Proposal for a parallel scheduling algorithm

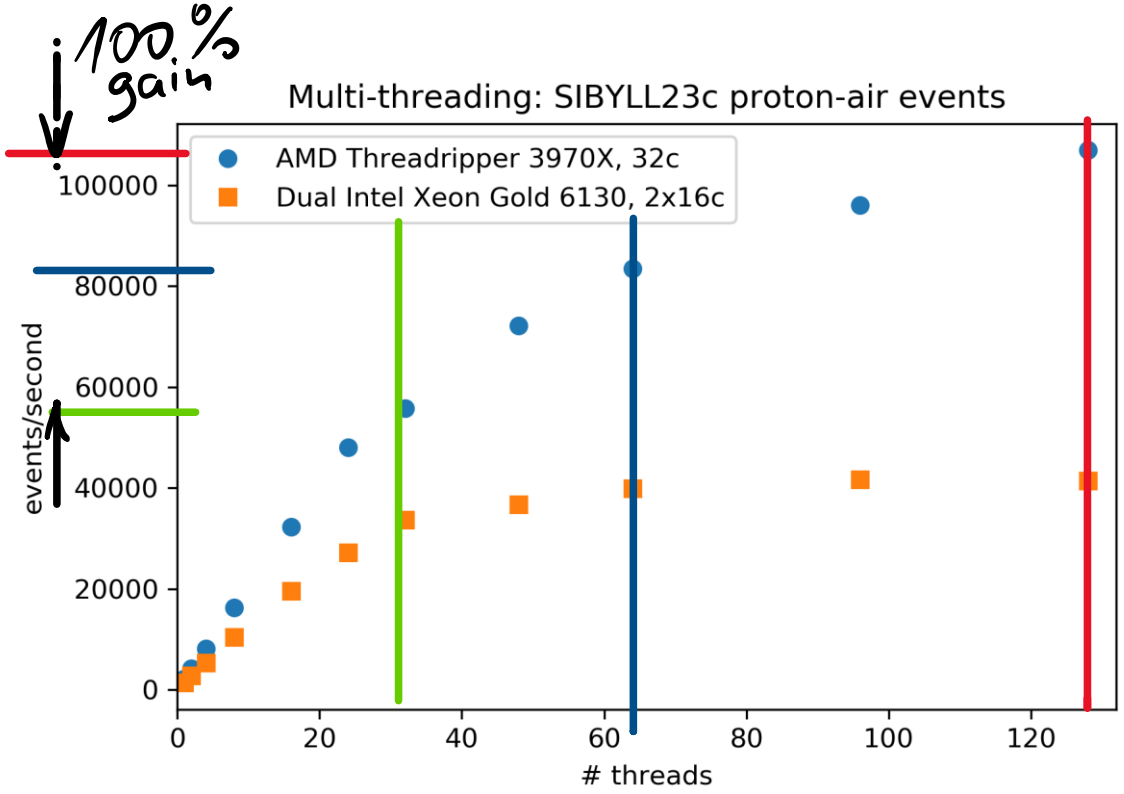
Anatoli Fedynitch
@ ICRR

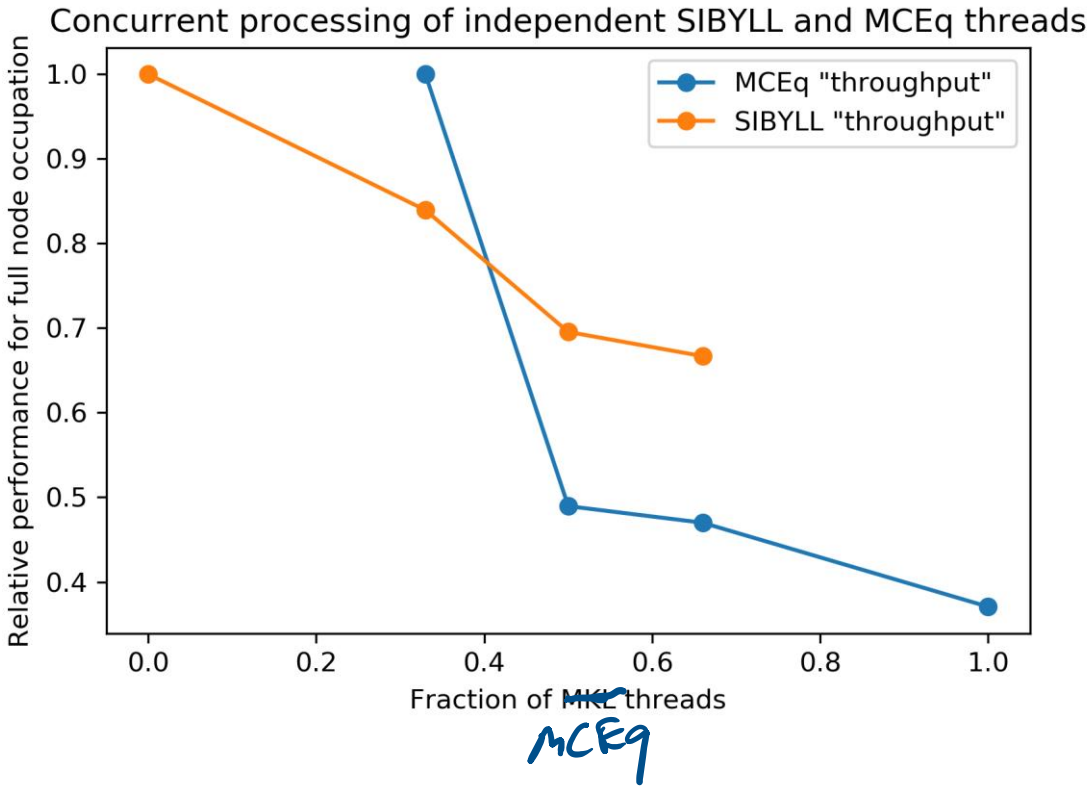
February 3rd 2020

ICRR
