Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

Pin: building customized program analysis tools with dynamic instrumentation

C.-K. Luk & a bunch of other Intel guys

presented by Bjoern Doebel

Program analysis

Static

GCC, (SP)Lint, Coverity, …

- Compile-time
- Heavy-weight
- No environmental information

Dynamic

Valgrind, Pin, DynamoRIO, …

- Runtime
- Trade-off overhead vs. realistic observations
- Path coverage issues

Source-level

Lint, Coverity

- Exact information
- Problem: 3rd party tools

Binary analysis

Bitblaze/Vine, Valgrind, Pin, …

- Inexact w.r.t. source code
- Support for any kind of application / library

Dynamic binary analysis

- Approach:
  - Disassemble binary (→ intermediate language)
  - Insert tool-specific instrumentation at IL level
  - Recompile into machine code
- Usually JIT-based
  - Disassemble & Resynthesize
  - Copy & Annotate
- Main focus of research: runtime optimizations

Pin: architecture

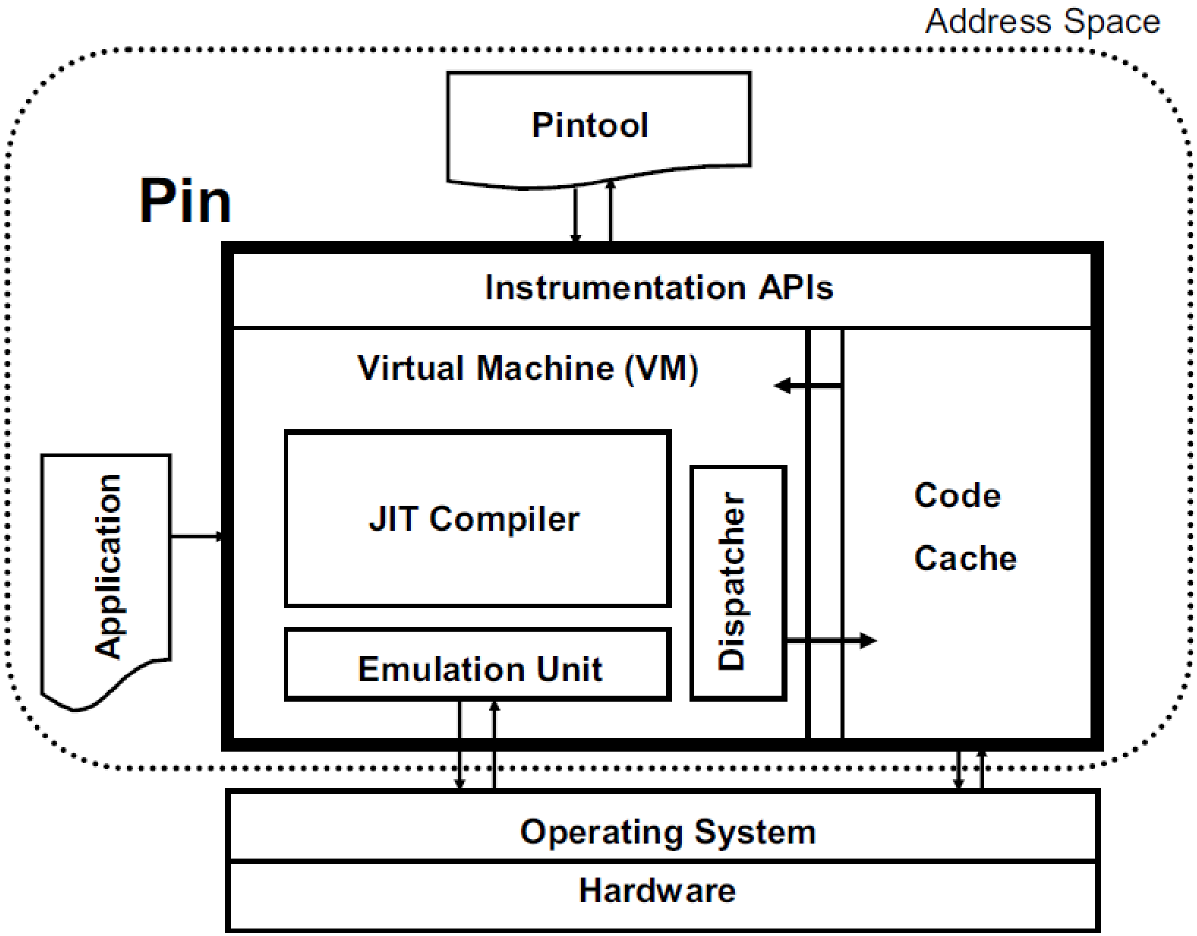
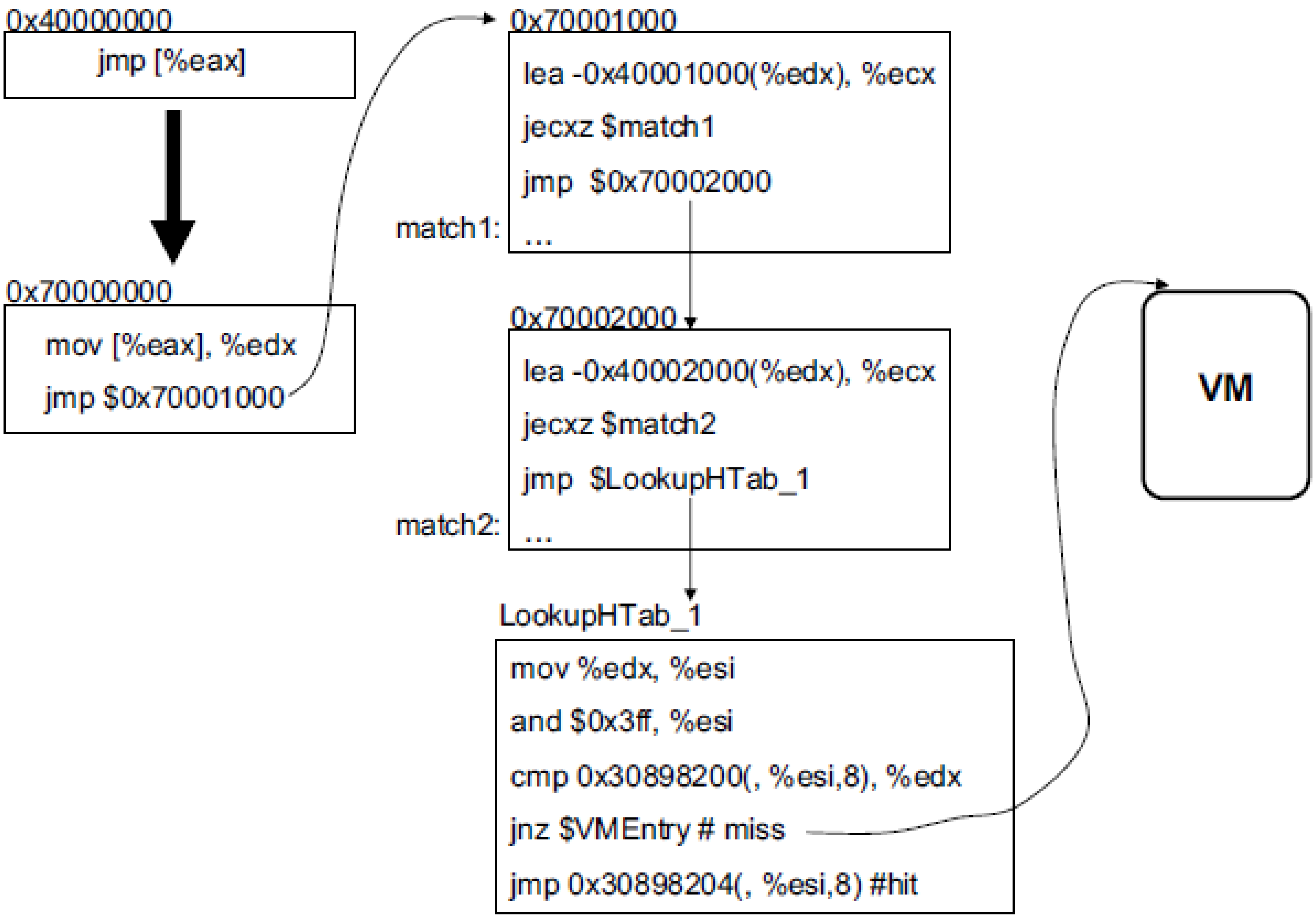


