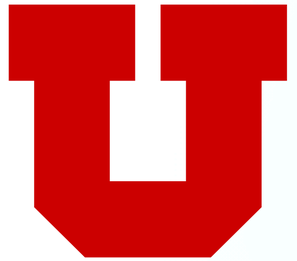
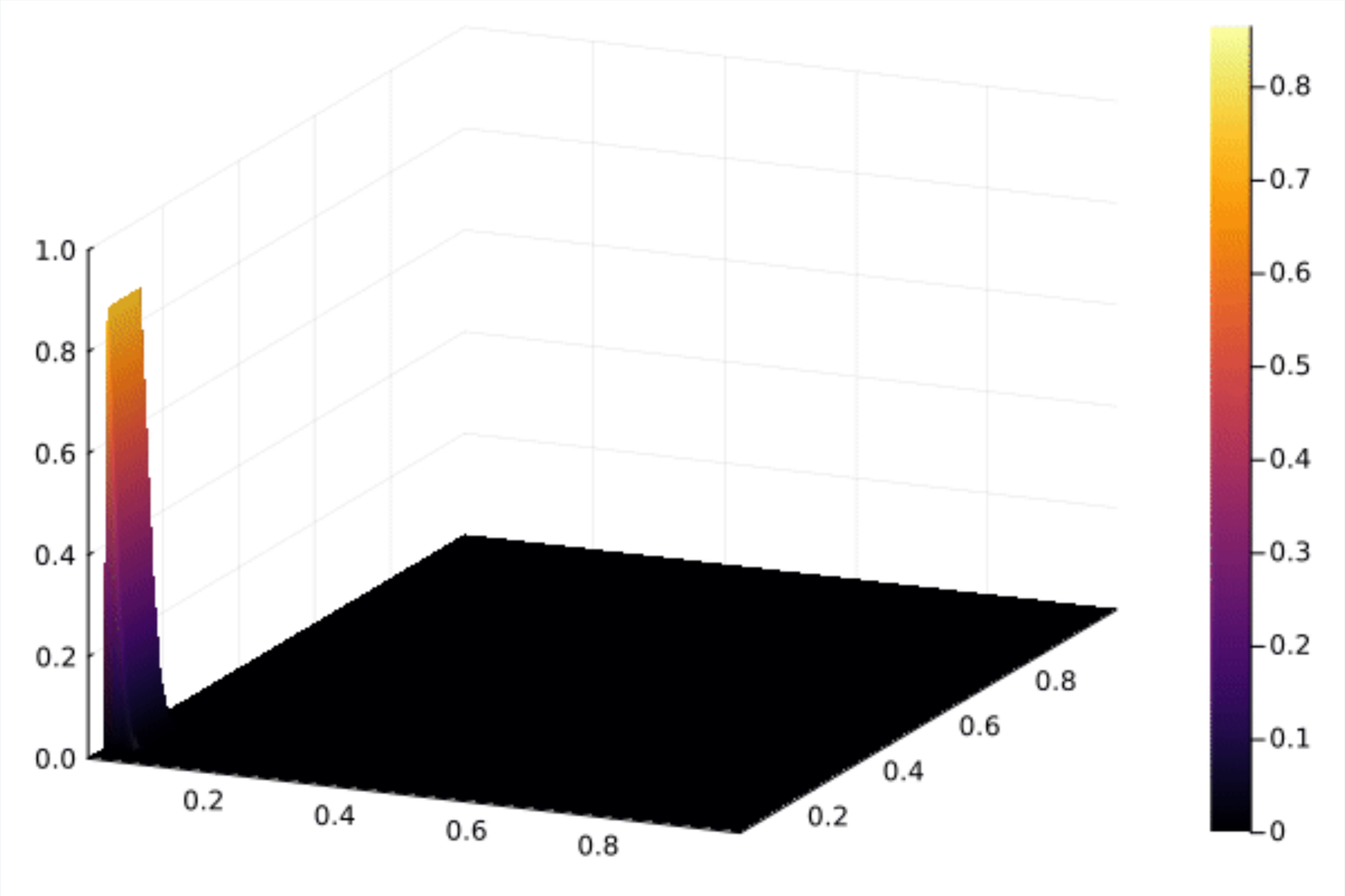


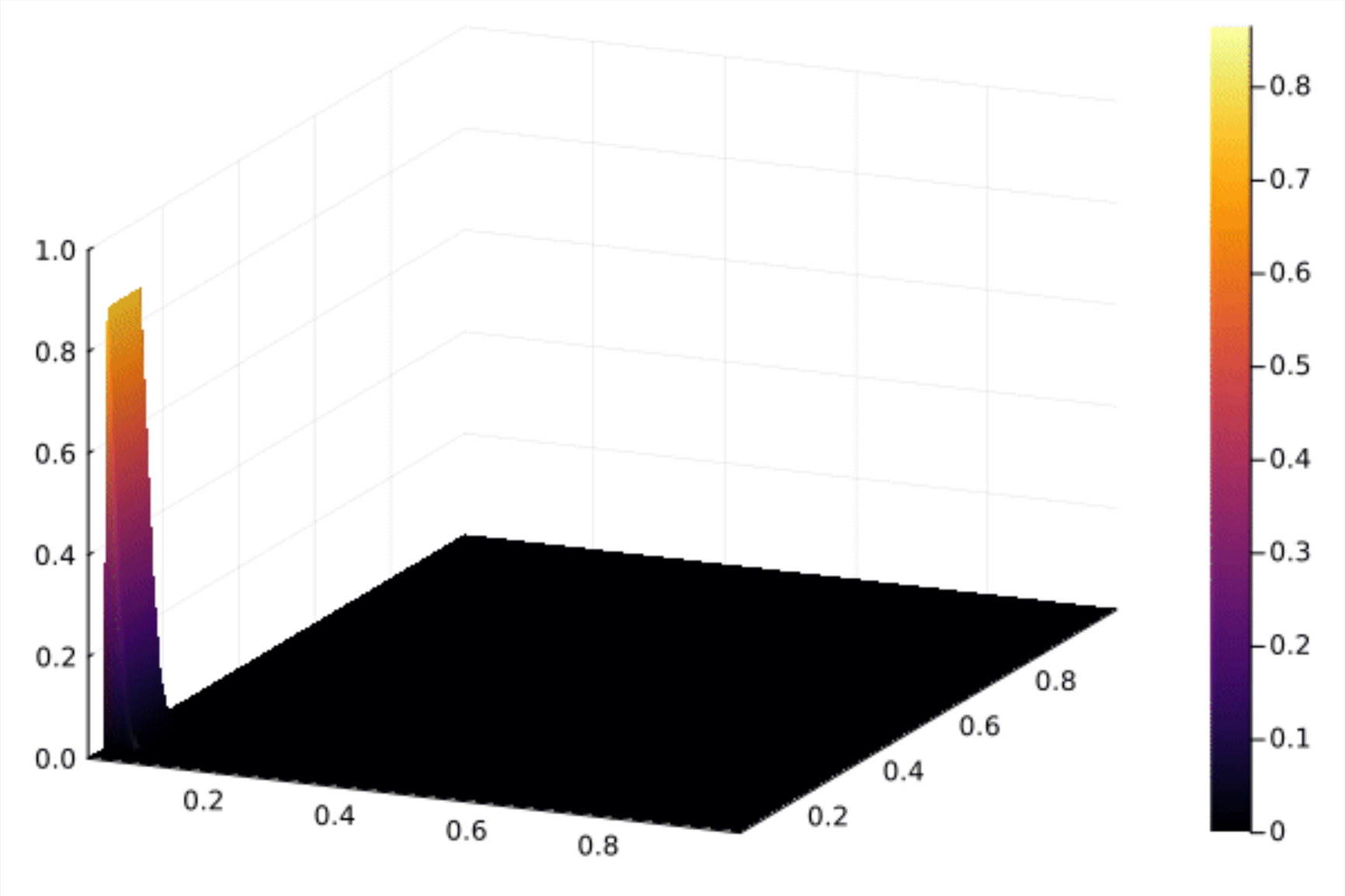
FlowFPX

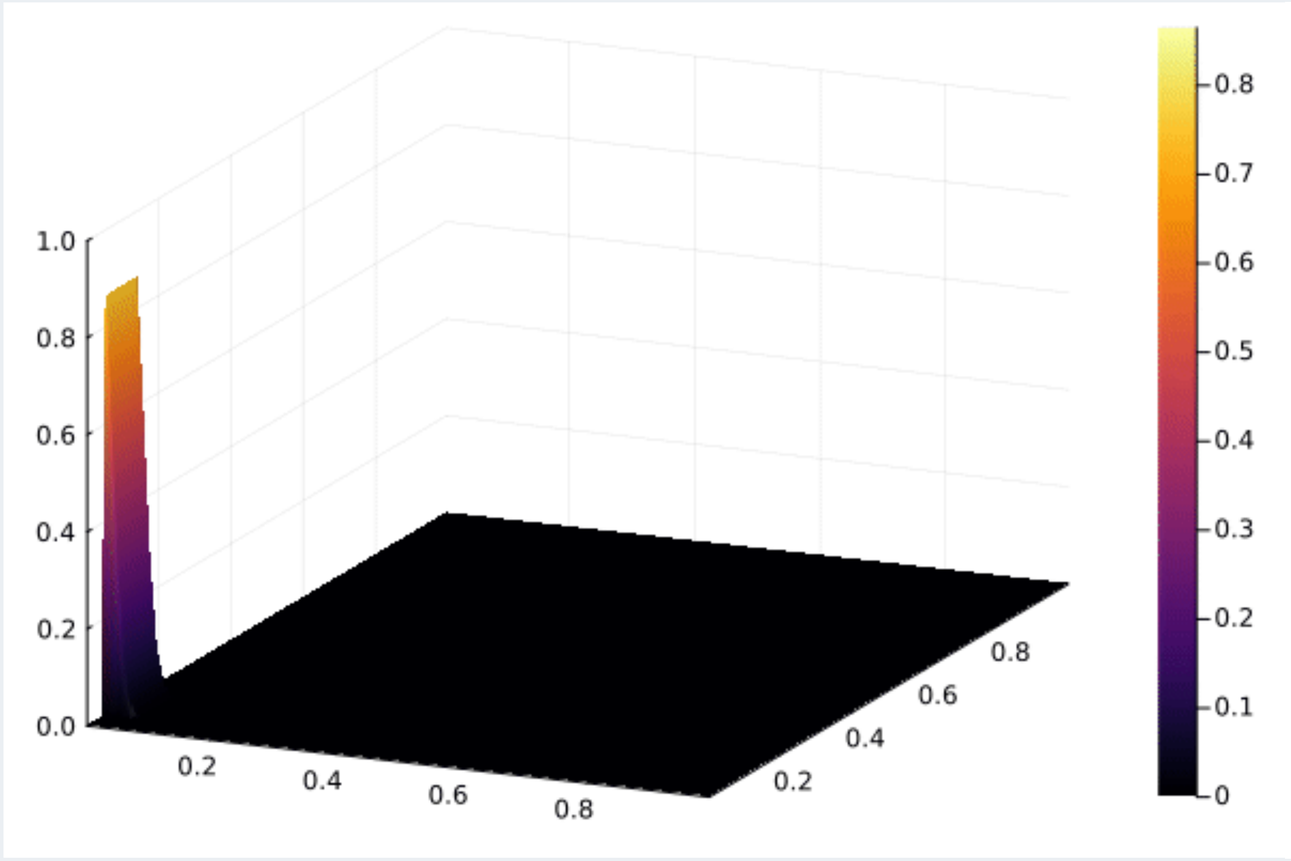
Nimble tools for debugging floating-point exceptions

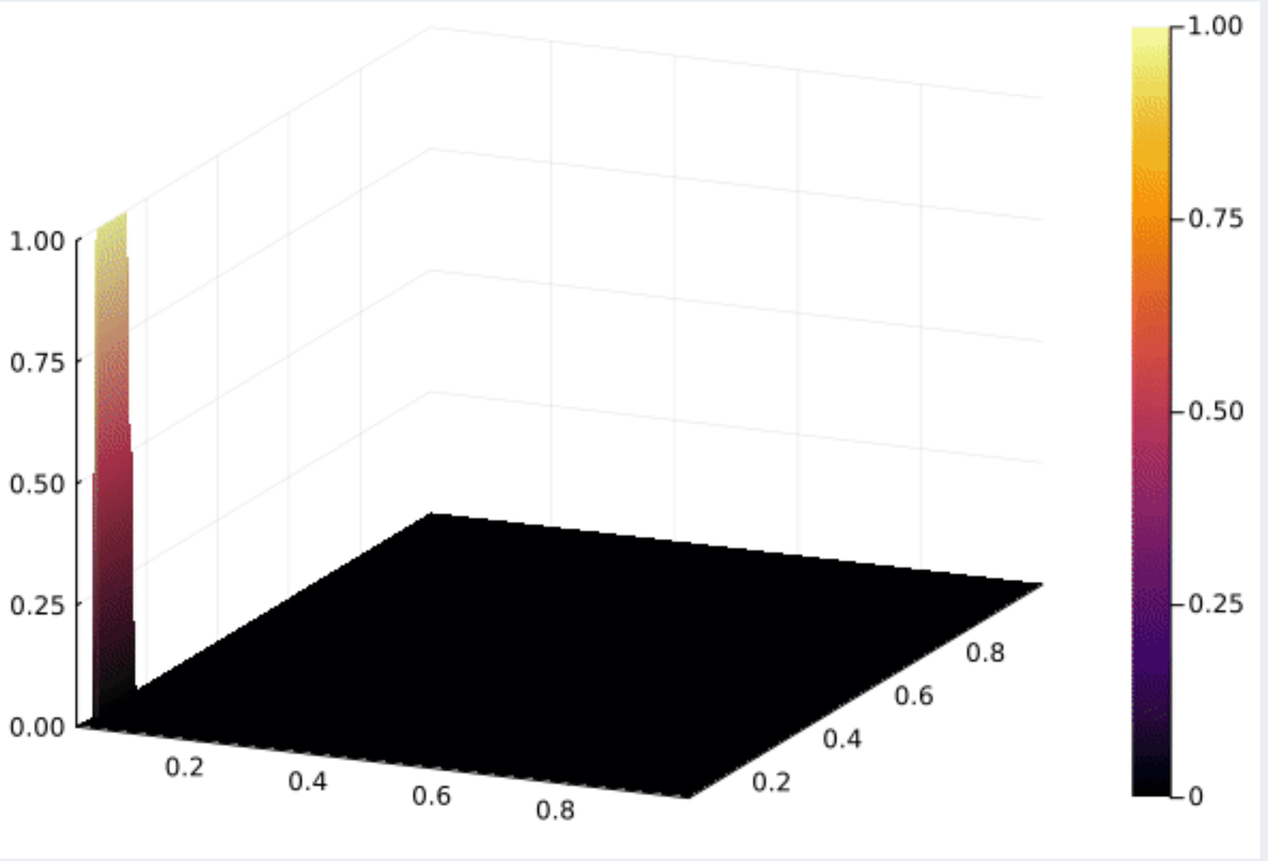
Taylor Allred, Ashton Wiersdorf, Xinyi Li, Ben Greenman, and Ganesh Gopalakrishnan

The Busy Scientist


1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
0.2
0.4
0.6
0.8
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
0.2
0.4
0.6
0.8
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
0.2
0.4
0.6
0.8
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

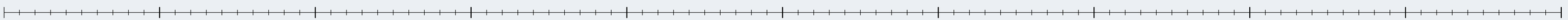

1.00
0.75
0.50
0.25
0.00
0.2
0.4
0.6
0.8
0.2
0.4
0.6
0.8
1.00
0.75
0.50
0.25
0

What happened?

