SAFE MEMORY RECLAMATION

EPOCH RECLAMATION

appnexus

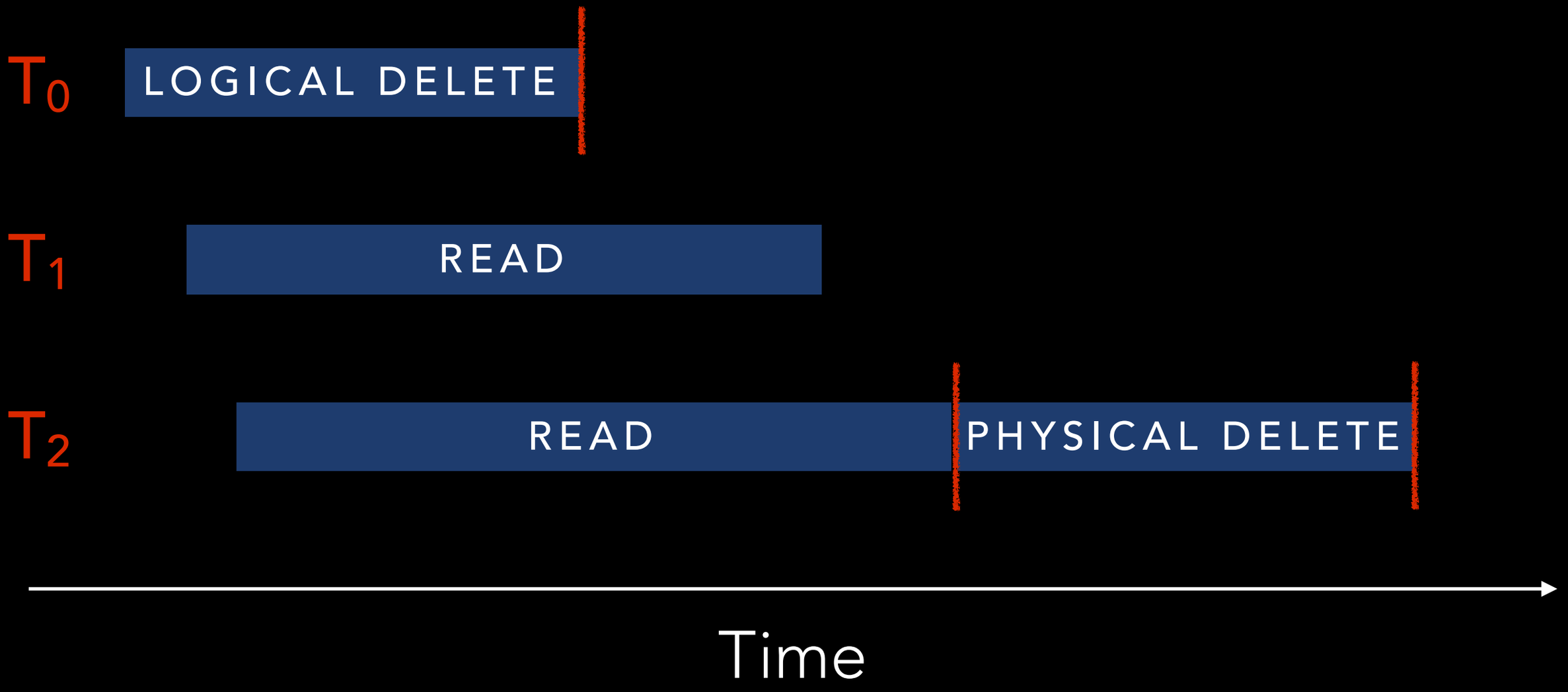
BACKGROUND

- As reachability and liveness are delineated, more complex memory management mechanisms are necessary for the safe destruction of objects.

- Object reference counting is the most common solution to this problem where blocking synchronization is permitted.

BACKGROUND

REFERENCE COUNTING



BACKGROUND

- Reference counting has performance limitations that can affect both scale and fast path latency.
- Object reference counting is generally insufficient for concurrent synchronization with linearization guarantees.

INTRODUCTION

- Safe memory reclamation mechanisms have been developed for performance and for correctness.

