Introduction to DrCCTProf

Xu Liu

DrCCTProf Overview

- DrCCTProf functionality
  - Binary analysis
    - Analyze compiler code generation
    - Compiler behavior, inefficiencies
  - Dynamic analysis
    - Analyze a program during the program execution with high accuracy

- An open-source project
  - Hands-on experiences with a real project
    - Code with high quality will be committed to the project
  - Better understand the compiler in real world

Compilers Do NOT Eliminate All Inefficiencies

- Binary analysis is necessary to check compiler code generation
- Compilers have limitations with their static analysis
  - ✦ Aliasing and pointers
  - ✦ Limited optimization scopes: compilation units
  - ✦ Input-sensitive inefficiencies
  - ✦ Flow-sensitive inefficiencies

DrCCTProf: Fine-grained Profiling

- Track each instruction
  - ✦ Operator
  - ✦ Operands
- Track each register
  - ✦ General registers
  - ✦ SIMD registers
- Track each memory location
  - ✦ Effective addresses
- Track each value in storage location
  - ✦ Value in registers
  - ✦ Value in memory

DynamoRIO for Binary Instrumentation

- Robust
  - ✦ Google maintains it
  - ✦ Production-level quality

- Dynamic binary instrumentation
  - ✦ Do not need source code, recompilation, and re-linking.

- Programmable instrumentation
  - ✦ Call-based APIs support to build different instrumentation tools (DynamoRIO clients)
  - ✦ Collecting all aforementioned fine-grained information

DynamoRIO for Binary Instrumentation

- Robust
  - ✦ Google maintains it
  - ✦ Production-level quality

- Dynamic binary instrumentation
  - ✦ Do not need source code, recompilation, and re-linking.

- Programmable instrumentation
  - ✦ Call-based APIs support to build different instrumentation tools (DynamoRIO clients)
  - ✦ Collecting all aforementioned fine-grained information

$ drrun -t client -- application

Client Example: Dynamic Basic Block Count

```
#include “dr_api.h”
DR_EXPORT void
dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_register_bb_event(event_bb);
}
```

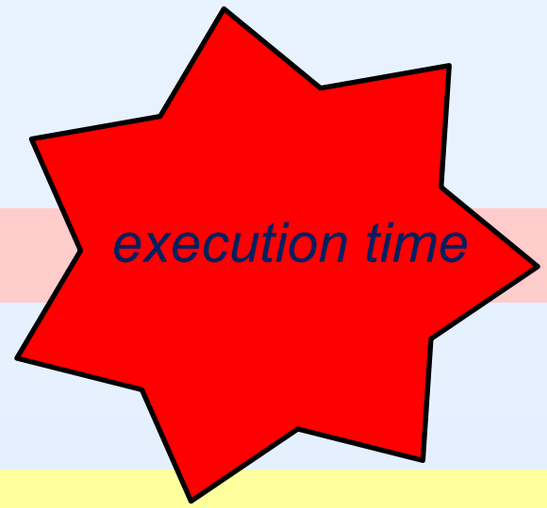
```
static void bbcount() { global_count++; }
```

```
static dr_emit_flags_t
event_bb(void *drcontext, void *tag, instrlist_t *bb,
                bool for_trace, bool translating) {
    dr_insert_clean_call(drcontext, bb, instrlist_first_app(bb),
                                (void *)bbcount, false /* save fpstate */, 0);
    return DR_EMIT_DEFAULT;
}
```

Client Example: Dynamic Basic Block Count

```
#include “dr_api.h”
DR_EXPORT void
dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_register_bb_event(event_bb);
}
```

```
static void bbcount() { global_count++; }
```




```
static dr_emit_flags_t
event_bb(void *drcontext, void *tag, instrlist_t *bb,
                bool for_trace, bool translating) {
    dr_insert_clean_call(drcontext, bb, instrlist_first_app(bb),
                                (void *)bbcount, false /* save fpstate */, 0);
    return DR_EMIT_DEFAULT;
}
```

Client Example: Dynamic Basic Block Count

```
#include “dr_api.h”
DR_EXPORT void
dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_register_bb_event(event_bb);
}
```

```
static void bbcount() { global_count++; }
```

execution time

```
static dr_emit_flags_t
event_bb(void *drcontext, void *tag, instrlist_t *bb,
                bool for_trace, bool translating) {
    dr_insert_clean_call(drcontext, bb, instrlist_first_app(bb),
                                (void *)bbcount, false /* save fpstate */, 0);
    return DR_EMIT_DEFAULT;
}
```

transformation time

Client Example: Dynamic Basic Block Count

```
#include “dr_api.h”
DR_EXPORT void
dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_register_bb_event(event_bb);
}
```

```
static void bbcount() { global_count++; }
```

execution time

```
call  bbcount()
sub $0xff, %edx
cmp %esi, %edx
jle   <L1>
call  bbcount()
add  %eax, %ebx
mov  $0x1,  %edi
jmp   <L2>
```

```
static dr_emit_flags_t
event_bb(void *drcontext, void *tag, instrlist_t *bb,
                bool for_trace, bool translating) {
    dr_insert_clean_call(drcontext, bb, instrlist_first_
                         (void *)bbcount, false /* save
    return DR_EMIT_DEFAULT;
}
```

sformation time

DynamoRIO: Instrumentation Granularity

- Instructions
  - Arithmetic instructions, memory accesses, function calls and returns
- Basic blocks
  - A sequence of codes ended at a control-flow changing instruction
  - Single entry, single-exit
- Traces
  - A sequence of codes ended at an **unconditional** control-flow changing instruction
  - Single entry, multiple exits

DynamoRIO: Instrumentation Granularity

- Instructions
  - ✦ Arithmetic instructions, memory accesses, function calls and returns
- Basic blocks
  - ✦ A sequence of codes ended at a control-instruction
  - ✦ Single entry, single-exit
- Traces
  - ✦ A sequence of codes ended at an **uncon**changing instruction
  - ✦ Single entry, multiple exits

```
sub     $0xff, %edx
cmp     %esi, %edx
jle     <L1>
```

```
mov     $0x1, %edi
add     $0x10, %eax
jmp     <L2>
```

1 trace
2 basic blocks
6 instructions

Instrumentation: instruction counting

```
DR_EXPORT void
dr_client_main(client_id_t id, int argc, const char *argv[]) {
    …
    drmgr_register_bb_instrumentation_event
        (event_bb_analysis, event_app_instruction, NULL);
}
```

```
static uint global_count;
static void inscount(uint num_instrs) { global_count += num_instrs; }
```

Instrumentation: instruction counting

```
static dr_emit_flags_t
event_bb_analysis(void *drcontext, void *tag, instrlist_t *bb,
                  bool for_trace, bool translating, void **user_data)
{
    instr_t *instr;
    uint num_instrs;

    for (instr = instrlist_first_app(bb), num_instrs = 0;
         instr != NULL;
         instr = instr_get_next_app(instr)) {
        num_instrs++;
    }

    *user_data = (void *)(ptr_uint_t) num_instrs;

    return DR_EMIT_DEFAULT;
}
```

Part 3: Instrumentation: instruction counting

```
static dr_emit_flags_t
event_app_instruction(void *drcontext, void *tag, instrlist_t *bb, instr_t *instr,
                      bool for_trace, bool translating, void *user_data)
{
    uint num_instrs;

    if (!drmgr_is_first_instr(drcontext, instr))
        return DR_EMIT_DEFAULT;
    num_instrs = (uint)(ptr_uint_t) user_data;
    dr_insert_clean_call(drcontext, bb, instrlist_first_app(bb),
                         (void *)inscount, false /* save fpstate */, 1,
                         OPND_CREATE_INT32(num_instrs));

    return DR_EMIT_DEFAULT;
}
```

Cross-Platform IR Support

- Instruction creation: XINST_CREATE_* macros
  - XINST_CREATE_load(), XINST_CREATE_jump(), etc.
- Generic instruction and operand queries
  - E.g.: instr_writes_memory(), instr_reads_from_reg(), instr_compute_address(), instr_is_return()
  - E.g.: opnd_is_memory_reference(), opnd_uses_reg()
- Generic instrumentation creation helpers
  - E.g.: instrlist_insert_mov_immed_ptrsz(), instrlist_insert_mov_instr_addr()
- ISA-specific concepts applied to all ISA's
  - E.g., predication
- *drreg* Extension for using registers without naming them

Cross-Platform IR Support

- Instruction creation: XINST_CREATE_* macros
  - XINST_CREATE_load(), XINST_CREATE_jump(), etc.
- Generic instruction and operand queries
  - E.g.: instr_writes_memory(), instr_reads_from_reg(), instr_compute_address(), instr_is_return()
  - E.g.: opnd_is_memory_reference(), opnd_uses_reg()
- Generic instrumentation creation helpers
  - E.g.: instrlist_insert_mov_immed_ptrsz(), instrlist_insert_mov_instr_addr()
- ISA-specific concepts applied to all ISA's
  - E.g., predication
- *drreg* Extension for using registers without naming them

DynamoRIO works for ARM32/64, x86

Config, Build, and Run

- CMake
  - http://www.cmake.org/
  - Generates build files for native compiler of choice
    - Makefiles for UNIX, nmake, etc.
    - Visual Studio project files
- CMakeLists.txt

```
add_library(myclient SHARED myclient.c)
find_package(DynamoRIO)
if (NOT DynamoRIO_FOUND)
  message(FATAL_ERROR "DynamoRIO package required to
      build")
endif(NOT DynamoRIO_FOUND)
configure_DynamoRIO_client(myclient)
use_DynamoRIO_extension(myclient drmgr)
```

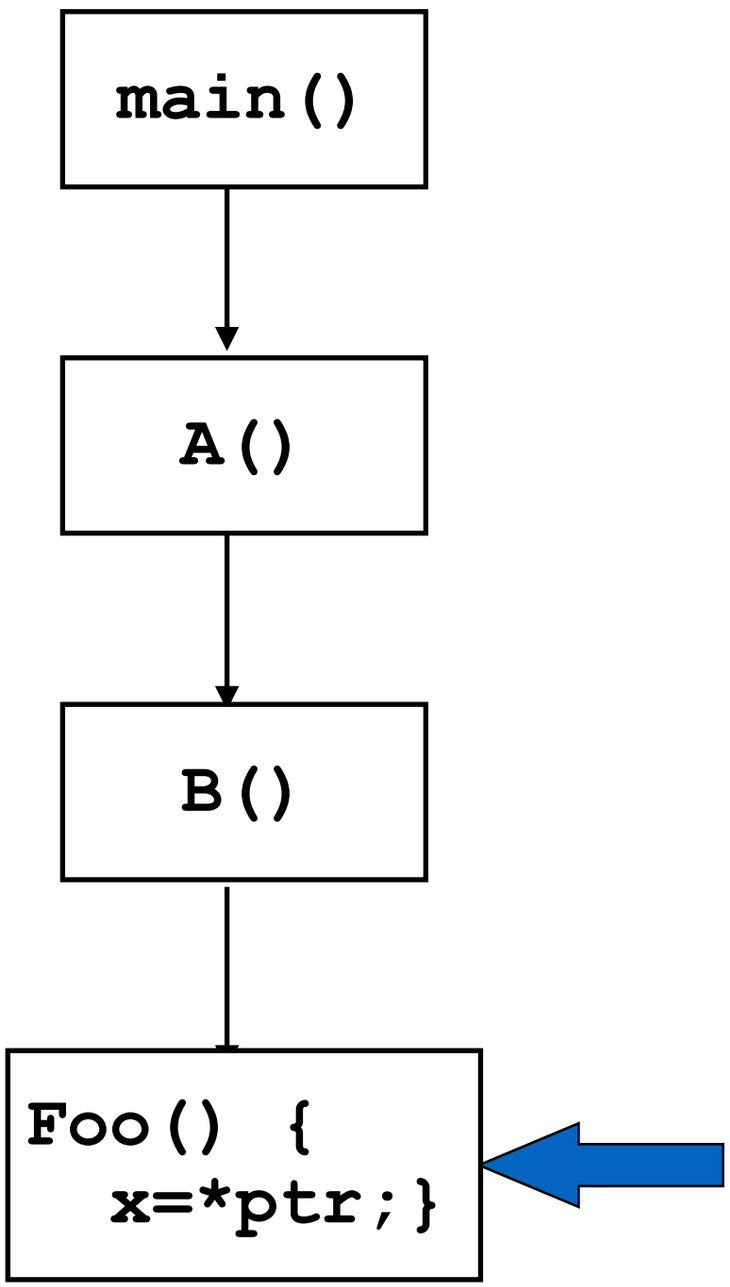
Config, Build, and Run

- Config
  - cmake /path/to/your/client/
  - ccmake, cmake-gui
- Build
  - make
  - cmake --build .
- Run
  - Method 1 (one step)
    - drrun -c <client> <client options> -- <app cmdline>
  - Method 2 (two steps, for better child process blacklisting, etc.)
    - drconfig -reg <appname> -c <client> <client options>
    - drinject <app cmdline>

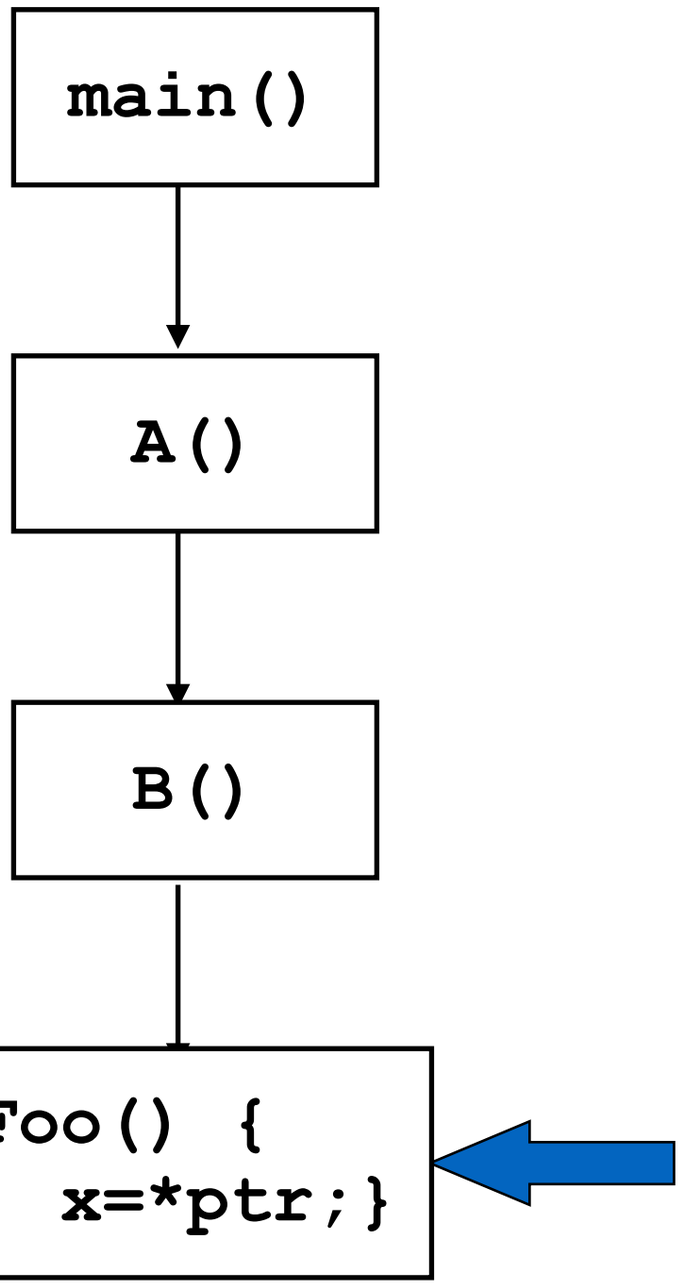
Runtime Options

- Pass options to drconfig/drrun
- A large number of options; the most relevant are:
  - -t <tool>
  - -c <client lib> <client options>
  - -thread_private
  - -follow_children
  - -opt_cleancall
  - -tracedump_text and –tracedump_binary
  - -prof_pcs
  - -code_api
  - -help

