Find and Squash Races, Deadlocks, and Memory Bugs with Intel® Inspector

Memory & Threading Debugger

intel

Kevin O'Leary
Intel

SPDK, PMDK, Intel® Performance Analyzers | Virtual Forum

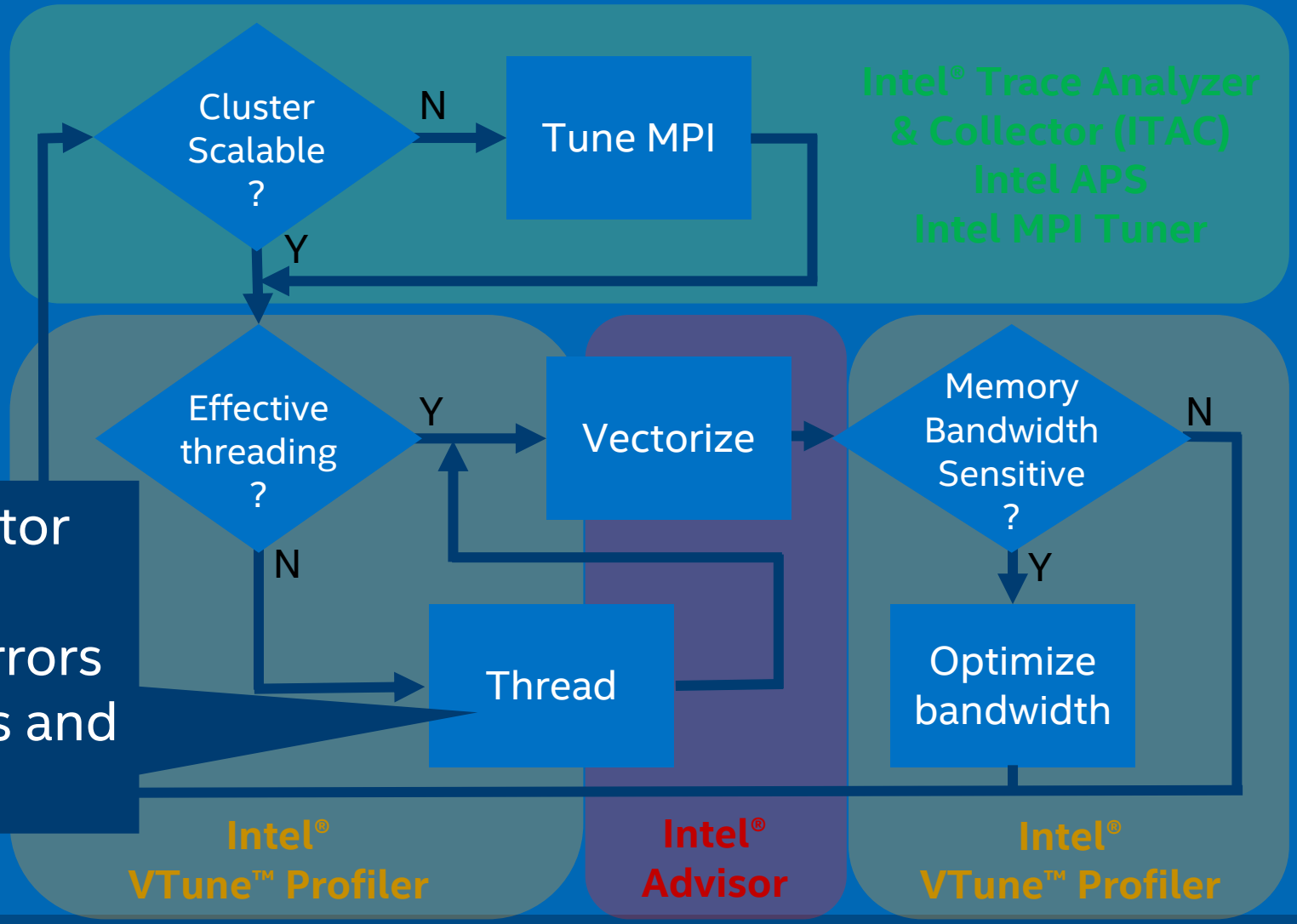
Agenda

01 Intel Inspector overview

02 Intel Inspector features

03 Summary

Analysis Tools for Diagnosis



Intel® Inspector
Find any correctness errors in your threads and memory!

Correctness Tools Increase ROI By 12%-21%

Size and complexity of applications is growing

Reworking defects is 40%-50% of total project effort

Correctness tools find defects during development prior to shipment

Reduce time, effort, and cost to repair

Find errors earlier when they are less expensive to fix

Find & Debug Memory & Threading Errors

Intel® Inspector – Memory & Thread Debugger

Correctness Tools Increase ROI By 12%-21%[1]

- Errors found earlier are less expensive to fix
- Several studies, ROI% varies, but earlier is cheaper

Diagnosing Some Errors Can Take Months

- Races & deadlocks not easily reproduced
- Memory errors can be hard to find without a tool

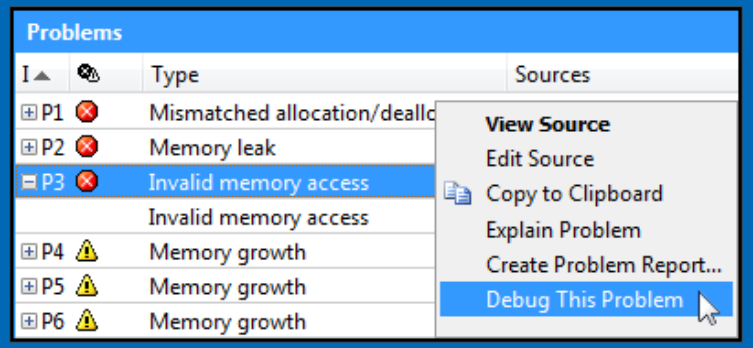
Debugger Integration Speeds Diagnosis

- Breakpoint set just before the problem
- Examine variables & threads with the debugger

Diagnose in hours instead of months

[1] **Cost Factors – Square Project Analysis**
CERT: U.S. Computer Emergency Readiness Team, and Carnegie Mellon CyLab
NIST: National Institute of Standards & Technology : Square Project Results

Debugger Breakpoints



Intel® Inspector dramatically sped up our ability to track down difficult to isolate threading errors before our packages are released to the field.

