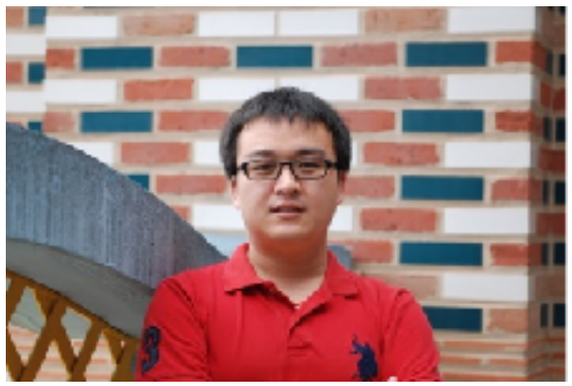
CCTLib: Pinpointing Software Inefficiencies with Fine-grained Program Monitoring

Milind Chabbi

Shasha Wen

Xu Liu

Who Are We?



Xu Liu
Assistant Professor
Department of Computer Science
College of William and Mary
xl10@cs.wm.edu

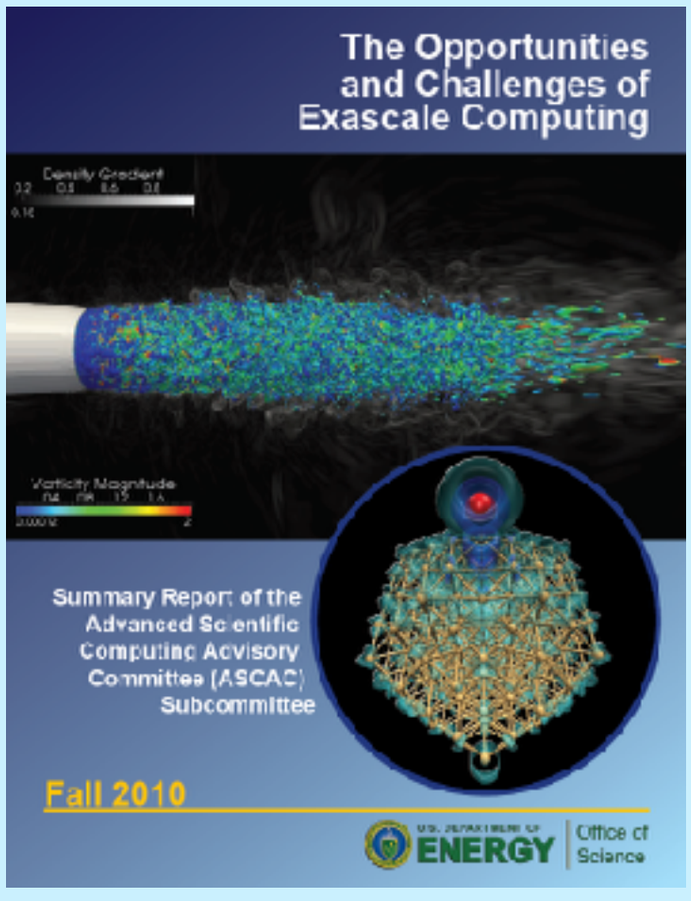
Shasha Wen
Ph.D. student
Department of Computer Science
College of William and Mary
swen@email.wm.edu

Milind Chabbi
Research Scientist
Hewlett Packard Labs
Palo Alto, CA
milind.chabbi@hpe.com

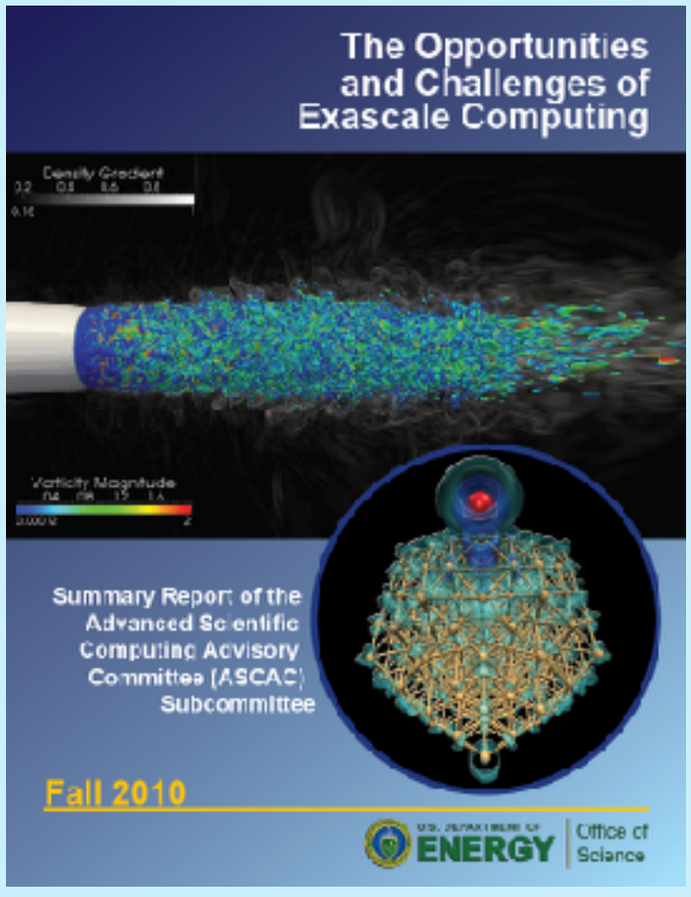
Introduction

Importance of Code Efficiency

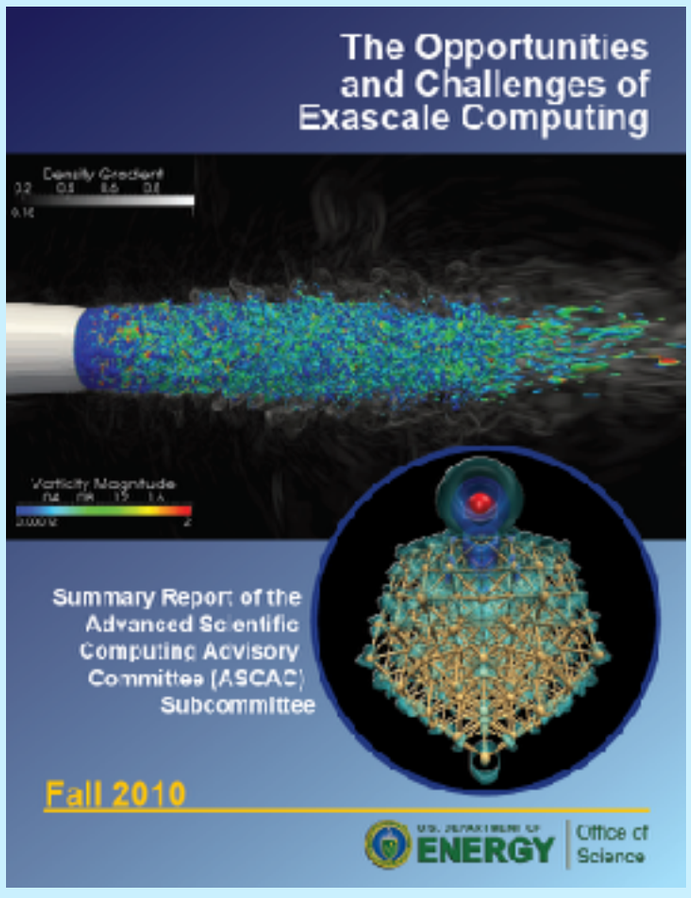
Importance of Code Efficiency

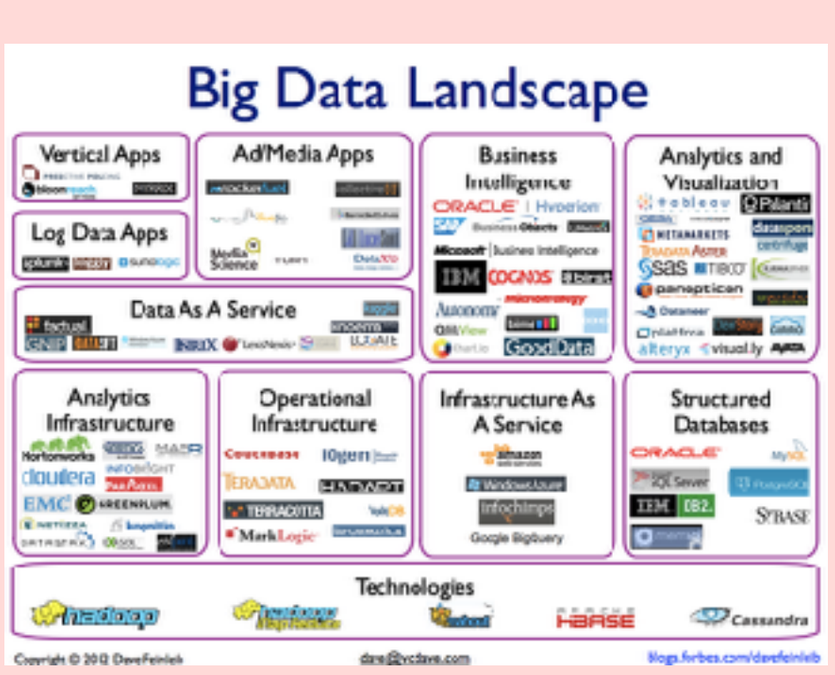


Importance of Code Efficiency

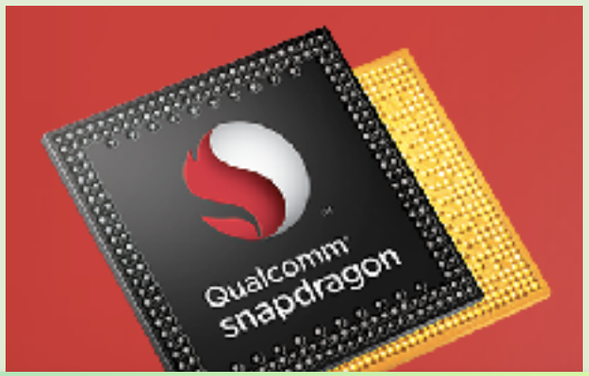


Importance of Code Efficiency

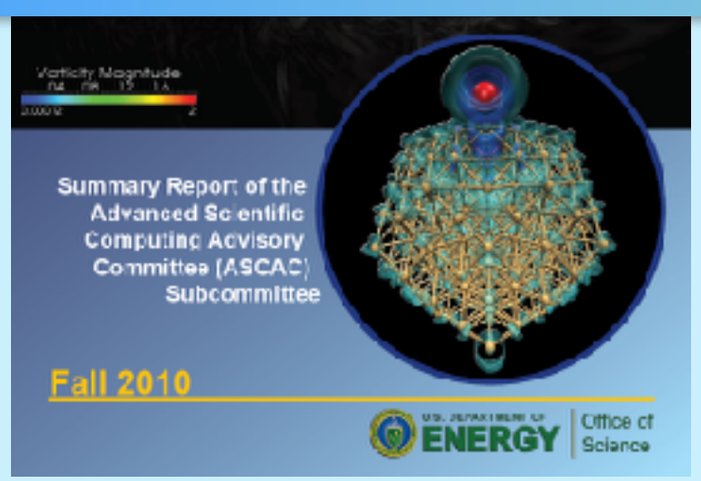




Importance of Code Efficiency



Programs need to be efficient at all scales



Sources of Performance Bottlenecks

- Code design
  - Algorithms
  - Data structures
- Programming practice
  - Aware of functionality but not performance
- Compiler optimization
  - Sometimes optimization may cause more harm than good
  - Code must be tailored to enable some optimization

Sources of Performance Bottlenecks

- Code design
  - Algorithms
  - Data structures

- Programming practice
  - Aware of functionality but not performance

- Compiler optimization
  - Sometimes optimization may cause more harm than good
  - Code must be tailored to enable some optimization

A tool set is necessary to pinpoint inefficiencies

Classical Performance Analysis

- Identify hot spots — high resource utilization
  - Time / CPU cycles
  - Cache misses on different levels
  - Floating point operations, SIMD
  - Derived metrics such as instruction per cycle (IPC)
- Improve code in hot spots
- Hot spot analysis is indispensable, but
  - Cannot tell if resources were **“well spent”**
  - Hot spots may be symptoms of performance problems
  - Need significant manual efforts to investigate root causes

From Resource Usage to Wastage

- Wasted data movement
  - Redundant memory accesses
    - Redundant stores: write same values to a memory location
  - Useless memory accesses
    - Dead stores: stored value got overwritten without use

- Wasted arithmetic computation
  - Symbolic equivalent computation
    - a=b+c; d=b+c
  - Result equivalent computation
    - a=b*b-c*c; d=(b+c)*(b-c)

- Unnecessary synchronization (locks and barriers)

From Resource Usage to Wastage

- Wasted data movement
  - Redundant memory accesses
    - Redundant stores: write same values to a memory location
  - Useless memory accesses
    - Dead stores: stored value got overwritten without use

- Wasted arithmetic computation
  - Symbolic equivalent computation
    - a=b+c; d=b+c
  - Result equivalent computation
    - a=b*b-c*c; d=(b+c)*(b-c)

- Unnecessary synchronization (locks and barriers)

Need new profiling techniques
fine-grained profiling

HMMER: A Example for Resource Wastage

Unoptimized

```
for (i = 1; i <= L; i++) {
  for (k = 1; k <= M; k++) {
      mc[k] = mpp[k-1] + tpmm[k-1];
if ((sc = ip[k-1] + tpim[k-1]) > mc[k])
      mc[k] = sc;
```

-O3 optimized

```
for (i = 1; i <= L; i++) {
    for (k = 1; k <= M; k++) {
        R1= mpp[k-1] + tpmm[k-1];
        mc[k] = R1;
if ((sc = ip[k-1] + tpim[k-1]) > R1)
      mc[k] = sc;
```

HMMER: A Example for Resource Wastage

Unoptimized

```
for (i = 1; i <= L; i++) {
  for (k = 1; k <= M; k++) {
      mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k])
      mc[k] = sc;
```

-O3 optimized

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
       R1= mpp[k-1] + tpmm[k-1];
       mc[k] = R1;
   if ((sc = ip[k-1] + tpim[k-1]) > R1)
       mc[k] = sc;
   else
       mc[k] = R1;
```

HMMER: A Example for Resource Wastage

Unoptimized

```
for (i = 1; i <= L; i++) {
  for (k = 1; k <= M; k++) {
      mc[k] = mpp[k-1] + tpmm[k-1];
if ((sc = ip[k-1] + tpim[k-1]) > mc[k])
      mc[k] = sc;
```

-O3 optimized

```
for (i = 1; i <= L; i++) {
    for (k = 1; k <= M; k++) {
        R1= mpp[k-1] + tpmm[k-1];
        mc[k] = R1;
if ((sc = ip[k-1] + tpim[k-1]) > R1)
      mc[k] = sc;
else
      mc[k] = R1;
```

Never Alias.
Declare as “restrict” pointers.
Can vectorize.

HMMER: A Example for Resource Wastage

Unoptimized

```
for (i = 1; i <= L; i++) {
  for (k = 1; k <= M; k++) {
      mc[k] = mpp[k-1] + tpmm[k-1];
if ((sc = ip[k-1] + tpim[k-1]) > mc[k])
      mc[k] = sc;
```

-O3 optimized

```
for (i = 1; i <= L; i++) {
    for (k = 1; k <= M; k++) {
        R1= mpp[k-1] + tpmm[k-1];
        mc[k] = R1;
if ((sc = ip[k-1] + tpim[k-1]) > R1)
        mc[k] = sc;
else
        mc[k] = R1;
```

Never Alias.
Declare as “restrict” pointers.
Can vectorize.

> 16% running time improvement
> 40% with vectorization

Compilers Do NOT Eliminate All Inefficiencies

- Compilers have limitations with their static analysis
  - Aliasing and pointers
  - Limited optimization scopes: compilation units
  - Input-sensitive inefficiencies
  - Flow-sensitive inefficiencies

Coarse-grained Profilers Lack

- State-of-the-art coarse-grained profilers
  - Intel VTune
  - Rice HPCToolkit
  - Oracle Solaris Studio
  - ARM allinea

- Coarse-grained analysis
  - Sample instructions or events via hardware performance monitoring units (PMU)
    - One sample per 1M instructions
  - Do not track consecutive sequence of instructions or memory references —> cannot detect wasteful operations
  - Never capture semantic meaning of execution

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
- One step closer to reconstructing the semantic meaning (or lack there of) in execution

HMMER Example

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      R1= mpp[k-1] + tpmm[k-1];
     mc[k] = R1;
     if ((sc = ip[k-1] + tpim[k-1]) > R1)
        mc[k] = sc;
```

HMMER Example

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      R1= mpp[k-1] + tpmm[k-1];
     mc[k] = R1;
     if ((sc = ip[k-1] + tpim[k-1]) > R1)
        mc[k] = sc;
```

```
1  mov %r10,%rax,4),%ecx
2  add 0x0(%r13,%rax,4),%ecx #mpp[k-1]+tpmm[k-1]
3  mov %ecx, 0x4(%rdx)          #assign mc[k]
4  mov 0x18(%rsp),%rbx
5  mov (%r9,%rax,4),%r15d
6  add (%rbx,%rax,4),%r15d    #dpp[k-1]+tpdm[k-1]
7  mov   0x20(%rsp),%rbx
8  cmp   %ecx,%r15d           #%ecx is mc[k]
9  cmovge %r15d, %ecx
10 mov %ecx, 0x4(%rdx)        #assign mc[k]
```

HMMER Example

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      R1= mpp[k-1] + tpmm[k-1];
      mc[k] = R1;
      if ((sc = ip[k-1] + tpim[k-1]) > R1)
         mc[k] = sc;
```

```
1   mov %r10,%rax,4),%ecx
2   add 0x0(%r13,%rax,4),%ecx #mpp[k-1]+tpmm[k-1]
3   mov %ecx, 0x4(%rdx)          #assign mc[k]
4   mov 0x18(%rsp),%rbx
5   mov (%r9,%rax,4),%r15d
6   add (%rbx,%rax,4),%r15d    #dpp[k-1]+tpdm[k-1]
7   mov   0x20(%rsp),%rbx
8   cmp   %ecx,%r15d           #%ecx is mc[k]
9   cmovge %r15d, %ecx
10 mov %ecx, 0x4(%rdx)         #assign mc[k]
```

HMMER Example

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      R1= mpp[k-1] + tpmm[k-1];
     mc[k] = R1;
     if ((sc = ip[k-1] + tpim[k-1]) > R1)
        mc[k] = sc;
```

```
1   mov %r10,%rax,4),%ecx
2   add 0x0(%r13,%rax,4),%ecx #mpp[k-1]+tpmm[k-1]
3   mov %ecx, 0x4(%rdx)          #assign mc[k]
4   mov 0x18(%rsp),%rbx
5   mov (%r9,%rax,4),%r15d
6   add (%rbx,%rax,4),%r15d    #dpp[k-1]+tpdm[k-1]
7   mov   0x20(%rsp),%rbx
8   cmp   %ecx,%r15d           #%ecx is mc[k]
9   cmovge %r15d, %ecx
10 mov %ecx, 0x4(%rdx)         #assign mc[k]
```

HMMER Example

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      R1= mpp[k-1] + tpmm[k-1];
      mc[k] = R1;
      if ((sc = ip[k-1] + tpim[k-1]) > R1)
         mc[k] = sc;
```

```
1   mov %r10,%rax,4),%ecx
2   add 0x0(%r13,%rax,4),%ecx #mpp[k-1]+tpmm[k-1]
3   mov %ecx, 0x4(%rdx)          #assign mc[k]
```

the value in memory location 0x4(%rdx) is unused

```
10 mov %ecx, 0x4(%rdx)          #assign mc[k]
```

HMMER Example

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      R1= mpp[k-1] + tpmm[k-1];
      mc[k] = R1;
      if ((sc = ip[k-1] + tpim[k-1]) > R1)
         mc[k] = sc;
```

```
1   mov %r10,%rax,4),%ecx
2   add 0x0(%r13,%rax,4),%ecx #mpp[k-1]+tpmm[k-1]
3   mov %ecx, 0x4(%rdx)          #assign mc[k]
```

the value in memory location 0x4(%rdx) is unused

```
10 mov %ecx, 0x4(%rdx)          #assign mc[k]
```

dead store

Call Path Profiling for Fine-grained Analysis

- Associate problematic instructions with their call paths
  - ✦ Expose more semantic information about the instructions
  - ✦ Understand context-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
```

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

CCTLib Overview

- Functionality
  - ✦ Can capture call path for each dynamic instruction
  - ✦ Can 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

CCTLib Software

- git clone https://github.com/CCTLib/cctlib.git
  - Supported on x86_64 linux, gcc > 4.8.2
  - Pin 3.0 not yet supported.
- cd cctlib
- sh build.sh

```
PIN ROOT is NOT set!
+ echo   (1) Download Pin from the WWW and automatically set PIN_ROOT?
  (2) Enter PIN_ROOT in the commandline?
  (any key) Exit?
  (1) Download Pin from the WWW and automatically set PIN_ROOT?
  (2) Enter PIN_ROOT in the commandline?
  (any key) Exit?
```

- Choose (1)
- Successful installation will end with this message

```
**********************************************************
********** ALL TESTS PASSED **********
**********************************************************
```

CCTLib Software

- Distributed under MIT license

- Contact

  cctlib-forum@lists.wm.edu

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}**

Challenges in Ubiquitous Call Path Collection

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

Store History of Contexts Compactly

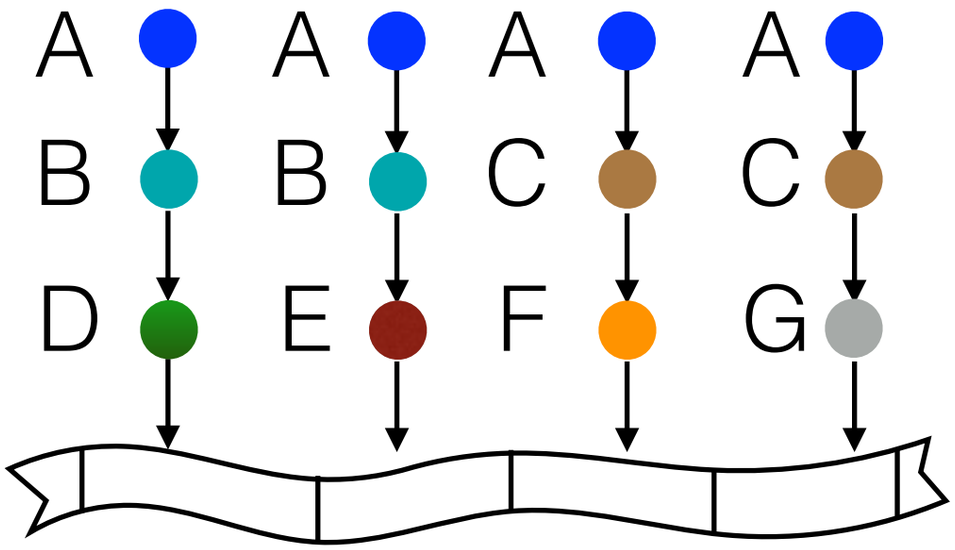
Problem:

Deluge of call paths

Store History of Contexts Compactly

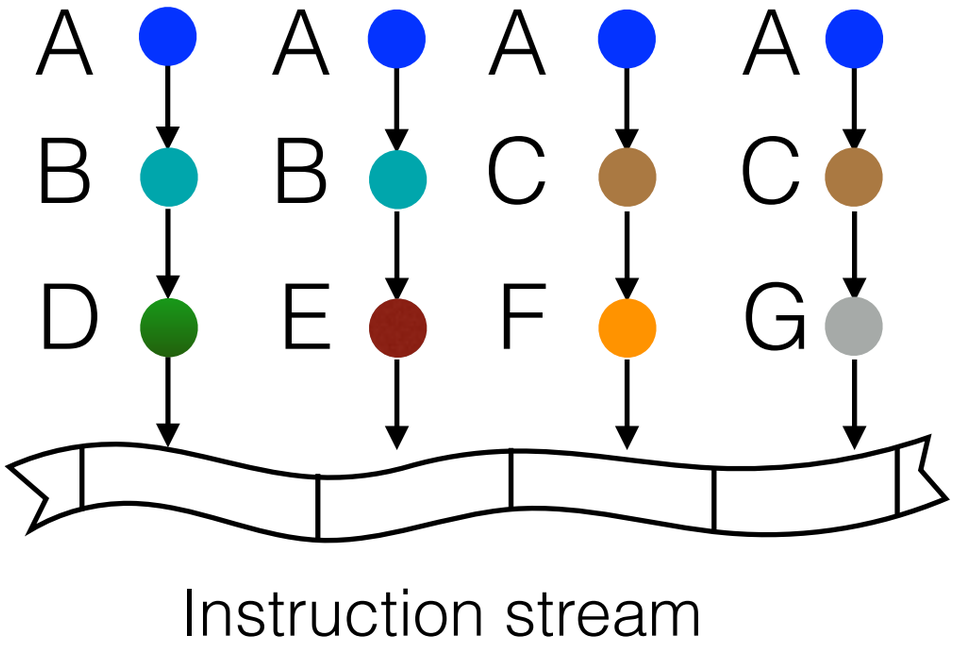
<u>Problem:</u>

Deluge of call paths



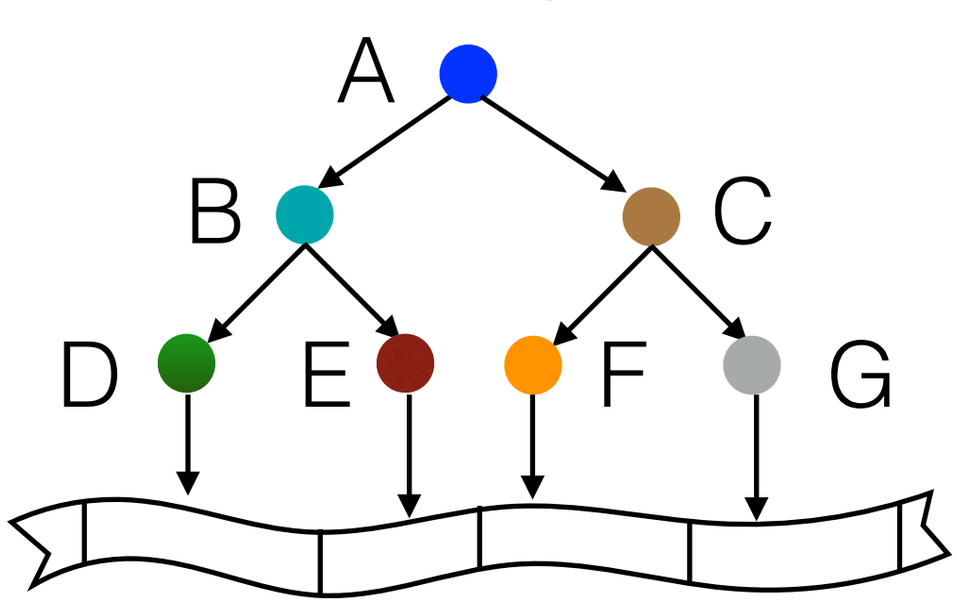
Store History of Contexts Compactly

Problem:
Deluge of call paths



Solution

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

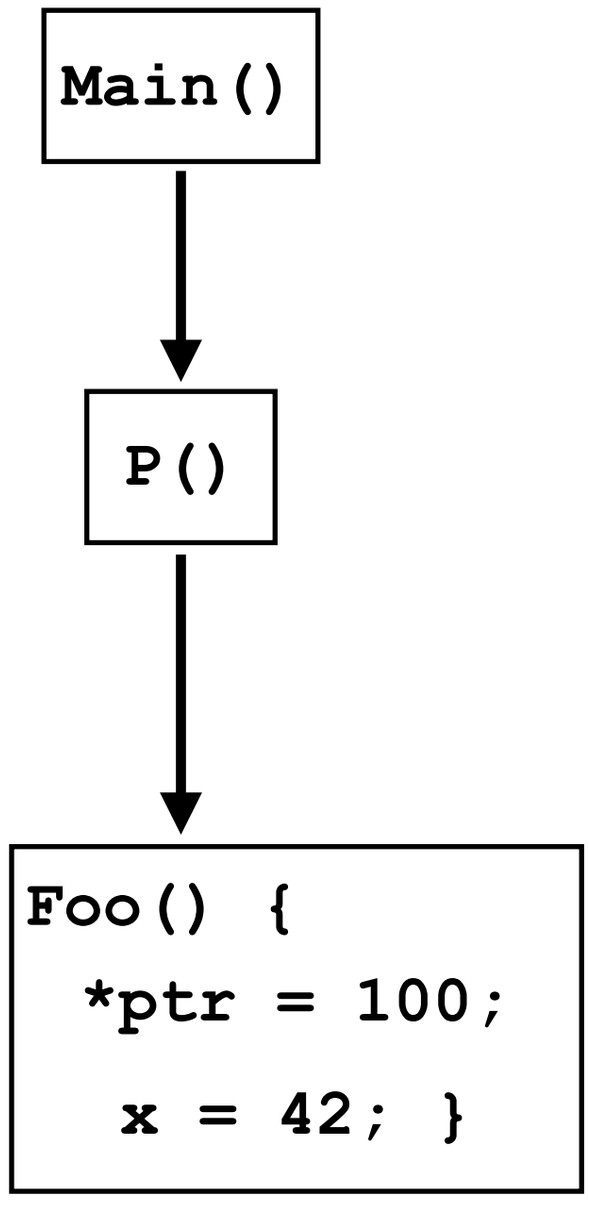


Exploiting Hardware Performance Counters with Flow and Context Sensitive Profiling: Ammons et al. PLDI'97

Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

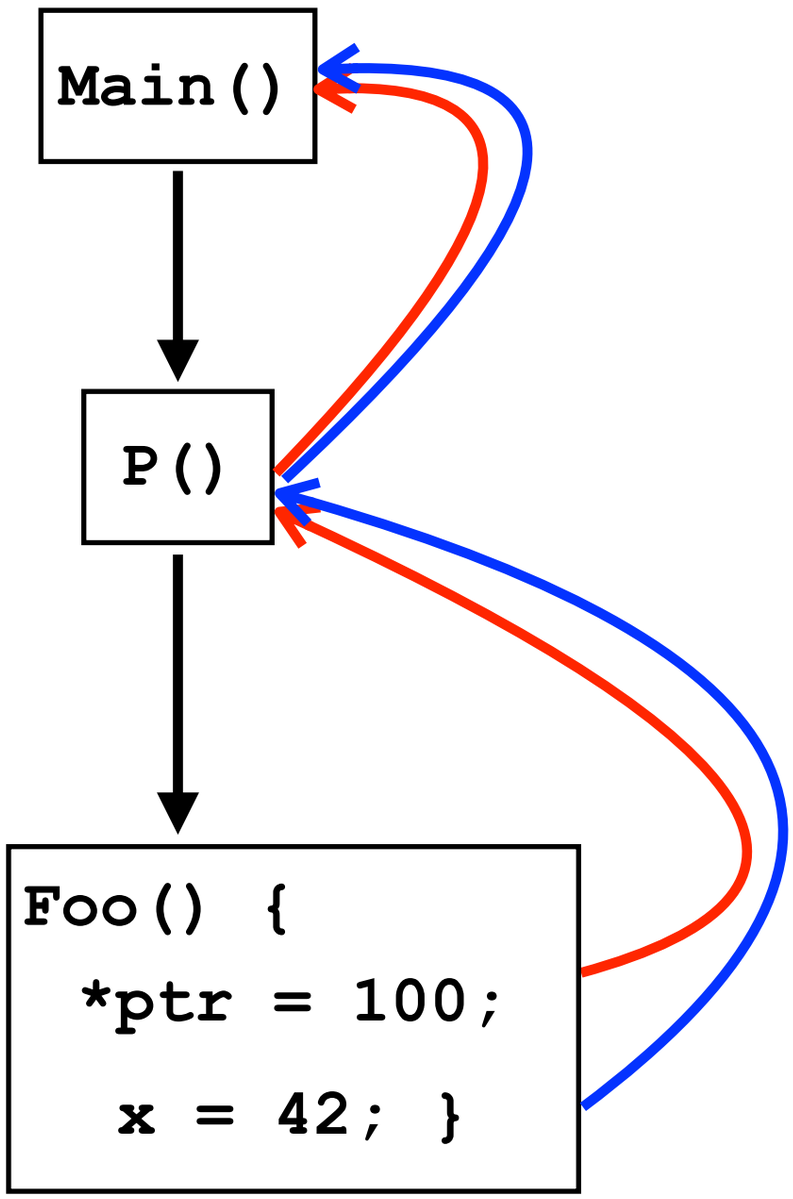
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.



Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

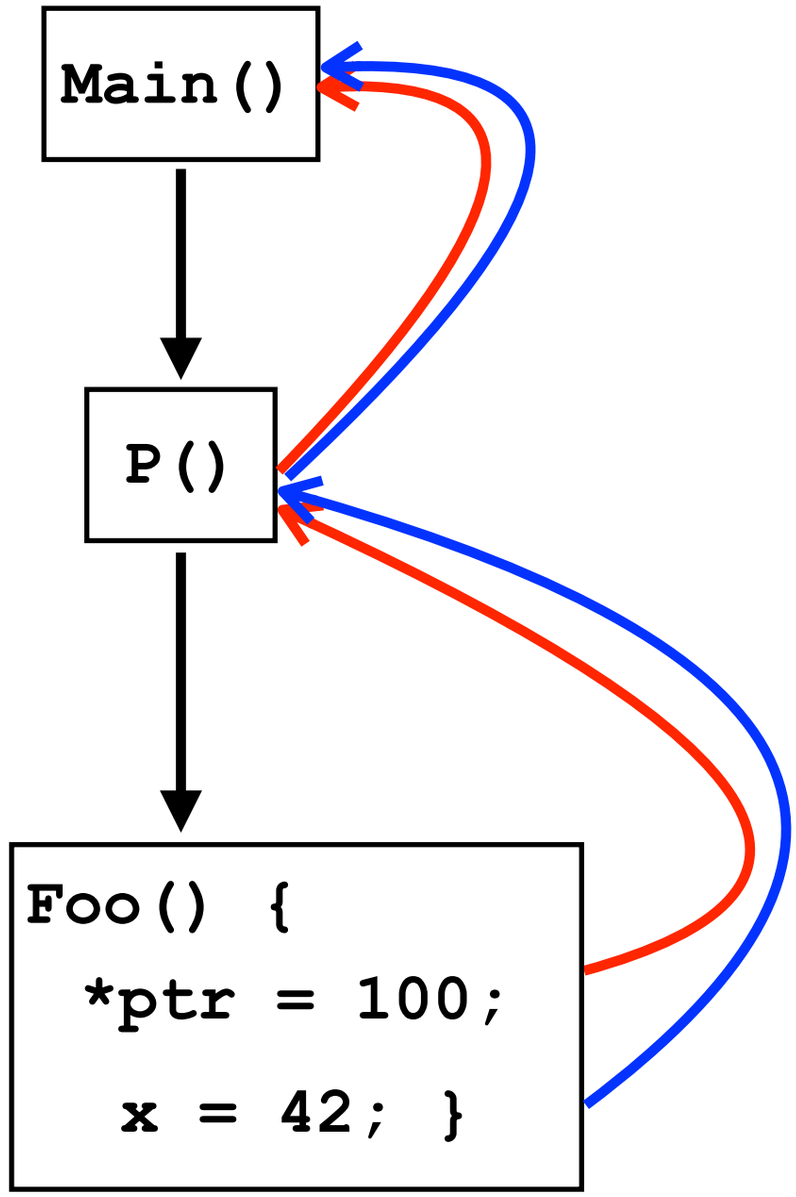
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.



Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

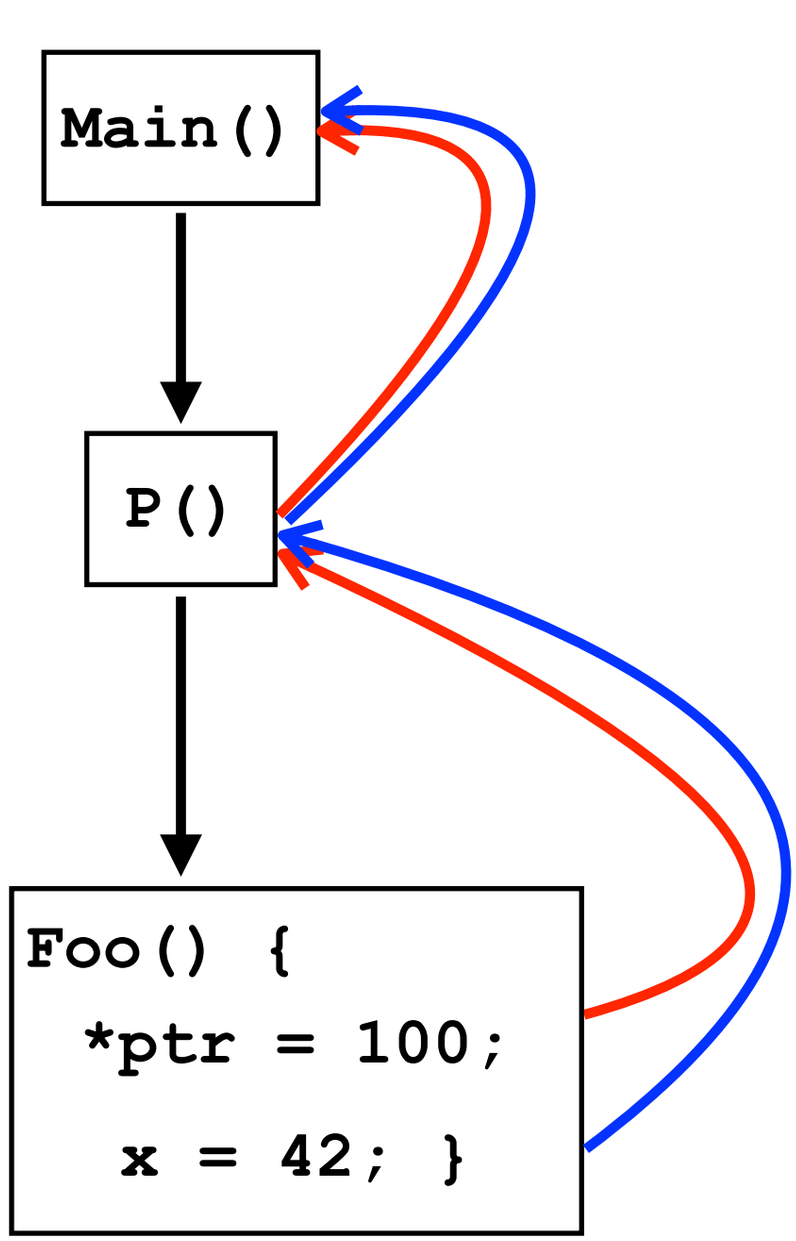
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

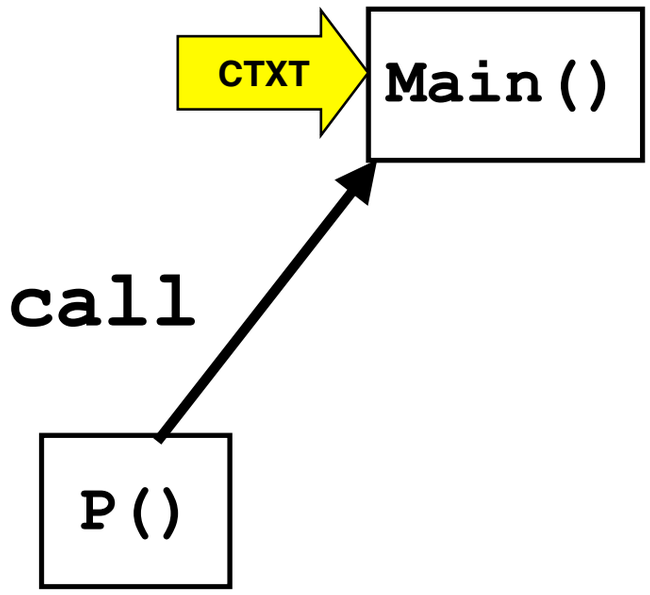


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

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

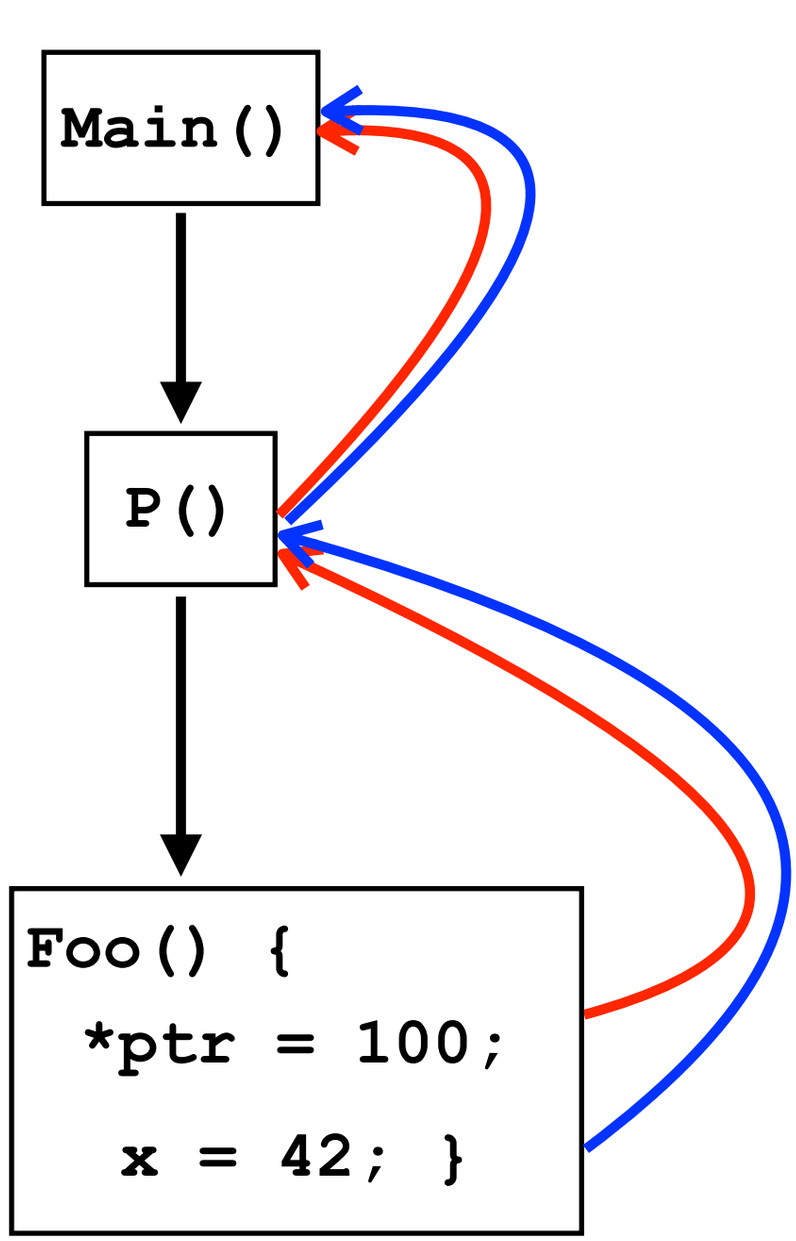


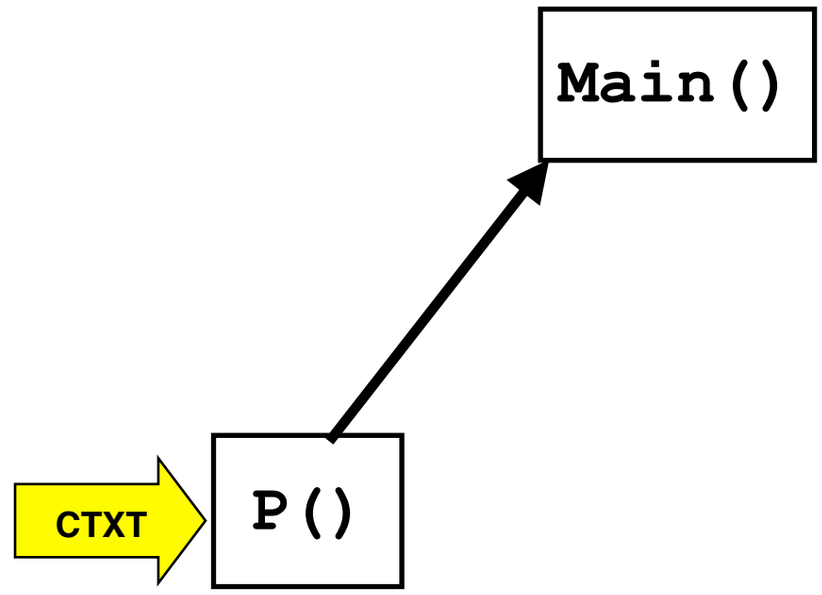


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

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

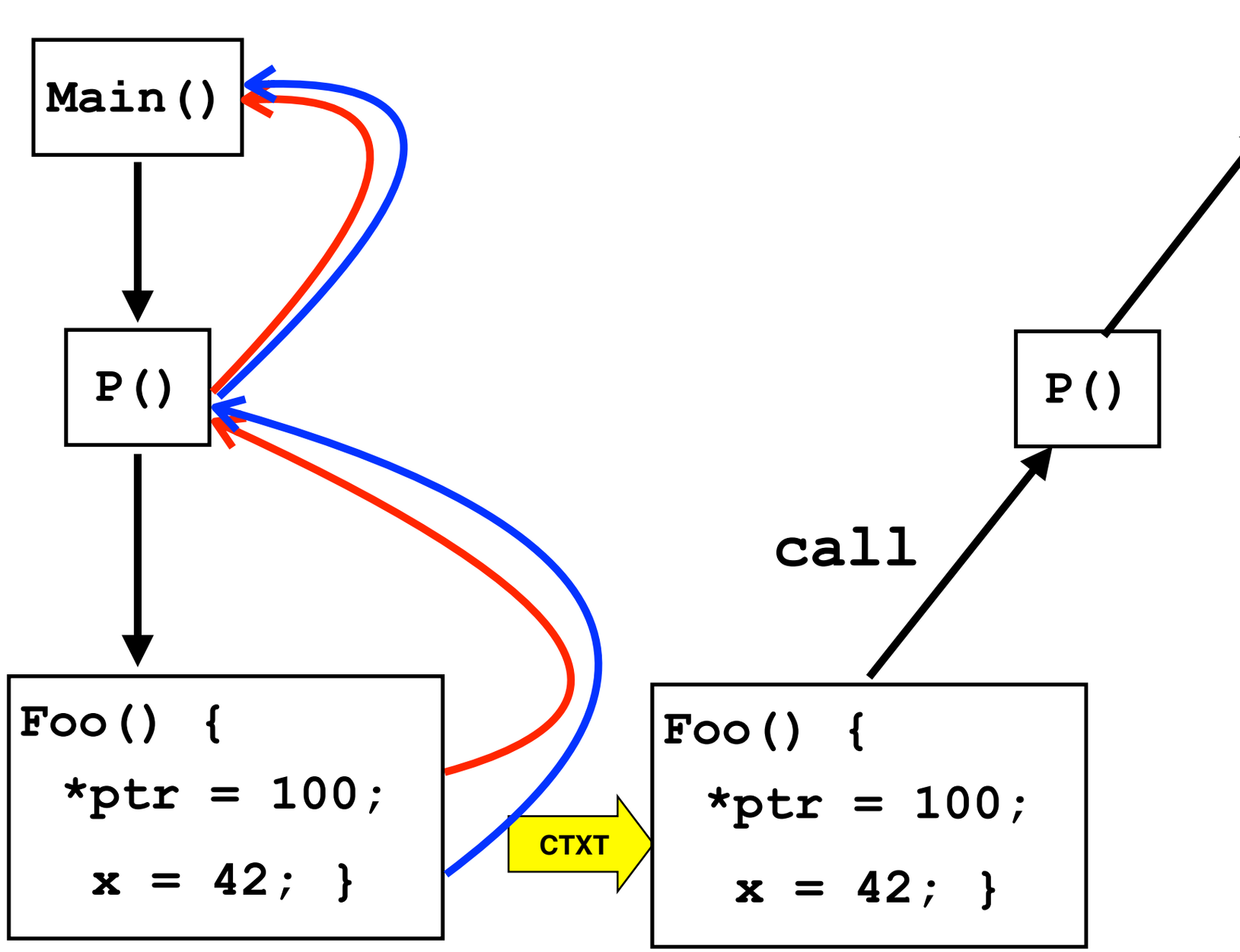




Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

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

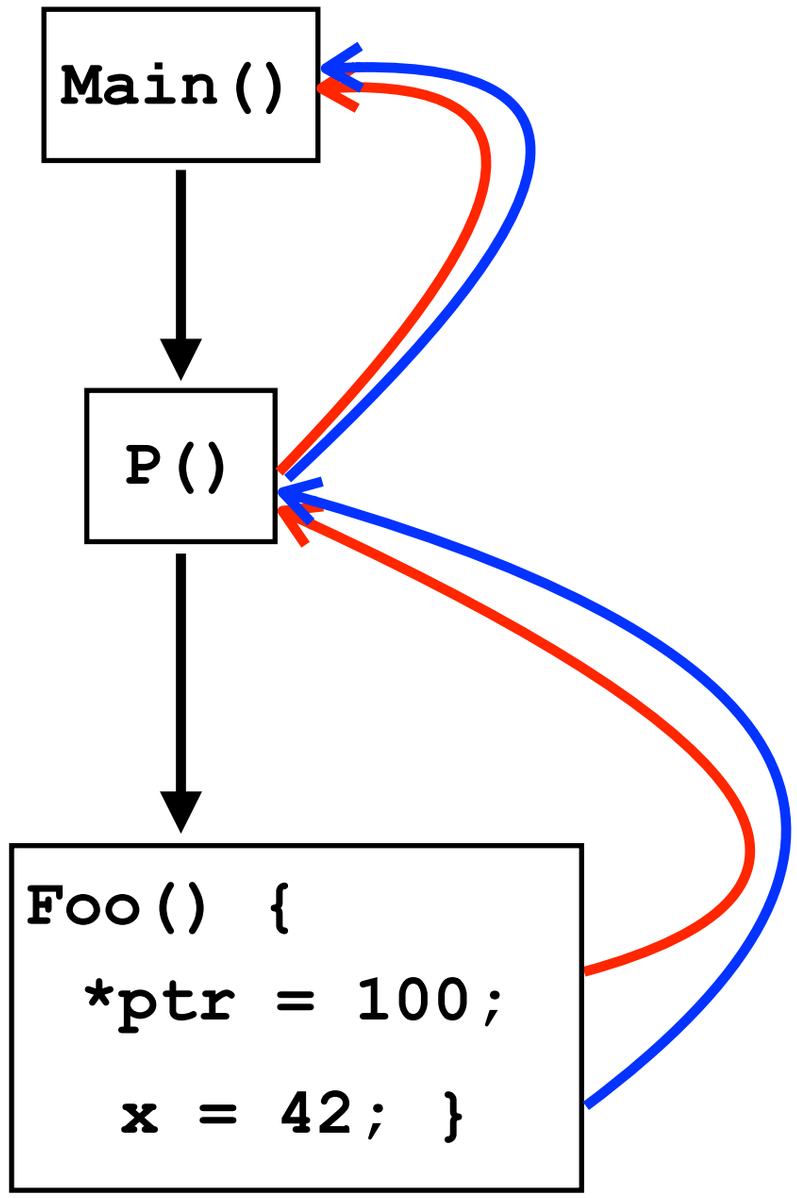


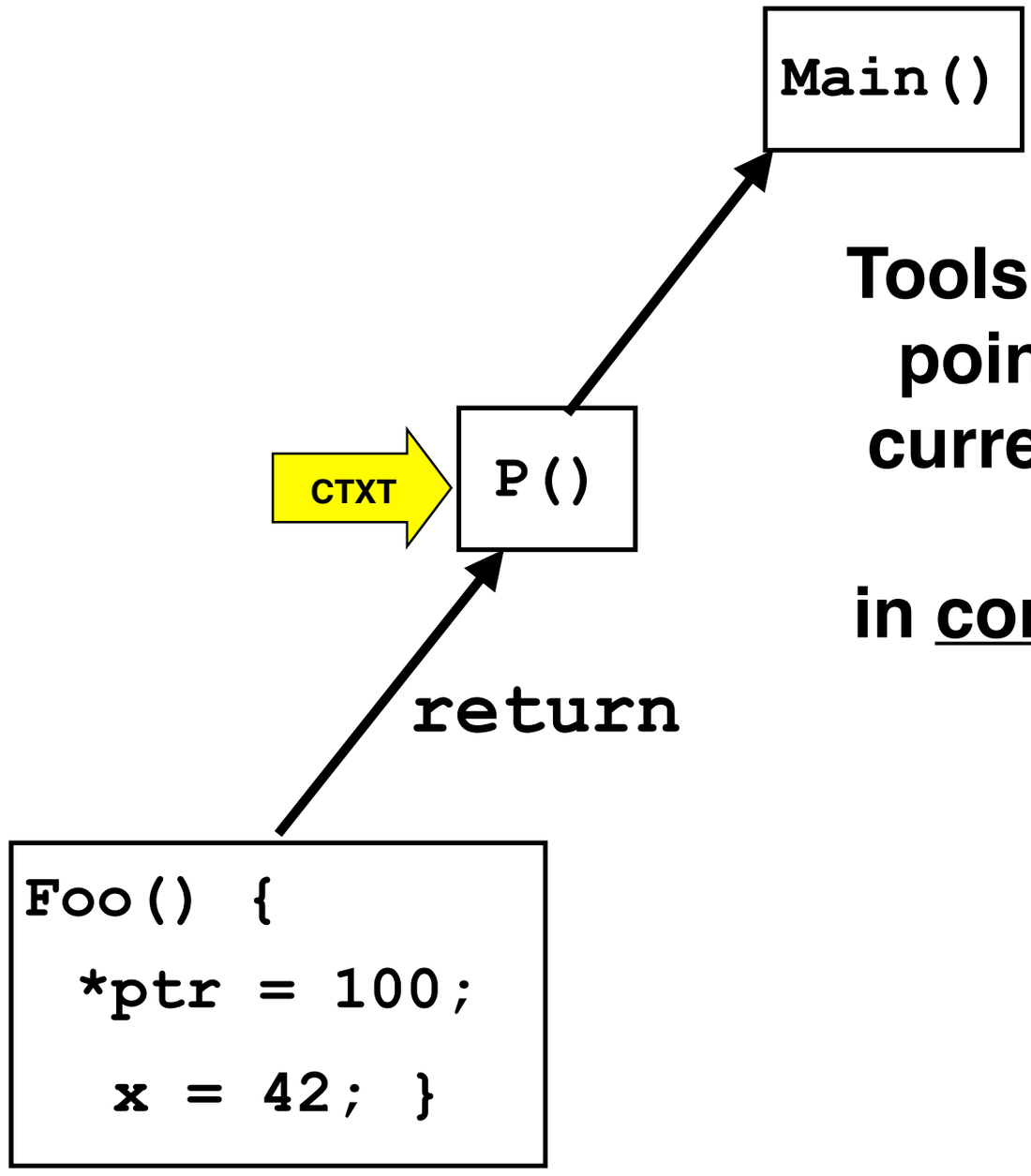
Tools can obtain pointer to the current context via CTXT **in <u>constant time</u>**

Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

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



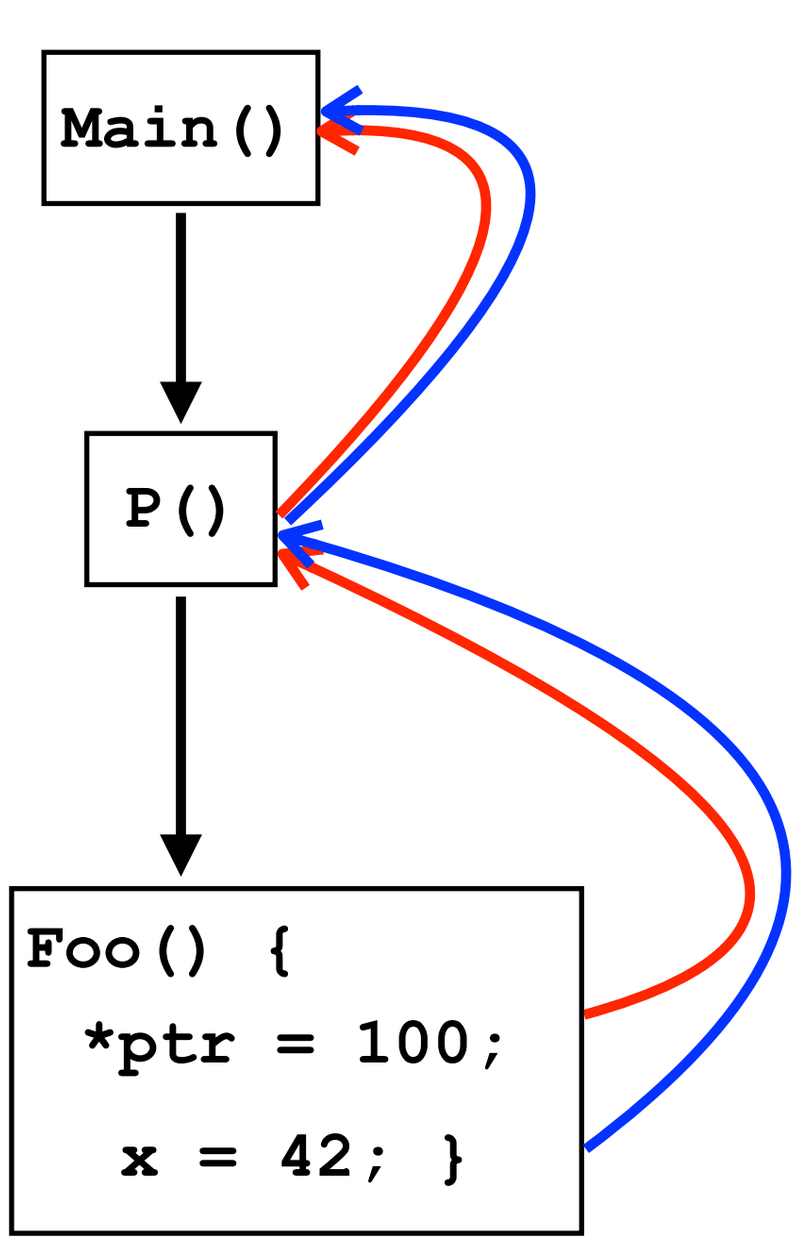


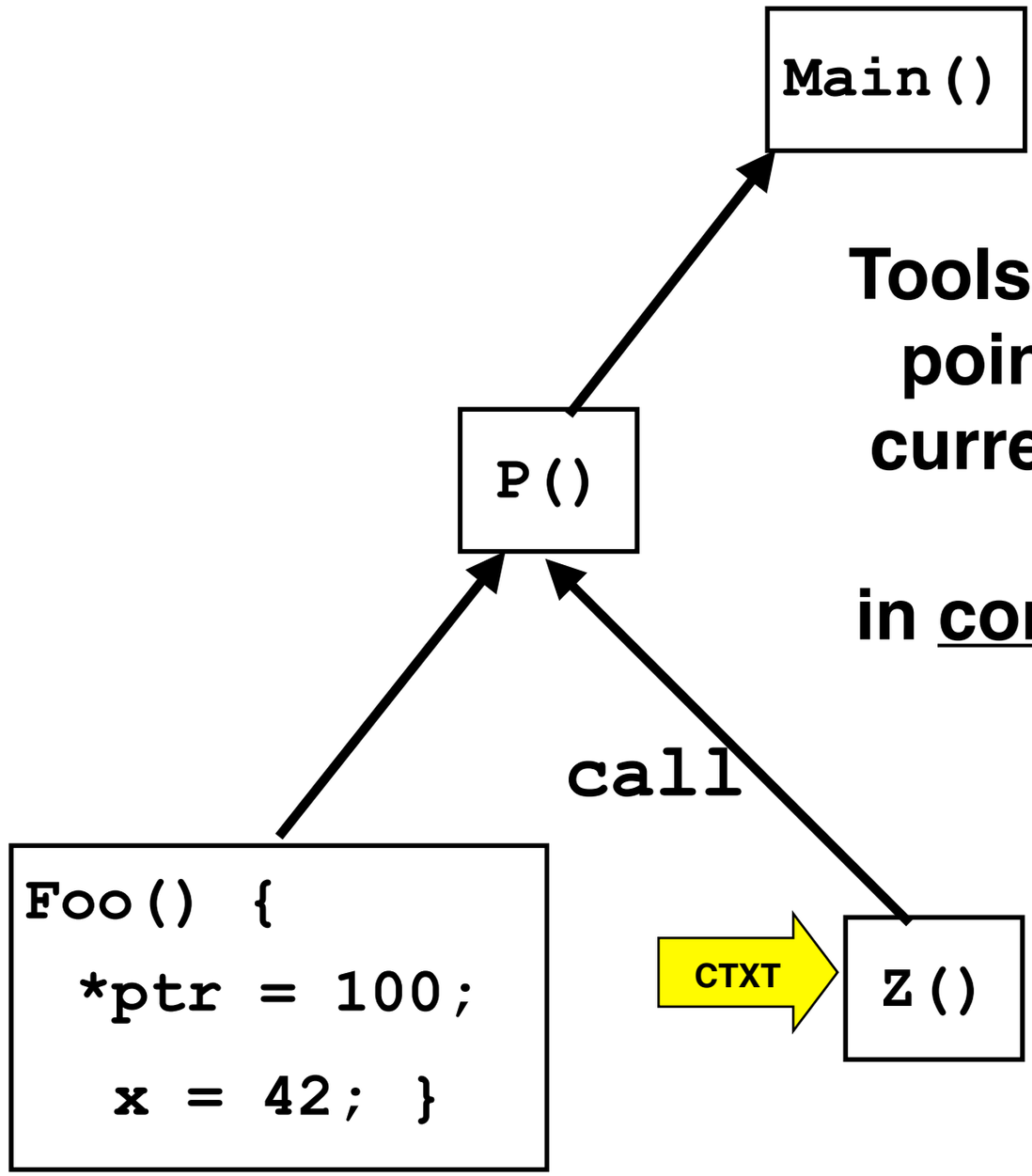
Tools can obtain pointer to the current context via CTXT in constant time

Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

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

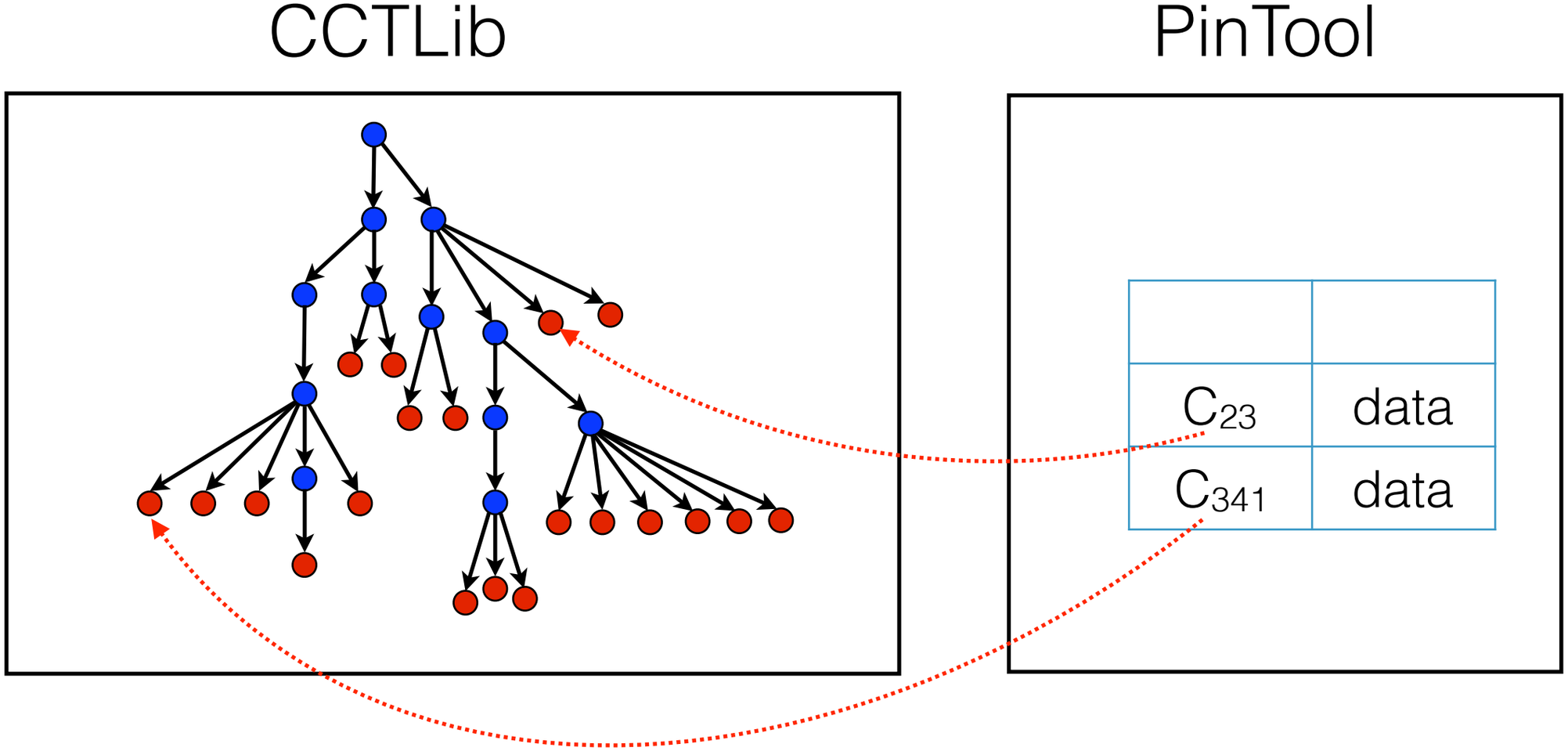




Tools can obtain pointer to the current context via CTXT in constant time

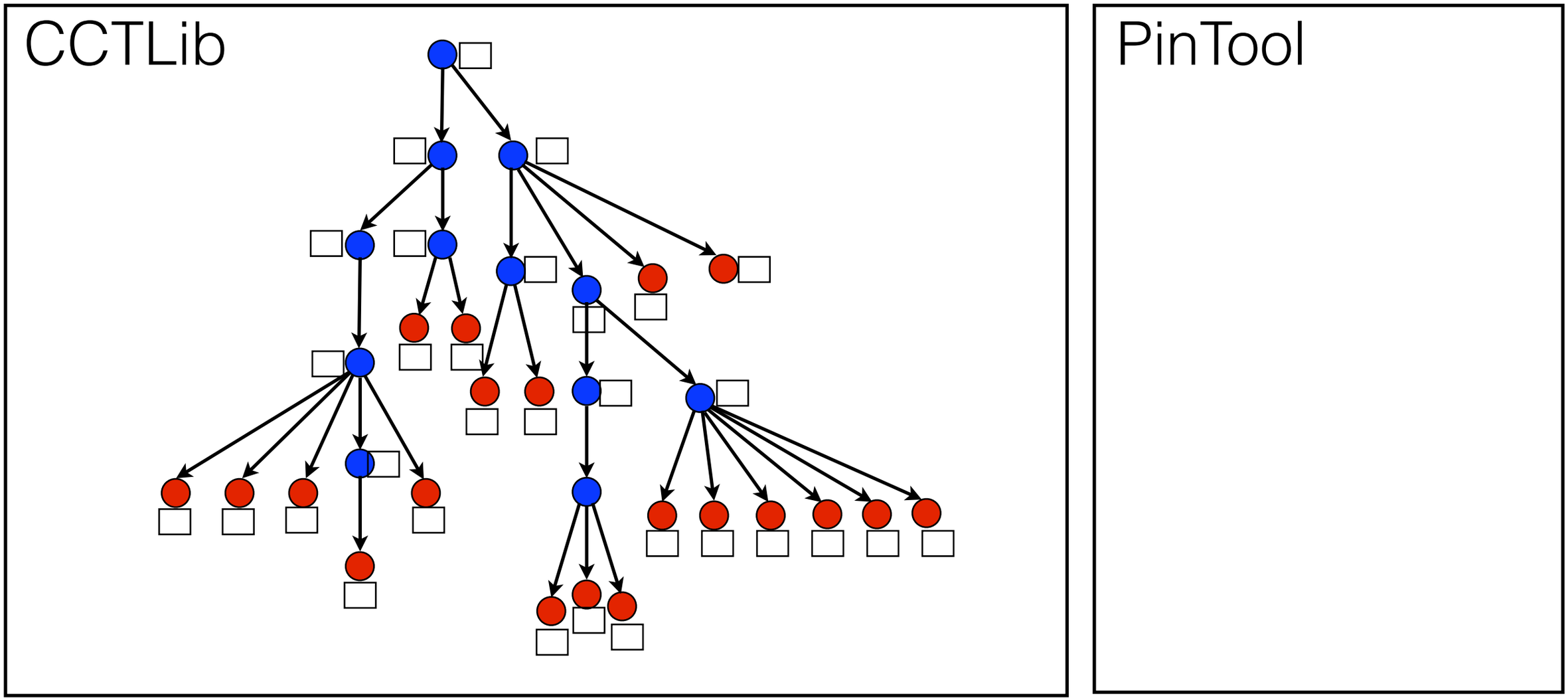
Two Ways to Use CCTLib By Pin Tools

- Option 1: Store context handles (ContextHandle_t) within the Pin tool and access the context (traverse full call chain) as needed



Two Ways to Use CCTLib By Pin Tools

- Option 2: Associate a user-defined “metric” with each `ContextHandle_t` and store it in the calling context tree. Perform a tree traversal as needed.



Associating Address to 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

Context To DOT

Execution-wide calling context tree for
NWChem—a six-million line computational chemistry code

```
/* Description:
            Dumps all CCTs into DOT files for visualization.
*/
    void DottifyAllCCTs();
```

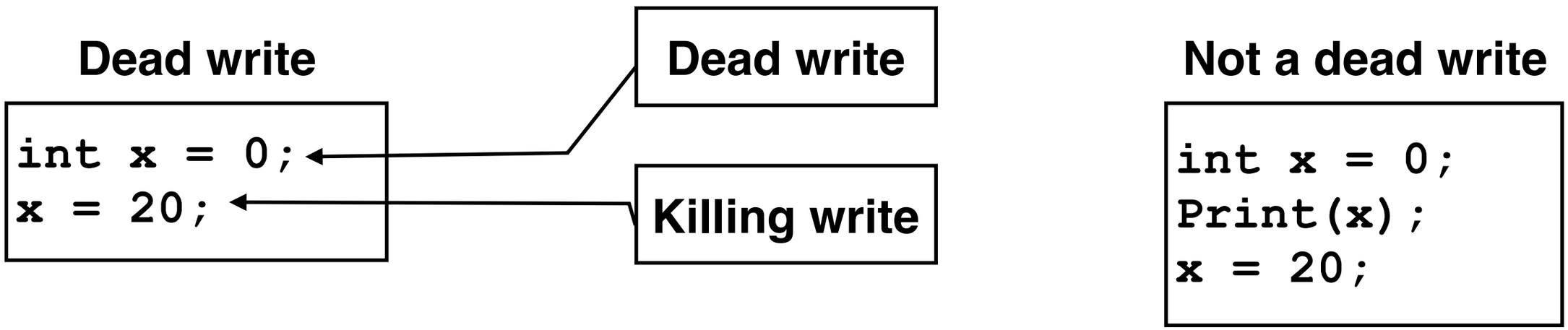
CCTLib Client Tools

- DeadSpy: Pinpoining dead stores in a program
  - ✦ Detects dead writes in an execution
- RedSpy: Pinpointing silent stores in a program
  - ✦ Detects redundant data movement in an execution
- RVN: Runtime Value Numbering
  - ✦ Detects useless computations in an execution
- Metric client
  - ✦ Captures hot paths
- Footprint client
  - ✦ Computes context-sensitive memory footprint of a data object

Pinpointing Useless Memory Accesses

- Accessing memory is expensive on modern architectures
  - Multiple levels of hierarchy
  - Cores share cache
  - Limited bandwidth per core
- Unnecessary writes
  - Cause unnecessary capacity miss and coherence traffic —> affects resource shared system
  - Wear out NVM-based or disk-based memory

Dead write: Two writes happen to the same memory location without an intervening read



Dead Writes: Example

Riemann solver kernel
3-level nested loop
20% execution time

```
do k
 do j
  do i
   Wgdnv(i, j, k, 0) = …
   Wgdnv(i, j, k, inorm) = …
   Wgdnv(i, j, k, 4) = …
   if (spout.le.0.0d0) then
    Wgdnv(i, j, k, 0) = …
    Wgdnv(i, j, k, inorm) = …
    Wgdnv(i, j, k, 4) = …
   endif
   if (spin.gt.0.0d0) then
    Wgdnv(i, j, k, 0) = …
    Wgdnv(i, j, k, inorm) = …
    Wgdnv(i, j, k, 4) = …
   endif
```

- Chombo: AMR framework for solving PDEs
- Compilers can't eliminate all dead writes because of:
  - Aliasing / ambiguity
  - Aggregate variables
  - Function boundaries
  - Late binding
  - Partial deadness

Dead Writes: Example

Code lacked "design for performance"

```
do k
 do j
  do i
   Wgdnv(i, j, k, 0) = …
   Wgdnv(i, j, k, inorm) = …
   Wgdnv(i, j, k, 4) = …
   if (spout.le.0.0d0) then
    Wgdnv(i, j, k, 0) = …
    Wgdnv(i, j, k, inorm) = …
    Wgdnv(i, j, k, 4) = …
   endif
   if (spin.gt.0.0d0) then
    Wgdnv(i, j, k, 0) = …
    Wgdnv(i, j, k, inorm) = …
    Wgdnv(i, j, k, 4) = …
   endif
```

Better code: Use else-if nesting

```
do k
 do j
  do i
   if (spin.gt.0.0d0) then
    Wgdnv(i, j, k, 0) = …
    Wgdnv(i, j, k, inorm) = …
    Wgdnv(i, j, k, 4) = …
   elif (spout.le.0.0d0) then
    Wgdnv(i, j, k, 0) = …
    Wgdnv(i, j, k, inorm) = …
    Wgdnv(i, j, k, 4) = …
   else
    Wgdnv(i, j, k, 0) = …
    Wgdnv(i, j, k, inorm) = …
    Wgdnv(i, j, k, 4) = …
   endif
```

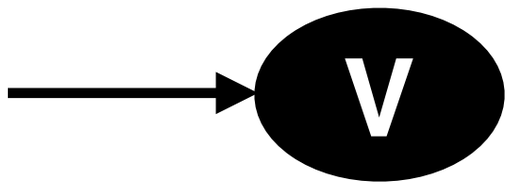
Detection Scheme

- Monitor every load and store in a program
- Maintain state information for each memory byte referenced by the program
- Detect every dead write in an execution with an automaton

[CGO'12] "DeadSpy: A Tool to Pinpoint Program Inefficiencies"

Detection Scheme

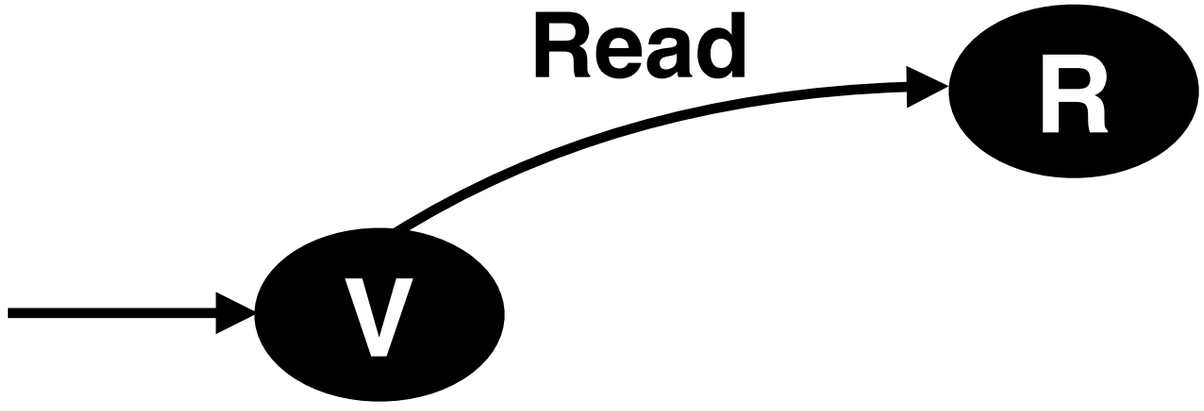
- Monitor every load and store in a program
- Maintain state information for each memory byte referenced by the program
- Detect every dead write in an execution with an automaton



[CGO'12] "DeadSpy: A Tool to Pinpoint Program Inefficiencies"