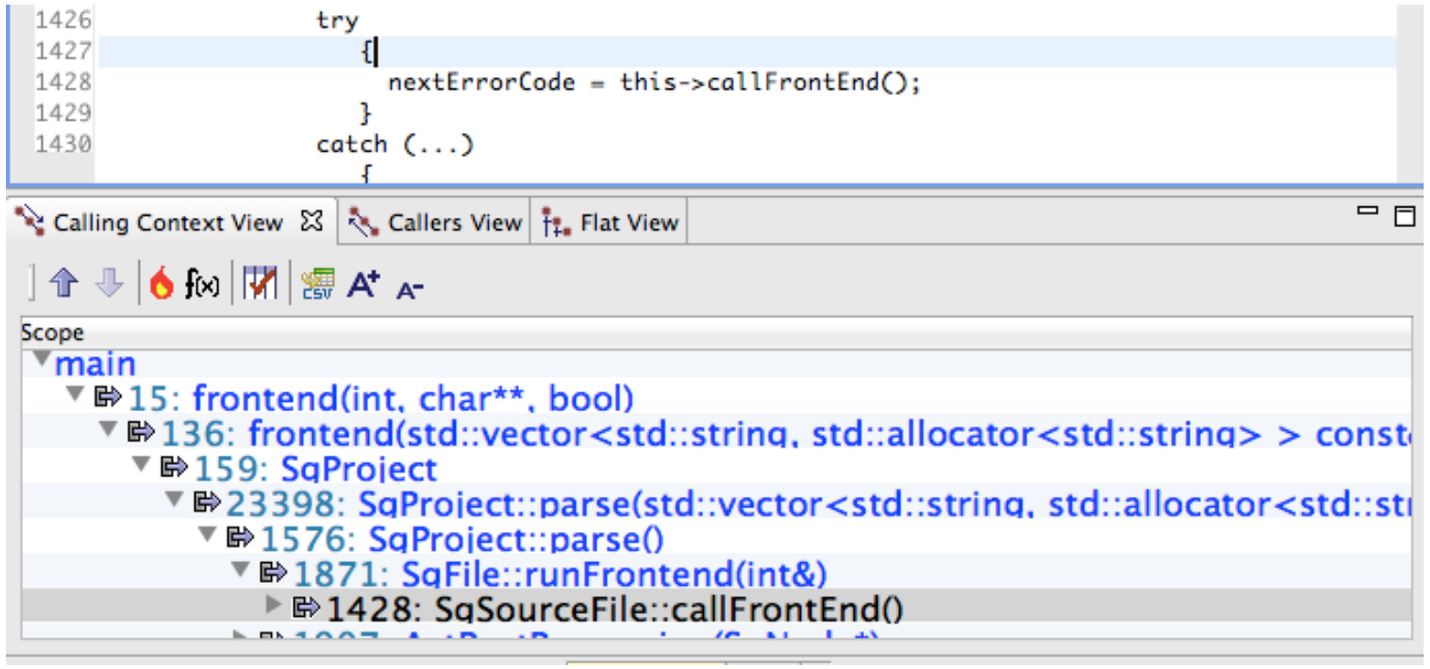
Collecting Call Path



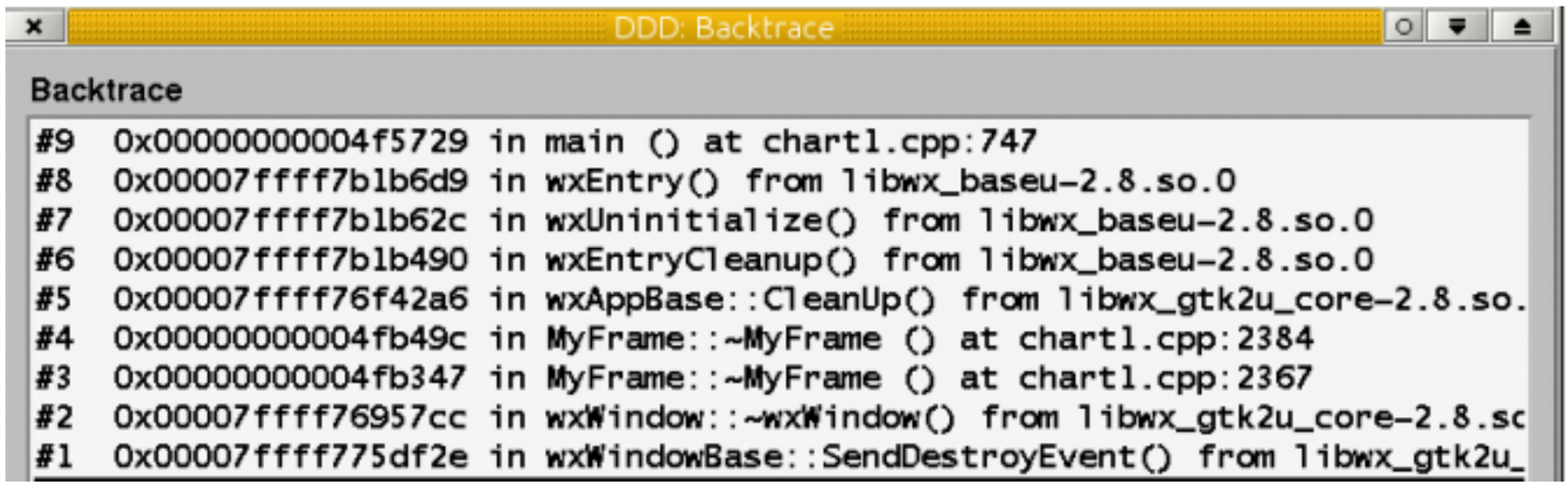
Collecting Call Path



Performance Analysis Tools



Debugging Tools

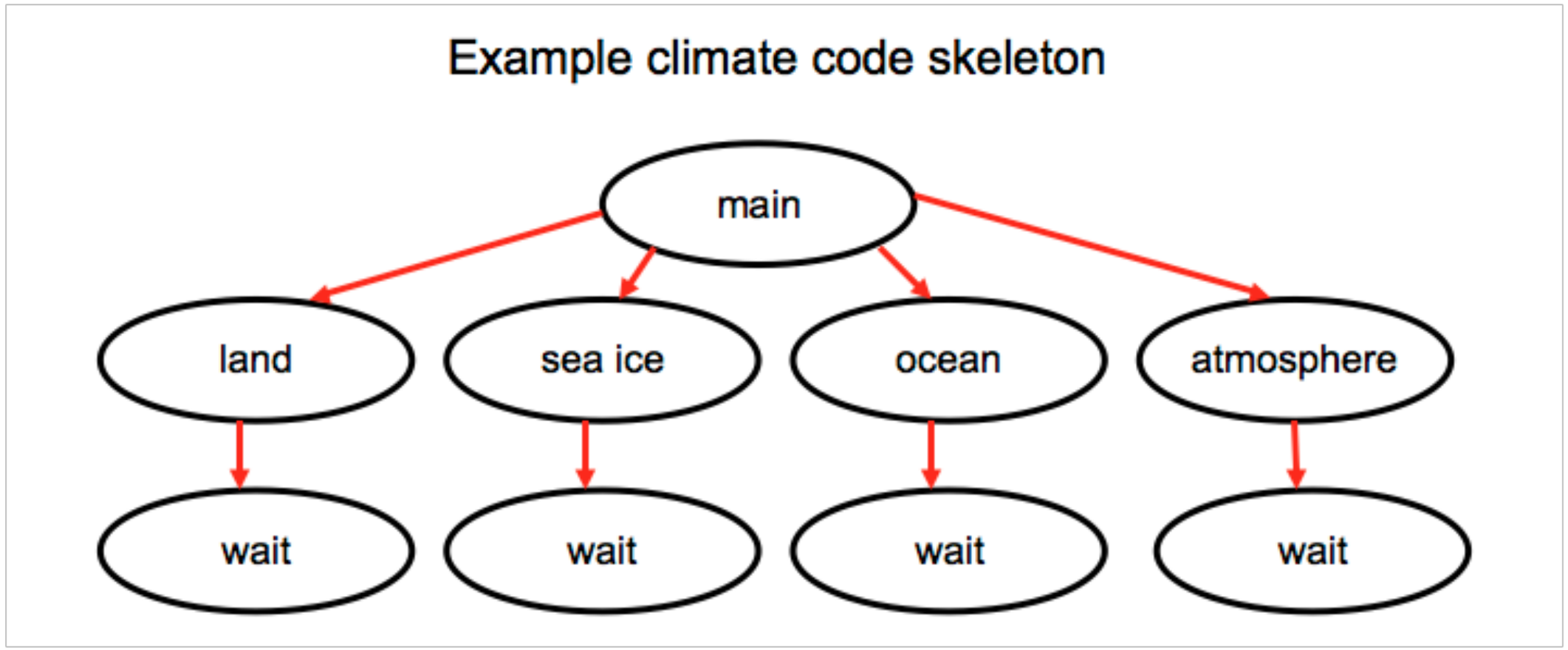


Call Path Profiling for Fine-grained Analysis

- Associate problematic instructions with their call paths
  - ✦ Expose more semantic information about the instructions
  - ✦ Understand path-sensitive performance issues

- If no call path collected for fine-grained analysis
  - ✦ Do not provide root causes of the problem

Importance of Call Paths



Call Path Profiling for Fine-grained Analysis

- Associate problematic instructions with their call paths
  - ✦ Expose more semantic information about the instructions
  - ✦ Understand path-sensitive performance issues

- If no call path collected for fine-grained analysis
  - ✦ Do not provide root causes of the problem
  - ✦ Do not guide source code optimization

An Example: SPEC bwaves

A pair of redundant computation

---

movsdq 0x8(%rdi,%r10,8), %xmm0:__mul:<no src>

**********REDUNDANT WITH **********

movsdq 0x8(%rdi,%r10,8), %xmm0:__mul:<no src>

An Example: SPEC bwaves

A pair of redundant computation

```
------------------------------------------------------------
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
  __dvd: <no src>
   __mpexp:<no src>
    __mplog:<no src>
     __slowpow:<no src>
      __ieee754_pow_sse2:<no src>
       pow:<no src>
        jacobian_:jacobian_lam.f:47
         shell_:shell_lam.f:193
          MAIN__:flow_lam.f:63
           main:flow_lam.f:67
**************REDUNDANT WITH **************
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
 __dvd:<no src>
  __mpexp:<no src>
   __mplog:<no src>
    __slowpow:<no src>
     __ieee754_pow_sse2:<no src>
      pow:<no src>
       jacobian_:jacobian_lam.f:47
        shell_:shell_lam.f:193
         MAIN__:flow_lam.f:63
          main:flow_lam.f:67
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

An Example: SPEC bwaves

A pair of redundant computation

```
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
  __dvd: <no src>
   __mpexp:<no src>
    __mplog:<no src>
     __slowpow:<no src>
      __ieee754_pow_sse2:<no src>
       pow:<no src>
        jacobian_:jacobian_lam.f:47
         shell_:shell_lam.f:193
          MAIN__:flow_lam.f:63
           main:flow_lam.f:67
**************REDUNDANT WITH *************
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
 __dvd:<no src>
  __mpexp:<no src>
   __mplog:<no src>
    __slowpow:<no src>
     __ieee754_pow_sse2:<no src>
      pow:<no src>
       jacobian_:jacobian_lam.f:47
        shell_:shell_lam.f:193
         MAIN__:flow_lam.f:63
          main:flow_lam.f:67
```

An Example: SPEC bwaves

```
41.  ros = q(1, ip1, jp1, kp1)
42.  us = q(2, ip1, jp1, kp1)/ros
       …
47.  mu = (mu +
((gm-1.0d0)*(q(5,ip1,jp1,kp1)/ros-
    0.5d0*(us*us+vs*vs+ws*ws)))**0.75d0)/
2.0d0
```

A pair of redundant computation

```
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
  __dvd: <no src>
   __mpexp:<no src>
    __mplog:<no src>
     __slowpow:<no src>
      __ieee754_pow_sse2:<no src>
       pow:<no src>
        jacobian_:jacobian_lam.f:47
         shell_:shell_lam.f:193
          MAIN__:flow_lam.f:63
           main:flow_lam.f:67
 **************REDUNDANT WITH *************
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
 __dvd:<no src>
  __mpexp:<no src>
   __mplog:<no src>
    __slowpow:<no src>
     __ieee754_pow_sse2:<no src>
      pow:<no src>
       jacobian_:jacobian_lam.f:47
        shell_:shell_lam.f:193
         MAIN__:flow_lam.f:63
          main:flow_lam.f:67
```

An Example: SPEC bwaves

```
41.  ros = q(1, ip1, jp1, kp1)
42.  us = q(2, ip1, jp1, kp1)/ros
       …
47.  mu = (mu +
((gm-1.0d0)*(q(5,ip1,jp1,kp1)/ros-
    0.5d0*(us*us+vs*vs+ws*ws)))**0.75d0)/
2.0d0
```

A pair of redundant computation

```
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
  __dvd: <no src>
   __mpexp:<no src>
    __mplog:<no src>
     __slowpow:<no src>
      __ieee754_pow_sse2:<no src>
       pow:<no src>
        jacobian_:jacobian_lam.f:47
         shell_:shell_lam.f:193
          MAIN__:flow_lam.f:63
           main:flow_lam.f:67
**************REDUNDANT WITH *************
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
 __dvd:<no src>
  __mpexp:<no src>
   __mplog:<no src>
    __slowpow:<no src>
     __ieee754_pow_sse2:<no src>
      pow:<no src>
       jacobian_:jacobian_lam.f:47
        shell_:shell_lam.f:193
         MAIN__:flow_lam.f:63
          main:flow_lam.f:67
```

An Example: SPEC bwaves

```
41.  ros = q(1, ip1, jp1, kp1)
42.  us = q(2, ip1, jp1, kp1)/ros
       …
47.  mu = (mu +
((gm-1.0d0)*(q(5,ip1,jp1,kp1)/ros-
    0.5d0*(us*us+vs*vs+ws*ws)))**0.75d0)/
2.0d0
```

No insights without call path profiling

A pair of redundant computation

```
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
  __dvd: <no src>
   __mpexp:<no src>
    __mplog:<no src>
     __slowpow:<no src>
      __ieee754_pow_sse2:<no src>
       pow:<no src>
        jacobian_:jacobian_lam.f:47
         shell_:shell_lam.f:193
          MAIN__:flow_lam.f:63
           main:flow_lam.f:67
**************REDUNDANT WITH **************
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
 __dvd:<no src>
  __mpexp:<no src>
   __mplog:<no src>
    __slowpow:<no src>
     __ieee754_pow_sse2:<no src>
      pow:<no src>
       jacobian_:jacobian_lam.f:47
        shell_:shell_lam.f:193
         MAIN__:flow_lam.f:63
          main:flow_lam.f:67
```

An Example: SPEC bwaves

```
41.  ros = q(1, ip1, jp1, kp1)
42.  us = q(2, ip1, jp1, kp1)/ros
       …
47.  mu = (mu +
((gm-1.0d0)*(q(5,ip1,jp1,kp1)/ros-
    0.5d0*(us*us+vs*vs+ws*ws)))**0.75d0)/
2.0d0
```

No insights without call path profiling

A pair of redundant computation

```
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
  __dvd: <no src>
   __mpexp:<no src>
    __mplog:<no src>
     __slowpow:<no src>
      __ieee754_pow_sse2:<no src>
       pow:<no src>
        jacobian_:jacobian_lam.f:47
         shell_:shell_lam.f:193
          MAIN__:flow_lam.f:63
           main:flow_lam.f:67
 **************REDUNDANT WITH **************
movsdq  0x8(%rdi,%r10,8), %xmm0:__mul:<no src>
 __dvd:<no src>
  __mpexp:<no src>
   __mplog:<no src>
    __slowpow:<no src>
     __ieee754_pow_sse2:<no src>
      pow:<no src>
       jacobian_:jacobian_lam.f:47
        shell_:shell_lam.f:193
         MAIN__:flow_lam.f:63
          main:flow_lam.f:67
```

DrCCTProf: a framework that collects calling context for fine-grained profilers

DrCCTProf Overview

- Functionality
  - capture call path for each dynamic instruction
  - capture the data object read/written by each memory access
    - heap data objects: call paths to the allocations
    - static data objects: names from symbol table

- Programmability
  - APIs provide request-based service for clients

- Overhead
  - moderate overhead in both runtime and space

DrCCTProf Tutorial Outline

software download
https://github.com/Xuhpclab/DrCCTProf

- DrCCTProf APIs
- General usage of DrCCTProf framework for fine-grained profilers
- DrCCTProf framework internals

DrCCTProf Tutorial Outline

software download
https://github.com/Xuhpclab/DrCCTProf

- DrCCTProf APIs
- General usage of DrCCTProf framework for fine-grained profilers
- DrCCTProf framework internals

A Simple DrCCTProf Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
	void *drcontext = dr_get_current_drcontext();
	context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
	dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
		(void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
	drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
	dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
			"http://dynamorio.org/issues");
	ClientInit(argc, argv);

	drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);

	dr_register_exit_event(ClientExit);
}
```

A Simple DrCCTProf Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
	void *drcontext = dr_get_current_drcontext();
	context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
	dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
		(void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
	drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
	dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
			"http://dynamorio.org/issues");
	ClientInit(argc, argv);

	drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);

	dr_register_exit_event(ClientExit);
}
```

Functional APIs

```
drcctlib_init(bool (*IsInterestingIns)(instr_t *), file_t logFile,
              void (*userCallBack)(void *, instr_instrument_msg_t *),
              bool doDataCentric);
```

- Description:
  - DrCCTProf clients must call this before using DrCCTProf
- Arguments:
  - IsInterestingIns: tells whether a given INS needs to collect context; predefined: INTERESTING_INS_ALL, INTERESTING_INS_NONE, INTERESTING_INS_MEMORY_ACCESS.
  - logFile: file pointer where DrCCTProf put output data
  - userCallBack: a client callback on each INS that IsInterestingIns is true
  - doDataCentric: should be set if the client wants DrCCTProf to do data-centric attribution

A Simple CCTLib Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
    void *drcontext = dr_get_current_drcontext();
    context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
    dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
        (void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
    drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
                "http://dynamorio.org/issues");
    ClientInit(argc, argv);
    drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);

    dr_register_exit_event(ClientExit);
}
```

A Simple CCTLib Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
    void *drcontext = dr_get_current_drcontext();
    context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
    dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
        (void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
    drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
                "http://dynamorio.org/issues");
    ClientInit(argc, argv);
    drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);
                        monitor every instruction
    dr_register_exit_event(ClientExit);
}
```

A Simple CCTLib Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
    void *drcontext = dr_get_current_drcontext();
    context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
    dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
        (void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
    drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
                "http://dynamorio.org/issues");
    ClientInit(argc, argv);
    drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);
                    monitor every instruction
    dr_register_exit_event(ClientExit);
}
```

