Well-Behaved Native Implemented Functions for Elixir

potatosalad

September 8, 2017

Well-Behaved

Native Implemented Functions for Elixir

---

gitpitch.com/potatosalad/elixirconf2017

Andrew Bennett

 potatosalad

 potatosaladx

Native Implemented Functions (NIFs)
still experimental but very useful.

Feedback is welcome.

— *OTP-R13B03 announcement (25 Nov 2009)*

Type	Isolation	Latency*
Node	Network	~100µs
Port	Process	~100µs
Port Driver	Shared	~10µs
NIF	Shared	~0.1µs

*Rounded to nearest order of magnitude.

potatosalad.io/2017/08/05/latency-of-native-functions-for-erlang-and-elixir

Don’t write a NIF unless you have to

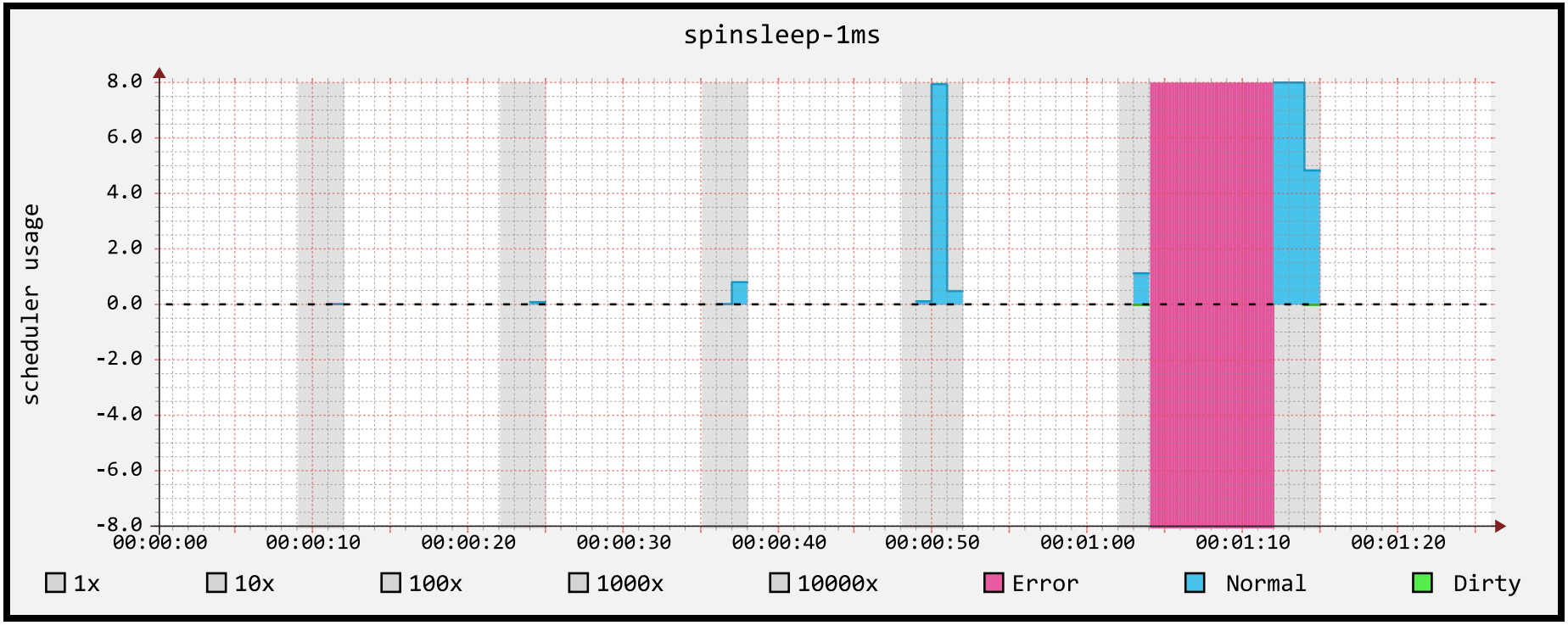
A well-behaving native function is to return to its caller within 1 millisecond.

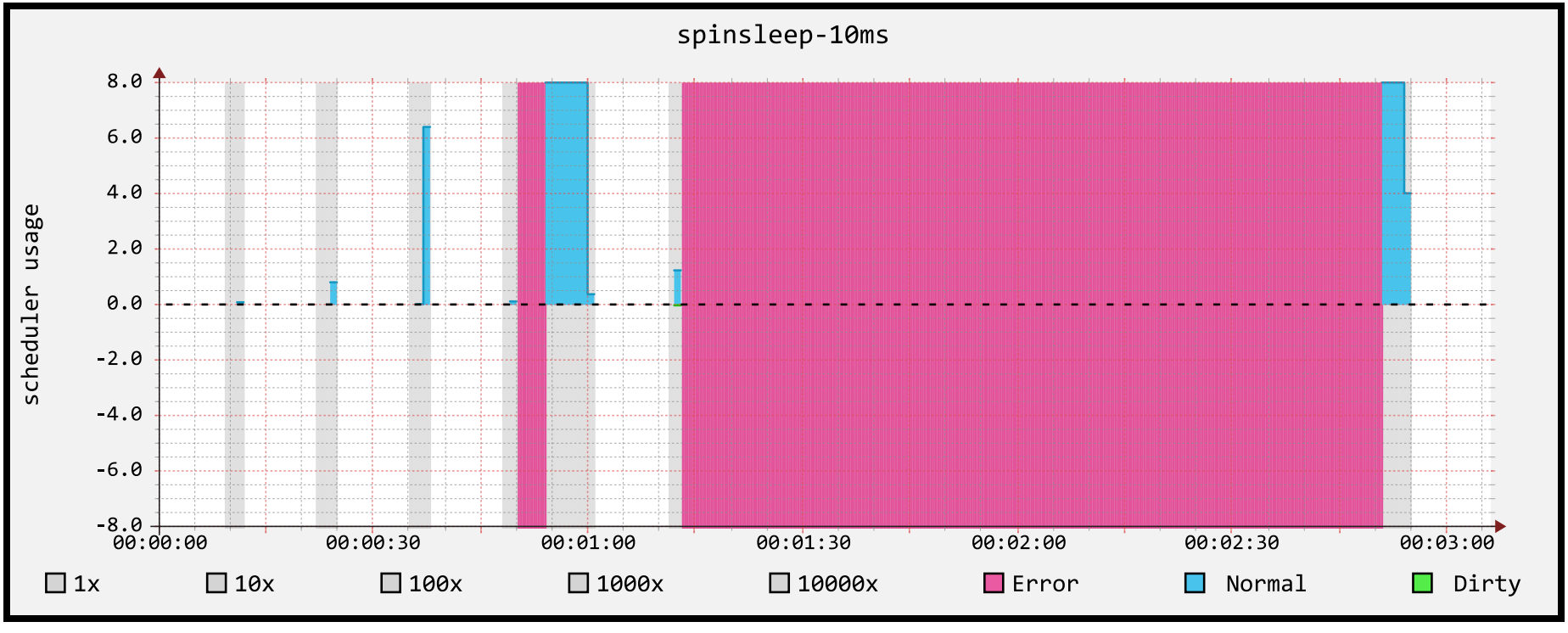
— *ERTS 9.0 erl_nif docs*

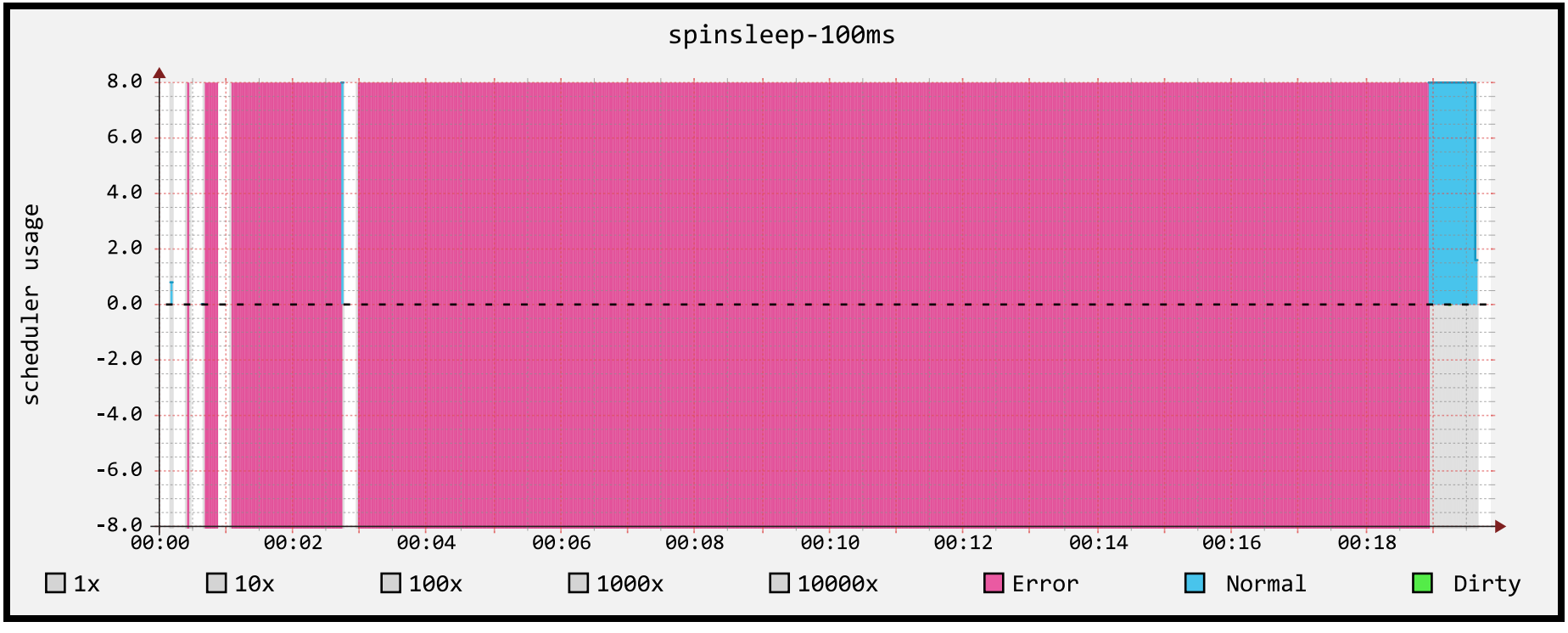
```
void
spin(ErlNifSInt64 count)
{
  for (; count > 0; --count) {}
}
```

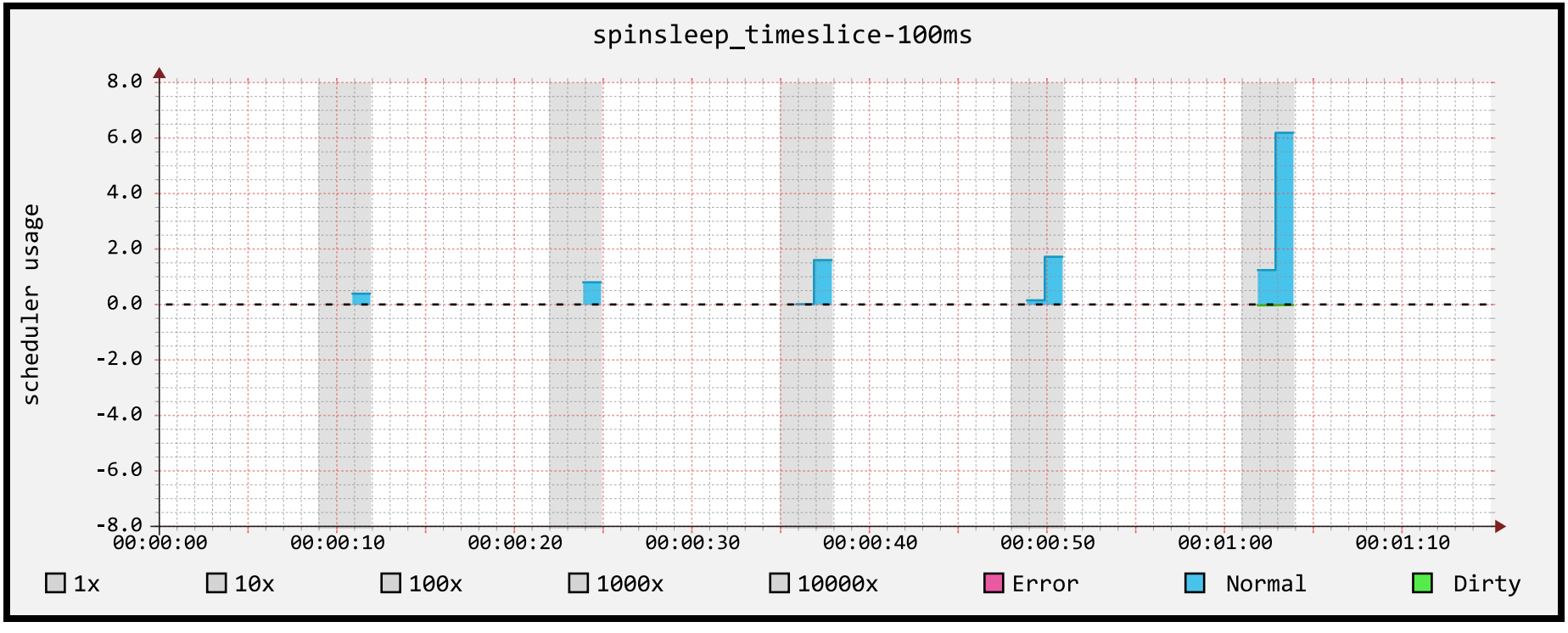
```
ErlNifSInt64
spinsleep(ErlNifSInt64 microseconds)
{
  ErlNifTime start, current, stop;
  ErlNifSInt64 count;
  start = enif_monotonic_time(ERL_NIF_NSEC);
  stop = start + ((ErlNifTime)microseconds * 1000);
  do {
    current = enif_monotonic_time(ERL_NIF_NSEC);
    count = (stop - current) / 2;
    (void)spin(count);
  } while (stop > current);
  return ((current - start) / 1000);
}
```

```
def spinsleep(microseconds, multiplier) do
  function = fn () ->
    :my_nif.spinsleep(microseconds)
  end
  count = :erlang.system_info(:schedulers_online) * multipler
  spawn_multiple(function, count)
end
```


spinsleep-1ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
00:01:20
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep-10ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:30
00:01:00
00:01:30
00:02:00
00:02:30
00:03:00
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep-100ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00
00:02
00:04
00:06
00:08
00:10
00:12
00:14
00:16
00:18
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice-100ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
1x
10x
100x
1000x
10000x
Error
Normal
Dirty

Long-running NIF Causes

- Variable input/output
  - `binary`
  - `list`
  - `map`
  - `tuple`
- CPU blocking
- I/O blocking

Long-running NIF Solutions

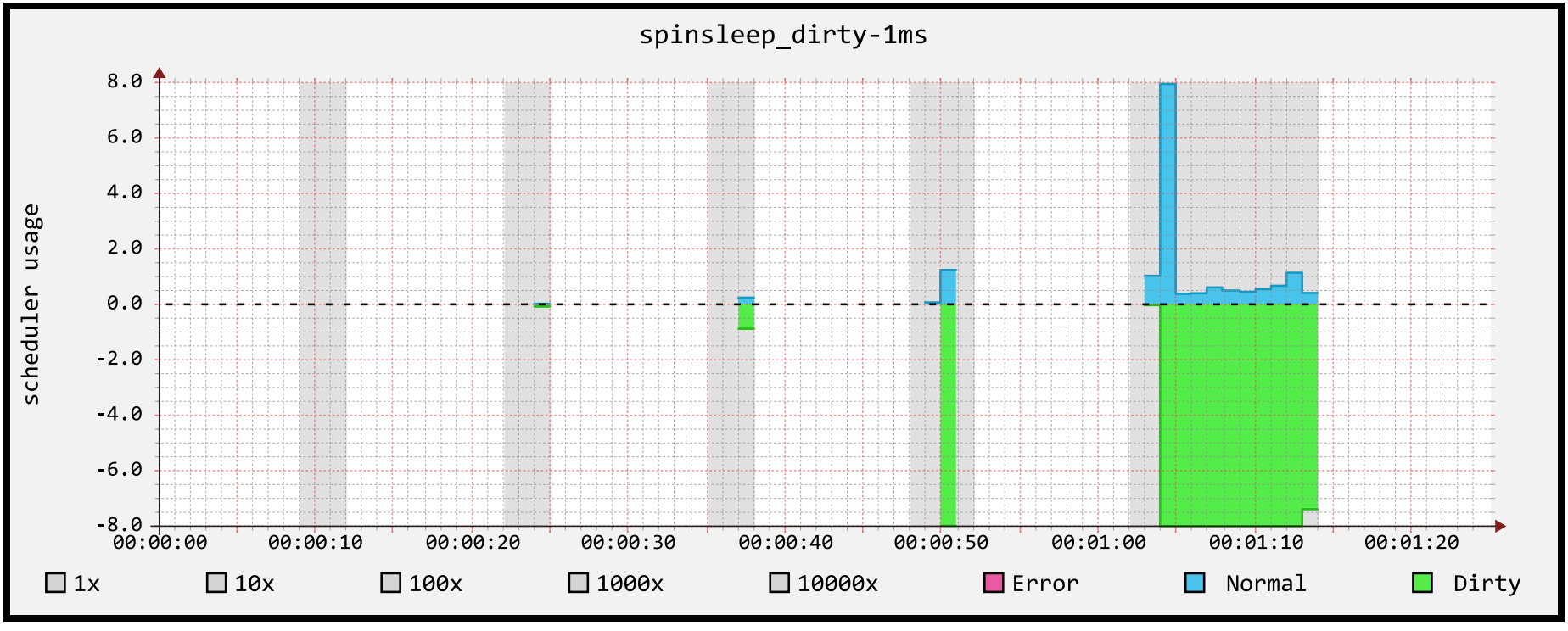
1. Dirty NIF
2. Yielding NIF (timeslice)
3. Yielding Dirty NIF
4. Threaded NIF

