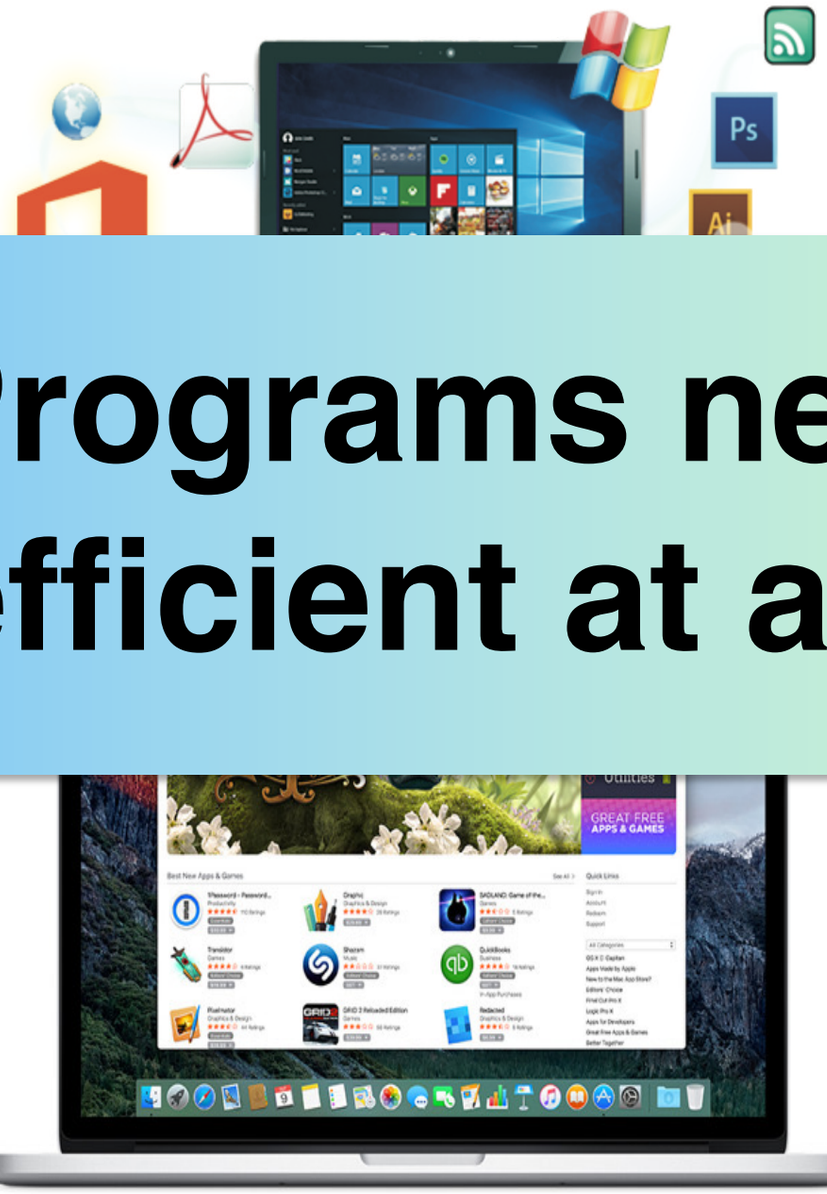
Pinpointing Program Inefficiencies with DrCCTProf Clients -- LoadSpy

Pengfei Su
UC, Merced

Performance Concerns are Everywhere



Programs need to be efficient at all scales

Performance is Money

Drop sales by 1%
every 0.1s of latency

Drop traffic by 20%
every 0.5s of latency

Drop visitors by 40%
after 3s of latency

Wasteful Memory Operations

Silent load

Account for 90% of memory loads on SPEC CPU2006

```
x = A[i];
y = A[i];     y = x;
```

Silent store

Account for 6% of memory stores on SPEC CPU2006

```
A[i] = 10;
   x = A[i];
A[i] = 10;
```

