parallel tools platform

http://eclipse.org/ptp

A New and Improved Eclipse Parallel Tools Platform: Advancing the Development of Scientific Applications

Greg Watson, IBM
g.watson@computer.org

Jay Alameda, NCSA
jalameda@ncsa.uiuc.edu

Beth Tibbitts, IBM
tibbitts@us.ibm.com

Jeff Overbey, UIUC
overbey2@illinois.edu

November 13, 2011

The latest version of these slides will be available at
http://wiki.eclipse.org/PTP/tutorials/SC11

Portions of this material are supported by or based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under its Agreement No. HR0011-07-9-0002, the United States Department of Energy under Contract No. DE-FG02-06ER25752 and the SI2-SSI Productive and Accessible Development Workbench for HPC Applications, which is supported by the National Science Foundation under award number OCI 1047956

SC11 Seattle, WA

Tutorial Outline

Time (Tentative!)	Module	Topics	Presenter
8:30-9:00	1. Eclipse and PTP Installation	✦ Installation of Eclipse and PTP	Greg/Beth
9:00-9:30	2. Introduction & Overview	✦ Eclipse architecture & organization overview	Greg
10:00-10:30	BREAK		
10:30-12:00	3. Developing with Eclipse *Note: try to start this before break. This is the longest module*	✦ Eclipse basics; Creating a new project ✦ Local, remote, and synchronized projects ✦ C and Fortran features; Parallel dev. features ✦ Building managed and Makefile projects ✦ Resource Managers and launching a parallel app	Beth, Jeff, Jay
12:00 – 1:00	Lunch		
1:00-2:30	4. Debugging	✦ Debugging an MPI program	Greg
2:30-3:00	BREAK		
3:00-4:30	5. Performance Tuning & Analysis Tools	✦ TAU, ETFw including hands-on exercise ✦ Overview of GEM, PPW	Wyatt Spear
4:30-5:00	6. Other Tools, Wrapup	✦ NCSA HPC Workbench, Other Tools, website, mailing lists, future features	Jay/Beth

Final Slides, Installation Instructions

- Please go to http://wiki.eclipse.org/PTP/tutorials/SC11 for slides and installation instructions

Module 1: Installation

- Objective
  - To learn how to install Eclipse and PTP
- Contents
  - System Prerequisites
  - Eclipse Download and Installation
  - PTP Installation from an Update Site
  - Installation Confirmation

About the Tutorial Installation

- This tutorial assumes you have Eclipse and PTP pre-installed on your laptop
- If you already have Eclipse installed, go directly to "Starting Eclipse", slide 5
- If you don't have Eclipse installed, you will need to follow the handouts so that you can catch up with the rest of the class
- Note: up-to-date info on installing PTP and its pre-reqs is available from the release notes:
  - http://wiki.eclipse.org/PTP/release_notes/5.0
  - This information may supersede these slides

System Prerequisites

- Local system (running Eclipse)
  - Linux (just about any version)
  - MacOSX (10.5 Leopard or 10.6 Snow Leopard)
  - Windows (XP on)
- Java: Eclipse requires Sun or IBM Java
  - Only need Java runtime environment (JRE)
  - Java 1.5 or higher
    - Java 1.5 is the same as JRE 5.0
  - Note: The GNU Java Compiler (GCJ), which comes standard on Linux, will not work!
  - Note 2: OpenJDK, distributed with some linux distributions, has not been tested with Eclipse.
  - See http://wiki.eclipse.org/PTP/installjava

Eclipse Packages

- Eclipse is available in a number of different packages for different kinds of development
  - http://eclipse.org/downloads
  - This is Eclipse 3.7, also known as "Indigo"
- With Indigo, there is a new package directly relevant for HPC:
  - Eclipse IDE for Parallel Application Developers
  - This is recommended for all new installs
- Can also add PTP to an existing Eclipse installation

Eclipse Installation

- Download the appropriate package
- If your machine is Linux or Mac OS X, untar the file
  - On Mac OS X you can just double-click in the Finder
- If your machine is Windows, unzip the file
- This creates an **eclipse** folder containing the executable as well as other support files and folders

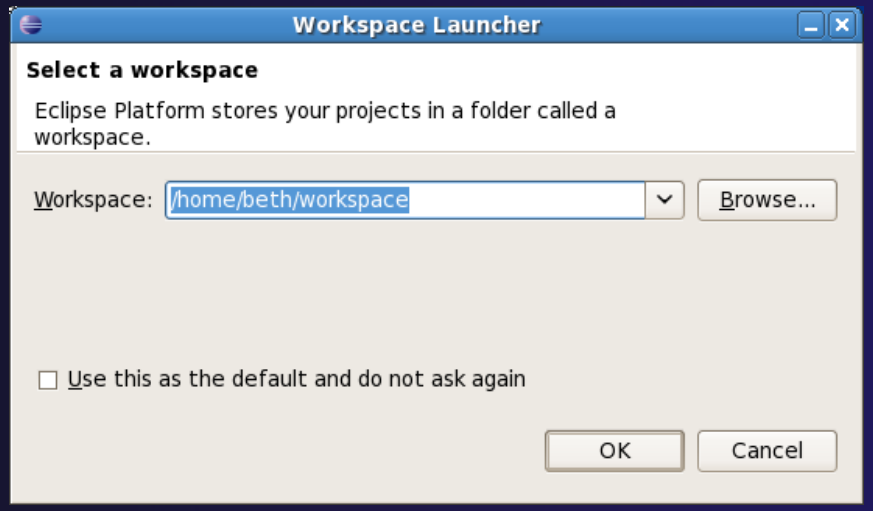
Starting Eclipse

- **Linux**
  - From a terminal window, enter "<eclipse_installation_path>/eclipse/eclipse &"
- **Mac OS X**
  - From finder, open the **eclipse** folder where you installed
  - Double-click on the **Eclipse** application
  - Or from a terminal window
- **Windows**
  - Open the **eclipse** folder
  - Double-click on the **eclipse** executable

Specifying A Workspace

- Eclipse prompts for a workspace location at startup time
- The workspace contains all user-defined data
  - Projects and resources such as folders and files

The prompt can be turned off



Eclipse Welcome Page

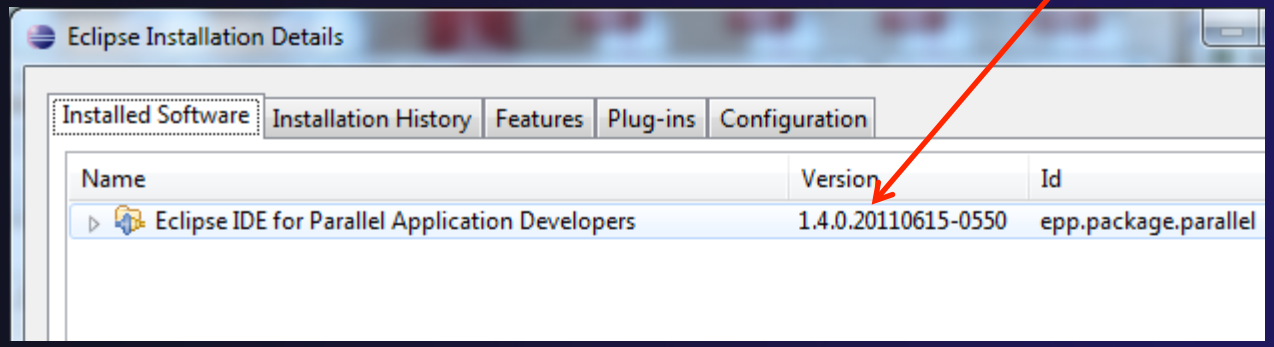
- Displayed when Eclipse is run for the first time

Select "Go to the workbench"

Check Installation Details

- To confirm you have installed OK
  - Mac: **Eclipse>About Eclipse**
  - Others: **Help>About**
- Choose **Installation Details**
- Confirm you have the following installed software

Differs depending on base download

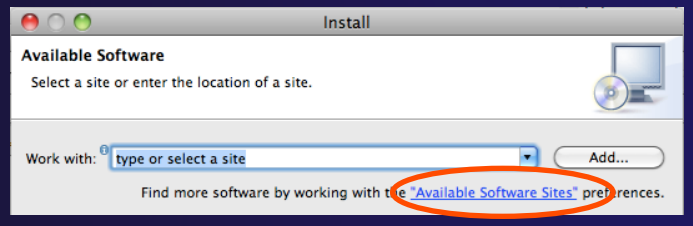


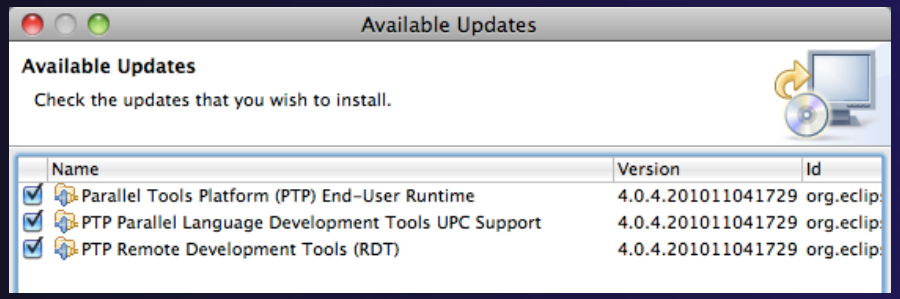
Checking for PTP Updates

- From time-to-time there may be newer PTP releases than the Indigo release
  - Indigo updates are released only in Sept and February
- PTP maintains its own update site with the most recent release
  - Bug fix releases can be more frequent than Indigo'
- You must enable the PTP-specific update site before the updates will be found

Updating PTP

- Enable PTP-specific update site
  - **Help>Install new software**
  - Click **Available Software Sites** link
  - Select checkbox for the PTP site:
    http://download.eclipse.org/tools/ptp/updates/indigo
  - Choose **OK**
  - Choose **Cancel** (to return to Eclipse workbench)
- Now select **Help>Check for updates**
  - Follow prompts like a normal installation





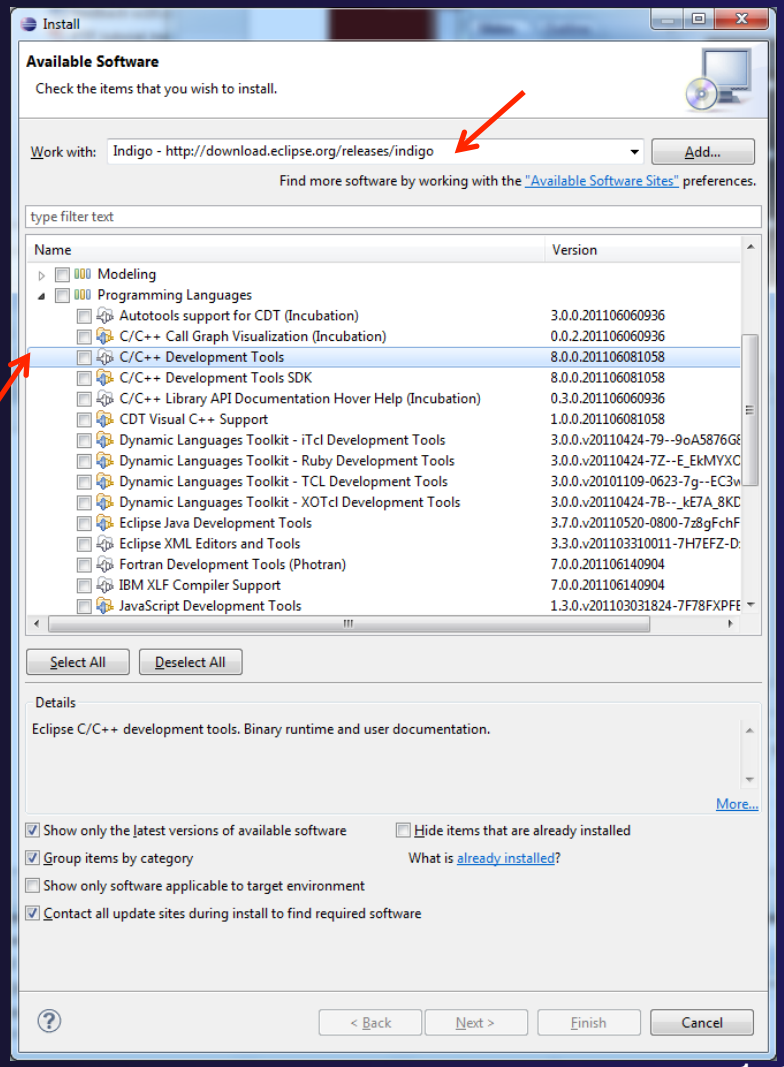
PTP Installation Into Existing Eclipse

- Only required if you're not using Eclipse IDE for Parallel Application Developers bundle
- New functionality is added to Eclipse using *features*
- Features are obtained and installed from
  - An update site on a web server, or
  - A local archive
- Eclipse 3.7 comes preconfigured with a link to the **Indigo** Update Site
  - This is a remote site that contains a large number of official features
  - Indigo projects are guaranteed to work with Eclipse 3.7

Indigo Update Site

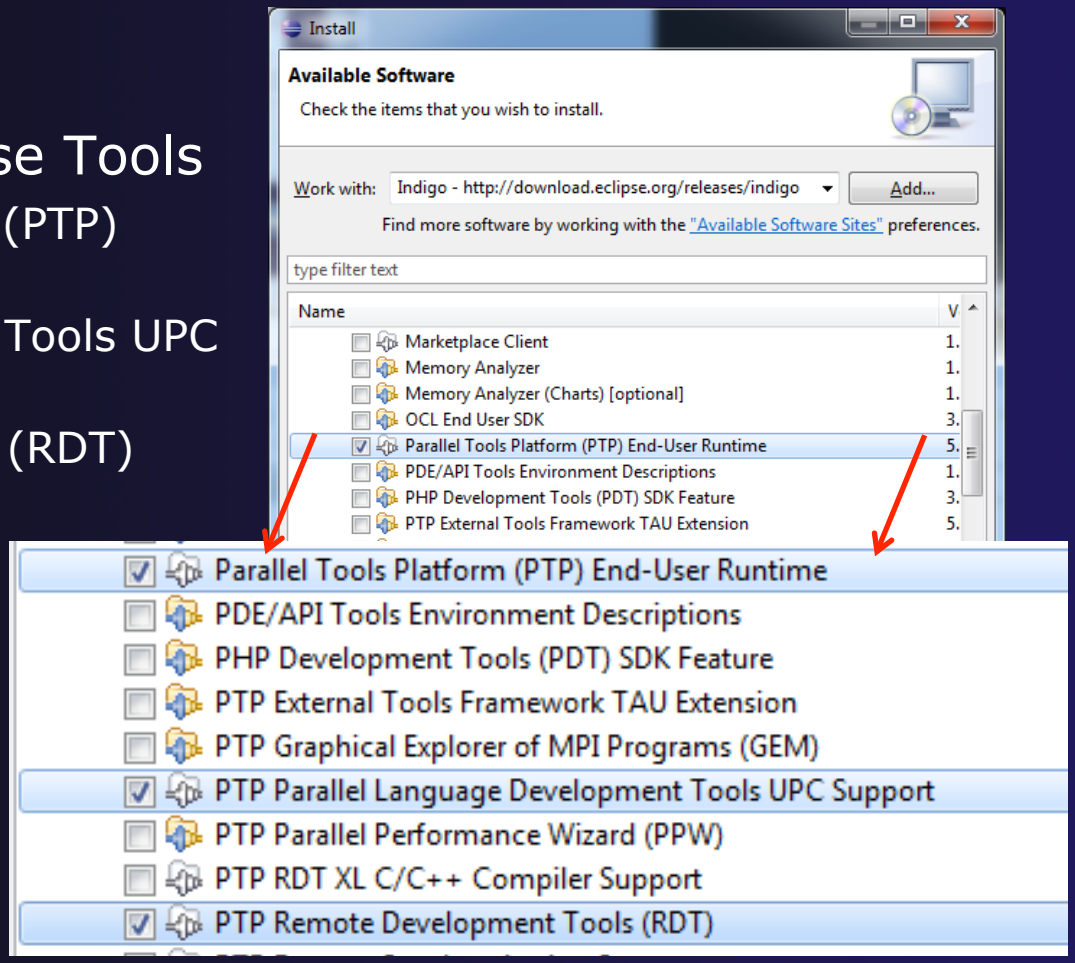
- From the **Help** menu, choose **Install New Software...**
- The Indigo site comes already configured with Eclipse
- We are going to install:
  - C/C++ Development Tools (CDT)*
  - Parallel Tools Platform (PTP) End-User Runtime
  - PTP Remote Development Tools (RDT)

*If you installed the C/C++ IDE, you already have CDT in your Eclipse installation and you can omit this.



Install PTP Features

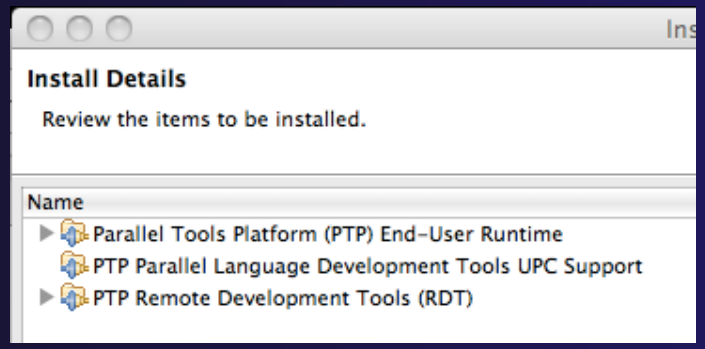
- Under General Purpose Tools
  - Parallel Tools Platform (PTP) End-User Runtime
  - PTP Parallel Lang Dev. Tools UPC Support*
  - PTP Remote Dev Tools (RDT)
- Check these and click 'Next'



* pre-req pulls in optional CDT UPC feature

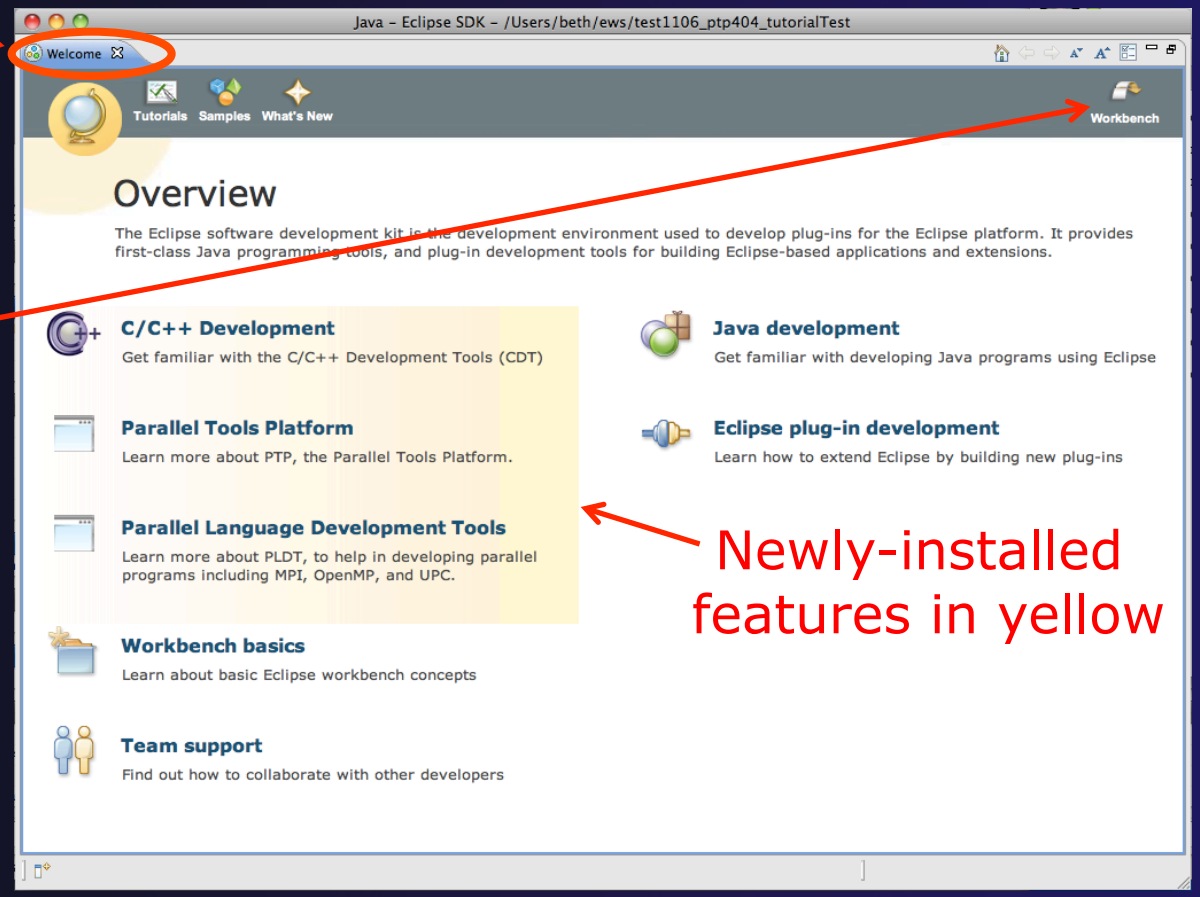
Finishing Installation

- Review the items to be installed
- Finish installing:
  - Choose **Next>**
  - Accept license terms
  - Choose **Finish**
  - Features are downloaded and installed
  - Any pre-requisites are also installed if available
- Restart Eclipse when prompted



Restart after Install

- Welcome page informs you of new features installed
- Click to learn more, or…
- Select workbench icon to go to workbench



Module 2: Introduction

- Objective
  - To introduce the Eclipse platform and PTP
- Contents
  - New and Improved Features
  - What is Eclipse?
  - What is PTP?

New and Improved Features

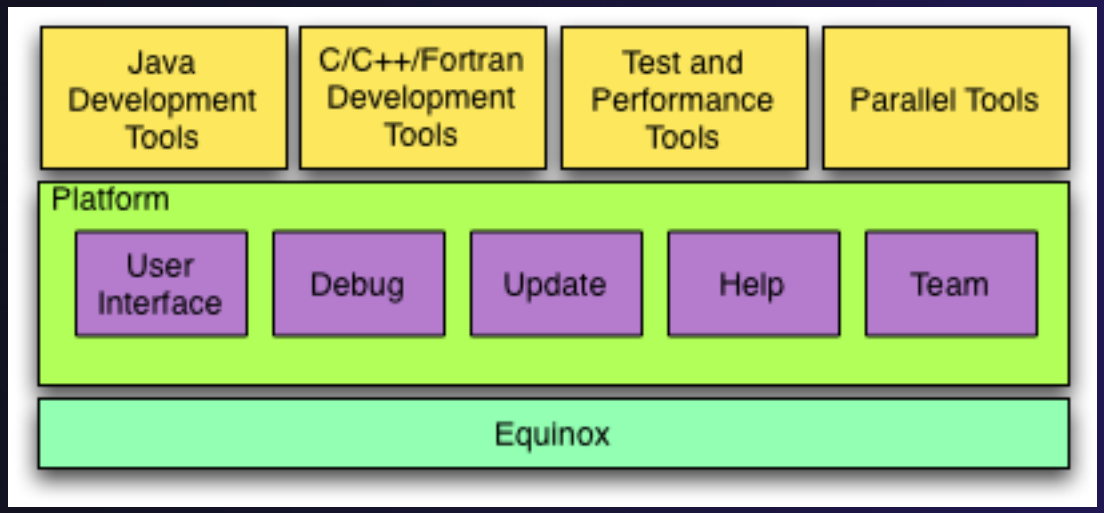
- More flexible projects
  - Synchronized projects overcome many problems of remote projects
  - Allows development when “off-line”
  - Works with non-C/C++ projects
- More customizable resource managers
  - Resource managers can now be added by users
  - Able to have site-specific configurations
  - Interactive launch using job schedulers now supported

New and Improved Features (2)

- Scalable system/job monitoring
  - New perspective allows monitoring of systems of virtually any size
  - View shows location of jobs on cluster
  - Active and inactive jobs views
- Remote support for performance tools
  - External Tools Framework has been extended to support remote systems
  - Performance tools such as TAU can now launch and collect data from remote systems

What is Eclipse?

- A vendor-neutral open-source workbench for multi-language development
- A extensible platform for tool integration
- Plug-in based framework to create, integrate and utilize software tools

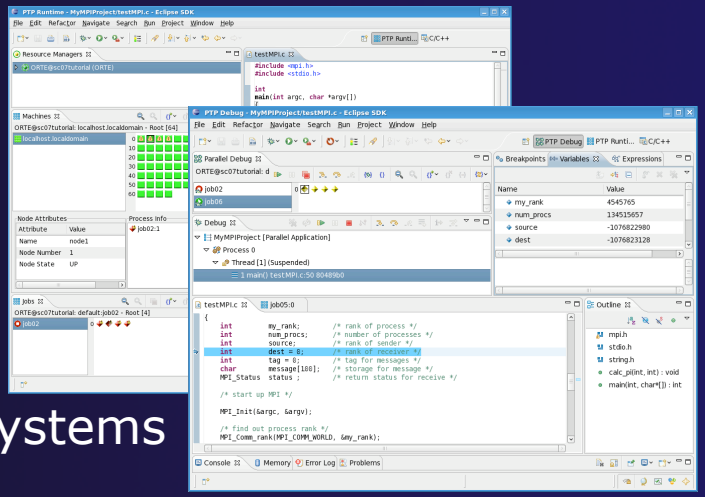


Eclipse Features

- Full development lifecycle support
- Revision control integration (CVS, SVN, Git)
- Project dependency management
- Incremental building
- Content assistance
- Context sensitive help
- Language sensitive searching
- Multi-language support
- Debugging

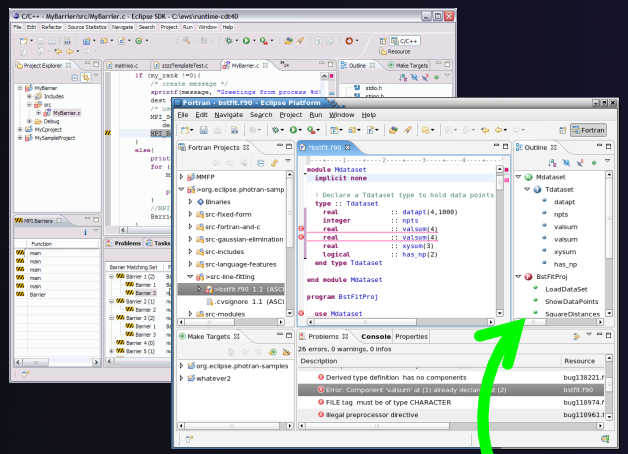
Parallel Tools Platform (PTP)

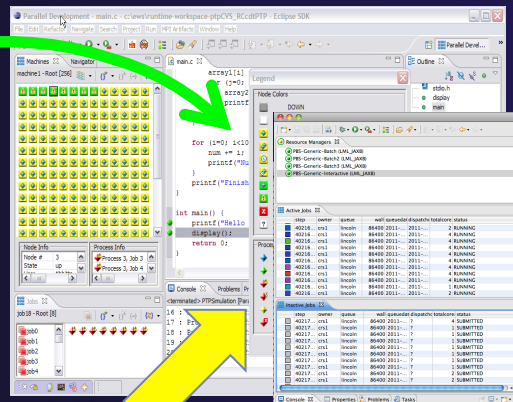
- The Parallel Tools Platform aims to provide a highly integrated environment specifically designed for parallel application development
- Features include:
  - An integrated development environment (IDE) that supports a wide range of parallel architectures and runtime systems
  - A scalable parallel debugger
  - Parallel programming tools (MPI, OpenMP, UPC, etc.)
  - Support for the integration of parallel tools
  - An environment that simplifies the end-user interaction with parallel systems
- http://www.eclipse.org/ptp



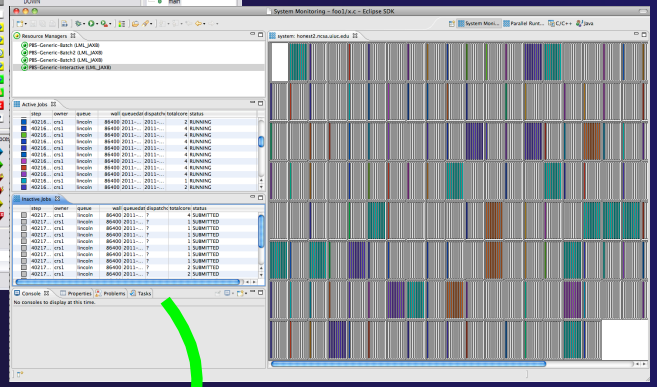
Eclipse PTP Family of Tools

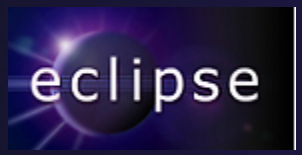
Coding & Analysis
(C, C++, Fortran)

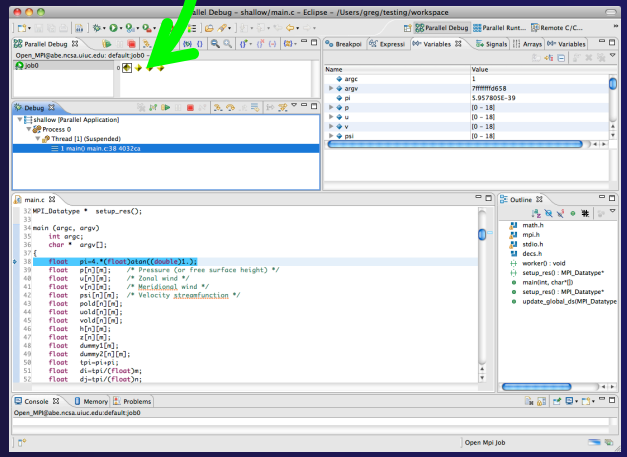




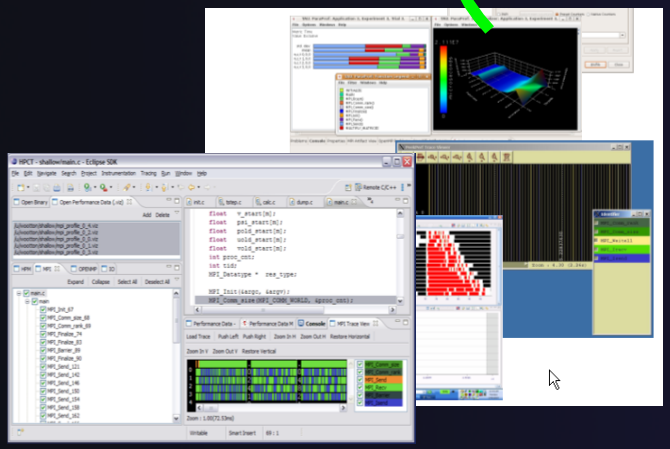
Launching &
Monitoring







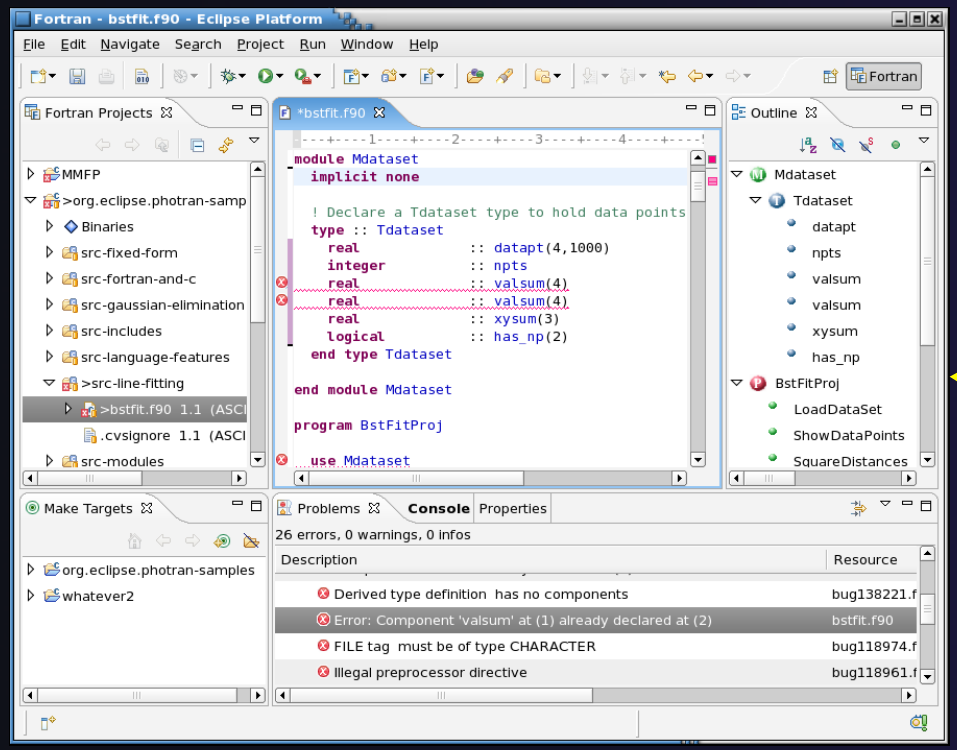
Parallel Debugging

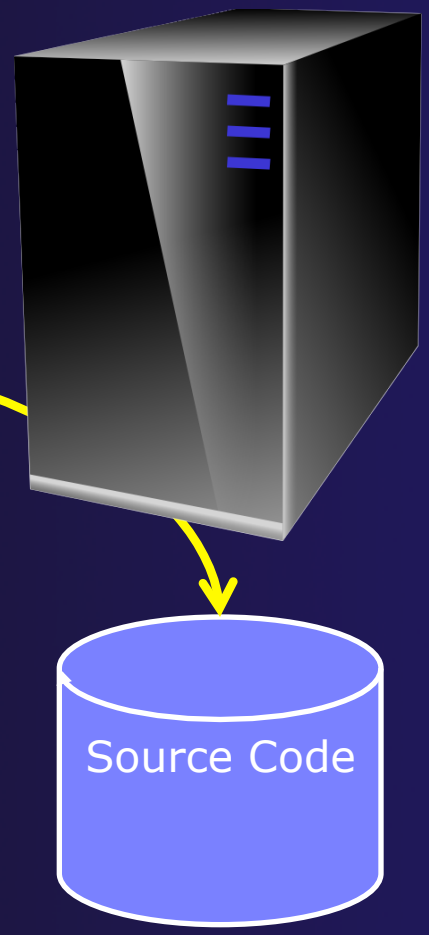


Performance Tuning
(TAU, PPW, …)

How Eclipse is Used

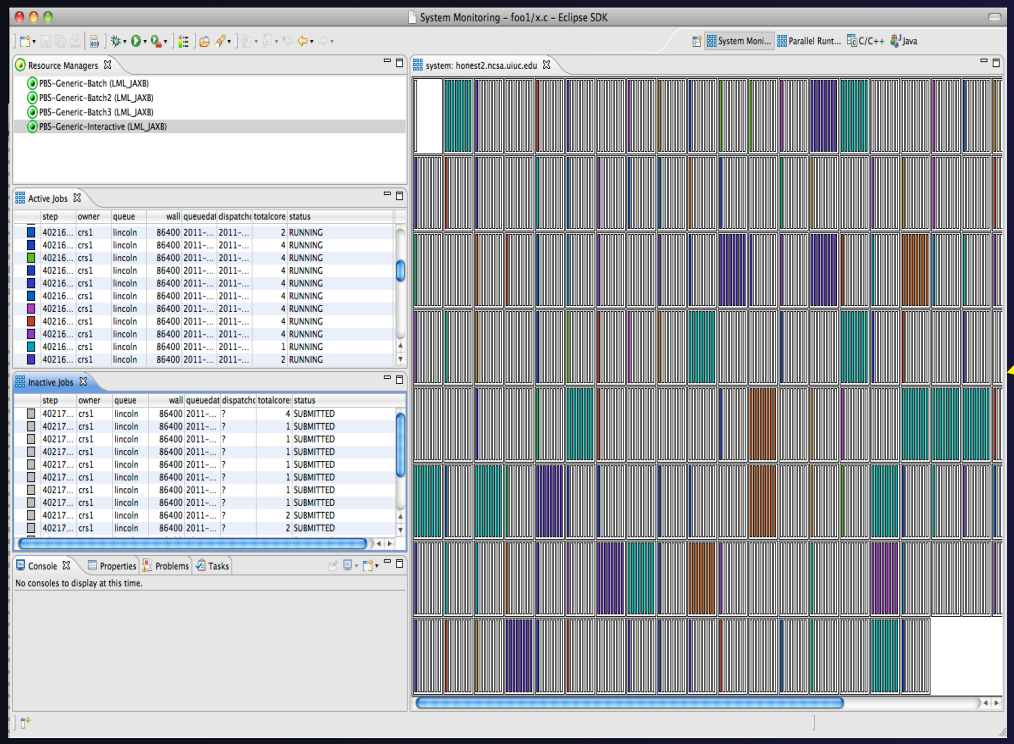
Editing/Compiling

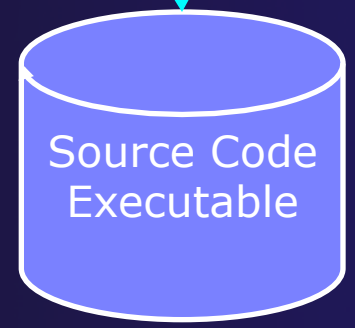




How Eclipse is Used

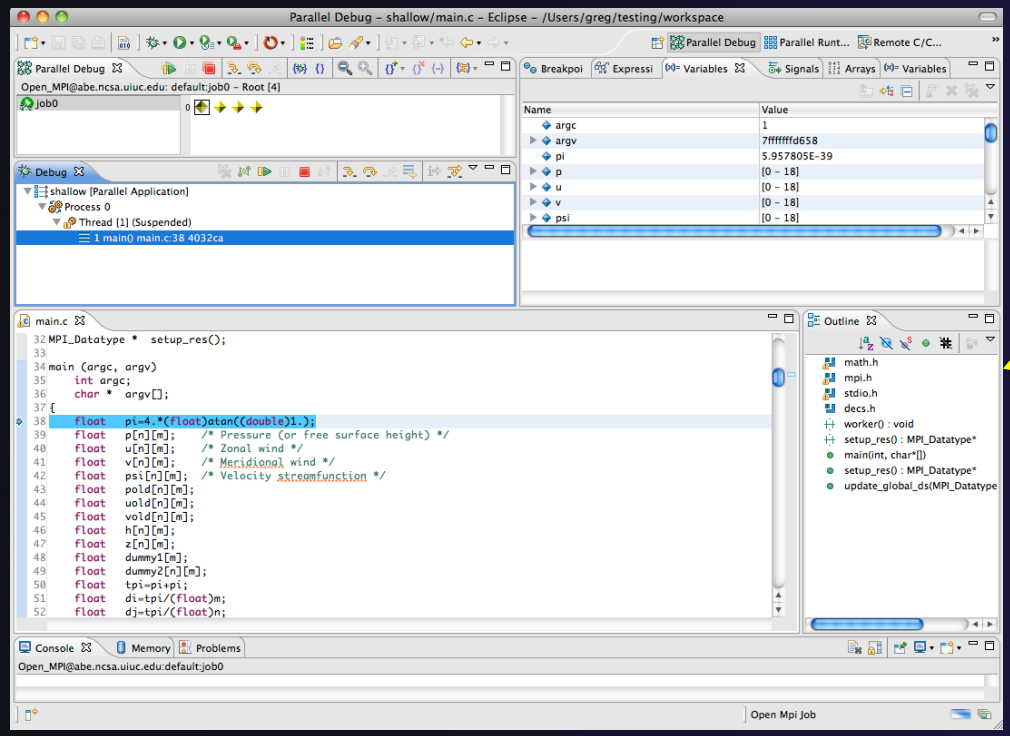
Launching/Monitoring

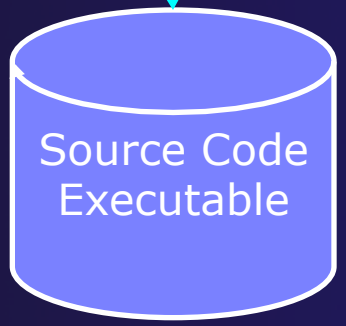




How Eclipse is Used

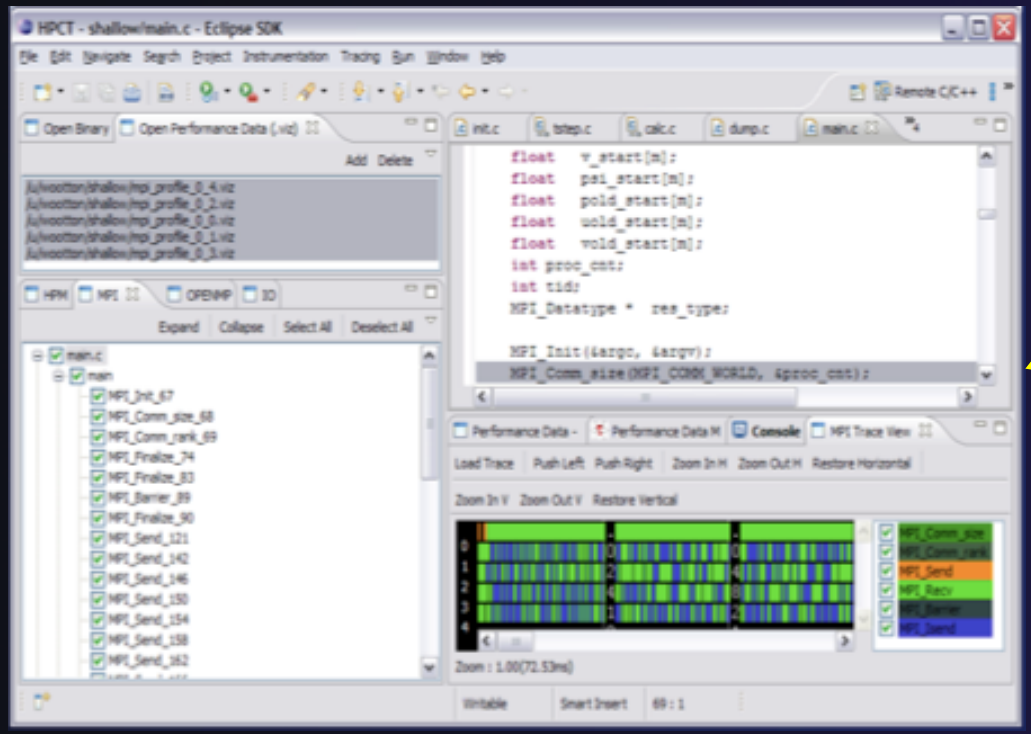
Debugging

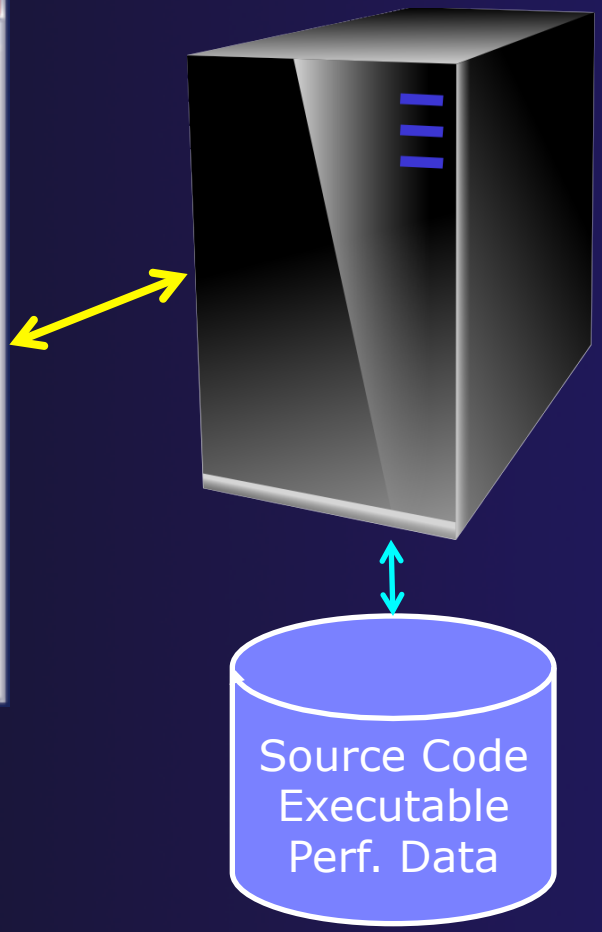




How Eclipse is Used

Performance Tuning





Module 3: Developing with Eclipse

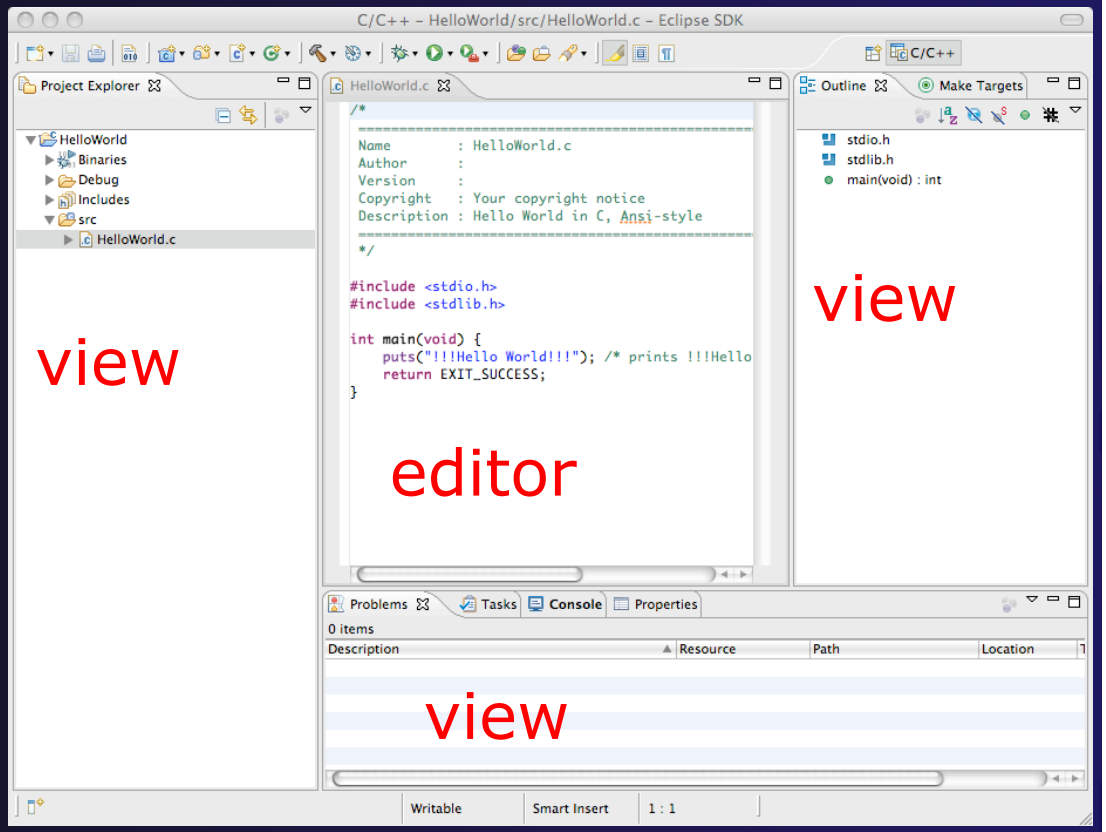
- Objective
  - Learn basic Eclipse concepts: Perspectives, Views, …
  - Learn about local, remote, and synchronized projects
  - Learn how to create and manage a C project
  - Learn about Eclipse editing features
  - Learn about Eclipse Team features
  - Learn about MPI features
  - Learn how to build and launch an MPI program on a remote system
  - Learn about Fortran projects
  - Learn about searching, refactoring, etc.

