Building Customized Dynamic Program Inspectors

Derek Bruening
Qin Zhao

Motivation

Profile, monitor, or inspect application binaries as they run

- Build *customized dynamic program inspectors*

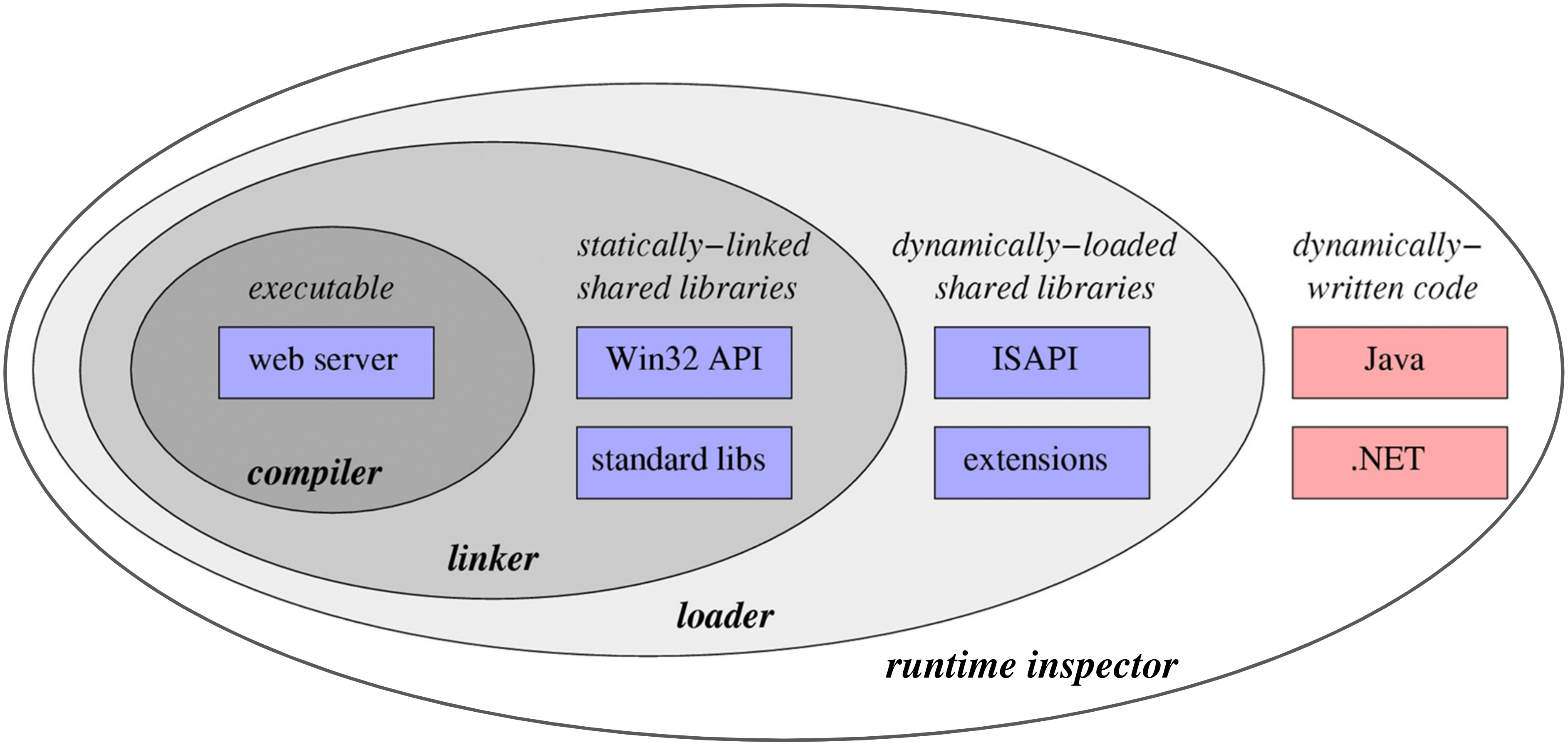
Target production workloads

- Profile or inspect actual deployed application with no overhead when not in inspection mode

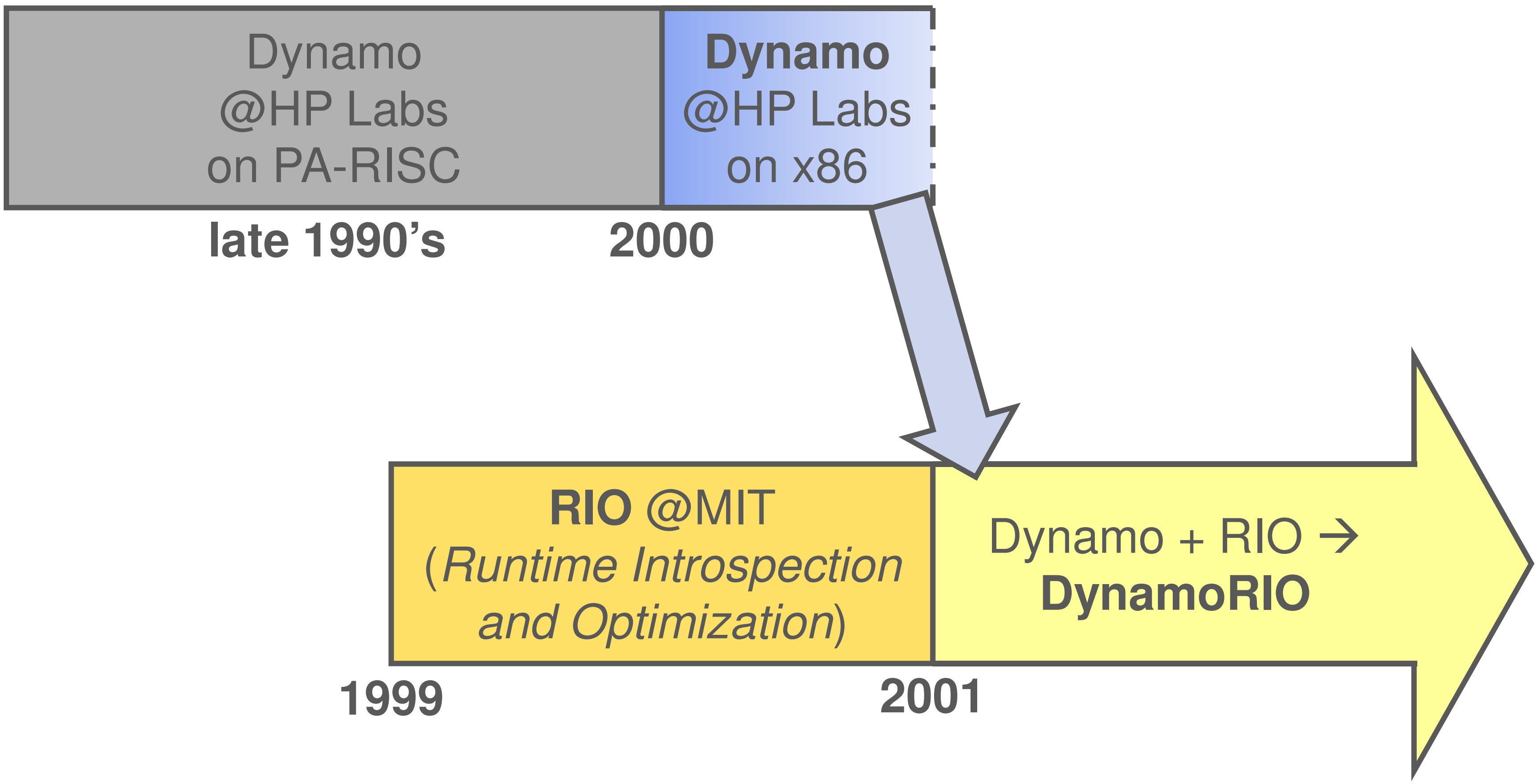
Target applications that include legacy components, third-party libraries, or dynamically-generated code

- Want to inspect whole program even if cannot recompile it all

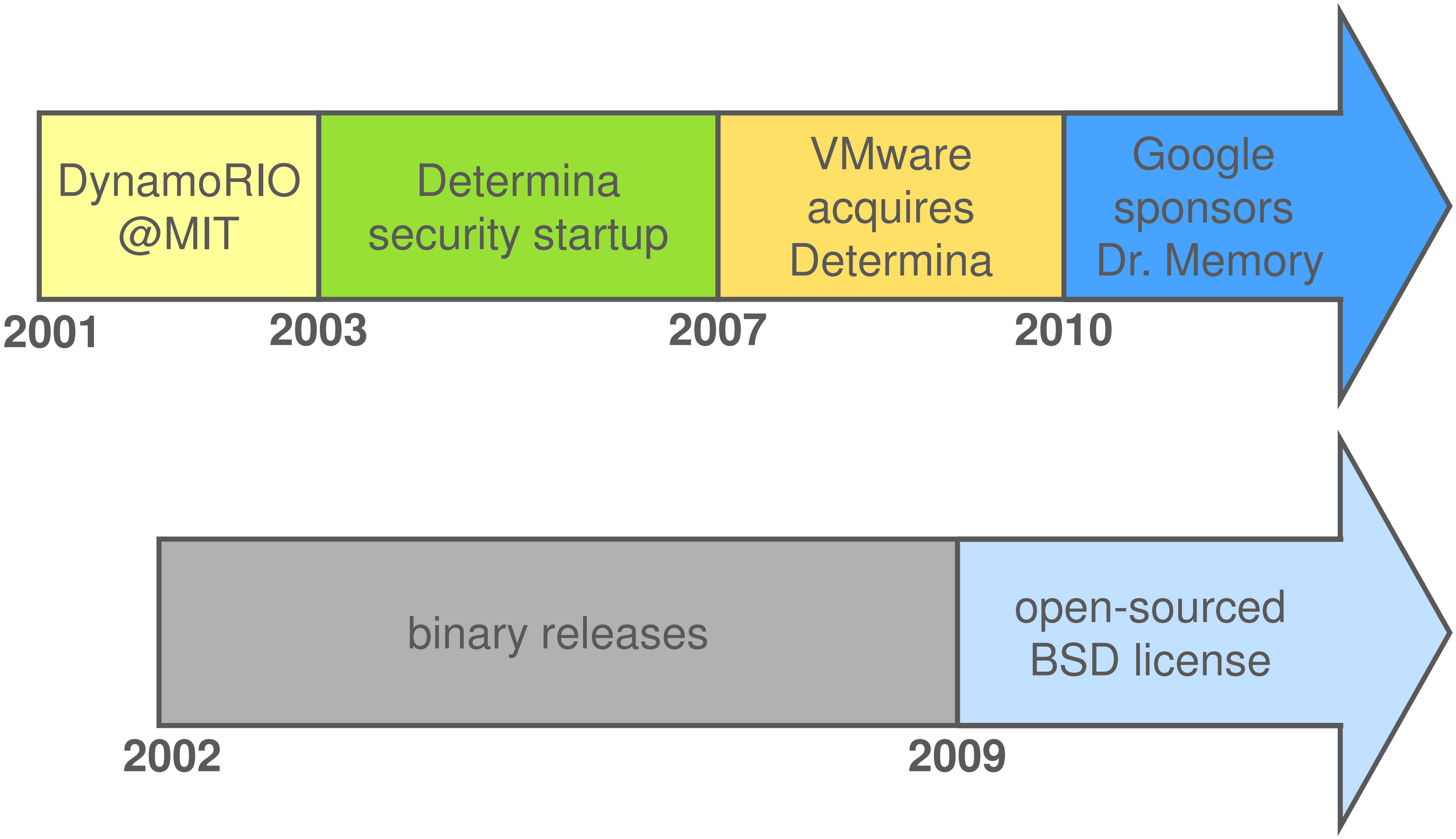
Reach of Toolchain Control Points



DynamoRIO


Dynamo
@HP Labs
on PA-RISC
Dynamo
@HP Labs
on x86
late 1990's
2000
RIO @MIT
(Runtime Introspection
and Optimization)
Dynamo + RIO →
DynamoRIO
1999
2001

