# Examples - Pegasus WMS

- Outline
  1. Set up for examples
     - *Work on Pegasus head node (hpc-pegasus.usc.edu)*
  2. Generate an example workflow
     - *Refer to tutorial for steps (https://pegasus.isi.edu/tutorial/usc/tutorial.php)*
  3. Modify the example pipeline to create a real-world workflow
     - *Show how this pipeline can be used to submit jobs in parallel*
  4. Generate an example merge workflow
     - *Show how this example can be modified to extend the real-world workflow*

# Examples - Pegasus WMS

- Outline
  1. Set up for examples
     - *Work on Pegasus head node (hpc-pegasus.usc.edu)*
  2. Generate an example workflow
     - *Refer to tutorial for steps (https://pegasus.isi.edu/tutorial/usc/tutorial.php)*
  3. Modify the example pipeline to create a real-world workflow
     - *Show how this pipeline can be used to submit jobs in parallel*
  4. Generate an example merge workflow
     - *Show how this example can be modified to extend the real-world workflow*

# Setup for examples

- Work on Pegasus head node

  $ ssh <netID>@hpc-pegasus.usc.edu

  - Pegasus is installed and running here

You type this blue text*

- Create a working directory in your project directory

  $ mkdir /home/rcf-proj/<your project>/pegasus

- CD there, then copy the HPC workshop files

  $ cd /home/rcf-proj/<your project>/pegasus

  $ cp /home/rcf-proj/workshop/pegasus/* .

- Extract "tarball"

  $ tar xvf fsl-pipeline.tar

* Modified examples will be available in completed form, in the workshop tar files

<= Don't forget the dot!
* (wildcard) matches everything
. (dot) resolves to **current directory**

# Examples - Pegasus WMS

- Outline
  1. Set up for examples
     - *Work on Pegasus head node (hpc-pegasus.usc.edu)*
  2. Generate an example workflow
     - *Refer to tutorial for steps (https://pegasus.isi.edu/tutorial/usc/tutorial.php)*
  3. Modify the example pipeline to create a real-world workflow
     - *Show how this pipeline can be used to submit jobs in parallel*
  4. Generate an example merge workflow
     - *Show how this example can be modified to extend the real-world workflow*

# Split workflow

- Let's generate a pipeline workflow from the Pegasus tutorial

  $ pegasus-init split

  - *Answer questions, pegasus will create an example directory and populate it with everything you need to run a sample workflow of the type you choose*
  - *View README and all other files (view catalog files after submit)*

- Generate and plan the dax, then view submit/ output

  $ ./generate_dax.sh split.dax

  $ cat split.dax

  $ ./plan_dax.sh split.dax

  $ ls  submit<tab><tab><tab><tab>

  Browser: https://hpc-pegasus.usc.edu/u/erinshaw

```
[me@hpc-pegasus]$ cd pegasus
[me@hpc-pegasus]$ pegasus-init split
Do you want to generate a tutorial workflow?
(y/n) [n]: y
1: Local Machine
2: USC HPCC Cluster
3: OSG from ISI submit node
4: XSEDE, with Bosco
5: Bluewaters, with Glite
What environment is tutorial to be setup for?
(1-5) [1]: 2
1: Process
2: Pipeline
3: Split
4: Merge
5: EPA (requires R)
6: Diamond
What tutorial workflow do you want? (1-6) [1]:
3
[Pegasus Tutorial setup for example workflow
- split for execution on usc-hpcc in directory /
auto/rcf-proj/ess/erinshaw/workshop/
pegasus/split
```

```
[me@hpc-pegasus]$ cd split

[me@hpc-pegasus]$ ls
daxgen.py
input/
pegasus.properties
rc.txt
sites.xml
generate_dax.sh*
output/
plan_dax.sh*
README.md
tc.txt

[me@hpc-pegasus]$ cat README.md
[me@hpc-pegasus]$ ls input
[me@hpc-pegasus]$ ls output
[me@hpc-pegasus]$ cat pegasus.properties
[me@hpc-pegasus]$ cat sites.xml
[me@hpc-pegasus]$ cat rc.txt
[me@hpc-pegasus]$ cat tc.txt
[me@hpc-pegasus]$ cat daxgen.py
```

# Split workflow

- Let's generate a pipeline workflow from the Pegasus tutorial

  $ pegasus-init split

  - *Answer questions, pegasus will create an example directory and populate it with everything you need to run a sample workflow of the type you choose*
  - *View README and all other files (view catalog files after submit)*

- Generate and plan the dax, then view submit/ output

  $ ./generate_dax.sh split.dax

  $ cat split.dax

  $ ./plan_dax.sh split.dax

  $ ls submit<tab><tab><tab><tab>

  Browser: https://hpc-pegasus.usc.edu/u/erinshaw

# Examples - Pegasus WMS

- Outline
  1. Set up for examples
     - *Work on Pegasus head node (hpc-pegasus.usc.edu)*
  2. Generate an example workflow
     - *Refer to tutorial for steps (https://pegasus.isi.edu/tutorial/usc/tutorial.php)*
  3. Modify the example pipeline to create a real-world workflow
     - *Show how this pipeline can be used to submit jobs in parallel*
  4. Generate an example merge workflow
     - *Show how this example can be modified to extend the real-world workflow*

# Real-world pipeline workflow

- One of the goals of this workshop is to enable HPC researchers to create their own workflows for their current projects.
- Example: If I wanted to create a Pegasus workflow for my brain analysis pipeline, how would I do it?
- Let's start with a simple example…

# ASL fMRI example

- The Addiction and Self-Control Lab studies mechanisms underlying human self-control from the combined perspectives of behavioral economics and cognitive neuroscience (sometimes collectively referred to as “neuroeconomics”).
- They use functional Magnetic Resonance Imaging (fMRI) in some of their work.

