

# **Negotiator Policy and Configuration**

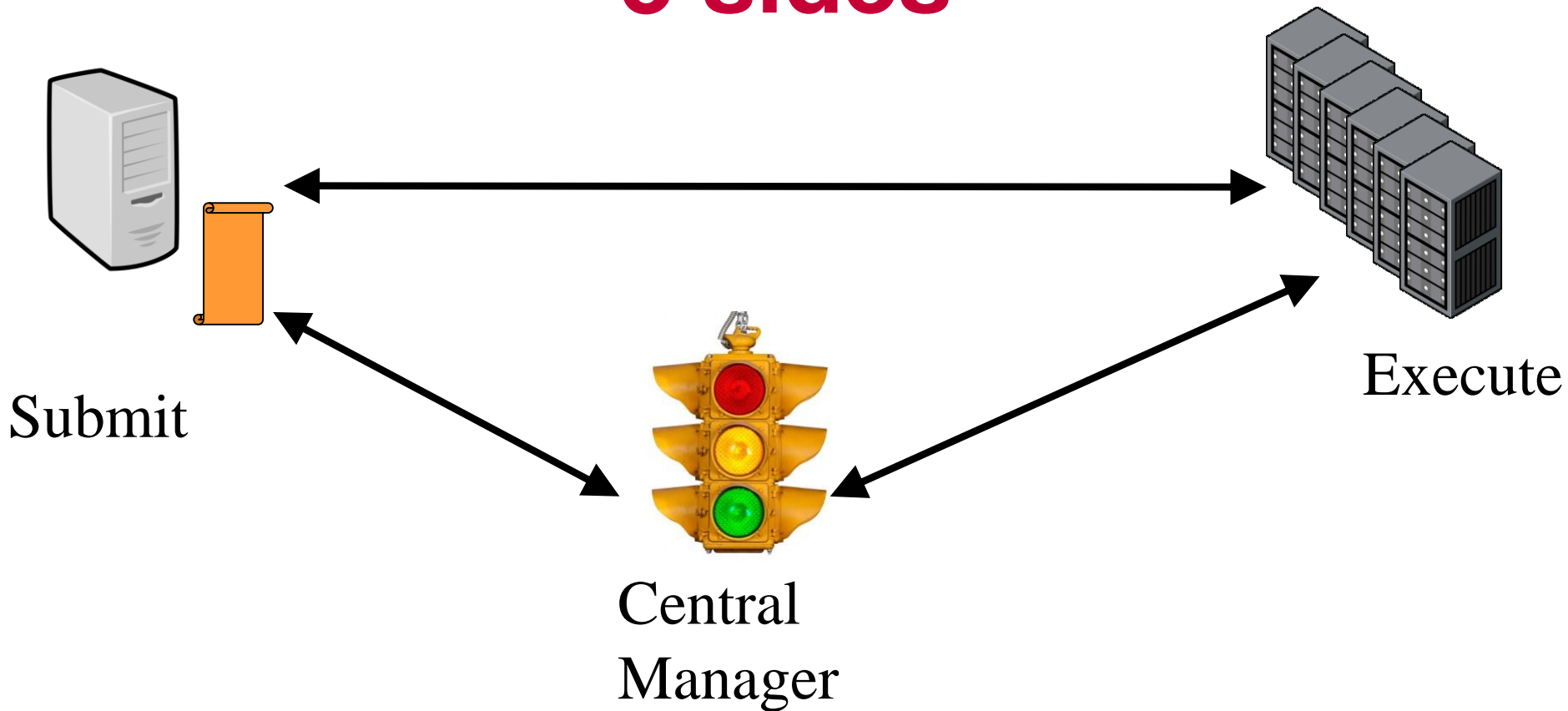
Greg Thain

# **Fairness in HTCondor and how to avoid it**

# Agenda

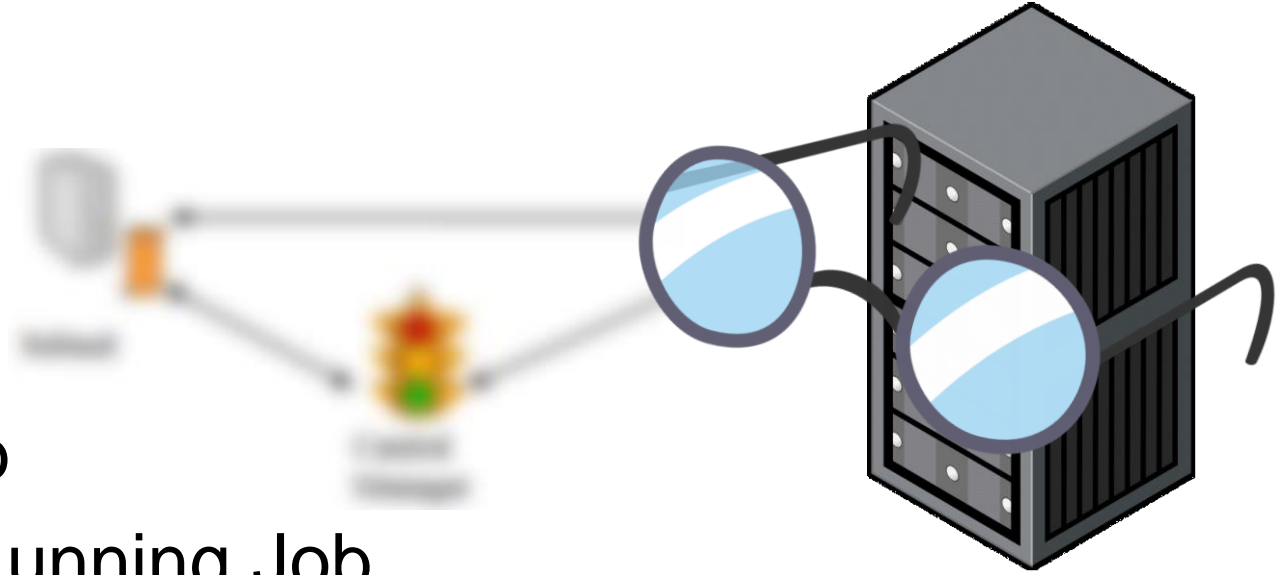
- › Understand role of negotiator
- › Learn how priorities work
- › Learn how quotas work
  
- › Encourage thought about possible policies!

