Path Projection for User-Centered Static Analysis Tools

Jeff Foster
University of Maryland, College Park

Joint work with YitPhangKhoo and Mike Hicks

(Not yet published, so don’t steal these ideas!)

Introduction

- Many recent successes in static analysis tools for defect detection / prevention
  - Lots of progress in the research community
  - Coverity, Fortify, and others sell static analysis tools
  - Microsoft has had great success with static analysis

- Major research focus: Building tools that are...
  - “Precise enough”

Motivation

- Static analysis tools are not perfect
  - Users must *triage* bug reports
    - Decide whether true or false positives, and how important
  - Users must *remediate* true bugs

- Conclusion: successful static analysis requires cooperation between the user and the tool

- How do we build more user-centered static analyses?

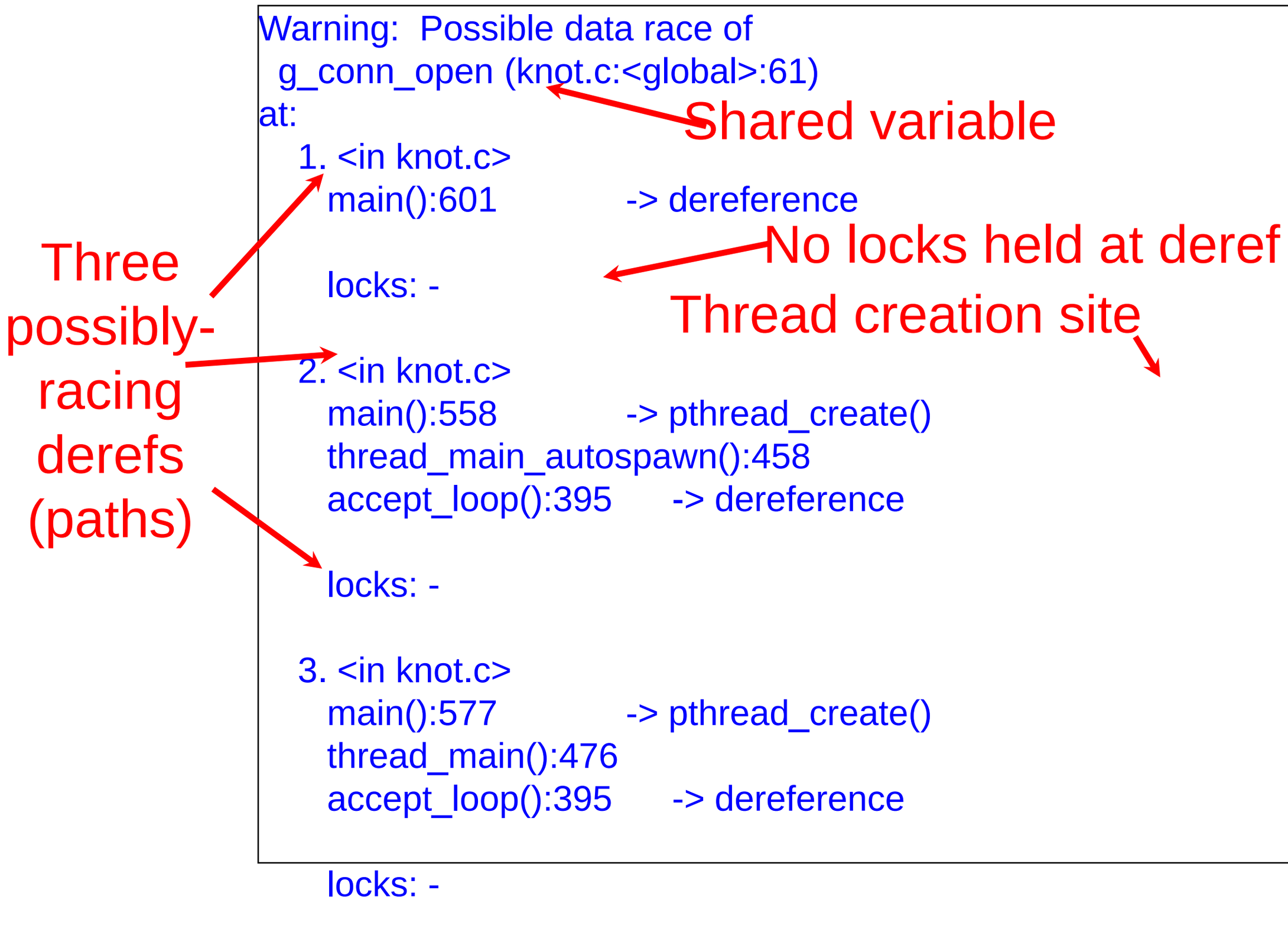
Path Projection

- A new interface to help users visualize code paths
  - E.g., call stacks, control flow paths, data flow paths
- Core principles
  - Remain true to original source code
  - Fit as much on one screen at a time as possible

Contributions

- Prototype implementation in WebKit
- Controlled user study
  - Task: Triaging Locksmith error reports
  - Compared to “standard viewer” (similar to IDEs)
- Experimental results
  - Improved performance (completion time)
  - Same accuracy
  - Qualitatively better

Sample Locksmith Error Report



Triaging a Locksmith Report

- Three things to check:
  1. Both accesses refer to same location
     - Locksmith's alias analysis may be imprecise
  2. Locksmith has not missed a held lock
     - Could happen at join points in cfg
  3. Potentially-racing accesses can occur, simultaneously
     - Both stack traces given must be *simultaneously realizable*

