Big Computing Challenges
Part 1: Getting Going

"The Angry Penguin", used under creative commons licence from Swantje Hess and Jannis Pohlmann.

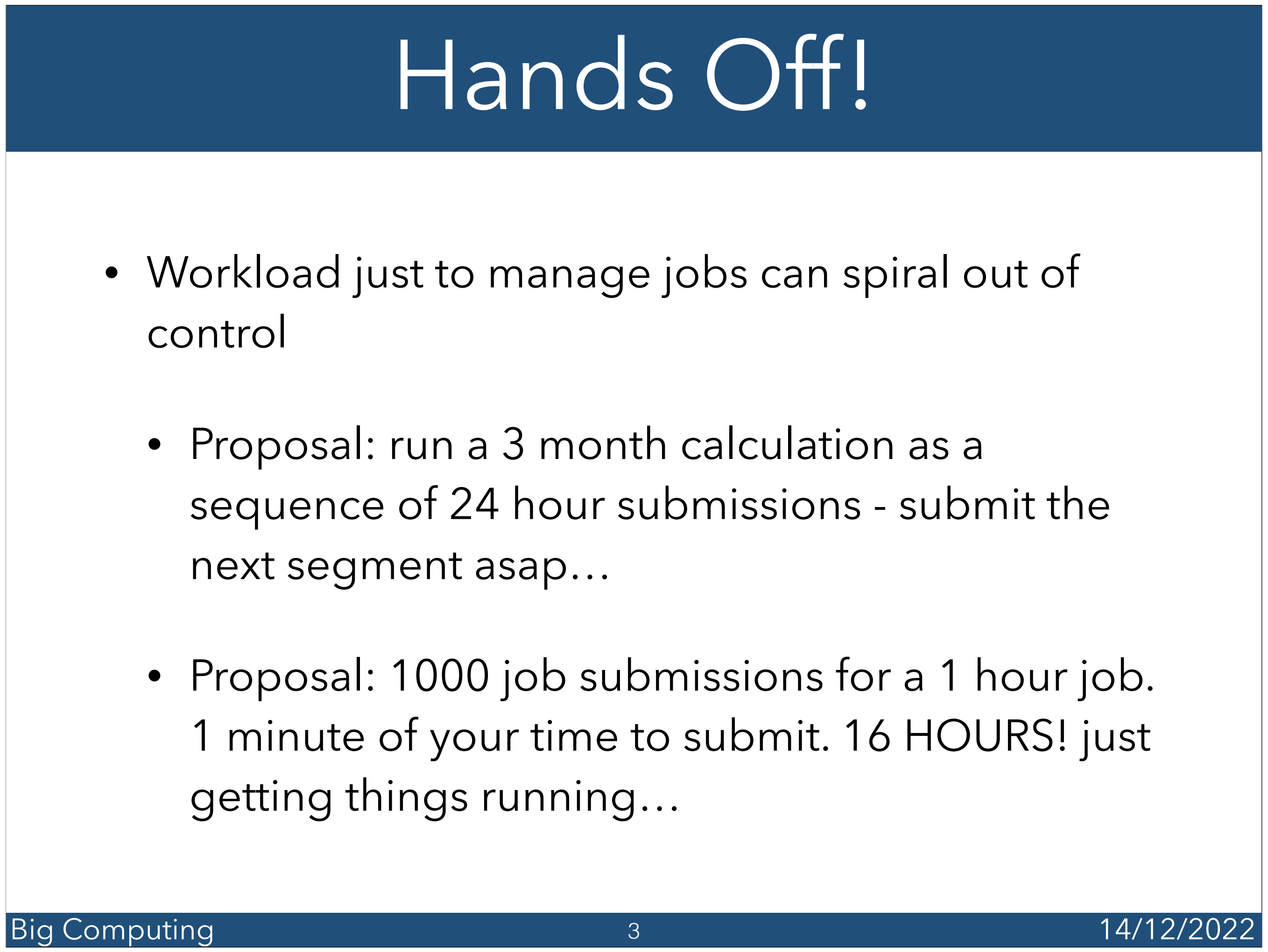
Warwick RSE

December 2022

Hands Off!

- Computing at scale works best if you mostly keep your hands off it
  - DO NOT plan to micromanage
- You do not control when things happen anymore!
  - Resources become available at all sorts of times
  - Most queues are not “FIFO”

FIFO - First In First Out. Like queuing in person, the first to arrive is served first. Compare this to making a pile of things and both add and taking them from the top (LIFO Last In First Out)



Hands Off!

- Workload just to manage jobs can spiral out of control
  - Proposal: run a 3 month calculation as a sequence of 24 hour submissions - submit the next segment asap…
  - Proposal: 1000 job submissions for a 1 hour job. 1 minute of your time to submit. 16 HOURS! just getting things running…

Your job becomes a very needy baby! Do you really want to drop everything just to resubmit? Or wake up at 2am in case something needs doing?

Nothing new, Horatio!

- Any task you find yourself with has probably come up before
    - Somebody may have a very good solution!
    - Ask, research etc
- Let systems work for you
    - If the order of jobs doesn't matter, don't enforce it
    - Don't force an unneeded timescale

I am combining two quotes here - there might be nothing new under the sun but there are more things in heaven and earth Horatio
Quotes aside, it is easy to jump into trying to solve your problems directly, but it's always worth a quick look about to check you aren't re-inventing the wheel.

