Call Paths for Pin Tools

Milind Chabbi, Xu Liu, and John Mellor-Crummey
Department of Computer Science
Rice University

Comp 600, January 27, 2014

Star Graduate Student

Star Graduate Student

Invention!

Data race detection tool

Inventor

Found Client for His Tool

Data race detection tool

Inventor

User

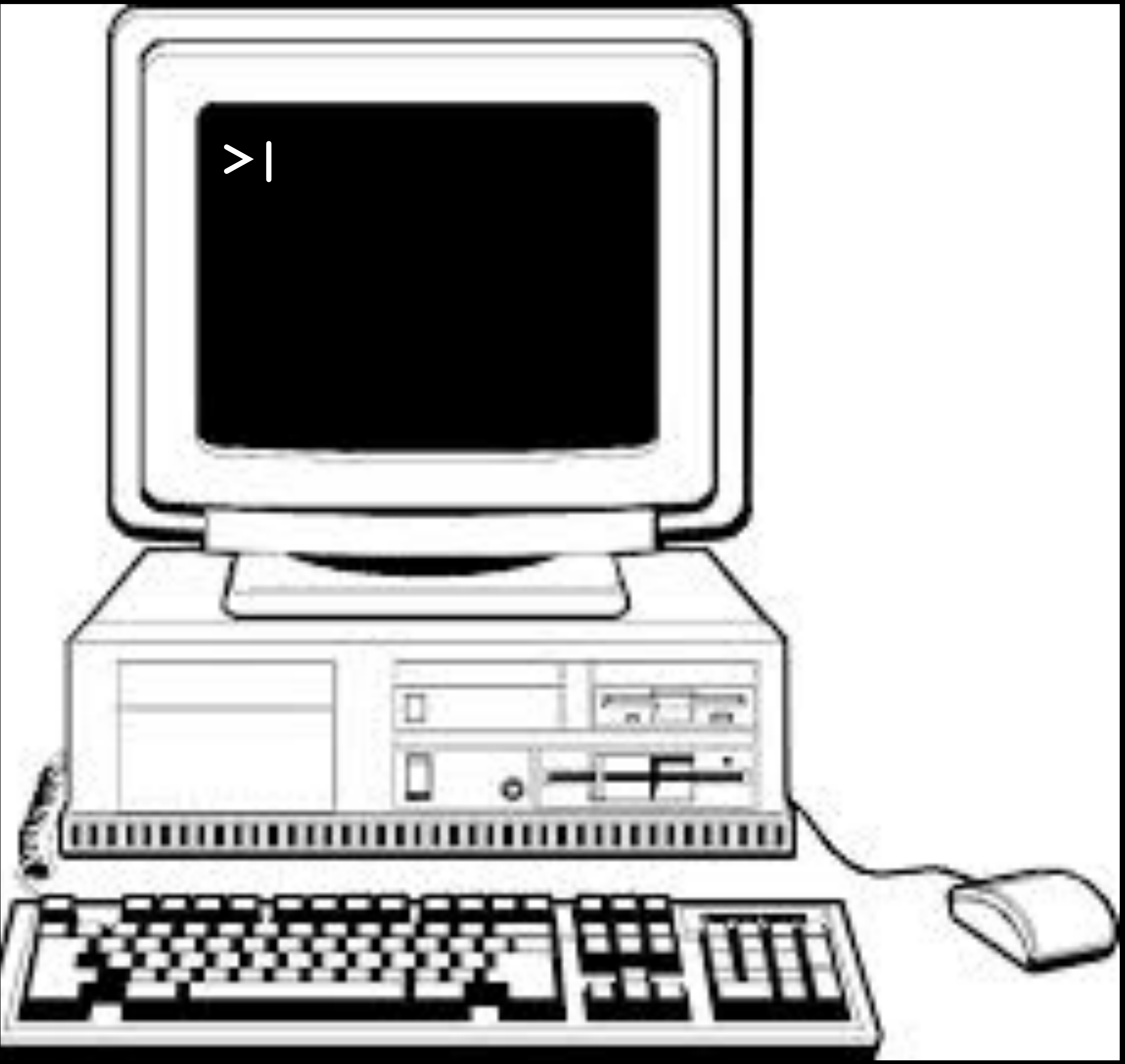
Found Client for His Tool

Data race detection tool

User

Client Uses the Tool

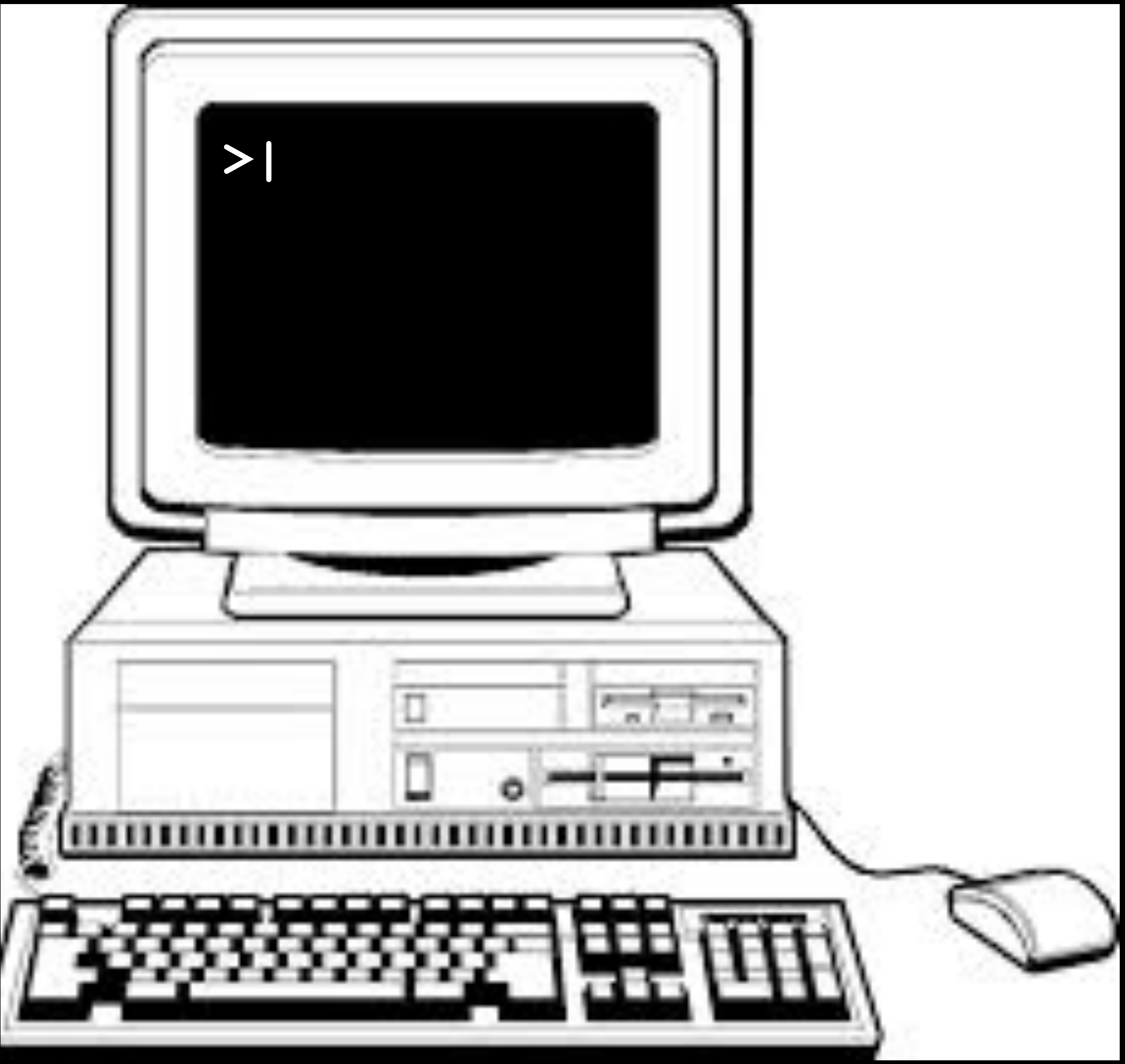
Data race detection tool



User

Client Uses the Tool

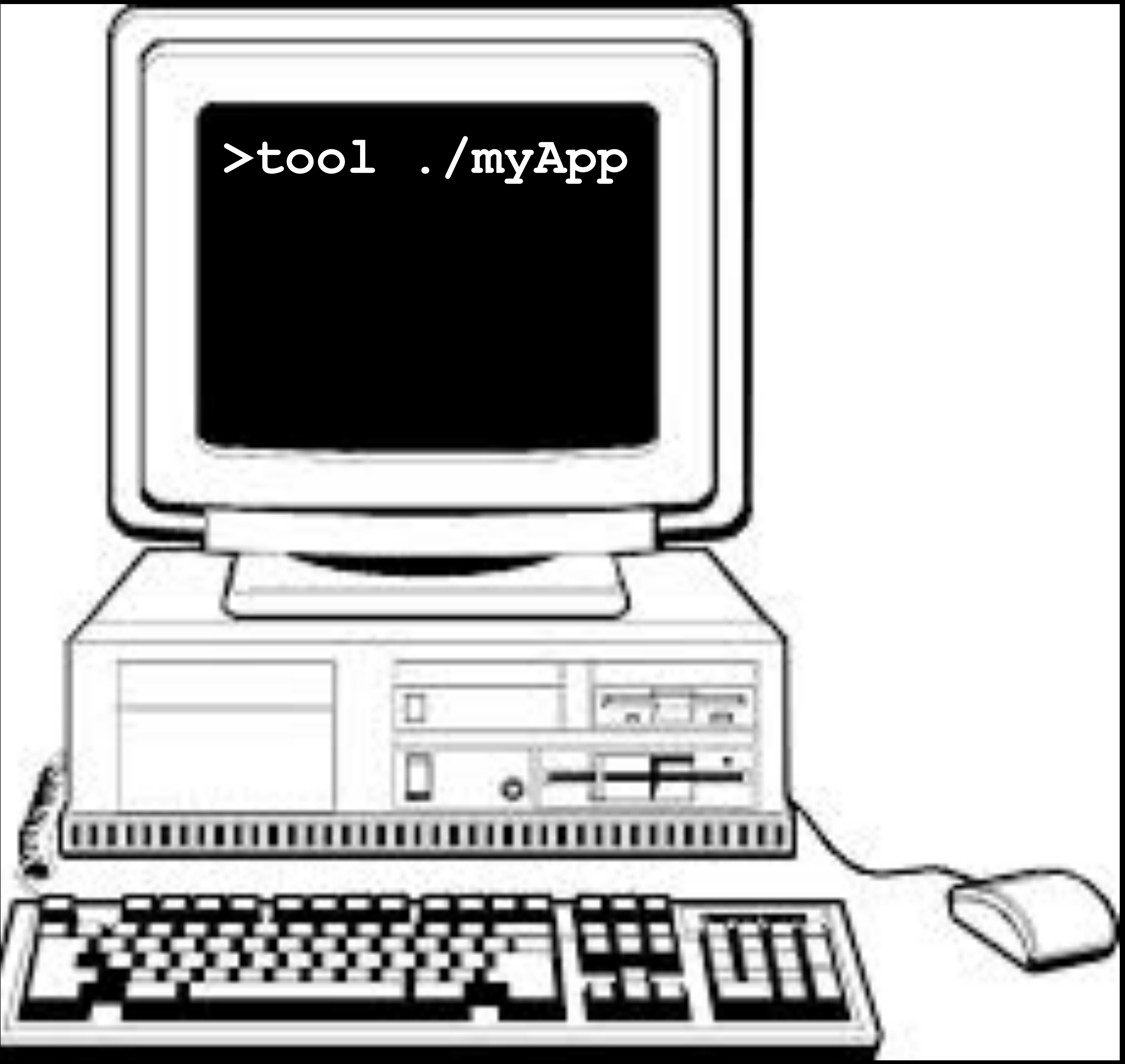
Data race detection tool



User

Client Uses the Tool

Data race detection tool



User

Client Uses the Tool

Data race detection tool

User

Where is My Concurrency Bug?

Data race detection tool

- 100s of files, 1000s of LOC, numerous functions
- Existing tools lack this capability
- We demonstrate how it is possible with acceptable overhead

User

Where is My Concurrency Bug?

Data race detection tool

- 100s of files, 1000s of LOC, numerous functions
- Existing tools lack this capability
- We demonstrate how it is possible with acceptable overhead

User

!

Need Better Diagnostic Capabilities for Tools

Data race detection tool

Thread 1

Thread 2

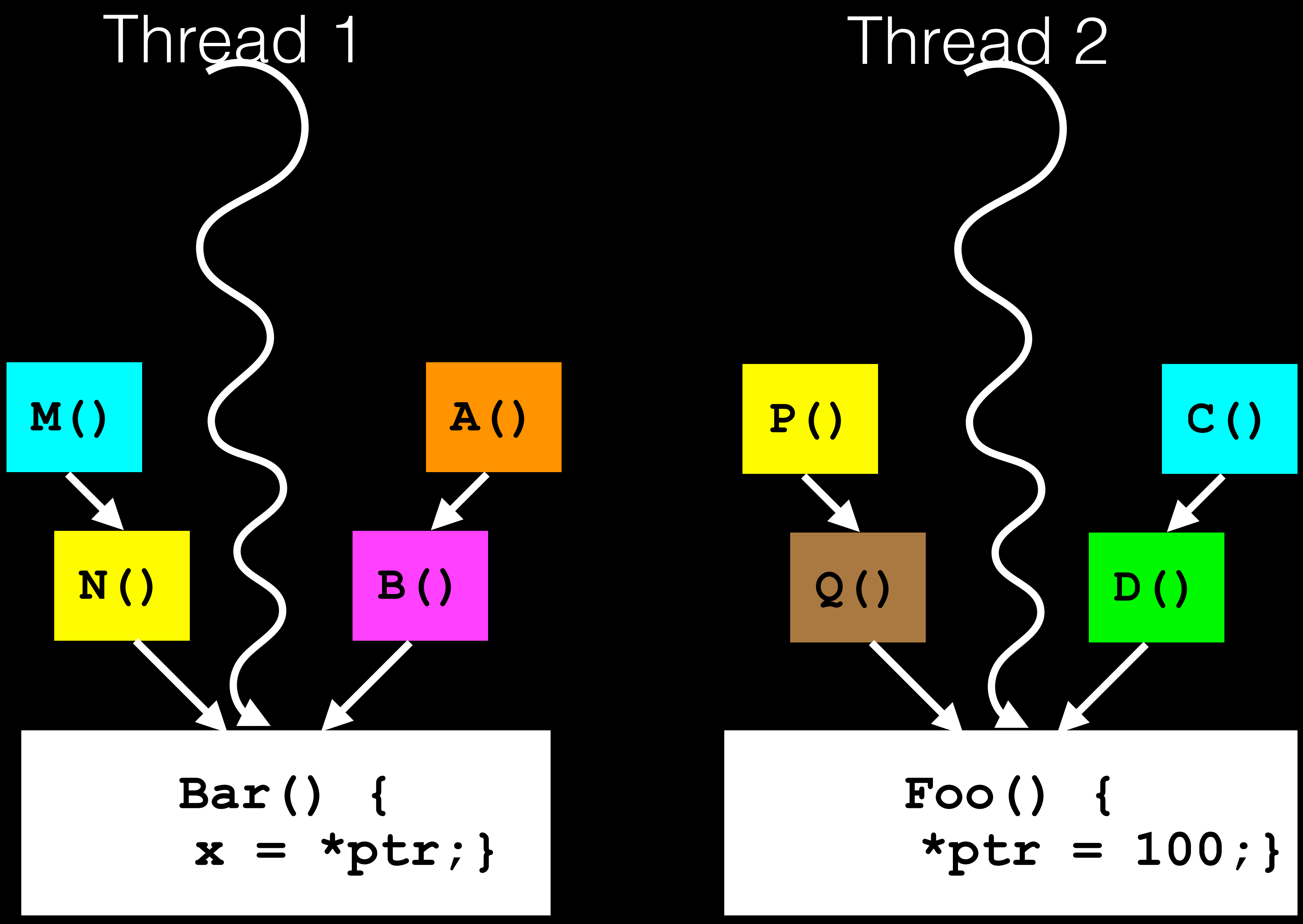
<u>User</u>


```
Bar() {
 x = *ptr;}
```

```
Foo() {
 *ptr = 100;}
```

Need Better Diagnostic Capabilities for Tools

Data race detection tool

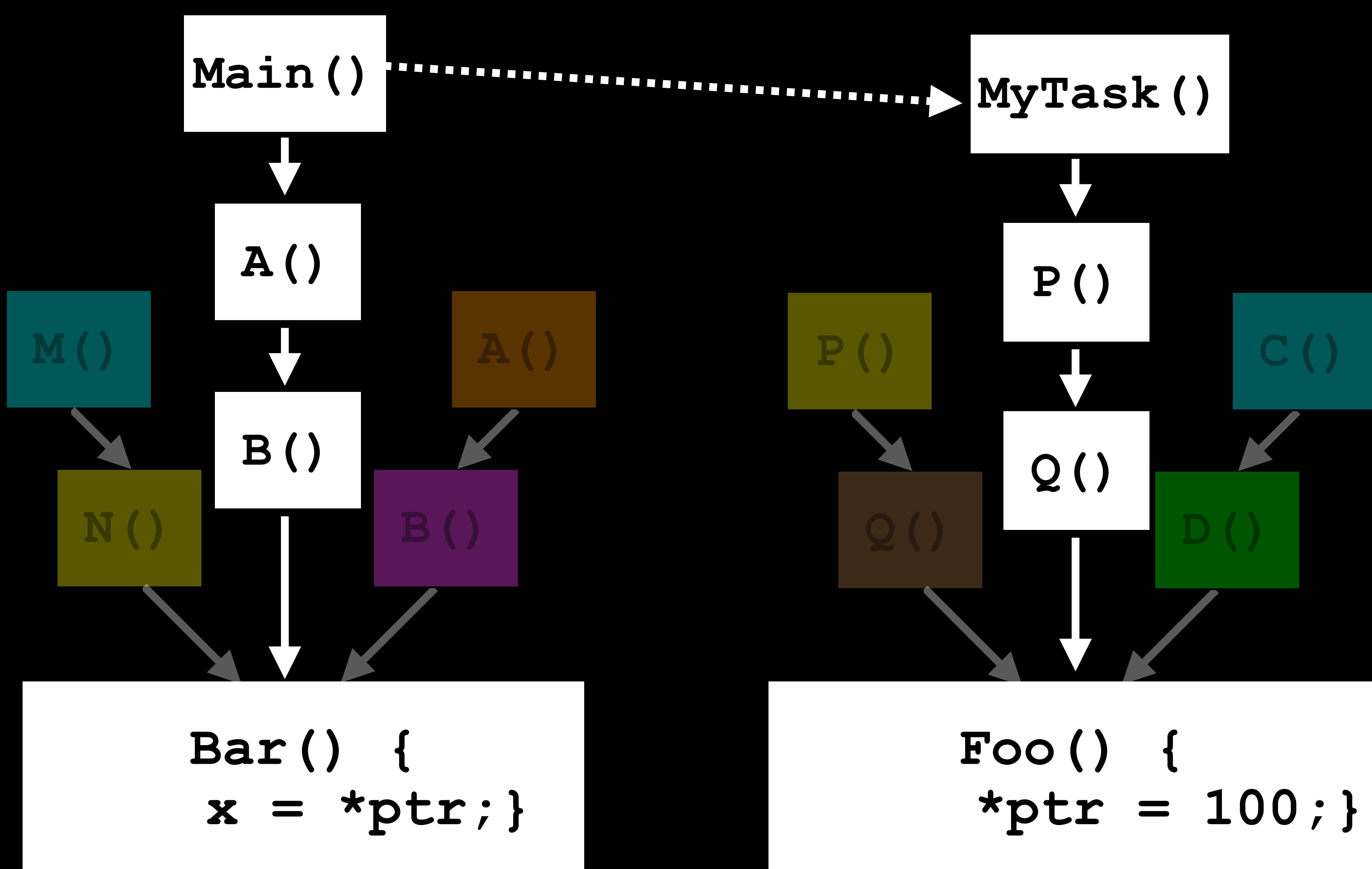


User

Need Better Diagnostic Capabilities for Tools

Data race detection tool

Thread 1 | Thread 2 | User



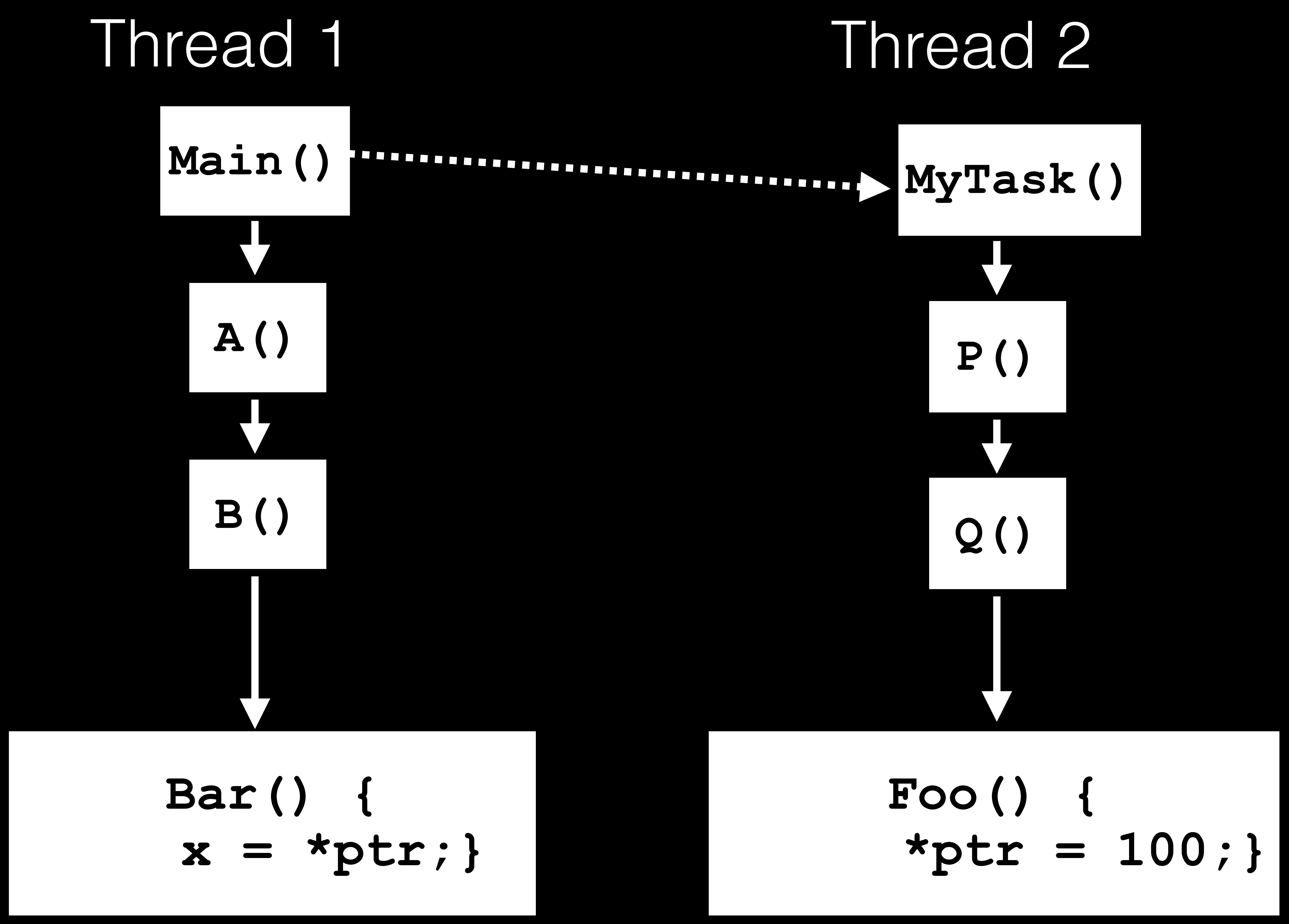
Calling context enhances tool's capability/usability

How Data Race Detection Works

- Tool executes the program
- Tool <u>monitors every memory access</u> by each thread
- Tool maintains abbreviated history of previous accesses (thread id) for each memory address
- Tool inspects the access history and determines if any conflicting accesses happen in parallel

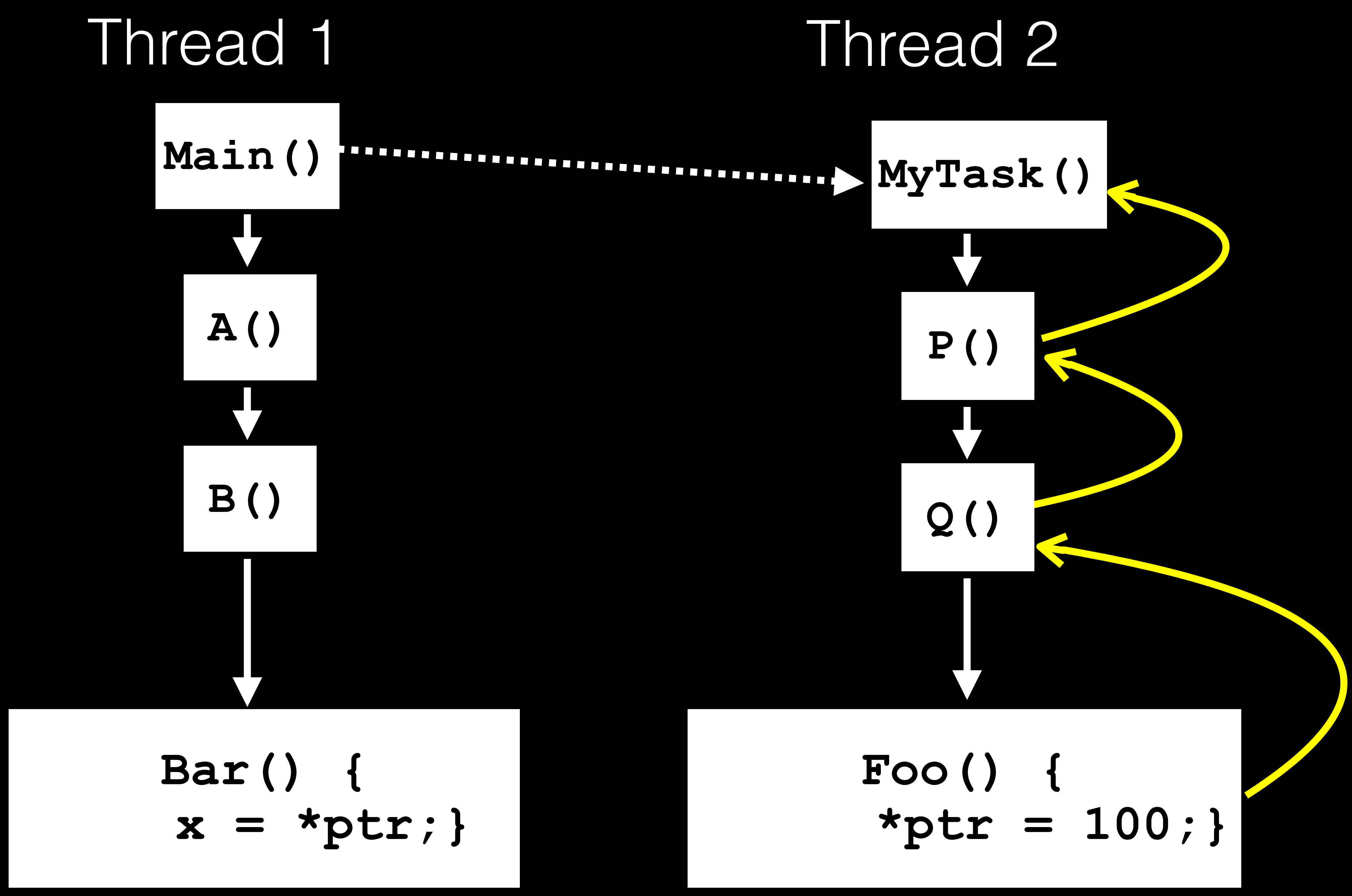
Challenges of Providing Calling Context

- Unwinding can collect current calling context
- Calling context of previous accesses is lost



