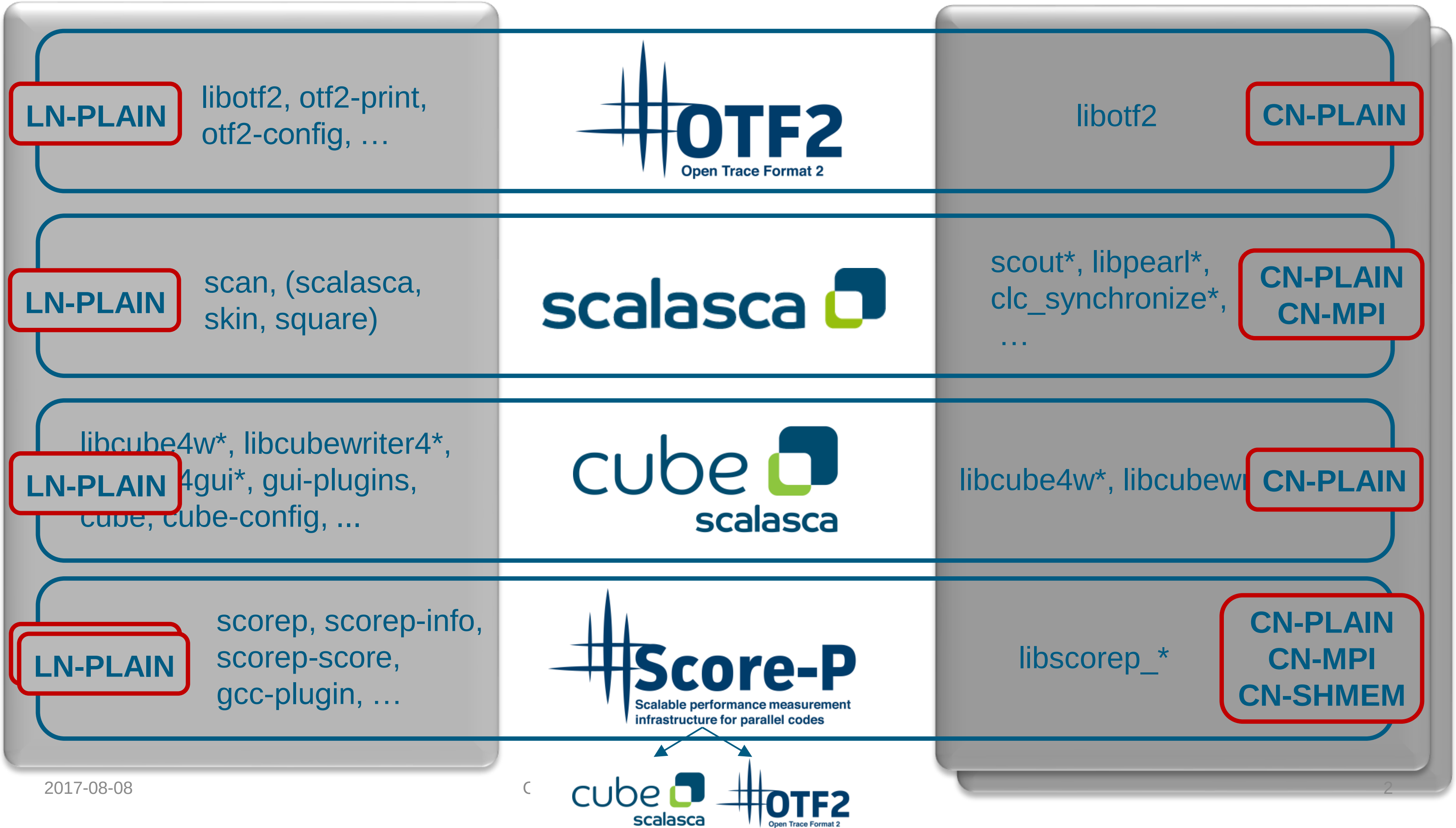
Autotools For Supercomputers (AFS)