# Overview of condor 3 sides



# Startd Mission Statement

- › Near sighted
- › 3 inputs only:
  - Machine
  - Running Job
  - Candidate Running Job
- › Knows nothing about the rest of the system!



# Schedd mission

Run *jobs* on  
*slots* the negotiator  
has assigned to *submitters*.

Inputs:

All the jobs in that schedd

All the slots given to it by the negotiator

# Schedd mission

Schedd Can:

- Re-use a slot for  $> 1$  job (in succession)

- Pick which job for a user goes first

Schedd cannot:

- Reassign slots from one submitter to other

# Submitter vs User

- › Submitters: what are they?
- › User: an OS construct
- › Submitter: Negotiator construct

# User not one-to-one w/Submitter

nice\_user = true in submit file

Creates 2<sup>nd</sup> submitter with poor priority

But accounted for separately:

# Negotiation Mission

Assign the *slots* of the whole pool  
to *users* based on some *policy* that's 'fair'



# Negotiator Inputs

- › All the slots in the pool
- › All the submitters in the pool
- › All the submitters' priorities and quotas
- › One request per submitter at a time

# How the Negotiator Works

Periodically tries to:

***Rebalance*** %age of slots assigned to users

*Via preemption, if enabled*

*Via assigning empty slots if not*

*Negotiator is always a little out of date*

# Concurrency Limits

- › Simplest Negotiator (+ schedd) policy
- › Useful for pool wide, across user limits,

# Useful Concurrency Limits:

> 100 running NFS jobs crash my server

License server only allows X concurrent uses

Only want 10 database jobs running at once

# Concurrency Limits: How to Configure

add to negotiator config file  
(condor\_reconfig needed):

```
NFS_LIMIT = 100  
DB_LIMIT = 42  
LICENSE_LIMIT = 5
```

# Concurrency Limits: How to use

Add to job ad

```
Executable = somejob
```

```
Universe = vanilla
```

```
...
```

```
ConcurrencyLimits = NFS
```

```
queue
```

# Concurrency Limits: How to use

OR

```
Executable = somejob
```

```
Universe = vanilla
```

```
...
```

```
ConcurrencyLimits = NFS:4
```

```
queue
```

# Concurrency Limits: How to use

Add to job ad

```
Executable = somejob
```

```
Universe = vanilla
```

```
...
```

```
ConcurrencyLimits = NFS,DB
```

```
queue
```

# Part of the picture

- › Concurrency limits very “strong”
- › Can throw off other balancing algorithms
- › No “fair share” of limits

**“Fair Share of Users”**

# Main Loop of Negotiation Cycle\*

1. Get all slots in the pool
2. Get all ~~jobs~~ submitters in pool
3. Compute # of slots submitters should get
4. In priority order, hand out slots to submitters
5. Repeat as needed

# The Negotiator as Shell Script

1. Get all slots in the pool
2. Get all ~~jobs~~ submitters in pool
3. Compute # of slots submitters should get
4. In priority order, hand out slots to submitters
5. Repeat as needed

# 1: Get all slots in pool

# 1: Get all slots in pool

```
$ condor_status
```

# 1: Get all slots\* in pool

```
NEGOTIATOR_SLOT_CONSTRAINT = some classad expr
```

```
NEGOTIATOR_SLOT_CONSTRAINT
```

Defaults to true, what subset of pool to use

For sharding, etc.

# 1: Get all slots in pool

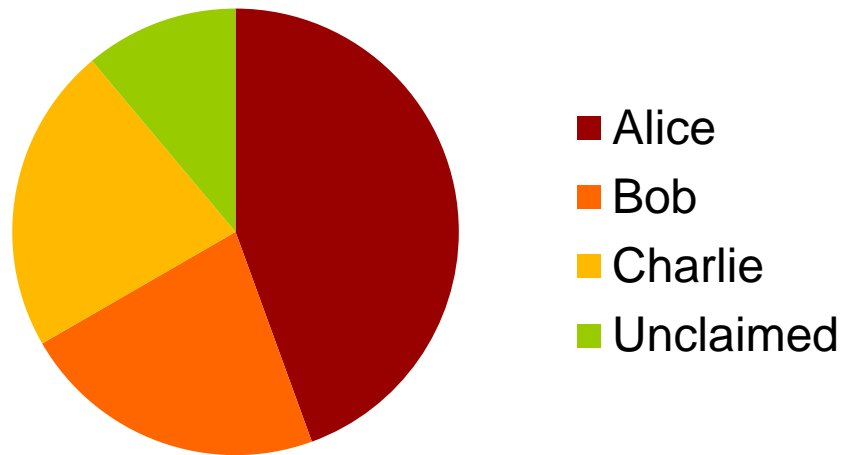
```
$ condor_status -af Name State RemoteOwner
```

```
slot1@... Claimed Alice  
slot2@... Claimed Alice  
slot3@... Claimed Alice  
slot4@... Unclaimed undefined  
slot5@... Claimed Bob  
slot6@... Claimed Bob  
slot7@... Claimed Charlie  
slot8@... Claimed Charlie
```

# 1: Get all slots in pool

```
$ condor_status -af Name RemoteOwner
```

**Slots**



## 2: Get all submitters in pool

```
$ condor_status -submitters
```

## 2: Get all submitters in pool

```
$ condor_status -submitters
```

Name	Machine	RunningJobs	IdleJobs
Alice	submit1	4	4
Bob	submit1	2	100
Charlie	submit1	2	0
Danny	submit1	0	50

