SLASH 24

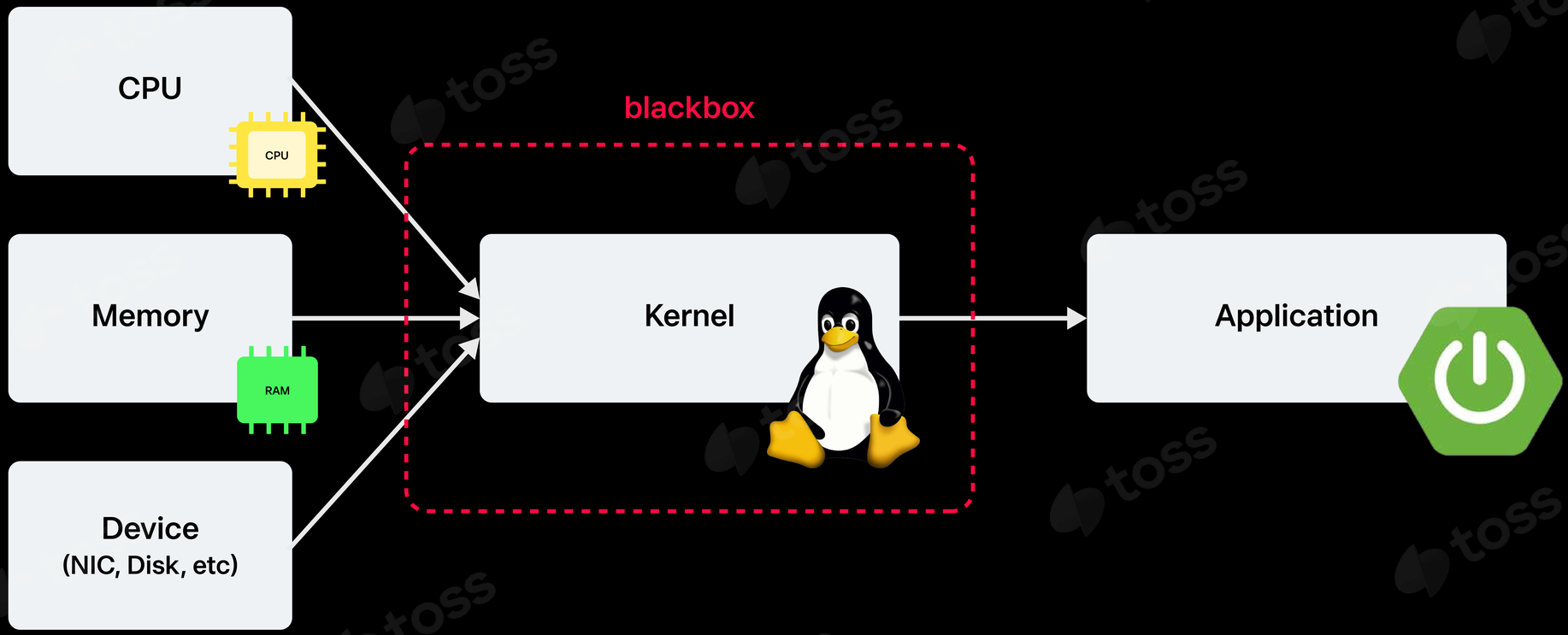
미처 알지 못했던 Kernel까지 Observability 향상시키기

이항령 Head of Server
이재성 DevOps Team Leader

toss

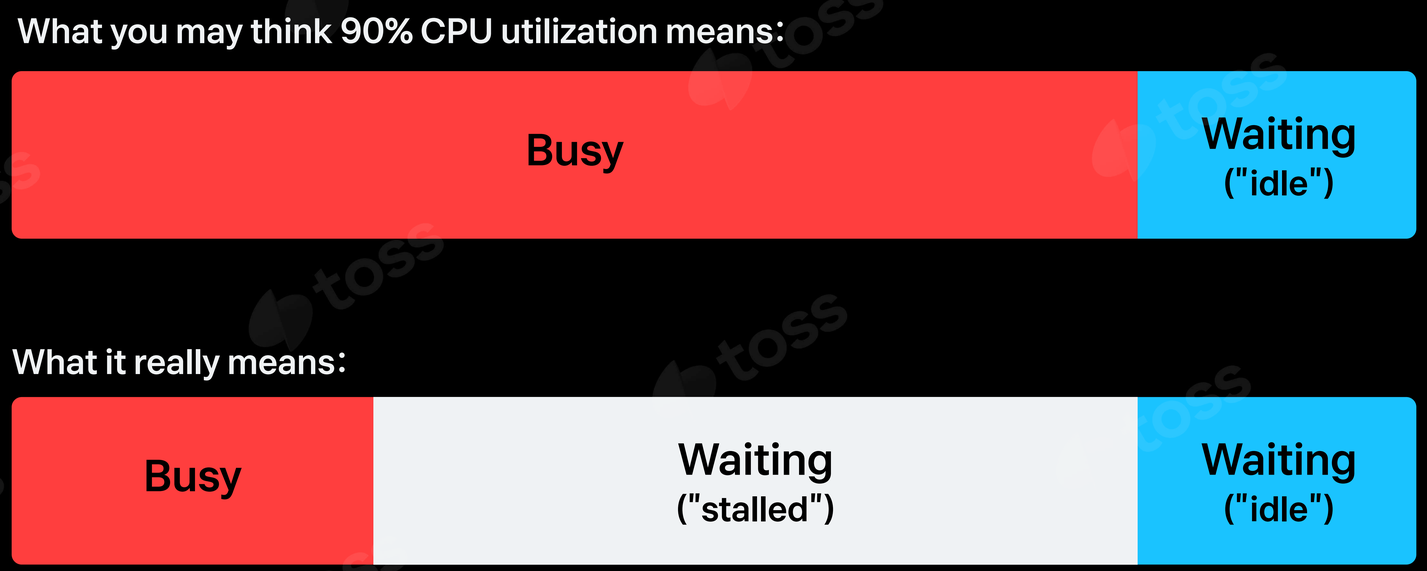
목차

kernel


CPU
CPU
Memory