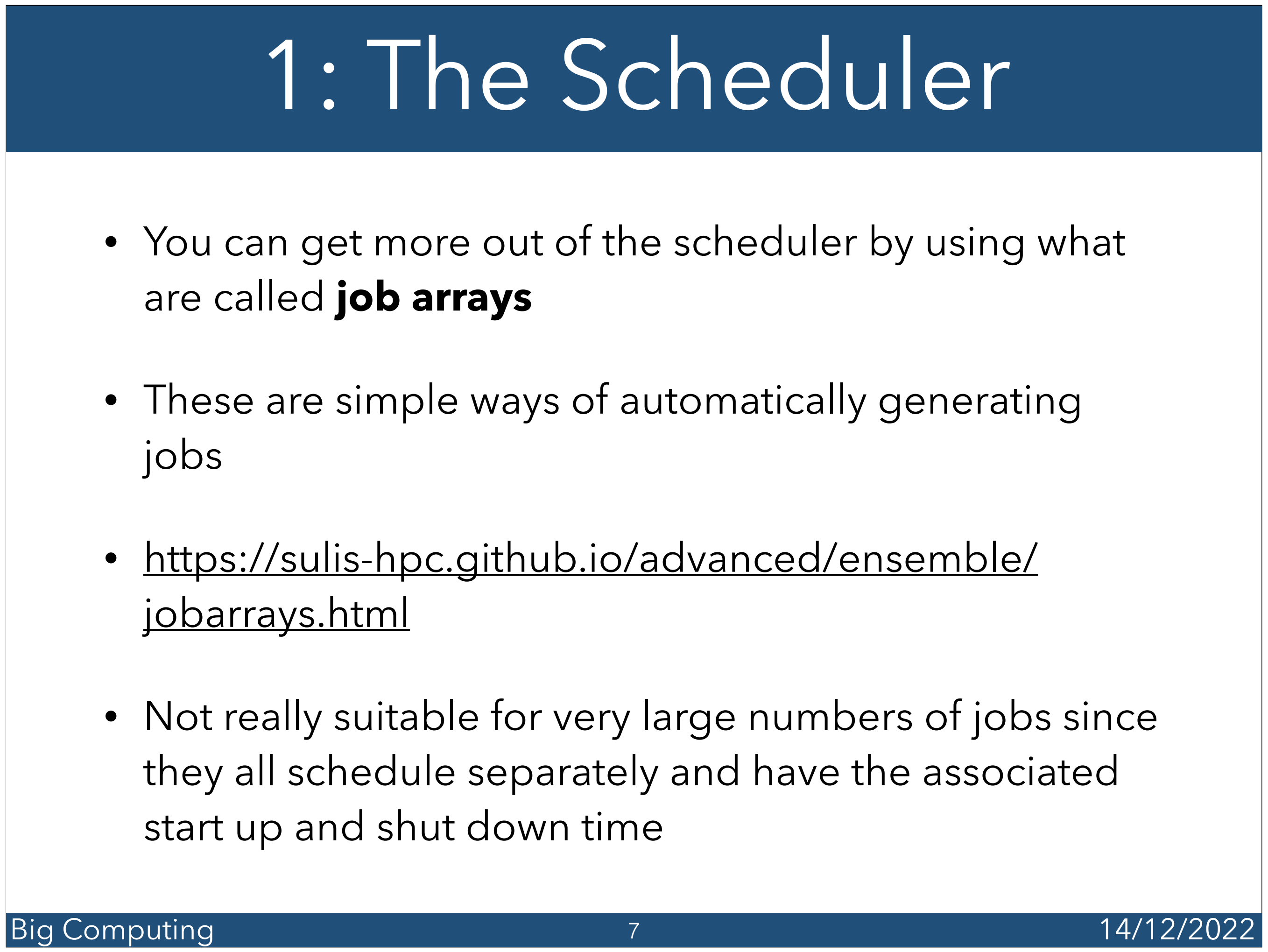
1: The Scheduler

- Scheduling systems can have a lot of power
  - How much depends on local configuration
- In general, they can:
  - Make one job contingent on another, or on certain conditions
  - Run many jobs from a single submission
- And they absolutely WILL limit your running job/total simultaneous usage according to quota!

1: The Scheduler

- Exploit the scheduler where you can!
- There are usually limits on the number of jobs that you can have running on a given system at a time (i.e. 200 for Sulis)
- There are also limits on the number of jobs that you can have queued (500 for Sulis)
- Those numbers shouldn't be very restrictive for most people so you may as well submit all your work as soon as it is ready and just let the scheduler do its thing

As always, check the docs for details of these quotas. If you exceed the running limit, your jobs will sit in the queue even if resources are available. If you exceed the queued limit, attempts to submit new jobs will fail



1: The Scheduler

- You can get more out of the scheduler by using what are called **job arrays**
- These are simple ways of automatically generating jobs
- https://sulis-hpc.github.io/advanced/ensemble/jobarrays.html
- Not really suitable for very large numbers of jobs since they all schedule separately and have the associated start up and shut down time

Job arrays make it easier for you, but don’t really reduce load on the scheduler much

2: Workflow managers

- What the scheduler can't do, other systems can
  - Job contingencies
    - IF X gave *this* answer, run Y, otherwise Z
    - Once *these* jobs have run, start *those*
  - Very large numbers of very small jobs
    - Generally the overhead of starting many small jobs would be unmanageable anyway

2: Workflow managers

- Some suggestions:
- GNU Parallel - Works for unconnected tasks well, can be bodged for contingent tasks (reading files from disk etc.)
- Discussed this on Monday
  - Great tutorial link in those slides

