Managing Heterogeneous HTCondor Workloads

(or how I learned to stop worrying and love multicore)

William Strecker-Kellogg <willsk-at-bnl.gov>

Support for Multicore Jobs in HTCondor

- Paritionable Slots
  - Each machine has one slot with all resources
  - Jobs request portions of resources (CPUs / RAM) from these
    - Sliced into dynamic slots
    - Matching is done to "parent" slot
  - See my Condor Week 2014 talk
  - Working well since 2013
- Competition
  - Starvation of multicore jobs in direct competition with smaller jobs
    - "Greedy" scheduling
  - E.g. 3 cpus become free->3 single-core jobs start
- Defragmentation
  - Depth-first filling obviates need for a lot of defrag

ATLAS Configuration

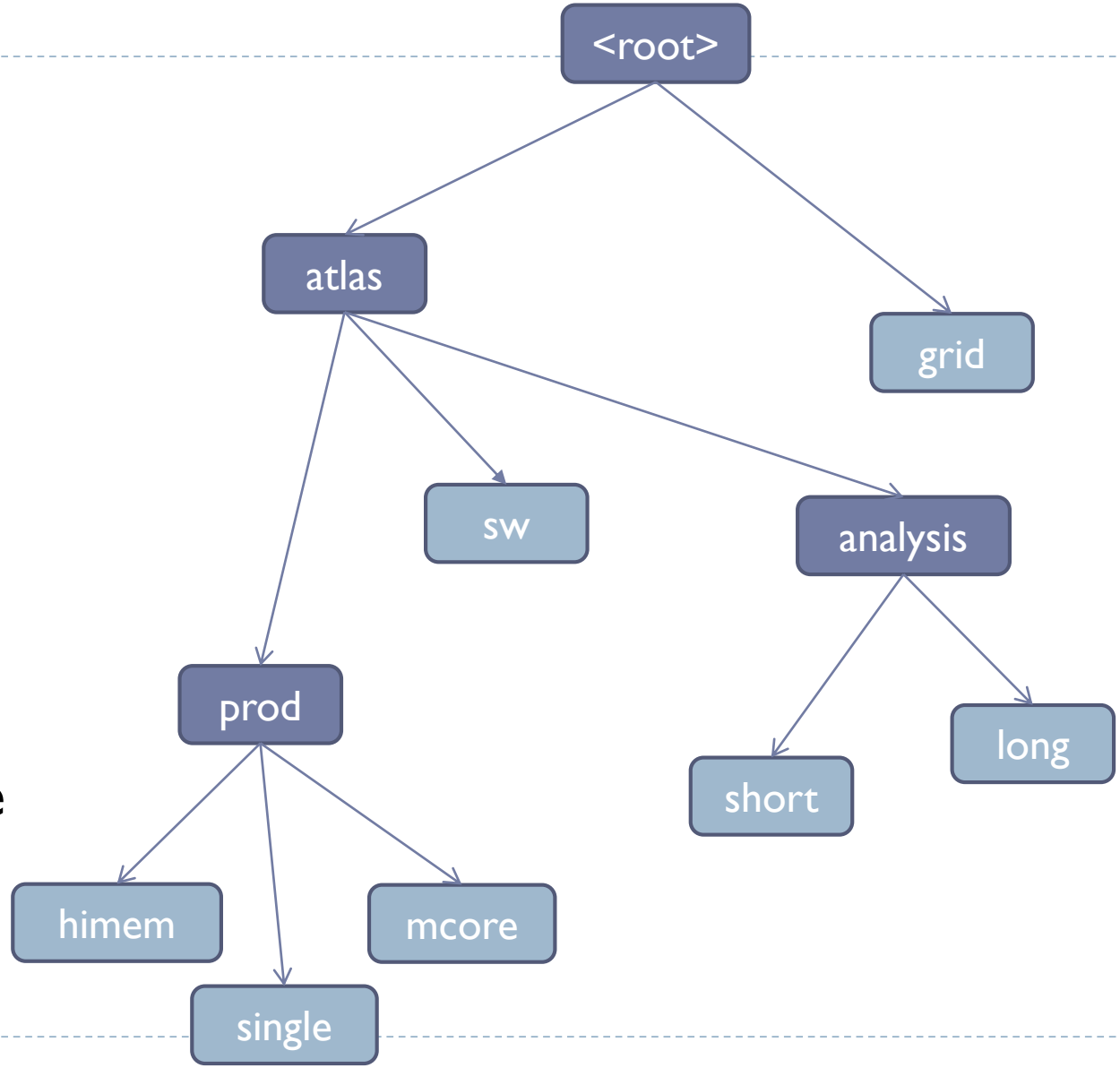
- All farm has one STARTD config

```
SLOT_TYPE_1=100%
NUM_SLOTS=1
NUM_SLOTS_TYPE_1=1
SLOT_TYPE_1_PARTITIONABLE=True
SlotWeight=Cpus
```

