

TypeCraft: A Lightweight Data Type Profiler with High Resolution

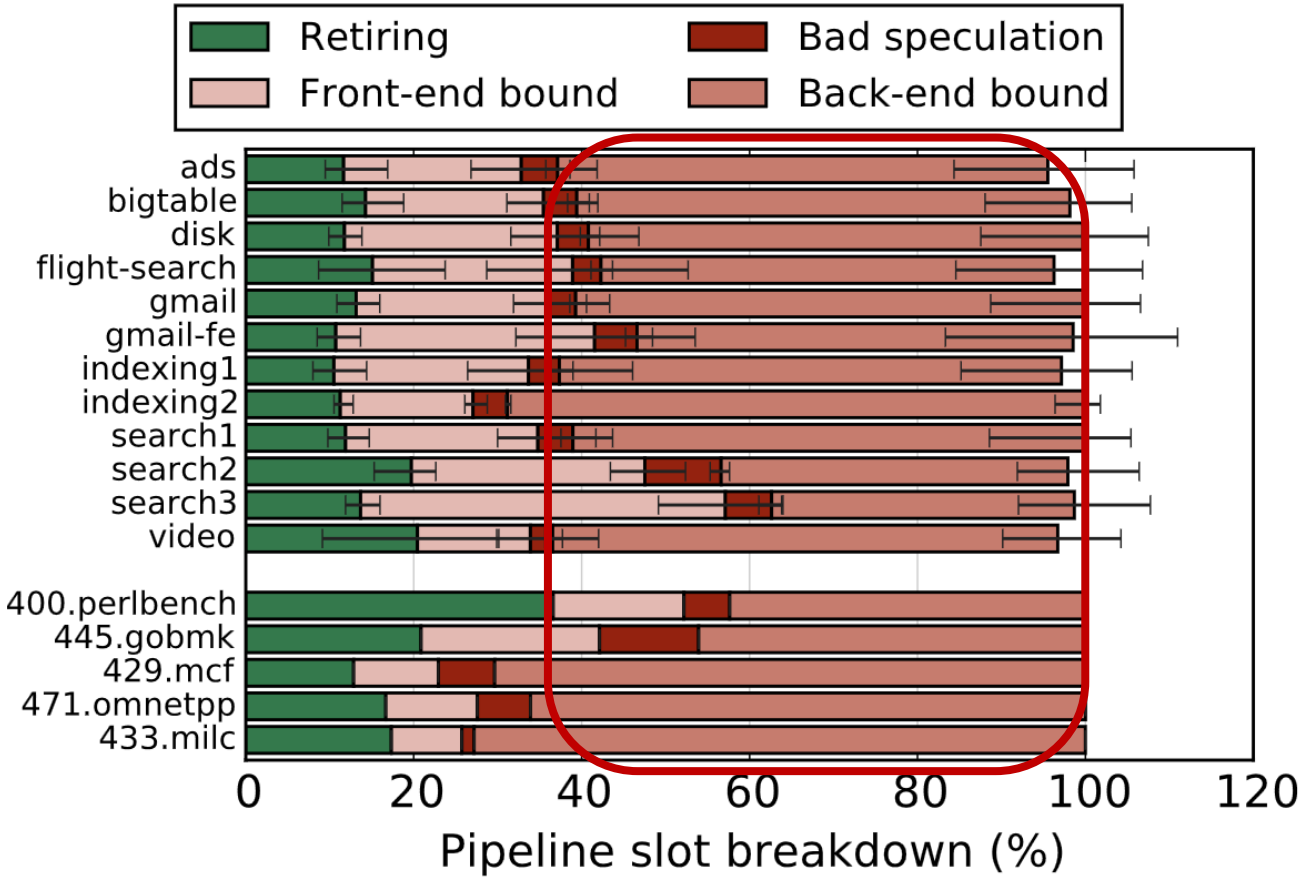
Zecheng Li[1,2], Xu Liu[1], Namhyung Kim[1], Blake Jones[1], Alexey Alexandrov[1], Jiajia Li[2]

[1]Google, [2]North Carolina State University





Motivation: Memory Stalls



40–60% of data-center CPU cycles stall on memory.

Figure from Kanev et al, ‘Profiling a warehouse-scale computer’, ISCA ’15.

Motivation: Memory Layout Optimization

```
- unsigned int  clock_update_flags;
- u64           clock;
+ /*The following fields of clock data are frequently
+  *referenced and updated together, and should go
+  *on their own cache line.*/
  u64           clock_task ____cacheline_aligned;
  u64           clock_pelt;
+ u64           clock;
  unsigned long lost_idle_time;
+ unsigned int  clock_update_flags;
  u64           clock_pelt_idle;
  u64           clock_idle;
```

Listing 1: The part of patch for reordering the fields of `rq`.

Field reorder for a struct improves kernel IPC by 5.1% on a microbenchmark.

Motivation: Memory Layout Optimization

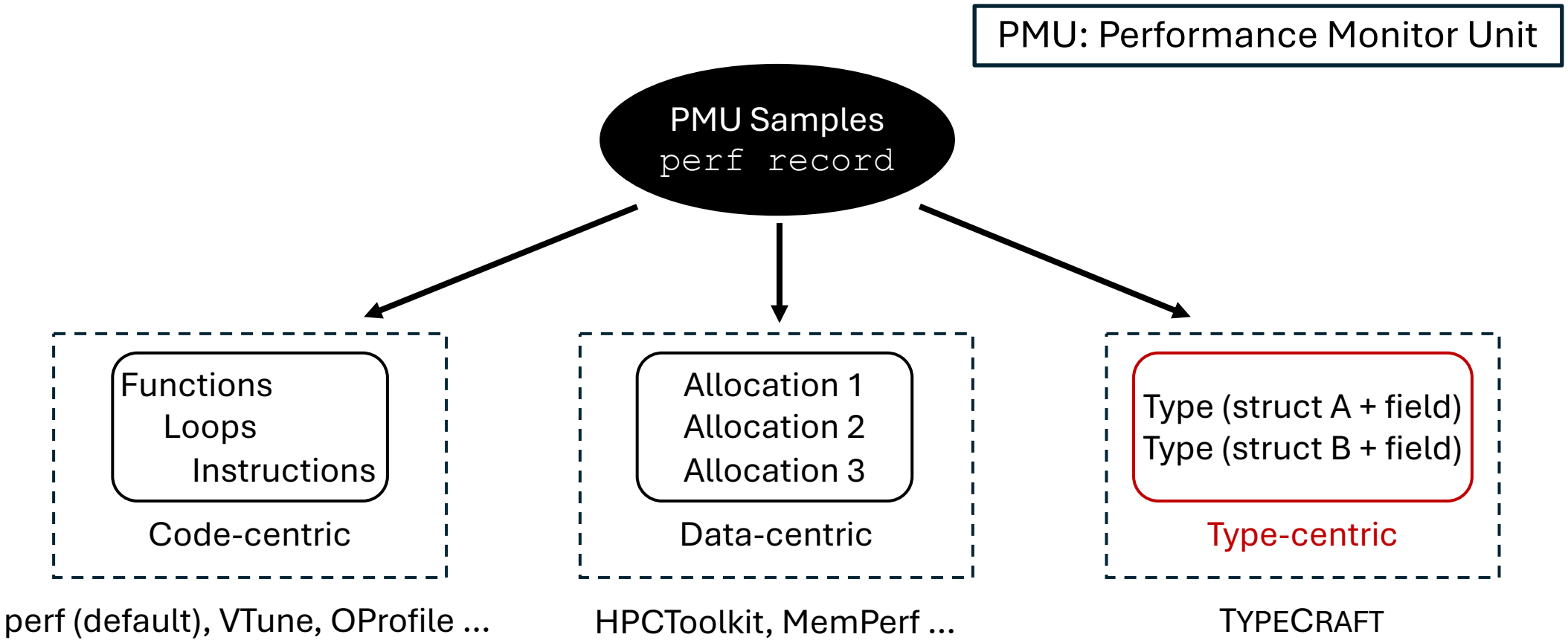
```
- unsigned int  clock_update_flags;
- u64           clock;
+ /*The following fields of clock data are frequently
+  *referenced and updated together, and should go
+  *on their own cache line.*/
  u64           clock_task ____cacheline_aligned;
  u64           clock_pelt;
+ u64           clock;
  unsigned long lost_idle_time;
+ unsigned int  clock_update_flags;
  u64           clock_pelt_idle;
  u64           clock_idle;
```

Listing 1: The part of patch for reordering the fields of `rq`.

Field reorder for a struct improves kernel IPC by 5.1% on a microbenchmark.

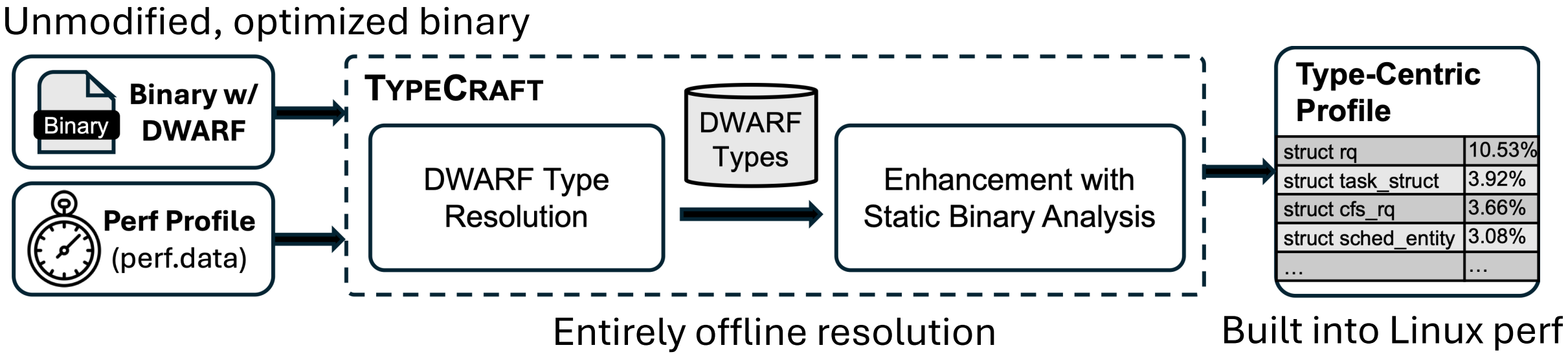
How can we find such optimizations at data-center scale?

Profiler Taxonomy



TYPECRAFT requires only the **binary** and its **DWARF** to attribute types.

TYPECRAFT Overview



TYPECRAFT attributes every x86 **memory-access instruction** in C binaries to a data type and field and reports a **type-centric profile**.

Interface

Metric grouped by type + field

Overhead	Data type / field
10.93%	(non-memory)
1.82%	(unknown memory type)
1.75%	(stack operation)
0.58%	struct qspinlock +0 val.counter
0.46%	long unsigned int
0.40%	struct cfs_rq +0x1a4 throttle_count
0.31%	struct file +0xb0 f_ref.refcnt.counter
0.31%	struct cfs_rq +0x58 curr
0.30%	int
0.22%	struct sched_entity +0xe0 avg.load_avg
0.18%	struct cfs_rq +0x138 rq
0.16%	unsigned int
0.16%	struct sched_entity +0x50 on_rq

Interface

Per-instruction view

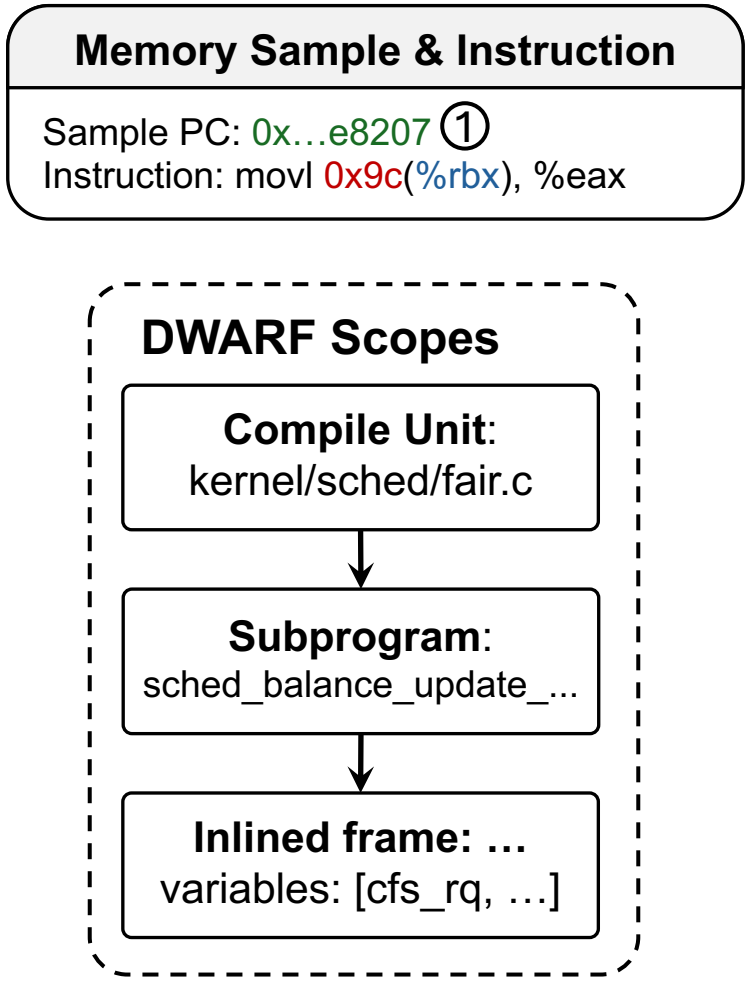
Overhead		Instruction		Data type / field
0.53	f0:	movl	0x1a4(%rbx),%r10d	struct cfs_rq +0x1a4 throttle_count
		movq	0x188(%rbx),%rsi	struct cfs_rq +0x188 throttled_clock_pelt_time
0.63		testl	%r10d,%r10d	
		↓ jne	5d7	
2.88		movq	0x138(%rbx),%rdx	struct cfs_rq +0x138 rq
1.39		cmpl	$1,0xb30(%rdx)	struct rq +0xb30 clock_update_flags
0.04		↓ jbe	5e6	
	11b:	movq	0xb48(%rdx),%rax	struct rq +0xb48 clock_pelt
0.61		subq	%rsi,%rax	
0.09		subq	0xb50(%rdx),%rax	struct rq +0xb50 lost_idle_time
2.35	12c:	movl	0xc4(%rbx),%r9d	struct cfs_rq +0xc4 removed.nr
		testl	%r9d,%r9d	
		↓ jne	26c	
0.83		movq	%rbx,%rsi	
0.02		movq	%rax,%rdi	