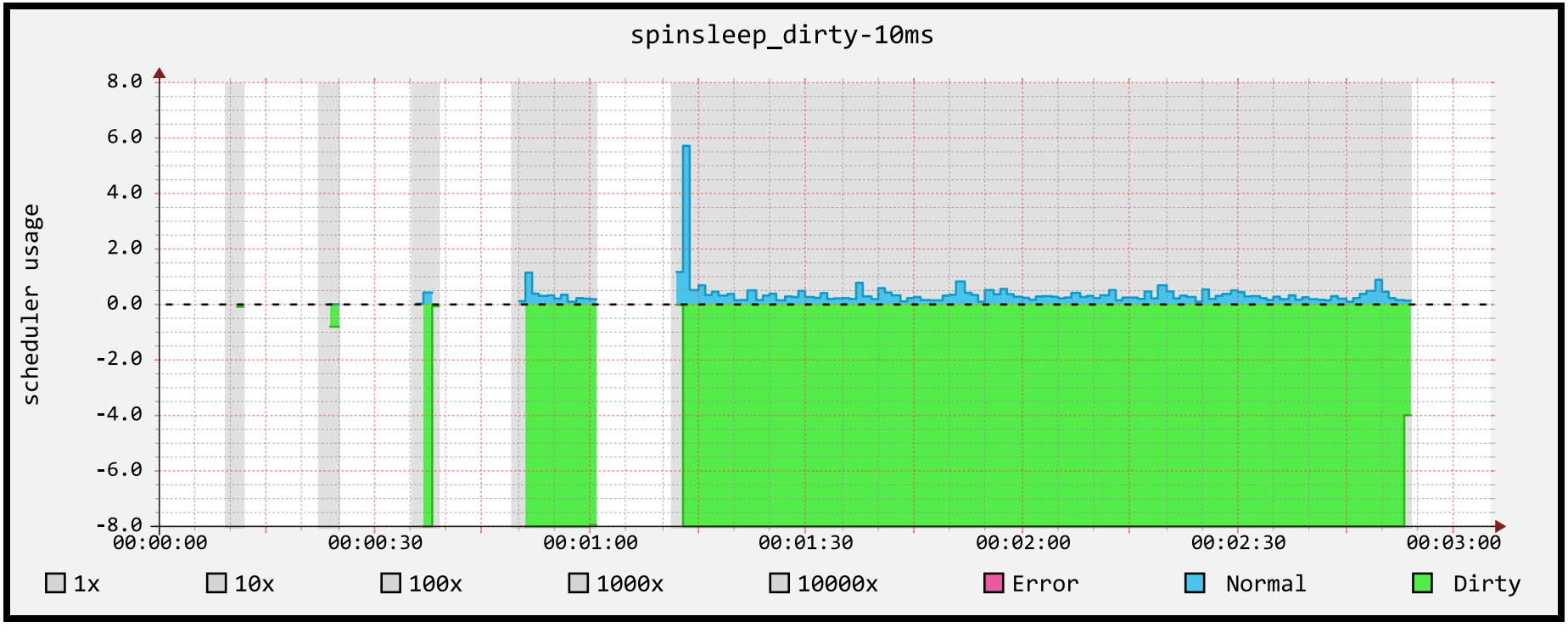
Dirty NIF

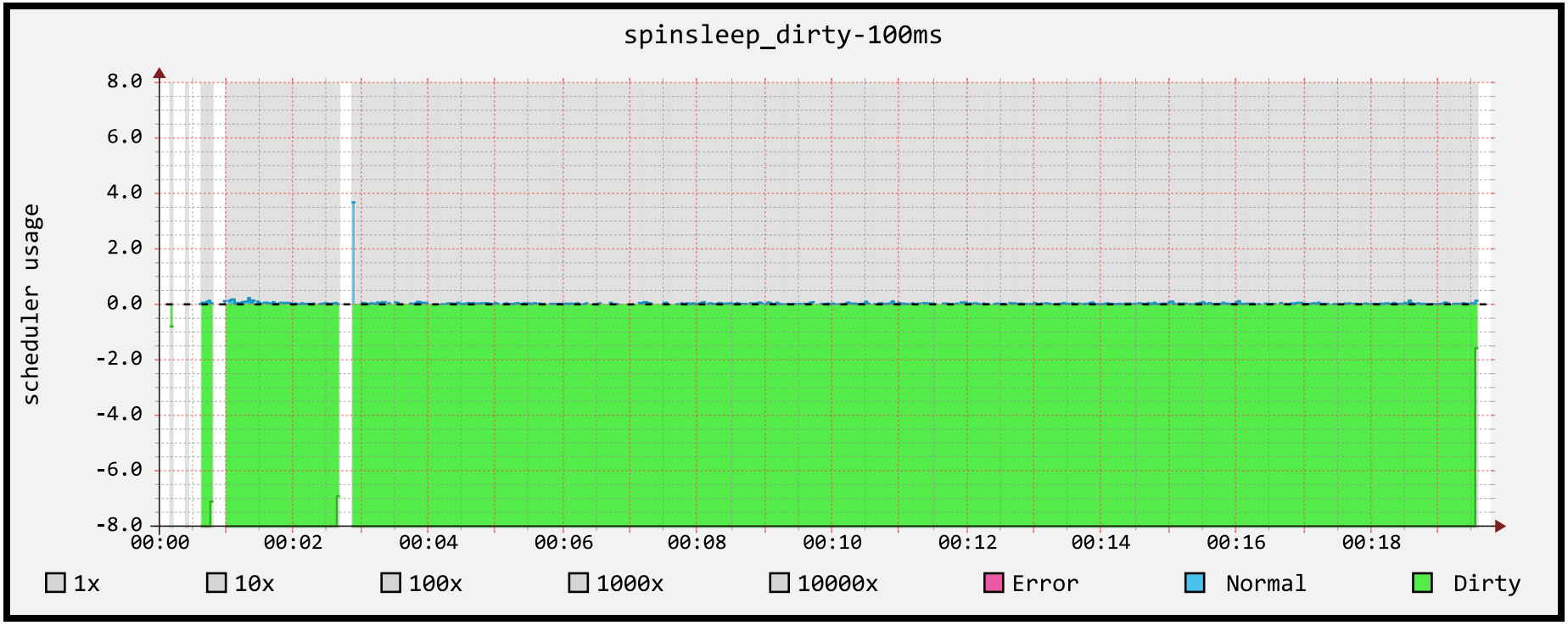
A NIF that cannot be split and cannot execute in a millisecond or less.

— *ERTS 9.0 erl_nif docs*

```
{"spinsleep", 1, spinsleep},
{"spinsleep_dirty", 1, spinsleep, ERL_NIF_DIRTY_JOB_CPU_BOUND},
```


spinsleep_dirty-1ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
00:01:20
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_dirty-10ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:30
00:01:00
00:01:30
00:02:00
00:02:30
00:03:00
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_dirty-100ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00
00:02
00:04
00:06
00:08
00:10
00:12
00:14
00:16
00:18
1x
10x
100x
1000x
10000x
Error
Normal
Dirty

“Dirty NIF” does not mean “Faster NIF”

Yielding NIF (timeslice)

This approach is always preferred over the other alternatives.

— *ERTS 9.0 erl_nif docs*

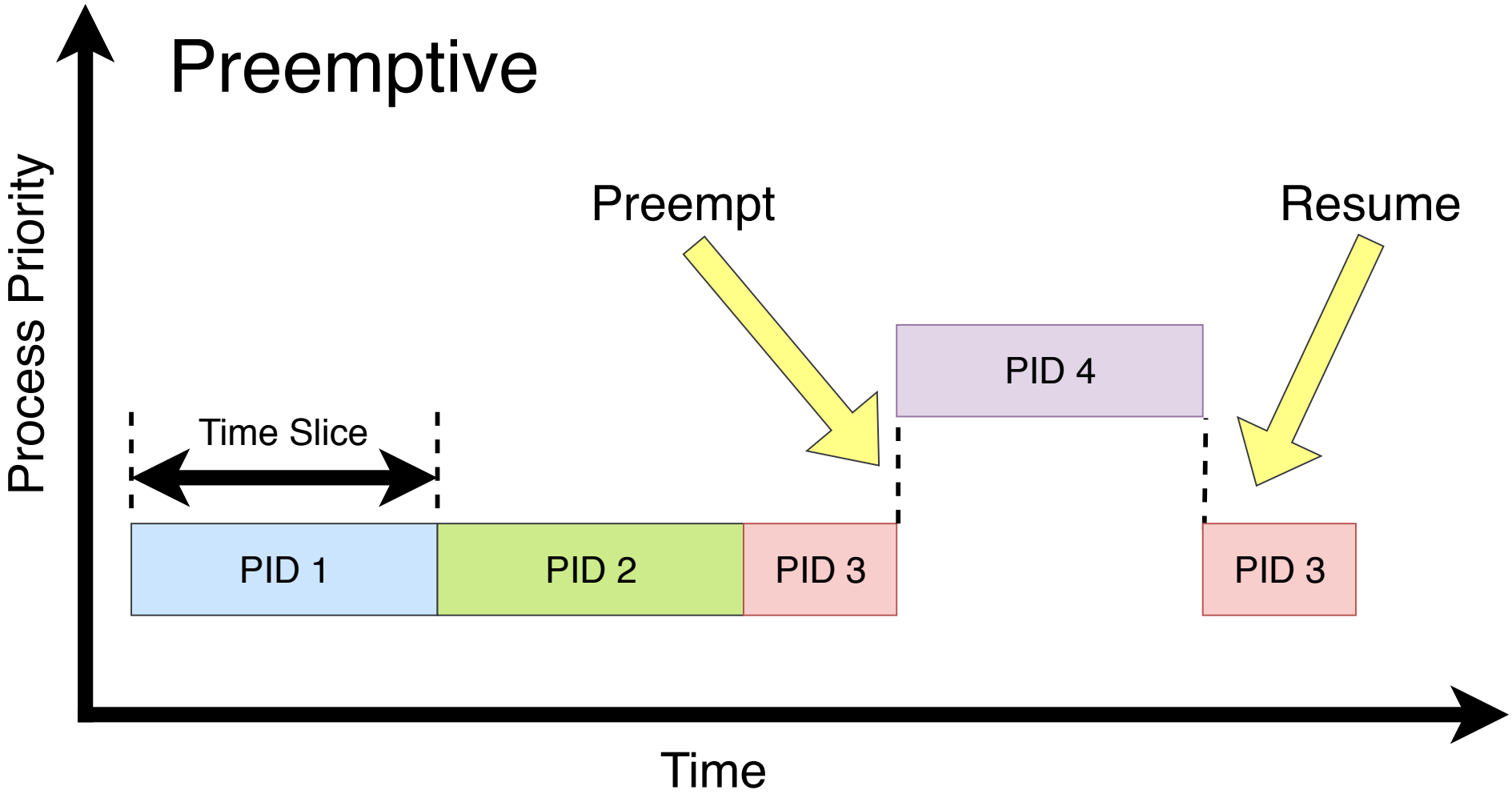
Bogdan/Björn's Erlang Abstract Machine

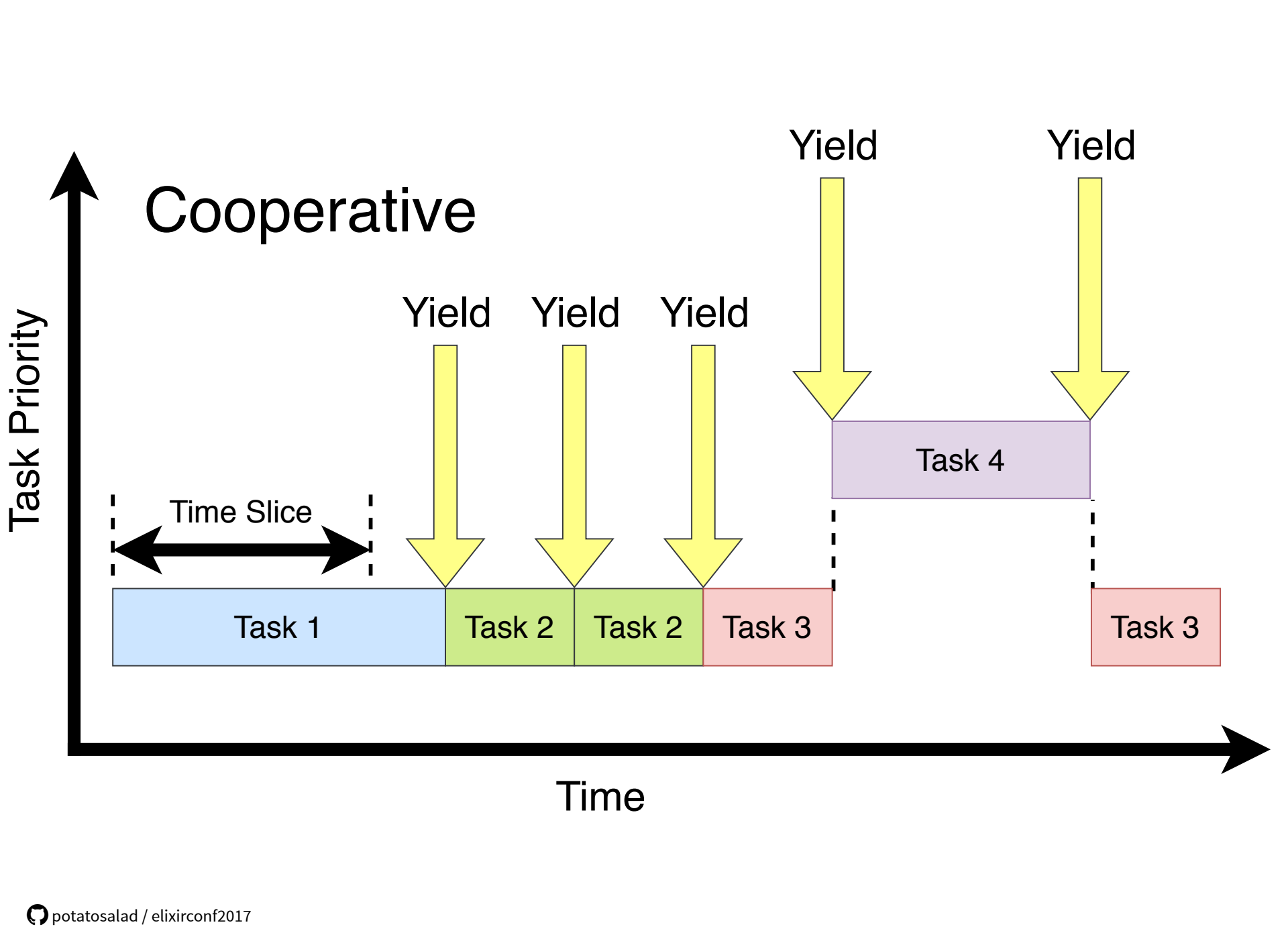
BEAM

BEAM Scheduling

The scheduler is responsible for the [soft] real-time guarantees of the system.

— *The BEAM Book*


Preemptive
Process Priority
Preempt
Resume
PID 4
Time Slice
PID 1
PID 2
PID 3
PID 3
Time


Cooperative
Yield
Yield
Yield
Yield
Yield
Task Priority
Time Slice
Task 4
Task 1
Task 2
Task 2
Task 3
Task 3
Time

BEAM Scheduling

One can describe the scheduling in BEAM as preemptive scheduling on top of cooperative scheduling.

— *The BEAM Book*

Preemptive vs Cooperative

- Elixir functions are preemptive
- C functions are cooperative…hopefully

Elixir Functions

A process can only be suspended at certain points of the execution, such as at a receive or a function call.

— *The BEAM Book*

Estimating Time

When a process is scheduled it will get a number of reductions defined by `CONTEXT_REDS` (currently 4000).

— *The BEAM Book*

What is a Reduction?

It is not completely defined what a reduction is, but at least each function call should be counted as a reduction.

— *The BEAM Book*

```
def echo(term) do
  term
end
```

```
self = :erlang.self()
{:reductions, r1} = :erlang.process_info(self, :reductions)
:ok = echo(:ok)
{:reductions, r2} = :erlang.process_info(self, :reductions)
rdiff = r2 - r1 # ~400
```

```elixir
def collect(_ref, 0, replies) do
  replies
end
def collect(ref, n, replies) do
  receive do
    {^ref, reply} ->
      collect(ref, n - 1, [reply | replies])
  end
end
```

```elixir
parent = self()
ref = make_ref()
:ok =
  Enum.reduce(1..1000, :ok, fn (_, ok) ->
    _ = spawn(:erlang, :send, [parent, {ref, 1}])
    ok
  end)
{:reductions, r1} = :erlang.process_info(parent, :reductions)
replies = collect(ref, 1000, [])
{:reductions, r2} = :erlang.process_info(parent, :reductions)
1000 = Enum.sum(replies)
rdiff = r2 - r1 # ~1400
```

C Functions

There is a risk that a function implemented in C takes many more clock cycles per reduction than a normal Erlang function.