- Minimal Defragmentation
  - Depth-first filling with `NEGOTIATOR_POST_JOB_RANK`
  - Start 2/hr, stop when 10 free CPUs appear

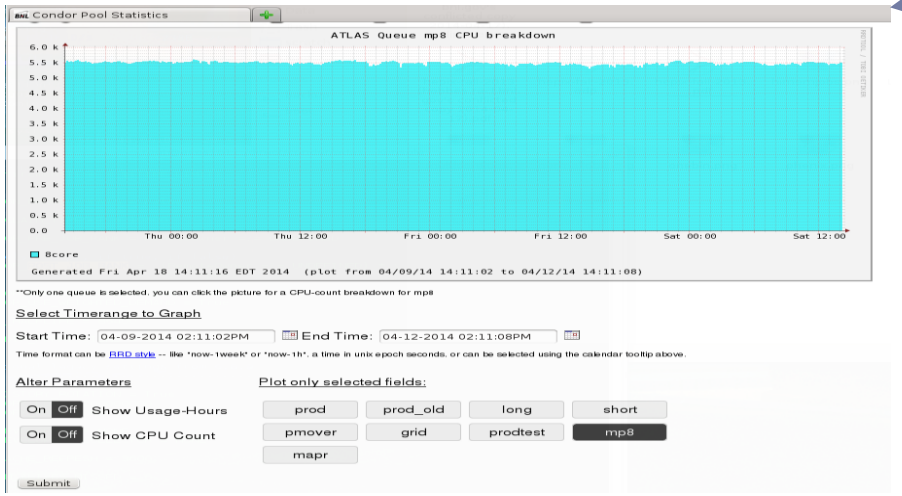
Provisioning

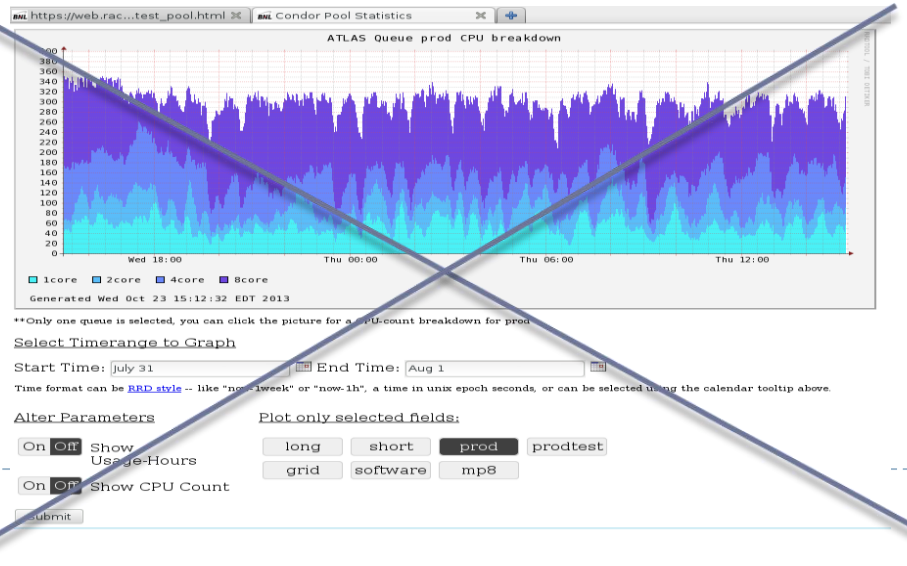
- Use Hierarchical Group Quotas
  - Partition ATLAS into production / analysis
  - Partition jobs of different sizes into different groups at same level of tree
  - Logical divisions along other job-qualities (length)
  - Leaf nodes receive jobs
- Up to site administrators to define



