Answer all questions. The questions are worth 50 points each, and you earn 50 points for stapling your answers together and writing your name on each page, for a total of 300 points. You have three hours, which should be more than enough.

Please do not waste any words: answers are graded on content, not style. Any kind of English or pseudocode is fine as long as its meaning is clear. Explain any assumptions that are not obvious. Feel free to include any relevant knowledge that you have, but if you think carefully before answering you may come up with a shorter and/or better answer.

1. **Say cheese.** It is often useful to take a consistent snapshot of a distributed/parallel system. Consider a system in which the state of each node is given by the sequence of events applied to that node, as in the classical state machine model. Then a global state of the system is described by a set of event sequences, with one sequence for each node in the system. Such a global state is **consistent** if and only if the following condition is true: for any event that is present in the set, all events that happened-before it are also present in the set. Here *happened-before* is the Lamport relation for potential causality.

Suppose the nodes in the system communicate only by sending and receiving messages, and that they maintain vector clocks in the usual fashion. Express the consistency property as a predicate on the vector clocks. In other words: sketch a function that takes the set of vector clocks for a global state as an input, and returns a boolean value that is true if and only if the global state is consistent.

2. **Off to the races.** Consider a shared block storage service (e.g., like FAB or Petal) in which access to blocks is protected by locks. Let’s call it Blockss. The locks for Blockss are served by a separate lock service (e.g., like Chubby), except that the clients may exchange lock/block ownership directly by passing messages (e.g., like Treadmarks). The designers of the Blockss server would like to enforce the locking protocol by aggregating or rejecting any block read or write requests that appear to violate proper mutual exclusion.

Suppose the nodes in the system communicate only by sending and receiving messages, and that they maintain vector clocks in the usual fashion. Express the mutual exclusion property as a predicate on the vector clocks. In other words: sketch a function that takes as input a set of block read and write requests (with the caller’s vector clock at the time of each request) and returns a boolean value that is true if and only if the server determines that mutual exclusion was violated.

3. **Out of time.** This question is about leases, as used in leased locks (e.g., for NQ-NFS or Frangipani) or session leases (e.g., for Google tools and the Chubby cell master). Leases allow the lease holder to take over specific system functions while protecting system consistency and availability in the event of failures and network partitions. Please discuss the following aspects of leases, illustrating with examples from the relevant systems we have discussed.

   - Discuss the actions that a system or application might take to preserve consistency on lease expiration.
   - Discuss the actions that a system or application might take to preserve availability on lease expiration.
   - Leases involve various time-related parameters. Summarize the time-related parameters and the issues involved in setting them.
• The “CAP theorem” claims that no system or mechanism can preserve both consistency and availability in the presence of partitions. Where is the flaw, i.e., what are the limitations of these lease-based systems that prevent them from running afoul of CAP?

4. **By any other name.** An infamous 1993 paper by Cheriton and Skeen disparages process group structures (CATOCS, also known as atomic multicast and virtual synchrony), arguing that various alternatives (“state-based solutions”) are more efficient and easier to use. This question is not asking you to address the merits of the specific Cheriton/Skeen arguments. But since that paper, Google has deployed robust massive-scale distributed systems whose core concepts appear quite different from CATOCS. In your opinion, why don’t they use CATOCS? Or do they? Or are Google’s needs different from those addressed by CATOCS? Or does Google meet the same needs in a different way? How and why?

5. **Still making progress.** Pat Helland of Amazon says that system builders “increasingly need to think about scaling without distributed transactions...because attempts to use distributed transactions are too fragile and perform poorly.” On the other hand, Mark McKeown says: “Any argument that distributed transactions should not be used because 2PC is blocking is a void, because Paxos Commit addresses the blocking issue.” Discuss.