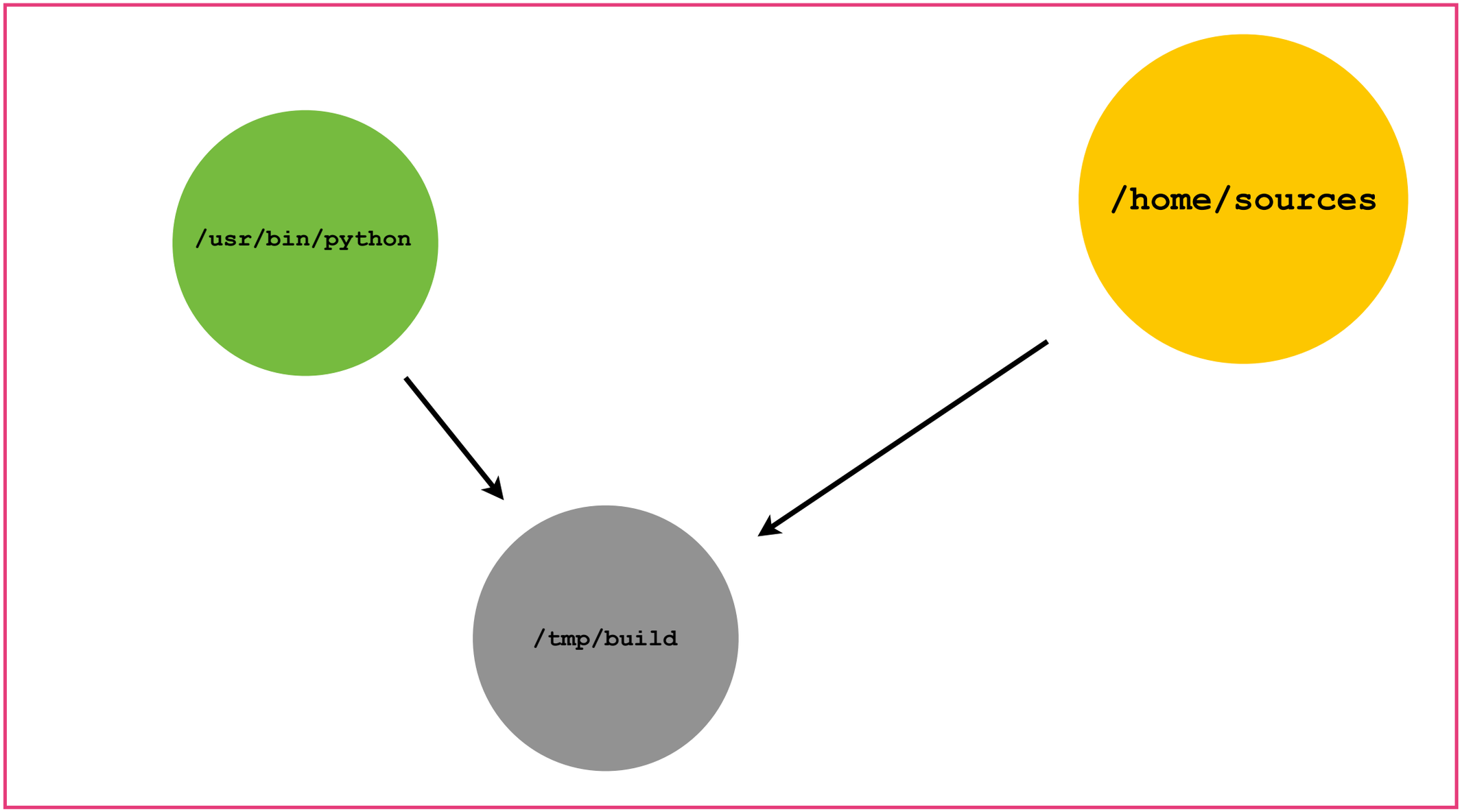
Source directory structure

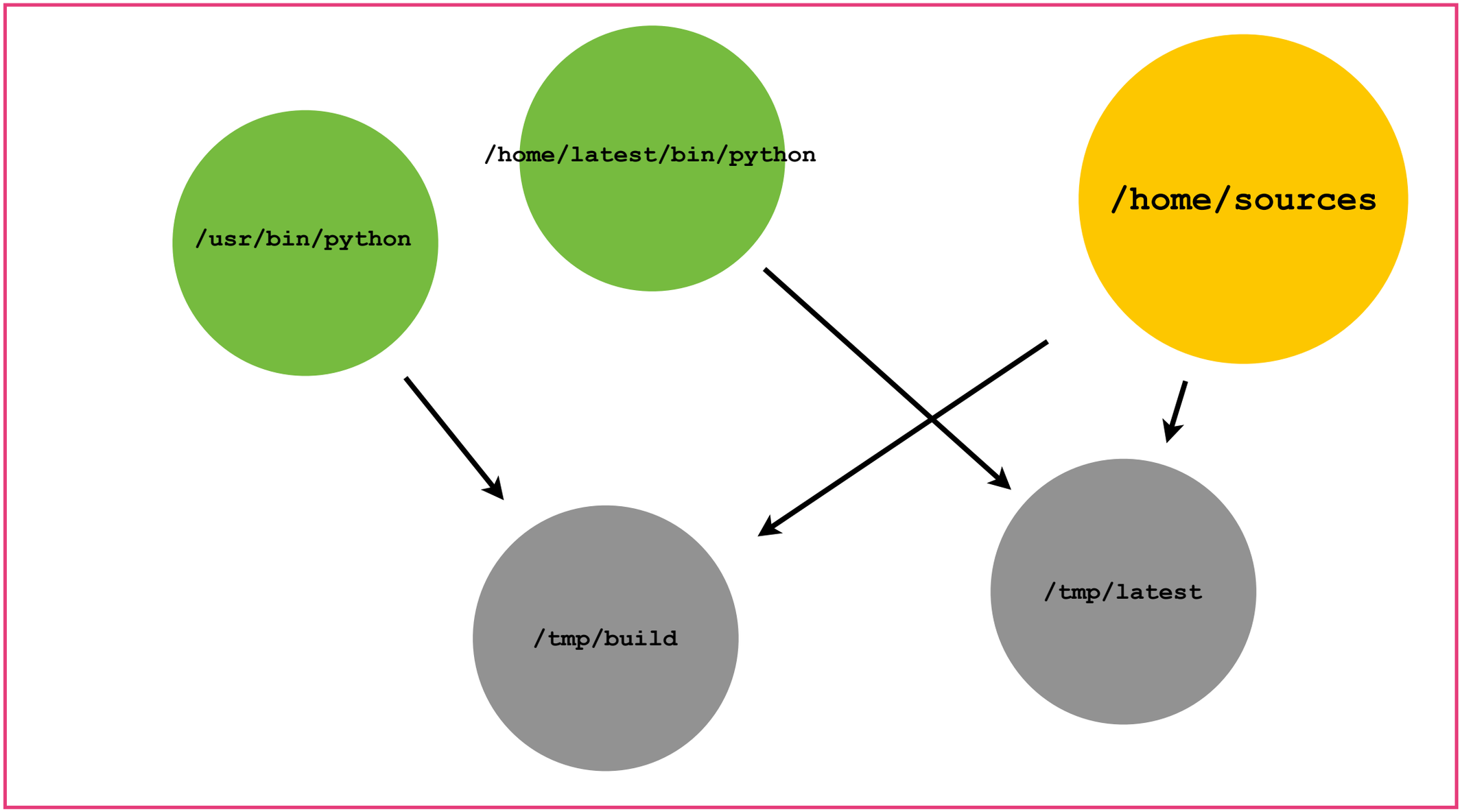
- scons
- boost
- cctbx_project
  - libtbx
  - tbxx
  - boost_adaptbx
  - omptbx
  - fable
  - scitbx
  - chiltbx
  - cctbx
  - iotbx
  - mmtbx
  - spotfinder
  - rstbx
  - smtbx
  - gltbx
  - crys3d
  - wxtbx
  - compcomm
  - dox
  - dox.sphinx
  - rpmbuild
  - cbflib_adaptbx
  - clipper_adaptbx
  - fftw3tbx
- antlr3
- gui_resources
- lapack_fem
- cbflib
- ccp4io
- ccp4io_adaptbx
- clipper
- annlib
- annlib_adaptbx
- tntbx

