A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates

Kathie Lindlan
Computer and Information Science
University of Oregon

Supercomputing 2000

Credits

Bernd Mohr

Zentralinstitut fur Angewandte Mathematik
Forschungszentrum Julich

Janice Cuny
Allen D. Malony
Sameer Shende

Computer and Information Science
University of Oregon

Reid Rivenburgh

Computer Research and Applications Group

Craig Rasmussen

Advanced Computing Laboratory
Los Alamos National Laboratory

Outline

- **Context**
- Program Database Toolkit (PDT)
- Applications Using PDT
- PDT Availability

Context

- High-performance computing environments
- Infrastructure for large-scale scientific computation
  - Object-oriented frameworks
  - Scalable run-time systems
  - Software component architectures

Problem

- Automatic instrumentation of C++ source code

Difficulties with C++ Templates

- Template parsing is a challenge for some compilers
- Output from compiler front end may not include template info
- Templates may be instantiated at pre-link time
- Original template must be instrumented

Outline

- Context
- **Program Database Toolkit (PDT)**
- Applications Using PDT
- PDT Availability

Program Database Toolkit (PDT)

Edison Design Group (EDG) Front End

- Parses a C++ source file
- Creates an intermediate language (IL) tree

IL Analyzer

- Processes the intermediate language tree
- Creates a structured “program database” file

Program Database (PDB)

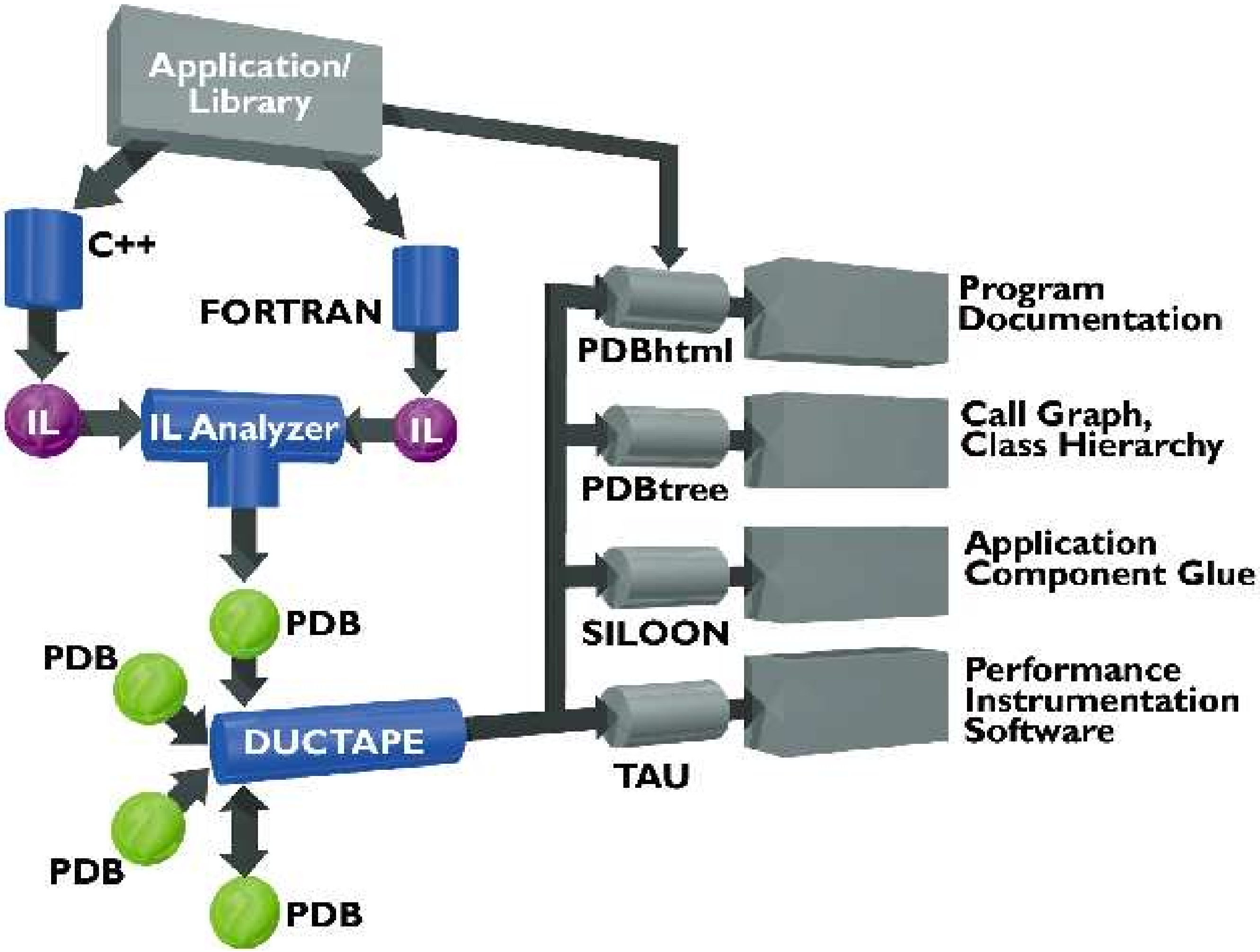
- Can be easily and efficiently read by a programming or a scripting language