— *The BEAM Book*

```
nif_bif_result = (*fp)(&env, bif_nif_arity, reg);
```

erts/emulator/beam/bif_instrs.tab

blocks thread until the NIF returns

```c
static ERL_NIF_TERM
echo(ErlNifEnv *env, int argc, const ERL_NIF_TERM argv[])
{
  return argv[0];
}
```

```
self = :erlang.self()
{:reductions, r1} = :erlang.process_info(self, :reductions)
:ok = :my_nif.echo(:ok)
{:reductions, r2} = :erlang.process_info(self, :reductions)
rdiff = r2 - r1 # ~200
```

```
static ERL_NIF_TERM
echo(ErlNifEnv *env, int argc, const ERL_NIF_TERM argv[])
{
  (void) spinsleep(1000 * 1000); // spin for 1 second
  return argv[0];
}
```

reductions = ~200

enif_consume_timeslice

```c
static ERL_NIF_TERM
echo(ErlNifEnv *env, int argc, const ERL_NIF_TERM argv[])
{
  (void) enif_consume_timeslice(env, 100);
  return argv[0];
}
```

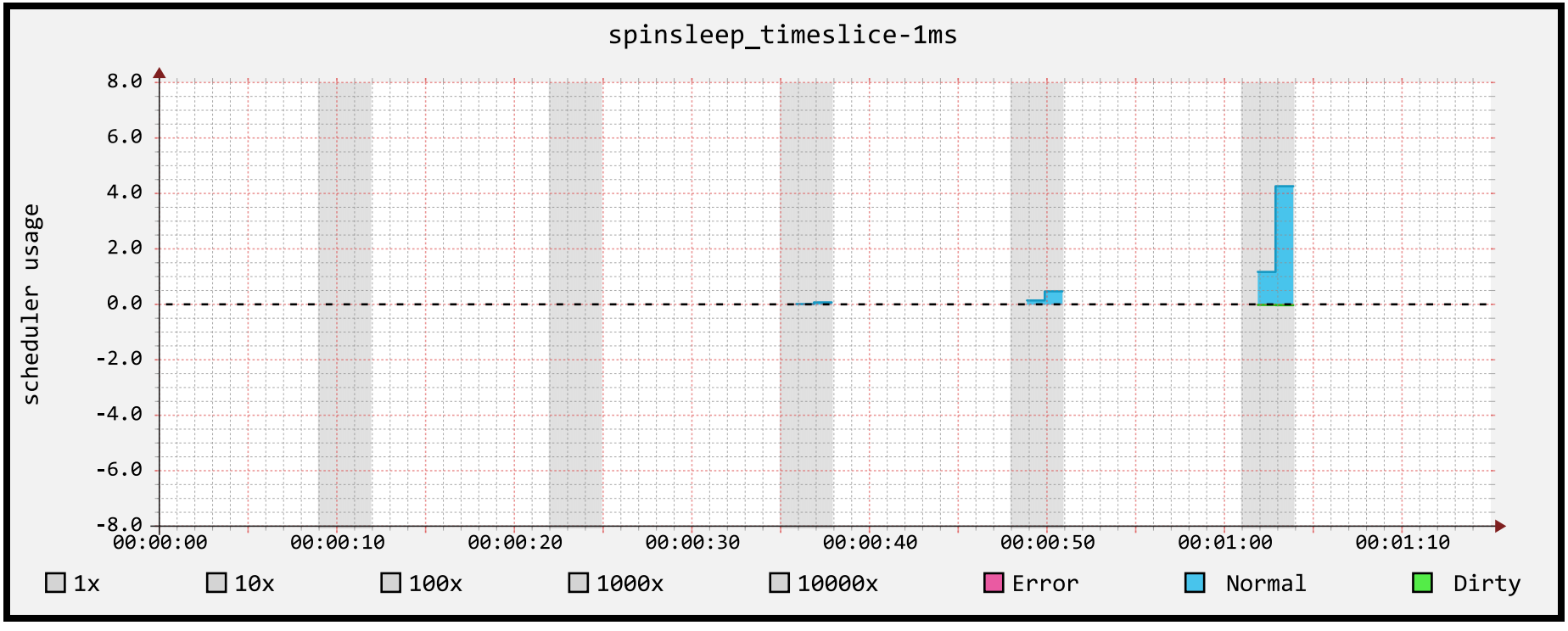
reductions = ~4200

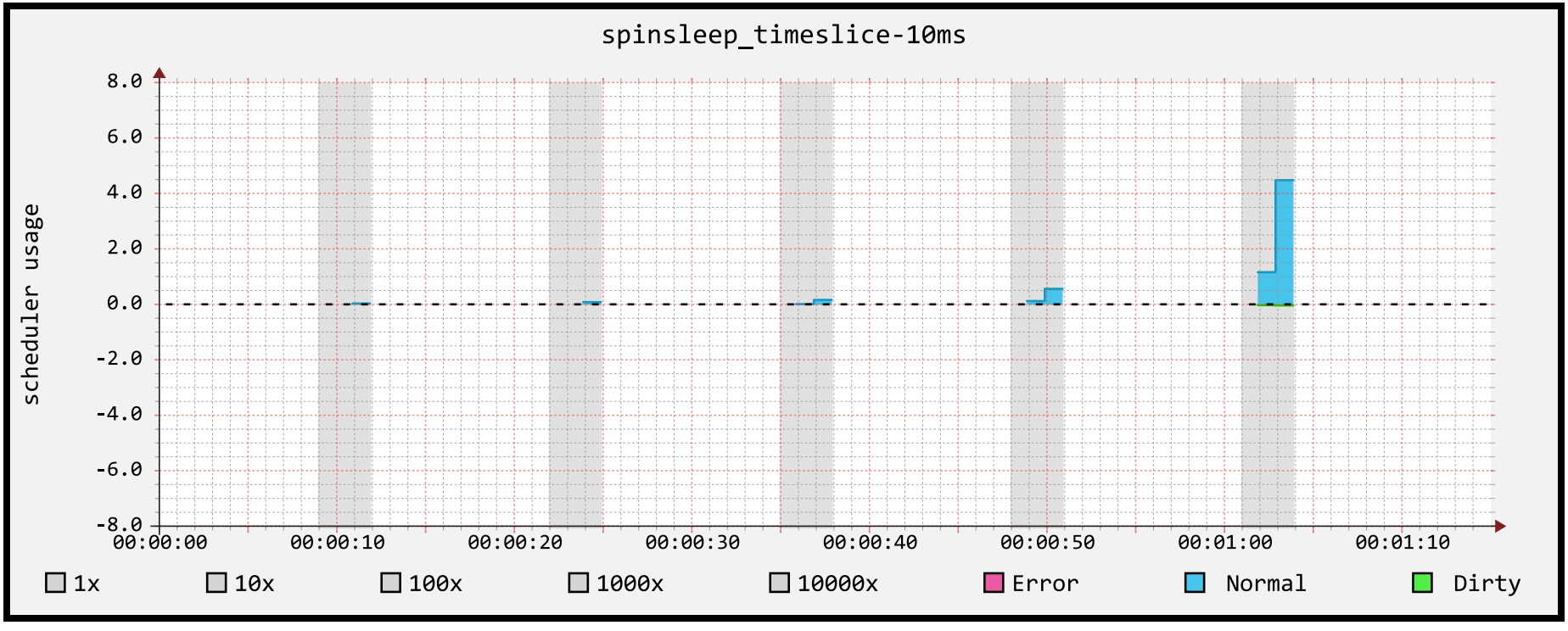
```
if (microseconds > 1000) {
  ERL_NIF_TERM newargv[1];
  newargv[0] = enif_make_int64(env, (ErlNifSInt64) start);
  newargv[1] = enif_make_int64(env, (ErlNifSInt64) stop);
  newargv[2] = enif_make_int64(env, 1000 * 1000);
  return enif_schedule_nif(env, "spinsleep_timeslice", 0,
                           spinsleep_ts, 3, newargv);
}
```

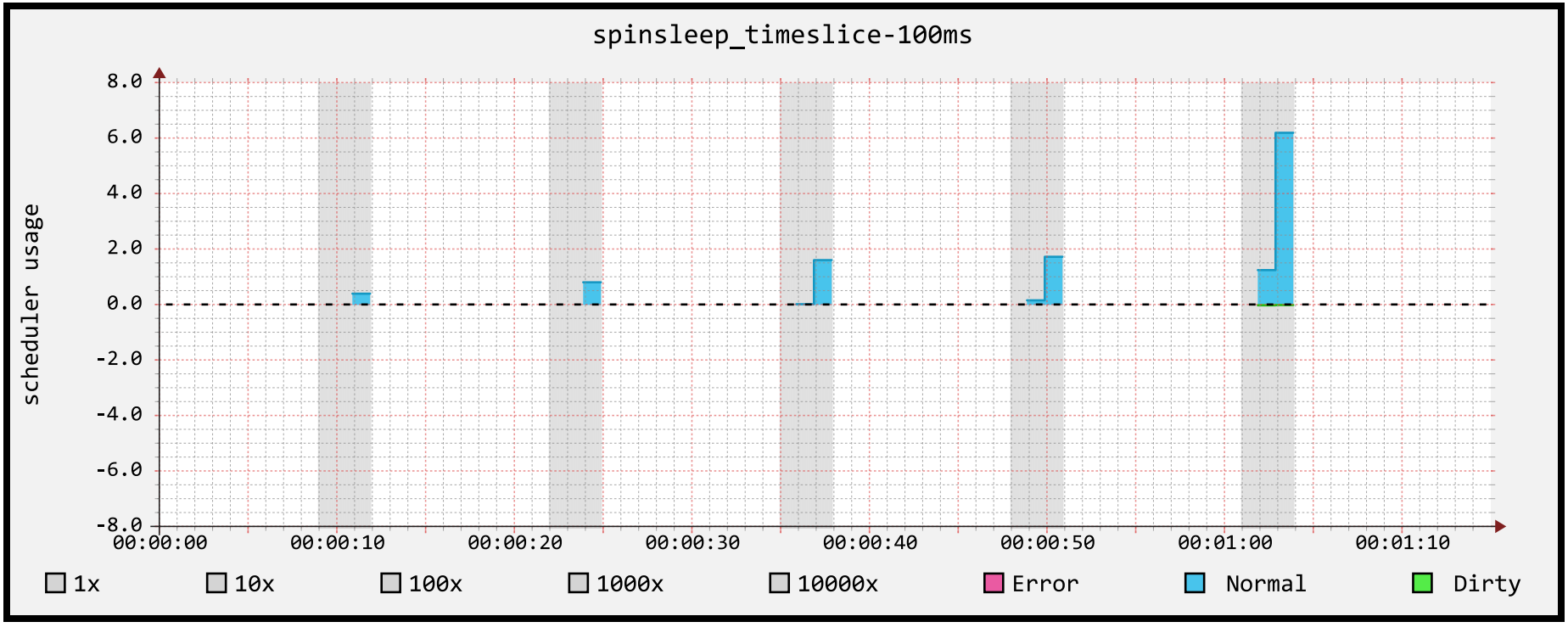
```c
static ERL_NIF_TERM
spinsleep_ts(ErlNifEnv *env, int argc, const ERL_NIF_TERM argv[])
{
  ErlNifTime start, stop, current;
  ErlNifSInt64 max_per_slice, offset = 0;
  int percent, total = 0;
  if (argc != 3
      || !enif_get_int64(env, argv[0], &start)
      || !enif_get_int64(env, argv[1], &stop)
      || !enif_get_int64(env, argv[2], &max_per_slice)) {
    return enif_make_badarg(env);
  }
  // ...
}
```

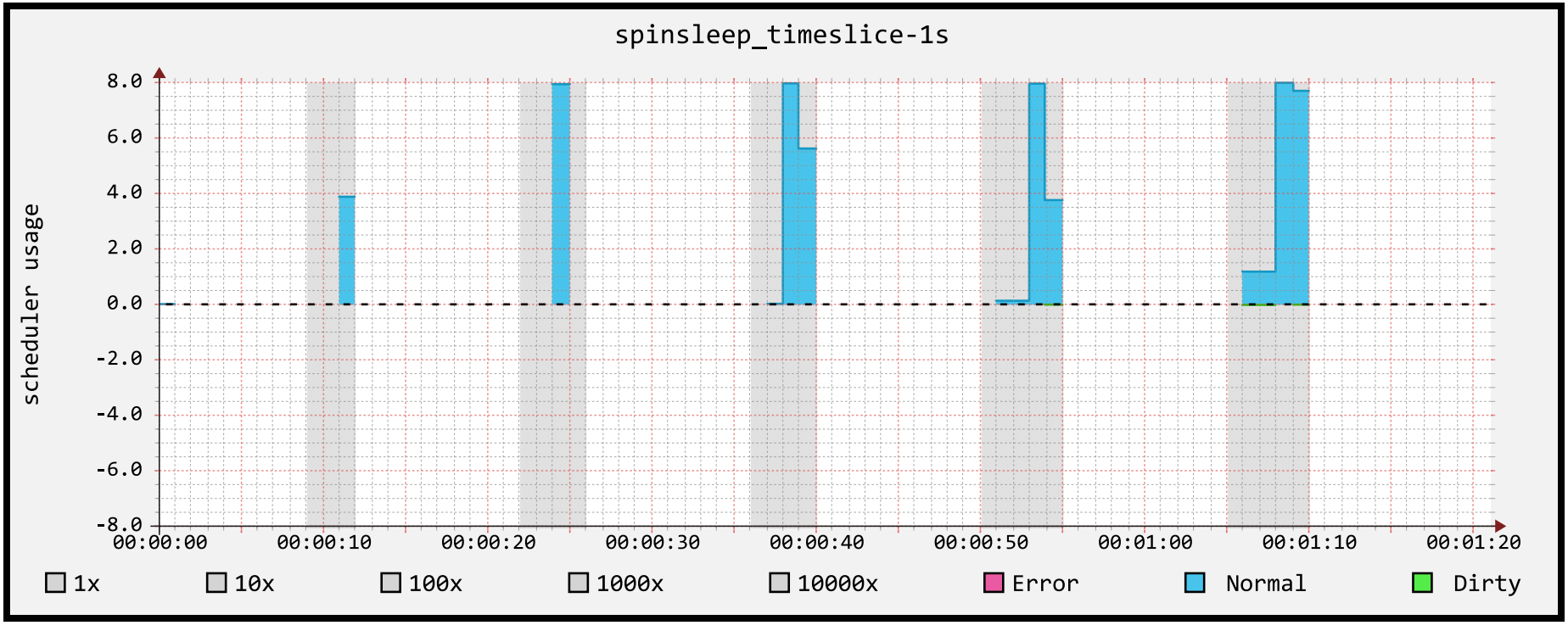
```
current = enif_monotonic_time(ERL_NIF_NSEC);
while (stop > current) {
  (void)spin(max_per_slice);
  offset += max_per_slice;
  diff = enif_monotonic_time(ERL_NIF_NSEC) - current;
  current += diff;
  percent = (int)(diff / 1000 / 1000);
  total += percent;
  if (enif_consume_timeslice(env, percent)) {
    // ...
  }
}
return enif_make_int64(env, (current - start) / 1000);
```

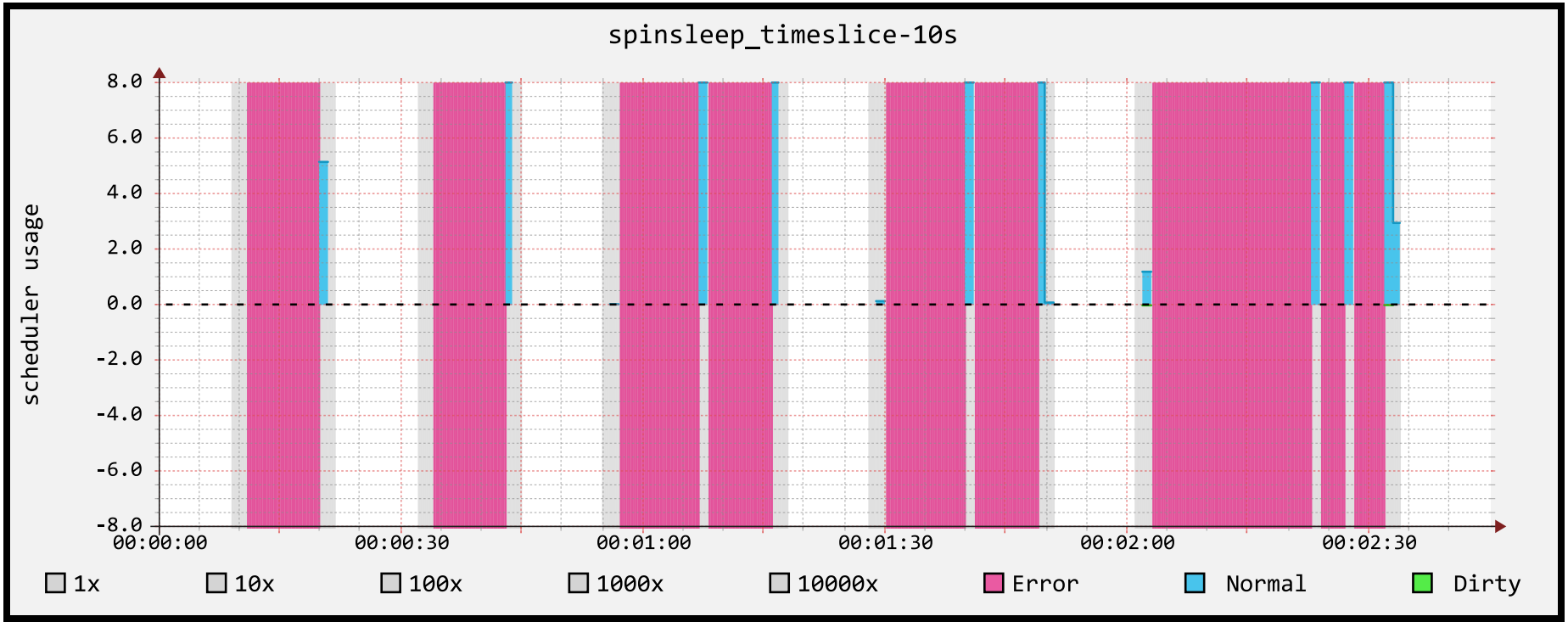
```
max_per_slice = offset;
if (total > 100) {
  int m = (int)(total / 100);
  if (m == 1) {
    max_per_slice -= (max_per_slice * (total - 100) / 100);
  } else {
    max_per_slice = (max_per_slice / m);
  }
}
ERL_NIF_TERM newargv[1];
newargv[0] = argv[0]; // start
newargv[1] = argv[1]; // stop
newargv[2] = enif_make_int64(env, max_per_slice);
return enif_schedule_nif(env, "spinsleep_timeslice", 0,
                         spinsleep_ts, 3, newargv);
```

