Atomic Stripe

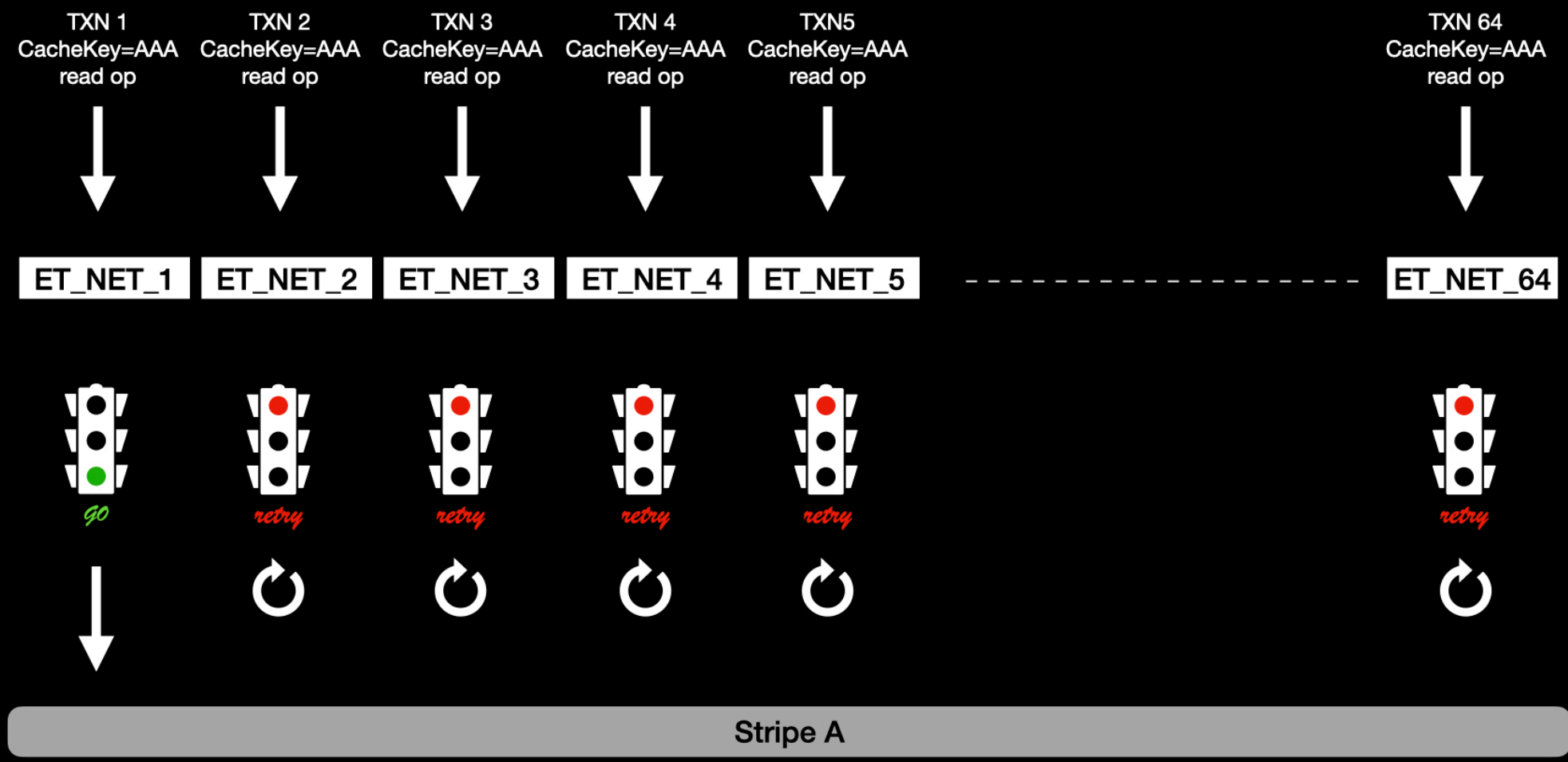
Hazard Pointer and Cache Stripe

2024 ATS Fall Summit

Masaori Koshiba (masaori@apache.org)