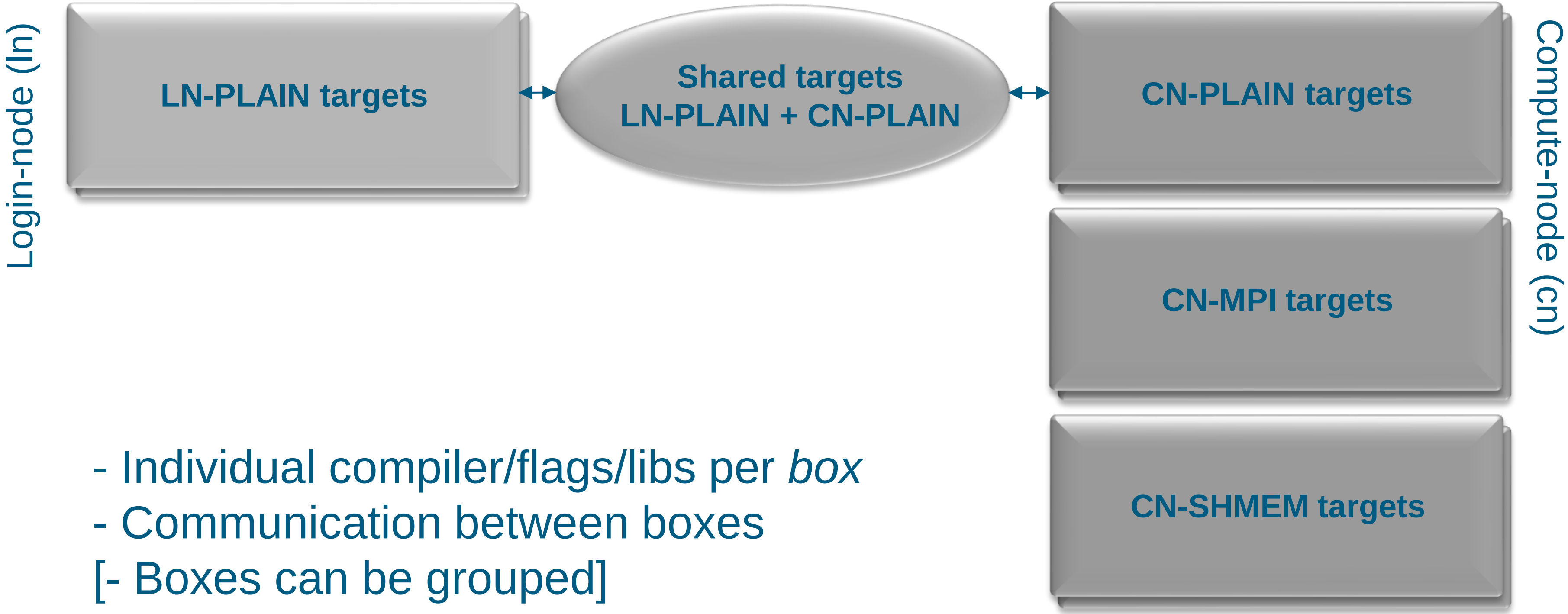
A build-infrastructure on top of autotools

2017-08-08 | Christian Feld



Login-node (ln)			Compute-node (cn)	
LN-PLAIN	libotf2, otf2-print, otf2-config, …	OTF2 Open Trace Format 2	libotf2	CN-PLAIN
LN-PLAIN	scan, (scalasca, skin, square)	scalasca	scout*, libpearl*, clc_synchronize*, …	CN-PLAIN CN-MPI
LN-PLAIN	libcube4w*, libcubewriter4*, 4gui*, gui-plugins, cube, cube-config, ...	cube scalasca	libcube4w*, libcubew	CN-PLAIN
LN-PLAIN	scorep, scorep-info, scorep-score, gcc-plugin, …	Score-P Scalable performance measurement infrastructure for parallel codes	libscorep_*	CN-PLAIN CN-MPI CN-SHMEM

General structure



- Individual compiler/flags/libs per *box*
- Communication between boxes

[- Boxes can be grouped]

Why autotools?

