Performance Measurement and Prediction

Luiz DeRose

Dan Reed

University of Illinois

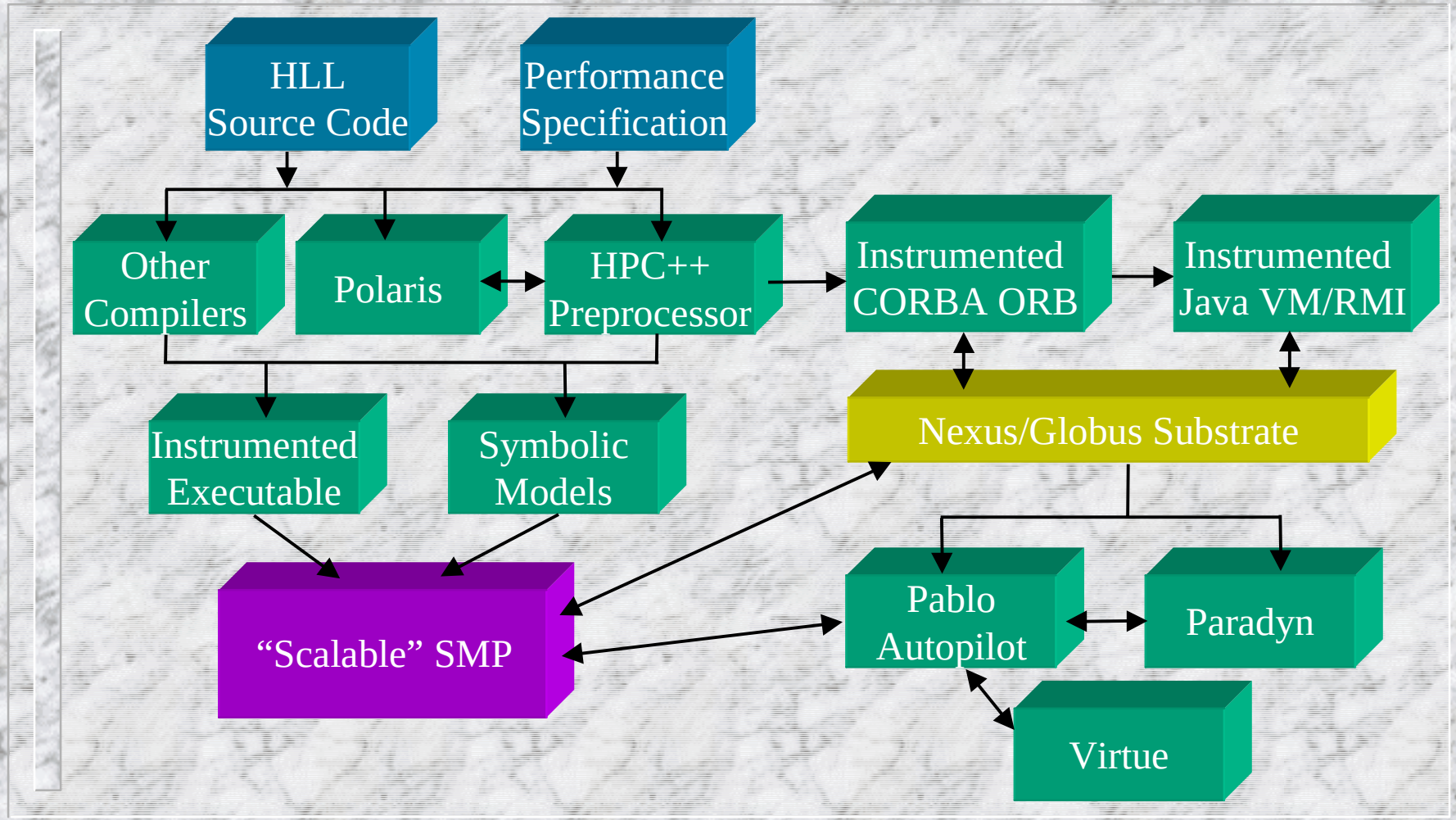
Presentation Outline

