

DSECRG

# DBI: Intro



Dmitriy “D1g1” Evdokimov

@evdokimovds

DSecRG

# Code analysis

A stylized owl with a glowing orange eye and a 'DSECRG' watermark. The owl is composed of many sharp, translucent, crystalline facets, giving it a geometric, low-poly appearance. The background is a light, textured surface, possibly representing feathers or a similar material.

- Static
  - Without run
- Dynamic
  - With run

# Dynamic analysis

DSECRG

- With source code
  - Source code
    - » Source code instrumentation
  - Compiler
    - » Compiler Instrumentation
- Without source code
  - Binary file
    - » Static Binary Instrumentation
  - Process
    - » Dynamic Binary Instrumentation
    - » Debugging API
  - Environment
    - » Emulation
    - » Virtualization
    - » Paravirtualization
  - Hardware
    - » Debug register, etc

# DBI

DSECRG

Dynamic Binary Instrumentation (DBI) is a process control and analysis technique that involves injecting instrumentation code into a running process.

Dynamic binary analysis (DBA) tools such as profilers and checkers help programmers create better software.

Dynamic binary instrumentation (DBI) frameworks make it easy to build new DBA tools.

# Non-security use DBI

DSECRG

- Simulation / Emulation
- Performance analysis
- Correctness checking
- Memory debugging
  - Parallel optimization
  - Call graphs
  - Collecting code metrics
  - Automated debugging

# Security use DBI

DSECRG

- Taint analysis
- Taint (DATA FLOW) analysis
- Control flow analysis
- Privacy monitoring
- KNOWN vulnerability detection
- Unknown vulnerability detection
- Vulnerability Detection
- Fuzzing / security test case generation
  - Advanced monitoring
  - Automated exploit development
  - Automated vaccinations
  - Pre-patching of vulnerabilities
  - Reversing
  - Transparent debugging
  - Behavior based security

# Existing tools for security

DSECRG

- Avalanche
- Flayer
- tartetatintools
- Sweeper
- Determina
- Flayer
- Pincov
- Taintdroid
- Privacy Scope
- DynTrace
- Code Coverage
- RunTracer
- VERA
- Tripoux
- TraceSurfer
- SHAN
- Fuzzgrind

# DBI Frameworks

The logo for Pin, featuring the word "Pin" in a white, serif font inside a blue rectangular box.The logo for Dynamic RIO, featuring the word "Dynamic" in a small, black, sans-serif font above the word "RIO" in a large, bold, blue, sans-serif font. A yellow lightning bolt graphic strikes through the "IO" part of "RIO".

The DR. is in.

The logo for Valgrind, featuring the word "Valgrind" in a large, brown, serif font.The logo for Dyninst, featuring the word "Dyn" in a large, blue, serif font above the word "inst" in a smaller, blue, serif font.

# PIN

site: [www.pintool.org](http://www.pintool.org)

Development: Intel (Open Source)

Modes:

- Probe
- JIT

Start:

```
pin.exe [pin_options] -t pintool_name.dll [pintool_options] -- app_name.exe
```