Peter von Kaenel, Director, Software Development, ***Harmonic Inc.***

http://intel.ly/inspector-xe

Debug Memory & Threading Errors

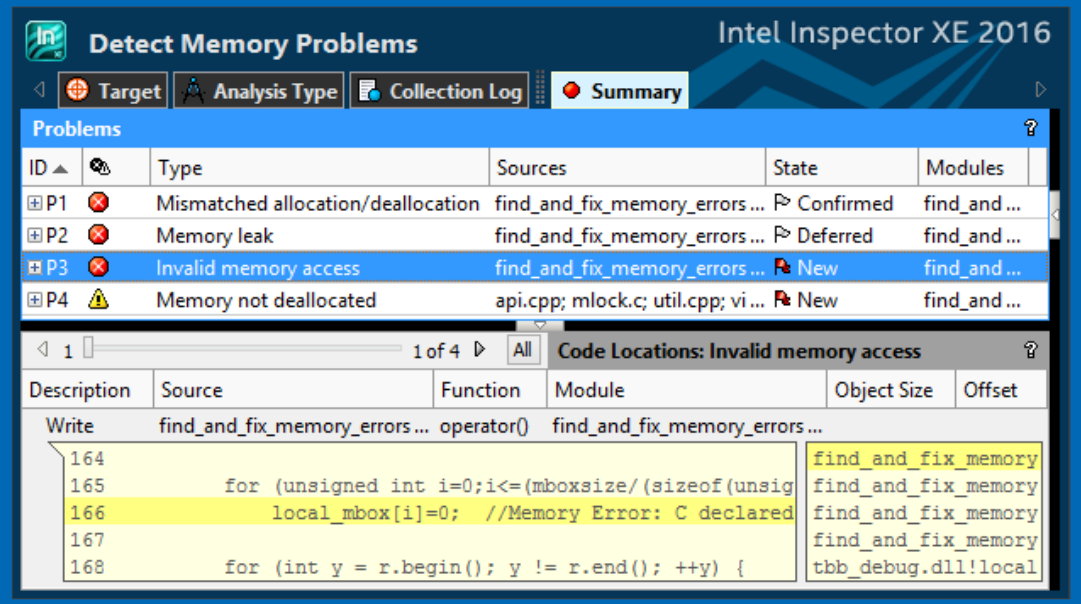
Intel® Inspector

Find and eliminate errors

- Memory leaks, invalid access…
- Races & deadlocks
- Persistence memory issues
- C, C++, C#, F# and Fortran (or a mix)

Simple, Reliable, Accurate

- No special recompiles
  Use any build, any compiler[1]
- Analyzes dynamically generated or linked code
- Inspects 3rd party libraries without source
- Productive user interface + debugger integration
- Command line for automated regression analysis



Clicking an error instantly displays source code snippets and the call stack

Fits your existing process

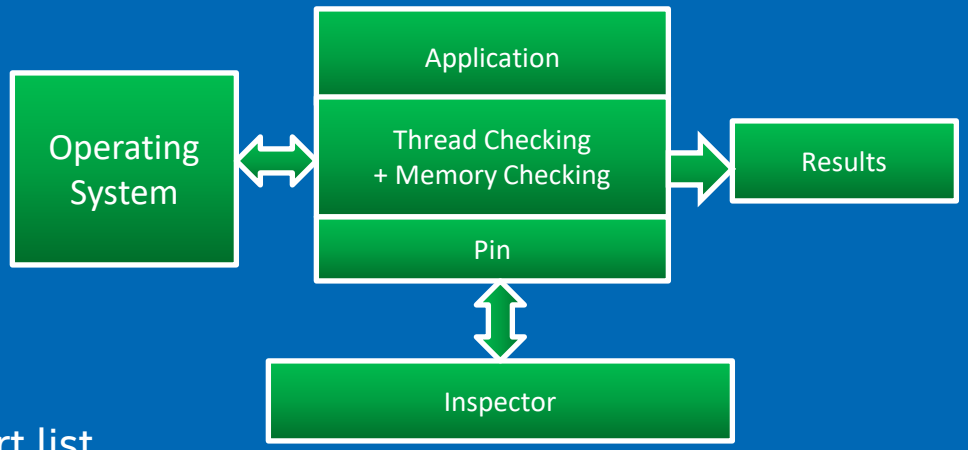
[1]That follows common OS standards.

Intel® Inspector dynamic analysis

Data Collection Techniques

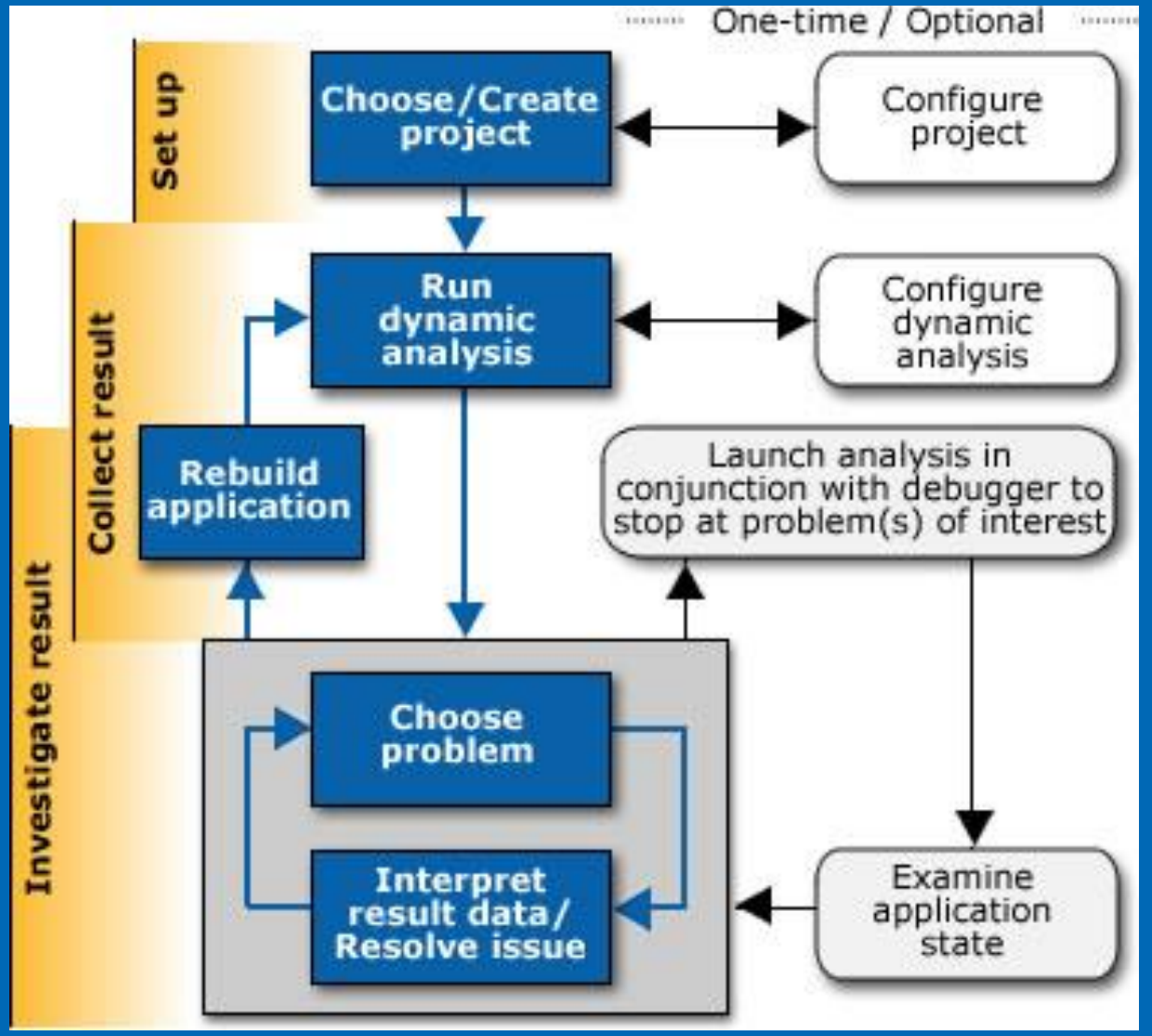
Inspector tracks all memory allocations and threading APIs using a binary instrumentation tool called Pin