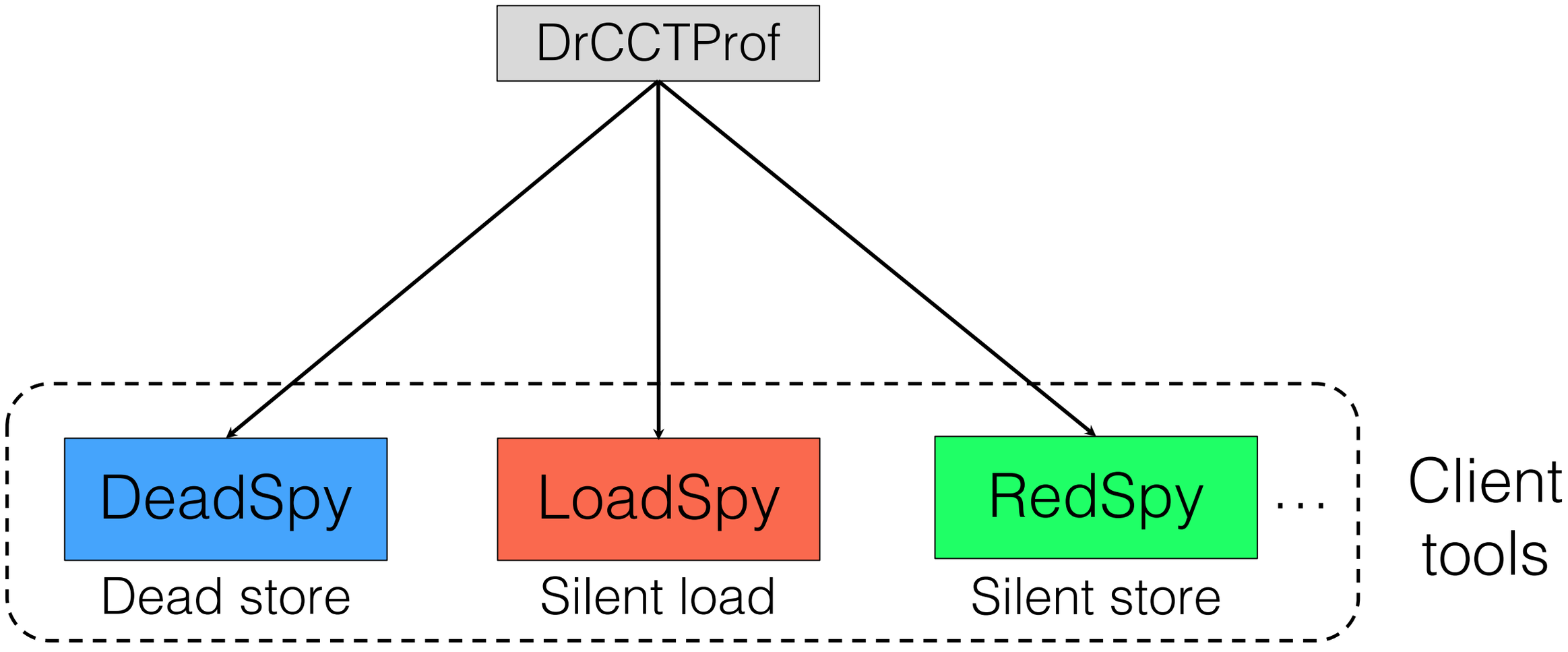
Dead store

Account for 20% of memory stores on SPEC CPU2006

```
A[i] = 0;
A[i] = 10;
```

Two operations involved:
one is dead/silent
because of the killing one

DrCCTProf: a Fine-grained Call Path Profiler

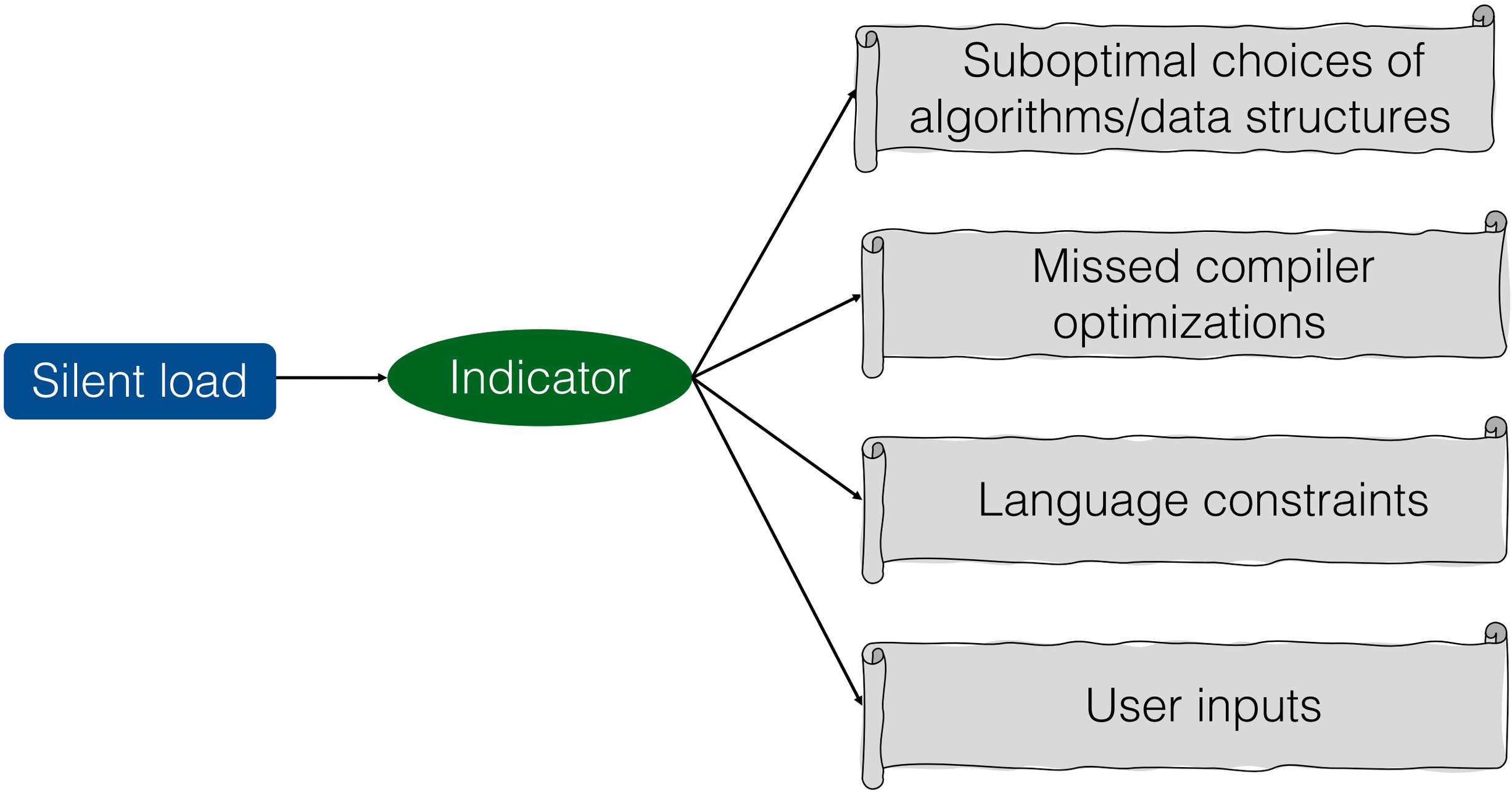


