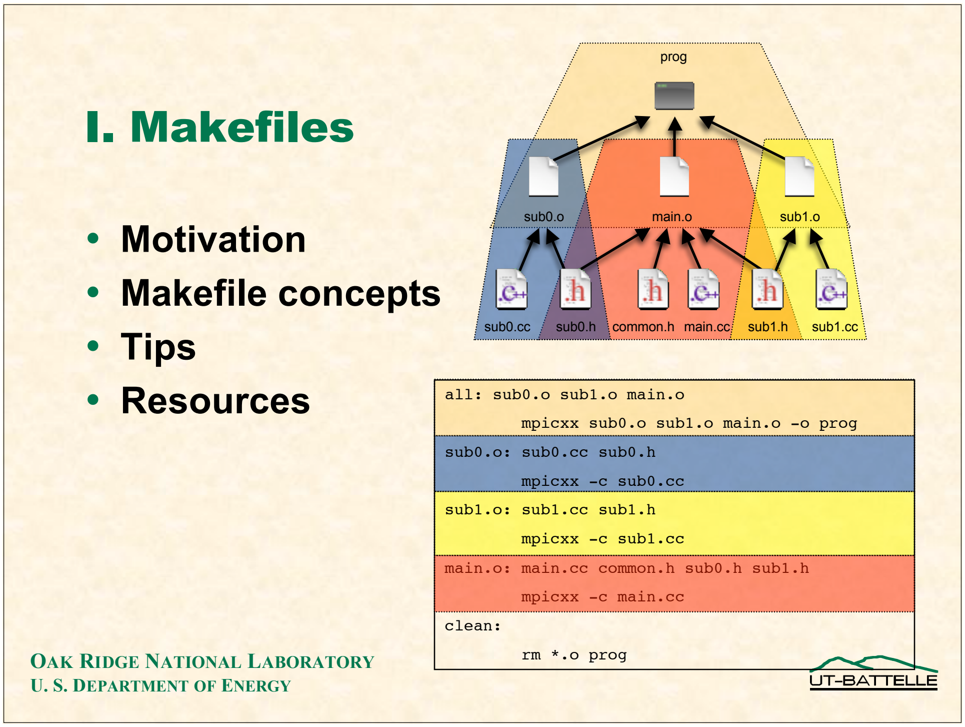
A Crash Course in Supercomputing: Makefiles and Batch Scripts

Rebecca Hartman-Baker
Oak Ridge National Laboratory
hartmanbakrj@ornl.gov

Outline

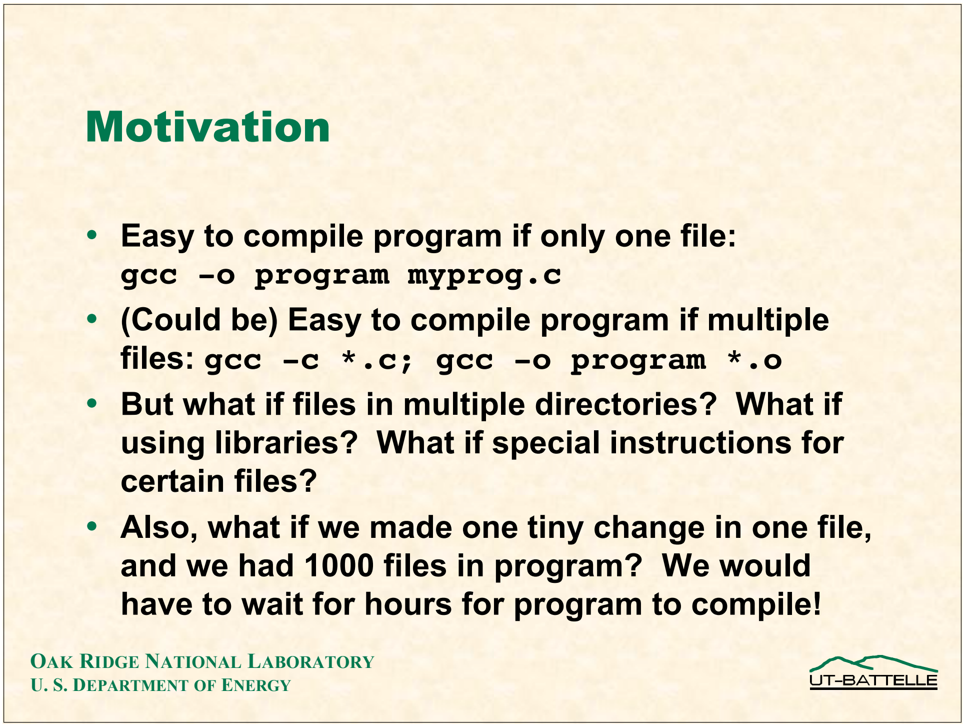
I. Makefiles
II. Batch Scripts



I. Makefiles

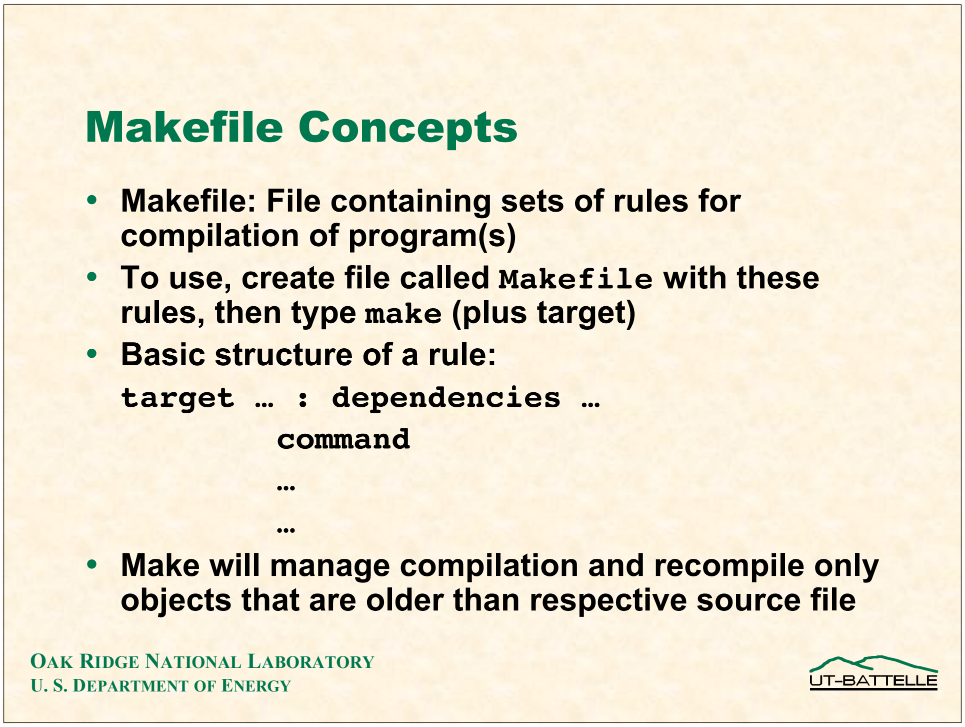
- Motivation
- Makefile concepts
- Tips
- Resources

```
all: sub0.o sub1.o main.o
        mpicxx sub0.o sub1.o main.o -o prog
sub0.o: sub0.cc sub0.h
        mpicxx -c sub0.cc
sub1.o: sub1.cc sub1.h
        mpicxx -c sub1.cc
main.o: main.cc common.h sub0.h sub1.h
        mpicxx -c main.cc
clean:
        rm *.o prog
```