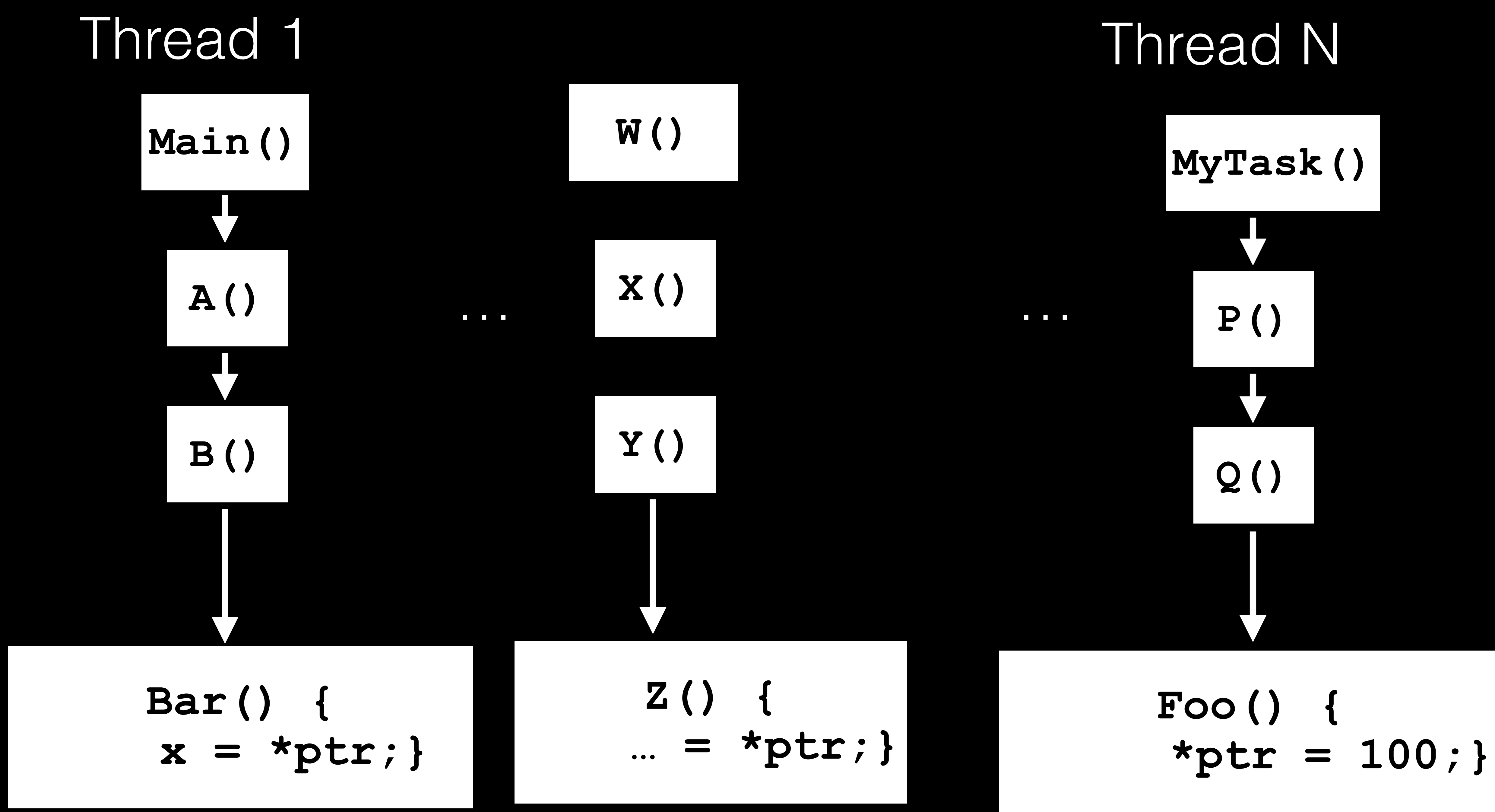
Challenges of Providing Calling Context

- Unwinding can collect current calling context
- Calling context of previous accesses is lost



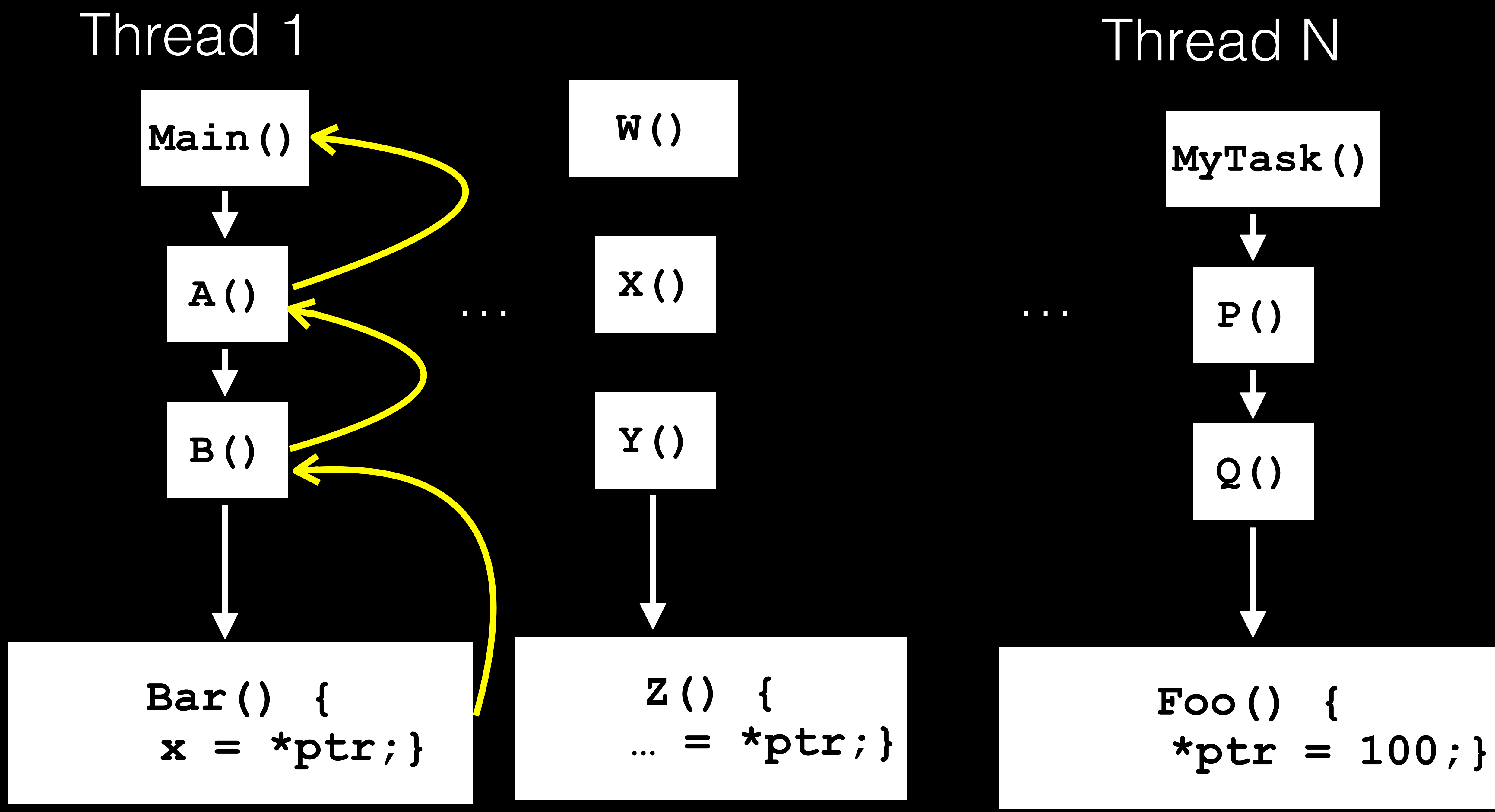
Naive Solution: Maintain a History of Contexts

Unwind and store call path on each access



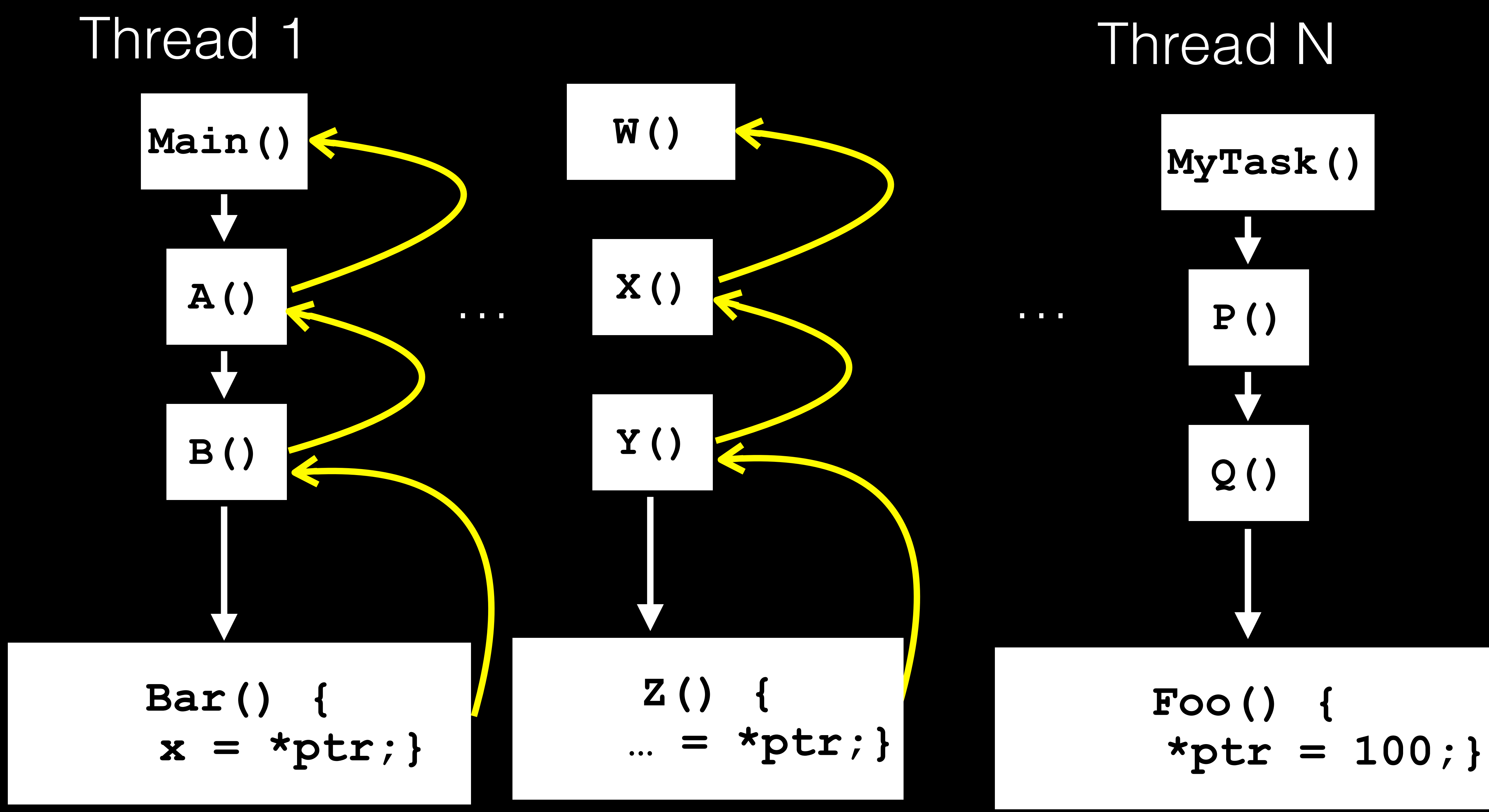
Naive Solution: Maintain a History of Contexts

Unwind and store call path on each access



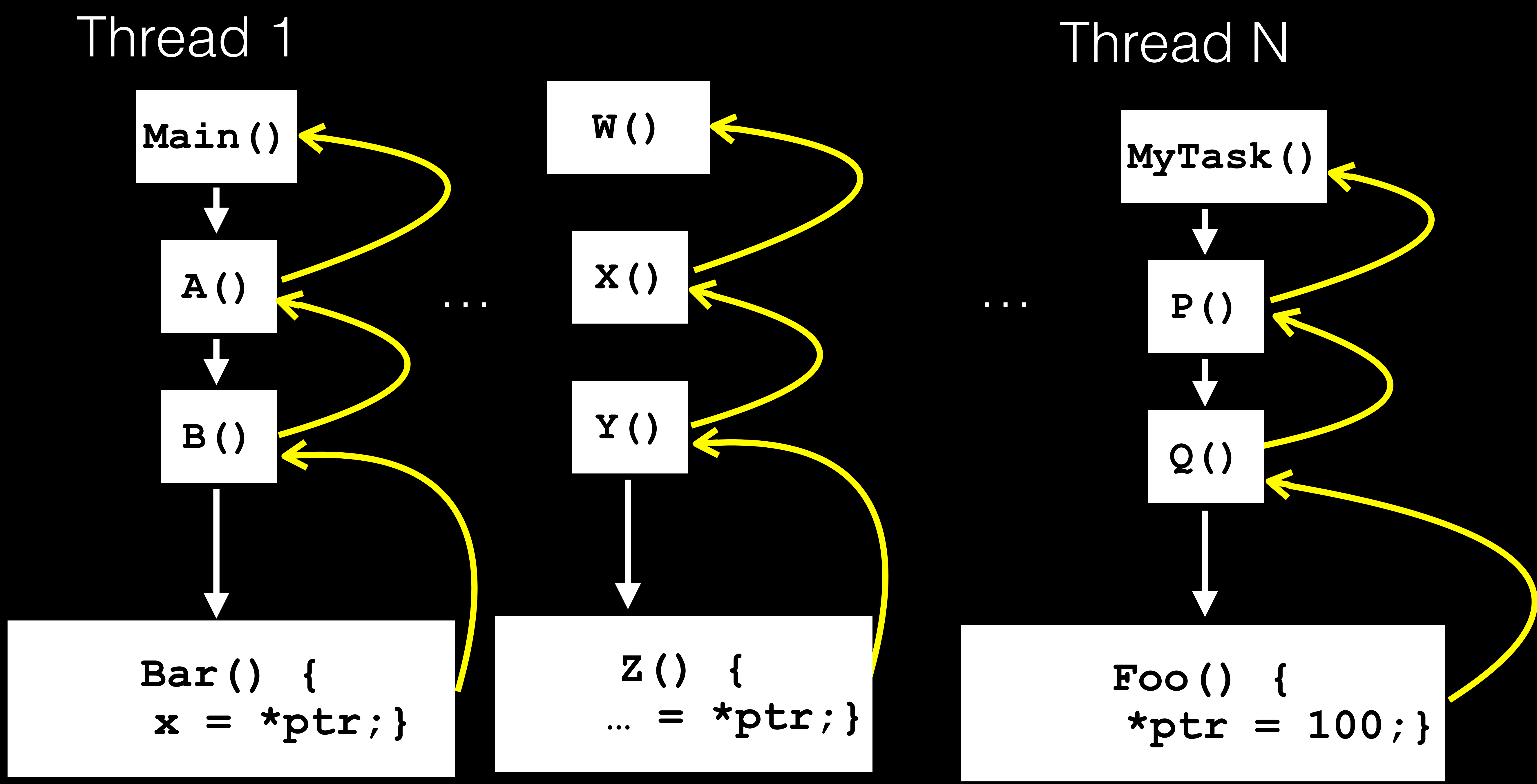
Naive Solution: Maintain a History of Contexts

Unwind and store call path on each access



Naive Solution: Maintain a History of Contexts

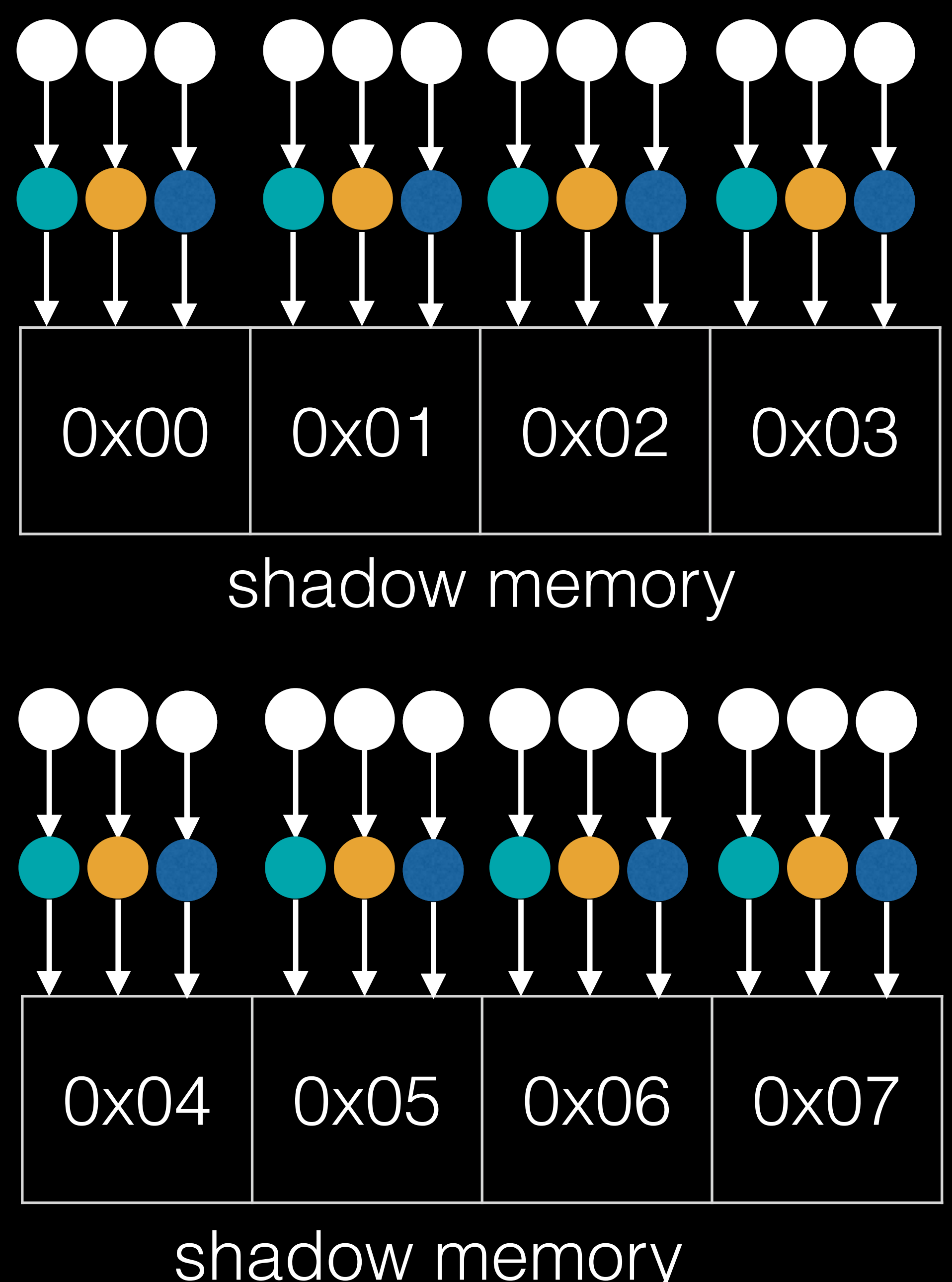
Unwind and store call path on each access



Overheads of Naive Solution

Naive Solution

Unwind and record the calling context on each memory access



Problems

1. **Space overhead** for maintaining many calling contexts
2. **Time overhead** for call stack unwinding at each memory access

Frameworks for Fine-Grained Program Monitoring

Valgrind

"Getting calling contexts using VG_(get_StackTrace) is done via stack unwinding. Unwinding for each memory access will be very slow."

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

Dyninst

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

Pin

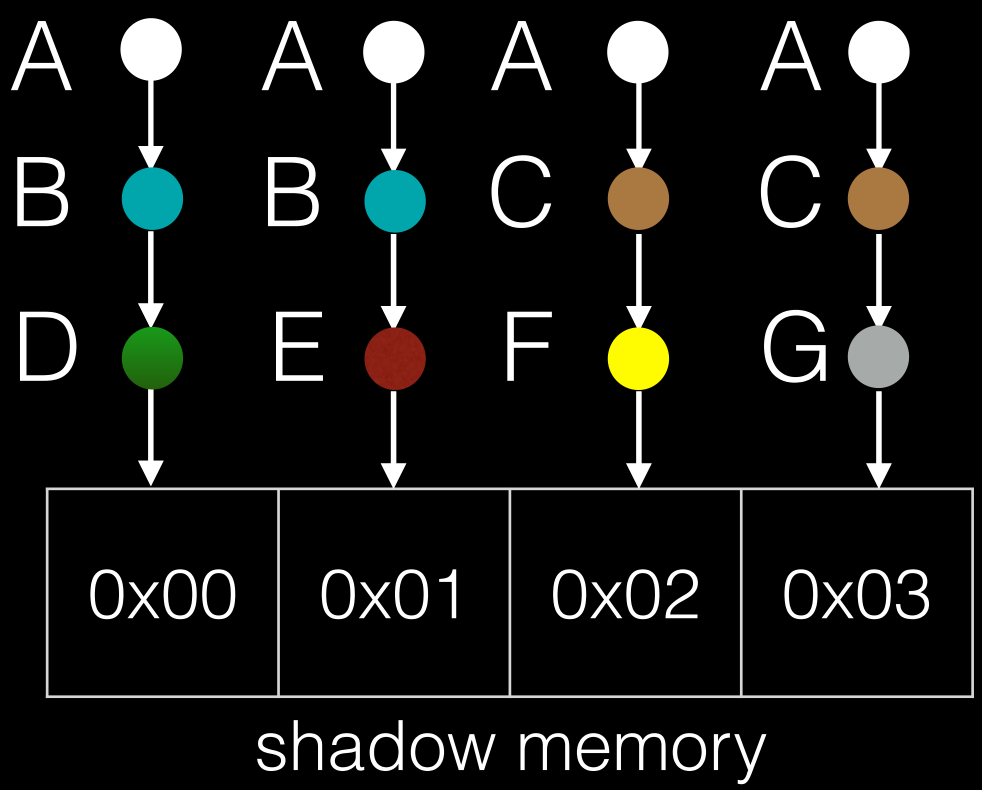
- No support for collecting calling contexts
- We built it ourselves—**CCTLib**
- Demonstrate how it is possible to gather calling context ubiquitously with **CCTLib**

Many Tools Require Fine-grained Program Monitoring

- Performance analysis tools
  - Cache simulators
  - Reuse-distance analysis
  - False sharing detection
  - Memory / computation redundancy
- Software correctness
  - Taint analysis
  - Malware detection
  - Memory leak / array out of bounds
- Many other tools, e.g.,
  - Debugging, testing, resiliency, replay, etc.