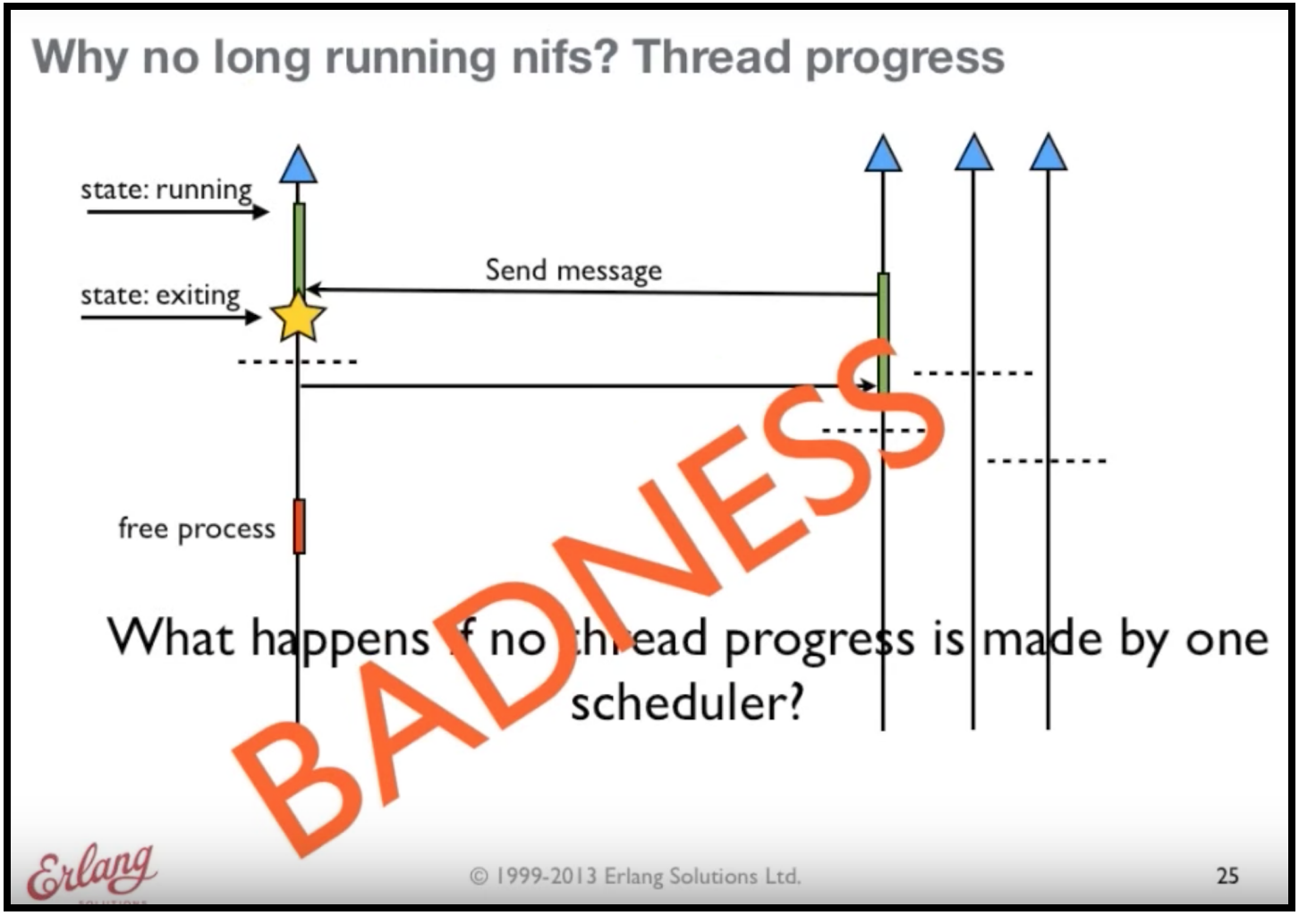

spinsleep_timeslice-1ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice-10ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice-100ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice-1s
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
00:01:20
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice-10s
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:30
00:01:00
00:01:30
00:02:00
00:02:30
1x
10x
100x
1000x
10000x
Error
Normal
Dirty



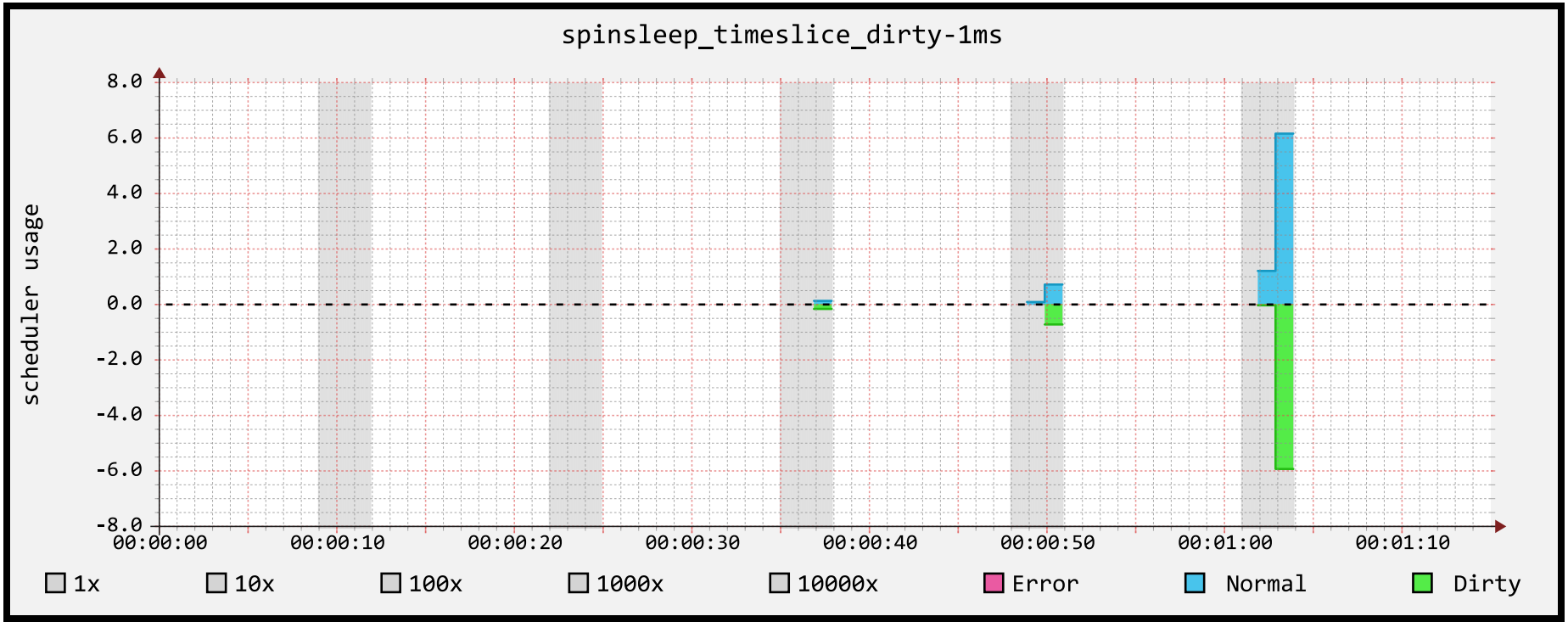
youtu.be/tBAM_N9qPno

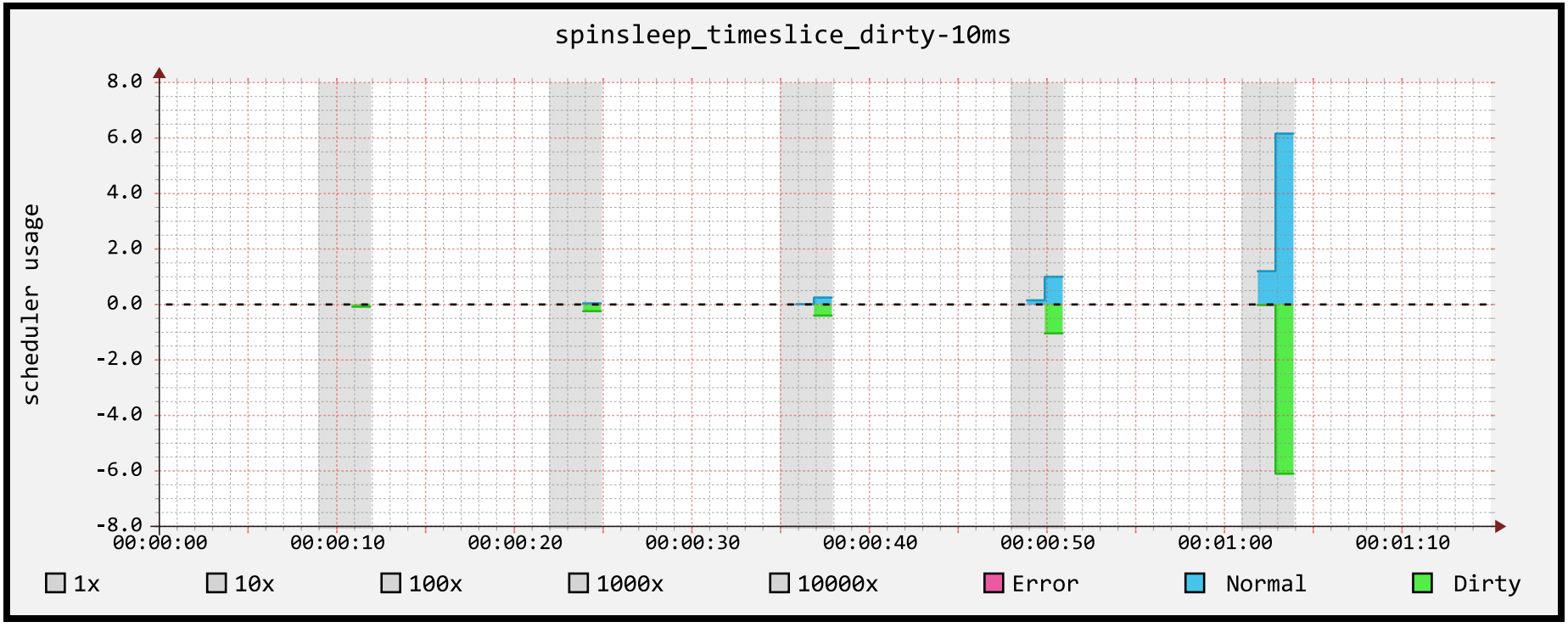
Yielding Dirty NIF

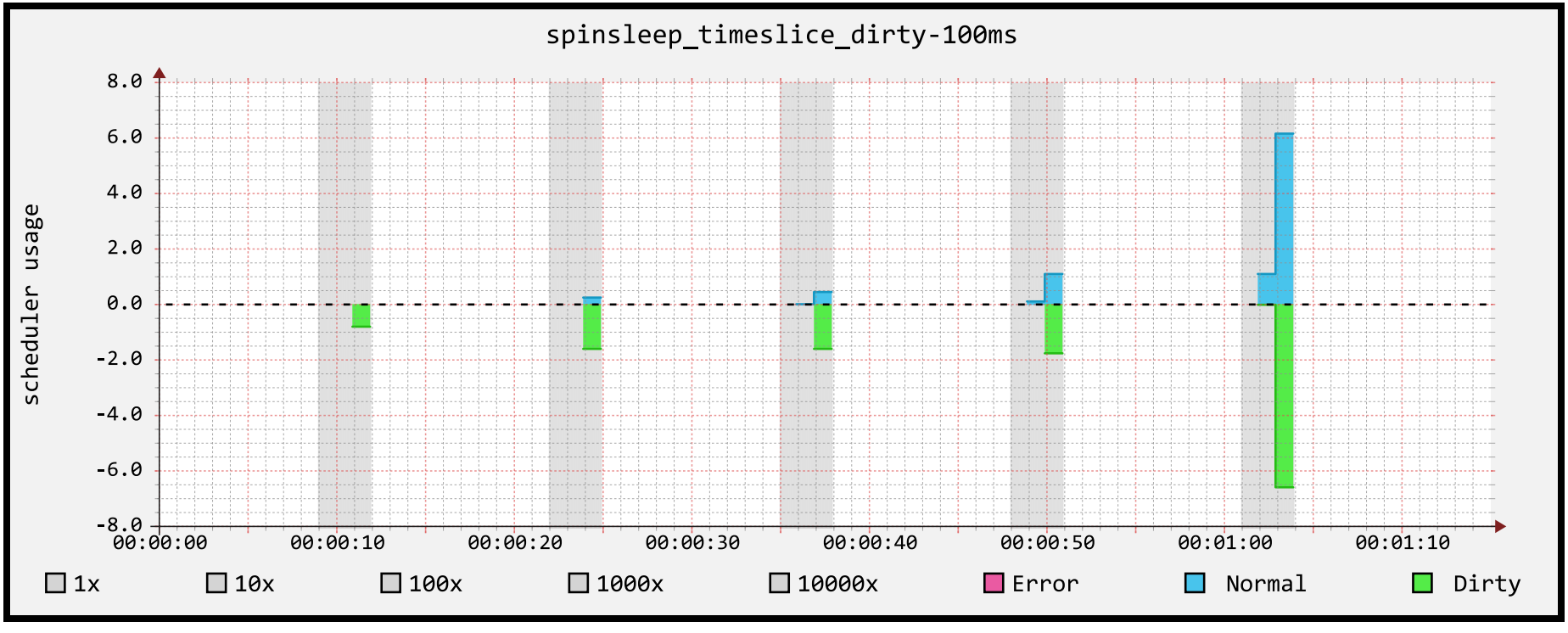
This isn't really mentioned in the documentation.

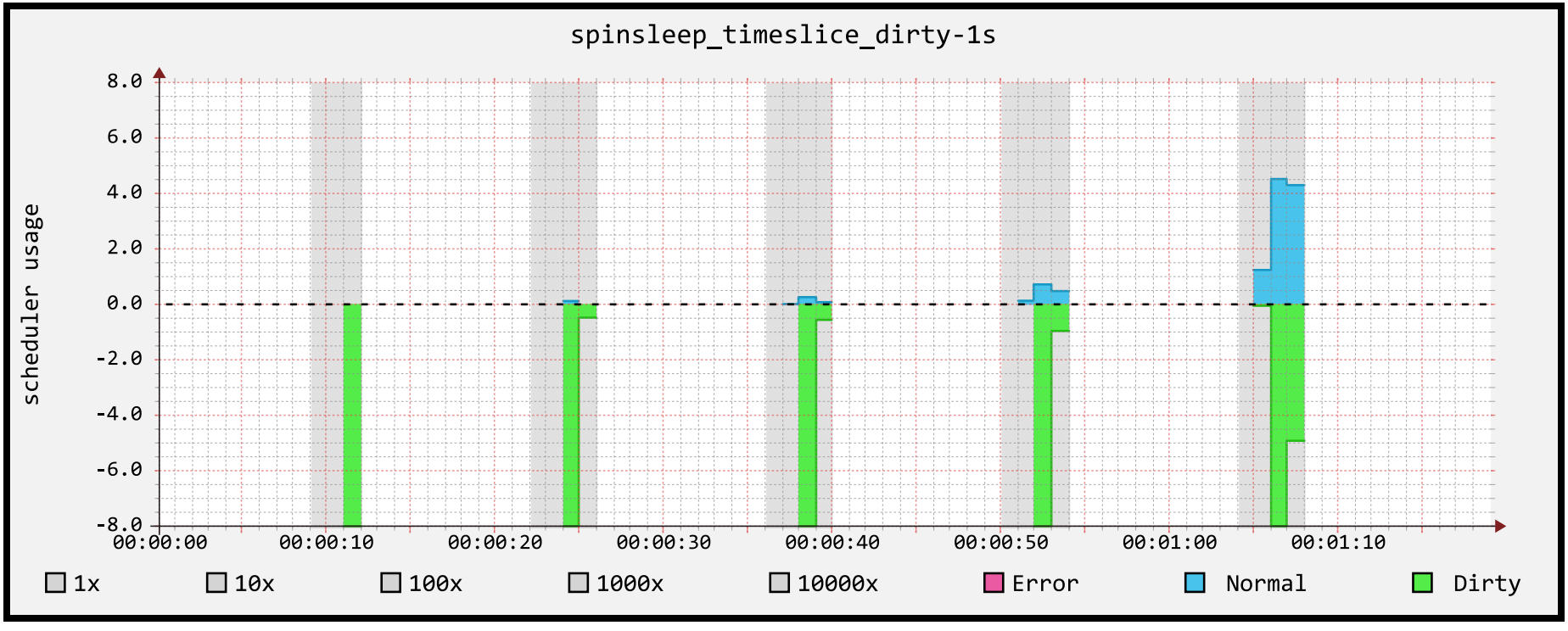
— Me

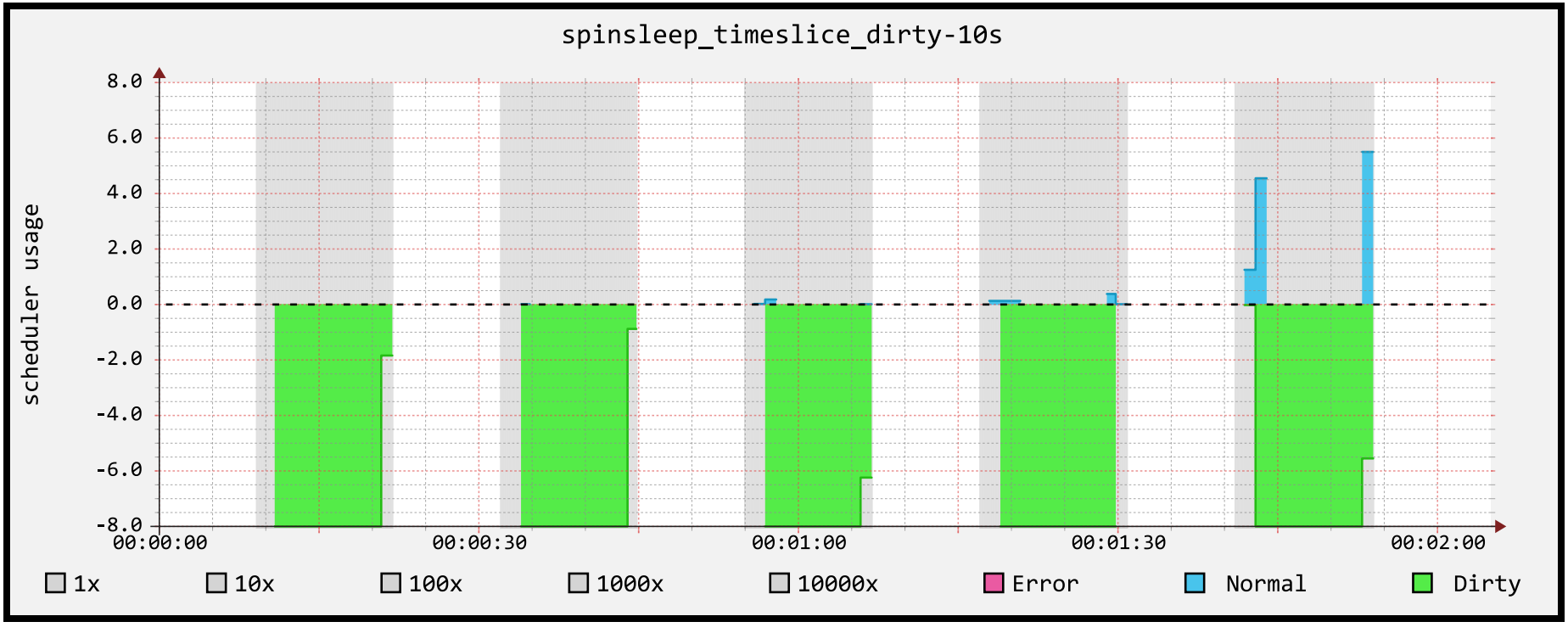
```
return enif_schedule_nif(env, "spinsleep_timeslice_dirty",
                         ERL_NIF_DIRTY_JOB_CPU_BOUND,
                         spinsleep_tsd, 3, newargv);
```