- 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

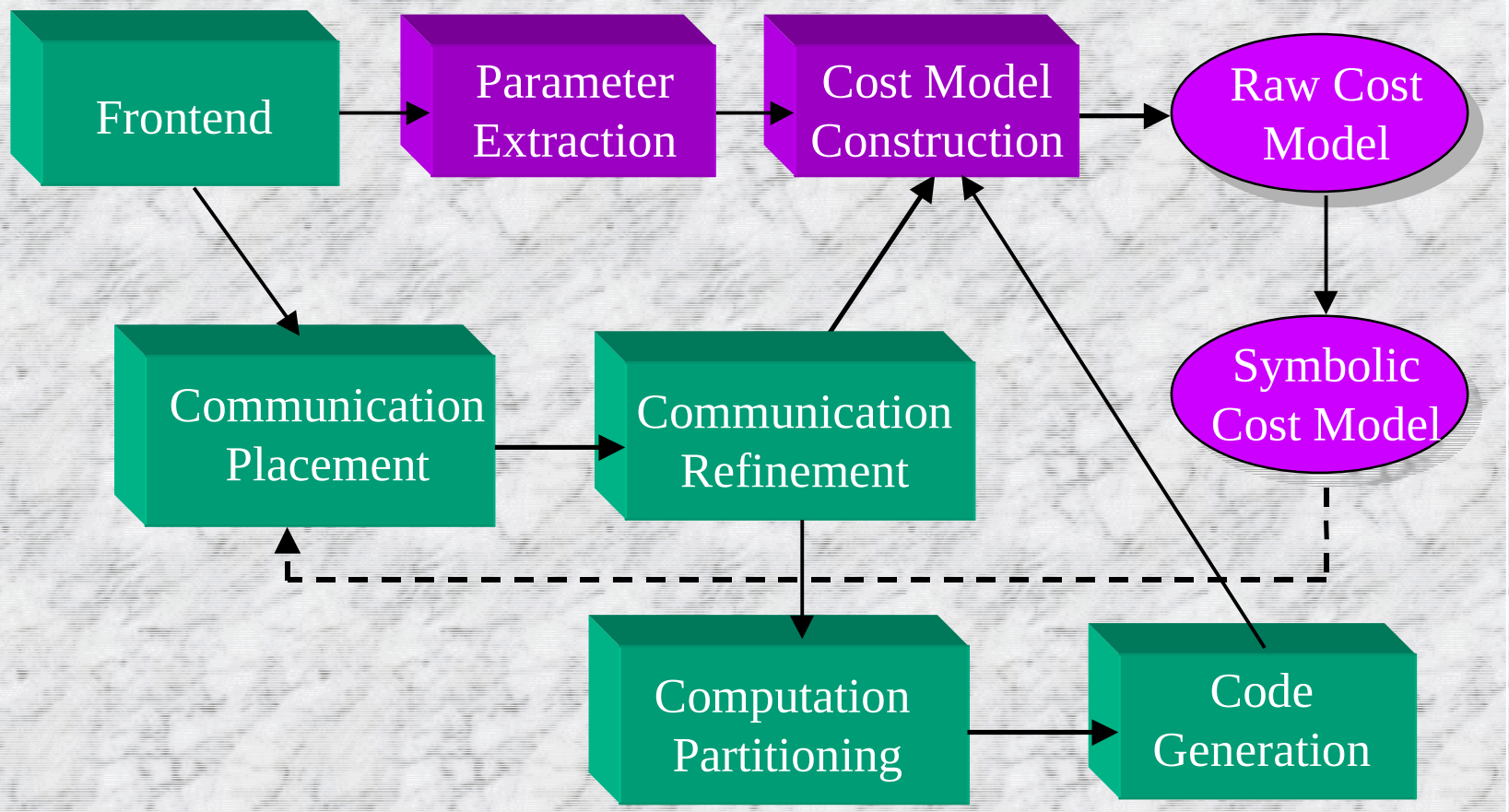
High-Level Language Integration



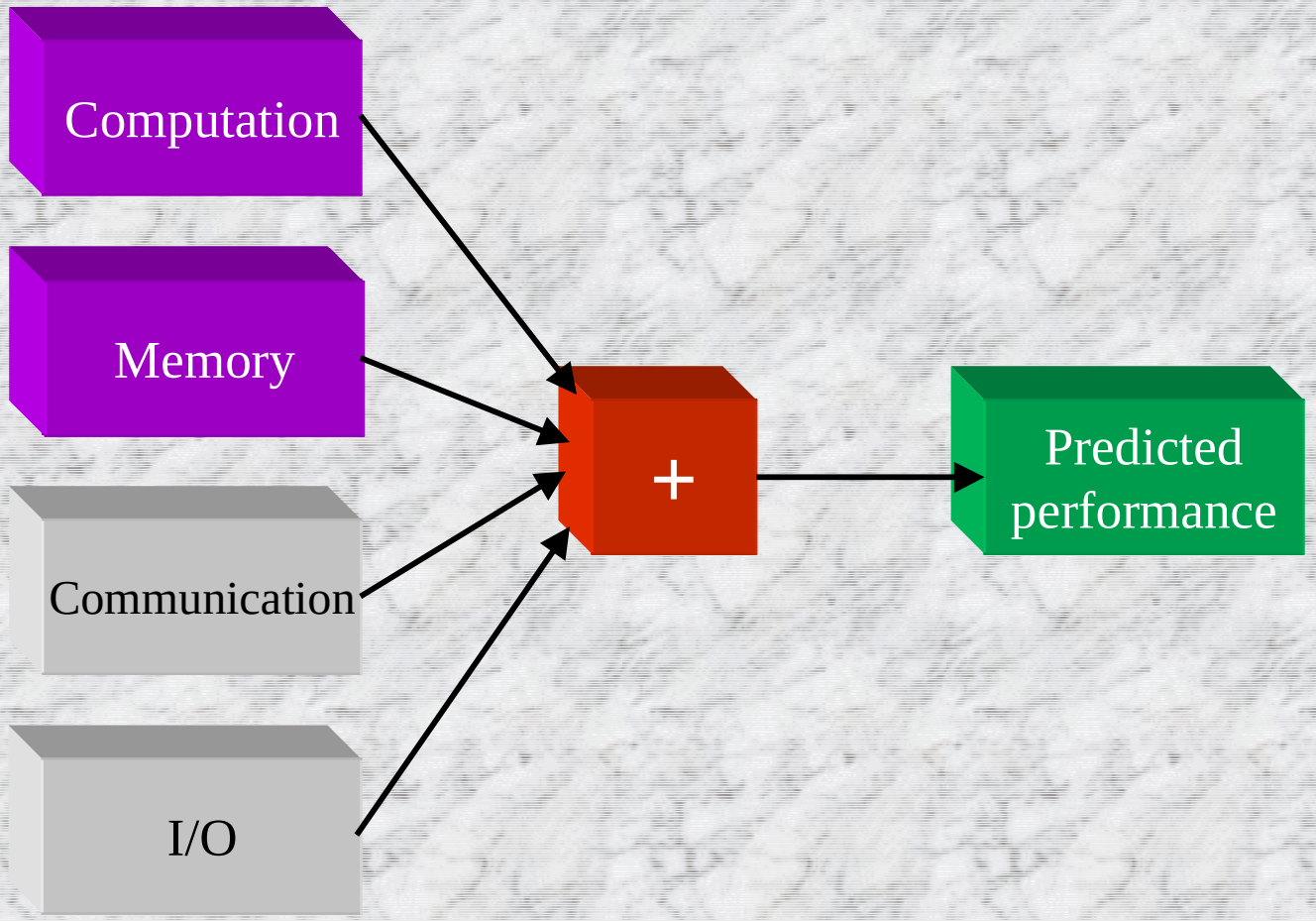
