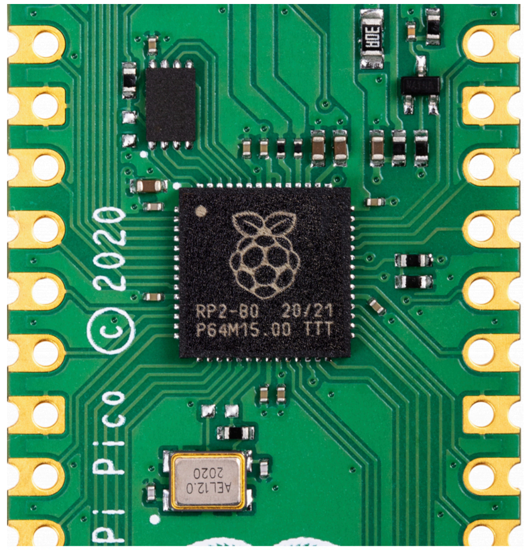
NC STATE

DrCCTProf: Supporting Fine-Grained Call Path Profiling on ARM and X86

Xu Liu
NC State University

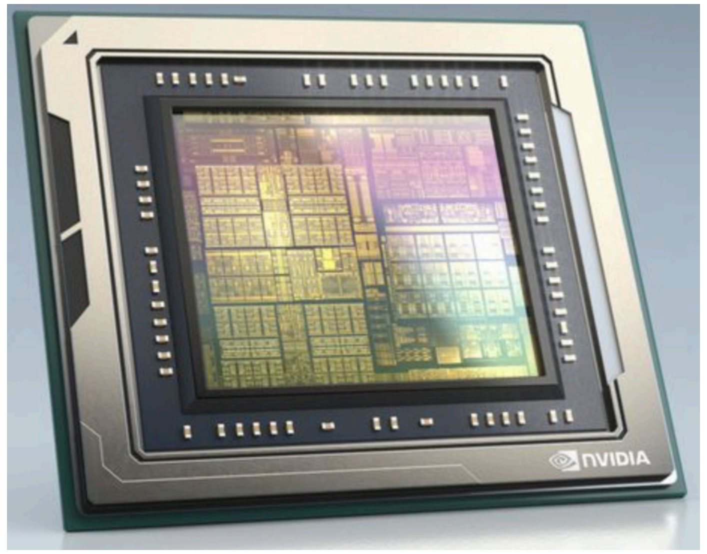
ARM is Popular

ARM for Smartphones



ARM for SoC

ARM for PC (laptop, tablet)



ARM for High Performance Computing

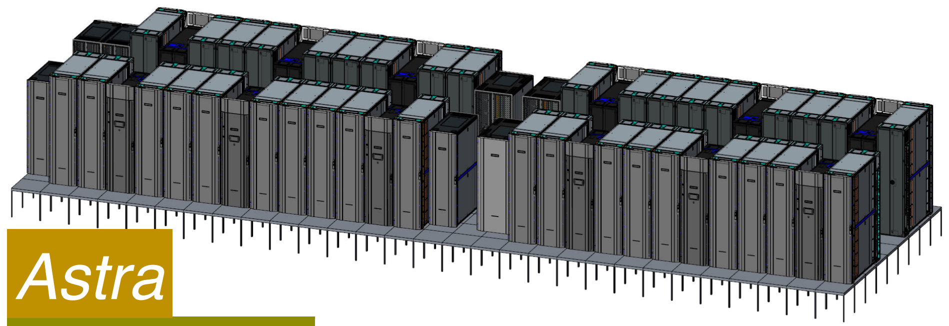
ARM Cloud Platform

Graviton
processors

Kunpeng
processors

Marvell ThunderX2
processors

ARM Supercomputer



Astra
Aug 2018
Consists of 145152 ARM processors

Fugaku
June 2020
No.1 Supercomputer

Tianhe-3E
Before 2021
The exascale supercomputer

ARM Ecosystem

Example End Users

Sandia National Laboratories

THE UNIVERSITY OF UTAH

Hartree Centre
Science & Technology Facilities Council

PayPal

ARM

Key Applications Middleware

MySQL

APACHE HTTP SERVER

hadoop

MariaDB

Couchbase

ceph

OpenJDK

JAVA

openstack

MEMCACHED

NGINX

mongoDB

GlusterFS

Operating System, Virtualization & Firmware

efi

ACPI

openSUSE

KVM

Xen Project

fedora

redhat

ubuntu
Supported by Canonical

debian

OEMs and ODMs

CRAY
THE SUPERCOMPUTER COMPANY

MITAC
MITAC INTERNATIONAL CORP.

DELL

FOXCONN

Inventec

STACK VELOCITY

GIGABYTE

wiwynn

hp

hyve solutions

lenovo

PEGATRON

ASUS

ARM SoC

AMD

annapurna labs

apm applied micro

BROADCOM

CAVIUM

MARVELL

QUALCOMM

TEXAS INSTRUMENTS

https://www.nextplatform.com/2015/10/06/why-are-we-still-waiting-for-arm-servers/

ARM Ecosystem

Demand **performance tools** on **various** systems

https://www.nextplatform.com/2015/10/06/why-are-we-still-waiting-for-arm-servers/

Mainstream Profilers

Popular profilers

MAP / Perf / TAU / Scalasca / HPCToolkit / VTune / …

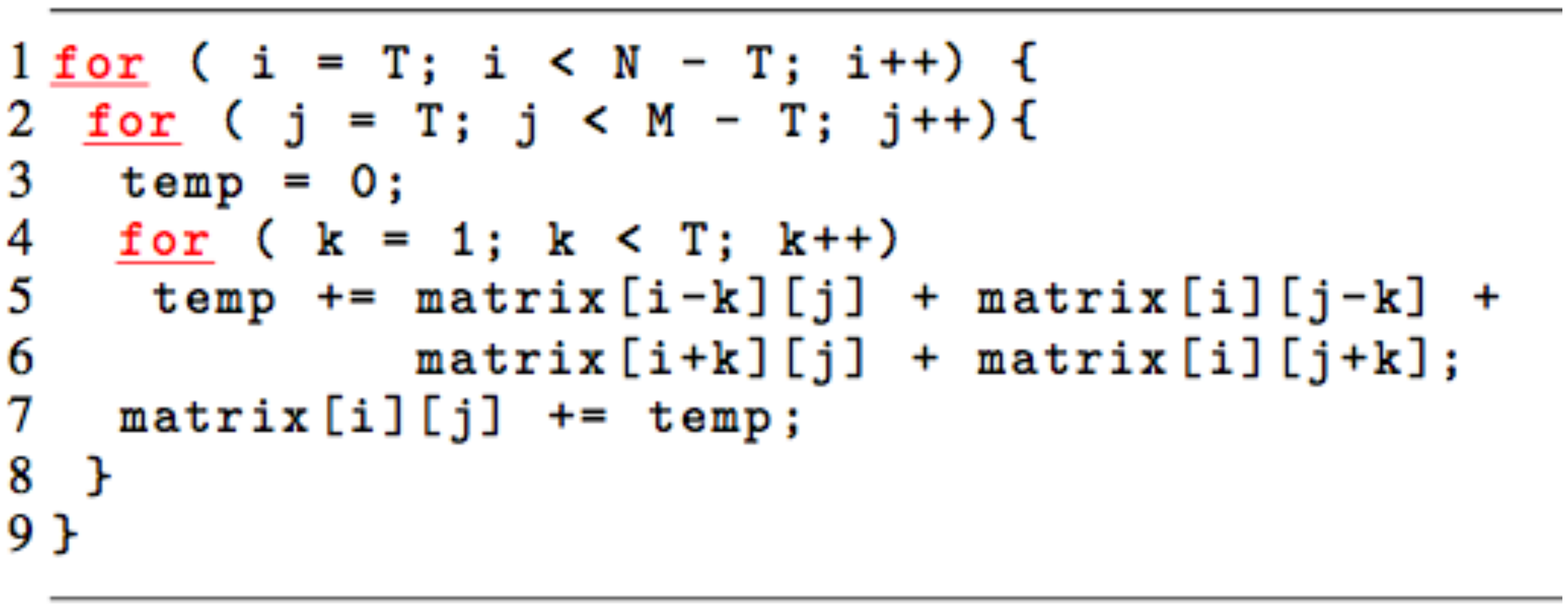
- Hotspot analysis: high resource utilization
- Various hardware metrics: cache misses, instructions per cycle, …

However,

They cannot tell if resources were **“well spent”**

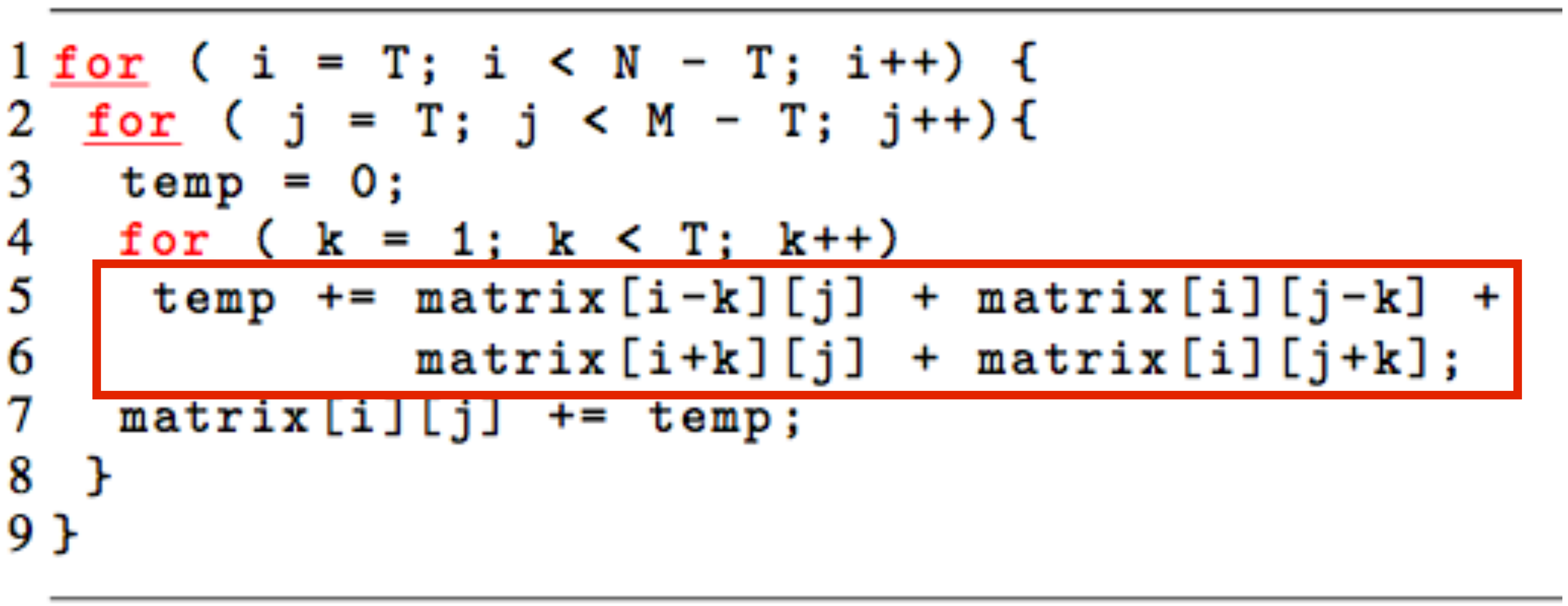
Hotspots may be symptoms rather than root causes

A Motivating Example



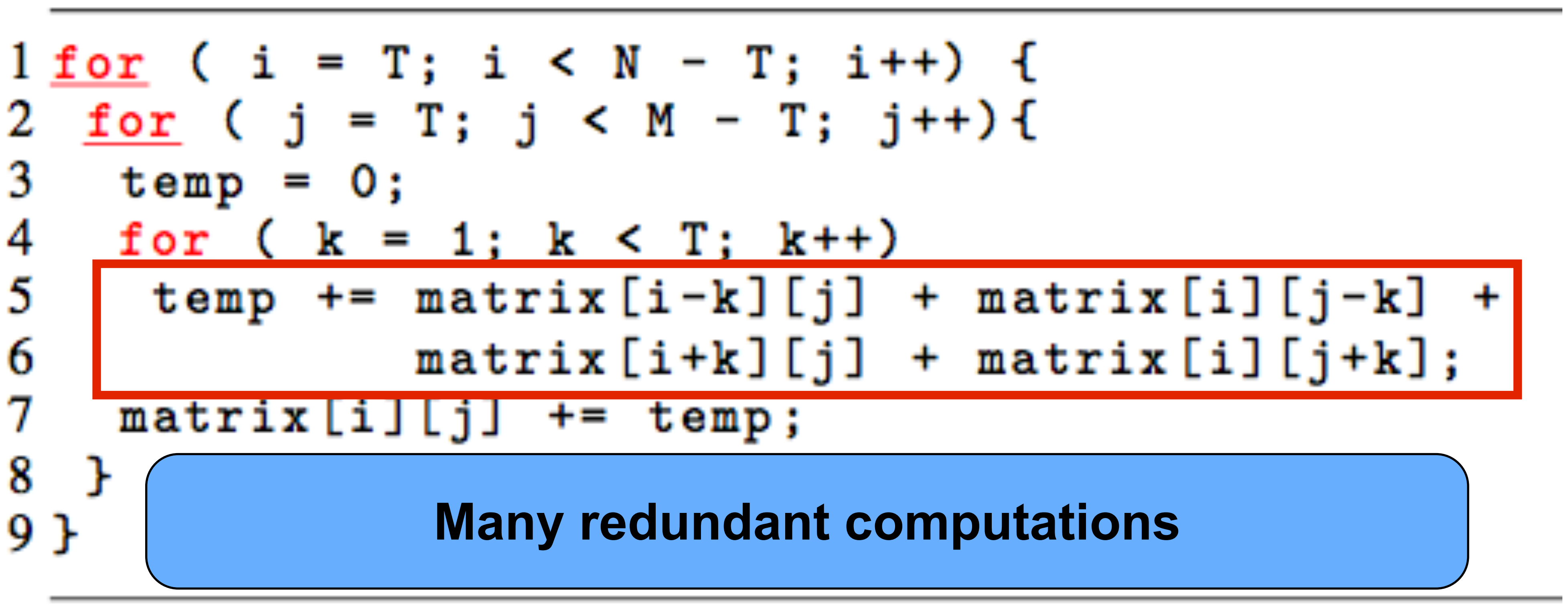
```
for ( i = T; i < N - T; i++) {
 for ( j = T; j < M - T; j++){
  temp = 0;
  for ( k = 1; k < T; k++)
   temp += matrix[i-k][j] + matrix[i][j-k] +
           matrix[i+k][j] + matrix[i][j+k];
  matrix[i][j] += temp;
 }
}
```

A Motivating Example



```
for ( i = T; i < N - T; i++) {
 for ( j = T; j < M - T; j++){
  temp = 0;
  for ( k = 1; k < T; k++)
   temp += matrix[i-k][j] + matrix[i][j-k] +
           matrix[i+k][j] + matrix[i][j+k];
  matrix[i][j] += temp;
 }
}
```

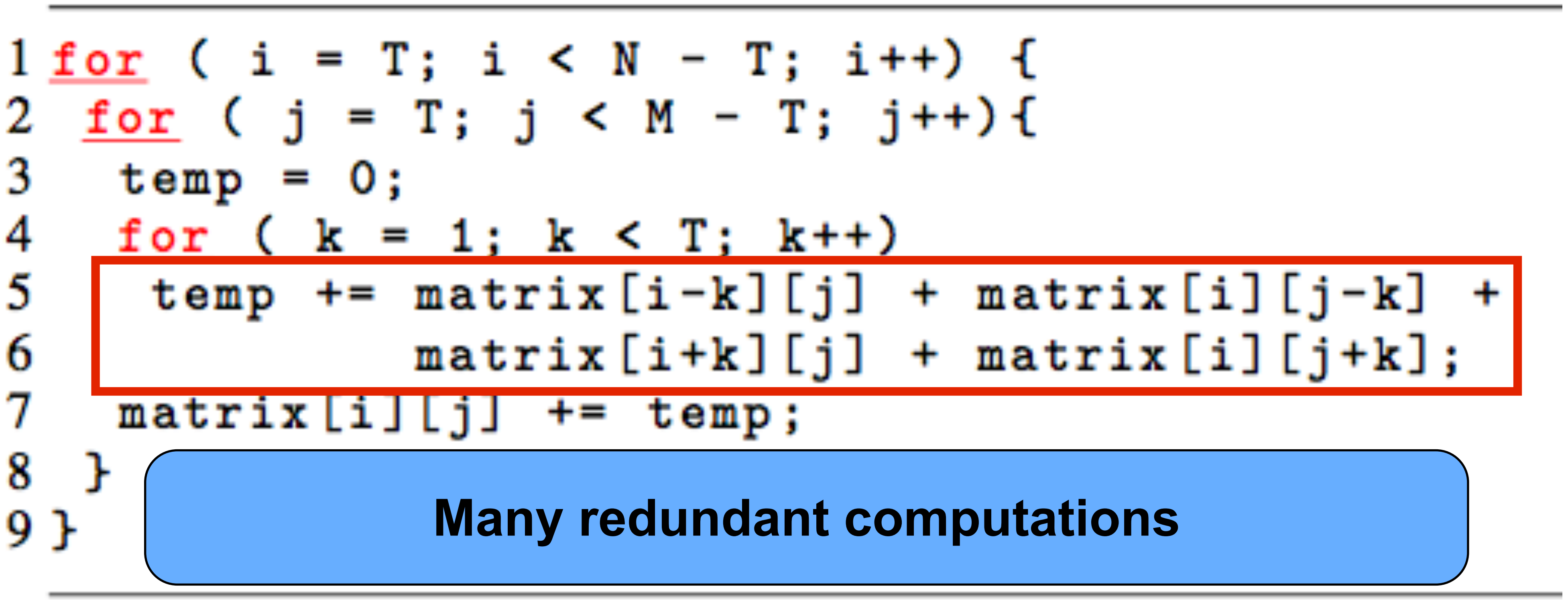
A Motivating Example



```
for ( i = T; i < N - T; i++) {
 for ( j = T; j < M - T; j++){
  temp = 0;
  for ( k = 1; k < T; k++)
   temp += matrix[i-k][j] + matrix[i][j-k] +
           matrix[i+k][j] + matrix[i][j+k];
  matrix[i][j] += temp;
 }
}
```

Many redundant computations

A Motivating Example

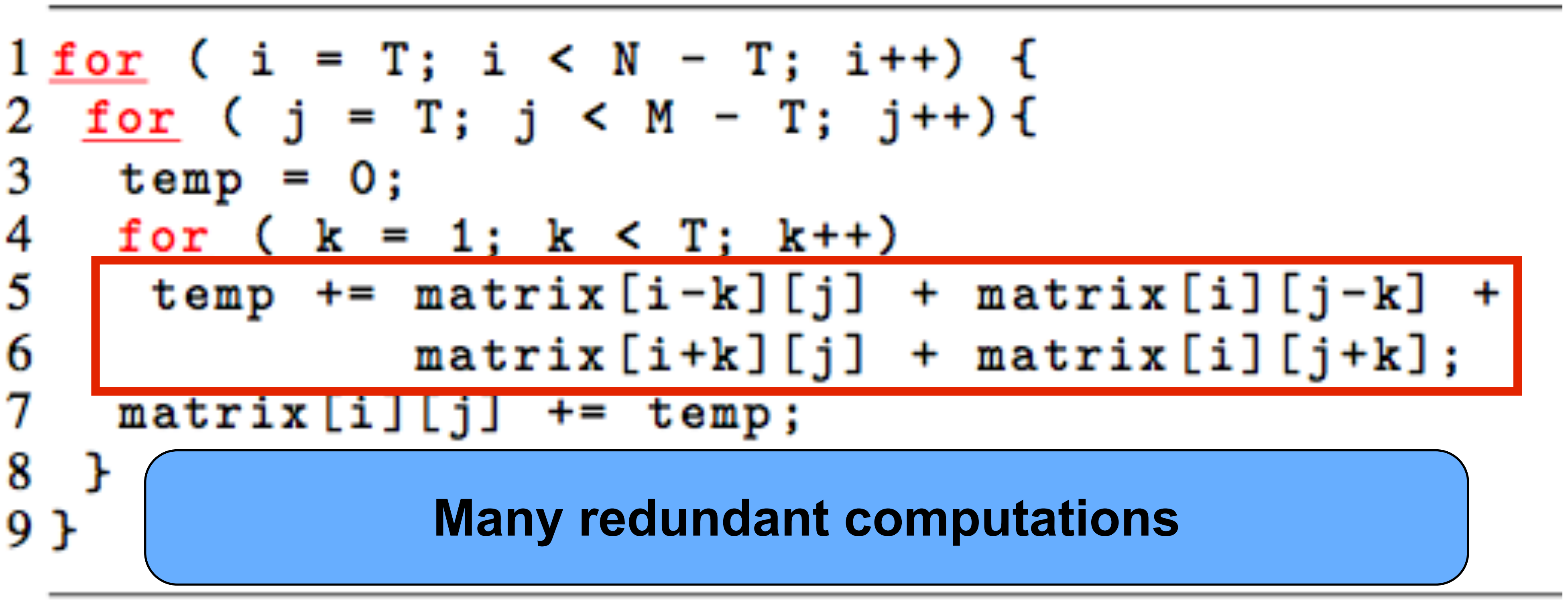


```
for ( i = T; i < N - T; i++) {
 for ( j = T; j < M - T; j++){
  temp = 0;
  for ( k = 1; k < T; k++)
   temp += matrix[i-k][j] + matrix[i][j-k] +
           matrix[i+k][j] + matrix[i][j+k];
  matrix[i][j] += temp;
 }
}
```

Many redundant computations

metrics	original	optimized	&redunction
#instructions	$73*10^9$	$40*10^9$	0.46
#cycles	$33*10^9$	$21*10^9$	0.36
IPC	2.2	1.9	

A Motivating Example

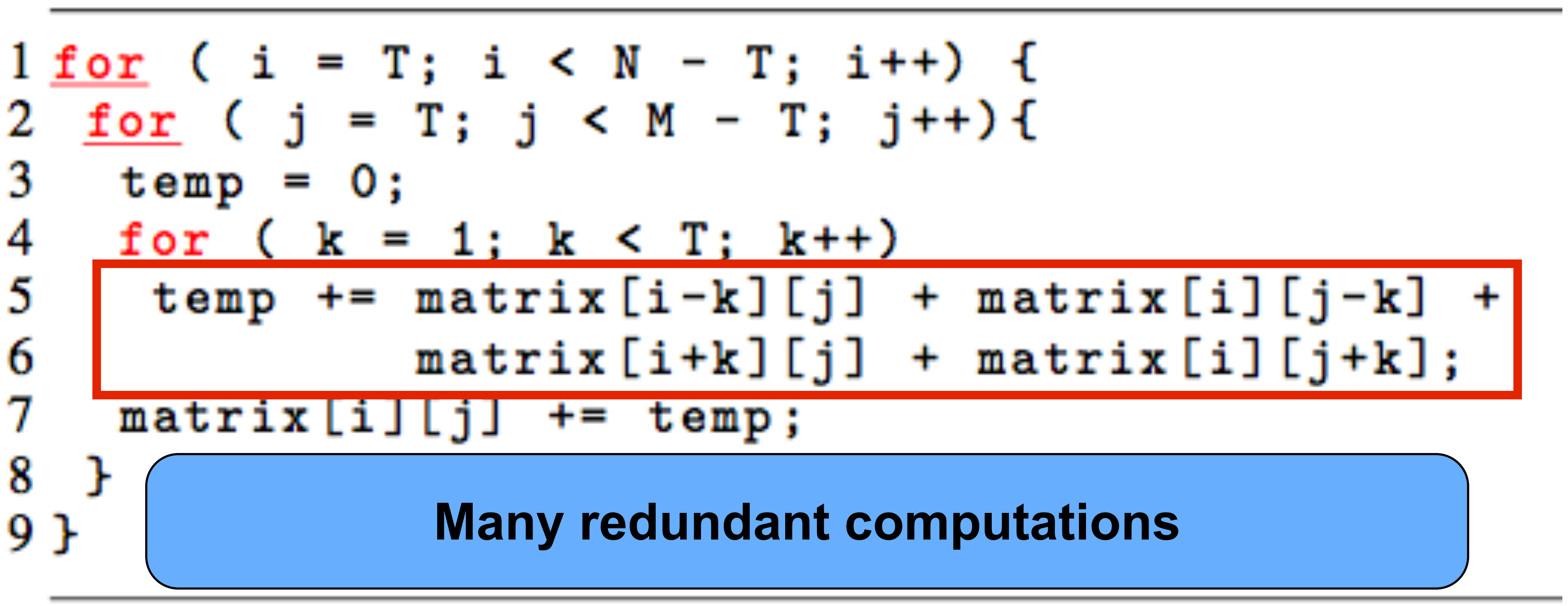


```
for ( i = T; i < N - T; i++) {
 for ( j = T; j < M - T; j++){
  temp = 0;
  for ( k = 1; k < T; k++)
   temp += matrix[i-k][j] + matrix[i][j-k] +
           matrix[i+k][j] + matrix[i][j+k];
  matrix[i][j] += temp;
 }
}
```

Many redundant computations

metrics	original	optimized	&redunction
#instructions	$73*10^9$	$40*10^9$	0.46
#cycles	$33*10^9$	$21*10^9$	0.36
IPC	2.2	1.9	

A Motivating Example



```
for ( i = T; i < N - T; i++) {
 for ( j = T; j < M - T; j++){
  temp = 0;
  for ( k = 1; k < T; k++)
   temp += matrix[i-k][j] + matrix[i][j-k] +
           matrix[i+k][j] + matrix[i][j+k];
  matrix[i][j] += temp;
 }
}
```

Many redundant computations

metrics	original	optimized	&redunction
#instructions	$73*10^9$	$40*10^9$	0.46
IPC	2.2	1.9	

Need techniques to shift from hotspot analysis to

wastage analysis

Wastage Analysis

Wasted memory accesses

- Redundant memory accesses
    - Redundant memory accesses: the same values involved
- Useless memory accesses
    - Dead stores: stored value got overwritten without use

Wasted arithmetic computation

- Symbolic equivalent computation
- Result equivalent computation

HMMER: An 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: An 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: An 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: An 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

HMMER: From the Binary Perspective

```
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: From the Binary Perspective