Figure 2. Pin's software architecture

Pin: JIT

- No intermediate language
- Trace-based recompilation
- Can attach to running program

Optimization 1: Trace linking



0x40000000

```
jmp [%eax]
```

0x70000000

```
mov [%eax], %edx
jmp $0x70001000
```

0x70001000

```
        lea -0x40001000(%edx), %ecx
        jecxz $match1
        jmp  $0x70002000
match1: ...
```

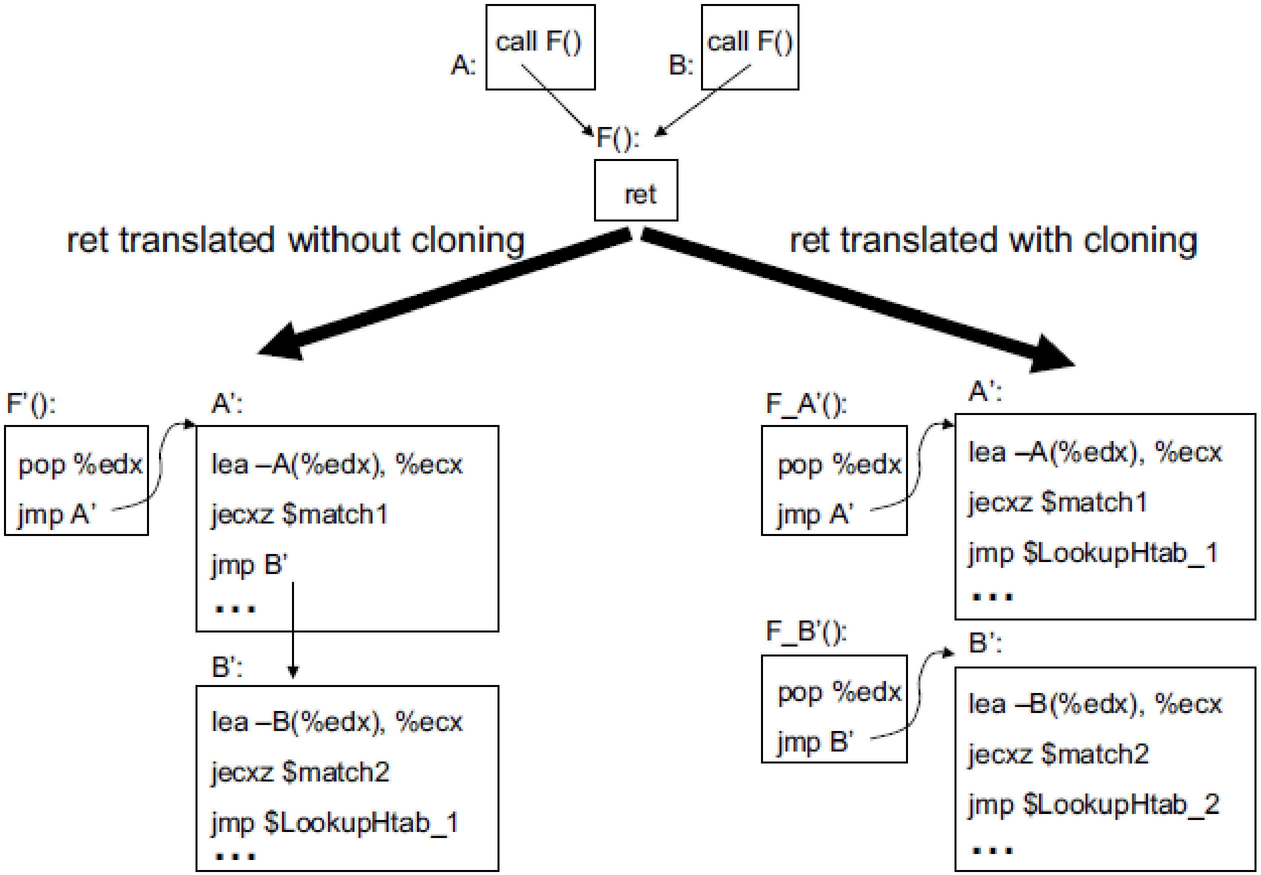
0x70002000

```
        lea -0x40002000(%edx), %ecx
        jecxz $match2
        jmp  $LookupHTab_1
match2: ...
```