source/build strictly separated



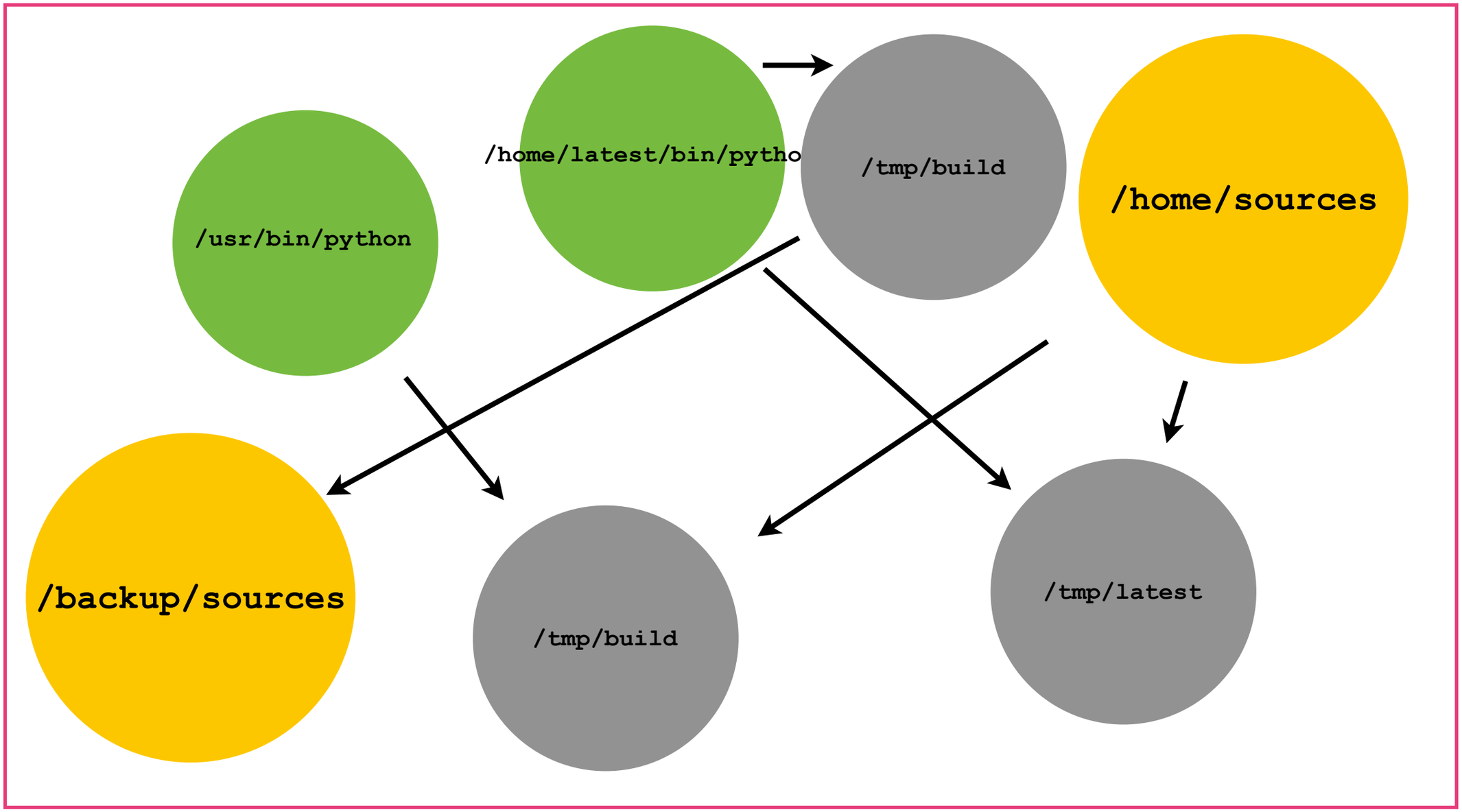

```
cd /tmp/build
/usr/bin/python /home/sources/cctbx_project/libtbx/configure.py scitbx
source setpaths.sh
make
```

