

perf stories

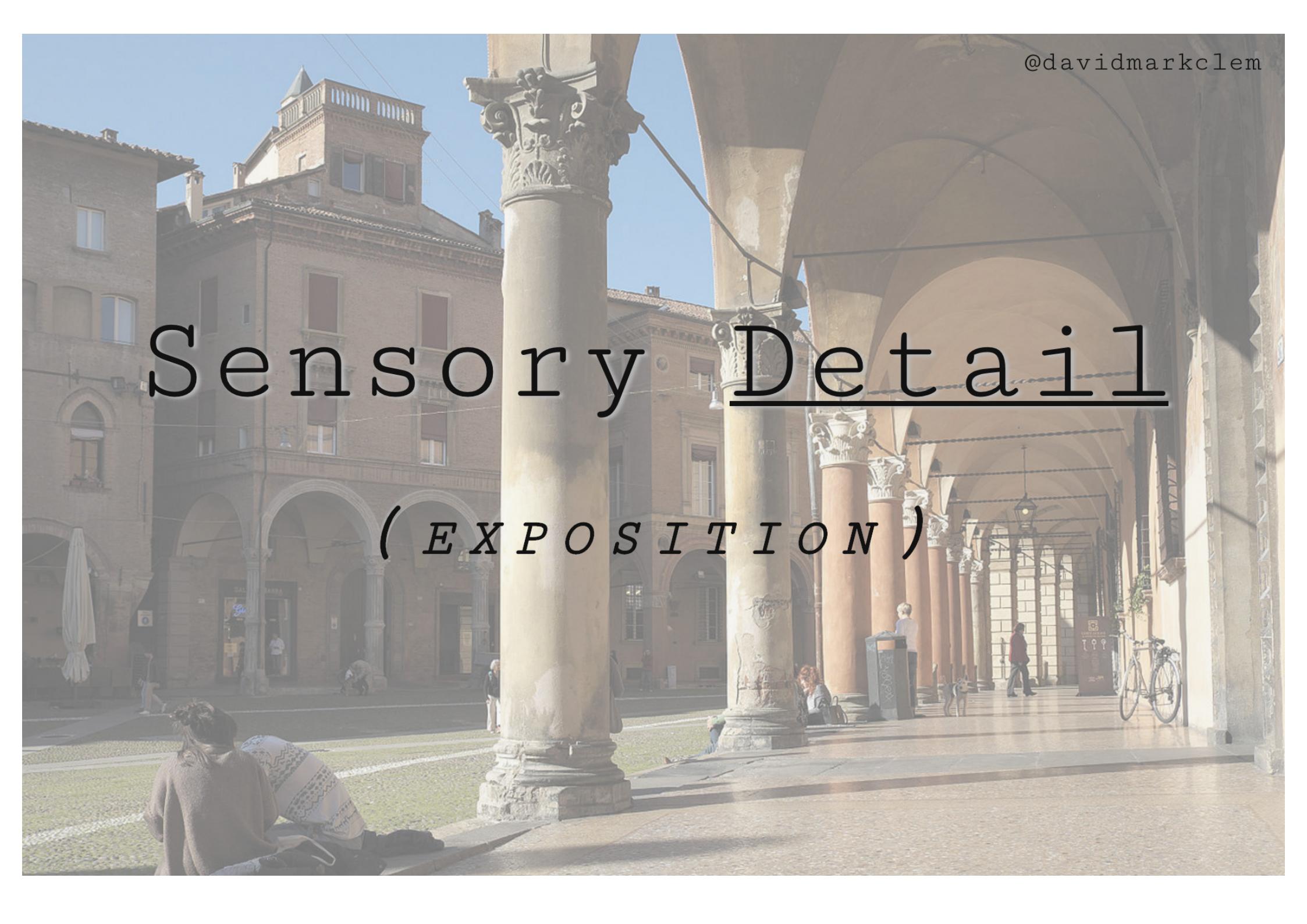
davidmarkclements.github.io/perf-stories

Framing Device

Narrative
Techniques

Flashback: January 2016

The Frame Story



@davidmarkclem

Sensory Detail (EXPOSITION)

telling
a story
within
a story

Narrative Hook

via hypodiegesis



The network card...
became the bottleneck!

[http://www.nearform.com/
nod crunch/client-case-
study-news-uk](http://www.nearform.com/nod crunch/client-case-study-news-uk)

Moral:

Comment Optimized Code

Ticking Clock Scenario

Ticking Clock Scenario

A photograph of a woman in a floral dress and a man in a suit sitting at a table in a restaurant.