# FSL fMRI analysis tools

## FMRIB Software Library v5.0

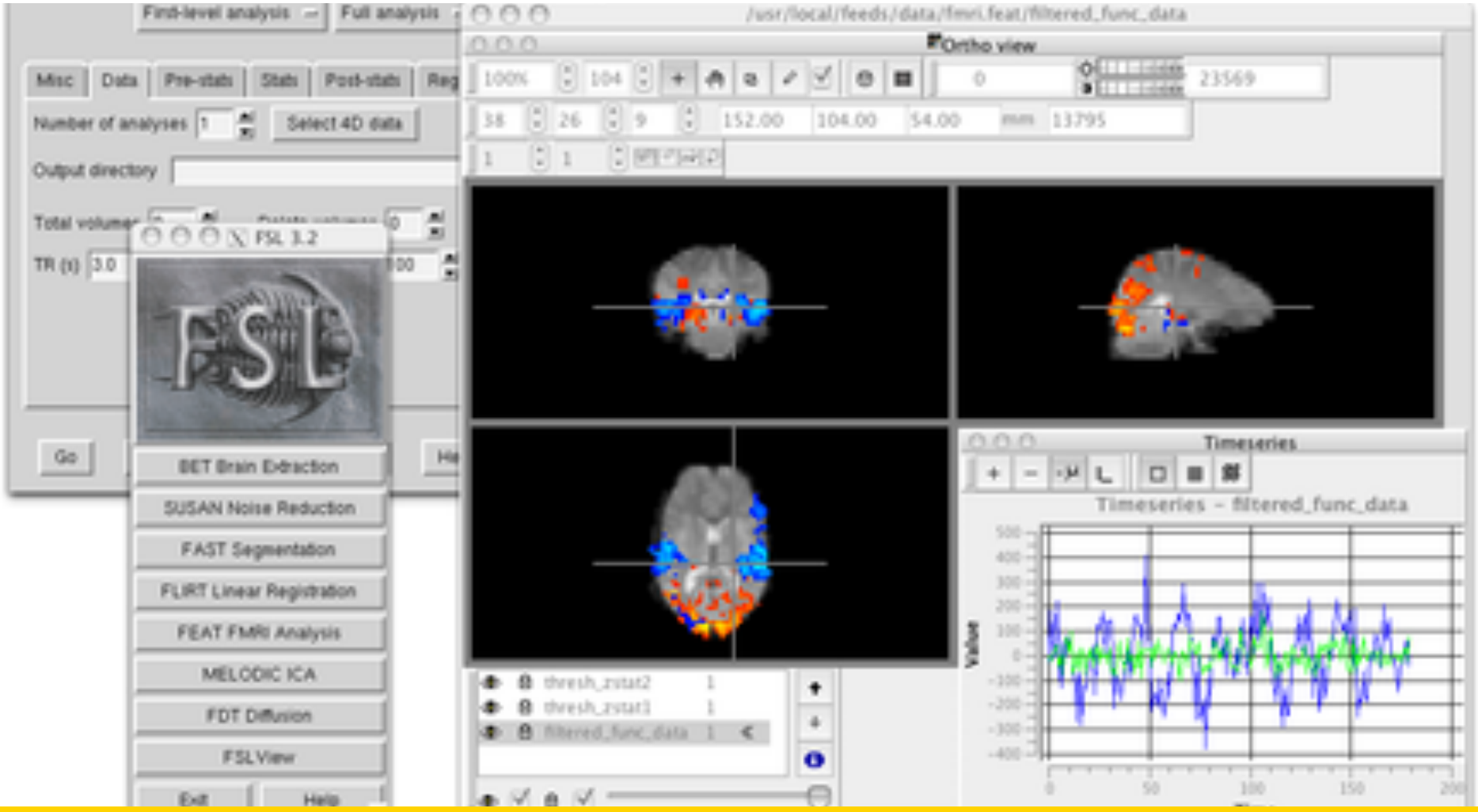
Created by the Analysis Group, FMRIB, Oxford, UK.

FSL is a comprehensive library of analysis tools for FMRI, MRI and DTI brain imaging data. It runs on Apple and PCs (both Linux, and Windows via a Virtual Machine), and is very easy to install. Most of the tools can be run both from the command line and as GUIs ("point-and-click" graphical user interfaces). To quote the relevant references for FSL tools you should look in the individual tools' manual pages, and also please reference one or more of the FSL overview papers:

1. *M.W. Woolrich, S. Jbabdi, B. Patenaude, M. Chappell, S. Makni, T. Behrens, C. Beckmann, M. Jenkinson, S.M. Smith. Bayesian analysis of neuroimaging data in FSL. NeuroImage, 45:S173-86, 2009*

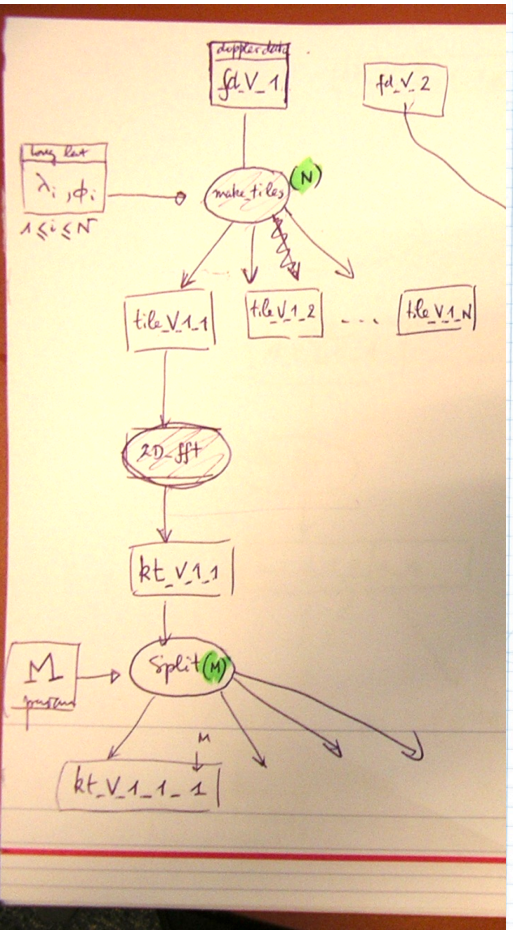
2. *S.M. Smith, M. Jenkinson, M.W. Woolrich, C.F. Beckmann, T.E.J. Behrens, H. Johansen-Berg, P.R. Bannister, M. De Luca, I. Drobnjak, D.E. Flitney, R. Niazy, J.*

