

Performance Measurement and Prediction

Luiz DeRose

Dan Reed

University of Illinois

Presentation Outline

- Source code performance prediction
 - high-level language integration
 - symbolic performance model
 - performance prediction experiments
- SvPablo performance browser

High-Level Language Integration

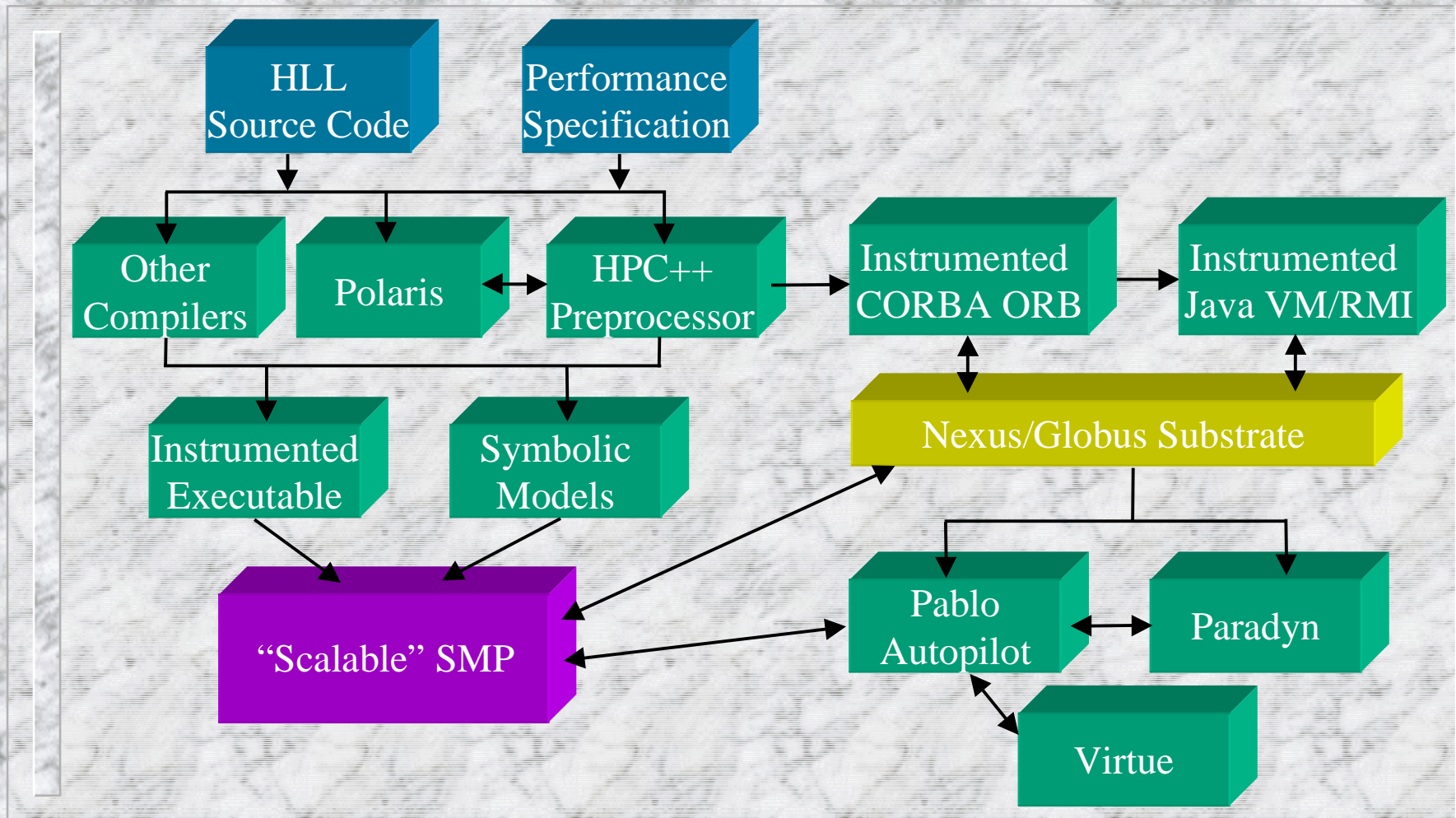
■ Motivations

- emerging high-level languages (HPF and HPC++)
- aggressive code transformations for parallelism
- large semantic gap between user and code

■ Goals

- relate dynamic performance data to source
- generate instrumented executable/simulated code
- support performance prediction
- support scalability analysis