Queues & Quotas

One size per-queue





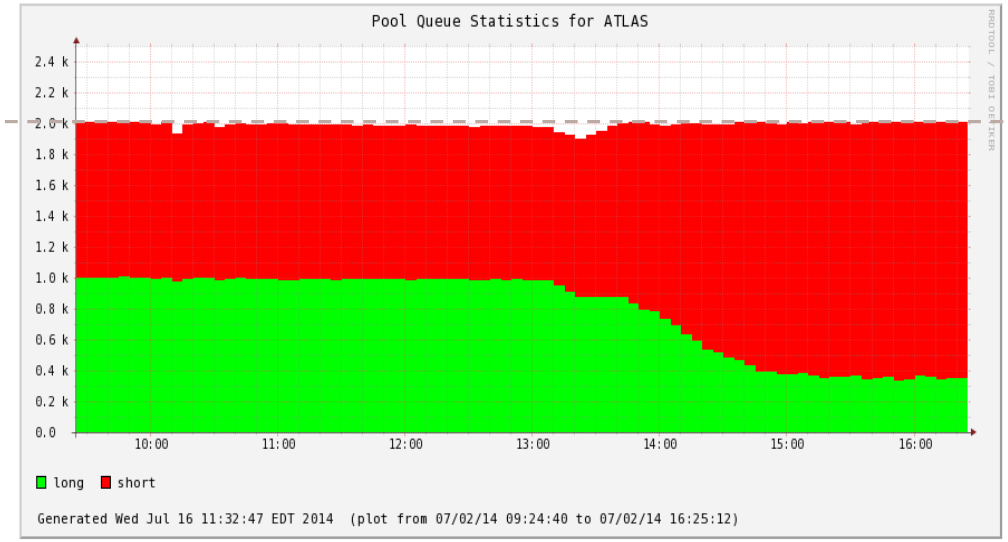
- Quotas on intermediate groups are sum(child-quotas)
  - Up the tree till the root-node has a quota the size of farm
- Jobs are segregated by resource-usage profile into different groups
- In ATLAS, Groups are 1-to-1 with Panda Queues

Surplus Sharing

- Surplus sharing is controlled by boolean *accept_surplus* flag on each queue
  - Quotas are normalized in units of SlotWeight (CPUs)
- Groups with flag set to True can take unused slots from their siblings
  - Parent groups with flag allow surplus to "flow down" the tree from their siblings to their children
  - Parent groups without *accept_surplus* flag constrain surplus-sharing to among their children

Surplus Sharing

- Scenario: **analysis** has quota of *2000* and no *accept_surplus*; **short** and **long** have a quota of *1000* each and *accept_surplus* on
  - short=1600, long=400…possible
  - short=1500, long=700…impossible (violates analysis quota)



Management of Queues

- We have a solution
  1. Database to hold group-tree information and parameters
  2. Flask website to manage quotas easily
  3. Scripts to inject changes from database into your condor pool

- Available on my github
  - Installable as python PIP package or RPM
  - Site in production at BNL for the last 3 months

- Demo…

Sharing with Multicore

- Multicore queues can't have accept-surplus set alongside single-core queues
  - Scheduling is greedy, and would require much more defragmentation to work

- Solutions:
  - Factor your tree so this never happens?
    - This is nice but we can do better!
  - Automatically rebalance surplus based on demand
    1. Need a way to read demand… this is site-specific
    2. Assign parameters to determine "weight" of queues

Weight, Demand & Threshold

- Each queue's "weight" is the size of its jobs
  - 8 for mcore…
  - 2 for high-memory…
- Demand is read with a script from PANDA
  - This is the only site-specific code that needs to be written
  - Could be a simple condor_q lookup of Idle jobs
- Threshold is what average demand for last hour must be above in order for queue to be considered "full"
  - Adjust to best fit how "fast" each queue moves vs. how big the jobs are