Institute for Cosmic Ray Research
University of Tokyo

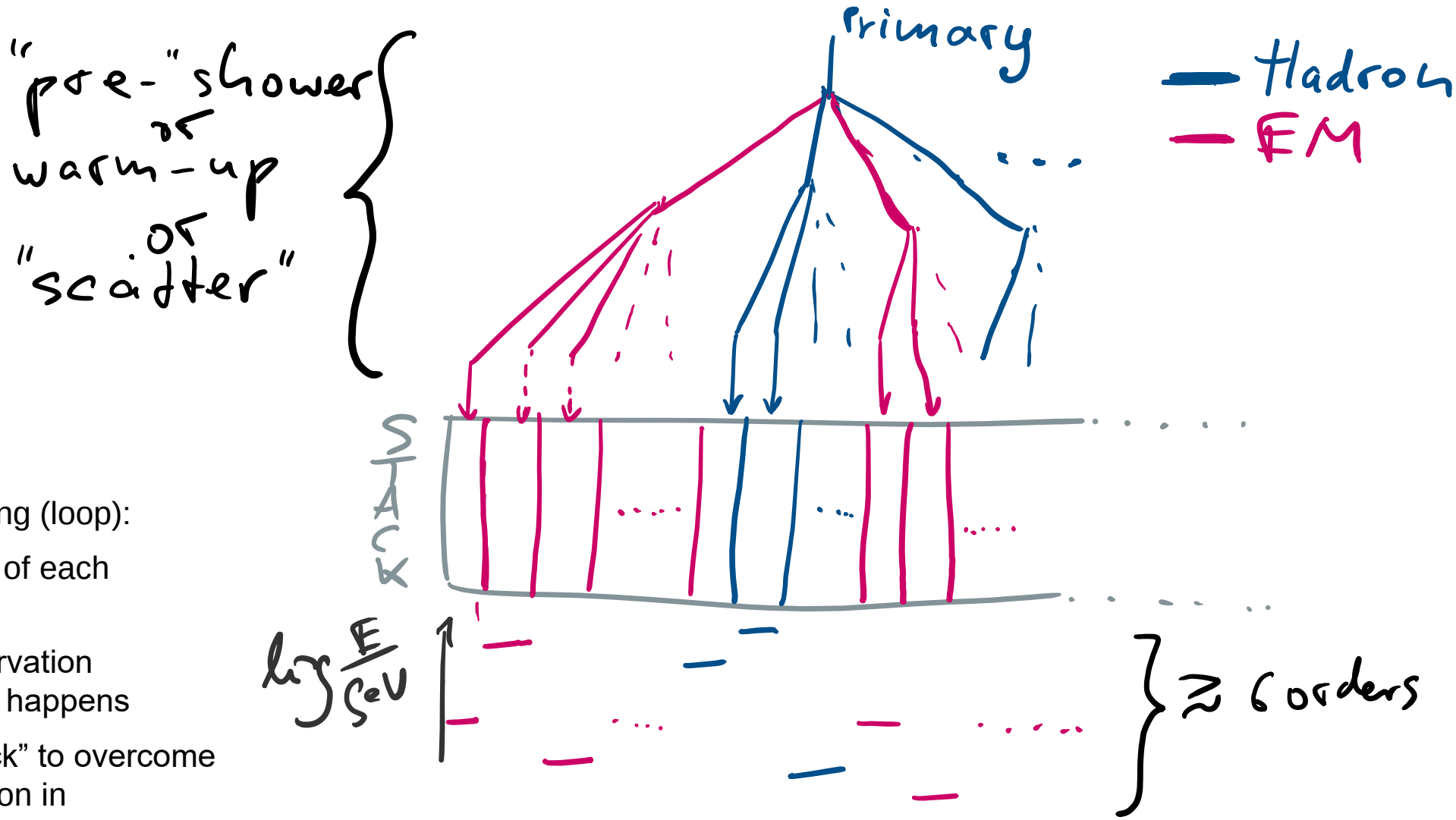
Recap from last talk

- **Scaling up CORSIKA simulations and use modern methods (heterogenic compute: multicore + accelerator) means that parallelization has to be managed**
  - This means that there is “no main loop” ☹
  - Instead, there has to be a scheduler.
  - More on this: next time