DUCTAPE

(C++ program Database Utilities and Conversion Tools APplication Environment)

- Processes and merges PDB files
- Provides object-oriented, C++ interface to the PDB file

PDT Architecture



Program Database (PDB)

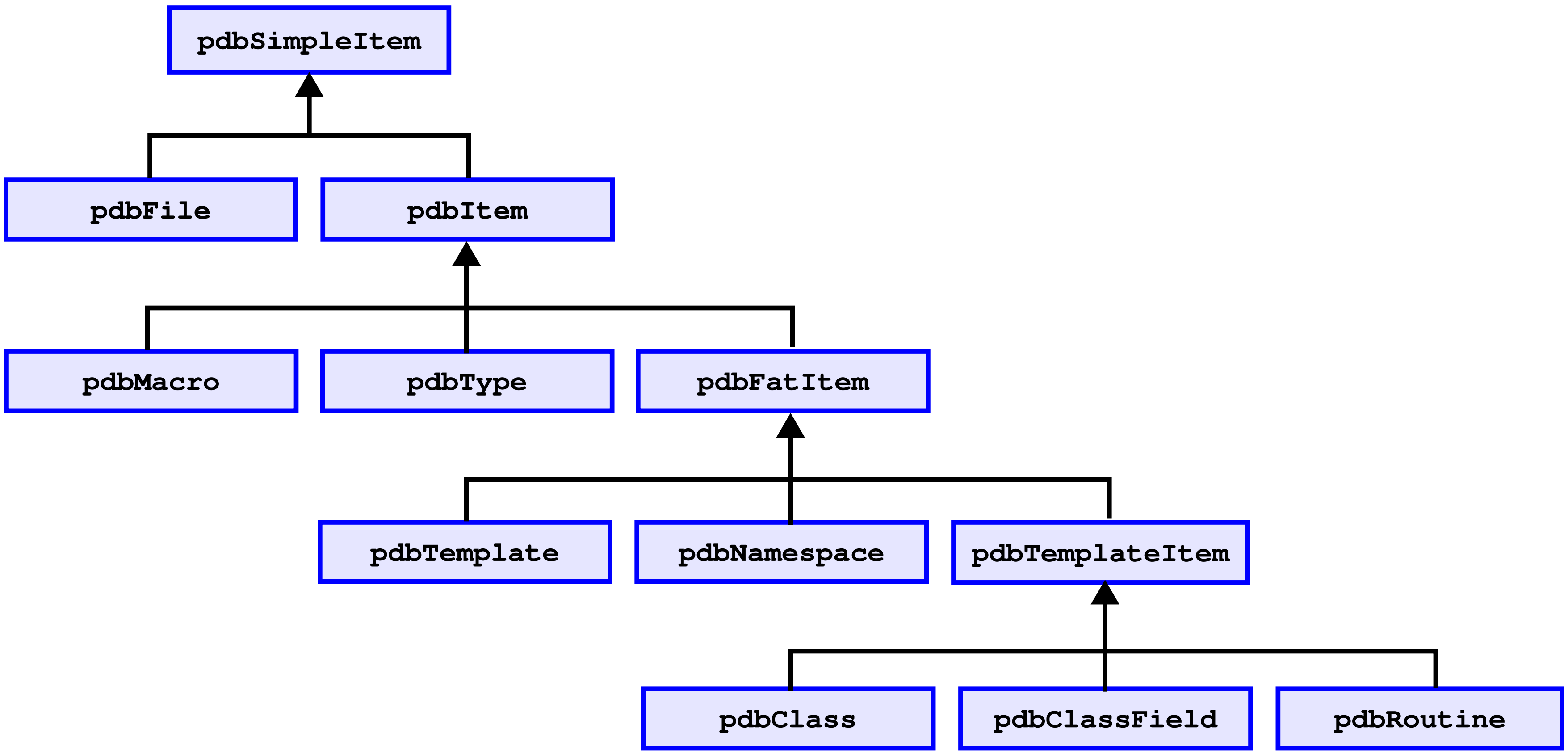
PDB comprises the "interface" of the original source:

