Workflows: from Development to *Automated* Production

Friday

Lauren Michael <lmichael@wisc.edu>
Research Computing Facilitator
University of Wisconsin - Madison

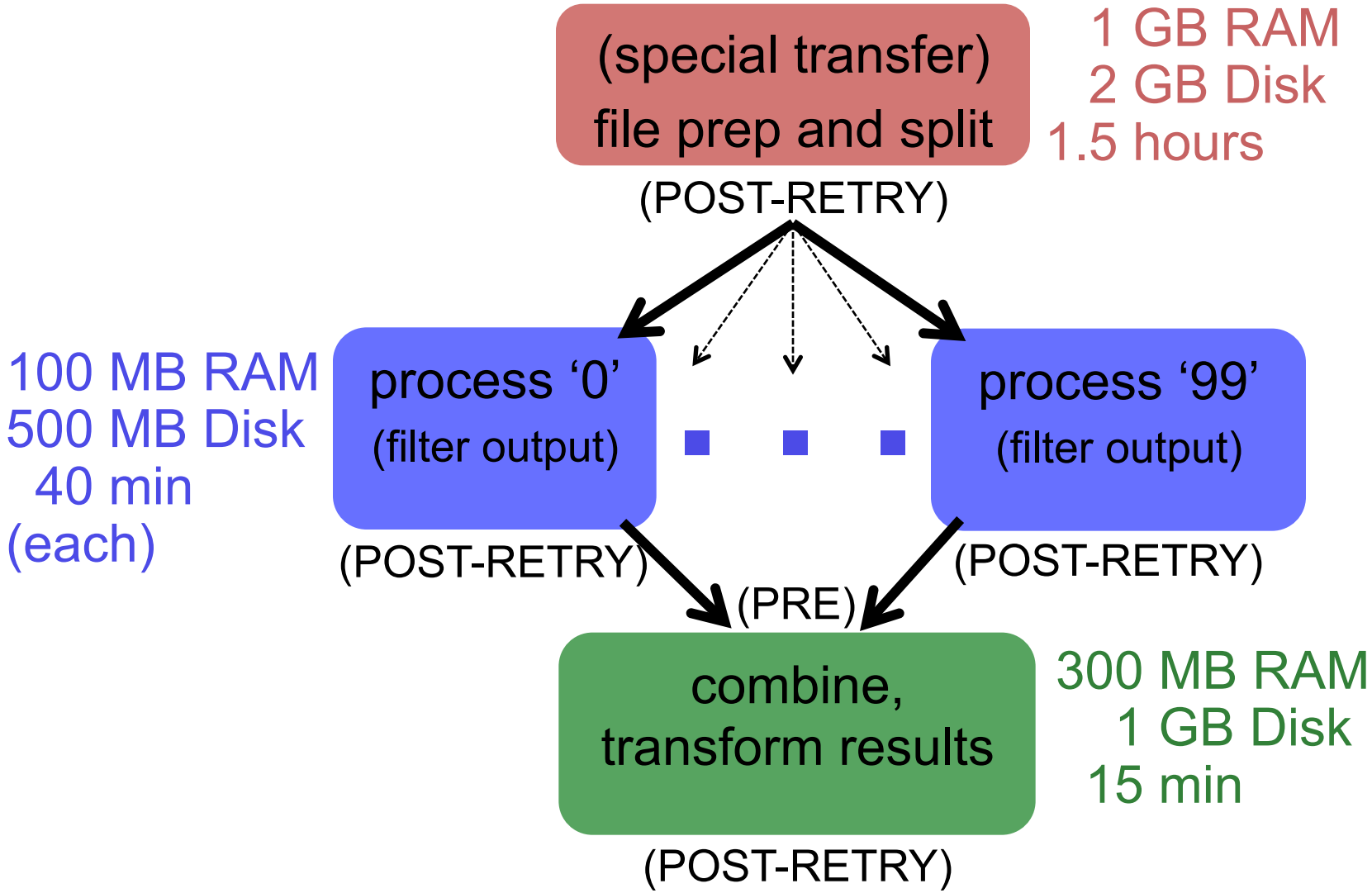
Open Science Grid

Building a Good Workflow

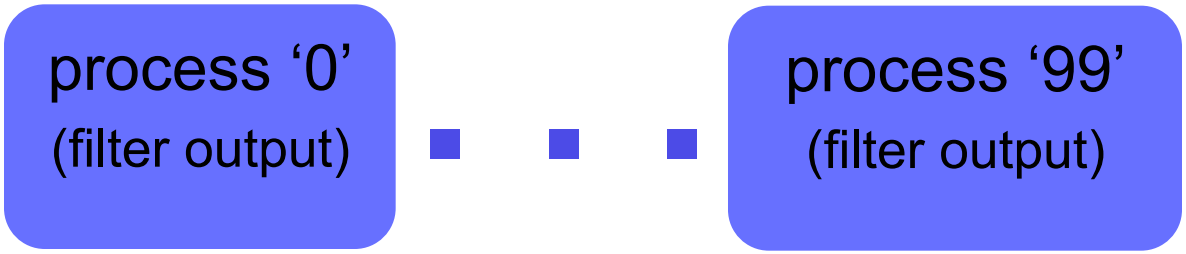
1. Draw out the *general* workflow
2. Define details (test 'pieces' with HTCondor jobs)
   - divide or consolidate 'pieces'