## 2: Get all submitters in pool

```
$ condor_status -submitters
```

Name	Machine	RunningJobs	IdleJobs
Alice	submit1	4	4
Bob	submit1	2	100
<del>Charlie</del>	<del>submit1</del>	<del>2</del>	<del>0</del>
Danny	submit1	0	50

# 3: Compute per-user “share”

- › Tricky
- › Based on historical usage

# 3a: Get historical usage

```
$ condor_userprio -all
```

# 3a: Get historical usage

```
$ condor_userprio -all
```

UserName	Effective Priority	Real Priority	Priority Factor	Res in use
Alice	3100	3.1	1000	4
Bob	4200	4.2	1000	2
Charlie	1500	1.5	1000	2
Danny	8200	8.2	1000	0

# 3a: Get historical usage

$$\textit{EffectivePrio} = \textit{RealPrio} \times \textit{PrioFactor}$$

UserName	Effective Priority	Real Priority	Priority Factor	Res in use
Alice	3100	3.1	1000	4
Bob	4200	4.2	1000	2
Charlie	1500	1.5	1000	2
Danny	8200	8.2	1000	0

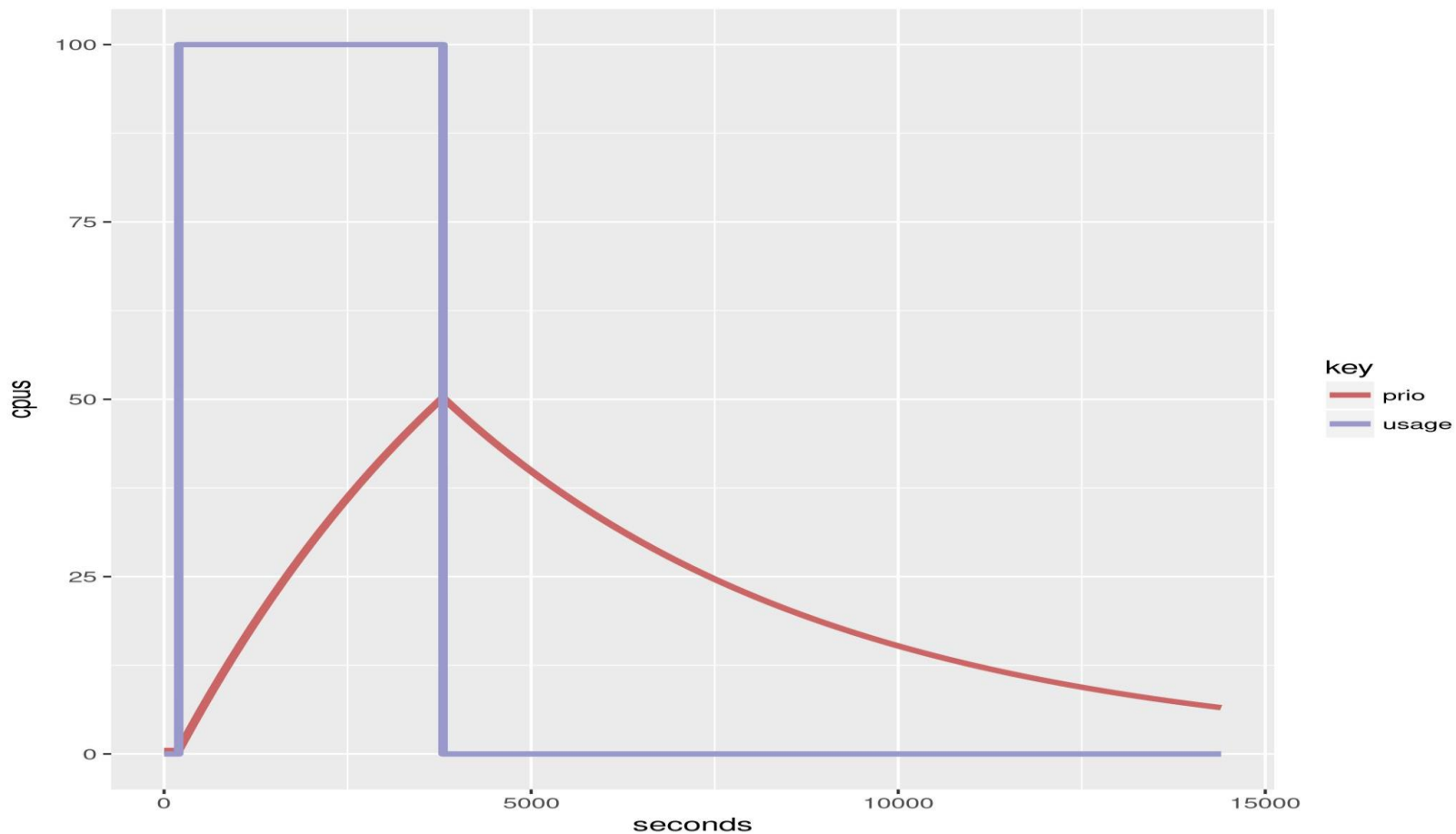
# So What is Real Priority?

