Periscope
Tutorial Exercise
NPB-MPI/BT

M. Gerndt, V. Petkov, Y. Oleynik, S. Benedict
Technische Universität München
petkovve@in.tum.de

March 2010

NPB-BT Exercise

- Intermediate-level tutorial example
- Available in MPI, OpenMP, hybrid OpenMP/MPI variants
  - also MPI File I/O variants (collective & individual)
- Automatic performance properties search with Periscope:
  - Source code instrumentation
    - Loops, MPI & application function calls
  - Automatic search for slow MPI communication patterns
  - Results exploration with Eclipse based GUI
- Manual instrumentation optimization

Exercise steps

0. Configuration of Periscope
1. Program instrumentation: `psc_instrument`
2. Periscope analysis: `psc_frontend`
3. Performance properties exploration: `Periscope GUI`

Configuring Periscope

- Before first use of Periscope, one has to create the configuration file `.periscope` in the home directory. Configuration could be copied from `$PERISCOPE_ROOT`:

```
% cp $PERISCOPE_ROOT/etc/periscope.sample ~/.periscope
```

- It should look like:

```
MACHINE          = localhost       //hostname
SITE             = LiveDVD
REGSERVICE_HOST  = localhost       //host of registry
REGSERVICE_PORT  = 50001           //port of the registry
APPL_BASEPORT    = 51000           //first port for application
AGENT_BASEPORT   = 50002           //first port agent hierarchy
```

Instrumenting application

- The Periscope agents and the application processes register with a `registry`. The registry is started via:

```
% psc_regsrv &
```

- To enable performance measurement, the program has to be instrumented. This is done with `psc_instrument`:

```
% psc_instrument
Periscope Source-to-Source Instrumentation Wrapper
Usage: psc_instrument [-t regions] [-n] [-s sir] [-v] [-d] compiler
                  [options] file [libs]
        -t Types of regions to instrument separated by spaces
                  (e.g. -t "user loop call")
        -s Filename for the resulting SIR file (default: appl.sir)
        -v Verbose output
        -d Debug mode: keeps the instrumented source files
                  after the compilation
        -n Prints each step of the compilation instead of executing them
```

Instrumenting NPB-MPI-BT

- Substitute compile/link commands in Makefile definitions (`config/make.def`) with `psc_instrument:`

```
MPIF77 = psc_instrument -s bt.sir -t "user loop call" \
         mpif77 -I/usr/local/packages/openmpi/include

FLINK = $(MPIF77)
FFLAGS = -O

mpi-bt: $(OBJECTS)
        $(FLINK) $(FFLAGS) -o mpi-bt $(OBJECTS)
.f.o:
        $(MPIF77) $(FFLAGS) -c $<
```

NPB-MPI-BT instrumentation and build

- Return to root directory and clean-up

```
% make clean
```

- Re-build BT with the original command

```
% make bt CLASS=W NPROCS=16
   ========================================
   =      NAS Parallel Benchmarks 3.3      =
   =      MPI/F77/C                        =
   ========================================
cd BT; make NPROCS=16 CLASS=W SUBTYPE= VERSION=
make[1]: Entering directory `BT'
...
psc_instrument -s bt.sir -t "user loop call" mpif77 -c  -O -g bt.f
psc_instrument -s bt.sir -t "user loop call" mpif77 -c  -O -g make_set.f
…
psc_instrument -s bt.sir -t "user loop call" mpif77 -O \
-o ../bin.periscope/bt_W.16 bt.o ...
Built executable ../bin.periscope/bt_W.16
make[1]: Leaving directory `BT'
```

- Change directory to bin.periscope

```
% cd bin.periscope
```

Running Periscope

- Periscope is started via the frontend. It automatically starts application and hierarchy of analysis agents.
- Run `psc_frontend --help` for brief usage information

```
% psc_frontend --help
Usage: psc_frontend <options>
  [--help]                (displays this help message)
  [--quiet]               (do not display debug messages)
  [--registry=host:port]  (address of the registry service, optional)
  [--port=n]              (local port number, optional)
  [--maxfan=n]            (max. number of child agents, default=4)
  [--timeout=secs]        (timeout for startup of agent hierarchy)
  [--delay=n]              (search delay in phase executions)
  [--appname=name]
  [--apprun=commandline]
  [--mpinumprocs=number of MPI processes]
  [--ompnumthreads=number of OpenMP threads]
…
  [--strategy=name]
  [--sir=name]
  [--phase=(FileID,RFL)]
  [--debug=level]
```