- Dynamic instrumentation system provided by Intel (http://www.pintool.org)
- Injected code used for observing the behaviour of the program
- Source modification/recompilation is not needed



- OS has to be in the support list
- One process is analysed at a time

Recommended Methodology



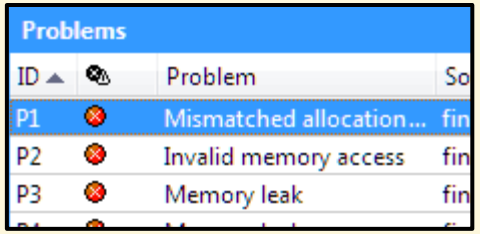
Deliver More Reliable Applications

Intel® Inspector

Intel® Inspector

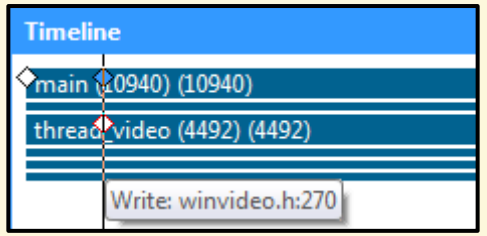
- Dynamic instrumentation
- No special builds
- Any compiler[1]
- Source not required

Memory Errors



- Invalid Accesses
- Memory Leaks
- Uninit. Memory Accesses

Threading Errors



- Races
- Deadlocks
- Cross Stack References

Find errors earlier with less effort

[1]That follows common OS standards.

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

```
// Uninitialized Memory Access

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

Invalid Memory Access

- Read or write instruction references memory that is logically or physically invalid

```
// Invalid Memory Access

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

Kernel Resource Leak

- Kernel object handle is created but never closed

```
// Kernel Resource Leak

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

GDI Resource Leak

- GDI object is created but never deleted

```
// GDI Resource Leak

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

Threading problem Analysis

Analyzed as software runs

- Data (workload) -driven execution
- Program needs to be multi-threaded
- Diagnostics reported incrementally as they occur

Includes monitoring of:

- Thread and Sync APIs used
- Thread execution order
  - Scheduler impacts results
- Memory accesses between threads

Analysis scope

- Native code: C, C++, Fortran
- Managed or mixed code: C# (.NET 2.0 to 3.5, .NET 4.0 with limitations)
- Code path must be executed to be analyzed
- Workload size doesn't affect ability to detect a problem

Race Conditions Are Difficult to Diagnose

They only occur occasionally and are difficult to reproduce

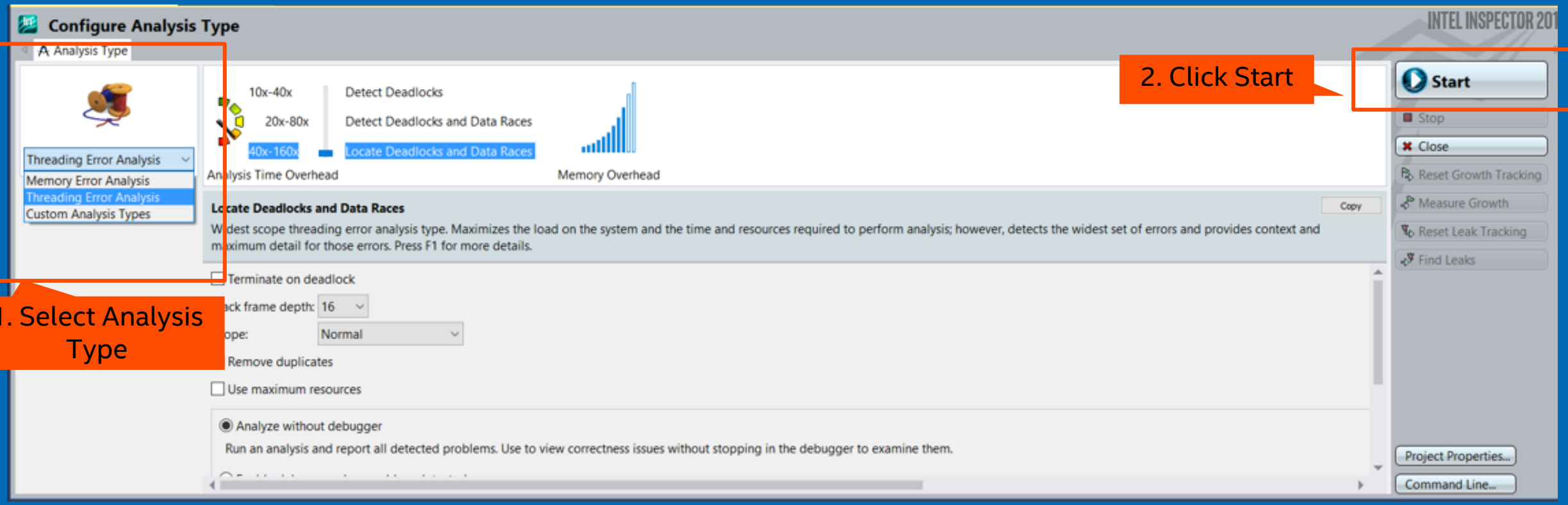
Correct

Thread 1	Thread 2		Shared Counter
			0
Read count		←	0
Increment			0
Write count		→	1
	Read count	←	1
	Increment		1
	Write count	→	2

Incorrect

Thread 1	Thread 2		Shared Counter
			0
Read count		←	0
	Read count	←	0
Increment			0
	Increment		0
Write count		→	1
	Write count	→	1