Real Priority is smoothed historical usage

Smoothed by `PRIORITY_HALFLIFE`

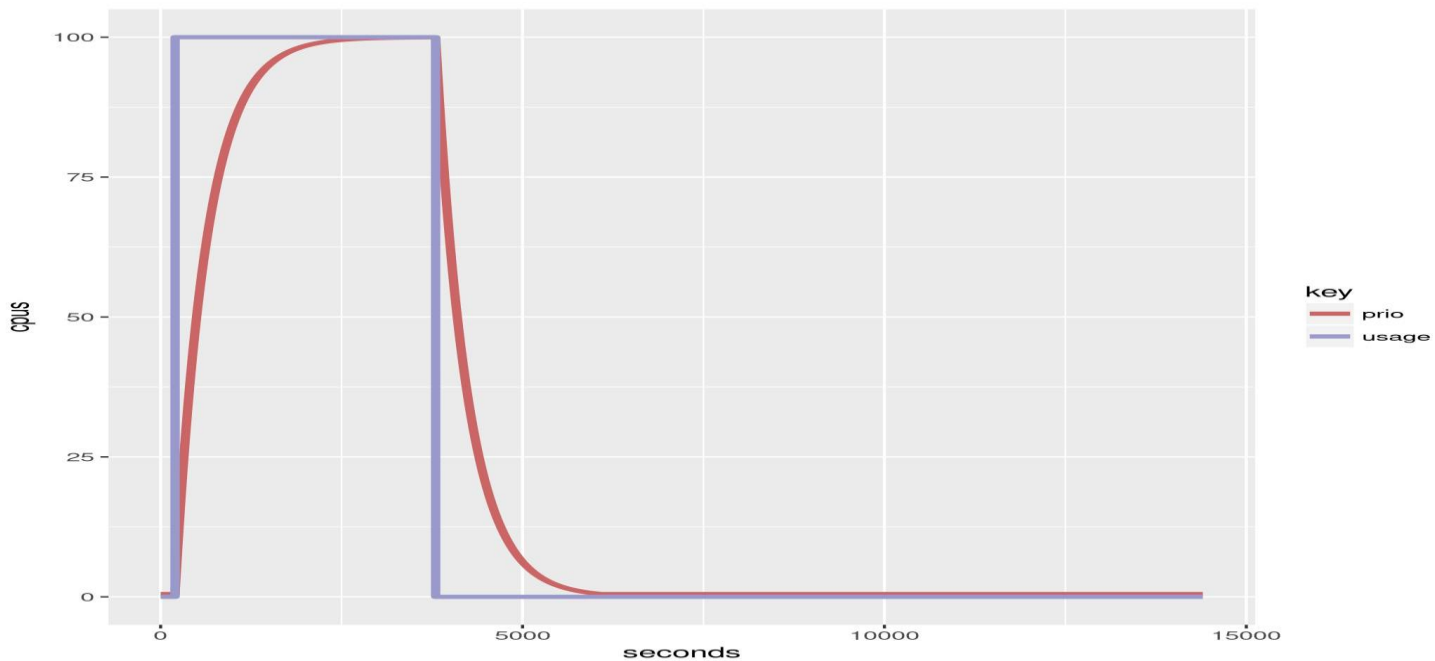
`PRIORITY_HALFLIFE` defaults 86400s (24h)

# Actual Use vs Real Priority



# Another PRIORITY\_HALFLIFE

PRIORITY\_HALFLIFE = 1



# 3a: Get historical usage

```
$ condor_userprio -all
```

UserName	Effective Priority	Real Priority	Priority Factor	Res in use
Alice	3100	3.1	1000	4
Bob	4200	4.2	1000	2
Charlie	1500	1.5	1000	2
Danny	8200	8.2	1000	0

# Effective priority:

- › Effective Priority is the *ratio* of the pool that the negotiator tries to allot to *users*

Lower is better, 0.5 is the best real priority

UserName	Effective Priority	Real Priority	Priority Factor	Res in use
Alice	1000	1.0	1000	4
Bob	2000	2.0	1000	2
Charlie	2000	2.0	1000	2

