



Intel® Inspector XE 2015

Memory and thread debugger



Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

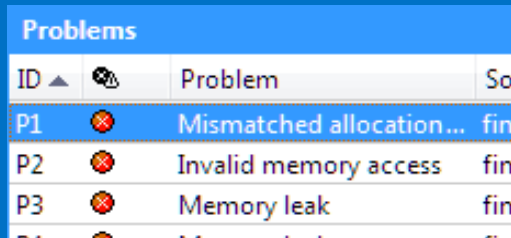
User API

Using the Intel® Inspector XE with MPI

Summary

Motivation for The Inspector XE

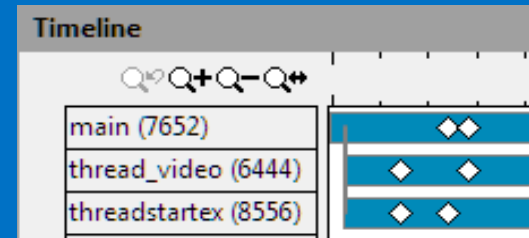
Memory Errors



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

- Invalid Accesses
- Memory Leaks
- Uninitialized Memory Accesses

Threading Errors



- Data Races
- Deadlocks
- Cross Stack References

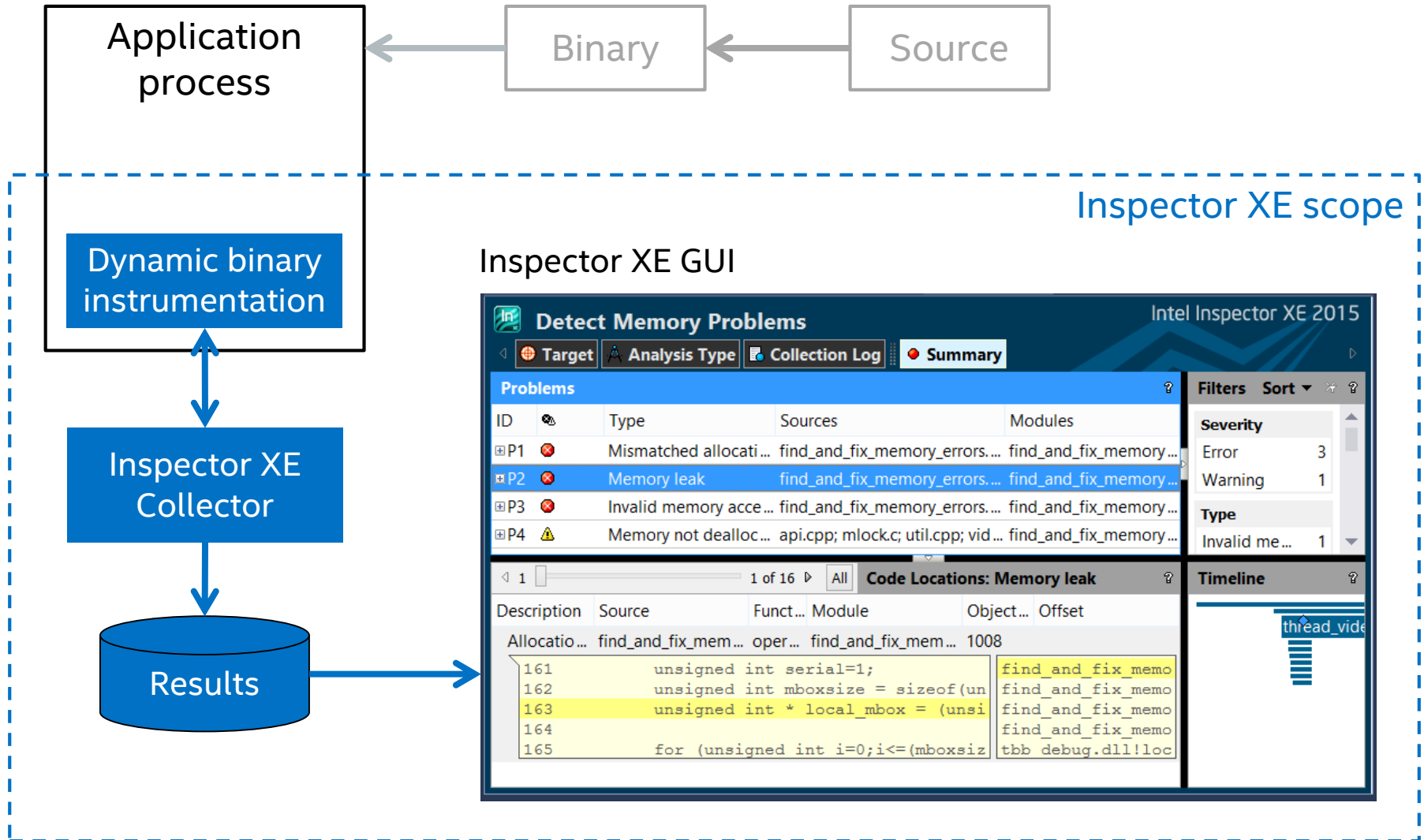
Multi-threading problems

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



Let the tool do it for you

Intel Inspector XE: Dynamic analysis



How it looks: Visual Studio* Integration

tachyon_insp_xe - Microsoft Visual Studio (Administrator)

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP

Local Windows Debugger Auto Debug

Solution Explorer

Search Solution Explorer

find_and_fix_threading_errors.cpp r000ti2 r001mi2

Detect Memory Problems

Target Analysis Type Collection Log Summary

Problems

ID	Type	Modules
P1	Mismatched allocati...	find_and_fix_memory...
P2	Memory leak	find_and_fix_memory...
P3	Invalid memory acce...	find_and_fix_memory...
P4	Memory not dealloc...	api.cpp; mlock.c; util.cpp; vid...

Filters Sort

Severity

Error	3
Warning	1

Type

Invalid me...	1
---------------	---

Timeline

thread_vide

Choose existing project, no special configuration

Run analysis from toolbar

Problems found: memory leaks

Memory allocation site in source code

Call stack

```
161 unsigned int serial=1;
162 unsigned int mboxsize = sizeof(un
163 unsigned int * local_mbox = (unsi
164
```

Code Locations: Memory leak

1 of 16 All

Allocatio... find_and_fix_mem... oper... find_and_fix_mem... 1008

find_and_fix_memo
find_and_fix_memo
find_and_fix_memo
find_and_fix_memo
debug.dll!loc

Standalone GUI: Windows* and Linux*

The screenshot displays the Intel Inspector XE 2015 standalone GUI. The main window is titled "C:\Temp\My Inspector XE Results - Intel Inspector". The interface includes a menu bar (File, View, Help), a Project Navigator on the left showing a tree view of "My Inspector XE" with sub-items "r000ti2", "r001ti2", "r002ti2", and "r003ti2", and a main configuration area titled "Configure Analysis Type".

The "Configure Analysis Type" dialog is open, showing the "Analysis Type" dropdown set to "Memory Error Analysis". The "Analysis Time Overhead" slider is set to "10x-40x" and the "Memory Overhead" slider is set to "20x-80x". The "Detect Memory Problems" section is expanded, showing a description: "Medium scope memory error analysis type. Increases the load on the system and the time and resources required to perform analysis. Press F1 for more details." Below this, several checkboxes are visible:

- Detect uninitialized memory reads
- Revert to previous uninitialized memory algorithm (not recommended)
- Detect memory leaks upon application exit
- Detect resource leaks
- Enable interactive memory growth detection

The "Start" button is highlighted in the top right. The "Project Properties..." and "Command Line..." buttons are also visible in the bottom right.

Key Features at a glance

Feature	Details
Data collection	<ul style="list-style-type: none">• Dynamic Memory and Threading Analysis (including .NET* analysis)• MPI applications analysis
Result analyses	<ul style="list-style-type: none">• GUI data mining: source code analysis, filtering, exploring call paths, etc.• Debugger integration• Result comparison• Problem life cycle management• Command line interface (especially useful for regression testing)
GUI	<ul style="list-style-type: none">• Microsoft* Visual Studio IDE integration (2010, 2012 and 2013)• Stand alone GUI on both Windows* and Linux*
Compilers supported	<ul style="list-style-type: none">• Microsoft* Visual* C++ and .NET*• Intel® C/C++ Compiler XE 12.0 or higher• Intel® Visual Fortran Compiler XE 12.0 or higher• gcc
OS	<ul style="list-style-type: none">• Windows* 7, 8, 8.1,• Windows* Server 2008, 2008 R2, 2012• Linux*: RedHat, Fedora, CentOS, SUSE, Debian, Ubuntu
Languages	<ul style="list-style-type: none">• C/C++• C# (.NET 2.0 to 3.5, .NET 4.0 with limitations)• Fortran