Workflow: select analysis and start

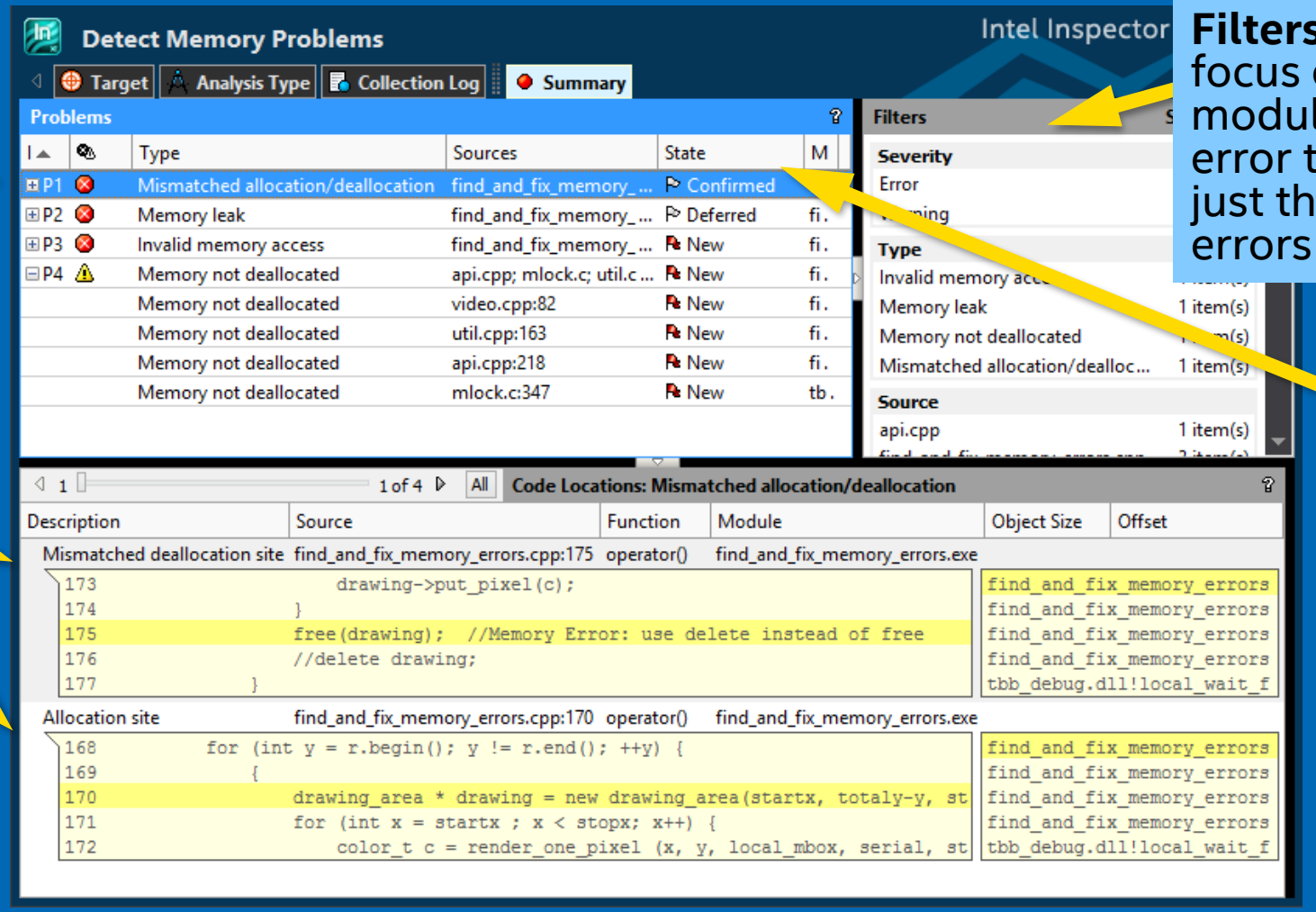


Productive User Interface Saves Time

Intel® Inspector

Select a **problem set**

Code snippets displayed for selected problem



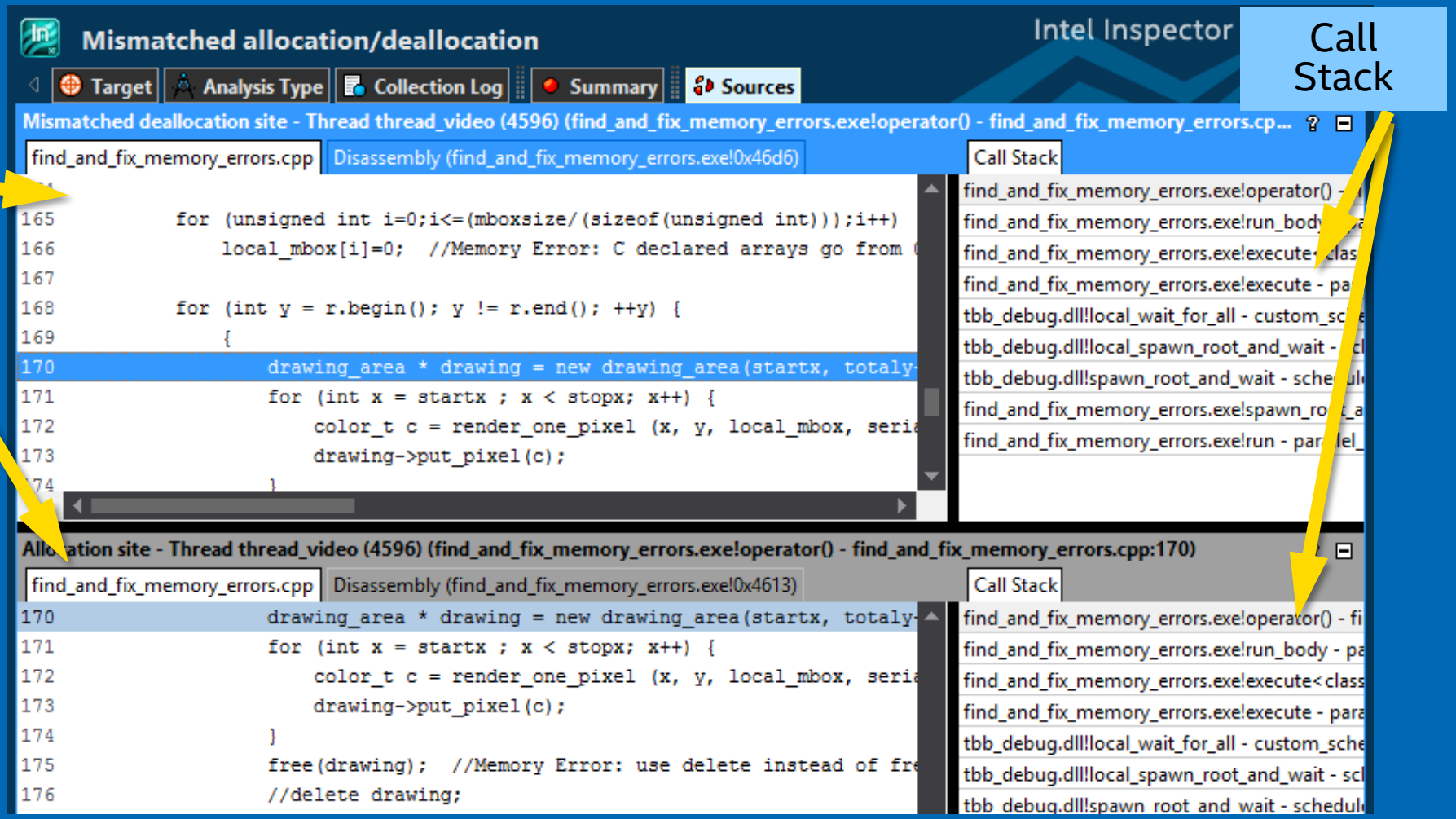
Filters let you focus on a module, or error type, or just the new errors or...

