Parallel Debugging

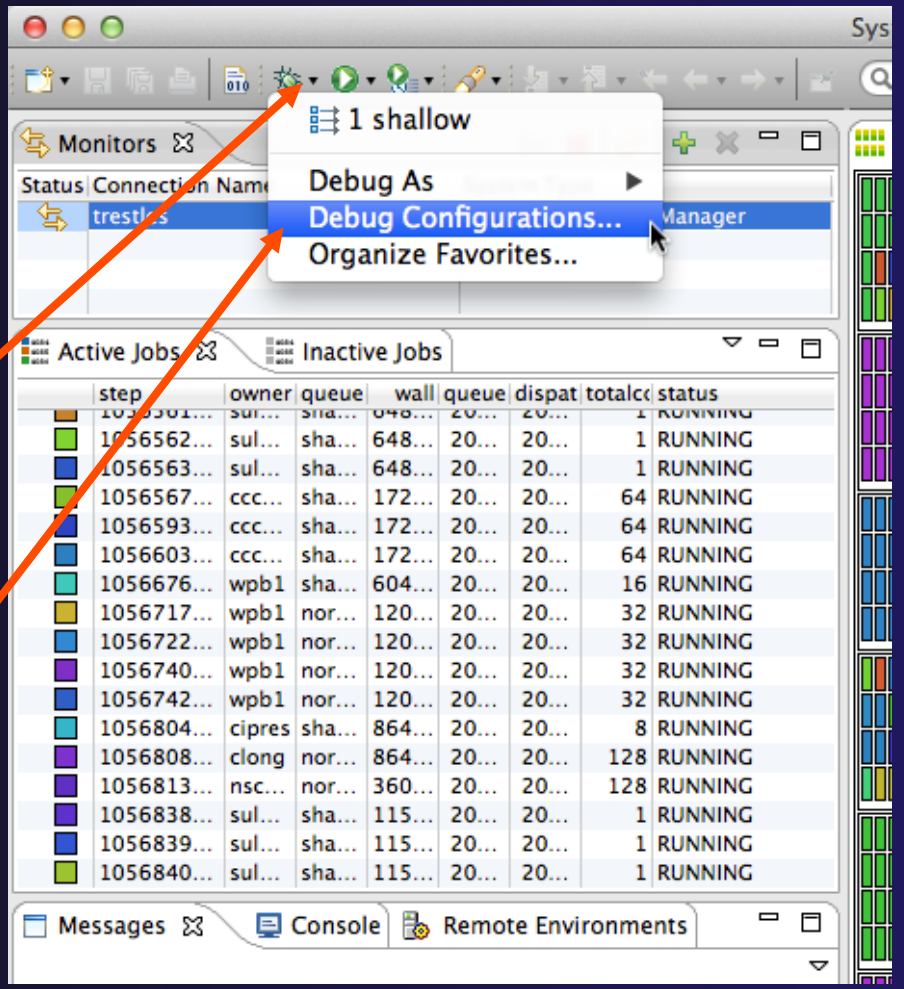
- Objective
  - Learn the basics of debugging parallel programs
- Contents
  - Launching a debug session
  - The Parallel Debug Perspective
  - Controlling sets of processes
  - Controlling individual processes
  - Parallel Breakpoints
  - Terminating processes

Debugging Setup

- Debugging requires interactive access to the application
- Can use any of the -*Interactive* target configurations
  - *Torque-Generic-Interactive*
  - *PBS-Generic-Interactive*
  - *OpenMPI-Generic-Interactive*

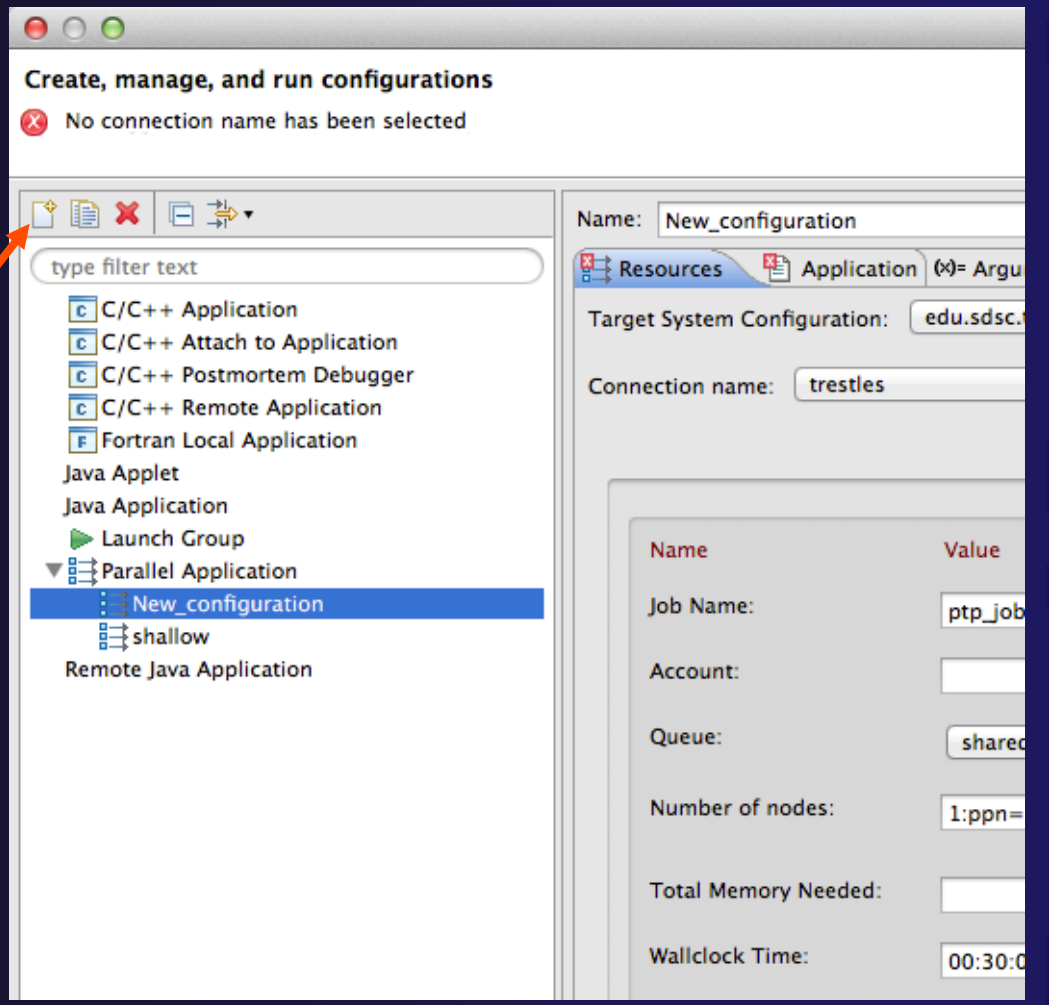
Create a Debug Configuration

- A debug configuration is essentially the same as a run configuration (like we used in the *Running an Application* module)
- It is possible to re-use an 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



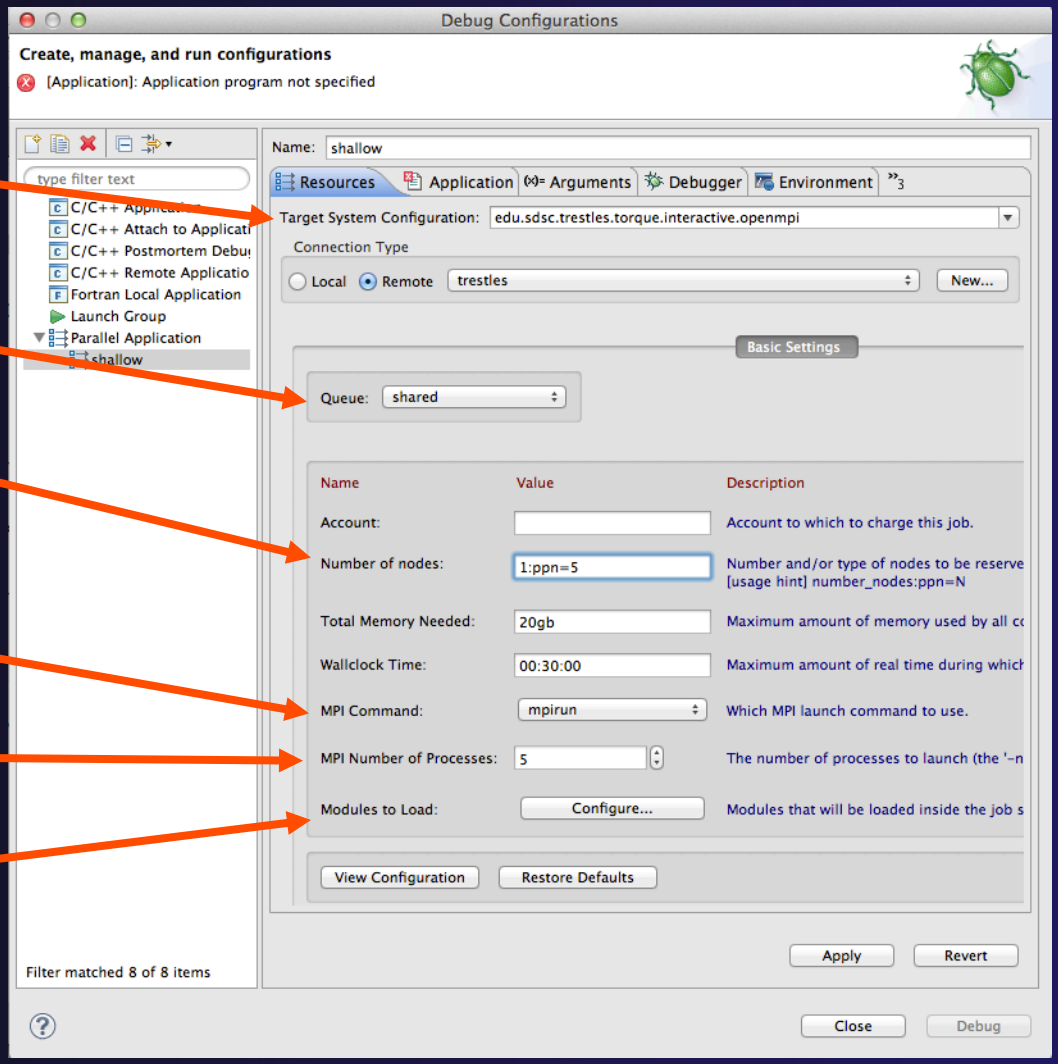
Create a New Configuration

- Select the existing configuration
- Click on the **new** button to create a new configuration