- This work: Focus on task 3
  - Leave 1 and 2 as future work

Path Visualizer

knot.c split close

Splittable file viewer

Basic searching

Hyperlinked error report

Triaging checklist

```
589         start = current_usecs();
590         sleep(g_timer_interval);
591
592         while (1)
593         {
594             long long bytes_sent;
595             int conn_open, conn_succeed, conn_fail, conn_active, cach
596
597             now = current_usecs();
598
599             ;
600             bytes_sent    = g_bytes_sent;       g_bytes_sent = 0
601             conn_open     = g_conn_open;        g_conn_open = 0
602             conn_succeed = g_conn_succeed;     g_conn_succeed
603             conn_fail     = g_conn_fail;       g_conn_fail = 0;
604             conn_active   = g_conn_active;
605             cache_hits    = g_cache_hits;      g_cache_hits = 0;
606             cache_misses = g_cache_misses;     g_cache_misses = 0;
607             ;
608
609
610             //printf("rate: %.3g Mbits/sec   %.0f open/sec   %.0f suc
611             printf("rate: %.3g Mbits/sec   %.0f open/sec   %.0f succ/
612                    ((double)bytes_sent * 8 * 1000000) / ((double)(now
613                    ((double)conn_open * 1000000) / (double)(now-start
614                    ((double)conn_succeed * 1000000) / (double)(now-st
615                    ((double)conn_fail * 1000000) / (double)(now-start
616                    conn_active,
617                    cache_misses,
618                    100*((double)cache_hits)/(double)(cache_hits+cache
619                    );
620                 printf("\n");;
621
622
623                 start = now;
624
625                 sleep(g_timer_interval);
626             }
627         }
628
629         pthread_exit(0);
630
631         return 0;
632
633         //close(s);
634         //exit(0);
635     }
636
637
638     ////////////////////////////////////////////////////
639     // Set the emacs indentation offset
640     // Local Variables: ***
```

Find

Find Cancel

Help

Path Report

```
Warning: Possible data race of
    g_conn_open (knot.c:<global>:61)
at:
    1.  <in knot.c>
        main():601                      -> dereference

        locks: -


    2.  <in knot.c>
        main():558                      -> pthread_create()
        thread_main_autospawn():458
        accept_loop():395               -> dereference

        locks: -


    3.  <in knot.c>
        main():577                      -> pthread_create()
        thread_main():476
        accept_loop():395               -> dereference

        locks: -
```

Locksmith Checklist

Locks | Path 1 | Path 2 | Path 3 | 1-2 | 1-3 | 2-3 | Submit

Please verify that the locks reported at the end of each path in the path report, if any, are correctly acquired in the source code (list the *function:line*, or *none*):

Path 1 : ????

Path 2 : ????

Path 3 : ????

Standard Interface Demo

Challenges with Standard Interface

- Requires lots of scrolling through code
  - Hyperlinks help a little
  - Still hard to keep track of the *context* along the path
- Need to compare multiple paths together
  - Are both realizable?
- Need to visually switch between error report and program source code
  - Adds cognitive burden

Path Visualizer

Path 1 of 3

```
)

(&thread, NULL, &thread_main_autospawn, (void*) s) < (

d_create");


hreads; i++)

eate(&thread, NULL, &thread_main, (void *) targs) < 0)

hread_create");


 g_conn_open;       g_conn_open = 0;
```

Path 2 of 3

```
482 int
483 main(int argc, char **argv)
484 {
554     if (g_spawn_on_demand)
555     {
558         if (pthread_create(&thread, NULL, &thread_main_autospawn, (void*)
```

knot.c - main()

knot.c - thread_main_autospawn()

```
453 void *
454 thread_main_autospawn(void *arg)
455 {
458     accept_loop(-1, s);
```

knot.c - accept_loop()

```
371 void
372 accept_loop(int id, int s)
373 {
376     while (1)
377     {
395         g_conn_open++;
415         if( g_spawn_on_demand ) {
425             if (pthread_create(&thread, &attr, &thread_process_client
426                 //perror("pthread_create");
432             }
433         } else {
442     }
443 }
461 }
559         {
560             perror("pthread_create");
561         }
562     }
563     else
564     {
566         for (i = 0; i < nthreads; i++)
567         {
577             if (pthread_create(&thread, NULL, &thread_main, (void *) targs
578             {
579                 perror("pthread_create");
580             }
581         }
583     }
635 }
```

Error report

Multi-query ▶Help

Find

- ✘ "pthread_create"
- ✘ "pthread_join"
- ✘ "pthread_mutex_lock"
- ✘ "pthread_mutex_unlock"

Path Report

```
Warning: Possible data race of
    g_conn_open (knot.c:<global>:61)
at:
    1.  <in knot.c>
        main():601                  -> dereference

        locks: -


    2.  <in knot.c>
        main():558                  -> pthread_create()
        thread_main_autospawn():458
        accept_loop():395           -> dereference

        locks: -


    3.  <in knot.c>
        main():577                  -> pthread_create()
        thread_main():476
        accept_loop():395           -> dereference

        locks: -
```

Locksmith Checklist

Locks | Path 1 | Path 2 | Path 3 | 1-2 | 1-3 | 2-3 | Submit

Please verify that the locks reported at the end of each path in the path report, if any, are correctly acquired in the source code (list the *function:line*, or *none*):

Path 1 : ????

Path 2 : ????

Path 3 : ????

Side-by-side paths

Checklist

Function call inlining

Code folding (non-consecutive line numbers)

```
482  int
483  main(int argc, char **argv)
484  {
554      if (g_spawn_on_demand)
555      {
558          if (pthread_create(&thread, NULL
```

knot.c - thread_main_autospawn()

```
453  void *
454  thread_main_autospawn(void *arg)
455  {
458      accept_loop(-1, s);
```

knot.c - accept_loop()

```
371  void
372  accept_loop(int id, int s)
373  {
376      while (1)
377      {
395          g_conn_open++;
415          if( g_spawn_on_demand ) {
425              if (pthread_create(&thr
426                  //perror("pthread_c
432              }
433          } else {
442      }
443  }
461  }
559          {
560              perror("pthread_create");
561          }
```

Path Visualizer

Path 2 of 3

```
*argv)

nand)

reate(&thread, NULL, &thread_main_autospawn, (void*)
spawn()

wn(void *arg)

  s);

id, int s)


pen++;
awn_on_demand ) {
pthread_create(&thread, &attr, &thread_process_client
//perror("pthread_create");
```

Multiple simultaneous searches

Multi-query

- ✘ "pthread_create"
- ✘ "pthread_join"
- ✘ "pthread_mutex_lock"
- ✘ "pthread_mutex_unlo

Path Report

```
Warning: Possible data
    g_conn_open (knot.
at:
    1.  <in knot.c>
        main():601

        locks: -


    2.  <in knot.c>
        main():558
        thread_main_au
        accept_loop()
```

