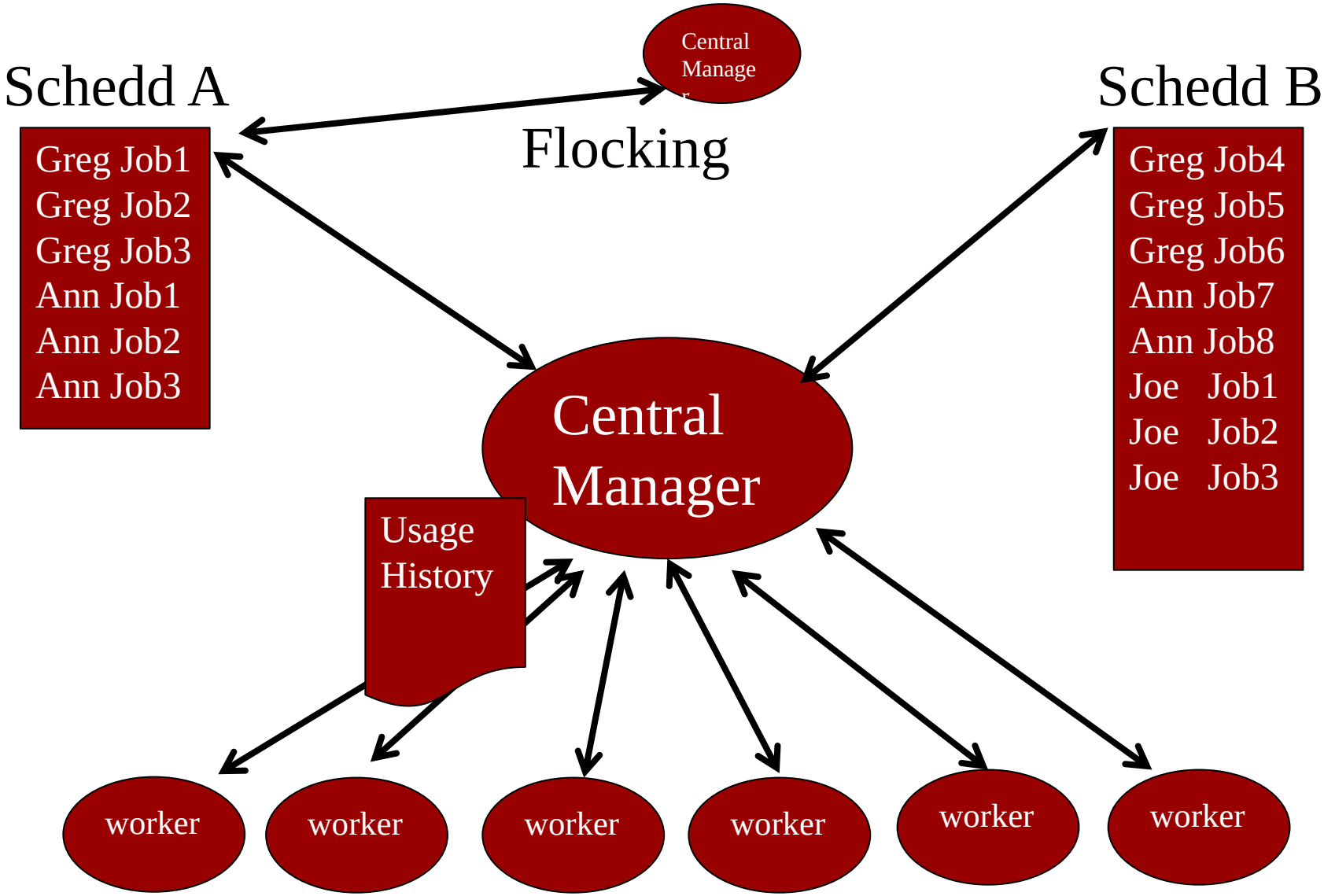
HTCondor scheduling policy

- All kinds of mechanisms

Overview of Condor Architecture



Schedd Policy: Job Priority

- Set in submit file with
- `JobPriority = 7`
- … or dynamically with condor_prio cmd
- Users can set priority of their own jobs
- Integers, larger numbers are better priority
- Only impacts order between jobs for a single user on a single schedd
- A tool for users to sort their own jobs

Schedd Policy: Job Rank

- Set with
  - `RANK = Memory`
- In condor_submit file
- Not as powerful as you may think:
  - Remember steady state condition

Concurrency Limits

- Another Central manager
- In central manager config
  - `FOO_LIMIT = 10`
- In submit file
  - `concurrency_limits = foo`

Rest of this talk: Provisioning, or Scheduling

- schedd sends all idle jobs to the negotiator
- Negotiator picks machines (idle or busy) to match to these idle jobs
- How does it pick?