Pass-The-Buck

Hazard Pointers

Read-Copy-Update

Proxy Collection

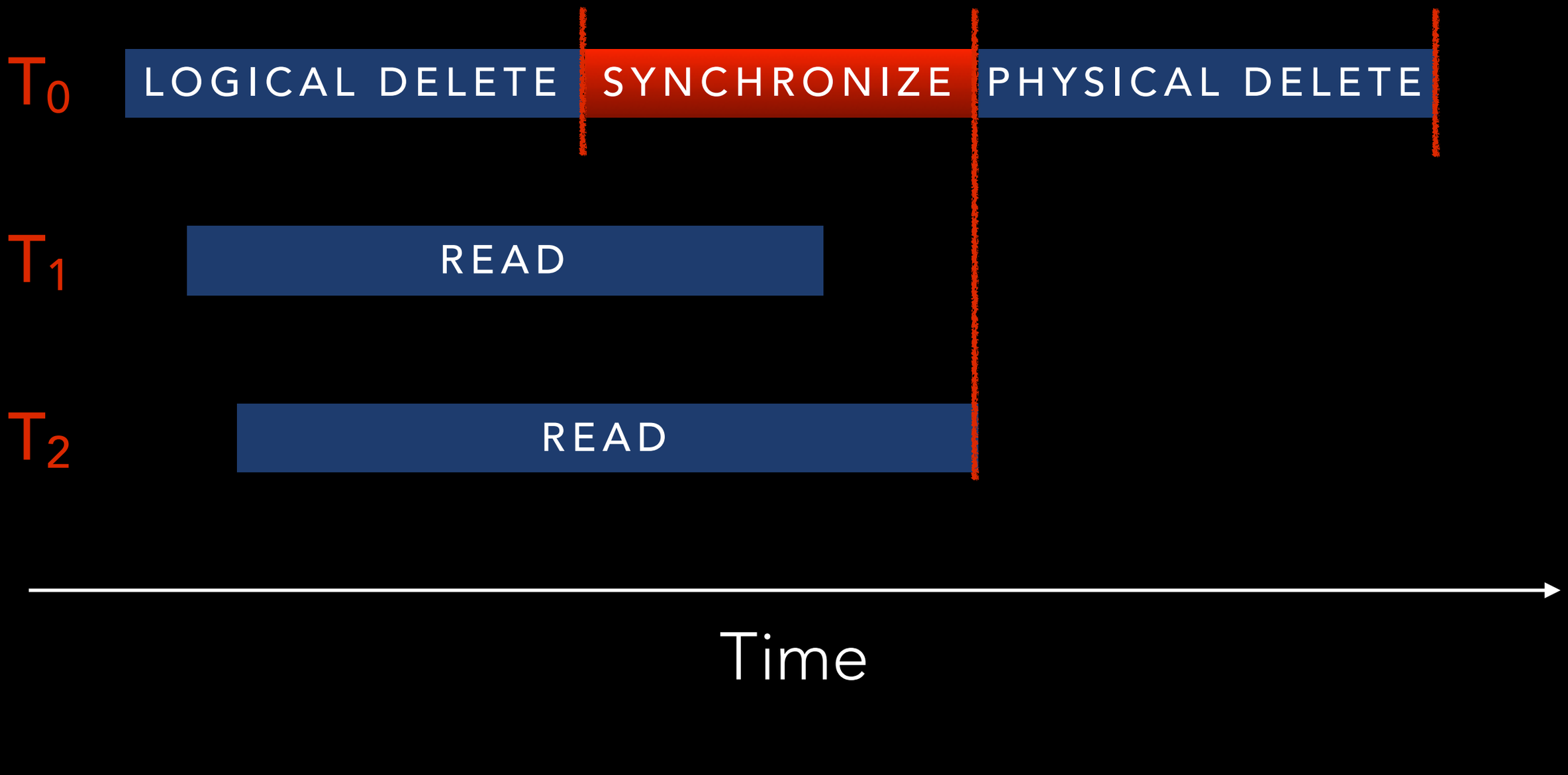
Time-Based Deferral

appnexus

INTRODUCTION

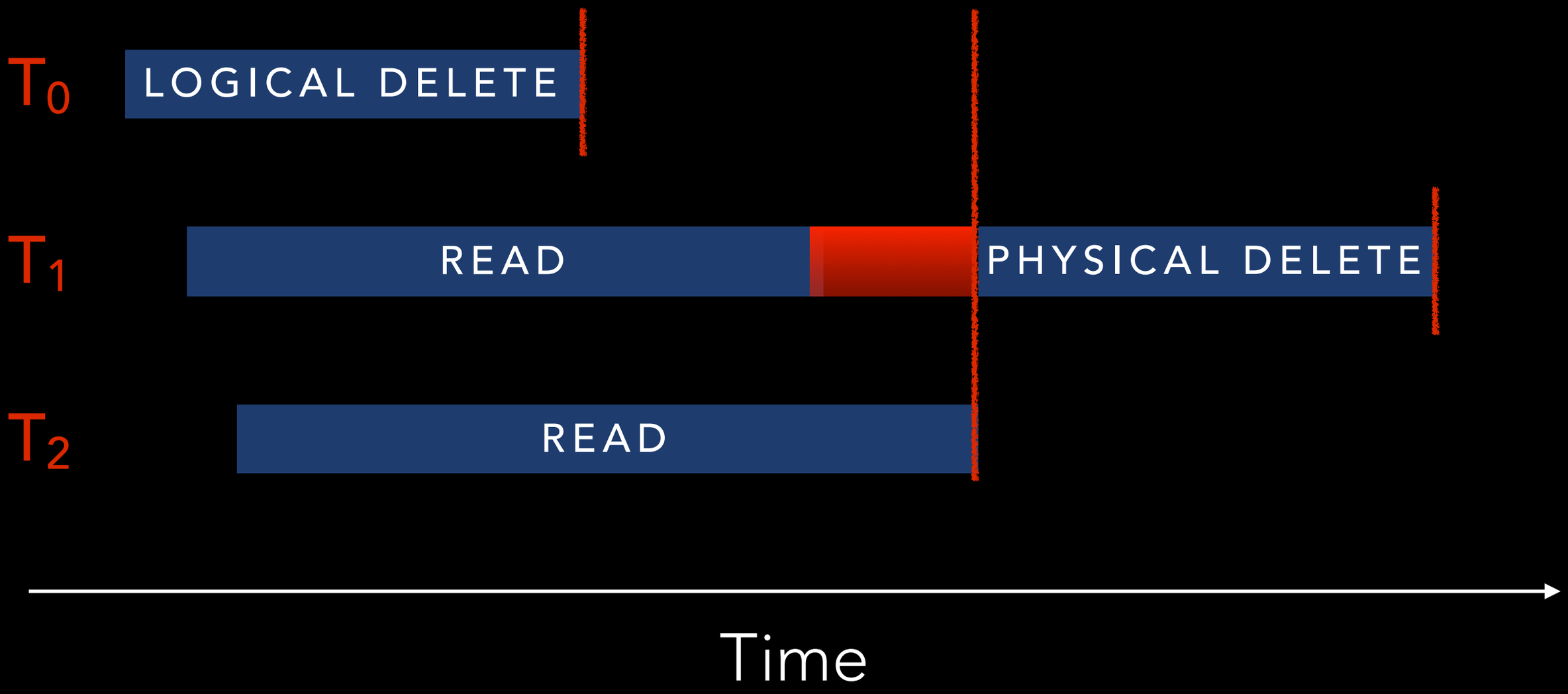
- Passive interfaces such as the one provided by RCU and Concurrency Kit allow for versatility across all types of workloads and algorithms.
- Central to passive mechanisms is **grace period detection.**

GRACE PERIOD DETECTION



appnexus

GRACE PERIOD DETECTION



GRACE PERIOD DETECTION

```
void
reader(void)
{
     object_t *object;
     struct node *n;

     for (;;) {
          smr_begin();
          object = lookup(something);
          CK_LIST_FOREACH(n, my_list, linkage)
               function(n, *object);
          smr_end();
     }
}
```

```
void
single_writer(void)
{
     object_t *object;
     struct node *n;

     for (;;) {
          object = remove(something);
          n = CK_LIST_HEAD(my_list);
          CK_LIST_REMOVE(n, linkage);
          synchronize();
          free(object);
          free(n);
     }
}
```

- After `synchronize`, it is guaranteed no thread executing `reader` could have a reference to any of the logically deleted objects.

