How to debug your parallel / concurrent program using Concurrent Breakpoints

**Concurrent Breakpoints**

**Definition**
- Concurrent Breakpoints defined as a triple \((L_1, L_2, \varphi)\)
- Suspend program when \(PC_{Thread1}==L_1, PC_{Thread2}==L_2\), and \(\varphi\) holds

**Unlikely (hard to determine) that two threads are at specific locations simultaneously**
- \(\varphi\) can be partitioned as local predicates \(\varphi_1, \varphi_2\) and global predicate \(\varphi_{12}\)
- When a thread at \(L_i\) satisfies local predicate \(\varphi_i\), wait for another thread to satisfy remaining conditions

**Resume by starting with Thread1 or Thread2**
- Specific order can lead to (or prevent) bugs

**Uses of Concurrent Breakpoints**
- Express common concurrency bugs such as races, deadlocks, atomicity violations, etc.
- Concise and programmatic bug report
- Cheap way to explore different schedules and reproduce bugs
- Gather information from various executions
- Enables cyclic debugging

**Evaluation**

- Concurrent Breakpoints for 15 programs
- Reproduced 32 different concurrency bugs
- Low overhead (<40%), high reproducibility (~99%)
- Longer wait time for higher trigger rate
- Refine local predicates for lower overhead
- Bound number of breakpoint triggers
- Utilize more contextual information

**Case study for cyclic debugging**
- Used Concurrent Breakpoints and cyclic debugging methodology to reproduce the missed notification Heisenbug in log4j

**Future Work**
- More case studies of using Concurrent Breakpoints for cyclic debugging of Heisenbugs
- Using Concurrent Breakpoints to specify constrained thread schedules