Path Projection Demo

Informational Visualization Strategies

- *Increase user's memory and processing resources, and reduce the search for information*
  - Make important lines of code visible on screen
  - Place related lines of code close together

- *Use visual representation to enhance pattern matching*
  - Put function definition in colored boxes
  - Format/color code to reveal program structure

- *Encode information in a manipulable medium*
  - Allow users to search/highlight

User Study: Overview

- Standard Viewer (SV) vs. Path Projection (PP)
- Task: Triage a Locksmith error report
  - Decide whether it is a false positive or not
- Measurements
  - Completion time for the task
  - Qualitative feedback from users
  - Observations of user behavior

Locksmith Task Details

- All trials from Locksmith benchmarks
  - E.g., web server, ftp client, etc.
  - Roughly 1,500 lines each
  - Unfamiliar to participants
- One warning per trial
  - No need to manage warnings

Locksmith Task Details (cont'd)

- No potential aliasing issues
  - All shared variables and locks are global
  - (Just a property of these particular warnings)

- Semantics-preserving simplifications:
  - Made local static variables global
  - Changed wait()/signal() to join()
  - Deleted #if 0 or other conditional macros
  - Converted some goto/switch statements to if

Within-subjects Design

- Two possible schedules for a participant
  - Same problems in same order

	Session 1	Session 2
Group 1	PP (1.1, 1.2, 1.3)	SV (2.1, 2.2, 2.3)
Group 2	SV (1.1, 1.2, 1.3)	PP (2.1, 2.2, 2.3)

- Pros:
  - Participants can directly compare both interfaces
  - Fewer problems due to individual variances
- Cons:
  - Order effect: may prefer first interface
  - Learning effect: may become better at task over time

Experimental Procedure

- Pre-questionnaire (background/demographic)
- Each session
  - Tutorials on data races, Locksmith, the interface
  - One practice trial
  - Three measured trials
    - First, complete task; measure time
    - Then, repeat same task and explain aloud
      - Helps users learn faster, and gives us some insights
      - We *do not* tell users whether their reasoning is correct

The Learning Effect

- Triaging a possible data race is hard!

- Participants
  - Did not have a lot of experience with this
  - Were unfamiliar with Locksmith (except one user)
  - Tended to get side-tracked during the task

- Two solutions
  - Extensive pre-experiment tutorials
  - A *checklist* to guide the user

```
        locks: -


    3.  <in knot.c>
        main():577                      -> pthread_create()
        thread_main():476
        accept_loop():395               -> dereference

        locks: -
```

Warm-up task

One tab per task component

Locksmith Checklist

Locks | Path 1 | Path 2 | Path 3 | 1-2 | 1-3 | Submit

2-3

Please verify that the locks reported at the end of each path in the p[illegible]port, if any, are correctly acquired in the source code (list the *function:lin[illegible]* or *none*):

Path 1 : ????

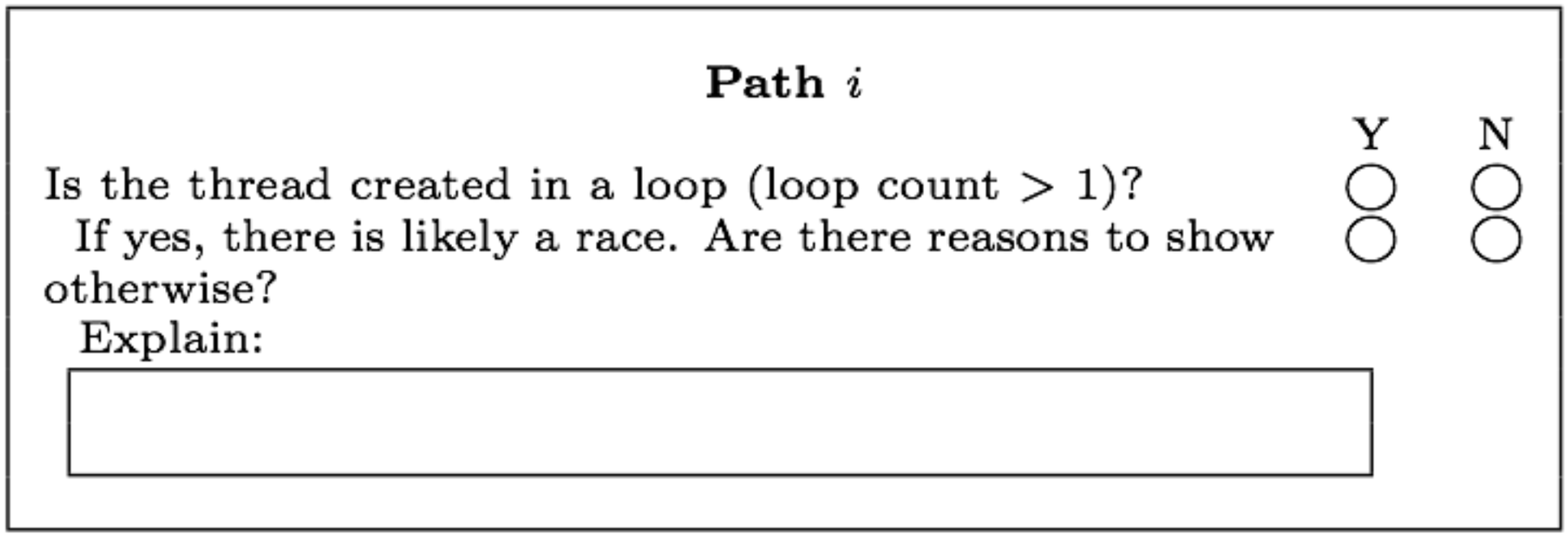
Path 2 : ????

Path 3 : ????

Submit completes checklist (and ends trial)

Anecdotally, checklist improves time by 41%

Tab for a Unprotected Access



- Deference with no locks held
  - May race with itself if called from multiple threads
    - while(1) { fork { *p++ } }
  - One special case to look for (only case in trials)
- Questions only enabled as appropriate

Tab for Two Accesses

For threads leading to dereferences in Paths i and j:

Are they parent-child (or child-parent), or child-child?

