Fair Tree

Fairshare Algorithm for Slurm

Ryan Cox and Levi Morrison
Brigham Young University

Slurm User Group Meeting – September 24, 2014 – Lugano, Switzerland

Outline

- Introduction to job prioritization
- BYU's setup
- Issues with existing fairshare algorithms
- Fair Tree
- Fairshare=parent on accounts
- Appendix

Job Prioritization

- Job Priority can include different components:
  - Age
  - JobSize
  - Partition
  - QOS
  - **Fairshare**

Fairshare Factor

- Usage and Shares are the two components of Fairshare Factor

Fairshare Factor

- Usage and Shares are the two components of Fairshare Factor
- Raw Shares are assigned to each association by an admin
  - “Shares” is Raw Shares when normalized to 0.0 .. 1.0
  - Similar to slices of a pie
  - Represents the part of the system that is “yours”

Fairshare Factor

- Usage and Shares are the two components of Fairshare Factor
- Raw Shares are assigned to each association by an admin
  - “Shares” is Raw Shares when normalized to 0.0 .. 1.0
  - Similar to slices of a pie
  - Represents the part of the system that is “yours”

- “Usage” is a value between 0.0 and 1.0 that represents your proportional usage of the system

Fairshare Equation

Basic premise of fairshare:

If Shares == Usage, you have hit your “fairshare target”

Fairshare Equation

$$\text{Fairshare Factor} = 2^{-\text{Usage/Shares}}$$

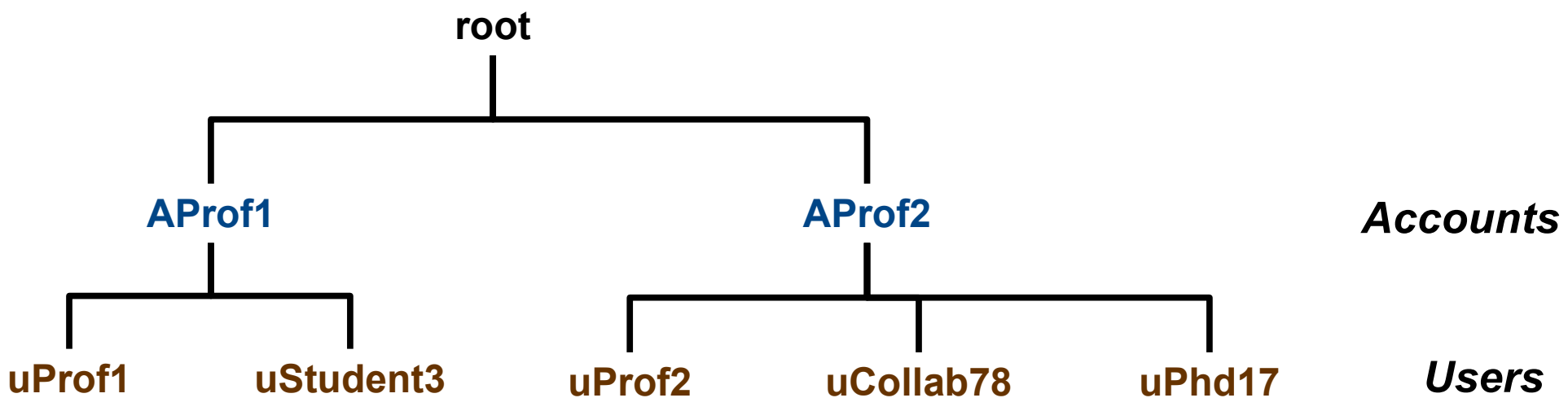
or

$$F = 2^{-U/S}$$

- Result is between 0.0 and 1.0
- When Usage increases, F decreases
- When Usage decreases, F increases

BYU's Setup

- Each professor has a Slurm account
  - Has *account coordinator* status for that account
  - BYU doesn't charge $$$ for usage
  - Accounts are treated equally (Shares are the same)
- Students and collaborators are users in a professor's account
  - Users are treated equally (Shares are the same, unless account coordinator changes them)



Our Definition of “Fair”

- **If accounts A and B are siblings and A has a higher fairshare factor than B, all children of A will have higher fairshare factors than all children of B**
  - True for all sibling associations at all levels

Disclaimer

- We will discuss BYU's use case and how to support it
  - It may also apply to many other use cases
- We are building on the hard work of others
- We identified some issues with existing algorithms but our work would not have been possible without the other algorithms

New Fairshare Algorithm

We wrote a new fairshare algorithm.

Why?

The Problem

Account with 35% usage had higher priority than accounts with only 10% usage

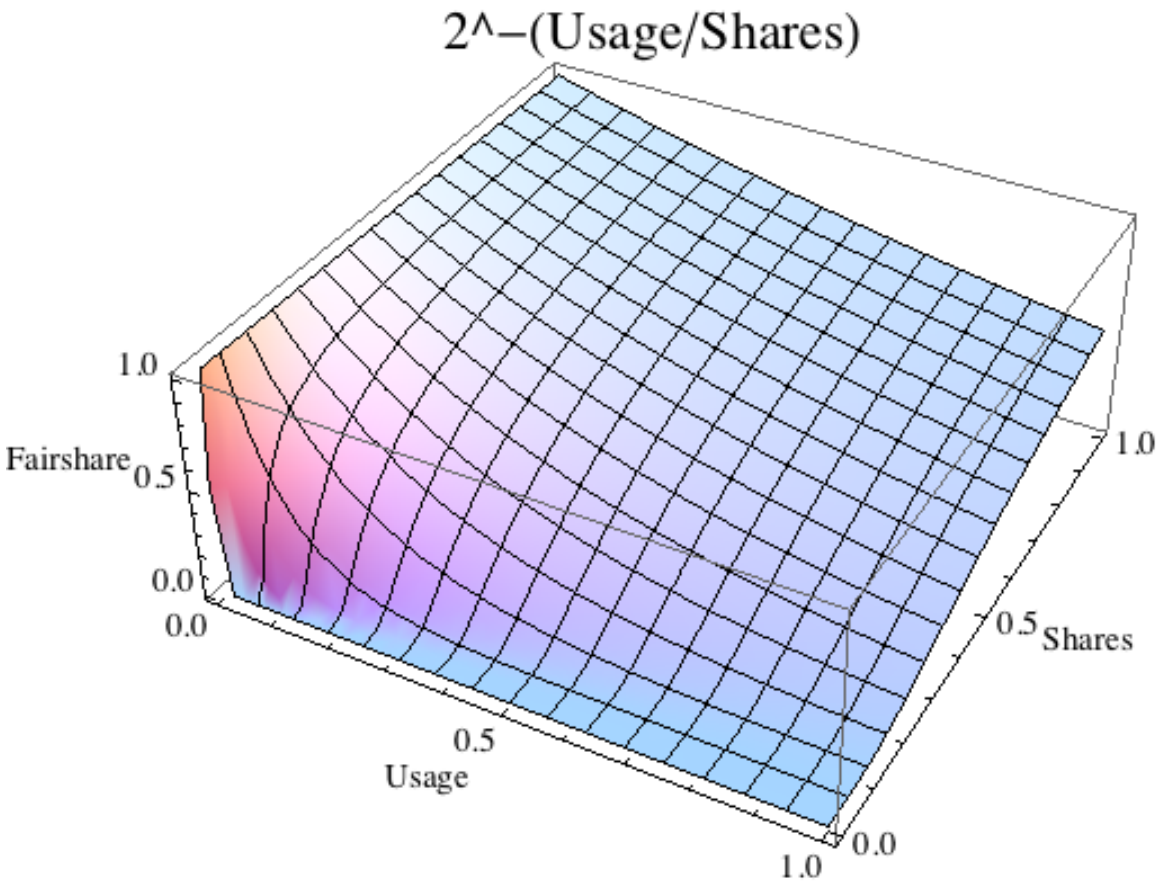
Fairshare Equation

$$\text{Fairshare Factor} = 2^{-\text{Usage/Shares}}$$

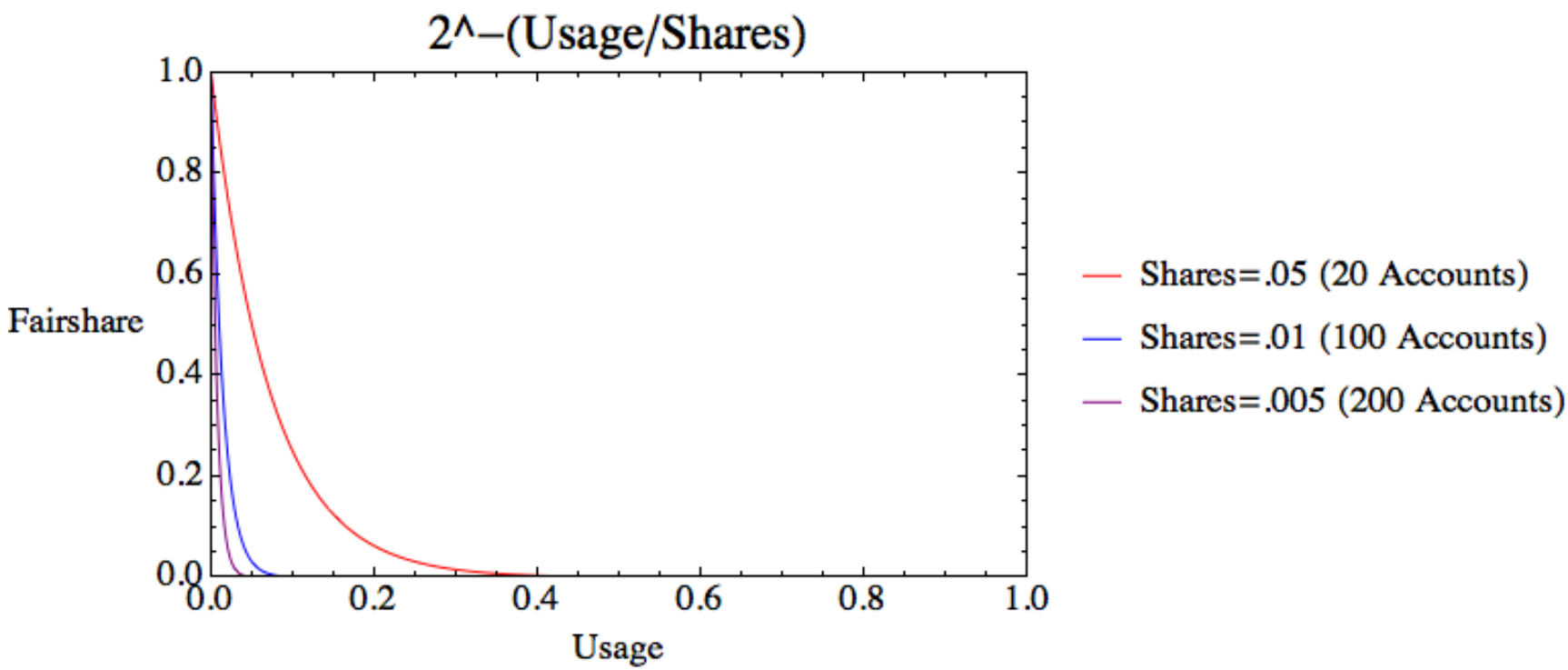
or

$$F = 2^{-U/S}$$

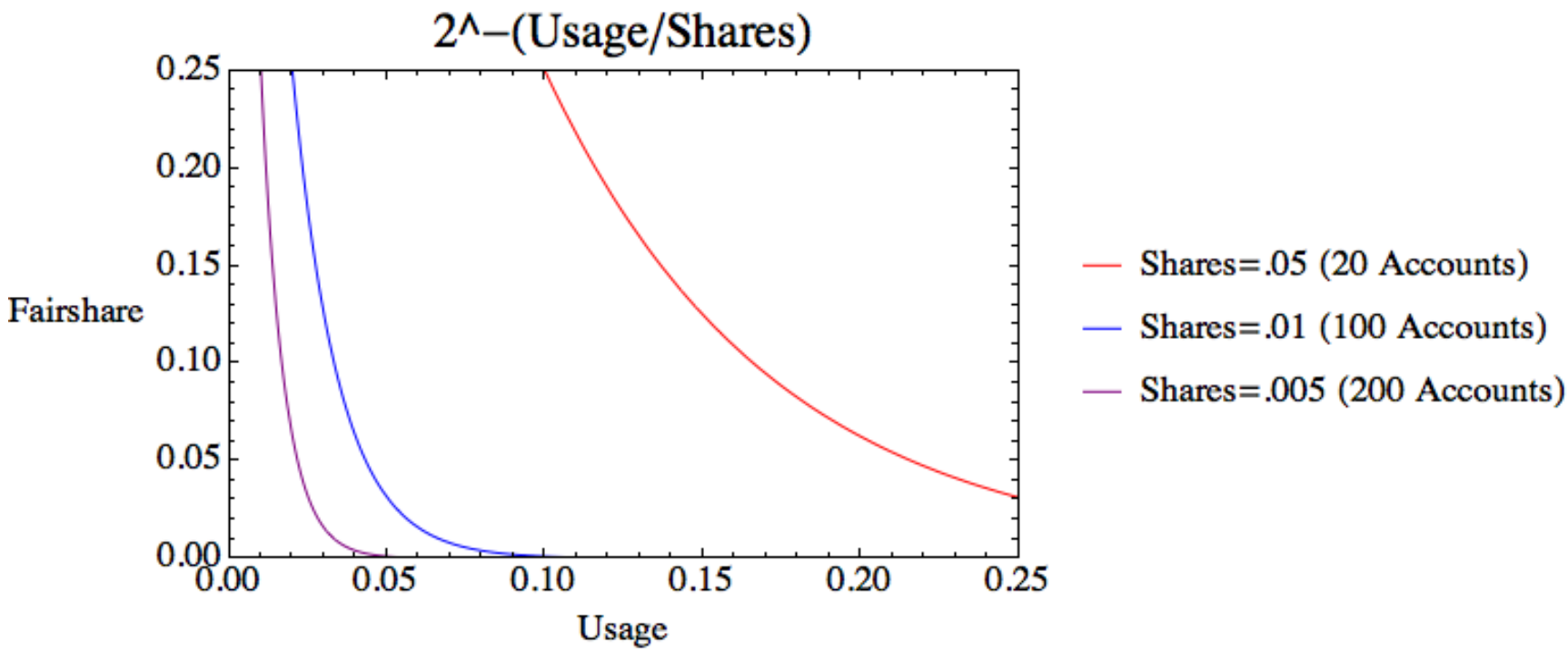
Fairshare 3D Graph



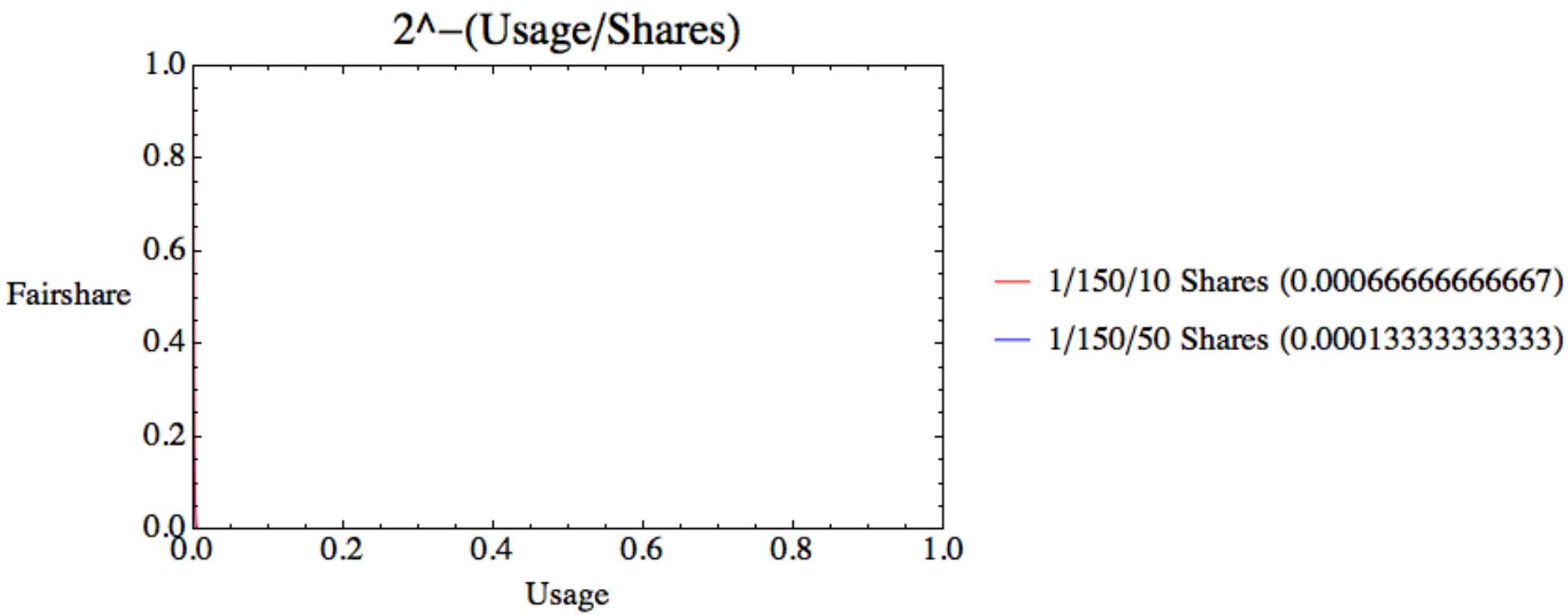
Fairshare Graph



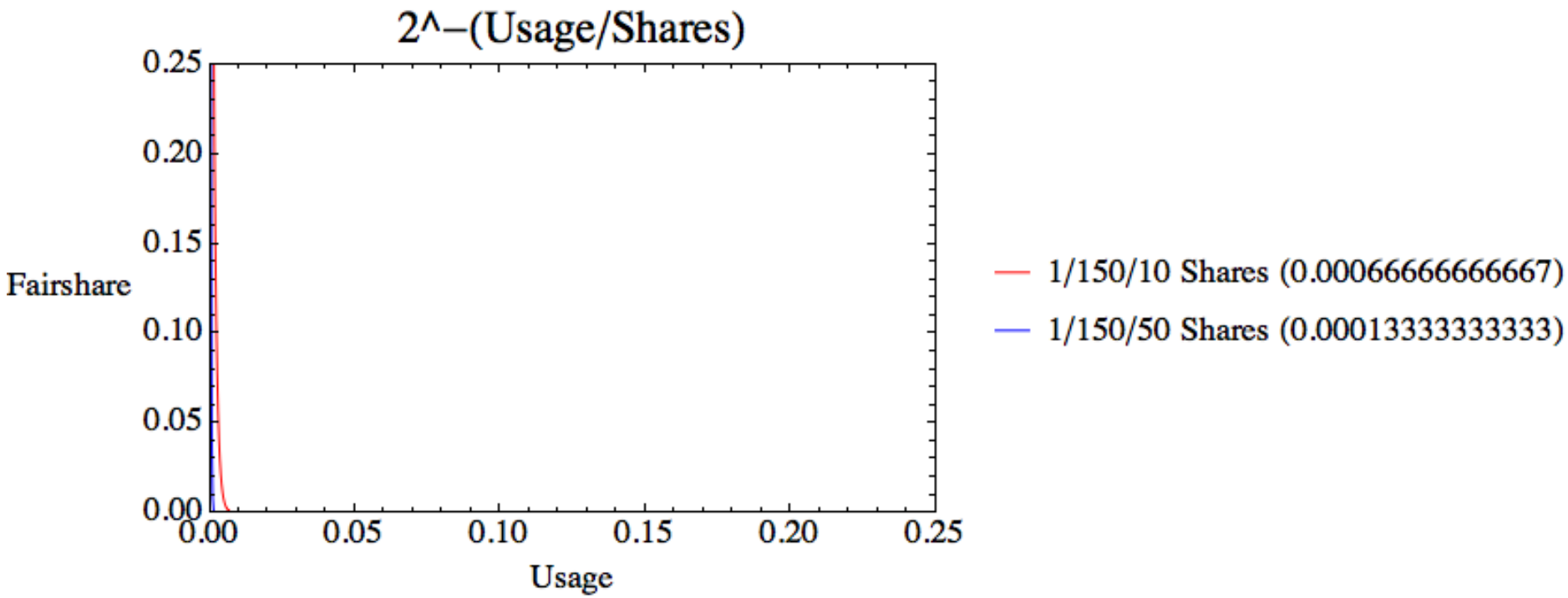
Fairshare Graph Zoomed



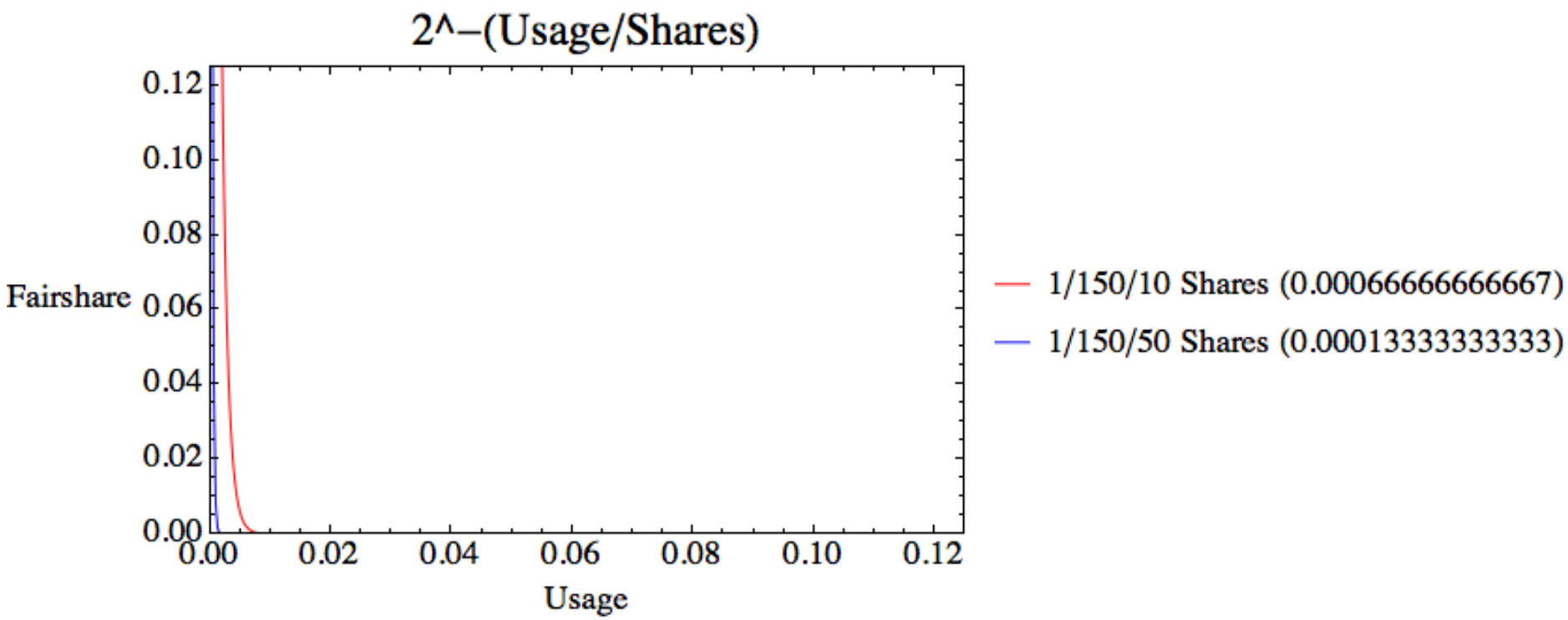
Actual Fairshare Graph



Actual Fairshare Zoomed



Actual Fairshare More Zoomed



Dampening Factor

- Added in 14.03
- 2^-((Usage/Shares)/Damp)
  - Damp is a positive integer (slurm.conf)
- Great idea
- Isn't a complete solution
  - Small values may still be too small
  - Requires manual guess and check
- Can be replaced by linear interpolation
  - Ticket-Based can use this instead of MAX()
  - See Appendix

Problems With Shares Calculation

- Normalized Shares equation (the "S" in `2^-(U/S)`) for all current algorithms:

  $S = (RawShares_{user} / RawShares_{siblings}) *$

  $(RawShares_{account} / RawShares_{sibling\text{-}accounts}) *$

  $(RawShares_{parent} / RawShares_{parent\text{-}siblings}) * \ldots$

Problems With Shares Calculation

- Normalized Shares equation (the "S" in `2^-(U/S)`) for all current algorithms:

$S = (RawShares_{user} / RawShares_{siblings}) *$ 1/4 *

$(RawShares_{account} / RawShares_{sibling\text{-}accounts}) *$ 1/10 *

$(RawShares_{parent} / RawShares_{parent\text{-}siblings}) * \ldots$ 1/20 * ...

1/4 * 1/10 * 1/20 = 1/800 = 0.00125

Problems With Shares Calculation

- Normalized Shares equation (the "S" in 2^-(U/S)) for all current algorithms:

  $S = (RawShares_{user} / RawShares_{siblings}) * \quad 1/4 *$

  $(RawShares_{account} / RawShares_{sibling\text{-}accounts}) * \quad 1/10 *$

  $(RawShares_{parent} / RawShares_{parent\text{-}siblings}) * \ldots \quad 1/20 * \ldots$

  $1/4 * 1/10 * 1/20 = 1/800 = 0.00125$

- Assume sibling associations have the same Raw Shares, tree has 2 levels:

Problems With Shares Calculation

- Normalized Shares equation (the "S" in `2^-(U/S)`) for all current algorithms:

  $S = (RawShares_{user} / RawShares_{siblings}) * \quad 1/4 *$

  $(RawShares_{account} / RawShares_{sibling\text{-}accounts}) * \quad 1/10 *$

  $(RawShares_{parent} / RawShares_{parent\text{-}siblings}) * \ldots \quad 1/20 * \ldots$

  $1/4 * 1/10 * 1/20 = 1/800 = 0.00125$

- Assume sibling associations have the same Raw Shares, tree has 2 levels:

  Equivalent to:

  S = (1 / number of users in account) * constant

Problems With Shares Calculation

- Normalized Shares equation (the "S" in 2^-(U/S)) for all current algorithms:

  $S = (RawShares_{user} / RawShares_{siblings}) * \quad 1/4 *$

  $(RawShares_{account} / RawShares_{sibling\text{-}accounts}) * \quad 1/10 *$

  $(RawShares_{parent} / RawShares_{parent\text{-}siblings}) * \ldots \quad 1/20 * \ldots$

  1/4 * 1/10 * 1/20 = 1/800 = 0.00125

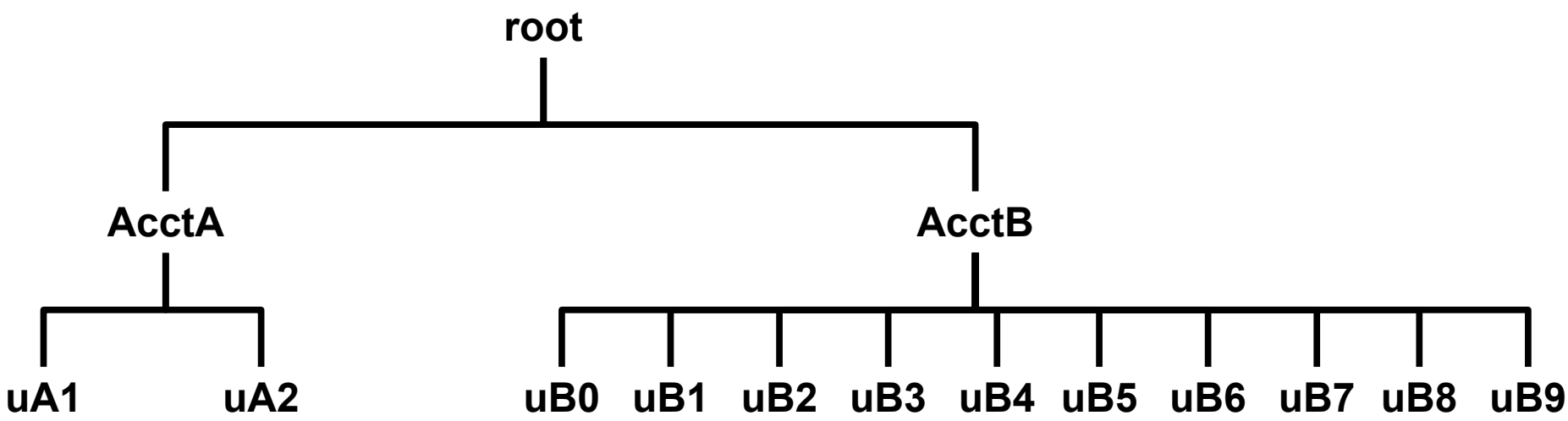
- Assume sibling associations have the same Raw Shares, tree has 2 levels:

  Equivalent to:

  S = (1 / number of users in account) * constant

- See appendix for demonstration of equivalence

Problems With Shares Calculation



S = 1/2 = 0.5

S = 1/10 = 0.1

Problem applies to more complicated scenarios but is harder to model

Usage Effective

- Usage Effective (the U in 2^-(U/S)) =

  $UA_{child}+((UE_{parent}-UA_{child})*S_{child}/S_{all\_siblings})$

Usage Effective

- Usage Effective (the `U` in `2^-(U/S)`) =

  $UA_{child}+((UE_{parent}-UA_{child})*S_{child}/S_{all\_siblings})$

- Assuming two levels in the tree, `2^-(U/S)` expands to:

$$2^{-\left(\frac{\frac{U_{User}}{U_{Total}}+\left[\frac{U_{Acct}}{U_{Total}}+\left(1-\frac{U_{Acct}}{U_{Total}}\right)*\frac{S_{Acct}}{S_{Acct}+S_{ASiblings}}-\frac{U_{User}}{U_{Total}}\right]*\frac{S_{User}}{S_{User}+S_{USiblings}}}{\frac{S_{User}}{S_{User}+S_{USiblings}}*\frac{S_{Acct}}{S_{Acct}+S_{ASiblings}}}\right)}$$

Usage Effective

- Usage Effective (the `U` in `2^-(U/S)`) =

  UA_{child}+((UE_{parent}-UA_{child})*S_{child}/$S_{all\_siblings}$)

- Assuming two levels in the tree, `2^-(U/S)` expands to:

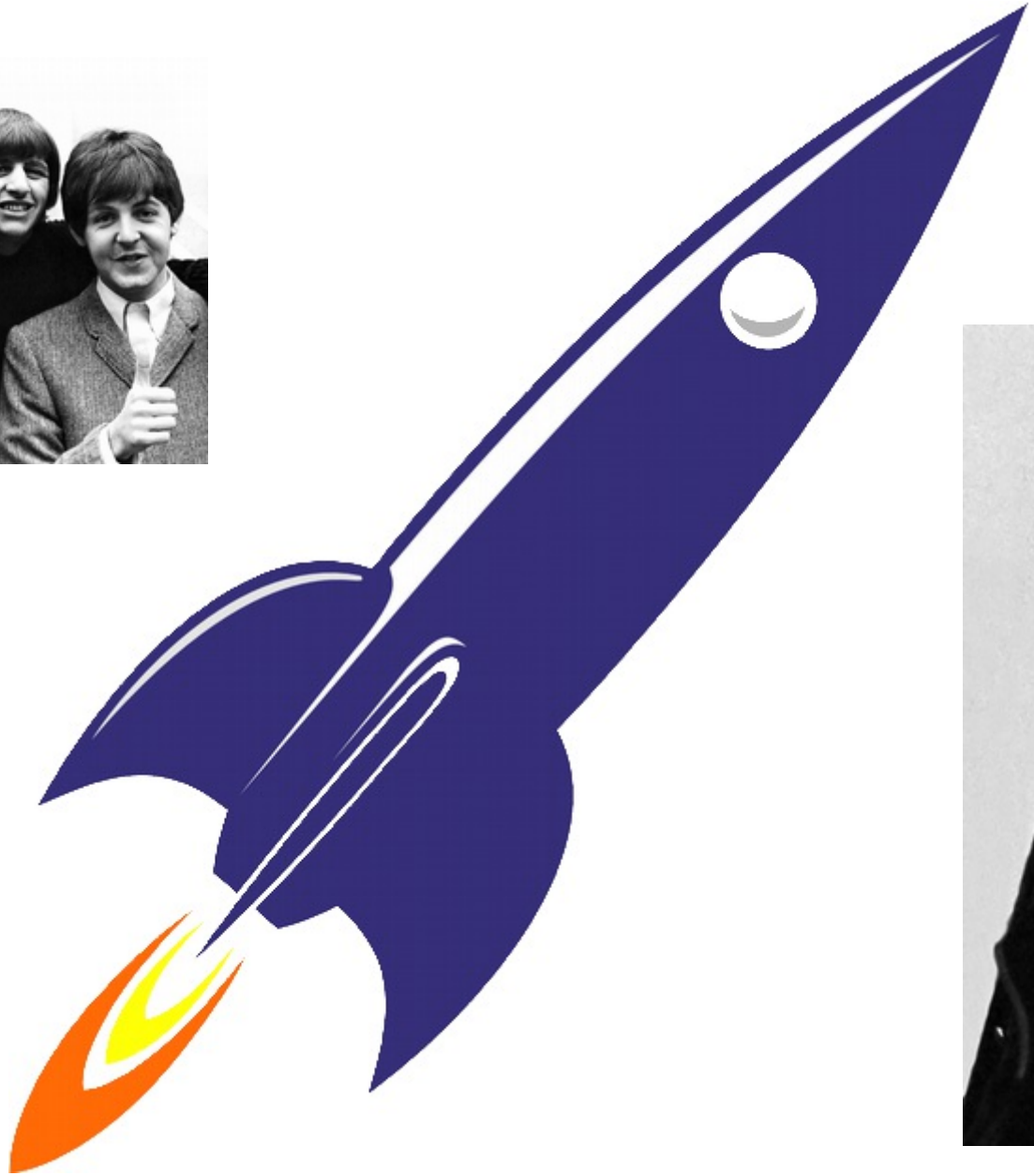
$$2^{-\left(\frac{\frac{U_{User}}{U_{Total}}+\left[\frac{U_{Acct}}{U_{Total}}+\left(1-\frac{U_{Acct}}{U_{Total}}\right)*\frac{S_{Acct}}{S_{Acct}+S_{ASiblings}}-\frac{U_{User}}{U_{Total}}\right]*\frac{S_{User}}{S_{User}+S_{USiblings}}}{\frac{S_{User}}{S_{User}+S_{USiblings}}*\frac{S_{Acct}}{S_{Acct}+S_{ASiblings}}}\right)}$$

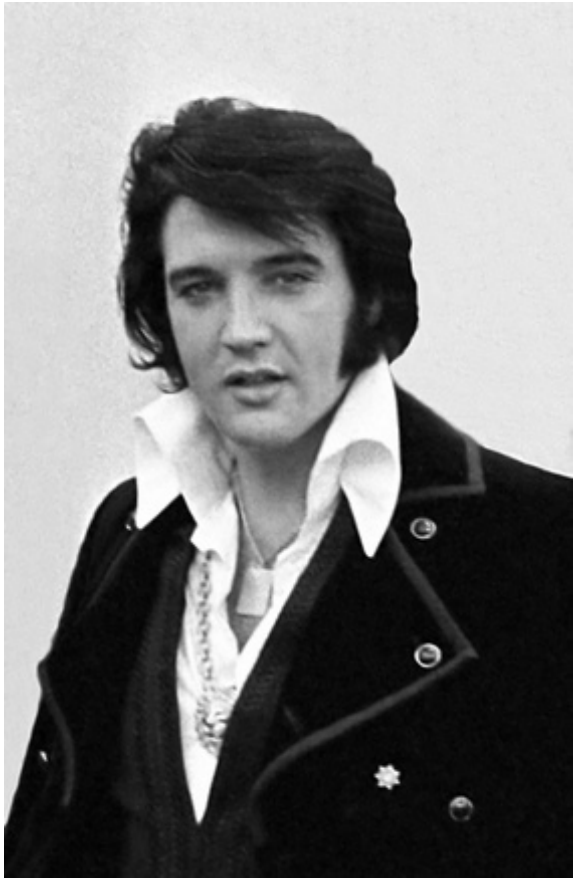
- Has some unpleasant side effects that we consider bugs
  - (Have 20 minutes and a whiteboard? Let's talk!)

Other Algorithms

- Ticket-Based
  - Depends on queue state
    - Hard to explain to users
  - $T = T_{parent} * S * F / SUM(S*F)_{active\_siblings}$
    - Similar issues to Norm Shares equation because ($S * F / SUM(S*F)_{active\_siblings}$) depends on the number of active users in an account
- Depth Oblivious
  - Very complex math
    - Hard to explain to users and admins
    - Hard to evaluate its fairness
  - Many opportunities for floating point precision issues

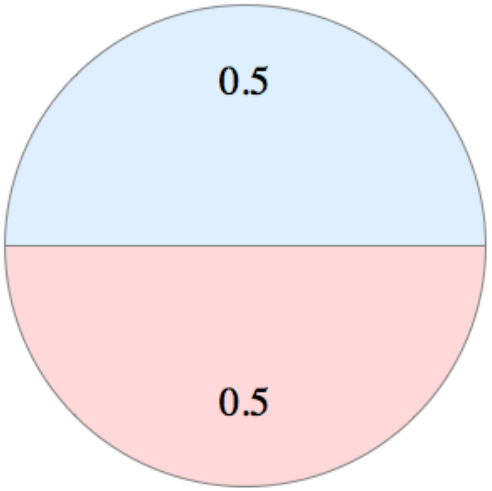
An Example



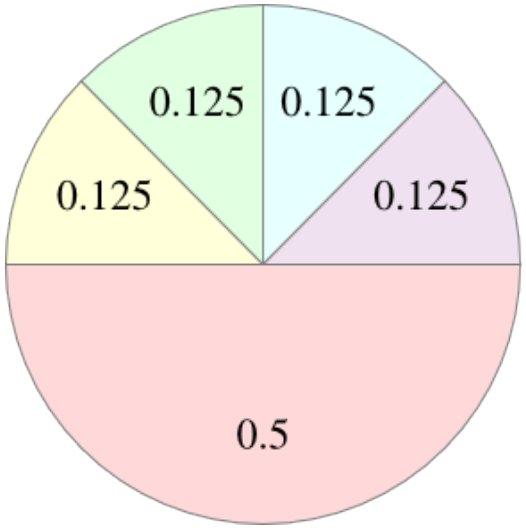


Timeshare on a Spaceship

- The Beatles and Elvis split the cost of a spaceship and crew
- One passenger spaceship
- Elvis pays 50%
- Beatles pay 50%
  - Each member pays 25% of the Beatles' 50% (equals 1/8 of the total)
  - **If a band member isn't waiting to use the spaceship, other band members can take a ride**



Accounts



Users

Lennon (Account=Beatles)
McCartney (Account=Beatles)
Harrison (Account=Beatles)
Starr (Account=Beatles)
Elvis (Account=Elvis)

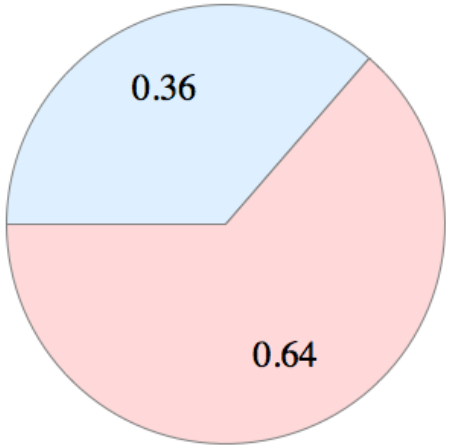
How well does Slurm perform in this scenario?

What actually happens?

