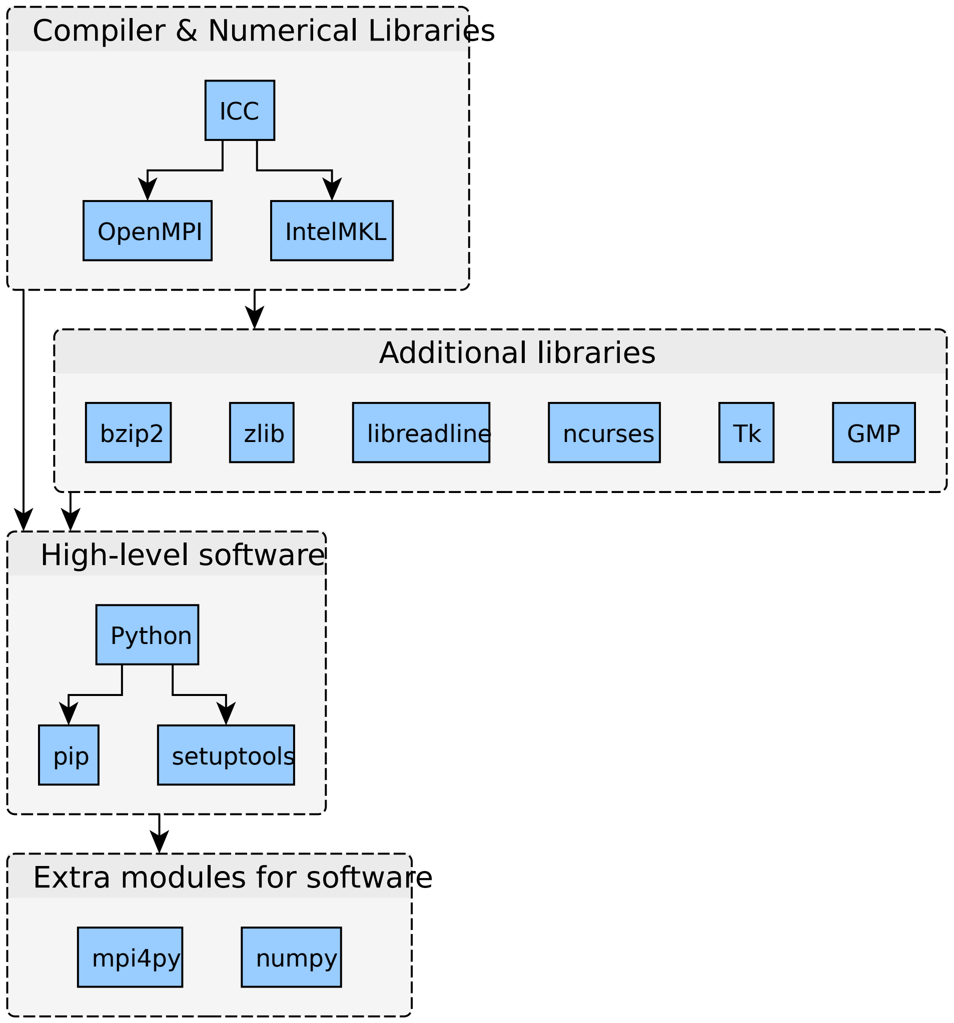
Triton software deployment & modules environment

Aalto University
School of Science

Software: Basic concepts of software deployment

- Modern software in Linux systems is typically not self-contained
- They depend on libraries and/or other software to work
- Usually programs are compiled dynamically against these dependencies
  - → This means that dependencies are not included in the resulting binaries, but they are loaded when the program runs
- The end result is a software stack where end product requires many layers of other software
- To provide these full software stacks, Triton uses *environment modules*

Software: Example software stack



Aalto University
School of Science

Software: Environment variables

- In Linux systems environment variables define other paths than system paths
- Most important for you:
  a) $PATH is the command lookup path
  b) $LD_LIBRARY_PATH is the runtime library lookup path
- Environment modules set these (and other variables)

  That is all they do!

Modules: Basics

- Specialized software in Triton is installed by admins and organized as *environment modules*
- When you load a module, the environment variables defined by the module will be loaded to the current shell
- This enables different versions of software to be installed at a same time for all users
- We admins will install widely used software as modules
- CSC provides their set of environment modules via their CVMFS (CernVM File System) server
- In future we might use specialized containers to allow higher software stacks