LookupHTab_1

```
mov %edx, %esi
and $0x3ff, %esi
cmp 0x30898200(, %esi,8), %edx
jnz $VMEntry # miss
jmp 0x30898204(, %esi,8) #hit
```

Optimization 2: Trace cloning



(b) Using cloning to help predict return targets

F'():
```
pop %edx
jmp A'
```

A':
```
lea –A(%edx), %ecx
jecxz $match1
jmp B'
...
```

B':
```
lea –B(%edx), %ecx
jecxz $match2
jmp $LookupHtab_1
...
```

F_A'():
```
pop %edx
jmp A'
```

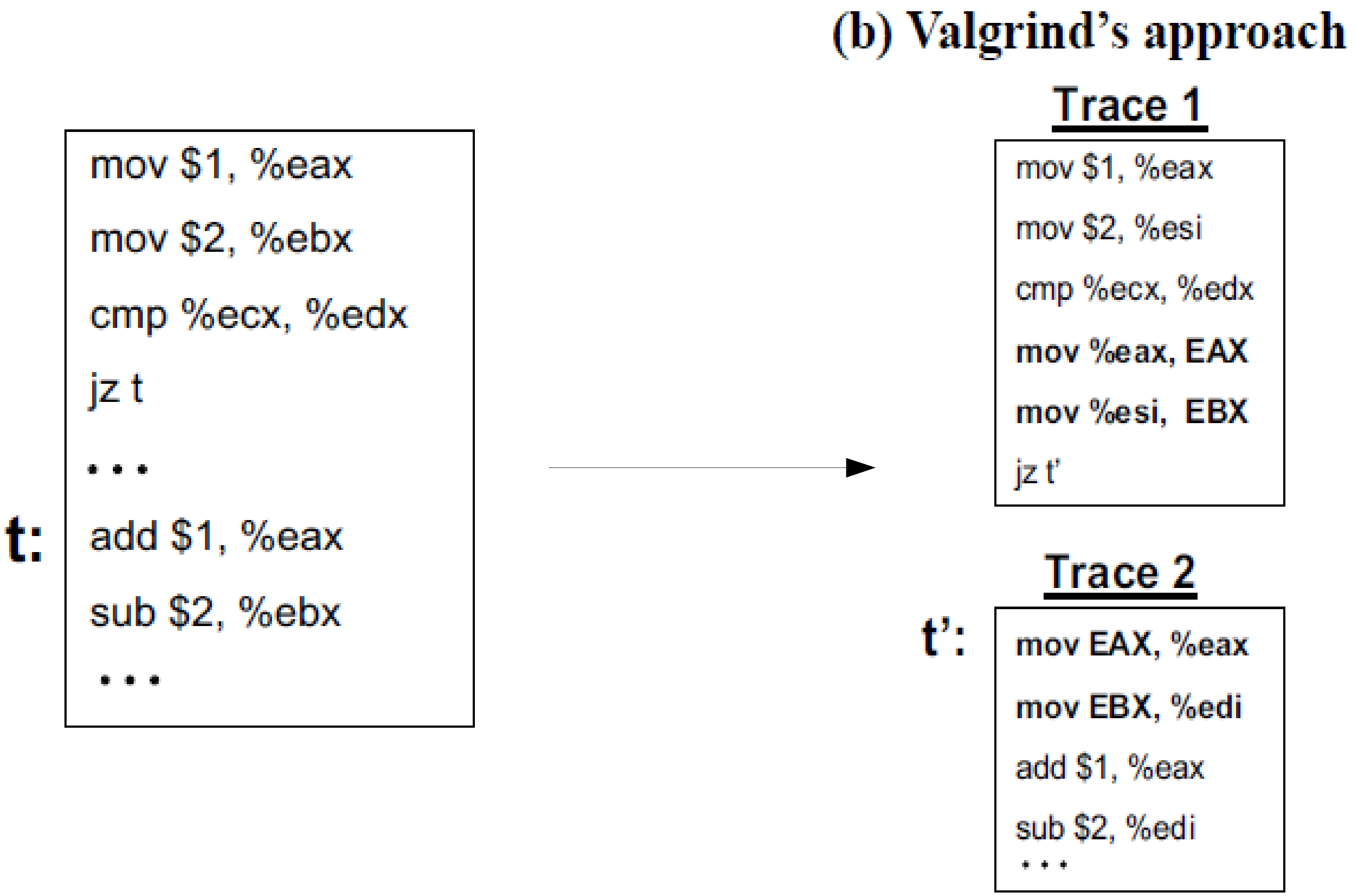
A':
```
lea –A(%edx), %ecx
jecxz $match1
jmp $LookupHtab_1
...
```