2: Workflow managers

- Some suggestions:
- Dask - Python based library with various methods for both manual and automatic parallelisation of workflows
  - https://sulis-hpc.github.io/advanced/ensemble/dask.html
  - https://tutorial.dask.org/00_overview.html

2: Workflow managers

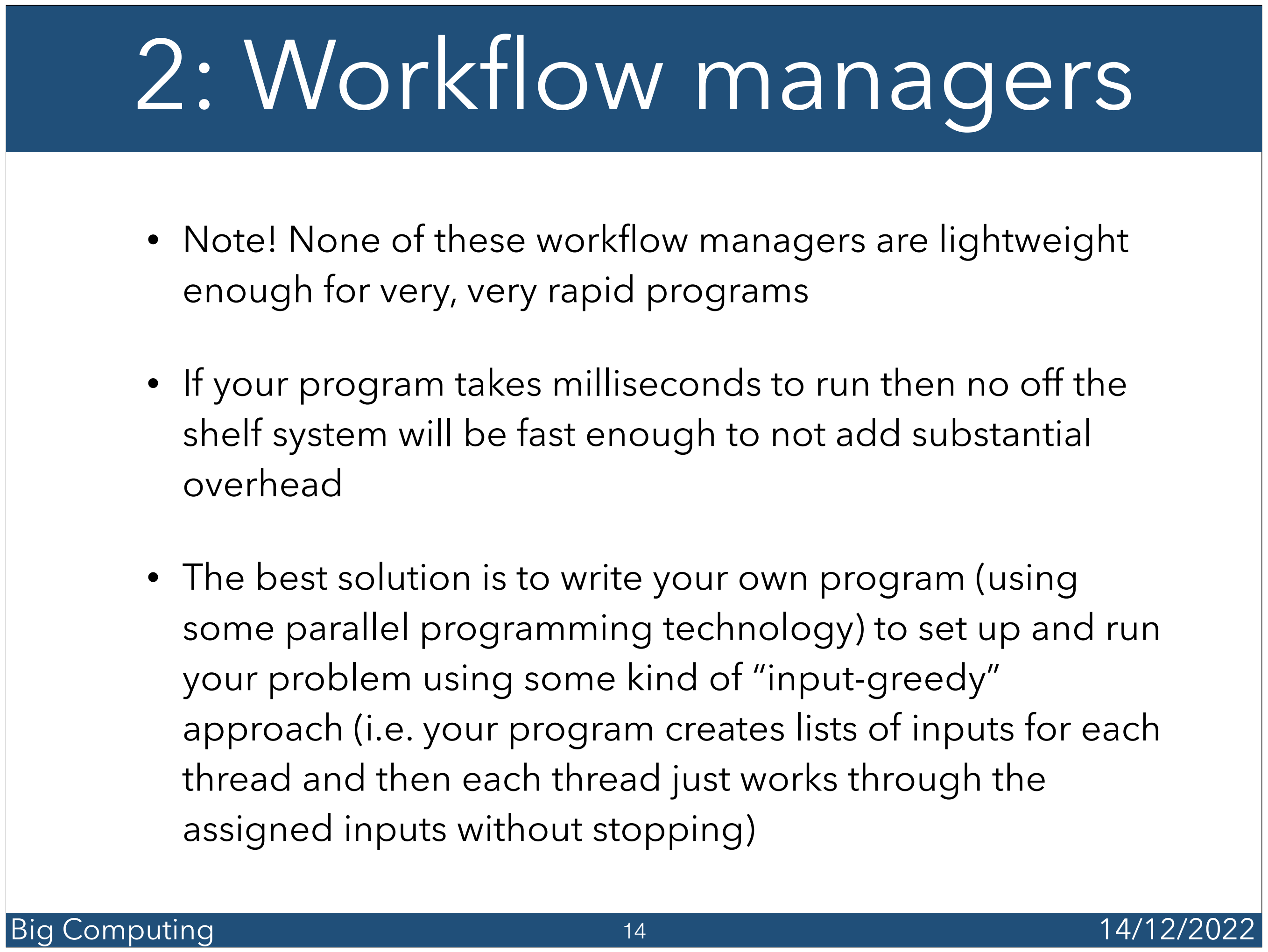
- Dask is worth a bit more detail
- There are a **lot** of ways to use it but the virtual cluster approach is pretty common
- dask.delayed - decorator for a function that produces an asynchronous version of that function. Write your normal workflow wrapping all of your functions in dask.delayed and dask will run as many bits as possible in parallel

2: Workflow managers

- dask.distributed - This allows you to create a virtual cluster within your resources on the actual cluster. You control how work is scheduled and when it runs so you can dispatch work as wanted - only bit of dask that allows you to dispatch work across multiple nodes in a single job
- dask.futures - We briefly mentioned futures before and this is dask's implementation of them
- Parallelised versions of other common python libraries (i.e data.dataframe = parallel pandas.dataframe)