   - determine resource requirements
   - identify steps to be automated or checked
3. **Build it modularly; test and optimize**
4. Scale-up gradually
5. What more can you automate or error-check?

(And remember to document!)

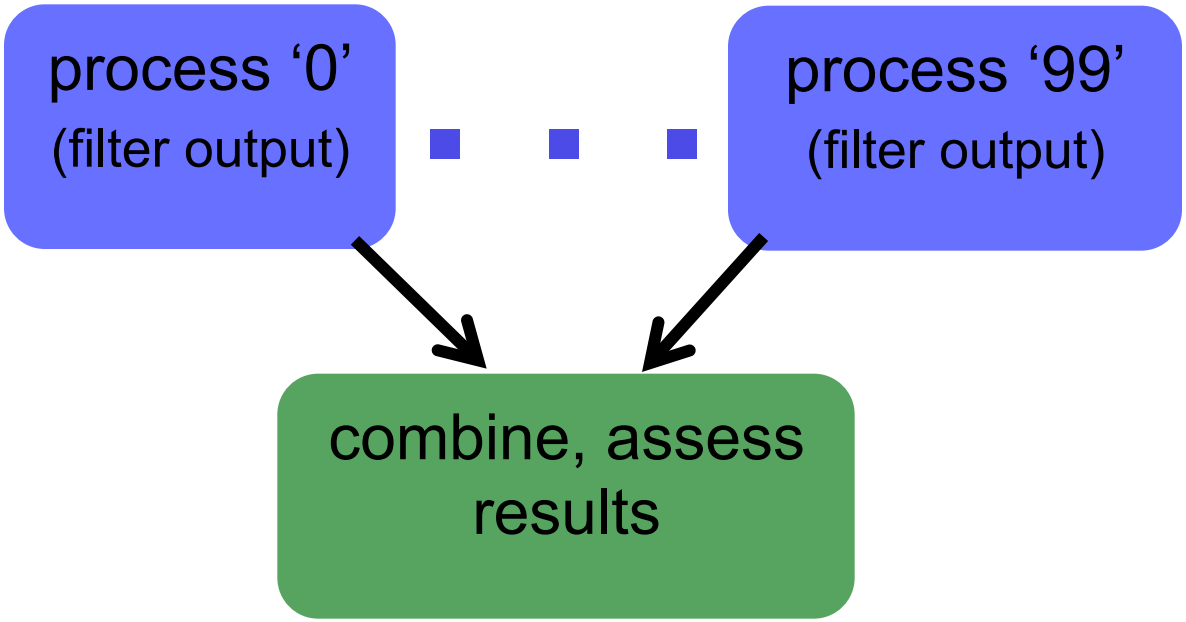
To Get Here …


(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)
(POST-RETRY)
(PRE)
combine,
transform results
300 MB RAM
1 GB Disk
15 min
(POST-RETRY)