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

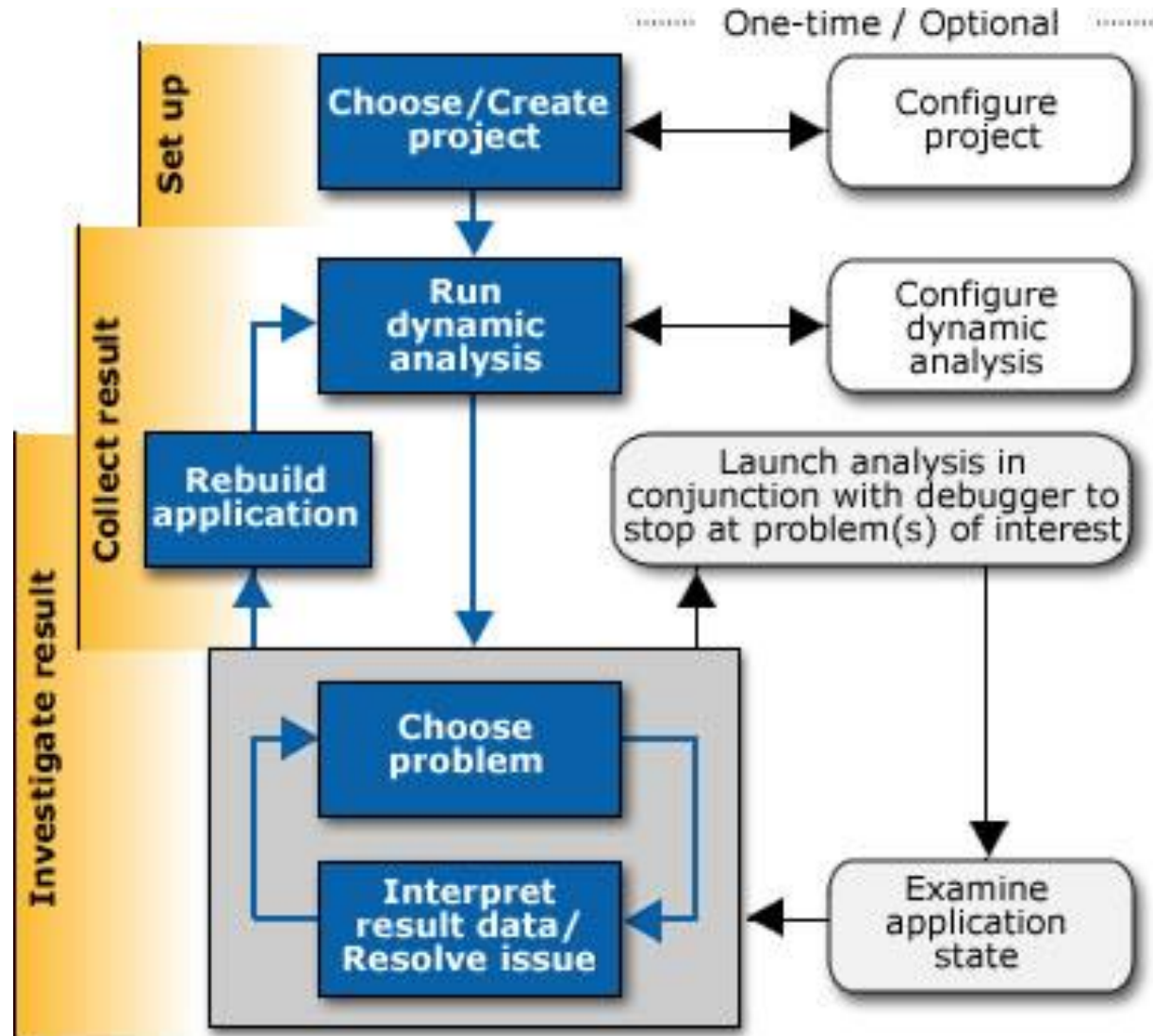
Automated regression testing

User API

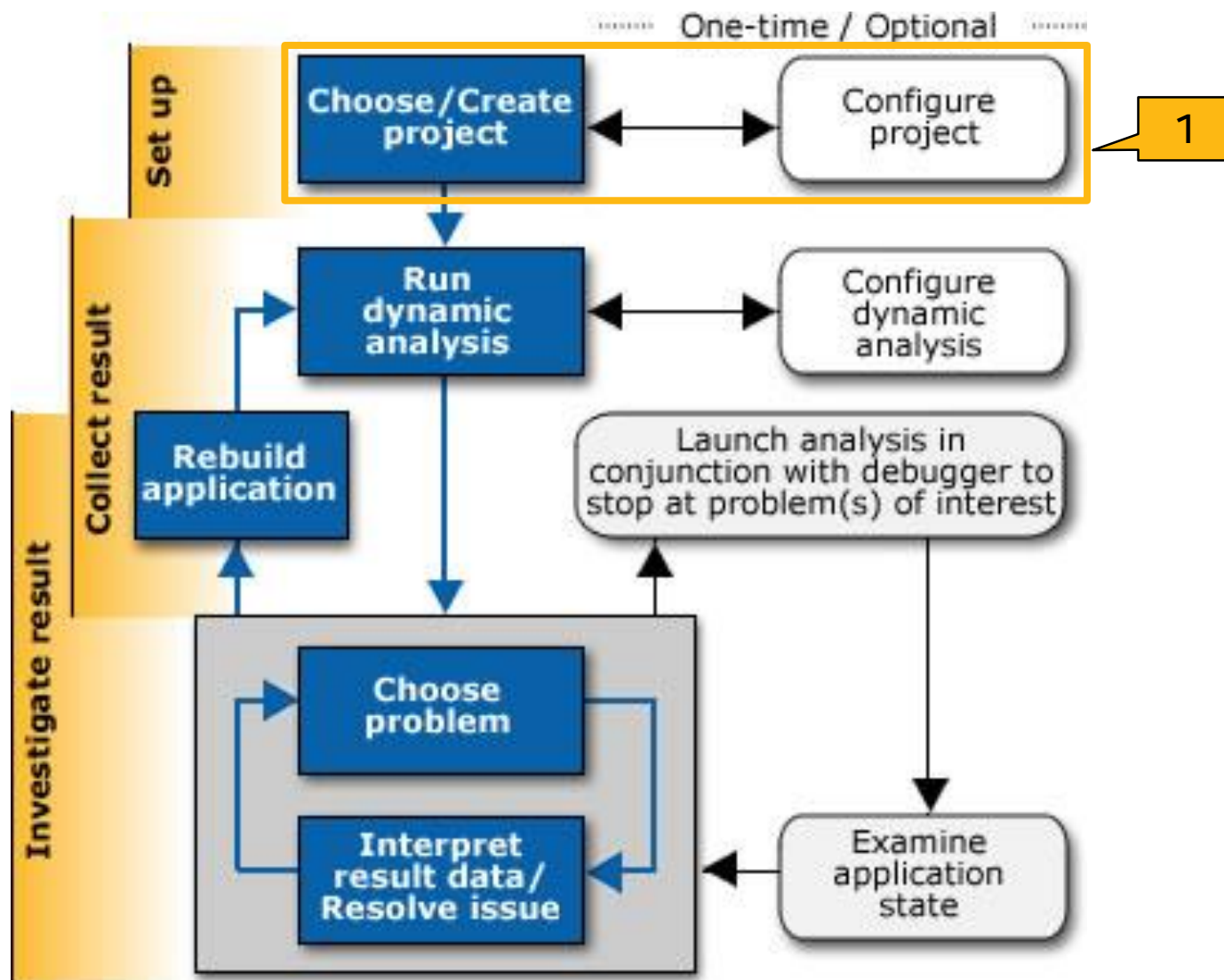
Using the Intel® Inspector XE with MPI

Summary

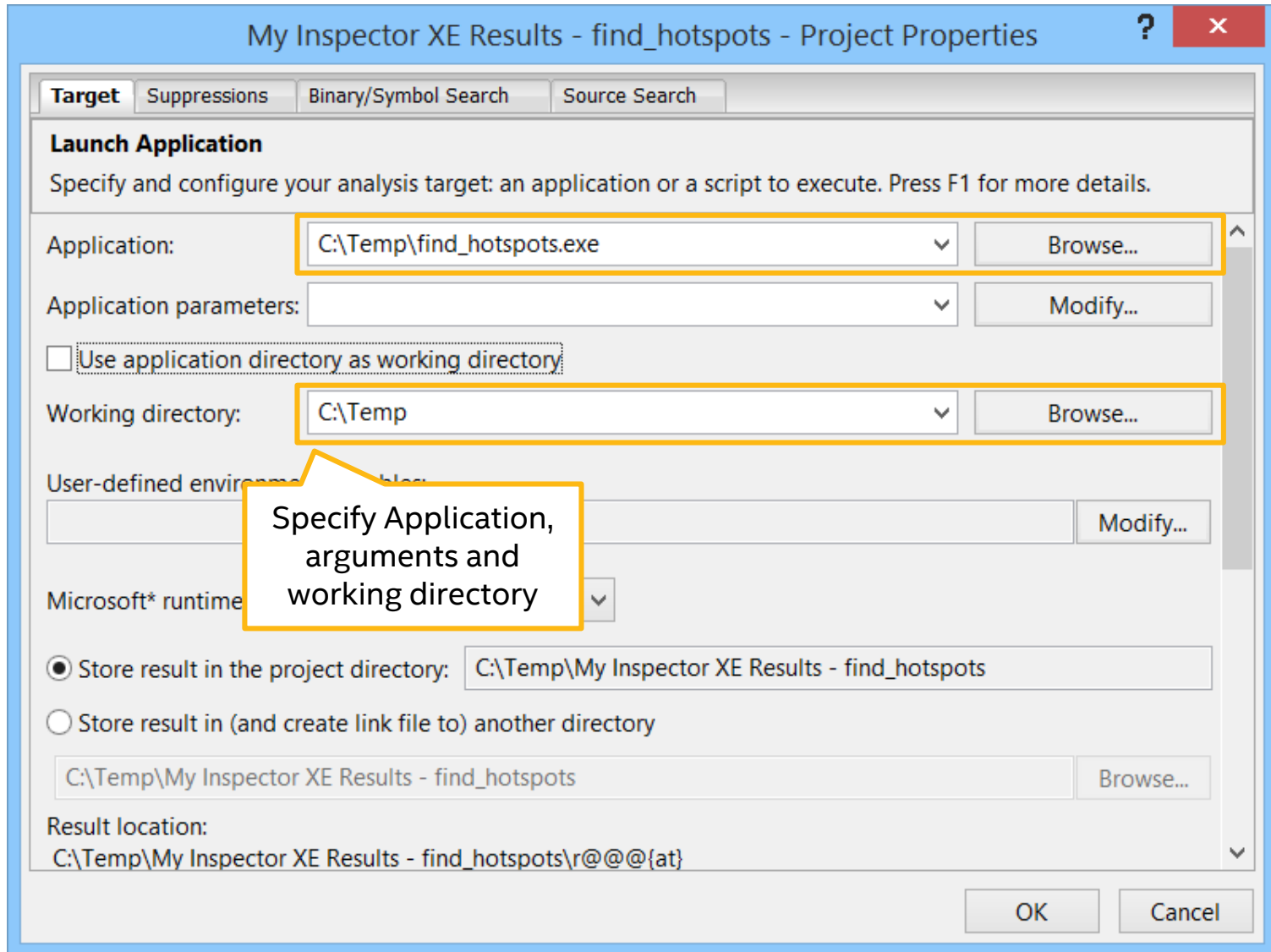
Workflow: Dynamic Analysis



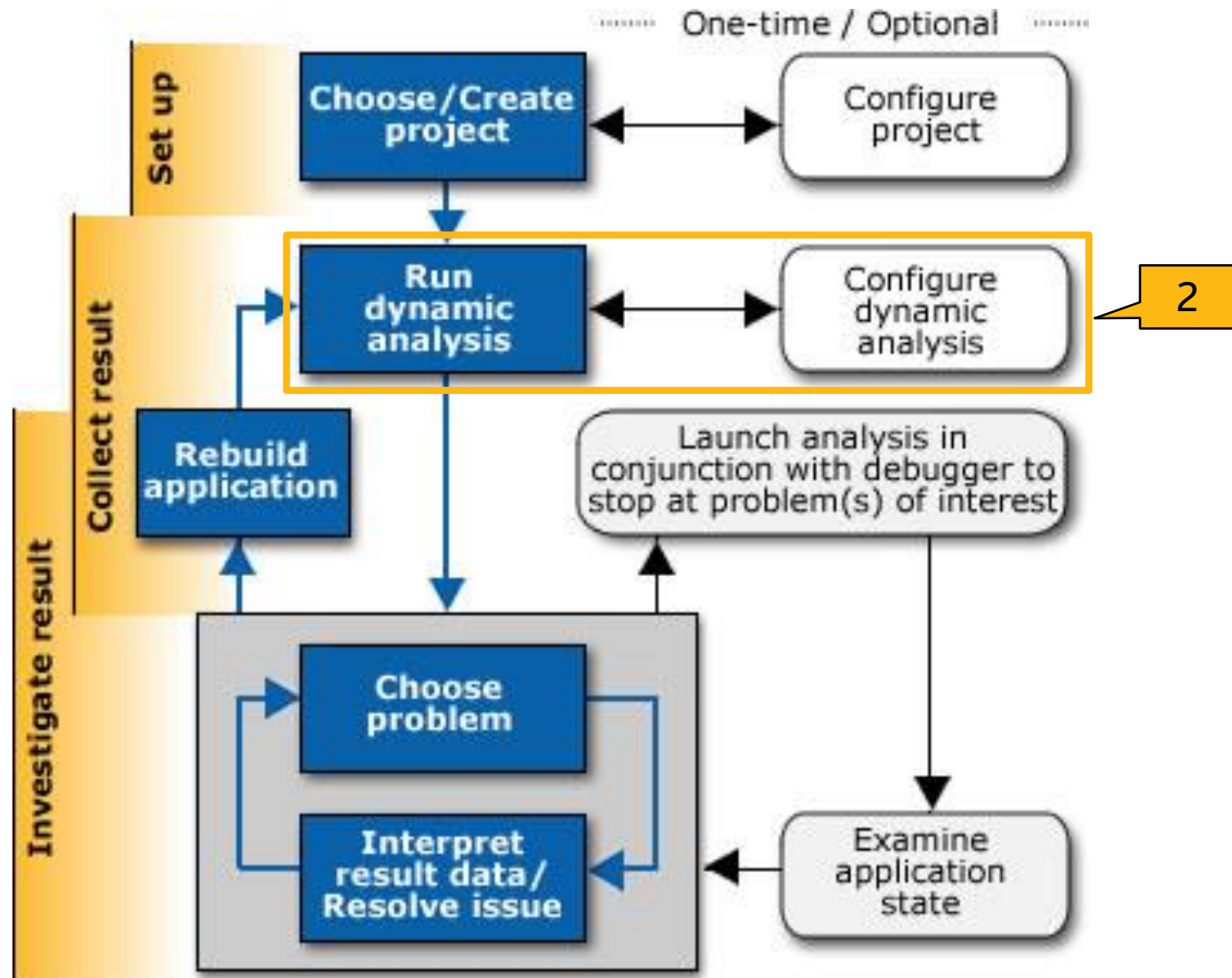
Workflow: Dynamic Analysis



Workflow: setup project



Workflow: Dynamic Analysis



Workflow: select analysis and start

Configure Analysis Type Intel Inspector XE 2015