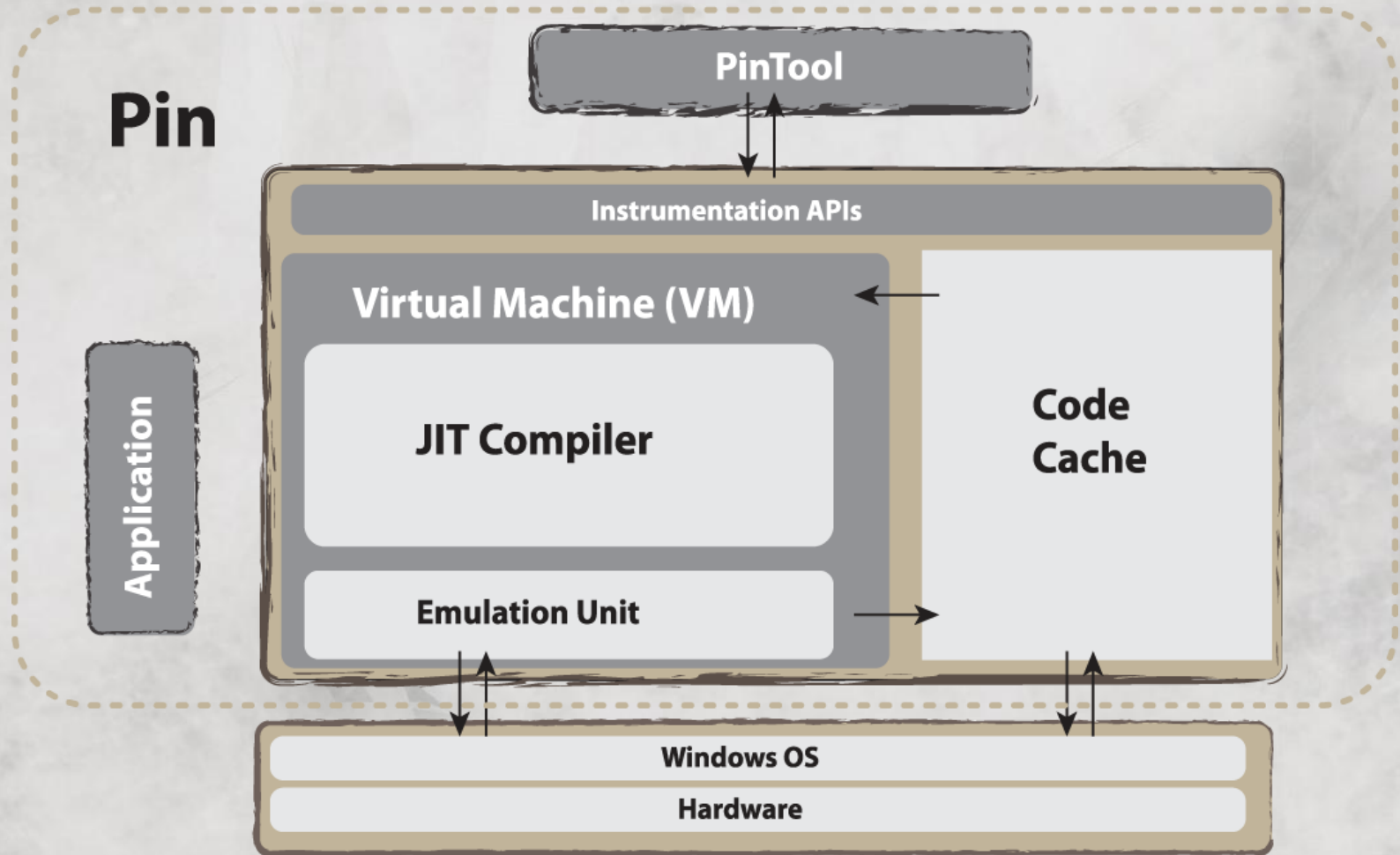
Terminology:

- Pintool

	Linux	Windows
x86	+	+
amd64	+	+
IA-64	+	+

# PIN

Address Space

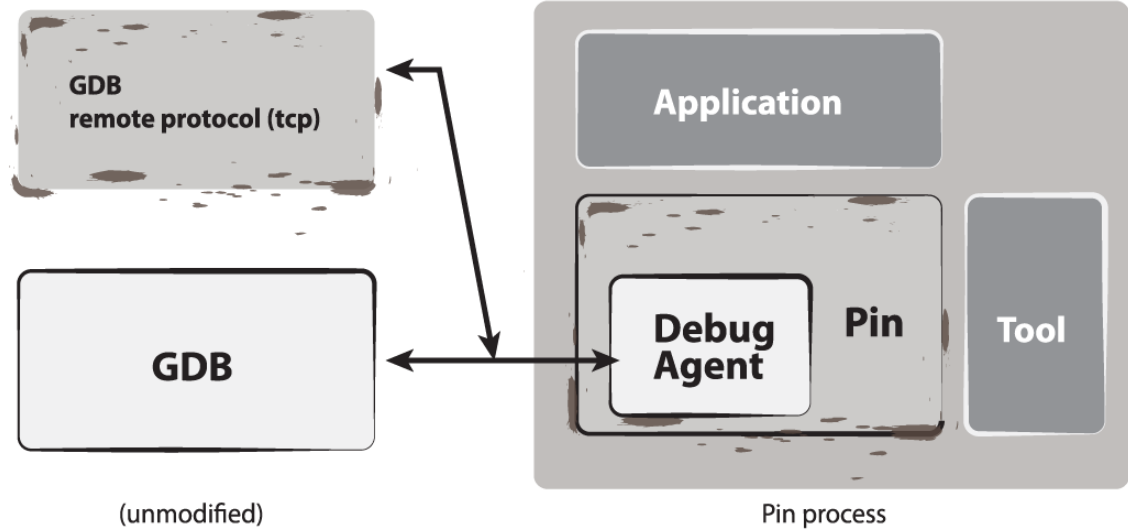


# PIN

Execution: dynamic binary compilation and caching

Granularity:

- - Instruction (INS)
- - Basic Block (BBL, not classic)
- - Trace (TRACE)
- - Function (RTN)
- - Section (SEC)
- - Module (IMG)



Features:

- - Debug Agent
  - "-appdebug" (on Linux)
- - XED library
  - Coder/decoder

# Valgrind

site: [www.valgrind.org](http://www.valgrind.org)

Development: Group of people from all over the world (GNU GPL v2)

Features:

- Intermediate Representation (VEX)

Terminology:

- Tool plug-in

Software package:

- Framework core
- Several tools

Start:

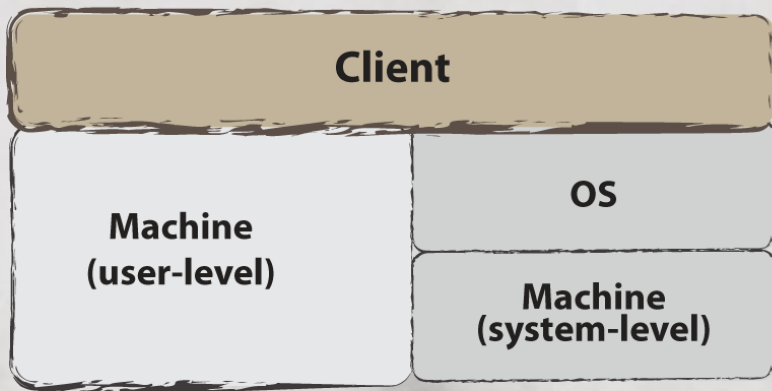
```
./valgrind [options] --tool=tool_name program [args]
```

Remarks:

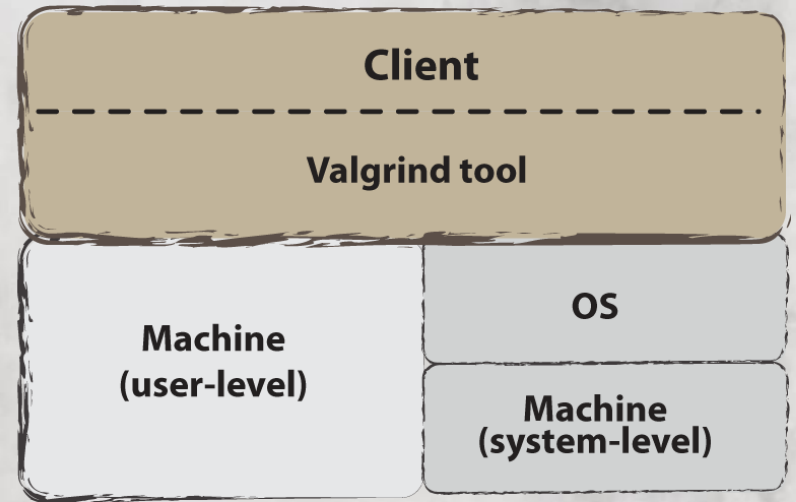
- Valgrind controls every instruction
- Self-modifying code won't run correctly
- Valgrind + Wine = Windows
- Interaction with source code
- Designed for heavyweight DBA tools