NaN

IEEE 754 Floating-Point

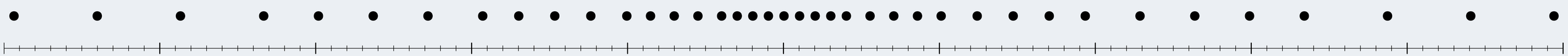
IEEE 754 Floating-Point



Real numbers

IEEE 754 Floating-Point

Floating-point numbers



Real numbers

IEEE 754 Floating-Point

An arithmetic exception arises when [there is] no result that would be acceptable universally.

—*William Kahan*

IEEE 754 Floating-Point

IEEE 754 Floating-Point

Not-a-Number (NaN)

IEEE 754 Floating-Point

Not-a-Number (NaN)

Infinity

IEEE 754 Floating-Point

Not-a-Number (NaN)

Infinity

Subnormal (underflow)

IEEE 754 Floating-Point

Not-a-Number (NaN)

Infinity

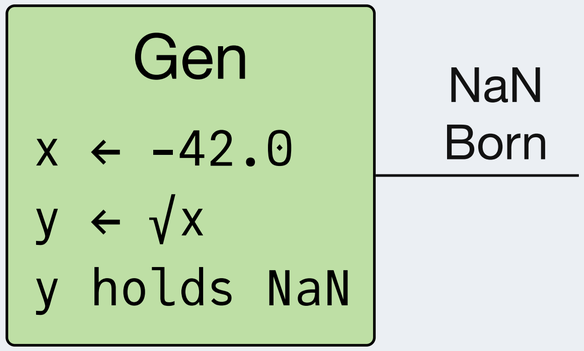
Subnormal (underflow)

```
  0 / 0      ⇛ NaN
  0 * Inf    ⇛ NaN
Inf - Inf    ⇛ NaN
```

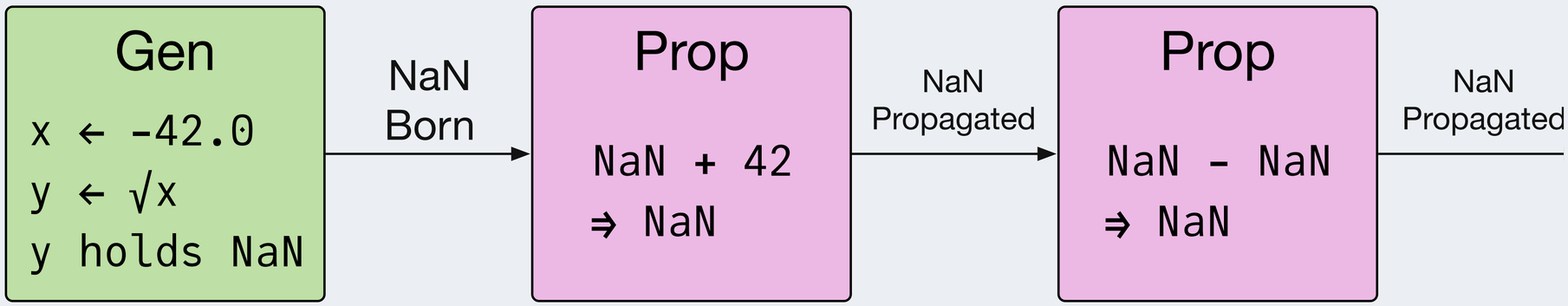
Why not throw an error?

Lifetime of an exceptional value

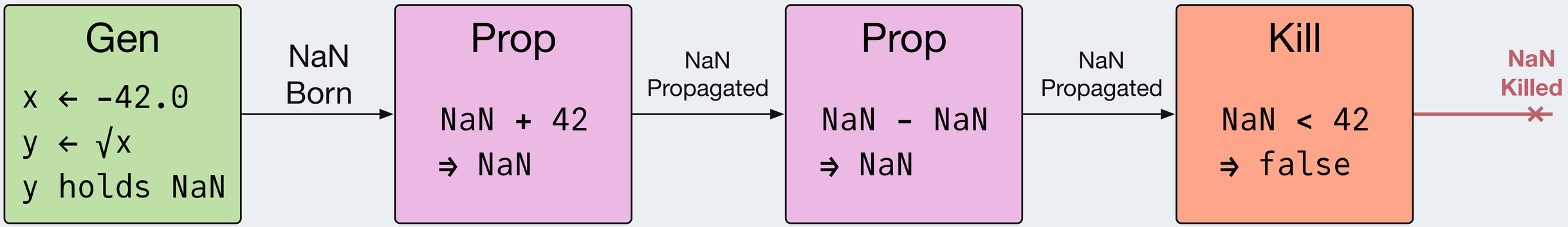
Lifetime of an exceptional value

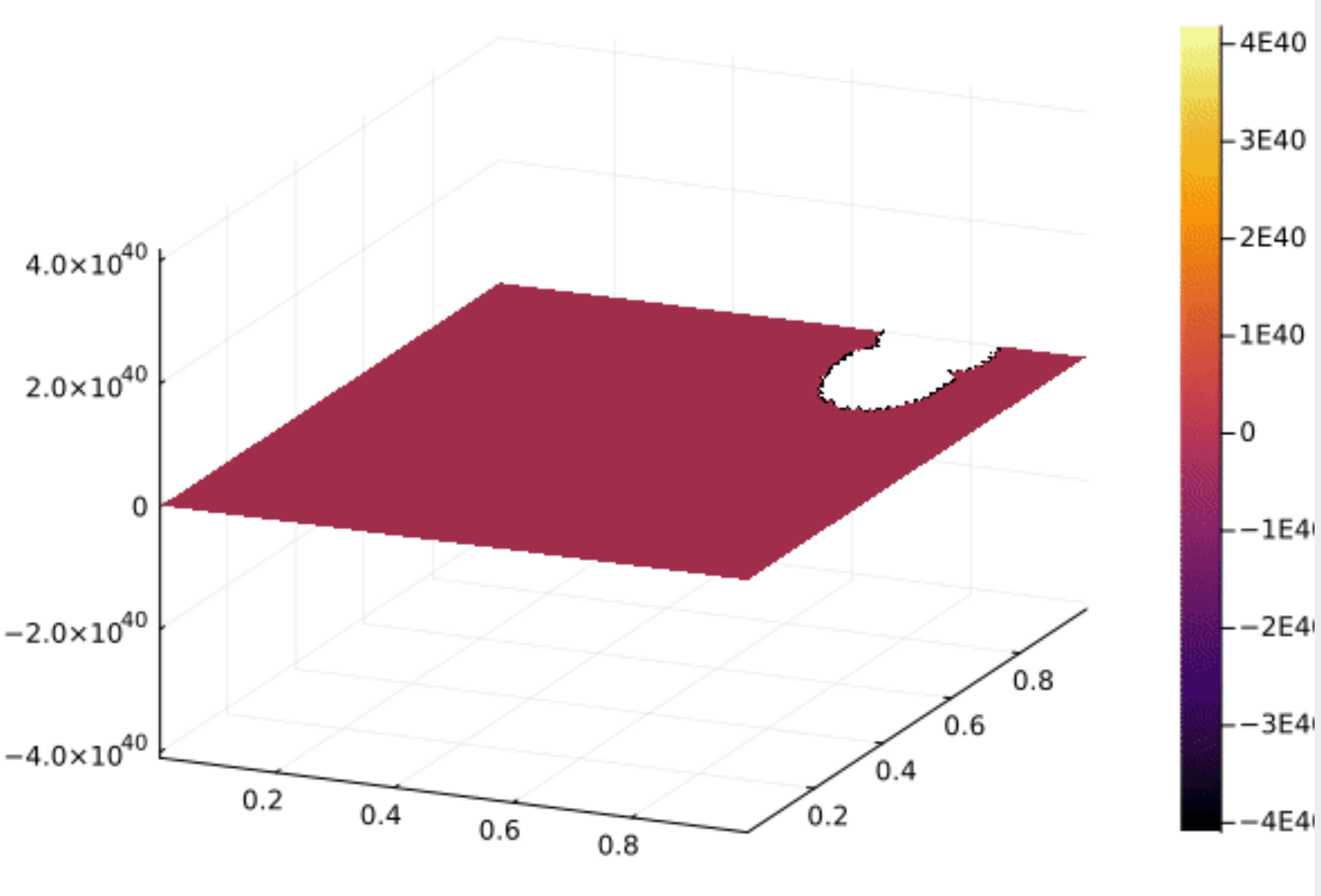


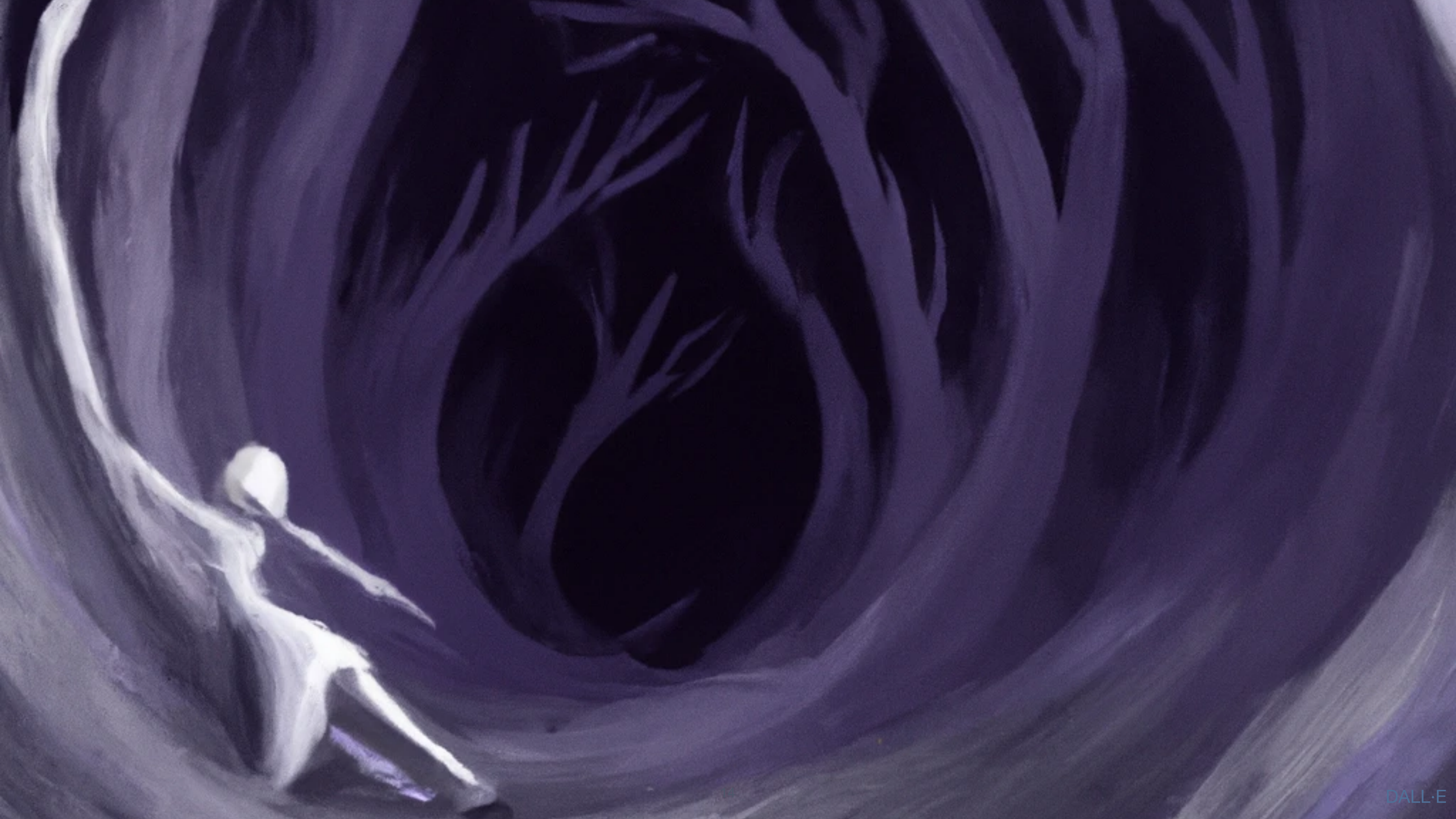
Lifetime of an exceptional value



Lifetime of an exceptional value




4.0×10^40
2.0×10^40
0
−2.0×10^40
−4.0×10^40
0.2
0.4
0.6
0.8
0.2
0.4
0.6
0.8
4E40
3E40
2E40
1E40
0
−1E4
−2E4
−3E4
−4E4


DALL·E

DALL·E

Photo by Elisa Ventur on Unsplash

100s of issues in Julia repos on GitHub mentioning “NaN”

FlowFPX to the rescue

RxInfer.jl

Bayesian Network Inference Library

https://github.com/biaslab/RxInfer.jl/issues/116

Now it is impossible to trace back the origin of the very first NaN without perform[ing] *a lot of manual work*. This limits the ability to debug the code and to prevent these NaNs in the first place.

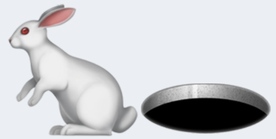
RxInfer.jl #116 after a day using FlowFPX's FloatTracker

Great package! I already found the location where a NaN was produced.

FlowFPX

FlowFPX Solutions

Tracking down exceptional values



Harden exception-vulnerable code

FlowFPX Technologies

FlowFPX Technologies

Julia’s type-based dispatch

FlowFPX Technologies

Julia’s type-based dispatch

Fuzzing

FlowFPX Technologies

Julia’s type-based dispatch

Fuzzing

Stack graphs

FlowFPX Technologies

Julia’s type-based dispatch

Fuzzing

Stack graphs

Flow graph diffing

FlowFPX Components

FloatTracker

Logs and tracks exceptions

Fuzzes for vulnerabilities

CSTG

Generates visual summaries from FloatTracker

General tool to visualize stack traces

`https://doi.org/10.1109/MCSE.2014.11`

FlowFPX Components

FloatTracker

Logs and tracks exceptions

Fuzzes for vulnerabilities

CSTG

Generates visual summaries from FloatTracker

General tool to visualize stack traces

https://doi.org/10.1109/MCSE.2014.11

More to come…

Demo: maximum

Demmel et al. Correctness@SC 2022
arXiv:2207.09281

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 0.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 0.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 0.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
➤ julia maximum.jl
Result: 4.0
➤
```

```
result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
➤ julia maximum.jl
Result: 4.0
➤
```

WAT

Let’s call FloatTracker 

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
using FloatTracker

function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
using FloatTracker

function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum(TrackedFloat32.([1, 5, 4, NaN, 4]))
println("Result: $result”)
```

```
using FloatTracker

function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum(TrackedFloat32.([1, 5, 4, NaN, 4]))
println("Result: $result”)
ft_flush_logs()
```

max_gen_logs.txt

max_prop_logs.txt

max_kill_logs.txt

Arguments	Output	
No NaN	Nan	gen
NaN	Nan	prop
NaN	No Nan	kill

max_gen_logs.txt

max_prop_logs.txt

max_kill_logs.txt

max_kill_logs.txt

```
[NaN] check_error([NaN, 4.0])
<                   at FloatTracker/src/TrackedFloat.jl:214
maximum             at examples/max_min_example.jl:0
top-level scope at examples/max_min_example.jl:15
```

max_kill_logs.txt

```
NaN < 4.0  ⇒  false
```

```
function maximum(lst)
  max_seen = 0.0
  for x in lst
    if ! (x < max_seen)
      max_seen = x          # swap if new val greater
    end
  end
  max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 1.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 1.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 1.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 5.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 5.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 5.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 5.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 5.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = NaN
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = NaN
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = NaN
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

