Comparing Optimization Strategies

Learn From My Mistakes

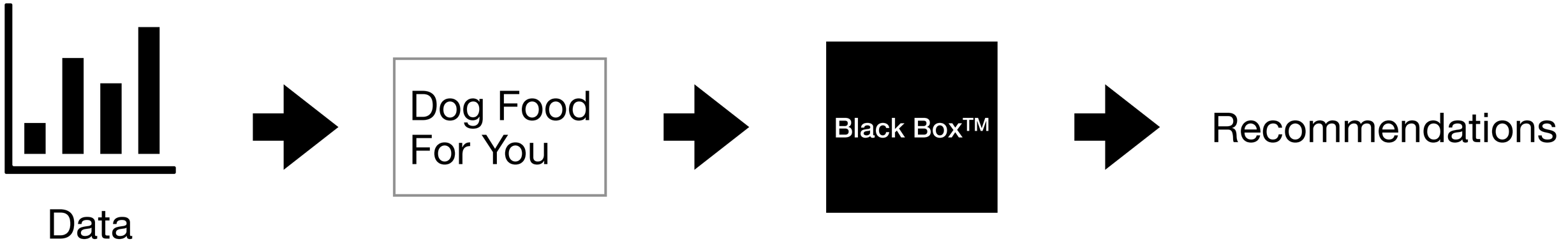
Mark Lodato

Goals

- Talk about optimizing real code
- Compare different approaches taken and their pay-offs
- Show how we can reach for use-case-specific algorithms
- Introduce a handy tool for analyzing code performance
- Start a conversation about optimization

Dog Food For You

State of the Art Black Box™ Technology



Optimization Strategy 1

Guess

```
pub fn write_file(...) {

    for dog in mapping.dogs() {









    }
}
```

Optimization Strategy 1

Guess

```
pub fn write_file(...) {

    for dog in mapping.dogs() {

        for _ in 0..lines_per_dog {




        }
    }
}
```

Optimization Strategy 1

Guess

```
pub fn write_file(...) {

    for dog in mapping.dogs() {

        for _ in 0..lines_per_dog {

            let line = walk(...);

            output_file.write_all(...);
        }
    }
}
```

Optimization Strategy 1

Guess

```
pub fn walk(...) -> String {

    let mut line: Vec<String> = ...;

    // bunch of line.push()s

    line.join(" ") + "\n"
}
```

Optimization Strategy 1

Guess

```
test array_concat                                ... bench:          38 ns/iter (+/- 27)
test array_join                                  ... bench:          38 ns/iter (+/- 22)
test array_join_long                             ... bench:          35 ns/iter (+/- 18)
test collect_from_array_to_string                ... bench:          80 ns/iter (+/- 17)
test collect_from_vec_to_string                  ... bench:          76 ns/iter (+/- 27)
test format_macro                                ... bench:         111 ns/iter (+/- 69)
test from_bytes                                  ... bench:           1 ns/iter (+/- 0)
test mut_string_push_str                         ... bench:          75 ns/iter (+/- 35)
test mut_string_push_string                      ... bench:         164 ns/iter (+/- 77)
test mut_string_with_capacity_push_str           ... bench:          32 ns/iter (+/- 14)
test mut_string_with_capacity_push_str_char      ... bench:          29 ns/iter (+/- 12)
test mut_string_with_too_little_capacity_push_str ... bench:         96 ns/iter (+/- 11)
test mut_string_with_too_much_capacity_push_str  ... bench:          37 ns/iter (+/- 8)
test string_from_all                             ... bench:         133 ns/iter (+/- 101)
test string_from_plus_op                         ... bench:          76 ns/iter (+/- 9)
test to_owned_plus_op                            ... bench:          81 ns/iter (+/- 15)
test to_string_plus_op                           ... bench:          83 ns/iter (+/- 17)
```