Running Periscope

- Run Periscope analysis by executing `psc_frontend` with the following command

```
% psc_frontend --sir=../BT/appl.sir --apprun=./bt_W.16 --mpinumprocs=16
[psc_frontend][DBG0:fe] Agent network UP and RUNNING. Starting search.

 NAS Parallel Benchmarks 3.3 -- BT Benchmark
 [...]
 Time step 200
 BT Benchmark Completed.

----------
End Periscope run! Search took 60.5 seconds (33.3 seconds for startup)
```

- Frontend will write the detected properties into the file `properties.psc` in the current directory. It should be copied into the BT source directory

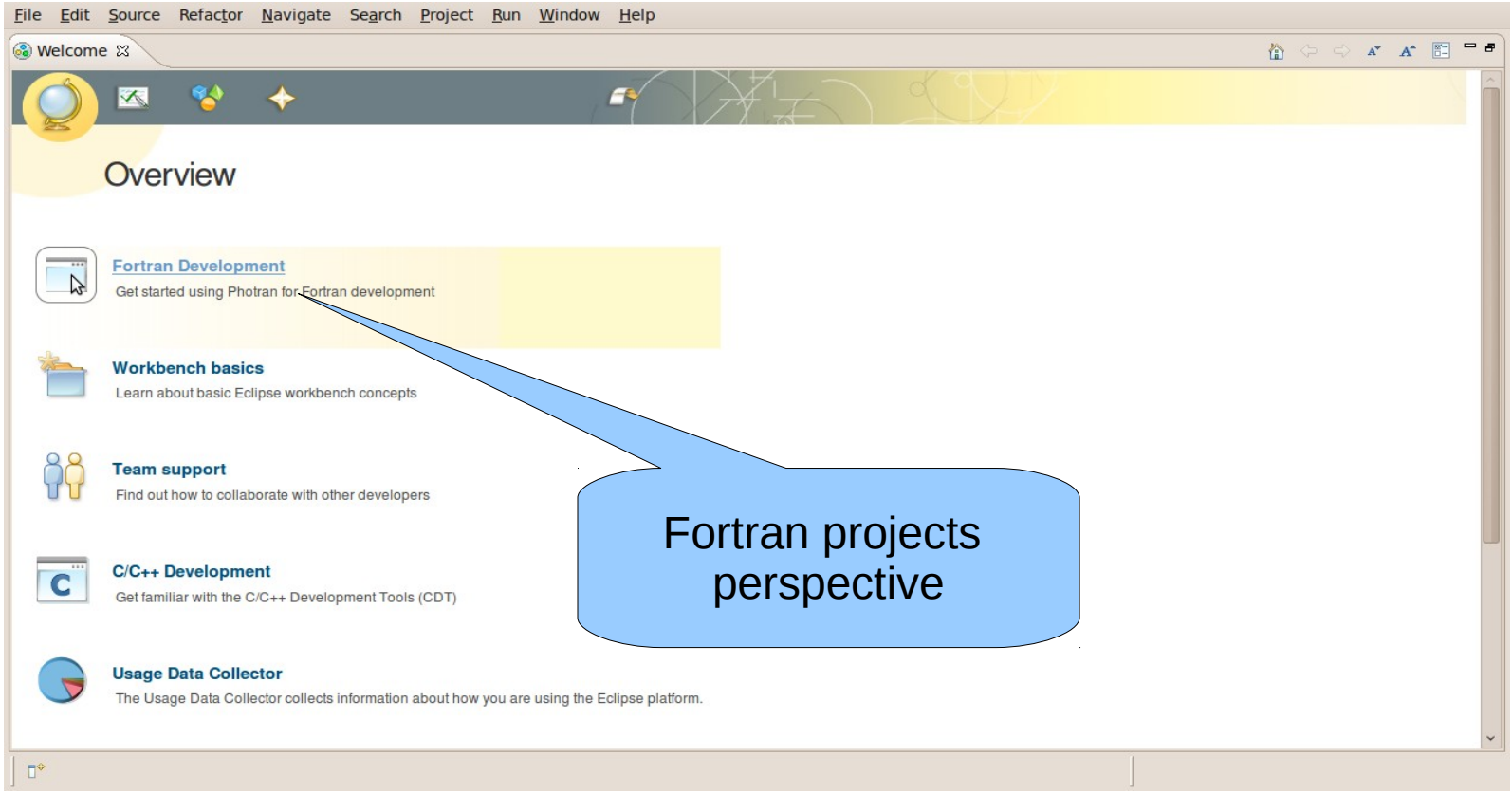
```
% cp properties.psc ../BT
```

Starting Periscope GUI