```
function maximum(lst)
    max_seen = 4.0
    for x in lst
        if ! (x < max_seen)
            max_seen = x          # swap if new val greater
        end
    end
    max_seen
end

result = maximum([1, 5, 4, NaN, 4])
println("Result: $result")
```

How to use FloatTracker

How to use FloatTracker

Require FloatTracker

How to use FloatTracker

Require FloatTracker

Wrap inputs with `TrackedFloat` types

How to use FloatTracker

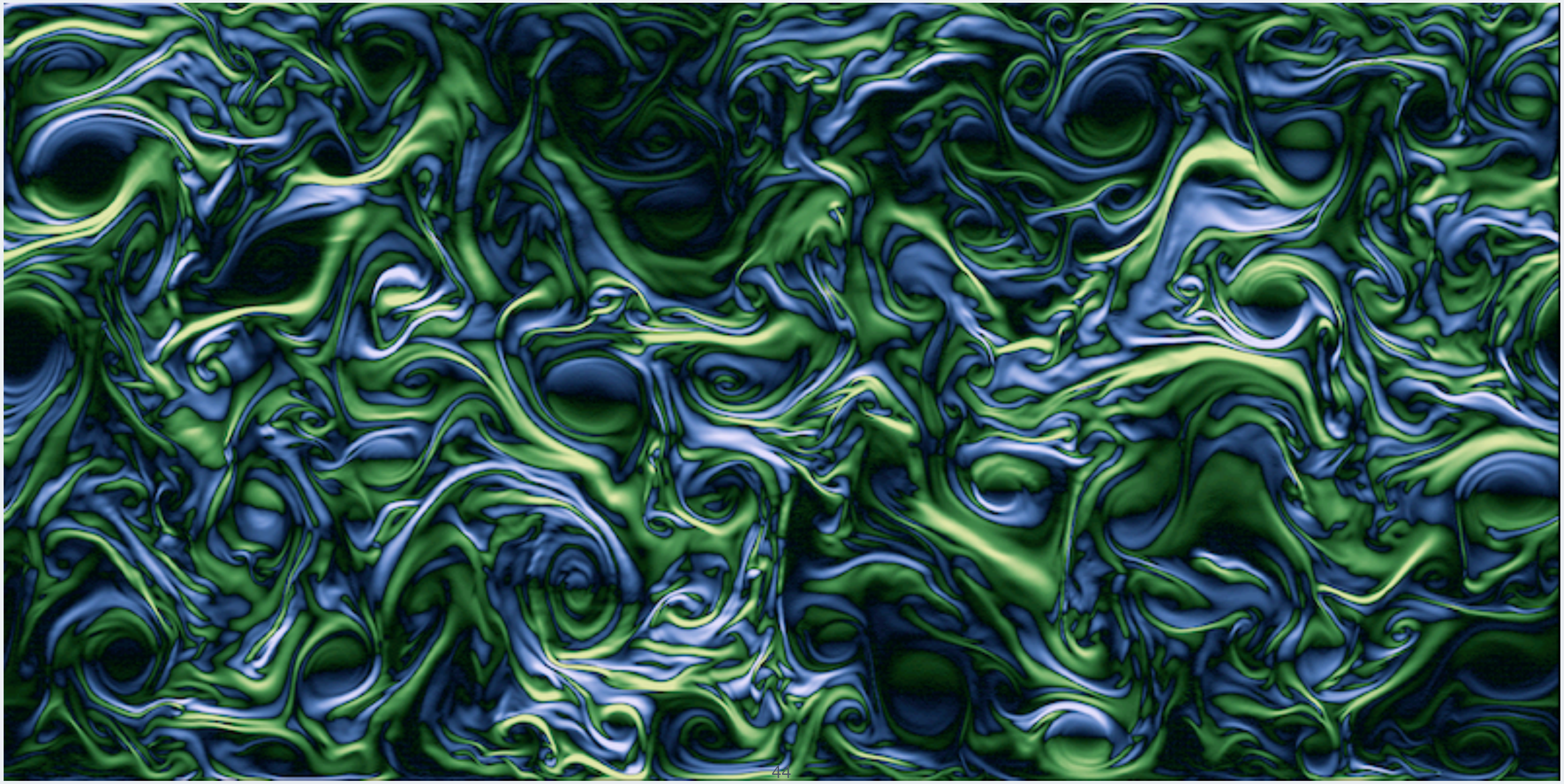
Require FloatTracker

Wrap inputs with `TrackedFloat` types

Flush any buffered logs

ShallowWaters.jl

ShallowWaters.jl



Milan Klöwer

ShallowWaters.jl

CFL Trade-offs

ShallowWaters.jl

CFL Trade-offs

More accuracy

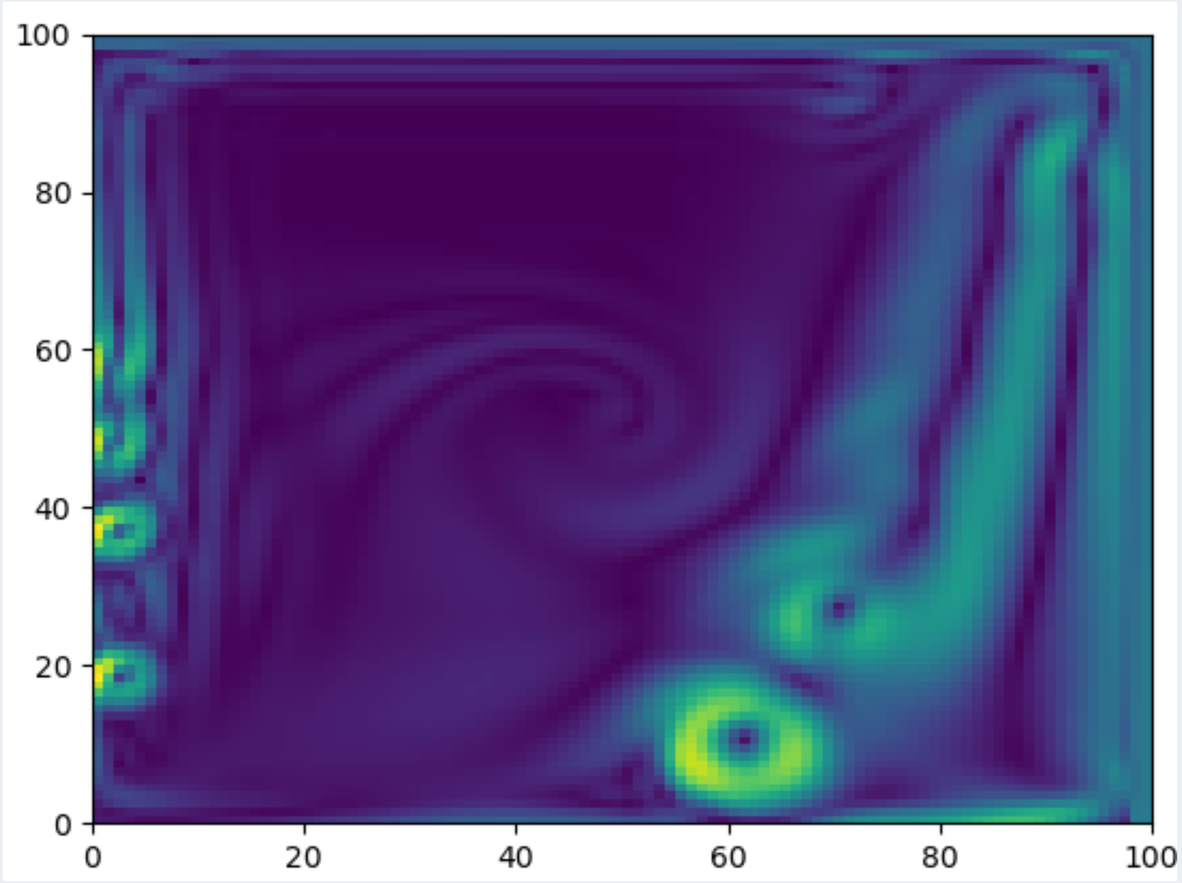
Slower renders

Faster renders

More instability

ShallowWaters.jl

Small ← CFL → Big



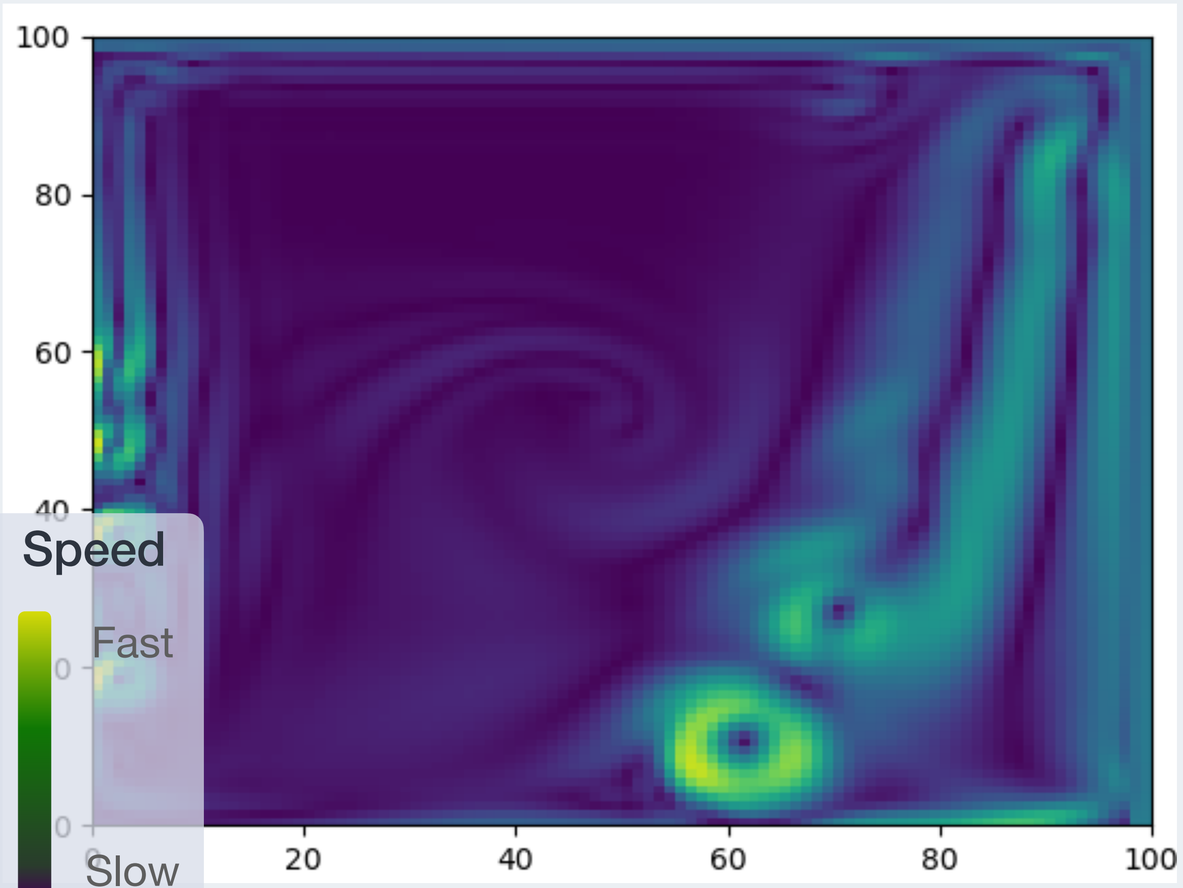
Speed

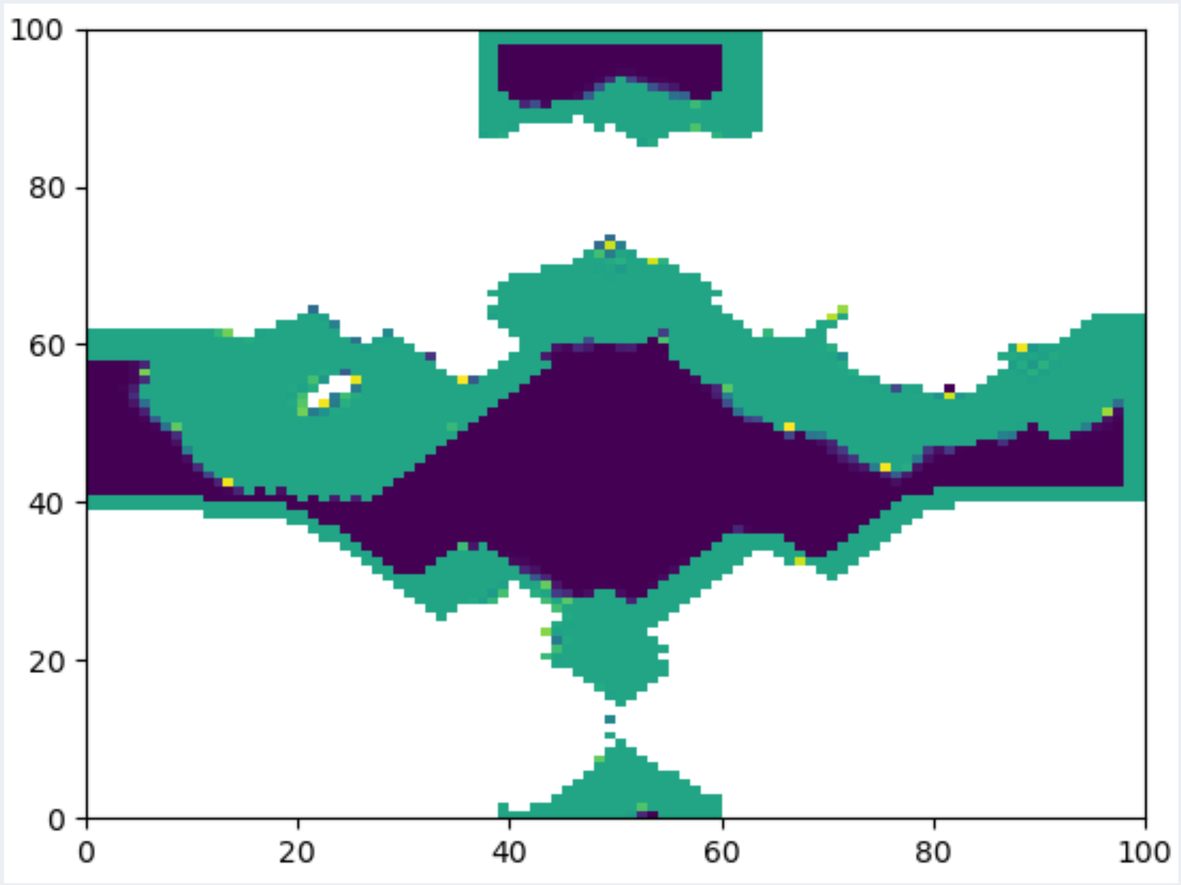
Fast

Slow

ShallowWaters.jl

Small ← CFL → Big





Let’s call FloatTracker 