https://github.com/hoodie/concatenation_benchmarks-rs

Optimization Strategy 1

Guess

```
test array_concat                                ... bench:          38 ns/iter (+/- 27)
test array_join                                  ... bench:          38 ns/iter (+/- 22)
test array_join_long                             ... bench:          35 ns/iter (+/- 18)
test collect_from_array_to_string                ... bench:          80 ns/iter (+/- 17)
test collect_from_vec_to_string                  ... bench:          76 ns/iter (+/- 27)
test format_macro                                ... bench:         111 ns/iter (+/- 69)
test from_bytes                                  ... bench:           1 ns/iter (+/- 0)
test mut_string_push_str                         ... bench:          75 ns/iter (+/- 35)
test mut_string_push_string                      ... bench:         164 ns/iter (+/- 77)
test mut_string_with_capacity_push_str           ... bench:          32 ns/iter (+/- 14)
test mut_string_with_capacity_push_str_char      ... bench:          29 ns/iter (+/- 12)
test mut_string_with_too_little_capacity_push_str ... bench:         96 ns/iter (+/- 11)
test mut_string_with_too_much_capacity_push_str  ... bench:          37 ns/iter (+/- 8)
test string_from_all                             ... bench:         133 ns/iter (+/- 101)
test string_from_plus_op                         ... bench:          76 ns/iter (+/- 9)
test to_owned_plus_op                            ... bench:          81 ns/iter (+/- 15)
test to_string_plus_op                           ... bench:          83 ns/iter (+/- 17)
```

https://github.com/hoodie/concatenation_benchmarks-rs

Optimization Strategy 1

Guess

```
pub fn walk(..., line: &mut String) {

    // bunch of line.push_str()s
    // and line.push(' ')s

    line.push('\n');
}
```

Test it out!

	Run 1 (ms)	Run 2 (ms)	Run 3 (ms)	Run 4 (ms)	Run 5 (ms)
join	59364	58472	59844	58018	58810
push_str	57407	57252	56730	57590	56278
diff	-3.3%	-2.09%	-5.2%	-0.74%	-4.31%


acer
WOODLAND

Guessing about performance is hard

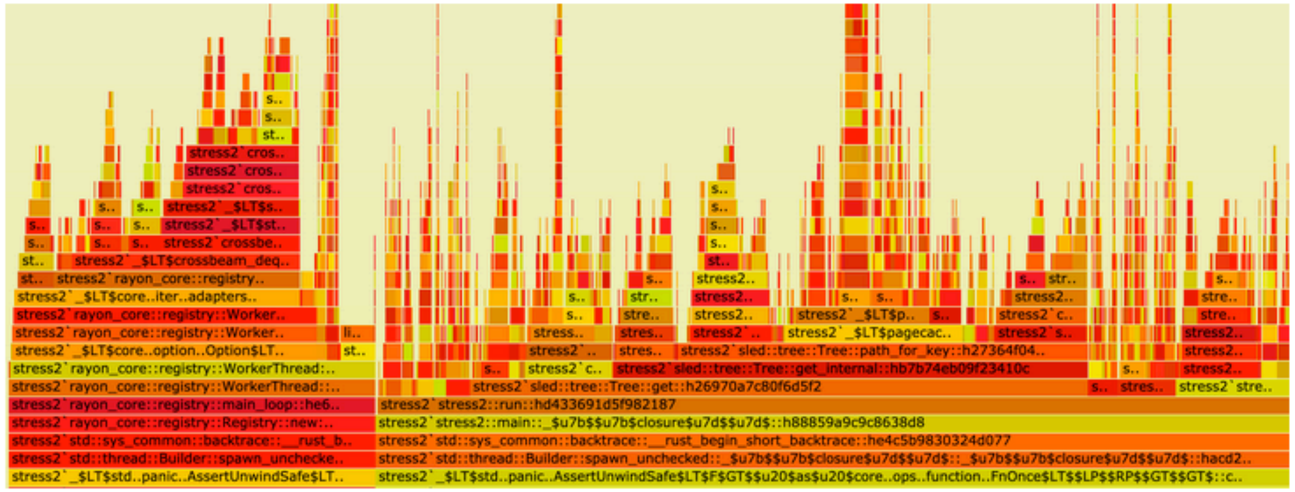
- Higher level languages exist for a reason

