HASHGRAPH CONSENSUS: DETAILED EXAMPLES

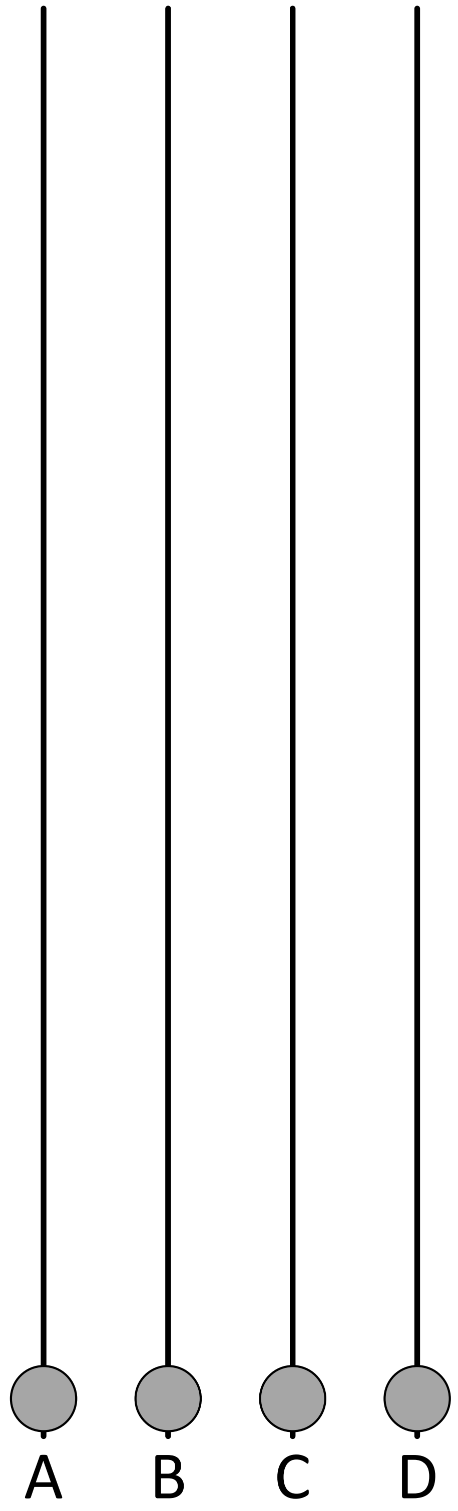
LEEMON BAIRD BAIRD@SWIRLDS.COM

DECEMBER 11, 2016

SWIRLDS TECH REPORT SWIRLDS-TR-2016-02

ABSTRACT: The Swirlds hashgraph consensus algorithm is explained through a series of examples on a hashgraph. Each page shows the hashgraph with annotations explaining a step of the algorithm. This covers the core algorithm, from creating transactions, through finding their consensus order and timestamps. The important terms are defined and illustrated on the following pages:

Revision date February 20, 2018

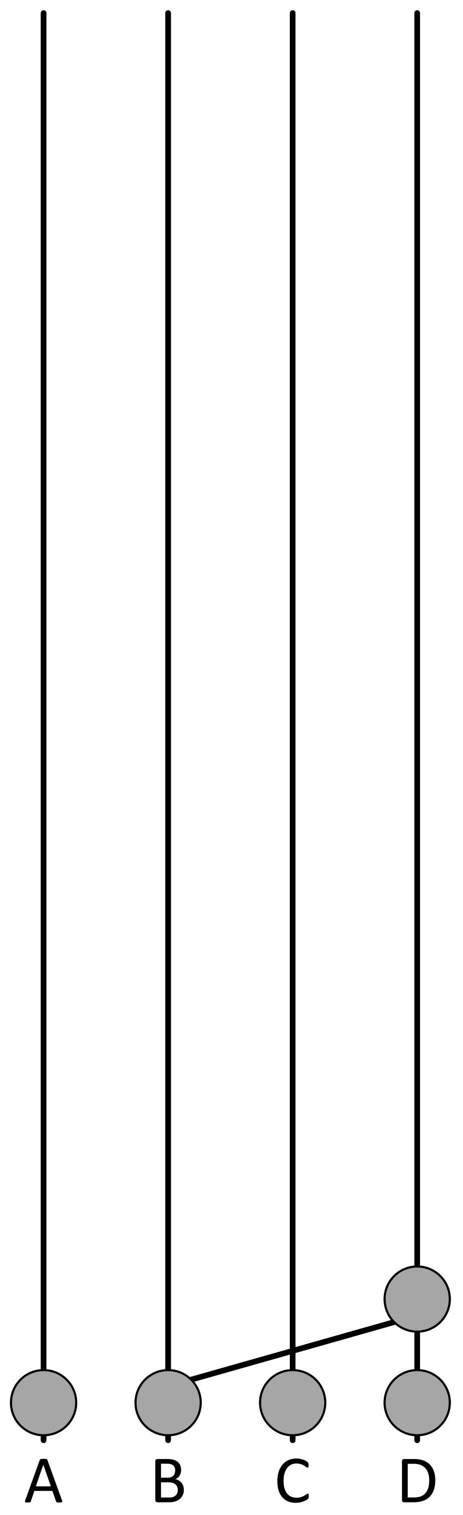


This figure is a *hashgraph*. It grows upward over time. Every participant keeps a copy of it in memory.

In this example, there are four *members* (full nodes) in the network. The members are Alice, Bob, Carol, Dave, and are represented by the 4 lines labeled A, B, C, D.

Each member starts by creating an *event*, which is a small data structure in memory, and which is represented here by a gray circle.

Each event is a container for zero or more *transactions*. The goal of the Swirlds hashgraph consensus algorithm is for the members of the community to come to a *consensus* (agreement) on the *order* of the events (and thus the order of transactions inside the events), and to agree on a *timestamp* for each event (and so for each transaction). It should be hard for attackers to prevent consensus, or to force different members to come to a different "consensus", or to unfairly influence the order and timestamps that are agreed.

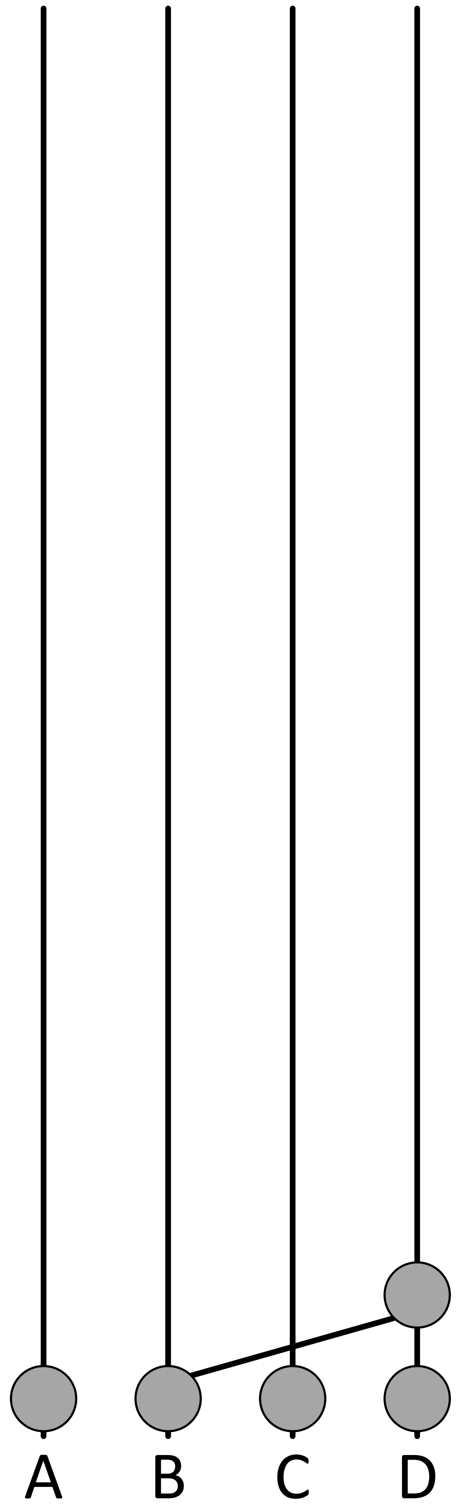


The community runs a *gossip protocol*, which means that each member repeatedly calls others at random to sync with them.

In this case, Bob randomly chose to call Dave. When they connected over the internet, Bob sent Dave all the events he knew that Dave did not yet know. In this case, it was just one event: the one that Bob had created at the start.

Dave records the fact that this sync happened by creating a new event. This is the new circle, which has lines going straight down to his own last event, and diagonally down to Bob's last event. Thus, the graph of events forms a record of how the members have communicated.

TECHNICAL DETAIL: Bob can avoid sending Dave events he already knows. Bob first tells Dave how many events he knows about that were created by each member (i.e., 4 integers). Dave tells Bob the same. Then they will both know exactly which events each should send the other. If Bob has 13 events by Alice and Dave has 10, then Bob sends Alice's last 3 events.

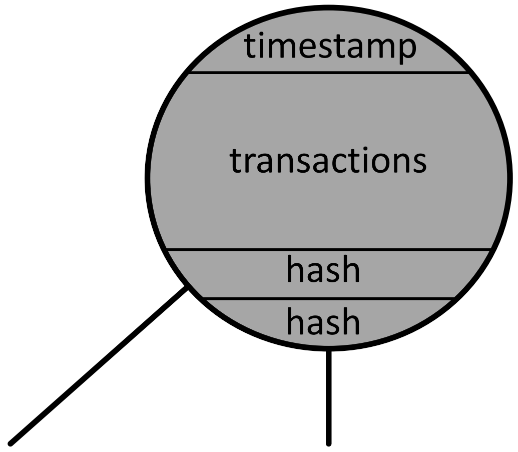


Dave's new event is illustrated here.

