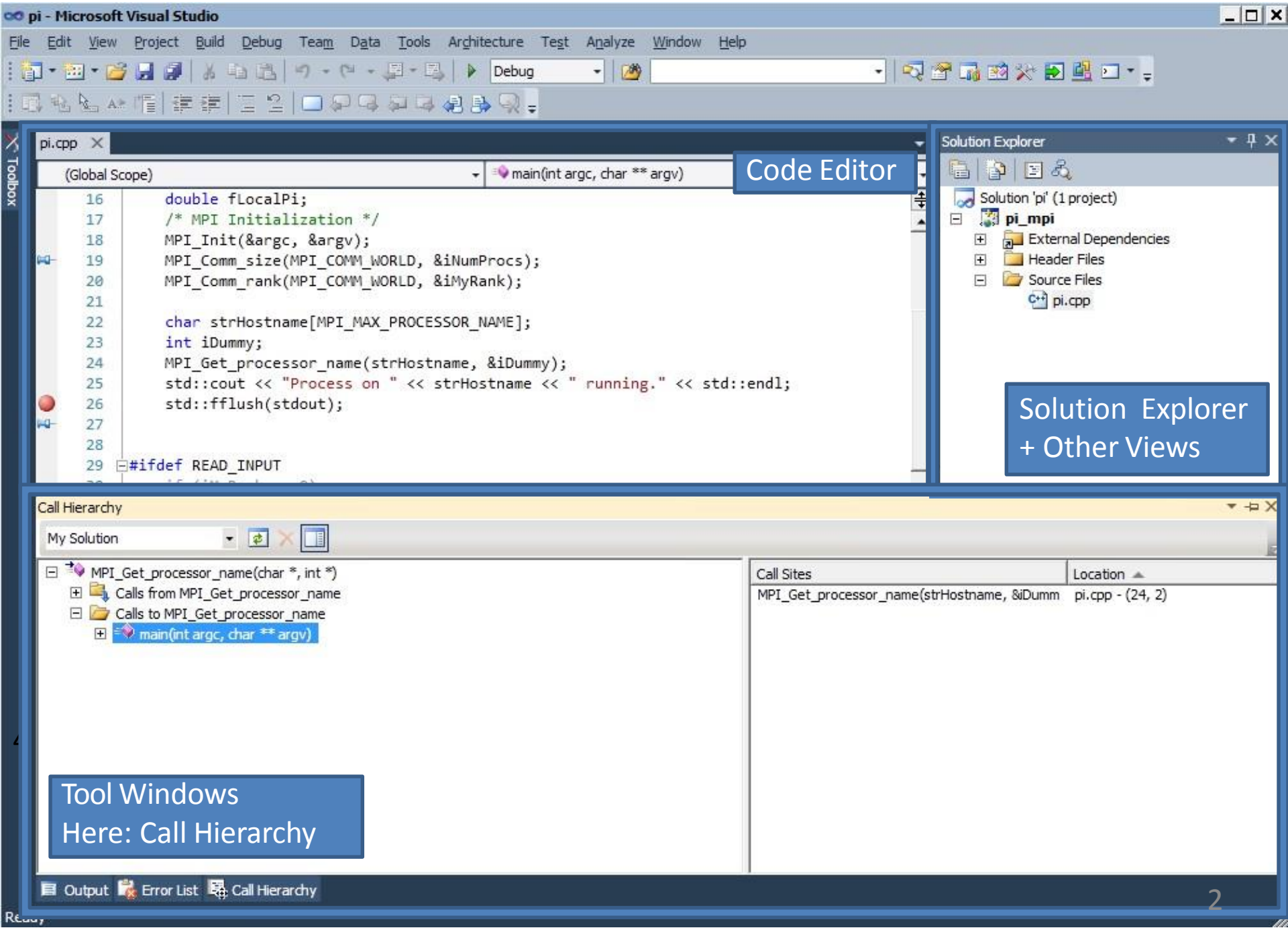
Visual Studio for parallel programming - Agenda

- Overview and Project Management
- The Microsoft and Intel compilers
- Using MPI
- Debugging MPI programs

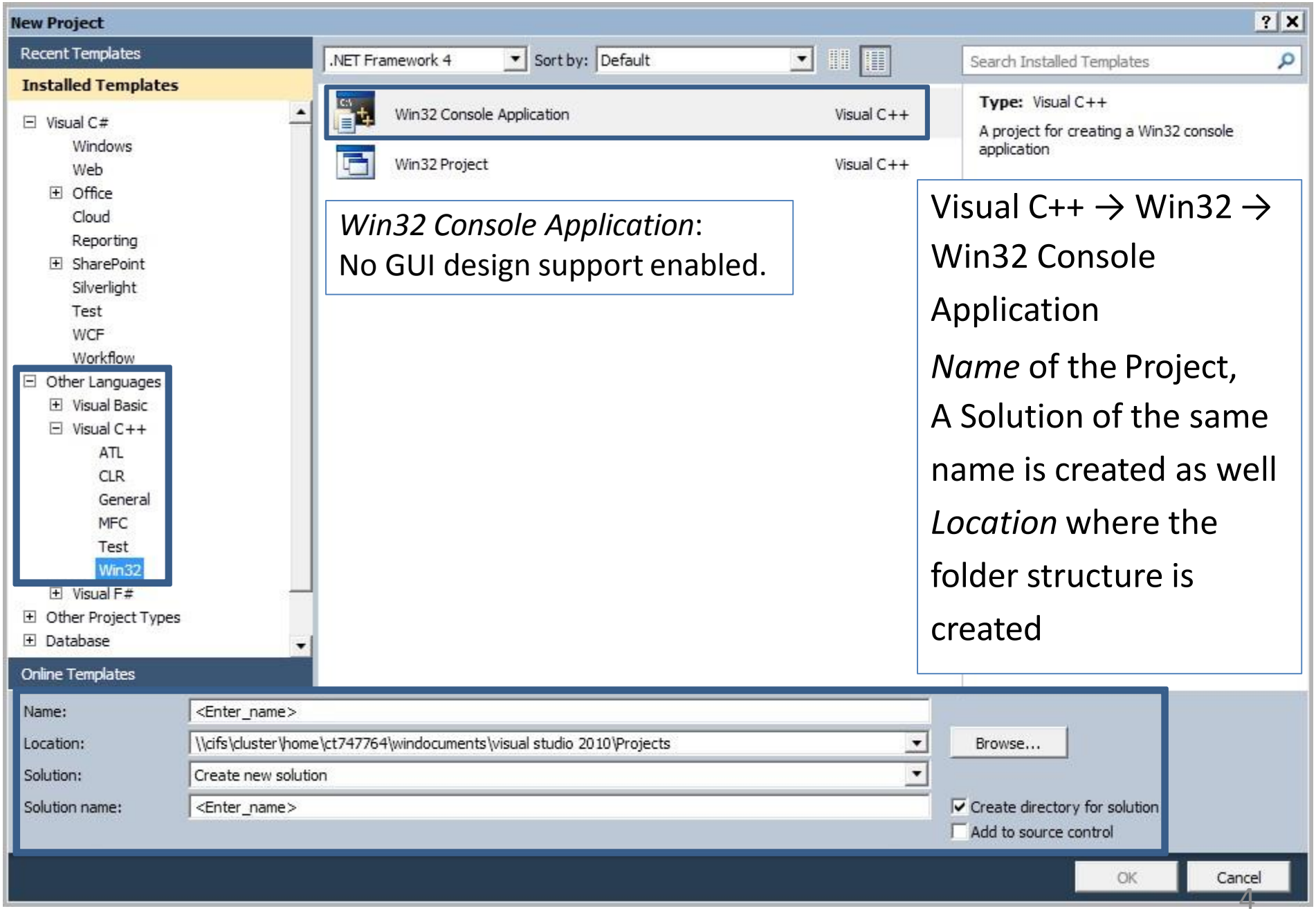

pi - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
pi.cpp
(Global Scope)
main(int argc, char ** argv)
Code Editor
16 double fLocalPi;
17 /* MPI Initialization */
18 MPI_Init(&argc, &argv);
19 MPI_Comm_size(MPI_COMM_WORLD, &iNumProcs);
20 MPI_Comm_rank(MPI_COMM_WORLD, &iMyRank);
21
22 char strHostname[MPI_MAX_PROCESSOR_NAME];
23 int iDummy;
24 MPI_Get_processor_name(strHostname, &iDummy);
25 std::cout << "Process on " << strHostname << " running." << std::endl;
26 std::fflush(stdout);
27
28
29 #ifdef READ_INPUT
Solution Explorer
Solution 'pi' (1 project)
pi_mpi
External Dependencies
Header Files
Source Files
pi.cpp
Solution Explorer + Other Views
Call Hierarchy
My Solution
MPI_Get_processor_name(char *, int *)
Calls from MPI_Get_processor_name
Calls to MPI_Get_processor_name
main(int argc, char ** argv)
Call Sites
Location
MPI_Get_processor_name(strHostname, &iDumm
pi.cpp - (24, 2)
Tool Windows Here: Call Hierarchy
Output
Error List
Call Hierarchy
Ready