◯ Parent-child / ◯ Child-child

Parent-child (or child-parent) threads.	Y	N
Does the parent's dereference occur after the child is spawned?	◯	◯
Before its dereference, does the parent wait (via `pthread_join()`) for the child?	◯	◯
If no, there is likely a race. Are there reasons to show otherwise?	◯	◯

Explain:

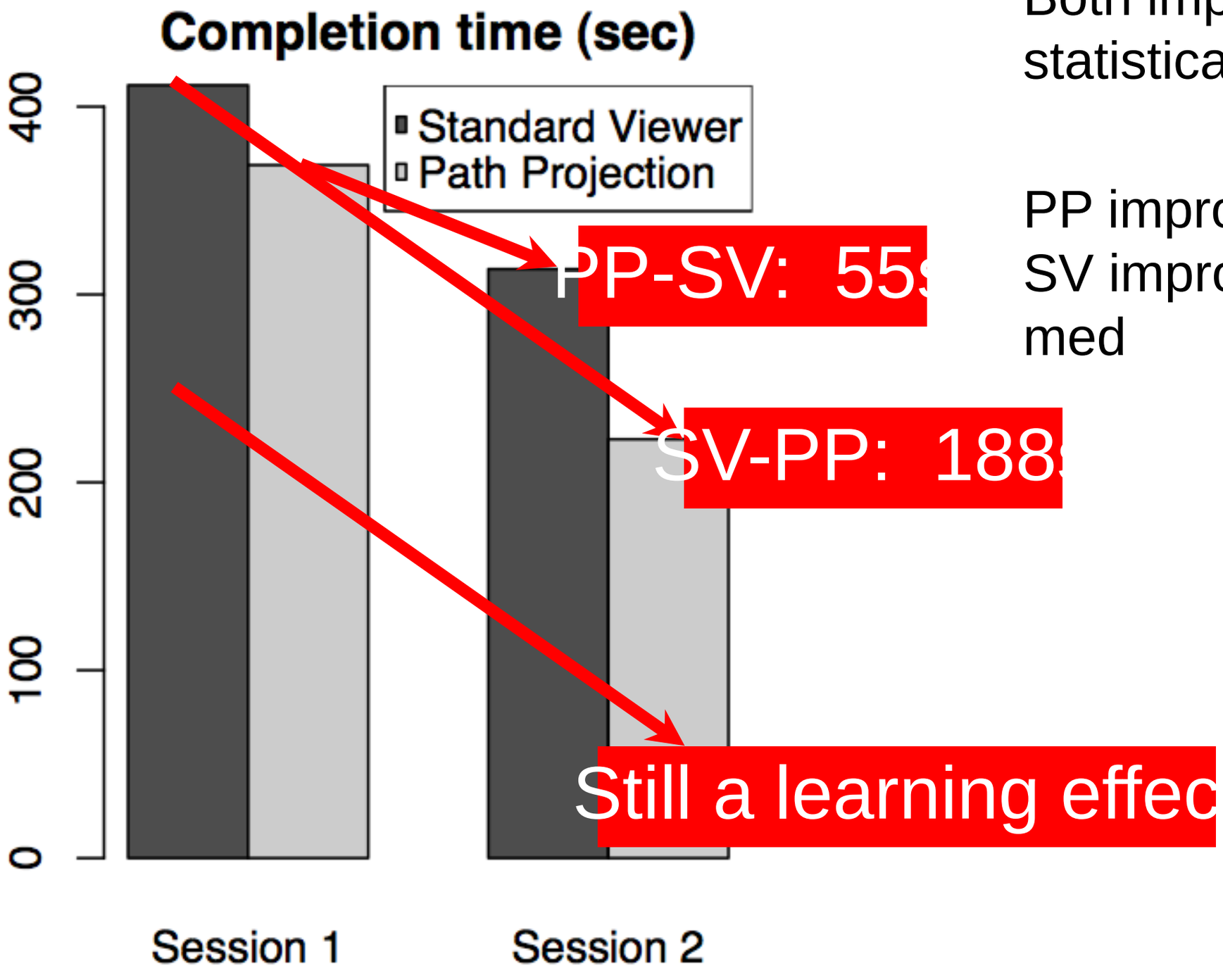
Child-child threads.	Y	N
Are the children mutually exclusive (i.e., only one can be spawned by their common parent/ancestor)?	◯	◯
If no, there is likely a race. Are there reasons to show otherwise?	◯	◯

Explain:

Participants and Equipment

- 8 student participants
  - 3 undergraduates, 5 graduates
  - Prior experience in C, multithreading (not necc. C)
  - Self-rated experience: 3 to 4
    - Scale of 1: no experience to 5: very experienced
  - 2 participants had experience in Locksmith and Eraser
- Apparatus
  - 24" 1920-by-1200 LCD
  - Mac OS X 10.5.2
    - All shortcuts disabled except for cut/copy/paste/find/find-next

