Q1: Record and Replay) In our discussion of ReVirt we identified a number of system-level features (or, use-cases) where deterministic record and replay is useful. Please describe one such system-level feature and how it is enabled by record and replay. Be sure to include a citation(s) of prior work that provides similar features. Hint: you don't have to create a novel use case, and the notes from ReVirt include many examples and some citations.

Revirt used the virtualization approach to record and replay attacks. Replays for single-processor are well understood. Non-deterministic events are meant to be logged. It can be done easily on single-processor but much more challenging on **multi-processor systems due to memory races**. However, detection and replay of **memory races** were not discussed. The SMP-Revirt[1] paper explores that idea. The SMP-Revirt systems record the results of memory races using the hardware page protection. The record and replay system is implemented on a modified Xen hypervisor that logs and uses the CREW protocol. According to CREW protocol, each shared object is a concurrent read or exclusive write. For reconstructing shared memory, each write is considered asynchronous in shared memory. Ordering matters only if two instruction access share memory and one is write. Preserving order reproduces execution. Replaying DMA is another issue. To solve this, it follows a transaction model where the device is a non-preemptible actor in the CREW protocol.

Record and replay help to **debug code.** Therefore if there is a lower overhead cost, it can help debug hard-to-reproduce test failure. Modifying the kernel is difficult and recording it in a virtual machine is costly. The rr project[2] is a C/C++ debugging tool for Linux and gives lower overhead for single-threaded workloads. It can also work on multiple-process workload as well[3].

[1] George W. Dunlap, Dominic G. Lucchetti, Michael A. Fetterman, and Peter M. Chen. Execution replay of multi-processor virtual machines. Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '08, page 121–130, New York, NY, USA, 2008. Association for Computing Machinery.
[2] Robert O'Callahan, Chris Jones, Nathan Froyd, Kyle Huey, Albert Noll, and Nimrod Partush. Engineering record and replay for deployability. Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference, USENIX ATC '17, page 377–389, USA, 2017. USENIX Association.
[3] https://rr-project.org/

(Q2: Reliability) Failure Oblivious Computing describes a systematic way to survive memory errors in server applications. Choose a different type of bug and a solution for how you might survive it in some/all cases? If you build/use an idea from prior work, please cite it!

Most of the server applications are designed to handle common errors. However, uncommon software errors go unhandled by the servers[1]. Even rollbacks fail for deterministic errors. Deterministic errors will cause problems in multiprocess models. If the shared space is corrupt,

rolling back will not restore the consistent state, and other processes will also fail that are dependent on it. In the paper, "treating bugs as allergies- a safe method to survive software failures," the bug is tackled by rolling back to the checkpoint and changing the execution environment for the given bug(e.g., 56% of faults in Apache HTTP server depend on execution environment). **Data Races** are avoided by trying different timing conditions. E.g., Increasing the length of a scheduling time slot avoids context switch in buggy critical sections. It also drops **unexpected user requests** if all the **possible execution environments fail in such a scenario**. Hence it probably survives all possible scenarios. It can help in memory management as well.

The paper[2], "Safe software updates via multi-version execution." also explores a similar approach to change the application version instead of changing the execution environment. It might work if the other version has some stable fit but otherwise won't work.

[1]F. Qin, J. Tucek, J. Sundaresan, and Y. Zhou, "Rx: treating bugs as allergies – a safe method to survive software failures," in ACM SIGOPS Operating Systems Review, vol. 39, no. 5, 2005, pp. 235–248.
[2] P. Hosek and C. Cadar, "Safe software updates via multi-version execution," in Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013, pp. 612–621.