Open Science Grid

SANBI
South African National Bioinformatics Institute

Getting the Most out of HTC with Workflows

Peter van Heusden pvh@sanbi.ac.za
South African National Bioinformatics Institute
adapted from slides of
Christina Koch ckoch5@wisc.edu

Why are we here?

Why are we here?

To do SCIENCE!!!

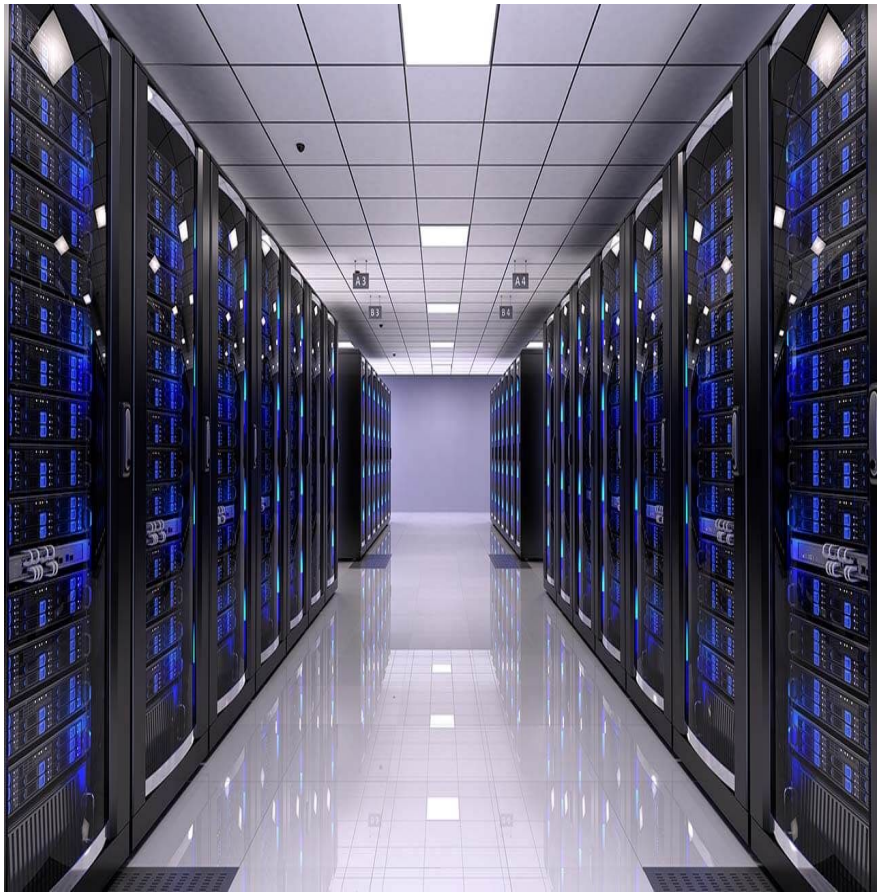
- A lot of science is best-done with computing – sometimes, LOTS of computing
- Science needs to be reproducible & sustainable
- And, we'd really like science to happen **fast**(er)

GETTING THE MOST OUT OF COMPUTING (FOR RESEARCH)

Scaling up computing

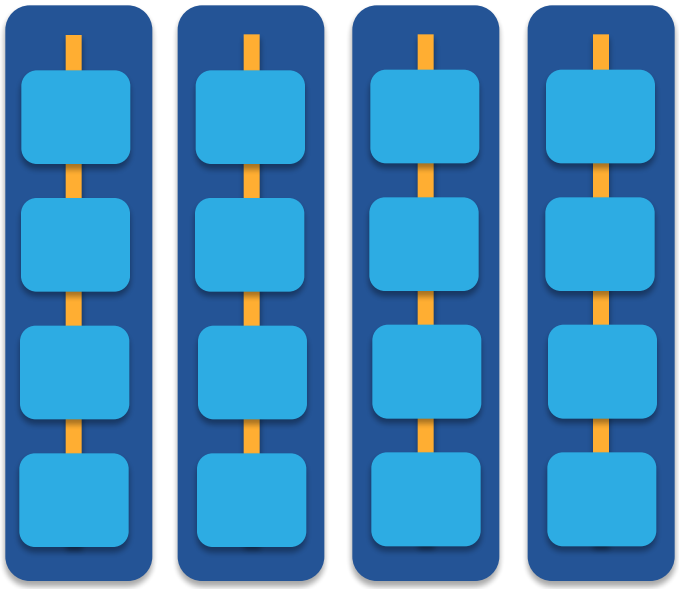
From

TO



Computing types

- Our challenge: how to make use of computers working together to tackle large compute tasks...



high-throughput

high-performance (e.g.MPI)

Two Strategies

Cloud

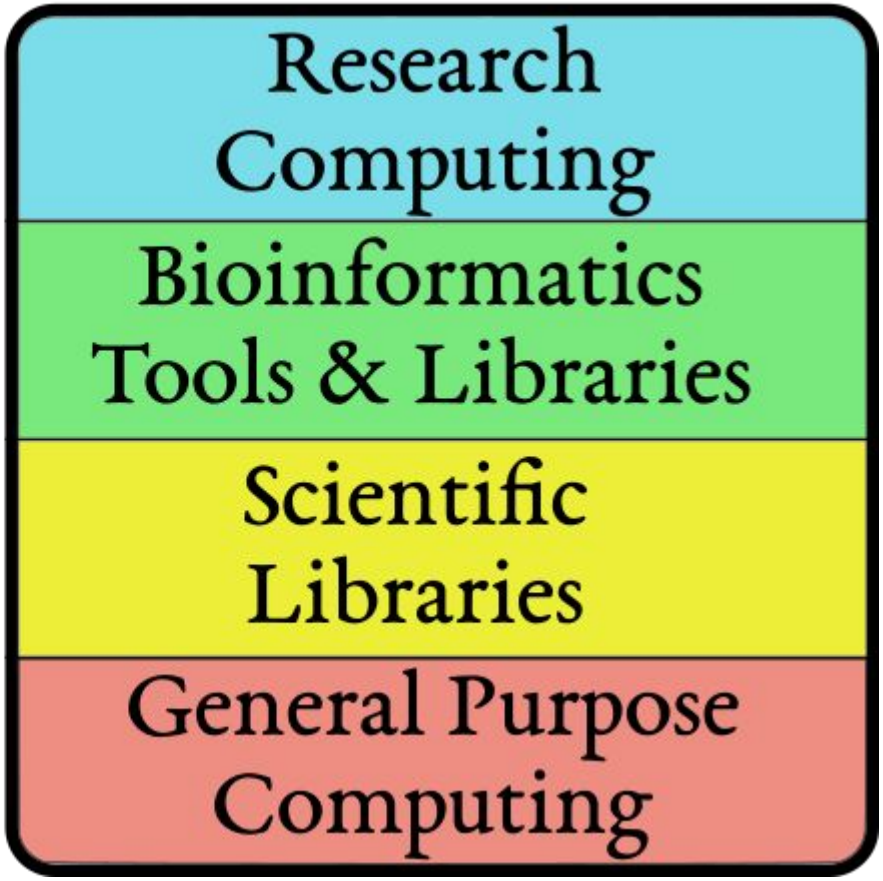
Focus: Service *many user groups* by providing *generic* computing

Skills focus: *systems* engineering to create *virtual infrastructure*, compose *multiple component* services

HPC Cluster

- Focus: Service *specialised* computing groups working on *computationally challenging* problems
- Skills focus: *research software* engineering and *parallel algorithms*