	Linux	Android (2.3.X)	Darwin (Mac OS X 10.6 & 10.7)
x86	+		+
amd64	+		+
ARM	+	+	
ppc32	+		
ppc64	+		

# Valgrind



(a) Normal execution



(b) Execution under a Valgrind tool

# Valgrind

Execution: dynamic binary compilation and caching

Before a code block is executed for the first time:

- 1) Core: machine code -> (architecture neutral) IR
- 2) Tool: IR -> instrumented IR
- 3) Core: instrumented IR -> instrumented machine code
- 4) Core: caches and links generated translations

Patch-based instrumentation:

- code cache
  - Disassembly
  - Optimization
  - Instrumentation
  - Register allocation
  - Code generation

```
0x24F275: movl -16180(%ebx,%eax,4),%eax
1: ----- IMark(0x24F275, 7) -----
2: t0 = Add32(Add32(GET:I32(12),# get %ebx and
Sh132(GET:I32(0),0x2:I8)), # %eax, and
0xFFFFC0CC:I32) # compute addr
3: PUT(0) = LDle:I32(t0) # put %eax
```

# DynamoRIO

## Development:

HP Labs -> MIT + HP -> Determina -> VMware -  
> Open Source Releases (BSD license)

site: [www.dynamorio.org](http://www.dynamorio.org)

## Goals:

- Runtime optimization
- Introspection system

## Terminology:

- Client
- Stub
- Nudge

Start 1: `drrun.exe -client clien_name.dll params prog.exe`

Start 2:

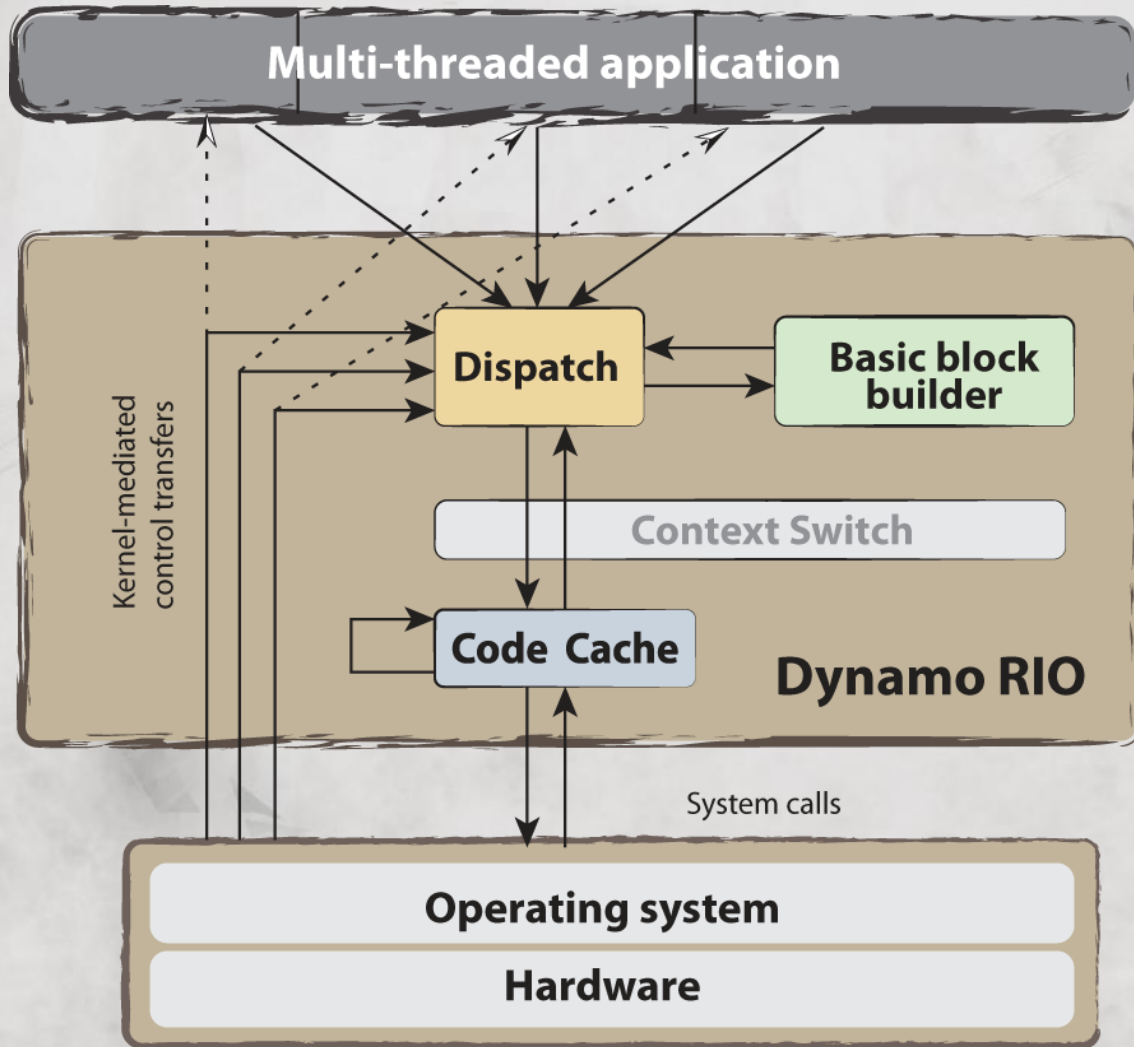
- 1) `drconfig.exe -reg prog.exe -client client_name.dll params`
- 2) `drinject.exe params prog.exe prog_params`