PC → Type mapping via DWARF

Memory Sample & Instruction

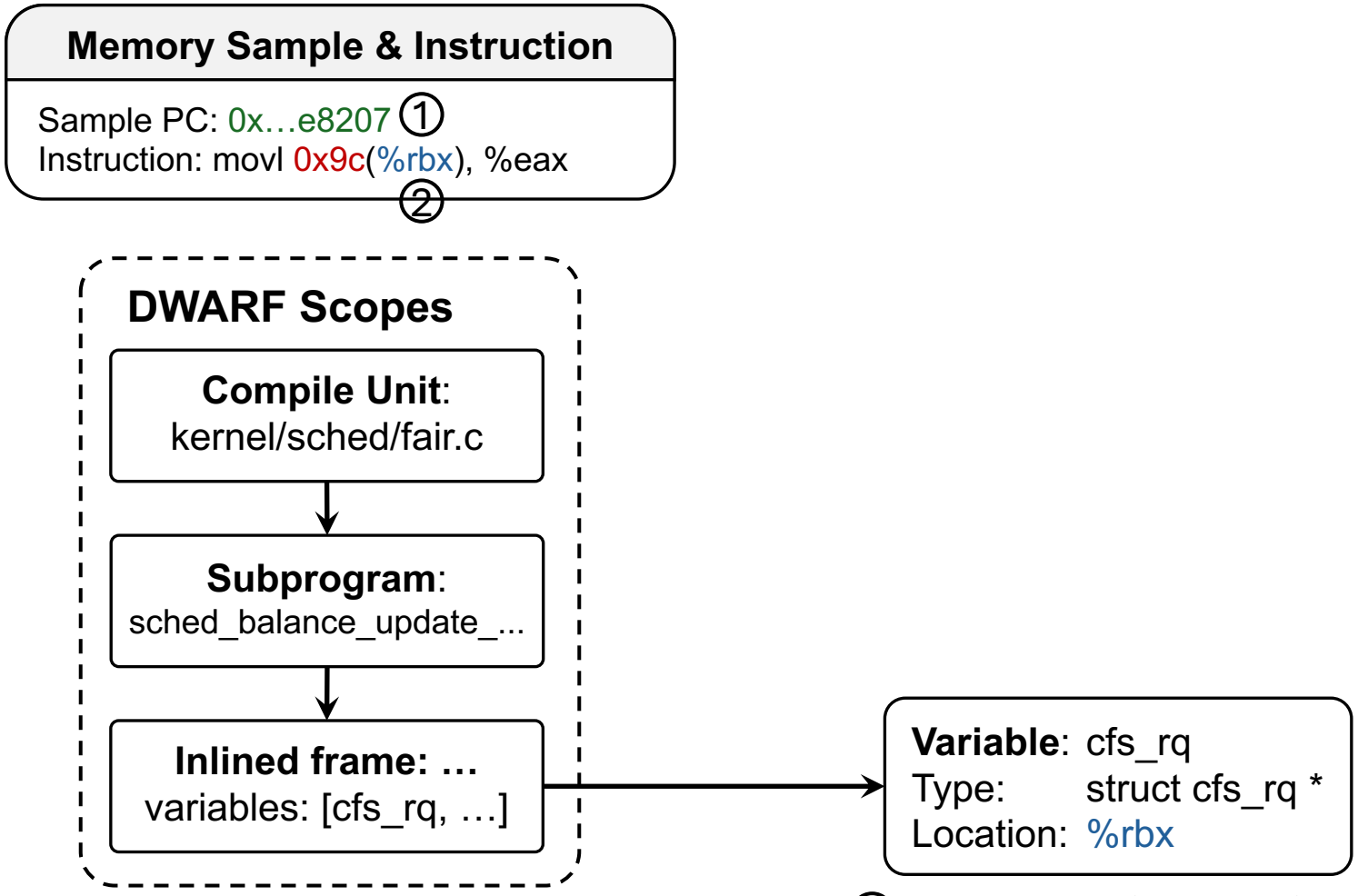
Sample PC: 0x…e8207
Instruction: movl 0x9c(%rbx), %eax

PC → Type mapping via DWARF

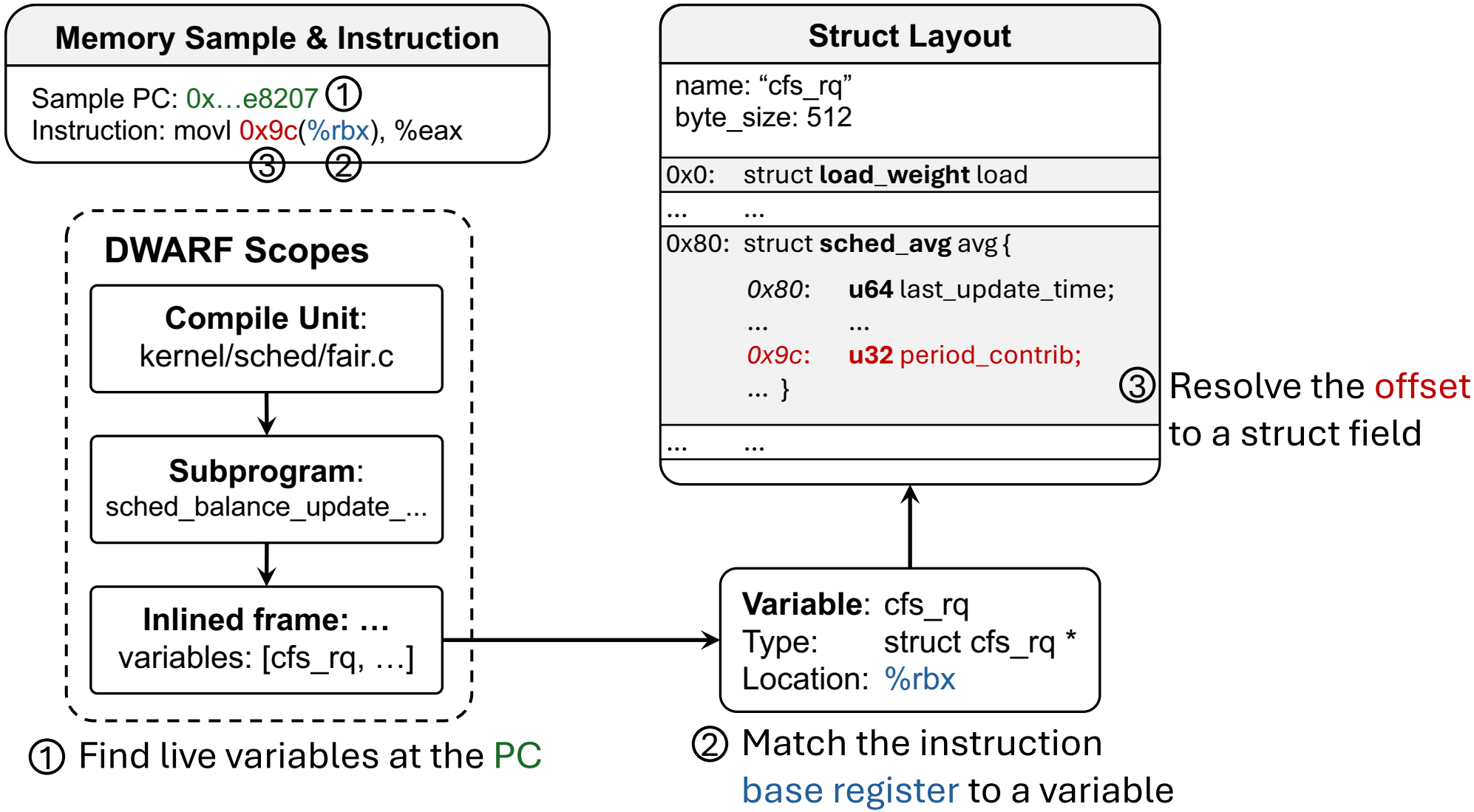


① Find live variables at the PC

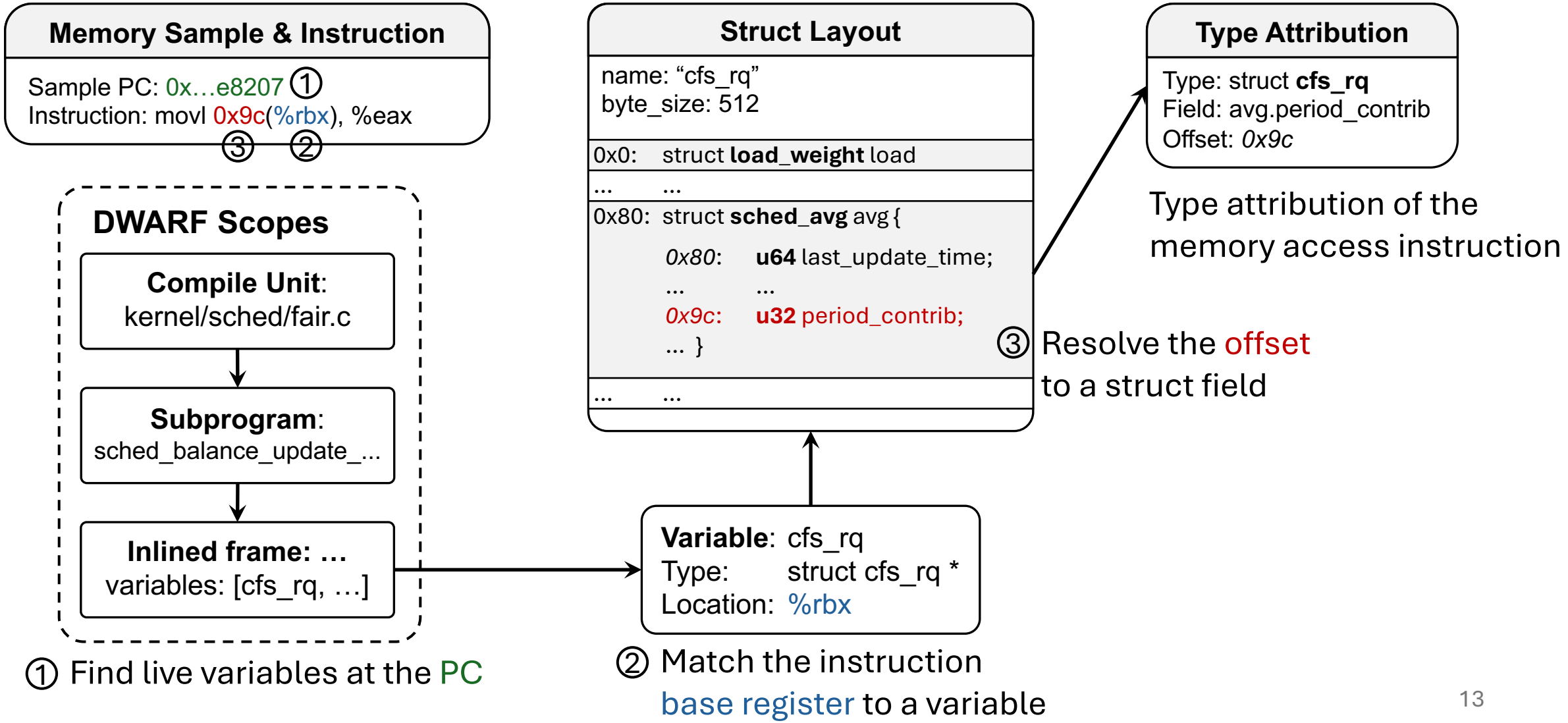
PC → Type mapping via DWARF



PC → Type mapping via DWARF



PC → Type mapping via DWARF



DWARF Limitations

- DWARF cannot resolve **pointer chasing patterns** with a chain of pointer dereferences because of missing variable information. Recent LLVM work adds optional support [1].