Contents

Basic Eclipse Features

Eclipse Basics

- A *workbench* contains the menus, toolbars, editors and views that make up the main Eclipse window
- The workbench represents the desktop development environment
  - Contains a set of tools for resource mgmt
  - Provides a common way of navigating through the resources
- Multiple workbenches can be opened at the same time
- Only one workbench can be open on a *workspace* at a time

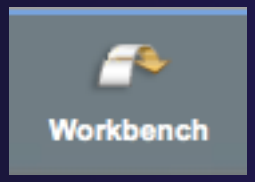


perspective

Perspectives

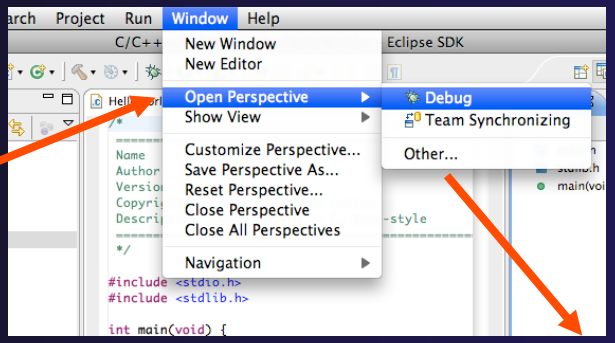
- Perspectives define the layout of views and editors in the workbench
- They are *task oriented*, i.e. they contain specific views for doing certain tasks:
  - There is a **Resource Perspective** for manipulating resources
  - **C/C++ Perspective** for manipulating compiled code
  - **Debug Perspective** for debugging applications
- You can easily switch between perspectives

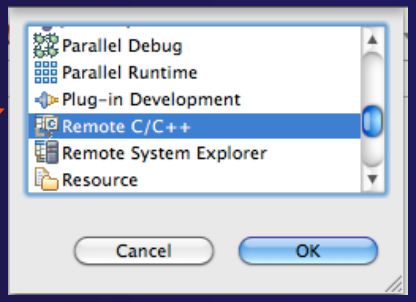
- If you are on the Welcome screen now, select "Go to Workbench" now

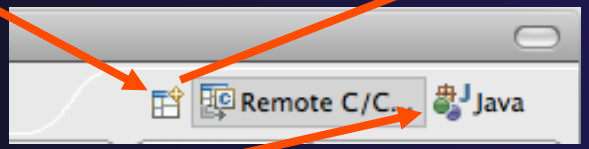


Switching Perspectives

- Three ways of changing perspectives
  - Choose the **Window>Open Perspective** menu option
  - Then choose **Other...**
  - Click on the **Open Perspective** button in the upper right corner of screen
  - Click on a perspective shortcut button
- Switch perspective on next slide…

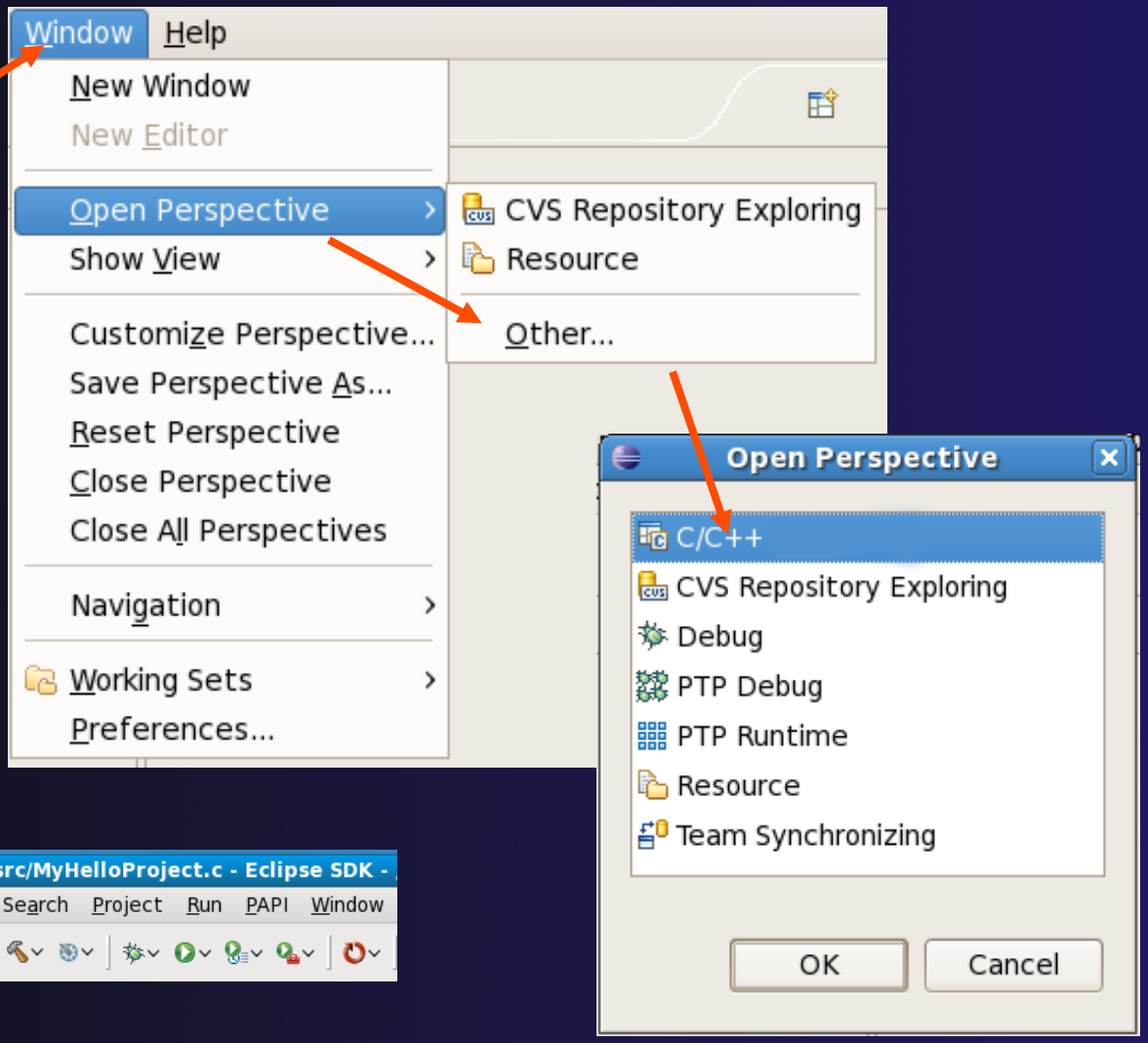


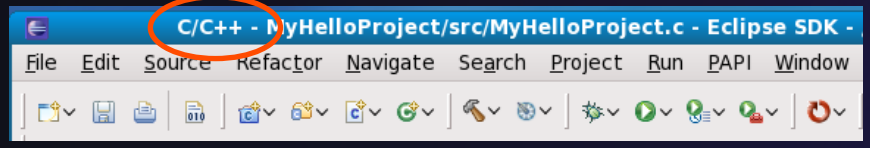




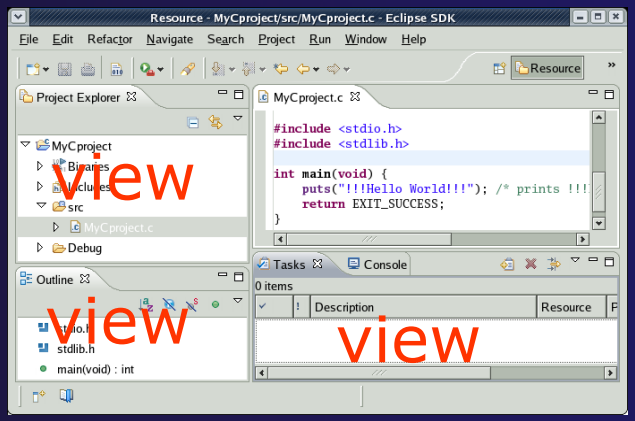
Switch to C/C++ Perspective

- Only needed if you're not already in the perspective
- What Perspective am in in? See Title Bar

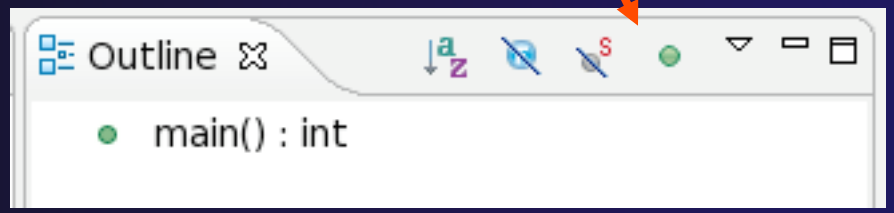




Views

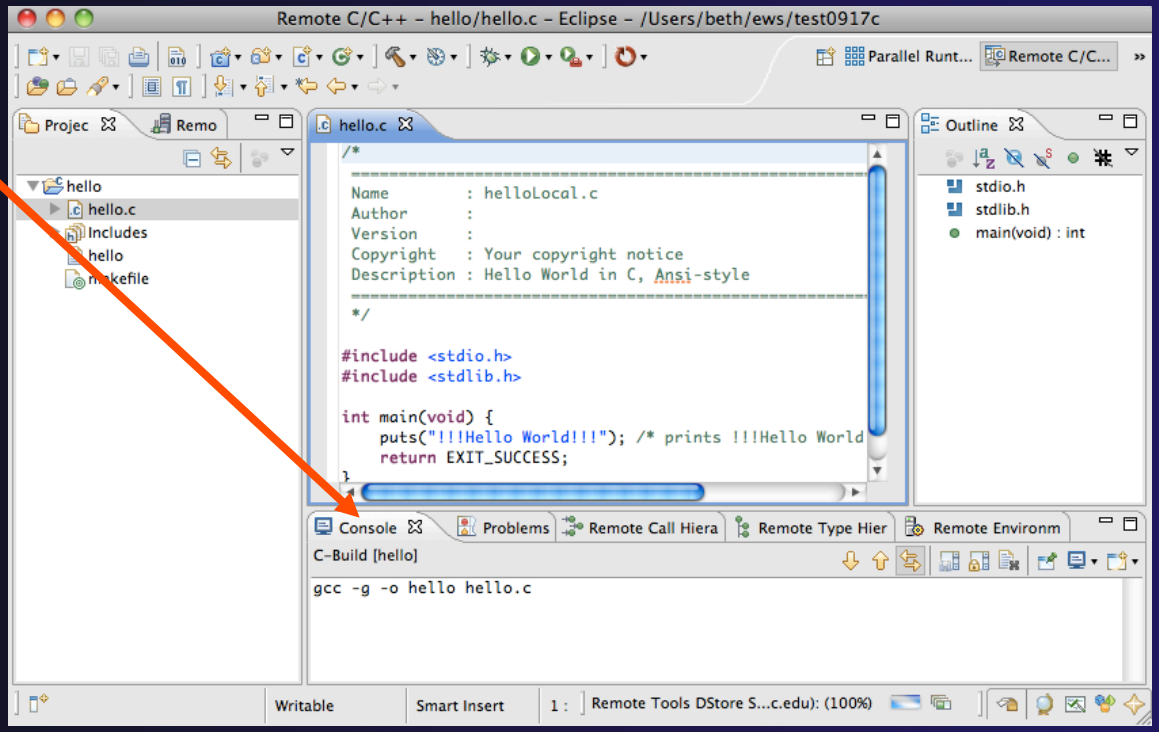


- The workbench window is divided up into Views
- The main purpose of a view is:
  - To provide alternative ways of presenting information
  - For navigation
  - For editing and modifying information
- Views can have their own menus and toolbars
  - Items available in menus and toolbars are available only in that view
  - Menu actions only apply to the view
- Views can be resized



Stacked Views

- Stacked views appear as tabs
- Selecting a tab brings that view to the foreground

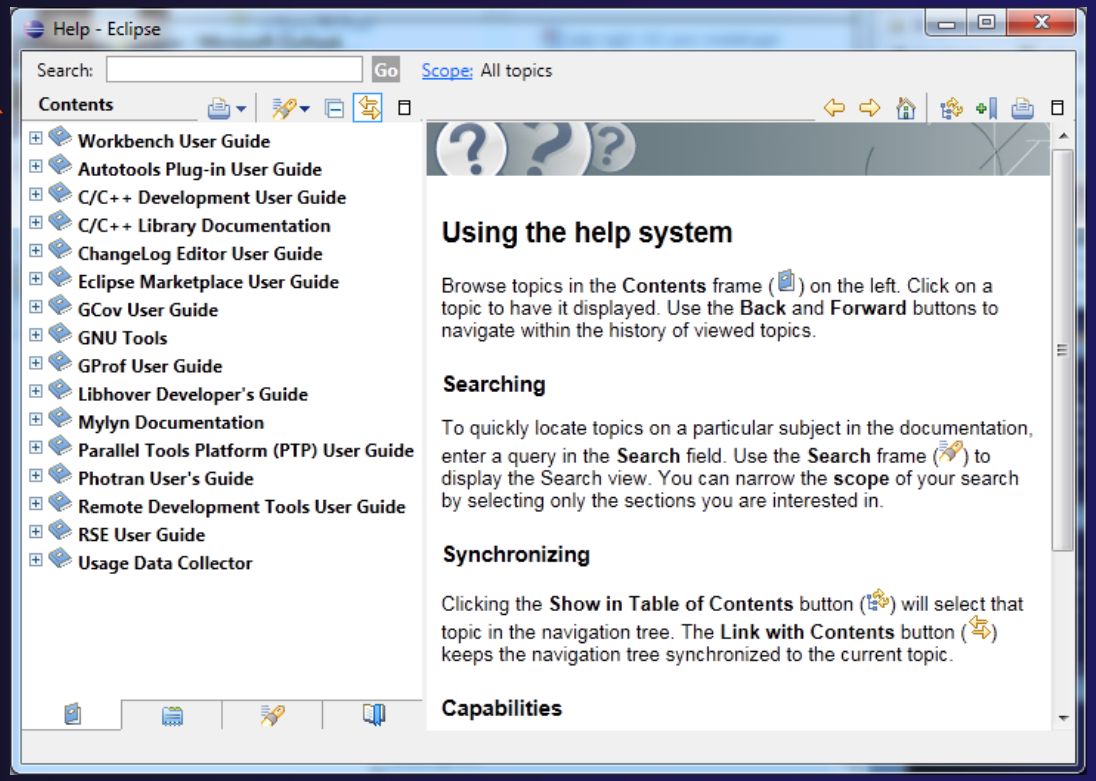


Expand a View

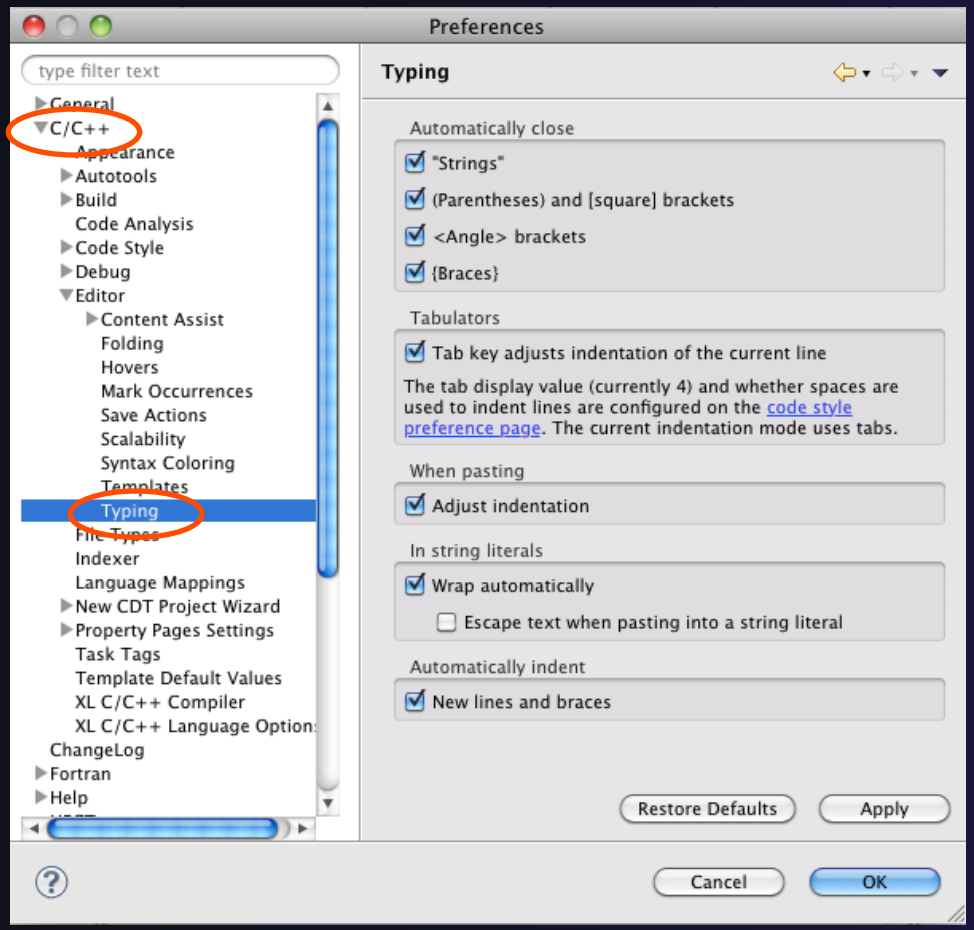
- Placeholder
- Double-click on a view/editor's tab to fill the workbench with its content; dclick again to return to original size

Help

- To access help
  - **Help>Help Contents**
  - **Help>Search**
  - **Help>Dynamic Help**
- **Help Contents** provides detailed help on different Eclipse features *in a browser*
- **Search** allows you to search for help locally, or using Google or the Eclipse web site
- **Dynamic Help** shows help related to the current context (perspective, view, etc.)

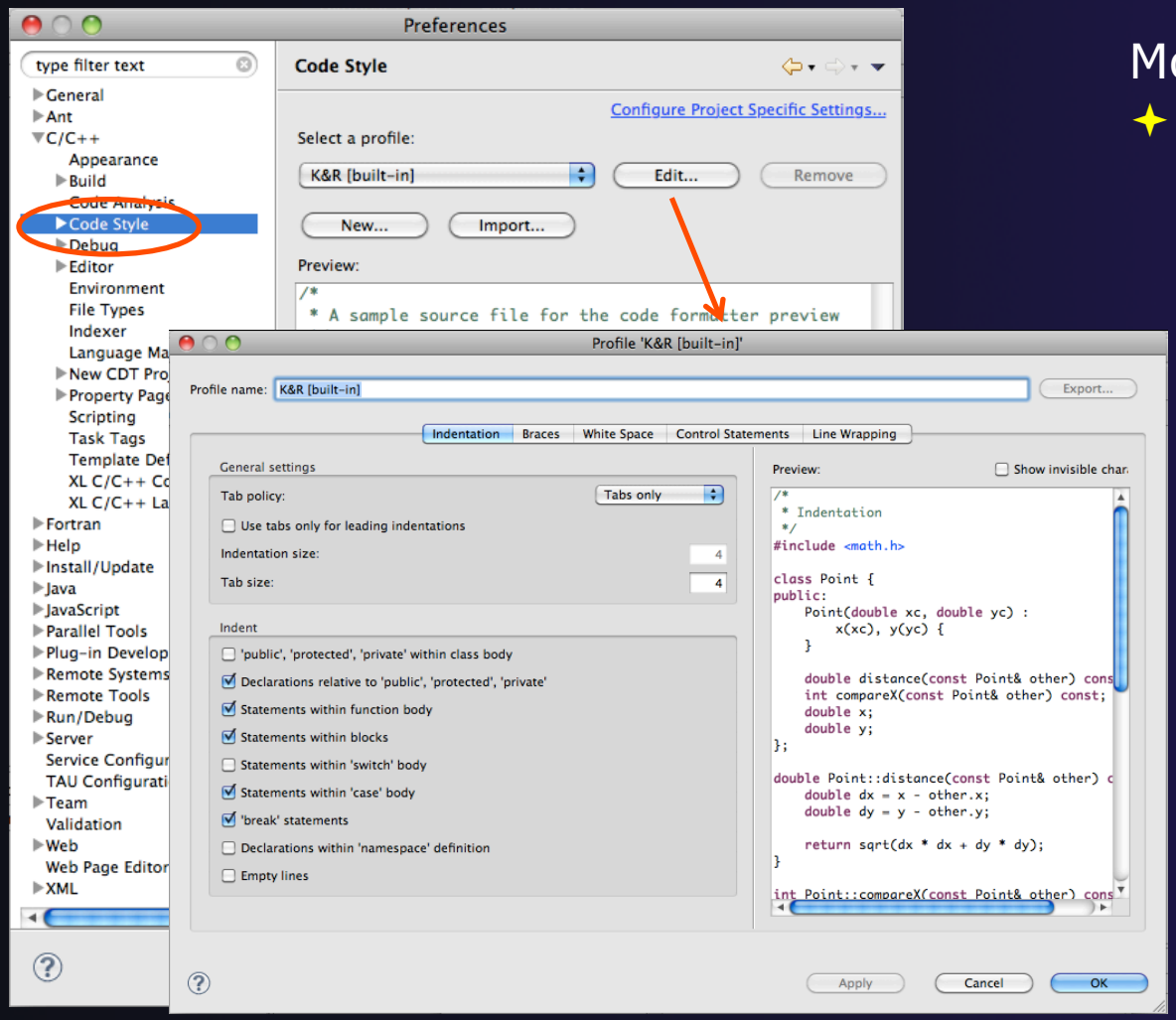


Eclipse Preferences



- Eclipse Preferences allow customization of almost everything
- To open use
  - Mac: **Eclipse>Preferences…**
  - Others: **Window>Preferences…**
- The C/C++ preferences allow many options to be altered
- In this example you can adjust what happens in the editor as you type.

Preferences Example



More C/C++ preferences:

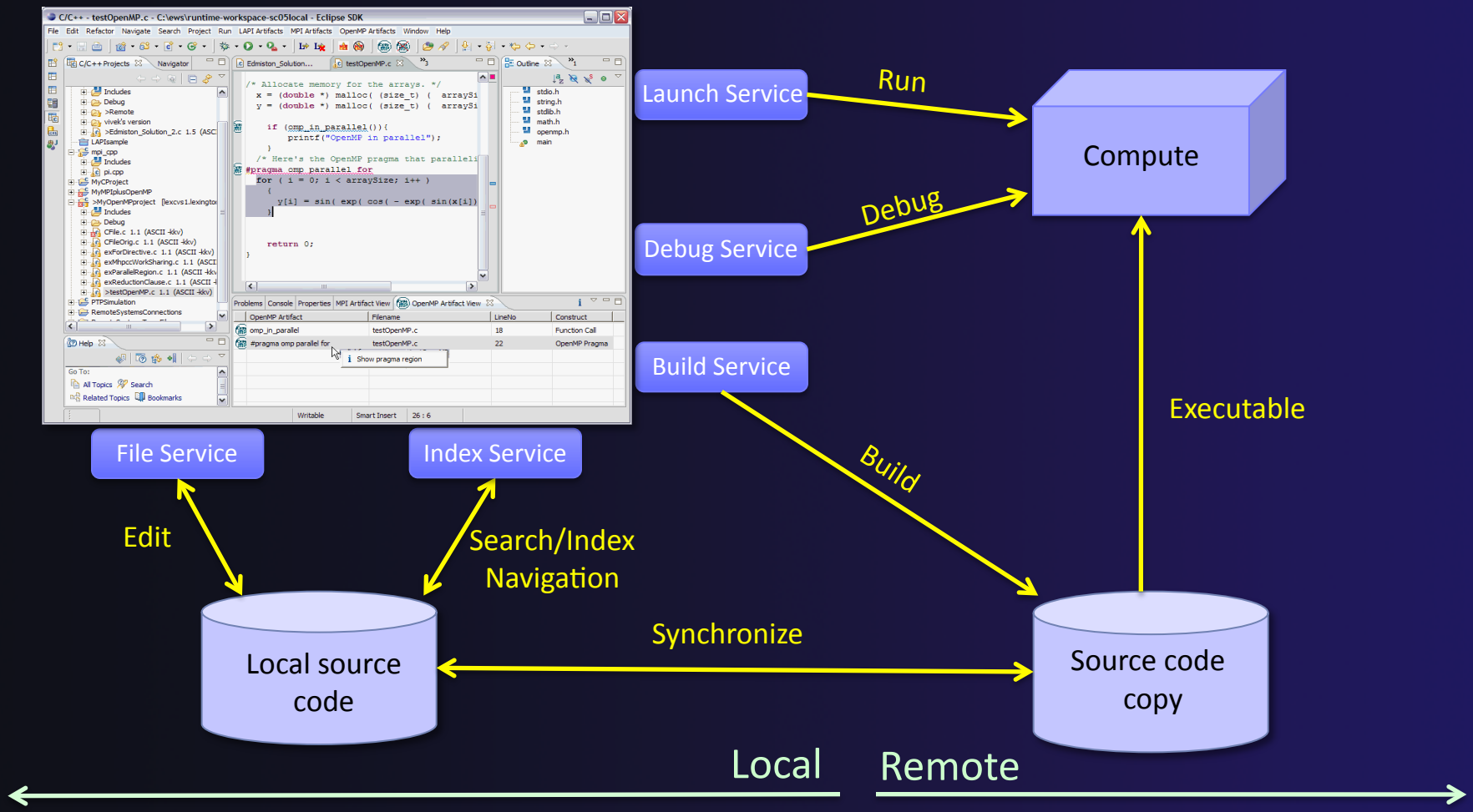
- In this example the Code Style preferences are shown
  - These allow code to be automatically formatted in different ways

Projects In Eclipse

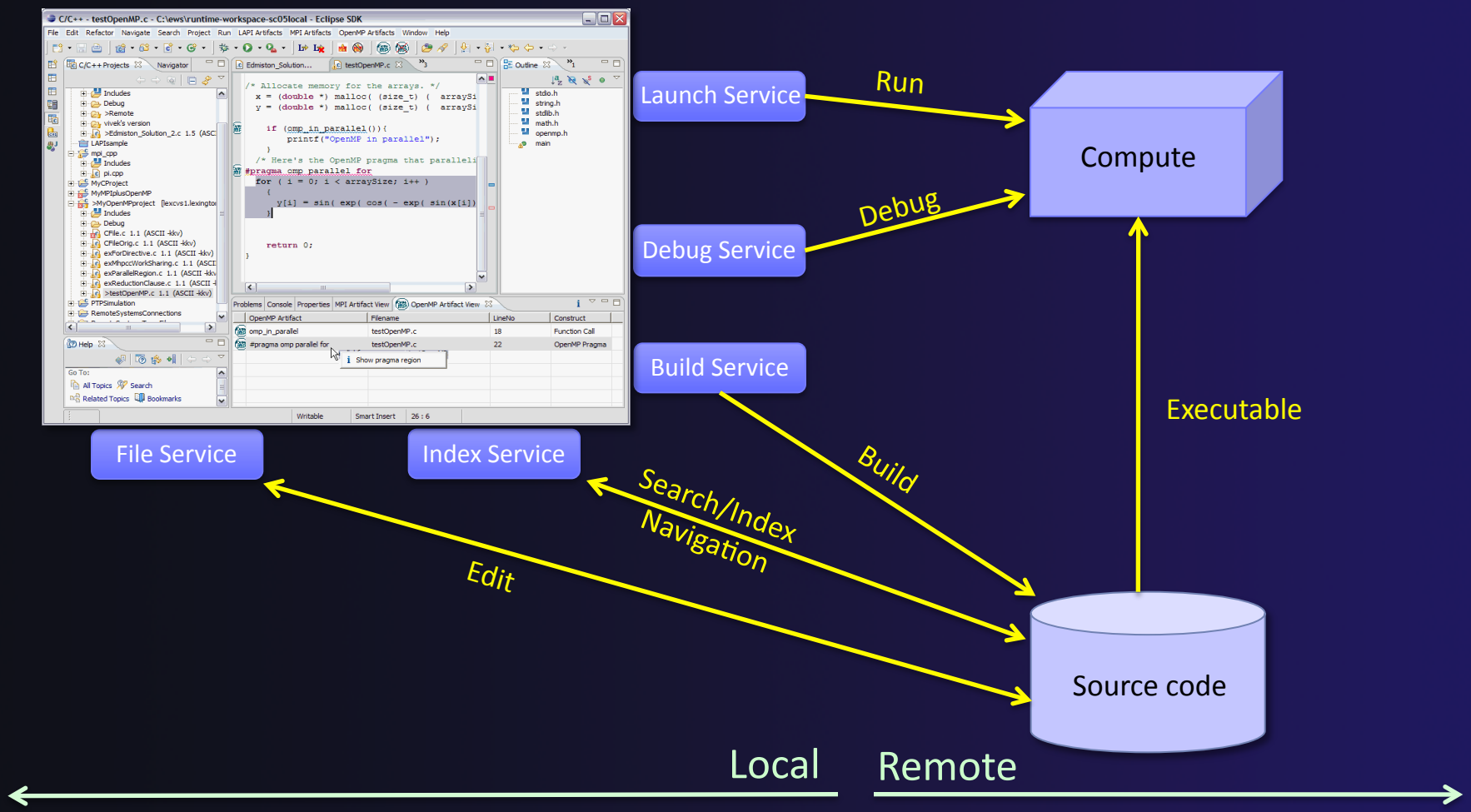
Project Types

- Local
  - Source is located on local machine, builds happen locally
- Synchronized
  - Source is local, then synchronized with remote machine
  - Building and launching happens remotely
- Remote
  - Source is located on remote machine, build and launch takes place on remote machine

Synchronized Projects



Remote Projects

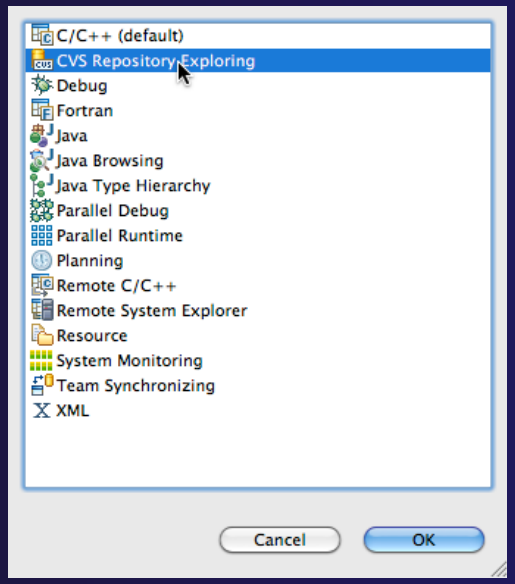


C, C++, and Fortran Projects

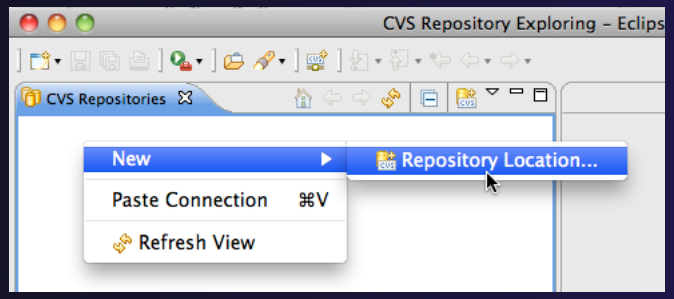
- Makefile-based
  - Project contains its own makefile (or makefiles) for building the application
- Managed
  - Eclipse manages the build process, no makefile required
- Parallel programs can be run on local machine or on a remote system
  - MPI needs to be installed
  - An application built locally probably can't be run on a remote machine unless their architectures are the same

Importing a Project from CVS

- Switch to **CVS Repository Exploring** perspective

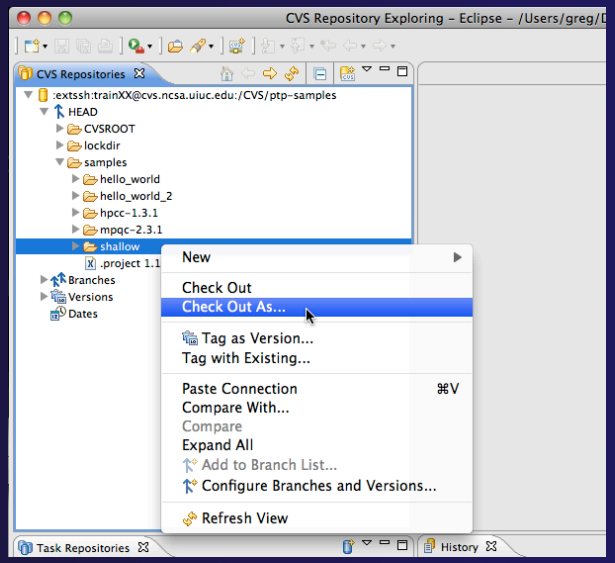


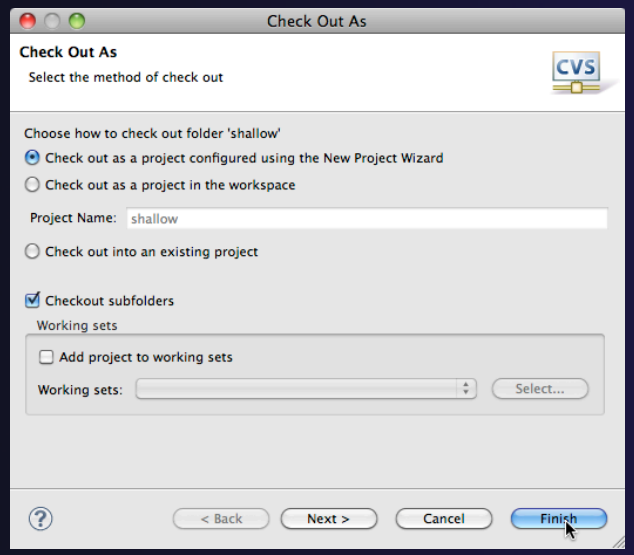
- Right click in **CVS Repositories** view and select **New>Repository Location...**



Checking out the Project

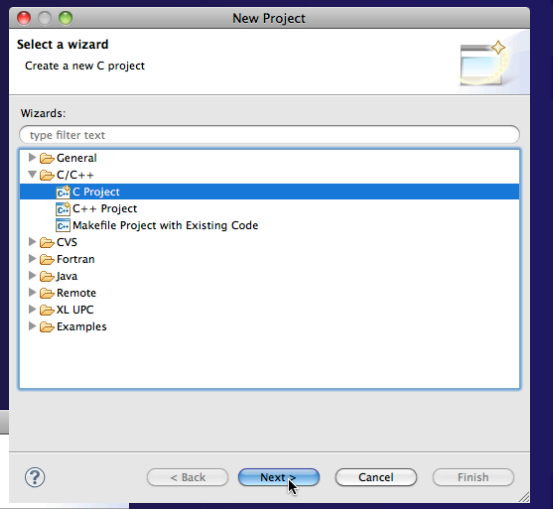
- Expand the repository location
- Expand **HEAD>samples**
- Right click on **shallow** and select **Check Out As…**
- On **Check Out As** dialog, select **Finish**



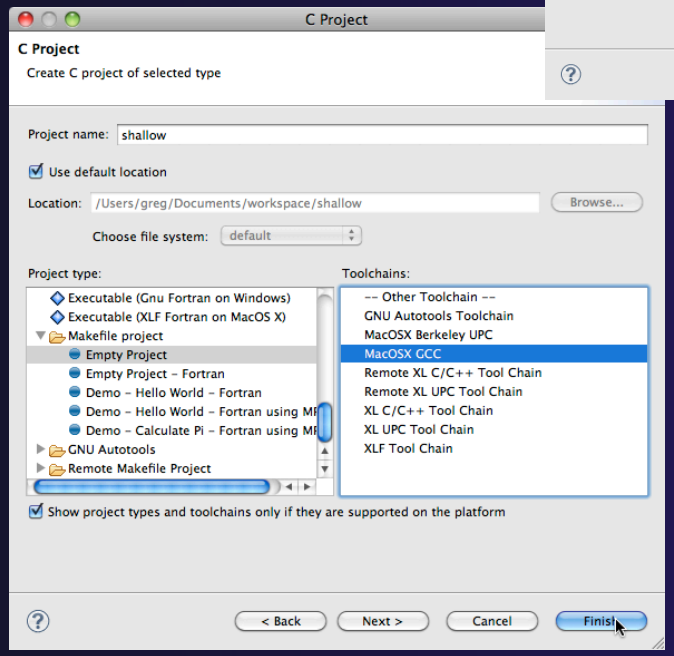


New Project Wizard

- Expand **C/C++**
- Select **C Project** and click on **Next>**

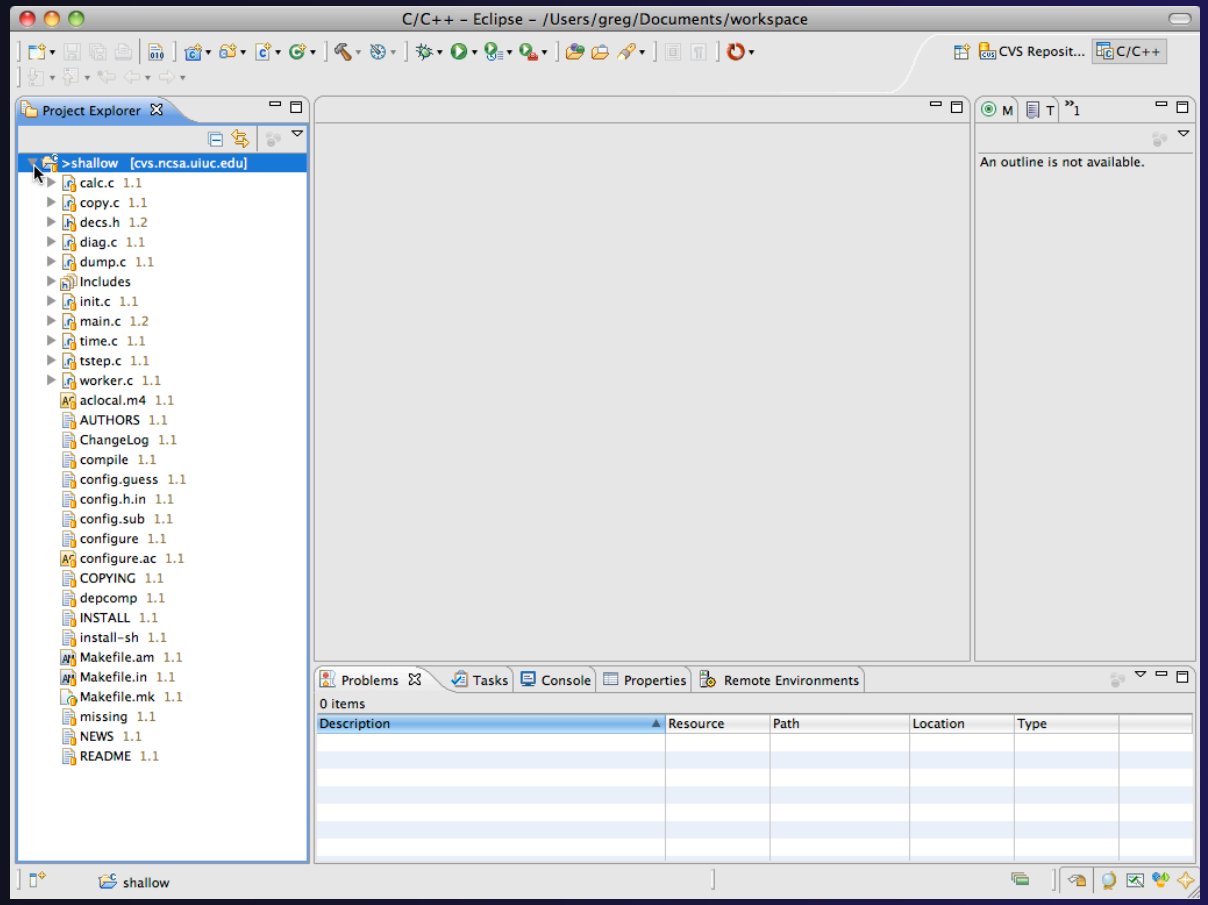


- Enter 'shallow' as **Project Name**
- Expand **Makefile project** in **Project Types**
- Select **Empty Project**
- Select a toolchain that matches your system from **Toolchains**
- Click on **Finish**



C/C++ Perspective

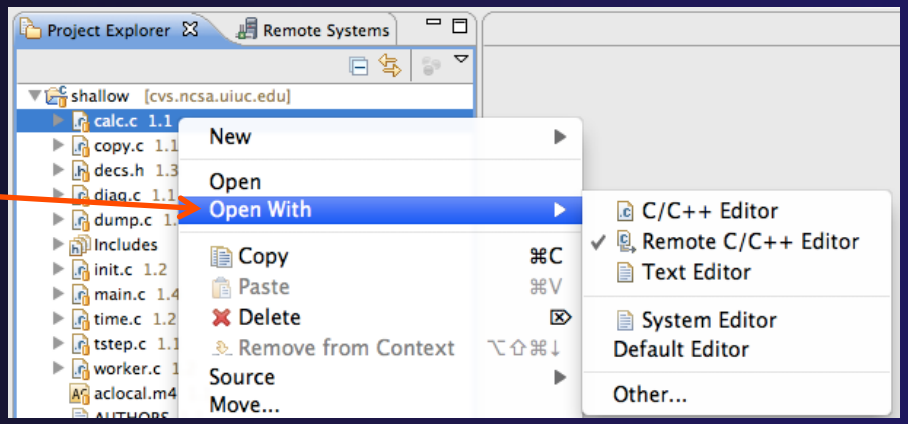
- You should now see the “shallow” project in your workspace



