Open Science Grid

Workflows: from Development to *Automated* Production

Thursday morning, 10:00 am

Lauren Michael <lmichael@wisc.edu>

Research Computing Facilitator

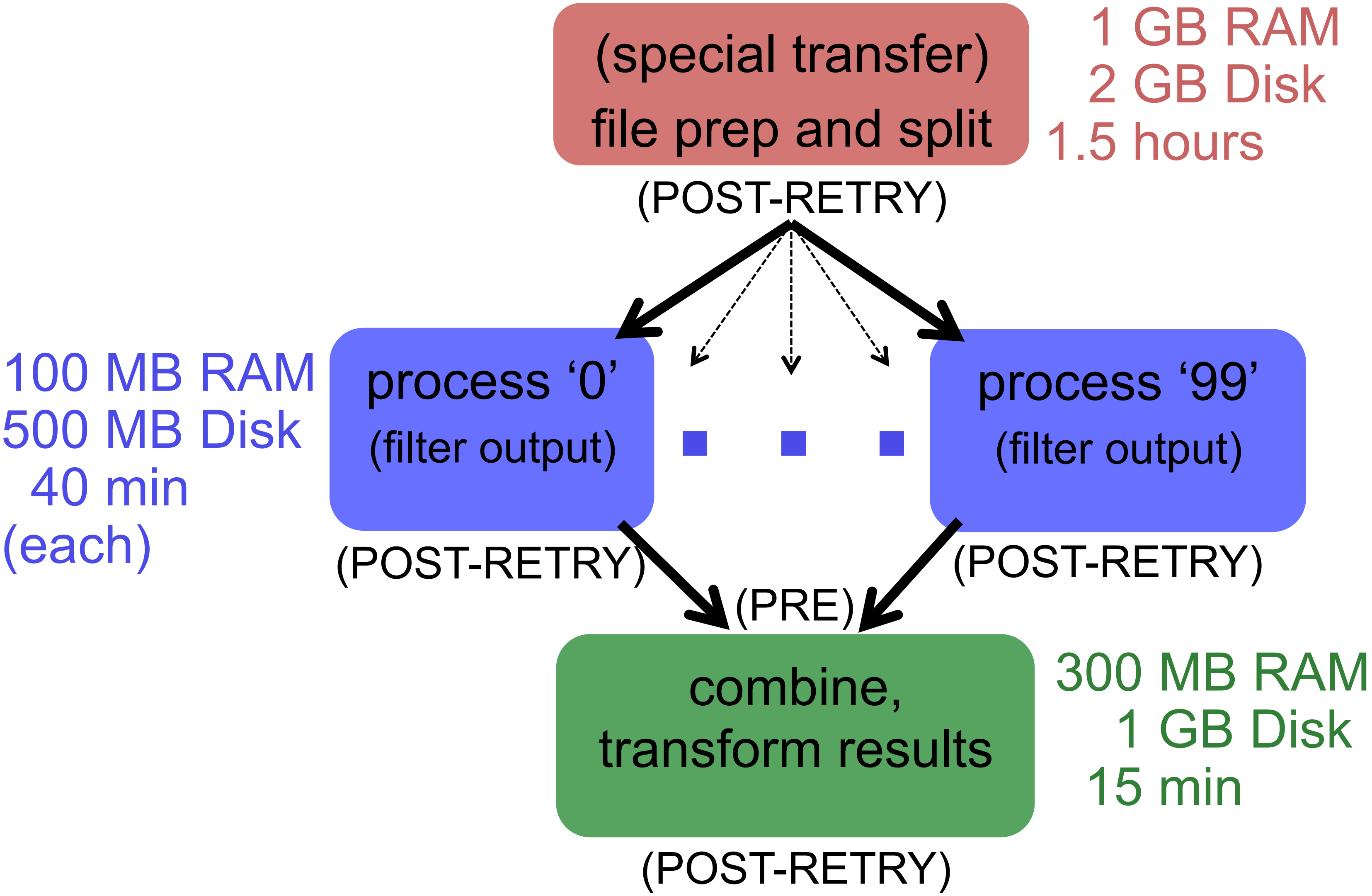
University of Wisconsin - Madison

‘Engineering’ a Good Workflow

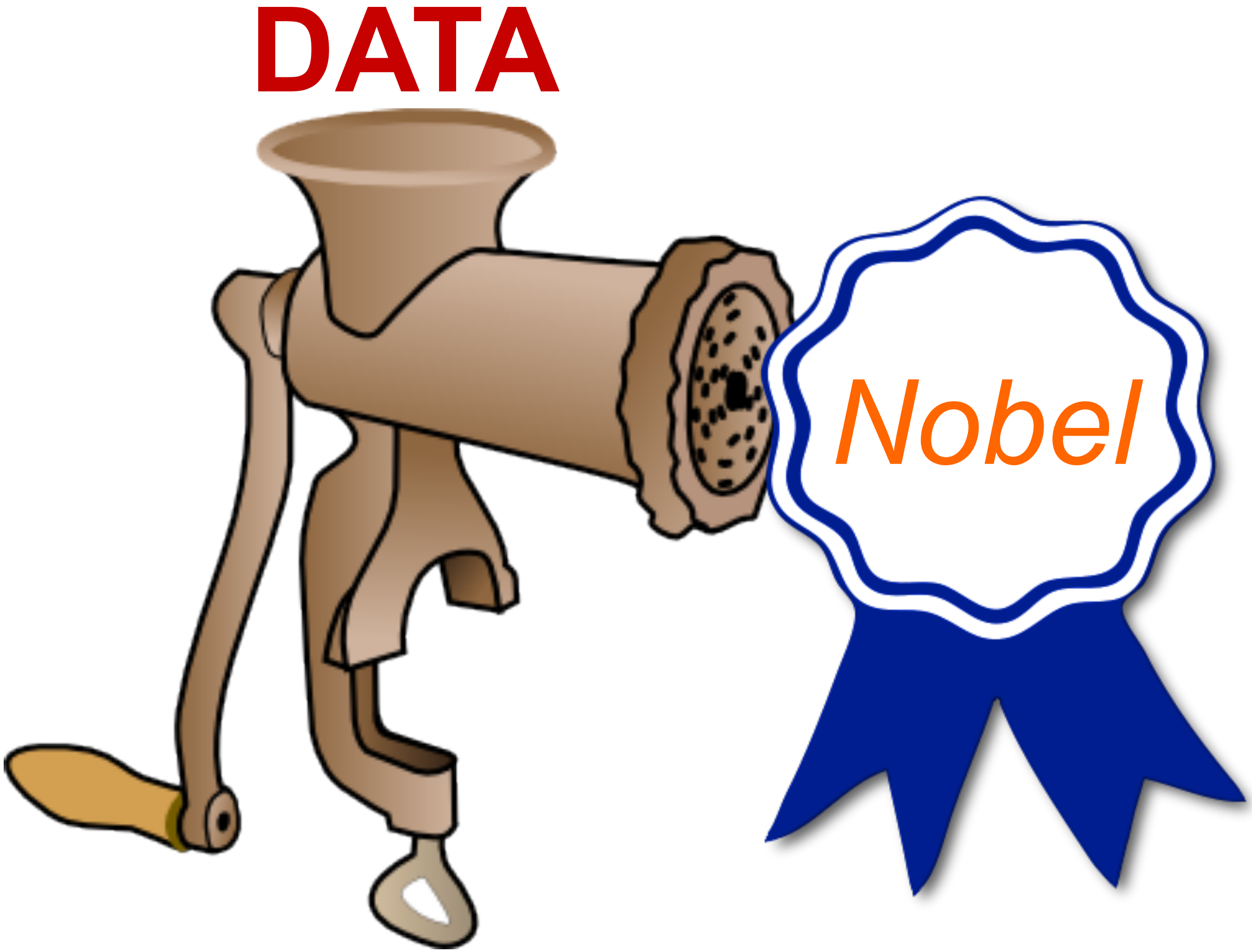
1. Draw out the *general* workflow
2. Define details (test ‘pieces’ with HTCondor jobs)
   - divide or consolidate ‘pieces’
   - off-load file transfers and consider file transfer times
   - identify steps to be automated or checked
3. Build it piece-by-piece; test and optimize
4. Scale-up: data and computing resources
5. What more can you automate or error-check?

(And remember to document)

From This . . .


(special transfer)
file prep and split
1 GB RAM
2 GB Disk
1.5 hours
(POST-RETRY)
100 MB RAM
500 MB Disk
40 min
(each)
process '0'
(filter output)
process '99'
(filter output)
(POST-RETRY)
(PRE)
(POST-RETRY)
combine,
transform results
300 MB RAM
1 GB Disk
15 min
(POST-RETRY)

End Up with This



Scaling and Optimization

- Your ‘small’ DAG runs (once)! Now what?
  - Need to make it run *full-scale*
  - Need to make it run *everywhere, everytime*
  - Need to make it run *unattended*
  - Need to make it run *when someone else tries*

Scaling Up – Things to Think About

- More jobs:
  - 50-MB files may be fine for 10 or 100 jobs, but not for 1000 jobs. Why?
  - Steps to identify rare errors
- Larger files:
  - more execute RAM and disk space
  - potentially more transfer and compute time
- Scale-up gradually