2: Workflow managers

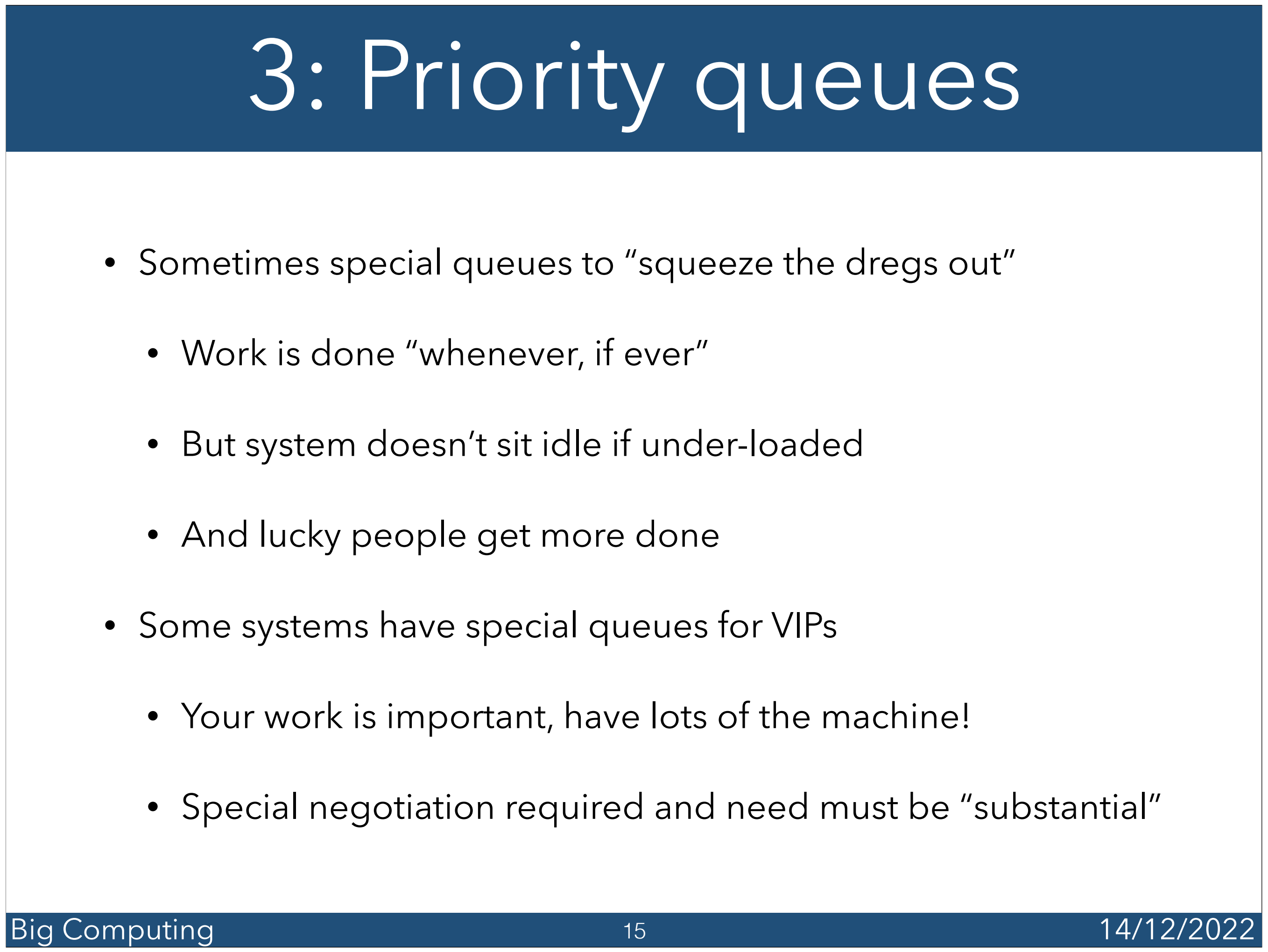
- Snakemake - Workflow manager originally designed for bioinformatics but with general applicability to highly contingent problems
- Celery - Workflow manager originally intended for handling “semi-offline” tasks in web server environments. Used in some research fields but not a brilliant map to cluster systems
- Many others but don’t get too attached to any particular system since they often don’t tend to be maintained for long



2: Workflow managers

- Note! None of these workflow managers are lightweight enough for very, very rapid programs
- If your program takes milliseconds to run then no off the shelf system will be fast enough to not add substantial overhead
- The best solution is to write your own program (using some parallel programming technology) to set up and run your problem using some kind of “input-greedy” approach (i.e. your program creates lists of inputs for each thread and then each thread just works through the assigned inputs without stopping)

We can discuss details if this is a problem you have!



3: Priority queues

- Sometimes special queues to “squeeze the dregs out”
  - Work is done “whenever, if ever”
  - But system doesn’t sit idle if under-loaded
  - And lucky people get more done
- Some systems have special queues for VIPs
  - Your work is important, have lots of the machine!
  - Special negotiation required and need must be “substantial”

Handy thing people don’t always know about. Don’t always exist
Pauper queues moderately common
Prince queues pretty unusual!
Pauper queues often have much stronger limits on how many jobs you can queue, so you can’t monopolise things.

Follow the Script

Notebooks

- Notebooks can be run on clusters
- Often not a great idea
- The biggest problem is that your resources can become available at any time
  - Now
  - In an hour
  - Middle of the night next Wednesday
- Interactive use of clusters is difficult and really only suitable for small jobs that schedule immediately
- Reproducibility also suggests that you shouldn't be manually tweaking things for large problems

Notebooks

- Turning them into scripts often better
  - More on how in a moment
  - NOTE: Just because the conversion worked, it doesn't always mean you have a working script!
  - You have to check and test!
  - Try and test (and handle) improper inputs as well as expected ones
    - Can save a lot of headaches later!