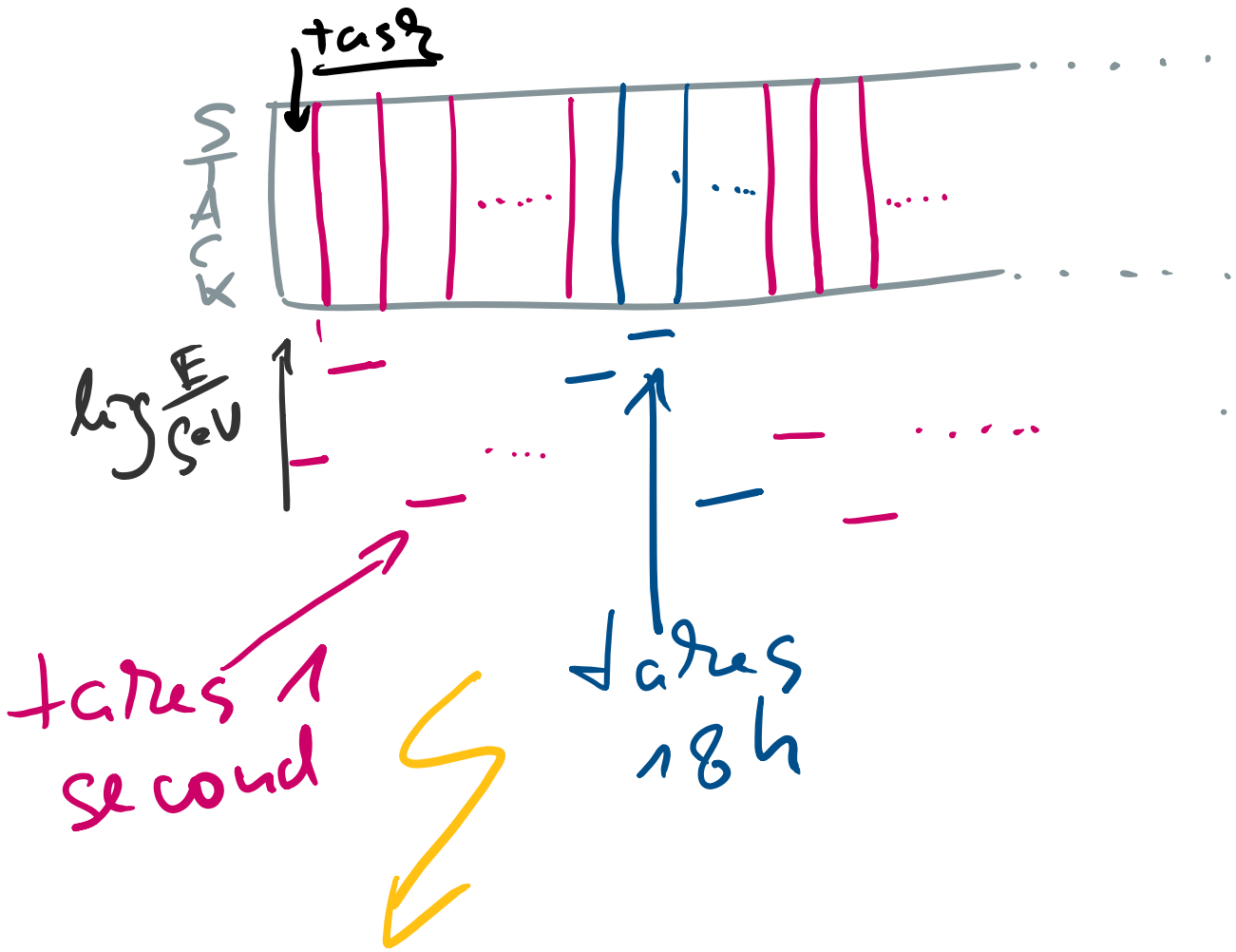