The Research Computing Stack



biopython

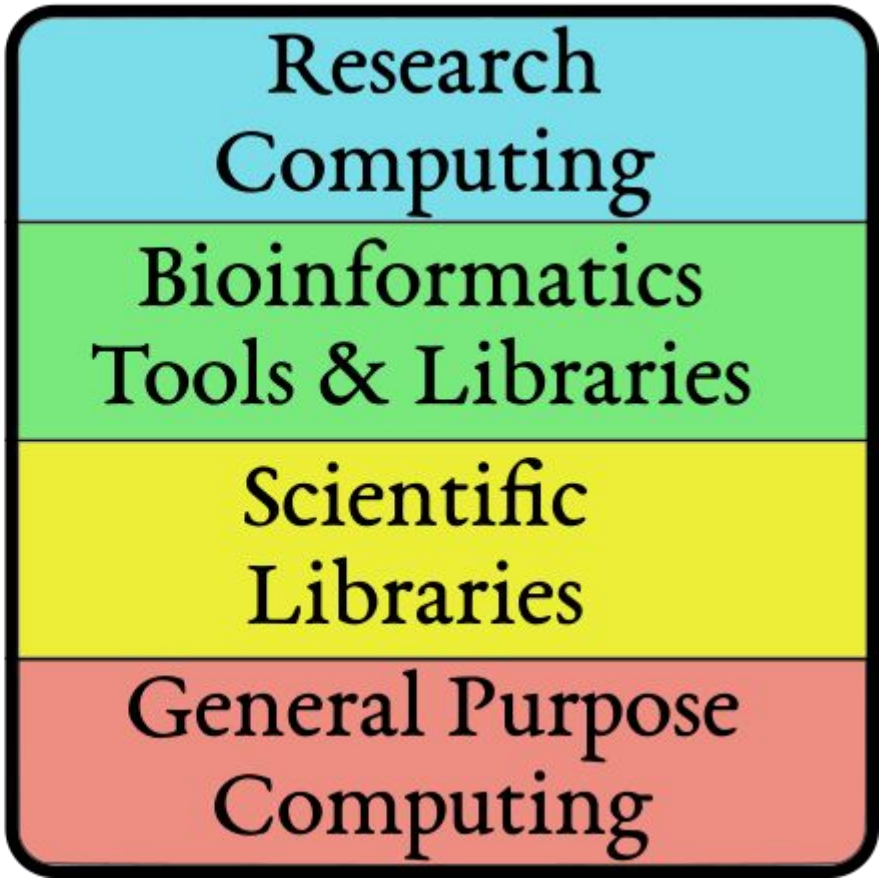
Open MPI

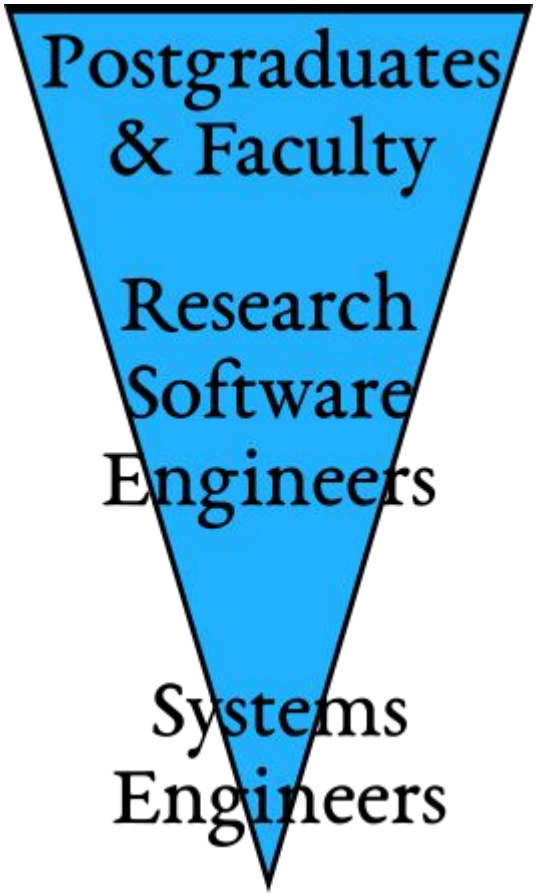
jupyter

Linux

openstack

Research Computing Roles





Two Architectures

High Throughput

Focus: Workflows with many *small*, *largely independent* compute tasks

Optimize: *throughput*, or time from *submission* to *overall completion*

High Performance

- Focus: Workflows with *large, highly coupled* tasks
- Optimize: *individual tasks*, software, communication between processes

Making Good Choices

- How do you choose the best approach?
- Guiding question:

Is your problem “HTC-able”?

Typical HTC Problems

- batches of similar program runs (>10)
- “loops” over independent tasks
- others you might not think of …
  - programs/functions that
    - process files that are already separate
    - process columns or rows, separately
    - iterate over a parameter space
  - *a lot* of programs/functions that use multiple CPUs on the same server

Ultimately: Can you break it up?

What is not HTC?

- fewer numbers of jobs
- jobs individually requiring significant resources
  - RAM, Data/Disk, # CPUs, time

  (though, “significant” depends on the HTC compute system you use)
- restrictive licensing

The Real World

- However, it's not just about finding the right computing approach to your problem.
- These approaches will be **most** effective if they're running on appropriate compute systems.

The Real World

- Not all compute systems are created equal.

- Two questions to ask:

What resources are available to me?

Which one is the best match for the kind of computing I want to do?

Campus Resources

- Start with your local campus compute system
- Some considerations:
  - Who has access? Are there allocations?
  - What kind of system? What is it optimized for?
- An HPC cluster may not handle lots of jobs well, in the same way that an HTC system has limited multicore capabilities - be aware of how a system matches/doesn't match your computation strategy.
- Ask questions! Be a good citizen!
- If local resources are limited, explore other options.

Beyond your campus

- Hosted Resources

Galaxy PROJECT

CYVERSE™

- Clouds and clusters
  - Commercial cloud systems
  - Research clouds and clusters

The payoff

- HTC is, beyond everything, scalable
  - If you can run 10 jobs, you can run 10,000, maybe even 10 million
- Worth pursuing the right kind of resources (if you can) for the right kind of problem.

GETTING THE MOST OUT OF HTC

Key HTC Tactics

1. **Increase Overall Throughput**
2. Utilize Resources Efficiently!
3. Bring Dependencies With You
4. Scale Gradually, Testing Generously
5. Automate As Many Steps As Possible

Throughput, revisited

- In HTC, we optimize *throughput*: time from submission to overall completion
- Instead of making individual jobs as fast as possible, optimize how long it takes for all jobs to finish.
- Time to completion includes *engineering* time
- We do this by breaking large processes into smaller pieces and re-using components

Breaking up is hard to do…

- Ideally into parallel (separate) jobs
  - reduced job requirements = more matches
  - not always easy or possible
- Strategies
  - break HTC-able steps out of a single program
  - break up loops
  - break up input
- Use self-checkpointing if jobs are too long
  - Often not supported by individual applications

Batching (Merging) is easy

- A single job can
  - execute multiple independent tasks
  - execute multiple short, sequential steps
  - avoid transfer of intermediate files
- Use scripts!
  - need adequate error reporting for each “step”
  - easily handle multiple commands and arguments

Key HTC Tactics

1. Increase Overall Throughput
2. **Utilize Resources Efficiently!**
3. Bring Dependencies With You
4. Scale Gradually, Testing Generously
5. Automate As Many Steps As Possible

Know and Optimize Job Use of Resources!

- **CPUs** ("1" is best for matching)
  - restrict, if necessary/possible
  - software that uses all available CPUs is BAD!
- **CPU Time**

  > ~5 min, < ~1 day; **Ideal: 1-2 hours**
- **RAM** (not always easily modified)
- **Disk** per-job (execute) and in-total (submit)
- **Network Bandwidth**
  - minimize transfer: filter/trim/delete, compress

Key HTC Tactics

1. Increase Overall Throughput
2. Utilize Resources Efficiently!
3. **Bring Dependencies With You**
4. Scale Gradually, Testing Generously
5. Automate As Many Steps As Possible

Bring *What* with You?

- Software
  - Dependency management
- Data and other input files
  - Have a Research Data Management strategy
  - Record parameters
    - do you use “reference data”?
    - including random number seeds
    - note data provenance

Each Workflow Step Has a Wrapper

- Before task execution
  - transfer/prepare files and directories
  - setup/configure software environment and other dependencies
- Task execution
  - prepare complex commands and arguments
  - batch together many ‘small’ tasks
- After task execution
  - filter/combine/compress files and directories
  - check for and report on errors
  - clean up temporary files

Software Dependency Management

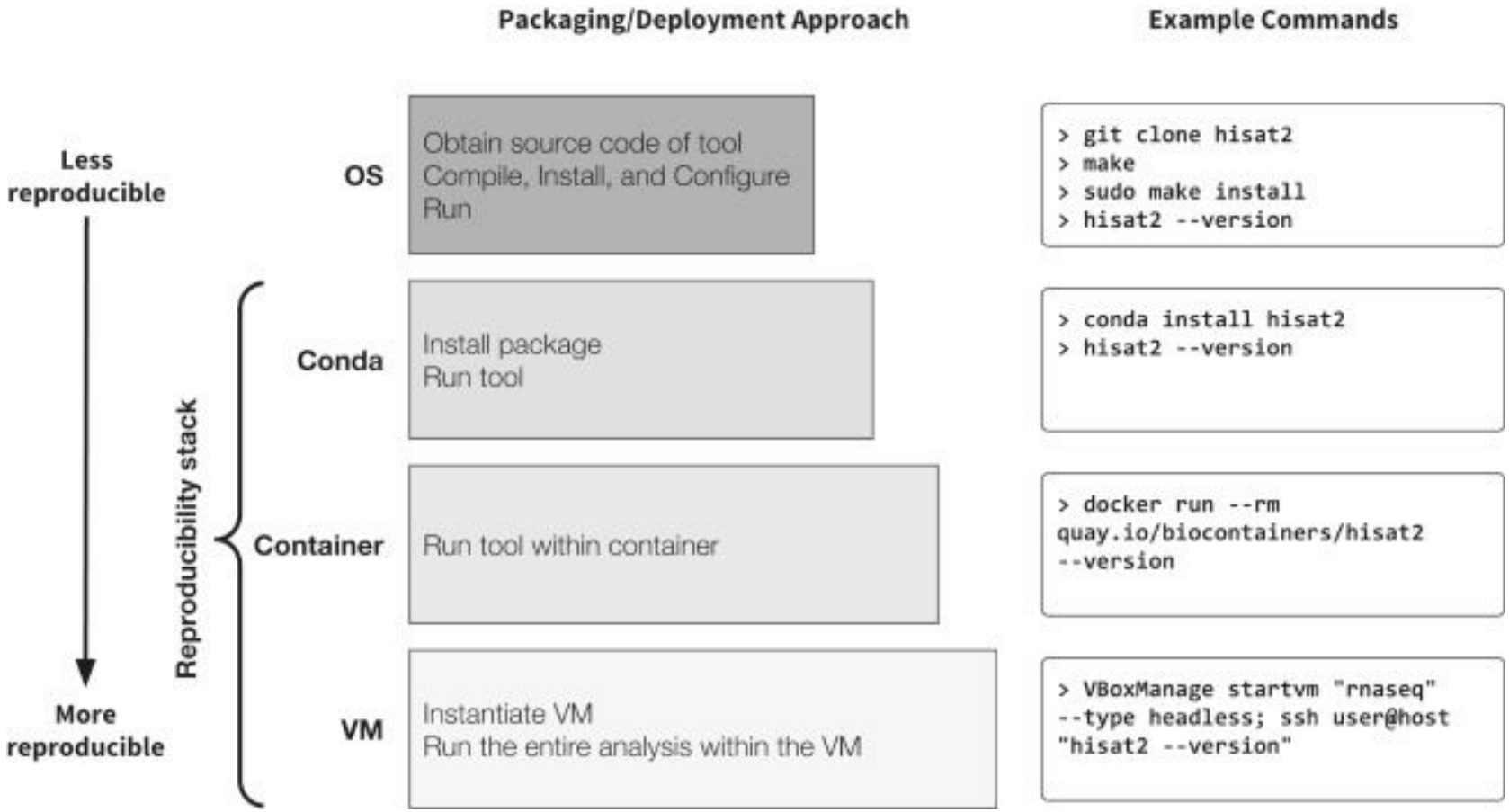


Image from Grüning et al 2018 "Practical Computational Reproducibility in the Life Sciences"

Key HTC Tactics

1. Increase Overall Throughput
2. Utilize Resources Efficiently!
3. Bring Dependencies With You
4. **Scale Gradually, Testing Generously**
5. Automate As Many Steps As Possible

Testing, testing, testing!