Spring 2023 ATS Summit

Try Lock Contention of Stripe Mutex



Mitigations

- increase number of Volumes (volume.config)

Concerns

- Number of CPU/Thread keep increasing
  - e.g. AMD EPYC™ 9754 has 128 cores and 256 threads

Model Specifications

9004 Series Processors | 9004 Series with 3D V-Cache™ Technology | 97x4 Processors | 8004 Series Processors

Name	# of CPU Cores	# of Threads	Max. Boost Clock	Base Clock	L3 Cache	Default TDP
AMD EPYC™ 9754S	128	128	Up to 3.1 GHz	2.25 GHz	256 MB	360W
AMD EPYC™ 9754	128	256	Up to 3.1 GHz	2.25 GHz	256 MB	360W
AMD EPYC™ 9734	112	224	Up to 3 GHz	2.2 GHz	256 MB	340W
AMD EPYC™ 9684X	96	192	Up to 3.7 GHz	2.55 GHz	1152 MB	400W
AMD EPYC™ 9654P	96	192	Up to 3.7 GHz	2.4 GHz	384 MB	360W
AMD EPYC™ 9654	96	192	Up to 3.7 GHz	2.4 GHz	384 MB	360W
AMD EPYC™ 9634	84	168	Up to 3.7 GHz	2.25 GHz	384 MB	290W
AMD EPYC™ 9554P	64	128	Up to 3.75 GHz	3.1 GHz	256 MB	360W
AMD EPYC™ 9554	64	128	Up to 3.75 GHz	3.1 GHz	256 MB	360W
AMD EPYC™ 9534	64	128	Up to 3.7 GHz	2.45 GHz	256 MB	280W
AMD EPYC™ 9474F	48	96	Up to 4.1 GHz	3.6 GHz	256 MB	360W
AMD EPYC™ 9454P	48	96	Up to 3.8 GHz	2.75 GHz	256 MB	290W
AMD EPYC™ 9454	48	96	Up to 3.8 GHz	2.75 GHz	256 MB	290W
AMD EPYC™ 9384X	32	64	Up to 3.9 GHz	3.1 GHz	768 MB	320W
AMD EPYC™ 9374F	32	64	Up to 4.3 GHz	3.85 GHz	256 MB	320W
AMD EPYC™ 9354P	32	64	Up to 3.8 GHz	3.25 GHz	256 MB	280W
AMD EPYC™ 9354	32	64	Up to 3.8 GHz	3.25 GHz	256 MB	280W
AMD EPYC™ 9334	32	64	Up to 3.9 GHz	2.7 GHz	128 MB	210W
AMD EPYC™ 9274F	24	48	Up to 4.3 GHz	4.05 GHz	256 MB	320W
AMD EPYC™ 9254	24	48	Up to 4.15 GHz	2.9 GHz	128 MB	200W
AMD EPYC™ 9224	24	48	Up to 3.7 GHz	2.5 GHz	64 MB	200W
AMD EPYC™ 9184X	16	32	Up to 4.2 GHz	3.55 GHz	768 MB	320W
AMD EPYC™ 9174F	16	32	Up to 4.4 GHz	4.1 GHz	256 MB	320W
AMD EPYC™ 9124	16	32	Up to 3.7 GHz	3 GHz	64 MB	200W

Hazard Pointer

```cpp
namespace std {
  // hazard_pointer_obj_base
  template<class T, class D = default_delete<T>>
  class hazard_pointer_obj_base;

  // hazard_pointer
  class hazard_pointer;

  // make_hazard_pointer
  hazard_pointer make_hazard_pointer();
  void swap(hazard_pointer&, hazard_pointer&) noexcept;
}
```

Document Number: P2530R3
Date: 2023-03-02
Project WG21, LWG
Reply to: Maged M. Michael
maged.michael@gmail.com