same sources, different build and/or python



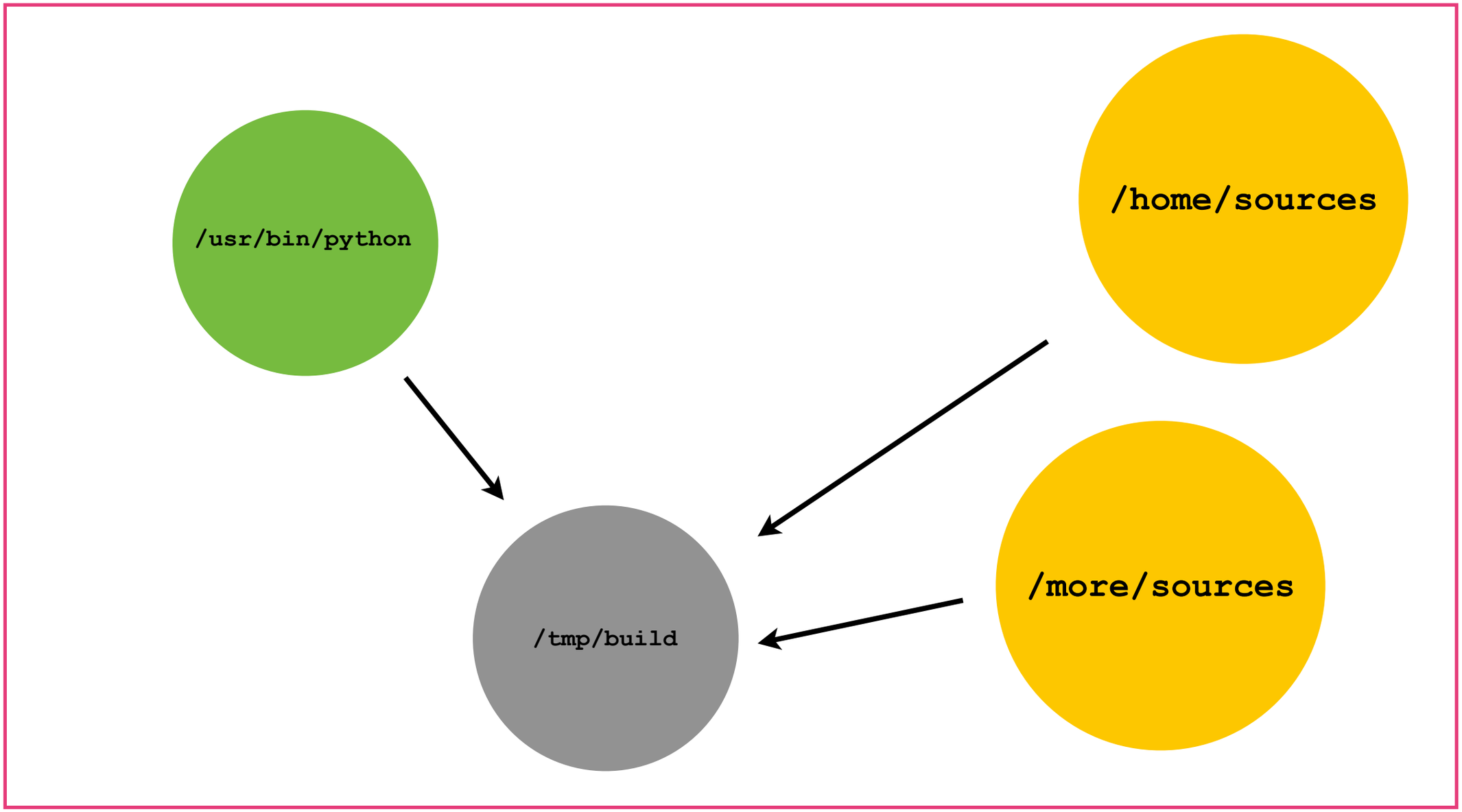

```
cd <build>
<python> /home/sources/cctbx_project/libtbx/configure.py scitbx
source setpaths.sh
make
```

