

Workflows: from Development to Automated Production

Friday

Christina Koch ckoch5@wisc.edu
Research Computing Facilitators
University of Wisconsin - Madison

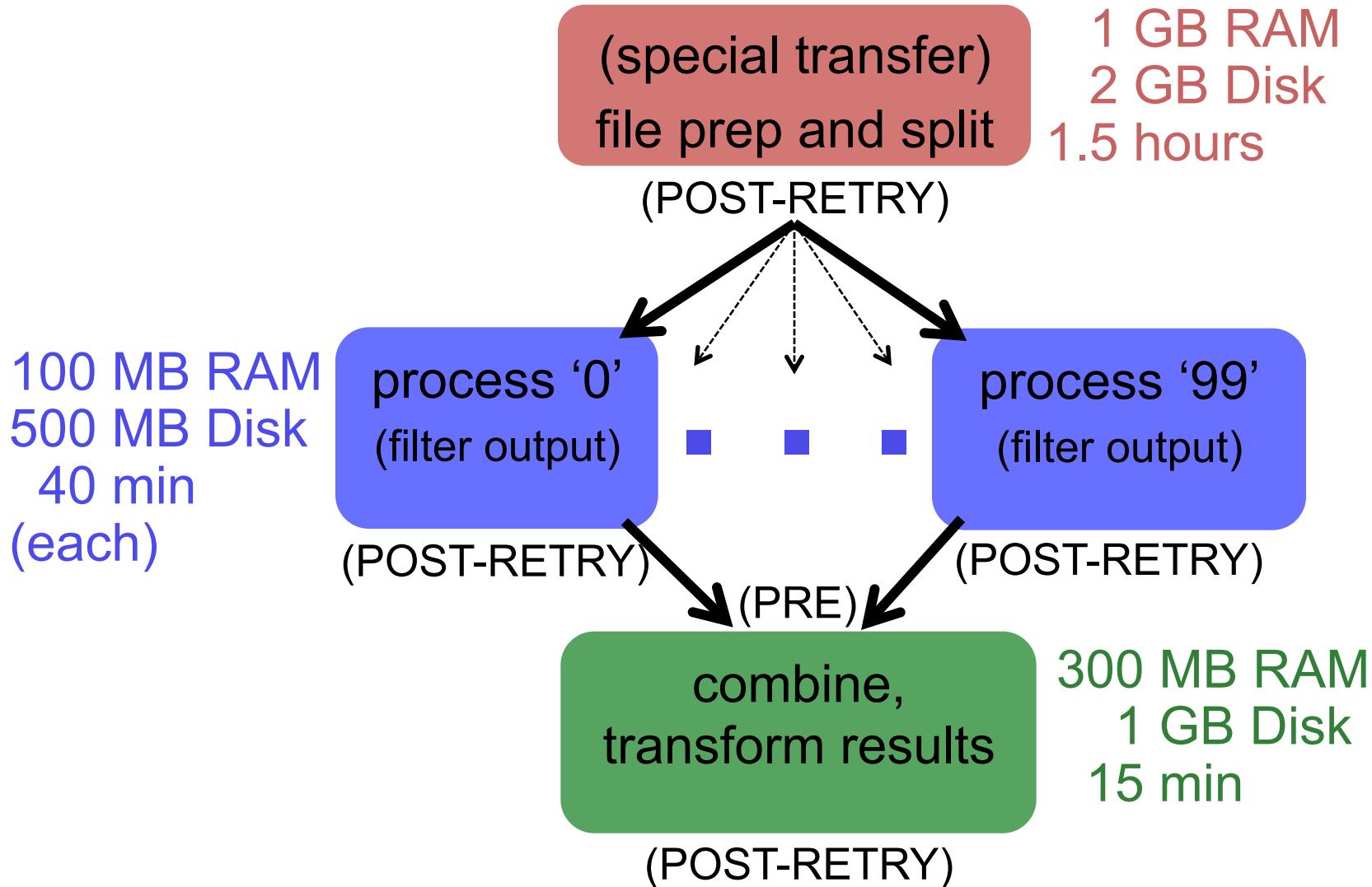
OR, GETTING THE MOST OUT OF WORKFLOWS, PART 2

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. Make it work consistently
6. What more can you automate or error-check?

