LRZ Workshop – 09.11.2022

Dynamic Debugging with Intel® Inspector

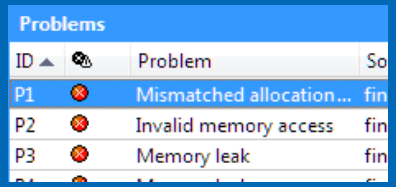
Heinrich Bockhorst

Agenda

- Introduction to Inspector
- GUI usage
- Command line usage
- Results
- Demo - Next steps

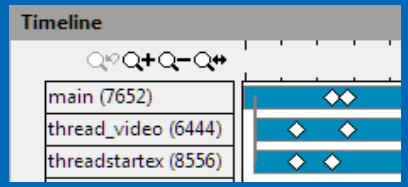
Motivation for Intel® Inspector

Memory Errors



- Invalid Accesses
- Memory Leaks
- Uninitialized Memory Accesses

Threading Errors



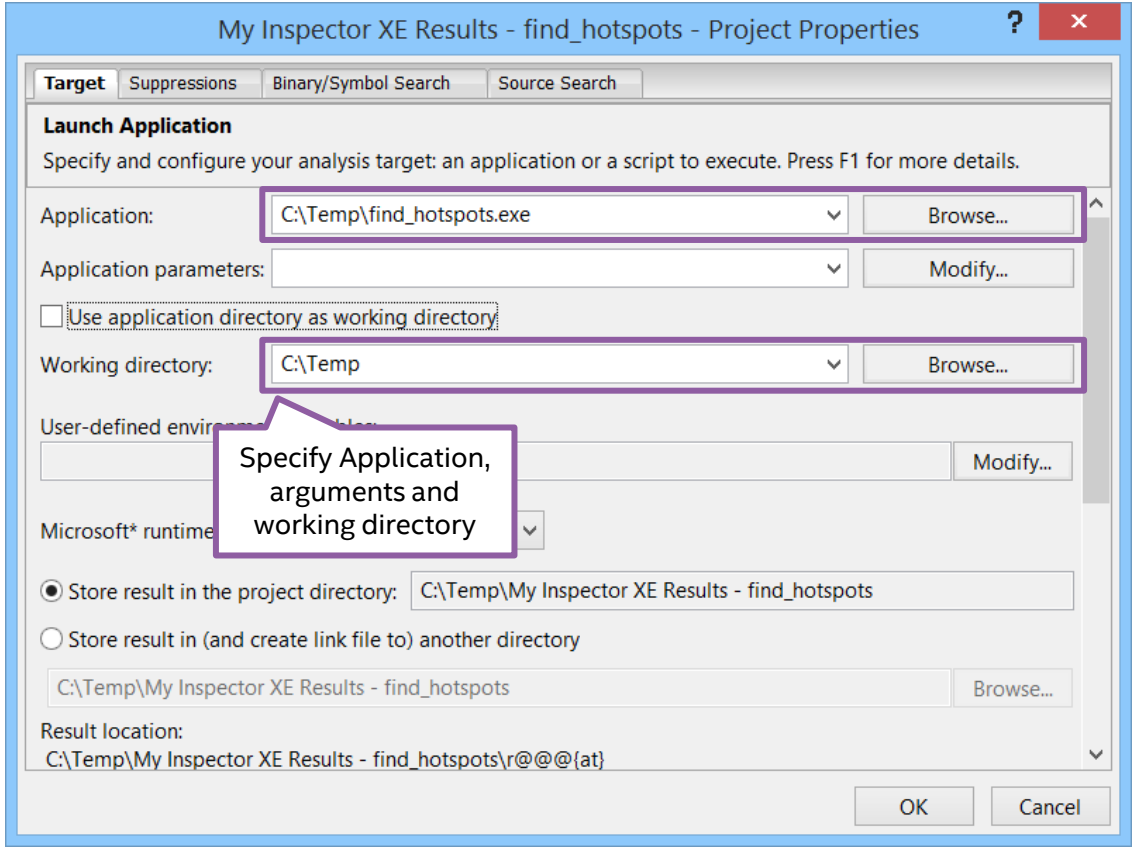
- Data Races
- Deadlocks
- Cross Stack References

