

# 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

---

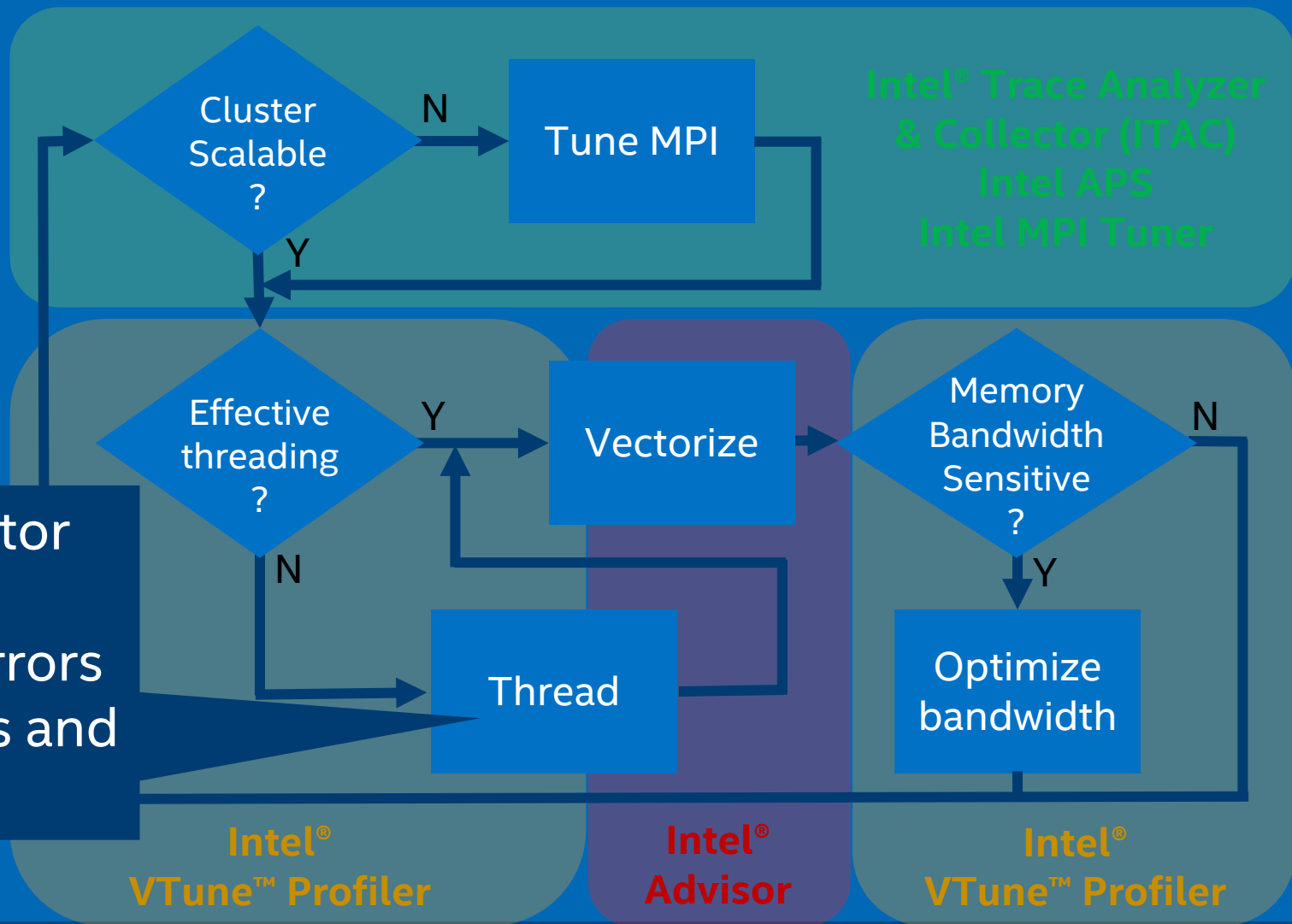
02 Intel Inspector features

---

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%<sup>1</sup>

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

## Diagnosing Some Errors Can Take Months

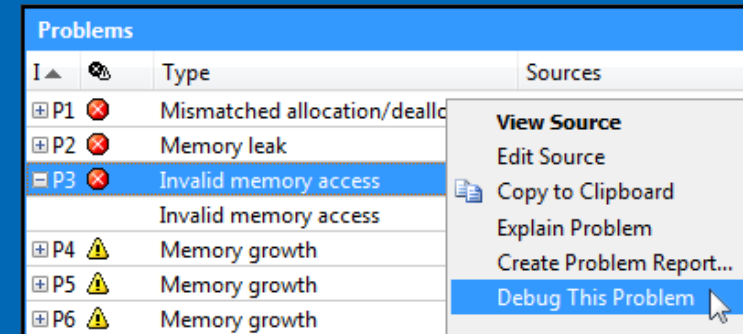
- Races & deadlocks not easily reproduced
- Memory errors can be hard to find without a tool