Detection Scheme

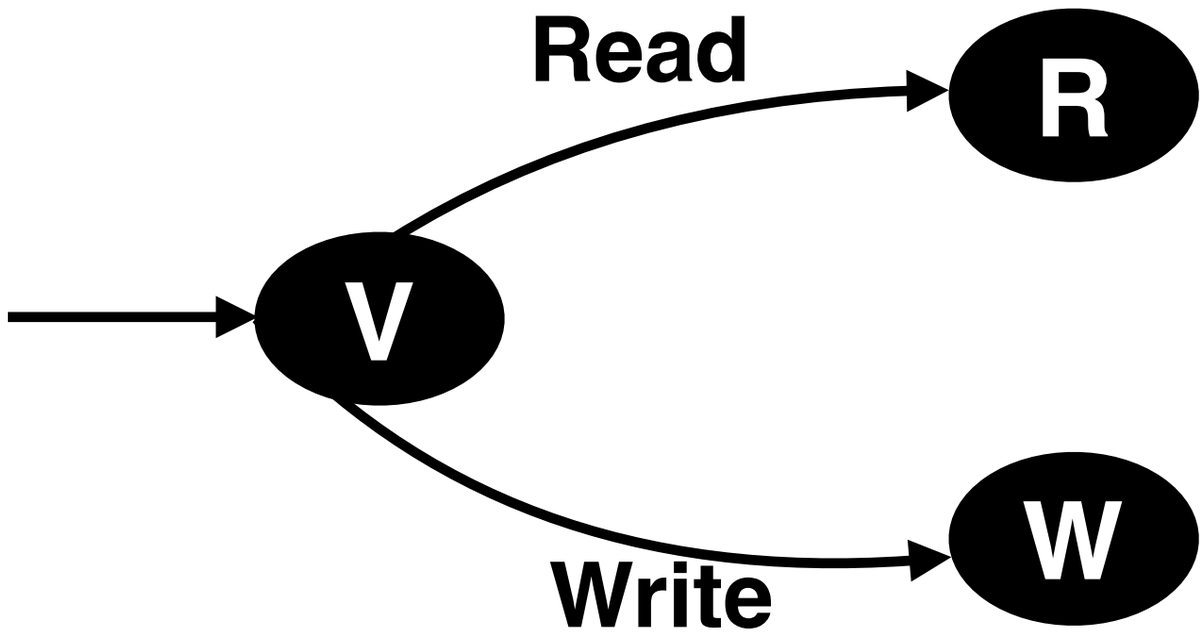
- Monitor every load and store in a program
- Maintain state information for each memory byte referenced by the program
- Detect every dead write in an execution with an automaton



[CGO'12] "DeadSpy: A Tool to Pinpoint Program Inefficiencies"

Detection Scheme

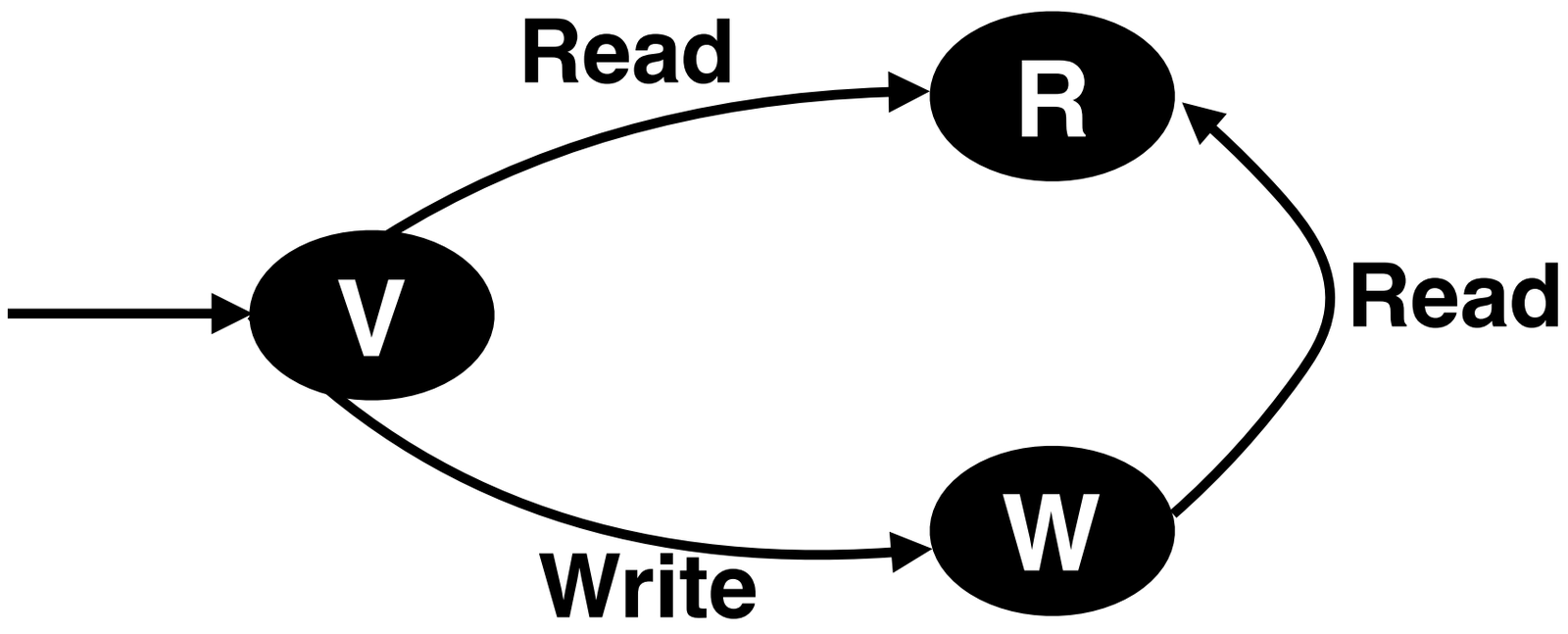
- Monitor every load and store in a program
- Maintain state information for each memory byte referenced by the program
- Detect every dead write in an execution with an automaton



[CGO'12] "DeadSpy: A Tool to Pinpoint Program Inefficiencies"

Detection Scheme

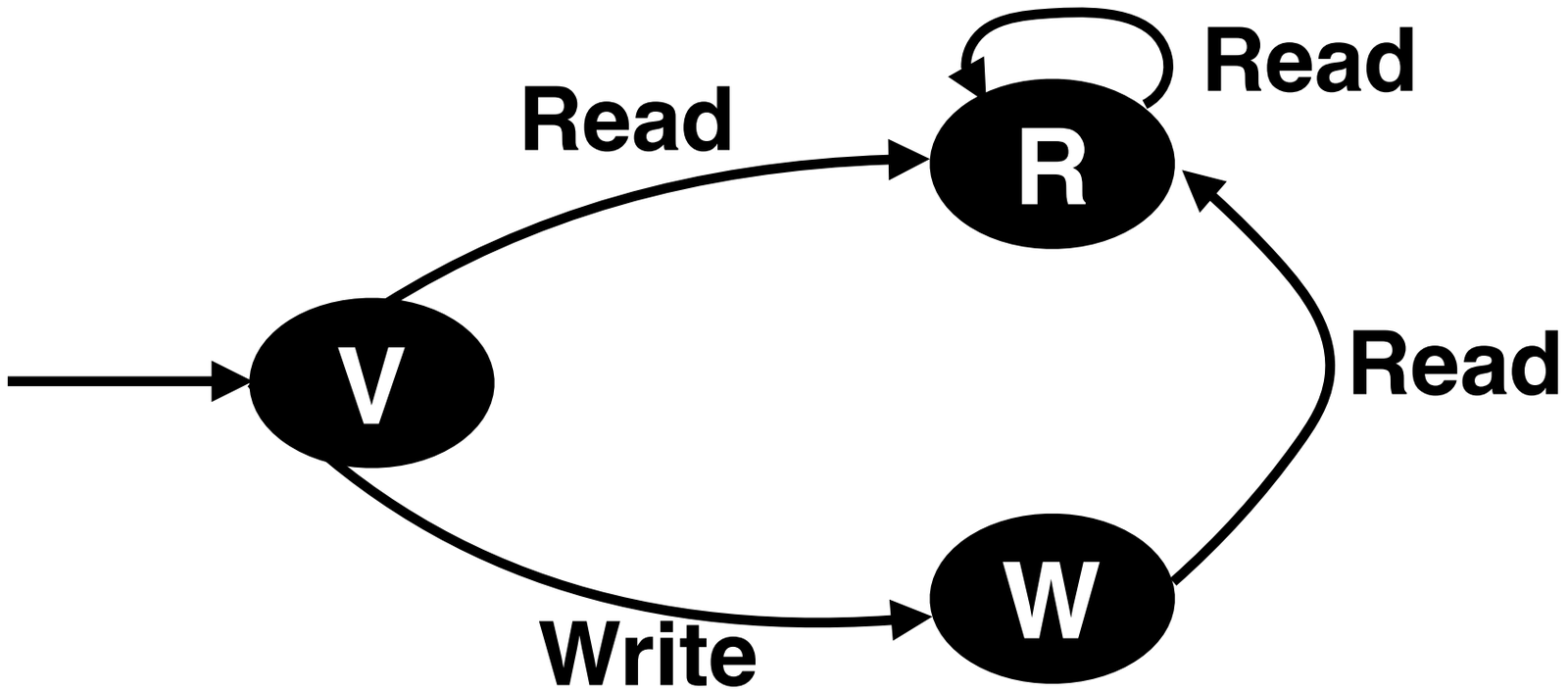
- Monitor every load and store in a program
- Maintain state information for each memory byte referenced by the program
- Detect every dead write in an execution with an automaton



[CGO'12] "DeadSpy: A Tool to Pinpoint Program Inefficiencies"

Detection Scheme

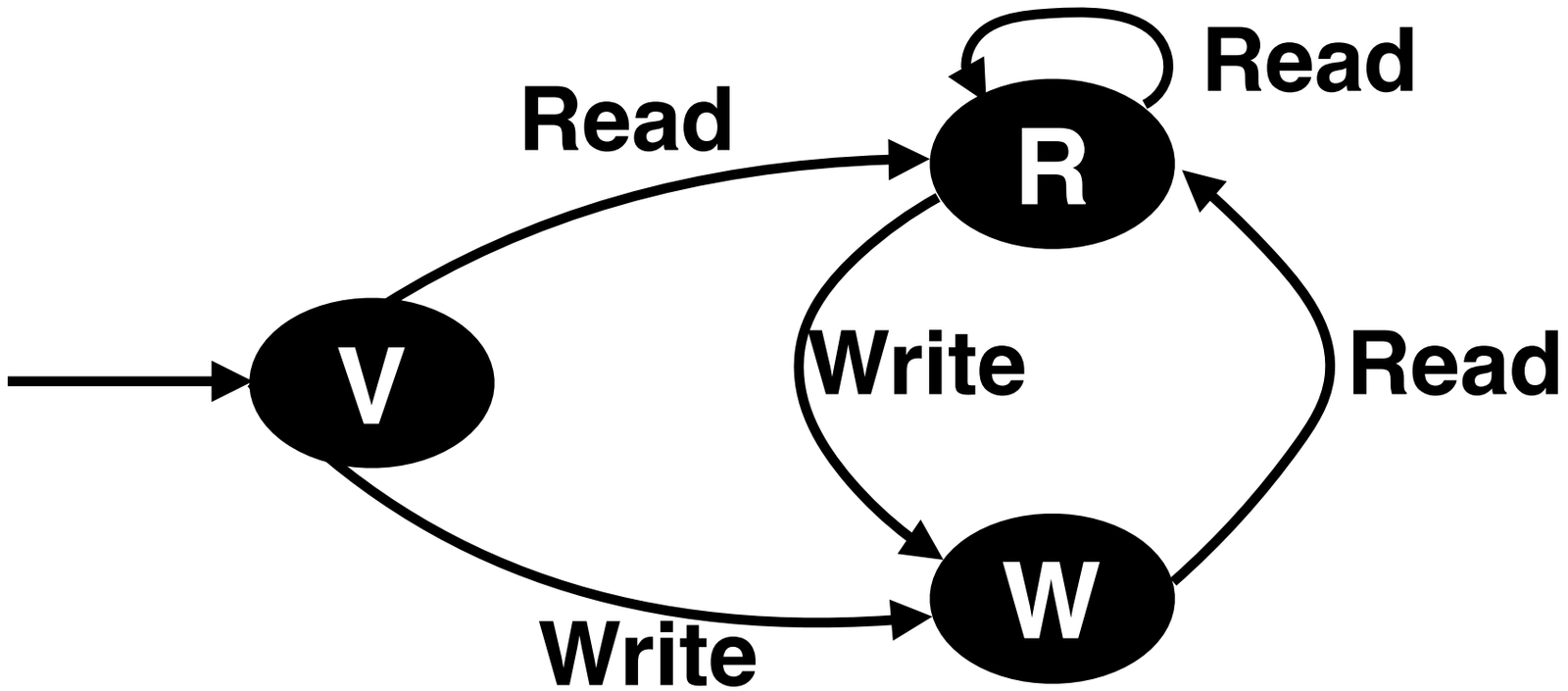
- Monitor every load and store in a program
- Maintain state information for each memory byte referenced by the program
- Detect every dead write in an execution with an automaton



[CGO'12] "DeadSpy: A Tool to Pinpoint Program Inefficiencies"

Detection Scheme

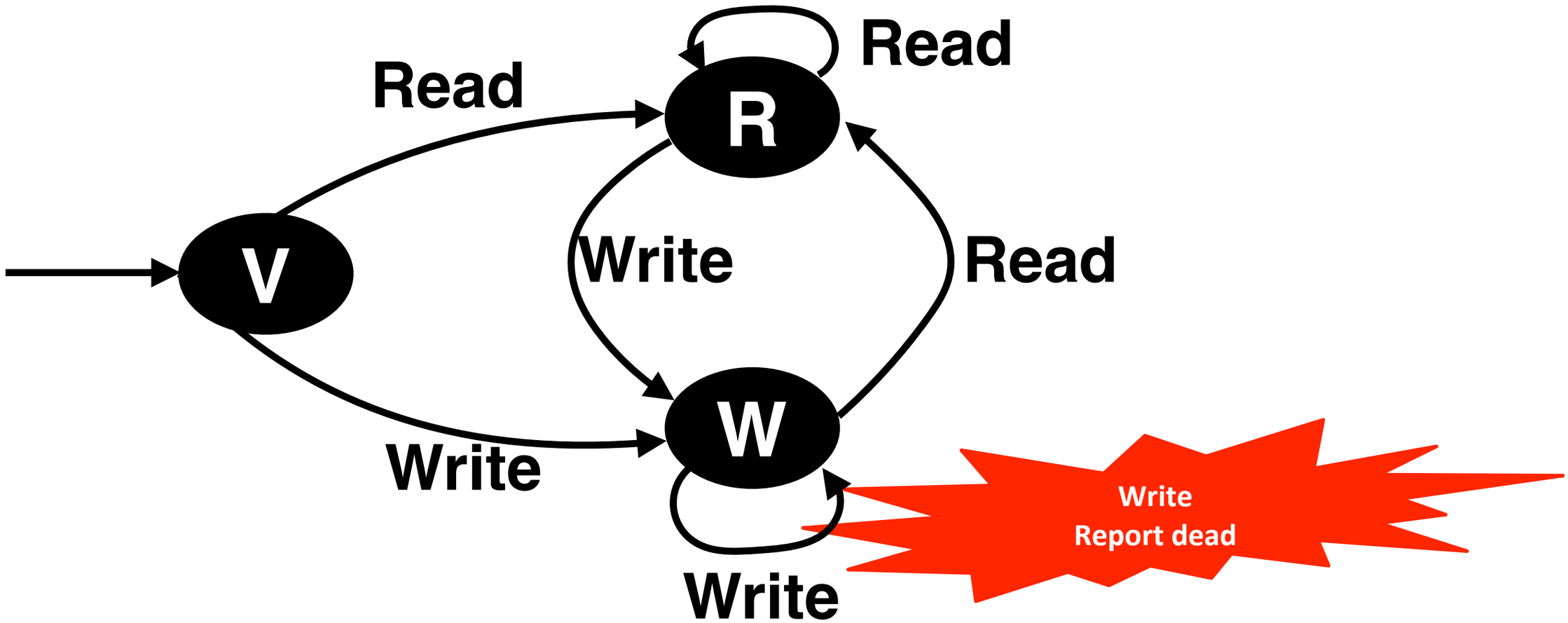
- Monitor every load and store in a program
- Maintain state information for each memory byte referenced by the program
- Detect every dead write in an execution with an automaton



[CGO'12] "DeadSpy: A Tool to Pinpoint Program Inefficiencies"

Detection Scheme

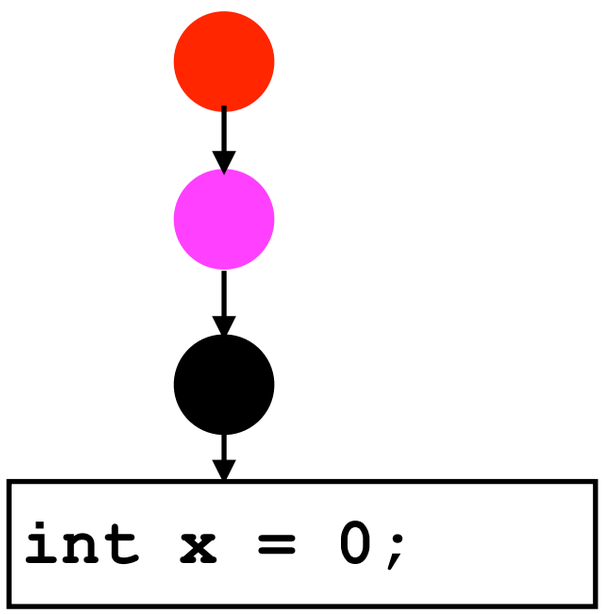
- Monitor every load and store in a program
- Maintain state information for each memory byte referenced by the program
- Detect every dead write in an execution with an automaton



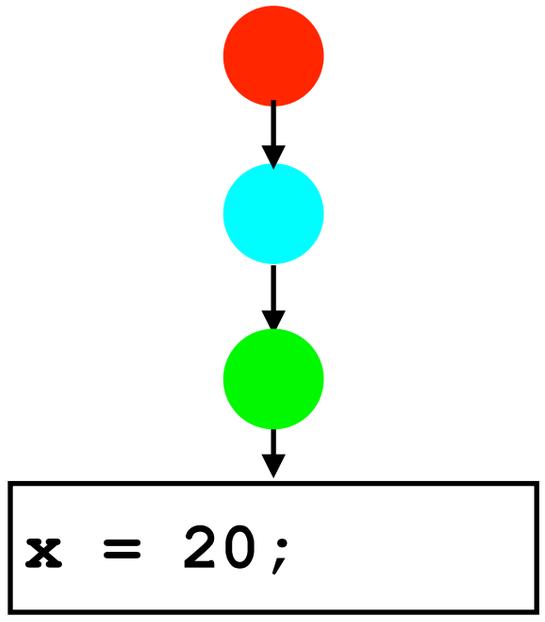
[CGO'12] "DeadSpy: A Tool to Pinpoint Program Inefficiencies"

DeadSpy: Measurement and Attribution

- Precise measurement
  - ✦ No false positives and no false negatives
- Precise attribution
  - ✦ Source-level feedback with calling context of dead and killing writes
  - ✦ On each dead write record <old-ctxt-handle, cur-ctxt-handle>