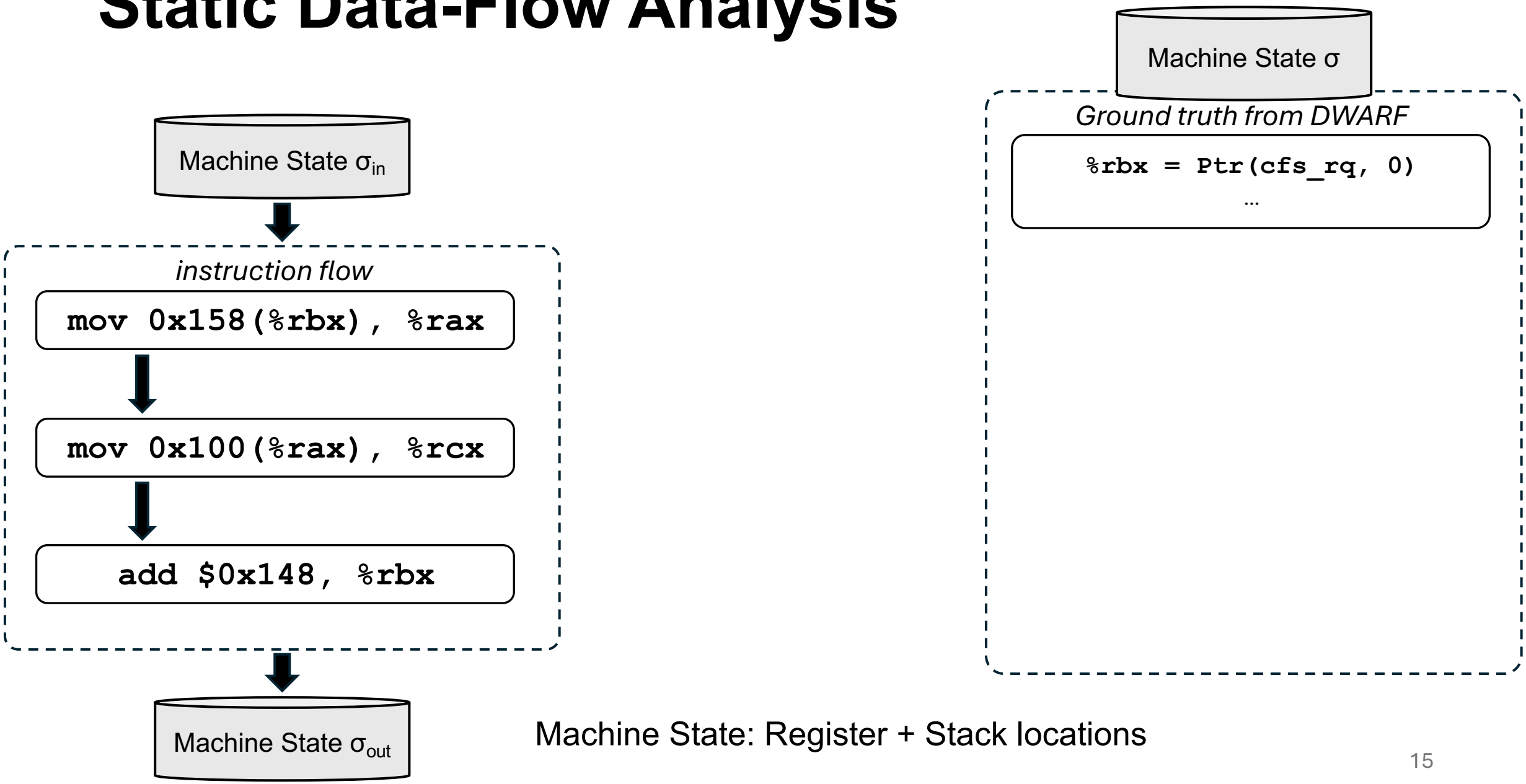
```
int i = foo->bar->baz
```

- DWARF quality also **degrades** in **heavily optimized code** under AutoFDO, LTO, and post-link optimizations such as BOLT and Propeller.

DWARF-only coverage: 75% on Ubuntu's kernel, 66% on CachyOS with ThinLTO

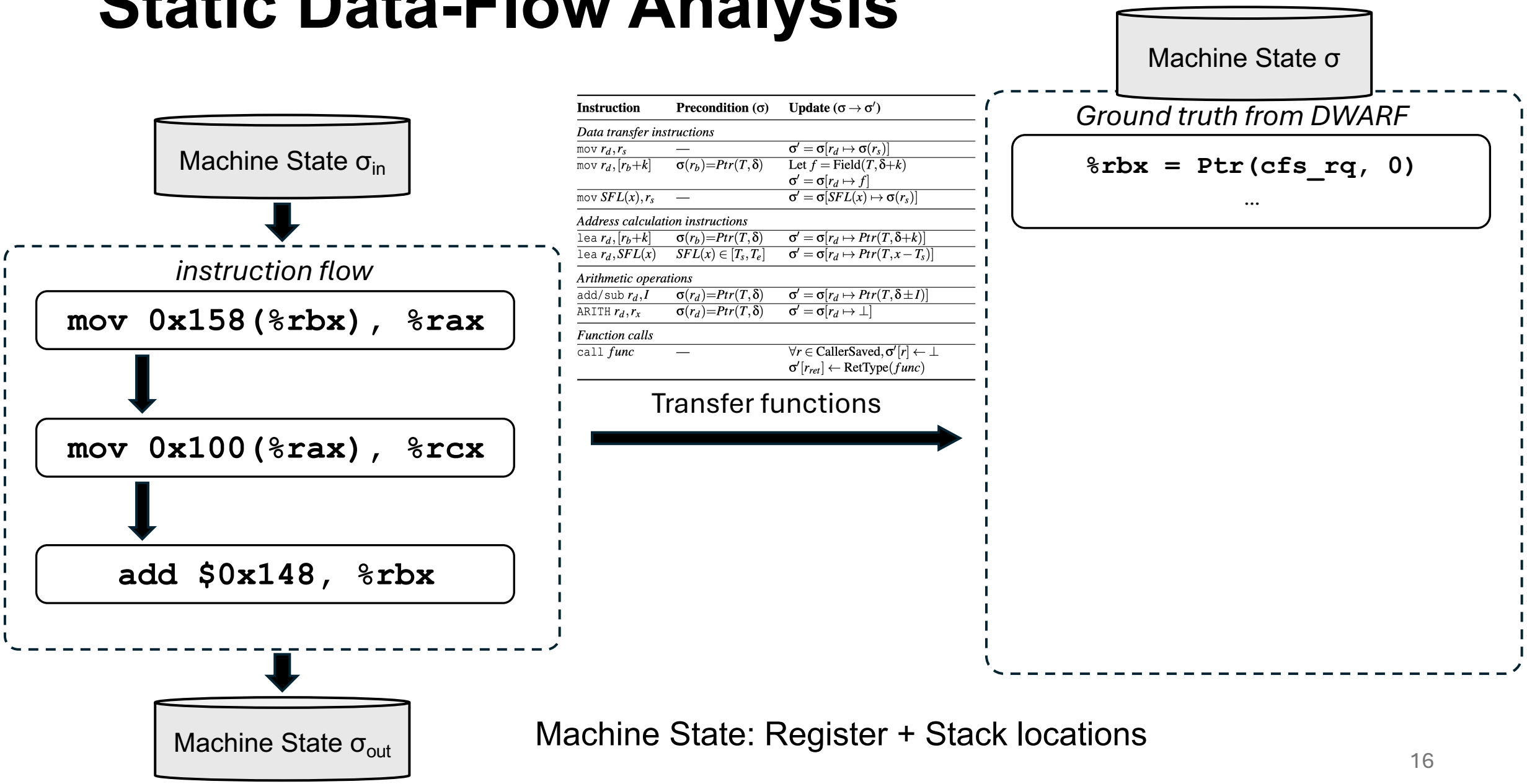
[1] William Junda Huang, Add option to generate additional debug info for expression dereferencing pointer to pointers, https://github.com/llvm/llvm-project/pull/95298

Static Data-Flow Analysis



Machine State: Register + Stack locations

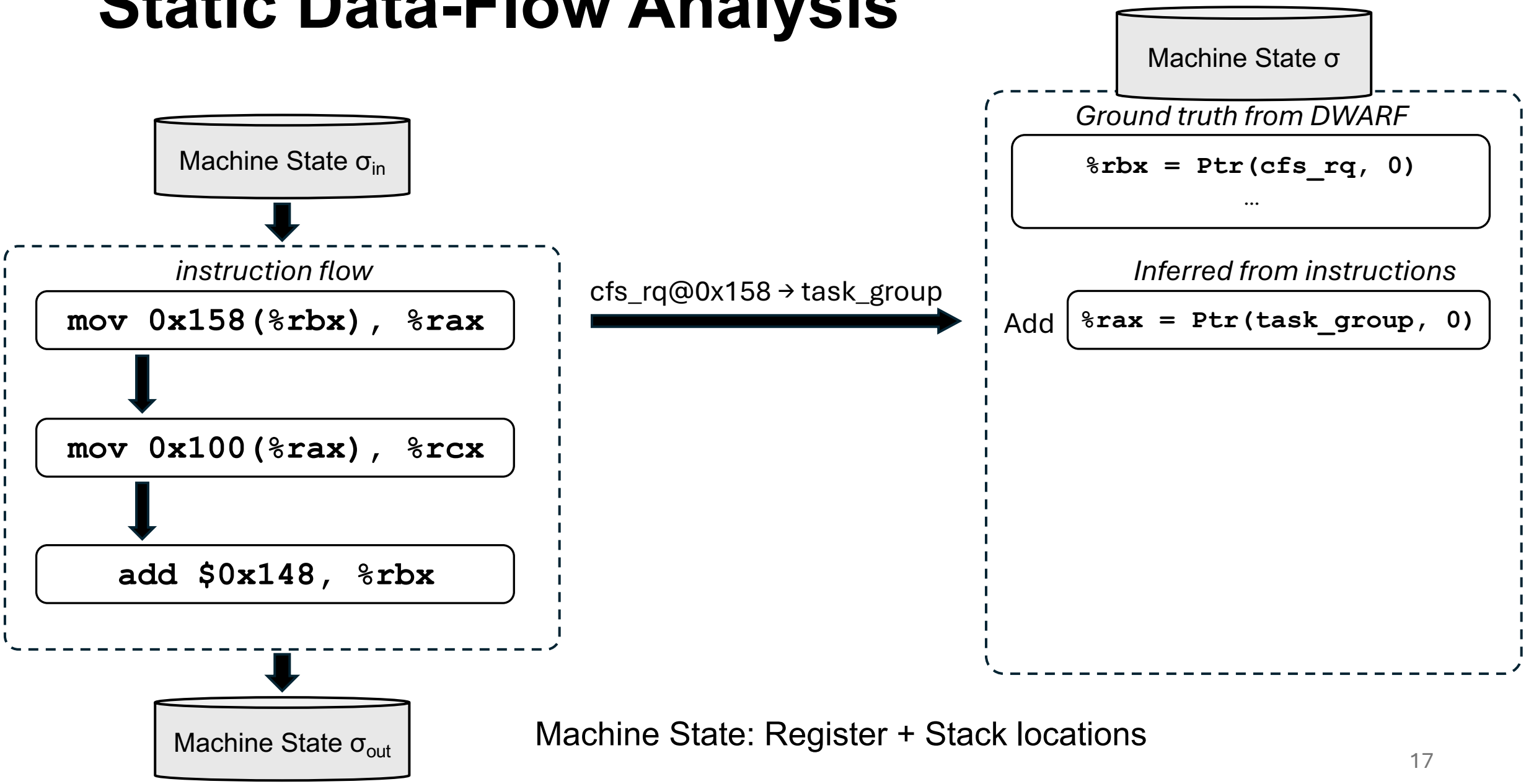
Static Data-Flow Analysis



Instruction	Precondition (σ)	Update ($\sigma \to \sigma'$)
Data transfer instructions		
`mov` r_d, r_s	—	$\sigma' = \sigma[r_d \mapsto \sigma(r_s)]$
`mov` $r_d, [r_b{+}k]$	$\sigma(r_b){=}Ptr(T, \delta)$	Let $f = \text{Field}(T, \delta{+}k)$ $\sigma' = \sigma[r_d \mapsto f]$
`mov` $SFL(x), r_s$	—	$\sigma' = \sigma[SFL(x) \mapsto \sigma(r_s)]$
Address calculation instructions		
`lea` $r_d, [r_b{+}k]$	$\sigma(r_b){=}Ptr(T, \delta)$	$\sigma' = \sigma[r_d \mapsto Ptr(T, \delta{+}k)]$
`lea` $r_d, SFL(x)$	$SFL(x) \in [T_s, T_e]$	$\sigma' = \sigma[r_d \mapsto Ptr(T, x - T_s)]$
Arithmetic operations		
`add/sub` r_d, I	$\sigma(r_d){=}Ptr(T, \delta)$	$\sigma' = \sigma[r_d \mapsto Ptr(T, \delta \pm I)]$
`ARITH` r_d, r_x	$\sigma(r_d){=}Ptr(T, \delta)$	$\sigma' = \sigma[r_d \mapsto \bot]$
Function calls		
`call` *func*	—	$\forall r \in \text{CallerSaved}, \sigma'[r] \leftarrow \bot$ $\sigma'[r_{ret}] \leftarrow \text{RetType}(func)$