Symbolic Performance Prediction

- 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



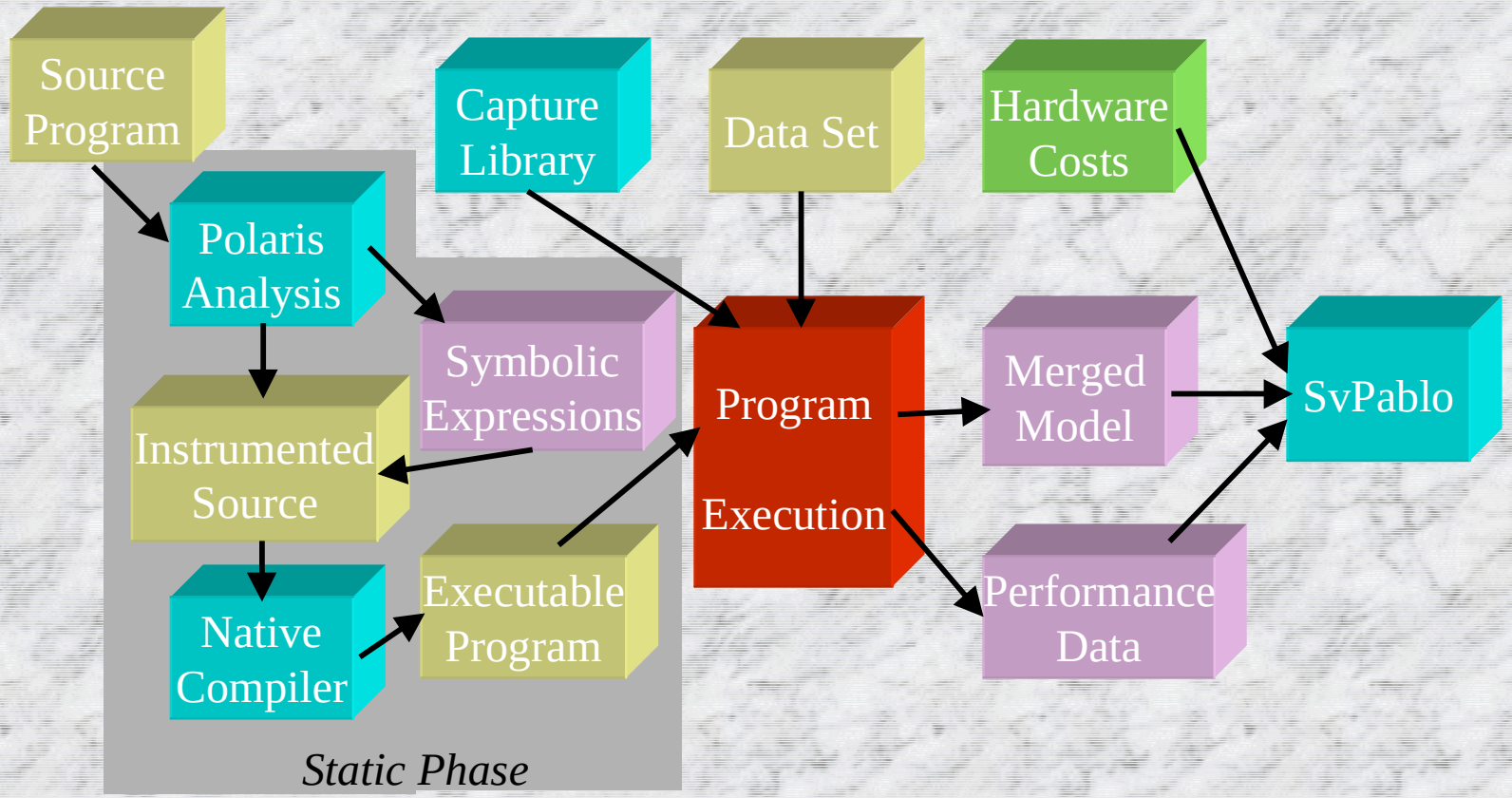
Symbolic Performance Prediction