Motivation

- Easy to compile program if only one file: `gcc -o program myprog.c`
- (Could be) Easy to compile program if multiple files: `gcc -c *.c; gcc -o program *.o`
- But what if files in multiple directories? What if using libraries? What if special instructions for certain files?
- Also, what if we made one tiny change in one file, and we had 1000 files in program? We would have to wait for hours for program to compile!



Makefile Concepts

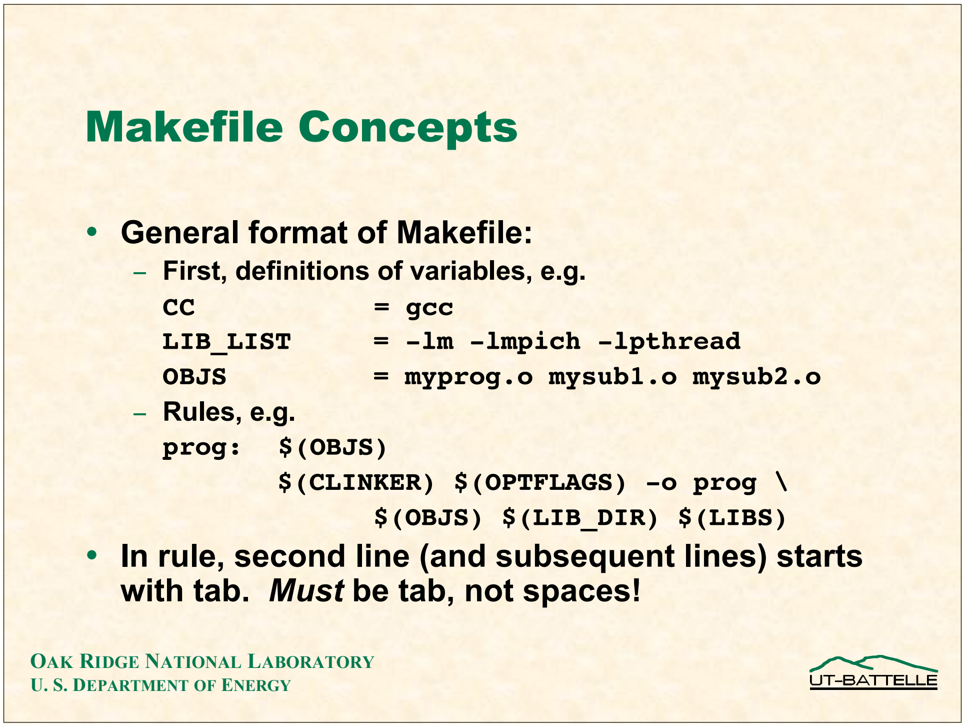
- Makefile: File containing sets of rules for compilation of program(s)
- To use, create file called `Makefile` with these rules, then type `make` (plus target)
- Basic structure of a rule:

```
target … : dependencies …
           command
           …
           …
```

- Make will manage compilation and recompile only objects that are older than respective source file

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

UT-BATTELLE



Makefile Concepts

- General format of Makefile:
  - First, definitions of variables, e.g.

```
CC             = gcc
LIB_LIST       = -lm -lmpich -lpthread
OBJS           = myprog.o mysub1.o mysub2.o
```

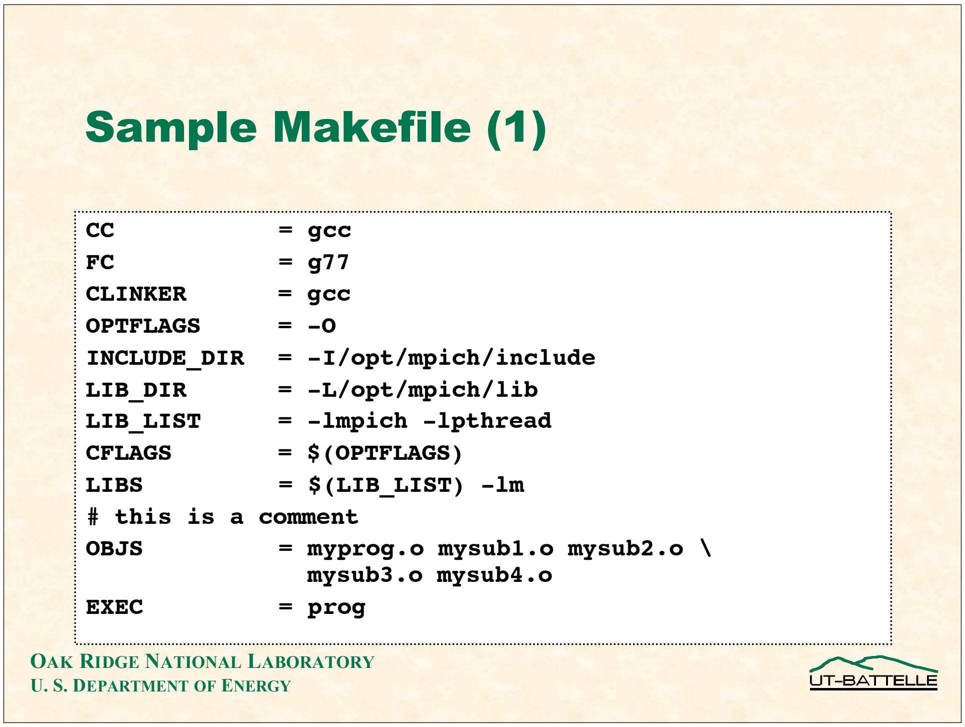
  - Rules, e.g.

```
prog:   $(OBJS)
        $(CLINKER) $(OPTFLAGS) -o prog \
              $(OBJS) $(LIB_DIR) $(LIBS)
```

- In rule, second line (and subsequent lines) starts with tab. *Must* be tab, not spaces!

