Vector Clocks vs Logical Clocks

- logical clocks seem to be a little easier to implement
- but vector clocks can be used to gain more granular control over the ordering process as they represent more information
- we decided to go with vector clocks

Vector Clocks for Consistency I

- every server has its own clock
- when a client posts a message to a server, this server increments the corresponding value in its clock (e.g. server 3 changes its clock from [2, 1, **3**, 2, 1, 5, 2, 4] to [2, 1, **4**, 2, 1, 5, 2, 4]
- it then pairs this clock irreversibly with the corresponding post
- it sends this pair to all other servers

Vector Clocks for Consistency II

- all clocks are unique (because the value at the servers clock position is only changed by this server and every action (send, receive) results in a change
- we defined a strict total order on the clock vectors
- this order allows us to order the database (list of entries)
- as the clocks are unique, immutable and bound to their entries the order (and the content) are the same on all servers

⟹ the database is eventually consistent

The order on the clock vectors

- to compare two vectors we first sum up the values from all dimensions per vector: the clock [2, 1, 3, 2] has a sum of 2 + 1 + 3 + 2 = 8
- if the second vector has a smaller sum (e.g. 2 + 1 + 1 + 2 = 6) it means it logically came before the first clock (and its entry) so it will be first in the database
- if the sums are the same, the first different dimension pair decides over the precedence: for vectors **v1** = [2, 2, 4, 2] and **v2** = [2, 2, 5, 1] the algorithm will loop through both vectors simultaneously and stop when at the third position and decide that **v2 < v1** (v2 will be inserted before v1)
- (we could have defined v1 < v2 because 4 < 5, but in the above way, servers with a higher id have precedence)