Start 3:

`drconfig.exe -nudge prog.exe params`

	Linux	Windows
x86	+	+
amd64	+	+

# DynamoRIO



# DynamoRIO

Execution: dynamic binary compilation and caching

Features :

- Multi client supporting
- Adaptive Level of Granularity
- IR support
- Translation support

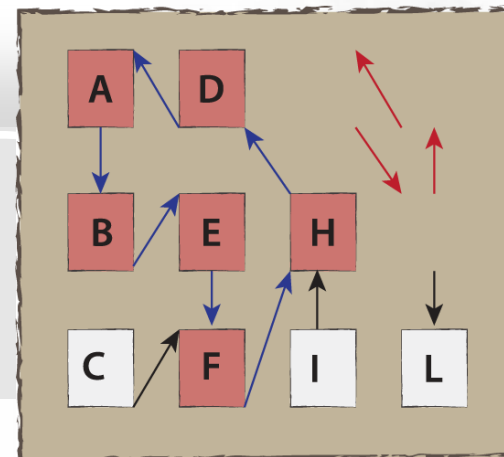
Patch-based instrumentation:

- code cache
  - Caching
  - Linking
  - Building traces

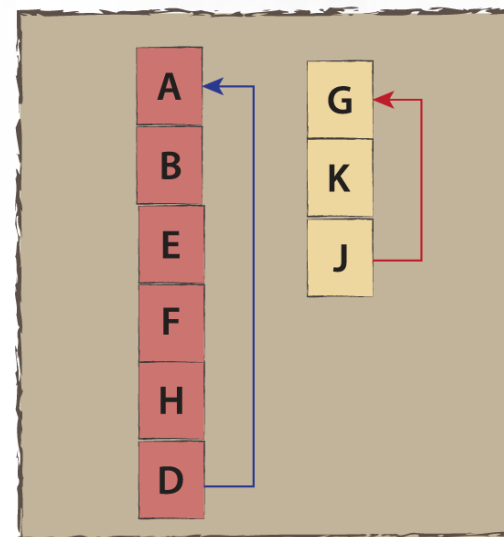
API:

- DynamoRIO
- Standalone (Static Binary Analyze)
- Start/Stop (Source Code Instrumentation)

Basic Block Cache



Trace Cache



# DynInst

Development:

site: [www.dyninst.org](http://www.dyninst.org)

University of Wisconsin-Madison, University of Maryland

Framework support:

- Static Binary Instrumentation
- Dynamic Binary Instrumentation

API:

- DynInstAPI
- SymtabAPI
- InstructionAPI
- ParseAPI
- StackwalkerAPI
- ProcControlAPI
- DynC

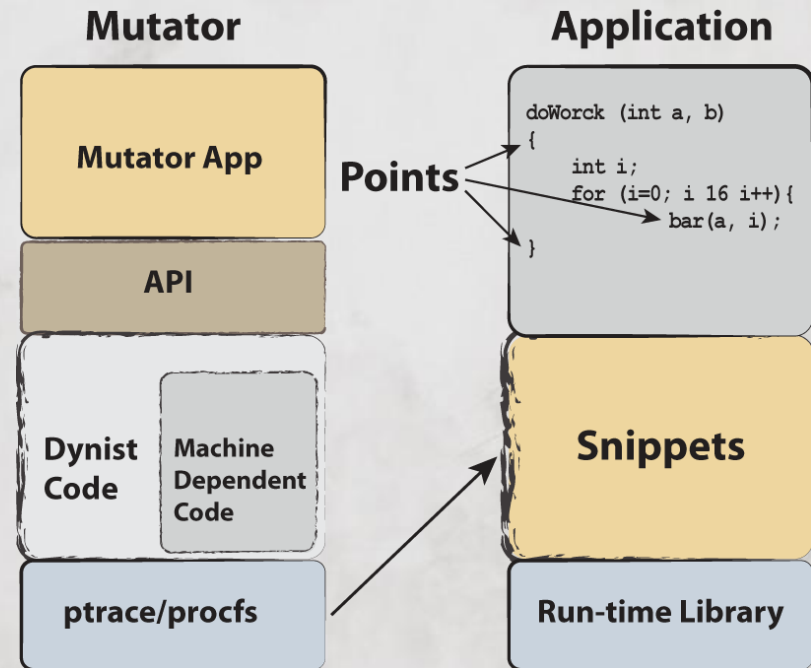
Start:

- 1) Set up the mutatee (link app with RTI library)
- 2) Run the mutator

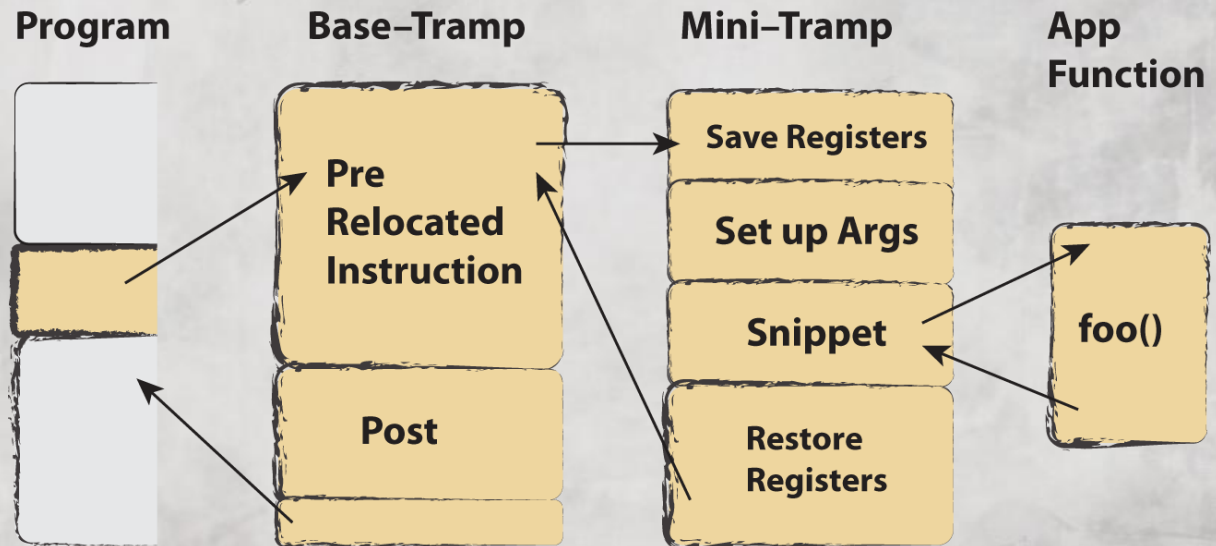
	Linux	BlueGene/P	FreeBSD	Windows
x86	+		+	+
amd64	+		+	
ppc32	+	+		
ppc64	+			