High-Level Language Integration



Symbolic Performance Prediction

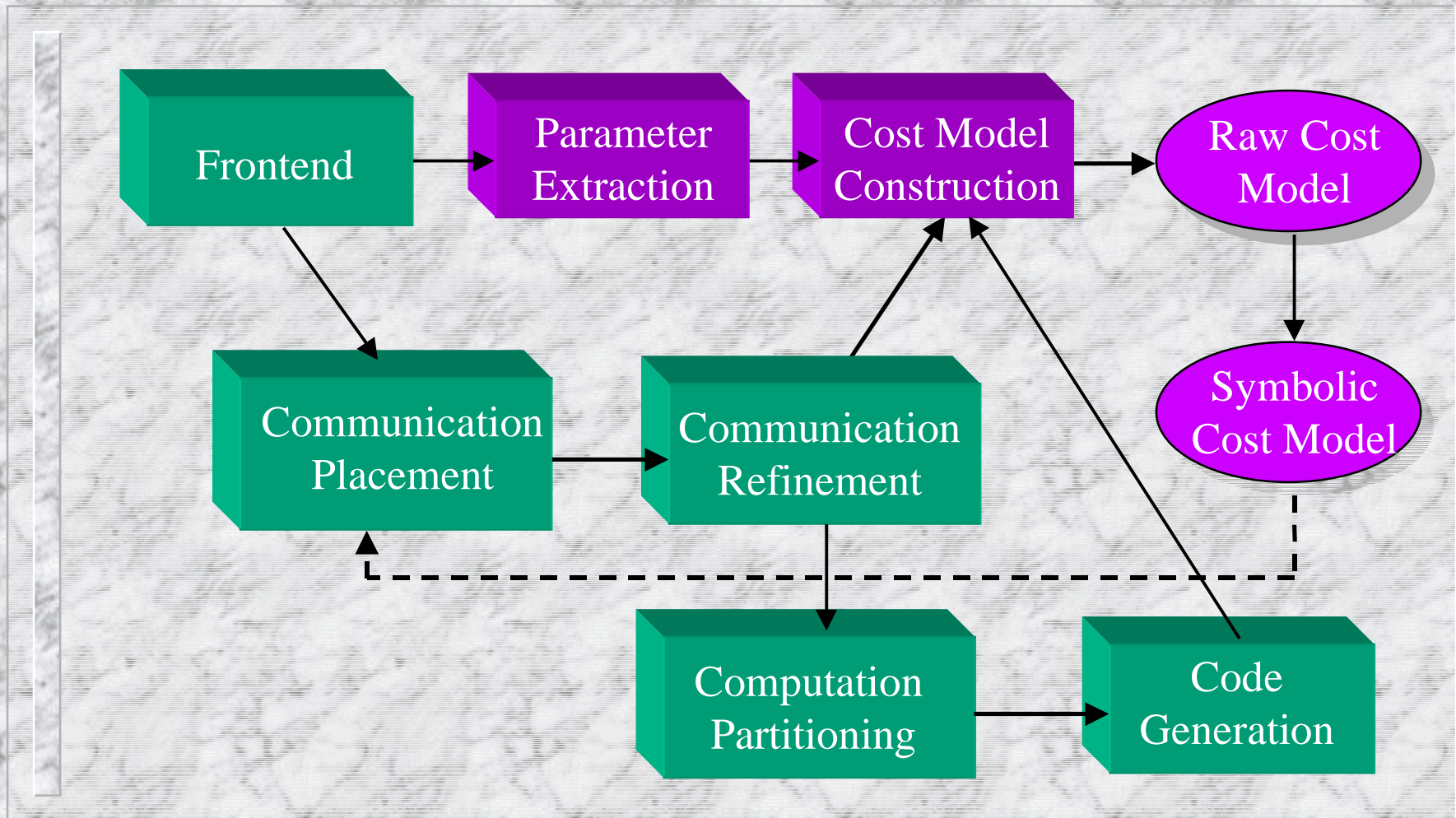
■ Rationale

- rapid assessment of design alternatives
- identification of performance bottlenecks