Optimization Strategy 2

Measure

README.md

[cargo-]flamegraph



A Rust-powered flamegraph generator with additional support for Cargo projects! It can be used to profile anything, not just Rust projects! No perl or pipes required <3

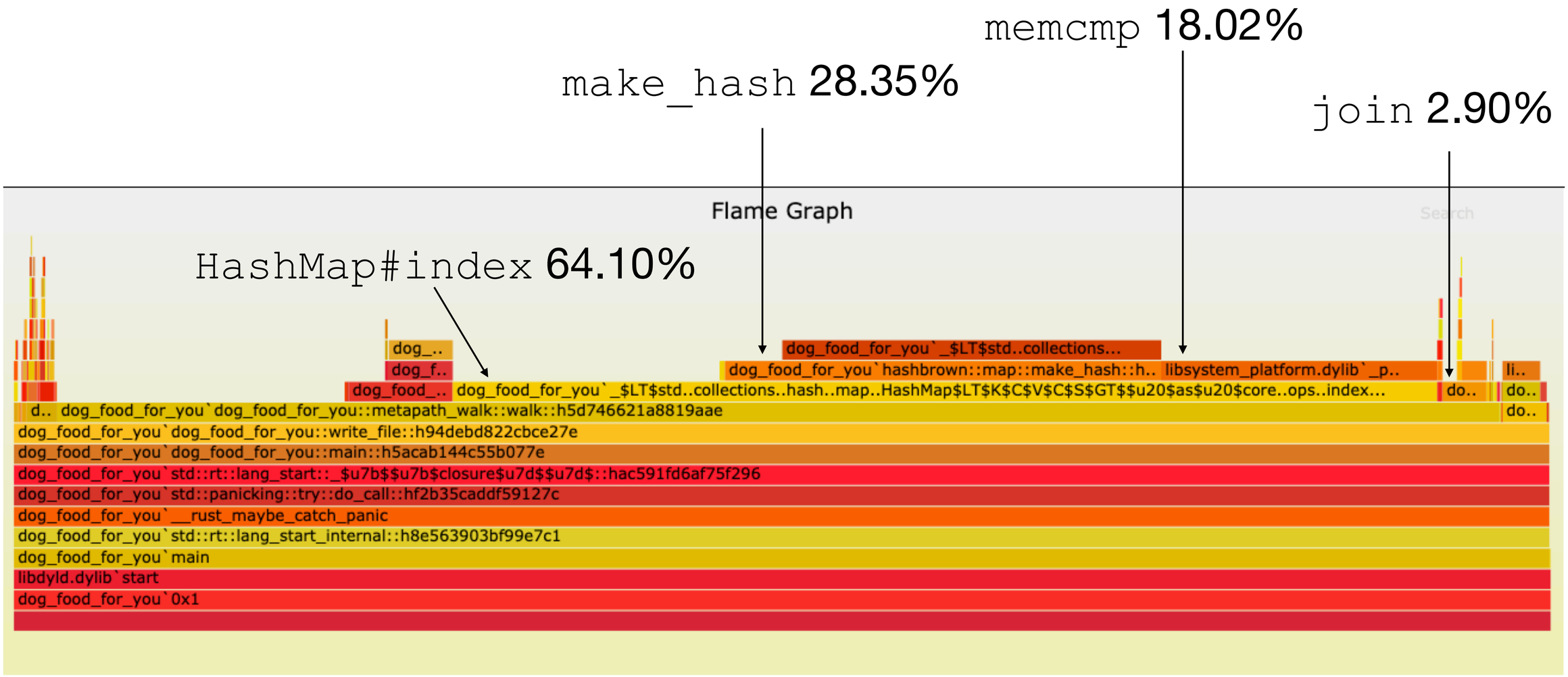
How to use flamegraphs: what's a flamegraph, and how can I use it to guide systems performance work?

Relies on perf on linux and dtrace otherwise. Built on top of @jonhoo's wonderful Inferno all-rust flamegraph generation library!

