Optimizing Binary Translation of Dynamically Generated Code

Byron Hawkins
Brian Demsky

University of California, Irvine

Derek Bruening
Qin Zhao

Google, Inc.
• Profiling
• Bug detection
• Program analysis
• Security
SPEC CPU 2006

- **DynamoRIO**: 12% overhead*
- **Pin**: 21% overhead*

*geometric mean
SPEC CPU 2006

- DynamoRIO: 12% overhead*
- Pin: 21% overhead*

*geometric mean
Octane JavaScript Benchmark

- 15x overhead on Chrome V8
- 4.4x overhead on Mozilla Ion

- 18x overhead on Chrome V8
- 8x overhead on Mozilla Ion
Octane JavaScript Benchmark

- DynamoRIO
  - 15x overhead on Chrome V8
  - 4.4x overhead on Mozilla Ion

- Pin
  - 18x overhead on Chrome V8
  - 8x overhead on Mozilla Ion
New Era of Dynamic Code

● Back in **2003**...
  - **Browsers**: one single-phase JIT engine
  - **Microsoft Office**: negligible dynamic code

● A decade later...
  - **Browsers**: at least 2 multi-phase JIT engines
  - **Microsoft Office**: one multi-phase JIT
    • Active at startup of all applications
New Era of Dynamic Code

• Back in 2003...
  – **Browsers**: one single-phase JIT engine
  – **Microsoft Office**: negligible dynamic code

• A decade later...
  – **Browsers**: at least 2 multi-phase JIT engines
  – **Microsoft Office**: one multi-phase JIT
    • Active at startup of all applications
Goals

- **Optimize** binary translation of dynamic code
- **Maintain** performance for static code

Evaluation Platform

- **DynamoRIO on 64-bit Linux for x86**
**Goals**

- Optimize binary translation of dynamic code
- Maintain performance for static code

**Evaluation Platform**

- DynamoRIO on 64-bit Linux for x86
Outline

• Background on binary translation
  – Current optimizations for statically compiled code
  – Dynamic code → wasting translation overhead
    • Coarse-grained detection of code changes

• New optimizations
  – Manual annotations
  – Automated inference

• Performance results

• Related Work
Outline

- Background on binary translation
  - Current optimizations for statically compiled code
  - Dynamic code → wasting translation overhead
    - Coarse-grained detection of code changes
- New optimizations
  - Manual annotations
  - Automated inference
- Performance results
- Related Work
Outline

• Background on binary translation
  – Current optimizations for statically compiled code
  – Dynamic code → wasting translation overhead
    • Coarse-grained detection of code changes

• New optimizations
  – Manual annotations
  – Automated inference

• Performance results

• Related Work
Translate application into code cache as it runs
Translate application into code cache as it runs
Translate application into code cache as it runs
Translate application into code cache as it runs
Translate application into code cache as it runs
Correlate indirect branch targets via hashtable
Hot paths are compiled into traces (10% speedup)
Cost

- Translate code
- Build traces

Benefit

- Repeated execution of translated code
- Optimized traces
  - Can beat native performance on SPEC benchmarks
Cost

- Translate code
- Build traces

Benefit

- Repeated execution of translated code
- Optimized traces
  - Can beat native performance on SPEC benchmarks
What if the target code is **dynamically generated**?
The code may be changed frequently at runtime
Indirect Branch Lookup

DynamoRIO Code Cache

Trace Cache

JIT Compiled Function

foo()

bar()

A

B

C'

D'

E

F

A

B

C

D

E

F

A

C

D

E

F

Indirect Branch Lookup

Corresponding translations become invalid
Stale translations must be deleted for retranslation
Stale translations must be deleted for retranslation
→ “cache consistency”
Stale translations must be deleted for retranslation

→ How to detect code changes?
Detecting Code Changes on x86

- Monitor all memory writes
Detecting Code Changes on x86

- Monitor all memory writes
  - Too much overhead!
Detecting Code Changes on x86

- Monitor all memory writes
  - Too much overhead!
- Instrument traces to check freshness
Detecting Code Changes on x86

- Monitor all memory writes
  - Too much overhead!