Problem States: New, Not Fixed, Fixed, Confirmed, Not a problem, Deferred, Regression

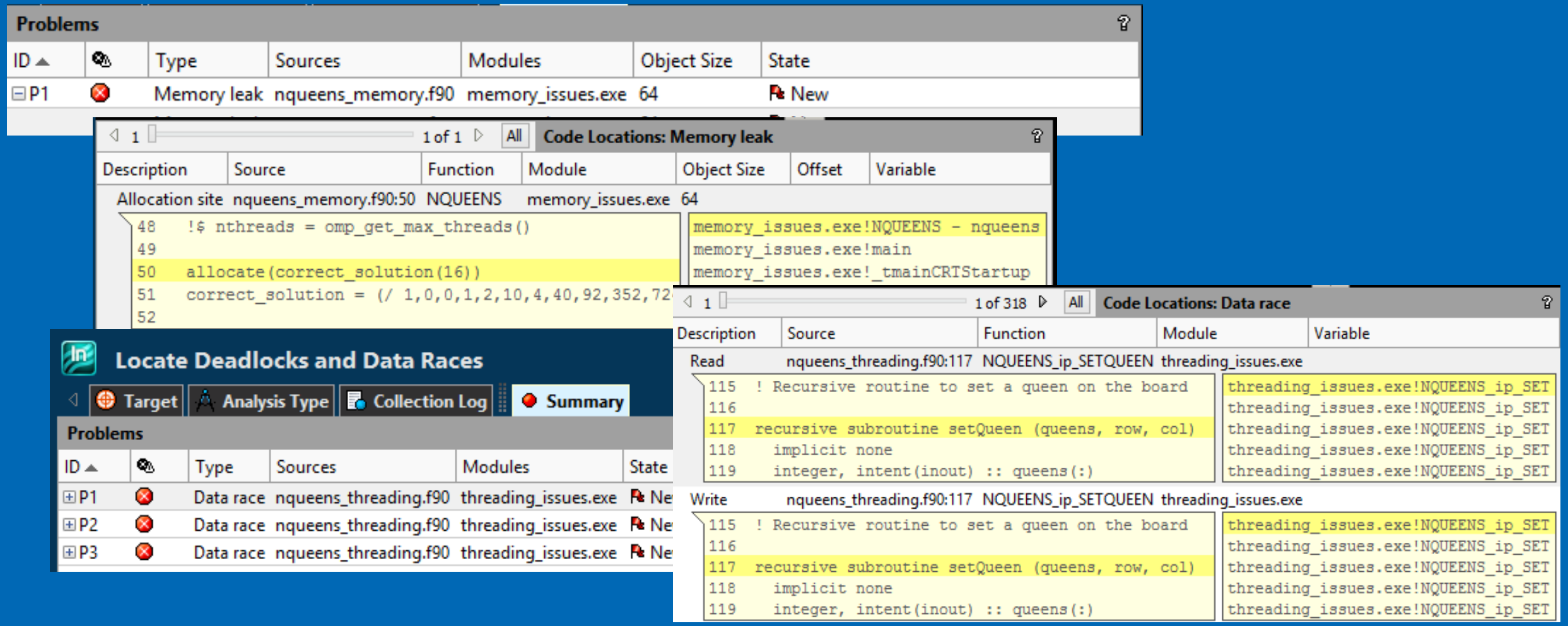
Double Click for Source & Call Stack

Intel® Inspector

Source code locations displayed for selected problem



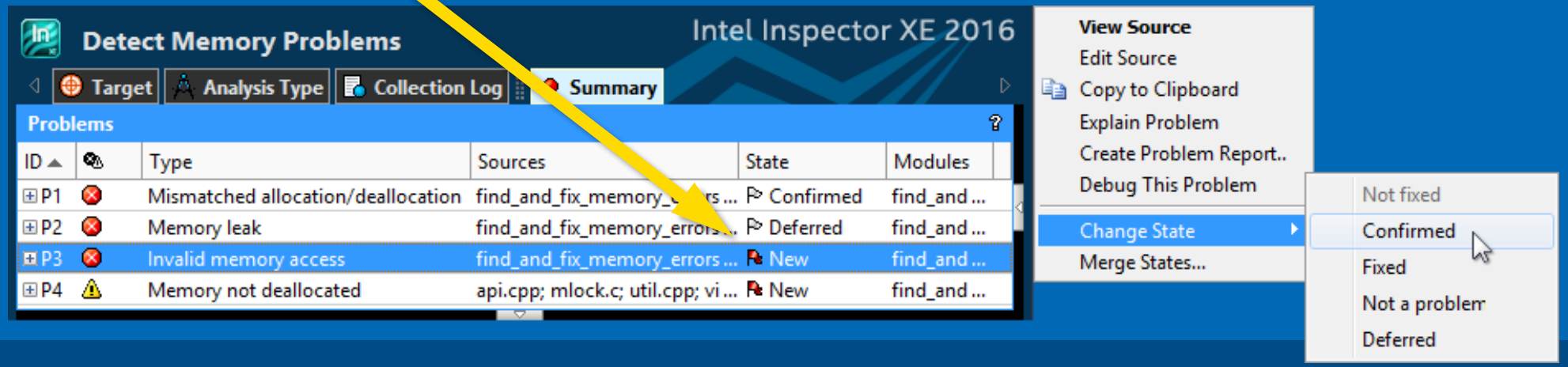
Quickly track down your Fortran issues!