- Items describing functions, classes, other programming language constructs
- Information on template instantiations

Item Type	Attributes of Item
all ITEMs	source position
HEADER	<PDB 1.0>
SOURCE FILES	files included by source file
ROUTINES	template from which instantiated, parent class or namespace, access mode, signature, functions called, characteristics specifying linkage, storage class, virtuality, *etc*.
CLASSES	template from which instantiated, parent class or namespace, access mode, direct base classes, friend classes and functions, characteristics, member functions, information on other members, including access, kind, and type
TYPES	parent class or namespace, access mode, various characteristics, depending on type: *e.g.*, for function types, return type, parameter types, presence of ellipsis, and exception class IDs
TEMPLATES	parent class or namespace, access mode, kind, text of template
NAMESPACES	members of namespace or alias
MACROS	kind, text of macro

DUCTAPE

The **DUCTAPE** C++ library provides access to **PDB** files via the class hierarchy:



Outline

- Context
- Program Database Toolkit (PDT)
- **Applications Using PDT**
- PDT Availability

Applications Using PDT

- Simple Utility
- Source-to-Source Translation
- Code Generation
- Static Analysis and Documentation Generation

Simple Utility

Excerpt from **DUCTAPE**'s ***pdbtree*** utility

```
static void printFuncTree(const pdbRoutine *r, int level) {
    r->flag(ACTIVE);
    pdbRoutine::callvec c = r->callees();
    for (pdbRoutine::callvec::iterator it=c.begin(); it!=c.end(); ++it) {
      const pdbRoutine *rr = (*it)->call();
      if ( level != 0 || rr->callees().size() ) {
        cout << setw((level-1)*5) << "";
        if ( level ) cout << "`--> ";
        cout << rr->fullName();
        if ( (*it)->isVirtual() ) cout << " (VIRTUAL)";
        if ( rr->flag() == ACTIVE ) {
          cout << " ..." << endl;
        } else {
          cout << endl;
          printFuncTree(rr, level+1);
        }
      }
    }
    r->flag(INACTIVE);
}
```

Source-to-Source Translation

PDT assists in automatic instrumentation of C++ source for

TAU (Tuning and Analysis Utilities) Instrumentor:

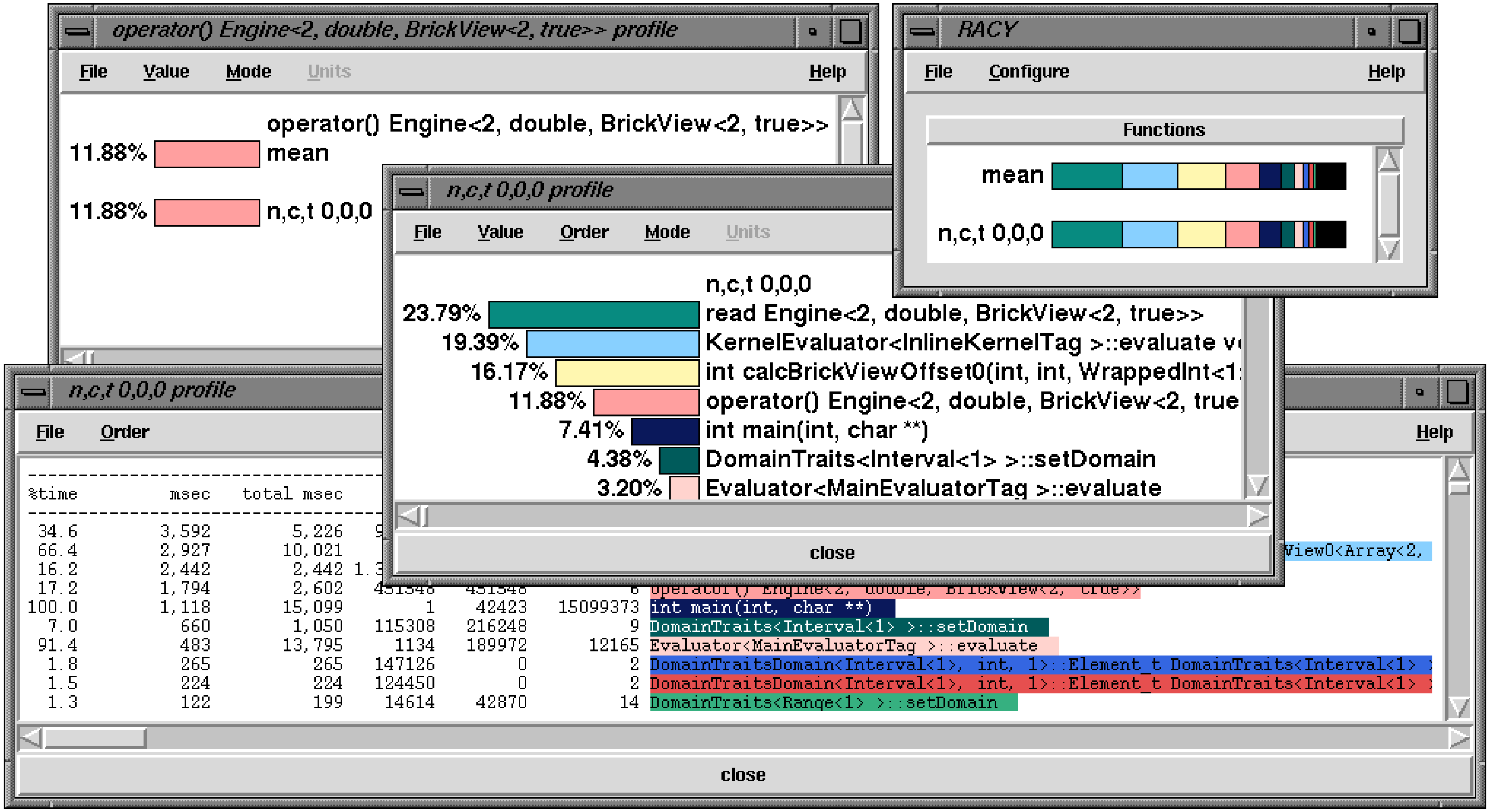
- Iterates through PDB list of functions and templates
- Generates and inserts TAU profiling macros in source code
- Application links with TAU library to create profile data files

```
template <class T>
class vector {
public:
        vector (int size) {
                TAU_PROFILE (“vector::vector ( )”, CT (*this), TAU_USER);
        ...

        }
}
```