A Simple CCTLib Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
    void *drcontext = dr_get_current_drcontext();
    context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
    dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
        (void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
    drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
                       "http://dynamorio.org/issues");
    ClientInit(argc, argv);
    drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);
                          monitor every instruction
    dr_register_exit_event(ClientExit);
}
```

A Simple CCTLib Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
    void *drcontext = dr_get_current_drcontext();
    context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
    dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
        (void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
    drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
                       "http://dynamorio.org/issues");
    ClientInit(argc, argv);
    drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);
                  monitor every instruction
    dr_register_exit_event(ClientExit);
}
```

instrumentation routine

A Simple CCTLib Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
    void *drcontext = dr_get_current_drcontext();
    context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
    dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
        (void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
    drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
                       "http://dynamorio.org/issues");
    ClientInit(argc, argv);
    drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);
    dr_register_exit_event(ClientExit);
}
```

instrumentation routine

monitor every instruction

no data-centric

A Simple CCTLib Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
    void *drcontext = dr_get_current_drcontext();
    context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
    dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
        (void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
    drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
                       "http://dynamorio.org/issues");
    ClientInit(argc, argv);
    drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);
    dr_register_exit_event(ClientExit);
}
```

instrumentation routine

monitor every instruction

no data-centric

A Simple CCTLib Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
    void *drcontext = dr_get_current_drcontext();
    context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}

void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
    dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
        (void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}

void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
    drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
                       "http://dynamorio.org/issues");
    ClientInit(argc, argv);
    drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);
    dr_register_exit_event(ClientExit);
}
```

instrumentation routine

monitor every instruction

no data-centric

A Simple CCTLib Client

```
#include <iterator>
#include "dr_api.h"
#include "drcctlib.h"
FILE* gTraceFile;

void InsertCleancall(int32_t slot) {
    void *drcontext = dr_get_current_drcontext();
    context_handle_t cur_ctxt_hndl = drcctlib_get_context_handle(drcontext, slot);
}
```

analysis routine

```
void InstrumentIns(void *drcontext, instr_instrument_msg_t *instrument_msg) {
    dr_insert_clean_call(drcontext,  instrument_msg->bb, instrument_msg->instr,
        (void *)InsertCleancall, false, 1, OPND_CREATE_INT32(instrument_msg->slot));
}
```

instrumentation routine

