The Deconstruction of Dyninst Part 1: The SymtabAPI

Giridhar Ravipati
University of Wisconsin, Madison

Motivation

- Binary tools are increasingly common
- Two categories of operation
  - Analysis : Derive semantic meaning from the binary code
    - Symbol tables (if present)
    - Decode (disassemble) instructions
    - Control-flow information: basic blocks, loops, functions
    - Data-flow information: from basic register information to highly sophisticated (and expensive) analyses.
  - Modification
    - Insert, remove, or change the binary code, producing a new binary.

Wide Use of Binary Tools

- Analysis and Modification are used in a wide variety of applications

- Binary Modification
  - Eel, Vulcan, Etch, Atom, Diablo, Diota
- Binary Matching
  - BMT
- Forensics
  - Fenris
- Reverse engineering
  - IDA Pro
- Binary Translation
  - Objcopy, UQBT
- Program tracing
  - QPT
- Program debugging
  - Total view, gdb, STAT
- Program testing
  - Eraser
- Performance modeling
  - METRIC
- Performance profiling
  - Paradyn, Valgrind, TAU, OSS

Lack of Code Sharing

- Some tools do analysis and some tools do modification
  - Only a few do both
- Tools usually depend on
  - Similar analysis
  - Similar modification techniques
- Too many different interfaces
  - Usually too low level
- Developers are forced to reinvent the wheel rather than use existing code

Lack of Portability

- Myriad number of differences between
  - File formats
  - Architectures
  - Operating systems
  - Compilers
  - …
- Building a portable binary tool is highly expensive
  - Many platforms in common use

High-level goals

- To build a toolkit that
  - Has components for analysis
  - Has components for modification
  - Is portable & extensible
  - Has an abstract interfaces
  - Encourage sharing of functionality
- Deconstruct Dyninst into a toolkit that can achieve these goals

DyninstAPI

- Library that provides a platform-independent interface to dynamic binary analysis and modification
- Goal
  - Simplify binary tool development
- Why is Dyninst successful?
  - Analysis and modification capabilities
  - Portability
  - Abstract interface

Drawbacks of Dyninst

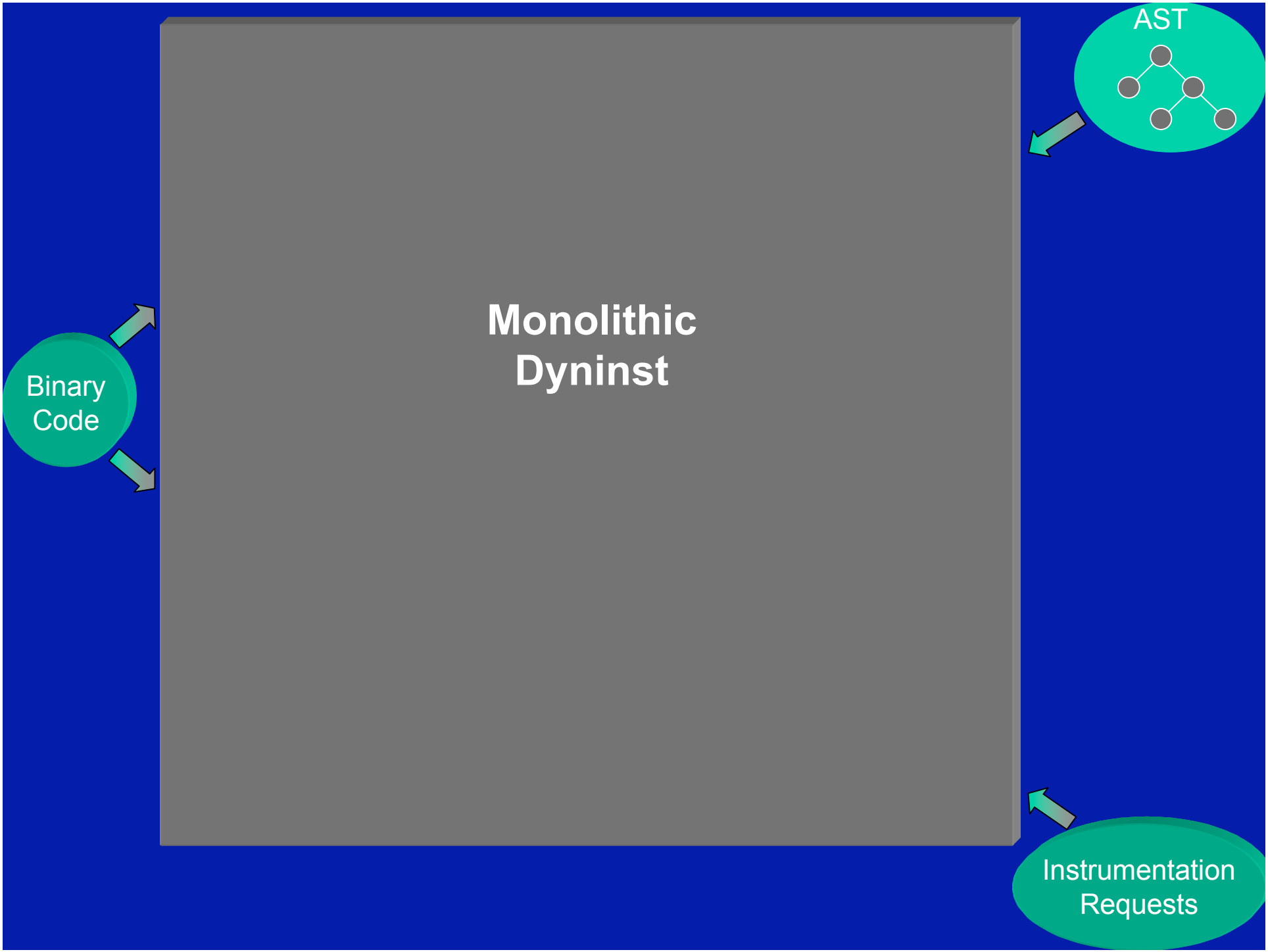
- Dyninst is complex
- Dyninst internal components are portable but not sharable
- Sometimes Dyninst is not a perfect match for user requirements
- Dyninst is feature-rich in some cases
  - Provides unnecessary extra functionality