ACM SIGSOFT Distinguished Paper Award (ICSE'19)

Outline

- Provenance of silent loads
  - ✦ Case studies
    - C/C++ programs
    - Rust programs
- Design of LoadSpy
- Evaluation

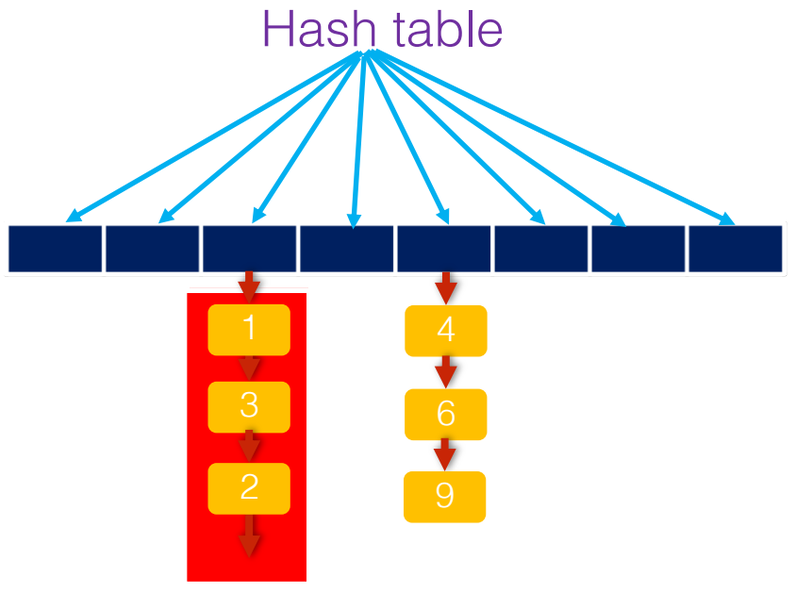
Silent Load: an Indicator of Performance Inefficiencies