Analysis Type

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

Analysis Time Overhead	Memory Overhead
10x-40x	Detect Deadlocks
20x-80x	Detect Deadlocks and Data Races
40x-160x	Locate Deadlocks and Data Races

Locate Deadlocks and Data Races Copy

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

Start

Stop

Close

Reset Growth Tracking

Measure Growth

Reset Leak Tracking

Find Leaks

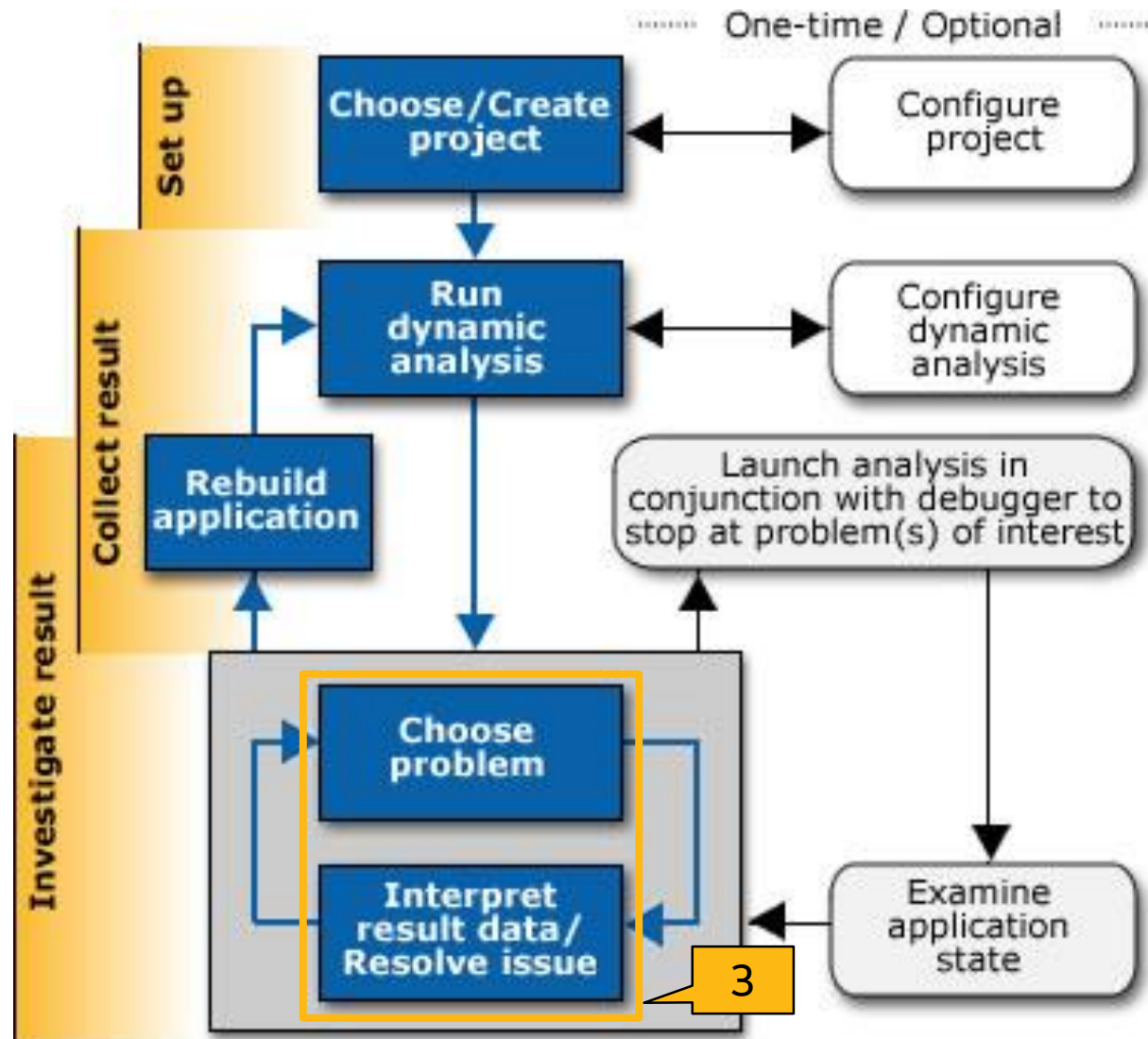
Project Properties...