different sources



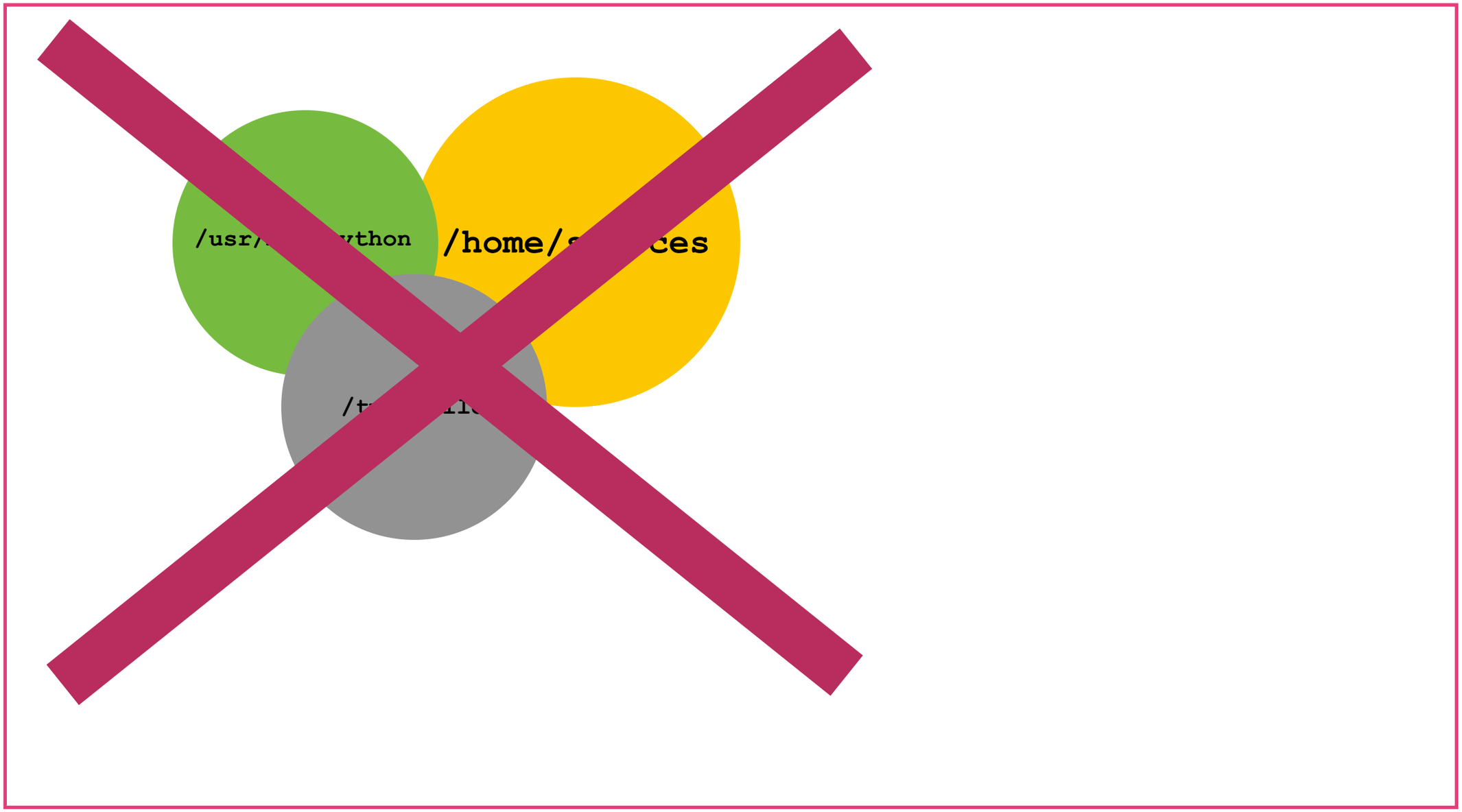

```
cd <build>
<python> <sources>/cctbx_project/libtbx/configure.py scitbx
source setpaths.sh
make
```