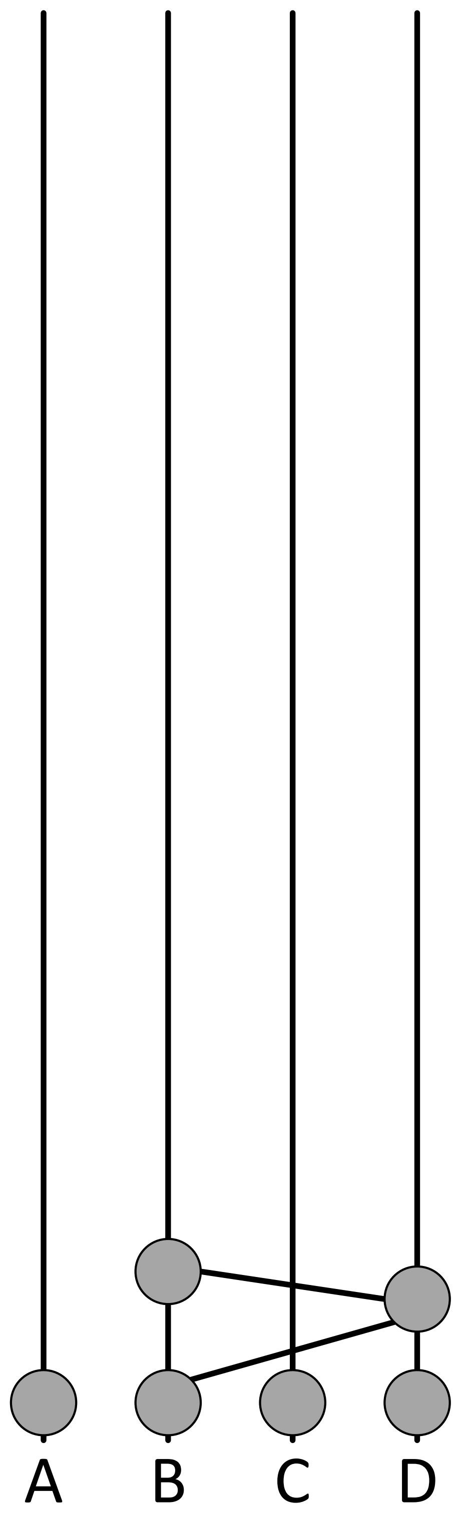
An event is a data structure containing the two hashes of the two events below itself (its *self-parent* and its *other-parent*). In this case, the self-parent is Dave's first event, and the other-parent is Bob's first event.

The event can optionally contain zero or more *transactions* that Dave wants to send to the network. So an event is a container for transactions. Dave also puts a timestamp of when he created the event. He then digitally signs it. When this event is gossiped, the signature will be sent along with it.

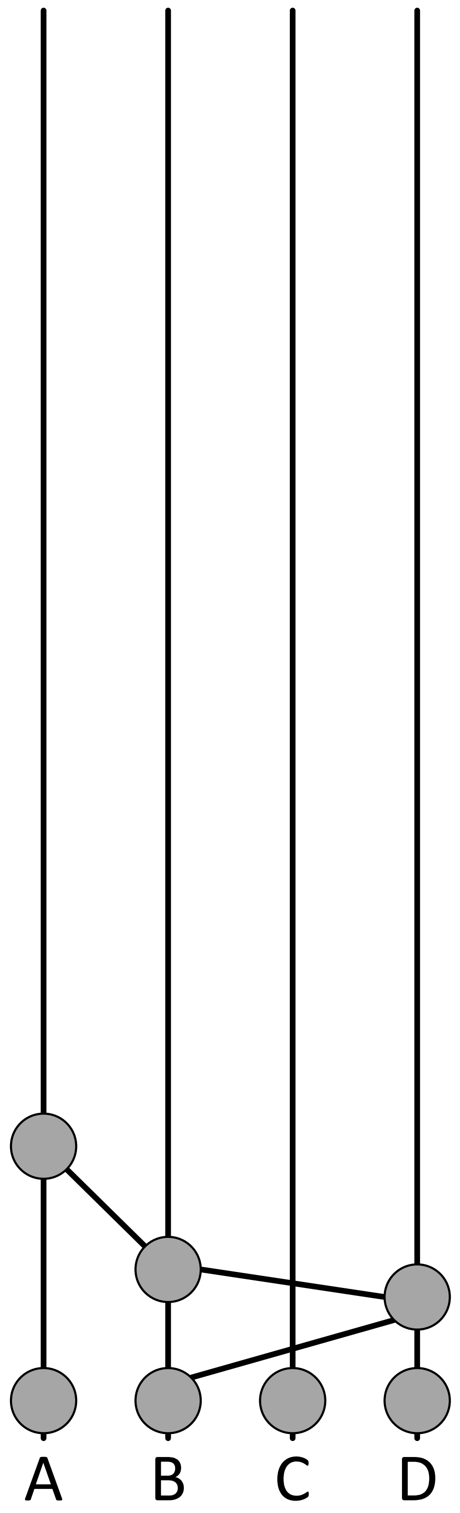
Event (signed by creator):



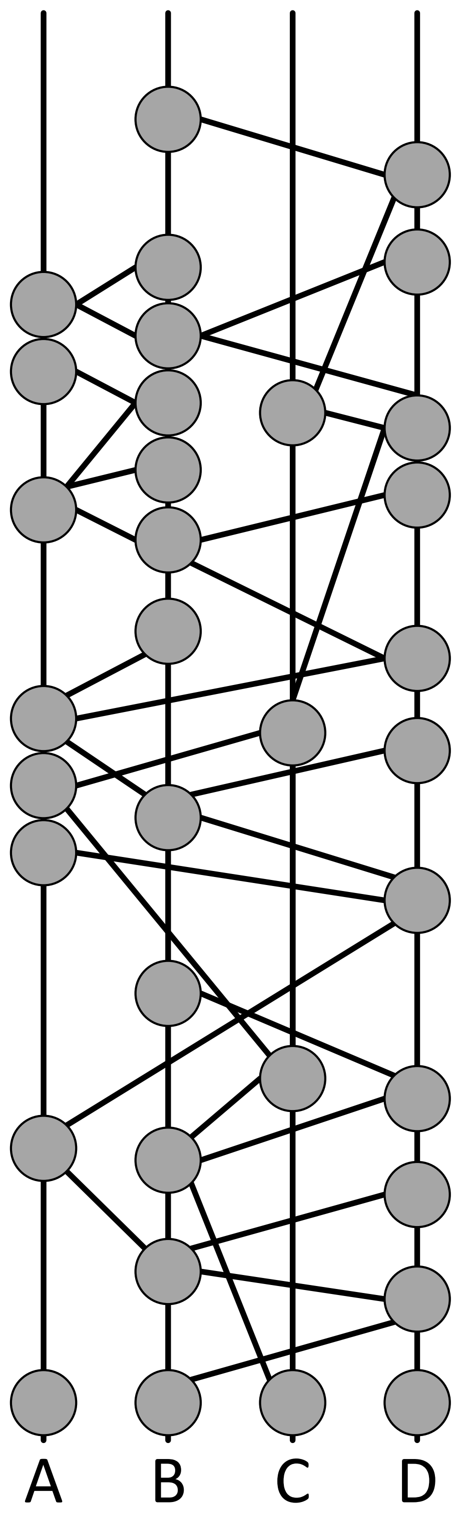
Swirlds



Dave then sends Bob all his events (including the new one he just created). Bob then creates a new event recording the fact they synced, and including the hashes of the most recent event by himself and the most recent event by Dave.



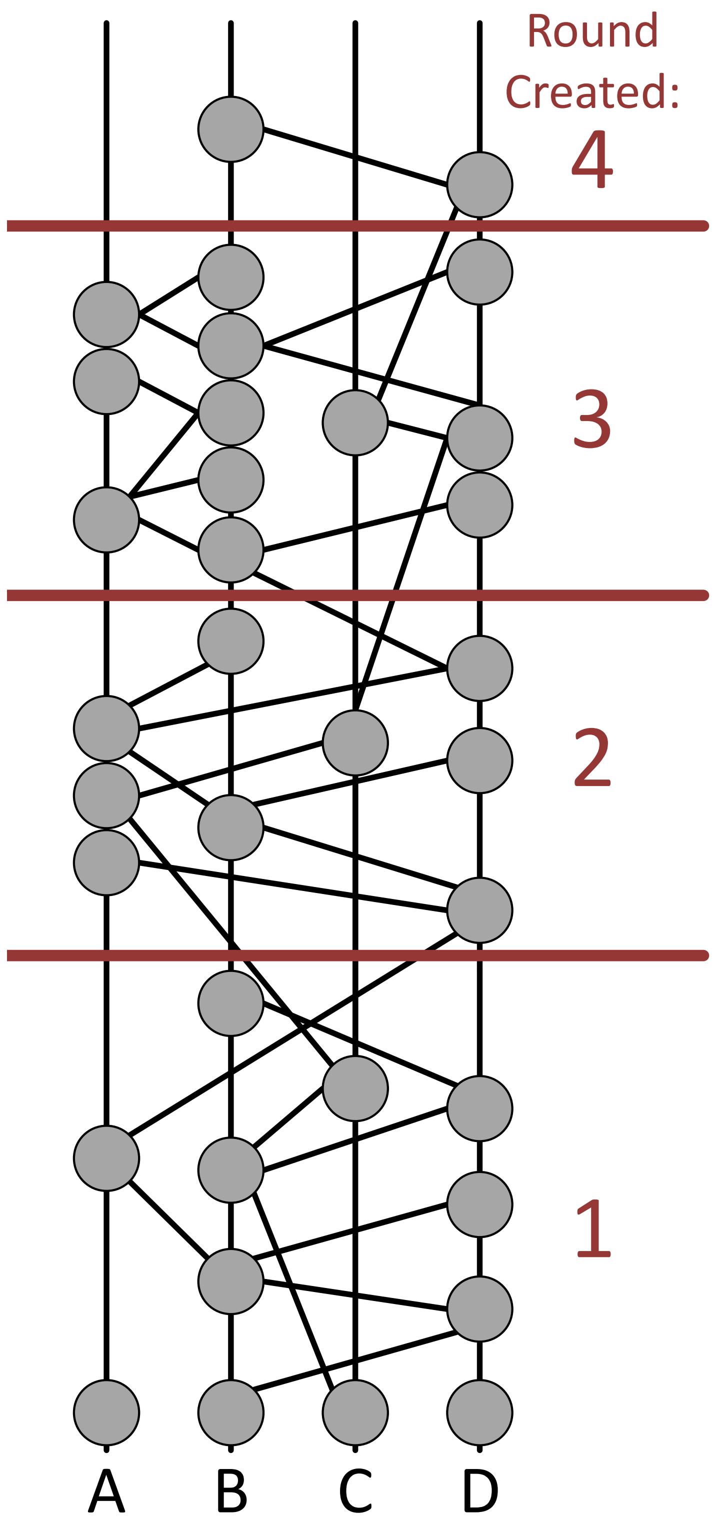
Bob then randomly chooses Alice, and sends her all 4 events he knows about. She creates a new one.



This continues forever, growing a directed acyclic graph upwards forever.

This is a *graph* connected by cryptographic *hashes*, so it is called a *hashgraph*.