```
void ClientInit(int argc, const char *argv[]) {

}

void ClientExit(void) {
    drcctlib_exit();
}

void dr_client_main(client_id_t id, int argc, const char *argv[]) {
    dr_set_client_name("DynamoRIO Client 'drcctlib_cct_only_clean_call'",
                       "http://dynamorio.org/issues");
    ClientInit(argc, argv);
    drcctlib_init(INTERESTING_INS_ALL, gTraceFile, InstrumentIns, false);
    dr_register_exit_event(ClientExit);
}
```

monitor every instruction

no data-centric

Functional APIs

- context_handle_t **drcctlib_get_context_handle**(void *drcontext, int32_t opaqueHandle);
  - Description:
    - Get the calling context handle (**context_handle_t**)
  - Arguments:
    - drcontext: Dynamorio's thread private context of the asking thread.
    - opaqueHandle: handle passed by DrCCTProf to the client tool in its userCallback.

- data_handle_t **drcctlib_get_data_hndl**(void *drcontext, void *address);
  - Description:
    - Call when need the handle to a data object (**data_handle_t**)
  - Arguments:
    - drcontext: Dynamorio's thread private context of the asking thread.
    - address: effective address for which the data object is needed.

Data Structures

- **context_handle_t**;
  - ✦ serve as ID of a context
- **data_handle_t**
  - ✦ enum objectType:
    - ✳ STACK_OBJECT, DYNAMIC_OBJECT, STATIC_OBJECT, UNKNOW
  - ✦ union {path_handle, sym_name}:
    - ✳ path_handle represents the allocation point of dynamic data
    - ✳ sym_name represents the name of static data
  - ✦ uint64_t beg_addr
  - ✦ uint64_t end_addr
  - ✦ Used in data-centric analysis

Functional APIs

- context_t * **drcctlib_get_full_cct**(context_handle_t ctxtHandle, int maxDepth);
  - Description:
    - Return the **full calling context list** whose handle is ctxtHandle
  - Arguments:
    - ctxtHandle: the context handle which request the call path
    - maxDepth: the max depth of the returned calling context

- void **drcctlib_print_full_cct**(context_handle_t ctxtHandle, int maxDepth);
  - Description:
    - Prints the full calling context whose handle is ctxtHandle
    - Often used when log the analysis result to file
  - Arguments:
    - ctxtHandle: the context handle which request the call path
    - maxDepth: the max depth of the printed calling context

Functional APIs

- char * **drcctlib_get_str_from_strpool**(int32_t index);
  - Description:
    - Get the char string for a symbol from string pool index
    - Often used to get the name of Static data objects(pass sym_name)
  - Arguments:
    - index: a string pool index

- bool **drcctlib_have_same_caller_prefix**(context_handle_t ctxt1, context_handle_t ctxt2);
  - Description:
    - Tell if the call path from root to leaves except the leaves themselves are the same for the given two context handles

- bool **drcctlib_have_same_source_line**(context_handle_t ctxt1, context_handle_t ctxt2);
  - Description:
    - Tell if the two given context handles point to the same source line, no matter whether the full call path is same or not

DrCCTProf Tutorial Outline

software download
https://github.com/Xuhpclab/DrCCTProf

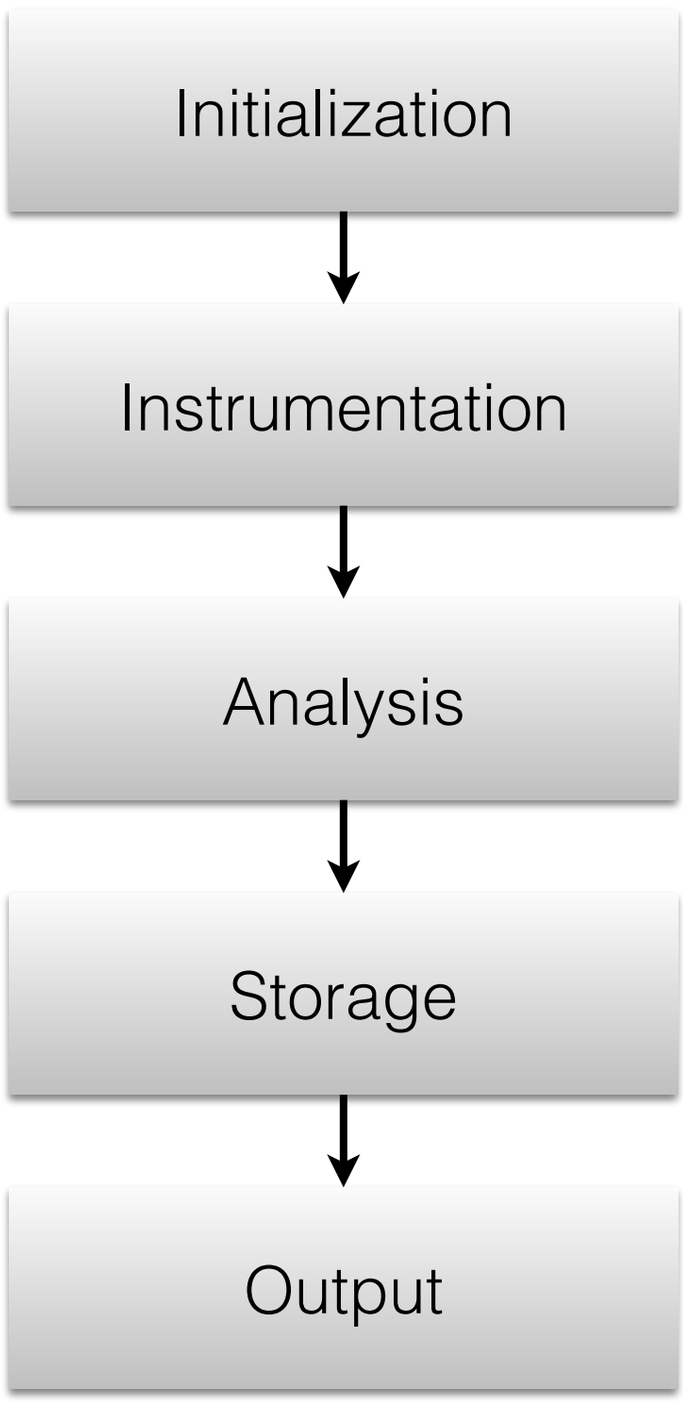
- DrCCTProf APIs
- General usage of DrCCTProf framework for fine-grained profilers
- DrCCTProf framework internals

DrCCTProf Tutorial Outline

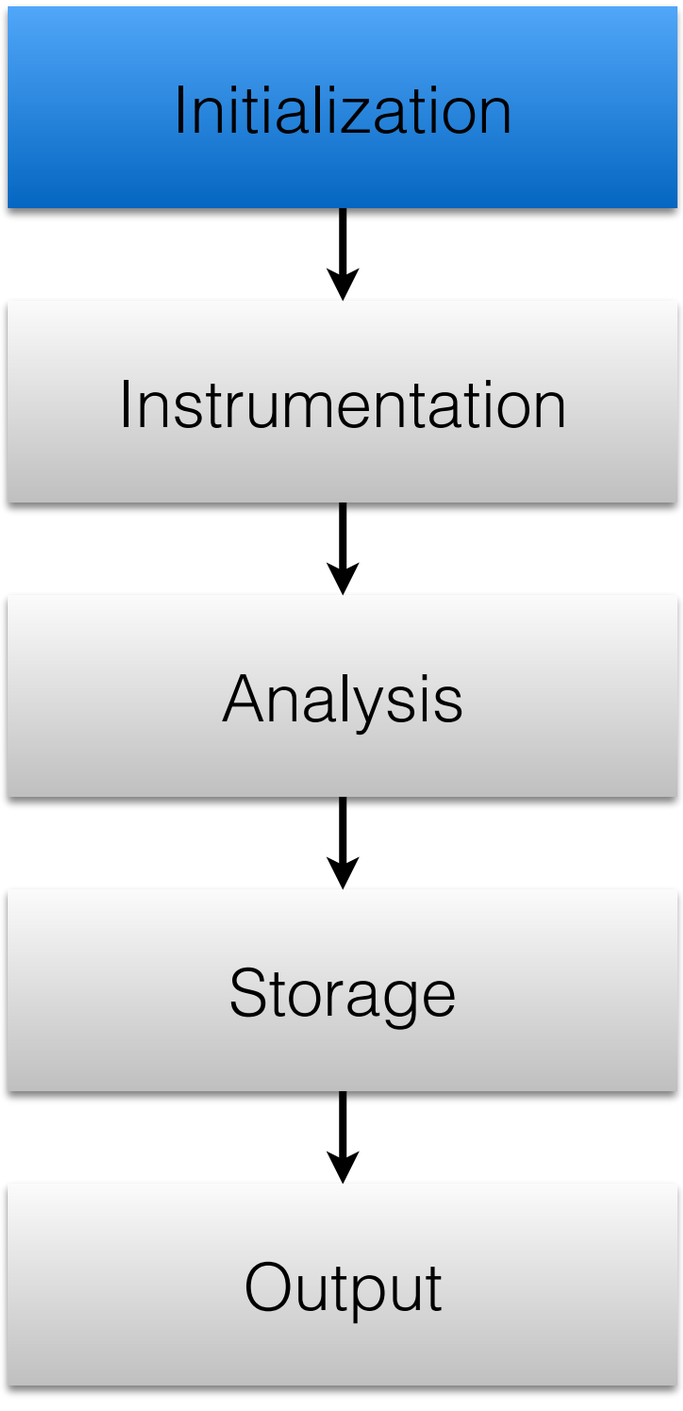
software download
https://github.com/Xuhpclab/DrCCTProf

- DrCCTProf APIs
- General usage of DrCCTProf framework for fine-grained profilers
- DrCCTProf framework internals

Work Flow to Prepare DrCCTProf Clients

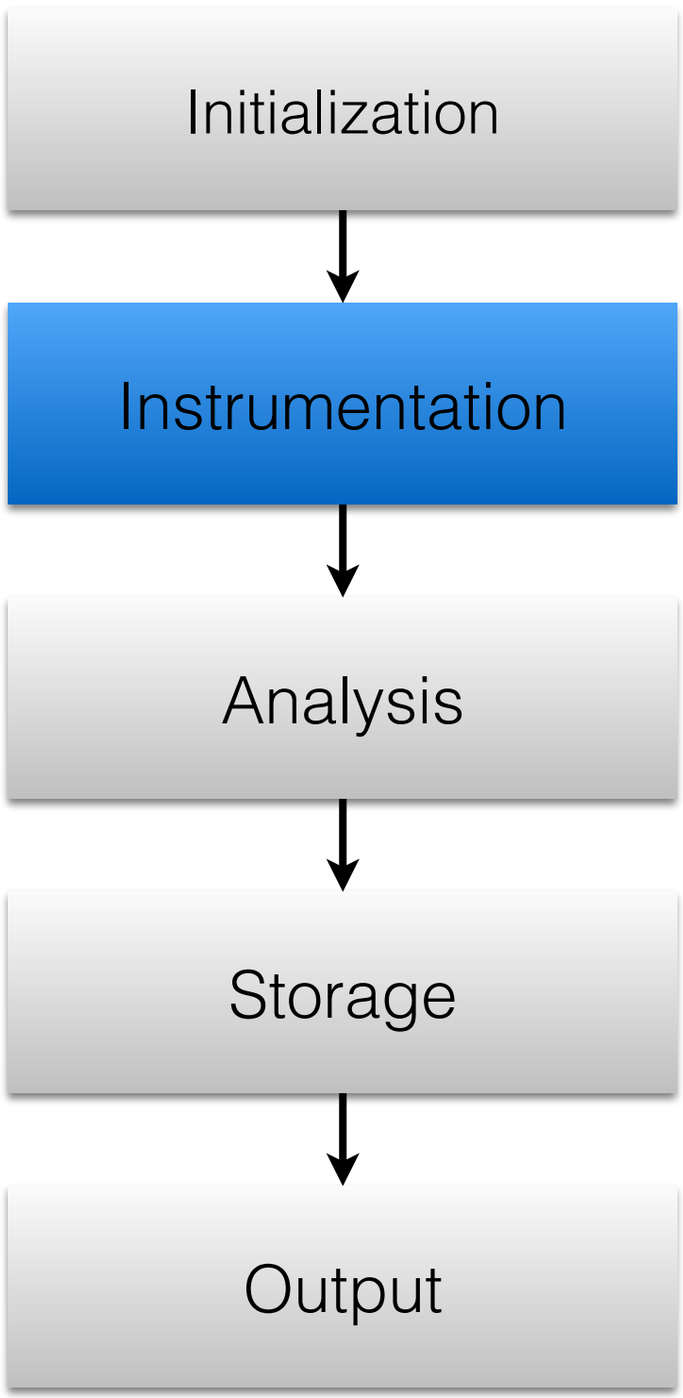


Work Flow to Prepare DrCCTProf Clients