Balancing Algorithm

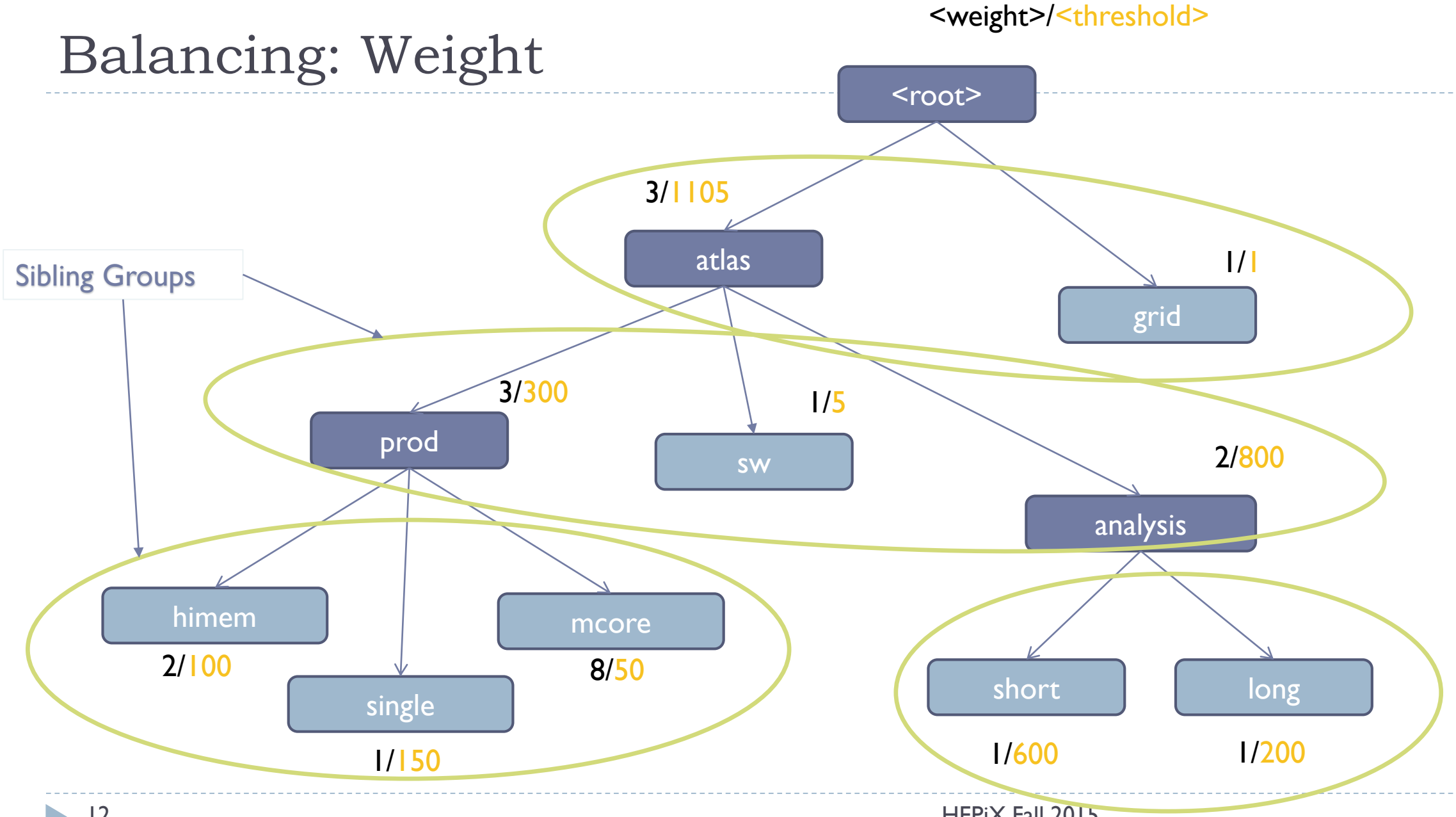
Assign Variables

- Weights of intermediate groups
  - $\text{avg}(w_{\text{children}})$
- Demand of intermediate groups
  - $\text{sum}(d_{\text{children}})$