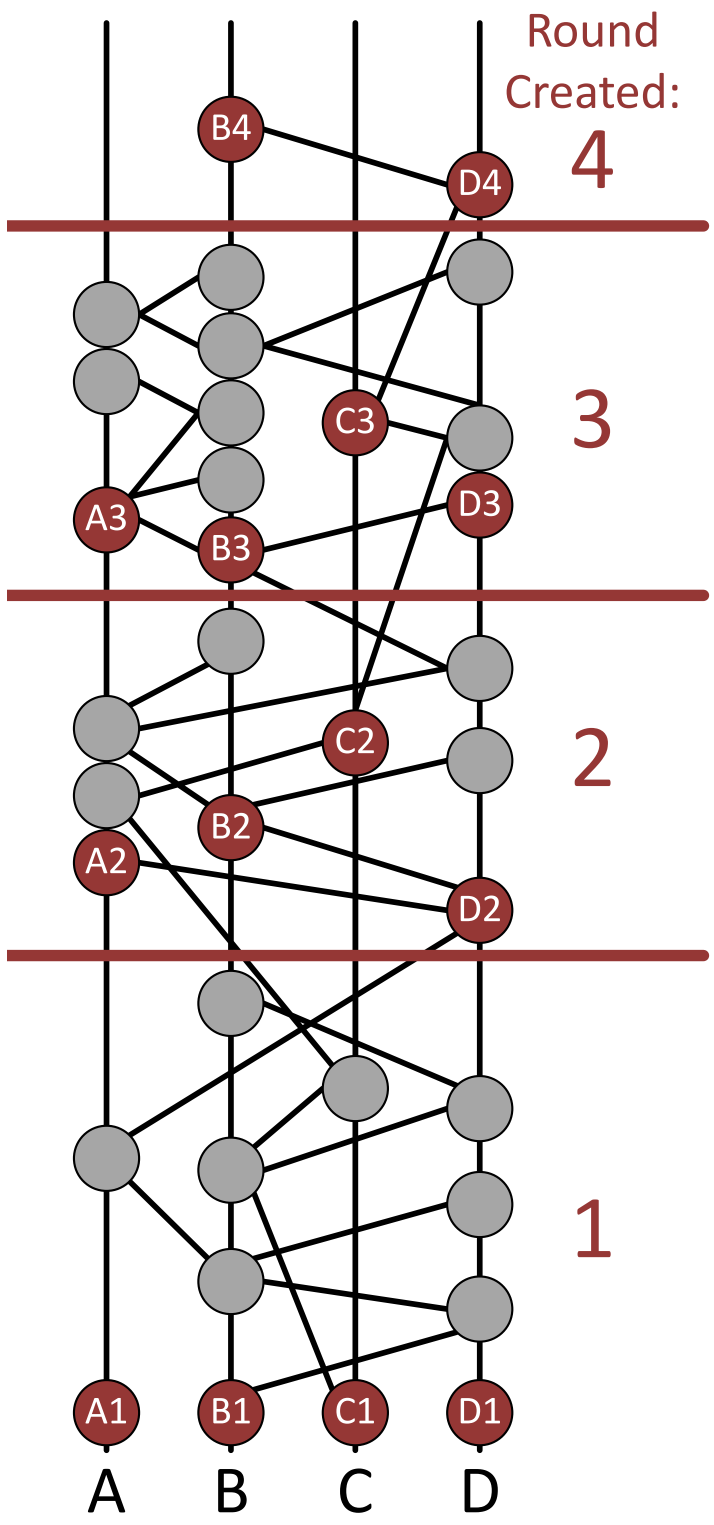
Each event contains the hashes of the events below it and is digitally signed by its creator. So the entire graph of hashes is cryptographically secure. It can always grow, but the older parts are immutable, as strong as the cryptographic hash and signature system used.



It is useful to define a *round created* for each event. A child never has a round created before one of its parents. So as time flows upward in the diagram, the round created can only stay the same or increase.

A later slide will describe how the round created is calculated. The important point is that as soon as you receive an event in a sync, you can immediately calculate its round created. And anyone else receiving it will calculate the same number. Guaranteed.

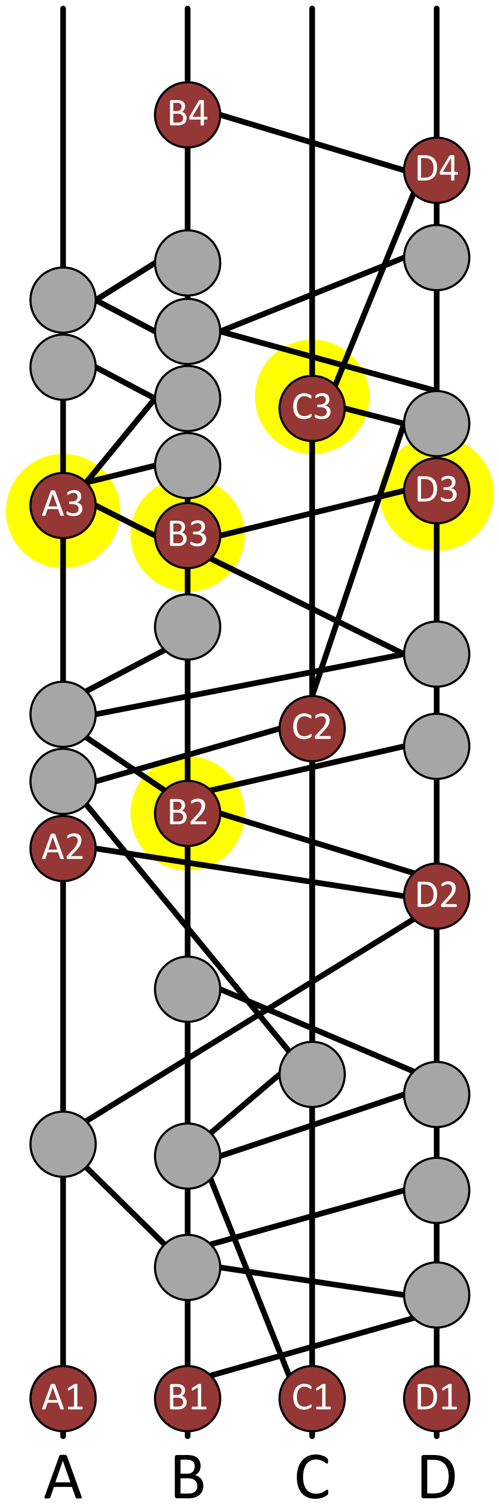
TECHNICAL DETAIL: The definition is: the round created for an event is R or R+1, where R is the max of the round created of its parents. It is R+1 if and only if it can strongly see a supermajority of round R witnesses. This is explained in greater detail in later slides.



The first event that Alice creates in each round is called a *witness*. Her witnesses are labeled A1, A2, and A3 here. The other members create witnesses similarly.

It is possible for a member to have no witnesses in a given round.

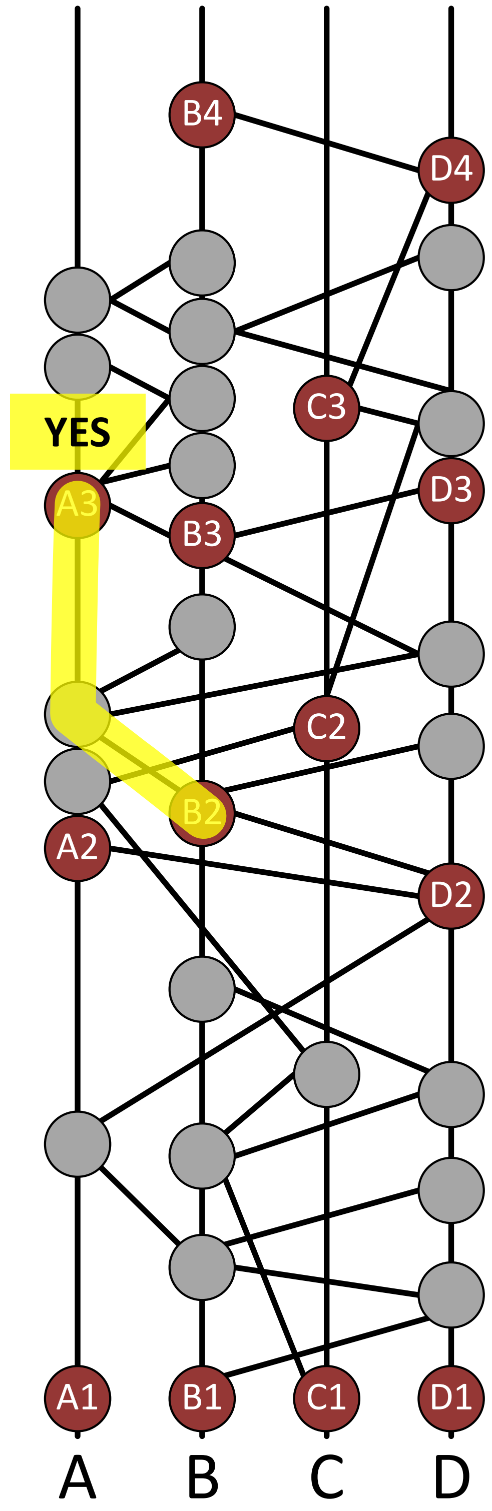
TECHNICAL DETAIL: It is possible for a member to cheat by *forking*, or creating two events with the same self parent. In that case, there might be two witnesses in the same round by the same member. There are theorems proving that this won't matter.



For each witness, we need to determine if it is a *famous witness*. For example, we will determine if the witness B2 is a famous witness.

This is done by considering the witnesses in the next round. So the fame of B2 will be determined by first considering the witnesses A3, B3, C3, and D3. The idea is for B2 to count as famous if it is seen by many of the witnesses in the next round.

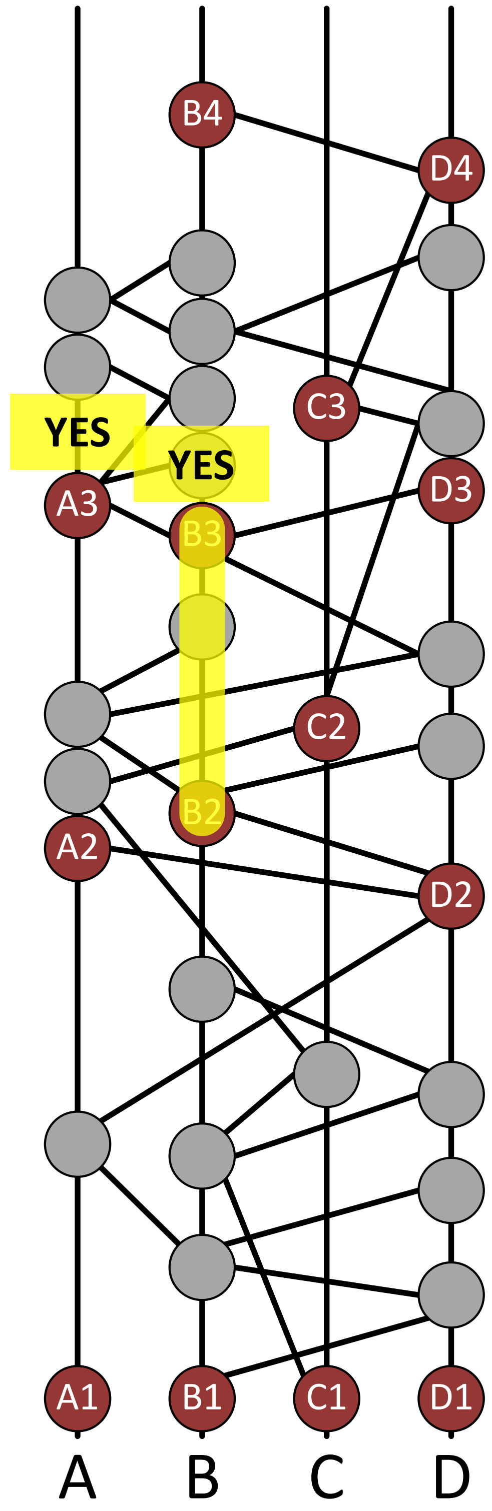
There is now an *election*, in which each of those witnesses will *vote* on whether B2 is famous. There will be a separate election for every witness, to determine its fame.