- 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

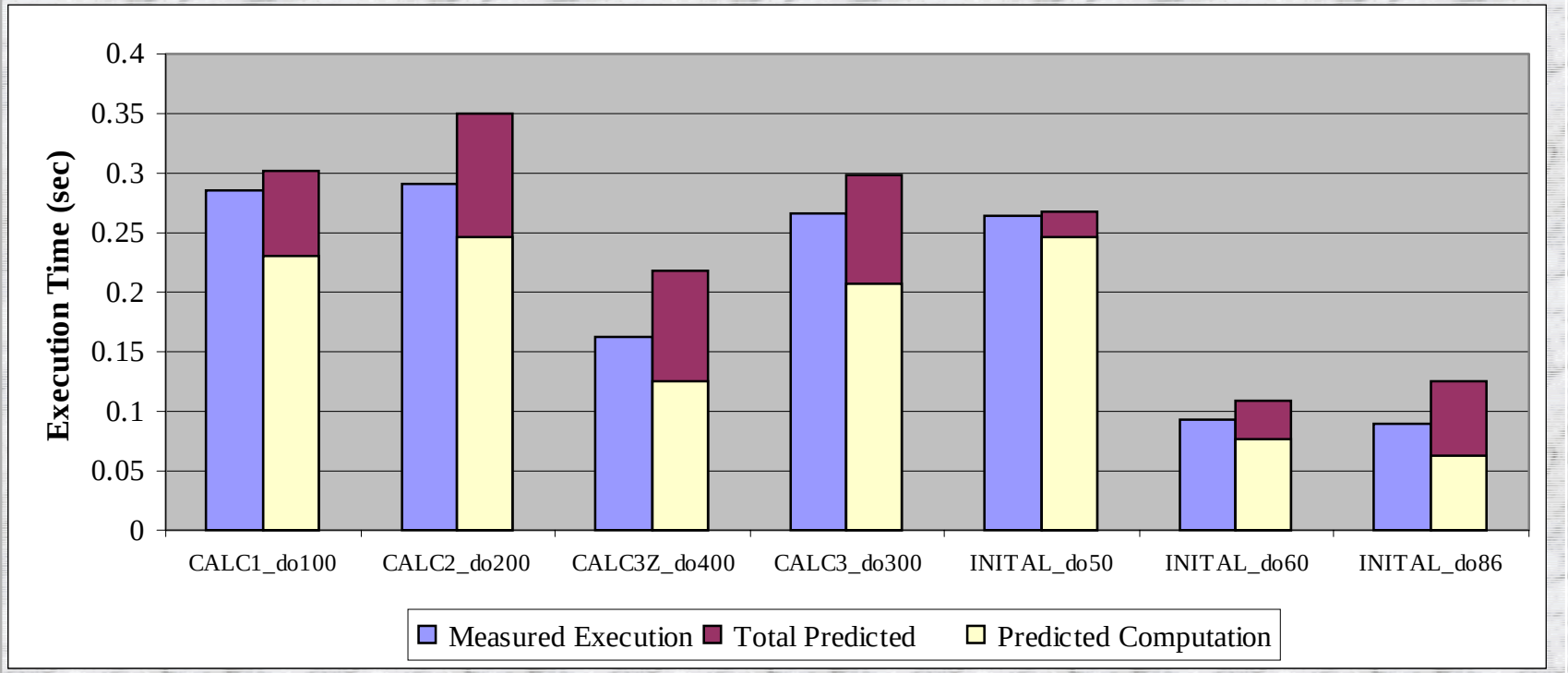
Loop 420 (HYDRO-2D)

```
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
```