ShallowWaters.jl

ShallowWaters.jl

sw_gen_logs.txt

sw_prop_logs.txt

sw_kill_logs.txt

ShallowWaters.jl

`sw_gen_logs.txt`

ShallowWaters.jl

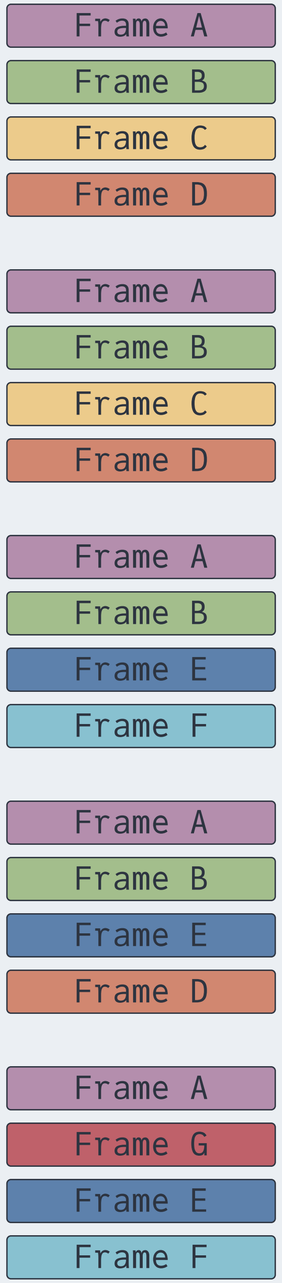
sw_gen_logs.txt 3.1 MB

CSTG summarizes flows

CSTG summarizes flows

Builds off of STAT from LLNL

`https://github.com/LLNL/STAT`


Frame A
Frame B
Frame C
Frame D
Frame A
Frame B
Frame C
Frame D
Frame A
Frame B
Frame E
Frame F
Frame A
Frame B
Frame E
Frame D
Frame A
Frame G
Frame E
Frame F

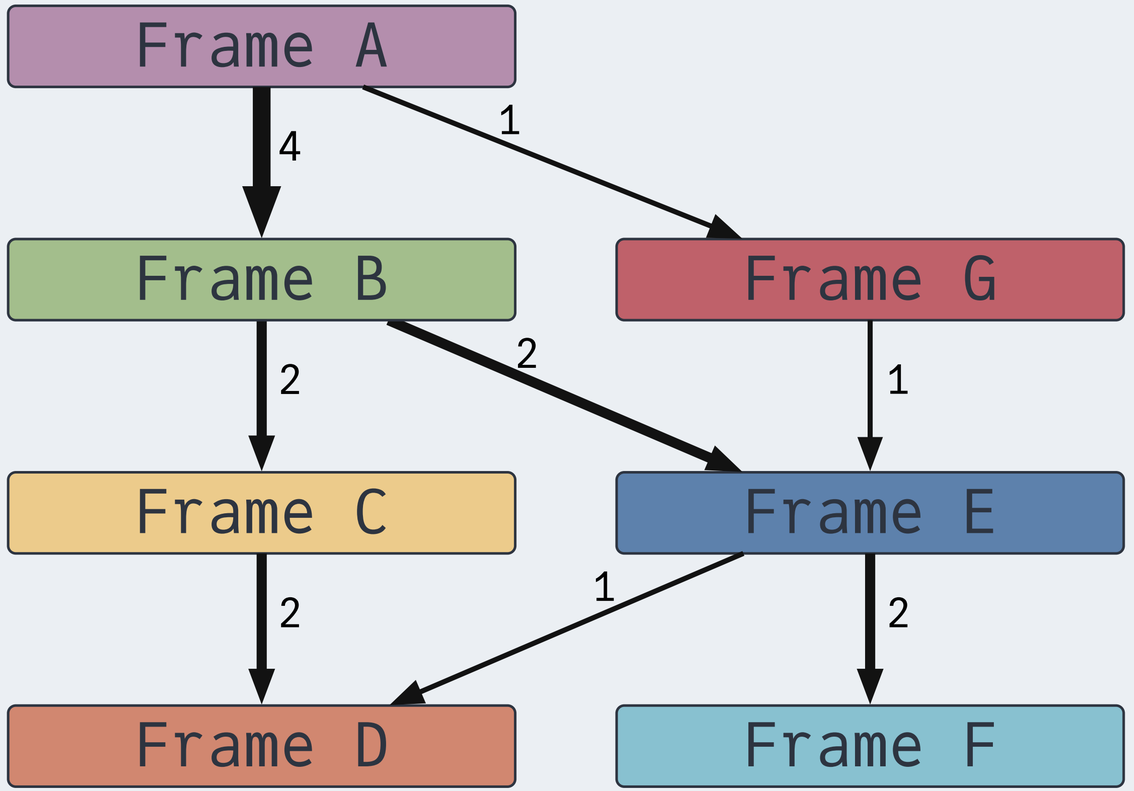
Frame A
Frame B
Frame C
Frame D

Frame A
Frame B
Frame C
Frame D

Frame A
Frame B
Frame E
Frame F

Frame A
Frame B
Frame E
Frame D

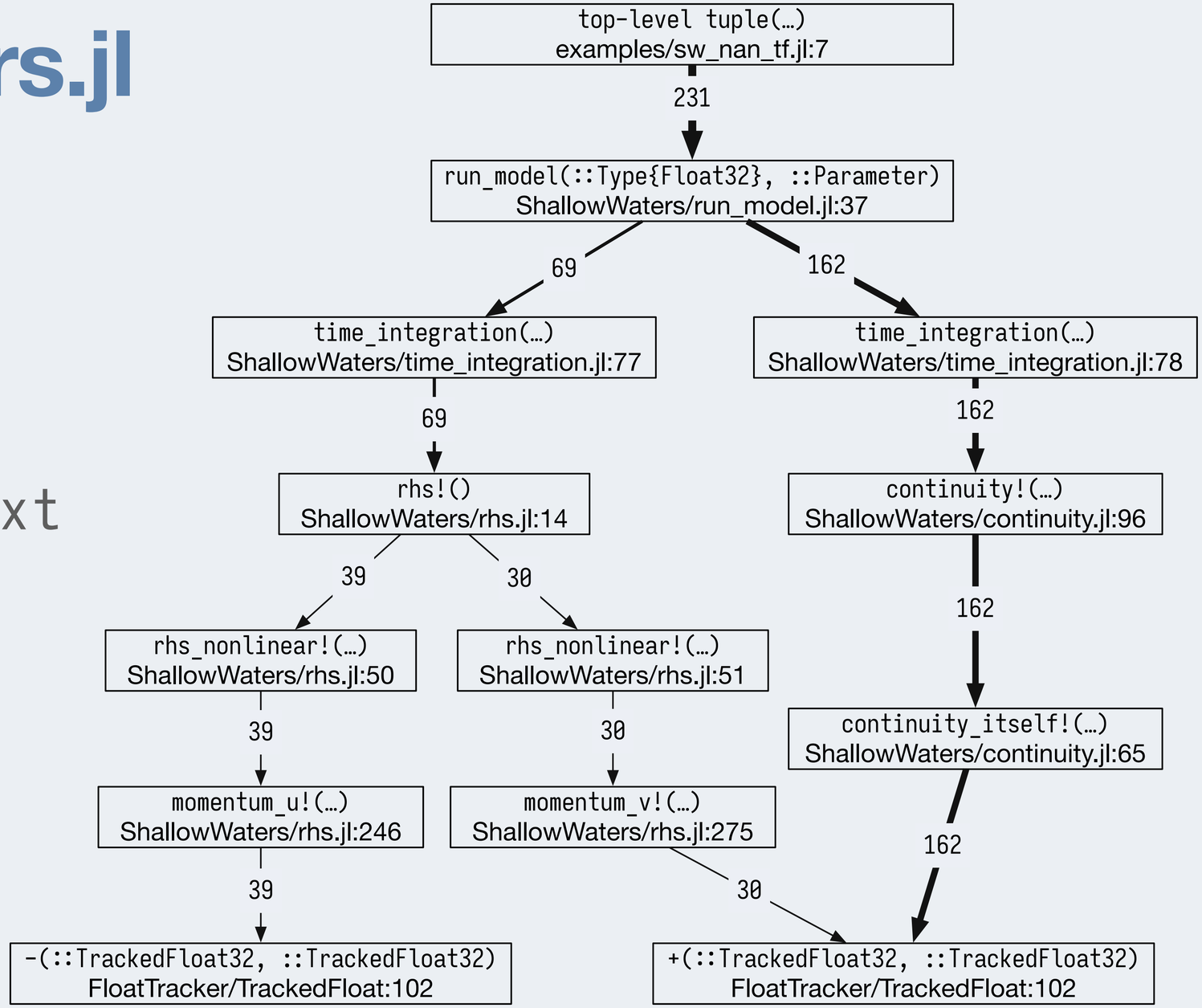
Frame A
Frame G
Frame E
Frame F



ShallowWaters.jl

CSTG

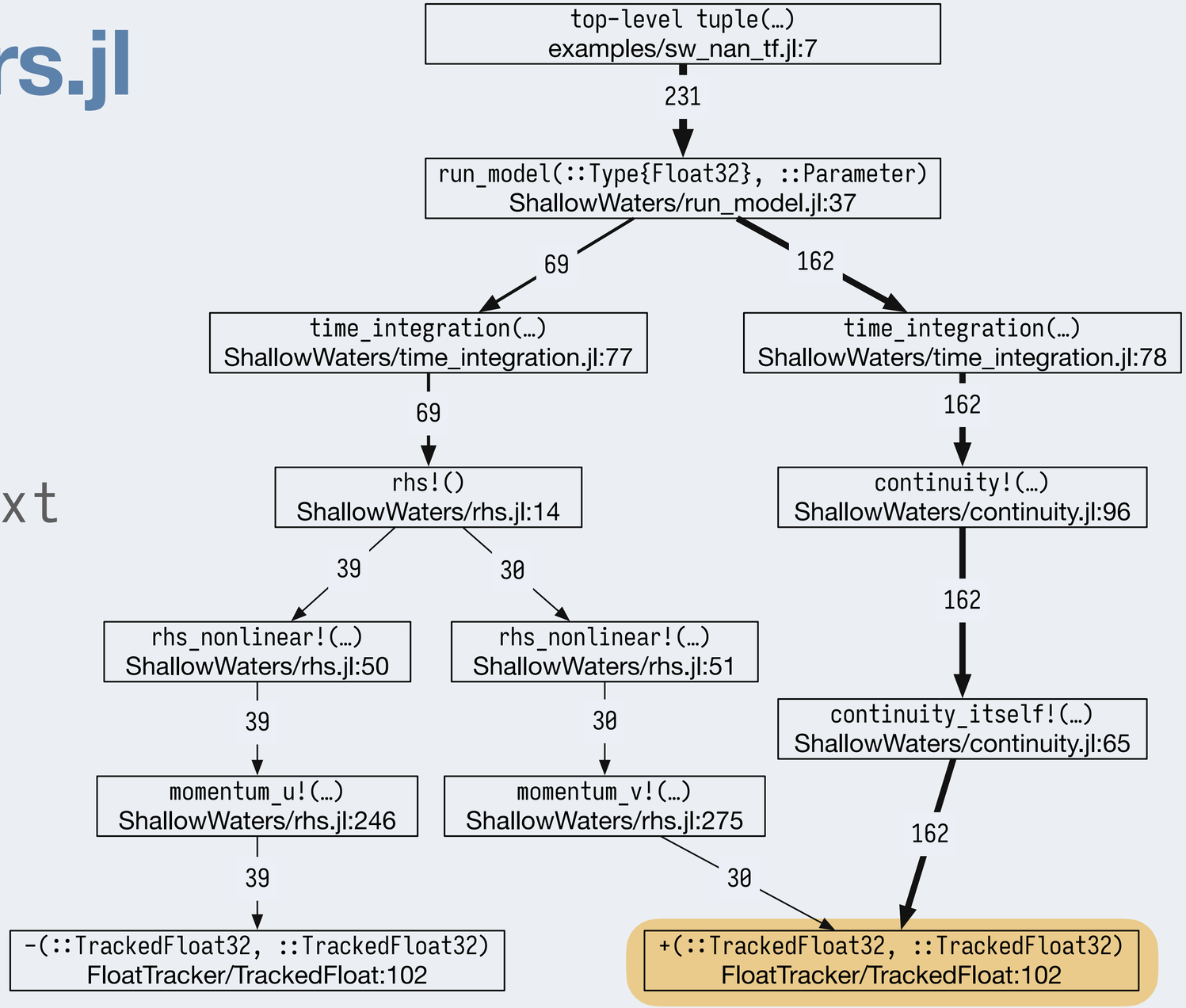
sw_gen_logs.txt



ShallowWaters.jl

CSTG

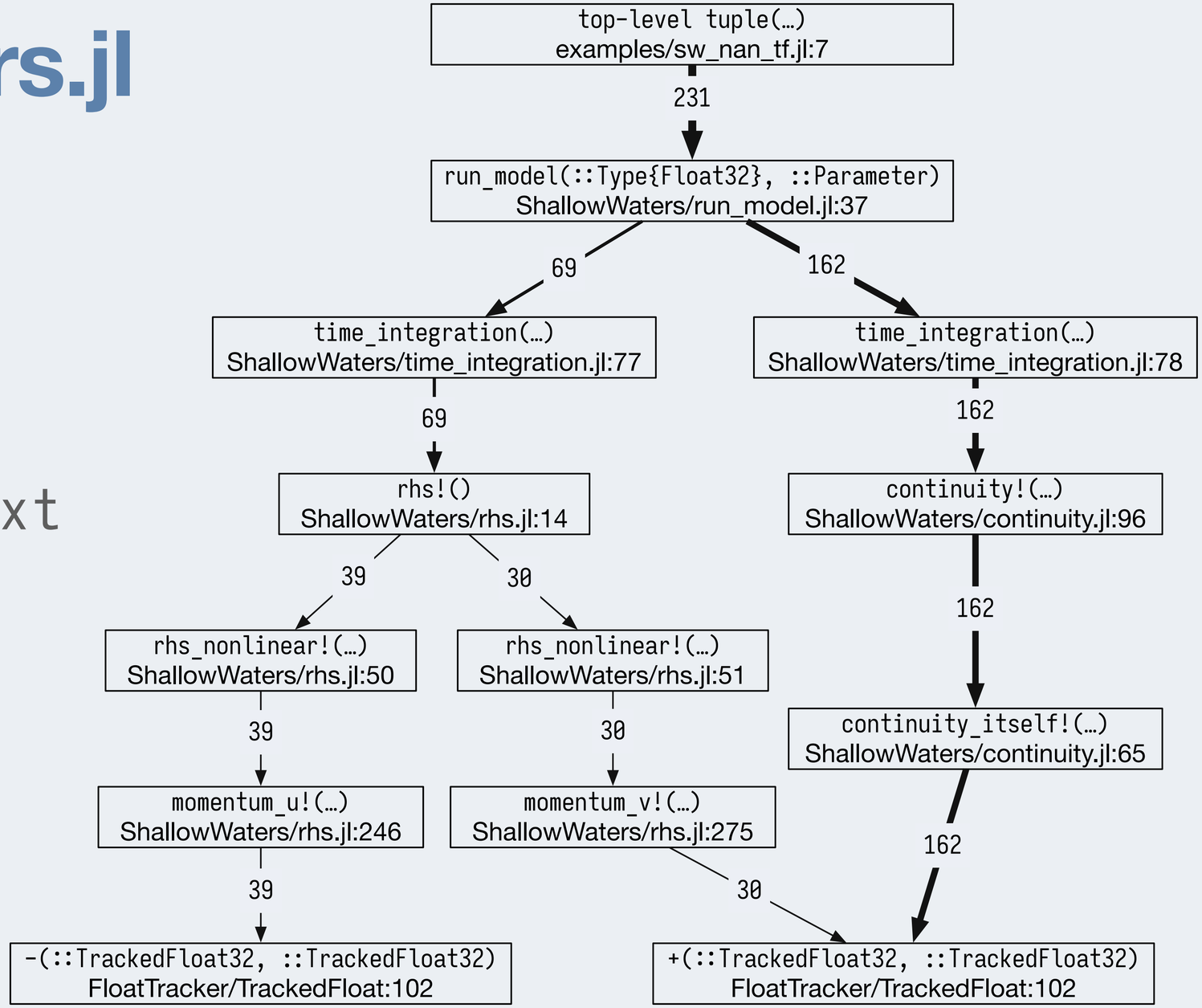
sw_gen_logs.txt



ShallowWaters.jl

CSTG

sw_gen_logs.txt



ShallowWaters.jl

sw_gen_logs.txt