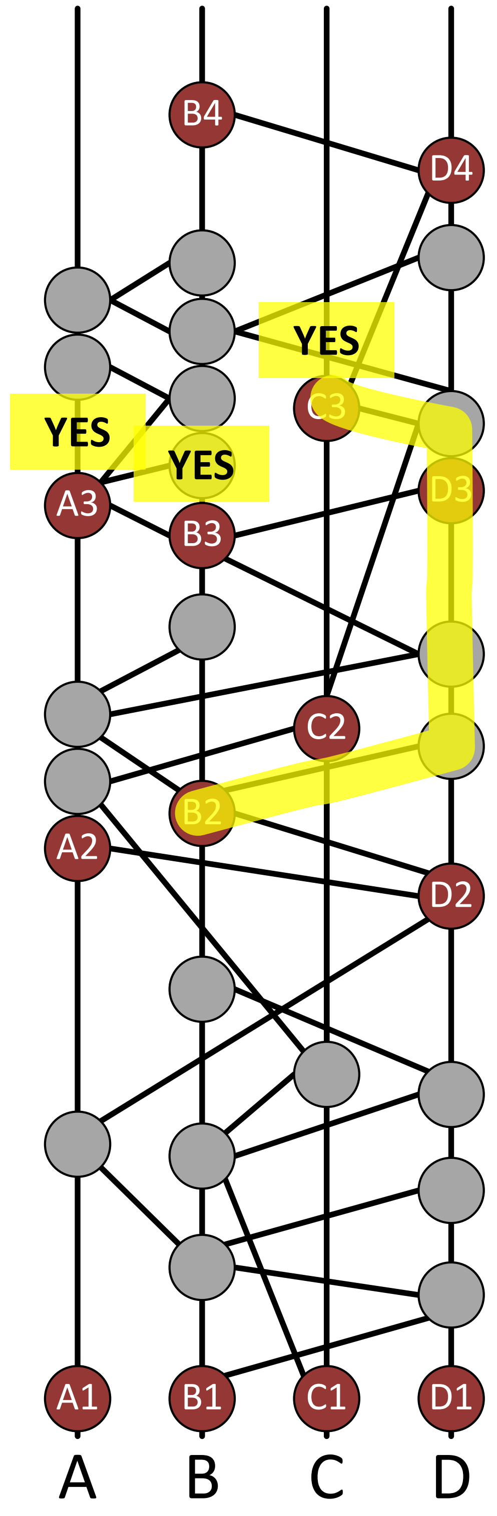
The witness A3 can *see* B2. That means that there is an entirely-downward path from A3 to B2. In other words, B2 is an ancestor of A3. And A3 is a descendent of B2

A3 can see B2, so A3 will vote YES in the election for whether B2 is famous.

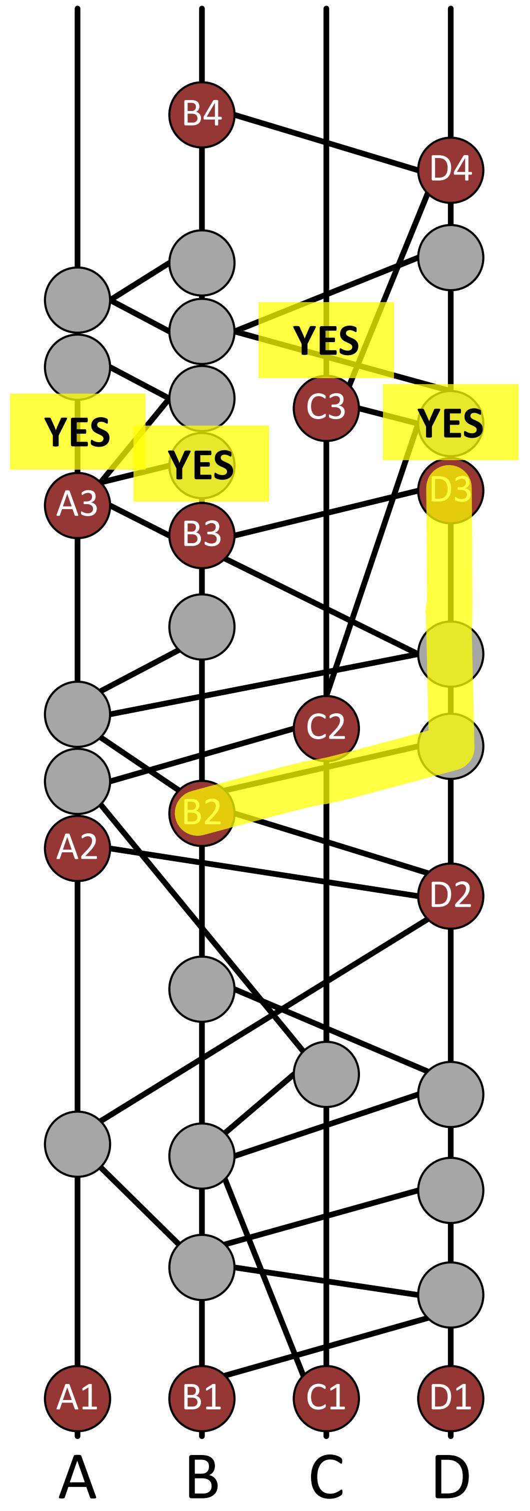
TECHNICAL DETAIL: A3 sees all its ancestors except for those created by a member who created a fork that is an ancestor of A3. In other words, A3 can see B2 if B2 is an ancestor of A3, and the creator of B2 (who is Bob) did not create two events X and Y that both have the same self-parent and are both ancestors of A3. So "seeing" is the same as "ancestor", except you can't "see" cheaters.



B3 sees B2, so it votes YES.



C3 sees B2, so it votes YES.

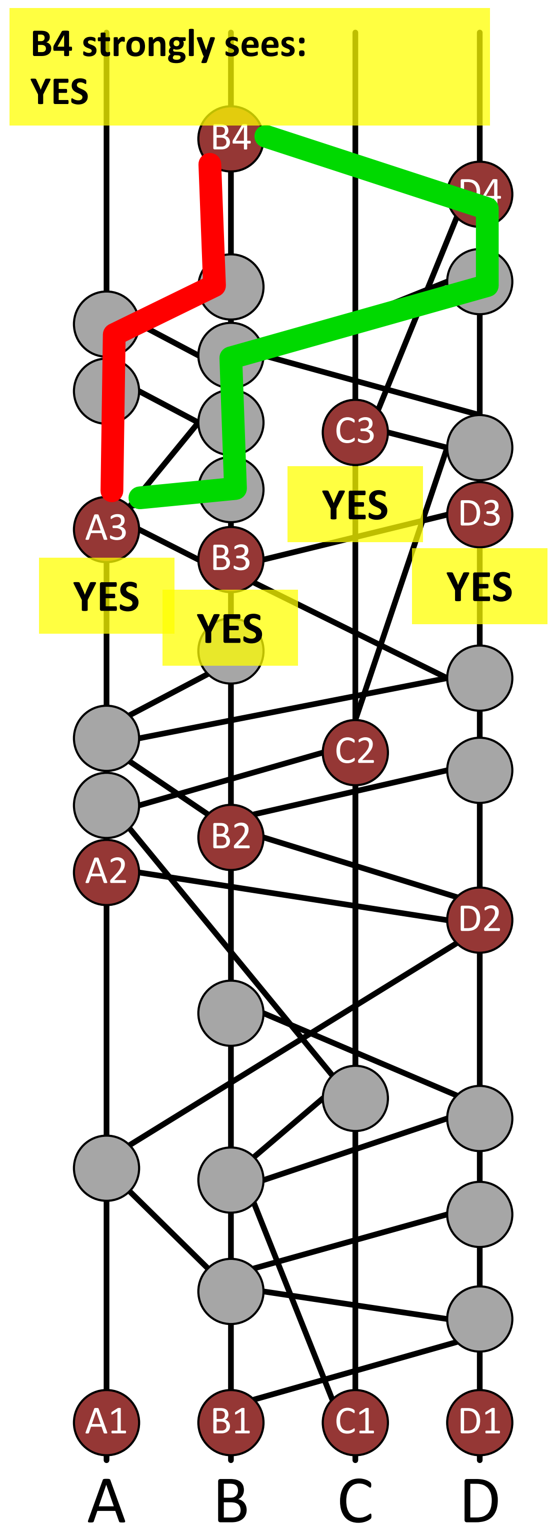


D3 sees B2, so it votes YES.

All four witnesses voted YES, so we would expect that B2 will be declared to be famous. But the election isn't over yet! An election isn't over until the votes are counted.

The votes will be counted by the witnesses in the following round. So B4 will count the votes. And D4 will also count the votes.

The hashgraph doesn't yet have an A4 or C4. But as time goes on and more gossiping occurs, there may eventually be an A4 and C4, and then they will count the votes, too.