- Initialize Dynamorio components required by the client
  - drmgr_init();
  - drsym_init();
  - drwrap_init();
  - …
- Initialize DrCCTProf and client
  - Create output file for client
  - **drcctlib_init()**
    - Initial instruction to focus
    - Enable Data-centric or not
- Initialize data structures
  - Obtain key for TLS
    - drmgr_register_tls_field()
  - Define & initialize data items
    - Define a structure with elements you want to store per thread

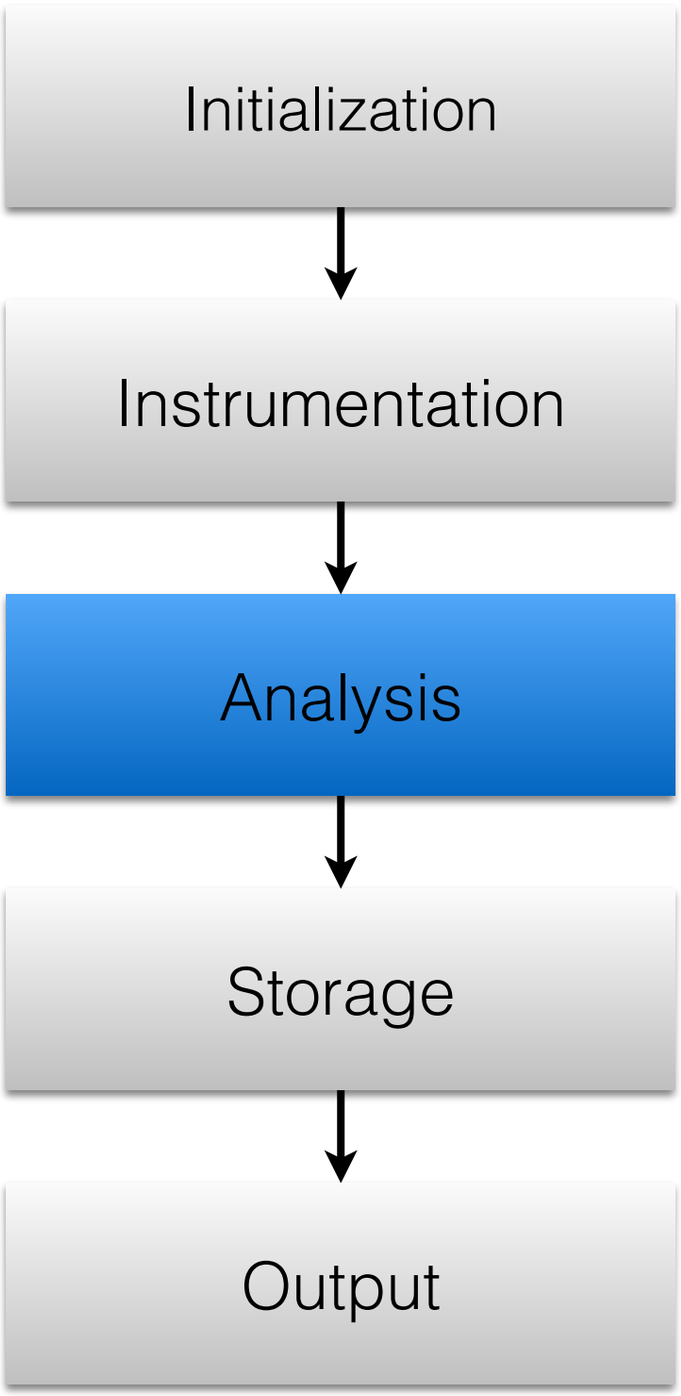
Work Flow to Prepare DrCCTProf Clients



- Instrumentation APIs
  - drmgr_register_bb_instrumentation_event()
  - drmgr_register_module_load_event()
  - drwrap_wrap()
  - ….
- INS insert calls
  - drcontext: Dynamorio’s thread private context of the asking thread.
  - internal_instrument_msg: contain Dynamorio's basic block struct, Dynamorio's instruction struct, and opaque handle passed by DrCCTProf

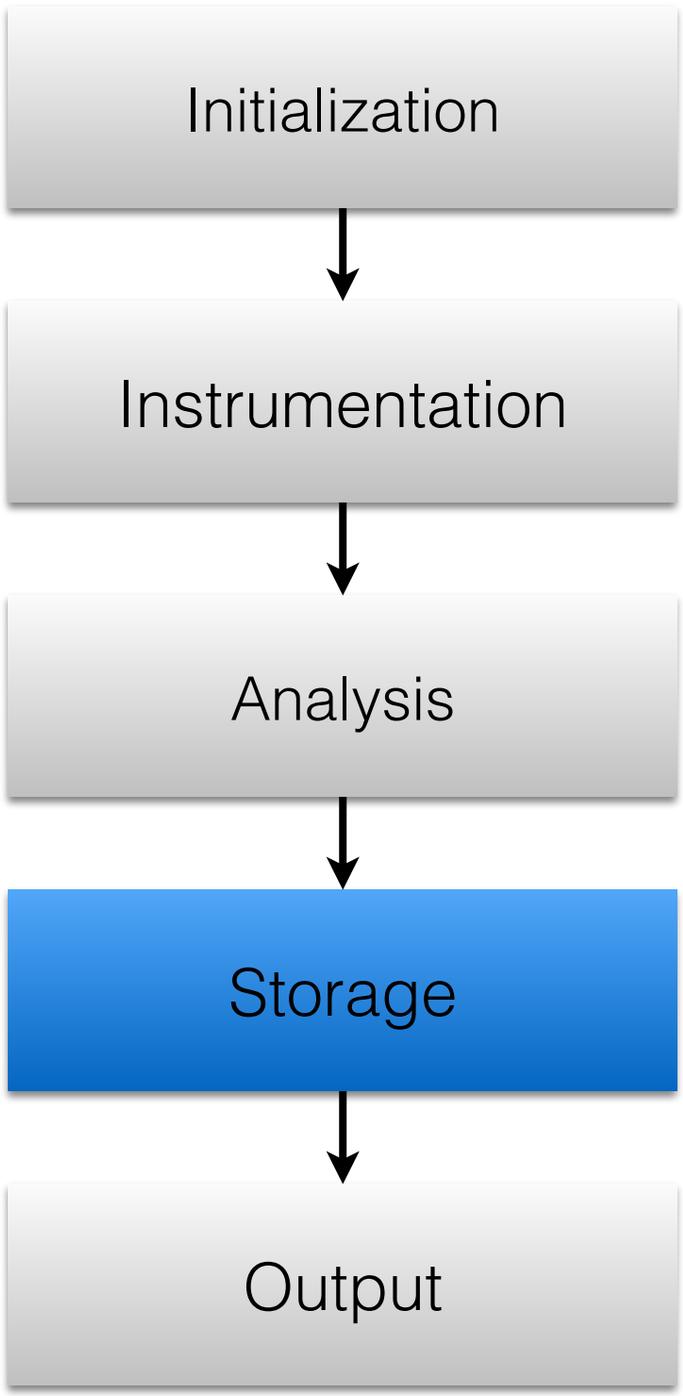
```
InstrumentIns(void *drcontext, instr_instrument_msg_t *internal_instrument_msg)
{
   ……
   ……
   dr_insert_clean_call(drcontext,
            internal_instrument_msg->bb, internal_instrument_msg->instr,
            (void *)InsertCleancall, false, 1,
            OPND_CREATE_INT32(internal_instrument_msg->slot));
}
```

Work Flow to Prepare DrCCTProf Clients



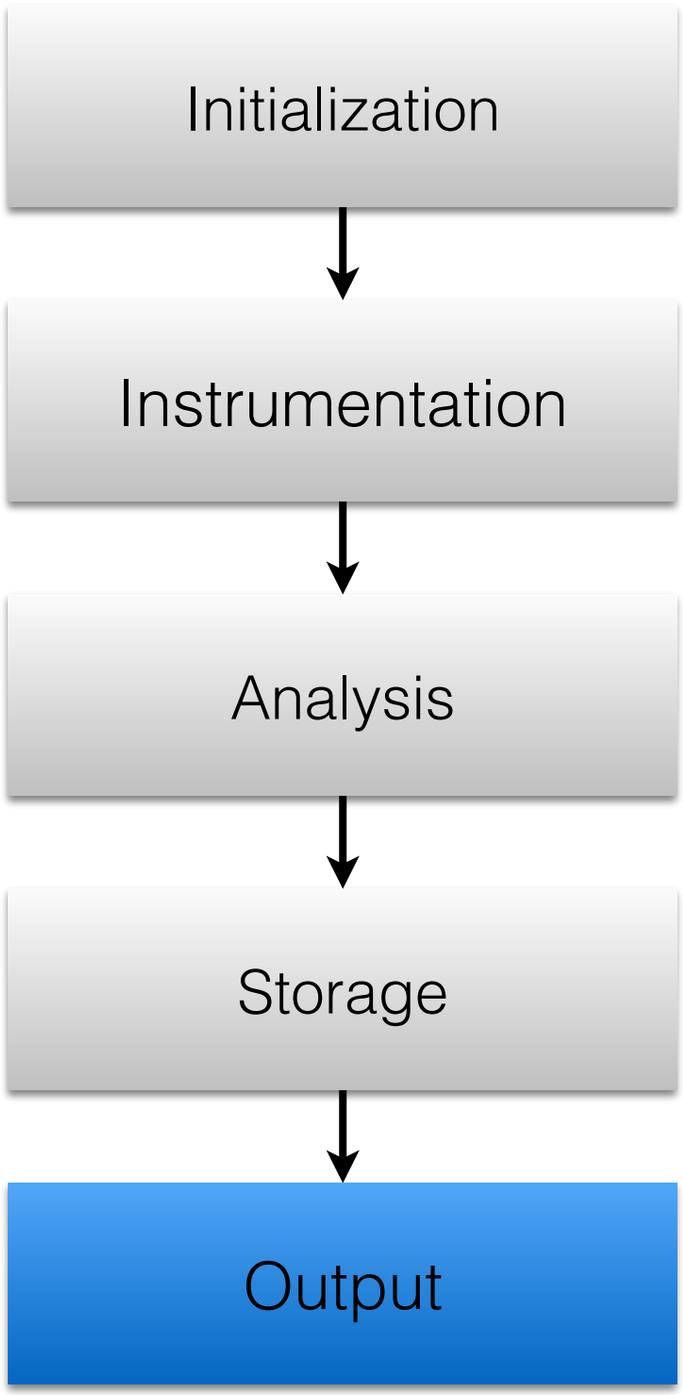
- Analysis code inside the inserted call
  - User define
  - Find some "**relation**" between instructions
    - redundancy, reuse, …
- Some common used routines
  - Get the data structure for current thread
  - Get the CCT handler of current ins
  - Instruction pair (**related**)
    - <CCT handler1, CCT handler2>

Work Flow to Prepare DrCCTProf Clients



- Shadow memory
  - ✦ Information corresponding to each memory address
- Map<uint64_t, uint64_t>
  - ✦ Key: build from two 32bits CCT handler
  - ✦ Value: frequency that the "**relation**" between the two instructions occur

Work Flow to Prepare DrCCTProf Clients



- Merge Items in Map
  - ✦ Aggregate metrics that associated with the same source line
- Sort the Map info
  - ✦ Based on the frequency
- Log “**relation**” to file
  - ✦ **drcctlib_print_full_cct**(handler)
  - ✦ Use **dr_mutex_lock**(void* lock) to manipulate one log file for multithreading

DrCCTProf Tutorial Outline

software download
https://github.com/Xuhpclab/DrCCTProf

- DrCCTProf APIs
- General usage of DrCCTProf framework for fine-grained profilers
- DrCCTProf framework internals

DrCCTProf Tutorial Outline

software download
https://github.com/Xuhpclab/DrCCTProf

- DrCCTProf APIs
- General usage of DrCCTProf framework for fine-grained profilers
- DrCCTProf framework internals

Why Maintain Call Path History?

- To associate context of one event of interest with the context of another event
  - No priory knowledge of whether the current context will be needed in the future

Example

- Correctness tools
  - Data race detection
  - Taint analysis
  - Array out of bound detection
- Performance analysis tools
  - Reuse-distance analysis
  - Cache simulation
  - False sharing detection
  - Redundancy detection (e.g. dead writes)
- Other tools
  - Debugging, testing, resiliency, replay, etc.

Scale of Call Paths

	Description	Original program running for 10 minutes
Debuggers	On each break point	**$< 10^3$**
Performance analysis tools	On each sample (1 sample/ms)	**6×10^5**
Fine-grained instrumentation tools	On each instruction (2GHz CPU)	**1.2×10^{12}**

State-of-the-art in Collecting Ubiquitous Call Paths