Negotiator metric: User Priority

- Negotiator computes, stores the user prio

- View with **`condor_userprio` tool**
- Inversely related to machines allocated (lower number is better priority)
  - A user with priority of 10 will be able to claim twice as many machines as a user with priority 20

What's a user?

- Bob in schedd1 same as Bob in schedd2?
- If have same UID_DOMAIN, the are.
- Prevents cheating by adding shedds

- We'll talk later about other user definitions.

- Map files can define the local user name

User Priority (2)

- (Effective) User Priority is determined by multiplying two components

- Real Priority * Priority Factor

Real Priority

- Based on actual usage
- Starts at 0.5
- Approaches actual number of machines used over time
  - Configuration setting **`PRIORITY_HALFLIFE`**
  - If PRIORITY_HALFLIFE = +Inf, no history
  - Default one day (in seconds)
- Asymptotically grows/shrinks to current usage

Priority Factor

- Assigned by administrator
  - Set/viewed with **`condor_userprio`**
  - Persistently stored in CM
- Defaults to 100 (**`DEFAULT_PRIO_FACTOR`**)
  - Used to default to 1
- Allows admins to give prio to sets of users, while still having fair share within a group
- “Nice user”s have Prio Factors of 1,000,000

condor_userprio

- Command usage:

```
condor_userprio –most
```

```
                                     Effective  Priority
User Name                            Priority    Factor   In Use (wghted-hrs) Last Usage
---------------------------------------------- --------- ------ ----------- ----------
lmichael@submit-3.chtc.wisc.edu            5.00     10.00      0        16.37    0+23:46
blin@osghost.chtc.wisc.edu                 7.71     10.00      0      5412.38    0+01:05
osgtest@osghost.chtc.wisc.edu             90.57     10.00     47     45505.99      <now>
cxiong36@submit-3.chtc.wisc.edu          500.00   1000.00      0         0.29    0+00:09
ojalvo@hep.wisc.edu                      500.00   1000.00      0    398148.56    0+05:37
wjiang4@submit-3.chtc.wisc.edu           500.00   1000.00      0         0.22    0+21:25
cxiong36@submit.chtc.wisc.edu            500.00   1000.00      0        63.38    0+21:42
```

A note about Preemption

- Fundamental tension between
  - Throughput vs. Fairness
- Preemption is required to have fairness

- Need to think hard about runtimes, fairness and preemption
- Negotiator implementation preemption
- (Workers implement eviction: different)

Negotiation Cycle

- Gets all the slot ads
- Updates user prio info for all users
- Based on user prio, computes submitter limit for each user
- Foreach user, finds the schedd, gets a job
  - Finds all matching machines for job
  - Sorts the jobs
  - Gives the job the best sorted machine

Sorting slots: sort levels

```
NEGOTIATOR_PRE_JOB_RANK =
    RemoteOwner =?= UNDEFINED

JOB_RANK = mips

NEGOTIATOR_POST_JOB_RANK =
    (RemoteOwner =?= UNDEFINED) *
(KFlops - SlotID)
```

If Matched machine claimed, extra checks required

- **`PREEMPTION_REQUIREMENTS`** and **`PREEMPTION_RANK`**
- Evaluated when **`condor_negotiator`** considers replacing a lower priority job with a higher priority job
- Completely unrelated to the **`PREEMPT`** expression (which should be called evict)

PREEMPTION_REQUIREMENTS

- MY = busy machine
- TARGET = candidate job
- If false will not preempt machine
  - Typically used to avoid pool thrashing
  - Typically use:
    - **RemoteUserPrio** – Priority of user of currently running job (higher is worse)
    - **SubmittorPrio** – Priority of user of higher priority idle job (higher is worse)
- **PREEMPTION_REQUIREMENTS=FALSE**

