Binary Translation

Issues with Static Binary Translation

- Code Discovery
- Self Modifying Code and Dynamic Linking
- Indirect Branching
- Run time Change of State

Integration of Gem5 and QEMU

- Gem5
- Resources: Output Assembly log from QEMU, Gem5
- Idea1: Reconstruct Host Binary
- Idea2: Use Trace Driven Approach

QEMU

- System Mode Emulation
- User Mode Emulation
- Virtualization, Cross Compilation development environments
- Supported OS: Linux, Windows, MacOS
- Supported ISAs: x86, ARM, MIPS, SPARC, [illegible]
- CPU Emulation (TCG code generator)
- [illegible]
- User Interface

Outline

- Introduction of Static and Dynamic BT (Done).
- Solution of problems of SBT by DBT.
- Structure of a Dynamic Binary Translator.
- Brief discussion of few DBT tools.
- Detailed discussion of QEMU (TCG).
- Integration of QEMU with gem5.
- Note: Guest/Source/Foreign -> Host/Target/Native

Dynamic Binary Tranlation Tools

Binary Translation

- "Emulation of one Instruction set by another through translation of binary code."
- Motivation: Running Legacy Code, Cost savings, Server Virtualization, Cross ISA virtual Machines (e.g. VMWare), Application Migration, Better Performance (e.g. Superoptimizer peephole), Memory and Profiling Tools (e.g. Valgrind)
- Types

Dynamic Binary Translation

- Translation on the fly (in time)
- It is easy to do correctly.
- Slower than Static Binary Translation.
- Examples: Hardware/Software
- Software: Rosetta dynamic translation layer (MacOSX), IA-32 Execution Layer DBT on Itanium based systems.
- Hardware: x86 -> [illegible]

Dynamic Binary Translator

It's just a host process which runs guest application in its own address space

Static Binary Translation

- Ahead of time.
- Usually fast than its alternative
- Difficult to do Correctly
- Problems: Code Discovery, Dynamic Linking, Self Modifying Code
- Example: A lot of video games have been statically translated, historically.

Binary Translation

Issues with Static Binary Translation

- Code Discovery
- Self Modifying Code and Dynamic Linking
- Indirect Branching
- Run time Change of State

Integration of GemS and QEMU

- GemS
- Resources: Output Assembly log from QEMU, GemS
- Idea1: Reconstruct Host Binary
- Idea2: Use Trace Driven Approach

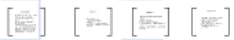
QEMU

- System Mode Emulation
- User Mode Emulation
- Virtualization, Cross Compilation development environments
- Supported OS: Linux, Windows, MacOSX
- Supported ISAs: x86, ARM, MIPS, SPARC,ALPHA
- CPU Emulator (TCG code generator)
- Emulated Devices
- User Interface

Outline

- Introduction of Static and Dynamic BT (Done).
- Solution of problems of SBT by DBT.
- Structure of a Dynamic Binary Translator.
- Brief discussion of few DBT tools.
- Detailed discussion of QEMU (TCG).
- Integration of QEMU with gemS.
- Note: Guest/Source/Foreign -> Host/Target/Native

Dynamic Binary Tranlation Tools



Binary Translation

- "Emulation of one Instruction set by another through translation of binary code."
- Motivation: Running Legacy Code, Cost savings, Server Virtualization, Cross ISA virtual Machines (e.g. VMWare), Application Migration, Better Performance (e.g. Superoptimizer peephole), Memory and Profiling Tools (e.g. Valgrind)
- Types

Dynamic Binary Translation

- Translation on the fly (In time)
- It is easy to do correctly.
- Slower than Static Binary Translation.
- Examples: Hardware/Software
- Software: Rozetta dynamic translation layer (MacOSx), IA-32 Execution Layer-DBT on Itanium based systems.
- Hardware: x86->iops

Dynamic Binary Translator

It's just a host process which runs guest application in its own address space

Static Binary Translation

- Ahead of time.
- Usually fast than its alternative
- Difficult to do Correctly
- Problems: Code Discovery, Dynamic Linking, Self Modifying Code
- Example: A lot of video games have been statically translated, historically.