“It will slow down execution by a factor of several thousand compared to native execution -- I'd guess -- so you'll wind up with something that is unusably slow on anything except the smallest problems.”

Dyninst

“If you tried to invoke Thread::getCallStack on every memory access there would be very serious performance problems … your program would probably never reach main.”

Pin

No support for collecting calling contexts

State-of-the-art in Collecting Ubiquitous Call Paths

“It will slow down execution by a factor of several thousand compared to native execution -- I'd guess -- so you'll wind up with something that is unusably slow on anything except the smallest problems.”

Dyninst

“If you tried to invoke Thread::getCallStack on every memory access there would be very serious performance problems … your program would probably never reach main.”

Pin

No support for collecting calling contexts

We built one ourselves—**DrCCTProf**

Challenges in Ubiquitous Call Path Collection

1. Overhead (space)
2. Overhead (time)
3. Overhead (parallel scaling)

Store History of Contexts Compactly

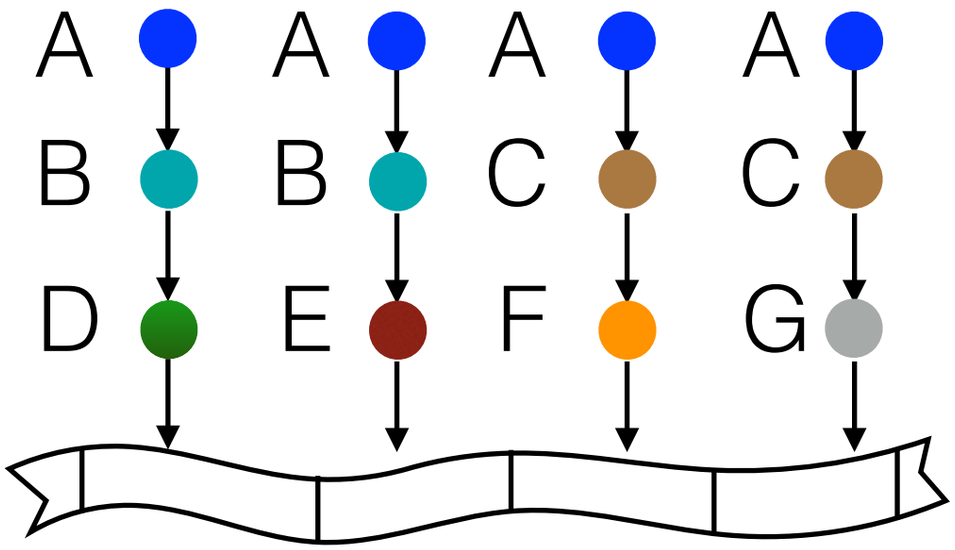
<u>Problem:</u>

Deluge of call paths

Store History of Contexts Compactly

<u>Problem:</u>

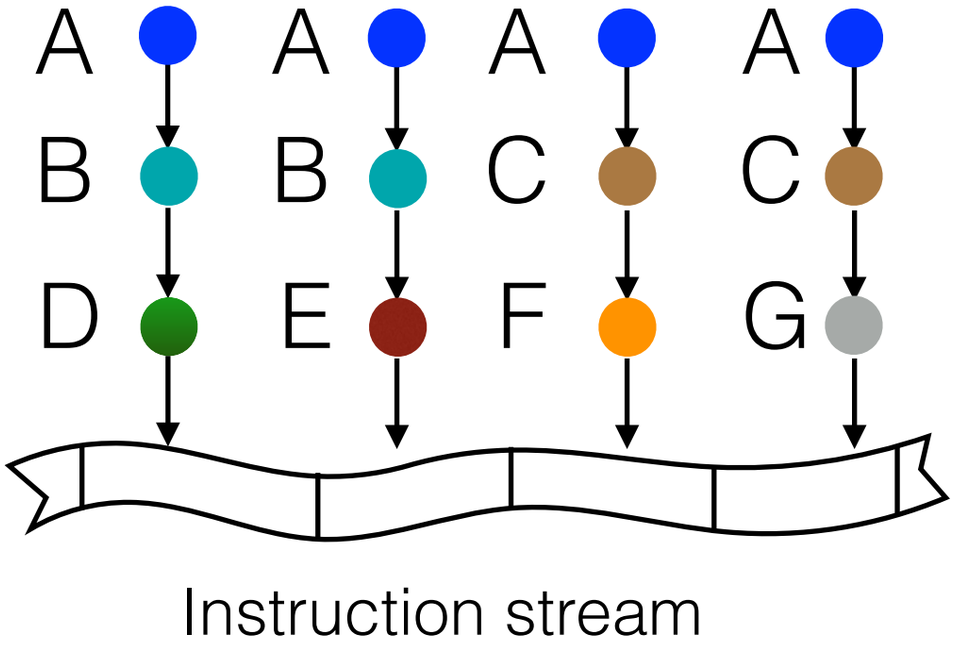
Deluge of call paths



Store History of Contexts Compactly

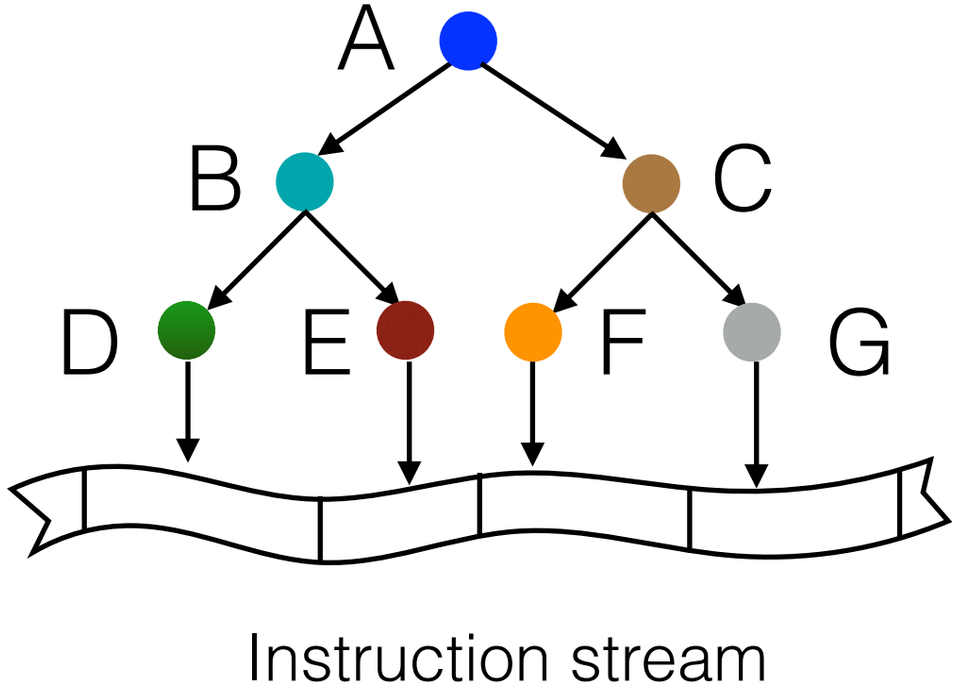
Problem:

Deluge of call paths



Solution

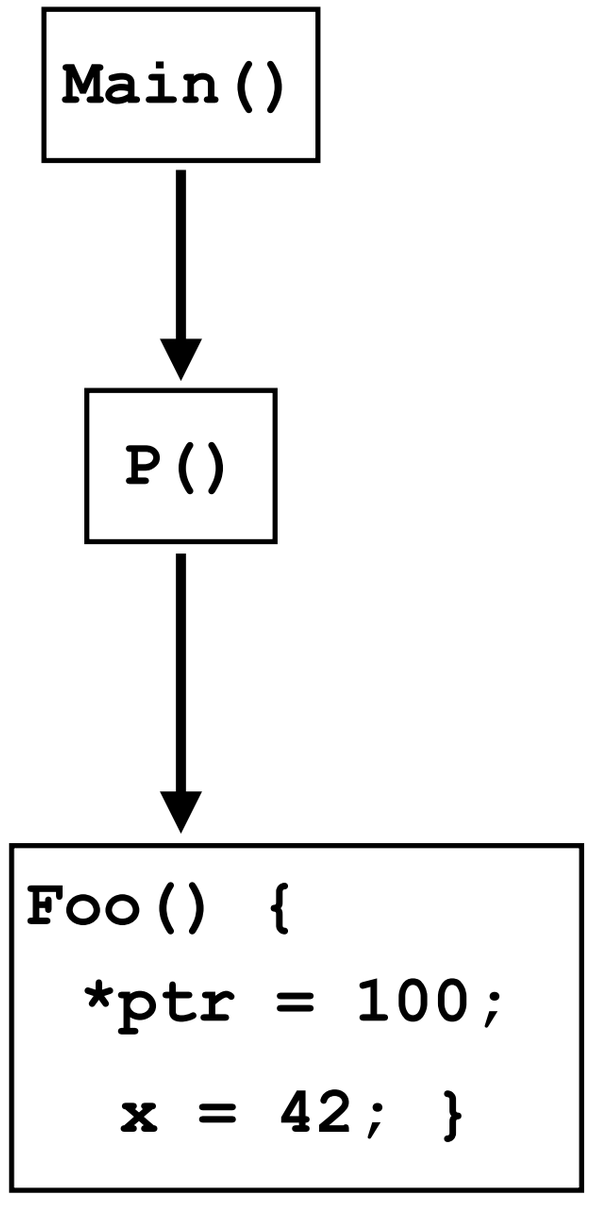
- Call paths share common prefix
- Store call paths as a calling context tree (CCT)
- One CCT per thread



Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

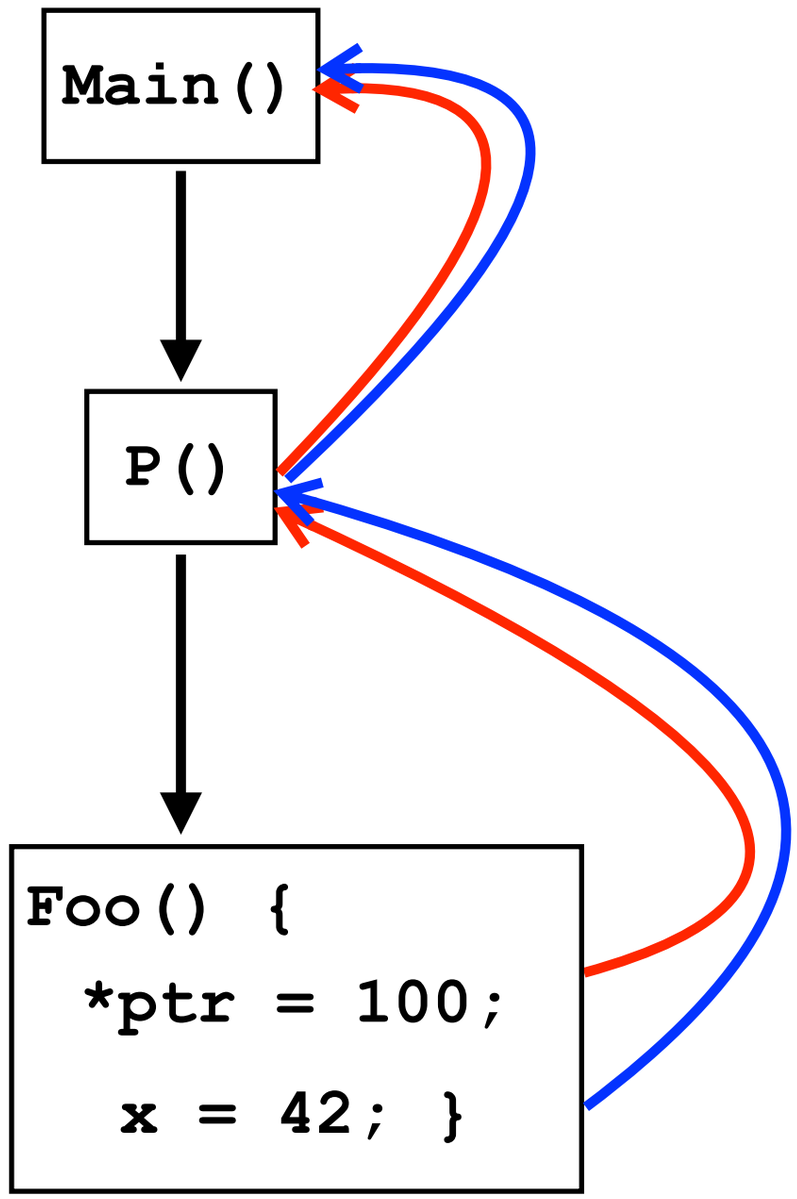
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.



Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

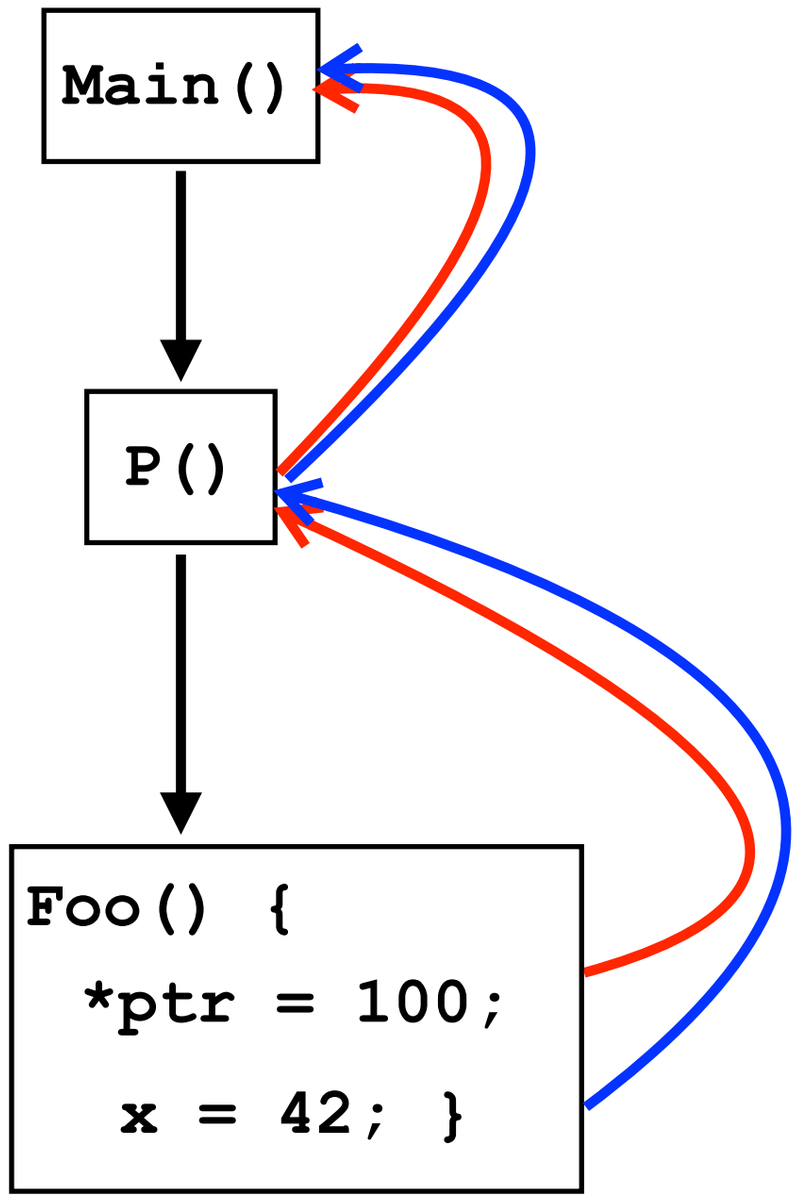
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

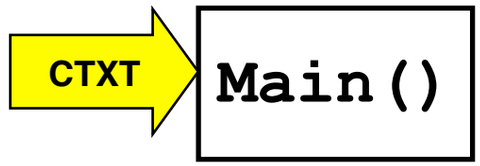


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

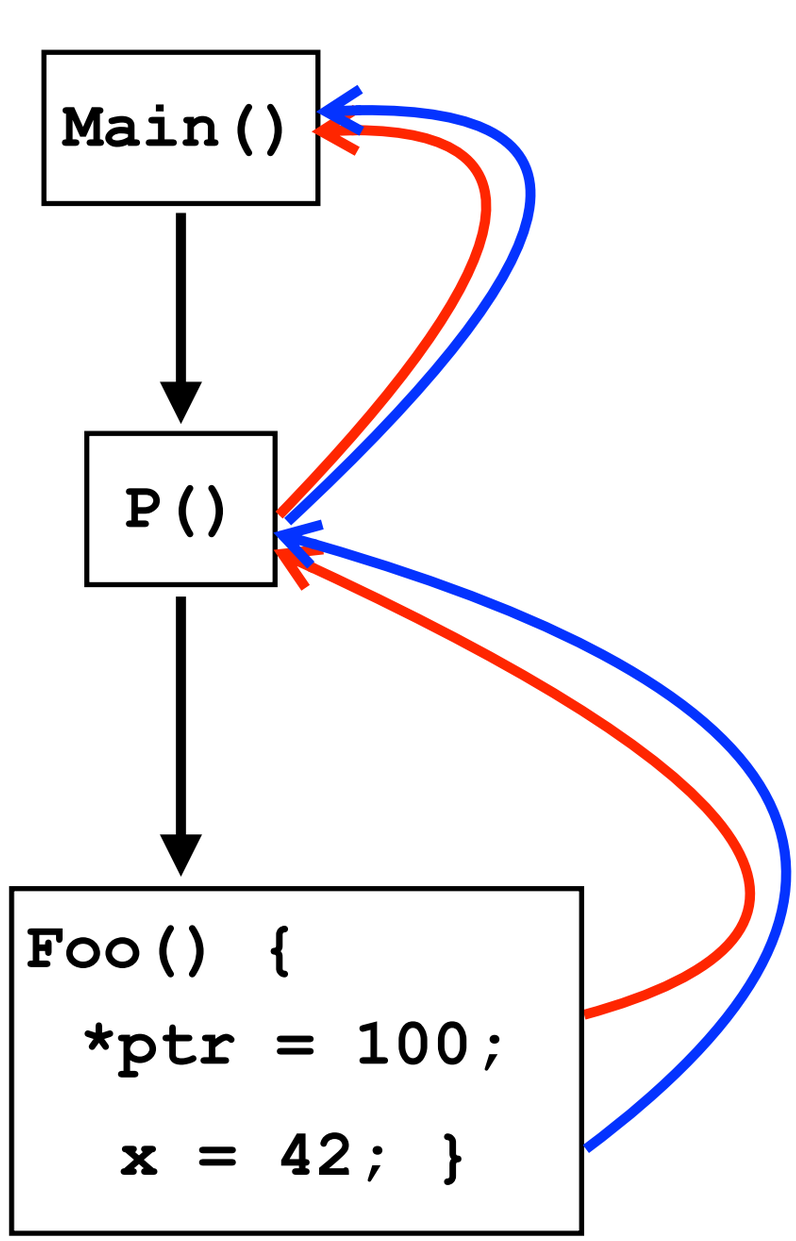


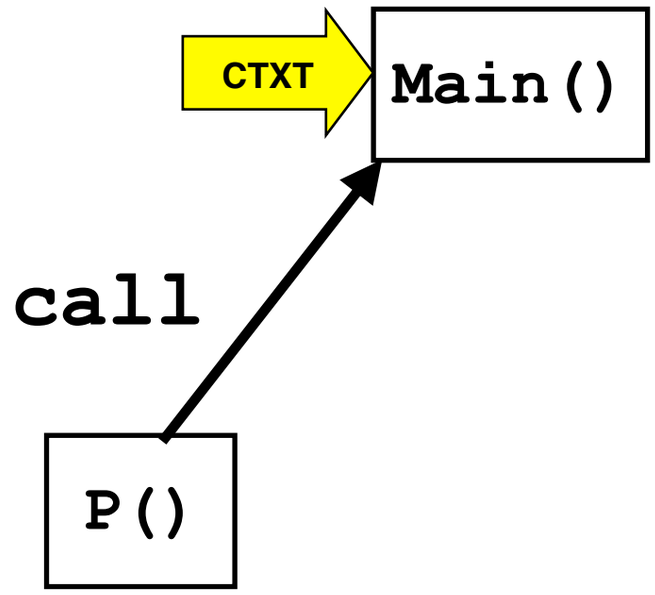


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

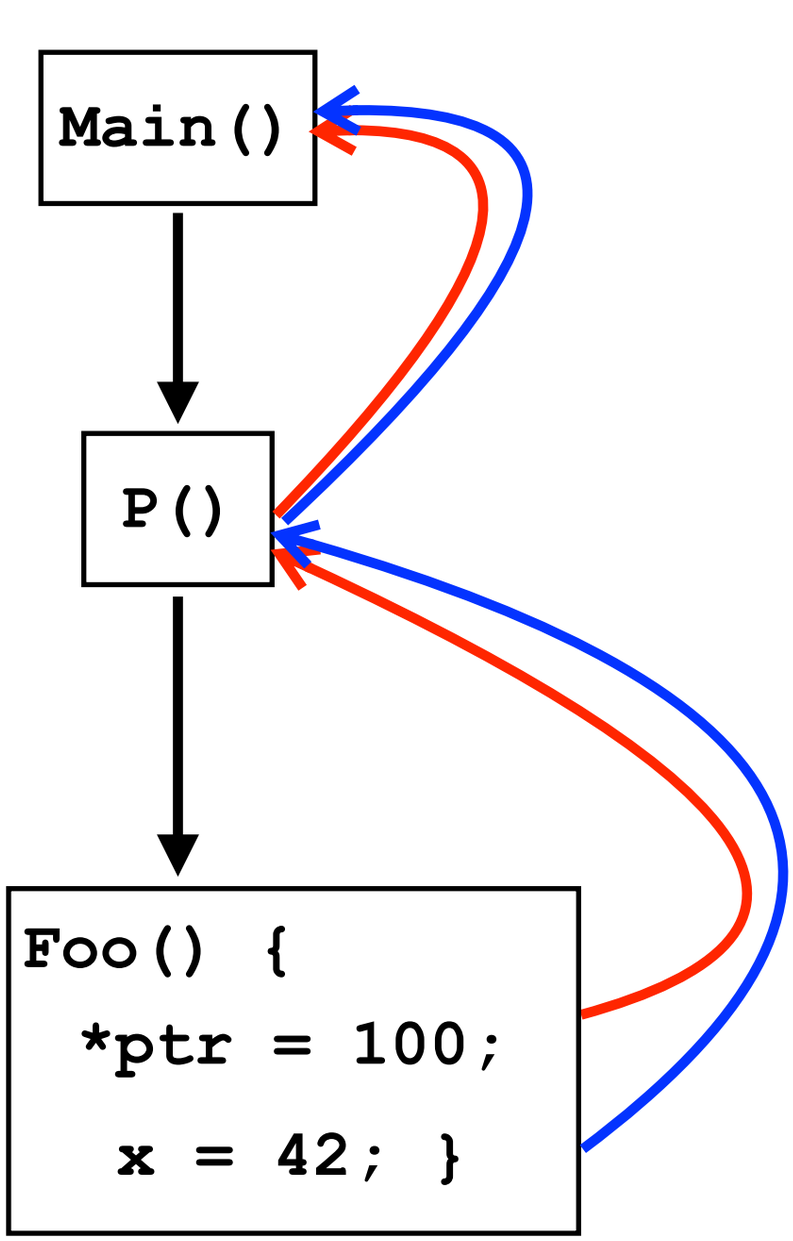


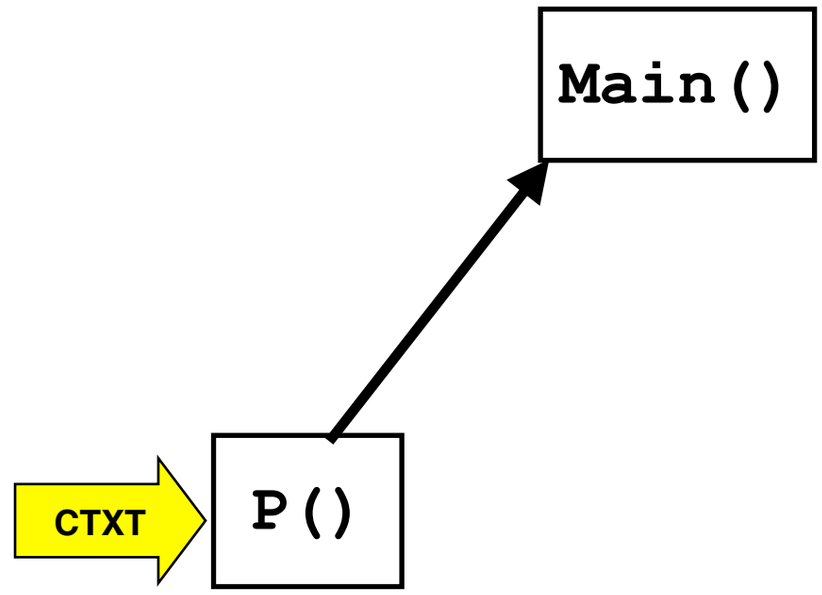


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

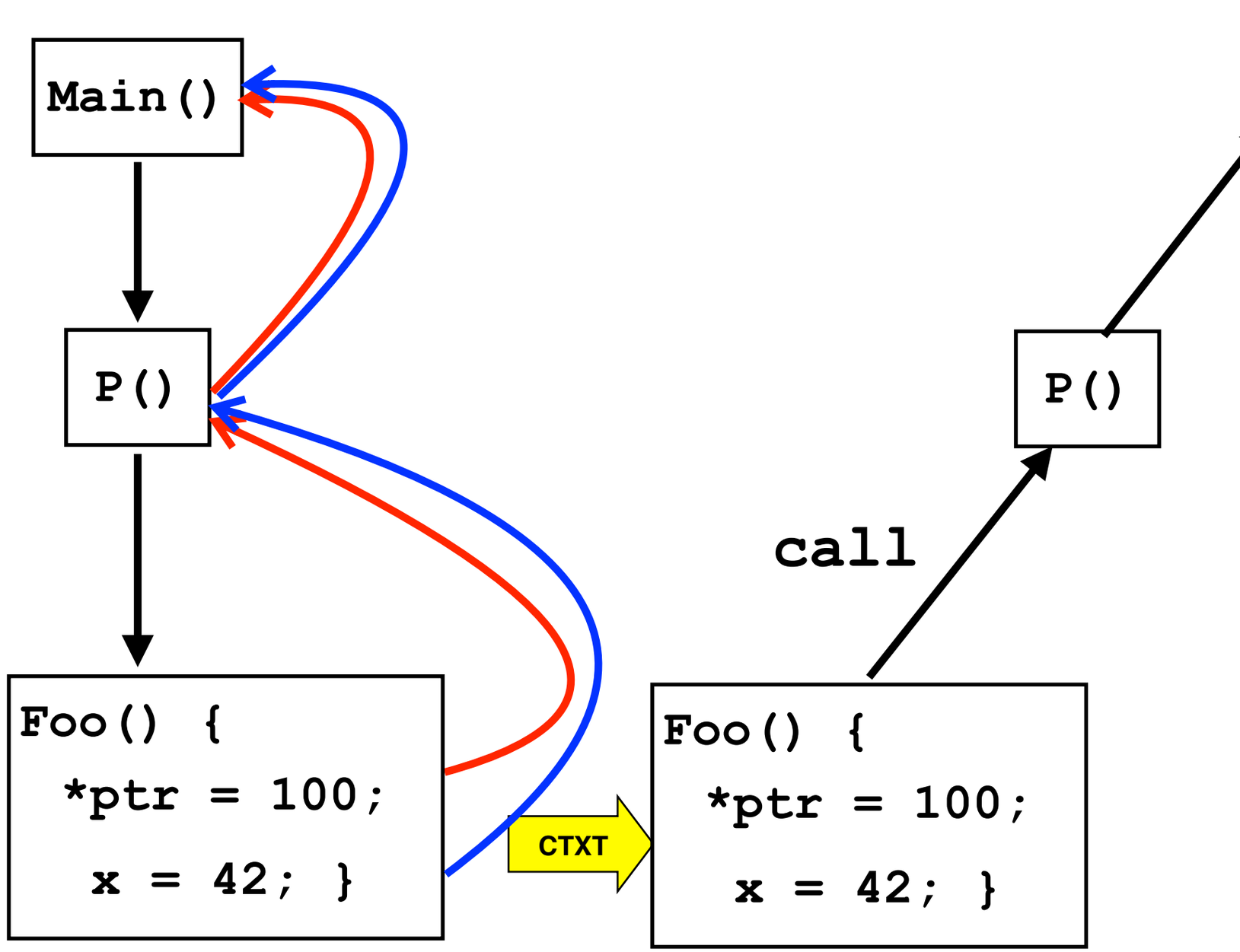




Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

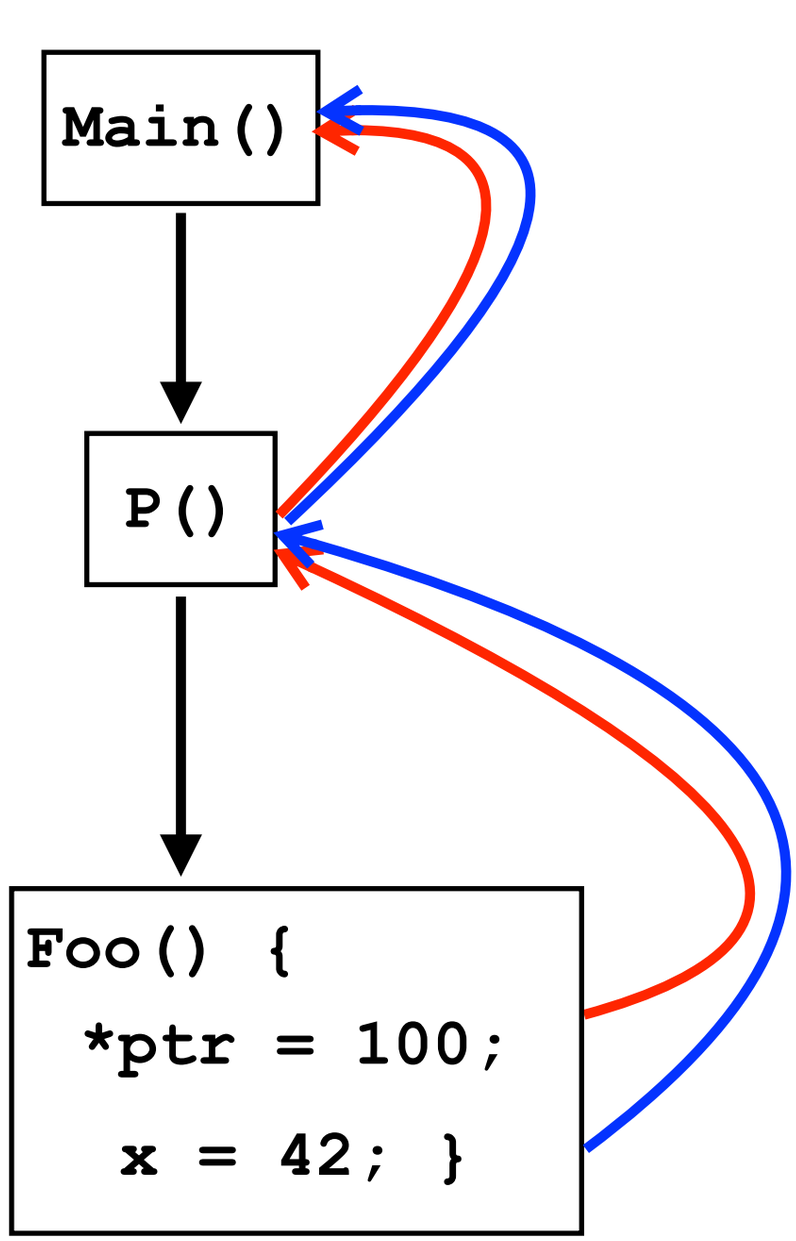


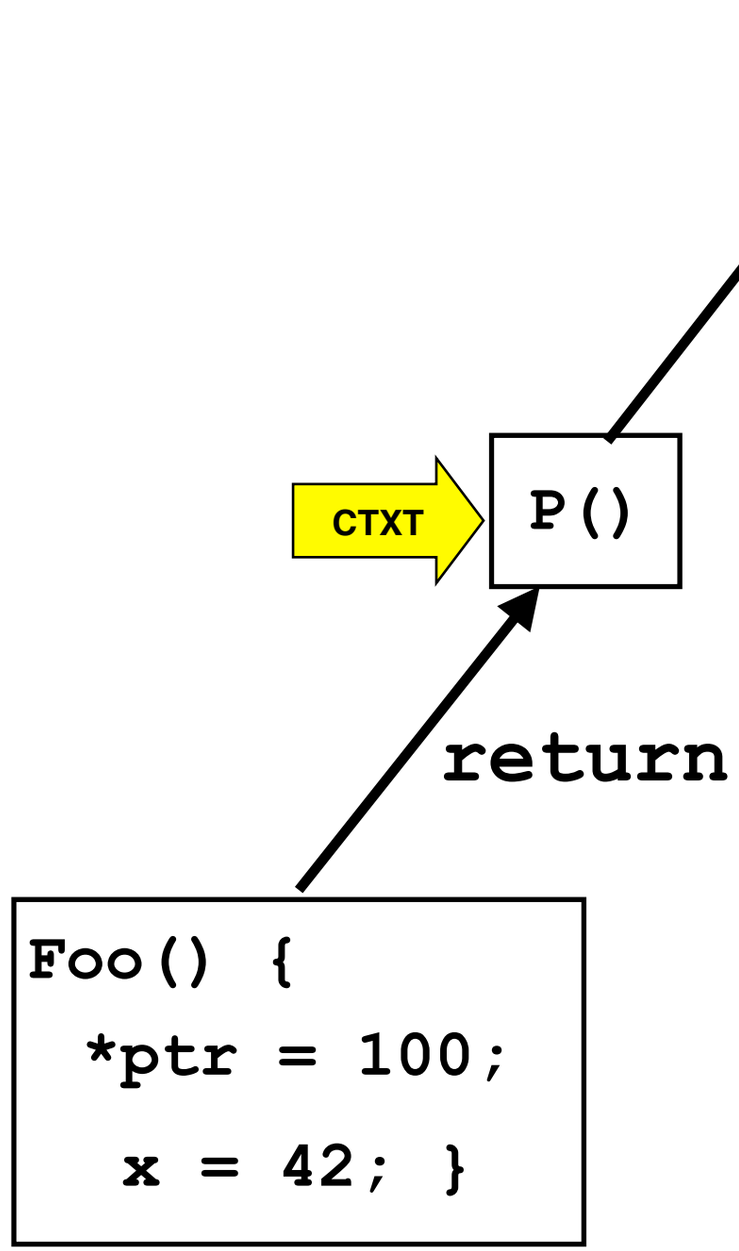
Tools can obtain pointer to the current context via CTXT in constant time

Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.



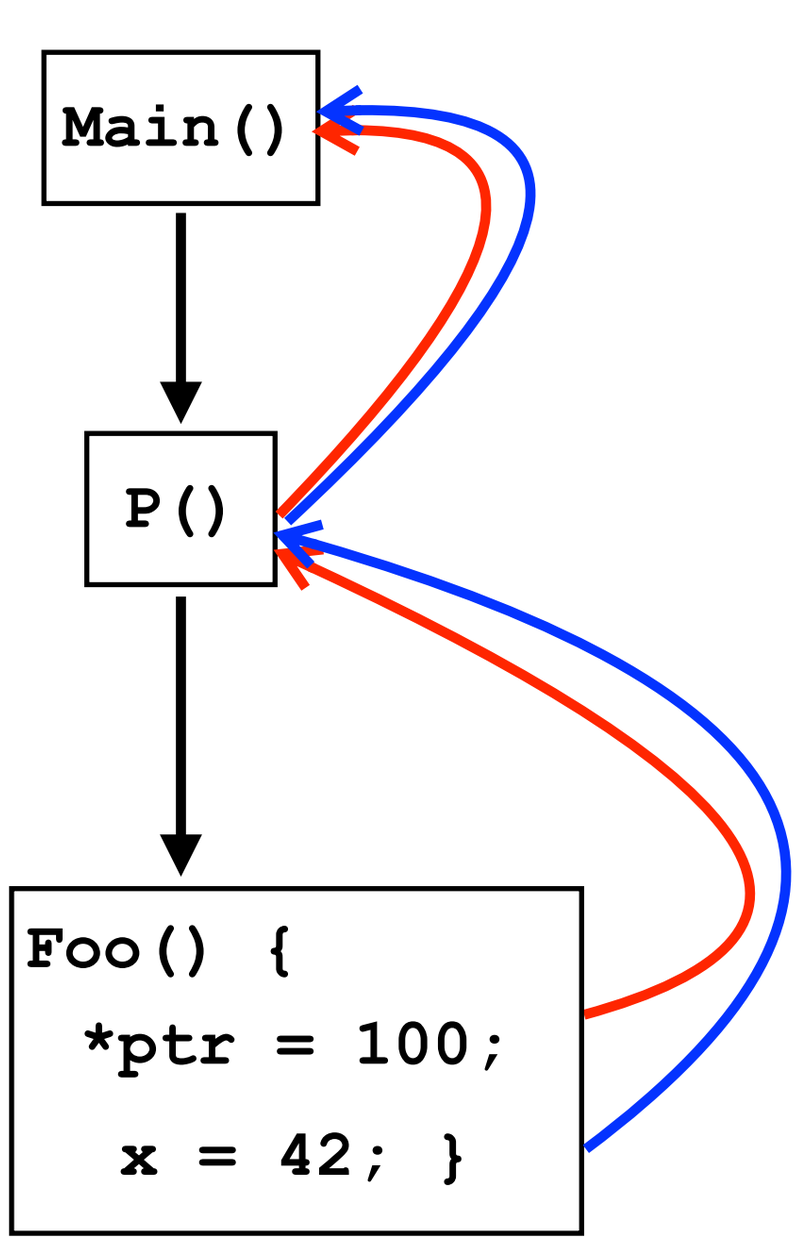


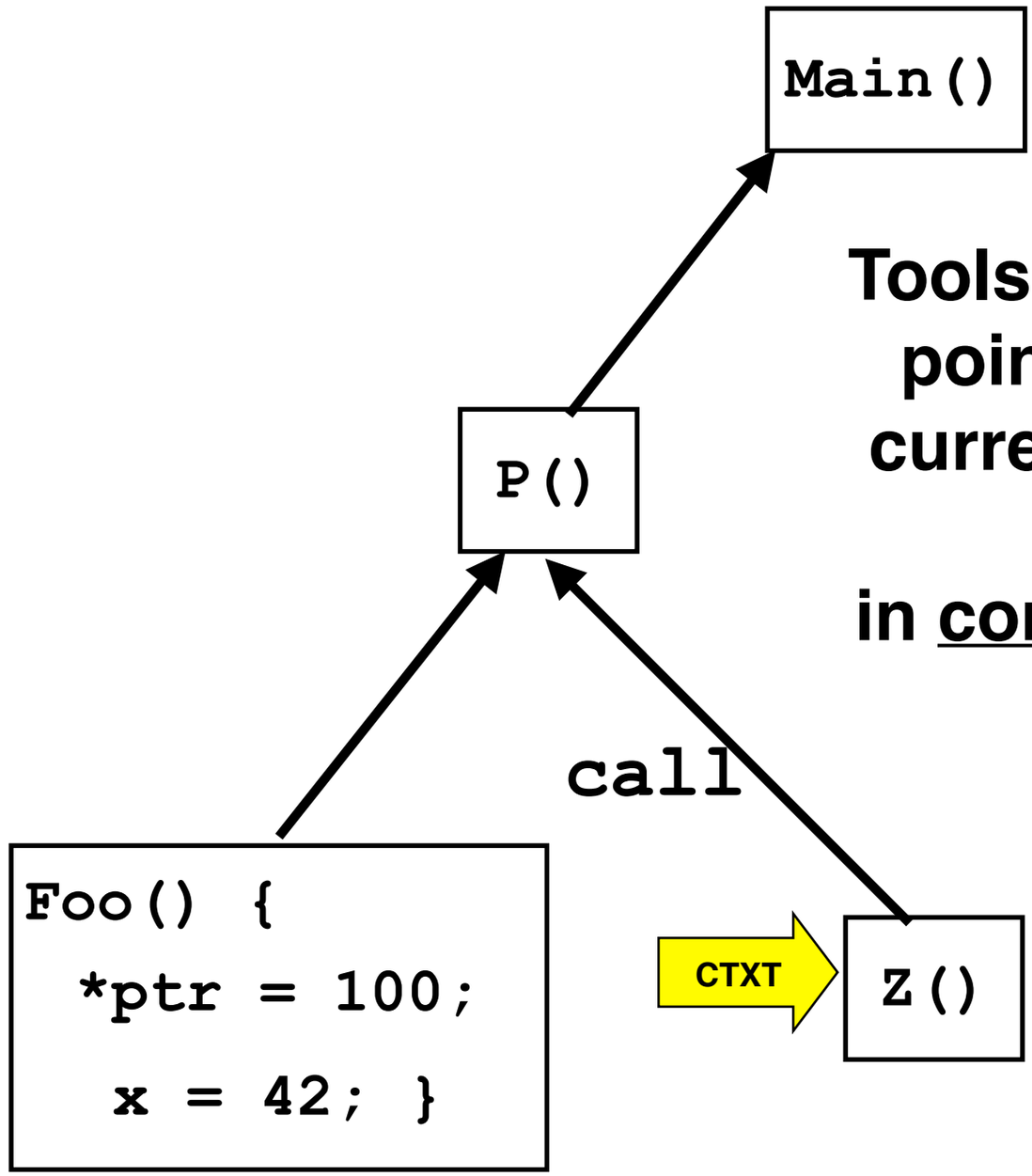
Tools can obtain pointer to the current context via CTXT in constant time

Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.





Tools can obtain pointer to the current context via CTXT in constant time

Complexities of Context Management

- Problem #1: Call/Return instructions are not simple to spot
  - ✦ Static disassembly is imperfect
    - ✳ Data embedded in instruction stream
    - ✳ Missing or incorrect function boundary information

- Problem #2: CTXT needs call-path + PC for completeness
  - ✦ Naively using the <Context:PC> tuple makes each handle >64 bits
  - ✦ We wish to keep a handle 32-bit since tools often use two 32-bit (64-bit) handles as a hash-table index

- Solution: basic block instrumentation
  - ✦ DynamoRIO instruments every BB
  - ✦ Call/return instructions discovered in a BB are reliable
  - ✦ DrCCTProf assigns a unique 32-bit id to each instruction appearing in each call path in each trace —> supports 4.2B unique contexts (including PC)

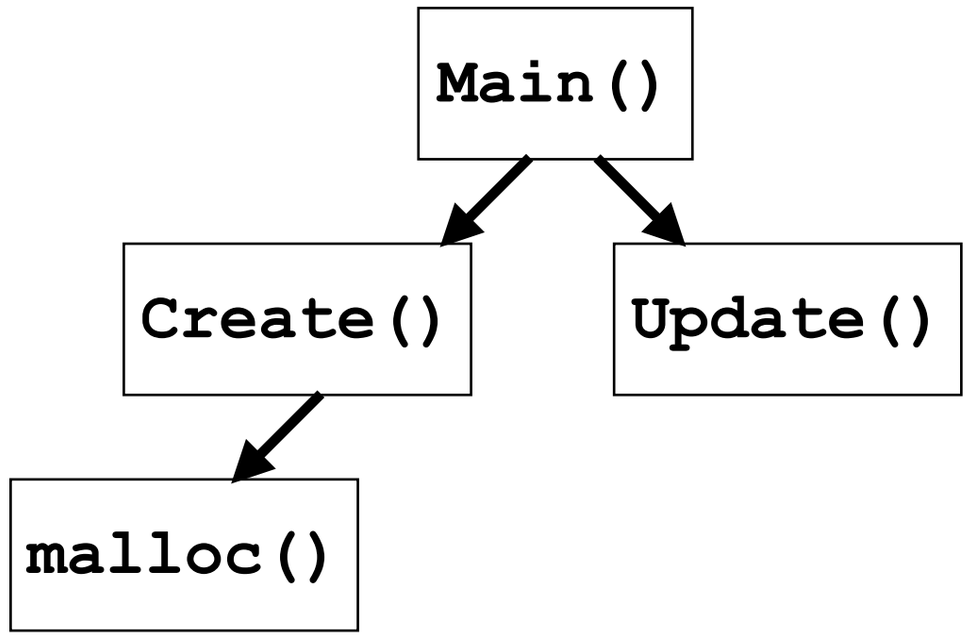
Associating Address With Data Objects