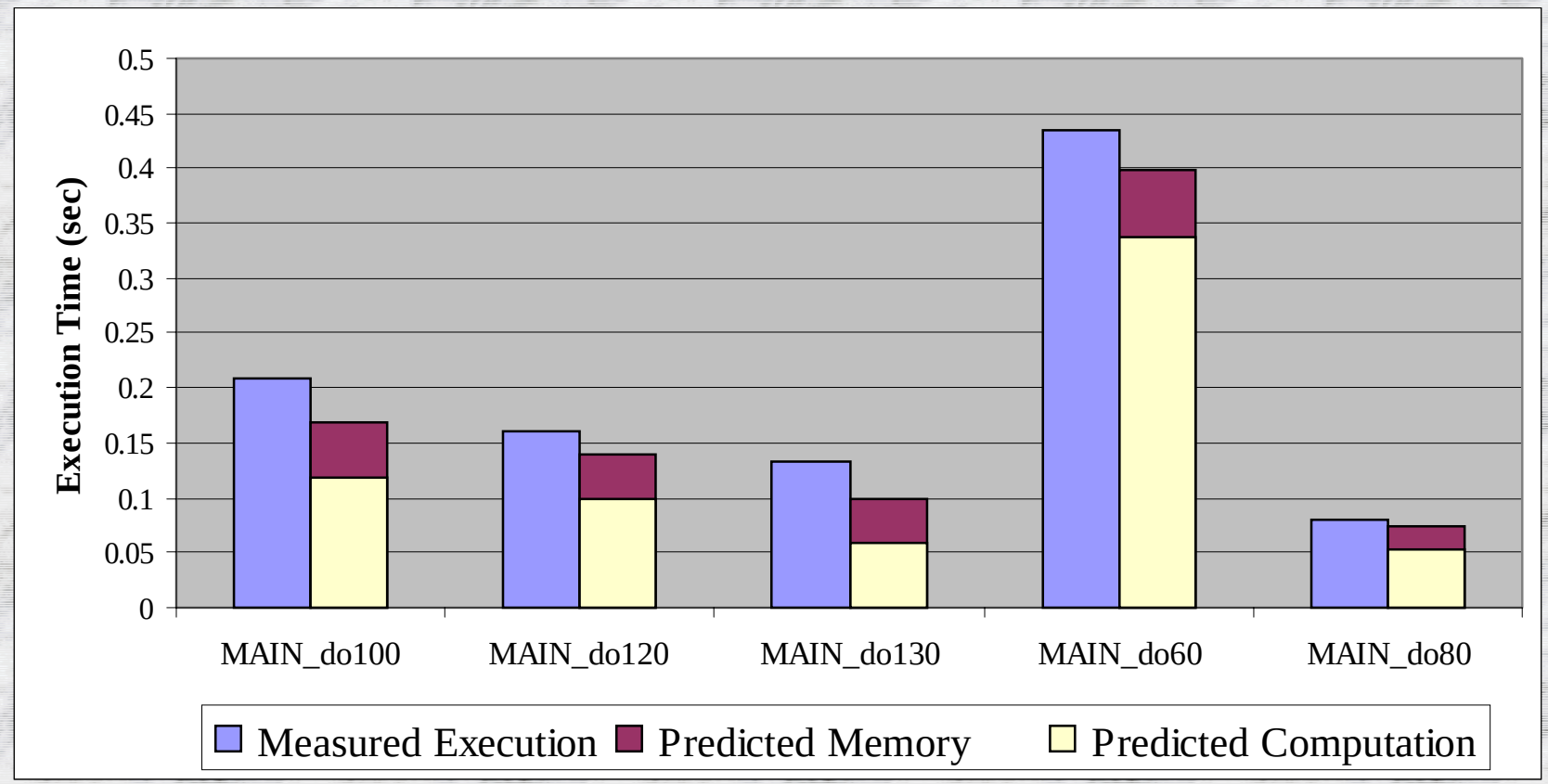
Cost =

(N+1) * Loop Overhead +
M*N * (1+%True) * Int Add +
M*N * Dbl Mul +
M*N *(1+%True)* Intr Dbl Min+
(N+1) * (1+M*(4+2*%True) *
Load Scalar Int +
M*N * Load Scalar Dbl +
M*N *(3+2*%True)*Load 2D Dbl +
(N+1) * Store Scalar Int +
M*N *(1+%True)*Load Scalar Dbl+
M*N * Store 2D Dbl