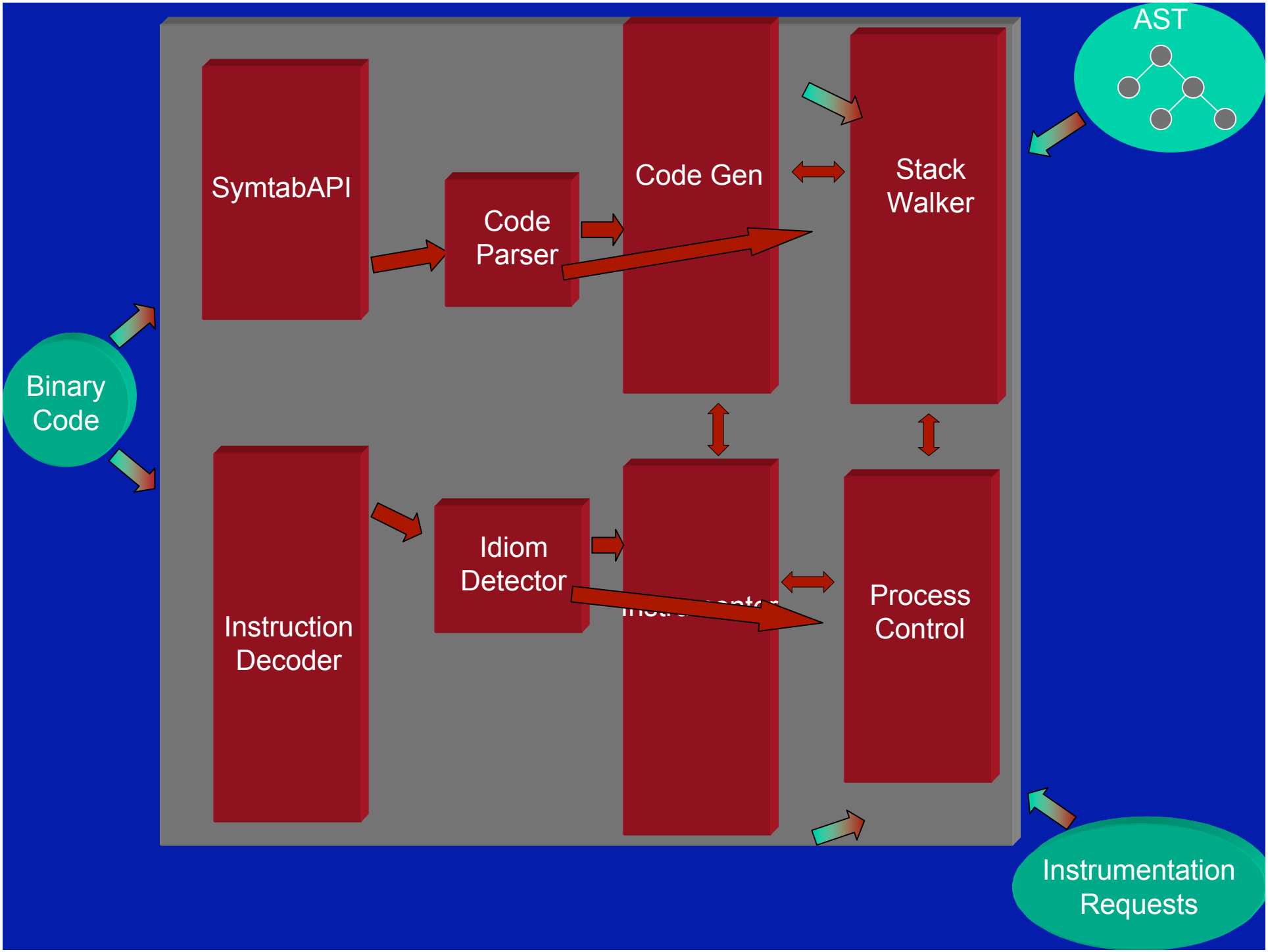
Example Scenarios

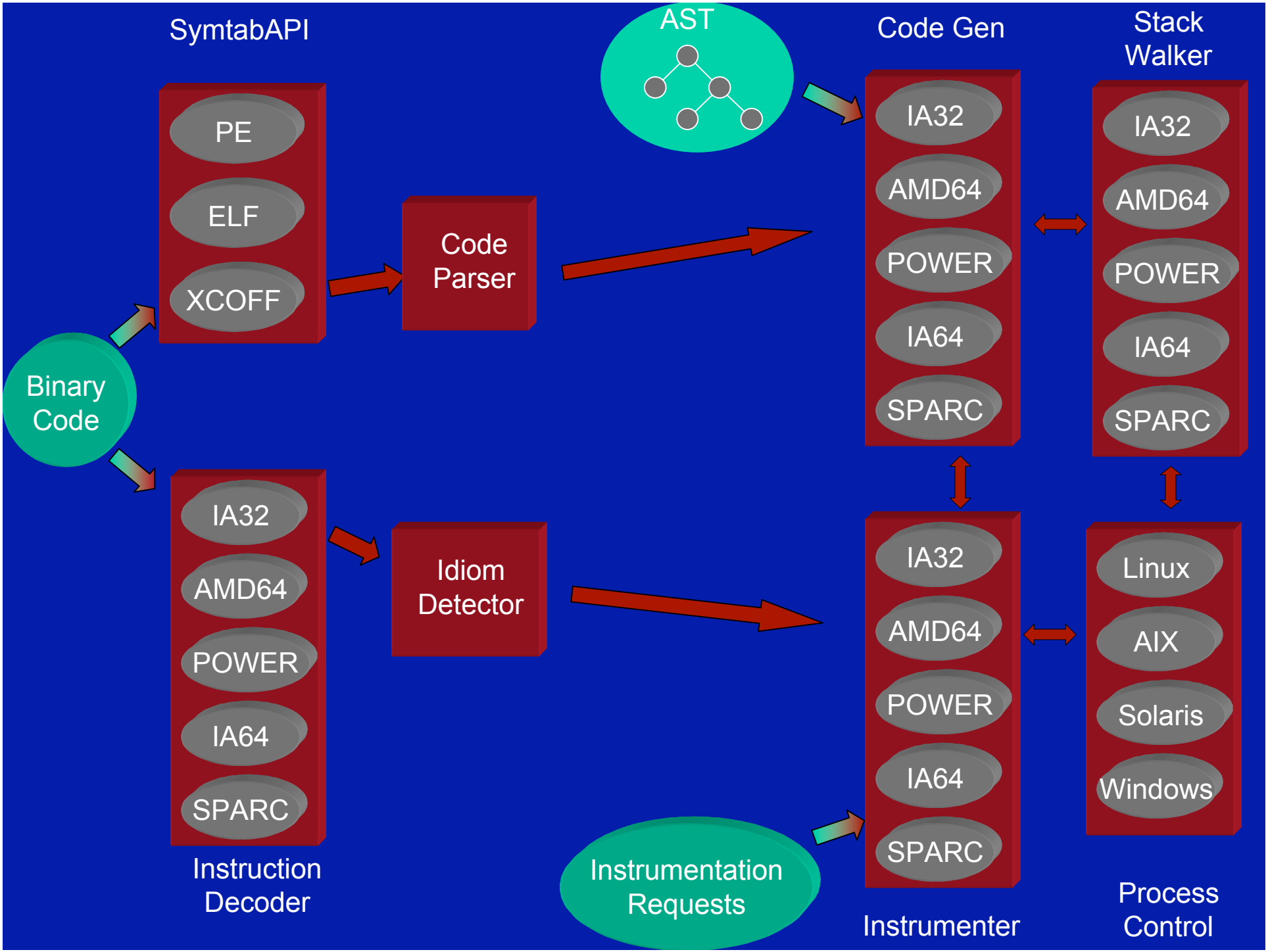
- Hidden functionality
  - Statically parse and analyze a binary without executing it
  - Just perform stackwalking on a binary compiled without frame pointer information
- Build new tools
  - Static binary rewriter
  - Tool to add a symbol table to stripped binaries