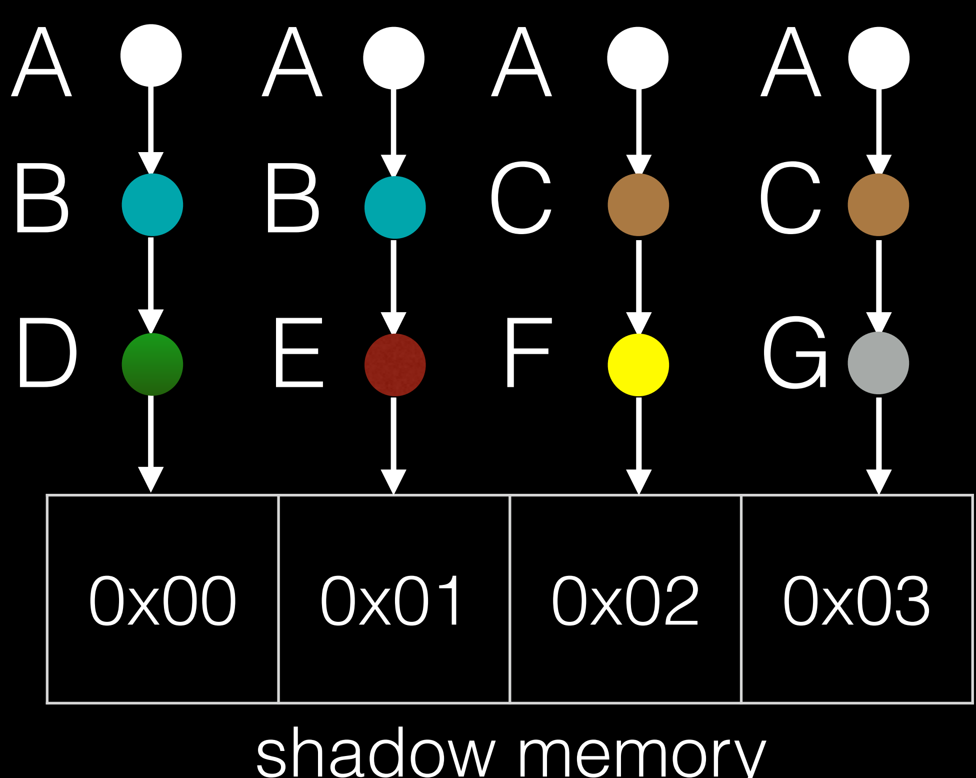
Store History of Contexts Compactly

Space bloat problem



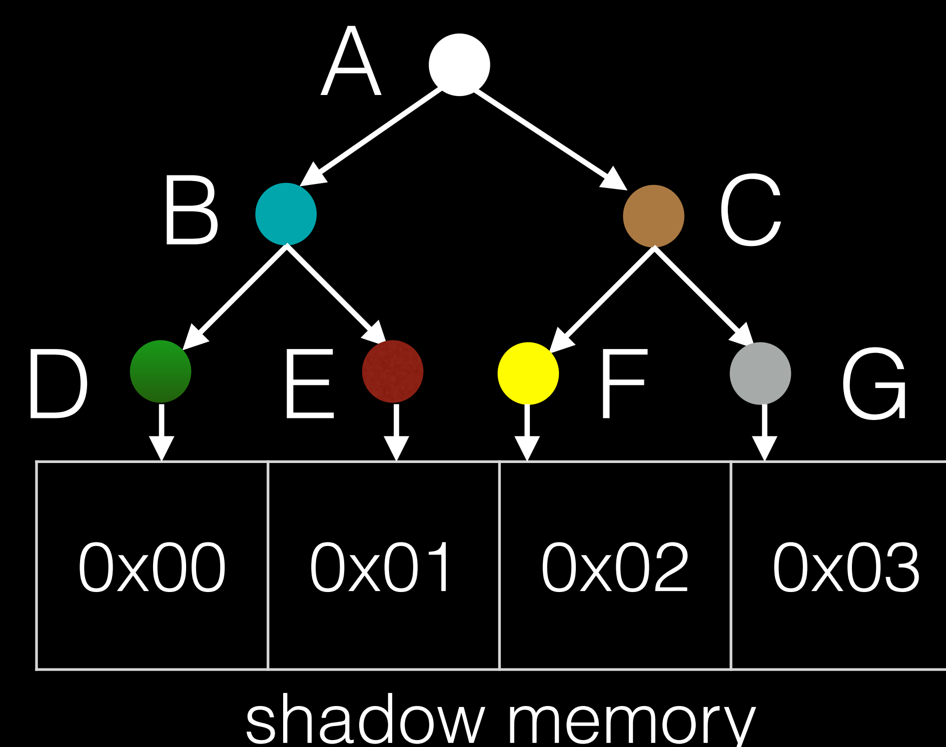
Store History of Contexts Compactly

Space bloat problem



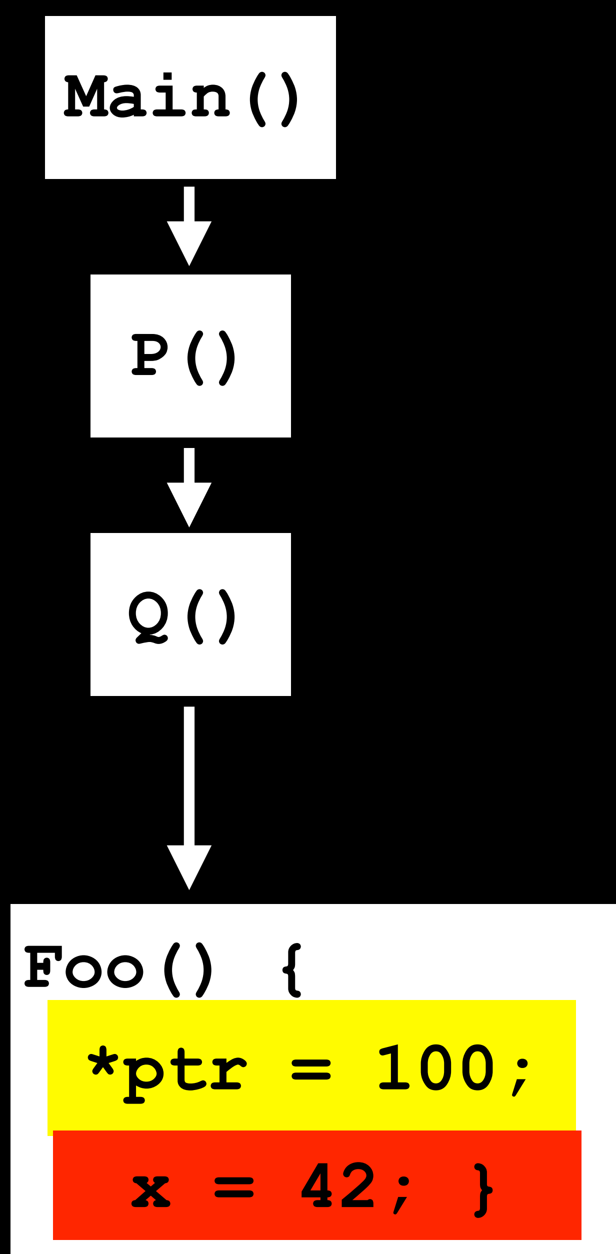
Solution

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



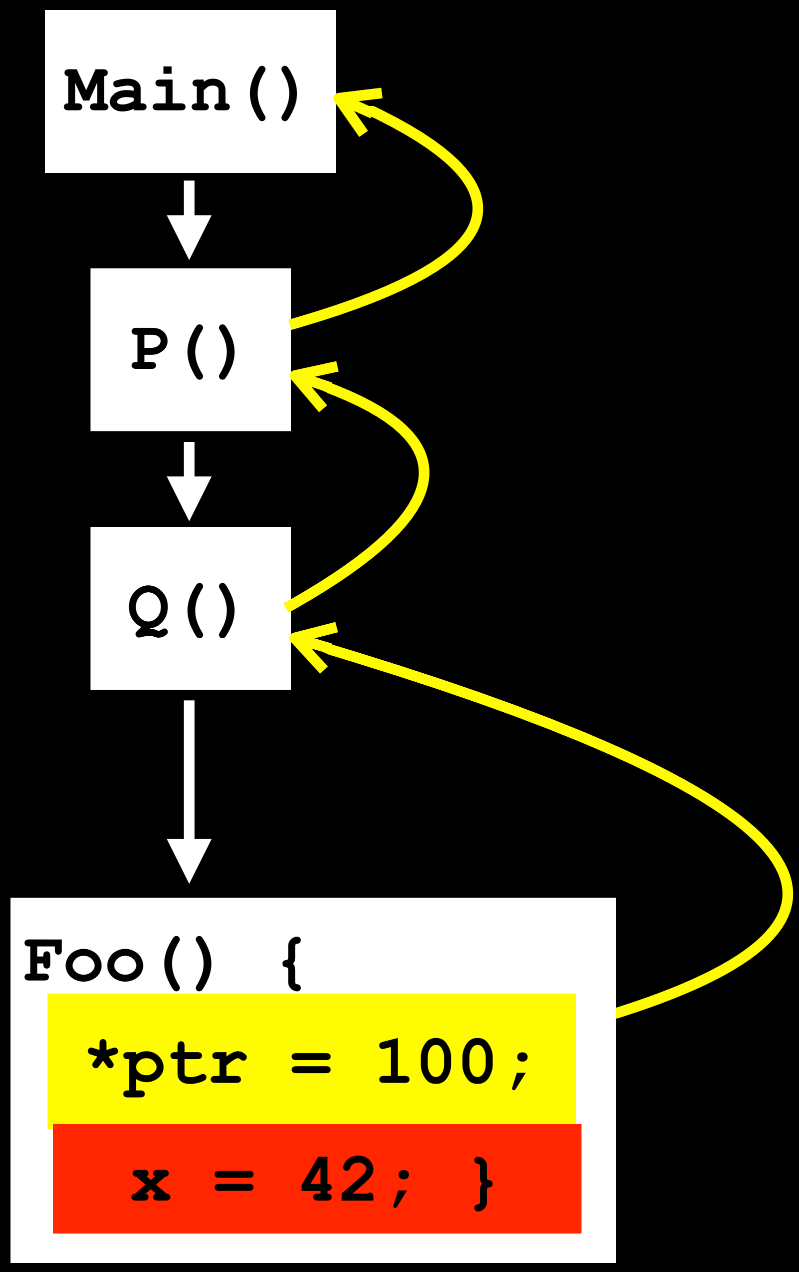
Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead



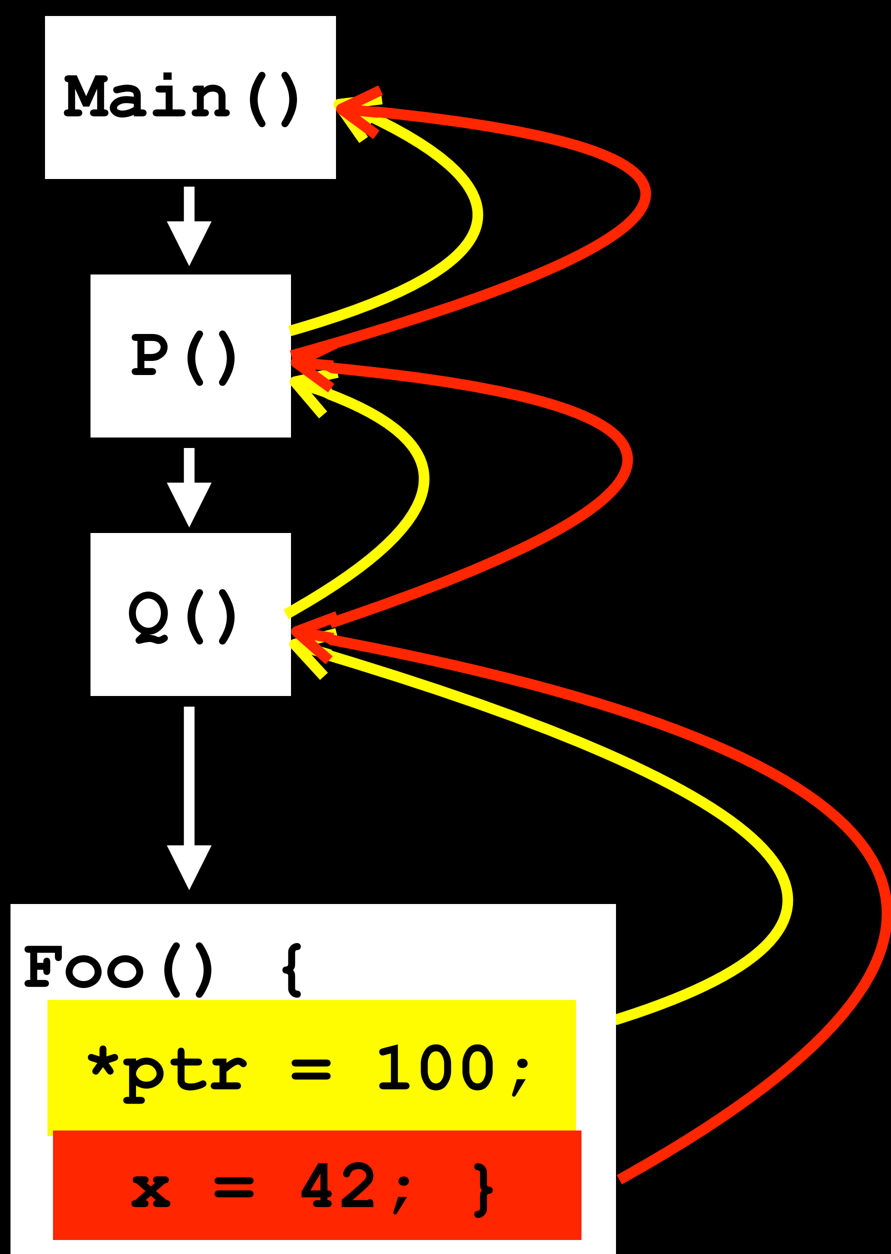
Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead



Shadow Stack to Avoid Unwinding Overhead

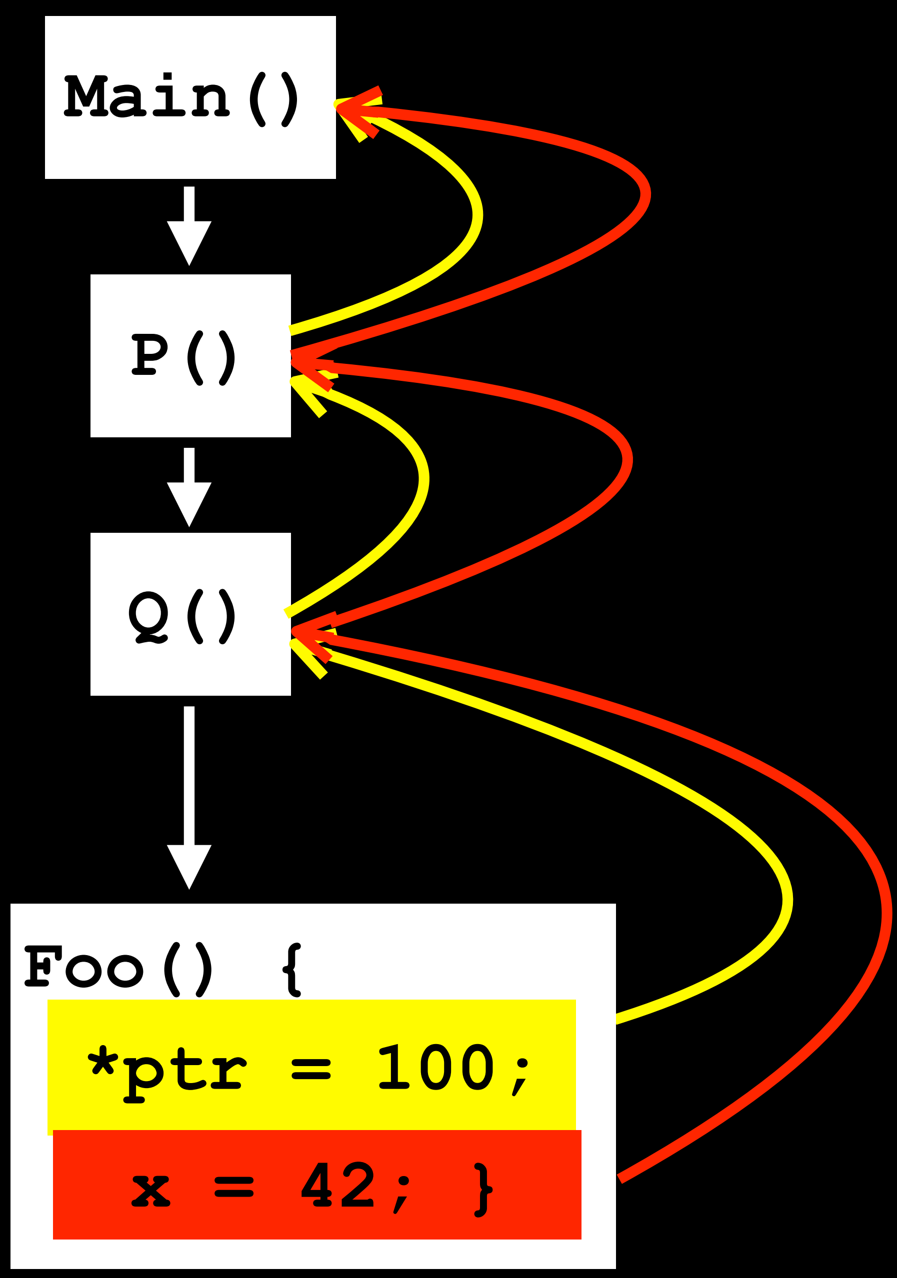
Problem:
Unwinding overhead



Shadow Stack to Avoid Unwinding Overhead

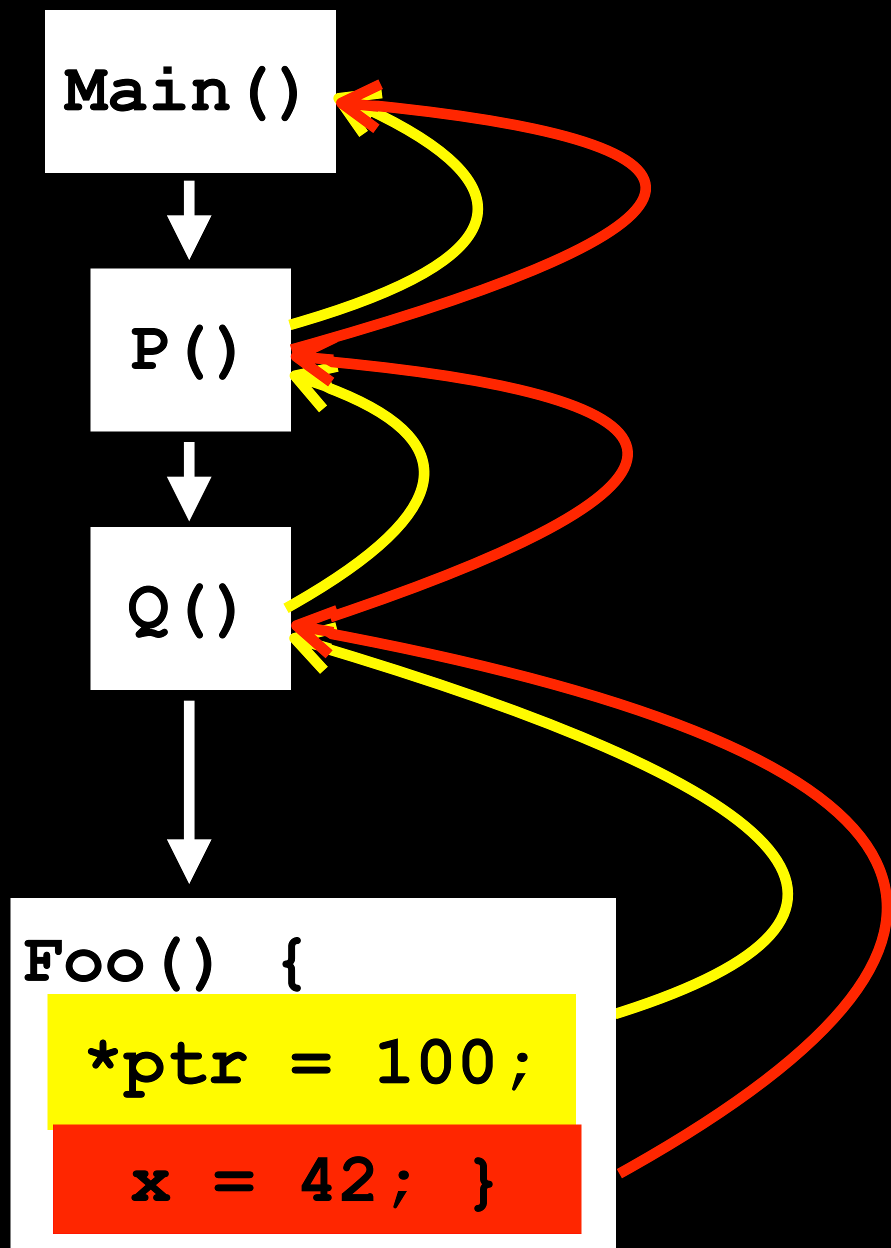
Problem:
Unwinding overhead

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

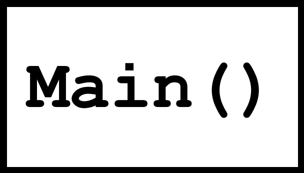


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead



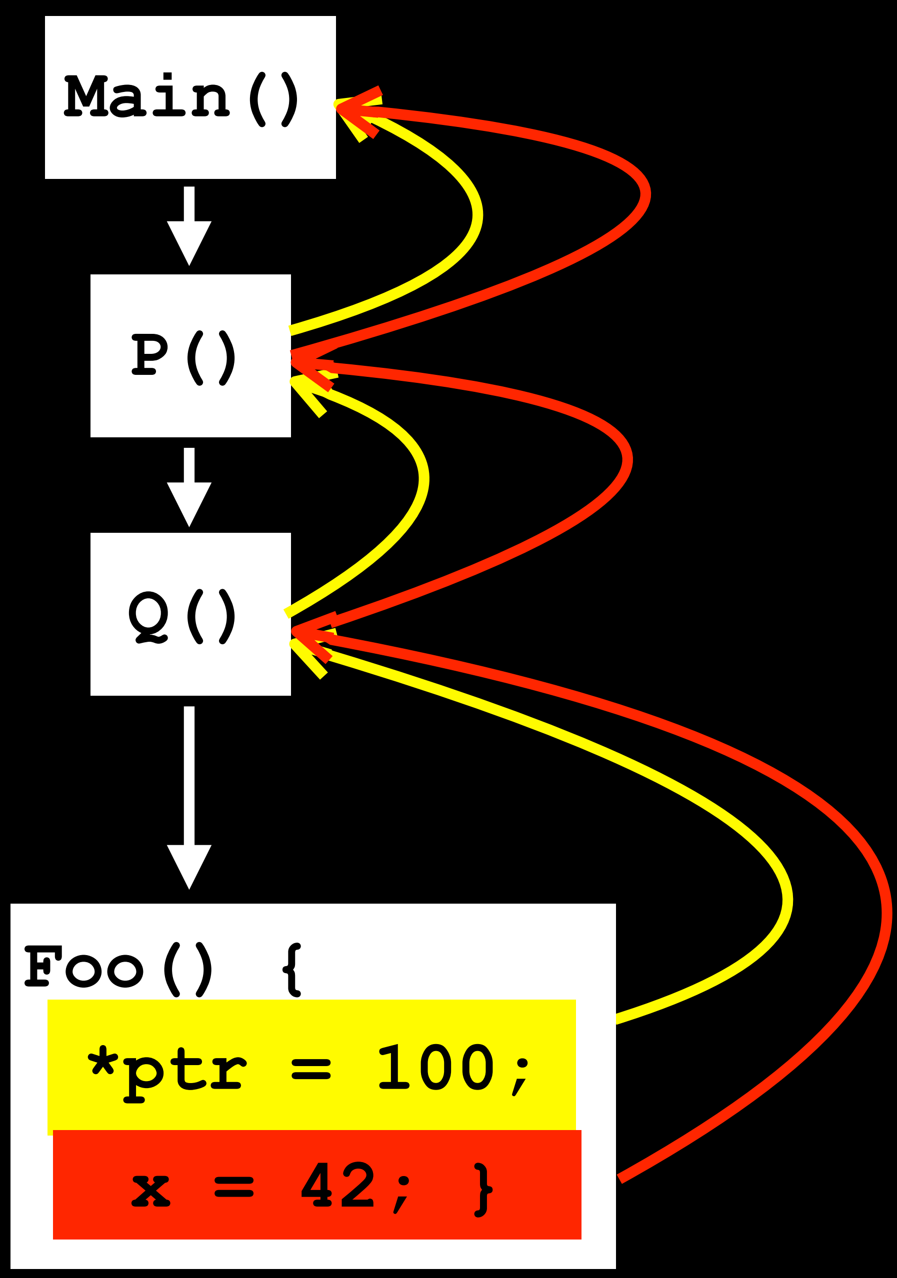
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.