```
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;
```

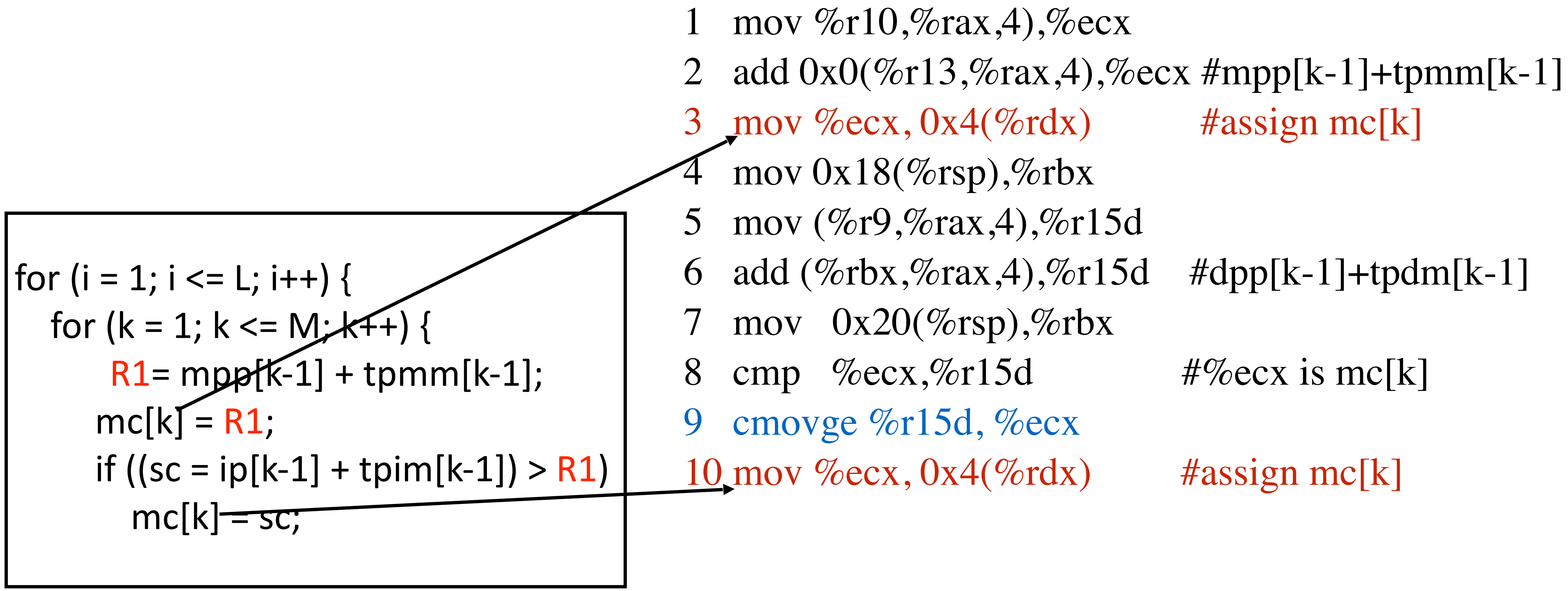
```
mov %r10,%rax,4),%ecx
add 0x0(%r13,%rax,4),%ecx #mpp[k-1]+tpmm[k-1]
mov %ecx, 0x4(%rdx)           #assign mc[k]
mov 0x18(%rsp),%rbx
mov (%r9,%rax,4),%r15d
add (%rbx,%rax,4),%r15d    #dpp[k-1]+tpdm[k-1]
mov   0x20(%rsp),%rbx
cmp   %ecx,%r15d              #%ecx is mc[k]
cmovge %r15d, %ecx
mov %ecx,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

HMMER: From the Binary Perspective

```
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;
```

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

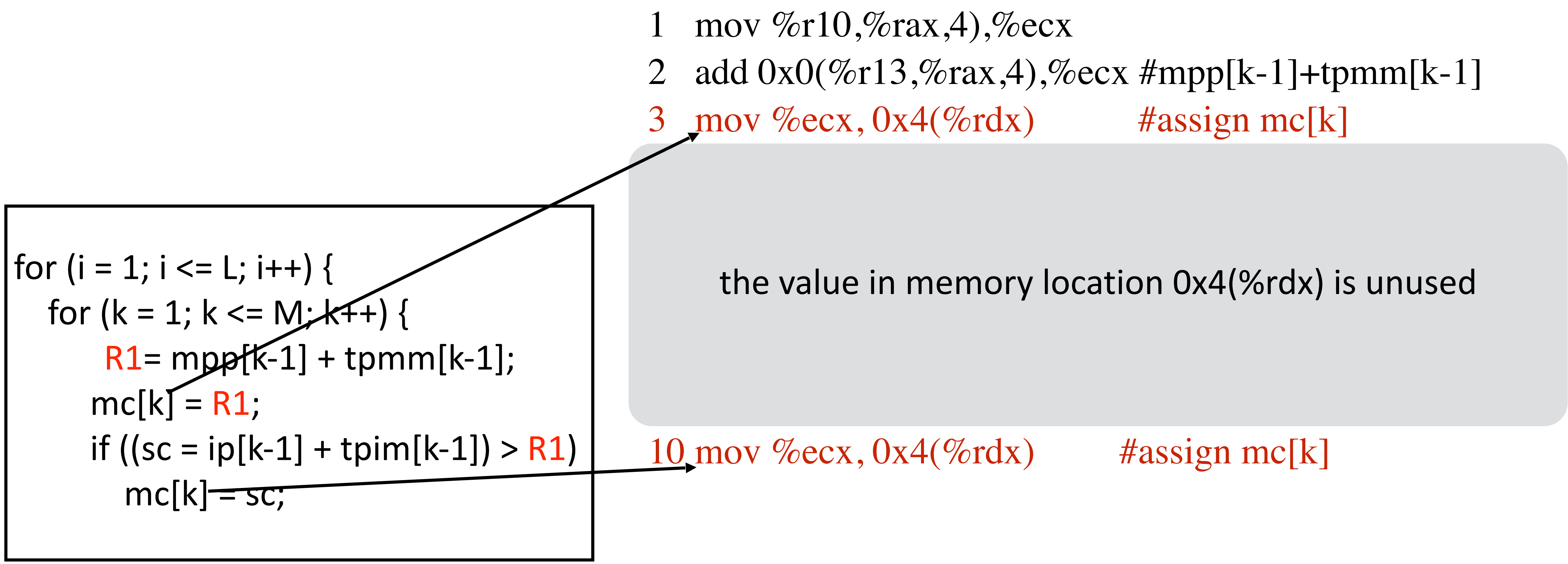
HMMER: From the Binary Perspective



```c
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;
```

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

HMMER: From the Binary Perspective



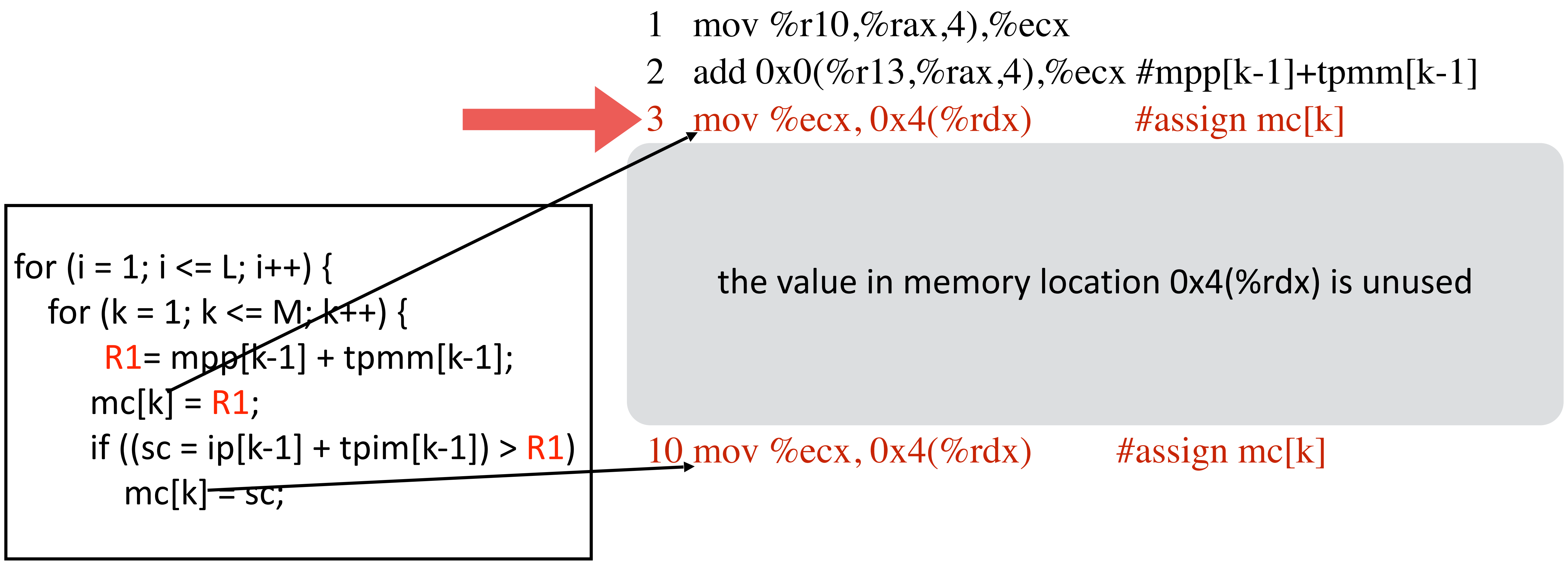
```
1   mov %r10,%rax,4),%ecx
2   add 0x0(%r13,%rax,4),%ecx #mpp[k-1]+tpmm[k-1]
3   mov %ecx, 0x4(%rdx)          #assign mc[k]
```

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

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

```
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: From the Binary Perspective



```
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]
```

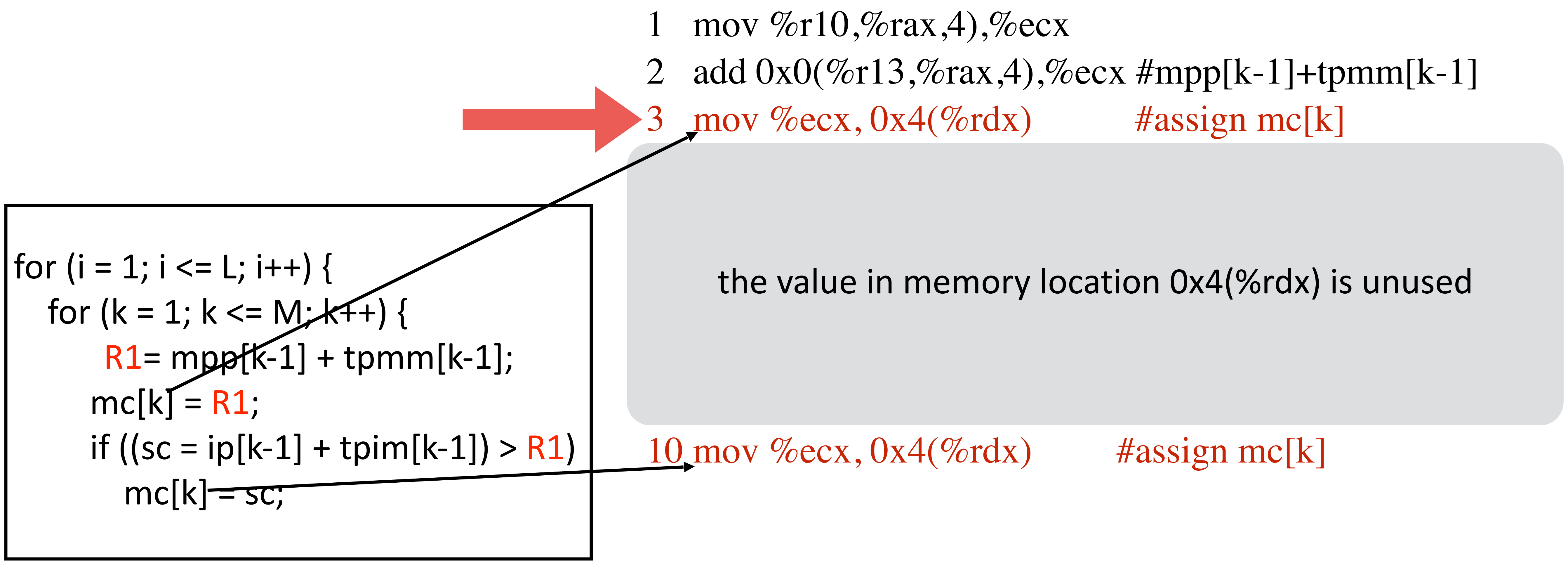
the value in memory location 0x4(%rdx) is unused

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

```
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;
```

dead store

HMMER: From the Binary Perspective



```
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]
```

```
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;
```

Dynamic fine-grained binary analysis
is necessary

Dynamic Fine-Grained Binary Tools

Microscopic insights

Track each instruction

- Operators and operands

Track each register

- General and SIMD registers

Track each memory location

- Effective addresses

Track values in storage locations

- Values in registers and memory

Existing Dynamic Fine-Grained Binary Tools

Tools

Valgrind / DynamoRIO / Dyninst / Pin

- High overhead but **microscopic insights**
- Suitable for analyzing software performance/correctness bugs

However,

Difficult to master

- Many complex APIs

Significant engineering efforts

- Obtaining deep insights
- Reducing measurement overhead

Existing Dynamic Fine-Grained Binary Tools

Tools

Valgrind / DynamoRIO / Dyninst / Pin

- High overhead but **microscopic insights**
- Suitable for analyzing software performance/correctness bugs

However,

Difficult to master

- Many complex APIs

Significant engineering efforts

- Obtaining deep insights
- Reducing measurement overhead

DrCCTProf facilitates the dynamic fine-grained binary analysis with easy interfaces

DrCCTProf Highlights

Rich insights

- Fine-grained **call path** profiling

Overhead optimization

- Handling efficient instrumentation
- Handling parallelism

Easy interfaces

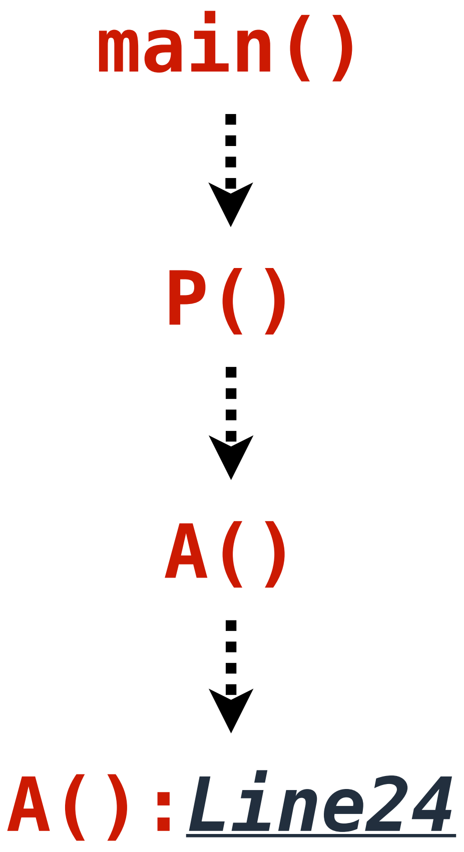
- Easy instrumentation
- Easy analysis
- Easy visualization

What is a Calling Context ?

```
void main(){
   P();
}
void P(){
   A();
   B();
   C();
   D();
   .......
   A();
}
void A(){
   *ptr = 100;
   x++;
   return;
}
```

What is a Calling Context ?

```
 1:    void main(){
 2:        P();
 3:    }
 4:    void P(){
 5:        A();
 6:        B();
 7:        C();
 8:        D();
...        .......
21:        A();
22:    }
23:    void A(){
24:        *ptr = 100;
25:        x++;
26:        return;
27:    }
```


main()

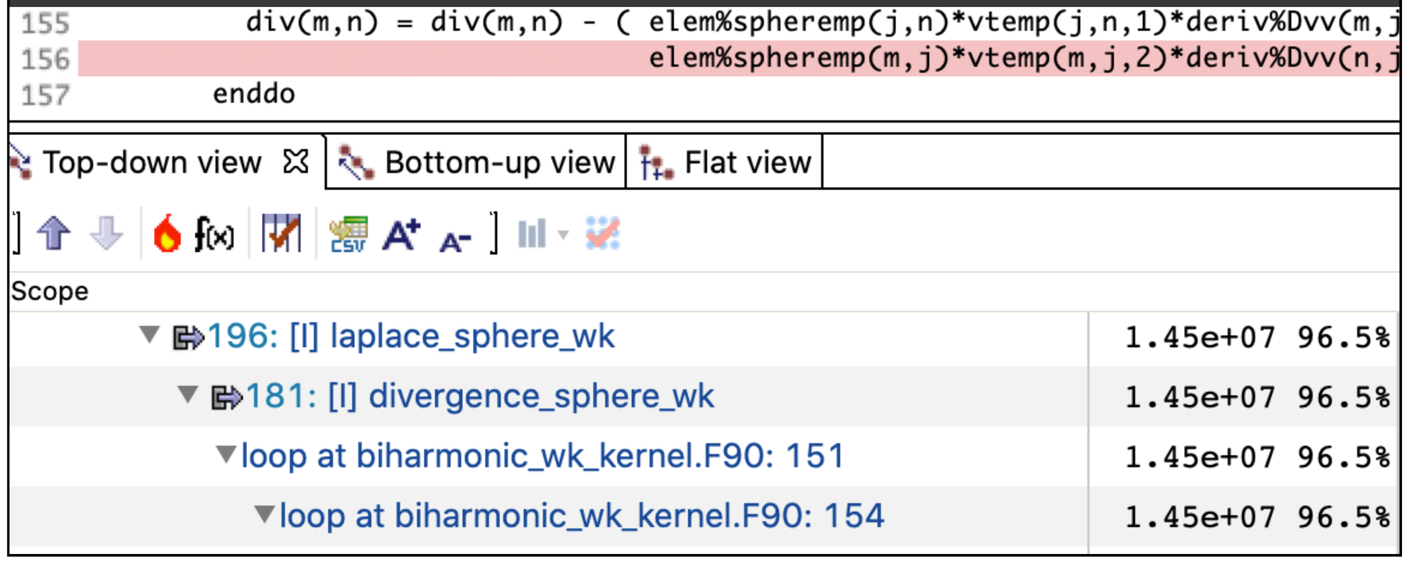
P()

A()

A():*Line24*

What is a Calling Context?

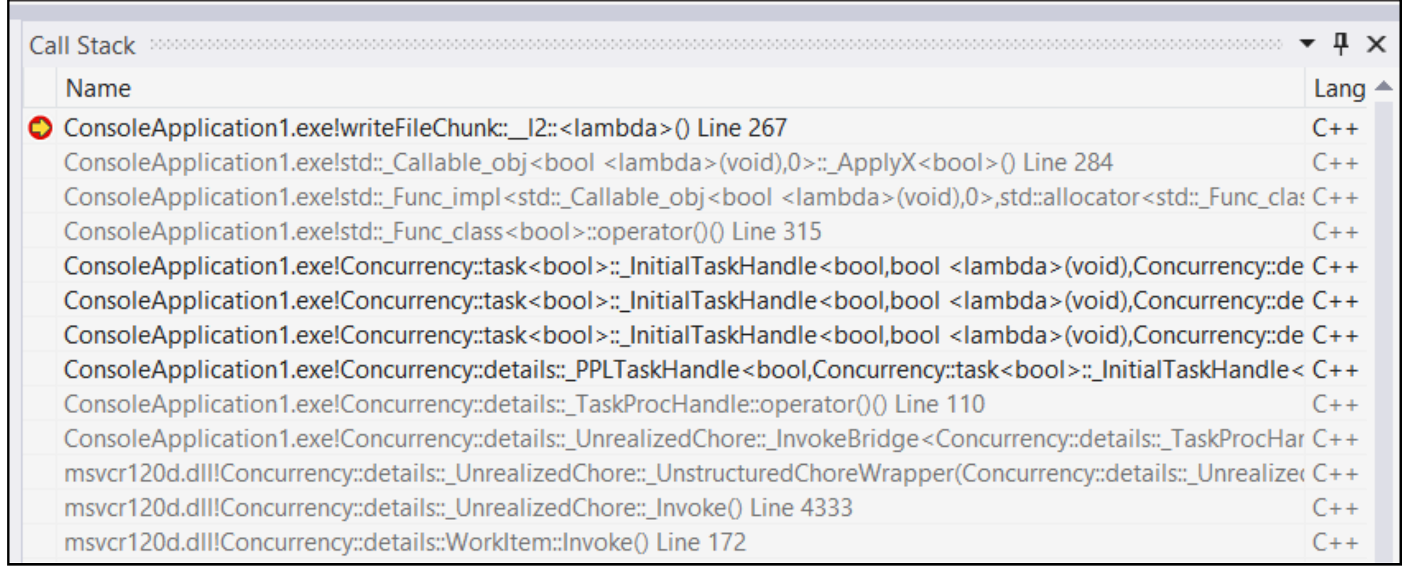
Performance Analysis Tools



```
        div(m,n) = div(m,n) - ( elem%spheremp(j,n)*vtemp(j,n,1)*deriv%Dvv(m,j
                                elem%spheremp(m,j)*vtemp(m,j,2)*deriv%Dvv(n,j
      enddo
```

Scope		
▼ 196: [I] laplace_sphere_wk	1.45e+07	96.5%
▼ 181: [I] divergence_sphere_wk	1.45e+07	96.5%
▼loop at biharmonic_wk_kernel.F90: 151	1.45e+07	96.5%
▼loop at biharmonic_wk_kernel.F90: 154	1.45e+07	96.5%

Debuggers



Name	Lang
ConsoleApplication1.exe!writeFileChunk::_I2::<lambda>() Line 267	C++
ConsoleApplication1.exe!std::_Callable_obj<bool <lambda>(void),0>::_ApplyX<bool>() Line 284	C++
ConsoleApplication1.exe!std::_Func_impl<std::_Callable_obj<bool <lambda>(void),0>,std::allocator<std::_Func_clas	C++
ConsoleApplication1.exe!std::_Func_class<bool>::operator()() Line 315	C++
ConsoleApplication1.exe!Concurrency::task<bool>::_InitialTaskHandle<bool,bool <lambda>(void),Concurrency::de	C++
ConsoleApplication1.exe!Concurrency::task<bool>::_InitialTaskHandle<bool,bool <lambda>(void),Concurrency::de	C++
ConsoleApplication1.exe!Concurrency::task<bool>::_InitialTaskHandle<bool,bool <lambda>(void),Concurrency::de	C++
ConsoleApplication1.exe!Concurrency::details::_PPLTaskHandle<bool,Concurrency::task<bool>::_InitialTaskHandle<	C++
ConsoleApplication1.exe!Concurrency::details::_TaskProcHandle::operator()() Line 110	C++
ConsoleApplication1.exe!Concurrency::details::_UnrealizedChore::_InvokeBridge<Concurrency::details::_TaskProcHar	C++
msvcr120d.dll!Concurrency::details::_UnrealizedChore::_UnstructuredChoreWrapper(Concurrency::details::_Unrealizec	C++
msvcr120d.dll!Concurrency::details::_UnrealizedChore::_Invoke() Line 4333	C++
msvcr120d.dll!Concurrency::details::WorkItem::Invoke() Line 172	C++

Chain of function calls that led to the current point in the program.
(a.k.a **Call Path** / **Call Stack** / **Backtrace** / **Activation Record**)

Why Calling Contexts are Necessary?

SPEC 2006: 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>
```

Why Calling Contexts are Necessary?

SPEC 2006: 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
```

Why Calling Contexts are Necessary?

SPEC 2006: 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
```

Why Calling Contexts are Necessary?

SPEC 2006: 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
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Why Calling Contexts are Necessary?

SPEC 2006: 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 Overview

Monitor unmodified, fully-optimized binary executables on ARM.

Ubiquitous call path collection **[code-centric]**

- Associate **calling context** with their metrics potentially on **every executed machine instruction**

Ubiquitous object attribution **[data-centric]**

- Associate **every memory address** with the **corresponding data object**.

Programmability

Framework to support various tools

Affordable overhead

Suitable for large scale execution

DrCCTProf

- Ubiquitous call path collection
- Attributing costs to data objects
- Handling parallelism
- Evaluation
- Case study
- Conclusions

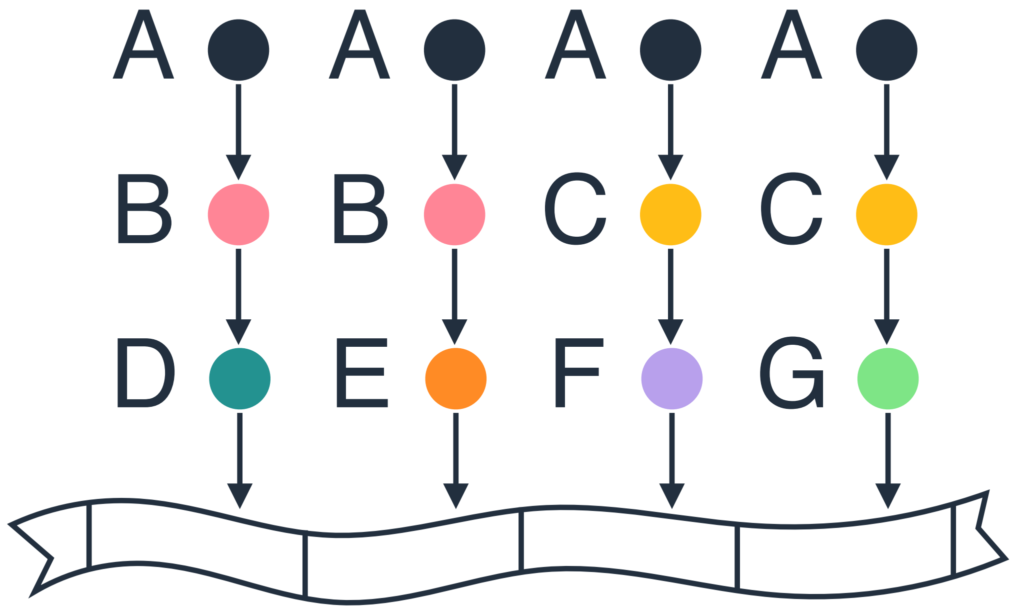
DrCCTProf

- **Ubiquitous call path collection**
- Attributing costs to data objects
- Handling parallelism
- Evaluation
- Case study
- Conclusions