RAM
Device
(NIC, Disk, etc)
blackbox
Kernel
Application

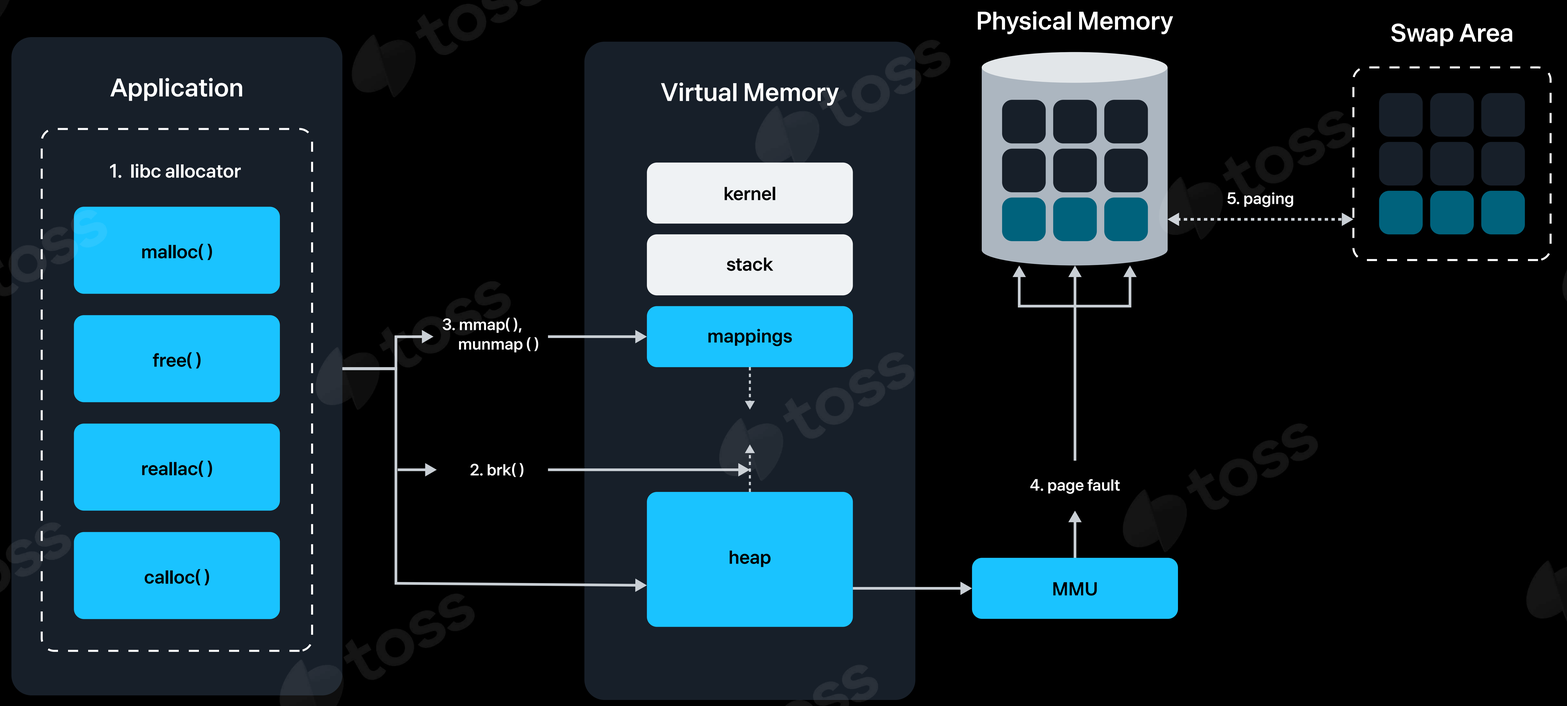
cpu 사용률이란?

cpu 사용률에는 memory io에 의한 기다리는 시간들이 존재함



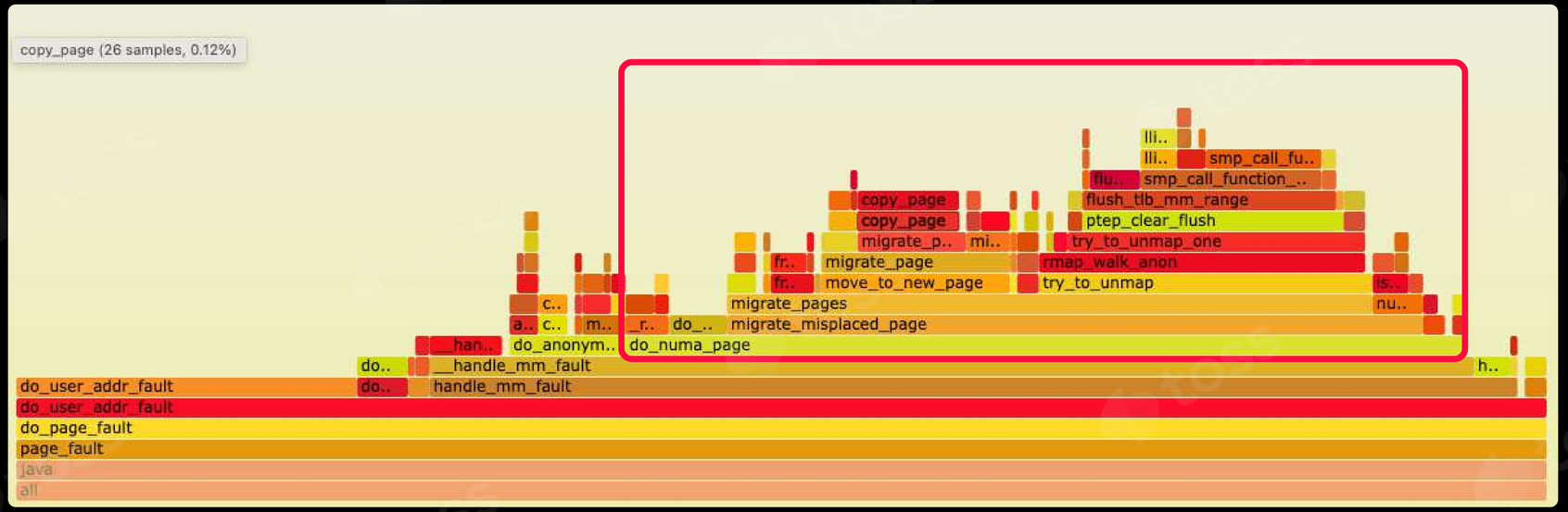
memory io를 추적해보려면?