Mean Time for All Participants



Both improvements statistically significant

PP improvement large
SV improvement small-med

Accuracy and Detailed Times

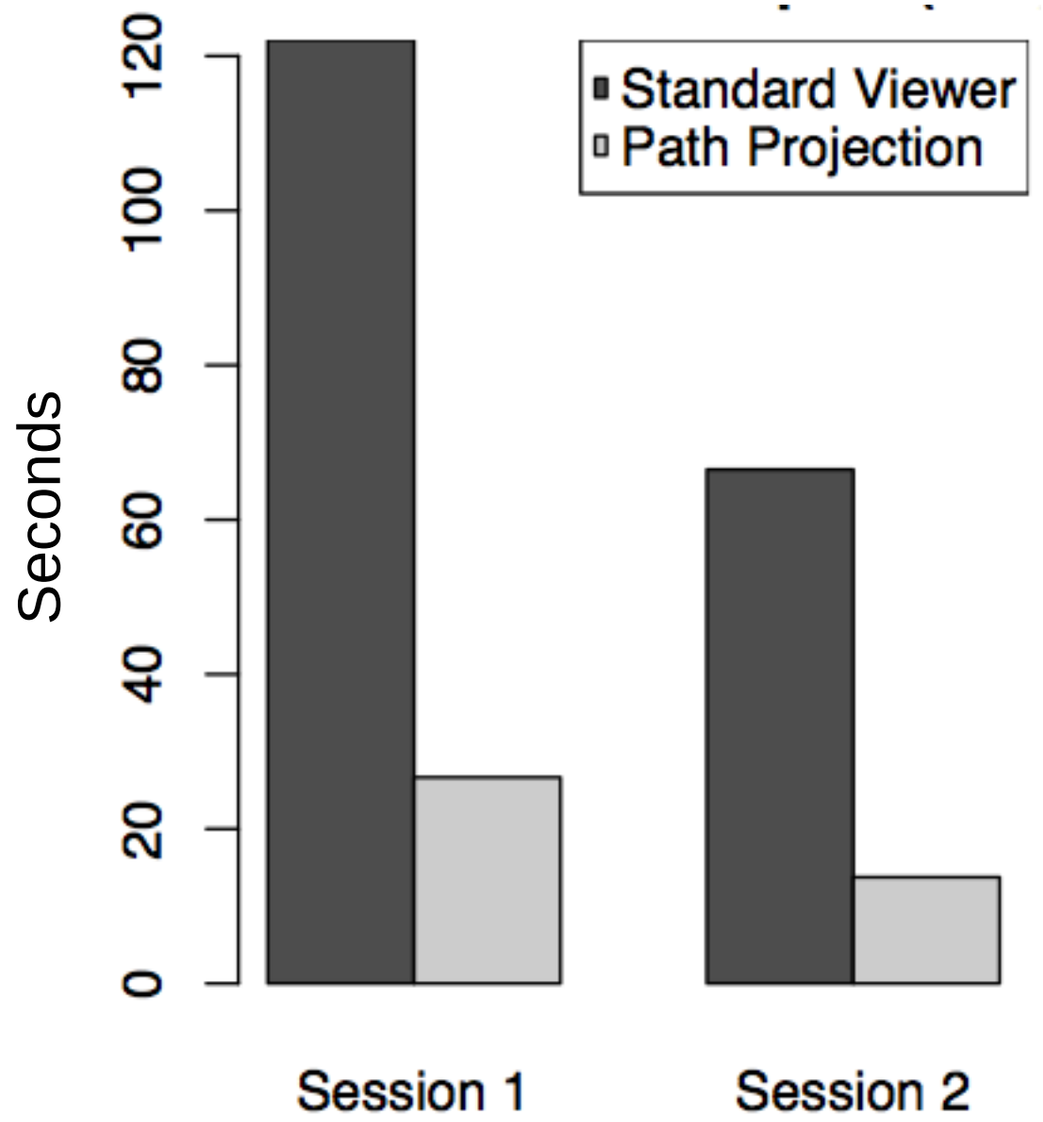
Completion times and accuracy for each trial

	Session 1				Session 2			
Trial	1.1	1.2	1.3	mean	2.1	2.2	2.3	mean
User	SV				PP			
1	8:36	14:14	9:44	10:51	7:07	4:48	4:02	5:19
2	5:07*	3:10	5:50	4:42	4:16	2:29*	2:10*	2:58
5	7:46	2:34	5:38	5:20	5:13	3:43	1:18	3:25
7	5:40	6:23	7:35	6:33	3:05	3:53	2:32	3:10
				6:51				**3:43**
User	PP				SV			
0	6:27	6:09	8:32	7:03	9:42	5:16	3:11	6:03
3	6:38	7:18	8:35	7:30	11:18	6:21	3:39	7:06
4	8:21	2:11	4:43	5:05	5:26*	4:27*	2:46*	4:13
6	7:11	2:52	4:50	4:57	4:33	4:06	1:58	3:32
				6:09				**5:14**
# Tabs	3	2	6		6	3	3	

* one incorrectly answered tab in the checklist

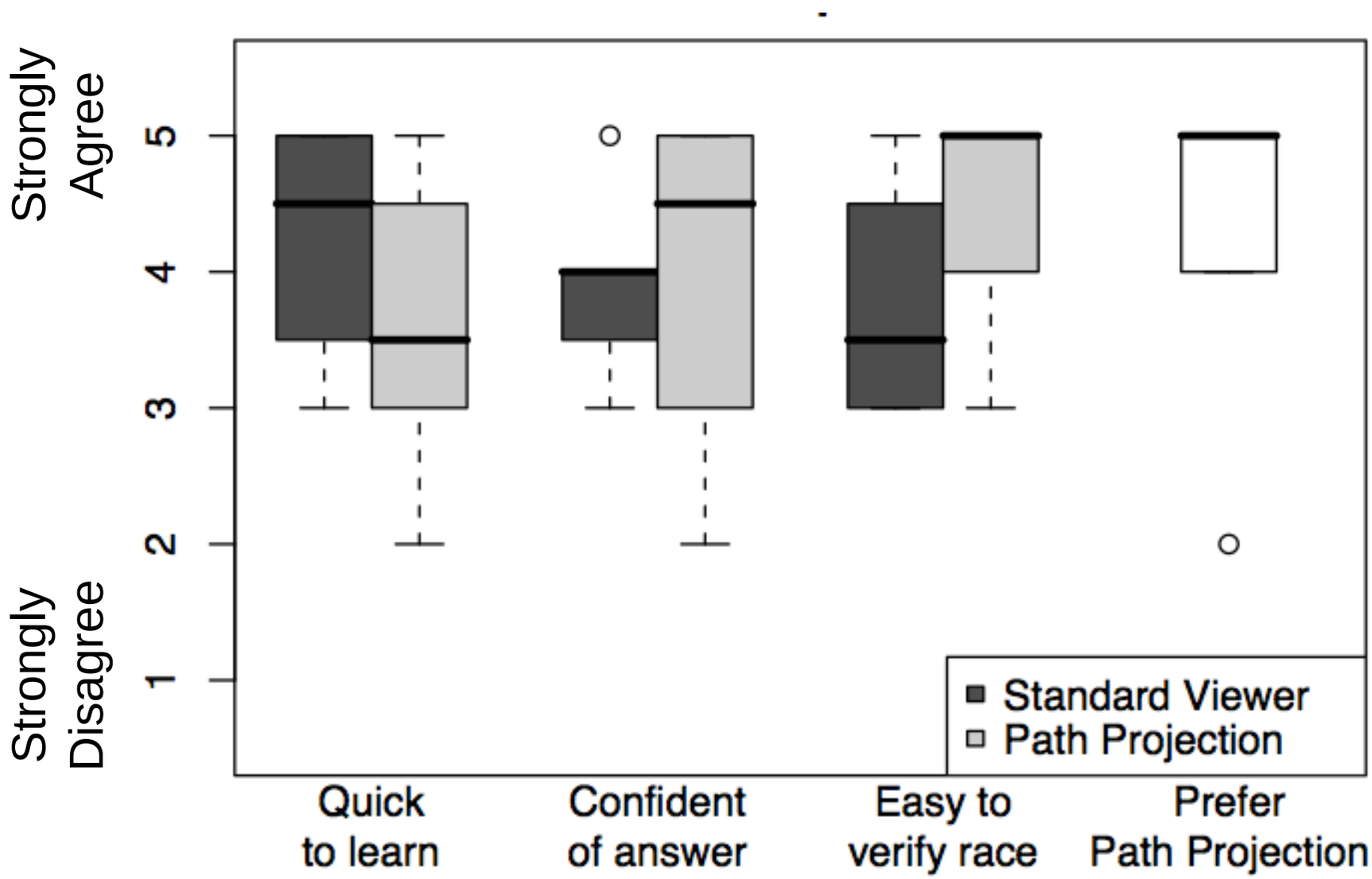
- Similar # of mistakes: 10 PP (10.9%), 9 SV (9.8%)
- Mistakes in 2.2, 2.3 due to common, unrealizable sub-path