- Instrument traces to check freshness
  - DynamoRIO supports standalone basic blocks
    → too much overhead!
Detecting Code Changes on x86

- Monitor all memory writes
  - Too much overhead!

- Instrument traces to check freshness
  - DynamoRIO supports standalone basic blocks
  - Too much overhead!

- Leverage page permissions and faults
Detecting Code Changes on x86

- Monitor all memory writes
  - Too much overhead!
- Instrument traces to check freshness
  - DynamoRIO supports standalone basic blocks
    → too much overhead!
- Leverage page permissions and faults
  - Make code pages artificially read-only
Detecting Code Changes on x86

- Monitor all memory writes
  - Too much overhead!
- Instrument traces to check freshness
  - DynamoRIO supports standalone basic blocks
    - too much overhead!
- Leverage page permissions and faults
  - Make code pages artificially read-only
  - Intercept page faults and invalidate translations
Detecting Code Changes on x86

- Monitor all memory writes
  - Too much overhead!

- Instrument traces to check freshness
  - DynamoRIO supports standalone basic blocks
    → too much overhead!

- Leverage page permissions and faults
  - Make code pages artificially read-only
  - Intercept page faults and invalidate translations
    → Acceptable overhead (for rare occurrence)
Detecting Code Changes on x86

- Monitor all memory writes
  - Too much overhead!

- Instrument traces to check freshness
  - DynamoRIO supports standalone basic blocks
    → too much overhead!

- Leverage page permissions and faults
  - Make code pages artificially read-only
  - Intercept page faults and invalidate translations
    → How does this work?
Chrome V8

```
foo()
```

```
bar()
```

```
compile_js()
```

DynamoRIO Code Cache

```
foo()
```

```
bar()
```

```
compile_js()
```

Modify code
Chrome V8

foo()

bar()

compile_js()

DynamoRIO Code Cache

foo()

bar()

compile_js()
### Chrome V8

<table>
<thead>
<tr>
<th>Address</th>
<th>Permissions</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8949ed31</td>
<td>rwx</td>
<td>0xe483e8d2</td>
</tr>
<tr>
<td>0x00401f0f</td>
<td>r-x</td>
<td>0x8468b48</td>
</tr>
<tr>
<td>0x004028c0</td>
<td>rwx</td>
<td>0xffd37e8</td>
</tr>
<tr>
<td>0x0f66f4ff</td>
<td></td>
<td>0x000041ff</td>
</tr>
</tbody>
</table>

#### Allow write

- foo_2()

- bar()

### DynamoRIO Code Cache

<table>
<thead>
<tr>
<th>Address</th>
<th>Permissions</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0401f0f</td>
<td>rwx</td>
<td>0xa83d8348</td>
</tr>
<tr>
<td>0x0002154</td>
<td></td>
<td>0x8c7f400</td>
</tr>
<tr>
<td>0x89480061</td>
<td></td>
<td>0x5dd0ffe5</td>
</tr>
<tr>
<td>0xfff7be9</td>
<td></td>
<td>0x801f0ff</td>
</tr>
<tr>
<td>0x31078b48</td>
<td></td>
<td>0xfef748d2</td>
</tr>
<tr>
<td>0xc3d08948</td>
<td></td>
<td>0x00401f0f</td>
</tr>
<tr>
<td>0x8b48c30f</td>
<td></td>
<td>0x1739416</td>
</tr>
<tr>
<td>0xc3f30674</td>
<td></td>
<td>0x00401f0f</td>
</tr>
<tr>
<td>0x8468b48</td>
<td></td>
<td>0x8473948</td>
</tr>
<tr>
<td>0xc3c940f</td>
<td></td>
<td>0x00401f0f</td>
</tr>
</tbody>
</table>

- bar()

- compile_js()
Chrome V8

```
foo_2()
```

```
bar_2()
```

```
compile_js()
```

DynamoRIO Code Cache

```
bar()
```

```
compile_more_js()
```

Thread A

Thread B

Concurrent modification!

Allow write!