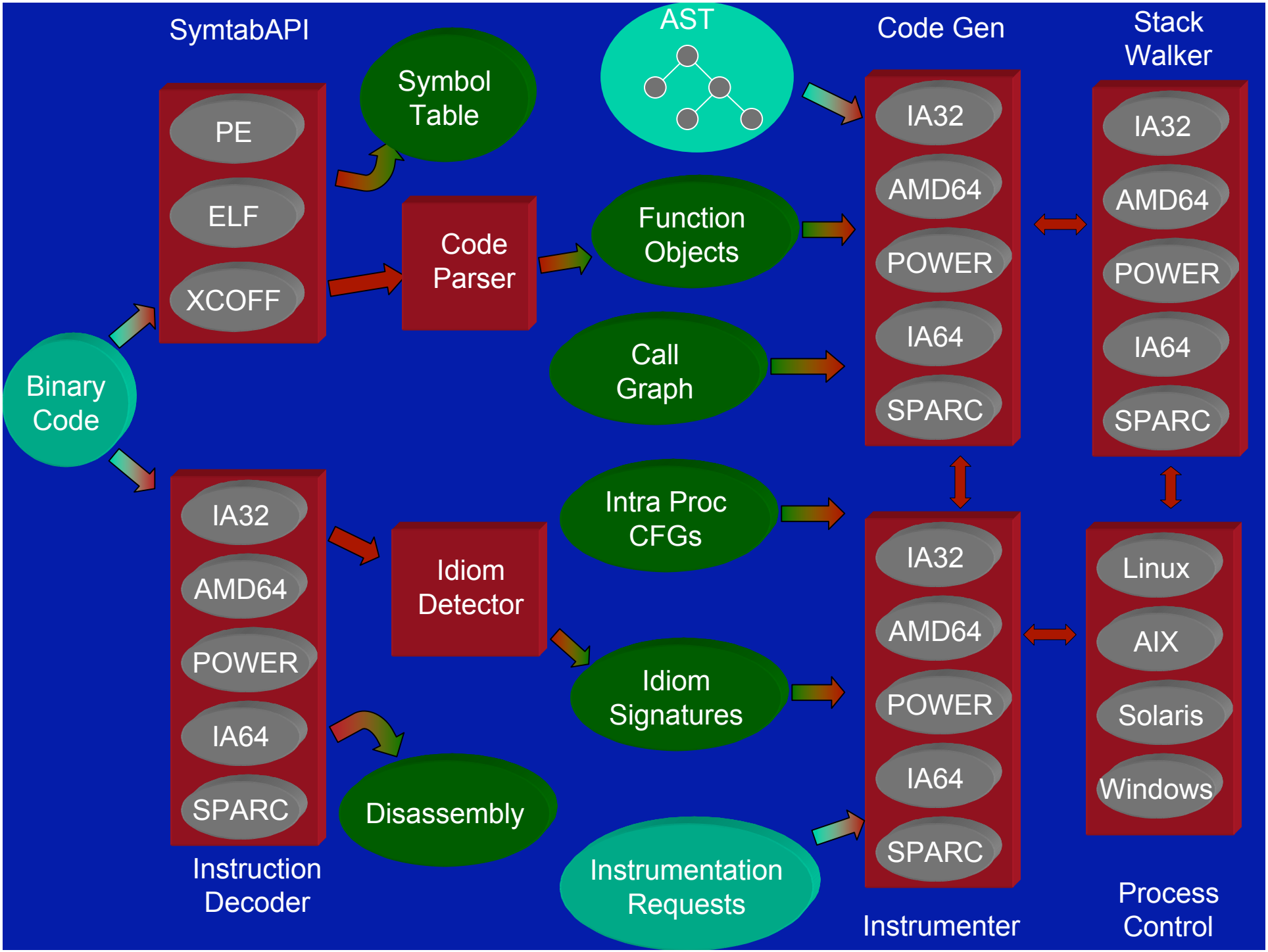
Our Approach

- Deconstruct the monolithic Dyninst into a suite of components
- Each component provides a platform-independent interface to a core piece of Dyninst functionality


AST
Monolithic
Dyninst
Binary
Code
Instrumentation
Requests


AST
SymtabAPI
Code Parser
Code Gen
Stack Walker
Binary Code
Instruction Decoder
Idiom Detector
Process Control
Instrumentation Requests


SymtabAPI
PE
ELF
XCOFF
Code Parser
AST
Code Gen
IA32
AMD64
POWER
IA64
SPARC
Stack Walker
IA32
AMD64
POWER
IA64
SPARC
Binary Code
IA32
AMD64
POWER
IA64
SPARC
Instruction Decoder
Idiom Detector
IA32
AMD64
POWER
IA64
SPARC
Instrumenter
Linux
AIX
Solaris
Windows
Process Control
Instrumentation Requests


SymtabAPI
PE
ELF
XCOFF
Symbol Table
Code Parser
Binary Code
AST
Code Gen
IA32
AMD64
POWER
IA64
SPARC
Function Objects
Call Graph
Stack Walker
IA32
AMD64
POWER
IA64
SPARC
IA32
AMD64
POWER
IA64
SPARC
Idiom Detector
Intra Proc CFGs
Idiom Signatures
Disassembly
Instruction Decoder
Instrumentation Requests
IA32
AMD64
POWER
IA64
SPARC
Instrumenter
Linux
AIX
Solaris
Windows
Process Control

Goals of Deconstruction

- Separate the key capabilities of Dyninst
- Each Component
  - Is responsible for a specific functionality
  - Provides a general solution
- Encourage sharing
  - Share our functionality when building new tools
  - Share functionality of other tools

Benefits of Deconstruction

- Access to the hidden features of Dyninst
- Interoperability with other tools
  - Standardized interfaces and sharing of components
- Finer grain testing of Dyninst

Benefits of Deconstruction [contd.]

- Code reuse among the tool community
- Make tools more portable
- Unexpected benefits with new application of components

Our Plan

- Identify the key functionality
- Refine and generalize the abstract interfaces to these components
- Extract and separate the functionality from Dyninst
- Rebuild Dyninst on top of these components
- Create new tools
  - Multi-platform static binary rewriter