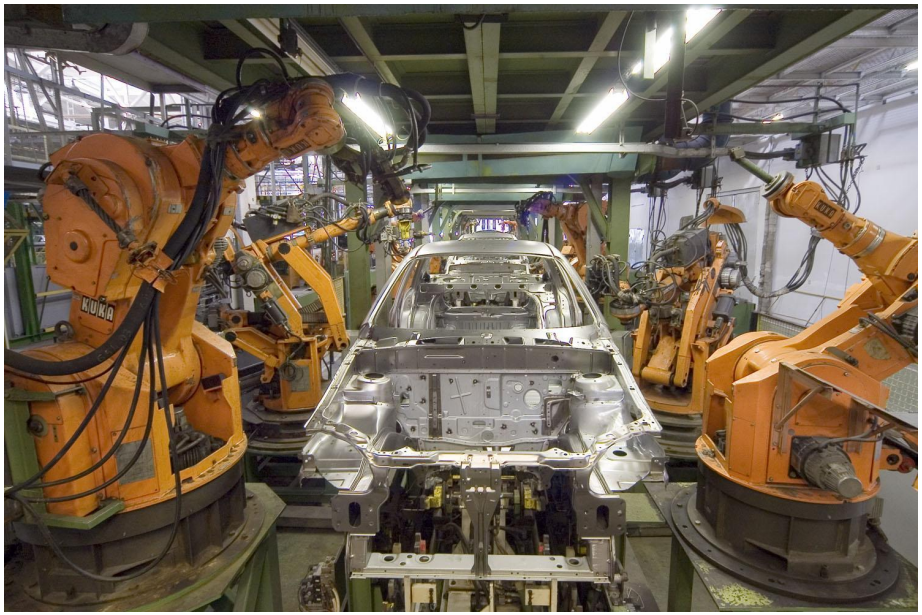
- Allows you to optimize resource use
- Just because it worked for 10 jobs, doesn't mean it will work for 10,000 jobs (scaling issues)
  - Data transfer (in and out)
  - Discover site-specific problems

Key HTC Tactics

1. Increase Overall Throughput
2. Utilize Resources Efficiently!
3. Bring Dependencies With You
4. Scale Gradually, Testing Generously
5. **Automate As Many Steps As Possible**

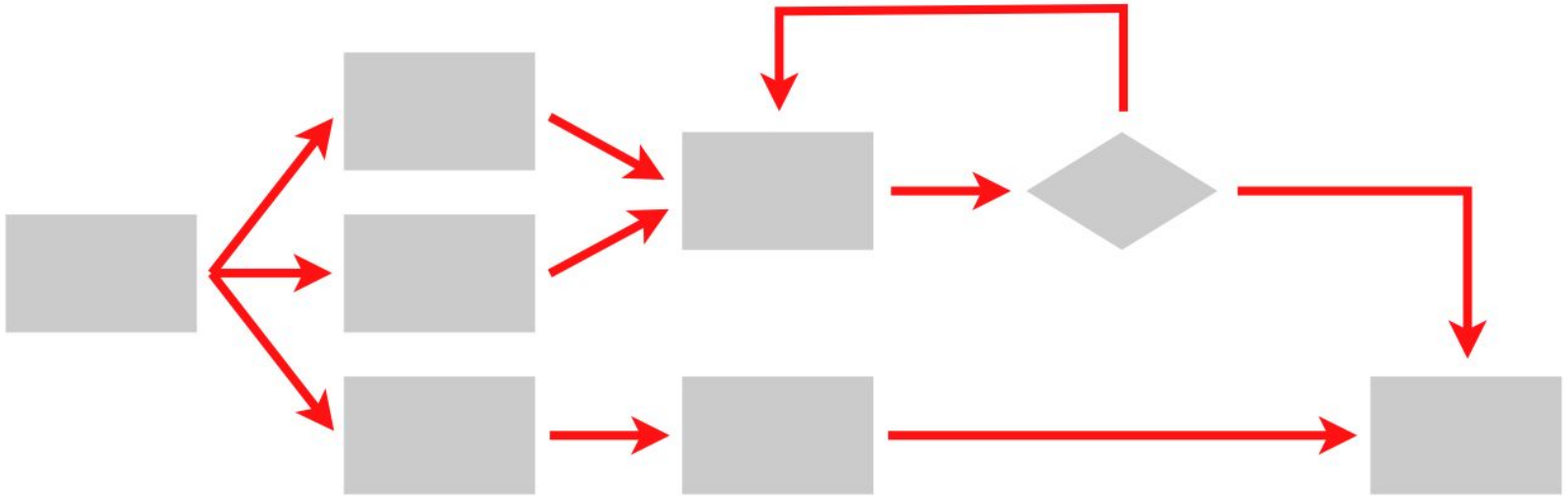
What to Automate?

- Submitting many jobs
- Writing submit files using scripts
- Running a series of jobs, or workflow



What is a workflow?

- A series of ordered steps
  - Steps
  - Connections
  - (Metadata)