Shadow Stack to Avoid Unwinding Overhead

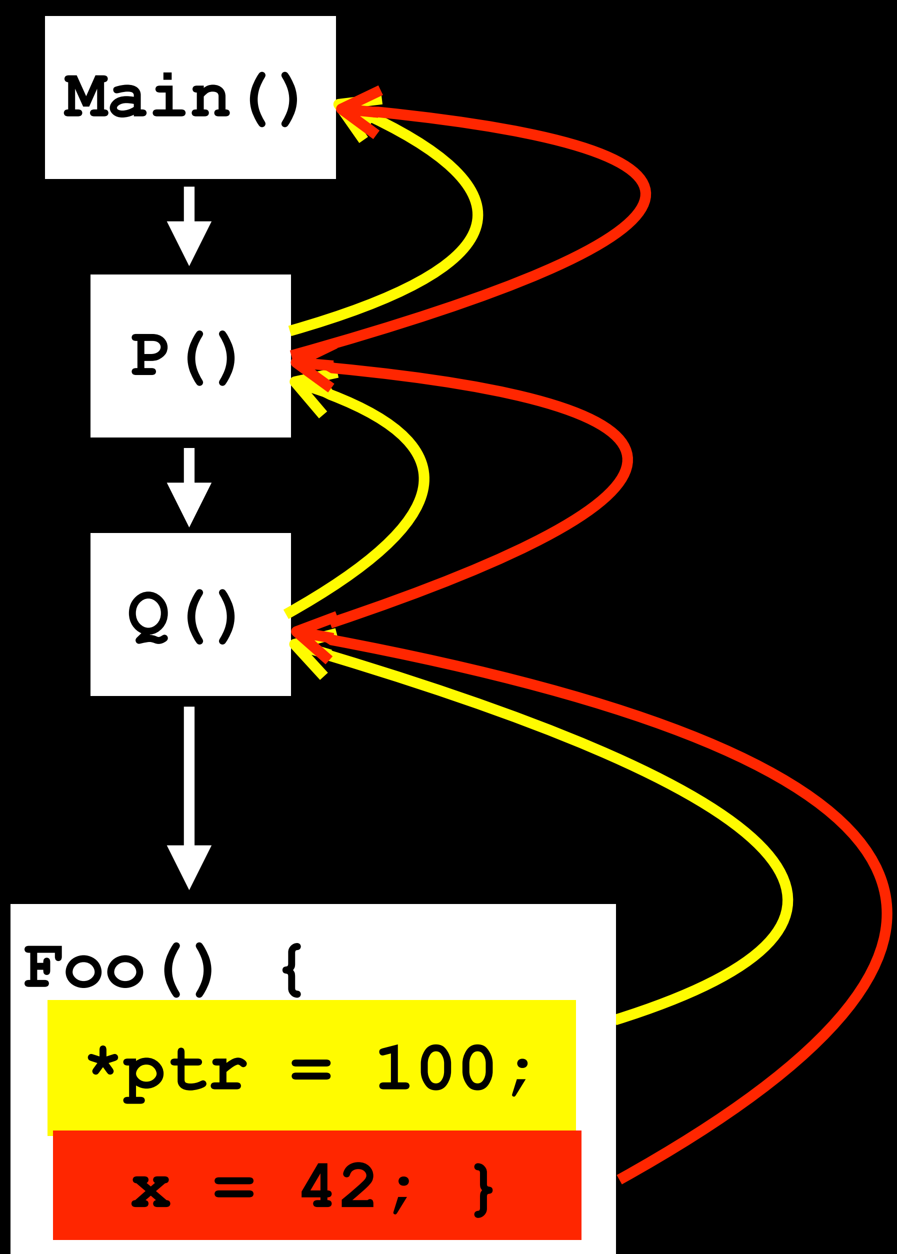
Problem:
Unwinding overhead

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

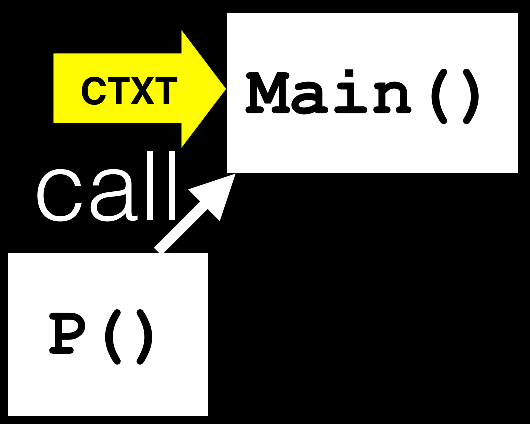


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

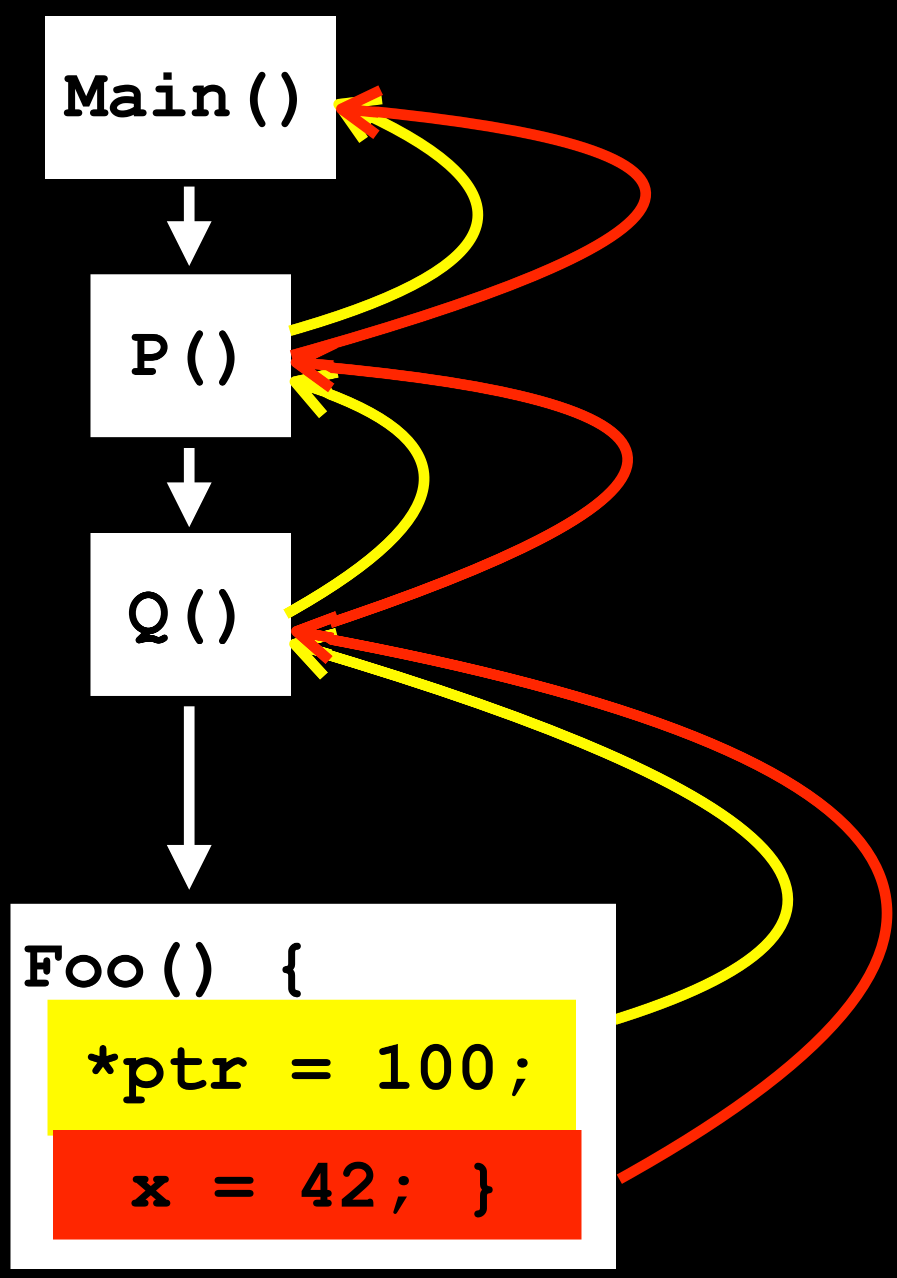


Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

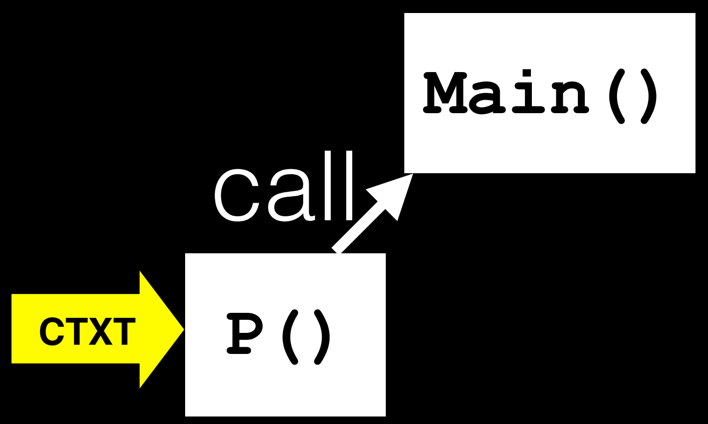


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

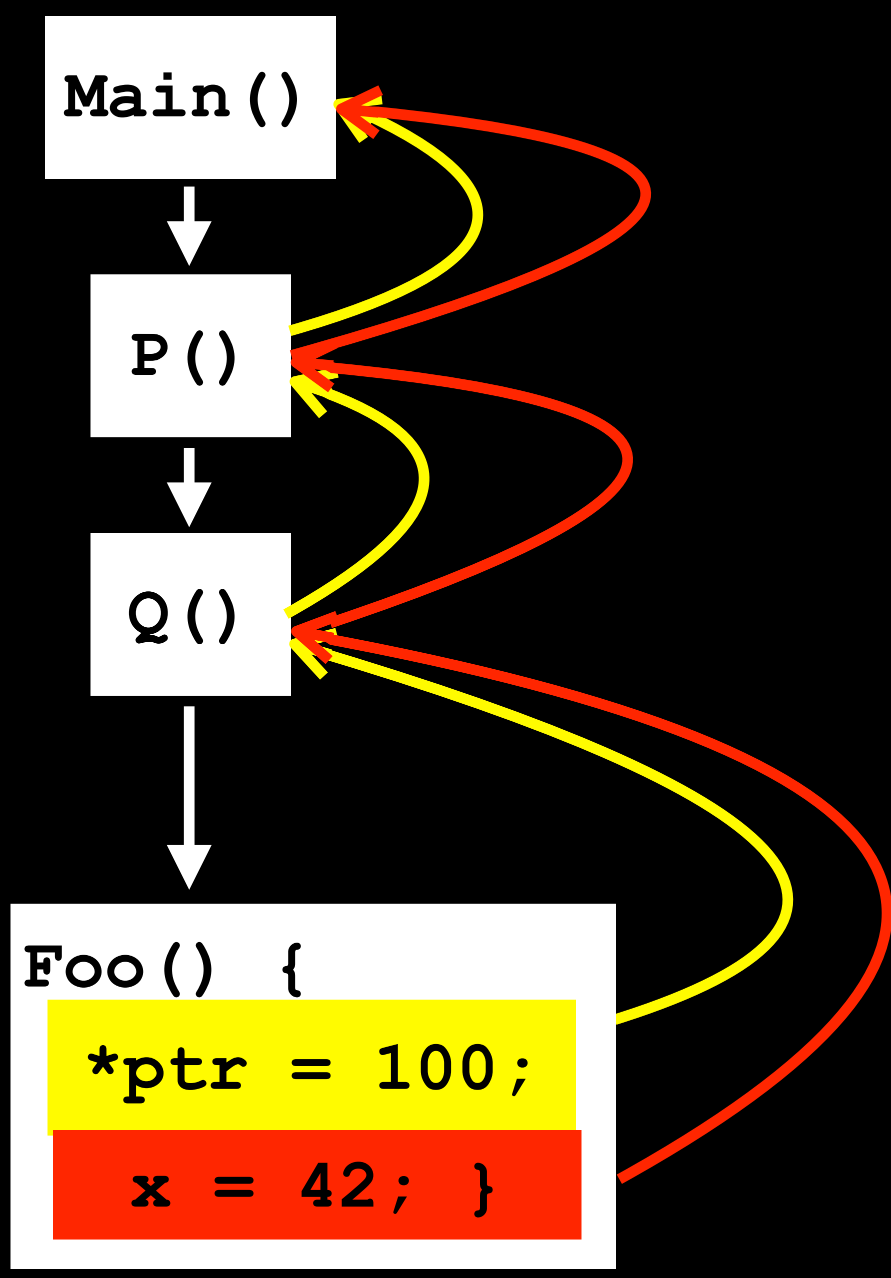


Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

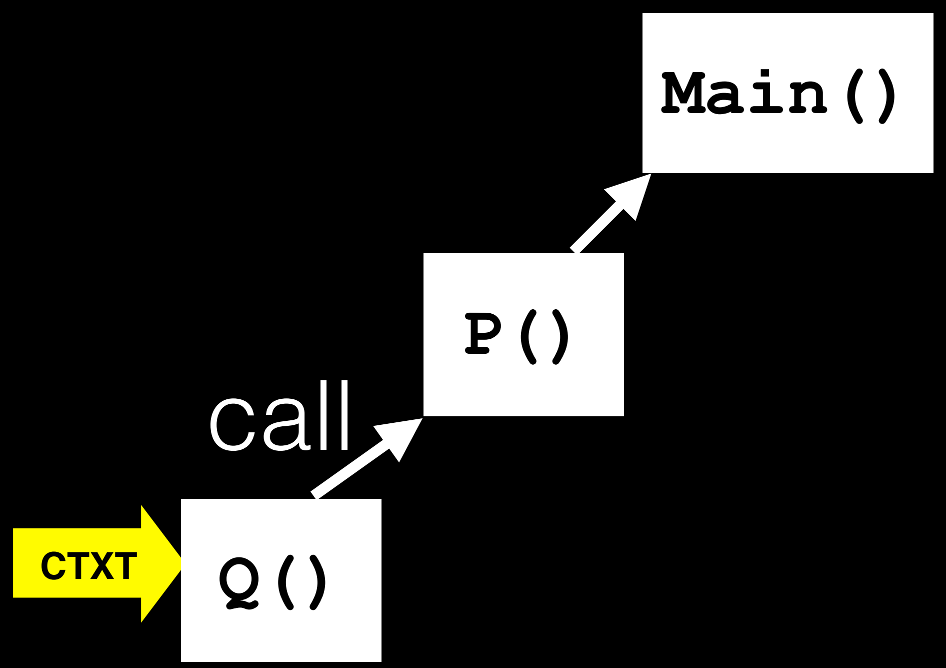


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead



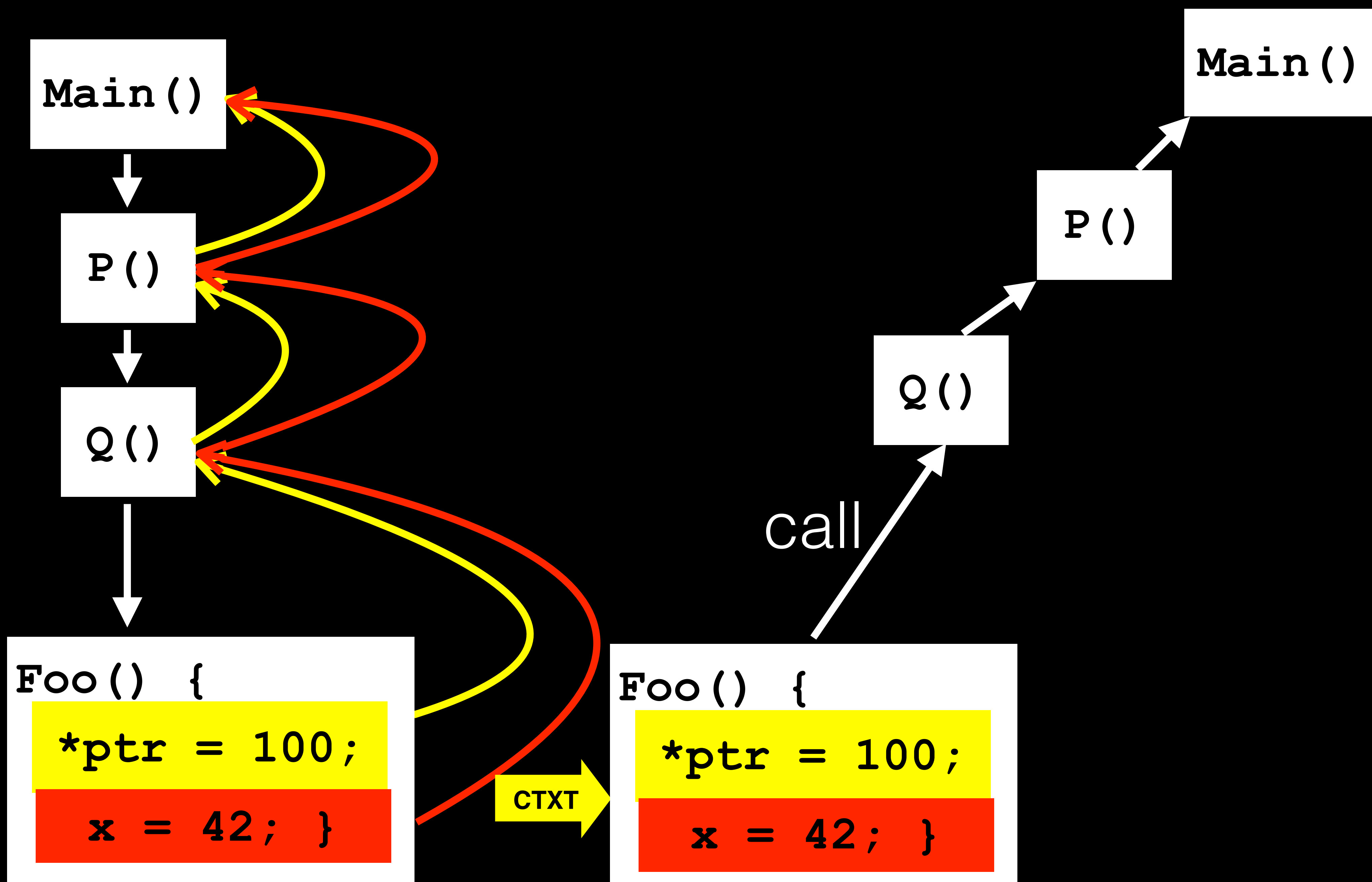
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.



Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

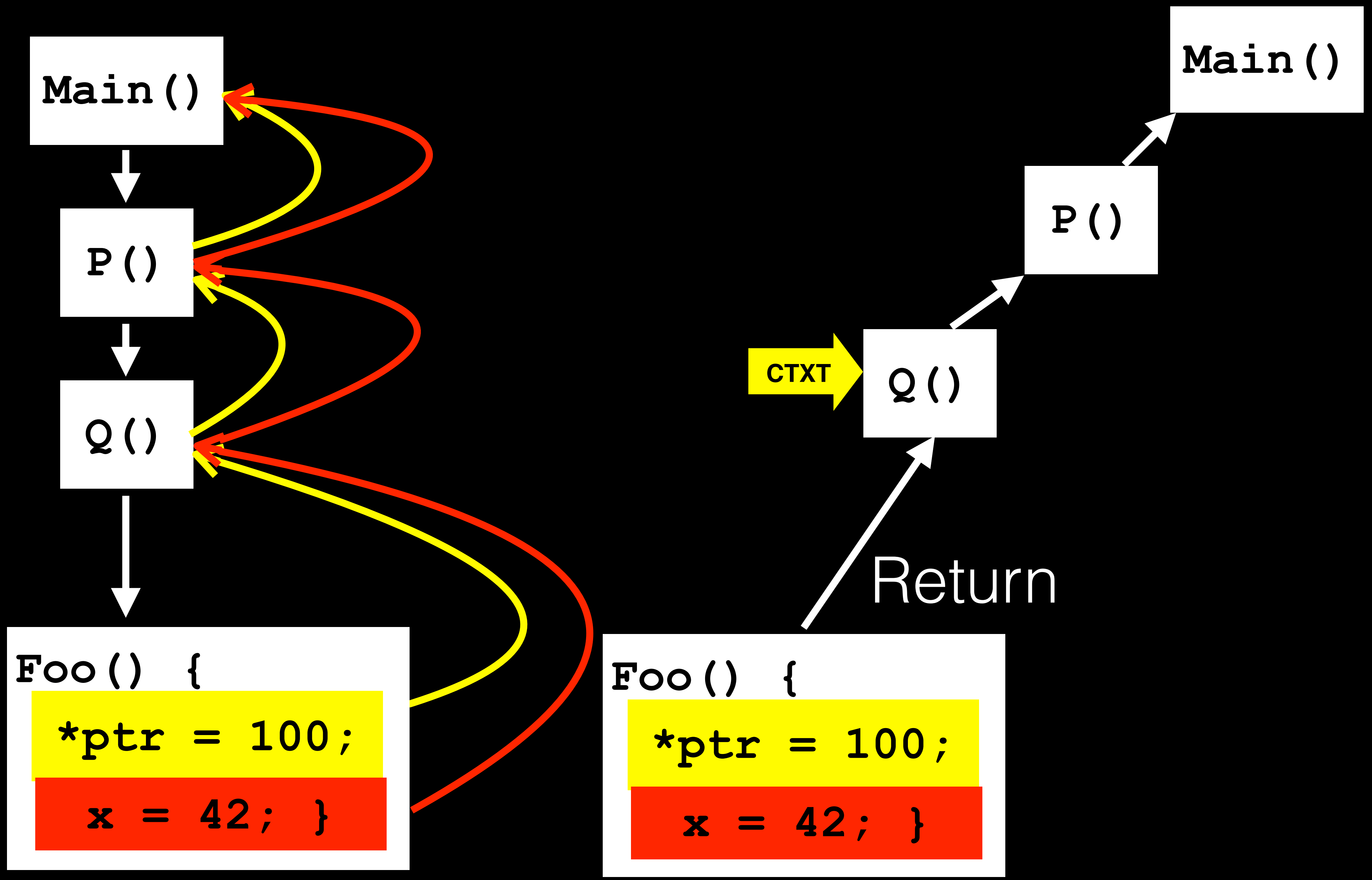
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.



Shadow Stack to Avoid Unwinding Overhead

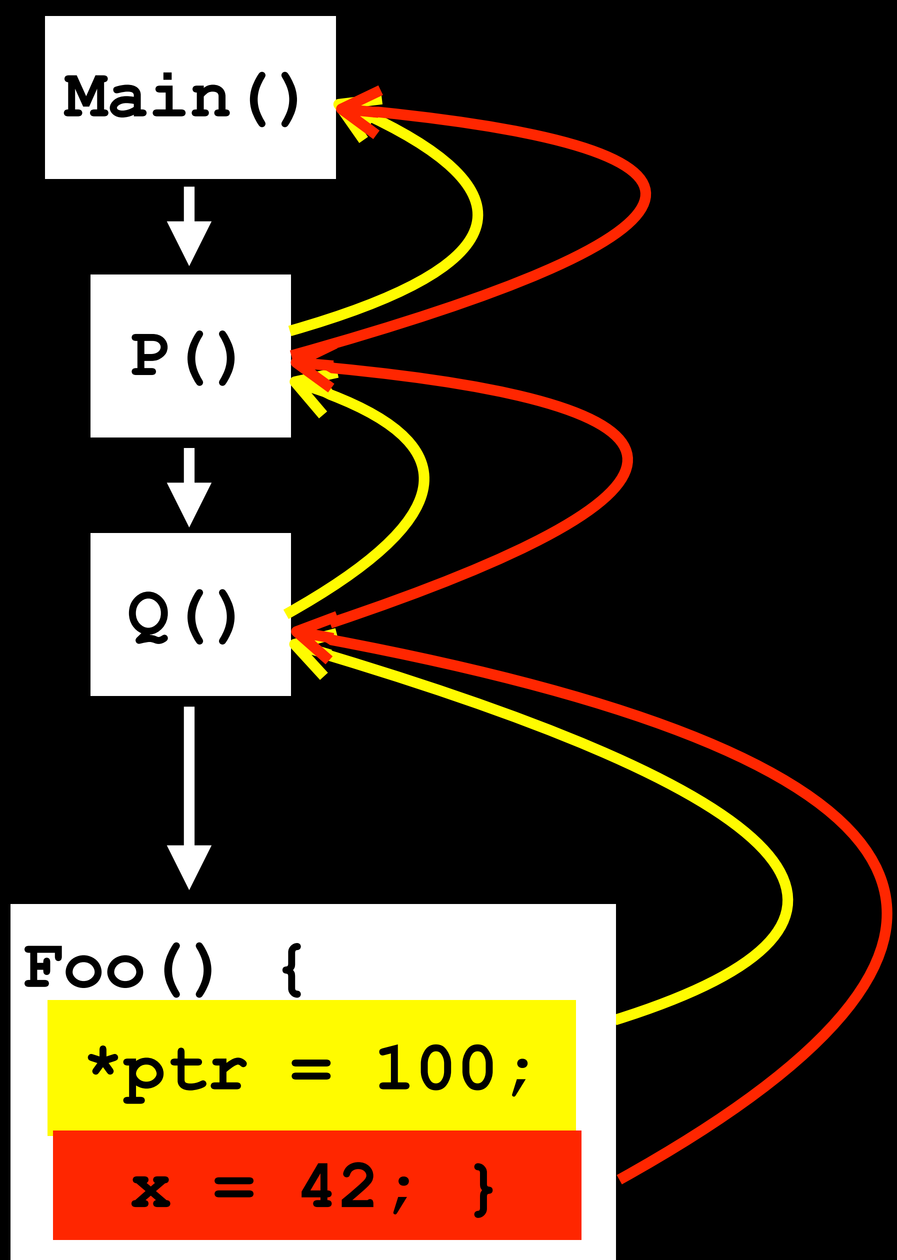
Problem:
Unwinding overhead

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

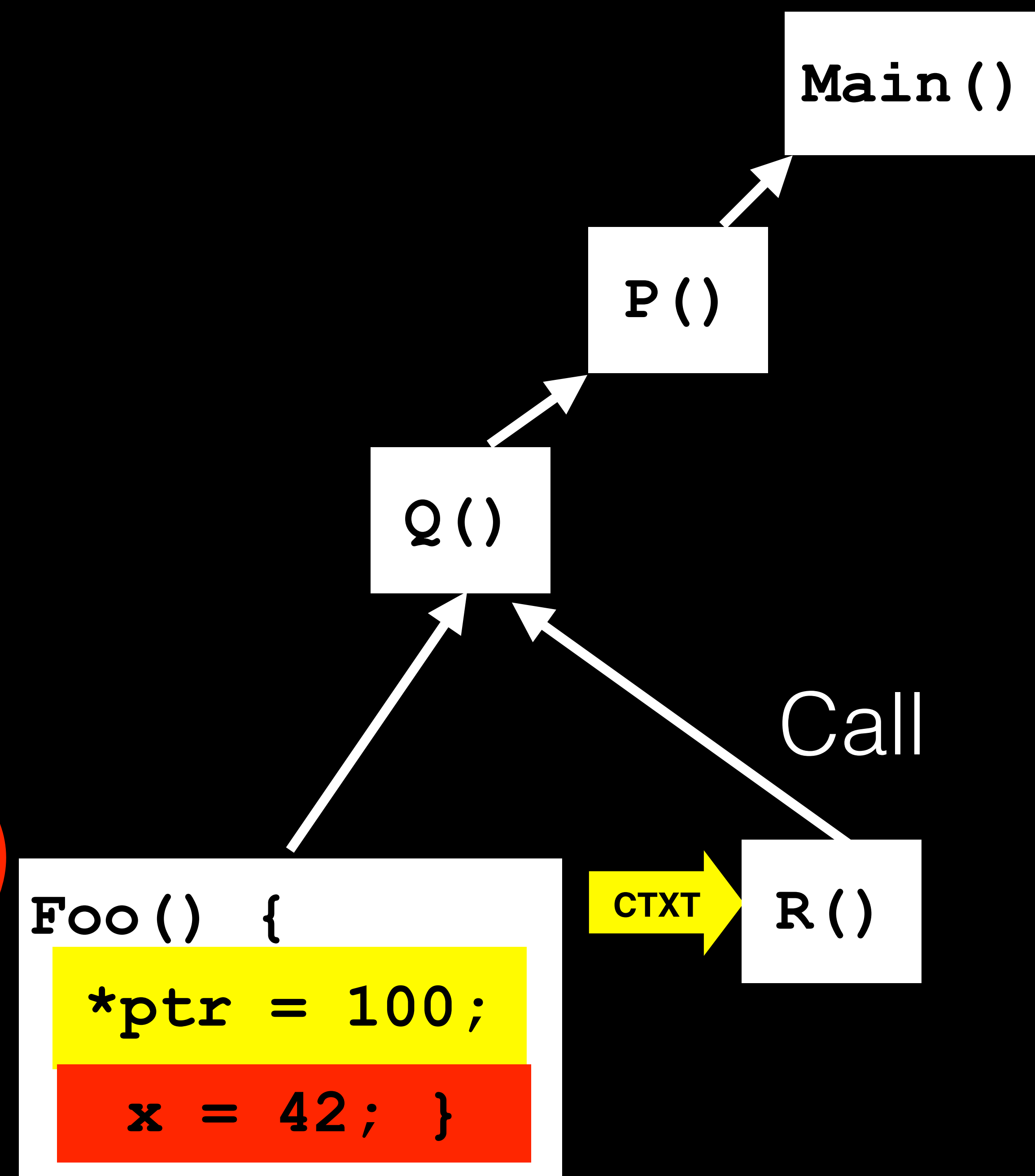


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead



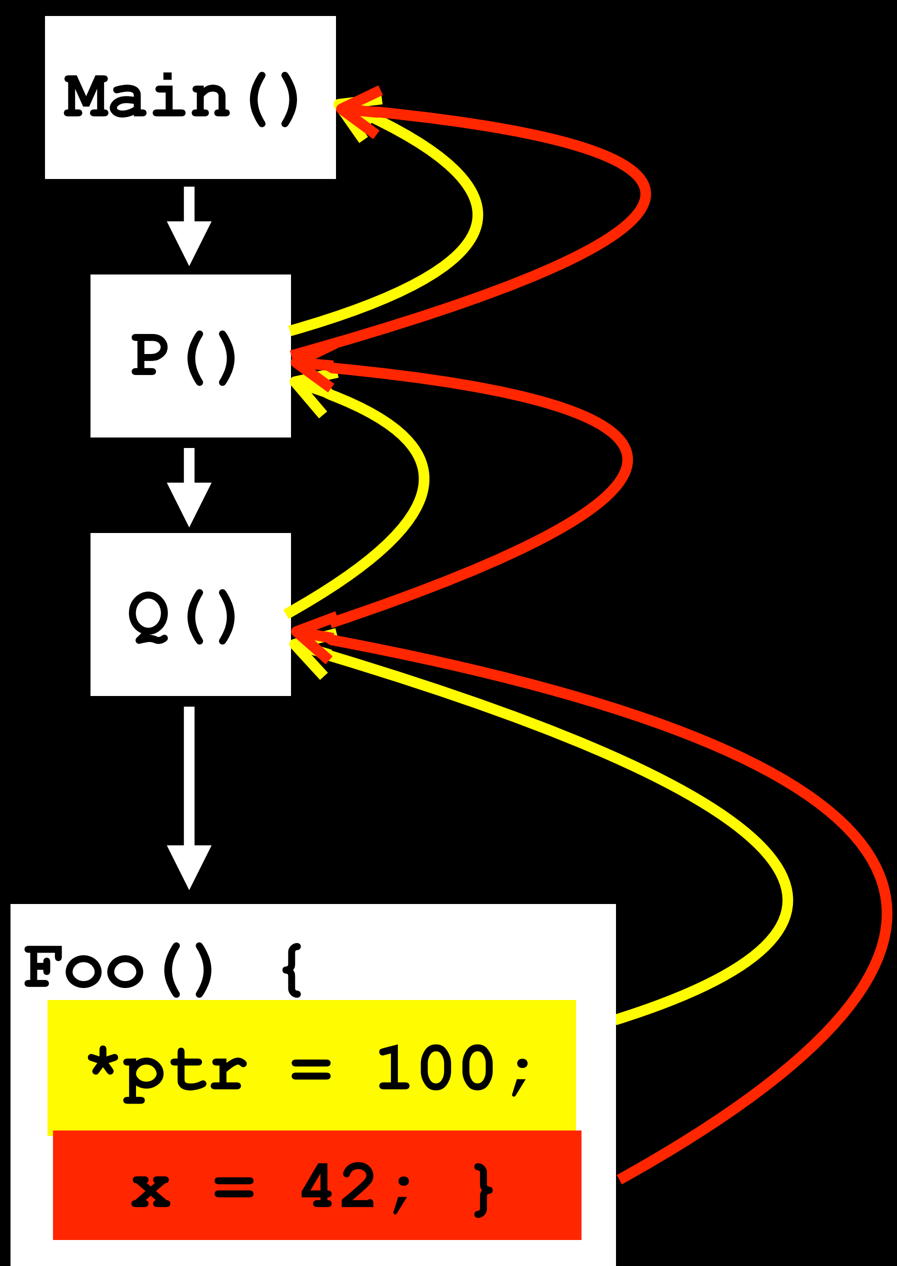
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