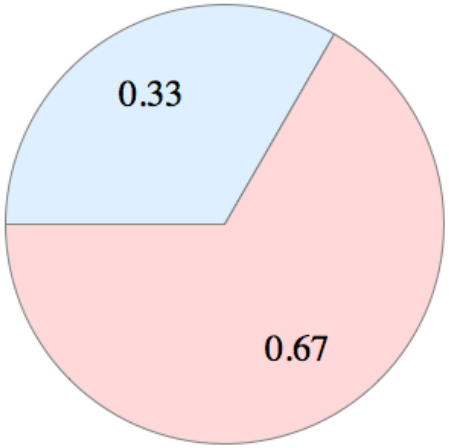
Simulated using the following account hierarchy:

```
        Account        User   Norm Shares
--------------- ----------  ------------
root                              0.000000
 beatles                          0.500000
  beatles          harrison       0.125000
  beatles             lennon      0.125000
  beatles          mccartney      0.125000
  beatles              starr      0.125000
 elvis                            0.500000
  elvis               elvis       0.500000
```

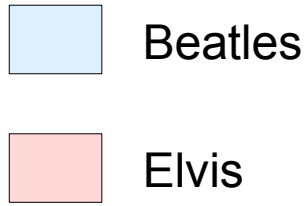
Traditional Multifactor

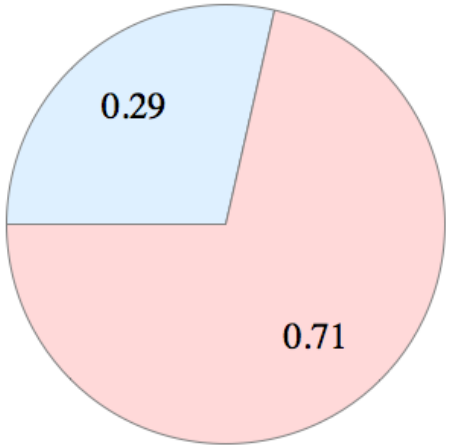


4 Beatles vs Elvis

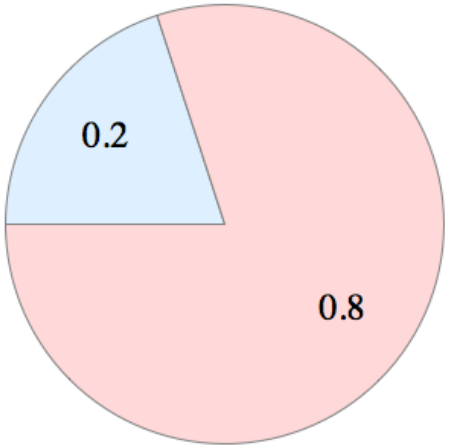


3 Beatles vs Elvis



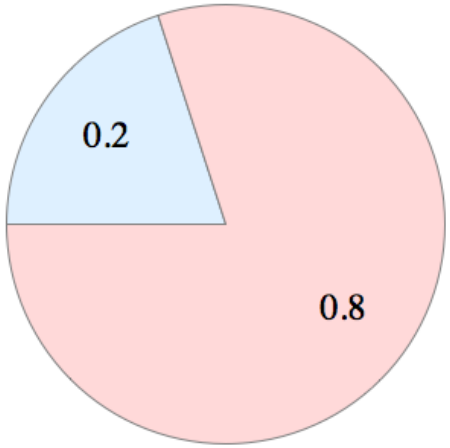


2 Beatles vs Elvis

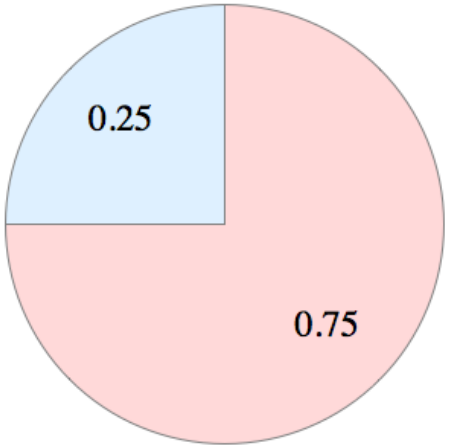


1 Beatle vs Elvis

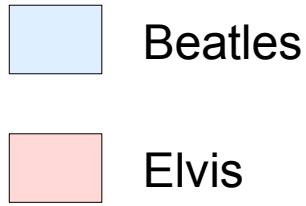
Ticket-Based

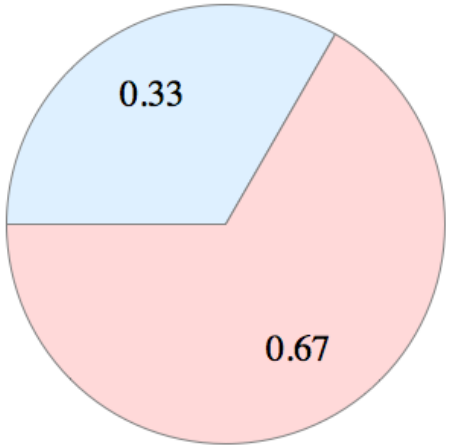


4 Beatles vs Elvis

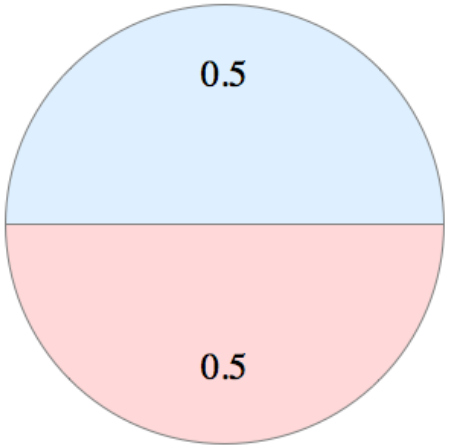


3 Beatles vs Elvis



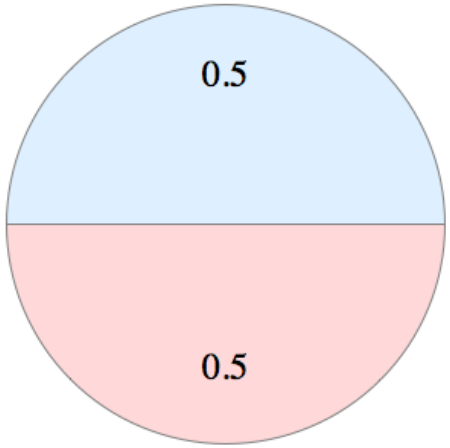


2 Beatles vs Elvis

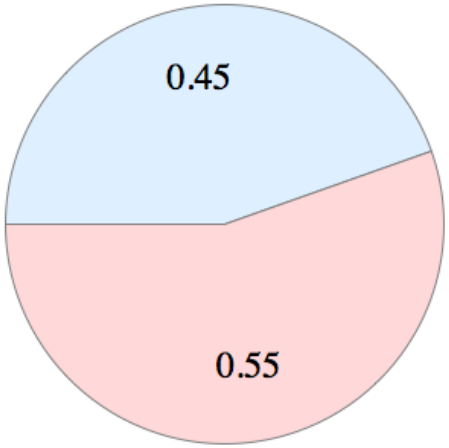


1 Beatle vs Elvis

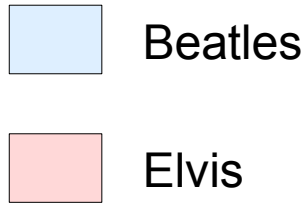
Depth Oblivious

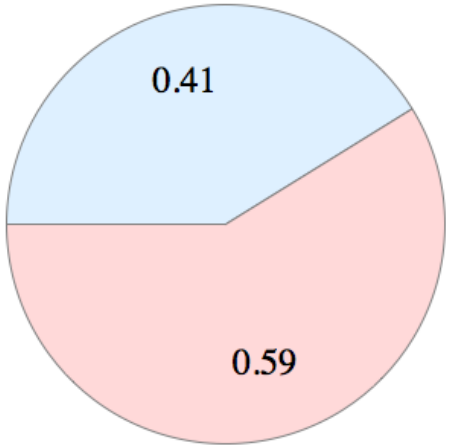


4 Beatles vs Elvis

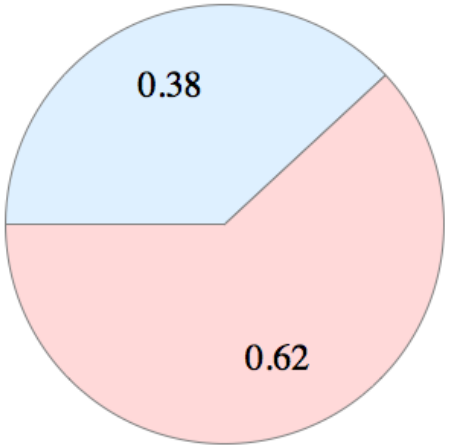


3 Beatles vs Elvis



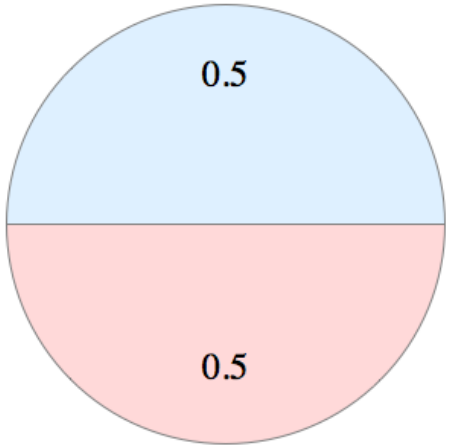


2 Beatles vs Elvis

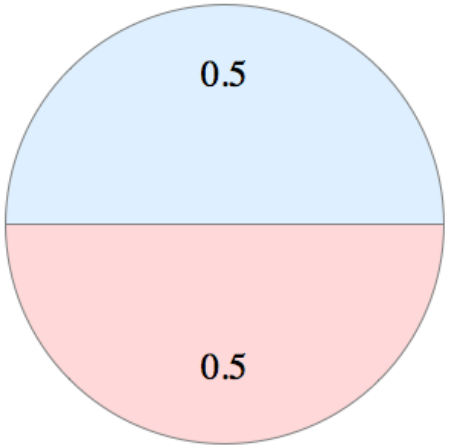


1 Beatle vs Elvis

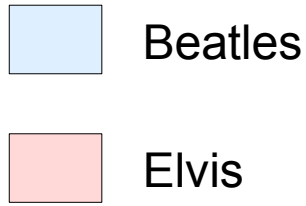
Fair Tree

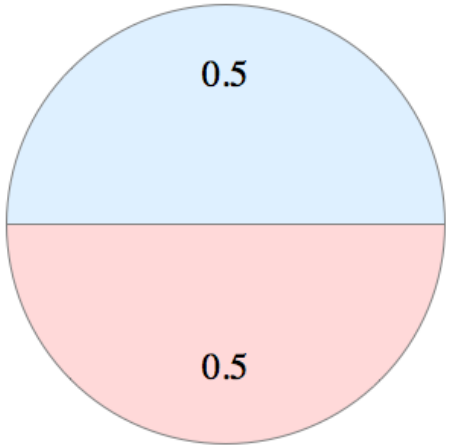


4 Beatles vs Elvis

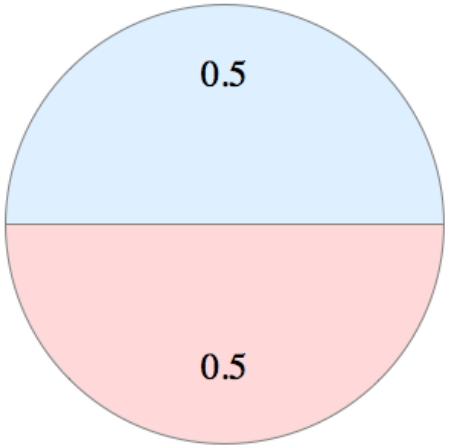


3 Beatles vs Elvis





2 Beatles vs Elvis



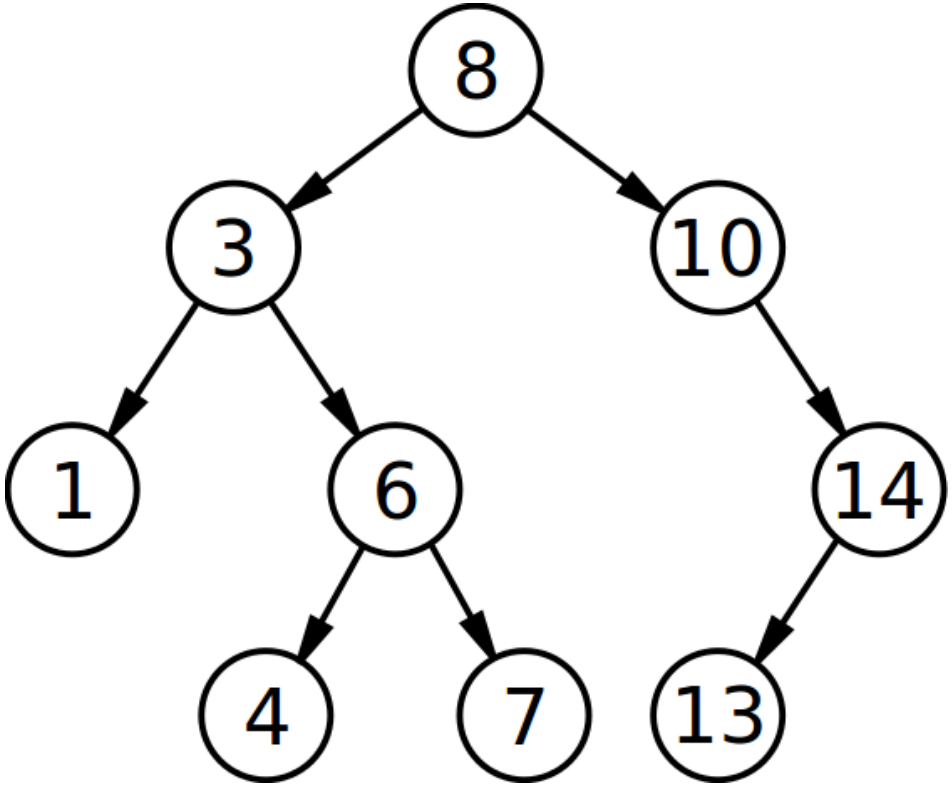
1 Beatle vs Elvis

Goals of Fair Tree

- **If accounts A and B are siblings and A has a higher fairshare factor than B, all children of A will have higher fairshare factors than all children of B**
  - True for all sibling associations at all levels
- Eliminate problems due to floating point precision loss
- sshare and debug support

Trees

- Similar idea to binary search tree:



Fairshare Calculation

- Level Fairshare = S / U
  - S = $RawShares_{self}$ / $RawShares_{self+siblings}$
  - U = $RawUsage_{self}$ / $RawUsage_{self+siblings}$
- Range: [0, infinity]
  - if U == S then LF = 1.0

- Replaces `2^-(U/S)`
- **Only used for sorting**

Tree Traversal

- Create “rooted plane tree”
- Traversal function, starting at root:
  - Calculate Level Fairshare for each child (S/U)
  - Sort children by Level Fairshare from highest to lowest

Tree Traversal

- Create “rooted plane tree”
- Traversal function, starting at root:
  - Calculate Level Fairshare for each child (`S/U`)
  - Sort children by Level Fairshare from highest to lowest
  - Visit the children in order
    - If account, recurse with account as new root
    - If user, assign a final fairshare factor based on ranking

Traversal in the Fair Tree Algorithm for Slurm

Fair Tree: Traversal

= **visit association**

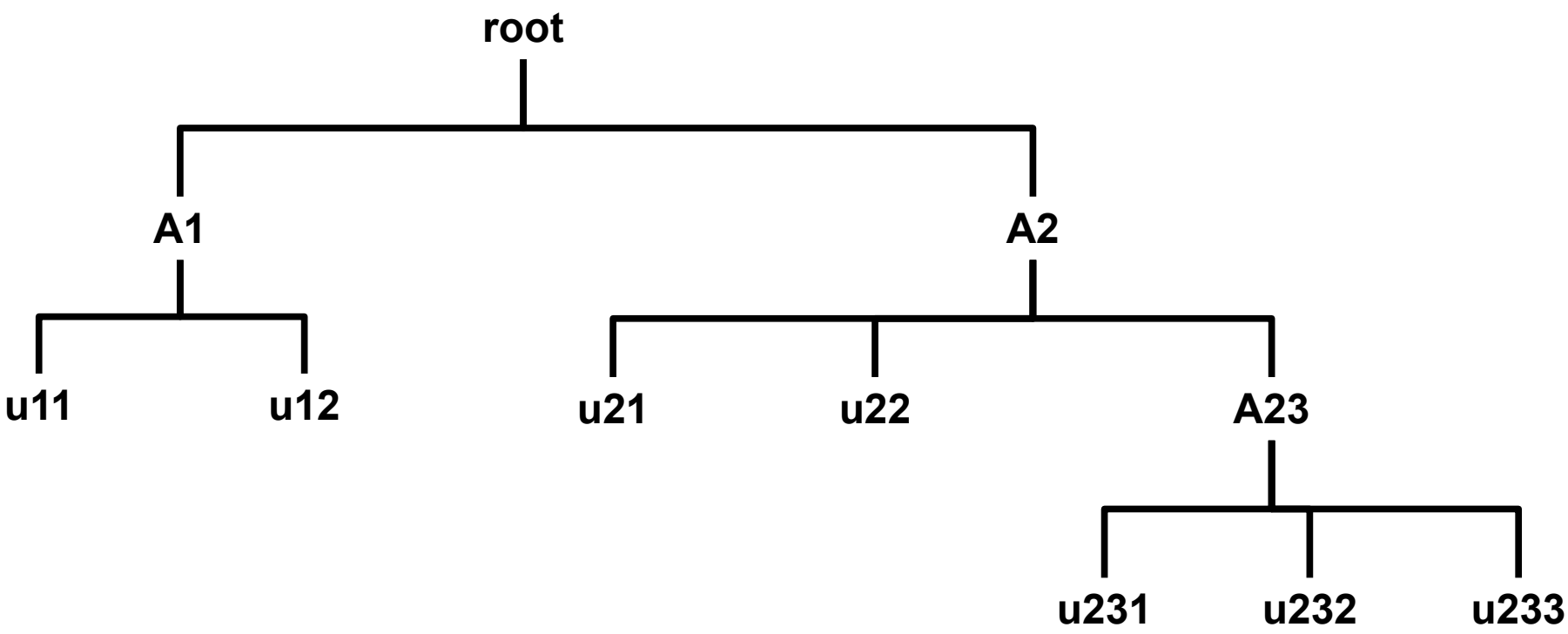
= **calculate level fairshare**

= **sort siblings**

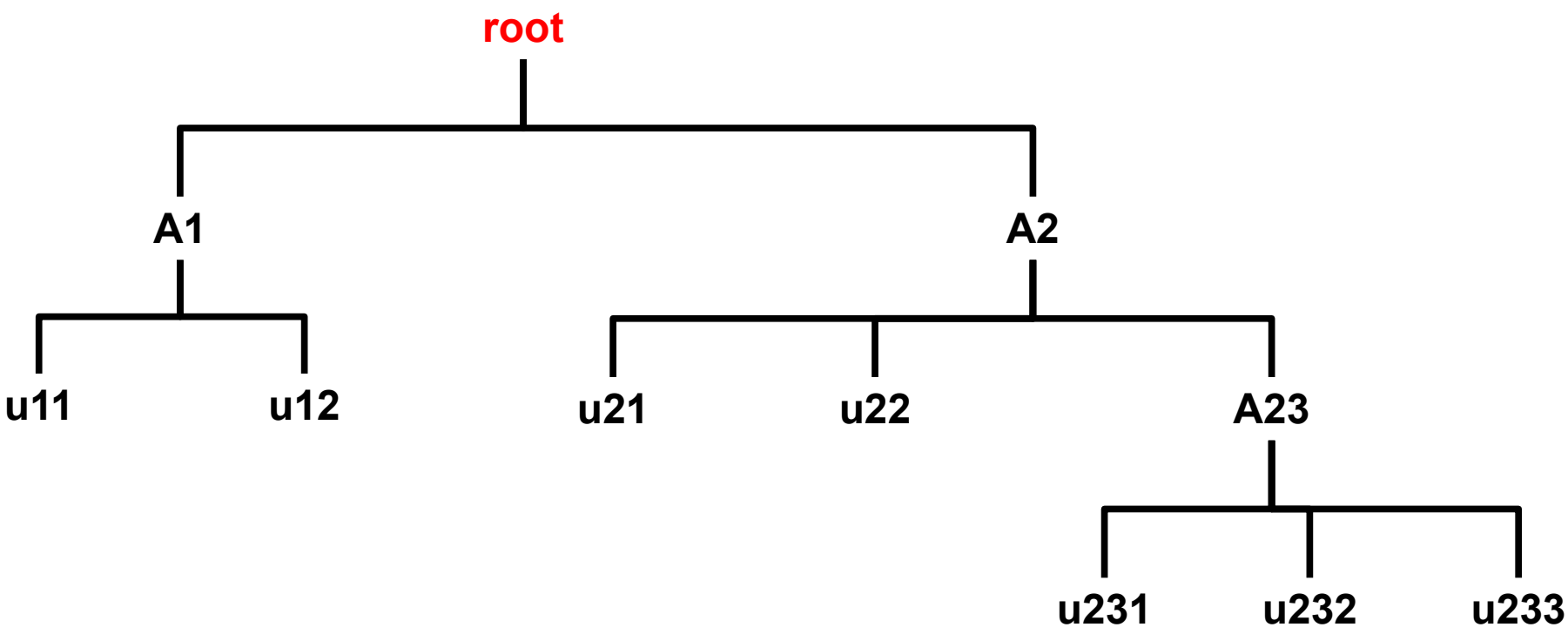
u = **calculate final fairshare for user**

Important Notes: Example level fairshare values are completely unrealistic. Final fairshare value not shown.

