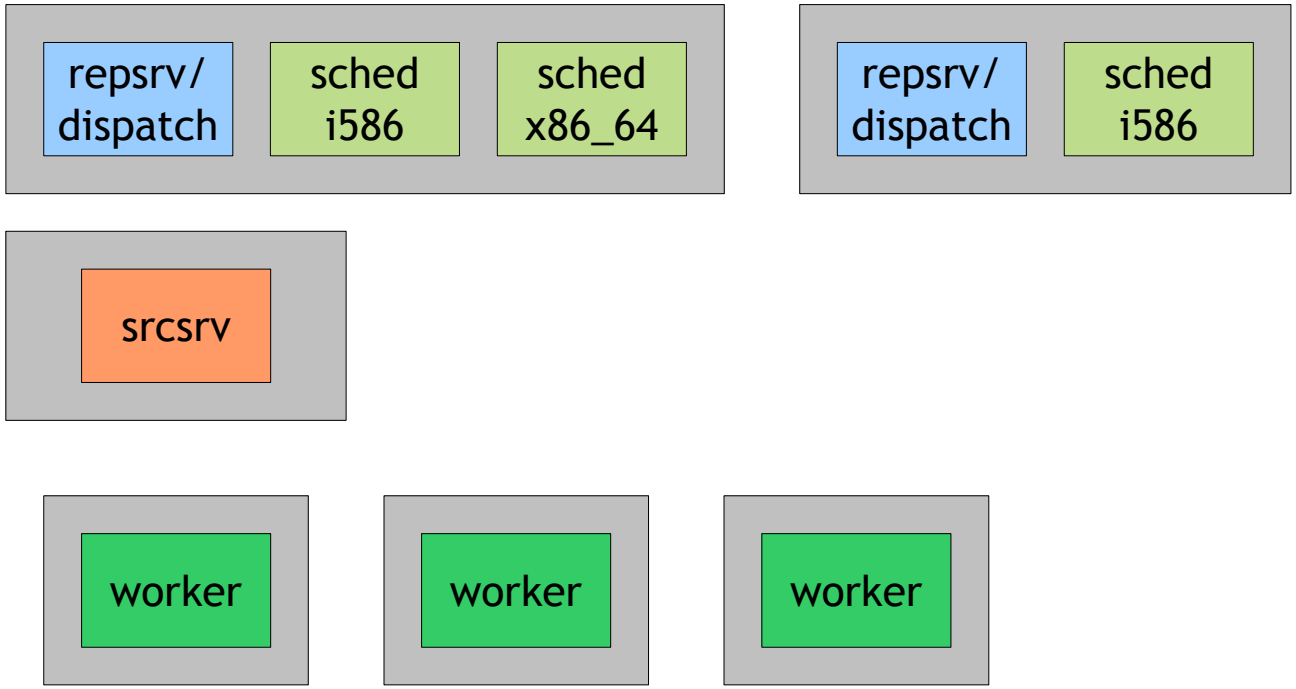
Put source server on extra host



Novell

Scalability

Partition repositories



Novell

Rebuild triggering

Packages automatically get rebuild if a package they depend on is changed

- naïve implementation that just looks for changed binary packages would lead to endless builds in case of dependency cycles
- Example:
  - gnome-keyring: BuildRequires: CASA-devel
  - CASA: BuildRequires: gtk-sharp
  - gtk-sharp: BuildRequires gtkhtml2-devel → gtkhtml2 → libgnome-keyring.so.0
- Solution: use source identifiers, track change propagation, and cut cycles

The dependency meta file

The dependency meta file consists of all package source identifiers used (direct and indirect)

Example: CASA package

- level 0: own source identifier
  ```
  e71a2f9181669e4c787e6fbf7ce63414 CASA
  ```
- level 1: source md5 of direct dependencies
  ```
  ...
  8e72d8e96df97cbea7707ba26a3fc31d gtk-sharp
  ...
  ```
- level 2: source md5 of dependencies over one hop
  ```
  ...
  0724adba09a56cea17d41fdb35450d45 gtk-sharp/gnome-keyring
  ...
  ```

Packages are triggered for rebuild *iff* the meta file of the last build is different from the calculated one

Conclusion

Data storage

- project and package data is stored in simple xml files
- the source repository stores package files in an efficient way, nothing gets deleted

Server Framework

- new functions can be added with just a couple of lines of perl code

Package building

- scheduler → dispatcher → build client → repository
- all data is transferred via http, no NFS

Rebuild triggering

- works by comparing meta files containing source md5sums

Novell®

General Disclaimer

Novell