Source-to-Source Translation (2)

TAU automatically instrumented POOMA 's Krylov solver using **PDT**

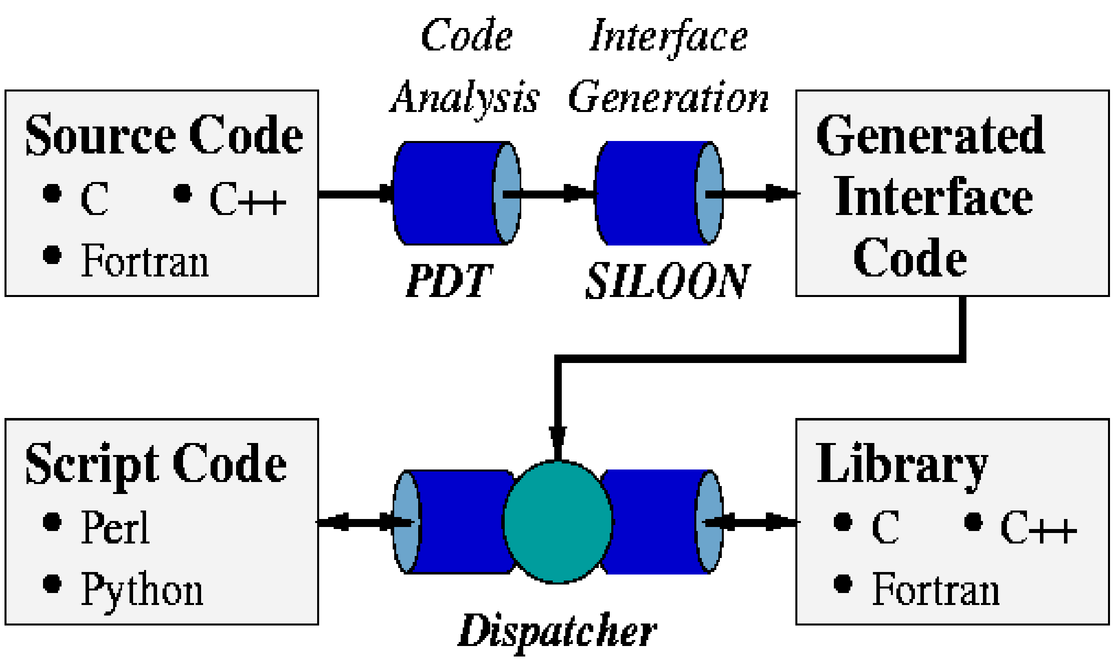


Code Generation

PDT assists in generating glue and skeleton code for

SILOON (Scripting Interface Language for Object-Oriented Numerics):

- Automates access to C++ and Fortran libraries from Python and Perl
- Enables rapid prototyping of scientific codes using scripting languages