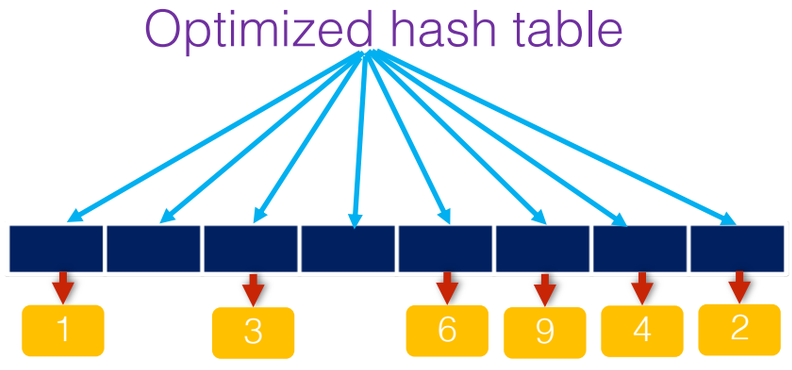
Suboptimal Choices of Algorithms/Data Structures

- Parsec-2.1 dedup (C/C++)

```
// invoked inside a loop
Hash_entry *linkedList_hashtable_search(hashtable *h, void *k) {
  …
  while (NULL != e) {
    if ((hashvalue == e->h) && (h->eqfn(k, e->k))) return e;
    e = e->next;
  }}
```



Hash buckets utilization: 2%



Hash buckets utilization: 80%

Speedup: 11%

Missed Compiler Optimizations

- SPEC CPU2006 H264ref (C/C++)

```
for (blky = 0; blky < 4; blky++) {
  for (y = 0; y < 4; y++) {
    refptr = funcPtr(…, img_height, img_width);
 }}
```

Missing function inlining

Loop invariants

Optimization: inline funcPtr() in its caller
Speedup: 28%

Language Constraints

- rustfmt
  - A tool for formatting code style
  - Developed and maintained by the Rust team

Iterators

```
let left_skip  = left.clone().skip(leading_equals).take(left_diff_size);
let right_skip = right.clone().skip(leading_equals).take(right_diff_size);

for (i, l) in left_skip.clone().enumerate() {
  for (j, r) in right_skip.clone().enumerate() {
    …
  }}
```

Lazy evaluation

Silent loads

Reason: for the iterator "right”: skip() and take() are invoked in each iteration of the outer loop whereas their parameters are loop invariants.

Language Constraints (Cont.)

- Optimized rustfmt

```
let left_skip  = left.clone().skip(leading_equals).take(left_diff_size);
let right_skip =
right.clone().skip(leading_equals).take(right_diff_size).enumerate().collect::<Vec<_>>();
for (i, l) in left_skip.clone().enumerate() {
  for (j, r) in right_skip.clone().enumerate() {

    …
  …
}}
```

Optimization: convert “right” to a vector

Language Constraints (Cont.)

- Optimized rustfmt
  - Further investigation

```
let left_skip  = left.clone().skip(leading_equals).take(left_diff_size);
let right_skip =
right.clone().skip(leading_equals).take(right_diff_size).enumerate().collect::<Vec<_>>();
for (i, l) in left_skip.clone().enumerate() {
  for (j, r) in right_skip.clone().enumerate() {

    ...
}}
```

Silent loads

Can we directly remove clone()?

No, due to the ownership constraint 😭

Language Constraints (Cont.)

- Further optimized rustfmt

```
let left_skip  = left.clone().skip(leading_equals).take(left_diff_size);
let right_skip =
right.clone().skip(leading_equals).take(right_diff_size).enumerate().collect::<Vec<_>>();
for (i, l) in left_skip.clone().enumerate() {
  for (j, r) in &right_skip.enumerate() {

    …
…
}}
```

“References allow you to refer to some value without taking ownership of it.”

Speedup: 7x after eliminating lazy evaluation and redundant clones

User Inputs

- Rodinia-3.1 backprop (C/C++)

```
1 for (j = 1; j <= ndelta; j++) {
2    new_dw = ETA * delta[j];
3    w[k][j] += new_dw;
4 }
```

Optimization

```
1 for (j = 1; j <= ndelta; j++) {
2    if (delta[j] == 0) continue;
3    new_dw = ETA * delta[j];
4    w[k][j] += new_dw;
5 }
```

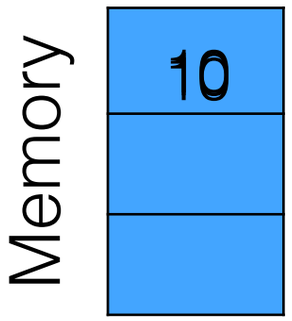
Optimization: conditional check
Speedup: 13%

Outline

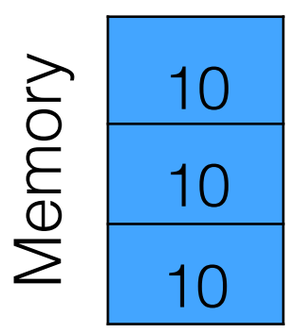
- Provenance of silent loads
  - ✦ Case studies
    - C/C++ programs
    - Rust programs
- Design of LoadSpy
- Evaluation