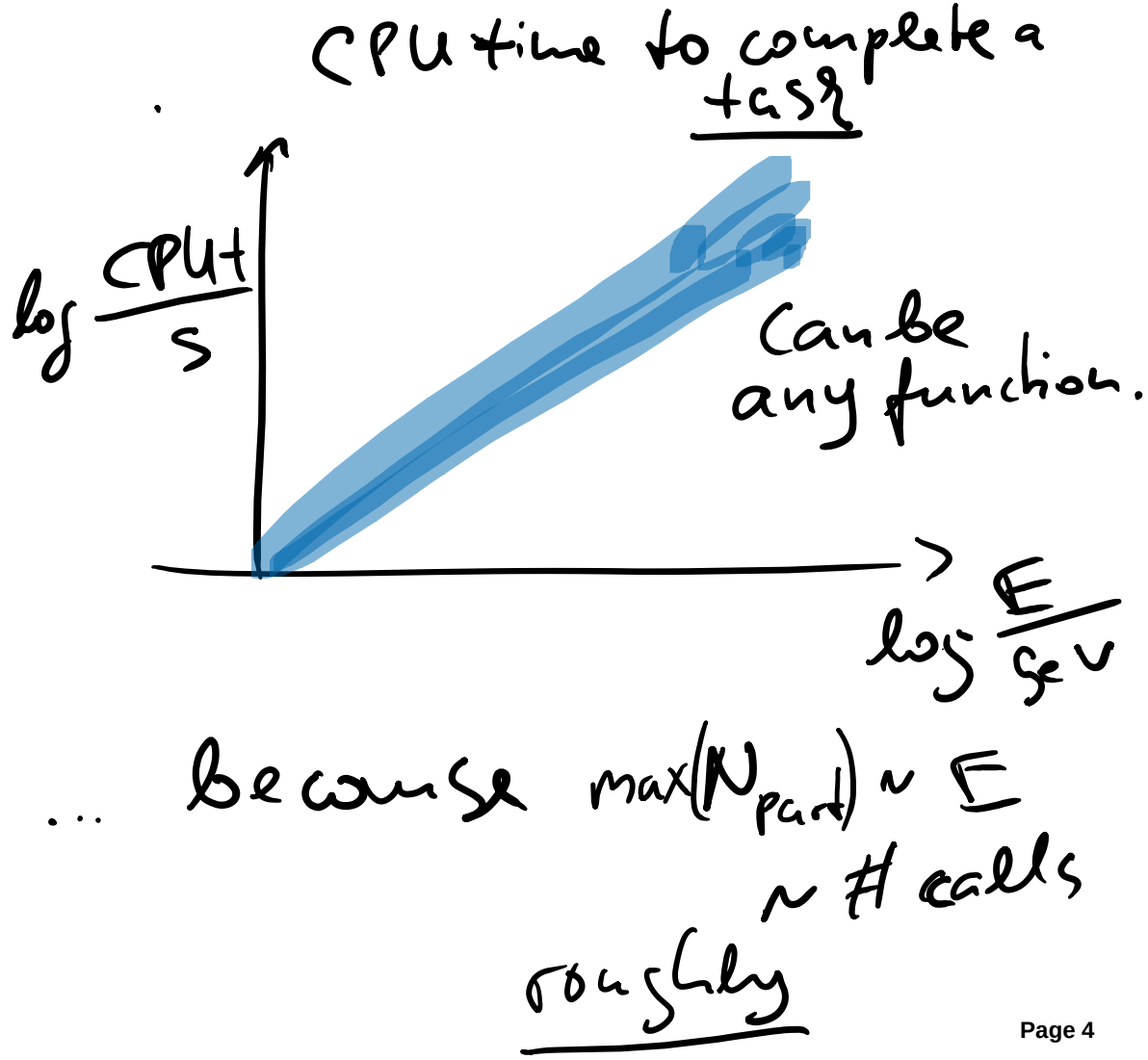
Shower "tree"