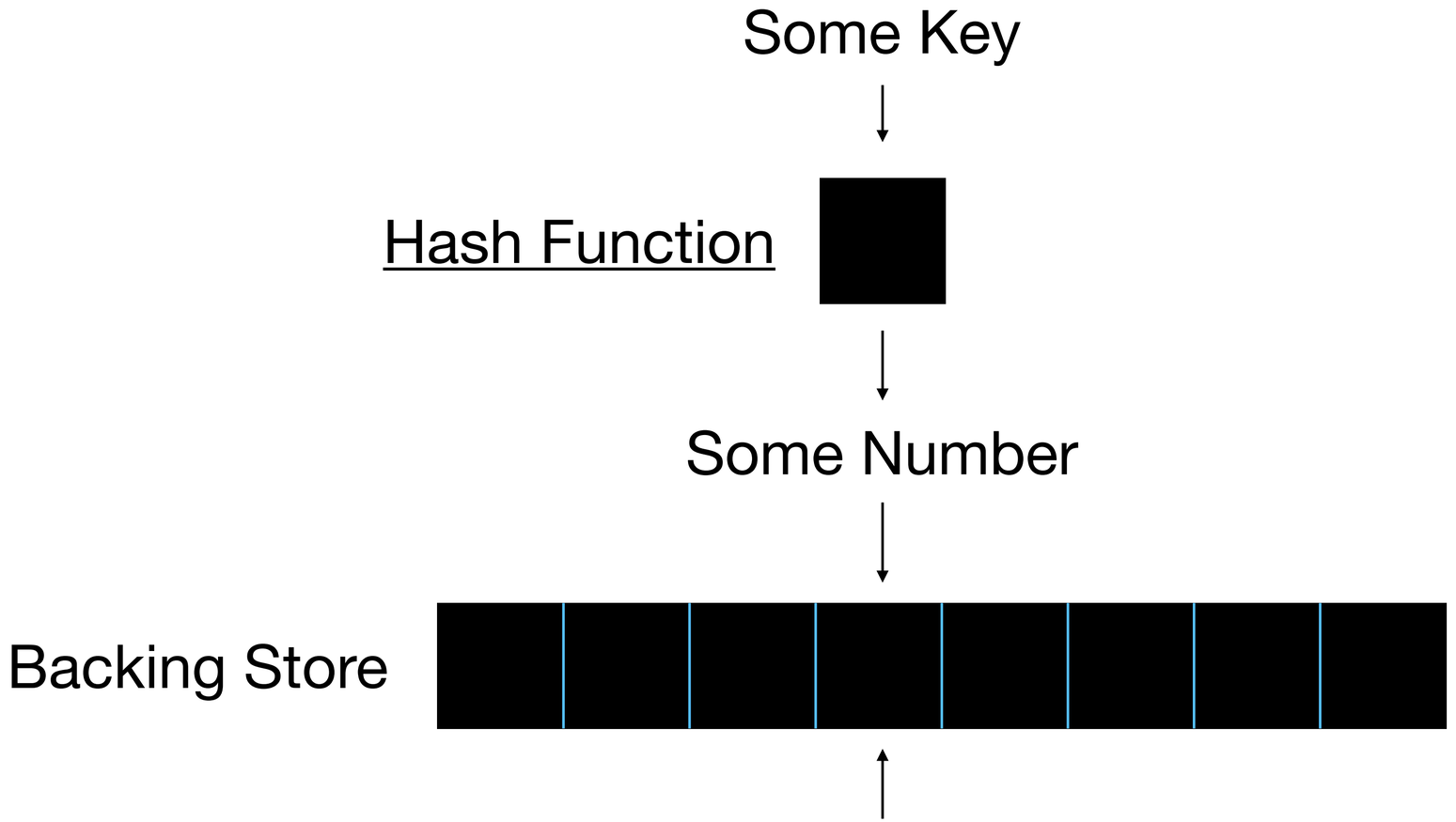
Windows is getting dtrace support, so if you try this out please let us know how it goes :D