Hazard Pointers for C++26

Authors:
Maged M. Michael, Michael Wong, Paul McKenney, Andrew Hunter, Daisy S. Hollman, JF Bastien, Hans Boehm, David Goldblatt, Frank Birbacher, Mathias Stearn

Email:
maged.michael@gmail.com, michael@codeplay.com, paulmck@kernel.org, andrewhhunter@gmail.com, dhollman@google.com, cxx@jfbastien.com, hboehm@google.com, davidtgoldblatt@gmail.com, frank.birbacher@gmail.com, redbeard0531+isocpp@gmail.com

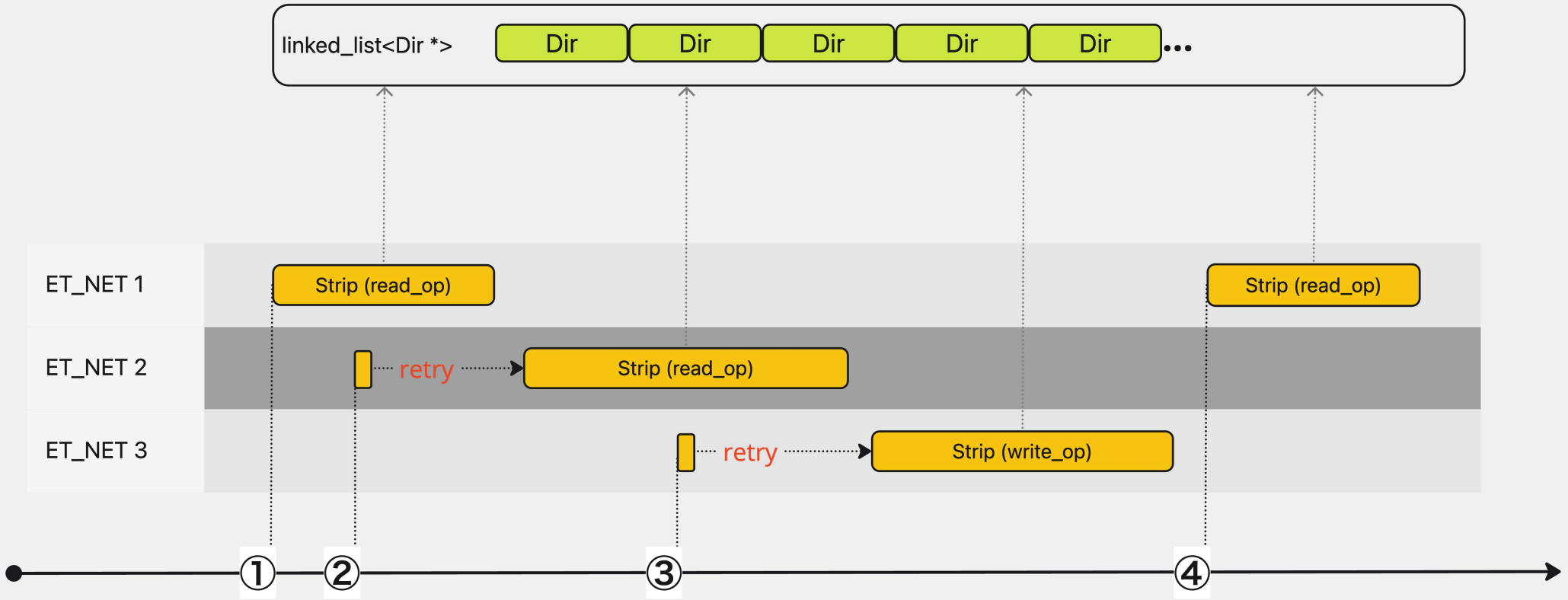
Contents

Atomic Stripe

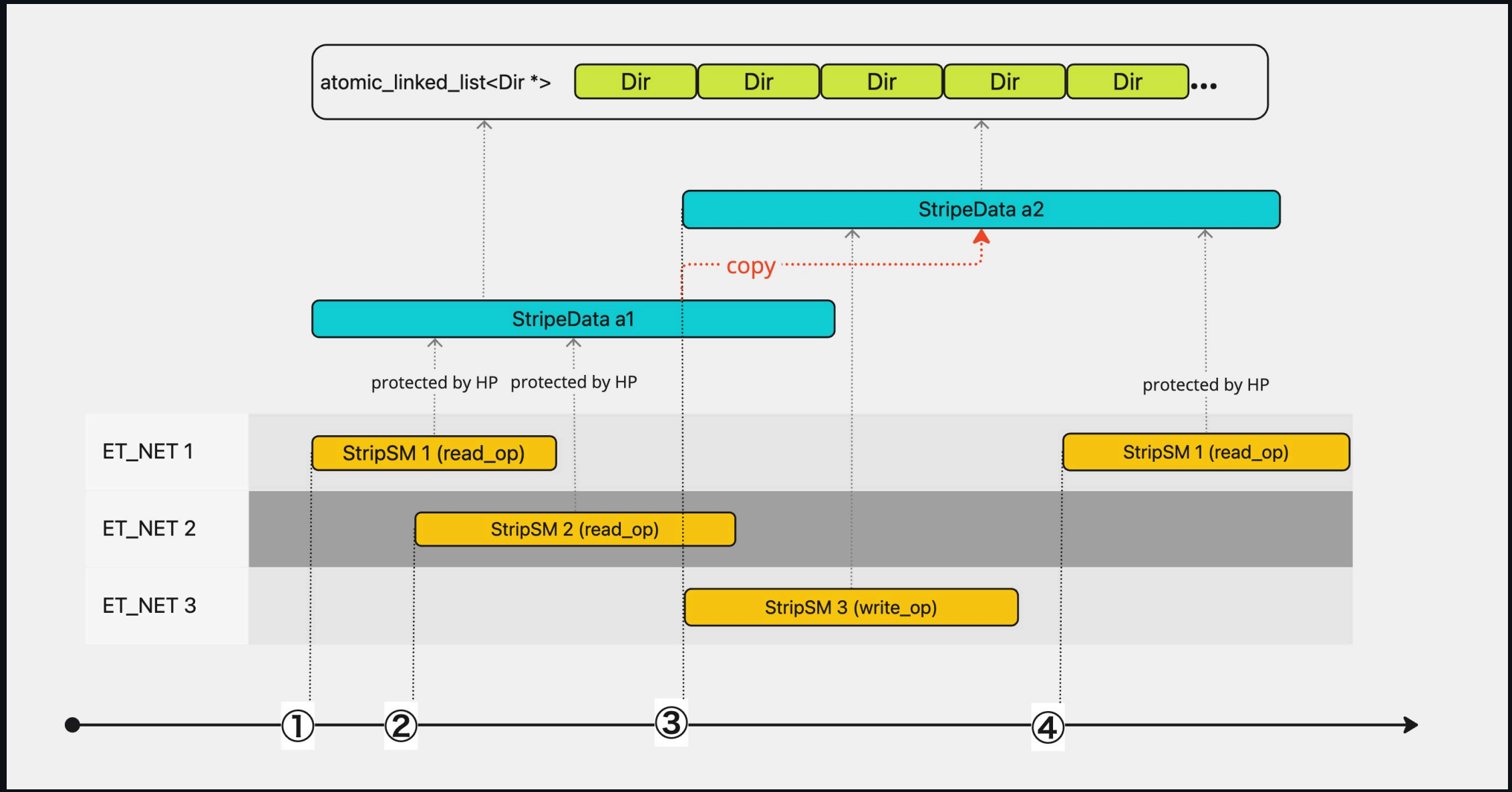
- Break `Stripe` down into `StripeSM` and `StripeData`
- Make `StripeData` shared data, but guard by Hazard Pointer

```
#include <hazard_pointer>

class StripeData : public std::hazard_pointer_obj_base<StripeData>;
std::array<std::atomic<StripeData *>, N> stripe_data_list;

class StripeSM : public Continuation;
```