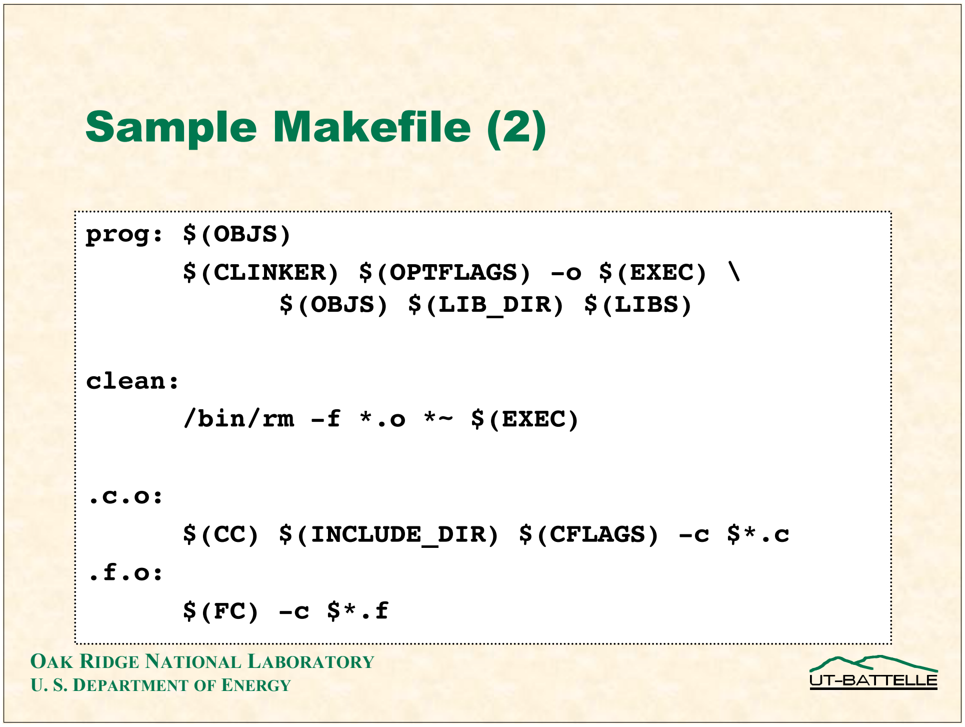
OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

UT-BATTELLE



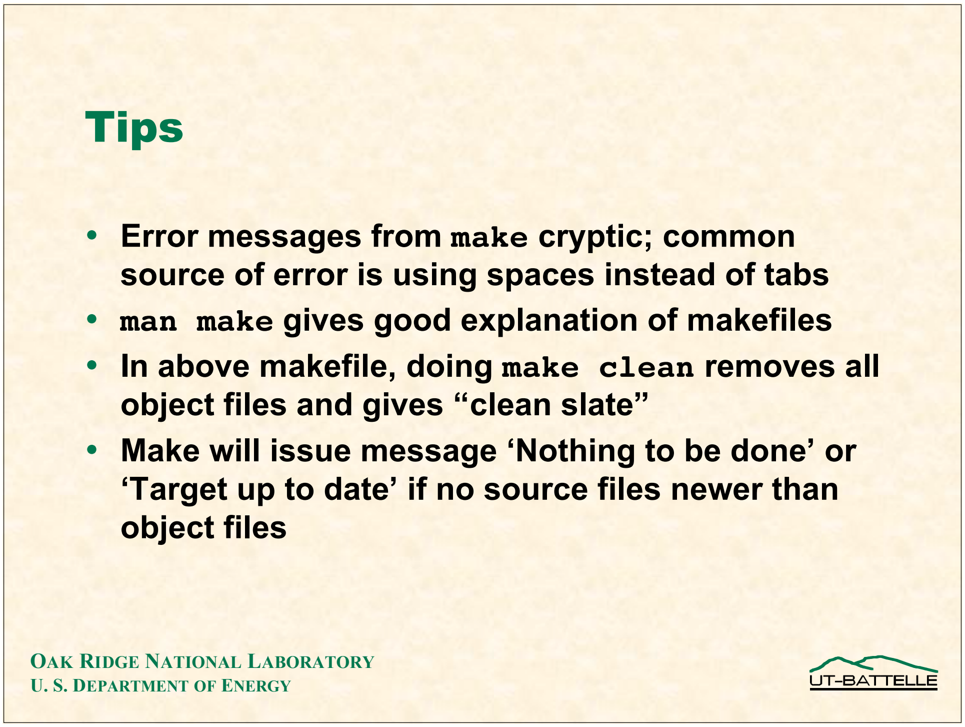
Sample Makefile (1)

```
CC              = gcc
FC              = g77
CLINKER         = gcc
OPTFLAGS        = -O
INCLUDE_DIR     = -I/opt/mpich/include
LIB_DIR         = -L/opt/mpich/lib
LIB_LIST        = -lmpich -lpthread
CFLAGS          = $(OPTFLAGS)
LIBS            = $(LIB_LIST) -lm
# this is a comment
OBJS            = myprog.o mysub1.o mysub2.o \
                  mysub3.o mysub4.o
EXEC            = prog
```