https://github.com/flamegraph-rs/flamegraph

Measure


memcmp 18.02%
make_hash 28.35%
join 2.90%
Flame Graph
HashMap#index 64.10%
dog_..
dog_f..
dog_food_..
dog_food_for_you`_LTstd..collections...
dog_food_for_you`hashbrown::map::make_hash::h..
libsystem_platform.dylib`_p..
li..
dog_food_for_you`_LTstd..collections..hash..map..HashMapLTKCVCS$GT$$u20asu20$core..ops..index...
do..
do..
d..
dog_food_for_you`dog_food_for_you::metapath_walk::walk::h5d746621a8819aae
do..
dog_food_for_you`dog_food_for_you::write_file::h94debd822cbce27e
dog_food_for_you`dog_food_for_you::main::h5acab144c55b077e
dog_food_for_you`std::rt::lang_start::_$u7b$$u7b$closure$u7d$$u7d$::hac591fd6af75f296
dog_food_for_you`std::panicking::try::do_call::hf2b35caddf59127c
dog_food_for_you`__rust_maybe_catch_panic
dog_food_for_you`std::rt::lang_start_internal::h8e563903bf99e7c1
dog_food_for_you`main
libdyld.dylib`start
dog_food_for_you`0x1

HashMap Refresh

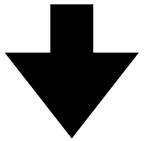


Multiple keys will map to the same array index

Collisions need to be handled and keys need to be compared

The Change

```
struct Map {
    dog_to_food: HashMap<String, Vec<String>>,
    food_to_ingredient: ...
    ...
}
```



```
struct Map {
    dog_to_food: Vec<Vec<u16>>,
    food_to_ingredient: ...
    ...
    dog_ids: Vec<String>,
    ...
}
```

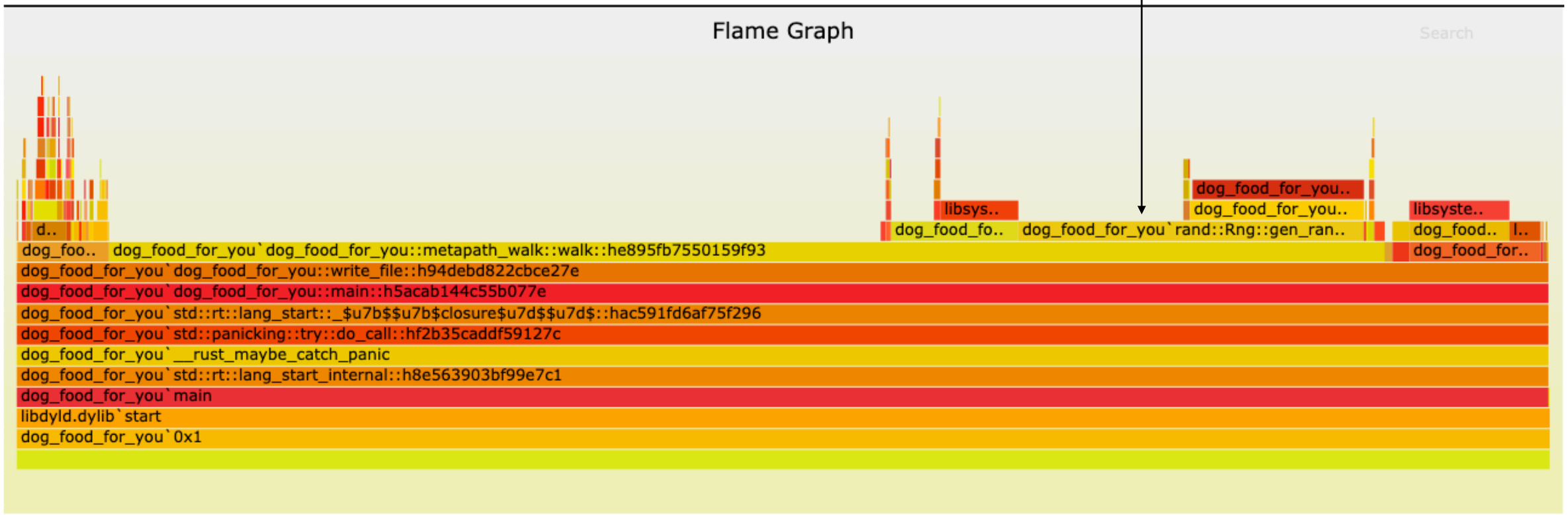
Test it out!