Command Line...

1. Select Analysis Type

2. Click Start

Workflow: Dynamic Analysis



Workflow: manage results

Intel Inspector XE 2015

Detect Deadlocks and Data Races

Target | Analysis Type | Collection Log | Summary

Problems

ID	Type	File	Module	Status
P1	Data race	find_and_fix_threading_errors.cp...	find_and_fix_threading_errors.exe	New
P2	Data race	winvideo.h	find_and_fix_threading_errors.exe	New
	Data race	winvideo.h:270	find_and_fix_threading_errors.exe	New
	Data race	winvideo.h:270	find_and_fix_threading_errors.exe	New
	Data race	winvideo.h:201; winvideo.h:270	find_and_fix_threading_errors.exe	New

Code Locations: Data race

Read winvideo.h:270 next_frame find_and_fix_threading_errors.exe

```
268 {
269     if(!running) return false;
270     g_updates++; // Fast but inaccura
271     if(!threaded) while(loop_once(thi
272     else if(g handles[1]) {
```

Timeline

- main (4960)
- thread_video (4672)
- TBB Worker Thread (2848)
- TBB Worker Thread (1724)
- TBB Worker Thread (6004)

Read: winvideo.h:270
Write: winvideo.h:270

Annotations:

- Double click on Problem to navigate to source
- Powerful filtration feature
- Code locations grouped into Problems to simplify results management

Workflow: navigate to sources

Data race

Intel Inspector XE 2015

Target Analysis Type Collection Log Summary Sources

Write - Thread TBB Worker Thread (1724) (find_and_fix_threading_errors.exe!next_frame - winvideo.h:270)

```
winvideo.h Disassembly (find_and_fix_threading_errors.exe!0x9257) Call Stack
267 bool video::next_frame()
268 {
269     if(!running) return false;
270     g_updates++; // Fast but inaccurate counter. The data race h
271     if(!threaded) while(loop_once(this));
272     else if(g_handles[1]) {
273         SetEvent(g_handles[1]);
274         YIELD_TO_THREAD();
275     }
```

Problematic line in source code

Call stacks

Read - Thread TBB Worker Thread (6004) (find_and_fix_threading_errors.exe!next_frame - winvideo.h:270)

```
winvideo.h Disassembly (find_and_fix_threading_errors.exe!0x924e) Call Stack
267 bool video::next_frame()
269     if(!running) return false;
270     g_updates++; // Fast but inaccur
271     if(!threaded) while(loop_once(th
272     else if(g_handles[1]) {
273         SetEvent(g_handles[1]);
274         YIELD_TO_THREAD();
275     }
```

All code locations for a problem

Switch to disassembly for more details

Workflow: timeline view

The screenshot displays the Intel Inspector XE 2015 interface. At the top, the title bar reads "Detect Deadlocks and Data Races" and "Intel Inspector XE 2015". Below the title bar are navigation buttons: "Target", "Analysis Type", "Collection Log", and "Summary".

The "Problems" table lists two data race issues:

ID	Type	Sources	Modules	State
P1	Data race	find_and_fix_threading_errors.cp...	find_and_fix_threading_errors.exe	New
P2	Data race	winvideo.h	find_and_fix_threading_errors.exe	New

A yellow callout box points to the "Sources" column of the second problem, stating: "Individual Code Locations are seen in Timeline view in the context of their respective threads".

The "Timeline" view is shown on the right, listing threads: "main (4960)", "thread_video (4672)", "TBB Worker Thread (2848)", "TBB Worker Thread (1724)", and "TBB Worker Thread (6004)". A yellow callout box points to the "TBB Worker Thread (6004)" entry, stating: "Hover gives details".

The "Code Locations" view is shown on the left, displaying the source code for the data race. The code is as follows:

```
268 {
269     if(!running) return false;
270     g_updates++; // Fast but i
271     if(!threaded) while(loop_c
272     else if(g_handles[1]) {
```

The "Code Locations" table has the following columns: "Description", "Source", "Function", and "Module". The "Description" column shows "Read" and "Write" operations. The "Source" column shows "winvideo.h:270 next_frame". The "Function" column shows "find_and_fix_threading_errors.exe". The "Module" column shows "find_and_fix_threading_errors.exe".

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Memory problem Analysis

Analyzed as software runs

- Data (workload) -driven execution
- Program can be single or multi-threaded
- Diagnostics reported incrementally as they occur

Includes monitoring of:

- Memory allocation and allocating functions
- Memory deallocation and deallocating functions
- Memory leak reporting
- Inconsistent memory API usage

Analysis scope

- Native code only: C, C++, Fortran
- Code path must be executed to be analyzed
- Workload size affects ability to detect a problem

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

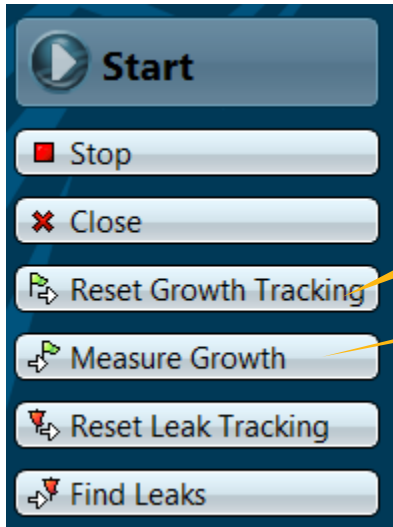
- GDI object is created but never deleted

```
// GDI Resource Leak

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

Analyze Memory Growth

During Analysis:



Set Start Point

Set End Point

Analysis Results:

Memory Growth Problem Set

Code location for each block of memory that was allocated but not de-allocated during the time period

Detect Memory Problems

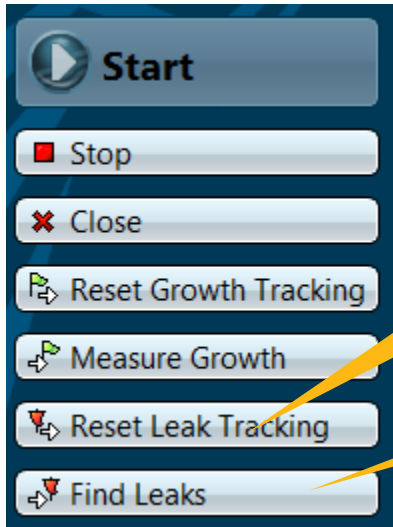
Target Analysis Type Collection Log Summary

Problems						
I	Type	Sources	Modules	Object ...	State	
P	Memory leak	ixe_mem_growth.cpp	ixe_mem_growth.e...	144	New	
P	Memory growth	[Unknown]; ixe_mem_gro...	Unknown; ixe_mem...	272	New	
	Start memory growth det...	[Unknown]	Unknown		Not fixed	
	Memory growth	ixe_mem_growth.cpp:7	ixe_mem_growth.e...	272	New	
	End memory growth det...	[Unknown]	Unknown		Not fixed	

Code Locations: Memory growth

Description	Source	Function	Module	Object Size	Offset
Allocation site	ixe_mem_growth.cpp:7	transaction	ixe_mem_growth.exe	272	
5	{				ixe_mem_growth.exe!transaction
6	char *str;				ixe_mem_growth.exe!main - ixe_m
7	str = (char*) malloc(16);				ixe_mem_growth.exe!_tmainCRTSta
8	}				ixe_mem_growth.exe!mainCRTStart
9					kernel32.dll!BaseThreadInitThun

On-demand leak detection



Set Start Point

Set End Point

- Check code regions between points 'A' and 'B' for leaks
- Check daemon processes for leaks
- Check crashing processes for leaks

Analysis Results:

Memory Leak shown during run time

Detect Memory Problems

Target Analysis Type Collection Log Summary

ID	Type	Sources	Modules	Object Size	State
P1	Memory leak	ixe_mem_growth.cpp	ixe_mem_growth.exe	192	New
	Memory leak	ixe_mem_growth.cpp:7	ixe_mem_growth.exe	192	New
P2	Memory growth [Unknown];	ixe_mem_gr...	Unknown; ixe_mem_gr...	368	New

Code Locations: Memory leak

Description	Source	Function	Module	Object Size	Offset
Allocation site	ixe_mem_growth.cpp:7	transaction	ixe_mem_growth.exe	192	
5	{		ixe_mem_growth.exe!transaction		
6	char *str;		ixe_mem_growth.exe!main - ixe_		
7	str = (char*) malloc(16);		ixe_mem_growth.exe!_tmainCRTSt		
8			ixe_mem_growth.exe!mainCRTStar		
9	malloc(4);		kernel32.dll!BaseThreadInitThu		

Define analysis scope in source code

```
#include <ittnotify.h>

void ProcessPipeline()
{
    __itt_heap_reset_detection(__itt_heap_leaks); // Start measuring memory leaks
    pipeline_stage1(); // Run pipeline stage 1
    __itt_heap_record(__itt_heap_leaks); // Report leaks in stage 1

    DoSomeOtherWork();

    __itt_heap_reset_detection(__itt_heap_growth); // Start measuring memory growth
    pipeline_stage2(); // Run pipeline stage 2
    __itt_heap_record(__itt_heap_growth); // Report memory growth in stage 2
}
```


Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- **Lab 1. Finding memory errors**

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

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

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);
```

