Open Build Service (OBS)

Cross-Architecture Build Capabilities

Classic Cross Building

Classic Cross Compile Is

- Building a software on architecture A for architecture B.

- Used when architecture B is not suitable for developing because:
  - Slow CPU
  - Low memory & storage
  - Convenience: Can't be used as developer workstation
  - Hardware device not available (yet)

Classic Cross Compile is

Embedded developers are used to do

- ./configure –host=armv7l-suse-linux

or

- rpmbuild --target=armv8l-suse-linux

on their ia32 or x86_64 workstation

Advantages

- Very fast results
- Work can be done on all kinds of hardware.

Disadvantages

- Sources need to support cross compile
- Need to handle dependencies for two architectures
- Build result may differ from native compile
- No real-life testing during development
- No test suite runs usually

Emulated Build

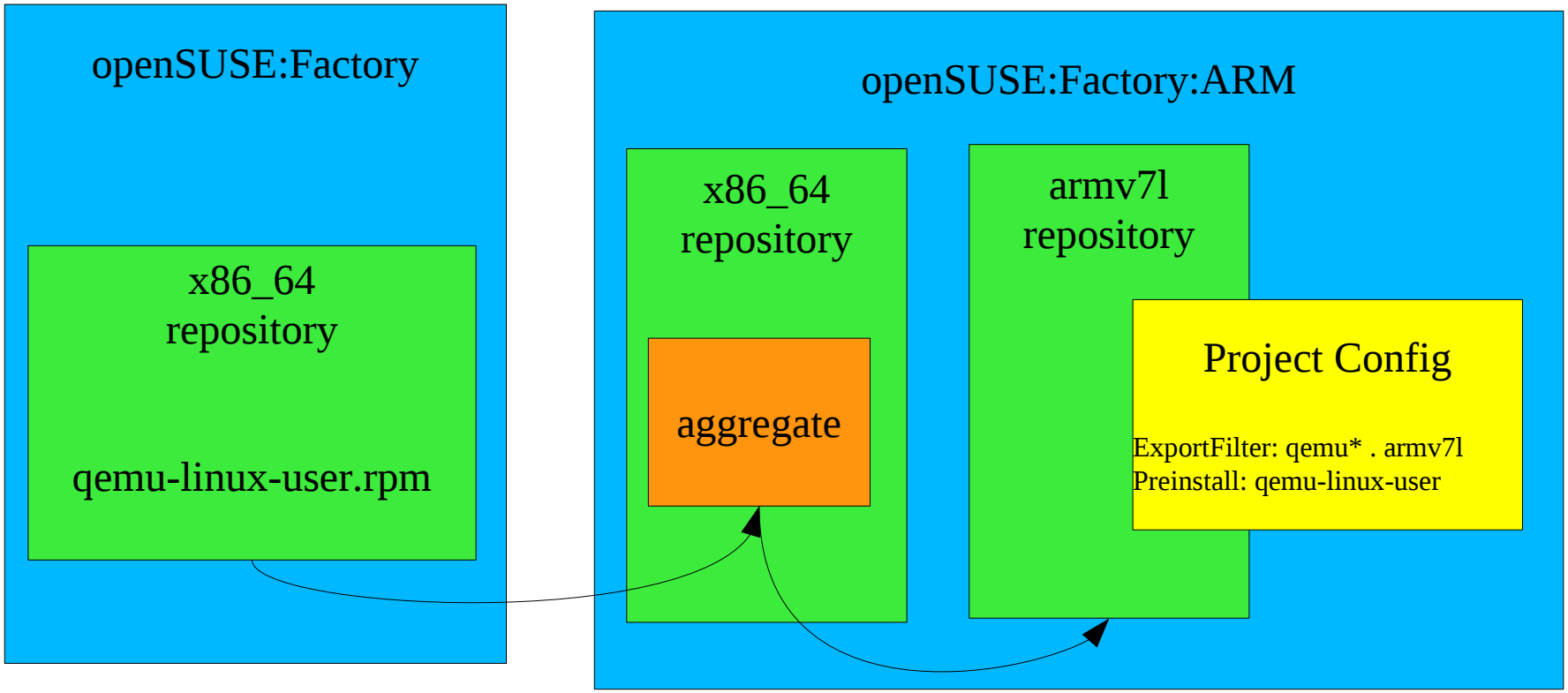
Emulated System

- Full emulation (booting target kernel) is usually too slow and not flexible enough.

- User land emulation using Qemu:

  - Install qemu-$ARCH binary for running emulated binaries
  - Register $ARCH binaries to be handled via qemu-$ARCH binary.
    The OBS build script can do this during VM setup.

Setup



Fetch and update rpm
via _aggregate

ExportFilter to arm repos

Advantages

- No changes in package sources are needed (almost)
- Using it in OBS allows every developer to build and run their tests for the target architectures without the need of real hardware.
  => This makes it possible at all to do maintenance updates for the openSUSE community for arm.

Disadvantages

- Qemu bugs or missing support may have an influence to the build result.

  => Lots if qemu fixes during 12.2:ARM development. Qemu thread handling problems are workarounded

- Emulation overhead may slow down build process too much.

- We rely currently on our patched qemu using qemu-$ARCH-binfmt handler, to be pushed upstream.

Example

- OpenSUSE:12.2:ARM is using this setup.