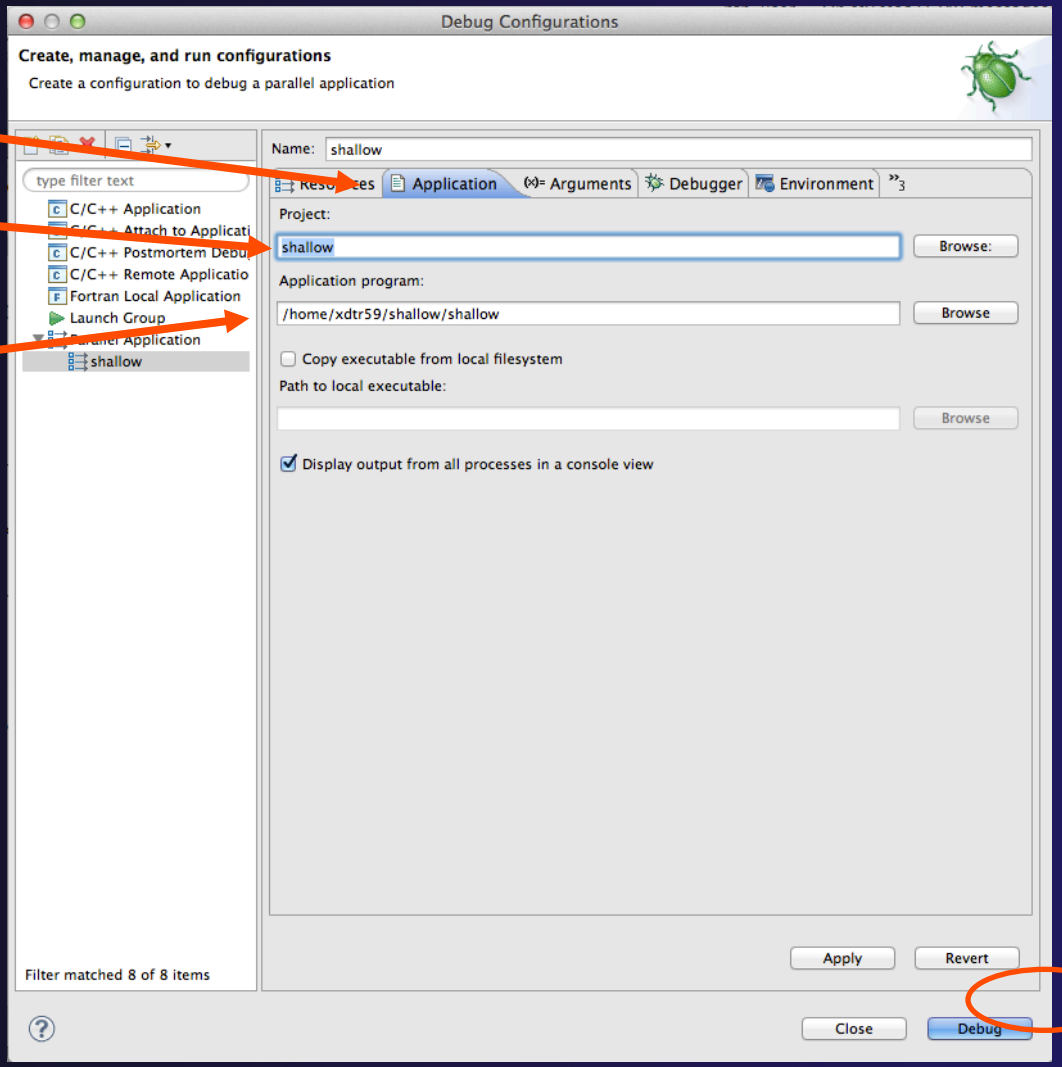
Configure the Resources Tab

- Select the new target system configuration
- Choose the queue
- Make sure number of nodes is correct
- Make sure the **mpirun** command is selected
- Select the number of processes (in this case use 5)
- Configure modules if required



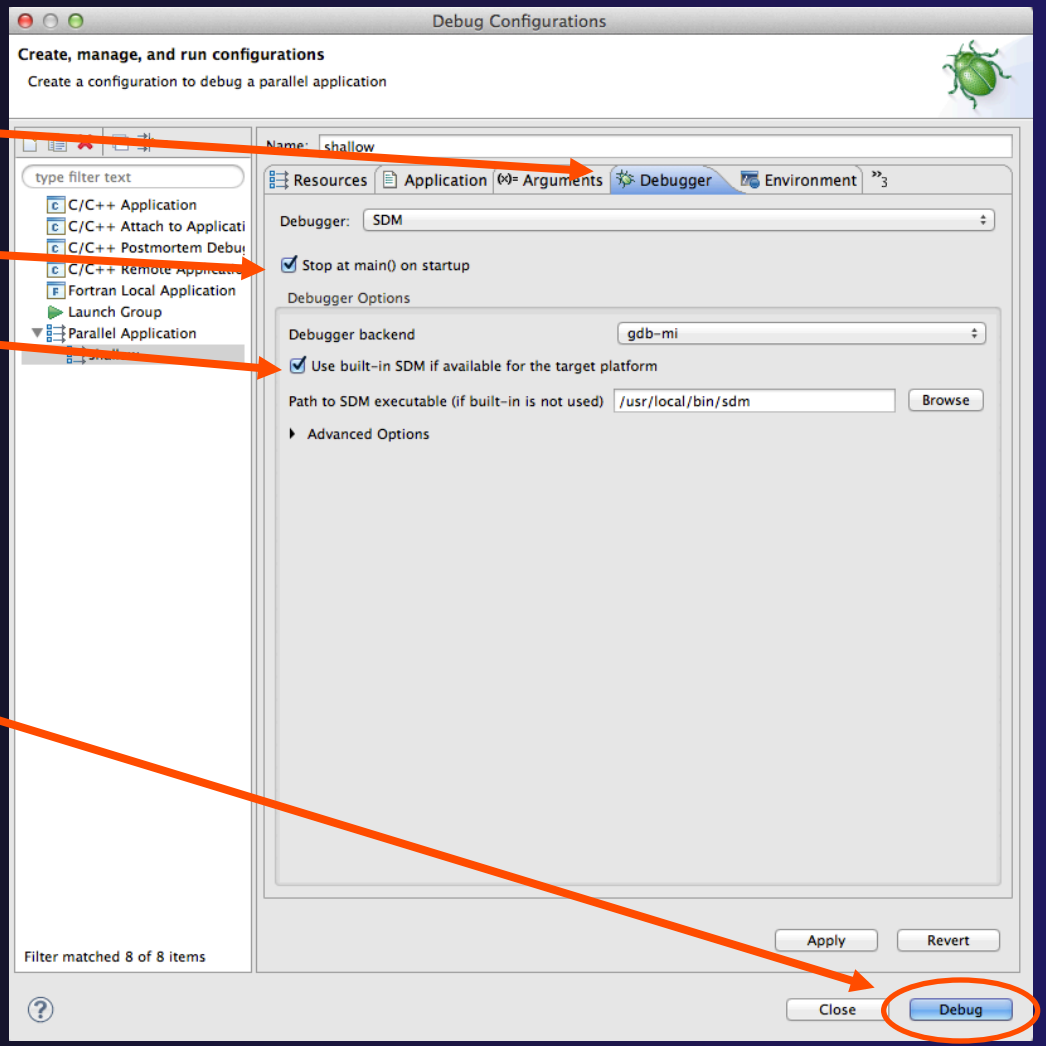
Configure the Application Tab (Optional)

- Select **Application** tab
- Make sure the **Project** is correct
- Select the application executable



Configure the Debug Tab (Optional)

- Select **Debugger** tab
- Debugger will stop at main() by default
- By default the built-in SDM will be used
  - Override this if you want to use your own SDM
- Click on **Debug** to launch the program