@davidmarkclem

NET-A-PORTER

[http://www.nearform.com/
nodocrunch/client-case-
study-net-a-porter](http://www.nearform.com/nodocrunch/client-case-study-net-a-porter)

Our Cambrian Explosion



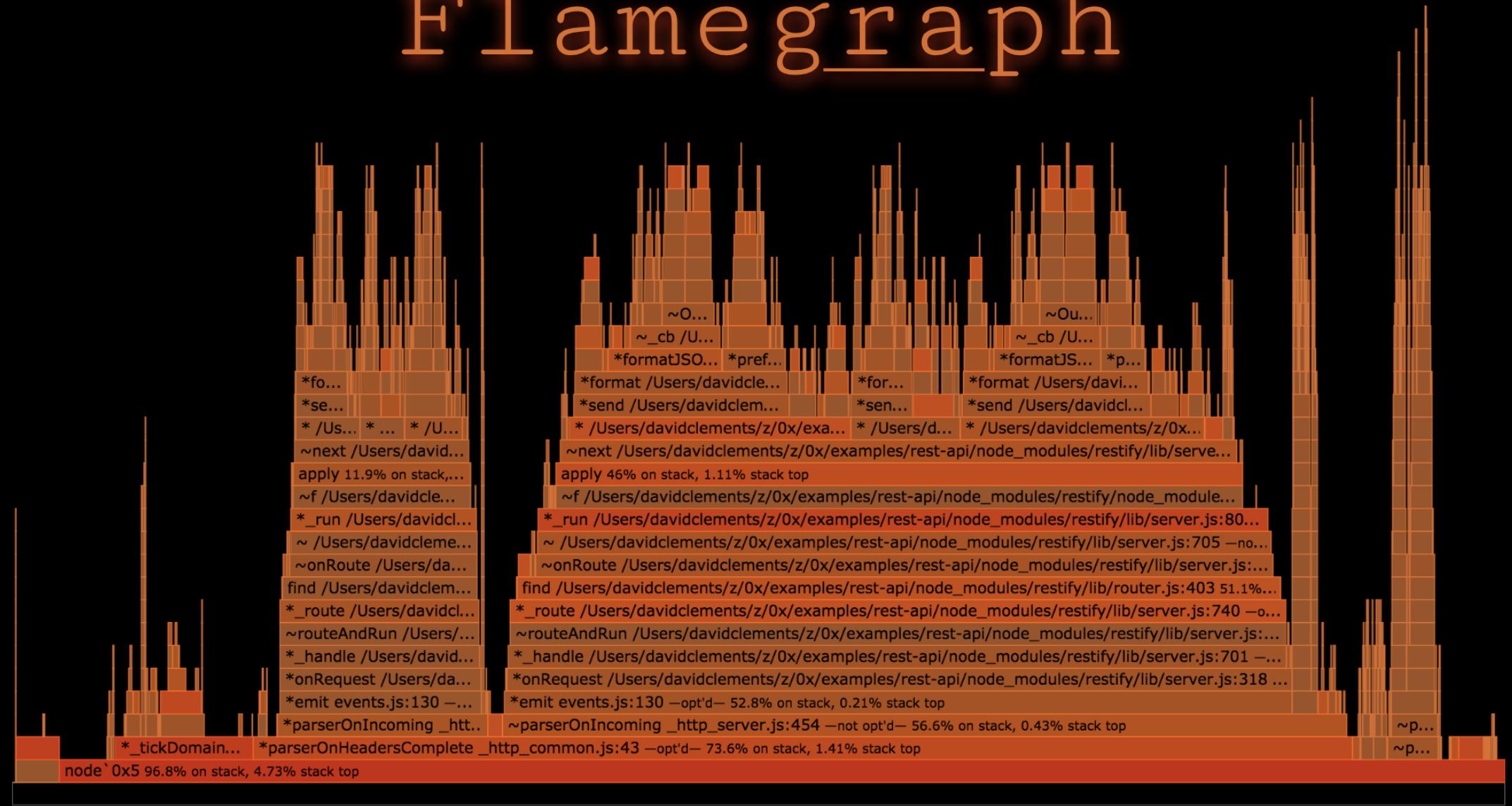
Our Cambrian Explosion



Deus Ex Machina

The Ox Story

Flame graph



perf - sym



Fighting Red Herrings

Autocannon Origins

a b & w r k

Science & Magic

The Birth of Pino

Science & Magic

The Birth of Pino

eval alt #1

@davidmarkclem

Merged

mcollina merged 1 commit into pinojs:master from davidmarkclements:master on 22 Feb

Conversation 10

Commits 1

Files changed 1



davidmarkclements commented on 22 Feb

Member

Delivers us from eval

On average I think we're losing maybe 15-30ms by not using eval, however I think there's more room to optimise here (seeing some deopts in both cases, e.g. Input Smi and Unexpected Object). Also util.format is unoptimizable so we might want to rewrite that part too.