## Debugger Integration Speeds Diagnosis

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

**Diagnose in hours instead of months**

## 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>

<sup>1</sup> 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

# Debug Memory & Threading Errors

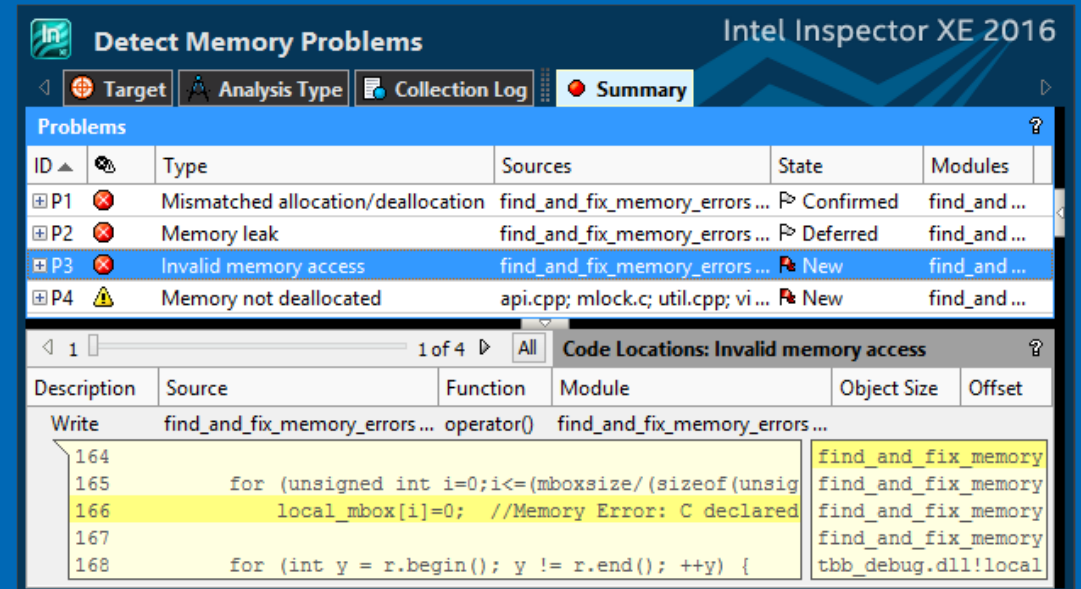
Intel® Inspector

## 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<sup>1</sup>
- Analyzes dynamically generated or linked code
- Inspects 3<sup>rd</sup> 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**

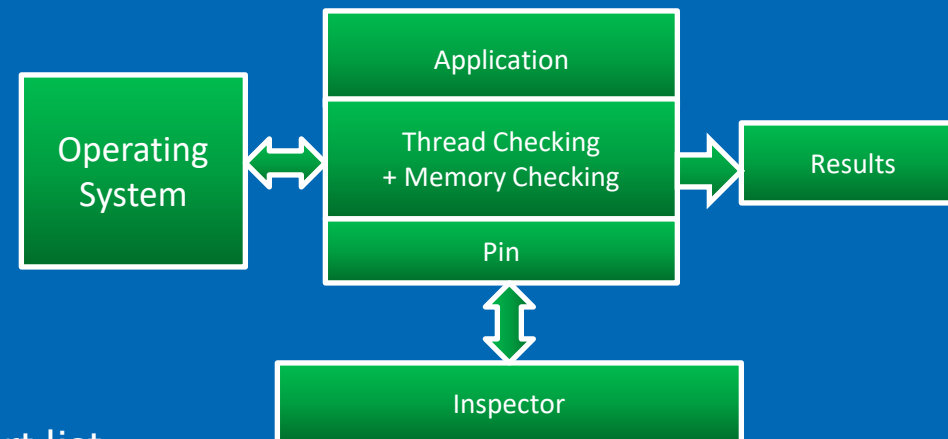
<sup>1</sup>That follows common OS standards.

