Tremor

How Rust killed thousands of cores and TB of memory at Wayfair

Agenda

- A bit about us
- What is tremor
- Tremor Script
- <3 OSS

What is Wayfair?

- Sells rugs (and couches!)
- Large online retailer for furniture
- US, UK, DE
- >1000 Employees in Berlin ~25000 worldwide
- We do have a few computers!

Who are we?

- Small team (2 in Berlin, 0.5 in Boston)
- We do Systems Engineering at Wayfair
- Built Tremor
- Sometimes talk about it!
- Darach (@darachennis)
- Heinz (@heinz_gies)
- Anup (@refusestousetwitter)

What is Tremor

- A event processing engine
- An ETL language
- A query language
- Replaced Logstash
- Replaced Telegraf
- We now integrating with k8s

Event processing Cartography Disclaimer

- This is not going to be a scientific accurate data
- But it’s fine
- I’ve made up my mind about this, come near me with your facts about reality
- It lacks nuance but adds fun

You’re-going-to-write-some-f**ing-java-if-you-want-to-or-not Island

This is where platforms like Flink or Spark live and the inhabitants happily write their Java code to customise event logic.

Usually experienced developers that are happy to go into low level code.

The Archipelago of lets-cobble-transformations-together

This is where Logstash for example lives, the inhabitants of those little islands tend to like putting together pre configured blocks together a lot.

Usually ops teams that ‘just need to get s***t done!’ without experience in writing software but incredible at configuring it.

The make-your-own-language Atoll

The atoll hosts a small language that coexists with the much larger runtime island next to it. It's inhabitants know to wield ~~fire~~ scripting but do not want to go all in into programming.

Ops teams that need the extra oompf! A mix of some hard coded operators (tansforms) for performance and a scripting language to customize logic without needing all the complexity of something like Java.

Tremor Script

- An ETL language
- Parse, Transform and filter JSON-esque data structures
- Not the language we want but the language we need
- Influenced by Rust and Erlang along with a good pinch of “what was needed”

New in 0.8 (to be pre released tomorrow)

- modules - logically encapsulated code
- use - include and preprocess code from other files.
- (I think you can see the Rust influence here ;)
- Functions - you can write and call
- Intrinsics - a easy way to expose functions written in rust as part of a library

modules

- Allow encapsulating
  - Constants
  - Functions
- Can be nested (modules in modules)
- Prevent clashes (i.e the len function)

```
1 | mod my_mod with
2 |   const answer = 42;
3 |   fn add(a, b) with
4 |     a + b
5 |   end;
6 | end;
7 | my_mod::add(my_mod::answer, -19)
```

use

- Split modules into files
- No need for **mod** everywhere
- Allows multiple search paths
- Will nest

```
1 | const snot = "badger";
```

```
1 | use foo;
2 |
3 | match event of
4 |   case {} => "snot {foo::snot}"
5 |   default => "ko"
6 | end;
```

```
 1 | #!line /home/heinz/Projects/tremor/tremor-runtime/tests/scripts/pp_nest0/foo.tremor 0
 2 | mod foo with #!line /home/heinz/Projects/tremor/tremor-runtime/tests/scripts/pp_nest0/foo.tremor 0
 3 | const snot = "badger";
 4 | end;
 5 | #!line /home/heinz/Projects/tremor/tremor-runtime/tests/scripts/pp_nest0/script.tremor 2
 6 |
 7 | match event of
 8 |   case {} => "snot {foo::snot}"
 9 |   default => "ko"
10 | end;
```

functions

- Can abstract logic and algorithms
- Parameters
- Always return a value!

functions - patterns

- Can match on arguments
- Executes part of the function based on them
- Can use all kinds of patterns down to extractors

```
1  fn snottify(s) of
2    case ("badger") => "snot badger, hell yea!"
3    case (~ json||) => let s.snot = true, s
4    case (s) when core::type::is_string(s) => "snot {s}"
5    default => "snot caller, you can't snottify that!"
6  end;
7
8  snottify("badger");
9  #  => "snot badger, hell yea!"
10 snottify("horse");
11 #  => "snot hores"
12 snottify(42);
13 #  => "snot caller, you can't snottify that!"
14 snottify(core::json::encode({"badger":42}));
15 #  => {"badger": 42, "snot": true}
```

functions - varargs

- Multiple arguments
- Can have a fixed number of named arguments in the beginning

```
1 | fn snot_all_the_things(...) with
2 |   "snot {core::array::join(args, " snot ")}"
3 | end;
4 |
5 | snot_all_the_things("badger", "horse", "cat")
6 | #  => "snot badger snot horse snot cat
```