Store History of Contexts Compactly

Problem

Deluge of call paths

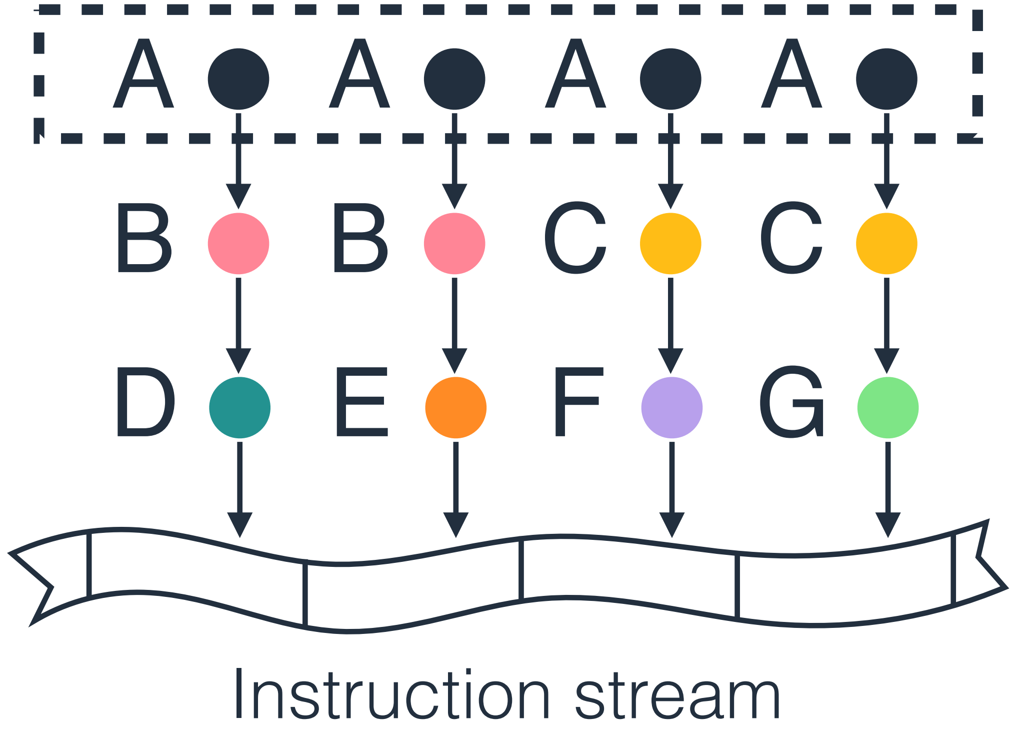


Instruction stream

Store History of Contexts Compactly

Problem

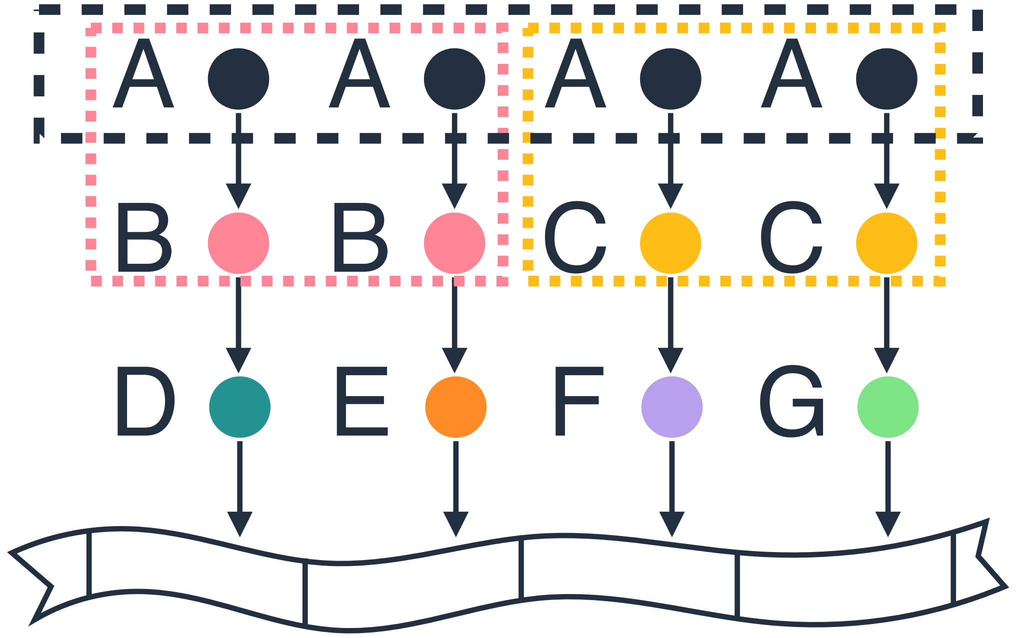
Deluge of call paths



Store History of Contexts Compactly

Problem

Deluge of call paths

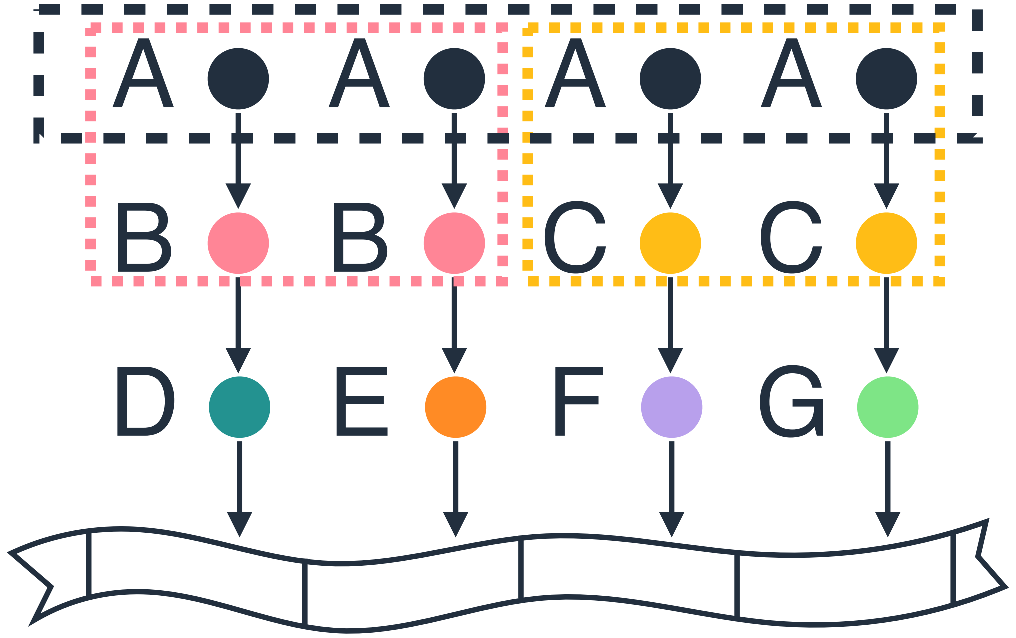


Instruction stream

Store History of Contexts Compactly

Solution

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

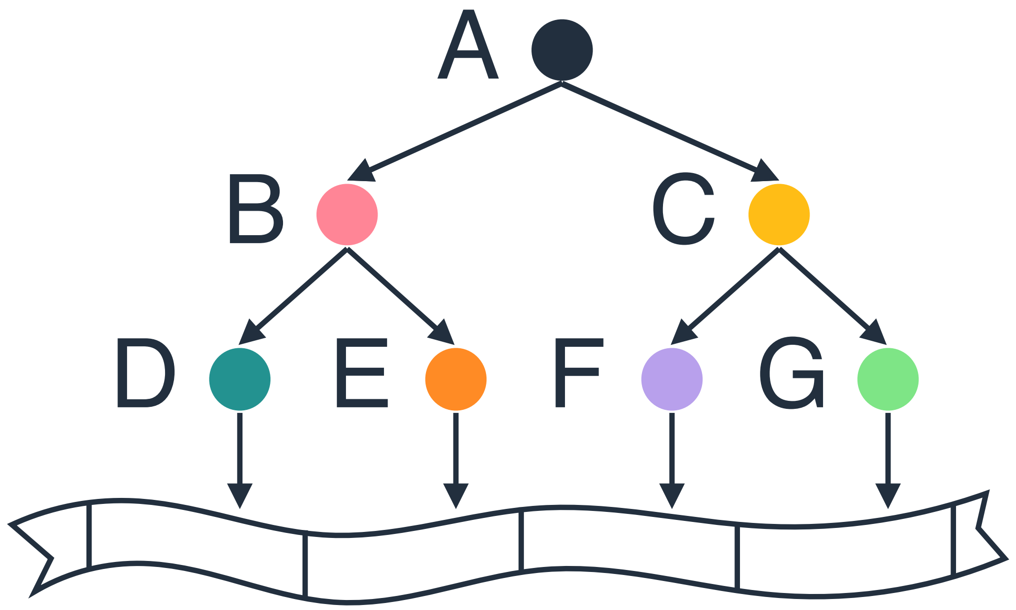


Instruction stream

Store History of Contexts Compactly

Solution

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



Instruction stream

Shadowing Call Stack

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

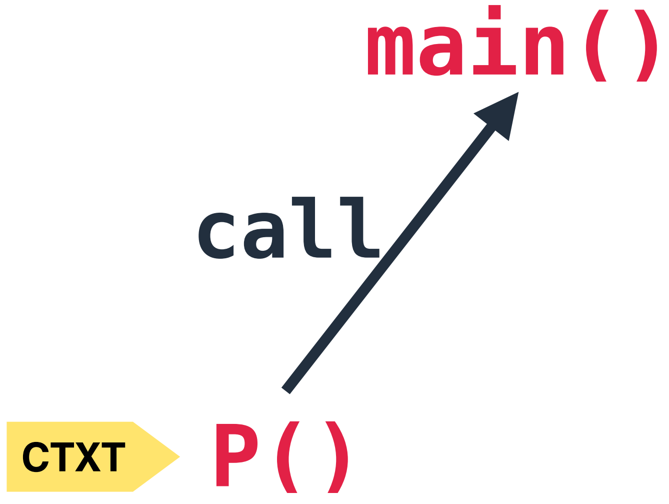
```
void main(){
   P();
}
void P(){
   A();
   B();
   C();
   D();
   ......
   A();
}
void A(){
   *ptr = 100;
   x++;
   return;
}
```

`main()`

Shadowing Call Stack

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

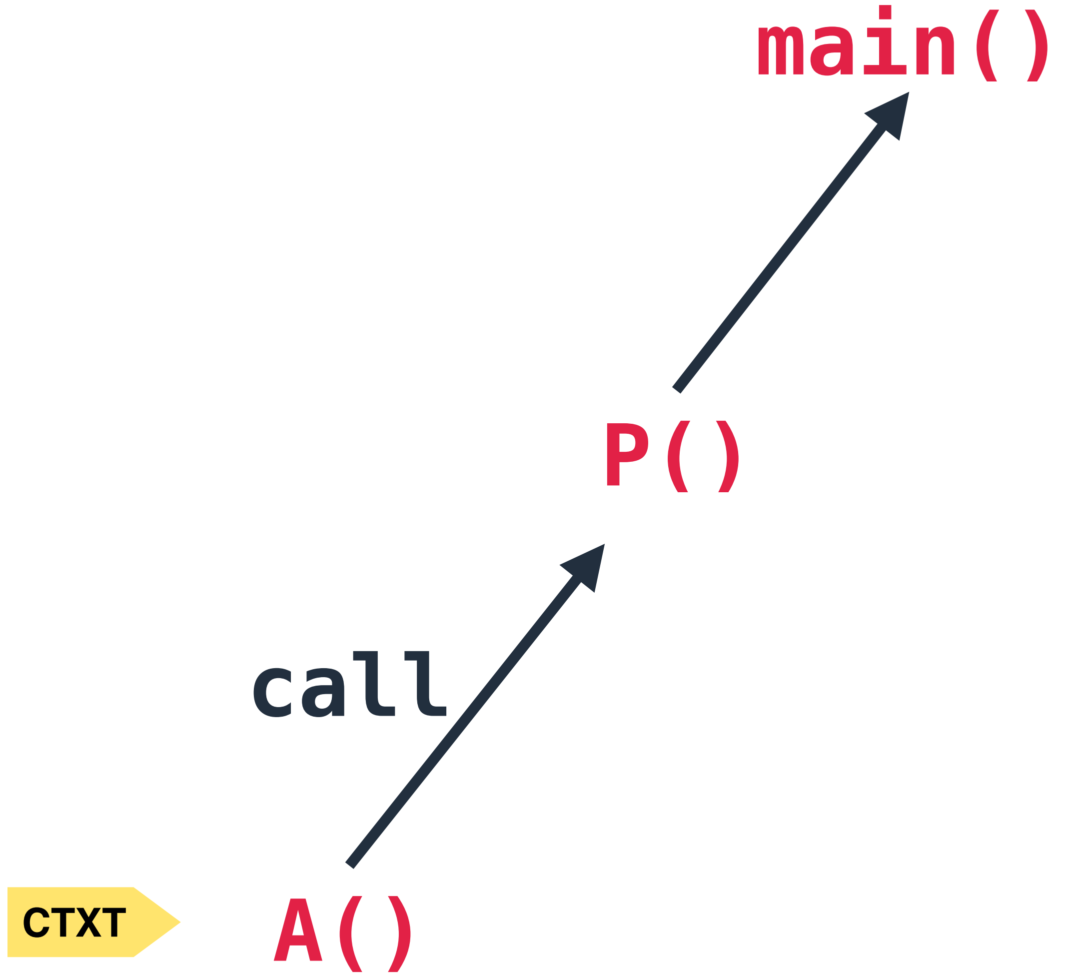
```
void main(){
    P();
}
void P(){
    A();
    B();
    C();
    D();
    .......
    A();
}
void A(){
    *ptr = 100;
    x++;
    return;
}
```



Shadowing Call Stack

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

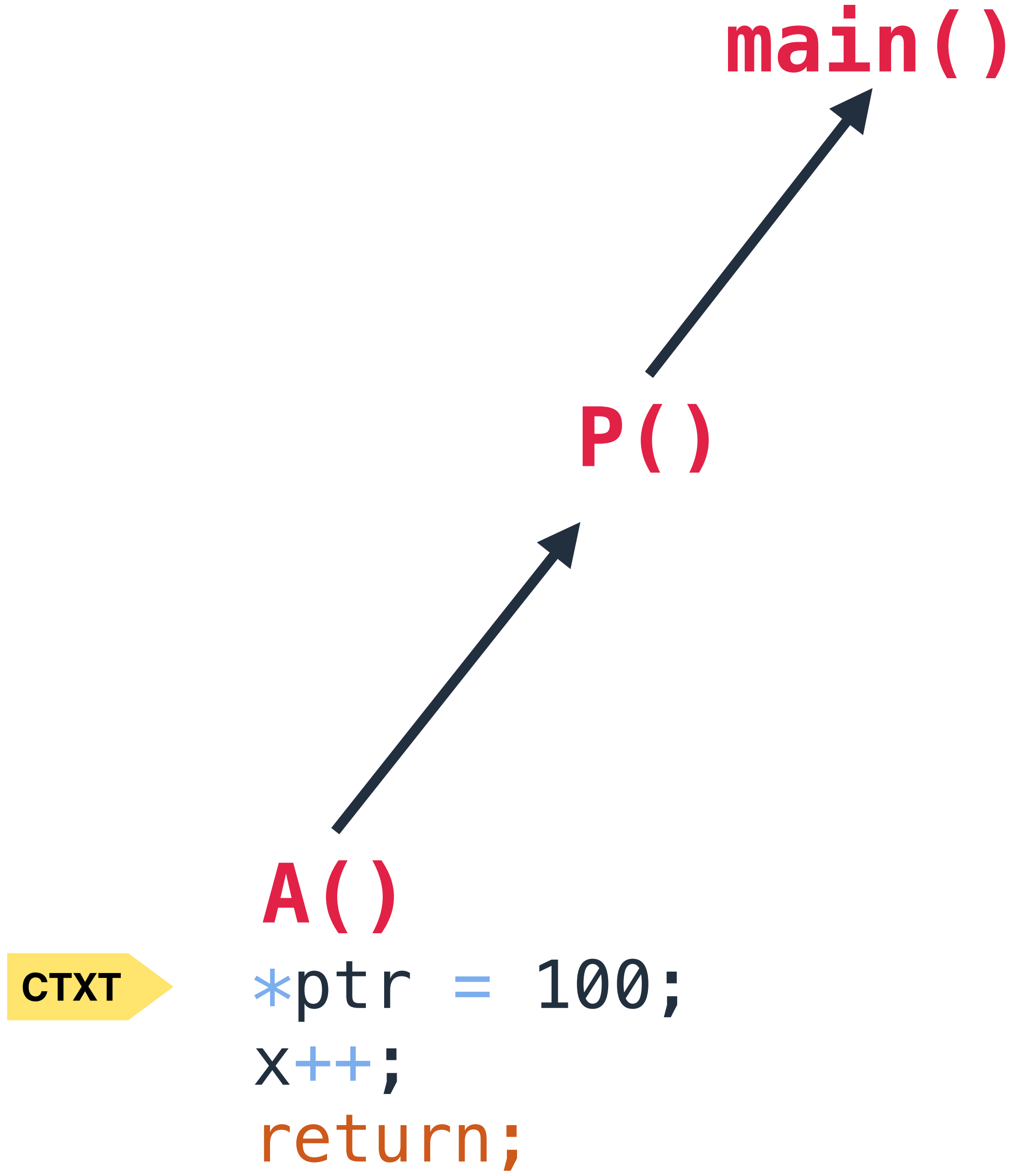
```
void main(){
    P();
}
void P(){
    A();
    B();
    C();
    D();
    .......
    A();
}
void A(){
    *ptr = 100;
    x++;
    return;
}
```



Shadowing Call Stack

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

```
void main(){
    P();
}
void P(){
    A();
    B();
    C();
    D();
    .......
    A();
}
void A(){
    *ptr = 100;
    x++;
    return;
}
```



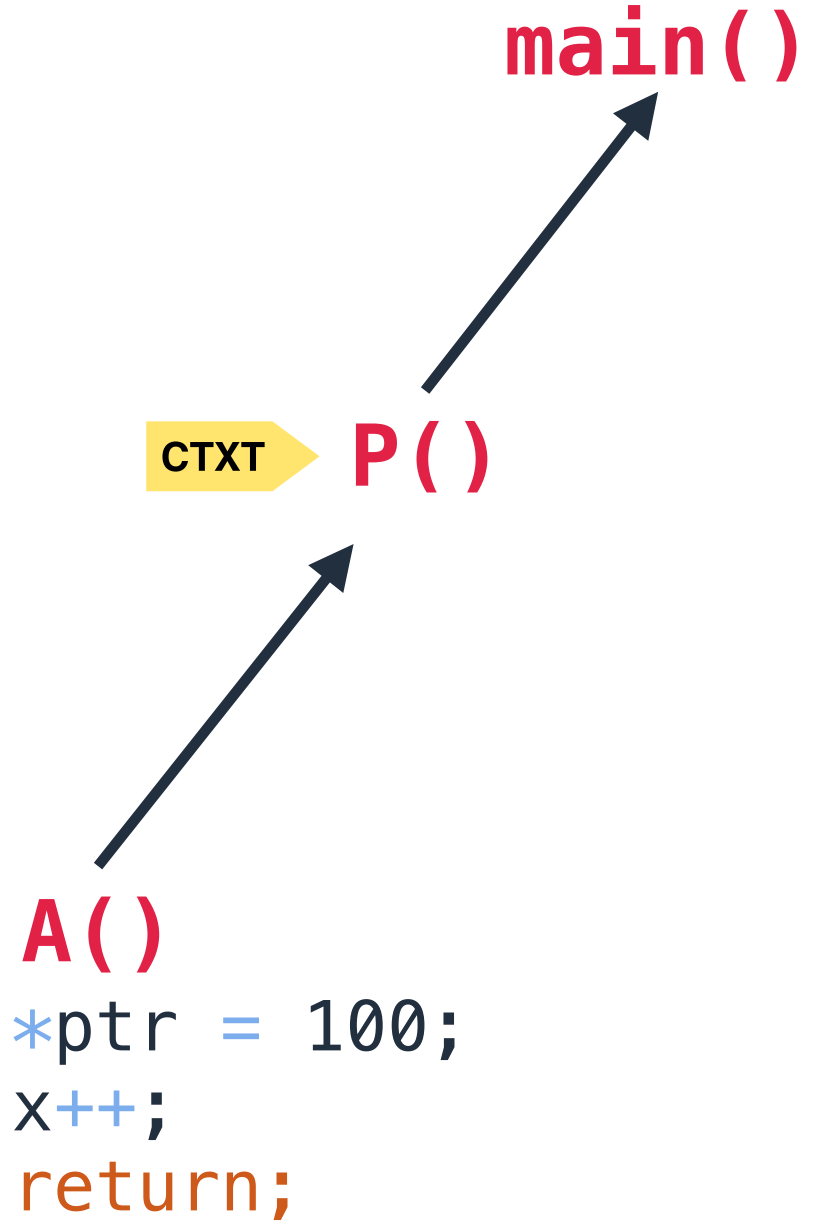
```
*ptr = 100;
x++;
return;
```

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

Shadowing Call Stack

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

```
void main(){
    P();
}
void P(){
    A();
    B();
    C();
    D();
    .......
    A();
}
void A(){
    *ptr = 100;
    x++;
    return;
}
```



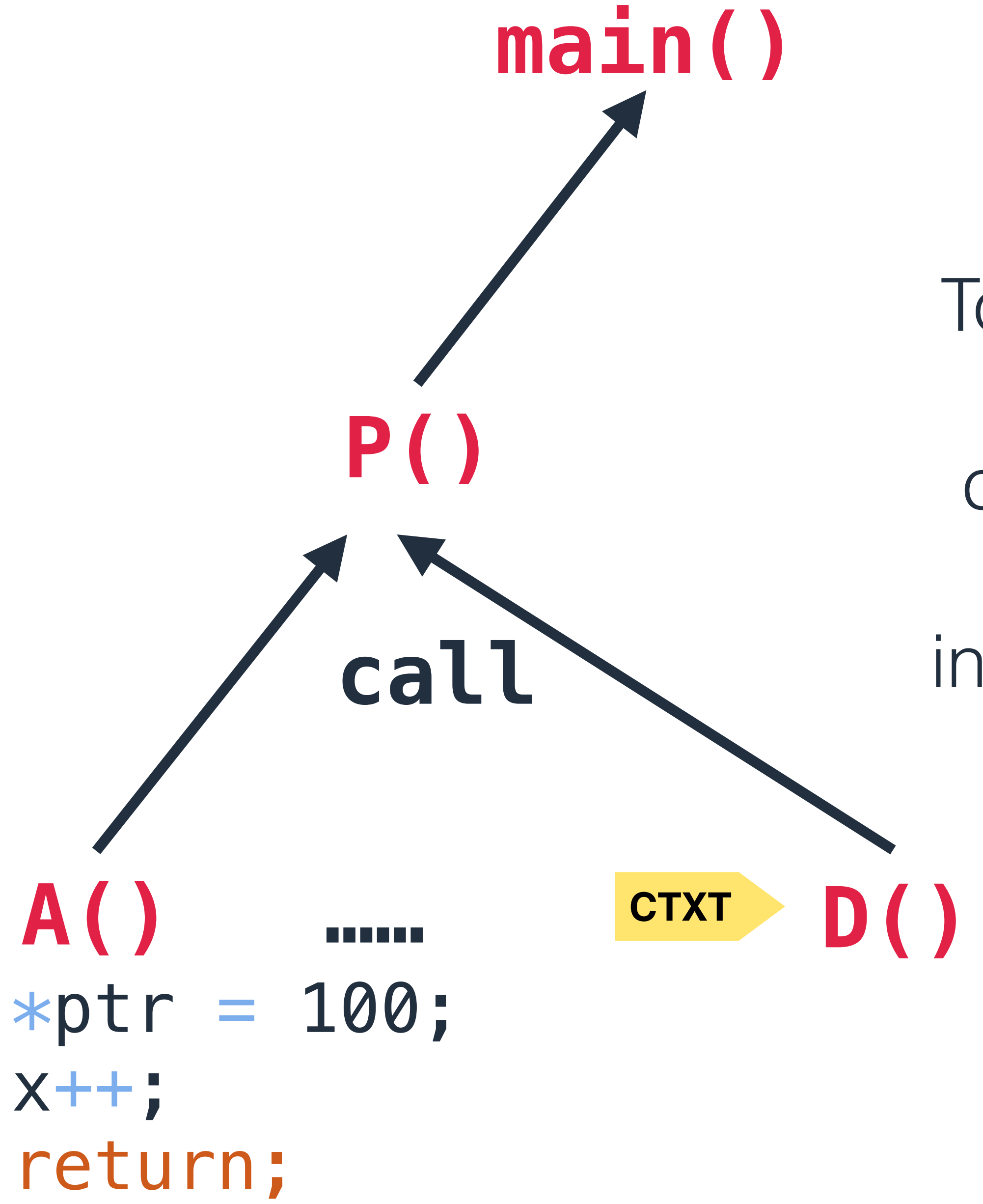
```
*ptr = 100;
x++;
return;
```

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

Shadowing Call Stack

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

```
void main(){
    P();
}
void P(){
    A();
    B();
    C();
    D();
    .......
    A();
}
void A(){
    *ptr = 100;
    x++;
    return;
}
```

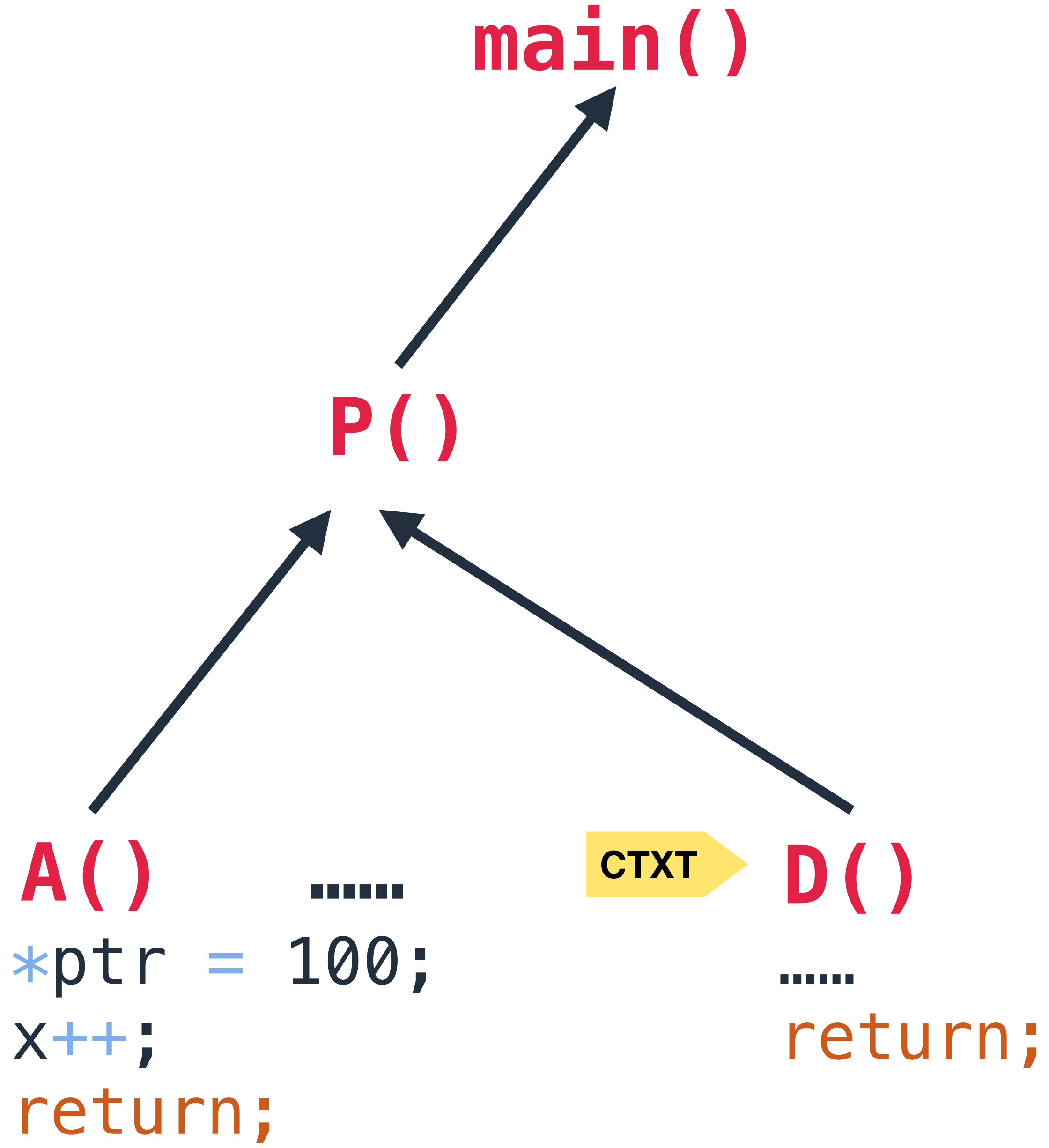


Tools can obtain pointer to the current context via CTXT in **constant time**

Shadowing Call Stack

Maintaining CTXT

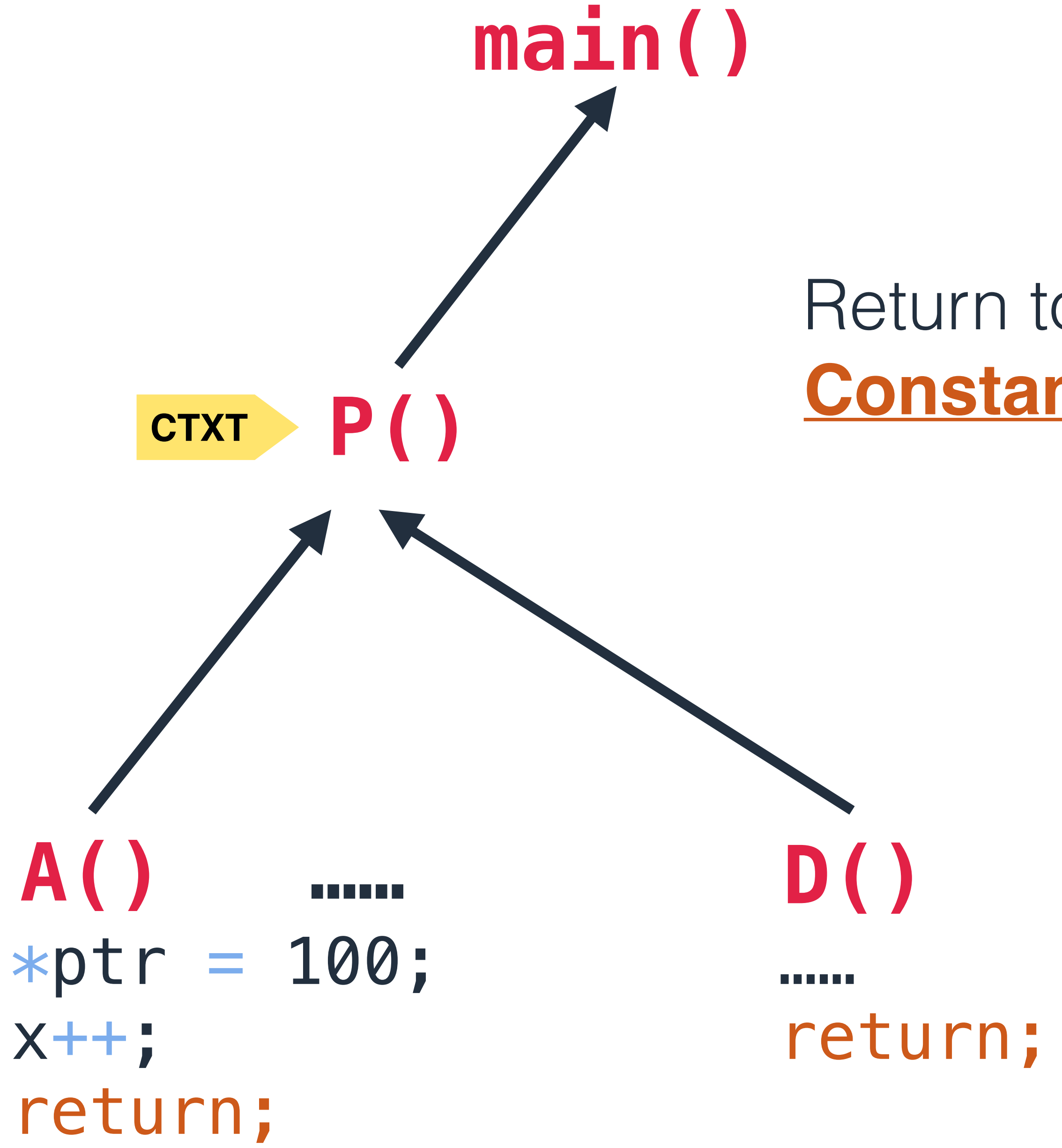
```
void main(){
    P();
}
void P(){
    A();
    B();
    C();
    D();
    .......
    A();
}
void A(){
    *ptr = 100;
    x++;
    return;
}
```