Easy Problem Management

Quickly see new problems and regressions

State	Description
New	Detected by this run
Not Fixed	Previously seen error detected by this run
Not a Problem	Set by user (tool will not change)
Confirmed	Set by user (tool will not change)
Fixed	Set by user (tool will change)
Regression	Error detected with previous state of “Fixed”

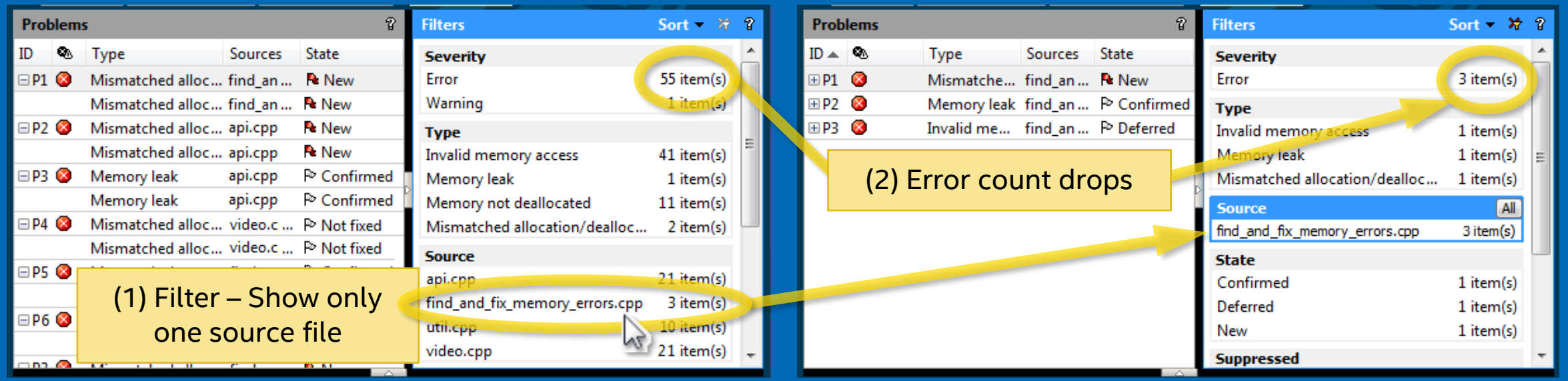


Filtering - Focus on What's Important

Example: See only the errors in one source file

Before – All Errors

After – Only errors from one source file



Tip: Set the "Investigated" filter to "Not investigated" while investigating problems.
This removes from view the problems you are done with, leaving only the ones left to investigate.