Binary Translation

- **"Emulation of one Instruction set by another through translation of binary code."**
- **Motivation: Running Legacy Code, Cost savings, Server Virtualization, Cross ISA virtual Machines (e.g. VMWare), Application Migration, Better Performance (e.g. Superoptimizer peephole), Memory and Profiling Tools (e.g. Valgrind)**
- **Types**

Static Binary Translation

- Ahead of time.
- Usually fast than its alternative
- Difficult to do Correctly
- Problems: Code Discovery, Dynamic Linking, Self Modifying Code
- Example: A lot of video games have been statically translated, historically.

Dynamic Binary Translation

- **Translation on the fly (In time)**
- **It is easy to do correctly.**
- **Slower than Static Binary Translation.**
- **Examples: Hardware/Software**
- **Software:** Rosetta dynamic translation layer (MacOSx), IA-32 Execution Layer-DBT on Itanium based systems.
- **Hardware:** x86->uops

Outline

- Introduction of Static and Dynamic BT (Done).
- Solution of problems of SBT by DBT.
- Structure of a Dynamic Binary Translator.
- Brief discussion of few DBT tools.
- Detailed discussion of QEMU (TCG).
- Integration of QEMU with gem5.
- Note: Guest/Source/Foreign -> Host/Target/Native

Issues with Static Binary Translation

- Code Discovery
- Self Modifying Code and Dynamic Linking
- Indirect Branching
- Run time Change of State

Code Discovery

- Data in Instruction Stream
- Compiler Optimizations
- Padding for Instruction Allignment.
- Example: Switch Statement in C.
- Jump table in .text segment (contains addresses not code)
- Not a problem for Dynamic Binary Translation

Self Modifying Code and Dynamic Linking

- No way to handle if code is modified at run time.
- Run time loading of some library , unloading and then loading something else at the same address (Plug in systems)

Indirect Branching

- Target Addresses unknown statically

x86 Source	PowerPC Target

movl %eax,4(%esp)
jmp *%eax
	addi r16, r11, 4
	lwzx r4, r2, r16
	mtctr r4
	bctr

- r4 contains source address, should be translated to native address
- not possible before time

Run time Change of State

- Some instructions can change runtime state and affect translation.
- Example: setend on ARM used to switch CPU endianness.

"Static Translation: never a complete solution for Von-Neuman architectures where code and data reside in same memory" [1]

[1]"Dynamic Binary Translation", Mark Probst

Code Discovery

- Data in Instruction Stream
- Compiler Optimizations
- Padding for Instruction Allignment.
- Example: Switch Statement in C.
- Jump table in .text segment (contains addresses not code)
- Not a problem for Dynamic Binary Translation

Self Modifying Code and Dynamic Linking

- No way to handle if code is modified at run time.
- Run time loading of some library , unloading and then loading something else at the same address (Plug in systems)

Indirect Branching

- Target Addresses unknown statically

x86 Source

```
movl  %eax,4(%esp)
jmp   %eax
```

PowerPC Target

```
addi r16, r11, 4
lwzx r4,  r2,  r16
mtctr  r4
bctr
```

- r4 contains source address, should be translated to native address
- not possible before time

Run time Change of State

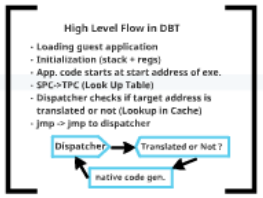
- Some instructions can change runtime state and affect translation.
- Example: setend on ARM used to switch CPU endianness.

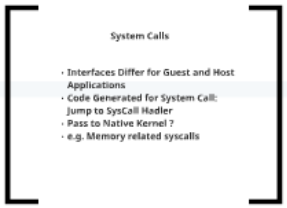
"Static Translation: never a complete solution for Von-Neuman architectures where code and data reside in same memory" [1]

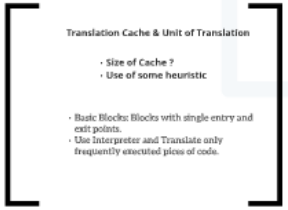
[1]"Dynamic Binary Translation", Mark Probst

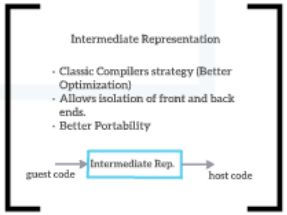
Dynamic Binary Translator

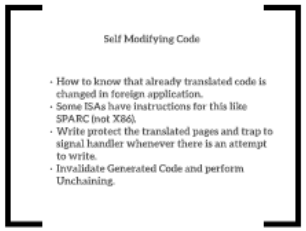
It's just a host process which runs guest application in its own address space