(And remember to document!)

To Get Here ...



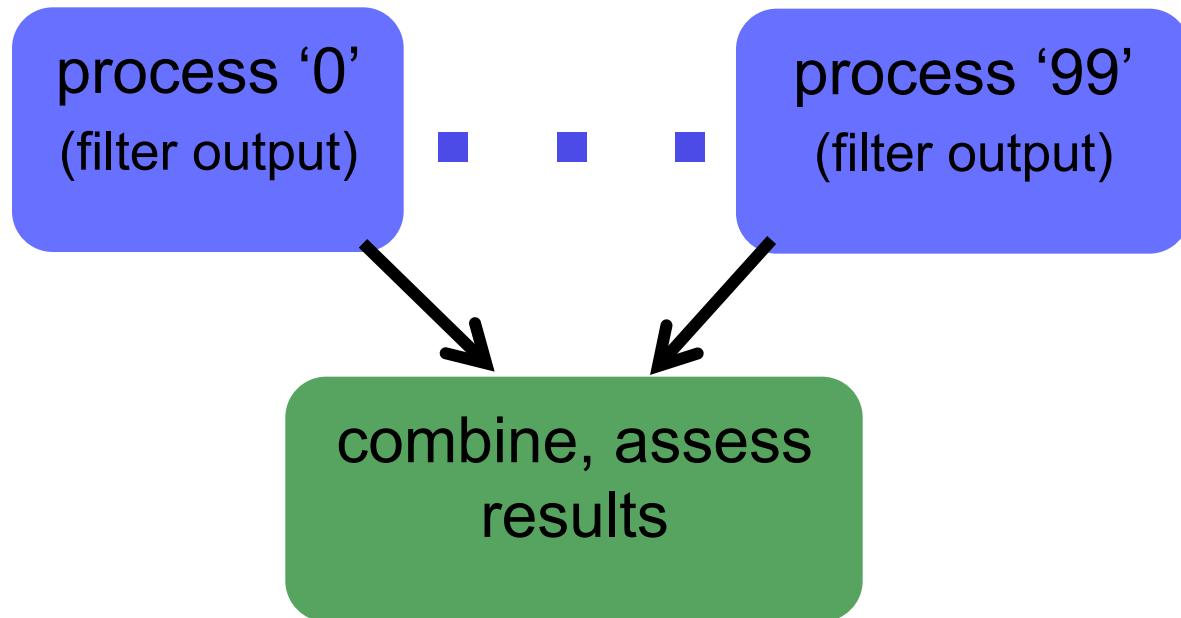
start with the HTC “step” in the DAG...

process '0'
(filter output)