Visual Studio: Project Management (1/5)

- Everything that you do in Visual Studio will take place within the context of a *Solution*.
  - A Solution is a higher-level container for other items, for example a *Project*. Any other kind of file type can also be added to a Solution, for example documentation items.
  - A Solution can not contain another Solution.
  - Solutions group and apply properties across projects.

- A *Project* maps one to one with a compiler target.
  - A Project organizes the code.

- To start your work, a new Project has to be created with *File → New → Project…*

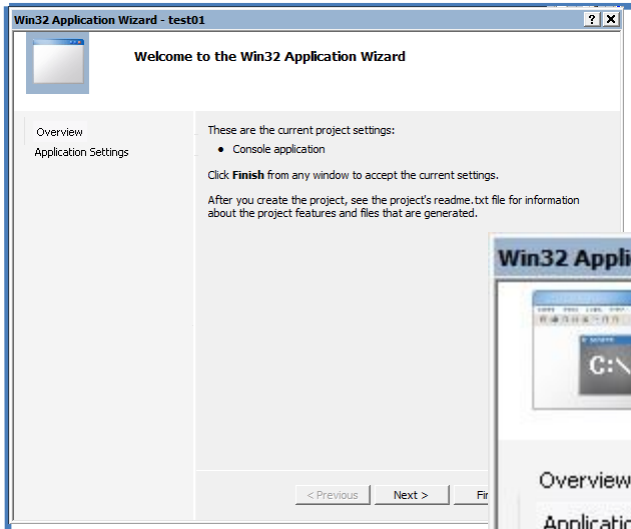


Win32 Console Application:
No GUI design support enabled.

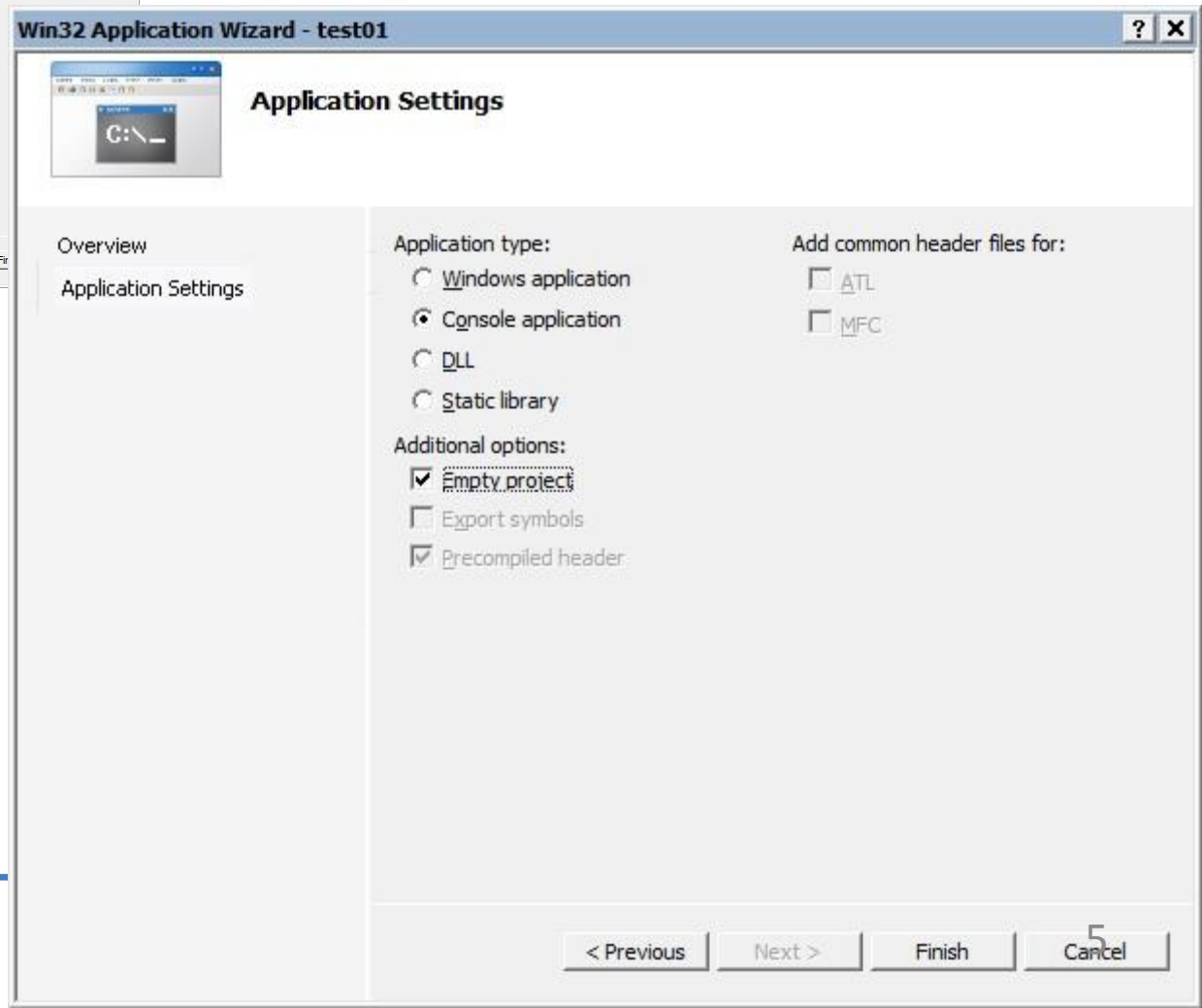
Visual C++ → Win32 → Win32 Console Application

Name of the Project,
A Solution of the same name is created as well
Location where the folder structure is created

Visual Studio: Project Management (3/5)

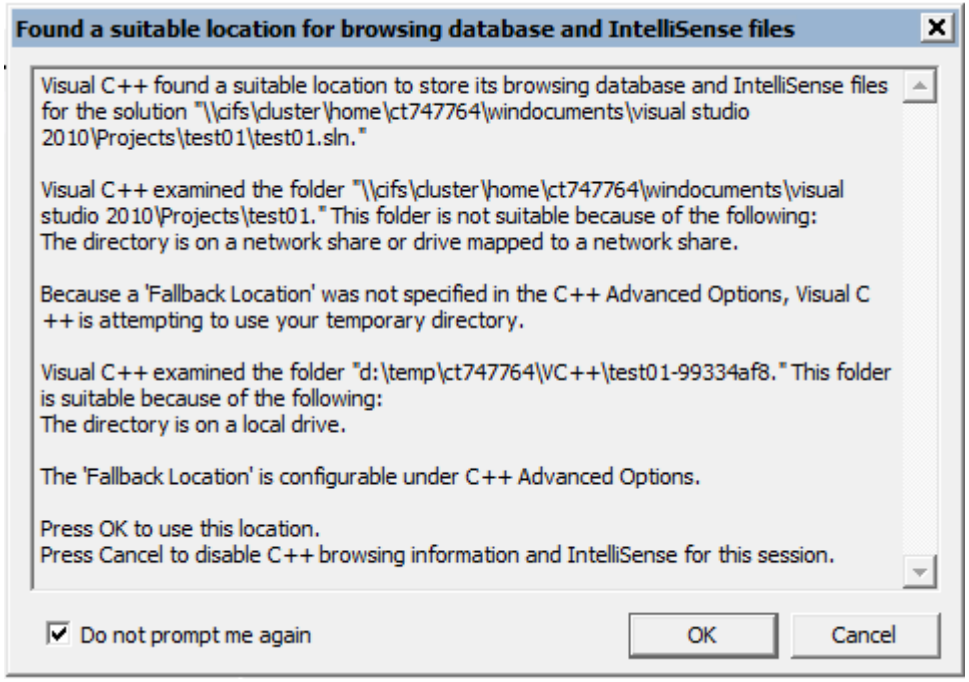


Choose *Empty project* if you already have source files.