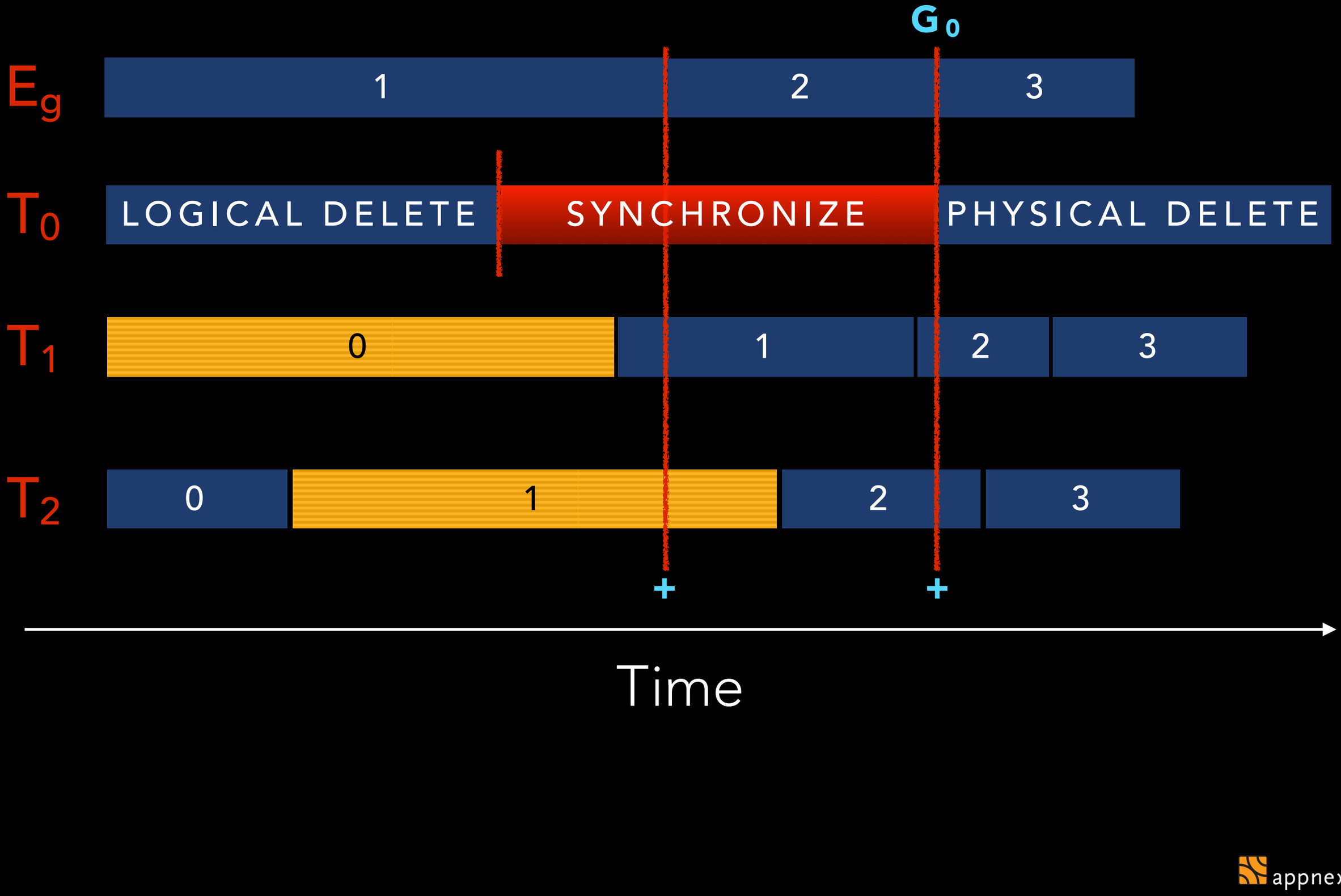
EPOCH RECLAMATION

- Upon entry into a read-side protected section, readers set an active bit and take a snapshot of a global epoch counter. **A memory barrier is required to avoid store to load re-ordering.**

- Synchronize operations increment the epoch counter only if all active threads have a snapshot of the latest value of the global counter.