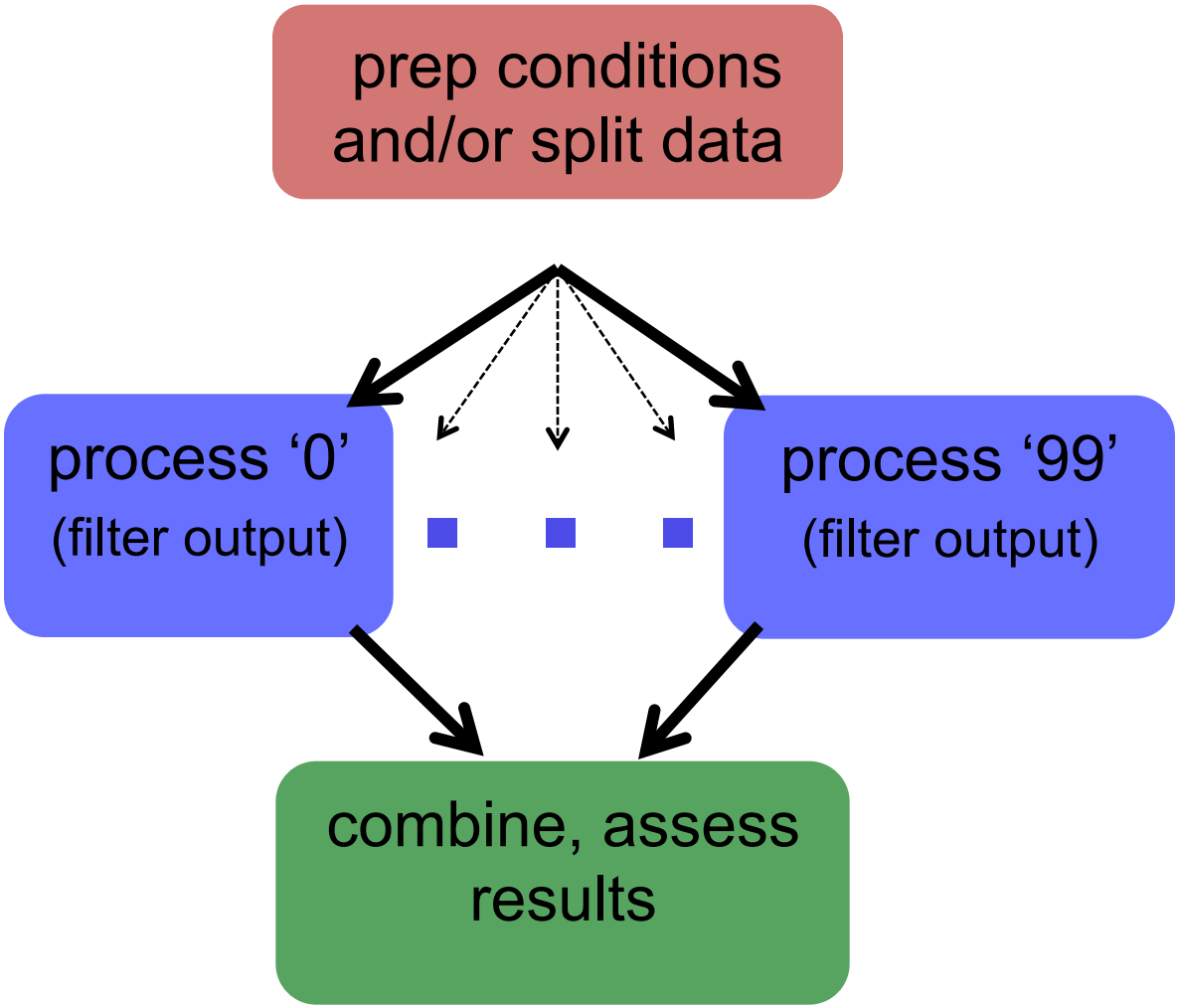
start with a major HTC "step" in the DAG…