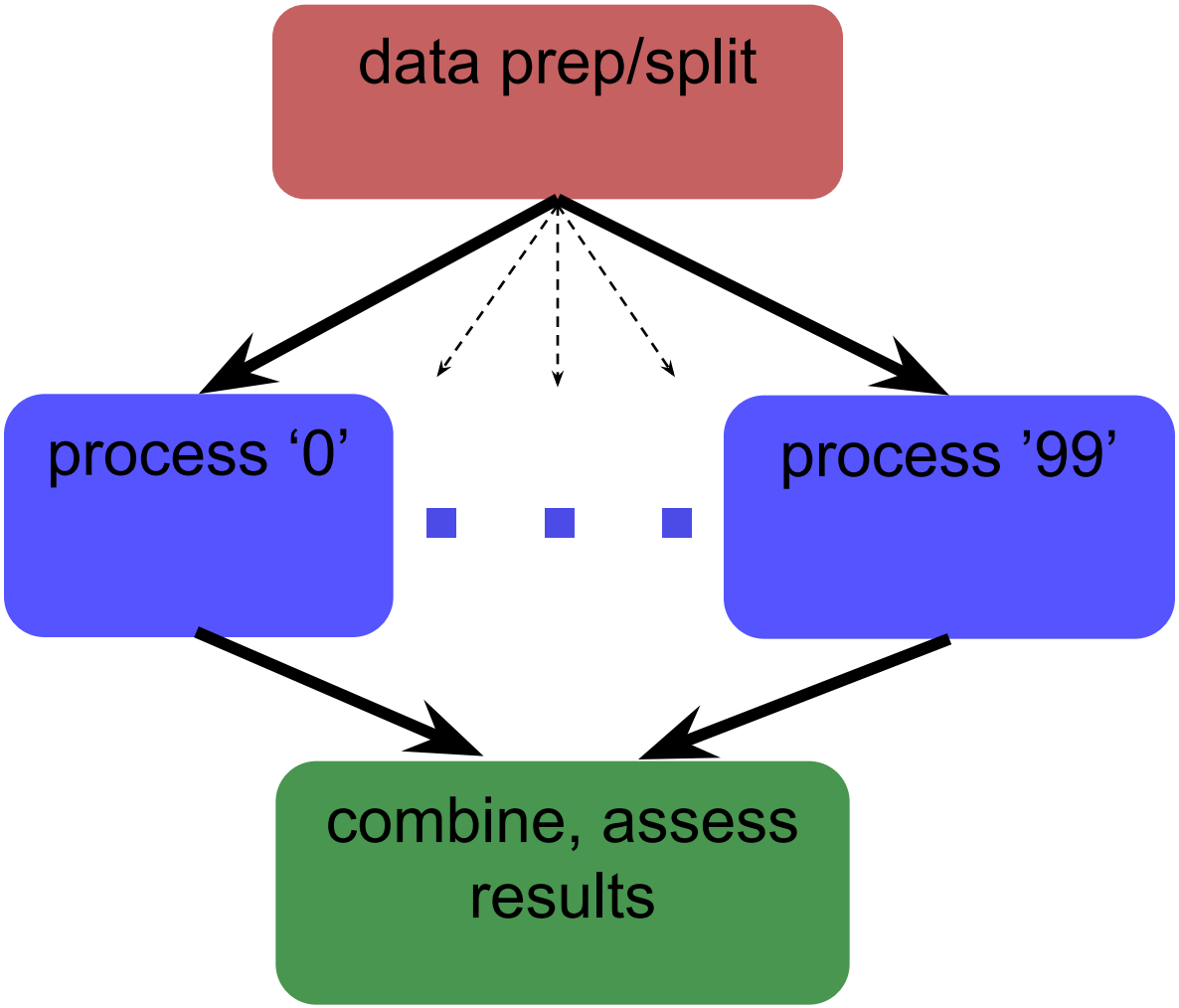


We ♥ workflows

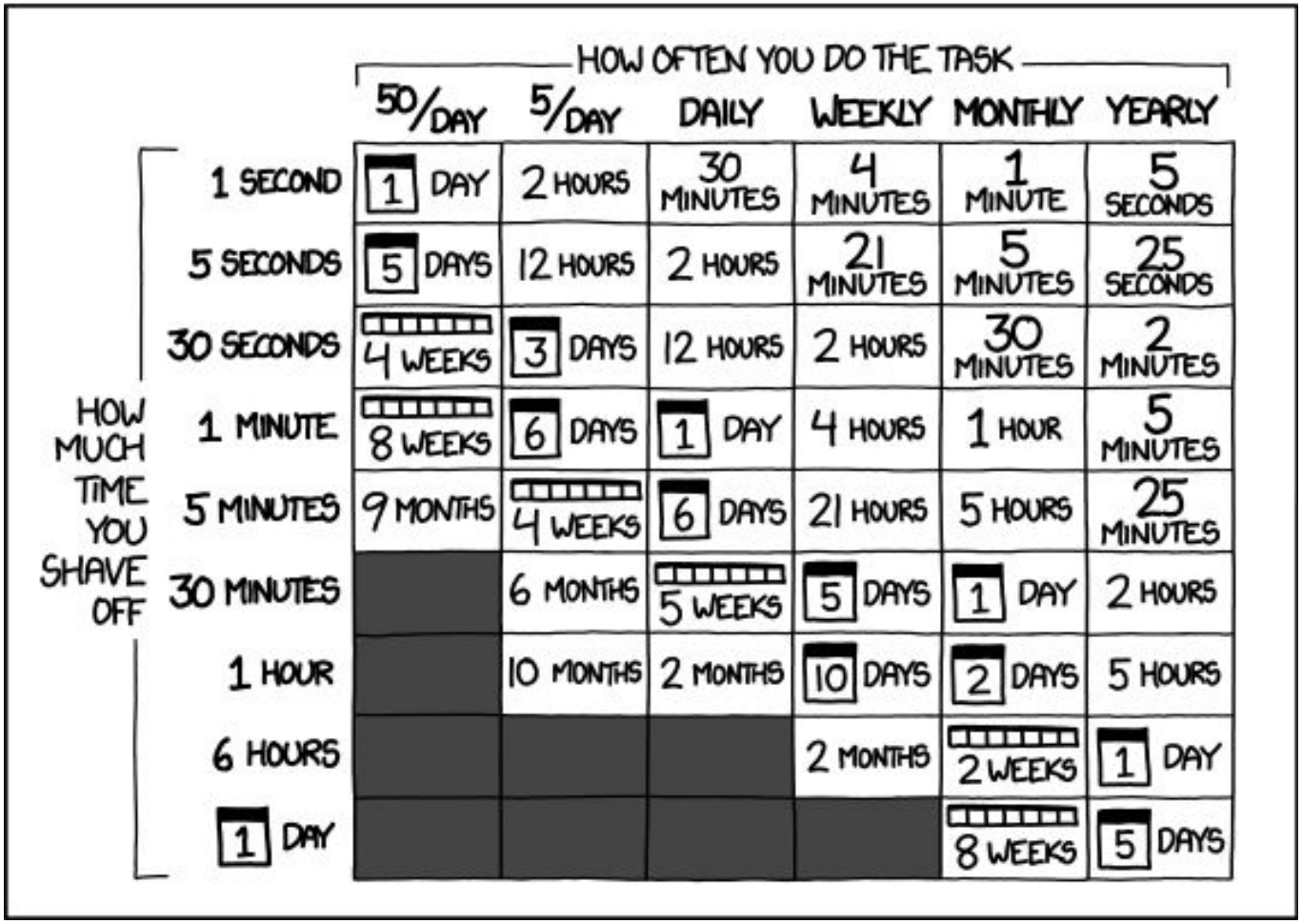
- non-computing "workflows" are all around you, especially in science
  - instrument setup
  - experimental procedures and protocols

- when planned/documented, workflows help with:
  - organizing and managing processes
  - saving time with **automation**
  - objectivity, reliability, and reproducibility (THE TENETS OF GOOD SCIENCE!)

Scientific Workflow Management



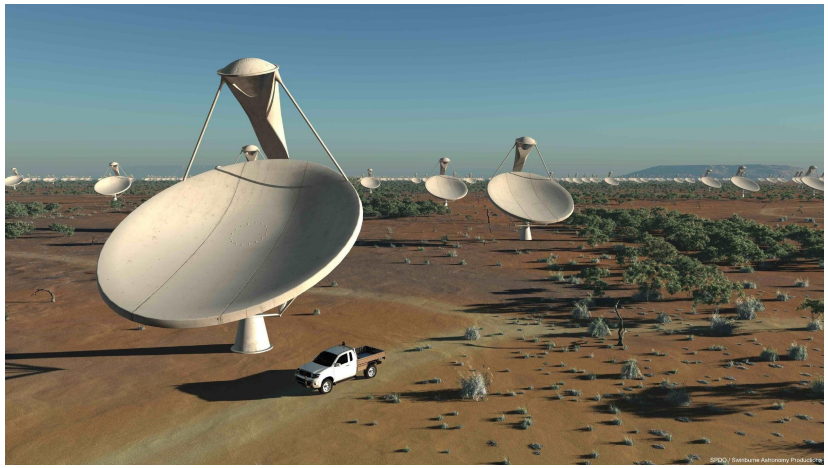
Automating workflows can save you time...



http://xkcd.com/1205/

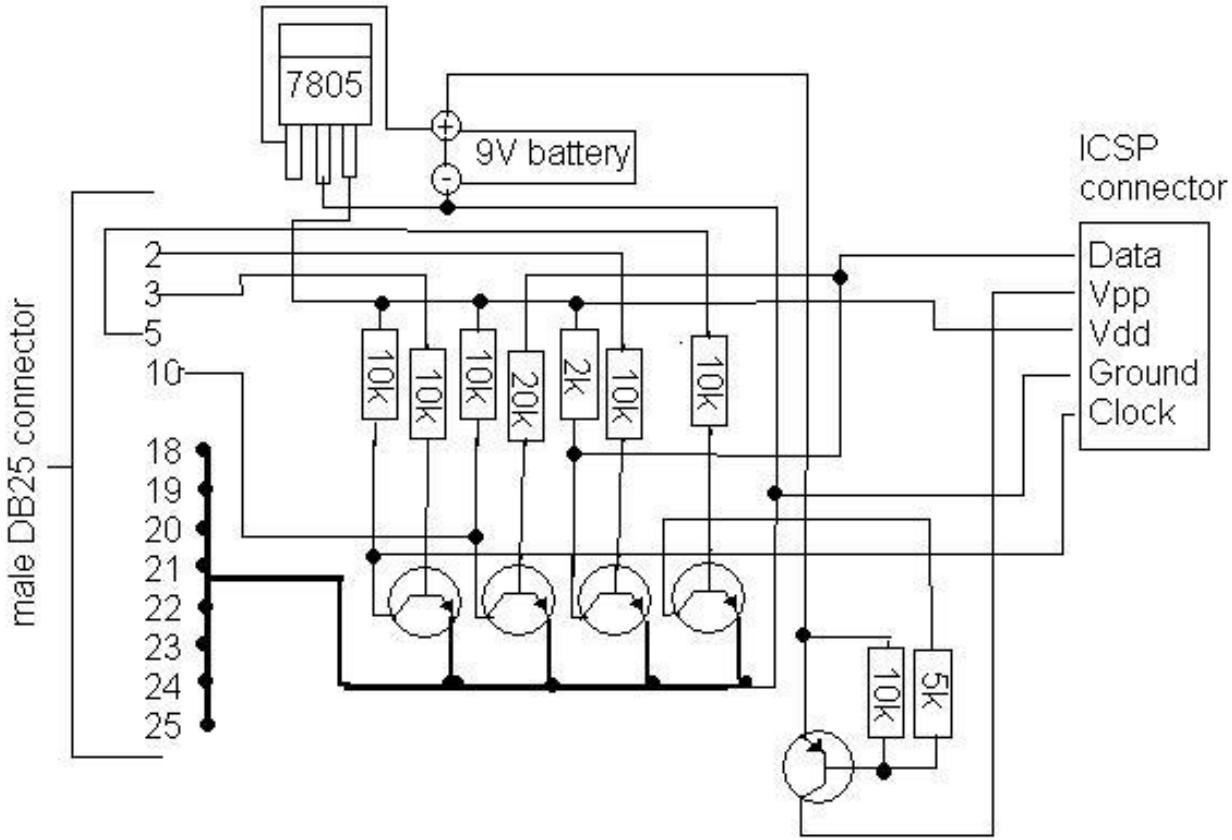
… but there are even more benefits of automating workflows

- Reproducibility
- Building knowledge and experience
- New ability to imagine greater scale, functionality, possibilities, and better SCIENCE!!