Notebooks

- If you have notebooks you want to convert
  - Export them from running notebook server
  - Offline conversion tool from jupyter
  - We have some materials for some of this in our course here https://warwick.ac.uk/research/rtp/sc/rse/training/hpcdatasci/

Notebooks

- You might want to program a bit differently for a script than a notebook
  - Need to preserve inputs for reproducibility (see later)
  - Prefer not to sprinkle user-interaction throughout a script
  - Images etc become input files not embeds

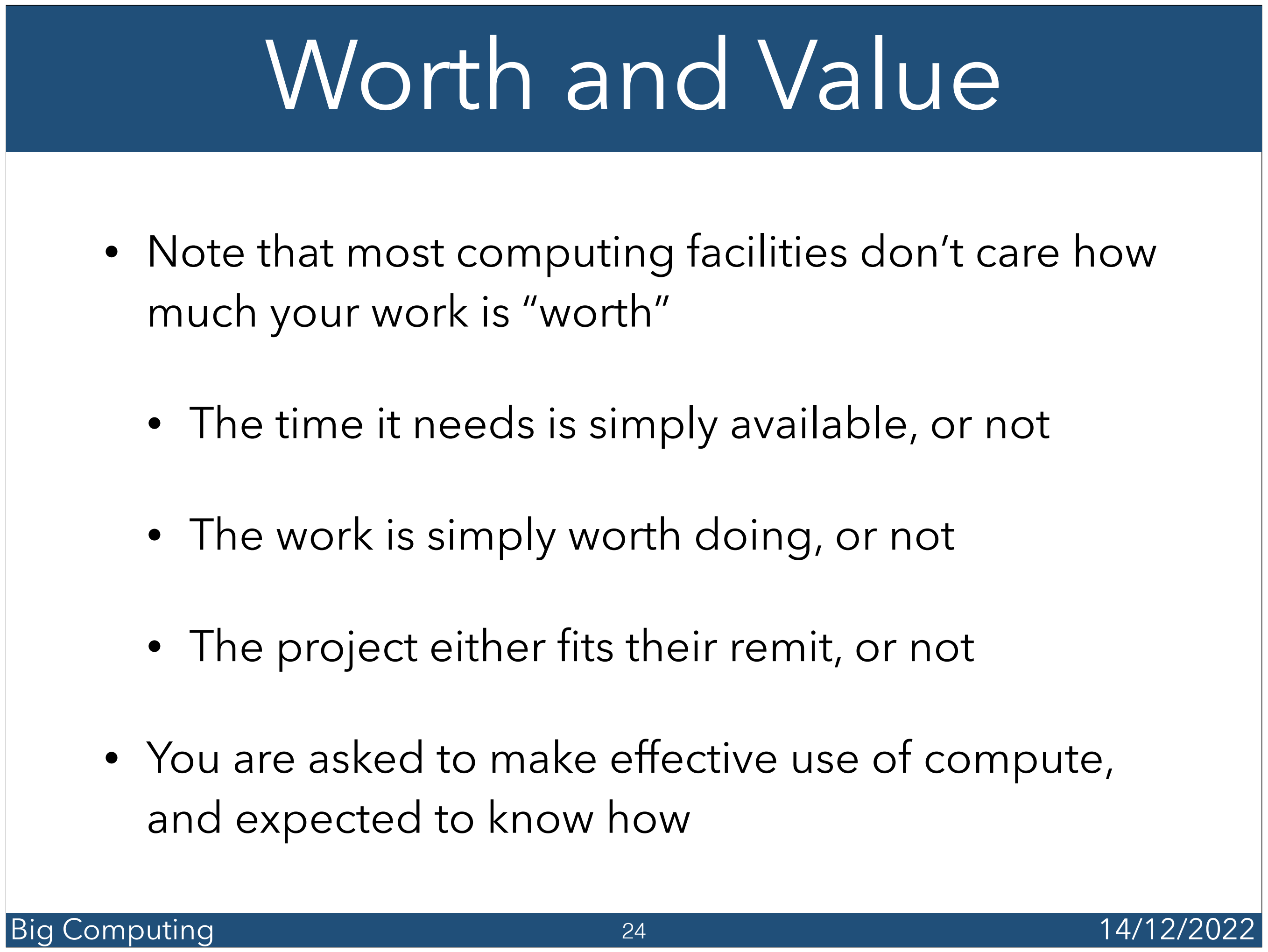
Notebooks

- Making this jump can really help the reproducibility of your research
    - See later for more on this
- For now, think about whether you are sure your notebook will give the same answer every time you use it