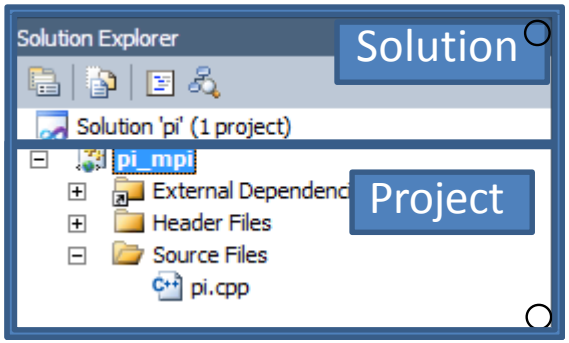


Visual Studio: Project Management (4/5)

- An issue specific to our Cluster: The IntelliSense database may not be stored on a network drive. VS2010 resolves this automatically for you by selecting *Ok*.



Visual Studio: Project Management (5/5)

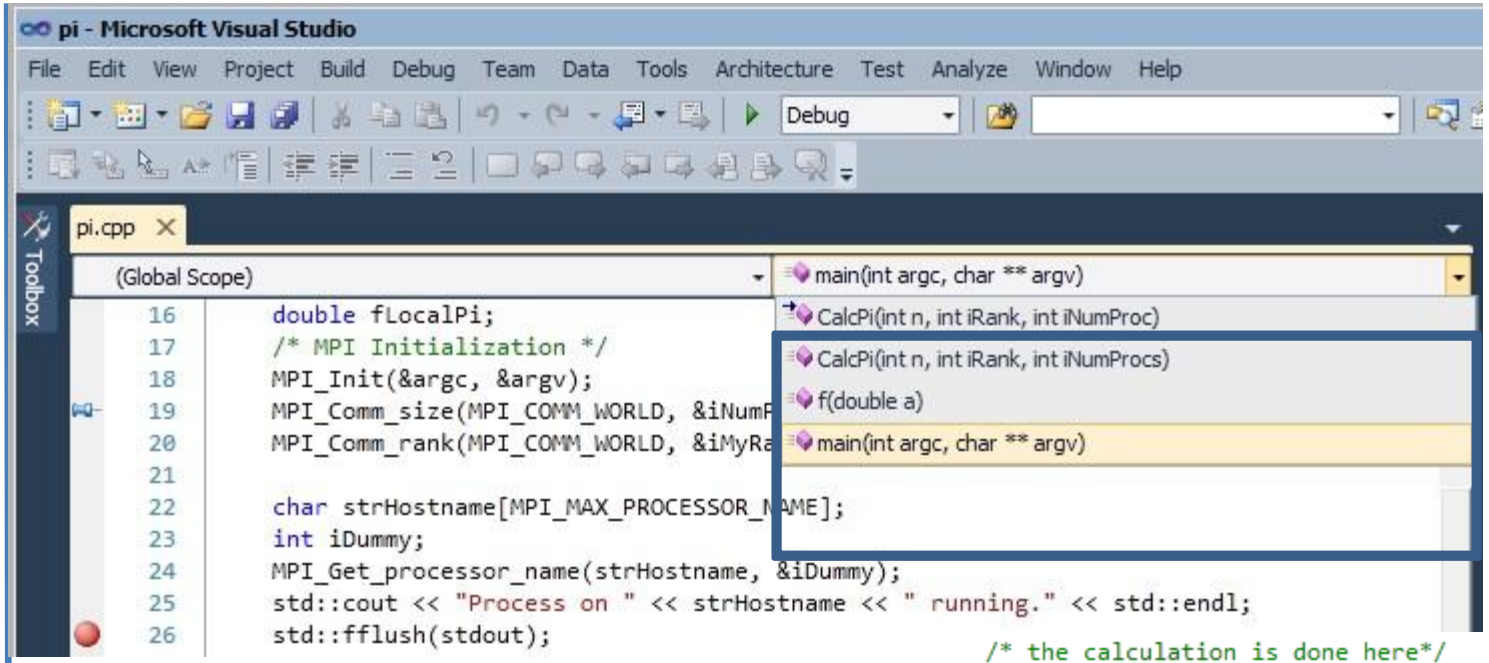


- In many cases, the shortest way to a desired operation can be found by right-clicking on a GUI element and using the context menu.

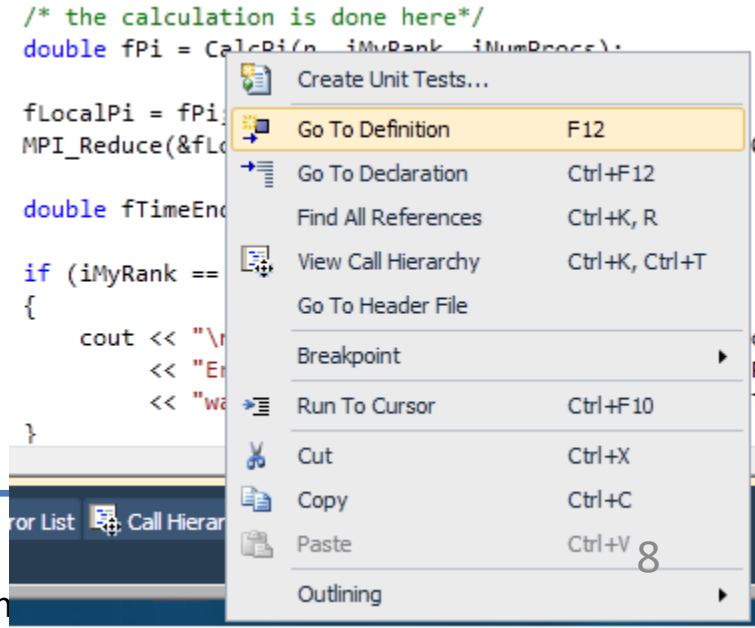
- Adding existing source code items (files) to a project: right-click on the Project (not the Solution !) and *Add → Existing Item…*
- Adding new items: right-click on the Project and *Add → New Item…*

- The folders (e.g. *Source Files*) do not have any other meaning than aiding you in structuring the files in a project. They do not map to physical folders. Creating your own folders may help to organize large projects.