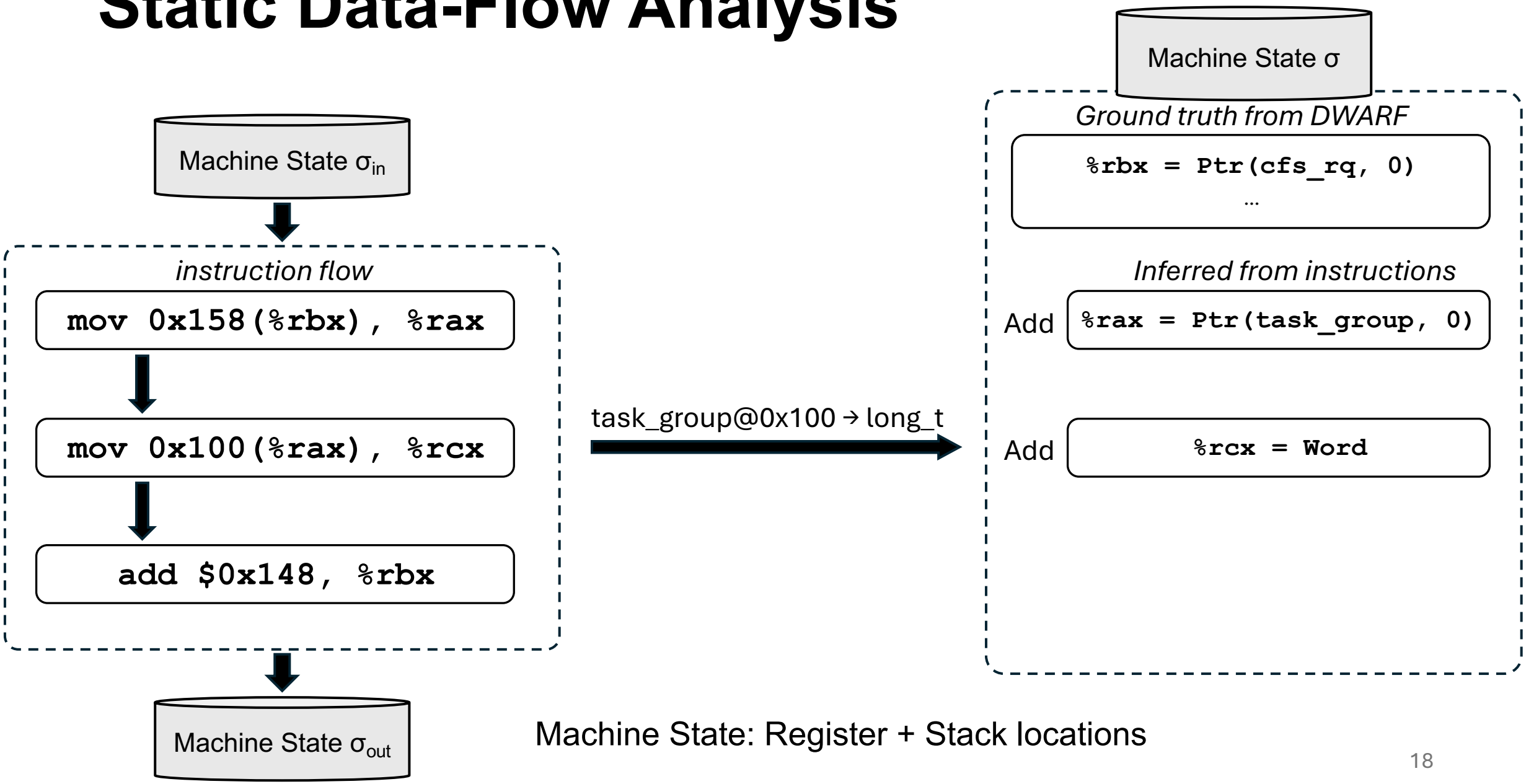
Machine State: Register + Stack locations

Static Data-Flow Analysis



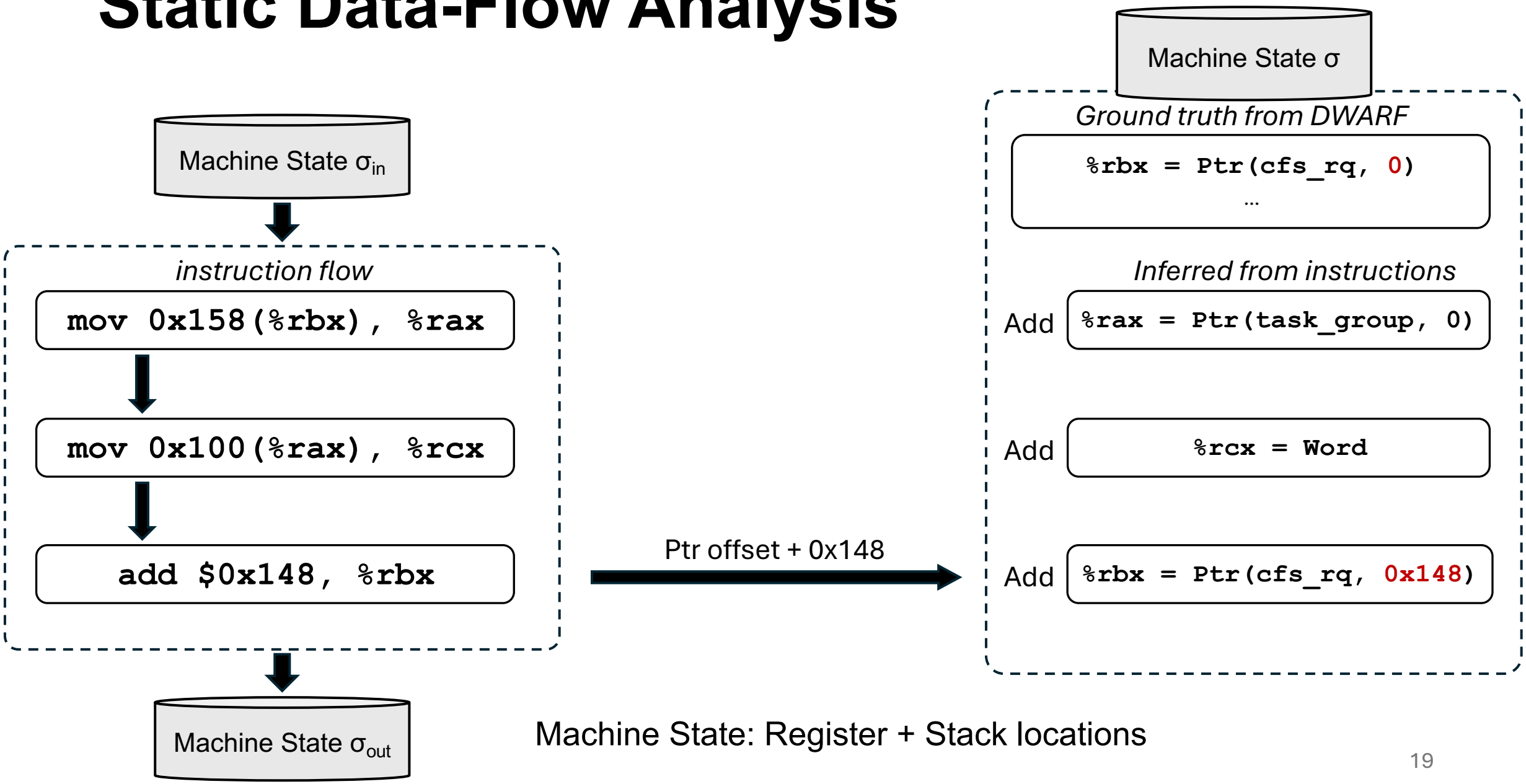
Machine State: Register + Stack locations

Static Data-Flow Analysis



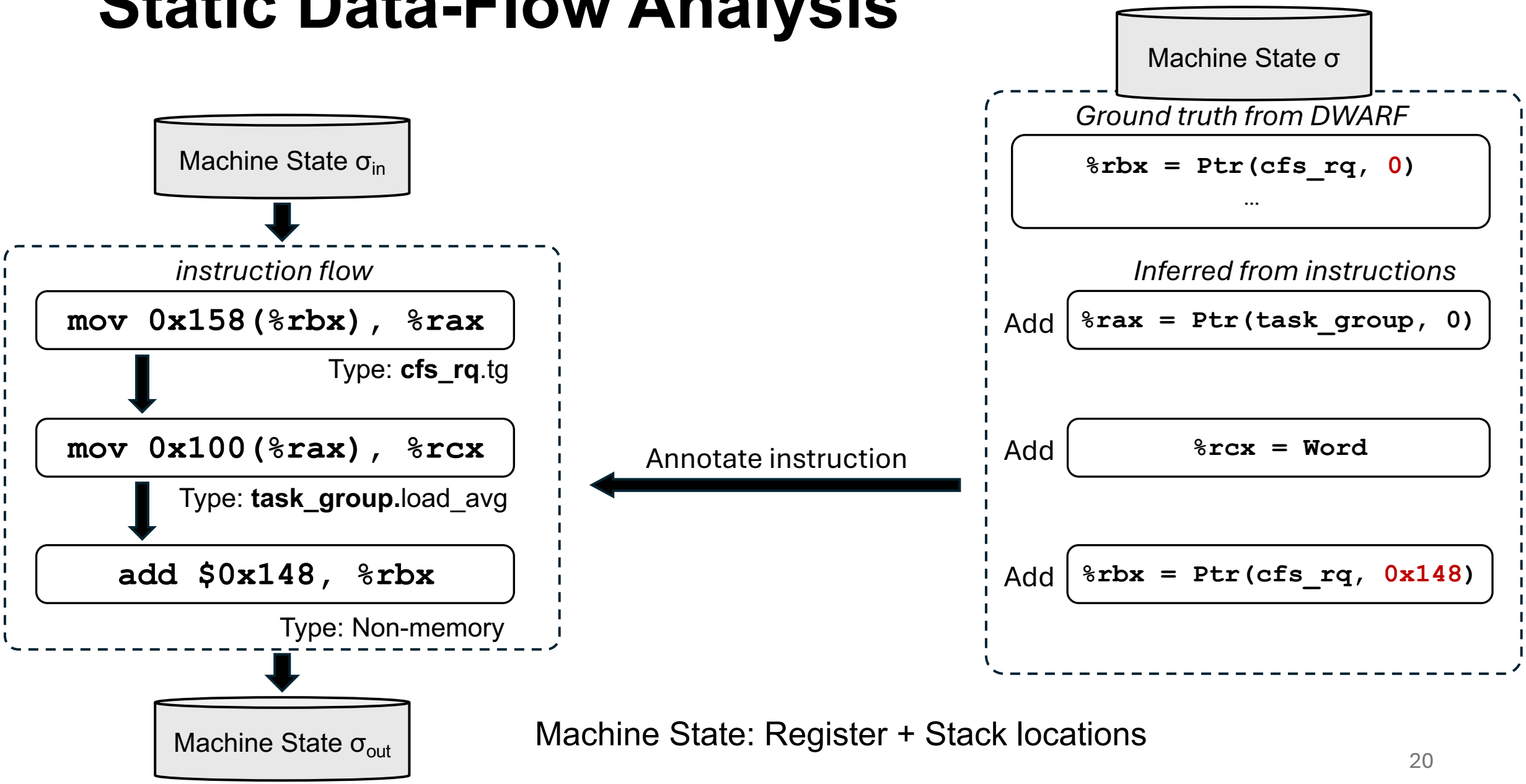
Machine State: Register + Stack locations