Fair Tree: Traversal

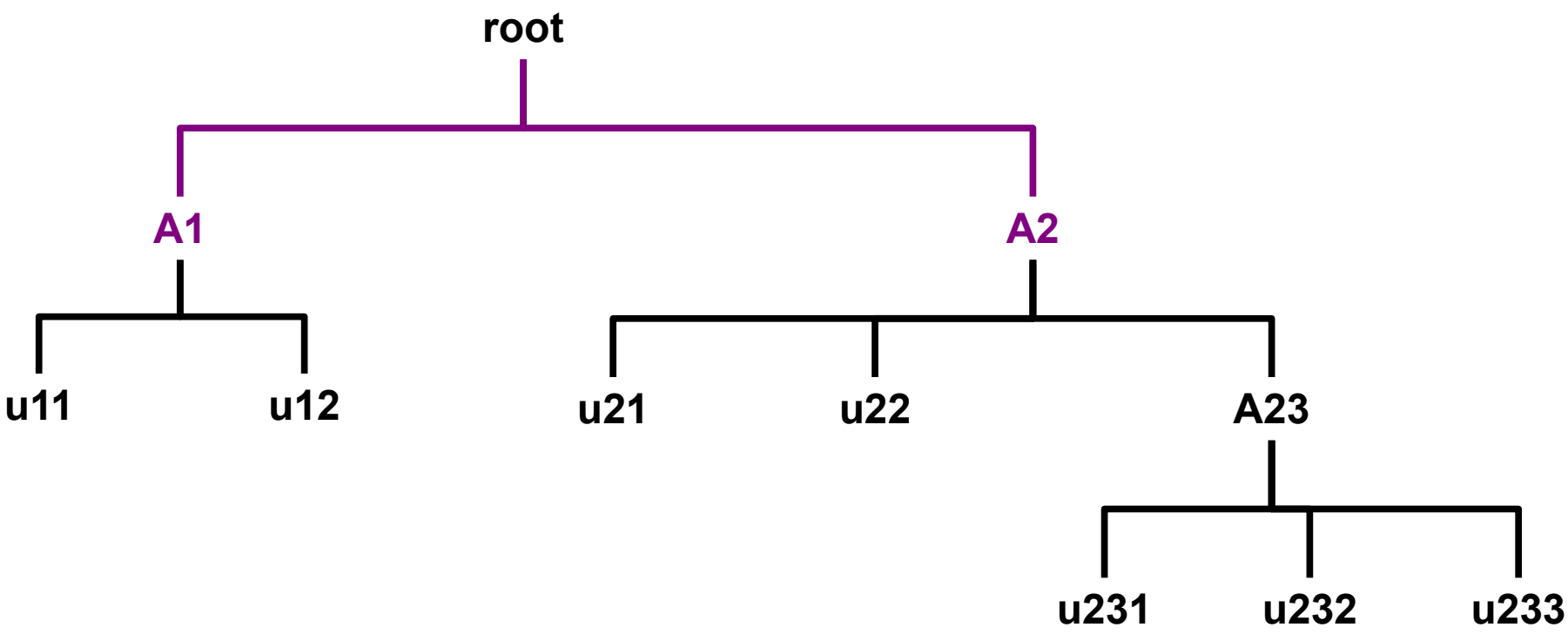


Fair Tree: Traversal



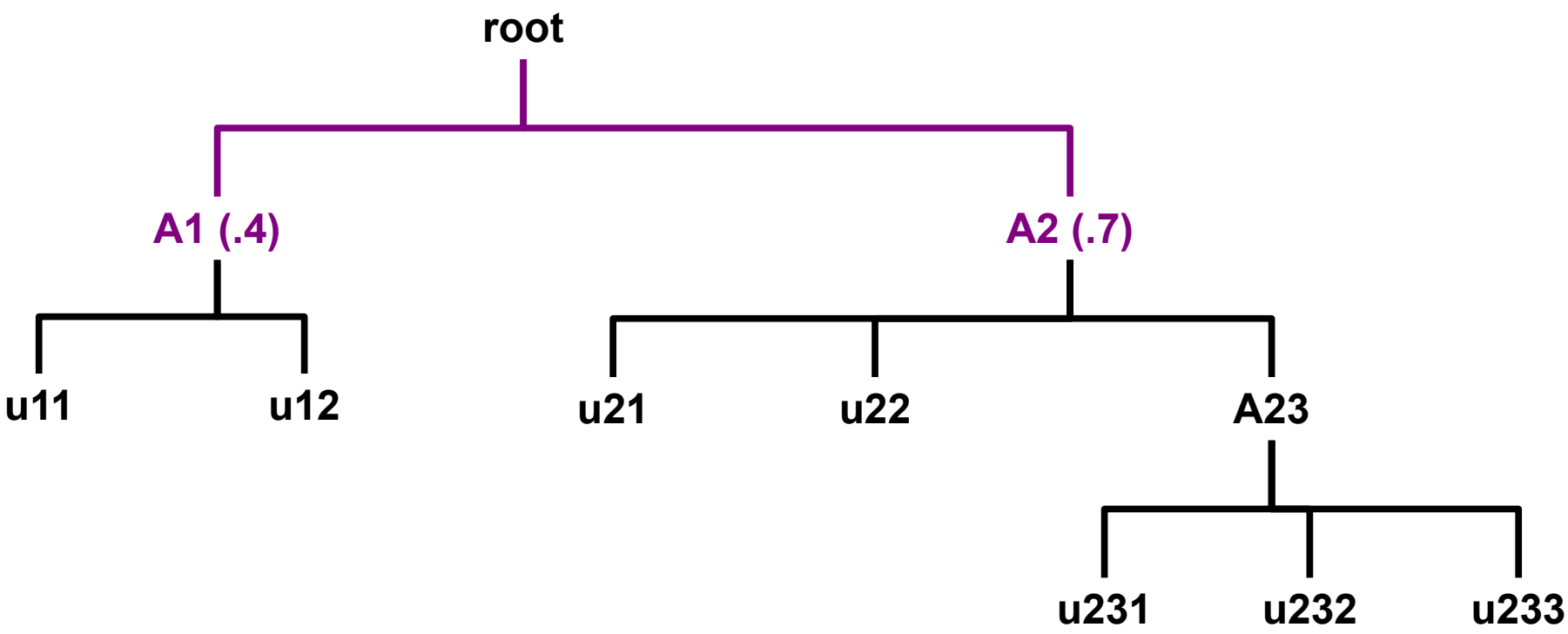
Visit association

Fair Tree: Traversal



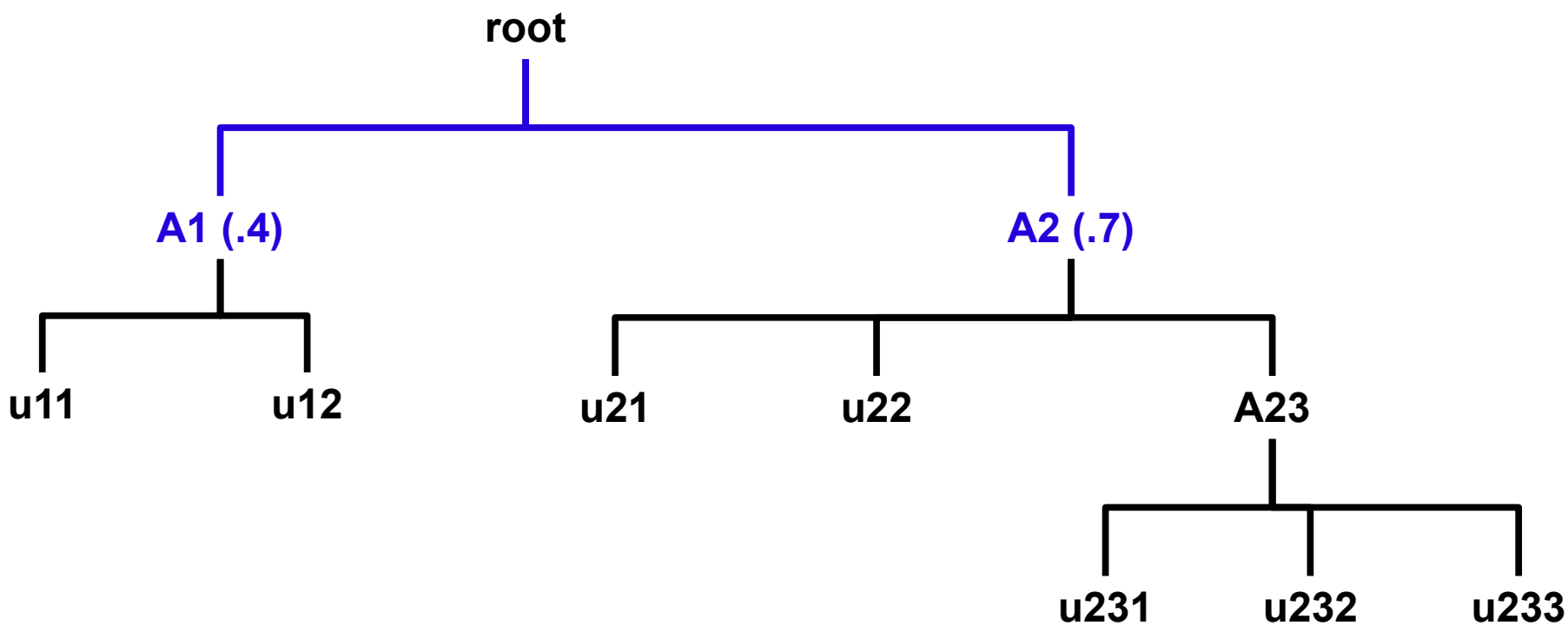
Calculate level fairshare

Fair Tree: Traversal



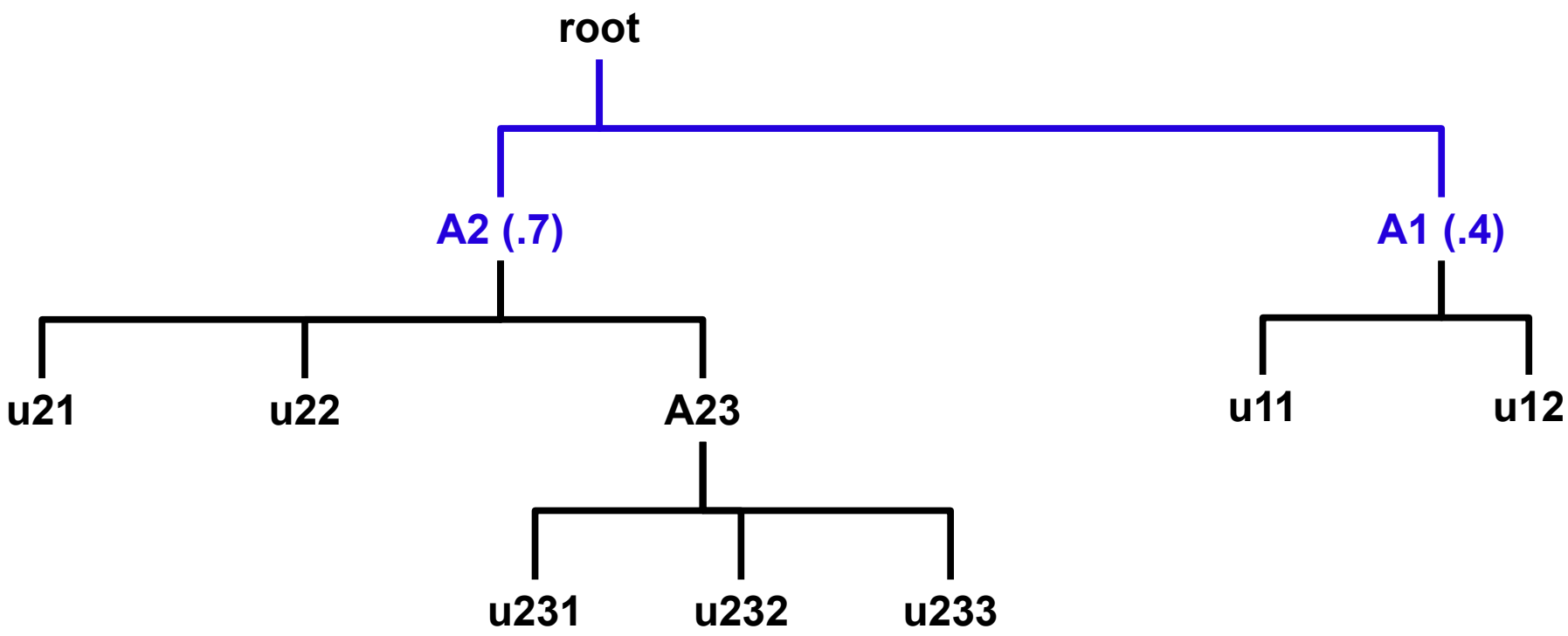
Calculate level fairshare

Fair Tree: Traversal



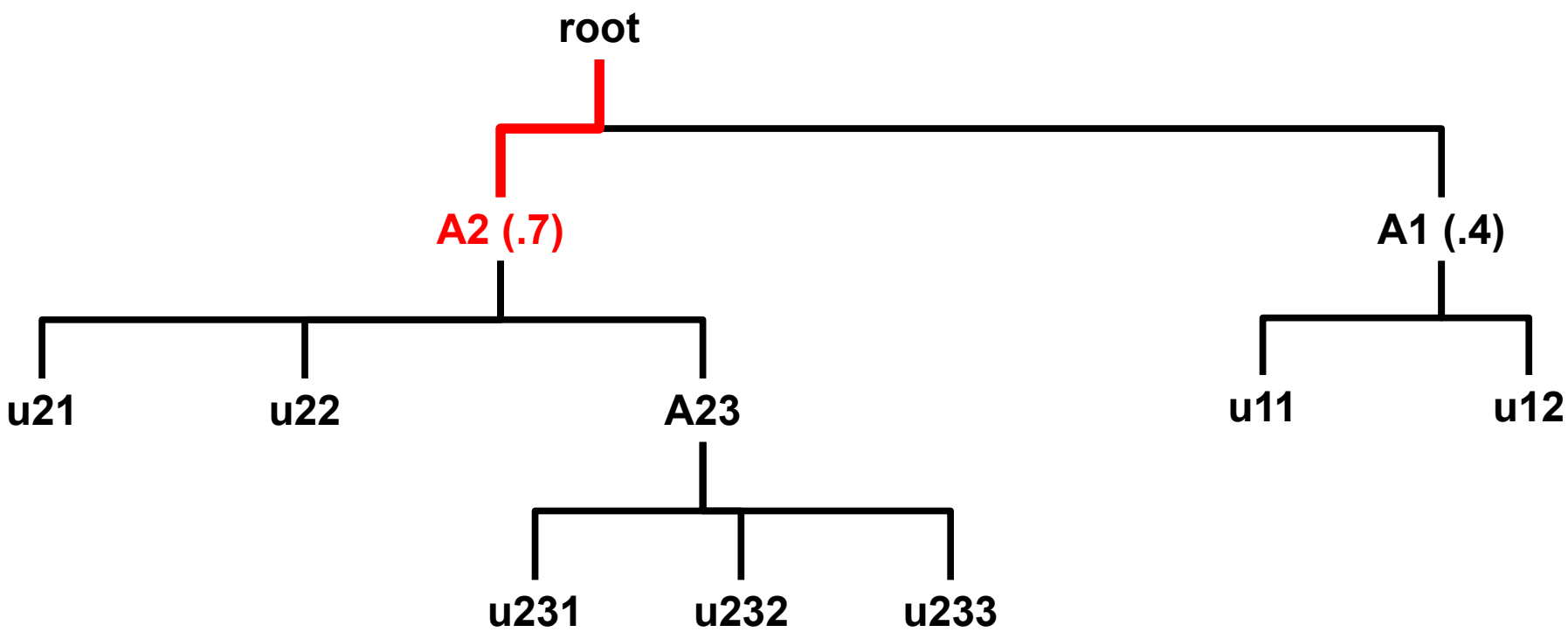
Sort by level fairshare

Fair Tree: Traversal



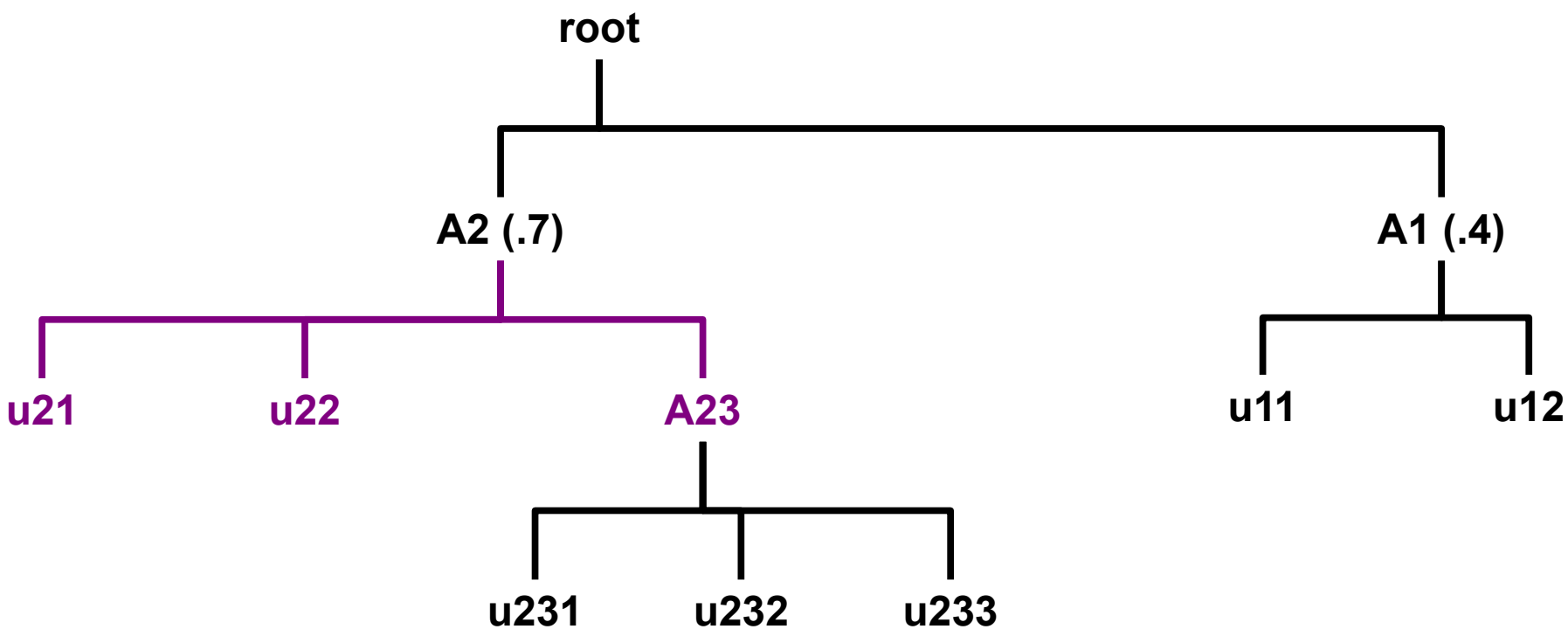
Sort by level fairshare

Fair Tree: Traversal



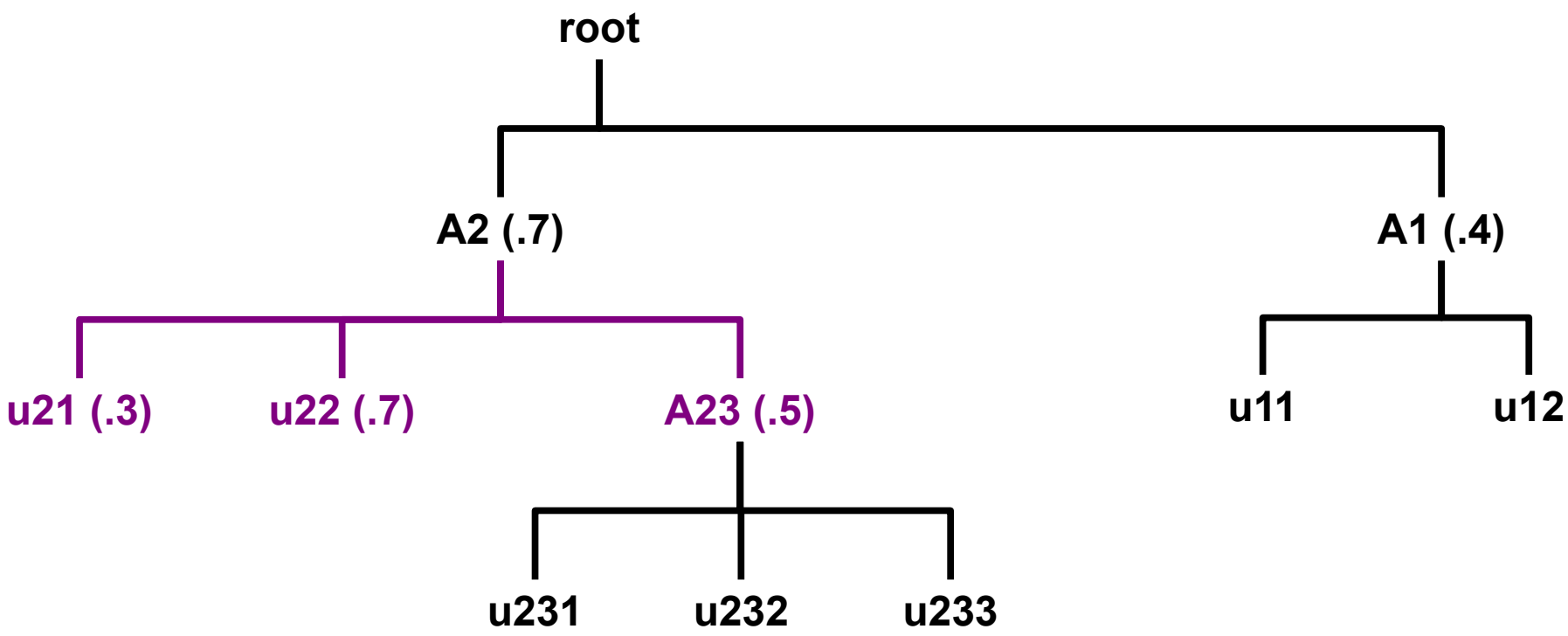
Visit association

Fair Tree: Traversal



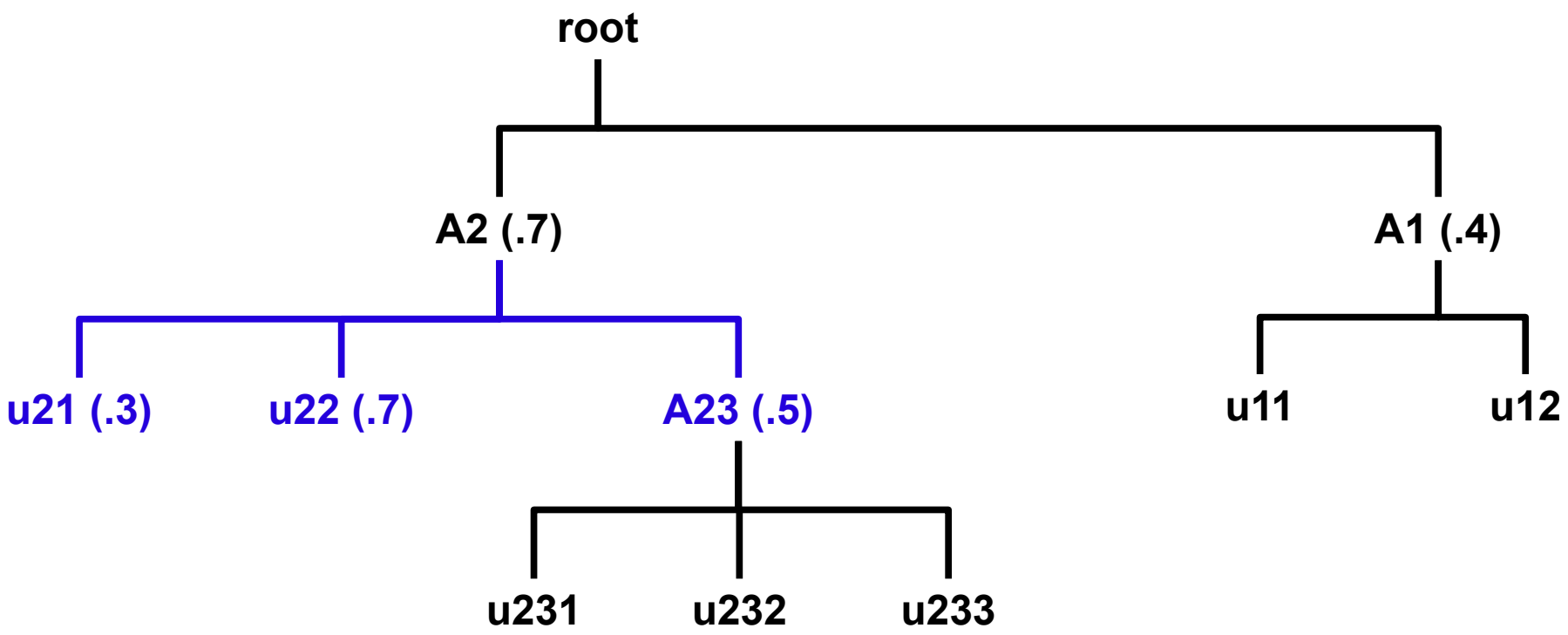
Calculate level fairshare

Fair Tree: Traversal



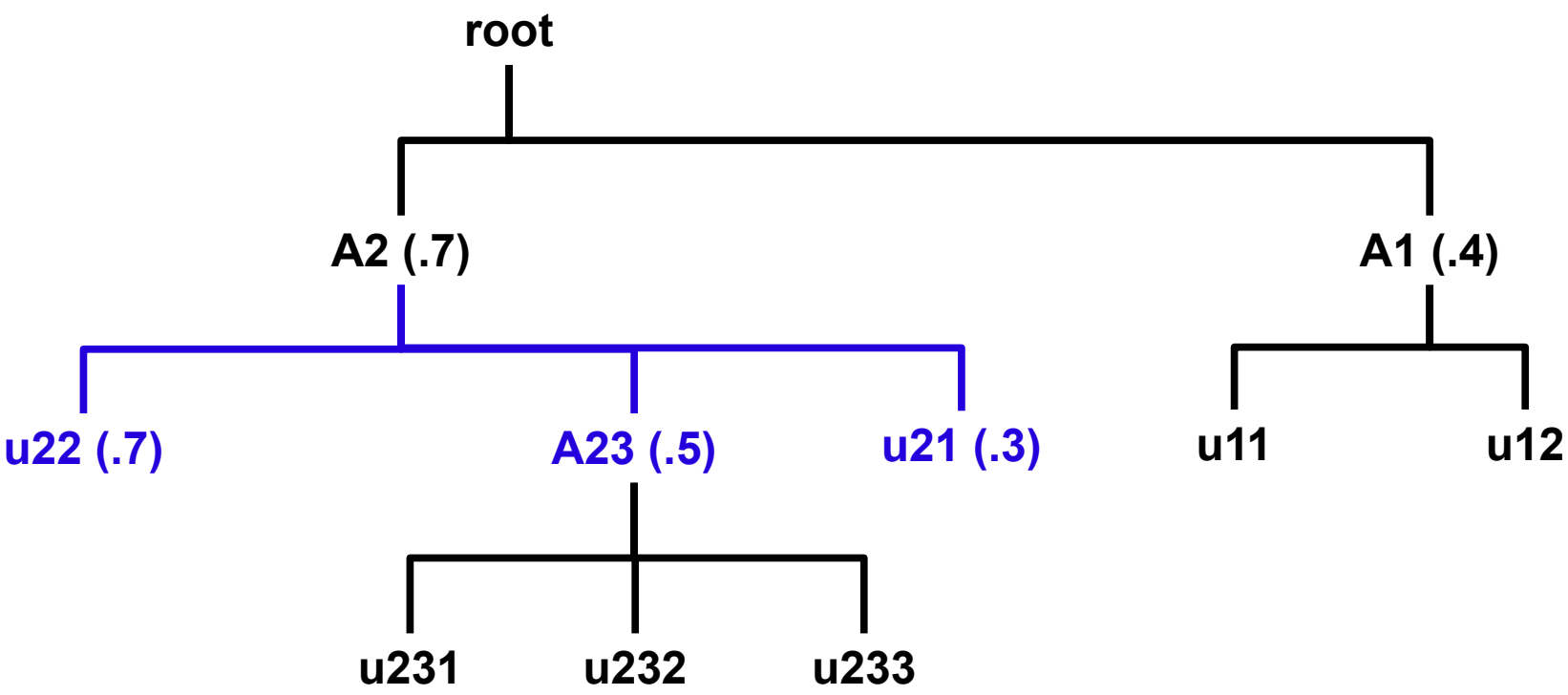
Calculate level fairshare

Fair Tree: Traversal



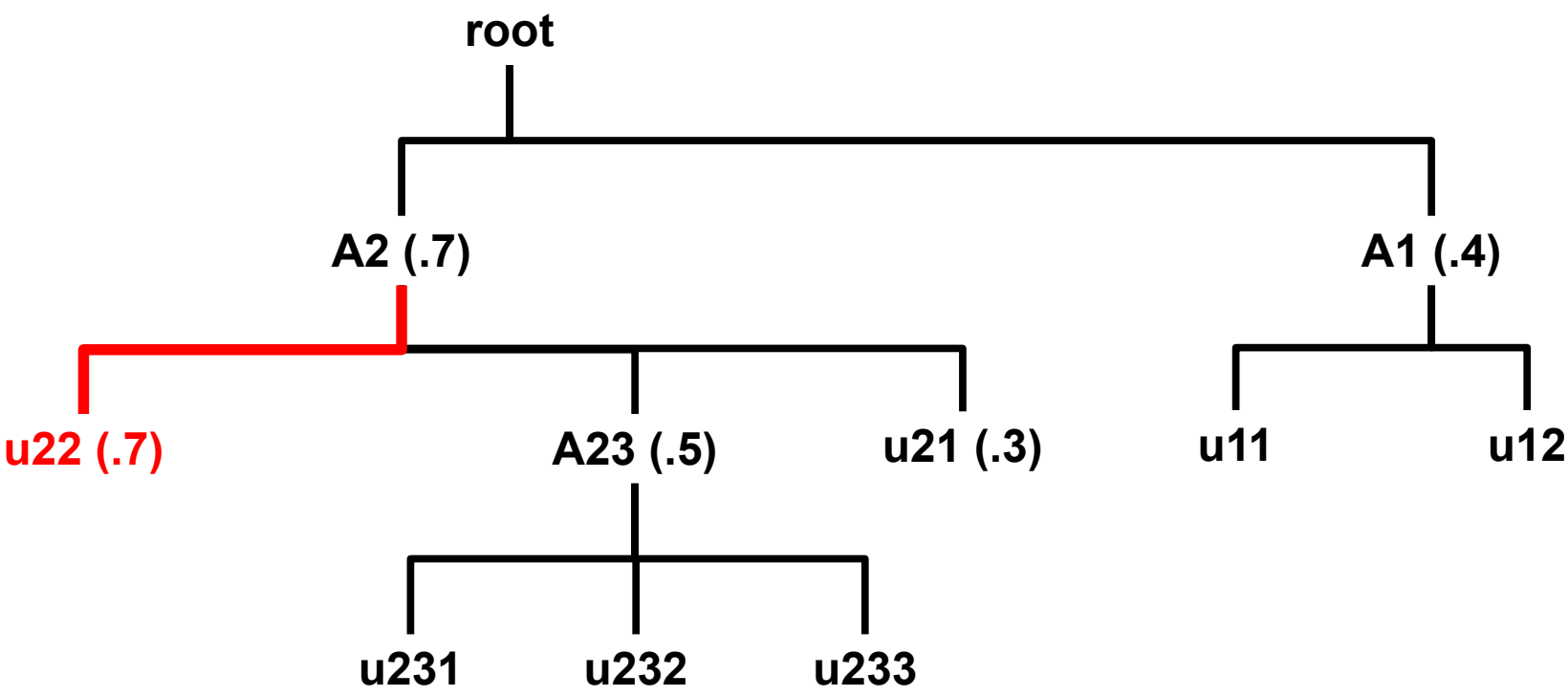
Sort by level fairshare

Fair Tree: Traversal



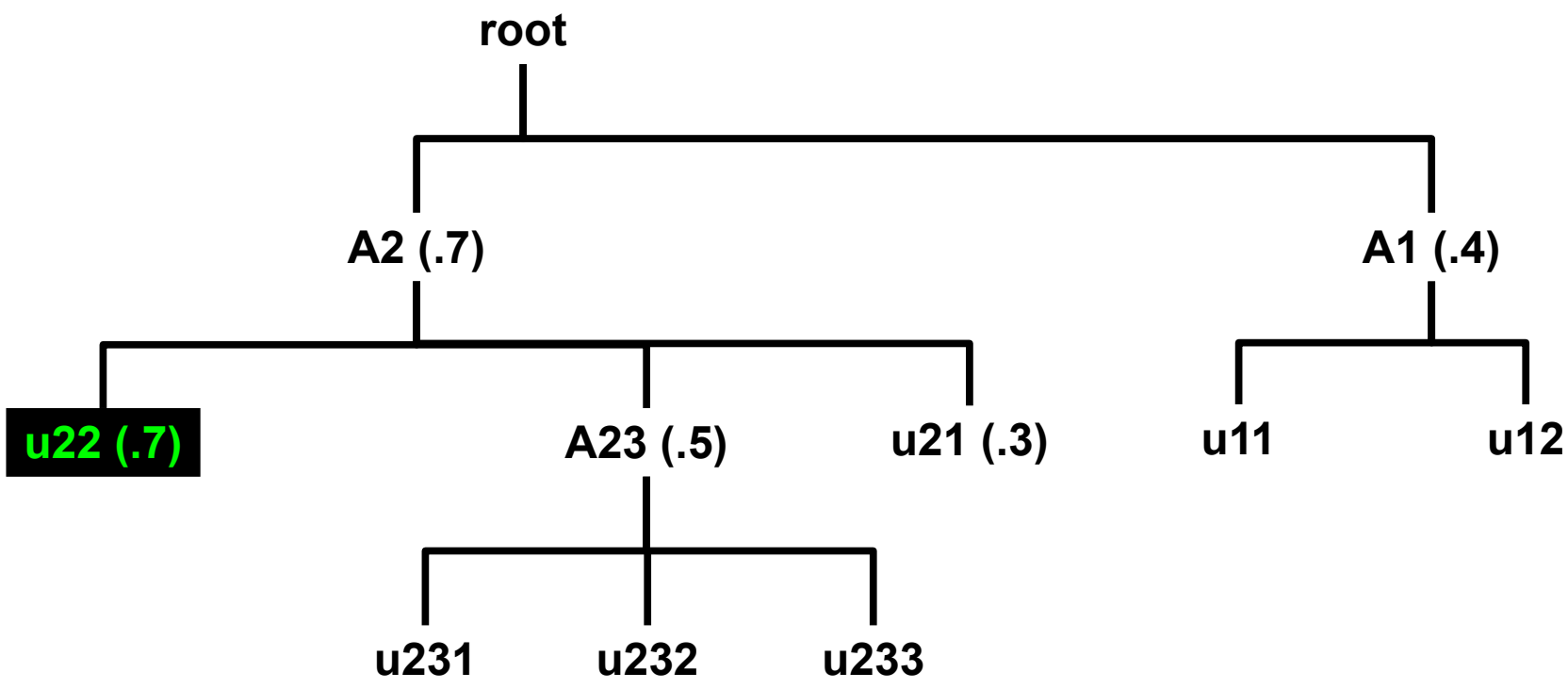
Sort by level fairshare

Fair Tree: Traversal



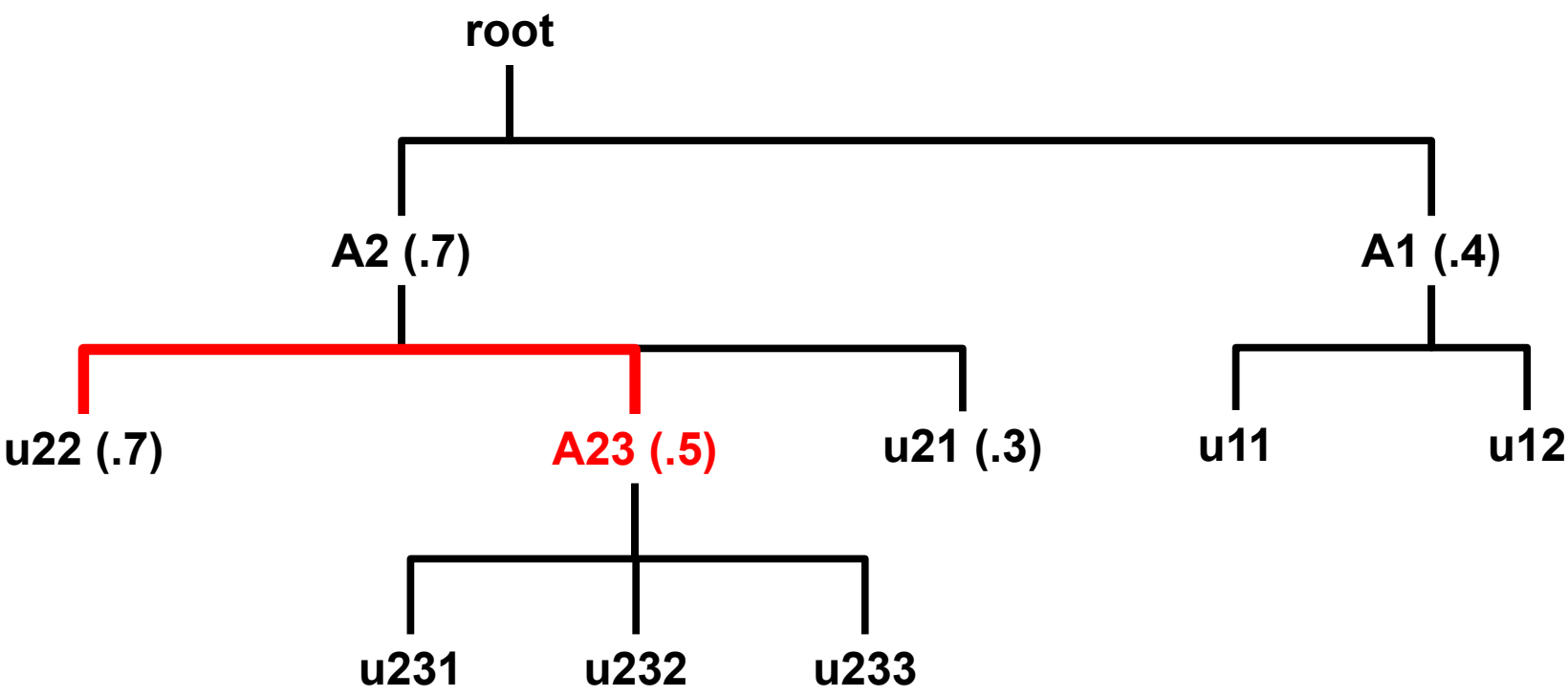
Visit association

Fair Tree: Traversal



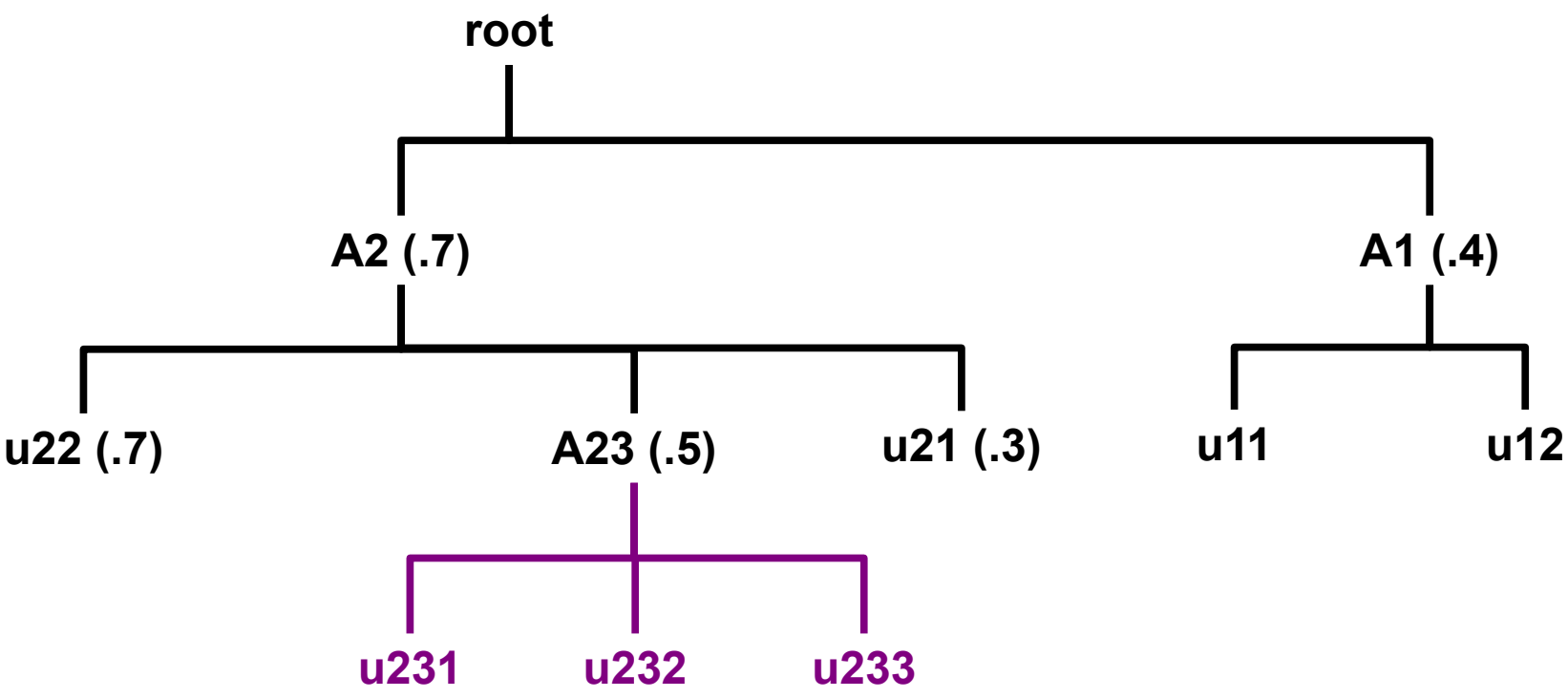
Final fairshare value = rank-- / user_count

Fair Tree: Traversal



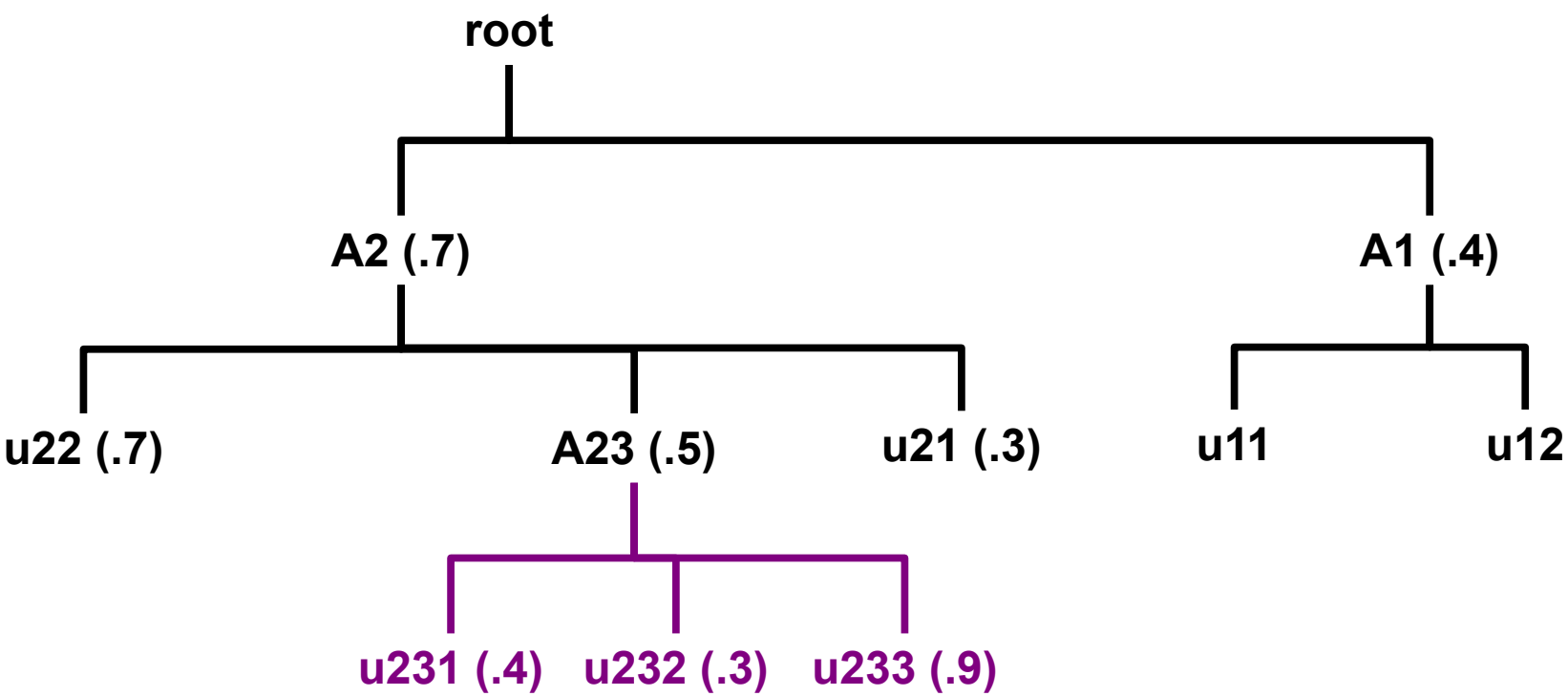
Visit association

Fair Tree: Traversal



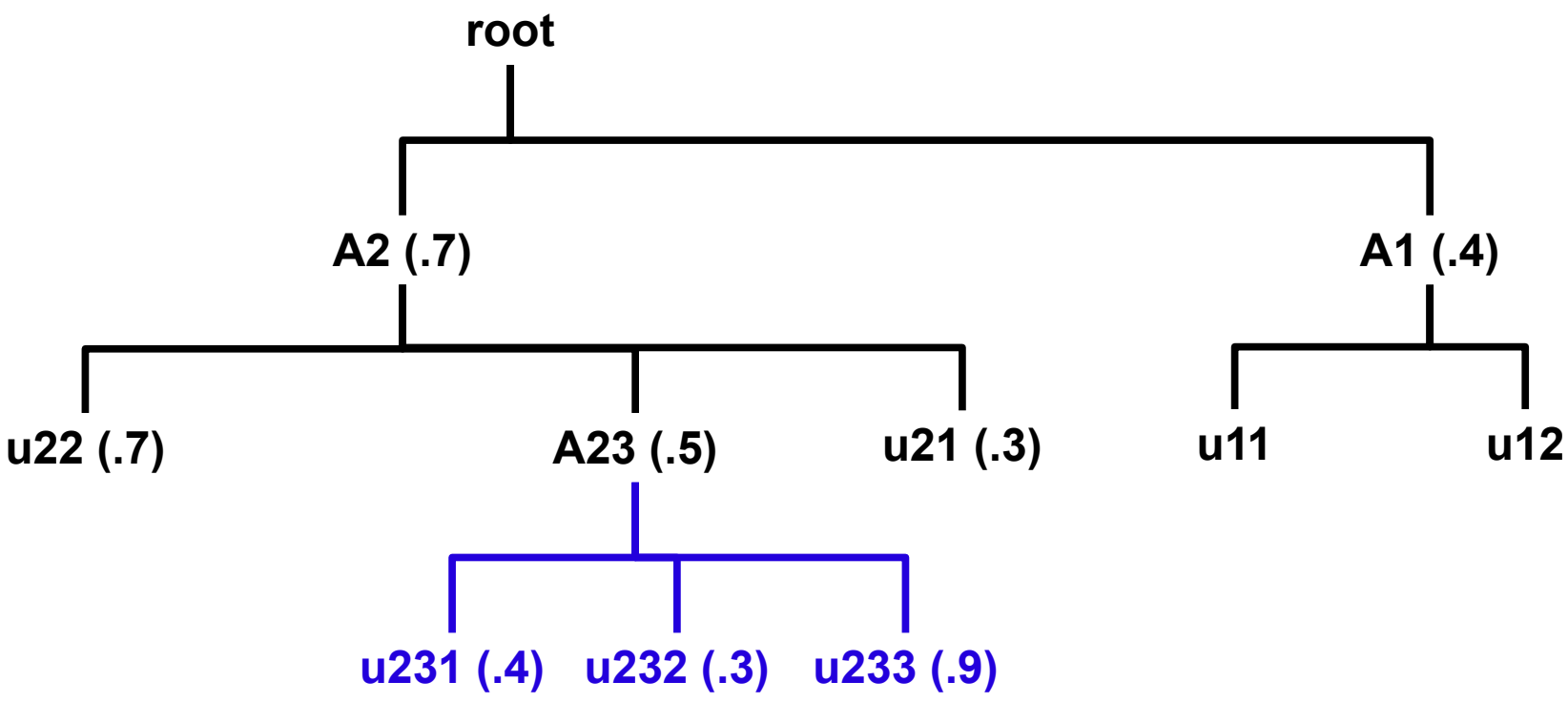
Calculate level fairshare

Fair Tree: Traversal



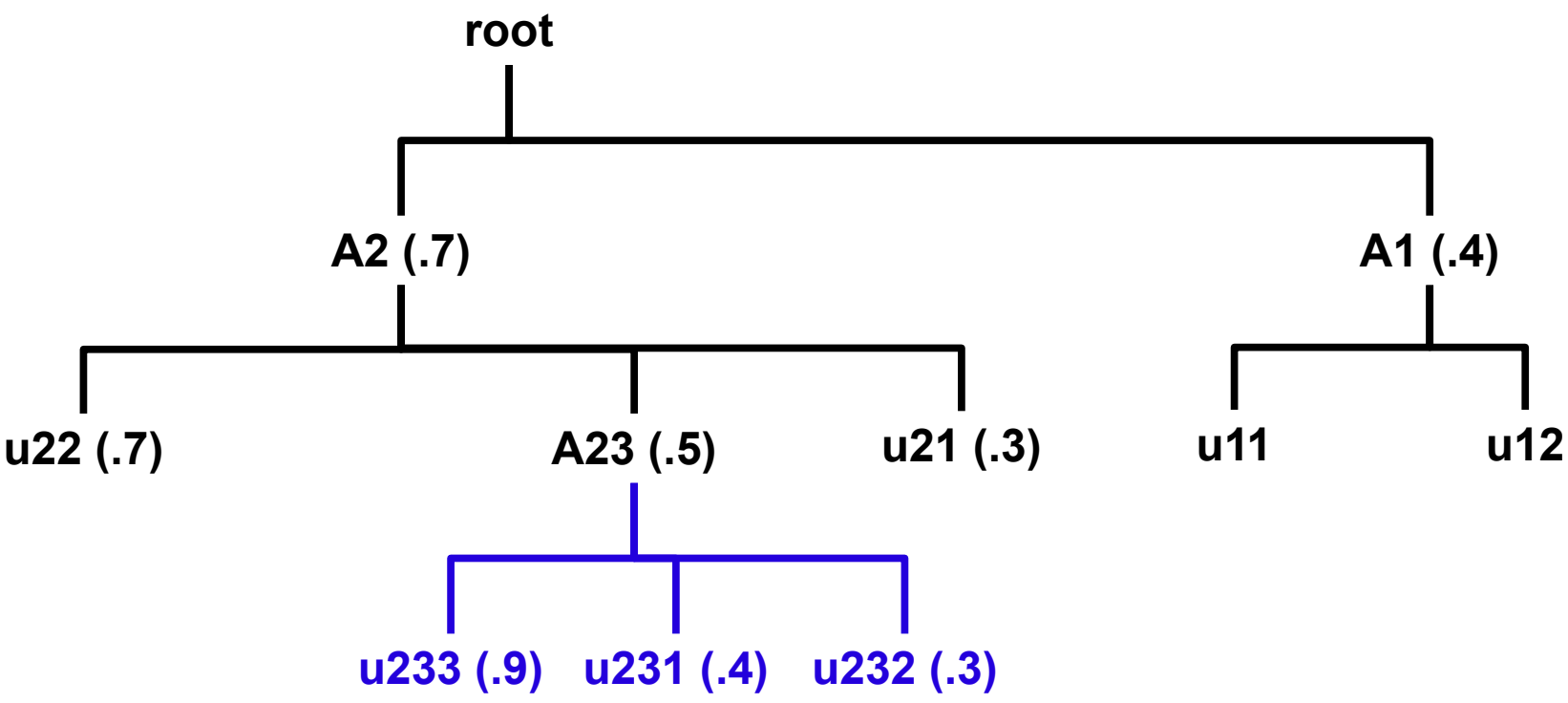
Calculate level fairshare

Fair Tree: Traversal



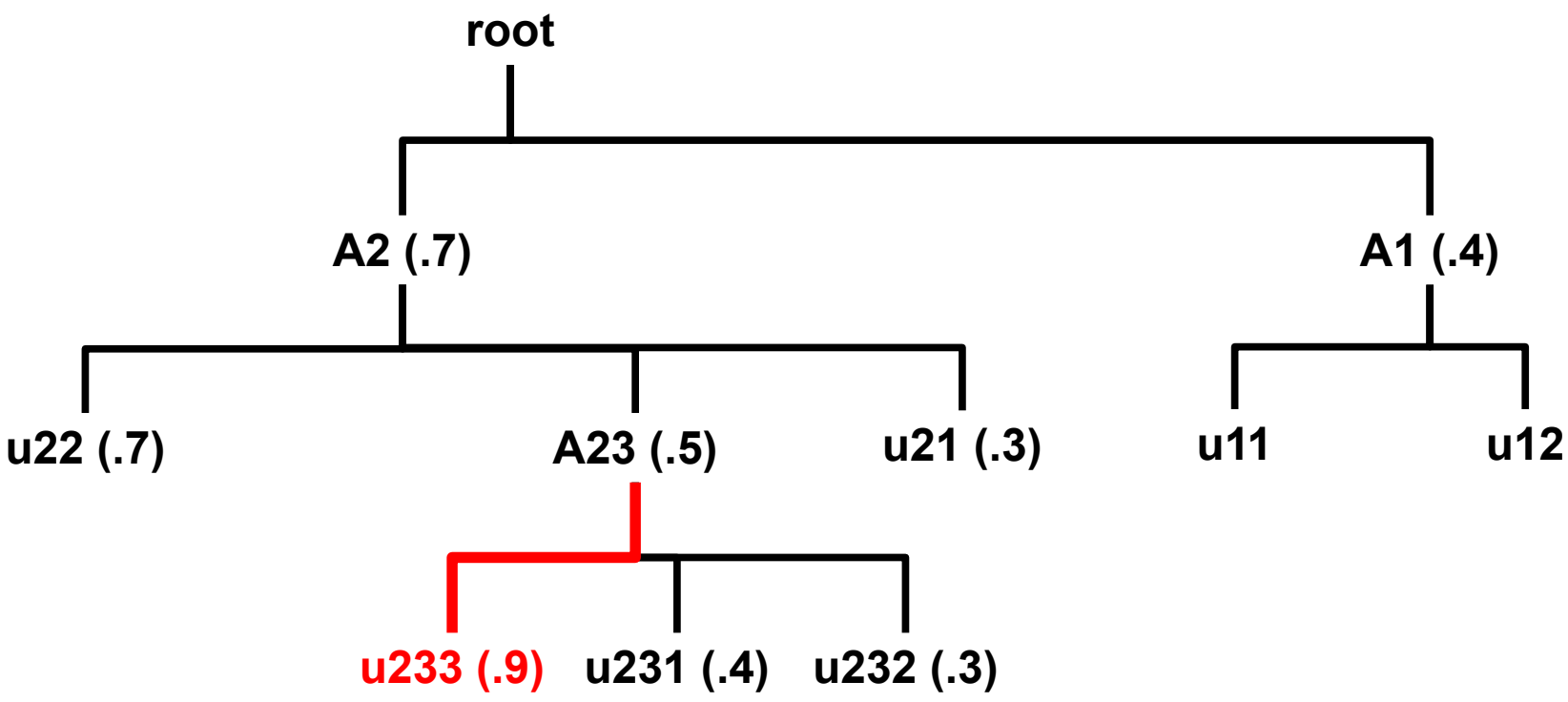
Sort by level fairshare

Fair Tree: Traversal



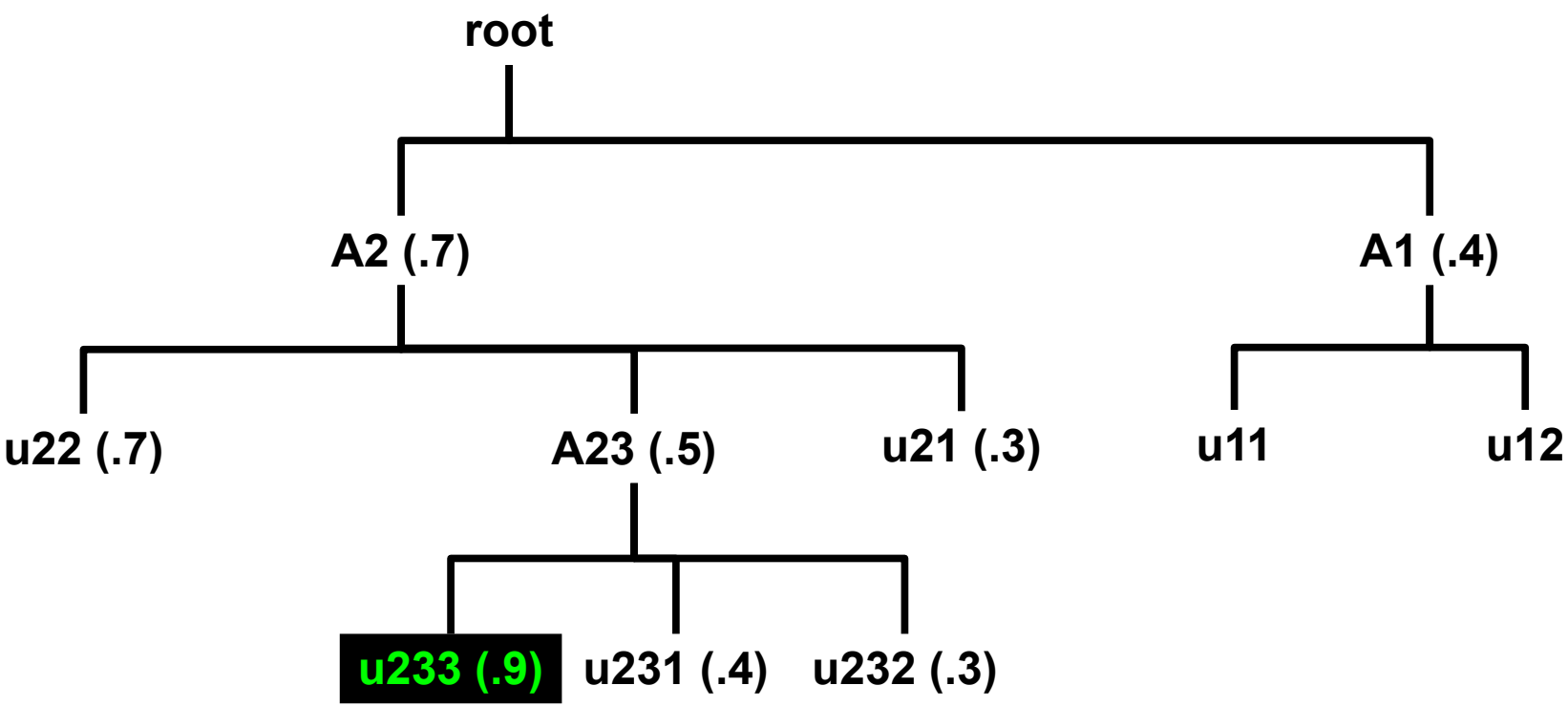
Sort by level fairshare

Fair Tree: Traversal



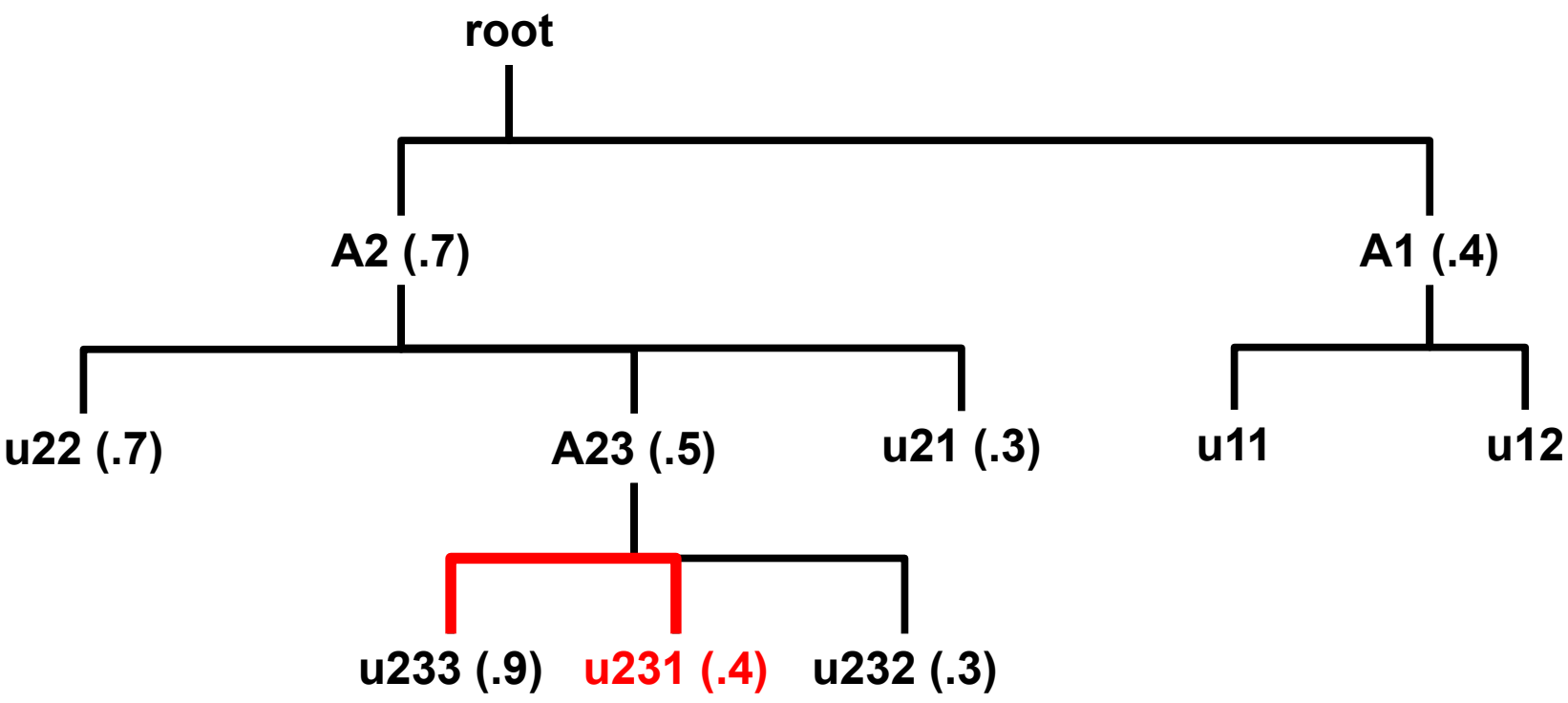
Visit association

Fair Tree: Traversal



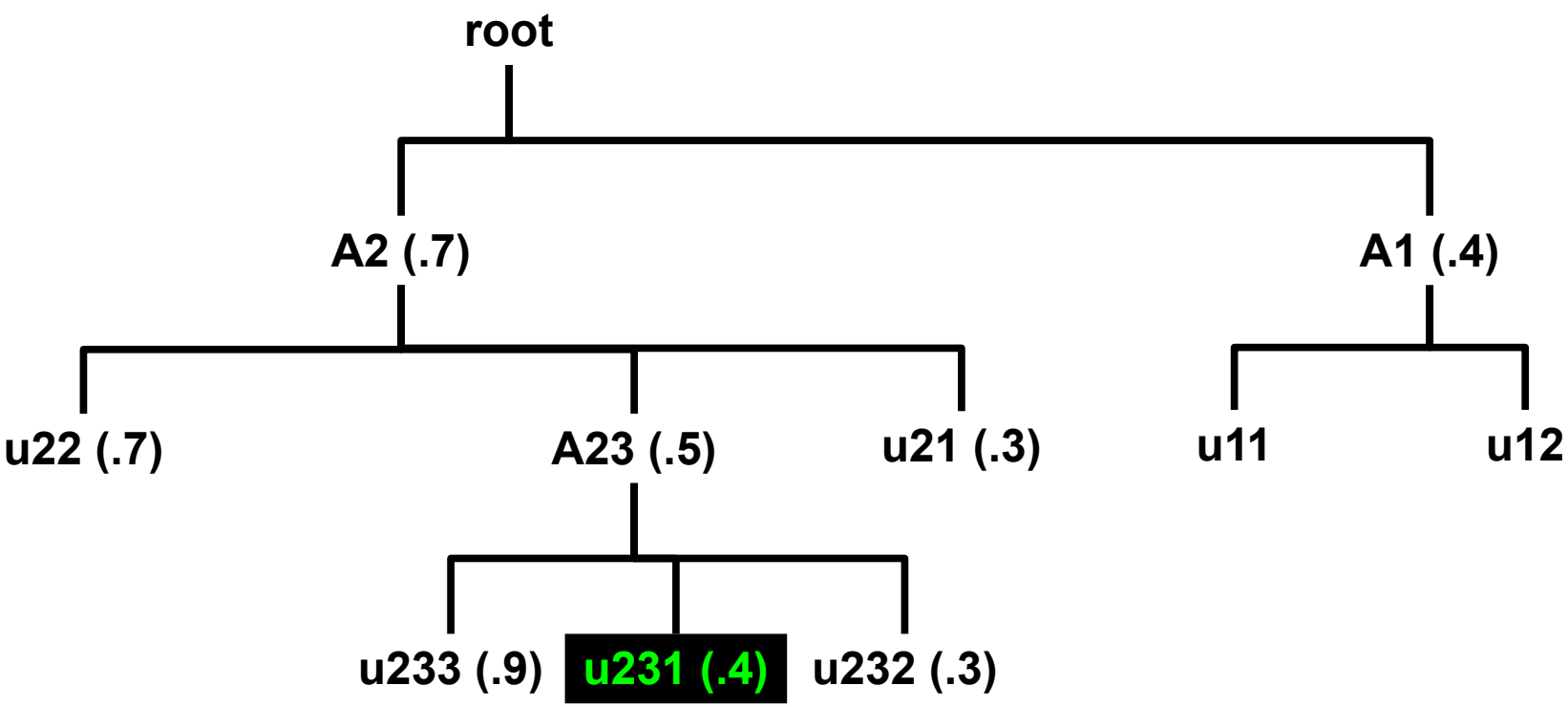
Final fairshare value = rank-- / user_count

Fair Tree: Traversal



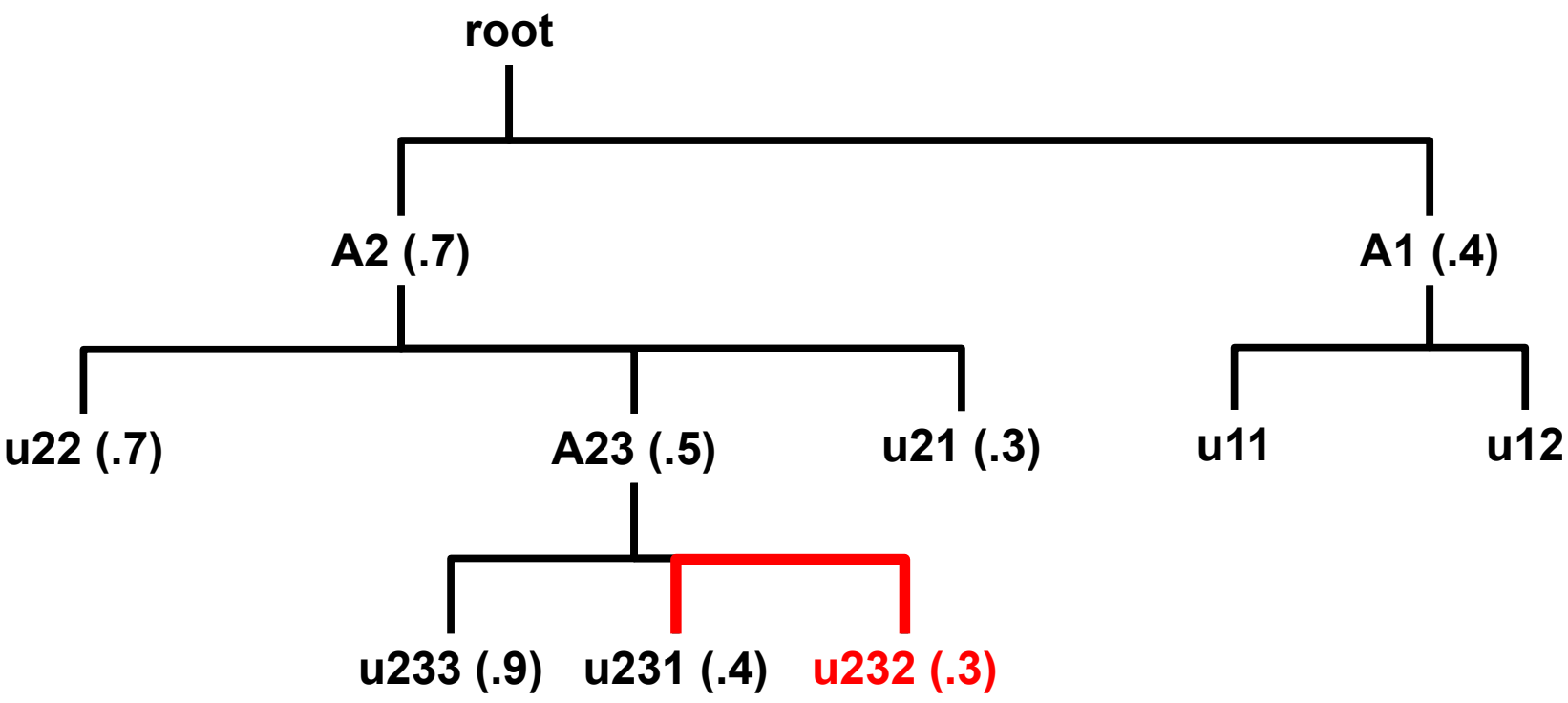
Visit association

Fair Tree: Traversal



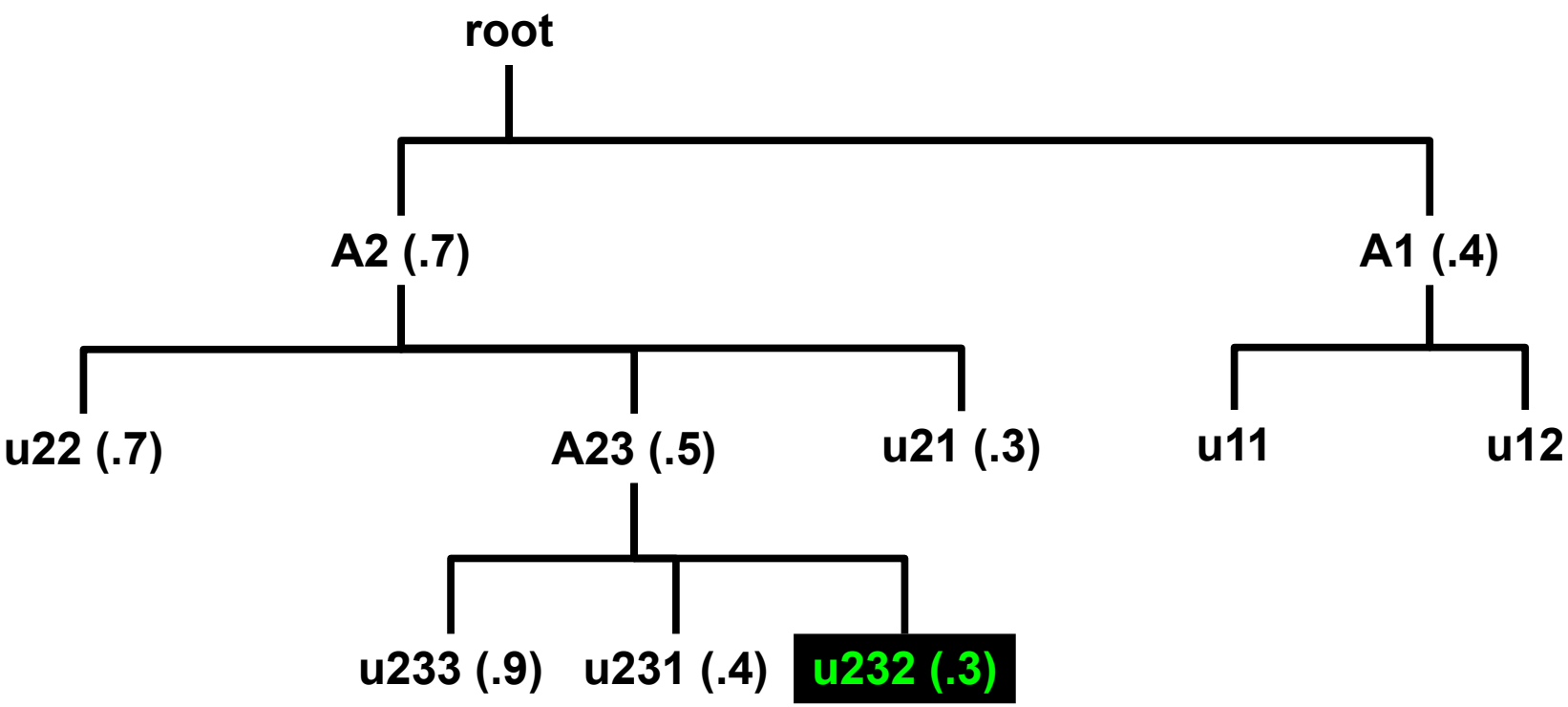
Final fairshare value = rank-- / user_count