F_B'():
```
pop %edx
jmp B'
```

B':
```
lea –B(%edx), %ecx
jecxz $match2
jmp $LookupHtab_2
...
```

Figure 3. Compiling indirect jumps and returns

Optimization 3: Register Re-allocation



(b) Valgrind's approach

```
    mov $1, %eax
    mov $2, %ebx
    cmp %ecx, %edx
    jz t
    . . .
t:  add $1, %eax
    sub $2, %ebx
    . . .
```

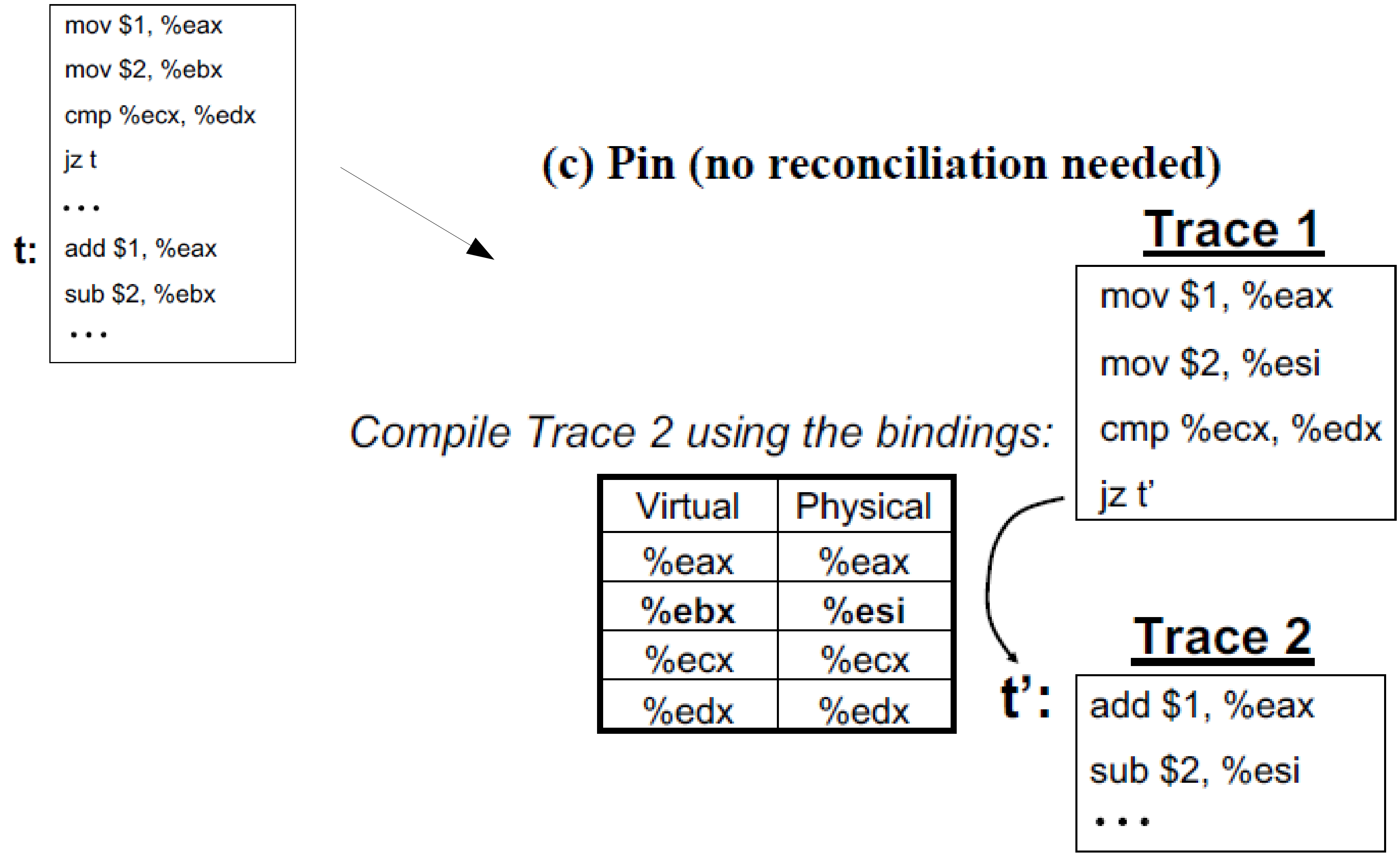
Trace 1

```
    mov $1, %eax
    mov $2, %esi
    cmp %ecx, %edx
    mov %eax, EAX
    mov %esi,  EBX
    jz t'
```