JÜLICH FORSCHUNGSZENTRUM

Everything started in 2009 with Score-P
Scalable performance measurement infrastructure for parallel codes

We came from:

VampirTrace
- autotools
- integrated into Open MPI (autotools required)
- weak support for cross-compile systems

Scalasca + TAU
- home-grown configure script
- handwritten Makefiles
- lots of systems supported
- single configure step

New, distributed team:

- some autotools experience
- no CMake experience
- handwritten Makefiles not considered feasible

- Open MPI integration desired
- minimal user-requirements

→ **autotools**

Our ambition

- Support relevant HPC systems, clusters, laptops, Unix-like
  - provide reasonable compiler defaults
- Make life easy for user
  - single configure, make, make install
  - easy, unambiguos customization
- Make life fairly easy for developer
  - just write Makefile.am (as usual)
  - decide where to build (ln, cn-plain, cn-mpi, ...)
  - communicate between ln, cn-plain, cn-mpi: Makefiles + sources (defines)
- Make life bearable for buildsystem maintainer
  - header/lib checks: prevent cross-compiling pitfalls
  - add new ln/cn subdirectories easily
  - provide/modify compiler defaults
  - provide means of communication between boxes
  - support subpackages
  - standalone AFS infrastructure, easy to apply
- Easy packaging
  - make distcheck, including subpackages
  - Linux distribution friendly: staged install
- New: Modular Supercomputing
  - support several cn-architectures easily

Goals reached in Score-P ecosystem (AFSv1)?

Partially – remainder about AFSv2 prototype

Single configure

- General structure leads to toplevel + one subdir per *box*
  - Set of compilers (`CC`, `CXX`, `F77`, `FC`) per directory
  - Our approach: don‘t merge cn-plain and cn-mpi, although possible

```
.
├── bootstrap
├── build-cn-mpi
│   ├── configure.ac
│   └── Makefile.am
├── build-cn-plain
│   ├── configure.ac
│   └── Makefile.am
├── build-ln
│   ├── configure.ac
│   └── Makefile.am
├── configure.ac
└── Makefile.am
```

- Structure could be realized with plain autoconf (`AC_CONFIG_SUBDIRS`), but all subdirs would get same options. We want subdir-unique or group options

- Replace `AC_CONFIG_SUBDIRS`:

```
#AC_CONFIG_SUBDIRS([build-ln build-cn-plain build-cn-mpi])
AFS_CONFIG_SUBDIR([build-ln], [], [ln], [--with-foo])
AFS_CONFIG_SUBDIR_IMMEDIATE([build-cn-plain], [cn], [plain], [])
AFS_CONFIG_SUBDIR_IMMEDIATE([build-cn-mpi], [cn], [mpi], [])
```

options

group

Single configure (2)

- Do something special at initialization: wrap `AC_INIT`

```
#            AC_INIT([Demo], [trunk], [c.feld@fz-juelich.de], [demo])
AFS_TOPLEVEL_AC_INIT([Demo], [trunk], [c.feld@fz-juelich.de], [demo])
```

```
#          AC_INIT(          [Demo], [trunk], [c.feld@fz-juelich.de], [demo])
AFS_SUBDIR_AC_INIT_LOGINNODE([Demo], [trunk], [c.feld@fz-juelich.de], [demo], [],
                             [Login node], [], [ln])
```

```
#          AC_INIT(            [Demo], [trunk], [c.feld@fz-juelich.de], [demo])
AFS_SUBDIR_AC_INIT_COMPUTENODE([Demo], [trunk], [c.feld@fz-juelich.de], [demo], [],
                               [Compute node plain], [cn], [plain])
```

  - detects system, wraps existing macros, setup names, `--with-target`, per-package help, sets basis to prevent common cross-compiling pitfalls, fixes linker preferences (Cray, BG, Fujitsu), ...

- Additionally, replace `AC_OUPUT` with `AFS_OUTPUT`
  - for technical reasons
  - provide summary output (`AFS_SUMMARY`)

These 6 macros provide
- single configure
- easy packaging (make distcheck)
- staged install
- per-subdir options
- per-package help
- arbitrary complex packages

Extensible to sub-packages with `AFS_CONFIG_SUBPACKAGE`

What is CC?
Unique variables/options

- Autoconf macros (`AC_ARG_WITH`, `AC_ARG_ENABLE`, `AC_ARG_VAR`) generate configuration options for each subdir:

```
Optional Features:
  ...
  --enable-silent-rules less verbose build output (undo: "make V=1")
  --enable-shared    build shared libraries [default=yes]
  --enable-static    build static libraries [default=yes]
  ...

Optional Packages:
  ...
  --with-pic         try to use only PIC/non-PIC objects [default=use both]
  ...

Some influential environment variables:
  CC           C compiler command
  CFLAGS       C compiler flags
  ...
```

- Options ambiguous: what is `CC=mpixlc` supposed to mean for ln or cn-plain?