Fair Tree: Traversal



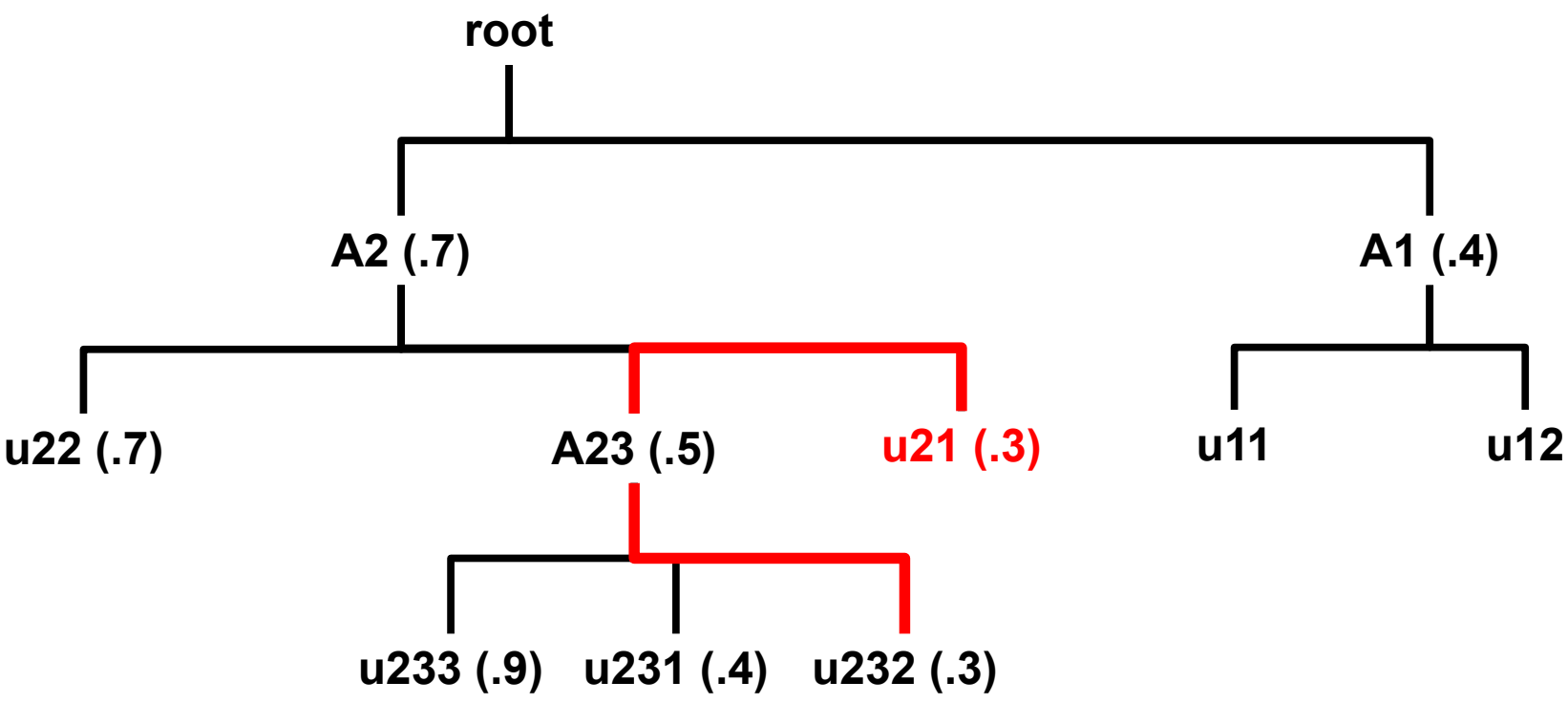
Visit association

Fair Tree: Traversal



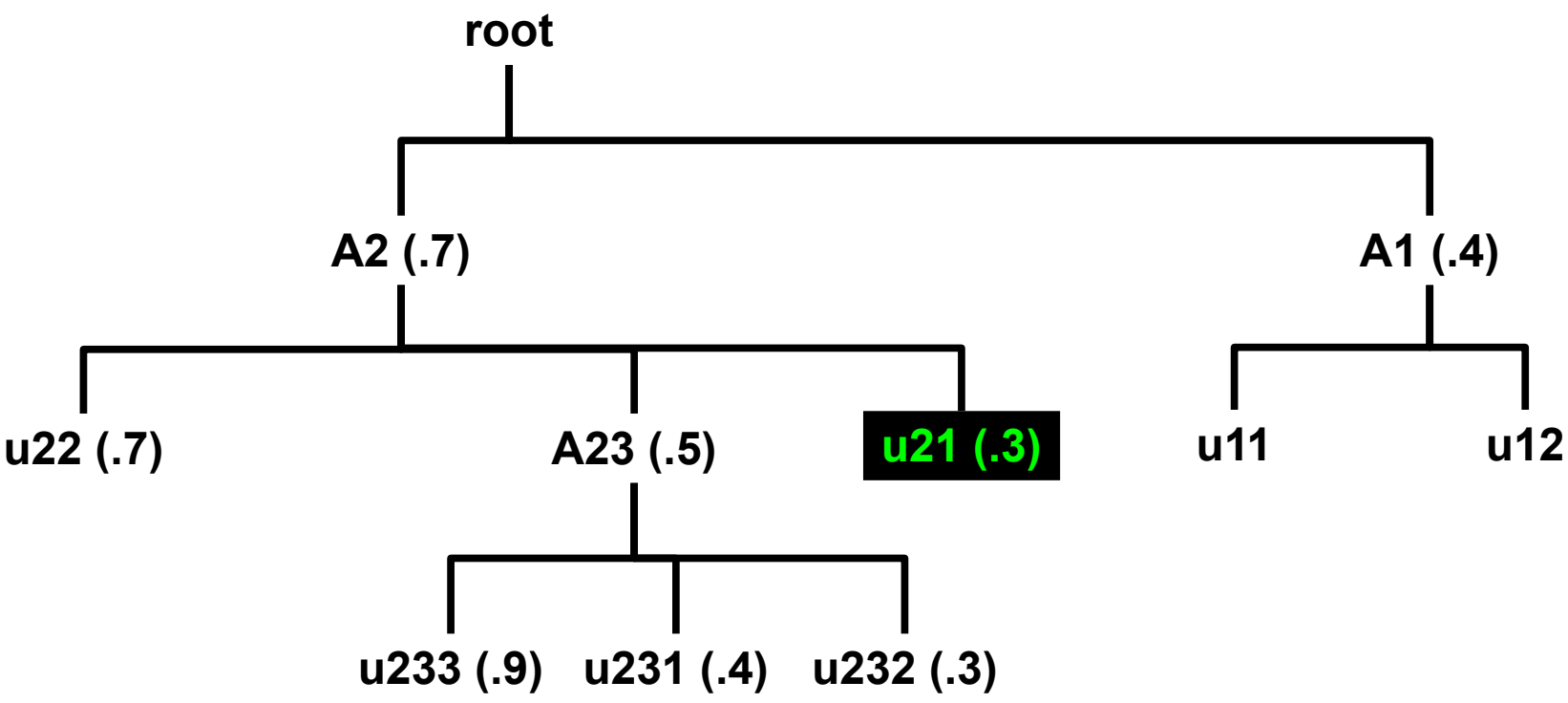
Final fairshare value = rank-- / user_count

Fair Tree: Traversal



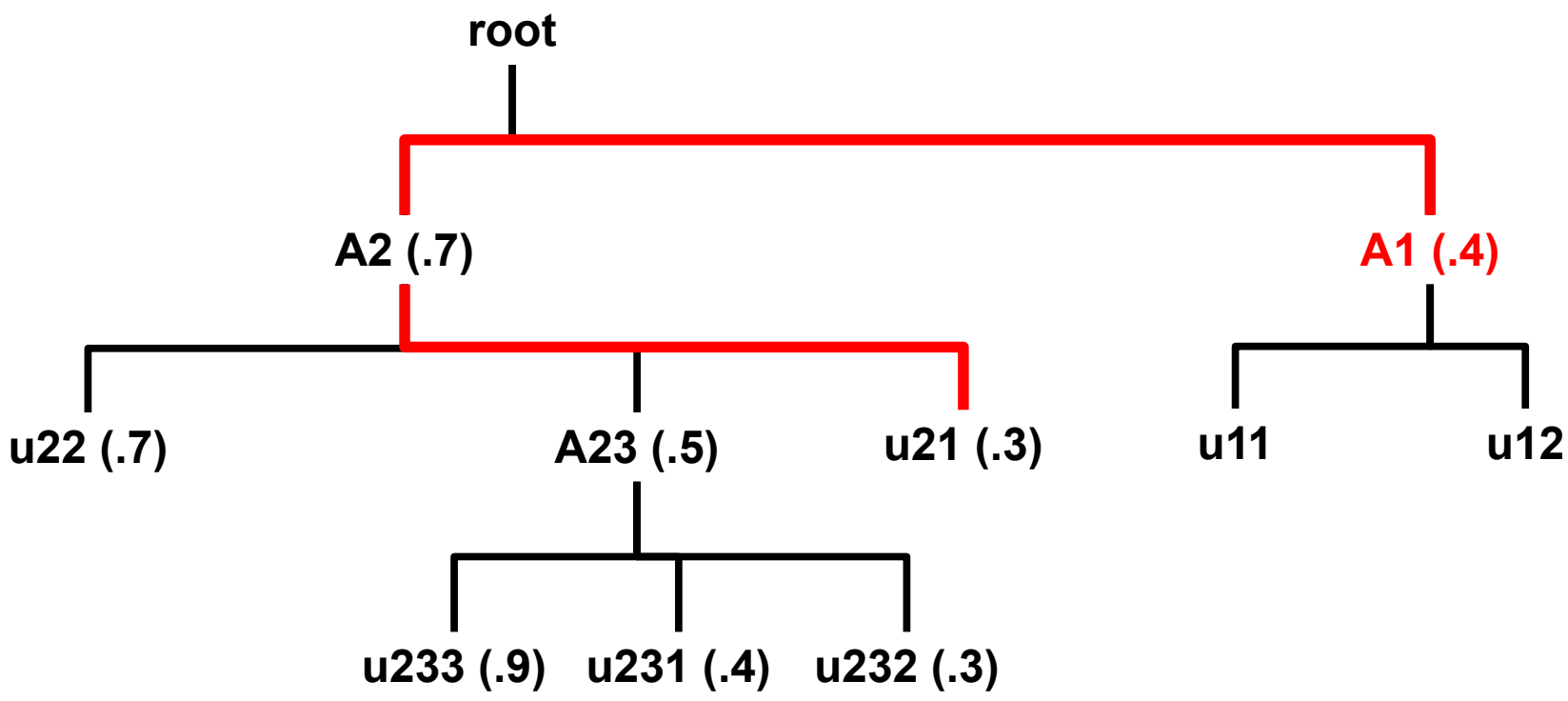
Visit association

Fair Tree: Traversal



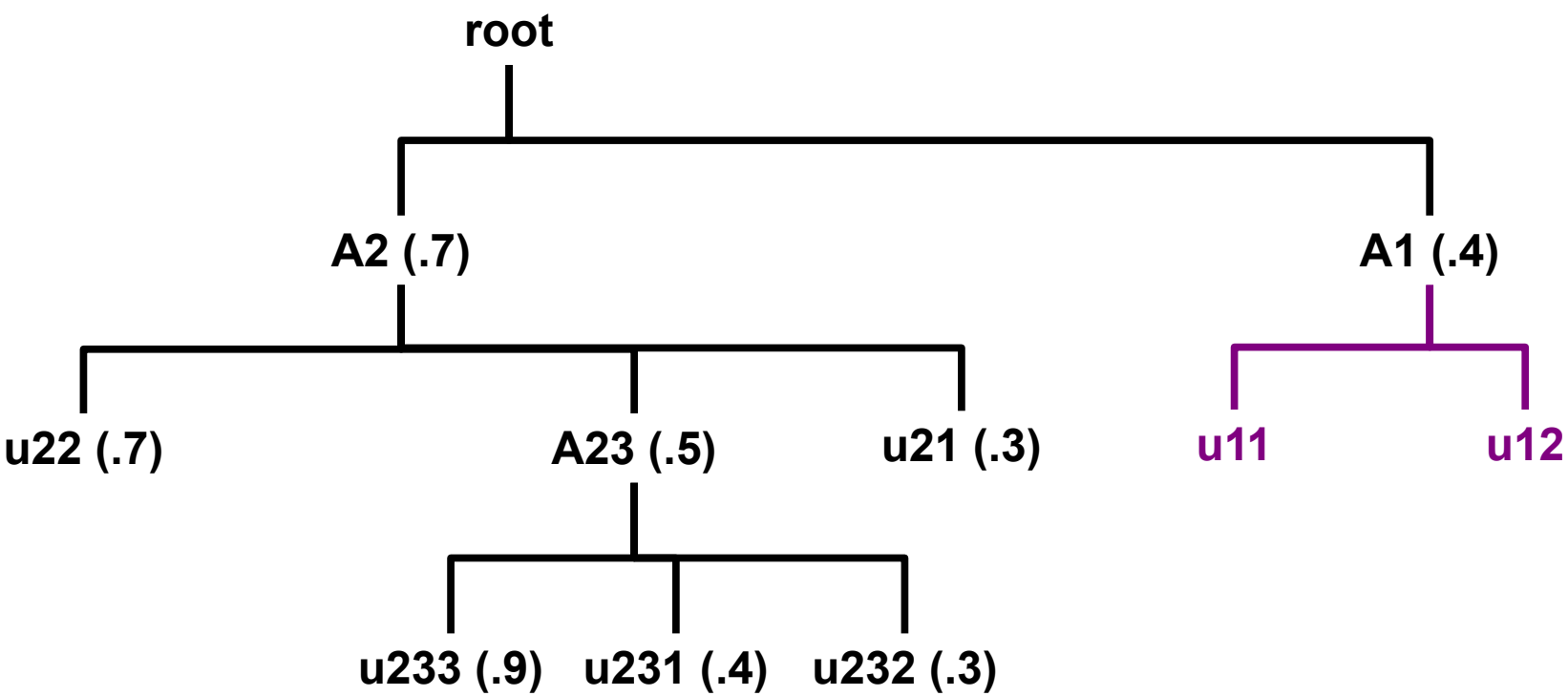
Final fairshare value = rank-- / user_count

Fair Tree: Traversal



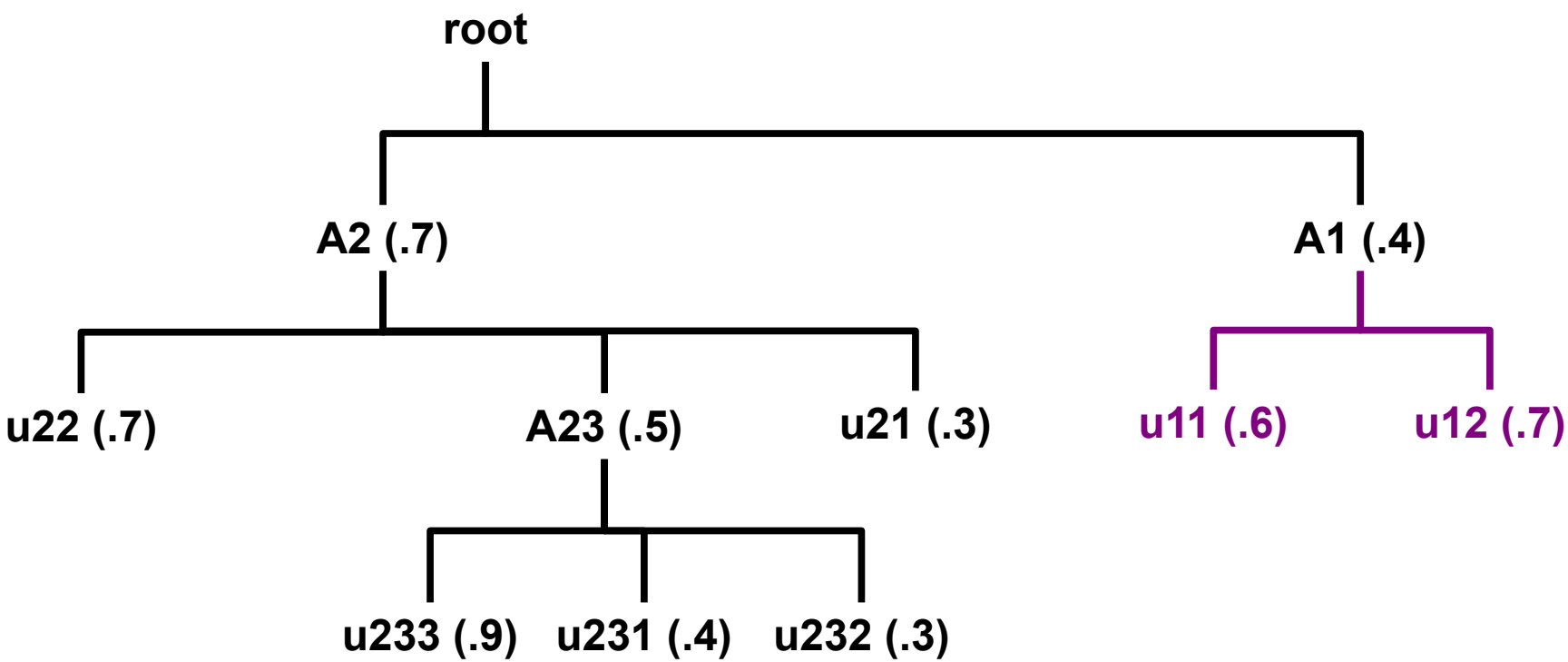
Visit association

Fair Tree: Traversal



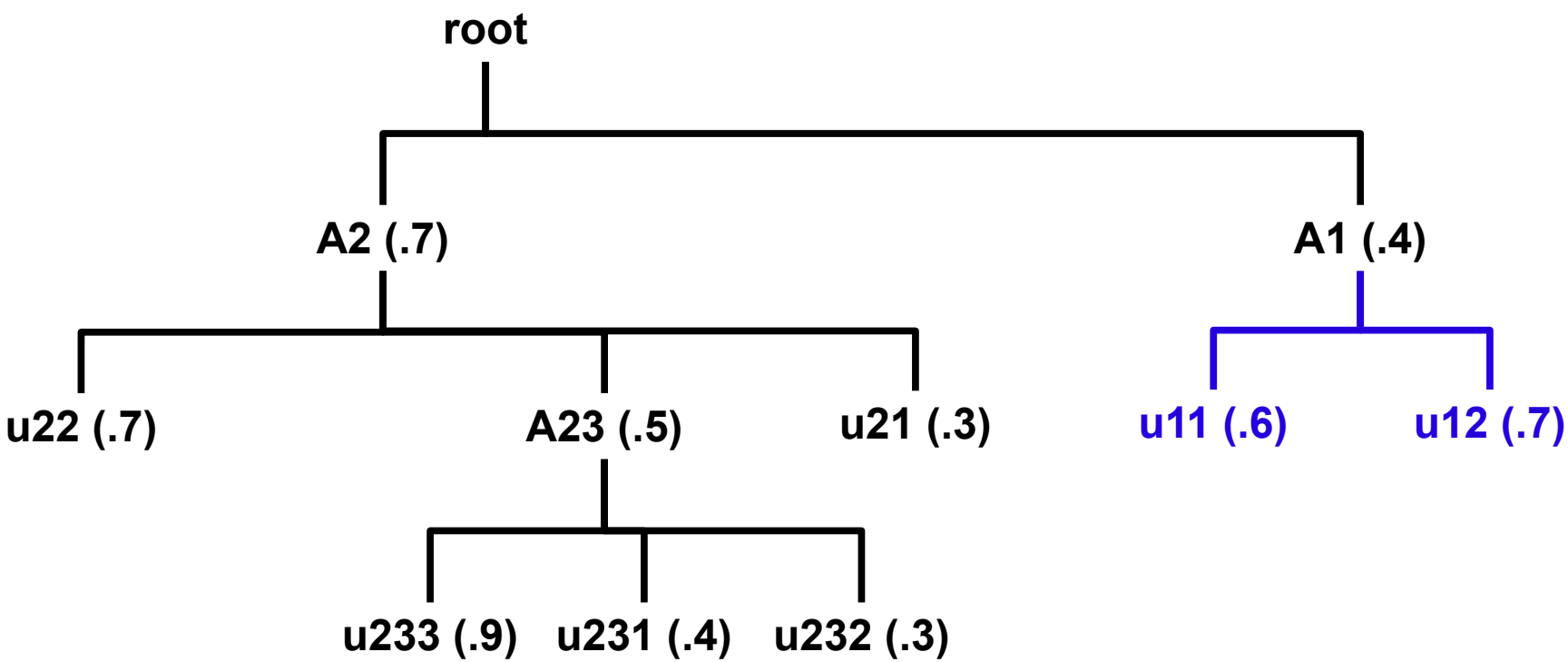
Calculate level fairshare

Fair Tree: Traversal



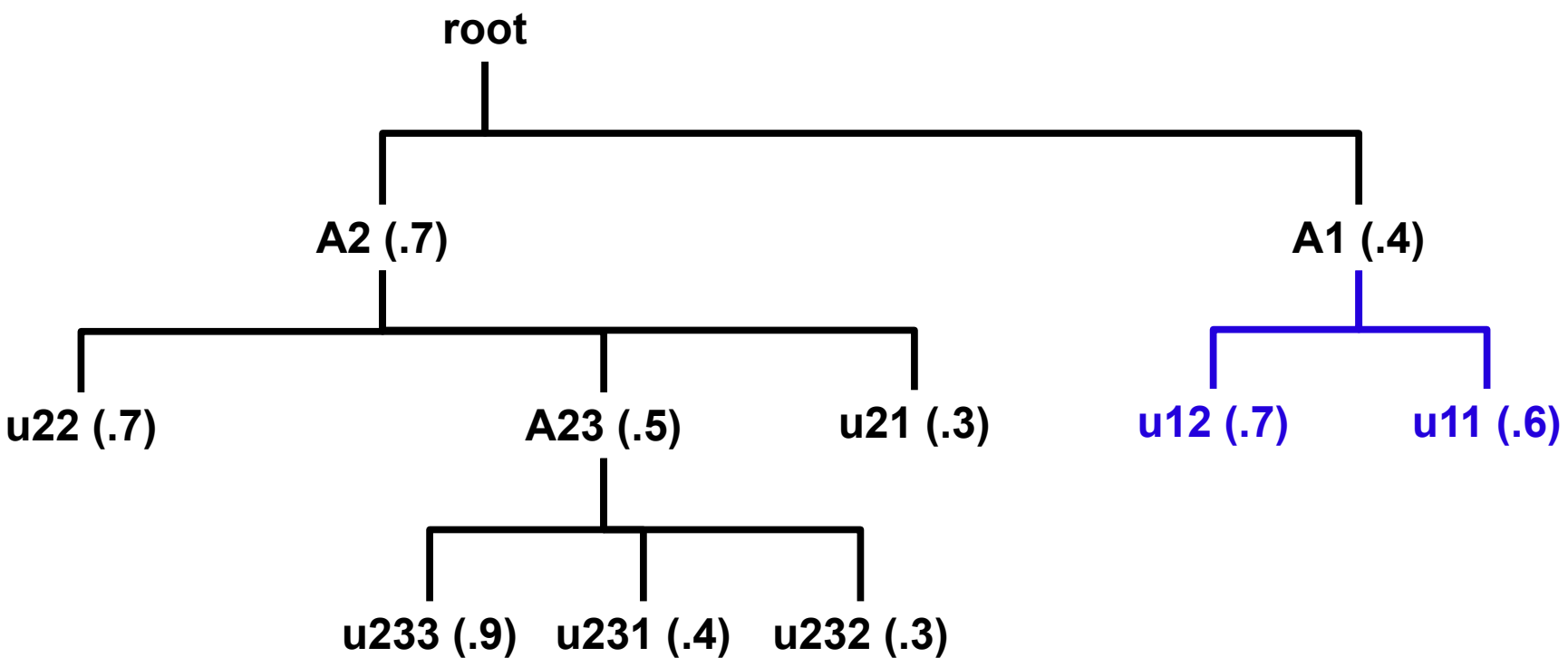
Calculate level fairshare

Fair Tree: Traversal



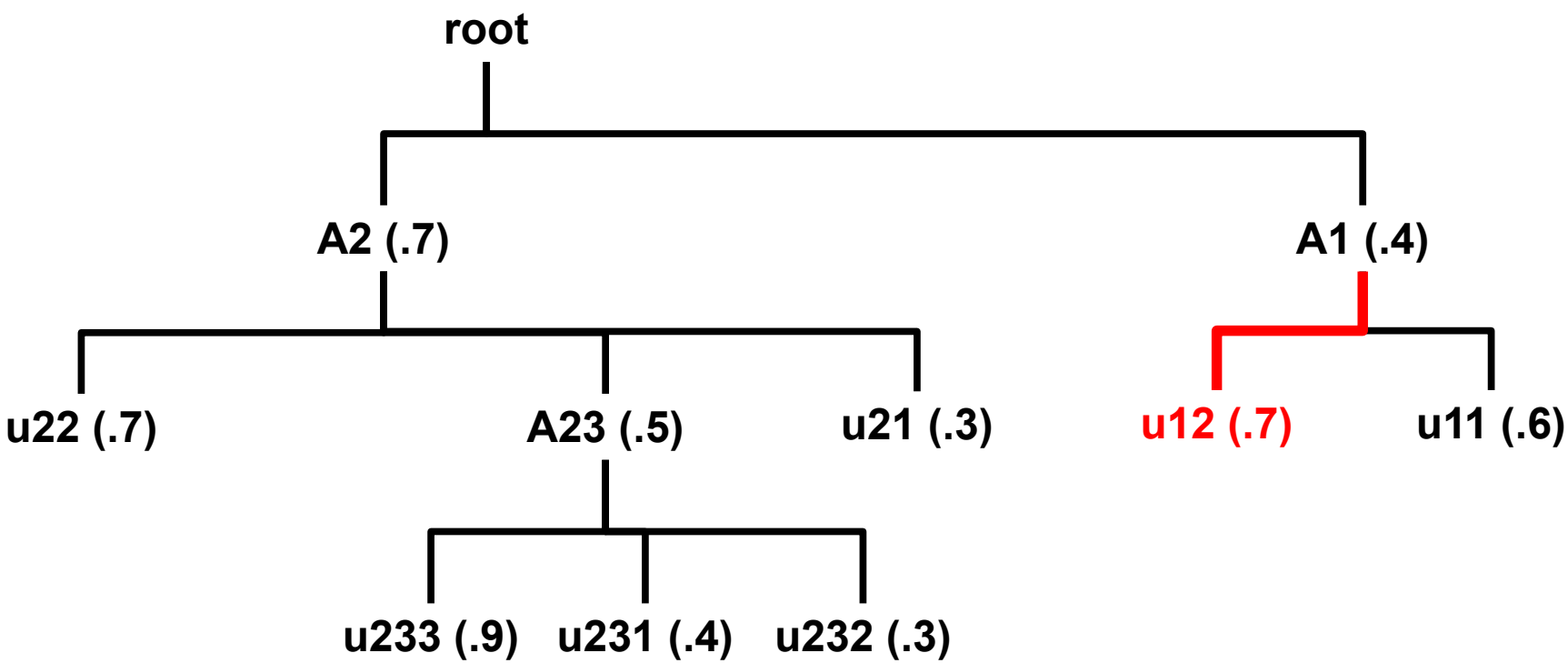
Sort by level fairshare

Fair Tree: Traversal



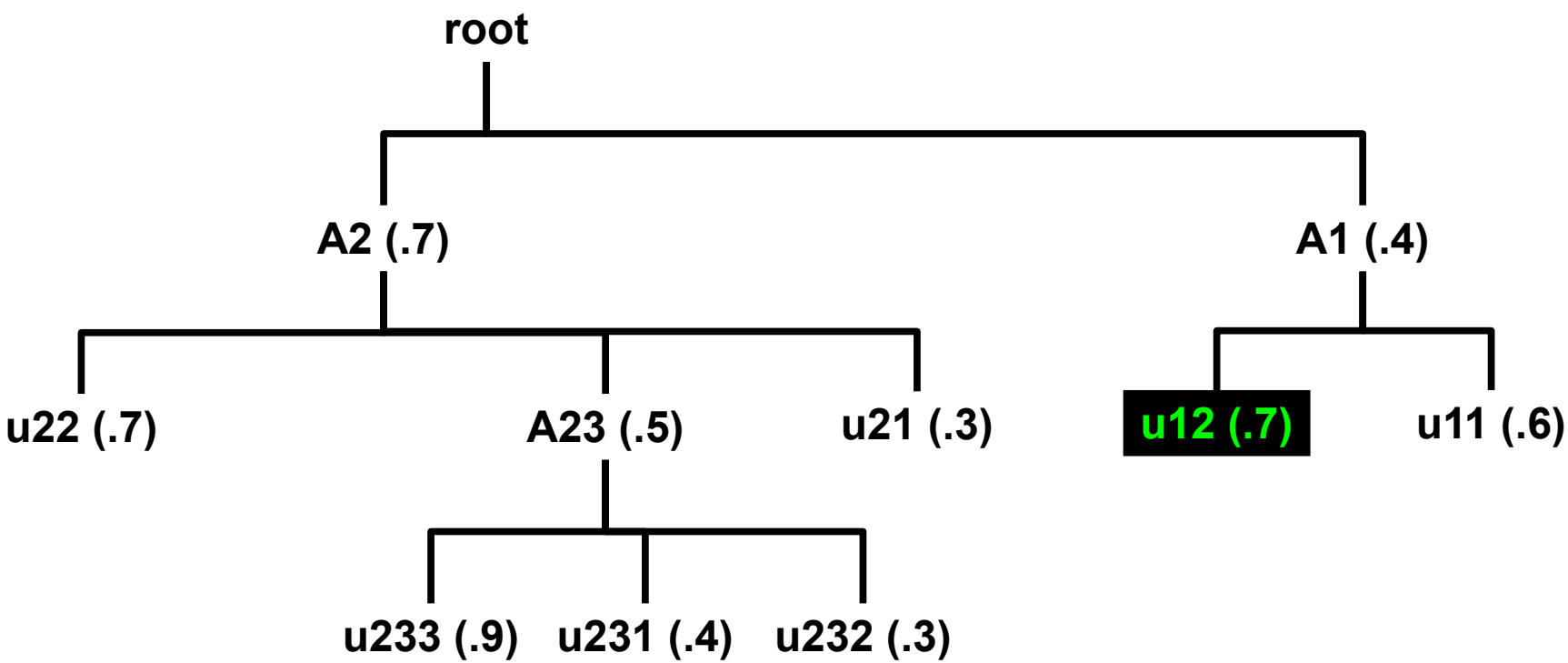
Sort by level fairshare

Fair Tree: Traversal



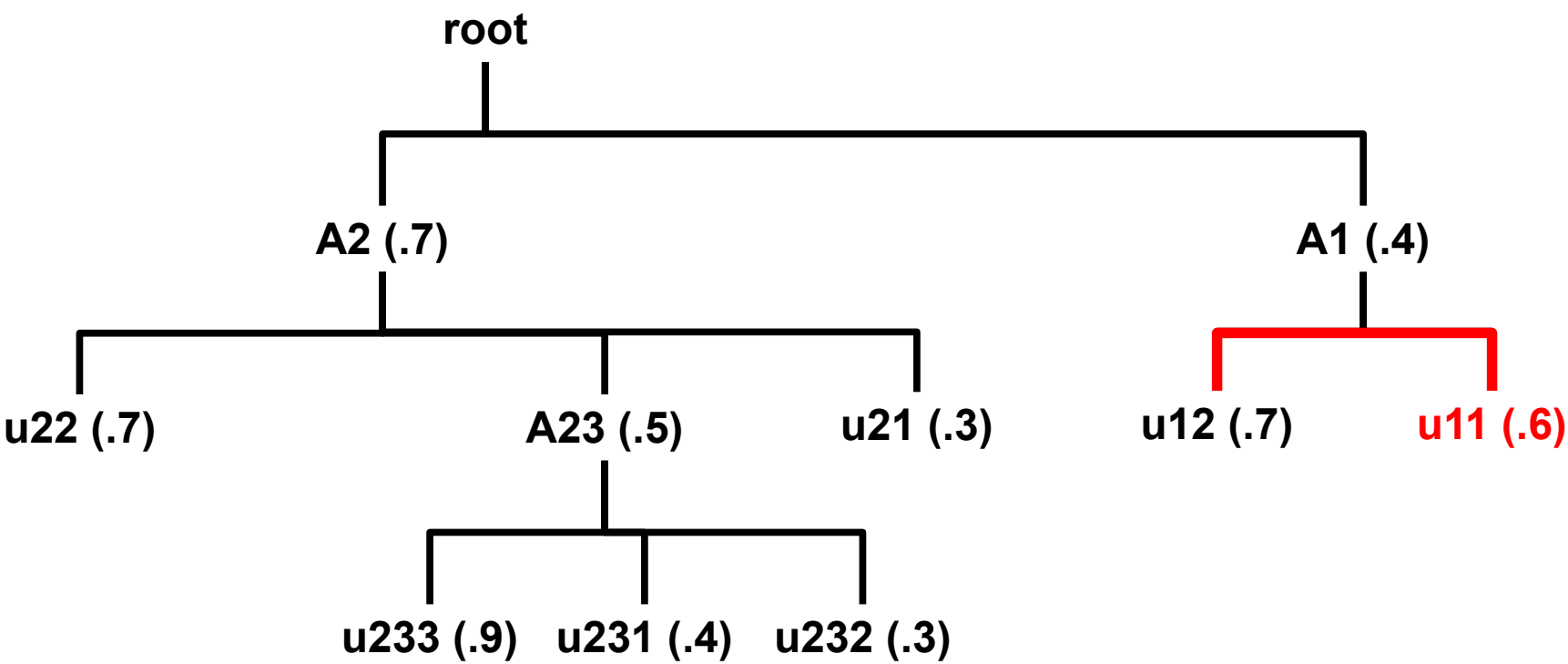
Visit association

Fair Tree: Traversal



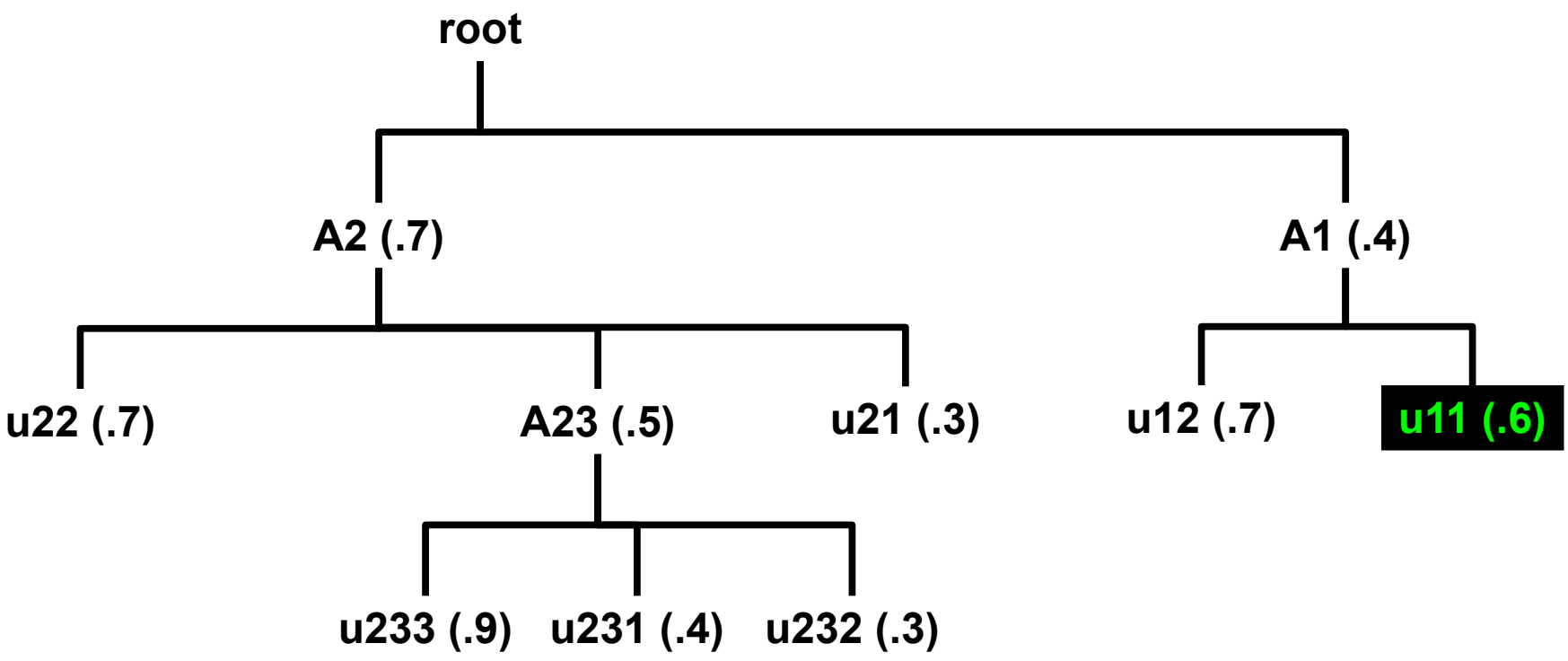
Final fairshare value = rank-- / user_count

Fair Tree: Traversal



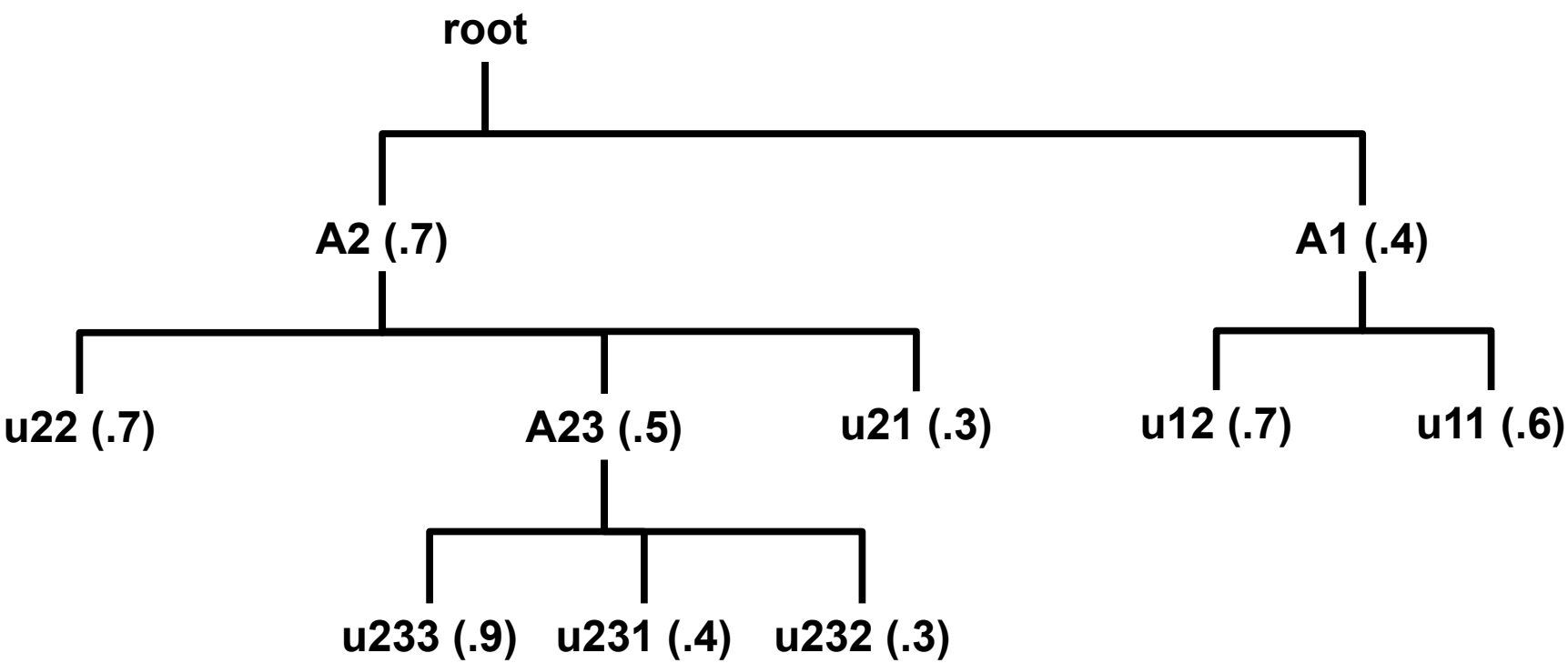
Visit association

Fair Tree: Traversal



Final fairshare value = rank-- / user_count

Fair Tree: Traversal Complete



Traversal Complete!
All final fairshare values were assigned

Ranking

- Users are ranked as they are found
  - Ties are allowed (e.g. 8,8,8,5,4) if Level Fairshare is equal
  - See the appendix for details of tie handling
- Avoids precision loss

sshare

- **Norm Shares** and **Effectv Usage** – reflect the Fair Tree approach

  assoc / (assoc+siblings)
- NEW **Level FS** (*with -l option*) – Fairshare compared to siblings
- **FairShare** – This is the final fairshare factor (i.e. it works like it should)

Account	User	Raw Shares	Norm Shares	Raw Usage	Norm Usage	Effectv Usage	FairShare	Level FS
root			0.000000	1230		1.000000		1.000000
beatles		500	0.500000	676	0.549593	0.549593		0.909763
beatles	harrison	25	0.250000	301	0.244715	0.445266	0.200000	0.561462
beatles	lennon	25	0.250000	102	0.082927	0.150888	0.600000	1.656863
beatles	mccartney	25	0.250000	37	0.030081	0.054734	0.800000	4.567568
beatles	starr	25	0.250000	236	0.191870	0.349112	0.400000	0.716102
elvis		500	0.500000	554	0.450407	0.450407		1.110108
elvis	elvis	1	1.000000	554	0.450407	1.000000	1.000000	1.000000

*Note: Some sshare columns were cut for brevity in this presentation

Debug

```
slurmctld: Fair Tree fairshare algorithm, starting at root:
slurmctld:     elvis (elvis):  1.11010830324909747294
slurmctld:         elvis (elvis):  1.00000000000000000000
slurmctld:     beatles (beatles):  0.90976331360946745562
slurmctld:         mccartney (beatles):  4.56756756756756756785
slurmctld:         lennon (beatles):  1.65686274509803921568
slurmctld:         starr (beatles):  0.71610169491525423724
slurmctld:         harrison (beatles):  0.56146179401993355479
```

- Live view of depth-first traversal
- Users are printed in order of final fairshare factor
- Number shown is Level FS

Easy to Try

- slurm.conf:
  - PriorityFlags=FAIR_TREE
- scontrol reconfigure
  - Wait for next iteration ($PriorityCalcPeriod minutes)
- Available starting in 14.11.0pre6

Advantages

- Uses entry-level computer science (ordered tree)
- Uses entry-level math
  - `S/U` is simple
  - `LevelFS < LevelFS` is simple
- Precision loss issues *extremely* unlikely
- Handles unbalanced trees
- Demonstrably fair

Possible Concerns

- A user can use all the account's shares if no one else is running
  - (Great!)
  - No algorithm can *prevent* this
  - User limits
    - Set by admin or account coordinator

Possible Concerns

- A user can use all the account's shares if no one else is running
  - (Great!)
  - No algorithm can *prevent* this
  - User limits
    - Set by admin or account coordinator
- Tiny user in a huge group has trouble running
  - Optional QOS w/UsageFactor=20 and Priority=100000000
    - User runs quickly but pays heavily
  - Secondary account with tight limits

Possible Concerns

- A user can use all the account's shares if no one else is running
  - (Great!)
  - No algorithm can *prevent* this
  - User limits
    - Set by admin or account coordinator
- Tiny user in a huge group has trouble running
  - Optional QOS w/UsageFactor=20 and Priority=100000000
    - User runs quickly but pays heavily
  - Secondary account with tight limits
- Strictness of hierarchical prioritization
  - Fairshare=parent on accounts
  - Fuzzy matching (see “Future Development”)

Fairshare=parent modifications (all algorithms)