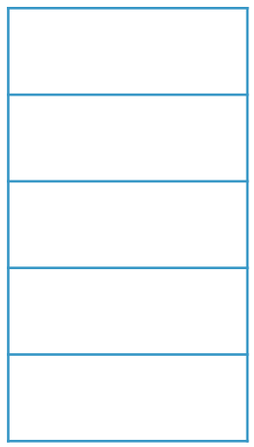
Dead write



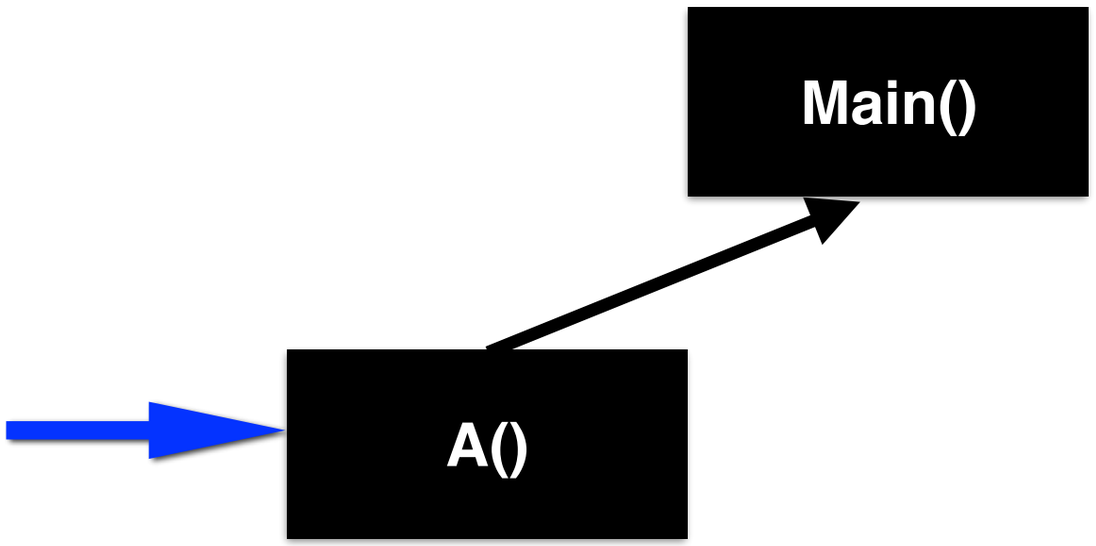
Killing write

DeadSpy + CCTLib in Action

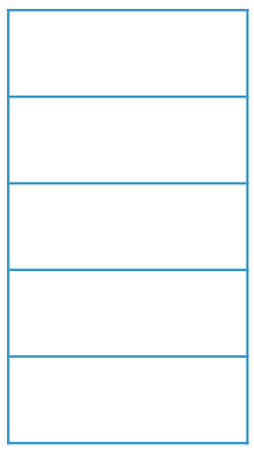
Memory



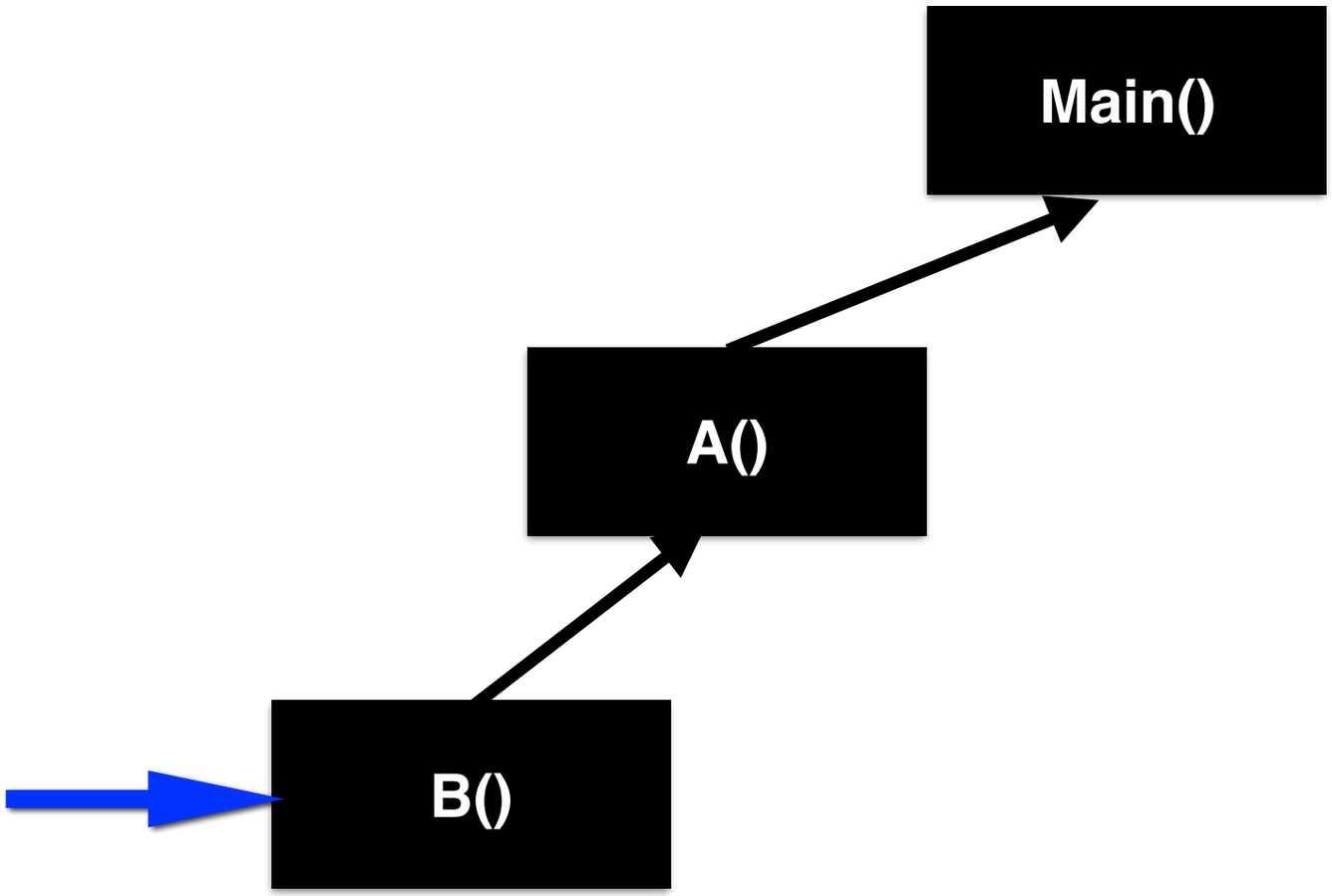
DeadSpy + CCTLib in Action



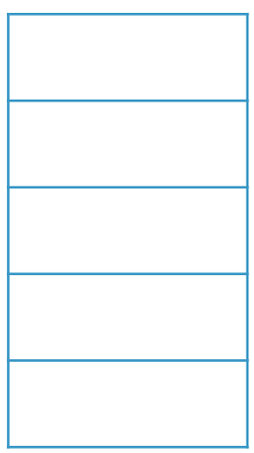
Memory



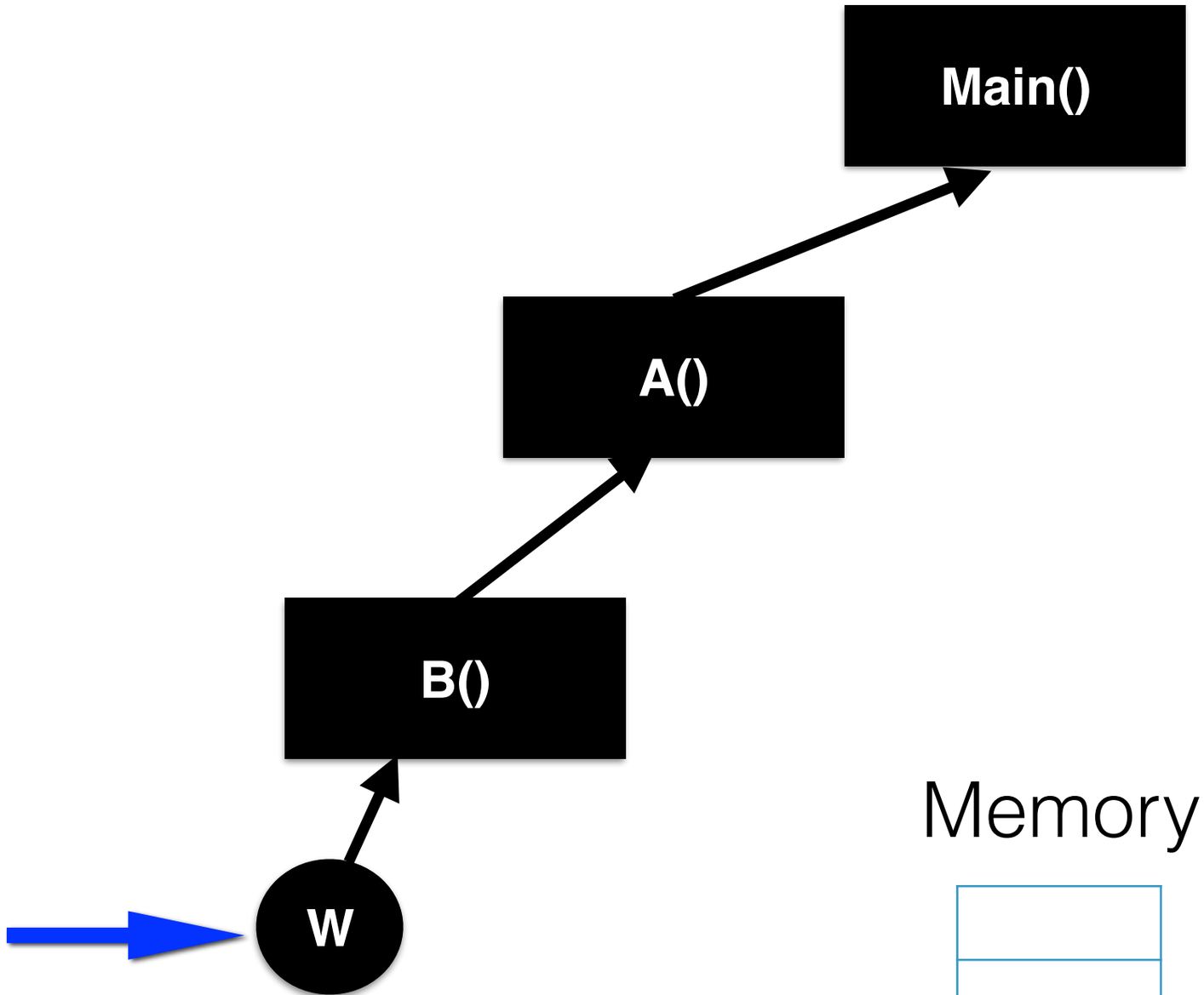
DeadSpy + CCTLib in Action



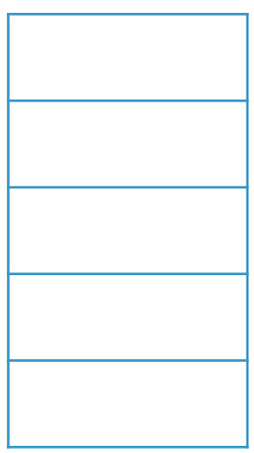
Memory



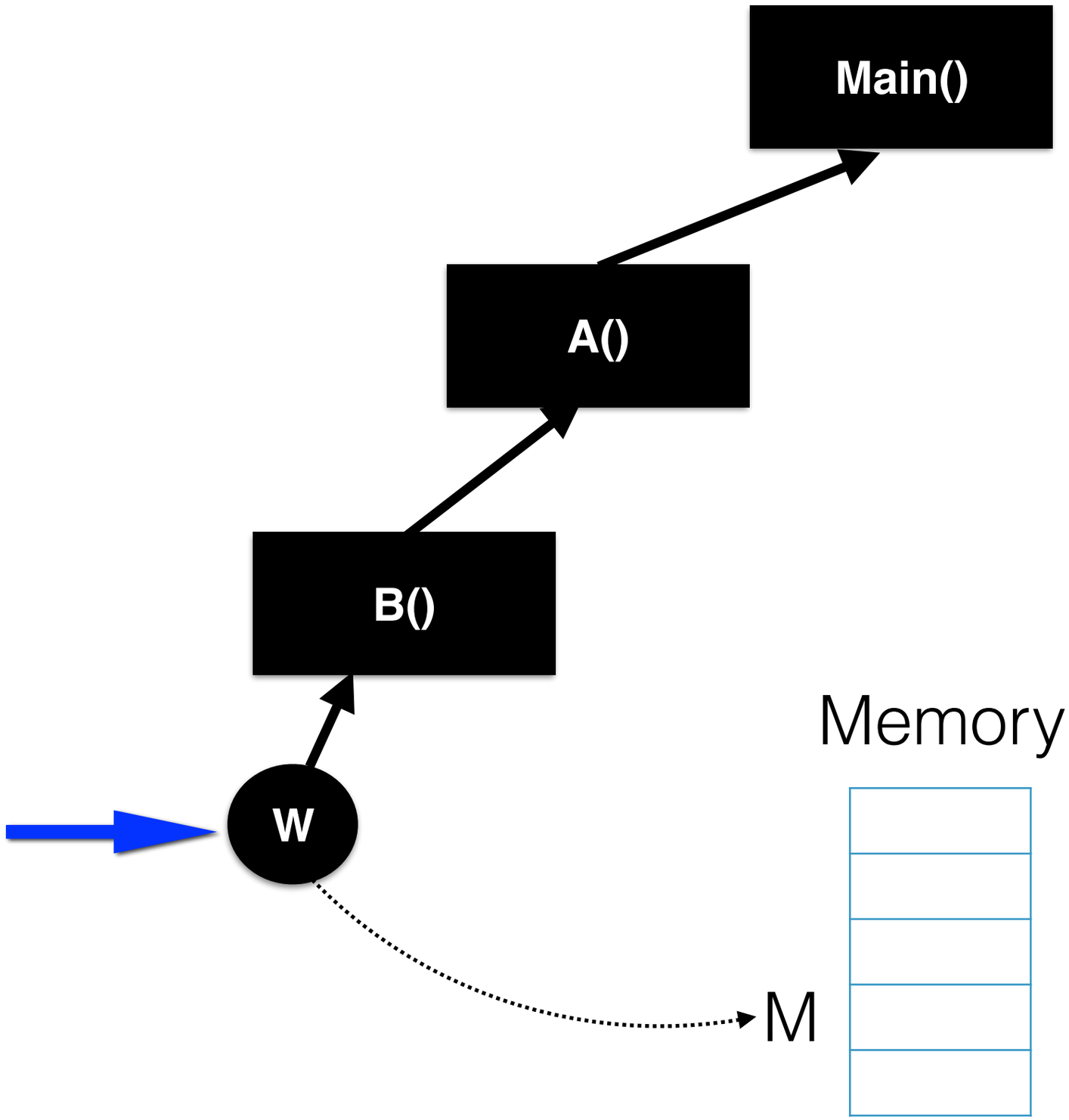
DeadSpy + CCTLib in Action



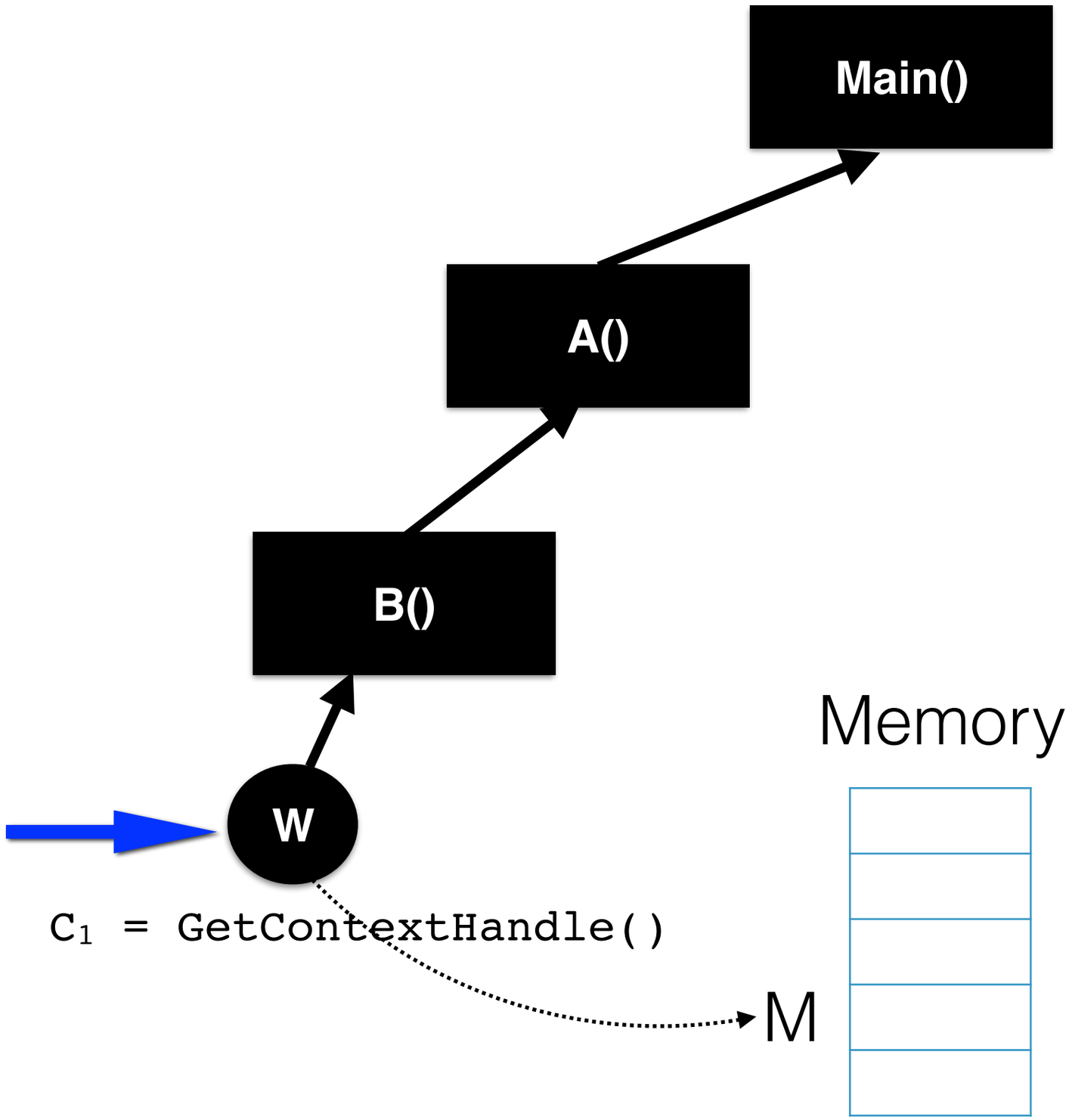
Memory



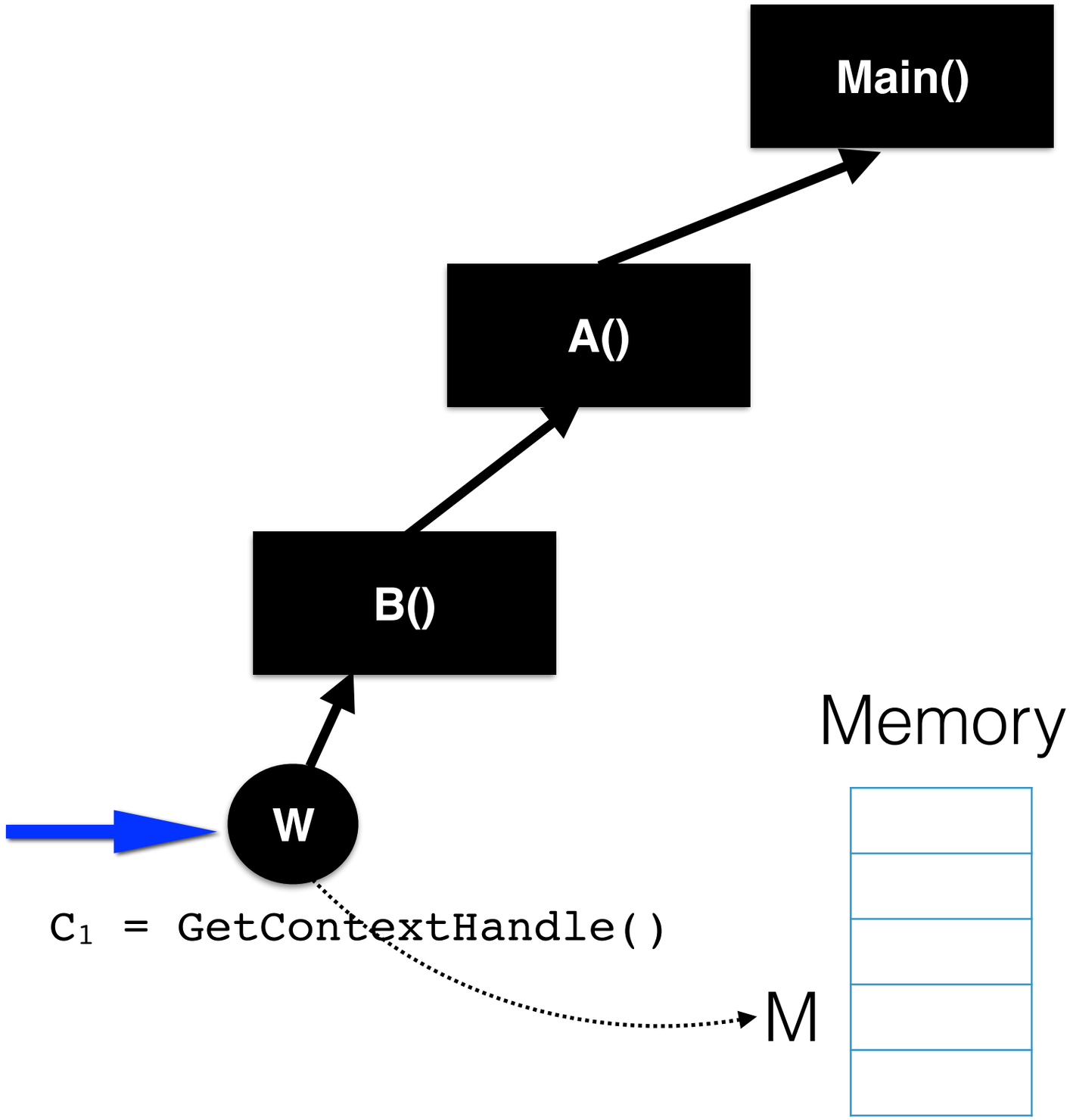
DeadSpy + CCTLib in Action



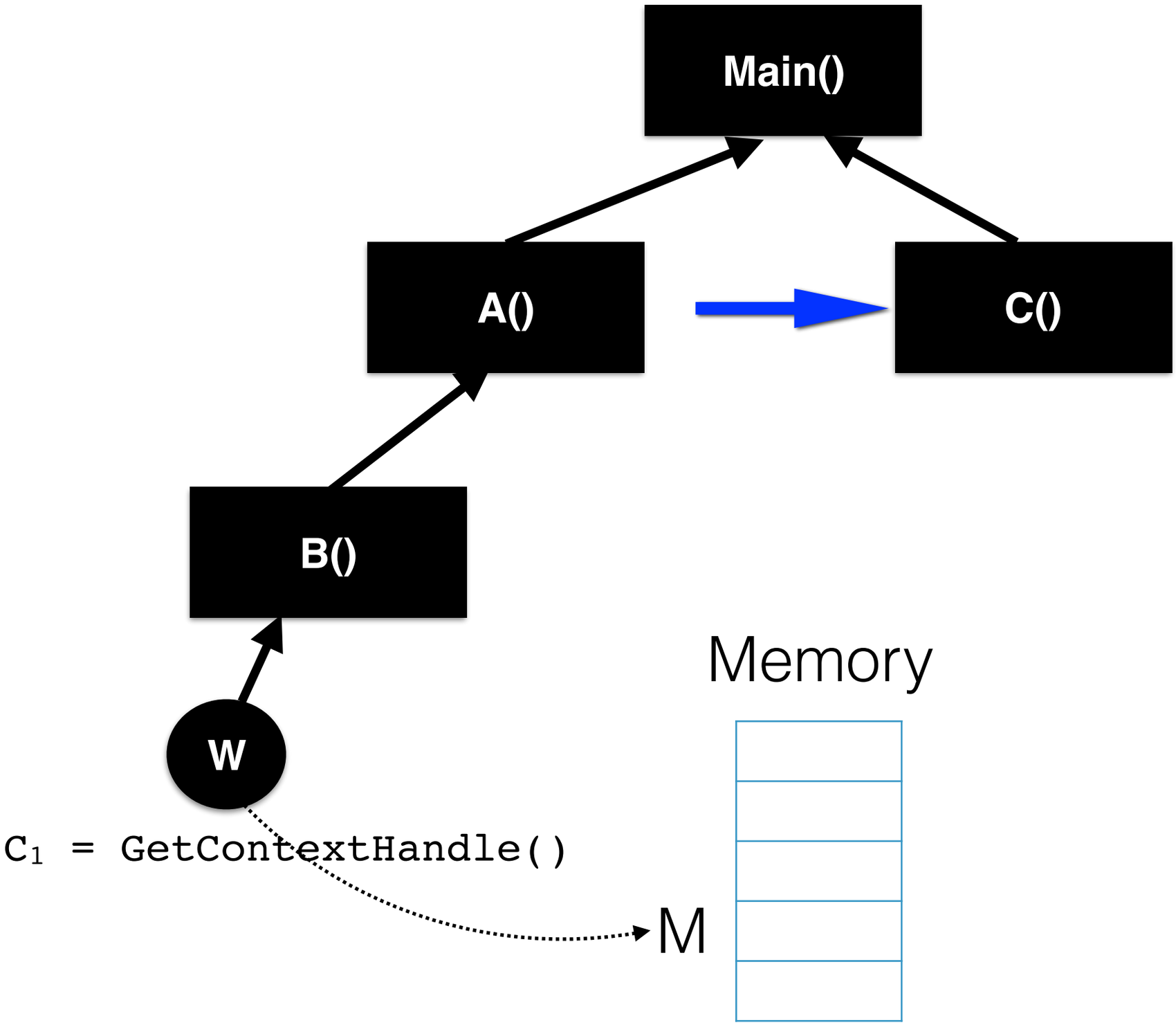
DeadSpy + CCTLib in Action



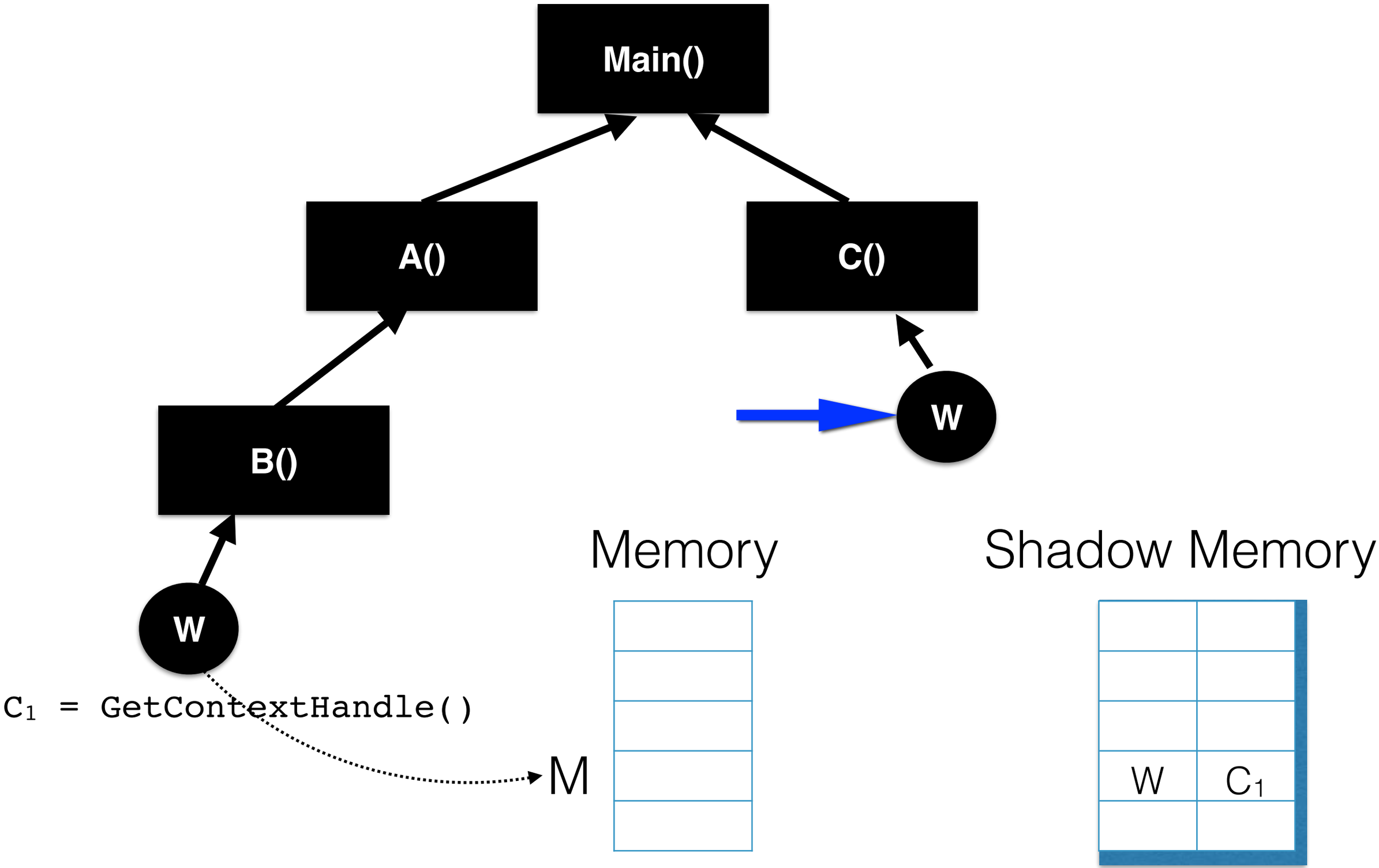
DeadSpy + CCTLib in Action



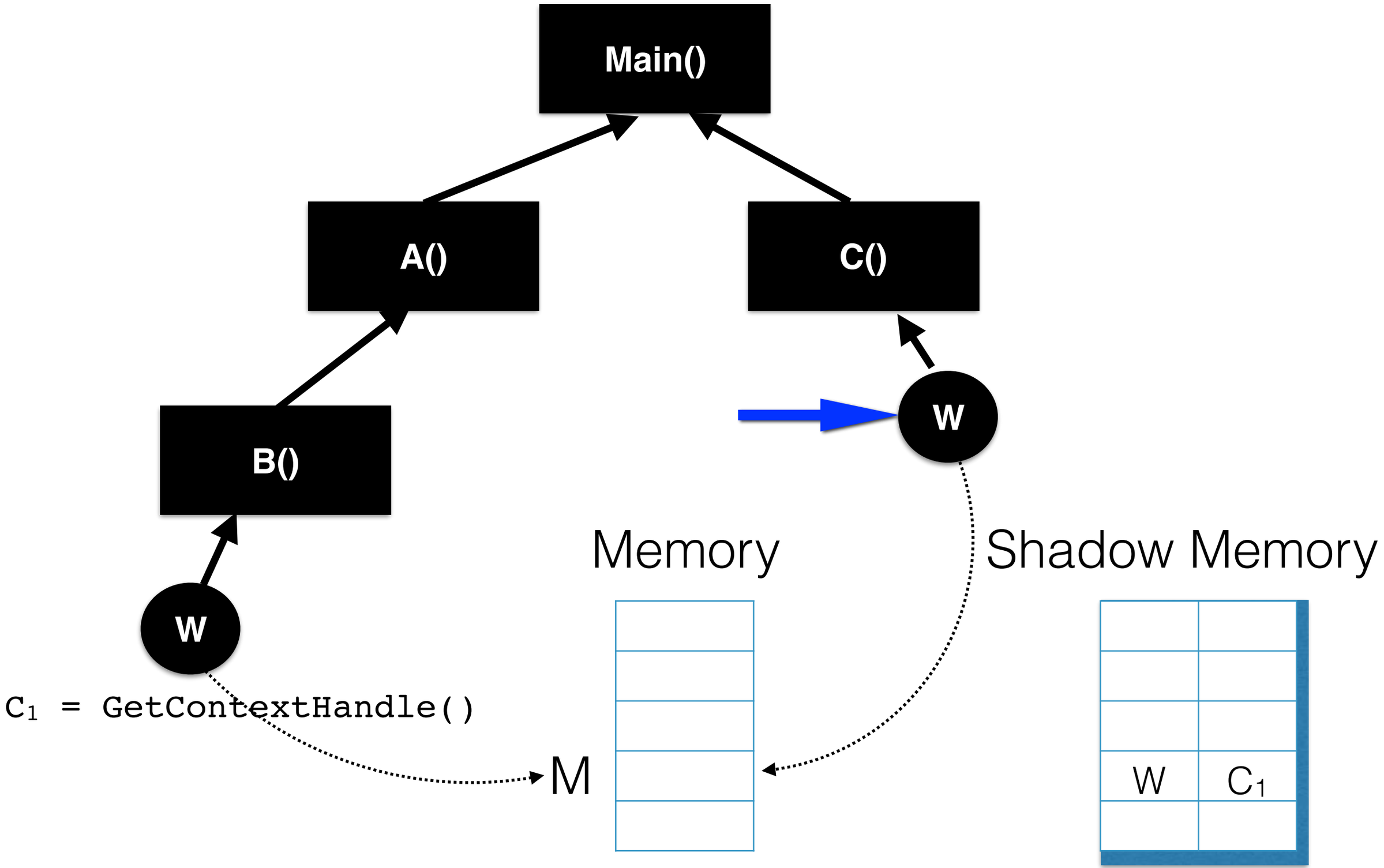
DeadSpy + CCTLib in Action



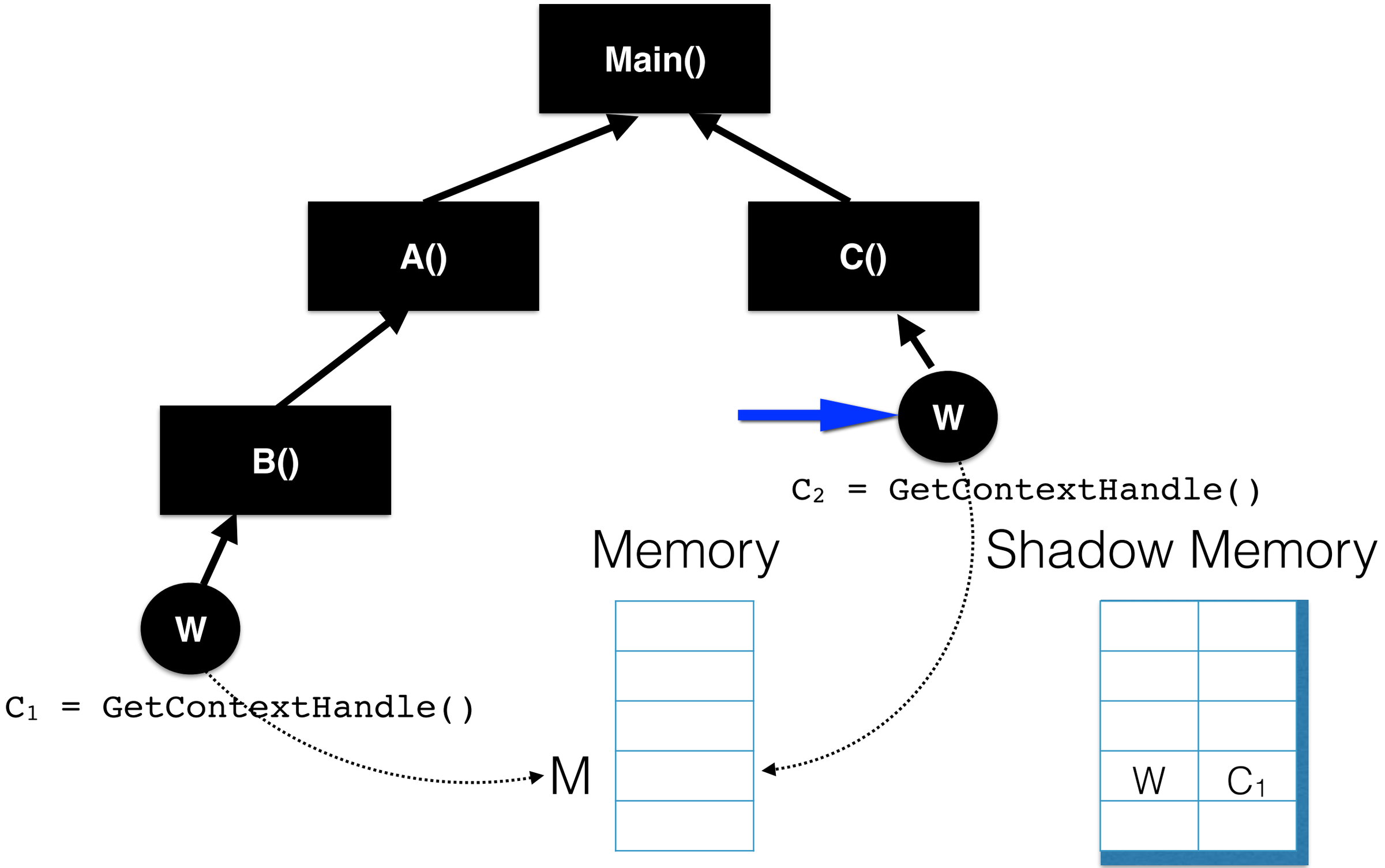
DeadSpy + CCTLib in Action



DeadSpy + CCTLib in Action



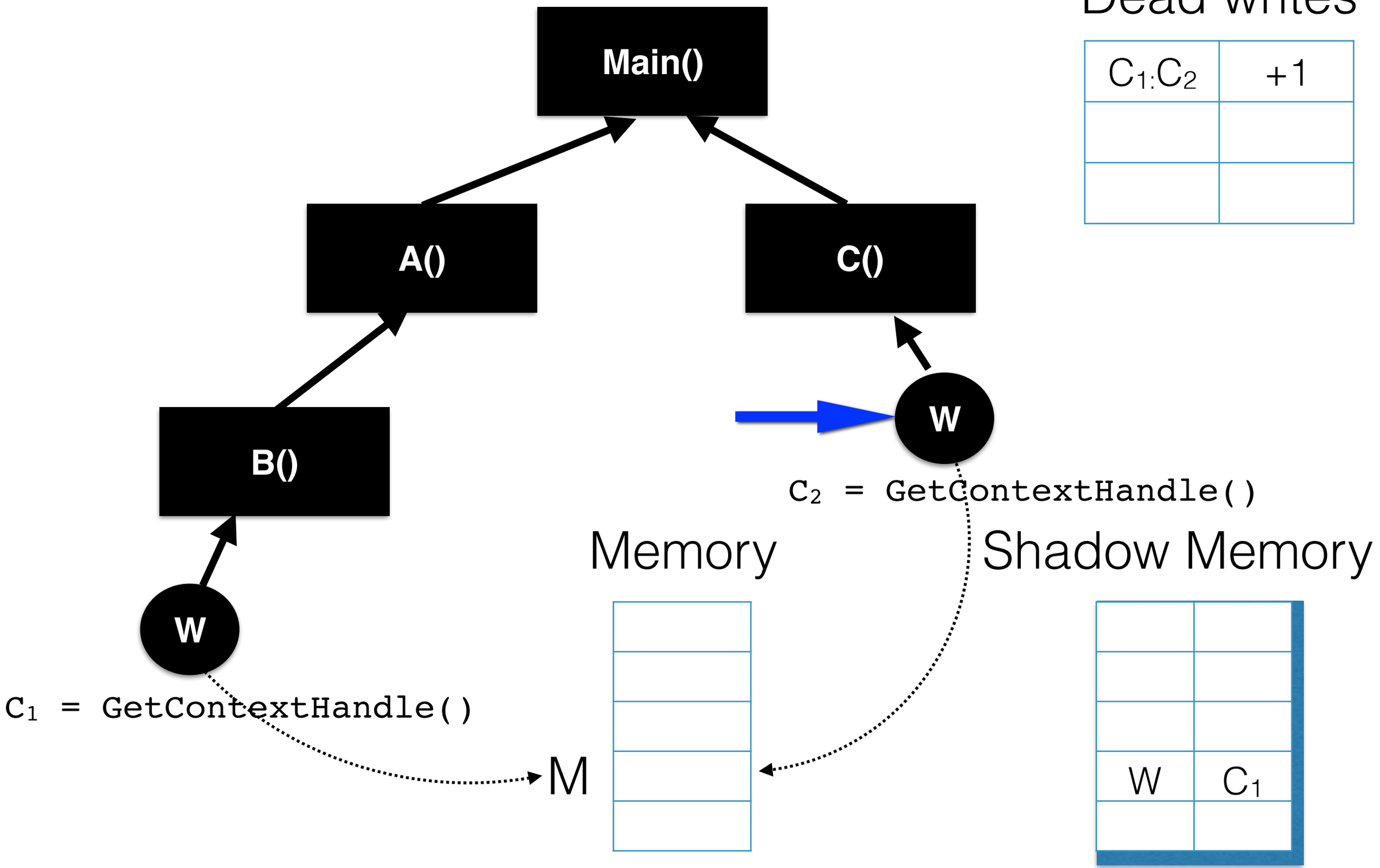
DeadSpy + CCTLib in Action



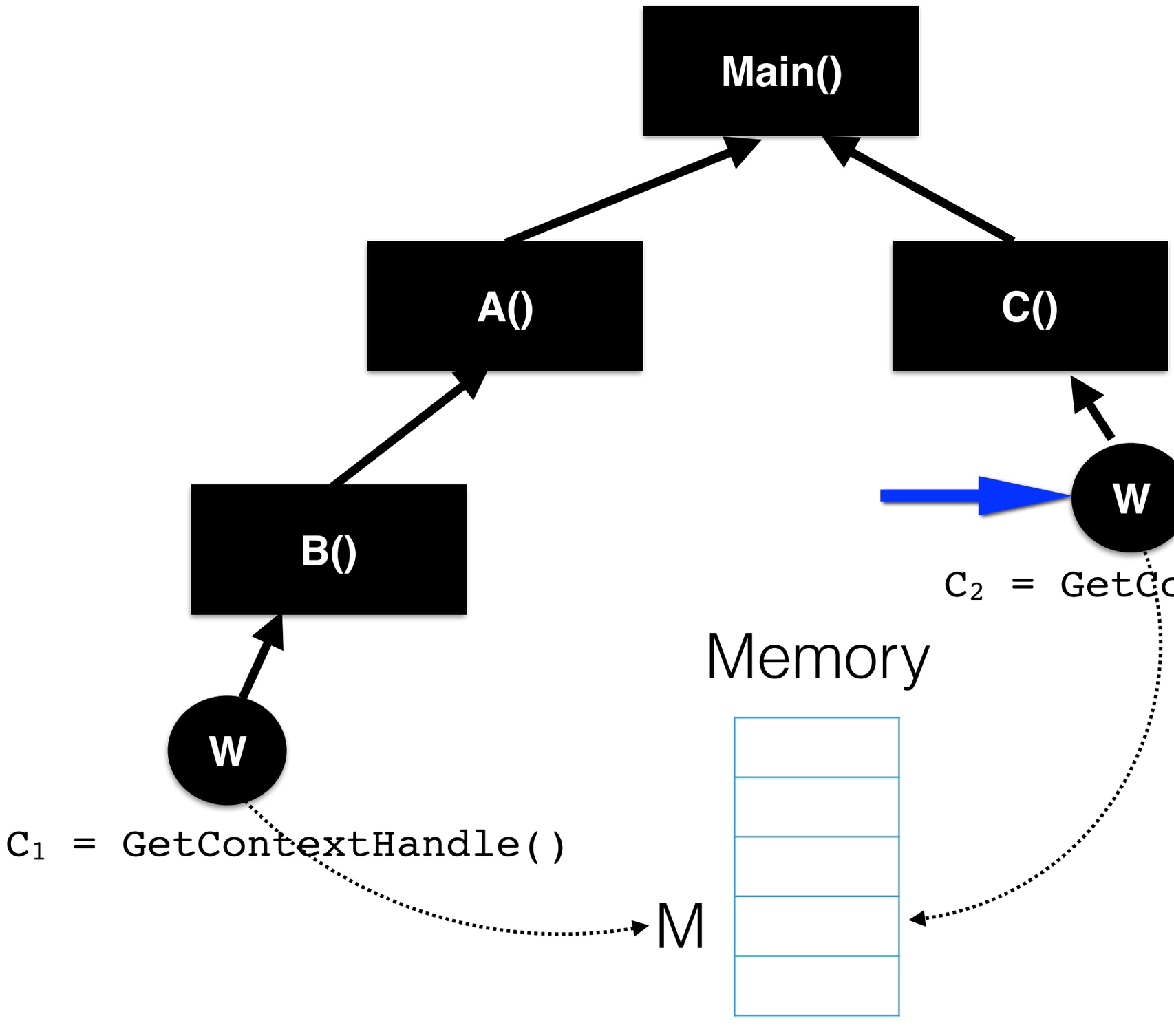
DeadSpy + CCTLib in Action

Dead writes

C_1:C_2	+1



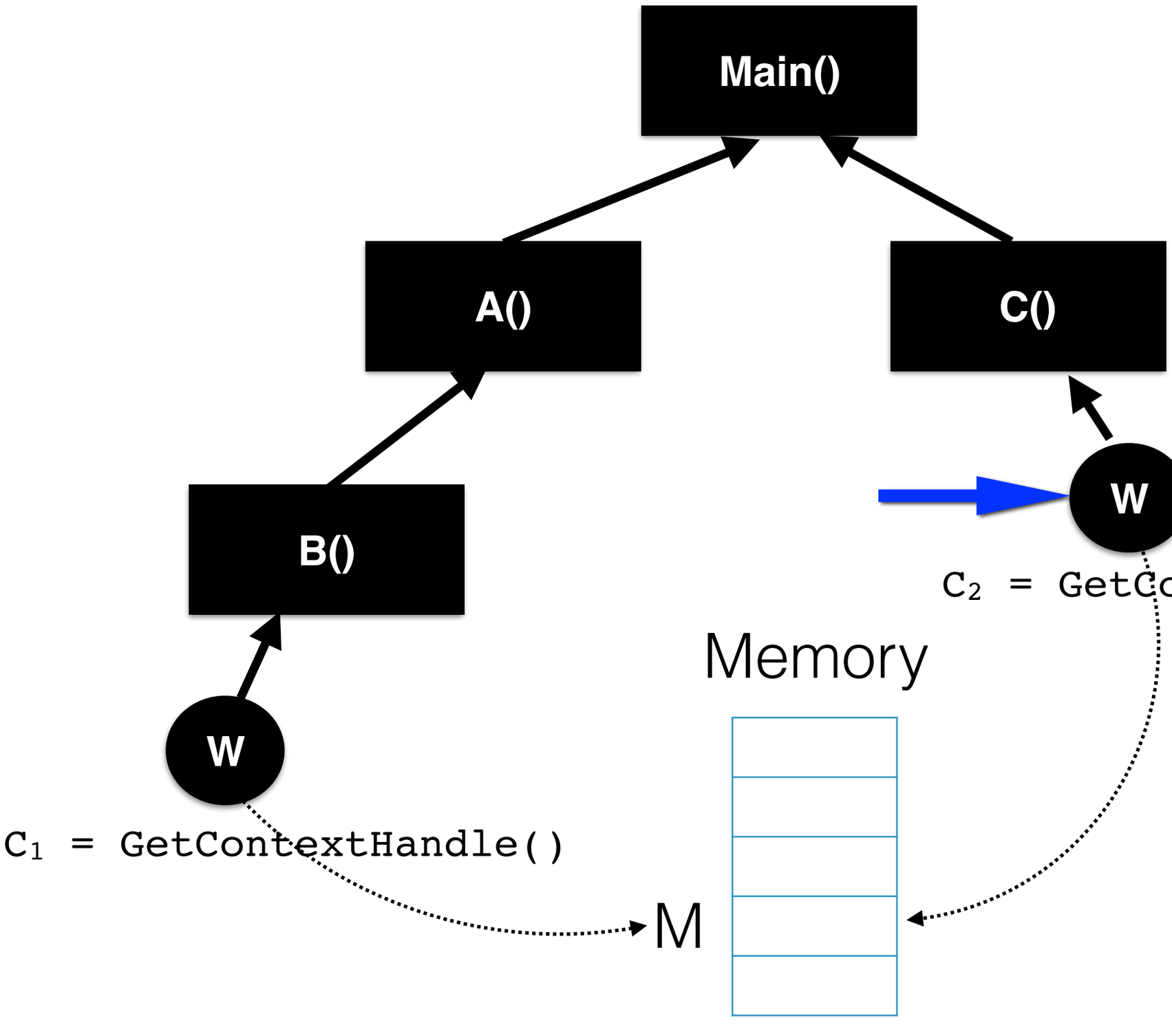
DeadSpy + CCTLib in Action



Dead writes

C_1:C_2	1234
C_3:C_{23}	959
C_2:C_{14}	546

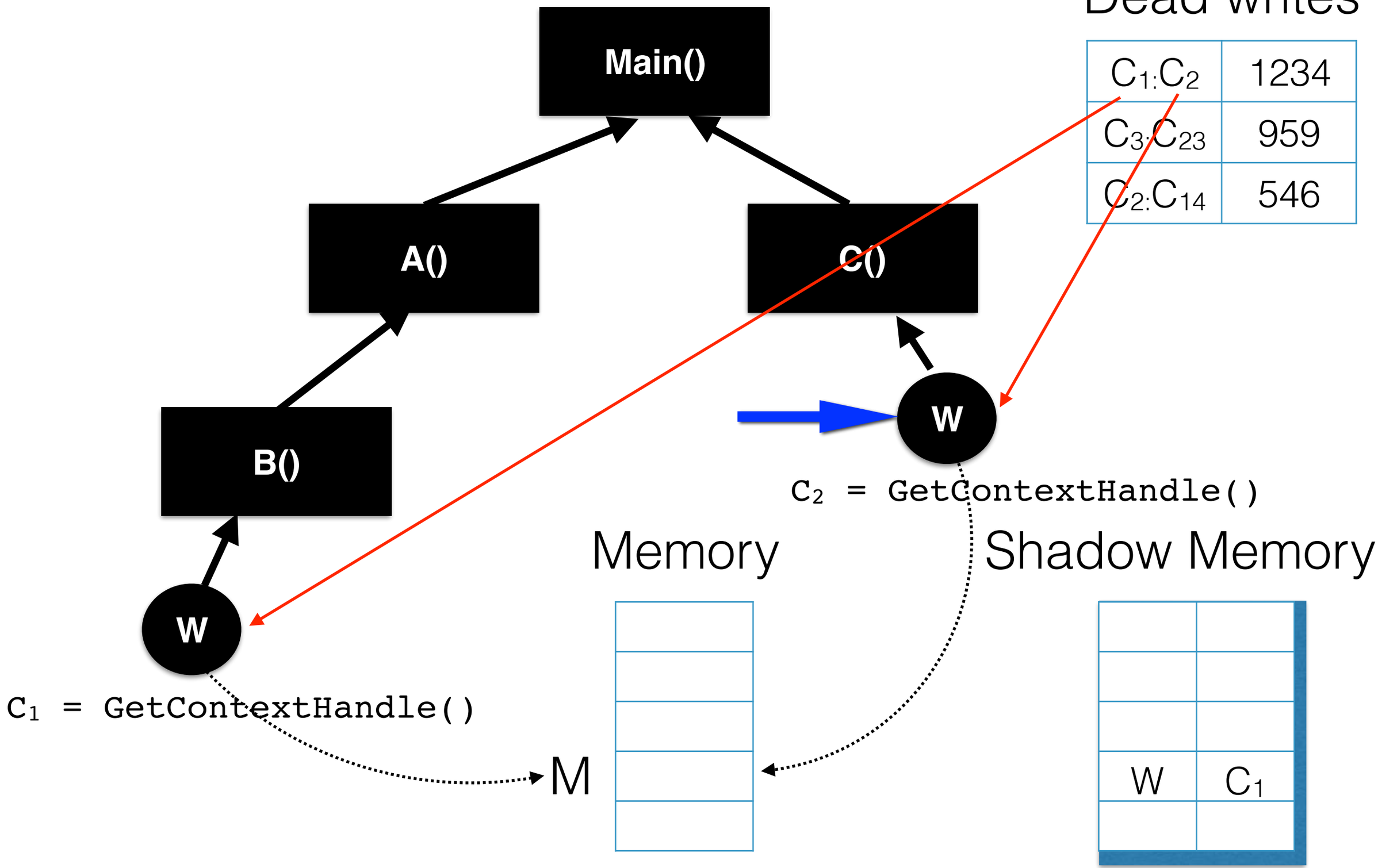
DeadSpy + CCTLib in Action



Dead writes

C_1:C_2	1234
C_3:C_{23}	959
C_2:C_{14}	546

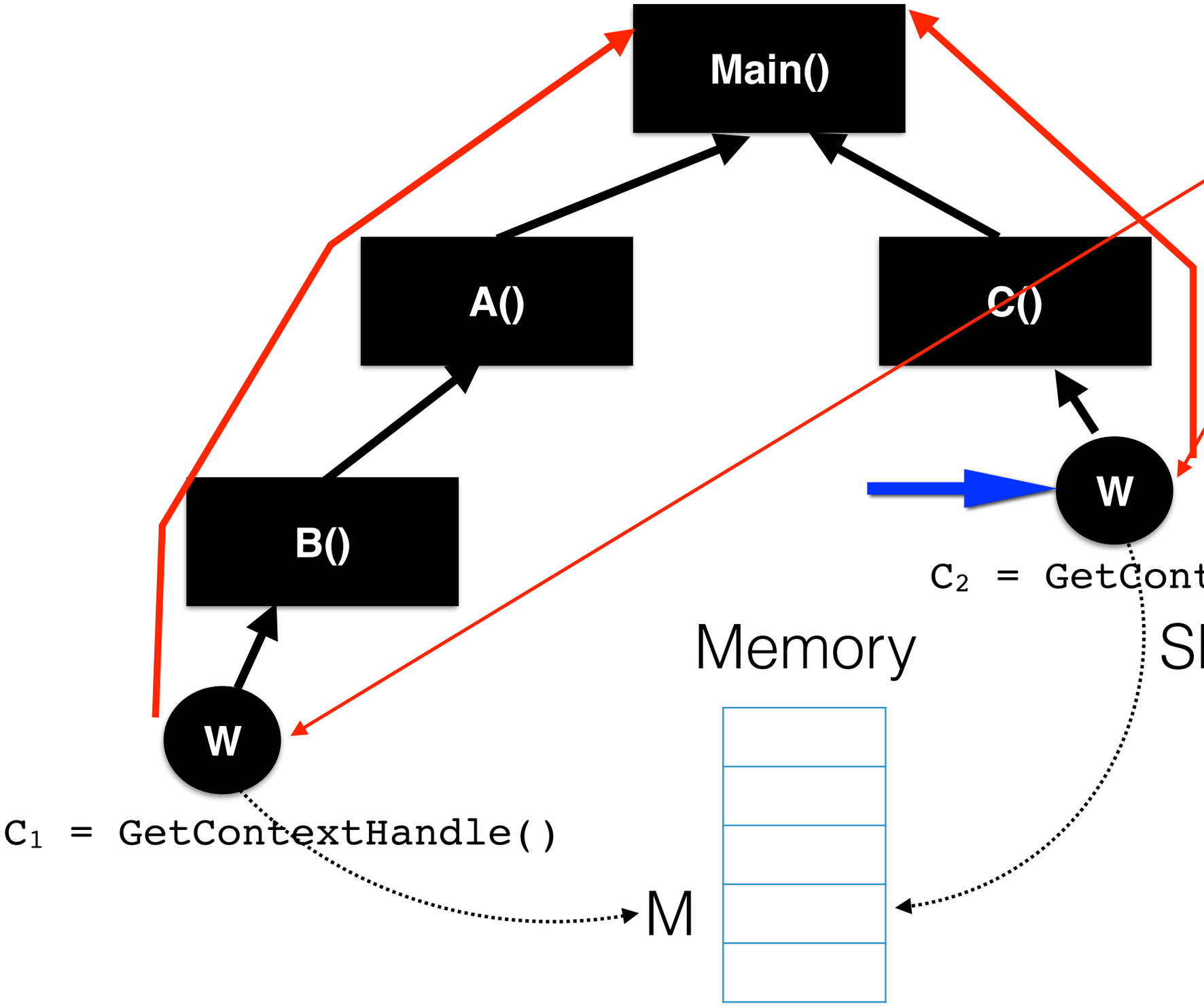
DeadSpy + CCTLib in Action



Dead writes

C_1:C_2	1234
C_3:C_{23}	959
C_2:C_{14}	546

DeadSpy + CCTLib in Action

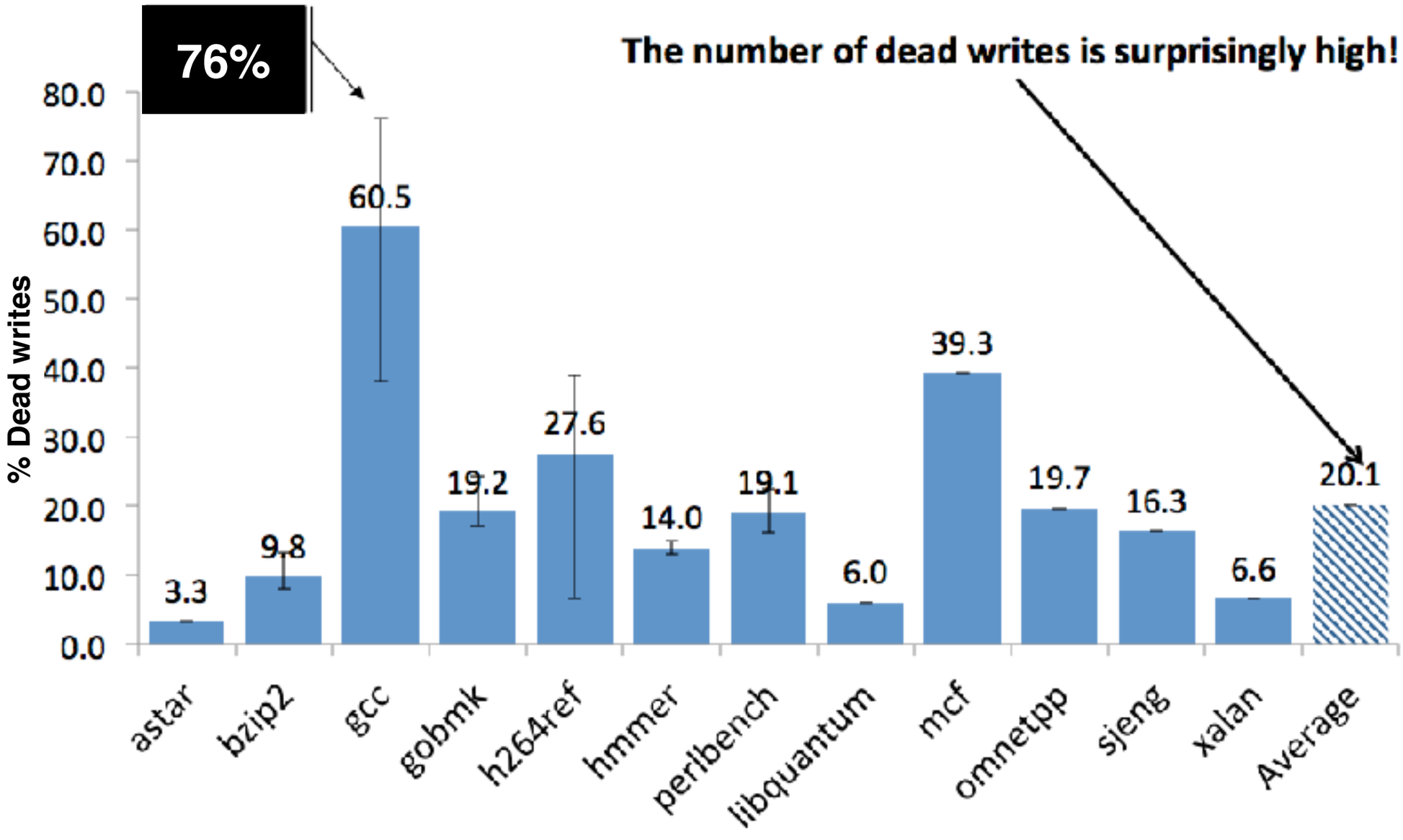


Dead writes

C_1:C_2	1234
C_3:C_{23}	959
C_2:C_{14}	546

Dead Writes in SPEC CPU 2006

Lower is better



Across compilers and optimization levels

HMMER: Lack of Design for Performance

Unoptimized

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      ...
      ic[k] = mpp[k] + tpmi[k];
      if ((sc = ip[k] + tpii[k]) > ic[k])
