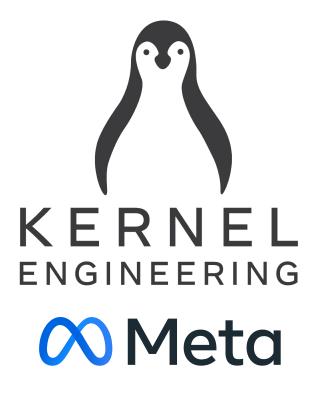
#### Reentrant kmalloc for any context



# Existing slab wrappers

```
- bpf_mem_alloc
```

- kretprobe objpool

- ...

# BPF's preallocated past...

- 11 years ago the only BPF use cases were networking and tracing
  - tracing context is unknown. NO kmalloc
  - networking context is typically BH. Ok to kmalloc
- By default BPF maps are preallocated
  - all elements of a hash table are preallocated in per-cpu freelists
  - Ex: max\_entries=10k hashtab on 100 cpus will have 100 elements in each per-cpu freelist
  - push/pop can happen on different cpus, simple round robin stealing from other cpu-s
  - freed elements are instantly reused
  - high performance
- Hash map rarely used at full capacity to maintain O(1) performance
  - Lots of preallocated memory is unused

# BPF's preallocated past...

- networking could use BPF\_F\_NO\_PREALLOC to mitigate memory waste
  - kmalloc(GFP\_ATOMIC), call\_rcu() in free path
    - can have memory spikes
  - performance could be several times slower depending on usage
  - rarely used in practice
  - cannot be used when tracing and networking share a map

#### bpf\_mem\_alloc to the rescue

- kmalloc-like per-cpu buckets {96, 192, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096}
- NMI-safe per-cpu freelists with low/hi watermark
  - average of 64 per-cpu elements for smaller buckets
- Once below low watermark raise irq\_work to refill
- High performance
- Global bpf\_mem\_alloc with buckets
- kmem\_cache\_create()-like bpf\_mem\_alloc for fixed size elements

#### bpf\_mem\_alloc issues

- allocations from irq disabled context are unpredictable
- irq\_work\_queue() is/was broken on some arm64
- Memory is pinned in free list for bpf system only
  - each bpf hash map uses its own one size bpf\_mem\_alloc instance
  - Rest of bpf uses global bpf mem\_alloc
- Not as bad memory waste as full prealloc, but
  - global bpf\_mem\_alloc mirrors kmalloc-\* slubs
  - fixed size bpf\_mem\_allocs mirror kmem\_cache\_create-d slubs
  - merging is needed
- Do not want to reinvent slub features one by one

# bpf\_mem\_alloc issues

- async refill is ok for small sizes
  - 1k+ objects have infeasible trade-off:
    - either excessive memory waste
    - or sporadic allocation failures
- kmalloc is synchronous. Good.

# Synchronous kmalloc stack

```
- kmalloc
- slab_alloc_node
- cmpxchg16
- __slab_alloc
- new_slab
- alloc_pages
- rmqueue_pcplist
- __rmqueue
- wakeup_kswapd // when __GFP_KSWAPD_RECLAIM is set
```

# Synchronous kmalloc stack

```
- kmalloc
- slab_alloc_node
- cmpxchg16
- __slab_alloc
- new_slab
- alloc_pages
- rmqueue_pcplist
- __rmqueue
- wakeup_kswapd // when __GFP_KSWAPD_RECLAIM is set
```

# try\_alloc\_pages implementation details

- rmqueue\_pcplist
  - already spin\_trylock(). No changes were necessary.
- \_\_rmqueue
  - convert spin\_lock\_irqsave to spin\_trylock\_irqsave if (alloc\_flags & ALLOC\_TRYLOCK)
- try\_charge\_memcg
  - convert local\_lock\_irqsave to localtry\_trylock\_irqsave if (gfpflags\_allow\_spinning(gfp\_mask))

```
static inline bool gfpflags_allow_spinning(const gfp_t gfp_flags)
         * !__GFP_DIRECT_RECLAIM -> direct claim is not allowed.
         * !__GFP_KSWAPD_RECLAIM -> it's not safe to wake up kswapd.
         * All GFP * flags including GFP_NOWAIT use one or both flags.
         * try alloc pages() is the only API that doesn't specify either flag.
         * This is stronger than GFP_NOWAIT or GFP_ATOMIC because
         * those are guaranteed to never block on a sleeping lock.
         * Here we are enforcing that the allocation doesn't ever spin
         * on any locks (i.e. only trylocks). There is no high level
         * GFP_$F00 flag for this use in try_alloc_pages() as the
         * regular page allocator doesn't fully support this
         * allocation mode.
         */
        return !!(gfp_flags & __GFP_RECLAIM);
```

# Reentrant kmalloc for any context

```
- kmalloc
- slab_alloc_node
- cmpxchg16
- __slab_alloc
- new_slab
- alloc_pages
- rmqueue_pcplist
- __rmqueue

- try_kmalloc
try_kmalloc
try_kmalloc
try_kmalloc
try_kmalloc
```

#### Plan so far

- in get\_freelist()... support \_\_CMPXCHG\_DOUBLE only (for now)
  - punt on try-locking bit\_spin\_lock to later
  - essentially no changes in fast path
- s/local\_lock/localtry\_trylock/ in slow path
- if pfmemalloc mismatch return ENOMEM
- in \_\_\_slab\_alloc() return ENOMEM when trylock fails
  - ignore node mismatch? pretend to be NUMA\_NO\_NODE
  - don't use get\_partial()
- new\_slab() -> try\_alloc\_pages()
- kmem\_cache\_debug() will work as-is with trylock