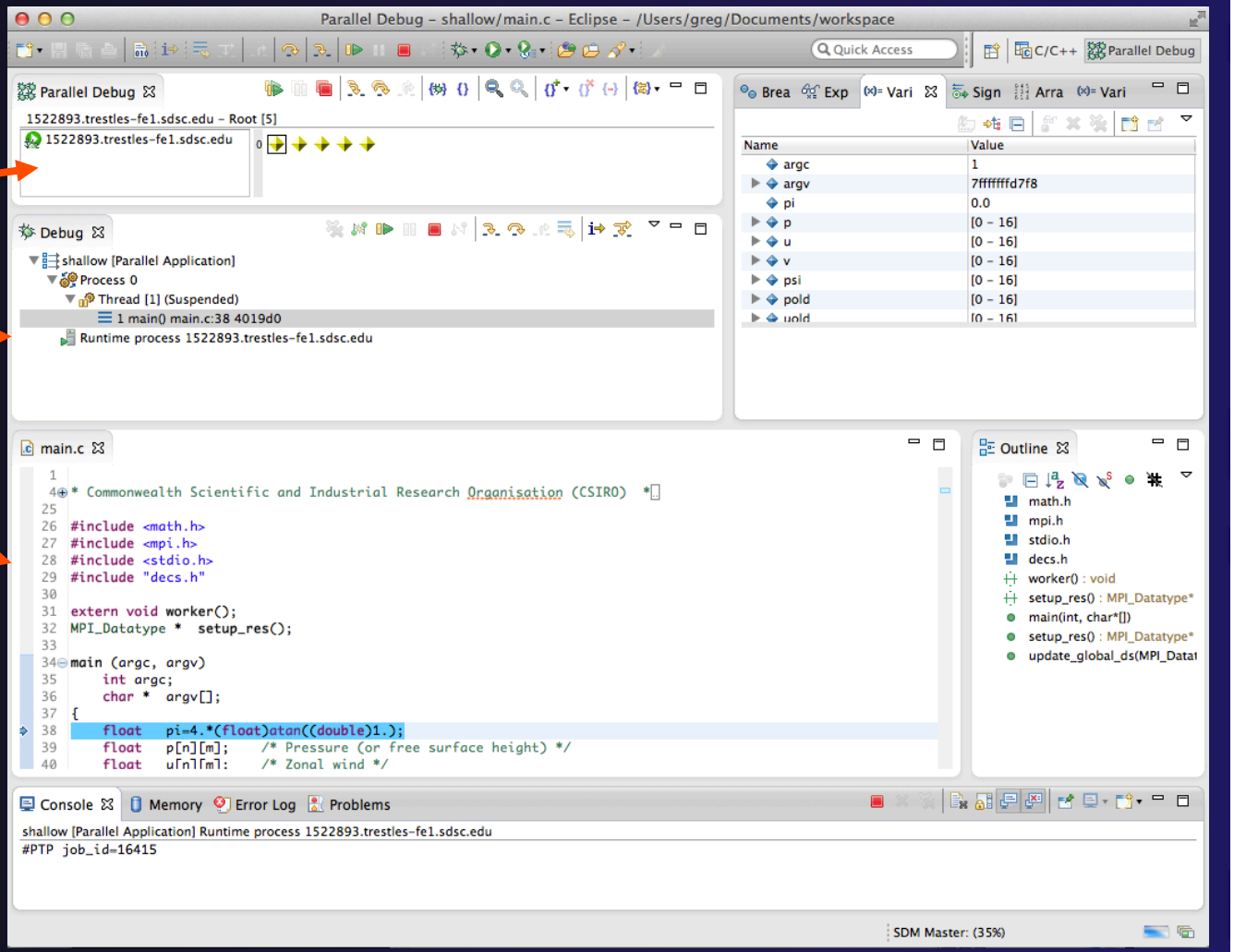


Exercise

1. Open the debug configuration dialog
2. Create a new configuration
3. Select the *edu.sdsc.trestles.torque.interactive.openmpi* target configuration
4. Configure the **Debug** tab
   - Queue: *shared*
   - Number of nodes: *1:ppn=5*
   - MPI Command: *mpirun*
   - MPI Number of Processes: *5*
5. Launch the debugger

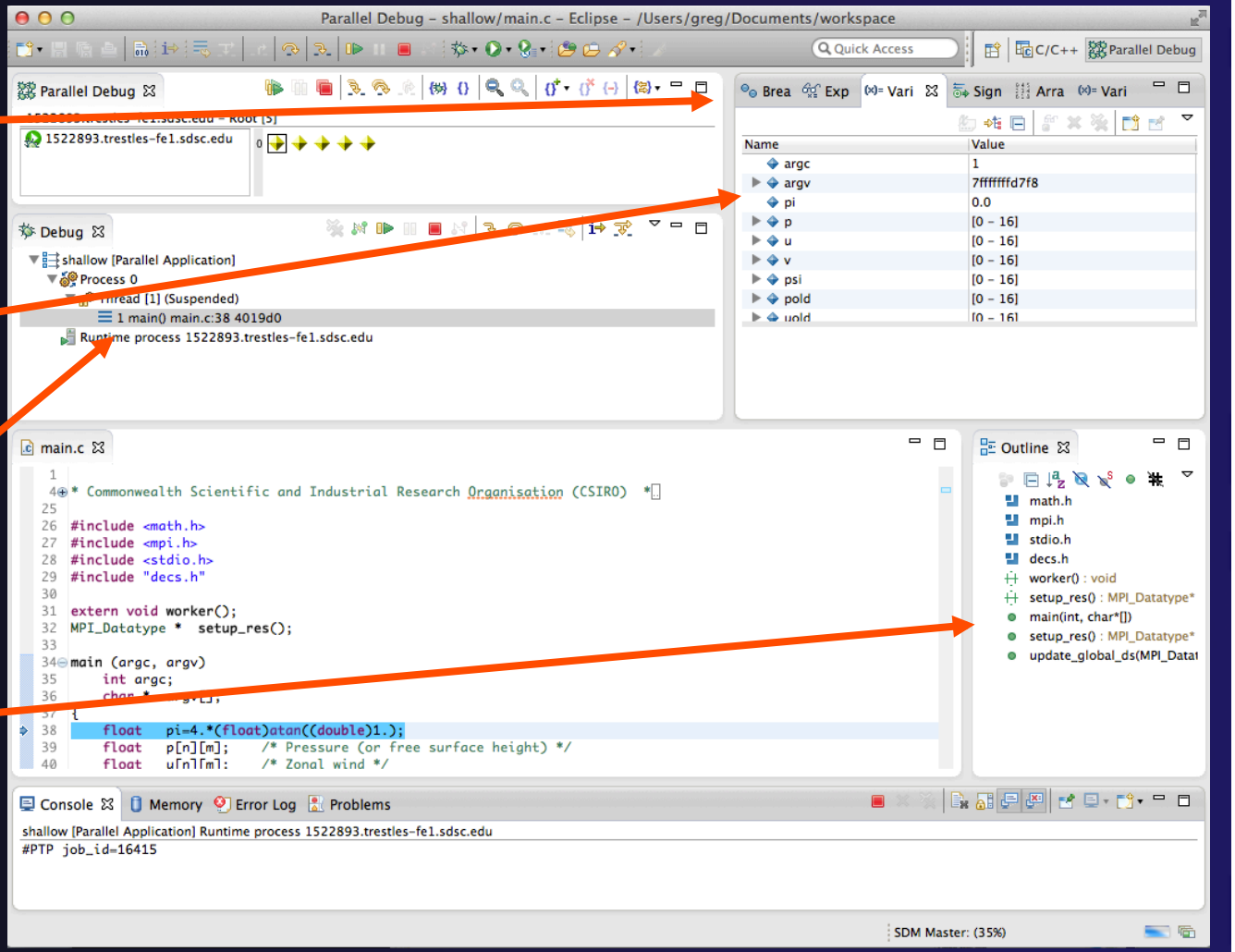
The Parallel Debug Perspective (1)

- **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



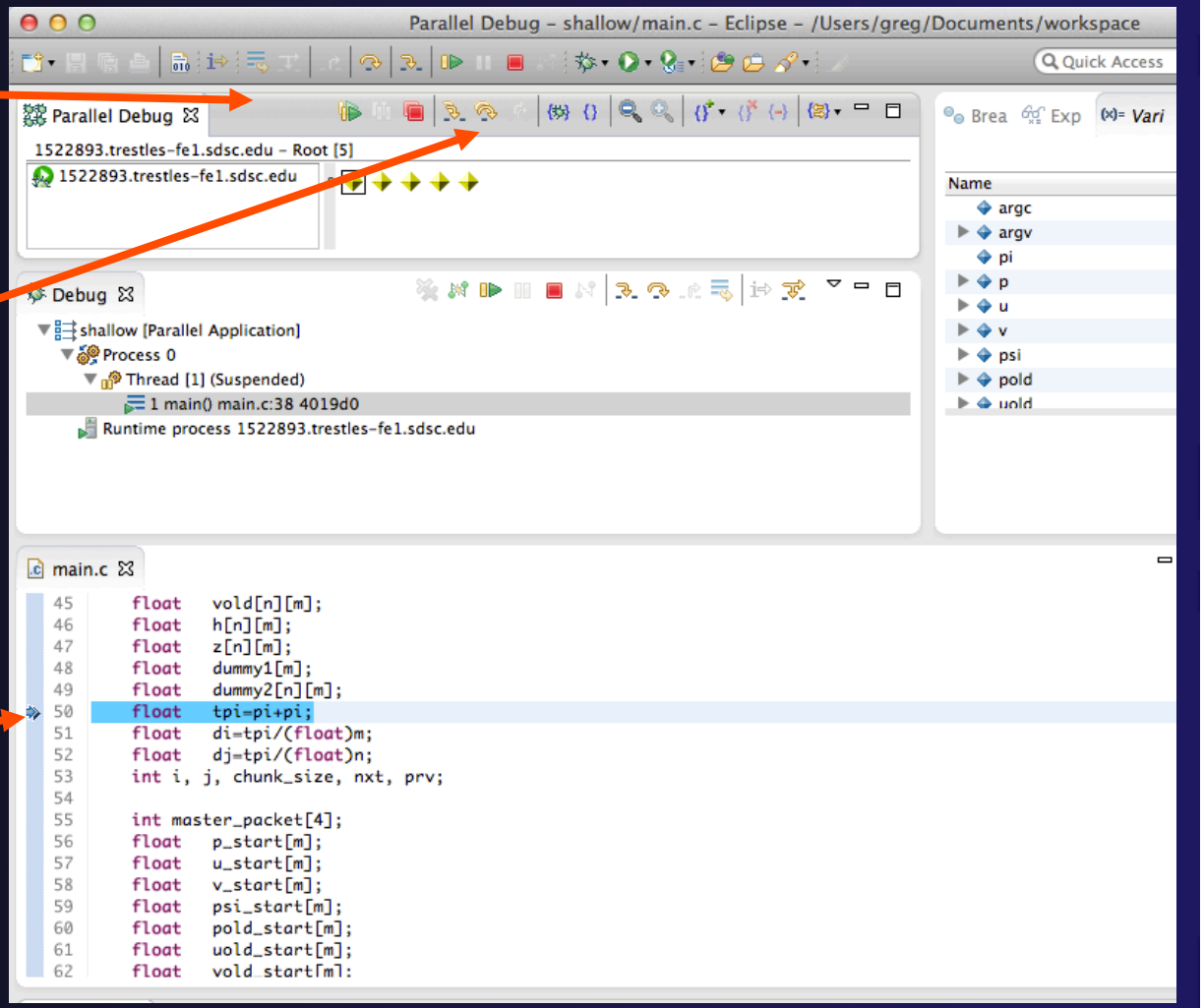
The Parallel Debug Perspective (2)