distributed sources




```
cd /tmp/build
/usr/bin/python /home/sources/cctbx_project/libtbx/configure.py \
  --repository /more/sources scitbx myworld
source setpaths.sh
make
```

don’t move build inside sources



libtbx = build + distribution system

- python ../cctbx_project/libtbx/configure.py iotbx
- source setpaths.csh
- make

- mkdir tmp; cd tmp
- libtbx.start_binary_bundle iotbx iotbx
- # customizations here, if any
- libtbx.bundle_as_selfx iotbx mac_os_10_5
- # cp iotbx_mac_os_10_5.selfx /html/current/

- perl iotbx_mac_os_10_5.selfx

- Also works on Windows: libtbx.bundle_as_exe
- Also works for source bundles

Adding a module

- mkdir myworld
- libtbx.configure myworld

- Definition of dependencies (recursive):
  - myworld/libtbx_config:

```
{
  "modules_required_for_use": ["cctbx", "fable"],
  "optional_modules": ["ccp4io", "cbflib", "antlr3"],
}
```

- Tying into the build system (compilation and linking):
  - myworld/SConscript
  - may call other SConscripts in subdirectories
- Adding a command ("myworld.show_help"):
  - myworld/command_line/show_help.py
  - command names can be redefined arbitrarily
- Complex customizations:
  - myworld/libtbx_refresh.py
  - e.g. used for building C++ sources

- Move myworld to SourceForge cctbx project?

Command dispatcher system

- Adding a command ("myworld.show_help"):
  - myworld/command_line/show_help.py
  - command names can be redefined arbitrarily

- libtbx/configure.py creates "build" directory
  - e.g. build/lib, build/bin, build/setpaths.csh
- source setpaths.csh: PATH=build/bin:$PATH
  - only one entry for the entire build
- build/bin/myworld.show_help = auto-generated shell script
  - export LD_LIBRARY_PATH, PYTHONPATH
  - exec python .../myworld/command_line/show_help.py "$@"
- Also works on Windows (using tricks to replace the shell script)
- Dispatcher system encapsulates the applications
- To the user the applications are equivalent to stand-alone executables
- No environment pollution
- Protects against corruption of runtime environment

libtbx vs. "configure-make-install"

- Unix-based systems:
  - "configure-make-install" and "setup.py" of 3rd party packages
  - Python installation as part of cctbx (labelit, phenix, phaser...) installation
    - ./configure --prefix=build/base
    - make
    - make install
  - similarly for other components (zlib, gtk2, wxPython, etc.):
    - everything is installed into build/base
- Windows systems:
  - Manual building of base (Python, wxPython, etc.) and archiving to a .zip file.
  - no "make install" equivalent (possible, but not necessary)
- Everything based on libtbx:
  - works identically under Unix and Windows
  - does not require any shell scripts
  - does not make copies of the libraries and executables, or any other derived files
  - no "make install" equivalent (possible, but not necessary)
- Other libtbx features:
  - "central source - multiple build directories"
    - different platforms, --build=release, --build=debug
  - to work on a subset only: libtbx.configure --only scitbx
  - to work with everything again: libtbx.configure myworld