PREEMPTION_REQUIREMENTS

- Only replace jobs running for at least one hour and 20% lower priority

```
StateTimer = \
  (CurrentTime - EnteredCurrentState)
HOUR = (60*60)
PREEMPTION_REQUIREMENTS = \
  $(StateTimer) > (1 * $(HOUR)) \
  && RemoteUserPrio > SubmittorPrio * 1.2
   NOTE: classad debug() function v. handy
```

PREEMPTION_RANK

- Of all claimed machines where PREEMPTION_REQUIREMENTS is true, picks which one machine to reclaim
- Strongly prefer preempting jobs with a large (bad) priority and a small image size

```
PREEMPTION_RANK = \
  (RemoteUserPrio * 1000000)\
  - ImageSize
```

MaxJobRetirementTime

- Can be used to guarantee minimum time
- E.g. if claimed, give an hour runtime, no matter what:

- MaxJobRetirementTime = 3600
- Can also be an expression

Partitionable slots

- What is the “cost” of a match?
  - SLOT_WEIGHT (cpus)
- What is the cost of an unclaimed pslot?
  - The whole rest of the machine
  - Leads to quantization problems
- By default, schedd splits slots
- “Consumption Policies”
  - Still some rough edges

Accounting Groups (2 kinds)

- Manage priorities across groups of users and jobs
- Can guarantee maximum numbers of computers for groups (quotas)
- Supports hierarchies
- Anyone can join any group

Accounting Groups as Alias

- In submit file
  - Accounting_Group = “group1”

- Treats all users as the same for priority
- Accounting groups not pre-defined
- No verification – condor trusts the job
- condor_userprio replaces user with group

Prio factors with groups

```
condor_userprio –setfactor 10 group1.wisc.edu
Condor_userprio –setfactor 20 group2.wisc.edu
```

Note that you must get UID_DOMAIN correct

Gives group1 members 2x resources as group2

Accounting Groups w/ Quota

- Must be predefined in cm's config file:

```
GROUP_NAMES = a, b, c
GROUP_QUOTA_a = 10
GROUP_QUOTA_b = 20
```

- And in submit file:

```
Accounting_Group = a
Accounting_User = gthain
```

Strict quotas then enforce

- "a" limited to 10
- "b" to 20,
- Even if idle machines
- What is the unit?
  - Slot weight.
- With fair share of uses within group

- Must be predefined in cm's config file:

```
GROUP_NAMES = a, b, c
GROUP_QUOTA_a = 10
GROUP_QUOTA_b = 20
```