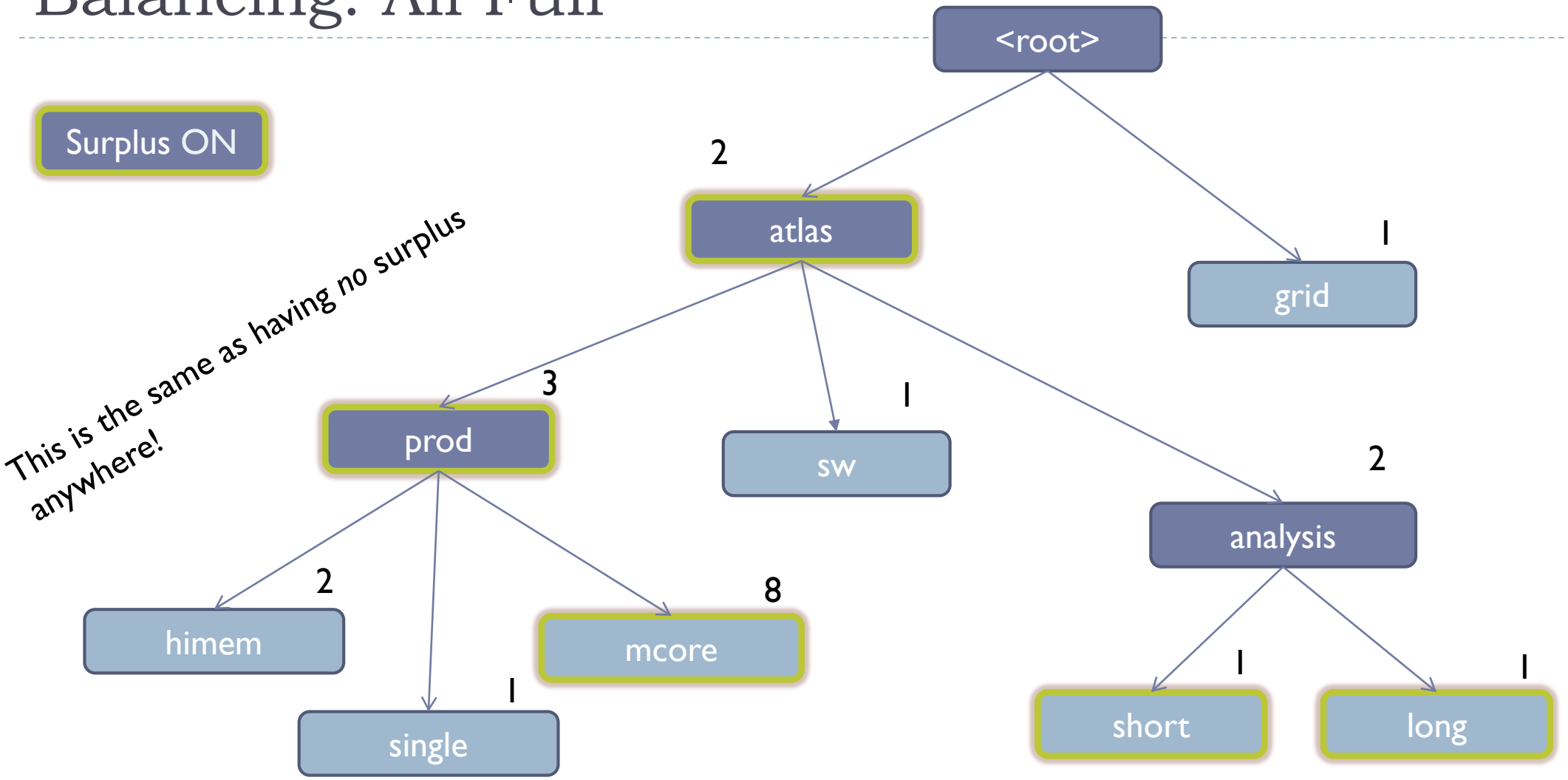
Algorithm

- Depth-first traversal
  - For each equal-level sibling-group
    - Set *accept_surplus* to TRUE for only all the highest-weighted queues that have demand