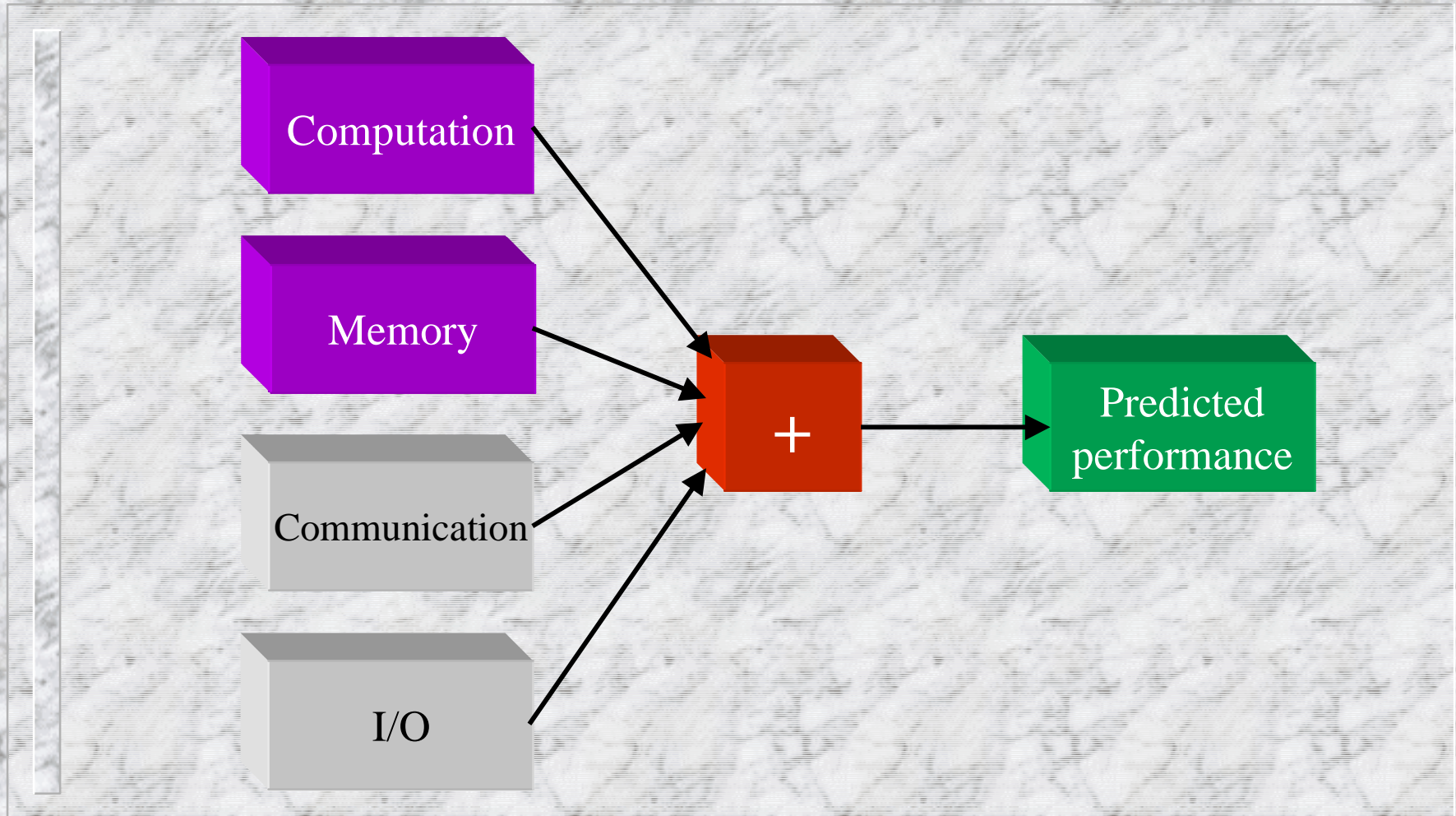
■ Approach

- compile-time generation of cost expressions
- augmentation with selected measurements