What is CC? (2)
Unique variables/options

- Make unique by **redefining** `AC_ARG_WITH`, `AC_ARG_ENABLE`, `AC_ARG_VAR`: use package, group and subdir name as pre/postfix:

```
Optional Features:
  ...
  --enable-demo-cn-mpi-silent-rules less verbose build output (undo: …
  --enable-demo-cn-mpi-shared  build shared libraries [default=yes]
  --enable-demo-cn-mpi-static  build static libraries [default=yes]
  ...

Optional Packages:
  ...
  --with-demo-cn-mpi-pic       try to use only PIC/non-PIC objects [def …
  ...

Some influential environment variables:
  demo_cn_mpi_CC          C compiler command
  demo_cn_mpi_CFLAGS      C compiler flags
  ...
```

- Applies to all `AC_ARG_WITH`, `AC_ARG_ENABLE`, `AC_ARG_VAR` in subdir → uniqueness reached

What is CC? (3)
Package/Group variables/options

- But ... number of options explodes.
  Fight by introducing **package** and **group-options**:

```
Optional Features:
  ...
  --enable-demo-~~cn-mpi-~~silent-rules less verbose build output (undo: …
  --enable-demo-cn-~~mpi-~~shared  build shared libraries [default=yes]
  --enable-demo-cn-~~mpi-~~static  build static libraries [default=yes]
  ...

Optional Packages:
  ...
  --with-demo-cn-~~mpi-~~pic       try to use only PIC/non-PIC objects [def …
  ...

Some influential environment variables:
  demo_cn_mpi_CC          C compiler command
  demo_cn_mpi_CFLAGS      C compiler flags
  ...
```

What is CC? (4)
Package/Group variables/options

- How to define package and group options/variables?
  Individual definition per package:

```
AC_DEFUN_ONCE([AFS_ARGS_PACKAGE], [
m4_define([_AFS_PACKAGE_ARG_ENABLES], [silent-rules])
m4_define([_AFS_PACKAGE_ARG_WITHS], [target])
# variables: _AFS_PACKAGE_ARG_VARS
])

AC_DEFUN([AFS_ARGS_GROUPS], [
m4_if(AFS_SUBDIR_GROUP, [cn], [
    m4_pushdef([_AFS_GROUP_ARG_ENABLES], [shared, static])
    m4_pushdef([_AFS_GROUP_ARG_WITHS], [pic])
    # variables: _AFS_GROUP_ARG_VARS])
])
```

- `AFS_CONFIG_SUBDIR[_IMMEDIATE]` passes user-provided group and local options selectively to subdirs

--with-target

- Motivation: build for different architectures into single `--prefix`, e.g.,
  - Xeon and Xeon Phi
  - Release and debug
  - Modular Supercomputing: program spawned over multiple architectures (targets)
  - One configure per target

- `--with-target` affects cn installation directories
  (currently hardcoded, will become customizable)
  - libdir
  - libexecdir
  - program-prefix

- Allows for per-target compiler and flags

```
Configuration summary:
...
    Demo [Compute node MPI]:
      Changes due to --with-target or afs_hpc_system:

        libdir:                    "${exec_prefix}/lib/knl"
        libexecdir:                "${exec_prefix}/libexec/knl"
        program_prefix:            "knl-"
...
```

Compiler + flags selection

- Applies to *boxes* ln and cn-plain (MPI done differently)

- Select from presets in package-specific configuration file:
  `unique_variable:system:vendor:wrapper:target=value`

- **Unique variable** names come from
  `AFS_COMPILER(languages)`

- **System** is detected to be either
  unknown or *bg_q*, *fujitsu_fx10, …*
  System compiler to be the default for known systems

- Default compiler **vendor** for *unknown* is *gnu*

```
…
demo_cn_plain_CC:bg_q:::=bgxlc_r
demo_cn_plain_CC:unknown:gnu::=gcc
demo_cn_plain_CC:unknown:intel::=icc
demo_ln_CXX:bg_q:::=g++
demo_ln_CXX:unknown:gnu::=g++
demo_ln_CXX:unknown:intel::=icpc
demo_ln_CXXFLAGS:unknown:::=-O0 –g
…
# also <Compiler>FLAGS, CPPFLAGS,
# LDFLAGS, LIBS
```

Compiler + flags selection (2)

Deviate from defaults:

- using the configuartion file (settings need to provided in configuration file):
    - System *unknown*: switch via options
      `--with-demo-ln-compiler=(gnu|intel|pgi|...)`
      `--with-demo-cn-compiler=(gnu|intel|pgi|...)`
    - *Known* systems: have no vendor, but (default) target.
      Switch to different settings by providing another target, e.g. `bg_q_gnu`

- using unique variables:
  `./configure demo_cn_CXXFLAGS='-O3'`

- Changes to configuration file are reflected in `./configure --help` after `autoreconf`

MPI detection and selection

- Select from presets in package-specific configuration file:
  `unique_variable:system:vendor:wrapper:target=value`
- **Unique variable** names come from `AFS_MPI(languages)`

```
demo_cn_mpi_CC:unknown::openmpi:=mpicc
demo_cn_mpi_CXX:unknown::openmpi:=mpicxx
demo_cn_mpi_CC:bg_q:::=mpixlc_r
demo_cn_mpi_CXX:bg_q:::=mpixlcxx_r
```

- Known systems (*bg_q*, *fujitsu_fx10, …*) just use system name to select
- System *unknown*:
  - Search for MPI wrapper in PATH. Try to detect wrapper type. Use type to select
  - Request user interaction if several candidates found

```
  --with-demo-cn-mpi=(bullxmpi|hp|ibmpoe|intel|intel2|intel3|intelpoe|\
                      lam|mpibull2|mpich|mpich2|mpich3|openmpi|platform|\
                      scali|sgimpt|sun)
```

- Credits: JSC: Bernd Mohr, Brian Wylie, Markus Geimer; TUD: Matthias Jurenz

Makefile structure

- Usual autotools structure: one or more Makefile.am, often recursive
- AFS structure:
  - One non-recursive Makefile.am per subdirectory + a toplevel one.
  - Subdirectory Makefiles `include` one or more reservoir Makefiles. Reservoir Makefiles correspond to general structure; also take into account **shared targets** (`ln-cn_plain`)
  - Developer just deals with reservoir Makefiles
    - writes Makefile.am snippet anywhere in src-tree; decides where to build and includes accordingly
    - cross-compiling or `--with-target` given: shared targets are build for cn-plain and ln, otherwise just cn-plain

```
# Reservoir Makefiles include Makefile
# snippets
$ cat cn_mpi.inc.am
include ../src/compute/Makefile.mpi.inc.am
```

```
# Makefile.am per dir
.
├── bootstrap
├── build-cn-mpi
│   ├── configure.ac
│   └── Makefile.am
├── build-cn-plain
│   ├── configure.ac
│   └── Makefile.am
├── build-ln
│   ├── configure.ac
│   └── Makefile.am
├── configure.ac
└── Makefile.am
```

```
# Reservoir Makefiles,
# correspond to structure
.
├── cn_mpi.inc.am
├── cn_plain.inc.am
├── ln.inc.am
└── ln-cn_plain.inc.am
```

Communication between ‘boxes’

CPP defines and make variables

- Our primary use case: In-targets (e.g., config-tools) are interested in cn-plain/cn-mpi configuration
- Configuration means CPP defines or Makefile variables. Export in one subdir, import in another
  - CPP variables, replace `AC_DEFINE[_UNQUOTED]`:

```
# Export CPP defines

#AC_DEFINE([HAVE_FOO], [1], [Define if FOO exists])
AFS_DEFINE_AND_EXPORT([HAVE_FOO], [1], [Define if FOO exists])
# i.e., original + prefixed defines
AC_DEFINE([HAVE_FOO], [1], [Define if FOO exists])
AC_DEFINE([EXPORT_cn_plain_HAVE_FOO], [1], [Define if FOO exists])

# the EXPORT-* end up in config-export.h

# Import from any source file
#include "<path_to_top>/build-cn-plain/config-export.h"
```

  - Makefile variables (via `AC_SUBST`) , for completeness
    - `AFS_EXPORT_MAKE_VAR(variable, value)`
    - `AFS_IMPORT_MAKE_VARS([group-]name, directory)`