- **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



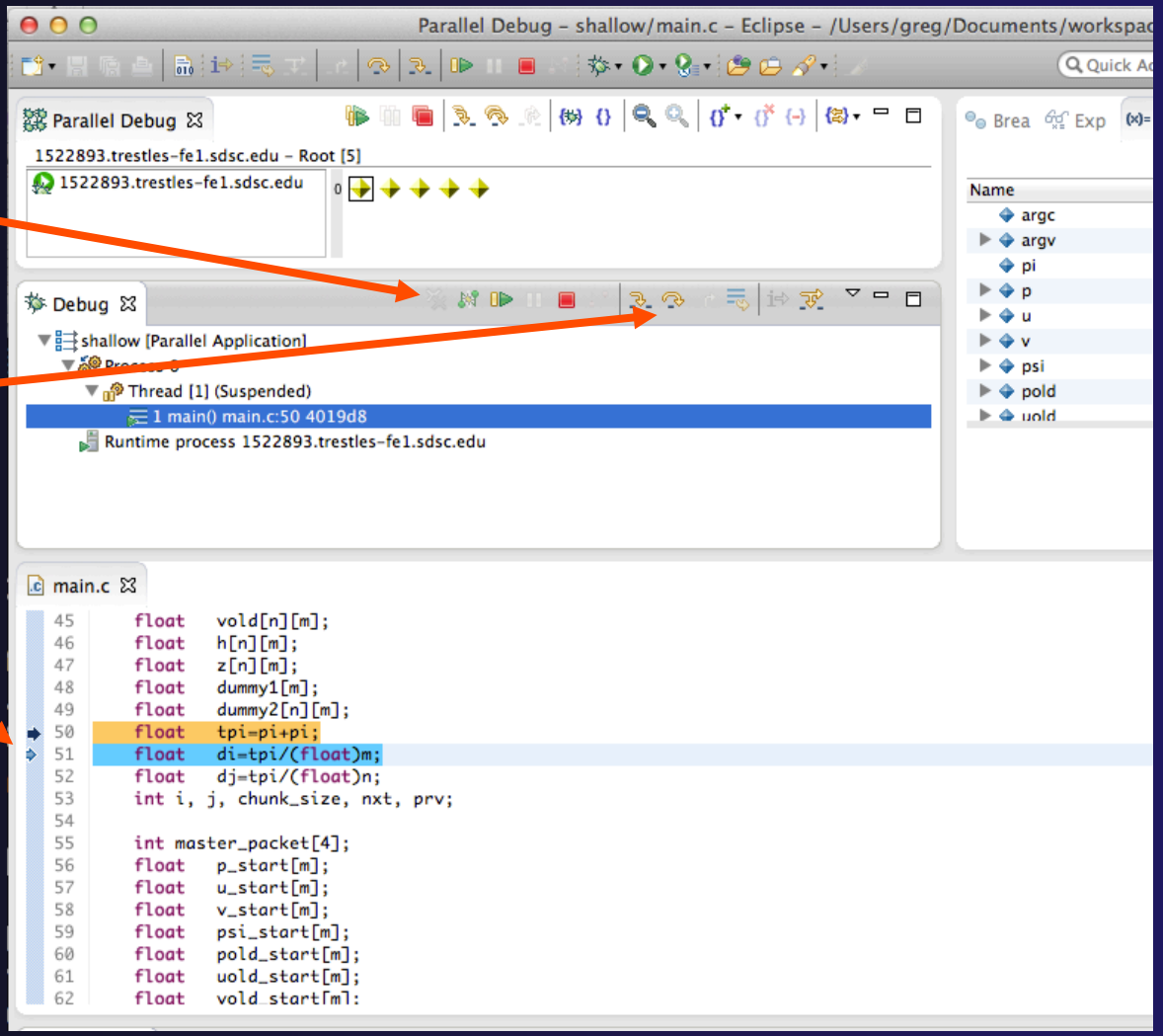
Stepping All Processes

- The buttons in the **Parallel Debug View** control groups of processes
- The **Step Over** button will step all processes one line
- The process icons will change to green (running), then back to yellow (suspended)
- The current line marker will move to the next source line



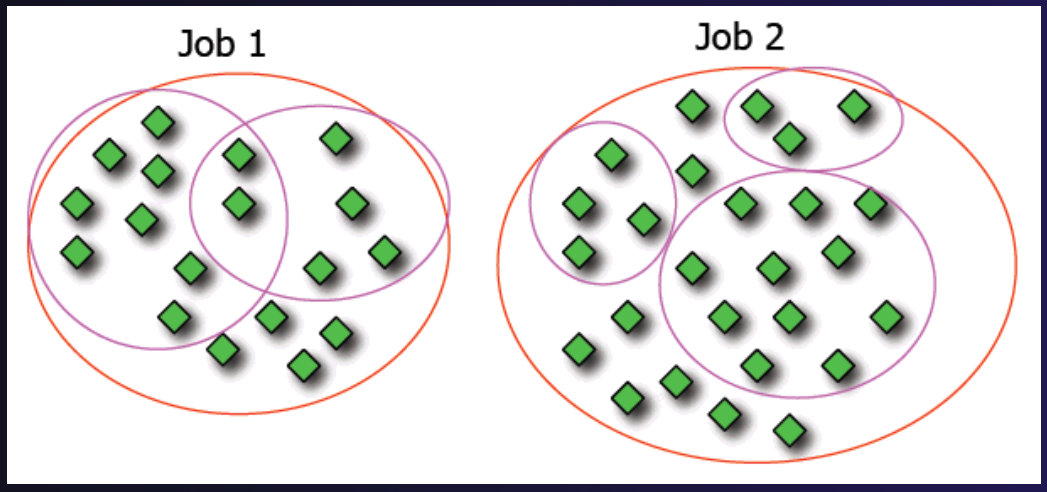
Stepping An Individual Process

- The buttons in the **Debug view** are used to control an individual process, in this case process 0
- The **Step Over** button will control just the one process
- There are now two current line markers, the first shows the position of process 0, the second shows the positions of processes 1-4



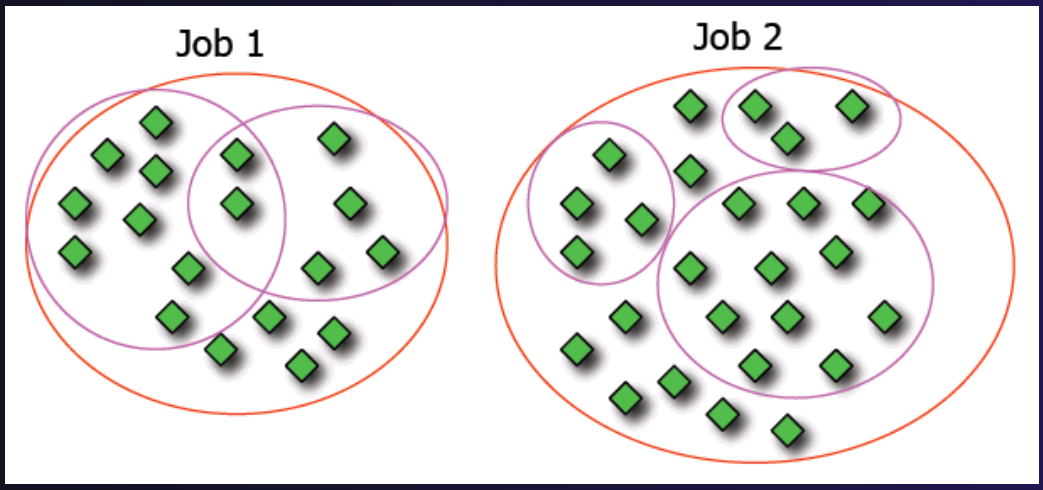
Process Sets (1)

- 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



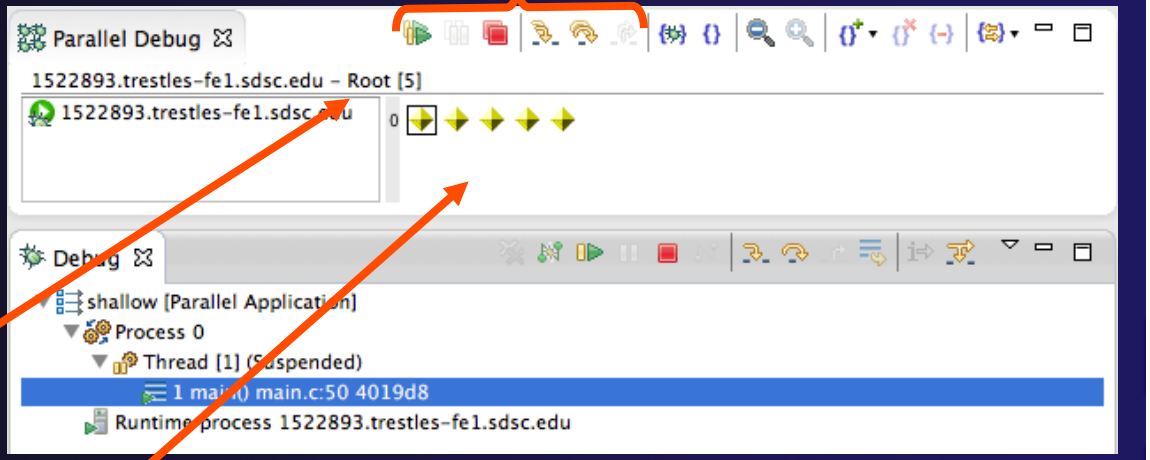
Process Sets (2)