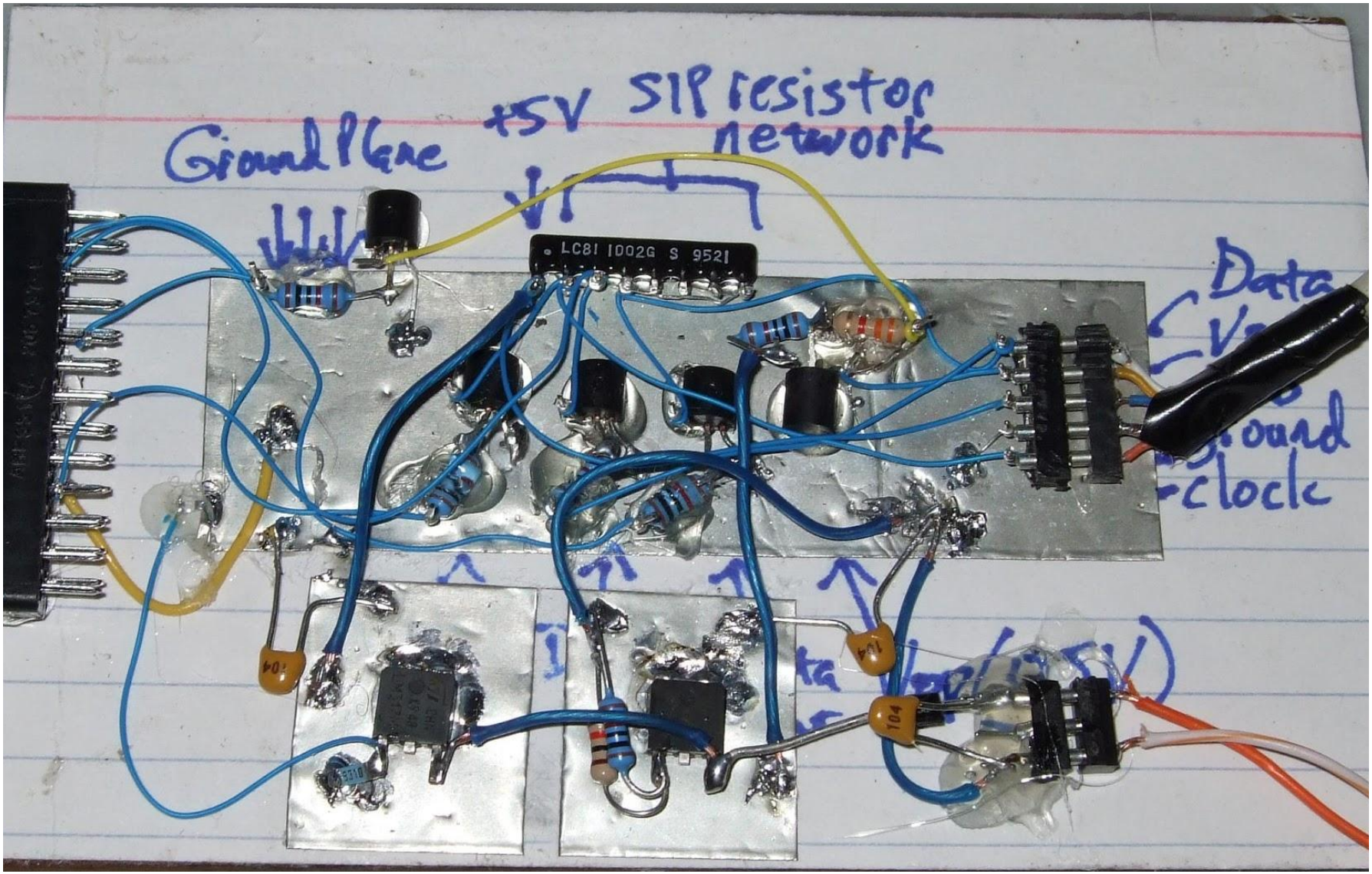


GETTING THE MOST OUT OF WORKFLOWS, PART 1

From schematics…



… to the real world

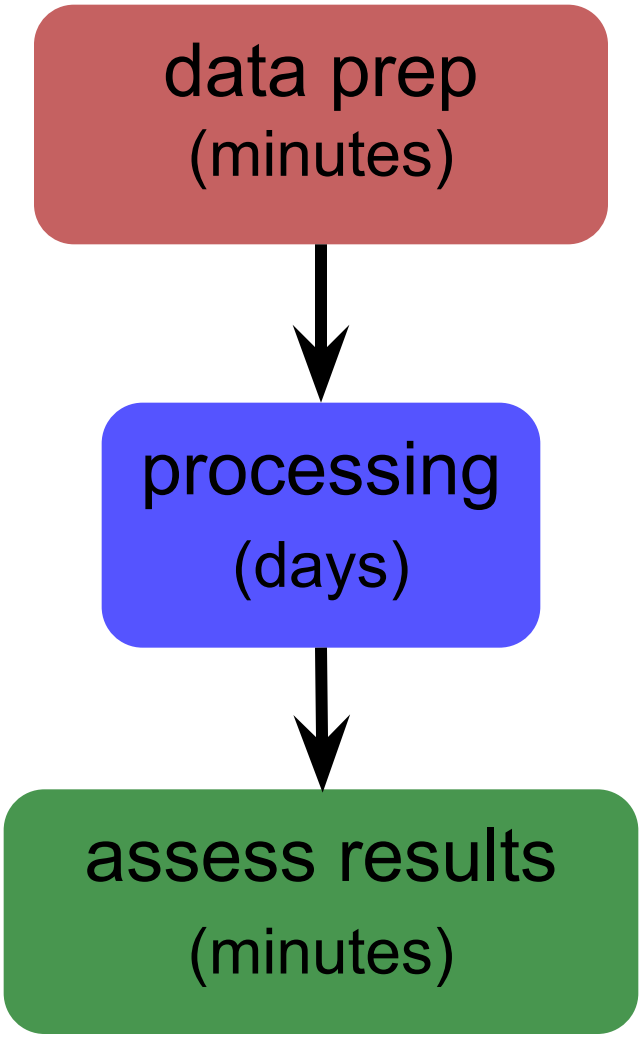


Building a Good Workflow

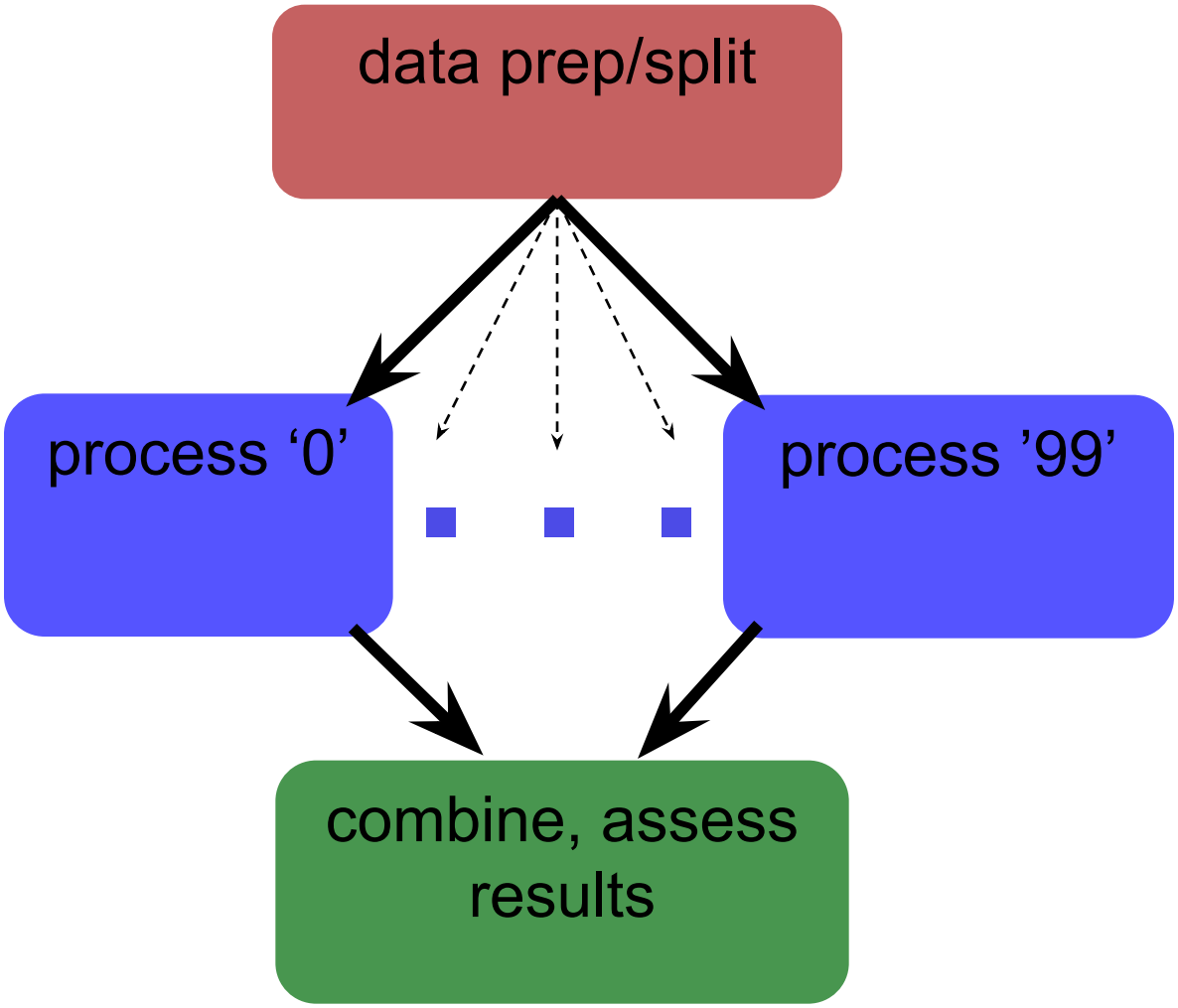
1. **Draw out the *general* workflow**
2. Define details (test ‘pieces’ with HTCondor jobs)
   - divide or consolidate ‘pieces’
   - determine resource requirements
   - identify steps to be automated or checked
3. Build it modularly; test and optimize
4. Scale-up gradually
5. Make it work consistently
6. What more can you automate or error-check?
7. Publish, share and re-use

(And remember to document!)