- Start `Eclipse` with Periscope GUI from console

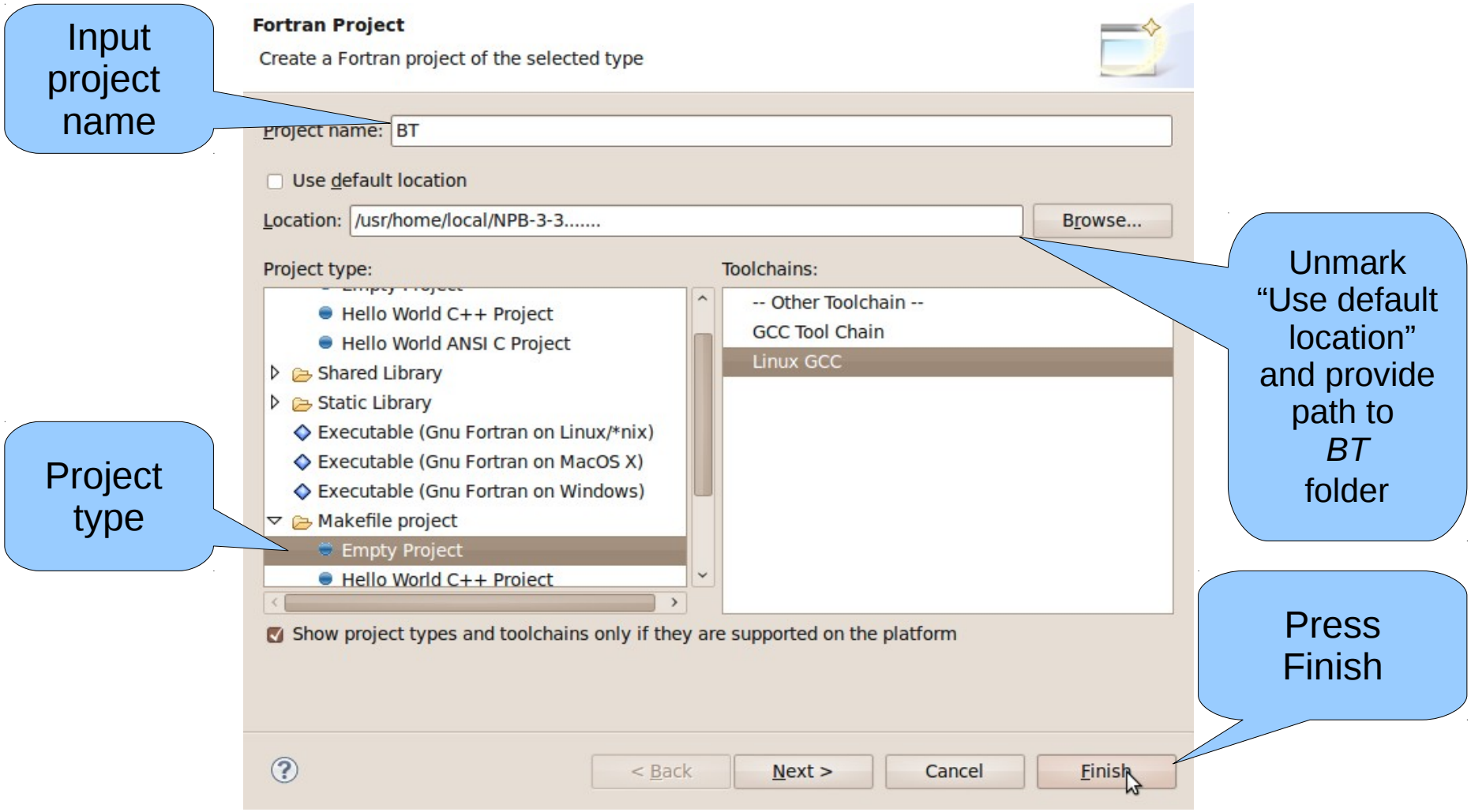
```
% eclipse
```

- Or by double-click on Eclipse pictogram on the Desktop

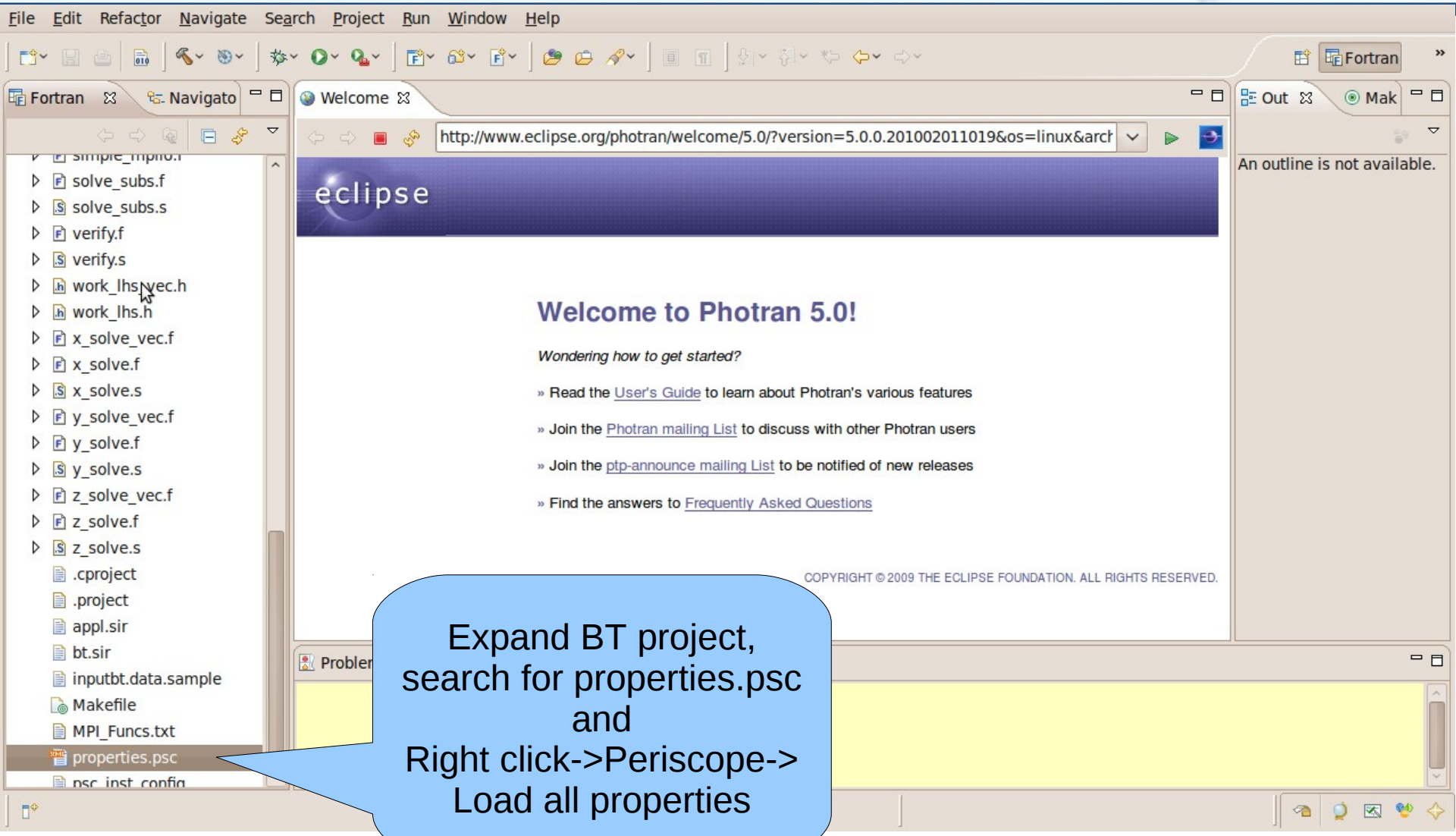


Creating Fortran Project

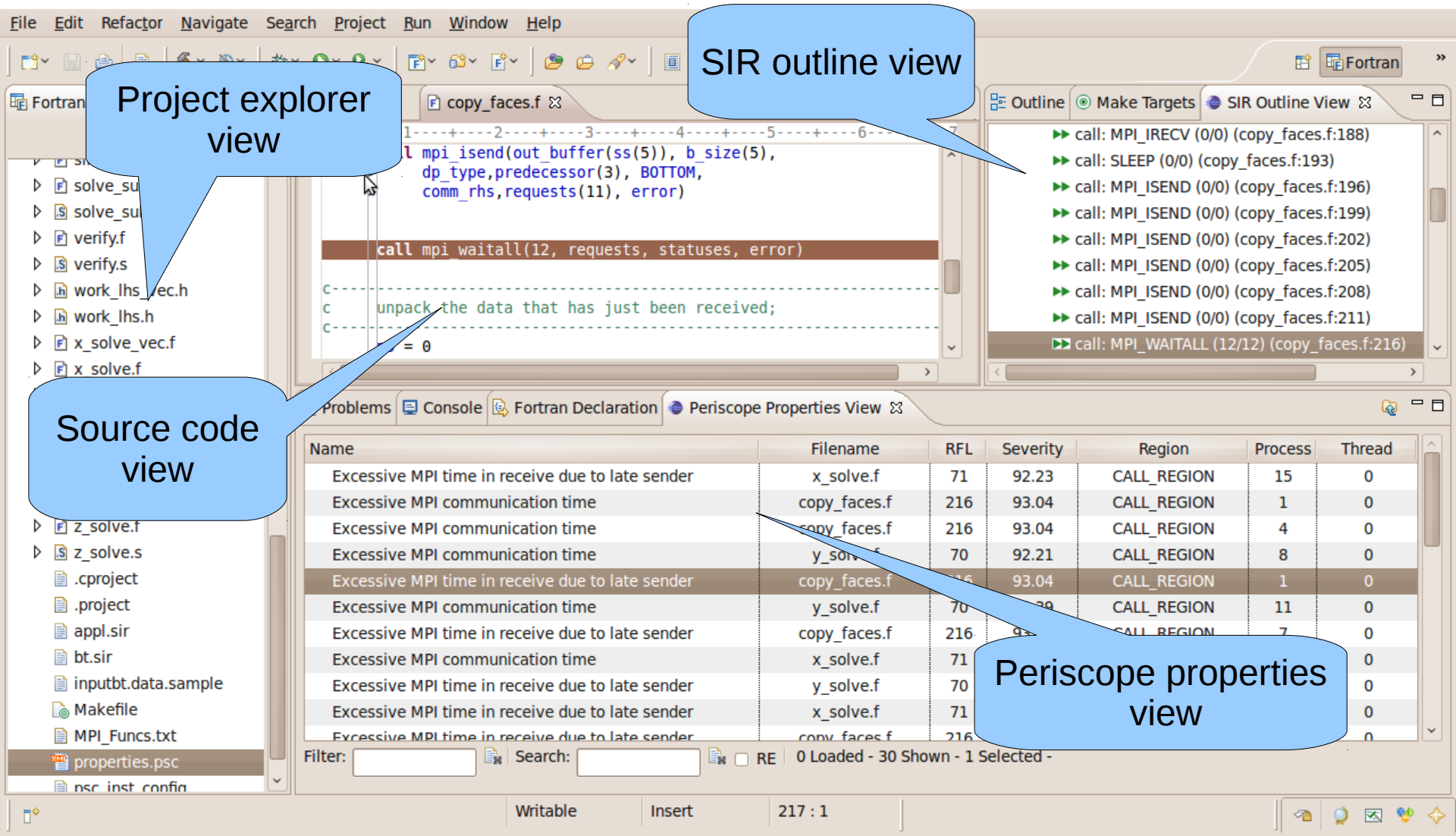
- File->New->Project... → Fortran->Fortran Project