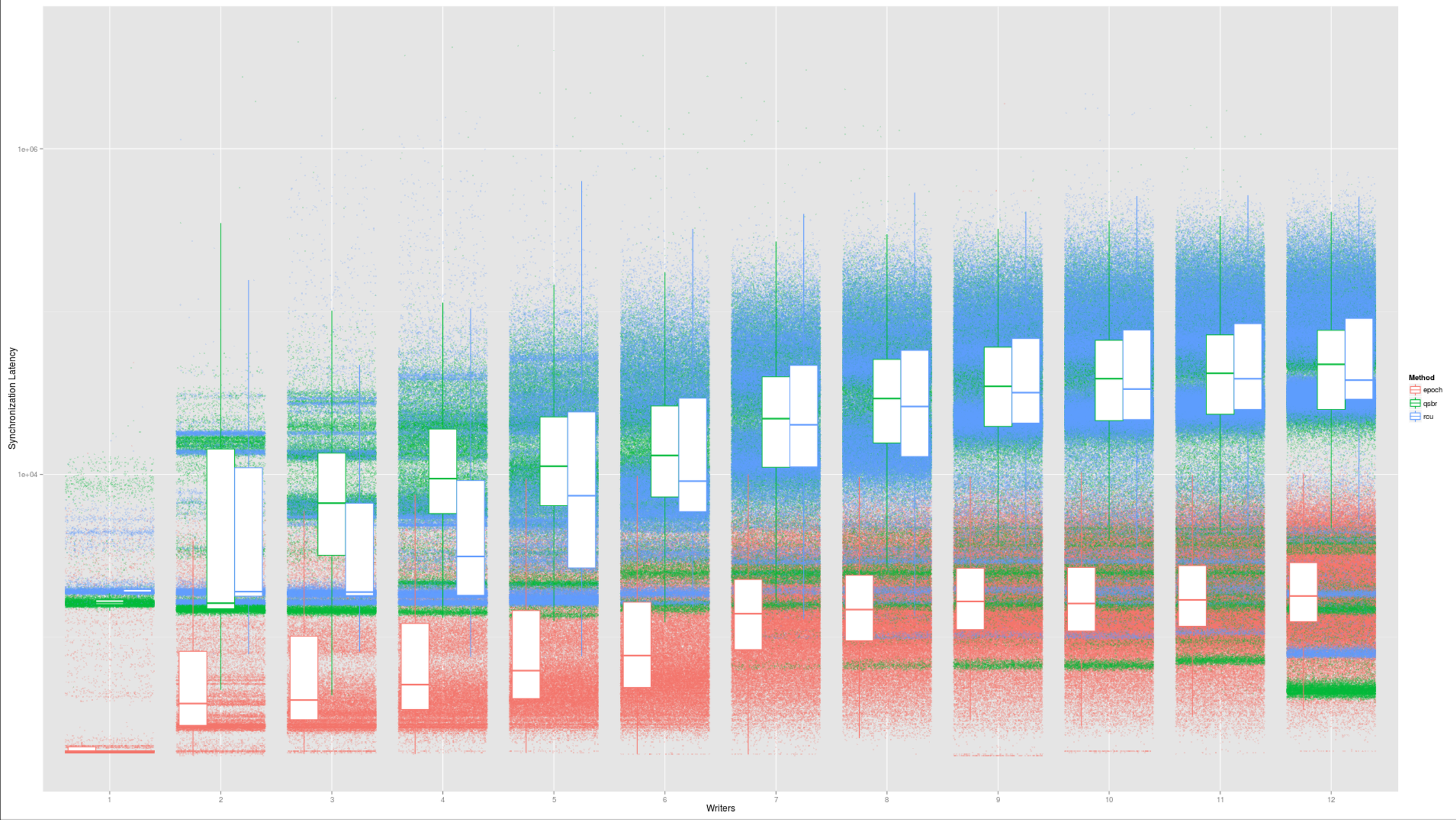
- If the epoch counter is successfully incremented twice from the time synchronize was called, then no references could exist to objects logically deleted before the synchronize call.

EPOCH RECLAMATION



appnexus

EPOCH RECLAMATION



SYNCHRONIZE COST (WRITE-MOSTLY WORKLOAD/ RDTSCP)