Trace 2

```
t': mov EAX, %eax
    mov EBX, %edi
    add $1, %eax
    sub $2, %edi
    . . .
```

Optimization 3: Register Re-allocation



```
     mov $1, %eax
     mov $2, %ebx
     cmp %ecx, %edx
     jz t
     ...
t:   add $1, %eax
     sub $2, %ebx
     ...
```

(c) Pin (no reconciliation needed)

Compile Trace 2 using the bindings:

Virtual	Physical
%eax	%eax
%ebx	%esi
%ecx	%ecx
%edx	%edx

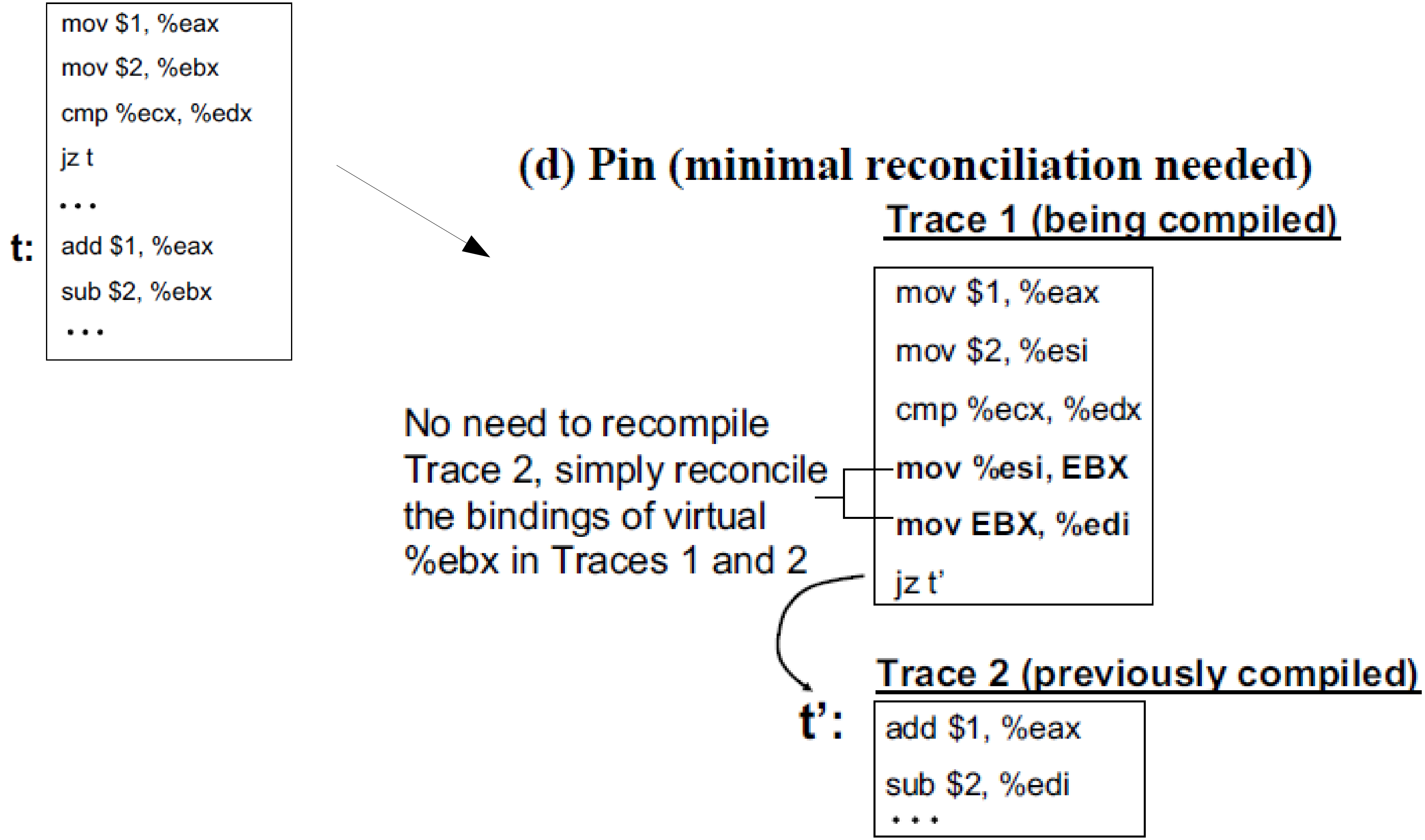
Trace 1

```
     mov $1, %eax
     mov $2, %esi
     cmp %ecx, %edx
     jz t'
```

Trace 2

```
t':  add $1, %eax
     sub $2, %esi
     ...
```

Optimization 3: Register Re-allocation



```
     mov $1, %eax
     mov $2, %ebx
     cmp %ecx, %edx
     jz t
     ...
t:   add $1, %eax
     sub $2, %ebx
     ...
```

(d) Pin (minimal reconciliation needed)

Trace 1 (being compiled)