# Intel® Inspector dynamic analysis

## Data Collection Techniques

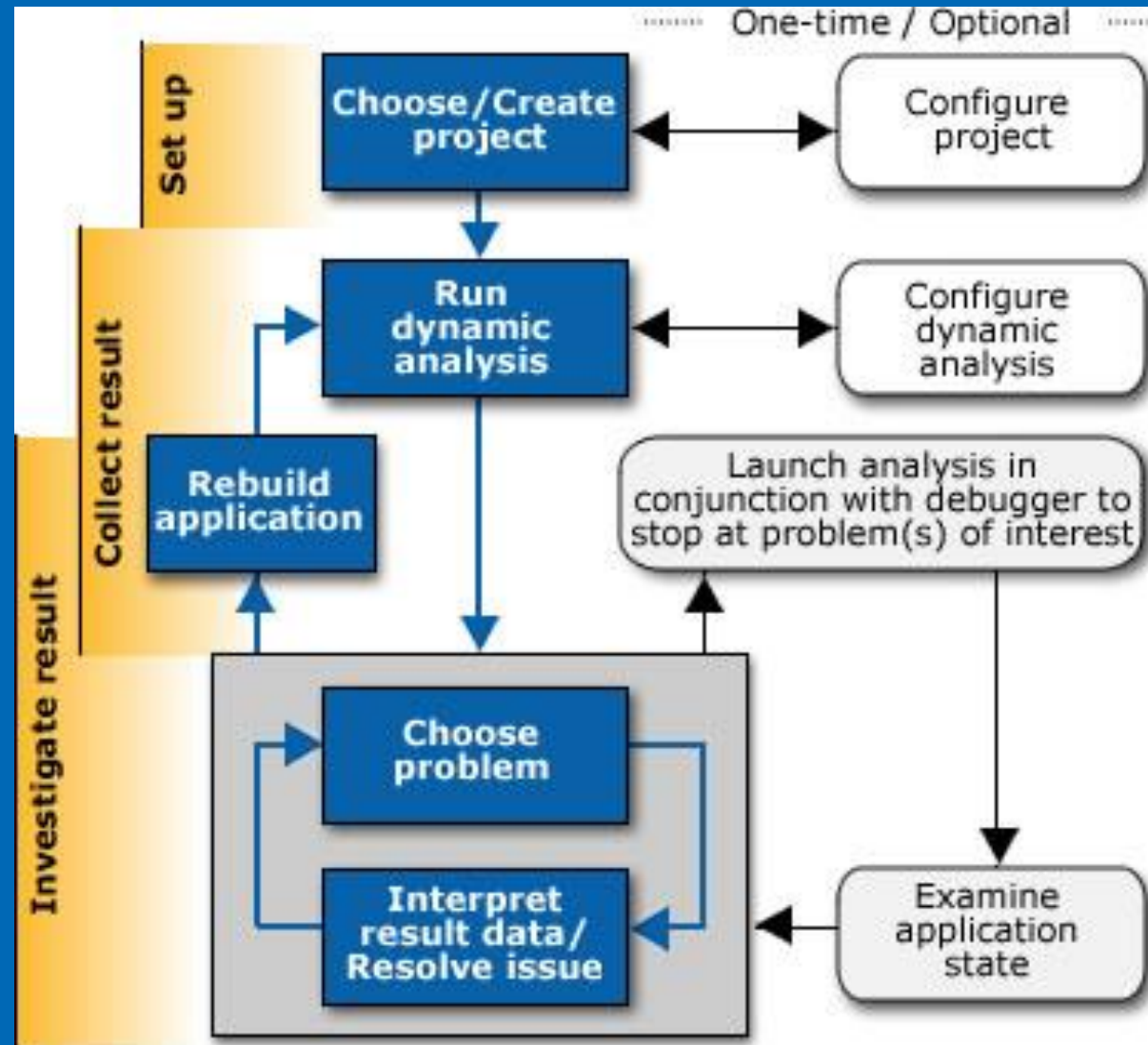
### Inspector tracks all memory allocations and threading APIs using a binary instrumentation tool called Pin

- 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





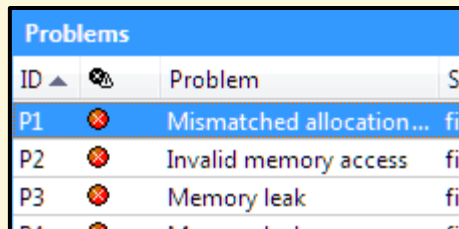
# Deliver More Reliable Applications

Intel® Inspector

## Intel® Inspector

- Dynamic instrumentation
- No special builds
- Any compiler<sup>1</sup>
- Source not required