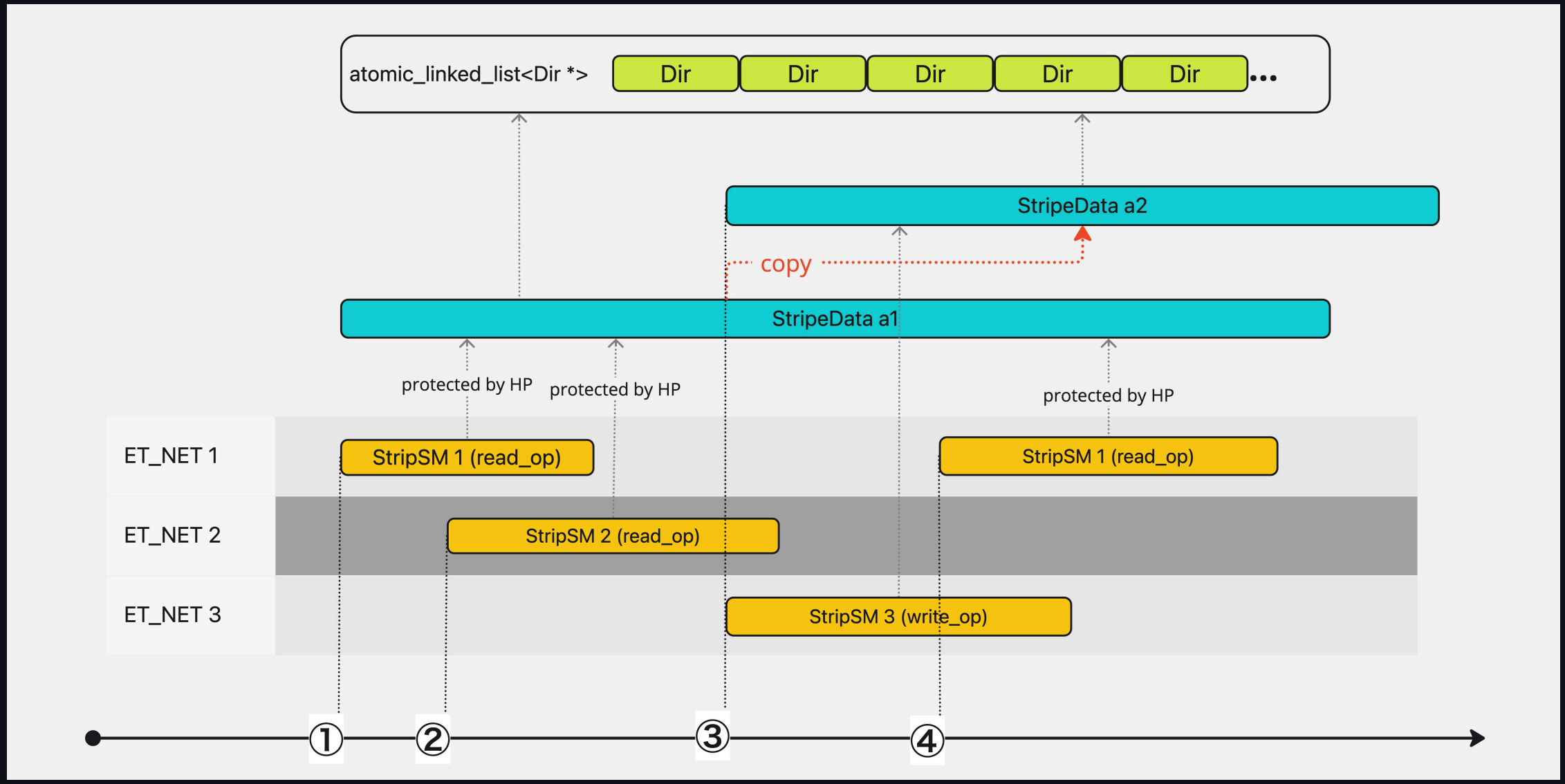
MUTEX_TRY_LOCK (10.0.x) - case 1



Atomic Stripe (proposal) - case 1



Atomic Stripe (proposal) - case 2



PoC & Benchmark

✅ Naive implementation of Hazard Pointer using C++26 APIs

✅ Break Stripe down into StripeSM and StripeData