SymtabAPI

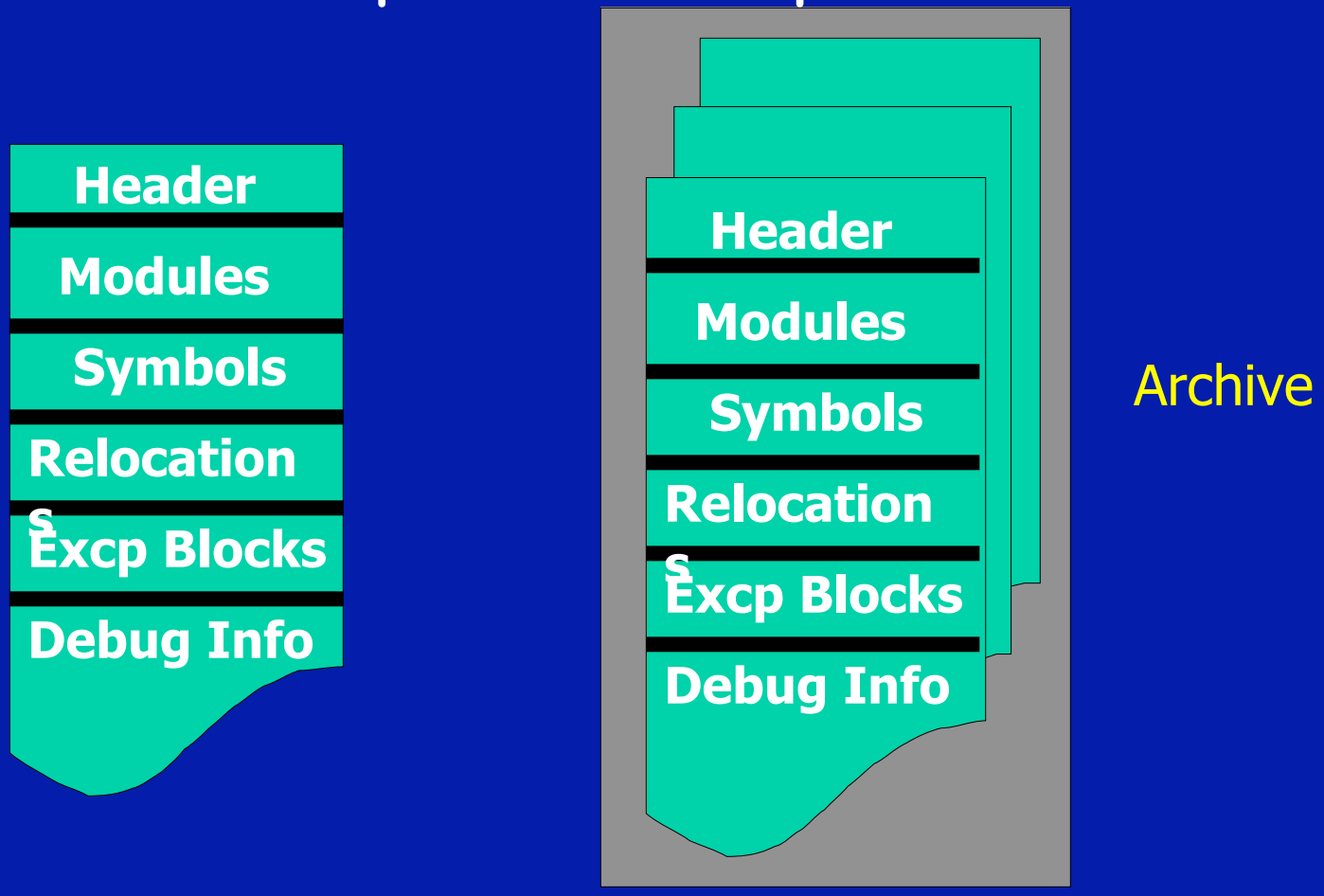
- The first component of the deconstructed Dyninst
- Multi platform library for parsing symbol table information from object files
- Leverages the experience and implementation gained from building the DyninstAPI

SymtabAPI Goals

- Abstraction
  - Be file format-independent
- Interactivity
  - Update data incrementally
- Extensibility
  - User-extensible data structures
- Generality
  - Parse ELF/XCOFF/PE object files
  - On-Disk/In-Memory parsing

SymtabAPI Abstractions

- Represents an object file in a canonical format
- Hides the multi-platform dependences



SymtabAPI Extensibility

- Abstractions are designed to be extensible
- Can annotate particular abstractions with tool specific data
  - e.g. : Store type information for every symbol in the symbol table

Interactivity/Extensibility

Symbol	Address	Size
func1	0x0804cc84	100
variable1	0x0804cd00	4
func2	0x0804cd1d	50
...	...	