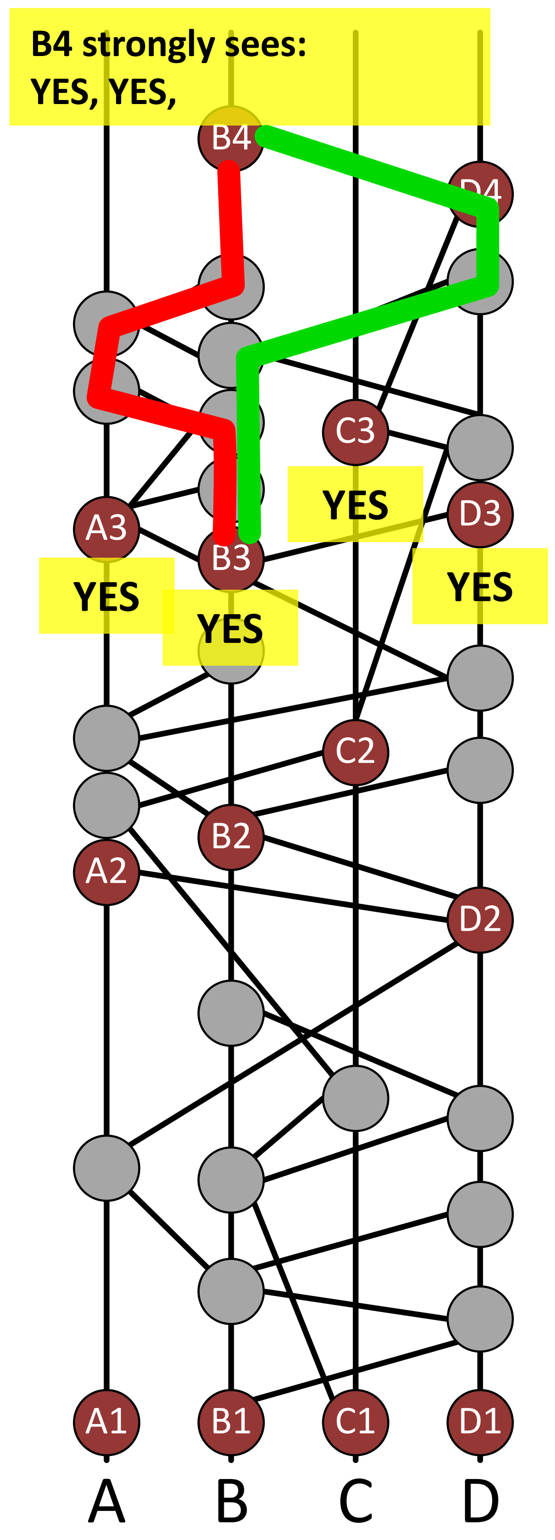
B4 will collect the votes from every round-3 witness that it *strongly sees*. To strongly see a witness, it isn't enough for there to be a single downward path to it. There must be enough different paths to it so that together, the paths go through a supermajority of the population.

A *supermajority* is any number that is more than two thirds of the population. In this example, there are 4 members in the population, so any 3 of them constitute a supermajority.

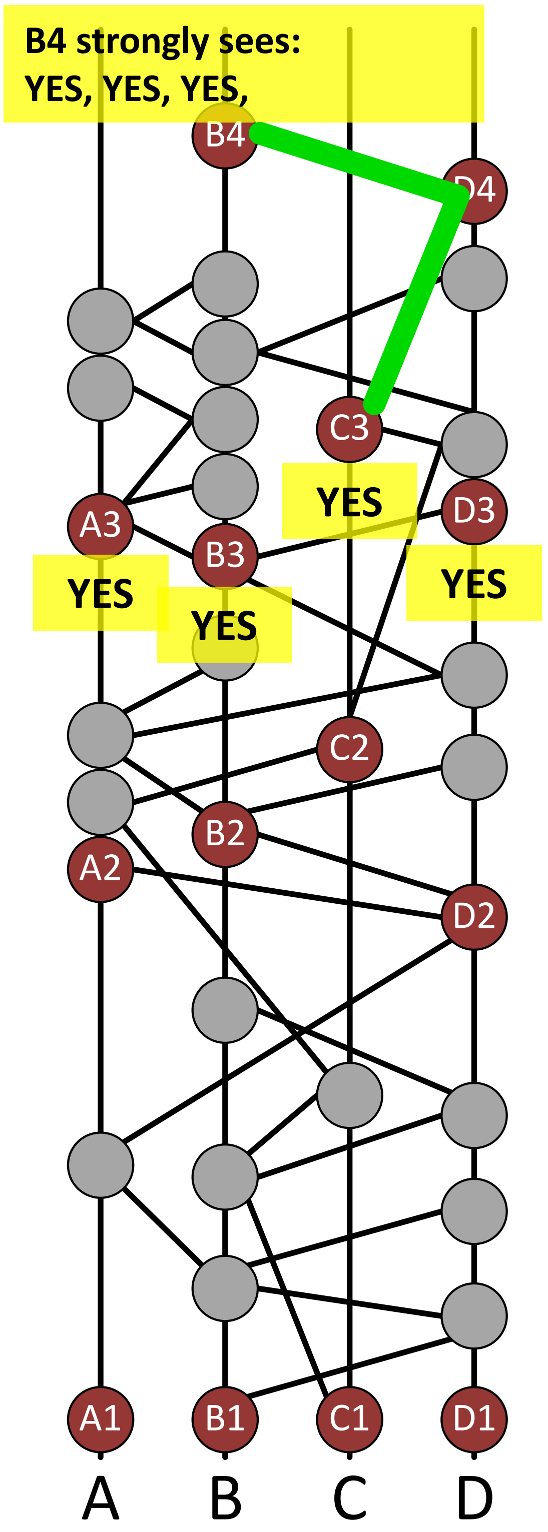
In this example, B4 is able to strongly see A3. The red path goes from B4 to A3 through Alice and Bob. The green path goes through Alice, Bob, and Dave. There are no paths from B4 that go through Carol to get to A3. But that's OK, because Alice, Bob, and Dave make up a supermajority. So Carol isn't needed. In fact, the green path alone would have been enough. The red path wasn't needed.

So, B4 can strongly see A3. Therefore, B4 collects the vote from A3 (which is YES).

Swirlds

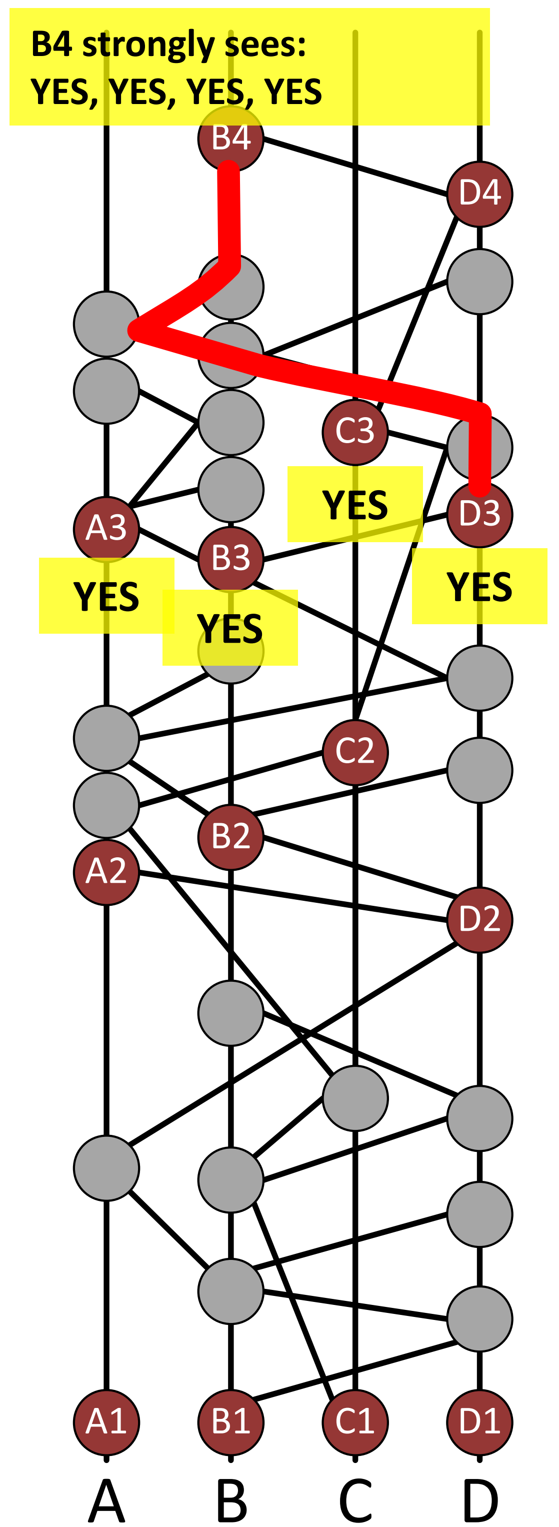


B4 strongly sees B3, because the red path goes through Alice and Bob, and the green path goes through Bob and Dave. In this case, both paths were needed to reach the supermajority.

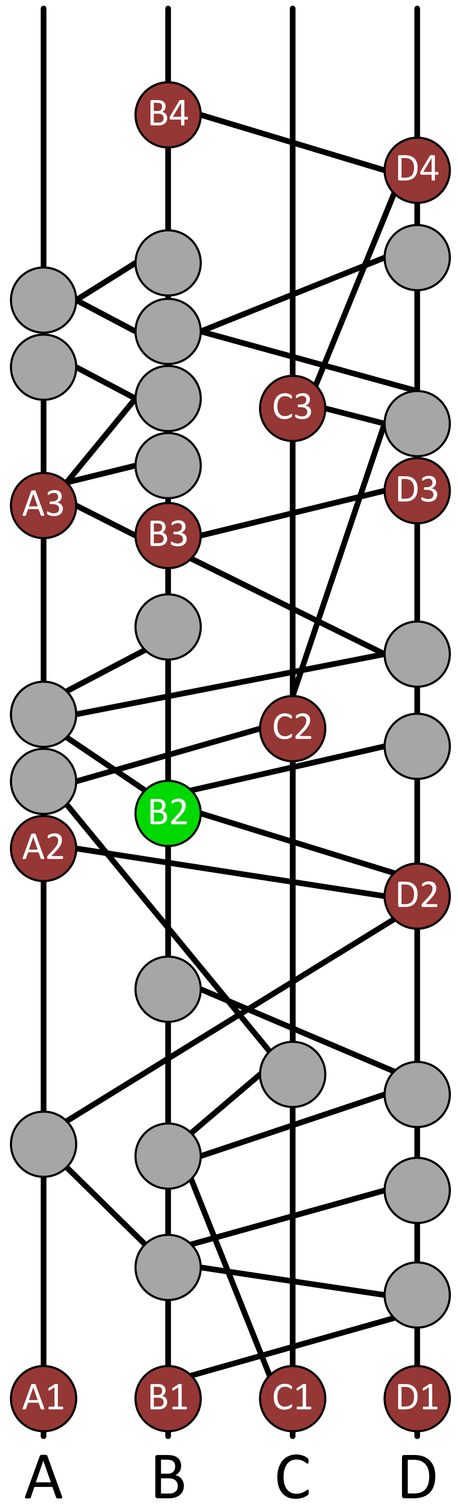


B4 strongly sees C3, and so collects another YES vote.

TECHNICAL DETAIL: if a path starts at Bob and ends at Carol, then it automatically counts as going through Bob and Carol. In other words, the endpoints of the path are counted, too.



B4 strongly sees D3, and so collects another YES vote.