Shadowing Call Stack

Maintaining CTXT

```
void main(){
    P();
}
void P(){
    A();
    B();
    C();
    D();
    .......
    A();
}
void A(){
    *ptr = 100;
    x++;
    return;
}
```

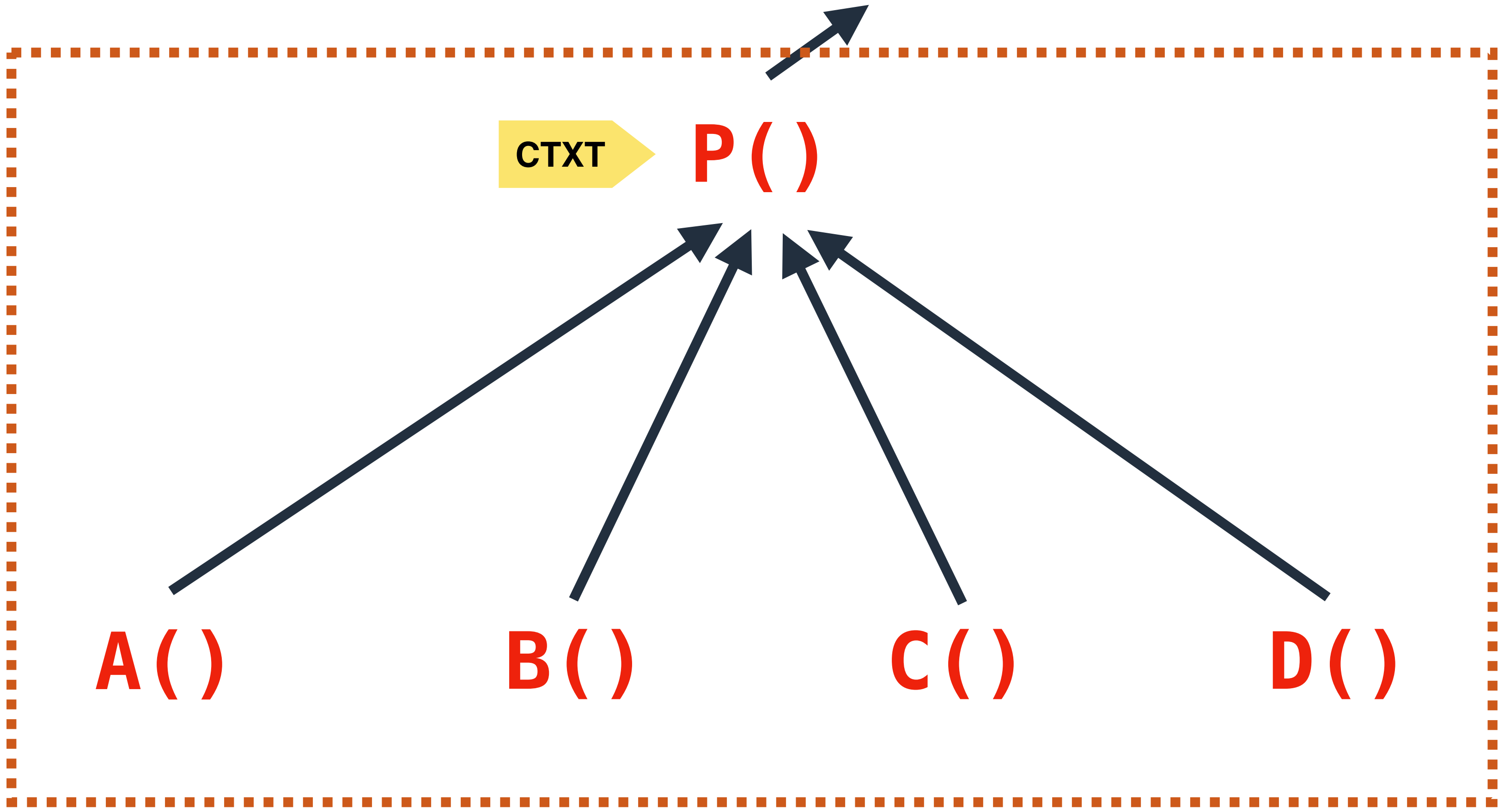


```
*ptr = 100;
x++;
return;
```

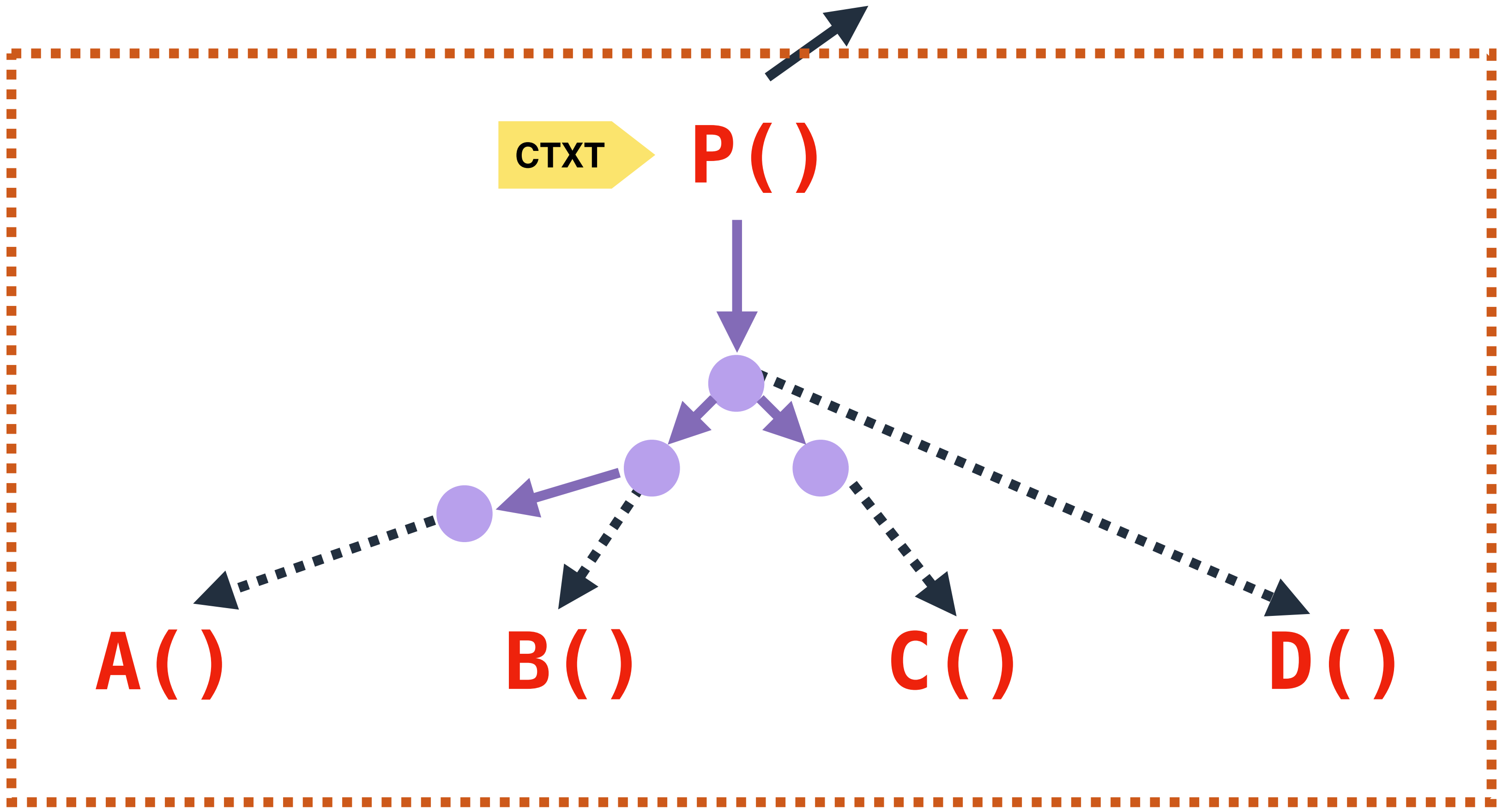
```
......
return;
```

Return to caller:
Constant time update

Accelerate Lookup with Splay Trees



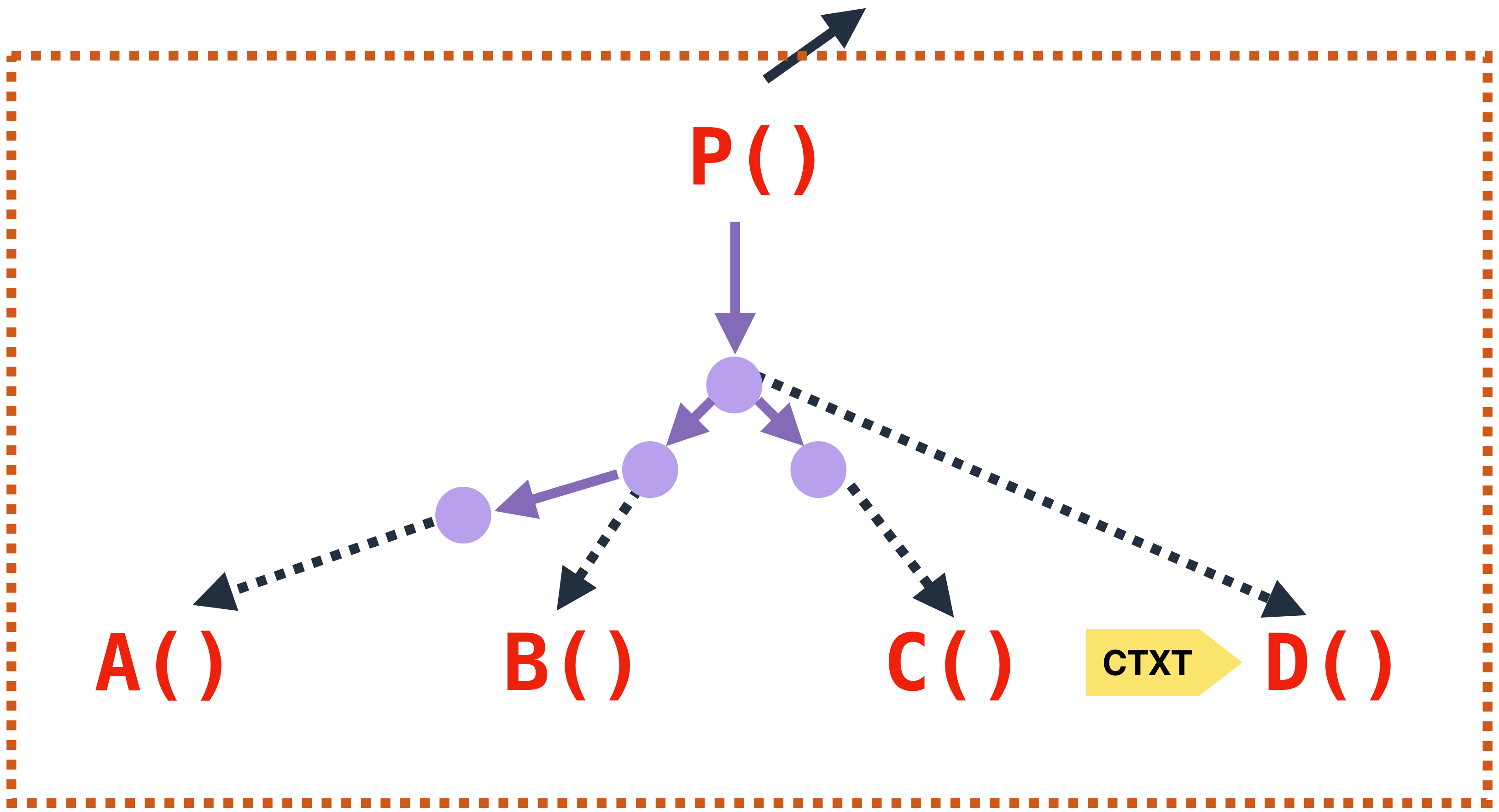
Accelerate Lookup with Splay Trees



Splay tree [“Self-adjusting binary search trees” by Sleator et al. 1985]

ensures frequently called functions are near the root of the tree

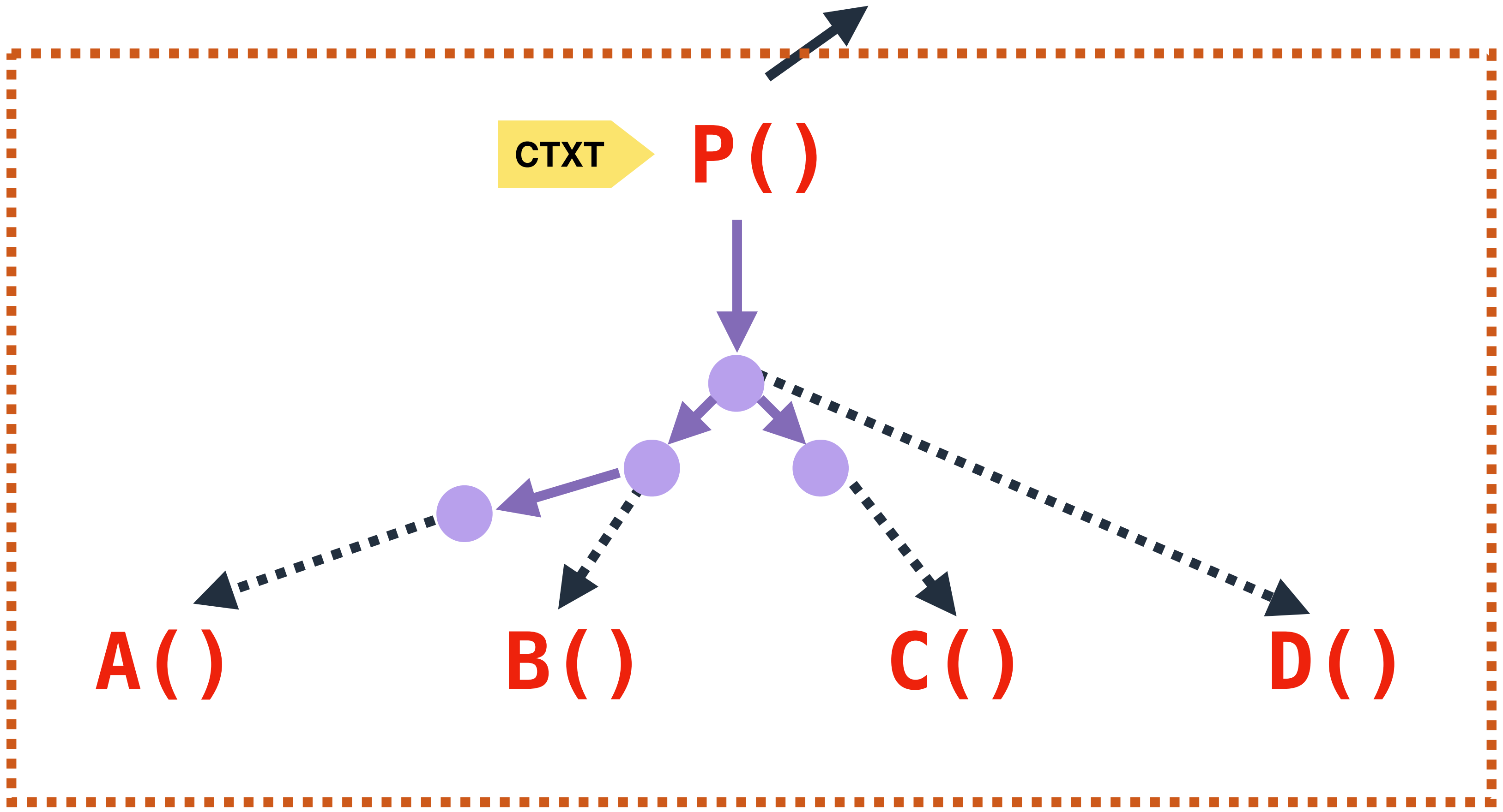
Accelerate Lookup with Splay Trees



Splay tree [“Self-adjusting binary search trees” by Sleator et al. 1985]

ensures frequently called functions are near the root of the tree

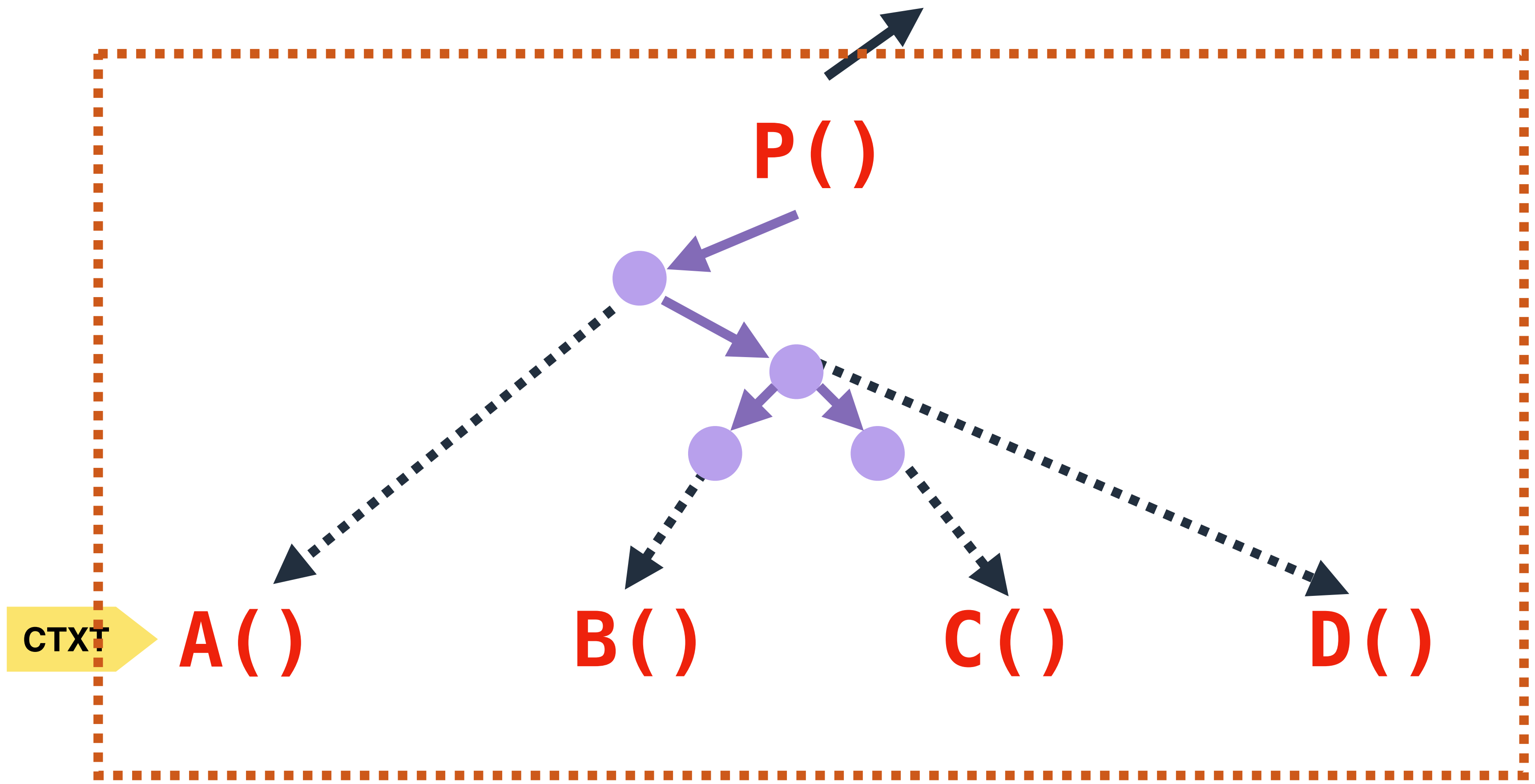
Accelerate Lookup with Splay Trees



Splay tree ["Self-adjusting binary search trees" by Sleator et al. 1985]

ensures frequently called functions are near the root of the tree

Accelerate Lookup with Splay Trees



Splay tree [“Self-adjusting binary search trees” by Sleator et al. 1985] ensures frequently called functions are near the root of the tree