Fortran 95D Symbolic Prediction



Performance Prediction Model



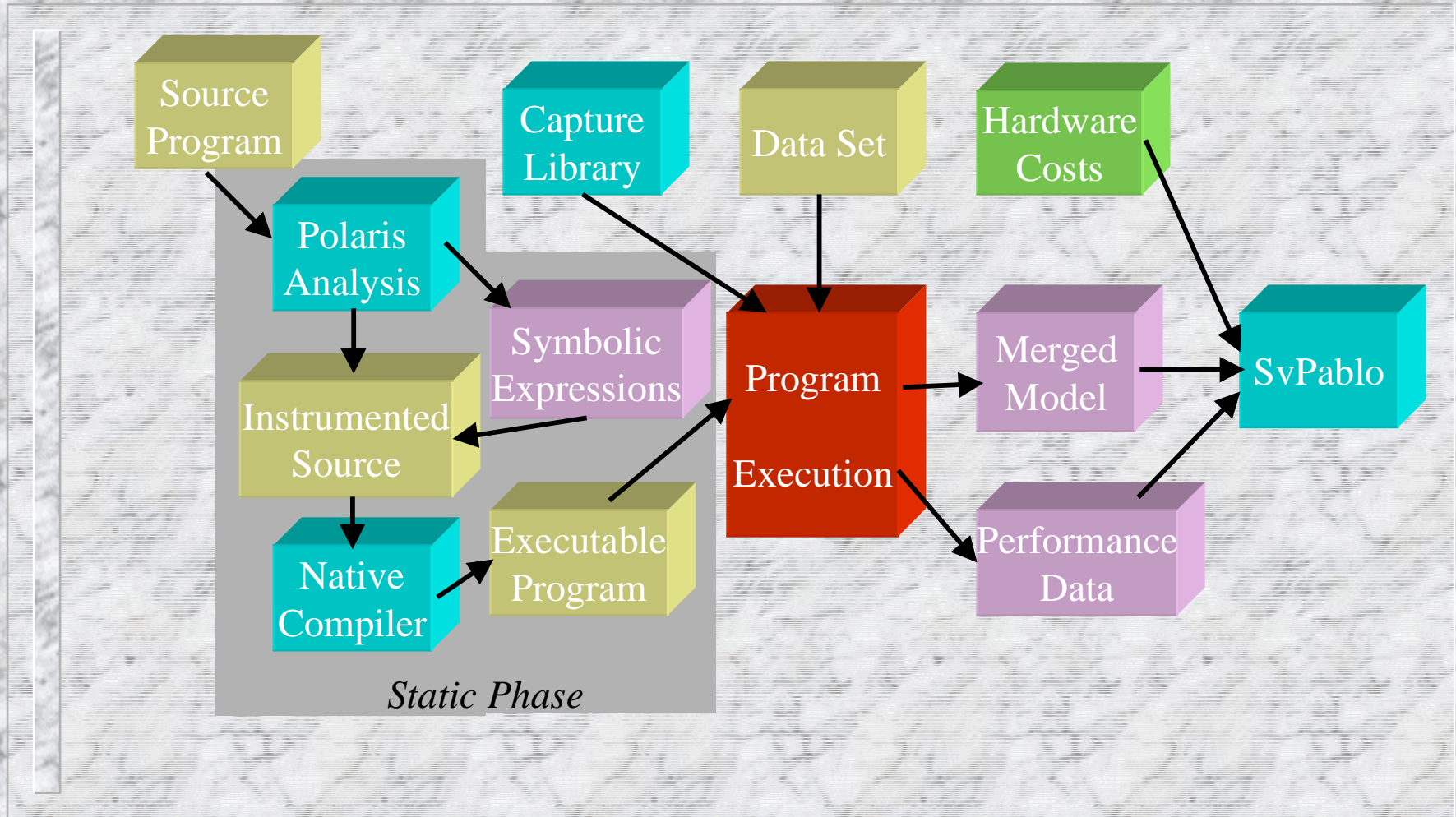
Symbolic Performance Prediction

- Two phase prediction approach
 - static traversal of compile-time AST
 - dynamic integration of measured data
- AST traversal
 - accumulate operation and memory access counts
 - aggregate for block statements
 - loops and conditional statements

Symbolic Performance Prediction

- Dynamic data integration
 - loop branch estimates
 - measured operation costs
- Performance predictions
 - instantiate array sizes (loop bounds)
 - specify number of processors
 - evaluate symbolic expression

SvPablo/Polaris Performance Prediction



Hardware Cost Variables

- Addition and Subtraction by type
- Logical Operation
- Multiplication by type
- Division by type
- Intrinsic Functions by type and class
 - (e.g., Min and Max, Trigonometric Functions, etc.)
- Load by number of bytes and rank
- Store by number of bytes and rank
- Overhead (Loop and Conditional Statement)

Symbolic Cost Expression

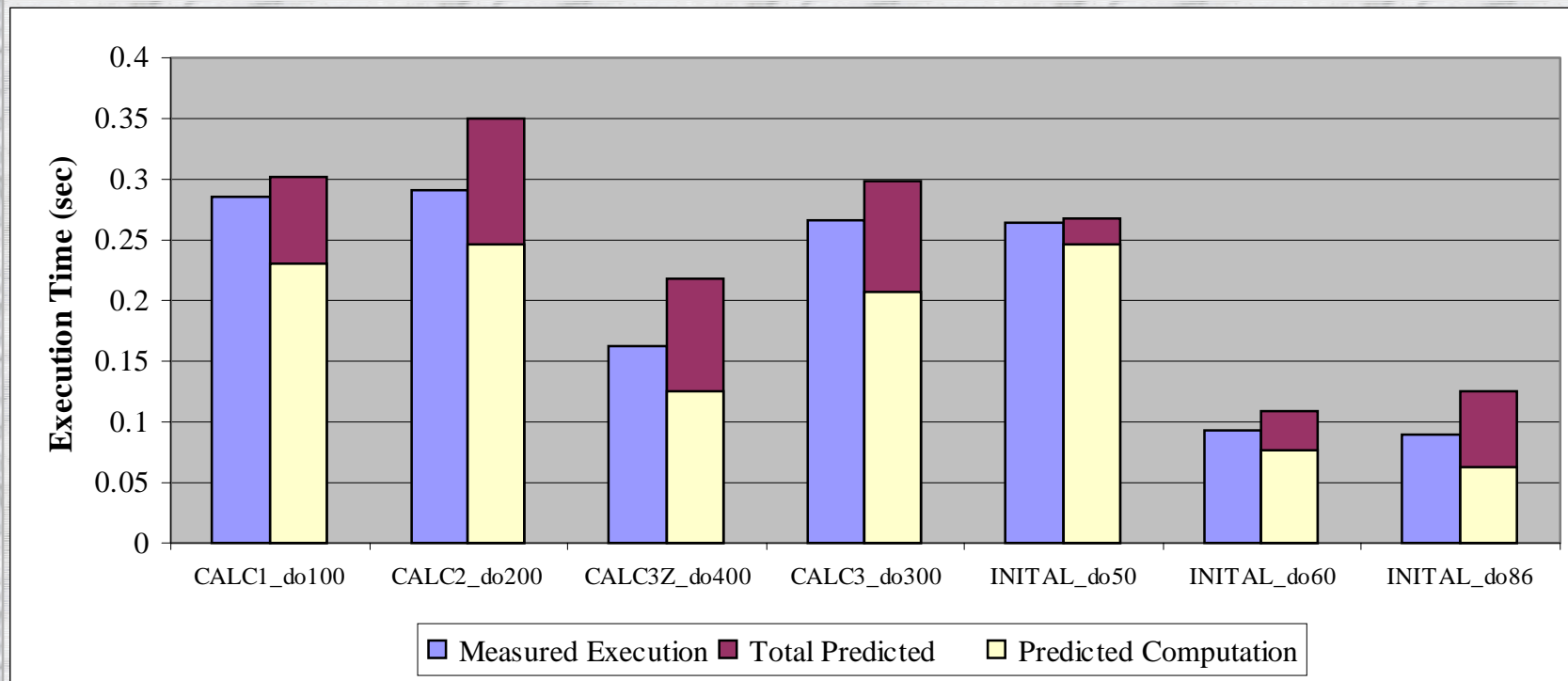
Loop 420 (HYDRO-2D)

```
DO 420 j=1,N
  DO 420 i=1,M
    C = dmin1(RP(i,j+1), RM(i,j))
    IF ( AR2(I,j) .LT. 0.0D0 ) THEN
      C = dmin1(RP(i,j), RM(i,j+1))
    ENDIF
    AR2(i,j) = C * AR2(i,j)
  420 CONTINUE
```