- 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

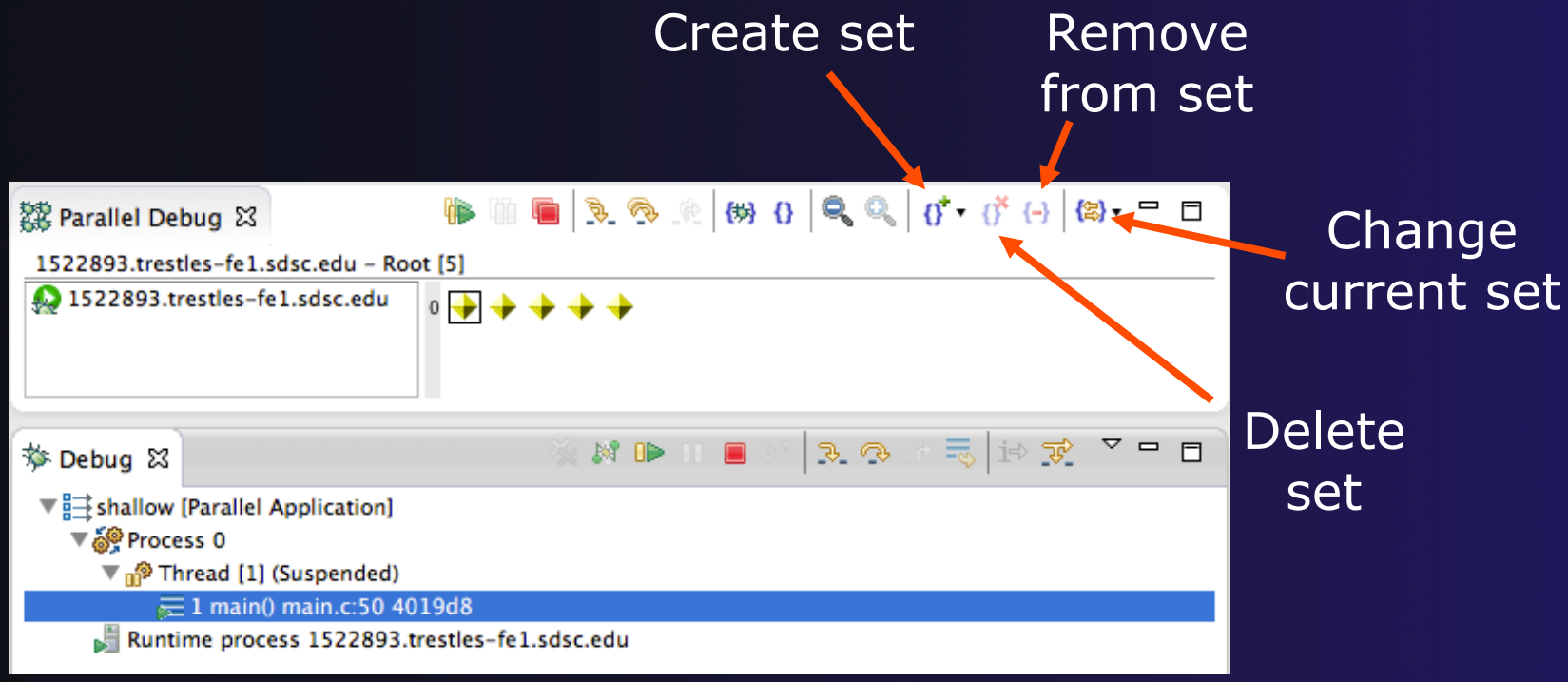
- Debug operations on the **Parallel Debug view** toolbar always apply to the current set:
  - Resume, suspend, stop, step into, step over, step return
- The current process set is listed next to job name along with number of processes in the set
- The processes in process set are visible in right hand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



Root set = all processes

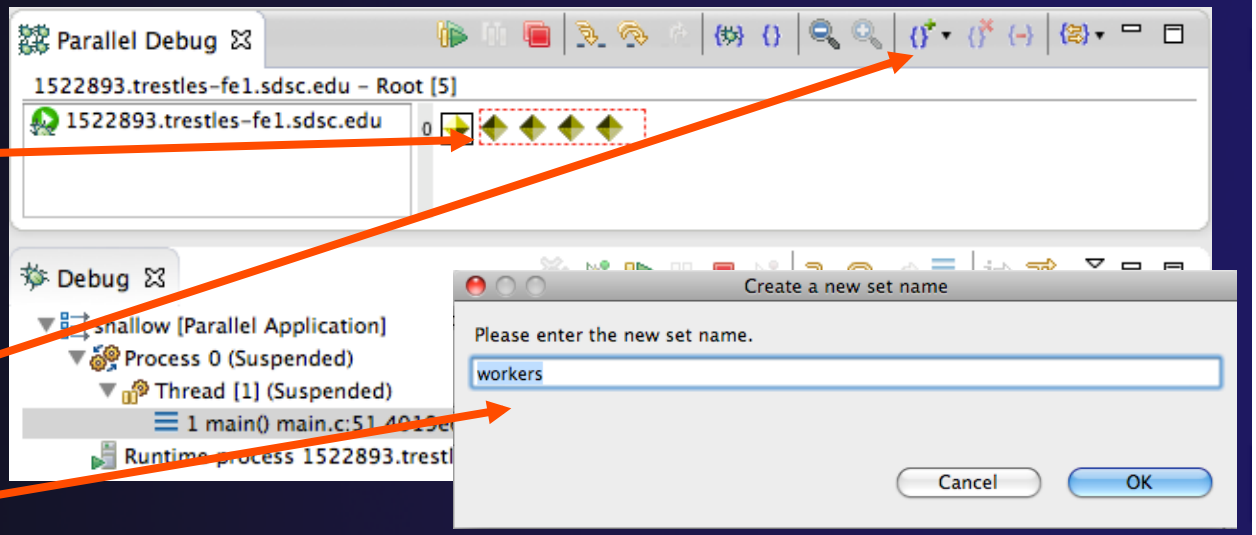
Managing Process Sets

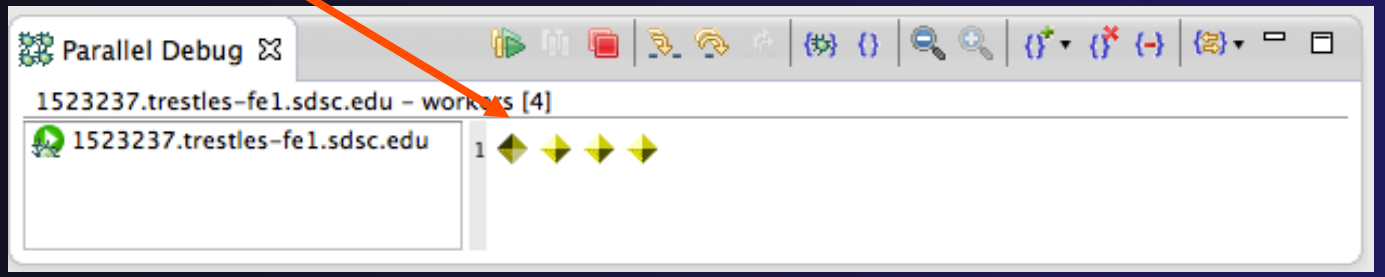
- The remaining icons in the toolbar of the **Parallel Debug view** allow you to create, modify, and delete process sets, and to change the current process set



Creating A New Process Set

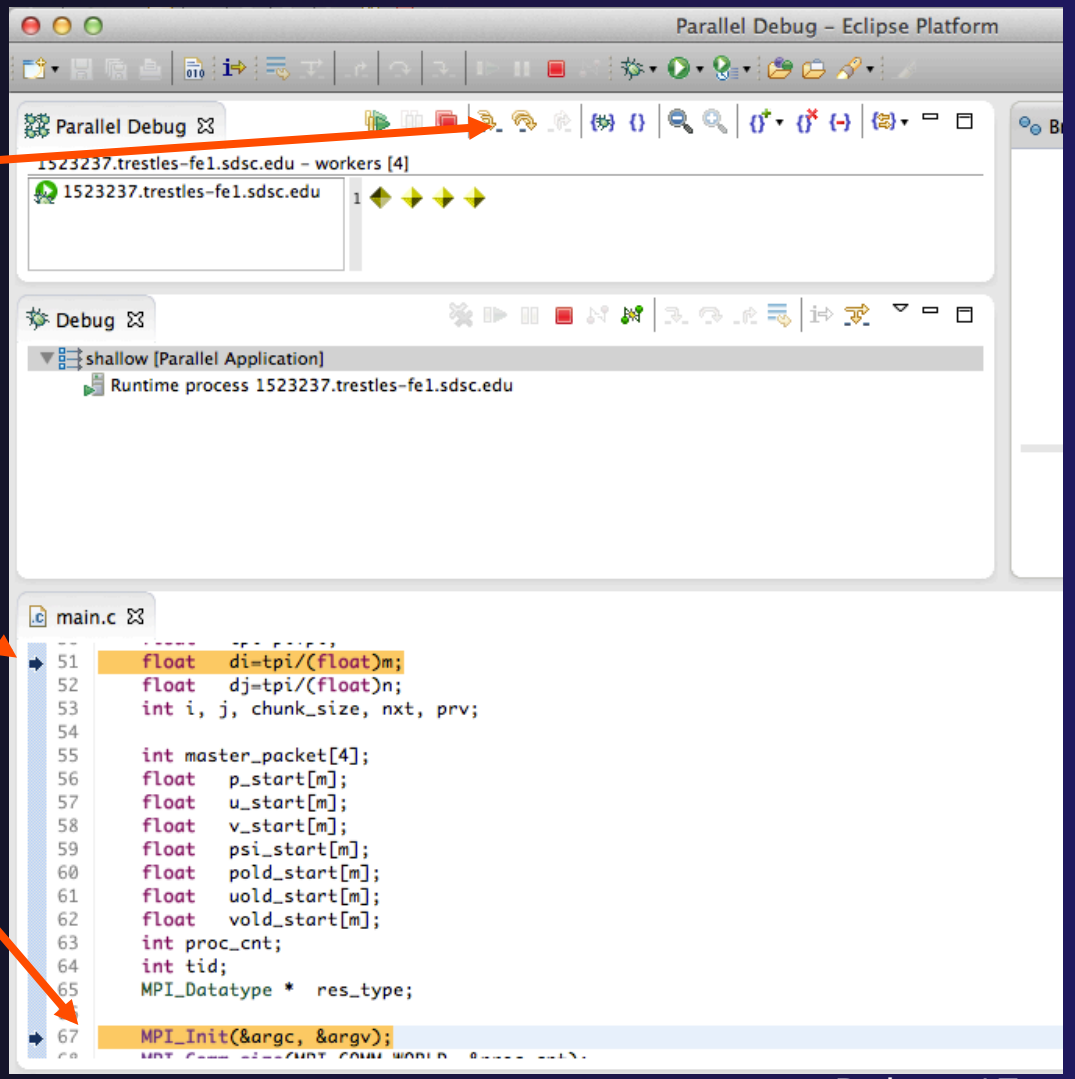
- Select the processes in the set by clicking and dragging, in this case, the last three
- The **Create Set** button enables a new process set to be created
- The set can be given a name, in this case **workers**
- The view is changed to display only the selected processes





Stepping Using New Process Set

- With the **workers** set active, the **Step Over** button will now operated on only these processes
- Only the first line marker will move
- After stepping a couple more times, two line markers will be visible, one for the single master process, and one for the 4 worker processes