Loading properties



Periscope GUI



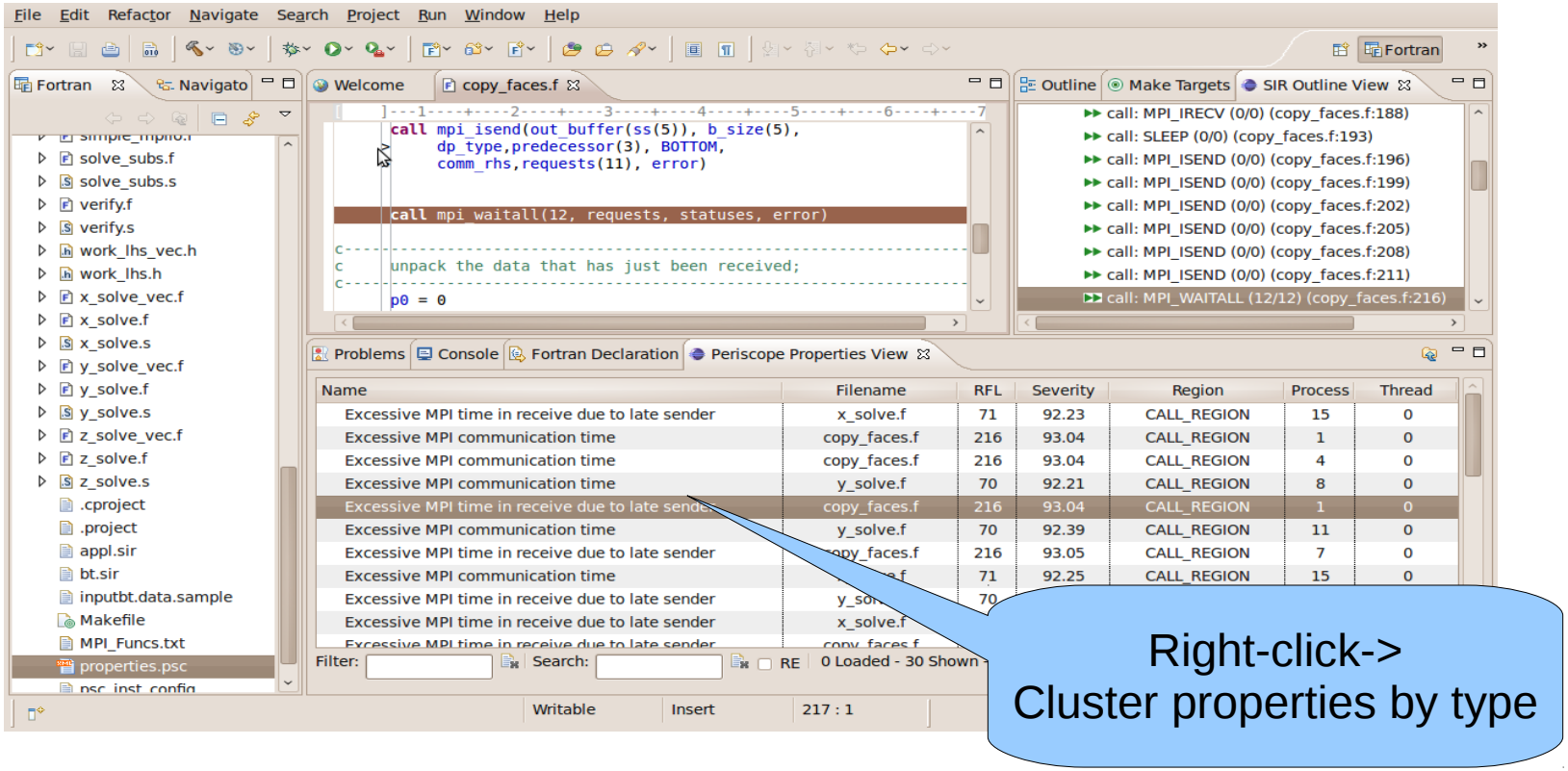
Periscope GUI report exploration features

- Multi-functional table is used in the GUI for Eclipse for the visualization of bottlenecks
  - Multiple criteria sorting algorithm
  - Complex categorization utility
  - Searching engine using Regular Expressions
  - Filtering operations
  - Direct navigation from the bottlenecks to their precise source