Communication between ‘boxes’ (2)
remote build dependencies

- Typical case: cn-plain builds "utilities" used by cn-mpi.
  - Can‘t trigger via `make -C <dir> <target>` directly: parallel make may fail
  - Collect all external targets and have a single rule building all external targets at once

```
AFS_CN_PLAIN_DIR = ../build-cn-plain/
$(AFS_CN_PLAIN_TARGETS): afs_build_external_targets

AFS_CN_PLAIN_TARGETS += $(AFS_CN_PLAIN_DIR)libutils.la

afs_build_external_targets:
    @targets=`echo $(AFS_CN_PLAIN_TARGETS) | \
    sed 's|^[ ^t]*||;s|[ ^]*$$||;s|$(AFS_CN_PLAIN_DIR)||g'`; \
    if test "x$$targets" != x; then \
        $(am__cd) $(AFS_CN_PLAIN_DIR) && $(MAKE) $(AM_MAKEFLAGS) $$targets; \
    fi
    # same for external targets from other directories
```

Header and library checks

- Problem: when cross-compiling or --with-target given, header/libraries from default search path usually wrong **for compute-node subdirectories**
- Our solution: provide wrapper around header/library check

Real header/lib check. Needs to provide additional variable:
`afs_lib_check_successful=yes/no`

```
AFS_CHECK_HEADER_AND_LIB_IFELSE([foo-bar], [DEMO_CHECK_LIB_FOO_BAR],
    [echo "cn-plain-foo-bar: success"], [echo "cn-plain-foo-bar: failure"])
```

  - let user provide path explicitly, otherwise ignore the header/library check for compute-node

```
 --with-demo-cn-plain-libfoo-bar[=yes|no|<Path to libfoo-bar installation>]
                         If you want to build with libfoo-bar support but do
                         not have a libfoo-bar in a standard location, you
                         need to explicitly specify libfoo-bar's installation
                         directory. On non-cross-compile systems and when the
                         …
  --with-demo-cn-plain-libfoo-bar-include=<Path to libfoo-bar headers>
  --with-demo-cn-plain-libfoo-bar-lib=<Path to libfoo-bar libraries>
```

  - maintains CPPFLAGS, LDFLAGS, LIBS

AFS_SUMMARY

- Problem: configure output quite chatty: cannot see the wood for the trees
  - even worse with nested configures

- Provide summary output at end of toplevel configure:
  - AFS_SUMMARY([description], [value])
  - AFS_SUMMARY_SECTION_BEGIN/END
  - AFS_SUMMARY_PUSH/POP
  - AFS_WARN (future work)

- Credits: Bert Wesarg, TUD

```
Configure command:
  ../configure                              '--prefix=/opt/demo'

Loaded modules:
  module load                               afs-dev/02 \
                                            scorep-dev/06

Configuration summary:
  Demo trunk [Toplevel]:
    HPC system:                             unknown
    Cross compile system:                   no
    Build CPU:                              x86_64
    Build OS:                               linux-gnu

    Demo [Compute node MPI]:
      MPI implementation:                   openmpi (detected)
      Compiler variables:
        demo_cn_mpi_CC:                     mpicc
        demo_cn_CFLAGS:                     -O3 -g
        demo_cn_mpi_CXX:                    mpicxx
        demo_cn_CXXFLAGS:                   -O1 -g

    Demo [Compute node plain]:
      Compiler variables:
        demo_cn_plain_CC:                   gcc
        demo_cn_CFLAGS:                     -O3 –g
…
```

Development requirements

- AFS can be used with
    - Latest autoconf-2.69 and libtool-2.4.6
    - automake >= 1.13.4
      we use 1.13.4, just to prevent annoying warnings
      (to be fixed in next automake release)

- libtool patches necessary for some complier/MPI combinations, GNU and Intel are fine without patches

- We provide afs-dev package with patched autotools
    - Use this to build release tarballs

- Requirements on user side: a Unix-like environment, nothing else

Future work

- Apply AFSv2 prototype to SIONlib
  - polish, fix, add consistency check
  - convenience additions
  - code restructuring
  - identify and isolate customization points
  - extensive, automated testing

- Release on github/gitlab
  - mostly 3-clause BSD
  - modified autoconf macros under GPL with exception
    (generated configure scripts to be distributed under any license)

- Apply v2 to v1 packages (as resources allow)

Acknowledgements

- Bert Wesarg, TUD
- Markus Geimer, JSC
- Bernd Mohr, JSC
- Brian Wylie, JSC
- Pavel Saviankou, JSC
- Andreas Beckmann, JSC
- Ronny Tschüter, TUD
- Matthias Jurenz, TUD
- Orion Poplawski, NWRA

Questions?