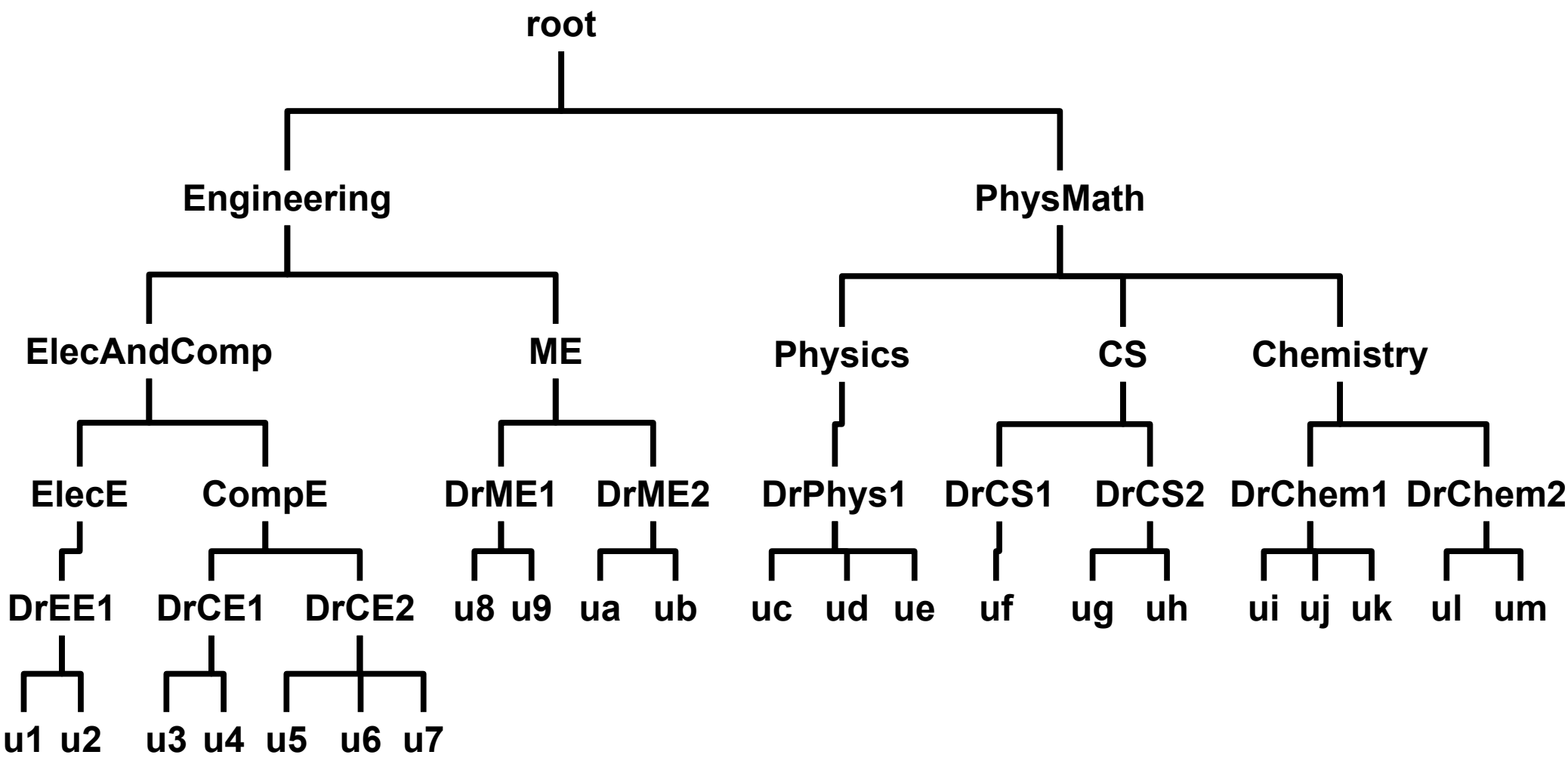
Fairshare=parent

- Fairshare=parent behavior on an <u>account</u> was previously <u>undefined</u>
- It is now defined:

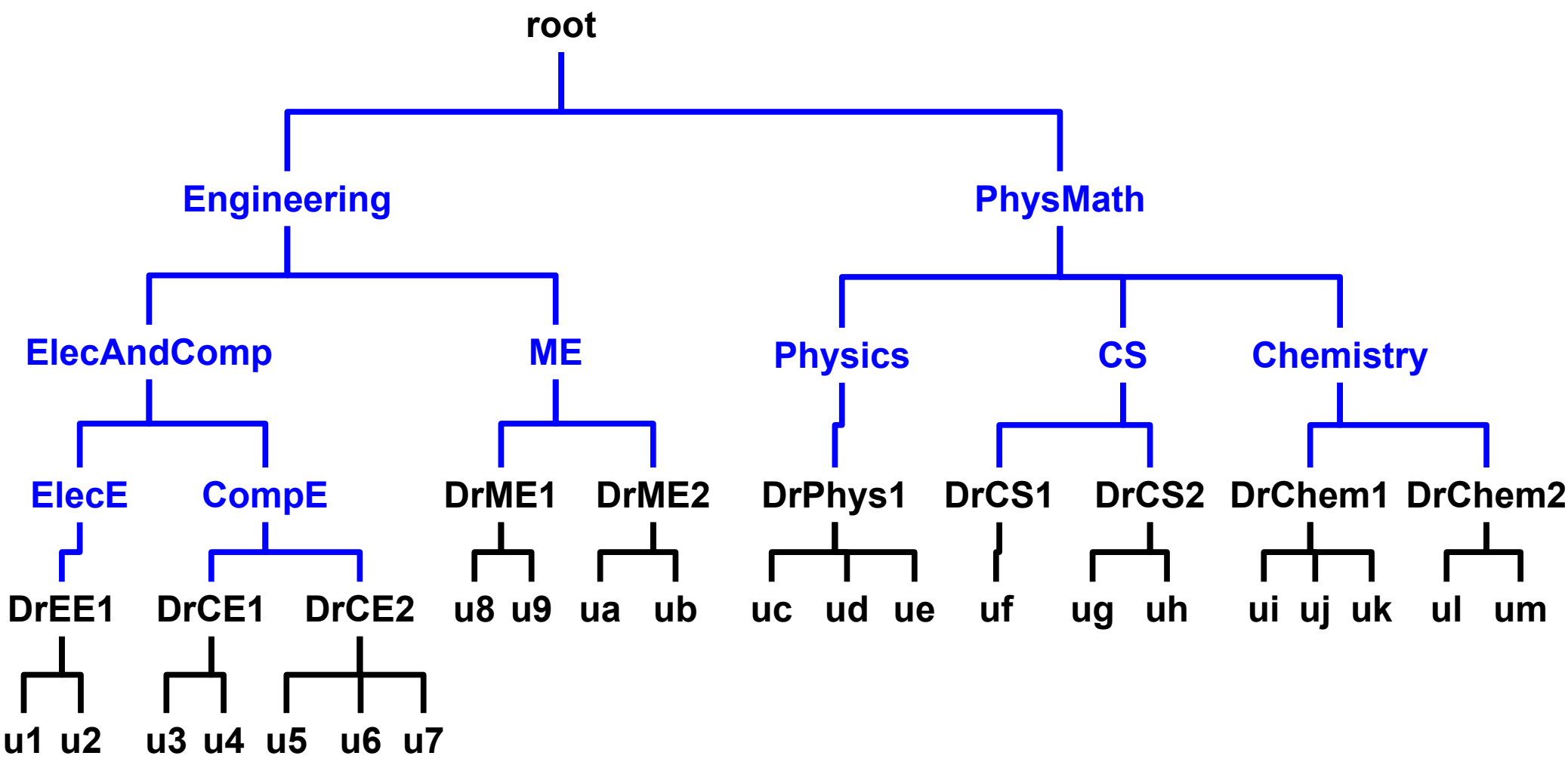
If Fairshare=parent is set on an account, that account's children will be effectively reparented for fairshare calculations to the first ancestor that is not Fairshare=parent.

- The behavior of limits is <u>unchanged</u>
- Available in 14.11 **<u>for all algorithms</u>**

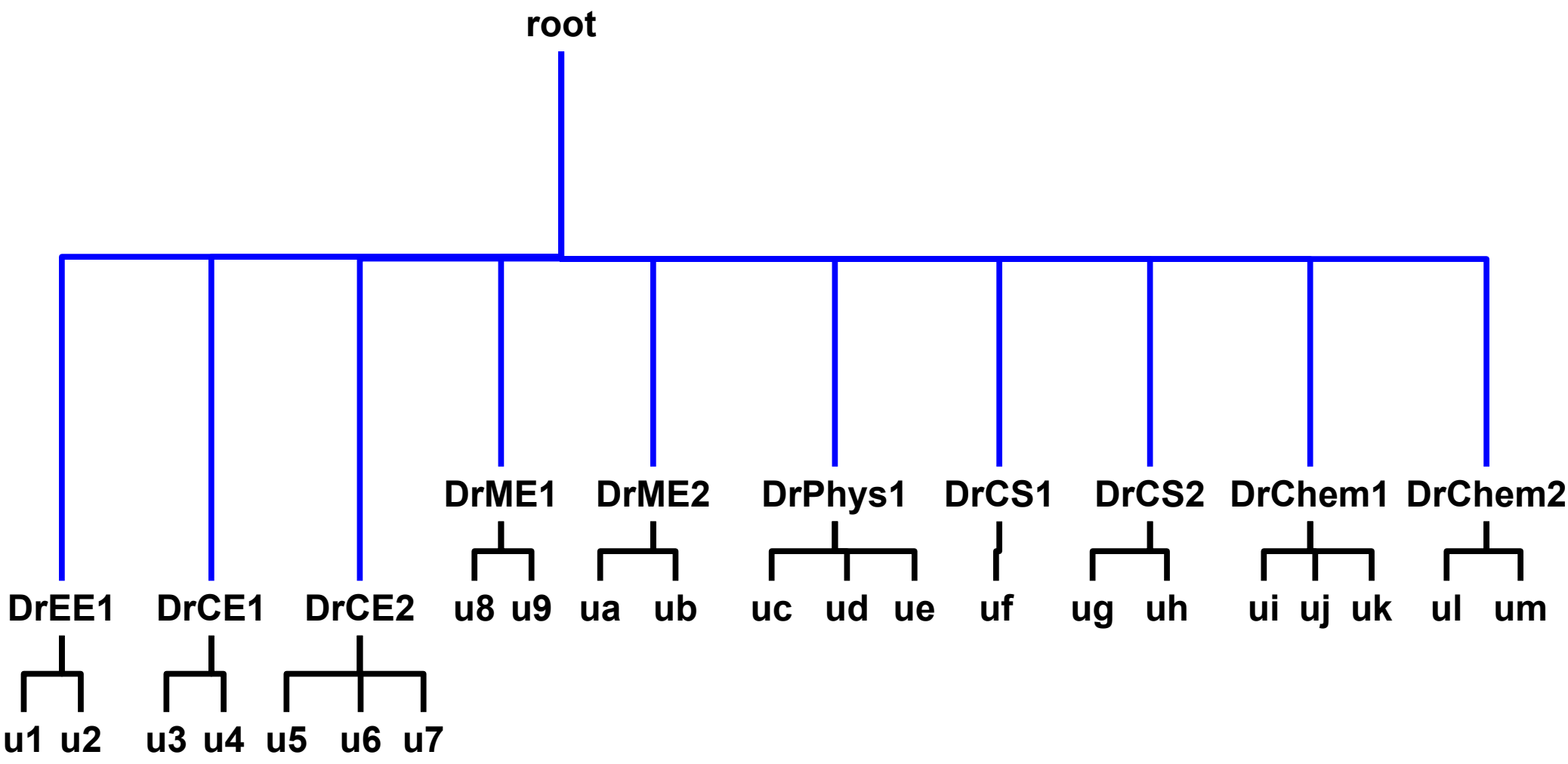
Fairshare=parent



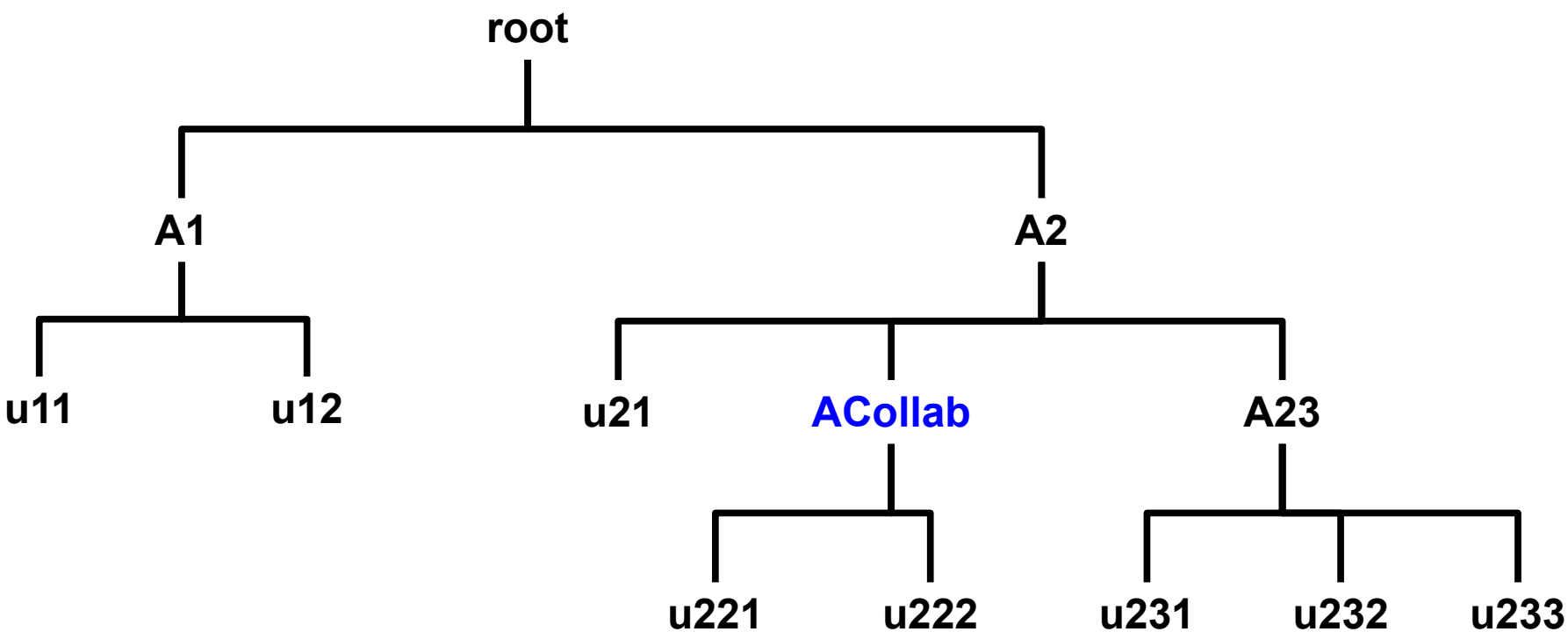
Fairshare=parent



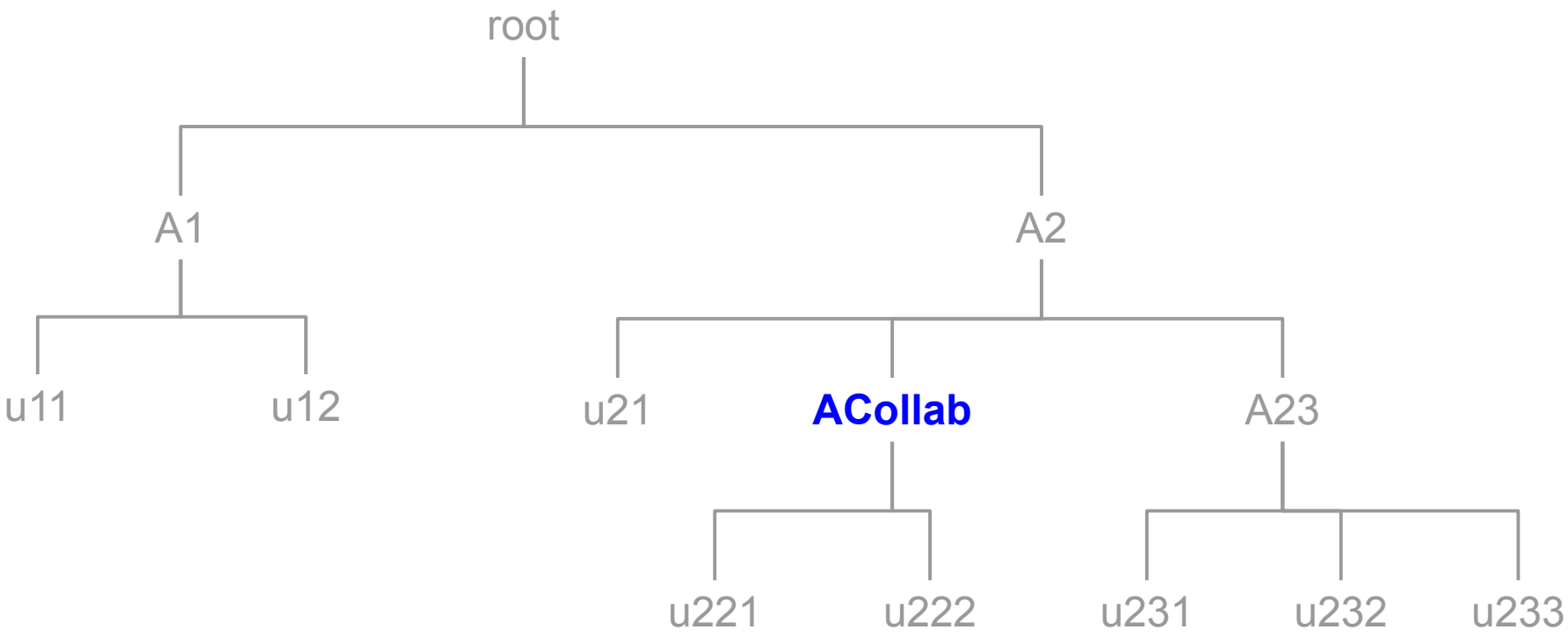
Fairshare=parent



Fairshare=parent

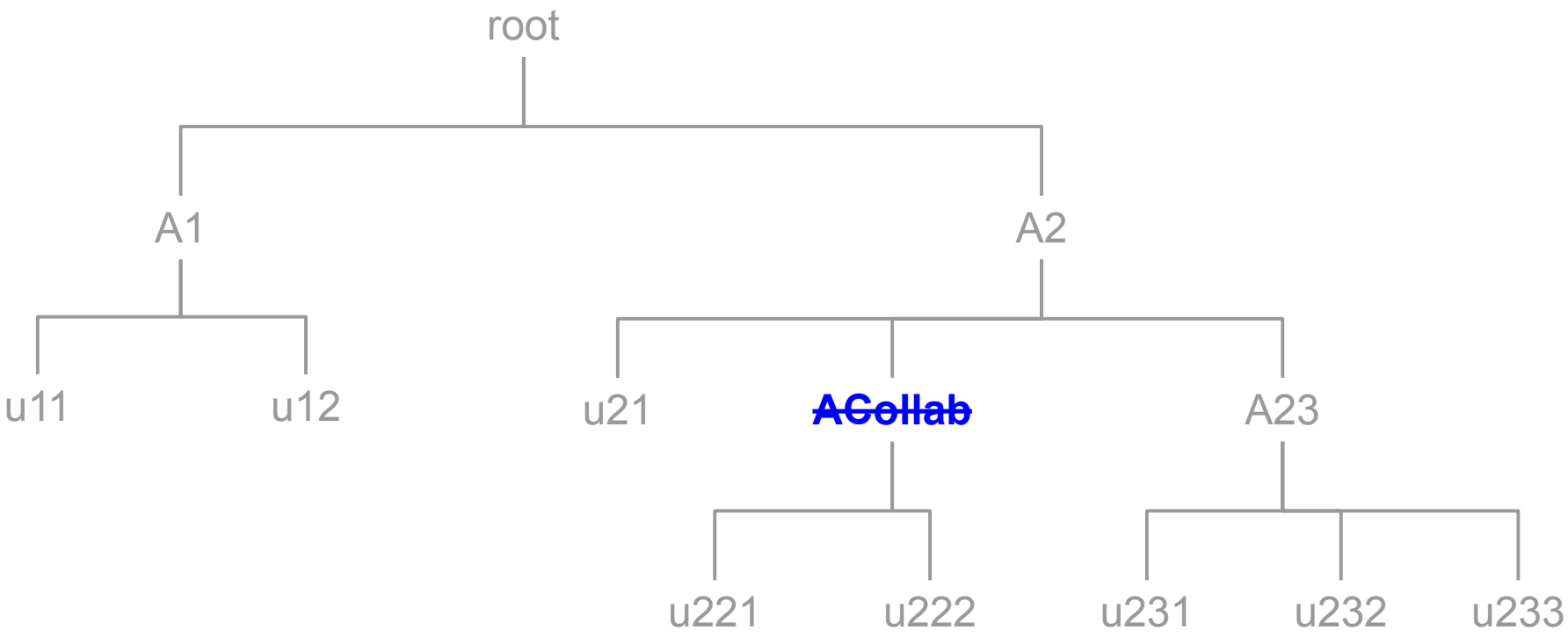


Fairshare=parent



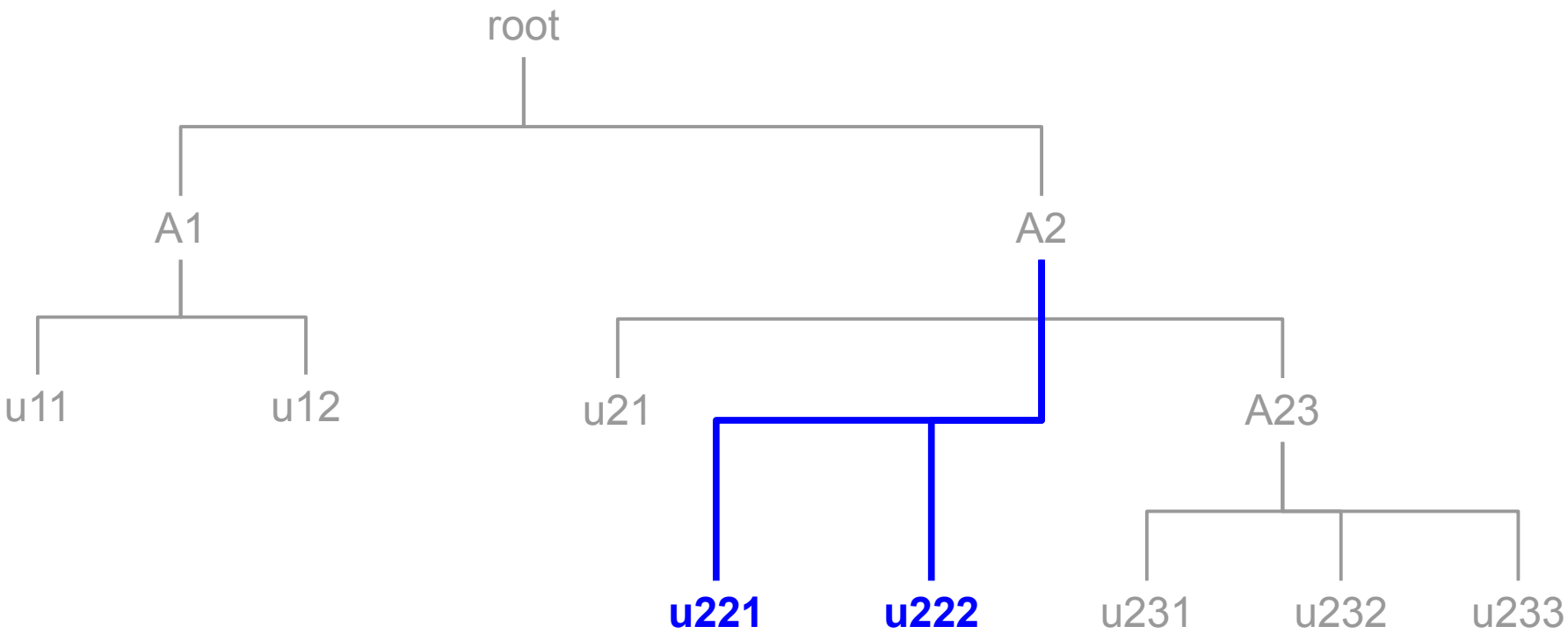
Example:
ACollab is the only association with Fairshare=parent

Fairshare=parent



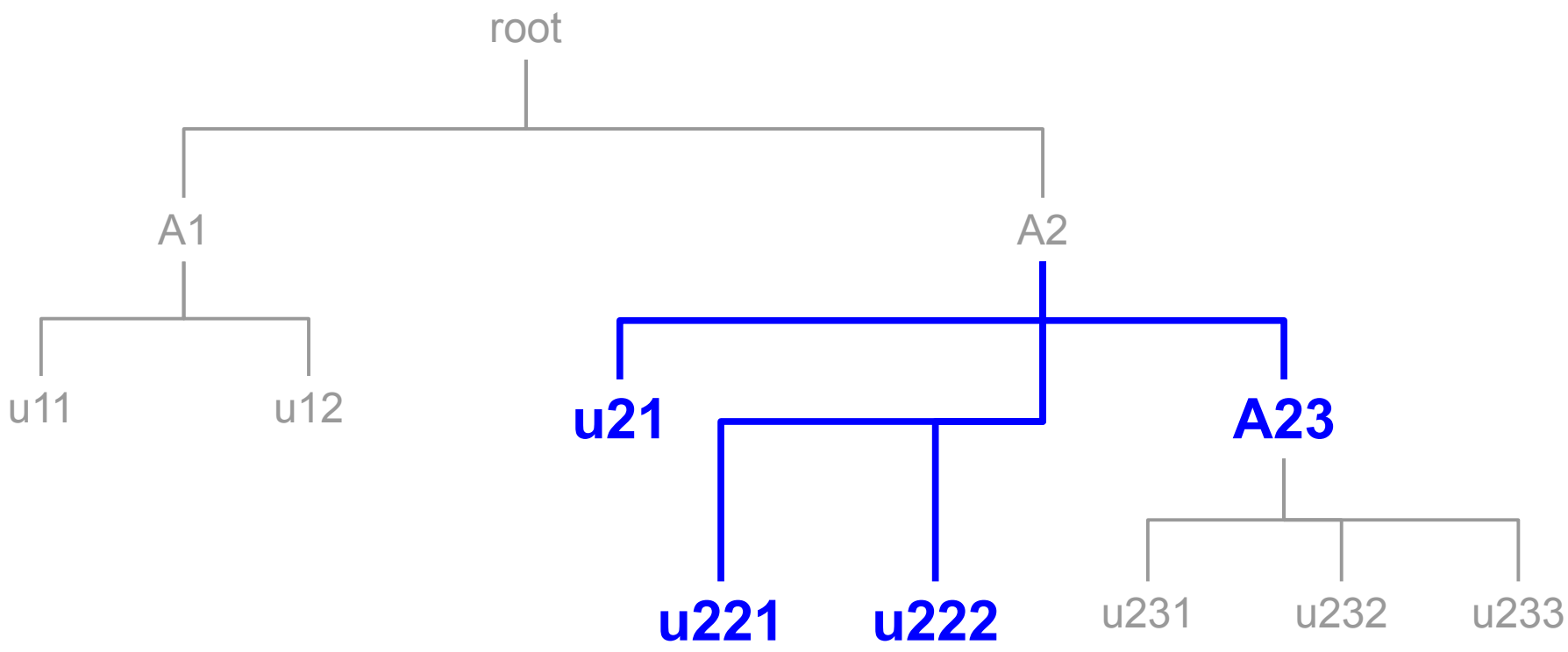
Example:
ACollab effectively disappears

Fairshare=parent



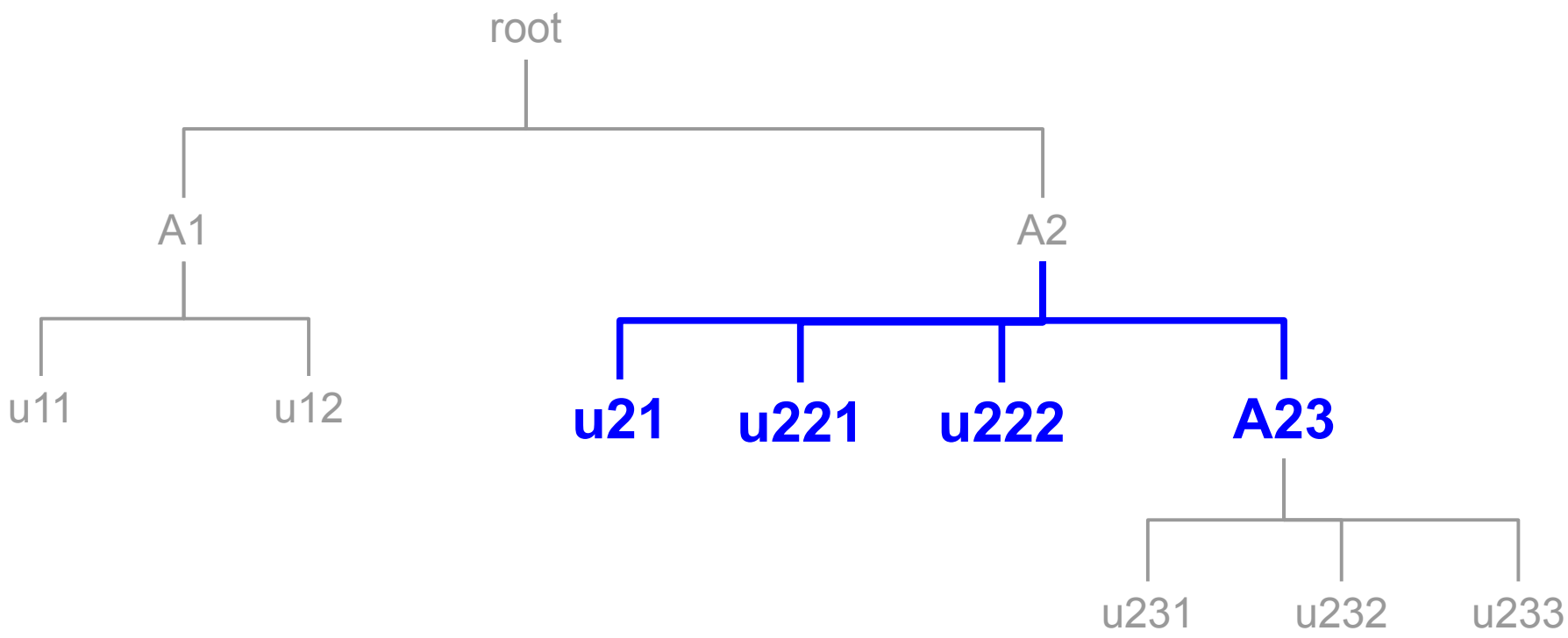
Example:
u221 and **u222** are reparented

Fairshare=parent



The following are now **siblings** for purposes of fairshare calculations:
u21, u221, u222, A23

Fairshare=parent



The following are now **siblings** for purposes of fairshare calculations:
u21, u221, u222, A23

Questions?

Appendix

Appendix Index

- More information
- Future Development
- Problems with Shares Calculation
- Linear Interpolation
- sshare example
- Tie Handling

More Information

- Until 14.11 is released, Fair Tree documentation is at:
  - https://fsl.byu.edu/documentation/slurm/fair_tree.php
- Available in 14.11.0pre6

Future Development

- Allow for less strictness
  - Merge accounts based on configurable delta
    - Allows reuse of tie handling. The comparison function needs a very minor change
  - LevelFSTieDelta=0.01,0.05,0.02
    - Values are the epsilon value at increasing depths
  - Either:
    - abs((a-b)<delta)
    - a>b*(1-delta) && a<b*(1+delta)

Problems With Shares Calculation

- Normalized Shares equation for all current algorithms:

$$S = (S_{user} / S_{siblings}) * (S_{account} / S_{sibling\text{-}accounts}) * (S_{parent} / S_{parent\text{-}siblings}) * \ldots$$

Problems With Shares Calculation

- Normalized Shares equation for all current algorithms:

$$S = (S_{user} / S_{siblings}) * (S_{account} / S_{sibling\text{-}accounts}) * (S_{parent} / S_{parent\text{-}siblings}) * \ldots$$

Example:

- The tree is no deeper than: root->account->user
- Assume sibling associations are treated equally (same Shares)
- Problem still applies for more complicated scenarios

Problems With Shares Calculation

- Normalized Shares equation for all current algorithms:

  $S = (S_{user} / S_{siblings}) *$

  $(S_{account} / S_{sibling\text{-}accounts}) *$

  $\cancel{(S_{parent} / S_{parent\text{-}siblings}) * \ldots}$

- The tree is no deeper than: root->account->user
- Therefore:

  $S = (S_{user} / S_{siblings}) * (S_{account} / S_{sibling\text{-}accounts})$

Problems With Shares Calculation

- Normalized Shares equation for all current algorithms:

  $S = (S_{user} / S_{siblings}) *$

  $(S_{account} / S_{sibling\text{-}accounts}) *$

  ~~$(S_{parent} / S_{parent\text{-}siblings}) * \ldots$~~

- If each account is Shares=500
- $S_{account} / S_{sibling\text{-}accounts} = 500 / (500 * \text{count(accounts)})$
- $S_{account} / S_{sibling\text{-}accounts} = 1 / \text{count(accounts)}$
- **This is a constant for <u>all</u> accounts**

Problems With Shares Calculation

- Normalized Shares equation for all current algorithms:

  $S = (S_{user} / S_{siblings}) * constant$

  ~~$(S_{account} / S_{sibling\text{-}accounts}) *$~~

  ~~$(S_{parent} / S_{parent\text{-}siblings}) * \ldots$~~

- $S = (S_{user} / S_{siblings}) * constant$
- Since we are comparing... constants are thrown out:
- $S = (S_{user} / S_{siblings})$

Problems With Shares Calculation

- Normalized Shares equation for all current algorithms:

  $S = (S_{user} / S_{siblings})$ ~~* constant~~

  ~~$(S_{account} / S_{sibling\text{-}accounts}) *$~~

  ~~$(S_{parent} / S_{parent\text{-}siblings}) * \ldots$~~

- If each user is Shares=100
- S = 100 / (100 * count(users_in_account))
- S = 1 / count(users_in_account)

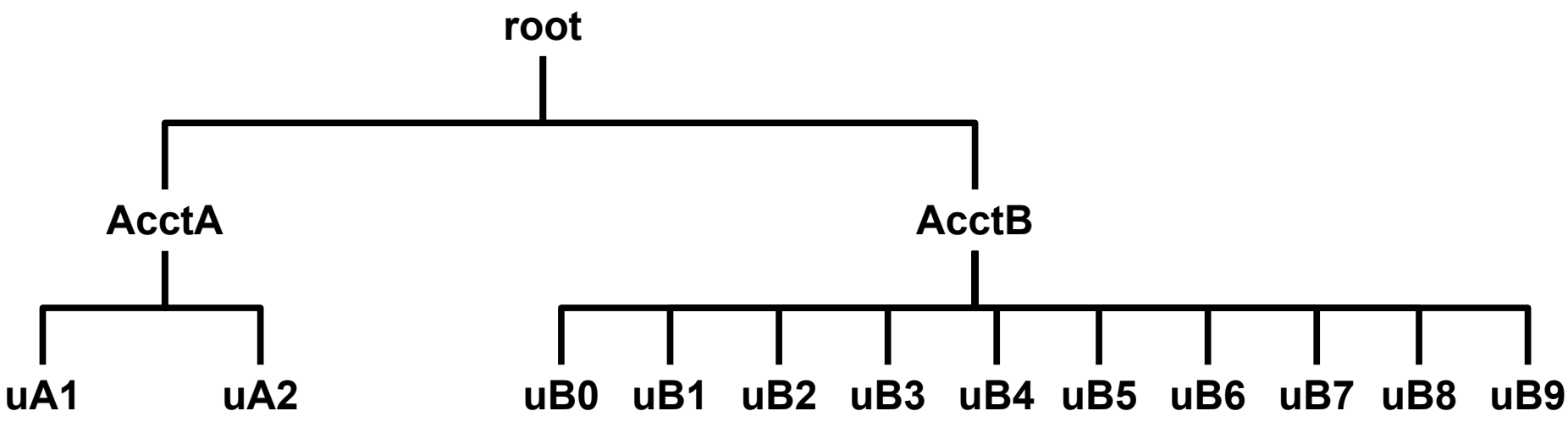
Problems With Shares Calculation

- Normalized Shares equation for all current algorithms when accounts are treated equally and users in an account are treated equally:

$$S_{user} = 1 / \text{count}(\text{users\_in\_account}) * \text{constant}$$

The problem still exists for other scenarios but is harder to model

Problems With Shares Calculation



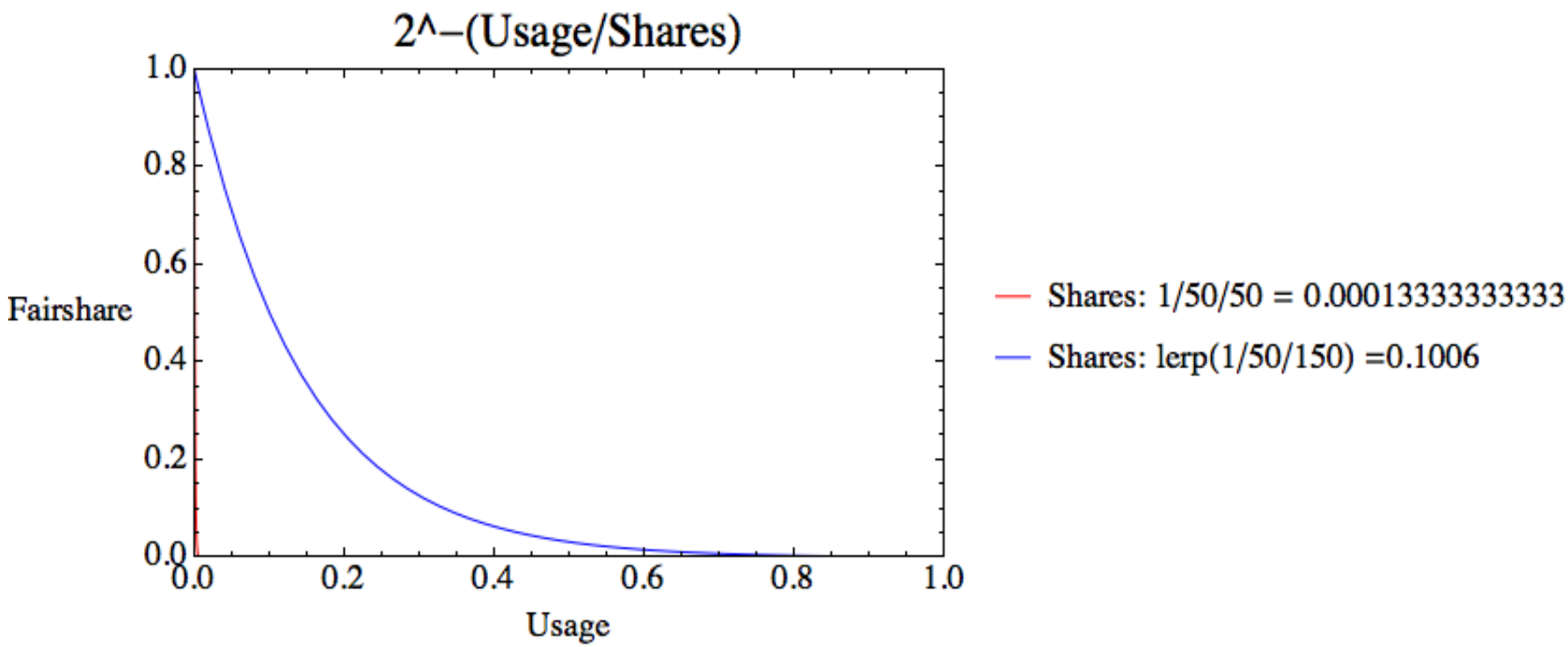
S = 1/2 = 0.5

S = 1/10 = 0.1

Linear Interpolation (lerp)

- Shares are between 0.0 .. 1.0
- Map them to the range 0.1 .. 1.0
  - Use linear interpolation (lerp)
- lerp(min, max, f) = min + f * (max – min)
  - Has floating point issues
- lerp(min, max, f) = min * (1.0L - f) + max * f
  - Shouldn't have floating point issues
- lerp(0.1, 1.0, 0.00001) = 0.100009

Linear Interpolation (lerp)



Unmodified vs lerp()

Usage	Shares	Fairshare (unmodified)	Fairshare (lerp)
0.15	.2	0.5946035575013606244709762904676 6	0.6898170601727024775539146017600 0
0.15	.1	0.3535533905932738418798066959425 3	0.5785551186413599328317060310755 7
0.15	.02	0.0055242717280199029120102416357 0	0.4143189060079367603368558081733 4
0.15	.01	0.0000305175781250000000000000000 0	0.3852463547649099601306345308504 6
0.15	.002	0.0000000000000000000000264697796 0	0.3601132513002794519551240381982 6
0.15	.001	0.0000000000000000000000000000000 0	0.3568475032991250242607808584871 5
0.15	.0002	0.0000000000000000000000000000000 0	0.3542144932886342472582764939748 0
0.15	.0001	0.0000000000000000000000000000000 0	0.3538840847911395129680653393045 0
0.15	.00000000002	0.0000000000000000000000000000000 0	0.3535533906594412004999964521667 6
0.15	.00000000001	0.0000000000000000000000000000000 0	0.3535533906263574881623928947149 7

- Fairshare factor is multiplied by PriorityWeightFairshare, making the problem much worse
- If lerp() is implemented, add an option (-L?) to sshare:
  - change Norm Shares → Lerp Shares

sshare

- **Norm Shares** and **Effectv Usage** – reflect the Fair Tree approach

  assoc / (assoc+siblings)
- NEW **Level FS** (*with -l option*) – Fairshare compared to siblings
- **FairShare** – This is the final fairshare factor (i.e. it works like it should)

Account	User	Raw Shares	Norm Shares	Raw Usage	Norm Usage	Effectv Usage	FairShare	Level FS
root			0.000000	1230		1.000000		1.000000
beatles		500	0.500000	676	0.549593	0.549593		0.909763
beatles	harrison	25	0.250000	301	0.244715	0.445266	0.200000	0.561462
beatles	lennon	25	0.250000	102	0.082927	0.150888	0.600000	1.656863
beatles	mccartney	25	0.250000	37	0.030081	0.054734	0.800000	4.567568
beatles	starr	25	0.250000	236	0.191870	0.349112	0.400000	0.716102
elvis		500	0.500000	554	0.450407	0.450407		1.110108
elvis	elvis	1	1.000000	554	0.450407	1.000000	1.000000	1.000000

McCartney: Why do I have a lower fairshare factor than Elvis?

*Note: Some sshare columns were cut for brevity in this presentation

sshare

- **Norm Shares** and **Effectv Usage** – reflect the Fair Tree approach

  assoc / (assoc+siblings)
- NEW **Level FS** (*with -l option*) – Fairshare compared to siblings
- **FairShare** – This is the final fairshare factor (i.e. it works like it should)

Account	User	Raw Shares	Norm Shares	Raw Usage	Norm Usage	Effectv Usage	FairShare	Level FS
root			0.000000	1230		1.000000		1.000000
beatles		500	0.500000	676	0.549593	0.549593		0.909763
beatles	harrison	25	0.250000	301	0.244715	0.445266	0.200000	0.561462
beatles	lennon	25	0.250000	102	0.082927	0.150888	0.600000	1.656863
beatles	mccartney	25	0.250000	37	0.030081	0.054734	0.800000	4.567568
beatles	starr	25	0.250000	236	0.191870	0.349112	0.400000	0.716102
elvis		500	0.500000	554	0.450407	0.450407		1.110108
elvis	elvis	1	1.000000	554	0.450407	1.000000	1.000000	1.000000

McCartney: Why do I have a lower fairshare factor than Elvis?

*Note: Some sshare columns were cut for brevity in this presentation

sshare

- **Norm Shares** and **Effectv Usage** – reflect the Fair Tree approach

  assoc / (assoc+siblings)
- NEW **Level FS** (*with -l option*) – Fairshare compared to siblings
- **FairShare** – This is the final fairshare factor (i.e. it works like it should)

Account	User	Raw Shares	Norm Shares	Raw Usage	Norm Usage	Effectv Usage	FairShare	Level FS
root			0.000000	1230		1.000000		1.000000
beatles		500	0.500000	676	0.549593	0.549593		0.909763
beatles	harrison	25	0.250000	301	0.244715	0.445266	0.200000	0.561462
beatles	lennon	25	0.250000	102	0.082927	0.150888	0.600000	1.656863
beatles	mccartney	25	0.250000	37	0.030081	0.054734	**0.800000**	4.567568
beatles	starr	25	0.250000	236	0.191870	0.349112	0.400000	0.716102
elvis		500	0.500000	554	0.450407	0.450407		1.110108
elvis	elvis	1	1.000000	554	0.450407	1.000000	**1.000000**	1.000000

McCartney: Why do I have a lower fairshare factor than Elvis?