location using the default IDE editor for that source file type (e.g. CDT/Photran editor).

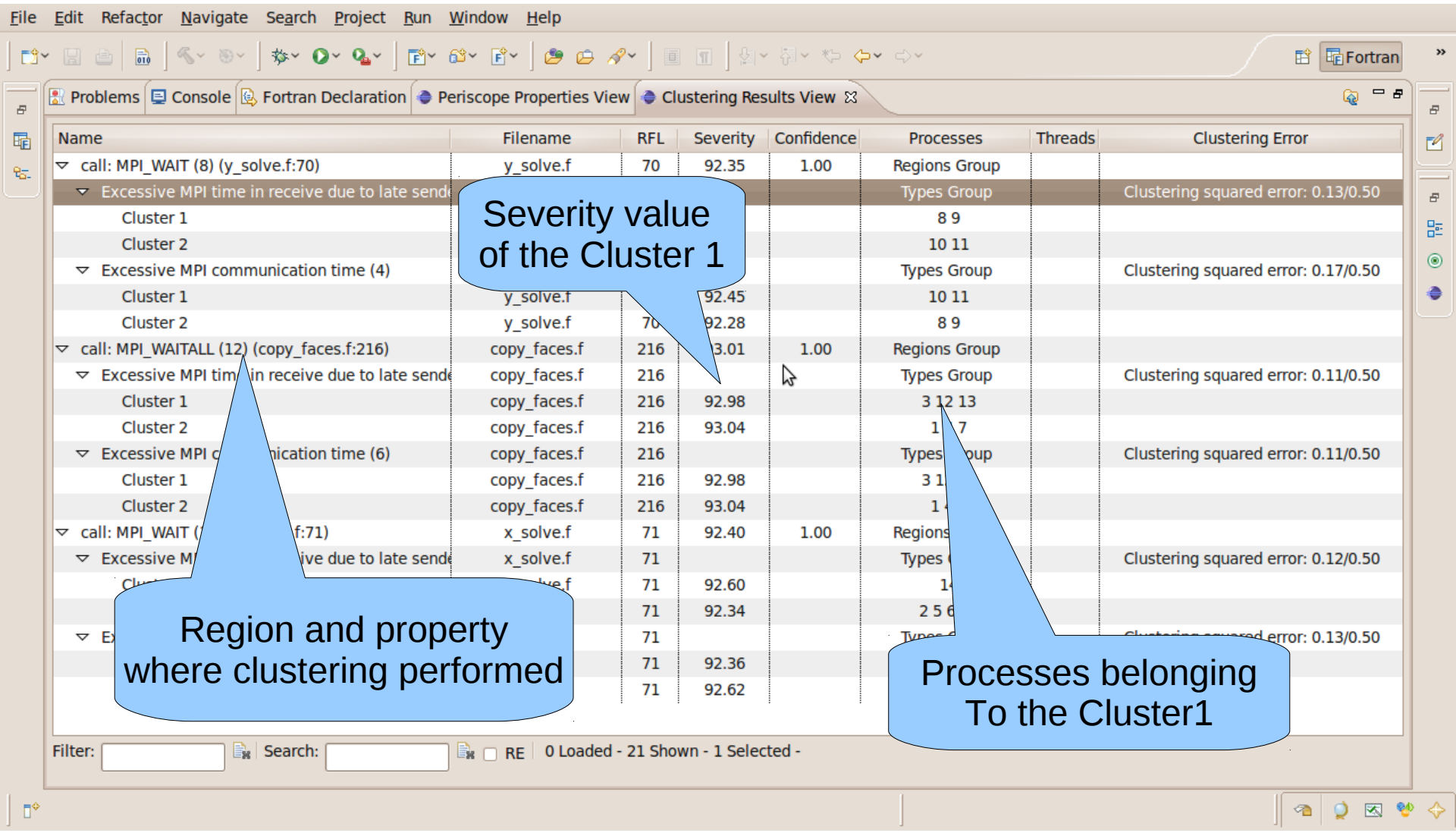
- SIR outline view shows a combination of the standard intermediate representation (SIR) of the analysed application and the distribution of its bottlenecks. The main goals of this view are to assist the navigation in the source code and attract developer's attention to the most problematic code areas.