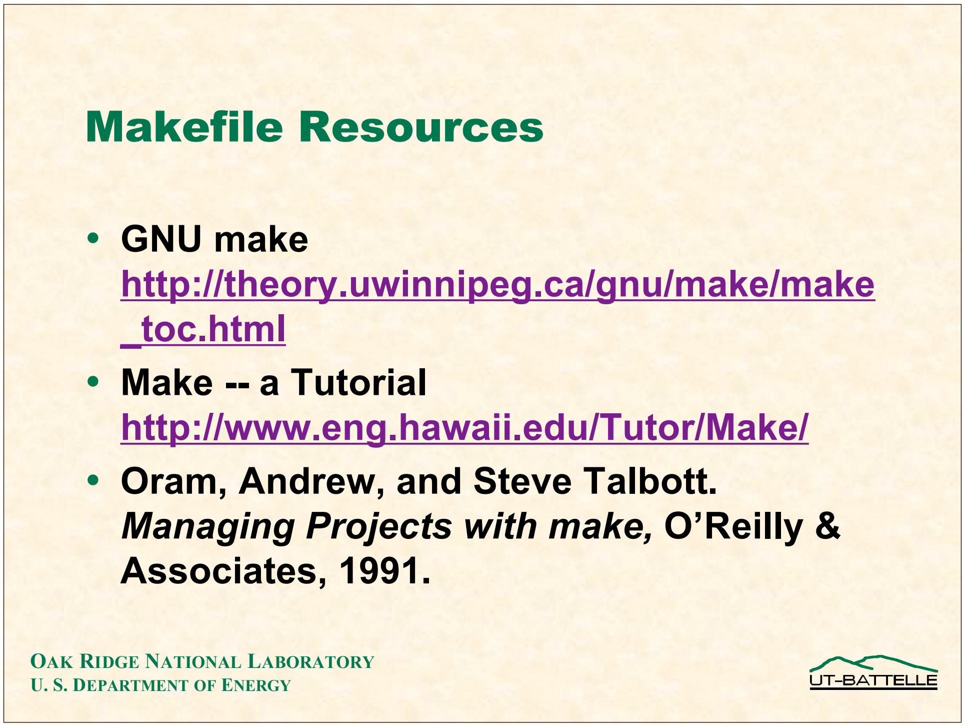
Sample Makefile (2)

```
prog: $(OBJS)
      $(CLINKER) $(OPTFLAGS) -o $(EXEC) \
            $(OBJS) $(LIB_DIR) $(LIBS)

clean:
      /bin/rm -f *.o *~ $(EXEC)

.c.o:
      $(CC) $(INCLUDE_DIR) $(CFLAGS) -c $*.c
.f.o:
      $(FC) -c $*.f
```



Tips

- Error messages from `make` cryptic; common source of error is using spaces instead of tabs
- `man make` gives good explanation of makefiles
- In above makefile, doing `make clean` removes all object files and gives “clean slate”
- Make will issue message ‘Nothing to be done’ or ‘Target up to date’ if no source files newer than object files

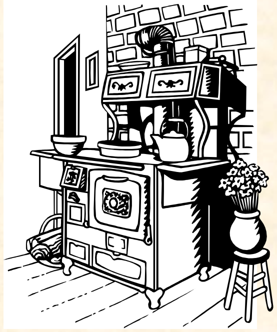


Makefile Resources

- GNU make http://theory.uwinnipeg.ca/gnu/make/make_toc.html
- Make -- a Tutorial http://www.eng.hawaii.edu/Tutor/Make/
- Oram, Andrew, and Steve Talbott. *Managing Projects with make,* O’Reilly & Associates, 1991.

II. Batch Scripts

- Batch system and Scheduling
- Concepts
- Useful commands
- Further help



Batch System and Scheduling

- Supercomputer: powerful computer consisting of many interlinked CPUs
- Users competing for computational resources
- How to launch and schedule jobs fairly?
- Job can run without user presence
- Must not allow one user to hog resources

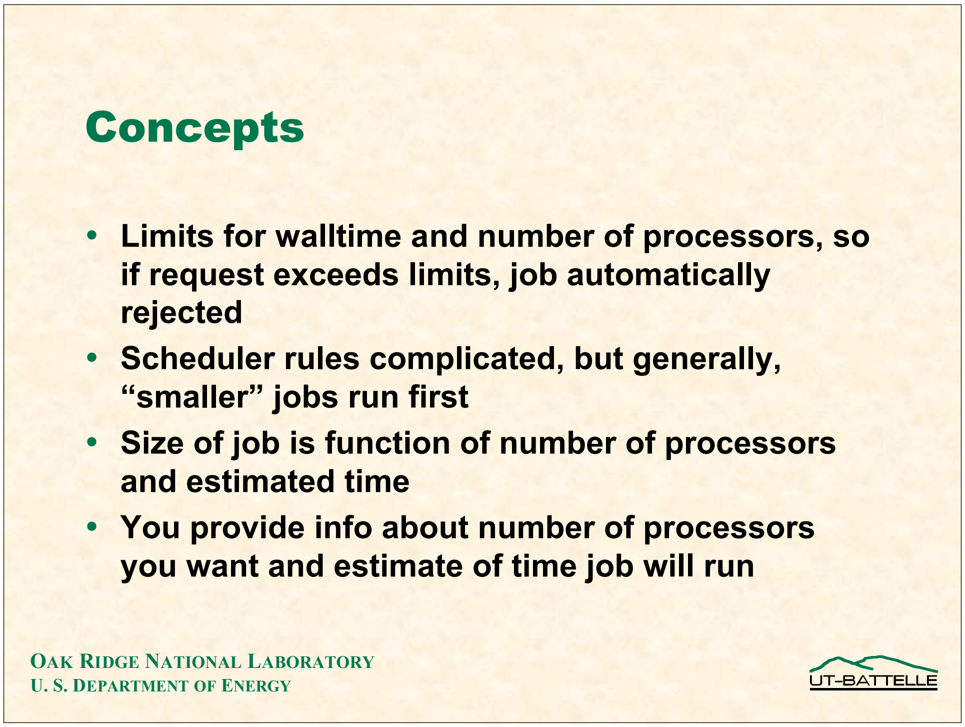
Batch System

- Batch system accepts input jobs into queue and launches them when resources available
- Many machines use batch system PBS (*P*ortable *B*atch *S*ystem)
- PBS developed for NASA in 1990s