DynamoRIO History



DynamoRIO Tool Platform Design Goals

Google

Efficient

- Near-native performance

Transparent

- Match native behavior

Comprehensive

- Control every instruction, in any application

Customizable

- Adapt to satisfy disparate tool needs

Outline

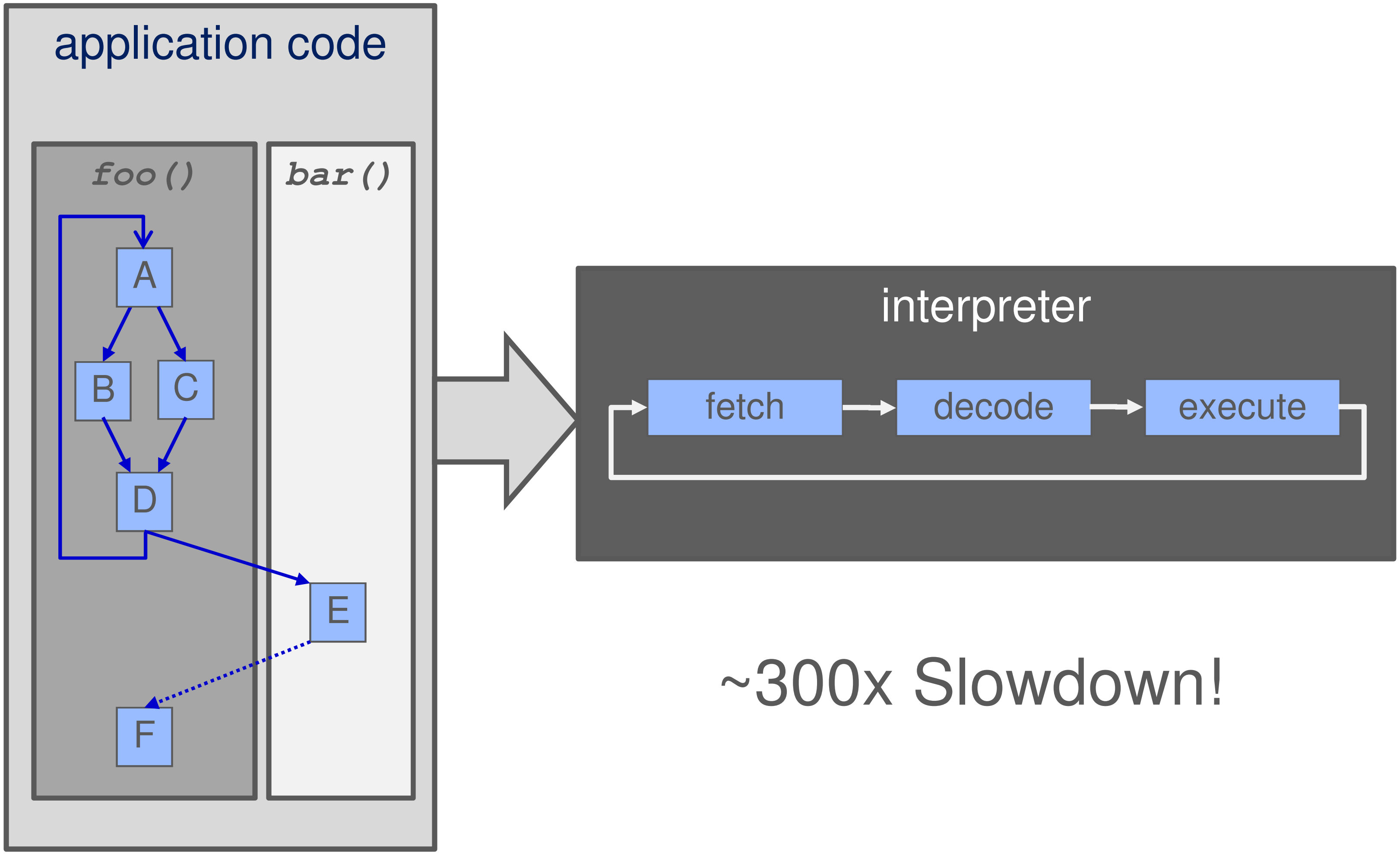
Base System: DynamoRIO

- Efficient
- Transparent
- Comprehensive
- Customizable

Dynamic Program Inspectors

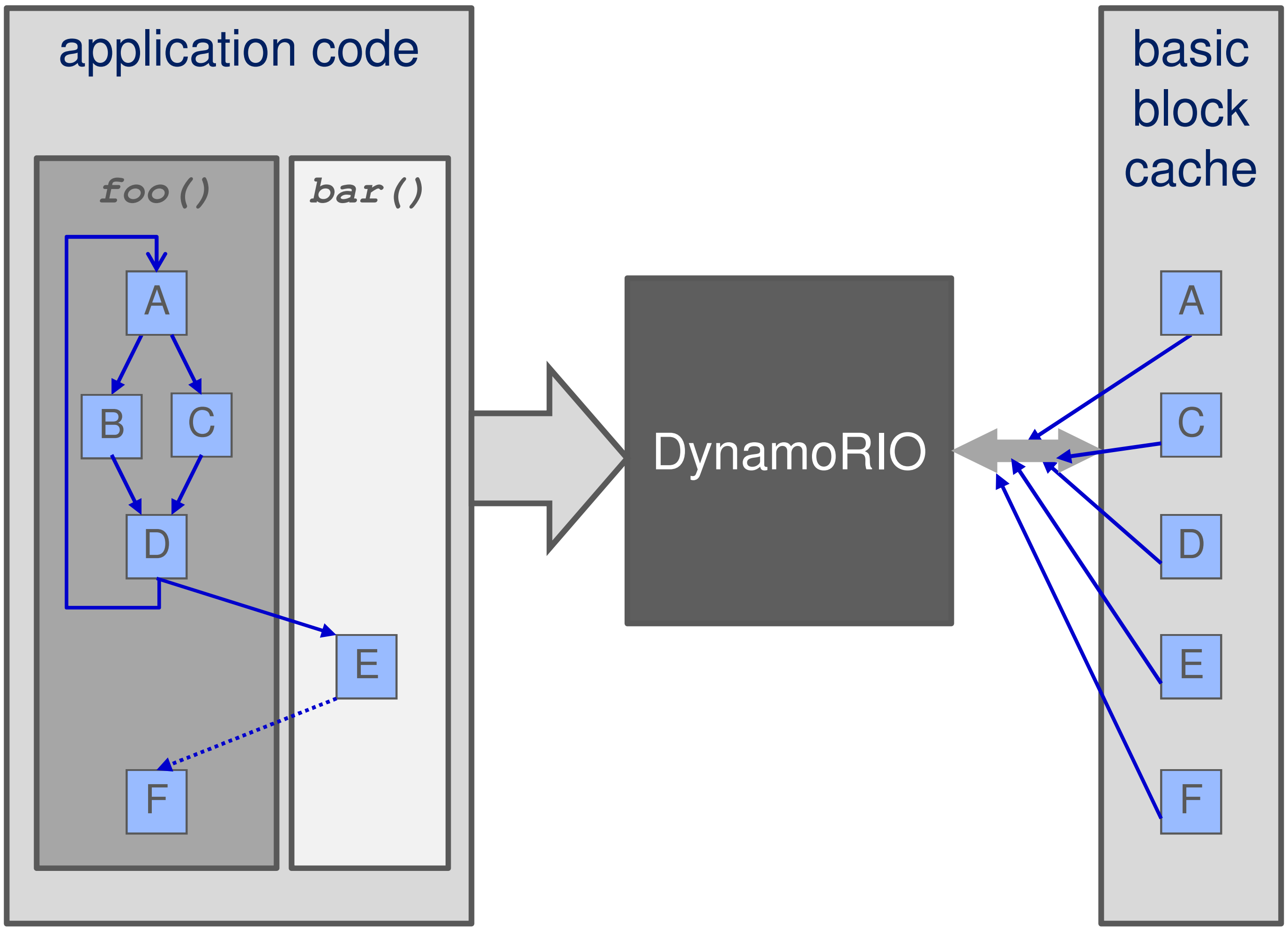
- Examples and Possibilities
- Case studies

Basic Interpreter



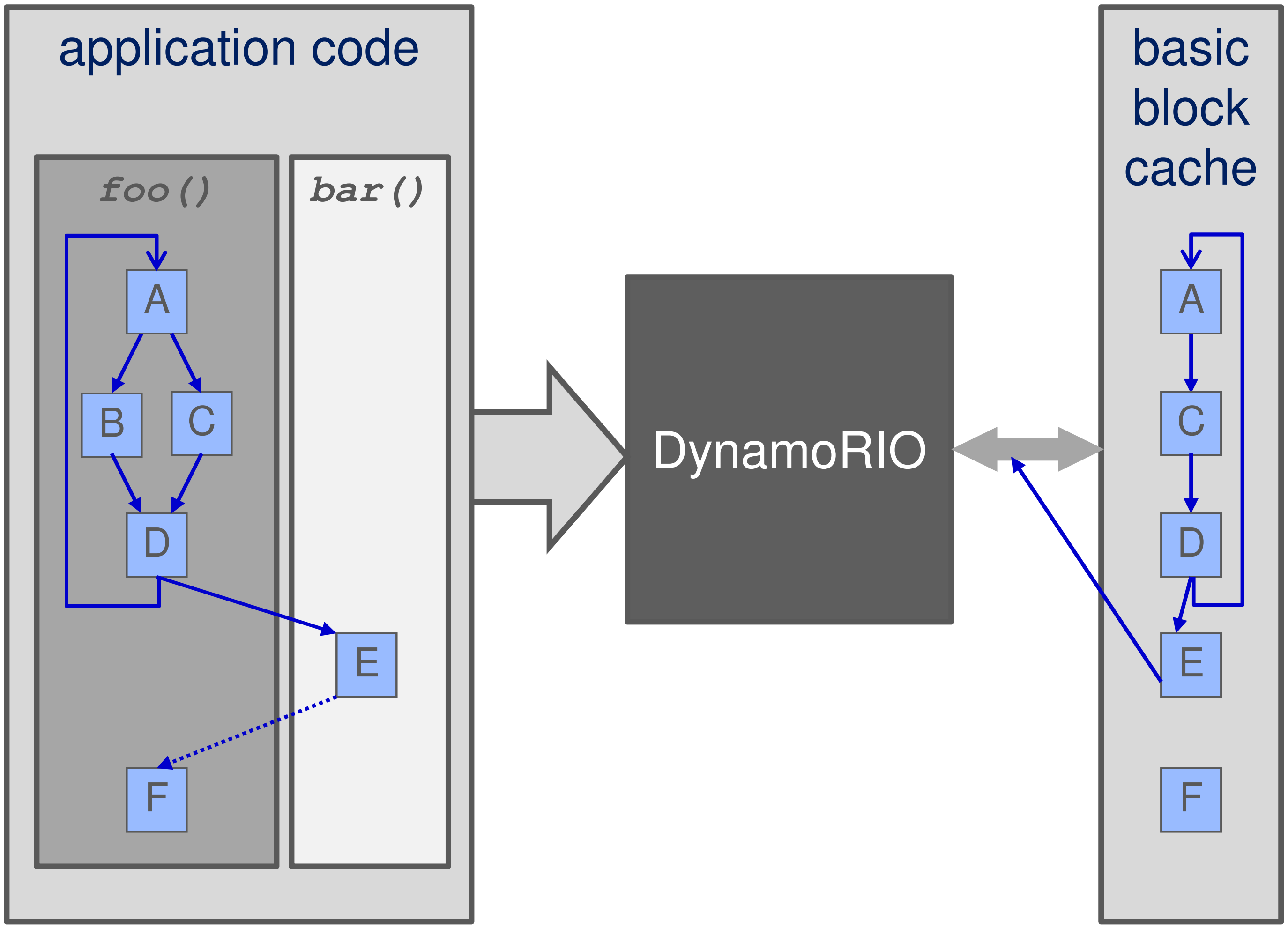
~300x Slowdown!