Data Scaling Solutions (larger files)

- File manipulations
  - split into pieces, when possible (HTC!)
  - filter files to only essential data
  - compression/decompression
- Listen to Lincoln’s methods (Yesterday):
  - Sandbox
  - Caching
  - Pre-staging
  - Storage Element (SE) horsepower

Make It Run Everywhere

- What does an OSG machine have?
  - Assume the worst: nothing
- Bring as much as possible with you:
  - Won’t that slow me down?
- Bring:
  - Executable
  - Environment
  - Parameters (using parameter files)
  - Random numbers (generate before-submission)

Scaling Out to OSG: Rules of Thumb

- CPU (single-threaded)
  - Best jobs run between **5 min** and **2 hours**
    - Upper limit somewhat soft
- Disk
  - Keep scratch working space < 20 GB
  - Intermediate needs (/tmp?)
  - Submit disk: think total and I/O transfer
- **Batch or Break Up Jobs**
- Use squid caching where appropriate

The expanding onion

- Laptop (1 machine)
  - You control everything!
- Local cluster (1000 cores)
  - You can ask an admin nicely
- Campus (5000 cores)
  - It better be important/generalizable
- OSG (50,000 cores)
  - Good luck finding the pool admins

Bringing It With You: MATLAB

- What’ s the problem with MATLAB?
  - license limitations
- What’ s the solution?
  - “compiling”
- Similar measures for other interpreter languages (R, Python, etc)

How to bring MATLAB along

1) Purchase & install MATLAB, which comes with a "compiler"
2) Run compiler as follows: (online guides)

```
$ mcc -m -R -singleCompThread -R -nodisplay -R -nojvm -nocache foo.m
```

3) This creates run_foo.sh (et. al.)
4) Create tarball of the runtime

```
    $ cd /usr/local/mathworks-R2014b
    $ tar cvzf ~/matlab.tgz ../mathworks-R2014b
```

More MATLAB

5) Edit the run_foo.sh

```
tar xzf matlab.tgz
mkdir cache
chmod 0777 cache
export MCR_CACHE_ROOT=`pwd`/cache
```

Make a submit file:

```
universe = vanilla
executable = run_foo.sh
arguments = ./mathworks-R2014b
should_transfer_files = yes
when_to_transfer_output = on_exit
transfer_input_files = matlab.tgz, foo
queue
```

Make It Work Everytime

- What could possibly go wrong?
  - Eviction
  - Non-existent dependencies
  - File corruption
  - Performance surprises
    - Network
    - Disk
    - …
  - *Maybe* even a bug in your code

Self-Checkpointing (for long jobs and shish-kebabs)

1. Changes to your code
   - Save information about progress to a new file, at least every 60 minutes
   - At the beginning of code:
     - If progress file exists, start from where the program (or script) left off
     - Otherwise, start from the beginning
2. Change to submit file:

```
when_to_transfer_output = ON_EXIT_OR_EVICT
```

Error Checks Are Essential

- If you don’t check, it will happen…

- Check expected file existence (transfer or creation), and repeat with a finite loop
  - better yet, check *rough* file size too
- Advanced:
  - Error-check with wrapper-RETRY combo

  (RETRY for *specific* error codes from wrapper)

What to do if a check fails

- Understand something about failure
- Use DAG “RETRY”, when useful
- Let the rescue dag continue…