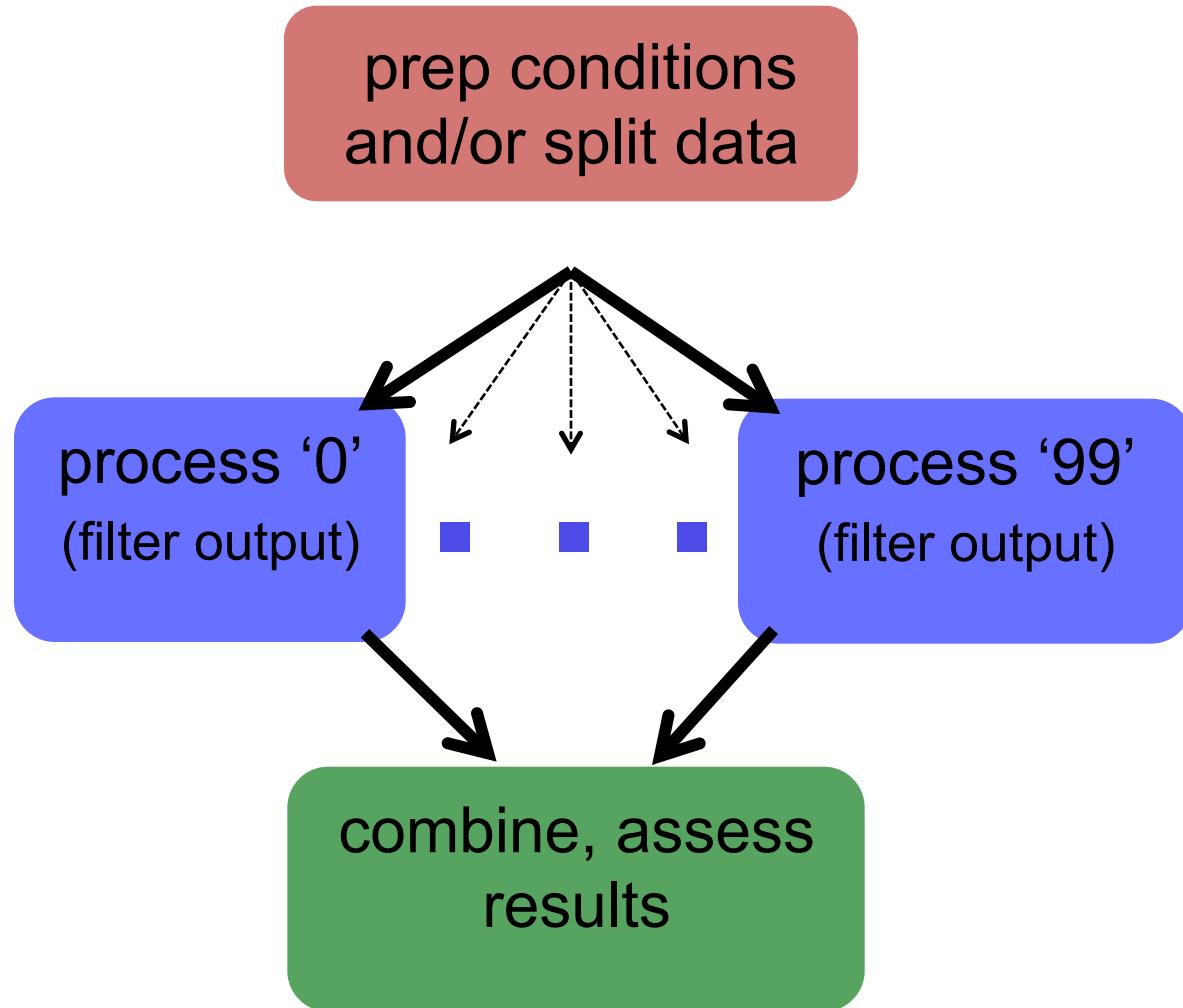


process '99'
(filter output)

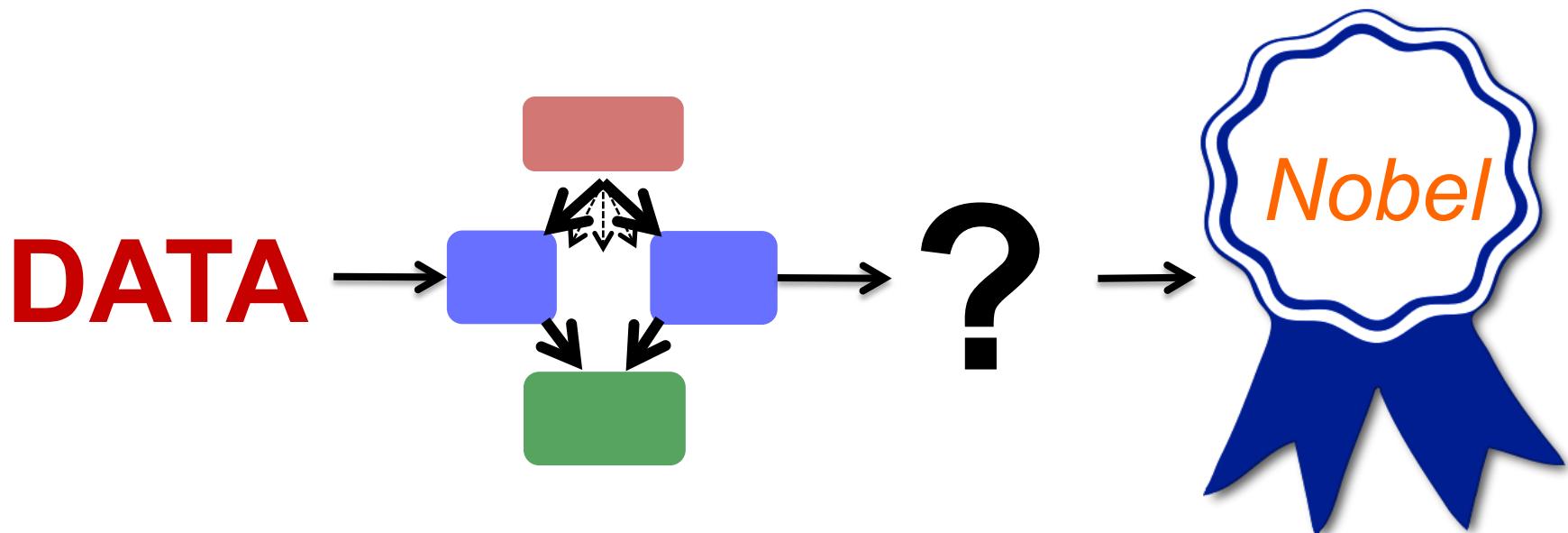
... then add in another step ...