Improvement #1: Basic Block Cache



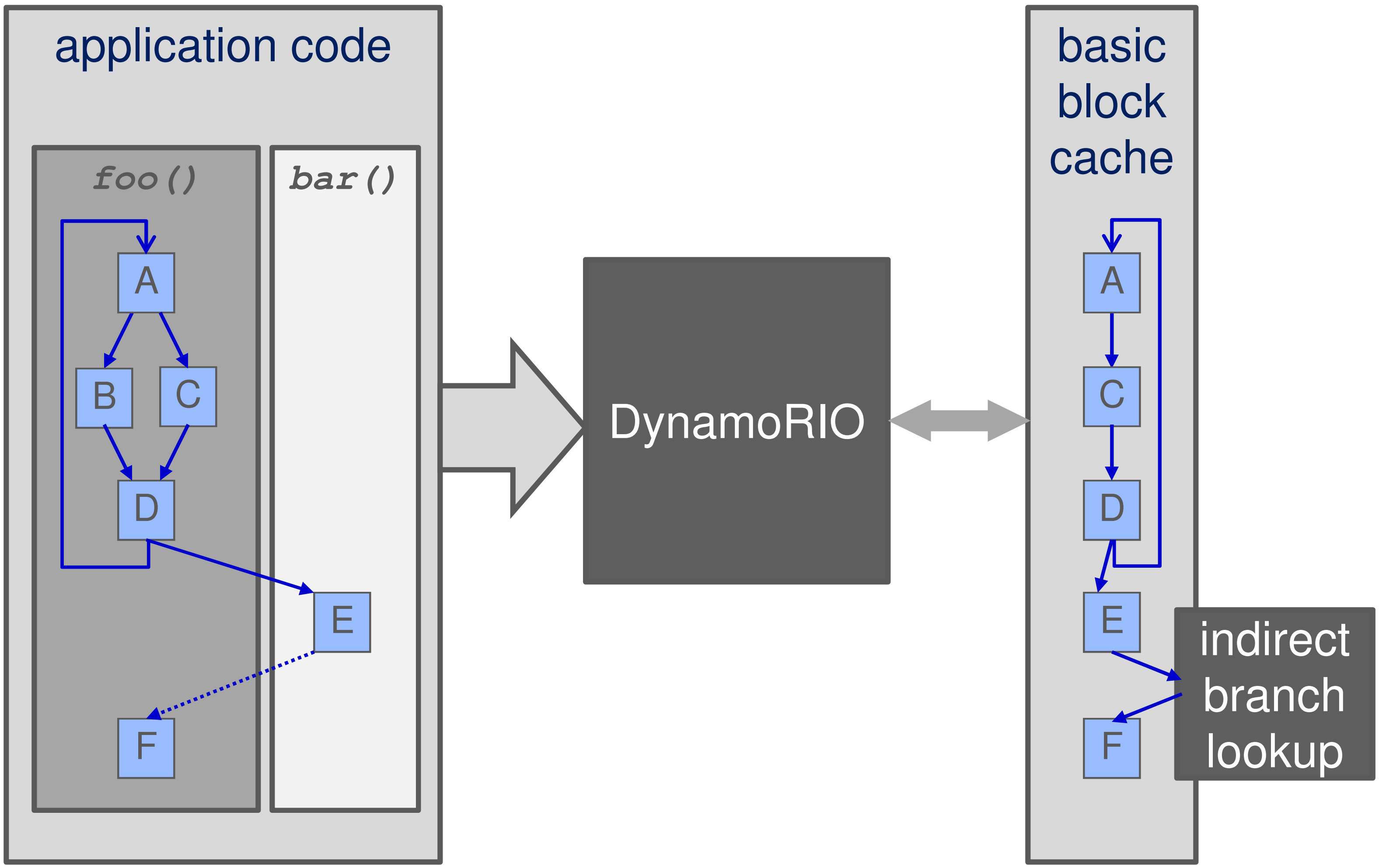
Slowdown: ~~300x~~ 25x

Improvement #2: Linking Direct Branches



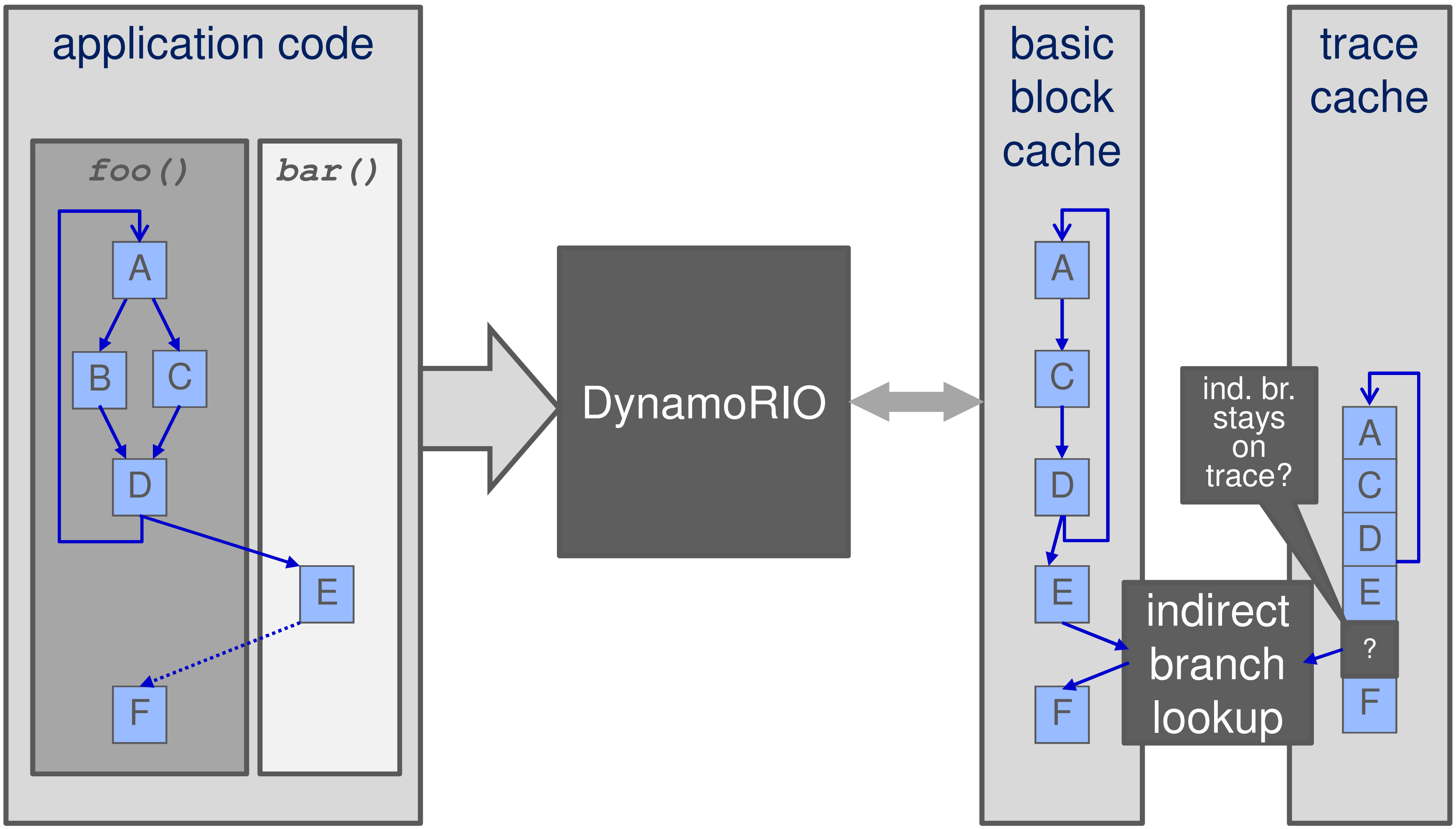
Slowdown: ~~300x~~ ~~25x~~ 3x

Improvement #3: Linking Indirect Branches



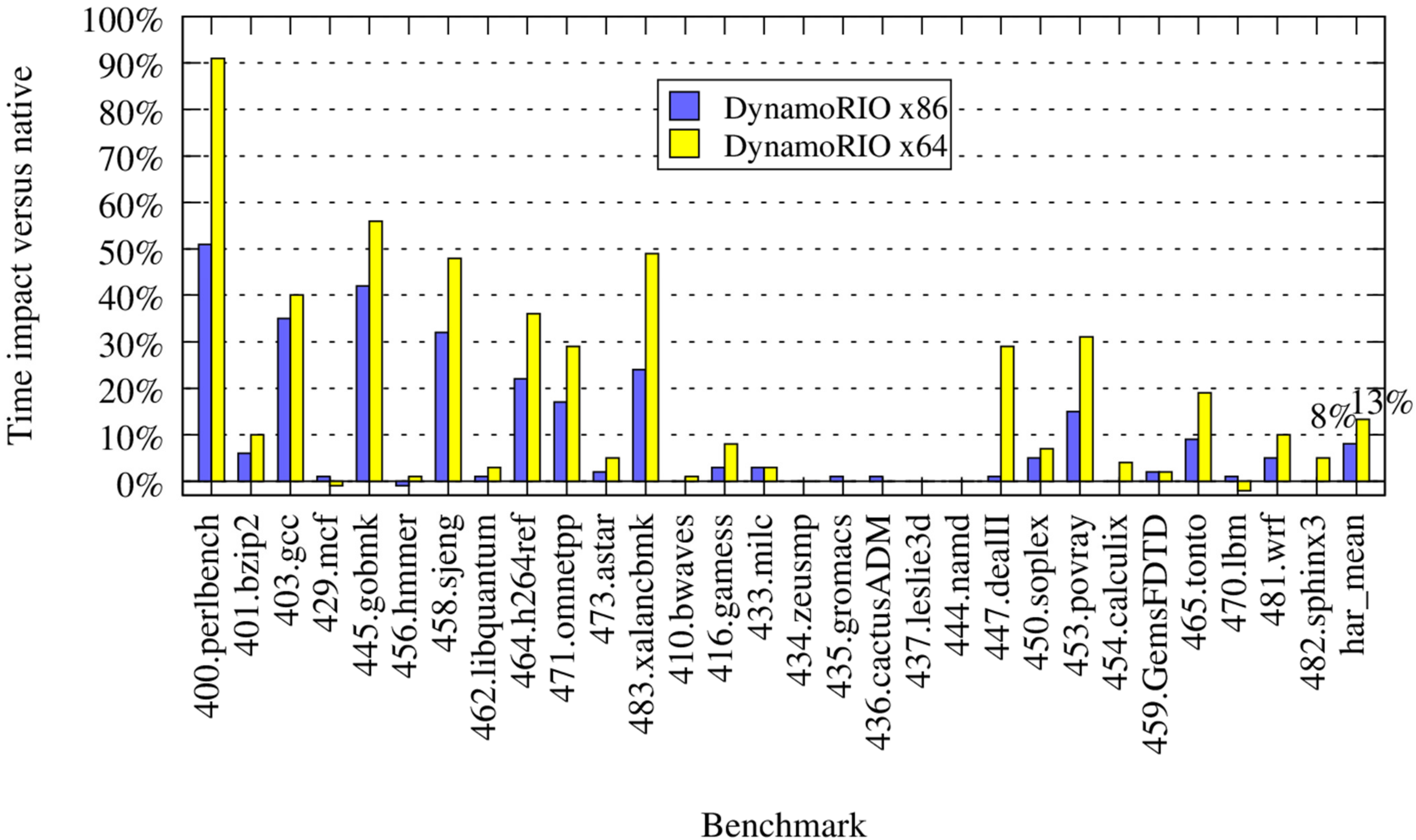
Slowdown: ~~300x~~ ~~25x~~ ~~3x~~ 1.2x

Improvement #4: Trace Building

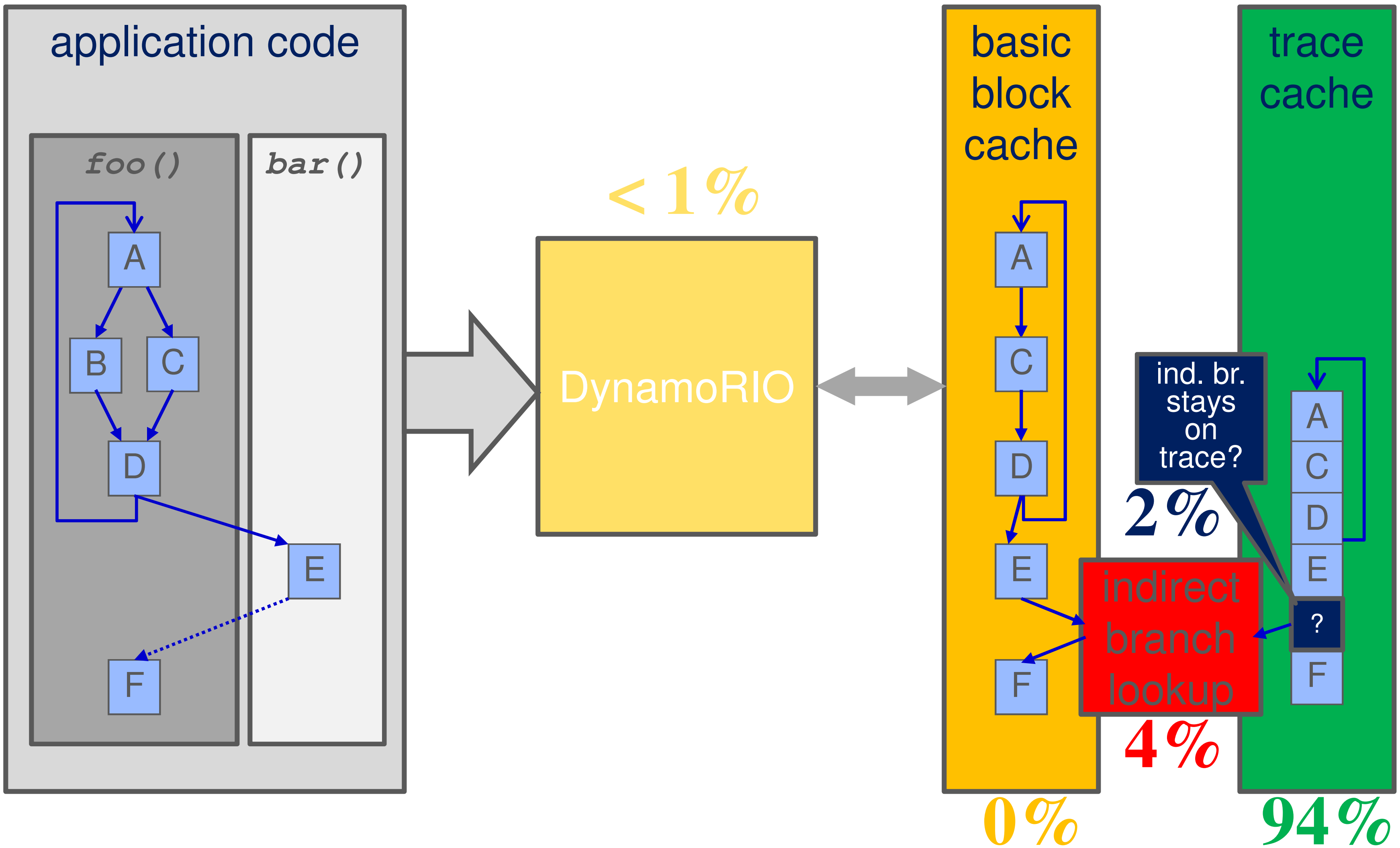


Slowdown: ~~300x~~ ~~25x~~ ~~3x~~ ~~1.2x~~ 1.1x

Base Performance: SPEC 2006



Time Breakdown for SPEC CPU INT



Outline

Google

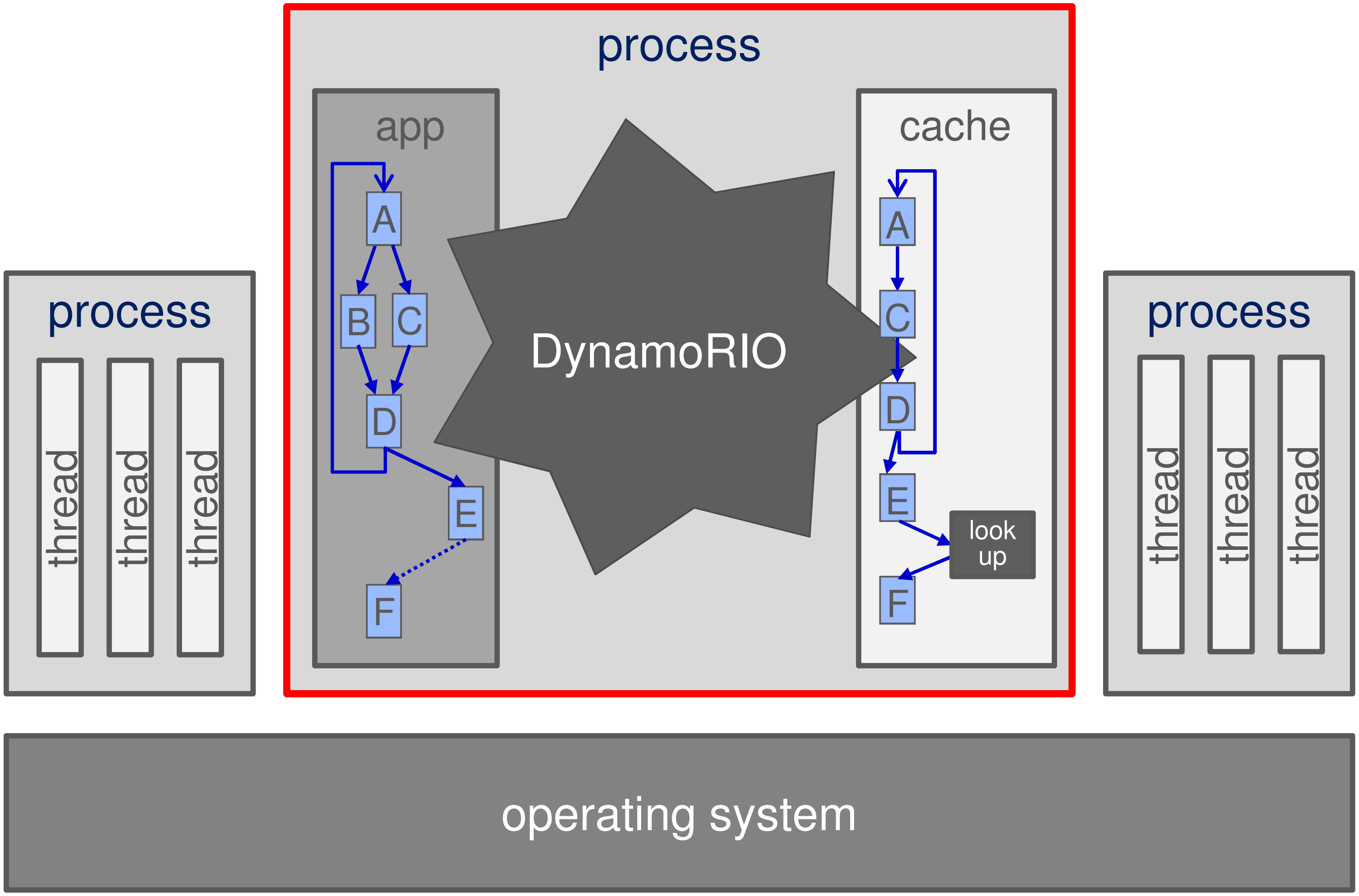
Base System: DynamoRIO

- Efficient
- Transparent
- Comprehensive
- Customizable

Dynamic Program Inspectors

- Examples and Possibilities
- Case studies

Unavoidably Intrusive



Outline

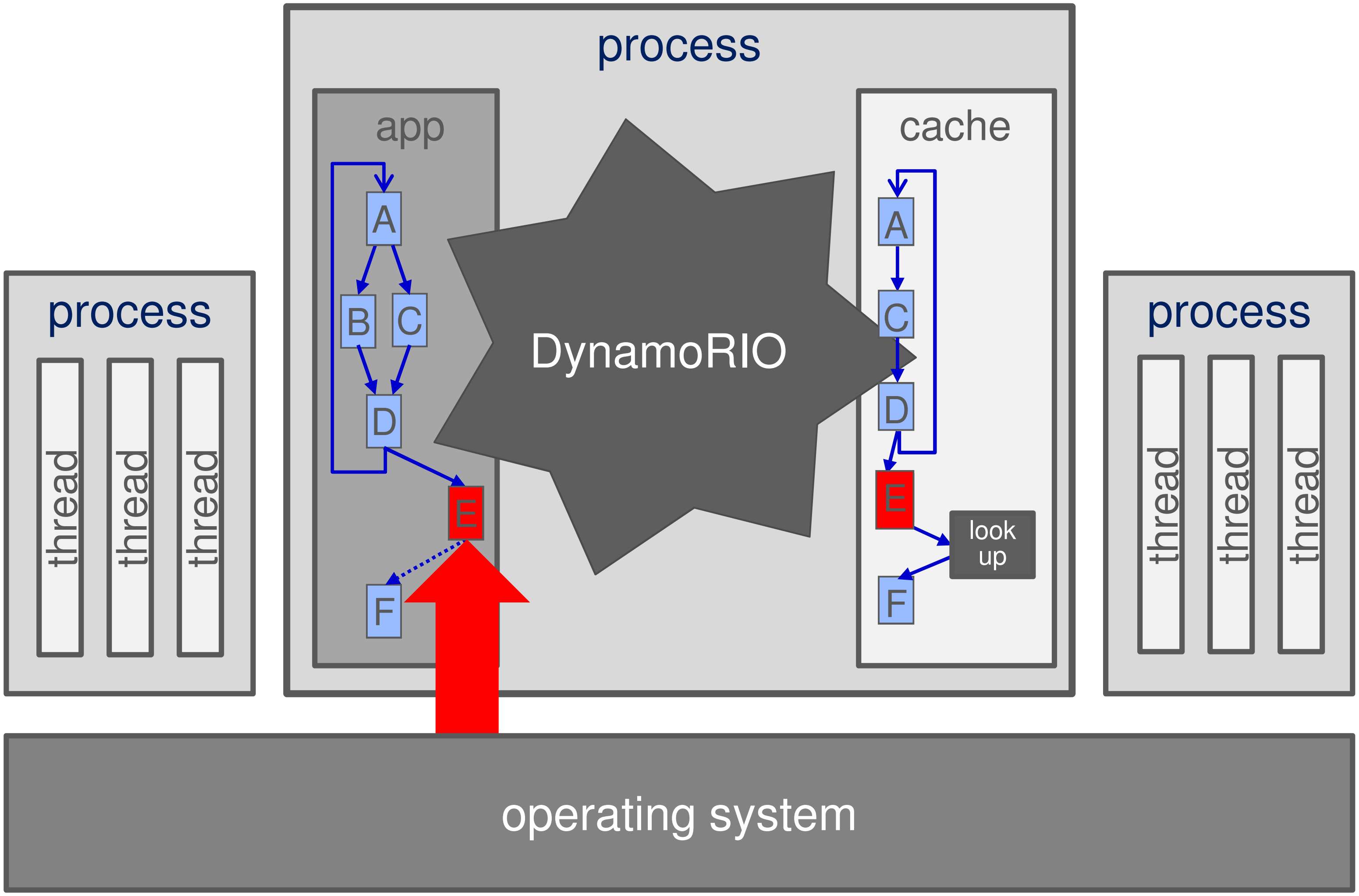
Base System: DynamoRIO

- Efficient
- Transparent
- Comprehensive
- Customizable

Dynamic Program Inspectors

- Examples and Possibilities
- Case studies

Above the Operating System



Outline

Google

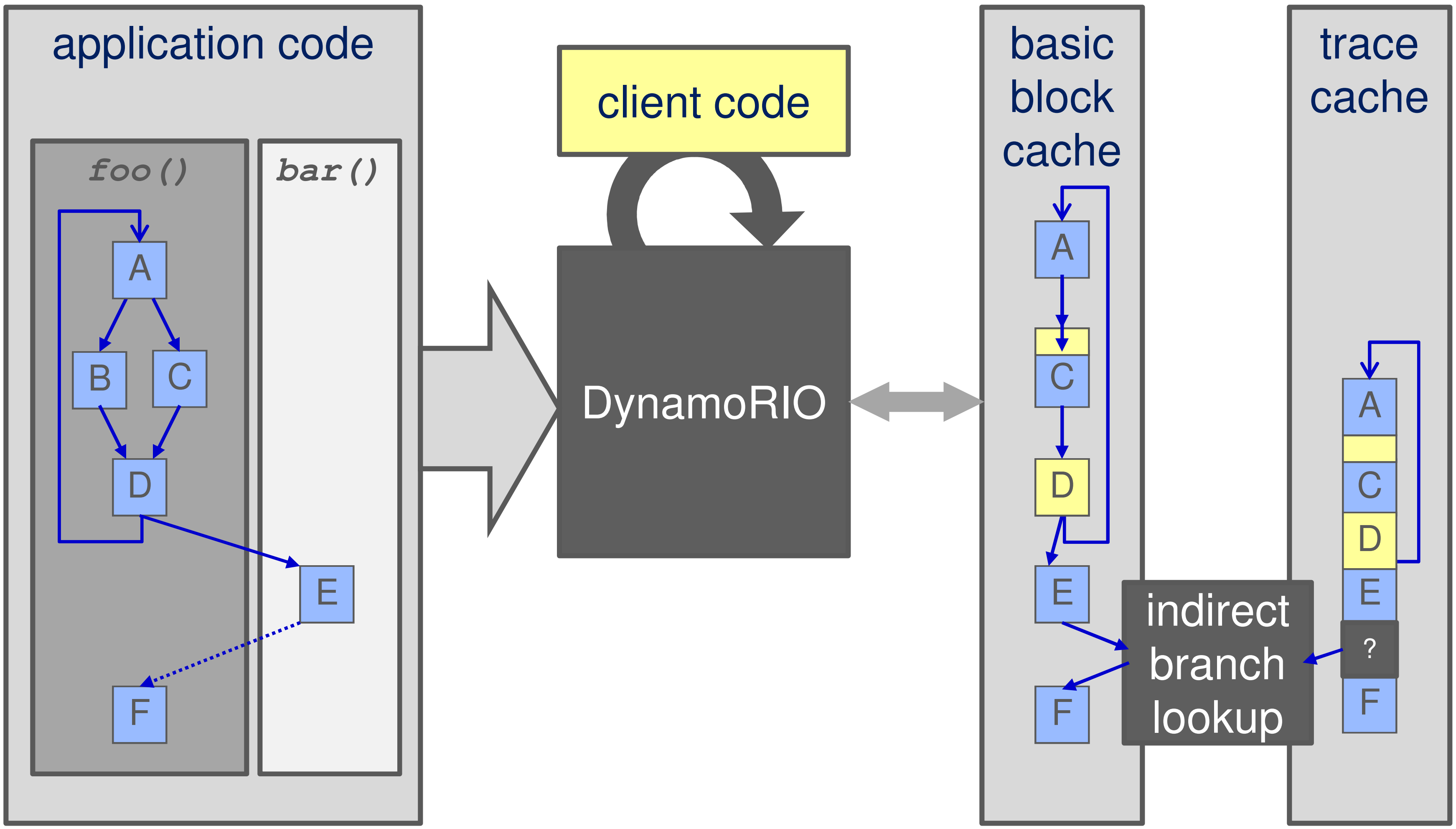
Base System: DynamoRIO

- Efficient
- Transparent
- Comprehensive
- Customizable

Dynamic Program Inspectors

- Examples and Possibilities
- Case studies

DynamoRIO + Client ➡ Program Inspector



Primary Client Events: Code Stream

Client has opportunity to inspect and potentially modify every single application instruction, immediately before it executes

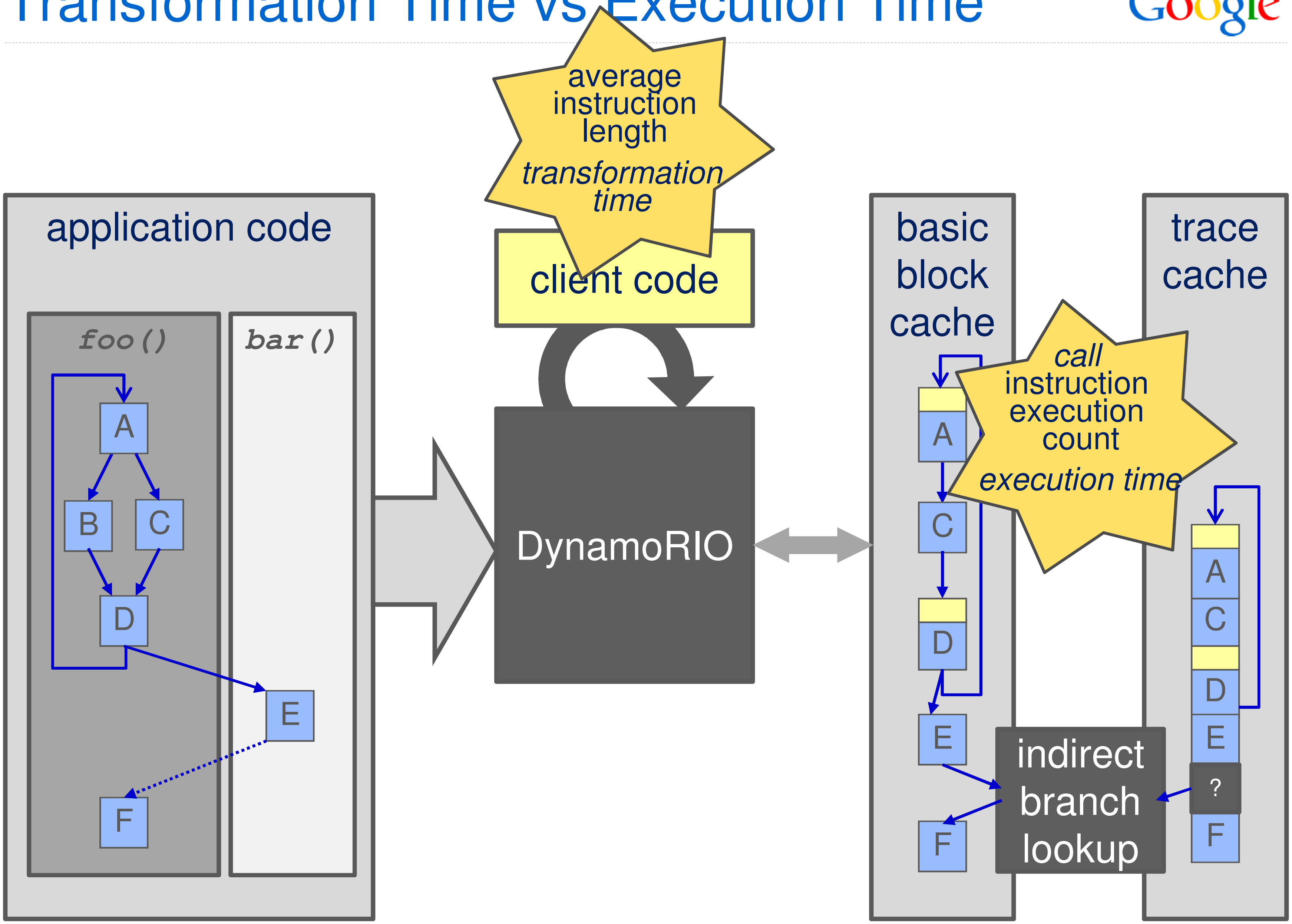
Entire application code stream