Mouse Hover Time in Error Report



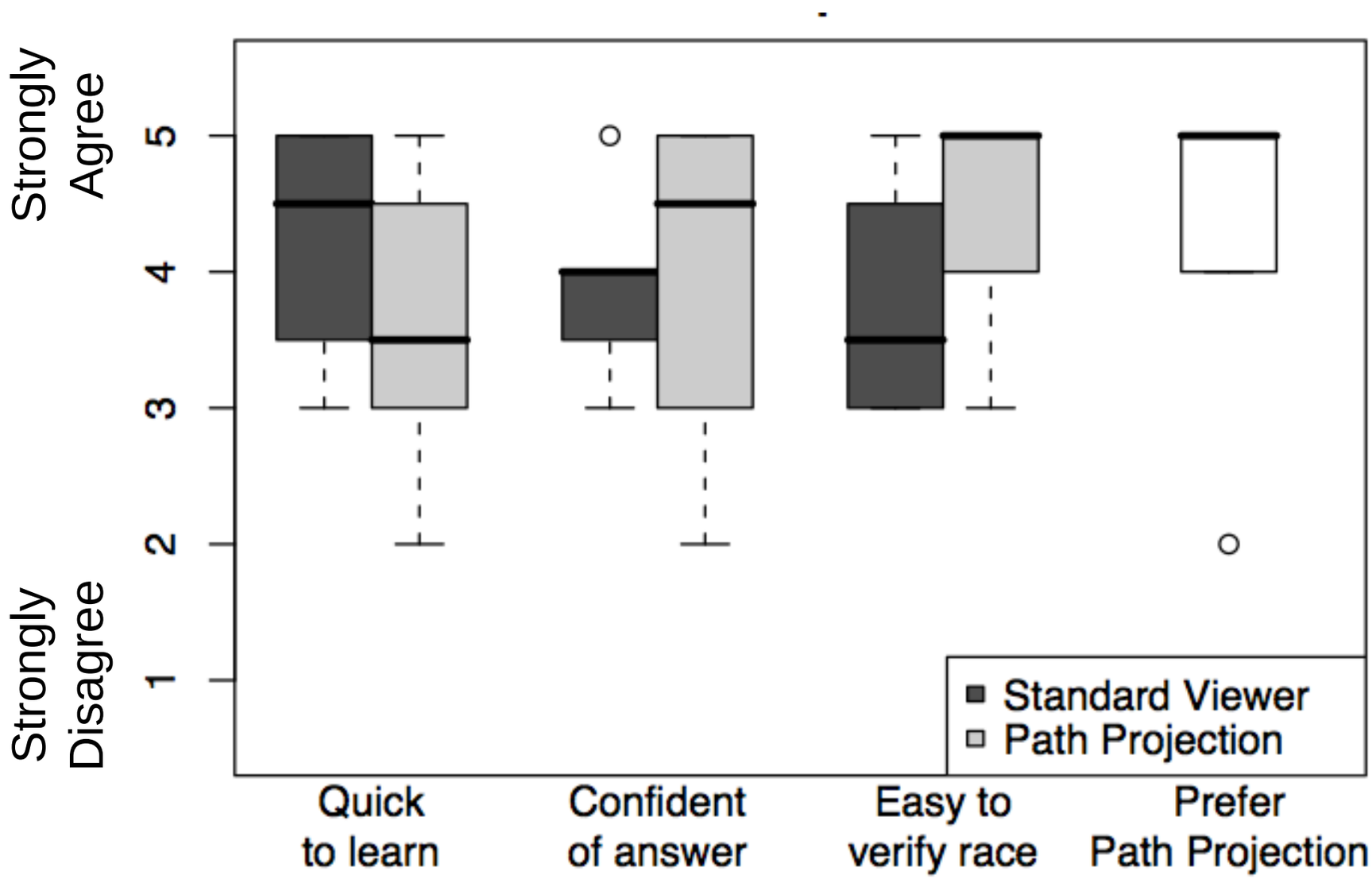
- 20s on average for PP, compared to 1:34 for SV
  - Little need to use hyperlinks under PP

Overall Impression (Qualitative)