	Run 1 (ms)	Run 2 (ms)	Run 3 (ms)	Run 4 (ms)	Run 5 (ms)
HashMap	58291	58060	55639	55264	56900
Vec	16052	15311	15865	16001	15324
diff	-72.46%	-73.63%	-71.49%	-71.05%	-73.07%

Optimization Strategy 2.5

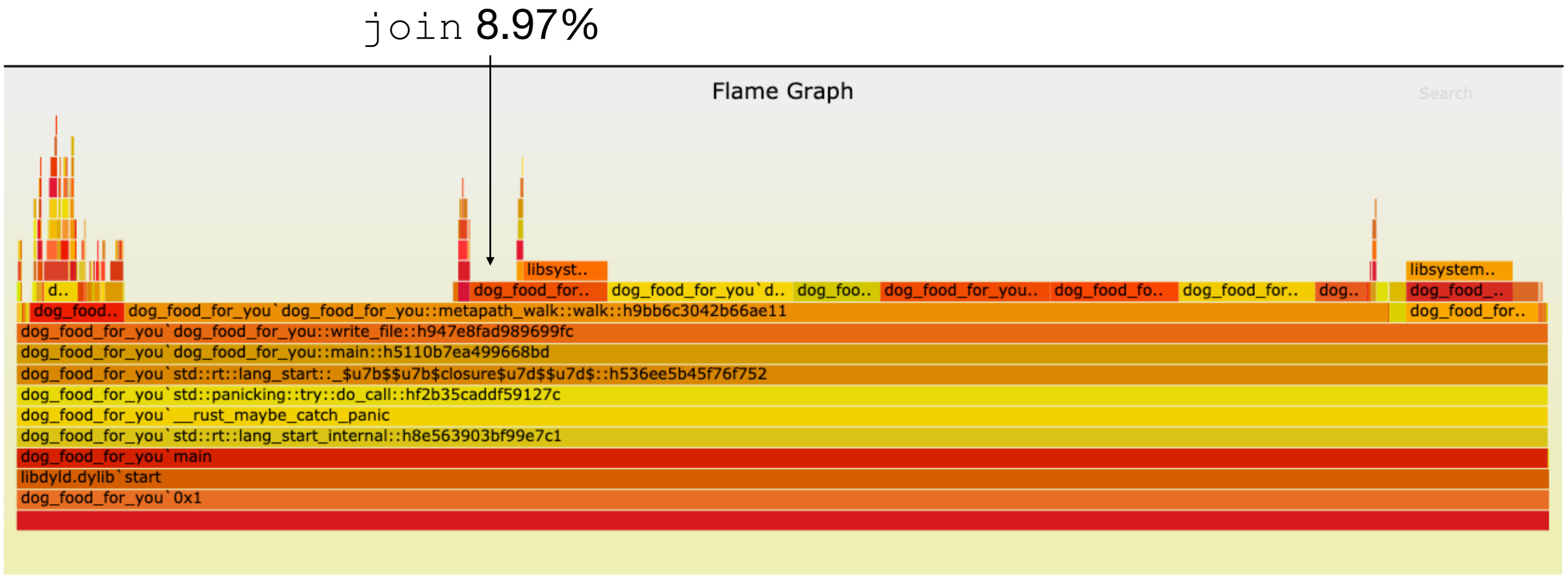
The Rabbit Hole

`rand::Rng::gen_range` 22.50%



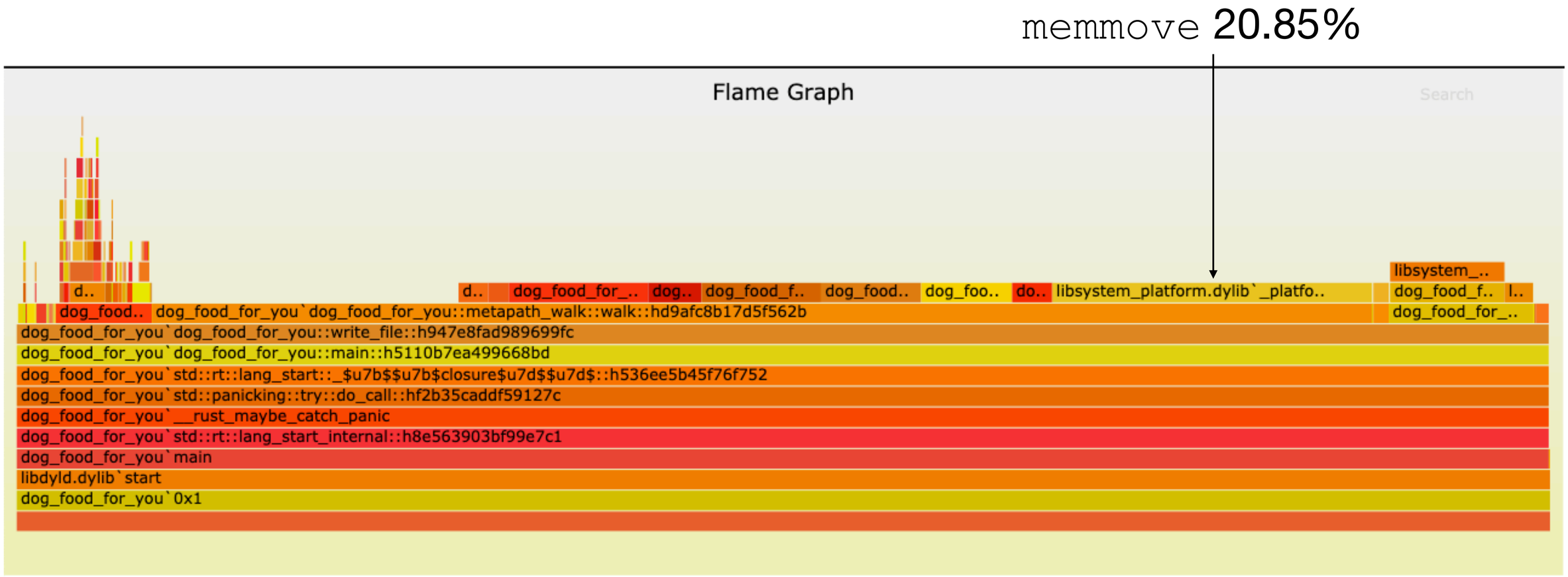
Optimization Strategy 2.75

The Rabbit Hole



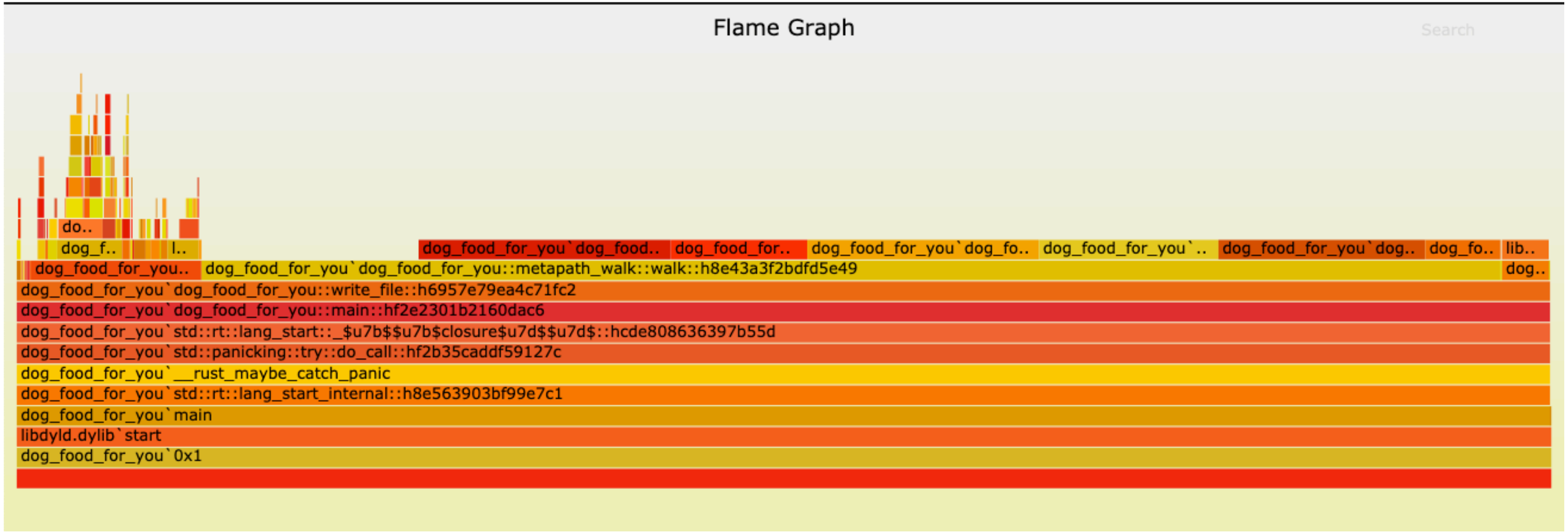
Optimization Strategy 2.875

The Rabbit Hole



Optimization Strategy 2.9375

The Rabbit Hole



Test it out!

	Run 1 (ms)	Run 2 (ms)	Run 3 (ms)	Run 4 (ms)	Run 5 (ms)
Vec	18275	18592	18182	18975	18500
Bottom of the Rabbit Hole	10193	10068	9977	9864	10031
diff	-44.22%	-45.85%	-45.13%	-48.02%	-45.78%

Profiling code helps focus efforts

- Algorithmic changes often result in larger changes than micro-optimizations
- It's easy to get fall down the rabbit hole fixing less important things