Properties clustering

- Clustering can effectively summarize displayed properties and identify a similar performance behavior possibly hidden in the large amount of data

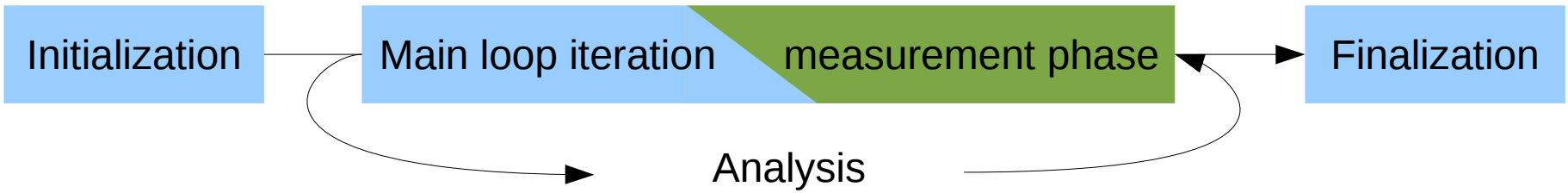


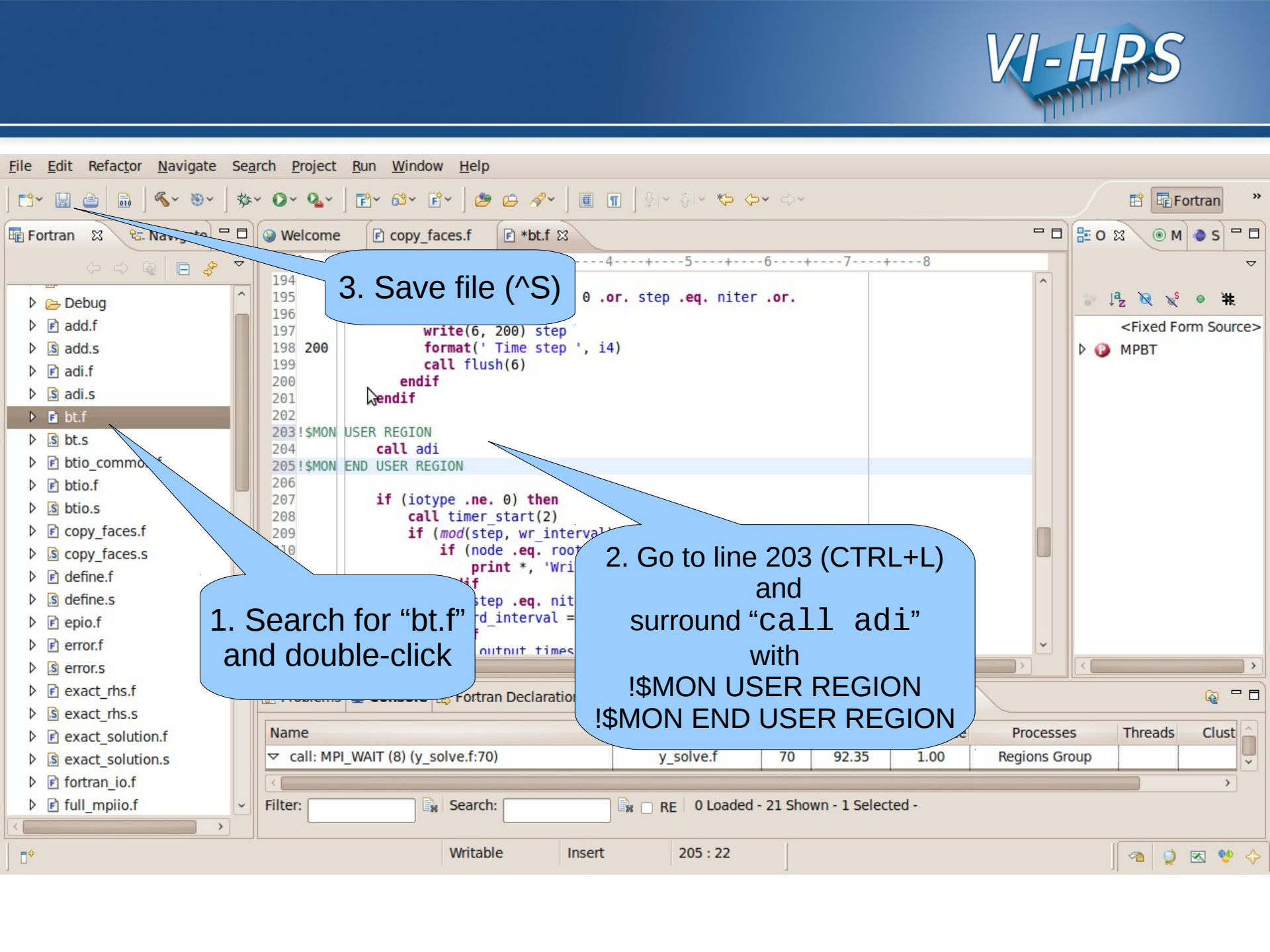
Properties clustering



Periscope Phases

- Periscope performs multiple iterative performance measurement experiments on the basis of *Phases:*
  - All measurements are performed inside phase
  - Begin and end of phase are global synchronization points
- By default phase is the whole program
  - Needs restart if multiple experiments required (single core performance analysis strategies require multiple experiments)
  - Unnecessary code parts also measured
- User specified region marked with `!$MON USER REGION` and `!$MON END USER REGION` will be used as phase:
  - Typically main loop of application → no need for restart, faster analysis
  - Unnecessary code parts are not measured → less measurements overhead
  - Severity value is normalized on the main loop iteration time → more precise performance impact estimation




VI-HPS
File Edit Refactor Navigate Search Project Run Window Help
Fortran
Welcome
copy_faces.f
*bt.f
3. Save file (^S)
1. Search for “bt.f” and double-click