Horses for Courses

Name that Tune

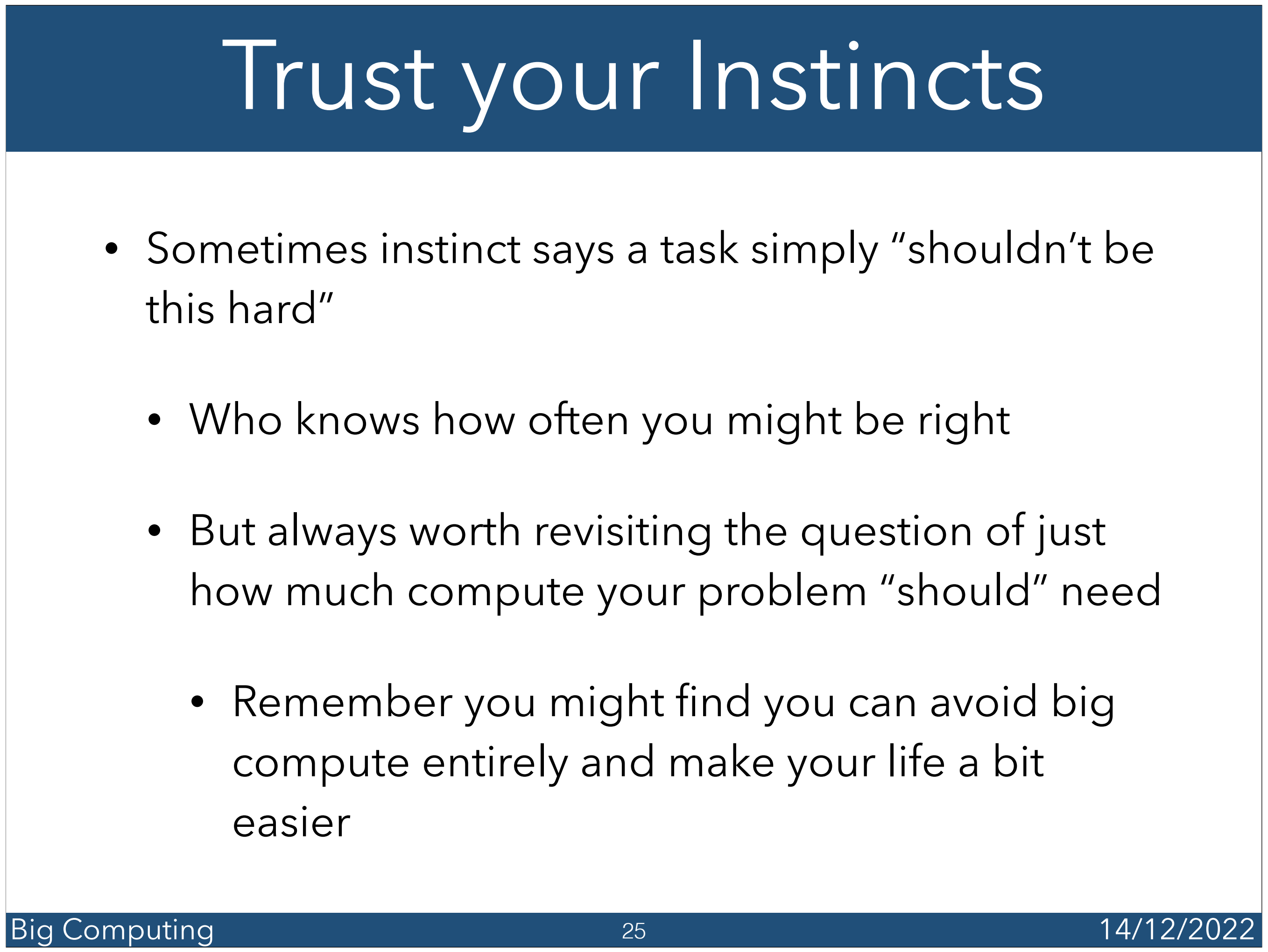
- To make good use of Big Computing you have to match what you want to what you can get
    - We can sort letters using 10 workers or 1000 workers, but too few takes time and too many get in each others way
- Sometimes (often?) many options are appropriate and its up to you to fit your work to your options



Worth and Value

- Note that most computing facilities don't care how much your work is "worth"
    - The time it needs is simply available, or not
    - The work is simply worth doing, or not
    - The project either fits their remit, or not
- You are asked to make effective use of compute, and expected to know how

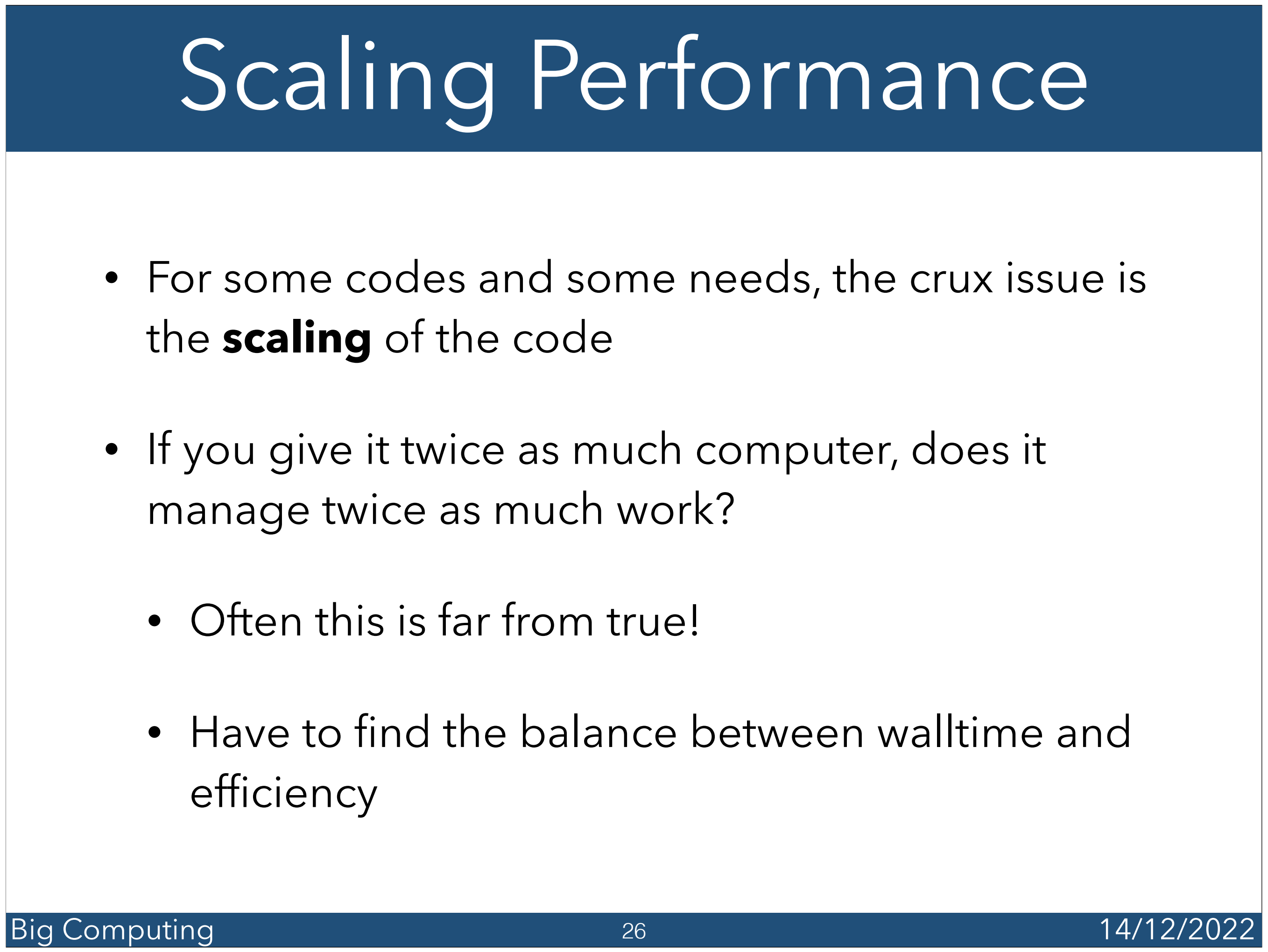
We've been on allocation committees for national compute, and this is often their perspective. If work has been funded by the research council, and if the approach and code are appropriate, they try to allocation the resources required. This may not be possible and quality may be used to scale allocations, but it is not about picking the most "papers per resource hour" stuff!