         ic[k] = sc;
```

-O3 optimized

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      ...
      R1= mpp[k] + tpmi[k];
      ic[k] = R1;
      if ((sc = ip[k] + tpii[k]) > R1)
         ic[k] = sc;
```

HMMER: Lack of Design for Performance

Unoptimized

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      ...
      ic[k] = mpp[k] + tpmi[k];
      if ((sc = ip[k] + tpii[k]) > ic[k])
         ic[k] = sc;
```

-O3 optimized

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      ...
      R1= mpp[k] + tpmi[k];
      ic[k] = R1;
      if ((sc = ip[k] + tpii[k]) > R1)
         ic[k] = sc;
      else
         ic[k] = R1;
```

HMMER: Lack of Design for Performance

Unoptimized

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      ...
      ic[k] = mpp[k] + tpmi[k];
      if ((sc = ip[k] + tpii[k]) > ic[k])
         ic[k] = sc;
```

Never Alias.
Declare as “restrict” pointers.
Can vectorize.

-O3 optimized

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      ...
      R1= mpp[k] + tpmi[k];
      ic[k] = R1;
      if ((sc = ip[k] + tpii[k]) > R1)
         ic[k] = sc;
   else
      ic[k] = R1;
```

HMMER: Lack of Design for Performance

Unoptimized

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      ...
      ic[k] = mpp[k] + tpmi[k];
      if ((sc = ip[k] + tpii[k]) > ic[k])
         ic[k] = sc;
```

-O3 optimized

```
for (i = 1; i <= L; i++) {
   for (k = 1; k <= M; k++) {
      ...
      R1= mpp[k] + tpmi[k];
      ic[k] = R1;
      if ((sc = ip[k] + tpii[k]) > R1)
         ic[k] = sc;
   else
      ic[k] = R1;
```

Never Alias.
Declare as “restrict” pointers.
Can vectorize.

> 16% running time improvement
> 40% with vectorization

Pinpointing Silent Store

A ***silent store*** is the one that does not change the system state.

Pinpointing Silent Store

A ***silent store*** is the one that does not change the system state.