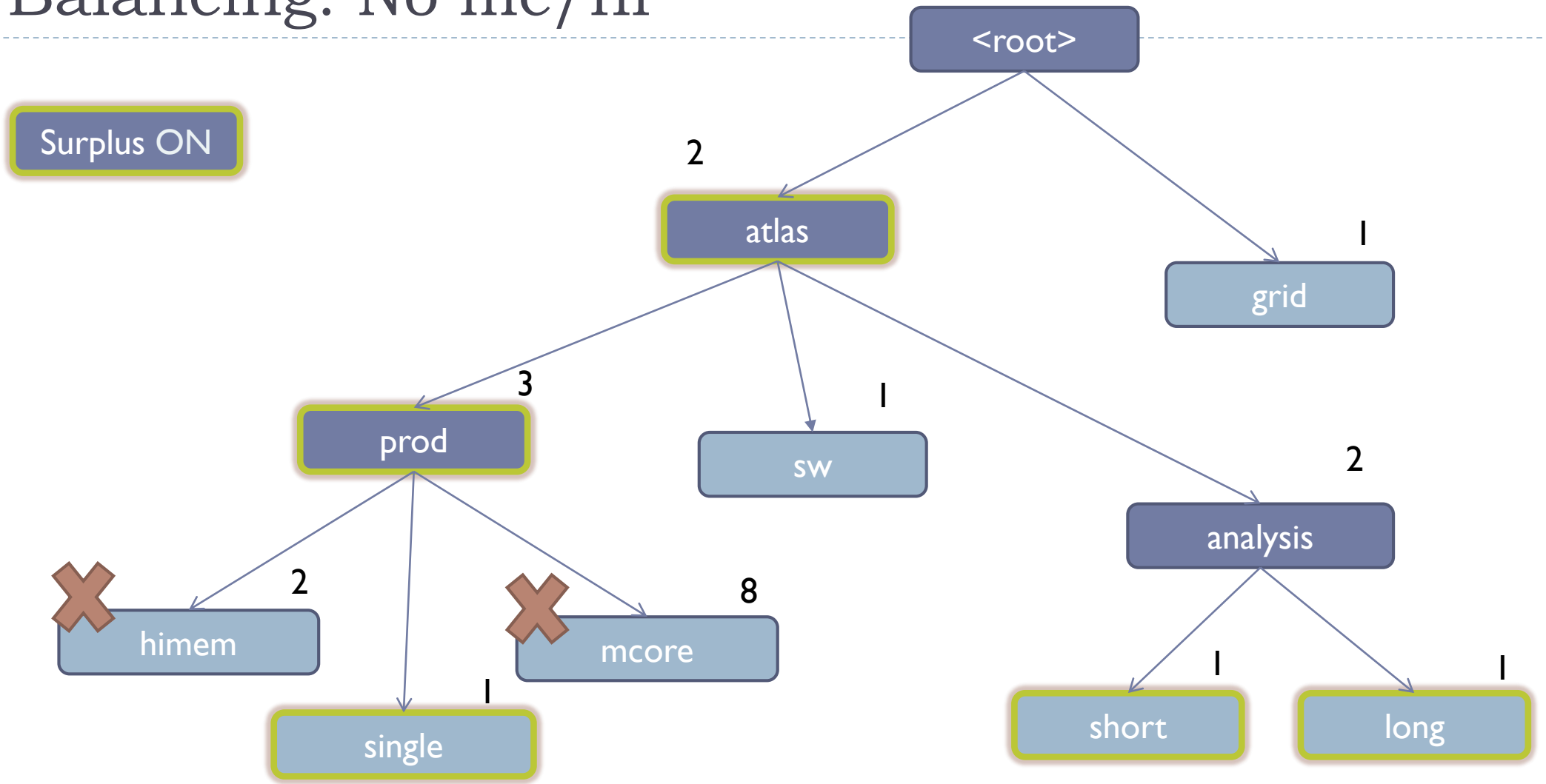
Balancing: Weight



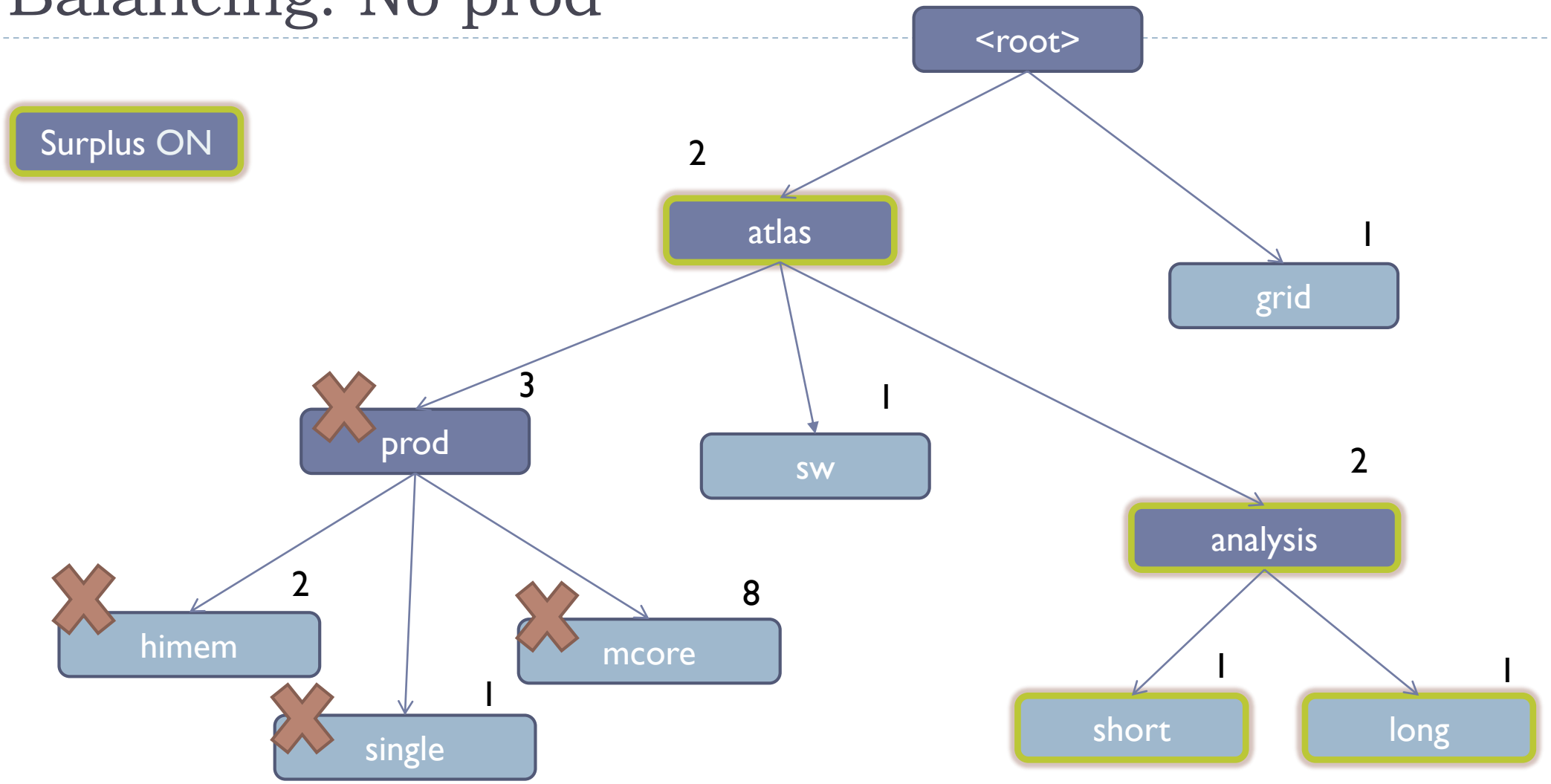
Balancing: All Full



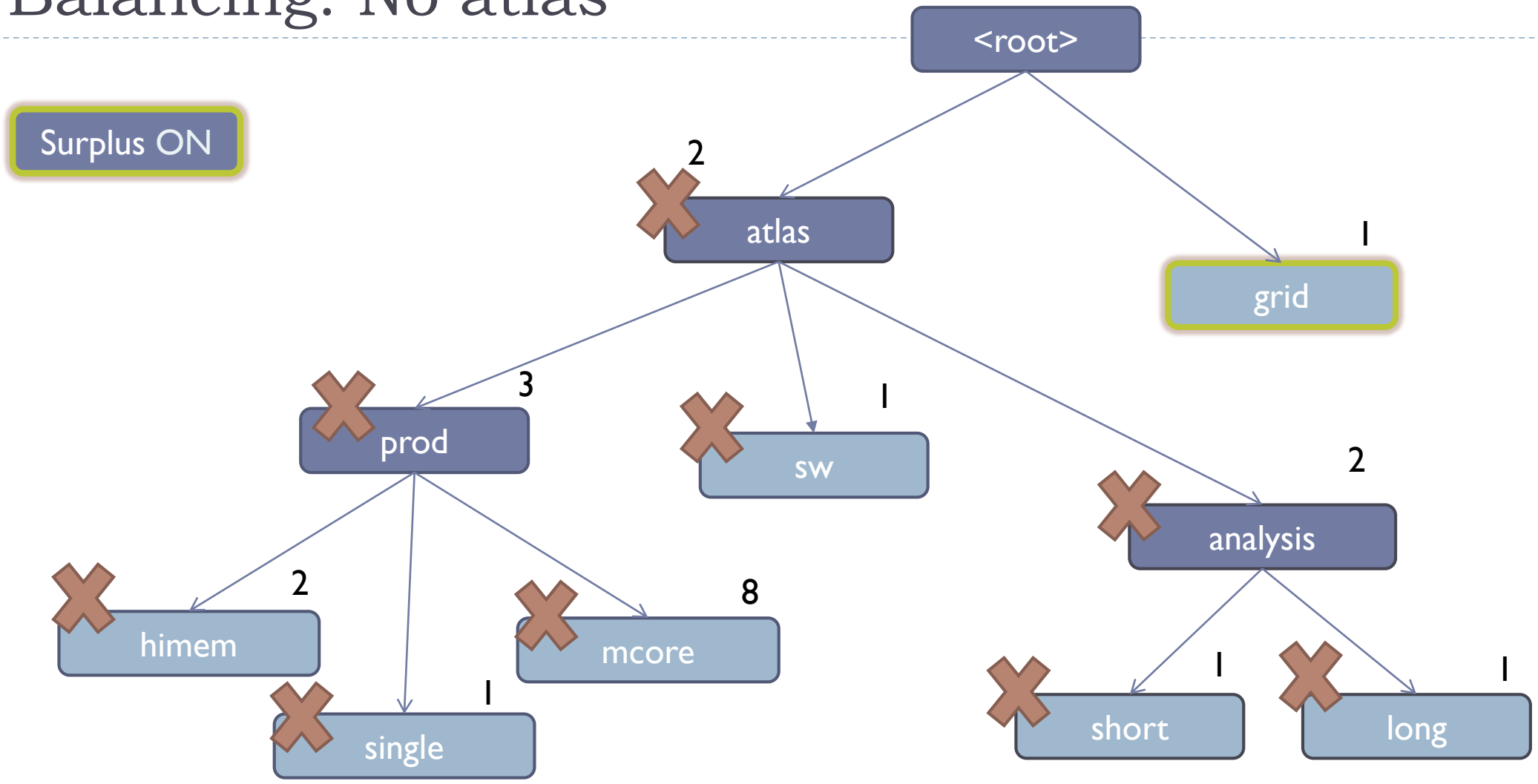
Balancing: No mc/hi



Balancing: No prod

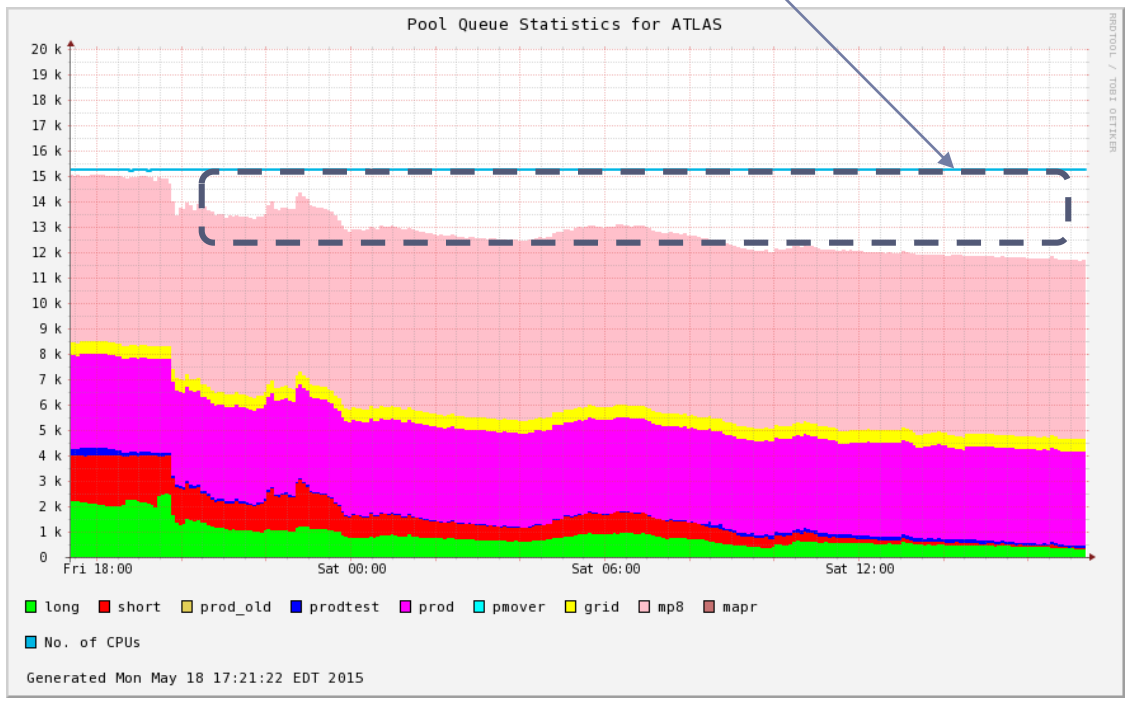


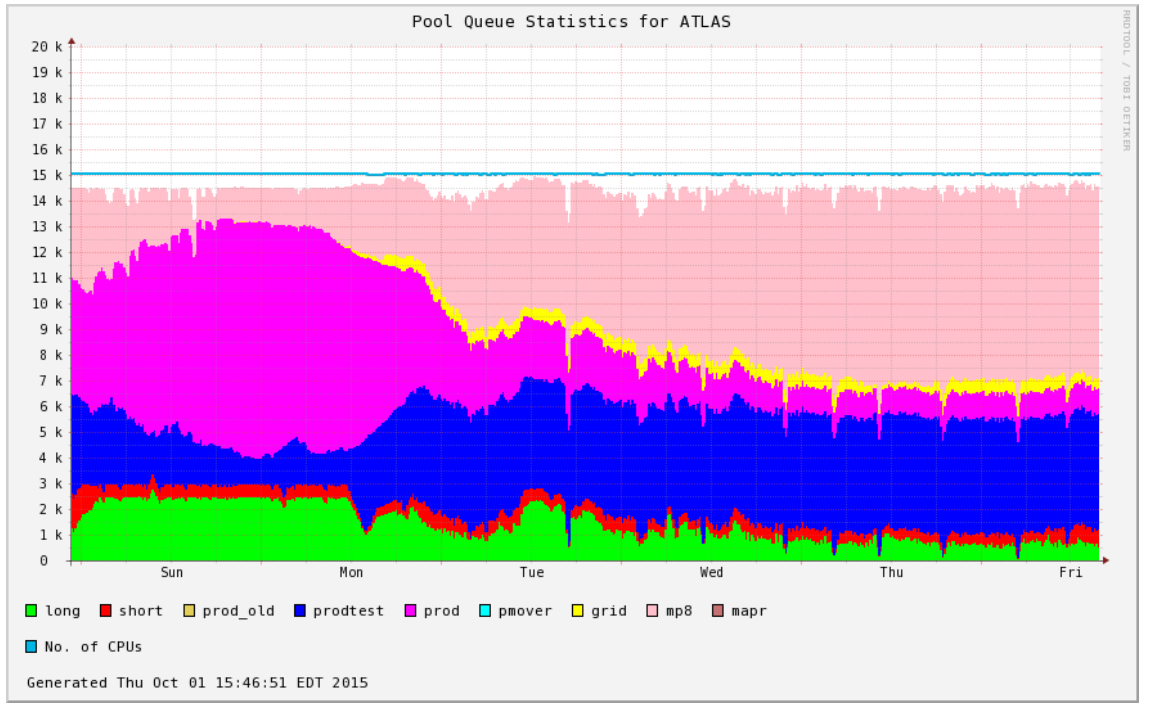
Balancing: No atlas



Results

Wasted Slots





Results & Context

- Multicore is ready to take slack from other production queues
- Spotty analysis-demand the past few months has allowed many millions of CPU-hours to go unwasted
- If all ATLAS has a lull in demand, OSG jobs can fill the farm

- Who is this useful for?
  - Algorithm works for any tree
    - Extensible beyond ATLAS where work is structured outside of batch system
  - A multi-tenant service provide with a hierarchy of priorities
    - Really a problem of efficient provisioning, not scheduling
- Constraints
  - Workflow defined outside of HTCondor
  - Must map job-species to groups

The End

Thank You!

Questions? Comments?