*Note: Some sshare columns were cut for brevity in this presentation

sshare

- **Norm Shares** and **Effectv Usage** – reflect the Fair Tree approach

  assoc / (assoc+siblings)
- NEW **Level FS** (*with -l option*) – Fairshare compared to siblings
- **FairShare** – This is the final fairshare factor (i.e. it works like it should)

Account	User	Raw Shares	Norm Shares	Raw Usage	Norm Usage	Effectv Usage	FairShare	Level FS
root			0.000000	1230		1.000000		1.000000
beatles		500	0.500000	676	0.549593	0.549593		0.909763
beatles	harrison	25	0.250000	301	0.244715	0.445266	0.200000	0.561462
beatles	lennon	25	0.250000	102	0.082927	0.150888	0.600000	1.656863
beatles	mccartney	25	0.250000	37	0.030081	0.054734	0.800000	4.567568
beatles	starr	25	0.250000	236	0.191870	0.349112	0.400000	0.716102
elvis		500	0.500000	554	0.450407	0.450407		1.110108
elvis	elvis	1	1.000000	554	0.450407	1.000000	1.000000	1.000000

Answer: Compare the accounts' Level FS values

*Note: Some sshare columns were cut for brevity in this presentation

sshare

- **Norm Shares** and **Effectv Usage** – reflect the Fair Tree approach

  assoc / (assoc+siblings)
- NEW **Level FS** (*with -l option*) – Fairshare compared to siblings
- **FairShare** – This is the final fairshare factor (i.e. it works like it should)

Account	User	Raw Shares	Norm Shares	Raw Usage	Norm Usage	Effectv Usage	FairShare	Level FS
root			0.000000	1230		1.000000		1.000000
beatles		500	0.500000	676	0.549593	0.549593		**0.909763**
beatles	harrison	25	0.250000	301	0.244715	0.445266	0.200000	0.561462
beatles	lennon	25	0.250000	102	0.082927	0.150888	0.600000	1.656863
beatles	mccartney	25	0.250000	37	0.030081	0.054734	0.800000	4.567568
beatles	starr	25	0.250000	236	0.191870	0.349112	0.400000	0.716102
elvis		500	0.500000	554	0.450407	0.450407		**1.110108**
elvis	elvis	1	1.000000	554	0.450407	1.000000	1.000000	1.000000

Answer: Compare the accounts' Level FS values

*Note: Some sshare columns were cut for brevity in this presentation

sshare

- **Norm Shares** and **Effectv Usage** – reflect the Fair Tree approach

  assoc / (assoc+siblings)
- NEW **Level FS** (*with -l option*) – Fairshare compared to siblings
- **FairShare** – This is the final fairshare factor (i.e. it works like it should)

Account	User	Raw Shares	Norm Shares	Raw Usage	Norm Usage	Effectv Usage	FairShare	Level FS
root			0.000000	1230		1.000000		1.000000
beatles		500	0.500000	676	0.549593	0.549593		**0.909763**
beatles	harrison	25	0.250000	301	0.244715	0.445266	0.200000	0.561462
beatles	lennon	25	0.250000	102	0.082927	0.150888	0.600000	1.656863
beatles	mccartney	25	0.250000	37	0.030081	0.054734	0.800000	4.567568
beatles	starr	25	0.250000	236	0.191870	0.349112	0.400000	0.716102
elvis		500	0.500000	554	0.450407	0.450407		**1.110108**
elvis	elvis	1	1.000000	554	0.450407	1.000000	1.000000	1.000000

McCartney: How is Level FS calculated?

*Note: Some sshare columns were cut for brevity in this presentation

sshare

- **Norm Shares** and **Effectv Usage** – reflect the Fair Tree approach

  assoc / (assoc+siblings)
- NEW **Level FS** (*with -l option*) – Fairshare compared to siblings
- **FairShare** – This is the final fairshare factor (i.e. it works like it should)

Account	User	Raw Shares	Norm Shares	Raw Usage	Norm Usage	Effectv Usage	FairShare	Level FS
root			0.000000	1230		1.000000		1.000000
beatles		500	**0.500000**	676	0.549593	**0.549593**		0.909763
beatles	harrison	25	0.250000	301	0.244715	0.445266	0.200000	0.561462
beatles	lennon	25	0.250000	102	0.082927	0.150888	0.600000	1.656863
beatles	mccartney	25	0.250000	37	0.030081	0.054734	0.800000	4.567568
beatles	starr	25	0.250000	236	0.191870	0.349112	0.400000	0.716102
elvis		500	0.500000	554	0.450407	0.450407		1.110108
elvis	elvis	1	1.000000	554	0.450407	1.000000	1.000000	1.000000

Answer: Norm Shares / Effectv Usage

*Note: Some sshare columns were cut for brevity in this presentation

sshare

- **Norm Shares** and **Effectv Usage** – reflect the Fair Tree approach

  assoc / (assoc+siblings)
- NEW **Level FS** (*with -l option*) – Fairshare compared to siblings
- **FairShare** – This is the final fairshare factor (i.e. it works like it should)

Account	User	Raw Shares	Norm Shares	Raw Usage	Norm Usage	Effectv Usage	FairShare	Level FS
root			0.000000	1230		1.000000		1.000000
beatles		500	0.500000	676	0.549593	0.549593		0.909763
beatles	harrison	25	0.250000	301	0.244715	0.445266	0.200000	0.561462
beatles	lennon	25	0.250000	102	0.082927	0.150888	0.600000	1.656863
beatles	mccartney	25	0.250000	37	0.030081	0.054734	0.800000	4.567568
beatles	starr	25	0.250000	236	0.191870	0.349112	0.400000	0.716102
elvis		500	0.500000	554	0.450407	0.450407		1.110108
elvis	elvis	1	1.000000	554	0.450407	1.000000	1.000000	1.000000

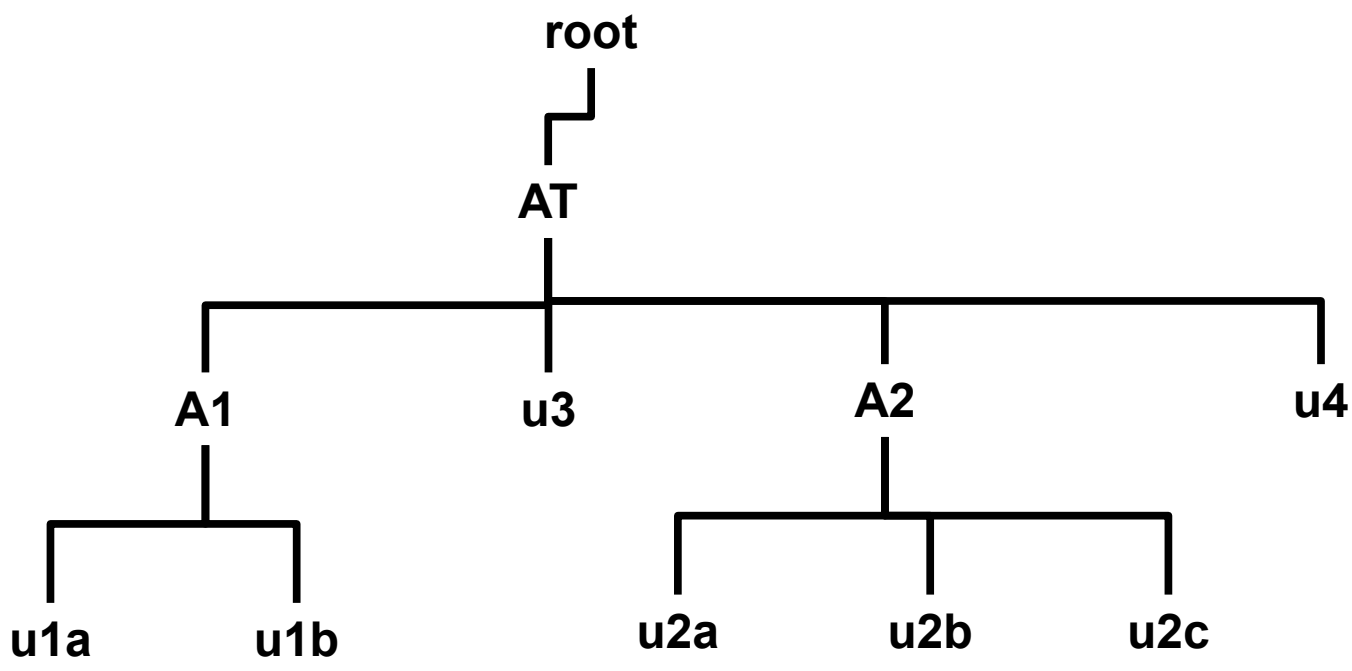
Answer: Norm Shares / Effectv Usage

*Note: Some sshare columns were cut for brevity in this presentation

Ties

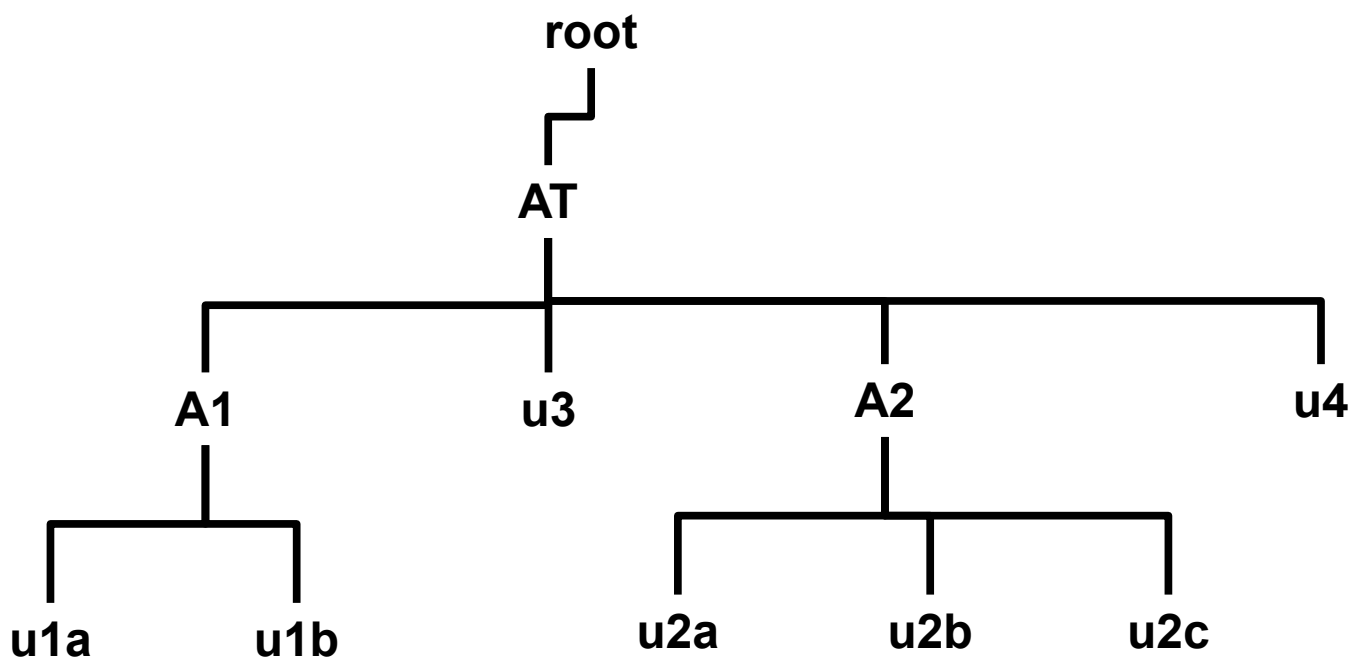
- Occur when sibling associations have the same level_fs
  - Extremely rare except when raw usage == 0.0
  - Added complexity to the code
- Rules:
  - Sibling **users** receive the same rank
  - Sibling **accounts** have their children lists merged
  - A **user** with the same level_fs as a sibling **account** will receive the same rank as the account's highest ranked user

Fair Tree: Tie Handling



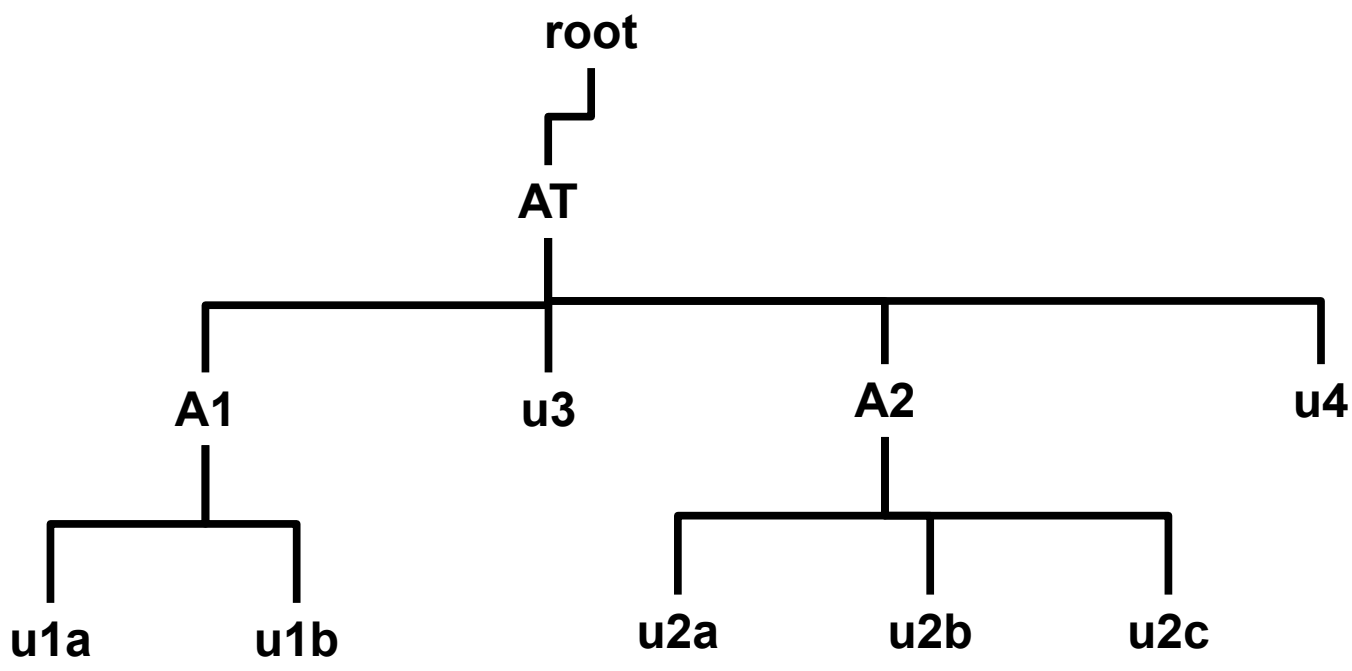
This is one portion of the tree. AT is pictured in its entirety.

Fair Tree: Tie Handling



There are 83 users (not all pictured).
Some have been visited.

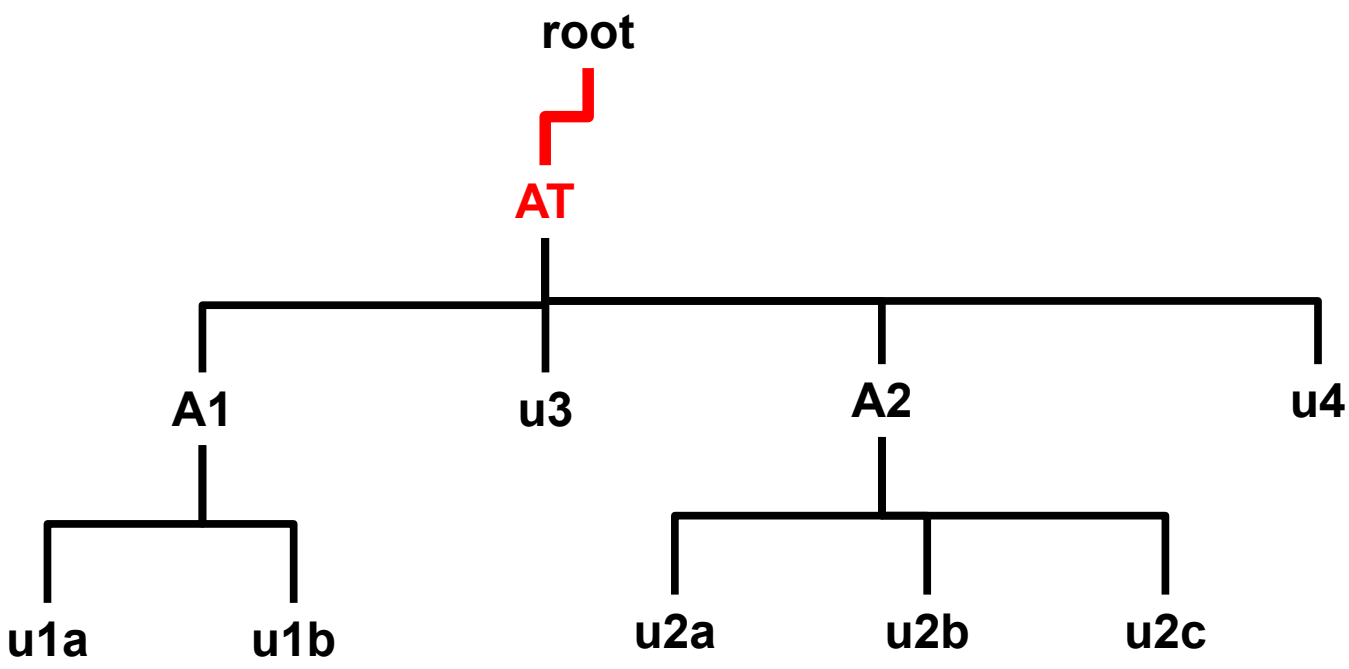
Fair Tree: Tie Handling



There are 83 users (not all pictured).
Some have been visited.

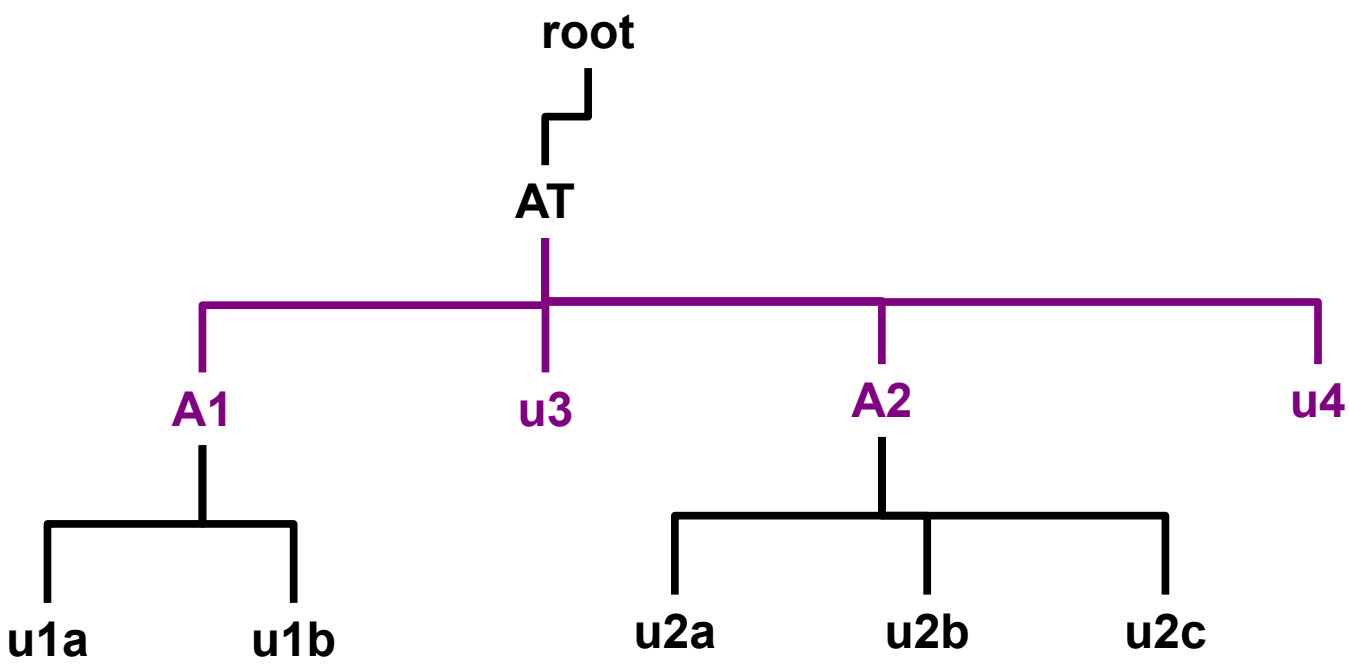
rank = 64

Fair Tree: Tie Handling



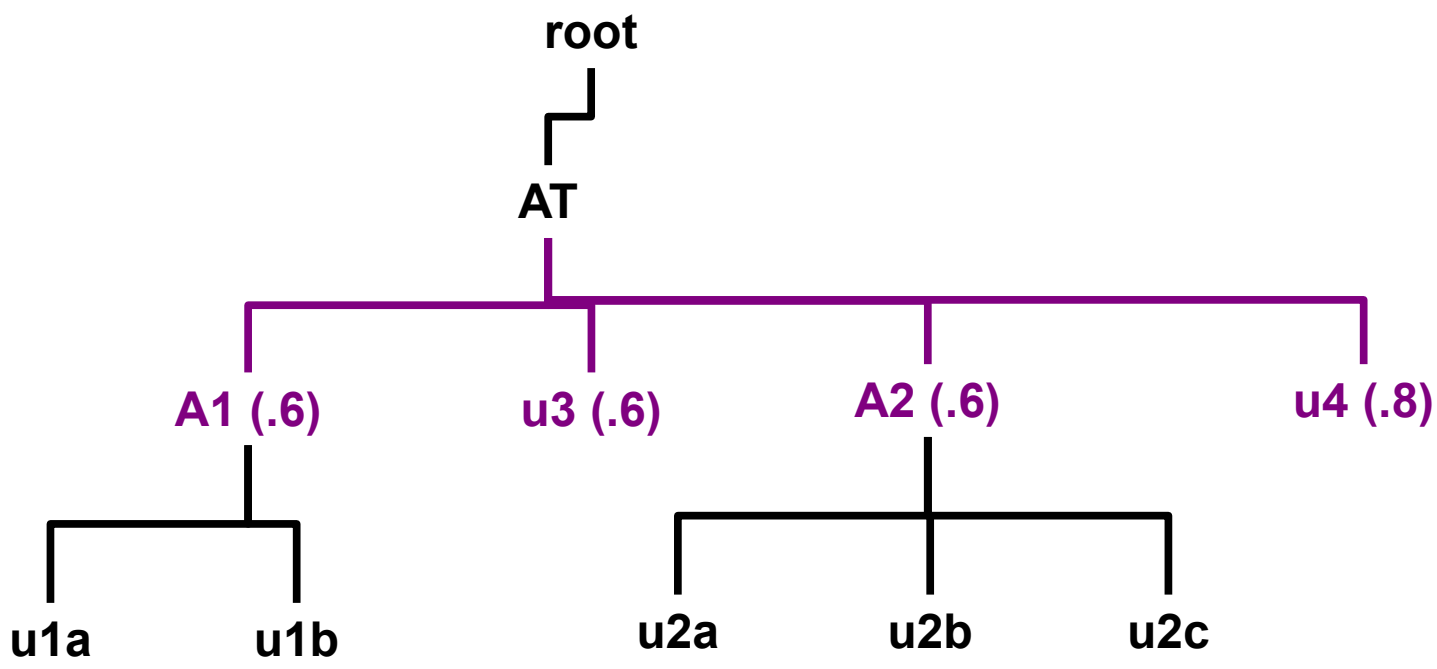
rank = 64

Fair Tree: Tie Handling



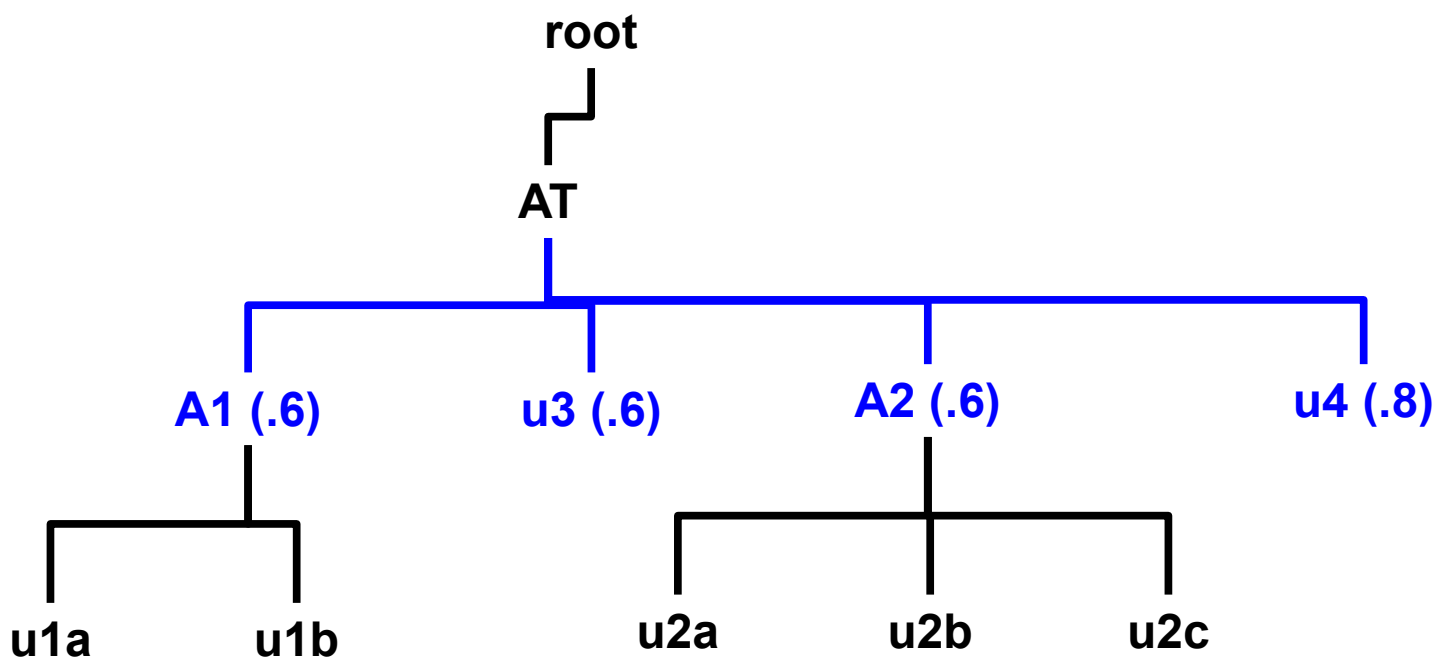
rank = 64

Fair Tree: Tie Handling



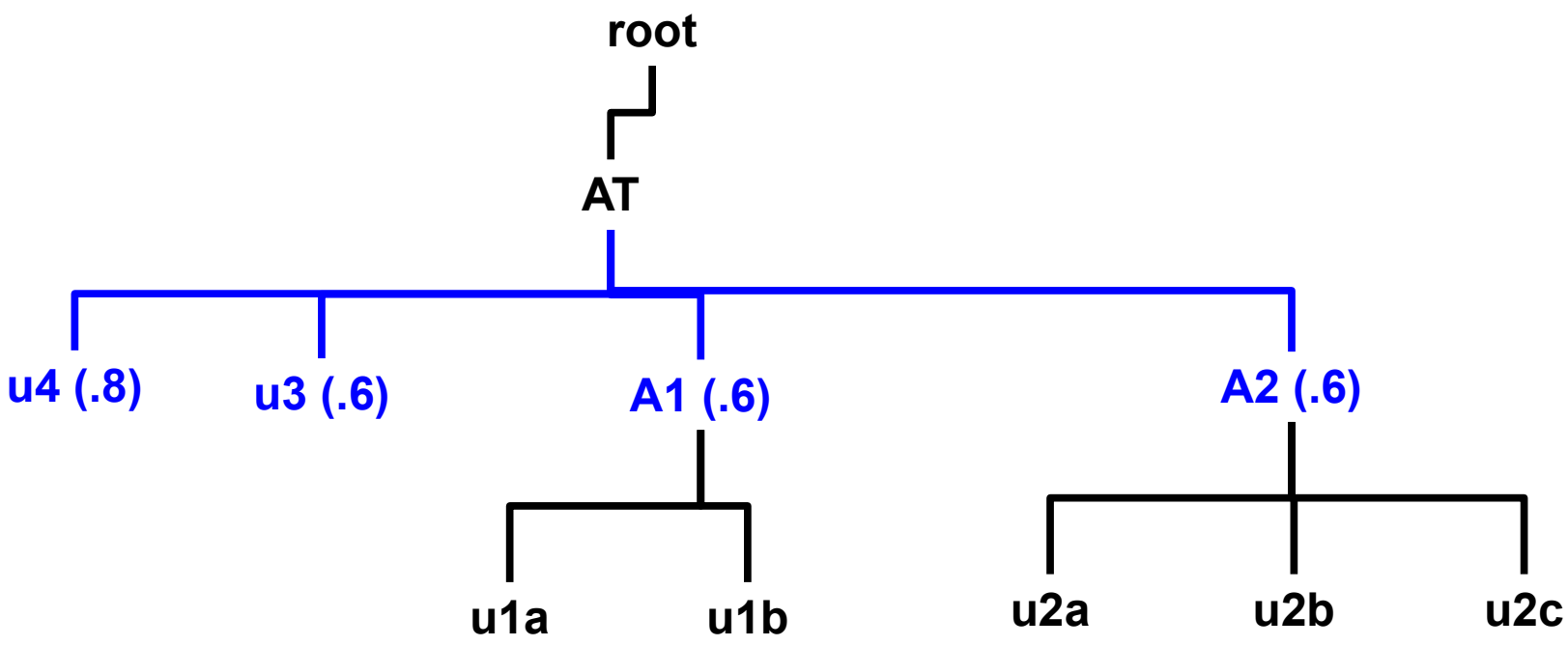
rank = 64

Fair Tree: Tie Handling



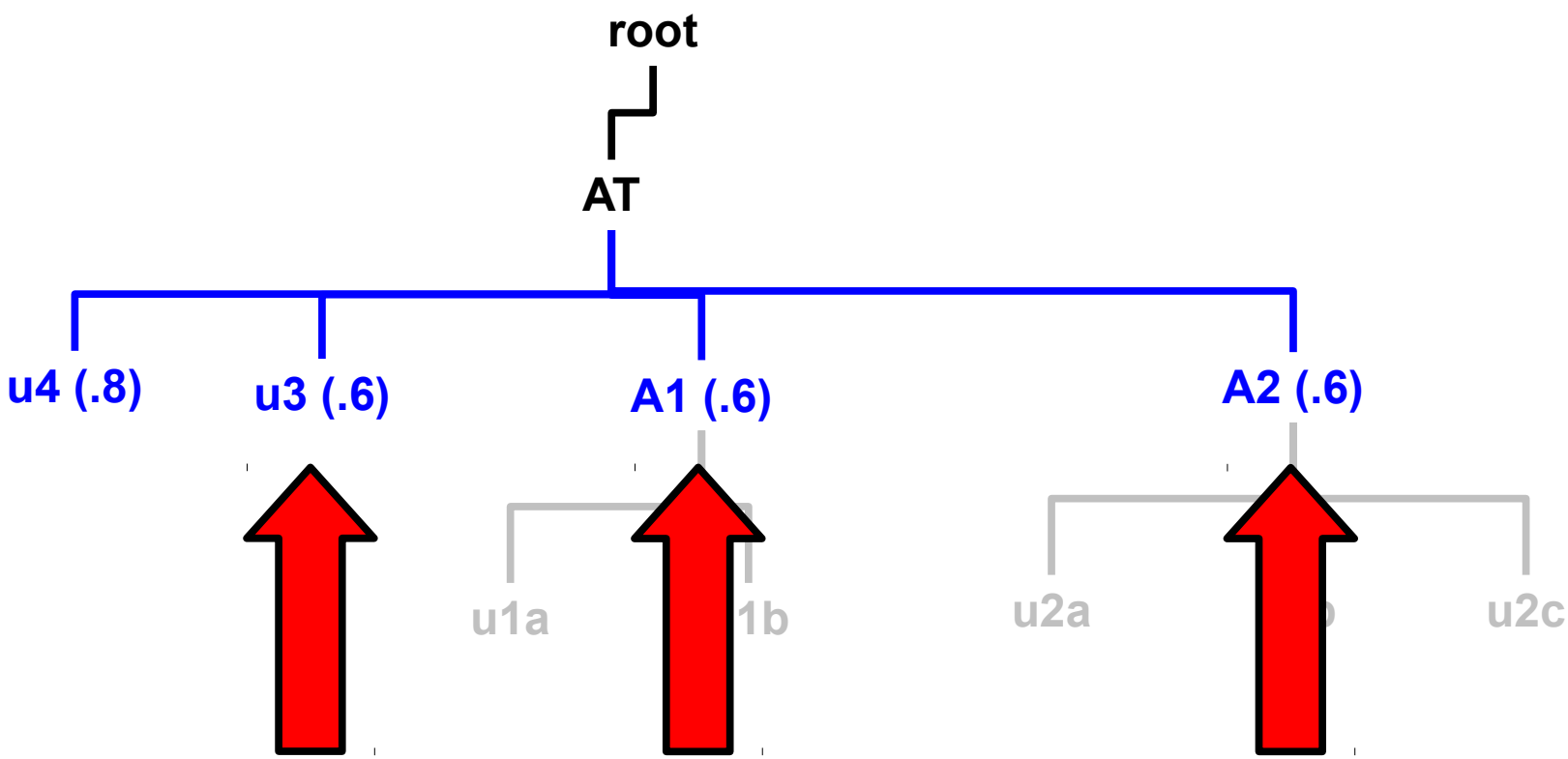
rank = 64

Fair Tree: Tie Handling



rank = 64

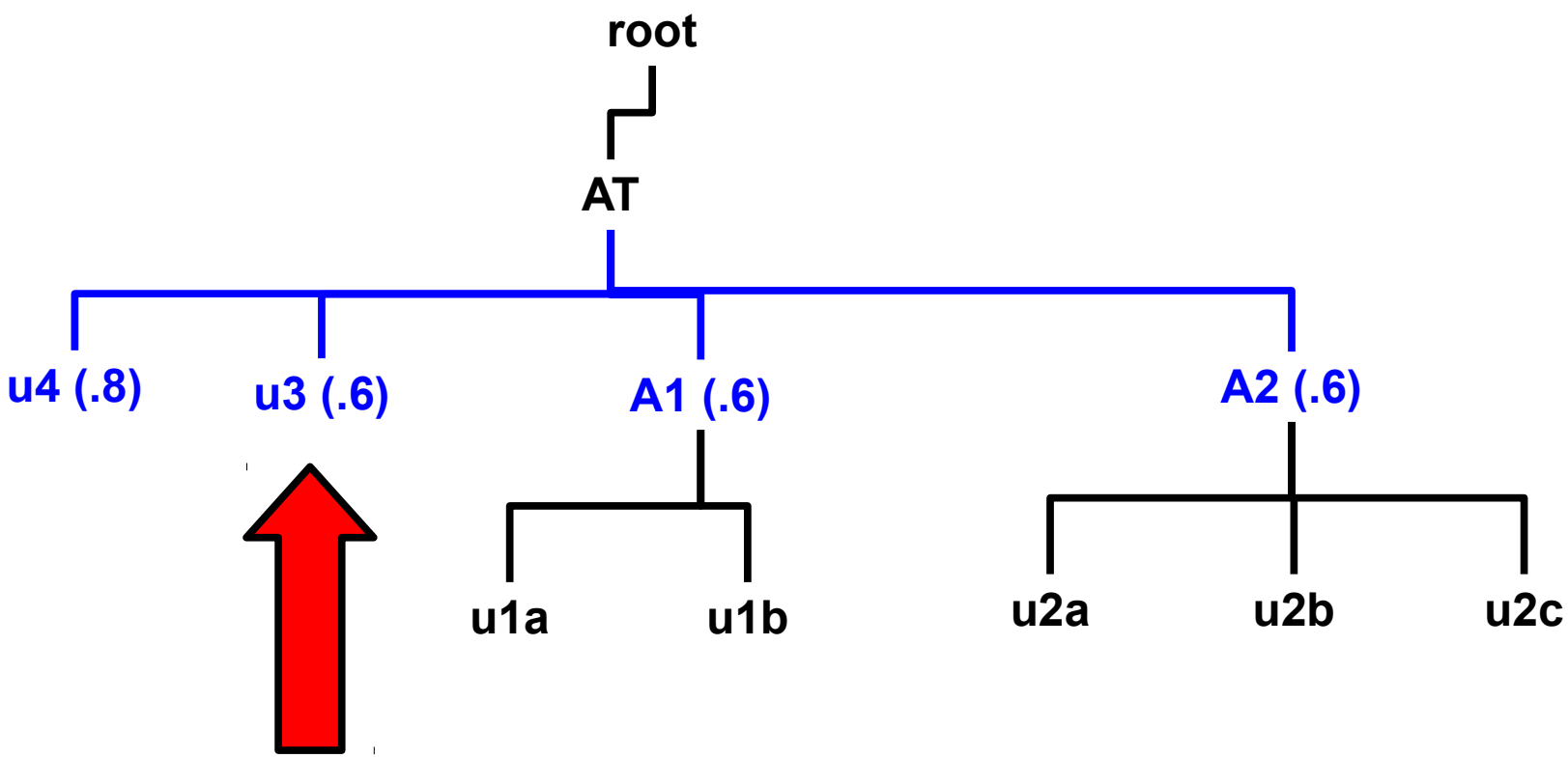
Fair Tree: Tie Handling



u3, A1, and A2 are tied at .6

rank = 64

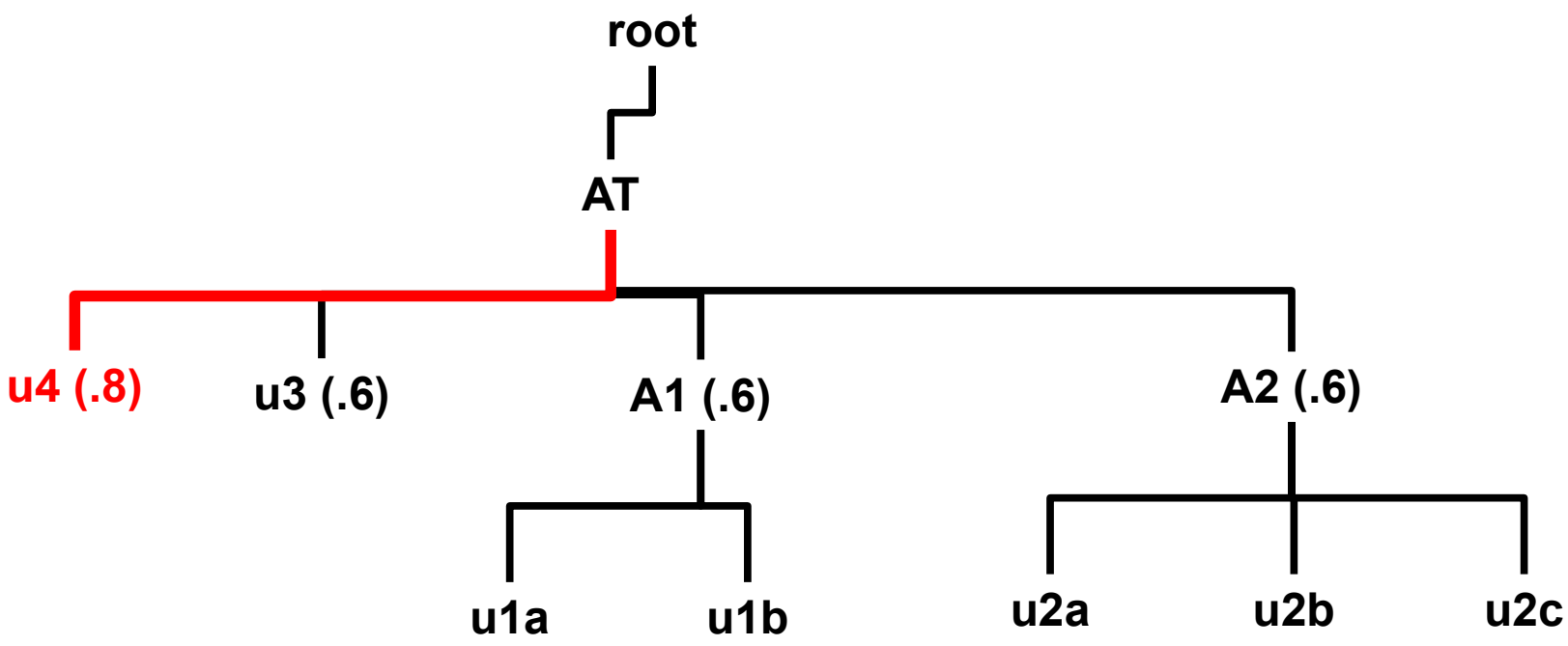
Fair Tree: Tie Handling



Tied users are placed to the left

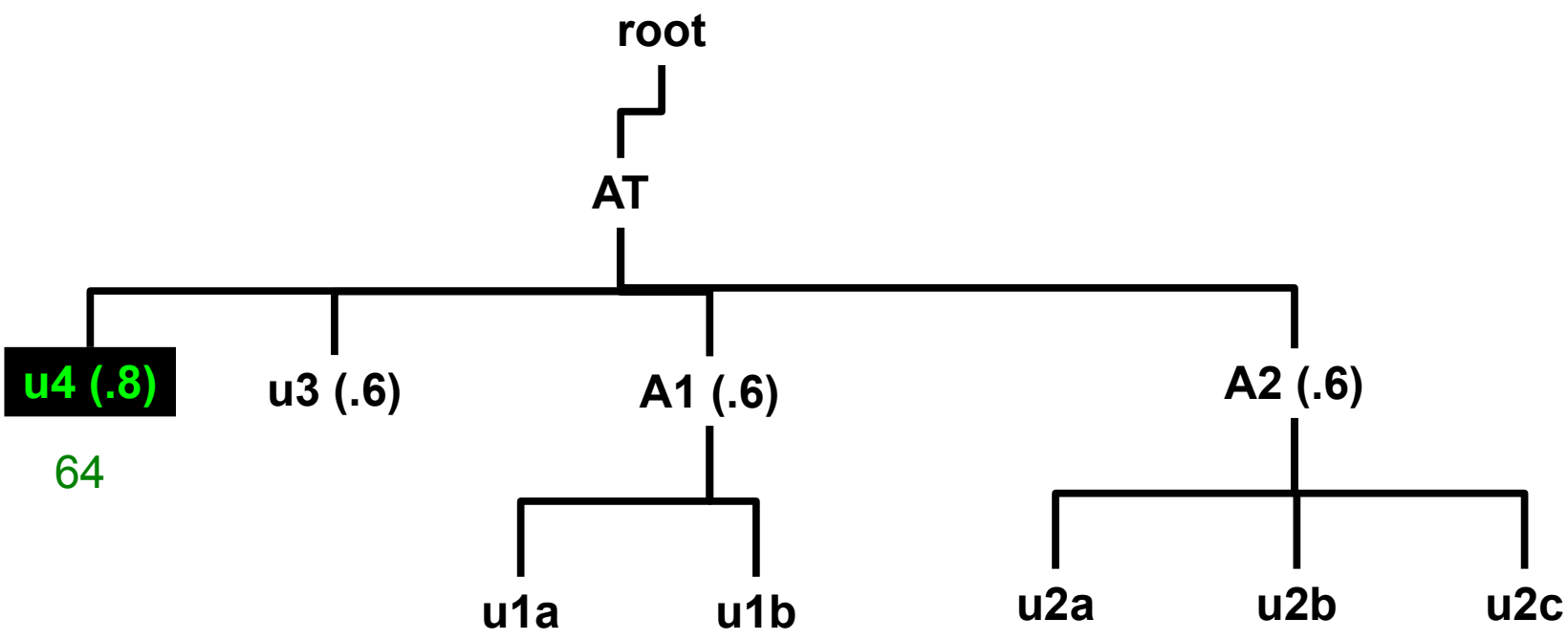
rank = 64

Fair Tree: Tie Handling



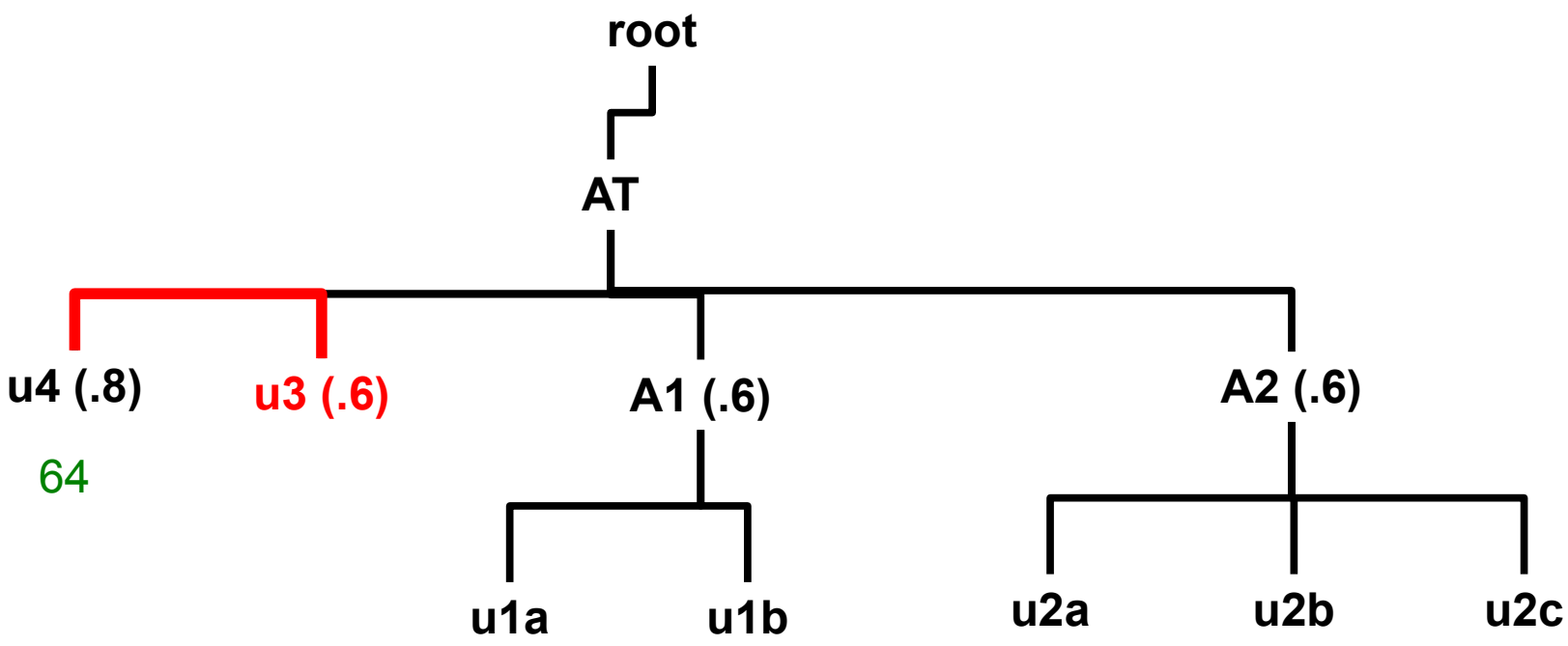
rank = 64

Fair Tree: Tie Handling



rank = 64

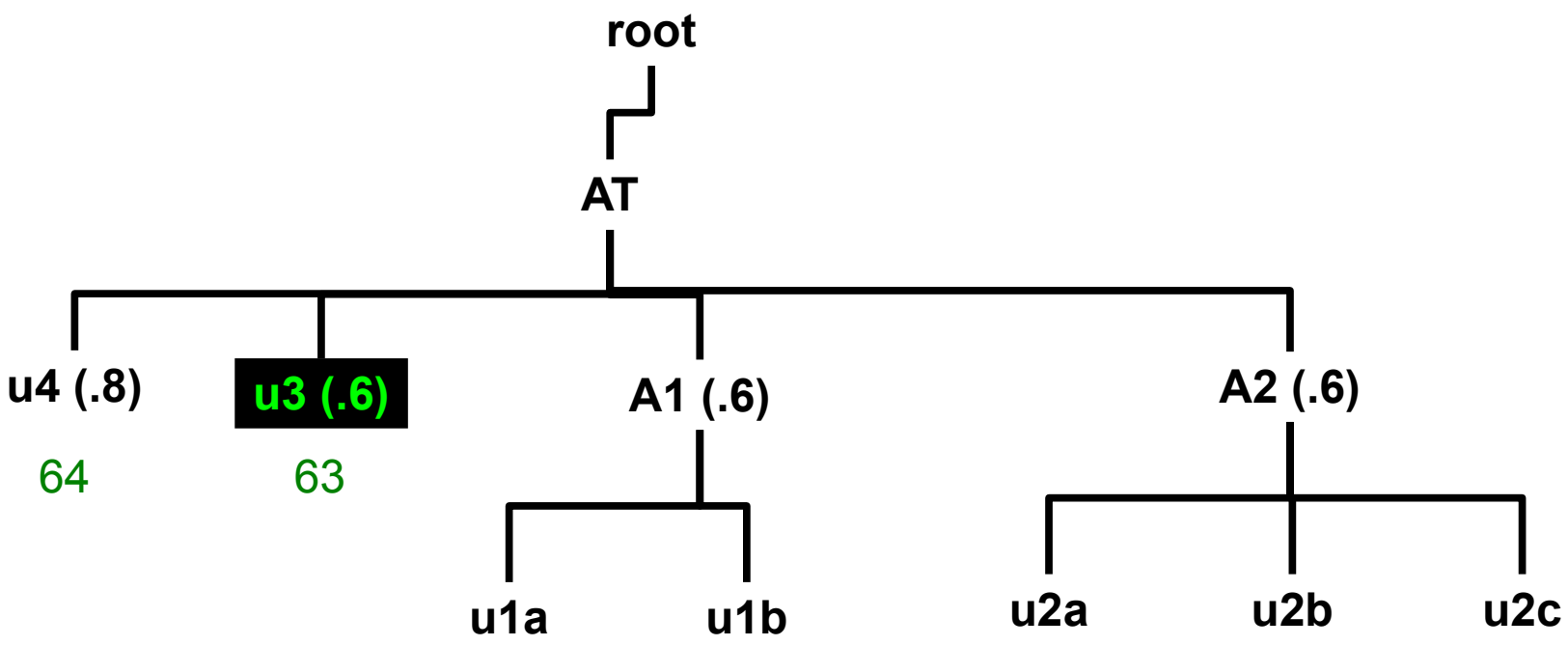
Fair Tree: Tie Handling



Next association **not** tied. Decrement rank.