Source navigation in Visual Studio 2010 (1/2)

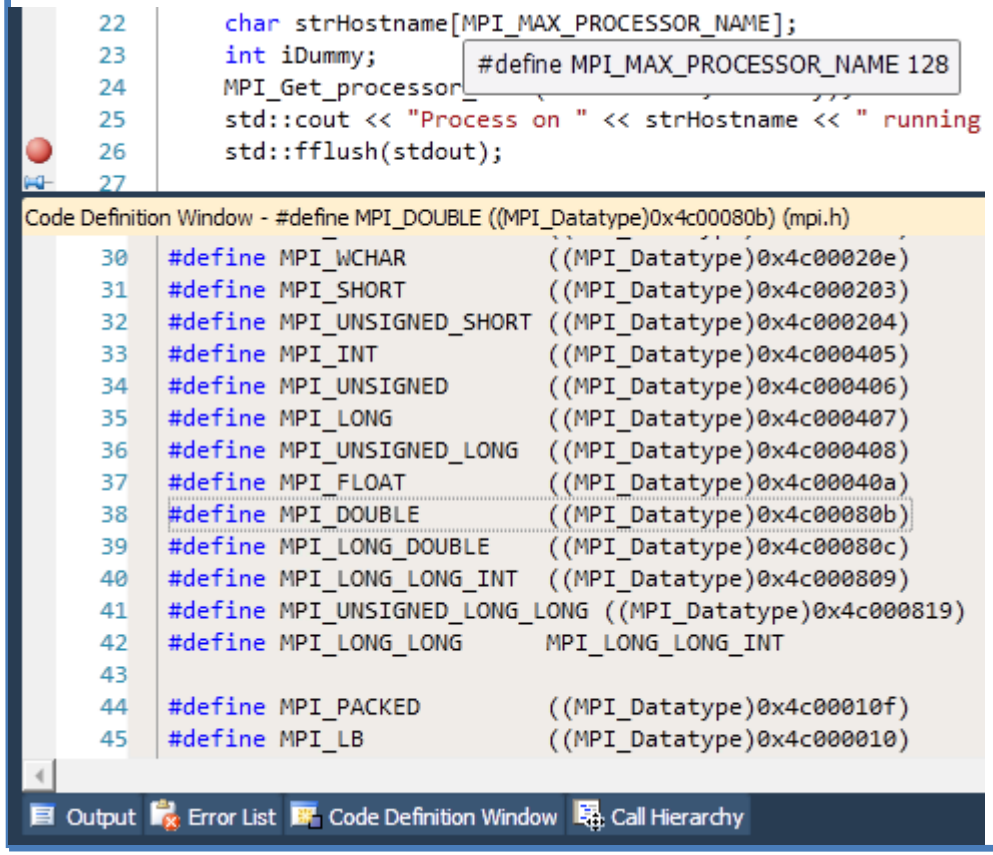


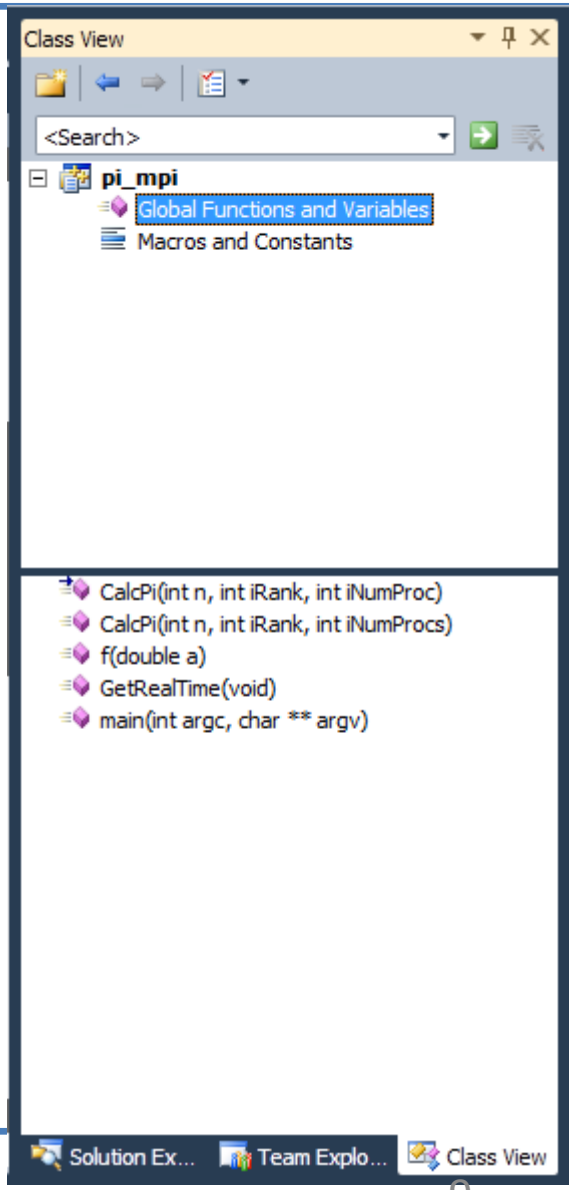
- Selecting a scope + function to navigate right into it‘s imple-mentation.
- Right-clicking a symbol opens up a corresponding context menu:



Source navigation in Visual Studio 2010 (2/2)

- The Class View is available from the menu via *View → Class View* as well as the *Code Definition Window*.





Directory layout of Visual Studio solutions

- The executable is created in the directory of the active configuration during the build process.

- Directory structure of a solution:

<top level>	Given user directory
<project name>	Created by VS2005 / VS2008 / VS2010
Debug	Configuration: *Debug*
Release	Configuration: *Release*
x64	Platform: *x64* (64bit for Amd64/Intel64)
Debug	Configuration: *Debug*
Release	Configuration: *Release*

Visual Studio Configurations (1/3)

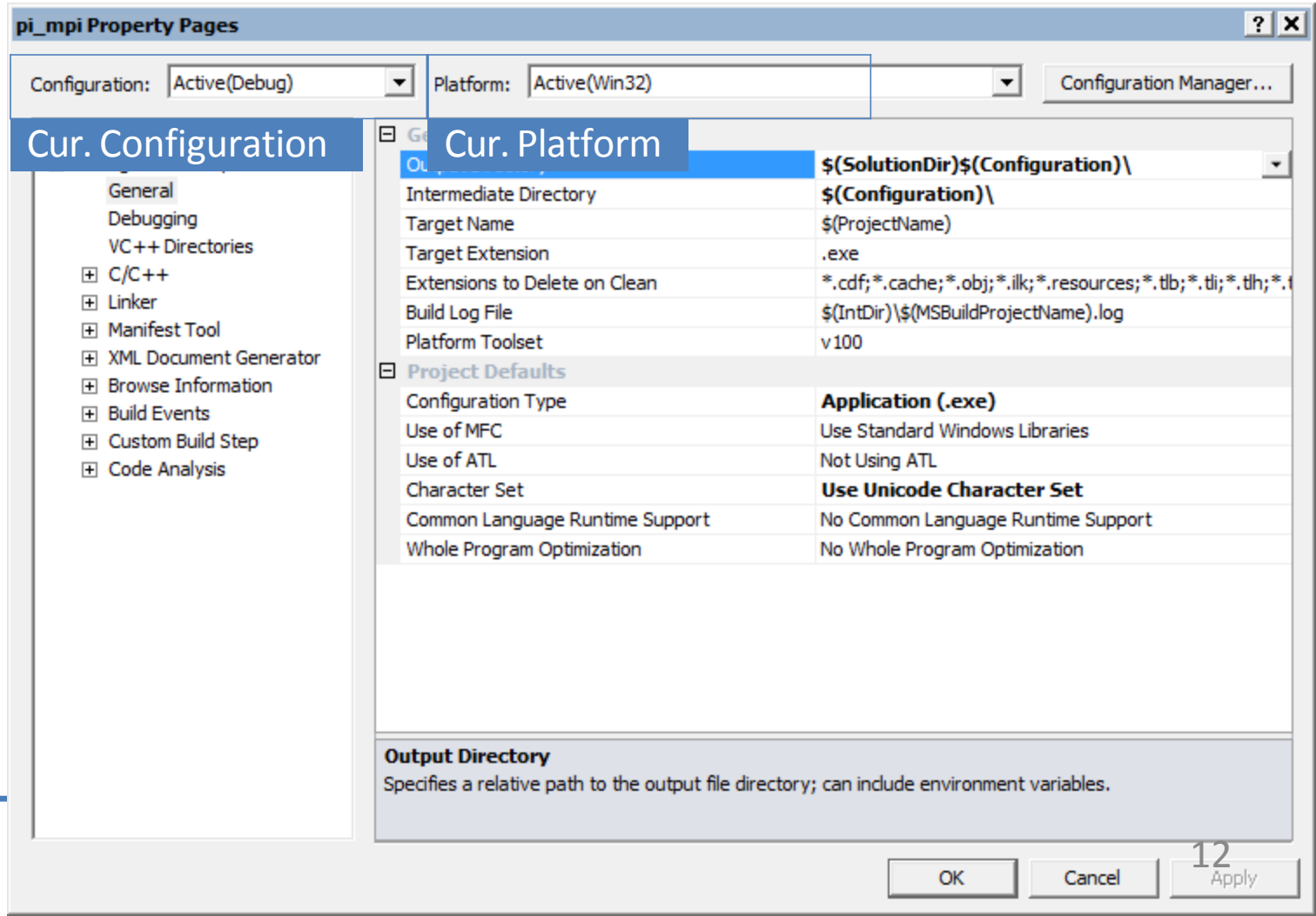
- The set of compiler options is managed in a *Configuration*.
- There are two configurations pre-defined: *Debug* and *Release*.
  - Debug: typical options for debugging, no optimization.
  - Release: debugging still possible, some optimization options.