page fault를 tracing



page fault flamegraph

cpu 사용률 중에 page fault 부분

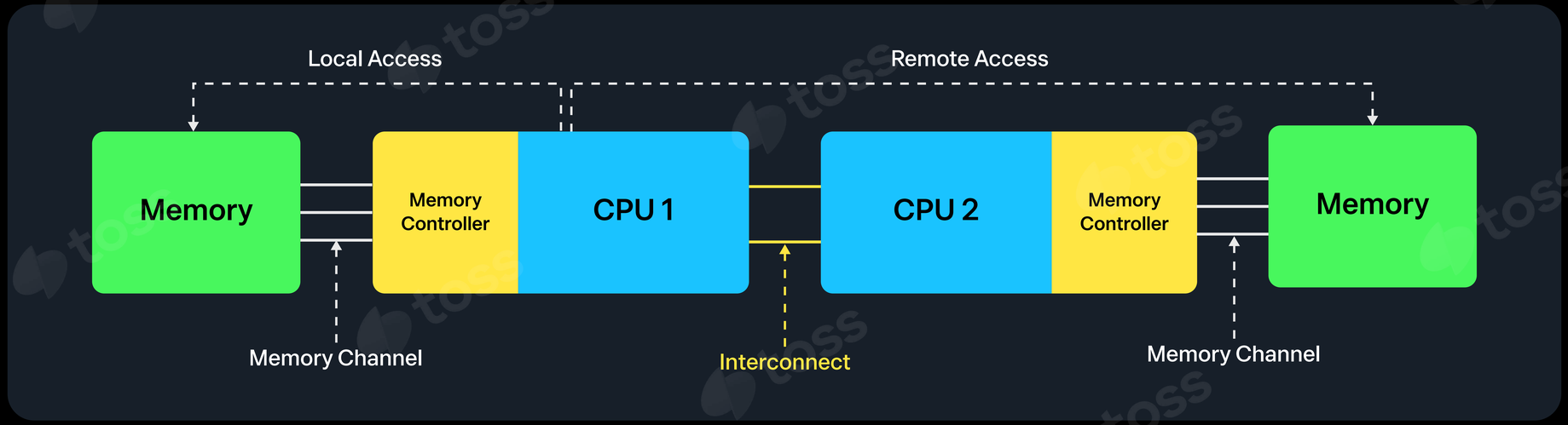


numa 란?

cpu/memory 칩 성능은 늘었지만, 채널 성능은 늘지 않았음

multi socket cpu

remote access 에서 홉이 추가되어 사이클이 더 필요함



numa가 얼마나 영향을 주고 있는지?

numa migration 으로 1초에
10~50ms 시간을 cpu 스케쥴러가 소모

TIM	NUMA_migrations	NUMA_migratims_ms
	706	6
00:08:19	1196	14
00:08:20	3993	37
03:08:21	787	a
03:08:22	422	3
03:08:23	14	0
03:08:20	231	2
03:08:25	13373	52
03:08:26	5475	22
03:08:27	1420	6
03:08:28	104	0
03:08:29	98	0
03:08:30	101	3

memory access

local dram(6243회), remote dram(4415회)

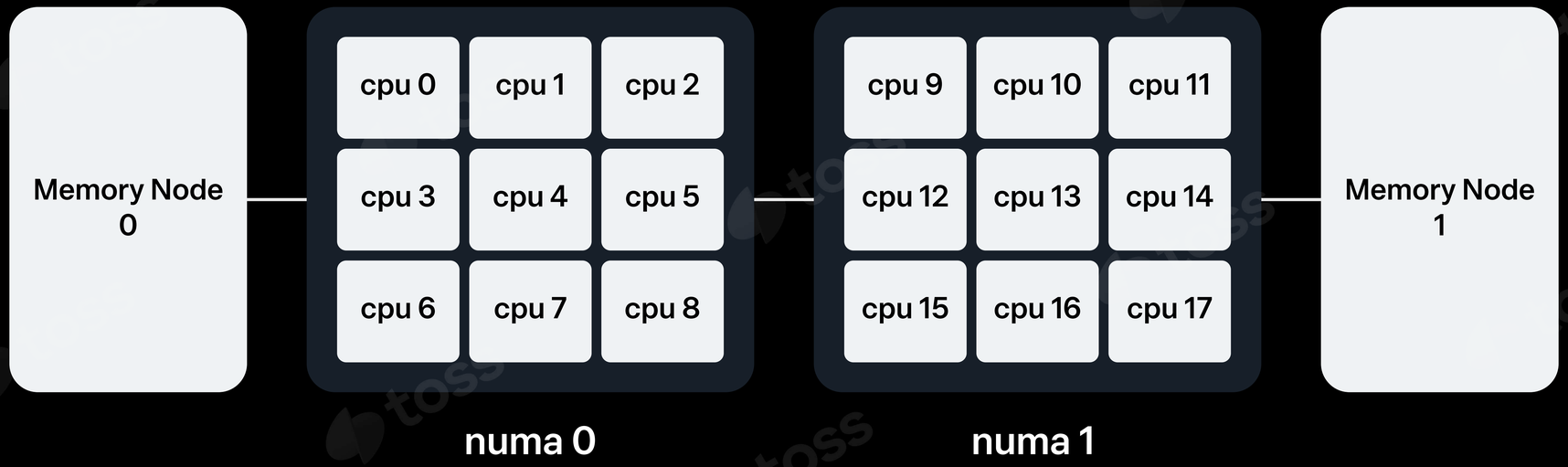
→ 40%가량 remote dram access

Trace Event Information	
Total records	262091
Locked lood/Store Operations	9234
Lood Operations	1011233
Loads - uncacheable	0
Loads - IO	0
Loads - Miss	318
Loads - no mapping	577
Load Fill Buffer Hit	33795
Load L1D hit	43973
Load L2D hit	4848
Load LLC hit	14064
Load Local HITM	2742
Load Remote HITM	1191
Load Remote HIT	
Load Local DRAM	6243
Load Remote DRAM	4415
Load MESI State Exclusive	4415
Load MESI State Shared	6243
Load LLC Misses	111149
Load access blocked by data	0
Load access blocked by address	0
LLC Misses to local DRAM	52.7%
LLC Misses to Remote D1M1M	37.3%
LLC Misses to Remote cache (MIT) :	0.0%