Deadlock

```
CRITICAL_SECTION cs1;  
CRITICAL_SECTION cs2;  
int x = 0;  
int y = 0;  
InitializeCriticalSection(&cs1); // Allocation Site (cs1)  
InitializeCriticalSection(&cs2); // Allocation Site (cs2)
```

Thread #1

1

```
EnterCriticalSection(&cs1);  
x++;  
2 EnterCriticalSection(&cs2);  
y++;  
LeaveCriticalSection(&cs2);  
LeaveCriticalSection(&cs1);
```

2

Thread #2

```
EnterCriticalSection(&cs2);  
y++;  
EnterCriticalSection(&cs1);  
x++;  
LeaveCriticalSection(&cs1);  
LeaveCriticalSection(&cs2);
```

Deadlock
Lock Hierarchy Violation

Deadlock

```
CRITICAL_SECTION cs1;  
CRITICAL_SECTION cs2;  
int x = 0;  
int y = 0;  
InitializeCriticalSection(&cs1); // Allocation Site (cs1)  
InitializeCriticalSection(&cs2); // Allocation Site (cs2)
```

Thread #1

```
EnterCriticalSection(&cs1);  
x++;  
    EnterCriticalSection(&cs2);  
    y++;  
    LeaveCriticalSection(&cs2);  
LeaveCriticalSection(&cs1);
```

Thread #2

```
EnterCriticalSection(&cs2);  
y++;  
    EnterCriticalSection(&cs1);  
    x++;  
    LeaveCriticalSection(&cs1);  
LeaveCriticalSection(&cs2);
```

Deadlock

1. EnterCriticalSection(&cs1); in thread #1
2. EnterCriticalSection(&cs2); in thread #2

Lock Hierarchy Violation

1. EnterCriticalSection(&cs1); in thread #1
2. EnterCriticalSection(&cs2); in thread #1
3. EnterCriticalSection(&cs2); in thread #2
4. EnterCriticalSection(&cs1); in thread #2

Cross-thread Stack Access

```
// A pointer visible for two threads
int *p;
CreateThread(..., thread #1, ...);
CreateThread(..., thread #2, ...);
```

Thread #1

```
// Allocated on Thread #1's stack
int q[1024];
p = q;
q[0] = 1;
```

Thread #2

```
// Thread #1's stack accessed
*p = 2;
```

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- **Lab 2. Finding threading errors**

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Prepare build for analysis

Compile

- Use dynamically linked thread-safe runtime libraries
`/MDd` on Windows
- Generate symbolic information
`/ZI` on Windows
- Disable optimization
`/Od` on Windows

Link

- Preserve symbolic information
`/DEBUG` on Windows
- Specify relocatable code sections
`/FIXED:NO` on Windows

Prior to using Inspector XE, sources should compile & link cleanly

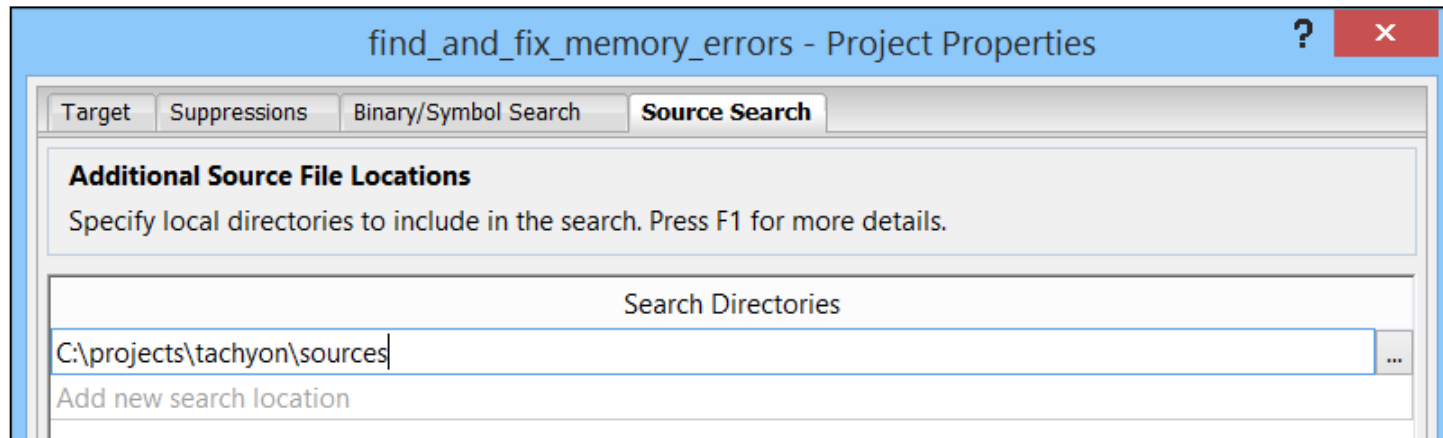
Search directories

Inspector XE needs to locate paths to:

- Binary files
- Symbol files
- Source files

No need for extra search directories configuration if:

- Binary, symbol and source files were not modified and moved
- Results are collected and viewed on the same machine



Correctness analyses overhead

Inspector XE tracks

- Thread and Sync APIs
- Memory accesses

Inspector XE performs binary instrumentation using PIN

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



Increases **execution time** and **memory consumed** (potentially significantly)

The Inspector XE dilates both time and memory consumed significantly!

Workload guidelines

Use small data set

- Smaller number of threads
- Minimize data set size (e.g. smaller image sizes)
- Minimize loop iterations or time steps
- Minimize update rates (e.g. lower frames per second)

Use small but representative data set

- Only **actually executed** code paths are analyzed

Scale down workload to speed up analysis!

Analysis scope guidelines

There is a trade-off between analysis speed and thoroughness

- Low level of analysis implies high speed and missing problems
- Start with low level analysis, then increase thoroughness, e.g.:
 1. Detect Leaks
 2. Detect Memory Problems
 3. Locate Memory Problems

Limit analysis scope

- Exclude unnecessary modules by configuring analysis
- Use collection control API

Scale down workload to speed up analysis!

Include and Exclude modules

The screenshot shows the 'Project Properties' dialog box for 'My Inspector XE Results - find_and_fix_threading_errors'. The 'Advanced' section is expanded, showing options for 'Suppressions' and 'Modules'. The 'Modules' section has two radio buttons: 'Include only the following module(s)' and 'Exclude the following module(s)'. The 'Exclude the following module(s)' option is selected. A 'Modify...' button is visible next to the 'Modules' list. An 'Edit Module' dialog box is open, showing a list of modules with 'C:\home\tbb_debug.dll' selected. A 'Delete' button is visible in the 'Edit Module' dialog. Three yellow callout boxes provide instructions: 1. There are two options: - Include modules of interest - Exclude unnecessary modules; 2. Press Modify; 3. Choose modules you want to include or exclude from analysis.

1. There are two options:

- Include modules of interest
- Exclude unnecessary modules

2. Press Modify

3. Choose modules you want to include or exclude from analysis

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Filtering - focus on what is important

Problems

I ▲	Type	Sources	Modules	Object...	St.
P1	Mismatched allocation/de...	find_and_fix_memory...	find_and_fix_mem...		
P2	Memory leak	find_and_fix_memory...	find_and_fix_mem...	28672	
P3	Invalid memory access	find_and_fix_memory...	find_and_fix_mem...		
P4	Memory not deallocated	api.cpp; util.cpp; vide...	find_and_fix_mem...	10376	
	Memory not deallocated	video.cpp:82	find_and_fix_mem...	8192	
	Memory not deallocated	util.cpp:163	find_and_fix_mem...	1808	
	Memory not deallocated	api.cpp:218	find_and_fix_mem...	376	

Filter - Show only one source file

Filters Sort ▼ ✖ ?

Source

- api.cpp 1
- find_and_fix_memory_error... 3
- util.cpp 1
- video.cpp 1

Module

- find_and_fix_memory_error... 4

State

Problems

I ▲	Type	Sources	Modules	Object...	St.
P4	Memory not deallocated	api.cpp; util.cpp; vide...	find_and_fix_mem...	10376	
	Memory not deallocated	video.cpp:82	find_and_fix_mem...	8192	
	Memory not deallocated	util.cpp:163	find_and_fix_mem...	1808	
	Memory not deallocated	api.cpp:218	find_and_fix_mem...	376	

Only related errors are shown

Filters Sort ▼ ✖ ?

Warning 1

Type

- Memory not deallocated 1

Source All

- api.cpp 1 item(s)

Module

- find_and_fix_memory_error... 1

State

Suppressions: manage false errors

The image shows three overlapping windows from Visual Studio illustrating how to create and manage suppressions:

- Problems window:** Shows a list of errors. A yellow callout points to the 'Memory leak' error, stating: "Suppressions are marked or hidden entirely".
- Create Suppression dialog:** Shows the 'Name' field set to 'Suppression' and the 'Problem type' dropdown set to 'Memory leak'. A yellow callout points to the dropdown, stating: "Choose problem type".
- Select Stack Frame(s) dialog:** Shows a table of call stack frames. A yellow callout points to the first row, stating: "Choose stack frames to match the rule".