- Basic block creation event: can modify the block
- For comprehensive instrumentation tools

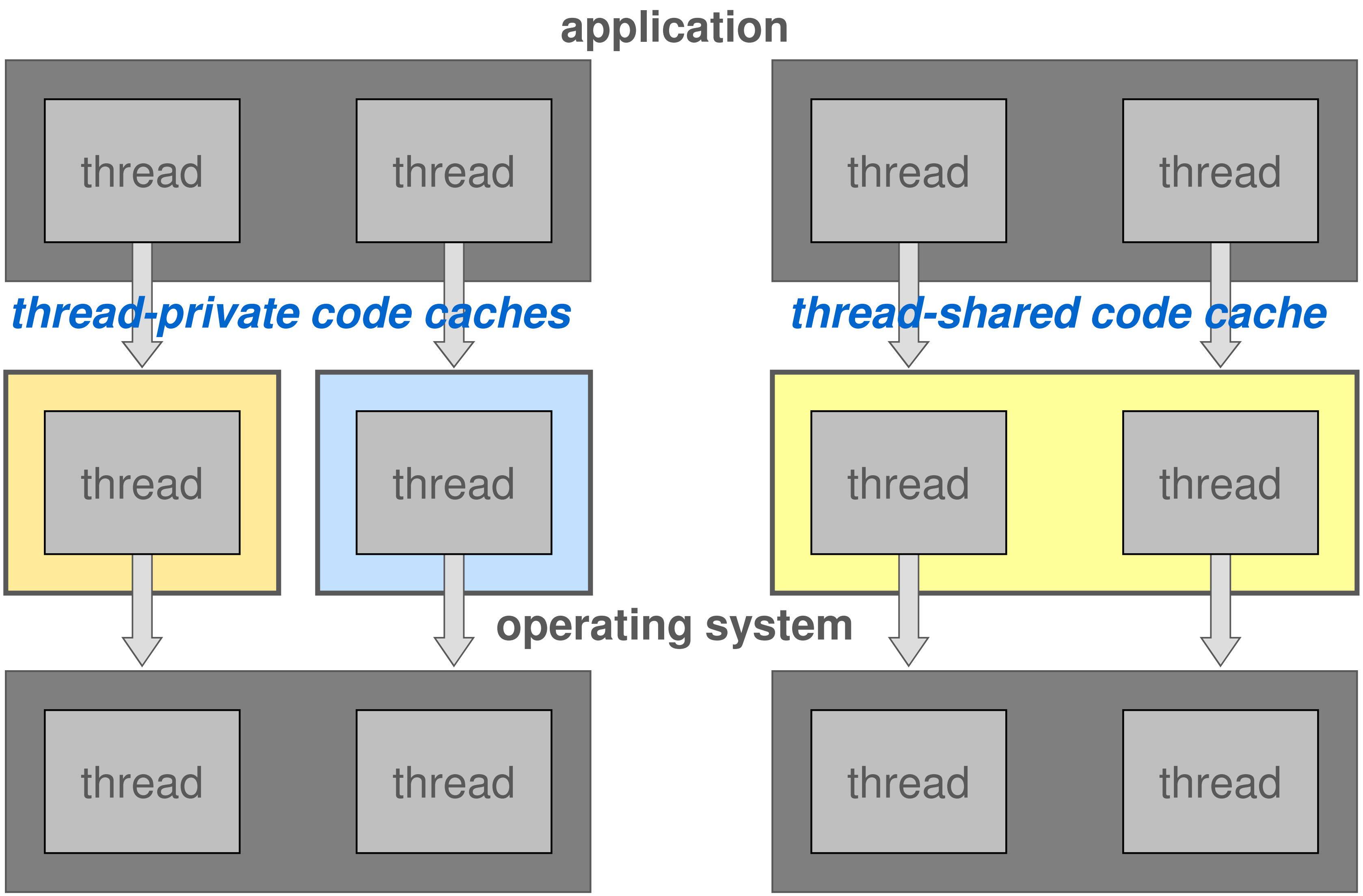
Or, focus on hot code only

- Trace creation event: can modify the trace
- Custom trace creation: can determine trace end condition
- For optimization and profiling tools

Transformation Time vs Execution Time



Code Cache Threading Models



Secondary Client Events

Application thread creation and deletion

Application library load and unload

Application exception/signal

- Client chooses whether to deliver, suppress, bypass the app handler, or redirect control

Application pre- and post- system call

- Client can inspect/modify call number, params, or return value

Bookkeeping: init, exit, cache management, etc.

DynamoRIO API: General Utilities

Safe utilities for maintaining transparency

- Separate stack, memory allocation, file I/O
- Thread-local storage, synchronization
- Create client-only thread or private itimer

Application control

- Suspend and resume all other threads

Application inspection

- Address space querying
- Module iterator
- Processor feature identification

DynamoRIO API: Code Manipulation

Clean calls to C or C++ code

- Automatically inlined for simple callees

Full IA-32/AMD64 instruction representation

- Includes implicit operands, decoding, encoding

State preservation

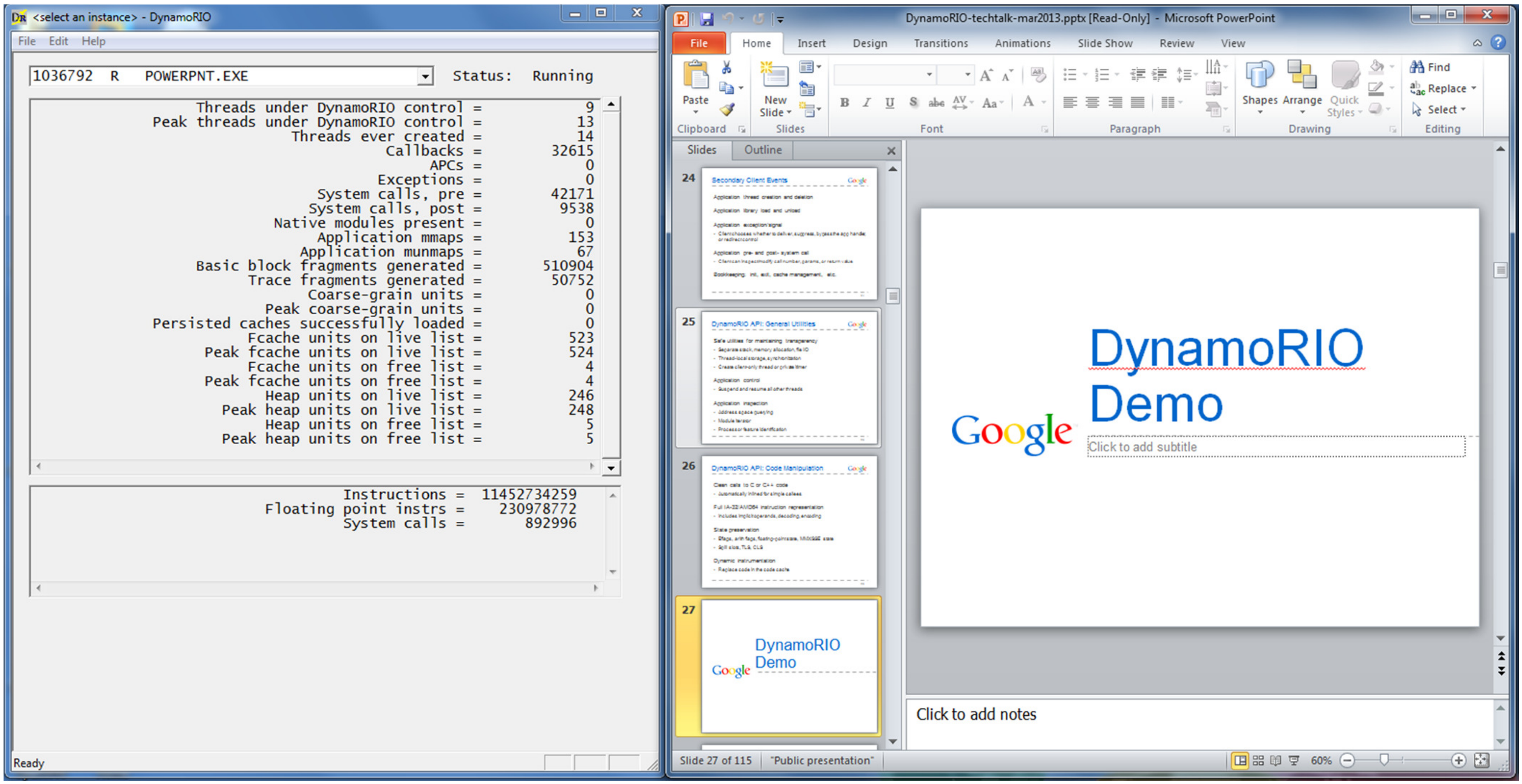
- Eflags, arith flags, floating-point state, MMX/SSE state
- Spill slots, TLS, CLS