```
[NaN] check_error(Any[Inf32, -Inf32])  at FloatTracker/src/TrackedFloat.jl:11
+(::TrackedFloat32, ::TrackedFloat32)  at FloatTracker/src/TrackedFloat.jl:103
momentum_v!(…)                         at ShallowWaters/src/rhs.jl:275
rhs_nonlinear!(…)                      at ShallowWaters/src/rhs.jl:51
rhs!()                                 at ShallowWaters/src/rhs.jl:14
time_integration(…)                    at ShallowWaters/src/time_integration.jl:77
run_model(…)                           at ShallowWaters/src/run_model.jl:37
#run_model#57()                        at ShallowWaters/src/run_model.jl:17
run_model##kw()                        at ShallowWaters/src/run_model.jl:12
run_model##kw(run_model)               at ShallowWaters/src/run_model.jl:12
top-level scopeCore.tuple(…)           at FTExamples/examples/sw_nan_tf.jl:7
```

ShallowWaters.jl

sw_gen_logs.txt

```
[NaN] check_error(Any[Inf32, -Inf32])  at FloatTracker/src/TrackedFloat.jl:11
+(::TrackedFloat32, ::TrackedFloat32)  at FloatTracker/src/TrackedFloat.jl:103
momentum_v!(…)                         at ShallowWaters/src/rhs.jl:275
rhs_nonlinear!(…)                      at ShallowWaters/src/rhs.jl:51
rhs!()                                 at ShallowWaters/src/rhs.jl:14
time_integration(…)                    at ShallowWaters/src/time_integration.jl:77
run_model(…)                           at ShallowWaters/src/run_model.jl:37
#run_model#57()                        at ShallowWaters/src/run_model.jl:17
run_model##kw()                        at ShallowWaters/src/run_model.jl:12
run_model##kw(run_model)               at ShallowWaters/src/run_model.jl:12
top-level scopeCore.tuple(…)           at FTExamples/examples/sw_nan_tf.jl:7
```

ShallowWaters.jl

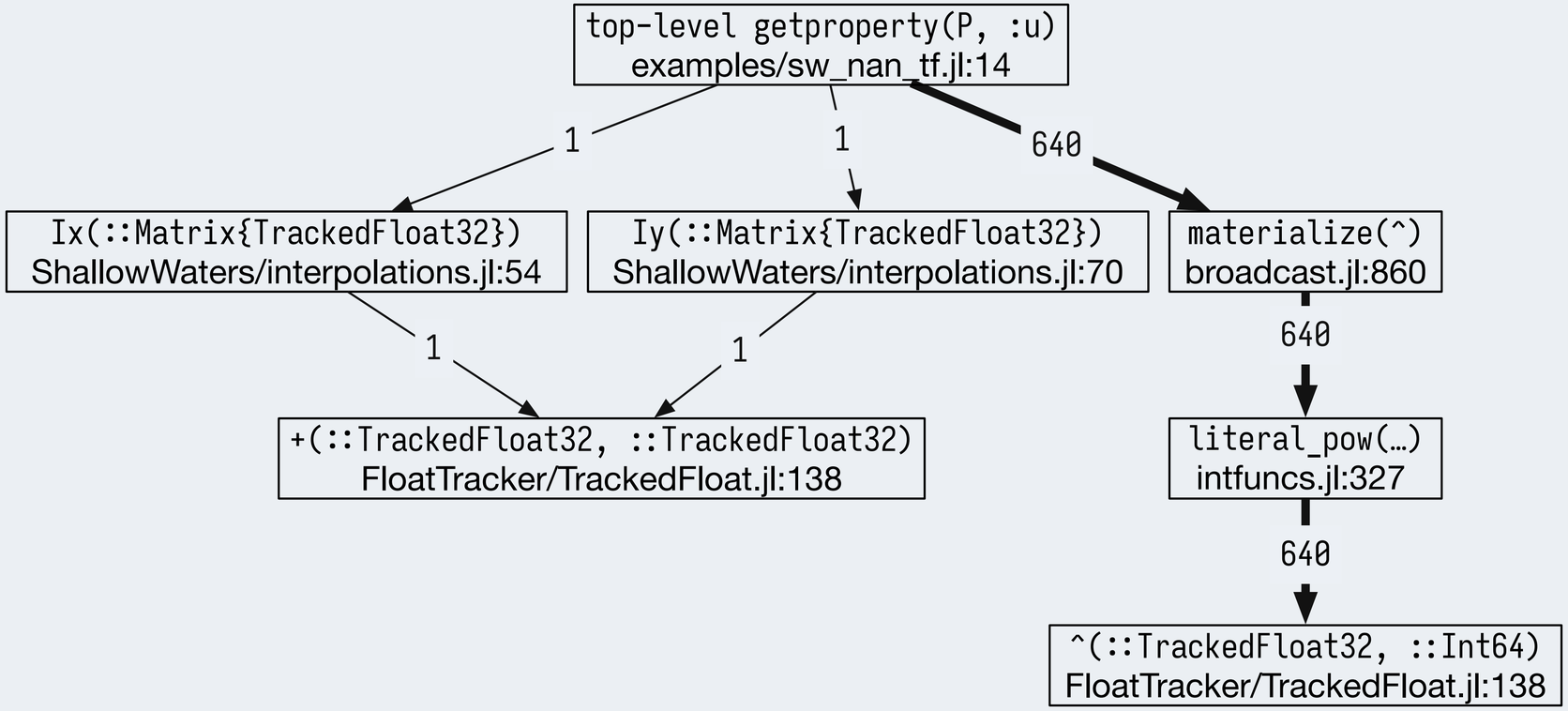
```
Inf32 + -Inf32 ⇒ NaN
```

Where is `Inf` coming from?

Let’s call FloatTracker 

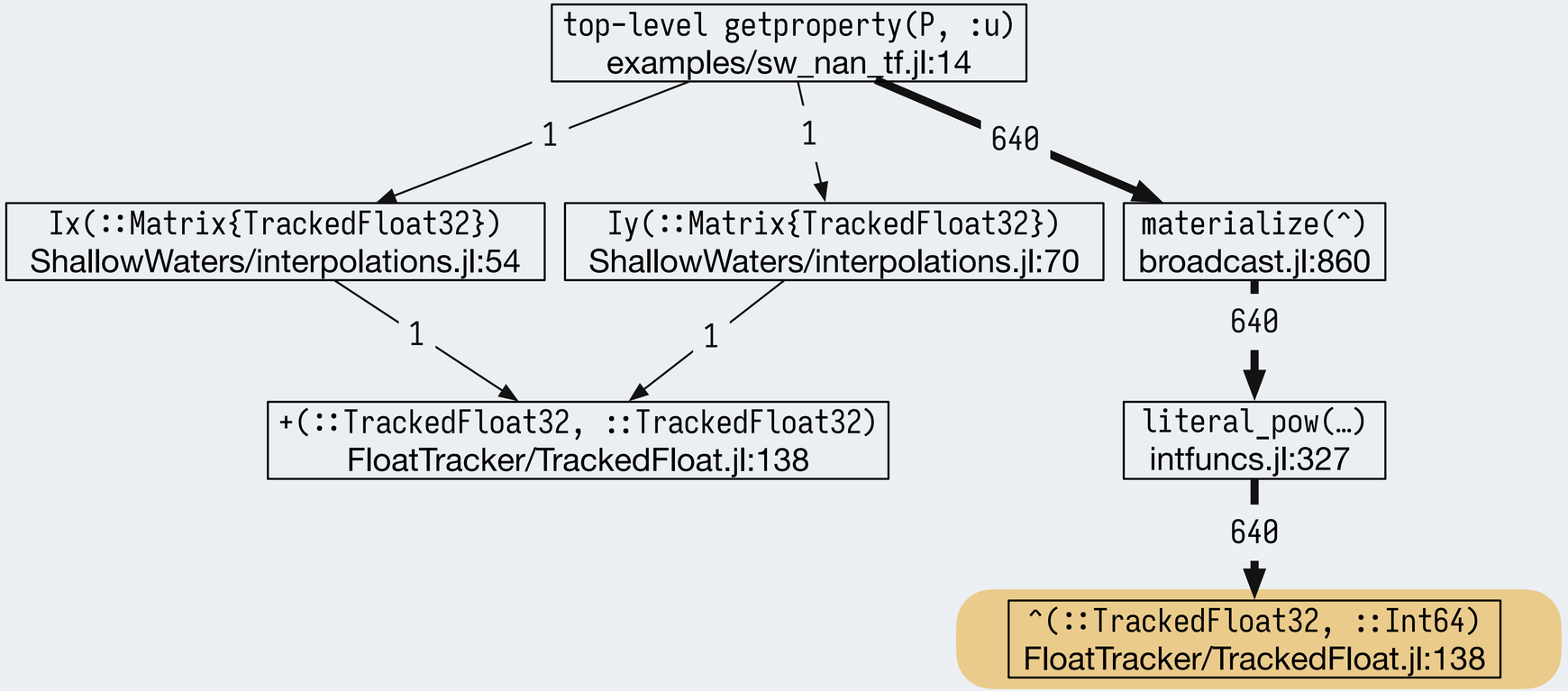
ShallowWaters.jl

CSTG



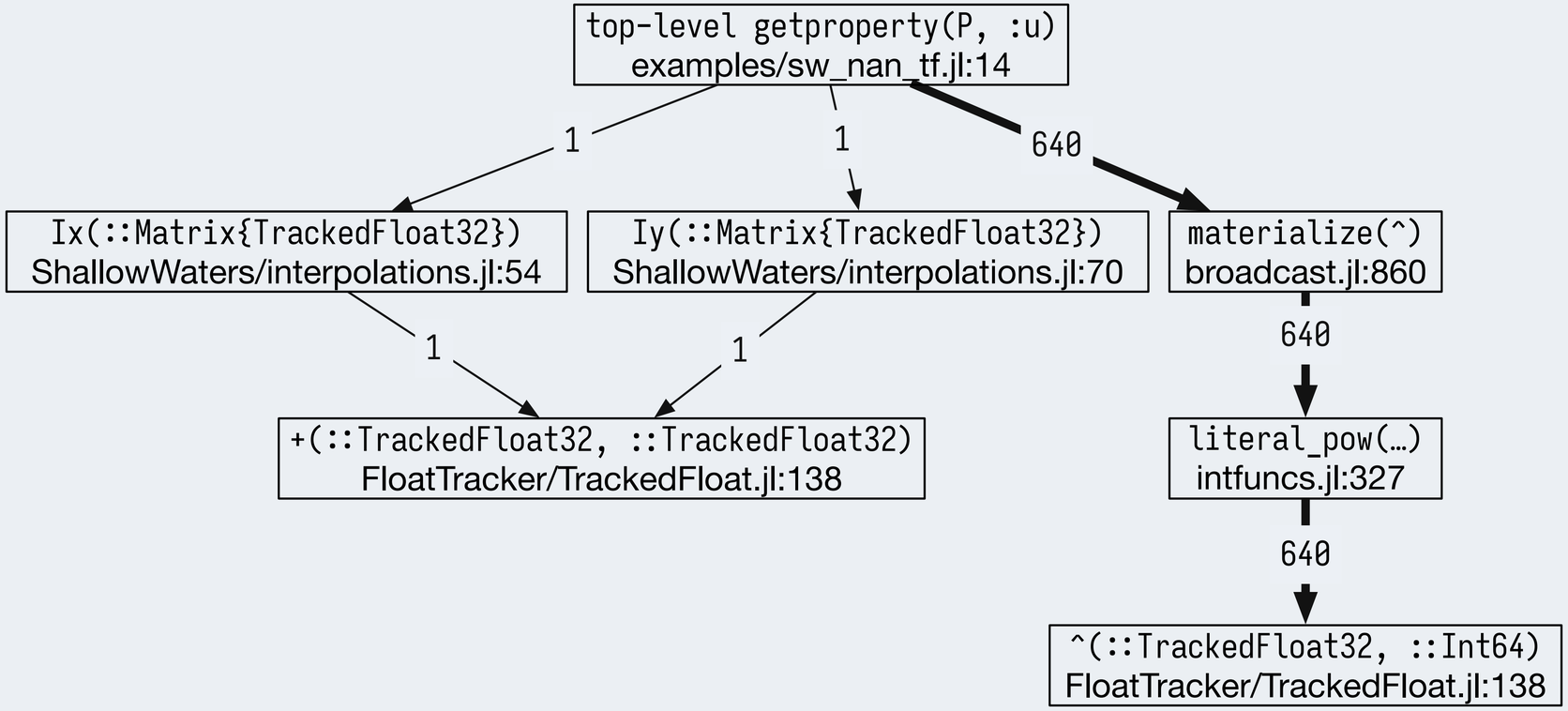
ShallowWaters.jl

CSTG



ShallowWaters.jl

CSTG



ShallowWaters.jl
CSTG

sw_gen_logs.txt

```
[Inf] check_error(Any[-1.5150702f31, 2])  at FloatTracker/src/TrackedFloat.jl:11
^(::TrackedFloat32, ::Int64)              at FloatTracker/src/TrackedFloat.jl:139
literal_pow()                             at ./intfuncs.jl:327
_broadcast_getindex_evalf()               at ./broadcast.jl:670
_broadcast_getindex()                     at ./broadcast.jl:643
getindex()                                at ./broadcast.jl:597
macro expansion()                         at ./broadcast.jl:961
macro expansion()                         at ./simdloop.jl:77
copyto!()                                 at ./broadcast.jl:960
copyto!()                                 at ./broadcast.jl:913
copy()                                    at ./broadcast.jl:885
top-level scopeBase.getproperty(P, :u)    at FTExamples/examples/sw_nan_tf.jl:14
```

ShallowWaters.jl