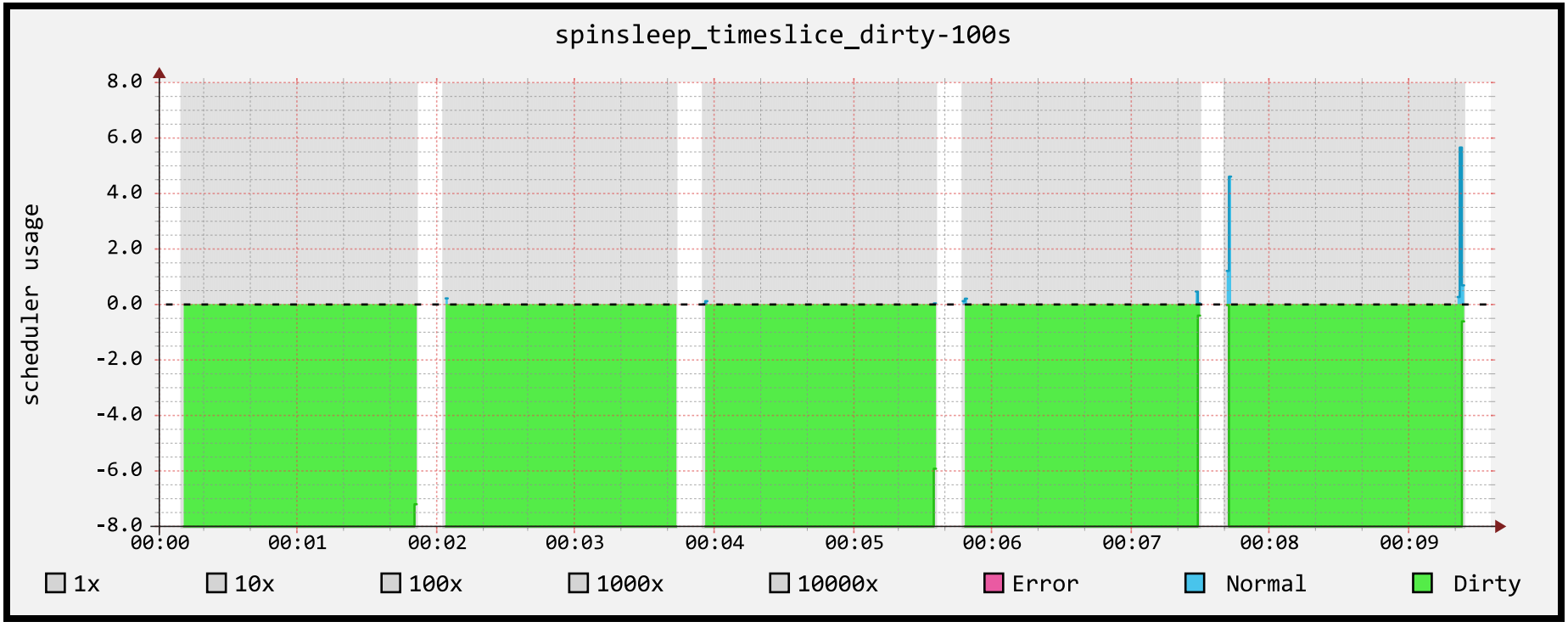

spinsleep_timeslice_dirty-1ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice_dirty-10ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice_dirty-100ms
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice_dirty-1s
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice_dirty-10s
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00:00
00:00:30
00:01:00
00:01:30
00:02:00
1x
10x
100x
1000x
10000x
Error
Normal
Dirty


spinsleep_timeslice_dirty-100s
scheduler usage
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
00:00
00:01
00:02
00:03
00:04
00:05
00:06
00:07
00:08
00:09
1x
10x
100x
1000x
10000x
Error
Normal
Dirty

1 millisecond

	1x	10x	100x	1000x	10000x
Normal				+1s	+15s
Dirty				+1s	+10s
Yielding					
Yielding Dirty					

10 milliseconds

	1x	10x	100x	1000x	10000x
Normal			+1s	+10s	+2m
Dirty			+1s	+10s	+2m
Yielding					
Yielding Dirty					

100 milliseconds

	1x	10x	100x	1000x	10000x
Normal	+1s	+15s	+30s	+2m	+16m
Dirty		+15s	+30s	+2m	+16m
Yielding					
Yielding Dirty					

1 second

	1x	10x	100x	1000x	10000x
Yielding		+1s	+1s	+2s	+3s
Yielding Dirty					

10 seconds

	1x	10x	100x	1000x	10000x
Yielding	+1s	+2s	+5s	+10s	+15s
Yielding Dirty					

100 seconds

	1x	10x	100x	1000x	10000x
Yielding Dirty					

1 millisecond

	1x	10x	100x	1000x	10000x
Normal				+1s	+15s
Dirty				+1s	+10s
Yielding					
Yielding Dirty					

10 milliseconds

	1x	10x	100x	1000x	10000x
Normal			+1s	+10s	+2m
Dirty			+1s	+10s	+2m
Yielding					
Yielding Dirty					

100 milliseconds

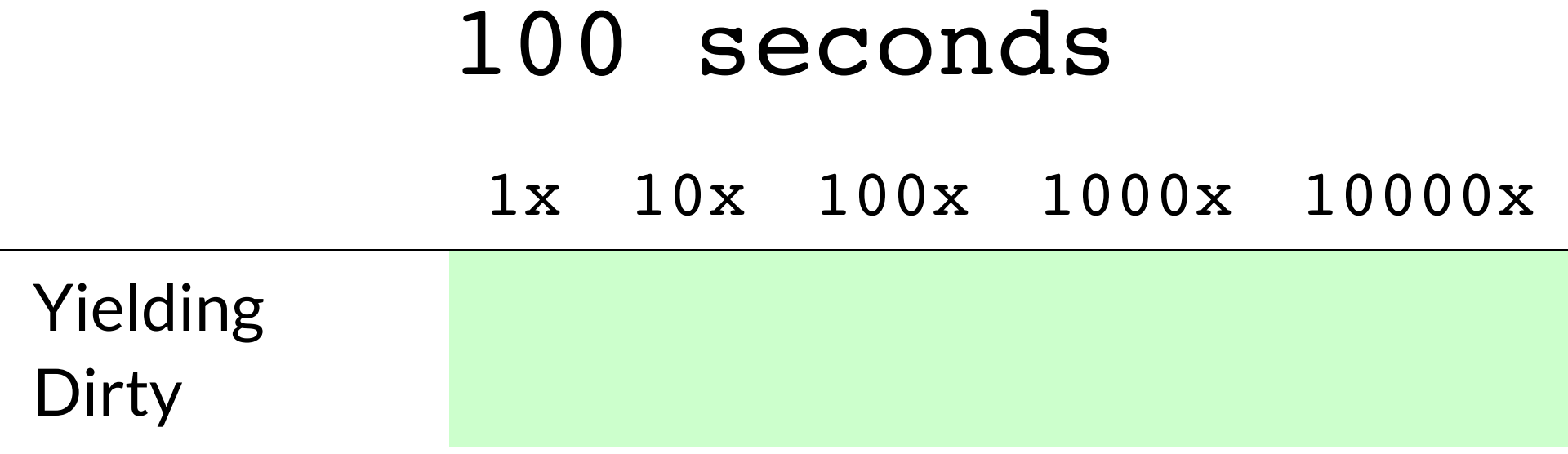
	1x	10x	100x	1000x	10000x
Normal	+1s	+15s	+30s	+2m	+16m
Dirty		+15s	+30s	+2m	+16m
Yielding					
Yielding Dirty					

1 second

	1x	10x	100x	1000x	10000x
Yielding		+1s	+1s	+2s	+3s
Yielding Dirty					

10 seconds

	1x	10x	100x	1000x	10000x
Yielding	+1s	+2s	+5s	+10s	+15s
Yielding Dirty					

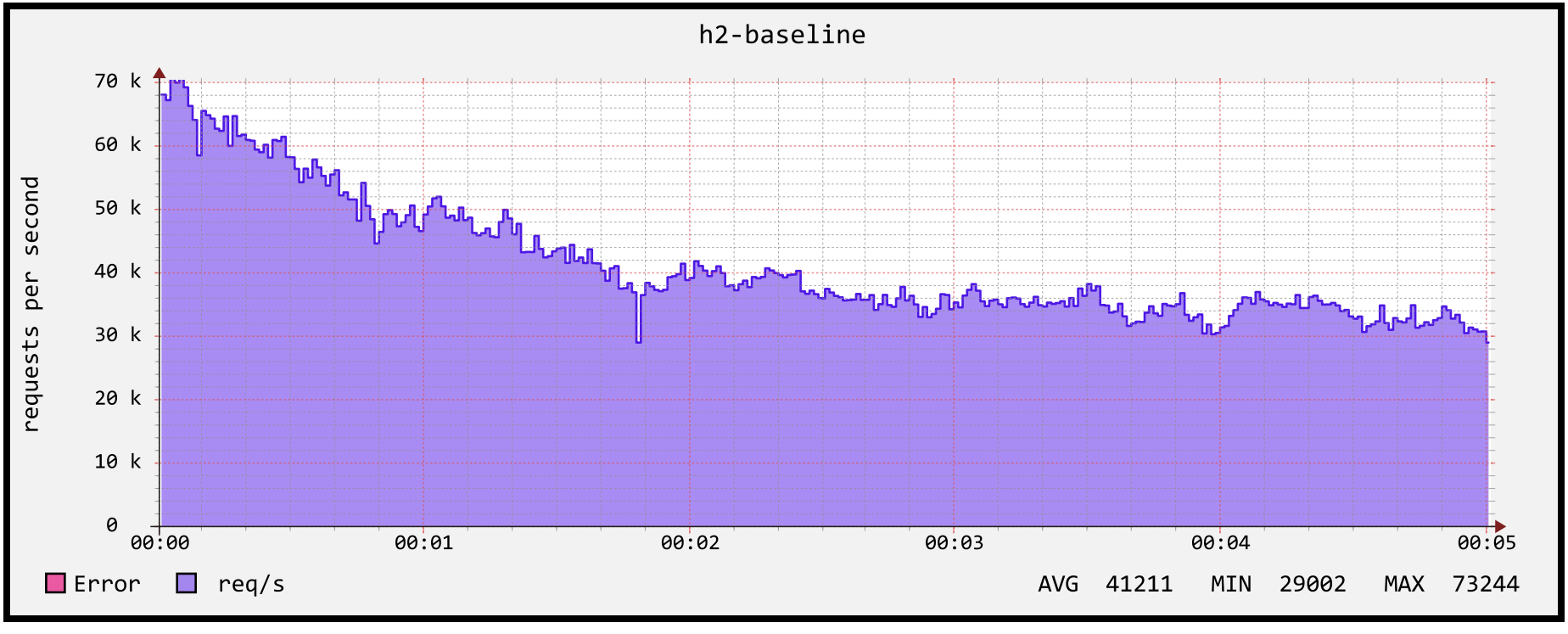

100 seconds
1x
10x
100x
1000x
10000x
Yielding
Dirty

Recommendations

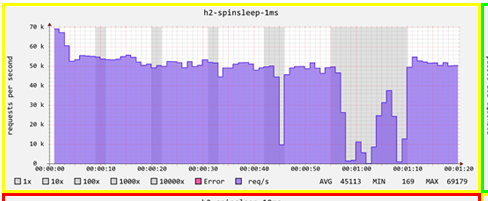
Duration	NIF Type
< 1ms	Normal
< 100ms	Yielding
≥ 100ms	Yielding Dirty

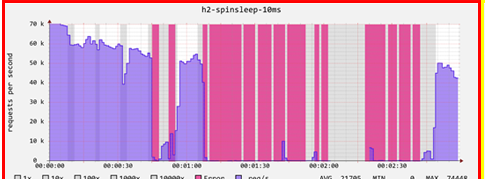
Real World Effects

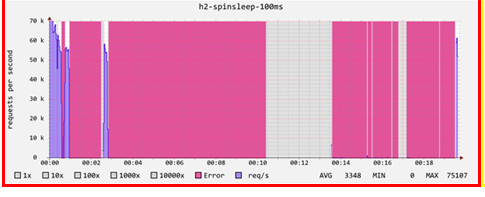
Load Testing cowboy 2 (HTTP/2)