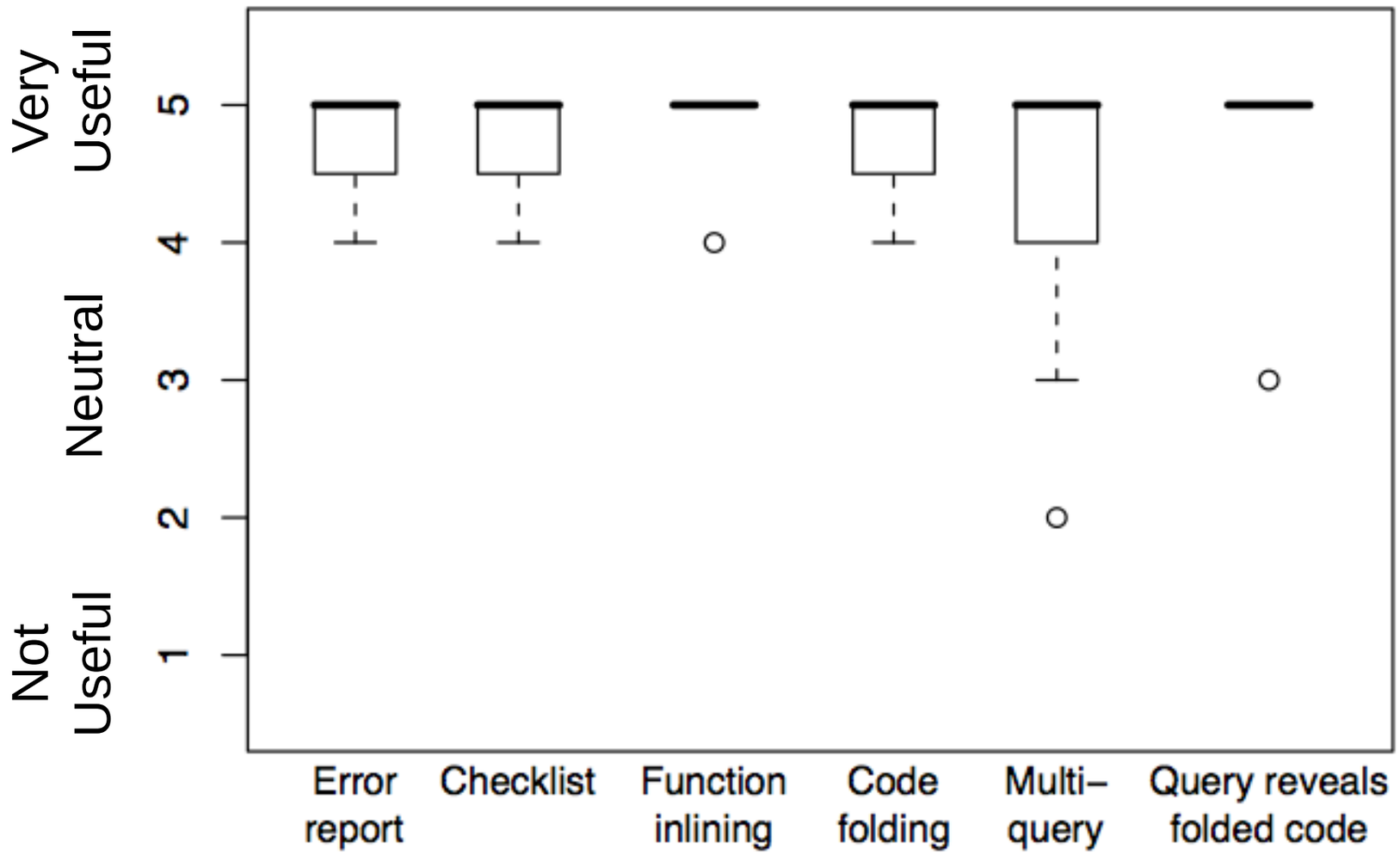
- Boxplot
  - Centerline = median Box extent = quartiles
  - Whiskers = min/max Dots = outliers

Overall Impression (Qualitative)



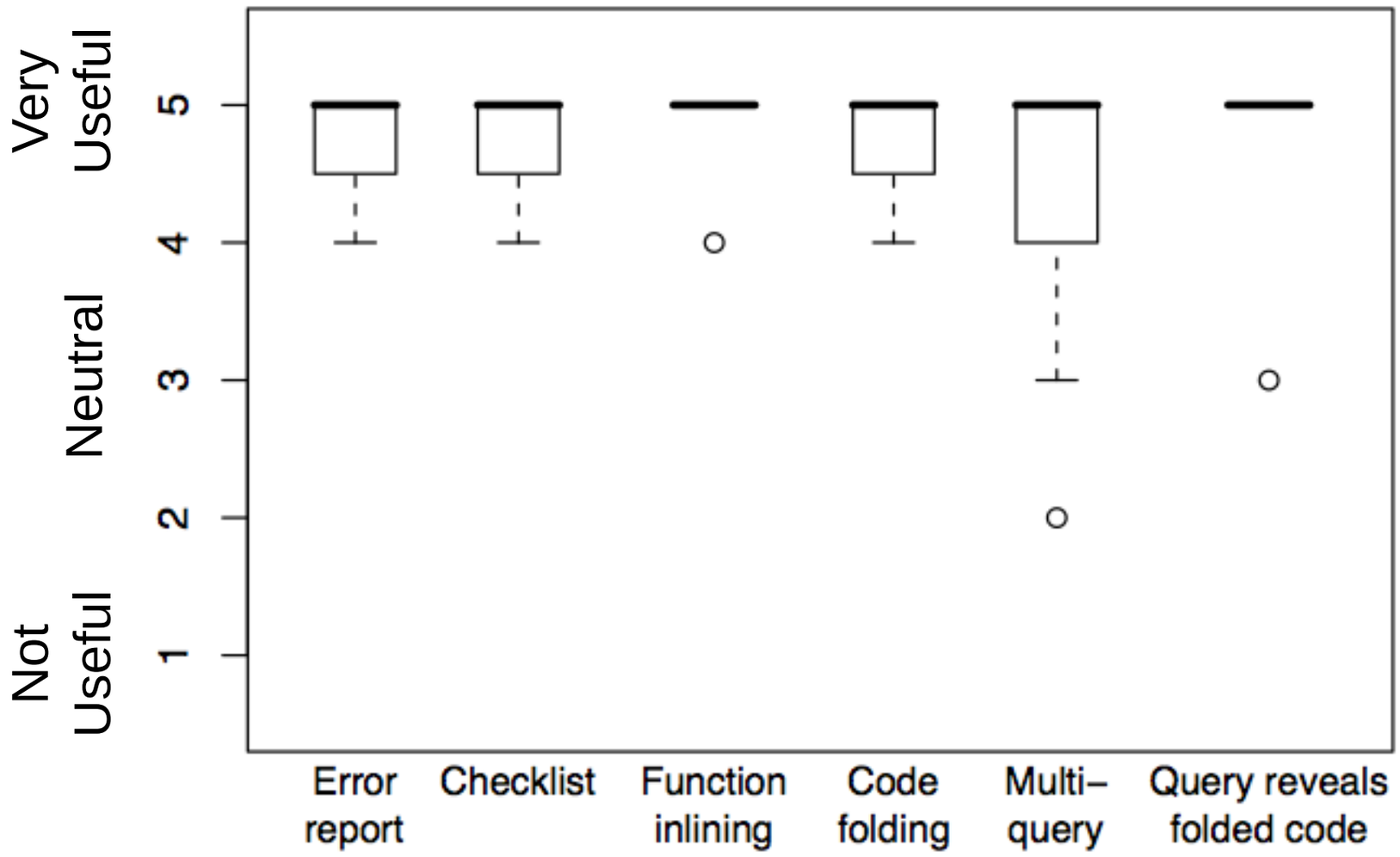
- No statistically significant differences in answers
  - Small sample? Limited exposure?
- All but one preferred PP