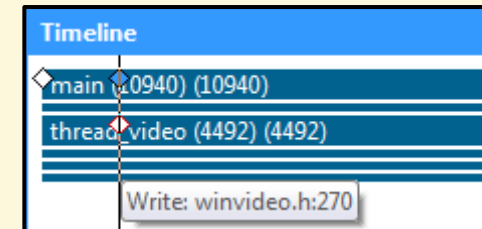
## Memory Errors



ID	Problem	So
P1	Mismatched allocation...	fin
P2	Invalid memory access	fin
P3	Memory leak	fin

- Invalid Accesses
- Memory Leaks
- Uninit. Memory Accesses

## Threading Errors



Thread	Event
main (10940) (10940)	Write: winvideo.h:270
thread_video (4492) (4492)	

- Races
- Deadlocks
- Cross Stack References

**Find errors earlier with less effort**

<sup>1</sup>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 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 growth

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

```
// Memory leak
```

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

```
// Memory not deallocated
```

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

```
// 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);
```

```
// 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

**1. Select Analysis Type**

**2. Click Start**

**Configure Analysis Type**

Analysis Type

- Threading Error Analysis
- Memory Error Analysis
- Threading Error Analysis
- Custom Analysis Types

10x-40x Detect Deadlocks

20x-80x Detect Deadlocks and Data Races

40x-160x **Locate Deadlocks and Data Races**

Analysis Time Overhead

Memory Overhead

**Locate Deadlocks and Data Races**

Widest scope threading error analysis type. Maximizes the load on the system and the time and resources required to perform analysis; however, detects the widest set of errors and provides context and maximum detail for those errors. Press F1 for more details.

Terminate on deadlock

Stack frame depth: 16

Scope: Normal

Remove duplicates

Use maximum resources

Analyze without debugger

Run an analysis and report all detected problems. Use to view correctness issues without stopping in the debugger to examine them.

Start

Stop

Close

Reset Growth Tracking

Measure Growth

Reset Leak Tracking

Find Leaks

Project Properties...

Command Line...

# Productive User Interface Saves Time

Intel® Inspector

Select a problem set

Code snippets displayed for selected problem

The screenshot displays the Intel Inspector interface for detecting memory problems. The main window is titled "Detect Memory Problems" and has tabs for "Target", "Analysis Type", "Collection Log", and "Summary".

The "Problems" table lists several issues:

ID	Type	Sources	State	M
P1	Mismatched allocation/deallocation	find_and_fix_memory_...	Confirmed	
P2	Memory leak	find_and_fix_memory_...	Deferred	fi.
P3	Invalid memory access	find_and_fix_memory_...	New	fi.
P4	Memory not deallocated	api.cpp; mlock.c; util.c...	New	fi.
	Memory not deallocated	video.cpp:82	New	fi.
	Memory not deallocated	util.cpp:163	New	fi.
	Memory not deallocated	api.cpp:218	New	fi.
	Memory not deallocated	mlock.c:347	New	tb.

The "Filters" panel on the right allows filtering by Severity (Error, Warning), Type (Invalid memory access, Memory leak, Memory not deallocated, Mismatched allocation/deallocation), and Source (api.cpp, find\_and\_fix\_memory\_errors.cpp).

The "Code Locations: Mismatched allocation/deallocation" panel shows the following code snippets:

```
Mismatched deallocation site find_and_fix_memory_errors.cpp:175 operator() find_and_fix_memory_errors.exe
173 drawing->put_pixel(c);
174 }
175 free(drawing); //Memory Error: use delete instead of free
176 //delete drawing;
177 }

Allocation site find_and_fix_memory_errors.cpp:170 operator() find_and_fix_memory_errors.exe
168 for (int y = r.begin(); y != r.end(); ++y) {
169 {
170 drawing_area * drawing = new drawing_area(startx, totaly-y, st
171 for (int x = startx ; x < stopx; x++) {
172 color_t c = render_one_pixel (x, y, local_mbox, serial, st
```

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

The screenshot displays the Intel Inspector interface for a memory error. The main window is titled "Mismatched allocation/deallocation". The error message is "Mismatched deallocation site - Thread thread\_video (4596) (find\_and\_fix\_memory\_errors.exe!operator() - find\_and\_fix\_memory\_errors.cp...".

The interface is divided into several panes:

- Target:** find\_and\_fix\_memory\_errors.cpp
- Analysis Type:** Disassembly (find\_and\_fix\_memory\_errors.exe!0x46d6)
- Collection Log:** (empty)
- Summary:** (empty)
- Sources:** find\_and\_fix\_memory\_errors.cpp
- Call Stack:** A list of function calls, with the top entry being find\_and\_fix\_memory\_errors.exe!operator() - find\_and\_fix\_memory\_errors.exe!run\_body...

The source code pane shows the following code:

```
165     for (unsigned int i=0;i<=(mboxsize/(sizeof(unsigned int)));i++)
166         local_mbox[i]=0; //Memory Error: C declared arrays go from 0
167
168     for (int y = r.begin(); y != r.end(); ++y) {
169     {
170         drawing_area * drawing = new drawing_area(startx, totaly
171         for (int x = startx ; x < stopx; x++) {
172             color_t c = render_one_pixel (x, y, local_mbox, serie
173             drawing->put_pixel(c);
174         }
175     }
176     free(drawing); //Memory Error: use delete instead of free
177     //delete drawing;
```

The code is highlighted in blue, indicating it is the selected problem. A yellow arrow points from the text "Source code locations displayed for selected problem" to this code. Another yellow arrow points from the text "Call Stack" to the call stack pane.



# Quickly track down your Fortran issues!

ID	Type	Sources	Modules	Object Size	State
P1	Memory leak	nqueens_memory.f90	memory_issues.exe	64	New

Description	Source	Function	Module	Object Size	Offset	Variable
Allocation site	nqueens_memory.f90:50	NQUEENS	memory_issues.exe	64		
48	!\$ nthreads = omp_get_max_threads()			memory_issues.exe!NQUEENS - nqueens		
49				memory_issues.exe!main		
50	allocate(correct_solution(16))			memory_issues.exe!_tmainCRTStartup		
51	correct_solution = (/ 1,0,0,1,2,10,4,40,92,352,72)					
52						

Description	Source	Function	Module	Variable
Read	nqueens_threading.f90:117	NQUEENS_ip_SETQUEEN	threading_issues.exe	
115	! Recursive routine to set a queen on the board			threading_issues.exe!NQUEENS_ip_SET
116				threading_issues.exe!NQUEENS_ip_SET
117	recursive subroutine setQueen (queens, row, col)			threading_issues.exe!NQUEENS_ip_SET
118	implicit none			threading_issues.exe!NQUEENS_ip_SET
119	integer, intent(inout) :: queens(:)			threading_issues.exe!NQUEENS_ip_SET
Write	nqueens_threading.f90:117	NQUEENS_ip_SETQUEEN	threading_issues.exe	
115	! Recursive routine to set a queen on the board			threading_issues.exe!NQUEENS_ip_SET
116				threading_issues.exe!NQUEENS_ip_SET
117	recursive subroutine setQueen (queens, row, col)			threading_issues.exe!NQUEENS_ip_SET
118	implicit none			threading_issues.exe!NQUEENS_ip_SET
119	integer, intent(inout) :: queens(:)			threading_issues.exe!NQUEENS_ip_SET

**Locate Deadlocks and Data Races**

Target Analysis Type Collection Log Summary

ID	Type	Sources	Modules	State
P1	Data race	nqueens_threading.f90	threading_issues.exe	New
P2	Data race	nqueens_threading.f90	threading_issues.exe	New
P3	Data race	nqueens_threading.f90	threading_issues.exe	New

# 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 <u>not</u> change)
Confirmed	Set by user (tool will <u>not</u> change)
Fixed	Set by user (tool <u>will</u> change)
Regression	Error detected with previous state of "Fixed"