**Alice deserves 2x Bob & Charlie**

**Alice: 4**

**Bob: 2**

**Charlie: 2**

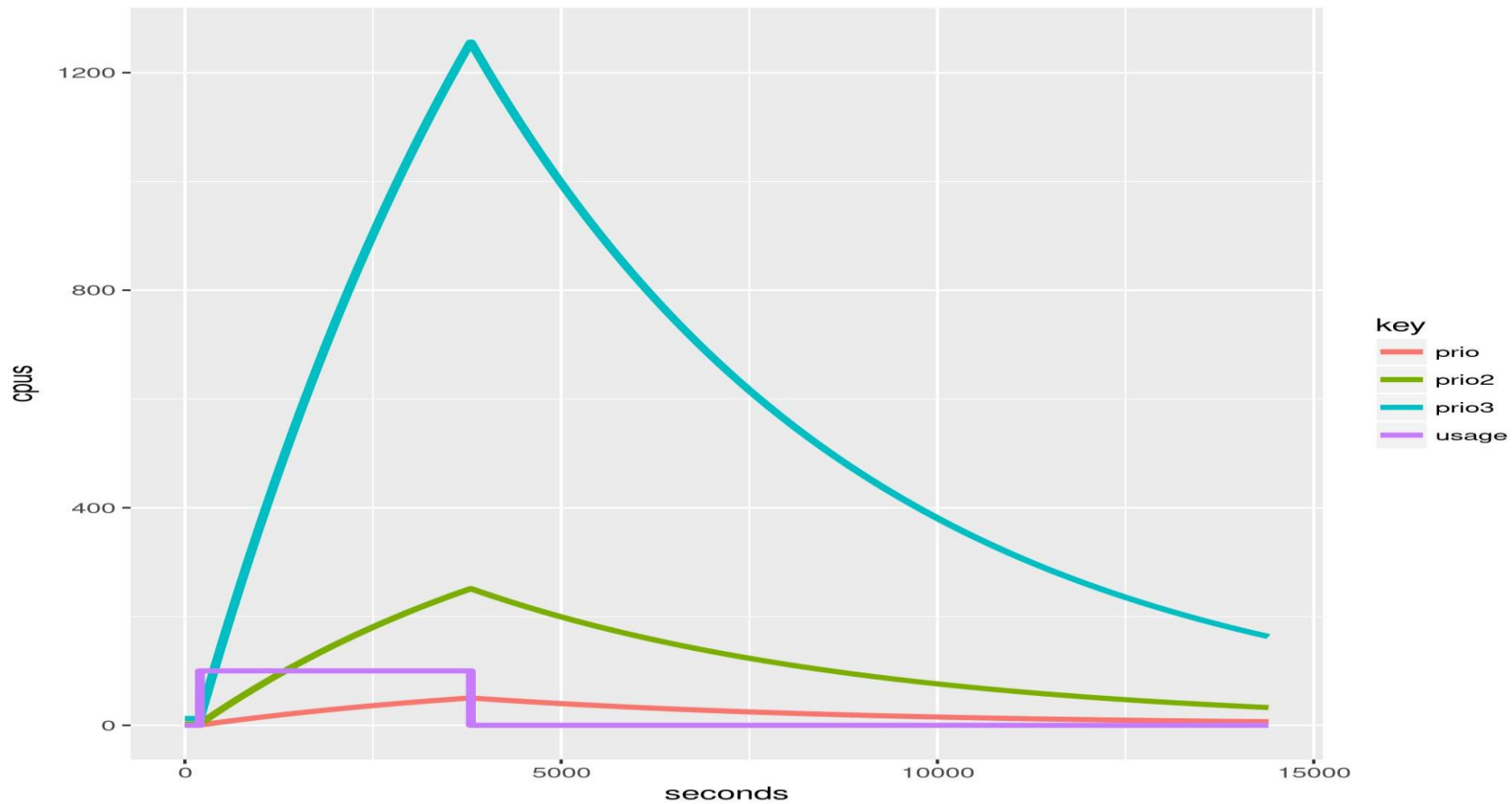
**(Assuming 8 total slots)**

UserName	Effective Priority	Real Priority	Priority Factor	Res in use
Alice	1000	1.0	1000	4
Bob	2000	2.0	1000	2
Charlie	2000	2.0	1000	2

Priority factor lets admin say  
If equal usage, User A gets  $1/n$ th User B

```
$ condor_userprio -setfactor alice 5000
```

# 3 different PrioFactors



# Whew! Back to negotiation

1. Get all slots in the pool
2. Get all ~~jobs~~ submitters in pool
3. Compute # of slots submitters should get
4. In priority order, hand out slots to submitters
5. Repeat as needed

# Target allocation from before

User	Effective Priority	Goal
Alice	1,000.00	4
Bob	2,000.00	2
Charlie	2,000.00	2

Assume 8 total slots (claimed or not)

# Look at current usage

User	Effective Priority	Goal	Current Usage
Alice	1,000.00	4	3
Bob	2,000.00	2	1
Charlie	2,000.00	2	0

# Diff the goal and reality

User	Effective Priority	Goal	Current Usage	Difference ("Limit")
Alice	1,000.00	4	3	1
Bob	2,000.00	2	1	1
Charlie	2,000.00	2	0	2

# “Submitter Limit” per user

User	Effective Priority	Goal	Current Usage	Difference (“Limit”)
Alice	1,000.00	4	3	1
Bob	2,000.00	2	1	1
Charlie	2,000.00	2	0	2

# Limits determined, matchmaking starts

In Effective User Priority order,

Find a schedd for that user, get the request

User	Effective Priority	Difference ("Limit")
Alice	1,000.00	1
Bob	2,000.00	1
Charlie	2,000.00	2

# “Requests”, not “jobs”

```
$ condor_q -autocluster Alice
```

Id	Count	Cpus	Memory	Requirements
20701	10	1	2000	OpSys == "Linux"
20702	20	2	1000	OpSys == "Windows"

# Match *all* machines to requests

```
Id          Count Cpus Memory Requirements
20701       10     1   2000  OpSys == "Linux"
```

```
slot1@... Linux X86_64 Idle 2048
slot2@... Linux X86_64 Idle 2048
slot1@... Linux X86_64 Idle 1024
slot2@... Linux X86_64 Claimed 2048
slot1@... WINDOWS X86_64 Claimed 1024
```

# Sort All matches

By 3 keys, in order

NEGOTIATOR\_PRE\_JOB\_RANK

RANK

NEGOTIATOR\_POST\_JOB\_RANK

# Why Three?

NEGOTIATOR\_PRE\_JOB\_RANK

**Strongest, goes first over job RANK**

RANK

**Allows User some say**

NEGOTIATOR\_POST\_JOB\_RANK

**Fallback default**

# Finally, give matches away!

```
slot1@... Linux X86_64 Unclaimed 2048
slot2@... Linux X86_64 Unclaimed 2048
slot1@... Linux X86_64 Claimed 2048
```

Up to the limit specified earlier

If below limit, ask for next job request

# Done with Alice, on to Bob

User	Effective Priority	Difference (“Limit”)
Alice	1,000.00	1
Bob	2,000.00	1
Charlie	2,000.00	2

# But, it isn't that simple...

- › Assumed every job matches every slot  
And infinite supply of jobs!
- › ... But what if they don't match?

There will be leftovers – then what?

# Lather, rinse, repeat

This whole cycle repeats with leftover slots

Again in same order...

# Big policy question

- › Preemption: Yes or no?
- › Tradeoff: fairness vs. throughput
- › (default: no preemption)

# Preemption: disabled by default

```
PREEMPTION_REQUIREMENTS = false
```

Evaluated with slot & request ad. If true,  
Claimed slot is considered matched, and  
Subject to matching

# Example PREEMPTION\_REQs

```
PREEMPTION_REQUIREMENTS=\
```

```
RemoteUserPrio > SubmitterPrio * 1.2
```

# PREEMPTION\_RANK

- › Sorts matched preempting claims

PREEMPTION\_RANK = -TotalJobRunTime

# MaxJobRetirementTime

- › Can be used to guarantee minimum time
- › E.g. if claimed, give an hour runtime, no matter what:
  
- ›  $\text{MaxJobRetirementTime} = 3600$
- › Can also be an expression

# Whew!

› Now, on to Groups.