SWIM Performance Predictions



TOMCATV Predictions



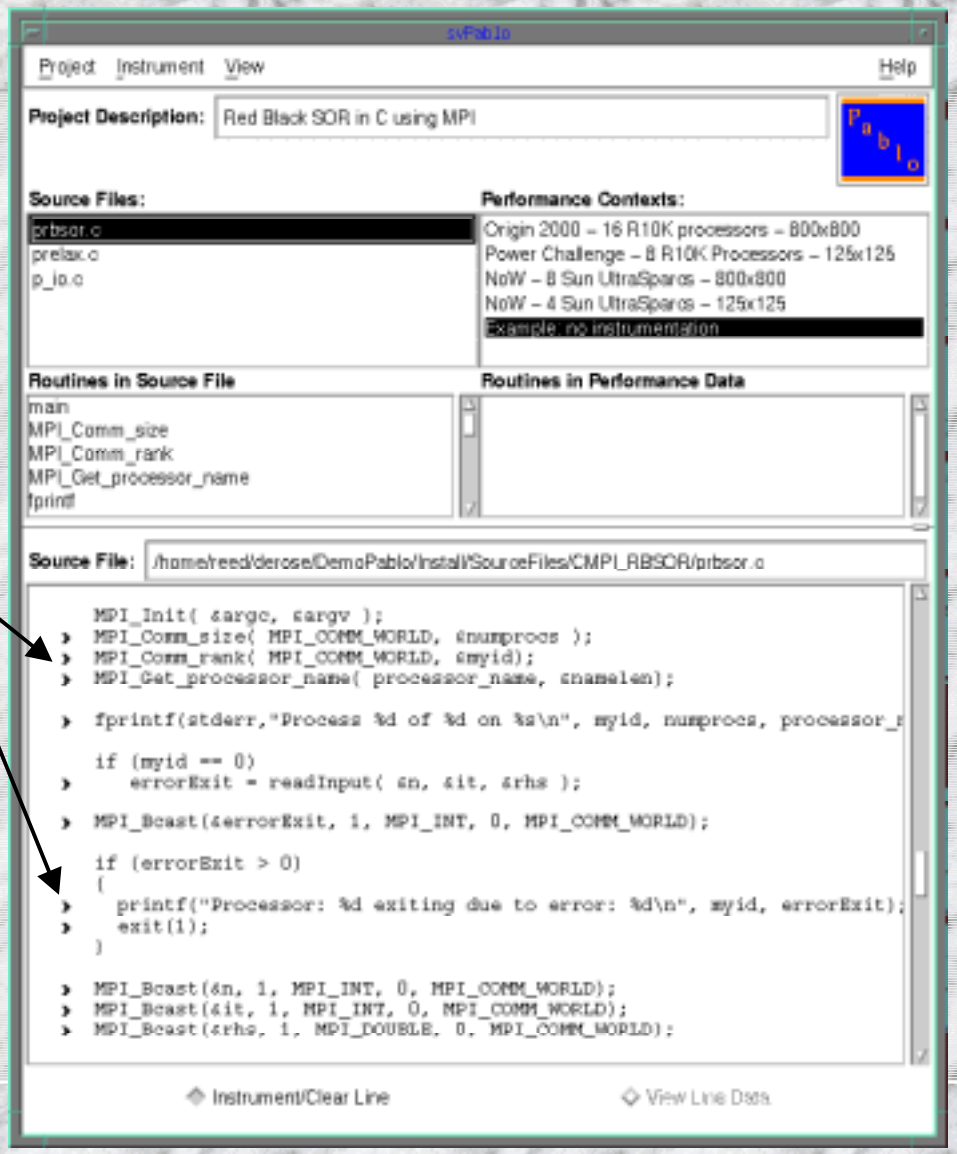
Instrumentation Integration

- 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

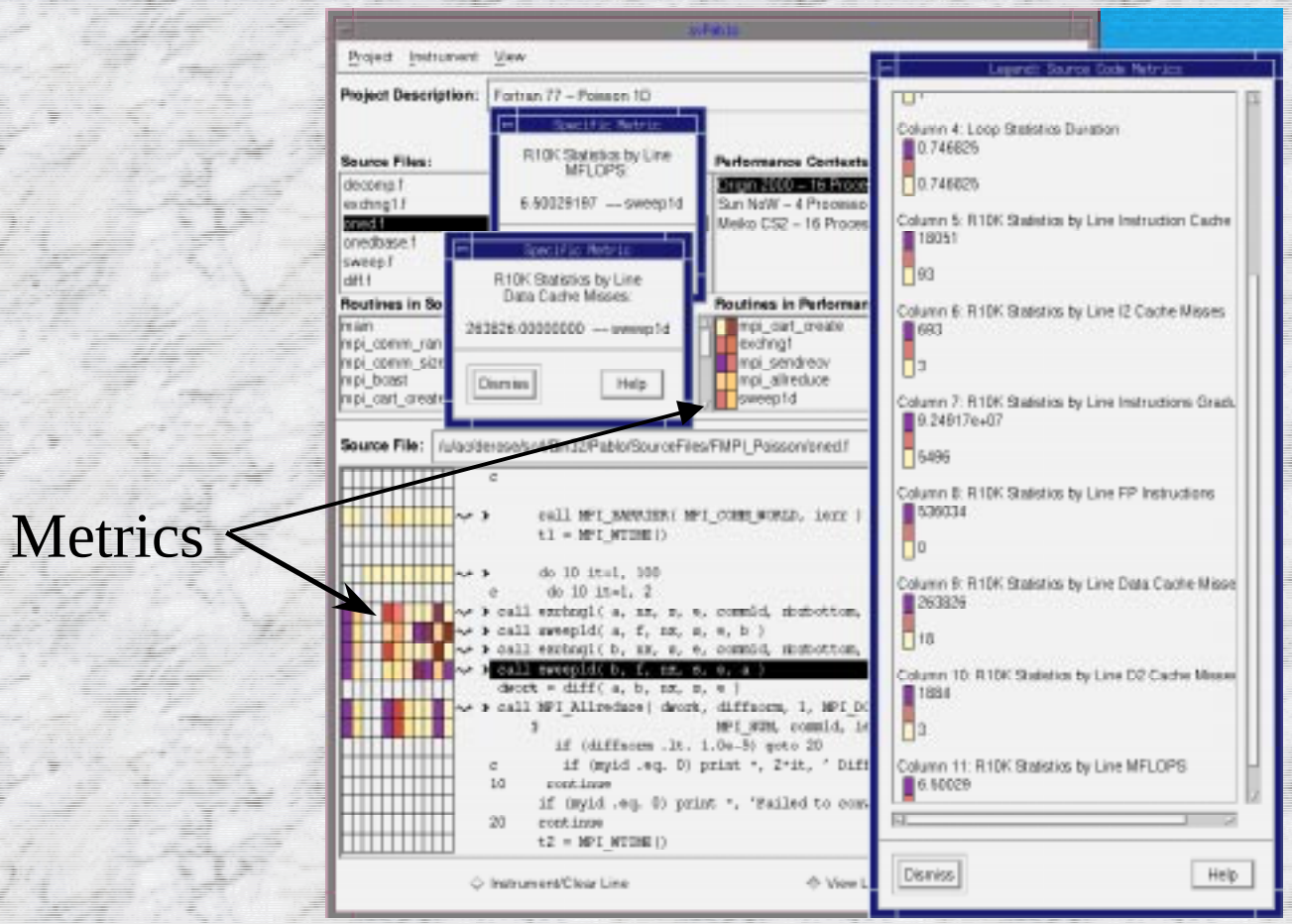
- Instrumentation
  - interactive
    - ANSI C
    - Fortran 77/Fortran 90
  - automatic
    - PGI HPF
- Data capture
  - dynamic software statistics
  - SGI R10000 counter values