Static Data-Flow Analysis



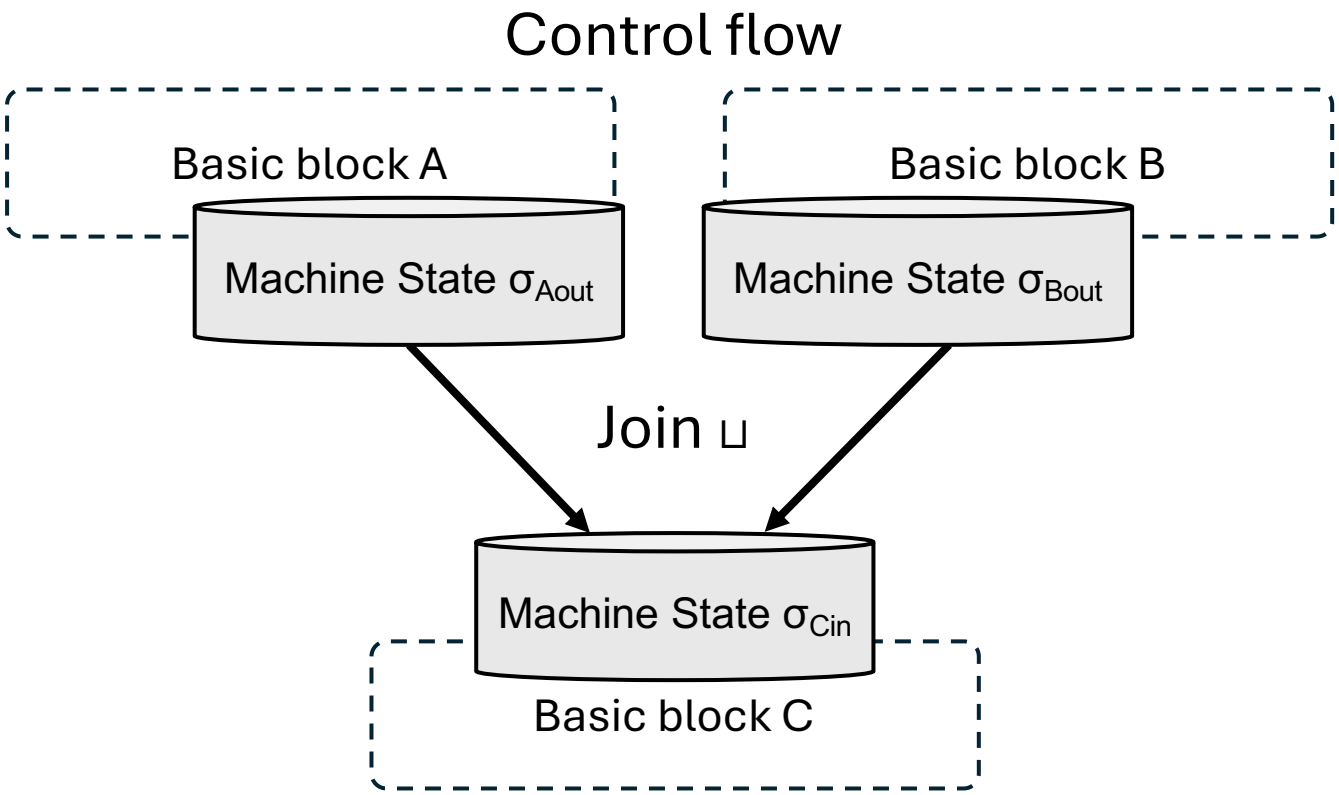
Machine State: Register + Stack locations

