1. Through k1 we know that a->b holds for all events local to that process which means we only need to consider interactions between processes. K2 tells us any event sent from p1 to p2 will have a have a higher C(x) at the receiver. If we have a scenario a->b at process p1 and we want to send both of these to p2. When we send a to p2 we know that C(a)p2>C(a)p1, and when we send b to p2 we know C(b)p1 > C(b)p2. We also know that a would be sent before b which means on the receiving side C(a)p2< C(b)p2. Since we know this holds between p1 and p2 we can expand this to p1 to pn.

4. One way for the clients to arrange them self’s in to a ring would be to organize themselves as they join. When a new client joins they will send out a join message that everyone will see, then some client will reply with their current id, since all clients will see this message. If any of them have a higher id they will also reply to the message once the highest current id is found the joining client will assign themselves that id plus one. The previous highest will make this new client its successor and the joining client will make the oldest client its successor.
6. The reason that the number of messages are unbounded is because the token can just go around the ring endlessly if no one wants the resource. To fix this we can create an on demand system where to whoever wants the resource will send out a request message and only then will the token be released. This mean that n messages will have to be sent every time.