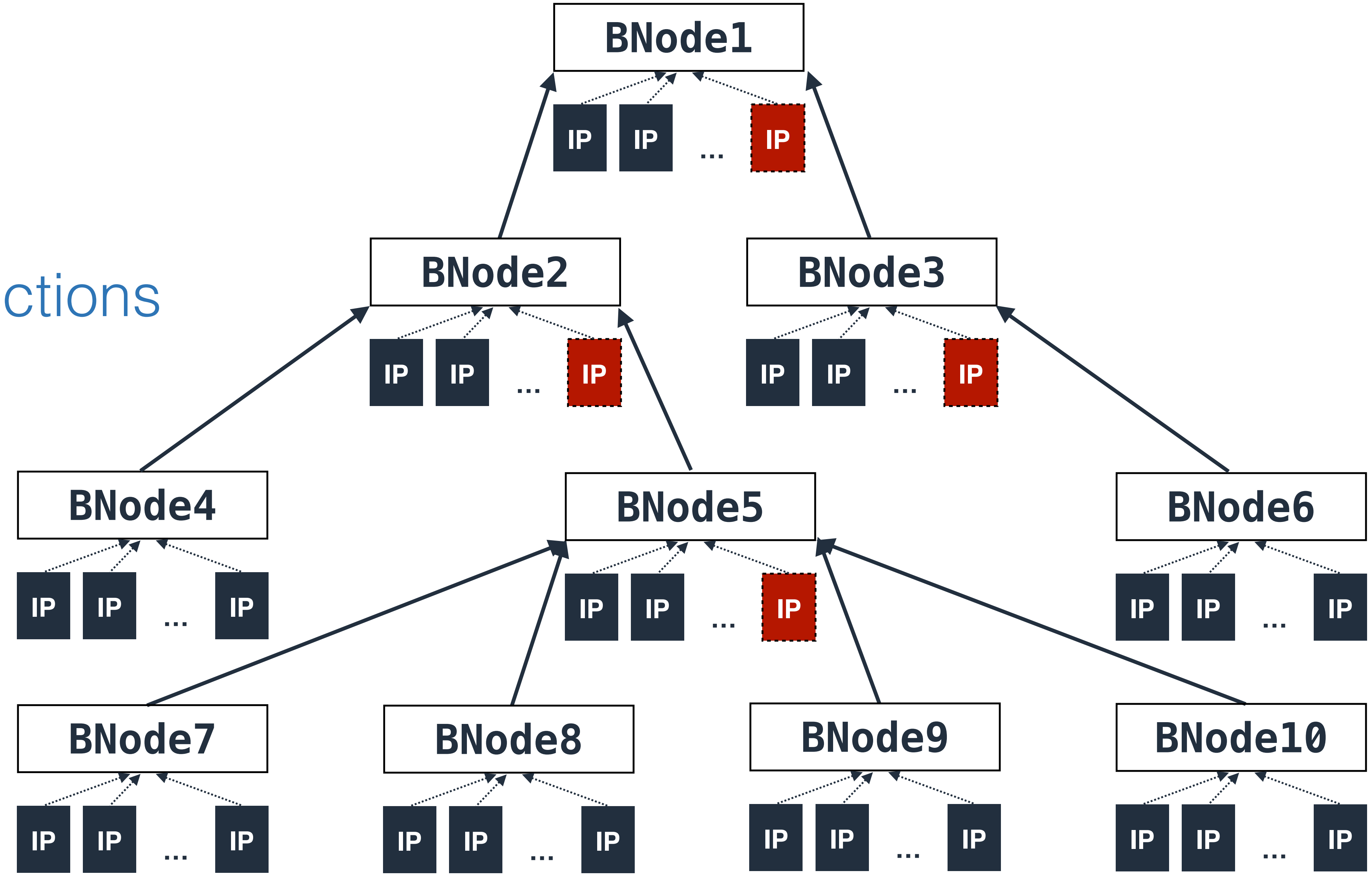
Overview of Calling Context Tree

BNode

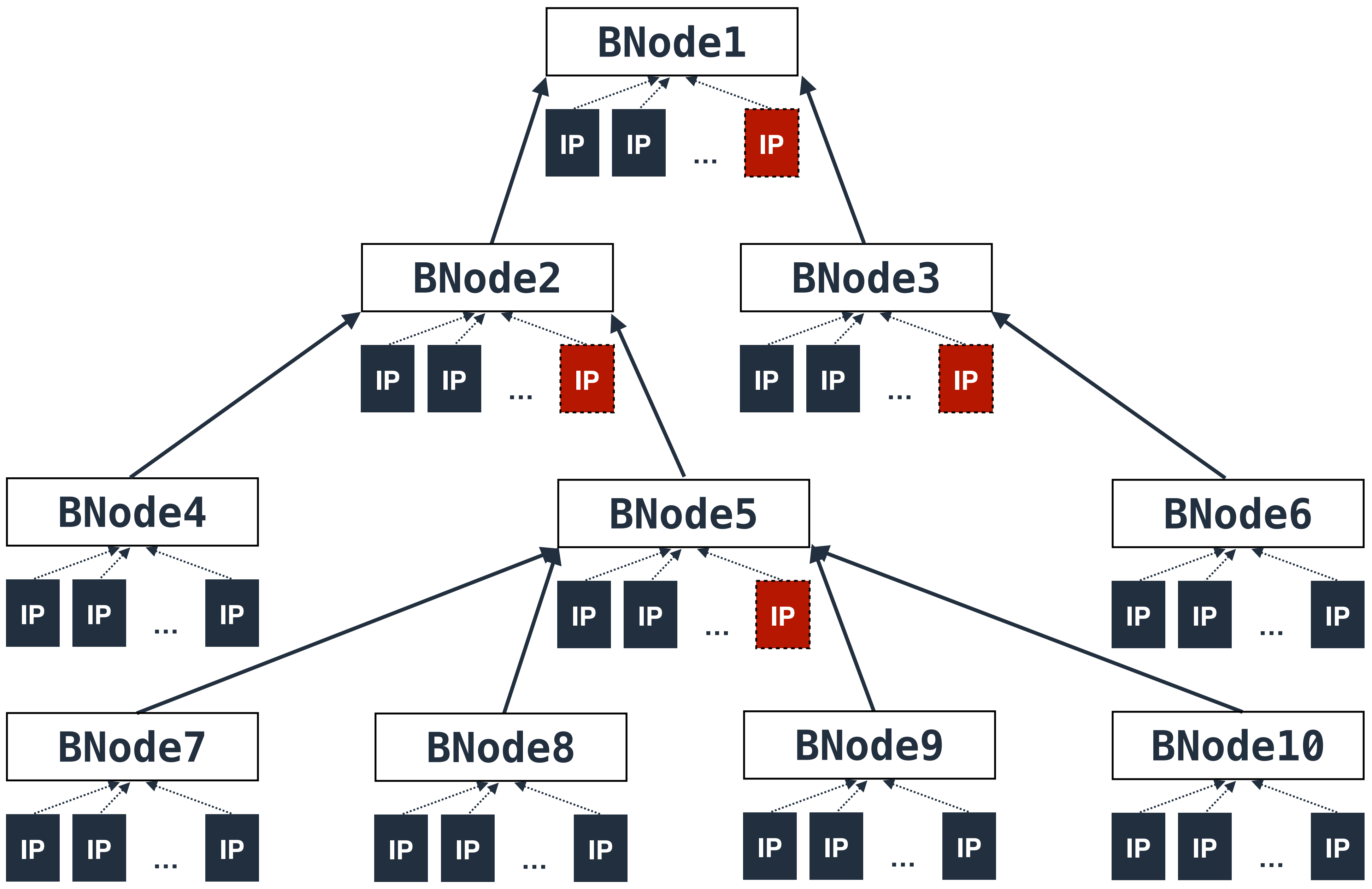
represents a basic block

IPNode

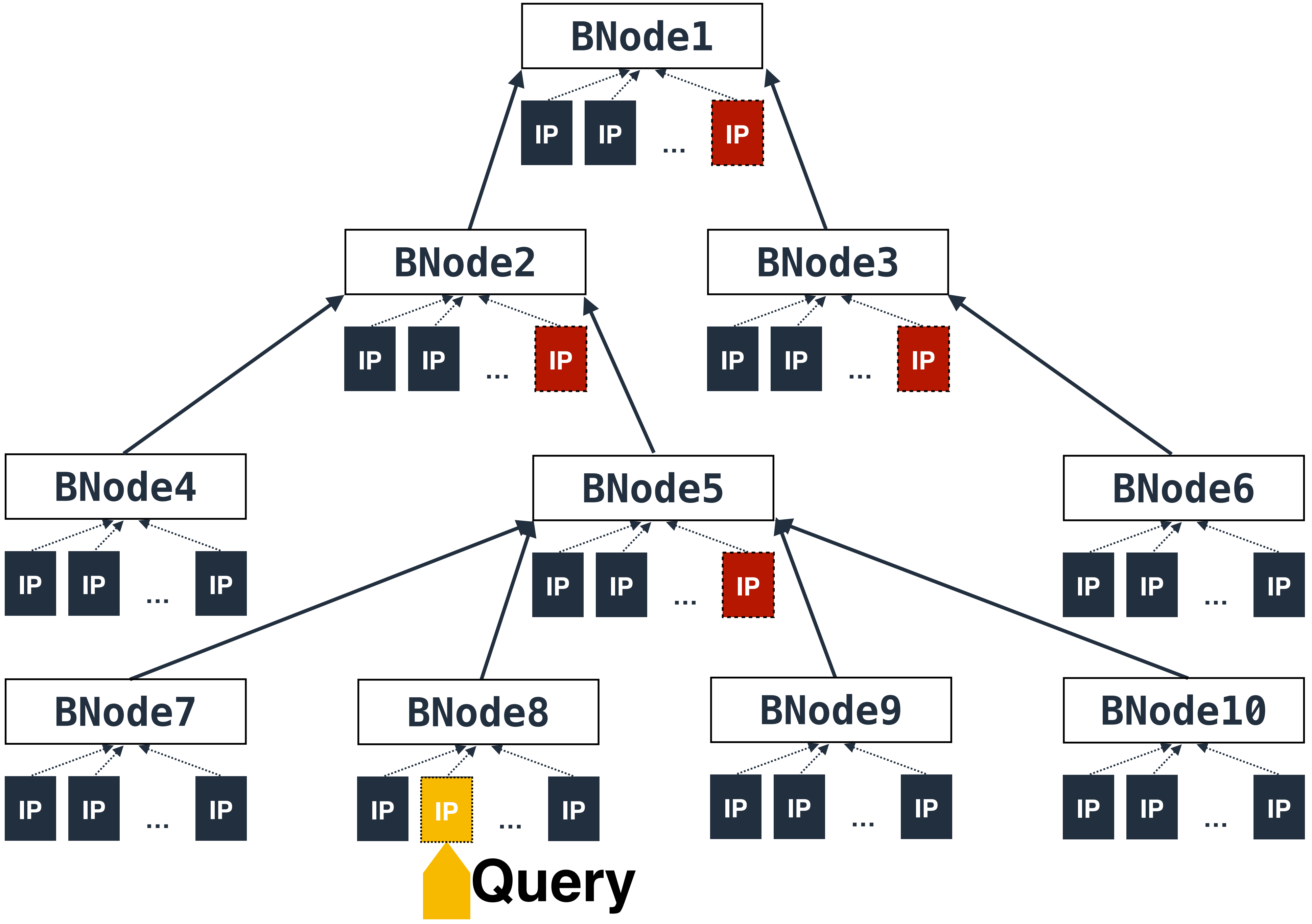
represents individual instructions



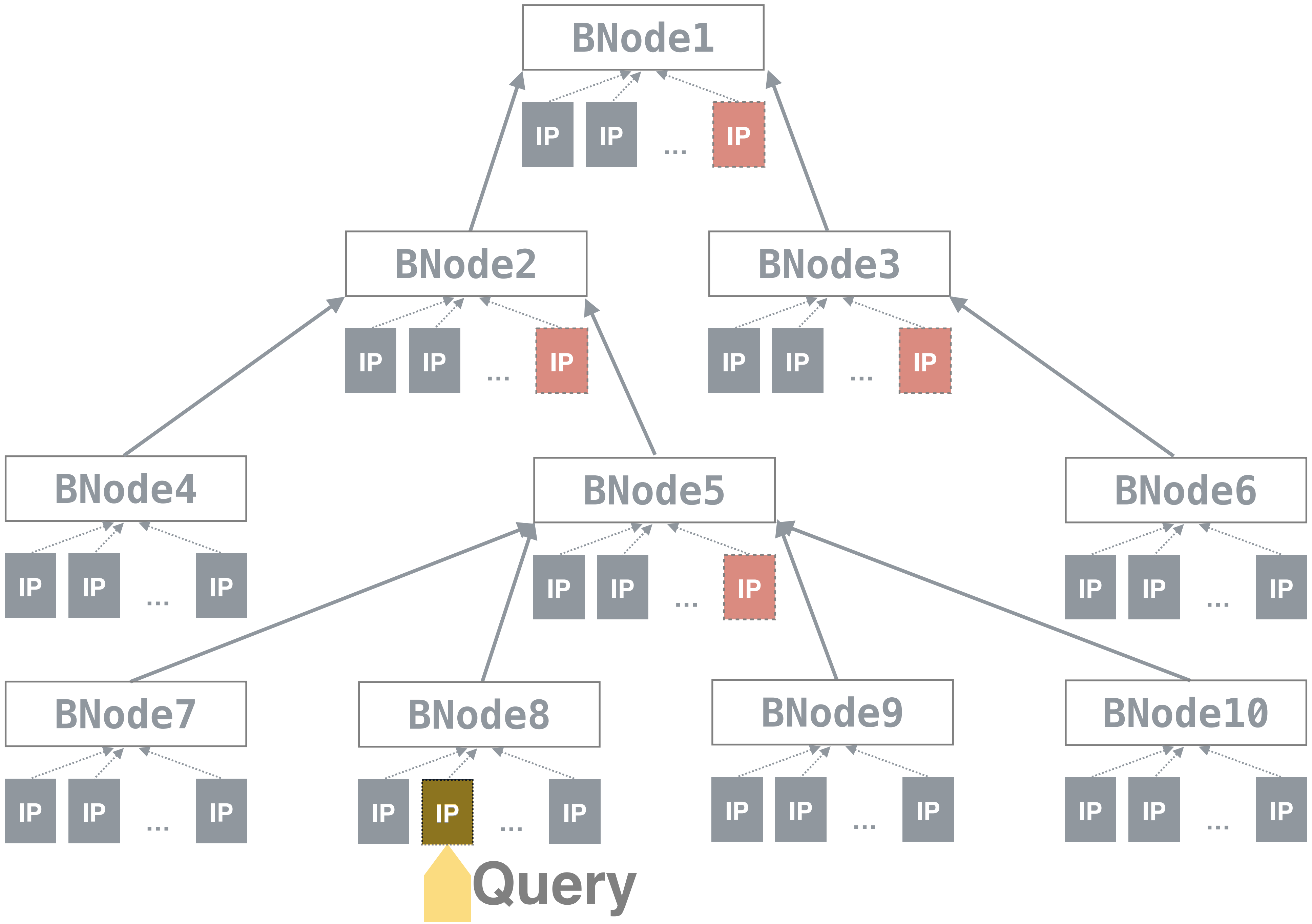
Query Call Path for Any Instruction

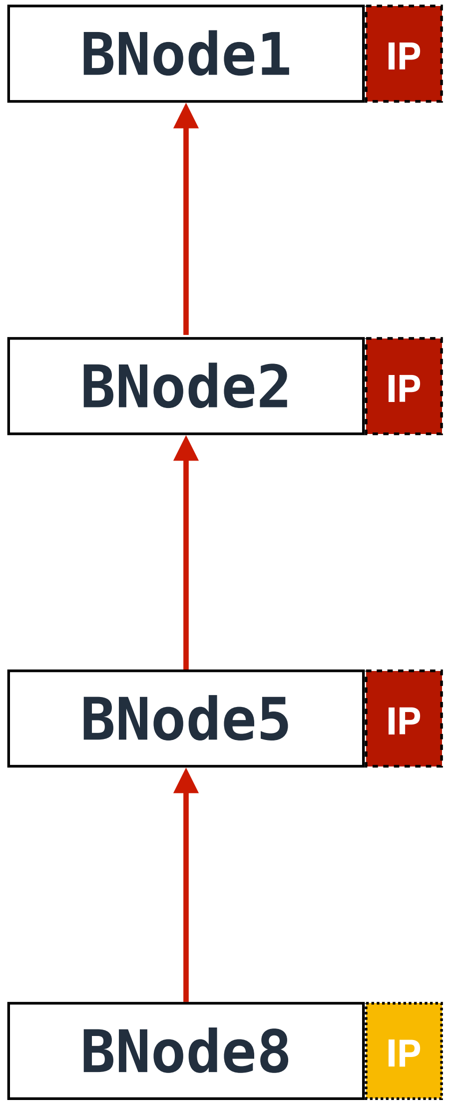


Query Call Path for Any Instruction



Query Call Path for Any Instruction





Call Path

- Ubiquitous call path collection
- **Attributing costs to data objects**
- Handling parallelism
- Evaluation
- Case study
- Conclusions

Data-Centric Attribution

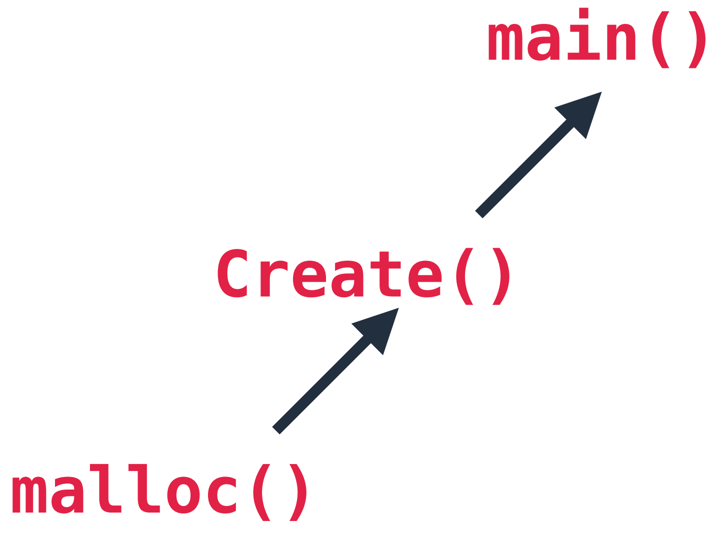
```
int MyArray[s];

void Create(){
    return malloc();
}

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

void main(){
    int *m;
    p1 = Create();
    Update(p1);
    p2 = MyArray;
    Update(p);
}
```

`main()`

Associate each data access with its **data object**

Data-Centric Attribution

```
int MyArray[s];

void Create(){
    return malloc();
}

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

void main(){
    int *m;
    p1 = Create();
    Update(p1);
    p2 = MyArray;
    Update(p);
}
```



main()

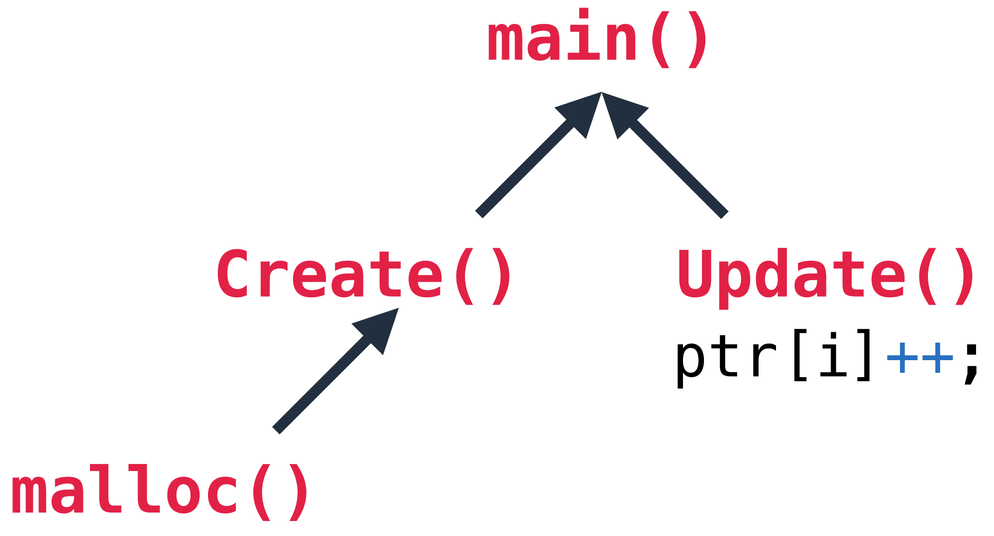
Create()

malloc()

Associate each data access with its **data object**

Data-Centric Attribution

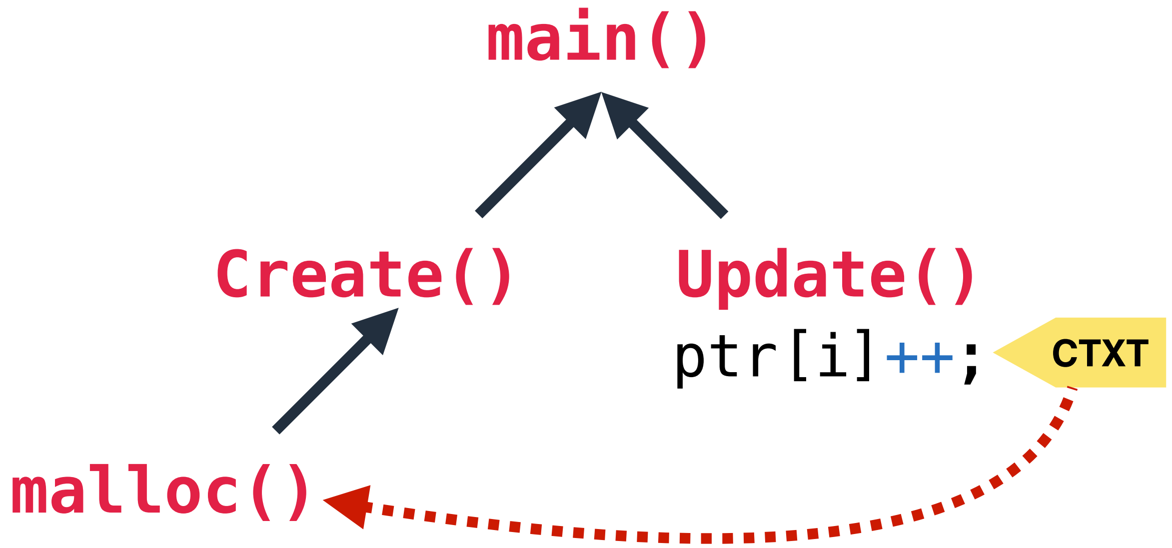
```
int MyArray[s];

void Create(){
    return malloc();
}

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

void main(){
    int *m;
    p1 = Create();
    Update(p1);
    p2 = MyArray;
    Update(p);
}
```



Associate each data access with its **data object**

Data-Centric Attribution

```
int MyArray[s];

void Create(){
    return malloc();
}

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

void main(){
    int *m;
    p1 = Create();
    Update(p1);
    p2 = MyArray;
    Update(p);
}
```

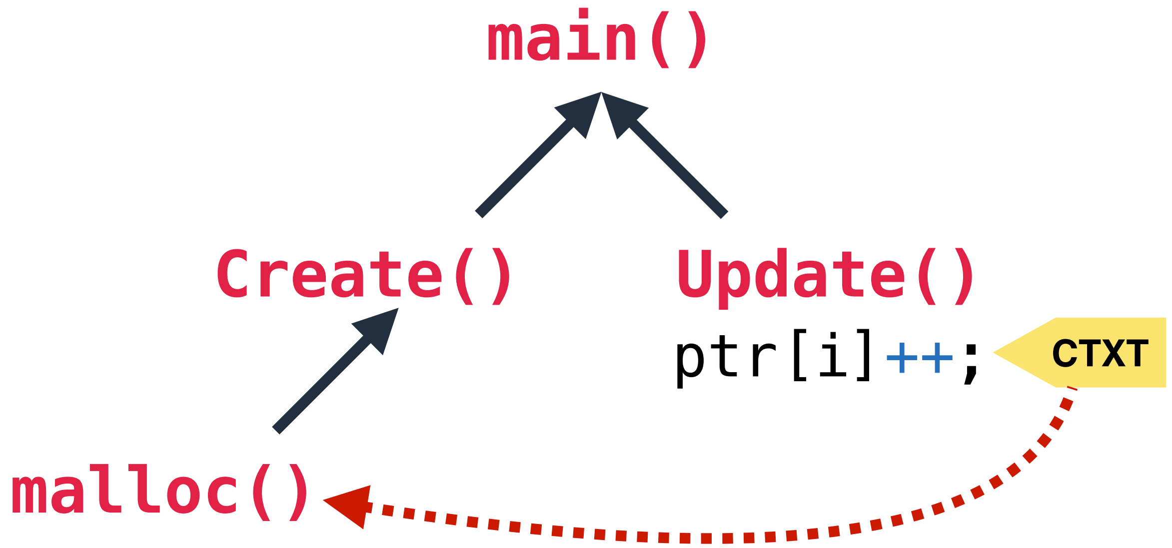


Associate each data access with its **data object**

- Dynamic allocation: **Call path of allocation site**

Data-Centric Attribution

```
int MyArray[s];

void Create(){
    return malloc();
}

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

void main(){
    int *m;
    p1 = Create();
    Update(p1);
    p2 = MyArray;
    Update(p);
}
```

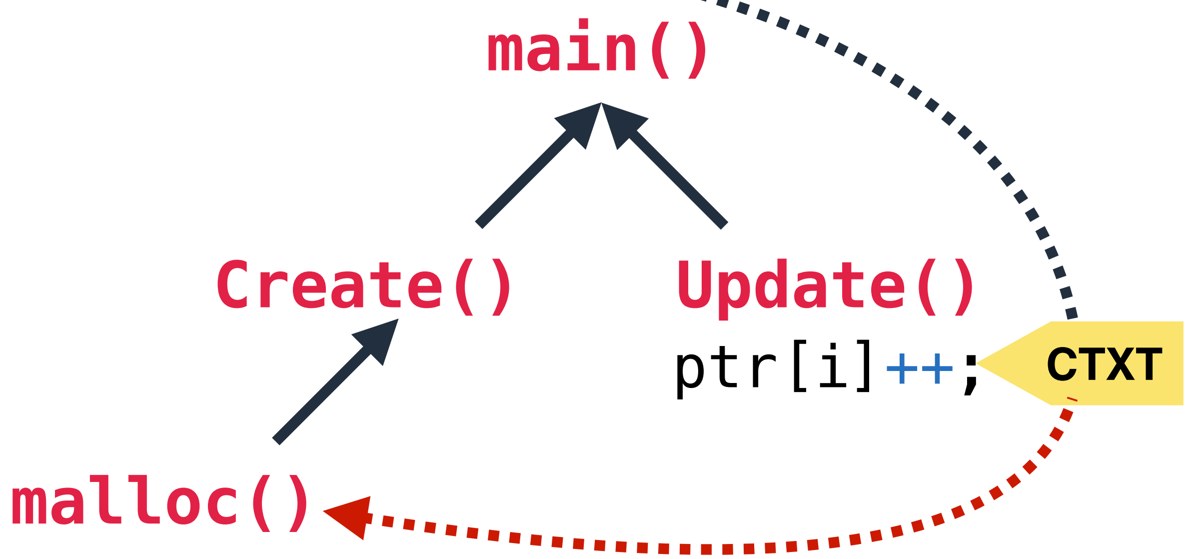


Associate each data access with its **data object**

- Dynamic allocation: **Call path of allocation site**

Data-Centric Attribution

```
int MyArray[s];

void Create(){
    return malloc();
}

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

void main(){
    int *m;
    p1 = Create();
    Update(p1);
    p2 = MyArray;
    Update(p);
}
```



main()

Create()

Update()

ptr[i]++;

CTXT

malloc()

Associate each data access with its **data object**

- Dynamic allocation: **Call path of allocation site**
- Static objects: **Variable name**

Data-Centric Attribution

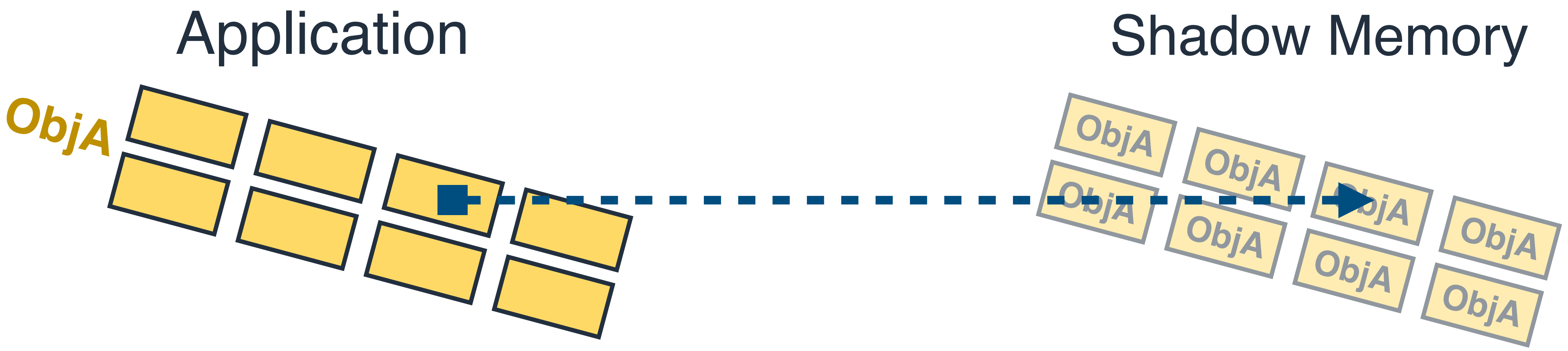
Implementation

Shadow memory

Data-Centric Attribution

Implementation

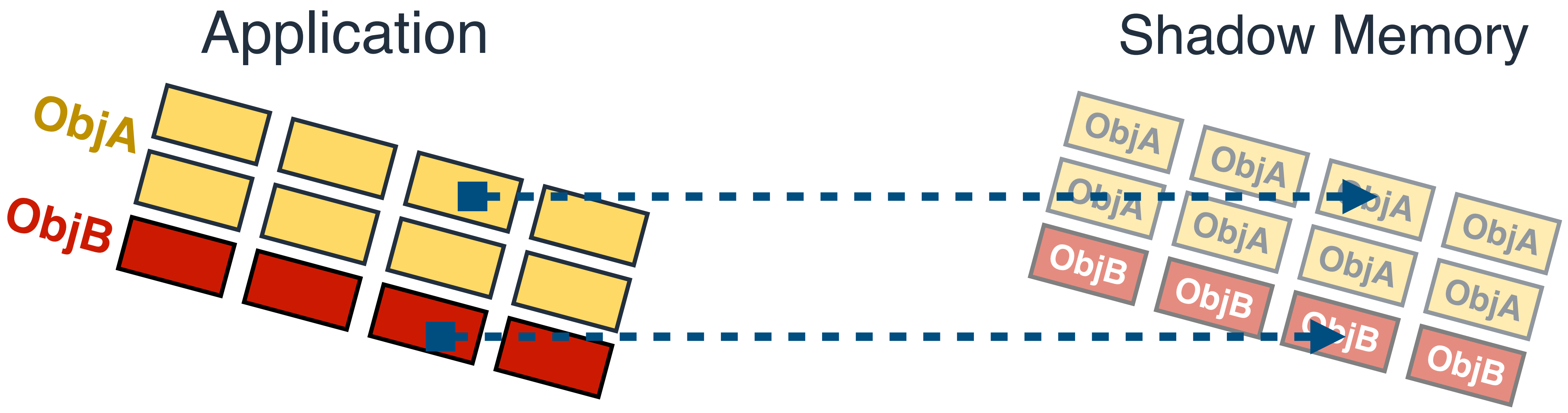
Shadow memory



Data-Centric Attribution

Implementation

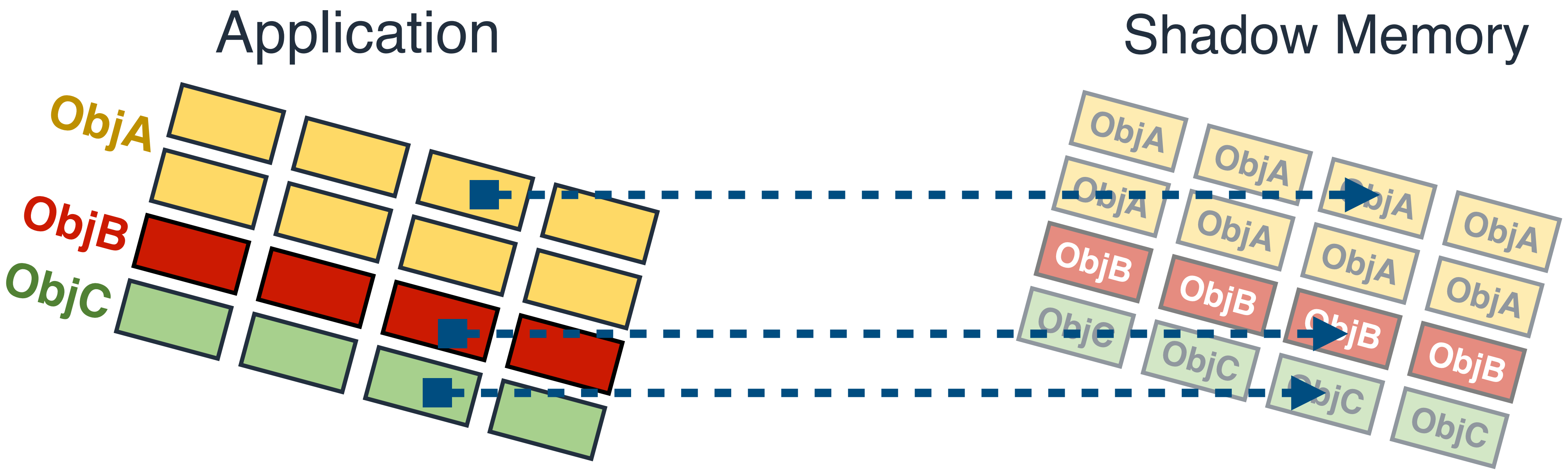
Shadow memory



Data-Centric Attribution

Implementation

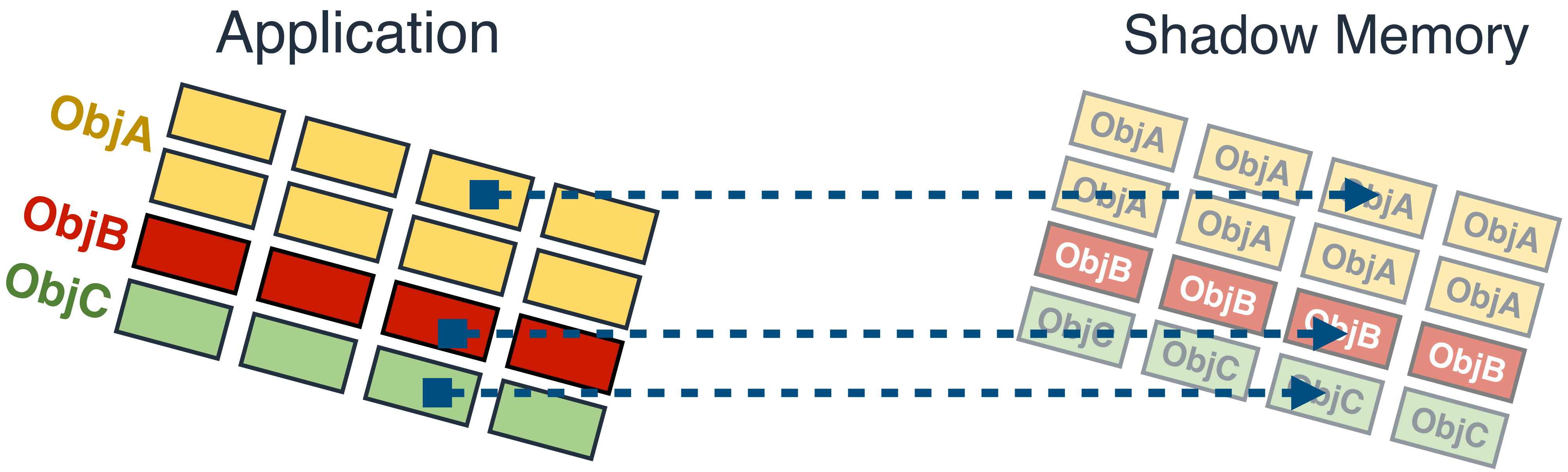
Shadow memory



Data-Centric Attribution

Implementation

Shadow memory



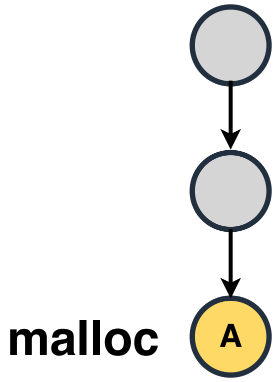
For each memory cell, a shadow cell holds a handle for the memory cell’s data object