Workflow, version 1



Workflow, version 2 (HTC)



Building a Good Workflow

1. Draw out the *general* workflow
2. **Define details (test 'pieces': steps / scripts)**
   - **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?

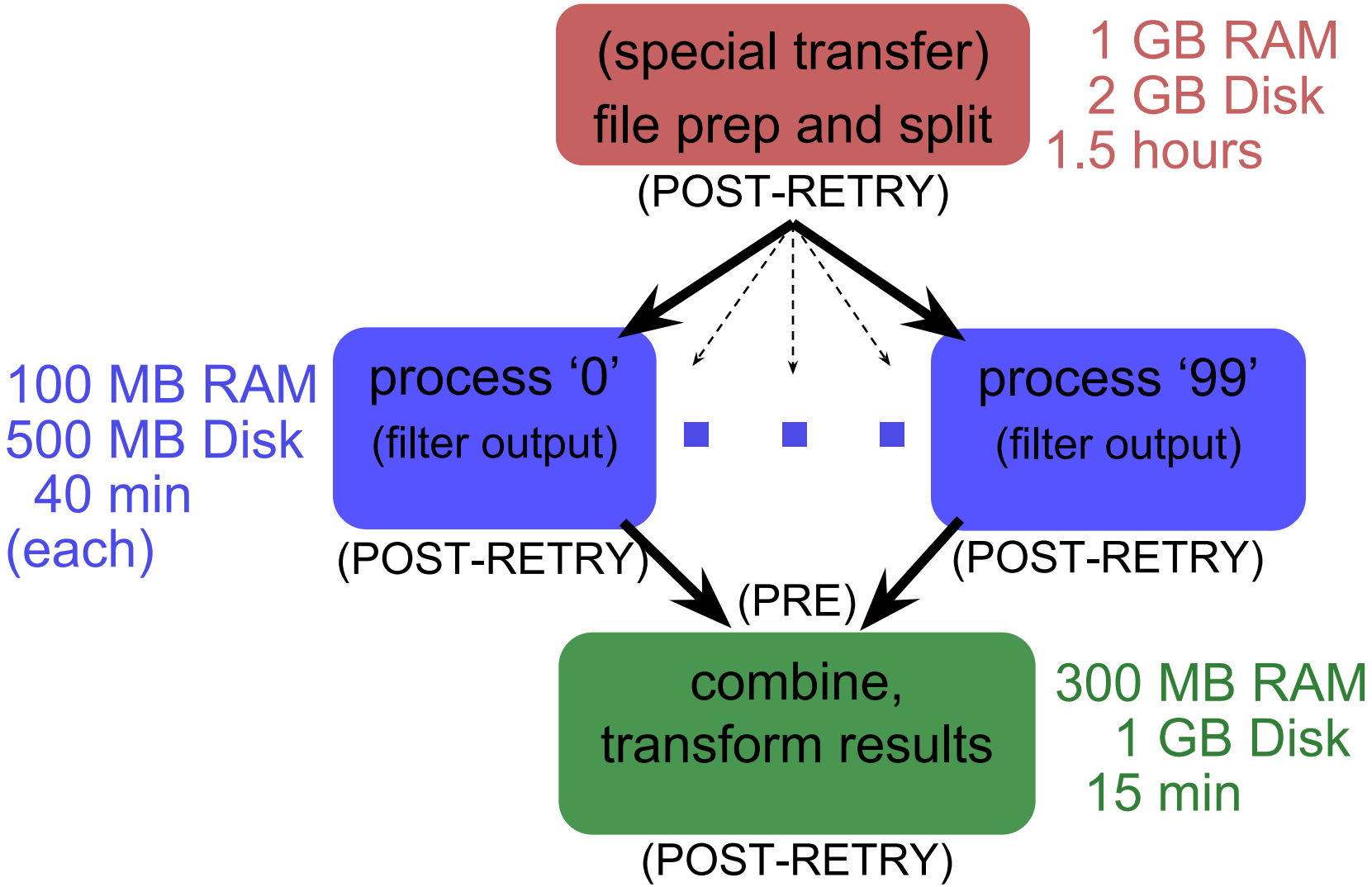
(And remember to document!)

Determine Resource Usage

- Run locally first
- Then get one job running remotely
  - (on execute machine, not submit machine)!
  - get the logistics correct! (job submission, file and software setup, etc.)
- Once working, run a couple of times
  - If big variance in resource needs, should you take the…

    Average? Median? Worst case?

End Up with This

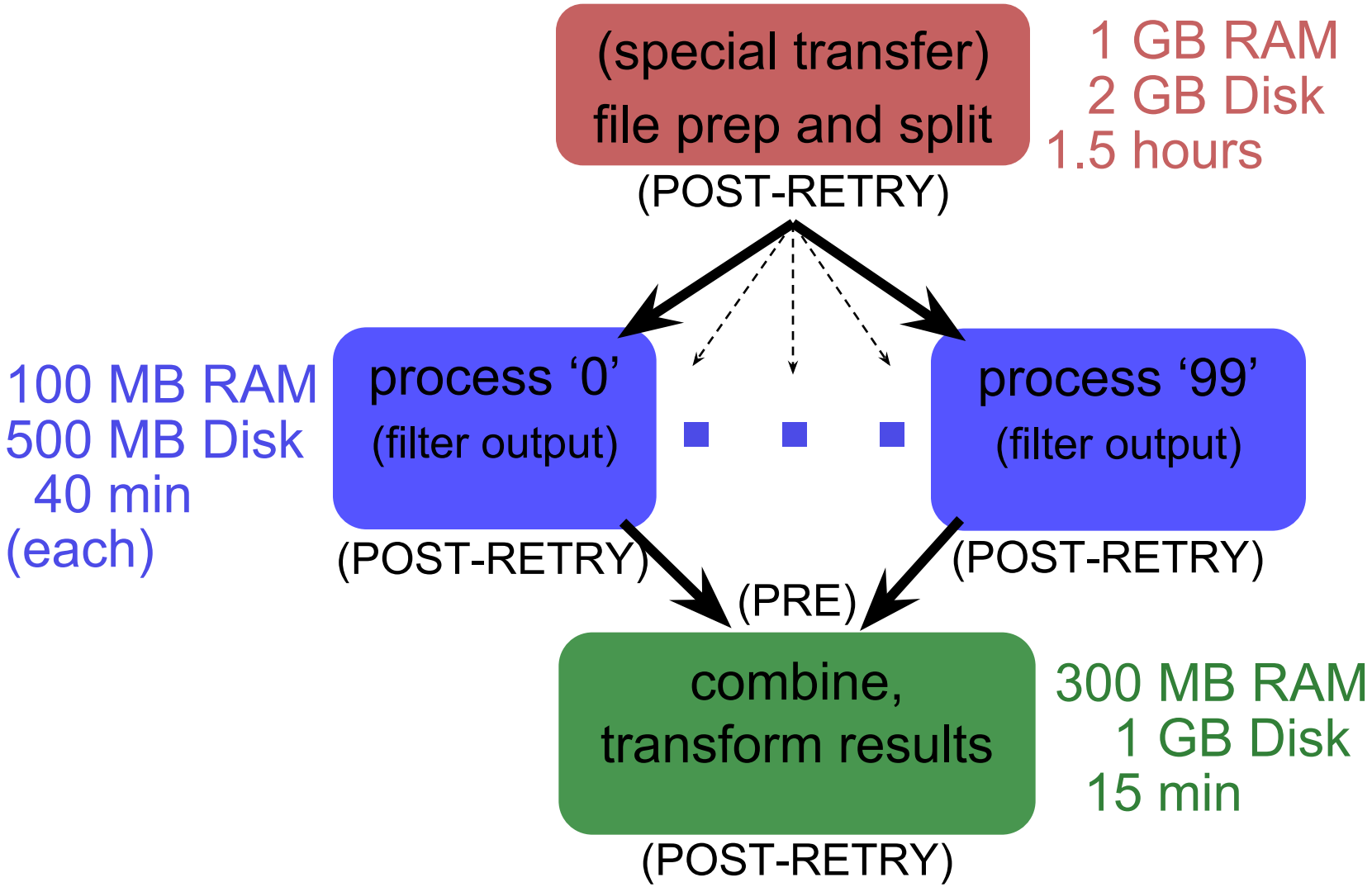


Building a Good Workflow

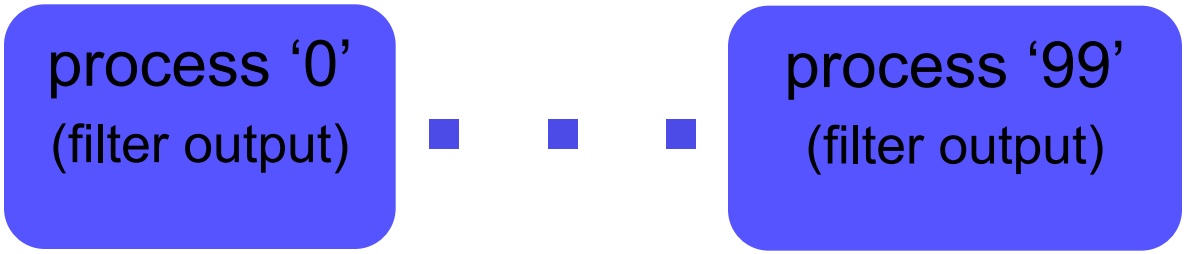
1. Draw out the *general* workflow
2. Define details (test ‘pieces’ with HTCondor jobs)
   - divide or consolidate ‘pieces’
   - determine resource requirements
   - identify steps to be automated or checked
3. **Build it modularly; test and optimize**
4. Scale-up gradually
5. Make it work consistently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

(And remember to document!)