Scheduler

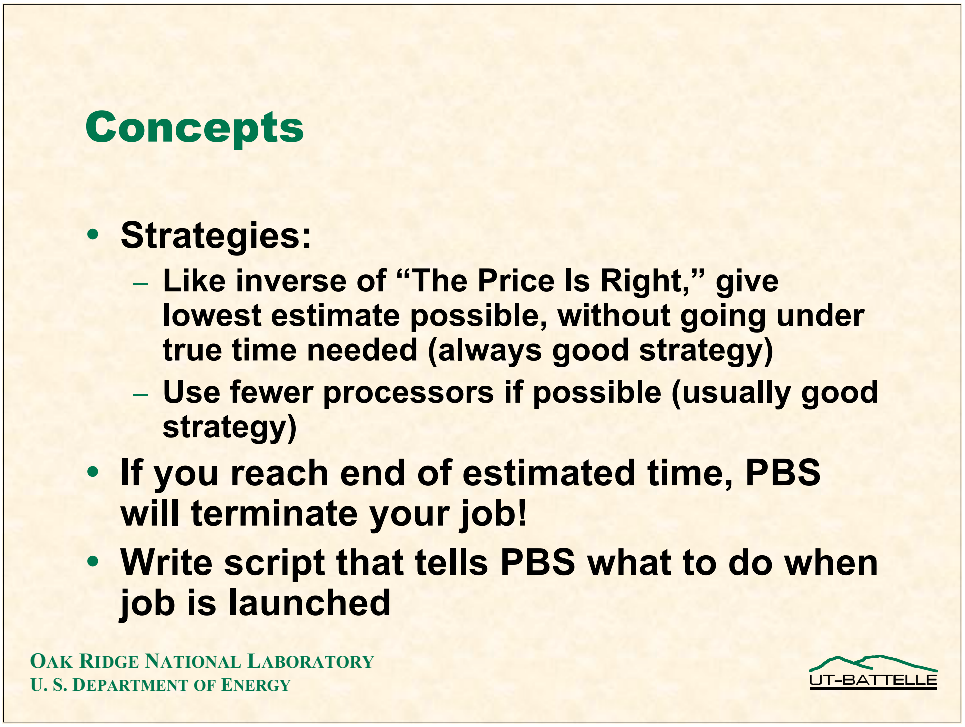
- Scheduler decides when jobs can be run based on scheduling policies, e.g. user priority, length of job, number of nodes requested, length of time in queue
- Many machines use Maui Scheduler
- Maui Scheduler extensively developed, supported by large segment of computation community including U.S. Dept. of Energy, NCSA

(source: www.the-hawaii-vacation-guide.com)



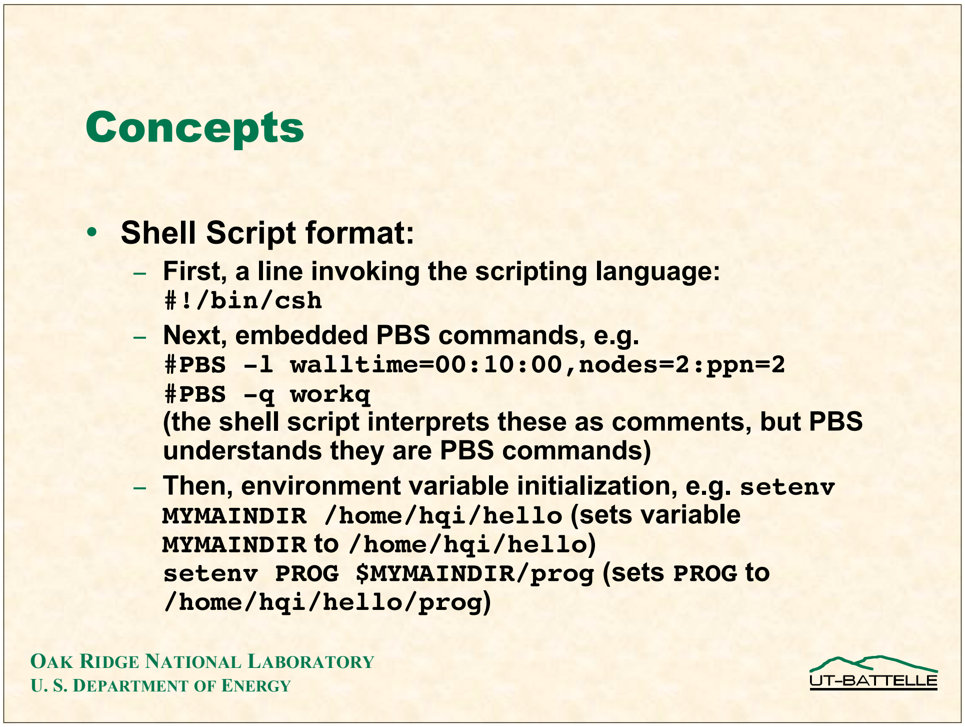
Concepts

- Limits for walltime and number of processors, so if request exceeds limits, job automatically rejected
- Scheduler rules complicated, but generally, “smaller” jobs run first
- Size of job is function of number of processors and estimated time
- You provide info about number of processors you want and estimate of time job will run