Multi-threading problems

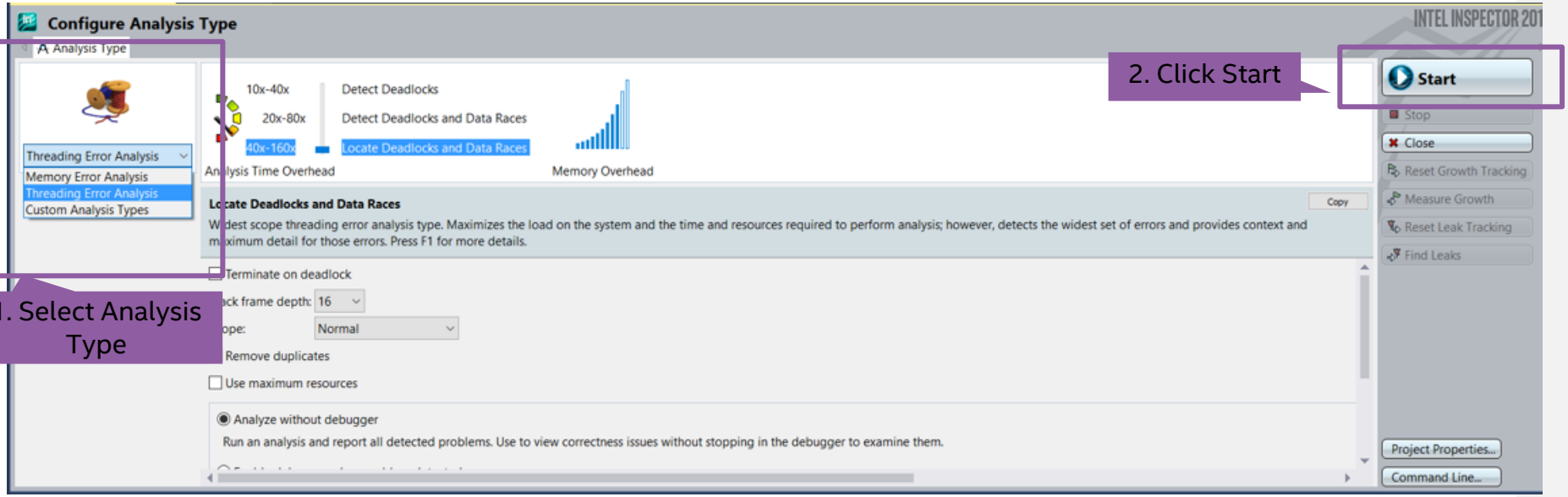
- Hard to reproduce,
- Difficult to debug
- Expensive to fix