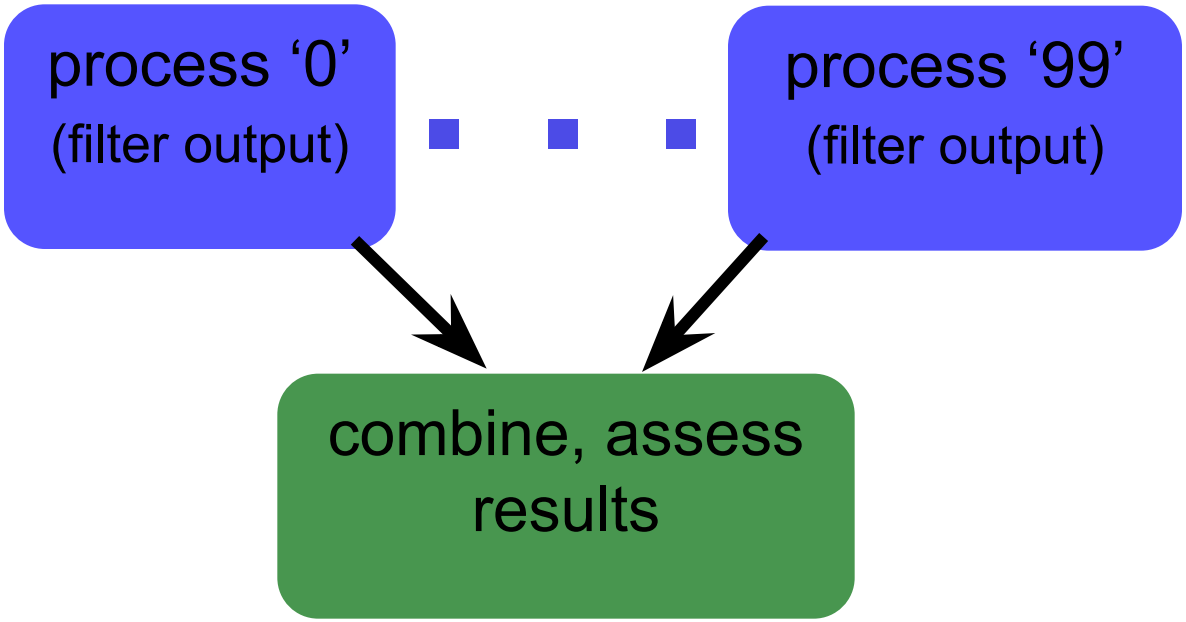
To Get Here …


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

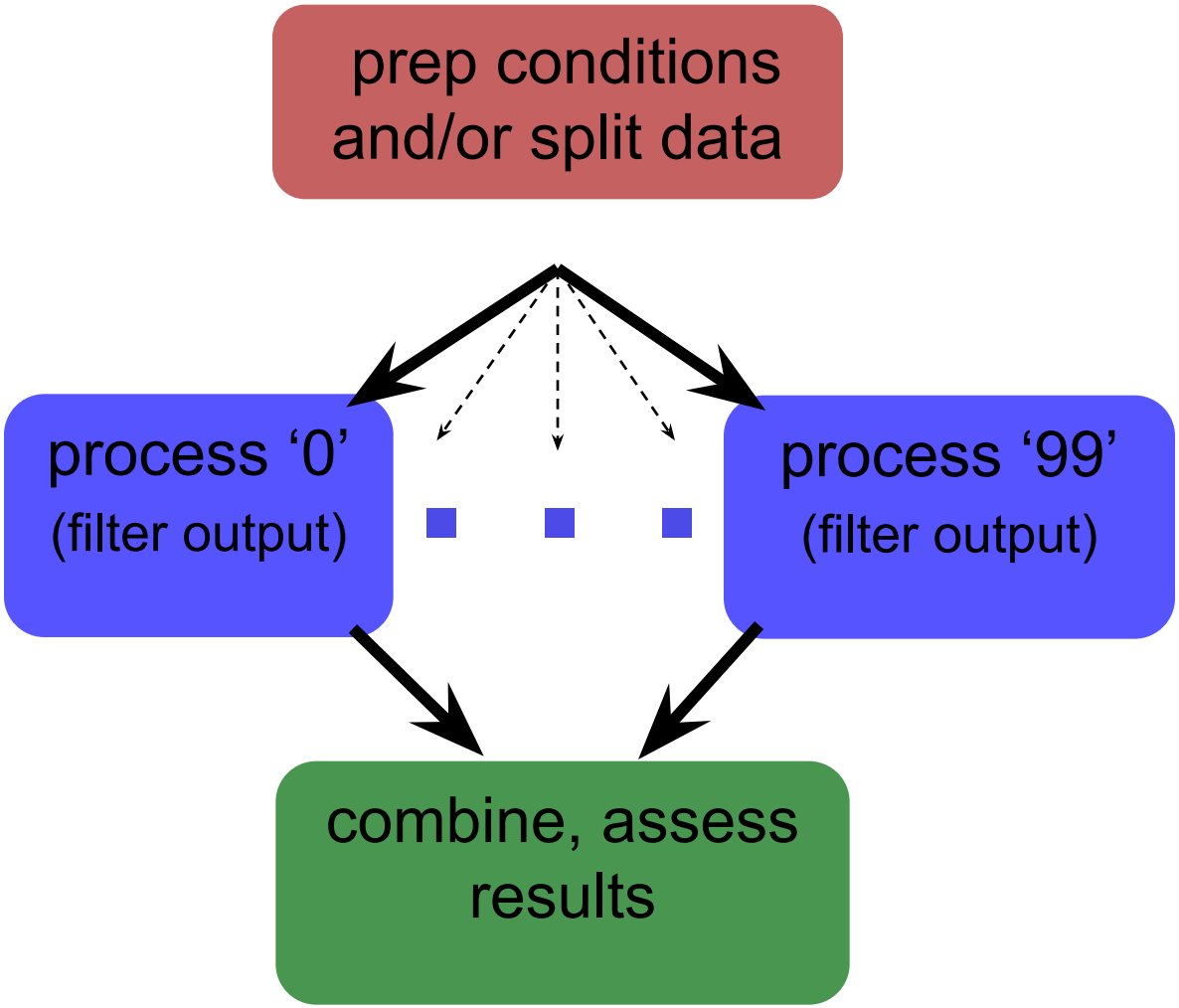
Start Here



Add a Step

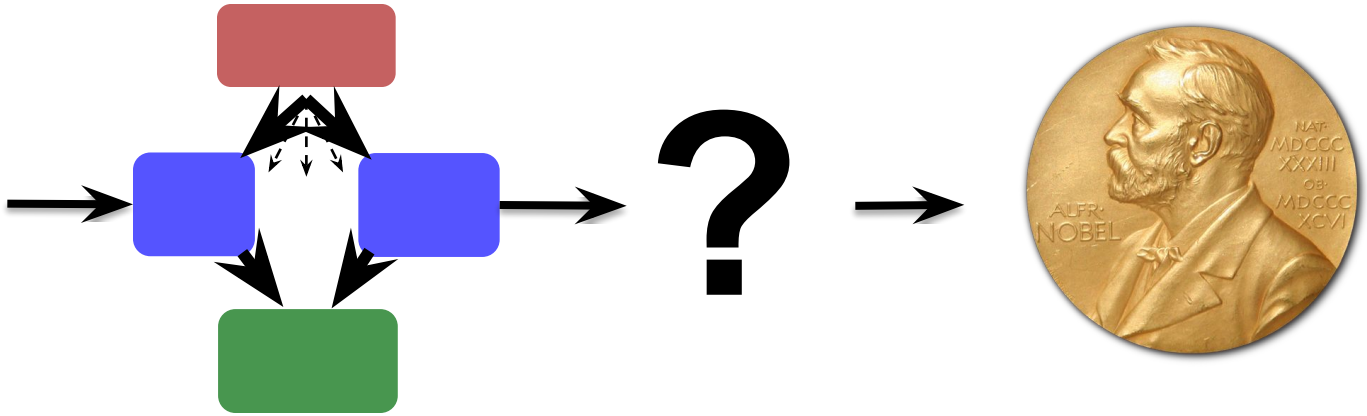

process ‘0’
(filter output)
process ‘99’
(filter output)
combine, assess
results

And Another Step



End Up With This?