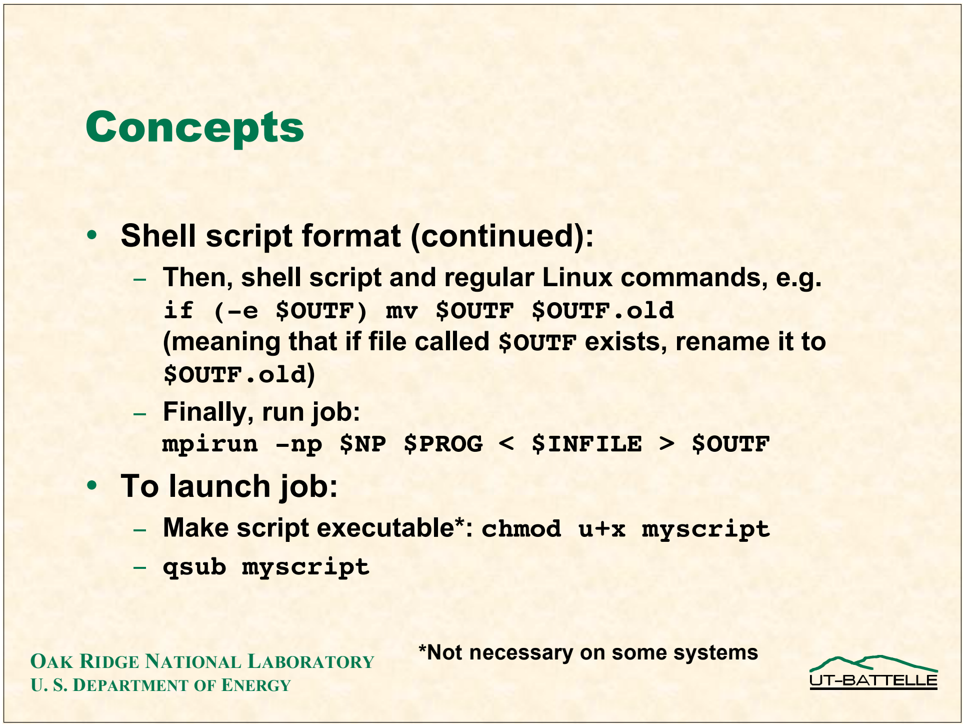
Concepts

- Strategies:
  - Like inverse of “The Price Is Right,” give lowest estimate possible, without going under true time needed (always good strategy)
  - Use fewer processors if possible (usually good strategy)
- If you reach end of estimated time, PBS will terminate your job!
- Write script that tells PBS what to do when job is launched



Concepts

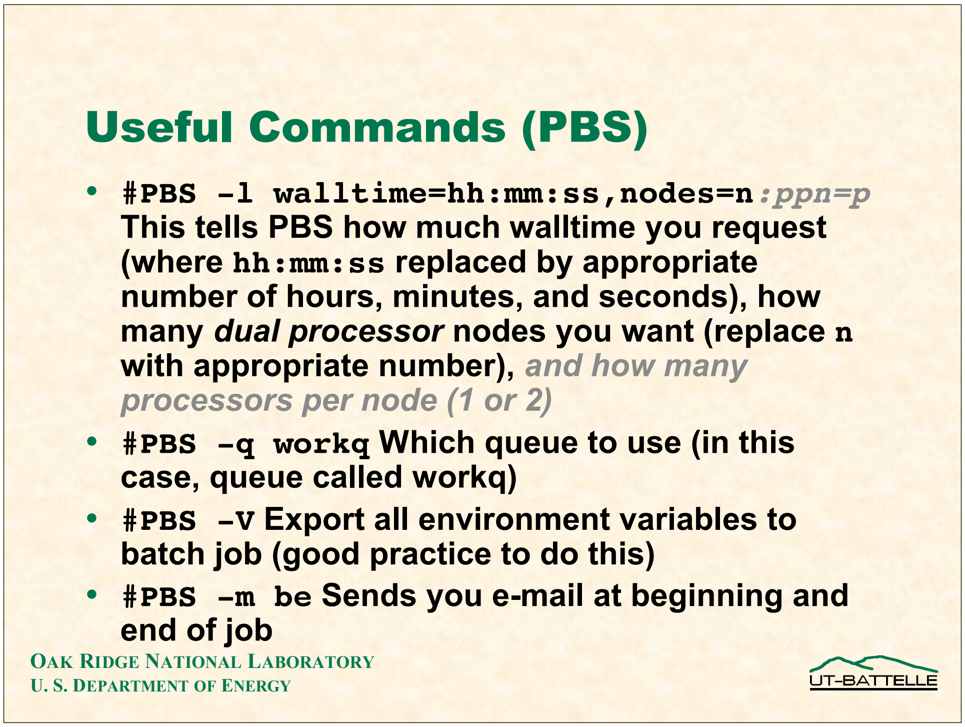
- Shell Script format:
  - First, a line invoking the scripting language:
    ```
    #!/bin/csh
    ```
  - Next, embedded PBS commands, e.g.
    ```
    #PBS -l walltime=00:10:00,nodes=2:ppn=2
    #PBS -q workq
    ```
    (the shell script interprets these as comments, but PBS understands they are PBS commands)
  - Then, environment variable initialization, e.g. `setenv MYMAINDIR /home/hqi/hello` (sets variable `MYMAINDIR` to `/home/hqi/hello`)
    `setenv PROG $MYMAINDIR/prog` (sets `PROG` to `/home/hqi/hello/prog`)



Concepts

- Shell script format (continued):
  - Then, shell script and regular Linux commands, e.g.
    ```
    if (-e $OUTF) mv $OUTF $OUTF.old
    ```
    (meaning that if file called `$OUTF` exists, rename it to `$OUTF.old`)
  - Finally, run job:
    ```
    mpirun -np $NP $PROG < $INFILE > $OUTF
    ```
- To launch job:
  - Make script executable*: `chmod u+x myscript`
  - `qsub myscript`

*Not necessary on some systems

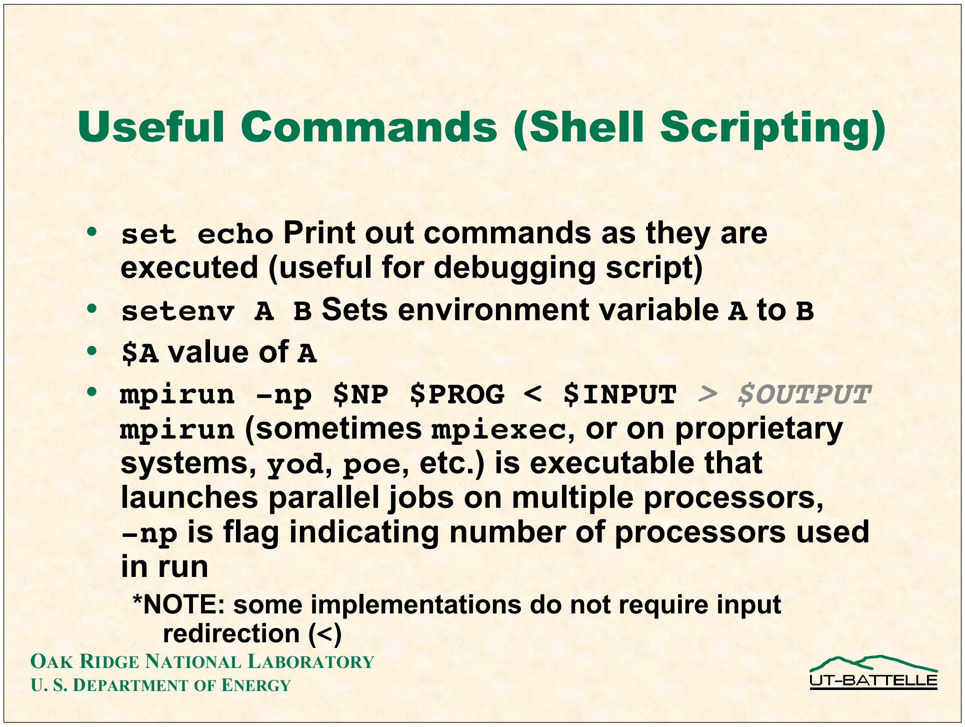


Useful Commands (PBS)