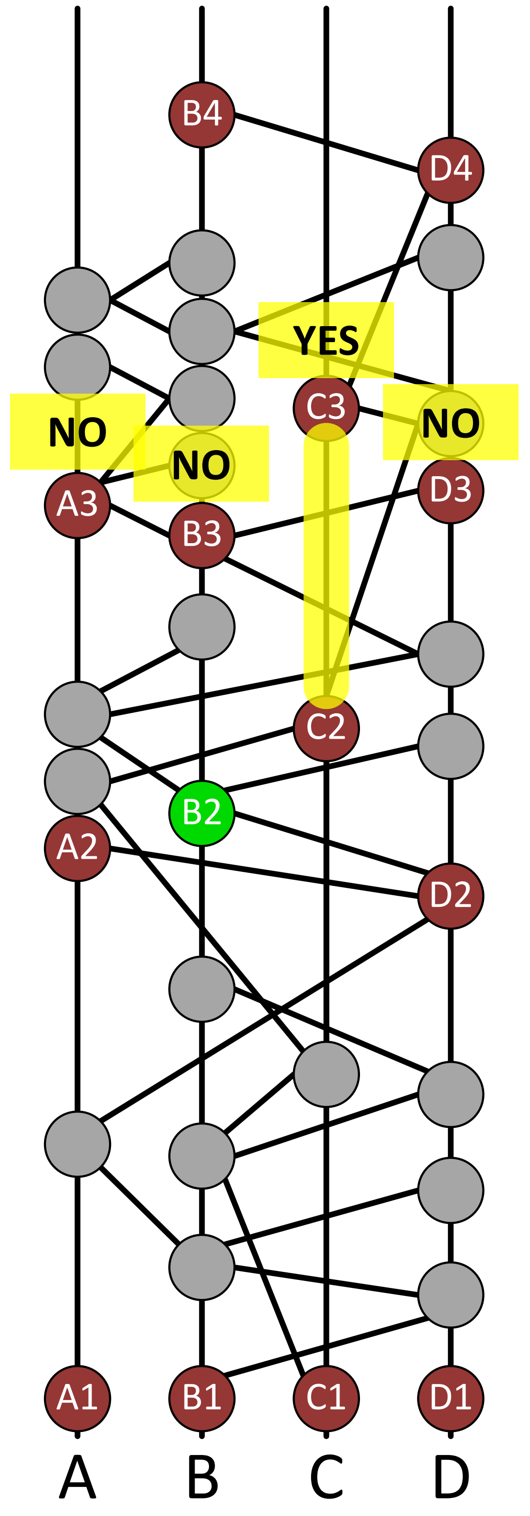


At this point, B4 has received YES notes from a supermajority, so B4 *decides* that the election result is YES. Color B2 green to show that it is now famous. That is the consensus decision.

If B4 had seen 3 YES and 1 NO, it would still decide YES, because that's a supermajority.

If B4 had seen 3 YES votes and no other votes (because it couldn't strongly see one of the witnesses), it would still decide YES, because that's a supermajority.

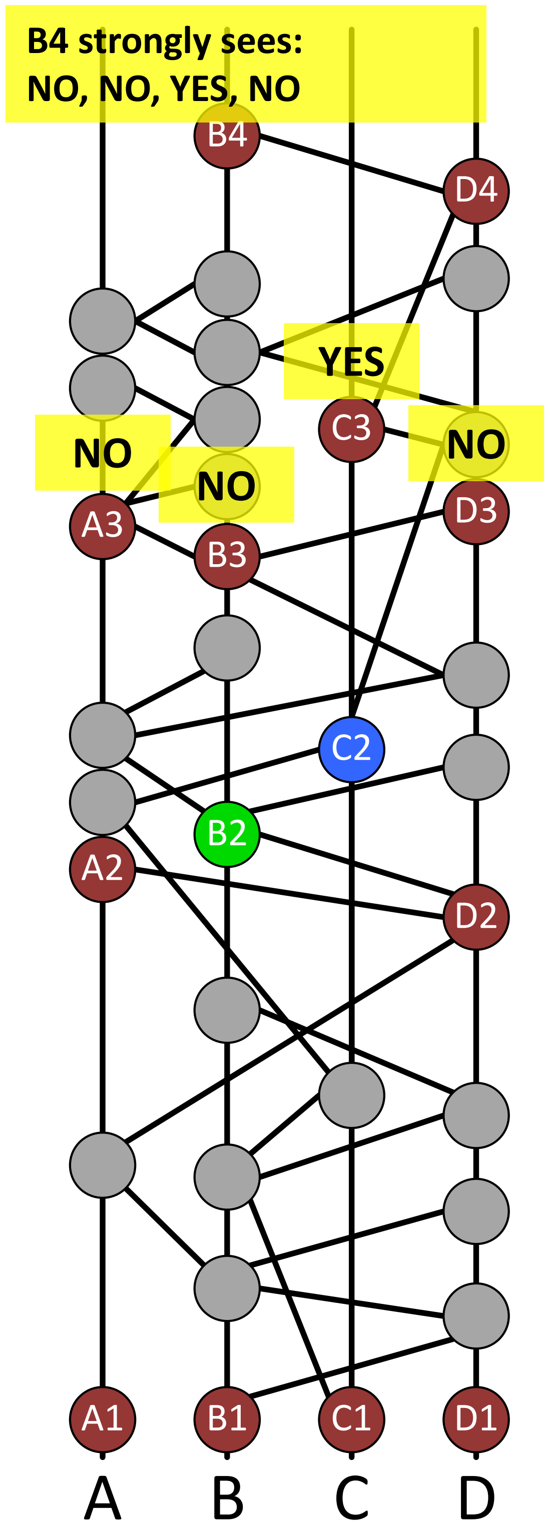
We need for B4 to strongly see a supermajority of witnesses, in order to even have a chance at deciding. Therefore, we use this to define the *round created*. If an event X has parents with a maximum round created of R, then that event will usually be round R, too. But if that event can strongly see a supermajority of round R witnesses, then that event is defined to be round R+1, and so is a witness. In other words, an event is promoted to the next round when it can strongly see a supermajority of witnesses in the current round.



Now we run an election for whether C2 is famous.

The yellow path shows that C3 can see C2, and so C3 votes yes.

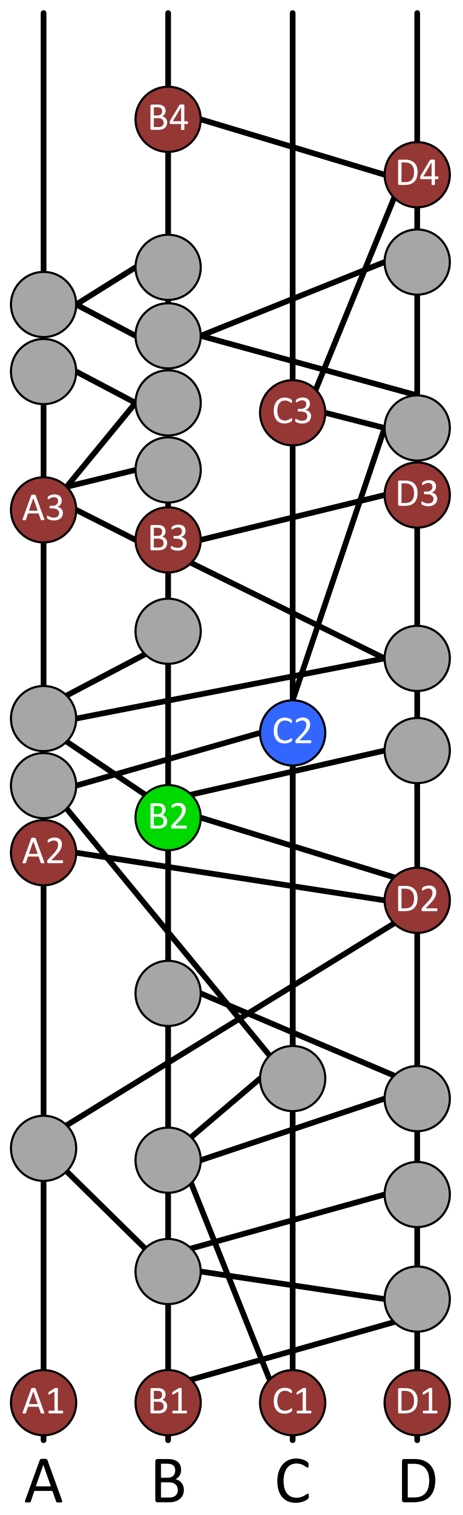
There are no downward paths from A3, B3, or D3 to C2, so they all vote NO.



Since B4 strongly sees all of A3, B3, C3, and D3, it will therefore collect votes from all of them.

The votes are NO, NO, YES, NO. So a supermajority is NO. So it decides NO.

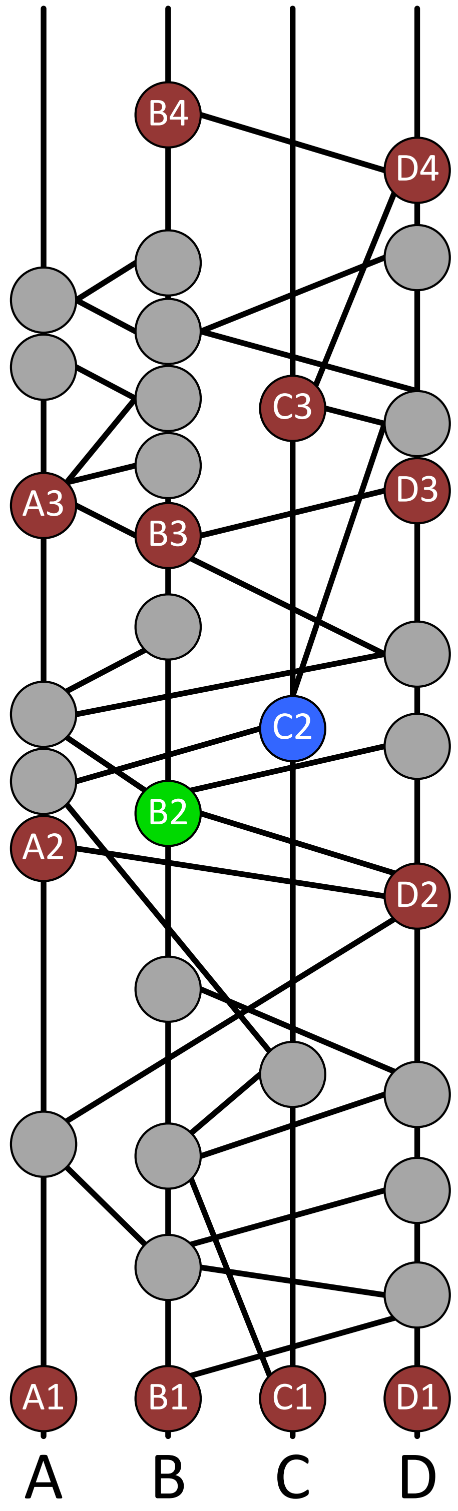
The election is over. C2 is not famous. Color it blue to show that it is not famous.