Dynamic instrumentation

- Replace code in the code cache

DynamoRIO Demo

Powerpoint Under Inspector



Outline

Google

Base System: DynamoRIO

- Efficient
- Transparent
- Comprehensive
- Customizable

Dynamic Program Inspectors

- Examples and Possibilities
- Case studies
  - Program shepherding
  - Dr. Memory

Examples and Possibilities

Code Inspection

- Code coverage
- Path profiling

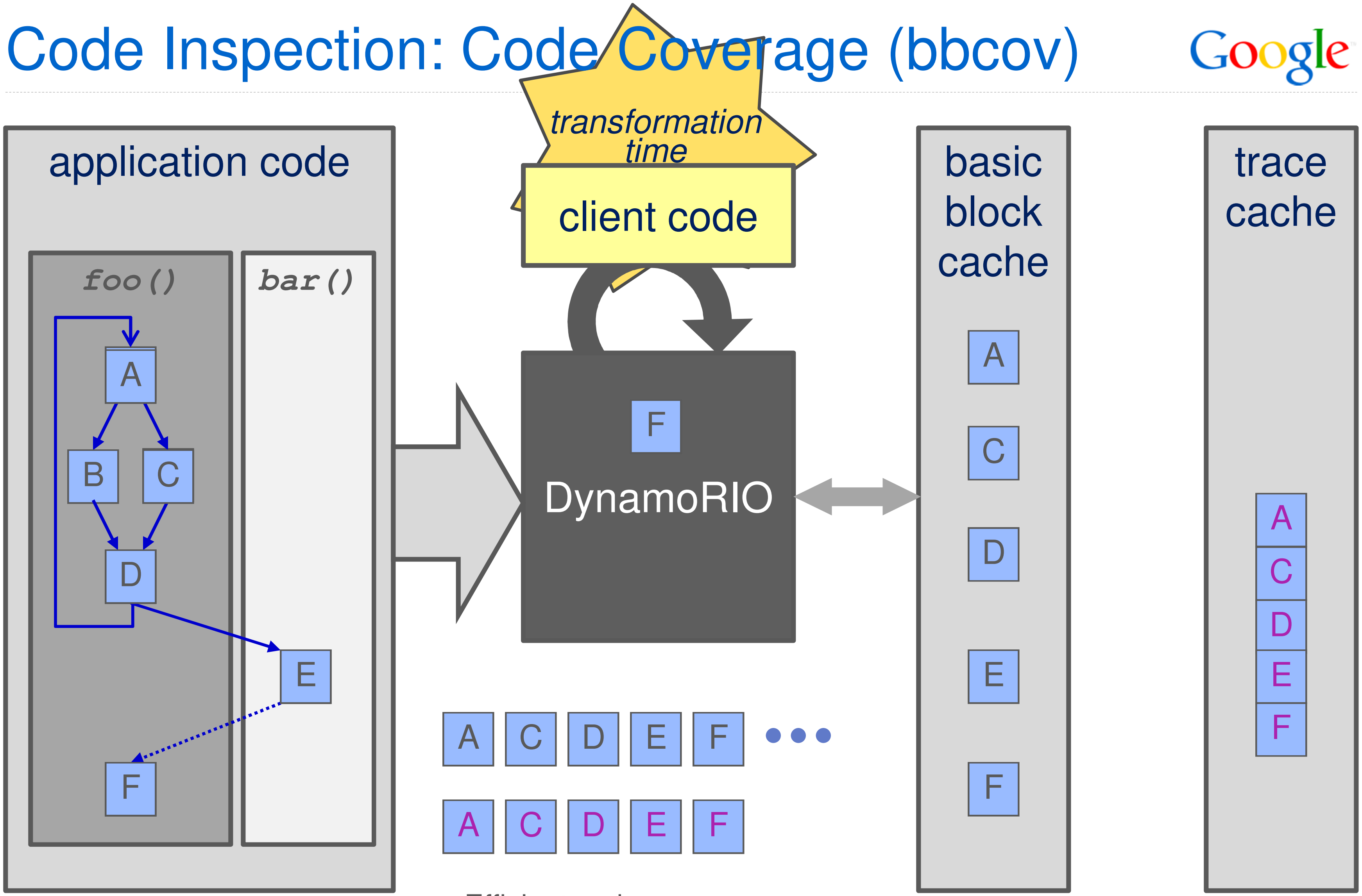
Data Inspection

- Heap overflow detection

Concurrency Inspection

- Cache contention detection

Code Inspection: Code Coverage (bbcov)



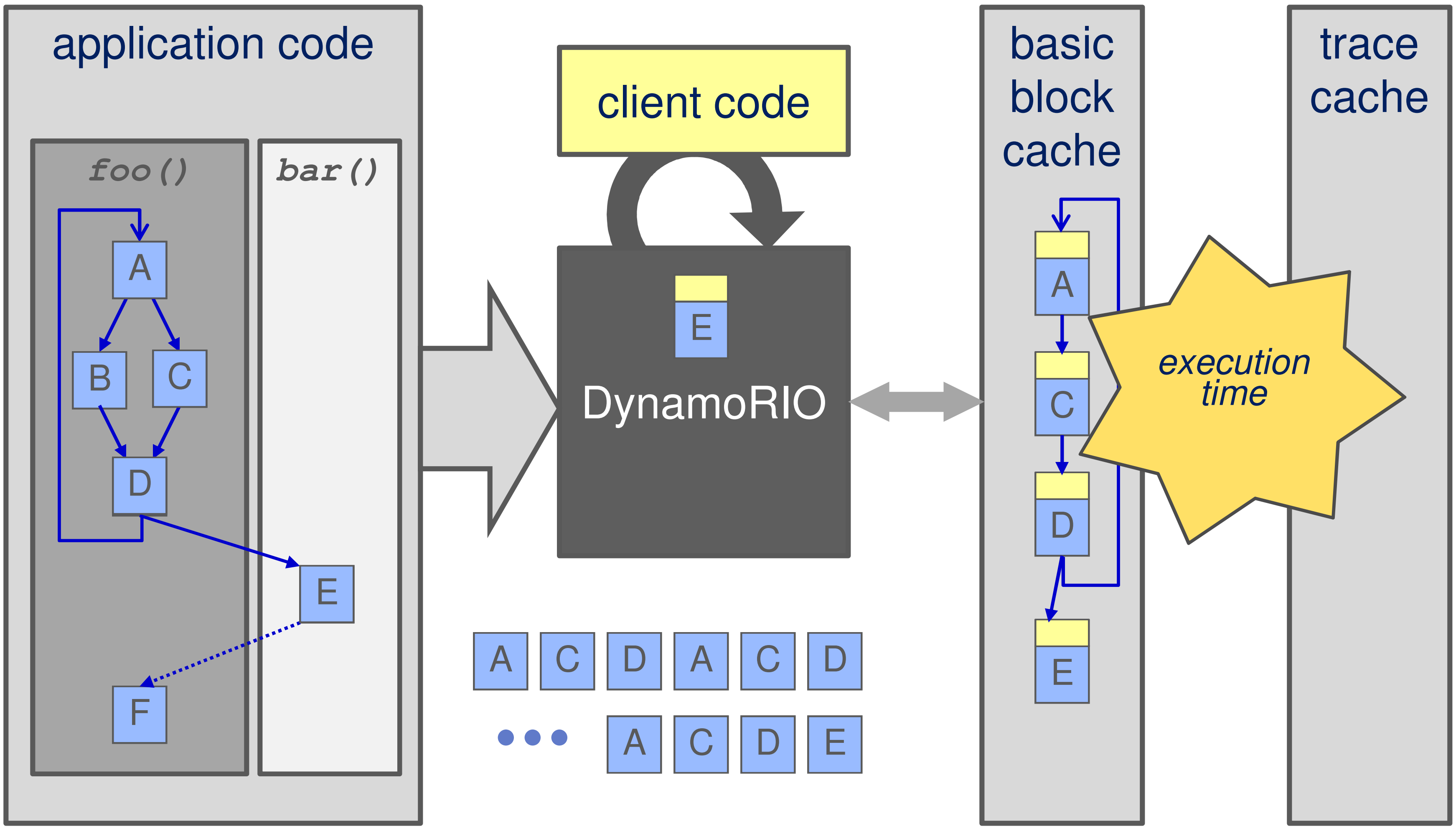
- Efficient code coverage
- Hot/cold code discovery
- Cold start optimization

Code Inspection: Code Coverage (bbcov)

```
void dr_init(client_id_t id)
 { …
    dr_register_bb_event(event_basic_block);
    …
    if (dr_using_all_private_caches())
        bbcov_per_thread = true;
}


dr_emit_flags_t  event_basic_block(void *dc, void *tag, instrlist_t *bb, bool trace, bool xl8)
{
    …
    for (instr = instrlist_first(bb); instr != NULL; instr = instr_get_next(instr)) { … }
    …
    bb_table_entry_add(dc, data, start_pc, cbr_tgt, (end_pc - start_pc), num_instrs, trace);
    return DR_EMIT_DEFAULT;
}
```

Code Inspection: Path Profiling (bbbuf)



Code Inspection: Path Profiling (bbbuf)

Google

```
void dr_init(client_id_t id)
 { …
   dr_register_bb_event(event_basic_block);
   if (!dr_raw_tls_calloc(&tls_seg, &tls_offs, 1, 0))
      DR_ASSERT(false);
}
```

```
start_pc = 0xf771bb9b
 mov    (%esp) → %ebx
 ret     %esp (%esp) → %esp
end_pc = 0xf771bb9f
```

```
dr_emit_flags_t  event_basic_block(void *dc, void *tag, instrlist_t *bb, bool trace, bool xl8)
 { …
   /* load buffer pointer from TLS field */
   MINS(bb, first, INSTR_CREATE_mov_ld
          (dc, opnd_create_reg(reg),
           opnd_create_far_base_disp(tls_seg, DR_REG_NULL, DR_REG_NULL,
                                               0, tls_offs, OPSZ_PTR)));
   /* store bb's start pc into the buffer */
   MINS (bb, first, INSTR_CREATE_mov_st
          (dc, OPND_CREATE_MEM32(reg, 0), OPND_CREATE_INT32(pc)));
   /* advance buffer, we use lea to avoid aflags save/restore */
   MINS(bb, first, INSTR_CREATE_lea
          (dc, opnd_create_reg(reg_16),
           opnd_create_base_disp(reg, DR_REG_NULL, 0,
                                            sizeof(app_pc), OPSZ_lea)));
   /* save buffer pointer */
   MINS(bb, first, INSTR_CREATE_mov_st
          (dc, opnd_create_far_base_disp(tls_seg, DR_REG_NULL, DR_REG_NULL,
                                                0, tls_offs, OPSZ_PTR),
           opnd_create_reg(reg)));
   return DR_EMIT_DEFAULT;
}
```

```
mov    %fs:0x4c → %ebx
mov    $0xf771bb9b → (%ebx)
lea     0x04(%ebx) → %bx
mov    %ebx → %fs:0x4c
```

Code Inspection

Profiling

- Instruction/edge/path/inter-procedural profiling
- Hot/cold code
- Control-flow/call graph

Debugging

- Execution recording
- Software breakpoint

Security

- Program shepherding
- Code de-obfuscation

Examples and Possibilities

Code Inspection

- Code coverage
- Path profiling

Data Inspection

- Heap overflow detection

Concurrency Inspection

- Cache contention detection

Data Inspection: Heap Overflow Detection

Catch heap underflow and overflow:

- Wrap allocation routines
  - Keep track of malloc chunks.
  - Insert *redzones* between application malloc chunks and put special value (pattern) like *0xf1fd* in the redzone.
- Instrumentation
  - Check value before every memory access: look for *0xf1fd*.
  - If found, check whether address is in redzone.

Instrumentation

```
void pattern_insert_cmp_jne_ud2a(void *dc, instrlist_t *ilist, instr_t *app, opnd_t ref, opnd_t pattern)
{
    instr_t *label;
    app_pc pc = instr_get_app_pc(app);
    label = INSTR_CREATE_label(drcontext);
    /* cmp ref, pattern */
    PREXL8M(ilist, app, INSTR_XL8
                (INSTR_CREATE_cmp(dc, ref, pattern), pc));
    /* jne label */
    PRE(ilist, app, INSTR_CREATE_jcc_short
            (dc, OP_jne_short, opnd_create_instr(label)));
    /* illegal instr */
    PREXL8M(ilist, app, INSTR_XL8(INSTR_CREATE_ud2a(dc), pc));
    /* label */
    PRE(ilist, app, label);
}

void dr_init(client_id_t id)
{   …
#ifdef LINUX
    dr_register_signal_event(event_signal);
#else
    dr_register_exception_event(event_exception);
#endif
}
```

```
cmp    0x00000084(%eax)  $0xf1fdf1fd

jnz    <label>

ud2a

<label>
mov    0x1c(%esp) → %eax
mov    0x00000084(%eax) → %edx
test   %edx %edx
jz     $0xf77e6ea2
```

Data Inspection

Google

Profiling

- Memory tracing
  - Cache simulation, data layout/prefetch optimization, etc.
- System call tracing
- Heap state inspection

Debugging

- Memory bug detection
  - Uninit error, buffer overflow/underflow, memory leak, etc.
- Software watchpoint

Security

- Dynamic data-flow tracking (taint-trace)

Examples and Possibilities

Code Inspection

- Code coverage
- Path profiling

Data Inspection

- Heap overflow detection

Concurrency Inspection

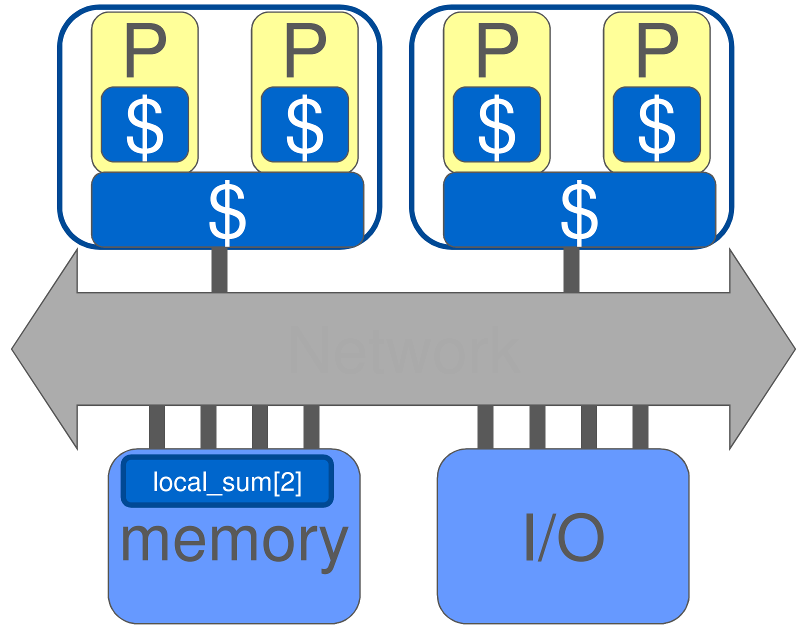
- Cache contention detection

Concurrency Inspection: Cache Contention

Motivating example:

```
uint64 local_sum[2];
uint64 global_sum;

parallel_sum(int myid, int start, int end) {
    for (int i = start; i < end; i++)
        local_sum[myid] += buf[i];
    lock();
    global_sum += local_sum[myid];
    unlock();
}
```