Incrementally Diagnose Memory Growth

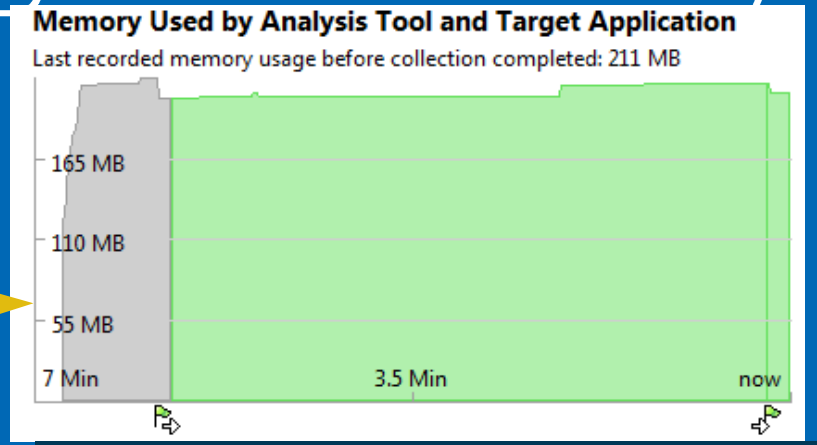
Intel® Inspector

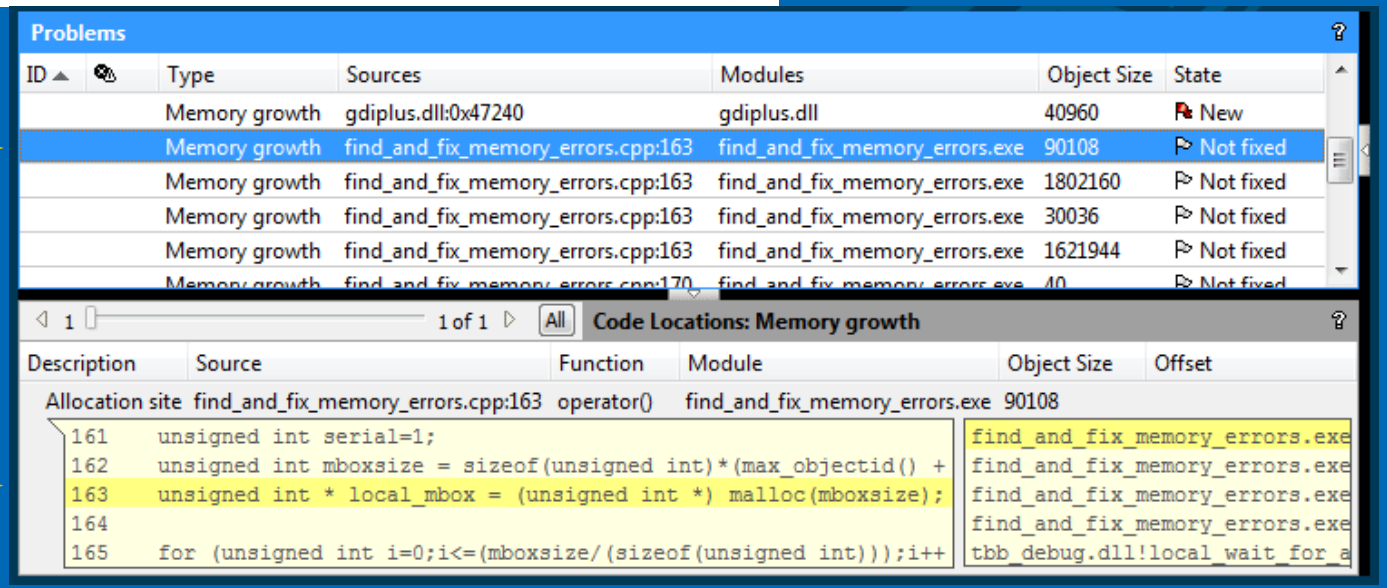
As your app is running…

Memory usage graph plots memory growth

Select a cause of memory growth

See the code snippet & call stack





Speed diagnosis of difficult to find heap errors

Automate Regression Analysis

Command Line Interface

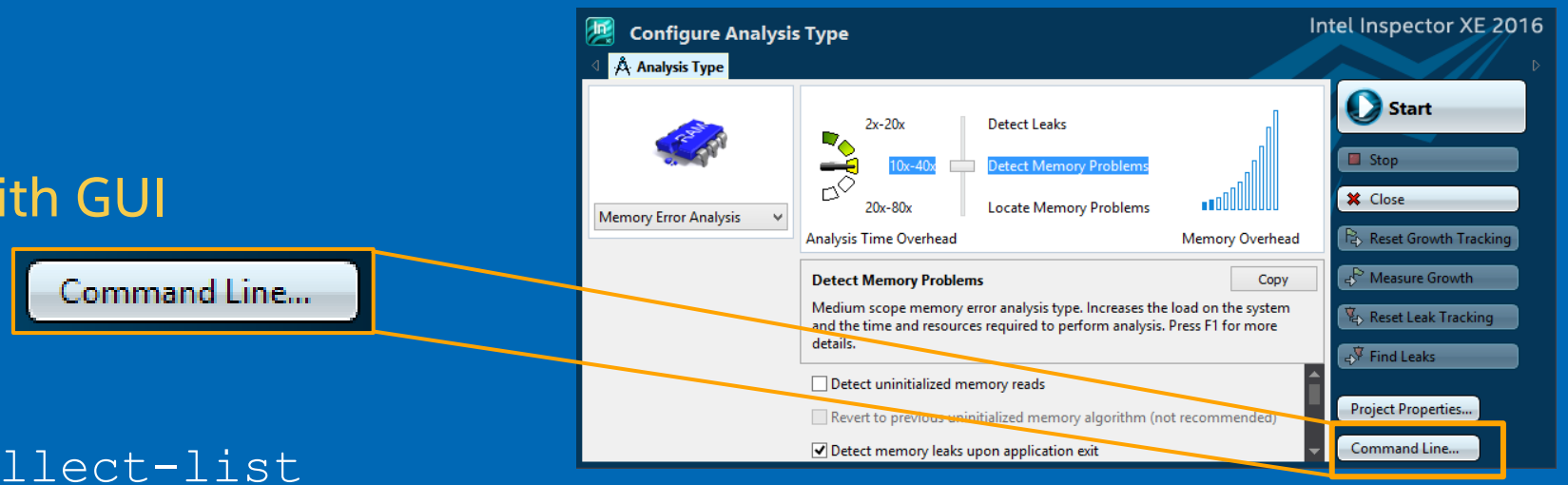
inspxe-cl is the command line:

- **Windows:** `C:\Program Files\Intel\Inspector\bin64\inspxe-cl.exe`
- **Linux:** `/opt/intel/inspector/bin64/inspxe-cl`

Help:

```
inspxe-cl -help
```

Set up command line with GUI



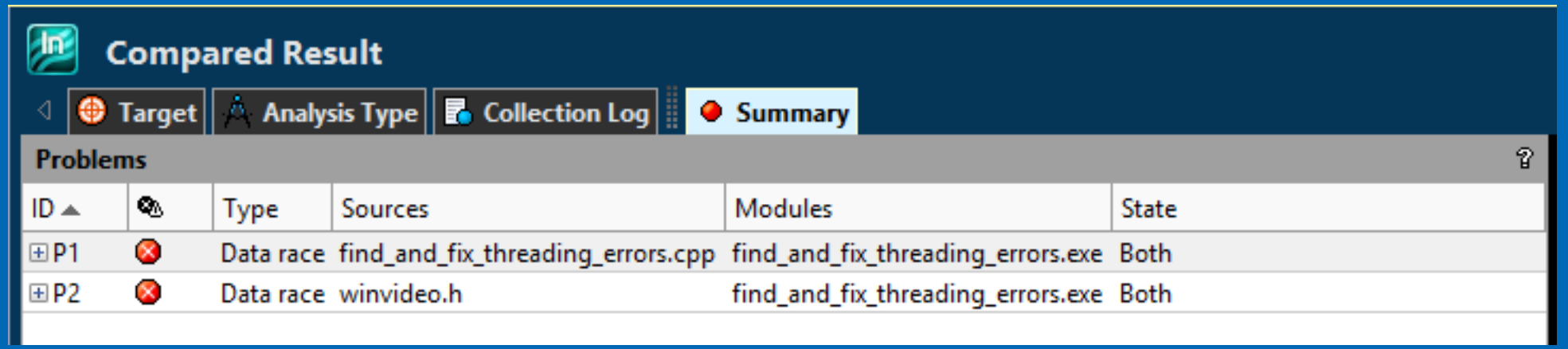
Command examples:

```
1.inspxe-cl -collect-list
2.inspxe-cl -collect ti2 -- MyApp.exe
3.inspxe-cl -report problems
```

Send results file to developer to analyze with the UI

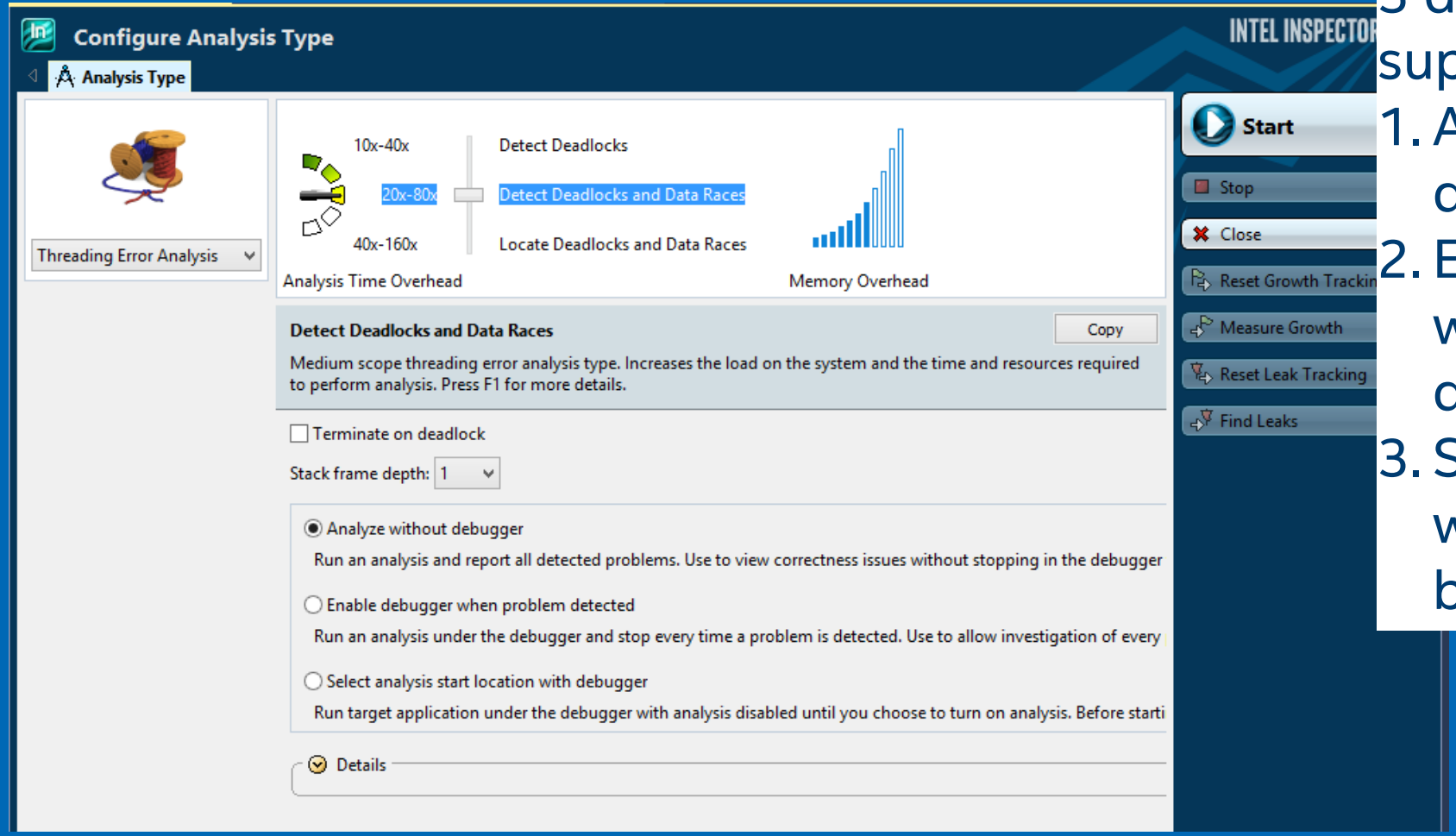
Compare results and see what has changed

Ideal for regression testing



Find problems quicker!