PP Feature Ratings (Qualitative)



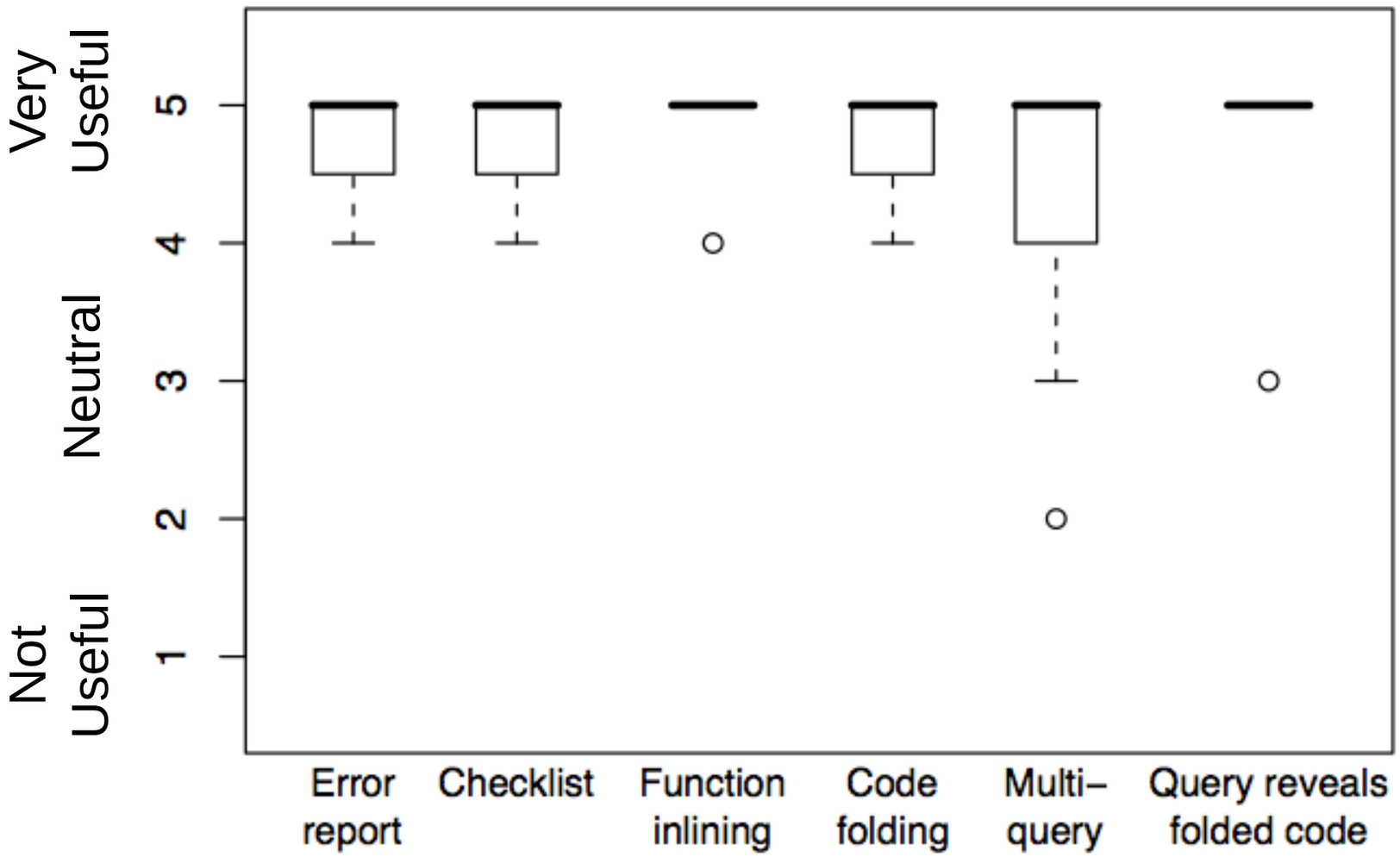
- All statistically significant vs. neutral response
- Generally favorable towards PP features

PP Feature Ratings (Qualitative)



- Surprisingly, liked code folding/function inlining
  - Code folding was “the best feature” or “my favorite feature”

PP Feature Ratings (Qualitative)



- Checklist: “saved me from having to memorize rules”
- Two participants did not favor multi-query
  - But forgot multi-query had 4 default items

Threats to Validity

- Results may not generalize
  - Small population, students, not data race experts
  - Small set of programs
  - Learning effect still present
- Changes to programs to make task easier
  - Task in experiments is very focused
  - Understanding error reports generally requires wider range of activities
- SV interface is not production quality
  - Deliberate choice, to avoid giving any advantages

Summary

- Introduced Path Projection, a new interface
  - Side-by-side display of paths
  - Function call inlining
  - Code folding
  - In general, tries to follow InfoViz principles
- Experimental results suggest PP
  - Improves completion times
  - Is liked by users
- Lots more to do on this topic!