Static Data-Flow Analysis

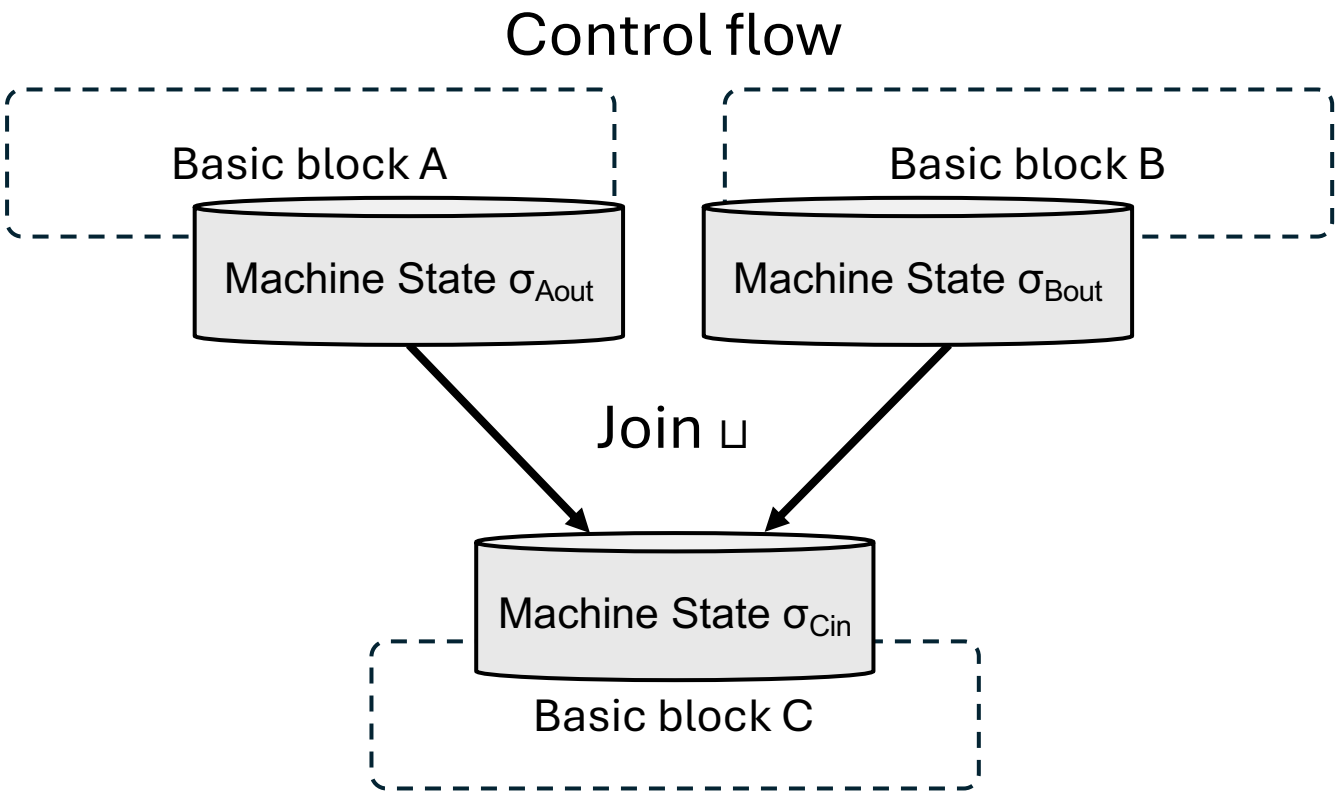


Machine State: Register + Stack locations

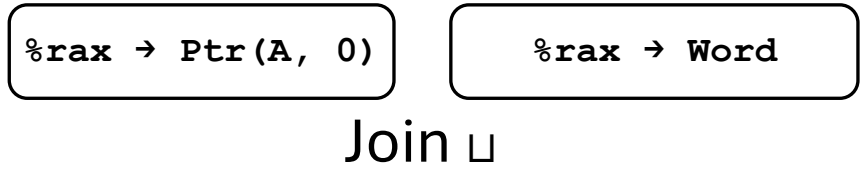
Static Data-Flow Analysis



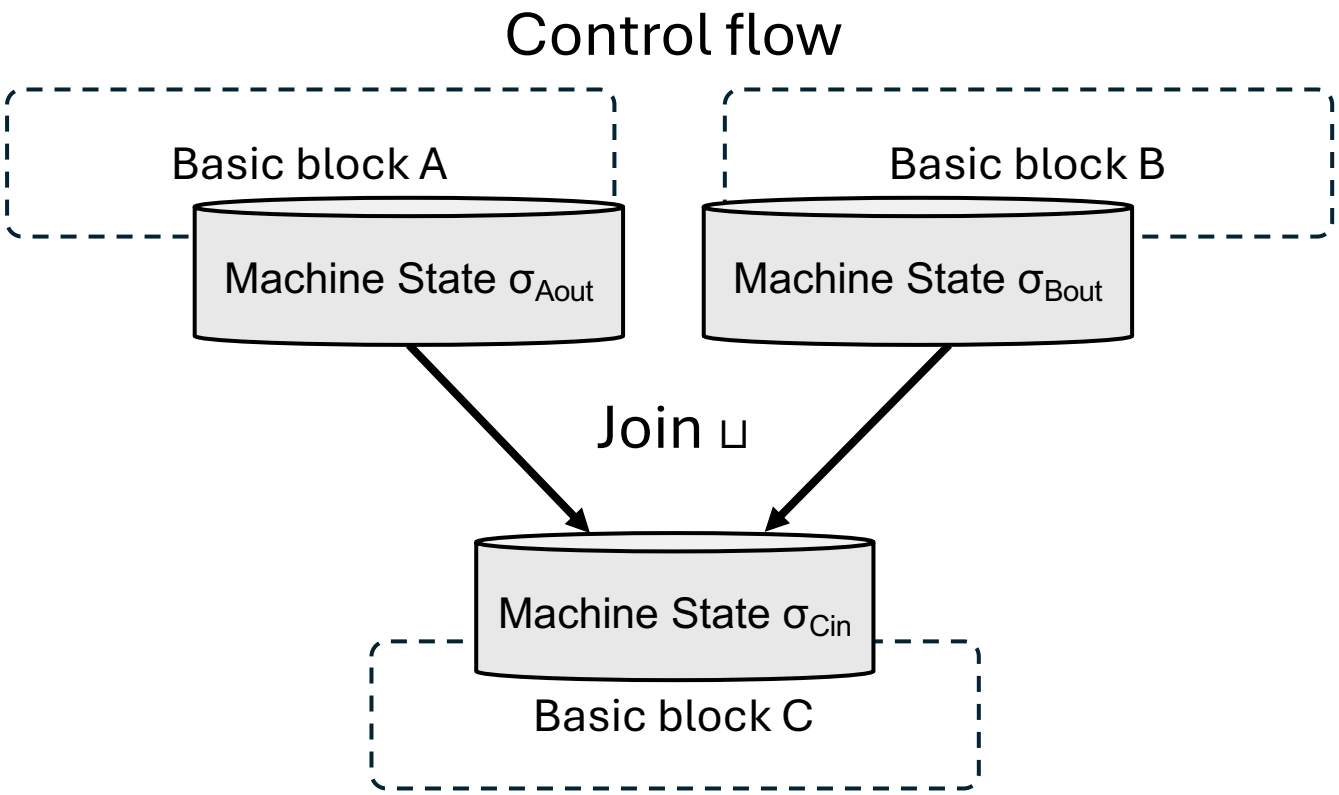
Static Data-Flow Analysis



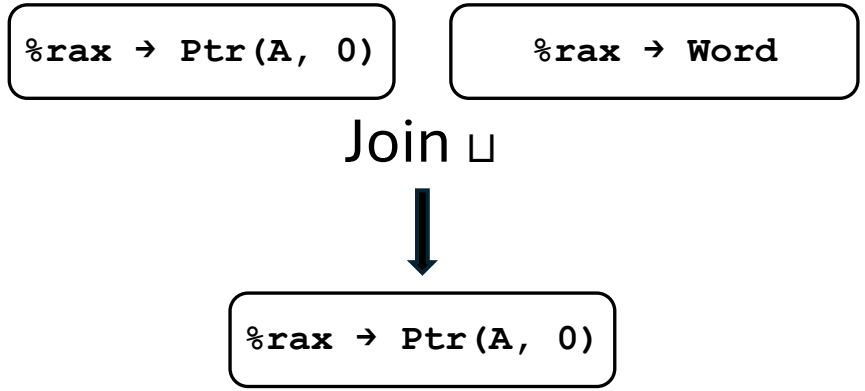
Optimistic promotion Word ⊑ Ptr



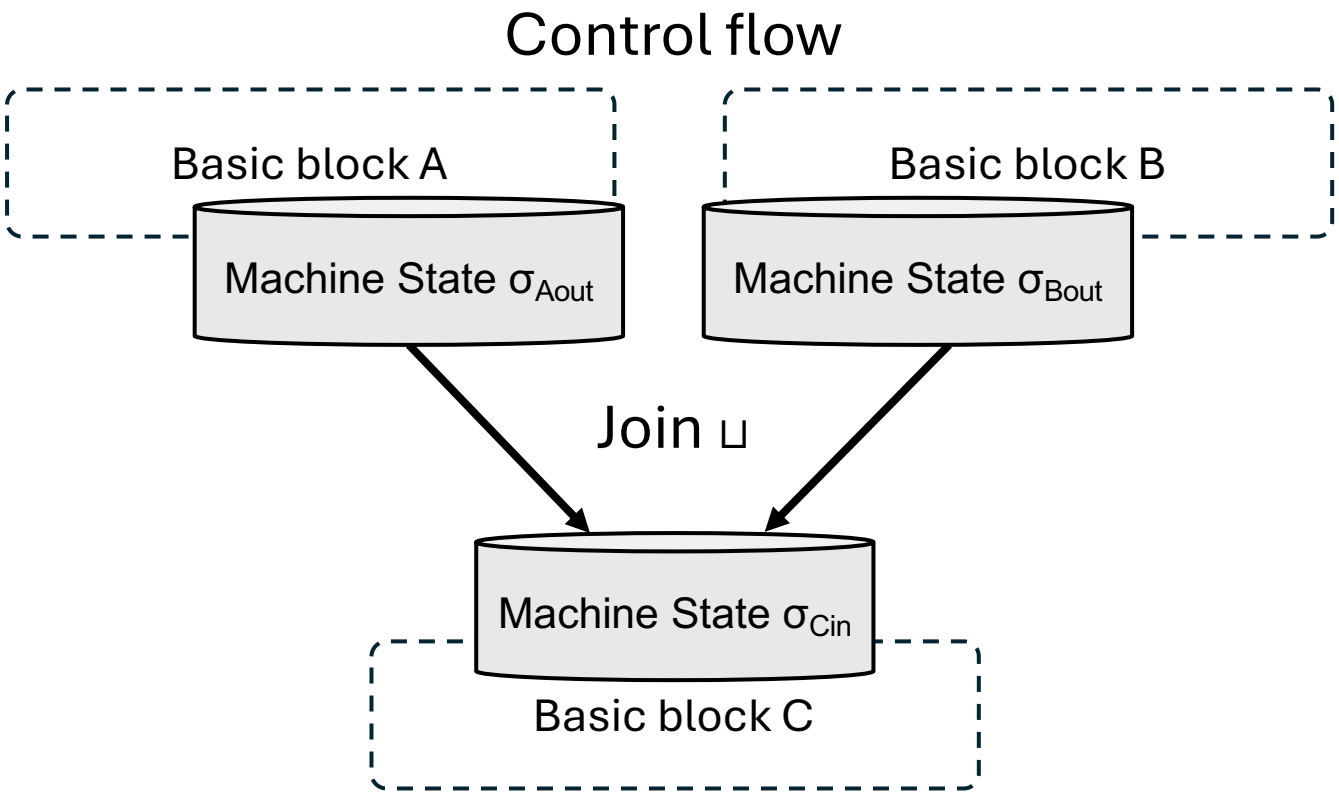
Static Data-Flow Analysis



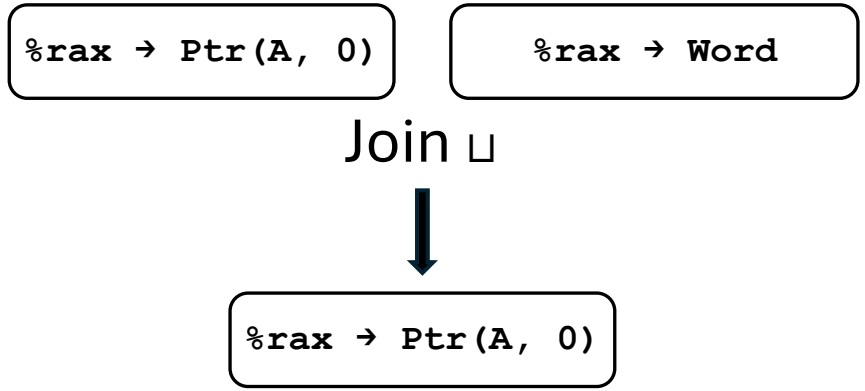
Optimistic promotion Word ⊑ Ptr



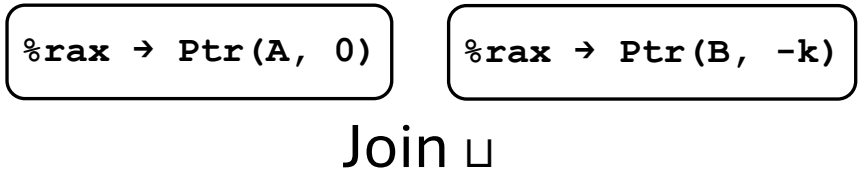
Static Data-Flow Analysis



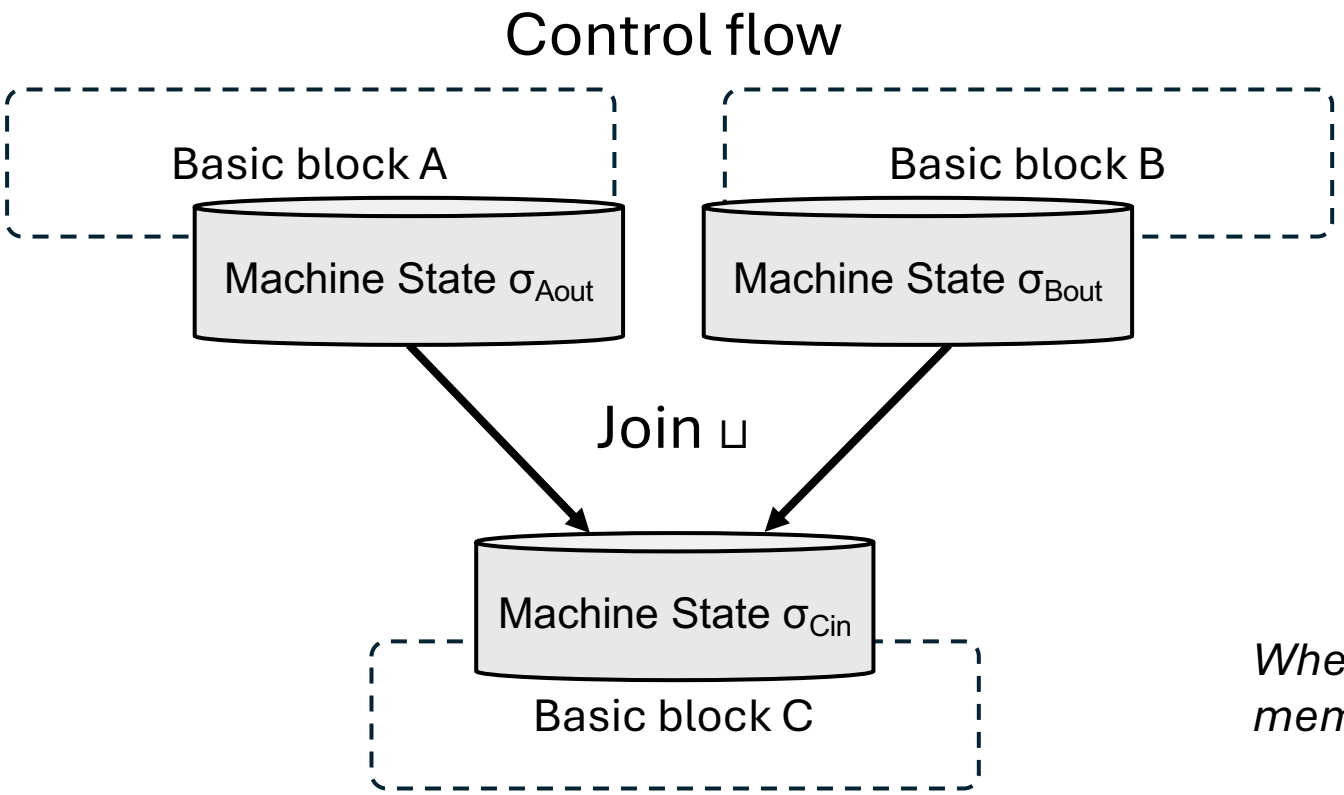
Optimistic promotion Word ⊑ Ptr



Type containment relations



Static Data-Flow Analysis



Optimistic promotion Word ⊑ Ptr

`%rax → Ptr(A, 0)` `%rax → Word`

Join ⊔

`%rax → Ptr(A, 0)`

Type containment relations

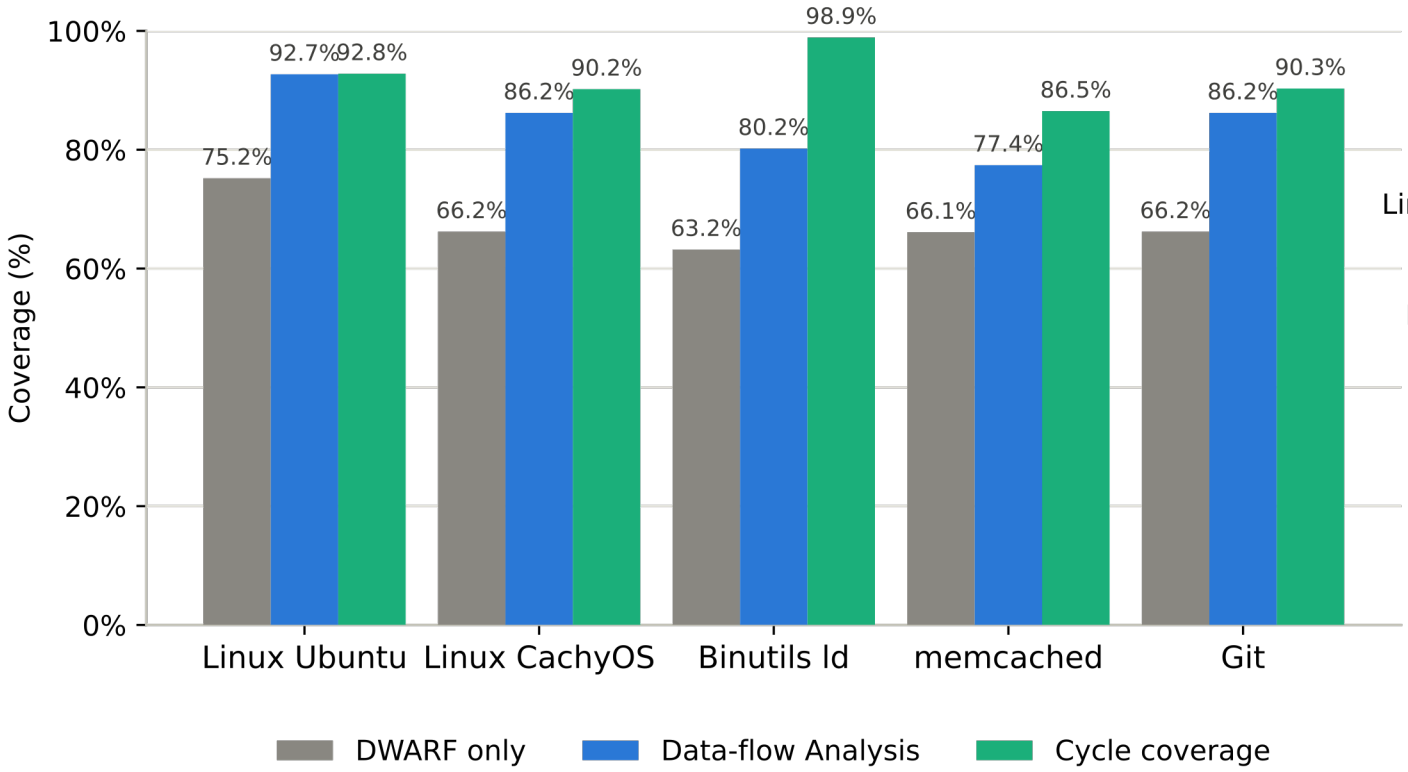
`%rax → Ptr(A, 0)` `%rax → Ptr(B, -k)`

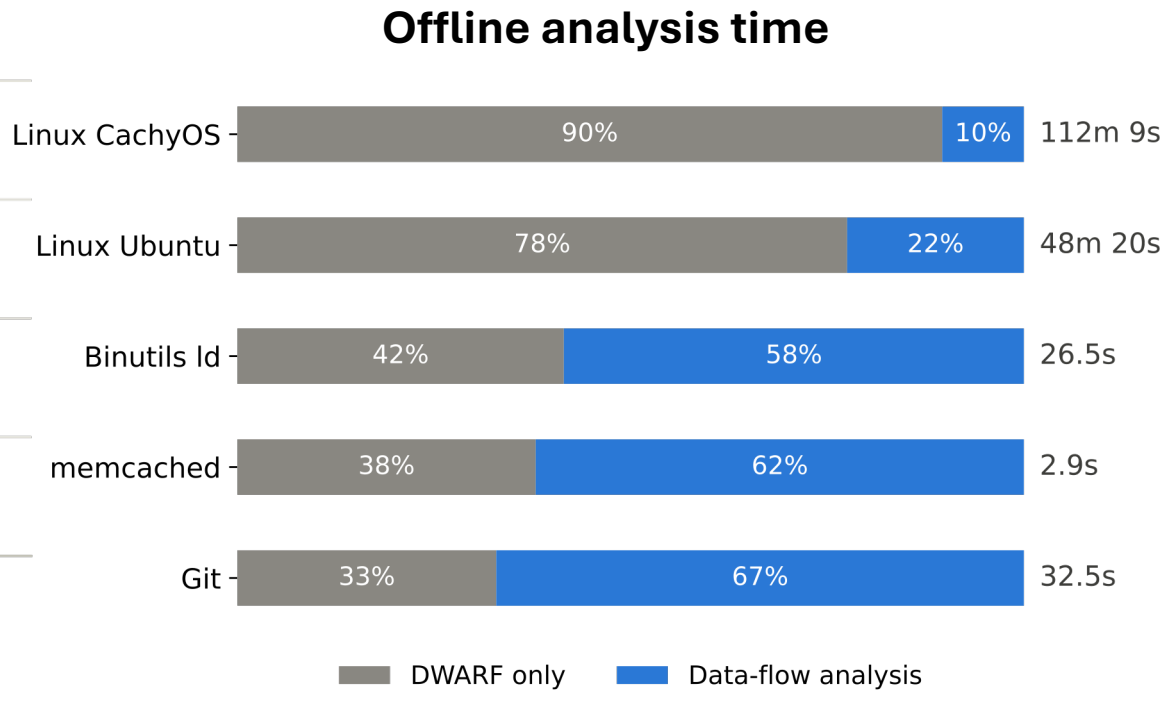
Join ⊔

When struct B is the type of a member of struct A at offset k

`%rax → Ptr(A, 0)`

Coverage & Overhead





Cycle-weighted coverage = 1 − (untyped memory cycles/all memory cycles), measured per workload under representative profiling runs.