Type of Silent Loads

- Temporal silent load
  - Repeatedly load same value from same memory location

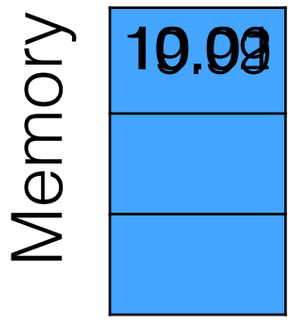


- Spatial silent load
  - Repeatedly load same value from nearby memory locations

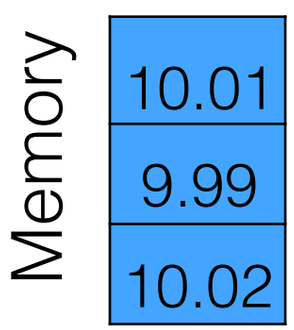


Type of Silent Loads

- Temporal silent load
  - (Floating-point operations) Repeatedly load (approximately) same value from same memory location

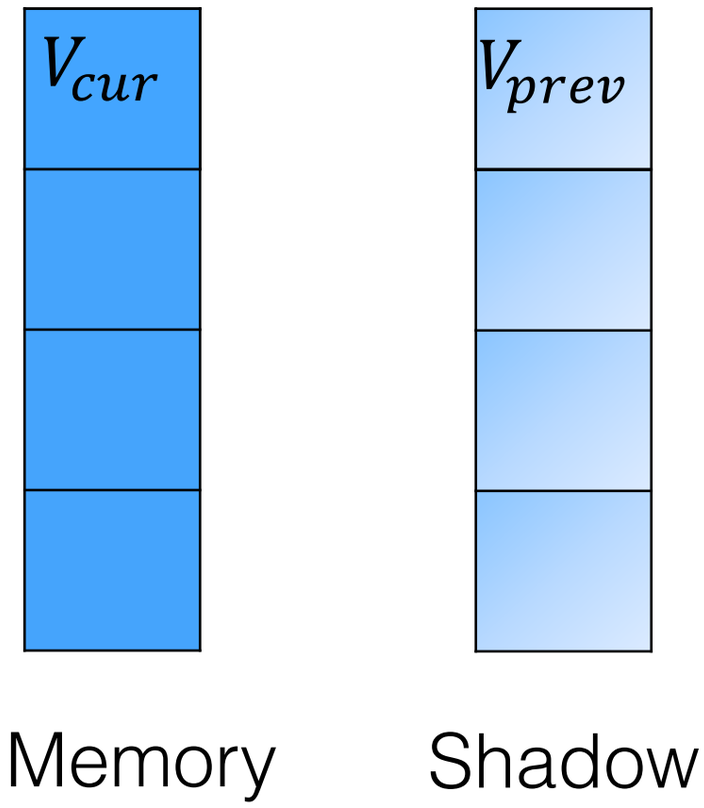


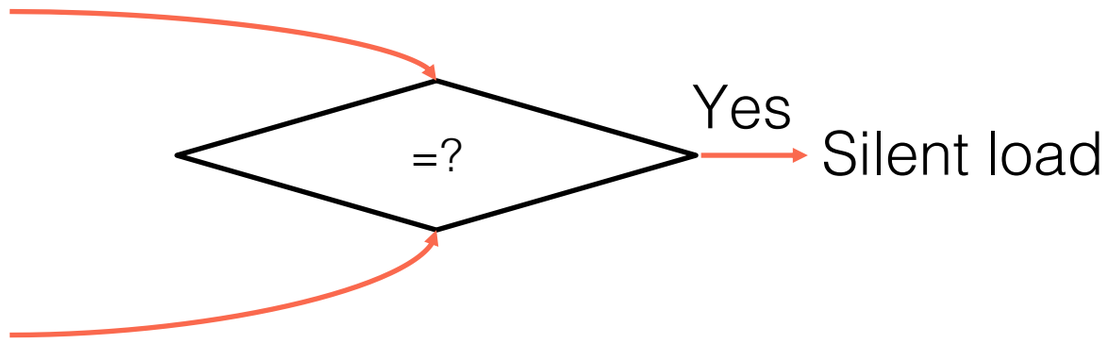
- Spatial silent load
  - (Floating-point operations) Repeatedly load (approximately) same value from nearby memory locations