container에서 하드웨어 할당

cpuset_cpu

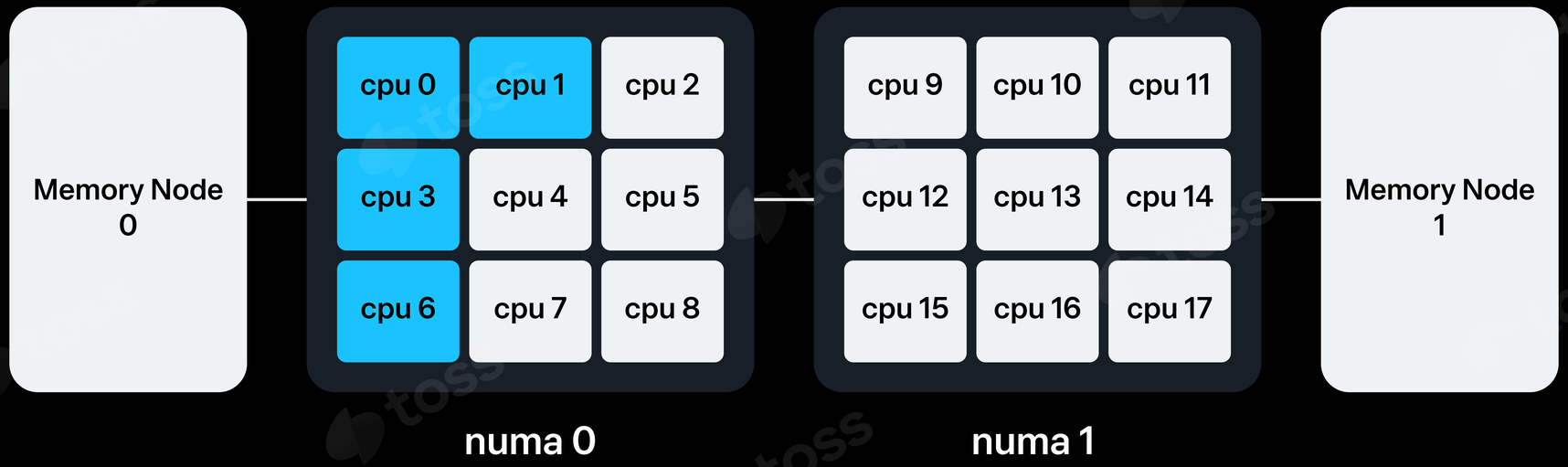
cpuset_memory



kubernetes에서 해결하는 방법

cpu pinning

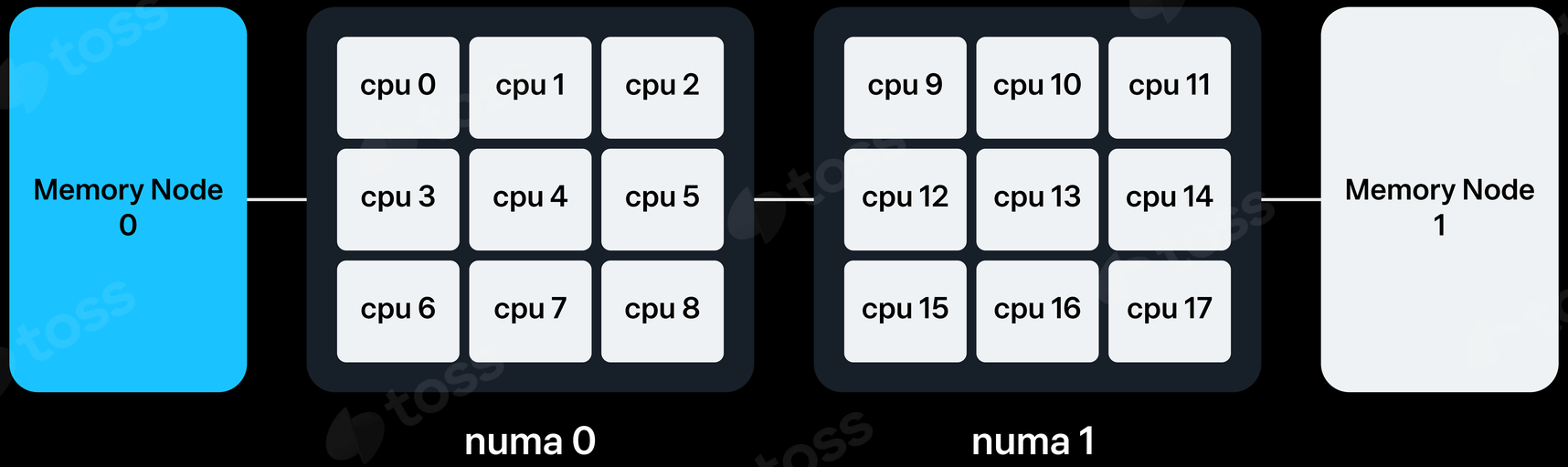
단점 → 메모리할당이 numa1에서도 일어날 수 있음, cpu를 피닝된 코어 이상 못씀