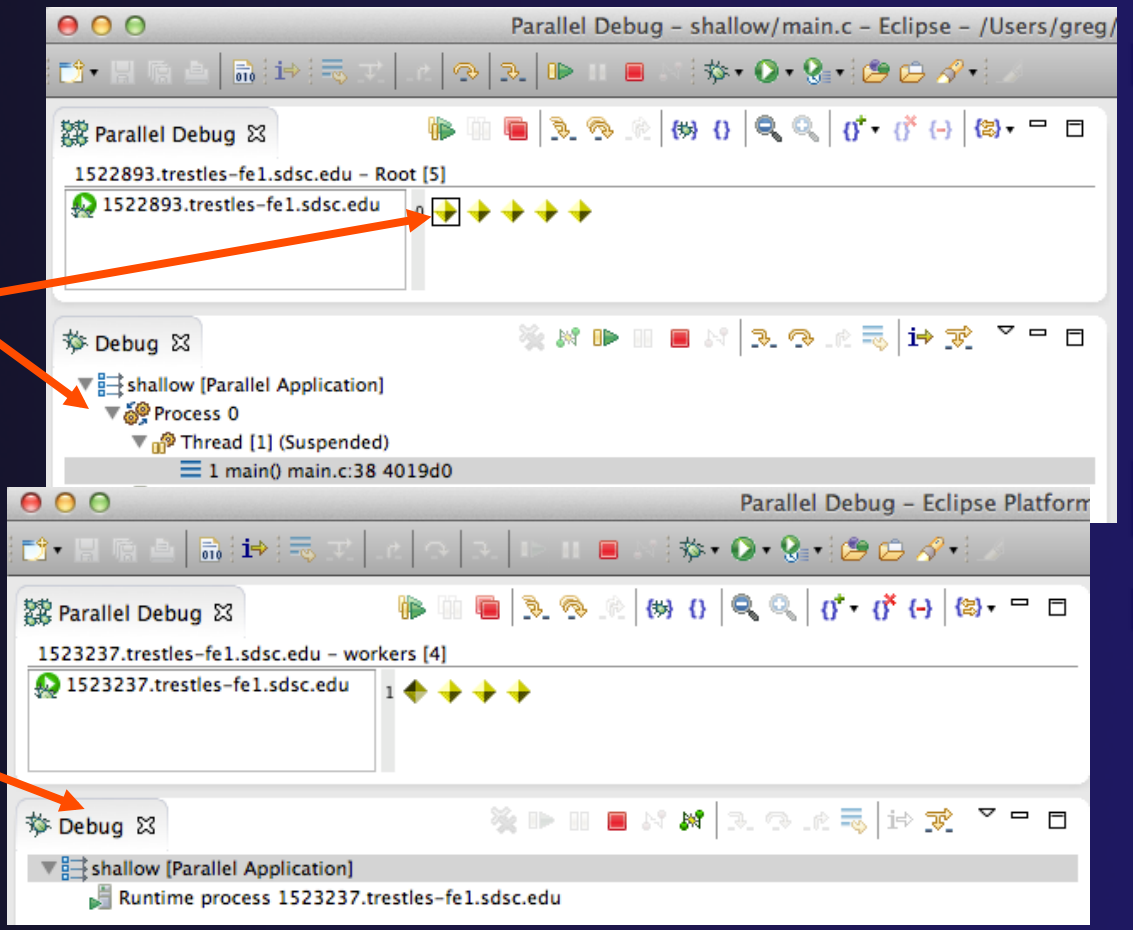


Process Registration

- 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)

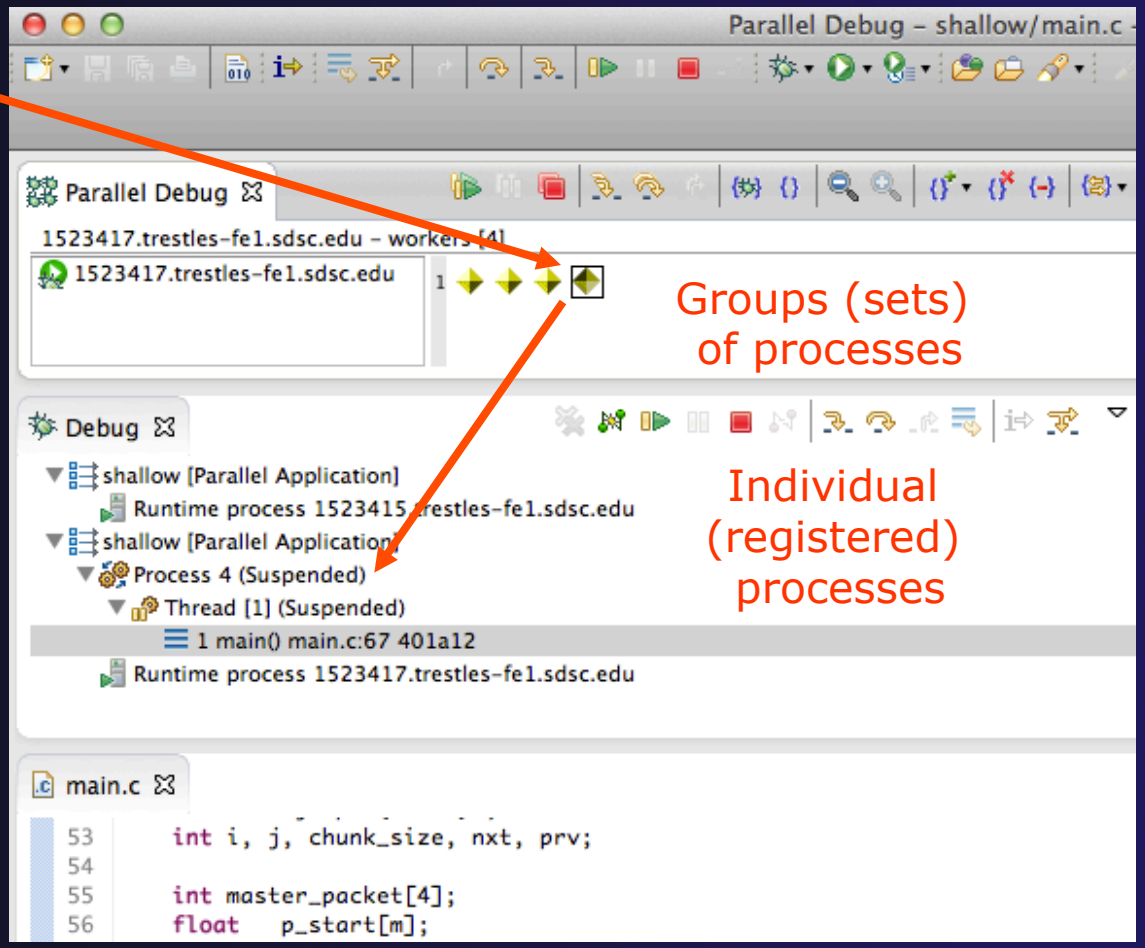
- By default, process 0 was registered when the debug session was launched
- Registered processes are surrounded by a box and shown in the Debug view



- 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

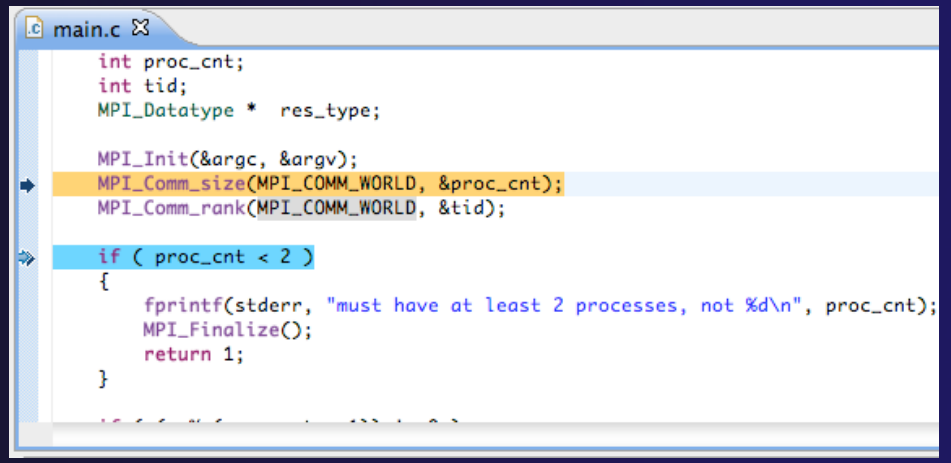


Current Line Marker

- 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

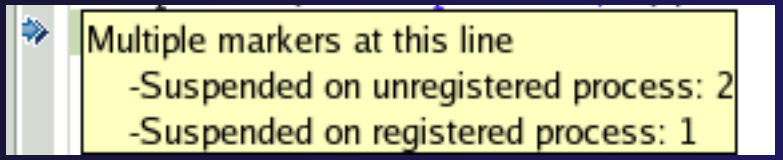
- The highlight color depends on the processes suspended at that line:
  - **Blue:** All registered process(es)
  - **Orange:** All unregistered process(es)
  - **Green:** Registered or unregistered process with no source line (e.g. suspended in a library routine)
- The marker depends on the type of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

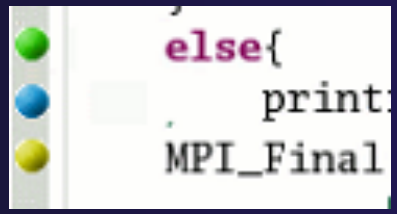


Exercise

1. From the initial debugger session, step all processes until the current line is just after MPI_Init (line 68)
2. Create a process set called “workers” containing processes 1-4
3. Step the “worker” processes twice, observe two line markers
4. Hover over markers to see properties
5. Switch to the “root” set
6. Step only process 0 twice so that all processes are now at line 71 (hint – use the debug view)