# First AccountingGroup

- › AccountingGroup as alias
- › Accounting\_Group\_User = Ishmael
- › **“Call me Ishmael”**
- › With no dots, and no other configuration
- › Means alias: Maps “user” to “submitter”
- › Complete trust in user job ad (or xform)
  - Viz-a-vis SUBMIT\_REQUIREMENTS

User	Effective Priority	Accounting Group
Alice	1,000.00	"Alice"
Bob	2,000.00	"Alice"
Charlie	2,000.00	



Merged to one user

No fair share between old Alice and old Bob!

# Accounting Groups With Quota

Only way to get “quotas” for users or groups

# quota, n.

View as: Outline | [Full entry](#)

**Pronunciation:** Brit. /'kwɒtə/, U.S. /'kwɒdə/

**Forms:**

α. 16– **quota**, 16– **quoto** (chiefly *U.S. regional*), 17 **cotta**, 17 **qotta**.

... (Show More)

**Etymology:** < post-classical Latin *quota*... (Show More)

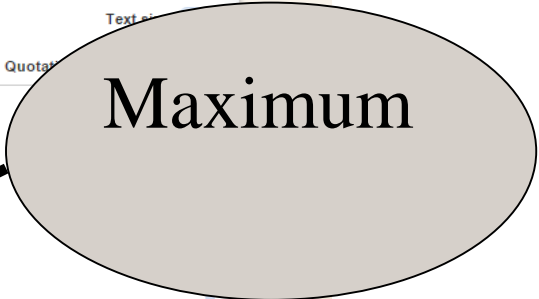
**1.**

**a.** Originally: the part or share which an individual is obliged to contribute to a total amount (in early use chiefly with reference to contributions of men, money, or supplies from a particular town, district, or country; cf. **CONTINGENT** *n.* 5). Later more widely: an amount contributed to a larger quantity.

1618–1968

(Show quotations)

**b.** *Econ.* A maximum quantity of a particular product which under official controls can be produced, exported, imported, or caught. Also: a target setting a minimum production for a particular factory, employee, etc.



quota act  
quota bill  
quota bou  
quota film  
quota-ho  
quota imr  
quota imr  
quota law  
quota lim  
quota-ma  
quota me  
quota per  
quota pla  
quota qui

Thesaurus »

Thesaurus »  
Categories »

contribution towards diocesan expenditure (more fully *diocesan* (formerly also *parochial*) quota).

1911–1995

(Show quotations)

2.

a. A share of a larger number or quantity; a portion, an allocation.

1688–1996

b. *Polit.* In a system of proportional representation: the minimum number of votes required to elect a candidate.

1857–2006

3. Chiefly *U.S.*

a. A maximum number of immigrants allowed to enter a country within a set period. Also: a maximum number of students (as of a particular racial or ethnic group) allowed to enrol for a course at a college, etc., in a particular year.

The Emergency Quota Act was passed by the U.S. Congress in 1921.

1921–2002

(Show quotations)

b. A minimum number or proportion (of racial or ethnic minorities, or women) sought in order to ensure a desired balance in a workforce, student body, etc.

1956–2005

(Show quotations)



Minimum

# Group Quotas: Big Picture

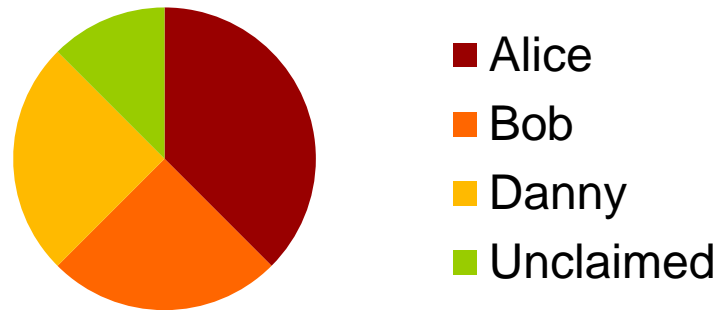
w/o quotas: Assign submitters

Name	Machine	RunningJobs	IdleJobs
Alice	submit1	4	4
Bob	submit1	2	100
Charlie	submit1	2	0
Danny	submit1	0	50

# Group Quotas: Big Picture




To the whole pool

**Slots in whole pool**



Name	Machine	RunningJobs	IdleJobs
Alice	submit1	4	4
Bob	submit1	2	100
Charlie	submit1	2	0
Danny	submit1	0	50

# Submitters opt into Groups

Name	Machine	RunningJobs	Idle	
Alice	submit1	4	4	 Group A
Bob	submit1	2	100	 Group B
Charlie	submit1	2	0	
Danny	submit1	0	50	 Group C