… then add in another step …

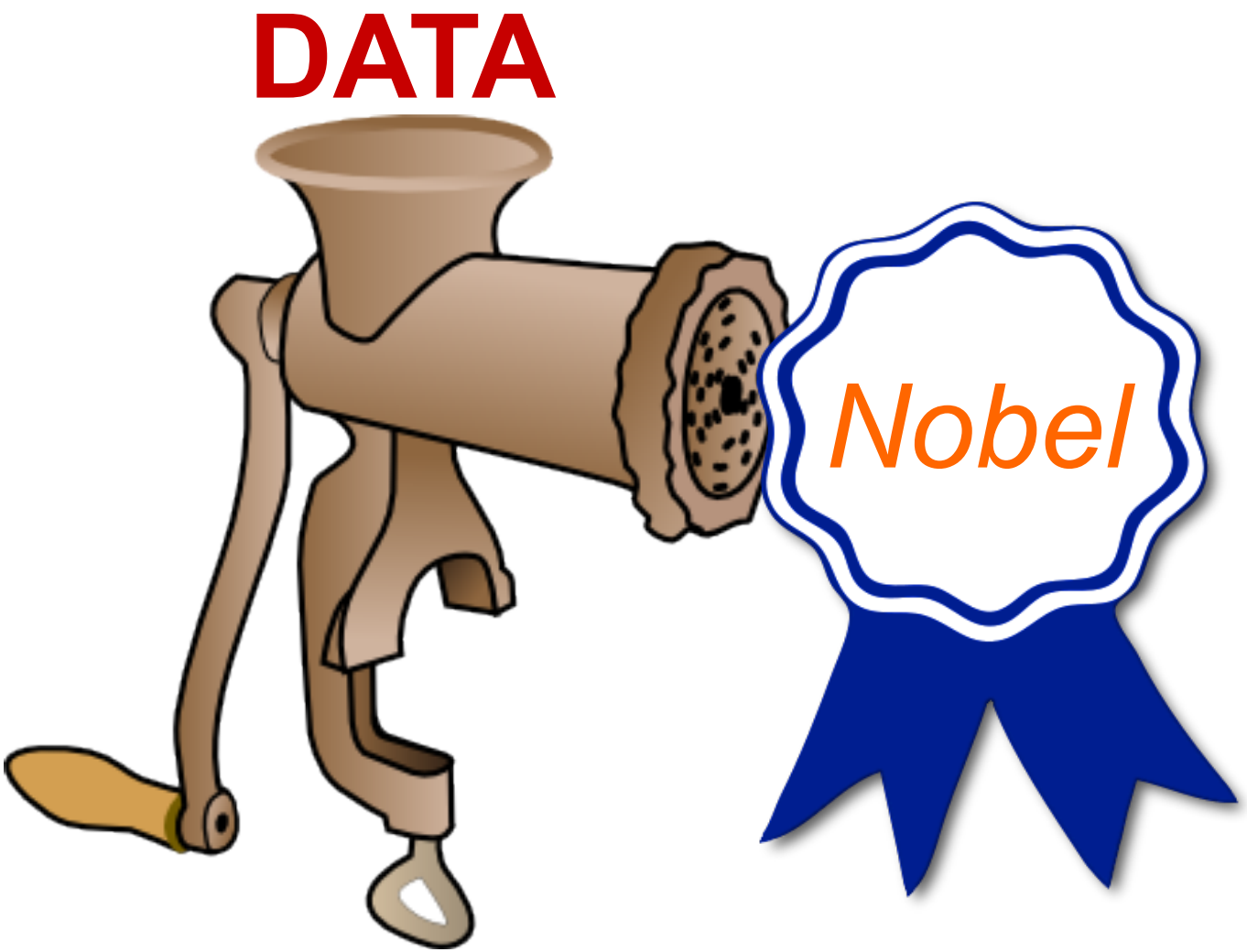


… and another step …



Open Science Grid

and End Up with This



Building a Good Workflow

1. Draw out the *general* workflow
2. Define details (test 'pieces' with HTCondor jobs)
   - divide or consolidate 'pieces'
   - determine resource requirements
   - identify steps to be automated or checked
3. Build it modularly; test and optimize
4. **Scale-up gradually**
5. What more can you automate or error-check?

(And remember to document!)

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*

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 100 per hour …

Waste the computer's time, not yours.

Maybe Not Worth The Time

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

- Others?

Scaling Out to OSG: Rules of Thumb

- CPU (single-threaded)
  - Best jobs run between **10 min** and **10 hrs**
    (Upper limit somewhat soft)
- Data (disk and network)
  - Keep scratch working space < 20 GB
  - Intermediate needs (/tmp?)
  - Use alternative data transfer appropriately
- Memory
  - Closer to 1 GB than 8 GB

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)

Scaling Up – Things to Think About

- More jobs:
  - 100-MB per input files may be fine for 10 or 100 jobs, but not for 1000 jobs. Why?
  - most submit queues will falter beyond ~10,000 total jobs
- Larger files:
  - more disk space, perhaps more memory
  - potentially more transfer and compute time

Be kind to your submit and execute nodes and to fellow users!