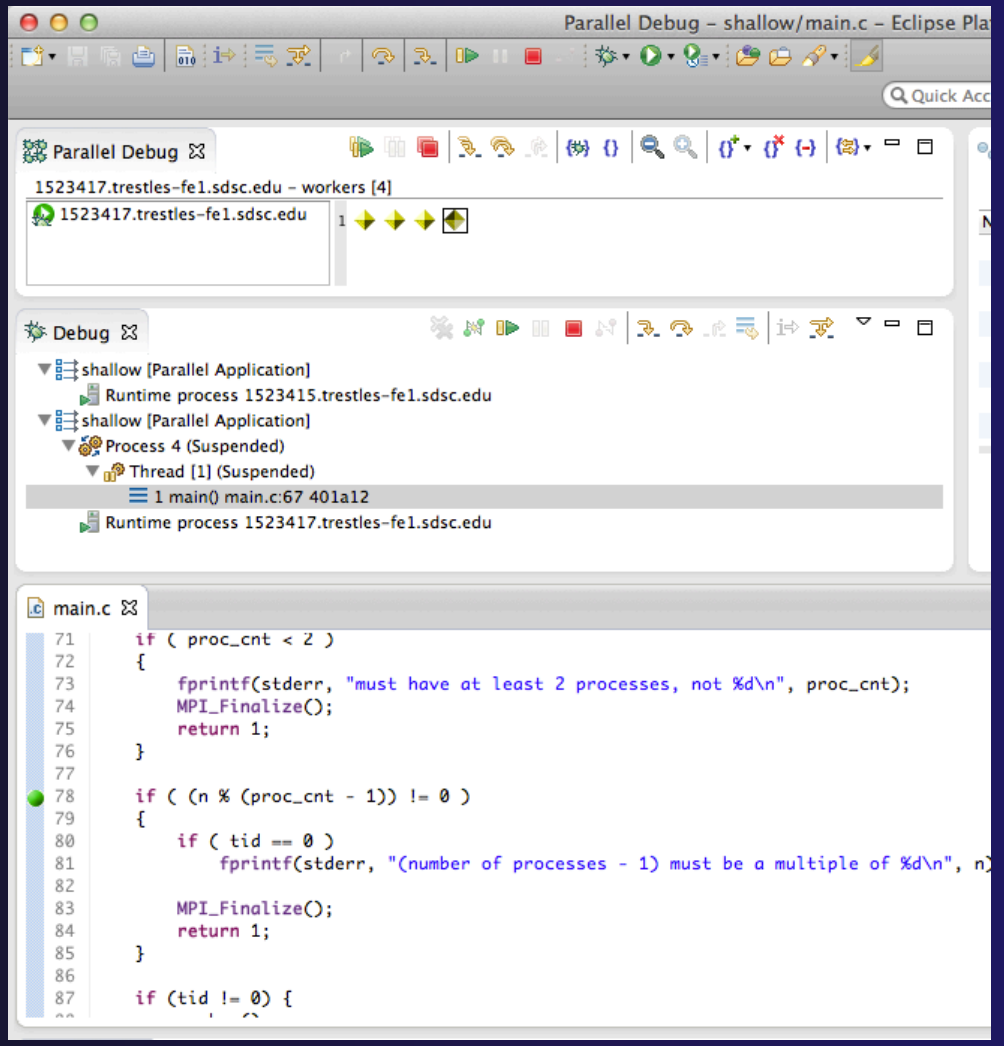
Breakpoints

- 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



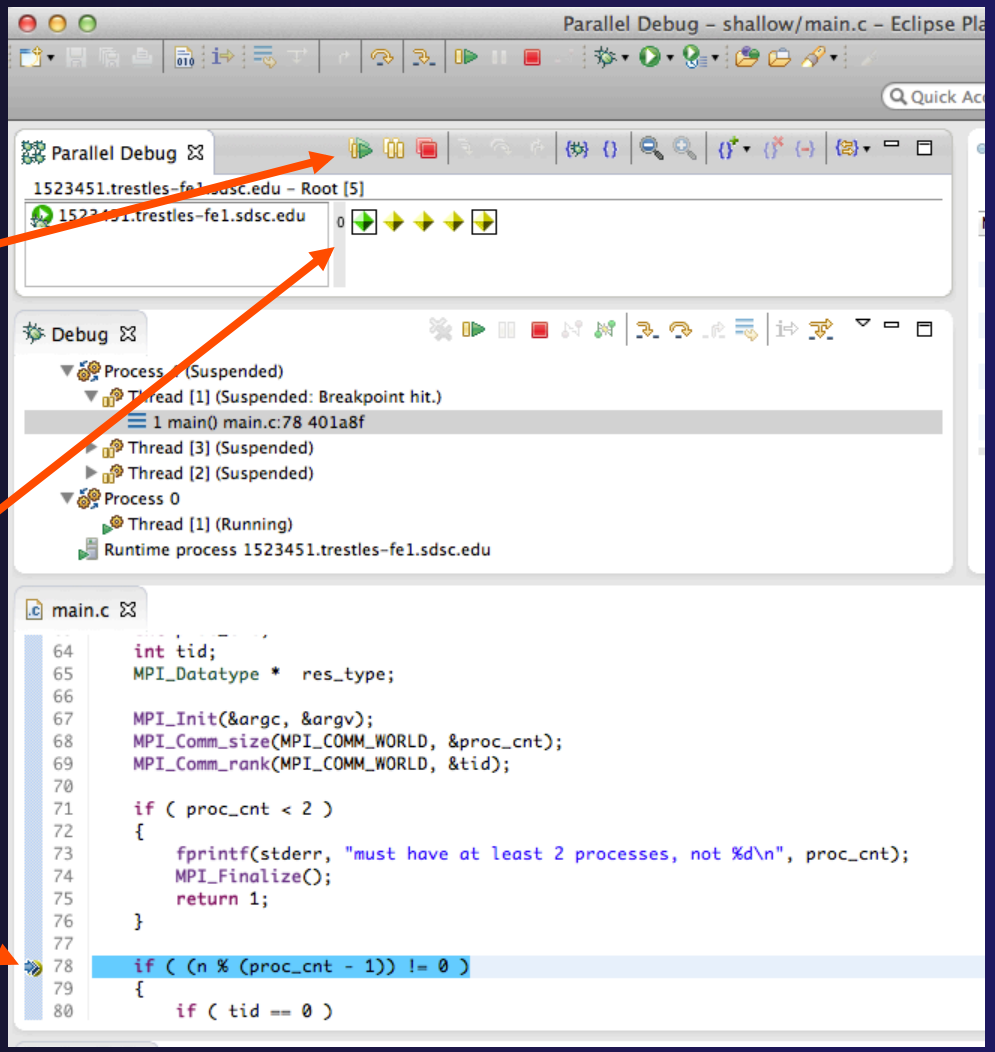
Creating A Breakpoint

- 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



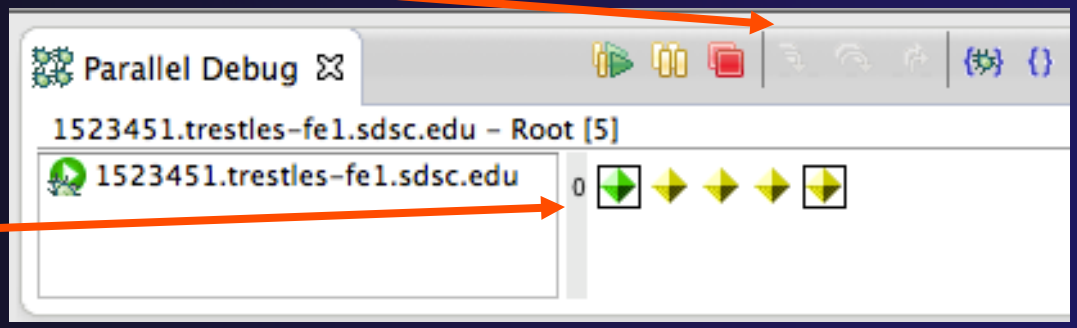
Hitting the Breakpoint

- 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 1-4 are suspended on the breakpoint

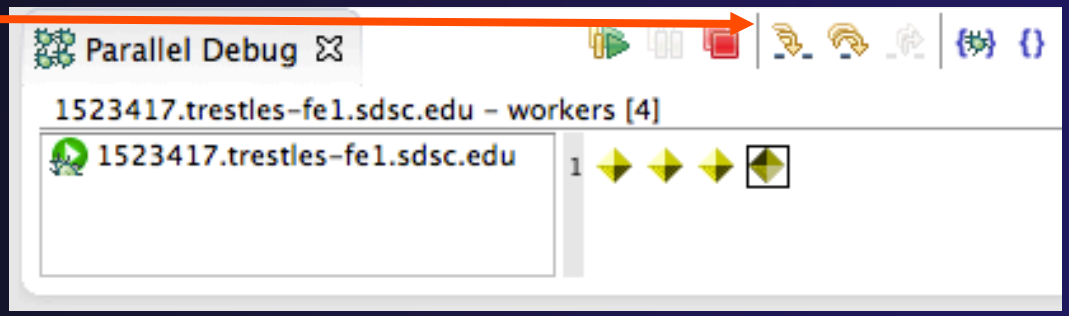


More On Stepping

- 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

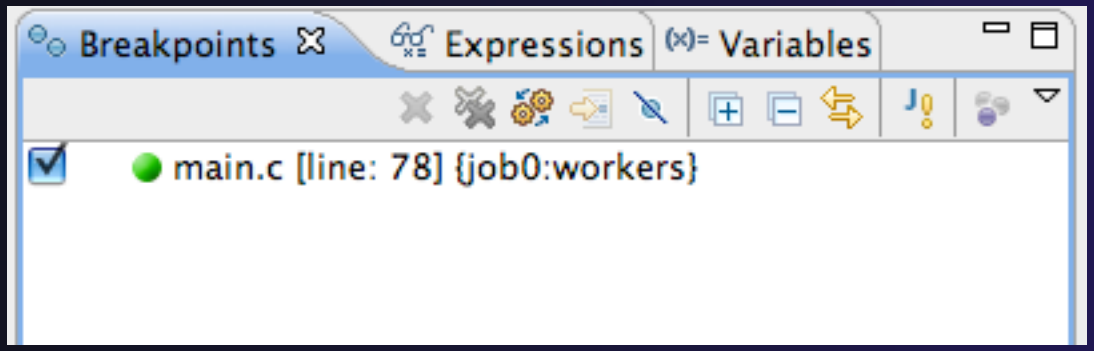


- Switch to the set of suspended processes (the **workers** set)
- You will now see the **Step** buttons become enabled