Use in Rule	Module	Function	Source	Line
<input checked="" type="checkbox"/>	ixe_mem_grow...	transaction	ixe_mem_gro ...	7
<input type="checkbox"/>	ixe_mem_grow...	main	ixe_mem_gro ...	16
<input type="checkbox"/>	ixe_mem_grow...	_tmainCRTStart...	crtexe.c	555
<input type="checkbox"/>	ixe_mem_grow...	mainCRTStartup	crtexe.c	370
<input type="checkbox"/>	kernel32.dll	BaseThreadInit...	* (any)	* (any)

- Suppressions are saved in one or more files
- Tool suppresses all files from specified folder(s)

Suppressions: extended features

Import suppressions from third party tools:

Valgrind* and Rational Purify*

```
inspxe-cl -convert-suppression-file -  
from=known_problems.pft -to=known_problems.sup
```

User editable suppressions

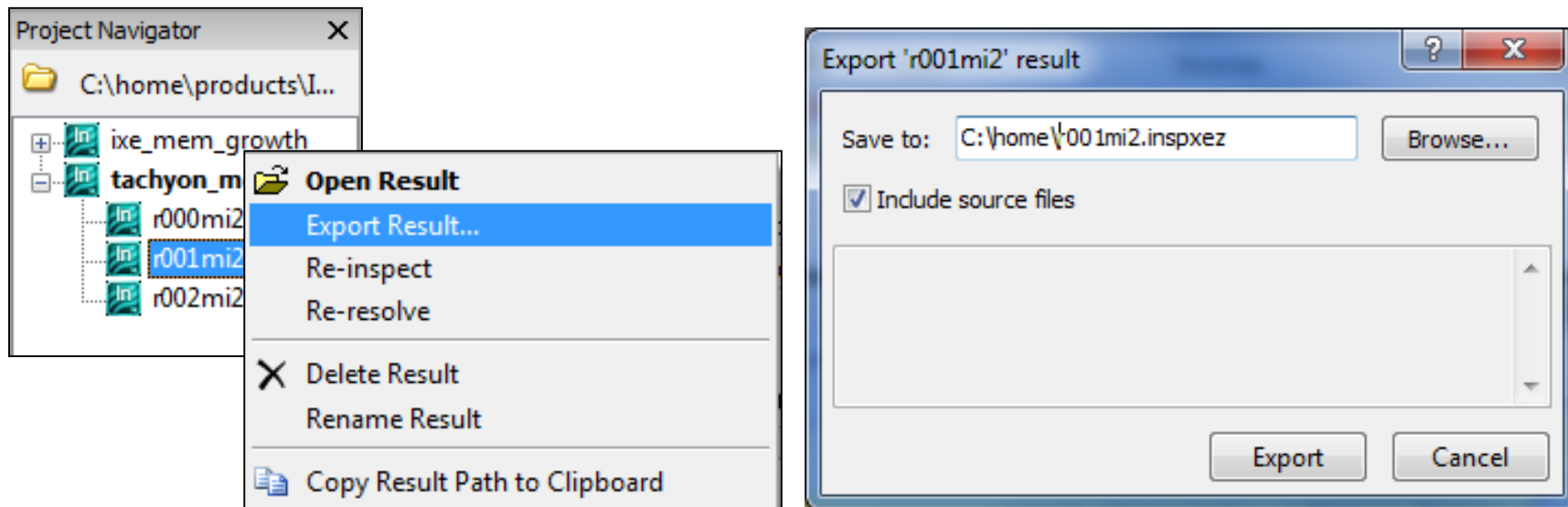
```
suppression = {  
    name = "Suppress all diagnostics on memory  
           which has been allocated by malloc() in alloc.c";  
    type = { reachable_leak }  
    stacks = {  
        allocation = {  
            func=malloc, src=alloc.c;  
        }  
    }  
}
```

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:



Problem State Lifecycle

Problems						
ID ▲		Problem	Sources	Modules	Obj...	State
⊕ P1	✖	Invalid memo ...	find_and_fi ...	find_and_ ...		Not fixed
⊕ P2	⚠	Memory not ...	api.cpp; uti ...	find_and_ ...		Not fixed
⊕ P3	✖	Memory leak	find_and_fi ...	find_and_ ...	1232	Not fixed
⊕ P4	✖	Memory leak	find_and_fi ...	find_and_ ...	672	Not fixed
⊕ P5	✖	Memory leak	find_and_fi ...	find_and_ ...	672	Not fixed

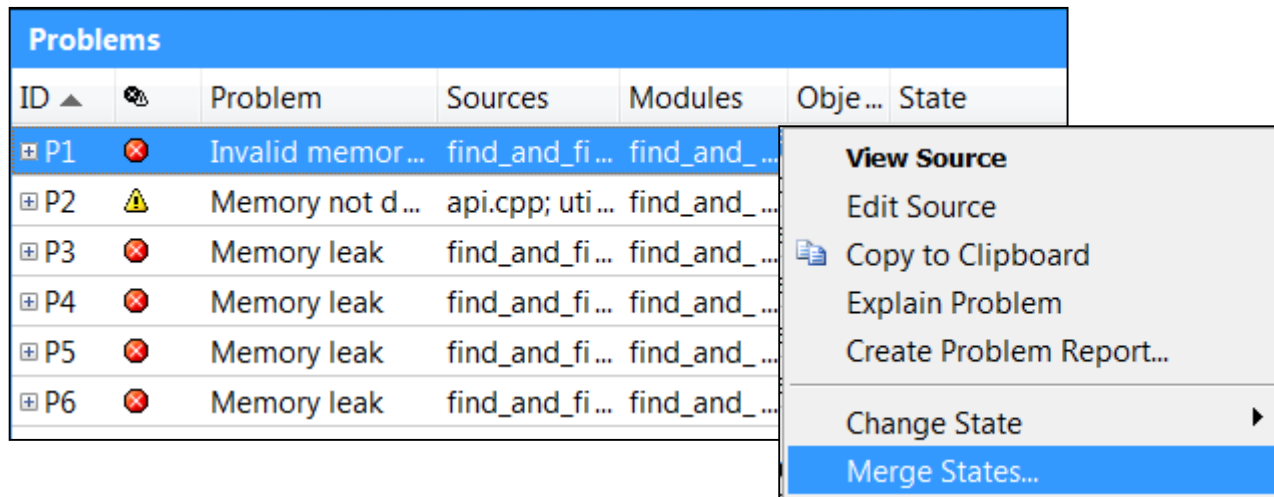
- View Source**
- Edit Source
- Copy to Clipboard
- Explain Problem
- Create Problem Report...
- Change State ▶
- Merge States...

- Not fixed
- Confirmed
- Fixed
- Not a problem
- Deferred

State	Description
New	Detected by this run
Not Fixed	Previously seen error detected by this run
Not a Problem	Set by user
Confirmed	Set by user
Fixed	Set by user
Regression	Error detected with previous state of "Fixed"
Deferred	Set by user

Merge arbitrary results

- Merge states from another result to current one
- Incorporate states from other users

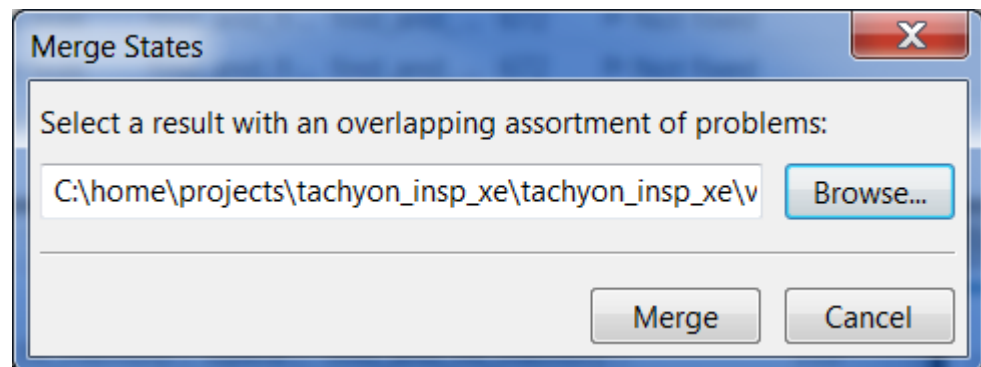


The screenshot shows a 'Problems' window with a table of error messages. A context menu is open over the first row (P1), with 'Merge States...' highlighted. The table has columns for ID, Problem, Sources, Modules, and State.

ID	Problem	Sources	Modules	State
P1	Invalid memor...	find_and_fi...	find_and_...	
P2	Memory not d...	api.cpp; uti...	find_and_...	
P3	Memory leak	find_and_fi...	find_and_...	
P4	Memory leak	find_and_fi...	find_and_...	
P5	Memory leak	find_and_fi...	find_and_...	
P6	Memory leak	find_and_fi...	find_and_...	

Context menu options:

- View Source
- Edit Source
- Copy to Clipboard
- Explain Problem
- Create Problem Report...
- Change State
- Merge States...



The 'Merge States' dialog box prompts the user to select a result with an overlapping assortment of problems. It shows a file path and buttons for 'Merge' and 'Cancel'.

Select a result with an overlapping assortment of problems:

C:\home\projects\tachyon_insp_xe\tachyon_insp_xe\v

Team collaboration

Problem reports

- Plain text reports with stacks and source snippets
- Exported Inspector XE results

Share suppression files with the team

Automated regression testing

The screenshot displays the Intel Memory Problem Detector interface. The main window shows a list of detected memory leaks under the 'Problems' tab. A 'Problems Report' dialog box is open, providing a detailed view of problem P105.