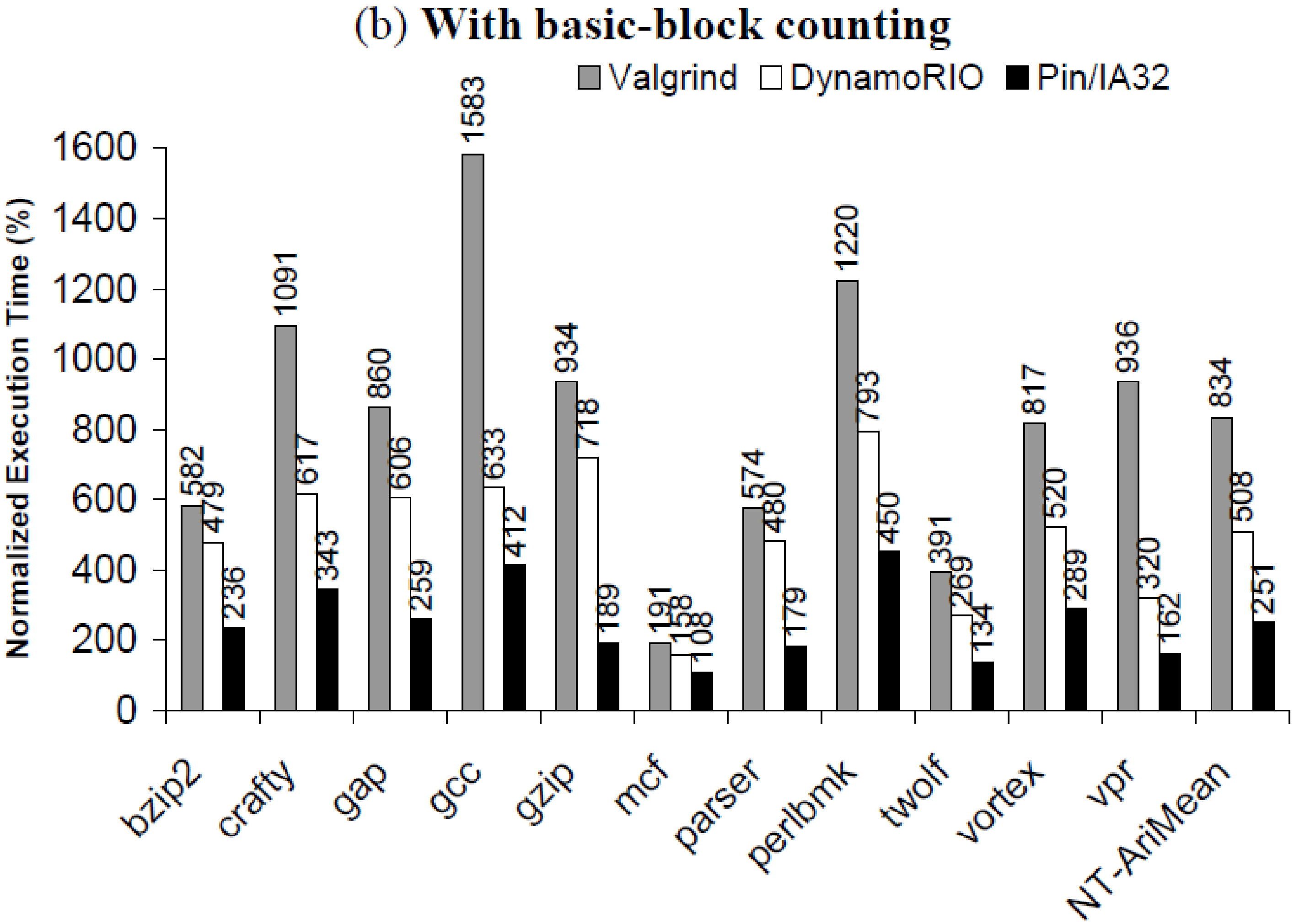
```
mov $1, %eax
mov $2, %esi
cmp %ecx, %edx
mov %esi, EBX
mov EBX, %edi
jz t'
```

No need to recompile Trace 2, simply reconcile the bindings of virtual %ebx in Traces 1 and 2

Trace 2 (previously compiled)

```
t':  add $1, %eax
     sub $2, %edi
     ...
```

Performance evaluation



(b) With basic-block counting

Discussion

- Is there any advantage from having no intermediate language?
  - In fact, the compiler has some kind of intermediate representation, even if it is no language.
- Comparing Pin with Valgrind: How would it perform, if Pin were required to provide
  - Shadow values
  - Address space management
  - System call interception

Performance Comparison with Valgrind

- *"[..] We implemented basic-block counting by modifying a tool in the Valgrind package named lackey [..]"* [1]
  - *"Lackey is a simple Valgrind tool that does various kinds of basic program measurement. It* **adds quite a lot of simple instrumentation** *to the program's code. It is primarily* **intended to be of use as an example tool**, *and consequently* **emphasises clarity of implementation over performance**. *"* [2]

[1] The paper
[2] http://valgrind.org/docs/manual/lk-manual.html