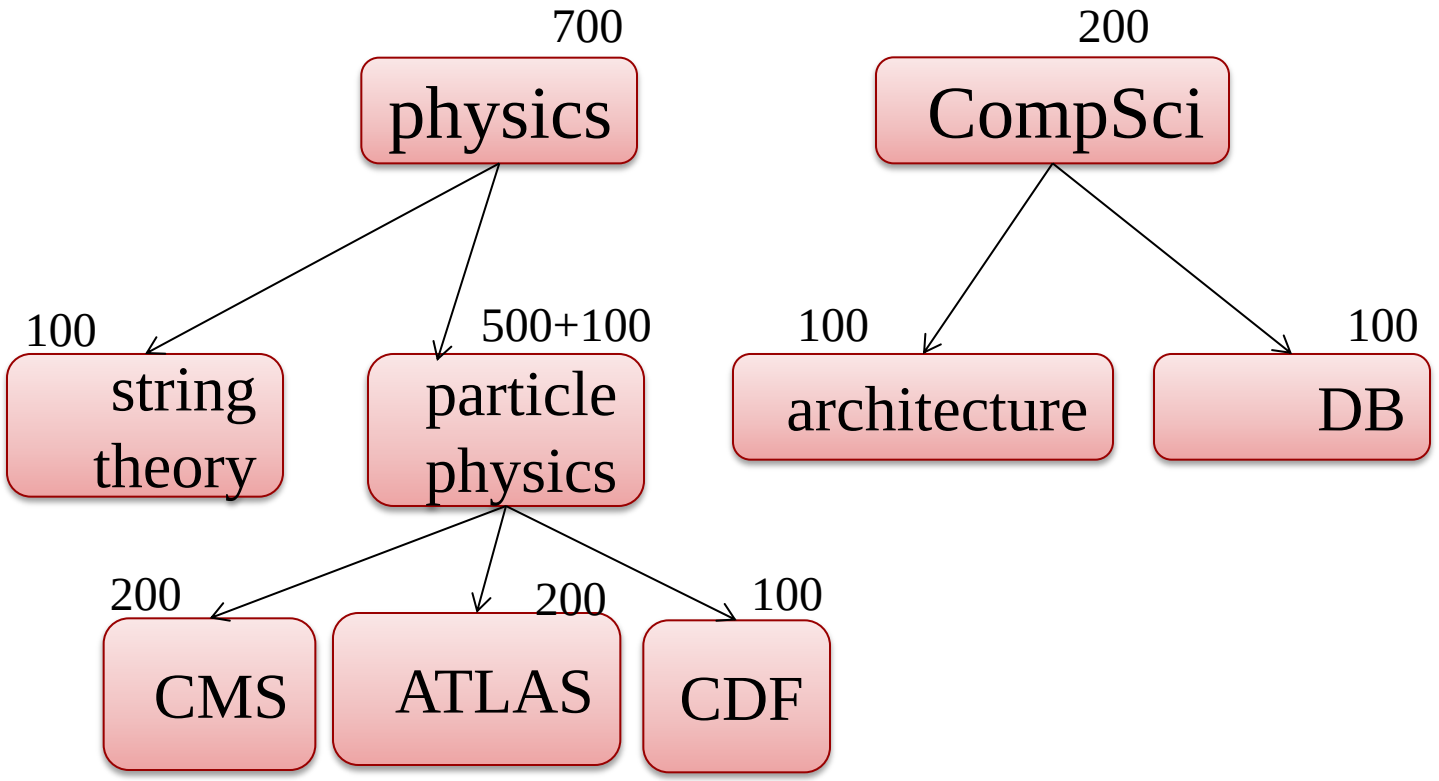
- And in submit file:

```
Accounting_Group = a
Accounting_User = gthain
```

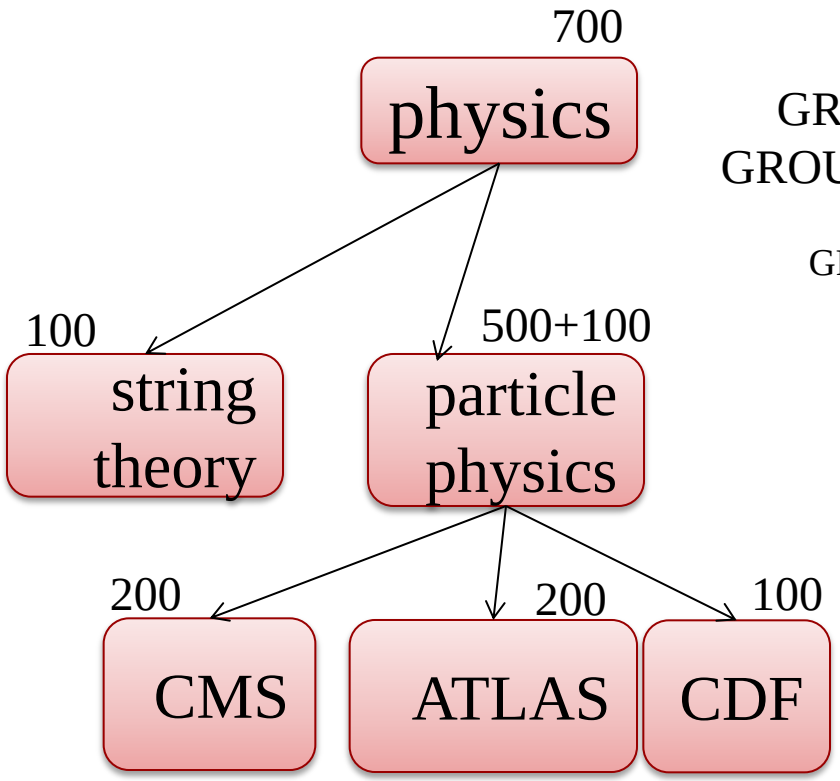
GROUP_AUTOREGROUP

- Allows groups to go over quota if idle machines
- “Last chance” round, with every submitter for themselves.