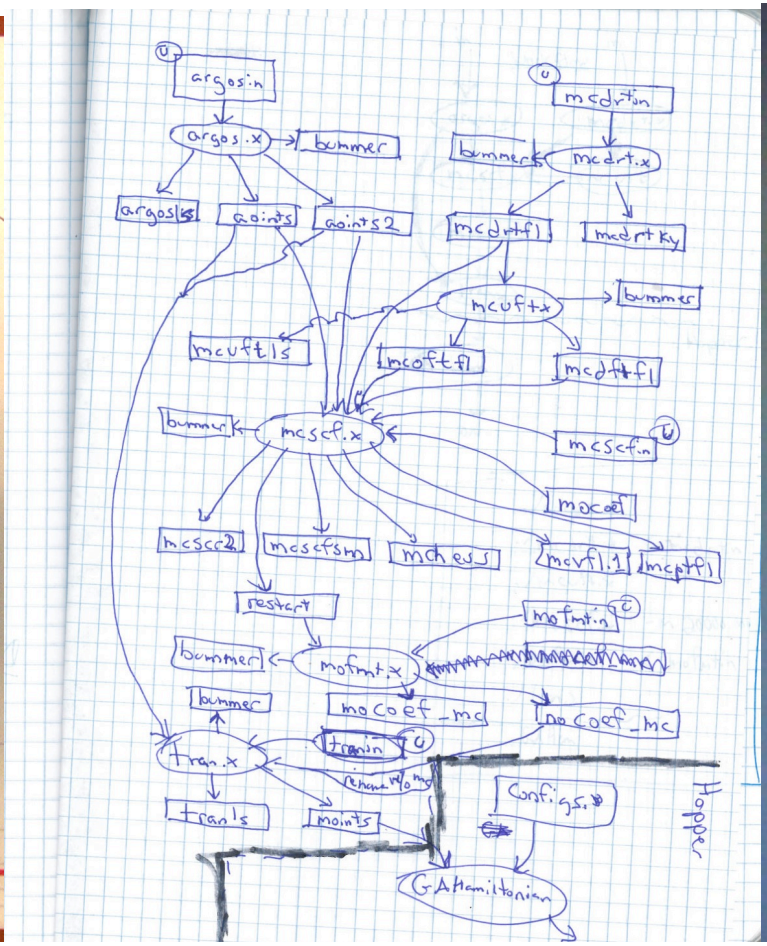
# FSL's application gui

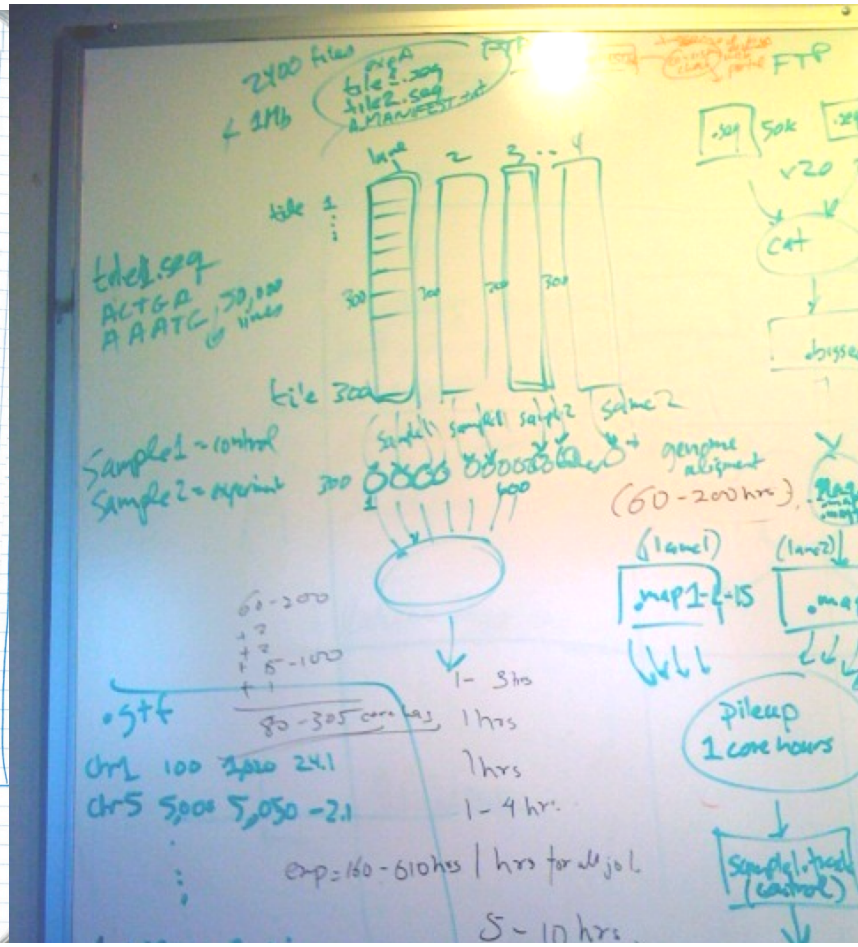


# FSL workflow

- First step is to select an example workflow to use as a template
- To choose a type, whiteboard your pipeline, etc.