Modify code
**Chrome V8**

- `foo_2()`
  - `rwx`
  - `0x8949ed31 0x8948ed31 0xe48348e2 0x495450f0`
  - `0x1ed0c0c7 0xc7480041 0x411e60c1 0xc74800`
  - `0x004028c0 0xffdc37e8 0x0f66f4ff 0x0000441f`

- `bar_2()`
  - `rwx`
  - `0x004028c0 0xffdc37e8 0x0f66f4ff 0x0000441f`
  - `0x61a5f8b8 0x2d485500 0x0061a5f8 0x0ef8348`
  - `0x77e58948 0xb8c35d02 0x00000000 0x74c08548`
  - `0xf8bf56df4 0xff0061a5 0x80110e0e 0x00000000`
  - `0x81a5f8b8 0x2d485500 0x0061a5f8 0x0f8c1c48`

**DynamoRIO Code Cache**

- `bar()`
  - `r-x`
  - `0x00401f0f 0x3d383348 0x00002154 0x00b81a74 0x48000000`
  - `0x1474c085 0xc9e00b5f55 0x89480616 0x5ddffe9 0x0f77be9`
  - `0x0010f0ff 0xff7e9e9 0x01f0ff 0xf31078b4f 0x1c748e2d2`
  - `0x2d08948 0x00401f0f 0x8b48c331 0x1739c516 0xc3730f74`
  - `0x00401f0f 0x89480616 0x084739e8 0x3c0940f 0x00401f0f`

---

**Concurrent Writer Problem**

All translations from the modified page must be removed.
Cache Consistency Overhead

• For non-JIT modules:
  – System call hooks (program startup only)
  – Self-modifying code (very rare)

• For JIT engines:
  – Code generation
  – Code optimization
  – Code adjustment for reuse
Cache Consistency Overhead

- For non-JIT modules:
  - System call hooks (program startup only)
  - Self-modifying code (very rare)

- For JIT engines:
  - Code generation
  - Code optimization
  - Code adjustment for reuse
Cache Consistency Overhead

Chrome V8

<table>
<thead>
<tr>
<th>r-x</th>
<th>foo()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8944d31</td>
<td>0x948d31</td>
</tr>
<tr>
<td>0x8945d31</td>
<td>0x9485d31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r-x</th>
<th>bar()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8944d37e</td>
<td>0x948d37e</td>
</tr>
<tr>
<td>0x8945d37e</td>
<td>0x9485d37e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r-x</th>
<th>compile js()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8944d3c8</td>
<td>0x948d3c8</td>
</tr>
<tr>
<td>0x8945d3c8</td>
<td>0x9485d3c8</td>
</tr>
</tbody>
</table>

DynamoRIO Code Cache

<table>
<thead>
<tr>
<th>r-x</th>
<th>foo()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xc85d4375</td>
<td>0xc85d4375</td>
</tr>
<tr>
<td>0xc85d4376</td>
<td>0xc85d4376</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r-x</th>
<th>compile js()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xc85d43c8</td>
<td>0xc85d43c8</td>
</tr>
<tr>
<td>0xc85d43c9</td>
<td>0xc85d43c9</td>
</tr>
</tbody>
</table>

JIT writes a second function to unused space in the page

JIT compiles a function
Cache Consistency Overhead

DynamoRIO must invalidate all translations from the page
Cache Consistency Overhead

<table>
<thead>
<tr>
<th>Chrome V8</th>
</tr>
</thead>
<tbody>
<tr>
<td>r-x</td>
</tr>
<tr>
<td>foo()</td>
</tr>
<tr>
<td>0x8940d231 0x89448a0 0x4834a5 0x483450 0x483450</td>
</tr>
<tr>
<td>0x14d8dc7 0x7480d4 0x81468c 0x7480d4 0x7480d4</td>
</tr>
<tr>
<td>bar()</td>
</tr>
<tr>
<td>0x004028d0 0x0066f6ff 0x0066f6ff 0x0066f6ff 0x0066f6ff</td>
</tr>
<tr>
<td>0x8f3c1686 0x8f3c1686 0x8f3c1686 0x8f3c1686 0x8f3c1686</td>
</tr>
<tr>
<td>0x748a6500 0x748a6500 0x748a6500 0x748a6500 0x748a6500</td>
</tr>
<tr>
<td>r-x</td>
</tr>
<tr>
<td>compile_js()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DynamoRIO Code Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo()</td>
</tr>
<tr>
<td>0x34d0d275 0x2880000 0x2880000 0x2880000 0x2880000</td>
</tr>
<tr>
<td>0x2880000 0x2880000 0x2880000 0x2880000 0x2880000</td>
</tr>
<tr>
<td>bar()</td>
</tr>
<tr>
<td>0x001000ff 0x001000ff 0x001000ff 0x001000ff 0x001000ff</td>
</tr>
<tr>
<td>0x001000ff 0x001000ff 0x001000ff 0x001000ff 0x001000ff</td>
</tr>
<tr>
<td>0x001000ff 0x001000ff 0x001000ff 0x001000ff 0x001000ff</td>
</tr>
<tr>
<td>compile_js()</td>
</tr>
</tbody>
</table>