Static Analysis and Documentation Generation

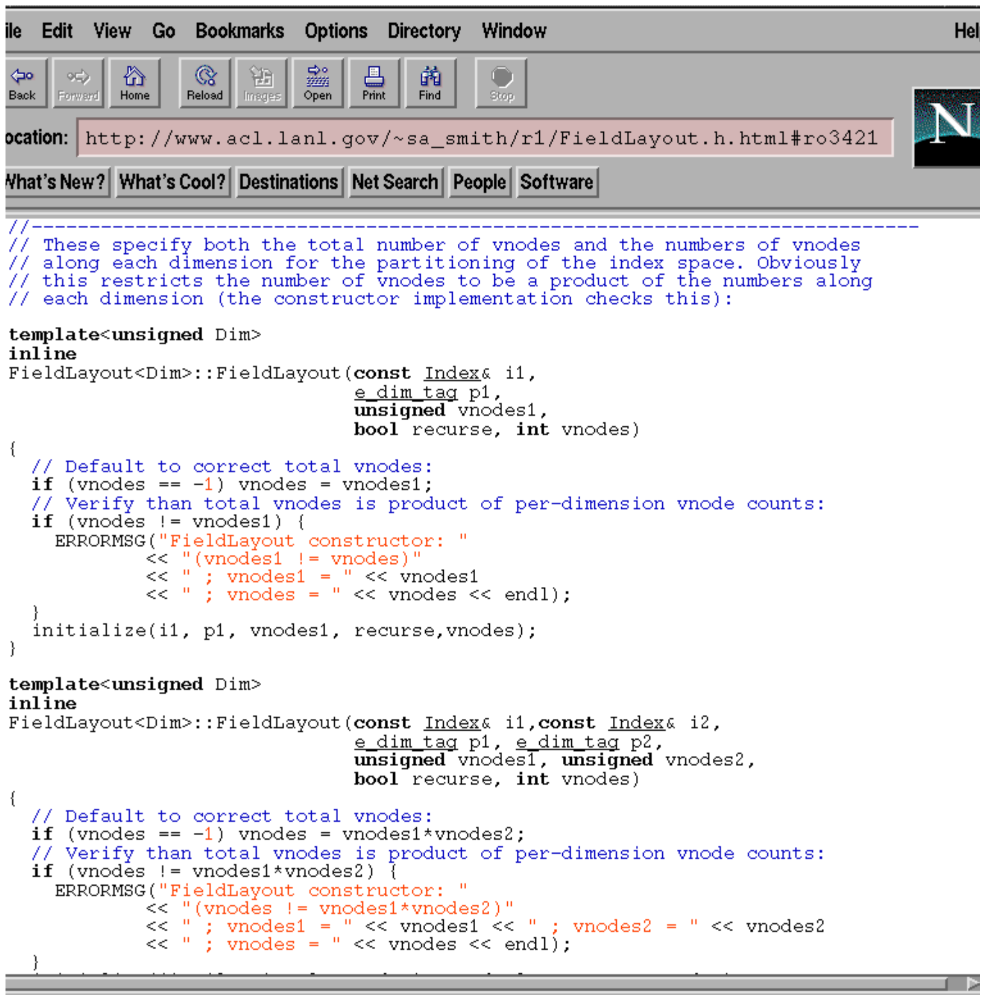
PDT assists in static analysis

Four **DUCTAPE** applications have been developed:

- ***pdbconv*** converts PDB files to a more readable format
- ***pdbmerge*** merges PDB files from separate compilations
  Merges template instantiations from separate compilations
- ***pdbtree*** prints file inclusion, class hierarchy, and call graph trees
  Supersedes TAU browsers
- ***pdbhtml*** “htmlizes” C++ source

Static Analysis and Documentation Generation (2)

pdbhtml output




```
//-----------------------------------------------------------------------------
// These specify both the total number of vnodes and the numbers of vnodes
// along each dimension for the partitioning of the index space. Obviously
// this restricts the number of vnodes to be a product of the numbers along
// each dimension (the constructor implementation checks this):

template<unsigned Dim>
inline
FieldLayout<Dim>::FieldLayout(const Index& i1,
                              e_dim_tag p1,
                              unsigned vnodes1,
                              bool recurse, int vnodes)
{
  // Default to correct total vnodes:
  if (vnodes == -1) vnodes = vnodes1;
  // Verify than total vnodes is product of per-dimension vnode counts:
  if (vnodes != vnodes1) {
    ERRORMSG("FieldLayout constructor: "
            << "(vnodes1 != vnodes)"
            << " ; vnodes1 = " << vnodes1
            << " ; vnodes = " << vnodes << endl);
  }
  initialize(i1, p1, vnodes1, recurse,vnodes);
}

template<unsigned Dim>
inline
FieldLayout<Dim>::FieldLayout(const Index& i1,const Index& i2,
                              e_dim_tag p1, e_dim_tag p2,
                              unsigned vnodes1, unsigned vnodes2,
                              bool recurse, int vnodes)
{
  // Default to correct total vnodes:
  if (vnodes == -1) vnodes = vnodes1*vnodes2;
  // Verify than total vnodes is product of per-dimension vnode counts:
  if (vnodes != vnodes1*vnodes2) {
    ERRORMSG("FieldLayout constructor: "
            << "(vnodes != vnodes1*vnodes2)"
            << " ; vnodes1 = " << vnodes1 << " ; vnodes2 = " << vnodes2
            << " ; vnodes = " << vnodes << endl);
  }
```