There is a theorem that if any witness is able to “decide” yes or no, then that is the result of the election, and it is guaranteed that all other witnesses that decide are going to decide the same way.

In this example, B4 was able to decide the election. If it had collected votes that were more evenly split between YES and NO, then it would have failed to decide. In that case, we can consider D4. If D4 also fails to decide, then perhaps A4 or C4 might decide.

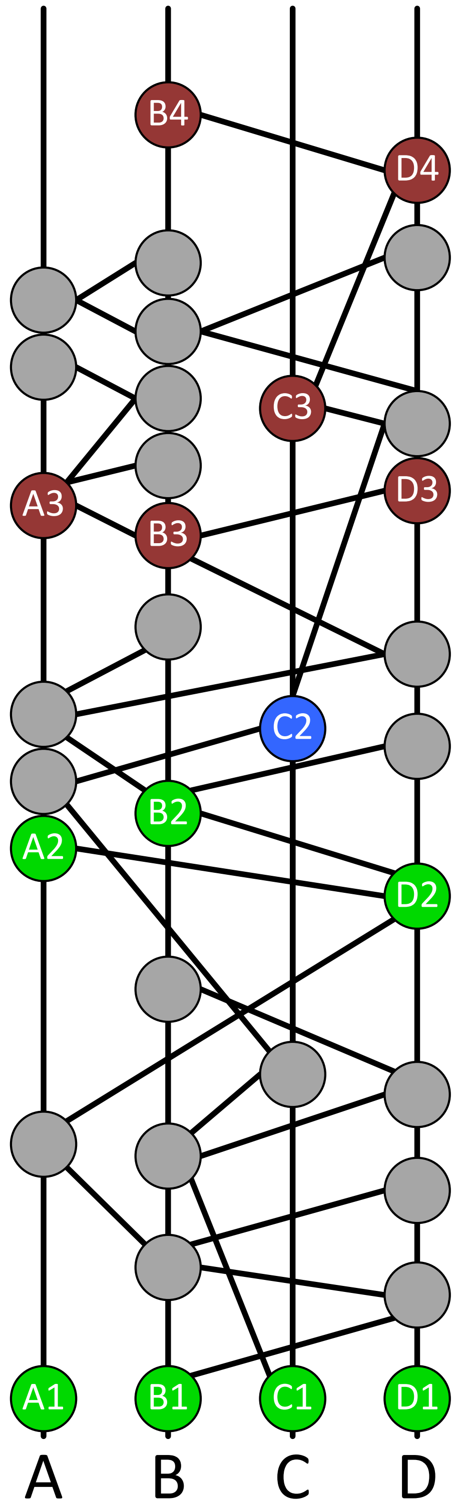
If none of the round-4 witnesses can decide, then each of them will simply vote in accordance with the majority of the votes they collected (voting YES in case of a tie). In that case, it will be up to the round-5 witnesses to collect votes from the round-4 witnesses. Perhaps the round-5 witnesses will be able to decide.



The voting continues until it eventually reaches a round where some witness can decide the election.

There is a theorem saying that the election will eventually end (with probability one) as long as we add in a coin round every 10th round of voting.

In a coin round, collecting a supermajority causes a witness to merely vote (not decide). And a non-supermajority causes it to vote pseudorandomly, by using the middle bit of its own signature as its vote.



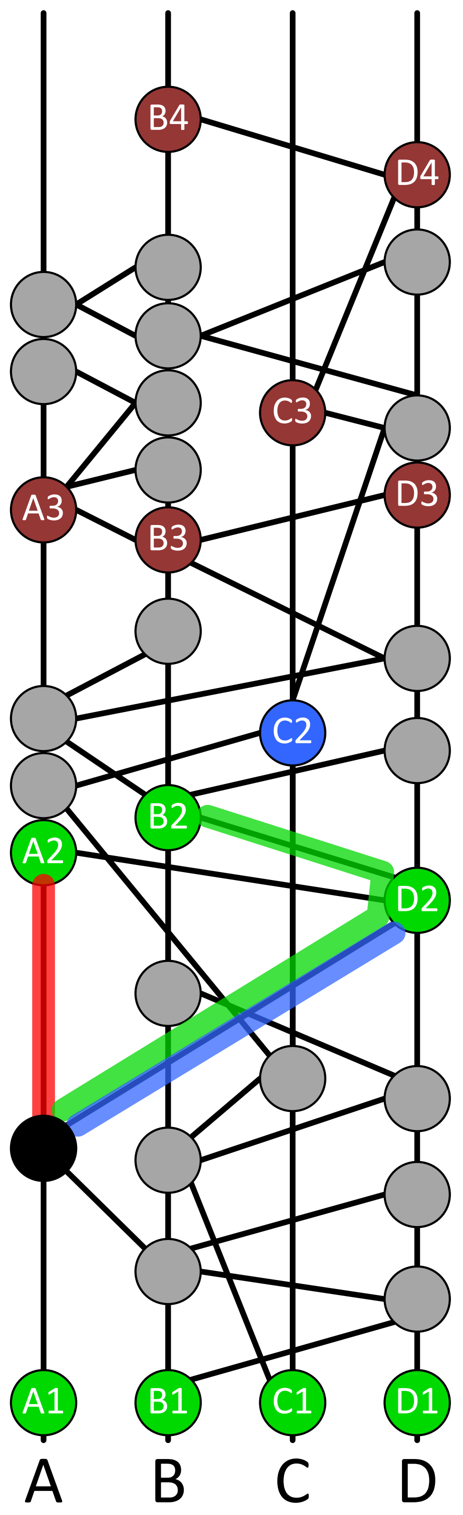
6 more elections are run. They decide that A2, D2, A1, B1, C1, and D1 are all famous.

In normal operation, most events are not witnesses, so there is no election for most events. And most witnesses are declared famous with an almost-unanimous vote in the first round of voting. So most elections do not last very long.

Notice that in this example, we have now decided the fame of every witness in round 2. Once a round has the fame decided for all of its witnesses, it is possible to find the *round received* and find the *consensus timestamp* for a new set of events.

Start by considering the gray event immediately below A2.

TECHNICAL DETAIL: If a member forks, they might have two famous witnesses in the same round. In that case, neither of them are used further. Only the remaining ones (the "unique famous witnesses") are used to determine round received and consensus timestamp.



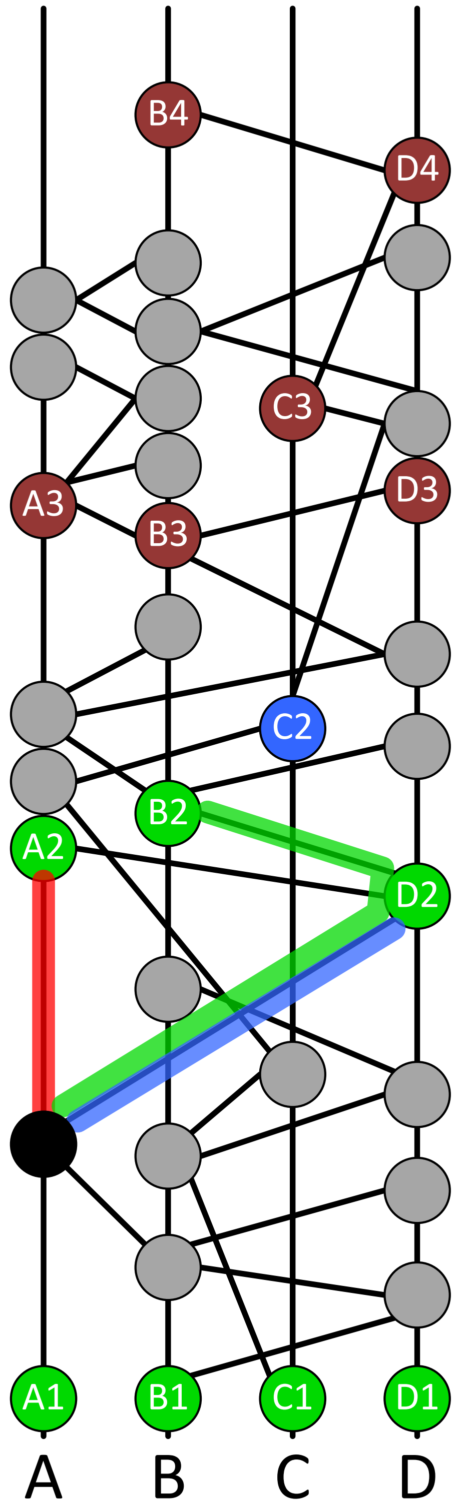
This event can be seen by every famous witness in round 2. The red, green, and blue paths show how A2, B2, and D2, respectively, can all see the black event.

This merely requires seeing, not strongly seeing.

This only requires seeing by the famous witnesses. It doesn't matter whether C2 can see the black event, because C2 is not famous.

Since the black event is seen by all of the famous witnesses in round 2 (but not in any earlier round), it is said to have a *round received* of 2.

TECHNICAL DETAIL: we don't need to limit ourselves to "seeing". It is sufficient to use the "ancestor" relationship instead. In other words, at this step, we don't worry about forking.

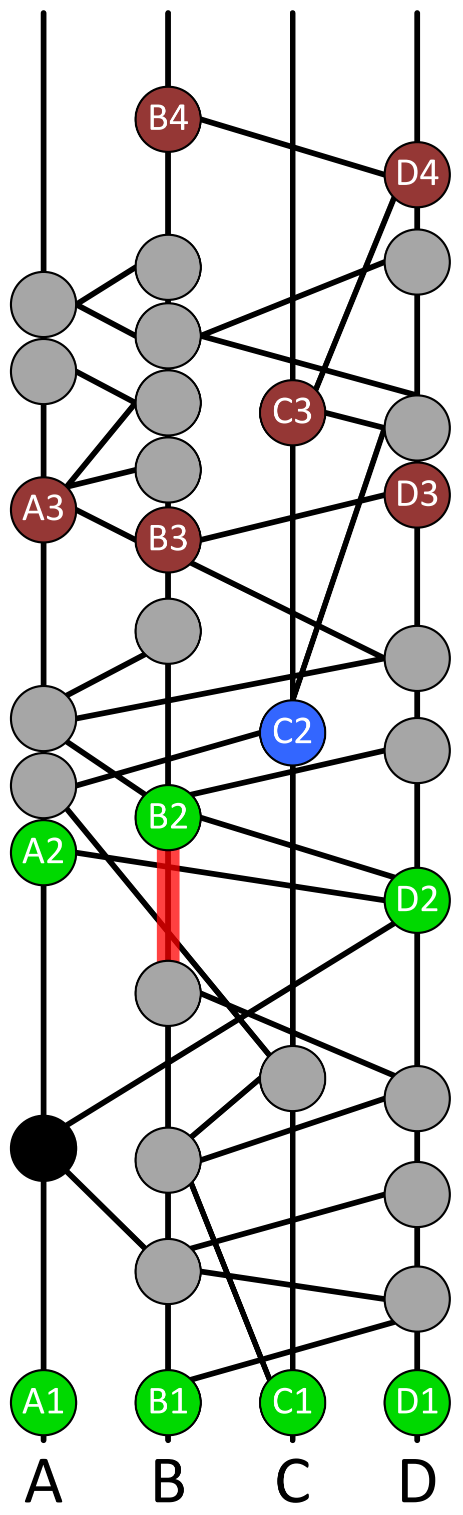


The consensus timestamp of the black event can be found as follows.