Dynamorio Handles Page Fault

JIT tweaks an instruction

Trivial code changes require flushing all translations
Cache Consistency Overhead

Even a data change requires flushing all translations
Cache Consistency Overhead

- For non-JIT modules:
  - System call hooks (program startup only)
  - Self-modifying code (very rare)

- For JIT engines:
  - Code generation
  - Code optimization
  - Code adjustment for reuse

- Disable traces to reduce translation overhead?
Cache Consistency Overhead

- For non-JIT modules:
  - System call hooks (program startup only)
  - Self-modifying code (very rare)
- For JIT engines:
  - Code generation
  - Code optimization
  - Code adjustment for reuse
- Disable traces to reduce translation overhead?
  
  25% slowdown on Octane!
Conserving DGC Translations

• Add annotation framework to DynamoRIO
  – Macros compile into DynamoRIO hooks
  – Native execution skips hook (2 direct jumps)

• Annotate the target application
  – Specify which memory allocations contain code
  – Notify DynamoRIO of all JIT code writes
Conserving DGC Translations

• Add annotation framework to DynamoRIO
  – Macros compile into DynamoRIO hooks
  – Native execution skips hook (2 direct jumps)

• Annotate the target application
  – Specify which memory allocations contain code
  – Notify DynamoRIO of all JIT code writes
Conserving DGC Translations

- Add annotation framework to DynamoRIO
  - Macros compile to DynamoRIO hooks
  - Native execution skips hook (2 direct jumps)

- Annotate the target application
  - Specify which memory allocations contain code
  - Notify DynamoRIO of all JIT code writes

```c
void* OS::Allocate(const size_t size, int prot) {
    void* mbase = mmap(0, size, prot,
                        MAP_PRIVATE | MAP_ANONYMOUS);
    if (mbase == MAP_FAILED) return NULL;
    if (IS_EXECUTABLE(prot))
        DYNAMORIO_MANAGE_CODE_AREA(mbase, size);
    return mbase;
}
```
Conserving DGC Translations

- Add annotation framework to DynamoRIO
  - Macros compile to DynamoRIO hooks
  - Native execution skips hook (2 direct jumps)

- Annotate the target application
  - Specify which memory allocations contain code
  - Notify DynamoRIO of all JIT code writes
Conserving DGC Translations

- Add annotation framework to DynamoRIO
  - Macros compile to DynamoRIO hooks
  - Native execution skips hook (2 direct jumps)
- Annotate the target application
  - Specify which memory allocations contain code
  - Notify DynamoRIO of all JIT code writes

```c
void CpuFeatures::FlushICache(void* start, size_t size) {
  /* no native action for Intel x86 */
  DYNAMORIO_FLUSH_FRAGMENTS(start, size);
}
```
Conserving DGC Translations

- Add annotation framework to DynamoRIO
  - Macros compile to DynamoRIO hooks
  - Native execution skips hook (2 direct jumps)

- Annotate the target application
  - Specify which memory allocations contain code
  - Notify DynamoRIO of all JIT code writes

→ How does this work?
Annotated JIT Writes

**Chrome V8**

- `foo()`:
  - `rwx`: offset 0x89485e8
  - `r-x`: offset 0x4864e0

- `bar()`:
  - `rwx`: offset 0x04000

**DynamoRIO Code Cache**

- `foo()`:
  - `rwx`: offset 0x354d275
  - `r-x`: offset 0x285400

- `compile_js()`:
  - `r-x`: offset 0x354d275

JIT compiles a function:

**Annotation handler flushes only the written region**
Annotated JIT Writes

Annotation handler flushes only the written basic block
Annotated JIT Writes

Data writes can be safely ignored

JIT writes a data field
Problems with Annotations

- Source code may not be available
- COTS binaries may be preferred
- Application may be difficult to annotate
  - Chrome V8 requires 4 annotations
    - Trivial to place the annotations correctly
  - Mozilla Ion requires 17 annotations
    - Complex analysis required to correctly place annotations
Problems with Annotations

- Source code may not be available
- COTS binaries may be preferred
- Application may be difficult to annotate
  - Chrome V8 requires 4 annotations
    - Trivial to place the annotations correctly
  - Mozilla Ion requires 17 annotations
    - Complex analysis required to correctly place annotations
Inference Approach

• Infer which store instructions are writing code

• Instrument those store instructions
  – Fine-grained cache consistency policy
  – Avoid page faults

• Other writers → default cache consistency
Inference Approach

- Infer which store instructions are writing code
- Instrument those store instructions
  - Fine-grained cache consistency policy
  - Avoid page faults
- Other writers → default cache consistency
- Incorrect inference → negligible overhead
Infer Code-Writing Instructions

Handle up to \(~10\) page faults (heuristic)
Infer Code-Writing Instructions

Flag the faulting page as a **JIT code page**
Parallel Memory Mapping

Map a second writable page to the same location
Conserving DGC Translations

Redirect the store's target to the writable mapping
Conserving DGC Translations

Locate and remove stale translations
### Comparing Approaches

<table>
<thead>
<tr>
<th>Annotations</th>
<th>Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Lower virtual memory usage (for 32-bit)</td>
<td>+ Requires no source code changes</td>
</tr>
<tr>
<td>+ Simple to implement</td>
<td></td>
</tr>
</tbody>
</table>
# Speedup – Octane on V8

<table>
<thead>
<tr>
<th></th>
<th>Score</th>
<th>Overhead</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original DynamoRIO</td>
<td>2,271</td>
<td>15.80x</td>
<td>-</td>
</tr>
<tr>
<td>Annotation DynamoRIO</td>
<td>14,532</td>
<td>2.47x</td>
<td>6.40x</td>
</tr>
<tr>
<td>Inference DynamoRIO</td>
<td>14,257</td>
<td>2.52x</td>
<td>6.28x</td>
</tr>
<tr>
<td>Native</td>
<td>35,889</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Speedup – Octane on V8

<table>
<thead>
<tr>
<th></th>
<th>Score</th>
<th>Overhead</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original DynamoRIO</td>
<td>2,271</td>
<td>15.80x</td>
<td>-</td>
</tr>
<tr>
<td>Annotation DynamoRIO</td>
<td>14,532</td>
<td>2.47x</td>
<td>6.40x</td>
</tr>
<tr>
<td>Inference DynamoRIO</td>
<td>14,257</td>
<td>2.52x</td>
<td>6.28x</td>
</tr>
<tr>
<td>Native</td>
<td>35,889</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Score</td>
<td>Overhead</td>
<td>Speedup</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>Original DynamoRIO</td>
<td>2,271</td>
<td>15.80x</td>
<td>-</td>
</tr>
<tr>
<td>Annotation DynamoRIO</td>
<td>14,532</td>
<td><strong>2.47x</strong></td>
<td>6.40x</td>
</tr>
<tr>
<td>Inference DynamoRIO</td>
<td>14,257</td>
<td><strong>2.52x</strong></td>
<td>6.28x</td>
</tr>
<tr>
<td>Native</td>
<td>35,889</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Speedup – Octane on Ion

<table>
<thead>
<tr>
<th></th>
<th>Score</th>
<th>Overhead</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original DynamoRIO</td>
<td>7,185</td>
<td>4.36x</td>
<td>-</td>
</tr>
<tr>
<td>Annotation DynamoRIO</td>
<td>11,914</td>
<td>2.27x</td>
<td>1.92x</td>
</tr>
<tr>
<td>Inference DynamoRIO</td>
<td>13,797</td>
<td>2.15x</td>
<td>2.03x</td>
</tr>
<tr>
<td>Native</td>
<td>31,340</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
# Speedup – Octane on Ion