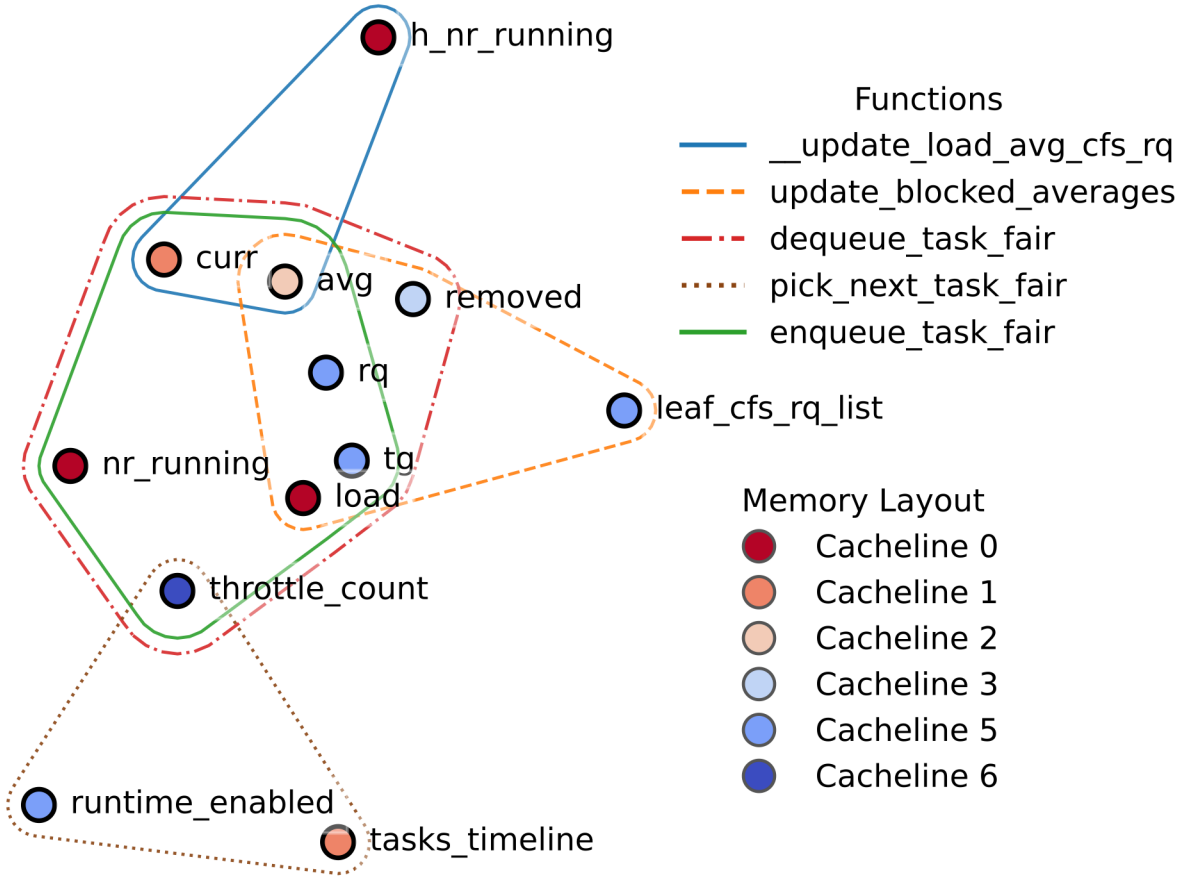
Case Study: Field Reordering

Data Type	Cycles %	LLC misses %
cfs_rq	7.58%	49.02%
sched_entity	4.73%	6.05%
rq	1.05%	0.35%
cfs_rq *	0.21%	0.86%
sched_entity *	0.16%	2.41%

Profile from a microbenchmark that stresses CFS in a cgroup hierarchy, Linux 6.17.

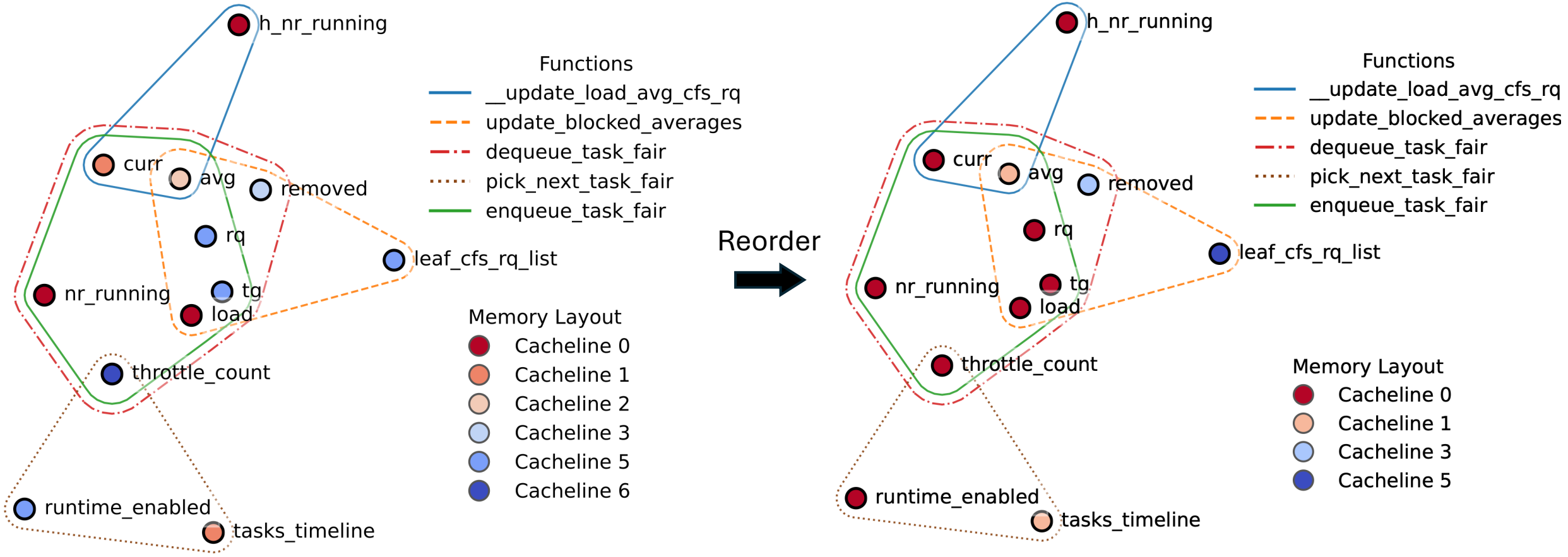
Case Study: Field Reordering

`struct cfs_rq`



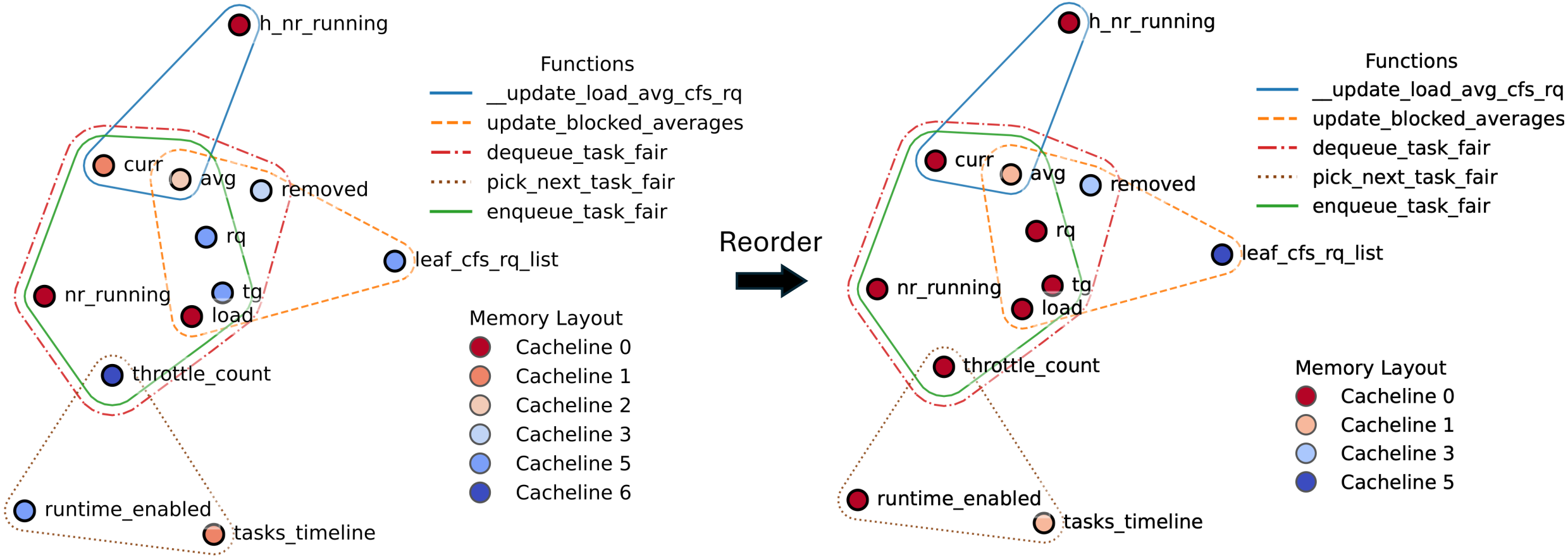
Case Study: Field Reordering

`struct cfs_rq`



Case Study: Field Reordering

`struct cfs_rq`



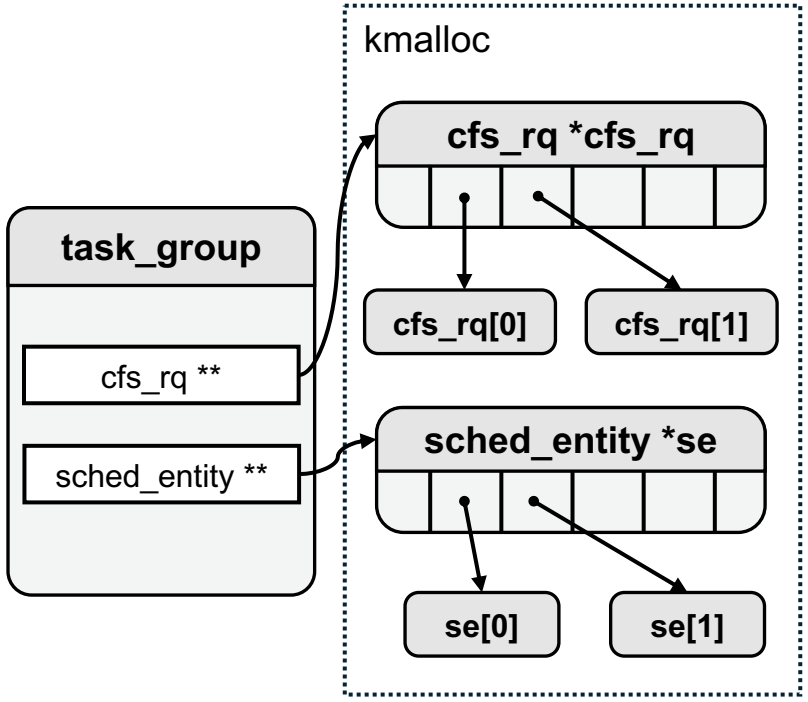
Result: 26.4% fewer kernel LLC misses.