- `#PBS -l walltime=hh:mm:ss,nodes=n`*`:ppn=p`* This tells PBS how much walltime you request (where `hh:mm:ss` replaced by appropriate number of hours, minutes, and seconds), how many *dual processor* nodes you want (replace `n` with appropriate number), *and how many processors per node (1 or 2)*
- `#PBS -q workq` Which queue to use (in this case, queue called workq)
- `#PBS -V` Export all environment variables to batch job (good practice to do this)
- `#PBS -m be` Sends you e-mail at beginning and end of job

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

UT-BATTELLE



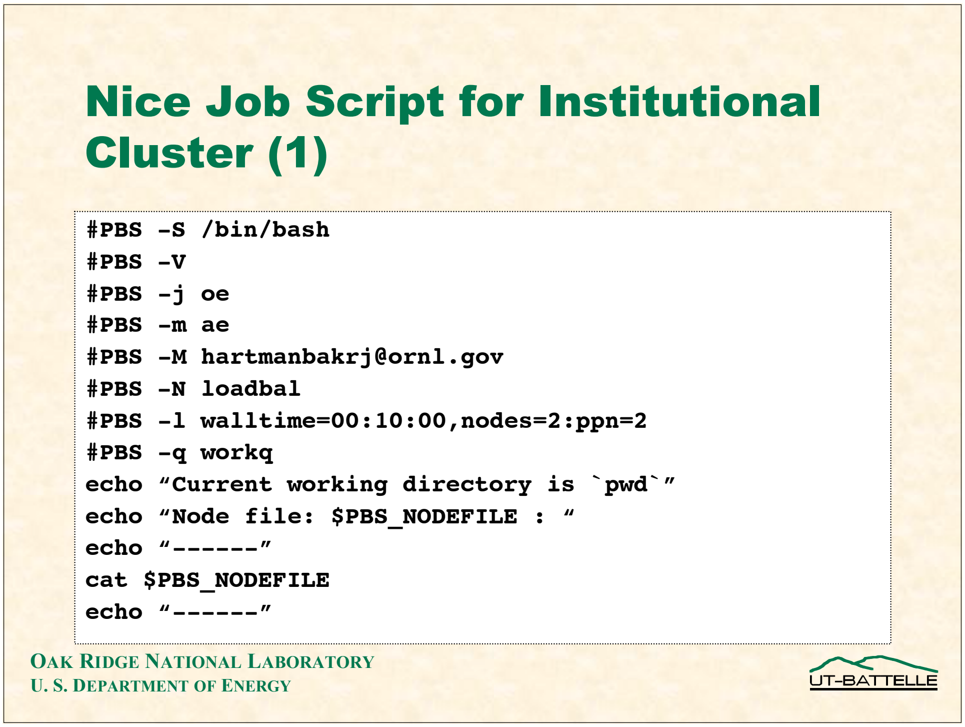
Useful Commands (Shell Scripting)

- `set echo` Print out commands as they are executed (useful for debugging script)
- `setenv A B` Sets environment variable `A` to `B`
- `$A` value of `A`
- `mpirun -np $NP $PROG < $INPUT` *`> $OUTPUT`*
  `mpirun` (sometimes `mpiexec`, or on proprietary systems, `yod`, `poe`, etc.) is executable that launches parallel jobs on multiple processors, `-np` is flag indicating number of processors used in run

  *NOTE: some implementations do not require input redirection (<)

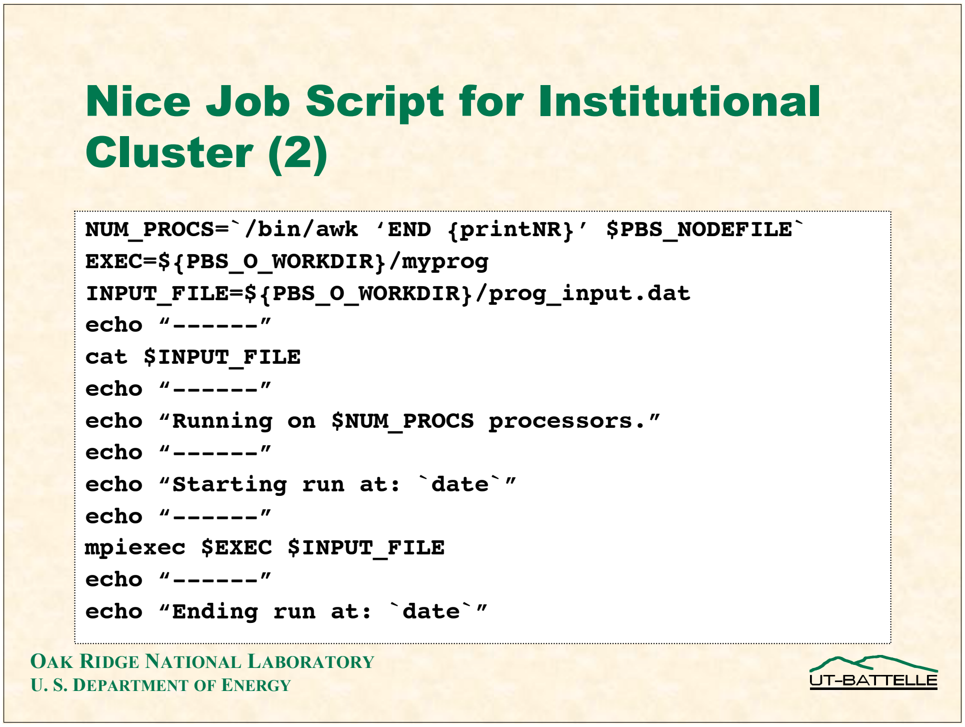
OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