✅ Protect StripeData by Hazard Pointer

❌ No change for other shared data, such as Dir and RAMCache (not protected)

Hardware

- Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz 64 core

ATS Configs

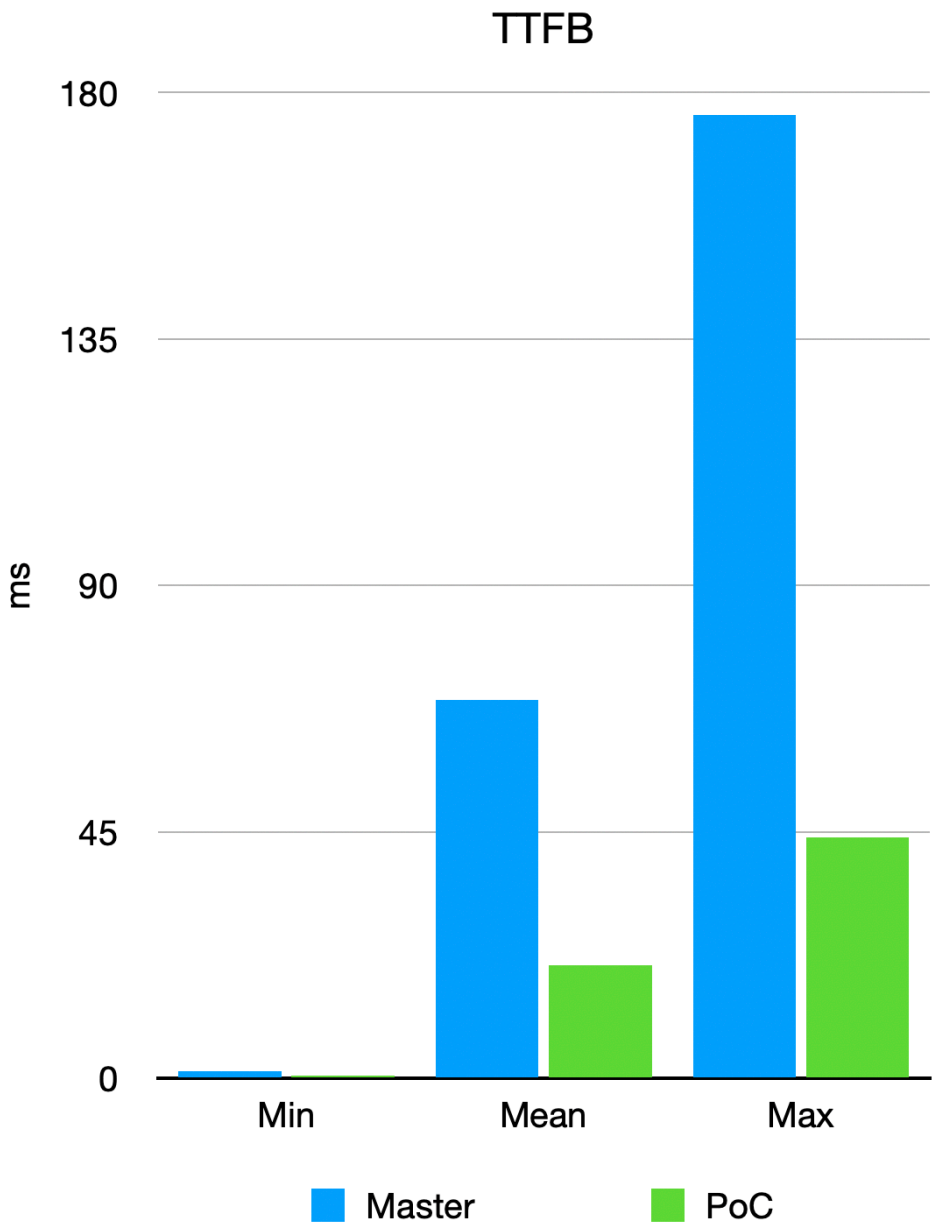
```
exec_thread:
  affinity: 4
  limit: 64
  listen: 1
```