Trust your Instincts

- Sometimes instinct says a task simply "shouldn't be this hard"
  - Who knows how often you might be right
  - But always worth revisiting the question of just how much compute your problem "should" need
    - Remember you might find you can avoid big compute entirely and make your life a bit easier

In a lot of cases you can directly estimate just how much compute load there is by estimating how many operations have to occur, on how much data. This is the same stuff as so-called "algorithmic efficiency" or "Big-O" which aims to work out how operations get harder as the size of their input data increases. For instance, finding something in an un-ordered list takes a time which goes up like the size of the list. If it takes 5 minutes for 5000 items, it will take an average of 10 minutes for 10,000 etc. You can estimate the absolute minimum time something should take by estimating (generously) how many simple operations (adds, multiplies etc) are involved, and hopefully your solutions are within a few factors of ten of this. If not, either your estimate is off, your code is doing a lot more than you thought, or there is some inefficiency to be fixed.
In reality, this sort of estimation is more an art than a science due to the huge amount of factors that can be involved, but it can be worth a try. In particular, you might find an estimate proves that a lot of compute is needed, in which case you either seek a cleverer solution, or commit to Big Computing loads.



Scaling Performance

- For some codes and some needs, the crux issue is the **scaling** of the code
- If you give it twice as much computer, does it manage twice as much work?
  - Often this is far from true!
  - Have to find the balance between walltime and efficiency

By balance, we mean that if you double the amount of computer, and get say 1.5x as much work, you will have to decide if the 0.5x shortfall is worth it to get your answer that 1.5x faster.

Batching Jobs

- Sometimes a task is serial, or limited to very few threads
- Or a task takes very little time but you need 1000s of them
- Too many job submissions is bad
  - Schedulers not usually designed for volume
  - Overheads of waiting and starting each can be punishing
- High throughput computing needs more than Slurm
  - Use one of those options we discussed earlier

The In-between

- It is not uncommon for things to fall into the space between large and small and hit both previous concerns
    - Benefit strongly from multi-threading
    - But too small to use a whole computer
- Sulis we have discussed has 128 cores in each node
    - 128 = 8 x 16 = 4 x 32 = 2 x 64
    - Many ways to “fully populate” your node!

Optimising this is tricksy, but doing it is nice and easy. Either use the scheduler or one of those options from earlier. See the next section for tips on how to do the optimising

Help!

Do you know everything?

- NOBODY expects you to know everything!
- Some systems won't run jobs from two people on one node
    - Use half a node, the rest sits idle (and might count against your allocation regardless)
    - Run 2 x 64 core jobs, will they be packed together?
- WHO KNOWS?!
    - Sysadmins do!

Do you know everything?

- Some schedulers can do arrays of jobs natively
- Other systems prefer you to wrap or combine tasks
- WHO KNOWS?!
  - Sysadmins do!
  - Support staff do!

Whose Problem?

- Sysadmins and computer support staff know what their machine can do
    - Know what it “likes”
- They do not know your research
    - Only you know that
- They do not (usually) know your code
    - Code author, documentation, other users know that

Whose Problem?

- In order to make good use of system you have to WORK WITH them
  - Some bad support requests:
    - My code has gone wrong. Please help!
    - I ran a job on 8 cores and it took too long. Please help!
    - How many nodes should I use to do my work?

Whose Problem?

- Some good support requests:
  - My code says it scales to 64 cores but I am not sure how well it will perform. Can you help me check?
  - I want to run sets of 8 core jobs, what is the best way to do this?
  - I got this error message [*Message details*] when I [*details of what you did*]. What should I do next to identify the problem?

We’re not saying the staff can always help you, but they won’t be upset for you asking!
Think about any job you’ve ever had, or any time anybody asks for help! What is a better question: what should I order for dinner? Or I like garlic but not spicy food, which dish would suit me best?

Whose Problem?