Editor Features

Editors

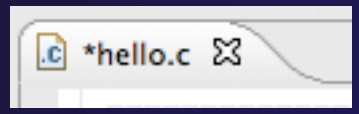
- An editor for a resource (e.g. a file) opens when you double-click on a resource
- The type of editor depends on the type of the resource
  - .c files are opened with the C/C++ editor by default
  - Use **Open With** to use another editor



  - Some editors do not just edit raw text
- When an editor opens on a resource, it stays open across different perspectives
- An active editor contains menus and toolbars specific to that editor

Saving File in Editor

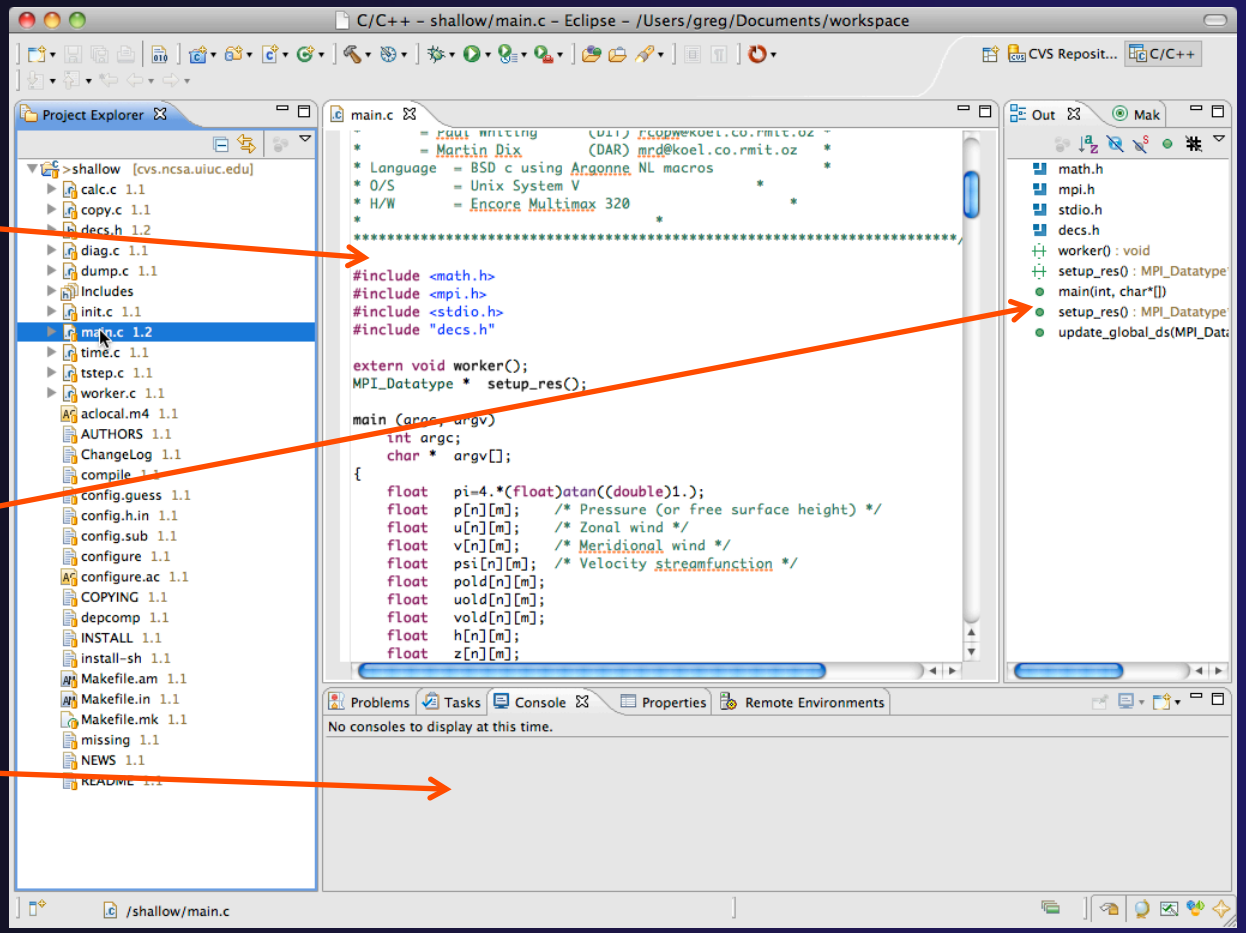
- When you change a file in the editor, an asterisk on the editor's title bar indicates unsaved changes



- Save the changes by using Command/Ctrl-S or **File>Save**

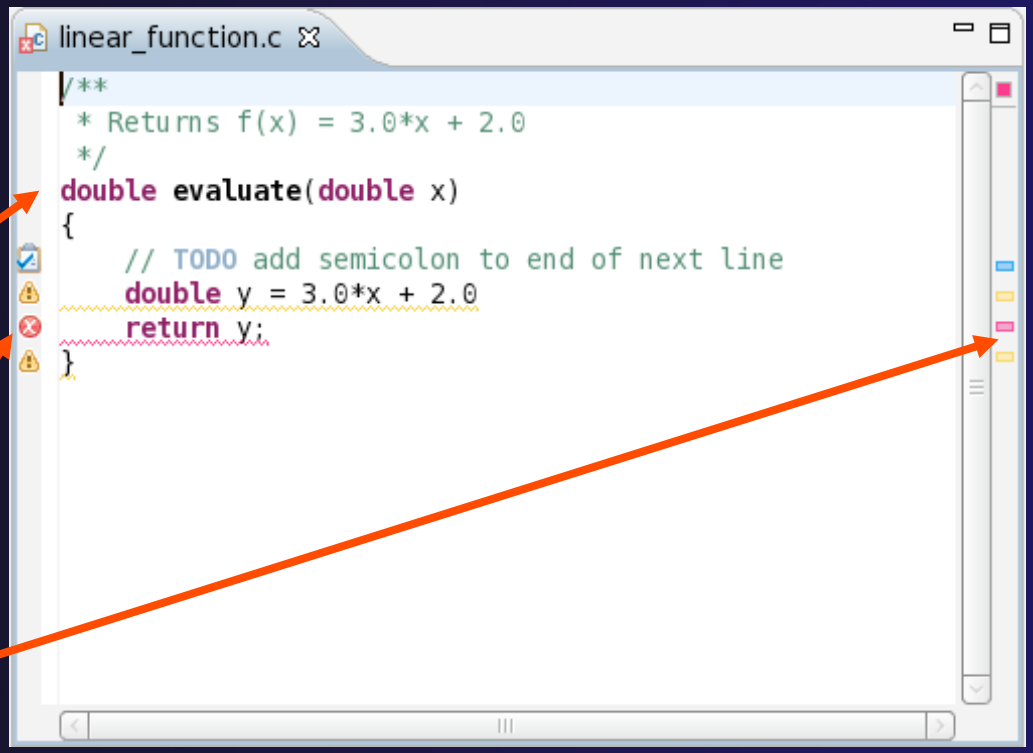
Editor and Outline View

- Double-click on source file
- Editor will open in main view
- Outline view is shown for file in editor
- Console shows results of build, local runs, etc.



Source Code Editors & Markers

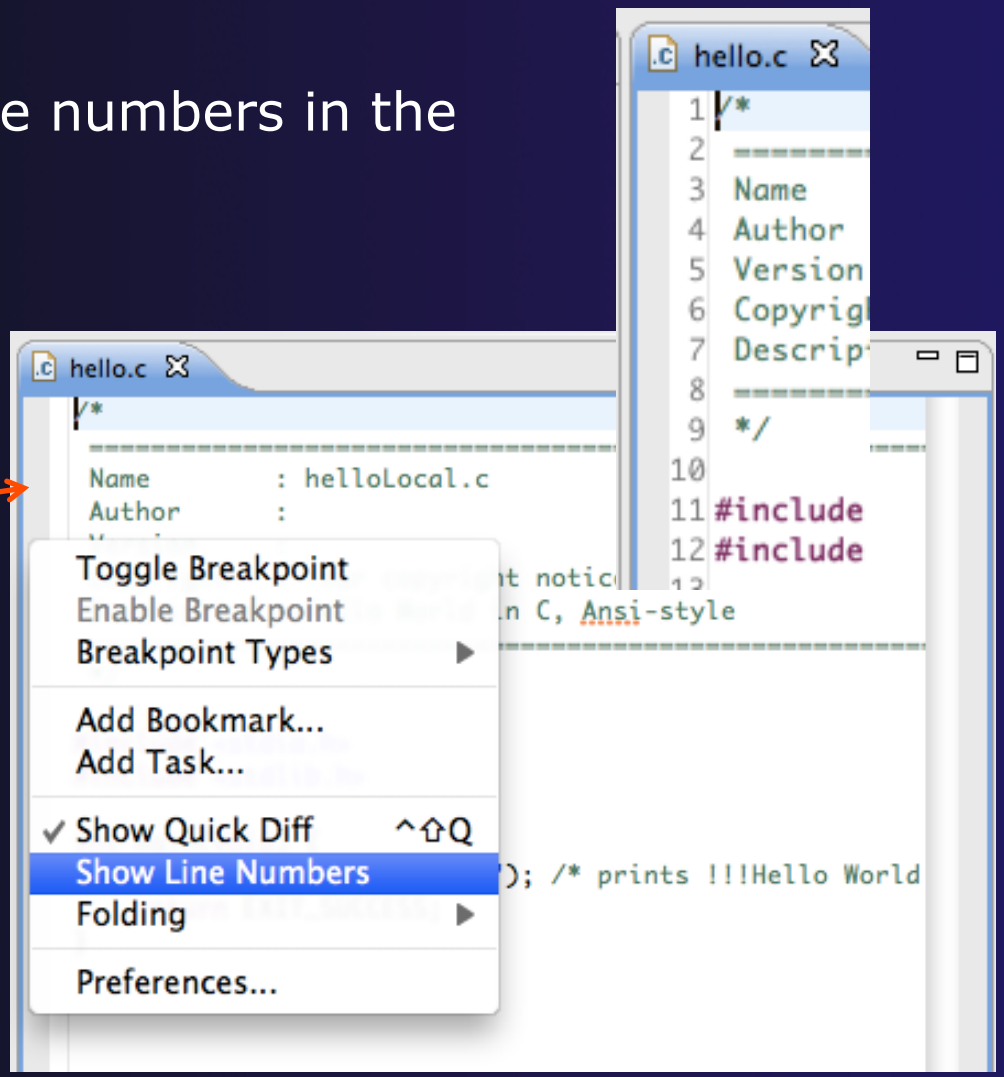
- A source code editor is a special type of editor for manipulating source code
- Language features are highlighted
- Marker bars for showing
  - Breakpoints
  - Errors/warnings
  - Task Tags, Bookmarks
- Location bar for navigating to interesting features in the entire file



Icons:

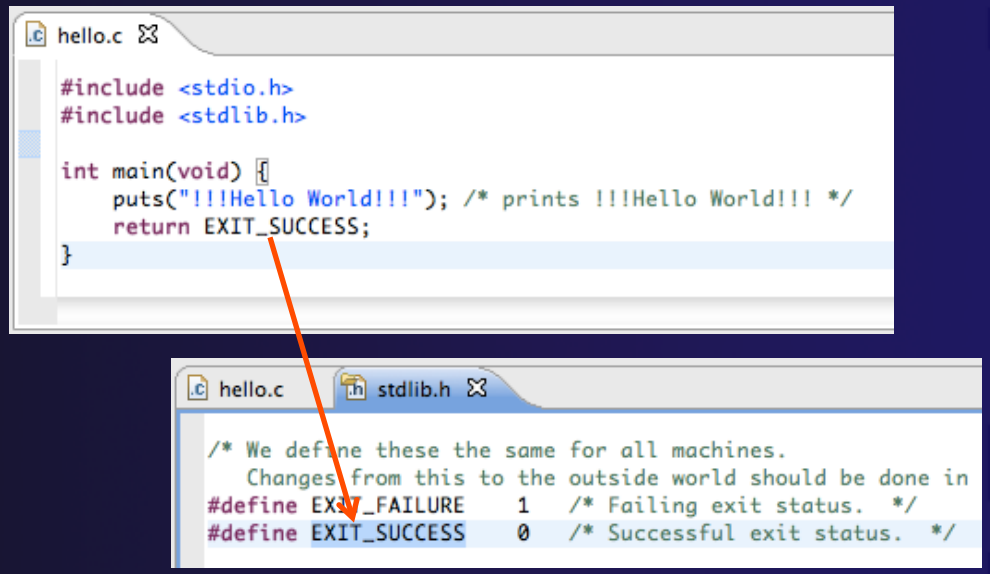
Line Numbers

- Text editors can show line numbers in the left column
- To turn on line numbering:
  - Right-mouse click in the editor marker bar
  - Click on **Show Line Numbers**

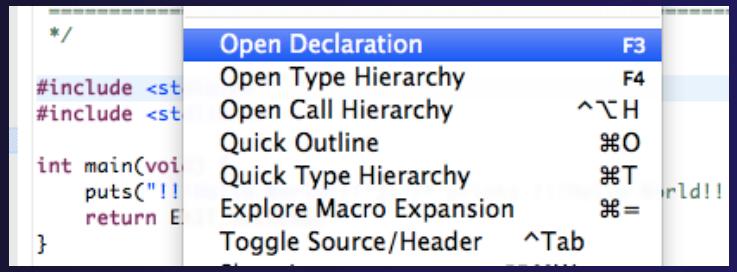


Navigating to Other Files

- On demand hyperlink
  - Hold down Command/Ctrl key
  - Click on element to navigate to its definition in the header file (Exact key combination depends on your OS)
  - E.g. Command/Ctrl and click on EXIT_SUCCESS



- Open declaration
  - Right-click and select **Open Declaration** will also open the file in which the element is declared
  - E.g. right-click on stdio.h and select **Open Declaration**



Content Assist & Templates

- Type an incomplete function name e.g. “get” into the editor, and hit **ctrl-space**
- Select desired completion value with cursor or mouse

```
13
14 int main(void) {
15     puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
16     get
17
18     ret
19 }
20
```

getdelim(char * * __lineptr,* __n,int __delimit
getenv(const char * __name) : char *
getline(char * * __lineptr,* __n,FILE * __strear
Press '^Space' to show Template Propos

- Code Templates: type ‘for’ and Ctrl-space

Hit ctrl-space again for code templates

```
17     for
18         for - for loop                          for (int var = 0; var < max; ++var) {
19     ret     for - for loop with temporary variable
20 }                                               }
21
```

Inactive code

- Inactive code will appear grayed out in the CDT editor

```
260 #define VAL
261 #ifdef VAL
262     acopy_one_to_two(VAL, ds, res.indx);
263 #else
264     acopy_one_to_two(res.row, ds, res.indx);
265 #endif
```

```
260 //#define VAL
261 #ifdef VAL
262     acopy_one_to_two(VAL, ds, res.indx);
263 #else
264     acopy_one_to_two(res.row, ds, res.indx);
265 #endif
```

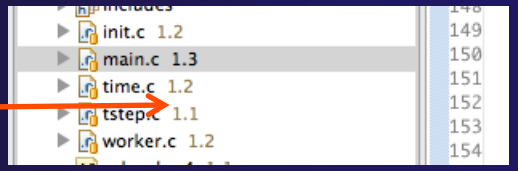
Team Features

"Team" Features

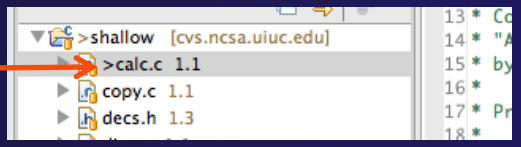
- Eclipse supports integration with multiple revision control systems (RCS)
  - CVS, SVN, Git, and others
  - Collectively known as "Team" services
- Many features are common across RCS
  - Compare/merge
  - History
  - Check-in/check-out
- Some differences
  - Version numbers
  - Branching

CVS Features

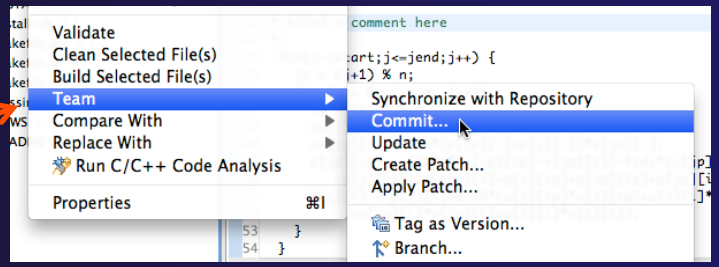
- Shows version numbers next to each resource



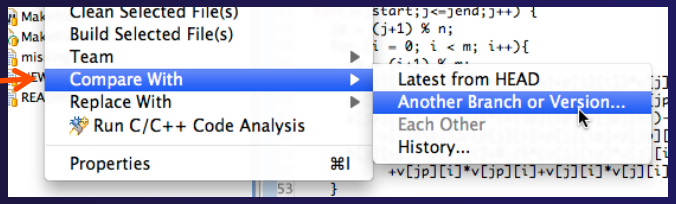
- Marks resources that have changed
  - Can also change color (preference option)



- Context menu for Team operations



- Compare to latest, another branch, or history

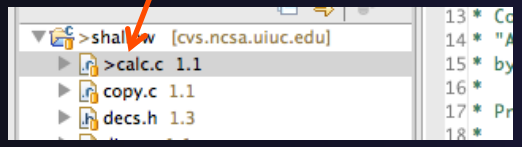


- Synchronize whole project (or any selected resources)

File Modification

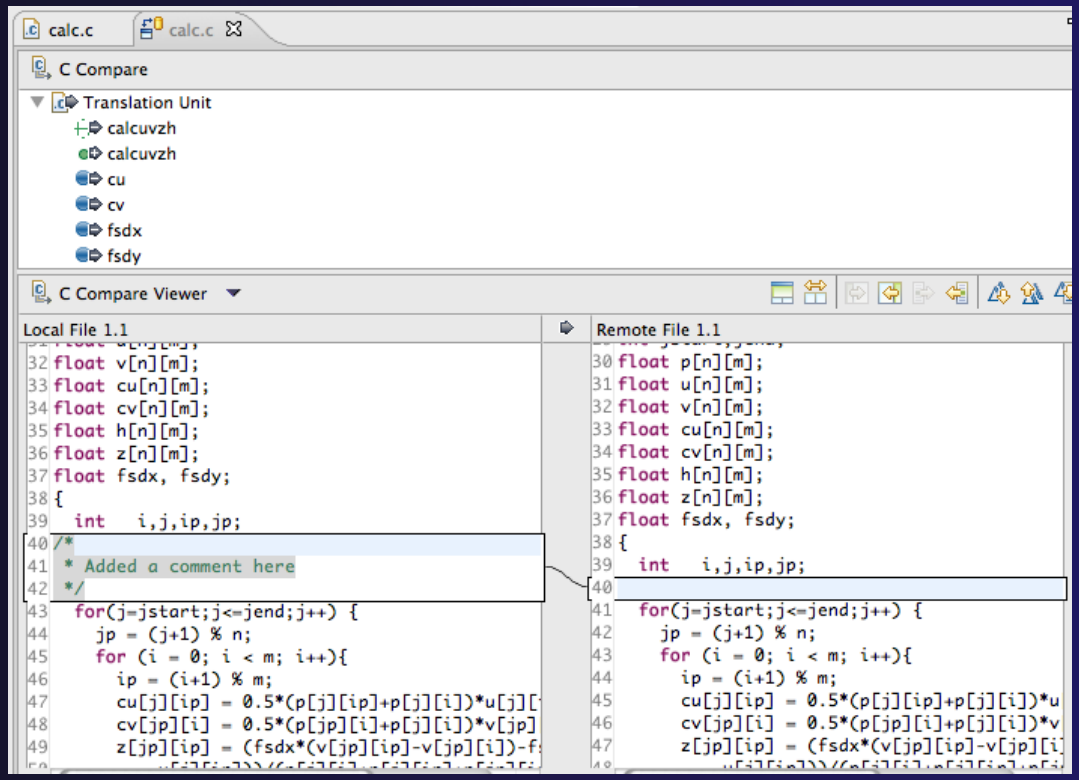
- Open “calc.c”
- Add comment at line 40
- Save file
- File will be marked to indicate that is has been modified

```
27
28 void calcuvzh(jstart,jend,p,u,v,cu,cv,h,z,fsdx,fsdy)
29 int jstart,jend;
30 float p[n][m];
31 float u[n][m];
32 float v[n][m];
33 float cu[n][m];
34 float cv[n][m];
35 float h[n][m];
36 float z[n][m];
37 float fsdx, fsdy;
38 {
39   int   i,j,ip,jp;
40 /*
41  * Added a comment here
42  */
43   for(j=jstart;j<=jend;j++) {
44     jp = (j+1) % n;
45     for (i = 0; i < m; i++){
46       ip = (i+1) % m;
47       cu[j][ip] = 0.5*(p[j][ip]+p[j][i])*u[j][ip];
48       cv[jp][i] = 0.5*(p[jp][i]+p[j][i])*v[jp][i];
49       z[jp][ip] = (fsdx*(v[jp][ip]-v[jp][i])-fsdy*(u[jp][ip]
50          -u[j][ip]))/(p[j][i]+p[j][ip]+p[jp][ip]+p[jp][i]);
51       h[j][i] = p[j][i]+0.25*(u[j][ip]*u[j][ip]+u[j][i]*u[j][i]
52            +v[jp][i]*v[jp][i]+v[j][i]*v[j][i]);
53     }
```



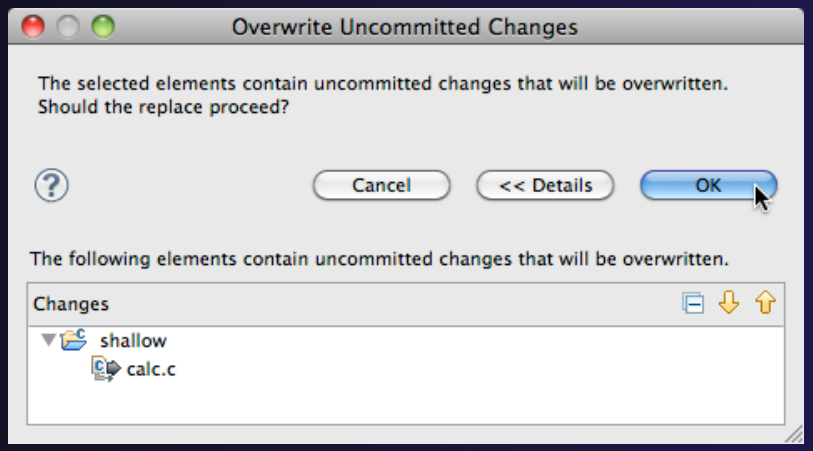
View Changes

- Right-click on "calc.c" and select **Compare With>Latest from HEAD**
- Compare editor will open showing differences between local (changed) file and the original
- Buttons allow changes to be merged from right to left
- Can also navigate between changes using buttons



Revert To The Latest Version

- Right-click on the "shallow" project and select **Replace With>Latest from HEAD**
- Review the resources that will be replaced, then click **OK**



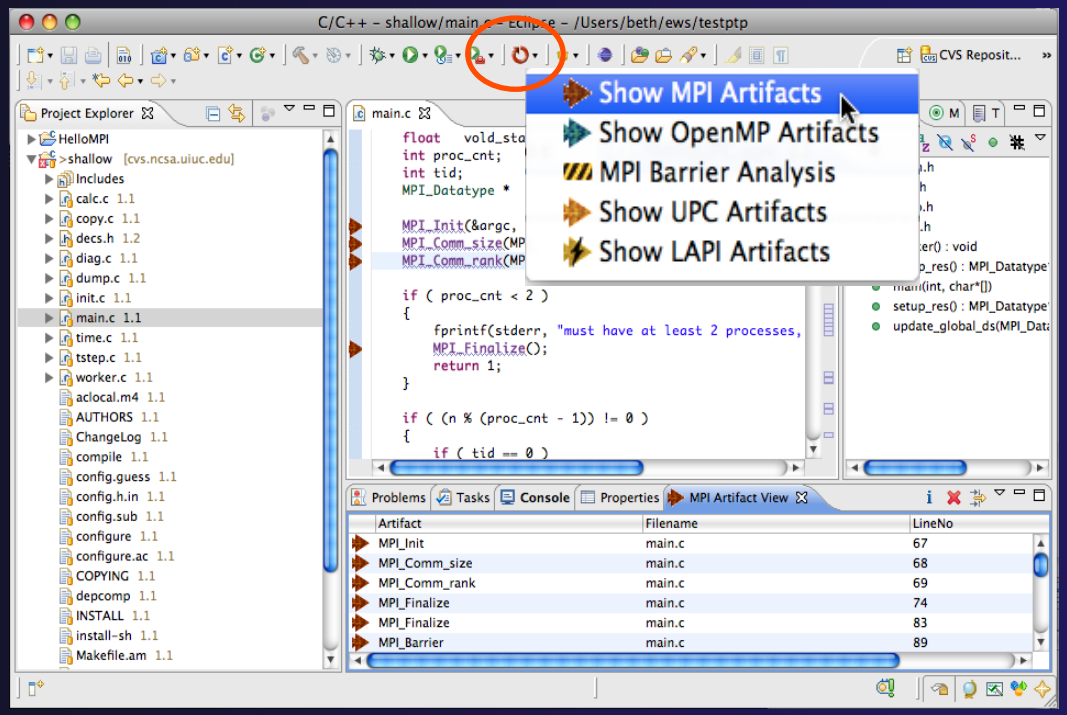
MPI Features

MPI-Specific Features

- PTP's Parallel Language Development Tools (PLDT) has several features specifically for developing MPI code
  - Show MPI Artifacts
  - Code completion
  - Context Sensitive Help for MPI
  - Hover Help
  - MPI Templates in the editor
  - MPI Barrier Analysis

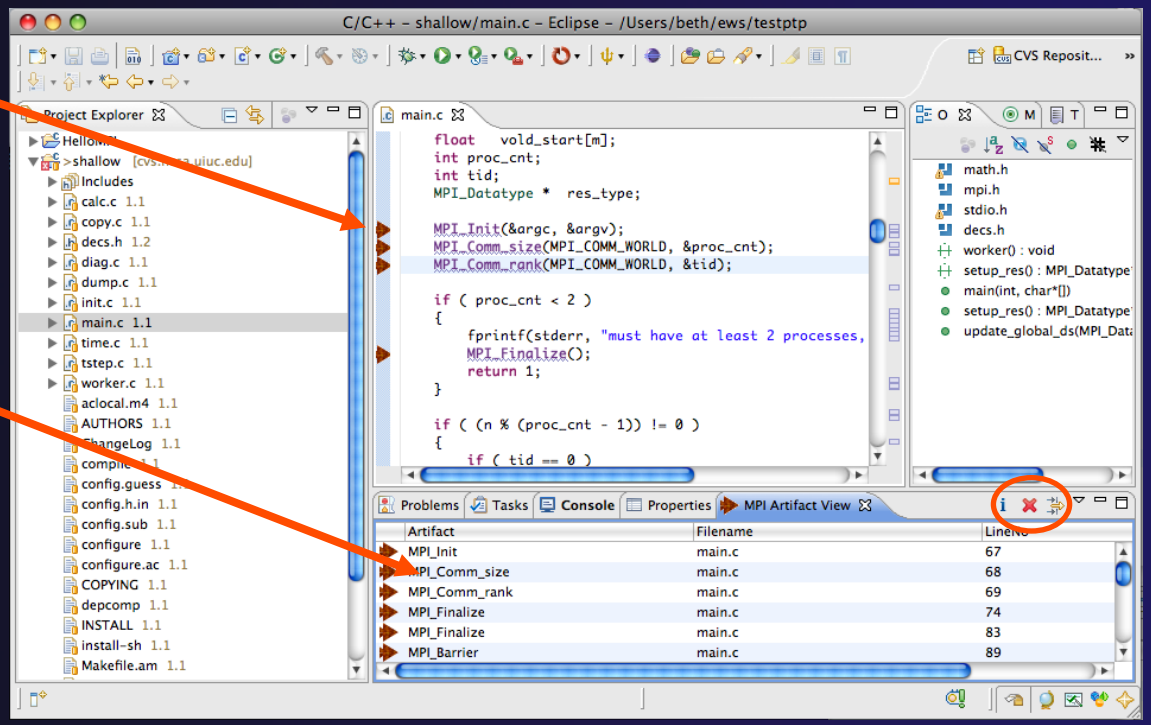
Show MPI Artifacts

- In Project Explorer, select a project, folder, or a single source file
  - The analysis will be run on the selected resources
- Select **Show MPI Artifacts**
- Run the analysis by clicking on drop-down menu next to the analysis button
- Works on local and remote files



MPI Artifact View

- Markers indicate the location of artifacts in editor
- The **MPI Artifact View** lists the type and location of each artifact
- Navigate to source code line by double-clicking on the artifact
- Run the analysis on another file (or entire project!) and its markers will be added to the view
- Remove markers via
- Click on column headings to sort



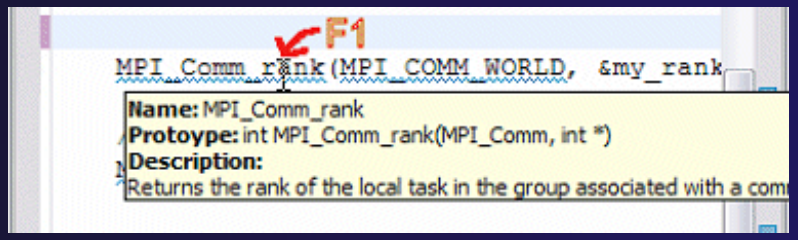
MPI Editor Features

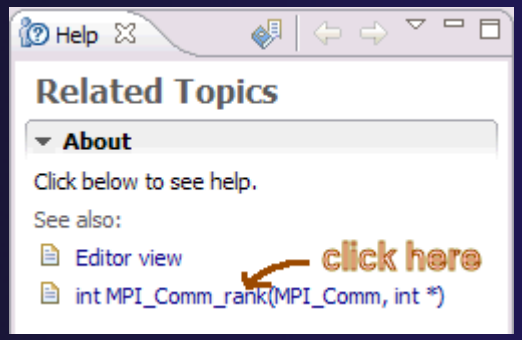
- Code completion will show all the possible MPI keyword completions
- Enter the start of a keyword then press <ctrl-space>

- Hover over MPI API
- Displays the function prototype and a description

Context Sensitive Help

- Click mouse, then press help key when the cursor is within a function name
  - Windows: **F1** key
  - Linux: **ctrl-F1** key
  - MacOS X: **Help** key or **Help▸Dynamic Help**
- A help view appears (**Related Topics**) which shows additional information (You may need to click on MPI API in editor again, to populate)
- Click on the function name to see more information
- Move the help view within your Eclipse workbench, if you like, by dragging its title tab



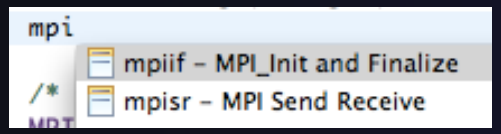


Some special info has been added for MPI APIs

MPI Templates

- Allows quick entry of common patterns in MPI programming

- Example:
  MPI send-receive
- Enter:
  `mpisr` <ctrl-space>
- Expands to a send-receive pattern
- Highlighted variable names can all be changed at once
- Type `mpi` <ctrl-space> <ctrl-space> to see all templates

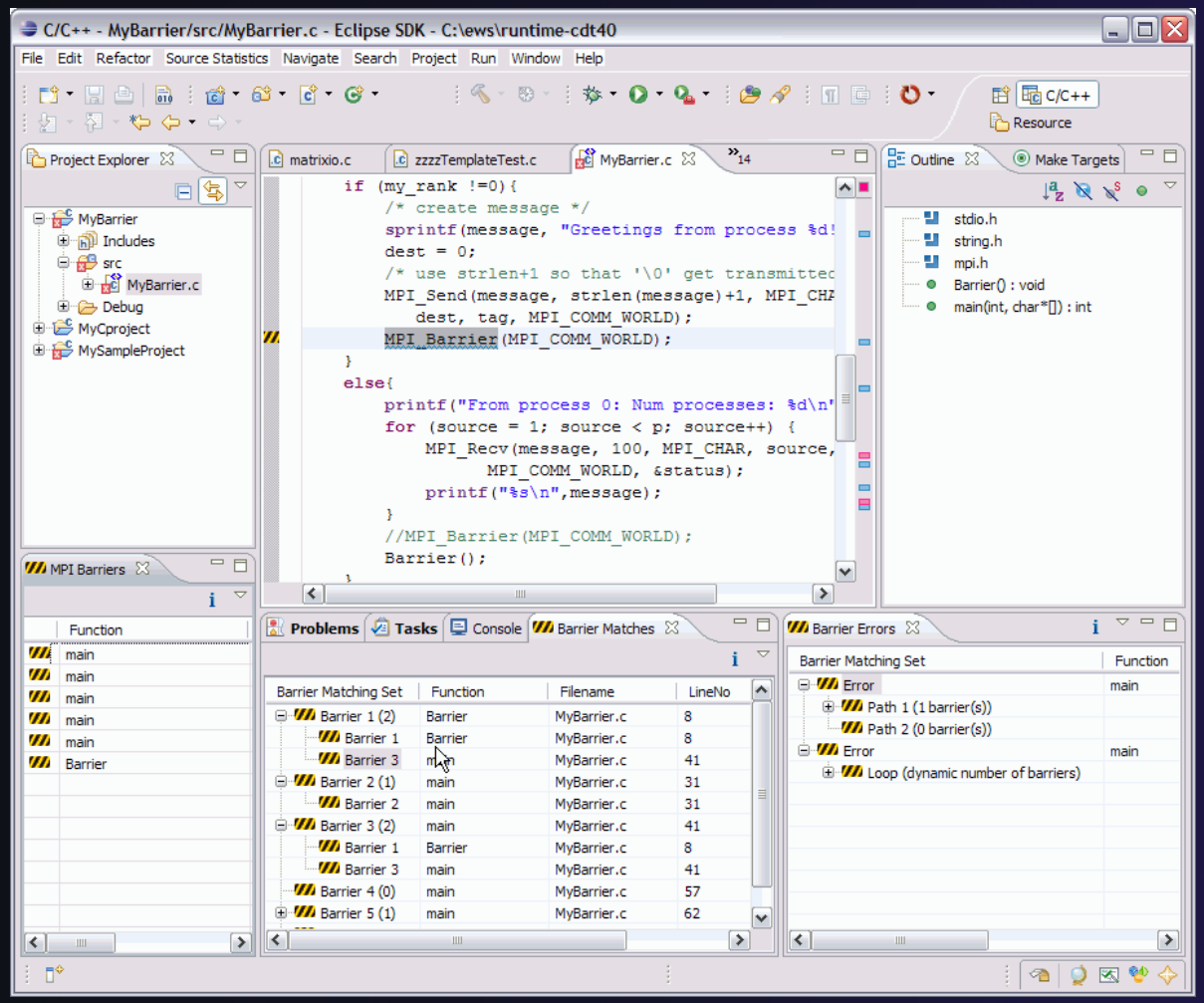



```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);
if (rank == 0){ //master task
        printf("Hello  From process 0: Num processes: %d\n",p);
        for (source = 1; source < p; source++) {
            MPI_Recv(message, 100, MPI_CHAR, source, tag,
                  MPI_COMM_WORLD, &status);
            printf("%s\n",message);
        }
    }
    else{  // worker tasks
        /* create message */
            sprintf(message, "Hello  from process %d!", my_rank);
            dest = 0;
            /* use strlen+1 so that '\0' get transmitted */
            MPI_Send(message, strlen(message)+1, MPI_CHAR,
               dest, tag, MPI_COMM_WORLD);
    }
```

Add more templates using Eclipse preferences!
C/C++>Editor>Templates
Extend to other common patterns

MPI Barrier Analysis

Local files only



Verify barrier synchronization in C/MPI programs

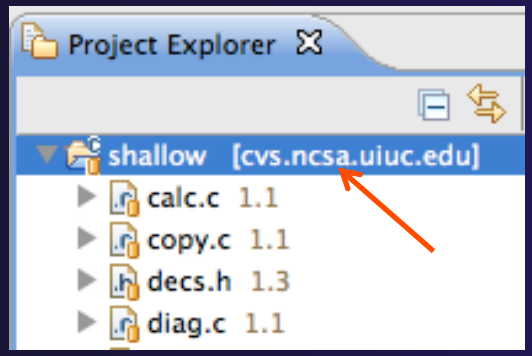
Interprocedural static analysis outputs:

- ✦For verified programs, lists barrier statements that synchronize together (match)
- ✦ For synchronization errors, reports counter example that illustrates and explains the error

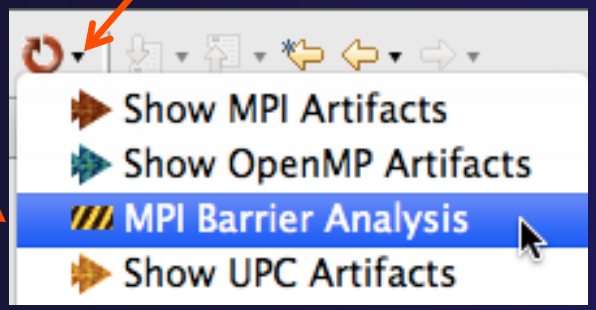
MPI Barrier Analysis – Try it

Run the Analysis:

- In the Project Explorer, select the project (or directory, or file) to analyze



- Select the MPI Barrier Analysis action in the pull-down menu



MPI Barrier Analysis – Try It (2)

- No Barrier Errors are found (no pop-up indicating error); Two barriers are found

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help

C/C++ - shallow/main.c - Eclipse - /Users/beth/ews/

Project Explorer

shallow [cvs.ncsa.uiuc.edu]
- calc.c 1.1
- copy.c 1.1
- decs.h 1.3
- diag.c 1.1
- dump.c 1.1
- Includes
- init.c 1.2
- main.c 1.4
- time.c 1.2
- tstep.c 1.1
- worker.c 1.2
- aclocal.m4 1.1
- AUTHORS 1.1
- ChangeLog 1.1
- compile 1.1
- config.guess 1.1

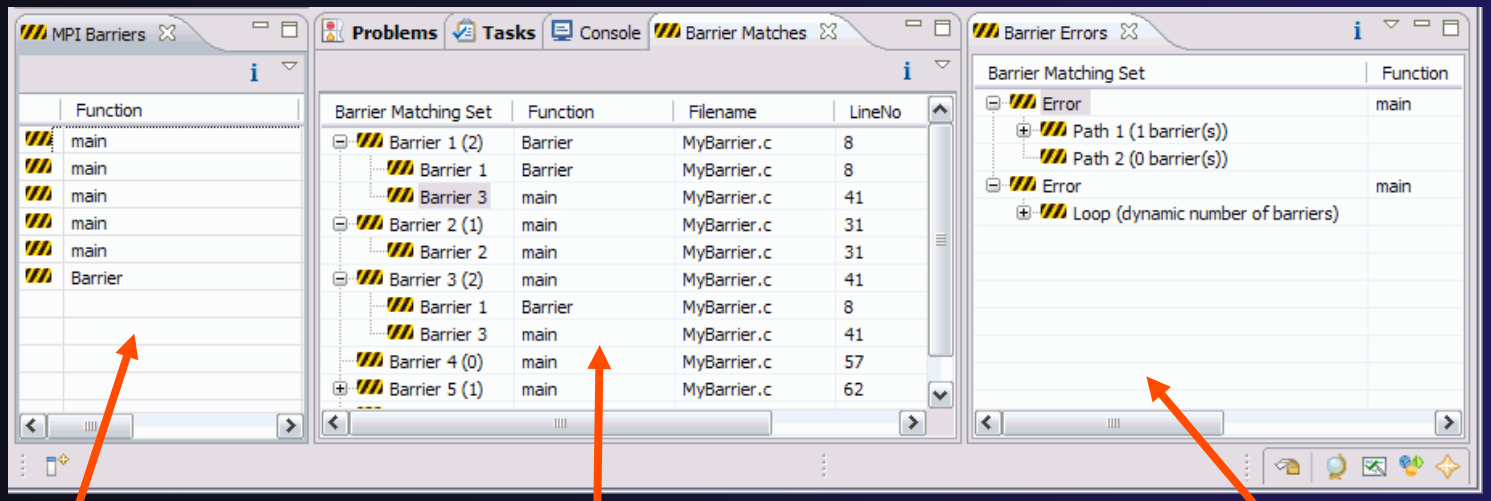
main.c

```
83          MPI_Finalize();
84          return 1;
85      }
86
87      if (tid != 0) {
88          worker();
89          MPI_Barrier(MPI_COMM_WORLD);
90          MPI_Finalize();
91      } else {
92
93      /* master process */
94
```

MPI Barrier Matches | MPI Barriers

Function	Filename	LineNo	IndexNo
main	main.c	89	1
main	main.c	206	2

MPI Barrier Analysis - views



MPI Barriers view

Simply lists the barriers

Like MPI Artifacts view, double-click to navigate to source code line (all 3 views)

Barrier Matches view

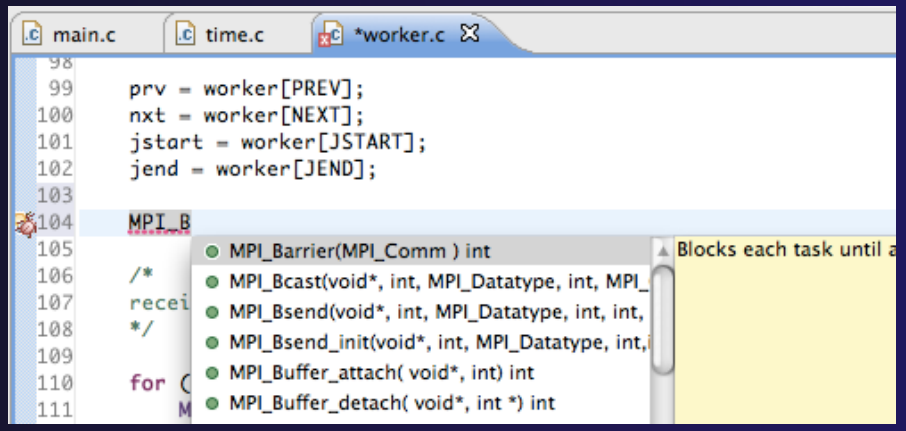
Groups barriers that match together in a barrier set – all processes must go through a barrier in the set to prevent a deadlock

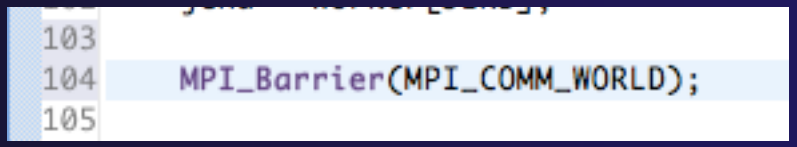
Barrier Errors view

If there are errors, a counter-example shows paths with mismatched number of barriers

Barrier Errors

- Let's cause a barrier mismatch error
- Open worker.c in the editor by double-clicking on it in Project Explorer
- At about line 104, enter a barrier:
  - Type MPI_B
  - Hit Ctl-space
  - Select MPI_Barrier
  - Add communicator arg MPI_COMM_WORLD and closing semicolon

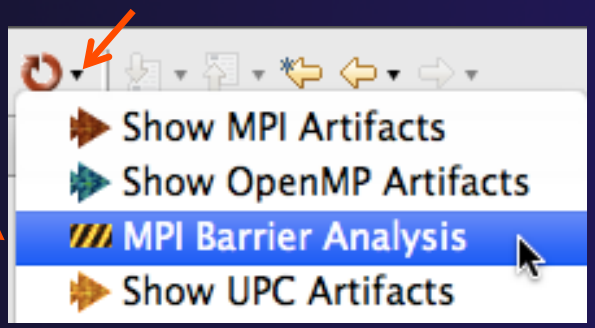




Barrier Errors (2)

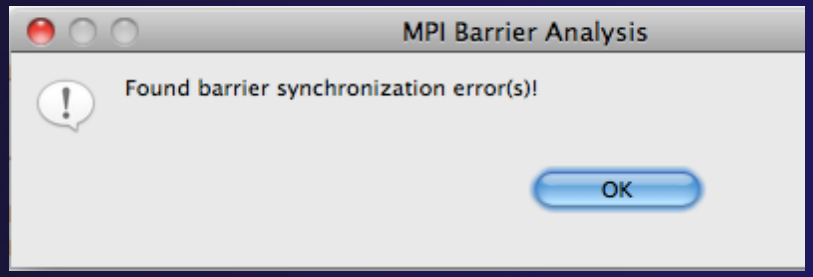
- Save the file
  - Ctl-S (Mac Command-S) or File > Save
  - Tab should lose asterisk indicating file saved