SvPablo Code Browser

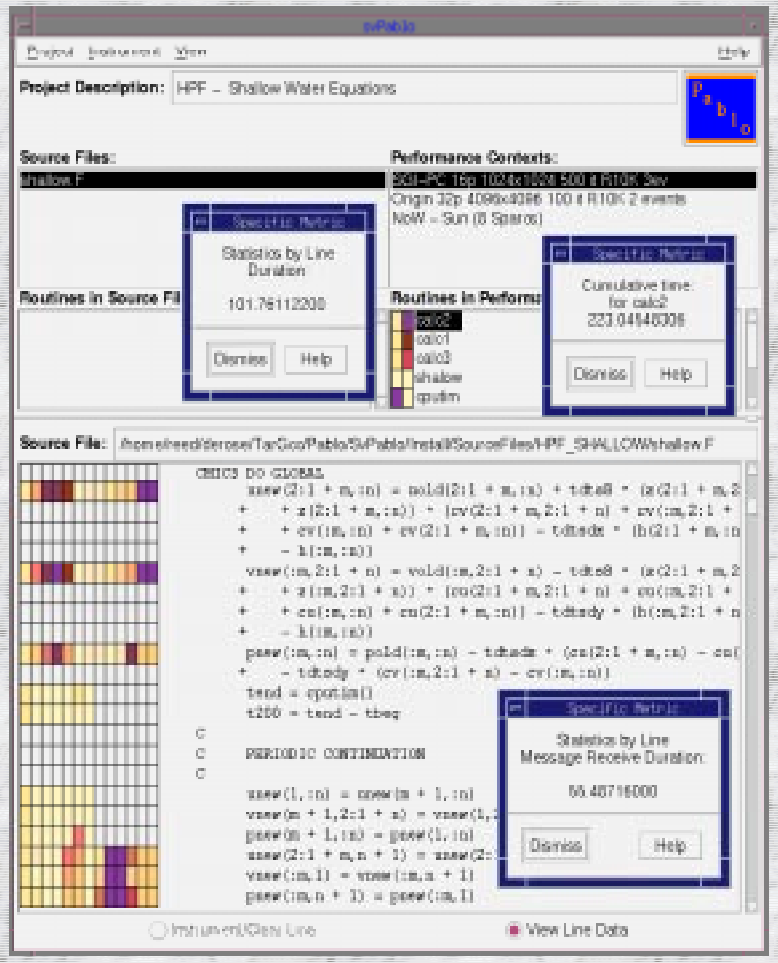


"Instrumentable" Constructs

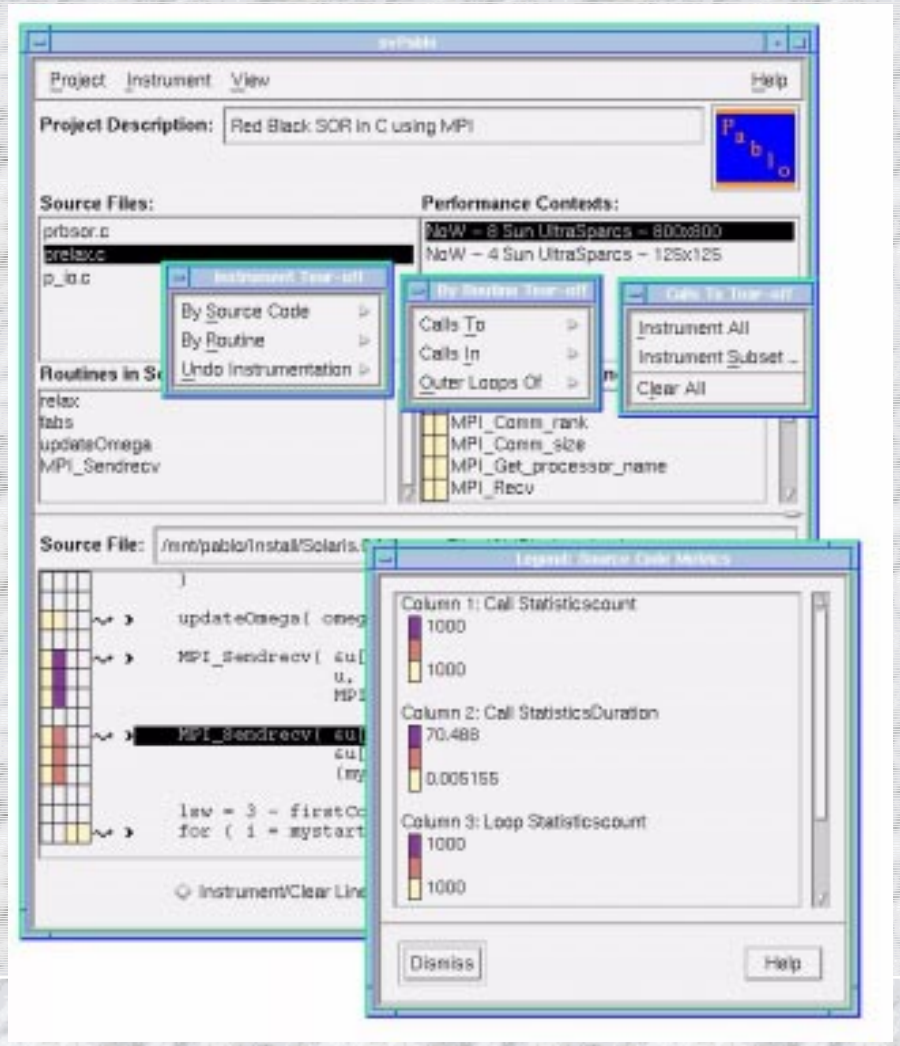
SvPablo Code Browser



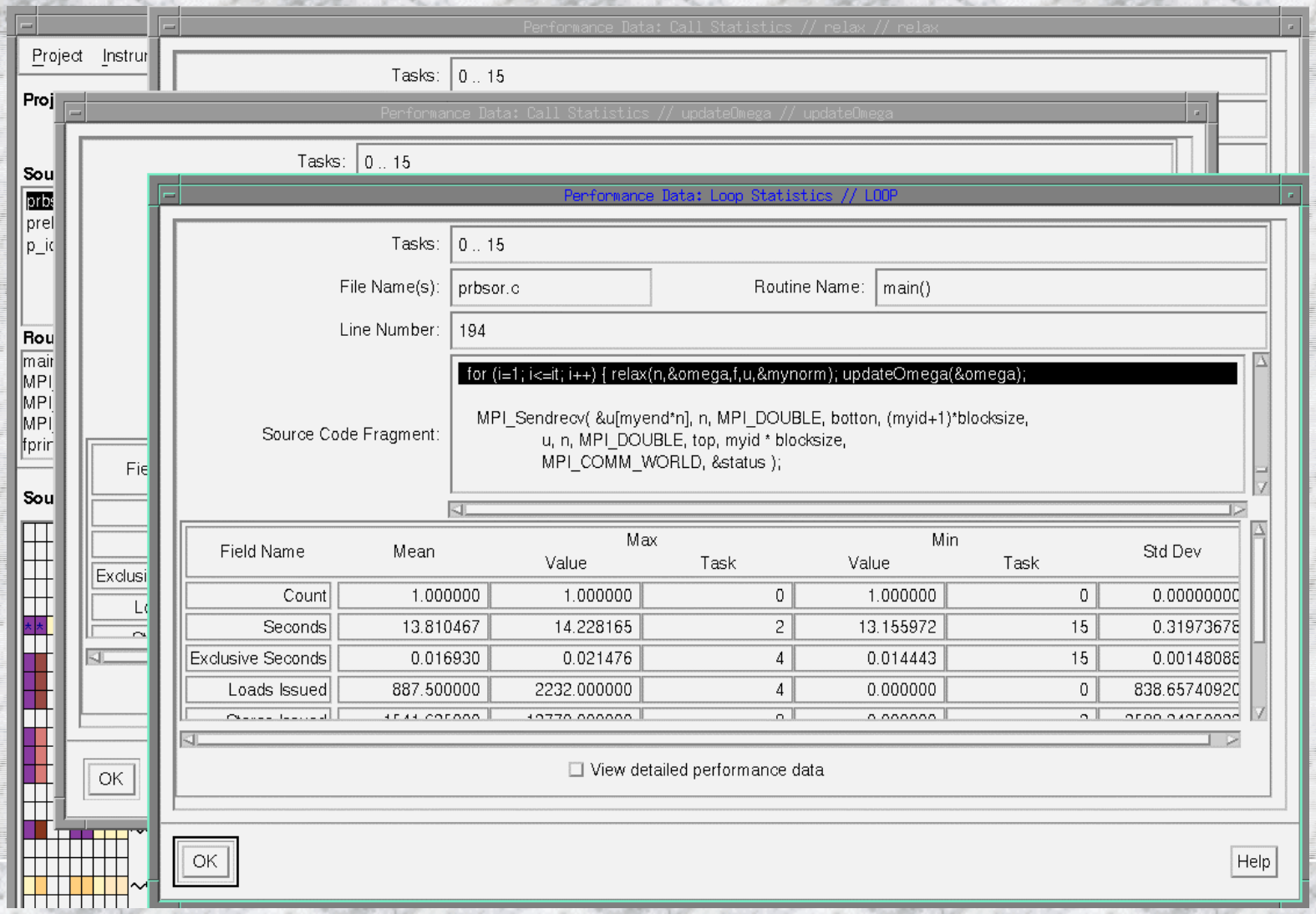