Temporal Silent Load Detection

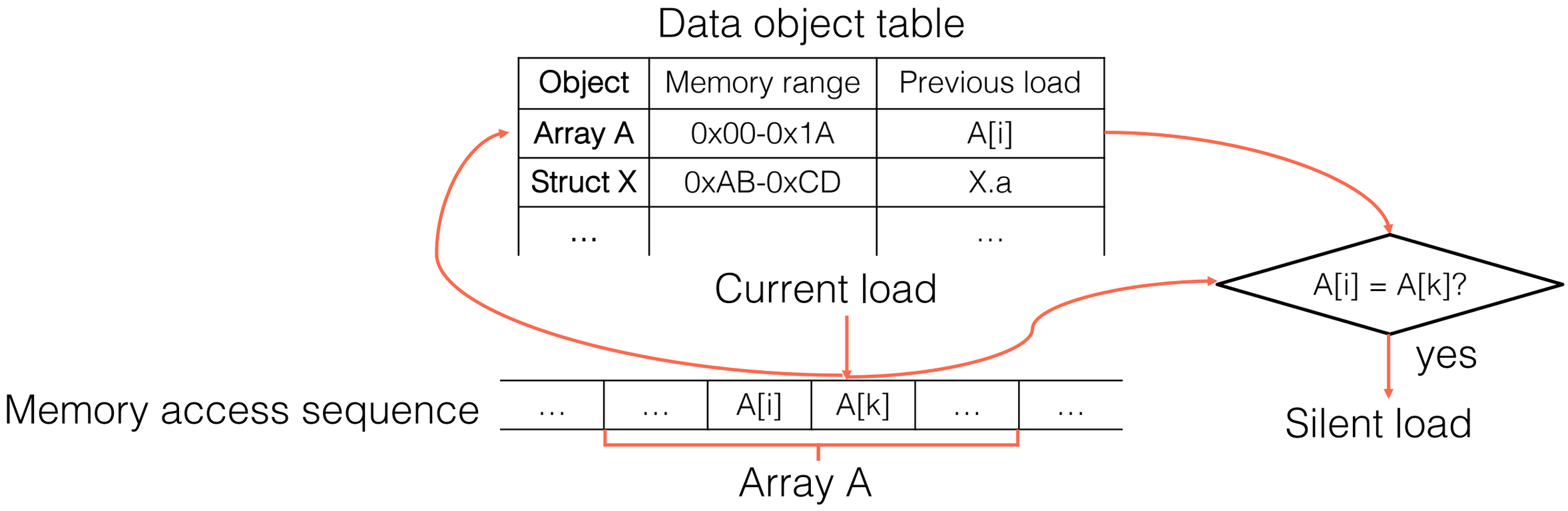
- Intercept every memory load to obtain current loaded value (V_{cur})
- Employ shadow memory to save previous loaded value (V_{prev})





Spatial Silent Load Detection

- Intercept every memory load to obtain its value
- Employ memory shadow to save previous loaded values
- Identify the memory range allocated for a data object
  - Static object: read symbol table
  - Dynamic object: intercept malloc() family of functions and mmap()



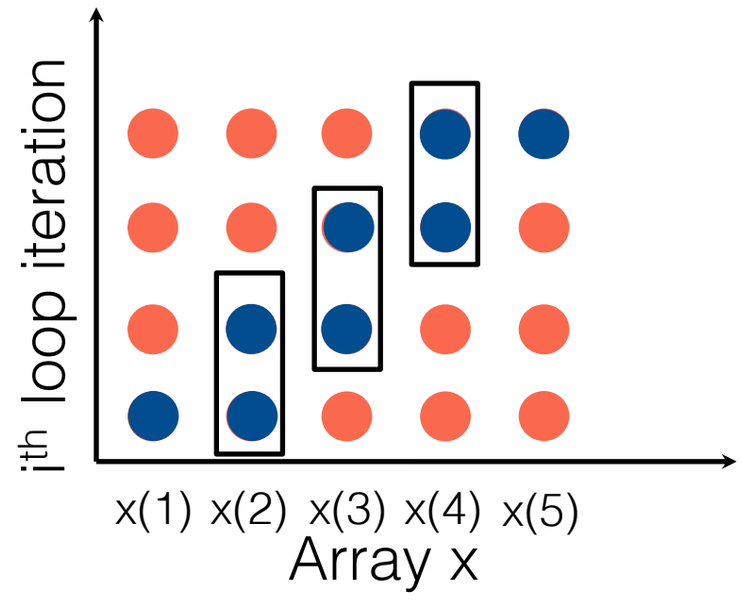
Redundancy Scope Detection

- Redundancy scope: which loop (if any) carries silent loads

```
1 do k = 1, kl
2   xx = x0 – deltt * …
3   do i = 1, i1
4     if (xx >= x(i) .and. xx <= x(i+1)) then
5       ixx = i; exit
6     endif
7   enddo
8 enddo
```