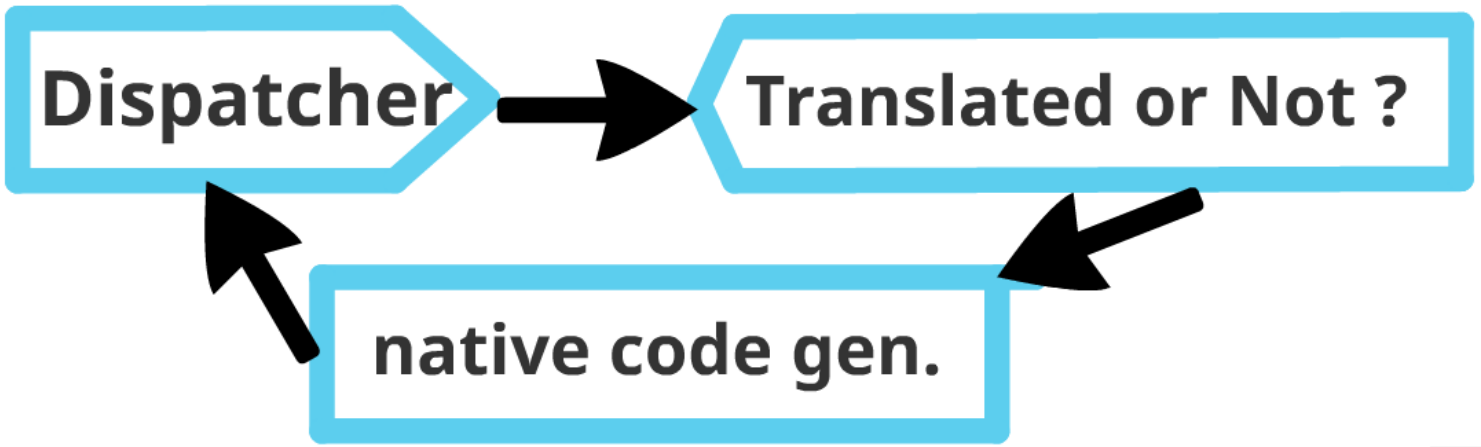






High Level Flow in DBT

- Loading guest application
- Initialization (stack + regs)
- App. code starts at start address of exe.
- SPC->TPC (Look Up Table)
- Dispatcher checks if target address is translated or not (Lookup in Cache)
- jmp -> jmp to dispatcher



System Calls

- Interfaces Differ for Guest and Host Applications
- Code Generated for System Call: Jump to SysCall Hadler
- Pass to Native Kernel ?
- e.g. Memory related syscalls

Patching of Direct Jumps

- Dispatcher can become a bottleneck
- Patch Direct Jumps
- Move Directly to Next Block / Fragment.
- "Translation Chaining"

Translation Cache & Unit of Translation

- **Size of Cache ?**
- **Use of some heuristic**

- Basic Blocks: Blocks with single entry and exit points.
- Use Interpreter and Translate only frequently executed pices of code.

Intermediate Representation

- Classic Compilers strategy (Better Optimization)
- Allows isolation of front and back ends.
- Better Portability

Endianness and Address Space

- Difference of Endiannes.
- Solution_1: Byte Swapping before store and after load (Expensive).
- Solution_2: Invert the whole address space

- Foreign Applications should not meddle with Translator's memory region.
- Solution: Control of guest's memory related syscalls by translator.
- Separate processes

Registers Mapping

- Trivial when native regs. are greater in number than foreign regs. -> Static Mapping.
- Registers in Memory: Loads and Stores become a bottleneck
- Hybrid Approach

Self Modifying Code

- How to know that already translated code is changed in foreign application.
- Some ISAs have instructions for this like SPARC (not X86).
- Write protect the translated pages and trap to signal handler whenever there is an attempt to write.
- Invalidate Generated Code and perform Unchaining.