SvPablo Code Browser



SvPablo Code Browser



SvPablo Code Browser



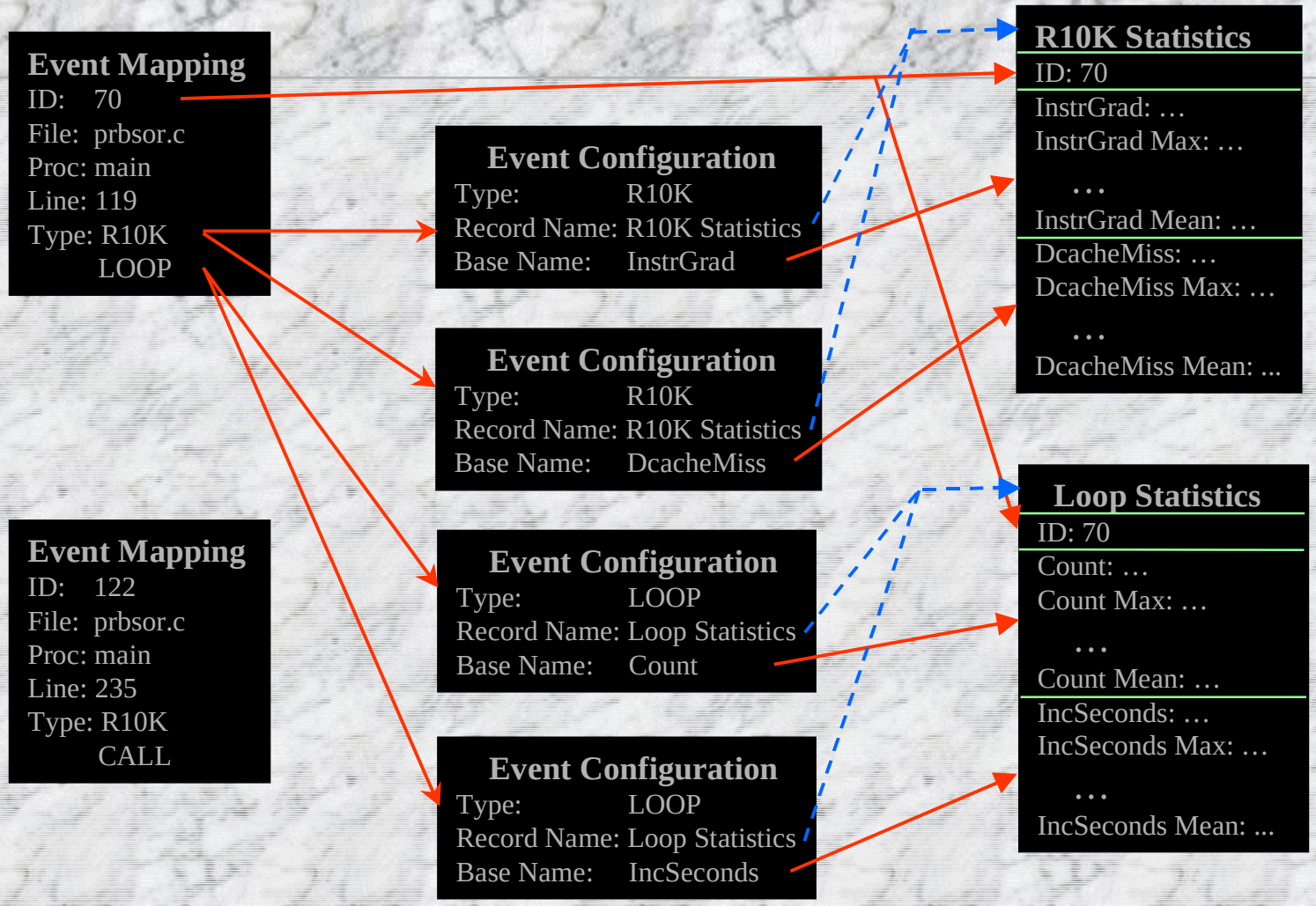
SvPablo Language Transparency

- 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