```
int MyArray[SZ];

int * Create(){
   return malloc(…);
}

void Update(int * ptr) {
   for( … )
      ptr[i]++;
}

int main(){
   int * p;
   if (…)
      p = Create();
   else
      p = MyArray;
   Update(p);
}
```



- Associate each *data access* to its *data object*
- Data object
  - ✦ Dynamic allocation: Call path of allocation site
  - ✦ Static objects: Variable name

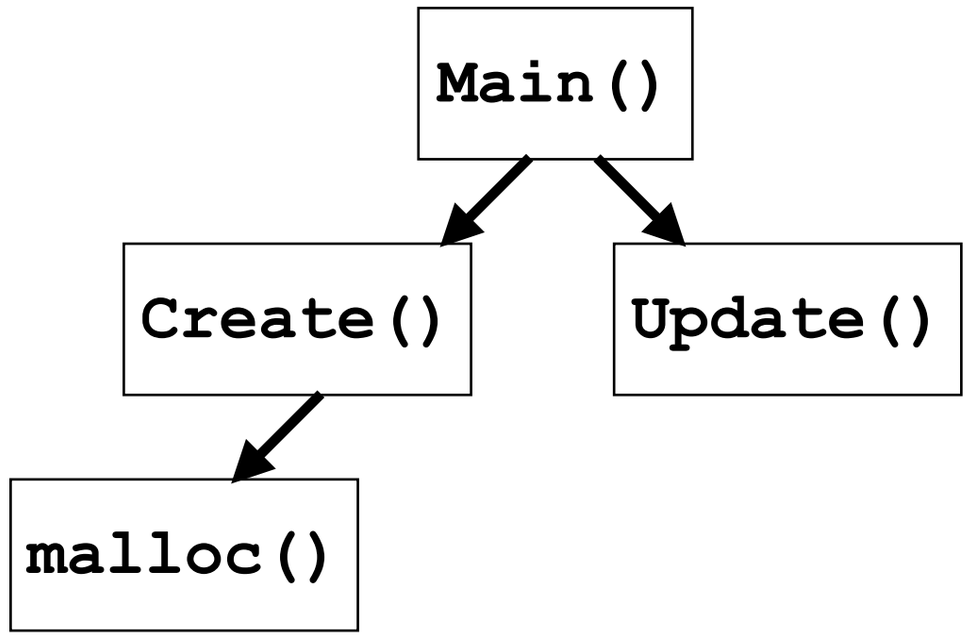
Associating Address With Data Objects

```
int MyArray[SZ];

int * Create(){
   return malloc(…);
}

void Update(int * ptr) {
   for( … )
      ptr[i]++;
}

int main(){
   int * p;
   if (…)
      p = Create();
   else
      p = MyArray;
   Update(p);
}
```



- Associate each *data access* to its *data object*
- Data object
  - ✦ Dynamic allocation: Call path of allocation site
  - ✦ Static objects: Variable name

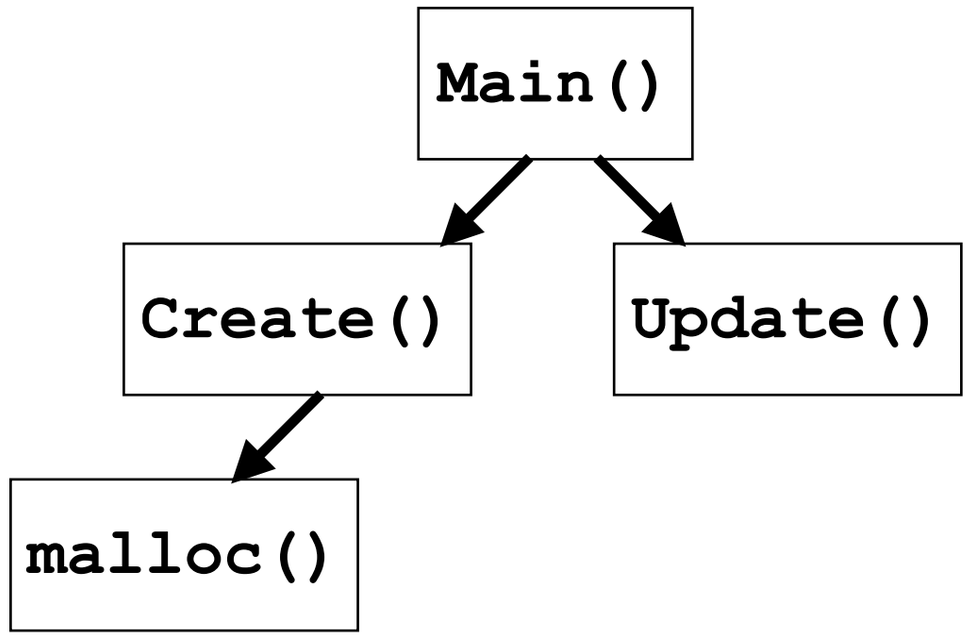
Associating Address With Data Objects

```
int MyArray[SZ];

int * Create(){
   return malloc(…);
}

void Update(int * ptr) {
   for( … )
      ptr[i]++;
}

int main(){
   int * p;
   if (…)
      p = Create();
   else
      p = MyArray;
   Update(p);
}
```



- Associate each *data access* to its *data object*
- Data object
  - ✦ Dynamic allocation: Call path of allocation site
  - ✦ Static objects: Variable name

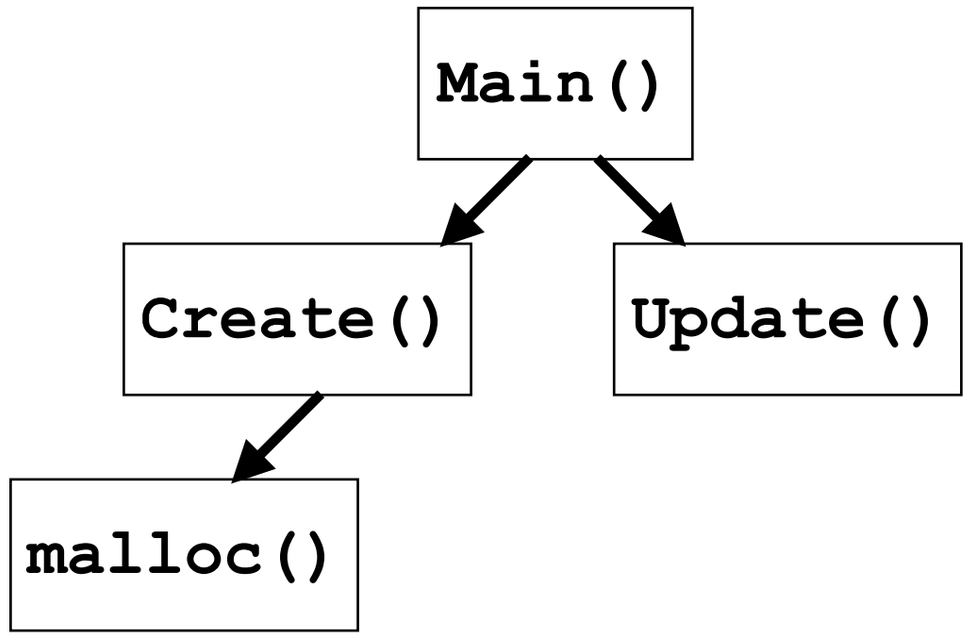
Associating Address With Data Objects

```
int MyArray[SZ];

int * Create(){
   return malloc(…);
}

void Update(int * ptr) {
   for( … )
      ptr[i]++;
}

int main(){
   int * p;
   if (…)
      p = Create();
   else
      p = MyArray;
   Update(p);
}
```



- Associate each *data access* to its *data object*
- Data object
  - ✦ Dynamic allocation: Call path of allocation site
  - ✦ Static objects: Variable name

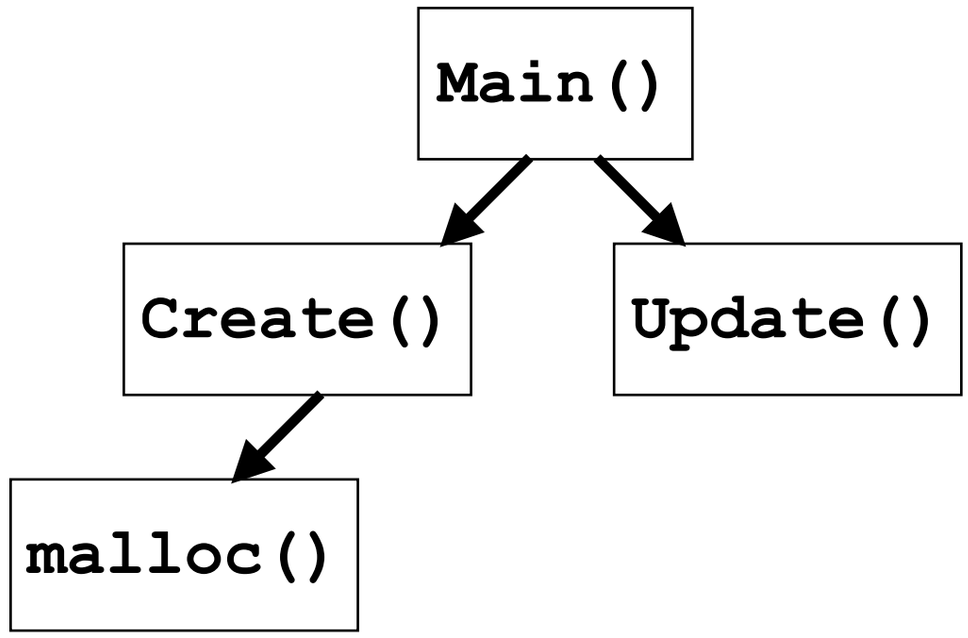
Associating Address With Data Objects

```
int MyArray[SZ];

int * Create(){
   return malloc(…);
}

void Update(int * ptr) {
   for( … )
      ptr[i]++;
}

int main(){
   int * p;
   if (…)
      p = Create();
   else
      p = MyArray;
   Update(p);
}
```



- Associate each *data access* to its *data object*
- Data object
  - ✦ Dynamic allocation: Call path of allocation site
  - ✦ Static objects: Variable name

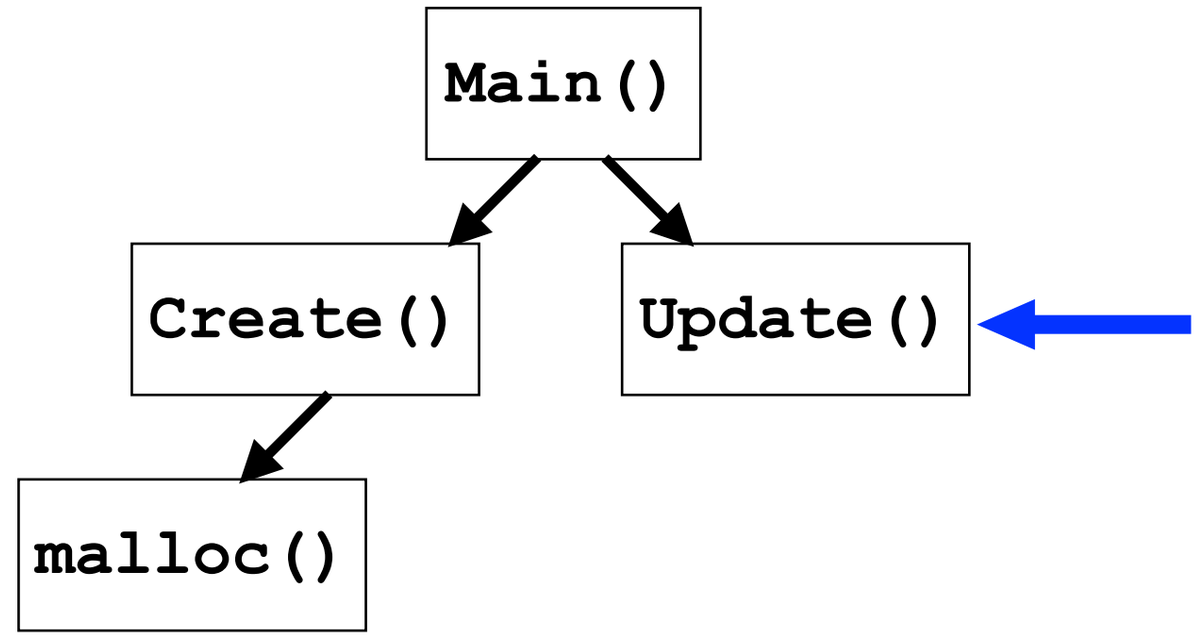
Associating Address With Data Objects

```
int MyArray[SZ];

int * Create(){
   return malloc(…);
}

void Update(int * ptr) {
   for( … )
      ptr[i]++;
}

int main(){
   int * p;
   if (…)
      p = Create();
   else
      p = MyArray;
   Update(p);
}
```



- Associate each *data access* to its *data object*
- Data object
  - ✦ Dynamic allocation: Call path of allocation site
  - ✦ Static objects: Variable name

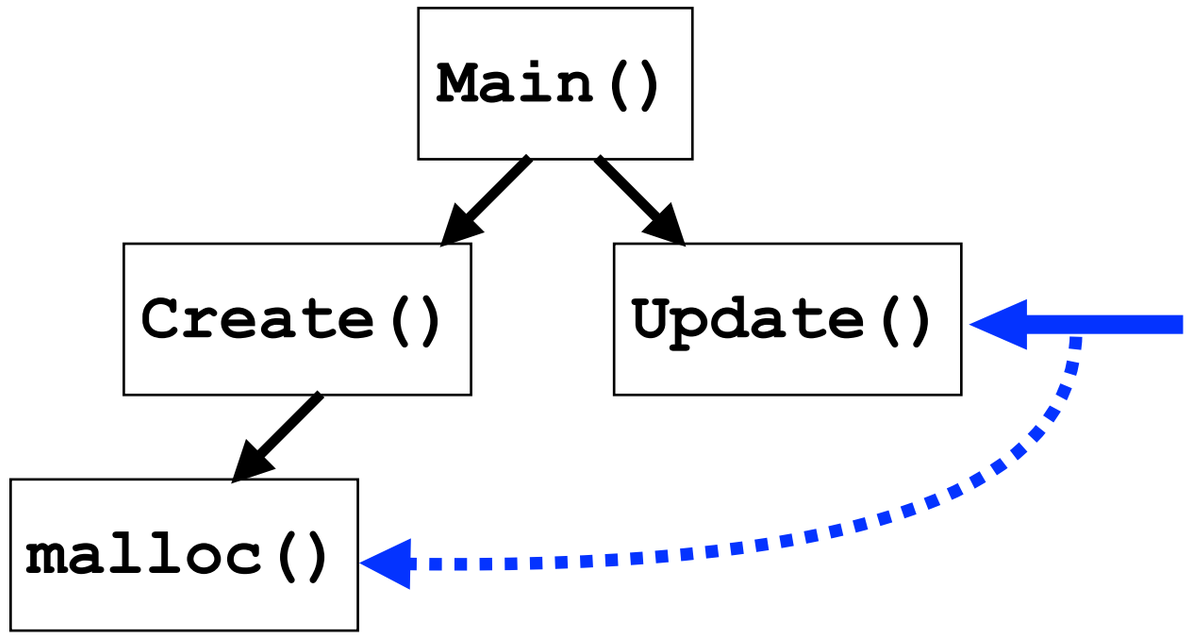
Associating Address With Data Objects