- Run barrier analysis on shallow project again
  - Select shallow project in Project Explorer first



Barrier Errors (3)

- Barrier Error is found
- Hit OK to dismiss dialog



- Code diverges on line 87
  - One path has 2 barriers, other has 1

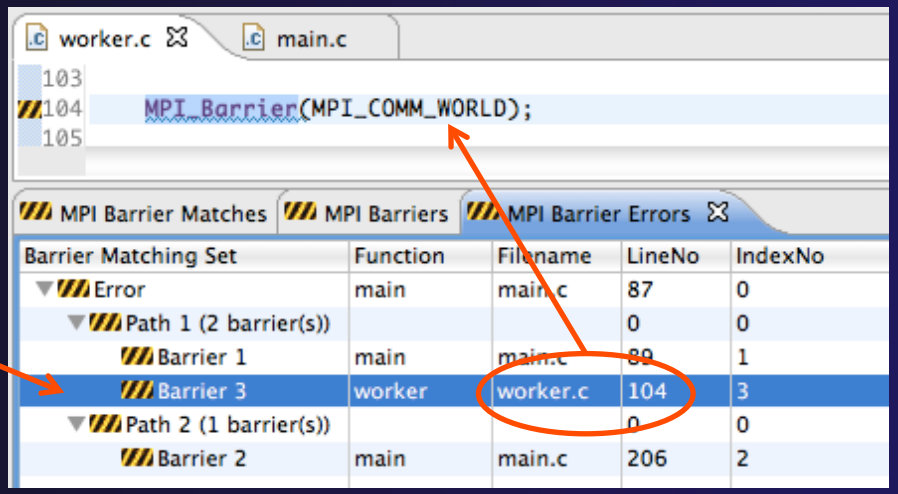
MPI Barrier Matches | MPI Barriers | MPI Barrier Errors

Barrier Matching Set	Function	Filename	LineNo	IndexNo
Error	main	main.c	87	0
Path 1 (2 barrier(s))			0	0
Barrier 1	main	main.c	89	1
Barrier 3	worker	worker.c	104	3
Path 2 (1 barrier(s))			0	0
Barrier 2	main	main.c	206	2

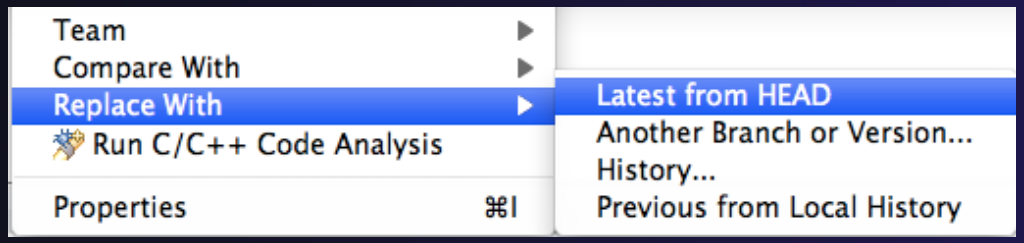
Double-click on a row in Barrier Errors view to find the line it references in the code

Fix Barrier Error

- Fix the Barrier Error before continuing
- Double-click on the barrier in worker.c to quickly navigate to it

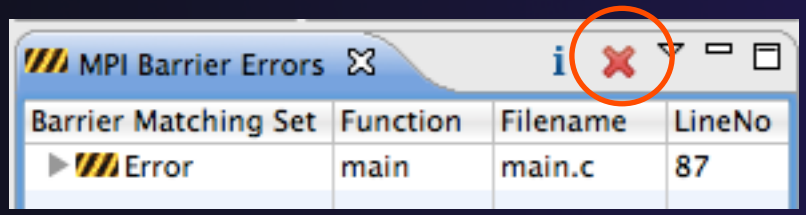


- Remove the line and save the file
  -or-
  Right mouse on worker.c in Project Explorer and do **Replace With > Latest from HEAD**



Remove Barrier Markers

- Run Barrier Analysis again to remove the error
  - and/or -
- Remove the Barrier Markers via the “X” in one of the MPI Barrier views



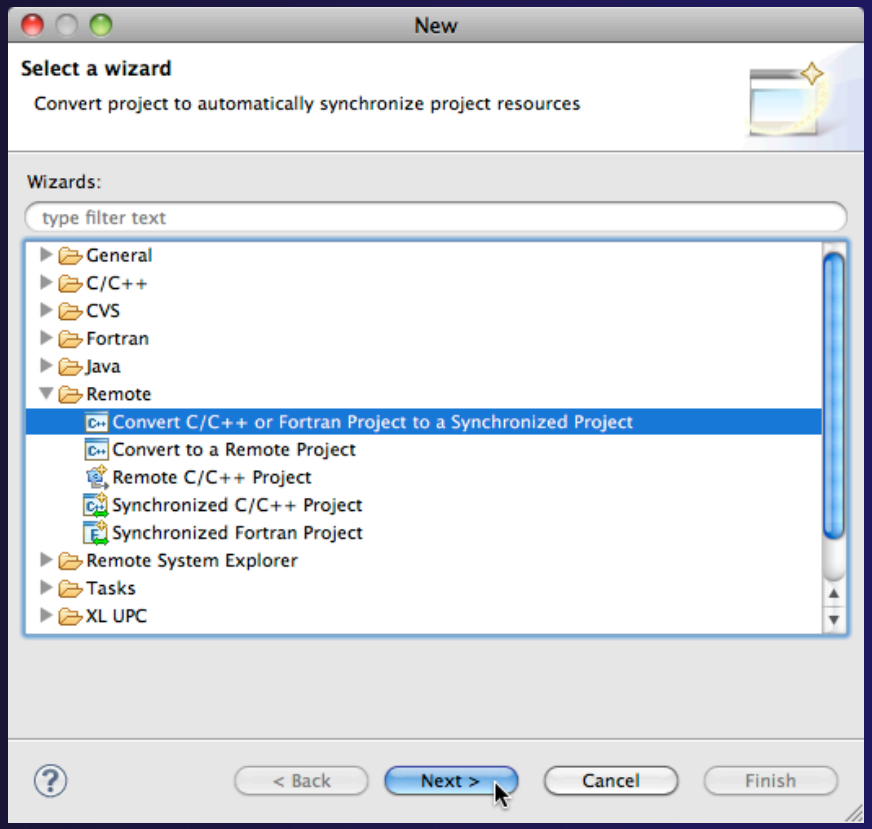
Synchronizing the Project

Synchronizing the Project

- Because we will be running on a remote system, we must also build on that system
- Source files must be available to build
- We will use a synchronized project to do this
  - Only needs to be done once for each project
  - A synchronized project could have been created initially
- Files are synchronized automatically when they are saved
- A full synchronize is also performed prior to a build

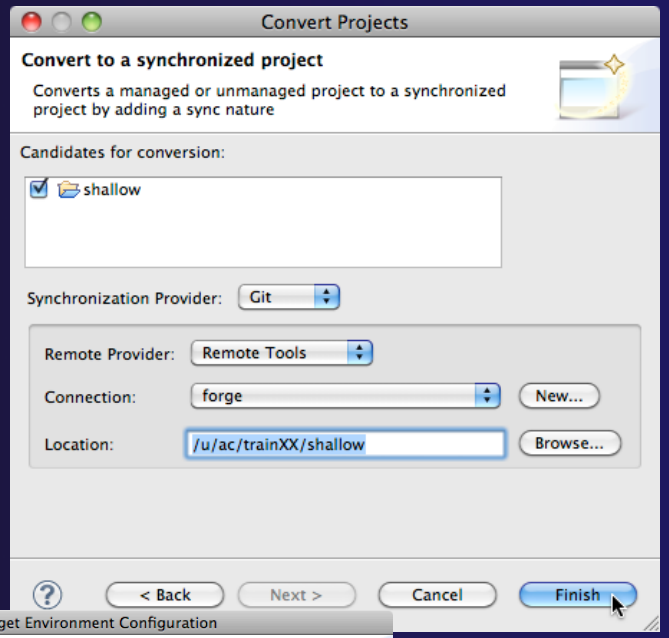
Converting To Synchronized

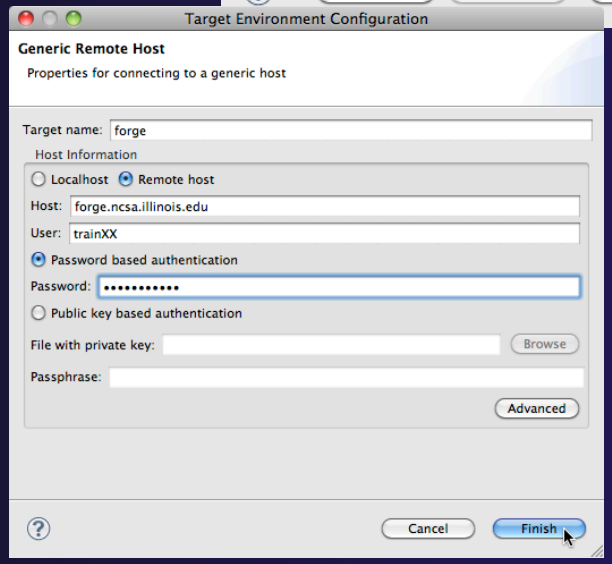
- Select **File>New>Other…**
- Open the Remote folder
- Select **Convert C/C++ or Fortran Project to a Synchronized Project**
- Click **Next>**



Convert Projects Wizard

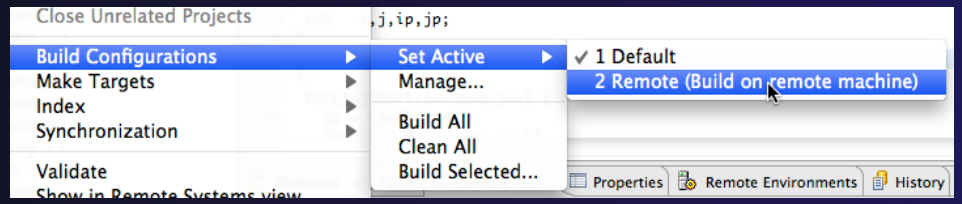
- Select checkbox next to "shallow"
- For **Connection:**, click on **New…**
- Enter "forge" in the **Target name** field
- Enter "forge.ncsa.illinois.edu" in the **Host** field
- Enter your trainXX user ID in the **User** field
- Enter your password in the **Password** field
- Click **Finish**
- Enter "/u/ac/trainXX/shallow" in the **Location** field
- Click **Finish**



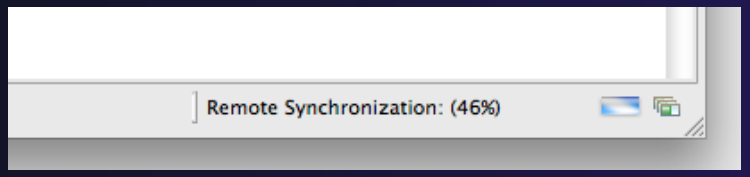


Set Active Build Configuration

- The "Active" build configuration determines which system will be used for both synchronizing and building
- Right-click on the project and select **Build Configurations>Set Active>Remote (Build on remote machine)**



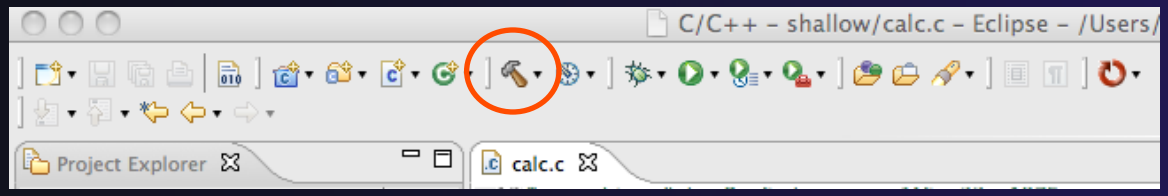
- The project should synchronize immediately



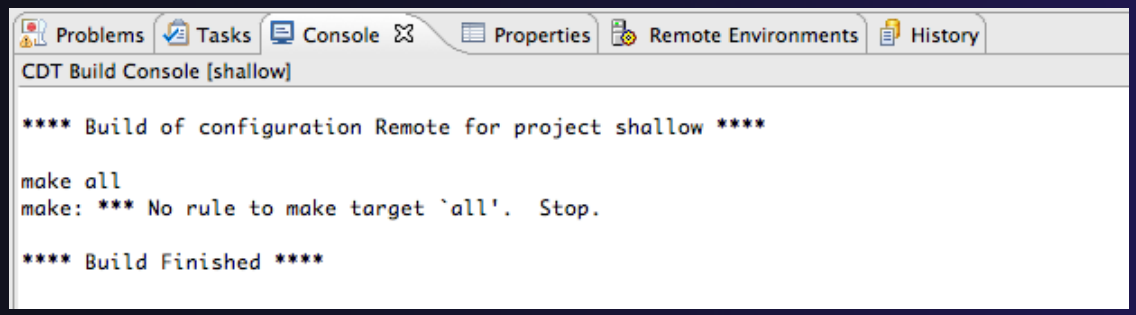
Building the Project

Building the Project

- Click on the button to run the build

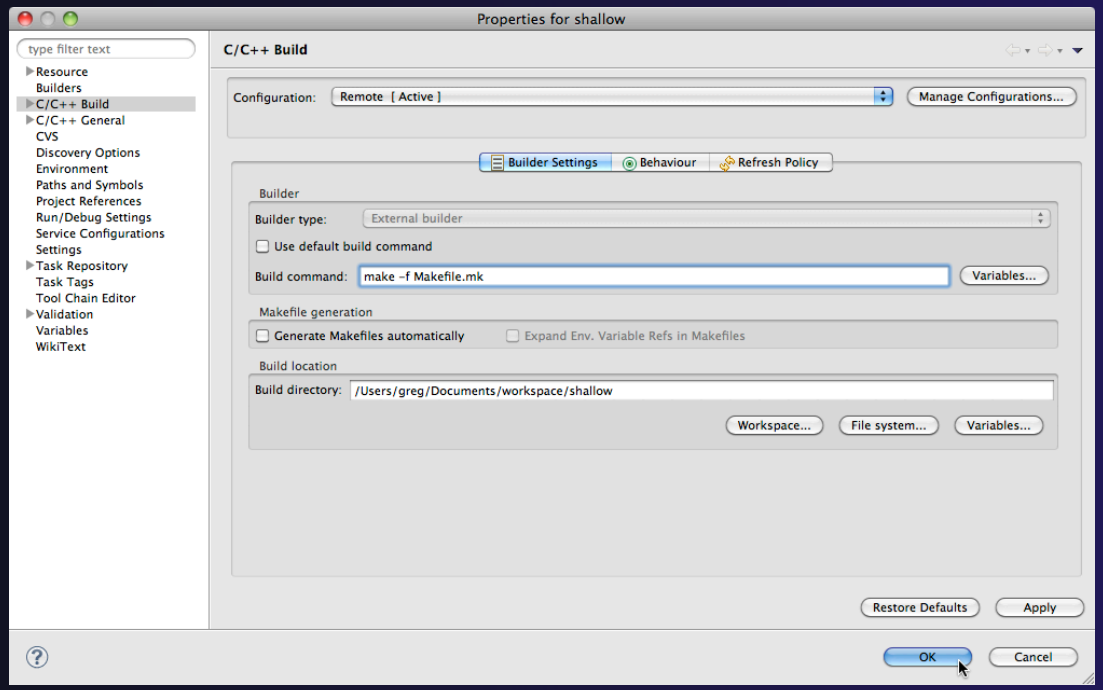


- By default, the Build Configuration assumes there is a Makefile (or makefile) for the project
- In this case, there is no Makefile, so the build will fail



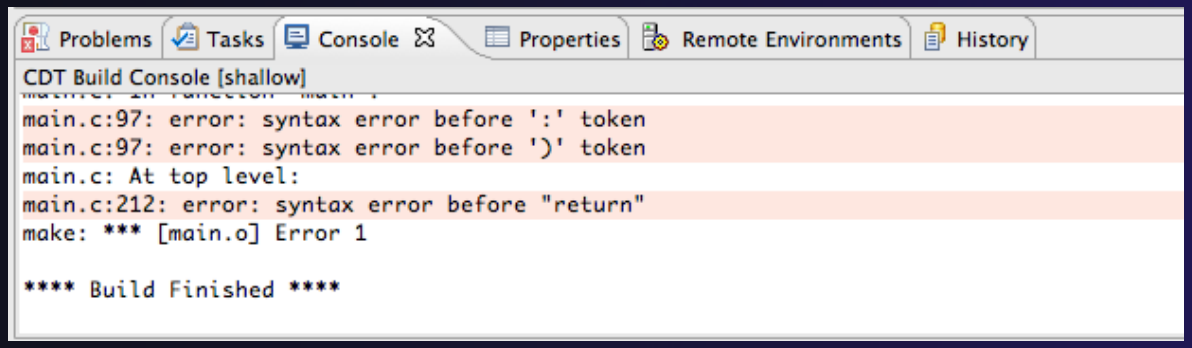
Fixing The Project Properties

- The build command is specified in the project properties
- Open the properties by right-clicking on "shallow" and selecting **Properties**
- Click on **C/C++ Build**
- Uncheck **Use default build command**
- Enter "make –f Makefile.mk" in the **Build Command** field
- Click **OK**



Re-Building the Project

- Click on the button again to run the build
- Build output will be shown in the **Console** view



Build Problems

- Build problems will be shown in a variety of ways
  - Marker on file
  - Marker on editor line
  - Line is highlighted
  - Marker on overview ruler
  - Listed in the **Problems view**
- Double-click on line in **Problems view** to go to location of error

C/C++ - shallow/main.c - Eclipse - /Users/greg/Documents/workspace

```
87  if (tid != 0) {
88      worker();
89      MPI_Barrier(MPI_COMM_WORLD);
90      MPI_Finalize();
91  } else {
92
93  /* master process */
94
95  chunk_size = n / (proc_cnt - 1);
96
97  for (i = 1; i <= proc_cnt; i++) {
98      /* calculate each worker's boundary */
99      master_packet[JSTART] = (i - 1) * chunk_size;
100
101     if (i == proc_cnt - 1)
102         master_packet[JEND] = n - 1;
103     else
104         master_packet[JEND] = i * chunk_size - 1;
105
106     if (i == 1)
107         prv = proc_cnt-1;
108     else
109         prv = i-1;
110
111     master_packet[PREV] = prv;
112
113     if (i == proc_cnt - 1)
114         nxt = 1;
115     else
116         nxt = i+1;
117
118     master_packet[NEXT] = nxt;
```

Problems: 3 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (3 items)				
syntax error before ':' token	main.c	/shallow	line 97	C/C++ Problem
syntax error before ')' token	main.c	/shallow	line 97	C/C++ Problem
syntax error before "return"	main.c	/shallow	line 212	C/C++ Problem

Fix Build Problems

- Fix errors by changing ':' to ';' on line 97
- Save the file
- Rebuild by pressing build button
- Error markers have been removed
- Check console for correct build output

```
92
93      /* master process */
94
95      chunk_size = n / (proc_cnt - 1);
96
97      for (i = 1; i < proc_cnt; i++) {
98          /* calculate each worker's boundary */
99          master_packet[JSTART] = (i - 1) * chunk_size;
100
101         if (i == proc_cnt - 1)
102             master_packet[JEND] = n - 1;
103         else
104             master_packet[JEND] = i * chunk_size - 1;
105
106         if (i == 1)
```

Problems | Tasks | Console | Properties | Remote Environments | History

CDT Build Console [shallow]

```
mpicc -g   -c -o time.o time.c
mpicc -g   -c -o tstep.o tstep.c
mpicc -g   -c -o worker.o worker.c
mpicc -g   -c -o dump.o dump.c
mpicc -g -o shallow calc.o copy.o diag.o init.o main.o time.o tstep.o worker.o dump.o -lm

**** Build Finished ****
```

Resource Manager Configuration

Resource Managers

- PTP uses the term “resource manager” to refer to any subsystem that controls the resources required for launching a parallel job.
- Examples:
  - Batch scheduler (e.g. LoadLeveler, PBS, SLURM)
  - Interactive execution (e.g. Open MPI, MPICH2, etc.)
- Each resource manager controls one target system
- Resource Managers can be local or remote

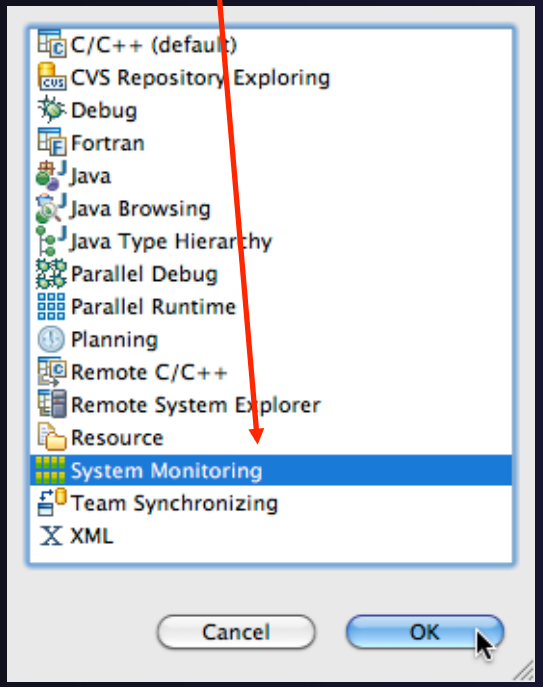
Monitoring/Runtime Perspectives

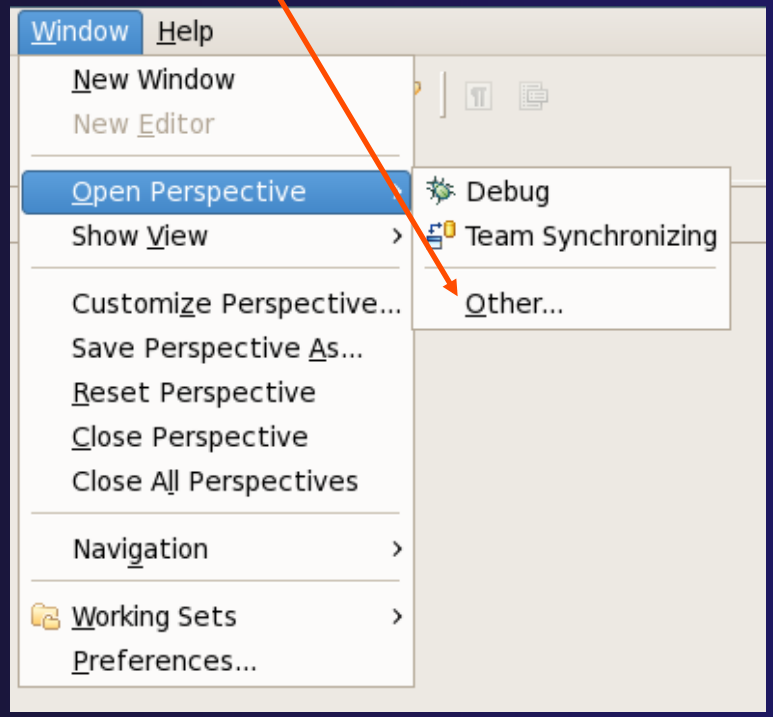
- Parallel Runtime Perspective
  - Used for legacy PTP Resource Managers
- System Monitoring Perspective
  - Used for newer Configurable Resource Managers (since PTP 5.0)
- Which one is used?

Resource Manager	System Monitoring	Parallel Runtime
IBM LoadLeveler		✔
IBM Parallel Env		✔
MPICH2		✔
Open MPI		✔
PBS-Batch-Generic	✔	
PBS-Batch-Interactive	✔	
Remote Launch		✔
SLURM		✔

Using a Job Scheduler

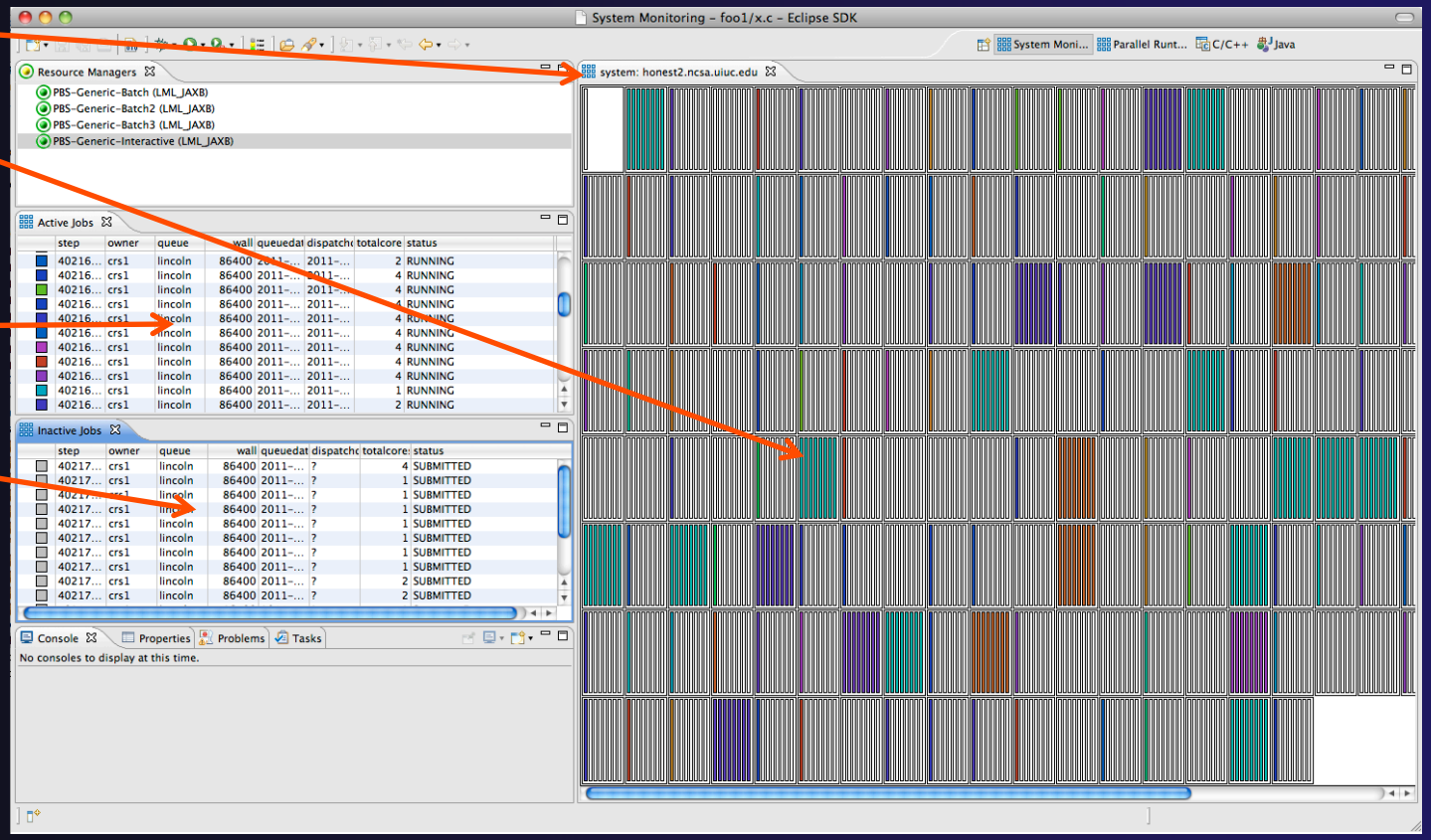
- Setting up a resource manager is done in the System Monitoring perspective
  - (For PTP 5.0, this applies to PBS)
- Select **Window>Open Perspective>Other**
- Choose **System Monitoring** and click **OK**





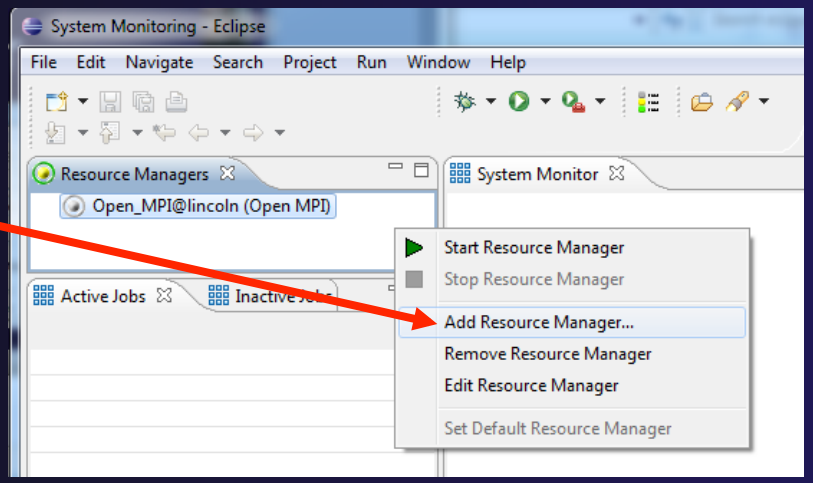
System Monitoring Perspective

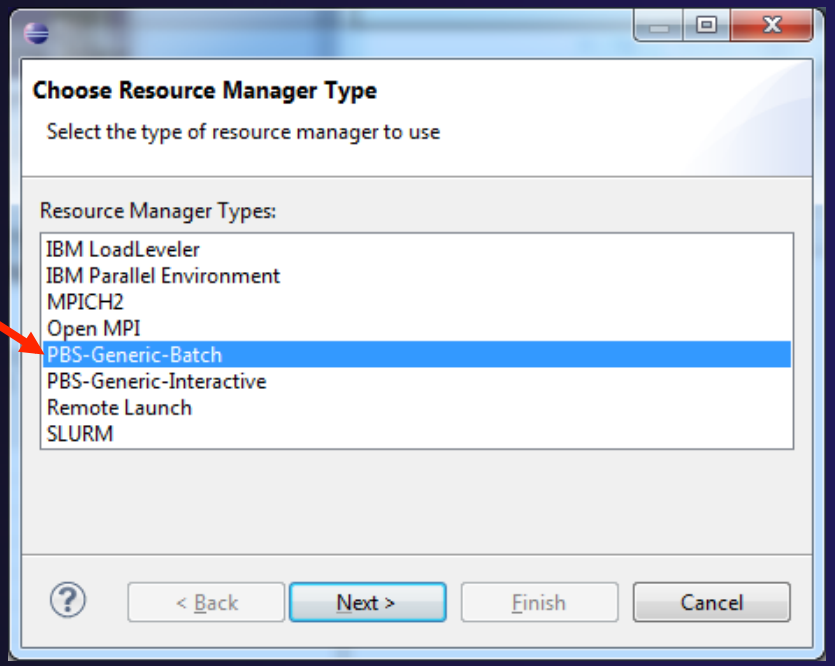
- System view
- Jobs running on system
- Active jobs
- Inactive jobs



Configuring Job Scheduler

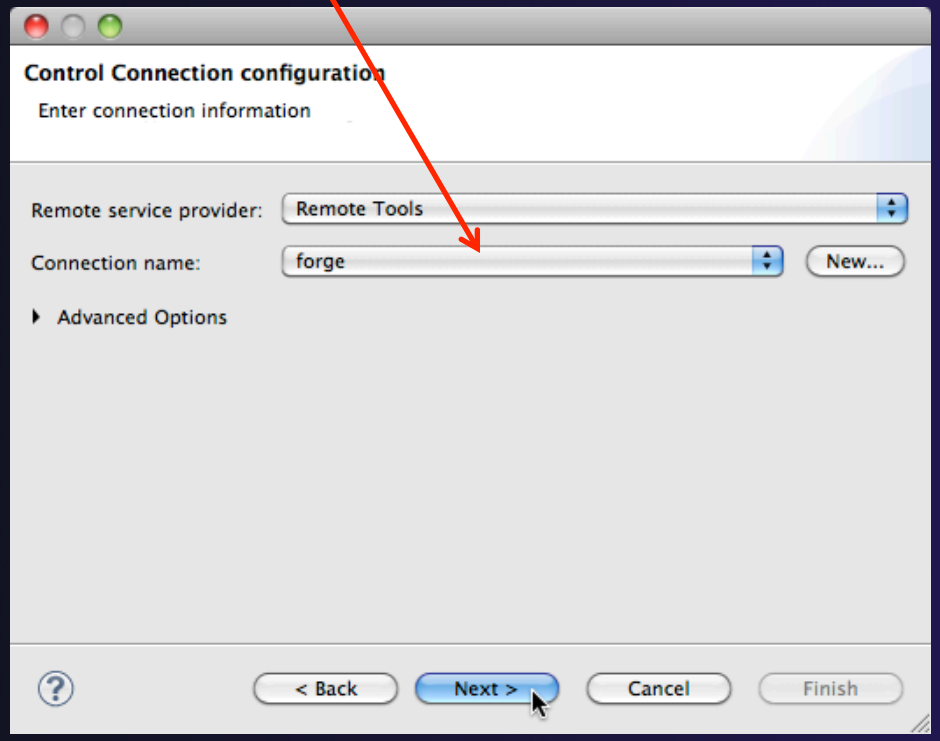
- Right-click in Resource Managers view and select **Add Resource Manager**
- Choose the **PBS-Generic-Batch** Resource Manager Type
- Select **Next>**





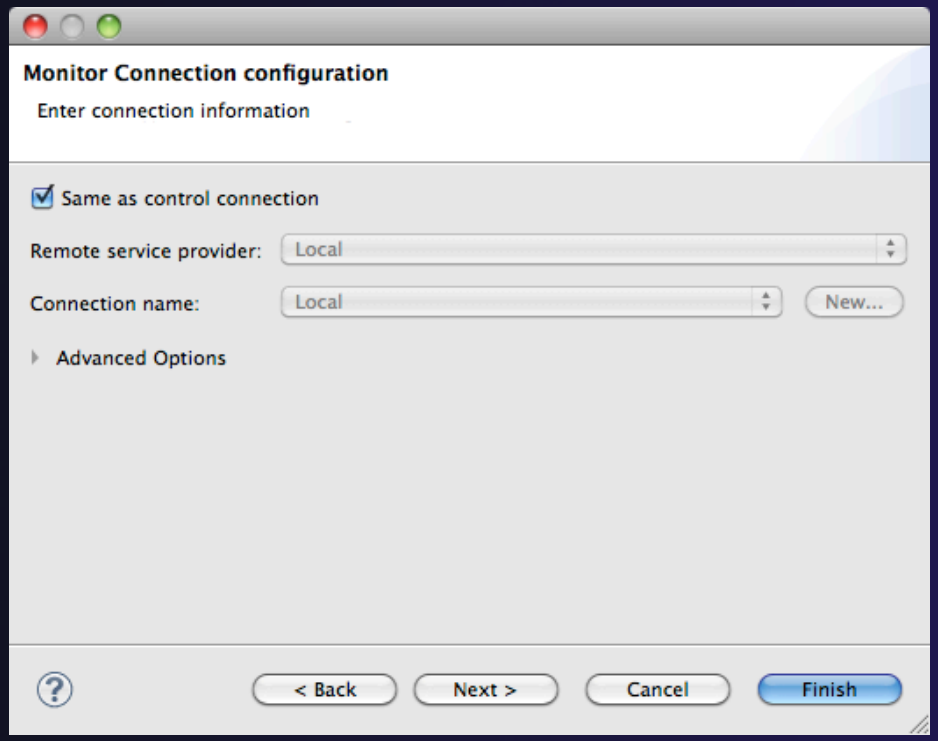
Configure Control Connection

- Choose **Remote Tools** for **Remote service provider**
- Choose the remote connection you made previously
- Click **Next>**



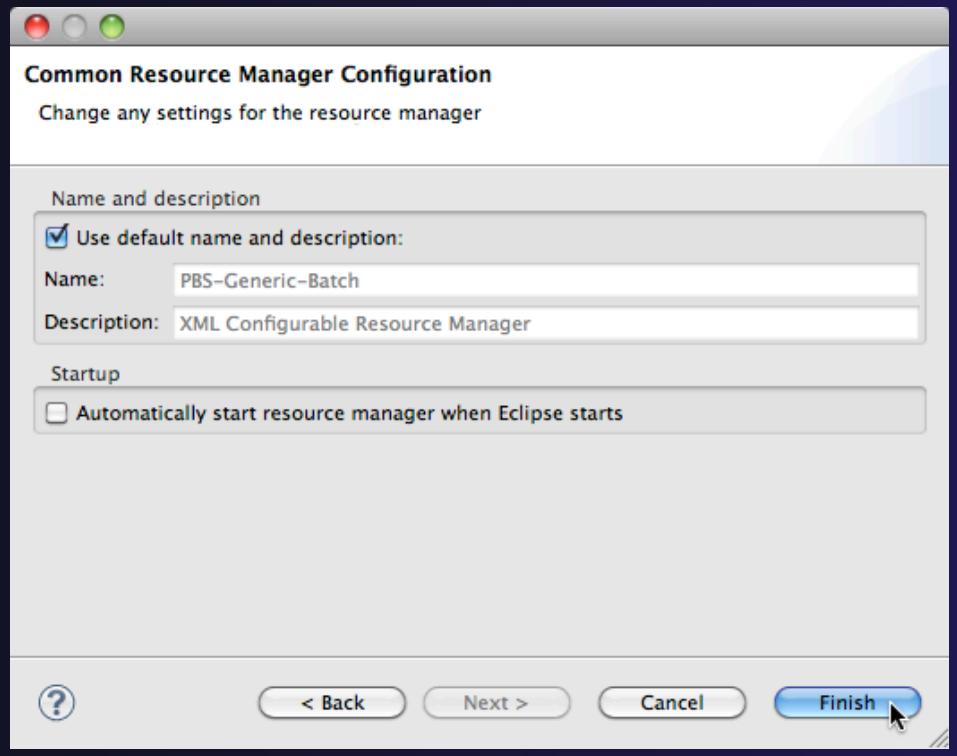
Configure Monitor Connection

- Keep default Monitor Connection (same as Control Connection), click **Next**



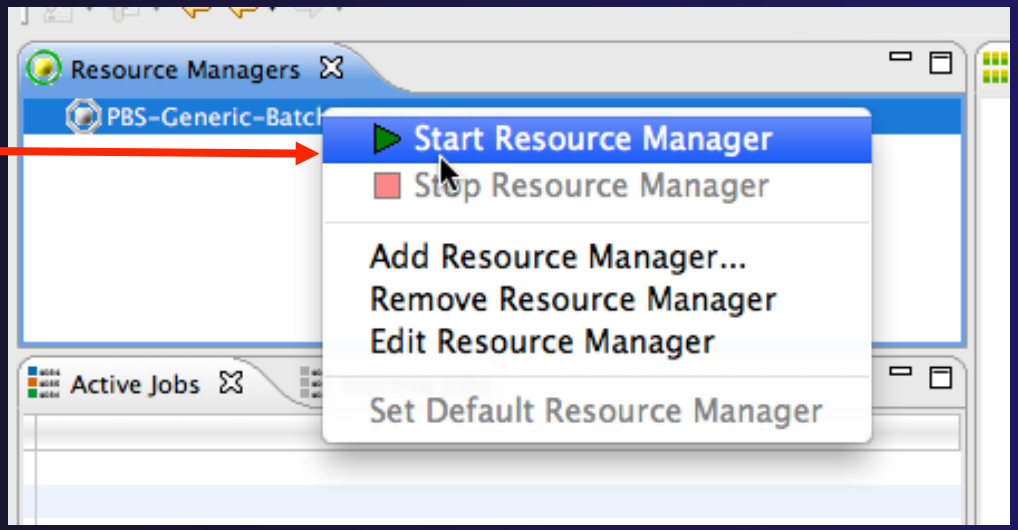
Common Configuration

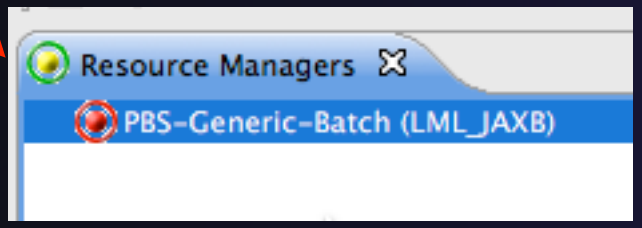
- Keep default name
- Can automatically start Resource Manager (leave unselected today)
- Click **Finish**

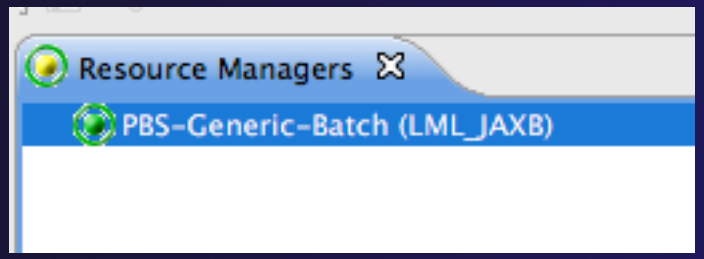


Starting the Resource Manager

- Right click on new resource manager and select **Start resource manager**
- If everything is ok, you should see the resource manager change to green
- If something goes wrong, it will change to red

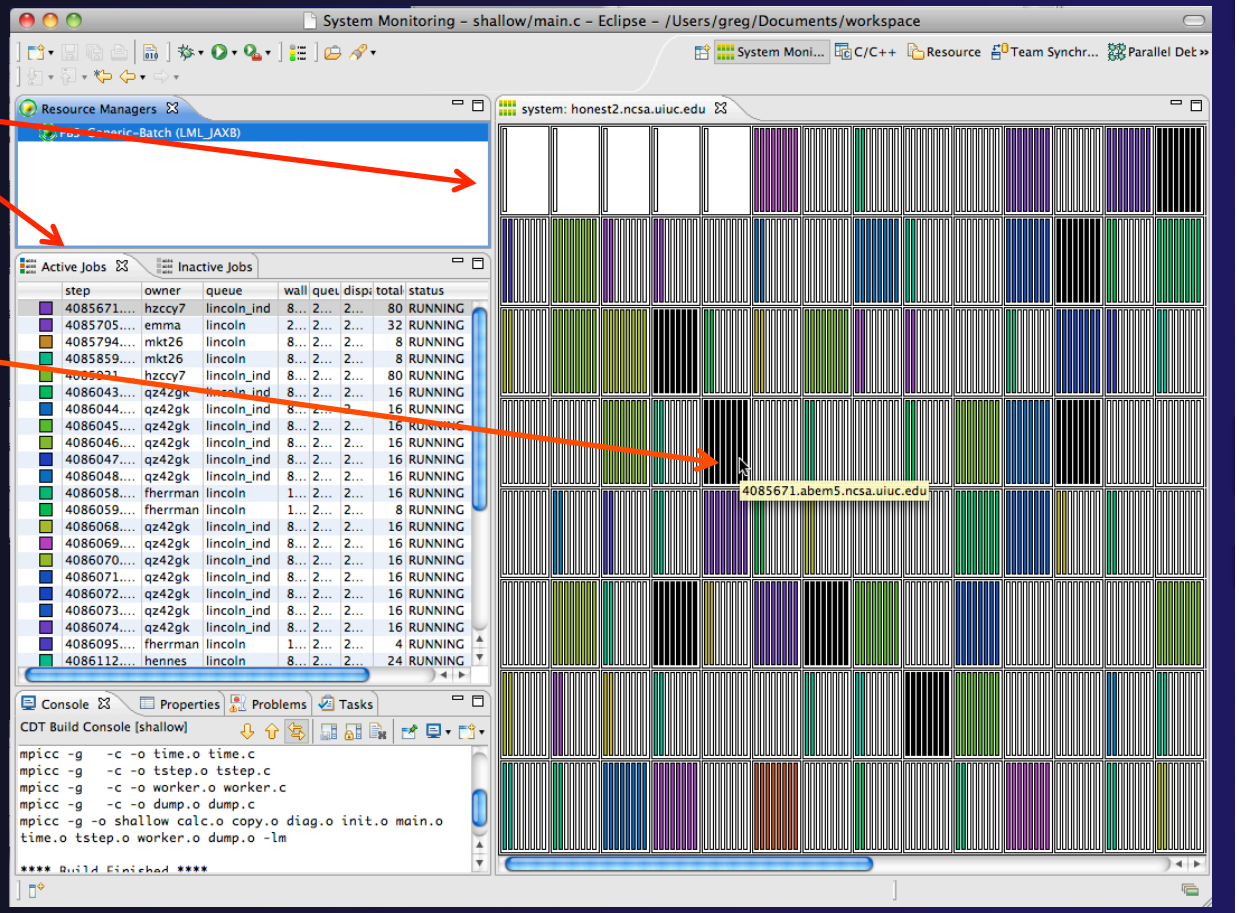






System Monitoring

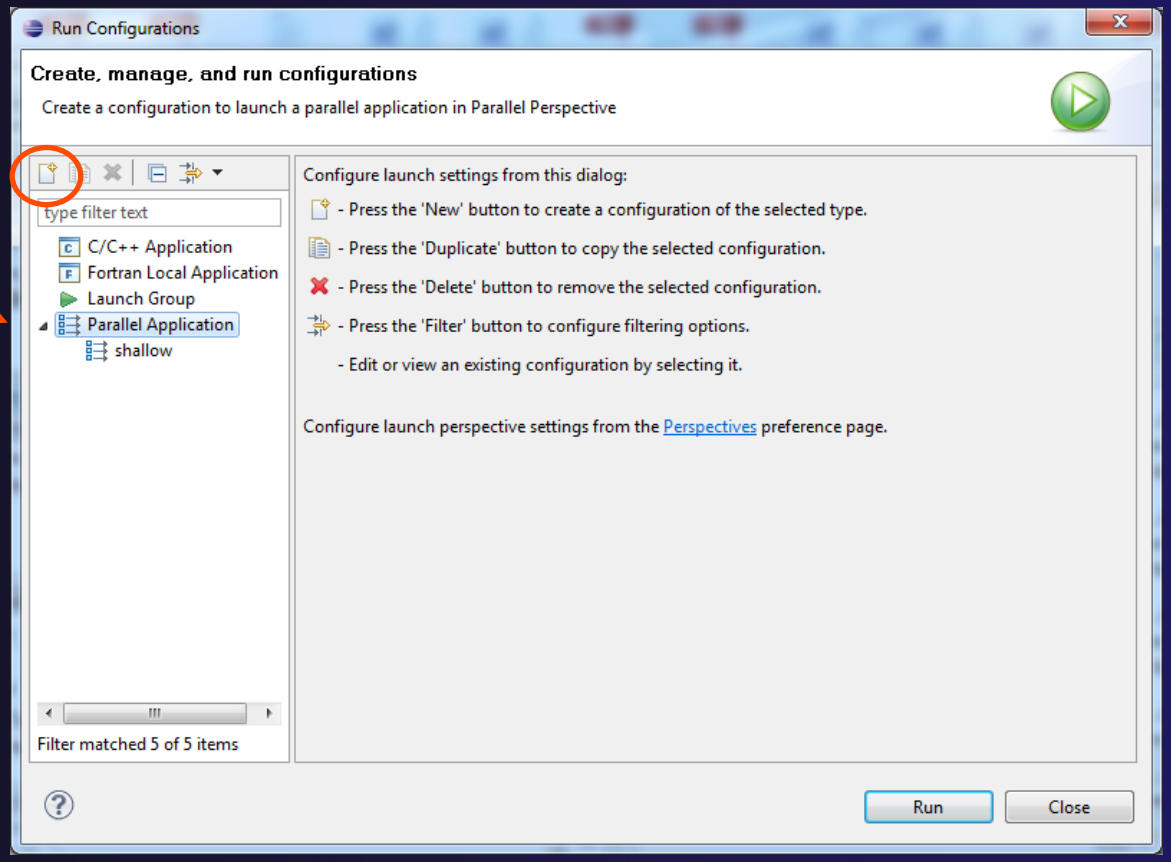
- System view, with abstraction of nodes
- Active and inactive jobs
- Hover over node to see job running on node