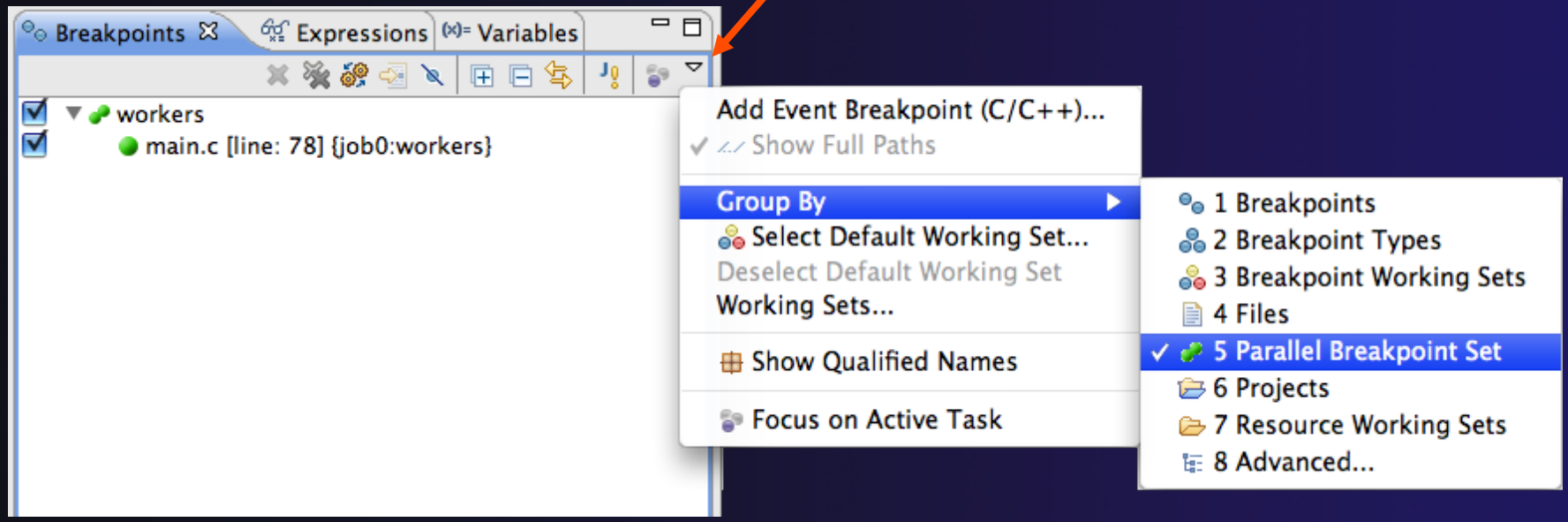
Breakpoint Information

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



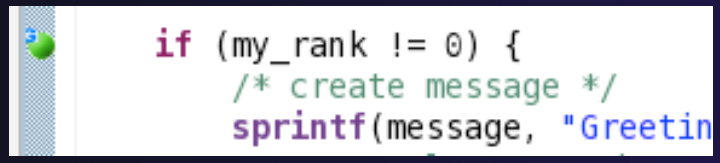
Breakpoints View

- 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

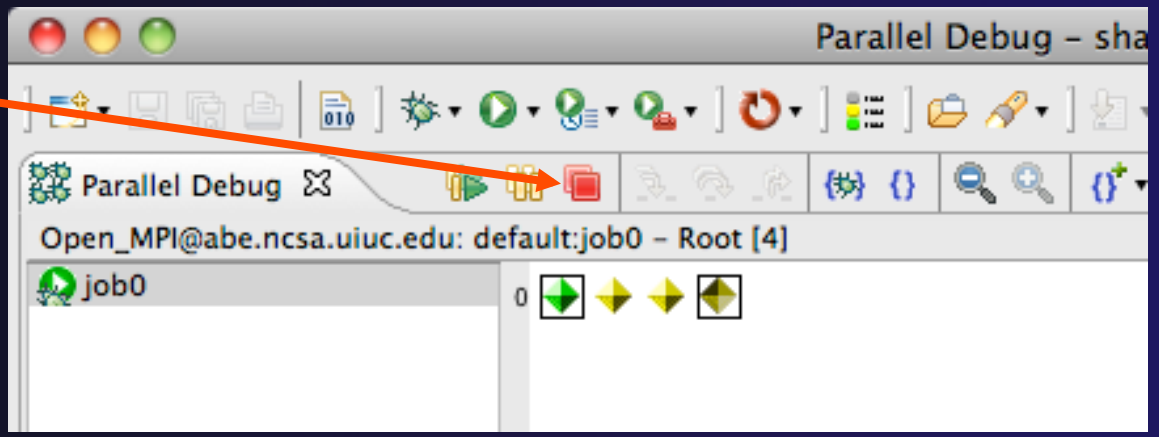


Exercise

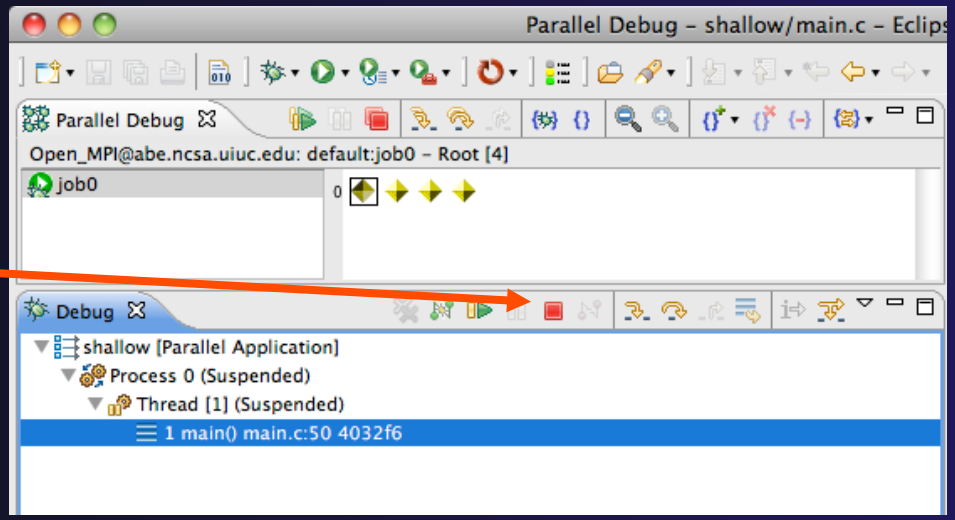
1. Select the “worker” process set
2. Create a breakpoint by double-clicking on right hand bar at line 88 (worker function)
3. Hover over breakpoint to see properties
4. Switch to “root” process set
5. Observer breakpoint color changes to blue
6. Resume all processes
7. Observe “worker” processes at breakpoint, and process 0 still running (green icon)
8. Switch to “worker” process set
9. Step “worker” processes over worker() function
10. Observe output from program

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

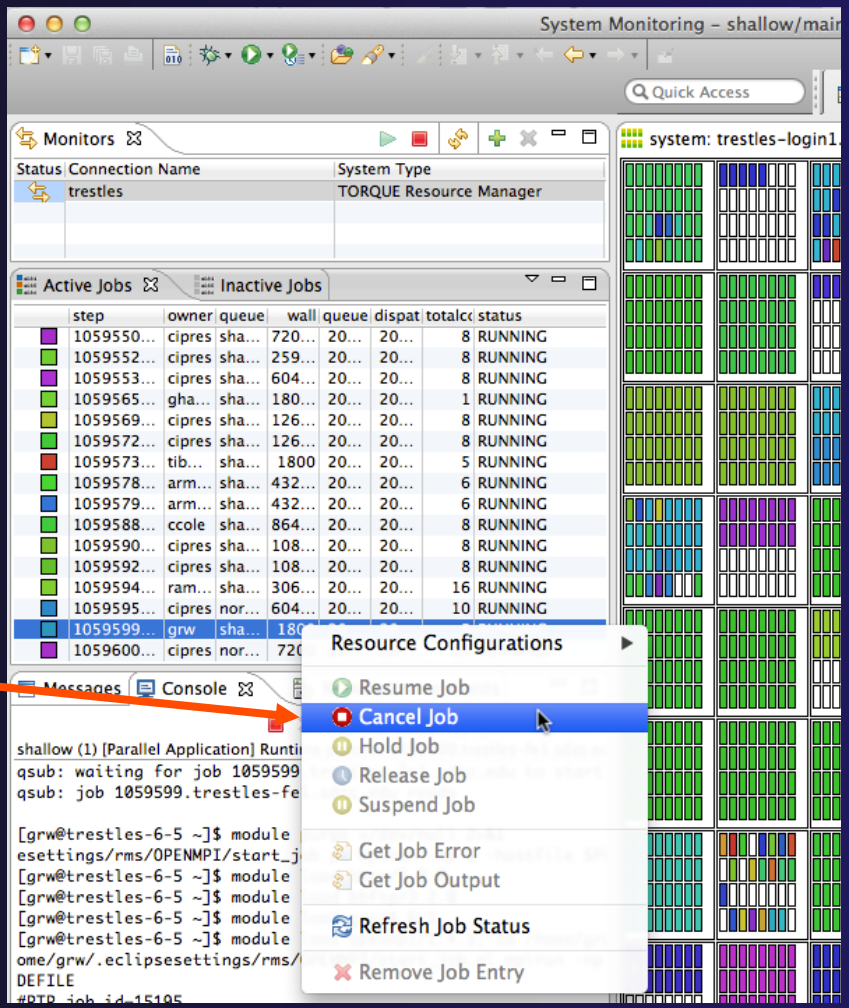


- You can also use the terminate icon in the **Debug** view to terminate the currently selected process



Cancelling The Job

- Interactive jobs will continue until the reservation time has expired
- You can cancel the job once the debug session is finished
- Locate the job in the **Active Jobs** view
  - Use the view menu to filter for only your jobs if there are too many
- Right click on the job and select **Cancel Job**



Exercise

1. Switch to the “root” set
2. Terminate all processes
3. Switch to the System Monitoring perspective
4. Right-click on your running job and select **Cancel**

Optional Exercise

1. Launch another debug job
2. Create a breakpoint at line 71 in main.c
3. Resume all processes
4. Select the Variables view tab if not already selected
5. Observe value of the “tid” variable
6. Register one of the worker processes
7. Select stack frame of worker process in Debug view
8. Observe value of the “tid” variable matches worker process
9. Switch to the breakpoints view, change grouping
10. Terminate all processes
11. Switch to the System Monitoring perspective and cancel the job