```
int MyArray[SZ];

int * Create(){
   return malloc(…);
}

void Update(int * ptr) {
   for( … )
      ptr[i]++;
}

int main(){
   int * p;
   if (…)
      p = Create();
   else
      p = MyArray;
   Update(p);
}
```



- Associate each *data access* to its *data object*
- Data object
  - ✦ Dynamic allocation: Call path of allocation site
  - ✦ Static objects: Variable name

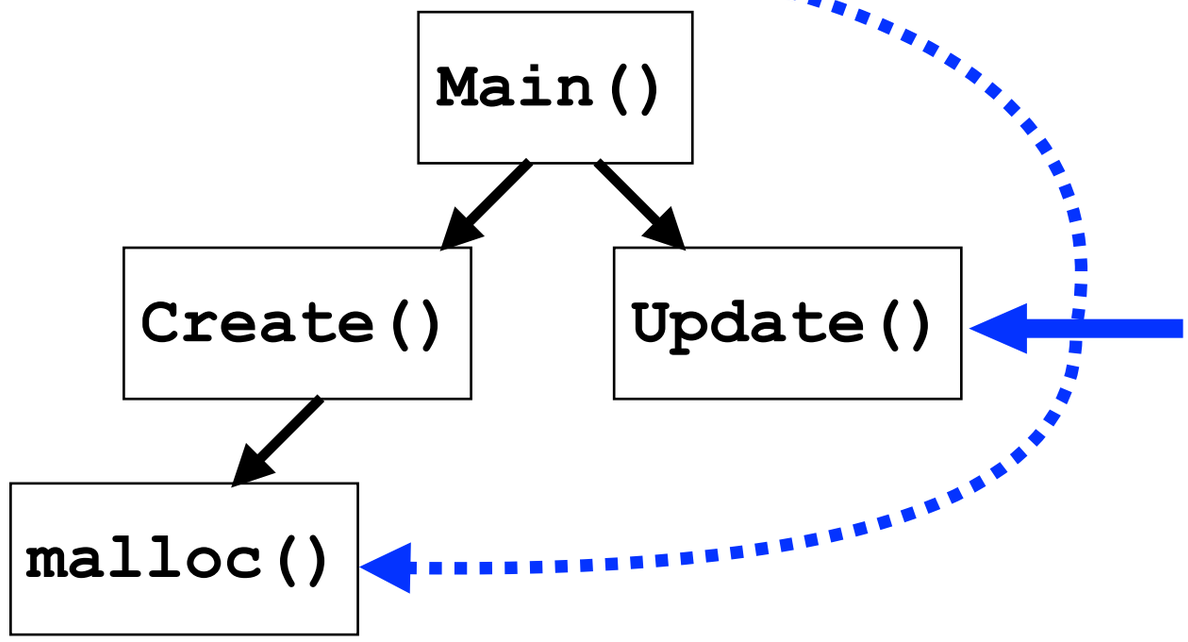
Associating Address With Data Objects

```
int MyArray[SZ];

int * Create(){
   return malloc(…);
}

void Update(int * ptr) {
   for( … )
      ptr[i]++;
}

int main(){
   int * p;
   if (…)
      p = Create();
   else
      p = MyArray;
   Update(p);
}
```



- Associate each *data access* to its *data object*
- Data object
  - ✦ Dynamic allocation: Call path of allocation site
  - ✦ Static objects: Variable name

How to Associate Address with Data Objects

- Static objects
  - ✦ Record all `<AddressRange, VariableName>` tuples in a map
- Dynamic allocations
  - ✦ Instrument all allocation/free routines
  - ✦ Maintain `<AddressRange, ContextId>` tuples in the map
- At each memory access: search the map for the address
- Problems
  - ✦ Searching the map on each access is expensive
  - ✦ Map needs to be concurrent for threaded programs

Address→Object Mapping via Shadow Memory

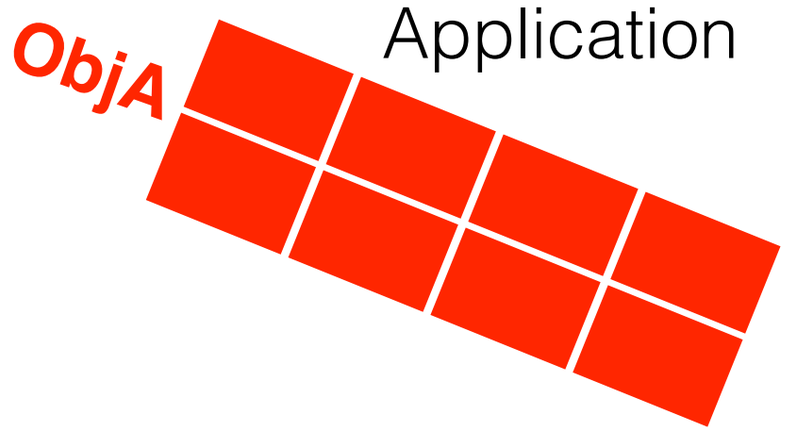
Solution: shadow memory

Application

DrCCTProf

Address→Object Mapping via Shadow Memory

Solution: shadow memory



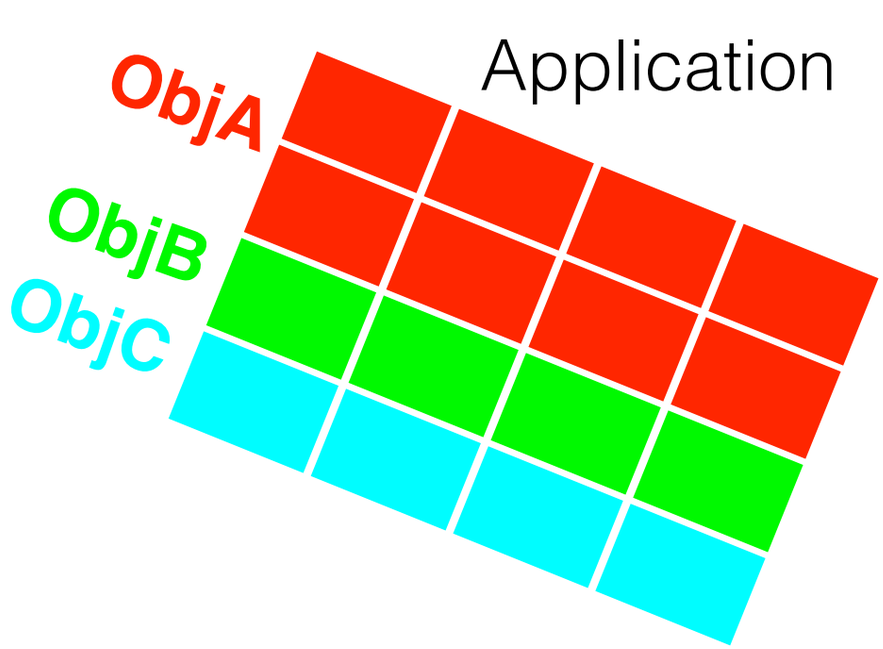
DrCCTProf

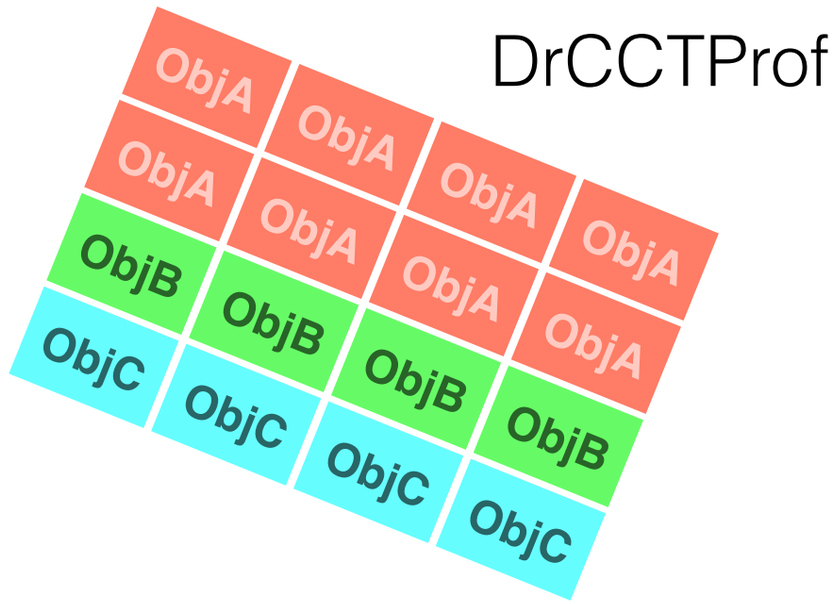
Address→Object Mapping via Shadow Memory

Solution: shadow memory

Address→Object Mapping via Shadow Memory

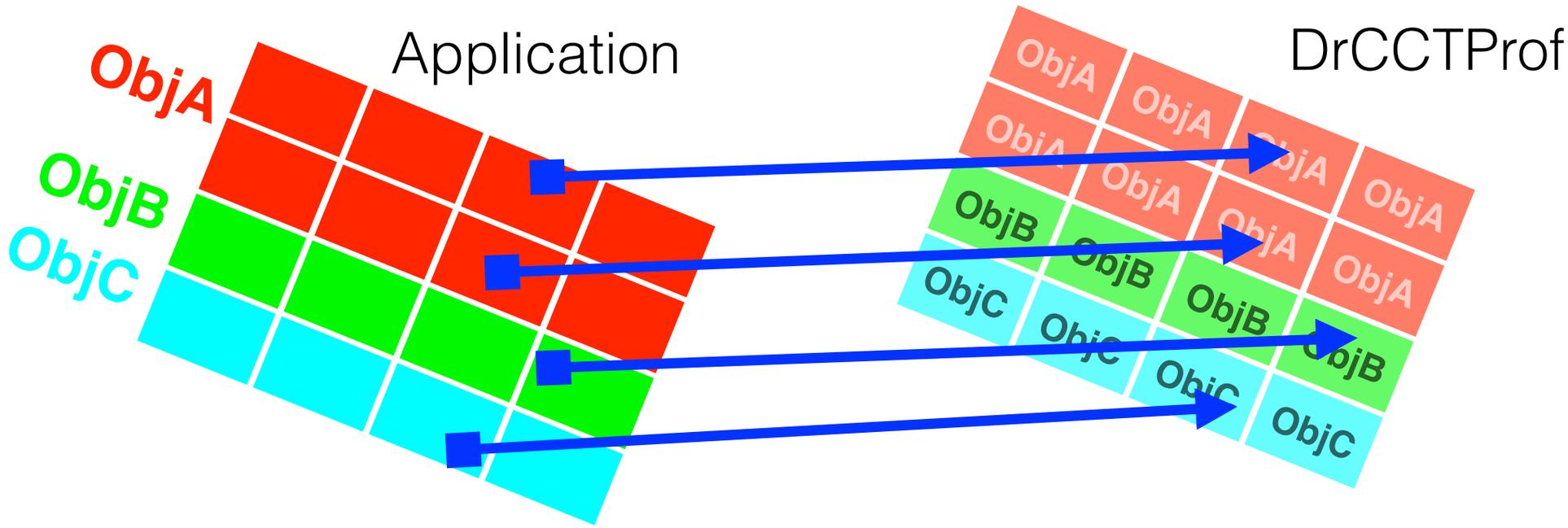
Solution: shadow memory





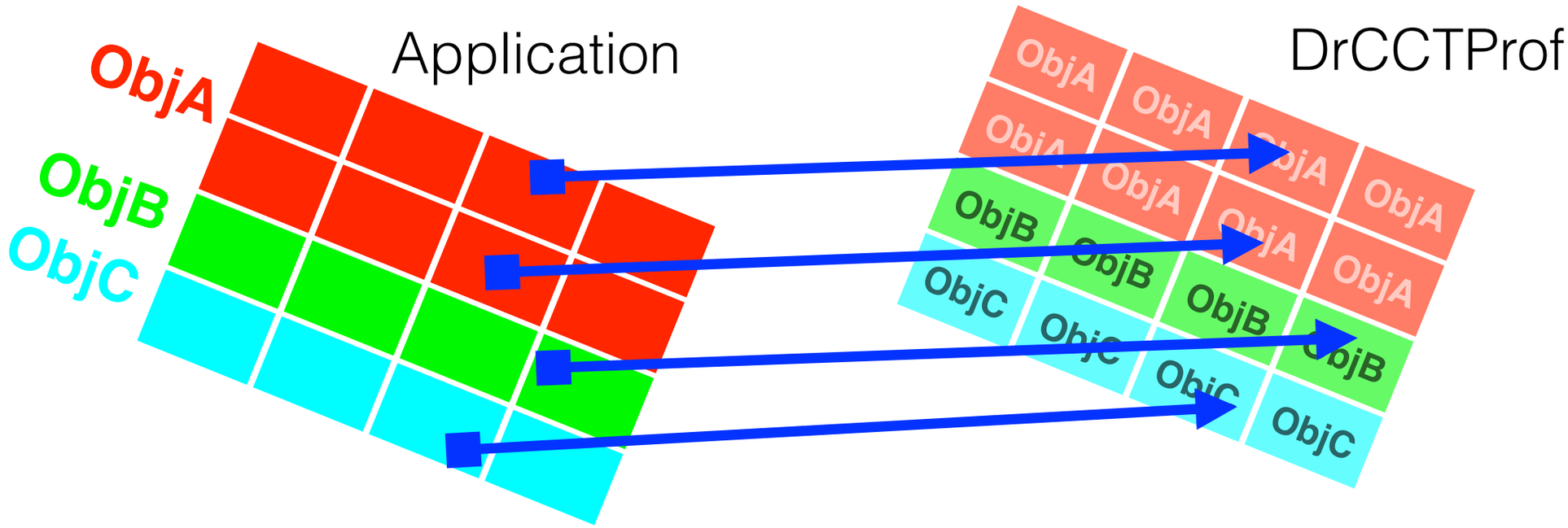
Address→Object Mapping via Shadow Memory

Solution: shadow memory



Address→Object Mapping via Shadow Memory

Solution: shadow memory



- For each memory cell, a shadow cell holds a handle for the memory cell's data object $O(\sum_{i=1}^{N} sizeof(Obj(i)))$
  - ✦ Low lookup cost—O(1), high memory cost—
  - ✦ Shadow memory supports concurrent access

Try DrCCTProf and Build Your Own Clients

- A number client tools available atop DrCCTProf
  - Check the example client tools at DrCCTProf/src/clients
- You need to develop your own client tools to understand compiler code generation
  - Project 0: learn how to use DrCCTProf
    - Learn and write a simple but useful client
    - Understand compiler code generation

Data Visualization

- Standard
  - ✦ Texts output: write human readable texts into a file
- Advanced
  - ✦ GUI-based visualization
    - ✳ Using HPCToolkit format
    - ✳ Using VSCode format

- Standar
  - Texts c a file
- Advance
  - HPCTo
    - Visua

test_app_cct.cxx

```
21 }
22
23 void t1_fun() {
24     for(int i = 0; i < 2222; i++){
25         t1_sub_fun();
26     }
27 }
28
29 void t2_fun() {
30     for(int i = 0; i < 1111; i++){
31         t2_sub_fun();
32     }
33 }
34 #endif
35
36 #ifdef MULTITHREADING
37 void *thread_1(void *arg)
```

Top-down view | Bottom-up view | Flat view

Scope	INS_COUNT
▶ <unknown procedure> 0x1093 [ld-2.27.so]	46276
▼ _start (test_app_cct)	43329
▼ <unknown procedure> 0x21b95 [libc-2.27.so]	43329
▼ main (test_app_cct)	43329
▼ 82: t1_fun (test_app_cct)	28886
▼ loop at test_app_cct.cxx: 24	28886
▶ 25: t1_sub_fun (test_app_cct)	17776
test_app_cct.cxx: 24	8888
test_app_cct.cxx: 25	2222
▼ 83: t2_fun (test_app_cct)	14443
▼ loop at test_app_cct.cxx: 30	14443
▶ 31: t2_sub_fun (test_app_cct)	8888
test_app_cct.cxx: 30	4444
test_app_cct.cxx: 31	1111