Xeon X5460 @ 3.16GHz, 2x Quad core

# Threads	1	2		
		same core	distinct cores	
			min	max
Time(s): no padding	4.798	4.842	3.883	5.219

Hardware Performance Counter

Hardware limitation

- Limited events: must deduce from supported counter

Hardware specific

- Cache configuration, particular cache line size, cache size, etc.
- Thread-CPU binding

Flexibility

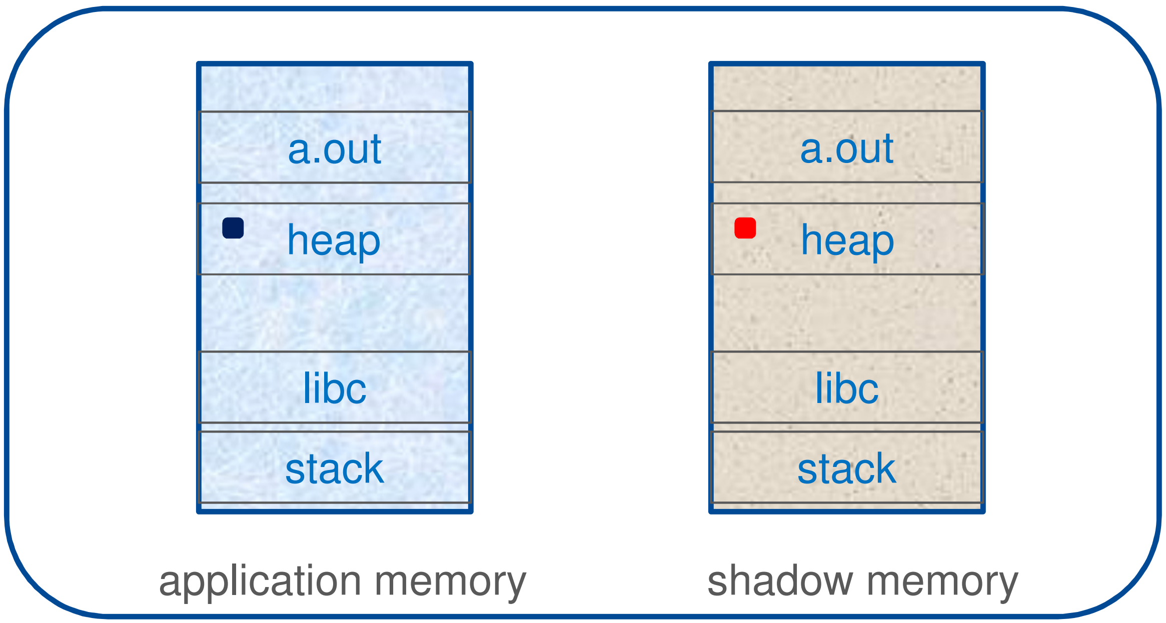
- Limited to sampling
- Hard to reconfigure

Software Shadow Memory

Store meta-data

- Track properties of application memory

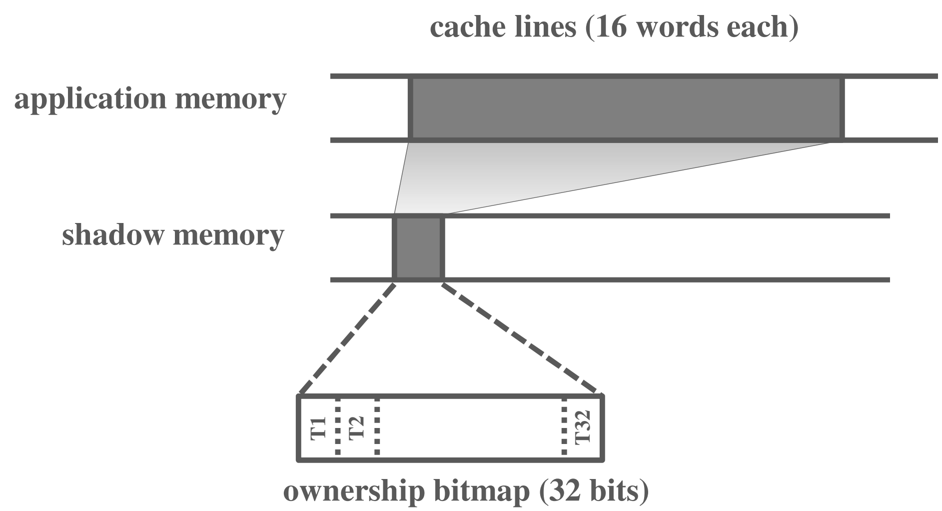
Update via instrumented code



process address space

Cache Contention Detection

Cacheline mapped to thread ownership bitmap



Memory reference:

- Test and set thread bit (cache miss)

Memory write:

- Compare and set only own bit (cache invalidation)

Concurrency Inspection

Profiling

- Cache contention
- False sharing
- Multi-thread communication

Debugging

- Data race detection
- Deterministic record and replay

Security

- Deterministic scheduling

Other Possible Applications

Performance

- Cross-architectural performance estimation

Debugging

- Integration with debugger with reverse execution

Security

- Sandboxing

Others

- Dynamic translation

Outline

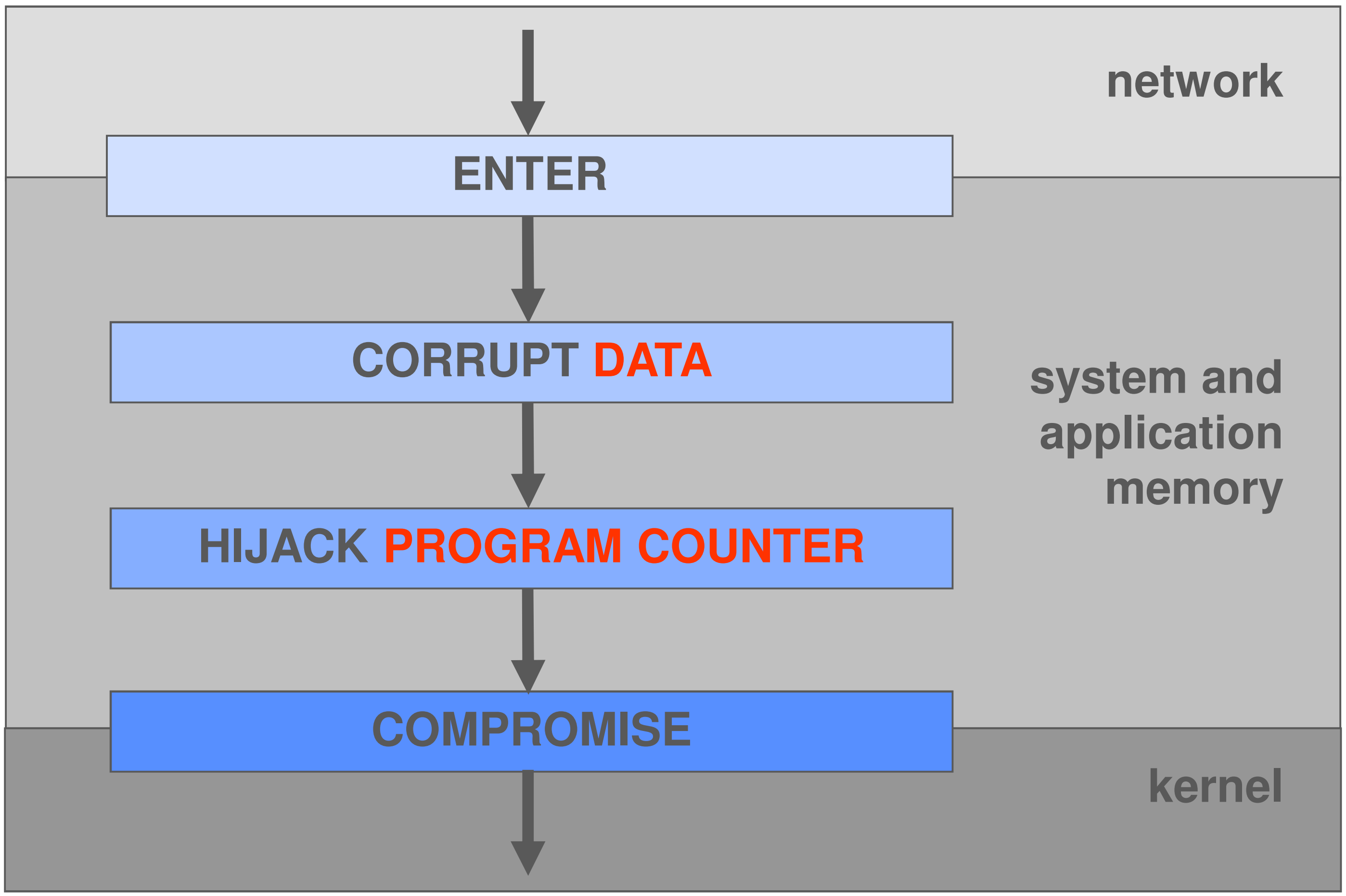
Base System: DynamoRIO

- Efficient
- Transparent
- Comprehensive
- Customizable

Dynamic Program Inspectors

- Examples and Possibilities
- Case studies
  - Program shepherding
  - Dr. Memory

Anatomy of a Memory-Based Attack



Critical Data: Control Flow Indirection

Subroutine calls

- Return address and activation records on visible stack

Dynamic library linking

- Function exports and imports

Object oriented polymorphism: dynamic dispatch

- Vtables

Callbacks – registered function pointers

- Event dispatch, atexit

Exception handling

`Any problem in computer science can be solved with another layer of indirection.`

`– David Wheeler`

Critical Data: Control Flow Exploits

Return address overwrite

- Classic buffer overflow

GOT overwrite

Object pointer overwrite or uninitialized use

Function pointer overwrite

- Heap, stack, data, PEB

Exception handler overwrites

- SEH exploits

Any problem in computer science can be solved with another layer of indirection. But that usually will create another problem.
– David Wheeler