Hierarchical Group Quotas



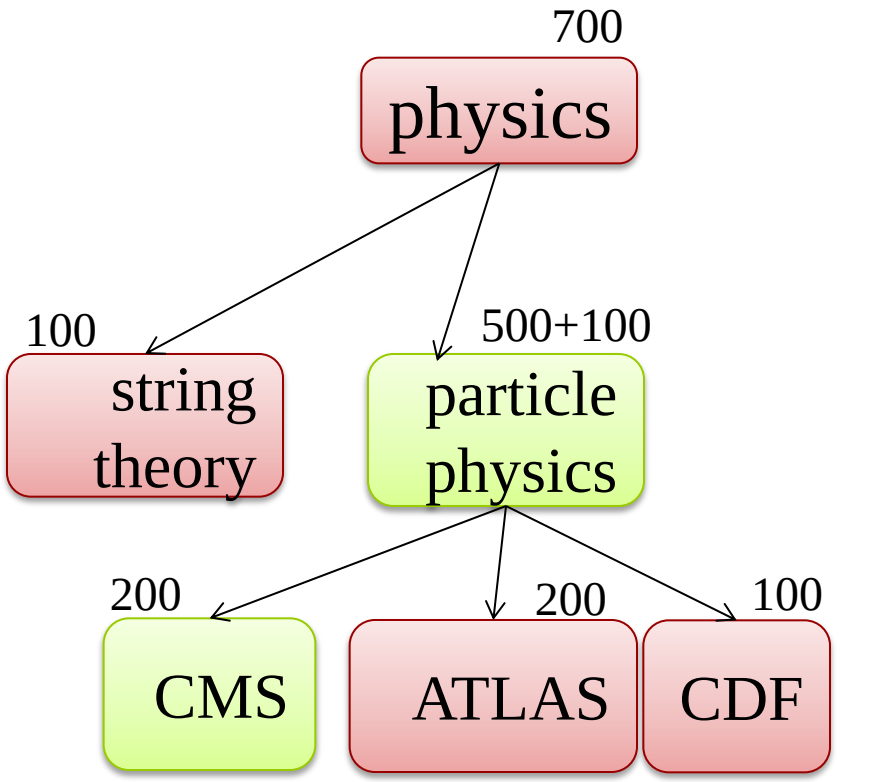
Hierarchical Group Quotas




```
GROUP_QUOTA_physics = 700
GROUP_QUOTA_physics.string_theory = 100
GROUP_QUOTA_physics.particle_physics = 600
GROUP_QUOTA_physics.particle_physics.CMS = 200
GROUP_QUOTA_physics.particle_physics.ATLAS = 200
GROUP_QUOTA_physics.particle_physics.CDF = 100
```