- The compile process can be triggered by right-clicking on the project and choosing *Build*. Or from the menu: *Build* → *Build <projectname>*.
- *Build* → *Build Solution* builds all projects in the solution.

- During and after the compile process compiler output (informational messages, warnings, errors) is displayed in the tool windows *Output* or *Error List*.
- By double-clicking on such a message, the cursor jumps to the corresponding place in the code.

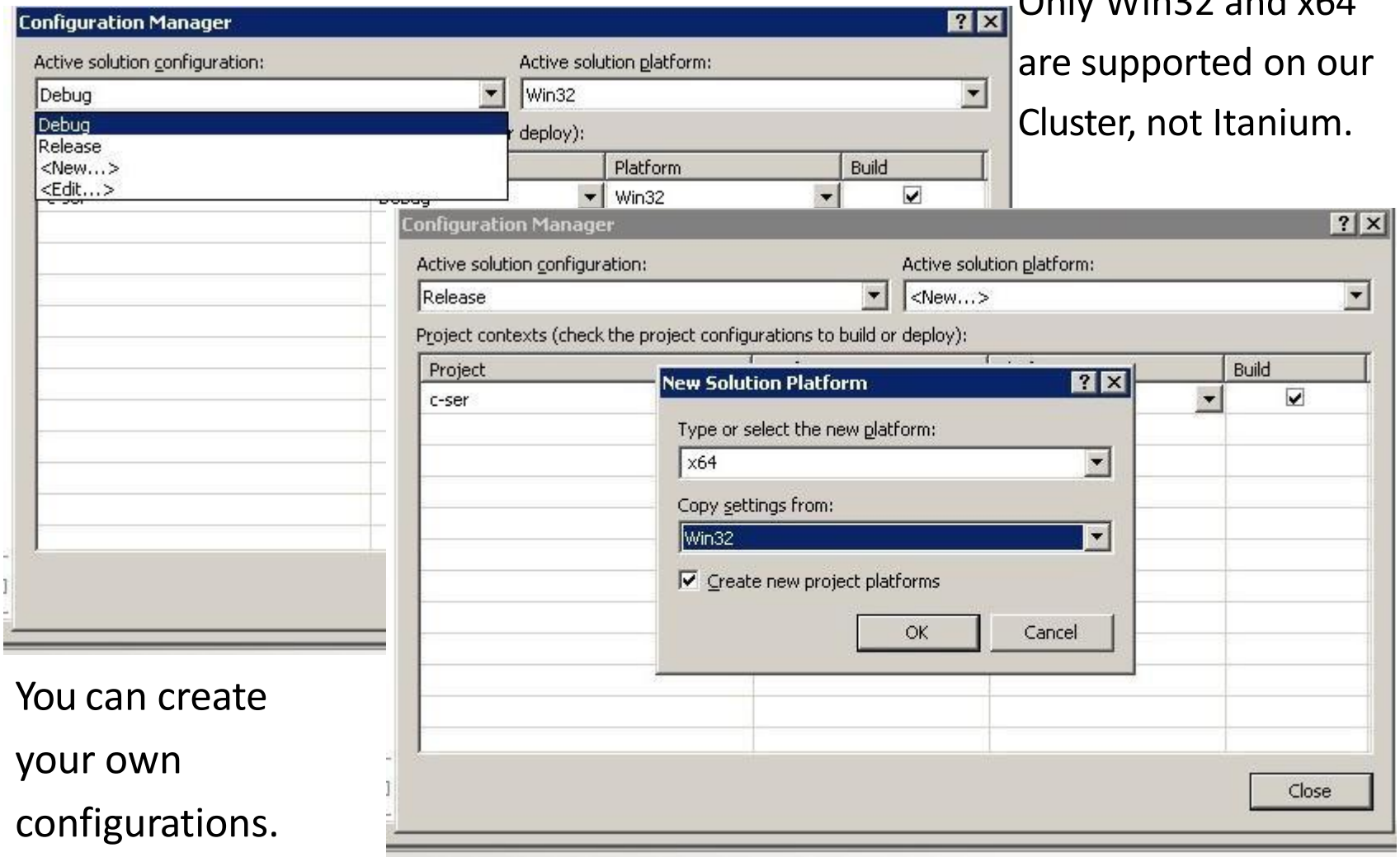
Visual Studio Configurations (2/3)

- Right-clicking on a project and choosing Properties leads to the project configuration dialog.



Visual Studio Configurations (3/3)

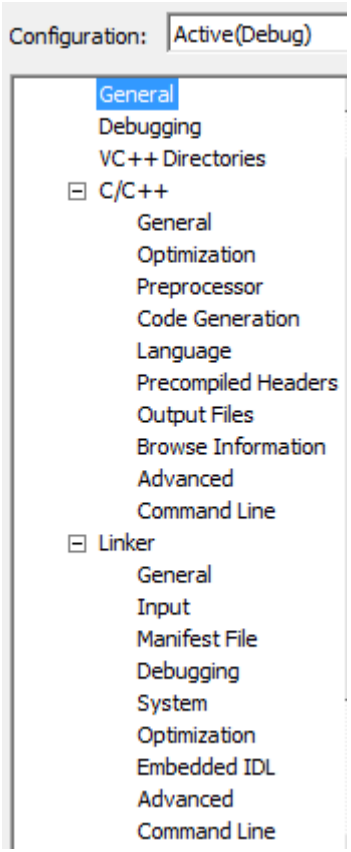
Build → Configuration Manager:



Only Win32 and x64 are supported on our Cluster, not Itanium.

You can create your own configurations.

Microsoft C/C++-specific settings



- Important General Settings:
  - C/C++ → General
    - Addition Include Directories: Include Path
  - Linker → General
    - Additional Library Directories: Library Path
  - Linker → Input
    - Additional Dependencies: Libraries to be used
- Important Optimization Settings:
  - C/C++ → Optimization
    - Optimization: General Optimization Level
    - Inline Function Expansion: Inlining
  - C/C++ → Code Generation
    - Enable Enhanced Instruction Set: Vectorization

Portable Time Measurement (1/3)

- Porting applications from Unix to Windows (or the other way around) can be quite hard … but it was not for most user codes (HPC) we tried on Windows.
  (1) The most common problem was time measurement as `gettimeofday()` is not available on Windows,
  (2) followed by directory management issues where ‚/‘ instead of ‚\‘ had been used before.

- In most cases we attacked (2) using `#ifdefs`.
- Handling (1) depends on the programming language:
  - C++: We have written a version of `double realtime()` for Windows and Unix.
  - FORTRAN: As the library (defined along with the language) already provides time measurement facilities, we used these.

Portable Time Measurement (3/3)

- Taking time the MPI way:

```
#include <mpi.h>
    ...
    double t1, t2, elapsed_seconds;
    t1 = MPI_Wtime();
    ...
    t2 = MPI_Wtime();
    elapsed_seconds = t2 - t1;
```

Enabling MPI (1/2)

- As MPI is implemented by a library, an application includes a file containing the type and function declarations named `mpi.h` and has to be linked with that library.