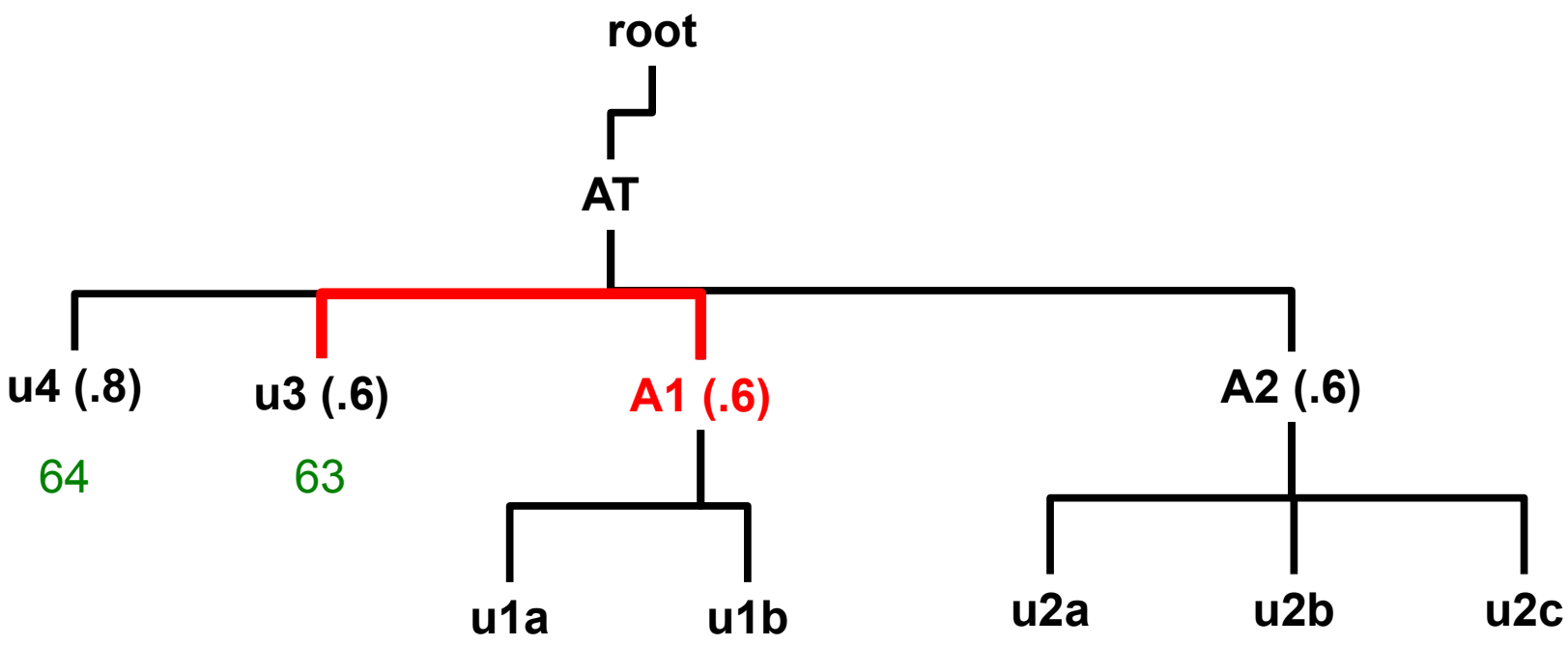
rank = 63

Fair Tree: Tie Handling



rank = 63

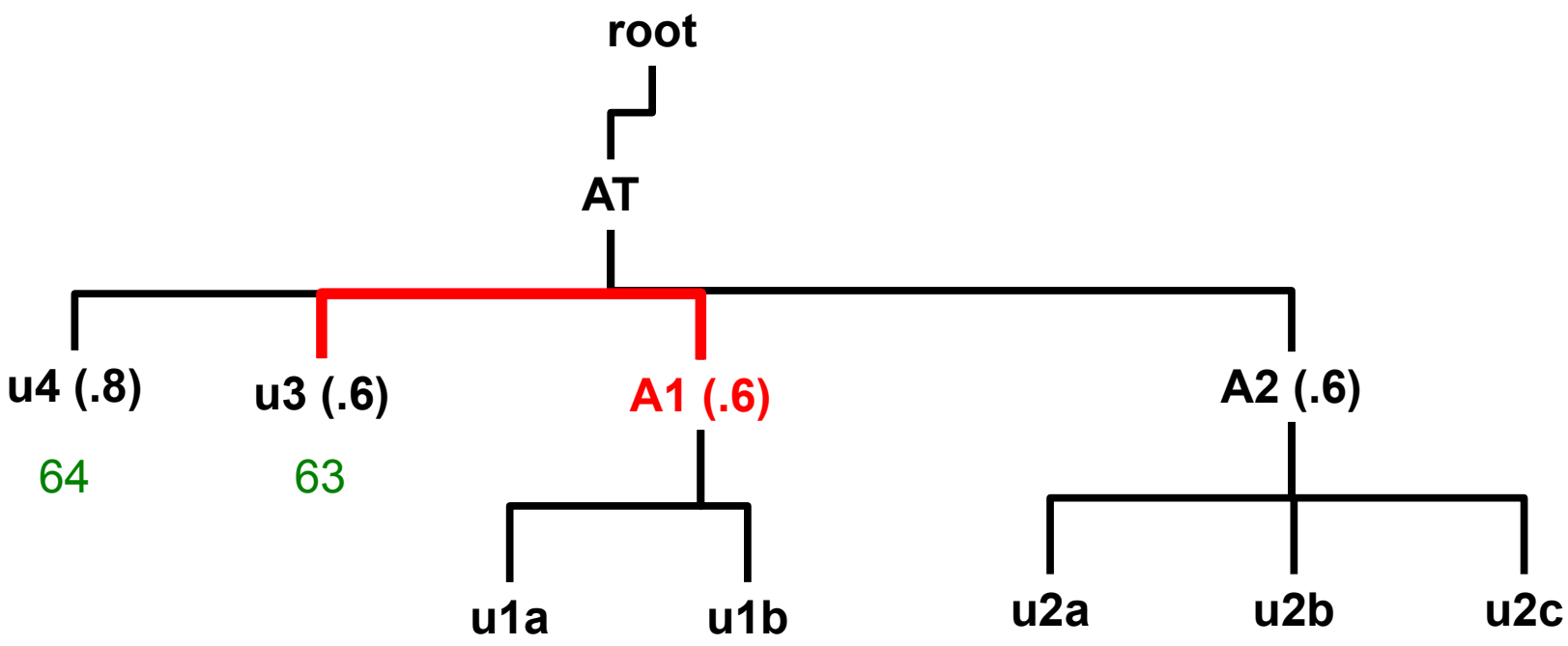
Fair Tree: Tie Handling



Next association **is** tied. rank stays the same

rank = 63

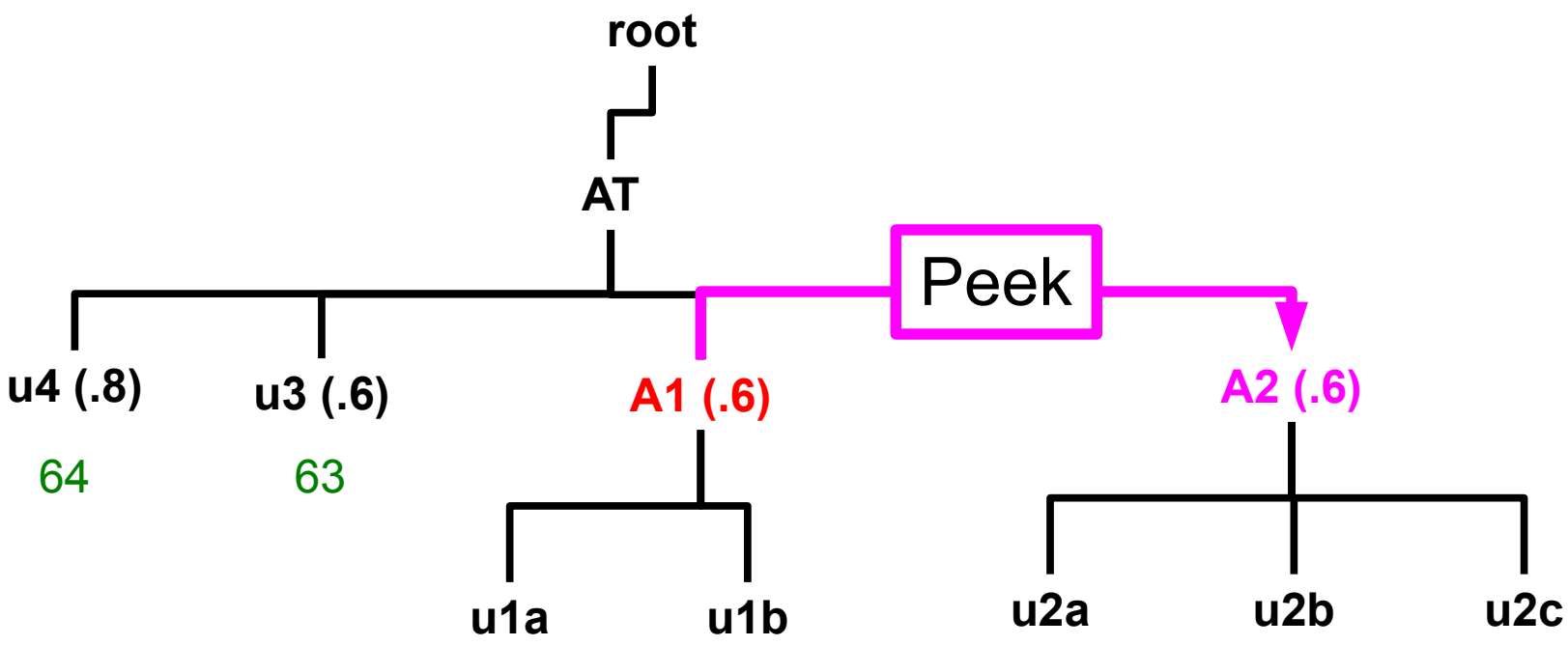
Fair Tree: Tie Handling



Next association **is** tied. rank stays the same

rank = 63

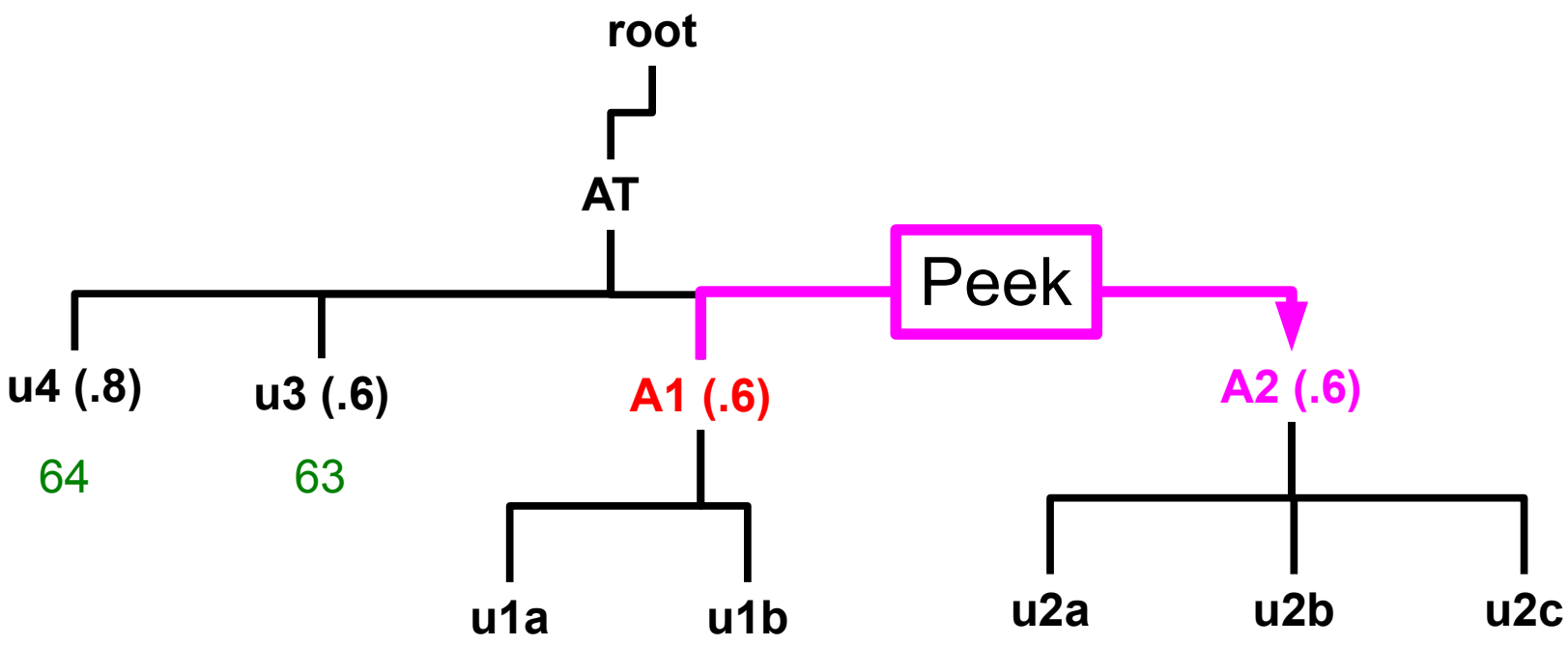
Fair Tree: Tie Handling



Peek at next account to check for a tie

rank = 63

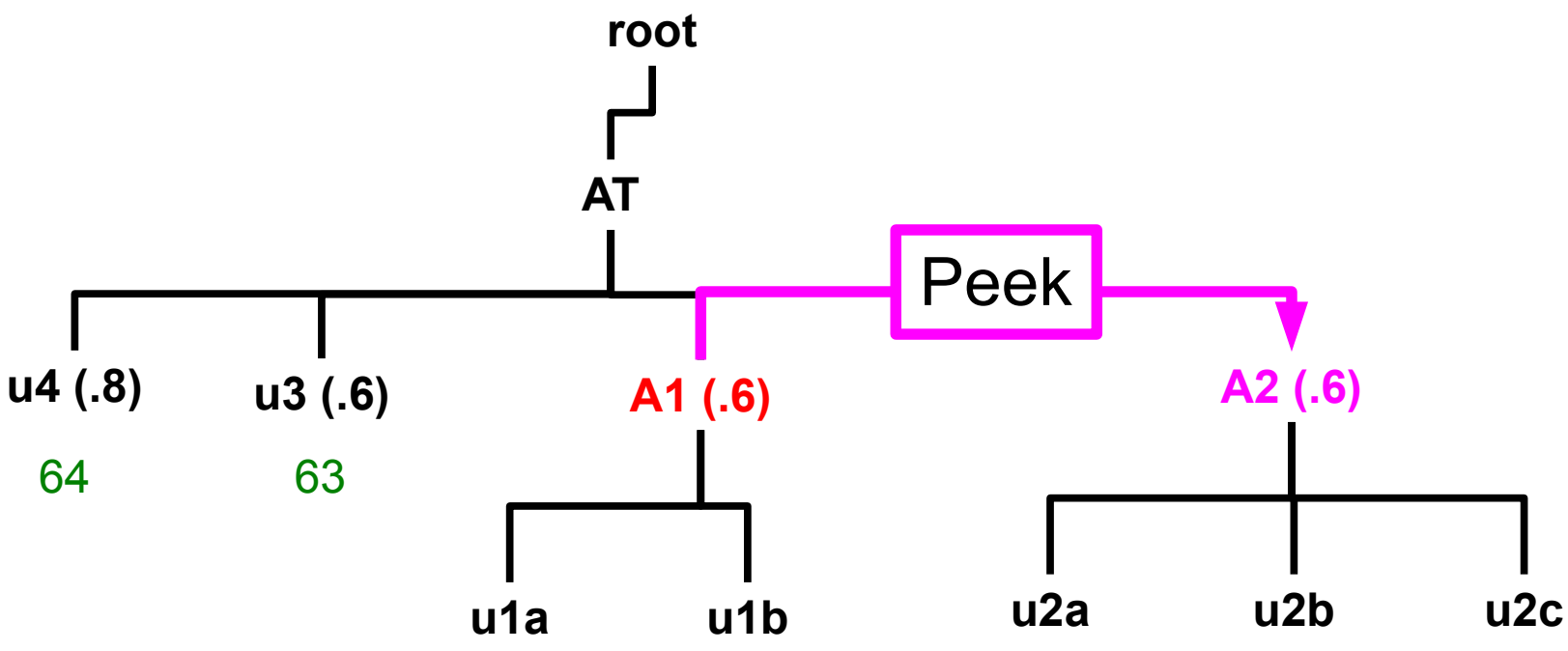
Fair Tree: Tie Handling



A1->level_fs == A2->level_fs

rank = 63

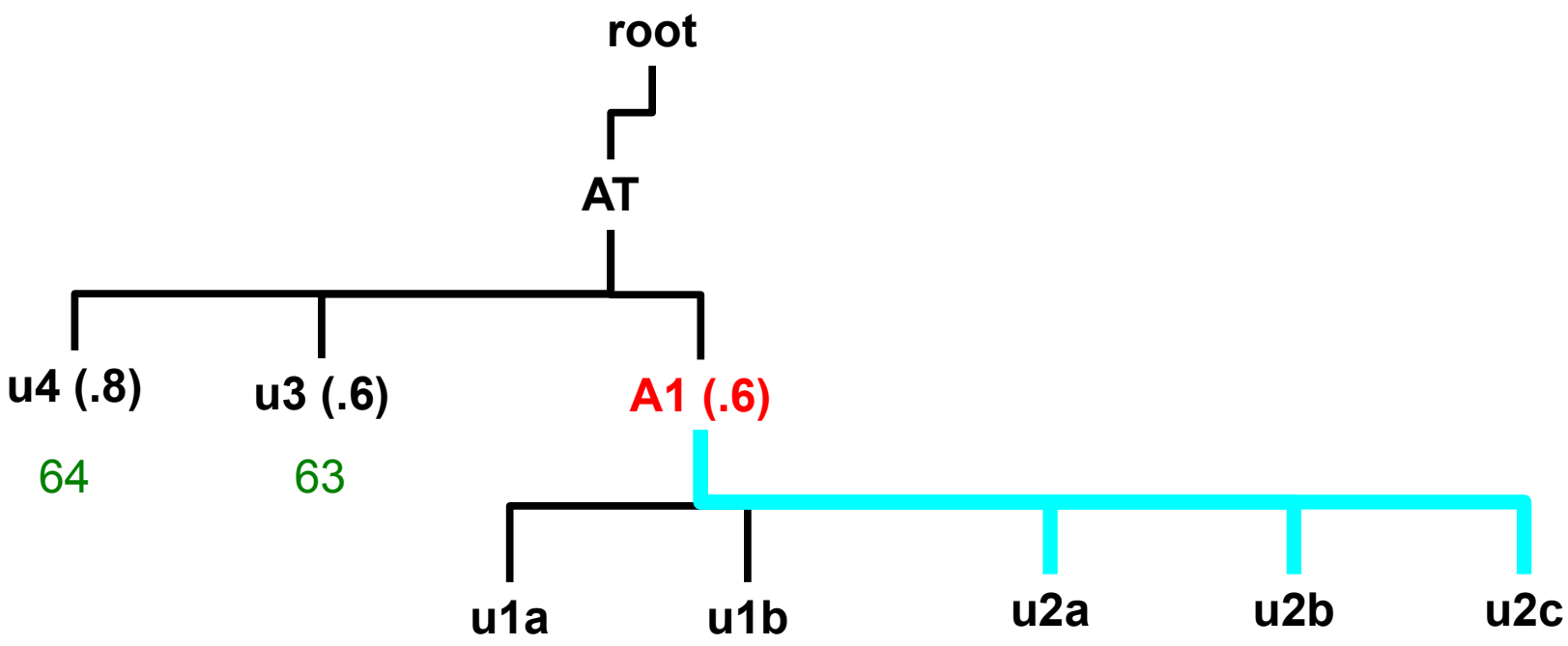
Fair Tree: Tie Handling



Accounts are equal. Merge!

rank = 63

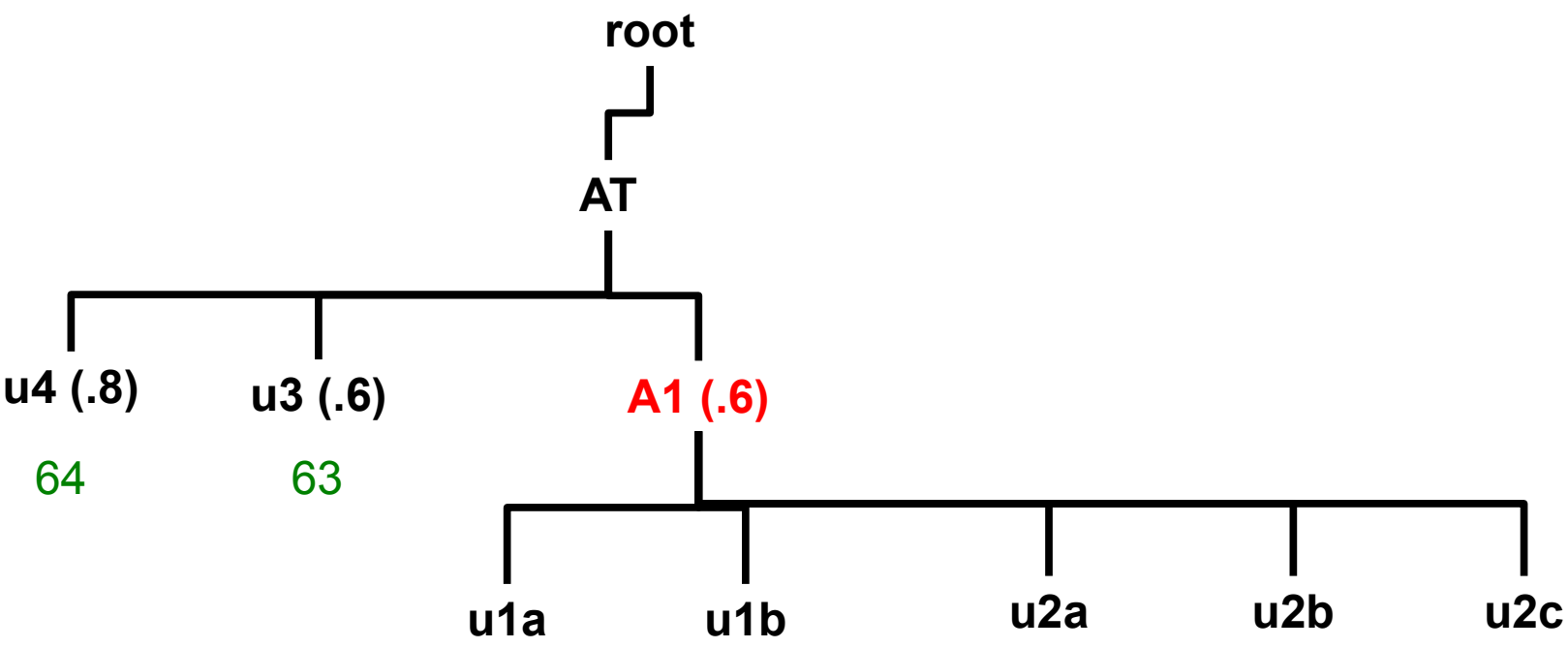
Fair Tree: Tie Handling



Merge complete

rank = 63

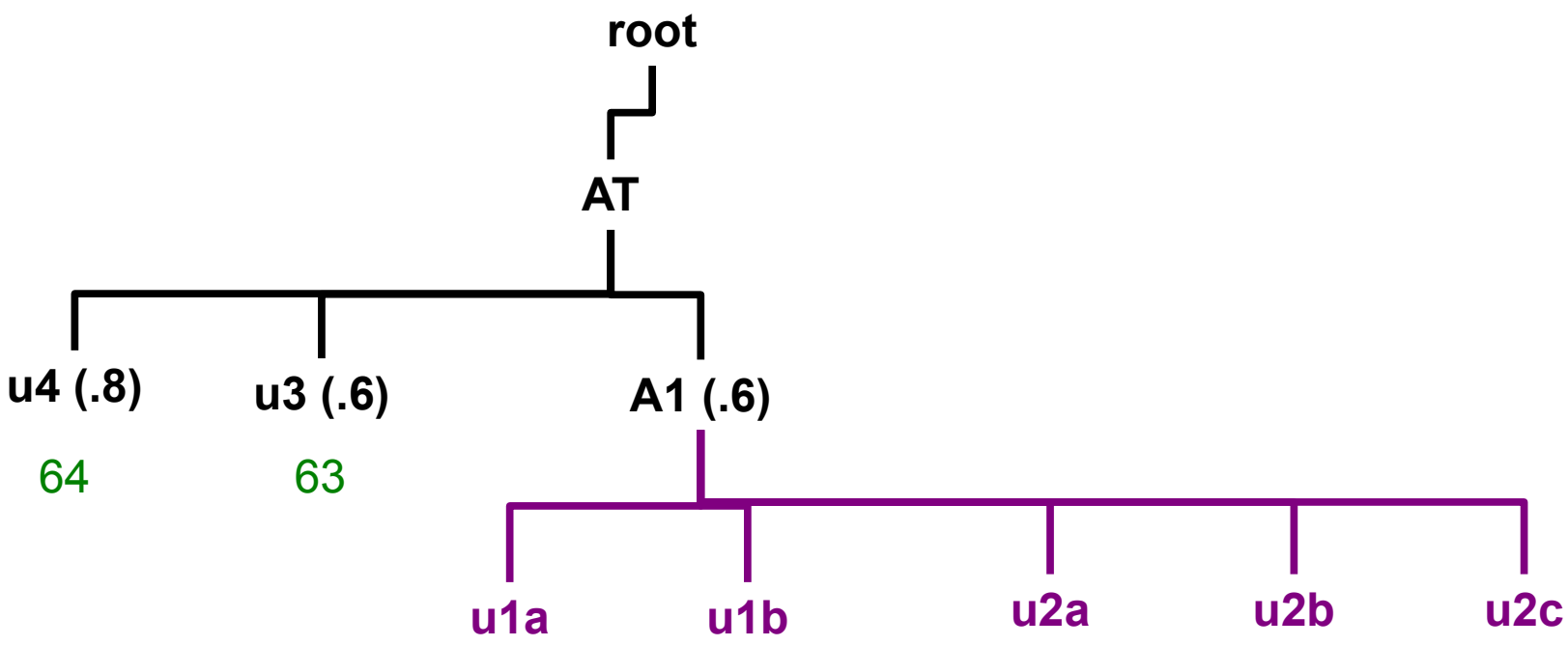
Fair Tree: Tie Handling



Merge complete

rank = 63

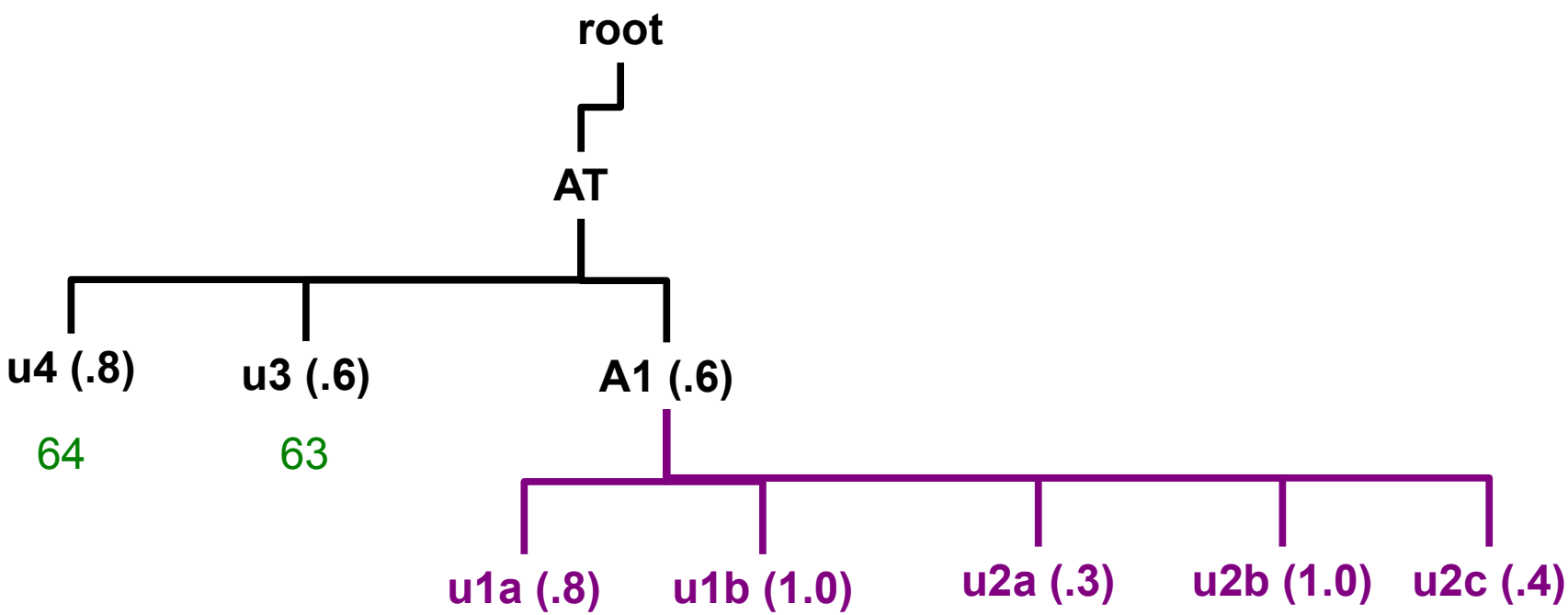
Fair Tree: Tie Handling



Calculate level_fs using **actual siblings**, ***not*** adopted ones.
The calculations pretend that the merge never happened.

rank = 63

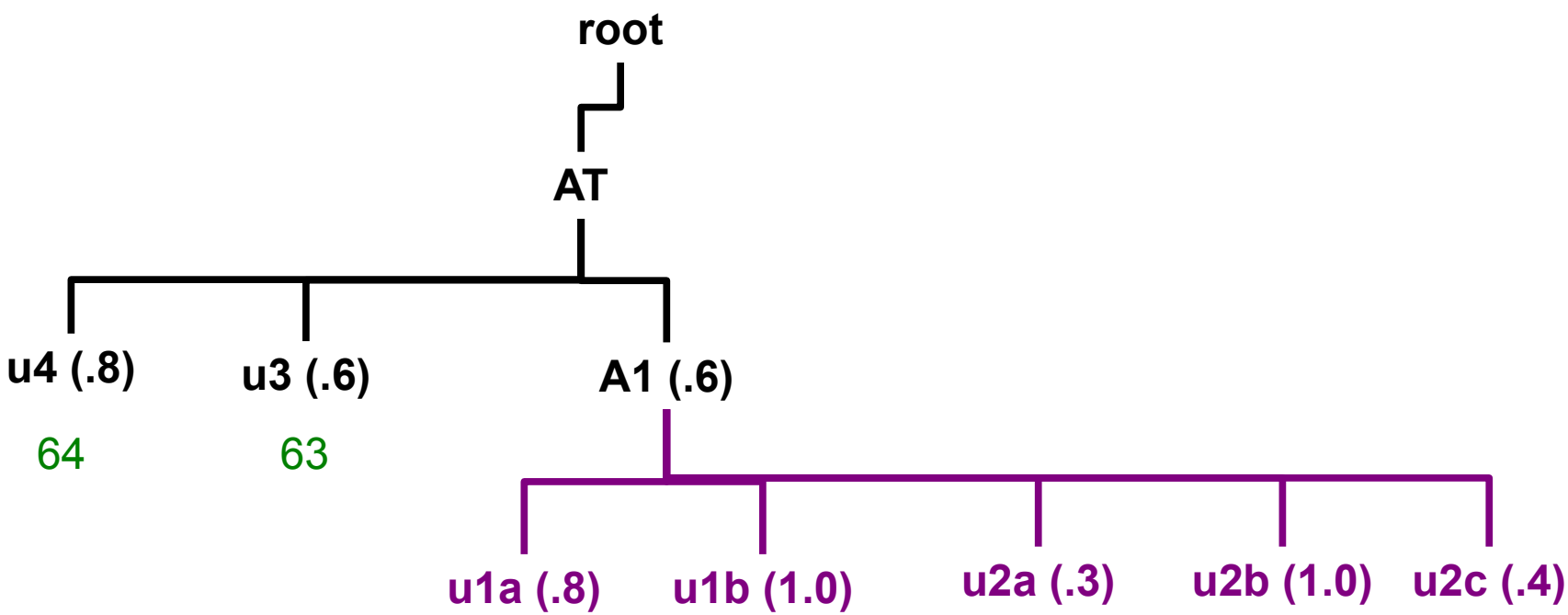
Fair Tree: Tie Handling



Calculate level_fs using **actual siblings**, ***not*** adopted ones.
The calculations pretend that the merge never happened.

rank = 63

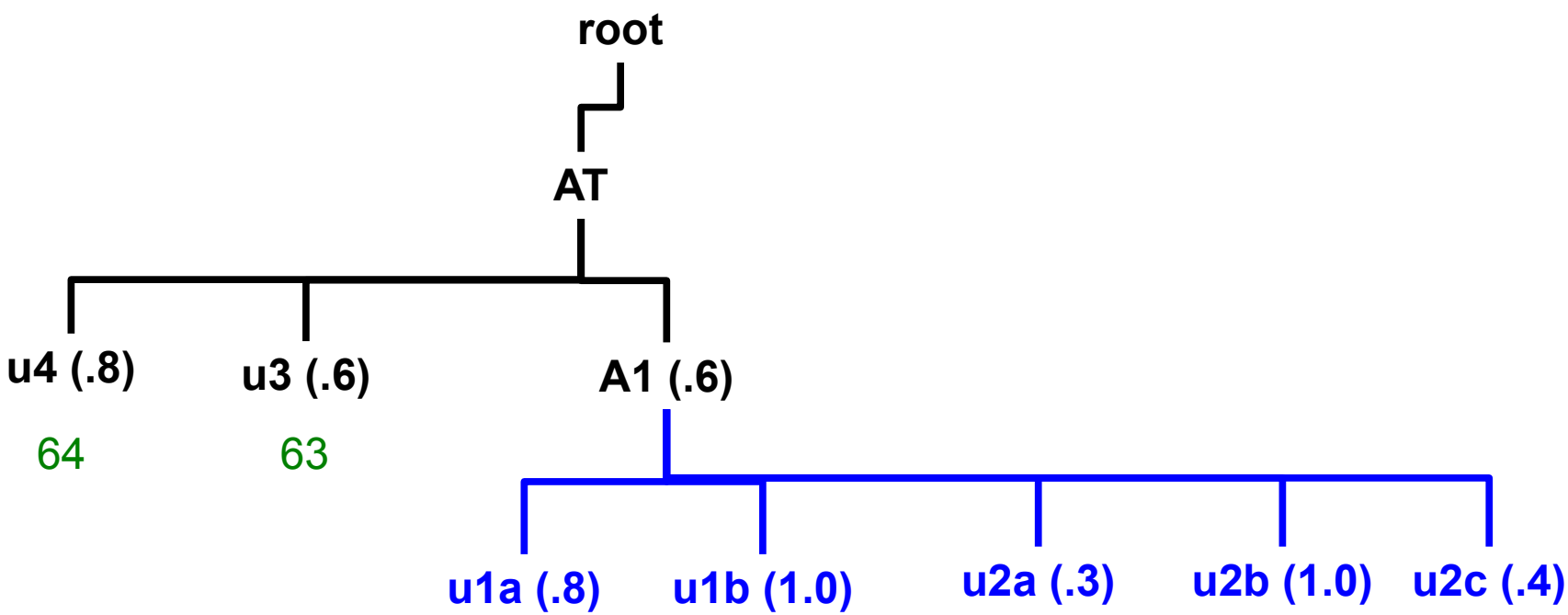
Fair Tree: Tie Handling



Calculate level_fs using **actual siblings**, ***not*** adopted ones.
The calculations pretend that the merge never happened.