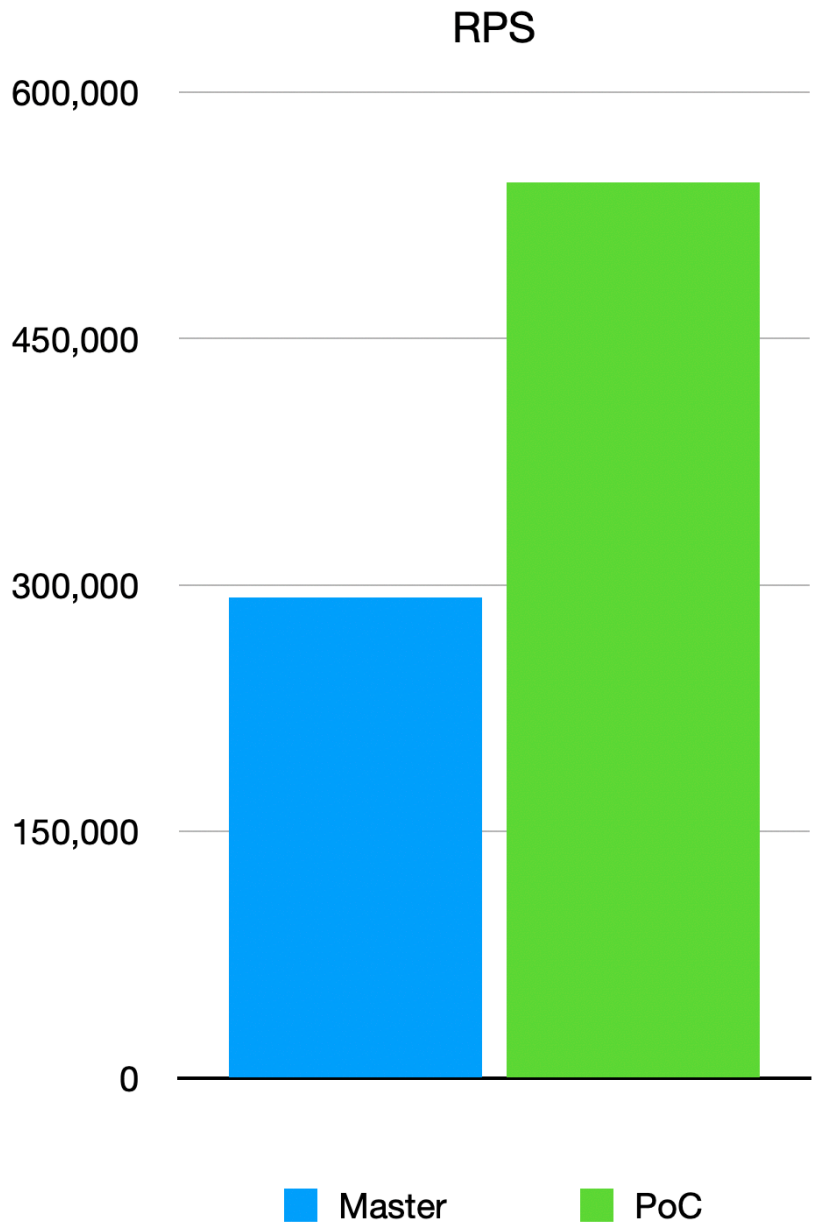
master (`eabe1e97`, Aug 29)

```
$ h2load -t 64 -c 640 -n 6400000 --h1 http://...
finished in 21.89s, 292320.44 req/s, 387.37MB/s
requests: 6400000 total, 6400000 started, 6400000 done, 6400000 succeeded, 0 failed, 0 errored, 0 timeout
status codes: 6400000 2xx, 0 3xx, 0 4xx, 0 5xx
traffic: 8.28GB (8892859878) total, 1.80GB (1936059878) headers (space savings 0.00%), 6.10GB (6553600000) data
                     min         max         mean         sd        +/- sd
time for request:       41us    161.97ms      1.75ms      4.45ms    94.99%
time for connect:       41us     29.84ms     11.73ms     10.33ms    41.56%
time to 1st byte:     1.21ms    175.80ms     69.12ms     24.56ms    71.09%
req/s           :     457.09      969.17      658.81      123.53    67.03%
```