kubernetes에서 해결하는 방법

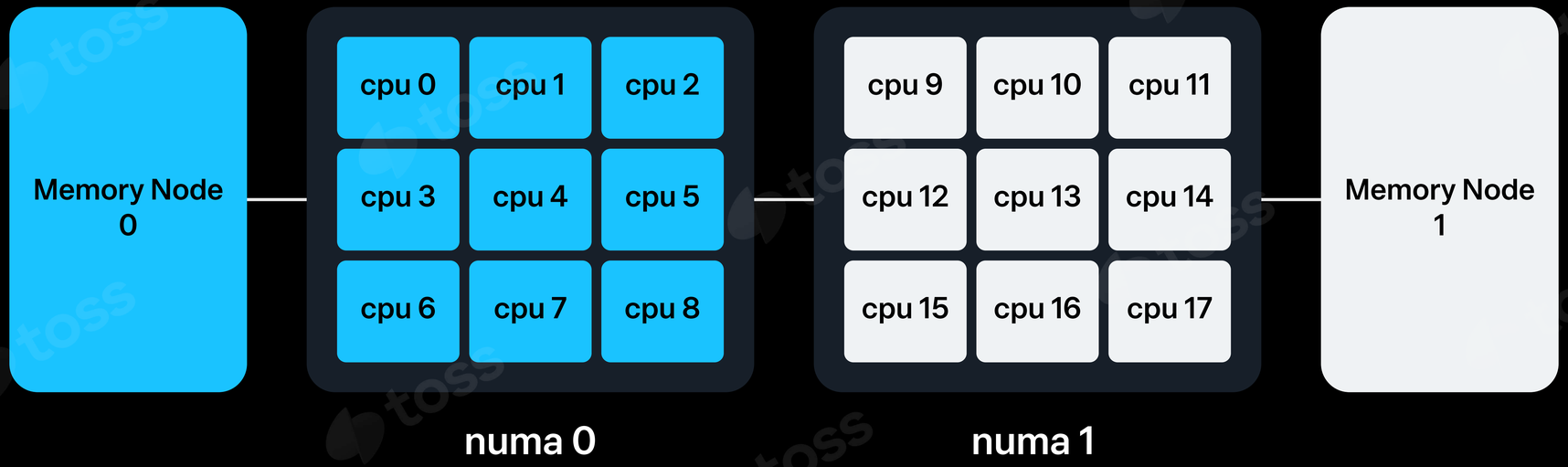
memory pinning

단점 → cpu가 소켓을 왔다 갔다 할 수 있음



socket 피닝 전략

socket pinning

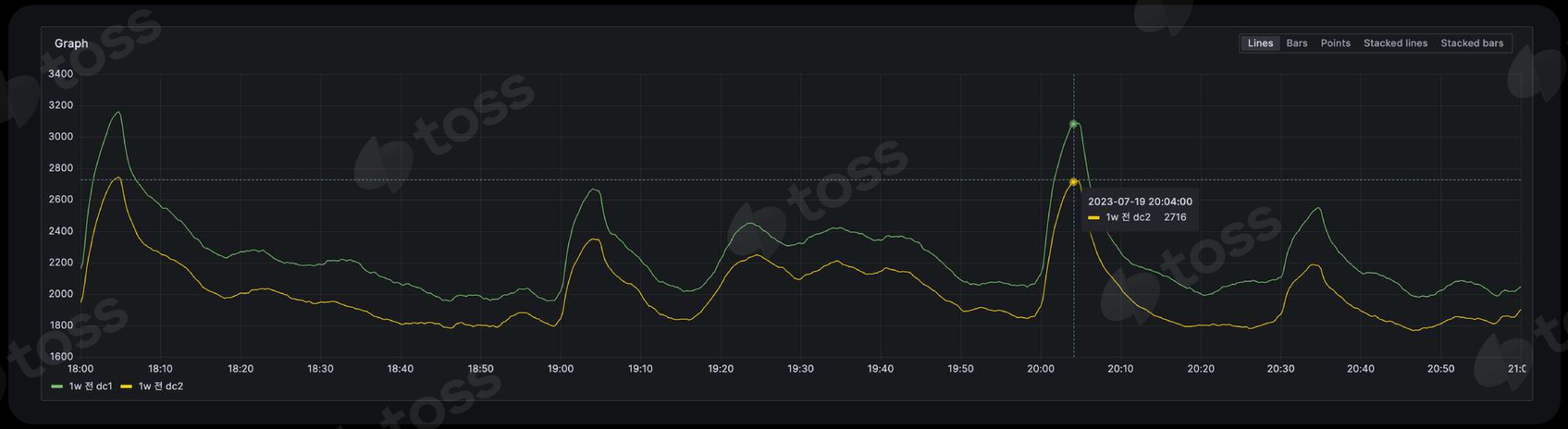


효과

```
TIM  NUMA_migrations NUMA_migratims_ms
15:39:10         27              0
15:39:11          0              0
15:39:12          0              0
15:39:13          3              0
15:39:14          1              0
15:39:15          2              0
15:39:16          0              0
15:39:17          2              0
15:39:18        323              0
15:39:19          1              0
15:39:20          0              0
15:39:21          0              0
15:39:22          0              0
15:39:23          0              0
15:39:24          2              0
15:39:25         10              0
```