Dynamic Binary Tranlation Tools

BinTrans

- Supports: PowerPc to Alpha, i386 to PowerPc and i386 to Alpha
- Supports user level applications only.
- Unit of Translation: Seq. blocks ending at jumps
- Concurrently run interpreter and native generated code, check states.
- Depending on foreign-native pair different reg. allocation and byte ordering strategies are applied.

fastBT

- Generator for DBTs
- Only User Space support
- IA32->ia32
- 0-10% overhead (some exceptions)
- Requires a high level instruction table from user

HDTrans

- General purpose dynamic translation system.
- Supports: IA32->IA32
- Resembles fastBT in structure
- Requires a low level instruction table from user.
- Overhead : approx. 25-30% average (SPEC INT 2000)

DynamoRIO

- Dynamic Instrumentation system
- Supports IA32 and AMD64 (Windows and Linux)
- Translator extracts and optimizes traces for hot regions
- Maintains basic block cache + trace cache

BinTrans

- Supports: PowerPc to Alpha, i386 to PowerPc and i386 to Alpha
- Supports user level applications only.
- Unit of Translation: Seq. blocks ending at jumps
- Concurrently run interpreter and native generated code, check states.
- Depending on foreign-native pair different reg. allocation and byte ordering strategies are applied.

fastBT

- Generator for DBTs
- Only User Space support
- iA32->ia32
- 0-10% overhead (some exceptions)
- Requires a high level instruction table from user

HDTrans

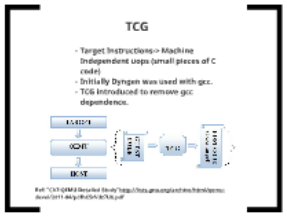
- General purpose dynamic translation system.
- Supports: iA32->iA32
- Resembles fastBT in structure
- Requires a low level instruction table from user.
- Overhead : approx. 25-30% average (SPEC INT 2000)

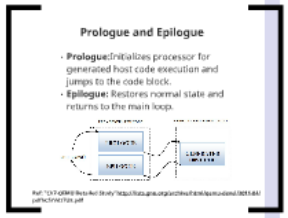
DynamoRIO

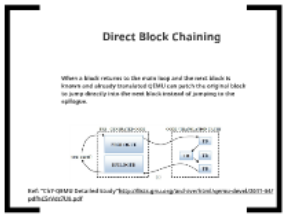
- Dynamic Instrumentation system
- Supports IA32 and AMD64 (Windows and Linux)
- Translator extracts and optimizes traces for hot regions
- Maintains basic block cache + trace cache
-

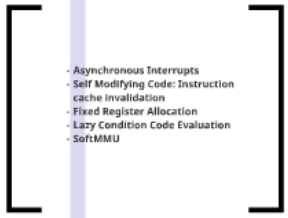
QEMU

- System Mode Emulation
- User Mode Emulation
- Virtualization, Cross Compilation development environments
- Supported OS: Linux, Windows, MacOSx
- Supported ISAs: x86, ARM, MIPS, SPARC, ALPHA
- CPU Emulator (TCG code generator)
- Emulated Devices
- User Interface