The trade off here is using eval vs limited args and possibly up to 30ms hit (quite a lot - could be optimized more though - also we're still miles ahead). Not using eval may help adoption.

with eval:

```
benchPino*10000: 467.518ms
benchPinoObj*10000: 515.426ms
benchPino*10000: 408.458ms
benchPinoObj*10000: 500.840ms
```

without:

```
benchPino*10000: 461.231ms
benchPinoObj*10000: 525.623ms
benchPino*10000: 449.021ms
```

#1

20-25% speed increase #2

@davidmarkclem

Merged

mcollina merged 1 commit into master from stringify-opt on 7 Mar

Conversation 0

Commits 1

Files changed 2



davidmarkclements commented on 7 Mar

Member

"String Theory"



We can also see the stringify function is not optimized (hence, slow)

By manually creating the initial string, we remove the stringify bottleneck

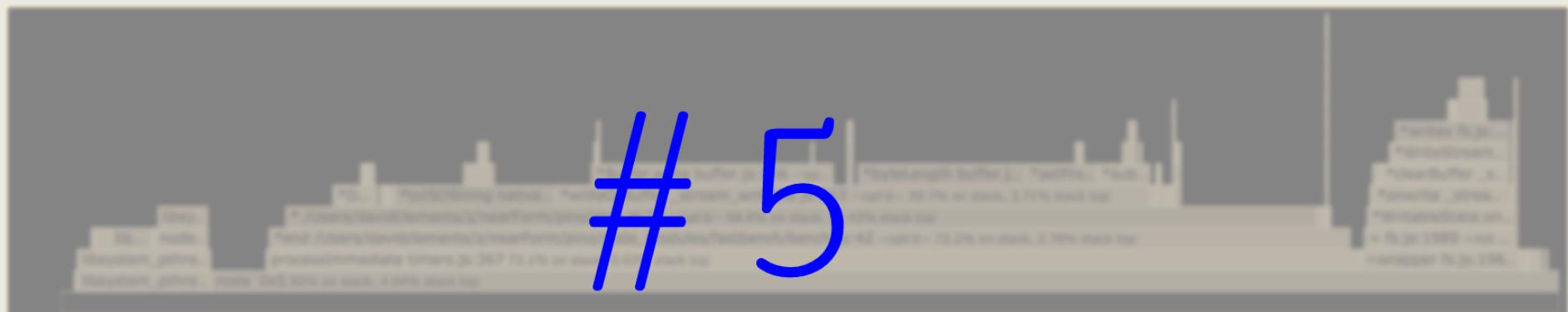
timestamp opt (20-25% speed increase) #5

@davidmarkclem



Killing Time

When we profile an isolated benchmark with `node --prof ./npm.im.js` (again) we can see that `toISOString` is hot (10.8% on stack 8.72% on top of stack)



+10-20% child creation, +7-8% basic/child logging,
JSON output fix, sub-child loggers #21

@davidmarkclem

Merged

davidmarkclem... merged 1 commit into master from child-ad-infinitum-and-perf-refactor on 1 Apr

Conversation 9

Commits 1

Files changed 5



davidmarkclements commented on 1 Apr

Member

fixes #20
closes #19

Waiting for Closure

Summary:

- child loggers of child loggers as many as you want with close to zero overhead
- 10-20% speed improvement in child creation
- 7-8% improvement in basic logging and child logging
- important JSON output fix (#20)
- improves express-pino-logger both requests per second and throughput by 15%

#21

child creation shows speed improvements of

60% perf increase !! #24

@davidmarkclem

Merged

davidmarkclem... merged 17 commits into master from insanity-mode on 6

Conversation 21

#24

Ok so this one breaks all the tests, but it still actually functionally equivalent.

```
module.exports = function flatstr(s) {  
    Number(s)  
    return s  
}
```

We should probably use the `death` module or some equivalent to make sure we handle edge cases, but just wanted to throw this out there for discussion.

Behind the Veil
Of whom we speak not

Behind the Veil
Of whom we speak not

Rigging

Realtime Profiling

Epilogue

A Prophecy

fin

Perf Stories