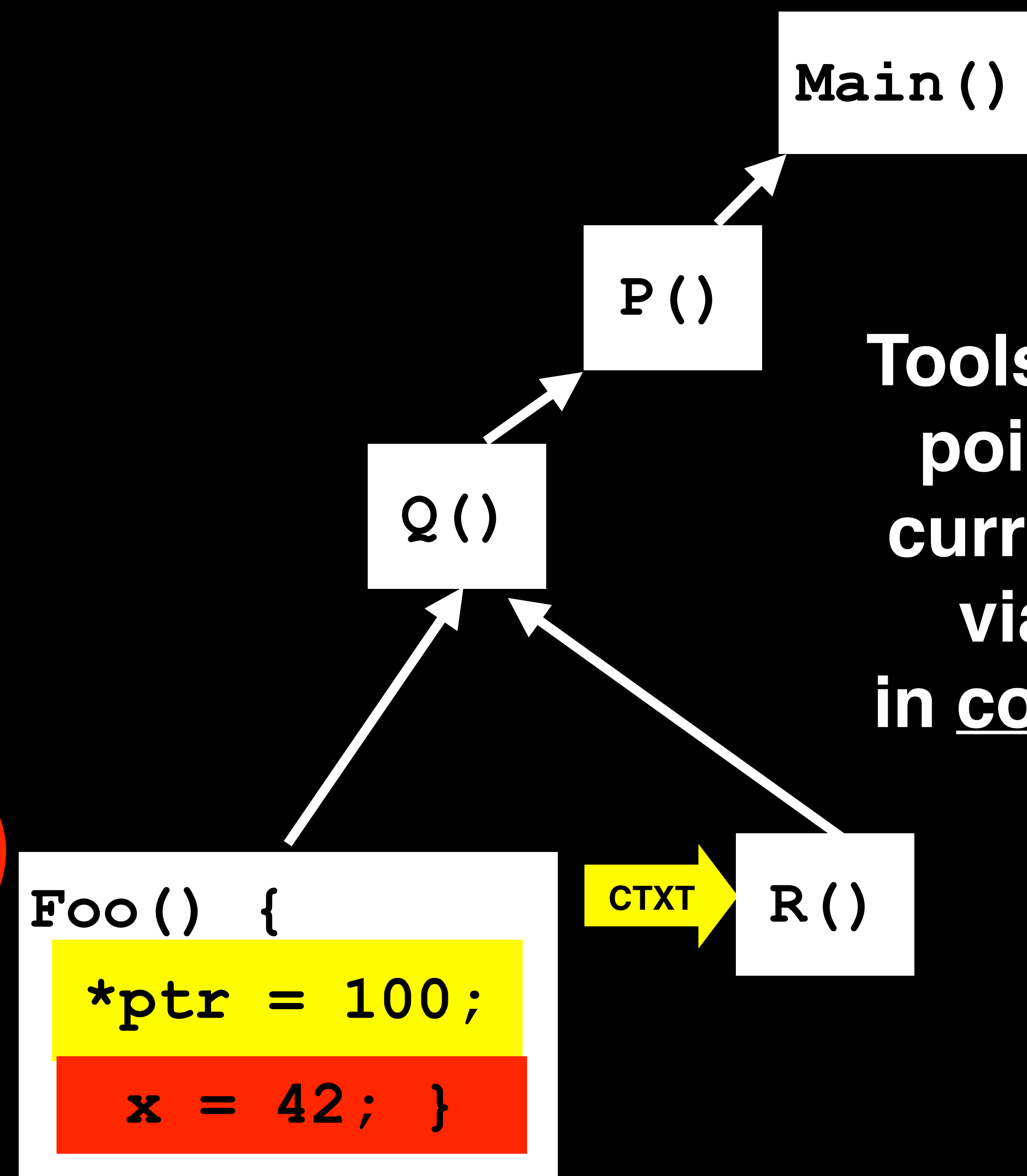


Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

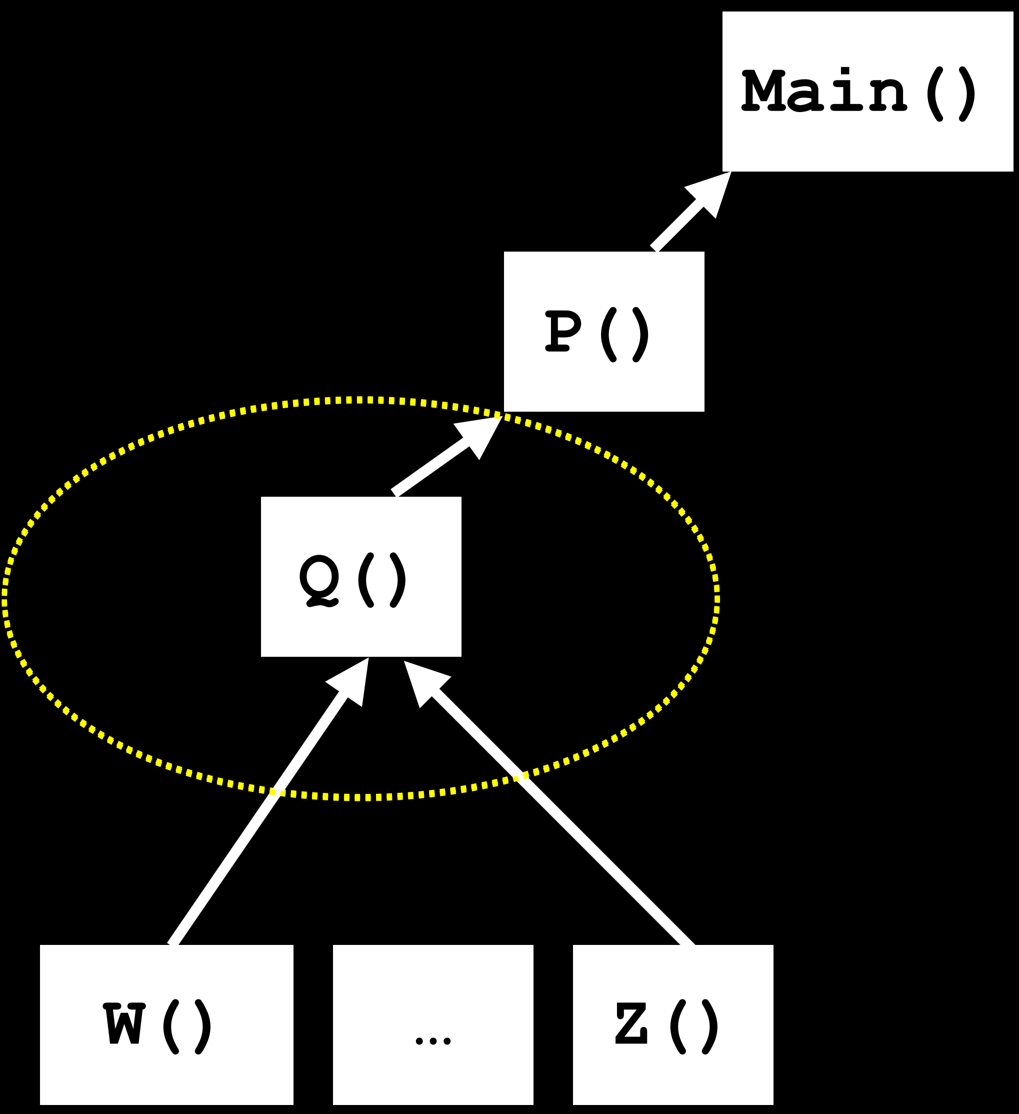
Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.



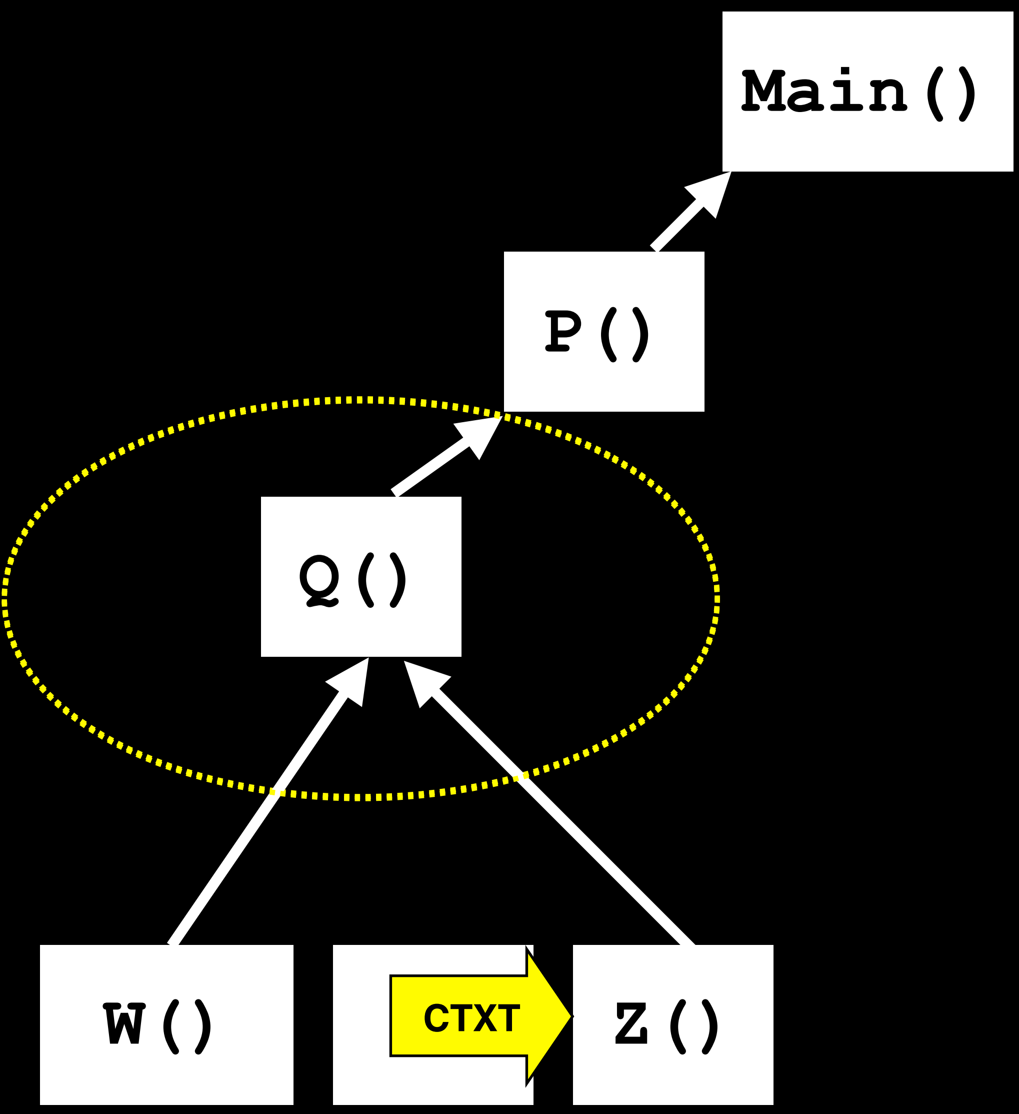


Tools can obtain pointer to the current context via "CTXT" in <u>constant time</u>

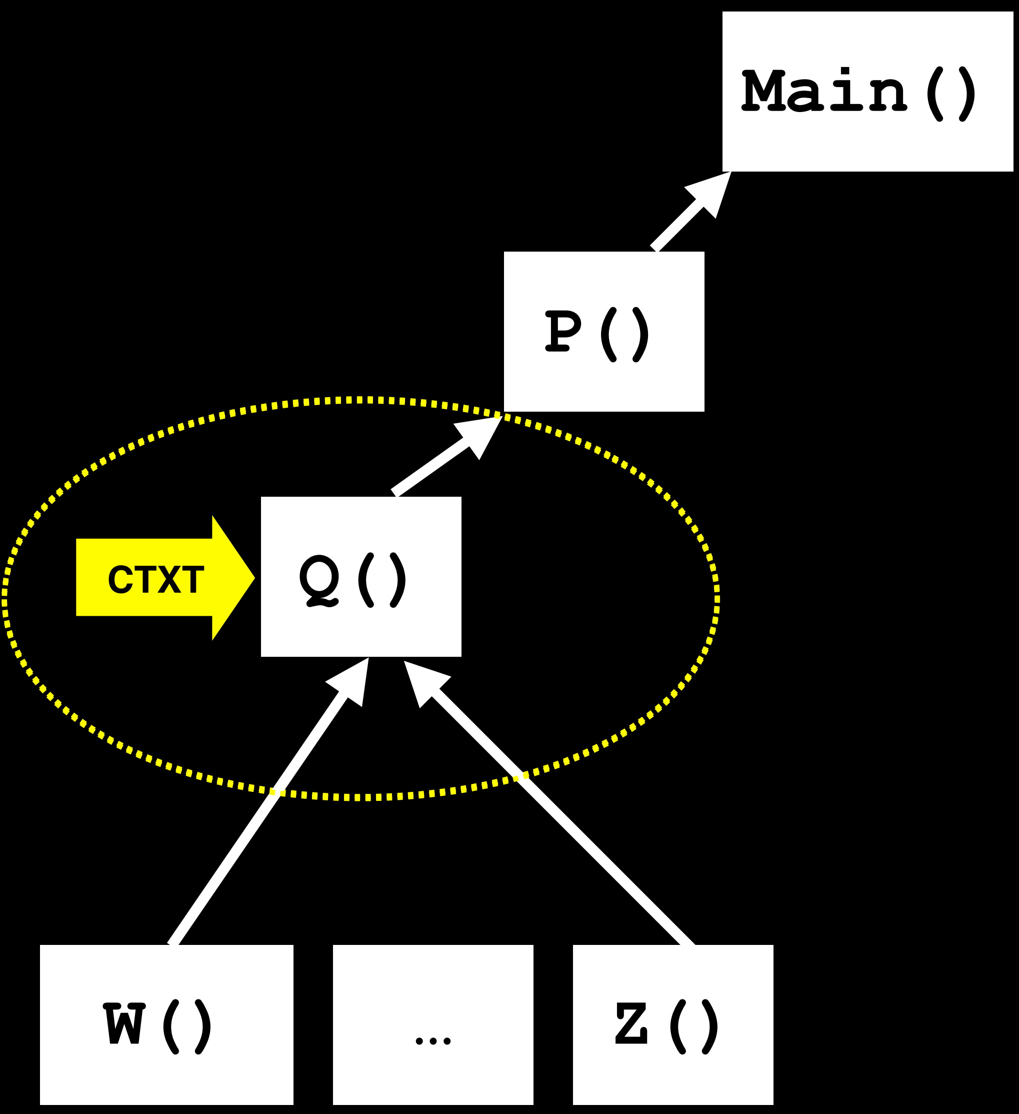
CTXT Update Cost



CTXT Update Cost

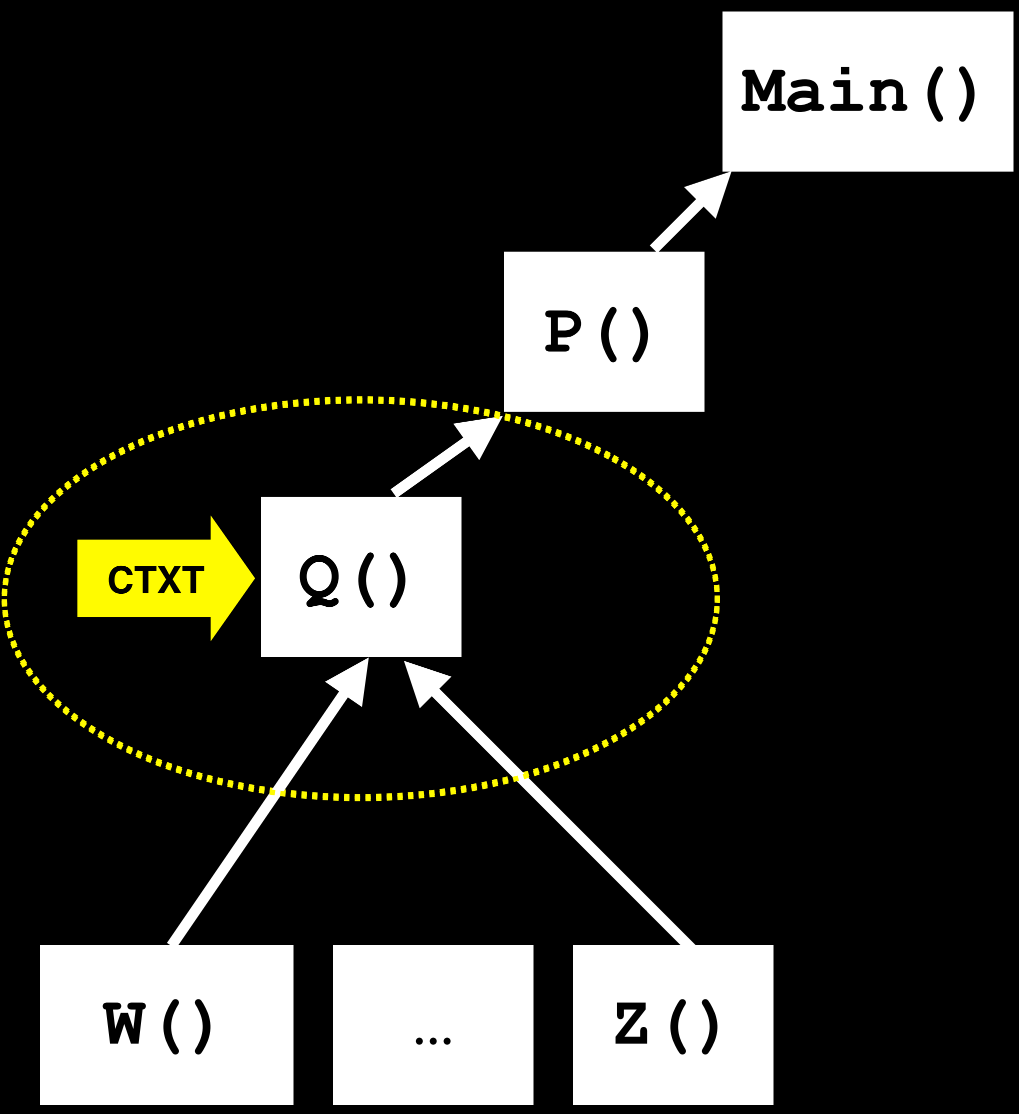


CTXT Update Cost

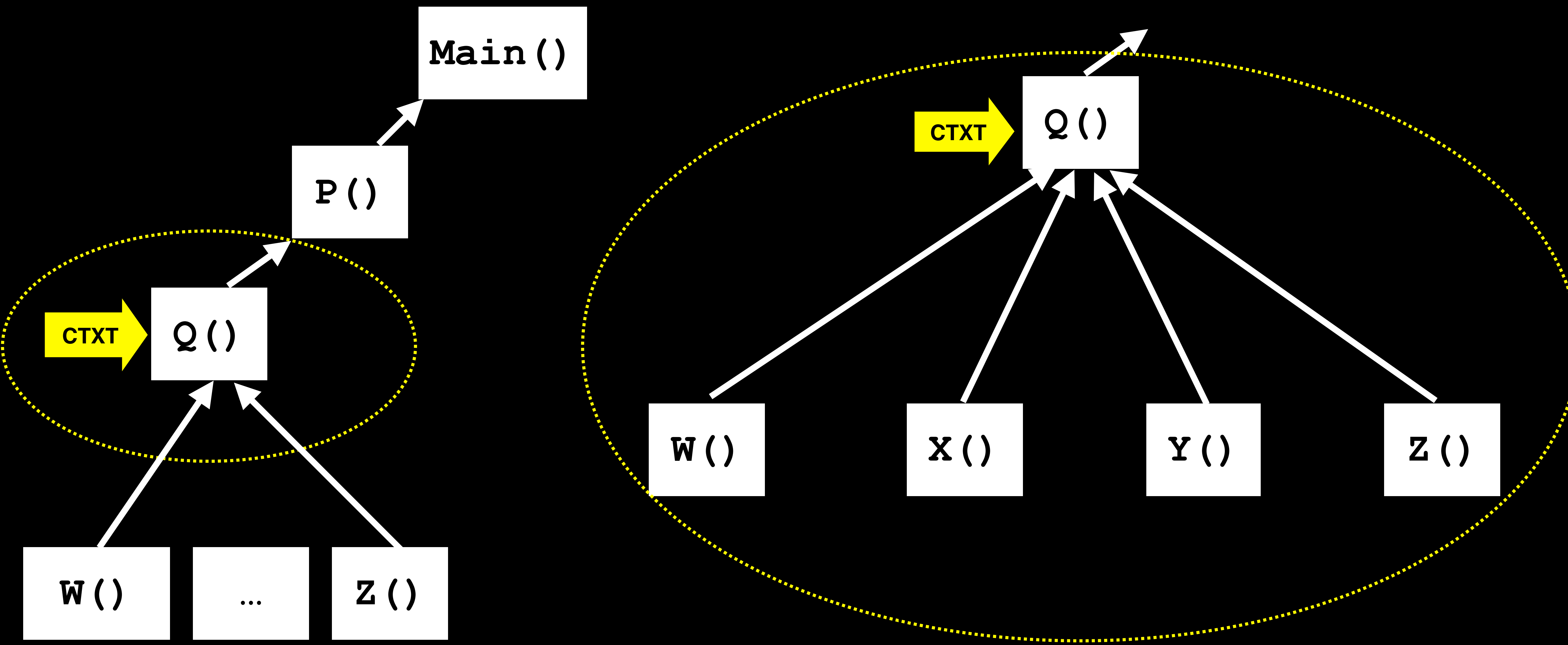


Return to caller
is constant time operation

Callee Lookup Could be Costly

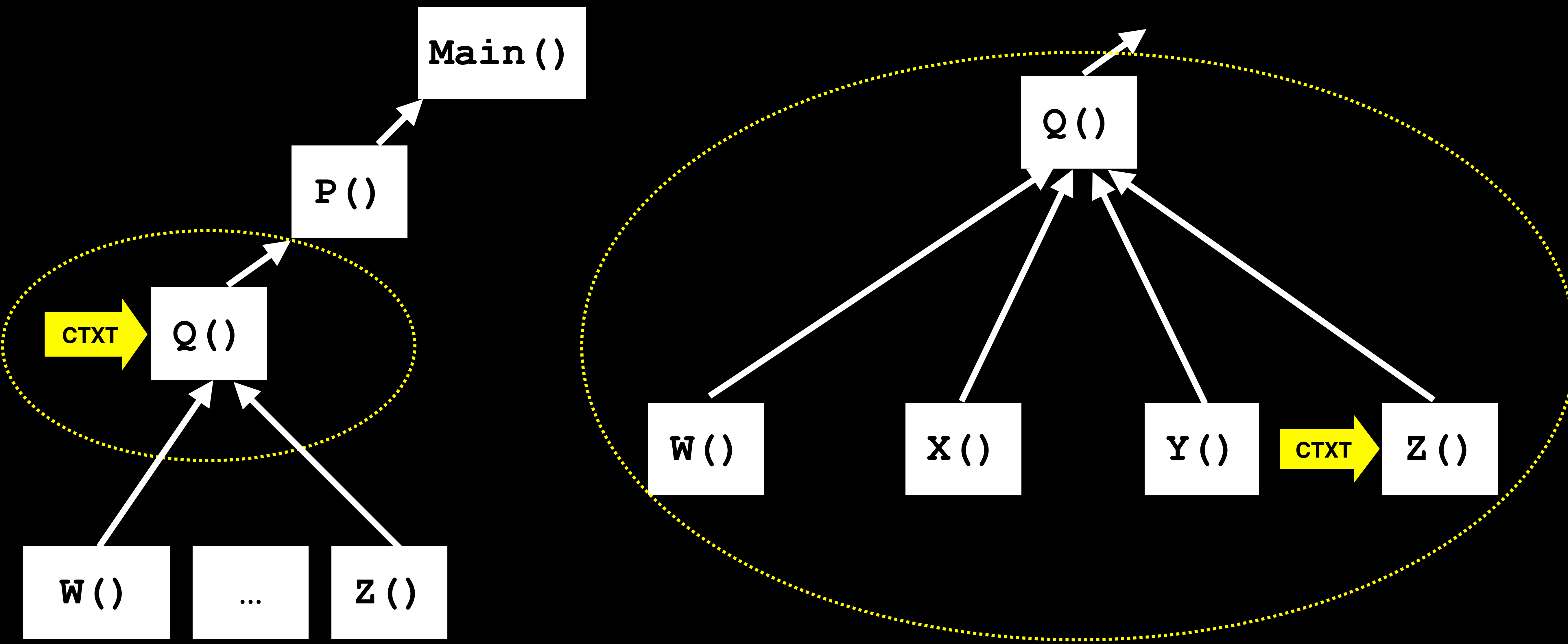


Callee Lookup Could be Costly



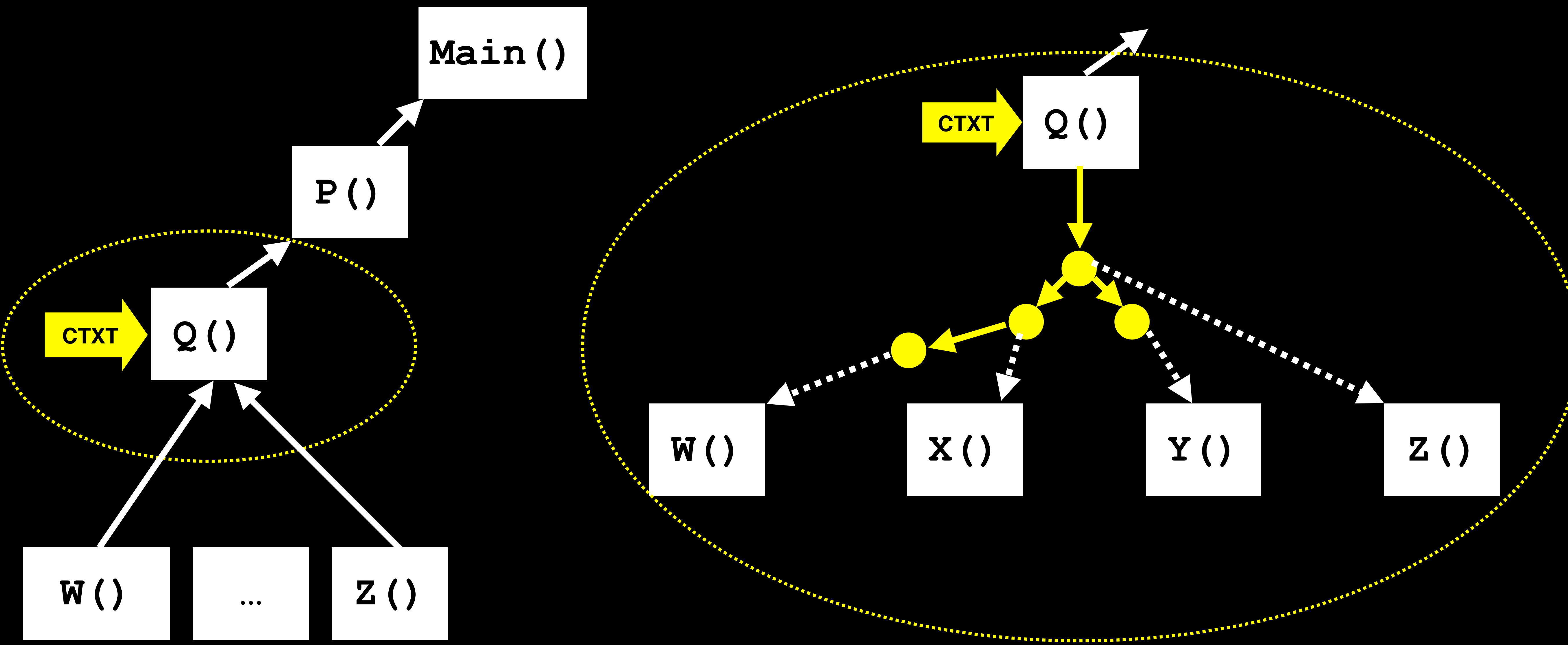
Each "Call" from caller "Q" to its callees incurs a **lookup cost**