→ Let the tool do it for you

Workflow: setup project



Workflow: select analysis and start



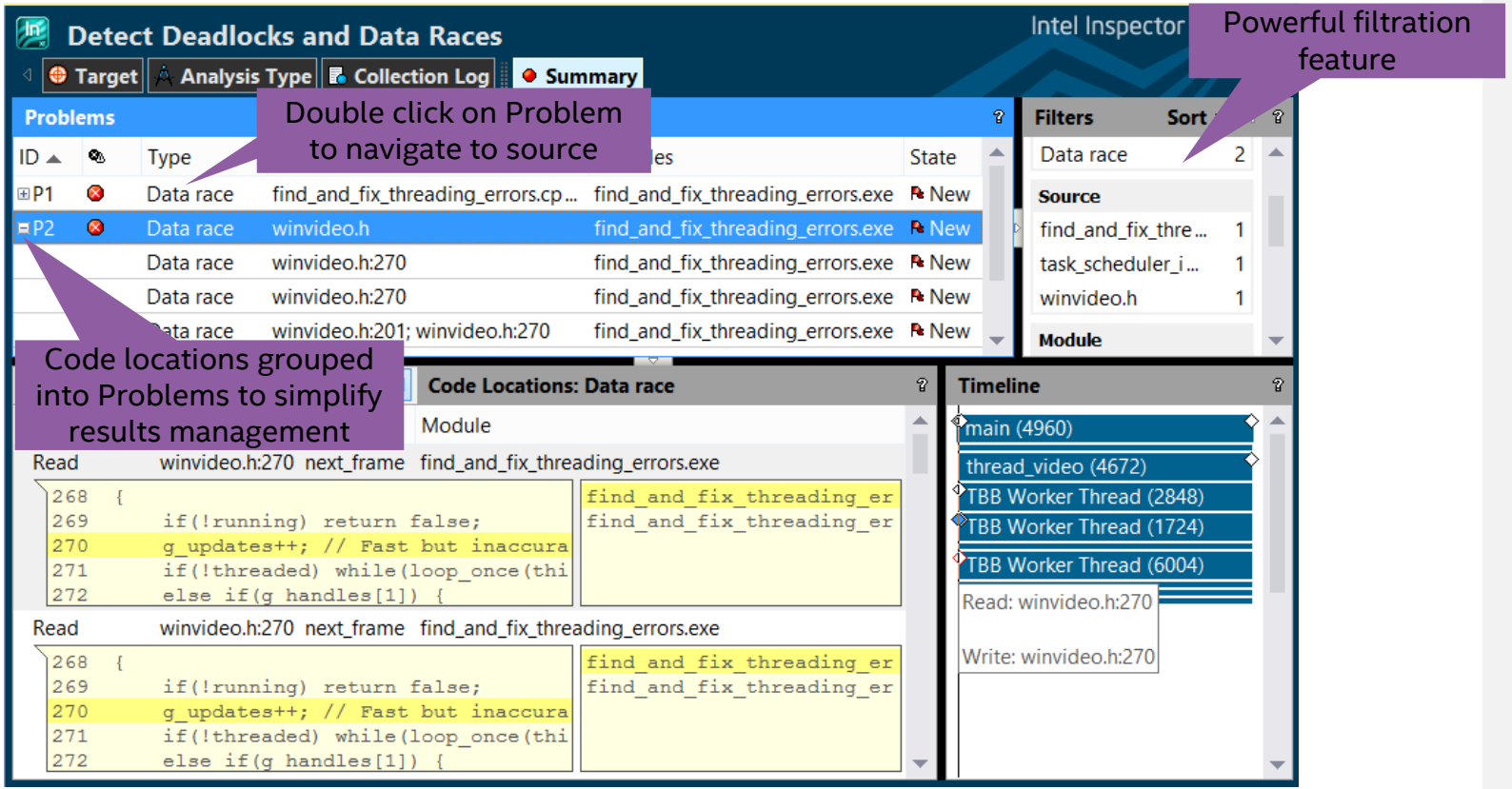
Command Line Interface

- Start analysis
  - *Memory:* inspxe-cl -c mi3 -- <app> [app_args]
  - *Threading:* inspxe-cl -c ti3 -- <app> [app_args]
- View results
  - inspxe-cl -report=problems -report-all –r <result-dir>
  - To open result in GUI, type:
    inspxe-gui <result folder>