ID	Problem	Sources	Modules	Object Size	State
P105	Memory leak	Window...	StreamingTestApplicatio...	520	New
P106	Memory leak	Streamin...	StreamingViewport.dll	19292	New
P107	Memory leak	Streamin...	StreamingViewport.dll	36	New
P108	Memory leak	Streamin...	StreamingViewport.dll	8556	New
P109	Memory leak	stream...	StreamingClient.dll	10000	New

Problems Report

```
\New Problem P105: Error: Memory leak
G:\ViewStore\Local_aa037014_view\voobs\ISIMG\components\Streaming\Streaming\App\PathologyViewer
\PathologyViewer\Window.xaml.cs(480): Error X153: P105: Memory leak: Allocation site: Function
StreamingTestApplication::MainFrame::PrefetchingOverViewStartingPoint: Module D:\Client_withLeak\Win32
\StreamingTestApplication.exe
Stack (1 of 1 instance(s))
>StreamingTestApplication.exe!StreamingTestApplication::MainFrame::PrefetchingOverViewStartingPoint - G:
\ViewStore\Local_aa037014_view\voobs\ISIMG\components\Streaming\Streaming\App\PathologyViewer
\PathologyViewer\Window.xaml.cs(480)
```

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Debugger integration

Break into debugger

- Analysis can stop when it detects a problem
- User is put into a standard debugging session

Windows*

- Microsoft* Visual Studio Debugger

Linux*

- gdb

2x-20x | Detect Leaks
10x-40x | Detect Memory Problems
20x-80x | Locate Memory Problems

Analysis Time Overhead

Detect Memory Problems Copy

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

- Analyze without debugger
Run an analysis and report all detected problems. Use to view correctness issues without stopping in the debugger to examine them.
- 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 problem detected.
- Select analysis start location with debugger
Run target application under the debugger with analysis disabled until you choose to turn on analysis. Before starting, set a code breakpoint to stop execution prior to where you want analysis to begin. Sele...

Debug this problem

Intel Inspector XE 2015

Target Analysis Type Collection Log Summary

Problems

ID	Type	State
P1	Memory leak	New
P2	Invalid memory access	Not f

Right click on a problem

Filters Sort

Severity Error 2

memory access 1

leak 1

op (5556)

Code Location

Description	Source	Function	Module	Object
Write	mc.cpp:150	main	mc.exe	
148			mc.exe!	
149	for (unsigned int i = 0;		mc.exe!	
150	local_mbox[i] = 0;		mc.exe!	
151			KERNEL32	
152	return 0;		ntdll.dll!RtlReA	

View Source
Edit Source
Copy to Clipboard
Explain Problem
Create Problem Report...
Debug This Problem
Change State
Merge S

Inspector XE will set breakpoint, and launch debug session at the place of the problem occurrence

Debug this problem

The screenshot displays the Intel Inspector XE interface. The main window shows the source code for `mc.cpp` in the `main()` function. The code is as follows:

```
{
    unsigned int max_objectid = 28;
    unsigned int mboxsize = sizeof(unsigned int)*max_objectid;
    unsigned int * local_mbox = (unsigned int *)malloc(mboxsize);

    for (unsigned int i = 0; i <= (mboxsize / (sizeof(unsigned int))); i++)
        local_mbox[i] = 0;
```

A yellow callout points to the line `local_mbox[i] = 0;` with the text: "Problematic code location with context values". Below this line, a tooltip shows the value of `local_mbox[i]` as `4261281277`.

The Autos window shows the following local variable values:

Name	Value	Type
i	28	unsigned int
local_mbox	0x002e5a50 {0}	unsigned int *
local_mbox[i]	4261281277	unsigned int
mboxsize	112	unsigned int

A yellow callout points to the Autos window with the text: "Local variable values".

The Problem Details window shows the error message: "Invalid memory access at 0x002e5ac0 for thread 5088". A yellow callout points to this message with the text: "Inspector XE problem context". Below the error message, the source code context is shown:

```
148
149     for (unsigned int i = 0; i <= (mbox
150         local_mbox[i] = 0;
151
```

Debugger options

Memory Error Analysis

2x-20x Detect Leaks
10x-40x Detect Memory Problems
20x-80x Locate Memory Problems

Analysis Time Overhead Memory Overhead

Detect Memory Problems Copy

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

- Analyze without debugger
Run an analysis and stop every time a problem is detected. Use to allow investigation of every problem detected.
- 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 problem detected.
- Select analysis start location with debugger
Run analysis until you choose a location to stop execution.

Start debugger session for each problem detected

Inspector XE starts analysis only after passing a breakpoint

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Automated regression testing

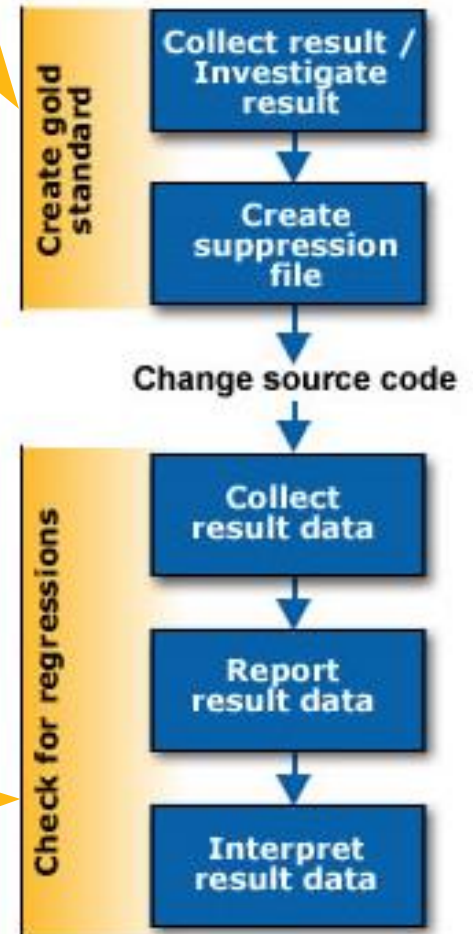
Data collection from script

- Command line interface (CLI) for running analysis
- Child process analysis

Reporting CLI

- Exporting results (pack and send)
- Text reports: XML, CSV and plain text
- Detect new problems automatically

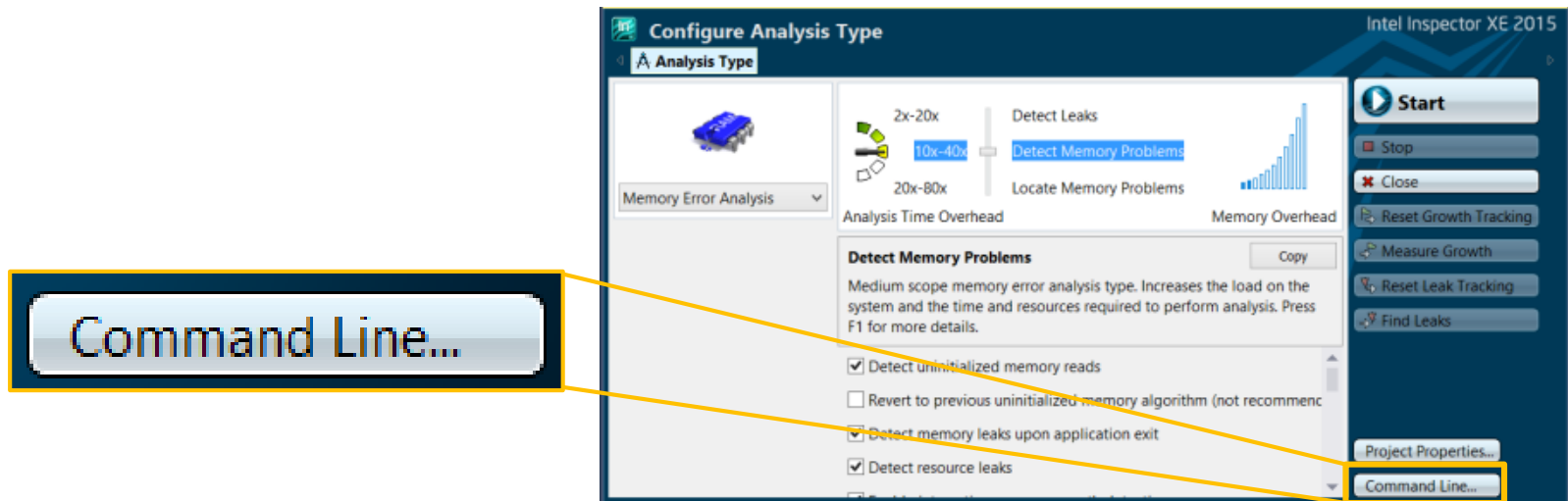
Create a baseline



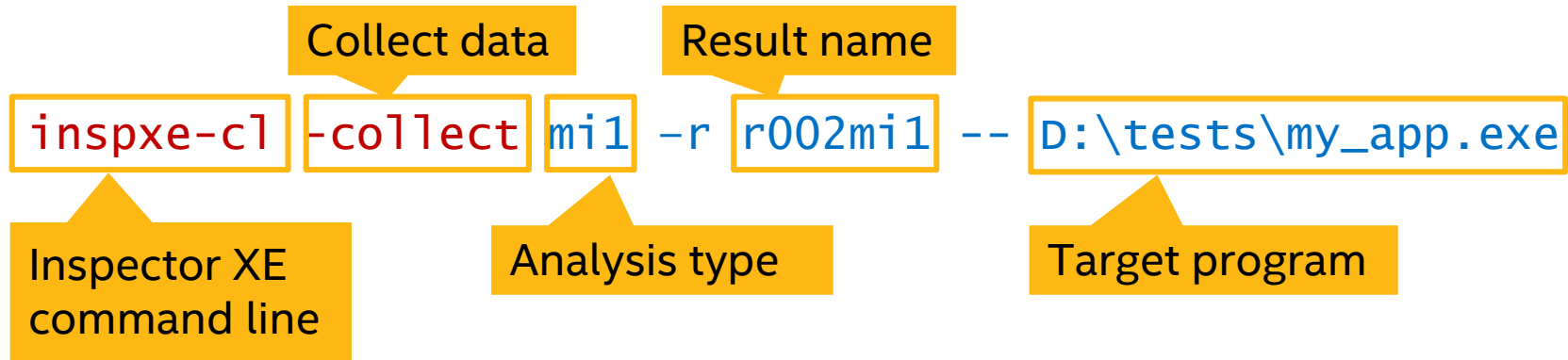
Check for regressions

Command Line Interface

- inspxe-cl is the command line:
 - windows: `C:\Program Files\Intel\Inspector XE\bin32\inspxe-cl.exe`
 - Linux: `/opt/intel/inspector_xe/bin64/inspxe-cl`
- Help:
`inspxe-cl -help`
- Set up command line with GUI