Callee Lookup Could be Costly



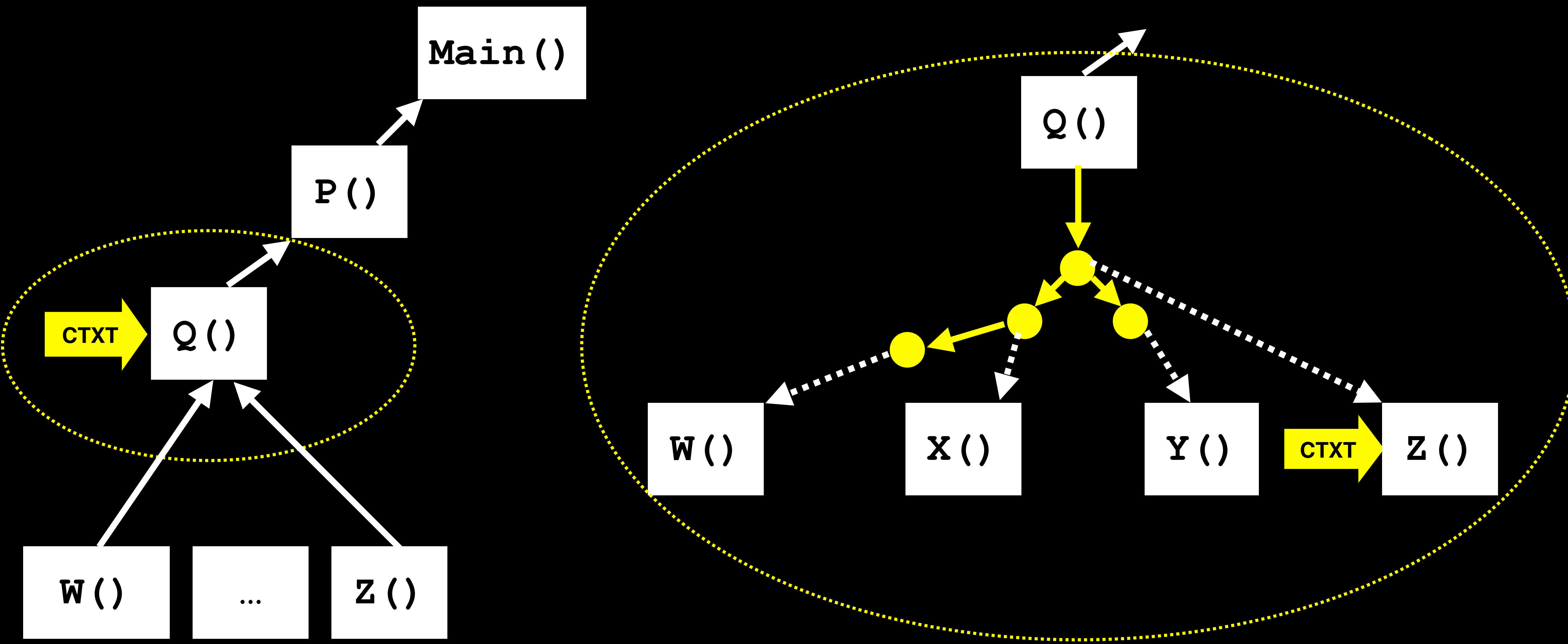
Each “Call” from caller “Q” to its callees incurs a **lookup cost**

Accelerating Lookup Cost with Splay Trees



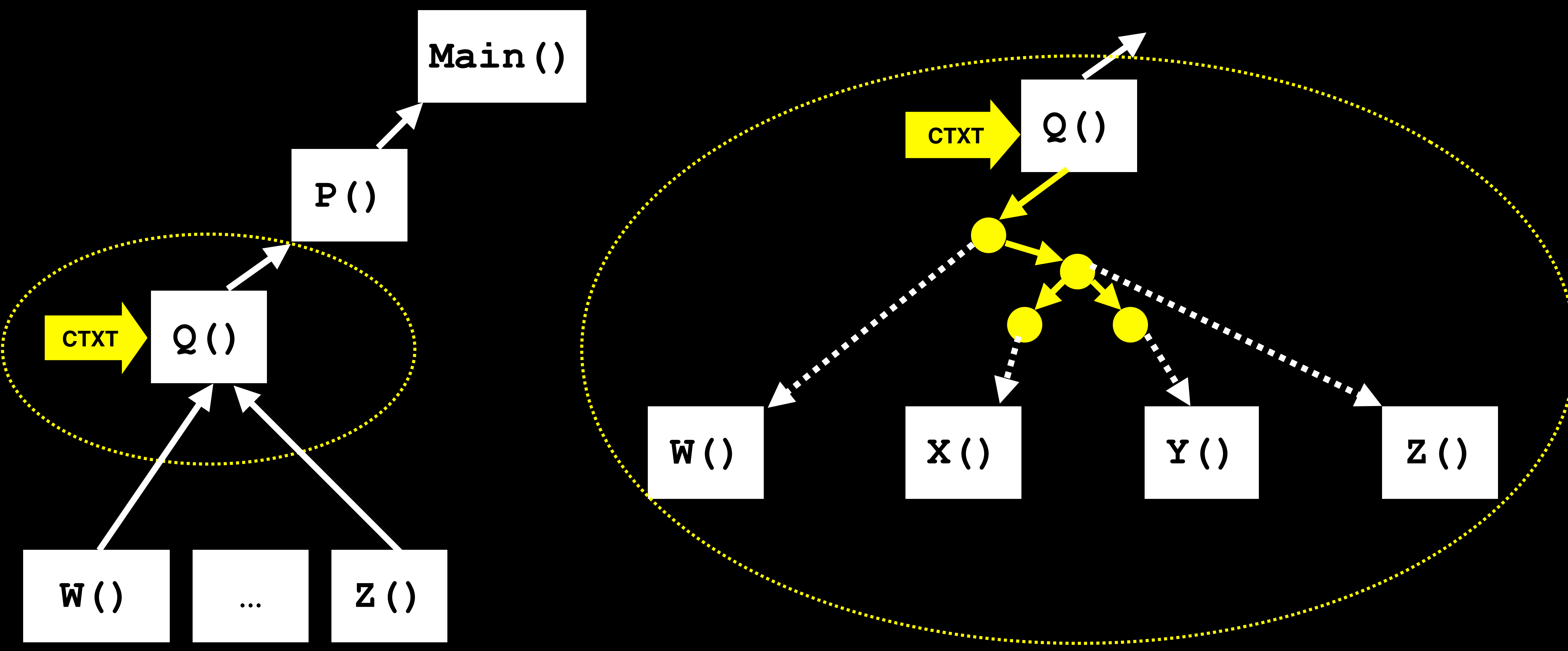
Splay tree ["Self-adjusting binary search trees" by Sleator et al. 1985] ensures frequently called functions are near the root of the tree

Accelerating Lookup Cost with Splay Trees



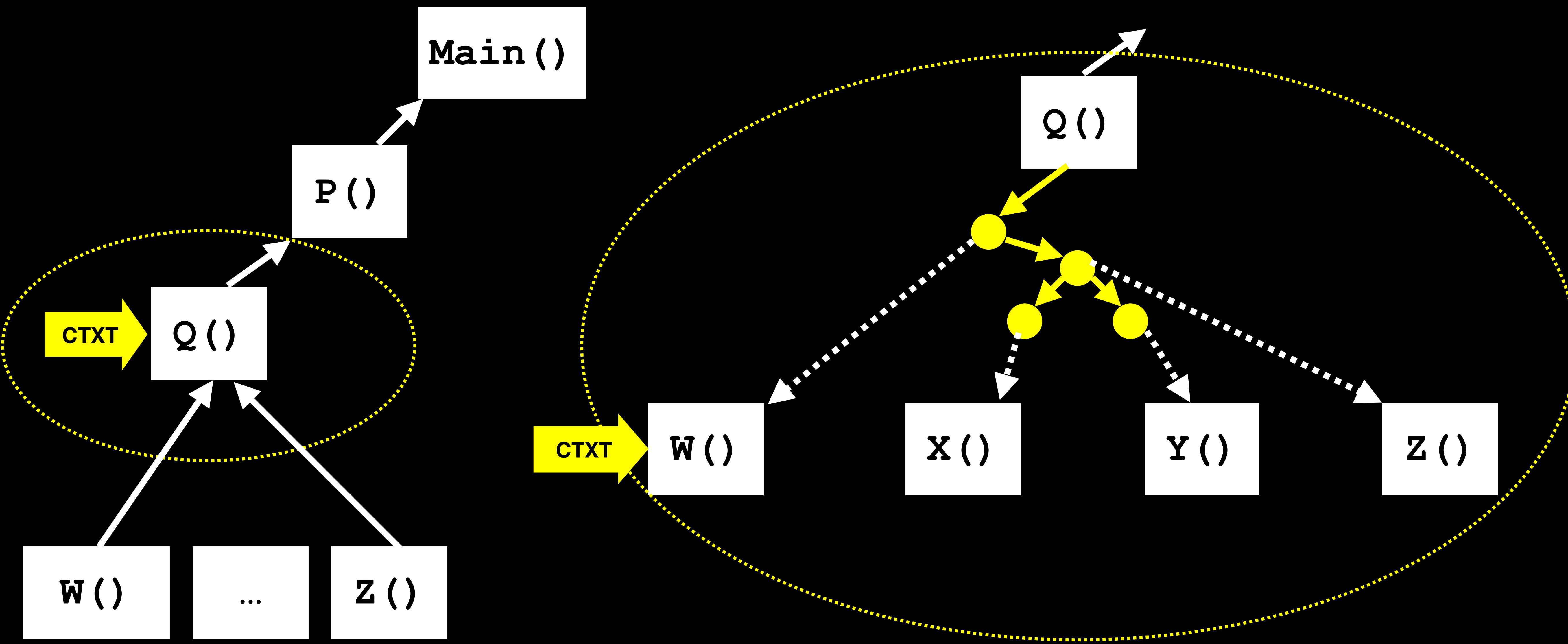
Splay tree ["Self-adjusting binary search trees" by Sleator et al. 1985] ensures frequently called functions are near the root of the tree

Accelerating Lookup Cost 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

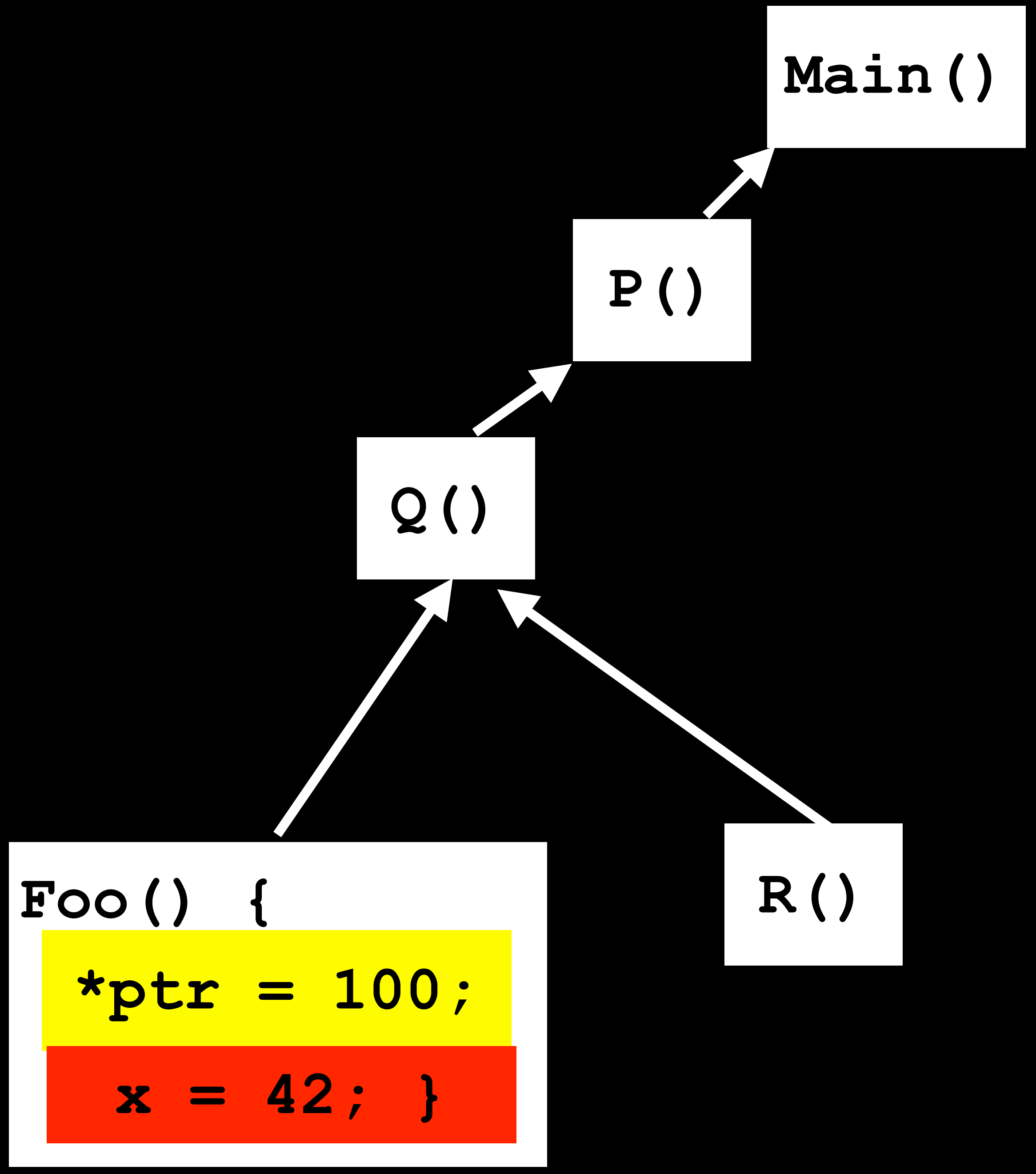
Accelerating Lookup Cost with Splay Trees



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

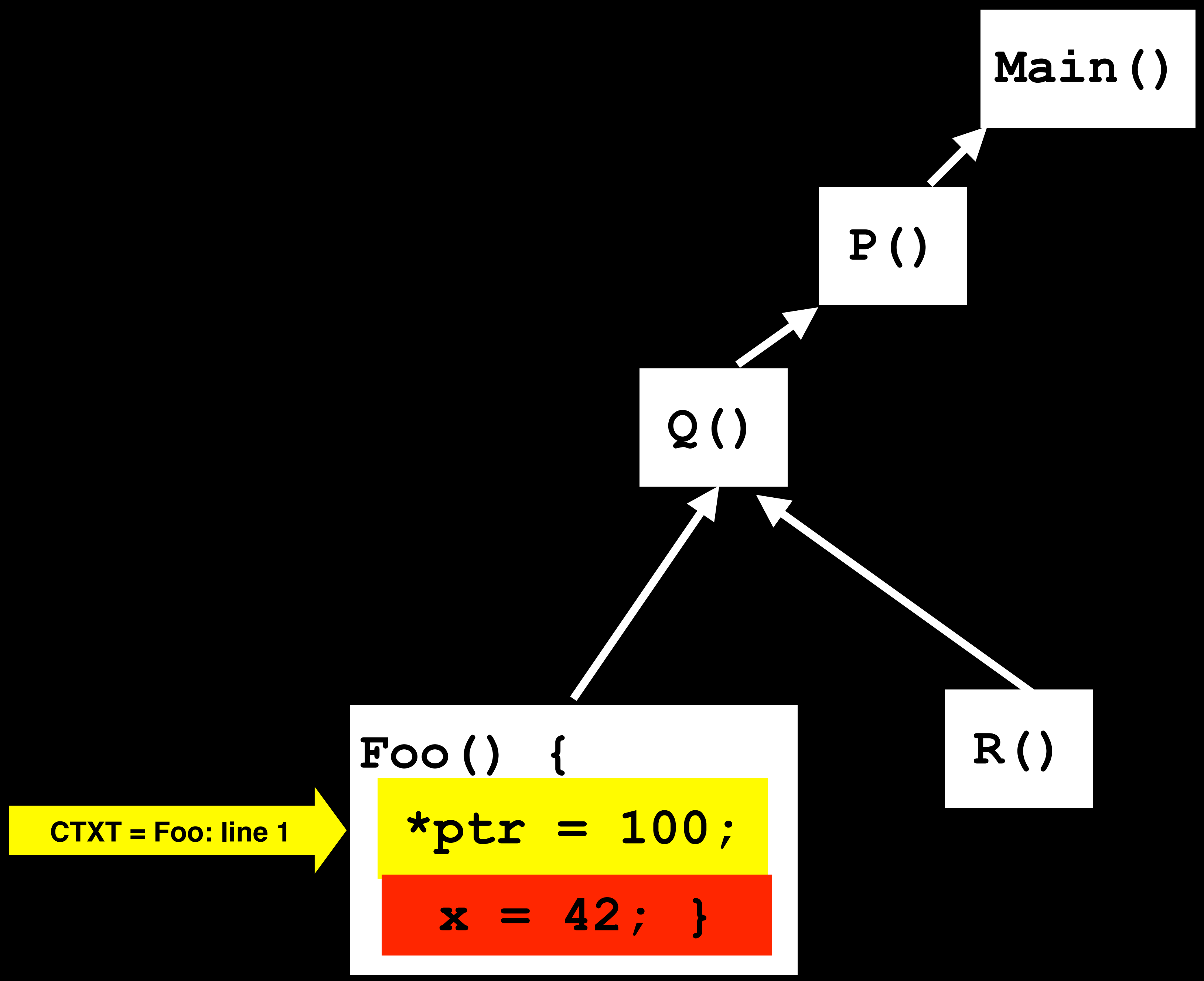
Other Complications in Real Programs

- Attributing to instructions/source lines (not just functions)



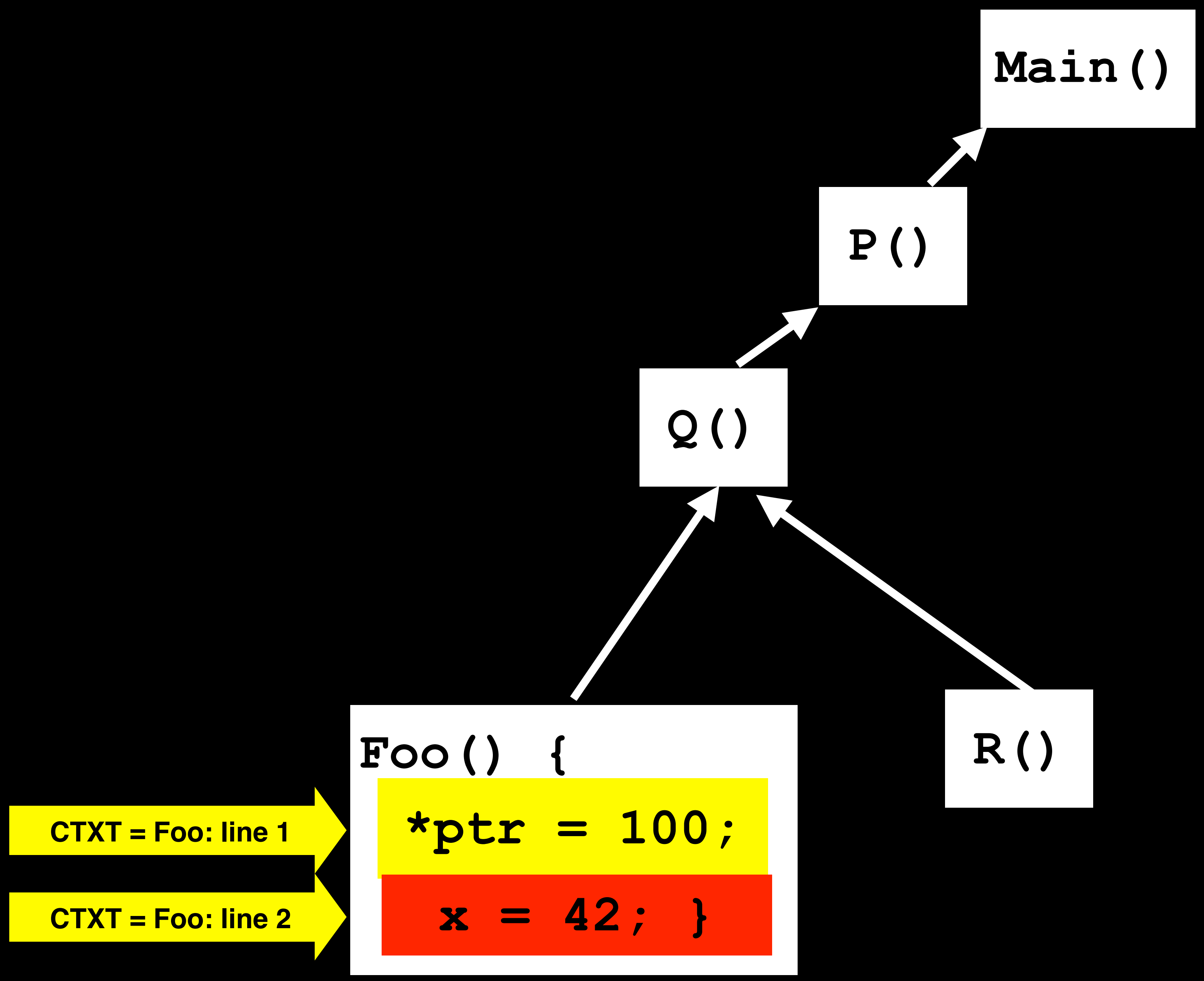
Other Complications in Real Programs

- Attributing to instructions/source lines (not just functions)



Other Complications in Real Programs

- Attributing to instructions/source lines (not just functions)



Other Complications in Real Programs

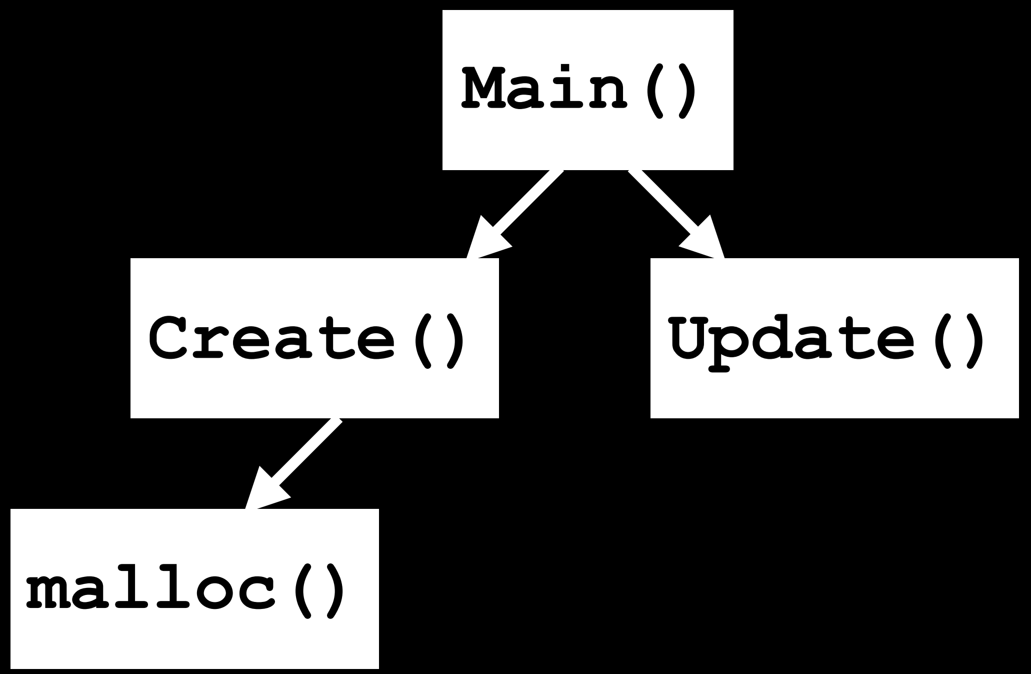
- Attributing to instructions/source lines (not just functions)
- Complex control flow
  - Signal handling
  - Setjmp-Longjmp
  - C++ exceptions (try-catch)
- Thread creation and destruction
  - Maintaining parent-child relationships between threads
  - Scalability to large number of threads