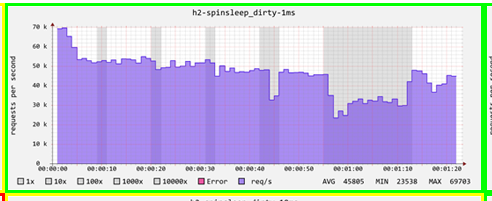
Normal

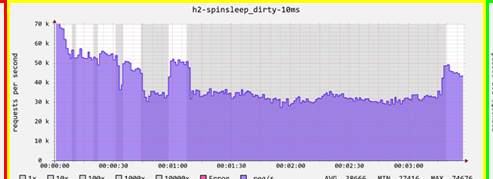


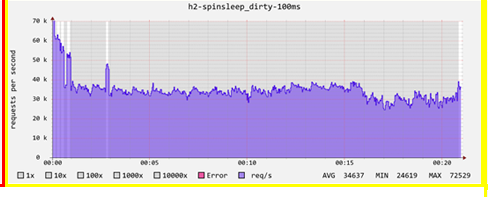




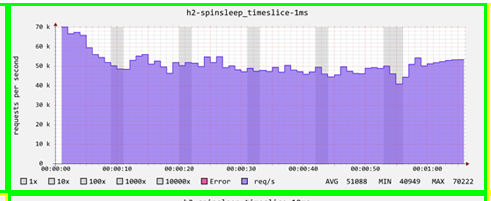
Dirty

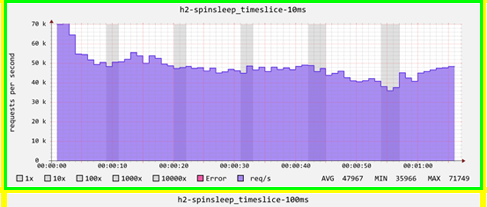


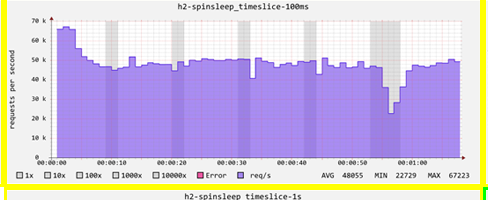


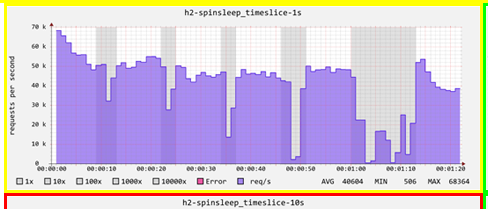


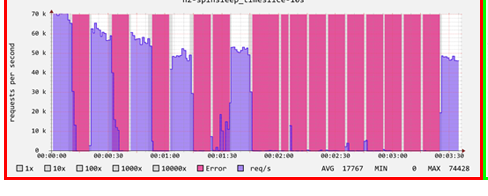
Yielding



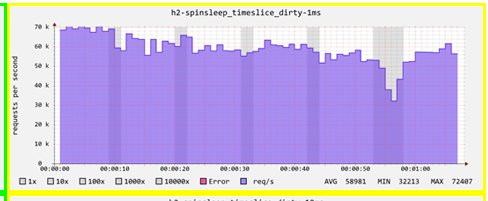


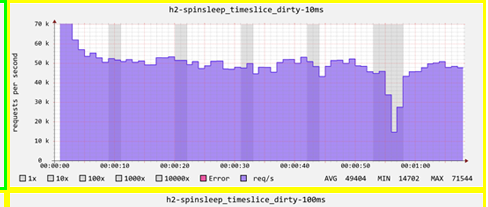


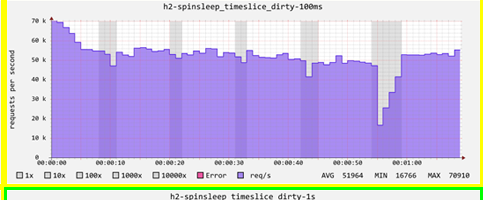


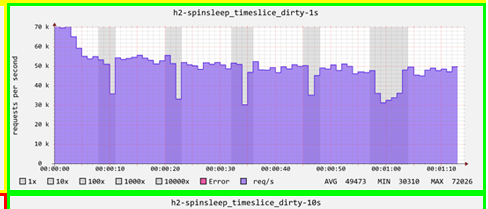


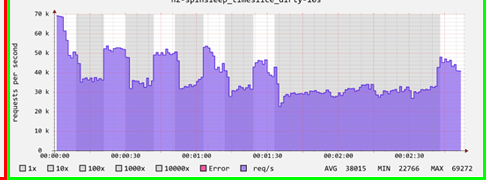
Yielding Dirty





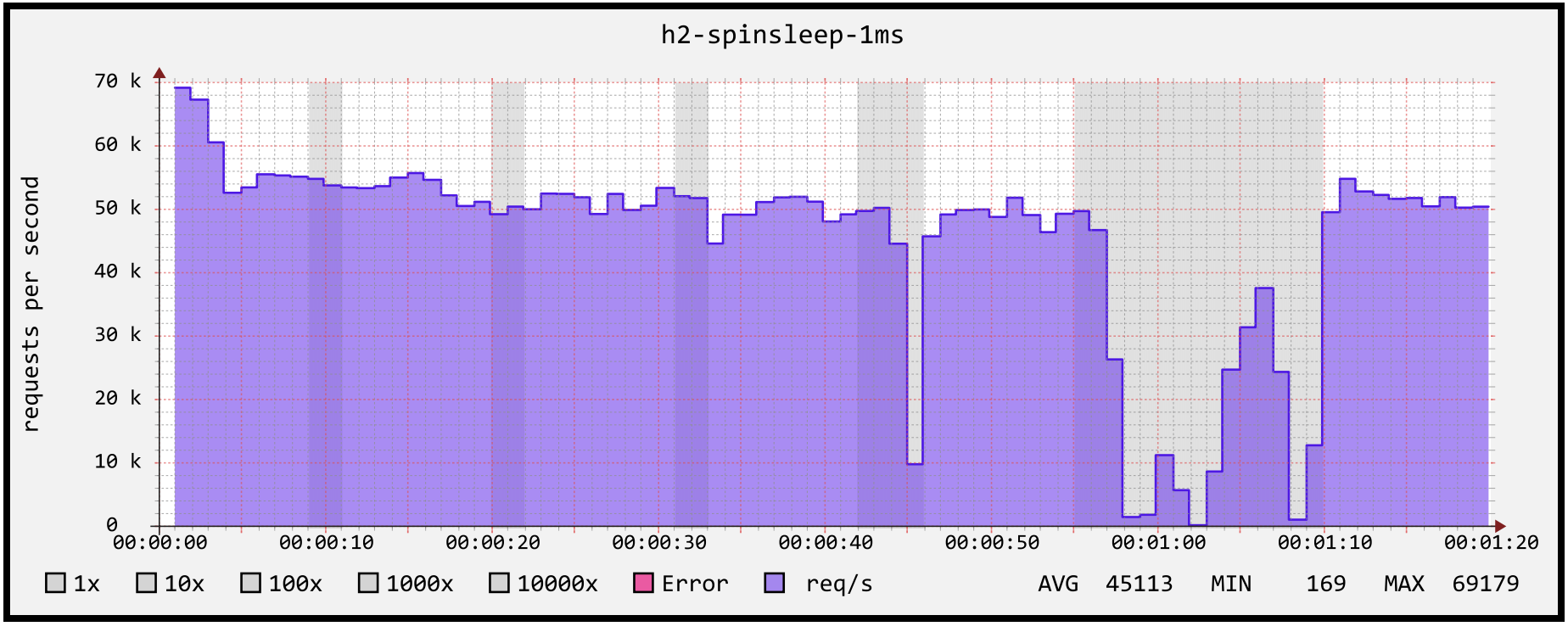






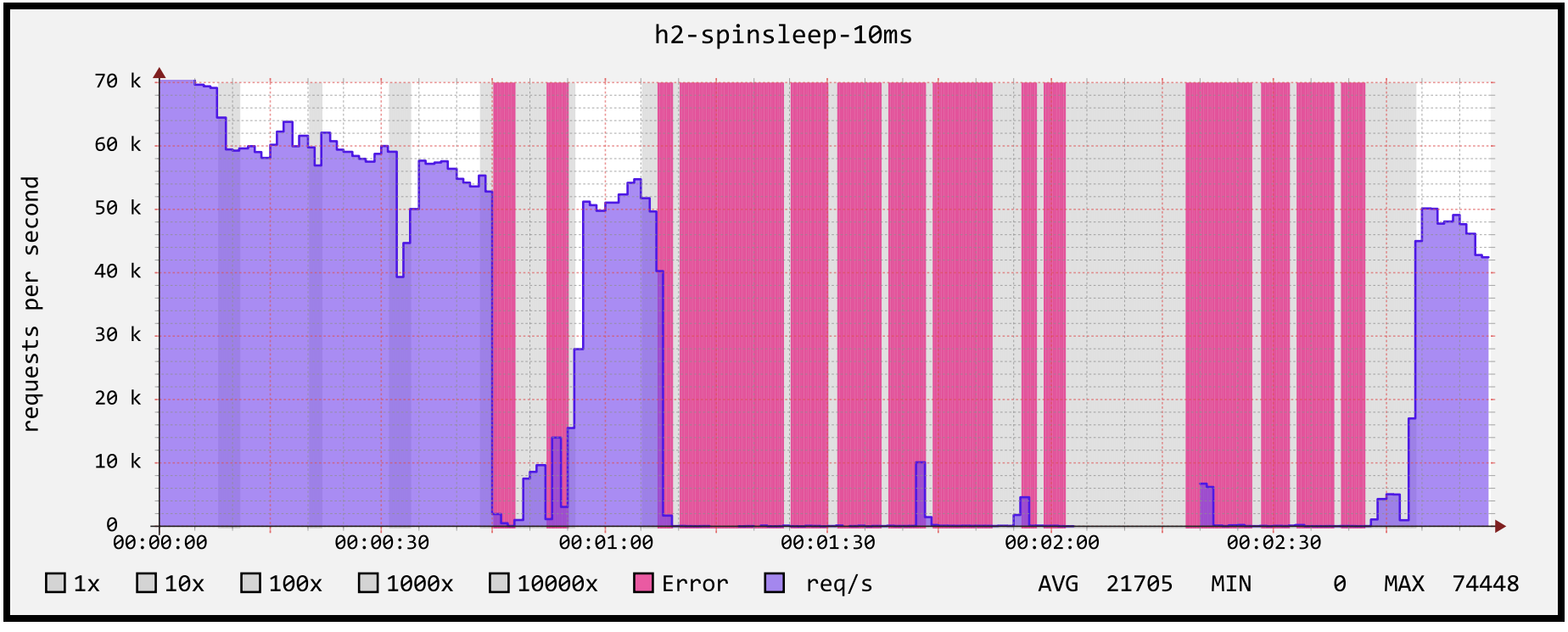
Normal NIF

cowboy 2 (HTTP/2) with 1ms



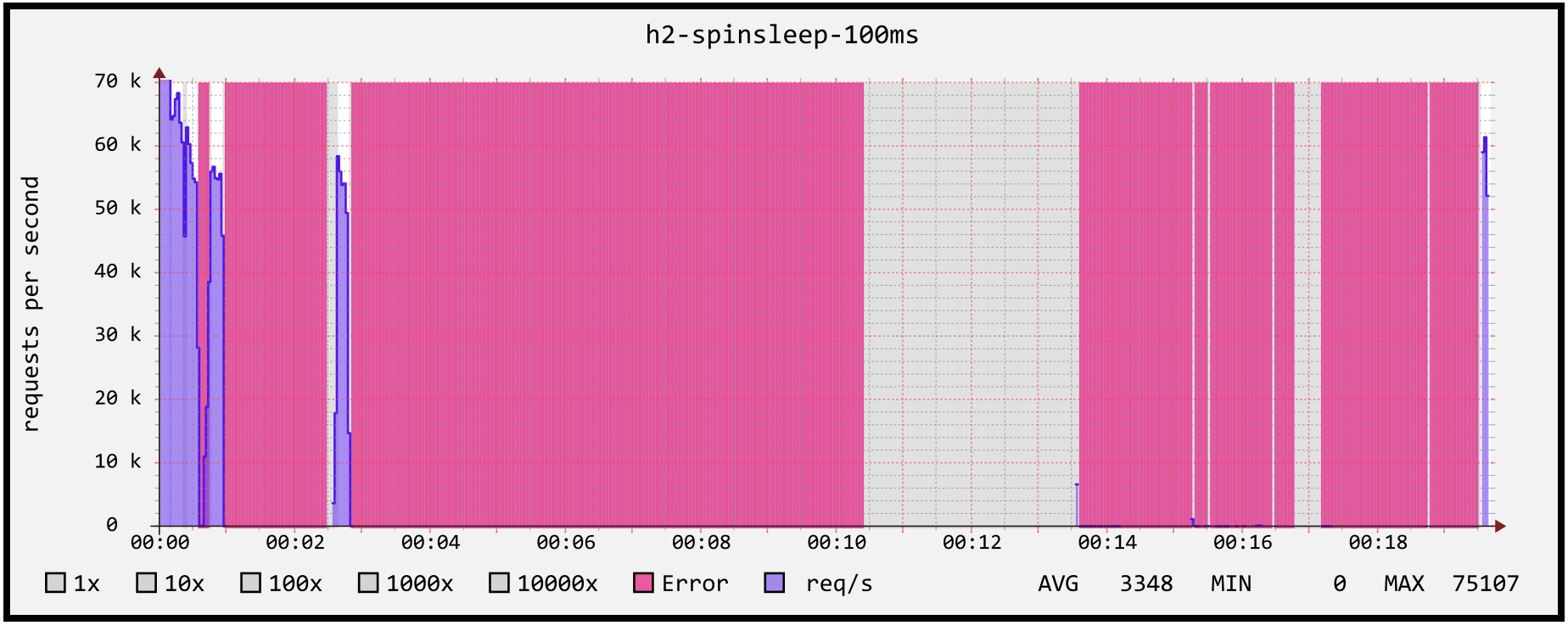
Normal NIF

cowboy 2 (HTTP/2) with 10ms



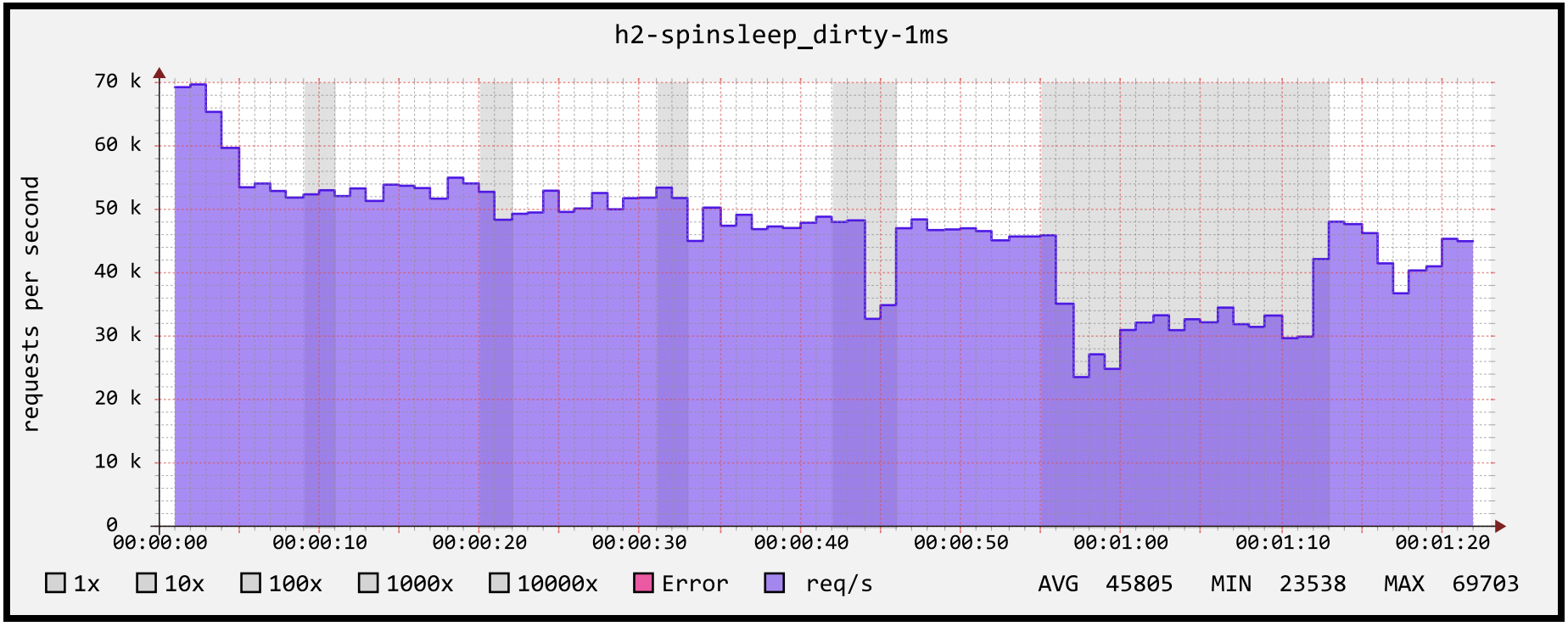
Normal NIF

cowboy 2 (HTTP/2) with 100ms



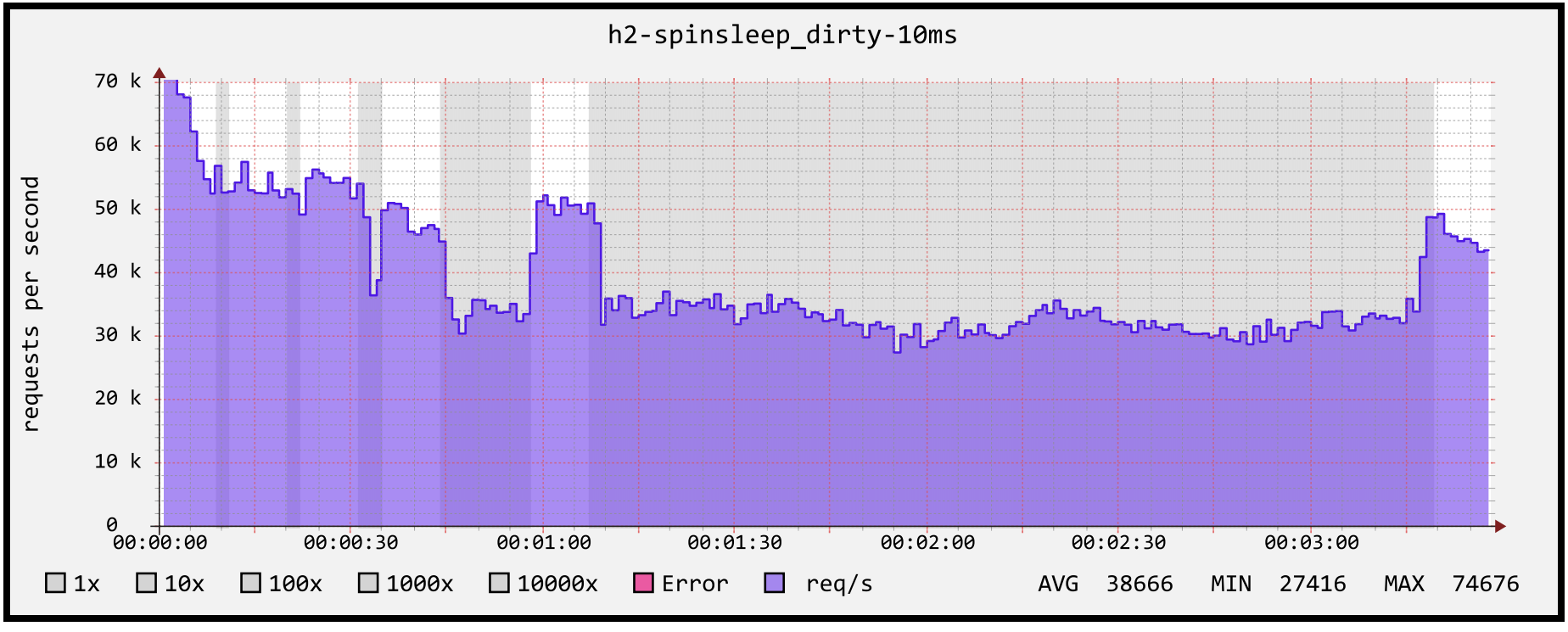