```
/**  Func has no side-effect **/
for (int i = 0 ; i < N; i++) {
  A[i] = 2 * Func(i);
   ...  = A[i];
  A[i] = Func(i)+Func(i);
  ...  = A[i];
}
```

Pinpointing Silent Store

A ***silent store*** is the one that does not change the system state.

write same value

```
/**  Func has no side-effect **/
for (int i = 0 ; i < N; i++) {
  A[i] = 2 * Func(i);
   ...  = A[i];
  A[i] = Func(i)+Func(i);
   ...  = A[i];
}
```

Pinpointing Silent Store

A ***silent store*** is the one that does not change the system state.

```
/**  Func has no side-effect **/
for (int i = 0 ; i < N; i++) {
  A[i] = 2 * Func(i);
   ... = A[i];
  A[i] = Func(i)+Func(i);
  ... = A[i];
}
```

write same value

not dead write

use A[i]

Pinpointing Silent Store

A ***silent store*** is the one that does not change the system state.

```
/**  Func has no side-effect **/
for (int i = 0 ; i < N; i++) {
  A[i] = 2 * Func(i);
   ... = A[i];
  A[i] = Func(i)+Func(i);
  ... = A[i];
}
```

write same value

not dead write

use A[i]

not dead write

use A[i]

Pinpointing Silent Store

A ***silent store*** is the one that does not change the system state.

```
/**  Func has no side-effect **/
for (int i = 0 ; i < N; i++) {
  A[i] = 2 * Func(i);
  ... = A[i];
  A[i] = Func(i)+Func(i);
  ... = A[i];
}
```

write same value

not dead write

use A[i]

not dead write

use A[i]

DeadSpy and traditional value profiling cannot detect this redundancy

Value Agnostic vs. Value Aware

- DeadSpy: Value Agnostic
  - ✦ Does not inspect the value at a location; merely inspects the operation (read/write) on a location
- RedSpy: Value Aware
  - ✦ Inspects value produced by each operation

Value Agnostic vs. Value Aware

- DeadSpy: Value Agnostic
  - ✦ Does not inspect the value at a location; merely inspects the operation (read/write) on a location
- RedSpy: Value Aware
  - ✦ Inspects value produced by each operation

silent store

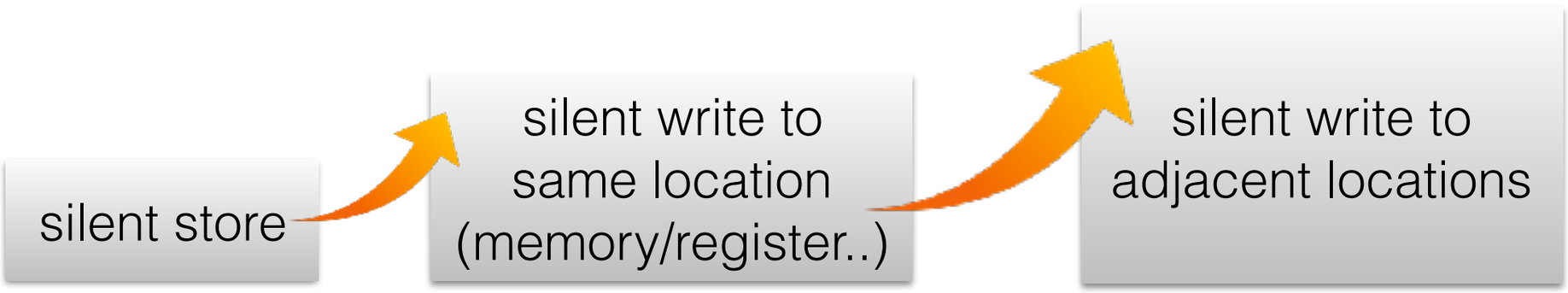
Value Agnostic vs. Value Aware

- DeadSpy: Value Agnostic
  - ✦ Does not inspect the value at a location; merely inspects the operation (read/write) on a location

- RedSpy: Value Aware
  - ✦ Inspects value produced by each operation

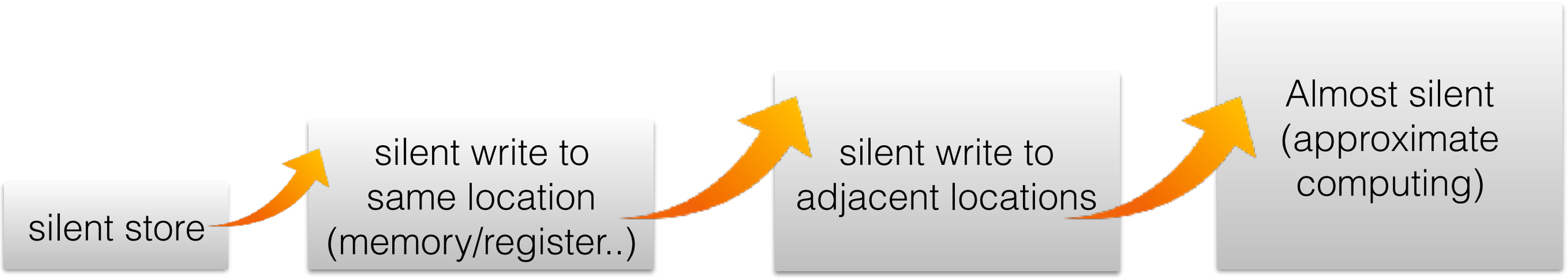
Value Agnostic vs. Value Aware

- DeadSpy: Value Agnostic
  - ✦ Does not inspect the value at a location; merely inspects the operation (read/write) on a location
- RedSpy: Value Aware
  - ✦ Inspects value produced by each operation



Value Agnostic vs. Value Aware

- DeadSpy: Value Agnostic
  - ✦ Does not inspect the value at a location; merely inspects the operation (read/write) on a location
- RedSpy: Value Aware
  - ✦ Inspects value produced by each operation



RedSpy: Value Locality

Value Locality implies producing the same value that is already present

Value Locality is often a symptom of some kinds of redundancy

RedSpy: Value Locality

Value Locality implies producing the same value that is already present

Value Locality is often a symptom of some kinds of redundancy

- Temporal value locality
  - The same value overwrites the same storage location
  - In memory or in register

- Spatial value locality
  - Nearby storage locations share a common value
  - Mainly in memory, big arrays

RedSpy: Value Locality

Value Locality implies producing the same value that is already present

Value Locality is often a symptom of some kinds of redundancy

- Temporal value locality
  - ✦ The same value overwrites the same storage location
  - ✦ In memory or in register

```
v1 = a + a;
… = v1;
v1 = a * 2;
```

- Spatial value locality
  - ✦ Nearby storage locations share a common value
  - ✦ Mainly in memory, big arrays

RedSpy: Value Locality

Value Locality implies producing the same value that is already present

Value Locality is often a symptom of some kinds of redundancy

- Temporal value locality
  - ✦ The same value overwrites the same storage location
  - ✦ In memory or in register

```
v1 = a + a;
… = v1;
v1 = a * 2;
```

- Spatial value locality
  - ✦ Nearby storage locations share a common value
  - ✦ Mainly in memory, big arrays

```
for(i=0; i<N; ++i){
    a[i]=i/2+1; // i is int
    b[i] = Func(a[i]);
}
```

RedSpy: Value Locality

Value Locality implies producing the same value that is already present

Value Locality is often a symptom of some kinds of redundancy

- Temporal value locality
  - ✦ The same value overwrites the same storage location
  - ✦ In memory or in register

```
v1 = a + a;
… = v1;
v1 = a * 2;
```

- Spatial value locality
  - ✦ Nearby storage locations share a common value
  - ✦ Mainly in memory, big arrays

```
for(i=0; i<N; ++i){
    a[i]=i/2+1; // i is int
    b[i] = Func(a[i]);
}
```

RedSpy: Value Locality

Approximately the same

Value Locality implies producing the same value that is already present

Value Locality is often a symptom of some kinds of redundancy

- Temporal value locality
  - ✦ The same value overwrites the same storage location
  - ✦ In memory or in register

```
v1 = a + a;
… = v1;
v1 = a * 2;
```

- Spatial value locality
  - ✦ Nearby storage locations share a common value
  - ✦ Mainly in memory, big arrays

```
for(i=0; i<N; ++i){
    a[i]=i/2+1; // i is int
    b[i] = Func(a[i]);
}
```

RedSpy: Value Locality

Approximately the same

Value Locality implies producing the same value that is already present

Value Locality is often a symptom of some kinds of redundancy

- Temporal value locality
  - ✦ The same value overwrites the same storage location
  - ✦ In memory or in register

```
v1 = a + a;
… = v1;
v1 = a * 2;
```

- Spatial value locality
  - ✦ Nearby storage locations share a common value
  - ✦ Mainly in memory, big arrays

```
for(i=0; i<N; ++i){
    a[i]=i/2+1; // i is int
    b[i] = Func(a[i]);
}
```

Exact & approximate

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write

target location ***t*** is written

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write

target location ***t*** is written

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write

target location ***t*** is written

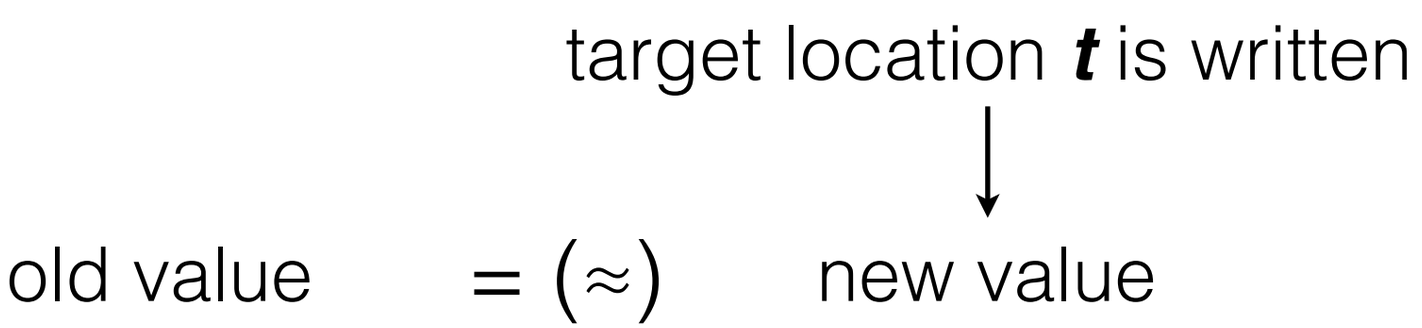
new value

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write



RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write

target location ***t*** is written

old value = (≈) new value

old operation

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write

target location ***t*** is written

old value = (≈) new value

old operation

current operation

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write

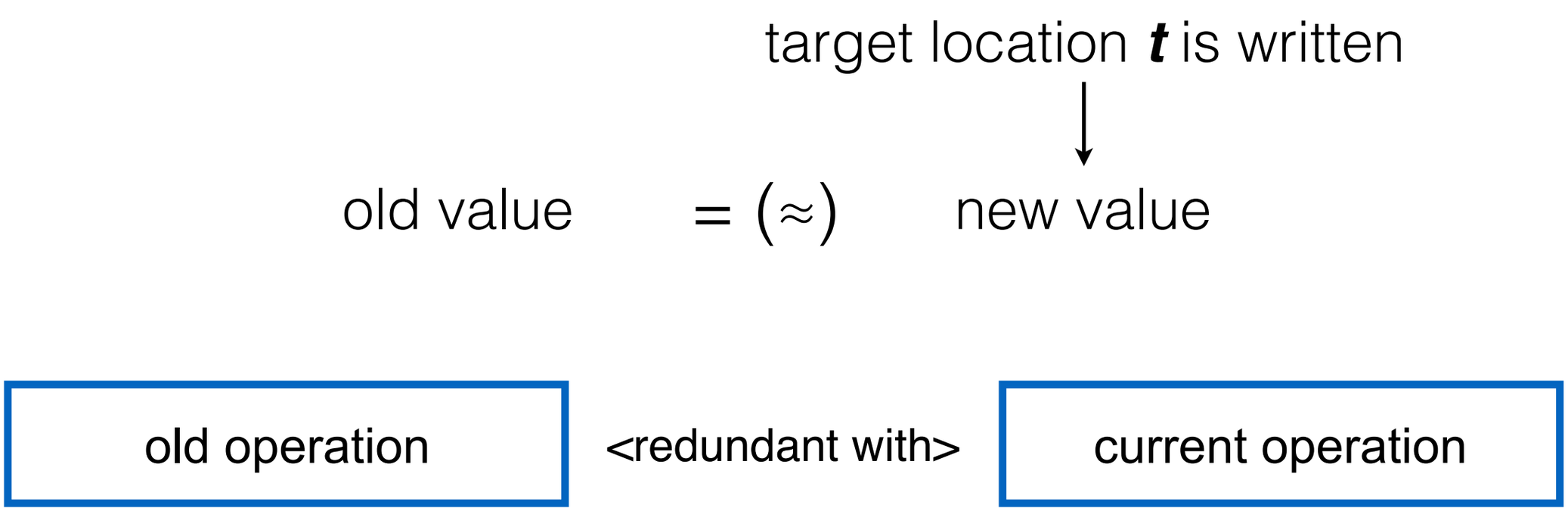
target location ***t*** is written

old value = (≈) new value

old operation <redundant with> current operation

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write



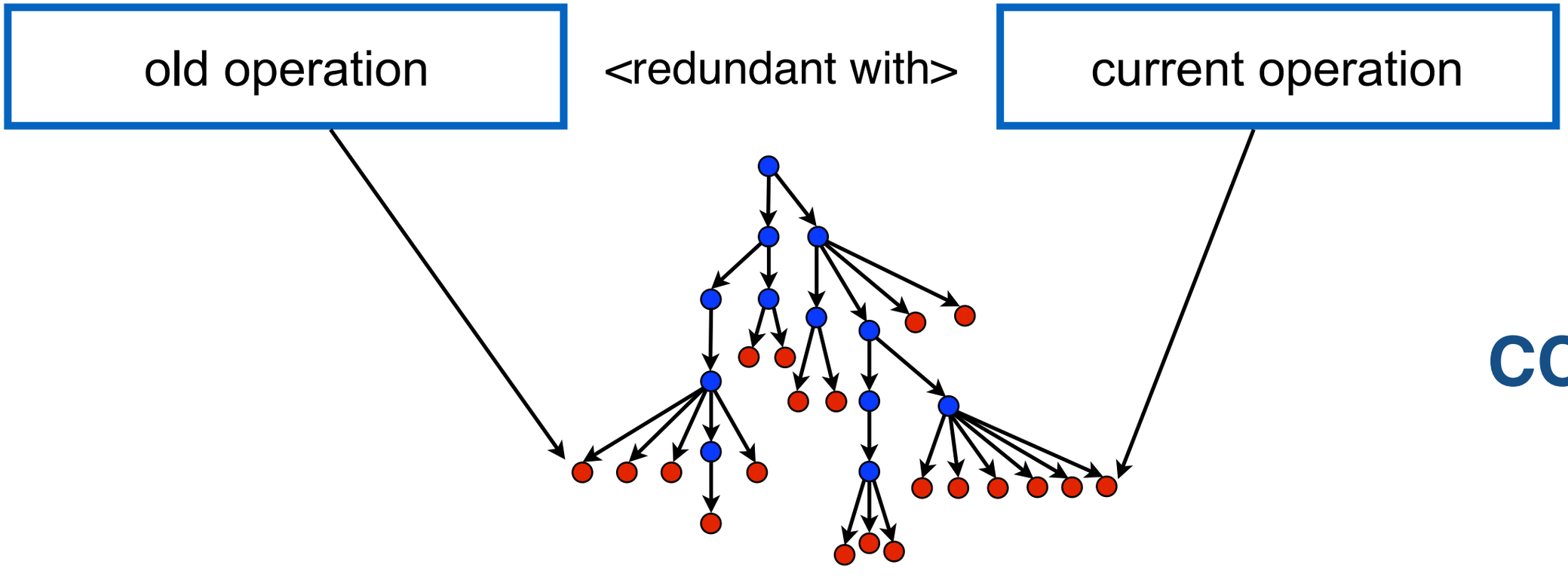
CCTLib

RedSpy: Value Locality Detection

- Temporal value locality (temporal redundancy)
  - ✦ Monitor memory write
  - ✦ Monitor register write

target location ***t*** is written

old value = (≈) new value



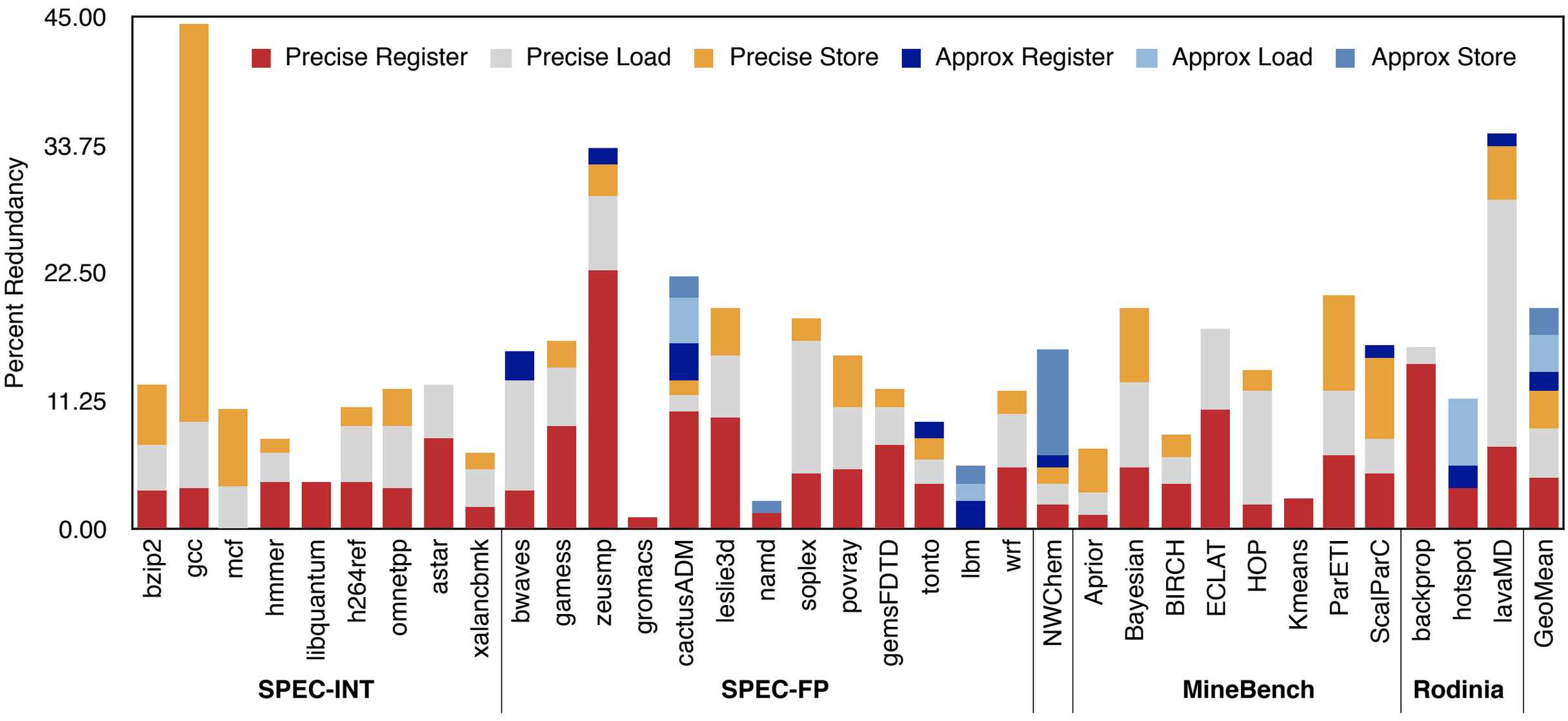
CCTLib

RedSpy: Spatial Value Locality

- User provides instrumentation points
  - Where? call predefined function
  - How? array, size, checking stride, approximation
- CCTLib scans while data structure and identifies the ratio of unique values to total elements

RedSpy: Experiments

- Temporal redundancy
  - ✦ GCC 4.8.5 -O3 PGO



GeoMean precise reg-reg = 4.46% GeoMean precise load = 4.45% GeoMean precise store = 3.13%

GeoMean approx reg-reg = 1.71% GeoMean approx load = 3.37% GeoMean approx store = 2.33%

Case Study: h264ref SPEC CPU2006

- Redundant writes to same location (temporal redundancy)
  - ✦ 13% loads and 13% stores are redundant

```
for (pos = 0; pos < max_pos; pos++) {
   ...
   if(...)  PelYline_11 = FastLine16Y_11;
   else  PelYline_11 = UMVLine16Y_11;

  for (blky = 0; blky < 4; blky++) {
for (y = 0; y < 4; y++) {
   refptr = PelYline_11(ref_pic, abs_y++, abs_x, img_height, img_width);
   ... } ... } ...}
```

Case Study: h264ref SPEC CPU2006

- Redundant writes to same location (temporal redundancy)
  - ✦ 13% loads and 13% stores are redundant

```
for (pos = 0; pos < max_pos; pos++) {
   ...
   if(...)  PelYline_11 = FastLine16Y_11;
   else  PelYline_11 = UMVLine16Y_11;

  for (blky = 0; blky < 4; blky++) {
for (y = 0; y < 4; y++) {
   refptr = PelYline_11(ref_pic, abs_y++, abs_x, img_height, img_width);
   ... } ... } ...}
```

parameters seldom change

Case Study: h264ref SPEC CPU2006

- Redundant writes to same location (temporal redundancy)
  - ✦ 13% loads and 13% stores are redundant

```
for (pos = 0; pos < max_pos; pos++) {
   ...
   if(...)  PelYline_11 = FastLine16Y_11;
   else  PelYline_11 = UMVLine16Y_11;
```

```
for (blky = 0; blky < 4; blky++) {
   for (y = 0; y < 4; y++) {
      refptr = PelYline_11(ref_pic, abs_y++, abs_x, img_height, img_width);
      ... } ... } ...}
```

push same value

parameters seldom change

Case Study: h264ref SPEC CPU2006

- Redundant writes to same location (temporal redundancy)
  - ✦ 13% loads and 13% stores are redundant

```
for (pos = 0; pos < max_pos; pos++) {
   ...
   if(...)  PelYline_11 = FastLine16Y_11;
   else  PelYline_11 = UMVLine16Y_11;
```

Prevent from “inline”

```
for (blky = 0; blky < 4; blky++) {
    for (y = 0; y < 4; y++) {
       refptr = PelYline_11(ref_pic, abs_y++, abs_x, img_height, img_width);
       ... } ... } ...}
```

push same value

parameters seldom change

Case Study: h264ref SPEC CPU2006

- Redundant writes to same location (temporal redundancy)
  - 13% loads and 13% stores are redundant