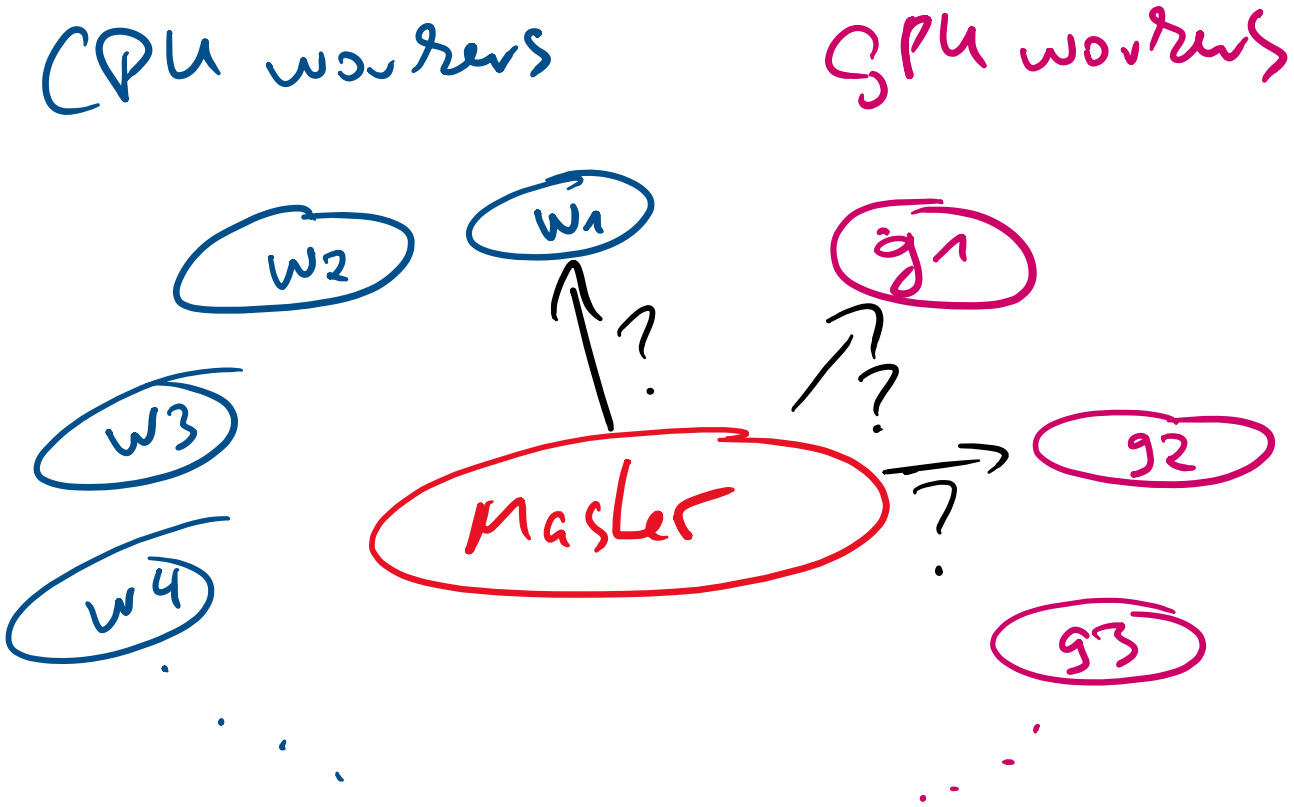
- Classical way of processing (loop):
  - Follow the first branch of each branching (interaction)
  - Terminate when “observation condition”/serialization happens
  - Use “intermediate stack” to overcome the absence of recursion in FORTRAN and save memory

CPU time requirements





Efficient distribution of work



- How to occupy all available recourses?
- Group tasks into batches of tasks (=jobs)
  - With controlled runtime
  - For the correct arch (GPU/CPU)

Scheduling algorithm

Steps:

1) Allocate workers CPU+GPU
2) Accumulate a STACK
3) Sort according to projected runtime (heuristic)
4) Dispatch EM → EM worker (CPU), Had → Had worker
5) Worker: • terminate and get new • return unfinished