rank = 63

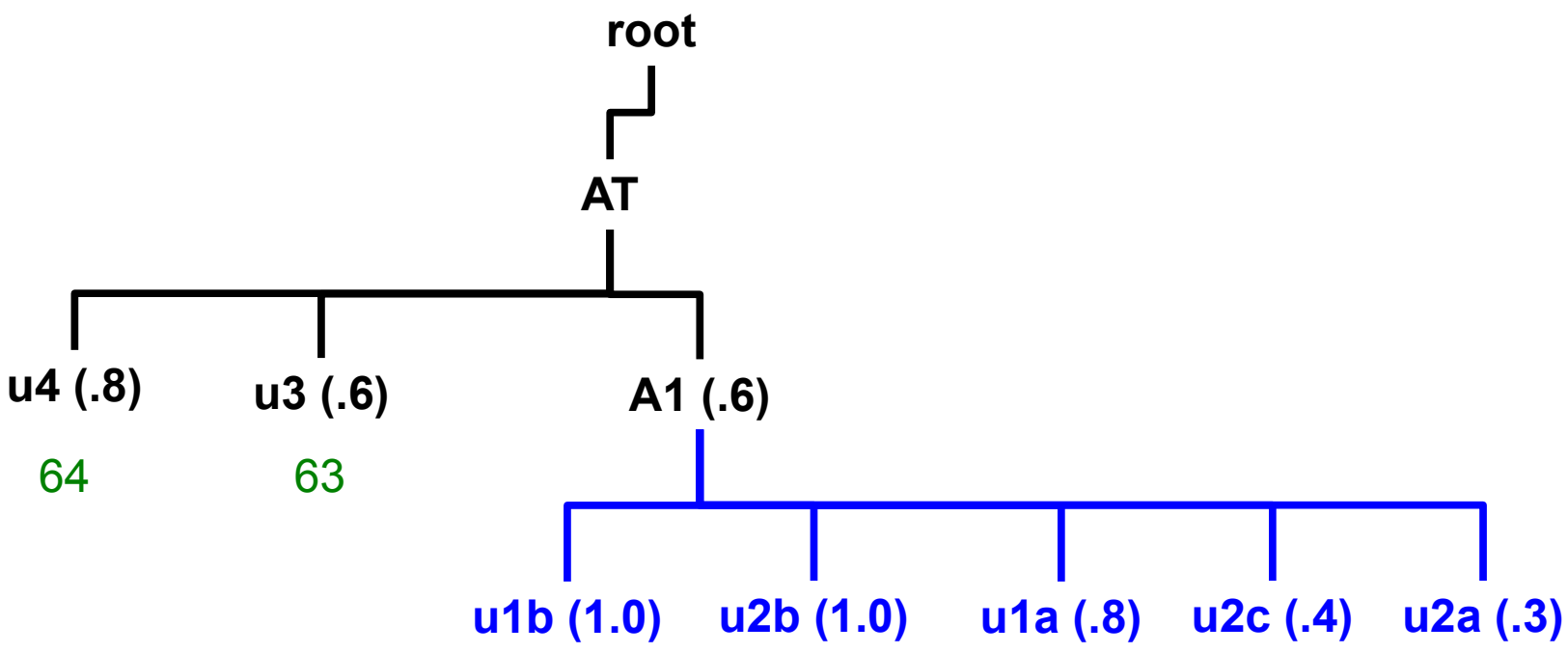
Fair Tree: Tie Handling



Sorting uses the merged list

rank = 63

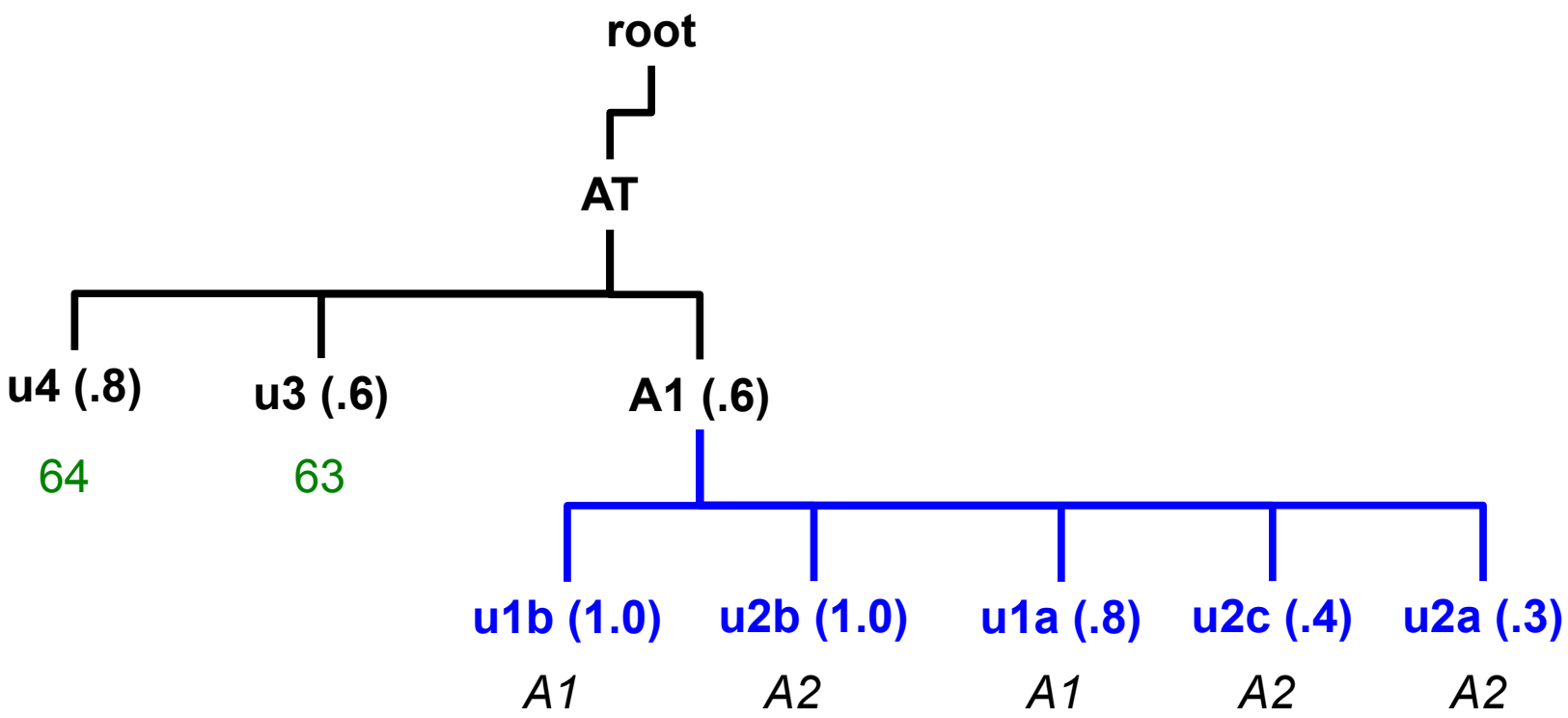
Fair Tree: Tie Handling



Sorting uses the merged list

rank = 63

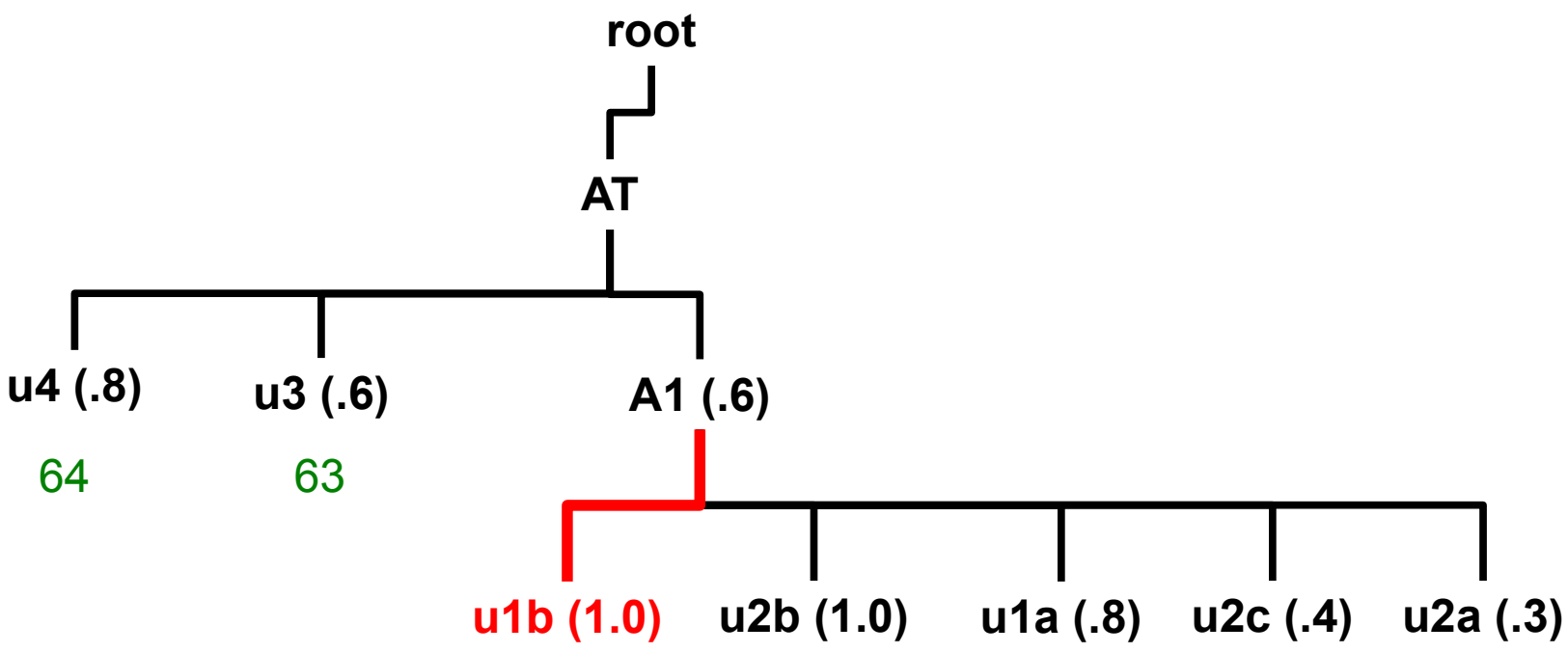
Fair Tree: Tie Handling



Note that the merged users are mixed

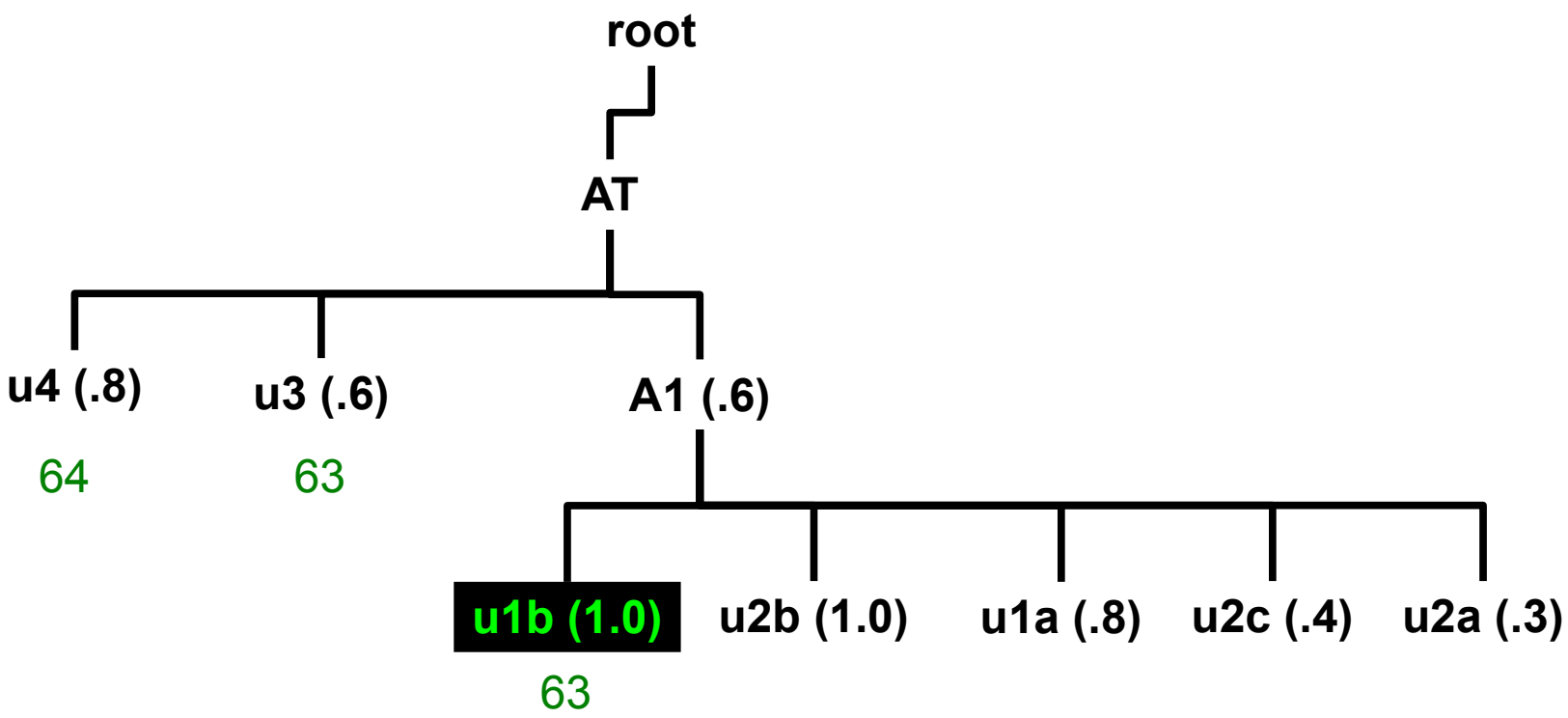
rank = 63

Fair Tree: Tie Handling



rank = 63

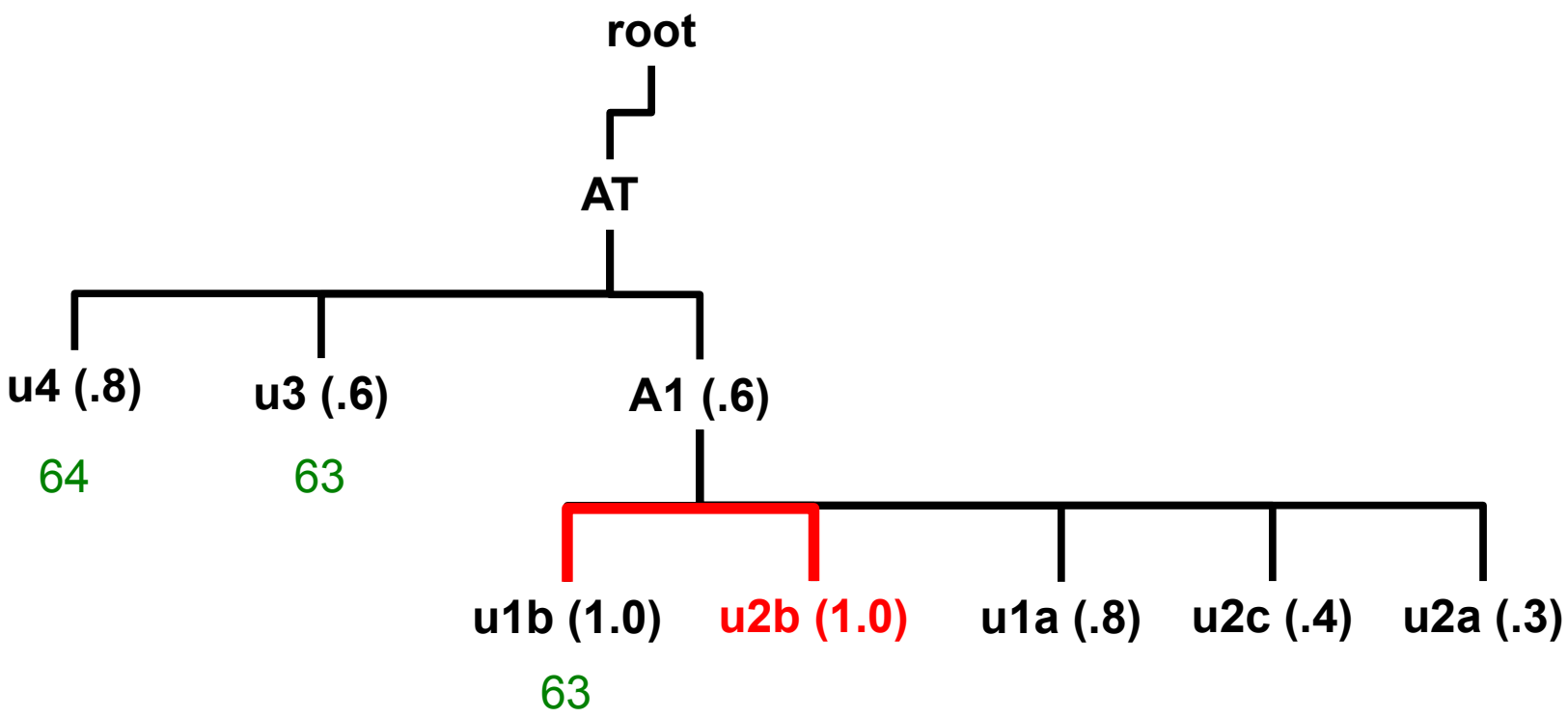
Fair Tree: Tie Handling



Note that rank has **not** been decremented.
u1b receives the same rank as u3 since A1 tied u3.

rank = 63

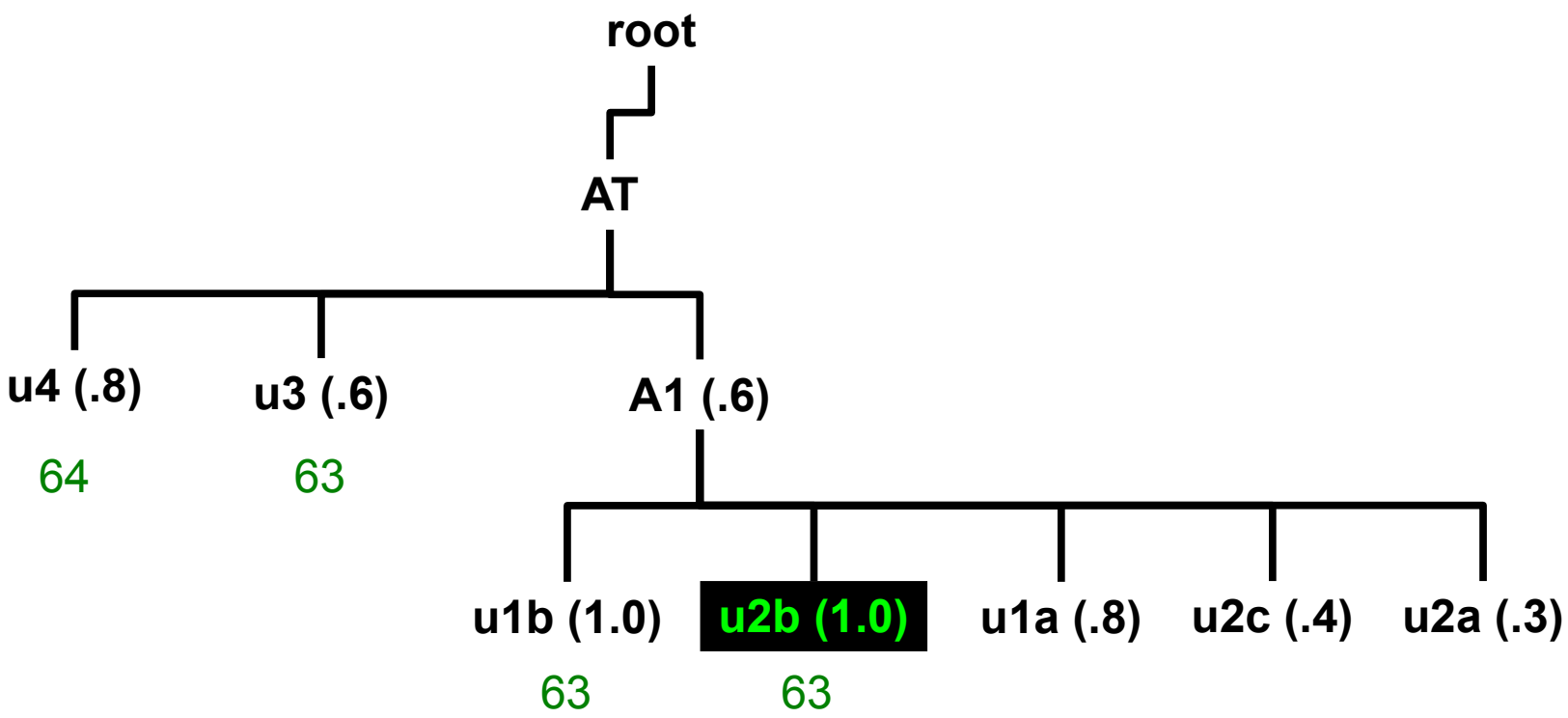
Fair Tree: Tie Handling



Next association **is** tied. Rank stays the same

rank = 63

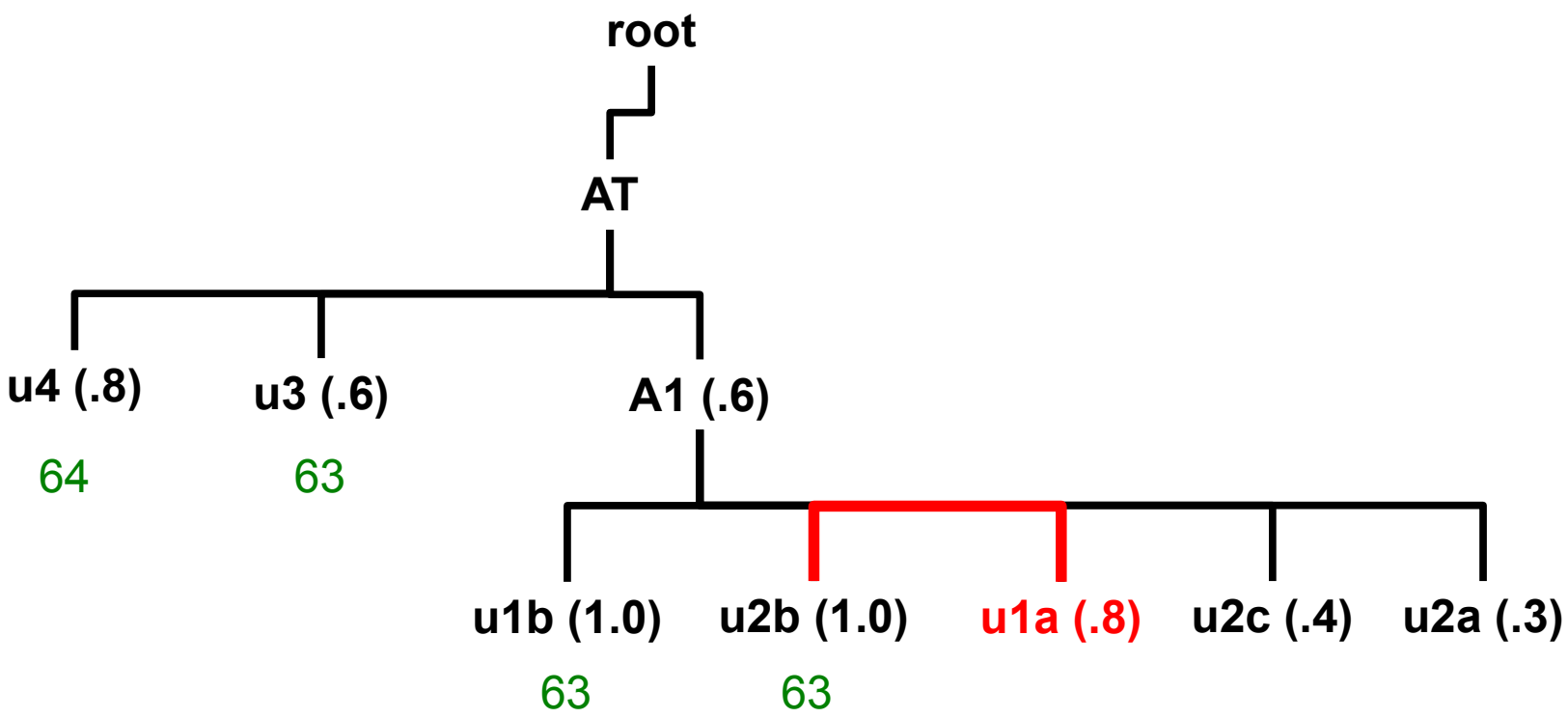
Fair Tree: Tie Handling



u3 == u1b == u2b

rank = 63

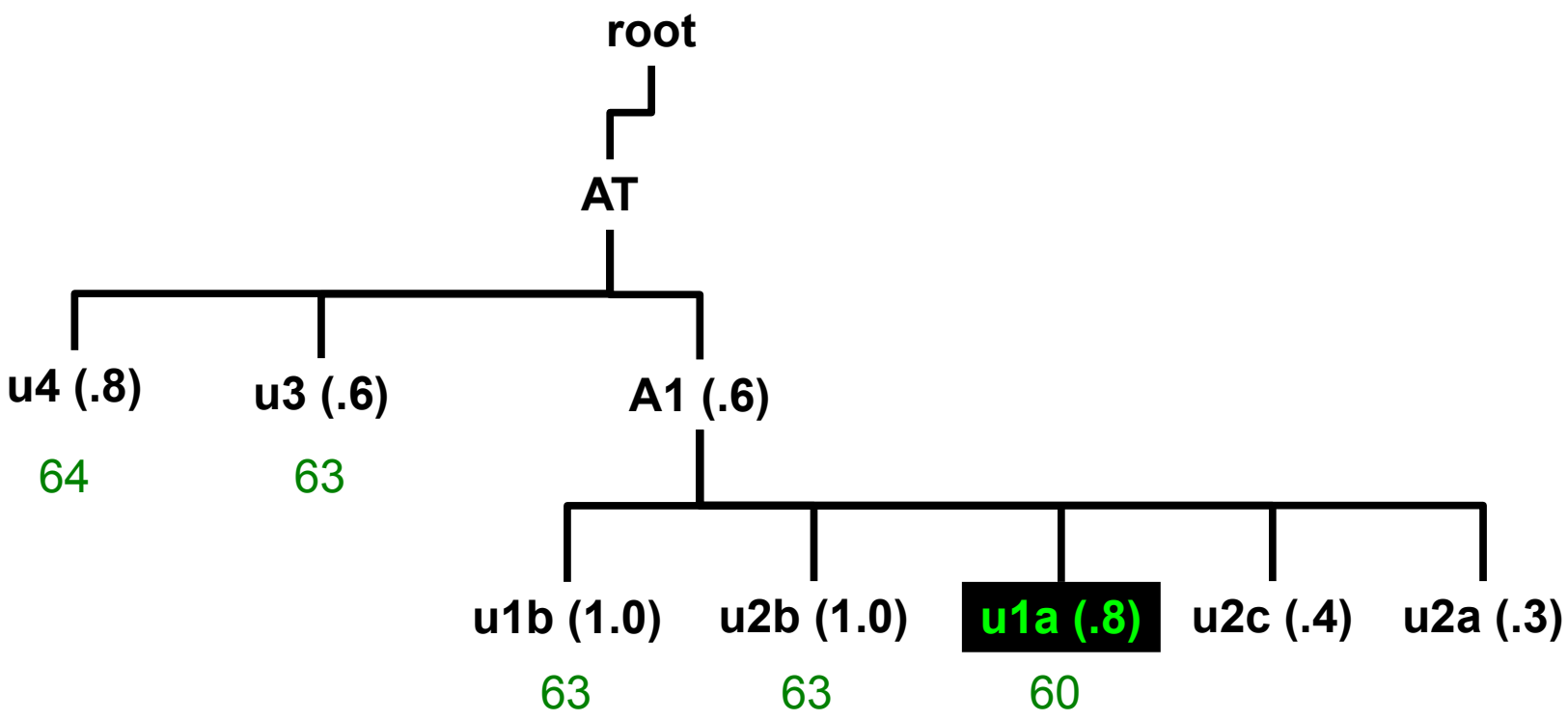
Fair Tree: Tie Handling



Next association **not** tied. Decrement rank by 3 (3 ties)

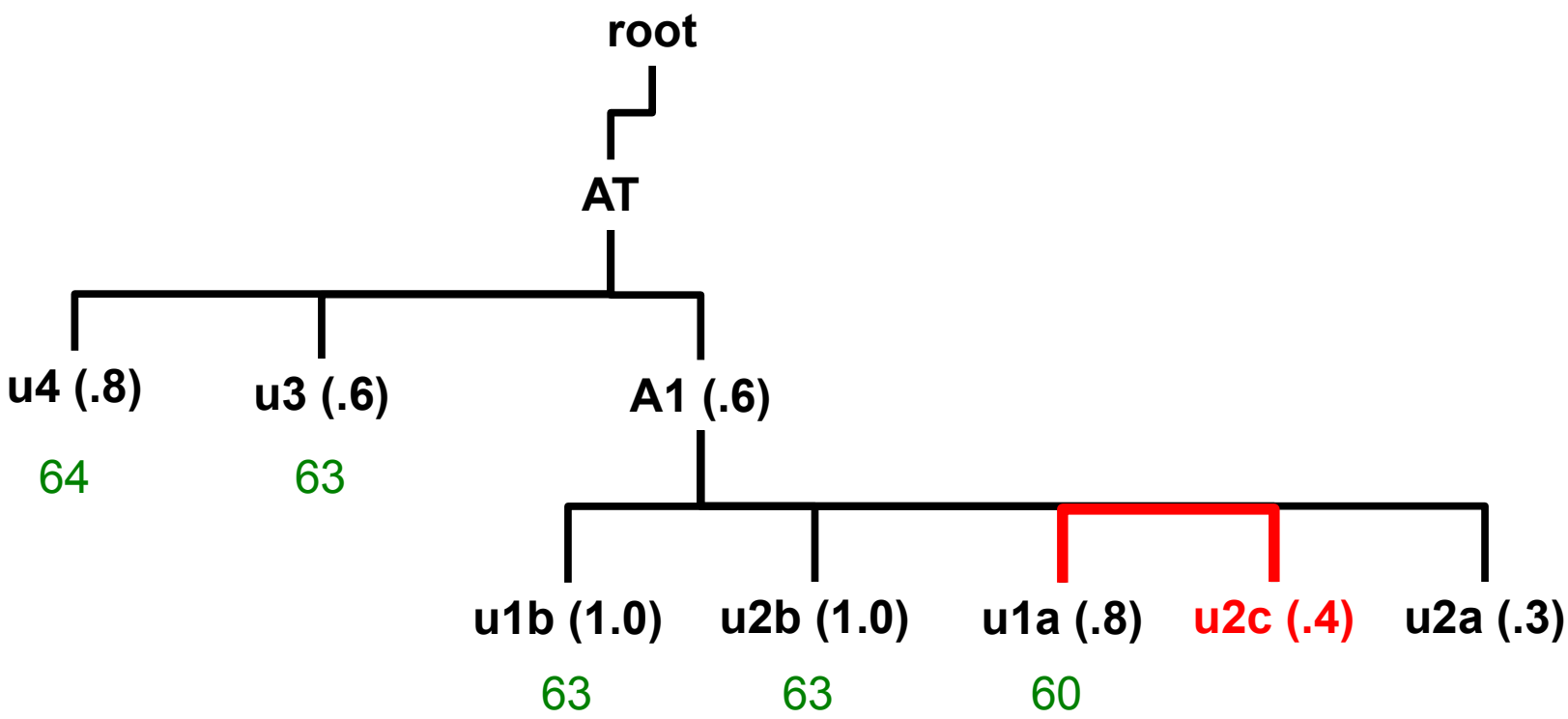
rank = 60

Fair Tree: Tie Handling



rank = 60

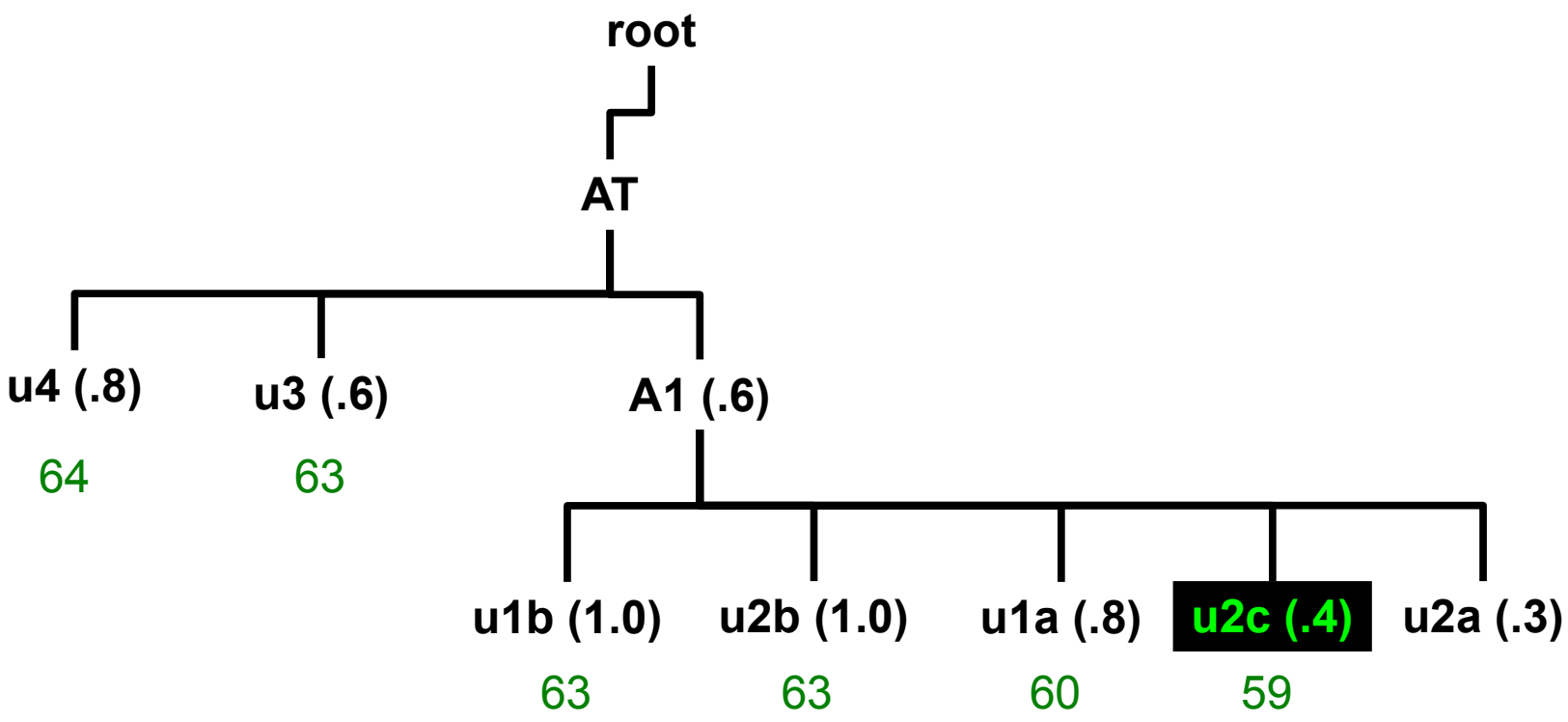
Fair Tree: Tie Handling



Next association **not** tied. Decrement rank

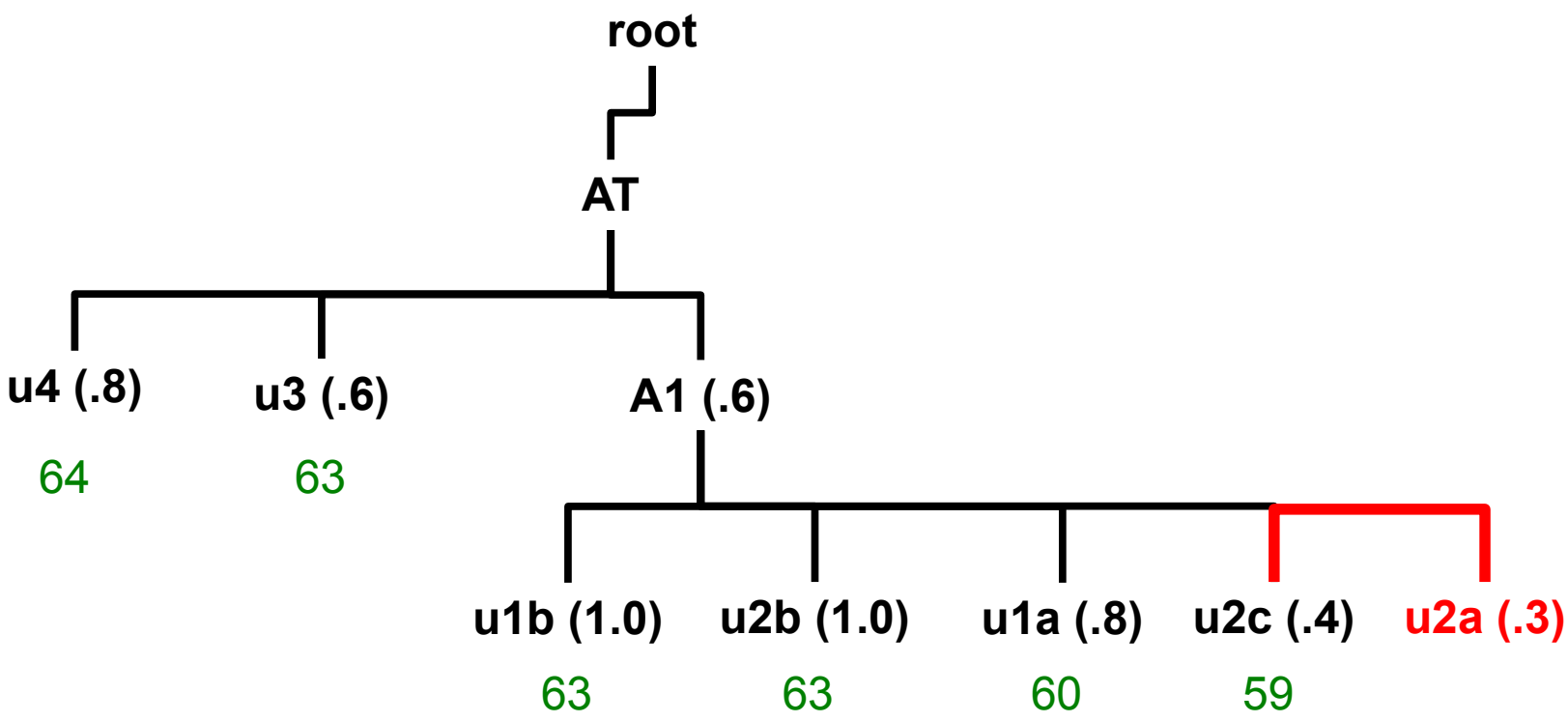
rank = 59

Fair Tree: Tie Handling



rank = 59

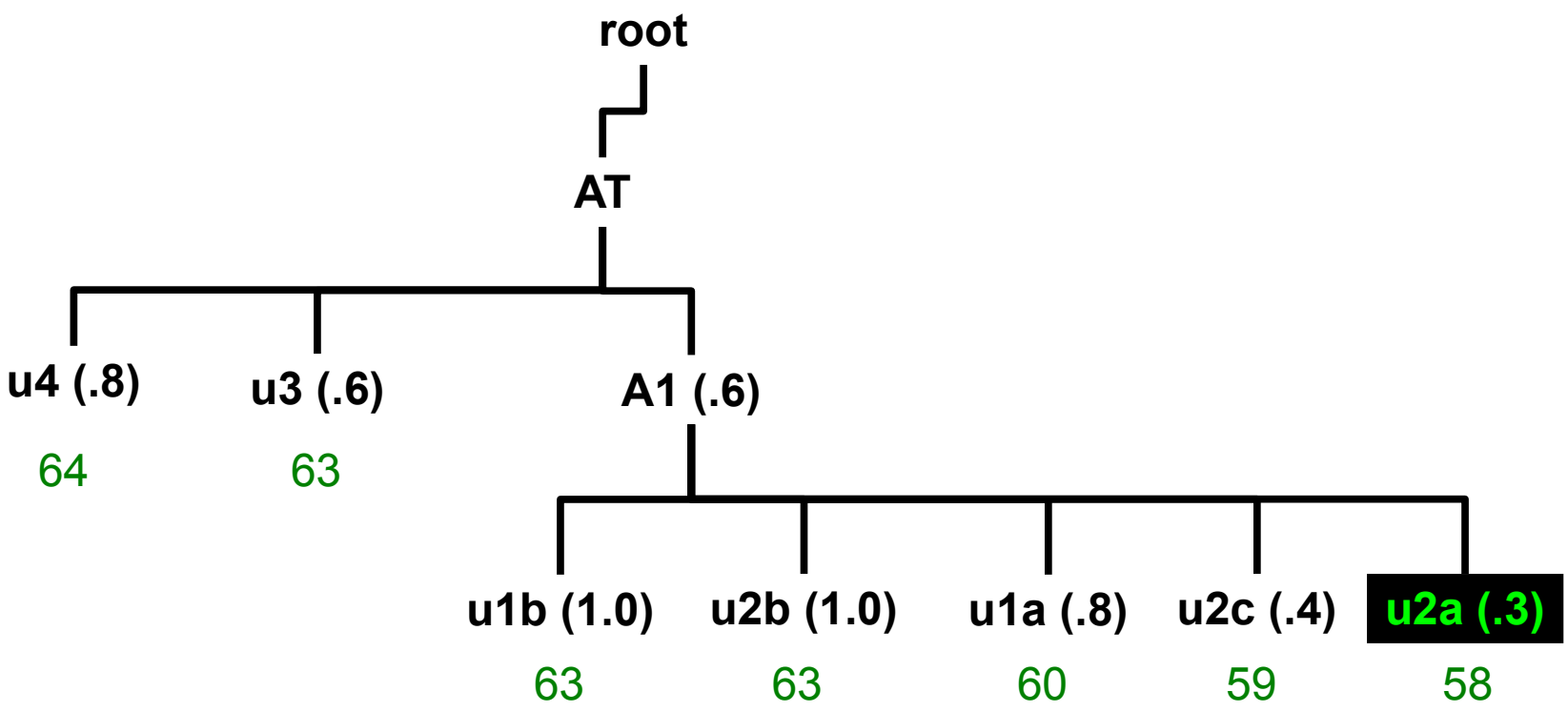
Fair Tree: Tie Handling



Next association **not** tied. Decrement rank

rank = 58

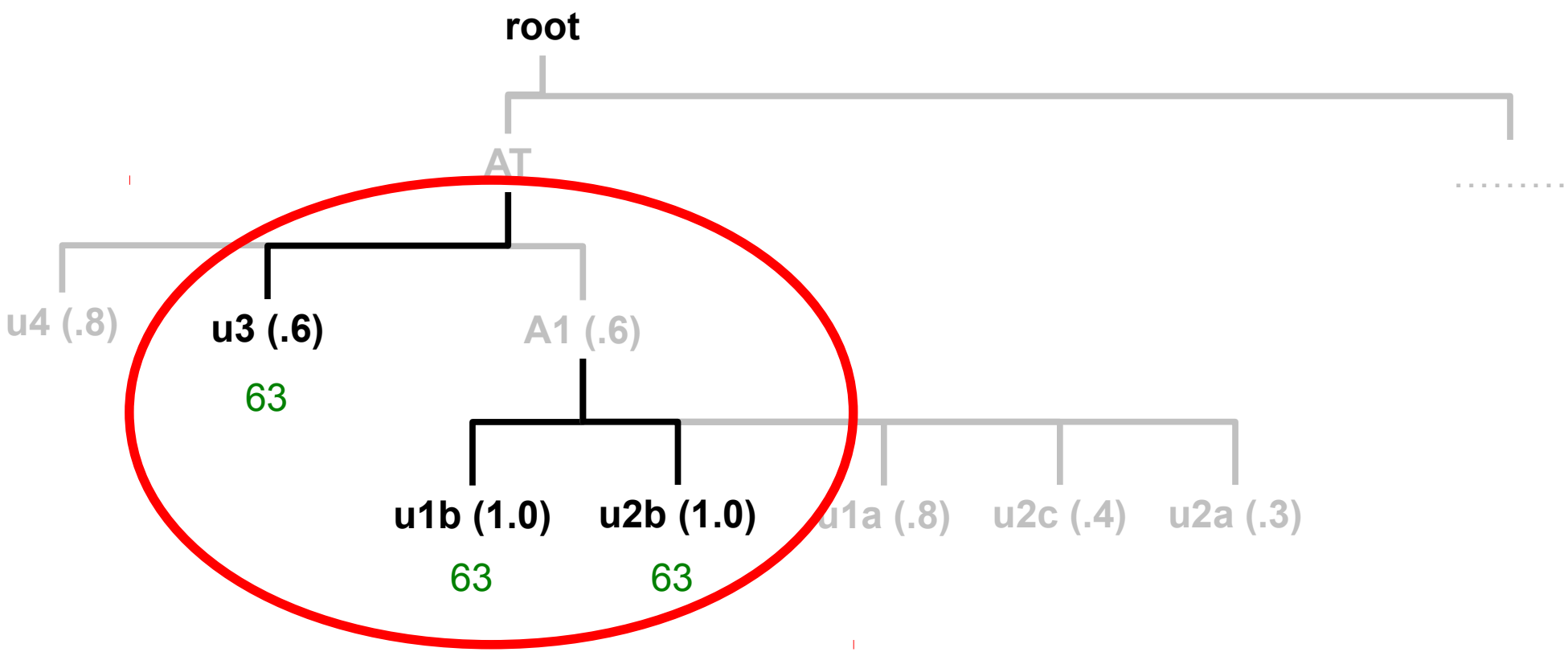
Fair Tree: Tie Handling



Sorting uses the merged list

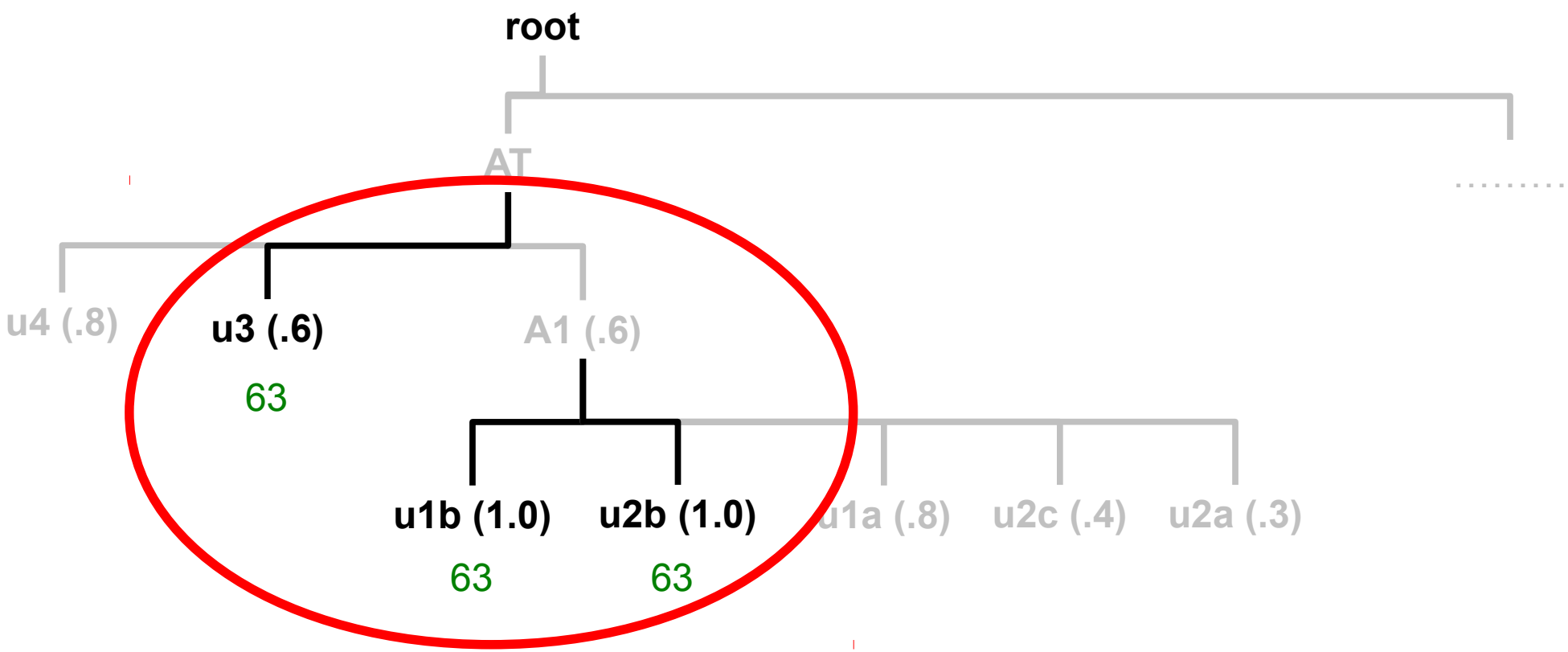
rank = 58

Fair Tree: Tie Handling



u3, u1b, and u2b receive equal rankings

Fair Tree: Tie Handling



- u3 tied A1, thus it tied A1's highest rank user, u1b
- u1b tied its adopted sibling user, u2b, since A1 and A2 were tied
- u3, u1b, and u2b receive the same fairshare factor