make and Makefiles

Ross Dickson, Computational Research Consultant

Building 3rd-party code

- Common idiom:

```
Unpack the downloaded file into directory
cd directory
Customize Makefile to your local conditions
make
make install
```

- How do you “customize the Makefile”?
- What if it fails at “make” or “make install”?

Want to follow along?

```
$ cp /home/rdickson/demo/Make/* .
$ ls
eispack.f   linpack.f         parnum.f
headers.f   Makefile.final    scfnum.f
input.f     Makefile.v1       scfnumx.f
lebedev.f   Makefile.v2       xc.f

$ time mpif90 -fast -g *.f \
  -llapack -lblas -o parnum
```

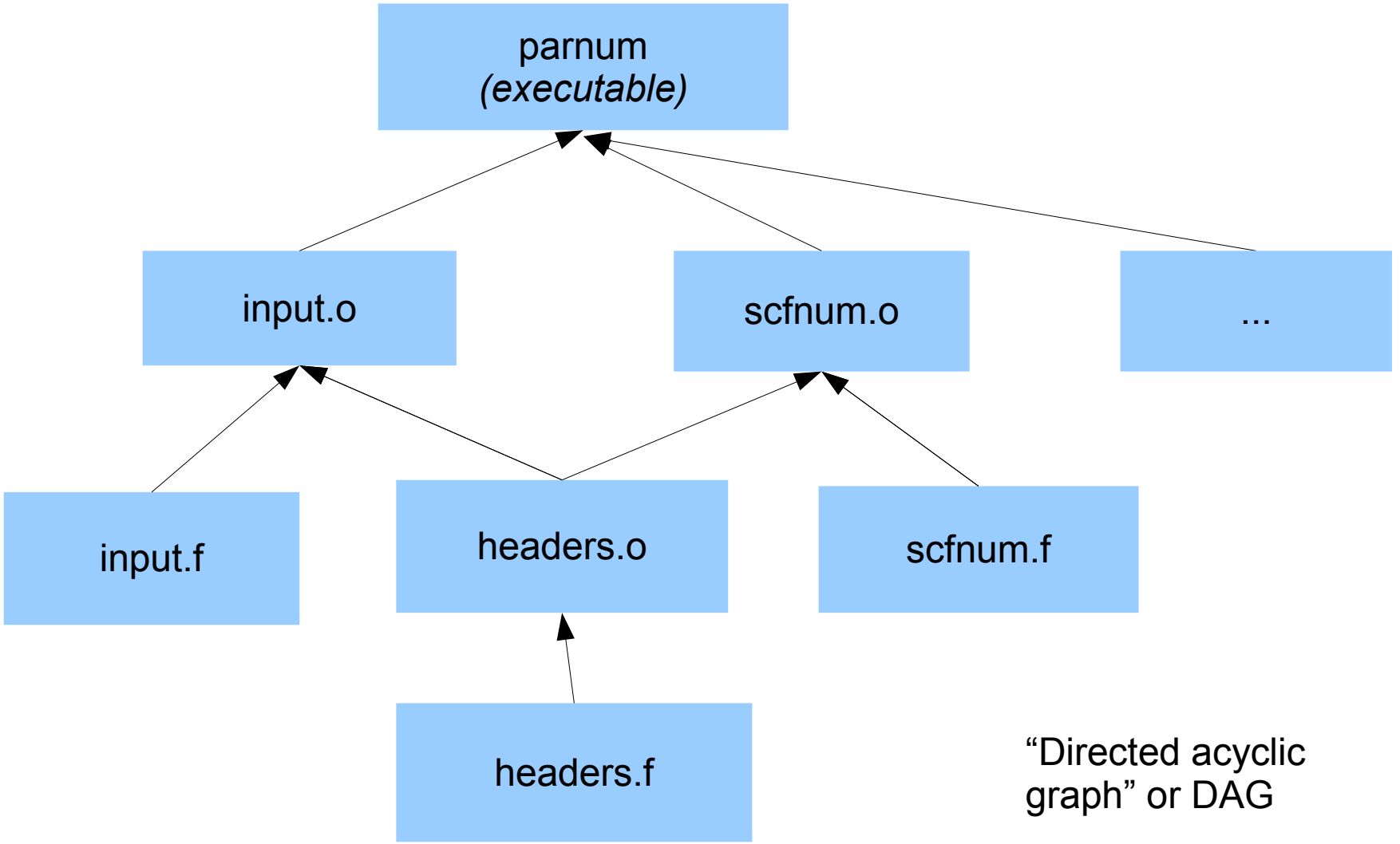
Independent compilation

Two steps:

- **compile:** Generate .o file from .c, .f, .cxx, ...
- **link:** Combine .o files and libs into executable
- Same program does both, use '-c' to only compile
- Save time by doing only necessary steps

```
$ mpif90 -fast -g -c linpack.f
$ mpif90 -fast -g -c headers.f
$ mpif90 -fast -g -c input.f
$ ...
$ mpif90 -fast -g *.o -llapack -lblas \
  -o parnum
```

Dependencies



"Directed acyclic graph" or DAG

make rule syntax

```
input.o: input.f headers.o
        mpif90 -fast -f -c input.f

target: prereq ...
--tab-- recipe
```

- First *target* in file is default
- *Recipe* must be preceded by *tab*, not spaces
- Recipe consists of shell commands

Invoking **`make`**

```
$ make -f makefile target
```

- If -f *makefile* not supplied, looks for 'Makefile'
- If *target* not supplied, takes first target in file
- -n for debugging – does not execute, just prints what will happen

Variables

```
FFLAGS = -fast -g
FC = mpif90

input.o: input.f headers.o
        $(FC) $(FFLAGS) -c input.f
```

- Spaces in definition okay
- Parentheses required in substitution
- Not bash syntax!
- Many variables have conventional meanings, *e.g.* CC “C Compiler”, FFLAGS “Fortran flags”

Prefix rules

```
.f.o:
        $(FC) $(FFLAGS) -c -o $@ $<
```

- "Produce a .o file from any .f file"
- Automatic variables:
  - $@ – name of target ("thing.o")
  - $< – name of first prereq ("thing.f")
  - there are lots more...
- See also Pattern rules
  - Similar idea, but with % for wild card
- There are implicit rules for common situations

More stuff...

```
clean:
        -rm *.o *.mod
```

- No prereq? Recipe always invoked
- Ignore errors with prefix “-”

```
realclean: clean
        -rm parnum
```

- Prereq not a file? Prereq always invoked.
  Recipe always invoked.

Better than make?

- More modern build-automation tools:
  - SCons
  - Ant
  - …
- Research automation tools:
  - Sweave
  - knitr
  - ...

References

- http://www.gnu.org/software/make/manual/make.html
- Example code in **`/home/rdickson/demo/Make/*`**
- http://software-carpentry.org/v4/make/index.html