효과

클러스터의 전체 cpu 사용률 13% 성능 향상

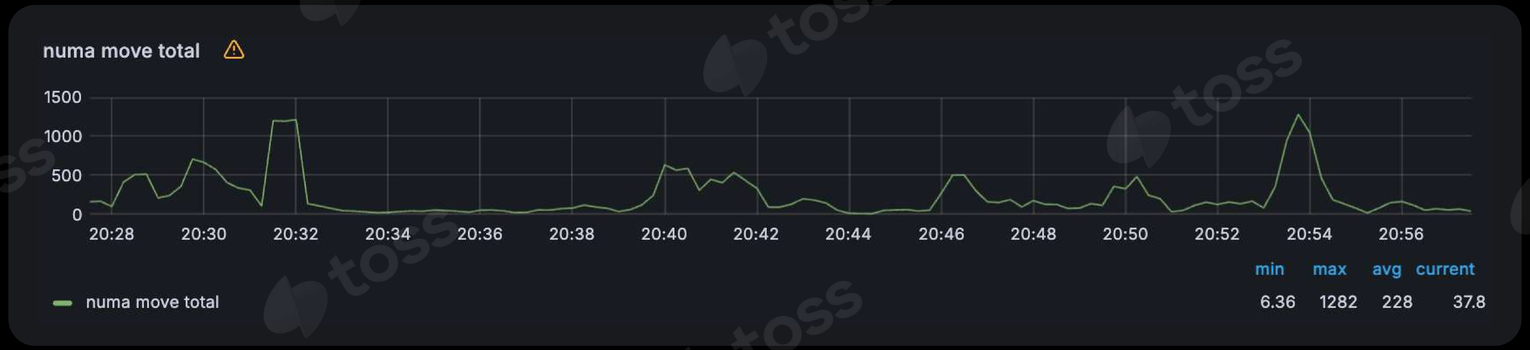


numa 모니터링

cloudflare 사에 ebpf exporter

ebpf_exporter_numa_latency_total

ebpf_exporter_numa_move_total



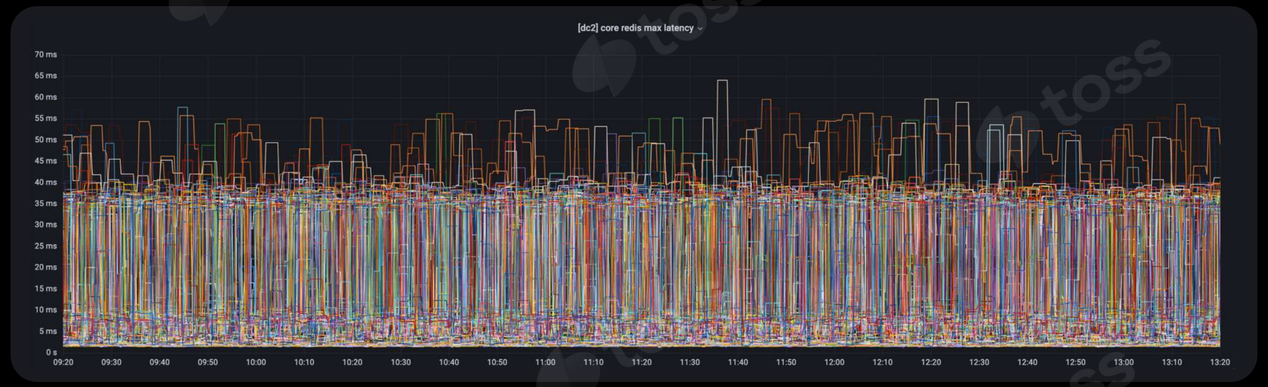
중간 마무리

ebpf 를 통해 cpu 비효율 찾기

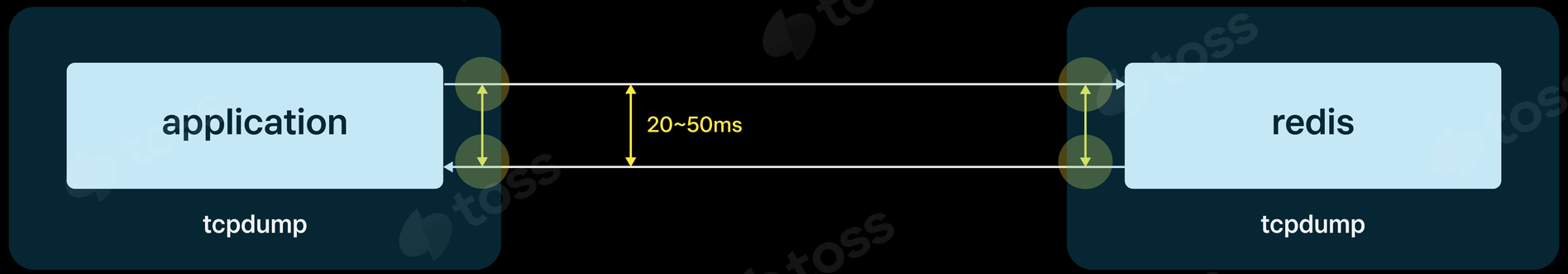
컨테이너의 CPU/Memory 물리적 위치를 조정해 최적화하기

redis latency 이상현상 파악

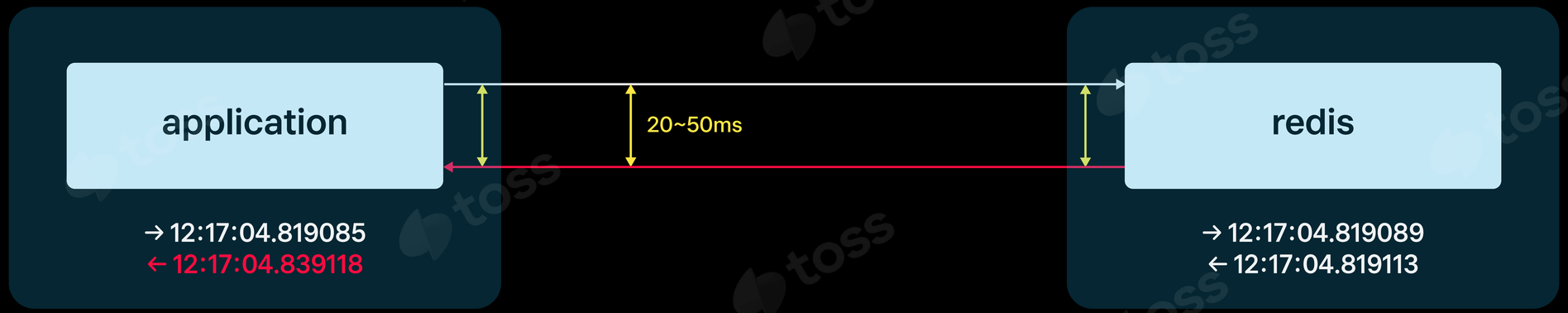
redis latency 에 가끔 튀는 메트릭이 발생



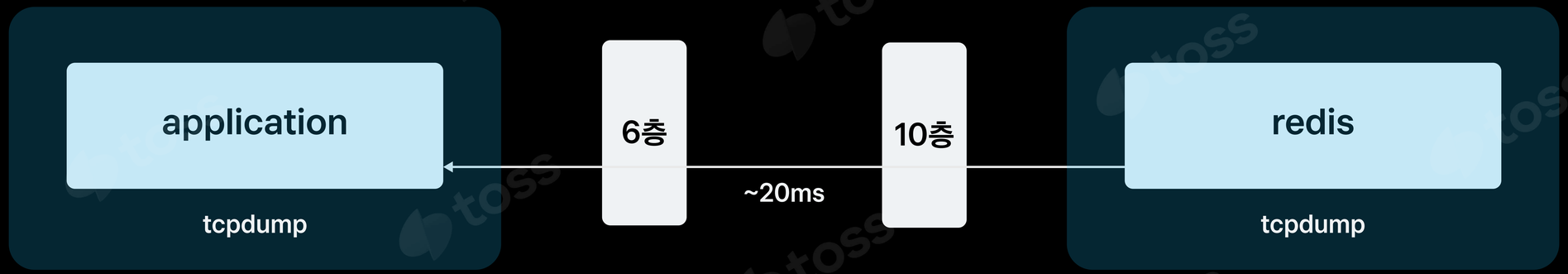
redis latency 이상현상 파악



redis latency 이상현상 파악



redis latency 이상현상 파악



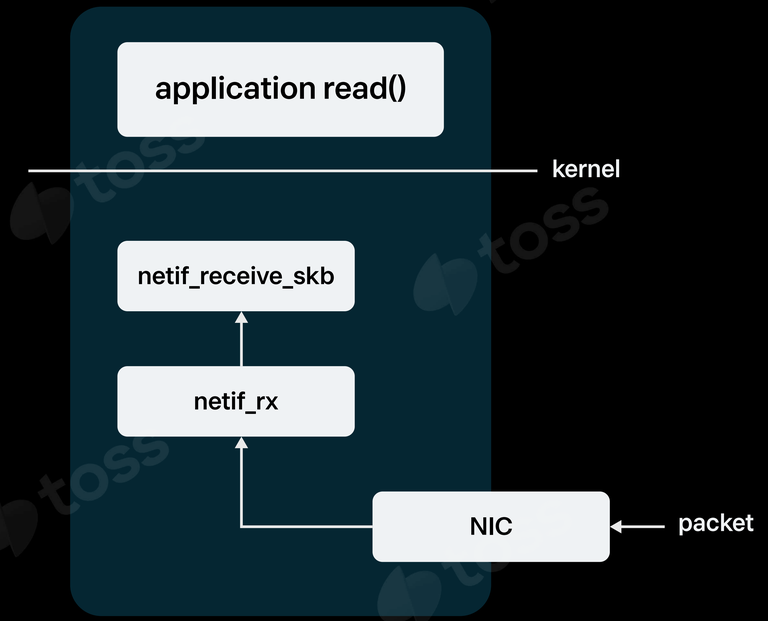
redis latency 이상현상 파악

ebpf 를 활용한 원인분석

perf trace -T -e 'net:*' -o output

스크립트로 분석 시도

```
816387609.780 :0/0 net:netif_rx_entry(name: "eth0", napi_id: 47, queue_mapping: 1, skbaddr: 0xffff912606d6c200, protocol: 2048…..
816387609.781 :0/0 net:netif_rx(skbaddr: 0xffff912606d6c200, len: 865, name: "eth0")
816387609.784 :0/0 net:netif_receive_skb(skbaddr: 0xffff912606d6c200, len: 865, name: "eth0")
```