Preventing Data Corruption Is Difficult

Stored program addresses legitimately manipulated by many different entities

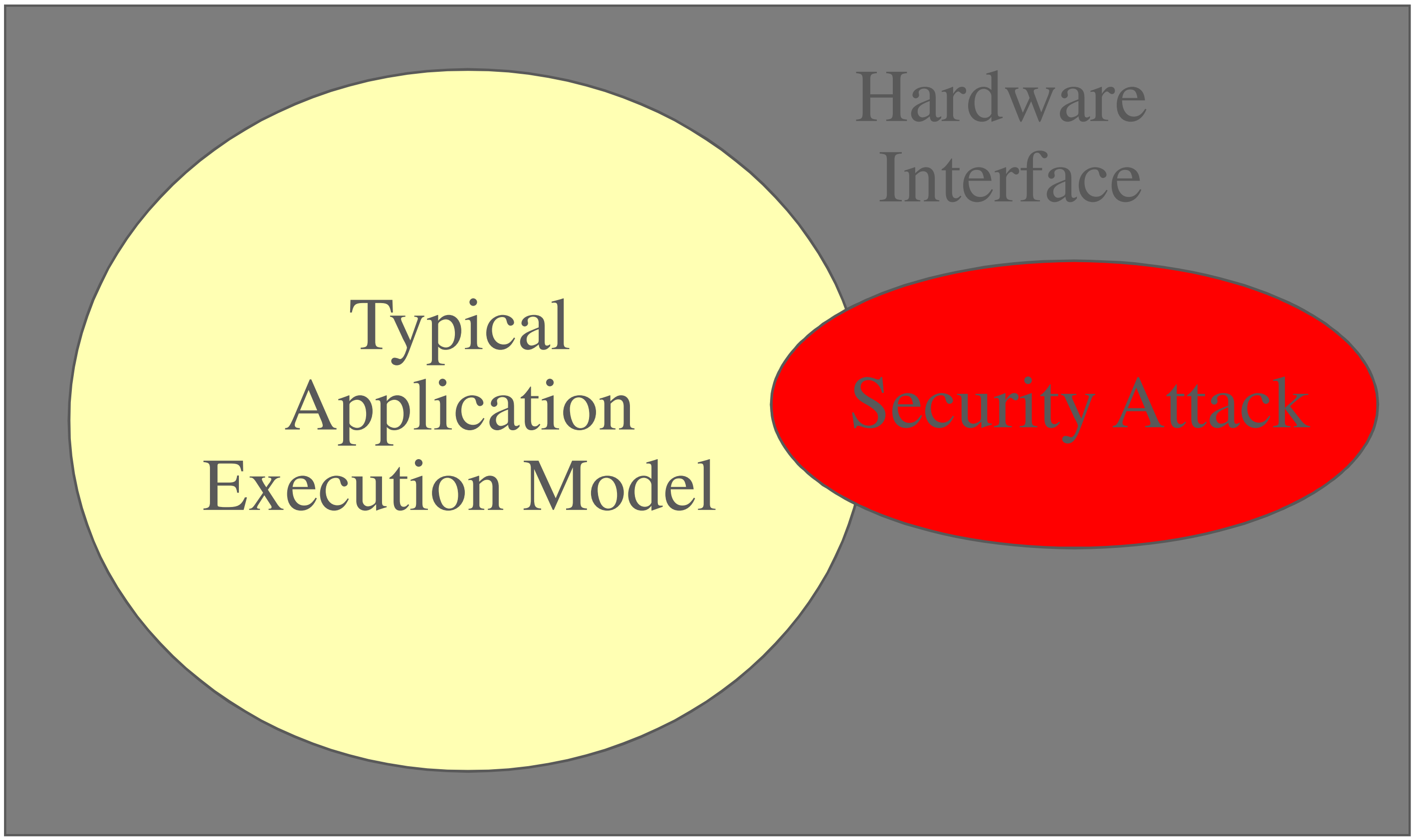
- Dynamic linker, language runtime

Intermingled with regular data

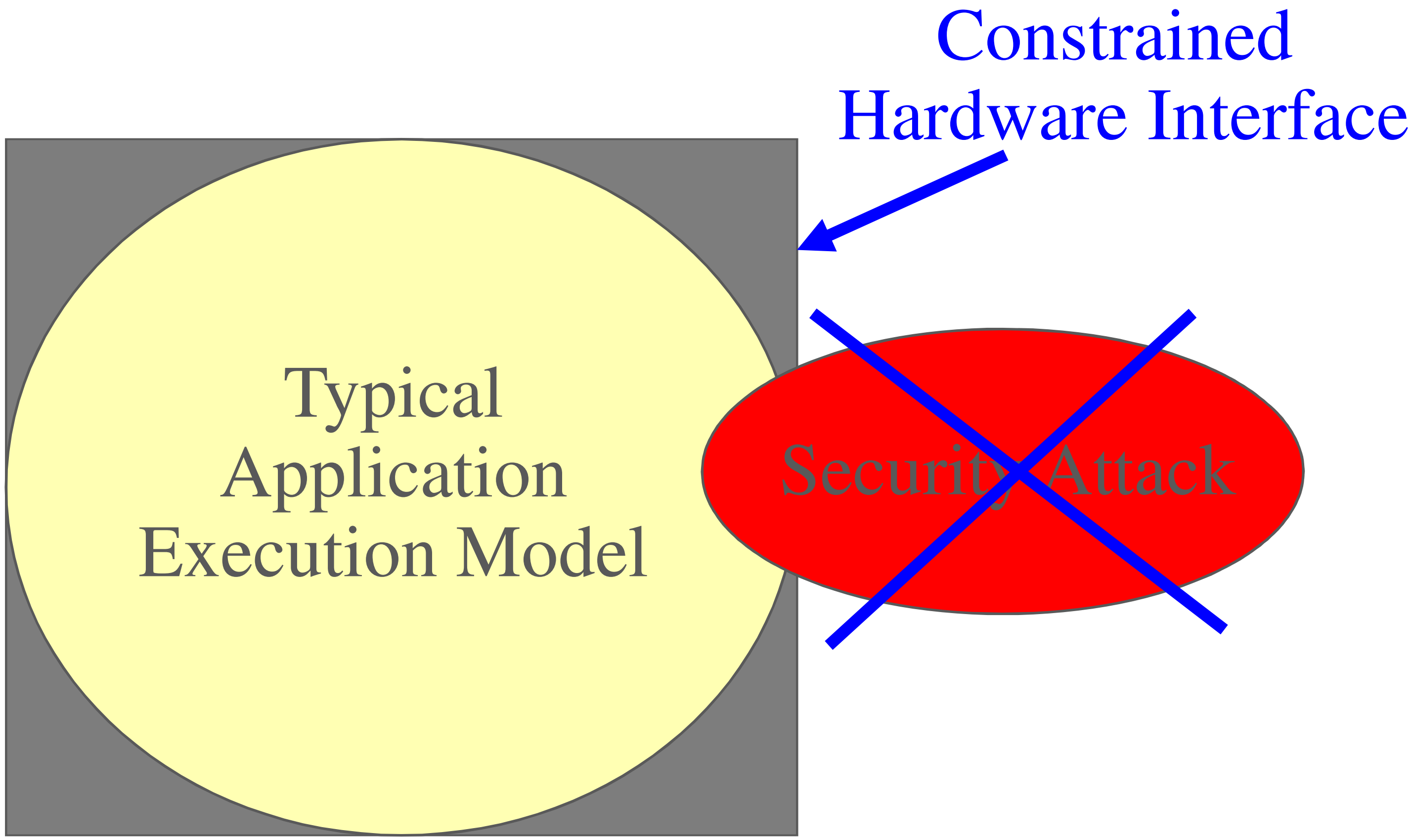
- Return addresses on stack
- Vtables in heap

Even if could distinguish a good write from a bad write, too expensive to monitor all data writes

Insight: Hijack Violates Execution Model



Goal: Shrink Hardware Interface



Program Shepherding

Monitor all control-flow transfers during program execution

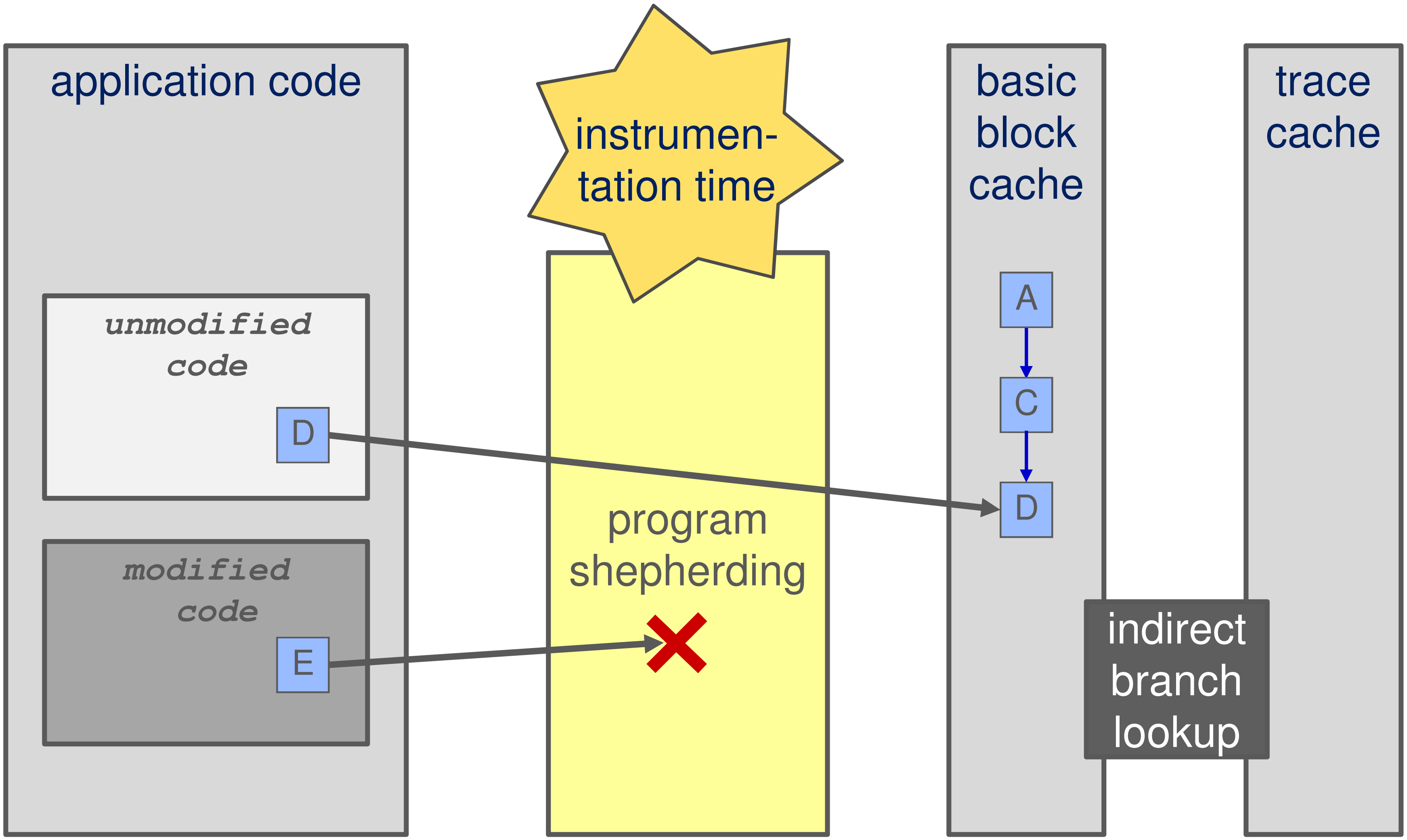
- DynamoRIO is in perfect position to do this

Validate that each transfer satisfies security policy based on execution model

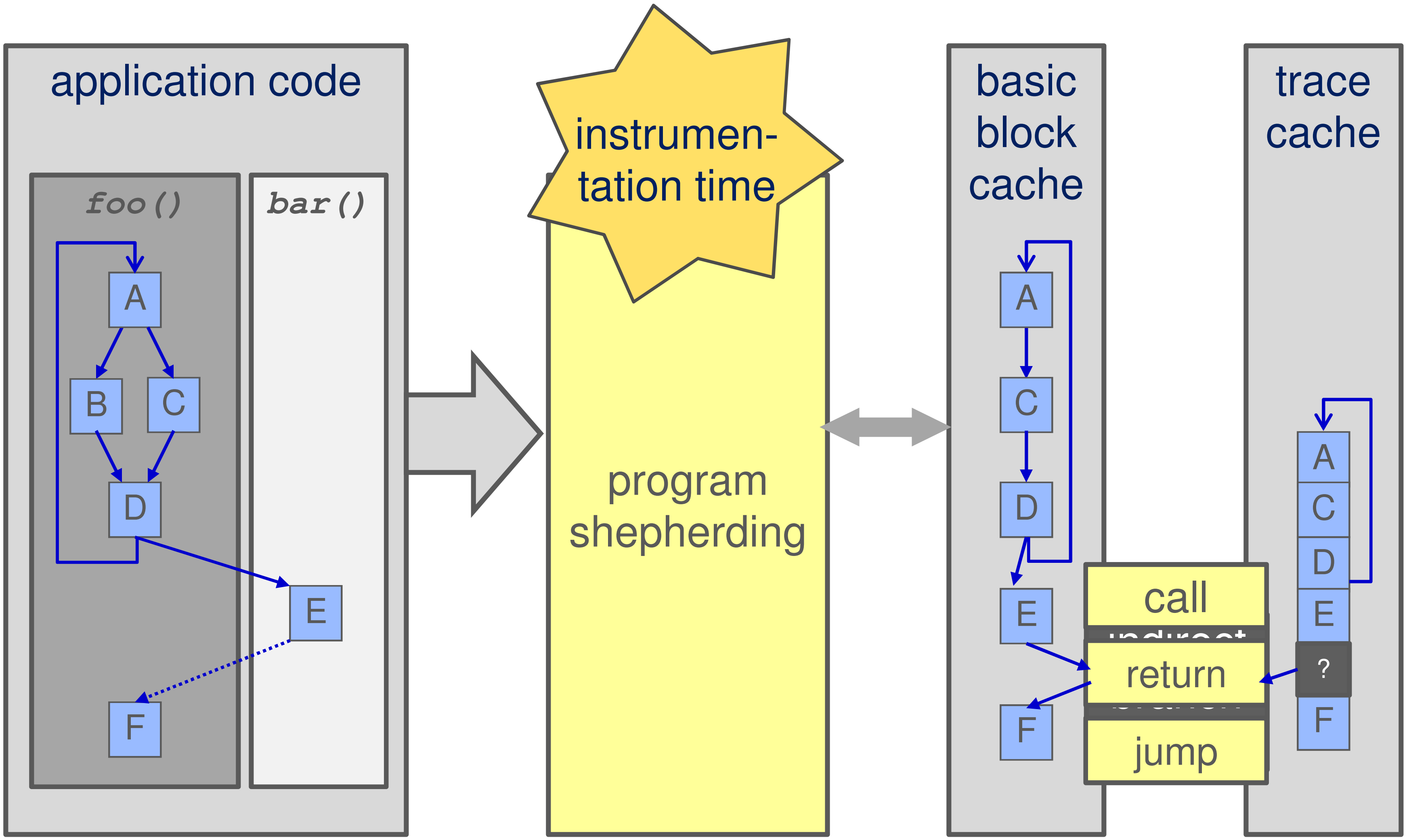
- Application Binary Interface (ABI): calling convention, library invocation

The application may be damaged by data corruption, but the system will not be compromised by hijacking control flow

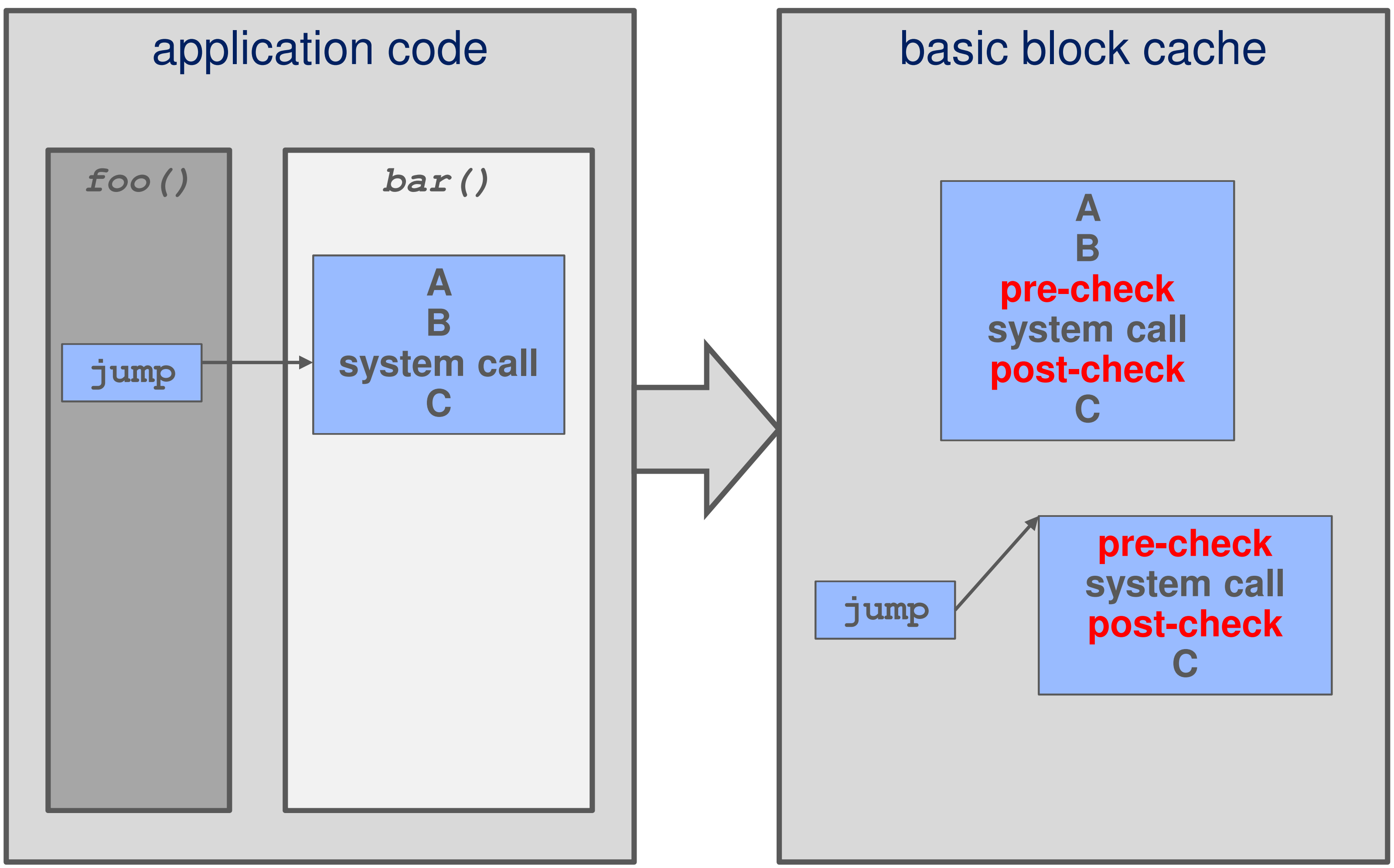
Technique 1: Restricted Code Origins



Technique 2: Restricted Control Transfers



Technique 3: Un-circumventable Sandboxing



Minimal False Positives

Carefully crafted security policies

Automated exemption generation: ‘staging mode’

Determina, Inc: 50 customers, 10,000 machines

- No false positives in MSFT apps
- <50 unique false positives in 3rd party libraries

We treated these false positives as bugs rather than customer driven policies

- Radically different from other security products

Outline

Base System: DynamoRIO

- Efficient
- Transparent
- Comprehensive
- Customizable

Dynamic Program Inspectors

- Examples and Possibilities
- Case studies
  - Program shepherding
  - Dr. Memory

Memory Bugs

Memory bugs are challenging to detect and fix

- Memory corruption, reading uninitialized memory, memory leaks

Observable symptoms resulting from memory bugs are often delayed and non-deterministic

- Errors are difficult to discover during regular testing
- Testing usually relies on randomly happening to hit visible symptoms
- The sources of these bugs are painful and time-consuming to track down from observed crashes

Memory bugs often remain in shipped products and can show up in customer usage

Dr. Memory

Detects *unaddressable memory accesses*

- Wild access to invalid address
- Use-after-free
- Buffer and array overflow and underflow
- Read beyond top of stack
- Invalid free, double free