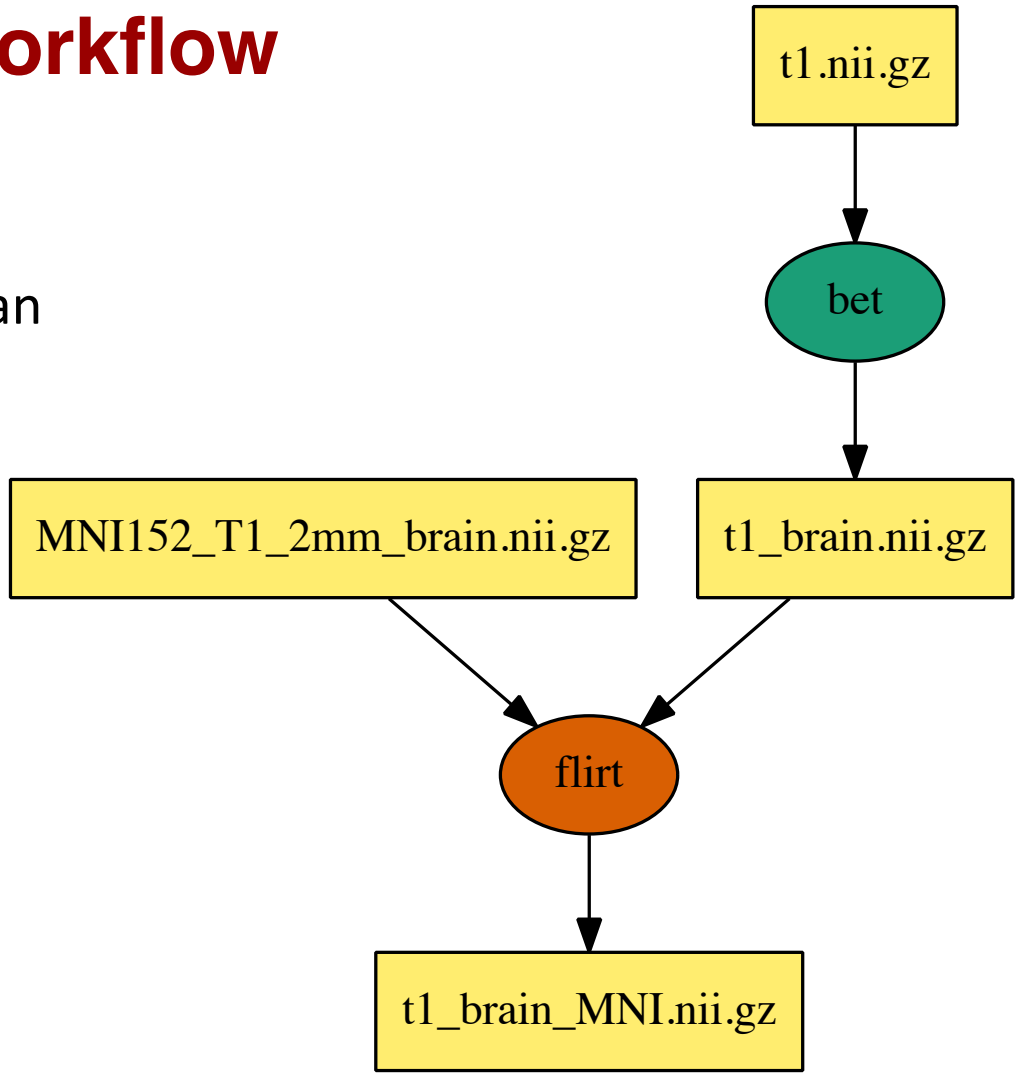


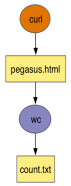



# FSL workflow

- Example, for fMRI processing
  - Start with a subject's fMRI scan
  - Apply FSL functions
  - Analyze across populations




- What type of workflow is it?




pipeline

$ pegasus-init pipeline

# Pegasus steps

- Generate template
  - $ pegasus-init fsl-pipeline
  - modify all appropriate files (daxgen.py, rc.txt, tc.txt)
- Generate dax
  - $ ./generate_dax.sh fsl.dax (runs python daxgen.py)
- Submit job
  - $ ./plan_dax.sh fsl.dax
- Monitor
  - $ pegasus-status –l /path/to/job (provided by pegasus)

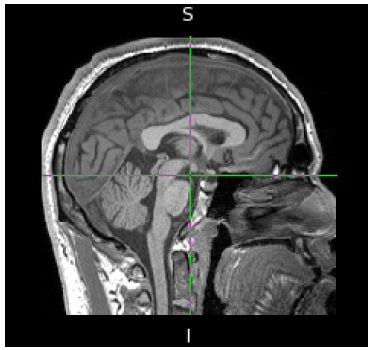
# Create pipeline workflow template

✓ = files user must modify

- fsl-pipeline/ files
  - ✓ daxgen.py *(main processing code, python version)*
  - generate_dax.sh*
  - plan_dax.sh*
  - ✓ rc.txt *(physical locations of data resources)*
  - ✓ tc.txt *(information on run-time processes)*
  - ✓ sites.xml *(configuration of site environment)*
  - pegasus.properties
  - README.md

# Step 1 (bet)

**Subject Image**



**t1.nii.gz**

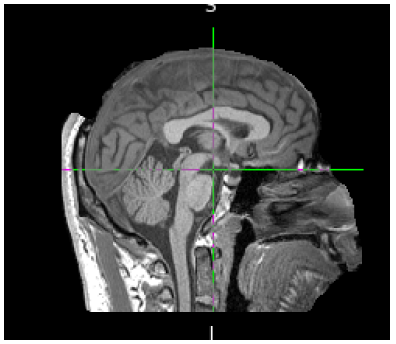
↓

logical name is
“t1_image”

bet
brain extraction tool

**bet <input> <output>  -m -n -f 0.3 –R**

**Subject NoSkull**



**t1_brain.nii.gz**

↓

logical name is
“t1_brain_image”

# Step 1 (bet)

- We need to tell Pegasus:
  - bet is a job
  - bet uses t1.nii.gz as an input file
  - bet produces t1_brain.nii.gz as an output file
  - bet is run with arguments like this:
    - bet t1.nii.gz t1_brain_nii.gz -m -n -f 0.3 –R

# Modify daxgen.py (add bet)

**pipeline**

```
#Input file
webpage = File("pegasus.html")
```

**fsl-pipeline**

```
#Input files
#subject image
t1_image = File("t1.nii.gz”)
#bet output
t1_brain_image = File("t1_brain.nii.gz”)
```

# Modify daxgen.py (add bet)

**pipeline**

```
#Executable
curl = Job("curl")
curl.addArguments("-o", webpage,
"http://pegasus.isi.edu")
curl.uses(webpage,
link=Link.OUTPUT)


#Add job to dax
dax.addJob(curl)
```

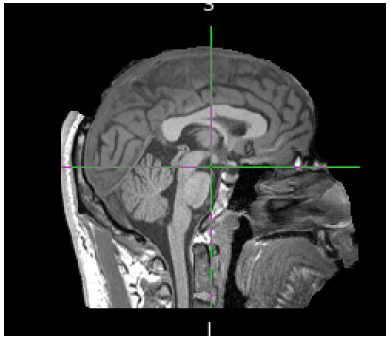
**fsl-pipeline**

```
#Brain extraction tool
bet = Job("bet")
bet.addArguments(t1_image,t1_brain_imag
e,"-m","-n","-f 0.3","-R")
bet.uses(t1_image, link=Link.INPUT)
bet.uses(t1_brain_image,
link=Link.OUTPUT)


#Add job to dax
dax.addJob(bet)
```