The screenshot shows the Intel Inspector XE 2016 interface. The main window displays a list of memory problems under the 'Problems' tab. A yellow arrow points from the 'New' state in the table above to the 'New' state of problem P3 in the list. A context menu is open over problem P3, showing options like 'View Source', 'Edit Source', 'Copy to Clipboard', 'Explain Problem', 'Create Problem Report..', 'Debug This Problem', 'Change State', and 'Merge States...'. The 'Change State' option is selected, and a sub-menu is visible with options: 'Not fixed', 'Confirmed', 'Fixed', 'Not a problem', and 'Deferred'. The 'Confirmed' option is highlighted by the mouse cursor.

ID	Type	Sources	State	Modules
P1	Mismatched allocation/deallocation	find_and_fix_memory_errors...	Confirmed	find_and ...
P2	Memory leak	find_and_fix_memory_errors...	Deferred	find_and ...
P3	Invalid memory access	find_and_fix_memory_errors...	New	find_and ...
P4	Memory not deallocated	api.cpp; mlock.c; util.cpp; vi ...	New	find_and ...

# 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**

The image shows two side-by-side screenshots of the Visual Studio 'Problems' window. The left screenshot shows a list of 21 error items across various source files. A yellow box with the text '(1) Filter – Show only one source file' points to the 'Filters' pane where 'find\_and\_fix\_memory\_errors.cpp' is selected. The right screenshot shows the same window after filtering, with only 3 error items remaining, all from the selected source file. A yellow box with the text '(2) Error count drops' points to the '3 item(s)' count in the 'Filters' pane. Arrows indicate the flow of information from the filter selection to the resulting filtered list.

Severity	Count
Error	55 item(s)
Warning	1 item(s)
Invalid memory access	41 item(s)
Memory leak	1 item(s)
Memory not deallocated	11 item(s)
Mismatched allocation/deallocation	2 item(s)
Source	
api.cpp	21 item(s)
find_and_fix_memory_errors.cpp	3 item(s)
util.cpp	10 item(s)
video.cpp	21 item(s)

Severity	Count
Error	3 item(s)
Type	
Invalid memory access	1 item(s)
Memory leak	1 item(s)
Mismatched allocation/deallocation	1 item(s)
Source	
find_and_fix_memory_errors.cpp	3 item(s)
State	
Confirmed	1 item(s)
Deferred	1 item(s)
New	1 item(s)
Suppressed	

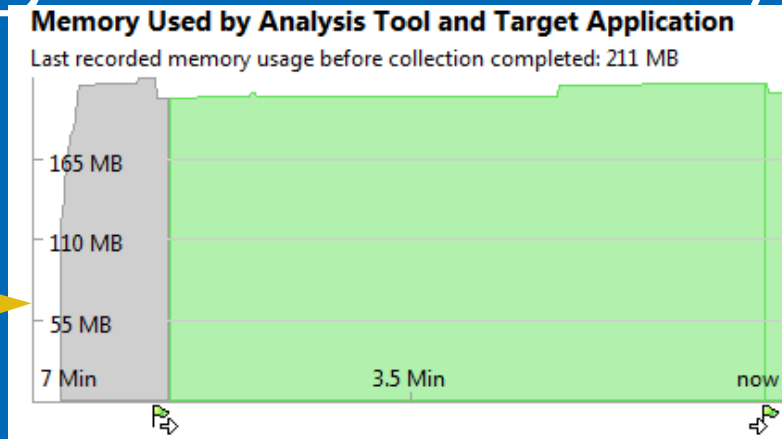
Tip: Set the "Investigated" filter to "Not investigated" while investigating problems. This removes from view the problems you are done with, leaving only the ones left to investigate.

# Incrementally Diagnose Memory Growth

Intel® Inspector

As your app is running...

Memory usage graph plots memory growth



Select a cause of memory growth

ID	Type	Sources	Modules	Object Size	State
	Memory growth	gdiplus.dll:0x47240	gdiplus.dll	40960	New
	Memory growth	find_and_fix_memory_errors.cpp:163	find_and_fix_memory_errors.exe	90108	Not fixed
	Memory growth	find_and_fix_memory_errors.cpp:163	find_and_fix_memory_errors.exe	1802160	Not fixed
	Memory growth	find_and_fix_memory_errors.cpp:163	find_and_fix_memory_errors.exe	30036	Not fixed
	Memory growth	find_and_fix_memory_errors.cpp:163	find_and_fix_memory_errors.exe	1621944	Not fixed
	Memory growth	find_and_fix_memory_errors.cpp:170	find_and_fix_memory_errors.exe	40	Not fixed

See the code snippet & call stack