# 1st: Each group gets a “Quota”

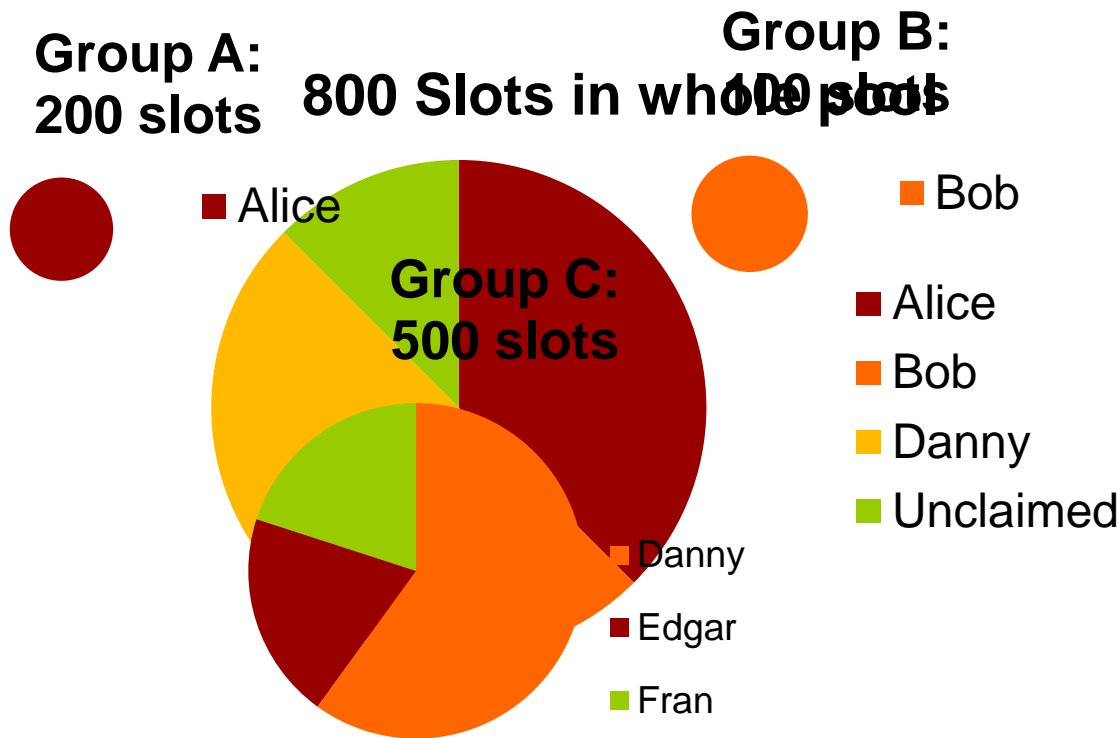
Name	Machine	RunningJobs	Idle	
Alice	submit1	4	4	Group A
Bob	submit1	2	100	Group B
Charlie	submit1	2	0	
Danny	submit1	0	50	Group C

Group	Quota
Group A	200 slots
Group B	100 slots
Group C	500 slots

(How? We'll get to that)

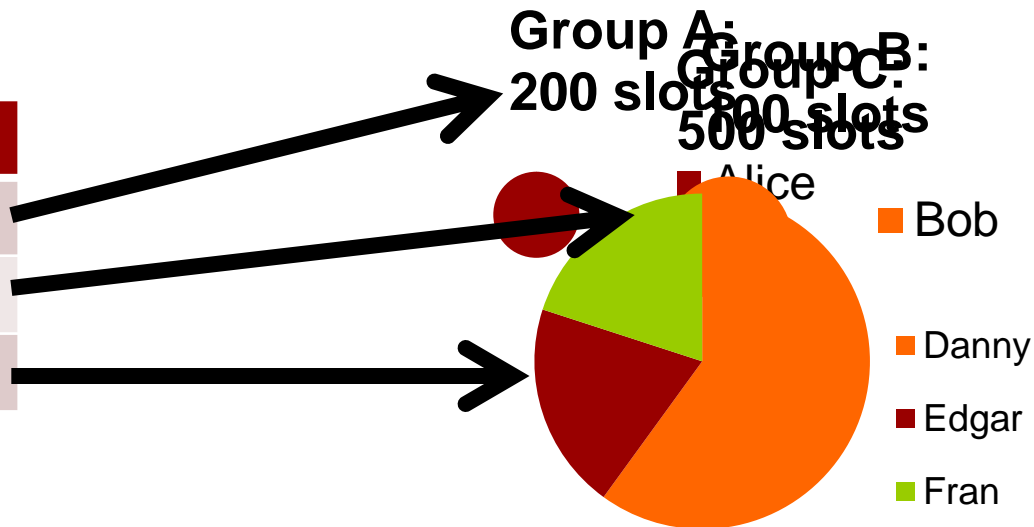
# 2<sup>nd</sup>: Make virtual sub-pool per group

Group	Quota
Group A	200 slots
Group B	100 slots
Group C	500 slots



# 3<sup>rd</sup>: Do fair share with each sub-pool in group turn

Group	Quota
Group A	200 slots
Group B	100 slots
Group C	500 slots



Groups with  $> 1$  submitter get fair share with prios as usual, but total size of the pool is the quota size

# Accounting Groups with quotas

- › Must be predefined in config file

```
GROUP_NAMES = group_a, group_b
```

```
GROUP_QUOTA_GROUP_A = 10
```

```
GROUP_QUOTA_GROUP_B = 20
```

Slot weight is the unit – default cpus

# Or, with Dynamic quotas

- › Can also be a percentage

```
GROUP_NAMES = group_a, group_b
```

```
GROUP_QUOTA_DYNAMIC_GROUP_A = 0.3
```

```
GROUP_QUOTA_DYNAMIC_GROUP_B = 0.4
```

If sum != 100, scaled

# And jobs opt in (again)

```
Accounting_Group      = group_a
```

But you retain identity within your group.

# AcctGroups w/quota

- › Reruns the whole cycle as before
  - But with pool size constrained to quota
  - And fair share, between users in group

# Order of groups?

- › By default, in **starvation** order
- › Creates overprovisioning trick for strict fifo:
  - › `GROUP_QUOTA_HIPRIO = 100000000`
- › Means this group always most starving
  - › `GROUP_SORT_EXPR` overrides

# Quotas can leave slots idle

- › If Group's demand  $<$  quota
- › Slots left idle
- › Can we go over quota in this case?

# Going over quota, slots available

One way is:

```
GROUP_AUTO_REGROUP = true
```

After all groups go, one last round with no groups, every user outside of their group.