group.sub-group.sub-sub-group…

Hierarchical Group Quotas

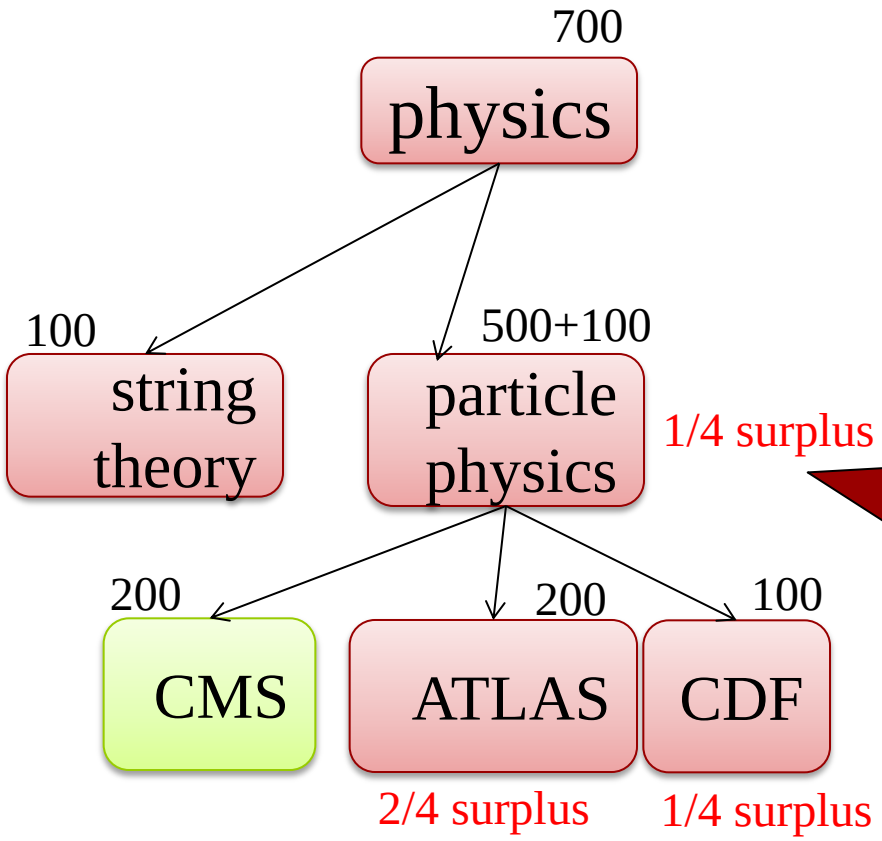


Groups configured to accept surplus will share it in proportion to their quota.

Here, unused particle physics surplus is shared by ATLAS and CDF.

GROUP_ACCEPT_SURPLUS_physics.particle_physics.ATLAS = true
GROUP_ACCEPT_SURPLUS_physics.particle_physics.CDF = true

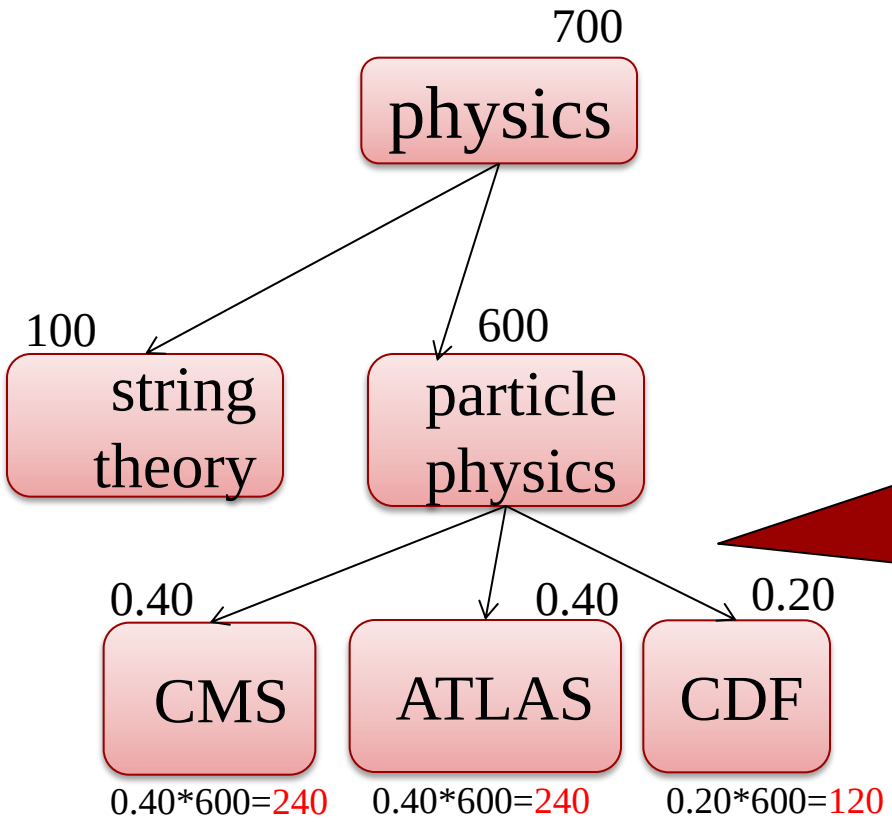
Hierarchical Group Quotas



Job submitters may belong to a parent group in the hierarchy.

Here, general particle physics submitters share surplus with ATLAS and CDF.