# Step 2 (flirt)

**Subject NoSkull**

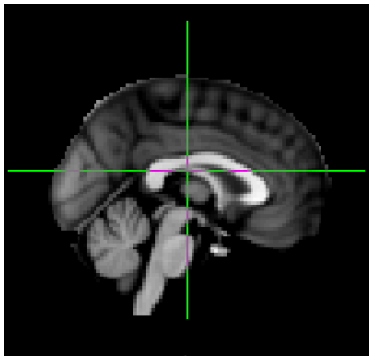


t1_brain.nii.gz

flirt
fsl linear image
registration tool

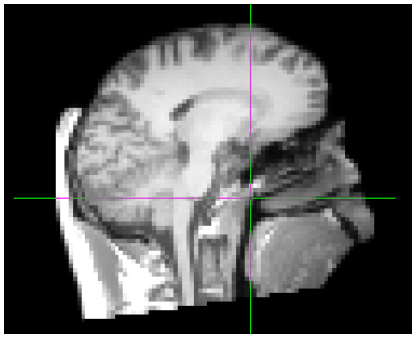
flirt -in <input> -out <output>

**Registration Image**



MNI152_T1_2mm_brain.nii.gz

**Subject Registered**



t1_brain_mni.nii.gz

logical name is
"t1_brain_mni_image"

# Step 2 (flirt)

- We need to tell Pegasus:
  - flirt is a job
  - flirt uses t1_brain_nii.gz as an input file
  - flirt uses MNI152_T1_2mm_brain.nii.gz as an input file
  - flirt produces t1_brain_mni.nii.gz as an output file
  - invoke flirt using this command:
    - flirt -in t1_brain_nii.gz -ref MNI152_T1_2mm_brain.nii.gz -out t1_brain_mni.nii.gz

# Modify daxgen.py (add flirt)

**pipeline**

```
#Input file
count = File("count.txt")
```

**fsl-pipeline**

```
#Input files
#reference image
MNI_image =
File("MNI152_T1_2mm_brain.nii.gz”)
#flirt output
t1_brain_mni_image =
File("t1_brain_MNI.nii.gz”)
```

# Modify daxgen.py (add flirt)

**pipeline**

#Executable

- wc = Job("wc")
- wc.addArguments("-l",webpage)
- wc.setStdout(count)
- wc.uses(webpage, link=Link.INPUT)
- wc.uses(count, link=Link.OUTPUT, transfer=True, register=False)

#Add job to dax

dax.addJob(wc)

**fsl-pipeline**

#Linear registration tool

flirt = Job("flirt")

flirt.addArguments("-in",t1_brain_image,"-ref",MNI_image,"-out",t1_brain_mni_image)

flirt.uses(t1_brain_image, link=Link.INPUT)

flirt.uses(MNI_image, link=Link.INPUT)

flirt.uses(t1_brain_mni_image, link=Link.OUTPUT, transfer=True, register=True)

#Add job to dax

dax.addJob(flirt)

# Modify daxgen.py (add dependencies)

**pipeline**

```
#Add dependencies
dax.depends(wc, curl)
```

**fsl-pipeline**

```
#Add dependencies
dax.depends(flirt, bet)
```

# Final fsl-pipeline daxgen.py

daxgen.py 1/2

```
# Create dax object with name
dax = ADAG("fsl-pipeline")

# Add some workflow-level metadata
USER = pwd.getpwuid(os.getuid())[0]
dax.metadata("creator", "%s@%s" % (USER, os.uname()[1]))
dax.metadata("created", time.ctime())
dax.metadata("researcher", "amelrose")

#Brain extraction tool
t1_image = File("t1.nii.gz")
t1_brain_image = File("t1_brain.nii.gz")

bet = Job("bet”)
bet.addArguments(t1_image,t1_brain_image,"-m","-n","-f 0.3","-R")
bet.uses(t1_image, link=Link.INPUT)
bet.uses(t1_brain_image, link=Link.OUTPUT)

dax.addJob(bet)
```

# Final fsl-pipeline daxgen.py

daxgen.py 2/2

```
#Linear registration tool
MNI_image = File("MNI152_T1_2mm_brain.nii.gz")
t1_brain_mni_image = File("t1_brain_MNI.nii.gz")

flirt = Job("flirt")
flirt.addArguments("-in",t1_brain_image,"-ref",MNI_image,"-out",t1_brain_mni_image)
flirt.uses(t1_brain_image, link=Link.INPUT)
flirt.uses(MNI_image, link=Link.INPUT)
flirt.uses(t1_brain_mni_image, link=Link.OUTPUT, transfer=True, register=True)
dax.addJob(flirt)

dax.depends(flirt, bet)

f = open(daxfile, "w")
dax.writeXML(f)
f.close()
```

A working daxgen.py file can be found at
/home/rcf-proj/workshop/pegasus/fsl-pipeline.tar.gz

# Modify replica catalog (rc.txt)

**pipeline**

```
# The format is:
# LFN     PFN    pool="SITE"
#
# For example:
#data.txt  file:///tmp/
data.txt  site="local"
```

**fsl-pipeline**

```
# The format is (on one line, with space between columns)
# LFN     PFN    pool="SITE"
t1.nii.gz   file:///home/rcf-proj/workshop/input/t1.nii.gz
site="usc-hpcc"

MNI152_T1_2mm_brain.nii.gz   file:///home/rcf-proj/
workshop/input/MNI152_T1_2mm_brain.nii.gz site="usc-
hpcc"
```

# Modify transformation catalog (tc.txt)

**pipeline**

```
tr curl {
    site usc-hpcc {
        pfn "/usr/bin/curl"
        arch "x86_64"
        os "LINUX"
        type "INSTALLED"
    }
}
tr wc {
    site usc-hpcc {
        pfn "/usr/bin/wc"
        arch "x86_64"
        os "LINUX"
        type "INSTALLED"
    }
}
```