Dirty NIF

cowboy 2 (HTTP/2) with 1ms



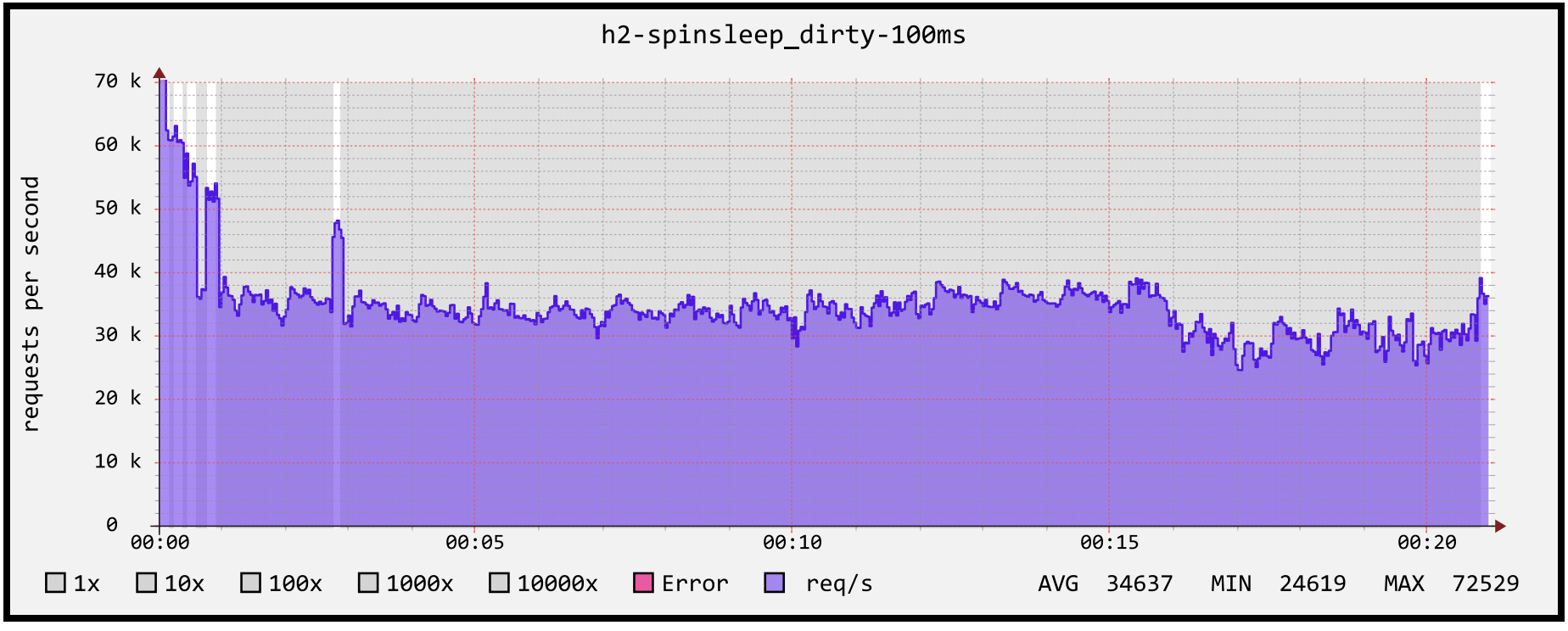
Dirty NIF

cowboy 2 (HTTP/2) with 10ms



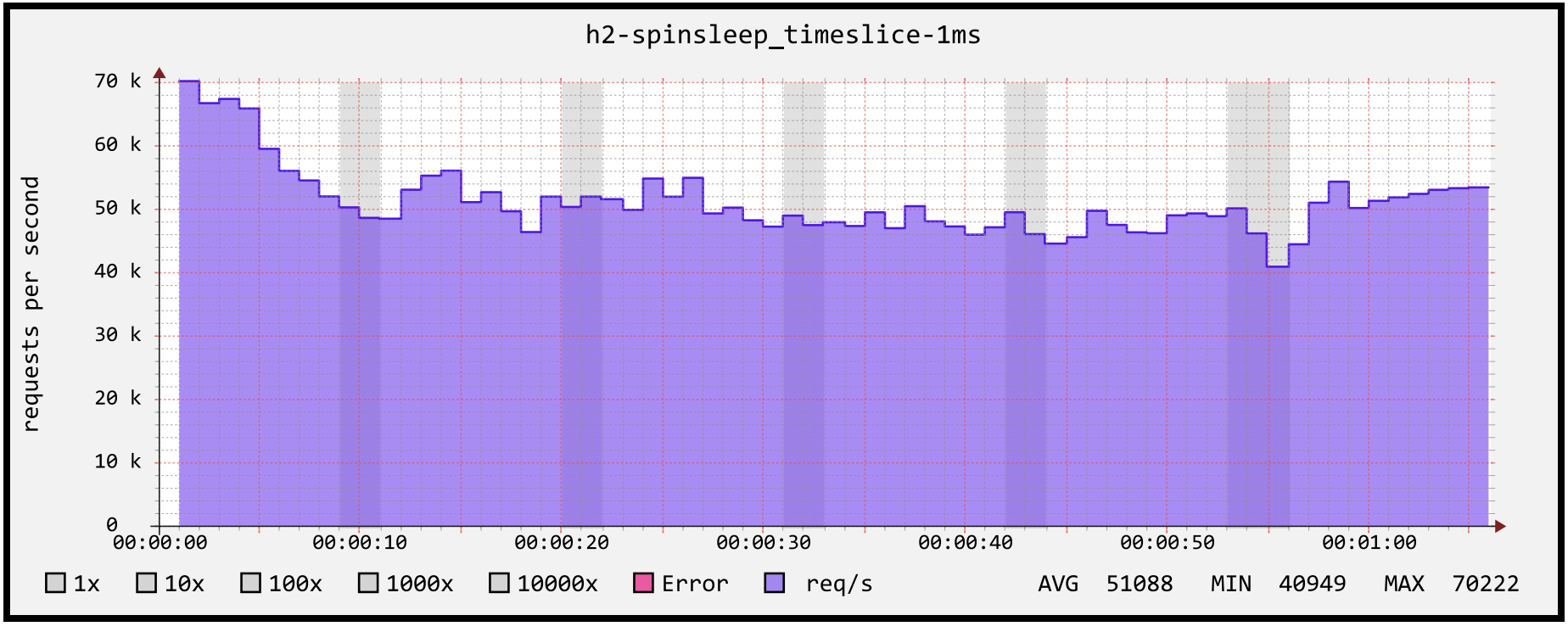
Dirty NIF

cowboy 2 (HTTP/2) with 100ms



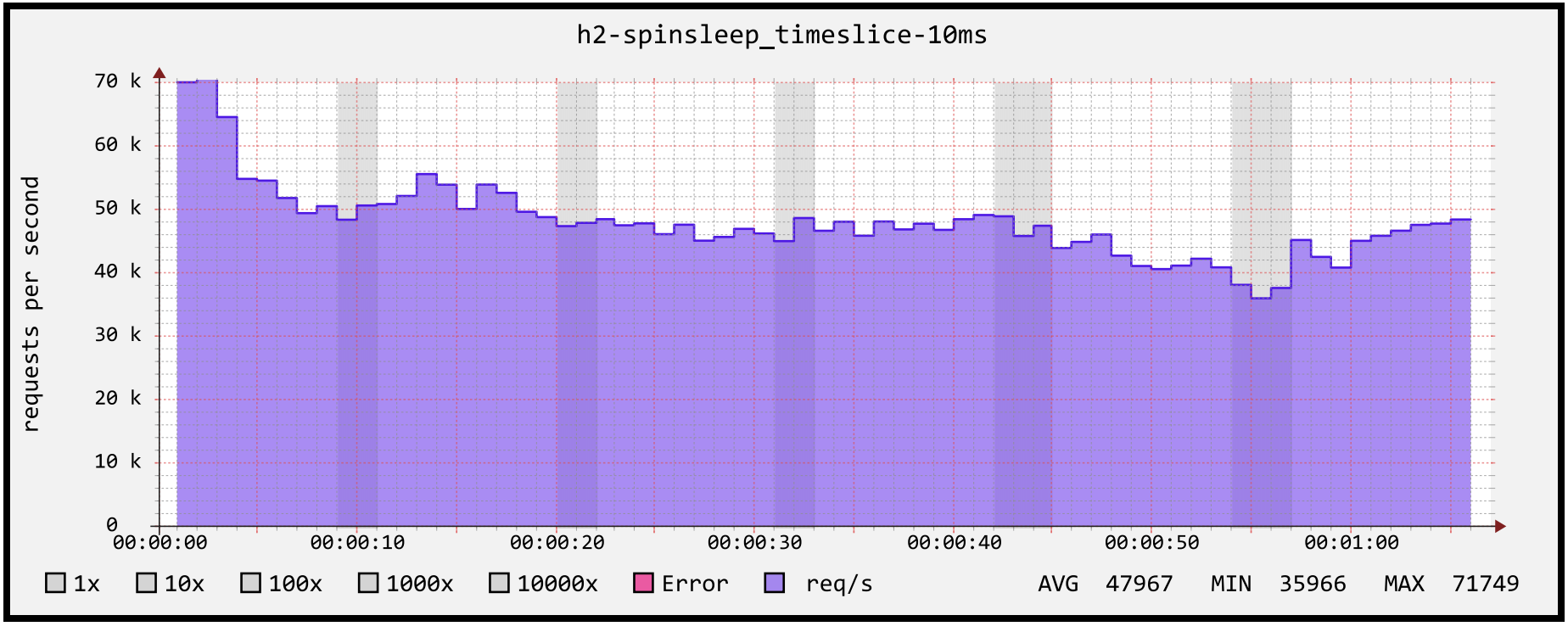
Yielding NIF

cowboy 2 (HTTP/2) with 1ms



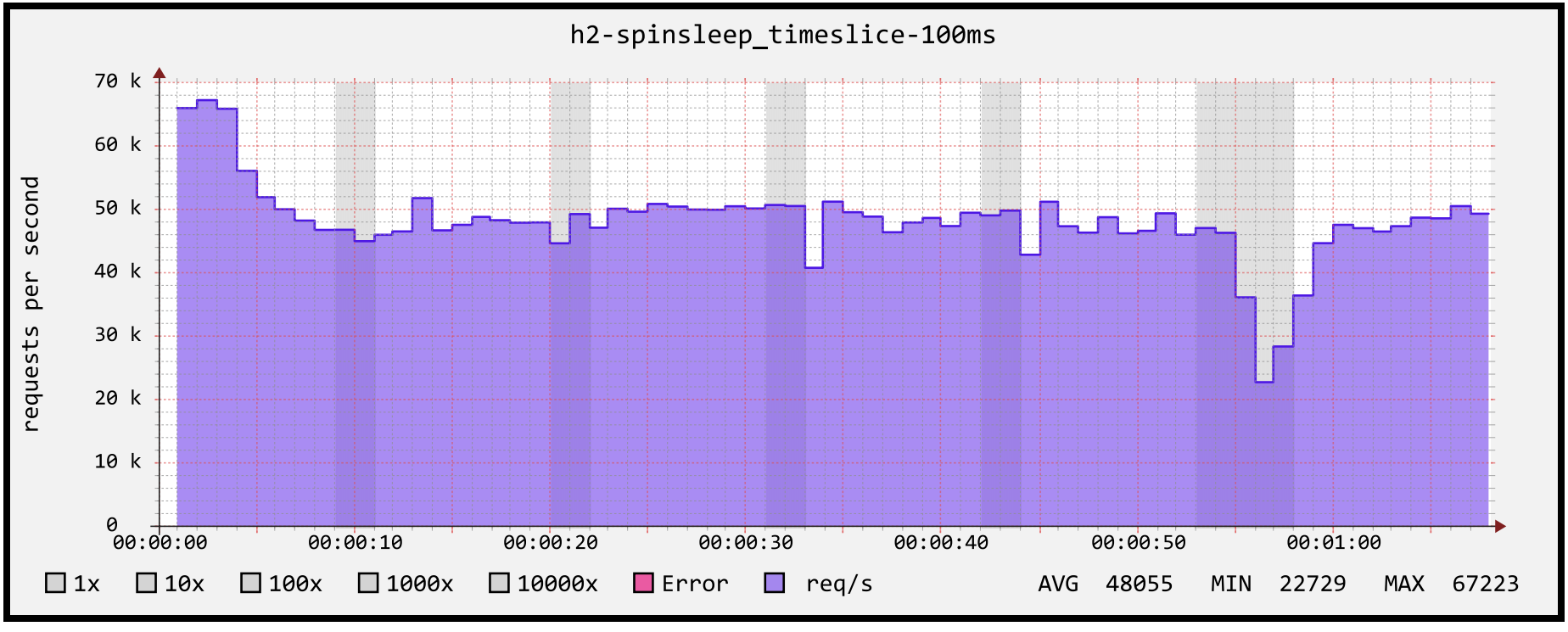
Yielding NIF

cowboy 2 (HTTP/2) with 10ms



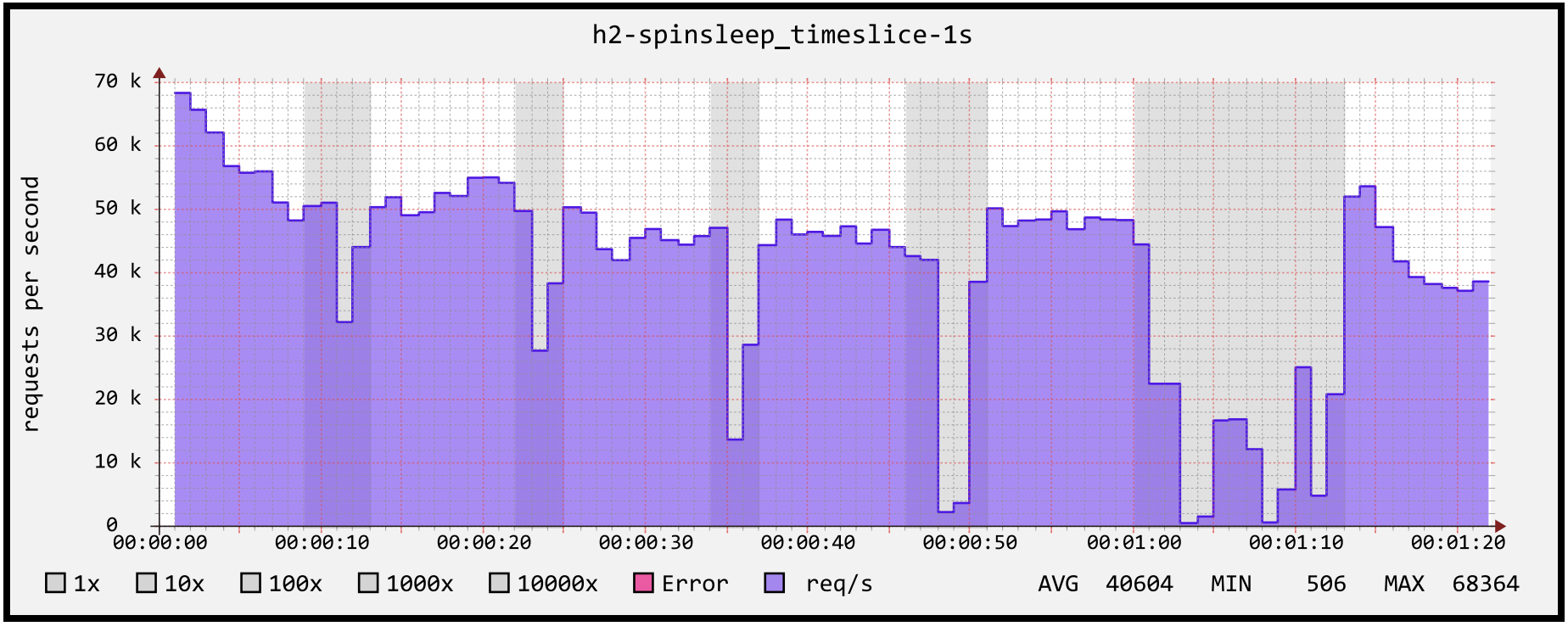
Yielding NIF

cowboy 2 (HTTP/2) with 100ms



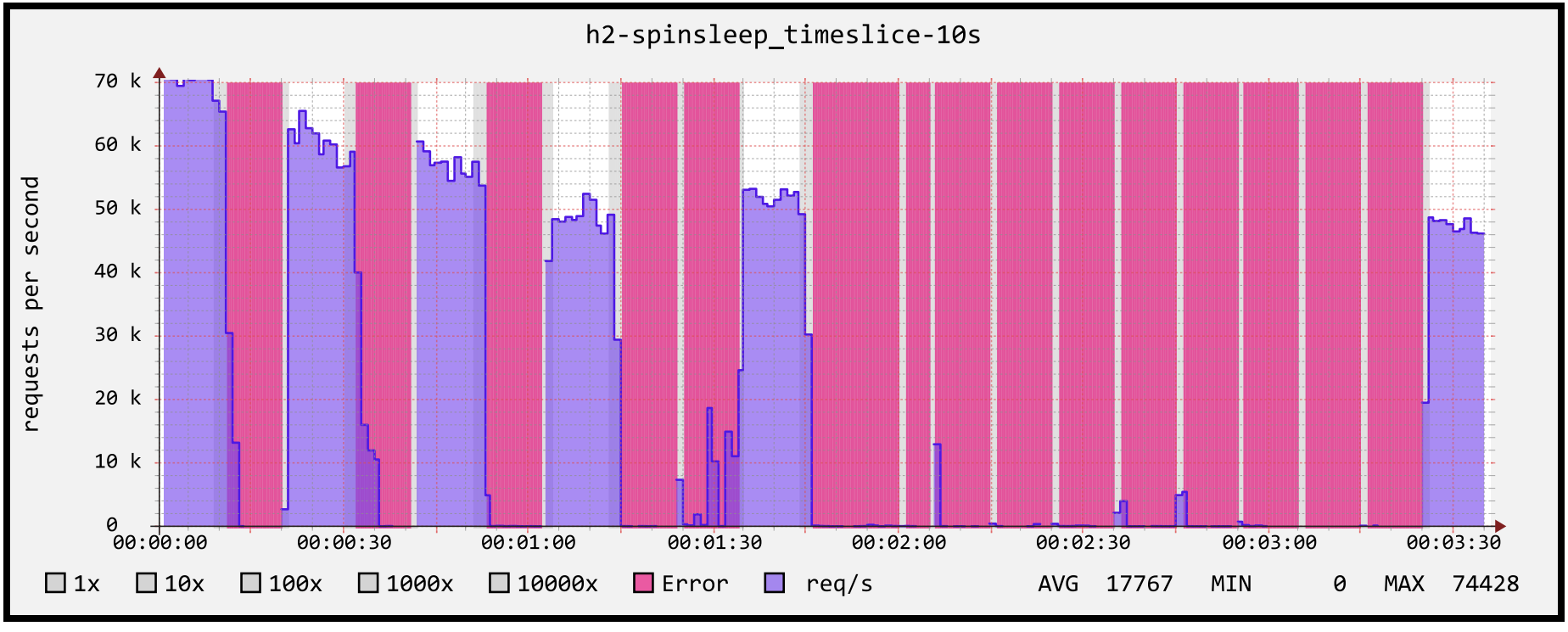
Yielding NIF

cowboy 2 (HTTP/2) with 1s



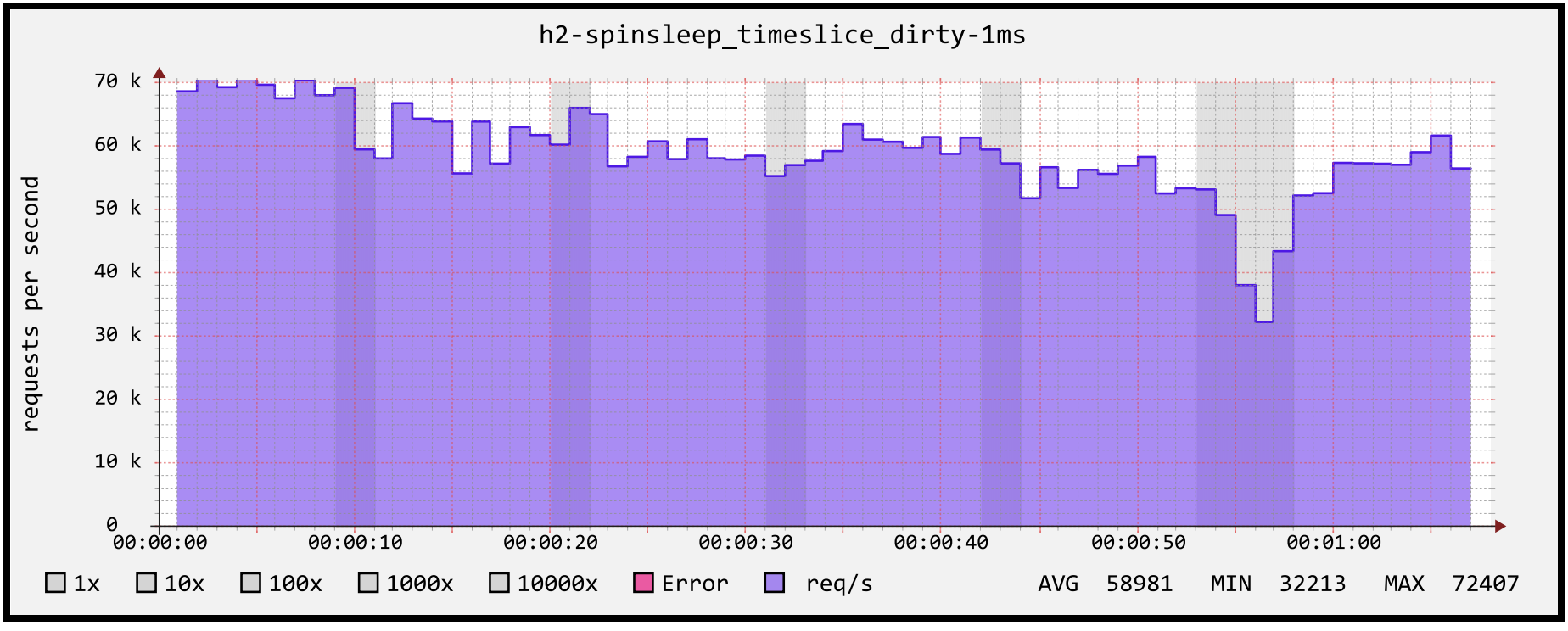
Yielding NIF

cowboy 2 (HTTP/2) with 10s



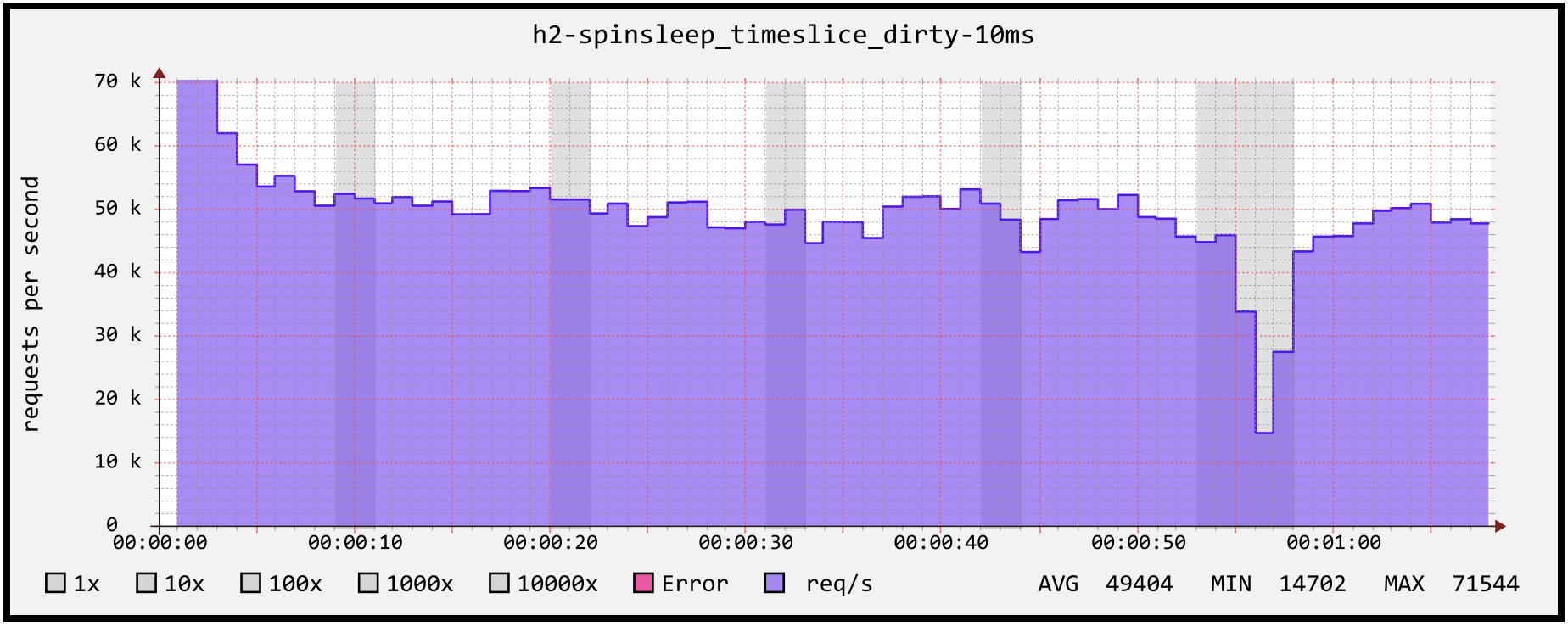
Yielding Dirty NIF

cowboy 2 (HTTP/2) with 1ms



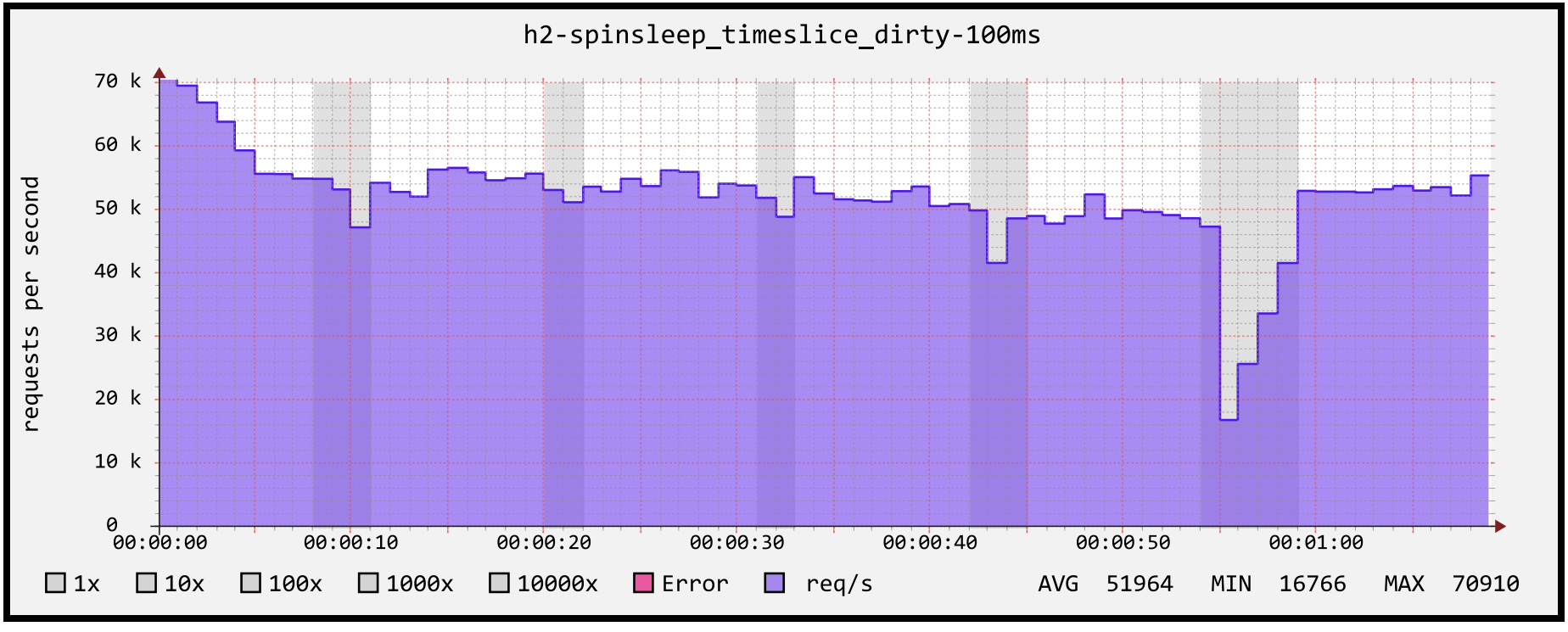
Yielding Dirty NIF

cowboy 2 (HTTP/2) with 10ms



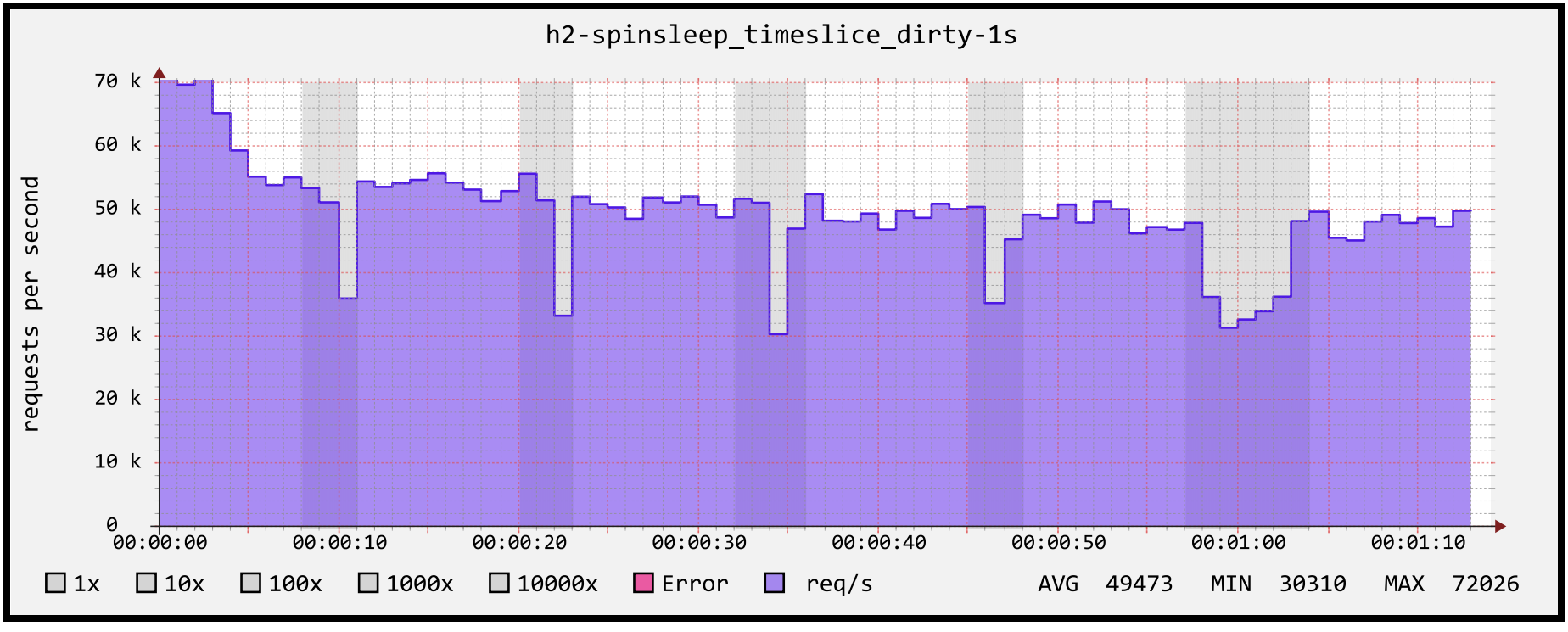
Yielding Dirty NIF

cowboy 2 (HTTP/2) with 100ms