- Want to see how it works ?

```
osc build \
  –alternative-project=openSUSE:Factory:ARM \
  standard armv7l $YOUR_FILE.spec
```

Tricks for speed improvements

The icecream/distcc trick

- Use native hardware or full emulation for building
- Use compiler backend on another, faster host.
- Remote host can have any architecture
- Use icecream or distcc to distribute the job
- **ADVANTAGE:** scales very very great
- **DISADVANTAGE:** hard to make it secure

- Code to sync build environments exists SUSE-

Native binaries in qemu-linux-user

- Use some selected binaries for native architecture during qemu-linux-user emulation.
- Building an acceleration package which repackages binaries and needed libs into other directory
- **ADVANTAGE:** Best possible performance
- **DISADVANTAGES:**
  - risk that tool is arch dependent
  - source revision needs to be kept in sync

Future:
Issues To Fix In OBS

Open Issues

- Qemu-linux-user thread handling is not reliable.
- kiwi qemu build works only with workaround atm

Work in Progress by B1-Systems: Transparent Cross Building

Transparent Cross Building

- Goal:
  - Do classic cross builds in OBS
  - Use toolchain from host architecture (speedup, no emulation)
  - Use the same spec-file for host and target architecture builds
  - Avoid faking/emulation of build steps where reasonable/possible

Advantages

- Very fast results
- Work can be done on all kinds of hardware.

Disadvantages

- Sources need to support cross compile
- Need to handle dependencies for two architectures
- Build result may differ from native compile
- No real-life testing during development
- No test suite runs usually

Transparent Cross Building

- **Not** the intended Goal:
  - Build existing distribution without any modification

Transparent Cross Building

- OBS scheduler need to take multiple architecture in account
- Toolchain BuildRequires get substituted with packages for the host architecture
- Setup separated /sysroot for target architecture(s):
  - /opt/cross-%_target_cpu/
  - With separated package database
- Exploit cross-compiling support of build environments:
  - Autotools, CMake, …

Transparent Cross Building

- What gets address so it get “transparent”?
  - Some BuildRequires: need to get substituted with Host Arch packages or cross-compilers
  - %check section gets NOOP in the first round
  - Common build environment calls in spec files need to make use of common packaging macros:
    - %configure
    - %cmake
    - …
- Goal: use the same build spec file
  - Move arch conditionals to the project config

Transparent Cross Building

- Is this the best solution for me?
  - Do you have enough time/resources to package your entire project/product yourself?
  - Are you willing to fix a lot of packages which are not aware of cross building?
  - Emulated cross builds are not fast enough for your packages/projects?
- If all of above applies, this might be the right solution
- Otherwise emulated or other OBS cross build might be the right thing for you

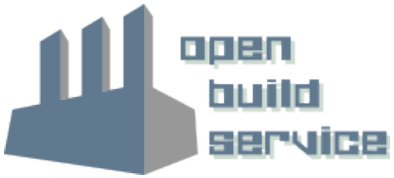
Transparent Cross Building

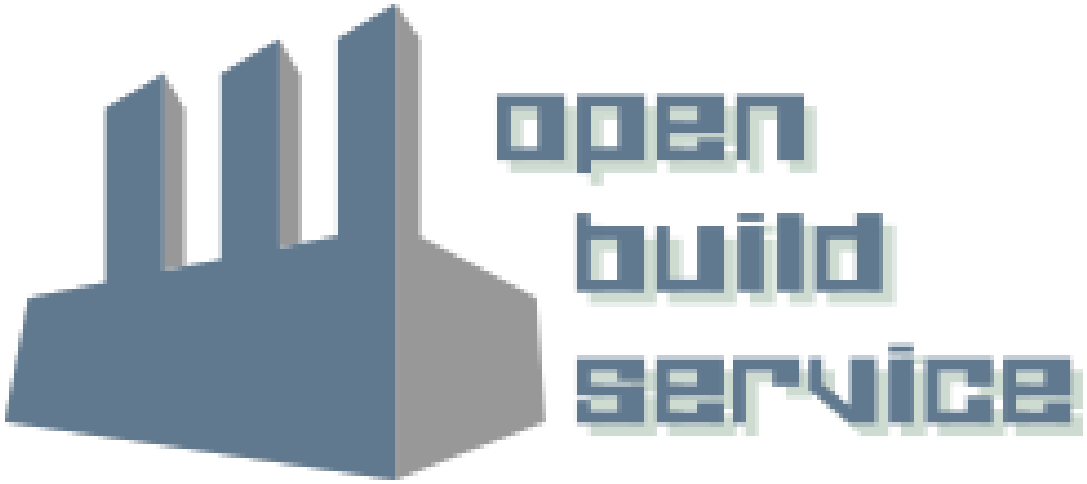
- First round:
  - Initial support to make scheduler aware of multiple architectures
  - Enable obs-build to setup separated /sysroot
  - … and do classic cross builds
- Later:
  - Optimize scheduler for taking multiple architectures in account
    - First round: requires that the entire toolchain of host arch is build
  - Make some use of binfmt Magic
    - invoke qemu-$ARCH to run %check or other things
    - But use as much host toolchain as possible

Demonstration

Learn more about the
Open Build Service
www.openbuildservice.org

Thank you.




open
build
service