Launching a Job

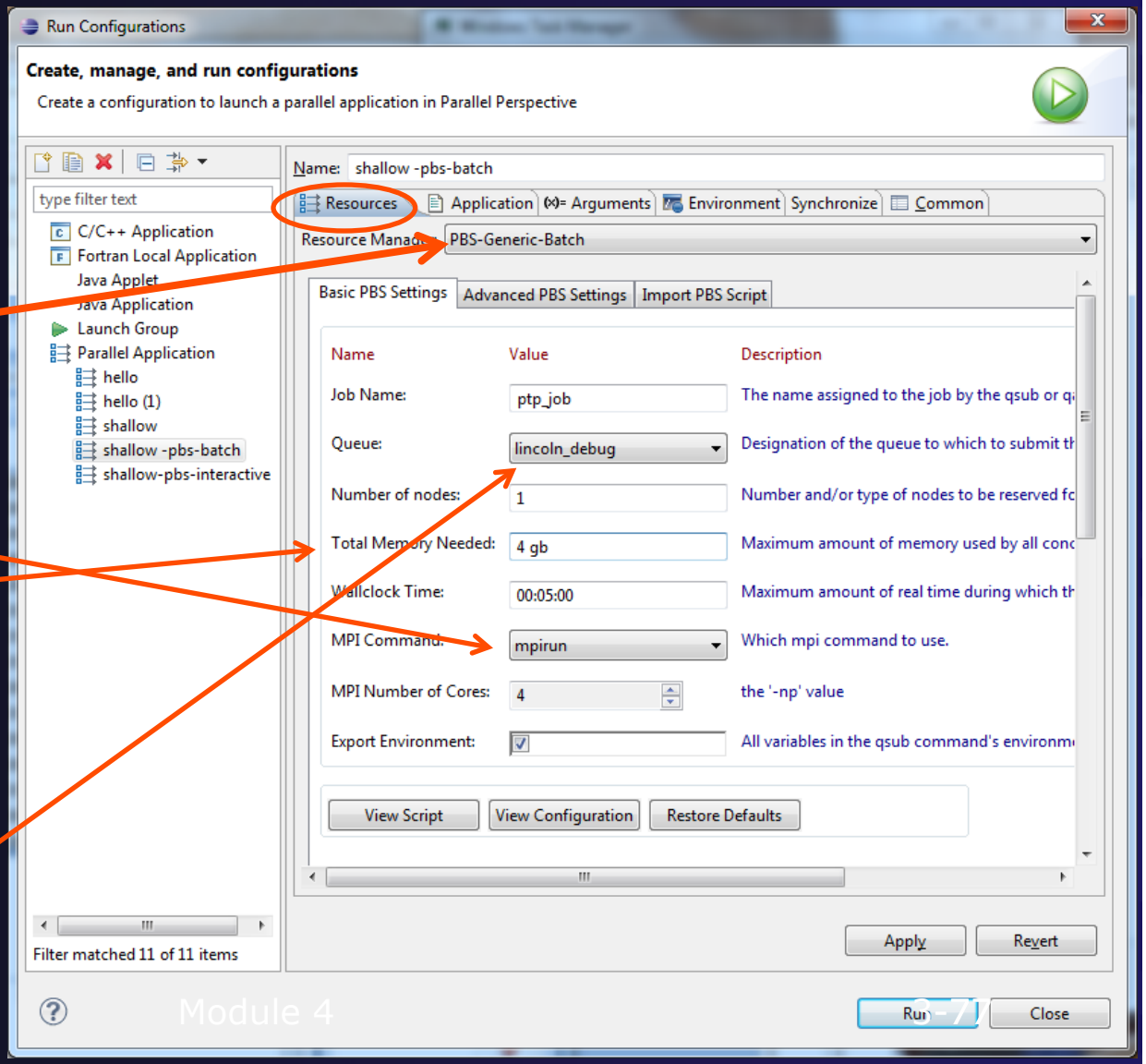
Create a Launch Configuration

- Open the run configuration dialog **Run>Run Configurations…**
- Select **Parallel Application**
- Select the **New** button



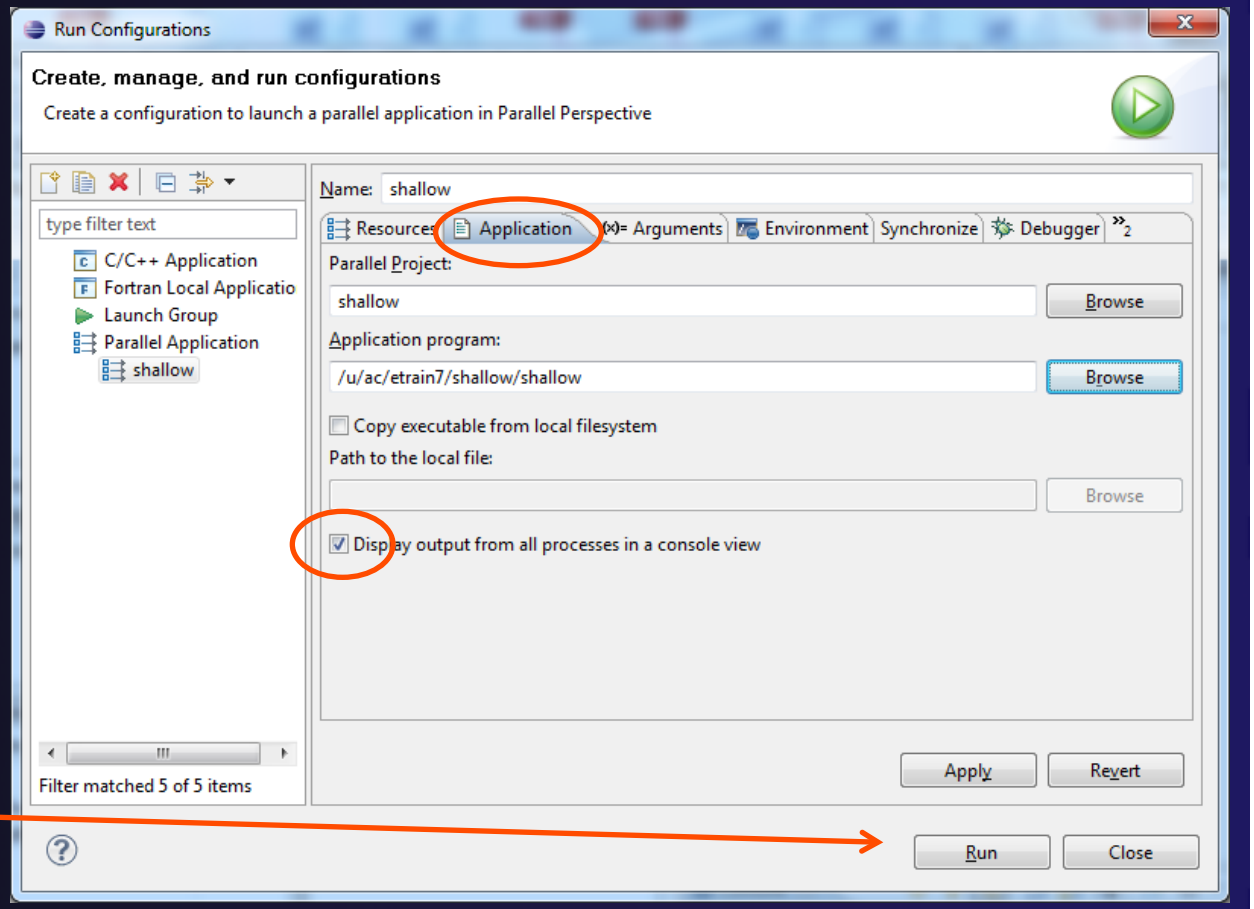
Complete the Resources Tab

- Enter a name for this launch configuration, e.g. "shallow-pbs-batch
- Choose the appropriate Resource Manager (PBS-Generic-Batch)
- In **Resources** tab, select the PBS resource manager you just created
- The **MPI Command** field allows this job to be run as an MPI job
  - Choose **mpirun**
- Enter the resources needed to run this job
  - Use 1 nodes, 4 gb memory, 4 cores
- Select the destination queue – **lincoln_debug**



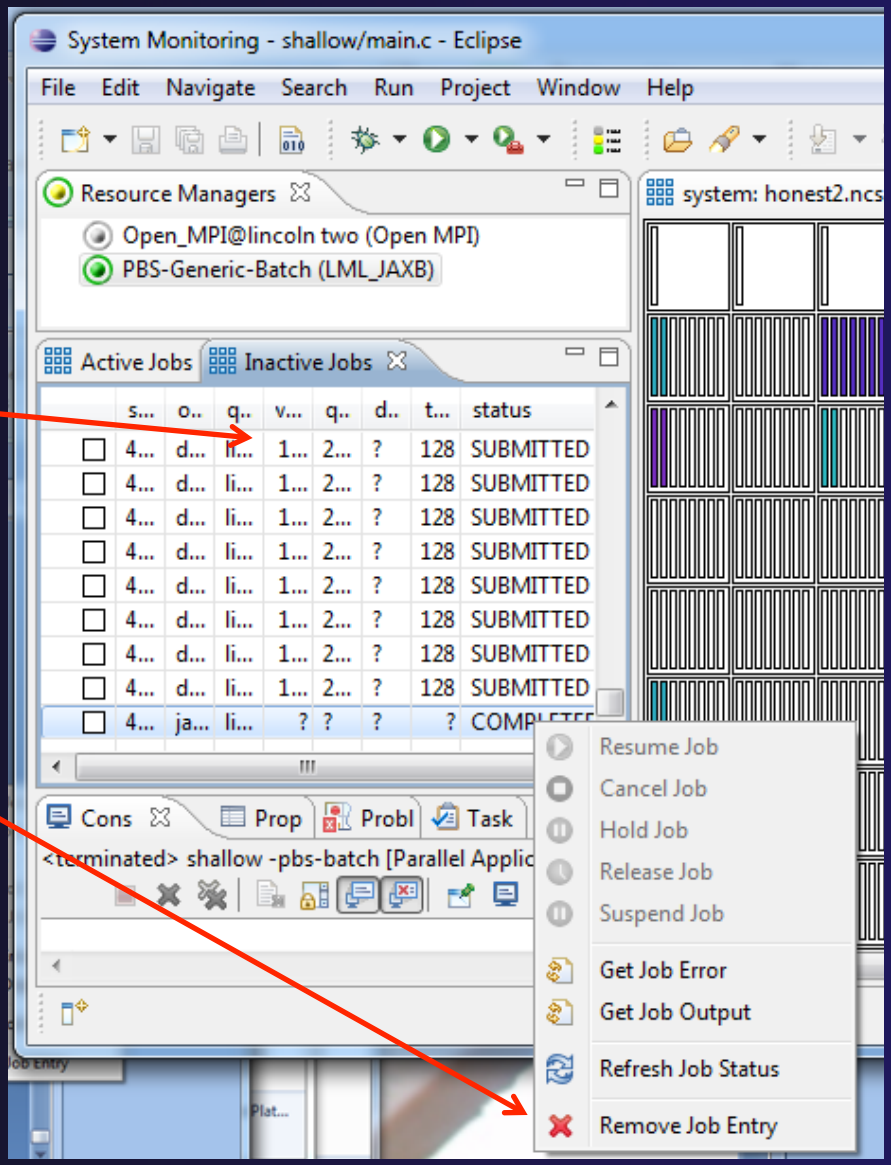
Complete the Application Tab

- Select the **Application** tab
- Choose the **Application program** by clicking the **Browse** button and locating the executable on the remote machine
  - Use the same "shallow" executable
- Select **Display output from all processes in a console view**
- If Debugger tab has error, select Debugger: **SDM**
- Click **Run** to submit the application to the job scheduler



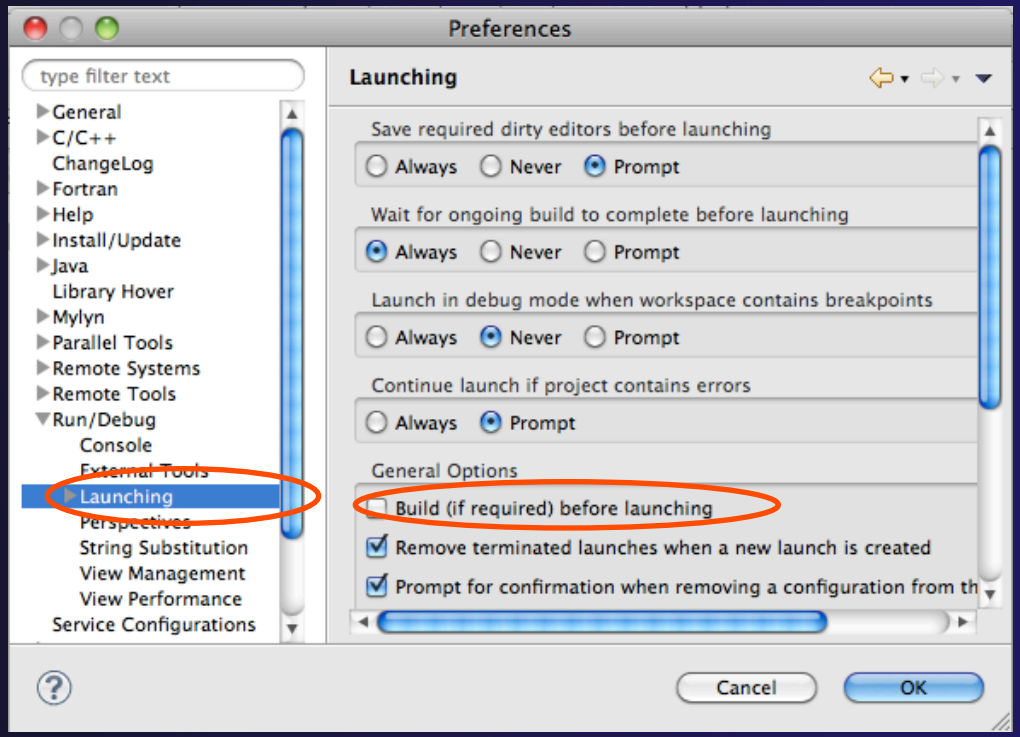
Job Monitoring

- Job initially appears in "Inactive Jobs", then in "Active Jobs", then returns to Inactive on completion
- Can view output or error by right clicking on job, selecting appropriate output



Building before Run

- If projects build prior to launch, you can turn it off.
  - Go into **Preferences>Run/Debug** and click on **Launching**.
  - Uncheck "**Build (if required) before launching**"

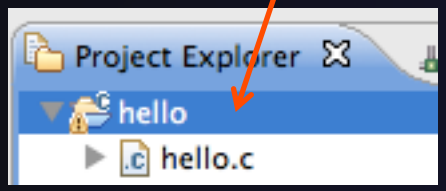


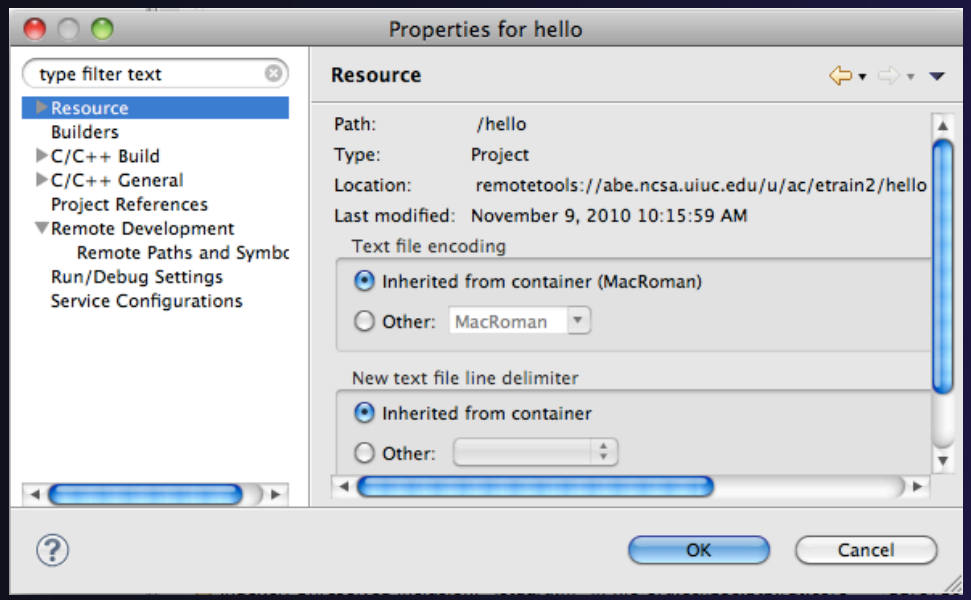
Advanced Features

Fortran
Project Properties

Project Properties

- Right-click Project
- Select **Properties**…



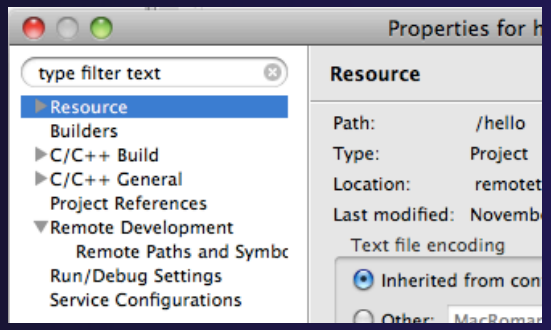


- *Project properties* are settings that can be changed for each project
  - Contrast with *workspace preferences*, which are the same regardless of what project is being edited
    - e.g., editor colors
    - Set in **Window ▸ Preferences** (on Mac, **Eclipse ▸ Preferences**)
    - Careful! Dialog is very similar

Converting to a Fortran Project

- Are there categories labeled **Fortran General** and **Fortran Build** in the project properties?

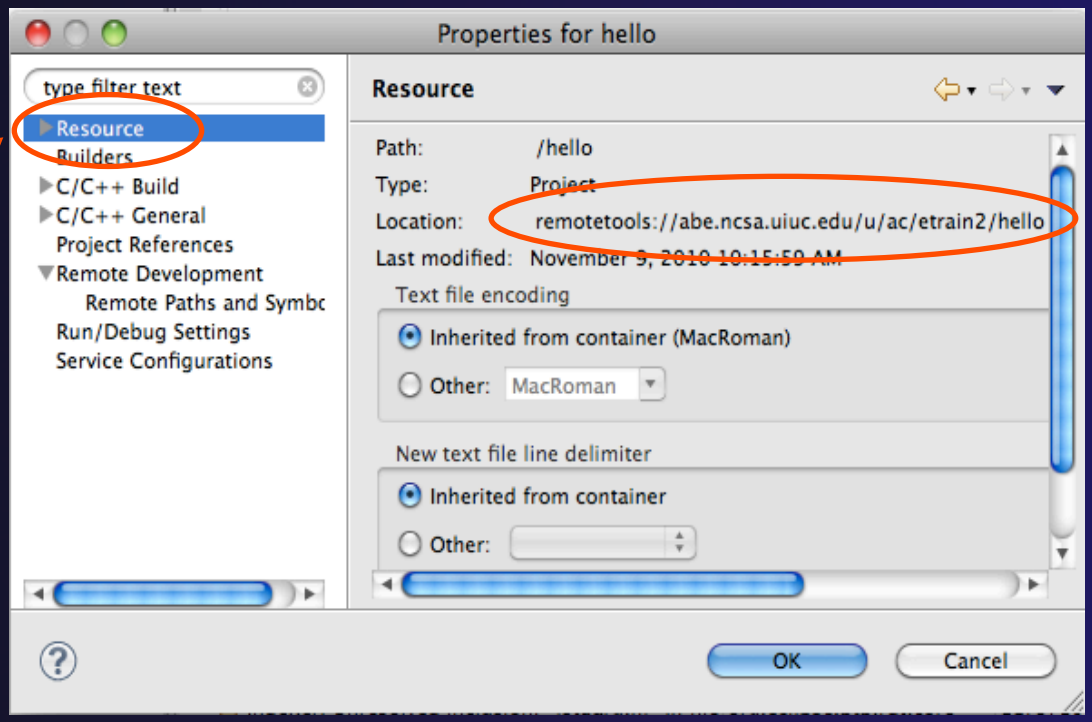
No Fortran categories →



- If not, the project is not a Fortran Project
  - Switch to the Fortran Perspective
  - In the Fortran Projects view, right-click on the project, and click **Convert to Fortran Project**
  - Don't worry; it's still a C/C++ project, too
- *Every* Fortran project is also a C/C++ Project

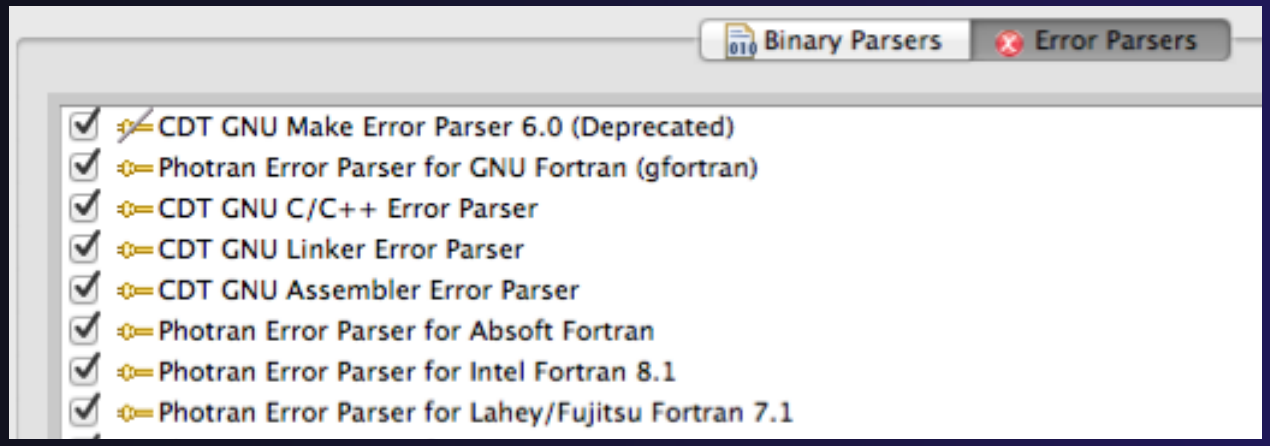
Project Location

- How to tell where a project resides?
- In the project properties dialog, select the **Resource** category



Error Parsers

- Are compiler errors not appearing in the Problems view?
  - Make sure the correct *error parser* is enabled
  - In the project properties, navigate to **C++ Build ▸ Settings** or **Fortran Build ▸ Settings**
  - Switch to the **Error Parsers** tab
  - Check the error parser(s) for your compiler(s)



Fortran Source Form Settings

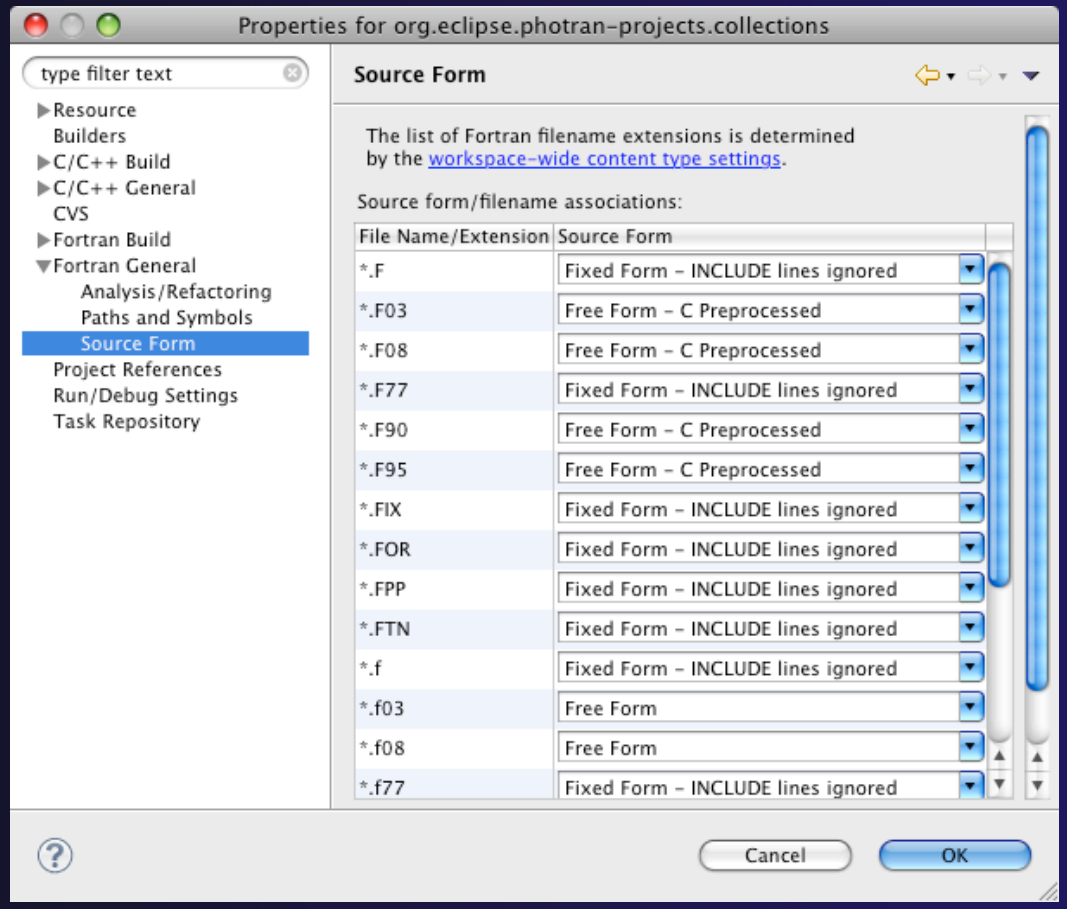
- Fortran files are either *free form* or *fixed form;* some Fortran files are *preprocessed* (#define, #ifdef, etc.)
  - Source form determined by filename extension
  - Defaults are similar to most Fortran compilers:

Fixed form:	.f	.fix	.for	.fpp	.ftn	.f77	
Free form:	.f08	.f03	.f95	.f90		< unpreprocessed	
	.F08		.F03	.F95	.F90		< preprocessed

- Many features *will not work* if filename extensions are associated with the wrong source form (outline view, content assist, search, refactorings, etc.)

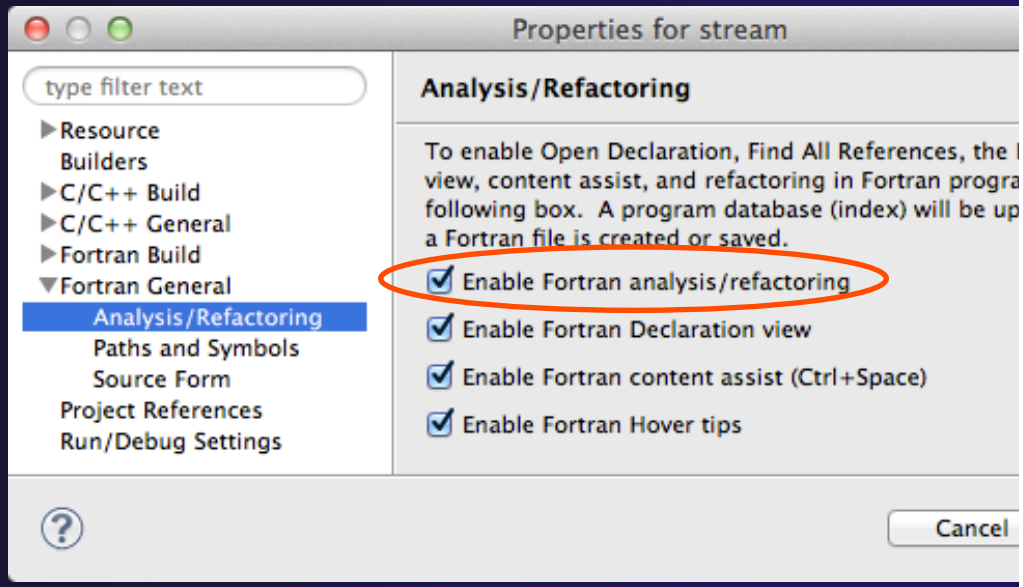
Fortran Source Form Settings

- In the project properties, select **Fortran General ▸ Source Form**
- Select source form for each filename extension
- Click **OK**



Enabling Fortran Advanced Features

- Some Fortran features are *disabled* by default
- Must be explicitly enabled
  - In the project properties dialog, select **Fortran General ▸ Analysis/Refactoring**
  - Click **Enable Analysis/ Refactoring**
  - Close and re-open any Fortran editors
- This turns on the "Photran Indexer"
  - Turn it off if it's slow



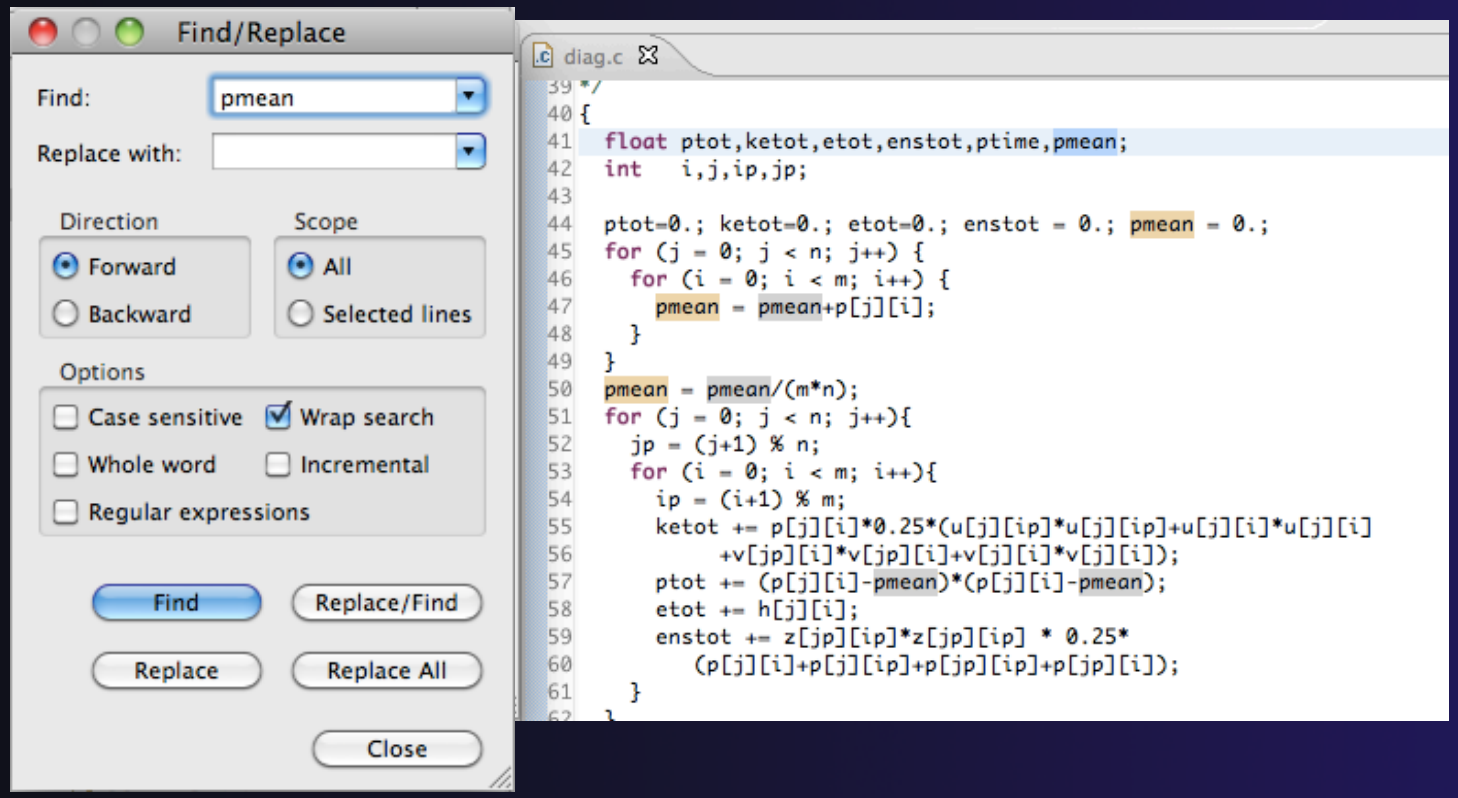
Project Properties – Try It!

1. Convert shallow to a Fortran project
2. Make sure errors from the GNU Fortran compiler will be recognized
3. Make sure *.f90 files are treated as unpreprocessed, free source form
4. Make sure search and refactoring will work in Fortran

Searching

Find/Replace within Editor

- Simple Find within editor buffer
- Ctrl-F (Mac: Command-F)



Mark Occurrences

(C/C++ Only)

- Double-click on a variable in the CDT editor
- All occurrences in the source file are highlighted to make locating the variable easier
- Alt-shift-O to turn off

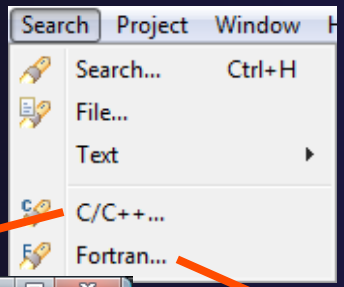
hello.c

```
13
14 int main(void) {
15     int var;
16     int max=99;
17     puts("!!!Hello World!!!"); /* prints !!!Hello
18     getenv("LANG");
19     for (var = 0; var < max; ++var) {
20     }
21     return EXIT_SUCCESS;
22 }
```

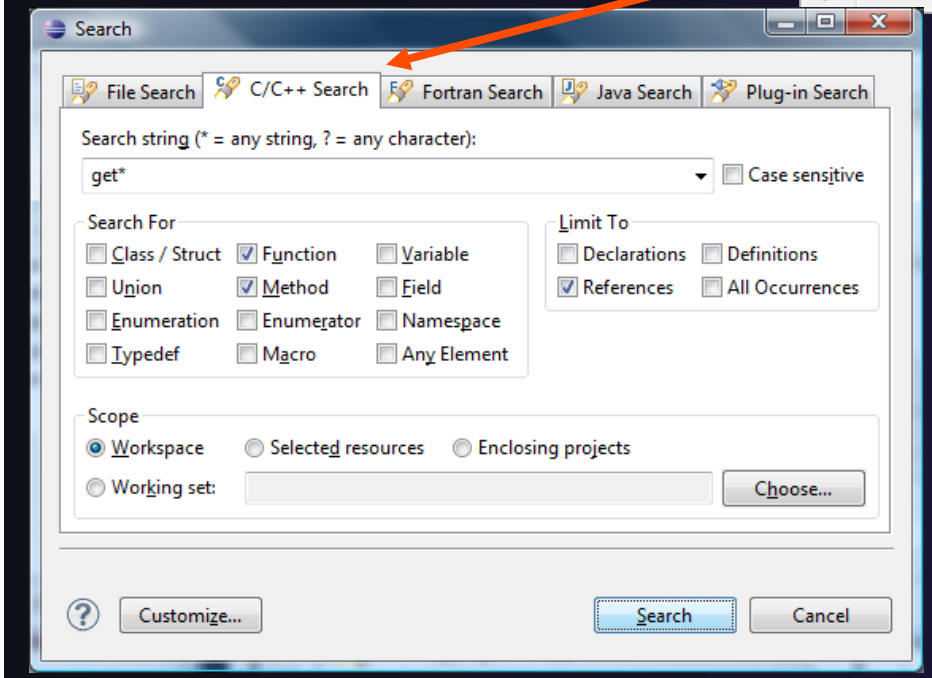
Language-Based Searching

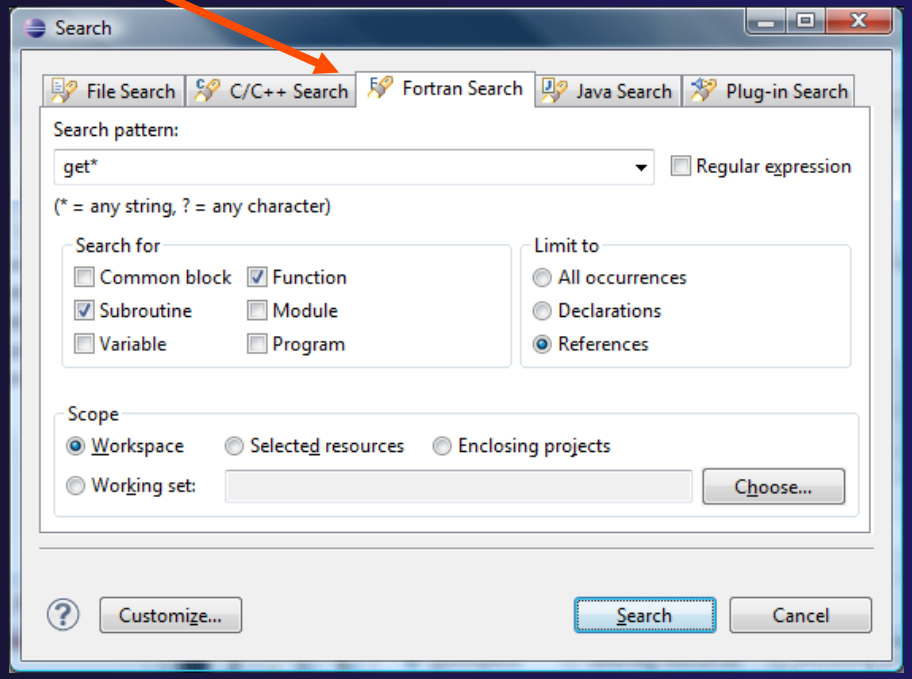
(C/C++ and Fortran)

- "Knows" what things can be declared in each language (functions, variables, classes, modules, etc.)



- E.g., search for every call to a function whose name starts with "get"
- Search can be project- or workspace-wide





Find References

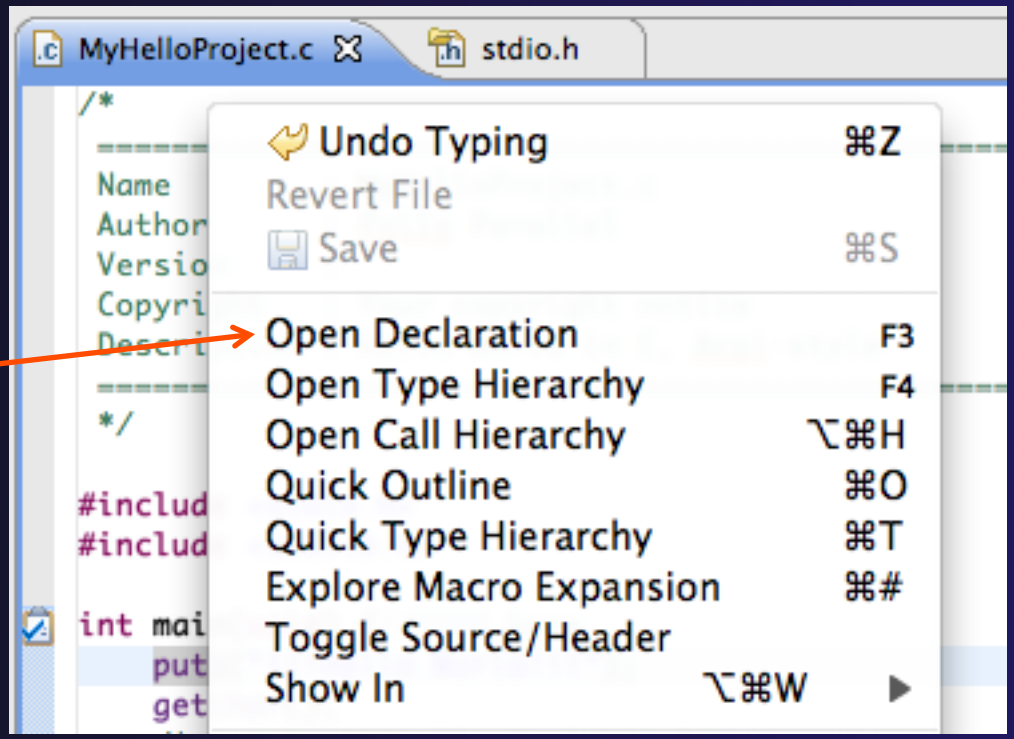
(C/C++ and Fortran)

- Finds all of the places where a variable, function, etc., is used
  - Right-click on an identifier
  - Click **References ▸ Workspace** or **References ▸ Project**

Open Declaration

(C/C++ and Fortran)

- Jumps to the declaration of a variable, function, etc., even if it's in a different file
- Right-click on an identifier
- Click **Open Declaration**
- Can also Ctrl-click (Mac: Cmd-click) on an identifier to "hyperlink" to its declaration



Search – Try It!