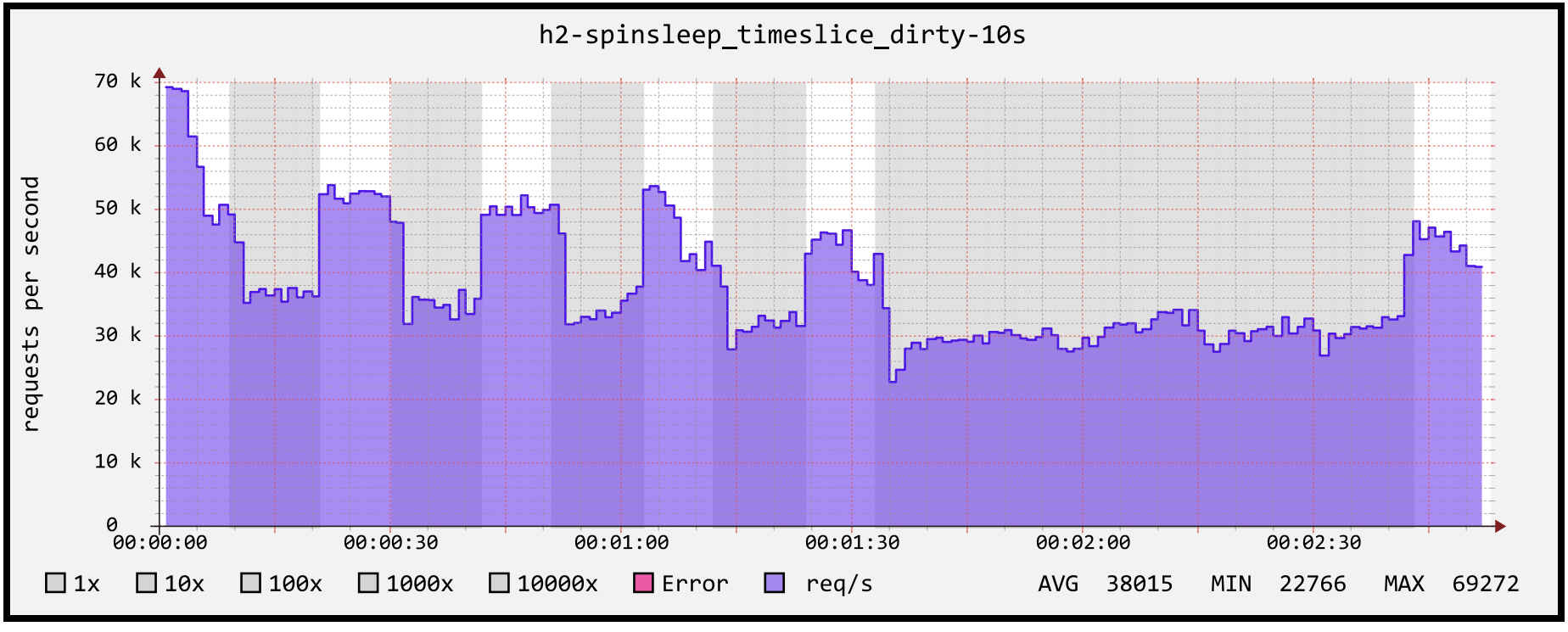
Yielding Dirty NIF

cowboy 2 (HTTP/2) with 1s



Yielding Dirty NIF

cowboy 2 (HTTP/2) with 10s



Examples in OTP

`crypto.c`

- NIF timeslice examples
- One dirty example:
  `crypto:rsa_generate_key_nif/2`

Examples in OTP

`erl_bif_binary.c`

- BIF timeslice examples
  - `binary:match/2,3`
  - `binary:matches/2,3`
  - `binary:split/2,3`
- More in #1480

NIF features in OTP-20

- Magic references instead of magic binaries
- `enif_whereis_pid`
- `enif_monitor_process`
- `enif_select`

(possible)

Network I/O Replacement

`sverk_tcp_nif.c`

- Example by Sverker Eriksson
- Uses `enif_select`
- NIF I/O Queue API: #1364

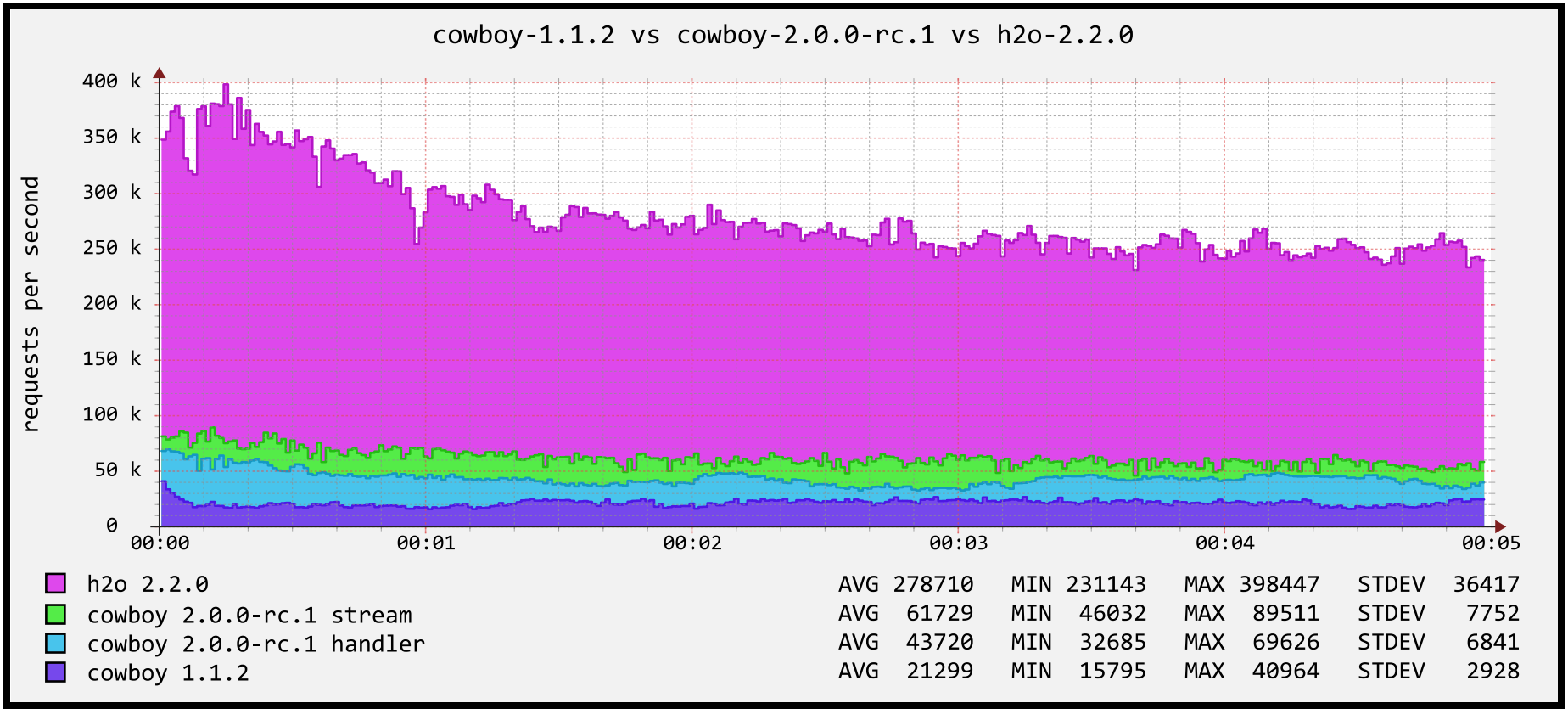
(possible)

TTY Replacement

`ttsl_nif.c`

- Example by Sverker Eriksson
- Uses `enif_select`

Fun with Threaded NIFs



potatosalad.io/2017/08/20/load-testing-cowboy-2-0-0-rc-1

potatosalad/elixirconf2017