Detects *uninitialized memory reads*

Detects *memory leaks*

Implementation Strategy

Track the state of application memory using *shadow memory*

- Track whether allocated and whether defined

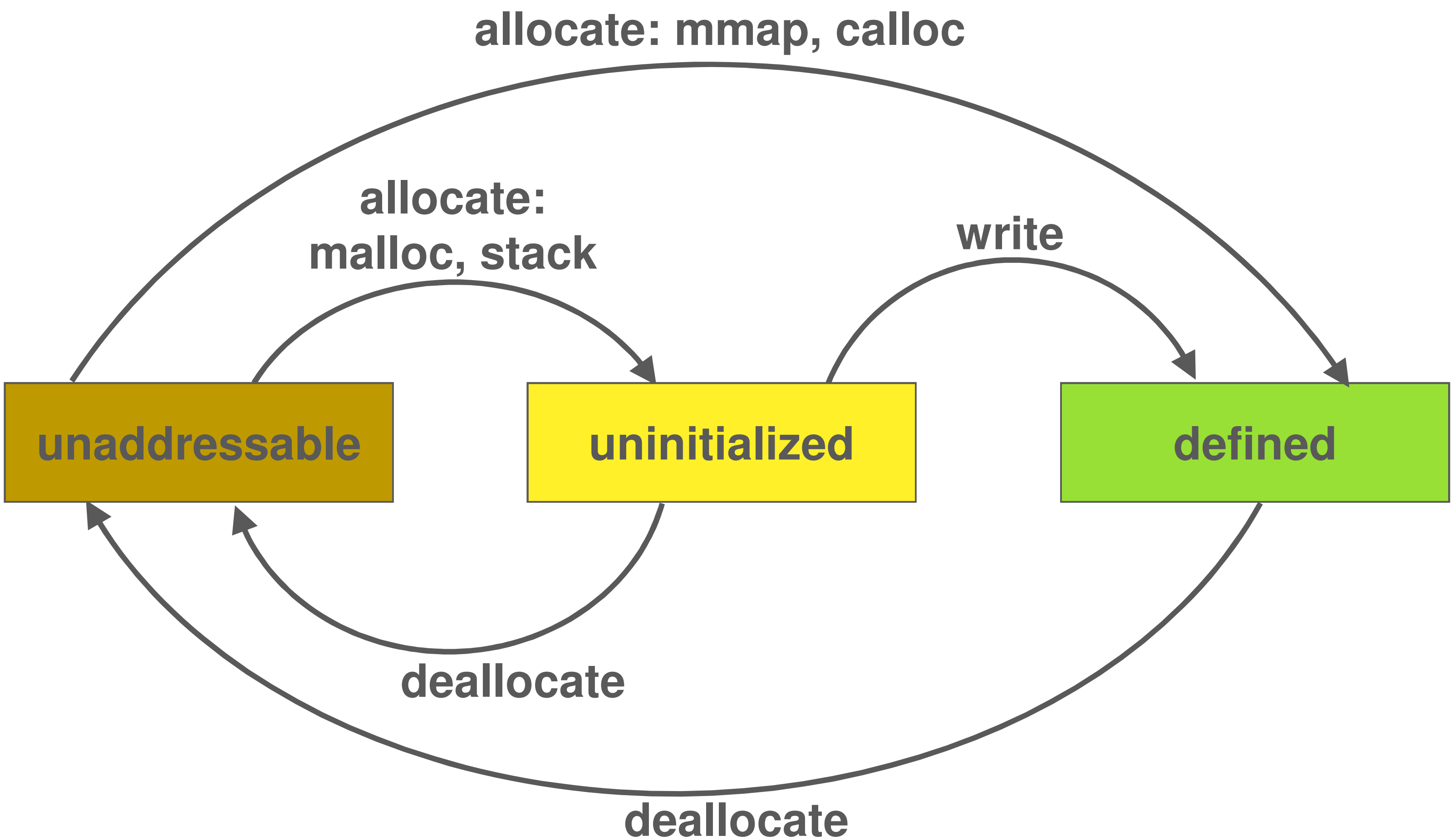
Monitor every memory-related action by the application:

- System call
- Malloc, realloc, calloc, free, mmap, mumap, mremap
- Memory read or write
- Stack adjustment

At exit or on request, scan memory to check for leaks

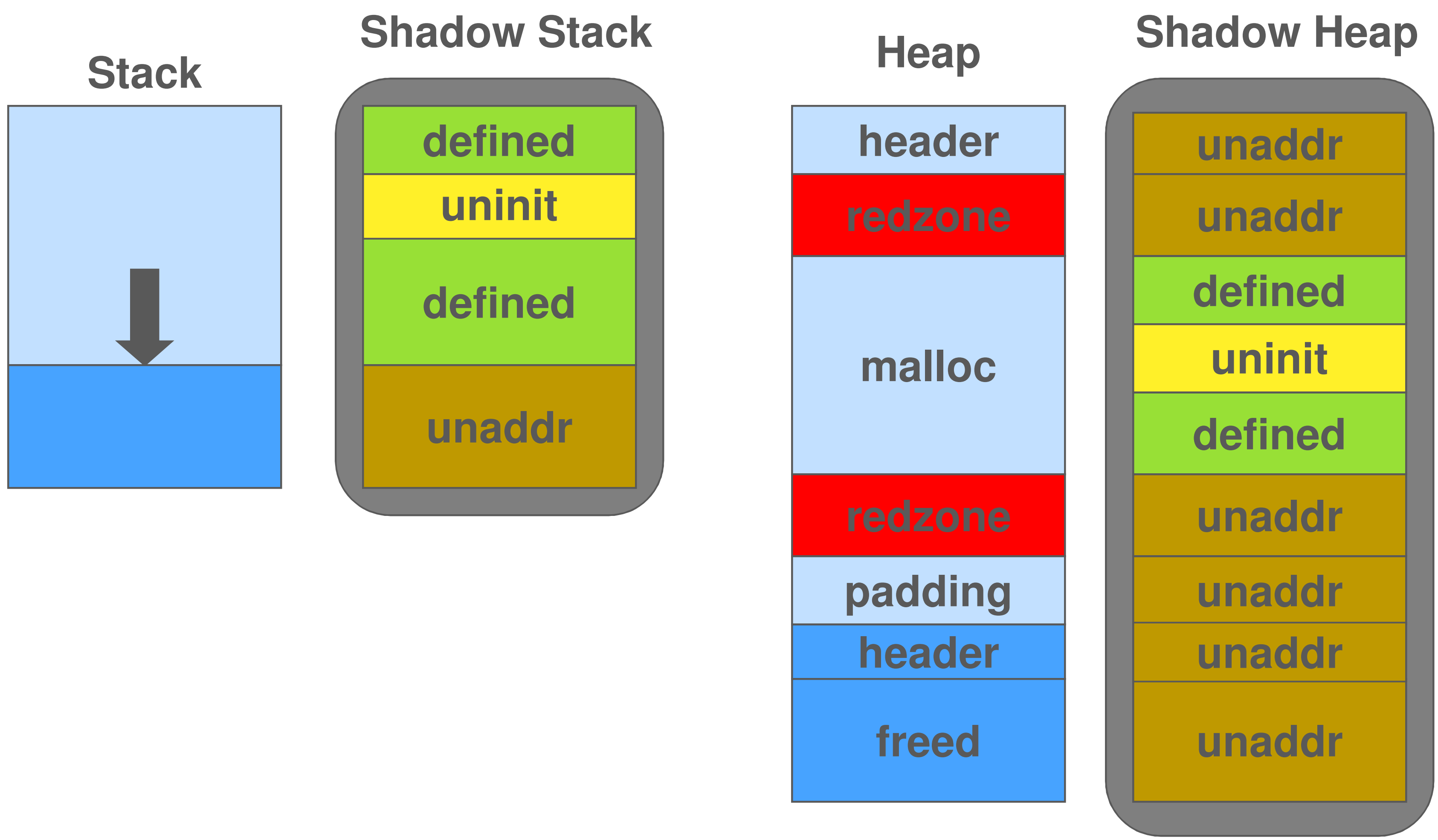
Shadow Metadata

Shadow each byte of memory + registers with 1 of 3 states:



Shadow Memory

Google

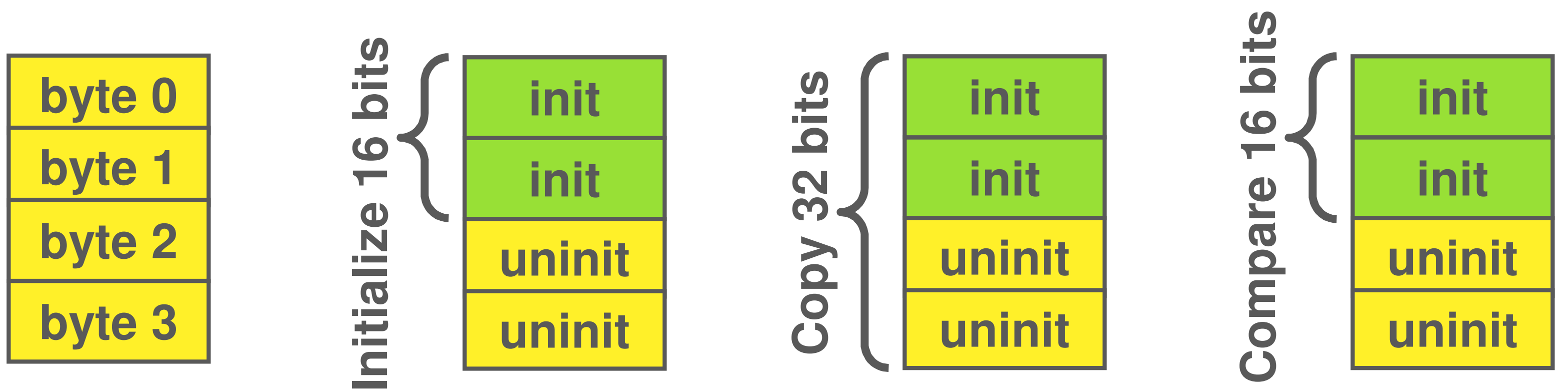


The Uninitialized Whole Word Problem

Google

Sub-word variables are moved around as whole words

- Sub-word field often initialized as sub-word yet copied as whole word
- Reads involved in copying should not raise errors



Solution: report errors on “meaningful” reads only

- Use in compare, conditional branch, address register, or system call

Requires propagating metadata and shadowing registers

- Shadow metadata mirrors application data flow

Memory Leaks

Dr. Memory uses *reachability-based* leak detection

- A *leak* is memory that is no longer reachable by the application
- Memory that is never freed is *not* considered a leak
  - Acceptable to not free memory whose lifetime matches process lifetime

At exit time, or on request, perform leak analysis

- Similar to mark-and-sweep garbage collection

Dr. Memory divides all allocated memory into categories based on how it can be reached by live application pointers

- Any pointer-aligned and *initialized* pointer-sized word is considered a potential pointer

Heap Usage and Staleness

Memory usage statistics

- Snapshots of memory usage spaced uniformly across execution
- Drill down by allocation callstack

"Staleness" information

- Record the time at which each allocation was last accessed
- Helps identify "logical memory leaks", where memory is still reachable but is no longer needed
- Also identifies "hotness" of heap objects

Approach

- Shadow memory state is touched or not touched
- Periodically sample shadow state and update timestamps

Fastpath and Slowpath

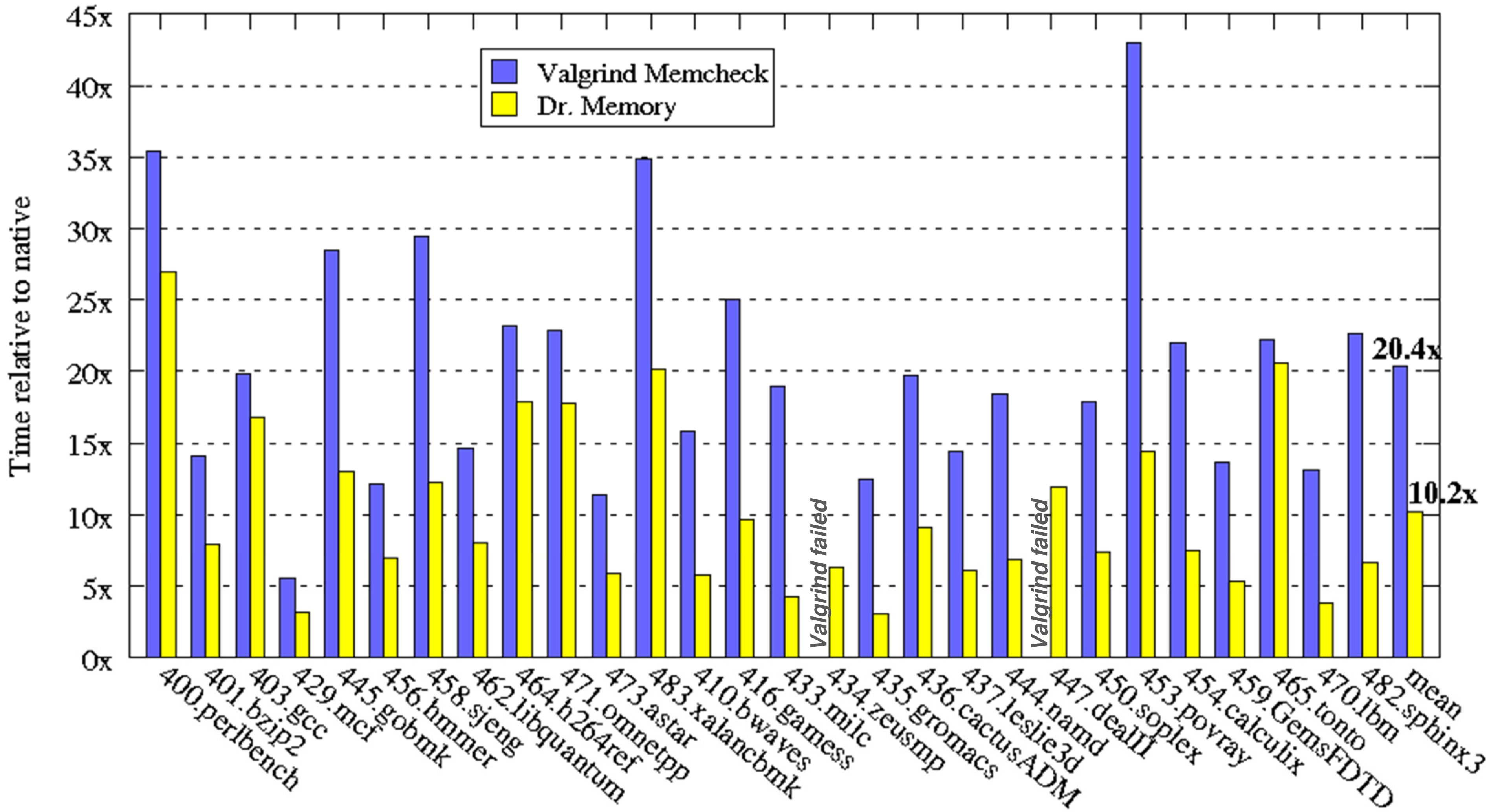
Fastpath = carefully hand-crafted machine-code kernels

- Obtain shadow metadata, combine, and propagate: inlined
- Handle stack pointer updates: lean procedure

Slowpath = clean call to C code

- Unaligned memory references
- Complex instructions
- Allocation library routine and system call handling
- Error reporting

Performance Comparison



Outline

Base System: DynamoRIO

- Efficient
- Transparent
- Comprehensive
- Customizable

Dynamic Program Inspectors

- Examples and Possibilities
- Case studies

Wrap-up

More Information

Web

- http://dynamorio.org
- http://drmemory.org

Email

- http://groups.google.com/group/dynamorio-users
- http://groups.google.com/group/drmemory-users