UT-BATTELLE



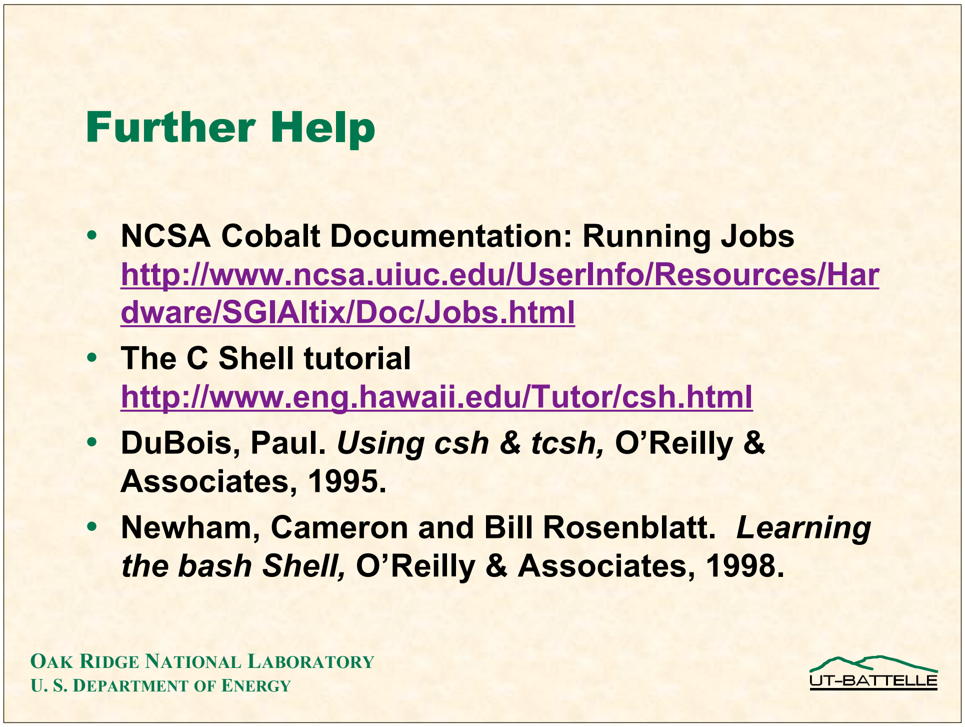
Nice Job Script for Institutional Cluster (1)

```
#PBS -S /bin/bash
#PBS -V
#PBS -j oe
#PBS -m ae
#PBS -M hartmanbakrj@ornl.gov
#PBS -N loadbal
#PBS -l walltime=00:10:00,nodes=2:ppn=2
#PBS -q workq
echo “Current working directory is `pwd`”
echo “Node file: $PBS_NODEFILE : “
echo “------”
cat $PBS_NODEFILE
echo “------”
```



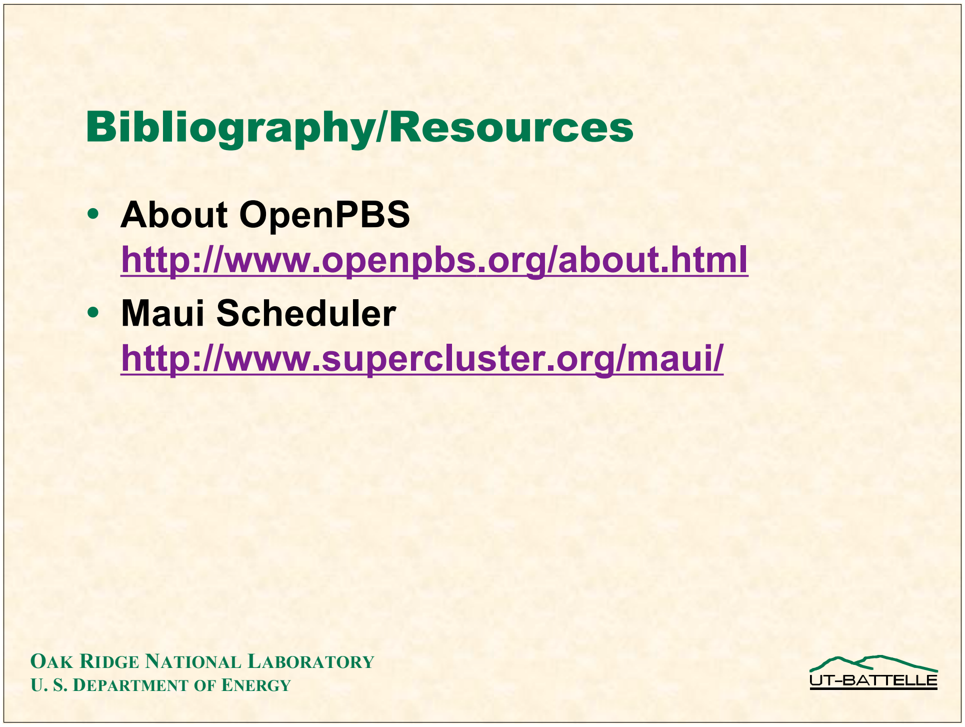
Nice Job Script for Institutional Cluster (2)

```
NUM_PROCS=`/bin/awk ‘END {printNR}’ $PBS_NODEFILE`
EXEC=${PBS_O_WORKDIR}/myprog
INPUT_FILE=${PBS_O_WORKDIR}/prog_input.dat
echo “------”
cat $INPUT_FILE
echo “------”
echo “Running on $NUM_PROCS processors.”
echo “------”
echo “Starting run at: `date`”
echo “------”
mpiexec $EXEC $INPUT_FILE
echo “------”
echo “Ending run at: `date`”
```



Further Help

- NCSA Cobalt Documentation: Running Jobs http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/SGIAltix/Doc/Jobs.html
- The C Shell tutorial http://www.eng.hawaii.edu/Tutor/csh.html
- DuBois, Paul. *Using csh & tcsh,* O'Reilly & Associates, 1995.
- Newham, Cameron and Bill Rosenblatt. *Learning the bash Shell,* O'Reilly & Associates, 1998.



Bibliography/Resources

- About OpenPBS http://www.openpbs.org/about.html
- Maui Scheduler http://www.supercluster.org/maui/