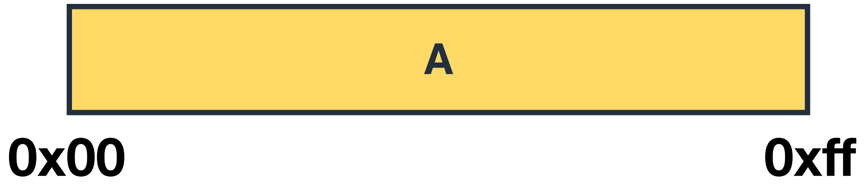
- Low lookup cost—**O(1)**
- Support **concurrent** access

Correlate Data-Attribution with Code-Attribution

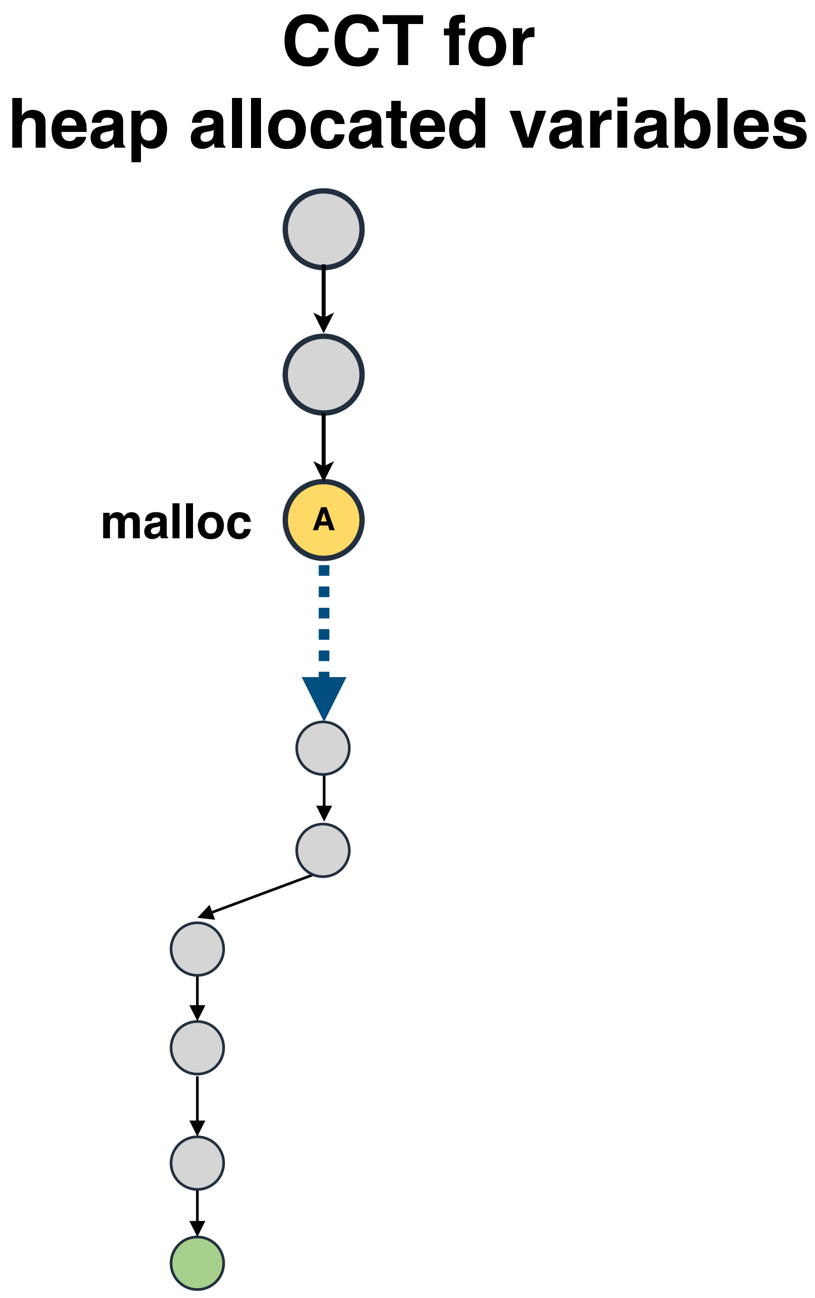
Correlate Data-Attribution with Code-Attribution