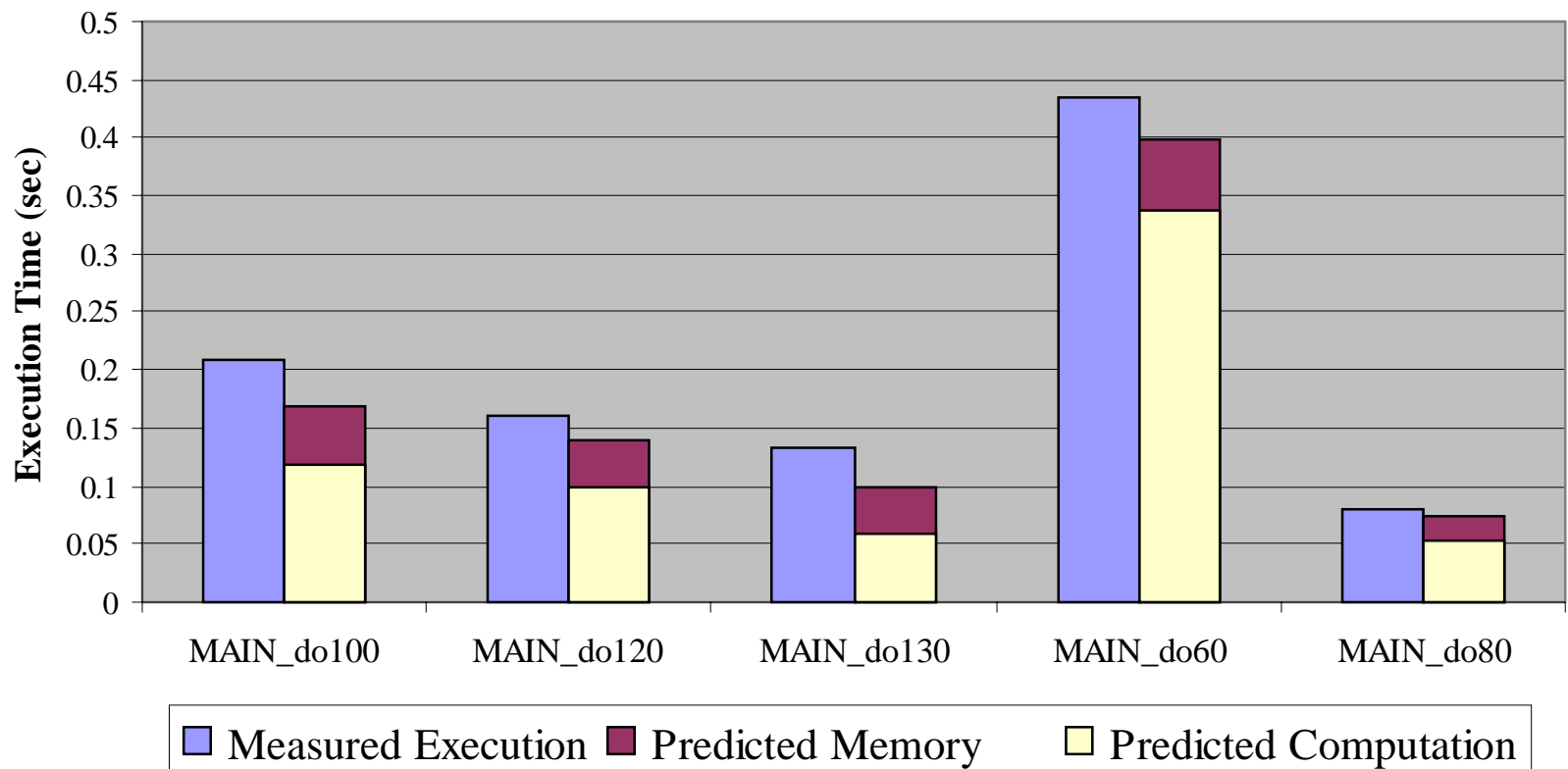
Cost =

$$\begin{aligned} & (N+1) * \text{Loop Overhead} + \\ & M*N * (1+\% \text{True}) * \text{Int Add} + \\ & M*N * \text{Dbl Mul} + \\ & M*N * (1+\% \text{True}) * \text{Intr Dbl Min} + \\ & (N+1) * (1+M*(4+2*\% \text{True})) * \\ & \quad \text{Load Scalar Int} + \\ & M*N * \text{Load Scalar Dbl} + \\ & M*N * (3+2*\% \text{True}) * \text{Load 2D Dbl} + \\ & (N+1) * \text{Store Scalar Int} + \\ & M*N * (1+\% \text{True}) * \text{Load Scalar Dbl} + \\ & M*N * \text{Store 2D Dbl} \end{aligned}$$