<table>
<thead>
<tr>
<th></th>
<th>Score</th>
<th>Overhead</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original DynamoRIO</td>
<td>7,185</td>
<td>4.36x</td>
<td>-</td>
</tr>
<tr>
<td>Annotation DynamoRIO</td>
<td>11,914</td>
<td>2.27x</td>
<td>1.92x</td>
</tr>
<tr>
<td>Inference DynamoRIO</td>
<td>13,797</td>
<td>2.15x</td>
<td>2.03x</td>
</tr>
<tr>
<td>Native</td>
<td>31,340</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Score</td>
<td>Overhead</td>
<td>Speedup</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>Original DynamoRIO</td>
<td>7,185</td>
<td>4.36x</td>
<td>-</td>
</tr>
<tr>
<td>Annotation DynamoRIO</td>
<td>11,914</td>
<td>2.27x</td>
<td>1.92x</td>
</tr>
<tr>
<td>Inference DynamoRIO</td>
<td>13,797</td>
<td>2.15x</td>
<td>2.03x</td>
</tr>
<tr>
<td>Native</td>
<td>31,340</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Speedup – Octane on V8

Average iteration time vs native

Benchmark

- Richards
- DeltaBlue
- Crypto
- RayTrace
- EarleyBoyer
- RegExp
- Splay
- NavierStokes
- zlib
- Mandreel
- Gameboy
- CodeLoad
- Box2D
- PdfJS
- Typescript
- SplayLatency
- MandreelLatency
- Geometric Mean
Benchmark Overhead

Average iteration time vs native

- Pin
- OriginalDR
- InferenceDR

Benchmark
### SPEC CPU 2006

<table>
<thead>
<tr>
<th></th>
<th>SPEC CPU</th>
<th>SPEC Int</th>
<th>SPEC fp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original DynamoRIO</td>
<td>12.27%</td>
<td>17.73%</td>
<td>8.60%</td>
</tr>
<tr>
<td>Inference DynamoRIO</td>
<td>12.35%</td>
<td>17.88%</td>
<td>8.60%</td>
</tr>
</tbody>
</table>

Performance is maintained for programs that do not dynamically generate code.
## Related Work

<table>
<thead>
<tr>
<th>Platform</th>
<th>Cache Consistency Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynamoRIO</td>
<td>Page protection → flush all</td>
</tr>
<tr>
<td>Pin</td>
<td>Instrument trace heads → flush trace</td>
</tr>
<tr>
<td>QEMU</td>
<td>Software TLB hook → flush basic block</td>
</tr>
<tr>
<td>Valgrind</td>
<td>Instrument basic block → flush</td>
</tr>
<tr>
<td>Transmeta</td>
<td>Page protection → flush region (HW support)</td>
</tr>
<tr>
<td>Librando</td>
<td>Page protection → hash compare and flush</td>
</tr>
</tbody>
</table>
Conclusion

• New trend towards dynamic code generation
• We explore two optimization approaches
  – Annotation-driven code change notification
  – Code change inference
• We improve performance of binary translation on the Octane benchmark by more than 6x
DynamoRIO Annotations

jmp 401985  # first jump
mov 0x202688,%rax  # name base
bsf 0xffffffffffffffff98,%rax  # name offset
jmp 401996  # second jump
mov $0x2,%esi  # argument 2
mov $0x1,%edi  # argument 1
callq 4024f2  # call annotation

Compiled annotation in x64 Linux
DynamoRIO Annotations

jmp 401985  # first jump
mov 0x202688,%rax  # name base
bsf 0xffffffffffffffff98,%rax  # name offset
jmp 401996  # second jump
mov $0x2,%esi  # argument 2
mov $0x1,%edi  # argument 1
callq 4024f2  # call annotation

Native execution jumps over the annotation
DynamoRIO transforms the annotation into a handler call.
DynamoRIO transforms the annotation into a handler call:

```c
dr_handle_annotation(name, 1, 2);
```