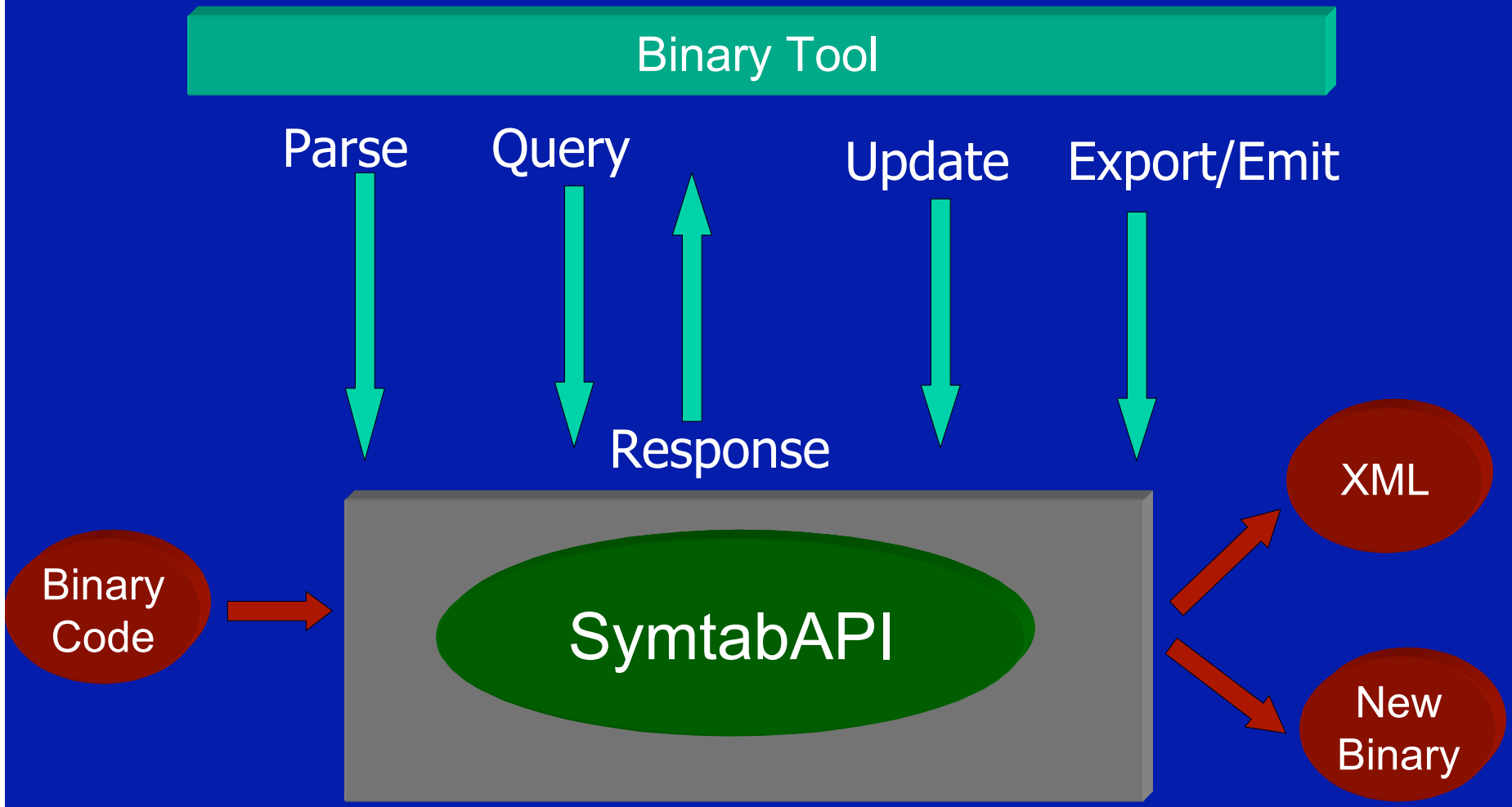
Type Information
int

Register	Is Live?
R1	Yes
R2	No
R3	Yes
R4	Yes

SymtabAPI Interface

- Information from a parsed-binary is kept in run time data structures
- Intuitive query-based interface
  - e.g. findSymbolByType(name,type)
  - Returns matching symbols
- Data can then be updated by the user
- Modifications available for future queries

Query/Update/Export/Emit



Summary of Operations

- Parse the symbols in a binary
- Query for symbols
- Update existing symbol information
- Add new symbols
- Export/Emit symbols

- More details/operations in the SymtabAPI programmer's guide

Current Status

- Released the initial version of SymtabAPI with the 5.1 release of Dyninst
- Dyninst on top of SymtabAPI
- XML export
- Emit on Linux and AIX

Ongoing & Future Work

- Import XML
- Emit a new binary on windows
- Debugging information for symbols
- Interfaces for the remaining components
- Multi-platform static binary rewriter

Demo

- Please stop by and see our demo of stripped binary parsing with the SymtabAPI's emit functionality on Linux

Tuesday, May 1, 2007

Room No – 206

2:00 PM – 3:00 PM

Downloads

- SymtabAPI
  - http://www.paradyn.org/html/downloads.html
- SymtabAPI Programmer's guide
  - http://www.paradyn.org/html/symtabAPI.html
- Ravipati, G., Bernat, A., Miller, B.P. and Hollingsworth, J.K., "Toward the Deconstruction of Dyninst", Technical Report