**fsl-pipeline**

```
tr bet {
    site usc-hpcc {
        pfn "/home/rcf-proj/workshop/pegasus/bin/bet"
        arch "x86_64"
        os "LINUX"
        type "INSTALLED”
    }
}
tr flirt {
    site usc-hpcc {
        pfn "/home/rcf-proj/workshop/pegasus/bin/flirt"
        arch "x86_64"
        os "LINUX"
        type "INSTALLED"
    }
}
```

# Future modification to transf. file (tc.txt)

**fsl-pipeline**

```
tr bet {
   site usc-hpcc {
      pfn ”/home/rcf-proj/workshop/pegasus/bin/bet"
      arch "x86_64"
      os "LINUX"
      type "INSTALLED”
      profile pegasus "walltime” “3600”
      profile pegasus “nodes" “1”
      profile pegasus “ppn" “8”
      profile pegasus “memory" “5gb”
   }
}
```

**fsl-pipeline**

```
tr flirt {
   site usc-hpcc {
      pfn ”/home/rcf-proj/workshop/pegasus/bin/flirt”
      arch "x86_64"
      os "LINUX"
      type "INSTALLED”
      <!--https://pegasus.isi.edu/documentation/glite.php#glite_mappings
      profile pegasus "walltime” “7200”
      profile pegasus “nodes" “2”
      profile pegasus “ppn" “8”
      profile pegasus “memory" “5gb”
      profile pegasus “glite.arguments” “-l gpus 2”
      -->
   }
}
```

# Modify sites catalog (sites.xml)

```
<site handle="usc-hpcc" arch="x86_64" os="LINUX">
                        :
                        :
  <profile namespace="pegasus" key="style">glite</profile>
  # Set environment variables and qsub arguments here
  <profile namespace="env" key="FSLDIR">/home/rcf-proj/workshop/pegasus/bin</profile>
  <profile namespace="env" key="FSLOUTPUTTYPE">NIFTI_GZ</profile>


  # Set environment variables and qsub arguments here
  # (See https://pegasus.isi.edu/documentation/glite.php#glite_mappings)
</site>
```

# Pegasus steps

- Generate template
  - $ pegasus-init fsl-pipeline
- Generate dax
  - $ ./generate-dax.sh fsl.dax (runs python daxgen.py)
- Submit job
  - $ ./plan-dax.sh fsl.dax
  - $ ls submit/<tab><tab>…
- Monitor
  - $ pegasus-status –l /path/to/job (provided by pegasus)
  - $ myqueue
  - Browser: https://hpc-pegasus.usc.edu/u/erinshaw

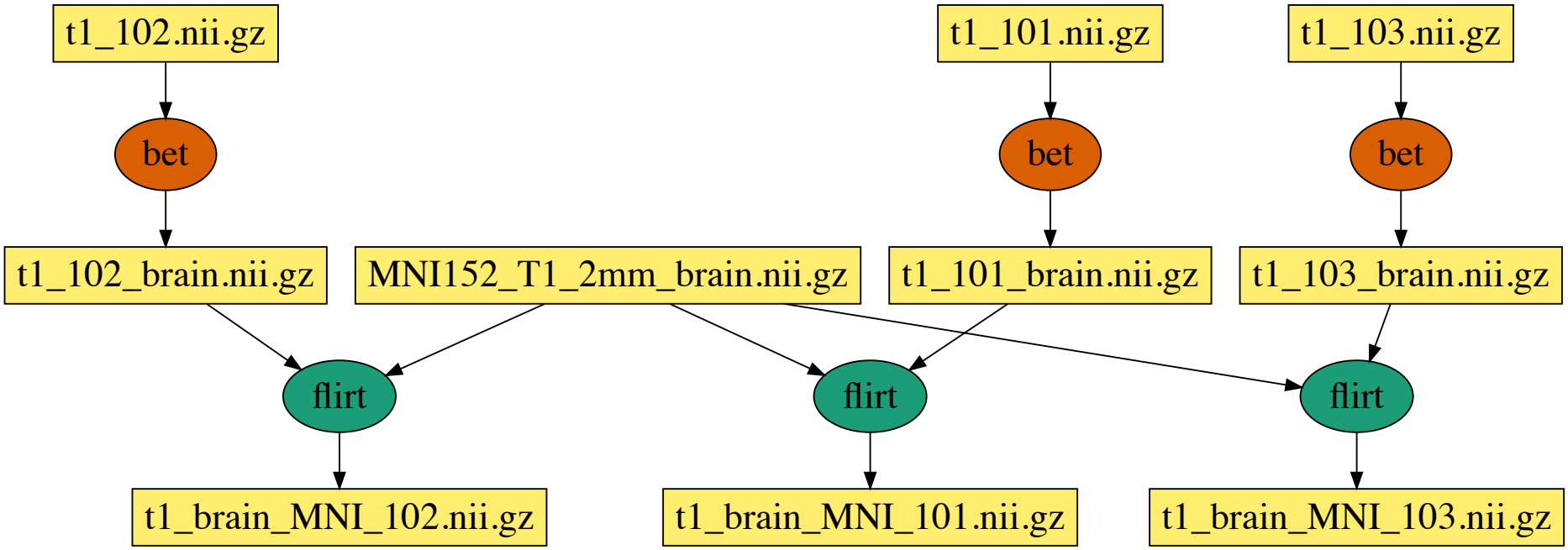
# Check newly created submit/ directory

```
[erin@hpc-pegasus]$ ls submit/erinshaw/pegasus/
fslpipeline/ pipeline/
[erin@hpc-pegasus]$ ls submit/erinshaw/pegasus/pipeline/run000
run0001/ run0002/
[erin@hpc-pegasus]$ ls submit/erinshaw/pegasus/pipeline/run0001/
00/          monitord-notifications.log          pipeline-0.dag.dagman.out   pipeline-0.log
braindump.txt  monitord.started                  pipeline-0.dag.lib.err      pipeline-0.metadata
catalogs/      monitord.subwf                    pipeline-0.dag.lib.out      pipeline-0.metrics
fsl.dax        pegasus.3356217682185801485.properties  pipeline-0.dag.metrics      pipeline-0.notify
jobstate.log   pipeline-0.cache                  pipeline-0.dag.metrics.out  pipeline-0.stampede.db
monitord.done  pipeline-0.dag                    pipeline-0.dag.nodes.log    pipeline-0.static.bp
monitord.info  pipeline-0.dag.condor.sub         pipeline-0.dot
monitord.log   pipeline-0.dag.dagman.log         pipeline-0.exitcode.log
```