Scaling Solutions (More Jobs)

- Use a DAG to throttle the number of idle or queued jobs ("max-idle" and/or "DAGMAN CONFIG")
- Add more resiliency measures
  - "RETRY" (works per-submit file)
  - "SCRIPT POST" (use $RETURN, check output)
- Use SPLICE, VAR, and DIR for modularity/organization

Scaling Solutions (Larger Files)

- File manipulations
  - split input files to **send minimal data** with each job
  - **filter** input *and* output files to transfer only essential data
  - use compression/decompression

- Follow file delivery methods from yesterday for files that are still “large”

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

1. Changes to your code
   - Periodically save information about progress to a new file (every hour?)
   - At the beginning of script:
     - If progress file exists, read it and 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
```

Make It Run Everywhere

- What does an OSG machine have?
  - assume the worst
- Bring as much as possible with you:
  - won't that slow me down?
- Bring:
  - executable
  - likely, more of the "environment"

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

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

Error Checks Are Essential

If you don't check, it will happen…

- Check expected file existence, and repeat with a finite loop or number of retries
  - better yet, check *rough* file size too
- Advanced:
  - RETRY for *specific* error codes from wrapper
  - "periodic_release" for specific hold reasons

Open Science Grid

What to do if a check fails

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

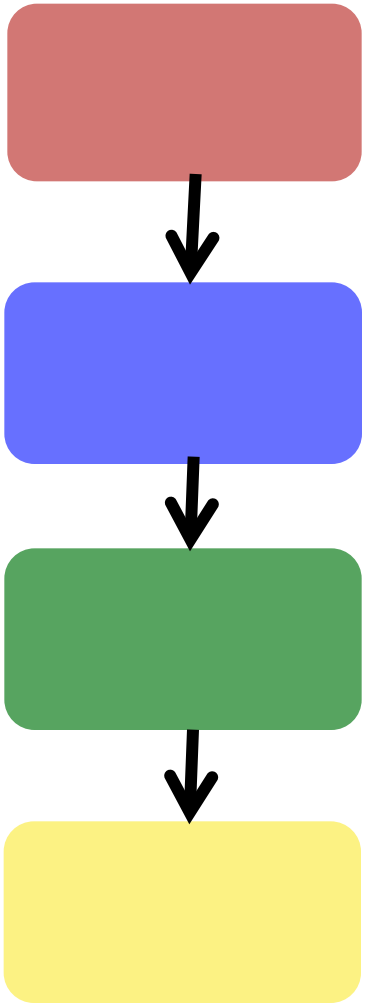
Performance Surprises

One bad node can ruin your whole day

- **"Black Hole" machines**
  - Depending on the error, email OSG!
- ***REALLY* slow machines**
  - use periodic_hold / periodic_release

Golden Rules for DAGs

- Beware of lo-ong shish kebabs!
  - (use self-checkpointing)