PoC

```
$ h2load -t 64 -c 640 -n 6400000 --h1 http://...
finished in 11.75s, 544864.57 req/s, 722.80MB/s
requests: 6400000 total, 6400000 started, 6400000 done, 6400000 succeeded, 0 failed, 0 errored, 0 timeout
status codes: 6400000 2xx, 0 3xx, 0 4xx, 0 5xx
traffic: 8.29GB (8902400000) total, 1.81GB (1945600000) headers (space savings 0.00%), 6.10GB (6553600000) data
                     min         max         mean         sd        +/- sd
time for request:       49us      5.53ms       880us       354us    71.79%
time for connect:       51us     40.47ms     18.04ms     14.45ms    42.19%
time to 1st byte:      543us     43.93ms     20.70ms     14.45ms    42.19%
req/s           :     852.34     5238.60     1240.47      425.30    89.84%
```


TTFB
180
135
90
45
0
ms
Min
Mean
Max
Master
PoC


RPS
600,000
450,000
300,000
150,000
0
Master
PoC

Current State

- A bunch of Stripe refactorings are already done by @JoshiaWI
- Using Hazard Pointer might resolve issue
  - Trade-off: Copy StripeVM and StripeData

TODOs

- Make StripSM and StripData small and copyable
- Protect shared data (like Dir and RAMCache)
- Reliable implementation of Hazard Pointer

StripeData and StripeSM

```
// Holding Dir and other
class StripeData : public std::hazard_pointer_obj_base<StripeData>
{
public:
  Dir                 *dir{};    ///< TODO: lock free
  StripteHeaderFooter *header{};
  StripteHeaderFooter *footer{};
  ...
}

std::array<std::atomic<StripeData *>, N> stripe_data_list;

// Having event handler and run as Continuation
class StripeSM : public Continuation
{
private:
  std::atomic<StripeData *> &_stripe;

public:
  // All access to StripeData through this read and write op
  template <typename U, typename Func> U stripe_read_op(Func) const;
  template <typename U, typename Func> U stripe_write_op(Func);
...
}
```

Reading and Writing

```
template <typename U, typename Func>
U
StripeSM::stripe_read_op(Func read_op) const
{
  std::hazard_pointer h      = std::make_hazard_pointer();
  const StripeData   *stripe = h.protect(_stripe);

  return read_op(stripe);
}

template <typename U, typename Func>
U
StripeSM::stripe_write_op(Func write_op)
{
  StripeData *current_stripe = _stripe.load(std::memory_order_relaxed);
  StripeData *new_stripe     = new Stripe(*current_stripe);              ///< COPY!!

  U res = write_op(new_stripe);

  StripeData *old_stripe = _stripe.exchange(new_stripe);
  old_stripe->retire();

  return res;
}
```

Example of read_op call

```
inline bool
StripeSM::dir_valid(const Dir *dir) const
{
  return this->stripe_read_op<bool>([&](const Stripe *stripe) {
    return stripe->dir_valid(dir);
  });
}
```