appnexus

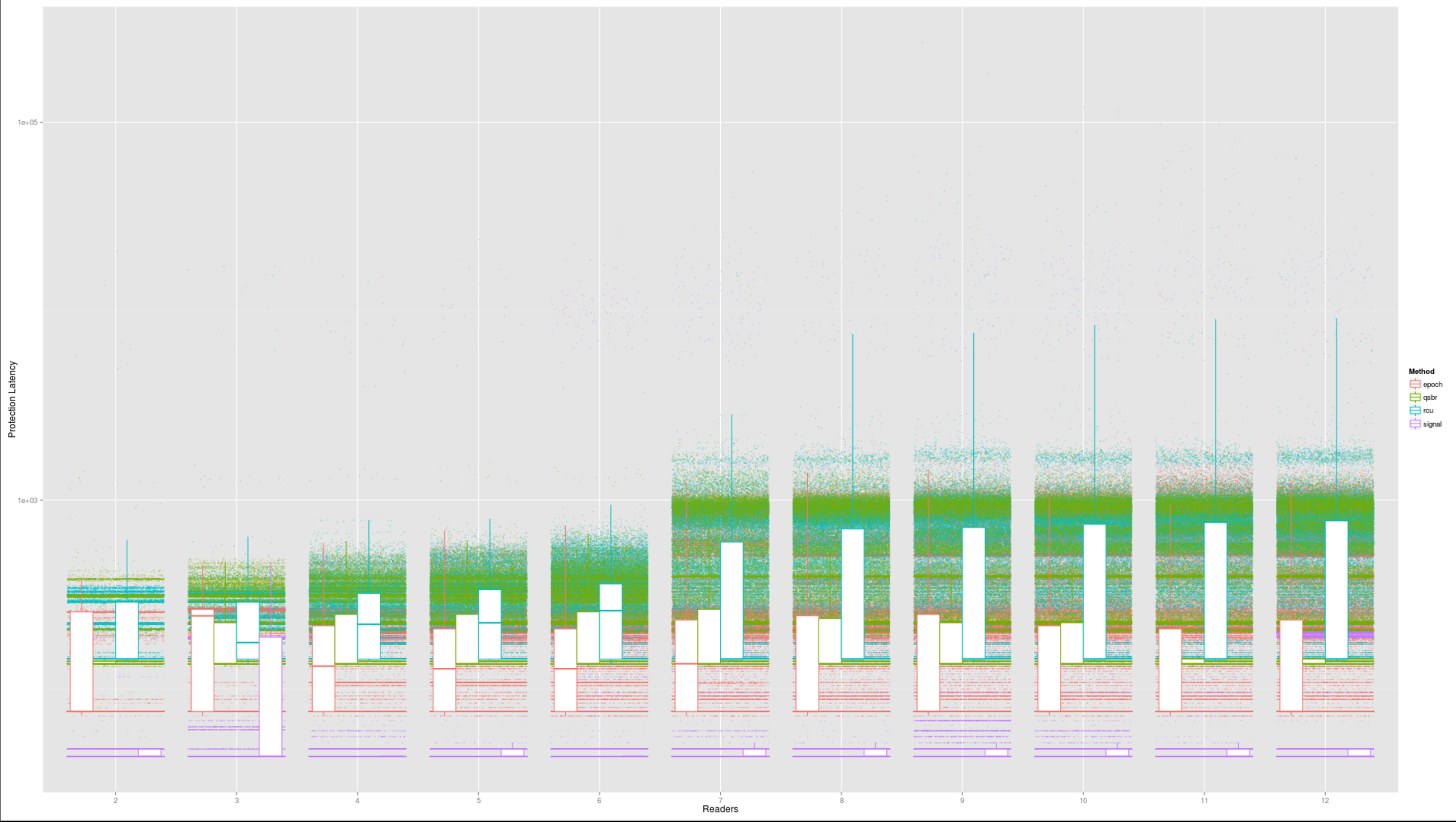
EPOCH RECLAMATION

	Min	1st Qu.	Median	Mean	3rd Qu.	Max
Epoch	188	772	1408	1781	2348	1888000
RCU	720	9260	26610	44080	60670	2181000
QSBR	400	10130	25420	38250	53590	4526000

SYNCHRONIZE COST (WRITE-MOSTLY WORKLOAD/ RDTSCP)

appnexus

EPOCH RECLAMATION



ENTER COST (READ-ONLY WORKLOAD / RTSCP)