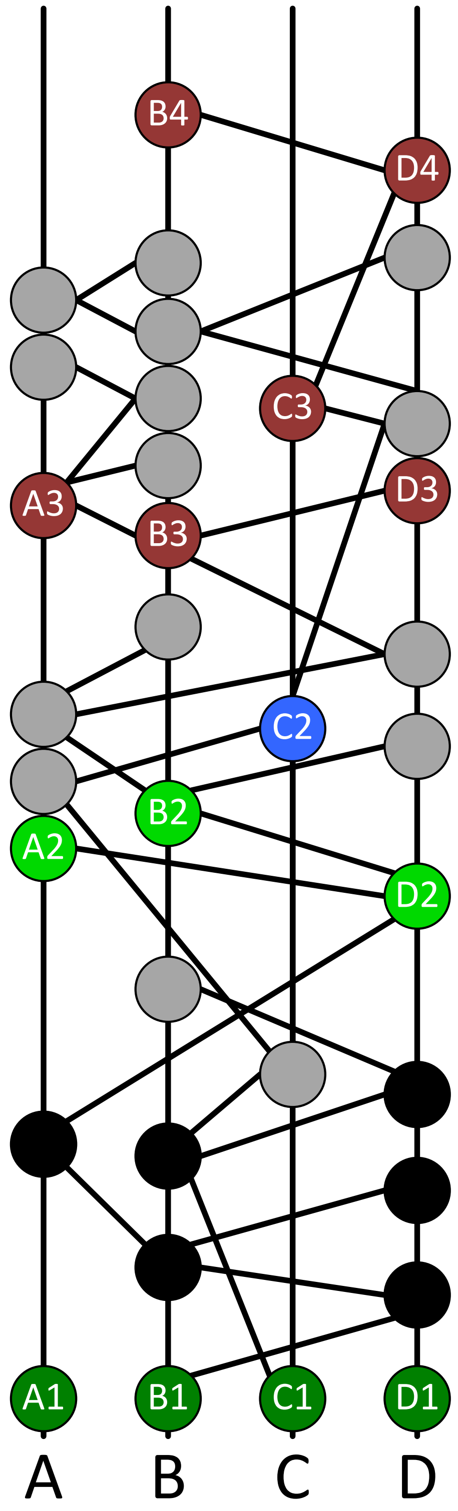
Find the earliest event X by Alice that is an ancestor of A2 and a descendent of the black event.

Similarly, find the earliest event Y by Bob that is an ancestor of B2 and descendent of the black event. And similarly for event Z by Dave.

Take the timestamps on the events X, Y, Z that were put in those events by their creators. Sort all of the timestamps in order. Take the middle one from the list (or the second of the two middle ones, if there are an even number of them). This median timestamp is the *consensus timestamp* for the black event.



Now consider the gray event below B2. It is seen by B2, but not seen by A2 or D2. So it was not seen by all the famous witnesses in round 2. So its received round will be later than round 2. Leave it colored gray to indicate that it doesn't yet have a received round.



Continuing, we find consensus for the 6 black events and the 4 dark green events. These are the 10 events that have a round received of 2. We need to sort these 10 events into an order that all members will agree on. This agreed order is the *consensus order*.

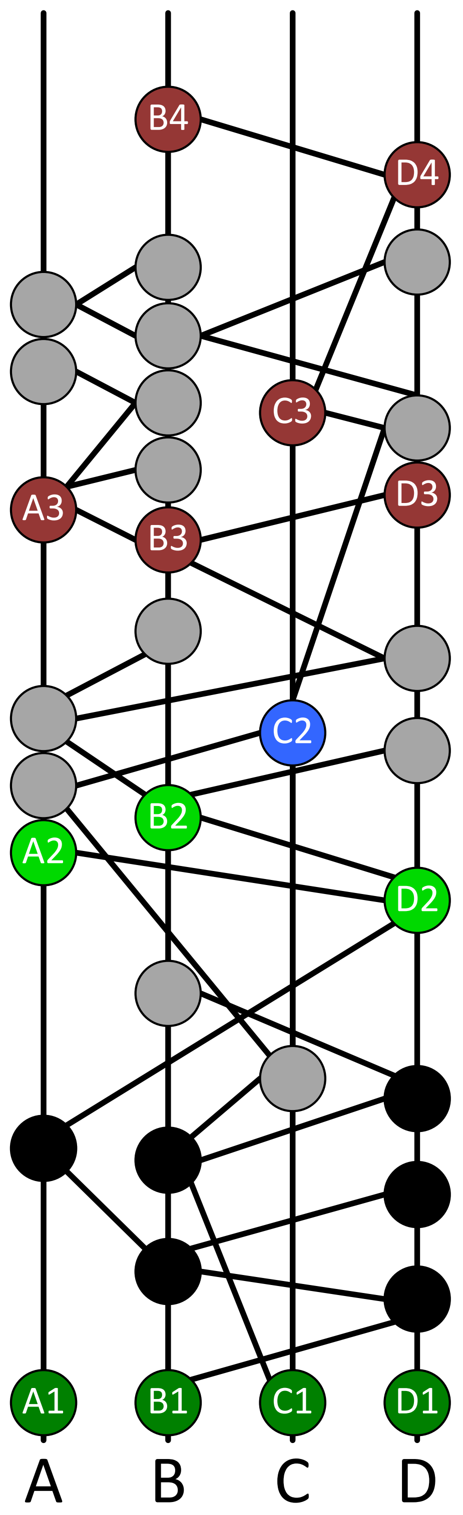
This is done by sorting them by *round received*.

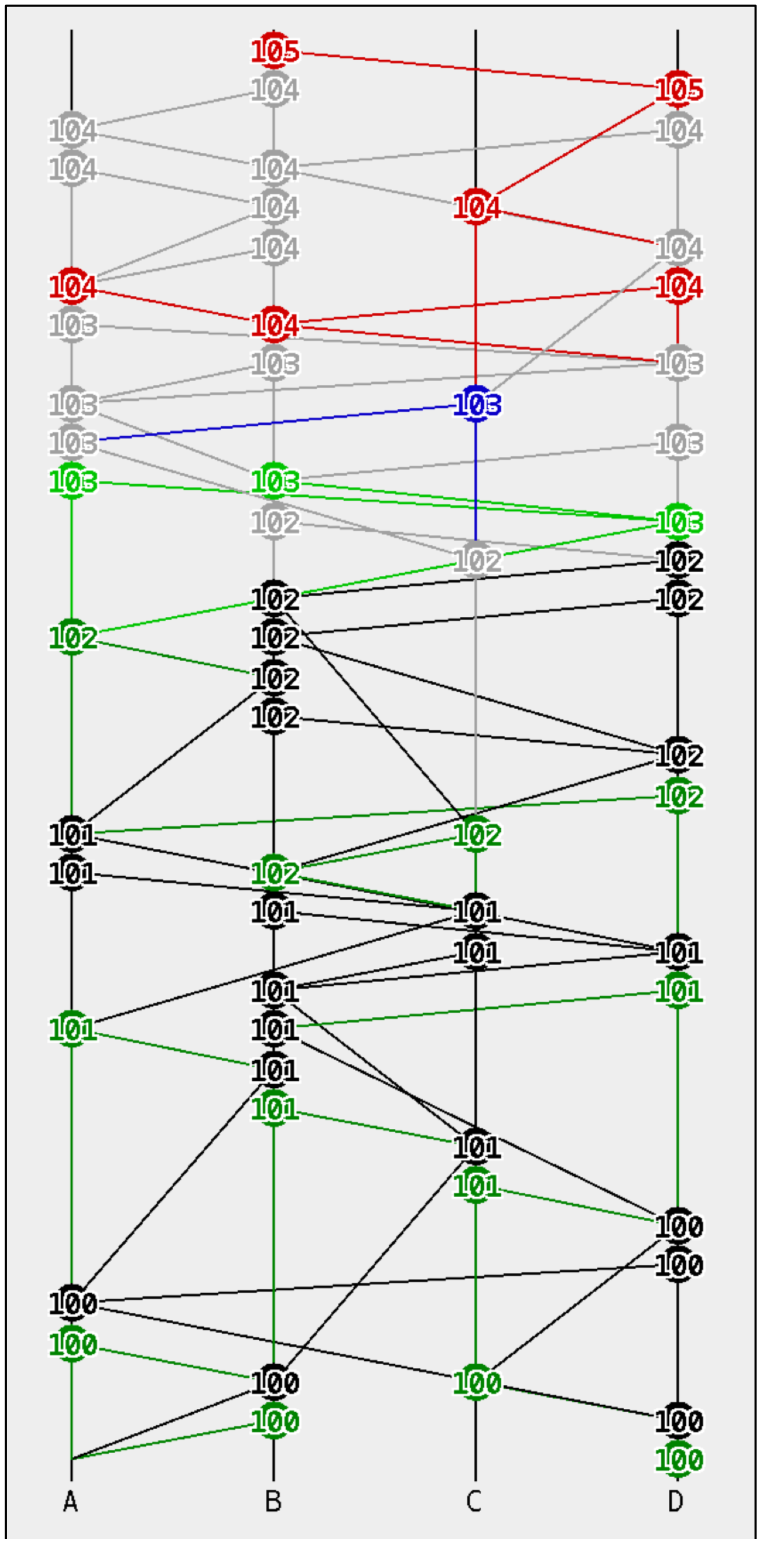
Ties are broken by sorting by the median timestamp (which is the *consensus timestamp*).

Further ties are broken by sorting by the extended median (which looks at more than just the middle element of each list).

Further ties are broken by sorting them by their signatures, after all the signatures have been XORed with a pseudorandom number. The pseudorandom number for a given round received is found by XORing the signatures of all the famous witnesses in that round.

Swirlds





This picture is a screenshot from the HashgraphDemo app that is part of the Swirlds SDK that can be downloaded from Swirlds.com.

This screenshot came from running it in slow mode with 4 members, with the checkbox checked to show the round created.

This screenshot shows the part of the hashgraph from about round 100 to 105. The example is a slightly-modified version of the top half of this screenshot.

Swirlds