... and another step ...



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. Make it work consistently
6. 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*



to the
moon!



Scaling Up: 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!**

Solutions for 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

Solutions for 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 (solution 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
```

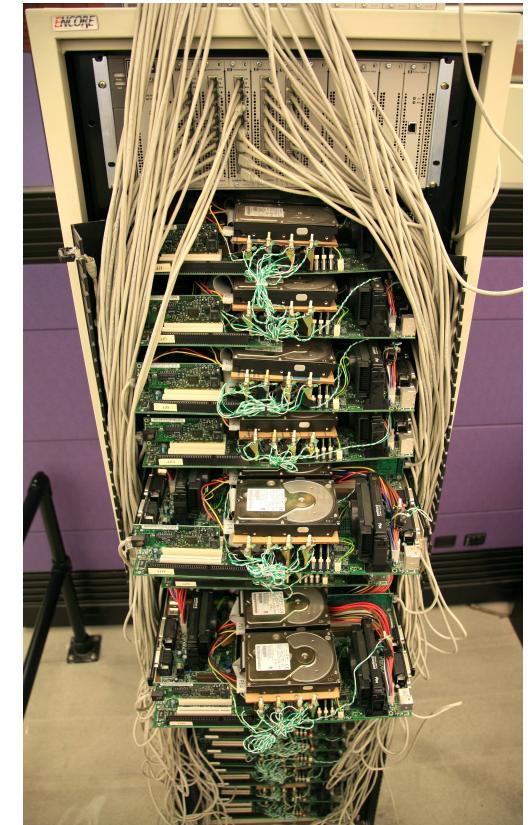
Building a Good Workflow

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. **Make it work consistently**
6. What more can you automate or error-check?

(And remember to document!)

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*



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`

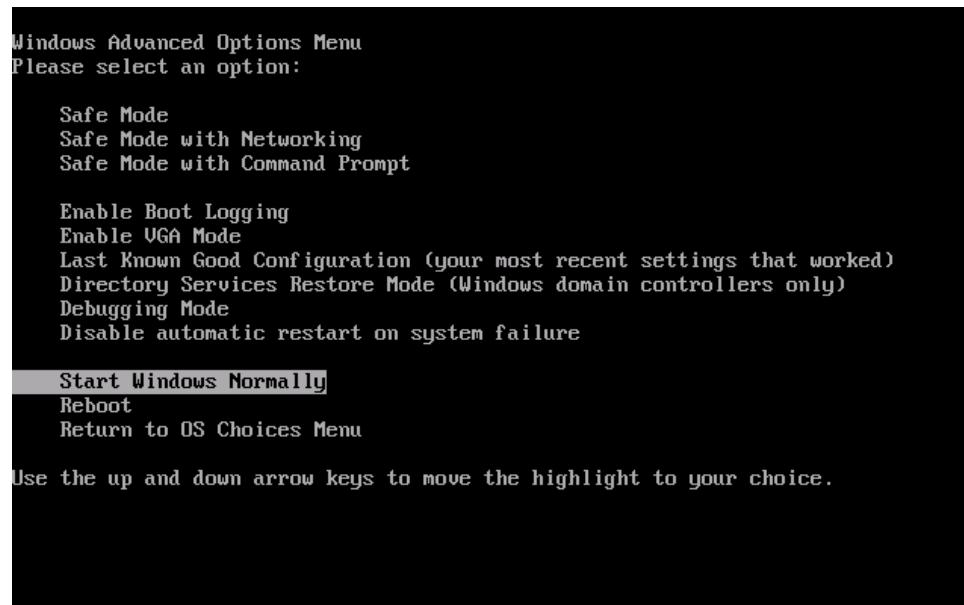
Error Checks Are Essential

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

What to do if a check fails

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



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. Make it work consistently
6. **What more can you automate or error-check?**

(And remember to document!)

