Oral Exam Script: RacerD

Based on Blackshear et al. (OOPSLA 2018)

Minute 1: The Problem (Context)

- **Industrial Context:** Facebook (Meta) moved their Android News Feed from sequential to multithreaded for performance.
- **The Challenge:** They needed to check millions of lines of code.
  - *Whole-program analysis* was too slow (hours).
  - *Annotation-heavy systems* (like 'rccjava') required too much manual work for developers.
- **Goal:** A tool that is fast enough to run during Code Review (CI/CD), scalable, and has a high "Fix Rate" (low false positives), even if it is technically unsound.

Minute 2-3: The Solution (Technical Insight)

- **Compositionality:** This is the key technical innovation. RacerD analyzes each method **independently** without knowing the whole call graph.
- **Summaries:** It produces a summary for each method containing "Access Snapshots".
- **Access Snapshot Components:**
  1. **Prefix:** The path to the object being accessed (e.g., 'this.field').
  2. **Lock Set:** Which locks are held during access?
  3. **Thread Status:** Is this running on the Main Thread or a Background Thread?
- **Syntactic Matching:** Instead of expensive alias analysis, it checks for races on syntactically identical paths. If 'A.f' is written without a lock in one thread and read in another, it flags it.

Minute 4: Evaluation

- **Deployment:** Deployed in production at Facebook.
- **Results:**
  - Detected over 2,500 concurrency bugs that were actually fixed by developers.
  - **Performance:** It analyzes code changes ("diffs") in minutes (median 12 mins), allowing it to comment directly on Pull Requests.
- **Comparison:** Compared to academic tools like CHORD, RacerD was orders of magnitude faster.

Minute 5: Critique & Discussion

- **Strengths:**
  - **Scalability:** Compositional analysis allows it to scale to millions of LOC.
  - **Workflow Integration:** By accepting "unsoundness" (missing some bugs), they achieved a tool developers actually use.
- **Weaknesses:**
  - **Unsoundness:** It explicitly misses races involving complex aliasing (if ‘x‘ and ‘y‘ point to the same object but have different names, RacerD might miss the race).
  - **Coarse Granularity:** It tracks lock counts rather than specific lock instances in some cases to save memory.
  - **Annotations:** Still requires ‘@ThreadSafe‘ or ‘@MainThread‘ annotations to define the threading model.