```
-1.5150702f31 ^ 2 ⇒ Inf
```

ShallowWaters.jl

We’re out of exception land! 🥳

Flow Summary

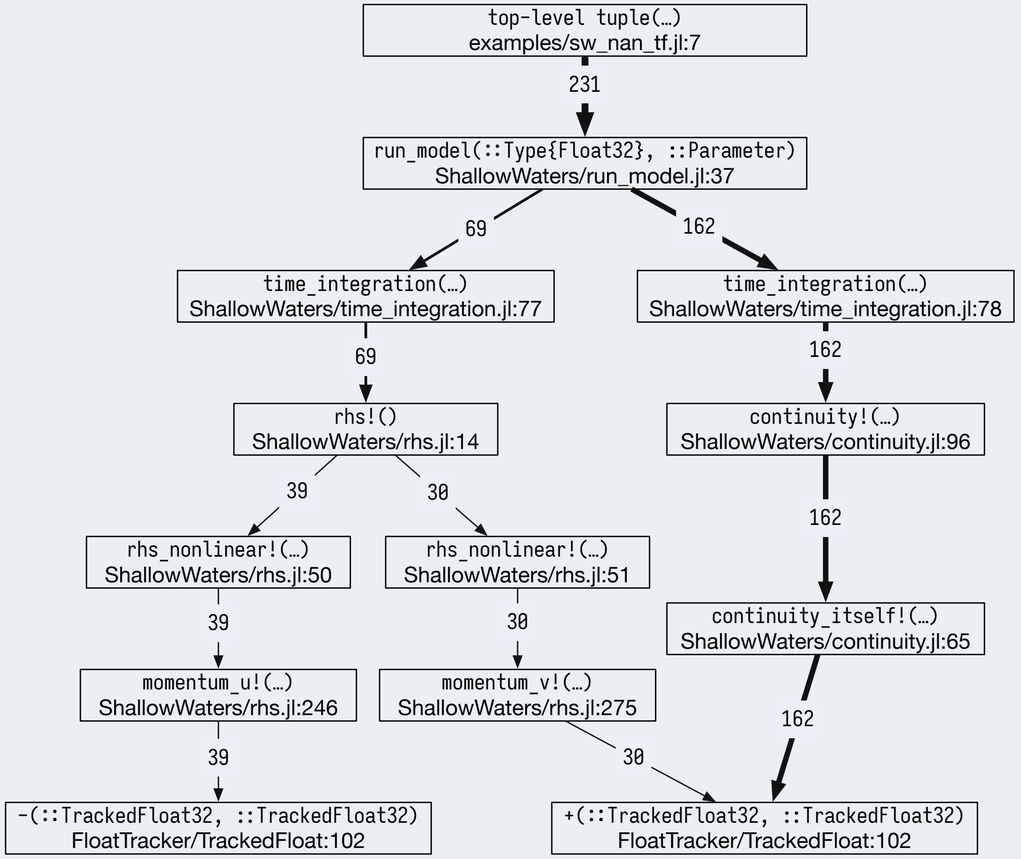
sw_gen_logs.txt

3.1 MB

Flow Summary

sw_gen_logs.txt

3.1 MB



How does the flow change over time?

ShallowWaters.jl

CSTG Diff

`sw_gen_logs.txt`

ShallowWaters.jl

CSTG Diff

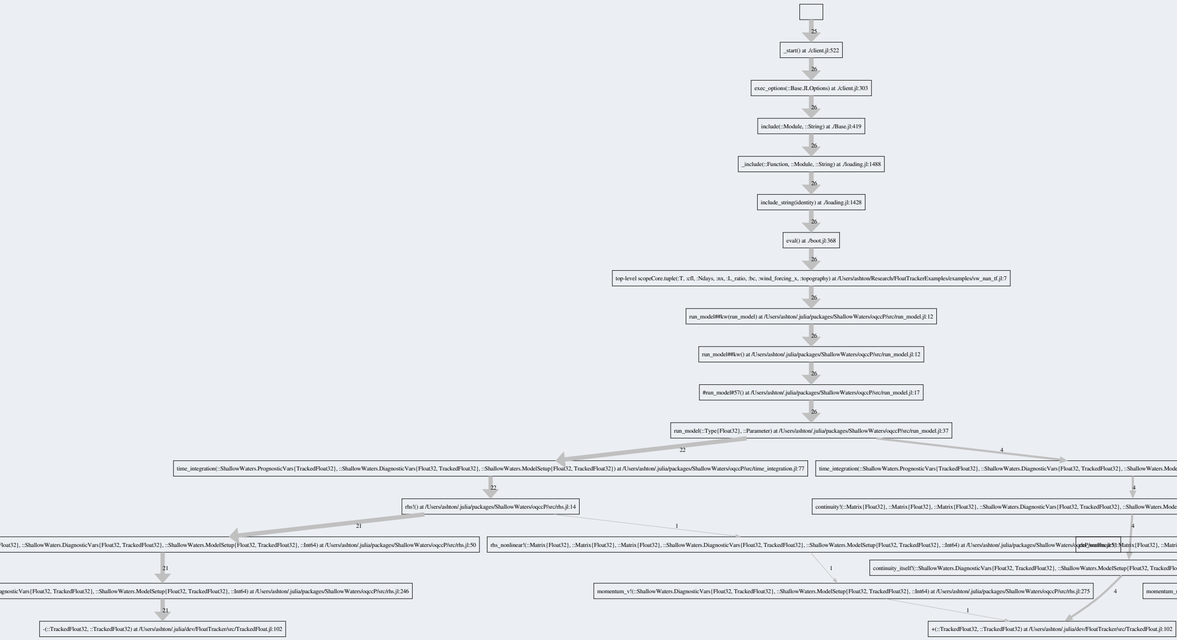
`sw_gen_logs_part1.txt`

`sw_gen_logs_part2.txt`

ShallowWaters.jl

CSTG Diff

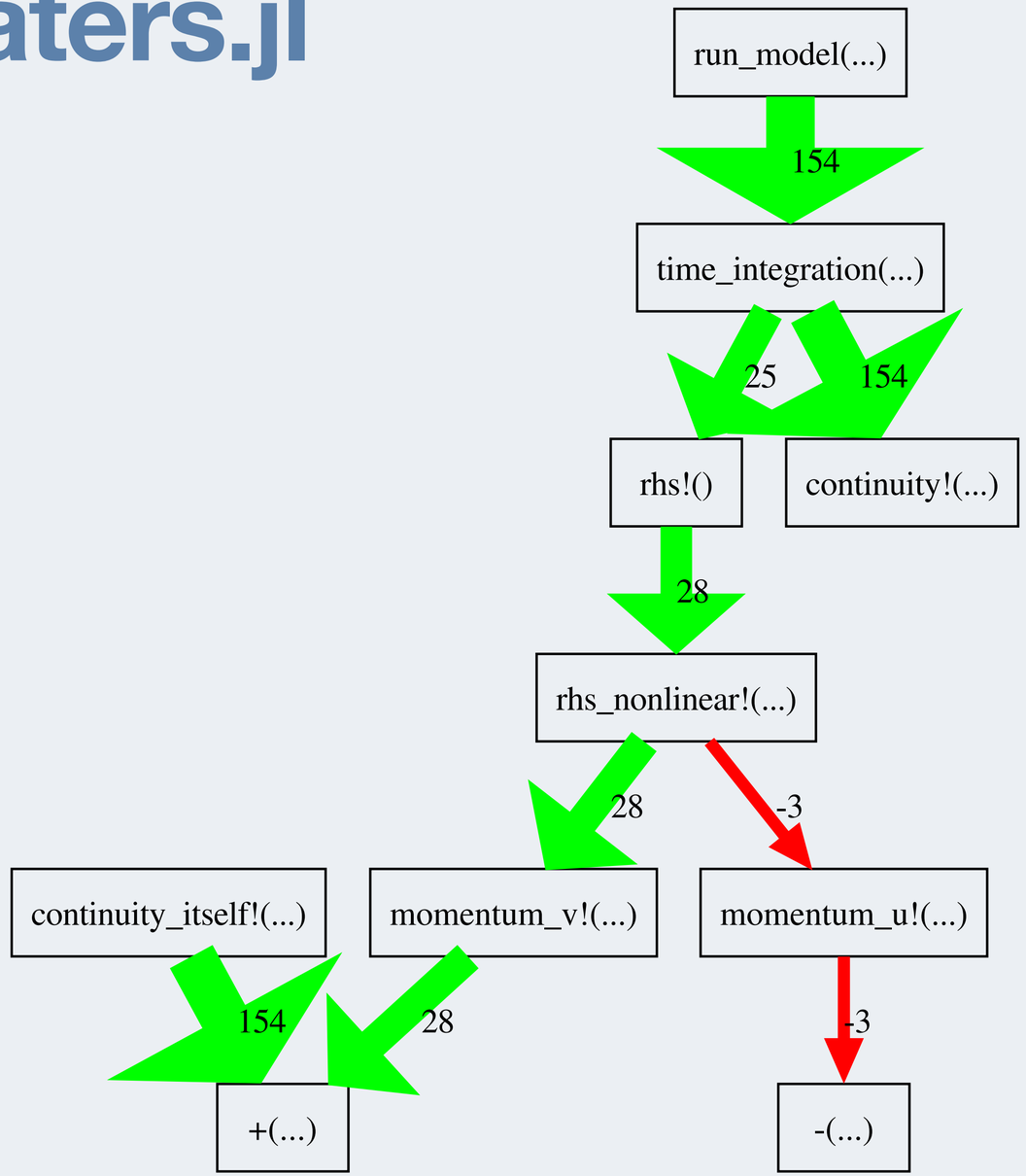
sw_gen_logs_part1.txt



sw_gen_logs_part2.txt

ShallowWaters.jl

CSTG Diff



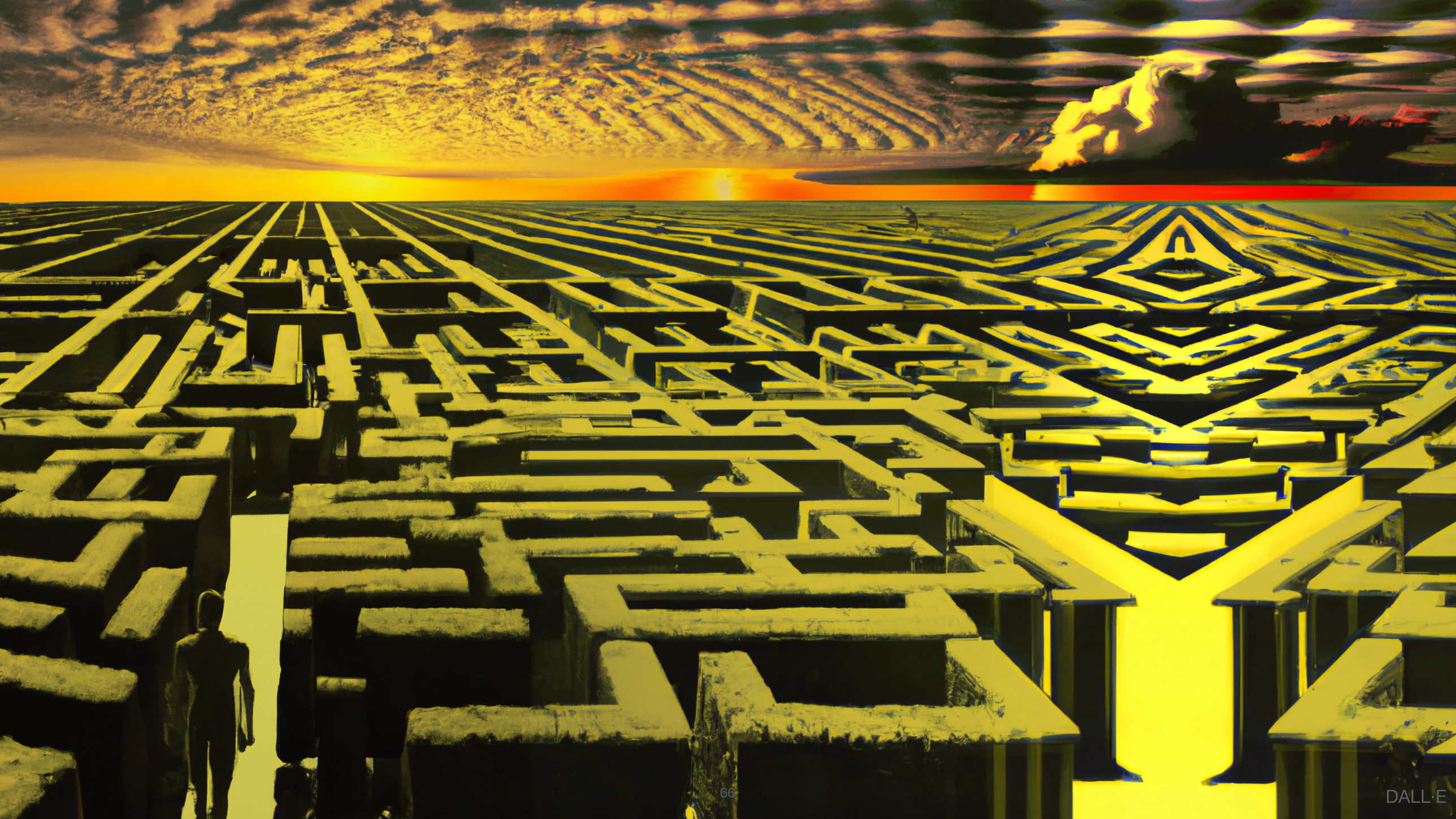
Fuzzing

Demmel et al. arXiv:2207.09281

https://www.youtube.com/watch?v=x4fdUx6d4QM

Demmel et al. arXiv:2207.09281

https://www.youtube.com/watch?v=x4fdUx6d4QM



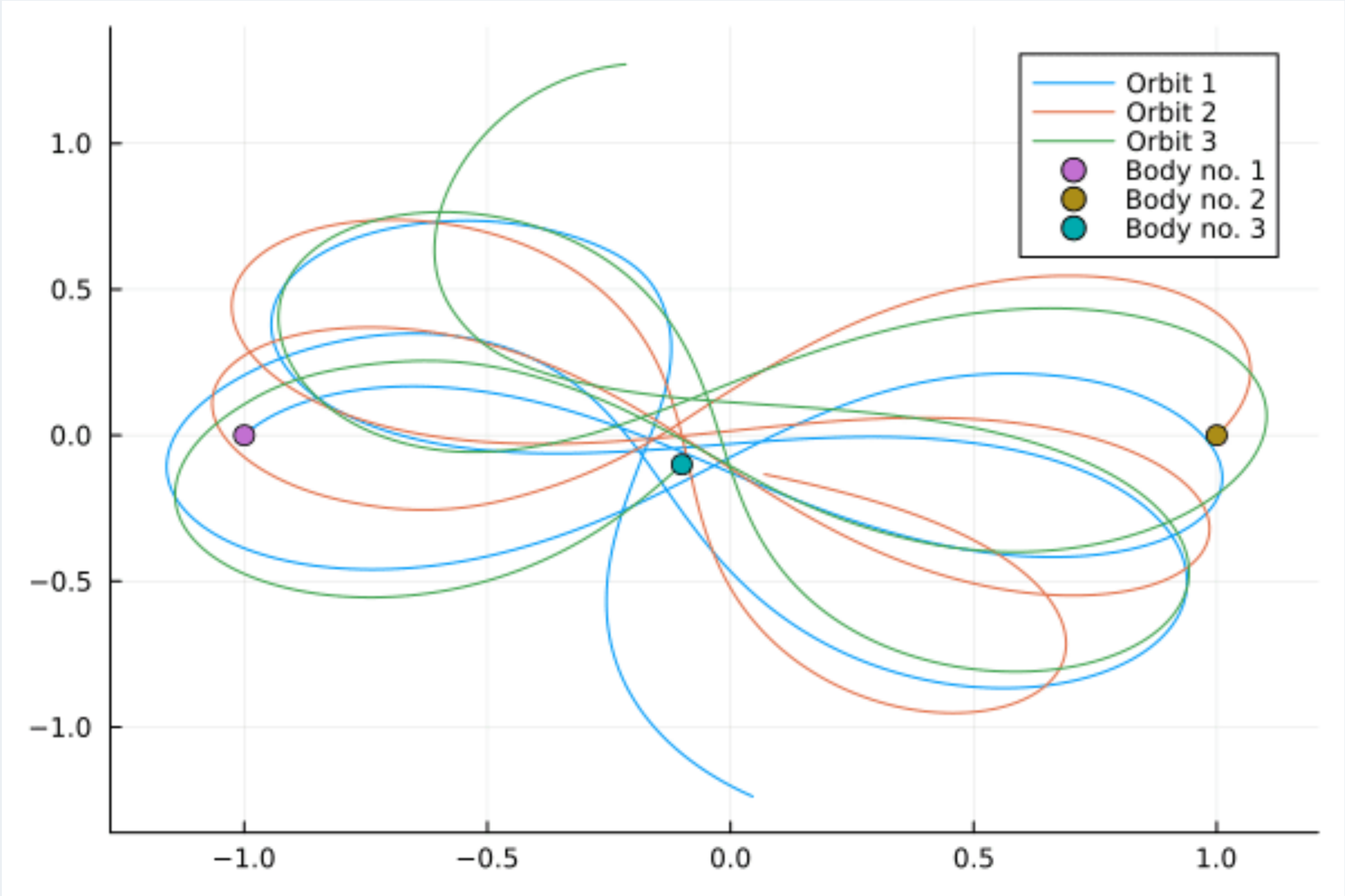
DALL·E

DALL·E

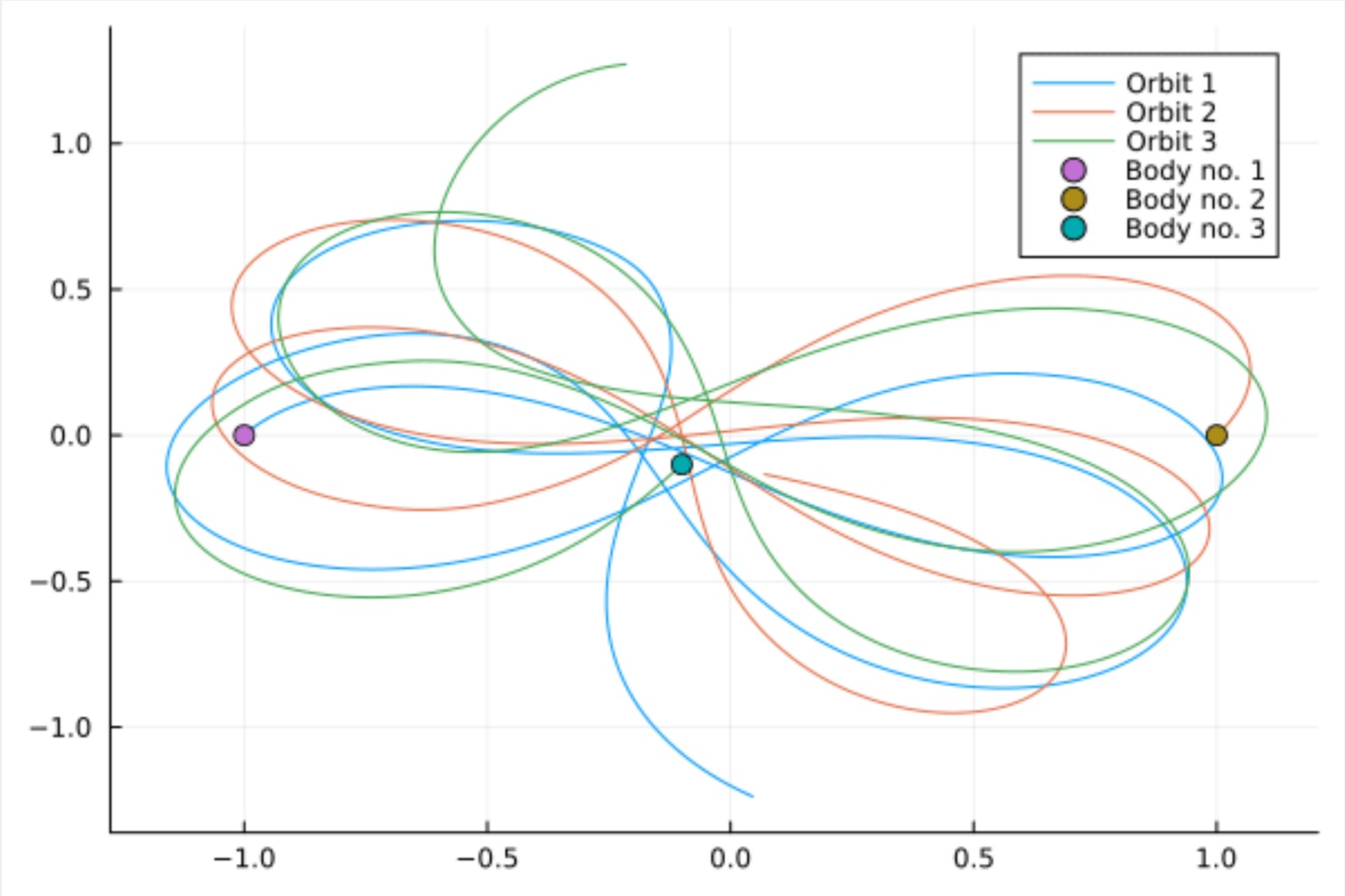
Let’s call FloatTracker 

NBodySimulator.jl

NBodySimulator.jl

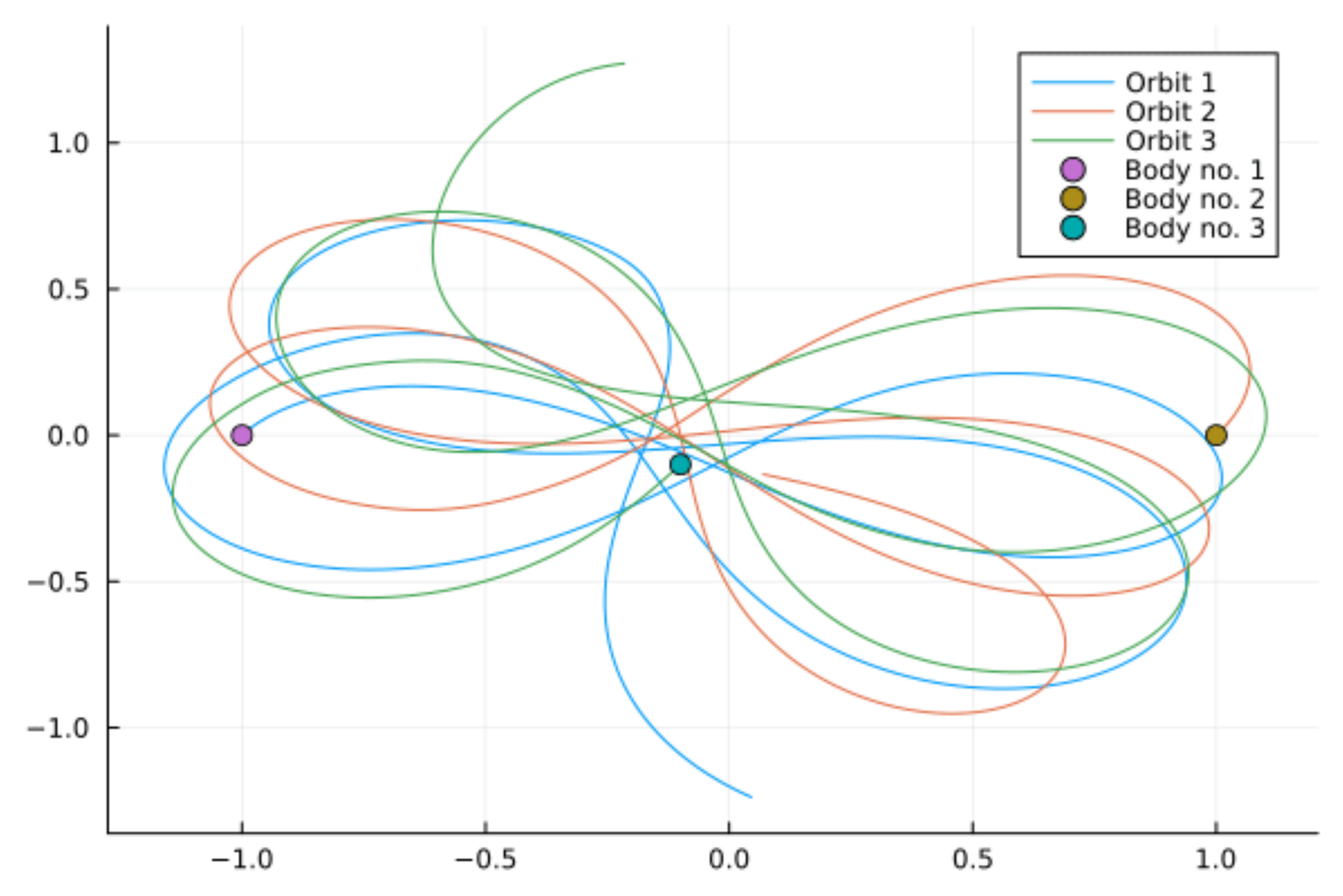


NBodySimulator.jl

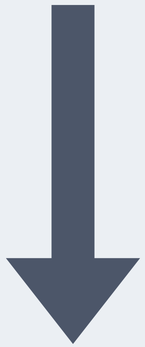


NBodySimulator.jl

Nothing—no FP operations!



NBodySimulator.jl



OrdinaryDiffEq.jl

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