```
for (pos = 0; pos < max_pos; pos++) {
   ...
   if(...)  PelYline_11 = FastLine16Y_11;
   else  PelYline_11 = UMVLine16Y_11;
```

Prevent from “inline”

```
for (blky = 0; blky < 4; blky++) {
   for (y = 0; y < 4; y++) {
      refptr = PelYline_11(ref_pic, abs_y++, abs_x, img_height, img_width);
      ... } ... } ...}
```

push same value

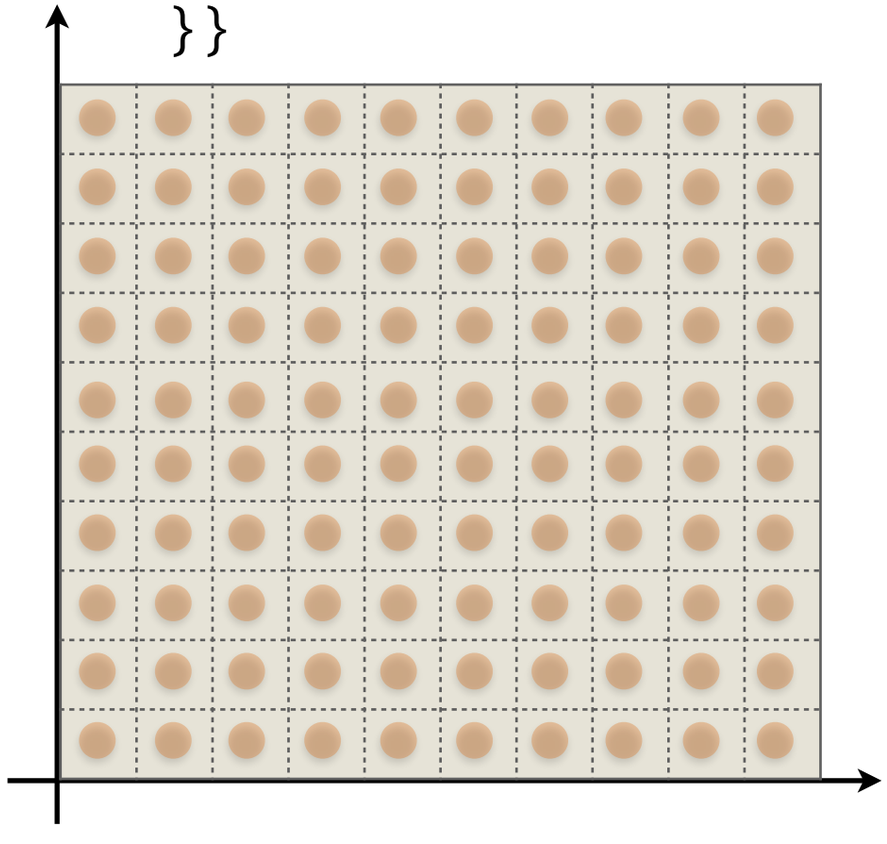
parameters seldom change

- Optimization
  - Inline the two functions
  - 1.34x speedup; 23% energy saving

Case Study: Rodinia hotspot

- Approximately same values
  - ✦ Elements in array `result` are similar (<1%)

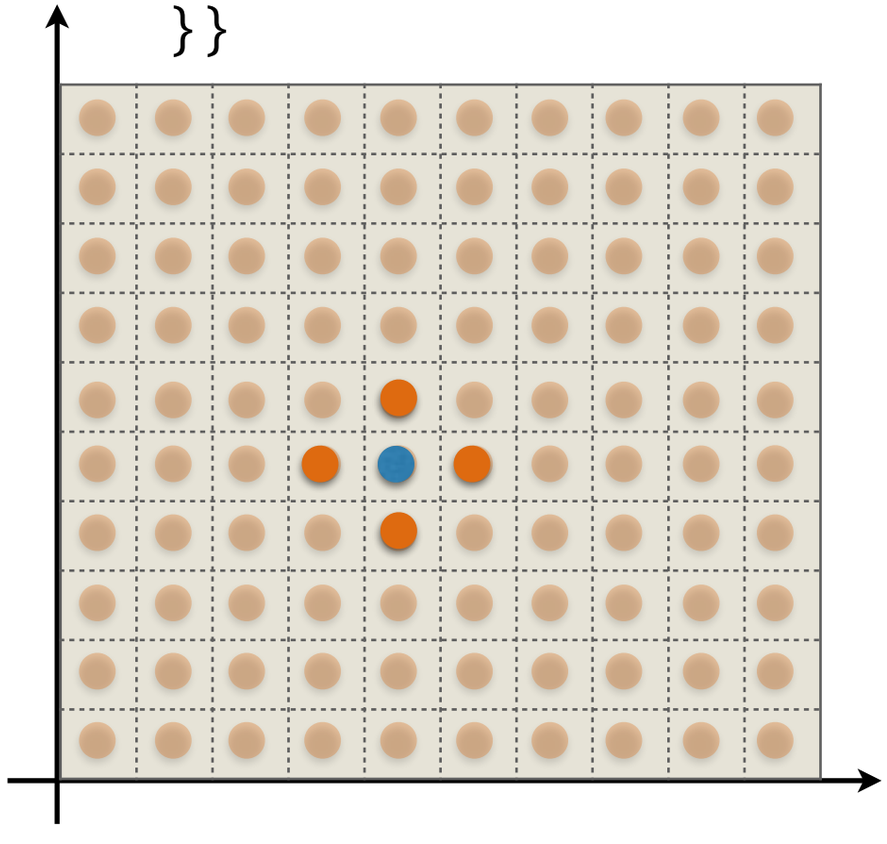
```
for (r = 0; r < row; r++) {
  for (c = 0; c < col; c++) {
  ...{
       delta = (step / Cap)*(power[r*col+c] + (temp[(r+1)*col+c]+temp[(r-1)*col +c]
       -2.0*temp[r*col+c]) / Ry + (temp[r*col+c+1]+ temp[r*col+c-1] -
       2.0*temp[r*col+c]) / Rx + (amb_temp - temp[r*col+c]) / Rz);
     }
       result[r*col+c] =temp[r*col+c]+ delta;
  } }
```



Case Study: Rodinia hotspot

- Approximately same values
  - ✦ Elements in array `result` are similar (<1%)

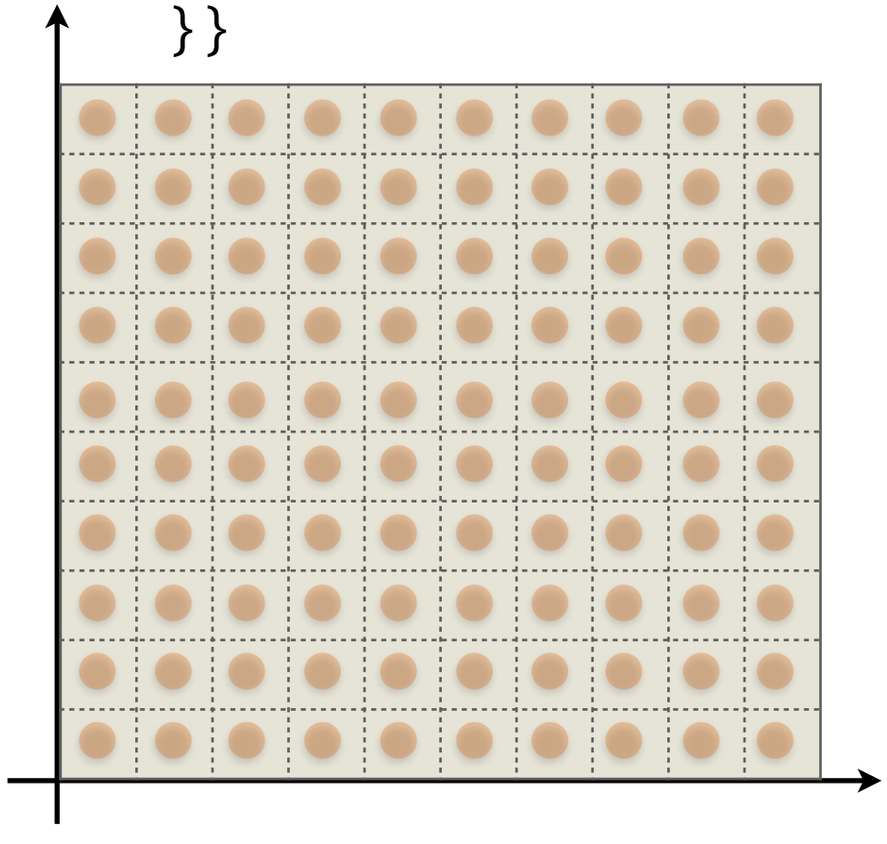
```
for (r = 0; r < row; r++) {
  for (c = 0; c < col; c++) {
  ...{
       delta = (step / Cap)*(power[r*col+c] + (temp[(r+1)*col+c]+temp[(r-1)*col +c]
       -2.0*temp[r*col+c]) / Ry + (temp[r*col+c+1]+ temp[r*col+c-1] -
       2.0*temp[r*col+c]) / Rx + (amb_temp - temp[r*col+c]) / Rz);
     }
       result[r*col+c] =temp[r*col+c]+ delta;
} }
```



Case Study: Rodinia hotspot

- Approximately same values
  - ✦ Elements in array `result` are similar (<1%)

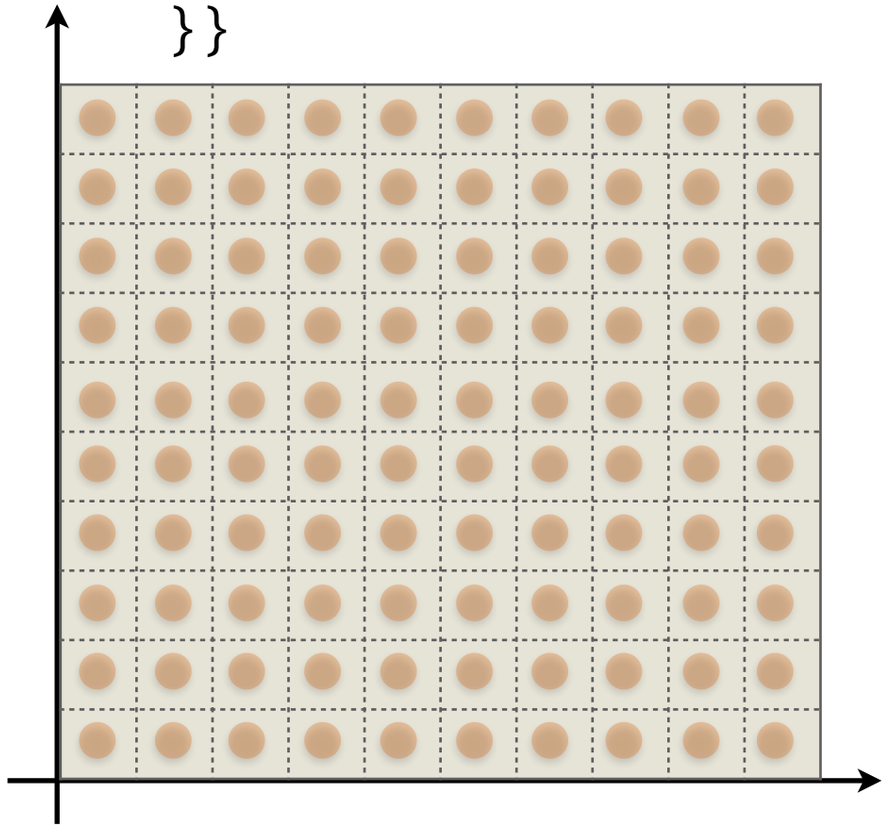
```
for (r = 0; r < row; r++) {
  for (c = 0; c < col; c++) {
  ...{
       delta = (step / Cap)*(power[r*col+c] + (temp[(r+1)*col+c]+temp[(r-1)*col +c]
       -2.0*temp[r*col+c]) / Ry + (temp[r*col+c+1]+ temp[r*col+c-1] -
       2.0*temp[r*col+c]) / Rx + (amb_temp - temp[r*col+c]) / Rz);
     }
       result[r*col+c] =temp[r*col+c]+ delta;
  } }
```



Case Study: Rodinia hotspot

- Approximately same values
  - Elements in array `result` are similar (<1%)

```
for (r = 0; r < row; r++) {
  for (c = 0; c < col; c++) {
  ...{
       delta = (step / Cap)*(power[r*col+c] + (temp[(r+1)*col+c]+temp[(r-1)*col +c]
       -2.0*temp[r*col+c]) / Ry + (temp[r*col+c+1]+ temp[r*col+c-1] -
       2.0*temp[r*col+c]) / Rx + (amb_temp - temp[r*col+c]) / Rz);
     }
     result[r*col+c] =temp[r*col+c]+ delta;
} }
```

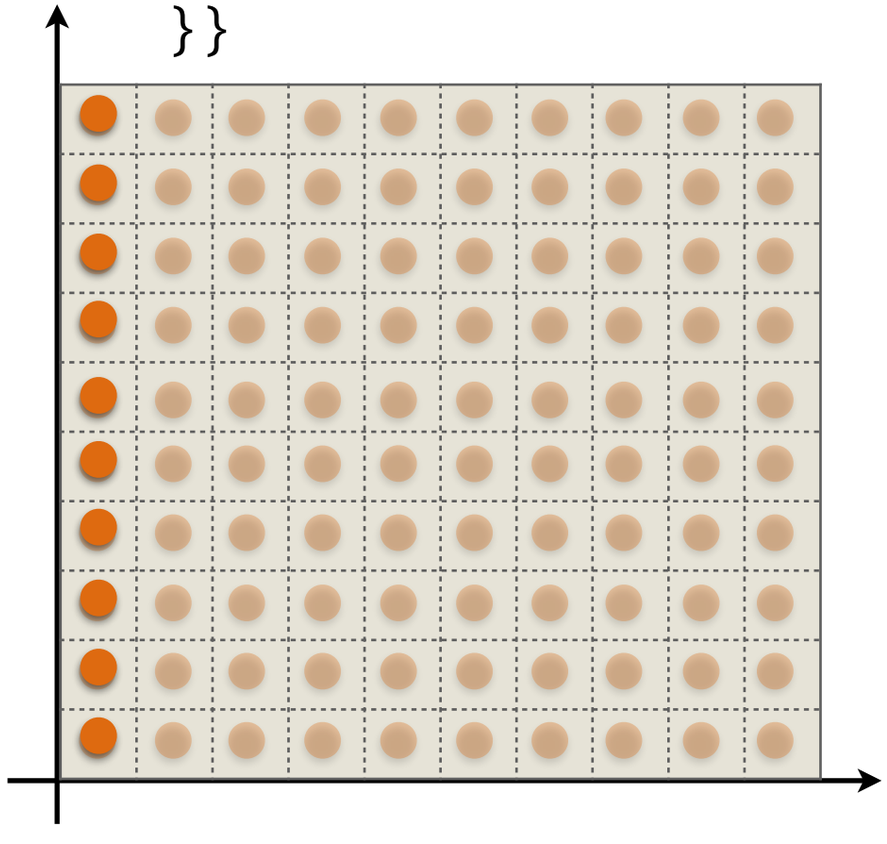


- Optimization
  - calculate the first and middle column
  - 2.21x speedup; 70% power saving
  - mean relative error: <0.6%

Case Study: Rodinia hotspot

- Approximately same values
  - Elements in array `result` are similar (<1%)

```
for (r = 0; r < row; r++) {
  for (c = 0; c < col; c++) {
  ...{
       delta = (step / Cap)*(power[r*col+c] + (temp[(r+1)*col+c]+temp[(r-1)*col +c]
       -2.0*temp[r*col+c]) / Ry + (temp[r*col+c+1]+ temp[r*col+c-1] -
       2.0*temp[r*col+c]) / Rx + (amb_temp - temp[r*col+c]) / Rz);
     }
     result[r*col+c] =temp[r*col+c]+ delta;
} }
```

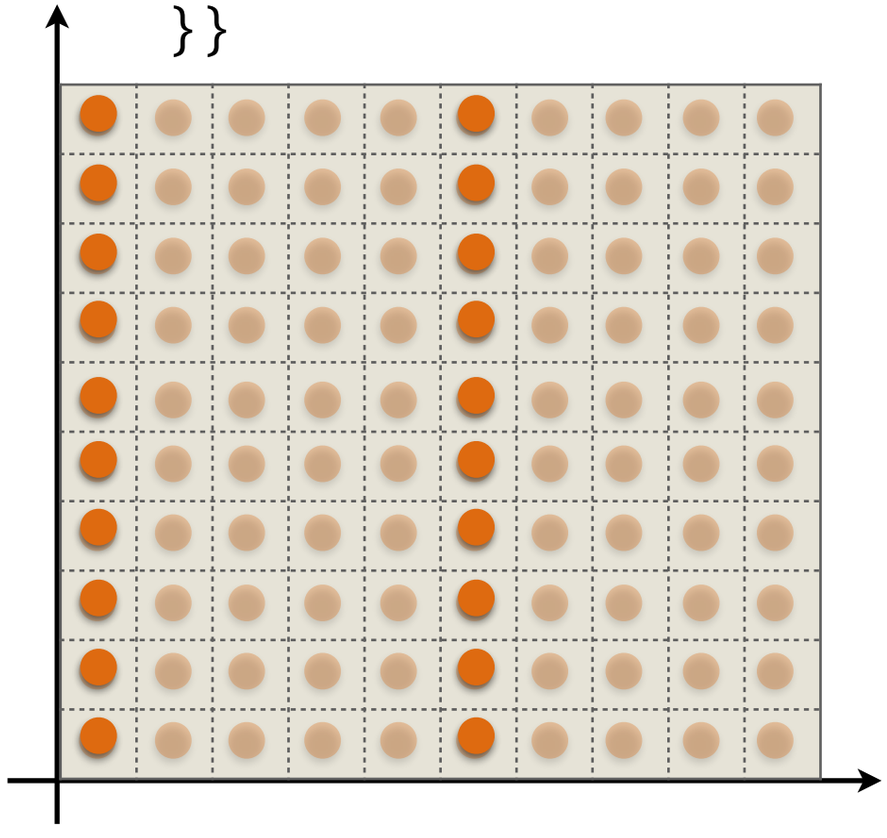


- Optimization
  - calculate the first and middle column
  - 2.21x speedup; 70% power saving
  - mean relative error: <0.6%

Case Study: Rodinia hotspot

- Approximately same values
  - Elements in array `result` are similar (<1%)

```
for (r = 0; r < row; r++) {
  for (c = 0; c < col; c++) {
  ...{
       delta = (step / Cap)*(power[r*col+c] + (temp[(r+1)*col+c]+temp[(r-1)*col +c]
       -2.0*temp[r*col+c]) / Ry + (temp[r*col+c+1]+ temp[r*col+c-1] -
       2.0*temp[r*col+c]) / Rx + (amb_temp - temp[r*col+c]) / Rz);
     }
     result[r*col+c] =temp[r*col+c]+ delta;
} }
```

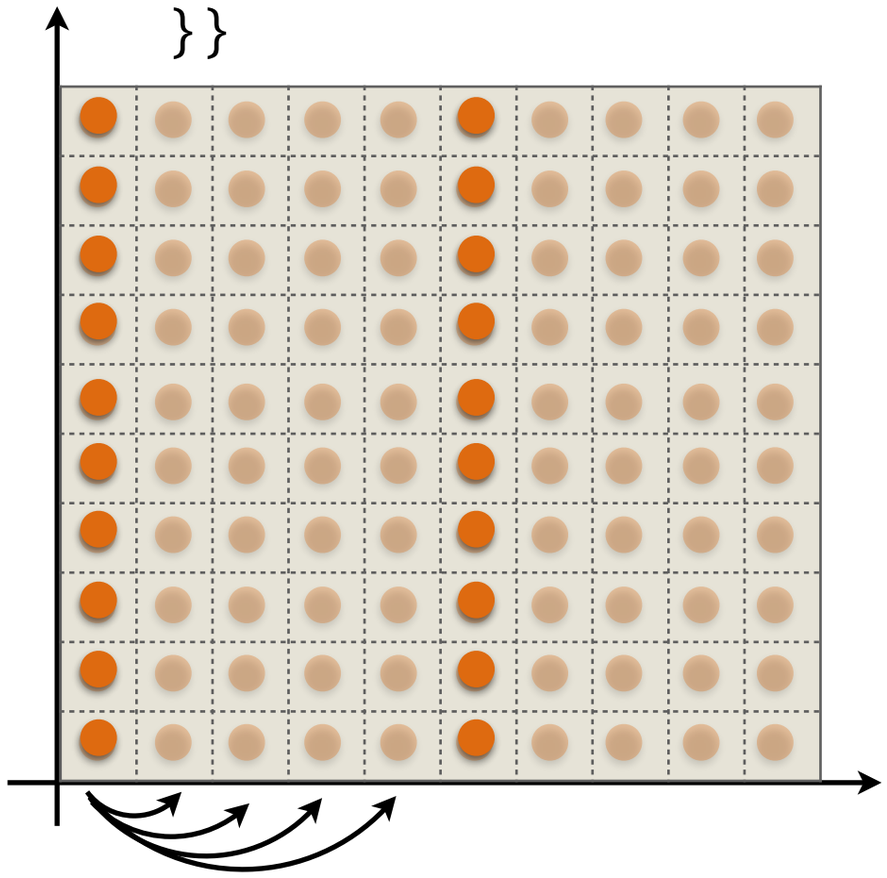


- Optimization
  - calculate the first and middle column
  - 2.21x speedup; 70% power saving
  - mean relative error: <0.6%

Case Study: Rodinia hotspot

- Approximately same values
  - Elements in array `result` are similar (<1%)

```
for (r = 0; r < row; r++) {
  for (c = 0; c < col; c++) {
  ...{
       delta = (step / Cap)*(power[r*col+c] + (temp[(r+1)*col+c]+temp[(r-1)*col +c]
       -2.0*temp[r*col+c]) / Ry + (temp[r*col+c+1]+ temp[r*col+c-1] -
       2.0*temp[r*col+c]) / Rx + (amb_temp - temp[r*col+c]) / Rz);
     }
     result[r*col+c] =temp[r*col+c]+ delta;
} }
```

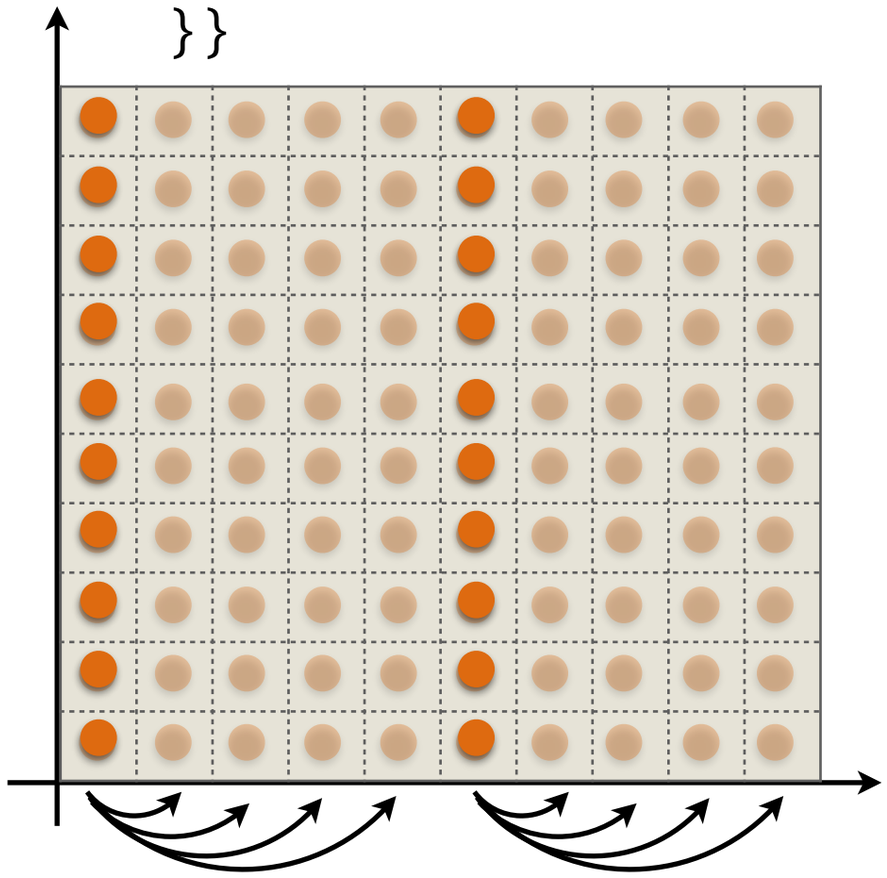


- Optimization
  - calculate the first and middle column
  - 2.21x speedup; 70% power saving
  - mean relative error: <0.6%

Case Study: Rodinia hotspot

- Approximately same values
  - Elements in array `result` are similar (<1%)

```
for (r = 0; r < row; r++) {
  for (c = 0; c < col; c++) {
  ...{
       delta = (step / Cap)*(power[r*col+c] + (temp[(r+1)*col+c]+temp[(r-1)*col +c]
       -2.0*temp[r*col+c]) / Ry + (temp[r*col+c+1]+ temp[r*col+c-1] -
       2.0*temp[r*col+c]) / Rx + (amb_temp - temp[r*col+c]) / Rz);
     }
     result[r*col+c] =temp[r*col+c]+ delta;
} }
```



- Optimization
  - calculate the first and middle column
  - 2.21x speedup; 70% power saving
  - mean relative error: <0.6%

Summary

- Production programs suffer from myriad inefficiencies in software
  - ✦ Compilers and traditional tools are insufficient
- Fine-grained monitoring tools are necessary for identifying several kinds of program inefficiencies
  - ✦ Fine-grained tools can provide semantic information for developer productivity
- CCTLib provides efficient calling context collection for production workloads at moderate overhead
- CCTLib is open source: https://github.com/CCTLib/cctlib
- Pin tools built with CCTLib pinpoint software inefficiencies and offer new venues to tuning