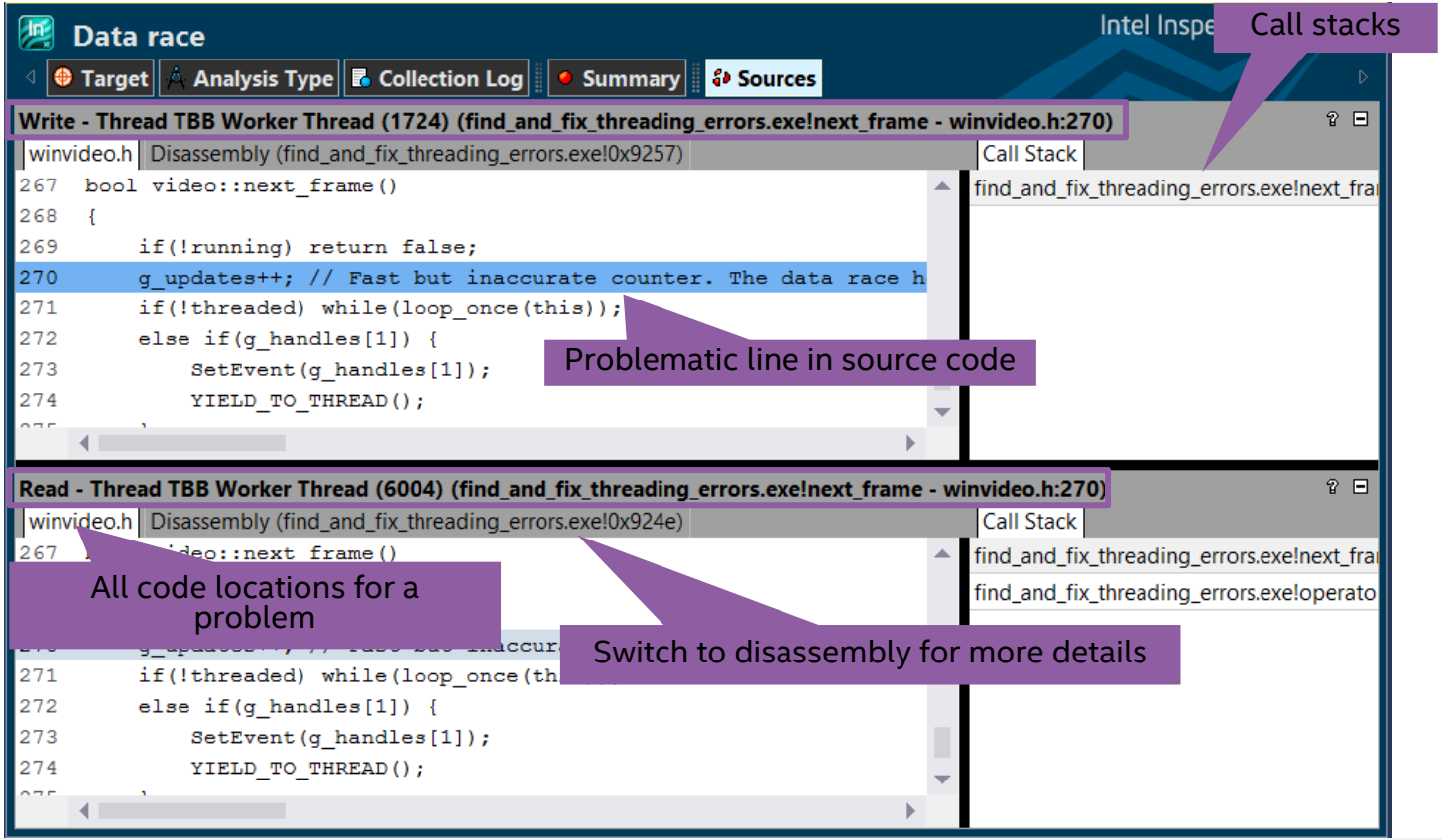
Command Line Interface – Intel MPI

- Use gtool flag or environment variable:
  - *flag: $ mpirun –gtool* “inspxe-cl -c mi3 –r <result_dir>:0” –n N <app> [app_args]
  - *env:* $ export I_MPI_GTOOL=“inspxe-cl -c mi3 –r <result_dir>:0”
- Gtool inserts tool on selected ranks
  - Analysis only on rank 0: use “:0”
  - Analysis on selected ranks: use “:m-n”
  - Analysis on all ranks: use “:all”
- In most cases it should be sufficient to do analysis on single rank!