Optimization Strategy 3

Fearless Concurrency

Optimization Strategy 3

Fearless Concurrency

```
pub fn write_file(...) {
    for dog in mapping.dogs() {
        for _ in 0..lines_per_dog {
            ...
        }
    }
}
```

- https://github.com/rayon-rs/rayon
- https://github.com/crossbeam-rs/crossbeam

The Diff

```
+     let (sender, receiver) = unbounded();
+
+     thread::spawn(move || {
+         for line in receiver {
+             output_file.write_all(line.as_bytes()).unwrap();
+         }
+         output_file.flush().unwrap();
      });

      for dog in mapping.dogs() {
-         for _ in 0..lines_per_dog {
+         (0..lines_per_dog).into_par_iter().for_each(|_| {
              let mut rng = rand::thread_rng();
              let line = walk(walks_per_line, &dog, &mapping, &
-             output_file.write_all(line.as_bytes()).unwrap();
-         }
+             sender.clone().send(line).unwrap();
+         });
      }
```

Send lines to thread for writing

Change `for` loop into `into_par_iter()`

Test it out!

	Run 1 (ms)	Run 2 (ms)	Run 3 (ms)	Run 4 (ms)	Run 5 (ms)
Original HashMap	61582	65611	61265	61307	63206
Parallelized HashMap	11377	11355	10931	11035	11528
diff	-81.53%	-82.69%	-82.16%	-82%	-81.76%

Combining these strategies

	Run 1 (ms)	Run 2 (ms)	Run 3 (ms)	Run 4 (ms)	Run 5 (ms)
Bottom of the Rabbit Hole	7507	7650	8116	8333	7294
Parallelized	1985	1943	2125	2118	2071
diff	-73.56%	-74.6%	-73.82%	-74.58%	-71.61%

Concurrency can be an easy win

- Note - some programs can't be made concurrent!

Summary

- Guessing about performance is hard
- Flamegraphs are a great tool to inform optimization choices
- Algorithmic changes are often more impactful than micro-optimizations
- Fearless concurrency is awesome