# Submit jobs in parallel

- Each original subject image can be processed independently
  - Considered “pleasantly (or embarrassingly) parallel”
  - Each job in the dax will be submitted as a separate PBS job
  - Pegasus will handle all submissions to queue
    - *Note that it will request a node for each job, by default*
    - *If you want to group jobs together to save on PBS job submission overhead you can set this up, beyond scope of this workshop*
- We will pre-select a set of subject fMRI files
  - You can acquire input data by walking a directory as well

# Submit jobs in parallel



# Pipeline-Parallel Workflow

daxgen.py 1/3

```
# Create dax object with name
dax = ADAG("fsl-pipeline-plus")

# Add some workflow-level metadata
USER = pwd.getpwuid(os.getuid())[0]
dax.metadata("creator", "%s@%s" % (USER, os.uname()[1]))
dax.metadata("created", time.ctime())
dax.metadata("researcher", "amelrose")
```

# Pipeline-Parallel Workflow

daxgen.py 2/3

```
subjects = [101,102,103]
for s in subjects:

    #Associate files with subject numbers
    t1_image = File("t1_%s.nii.gz" %s)
    t1_brain_image = File("t1_%s_brain.nii.gz" %s)

    #Brain extraction tool
    bet = Job("bet")
    bet.addArguments(t1_image,t1_brain_image,"-m","-n","-f 0.3","-R")
    bet.uses(t1_image, link=Link.INPUT)
    bet.uses(t1_brain_image, link=Link.OUTPUT)
    dax.addJob(bet)
```

# Pipeline-Parallel Workflow

daxgen.py 3/3

```
#Linear registration tool
MNI_image = File("MNI152_T1_2mm_brain.nii.gz")
t1_brain_mni_image = File("t1_brain_MNI_%s.nii.gz" % s)

flirt = Job("flirt")
flirt.addArguments("-in",t1_brain_image,"-ref",MNI_image,
      "-out",t1_brain_mni_image)
flirt.uses(t1_brain_image, link=Link.INPUT)
flirt.uses(MNI_image, link=Link.INPUT)
flirt.uses(t1_brain_mni_image, link=Link.OUTPUT, transfer=True, register=True)
dax.addJob(flirt)

dax.depends(flirt, bet)

# End of for loop
f = open(daxfile, "w")
dax.writeXML(f)
f.close()
```

# Modify the replica catalog (rc.txt)

```
# The format is:
# LFN    PFN    pool="SITE”
t1_101.nii.gz file://(your_path_here)/input/t1_101.nii.gz site="usc-hpcc"

t1_102.nii.gz file://(your_path_here)/input/t1_102.nii.gz site="usc-hpcc"

t1_103.nii.gz file://(your_path_here)/input/t1_103.nii.gz site="usc-hpcc"

MNI152_T1_2mm_brain.nii.gz file://(your_path_here)/input/MNI152_T1_2mm_brain.nii.gz site="usc-
hpcc"
```

# Pipeline-Parallel Workflow

- A copy of this completed daxgen.py is located in:
  - /home/rcf-proj/workshop/pegasus/fsl-pipeline-plus.tar.gz
- ./generate_dax.py pipeline-plus.dax
- ./plan pipeline-plus.dax
- We should see the t1_brain_MNI_XXX.nii.gz files in output directory when complete
- It's possible to group the bet and flirt jobs into one PBS job
  - Saves time due to decreased queue time overhead

# Examples - Pegasus WMS

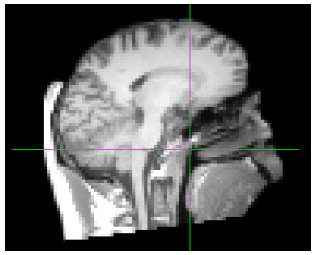
- Outline
  1. Set up for examples
     - *Work on Pegasus head node (hpc-pegasus.usc.edu)*
  2. Generate an example pipeline workflow
     - *Refer to tutorial for steps (https://pegasus.isi.edu/tutorial/usc/tutorial.php)*
  3. Modify the example pipeline to create a real-world workflow
     - *Show how this pipeline can be used to submit jobs in parallel*
  4. Generate an example merge workflow
     - *Show how this example can be modified to extend the real-world workflow*

# Merge workflow

- Most parallel workflows have a "merge" stage where a summary function is applied
- We will use fslmerge to combine the results of each flirt job