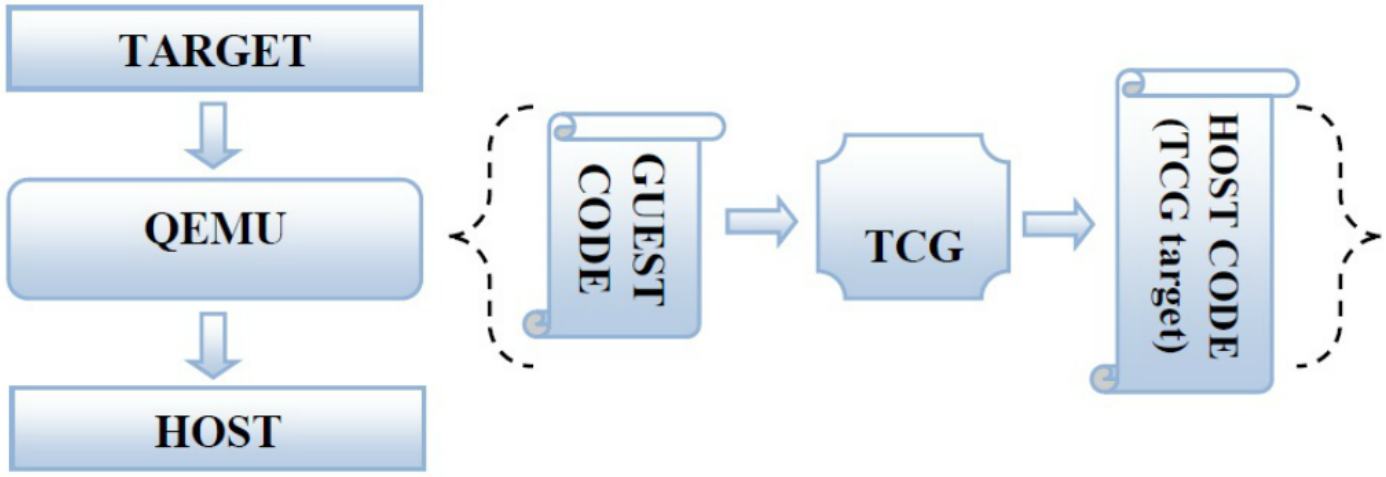






TCG

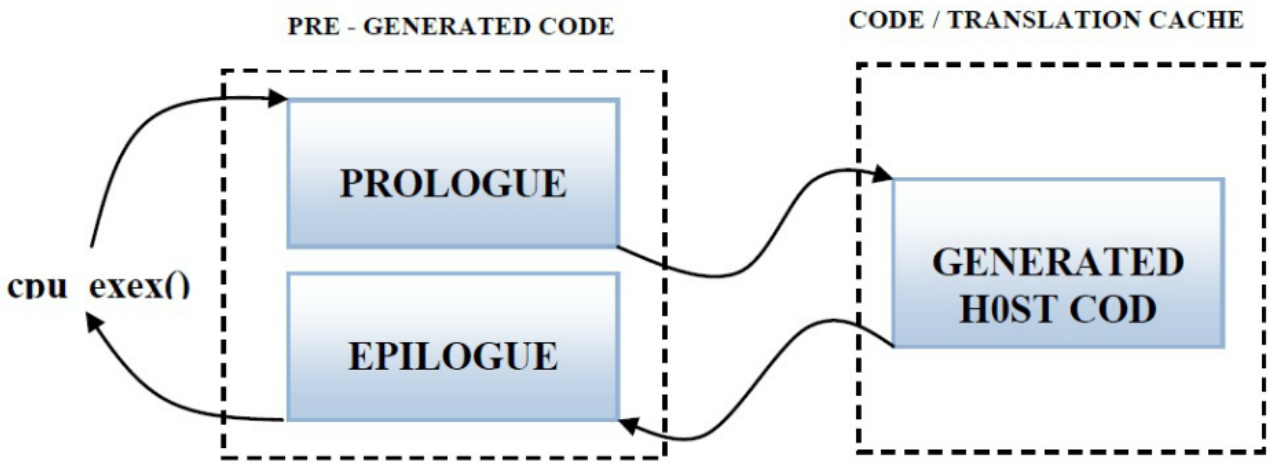
- Target Instructions-> Machine Independent uops (small pieces of C code)
- Initially Dyngen was used with gcc.
- TCG introduced to remove gcc dependence.



Ref: "Ch7-QEMU Detailed Study"http://lists.gnu.org/archive/html/qemu-devel/2011-04/pdfhC5rVdz7U8.pdf

Prologue and Epilogue

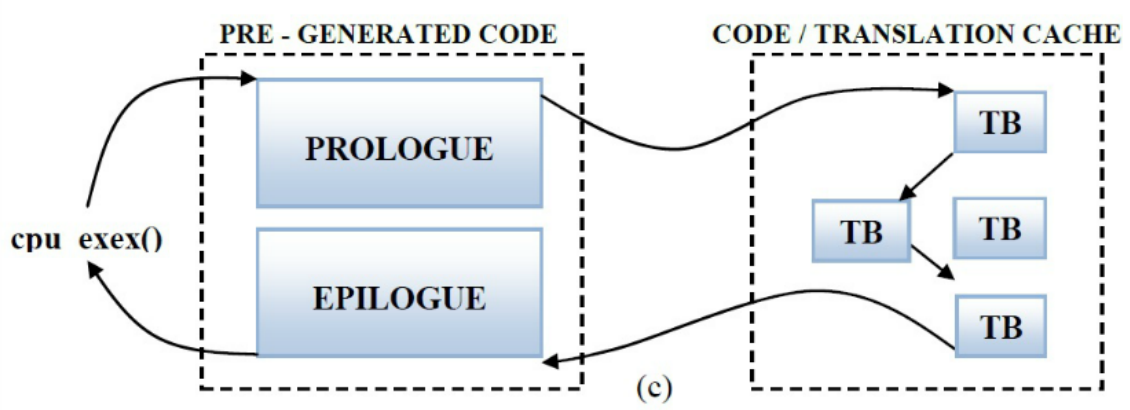
- **Prologue:**Initializes processor for generated host code execution and jumps to the code block.
- **Epilogue:** Restores normal state and returns to the main loop.