Workflow: manage results



Workflow: navigate to sources



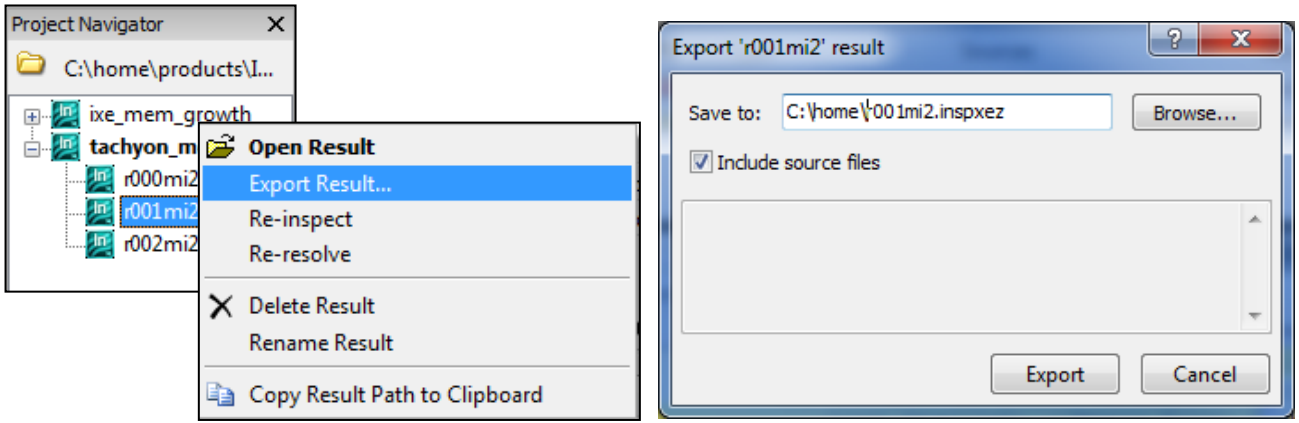
Exporting results

Save results with sources – copy and browse anywhere without setting search paths

CLI:

```
inspxe-cl -export -archive-name r000mi2.inspxez
   -include-sources -result-dir r000mi2
```

GUI:



Work on remote computer

- Working with GUI on remote system might be not possible
- Result generated by CLI on remote system can be exported to archive (plain file)
- Transfer archive from remote system to local desktop/laptop with local Inspector installation. Linux results can be analyzed by Windows Inspector

Demo – Hands on

- Try Inspector on DevCloud or local system
- Playbook will provide some sample command lines on DevCloud: **/data/comp/workshop/Playbook_Inspector.txt**
- More information:

  https://www.intel.com/content/www/us/en/developer/tools/oneapi/inspector.html

- Video:

  https://www.intel.com/content/www/us/en/developer/videos/introduction-to-intel-inspector.html

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Backup

Additional information

Using the Intel® Inspector with MPI (mpich etc)

- Use the command-line tool under the MPI run scripts to gather report data

```
mpirun -n 4 inspxe-cl –-result-dir insp_results
   -collect mi1 -- ./insp_example.exe
```

- Output is: a results directory for each MPI rank in the job

`ls | grep inspector_results` on Linux

- Launch the GUI and view the results for each particular rank

`inspxe-gui inspector_results.<rank#>` on Linux

Memory problems

- **Memory leak**
  - a block of memory is allocated
  - never deallocated
  - not reachable (there is no pointer available to deallocate the block)
  - Severity level = (Error)

```
// Memory leak

char *pStr = (char*) malloc(512);
return;
```

- **Memory not deallocated**
  - a block of memory is allocated
  - never deallocated
  - still reachable at application exit (there is a pointer available to deallocate the block).
  - Severity level = (Warning)

```
// Memory not deallocated

static char *pStr = malloc(512);
return;
```

- **Memory growth**
  - a block of memory is allocated
  - not deallocated, within a specific time segment during application execution.
  - Severity level = (Warning)

```
// Memory growth

// Start measuring growth
static char *pStr = malloc(512);
// Stop measuring growth
```

Memory problems

- **Uninitialized memory access**
  - Read of an uninitialized memory location

- **Invalid Memory Access**
  - Read or write instruction references memory that is logically or physically invalid

- **Kernel Resource Leak**
  - Kernel object handle is created but never closed

- **GDI Resource Leak**
  - GDI object is created but never deleted

```
// Uninitialized Memory Access

void func()
{
   int a;
   int b = a * 4;
}
```

```
// Invalid Memory Access

char *pStr = (char*) malloc(20);
free(pStr);
strcpy(pStr, "my string");
```

```
// Kernel Resource Leak

HANDLE hThread = CreateThread(0,
      8192, work0, NULL, 0,
NULL);
return;
```

```
// GDI Resource Leak

HPEN pen = CreatePen(0, 0, 0);
return;
```

Data race

```
CRITICAL_SECTION cs;              // Preparation
int *p = malloc(sizeof(int)); // Allocation Site
*p = 0;
InitializeCriticalSection(&cs);
```

Write -> Write Data Race

Thread #1

```
*p = 1; // First Write
```

Thread #2

```
EnterCriticalSection(&cs)
;
*p = 2; // Second Write
LeaveCriticalSection(&cs)
;
```

Read -> Write Data Race

Thread #1

```
int x;
x = *p; // Read
```

Thread #2

```
EnterCriticalSection(&cs)
;
*p = 2; // Write
LeaveCriticalSection(&cs)
;
```


intel