Data-Centric Attribution in CCTLib

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

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

Main(){
   int * p = Create();
   Update(p);
}
```



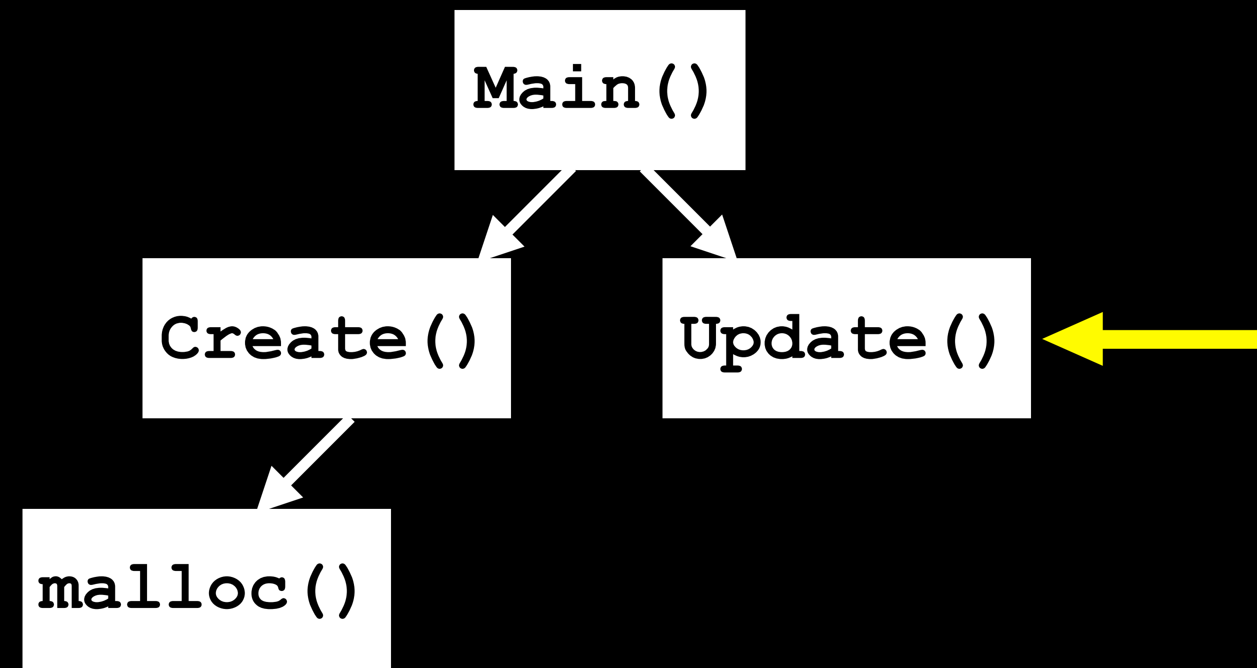
- Associate each data access to the corresponding <u>data object</u>
- <u>Data object:</u>
  - Dynamic allocation ➞ Call path of allocation site
  - Static objects ➞ Variable name

Data-Centric Attribution in CCTLib

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

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

Main(){
   int * p = Create();
   Update(p);
}
```



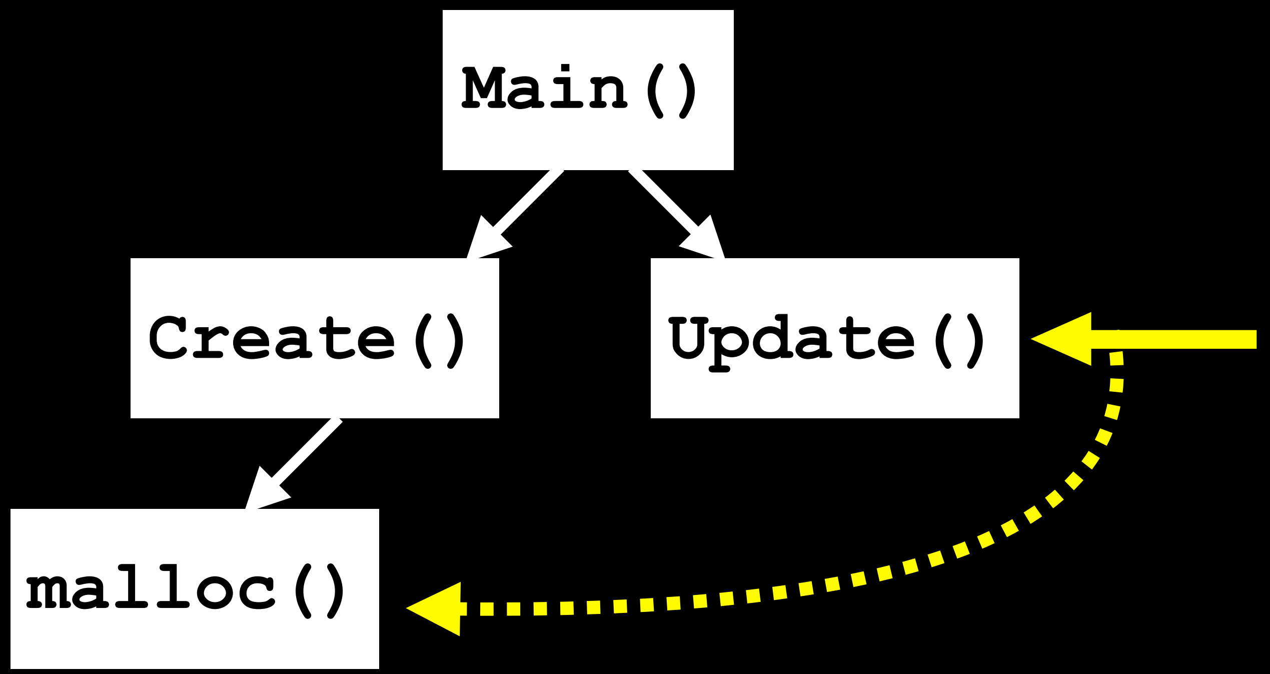
- Associate each data access to the corresponding <u>data object</u>
- <u>Data object:</u>
  - Dynamic allocation ➞ Call path of allocation site
  - Static objects ➞ Variable name