1. Find every call to `MPI_Recv` in Shallow.
2. In worker.c, on line 47, there is a declaration `float p[n][m]`.
   a) What is `m`?
   b) Where is `m` defined?
   c) How many times is `m` used in the project?
3. Find every function whose name contains the word `time`

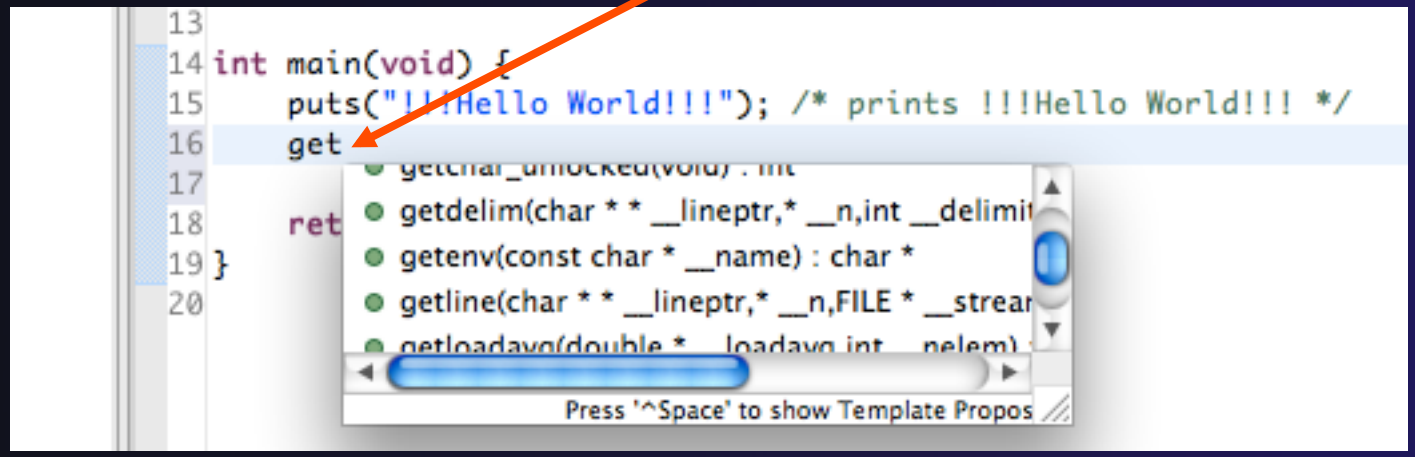
Advanced Editing

Content Assist and Code Templates

Content Assist

(C/C++ and Fortran)

- Auto-complete names of variables, functions, etc.
- Type an incomplete function name e.g. “get” into the editor, and hit **Ctrl-Space**
- Type more characters to narrow the list
- Use up/down arrow keys to browse the list
- Hit **Enter** to insert the highlighted completion



Code Templates

(C/C++ and Fortran)

- Auto-complete common code patterns
  - For loops/do loops, if constructs, etc.
  - Also MPI code templates
- Included with content assist proposals (when **Ctrl-Space** is pressed)
  - May need to press **Ctrl-Space** a 2nd time in C/C++
  - Press **Tab** to move between completion fields

```
17    for
18          for - for loop                    for (int var = 0; var < max; ++var) {
19    ret   for - for loop with temporary variable
20 }                                          }
21
```

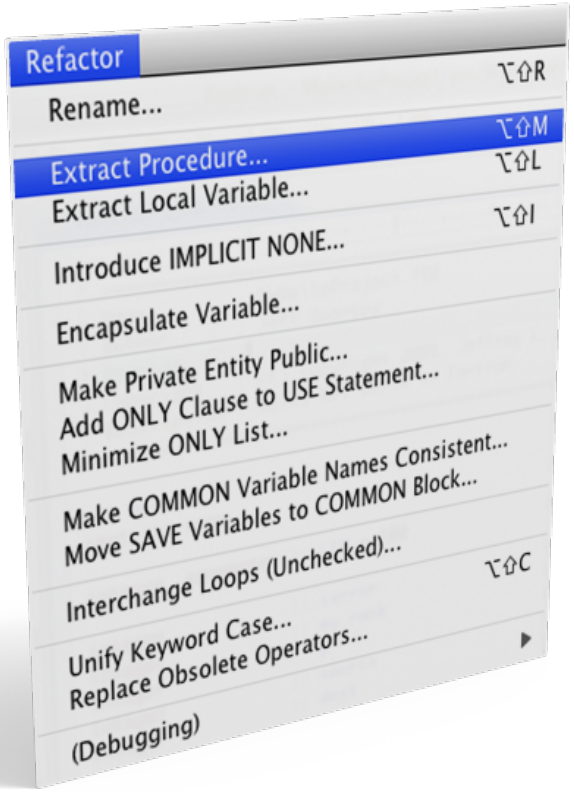
Advanced Editing – Try It!

- Open tstep.f90 and retype the last loop nest
  - Use the code template to complete the do-loops
  - Use content assist to complete variable names

Refactoring and Transformation

Refactoring

(making changes to source code that don't affect the behavior of the program)

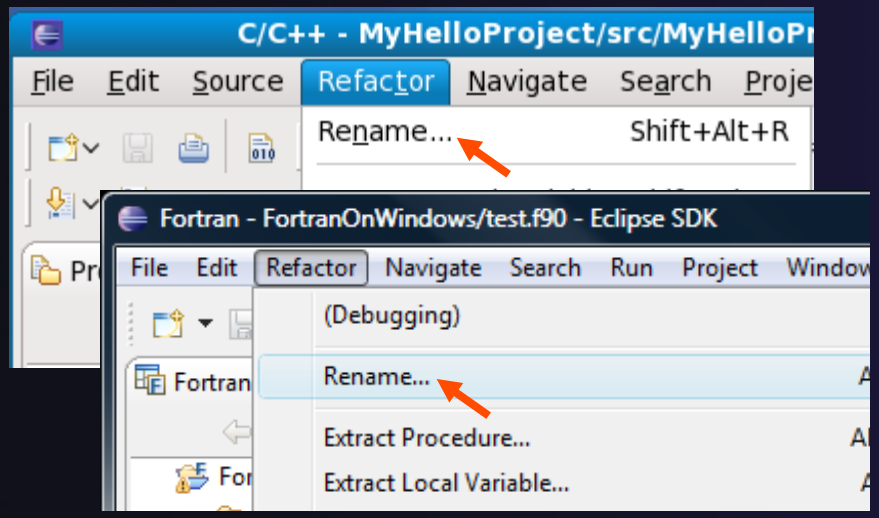


- Refactoring is the research motivation for Photran @ Illinois
  - Illinois is a leader in refactoring research
  - "Refactoring" was coined in our group (Opdyke & Johnson, 1990)
  - We had the first dissertation… (Opdyke, 1992)
  - …and built the first refactoring tool… (Roberts, Brant, & Johnson, 1997)
  - …and first supported the C preprocessor (Garrido, 2005)
  - Photran's agenda: refactorings for HPC, language evolution, refactoring framework
- Photran 7.0: 31 refactorings

Rename Refactoring

(also available in C/C++)

- Changes the name of a variable, function, etc., *including every use*
  (change is semantic, not textual, and can be workspace-wide)
- Only proceeds if the new name will be legal
  (aware of scoping rules, namespaces, etc.)

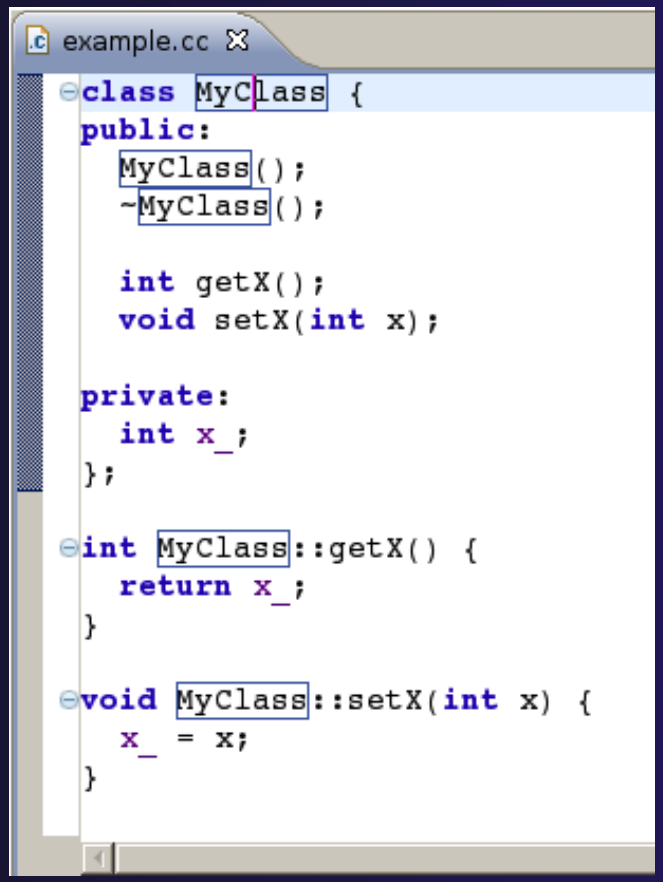


- Select **Fortran Perspective**
- Open a source file
- Click in editor view on declaration of a variable
- Select menu item **Refactor ▸ Rename**
  - Or use context menu
- Enter new name

Rename in File

(C/C++ Only)

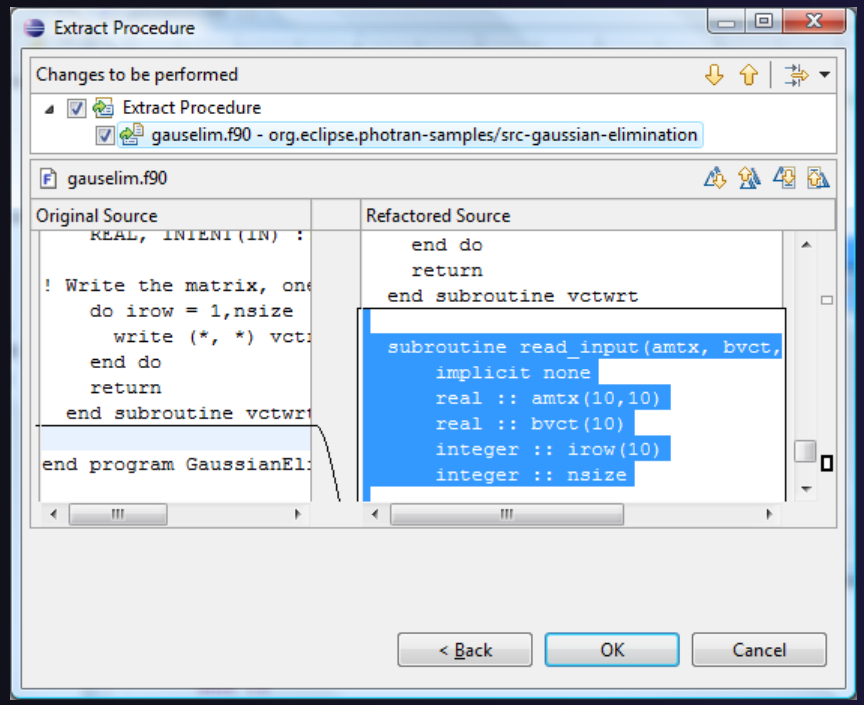
- Position the caret over an identifier.
- Press Ctrl+1 (Command+1 on Mac).
- Enter a new name. Changes are propagated within the file as you type.



Extract Procedure Refactoring

(also available in C/C++ - "Extract Function")

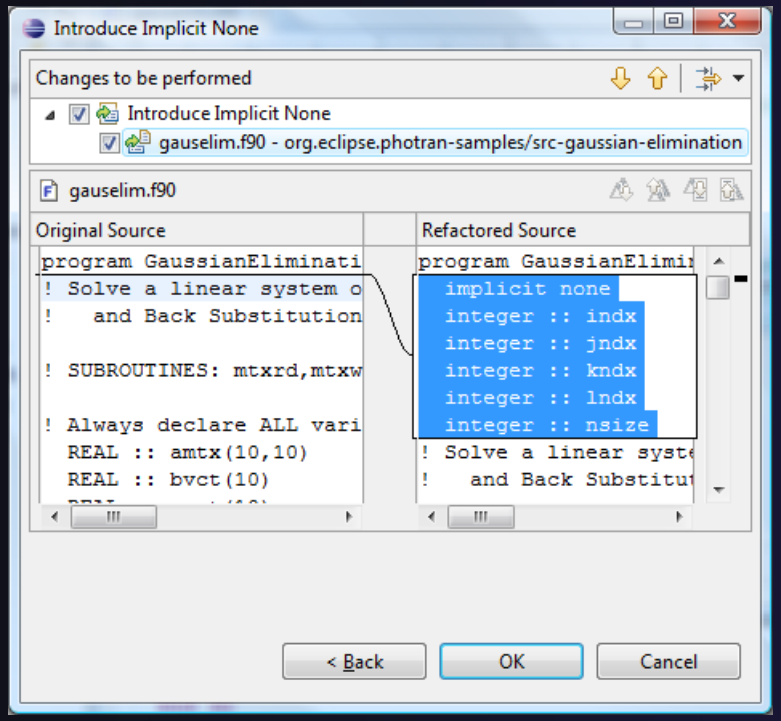
- Moves statements into a new subroutine, replacing the statements with a call to that subroutine
- Local variables are passed as arguments



- Select a sequence of statements
- Select menu item **Refactor ▸ Extract Procedure...**
  - Or use context menu
- Enter new name

Introduce IMPLICIT NONE Refactoring

- Fortran does not require variable declarations
  (by default, names starting with I-N are integer variables; others are reals)
- This adds an IMPLICIT NONE statement and adds explicit variable declarations for all implicitly declared variables



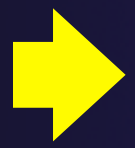
- Introduce in a single file by opening the file and selecting **Refactor ▸ Introduce IMPLICIT NONE...**
- Introduce in multiple files by selecting them in the Fortran Projects view, right-clicking on the selection, and choosing **Refactor ▸ Introduce IMPLICIT NONE...**

Loop Transformations

(Fortran only)

- **Interchange Loops** **CAUTION**: No check for behavior preservation

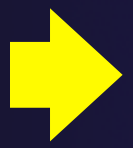
```
do i = 1, 10
  do j = 1, 5
    print *, i*10+j
  enddo
end do
```



```
do j = 1, 5
  do i = 1, 10
    print *, i*10+j
  enddo
end do
```

- **Unroll Loop**

```
do i = 1, 10
  print *, 10*i
end do
```



Unroll 5 times

```
do i = 1, 10, 5
  print *, 10*(i+0)
  print *, 10*(i+1)
  print *, 10*(i+2)
  print *, 10*(i+3)
  print *, 10*(i+4)
end do
```

Refactoring – Try It!

In tstep.f90…

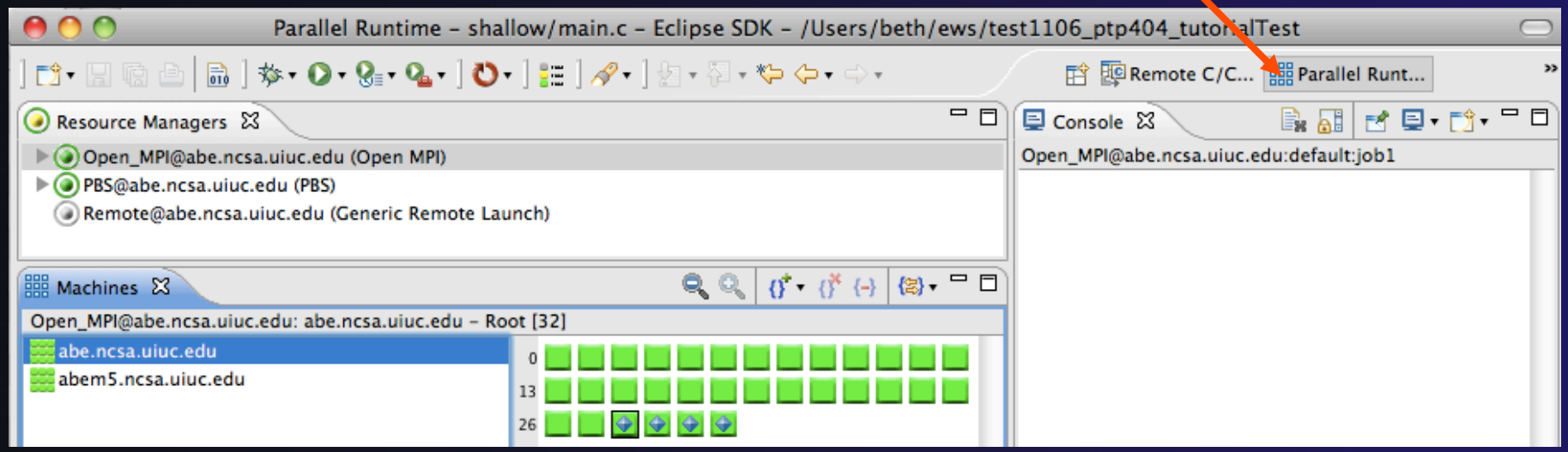
1. Make the `tstep` subroutine IMPLICIT NONE
2. Interchange the loops in all three of the double loop nests
   - Does this improve performance? If not, undo it.
3. Unroll the inner loop in each loop nest
   - Does this improve performance? If not, undo it.

Module 4: Parallel Debugging

- Objective
  - Learn the basics of debugging parallel programs
- Contents
  - Launching a debug session
  - The Parallel Debug Perspective
  - Controlling sets of processes
  - Controlling individual processes
  - Parallel Breakpoints
  - Terminating processes

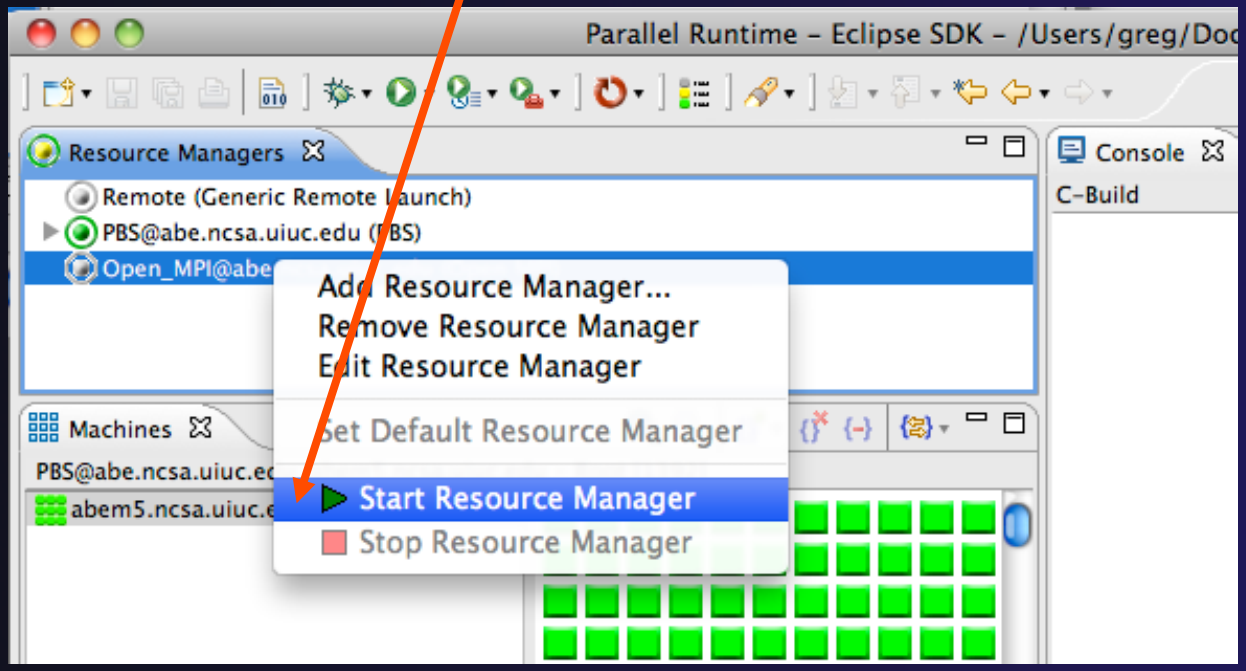
Debugging an Application

- Debugging requires interactive access to the application
- Since PBS is for batch execution, we will use Open MPI to provide interactive access to the machine (PBS will support interactive execution in the future)
- First switch to the Parallel Runtime perspective if not already there



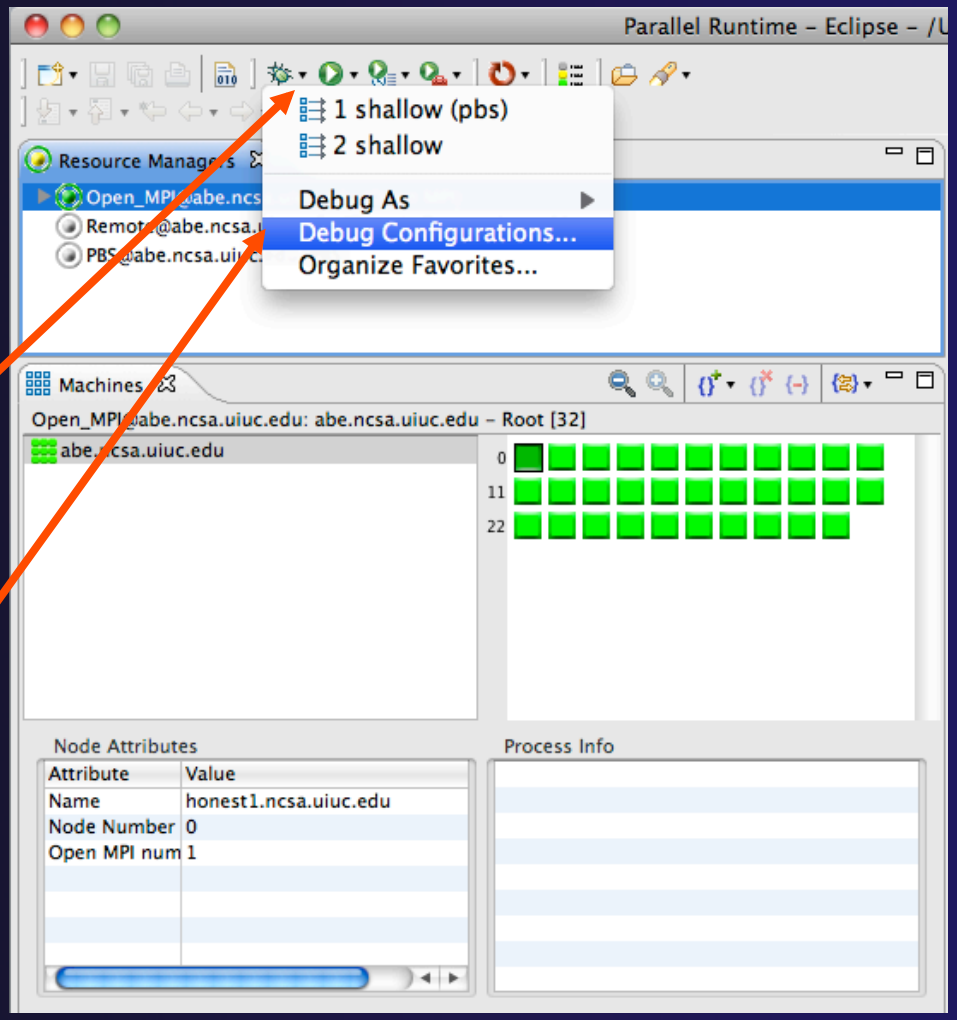
Start the Resource Manager

- If the Open_MPI Resource manager is not already started (green icon), start it now:
  - Right-click on the resource manager and select **Start Resource Manager** from the menu



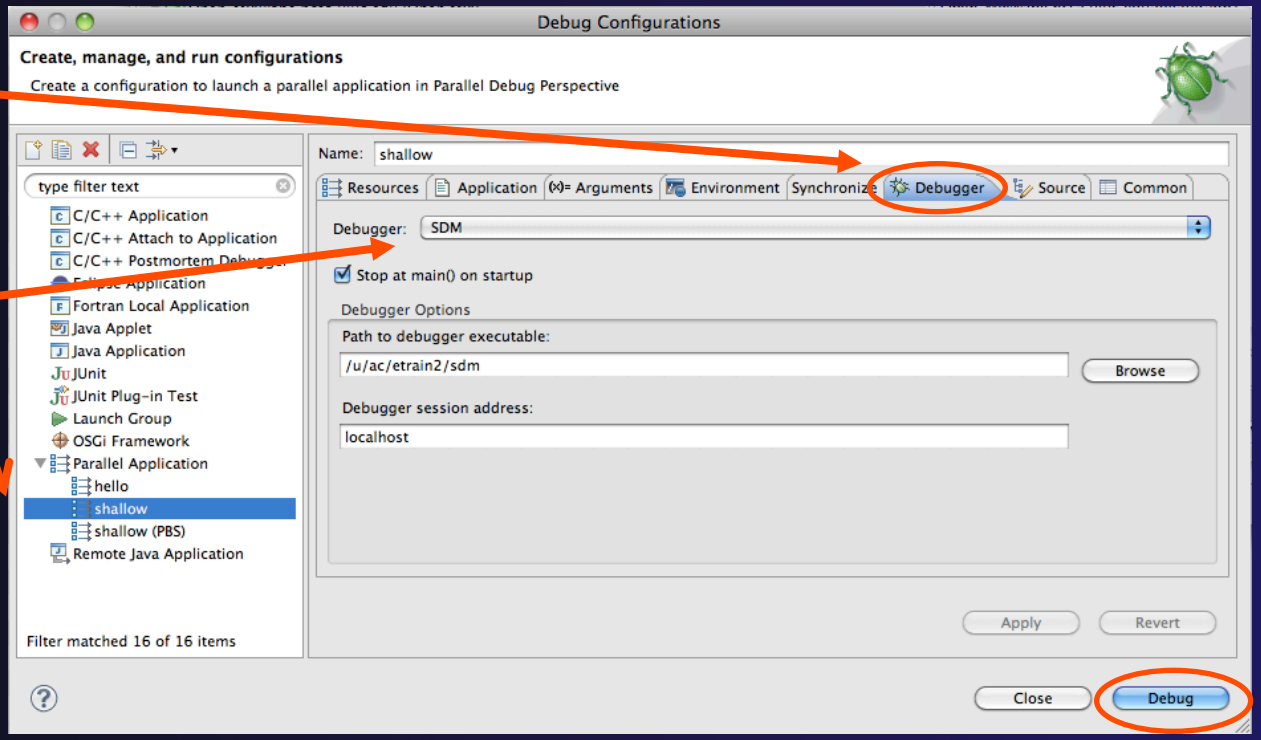
Create a Debug Configuration

- A debug configuration is essentially the same as a run configuration (like we used in modules 3 & 4)
- We will re-use the existing configuration and add debug information
- Use the drop-down next to the debug button (bug icon) instead of run button
- Select **Debug Configurations...** to open the **Debug Configurations** dialog



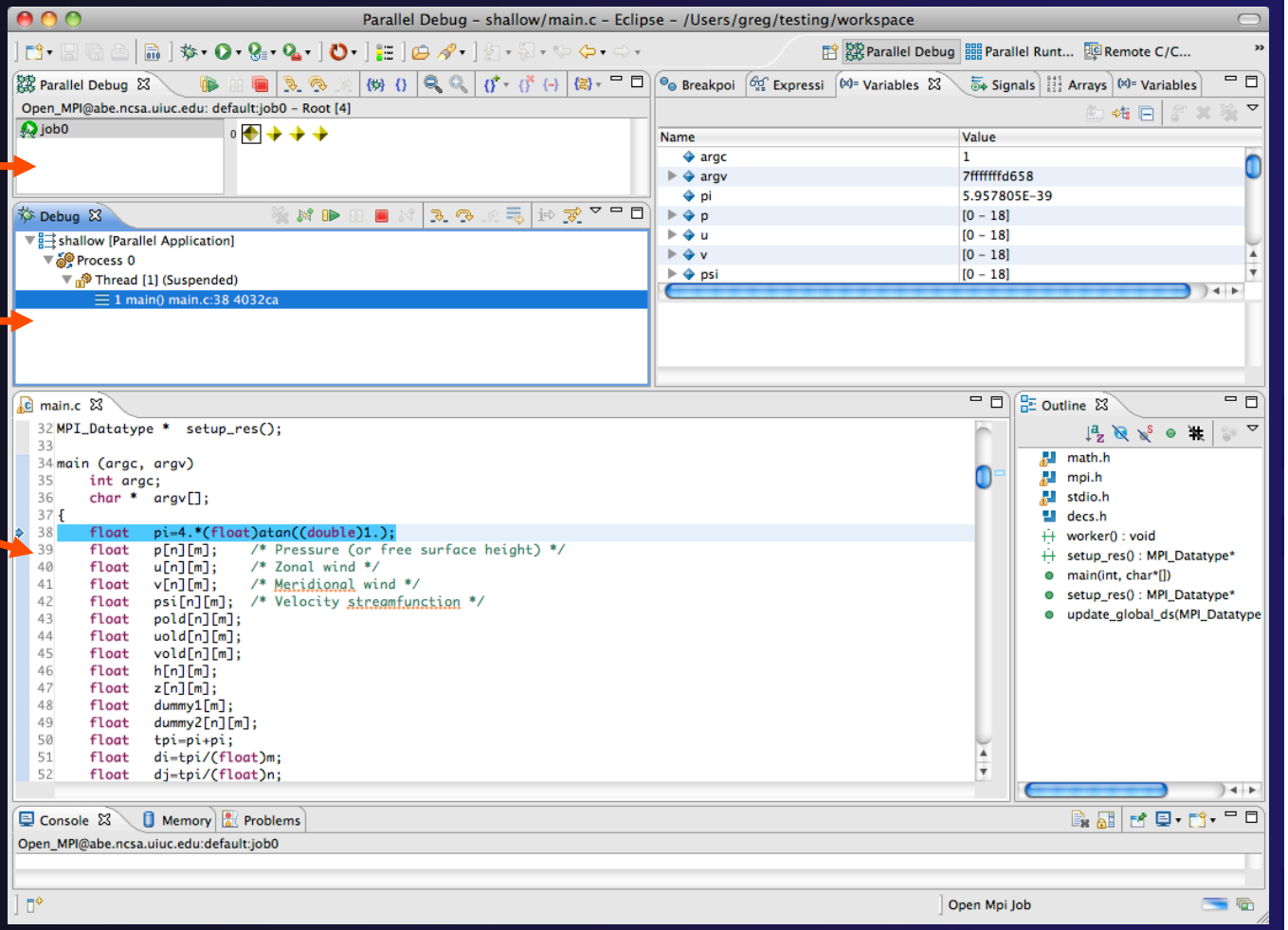
Configure the Debugger Tab

- Select **Debugger** tab
- Select the **shallow** configuration
- Make sure **SDM** is selected in the **Debugger** dropdown
- Check the debugger path is correct
  - Should be the path to the sdm executable on the remote system
- Debugger session address should not need to be changed
- Click on **Debug** to launch the program



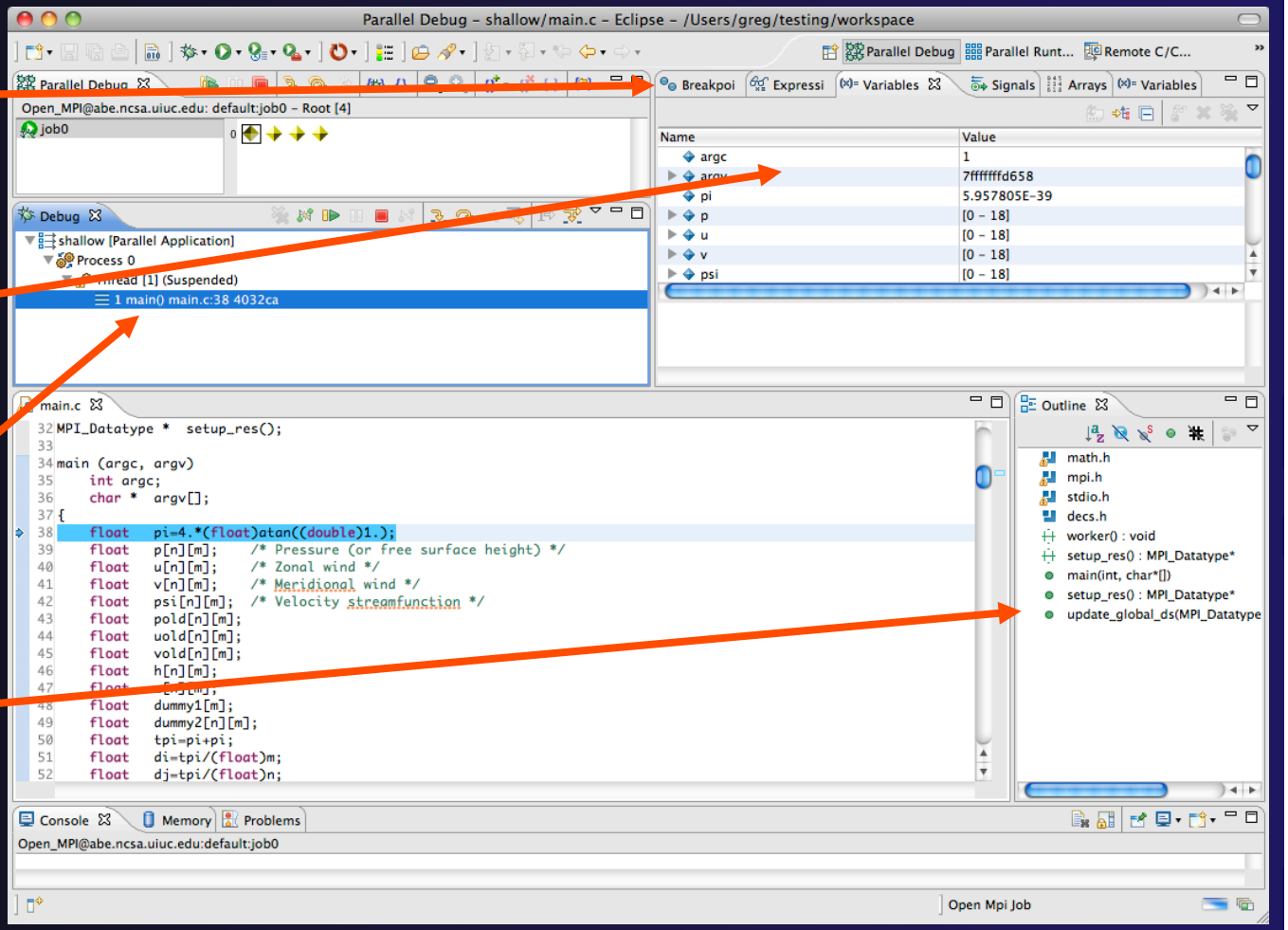
The Parallel Debug Perspective (1)

- **Parallel Debug view** shows job and processes being debugged
- **Debug** view shows threads and call stack for individual processes
- **Source** view shows a **current line marker** for all processes



The Parallel Debug Perspective (2)

- **Breakpoints** view shows breakpoints that have been set (more on this later)
- **Variables** view shows the current values of variables for the currently selected process in the **Debug** view
- **Outline** view (from CDT) of source code



Stepping All Processes

- The buttons in the **Parallel Debug View** control groups of processes
- Click on the **Step Over** button
- Observe that all process icons change to green, then back to yellow
- Notice that the current line marker has moved to the next source line

Parallel Debug – shallow/main.c – Eclipse – /Users/greg/testing/workspa

Open_MPI@abe.ncsa.uiuc.edu: default:job0 Root [4]

job0

Debug

shallow [Parallel Application]
Process 0 (Suspended)
Thread [1] (Suspended)
1 main() main.c:50 4032f6

Breakpoi Expressi Varial

Name
argc
argv
pi
p
u
v
psi

main.c

```
38  float   pi=4.*(float)atan((double)1.);
39  float   p[n][m];    /* Pressure (or free surface height) */
40  float   u[n][m];    /* Zonal wind */
41  float   v[n][m];    /* Meridional wind */
42  float   psi[n][m];  /* Velocity streamfunction */
43  float   pold[n][m];
44  float   uold[n][m];
45  float   vold[n][m];
46  float   h[n][m];
47  float   z[n][m];
48  float   dummy1[m];
49  float   dummy2[n][m];
50  float   tpi=pi+pi;
51  float   di=tpi/(float)m;
52  float   dj=tpi/(float)n;
53  int i, j, chunk_size, nxt, prv;
54
55  int master_packet[4];
56  float   p_start[m];
57  float   u_start[m];
58  float   v_start[m];
```

Stepping An Individual Process

- The buttons in the **Debug view** are used to control an individual process, in this case process 0
- Click the **Step Over** button
- You will now see two current line markers, the first shows the position of process 0, the second shows the positions of processes 1-3

Parallel Debug – shallow/main.c – Eclipse – /Users/greg/testing/works

Parallel Debug

Open_MPI@abe.ncsa.uiuc.edu: default:job0 – Root [4]

job0

Debug

shallow [Parallel Application]

Process 0 (Suspended)

Thread [1] (Suspended)

1 main() main.c:51 403309

Breakpoi | Expressi | Var

Name

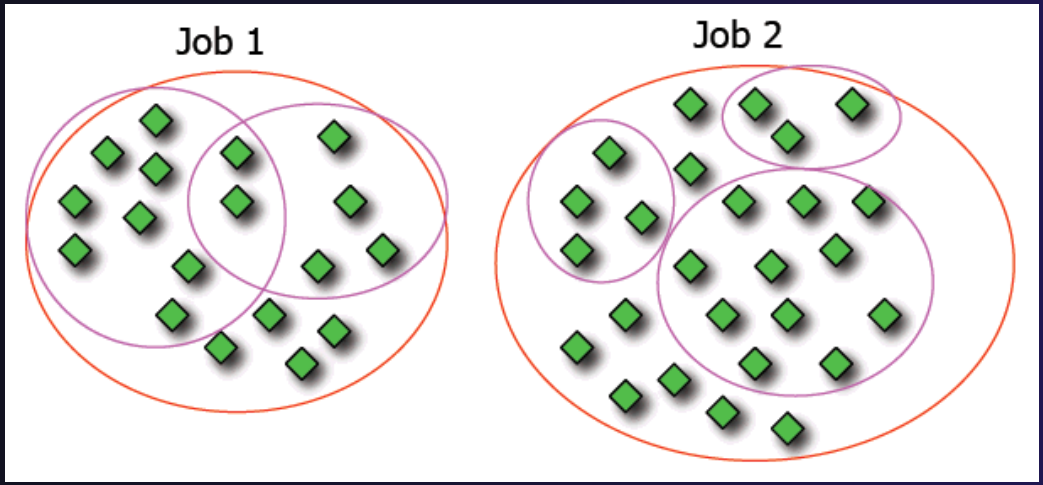
argc
argv
pi
p
u
v
psi

main.c

```
38  float   pi=4.*(float)atan((double)1.);
39  float   p[n][m];    /* Pressure (or free surface height) */
40  float   u[n][m];    /* Zonal wind */
41  float   v[n][m];    /* Meridional wind */
42  float   psi[n][m];  /* Velocity streamfunction */
43  float   pold[n][m];
44  float   uold[n][m];
45  float   vold[n][m];
46  float   h[n][m];
47  float   z[n][m];
48  float   dummy1[m];
49  float   dummy2[n][m];
50  float   tpi=pi+pi;
51  float   di=tpi/(float)m;
52  float   dj=tpi/(float)n;
53  int i, j, chunk_size, nxt, prv;
54
55  int master_packet[4];
56  float   p_start[m];
57  float   u_start[m];
```

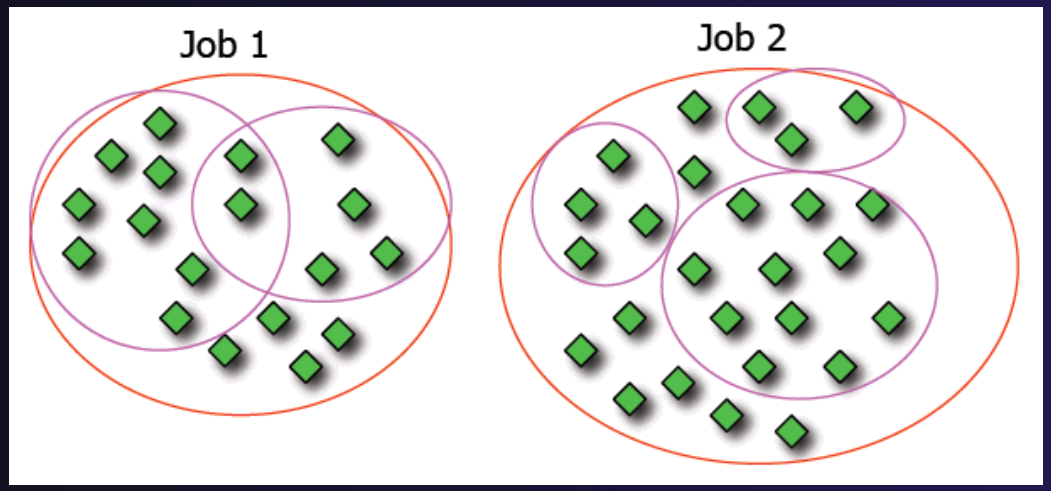
Process Sets (1)

- Traditional debuggers apply operations to a single process
- Parallel debugging operations apply to a single process or to arbitrary collections of processes
- A process set is a means of simultaneously referring to one or more processes



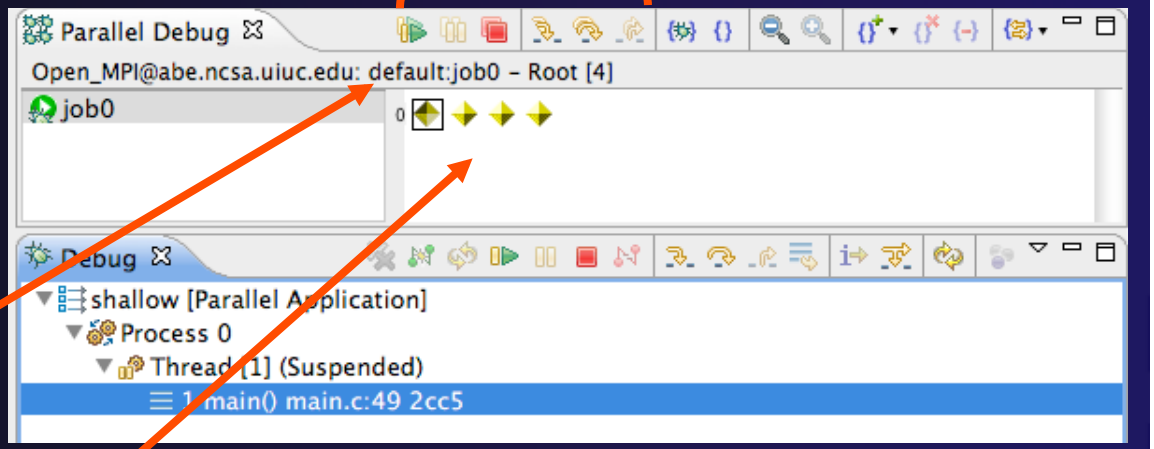
Process Sets (2)

- When a parallel debug session is first started, all processes are placed in a set, called the **Root** set
- Sets are always associated with a single job
- A job can have any number of process sets
- A set can contain from 1 to the number of processes in a job



Operations On Process Sets

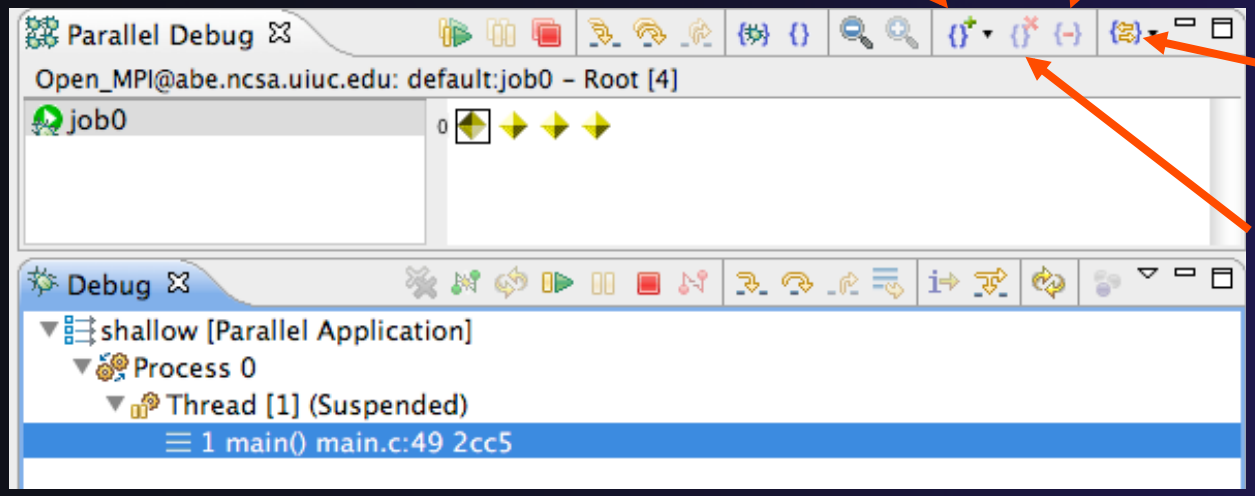
- Debug operations on the **Parallel Debug view** toolbar always apply to the current set:
  - Resume, suspend, stop, step into, step over, step return
- The current process set is listed next to job name along with number of processes in the set
- The processes in process set are visible in right hand part of the view