```
Code Locations: Memory growth
```

Description	Source	Function	Module	Object Size	Offset
Allocation site	find_and_fix_memory_errors.cpp:163	operator()	find_and_fix_memory_errors.exe	90108	

```
161 unsigned int serial=1;
162 unsigned int mboxsize = sizeof(unsigned int)*(max_objectid() +
163 unsigned int * local_mbox = (unsigned int *) malloc(mboxsize);
164
165 for (unsigned int i=0;i<=(mboxsize/(sizeof(unsigned int)));i++
```

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:

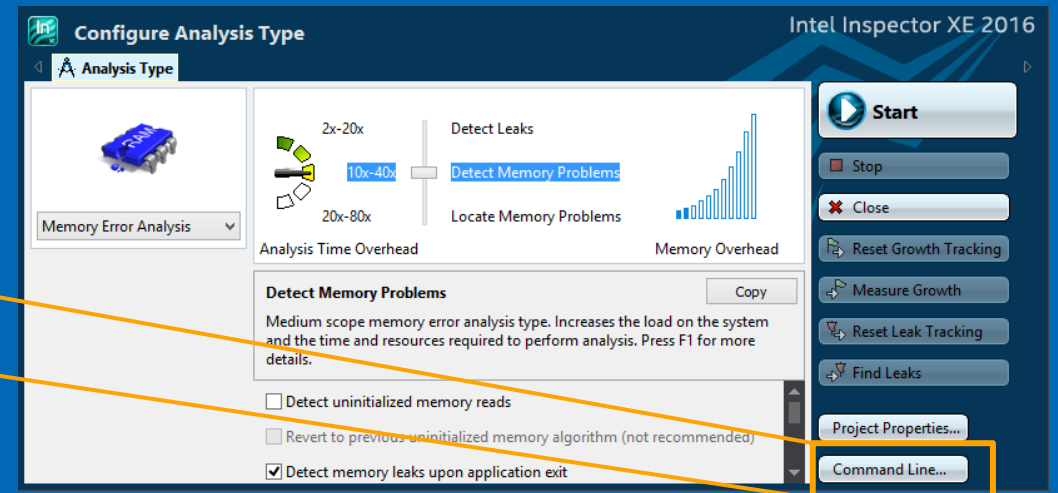
```
inspxe-cl -help
```

## Set up command line with GUI

Command Line...

## Command examples:

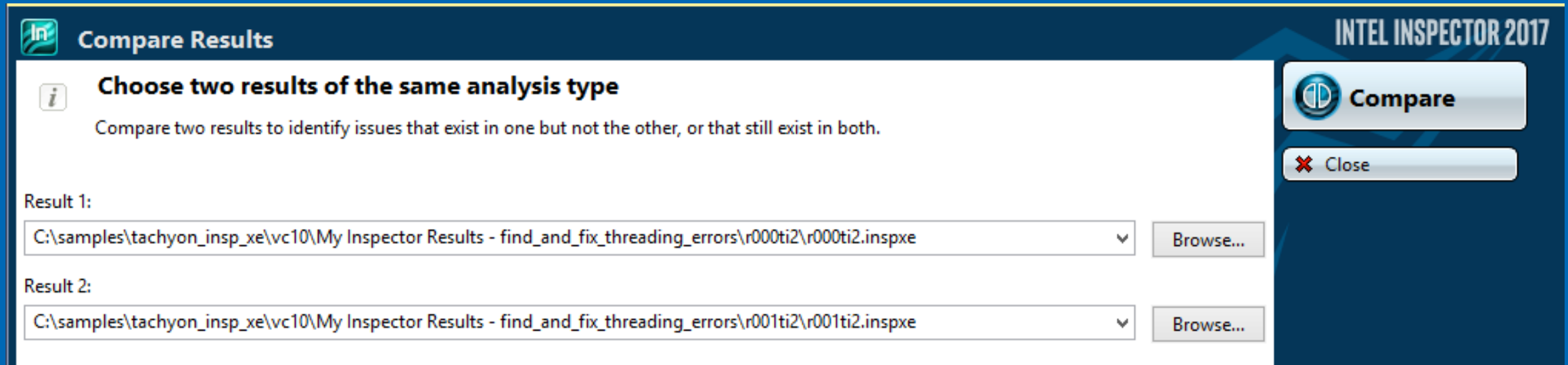
1. `inspxe-cl -collect-list`
2. `inspxe-cl -collect ti2 -- MyApp.exe`
3. `inspxe-cl -report problems`



**Send results file to developer to analyze with the UI**

# Compare results and see what has changed

Ideal for regression testing



**Compare Results**

**Choose two results of the same analysis type**  
Compare two results to identify issues that exist in one but not the other, or that still exist in both.

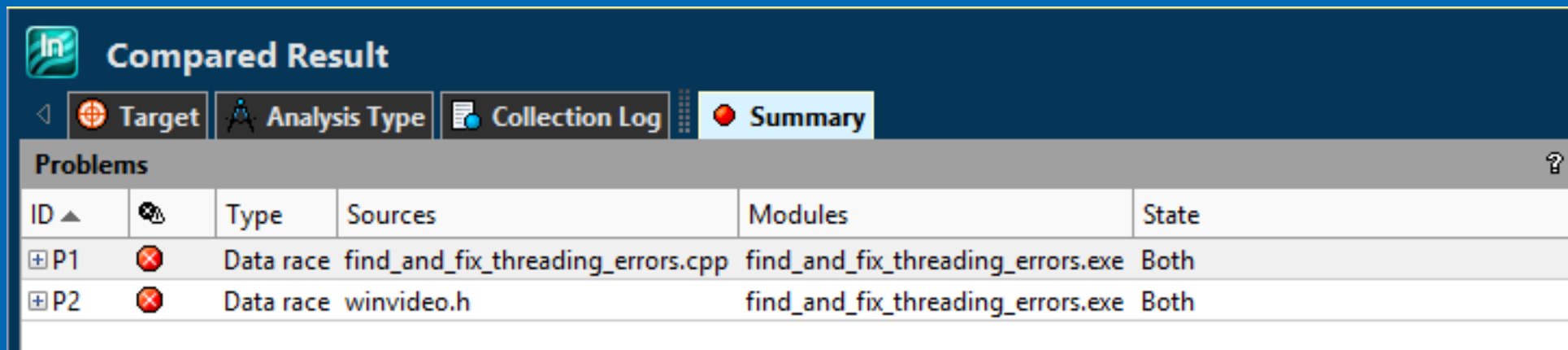
Result 1:  
C:\samples\tachyon\_insp\_xe\vc10\My Inspector Results - find\_and\_fix\_threading\_errors\r000ti2\r000ti2.inspxe Browse...

Result 2:  
C:\samples\tachyon\_insp\_xe\vc10\My Inspector Results - find\_and\_fix\_threading\_errors\r001ti2\r001ti2.inspxe Browse...

**INTEL INSPECTOR 2017**

**Compare**

**Close**



**Compared Result**

Target Analysis Type Collection Log **Summary**

**Problems**

ID	Type	Sources	Modules	State
P1	Data race	find_and_fix_threading_errors.cpp	find_and_fix_threading_errors.exe	Both
P2	Data race	winvideo.h	find_and_fix_threading_errors.exe	Both

# Find problems quicker!

Interactive debugging support

**Configure Analysis Type**

**Analysis Type**

Threading Error Analysis

10x-40x Detect Deadlocks

20x-80x Detect Deadlocks and Data Races

40x-160x Locate Deadlocks and Data Races

Analysis Time Overhead

Memory Overhead

