Paper title: FastTrack: Efficient and Precise Dynamic Race Detection
Paper authors: Cormac Flanagan, Stephen Freund
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
This paper addresses the issue of race conditions in multithreaded programs. It considers the speed and precision tradeoff of various race detectors. Precise race detectors are generally slow - they use vector clocks to represent the happens-before relation. These vector clocks however record information about every thread in the system, and are thus expensive. For n threads, it takes O(n) space and O(n) time. In turn, several race detectors have tried to eliminate the usage of vector clocks to improve performance but this raises false alarms and compromises precision.

DJIT$^+$ is a high-performance race detector that uses vector clocks to represent the happens-before relation. **BasicVC** is a traditional VC-based race detector using a vector clock for every read and write operation on each memory location. Both these race detectors use vector clocks and provide precise detections, but are prone to performance overheads.

In contrast, there are several works exploring faster but imprecise race detectors. **Eraser** uses a LockSet algorithm which follows a lock-based synchronization principle. They may, however, report incorrect cases when the synchronization scheme varies. **Goldilocks** is a precise race detector that uses a modified version of the above LockSet algorithm, but is a complex program. **MutiRace** combines LockSet algorithm with happens-before reasoning to improve precision.

Each of these race detectors are lacking either in precision or in performance. Building on the best aspects of precise race detectors, the paper develops a race detector that is precise and enhances performance by eliminating the general use of vector clocks for all cases.

What are the key contributions of the paper?
The paper demonstrates a new algorithm called FastTrack that has improved precision along with a good performance. The authors analyzed the different scenarios causing race conditions and realized the possibility of eliminating low-performing vector clocks in the more frequently occurring scenarios. Instead, an adaptive representation is suggested for the happens-before relation, which has lesser overheads. This efficient algorithm is also simple and easy to implement. FastTrack also enhances the performance of the several dynamic analysis tools by recognizing the numerous race-free accesses.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
The authors developed a prototype implementation of FastTrack that runs on RoadRunner, a framework for dynamic analyses on multithreaded software. For comparison, several other race detectors were also tuned to run on RoadRunner - these include Eraser, DJIT$^+$, MultiRace, Goldilocks and BasicVC.

FastTrack uses the following principles for faster and more precise detections. Instead of recording the clock of the last write operation to each variable by each thread, the FastTrack algorithm observes that every write on a variable occurs according to the happens-before relation and so it only records the last write on any variable and tracks the clock as well the thread. This pair is known as an epoch, and it requires only constant space as compared to the O(n) space requirement of vector clocks for n threads. In case of read operations, if they occur on thread-local and lock-protected data, the operations can be considered as ordered. So FastTrack records only the epoch of the last read on such data. The algorithm is also able to switch to vector clocks adaptively in certain cases to guarantee precision. This may happen when data becomes read-shared. It can also switch back to epochs when the condition changes.

The above principles have allowed FastTrack to perform better than the other 5 implementations that were compared. FastTrack is slightly faster than **Eraser**, while maintaining more precision than its counterpart. When compared to **DJIT$^+$** and **BasicVC**, FastTrack has equal precision but is 10x faster than BasicVC and 2.3x faster than DJIT$^+$ - a difference caused mainly due to lesser allocation of vector clocks by FastTrack. **MultiRace** performed similarly to DJIT$^+$ but had a memory footprint larger than DJIT$^+$ and had a small percentage of operations that required an Eraser operation which introduced additional overhead. Lastly, **Goldilocks** proved to be a complicated algorithm with issues in JVM integration and did not offer any significant improvement in comparison to FastTrack.

Write down one question you may want to ask the authors.
What kind of updates would be required for FastTrack to be able to efficiently detect race conditions across distributed systems?