# DynInst

## Abstractions Used in the API



## Inserting Code into a Running Program



# DynInst

Execution: Normal execution with inline trampolines

Abstractions:

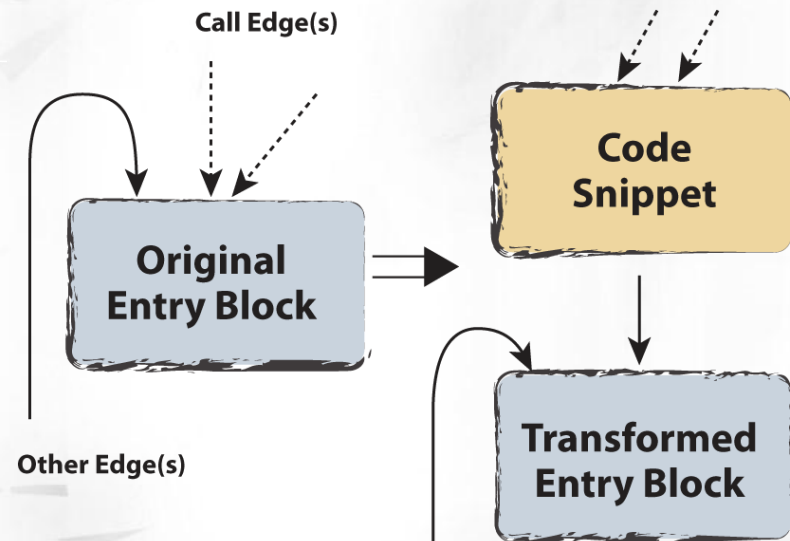
- Basic Blocks (classic)
- Edges (Edges are labeled with an edge type)
- Functions
- Loops (natural)

Patch-based instrumentation:

- interception branches
  - selection
  - relocation
  - patching

Terminology:

- Mutator
- Mutant
- InstPoints
- Snippets



# Useful links

## Pin:

- [http://www.cs.virginia.edu/kim/publicity/pin/tutorials/cgo11/cgo\\_2011\\_final\\_1.ppt](http://www.cs.virginia.edu/kim/publicity/pin/tutorials/cgo11/cgo_2011_final_1.ppt)
- [http://archive.hack.lu/2011/Binary%20Instrumentation%20for%20Hackers%20-%20hack.lu%20-%20Gal%20Diskin%20\(final\).pptx](http://archive.hack.lu/2011/Binary%20Instrumentation%20for%20Hackers%20-%20hack.lu%20-%20Gal%20Diskin%20(final).pptx)

## Valgrind:

- <http://www.valgrind.org/docs/phd2004.pdf>
- <http://valgrind.org/docs/iiswc2006.pdf>

## DynamoRIO:

- <http://code.google.com/p/dynamorio/downloads/list>
- <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.7639&rep=rep1&type=pdf>

## DynInst:

- <ftp://ftp.cs.wisc.edu/paradyn/papers/Bernat11AWAT.pdf>
- <http://www.dyninst.org/papers/apiPreprint.pdf>