- Modify the project properties (1/2):
  - Include Path: *C/C++* → *General* → *Additional Include Directories*
    - MS-MPI 2008 on the cluster in Aachen:
      C:\Program Files\Microsoft HPC Pack 2008 SDK\Include
    - I-MPI on the cluster in Aachen:
      C:\Program Files
      (x86)\Intel\ICT\3.1\mpi\3.1\[ia32|em64t]\include

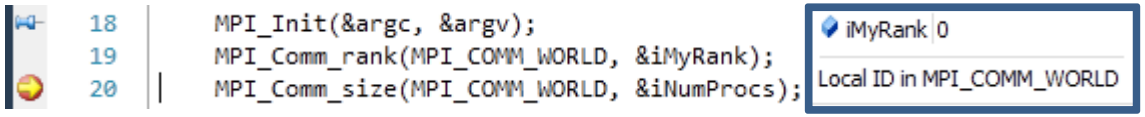
Enabling MPI (2/2)

- Modify the project properties (2/2):
  - Library Path: *Linker → General → Additional Library Directories*
    - MS-MPI 2008 on the cluster in Aachen:
      C:\Program Files\Microsoft HPC Pack 2008 SDK\Lib\[i386|amd64]
    - I-MPI on the cluster in Aachen:
      C:\Program Files (x86)\Intel\ICT\3.1\mpi\3.1\[ia32|em64t]\lib

- No significant performance difference, so our advise:
  - Use MS-MPI with Visual Studio MPI Debugger
  - Use I-MPI with Intel Trace Analyzer & Collector
  - Sometimes a program does not like a specific MPI, so it is always a good thing to have a second one available…

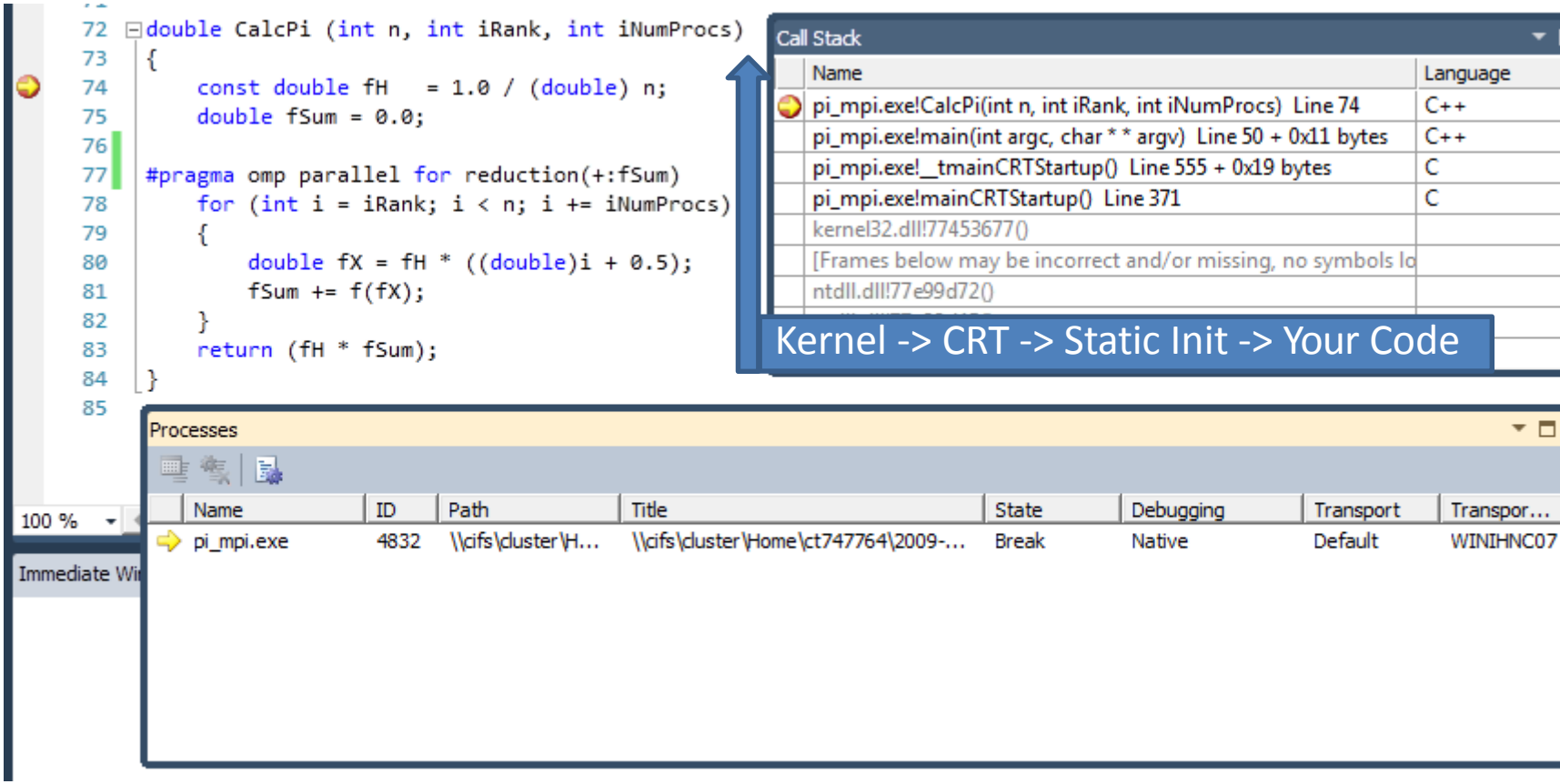
Debugging Basics (1/2)

- A breakpoint can be set by clicking in the grey area left of the line number. Clicking again removes the breakpoint.



Debugging Basics (2/2)

- During a debugging session, the actual program location is marked by a yellow arrow. You can drag this arrow up/down.



Debugging MPI programs (1/6)