ebpf 를 활용한 원인분석

perf trace → bpftrace → bcc tool

funclatency.py netif_rx

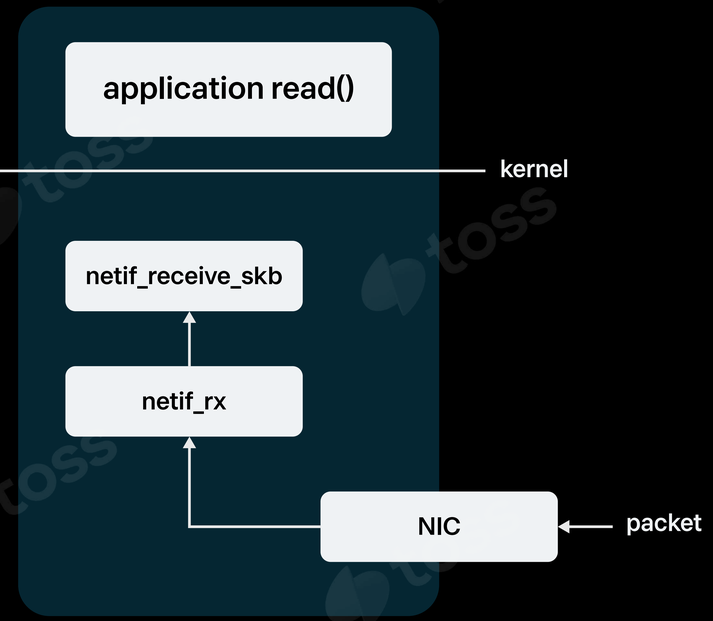
```
     nsecs               :count     distribution
         0 -> 1          :0         |                                        I
         2 -> 3          :0
         4 -> 7          :0
         8 -> 15         :0
        16 -> 31         :0
        32 -> 63         :0
        64 -> 127        :0
       128 -> 255        :2451
       256 -> 511        :1143420   |***************
       512 -> 1023       :1832756   |************************
      1024 -> 2047       :2344712   |*******************************
      2048 -> 4095       :2762852   |************************************   ******I
      4096 -> 8191       :906316    |************
      8192 -> 16383      :80498     |*
     16384 -> 32767      :7205      |
     32768 -> 65535      :685       |
     65536 -> 131071     :11        |

avg = 2110 nsecs, total: 19186838155 nsecs, count: 9092057
```

pixie (ebpf 모니터링)

syscall__probe_entry_readv

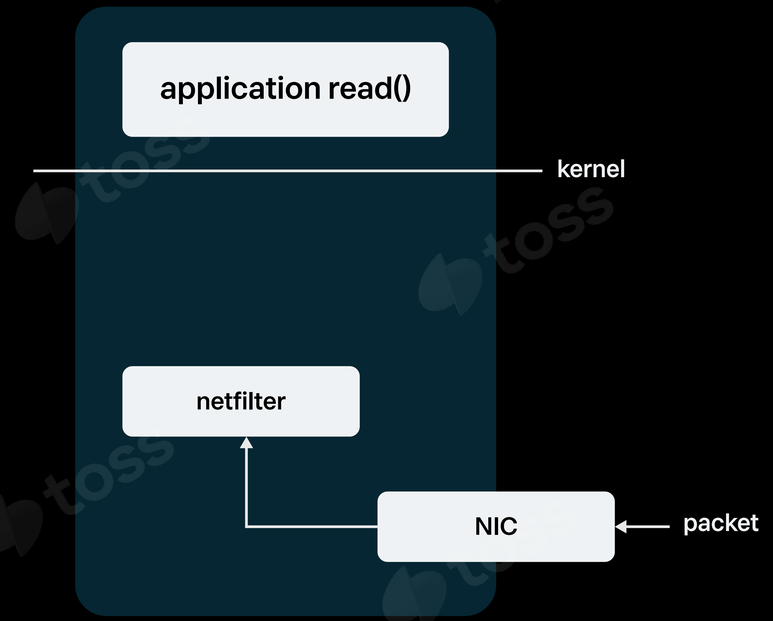
syscall__probe_ret_readv



ebpf 를 활용한 원인분석

k8s iptable → ipvs, calico policy(iptable)?

nflatency.py 로 확인



ebpf 를 활용한 원인분석

nflatency.py

PRE_ROUTING, FORWARD 등

microsecond(us) 로 성능 측정

```
Bucket = IPV4 FORWARD (tcp other)

    value              : count   distribution
        0 -> 1         : 17637   |****************************************
        2 -> 3         : 12826   |*****************************
        4 -> 7         : 12426   |****************************
        8 -> 15        : 2982    |******
       16 -> 31        : 472     |*
```

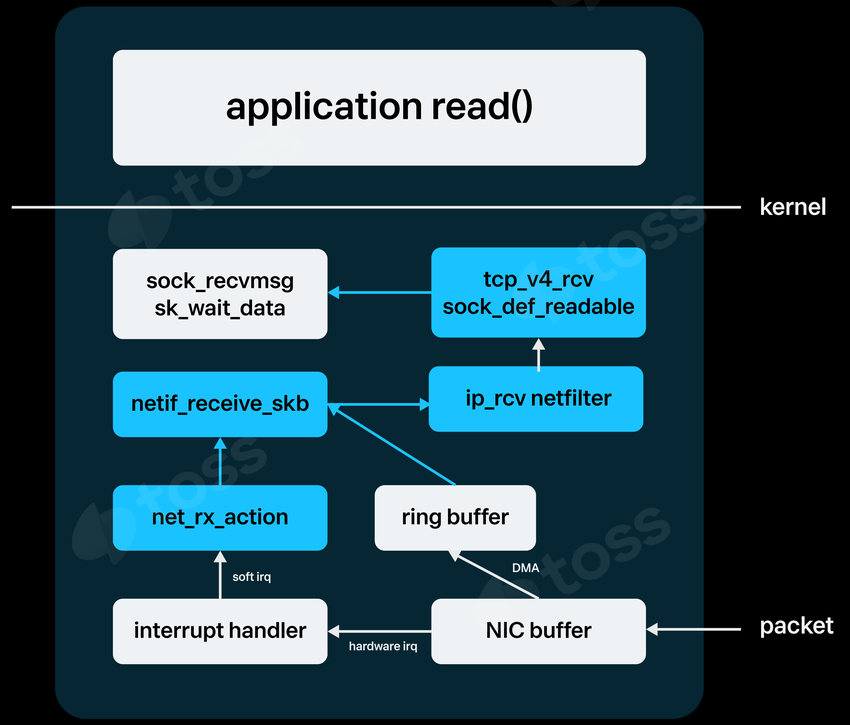
```
[DC1@root@k8s-slave469-dc1 tools]# ./funclatency nf_hook_slow
Tracing 1 functions for "nf_hook_slow"... Hit Ctrl-C to end.
AC
     nsecs                :count      distribution
         0 -> 1           :0          I                                      I
         2 -> 3           :0
         4 -> 7           :0
         8 -> 15          :0
        16  -> 31         :0
        32  -> 63         :0
        64  -> 127        :0
       128  -> 255        :2451
       256  -> 511        :1143420    I***************
       512  -> 1023       :1832756    i************************
      1024  -> 2047       :2344712    1*******************************
      2048  -> 4095       :2762852    I*********************************  ******I
      4096  -> 8191       :906316     1************
      8192  -> 16383      :80498      1*
     16384  -> 32767      :7205       I
     32768  -> 65535      :685        I
     65536  -> 131071     :11         I

avg = 2110 nsecs, total: 19186838155 nsecs, count: 9092057

Detachin....
```