Hierarchical Group Quotas

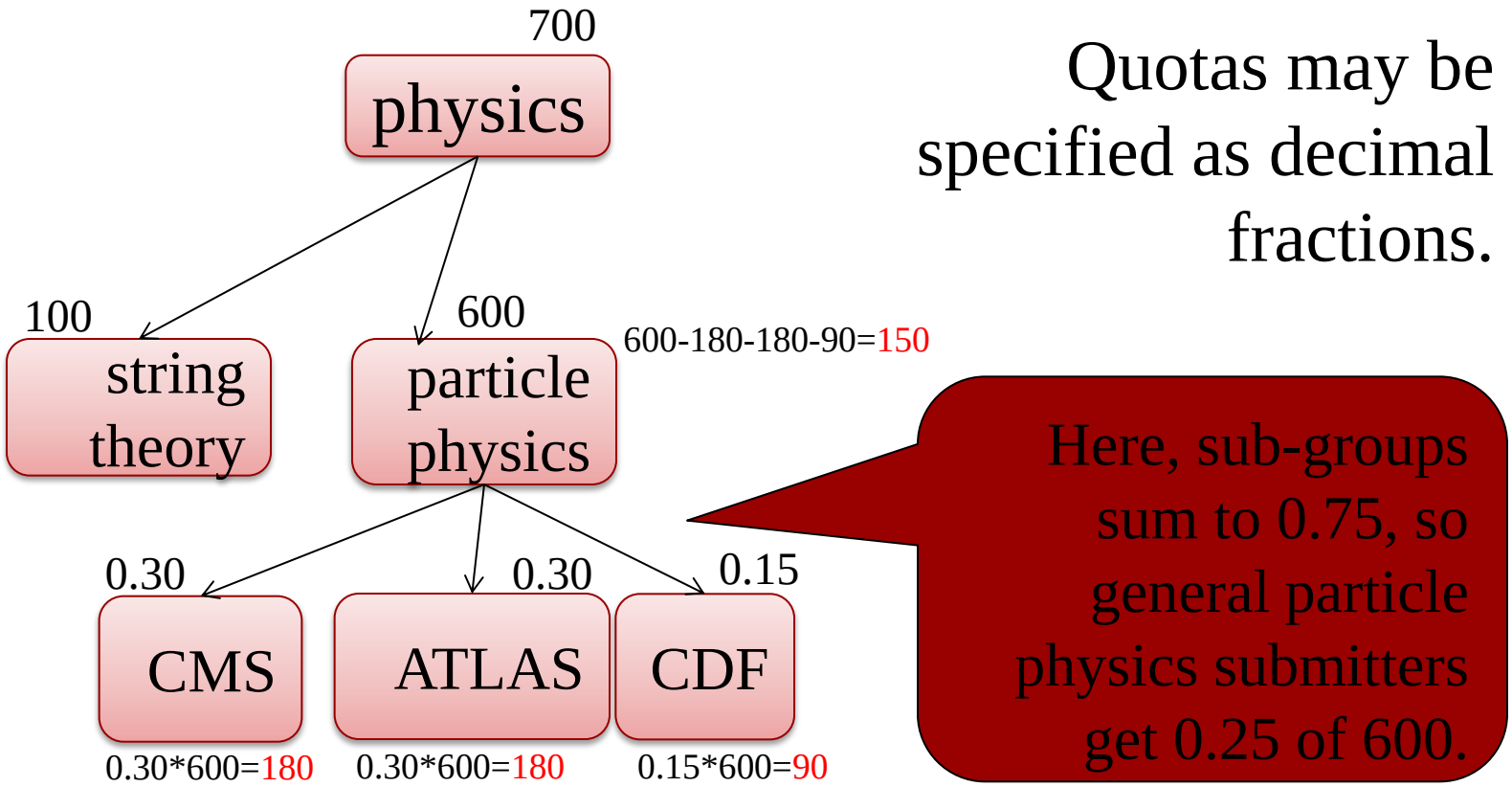


Quotas may be specified as decimal fractions.

Here, sub-groups sum to 1.0, so general particle physics submitters get nothing.