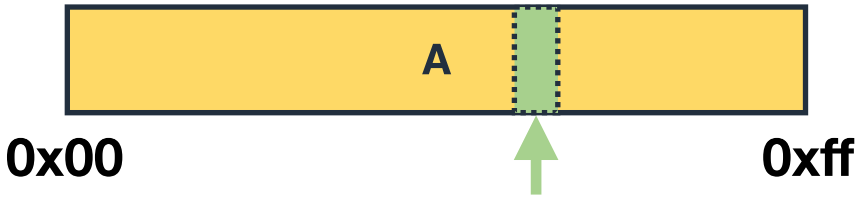
**CCT for
heap allocated variables**



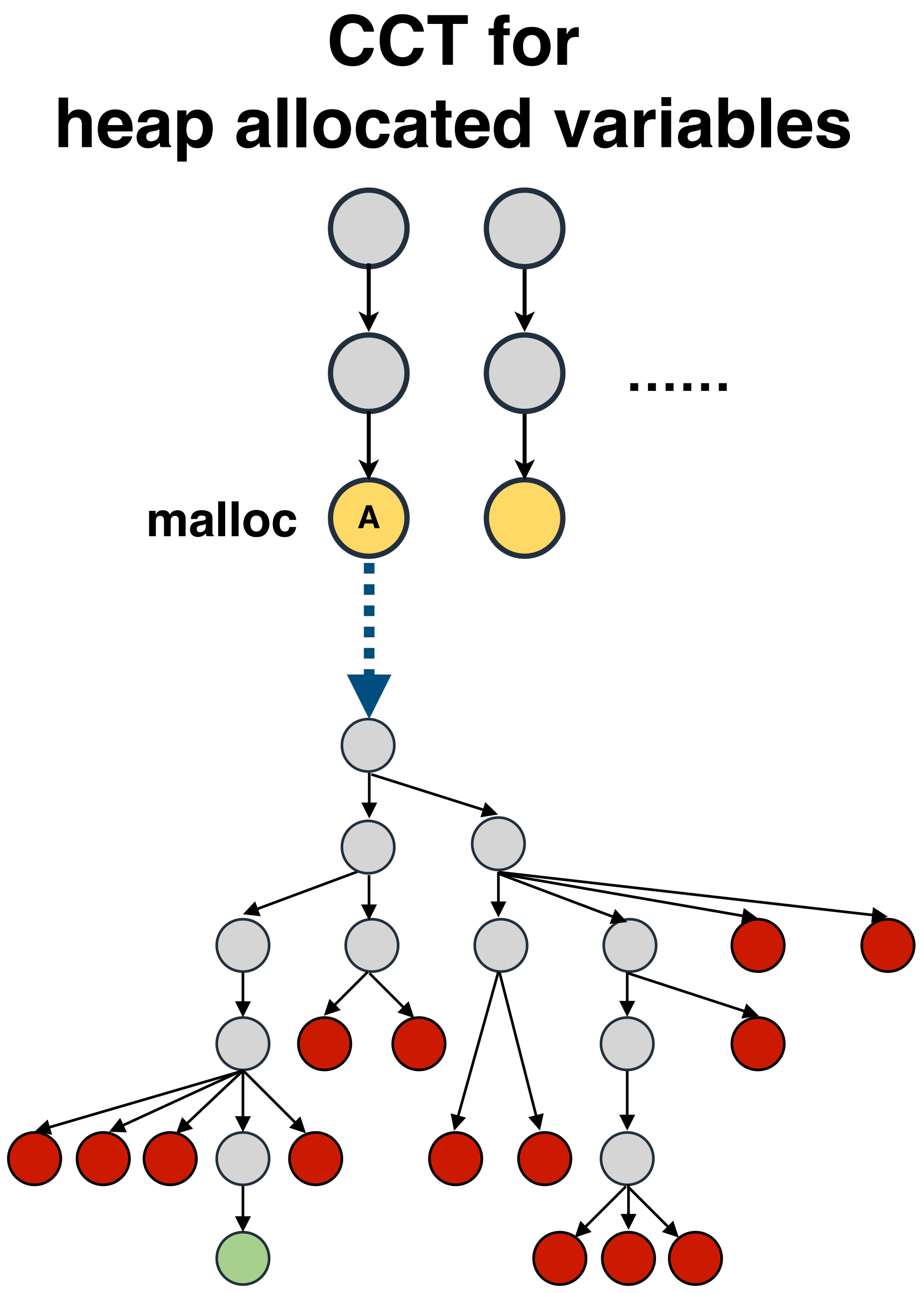


Correlate Data-Attribution with Code-Attribution



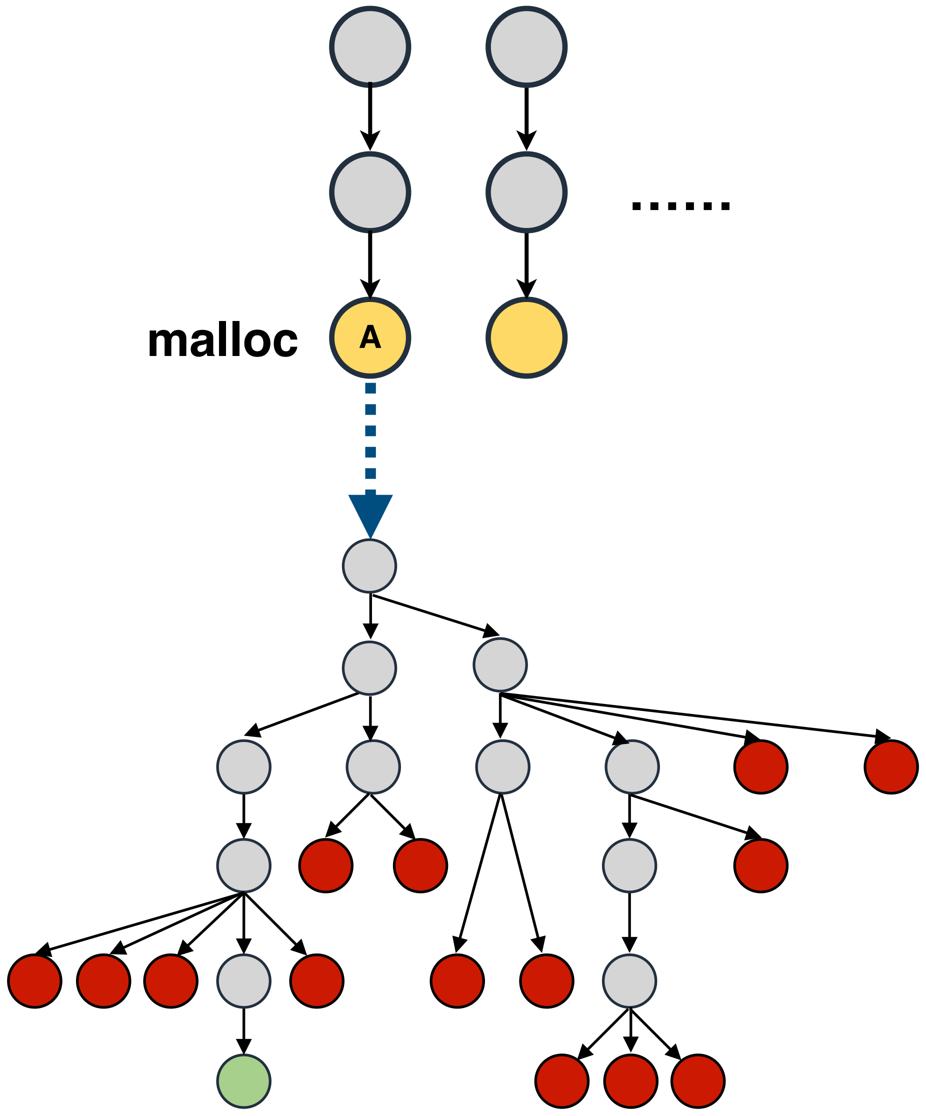


Correlate Data-Attribution with Code-Attribution

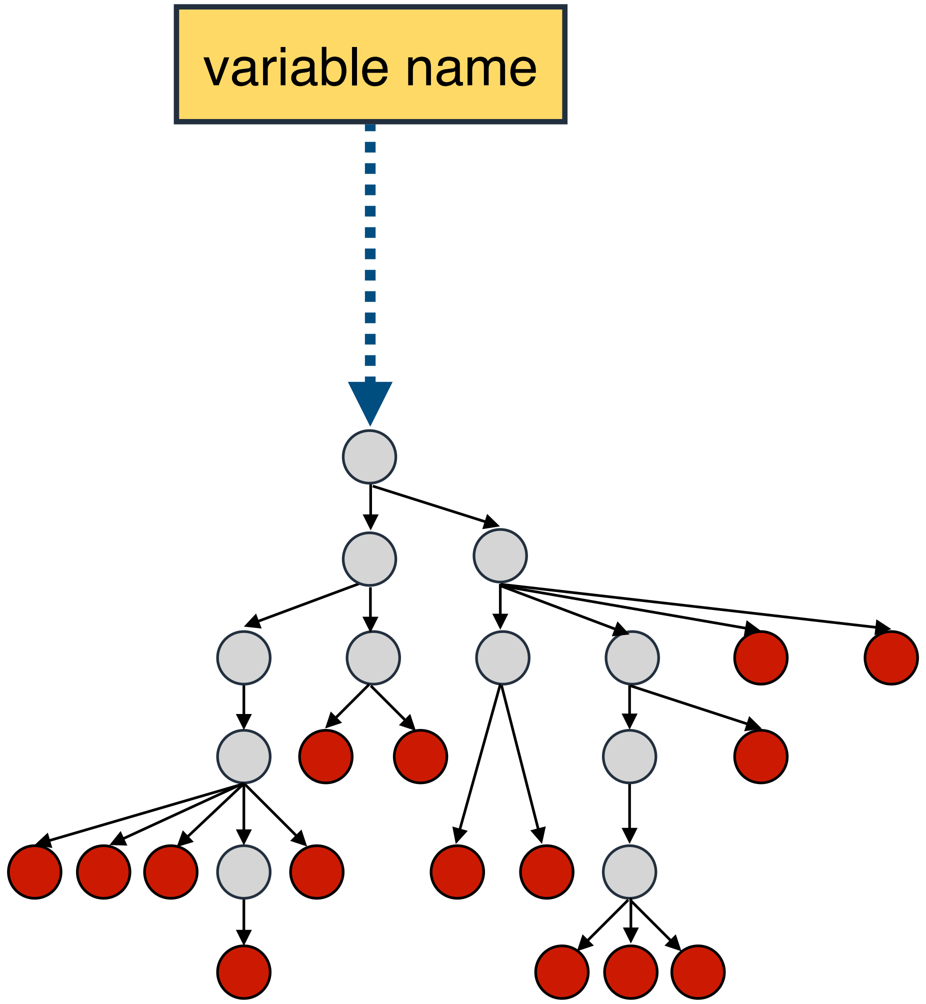


Correlate Data-Attribution with Code-Attribution

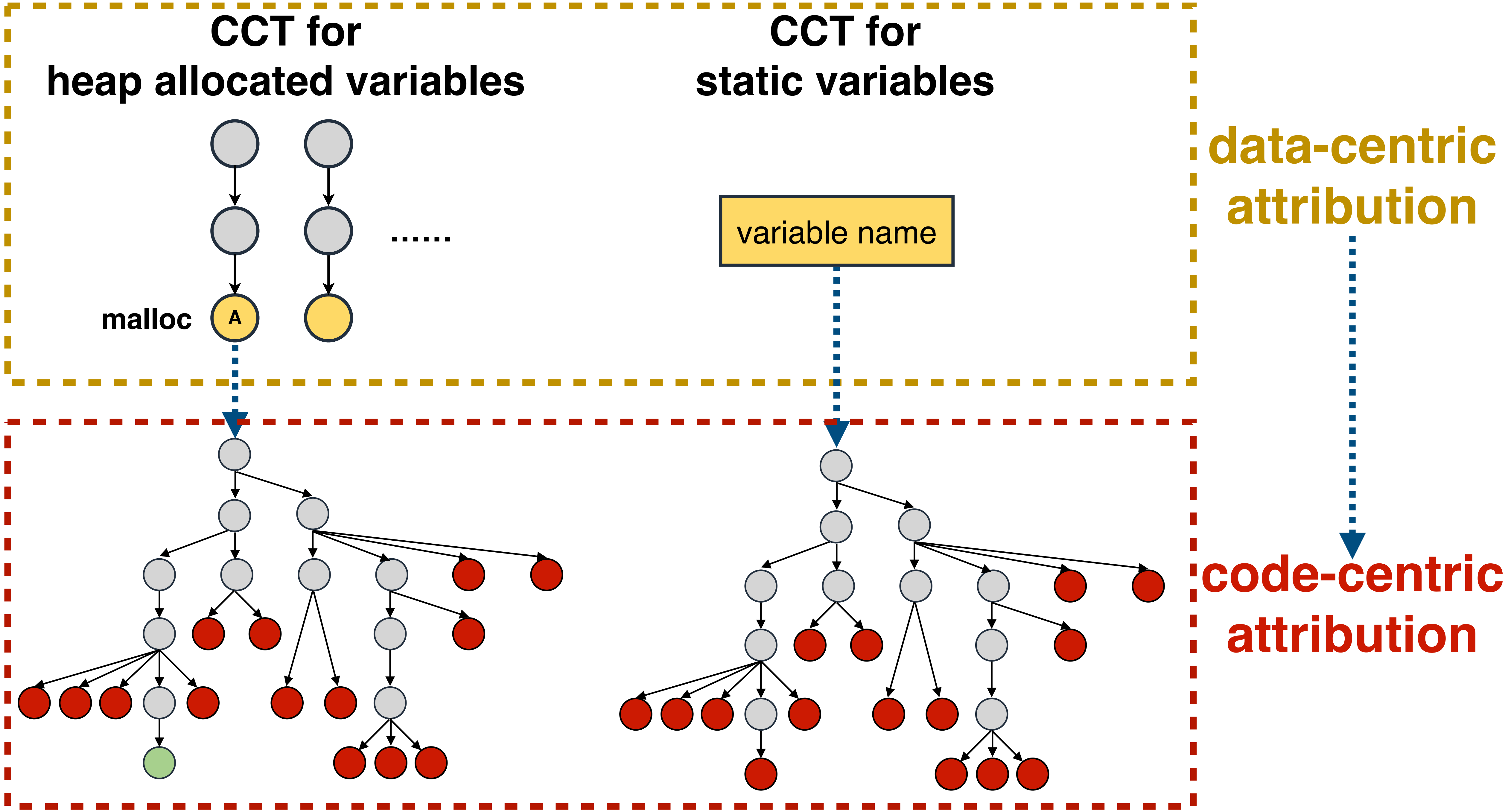
CCT for heap allocated variables



CCT for static variables



Correlate Data-Attribution with Code-Attribution



CCT for heap allocated variables

CCT for static variables

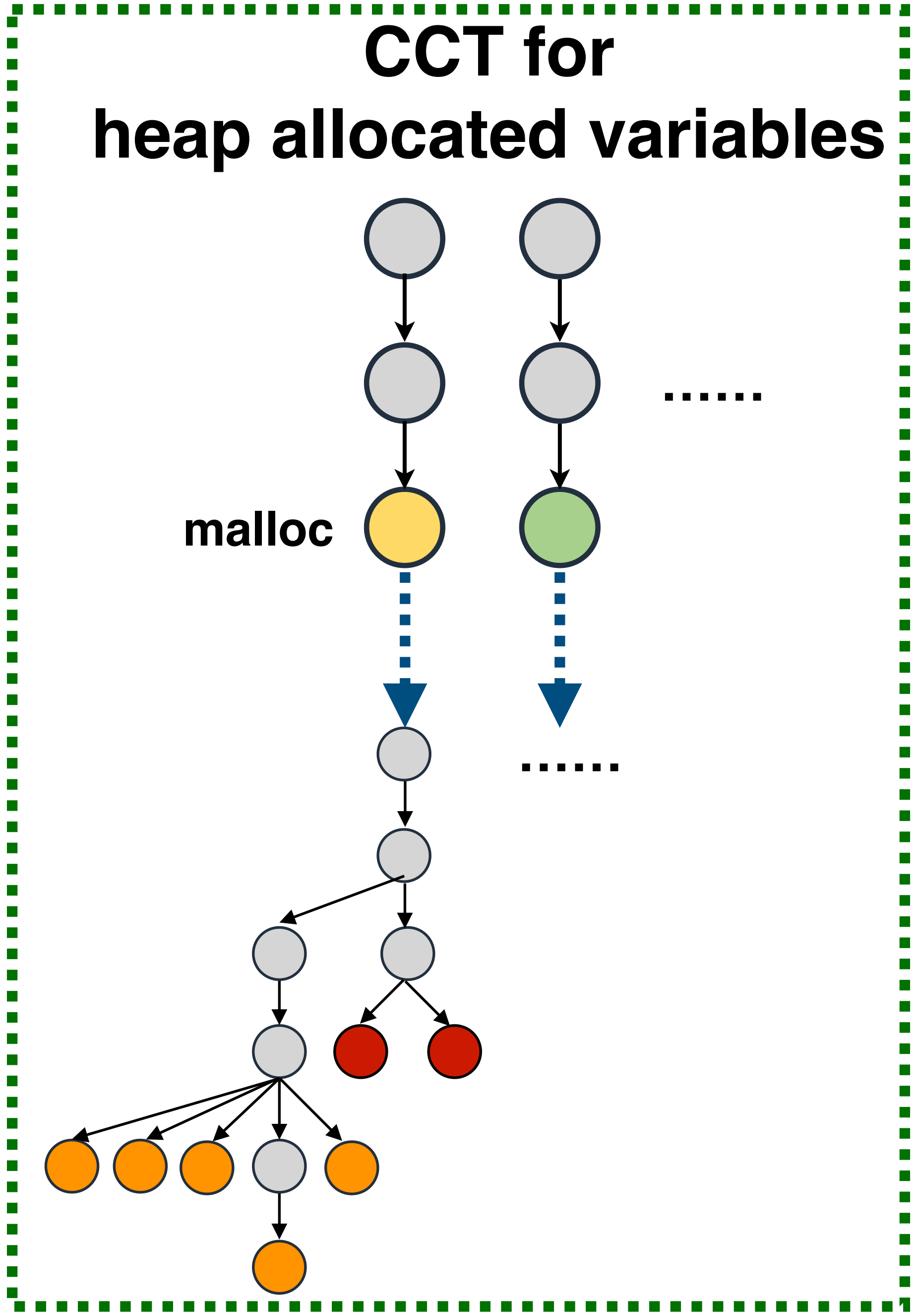
data-centric attribution

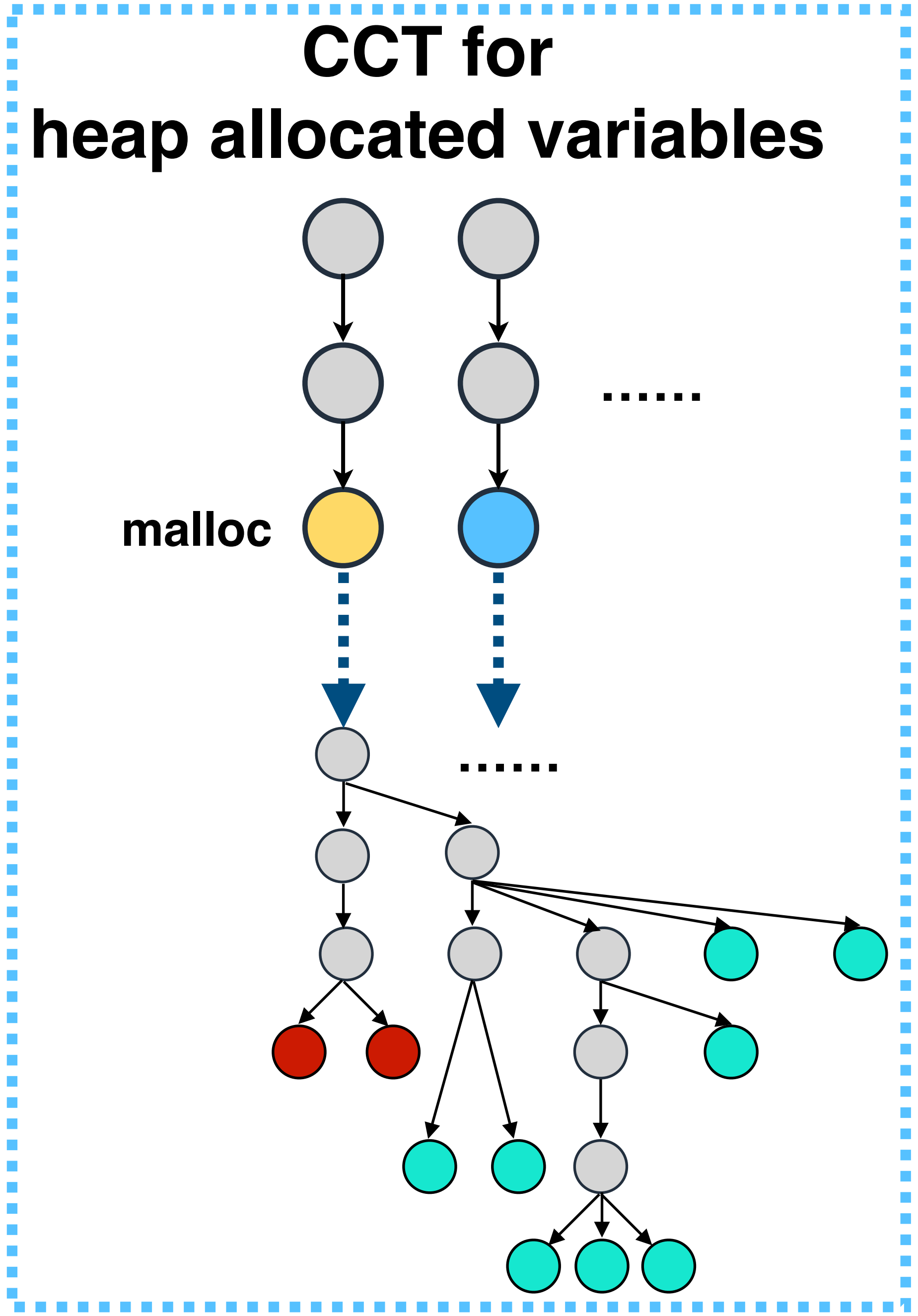
code-centric attribution

DrCCTProf

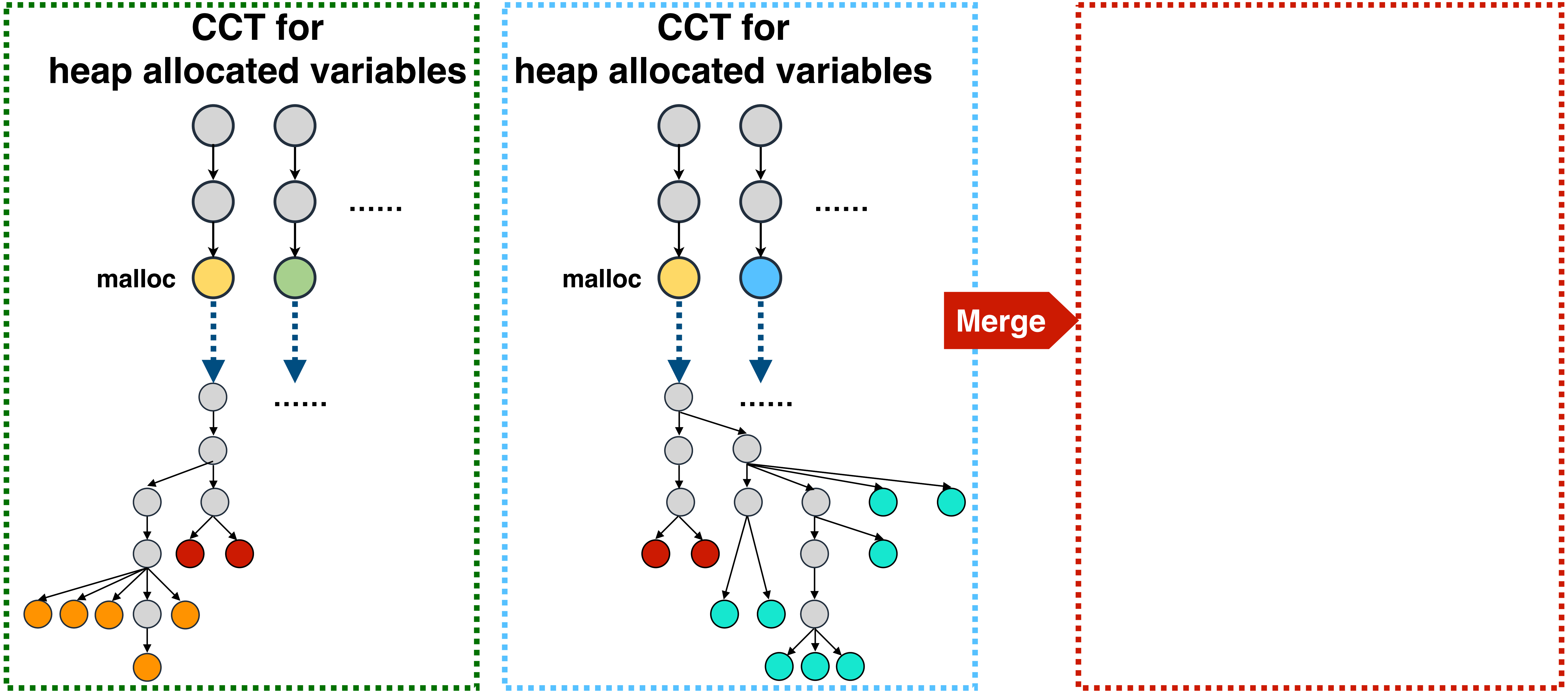
- Ubiquitous call path collection
- Attributing costs to data objects
- **Handling parallelism**
- Evaluation
- Case study
- Conclusions

Merge CCTs across Threads/Processes

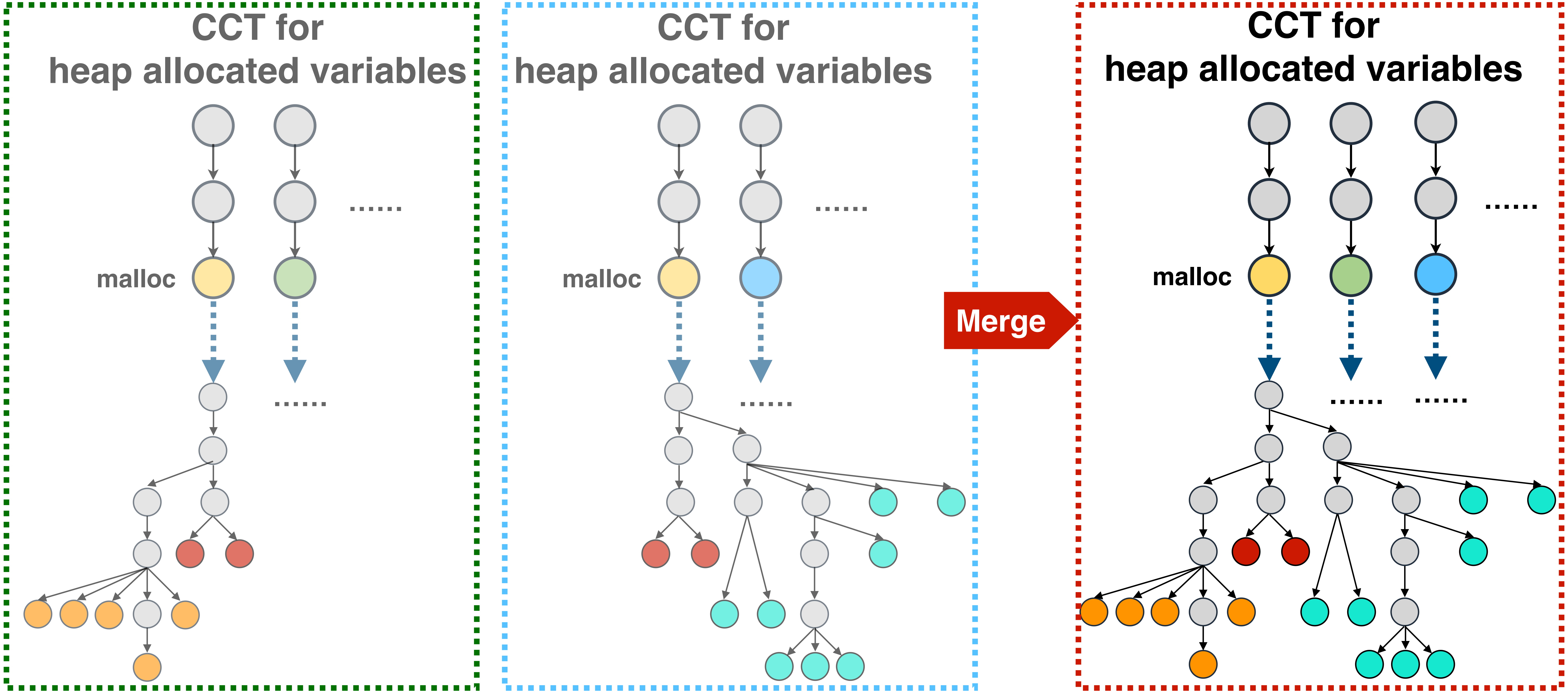




Merge CCTs across Threads/Processes



Merge CCTs across Threads/Processes



DrCCTProf

- Ubiquitous call path collection
- Attributing costs to data objects
- Merge attributions
- **Evaluation**
- Case study
- Conclusions

DrCCTProf Overhead

Machine configuration

Cluster

- Amazon Web Services (AWS)

Node

- Graviton2 CPU **32 ARMv8 cores** clocked at 2.3 GHz

Workloads

NPB-MPI

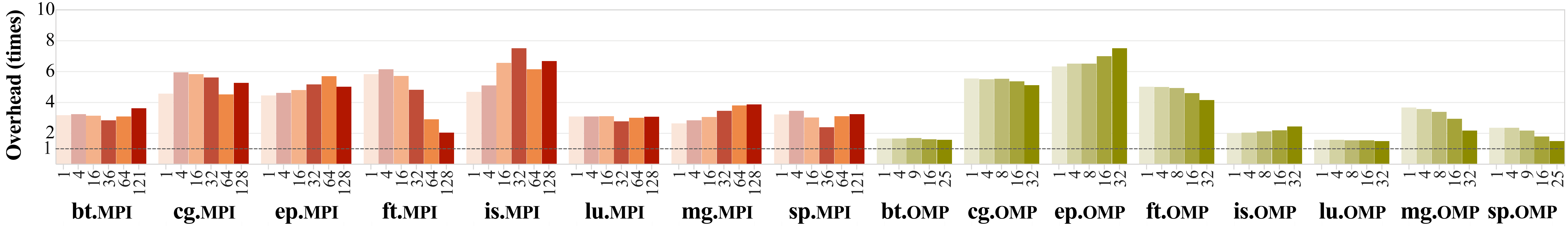
- **4 Nodes 128 processes in total**

NPB-OMP

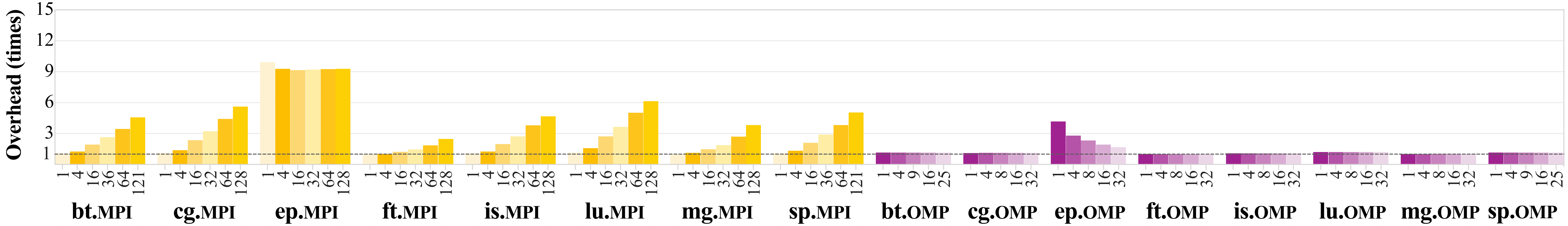
- **1 Node 32 threads in total**

NPB Code-Centric Overhead

Runtime Overhead [MPI 3.8×] [OMP 2.3×]

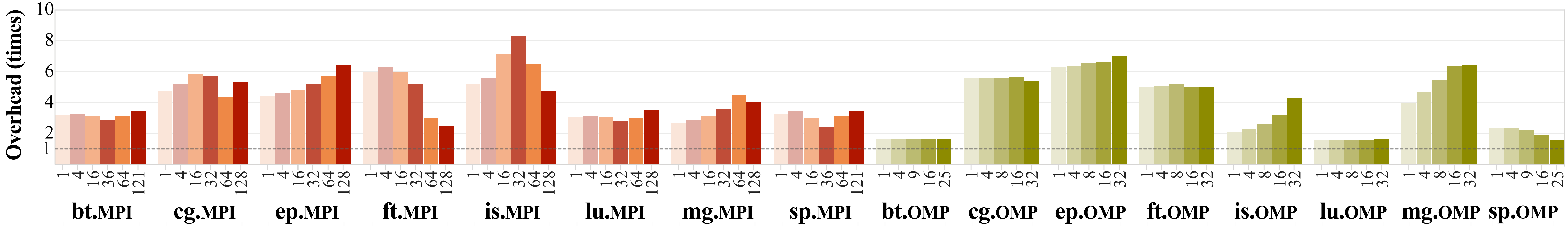


Space Overhead [MPI 4.9×] [OMP 1.2×]

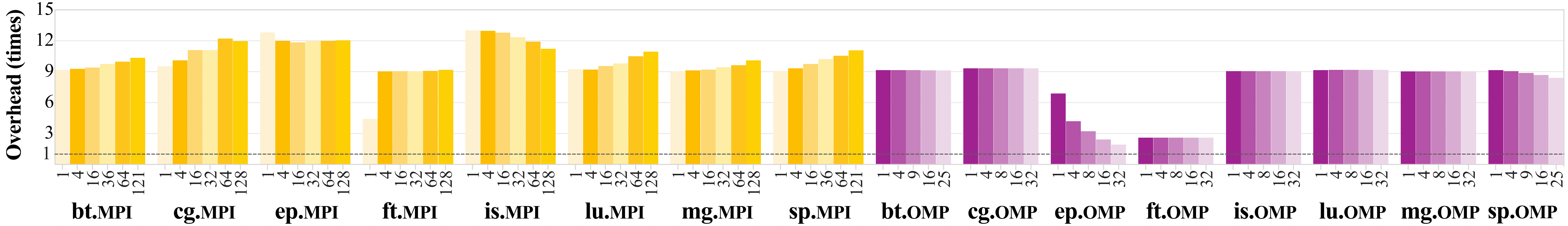


NPB Data-Centric Overhead

Runtime Overhead [MPI **3.8×**] [OMP **4.6×**]



Space Overhead [MPI **11×**] [OMP **9×**]



Real Application Overhead

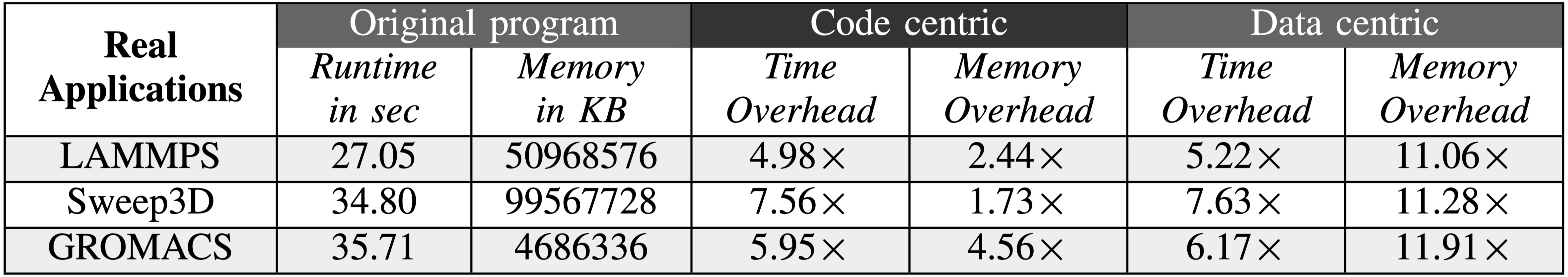
Real application configuration

LAMMPS, Sweep3D

- **16 Nodes** with **512 MPI processes**

GROMACS

- **16 Nodes** with **128 MPI processes** and **4 threads** in each process



Real Applications	Original program		Code centric		Data centric	
	Runtime in sec	*Memory in KB*	*Time Overhead*	*Memory Overhead*	*Time Overhead*	*Memory Overhead*
LAMMPS	27.05	50968576	4.98×	2.44×	5.22×	11.06×
Sweep3D	34.80	99567728	7.56×	1.73×	7.63×	11.28×
GROMACS	35.71	4686336	5.95×	4.56×	6.17×	11.91×

The runtime and memory overheads of real applications.

DrCCTProf

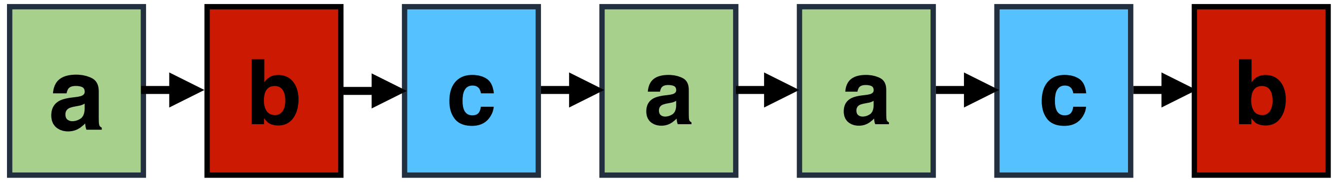
- Ubiquitous call path collection
- Attributing costs to data objects
- Merge attributions
- Evaluation
- **Case study**
- Conclusions

A DrCCTProf Client Tool: ARMREUSE

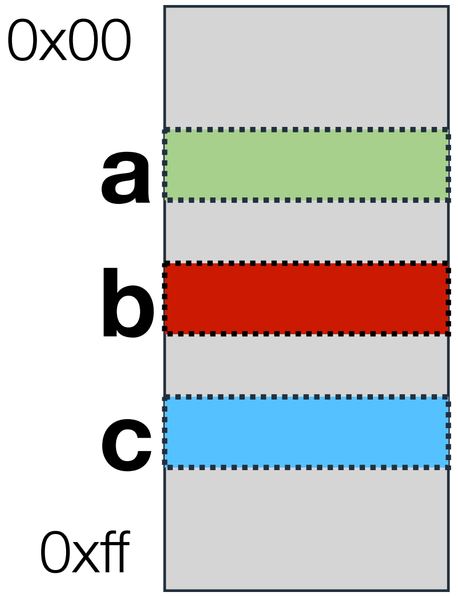
ARMREUSE

A DrCCTProf Client Tool: ARMREUSE

ARMREUSE



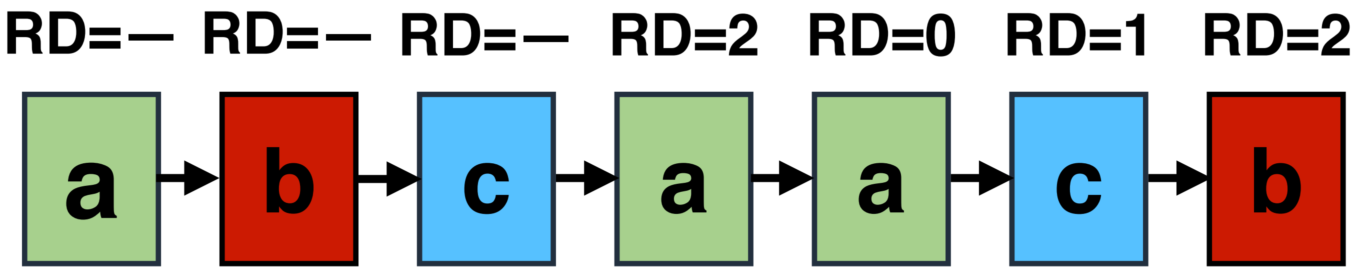
Memory Access Sequence



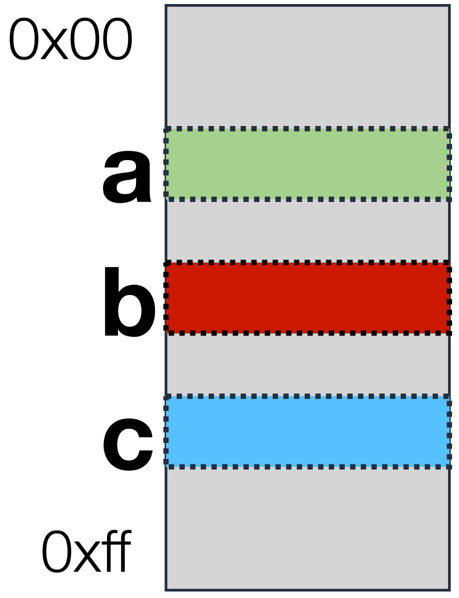
Memory

A DrCCTProf Client Tool: ARMREUSE

ARMREUSE



Memory Access Sequence

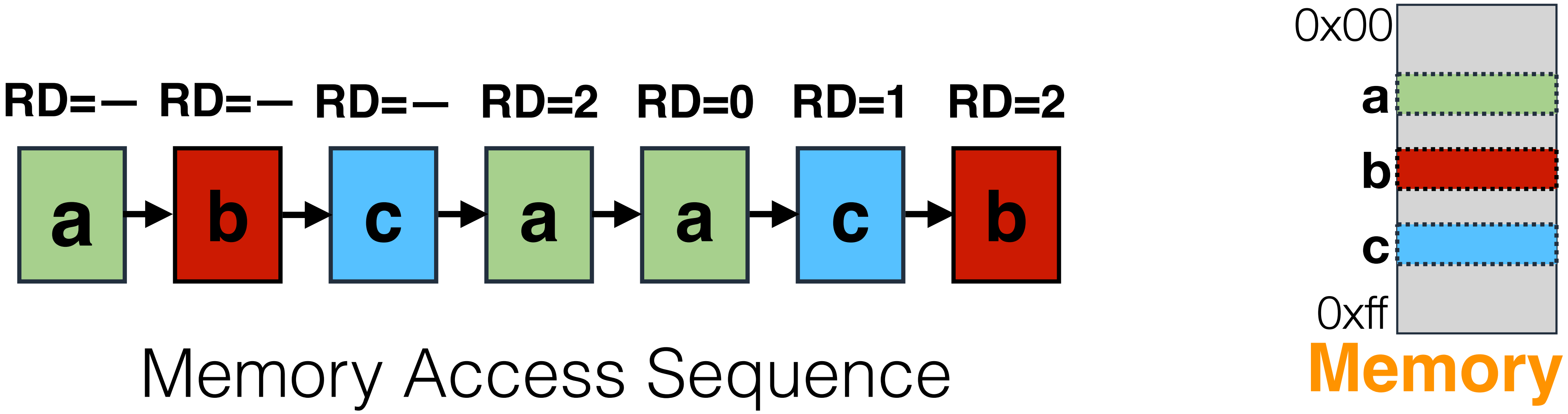


Memory

A DrCCTProf Client Tool: ARMREUSE

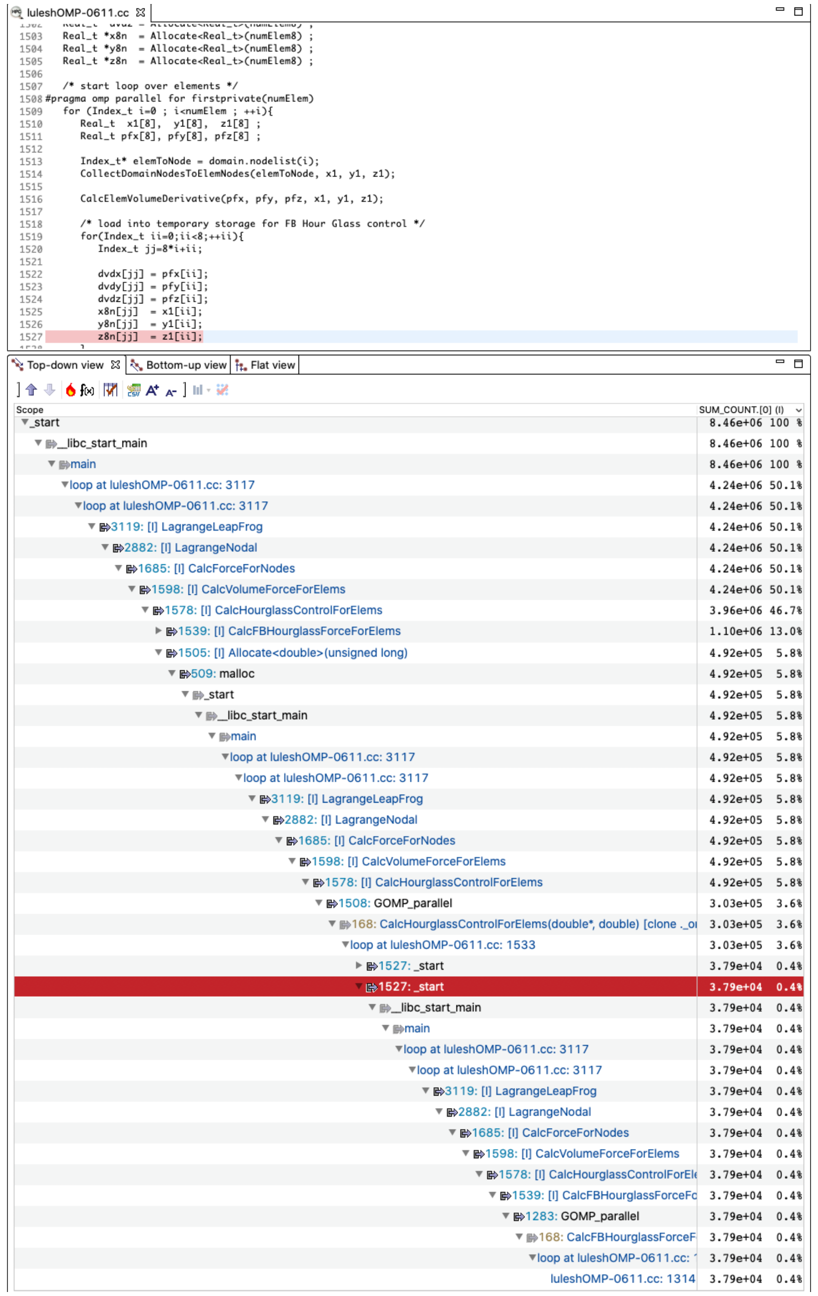
ARMREUSE

- Identifying temporal/spatial memory reuse pairs and computing reuse distances
- Obtaining rich insights: full call paths for use and reuse, data-centric attribution
- Providing intuitive analysis: GUI
- Implementing in an easy way: ~**500 lines** of code
- Guiding code optimization: LULESH, Sweep3D, E3SM ASME