2. Go to line 203 (CTRL+L) and surround “call adi” with !$MON USER REGION !$MON END USER REGION
!$MON USER REGION
call adi
!$MON END USER REGION
Name
Processes
Threads
call: MPI_WAIT (8) (y_solve.f:70)
y_solve.f
Regions Group
Filter:
Search:
RE
0 Loaded - 21 Shown - 1 Selected -
Writable
Insert
205 : 22

Repeating Periscope analysis

- Return to root directory and clean-up

```
% make clean
```

- Re-build BT with the original command

```
% make bt CLASS=W NPROCS=16
```

- Change directory into location of executable

```
% cd bin.periscope
```

Re-running Periscope

- Re-run Periscope analysis by executing `psc_frontend`

```
% psc_frontend --sir=../BT/appl.sir --apprun=./bt_W.16 --mpinumprocs=16
[psc_frontend][DBG0:fe] Agent network UP and RUNNING. Starting search.


 NAS Parallel Benchmarks 3.3 -- BT Benchmark
 [...]
 Time step 1
 BT Benchmark Completed.

----------
End Periscope run! Search took 37.2 seconds (33.3 seconds for startup)
```

- Only 1 iteration of BT required instead of 200 previous run!
- Frontend will overwrite the properties found into the file `properties.psc` in the current directory, which again need to be copied into the BT source directory

```
% cp properties.psc ../BT
```

- Re-load `properties.psc` in Periscope GUI. Now found properties should have more precise severities values