functions - recursion

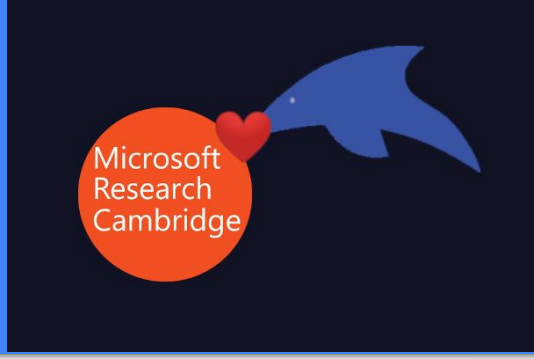
- Support for recursive functions
- Enforced tail recursion
- Uses recur keyword to make recursion obvious
- Limited recursion depth (no infinite loops!)

```
1  | fn fib_(a, b, n) of
2  |   case (a, b, n) when n > 0 =>  recur(b, a + b, n - 1)
3  |   default => a
4  | end;
5  |
6  | fn fib(n) with
7  |   fib_(0, 1, n)
8  | end;
9  |
10 | for core::range::range(0, 10) of
11 |   case (i, i) => fib(i)
12 | end;
13 | #  => [0, 1, 1, 2, 3, 5, 8, 13, 21, 34
```

Tremor <3 OSS

- We open sourced february this year
- All work is happening in the open now
- Collaboration is important to us
- We try to give back

snmalloc

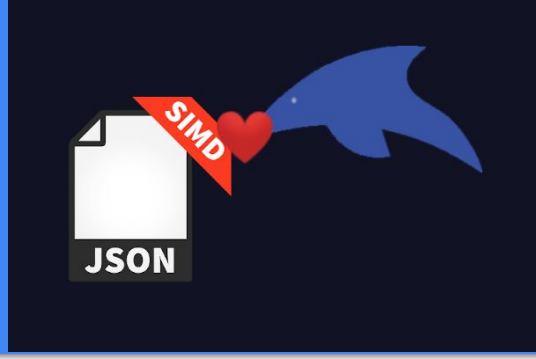


- An allocator specifically designed for producer/consumer patterns
- Aligns really well with what we're doing (produce -> modify -> consume)
- Working with Matthew P. @ MSR by sharing benchmarks and tracks
  - Result -> two performance improvement released, two more in progress
- https://github.com/microsoft/snmalloc
- @ParkyMatthew

Vector (https://vector.dev)

- Integration (send data between vector and tremor)
  - Example: log scraping (where vector excels) and classification (where tremor excels)
- Sharing of ideas and code
  - protobuf / wasm (thanks Ana!)
  - Generalised sinks / sources
  - simd-json

simd-json (https://simd-json.rs)



- Rust port of simdjson (we had to use a - because of crate squatting :/)
- Fastest JSON parser for rust (by far!)
- Contributing back bug fixes and performance tweaks
- https://github.com/simdjson
- @lemire

Thank you!

https://tremor.rs

@tremordebs

BACKUP

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[%{present rec}]} => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[%{present rec}]} => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[%{present rec}]} => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[%{present rec}]}  => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[%{present rec}] } => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[%{present rec}]} => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[ %{present rec}]} => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[%{present rec}]} => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[%{present rec}]} => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Structural Pattern Matching

```
# Filter incoming events for:
# * Records
#   * With an 'arr' array field
#     * That contains at least one record
#       * With a 'rec' field
#         * And return that array
match event of
 case r = %{ arr ~= %[%{present rec}]} => r.arr
 default => drop    # Drop non-matching events
end
```

Matching events

Given:

```
{ "arr": [
  { "rec": true },
  { "not-rec": true }
] }
```

Returns:

```
# Array of matches
[
  # [index,value]
 [ 0, { "rec": true }]
]
```

Tremor Query

- Original a YAML configuration file to describe processing graphs
- We hate YAML (sorry, not sorry)
- SQL is well known so we borrow the familiarity
- Does filtering, aggregation and graph building
- Structured not tabular

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Merge-capable aggregate functions

```
select {
   "measurement": event.measurement,
   "tags": patch event.tags of insert "window" => window end,
   "stats": stats::hdr(event.fields[group[2]], [ "0.5", "0.9", "0.99", "0.999" ]),
   "class": group[2],
   "timestamp": win::first(event.timestamp), # aggregate functions
}
from in[`10sec`, `1min`, `10min`, `1h`] # tilt frames
group by set(event.measurement, event.tags, each(record::keys(event.fields)))
into normalize;
```

Micro-Format Extraction

```
# match any valid CIDR
match event of
 case r = %{ ip ~= cidr|10.22.0.0/24| } => r.ip
 case r = %{ ip ~= cidr|| } => r.ip
 default => { "error": "bad IPv[46] addr" }
end;
```

Matching events

Given:

```
{ "ip": "10.22.0.24" }
```

Returns:

```
# Array of matches
{
 "prefix": [ 10, 22, 0, 24 ],
 "mask": [ 255, 255, 255, 255 ],
}
```

Convenient nested data structure templating

```
select {
 "measurement":  event.measurement,
 "fields":  {
   "min_{event.class}":  event.stats.min,
   "max_{event.class}":  event.stats.max,
   "mean_{event.class}":  event.stats.mean,
   "p99_{event.class}":  event.stats.percentiles["0.99"],
   # ...
 }
}
from normalize
into batch;
```