```
odds::Int64
n_inject::Int64
functions::Array{FunctionRef}
libraries::Array{String}
record::String
replay::String
```

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

```
odds::Int64
n_inject::Int64
functions::Array{FunctionRef}
libraries::Array{String}
record::String
replay::String
```

OrdinaryDiffEq.jl

```
config_injector(odds=2,
                libraries=["NBodySimulator", "OrdinaryDiffEq"])
record_injection("injection_recording.txt")
```

```
odds::Int64
n_inject::Int64
functions::Array{FunctionRef}
libraries::Array{String}
record::String
replay::String
```

OrdinaryDiffEq.jl

injection_recording.txt

OrdinaryDiffEq.jl

`injection_recording.txt`

Repeat interesting injections
after hardening the code

OrdinaryDiffEq

ode_kill_logs.txt

```
[NaN] check_error          at FloatTracker/src/TrackedFloat.jl:11
<                          at FloatTracker/src/TrackedFloat.jl:214
solve!                     at OrdinaryDiffEq/src/solve.jl:515
…
solve_call                 at DiffEqBase/src/solve.jl:466
…
solve                      at DiffEqBase/src/solve.jl:819
…
run_simulation             at NBodySimulator/src/nbody_simulation_result.jl:289
top-level scope            at FTExamples/examples/nbody_replay.jl:29
```

OrdinaryDiffEq

ode_kill_logs.txt

```
[NaN] check_error          at FloatTracker/src/TrackedFloat.jl:11
<                          at FloatTracker/src/TrackedFloat.jl:214
solve!                     at OrdinaryDiffEq/src/solve.jl:515
…
solve_call                 at DiffEqBase/src/solve.jl:466
…
solve                      at DiffEqBase/src/solve.jl:819
…
run_simulation             at NBodySimulator/src/nbody_simulation_result.jl:289
top-level scope            at FTExamples/examples/nbody_replay.jl:29
```

OrdinaryDiffEq

```
function DiffEqBase.solve!(integrator::ODEIntegrator)
  @inbounds while !isempty(integrator.opts.tstops)
      while integrator.tdir * integrator.t < first(integrator.opts.tstops)
          loopheader!(integrator)
          if integrator.do_error_check && check_error!(integrator) != ReturnCode.Success
              return integrator.sol
          end
          perform_step!(integrator, integrator.cache)
          loopfooter!(integrator)
          if isempty(integrator.opts.tstops)
              break
          end
      end
      handle_tstop!(integrator)
  end
  postamble!(integrator)
  …
```

OrdinaryDiffEq/src/solve.jl:514

OrdinaryDiffEq

```
function DiffEqBase.solve!(integrator::ODEIntegrator)
  @inbounds while !isempty(integrator.opts.tstops)
      while integrator.tdir * integrator.t < first(integrator.opts.tstops)
          loopheader!(integrator)
          if integrator.do_error_check && check_error!(integrator) != ReturnCode.Success
              return integrator.sol
          end
          perform_step!(integrator, integrator.cache)
          loopfooter!(integrator)
          if isempty(integrator.opts.tstops)
              break
          end
      end
      handle_tstop!(integrator)
  end
  postamble!(integrator)
  …
```

OrdinaryDiffEq/src/solve.jl:514

OrdinaryDiffEq

```
function DiffEqBase.solve!(integrator::ODEIntegrator)
  @inbounds while !isempty(integrator.opts.tstops)
      while integrator.tdir * integrator.t < first(integrator.opts.tstops)
          loopheader!(integrator)
          if integrator.do_error_check && check_error!(integrator) != ReturnCode.Success
              return integrator.sol
          end
          perform_step!(integrator, integrator.cache)
          loopfooter!(integrator)
          if isempty(integrator.opts.tstops)
              break
          end
      end
      handle_tstop!(integrator)
  end
  postamble!(integrator)
  …
```

OrdinaryDiffEq/src/solve.jl:514

OrdinaryDiffEq

```
function DiffEqBase.solve!(integrator::ODEIntegrator)
  @inbounds while !isempty(integrator.opts.tstops)
      while integrator.tdir * integrator.t < first(integrator.opts.tstops)
          loopheader!(integrator)
          if integrator.do_error_check && check_error!(integrator) != ReturnCode.Success
              return integrator.sol
          end
          perform_step!(integrator, integrator.cache)
          loopfooter!(integrator)
          if isempty(integrator.opts.tstops)
              break
          end
      end
      handle_tstop!(integrator)
  end
  postamble!(integrator)
  …
```

OrdinaryDiffEq/src/solve.jl:514

OrdinaryDiffEq

```
function DiffEqBase.solve!(integrator::ODEIntegrator)
  @inbounds while !isempty(integrator.opts.tstops)
      while integrator.tdir * integrator.t < first(integrator.opts.tstops)
          loopheader!(integrator)
          if integrator.do_error_check && check_error!(integrator) != ReturnCode.Success
              return integrator.sol
          end
          perform_step!(integrator, integrator.cache)
          loopfooter!(integrator)
          if isempty(integrator.opts.tstops)
              break
          end
      end
      handle_tstop!(integrator)
  end
  postamble!(integrator)
  …
```

OrdinaryDiffEq/src/solve.jl:514

OrdinaryDiffEq

```
function DiffEqBase.solve!(integrator::ODEIntegrator)
  @inbounds while !isempty(integrator.opts.tstops)
      while integrator.NaN * integrator.t < first(integrator.opts.tstops)
          loopheader!(integrator)
          if integrator.do_error_check && check_error!(integrator) != ReturnCode.Success
              return integrator.sol
          end
          perform_step!(integrator, integrator.cache)
          loopfooter!(integrator)
          if isempty(integrator.opts.tstops)
              break
          end
      end
      handle_tstop!(integrator)
  end
  postamble!(integrator)
  …
```

OrdinaryDiffEq/src/solve.jl:514

OrdinaryDiffEq

```
function DiffEqBase.solve!(integrator::ODEIntegrator)
  @inbounds while !isempty(integrator.opts.tstops)
      while integrator.tdir * integrator.t < first(integrator.opts.tstops)
          loopheader!(integrator)
          if integrator.do_error_check && check_error!(integrator) != ReturnCode.Success
              return integrator.sol
          end
          perform_step!(integrator, integrator.cache)
          loopfooter!(integrator)
          if isempty(integrator.opts.tstops)
              break
          end
      end
      handle_tstop!(integrator)
  end
  postamble!(integrator)
  …
```

OrdinaryDiffEq/src/solve.jl:514

OrdinaryDiffEq

```
function DiffEqBase.solve!(integrator::ODEIntegrator)
  @inbounds while !isempty(integrator.opts.tstops)
      while integrator.tdir * integrator.t < first(integrator.opts.tstops)
          loopheader!(integrator)
          if integrator.do_error_check && check_error!(integrator) != ReturnCode.Success
              return integrator.sol
          end
          perform_step!(integrator, integrator.cache)
          loopfooter!(integrator)
          if isempty(integrator.opts.tstops)
              break
          end
      end
      handle_tstop!(integrator)
  end
  postamble!(integrator)
  …
```

https://github.com/SciML/OrdinaryDiffEq.jl/issues/1939

OrdinaryDiffEq/src/solve.jl:514

FlowFPX Internals

Making FloatTracker Work

Intercept floating-point operations

```
Float16
Float32
Float64
```

Making FloatTracker Work

Intercept floating-point operations

```
TrackedFloat16
TrackedFloat32
TrackedFloat64
```

Making FloatTracker Work

Intercept floating-point operations