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

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE
EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES	6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS	
	1 HOUR	10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS	
	6 HOURS			2 MONTHS	2 WEEKS	1 DAY	
	1 DAY				8 WEEKS	5 DAYS	

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 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. Make it work consistently
6. 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



Open Science Grid

PARTING THOUGHTS

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 ...

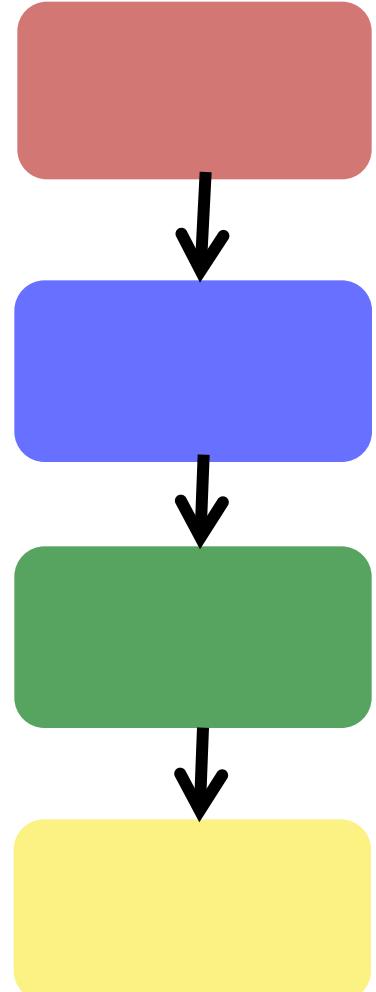
Waste the computer's time, not yours.

Maybe Not Worth The Time

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

Golden Rules for DAGs

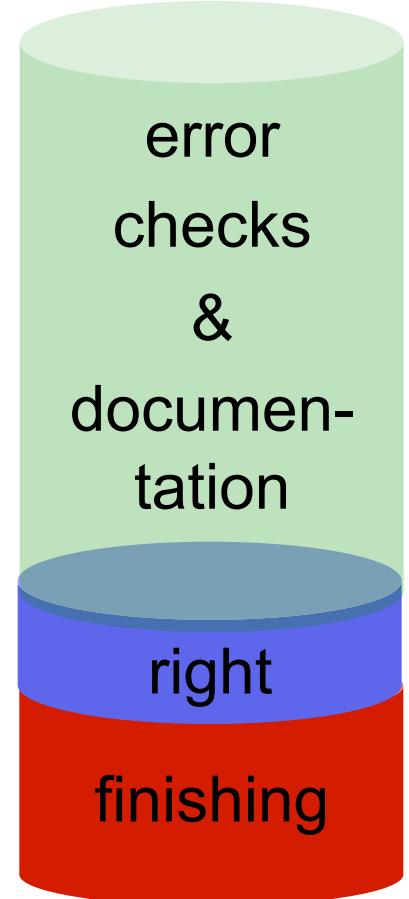
- Beware of lo-on^g shish kebabs!
 - (self-checkpointing)
- Use PRE and POST script generously
- RETRY is your friend
- DIR and VAR for organization
- DAGs of DAGs are good
 - SPLICE
 - SUB_DAG_EXTERNAL



If HTC workflows 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?

- Now: Exercises 2.1 (2.2 Bonus)
- Next:
 - HTC Showcase!