**Detect Deadlocks and Data Races** Copy

Medium scope threading error analysis type. Increases the load on the system and the time and resources required to perform analysis. Press F1 for more details.

Terminate on deadlock

Stack frame depth: 1

Analyze without debugger  
Run an analysis and report all detected problems. Use to view correctness issues without stopping in the debugger

Enable debugger when problem detected  
Run an analysis under the debugger and stop every time a problem is detected. Use to allow investigation of every

Select analysis start location with debugger  
Run target application under the debugger with analysis disabled until you choose to turn on analysis. Before starti

Details

INTEL INSPECTOR

Start

Stop

Close

Reset Growth Tracking

Measure Growth

Reset Leak Tracking

Find Leaks

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

Problems		
ID ▲	Type	Sources
P1	Mismatched allocation/deall...	
P2	Memory leak	
P3	Invalid memory access	
	Invalid memory access	
P4	Memory growth	
P5	Memory growth	
P6	Memory growth	

- View Source
- Edit Source
- Copy to Clipboard
- Explain Problem
- Create Problem Report...
- Debug This Problem

## Threading Errors

Problems			
ID ▲	Type	Sources	Modules
P1	Data race	winvideo.h	
	Data race	winvideo.h:270	
	Data race	winvideo.h:270	
	Data race	winvideo.h:201	
	Data race	winvideo.h:202	
	Data race	winvideo.h:202	

- View Source
- Edit Source
- Copy to Clipboard
- Explain Problem
- Create Problem Report...
- Debug This Problem

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<sup>®</sup> 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<sup>®</sup> 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.

The Intel logo is centered in the upper half of the image. It features the word "intel" in a white, lowercase, sans-serif font. A small blue square is positioned above the letter "i". To the right of the word "intel" is a registered trademark symbol (®). The background is a blue-tinted photograph of server racks in a data center.

intel®

SPDK, PMDK, Intel® Performance Analyzers

**Virtual Forum**