```
abstract type AbstractTrackedFloat <: AbstractFloat end

struct TrackedFloat32 <: AbstractTrackedFloat
  val::Float32
end
```

Making FloatTracker Work

Intercept floating-point operations

```
function Base.+(x::TrackedFloat32, y::TrackedFloat32)
  result = x.val + y.val
  check_error(+, result, x.val, y.val)
  TrackedFloat32(result)
end
```

Making FloatTracker Work

Intercept floating-point operations

```
function Base.+(x::TrackedFloat32, y::TrackedFloat32)
  result = x.val + y.val
  check_error(+, result, x.val, y.val)
  TrackedFloat32(result)
end
```

✨Type Dispatch✨

Making FloatTracker Work

Intercept floating-point operations

```
function Base.+(x::TrackedFloat32, y::TrackedFloat32)
  result = x.val + y.val
  check_error(+, result, x.val, y.val)
  TrackedFloat32(result)
end
```

Of course, this would be hard to maintain

Making FloatTracker Work

Making FloatTracker Work

Enter the Macros

Making FloatTracker Work

```
for TrackedFloatN in (:TrackedFloat16, :TrackedFloat32, :TrackedFloat64)
  for Op in (:+, :-, :/, :^)
    @eval function Base.$Op(x::$TrackedFloatN, y::$TrackedFloatN)
      result = $Op(x, y)
      check_error($Op, result, x.val, y.val)
      $TrackedFloatN(result)
    end
  end
end
```

Inspired by Milan Klöwer's Sherlogs.jl library

https://github.com/milankl/Sherlogs.jl

218

Lines of Code in TrackedFloat.jl

645

Generated Function Variants

You can write your own overrides too!

Custom Overrides: Clapeyron

```
Base.:+(x::ForwardDiff.Dual, y::TrackedFloat32) = x - y.val
```

Custom Overrides: Clapeyron

```
Base.:+(x::ForwardDiff.Dual, y::TrackedFloat32) = x - y.val
```

Conclusion

Tracking down instances of `NaN` and `Inf` doesn't have to be painful

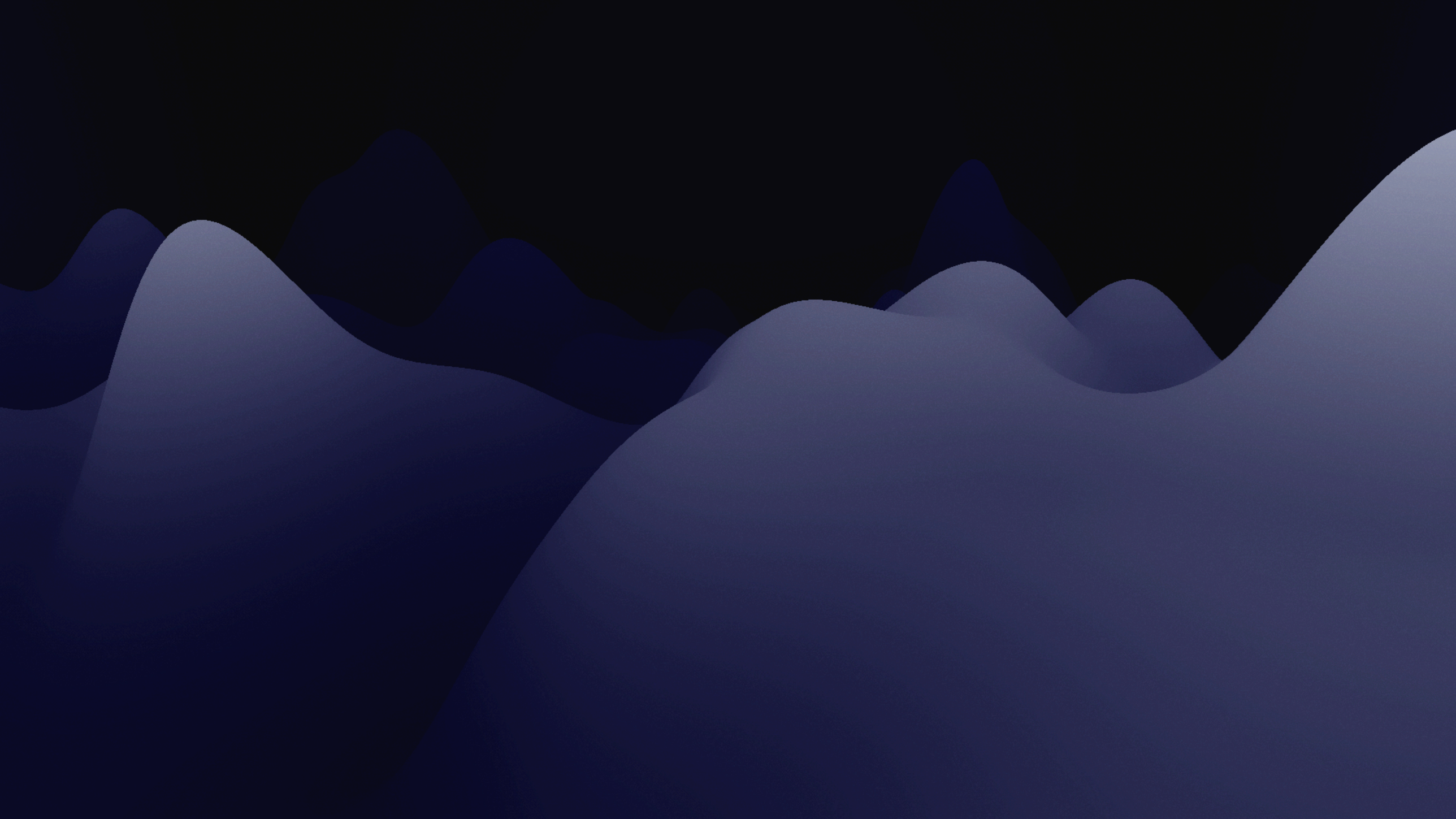
Tracking down instances of `NaN` and `Inf` doesn't have to be painful

FloatTracker + CSTG are ready to use

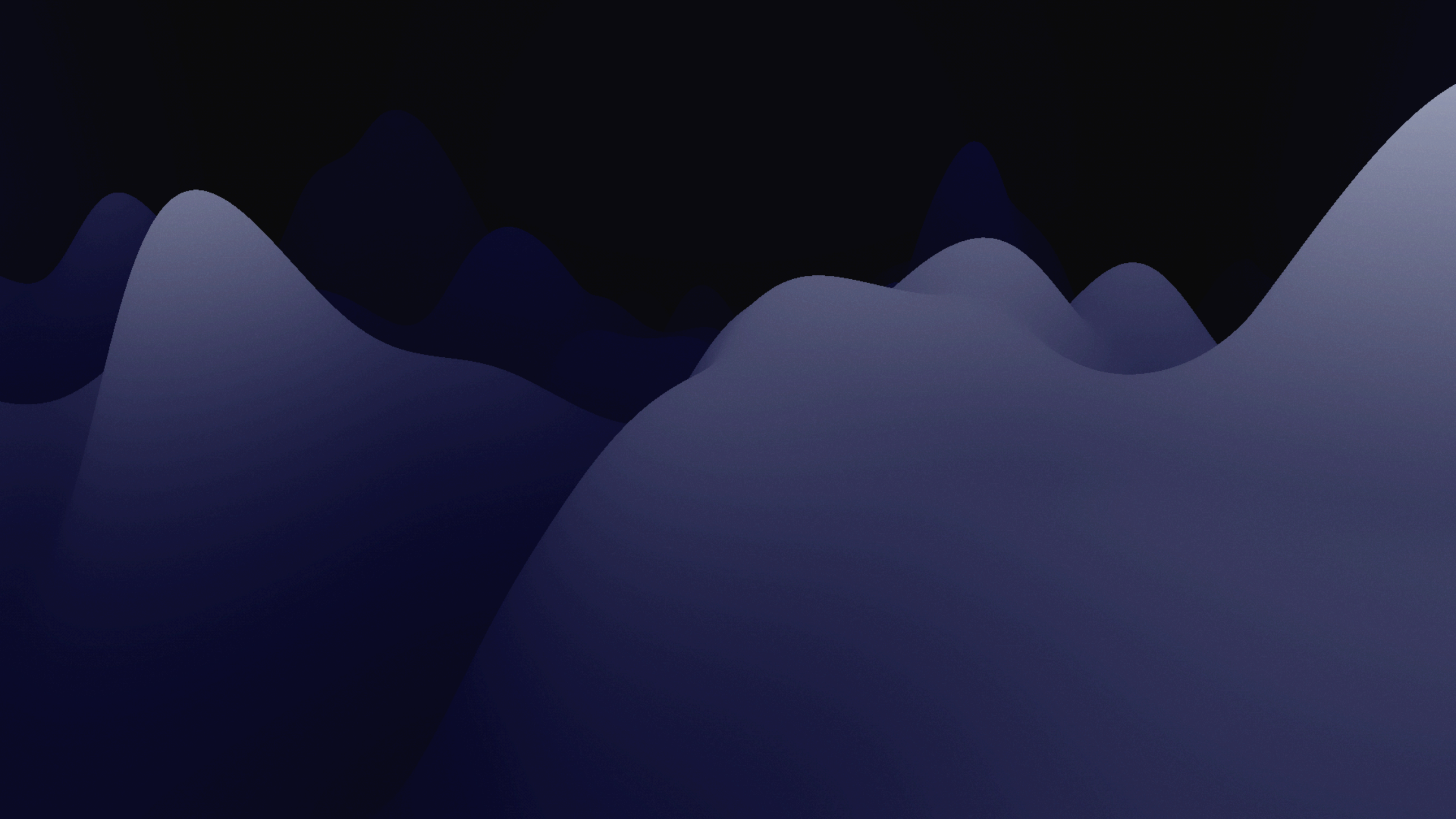
Tracking down instances of `NaN` and `Inf` doesn't have to be painful

FloatTracker + CSTG are ready to use

Type dispatch: the great enabler



Who you gonna call? 📞



FloatTracker

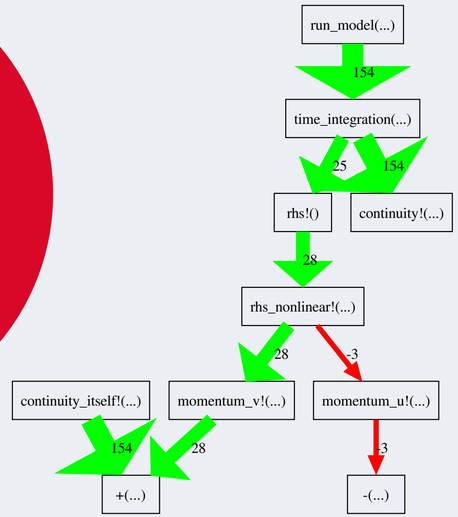
FlowFPX

FloatTracker

Track and fuzz exceptional values

`https://github.com/utahplt/FloatTracker.jl`

JuliaHub



CSTG

Summarize flows in graphs

`https://github.com/utahplt/CSTG`

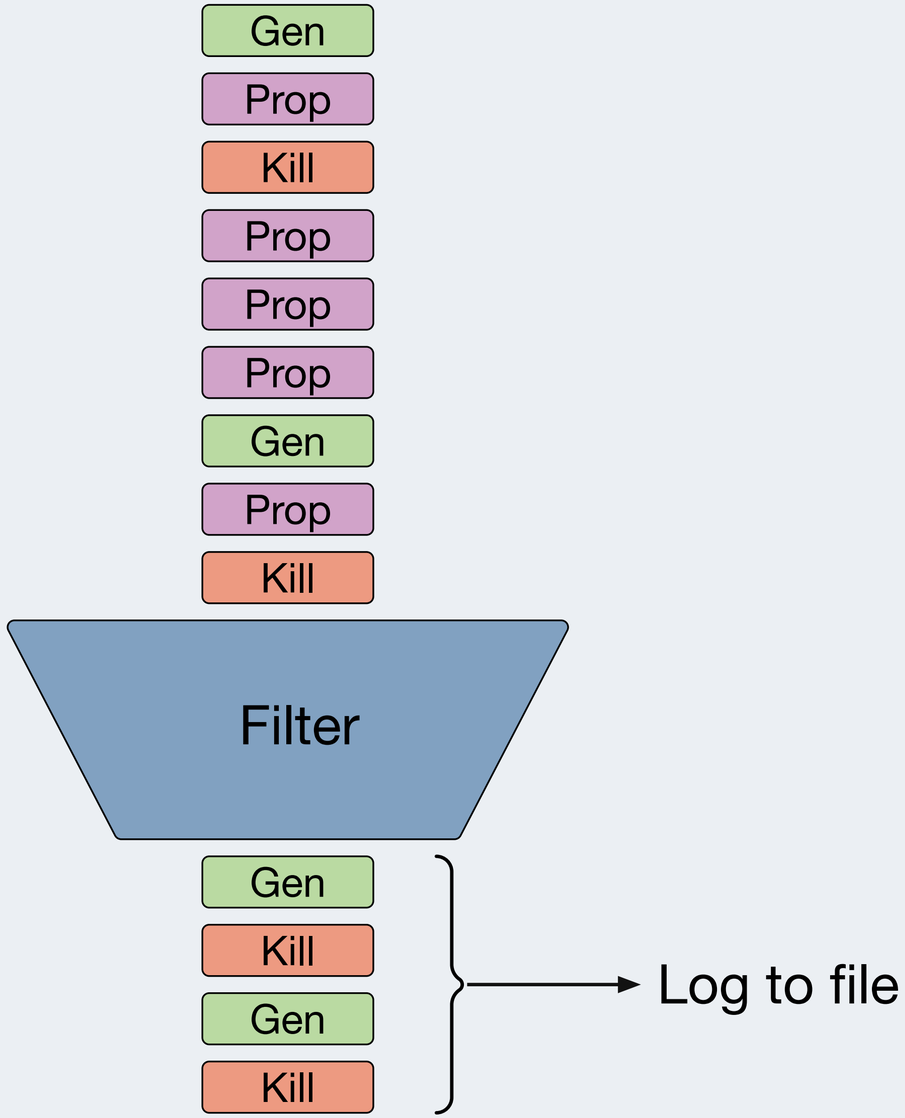
Performance (ShallowWaters)

3	s		(no NaN, no FloatTracker)
<1	s		(NaN, no FloatTracker)
79	s	25x	(no NaN, full FloatTracker)
66	s	9500x	(NaN, full FloatTracker)
1	s	164x	(NaN, FloatTracker logs first 100 gen events)
1	s	160x	(NaN, FloatTracker with logging turned off)

Performance

Filter event types

Limit number of events logged



Performance

- Short of it: DO NOT RUN IN PRODUCTION—this is a profiler
- Log filtering helps tremendously

Exception land

- Domain experts now get a *concrete number* to look at
- Also get some stack traces as hints as to where to look next

Comparison with flame graphs

- Flame graphs track time spent in each function (performance)
- CSTG tracks number of invocations (frequency)
- A routine frequently generating/killing NaN might run quickly—won't show up on a flame graph, but CSTG will highlight its role

GPU-FPX

Detects exceptions in the GPU at runtime

```
-- FP32 Operations --
Total NaN:             1
Total INF:             0
Total subnormal:       0
Total div0:            0
```

https://github.com/LLNL/GPU-FPX

https://doi.org/10.1145/3588195.3592991

$$x = 2e38$$

$$y = 1e38$$

$$(x + x) - y \Rightarrow Inf$$

$$x = 2e38$$

$$y = 1e38$$

$$(x + x) - y \Rightarrow Inf$$

```
x = 2e38
y = 1e38
(x + x) - y ⇛ Inf
x + (x - y) ⇛ 3e38
```