Root set = all processes

Managing Process Sets

- The remaining icons in the toolbar of the **Parallel Debug view** allow you to create, modify, and delete process sets, and to change the current process set



Create set

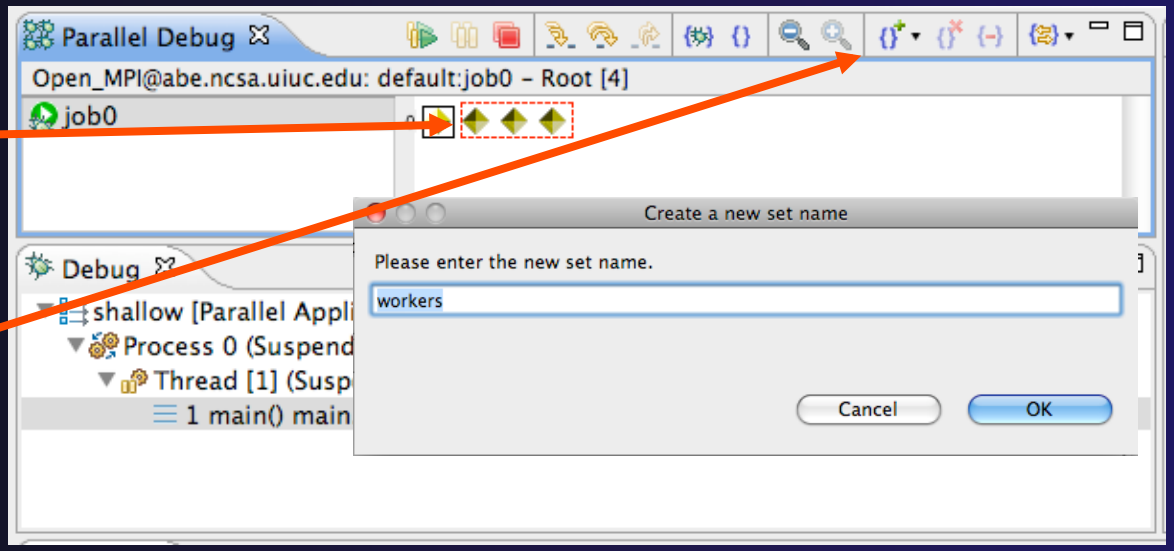
Remove from set

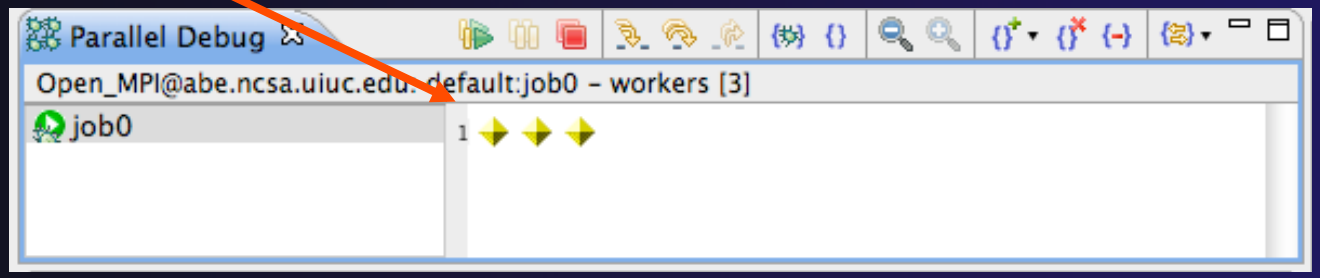
Change current set

Delete set

Creating A New Process Set

- Select the processes you want in the set by clicking and dragging, in this case, the last three
- Click on the **Create Set** button
- Enter a name for the set, in this case **workers**, and click **OK**
- You will see the view change to display only the selected processes





Stepping Using New Process Set

- With the **workers** set active, click the **Step Over** button
- You will see only the first current line marker move
- Step a couple more times
- You should see two line markers, one for the single master process, and one for the 3 worker processes

Parallel Debug – shallow/main.c – Eclipse – /Users/greg/testing/workspa

Parallel Debug

Open_MPI@abe.ncsa.uiuc.edu: default:job0 – workers [3]

job0

Debug

shallow [Parallel Application]

main.c

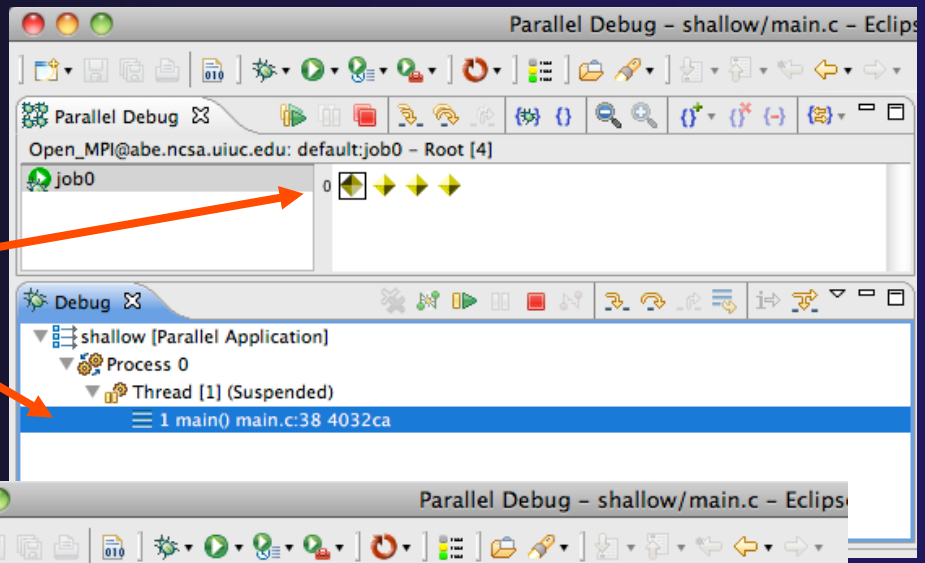
```
50    float   tpi=pi+pi;
51    float   di=tpi/(float)m;
52    float   dj=tpi/(float)n;
53    int i, j, chunk_size, nxt, prv;
54
55    int master_packet[4];
56    float   p_start[m];
57    float   u_start[m];
58    float   v_start[m];
59    float   psi_start[m];
60    float   pold_start[m];
61    float   uold_start[m];
62    float   vold_start[m];
63    int proc_cnt;
64    int tid;
65    MPI_Datatype *  res_type;
66
67    MPI_Init(&argc, &argv);
68    MPI_Comm_size(MPI_COMM_WORLD, &proc_cnt);//hello
69    MPI_Comm_rank(MPI_COMM_WORLD, &tid);
```

Process Registration

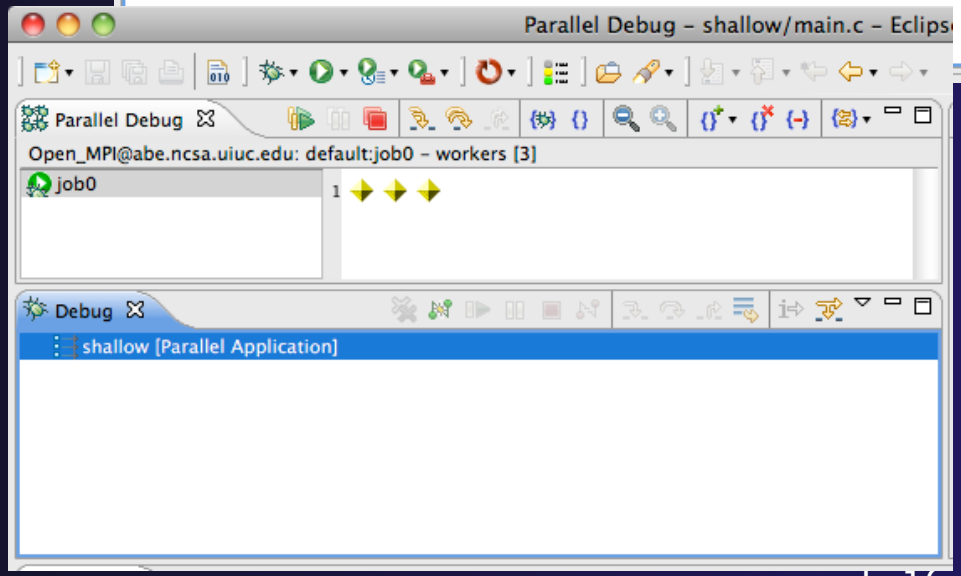
- Process set commands apply to groups of processes
- For finer control and more detailed information, a process can be registered and isolated in the **Debug view**
- Registered processes, including their stack traces and threads, appear in the **Debug view**
- Any number of processes can be registered, and processes can be registered or un-registered at any time

Process Registration (2)

- By default, process 0 was registered when the debug session was launched
- Registered processes are surrounded by a box and shown in the Debug view



- The Debug view only shows registered processes in the current set
- Since the "workers" set doesn't include process 0, it is no longer displayed in the Debug view



Registering A Process

- To register a process, double-click its process icon in the **Parallel Debug view** or select a number of processes and click on the **register** button
- To un-register a process, double-click on the process icon or select a number of processes and click on the **unregister** button

Parallel Debug – shallow/main.c –

Open_MPI@abe.ncsa.uiuc.edu: default:job0 – workers [3]

job0

Groups (sets) of processes

Debug

shallow [Parallel Application]
Process 3 (Suspended)
Thread [1] (Suspended)
1 main() main.c:67 403335

Individual (registered) processes

calc.c | main.c | test_global_bp.c

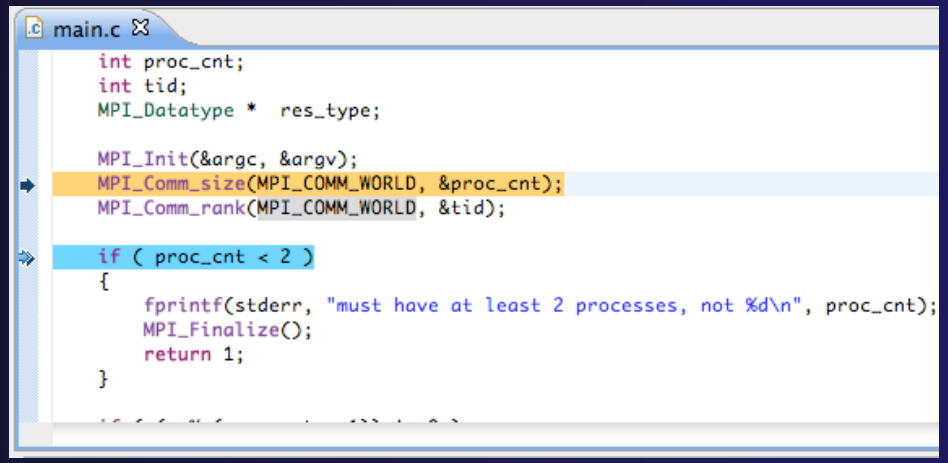
```
60    float   pold_start[m];
61    float   uold_start[m];
62    float   vold_start[m];
63    int proc_cnt;
64    int tid;
65    MPI_Datatype *  res_type;
66
67    MPI_Init(&argc, &argv);
68    MPI_Comm_size(MPI_COMM_WORLD, &proc_cnt);//hello
69    MPI_Comm_rank(MPI_COMM_WORLD, &tid);
70
71    fprintf(stdout, "my rank is %d\n", tid);
72
73    if ( proc_cnt < 2 )
74    {
```

Current Line Marker

- The current line marker is used to show the current location of suspended processes
- In traditional programs, there is a single current line marker (the exception to this is multi-threaded programs)
- In parallel programs, there is a current line marker for every process
- The PTP debugger shows one current line marker for every group of processes at the same location

Colors And Markers

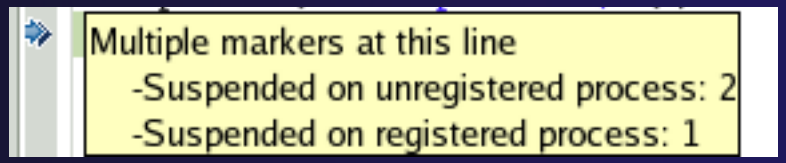
- The highlight color depends on the processes suspended at that line:
  - **Blue:** All registered process(es)
  - **Orange:** All unregistered process (es)
  - **Green:** Registered or unregistered process with no source line (e.g. suspended in a library routine)
- The marker depends on the type of process stopped at that location
- Hover over marker for more details about the processes suspend at that location




Multiple processes marker

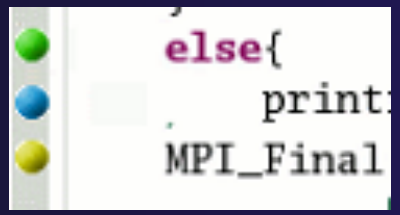

Registered process marker


Un-registered process marker



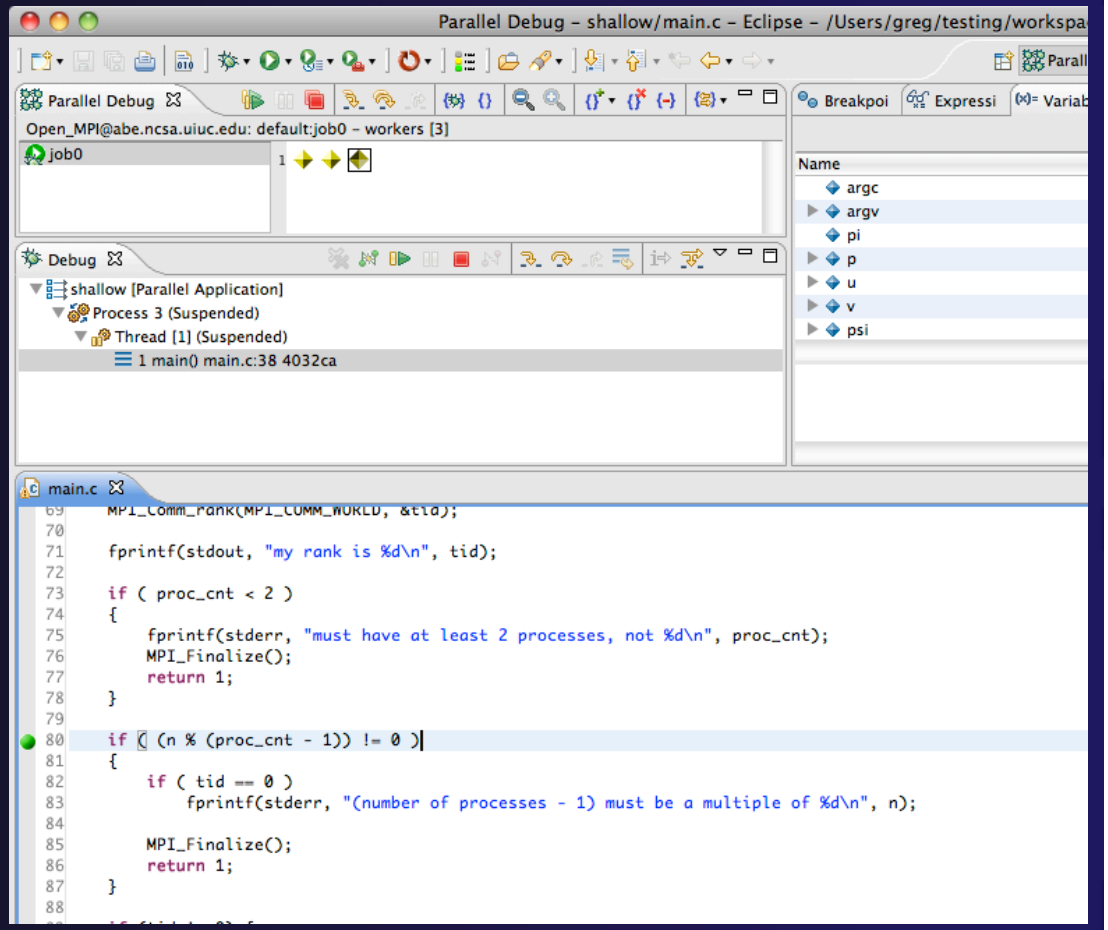
Breakpoints

- Apply only to processes in the particular set that is active in the **Parallel Debug view** when the breakpoint is created
- Breakpoints are colored depending on the active process set and the set the breakpoint applies to:
  - Green indicates the breakpoint set is the same as the active set.
  - Blue indicates some processes in the breakpoint set are also in the active set (i.e. the process sets overlap)
  - Yellow indicates the breakpoint set is different from the active set (i.e. the process sets are disjoint)
- When the job completes, the breakpoints are automatically removed



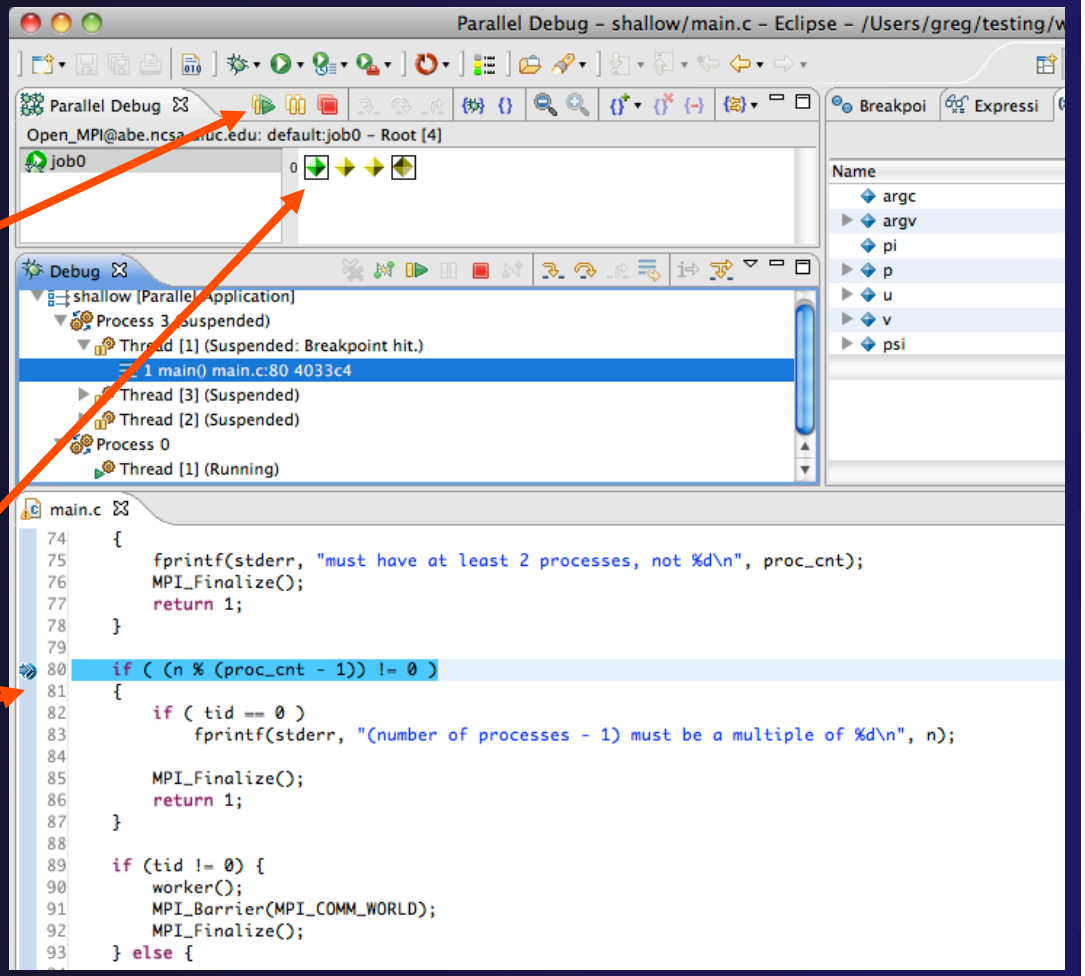
Creating A Breakpoint

- Select the process set that the breakpoint should apply to, in this case, the **workers** set
- Double-click on the left edge of an editor window, at the line on which you want to set the breakpoint, or right click and use the **Parallel Breakpoint ▶ Toggle Breakpoint** context menu
- The breakpoint is displayed on the marker bar



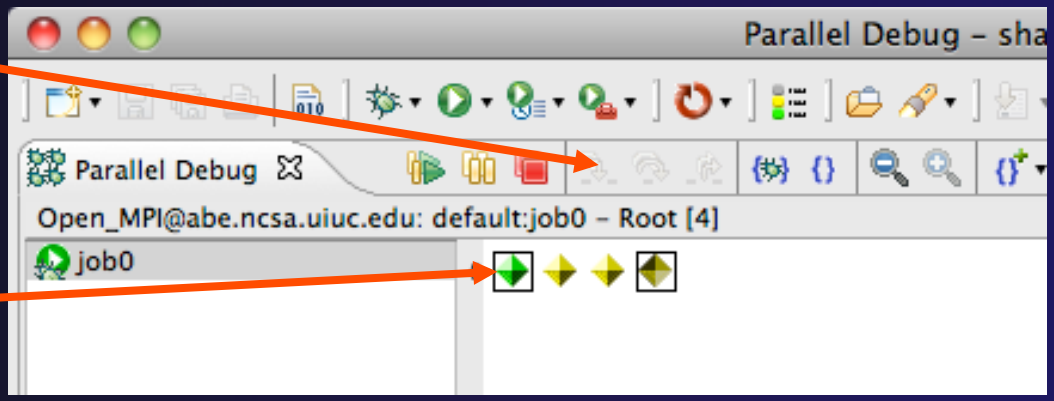
Hitting the Breakpoint

- Switch back to the **Root** set by clicking on the **Change Set** button
- Click on the **Resume** button in the **Parallel Debug view**
- In this example, the three worker processes have hit the breakpoint, as indicated by the yellow process icons and the current line marker
- Process 0 is still running as its icon is green
- Processes 1-3 are suspended on the breakpoint

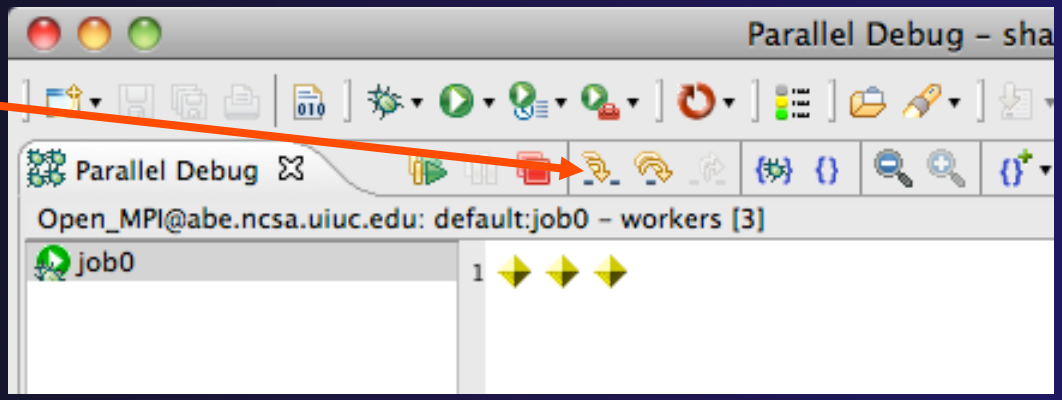


More On Stepping

- The **Step** buttons are only enabled when all processes in the active set are **suspended** (yellow icon)
- In this case, process 0 is still running

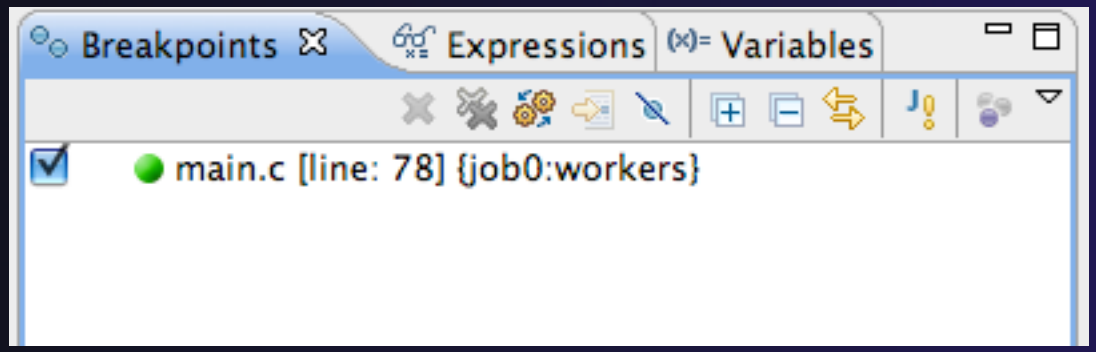


- Switch to the set of suspended processes (the **workers** set)
- You will now see the **Step** buttons become enabled



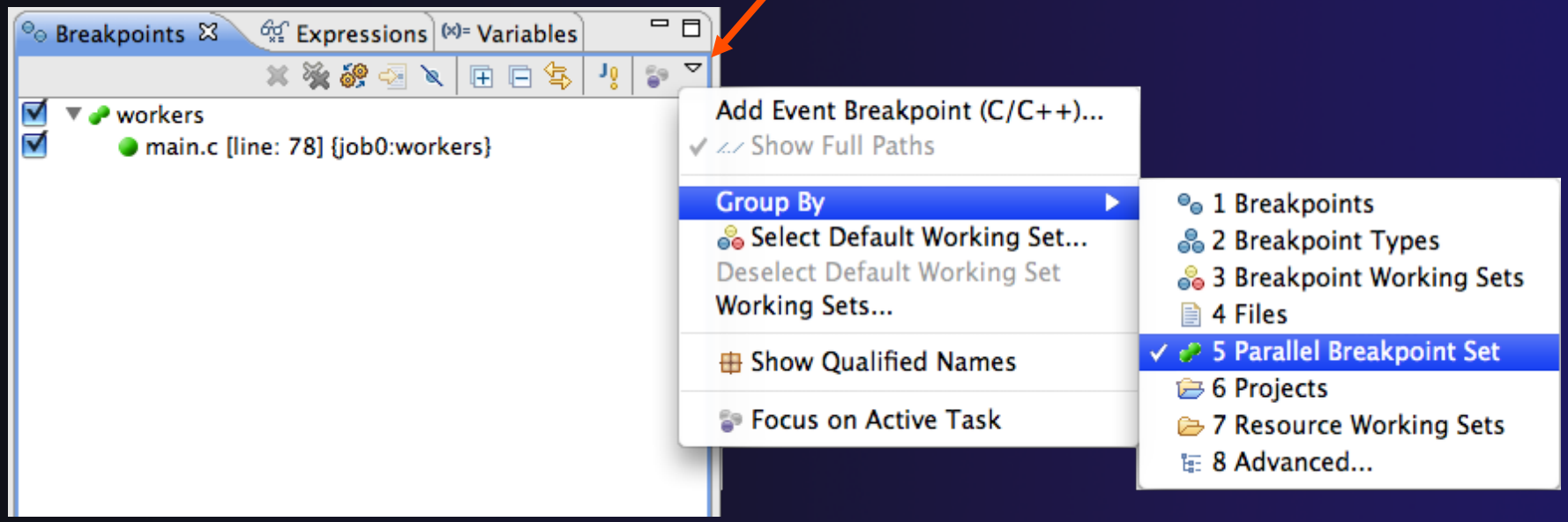
Breakpoint Information

- Hover over breakpoint icon
  - Will show the sets this breakpoint applies to
- Select **Breakpoints** view
  - Will show all breakpoints in all projects



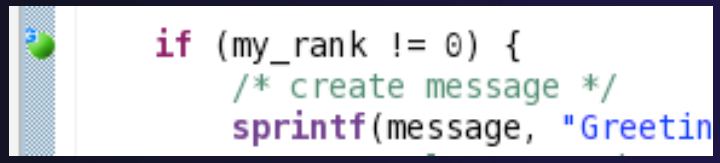
Breakpoints View

- Use the menu in the breakpoints view to group breakpoints by type
- Breakpoints sorted by breakpoint set (process set)



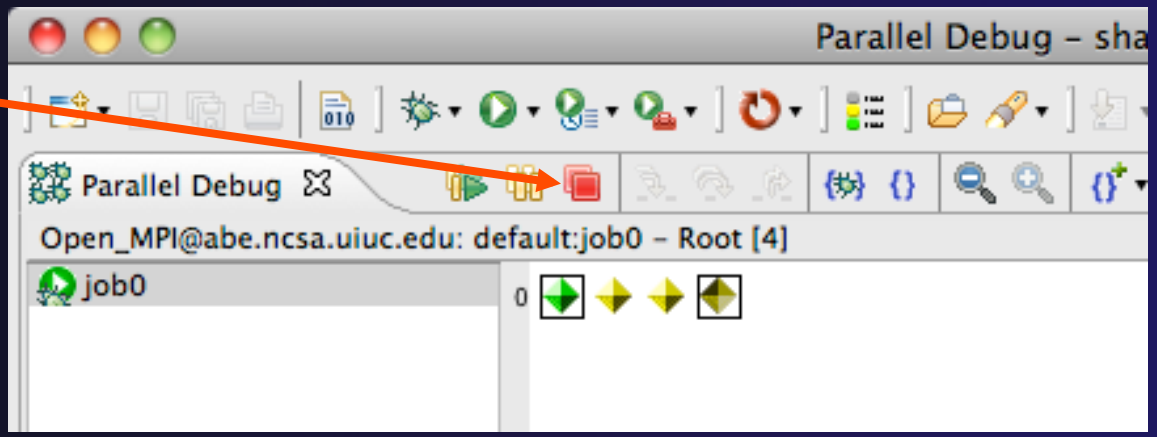
Global Breakpoints

- Apply to all processes and all jobs
- Used for gaining control at debugger startup
- To create a global breakpoint
  - First make sure that no jobs are selected (click in white part of jobs view if necessary)
  - Double-click on the left edge of an editor window
  - Note that if a job is selected, the breakpoint will apply to the current set

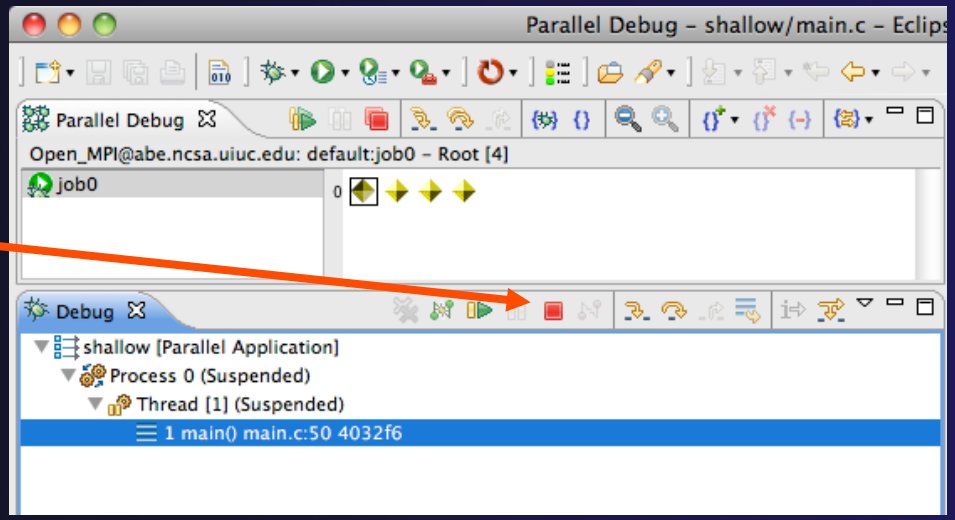


Terminating A Debug Session

- Click on the **Terminate** icon in the **Parallel Debug view** to terminate all processes in the active set
- Make sure the **Root** set is active if you want to terminate all processes



- You can also use the terminate icon in the **Debug** view to terminate the currently selected process



Module 5: Performance Tuning and Analysis Tools

- Objective
  - Become familiar with tools integrated with PTP, to help enhance performance of parallel applications
- Contents
  - Performance Tuning and other external tools:
    - PTP External Tools Framework (ETFw), TAU Hands-on exercise using TAU with PTP
    - Parallel Performance Wizard (PPW)
  - MPI Analysis: GEM (Graphical Explorer of MPI Programs)

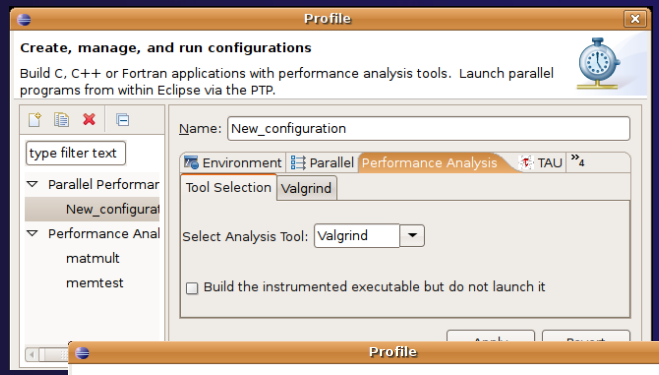
PTP/External Tools Framework

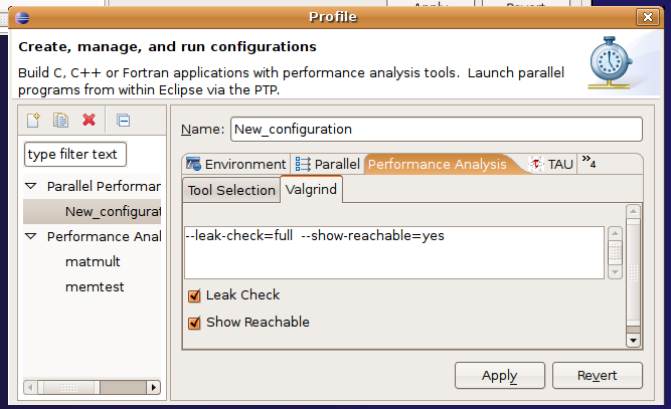
formerly "Performance Tools Framework"

Goal:

- **Reduce the "eclipse plumbing" necessary to integrate tools**
- Provide integration for instrumentation, measurement, and analysis for a variety of performance tools
  - Dynamic Tool Definitions: Workflows & UI
  - Tools and tool workflows are specified in an XML file
  - Tools are selected and configured in the launch configuration window
  - Output is generated, managed and analyzed as specified in the workflow

```
-<tool name="Valgrind">
 -<execute>
    <utility command="bash" group="inbin"/>
  -<utility command="valgrind" group="valgrind">
   -<optionpane title="Valgrind" seperatewith=" ">
      <togoption label="Leak Check" optname="--leak-check=full" tooltip
      <togoption label="Show Reachable" optname="--show-reachable=y
    </optionpane>
  </utility>
 </execute>
</tool>
```

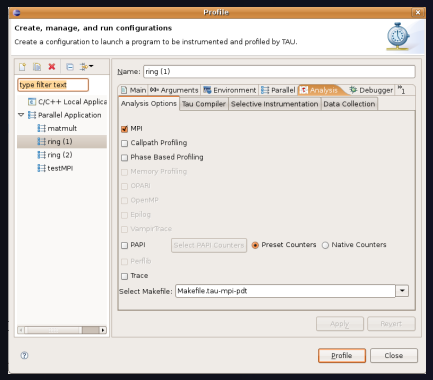


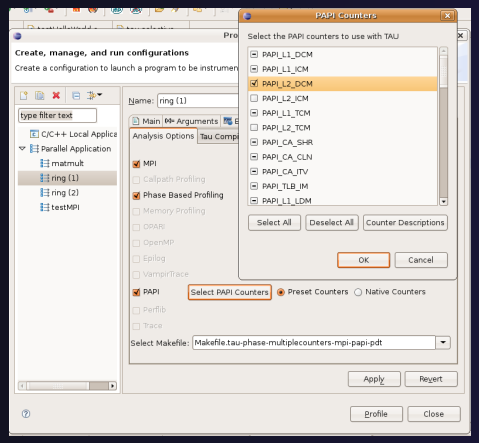


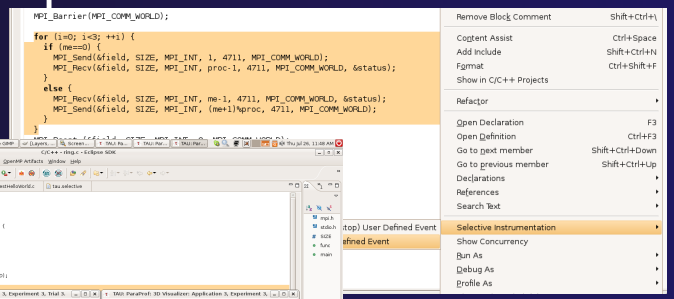
PTP TAU plug-ins

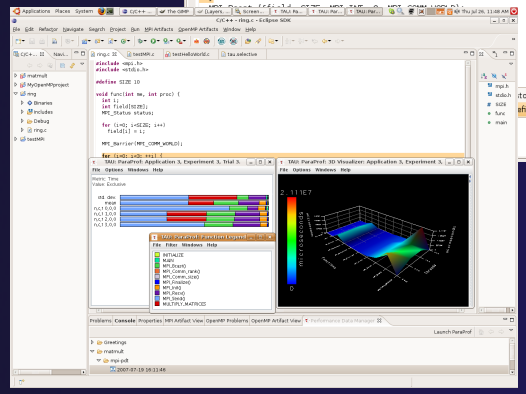
http://www.cs.uoregon.edu/research/tau

- TAU (Tuning and Analysis Utilities)
- First implementation of External Tools Framework (ETFw)
- Eclipse plug-ins wrap TAU functions, make them available from Eclipse
- Compatible with Photran and CDT projects and with PTP parallel application launching
- Other plug-ins launch Paraprof from Eclipse too



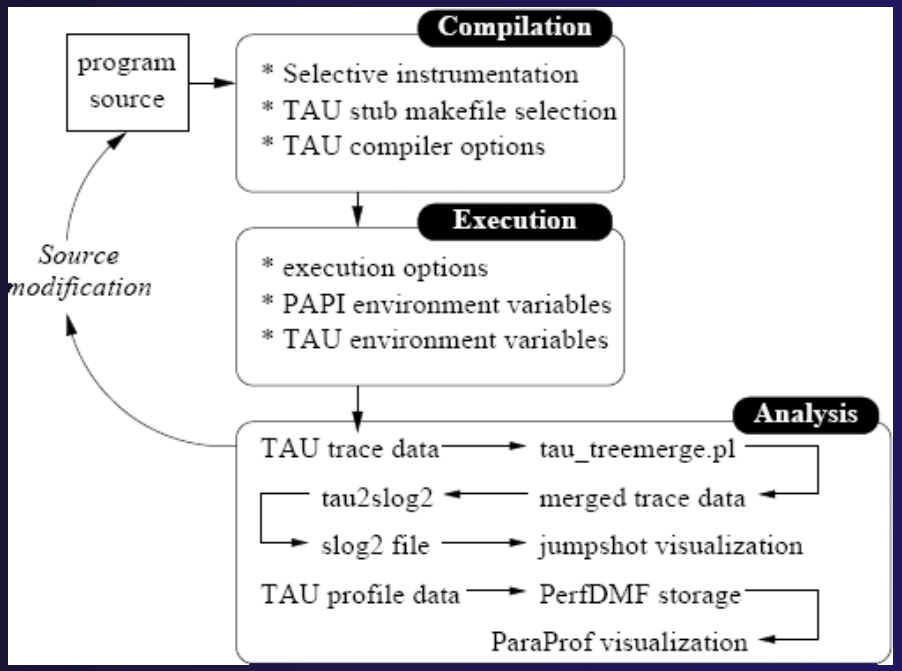






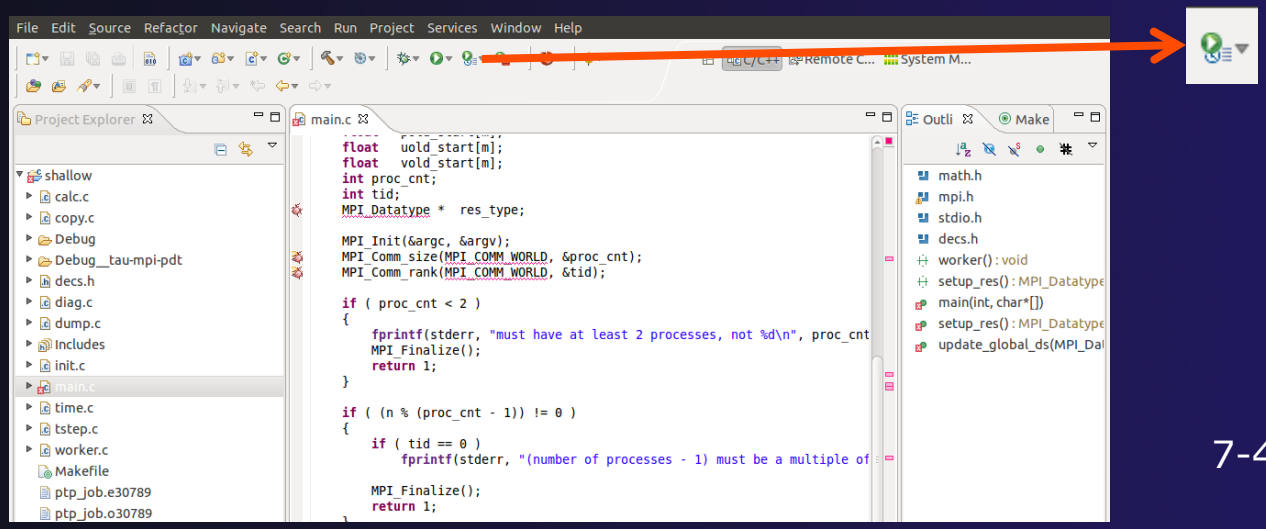
TAU Integration with PTP

- TAU: Tuning and Analysis Utilities
  - Performance data collection and analysis for HPC codes
  - Numerous features
  - Command line interface
- The TAU Workflow:
  - Instrumentation
  - Execution
  - Analysis



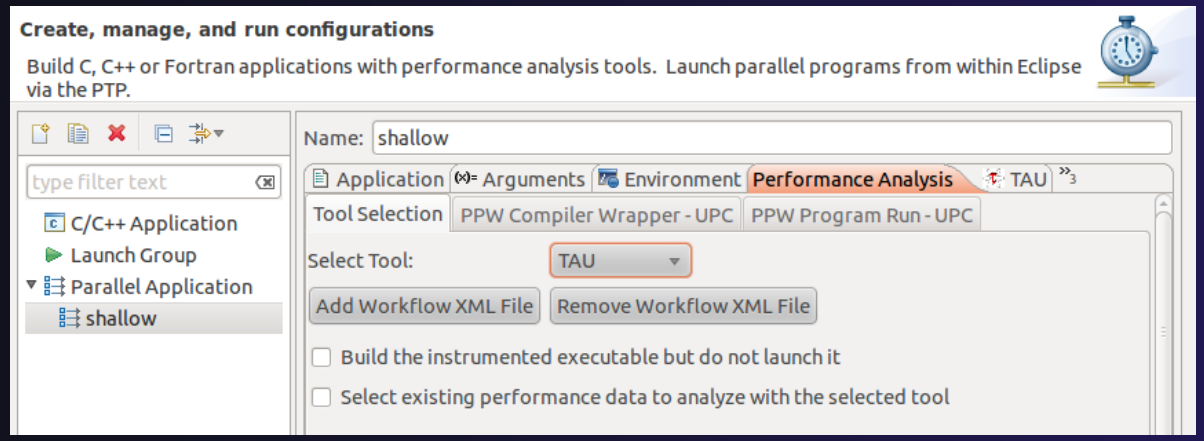
TAU/ETFw Hands-On

- Performance analysis/External tools use the same launch configurations and resource managers as debugging/launching
- The relevant tools must be in the $PATH on the remote machine
- Select the Profile button's "Profile Configurations…" option to begin:



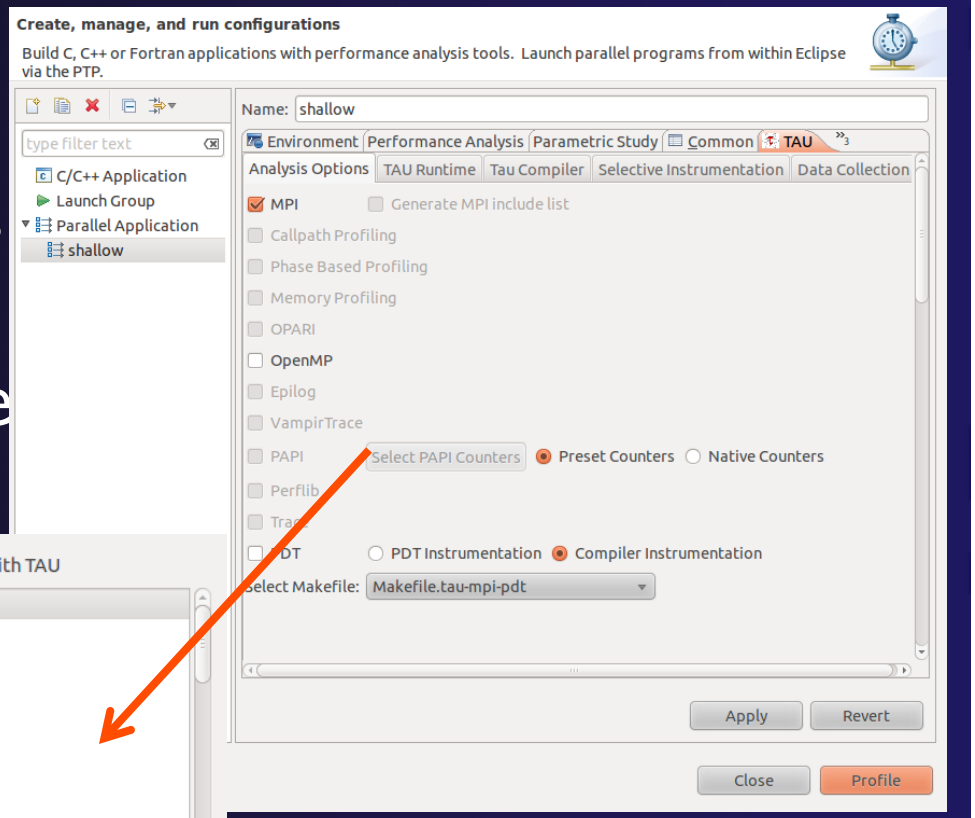
TAU/ETFw Hands-On (2)

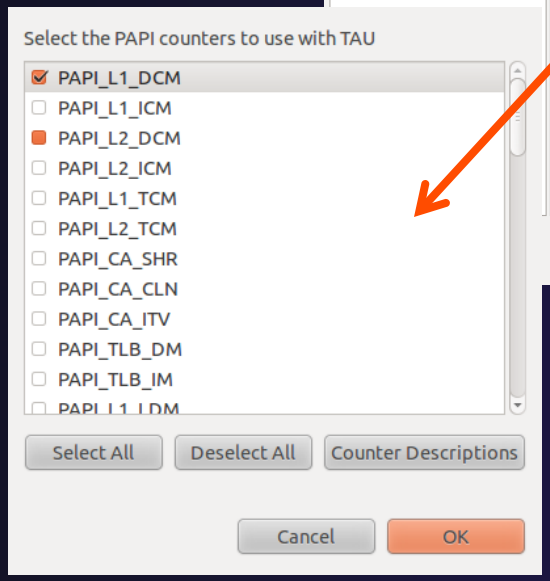
- Select an existing launch configuration or create a new one
- Select the Performance Analysis tab and choose the TAU tool set



TAU/ETFw Hands-On (3)

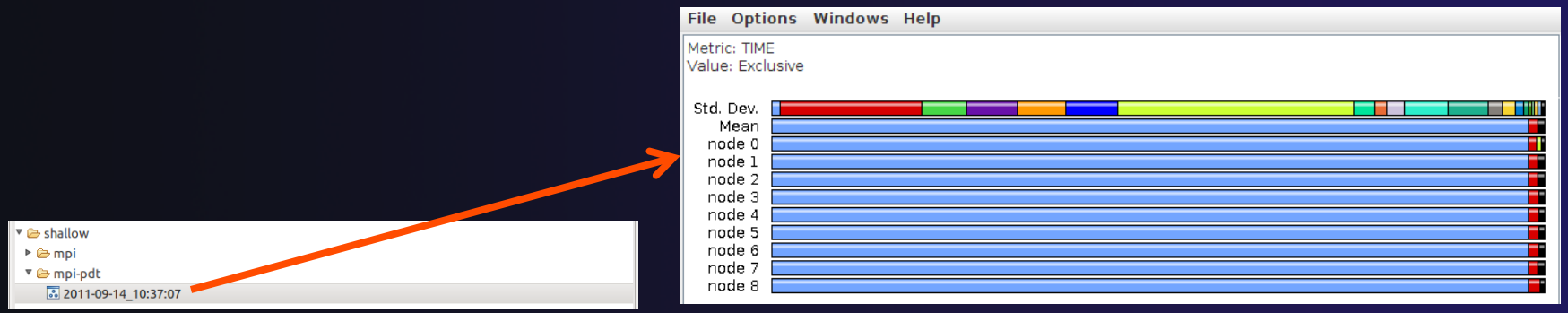
- Select the TAU tab and choose a Makefile with MPI, PDT and PAPI options
- Select PAPI counters
- Other options are available but not needed here
- Hit 'Profile'





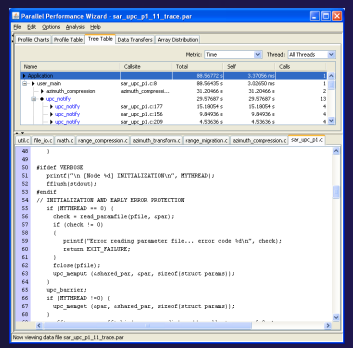
TAU/ETFW Hands-On (4)

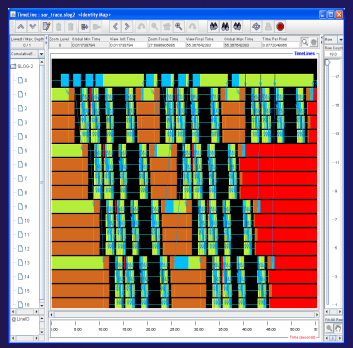
- The application will rebuild and launch.
- Performance data will appear in the Performance Data Management view
- Double click the new entry to view in ParaProf
- Right click on a function bar and select **Show Source Code** for source callback to Eclipse

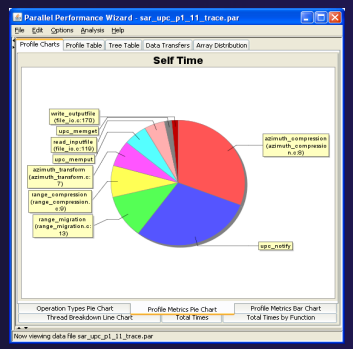


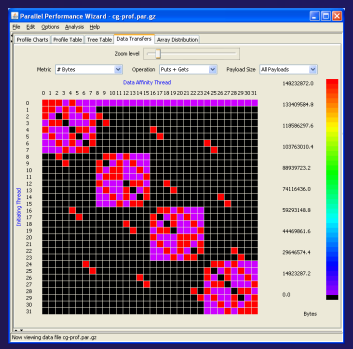
Parallel Performance Wizard (PPW)

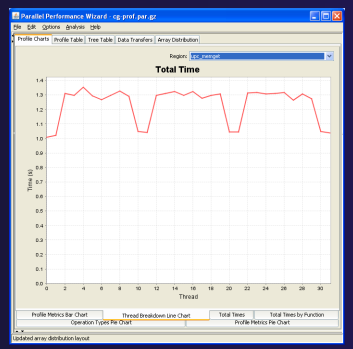
- Full-featured performance tool for PGAS programming models
  - Currently supports UPC, SHMEM, and MPI
  - Extensible to support other models
  - PGAS support by way of Global Address Space Performance (GASP) interface (http://gasp.hcs.ufl.edu)
- PPW features:
  - Easy-to-use scripts for backend data collection
  - User-friendly GUI with familiar visualizations
  - Advanced automatic analysis support
- More information and free download: http://ppw.hcs.ufl.edu

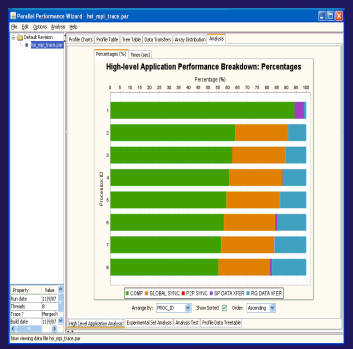






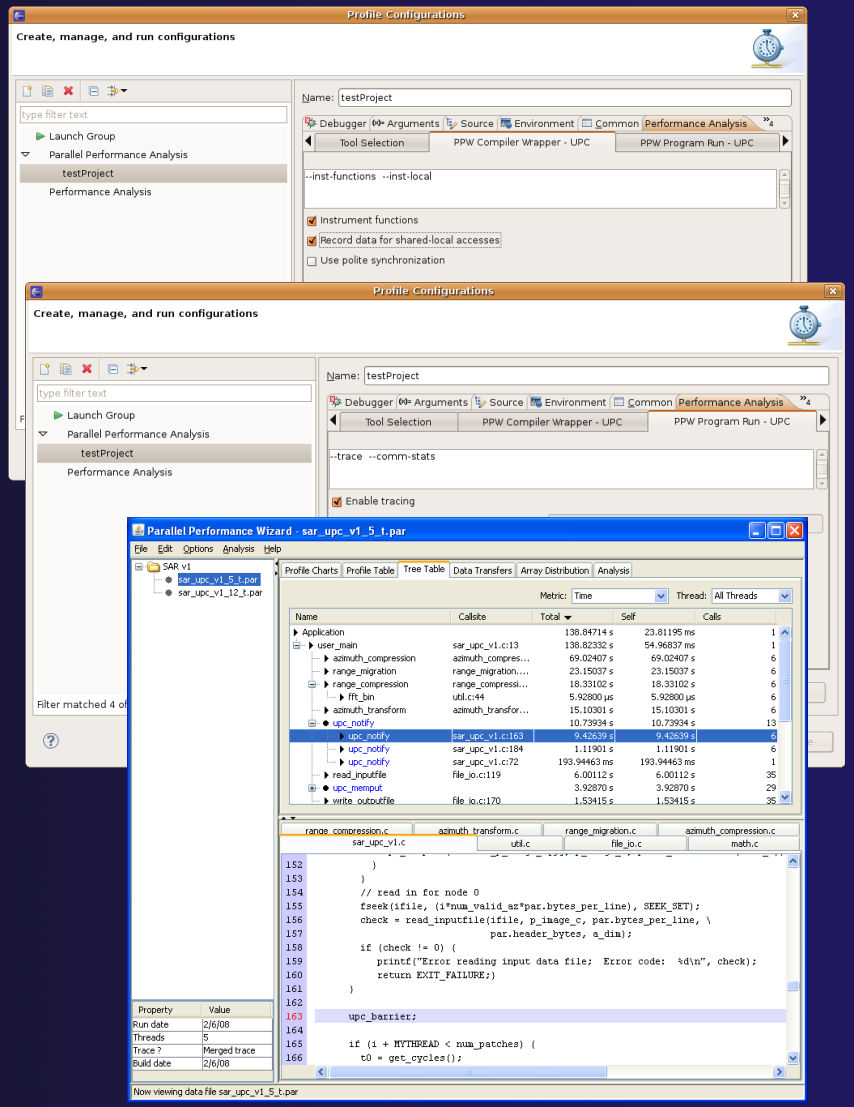






PPW Integration via ETFw

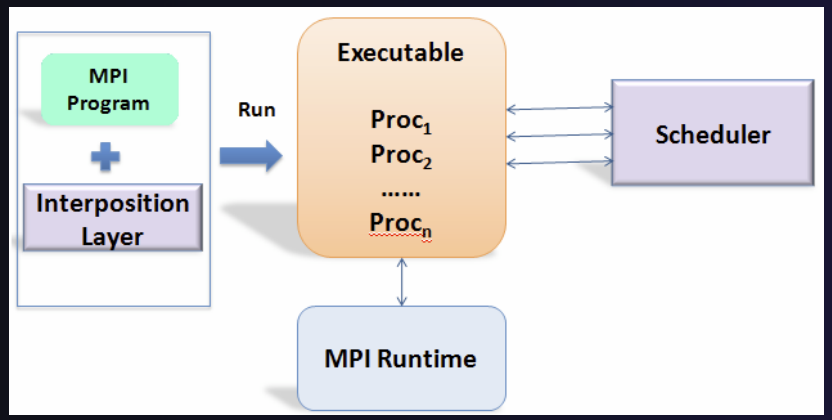
- We implement the ETFw to make PPW's capabilities available within Eclipse
  - Compile with instrumentation, parallel launch with PPW
  - Generates performance data file in workspace, PPW GUI launched
- PPW is often used for UPC application analysis
  - ETFw extended to support UPC
  - Many UPC features in PTP
- For more information:
  - http://ppw.hcs.ufl.edu
  - ppw@hcs.ufl.edu



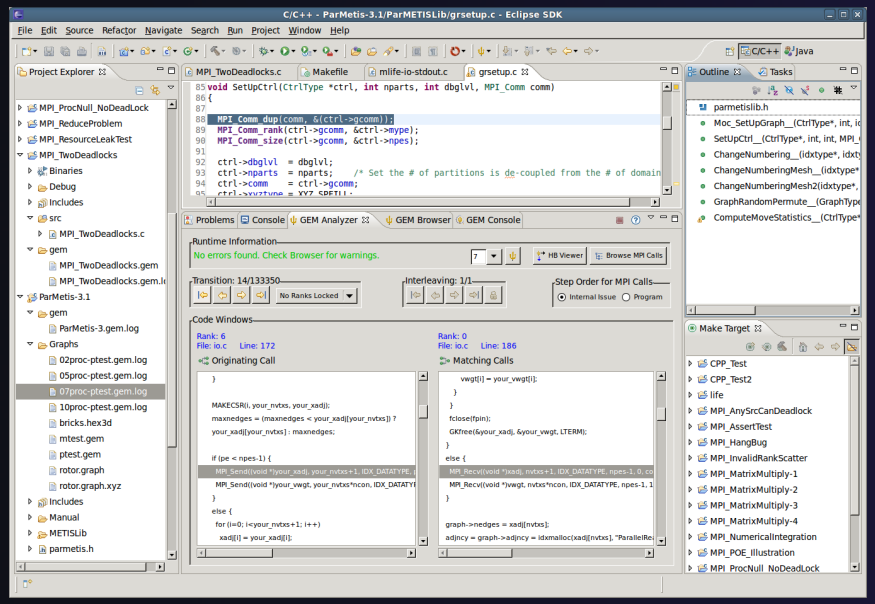
GEM
Graphical Explorer of MPI Programs

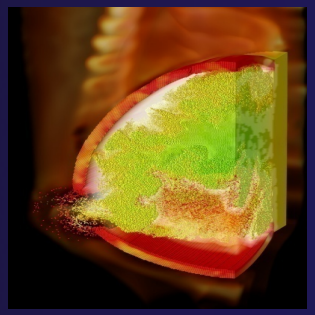
- Contributed to PTP by University of Utah in 2009
  - Available with PTP since v3.0
- Dynamic verification for MPI C/C++ that detects:
  - Deadlocks
  - MPI object leaks
  - Functionally irrelevant barriers
  - Local assertion violations
- Offers rigorous coverage guarantees
  - Complete nondeterministic coverage for MPI
  - Communication / synchronization behaviors
  - Determines relevant interleavings, replaying as necessary

GEM - Overview



- Front-end for In-situ Partial Order (ISP), Developed at U. Utah
- Introduces "push-button" verification into the MPI development cycle for PTP
- Automatically instruments and runs user code, displaying post verification results
- Variety of views & tools to facilitate debugging and code understanding



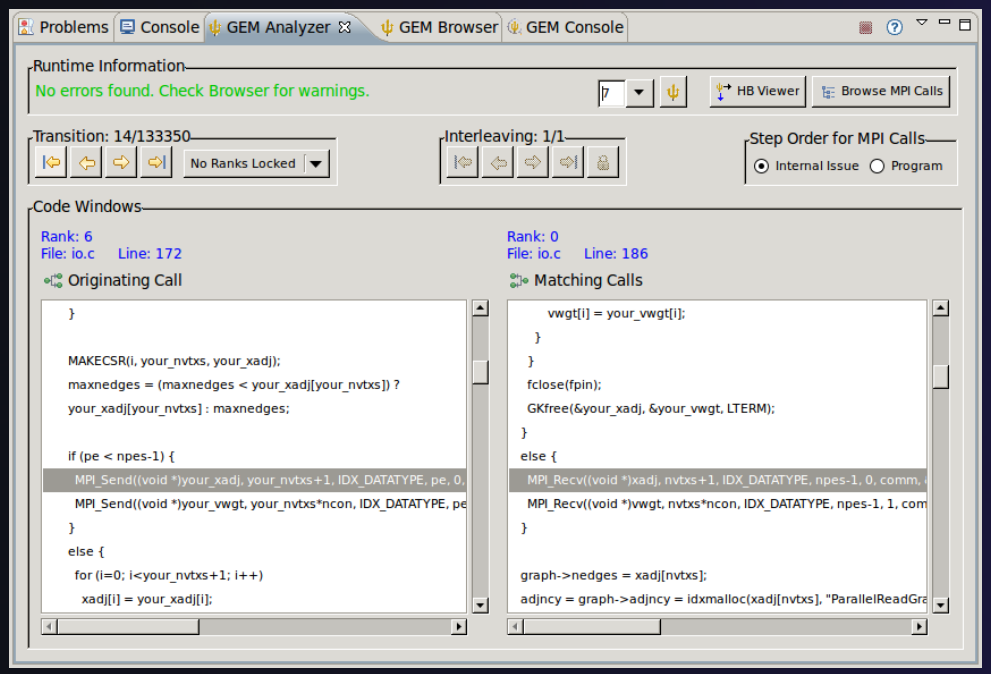


(Image courtesy of Steve Parker, U of Utah)

GEM – Views & Tools

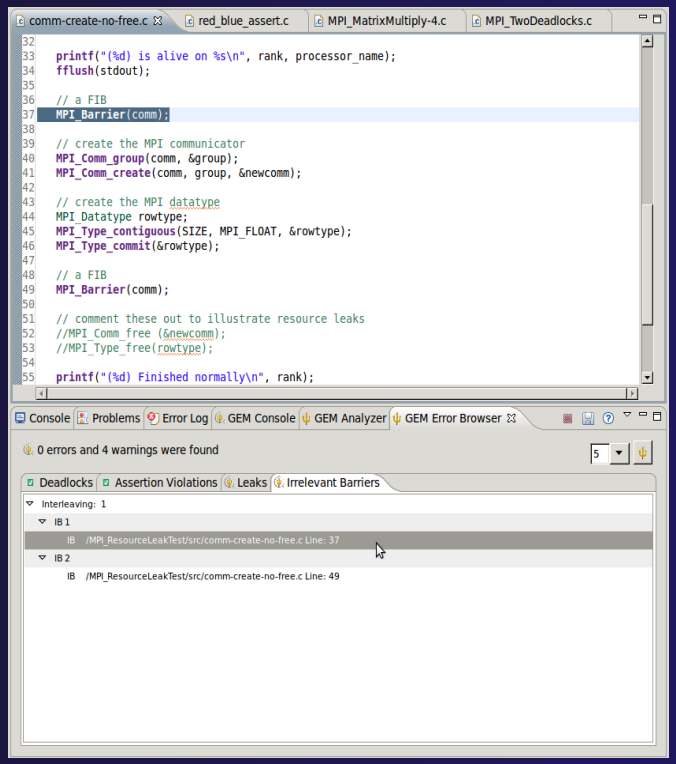
Analyzer View

Highlights bugs, and facilitates post-verification review / debugging



Browser View

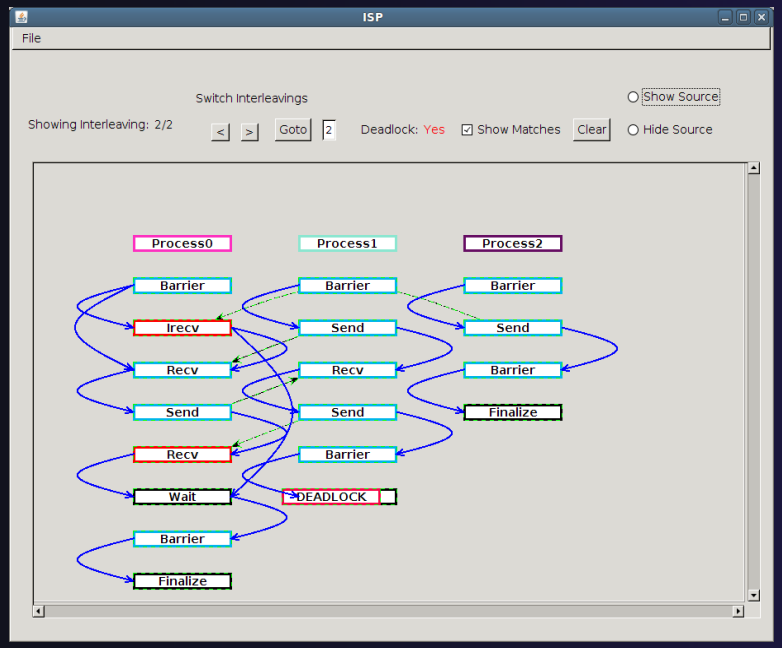
Groups & helps quickly localizes MPI problems. Maps errors to source code line in editor

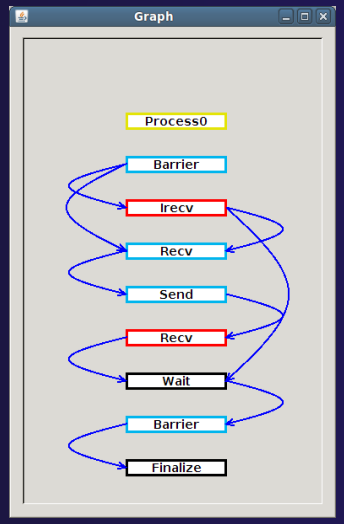


GEM – Views & Tools (cont.)

Happens-Before Viewer

Shows required orderings and communication matches
(currently an external tool)



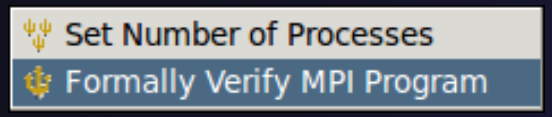


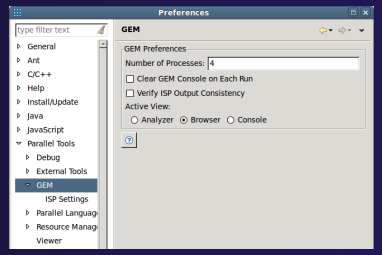
Using GEM – ISP Installation

- **<u>ISP itself must be installed prior to using GEM</u>**
  - Download ISP at http://www.cs.utah.edu/fv/ISP
- Make sure libtool, automake and autoconf are installed.
- Just untar isp-0.2.0.tar.gz into a tmp directory:
  - Configure and install
    - ./configure
    - make
    - make install
      - This installs binaries and necessary scripts

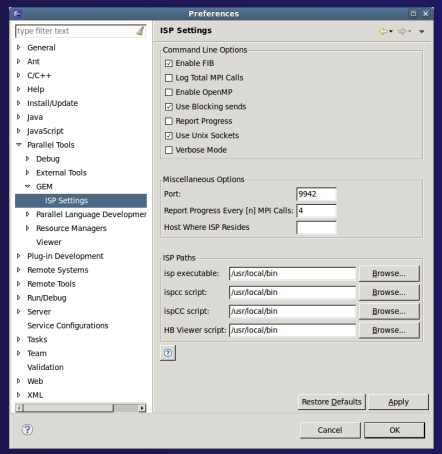
Using GEM

- Create local or remote MPI C/C++ project
  - Make sure your project builds correctly
  - Managed build and Makefile projects supported
- Set preferences via GEM Preference Pages
- From the trident icon or context menus user can:



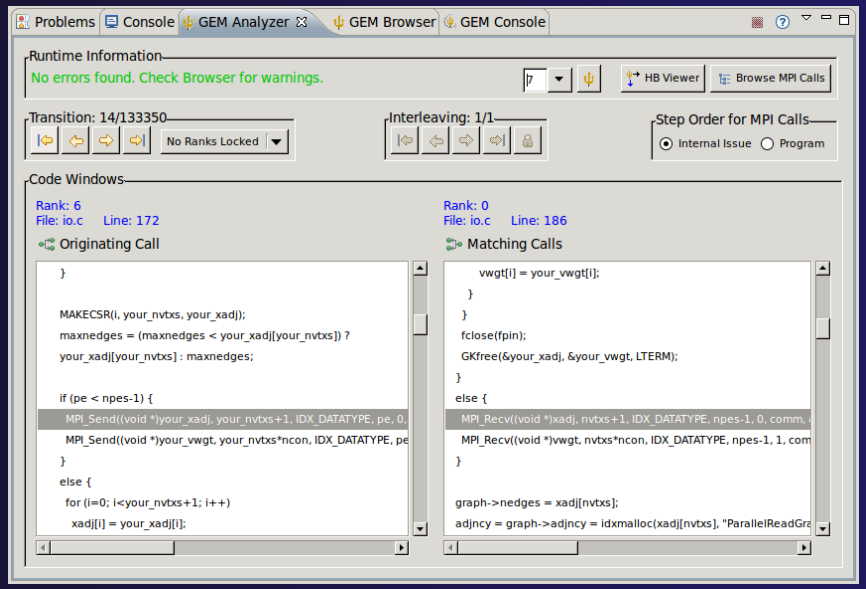


- Formally Verifying MPI Program
  - Launches verification engine ISP
  - Generates log file for post-verification analysis
  - Opens relevant GEM views



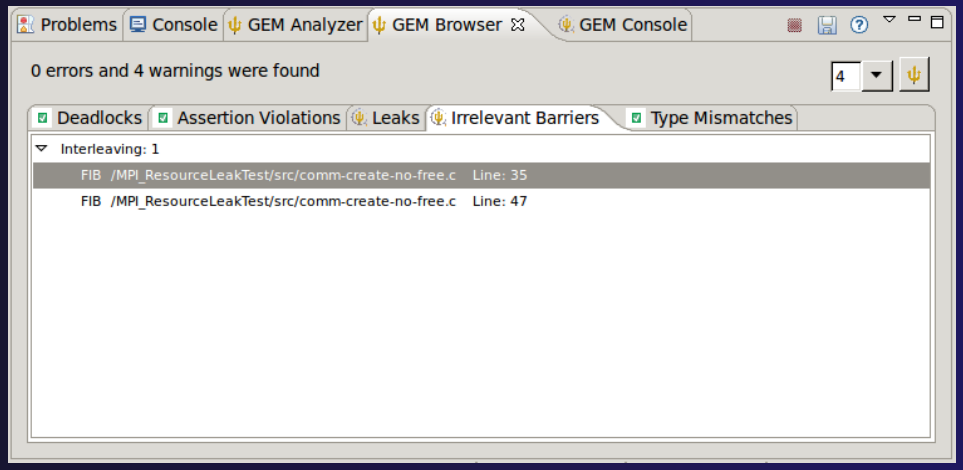
GEM Analyzer View

- Reports program errors, and runtime statistics
- Debug-style source code stepping of interleavings
  - Point-to-point / Collective Operation matches
  - Internal Issue Order / Program Order views
  - Rank Lock feature – focus on a particular process
- Also controls:
  - Call Browser
  - Happens Before Viewer launch
  - Re-launching of GEM



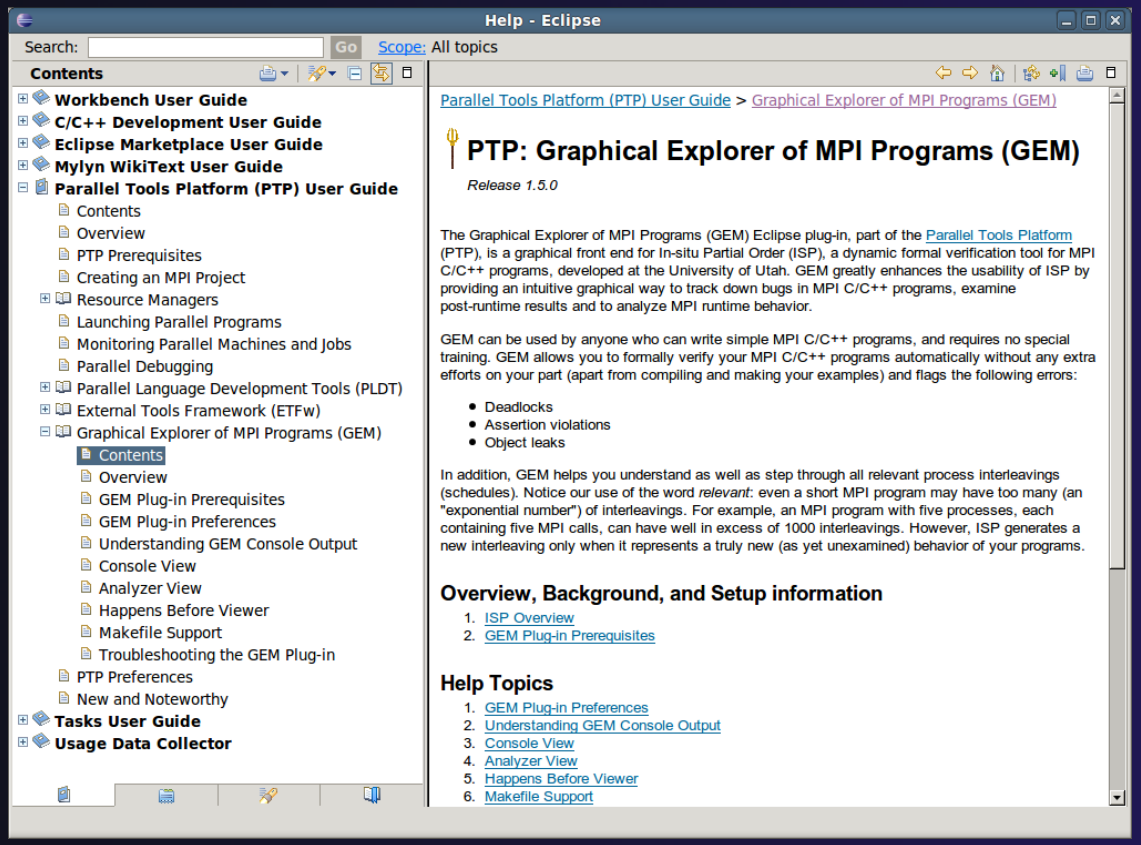
GEM Browser View

- Tabbed browsing for each type of MPI error/warning
- Each error/warning mapped to offending line of source code in Eclipse editor
- One click to visit the Eclipse editor, to examine:
  - Calls involved in deadlock
  - Irrelevant barriers
  - MPI Object Leaks sites
  - MPI type mismatches
  - Local Assertion Violations



GEM – Help Plugin

Extensive how-to sections, graphical aids and trouble shooting section

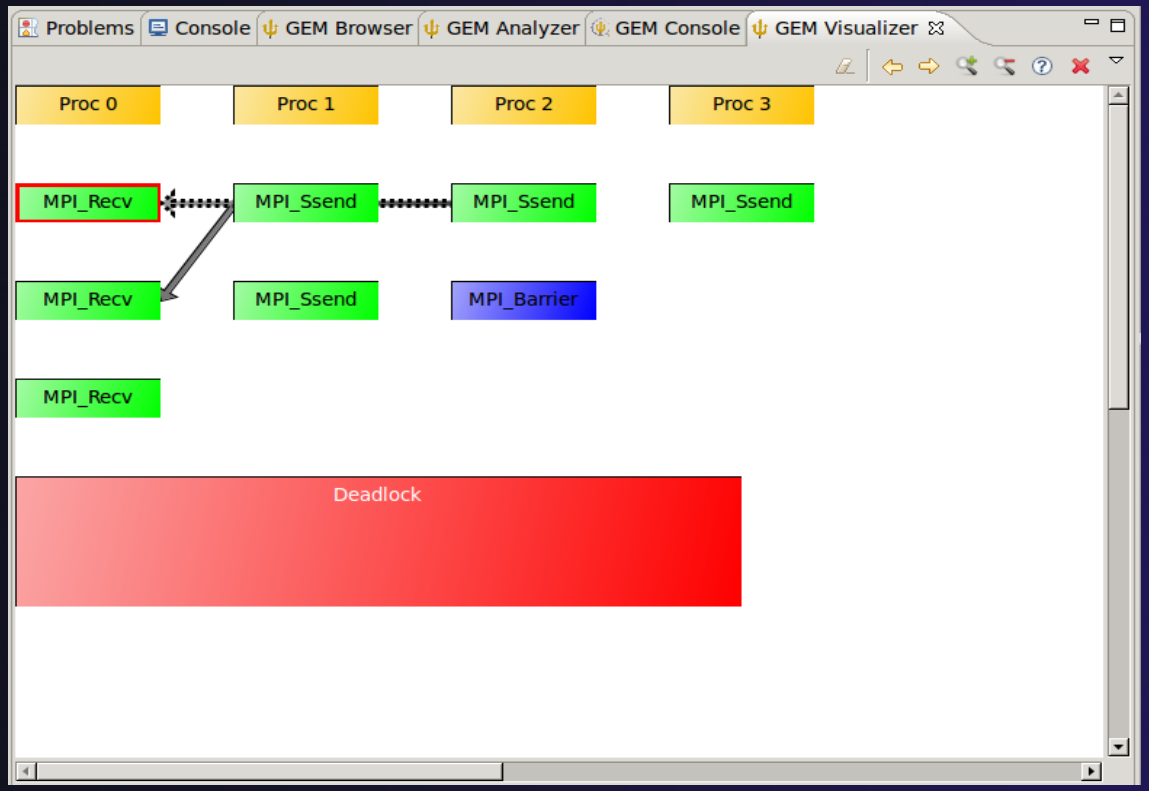


GEM/ISP Success Stories

- Umpire Tests
  - http://www.cs.utah.edu/fv/ISP-Tests
  - Documents bugs missed by tests, caught by ISP
- MADRE (EuroPVM/MPI 2007)
  - Previously documented deadlock detected
- N-Body Simulation Code
  - Previously unknown resource leak caught during EuroPVM/MPI 2009 tutorial !
- Large Case Studies
  - ParMETIS, MPI-BLAST, IRS (Sequoia Benchmark), and a few SPEC-MPI benchmarks could be handled
- Full Tutorial including LiveDVD ISO available
  - Visit http://www.cs.utah.edu/fv/GEM

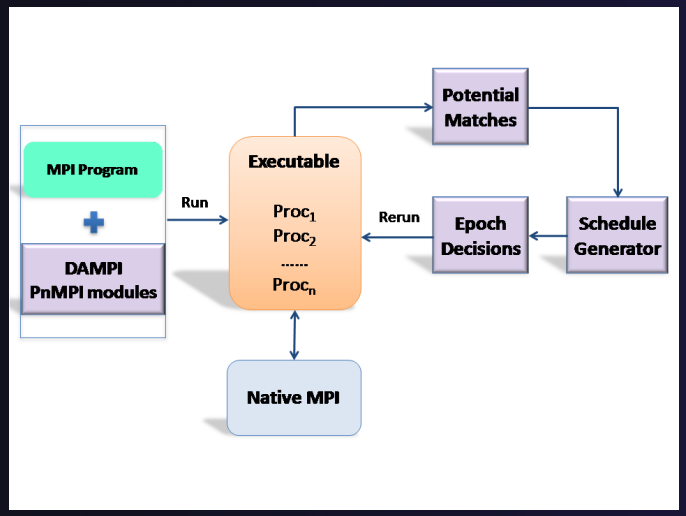
GEM Future Plans

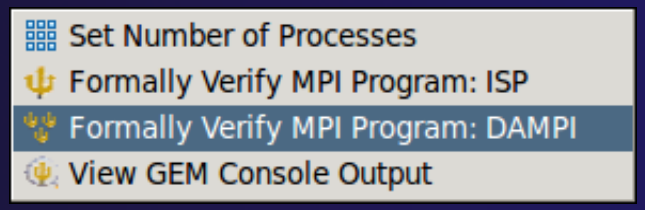
- Incorporation of HB Viewer into GEM as a new view
- Add Pthread support to visualize Pthread calls made from within MPI space



GEM Future Plans

- GEM will serve as a front-end for other tools
- Integration of Distributed Analyzer of MPI Programs (DAMPI), developed at University of Utah
  - ISP scales to 10s of processes
  - DAMPI scales to 1000s of processes (C/C++/Fortran)
  - Decentralized scheduler uses Lamport Clocks





Use **ISP** at small scale, then launch **DAMPI** at scale on a cluster

Performance Tools: Summary

- Performance Tools integrated with PTP
  - Performance Tools integrated with PTP help tune parallel applications
  - External Tools Framework (ETFw) eases integration of existing (command-line, etc.) tools
    - TAU Performance Tuning uses ETFw
    - PPW (Parallel Perf. Wizard) uses ETFw for UPC analysis
  - MPI Analysis: GEM
- A diversity of contributors too!
  - We welcome other contributions. Let us help!

Module 6: Other Tools and Wrap-up

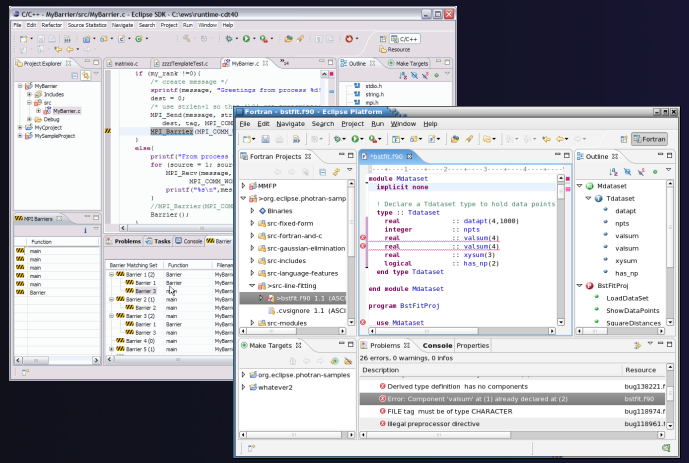
- Objective
  - How to find more information on PTP
  - Learn about other tools related to PTP
  - See PTP upcoming features
- Contents
  - Links to other tools, including performance tools
  - Planned features for new versions of PTP
  - Additional documentation
  - How to get involved

NCSA HPC Workbench

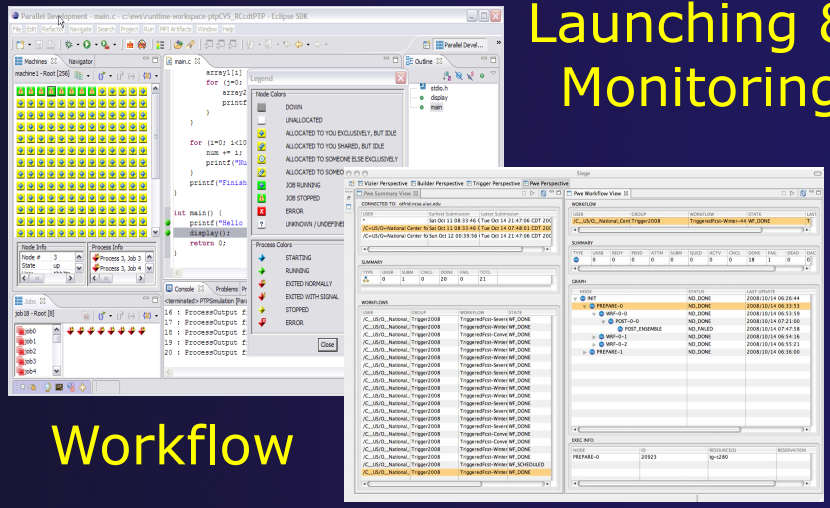
- Tools for NCSA Blue Waters
  - http://www.ncsa.illinois.edu/BlueWaters/
  - Sustained Petaflop system
- Based on Eclipse and PTP
- Includes some related tools
  - Performance tools
  - Workflow tools (https://wiki.ncsa.uiuc.edu/display/MRDPUB/MRD+Public+Space+Home+Page)
- Part of the enhanced computational environment described at:
  http://www.ncsa.illinois.edu/BlueWaters/ece.html

NCSA HPC Workbench

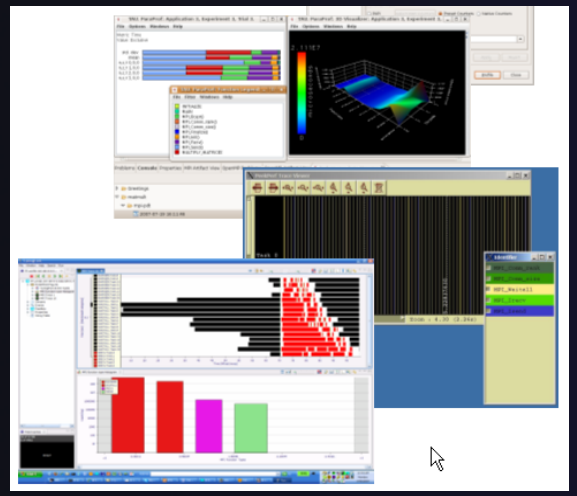
Coding & Analysis (C/C++, Fortran)



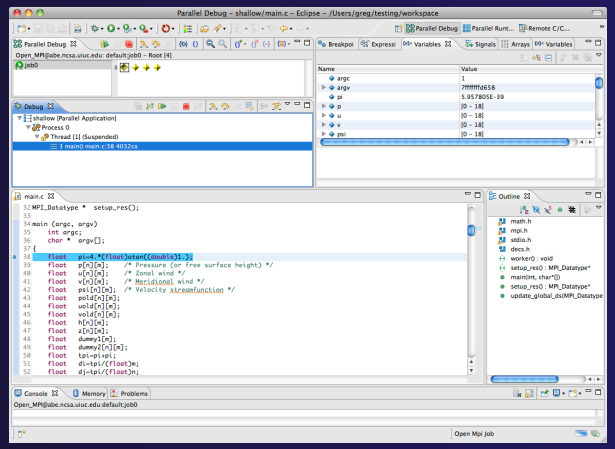
PTP Launching & Monitoring



Workflow



Performance Tuning



Parallel Debugger

Planned PTP Future Work

- Scalability improvements
  - UI to support 1M processes
  - Optimized communication protocol
  - Very large application support
- Resource Managers
  - More implementations of configurable resource managers
- Synchronized project improvements
  - Conversion wizard
  - Resolving merge conflicts
- Enhancements to the debugger
  - Stability enhancements
  - Transition to Scalable Communication Infrastructure (SCI)

Useful Eclipse Tools

- Linux Tools (autotools, valgrind, Oprofile, Gprof)
  - http://eclipse.org/linuxtools
- Python
  - http://pydev.org
- Ruby
  - http://www.aptana.com/products/radrails
- Perl
  - http://www.epic-ide.org
- Git
  - http://www.eclipse.org/egit
- VI bindings
  - Vrapper (open source) - http://vrapper.sourceforge.net
  - viPlugin (commercial) - http://www.viplugin.com

Online Information

- Information about PTP
  - Main web site for downloads, documentation, etc.
    - http://eclipse.org/ptp
  - Wiki for designs, planning, meetings, etc.
    - http://wiki.eclipse.org/PTP
  - Articles and other documents
    - http://wiki.eclipse.org/PTP/articles
- Information about Photran
  - Main web site for downloads, documentation, etc.
    - http://eclipse.org/photran
  - User's manuals
    - http://wiki.eclipse.org/PTP/photran/documentation

Mailing Lists

- PTP Mailing lists
  - Major announcements (new releases, etc.) - low volume
    - http://dev.eclipse.org/mailman/listinfo/ptp-announce
  - User discussion and queries - medium volume
    - http://dev.eclipse.org/mailman/listinfo/ptp-user
  - Developer discussions - high volume
    - http://dev.eclipse.org/mailman/listinfo/ptp-dev
- Photran Mailing lists
  - User discussion and queries
    - http://dev.eclipse.org/mailman/listinfo/photran
  - Developer discussions –
    - http://dev.eclipse.org/mailman/listinfo/photran-dev

Getting Involved

- See http://eclipse.org/ptp
- Read the developer documentation on the wiki
- Join the mailing lists
- Attend the monthly developer meetings
  - Conf Call Monthly: Second Tuesday, 1:00 pm ET
  - Details on the PTP wiki
- Attend the monthly user meetings
  - Teleconference Monthly
  - Each 4th Wednesday, 2:00 pm ET
  - Details on the PTP wiki

PTP will only succeed with your participation!

PTP Tutorial Feedback

- Please complete feedback form
- Your feedback is valuable!

Thanks for attending
We hope you found it useful