MASNUM (Fortran, 2016 ACM Gordon Bell Prize finalist)

Scope	Inefficiency	Optimization	Speedup
Inner loop	Stencil computation	Scala replacement: placing x(i+1) in a temporary variable	1%



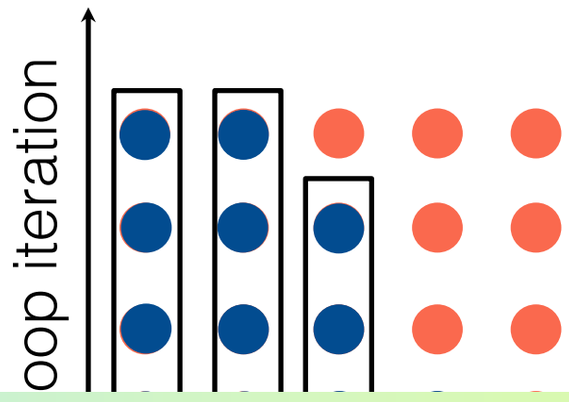
Redundancy Scope Detection

- Redundancy scope: which loop (if any) carries silent loads

```
1 do k = 1, kl
2   xx = x0 – deltt * …
3   do i = 1, i1
4     if (xx >= x(i) .and. xx <= x(i+1)) then
5       ixx = i; exit
6     endif
7   enddo
8 enddo
```

MASNUM (2016 ACM Gordon Bell Prize finalist)

Scope	Inefficiency	Optimization	Speedup
Inner loop	Stencil computation	Scala replacement: placing x(i+1) in a temporary variable	1%
Outer loop	Linear search	Binary search	30%



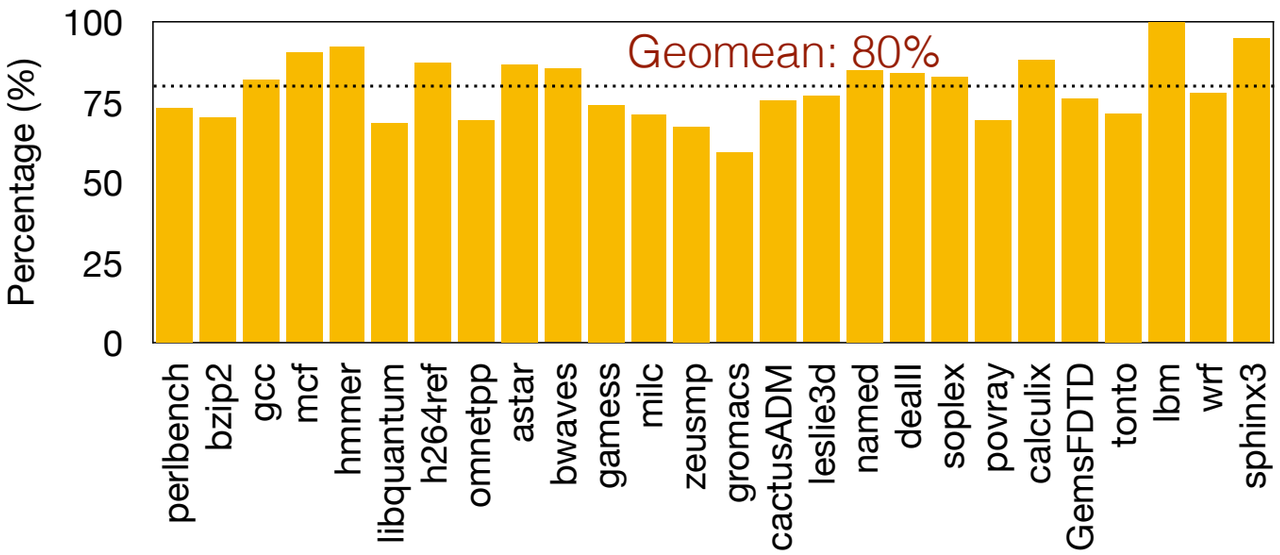
Solution: static interval analysis + dynamic instrumentation