SWIM Performance Predictions



TOMCATV Predictions



Instrumentation Integration

- Leverage best toolkit features

- Paradyn

- dynamic object code patching
 - standard software metrics
 - hardware performance data support

- Pablo/Autopilot/Virtue

- real-time data analysis
 - flexible data metaformat
 - adaptive resource control
 - performance visualization

SvPablo Performance Browser

■ Instrumentation

- interactive
 - ANSI C
 - Fortran 77/Fortran 90
- automatic
 - PGI HPF

■ Data capture

- dynamic software statistics
- SGI R10000 counter values

SvPablo Code Browser

The screenshot displays the SvPablo Code Browser interface. At the top, there is a menu bar with 'Project', 'Instrument', 'View', and 'Help'. Below the menu bar, the 'Project Description' is 'Red Black SOR in C using MPI'. A small 'Pablo' logo is visible in the top right corner. The interface is divided into several sections:

- Source Files:** A list of source files including 'prbsor.c', 'prelax.c', and 'p_b.c'. 'prbsor.c' is currently selected.
- Performance Contexts:** A list of performance contexts including 'Origin 2000 - 16 R10K processors - 800x800', 'Power Challenge - 8 R10K Processors - 125x125', 'NoW - 8 Sun UltraSparcs - 800x800', 'NoW - 4 Sun UltraSparcs - 125x125', and 'Example: no instrumentation'. 'Example: no instrumentation' is currently selected.
- Routines in Source File:** A list of routines including 'main', 'MPI_Comm_size', 'MPI_Comm_rank', 'MPI_Get_processor_name', and 'fprintf'.
- Routines in Performance Data:** A list of routines, currently empty.
- Source File:** The path is '/home/reed/derose/DemoPablo/instal/SourceFiles/CMPLRBSOR/prbsor.c'. The source code is displayed in a text area with the following content:

```
MPI_Init( &argc, &argv );
> MPI_Comm_size( MPI_COMM_WORLD, &numprocs );
> MPI_Comm_rank( MPI_COMM_WORLD, &myid);
> MPI_Get_processor_name( processor_name, &namelen);

> fprintf(stderr, "Process %d of %d on %s\n", myid, numprocs, processor_n

    if (myid == 0)
>     errorExit = readInput( &n, &it, &rhs );
> MPI_Bcast(&errorExit, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (errorExit > 0)
    {
>     printf("Processor: %d exiting due to error: %d\n", myid, errorExit);
>     exit(1);
    }

> MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
> MPI_Bcast(&it, 1, MPI_INT, 0, MPI_COMM_WORLD);
> MPI_Bcast(&rhs, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

At the bottom of the interface, there are two buttons: 'Instrument/Clear Line' and 'View Line Data'.

“Instrumentable”
Constructs



SvPablo Code Browser

Metrics

The screenshot displays the SvPablo Code Browser interface. The main window shows a Fortran source file with a color-coded heatmap overlaying the code. The heatmap uses a color scale from yellow (low) to dark purple (high) to indicate performance metrics. A legend on the right side of the window lists 11 different metrics, each with a corresponding color swatch and numerical value. The legend is titled 'Legend: Source Code Metrics'.

Legend: Source Code Metrics

- Column 4: Loop Statistics Duration: 0.746825
- Column 5: R10K Statistics by Line Instruction Cache: 18051
- Column 6: R10K Statistics by Line L2 Cache Misses: 693
- Column 7: R10K Statistics by Line Instructions Grab: 9.24917e+07
- Column 8: R10K Statistics by Line FP Instructions: 509034
- Column 9: R10K Statistics by Line Data Cache Miss: 263825
- Column 10: R10K Statistics by Line D2 Cache Misses: 1884
- Column 11: R10K Statistics by Line MFLOPS: 6.90029

The source code in the main window includes the following lines:

```
c
  call MPI_BARRIER(MPI_COMM_WORLD, ierr)
  t1 = MPI_WTIME()
  do 10 i=1, 100
  e
    do 10 j=1, 2
    c
      call exchngl( a, nx, n, e, comid, nbottom,
      c
      call sweepid( a, f, nx, n, e, b )
      call exchngl( b, nx, n, e, comid, nbottom,
      c
      call sweepid( b, f, nx, n, e, a )
    c
      dect = diff( a, b, nx, n, e )
    c
      call MPI_Allreduce( dect, diffacc, 1, MPI_D
      c
      if (diffacc .lt. 1.0e-5) goto 20
      if (myid .eq. 0) print *, 2*it, ' Diff
  10
  continue
  if (myid .eq. 0) print *, 'Failed to con
  continue
  20
  t2 = MPI_WTIME()
```

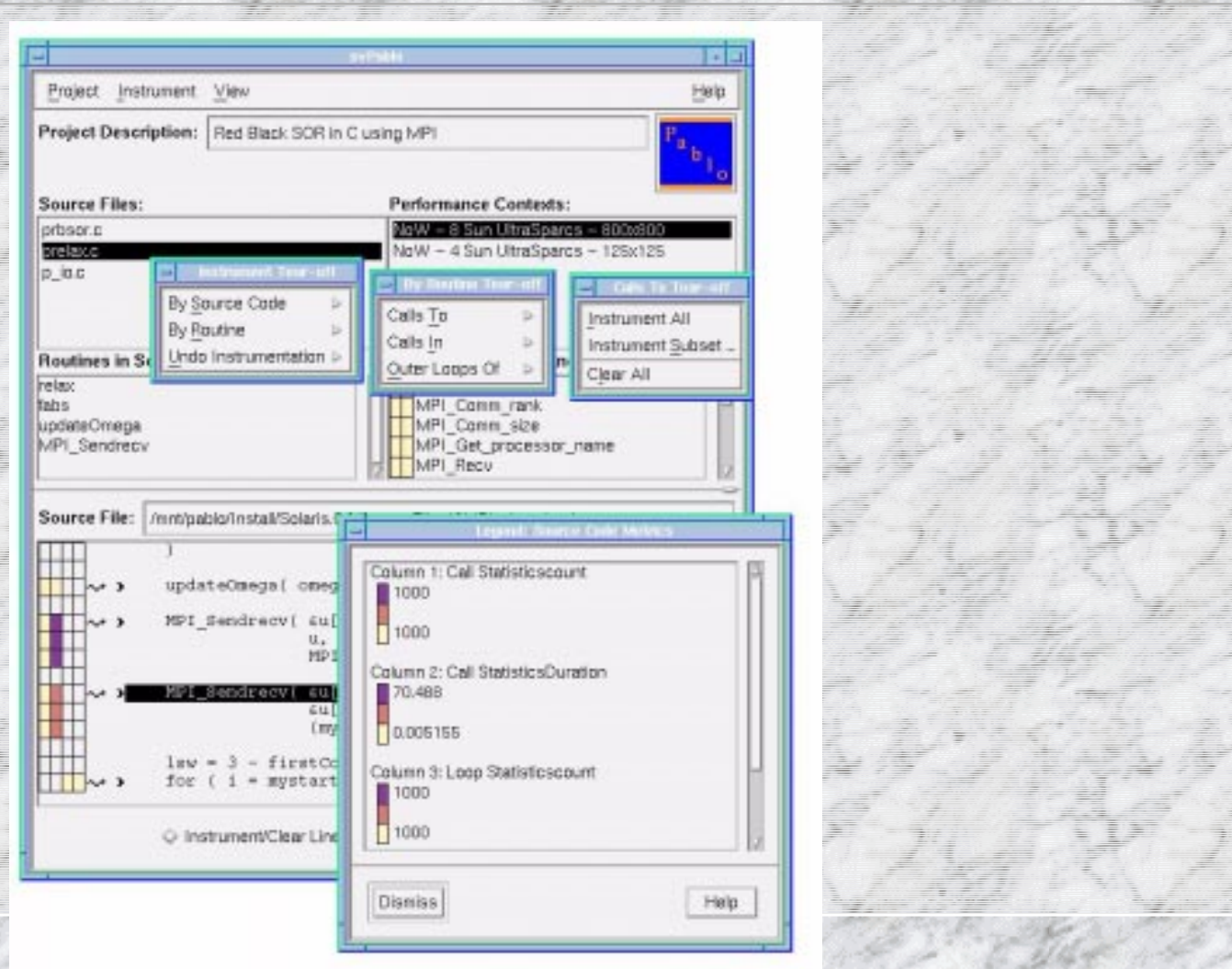

SvPablo Code Browser

The screenshot displays the SvPablo Code Browser interface for a project named "HPF - Shallow Water Equations". The interface is divided into several sections:

- Project Description:** HPF - Shallow Water Equations
- Source Files:** shallow.F
- Performance Contexts:** 200-PC 166 1024x1024 500 # R10K 2ev, Origin 32p 4096x4096 100 # R10K 2 events, NoW - Sun (8 Spikes)
- Routines in Source File:** 101.76112000
- Routines in Performs:** A color-coded legend for routines: calc2 (red), calc3 (orange), shallow (yellow), and optim (purple).
- Source File:** /home/steve/rose/TarGoo/Pablo/SvPablo/Install/SourceFiles/HPF_SHALLOW/shallow.F
- Code:** A block of Fortran code with a heatmap overlay. The heatmap uses colors to represent execution time per line, with red indicating the highest time and purple the lowest. The code includes comments like "CHIEFS DO CLOBARS" and "PERIODIC CONTINUATION".
- Statistics by Line Message Receive Duration:** 66.40716000

At the bottom of the window, there are two radio buttons: "Instrument/Class Line" (selected) and "View Line Data".

SvPablo Code Browser



SvPablo Code Browser

The screenshot displays the SvPablo Code Browser interface. The main window is titled "Performance Data: Loop Statistics // LOOP". It shows the following details:

- Tasks: 0 .. 15
- File Name(s): prbsor.c
- Routine Name: main()
- Line Number: 194
- Source Code Fragment:

```
for (i=1; i<=it; i++) { relax(n, &omega, f, u, &mynorm); updateOmega(&omega);  
  