ebpf 를 활용한 원인분석

funclatency.py로 rx_action 측정

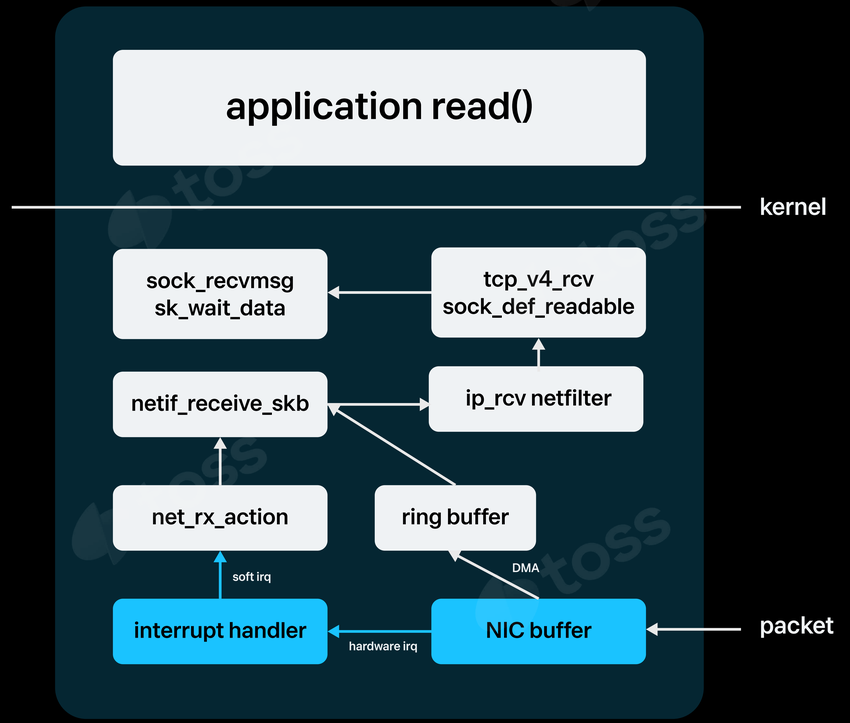


ebpf 를 활용한 원인분석

softirqs 로 네트워크 IRQ 측정

```
softirq = net_rx
     usecs               : count     distribution
         0 -> 1          : 143538   |********************                    |
         2 -> 3          : 63354    |*********                               |
         4 -> 7          : 168865   |************************                |
         8 -> 15         : 277198   |****************************************|
        16 -> 31         : 124451   |*****************                       |
        32 -> 63         : 14040    |**                                      |
        64 -> 127        : 965      |                                        |
       128 -> 255        : 12       |                                        |
       256 -> 511        : 3        |                                        |
       512 -> 1023       : 1        |                                        |
```

```
hardirq =
    usecs        :count  distribution
      0 ->1      : 0
      2 ->3       : 0
      4 ->7       : 0
      8 ->15      : 0
      16->31      : 0
      32->63       : 0
      64->127     : 0
     128->255     : 0
     256->511     : 0
     512->1023   :5102   |*********
    1024->2047   :20617  |****************************************
    2048->4095   : 4832   |*********
    4096->8191   :12
```



SLASH 24

ebpf 를 활용한 원인분석

왜 timer_softirq가 느리지?

```
softirq = timer
     usecs               : count     distribution
         0 -> 1          : 34718    |****************************************|
         2 -> 3          : 11238    |************                            |
         4 -> 7          : 6176     |*******                                 |
         8 -> 15         : 2907     |***                                     |
        16 -> 31         : 308      |                                        |
        32 -> 63         : 37       |                                        |
        64 -> 127        : 3        |                                        |
       128 -> 255        : 0        |                                        |
       256 -> 511        : 0        |                                        |
       512 -> 1023       : 0        |                                        |
      1024 -> 2047       : 0        |                                        |
      2048 -> 4095       : 0        |                                        |
      4096 -> 8191       : 0        |                                        |
      8192 -> 16383      : 0        |                                        |
     16384 -> 32767      : 6        |                                        |
```

```
./funclatency run_timer_softirq
Tracing 1 functions for "run_timer_softirq"... Hit Ctrl-C to end.
^C
     nsecs               : count     distribution
...
  16777216 -> 33554431   : 3        |                                        |
  33554432 -> 67108863   : 1        |                                        |

avg = 4493 nsecs, total: 376620502 nsecs, count: 83818
```

```
void __init init_timers(void)
{
     init_timer_cpus();
     open_softirq(TIMER_SOFTIRQ, run_timer_softirq);
}
```

ebpf 를 활용한 원인분석

timer는 리스트라서?

bpftrace로 timer 분석

```
trace_timer_expire_entry(timer);
fn(timer);
trace_timer_expire_exit(timer);
```

```
tracepoint:timer:timer_expire_entry
{
    @start[args->timer] = nsecs;
    @fn[args->timer] = args->function;
}

tracepoint:timer:timer_expire_exit
/@start[args->timer]/
{
    $d = (nsecs - @start[args->timer]) / 1000;
    @usecs = hist($d);
    if ($d > 16000) {
        printf("slow!! %s %ps, %s\n", comm, @fn[args->timer], ksym(@fn[args->timer]))
    }
    delete(@start[args->timer]);
    delete(@fn[args->timer]);
}
```

ebpf 를 활용한 원인분석

estimation_timer!!

stack 에는 run_timer_softirq!!

```
#bpftrace trace_timer.bt
Attaching 4 probes...
Tracing timer ... Hit Ctrl-C to end.
slow!! wrk:worker_0 0xffffffffc07c4a70s, estimation_timer
slow!! swapper/44 0xffffffffc07c4a70s, estimation_timer
```

```
./stackcount estimation_timer
  b'estimation_timer'
  b'call_timer_fn'
  b'run_timer_softirq'
  b'__softirqentry_text_start'
  b'irq_exit_rcu'
  b'irq_exit'
  b'smp_apic_timer_interrupt'
  b'apic_timer_interrupt'
```

ebpf 를 활용한 원인분석

규모가 큰 경우 estimation 문제

1. kernel 버전업 후 skip 설정
2. kernel patch
3. cillium ebpf 변경

```
--- a/net/netfilter/ipvs/ip_vs_est.c
+++ b/net/netfilter/ipvs/ip_vs_est.c
@@ -100,6 +100,9 @@ static void estimation_timer(struct timer_list *t)
     u64 rate;
     struct netns_ipvs *ipvs = from_timer(ipvs, t, est_timer);

+    if (!sysctl_run_estimation(ipvs))
+        goto skip;
+
     spin_lock(&ipvs->est_lock);
     list_for_each_entry(e, &ipvs->est_list, list) {
         s = container_of(e, struct ip_vs_stats, est);
@@ -131,6 +134,8 @@ static void estimation_timer(struct timer_list *t)
```

마무리하며

ebpf 로 그동안 알지 못했던 영역까지 observability 향상

근본적인 원인을 찾을 수 있었던 경험

SLASH 24