Modules: Commands

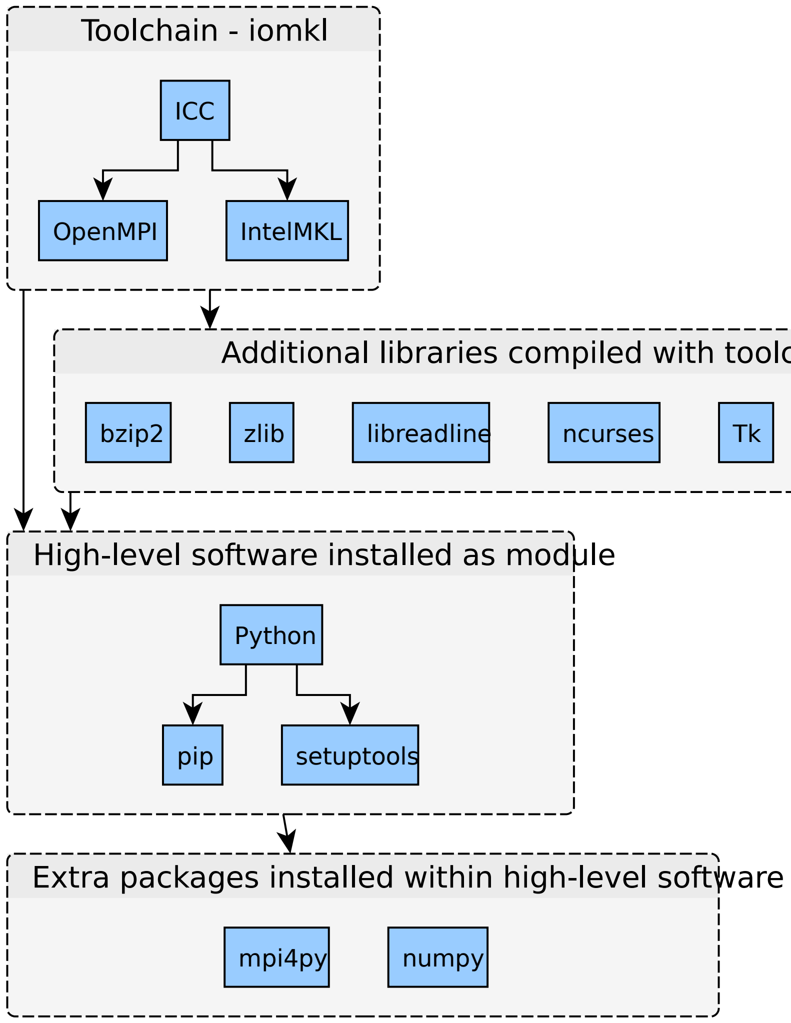
- module avail [name/word/reg exp] and module spider [name/word/reg exp] :
  Show/search from available module(s) (output between commands differ a bit)
- module show name[/version] :
  Show what a module does
- module load name[/version] :
  Load a module
- module unload name[/version] :
  Unload a module
- module list [name] :
  Show loaded module(s)
- module swap old[/version] new[/version] :
  Swap loaded module(s)
- module purge :
  Unload all modules
- module save [name] :
  Save a module collection
- module savelist :
  List stored collections
- module restore [name] :
  Load a module collection

Just run 'module' for more information

Modules: Organization

- Many of the modules are organized by *toolchains*
- Toolchains are the lowest layer of compilers/numerical libraries that are used as a starting point for higher level software
- Advanced software is compiled against a specific toolchain e.g. Python/2.7.13-iomkl-triton-2017a is compiled against the iomkl-toolchain
- Toolchains with ‘triton’ in them are maintained by us and updated to newest version twice a year (2016b, 2017a, 2017b, …)

Modules: Example software stack


Toolchain - iomkl
ICC
OpenMPI
IntelMKL
Additional libraries compiled with toolchain
bzip2
zlib
libreadline
ncurses
Tk
GMP
High-level software installed as module
Python
pip
setuptools
Extra packages installed within high-level software
mpi4py
numpy

Aalto University
School of Science

Modules: List of toolchains

- Naming convention → compiler, mpi, blas, lapack, fftw, cuda
  Examples:

Toolchain	Compiler	MPI	BLAS	LAPACK	FFTW	CUDA
gompi	GCC	OpenMPI	-	-	-	-
goolf	GCC	OpenMPI	OpenBLAS	LAPACK	FFTW	-
gmvolf	GCC	MVAPICH	OpenBLAS	LAPACK	FFTW	-
goolfc	GCC	OpenMPI	OpenBLAS	LAPACK	FFTW	CUDA
ioolf	icc	OpenMPI	OpenBLAS	LAPACK	FFTW	-
iomkl	icc	OpenMPI	Intel MLK	Intel MLK	Intel MLK	-

Modules: Other software

- matlab: module load matlab
- R: module load R
- GROMACS: module load GROMACS
- ...

Modules: Python toolchain or Anaconda

- Toolchain-type Python versions are mainly used for
  a) Multiprocess Python with MPI
  b) GPU requiring code
- If you do not use either, use Anaconda:

```
module load anaconda2 or
module load anaconda3
```

Aalto University
School of Science

Modules: Important notes

- Do not mix and match different toolchains
  → Library paths will most likely go awry
- If you have loaded modules when you build/install software, remember to load the same modules when you run the software (also in Slurm jobs)
- If you just load a module without specifying the version, remember that the default might have change since your last use
- Once you got a good collection of modules, save them into collections
  → Collections are faster to load and easier to maintain

Modules: Is this the correct program?

- When in doubt about which program/library you're about to use:
  - → Check your modules with 'module list'
  - → Command 'env' shows the current environment
  - → Command 'which' accompanied with a program name shows the full path to the command ($PATH lookup)
  - → Command 'ldd' accompanied with a program/library name shows which dynamic libraries the program needs and where the dynamic loader has found them ($LD_LIBRARY_PATH lookup)

Questions or comments regarding software in Triton?

Modules & Software: Exercises

Start all examples from empty (module purge) state.

1. Load one of the toolchain modules.
   List what modules it loaded.
2. Save your environment variables to a file with 'env > filename'.
   Swap a module, save variables to a new file and use 'diff' to check for changes.
3. Load a module with multiple dependencies e.g. R/3.3.2-iomkl-triton-2017a-libX11-1.6.3. Save the loaded modules as a collection.
4. Use 'time module load <module>'/'time module restore <collection>' to compare load times. Did you see a speedup?
5. Load GROMACS. Use 'which' to find where command 'gmx' is and then use 'ldd' to find out what libraries it uses. Load incompatible toolchain e.g. goolf. Check ldd output again.