MPI_Sendrecv( &u[myend*n], n, MPI_DOUBLE, botton, (myid+1)*blocksize,  
u, n, MPI_DOUBLE, top, myid * blocksize,  
MPI_COMM_WORLD, &status );
```

Below the code fragment is a performance data table:

Field Name	Mean	Max	Min	Std Dev		
	Value	Task	Value	Task		
Count	1.000000	0	1.000000	0	0.00000000	
Seconds	13.810467	14.228165	2	13.155972	15	0.31973676
Exclusive Seconds	0.016930	0.021476	4	0.014443	15	0.00148088
Loads Issued	887.500000	2232.000000	4	0.000000	0	838.65740920
Stores Issued	1544.000000	4070.000000	0	0.000000	0	2500.00000000

At the bottom of the window, there is a checkbox labeled "View detailed performance data" which is currently unchecked. The interface also includes "OK" and "Help" buttons.

SvPablo Language Transparency

■ Meta-metaformat for performance data

- language defined by line and byte offsets
- metrics defined by mapping to offsets
- SDDF records
 - performance mapping information
 - performance measurements

■ Result

- language independent performance browser
- mechanism for scalability model integration

SvPablo Language Transparency

- Three SDDF record types

- *configuration*

- definition of performance metric set
- set may contain multiple metrics

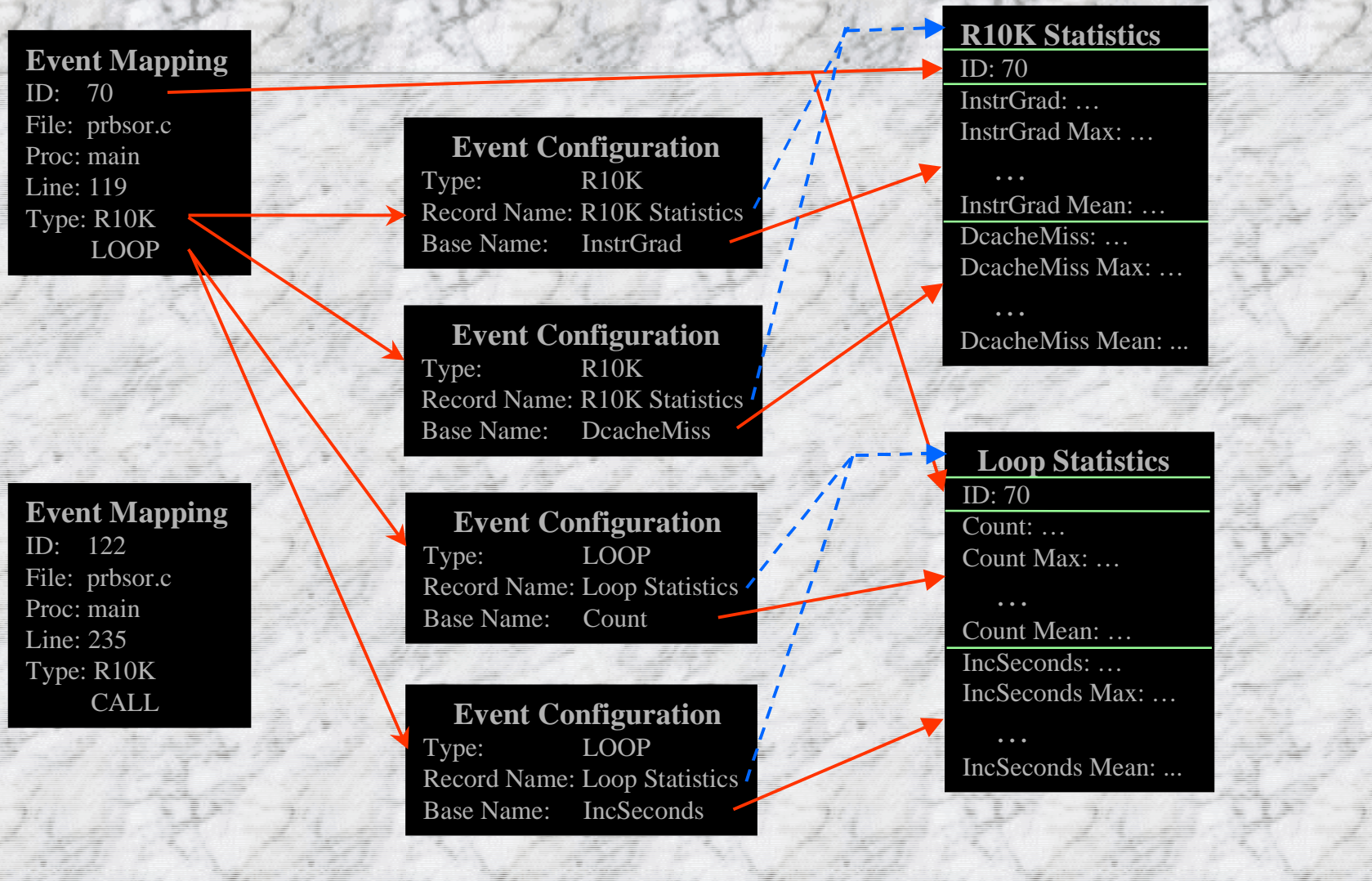
- *mapping*

- source code location
- “pointers” to actual set of performance statistics

- *statistic*

- one for each performance metric
- contain actual performance data

SvPablo SDDF Record Hierarchy



SvPablo Directions

■ Infrastructure extensions

- explore port to Windows NT
- add Pentium counter support (Paradyn)

■ Scalability extensions

- extend symbolic prediction model
- integrate predictions using SvPablo transparency