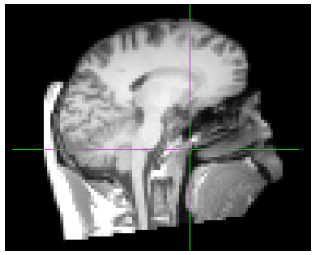
# Merge Workflow

**Subject 101**



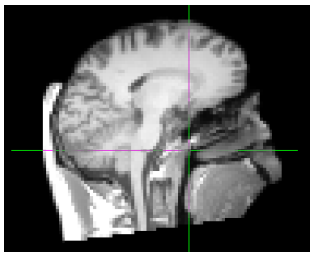
t1_brain_MNI_101.nii.gz

**Subject 102**



t1_brain_MNI_102.nii.gz

**Subject 103**

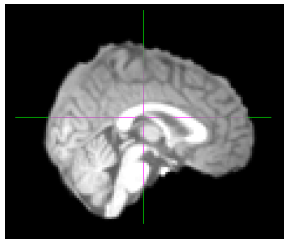


t1_brain_MNI_103.nii.gz

fslmerge
merge registered subject scans

fslmerge −t <output> <file1 file2 file3>

**Brain Blob**



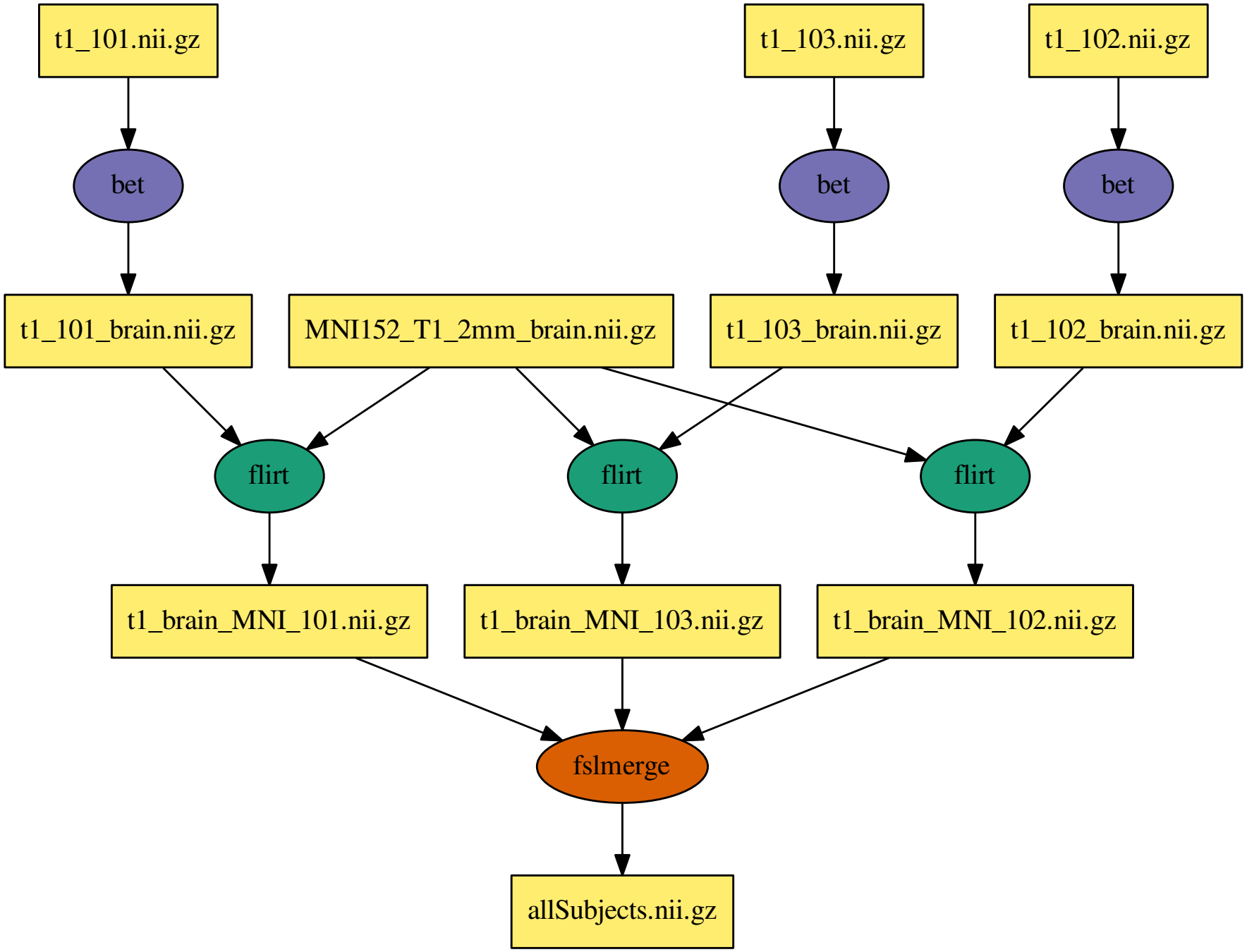
allSubjects.nii.gz

# Merge workflow

- Syntax for fslmerge is:

```
fslmerge -t <output> <file1 file2....>
```

- We need to tell Pegasus:
  - fslmerge is a job
  - fslmerge depends on each flirt job
  - fslmerge uses the output of flirt as input
  - fslmerge uses "allSubjects.nii.gz" as an outputfile (and that we want to keep it)
  - fslmerge needs the arguments "-t allsubjects.nii.gz file1 file2 file3"
    - file1 -> t1_brain_MNI_101.nii.gz
    - file2 -> t1_brain_MNI_102.nii.gz
    - file3 -> t1_brain_MNI_103.nii.gz


t1_101.nii.gz
t1_103.nii.gz
t1_102.nii.gz
bet
bet
bet
t1_101_brain.nii.gz
MNI152_T1_2mm_brain.nii.gz
t1_103_brain.nii.gz
t1_102_brain.nii.gz
flirt
flirt
flirt
t1_brain_MNI_101.nii.gz
t1_brain_MNI_103.nii.gz
t1_brain_MNI_102.nii.gz
fslmerge
allSubjects.nii.gz

# Merge workflow

- In daxgen.py we'll have to add:

```
merge =  Job("fslmerge")
allSubjects=File("allSubjects.nii.gz")
merge.uses(allSubjects,link=Link.OUTPUT,transfer=True, register=True)
merge.addArguments("-t", allSubjects)

#Here comes trouble
dax.depends(merge,flirt)
merge.uses(t1_brain_mni_image, link=Link.INPUT) # output from flirt
merge.addArguments(t1_brain_mni_image)
```

# Merge Workflow

- What is the trouble?
- There are multiple "flirt" jobs
- There are multiple "t1_brain_mni_image" files
- We can't reference these job and file objects outside of the subjects for loop
- In 'subjects' for loop we should have:

```
dax.depends(merge,flirt)
merge.uses(ti_brain_mni_image, link=Link.INPUT)
merge.addArguments(ti_brain_mni_image)
```

# Merge Workflow

- A copy of this completed daxgen.py is located in:
  - /home/rcf-proj/workshop/pegasus/fsl-merge.tar.gz
- ./generate_dax.py fsl-merge.dax
- ./plan fsl-merge.dax
- We should see allSubjects.nii.gz in the output directory when complete

# End of Examples

- Pegasus dashboard
  - Read more about Pegasus' dashboard tool
    - *https://pegasus.isi.edu/tutorial/usc/tutorial_wf_dashboard.php*
  - To access, make the workflow readable, then open web page
    - *$ chmod +r ~/.pegasus/workflow.db*
    - *Browser: https://hpc-pegasus.usc.edu/u/<your_NetID>*
- End of examples, see user guide for more information https://pegasus.isi.edu/documentation/pegasus-user-guide.pdf

- Back to main tutorial…