Case Study: Field Reordering

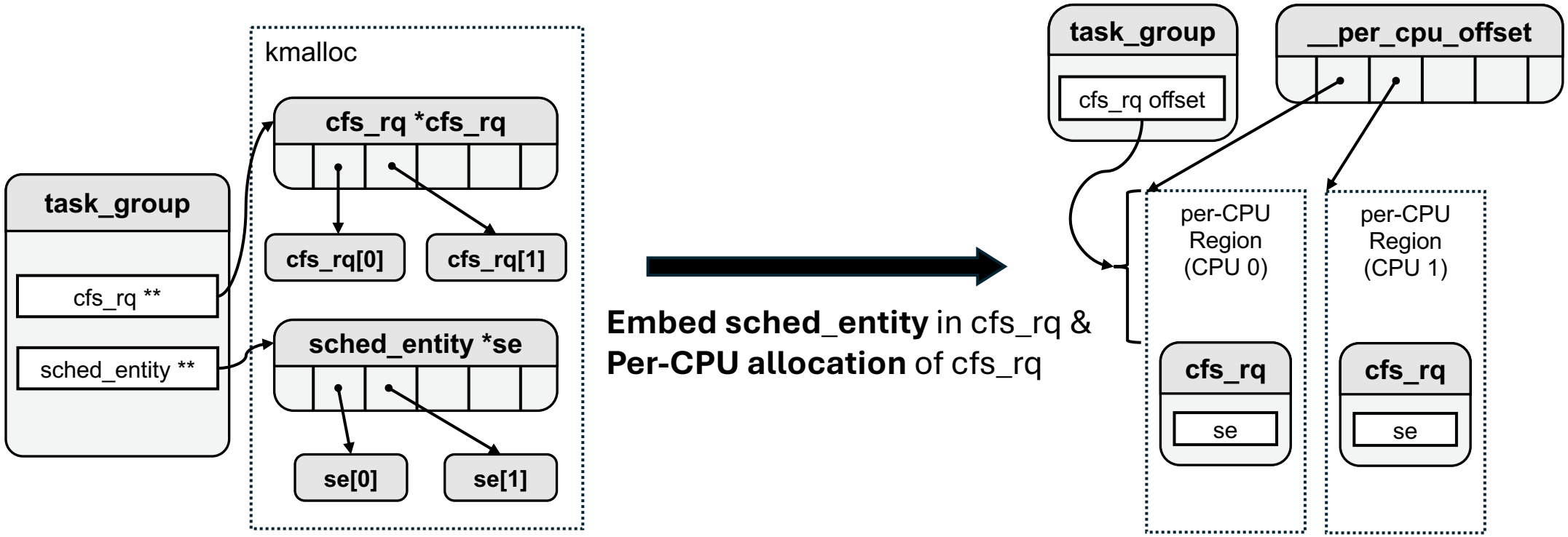
Data Type	Cycles %	LLC misses %
`cfs_rq`	7.58%	49.02%
`sched_entity`	4.73%	6.05%
`rq`	1.05%	0.35%
`cfs_rq *`	0.21%	0.86%
`sched_entity *`	0.16%	2.41%

Profile from a microbenchmark that stresses CFS in a cgroup hierarchy, Linux 6.17.

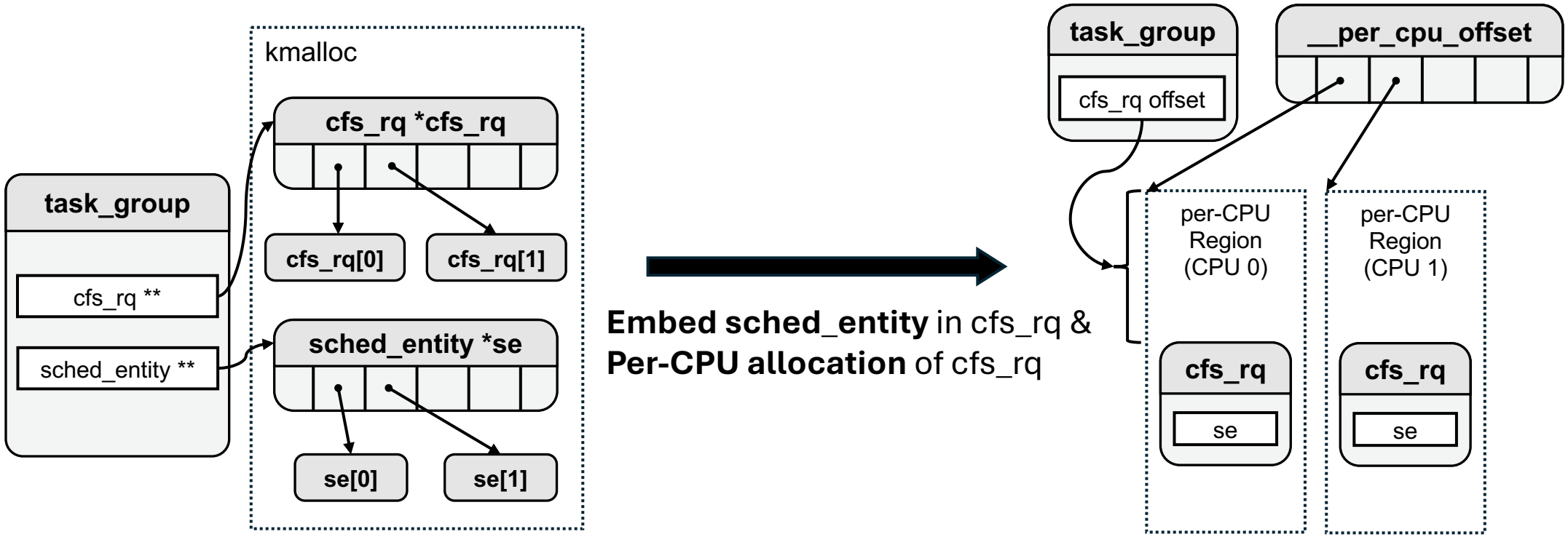
Case Study: Removing Indirection



Case Study: Removing Indirection



Case Study: Removing Indirection



Embedding caused a regression: the larger structure forced **1024-byte allocator alignment**, causing **cache conflict** misses. Per-CPU allocation fixed the regression.

Result: an additional 33% reduction in kernel LLC misses

Impact at Scale

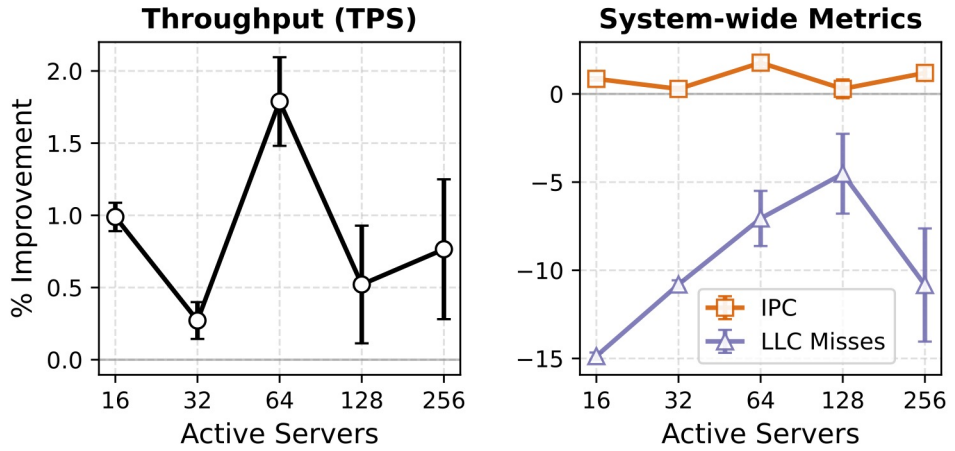
Microbenchmark:
schbench with a cgroup hierarchy
~50% fewer kernel LLC misses.

Includes both the field-reordering and indirection-removal patches.

Impact at Scale

Microbenchmark:
schbench with a cgroup hierarchy
~50% fewer kernel LLC misses.

MySQL OLTP:
Co-running MySQL servers
Up to 1.8% higher TPS



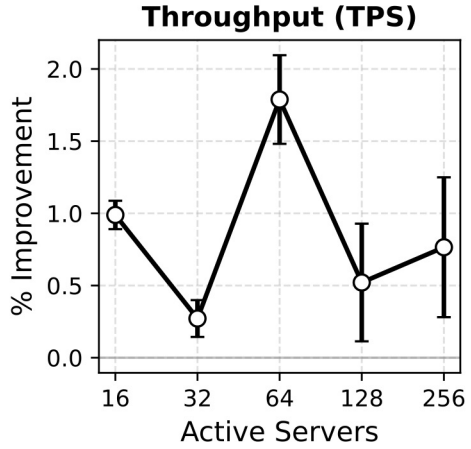
Includes both the field-reordering and indirection-removal patches.

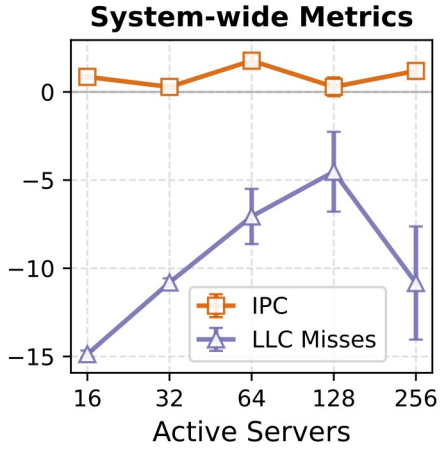
Impact at Scale

Microbenchmark:
schbench with a cgroup hierarchy
~50% fewer kernel LLC misses.

MySQL OLTP:
Co-running MySQL servers
Up to 1.8% higher TPS

Google workloads:
0.5% higher CPU throughput
and 0.5% lower memory use.





Internal metrics indicate a **0.5±0.02% increase** in CPU throughput alongside a **0.5±0.02% reduction** in RAM use.

Includes both the field-reordering and indirection-removal patches.

Open-source contributions

```
Namhyung Kim (17):
  perf dwarf-aux: Factor ou
  perf dwarf-regs: Add get_
  perf annotate-data: Add f
  perf annotate-data: Add d
  perf annotate: Factor out
  perf annotate: Add annota
  perf annotate: Implement
  perf report: Add 'type' s
  perf report: Support data
  perf annotate-data: Add m
  perf annotate-data: Updat
  perf report: Add 'typeoff
  perf report: Add 'symoff'
Namhyung Kim (7):
  perf annotate-data: Add ar
  perf annotate: Remove unus
  perf annotate: Pass annota
  perf annotate: Pass hist_e
Zecheng Li (5):
      perf annotate: Skip ar
      perf annotate: Track a
      perf annotate: Track a
      perf annotate: Save pc
      perf annotate: Invalic
```

```
Namhyung Kim (4):
  perf dwarf-aux: Handle bitfield members from pointer ac
  perf annotate-data: Update debug messages
  perf annotate-data: Update stack slot for the store
  perf annotate-data: Copy back variable types after move
Namhyung Kim (6):
  perf hist: Correct hist_entry->mem_info refcounts
  perf mem: Free the allocated sort string
  perf mem: Rework command option handling
  perf tools: Add mode argument to sort_help()
  perf mem: Add -s/--sort option
Zecheng Li (11):
      perf dwarf-aux: Add die_get_pointer_type to get
      perf dwarf-aux: Preserve typedefs in match_var_o
      perf dwarf-aux: Skip check_variable for variable
      perf annotate-data: Improve type comparison from
      perf dwarf-aux: Handle array types in die_get_me
      perf annotate-data: Collect global variables wit
      perf annotate-data: Handle global variable acc
Zecheng Li (5):
      perf dwarf-aux: Use signed variable types in
      perf dwarf-aux: More accurate variable type m
      perf dwarf-aux: Better variable collection fo
      perf dwarf-aux: Fix __die_find_scope_cb() for
      perf annotate: Rename TSR_KIND_POINTER to TSF
```

TYPECRAFT is integrated into Linux perf

```
Subject: [GIT PULL] Scheduler changes for v7.0
Date: Mon, 9 Feb 2026 08:46:16 +0100    [thread overview]
Message-ID: <aYmQyDHFx3lisCgm@gmail.com> (raw)

  - Reorder fields in 'struct rq' for better caching
    (Blake Jones)
```

The struct rq reorder optimization was merged into Linux 7.0.

```
Subject: [GIT PULL] Scheduler updates for v7.2
Date: Sun, 14 Jun 2026 17:48:41 +0200   [thread overview]
Message-ID: <ai7NWao6iQcUmRZg@gmail.com> (raw)

- A series to optimize cfs_rq and sched_entity allocation for
  better data locality (Zecheng Li)
```

The indirection-removal optimization was merged into Linux 7.2.

Takeaways

(1) Type/field attribution for **individual memory instructions** is achievable from optimized binaries with DWARF and data-flow analysis;

(2) All type resolution runs **offline**, so continuous data-center profiling adds **zero online overhead**.

(3) TYPECRAFT finds actionable optimizations validated from **microbenchmarks** through **Google's fleet**.

(4) TYPECRAFT is integrated into **Linux perf**, and the kernel optimizations have **landed in mainline Linux**.

Acknowledgments

We thank Namhyung Kim for the initial idea of data-type profiling, for developing the first perf-based version of TYPECRAFT [1], and for reviewing the upstream patch series. We also thank him for community work to bring TYPECRAFT to Arm and PowerPC.

We also thank our shepherd and all reviewers for their valuable comments.

This work was supported by Google and the National Science Foundation under Grant CISE-2316201.

[1] Namhyung Kim, perf tools: Introduce data type profiling (v3), https://lwn.net/Articles/954938/

Q & A