appnexus

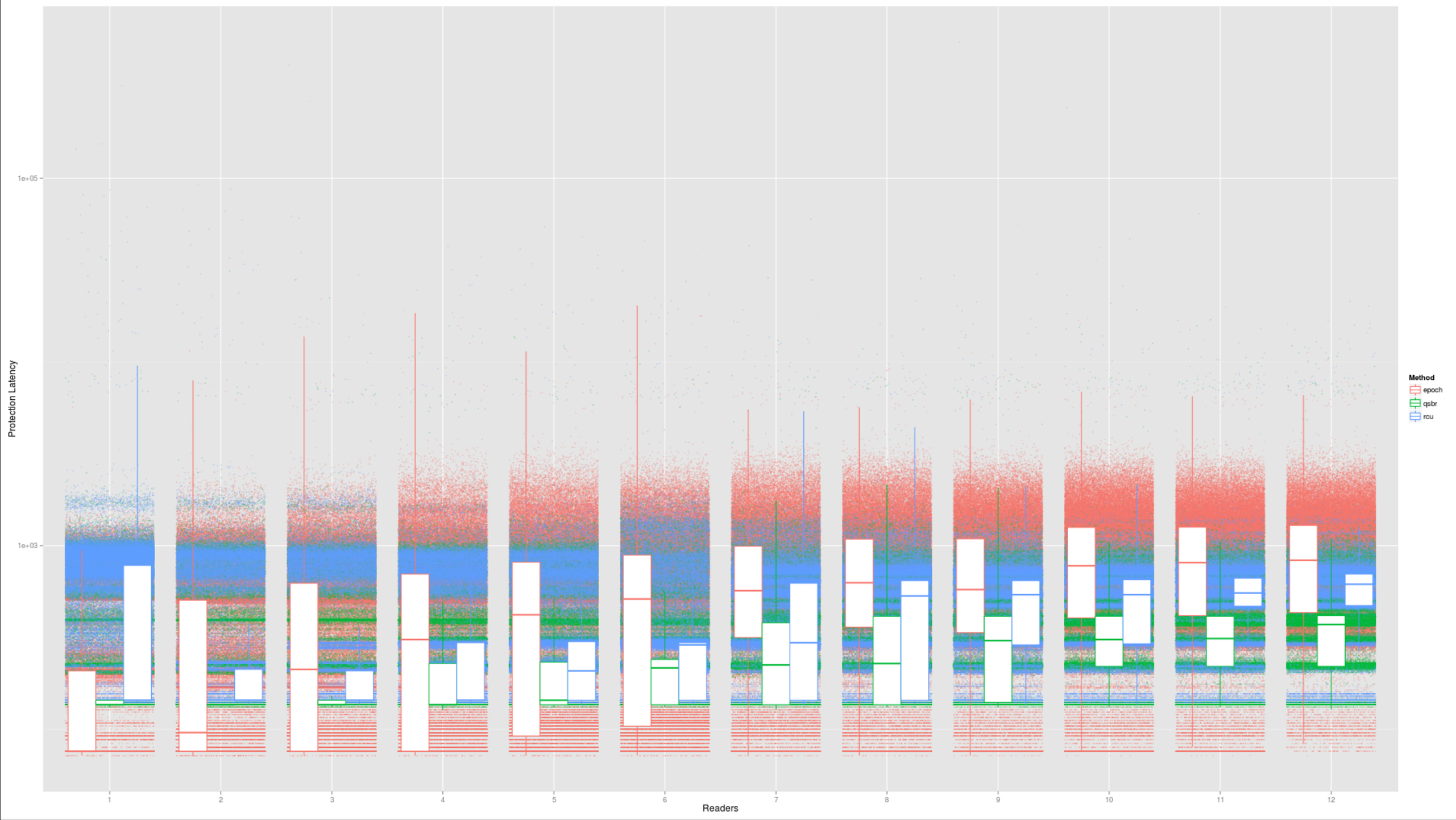
EPOCH RECLAMATION

	Min	1st Qu.	Median	Mean	3rd Qu.	Max
Epoch	72	76	76	204	236	53350
RCU	136	144	144	375	572	264600
QSBR	128	136	136	248	224	54330
Signal	44	44	44	62.35	48	63930

ENTER COST (READ-ONLY WORKLOAD / RTSCP)

appnexus

EPOCH RECLAMATION



ENTER COST (WRITE-MOSTLY WORKLOAD / RDTSCP)

appnexus

EPOCH RECLAMATION

	Min	1st Qu.	Median	Mean	3rd Qu.	Max
Epoch	72	100	484	619	940	551300
RCU	136	144	296	414	636	412800
QSBR	128	136	220	287	380	43050

ENTER COST (WRITE-MOSTLY WORKLOAD / RDTSCP)

appnexus

THE END

HTTP://CONCURRENCYKIT.ORG

appnexus