Outline

- Context
- Program Database Toolkit (PDT)
- Applications Using PDT
- **PDT Availability**

Implementation Summary

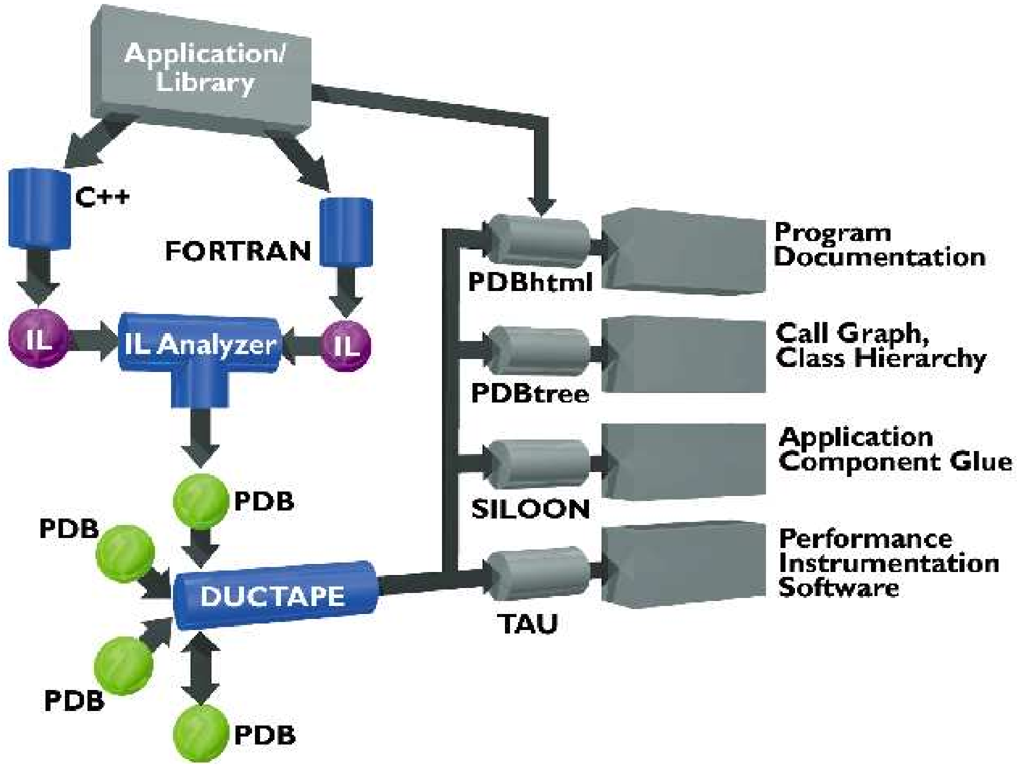
Programming languages supported:

- **C/C++**
- **Fortran 90**

Platforms supported:

Architecture	Platform	Operating System
hp9000s700	HP PA-RISC 1.1	HP-UX 10.20
linux	Intel i686	RedHat Linux 5.2
rs6000	IBM SP-2	AIX 4.2.1
sgi32	SGI R4400 IP22	IRIX 6.5.4
sgin32	SGI R10000 IP25	IRIX 6.5.4
sgi64	SGI R10000 IP25	IRIX 6.5.4
solaris2	SUN sun4m	Solaris SunOS 5.6
t3e	SGI Cray T3E	Unicos MK 2.0.4.61

Program Database Toolkit (PDT)



http://www.acl.lanl.gov/pdtoolkit/