Data-Centric Attribution in CCTLib

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

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

Main(){
   int * p = Create();
   Update(p);
}
```



- Associate each data access to the corresponding data object
- Data object:
  - Dynamic allocation ➞ Call path of allocation site
  - Static objects ➞ Variable name

Details of Data-Centric Attribution

- How to perform data-centric attribution
  - Record all **<AddressRange, VariableName>** tuples in a map
  - Intercept all allocation/free routines and maintain **<AddressRange, CallPath>** tuples in a map
  - On each memory access search these maps for the address
- Problems:
  - Searching maps on each access is expensive
  - Maps need to be concurrent for threaded programs

Data-Centric Attribution via Balanced Trees

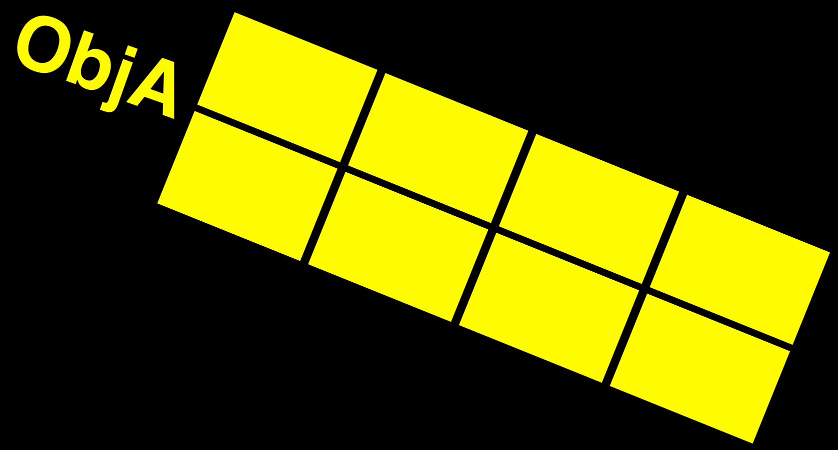
- Observation:
  - ✦ Updates to maps are infrequent
  - ✦ Lookups in maps are frequent
- Solution #1: sorted map
  - ✦ Keep N objects and associated address range in balanced binary trees
    - ★ Low memory cost—O(N), moderate lookup cost—O(log N)
    - ★ Concurrent access is handled by a novel replicated tree data structure

Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory

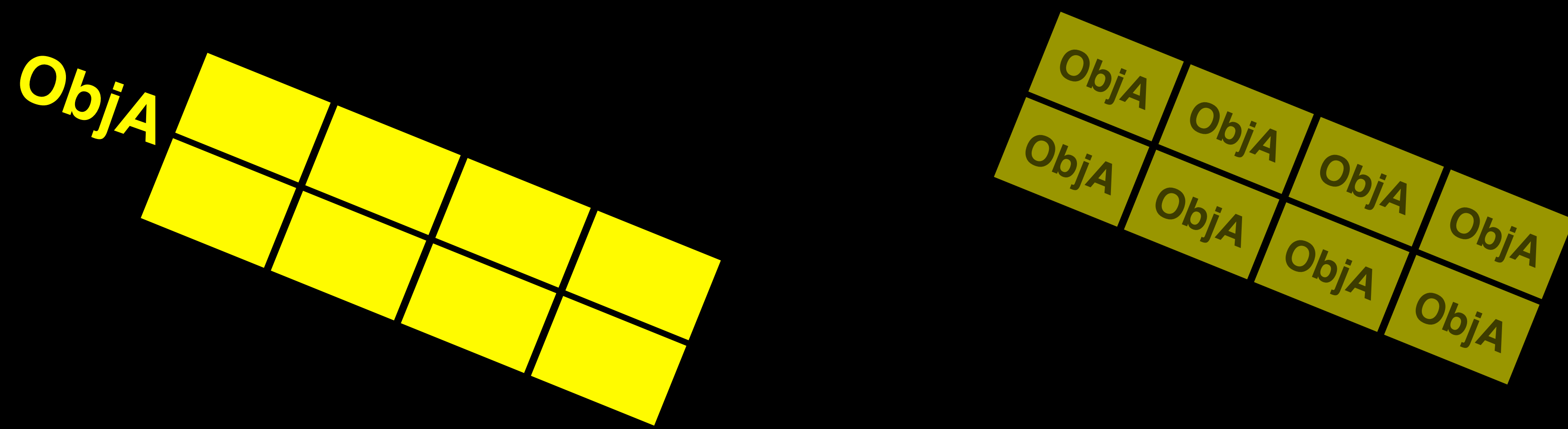
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory



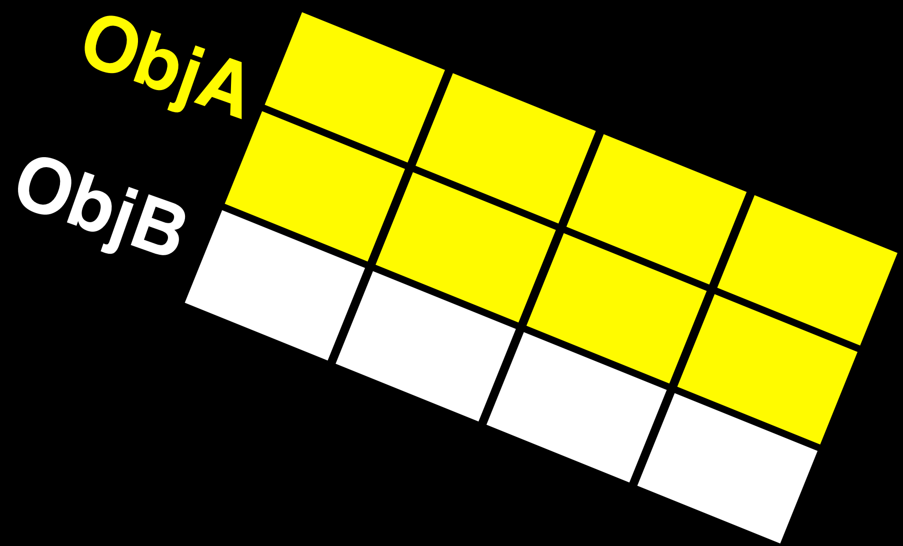
Data-Centric Attribution via Shadow Memory

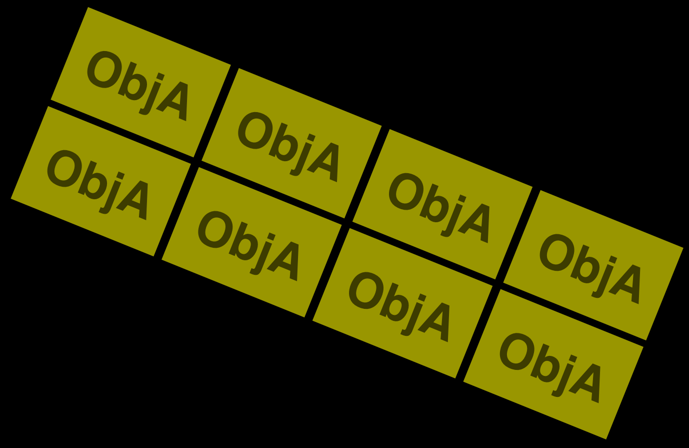
- Solutions #2: shadow memory



Data-Centric Attribution via Shadow Memory

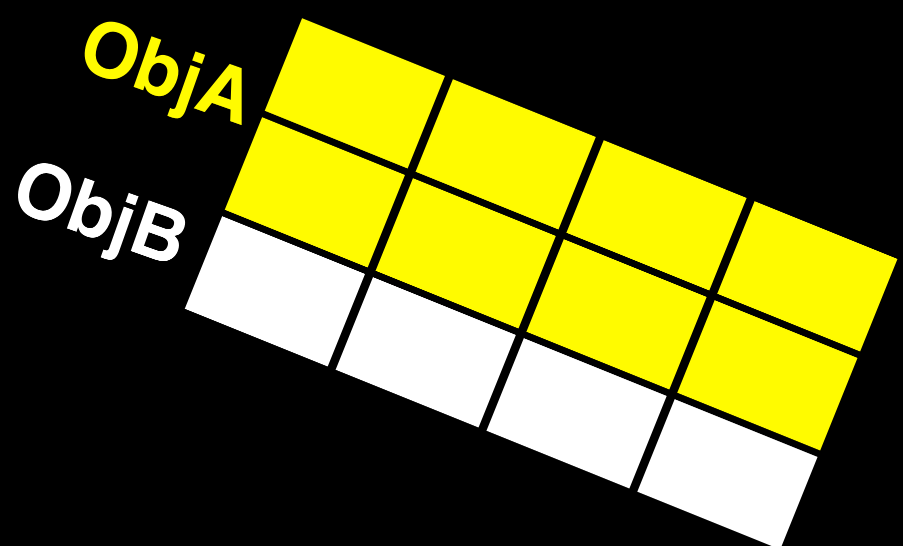
- Solutions #2: shadow memory

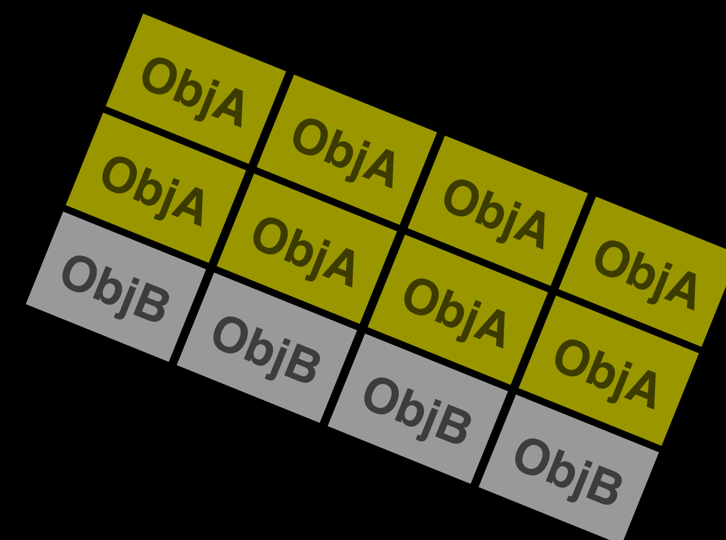




Data-Centric Attribution via Shadow Memory

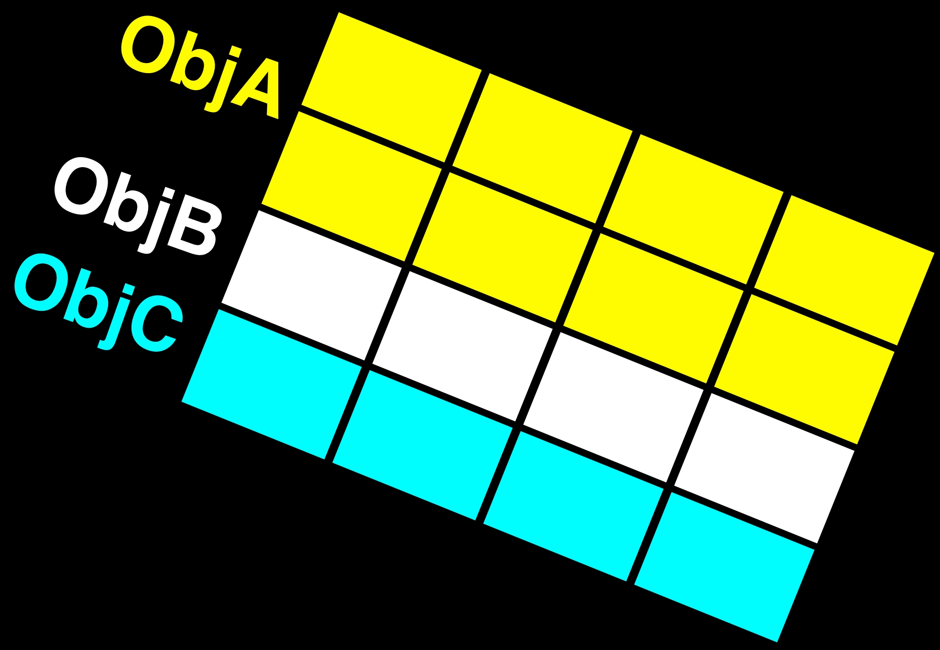
- Solutions #2: shadow memory





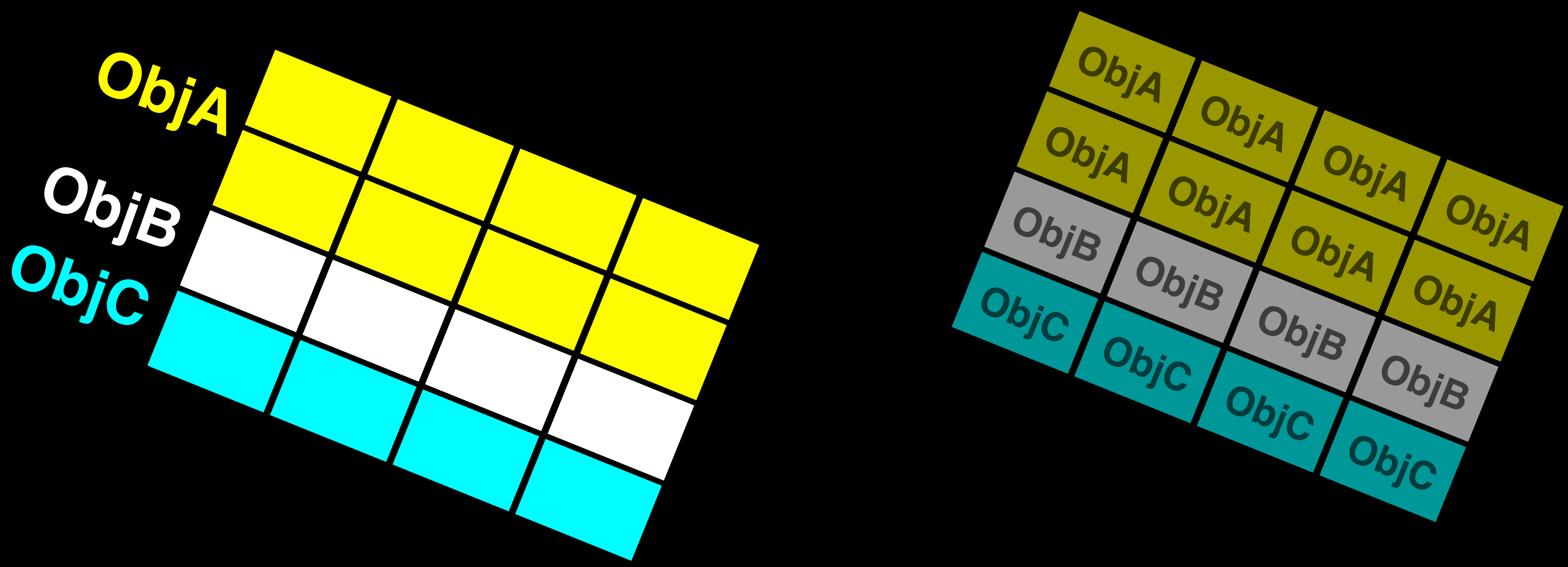
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory



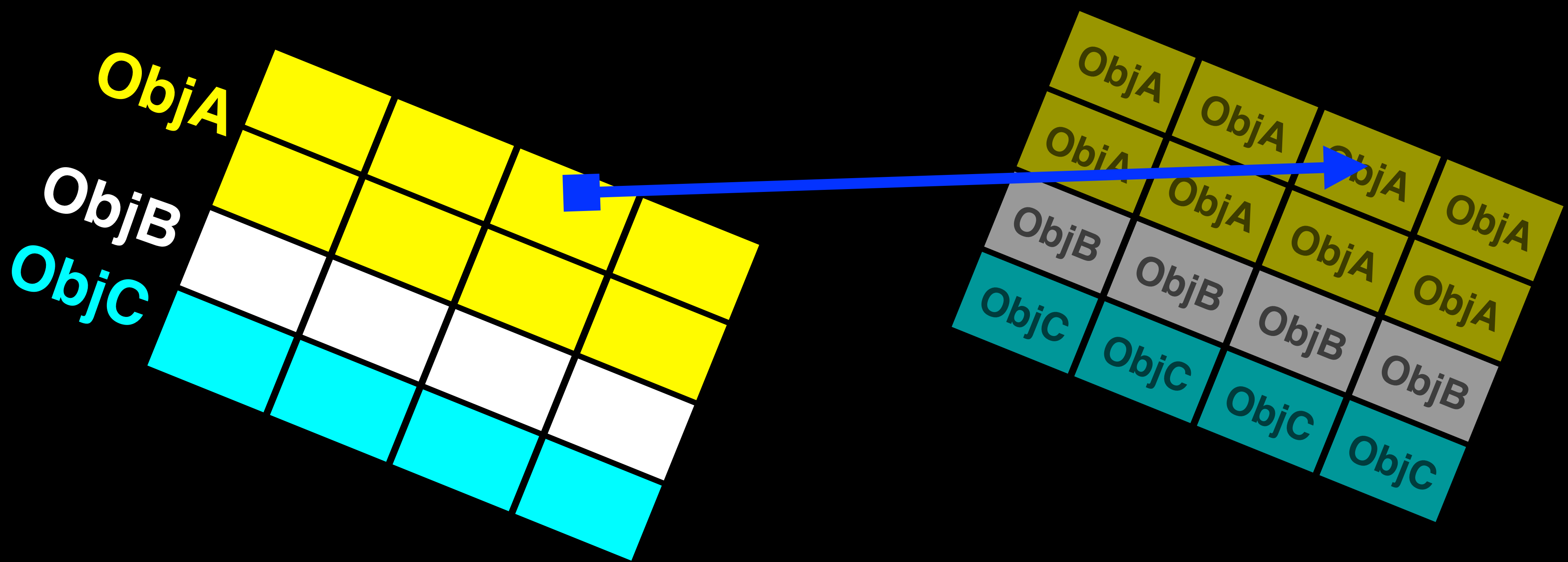
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory



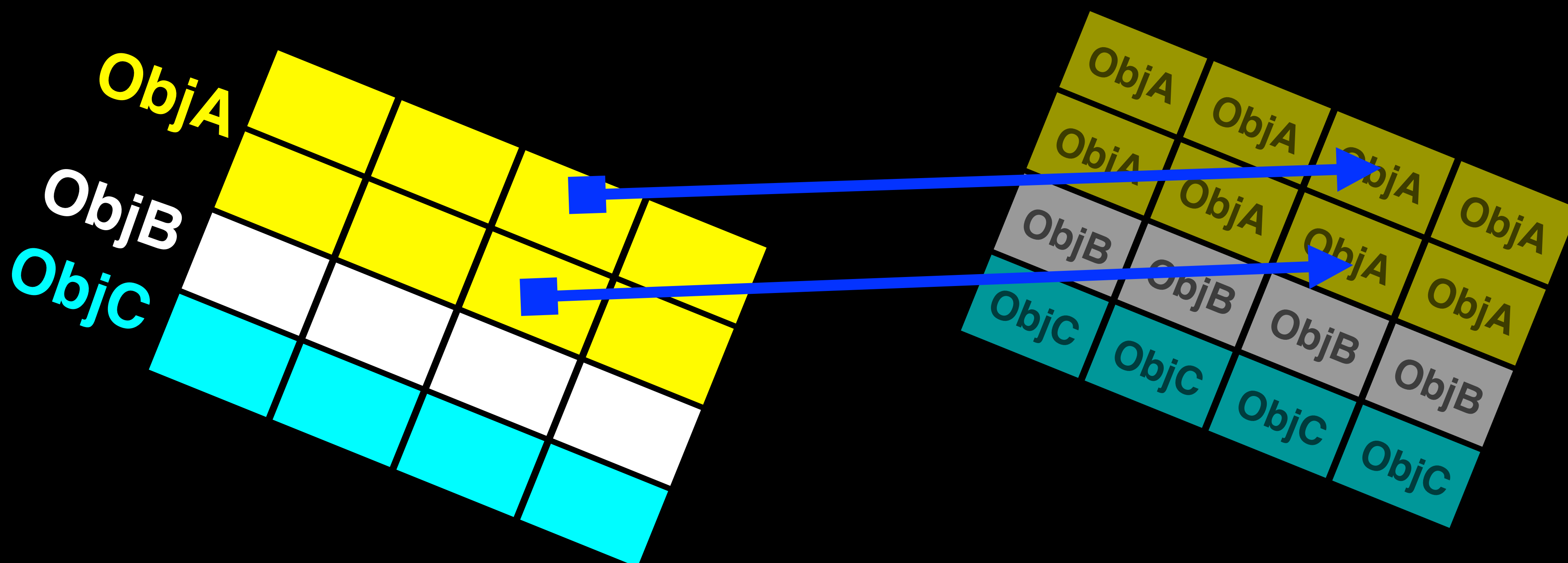
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory



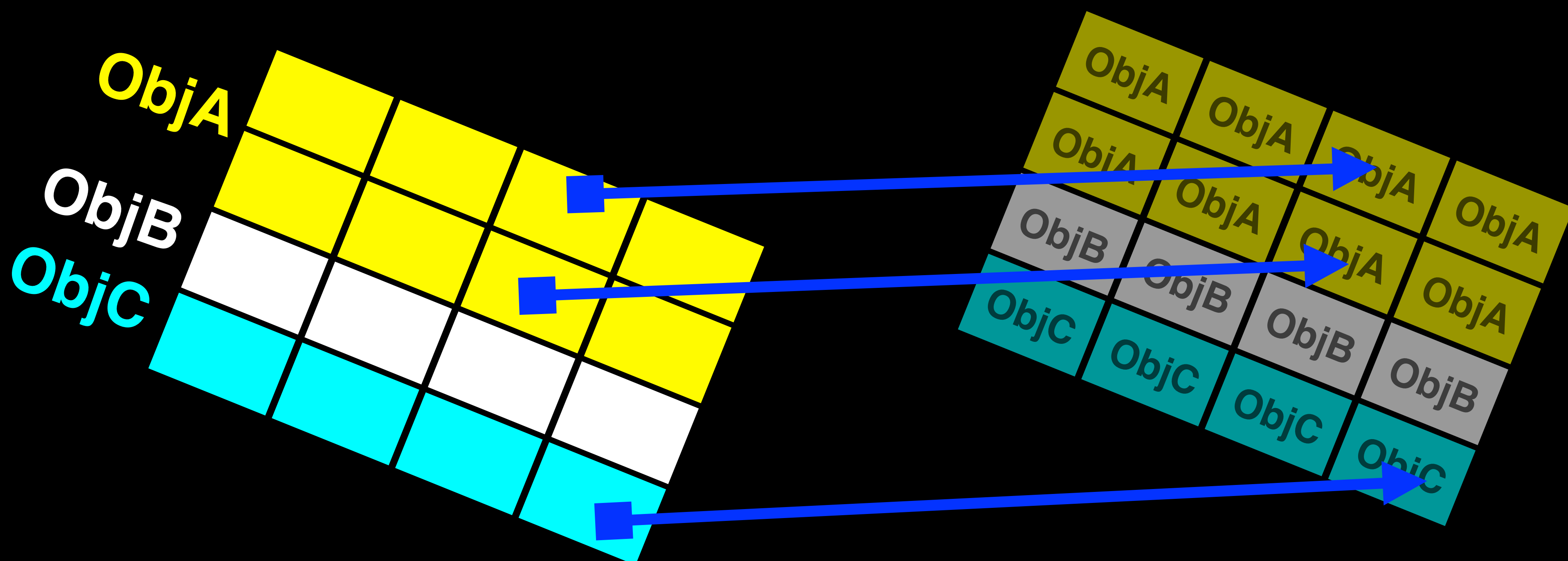
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory



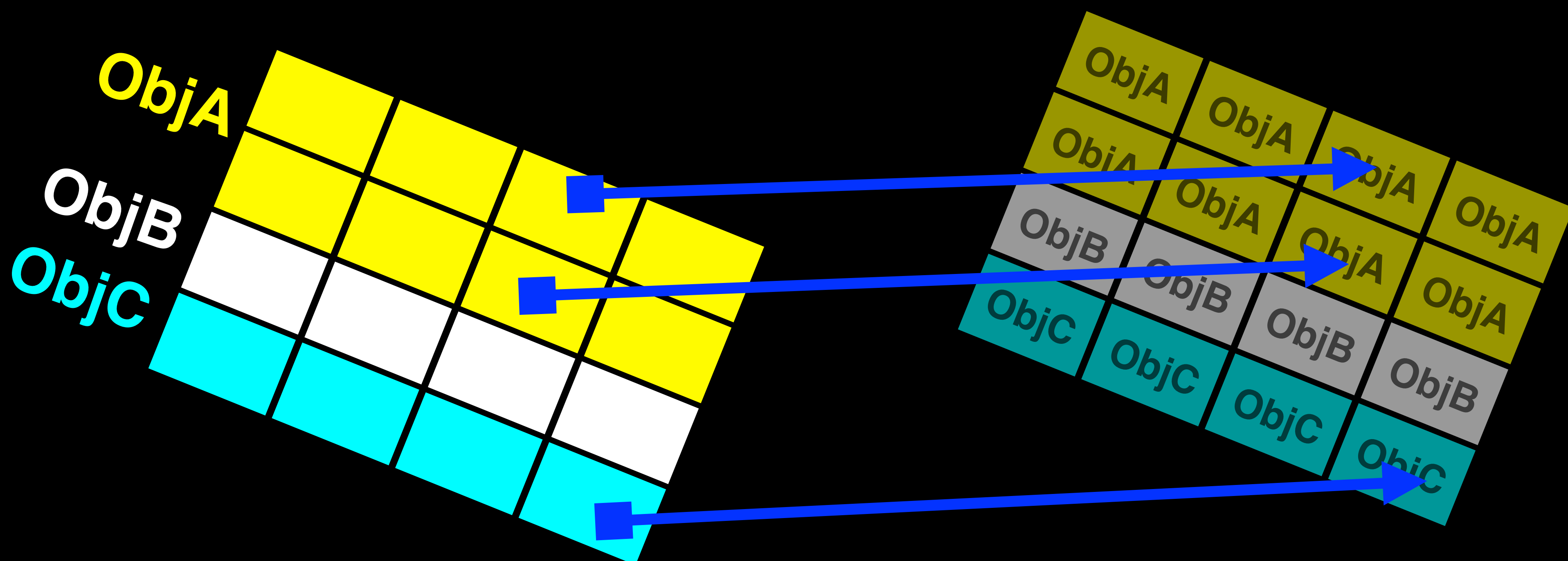
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory



Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory



- Have shadow memory for each memory cell and record a handle to the corresponding data object in the shadow memory
  - ★ Low lookup cost—O(1), high memory cost— $O(\sum_{i=1}^{N} sizeof(Obj(i)))$
  - ★ Concurrent access is not a problem
- CCTLib supports both solutions, clients can choose

Evaluation

- Time overhead
- Memory overhead
- Scaling on multithreaded programs
- Real world, long running programs

Time Overhead of CCTLib

Program	Time in sec	Call path on each memory access instruction	Call path on each instruction	Data-centric attribution on each instruction
astr	276.26	14x	22x	28x
bzip2	111.71	19x	32x	42x
gcc	44.61	23x	35x	44x
h264ref	260.12	31x	48x	67x
hmmer	326.32	21x	30x	47x
libquantum	462.38	22x	39x	46x
mcf	319.97	6x	10x	15x
omnetpp	352.30	14x	23x	34x
Xalan	294.80	32x	50x	65x
ROSE	23.64	30x	41x	49x
LAMMPS	99.28	17x	29x	40x
LULESH	67.29	20x	36x	48x
GEOMEAN		**19x**	**30x**	**41x**

Time Overhead of CCTLib

Spec Int 2006 reference benchmark

		Call path on each memory access instruction	Call path on each instruction	Data-centric attribution on each instruction
astr	276.26	14x	22x	28x
bzip2	111.71	19x	32x	42x
gcc	44.61	23x	35x	44x
h264ref	260.12	31x	48x	67x
hmmer	326.32	21x	30x	47x
libquantum	462.38	22x	39x	46x
mcf	319.97	6x	10x	15x
omnetpp	352.30	14x	23x	34x
Xalan	294.80	32x	50x	65x
ROSE	23.64	30x	41x	49x
LAMMPS	99.28	17x	29x	40x
LULESH	67.29	20x	36x	48x
GEOMEAN		**19x**	**30x**	**41x**

Time Overhead of CCTLib

Program	Time in sec	Call path on each memory access instruction	Call path on each instruction	Data-centric attribution on each instruction
astr	276.26	14x	22x	28x
bzip2	111.71	19x	32x	42x
gcc	44.61	23x	35x	44x
h264ref	260.12	31x	48x	67x
hmmer	[illegible]	[illegible]	[illegible]	47x
libquantum	[illegible]	[illegible]	[illegible]	46x
mcf	[illegible]	[illegible]	[illegible]	15x
omnetpp	352.30	14x	23x	34x
Xalan	294.80	32x	50x	65x
ROSE	23.64	30x	41x	49x
LAMMPS	99.28	17x	29x	40x
LULESH	67.29	20x	36x	48x
GEOMEAN		**19x**	**30x**	**41x**

Source-to-source compiler from LLNL
3M LOC compiling 70K LOC
Deep call chains

Time Overhead of CCTLib

Program	Time in sec	Call path on each memory access instruction	Call path on each instruction	Data-centric attribution on each instruction
astr	276.26	14x	22x	28x
bzip2	111.71	19x	32x	42x
gcc	44.61	23x	35x	44x
h264ref	260.12	31x	48x	67x
hmmer				47x
libquantum				46x
mcf				15x
omnetpp				34x
Xalan	294.80	32x	50x	65x
ROSE	23.64	30x	41x	49x
LAMMPS	99.28	17x	29x	40x
LULESH	67.29	20x	36x	48x
GEOMEAN		**19x**	**30x**	**41x**

Molecular dynamic code
500K LOC
Deep call chains
Multithreaded

Time Overhead of CCTLib

Program	Time in sec	Call path on each memory access instruction	Call path on each instruction	Data-centric attribution on each instruction
astr	276.26	14x	22x	28x
bzip2	111.71	19x	32x	42x
gcc	44.61	23x	35x	44x
h264ref	260.12	31x	48x	67x
hmmer	[illegible]	[illegible]	[illegible]	[illegible]
libquantum	[illegible]	[illegible]	[illegible]	[illegible]
mcf	[illegible]	[illegible]	[illegible]	[illegible]
omnetpp	[illegible]	[illegible]	[illegible]	[illegible]
Xalan	[illegible]	[illegible]	[illegible]	[illegible]
ROSE	23.64	30x	41x	49x
LAMMPS	99.28	17x	29x	40x
LULESH	67.29	20x	36x	48x
GEOMEAN		**19x**	**30x**	**41x**

Hydrodynamics mini-app from LLNL
Frequent data allocation and de-allocations
Memory bounded
Multithreaded, Poor scaling

Time Overhead of CCTLib

Program	Time in sec	Call path on each memory access instruction	Call path on each instruction	Data-centric attribution on each instruction
astr	276.26	14x	22x	28x
bzip2	111.71	19x	32x	42x
gcc	44.61	23x	35x	44x
h264ref	260.12	31x	48x	67x
hmmer	326.32	21x	30x	47x
libquantum	462.38	22x	39x	46x
mcf	319.97	6x	10x	15x
omnetpp	352.30	14x	23x	34x
Xalan	294.80	32x	50x	65x
ROSE	23.64	30x	41x	49x
LAMMPS	99.28	17x	29x	40x
LULESH	67.29	20x	36x	48x
GEOMEAN		**19x**	**30x**	**41x**

Memory Overhead of CCTLib

Program	Original resident memory in MB	Call path on each memory access instruction	Call path on each instruction	Data-centric attribution via binary tree	Data-centric attribution via shadow memory
astr	230	1.16x	1.17x	1.34x	8.65x
bzip2	561	1.11x	1.12x	1.12x	8.03x
gcc	453	15.62x	25.97x	26.03x	36.38x
h264ref	37	2.49x	2.69x	2.91x	11.47x
hmmer	15	4.38x	4.36x	5.13x	29.39x
libquantum	96	1.28x	1.30x	1.32x	11.91x
mcf	1677	1.02x	1.03x	1.03x	6.53x
omnetpp	170	1.87x	2.35x	3.76x	10.54x
Xalan	419	25.86x	38.60x	39.12x	46.63x
ROSE	380	64.38x	98.15x	100.12x	105.43x
LAMMPS	110	1.58x	1.57x	1.70x	16.88x
LULESH	26	2.28x	2.27x	2.51x	9.11x
GEOMEAN		**3.49x**	**4x**	**4.38x**	**16.86x**

Memory Overhead of CCTLib

Program	Original resident memory in MB	Call path on each memory access instruction	Call path on each instruction	Data-centric attribution via binary tree	Data-centric attribution via shadow memory
astr	230	1.16x	1.17x	1.34x	8.65x
bzip2	561	1.11x	1.12x	1.12x	8.03x
gcc	453	15.62x	25.97x	26.03x	36.38x
h264ref	37	2.49x	2.69x	2.91x	11.47x
hmmer	15	4.38x	4.36x	5.13x	29.39x
libquantum	96	1.28x	1.30x	1.32x	11.91x
mcf	1677	1.02x	1.03x	1.03x	6.53x
omnetpp	170	1.87x	2.35x	3.76x	10.54x
Xalan	419	25.86x	38.60x	39.12x	46.63x
ROSE	380	64.38x	98.15x	100.12x	105.43x
LAMMPS	110	1.58x	1.57x	1.70x	16.88x
LULESH	26	2.28x	2.27x	2.51x	9.11x
GEOMEAN		**3.49x**	**4x**	**4.38x**	**16.86x**

Memory Overhead of CCTLib

Program	Original resident memory in MB	Call path on each memory access instruction	Call path on each instruction	Data-centric attribution via binary tree	Data-centric attribution via shadow memory
astr	230	1.16x	1.17x	1.34x	8.65x
bzip2	561	1.11x	1.12x	1.12x	8.03x
gcc	453	15.62x	25.97x	26.03x	36.38x
h264ref	37	2.49x	2.69x	2.91x	11.47x
hmmer	15	4.38x	4.36x	5.13x	29.39x
libquantum	96	1.28x	1.30x	1.32x	11.91x
mcf	1677	1.02x	1.03x	1.03x	6.53x
omnetpp	170	1.87x	2.35x	3.76x	10.54x
Xalan	419	25.86x	38.60x	39.12x	46.63x
ROSE	380	64.38x	98.15x	100.12x	105.43x
LAMMPS	110	1.58x	1.57x	1.70x	16.88x
LULESH	26	2.28x	2.27x	2.51x	9.11x
GEOMEAN		**1.71x**	**1.77x**	**1.99x**	**11.28x**

CCTLib is Scalable

CCTLib overhead of n threads: $$OH(n) = \frac{R_c(n)}{R_o(n)}$$

CCTLib scalability for n threads: $$S(n) = \frac{OH(1)}{OH(n)}$$

<u>Higher scalability is better, 1.0 is ideal</u>

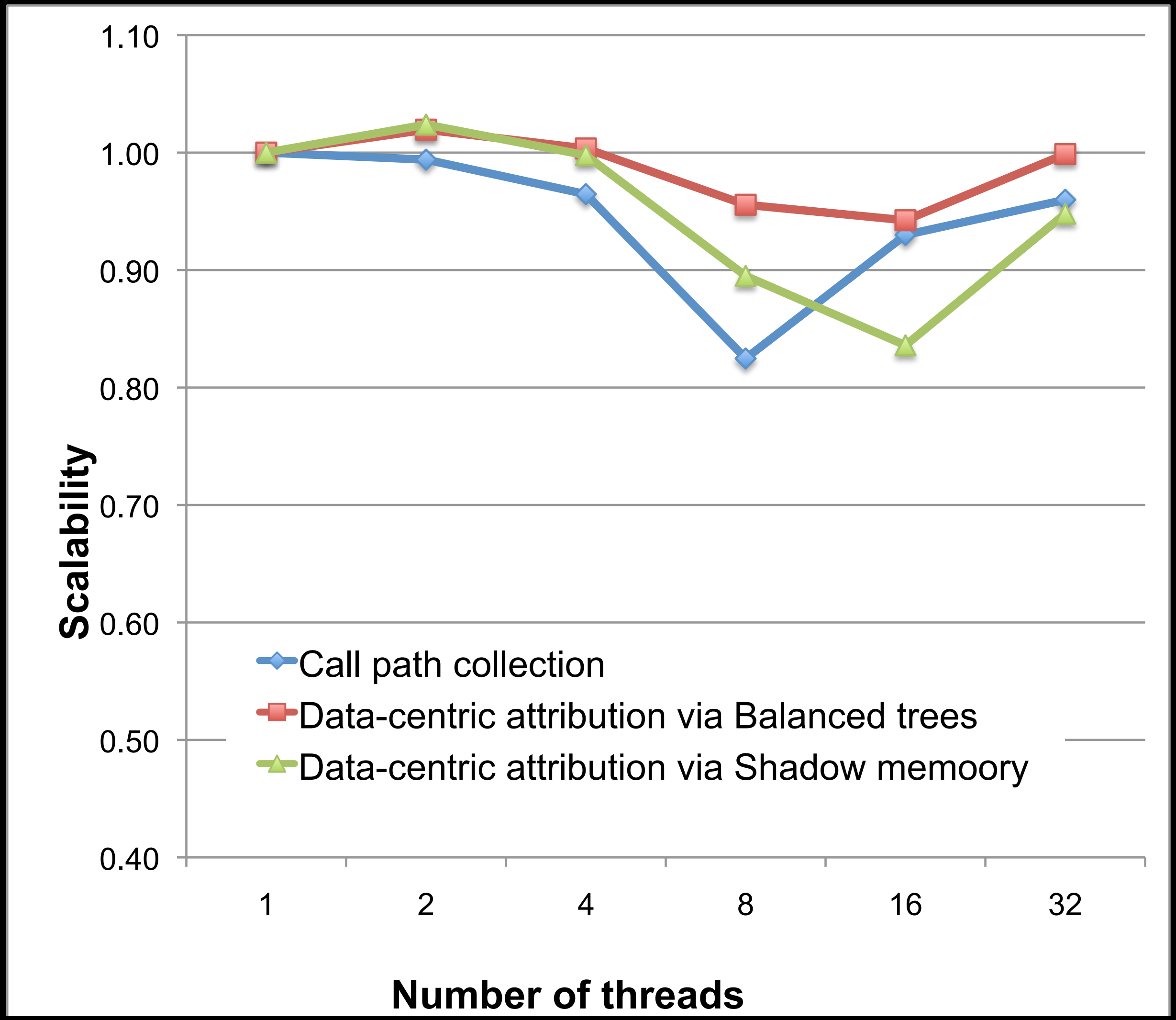
CCTLib is Scalable

CCTLib overhead of n threads: $OH(n) = \frac{R_c(n)}{R_o(n)}$

CCTLib scalability for n threads: $S(n) = \frac{OH(1)}{OH(n)}$

Higher scalability is better, 1.0 is ideal

CCTLib scalability on LAMMPS



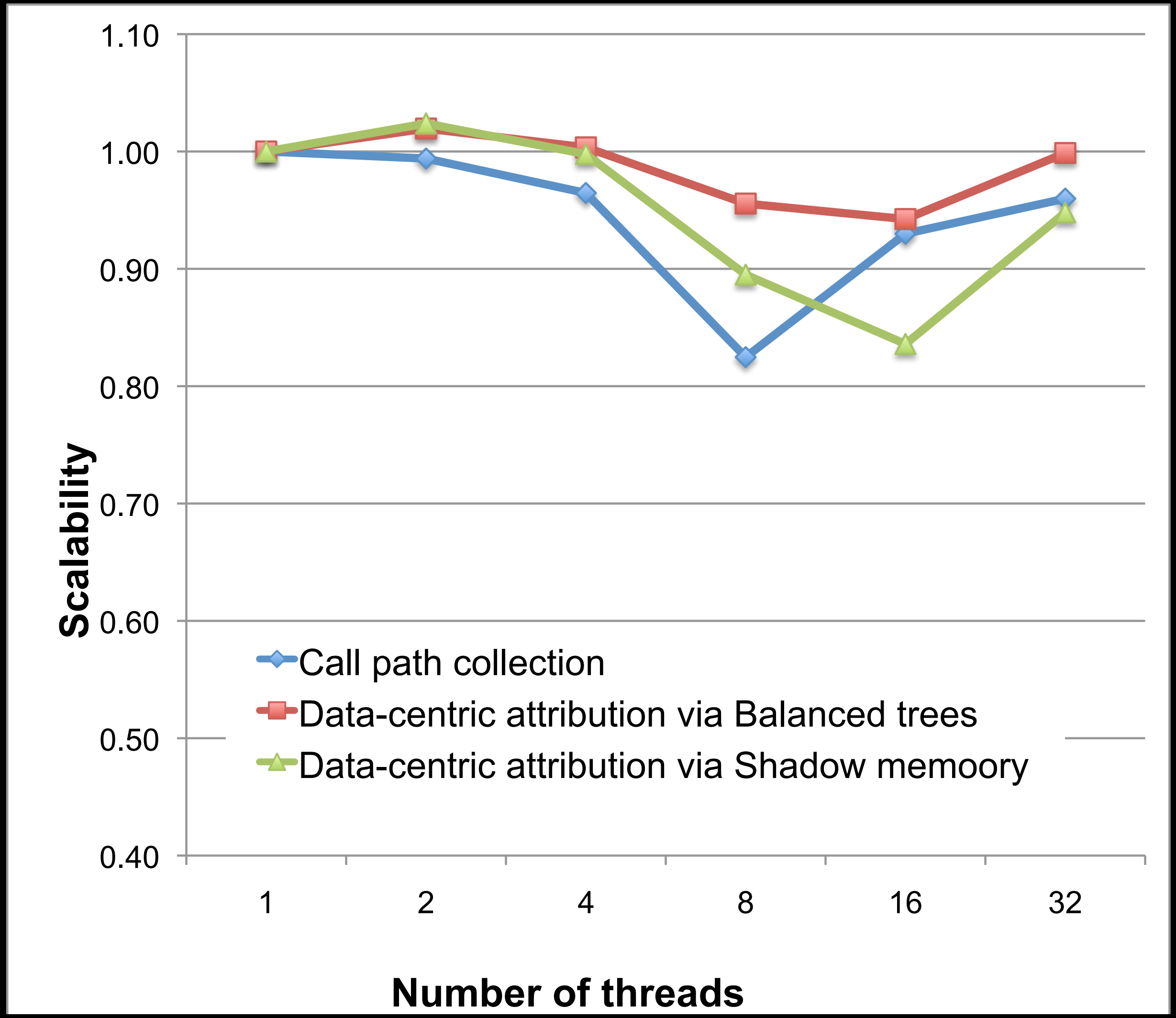
CCTLib is Scalable

CCTLib overhead of n threads: $OH(n) = \frac{R_c(n)}{R_o(n)}$

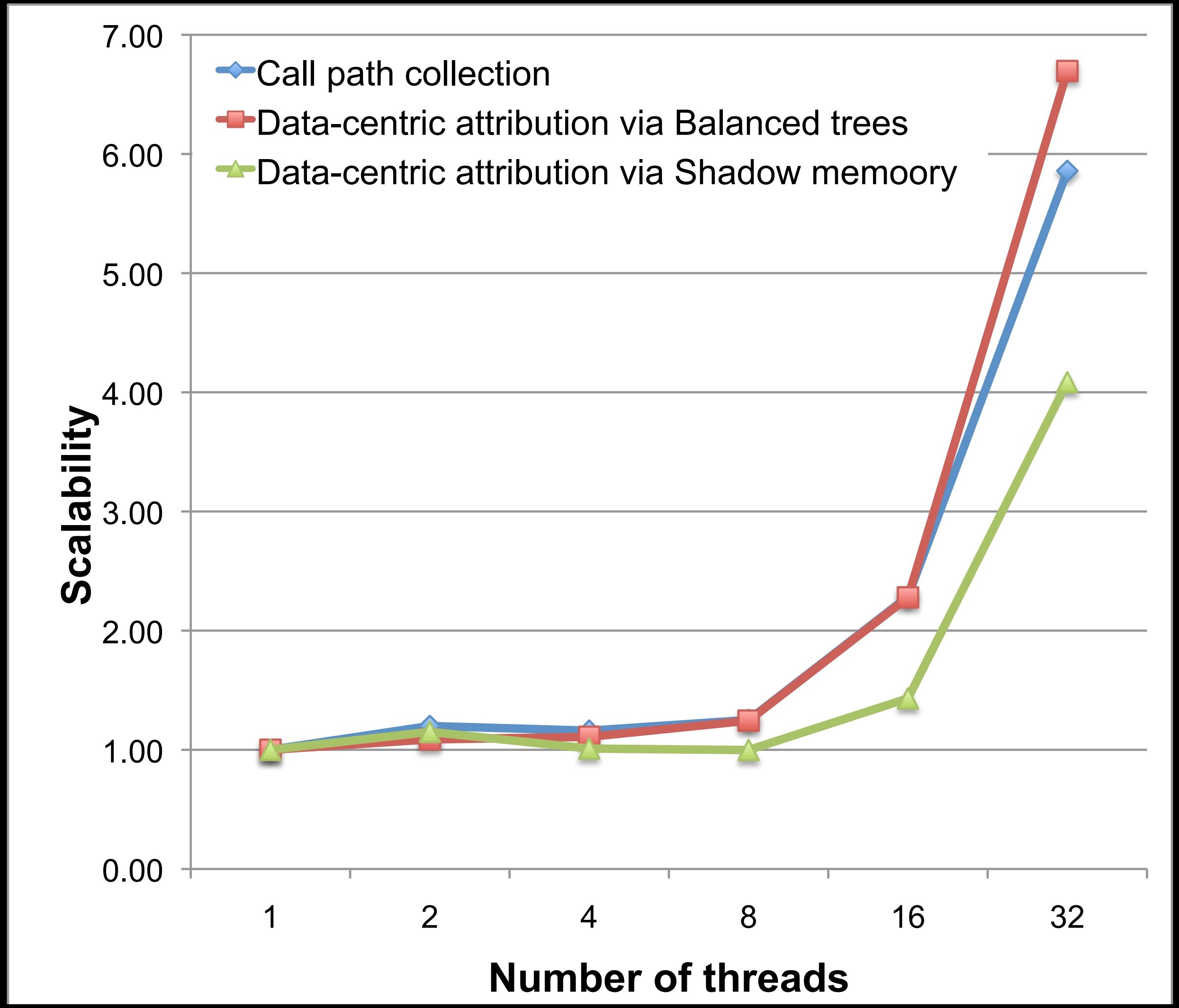
CCTLib scalability for n threads: $S(n) = \frac{OH(1)}{OH(n)}$

<u>Higher scalability is better, 1.0 is ideal</u>

<u>CCTLib scalability on LAMMPS</u>



<u>CCTLib scalability on LULESH</u>



Conclusions

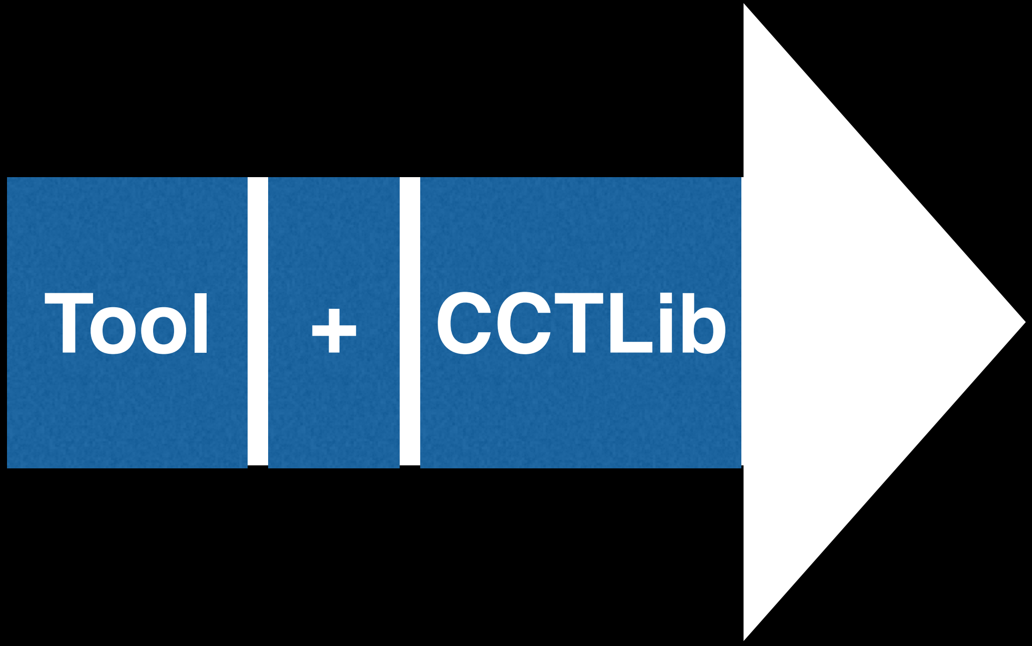
- Enhance diagnostic capabilities of fine-grained execution monitoring tools by associating each
  - Instruction ➞ calling context (code-centric attribution)
  - Memory address ➞ data object (data-centric attribution)
- Ubiquitous calling context collection and data-centric attribution is expensive (both memory and time)
- **CCTLib**
  - Provides calling context for Pin tools
  - Achieves ubiquitous code- and data-centric attribution via appropriate choice of algorithms and data structures
- **CCTLib** enables efficient construction of Pin tools that need detailed attribution of costs to contexts or data

CCTLib Enhances Tool Usability

User

CCTLib Enhances Tool Usability

Inventor



User

https://code.google.com/p/cctlib/