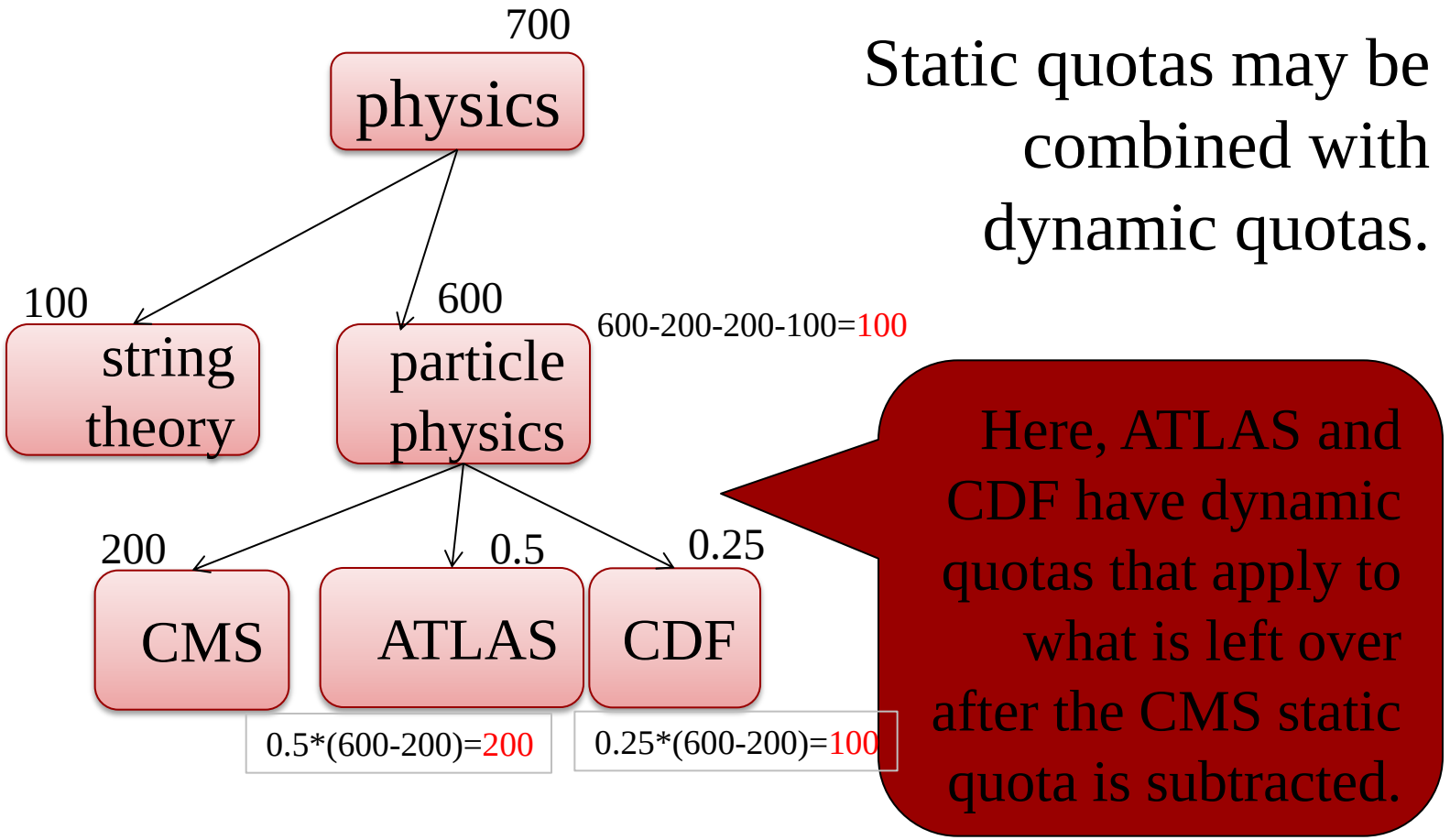
GROUP_QUOTA_DYNAMIC_physics.particle_physics.CMS=0.4

Hierarchical Group Quotas



Quotas may be specified as decimal fractions.

Hierarchical Group Quotas



Static quotas may be combined with dynamic quotas.

Here, ATLAS and CDF have dynamic quotas that apply to what is left over after the CMS static quota is subtracted.

Preemption with HQG

By default, won't preempt to make quota

But, "there's a knob for that"

```
PREEMPTION_REQUIREMENTS =
(SubmitterGroupResourcesInUse <
SubmitterGroupQuota) &&
(RemoteGroupResourcesInUse >
RemoteGroupQuota) && ( RemoteGroup =!=
SubmitterGroup
```

Group_accept_surplus

- Group_accept_surplus = true
- Group_accept_surplus_a = true
- This is what creates hierarchy
  - But only for quotas

Gotchas with quotas

- Quotas don't know about matching
- Assuming everything matches everything
- Surprises with partitionable slots
- Preempting multiple slots a problem

- May want to think about draining instead.

Summary

- Many ways to schedule