# 2<sup>nd</sup> way to over quota

› “Surplus”

› Assumes a hierarchy of groups:

```
GROUP_NAMES = group_root, group_root.a, group_root.b,  
group_root.c
```

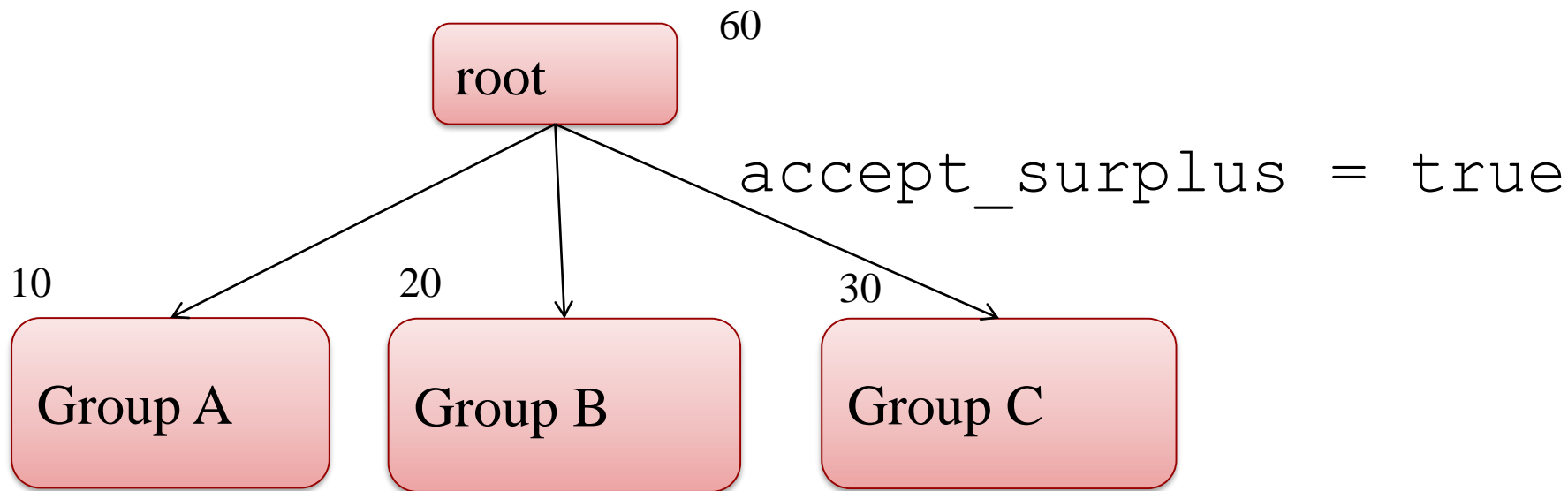
```
GROUP_QUOTA_GROUP_root = 60
```

```
GROUP_QUOTA_GROUP_root.a = 10
```

```
GROUP_QUOTA_GROUP_root.b = 20
```

```
GROUP_QUOTA_GROUP_root.b = 30
```

```
GROUP_ACCEPT_SURPLUS = true
```

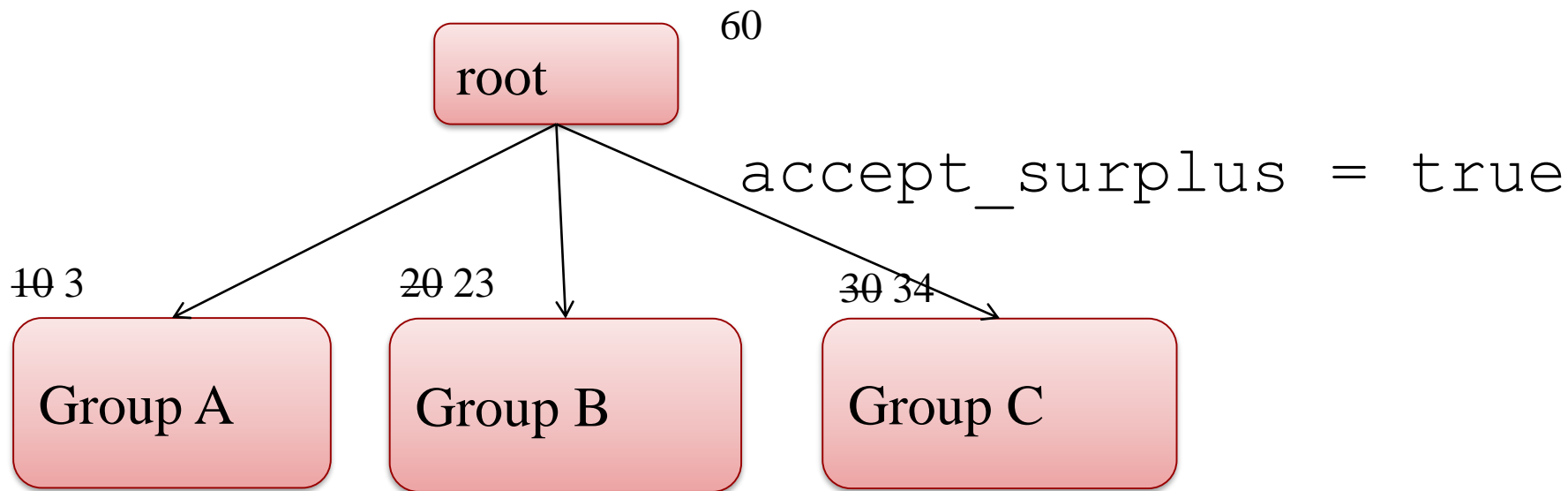


3 slots of demand at A



7 quota slots moved to B & C

Proportional to B & C quota



# Gotchas with quotas

- › Quotas don't know about matching
- › Assuming everything matches everything
- › Surprises with partitionable slots
- › Managing groups not easy

# In summary

- › Negotiator is very powerful, often ignored
- › Lots of opportunity to tune system
- › Many ways to peak under the hood

# Thank you

- › Questions?
- › Talk to us
- › htcondor-users
- › manual