Interactive debugging support



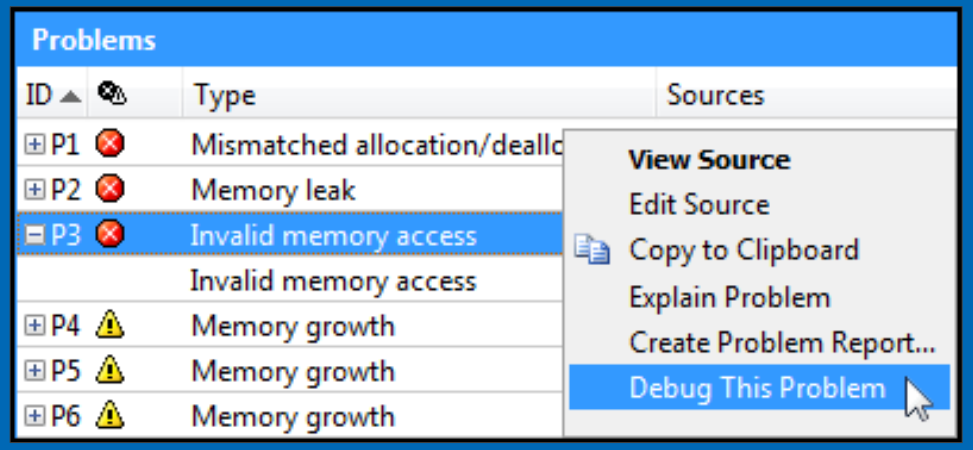
3 debugging modes supported

1. Analyze without debugger
2. Enable debugger when problem detected
3. Start analysis when a debug breakpoint is hit.

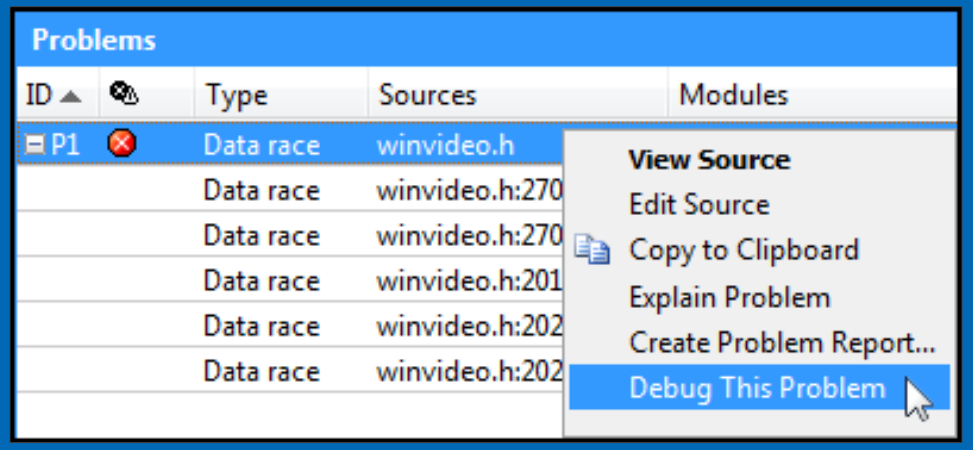
Break At Just The Right Time

Intel® Inspector - Memory & Thread Debugger

Memory Errors



Threading Errors



Break into the debugger just before the error occurs.

Examine the variables and threads.

Diagnose the problem.

Save time. Find and diagnose errors with less effort.

Work Smarter & Faster

Intel® Inspector - Memory & Thread Debugger

Precise Error Suppression

```
Suppression = {
    Name = "Example";
    Type = { uninitialized_memory_access }
    Stacks = {
        {
            mod=a.out, func=update_x;
            func=main;
```

Precise, easy to edit, team shareable.

Choose which stack frame to suppress.

Eliminate the false, not the real errors.

Pause/Resume Collection

```
__itt_suppress_push(__itt_suppress_threading_errors);
    /* Any threading errors here are ignored */
__itt_suppress_pop();
    /* Any threading errors here are seen */
```

Speed-up analysis by limiting its scope.

Analyze only during the execution of the suspected problem.

Find and diagnose errors with less effort.

Productive Memory & Threading Debugger

Intel® Inspector

	Memory Analysis	Threading Analysis
View Context of Problem		
Stack	✓	✓
Multiple Contributing Source Locations	✓	✓
Collapse multiple “sightings” to one error (e.g., memory allocated in a loop, then leaked is 1 error)	✓	✓
Suppression, Filtering, and Workflow Management	✓	✓
Visual Studio* Integration (Windows*)	✓	✓
Command line for automated tests	✓	✓
Time Line visualization	✓	✓
Memory Growth during a transaction	✓	
Trigger Debugger Breakpoint	✓	✓

Easier & Faster Debugging of Memory & Threading Errors

SPDK, PMDK, Intel® Performance Analyzers | **Virtual Forum**

Persistence Inspector

Persistent Memory Programming Challenges

- When to flush stored data out of cache hierarchy?
  - Memory store does not become persistent immediately
  - Data gets persistent only after it is out of cache and arrives at the memory subsystem
- Missing or incorrect cache flushes can leave data in inconsistent or unrecoverable state in case of power failure or system crash
- Excessive cache flushes hurt performance
- Testing and existing development tools do not find missing/incorrect/ excessive cache flushes

What is the Intel® Inspector – Persistence Inspector tool?

Overview

- A run-time tool developers can use to detect programming errors in Persistent Memory programs. In addition to cache flush misses, this tool detects:
  - Redundant cache flushes and memory fences
  - Out-of-order persistent memory stores
  - Incorrect undo logging for the Persistent Memory Development Kit (PMDK)
- You can use the Intel® Inspector GUI to visualize the data collected

Call to Action

Modernize your Code

- To get the most out of your hardware, you need to modernize your code with vectorization and threading.
- Taking a methodical approach such as the one outlined in this presentation, and taking advantage of the powerful tools in Intel® oneAPI, can make the modernization task dramatically easier.


intel®
SPDK, PMDK, Intel® Performance Analyzers
Virtual Forum