Performance Surprises

One bad node can ruin your whole day

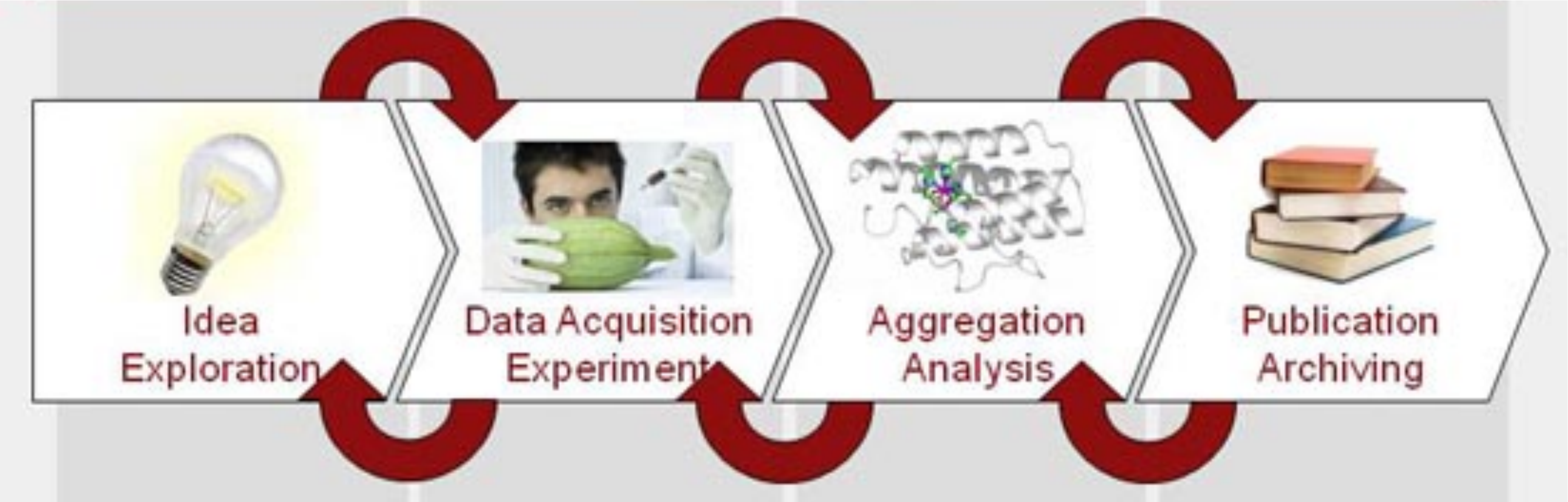
- “Black Hole” machines
  - GLIDEIN whitelist/blacklist if a site is somehow ‘bad’. (But talk to the GOC first!)
- *REALLY* slow machines
  - Use periodic_hold / periodic_release

Make It Work Unattended

- Remember the ultimate goal: Automation! Time savings!

- Need to automate:
  - Data collection?
  - Data cleansing
  - Submission (condor cron)
  - Analysis and verification
  - LaTeX and paper submission ☺

Make *Science* Work Unattended?



Well, maybe not, but a scientist can dream …

Make It Run(-able) for Someone Else

- If others can’t reproduce your work, it isn’t real science!
  - Work hard to make this happen.
  - It’s *their* throughput, too.

Only ~10% of published cancer research is reproducible!

(Yet another argument for automation)

Open Science Grid

Documentation at Multiple Levels

- In job files: comment lines
  - submit files, wrapper scripts, executables

- In README files
  - describe file purposes
  - define overall workflow, justifications

- In a Document
  - draw the workflow!

Make It Run Faster? Maybe.

- Throughput, throughput, throughput
  - Resource reductions (match more slots!)
  - Wall-time reductions
    - if significant *per workflow*
    - Why not *per job*?
- Think in orders of magnitude:
  - Say you have 1000 hour-long jobs that are matched at a rate of 1 per minute …

Waste the computer’s time, not yours.

Maybe Not Worth It

Maybe Not Worth It:

- Rewriting your code in a different language
- Targeting “faster” machines
- Targeting machines that will run longer jobs
- Others?

Waste the computer’s time, not yours.

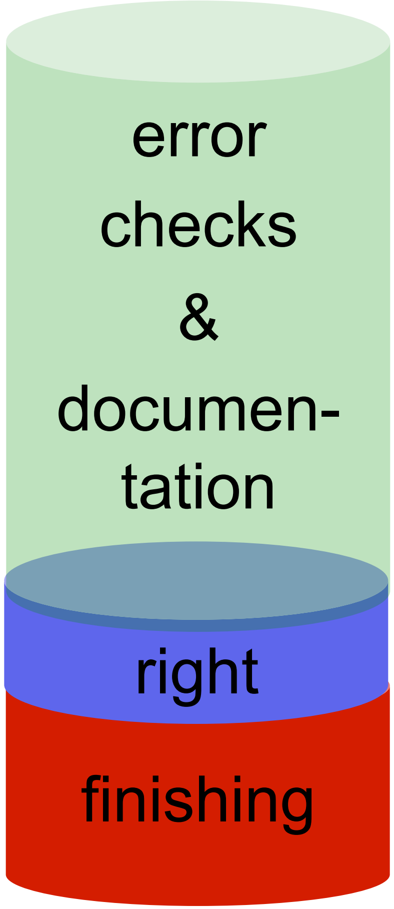
Testing, Testing, 1-2-3 …

- ALWAYS test a subset after making changes
  - How big of a change needs retesting?
- Scale up gradually
- Avoid making problems for others (and for yourself)

If this were a test…

- 20 points for finishing at all
- 10 points for the right answer
- 1 point for every error check
- 1 point per documentation line

Out of 100 points? 200 points?



Questions?

- Feel free to contact me:
  - lmichael@wisc.edu
- Now: Break
  - 10:30-10:45am
- Next:
  - **10:45am-12:15pm: Exercises 6.2, 6.3**
  - 12:15pm: Lunch
  - 1:15-2:30pm: Principles of High-Throughput Computing