Memory Access Sequence

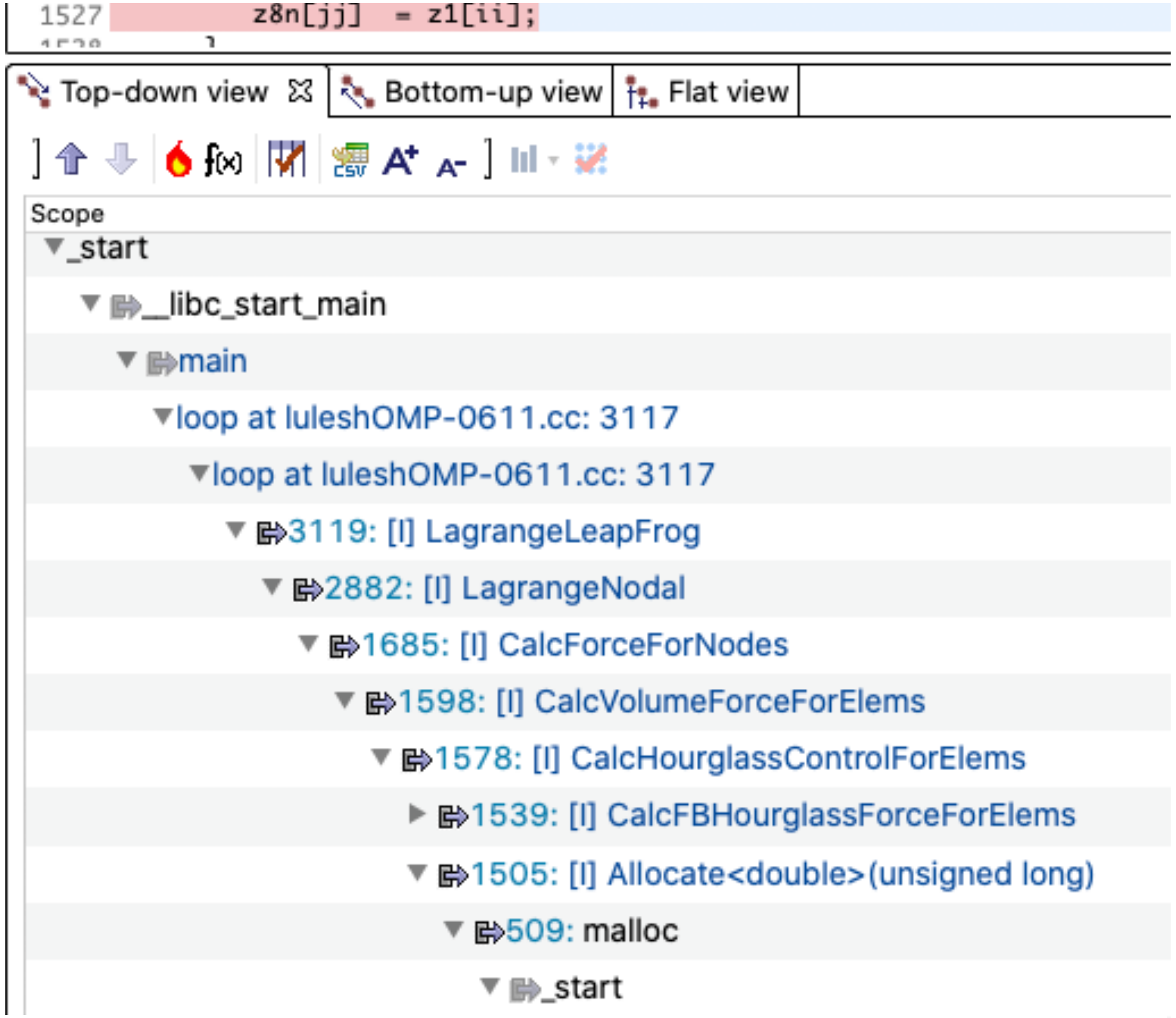
LULESH: An LLNL Proxy Application



```
   Real_t *x8n  = Allocate<Real_t>(numElem8) ;
   Real_t *y8n  = Allocate<Real_t>(numElem8) ;
   Real_t *z8n  = Allocate<Real_t>(numElem8) ;

   /* start loop over elements */
#pragma omp parallel for firstprivate(numElem)
   for (Index_t i=0 ; i<numElem ; ++i){
      Real_t  x1[8],  y1[8],  z1[8] ;
      Real_t pfx[8], pfy[8], pfz[8] ;

      Index_t* elemToNode = domain.nodelist(i);
      CollectDomainNodesToElemNodes(elemToNode, x1, y1, z1);

      CalcElemVolumeDerivative(pfx, pfy, pfz, x1, y1, z1);

      /* load into temporary storage for FB Hour Glass control */
      for(Index_t ii=0;ii<8;++ii){
         Index_t jj=8*i+ii;

         dvdx[jj] = pfx[ii];
         dvdy[jj] = pfy[ii];
         dvdz[jj] = pfz[ii];
         x8n[jj]  = x1[ii];
         y8n[jj]  = y1[ii];
         z8n[jj]  = z1[ii];
```

```
Scope                                                                      SUM_COUNT.[0] (I)
▼_start                                                                    8.46e+06 100 %
  ▼__libc_start_main                                                       8.46e+06 100 %
    ▼main                                                                  8.46e+06 100 %
      ▼loop at luleshOMP-0611.cc: 3117                                     4.24e+06 50.1%
        ▼loop at luleshOMP-0611.cc: 3117                                   4.24e+06 50.1%
          ▼3119: [I] LagrangeLeapFrog                                      4.24e+06 50.1%
            ▼2882: [I] LagrangeNodal                                       4.24e+06 50.1%
              ▼1685: [I] CalcForceForNodes                                 4.24e+06 50.1%
                ▼1598: [I] CalcVolumeForceForElems                         4.24e+06 50.1%
                  ▼1578: [I] CalcHourglassControlForElems                  3.96e+06 46.7%
                    ▶1539: [I] CalcFBHourglassForceForElems                1.10e+06 13.0%
                    ▼1505: [I] Allocate<double>(unsigned long)             4.92e+05  5.8%
                      ▼509: malloc                                         4.92e+05  5.8%
                        ▼_start                                            4.92e+05  5.8%
                          ▼__libc_start_main                               4.92e+05  5.8%
                            ▼main                                          4.92e+05  5.8%
                              ▼loop at luleshOMP-0611.cc: 3117             4.92e+05  5.8%
                                ▼loop at luleshOMP-0611.cc: 3117           4.92e+05  5.8%
                                  ▼3119: [I] LagrangeLeapFrog              4.92e+05  5.8%
                                    ▼2882: [I] LagrangeNodal               4.92e+05  5.8%
                                      ▼1685: [I] CalcForceForNodes         4.92e+05  5.8%
                                        ▼1598: [I] CalcVolumeForceForElems 4.92e+05  5.8%
                                          ▼1578: [I] CalcHourglassControlForElems 4.92e+05  5.8%
                                            ▼1508: GOMP_parallel           3.03e+05  3.6%
                                              ▼168: CalcHourglassControlForElems(double*, double) [clone ._o  3.03e+05  3.6%
                                                ▼loop at luleshOMP-0611.cc: 1533  3.03e+05  3.6%
                                                  ▶1527: _start            3.79e+04  0.4%
                                                  ▼1527: _start            3.79e+04  0.4%
                                                    ▼__libc_start_main     3.79e+04  0.4%
                                                      ▼main                3.79e+04  0.4%
                                                        ▼loop at luleshOMP-0611.cc: 3117  3.79e+04  0.4%
                                                          ▼loop at luleshOMP-0611.cc: 3117  3.79e+04  0.4%
                                                            ▼3119: [I] LagrangeLeapFrog  3.79e+04  0.4%
                                                              ▼2882: [I] LagrangeNodal  3.79e+04  0.4%
                                                                ▼1685: [I] CalcForceForNodes  3.79e+04  0.4%
                                                                  ▼1598: [I] CalcVolumeForceForElems  3.79e+04  0.4%
                                                                    ▼1578: [I] CalcHourglassControlForEl  3.79e+04  0.4%
                                                                      ▼1539: [I] CalcFBHourglassForceFo  3.79e+04  0.4%
                                                                        ▼1283: GOMP_parallel  3.79e+04  0.4%
                                                                          ▼168: CalcFBHourglassForceF  3.79e+04  0.4%
                                                                            ▼loop at luleshOMP-0611.cc:  3.79e+04  0.4%
                                                                               luleshOMP-0611.cc: 1314  3.79e+04  0.4%
```

LULESH: An LLNL Proxy Application



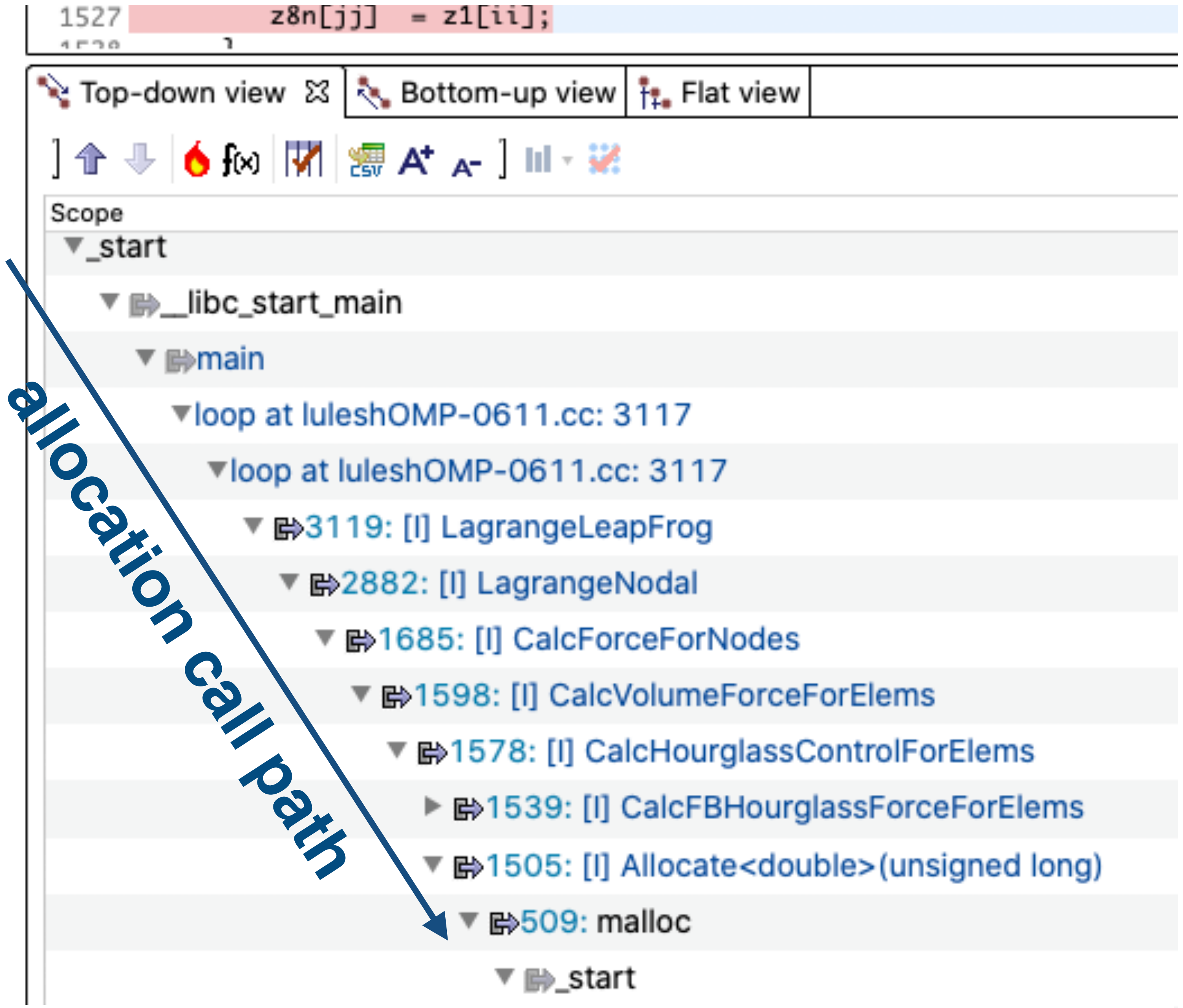
```
z8n[jj]  = z1[ii];
```

```
▼_start
  ▼ __libc_start_main
    ▼ main
      ▼loop at luleshOMP-0611.cc: 3117
        ▼loop at luleshOMP-0611.cc: 3117
          ▼ 3119: [I] LagrangeLeapFrog
            ▼ 2882: [I] LagrangeNodal
              ▼ 1685: [I] CalcForceForNodes
                ▼ 1598: [I] CalcVolumeForceForElems
                  ▼ 1578: [I] CalcHourglassControlForElems
                    ▶ 1539: [I] CalcFBHourglassForceForElems
                    ▼ 1505: [I] Allocate<double>(unsigned long)
                      ▼ 509: malloc
                        ▼ _start
```

LULESH: An LLNL Proxy Application

Allocation Call Path

The top reuse pairs on ***z8n*** accounts for 3.6% of total temporal reuse.



```
1527             z8n[jj]  = z1[ii];
```

```
▼_start
  ▼ __libc_start_main
    ▼ main
      ▼loop at luleshOMP-0611.cc: 3117
        ▼loop at luleshOMP-0611.cc: 3117
          ▼ 3119: [I] LagrangeLeapFrog
            ▼ 2882: [I] LagrangeNodal
              ▼ 1685: [I] CalcForceForNodes
                ▼ 1598: [I] CalcVolumeForceForElems
                  ▼ 1578: [I] CalcHourglassControlForElems
                    ▶ 1539: [I] CalcFBHourglassForceForElems
                    ▼ 1505: [I] Allocate<double>(unsigned long)
                      ▼ 509: malloc
                        ▼ _start
```

allocation call path

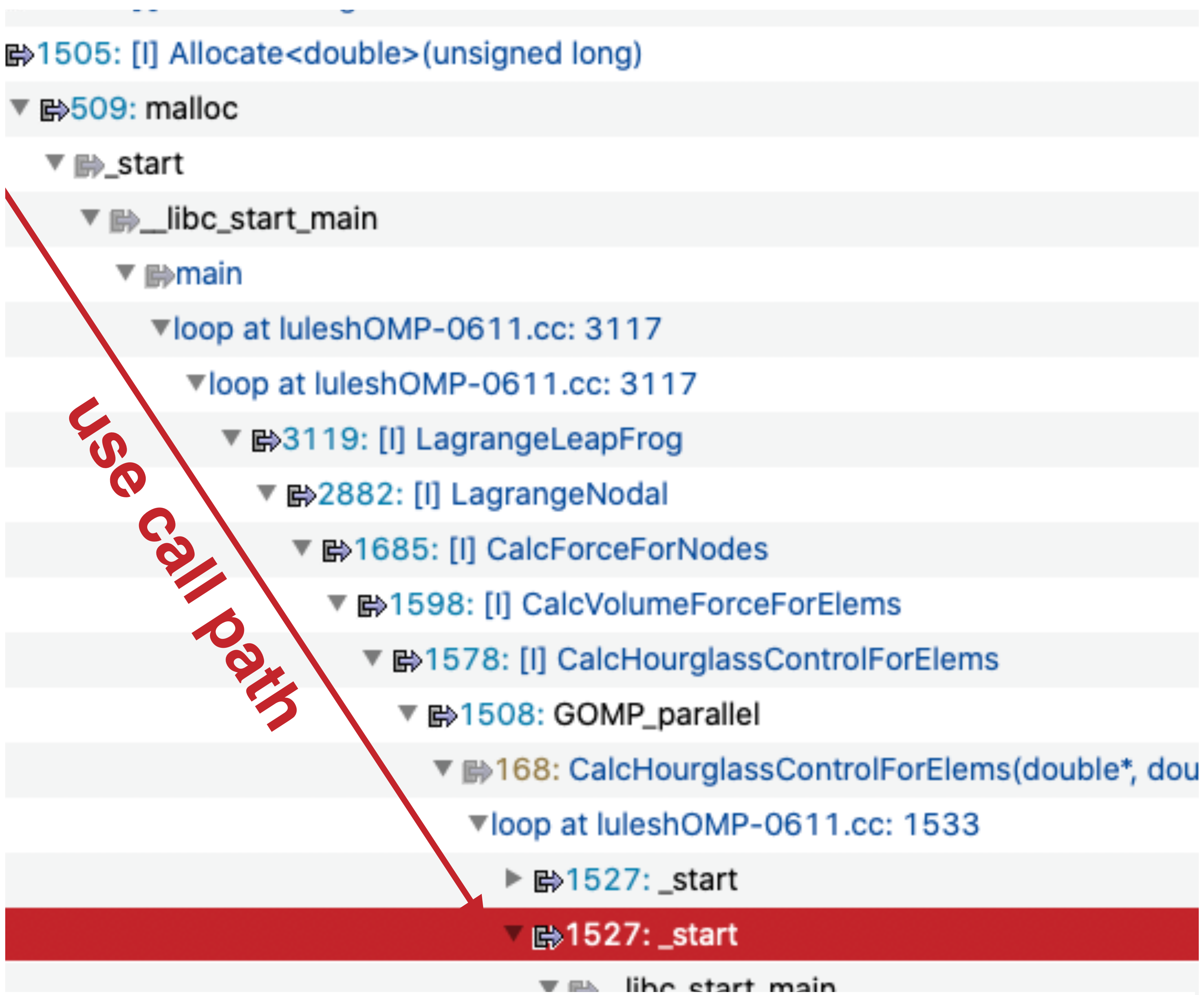
LULESH: An LLNL Proxy Application

Allocation Call Path

The top reuse pairs on ***z8n*** accounts for 3.6% of total temporal reuse.

Use Call Path

The use is the memory access at line 1527 (in the loop at line 1509)



```
1505: [I] Allocate<double>(unsigned long)
▼ 509: malloc
  ▼ _start
    ▼ __libc_start_main
      ▼ main
        ▼loop at luleshOMP-0611.cc: 3117
          ▼loop at luleshOMP-0611.cc: 3117
            ▼ 3119: [I] LagrangeLeapFrog
              ▼ 2882: [I] LagrangeNodal
                ▼ 1685: [I] CalcForceForNodes
                  ▼ 1598: [I] CalcVolumeForceForElems
                    ▼ 1578: [I] CalcHourglassControlForElems
                      ▼ 1508: GOMP_parallel
                        ▼ 168: CalcHourglassControlForElems(double*, dou
                          ▼loop at luleshOMP-0611.cc: 1533
                            ▶ 1527: _start
                            ▼ 1527: _start
                              ▼ __libc_start_main
```

LULESH: An LLNL Proxy Application

Allocation Call Path

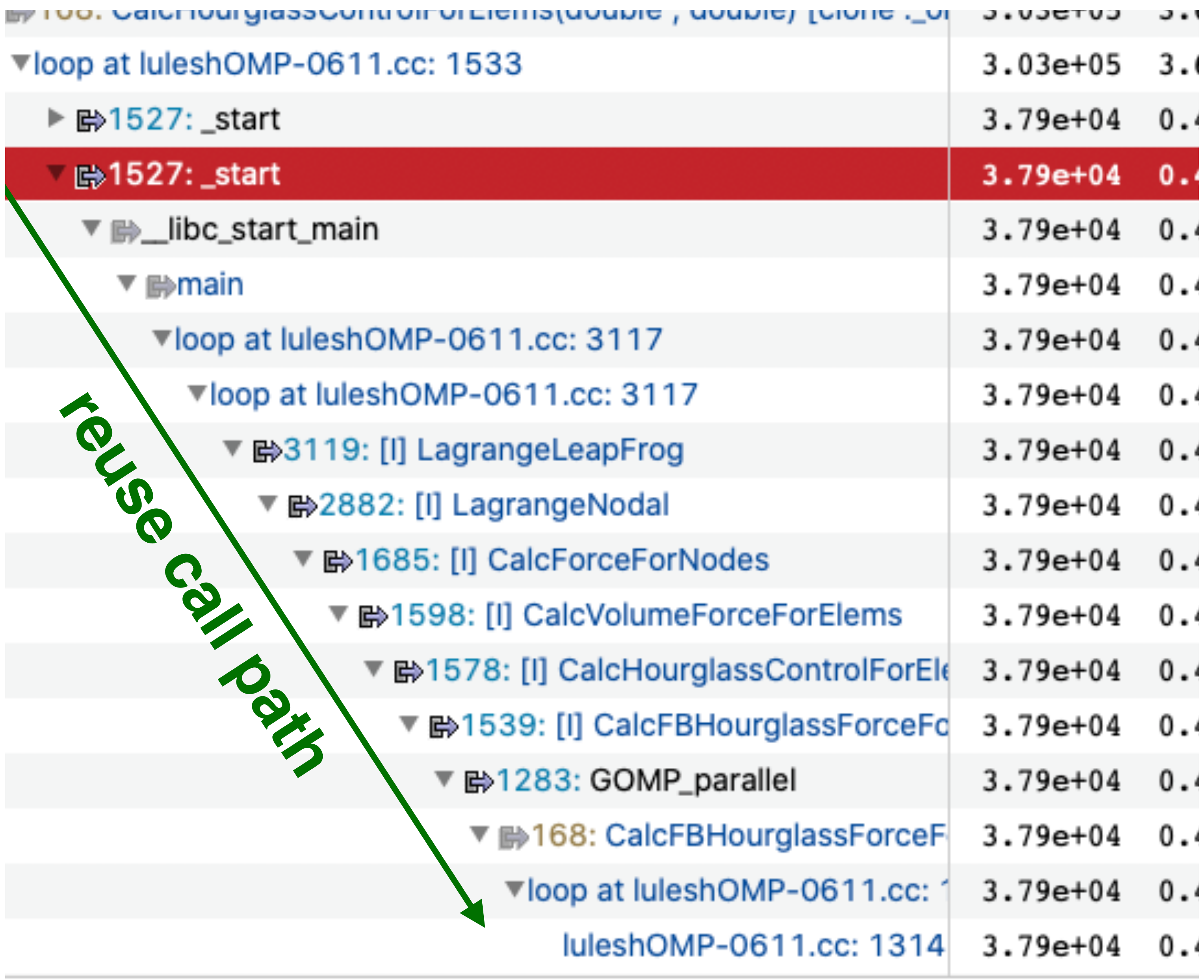
The top reuse pairs on ***z8n*** accounts for 3.6% of total temporal reuse.

Use Call Path

The use is the memory access at line 1527 (in the loop at line 1509)

Reuse Call Path

The reuse is the memory access at line 1314 (in the loop at line 1284)



```
▼loop at luleshOMP-0611.cc: 1533                     3.03e+05  3.
  ▶ ⇨1527: _start                                    3.79e+04  0.
  ▼ ⇨1527: _start                                    3.79e+04  0.
    ▼ ⇨__libc_start_main                             3.79e+04  0.
      ▼ ⇨main                                        3.79e+04  0.
        ▼loop at luleshOMP-0611.cc: 3117             3.79e+04  0.
          ▼loop at luleshOMP-0611.cc: 3117           3.79e+04  0.
            ▼ ⇨3119: [I] LagrangeLeapFrog            3.79e+04  0.
              ▼ ⇨2882: [I] LagrangeNodal             3.79e+04  0.
                ▼ ⇨1685: [I] CalcForceForNodes       3.79e+04  0.
                  ▼ ⇨1598: [I] CalcVolumeForceForElems 3.79e+04  0.
                    ▼ ⇨1578: [I] CalcHourglassControlForEle 3.79e+04  0.
                      ▼ ⇨1539: [I] CalcFBHourglassForceFo 3.79e+04  0.
                        ▼ ⇨1283: GOMP_parallel       3.79e+04  0.
                          ▼ ⇨168: CalcFBHourglassForceF 3.79e+04  0.
                            ▼loop at luleshOMP-0611.cc:  3.79e+04  0.
                               luleshOMP-0611.cc: 1314  3.79e+04  0.
```

LULESH: An LLNL Proxy Application

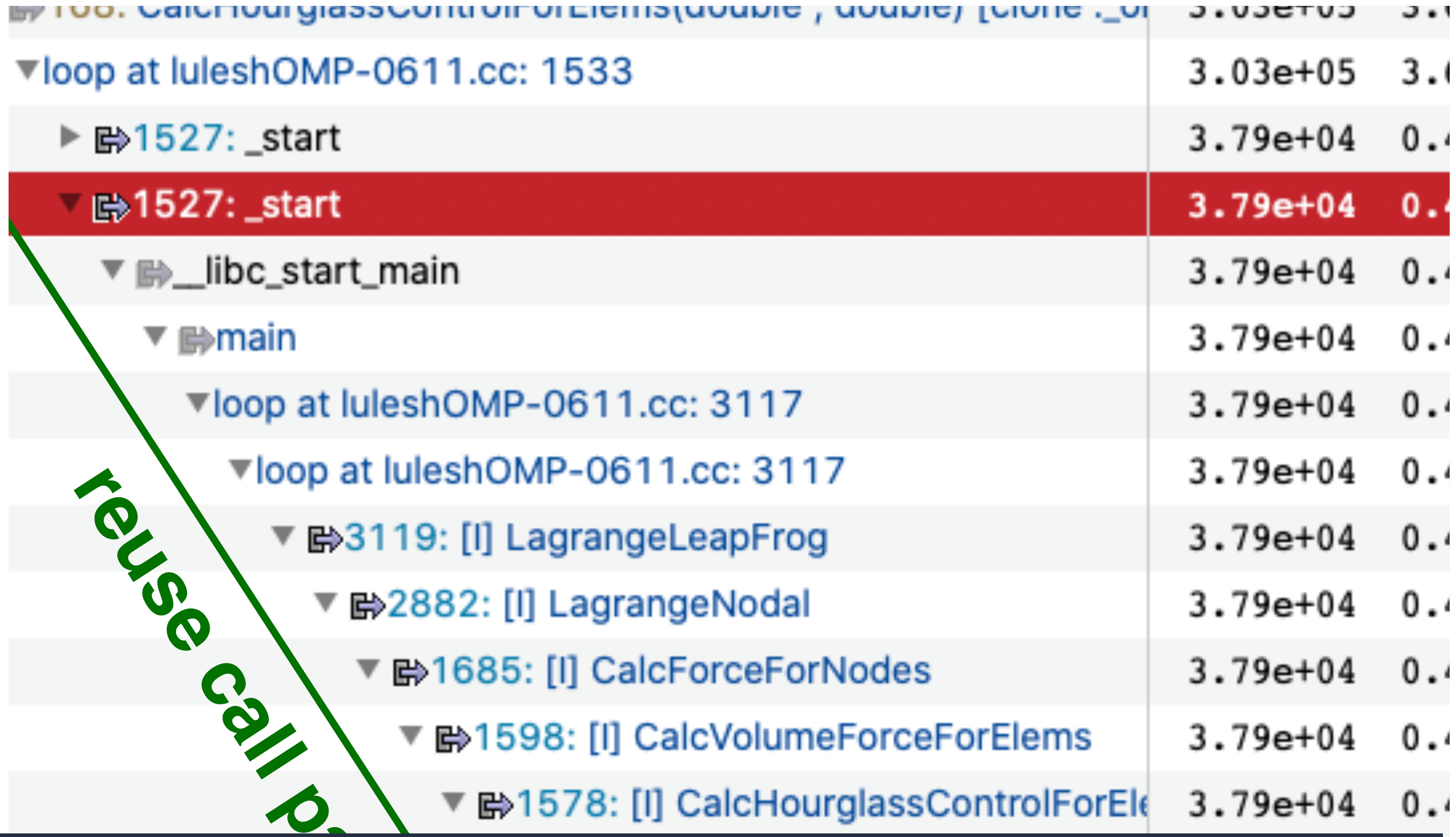
Allocation Call Path

The top reuse pairs on ***z8n*** accounts for 3.6% of total temporal reuse.

Use Call Path

The use is the memory access at line 1527 (in the loop at line 1509)

Reuse Call Path



```
▼loop at luleshOMP-0611.cc: 1533                           3.03e+05   3.
  ▶ 1527: _start                                           3.79e+04   0.
  ▼ 1527: _start                                           3.79e+04   0.
    ▼ __libc_start_main                                    3.79e+04   0.
      ▼ main                                               3.79e+04   0.
        ▼loop at luleshOMP-0611.cc: 3117                   3.79e+04   0.
          ▼loop at luleshOMP-0611.cc: 3117                 3.79e+04   0.
            ▼ 3119: [I] LagrangeLeapFrog                   3.79e+04   0.
              ▼ 2882: [I] LagrangeNodal                    3.79e+04   0.
                ▼ 1685: [I] CalcForceForNodes              3.79e+04   0.
                  ▼ 1598: [I] CalcVolumeForceForElems      3.79e+04   0.
                    ▼ 1578: [I] CalcHourglassControlForEl  3.79e+04   0.
```

Hoist the two loops (line 1508 and 1283) into their least common ancestor in the call paths and fuse them

1.28x speedup

DrCCTProf

- Ubiquitous call path collection
- Attributing costs to data objects
- Merge attributions
- Evaluation
- Case study
- **Conclusions**

Conclusions

DrCCTProf

- A practical fine-grained call path profiler framework for ARM/x86 binaries
- Strong support for various analysis tools
- Moderate time and memory overheads
- Applicable to large-scale executions

Open source with MIT license

https://github.com/Xuhpclab/DrCCTProf

On-Going Work (Will Release Soon)

Supporting various binaries

- Python
- Go
- Rust
- …

Supporting accelerators

- X86+GPU, ARM+GPU

NC STATE
UNIVERSITY

Q&A

https://github.com/Xuhpclab/DrCCTProf

Hands-on Lab

Machine access

```
ssh drcctprof1@rocco.cs.wm.edu -p 11111
Password: *drcctproftest1#
```

Create your own directory under this account

```
mkdir XX
cd XX
```

Download and build DrCCTProf

```
git clone --recurse https://github.com/Xuhpclab/drcctprof_tutorial.git
cd drcctprof_tutorial
./build.sh
```

Develop the client

src/client.cpp: 42 - 55