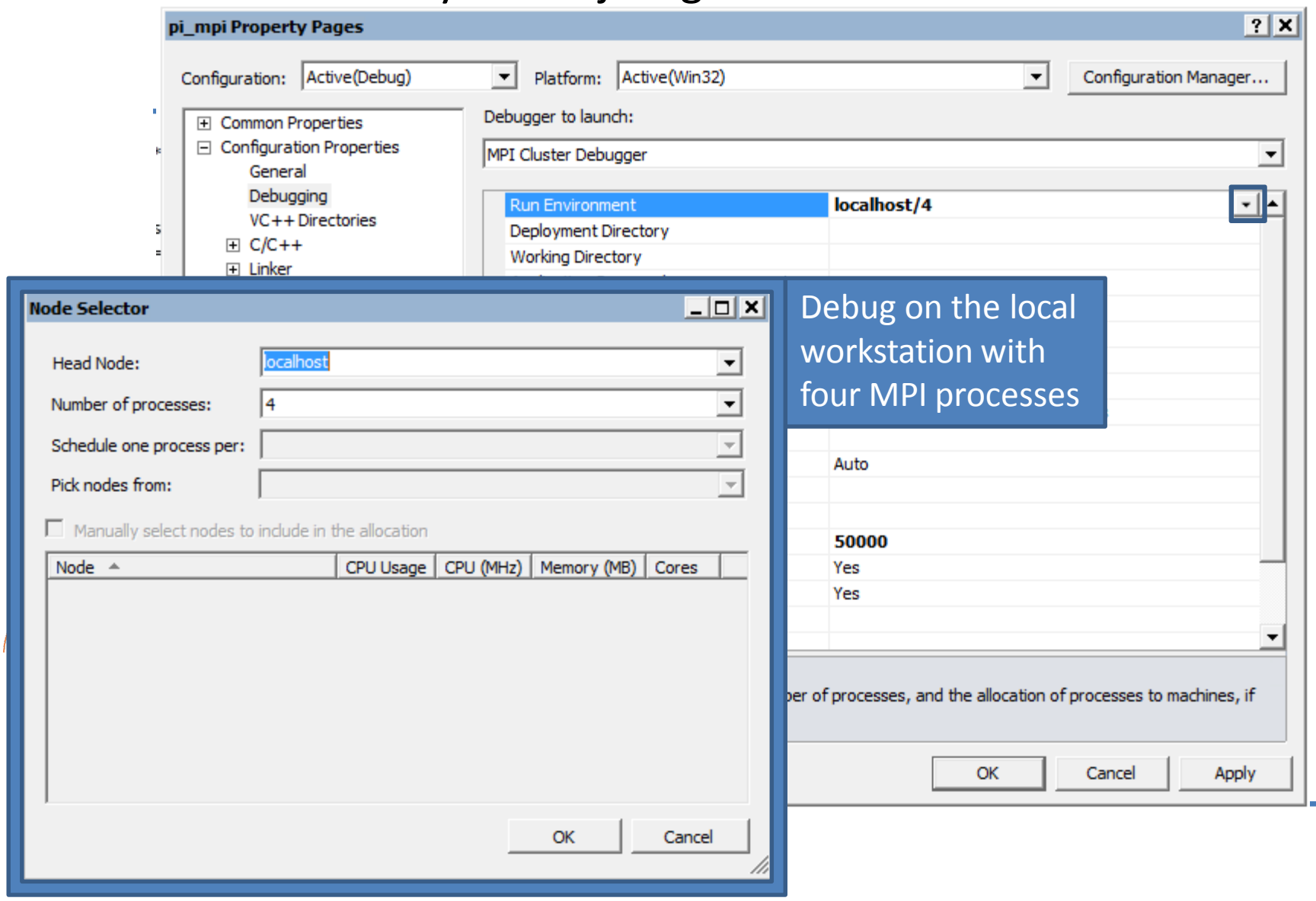
- MS-MPI works best, but you should be able to use I-MPI as well. At least the following instructions work for both.
- Visual Studio supports debugging of MPI programs using the *Cluster Debugger*. As far as I know – or was able to verify – the cluster debugger only works with the Microsoft C/C++ compiler and not with projects using the Intel C/C++ compiler or the Intel FORTRAN compiler.
- In the project properties under *Debugging*, choose the *MPI Cluster Debugger* as *Debugger to launch*. For VS2008 only:
  - MPIRun: *„C:\Program Files\Microsoft HPC Pack 2008 SDK\Bin “*
  - MPIRun Arguments: for example *–n 2*
  - MPIShim Location:
    It is not possible to specify a path containing empty spaces here, so you have to copy MPIShim from *c:\program files[(x86)]\microsoft visual studio 9.0 \common7\ide\RemoteDebugger\x86[or x64]\MPIShim* to a suitable location.

Debugging MPI programs (2/6)

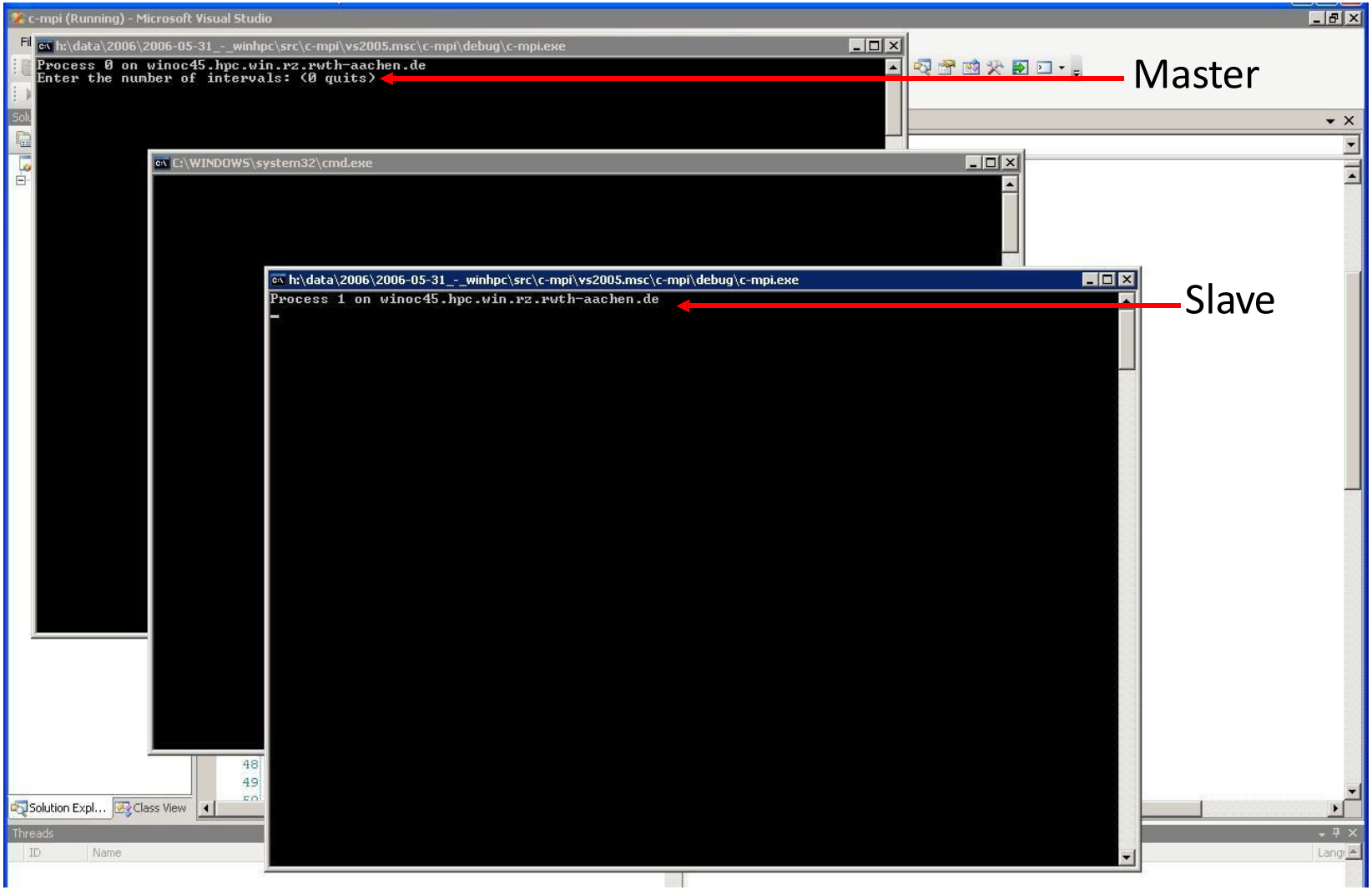
- In order to stop all processes at a breakpoint, please check for the following option: In *Tools* → *Options* → *Debugging* → *General* the checkbox *Break all processes when one process breaks* has to be activated.
- Select the current process using the *Processes* register.

Debugging MPI programs (3/6)

- In VS2010 you can just go with the defaults:

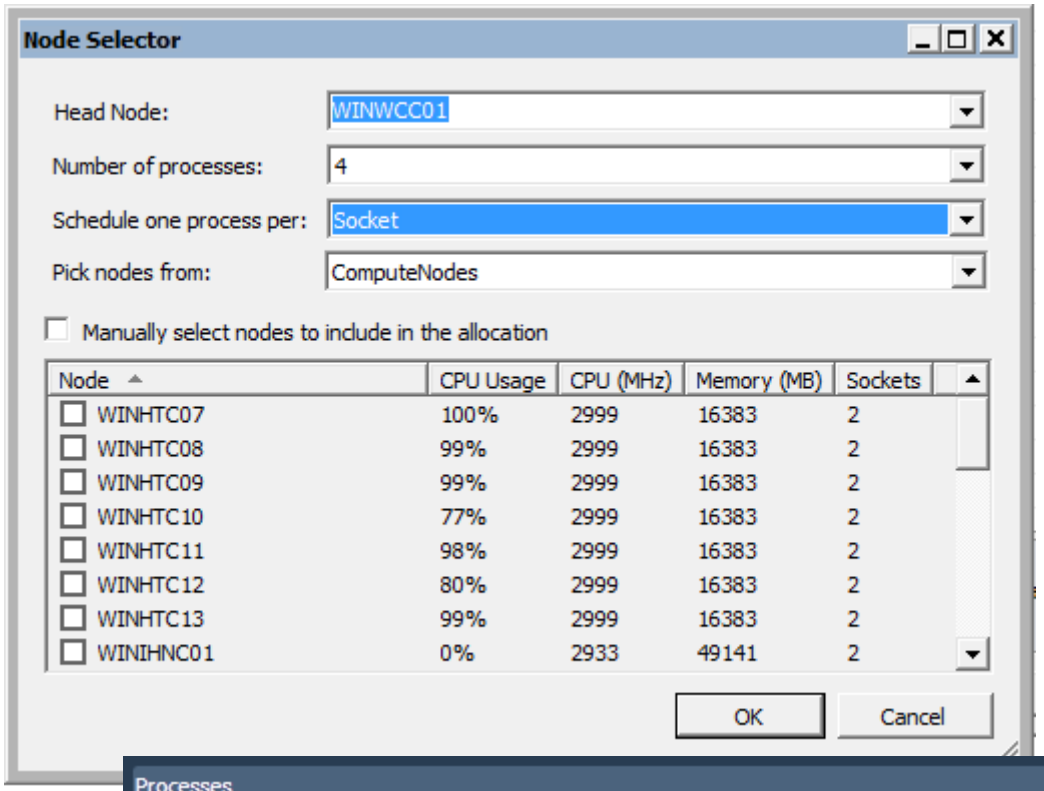


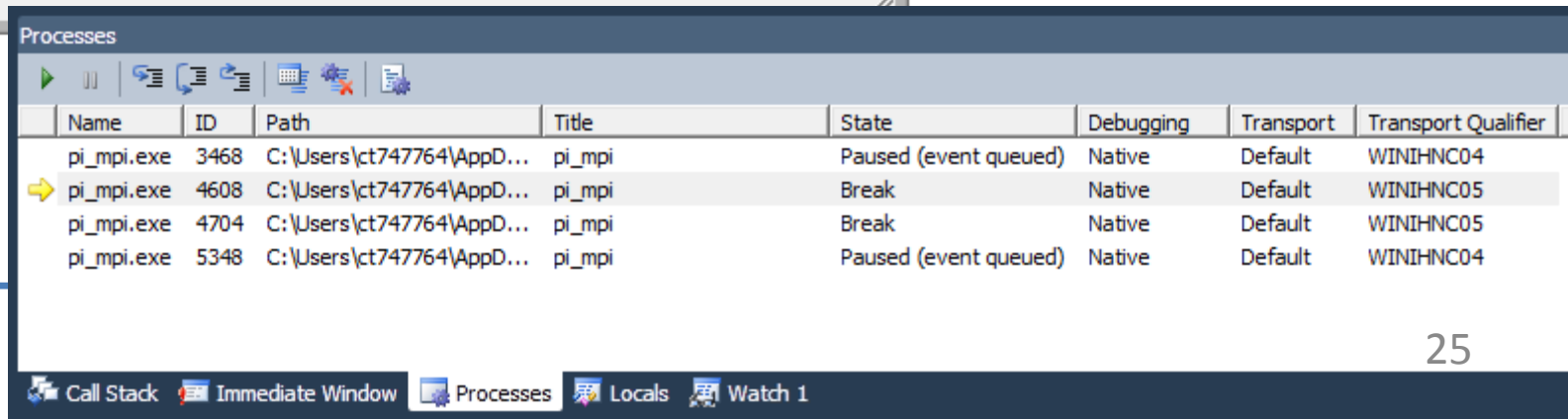
Debugging MPI programs (4/6)



Debugging MPI programs (5/6)

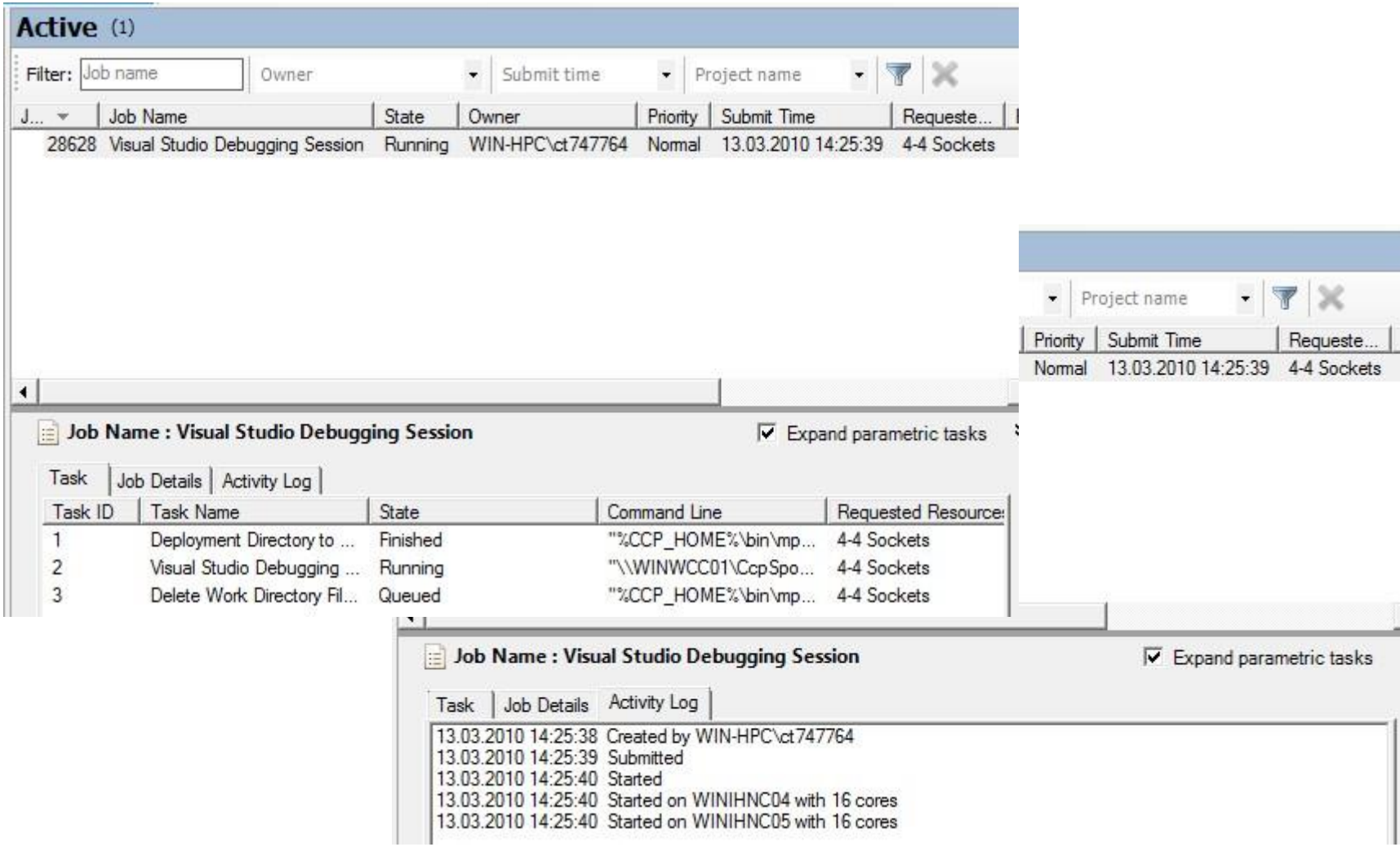
- You can also *F5 to the Cluster*, if there are free slots:





Debugging MPI programs (6/6)

- A VS2010 debug job running on our Cluster:



DDTlite: Overview

- Allinea DDT Lite is an add-in for Visual Studio 2008 SP1
  - Currently an additional patch to VS2008 is required
- Significantly improves the MPI debugging experience
  - Debug / Control MPI processes individually
  - Debug / Control groups of MPI processes individually
  - Display variable values per process side-by-side
  - Display MPI process stacks side-by-side
  - …
- For a trial version go to www.allinea.com