Collect results and create baseline



Collect results and create baseline

```
inspxe-cl -collect mi1 -r r002mi1 -- D:\tests\my_app.exe
```

Filter particular modules

```
inspxe-cl -collect mi1
```

```
-module-filter module1.dll,module2.dll -module-filter-  
mode exclude -- D:\tests\my_app.exe
```

Collect results and create baseline

```
inspxe-cl -collect mi1 -r r002mi1 -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1  
-module-filter module1.dll,module2.dll -module-filter-  
mode exclude -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1 -executable-of-interest  
mem_error.exe -- D:\tests\startup_script.bat
```

Analyze only target executable

Run application from script

Collect results and create baseline

```
inspxe-cl -collect mi1 -r r002mi1 -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1  
-module-filter module1.dll,module2.dll -module-filter-  
mode exclude -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1 -executable-of-interest  
mem_error.exe -- D:\tests\startup_script.bat
```

Suppress non-interesting problems

```
inspxe-cl -create-suppression-file "D:\tests\mySup"  
-result-dir r002mi1
```

Collect results and create baseline

```
inspxe-cl -collect mi1 -r r002mi1 -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1  
-module-filter module1.dll,module2.dll -module-filter-  
mode exclude -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1 -executable-of-interest  
mem_error.exe -- D:\tests\startup_script.bat
```

```
inspxe-cl -create-suppression-file "D:\tests\mySup"  
-result-dir r002mi1
```

Get baseline result

```
inspxe-cl -collect mi1 -suppression-file "D:\tests\mySup"  
-- D:\tests\my_app.exe
```

Collect results and create baseline

```
inspxe-cl -collect mi1 -r r002mi1 -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1  
-module-filter module1.dll,module2.dll -module-filter-  
mode exclude -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1 -executable-of-interest  
mem_error.exe -- D:\tests\startup_script.bat
```

```
inspxe-cl -create-suppression-file "D:\tests\mySup"  
-result-dir r002mi1
```

```
inspxe-cl -collect mi1 -suppression-file "D:\tests\mySup"  
-- D:\tests\my_app.exe
```

Find regressions comparing to the baseline

```
inspxe-cl -collect mi1 -baseline-result mi1_base --  
D:\tests\my_app.exe
```

Collect results and create baseline

```
inspxe-cl -collect mi1 -r r002mi1 -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1  
-module-filter module1.dll,module2.dll -module-filter-  
mode exclude -- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1 -executable-of-interest  
mem_error.exe -- D:\tests\startup_script.bat
```

```
inspxe-cl -create-suppression-file "D:\tests\mySup"  
-result-dir r002mi1
```

```
inspxe-cl -collect mi1 -suppression-file "D:\tests\mySup"  
-- D:\tests\my_app.exe
```

```
inspxe-cl -collect mi1 -baseline-result mi1_base --  
D:\tests\my_app.exe
```

Reporting: regression status

```
inspxe-cl -report status -r r002mi1
```

```
9 problem(s) found  
  2 Investigated  
  7 Not investigated
```

```
Breakdown by state:
```

```
  2 Confirmed  
  4 Not fixed  
  2 Regression  
  1 New
```

Reporting: problem list

```
inspxe-cl -report problems -r r002mi1
```

P1: Error: Memory leak

P1.9: Memory leak: 499500000 Bytes: New

/home/test.cpp(31): Error x9: Allocation site: Function
main: Module /home/test

P2: Error: Memory leak

P2.10: Memory leak: 999000000 Bytes: New

/home/test.cpp(32): Error x10: Allocation site:
Function main: Module /home/test

Reporting: extended problem list

```
inspxe-cl -report problems -report-all -r r002mi1
```

P1: Error: Memory leak

P1.9: Memory leak: 499500000 Bytes: New

/home/test.cpp(31): Error X9: Allocation site: Function
main: Module /home/test

Code snippet:

```
29  __itt_heap_record_memory_growth_begin();  
30      for (i=0; i < 1000; i++) {  
>31      a();  
32      b();  
33      free(p3);
```

Stack (1 of 1 instance(s))

>test!main - /home/test.cpp:31

libc.so.6!__libc_start_main - /lib64/libc.so.6:0x1ecd9

ex!_start - /home/test:0x3cb4

Reporting: other CLI options

XML output

```
inspxe-cl -report problems -format=xml -r r002mi3
```

CSV output

```
inspxe-cl -report problems -format csv -csv-delimiter tab  
-report-output ./out/observations.csv
```

Filter from CLI

```
inspxe-cl -report problems -filter source=combine.cpp -  
filter investigated=not_investigated
```

Export full result

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

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Intel Inspector XE: User APIs

Enable you to

- Control collection, limit analysis scope
- Specify non-standard synchronization primitives
- Specify custom memory allocation primitives

To use user APIs:

- Include `ittnotify.h`, located at `<install_dir>/include`
- Insert `__itt_*` notifications in your code
- Link to the `libittnotify.lib` file located at `<install_dir>/<lib32|lib64>`
- Available for C/C++ and Fortran

Collection control APIs

API	Description
<code>void __itt_suppress_push(unsigned int etype)</code>	Stop analyzing for errors on the current thread
<code>void __itt_suppress_pop(void)</code>	Resume analysis
<code>void __itt_suppress_mark_range (__itt_suppress_mode_t mode, unsigned int etype, void * address, size_t size);</code>	Suppress or unsuppress error detection for the specific memory range (object).
<code>void __itt_suppress_clear_range (__itt_suppress_mode_t mode, unsigned int etype, void * address, size_t size);</code>	Clear the marked memory range

Collection control APIs

```
#include <ittnotify.h>

...
#pragma omp parallel
    __itt_suppress_push(
        __itt_suppress_threading_errors);
    /* Any threading errors here will be
       ignored by the calling thread.
       In this case, each thread in the region */
    not_analyzed_code();

    __itt_suppress_pop();
    /* Any threading errors here will be
       seen by Inspector*/
}
```

Collection control APIs

```
#include <ittnotify.h>

int variable_to_watch;
int other_variable;
...
    // Change the default mode by using NULL and 0 as address and size
__itt_suppress_mark_range(
    __itt_suppress_range,
    __itt_suppress_threading_errors,
    NULL,
    0);

    // Ensure we see errors on variable_to_watch
__itt_suppress_mark_range(
    __itt_unsuppress_range,
    __itt_suppress_threading_errors,
    &variable_to_watch,
    sizeof(variable_to_watch));

#pragma omp parallel
    ...
    variable_to_watch++;           // Race will be reported
    other_variable++;             // Race will not be reported
}
```

Custom memory allocation

```
#include <ittnotify.h>

__itt_heap_function my_allocator;
__itt_heap_function my_reallocator;
__itt_heap_function my_freer;

void* my_malloc(size_t s)
{
    void* p;

    __itt_heap_allocate_begin (my_allocator, s, 0);
    p = user_defined_malloc (s);
    __itt_heap_allocate_end (my_allocator, &p, s, 0);

    return p;
}
... // Do similar markup for custom "realloc" and "free" operations

// Call this init routine before any calls to user defined allocators
void init_itt_calls()
{
    my_allocator = __itt_heap_function_create("my_malloc", "mydomain");
    my_reallocator = __itt_heap_function_create("my_realloc", "mydomain");
    my_freer = __itt_heap_function_create("my_free", "mydomain");
}
```


Custom synchronization

```
#include <ittnotify.h>

CSEnter (MyCriticalSection * cs)
{
    while(cs->LockIsUsed)
    {
        if(cs->LockIsFree)
        {
            // Code to acquire the lock goes here
            __itt_sync_acquired((void *) cs);
        }
    }
}

CSLeave (MyCriticalSection *cs)
{
    if(cs->LockIsMine)
    {
        __itt_sync_releasing((void *) cs);
        // Code to release the lock goes here
    }
}
```

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Using the Intel® Inspector XE with MPI

- Compile the `inspector_example.c` code with the MPI scripts
- 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

Agenda

Intro to Intel® Inspector XE

Analysis workflow

Memory problem analysis

- Lab 1. Finding memory errors

Threading problem Analysis

- Lab 2. Finding threading errors

Preparing setup for analysis

Managing analysis results

Integration with debugger

Automated regression testing

User API

Using the Intel® Inspector XE with MPI

Summary

Intel Inspector XE: Summary

Advanced correctness checking

- Find issues that traditional testing misses
- Dynamic memory and threading error detection

Automated regression

- Command line interface
- Suitable for scripting

Wide analysis capabilities

- GUI data management
- Debugger integration

Ship high quality software products!

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