DAT
A



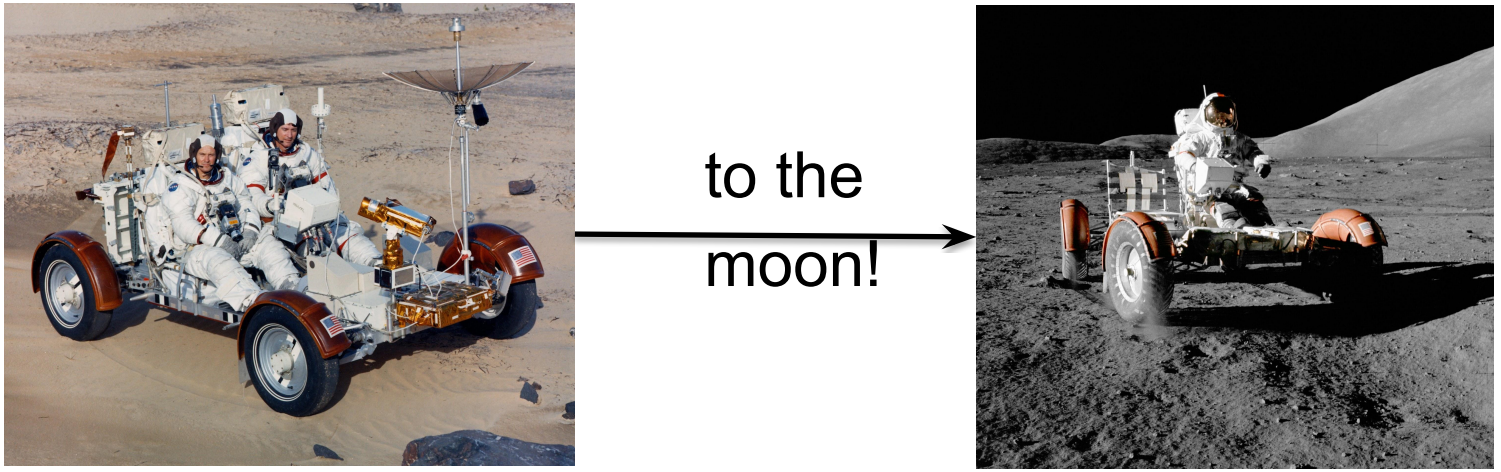
Building a Good Workflow

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 Workflows

- Your (“small”) DAG runs! Now what?
  - Need to make it run *full scale*



Scaling Up: Rules of Thumb

- CPU (single-threaded)
  - Best jobs run between **10 min** and **10 hrs**
    (Upper limit somewhat soft)
- Data (disk and network)
  - What is the balloon factor of your workflow?
  - Know when you are moving data and how to minimise it
- Memory
  - How much RAM / core do you have?

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:
  - 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!

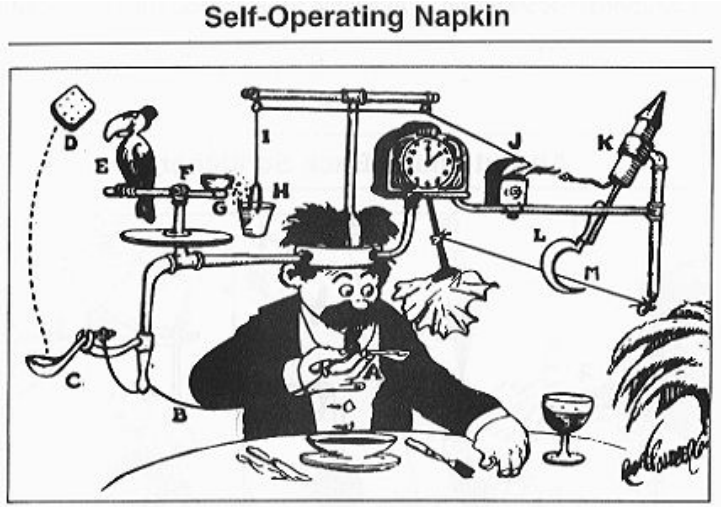
Building a Good Workflow

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!)

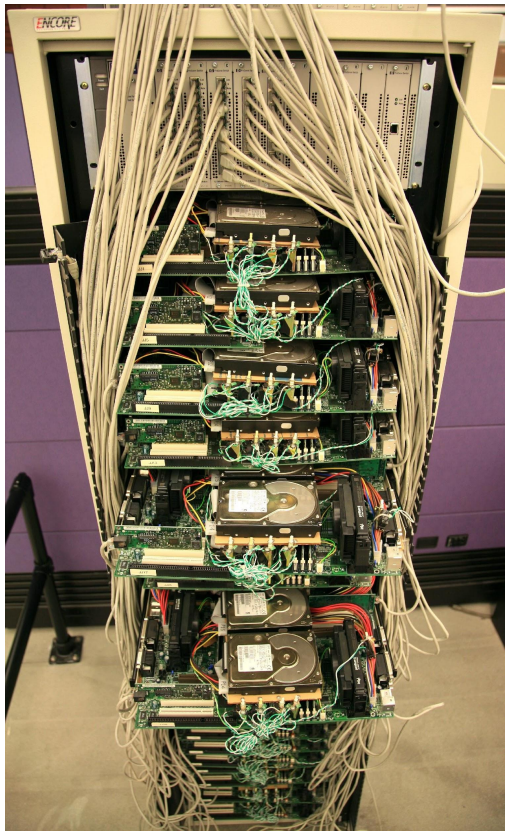
Robust Workflows

- Your workflow runs at scale! Now what?
  - 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 Everywhere

- What does the computing environment have?
  - Prepare for very little
- Bring as much as possible with you, including:
  - the executable you need
  - the software it depends on
  - data dependencies??



The Spectrum

- Laptop (1 machine)
  - You control everything!
- Local cluster (<100 - 1000 cores)
  - You can ask an admin nicely
- Cloud (core you pay for)
  - Be prepared to build your own cluster

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

Make It Run(-able) for Someone Else

- Automation is a step towards making your research reproducible by someone else
  - Work hard to make this happen.
  - It's *their* throughput, too.
- Can benefit those who want to do similar work

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?**

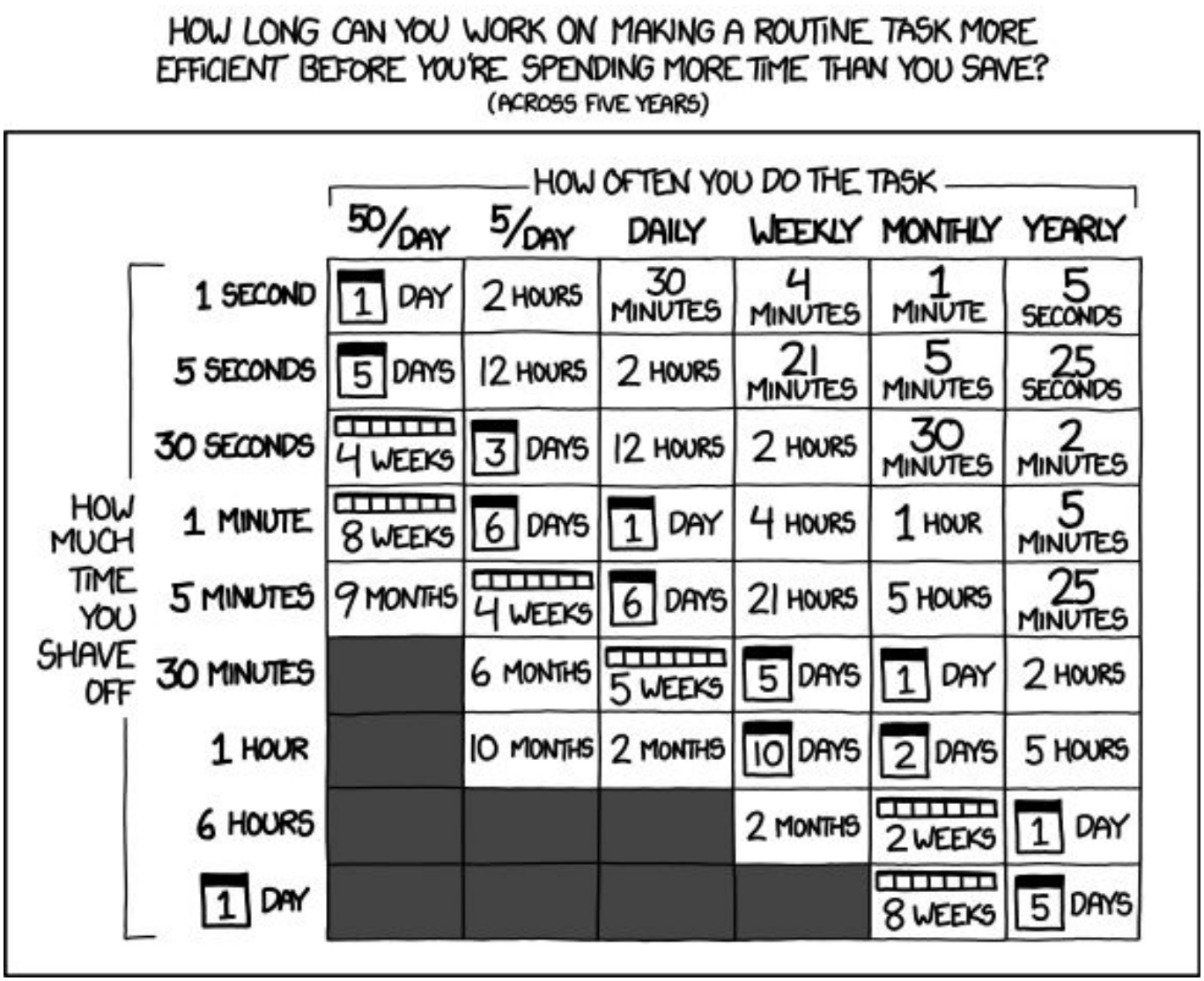
(And remember to document!)

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



http://xkcd.com/1205/

Make It Work Unattended

- Remember the ultimate goal:

  Automation! Time savings!

- Potential things to automate:
  - Data collection
  - Data preparation and staging
  - Submission
  - Analysis and verification
  - Workflow testing

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

PARTING THOUGHTS

Getting Research Done

- End goal: getting the research done
- Hopefully you now have the tools to get the most out of:
  - **Computing**: which approach and set of resources suit your problem?
  - **High Throughput computing**: optimize throughput, use portable data and software
  - **Workflows**: test, automate and scale