- Use PRE and POST script generously
- RETRY is your friend
- DIR and VAR for organization
- DAGs of DAGs are good
  - SPLICE
  - SUB_DAG_EXTERNAL



Building a Good Workflow

1. Draw out the *general* workflow
2. Define details (test 'pieces' with HTCondor jobs)
   - divide or consolidate 'pieces'
   - determine resource requirements
   - identify steps to be automated or checked
3. Build it modularly; test and optimize
4. Scale-up gradually
5. **What more can you automate or error-check?**

(And remember to document!)

Open Science Grid

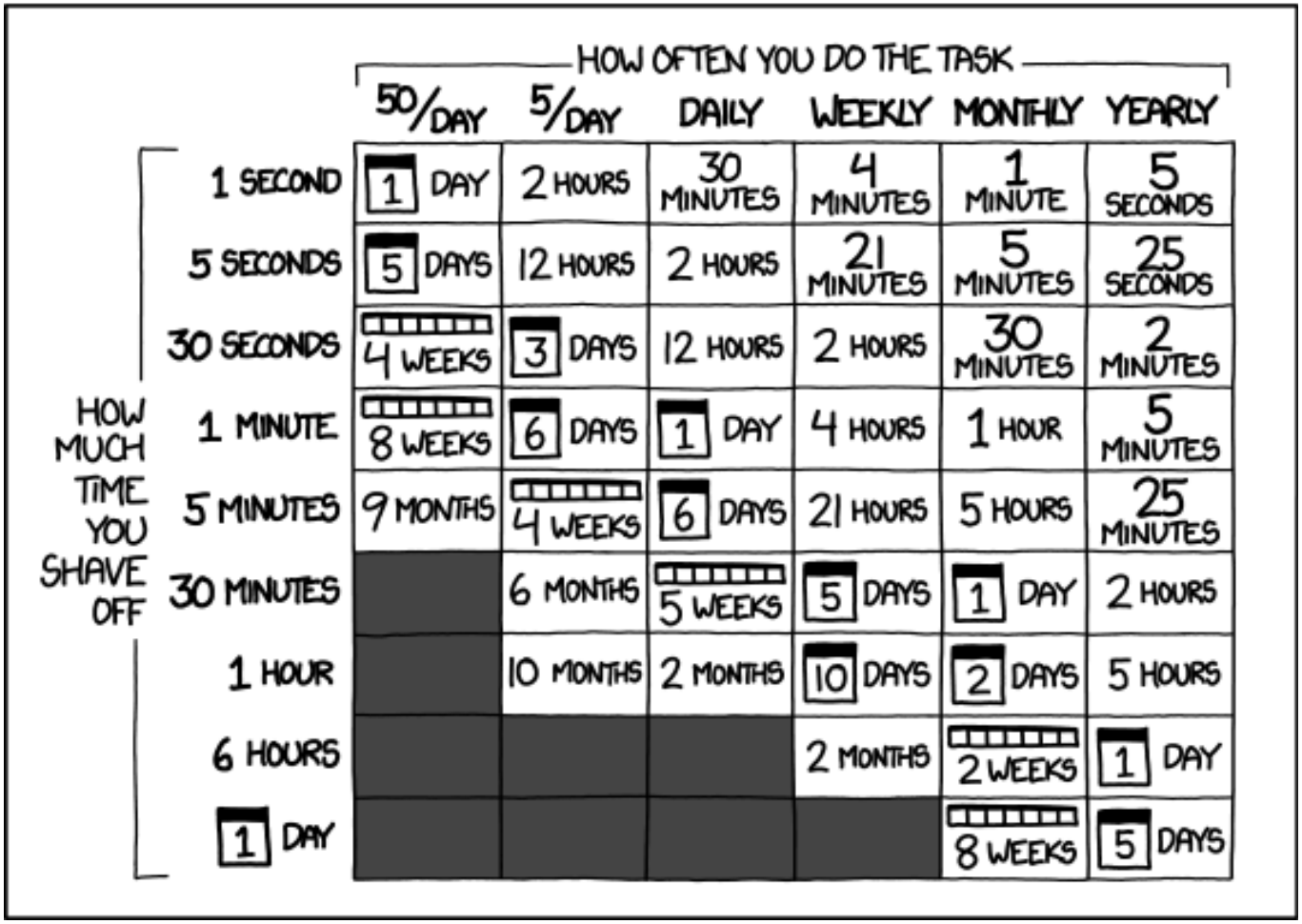
Automate *All* The Things

- well, not really, but kind of …
- Really: What is the minimal number of manual steps necessary?

  even 1 might be too many; zero is perfect!

- Consider what you get out of automation
  - time savings (including less ‘babysitting’ time)
  - reliability and reproducibility

Automation Trade-offs!



http://xkcd.com/1205/

Make It Work Unattended

- Remember the ultimate goal:

  Automation! Time savings!

- Things to automate:
  - Data collection?
  - Data preparation and staging
  - 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's *NOT* real science!
  - Work hard to make this happen.
  - It's *their* throughput, too.

Only ~10% of published cancer research is reproducible!

Building a Good Workflow

1. Draw out the *general* workflow
2. Define details (test 'pieces' with HTCondor jobs)
   - divide or consolidate 'pieces'
   - determine resource requirements
   - identify steps to be automated or checked
3. Build it modularly; test and optimize
4. Scale-up gradually
5. What more can you automate or error-check?

(And remember to document!)

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, explain the big picture

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?

Open Science Grid

Questions?

- Feel free to contact me:
  - lmichael@wisc.edu
- Now: Exercises 1.3 (1.4 Bonus)
- Next:
  - HTC Showcase!