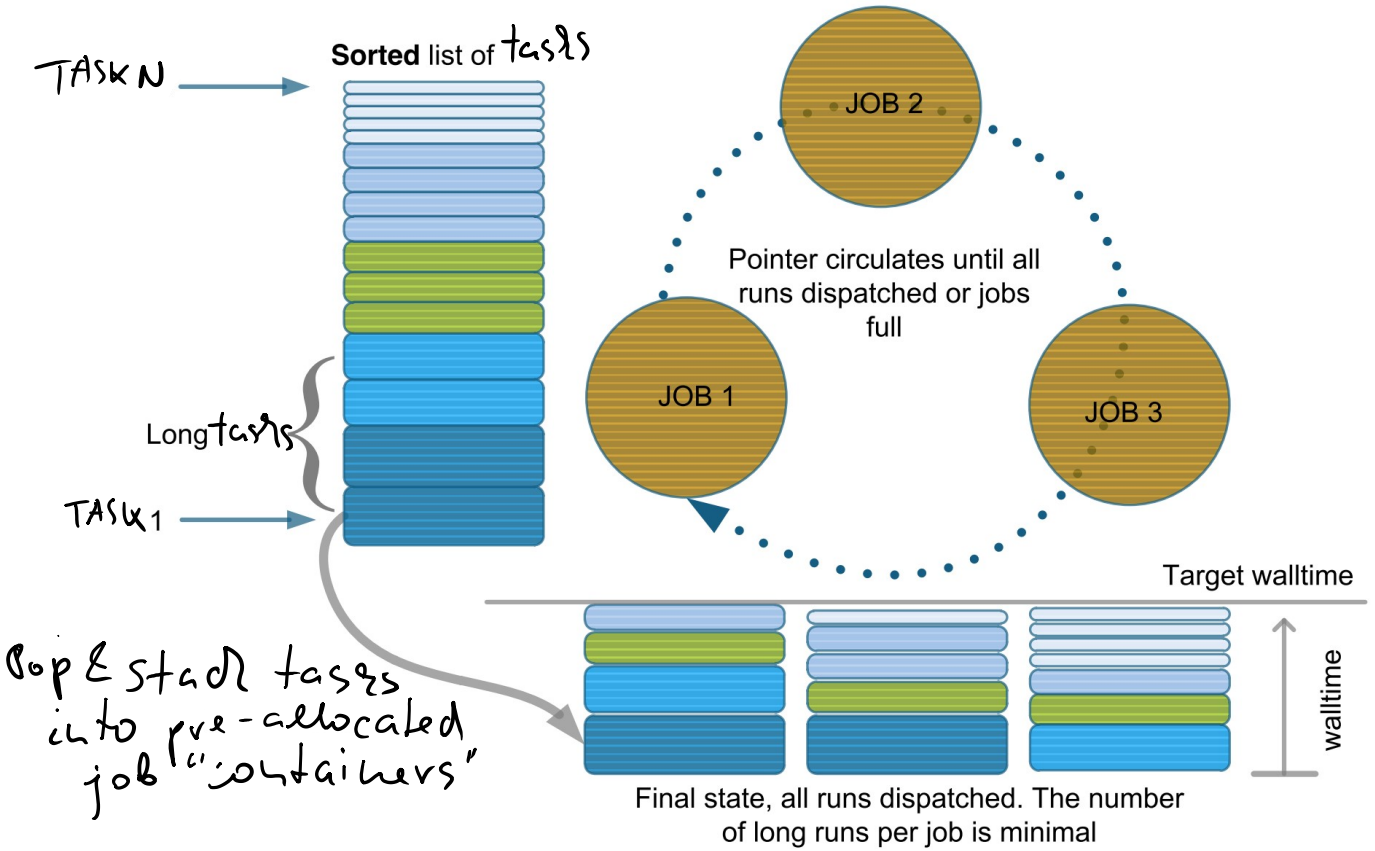


Figure 3.5: Circular dispatch algorithm. The stack of runs (chunks of the decomposed simulation) has to be sorted according to their CPU time (symbolized by the square boxes). Colors are used to emphasize that there are groups of runs with approx. same wall times. The very long runs are usually those exceeding t_{chunk}.

Scheduler implementation

- Use **MPI** for communication locally or via ethernet or strong interconnect. It is made for that.
- Accumulate first stack on the master in **sequential mode**
- Master keeps accumulating by receiving MPI messages from terminating workers.
- **Monitor** wall time and speed on each worker, migrate if necessary, log debug info locally on scratch space, transfer logs if crash is encountered to reconstruct the workers condition. Heuristics really crucial to set this up.
- Additional **storage workers may be required** to reduce the number of simultaneous data files. Or concurrent writing to few HDF files (but feel uncomfortable when thinking about it)
- Workers should be able to migrate their work: if one worker is too slow, terminate execution push the unfinished particles back to stack, and notify the master to reduce the number of workers on this machine.
- Similar approach worked great for independent CORSIKA6 runs. Target run-time 8h per-job was achieved within <10-15 minutes