Outline

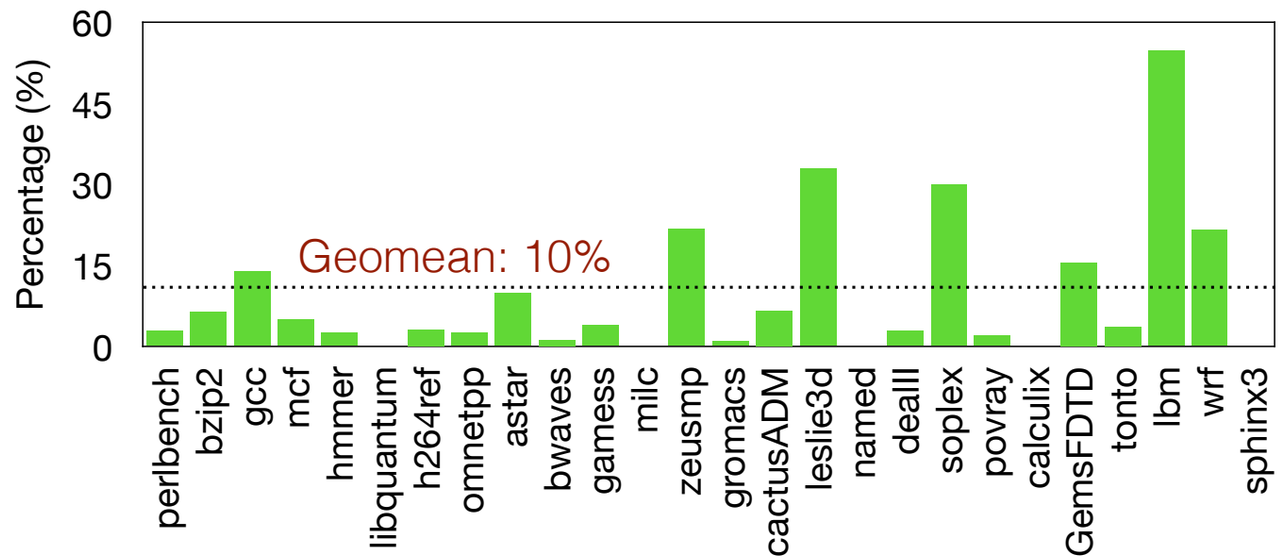
- Provenance of silent loads
  - ✦ Case studies
    - C/C++ programs
    - Rust programs
- Design of LoadSpy
- Evaluation

% of Silent Loads on SPEC CPU2006

- Compiled with GCC –O3 PGO LTO



Temporal silent loads



Spatial silent loads

$$\text{Temporal/spatial silent loads (\%)} = \frac{Temporal/spatial\ silent\ loads}{Total\ memory\ loads} \times 100\%$$

Case Studies

Program			LoC	Inefficiency	Speedup
Benchmark	SPEC CPU2006	h264ref	58K	Missing inline substitution	1.28x
		lbm	3K	Redundant computation	1.25x
	SPEC OMP2012	botsspar	3K	Inefficient register usage	1.77x
	SPEC CPU2017	imagick_r	274K	Redundant computation	1.25x
		povray	159K	Missing inline substitution	1.05x
	Rodinia-3.1	backprop	1K	Input-sensitive redundancy	1.13x
		hotspot3D	800	Inefficient register usage	1.13x
		lavaMD	800	Redundant function calls	1.39x
		srad_v1	600	Inefficient register usage	1.11x
		srad_v2	200	Inefficient register usage	1.12x
		particlefilter	600	Linear search	9.8x
	Stamp-0.9.10	vacation	44K	Redundant function calls	1.24x
	Parsec-2.1	dedup	11K	Poor hashing	1.11x
	NERSC-8	msgrate	2K	Missing constant propagation	3.03x
Real application	Apache Avro-1.8.2		46K	Missing inline substitution	1.19x
	Hoard-3.12		22K	Redundant computation	1.14x
	MASNUM-2.2		121K	Linear search	1.79x
	USQCD Chroma-3.43		929K	Missing inline substitution	1.06x
	Shogun-6.0		546K	Missing inline substitution	1.06x
	Facebook Stack RNN		2K	Redundant computation	1.09x
	Binutils-2.27		2M	Linear search	3.29x

Ongoing Work

- A benchmark suite for modern native languages
  - Develop a set of benchmarks with compiler- and language-related inefficiencies

    Rust

    Go
  - Give feedback to compiler and language developers for better code optimization

Conclusions

- Many kinds of software inefficiencies manifest as silent loads
  - E.g., algorithms, data structures, compiler transformations, language constraints
- We developed LoadSpy -- a tool to pinpoint and quantify silent loads
- LoadSpy
  - Works for a variety of natively compiled languages, e.g., C/C++, Fortran, Rust, Go, Swift
  - Automates important use cases to help developers investigate load redundancy
  - Opens a new avenue to tune software for high performance

Questions?