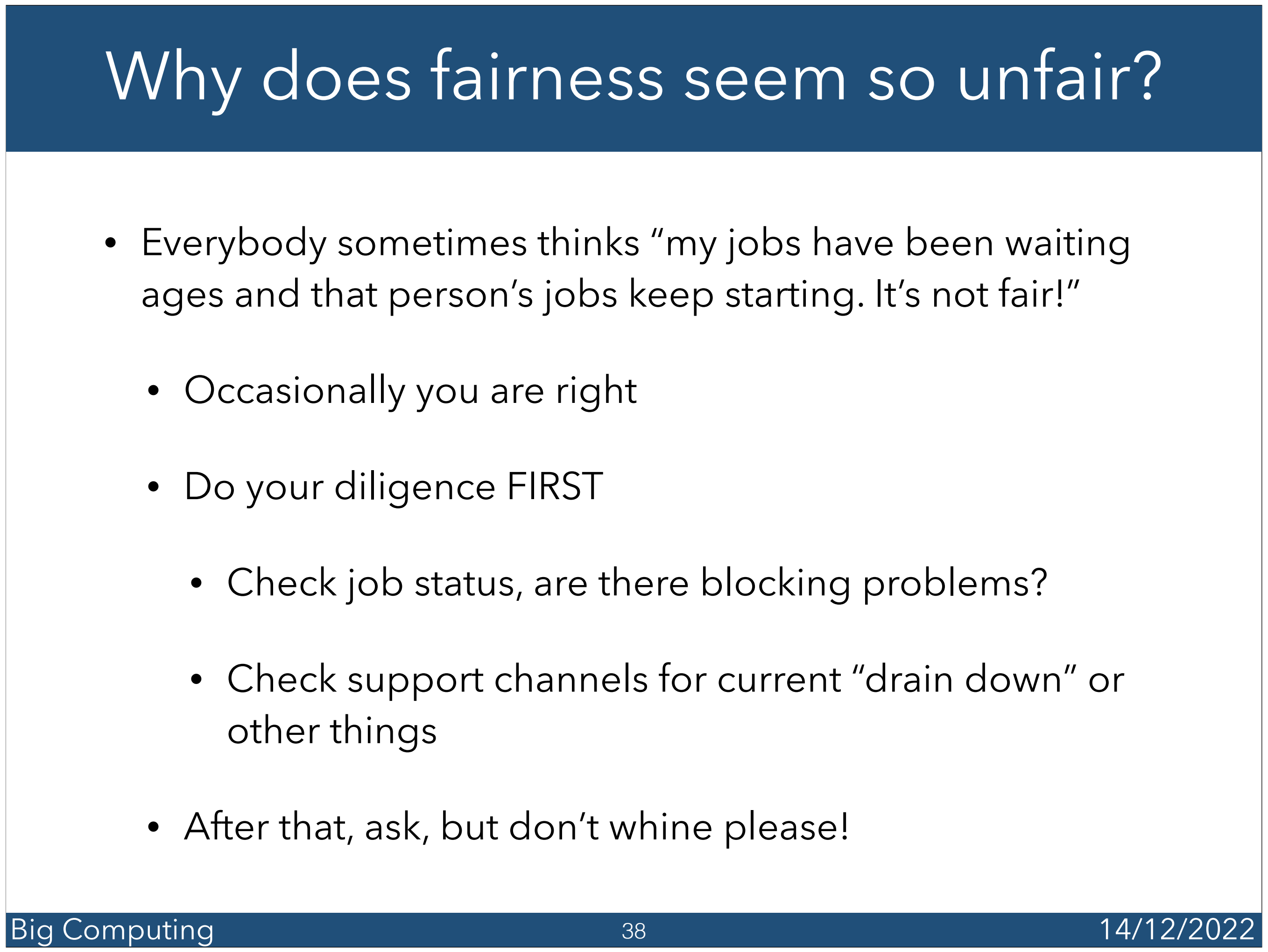
- The one support request you should never ever write:
  - I have known for months that there is an important conference on Monday but I left it until the last minute and despite my job being in the queue for an entire 5 minutes it is not running yet. What are you going to do about this terrible problem with my important research?

Yes, this is sarcastic, but you would be shocked how close people get. Your lack of planning is NOT anybody else's emergency! If you have a reputation as somebody who asks sensible questions and knows the score about "special favours" you are much more likely to get them.
By the way, this might sound a bit dodgy but all we mean by "favours" are things you get when they won't unduly impact others, but aren't documented operating procedure. For example if you have a large, well performing code, you might be asked to help "run in" a new machine and nobody minds if you happen to get some useful work done in the process.

But it isn’t FAIR!

Why does fairness seem so unfair?

- Most systems run using some kind of “fair sharing”
  - Maximum amount of running or queued stuff
  - Priority dependent on how much you have done recently (higher if less)
  - Priorities dependent on total allocation, department size, paid contributions etc
  - Other factors might also be considered



Why does fairness seem so unfair?

- Everybody sometimes thinks “my jobs have been waiting ages and that person’s jobs keep starting. It’s not fair!”
  - Occasionally you are right
  - Do your diligence FIRST
    - Check job status, are there blocking problems?
    - Check support channels for current “drain down” or other things
  - After that, ask, but don’t whine please!

e.g. you asked for more memory, the wrong queue, a resource that you can’t have etc
e.g. only jobs less than 1 hour long are being started right now

It’s still fairly likely it’s just “one of those things” so don’t dump on the poor admin! Just name the job, say its been waiting X long and you’re sure it’s not your end, and can they check it out. Ez.