Ref: "Ch7-QEMU Detailed Study"http://lists.gnu.org/archive/html/qemu-devel/2011-04/pdfhC5rVdz7U8.pdf

Direct Block Chaining

When a block returns to the main loop and the next block is known and already translated QEMU can patch the original block to jump directly into the next block instead of jumping to the epilogue.



Ref: "Ch7-QEMU Detailed Study"http://lists.gnu.org/archive/html/qemu-devel/2011-04/pdfhC5rVdz7U8.pdf

- Asynchronous Interrupts
- Self Modifying Code: Instruction cache invalidation
- Fixed Register Allocation
- Lazy Condition Code Evaluation
- SoftMMU

Integration of Gem5 and QEMU

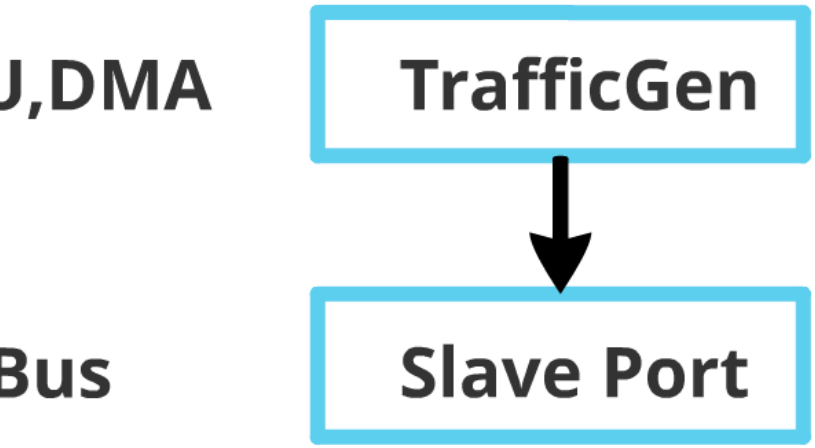
- **Gem5**
- **Resources: Output Assembly log from QEMU, Gem5**
- **Idea1: Reconstruct Host Binary**
- **Idea2: Use Trace Driven Approach**

Reconstructing Host Binary

- Use guest binary to generate a binary for host.
- A new text section will introduce incompatibility.
- Cross ISA translation

Trace Driven Approach

- TraceGen Module
- Uses Google Protobuf format
- Size of traces
- Compatibility of traces



Binary Translation

Issues with Static Binary Translation

- Code Discovery
- Self Modifying Code and Dynamic Linking
- Indirect Branching
- Run time Change of State

Integration of GemS and QEMU

- GemS
- Resources: Output Assembly log from QEMU, GemS
- Idea1: Reconstruct Host Binary
- Idea2: Use Trace Driven Approach

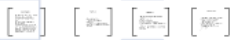
QEMU

- System Mode Emulation
- User Mode Emulation
- Virtualization, Cross Compilation development environments
- Supported OS: Linux, Windows, MacOSX
- Supported ISAs: x86, ARM, MIPS, SPARC,ALPHA
- CPU Emulator (TCG code generator)
- Emulated Devices
- User Interface

Outline

- Introduction of Static and Dynamic BT (Done).
- Solution of problems of SBT by DBT.
- Structure of a Dynamic Binary Translator.
- Brief discussion of few DBT tools.
- Detailed discussion of QEMU (TCG).
- Integration of QEMU with gemS.
- Note: Guest/Source/Foreign -> Host/Target/Native

Dynamic Binary Tranlation Tools



Binary Translation

- "Emulation of one Instruction set by another through translation of binary code."
- Motivation: Running Legacy Code, Cost savings, Server Virtualization, Cross ISA virtual Machines (e.g. VMWare), Application Migration, Better Performance (e.g. Superoptimizer peephole), Memory and Profiling Tools (e.g. Valgrind)
- Types

Dynamic Binary Translation

- Translation on the fly (In time)
- It is easy to do correctly.
- Slower than Static Binary Translation.
- Examples: Hardware/Software
- Software: Rozetta dynamic translation layer (MacOSx), IA-32 Execution Layer-DBT on Itanium based systems.
- Hardware: x86->iops

Dynamic Binary Translator

It's just a host process which runs guest application in its own address space

Static Binary Translation

- Ahead of time.
- Usually fast than its alternative
- Difficult to do Correctly
- Problems: Code Discovery, Dynamic Linking, Self Modifying Code
- Example: A lot of video games have been statically translated, historically.