A Memory Pool Allocator for eBPF Applications

Gyuyeong Kim and Dongsu Han

성신여자대학교
SUNGSHIN WOMEN'S UNIVERSITY

KAIST

In-Kernel Offloading with eBPF

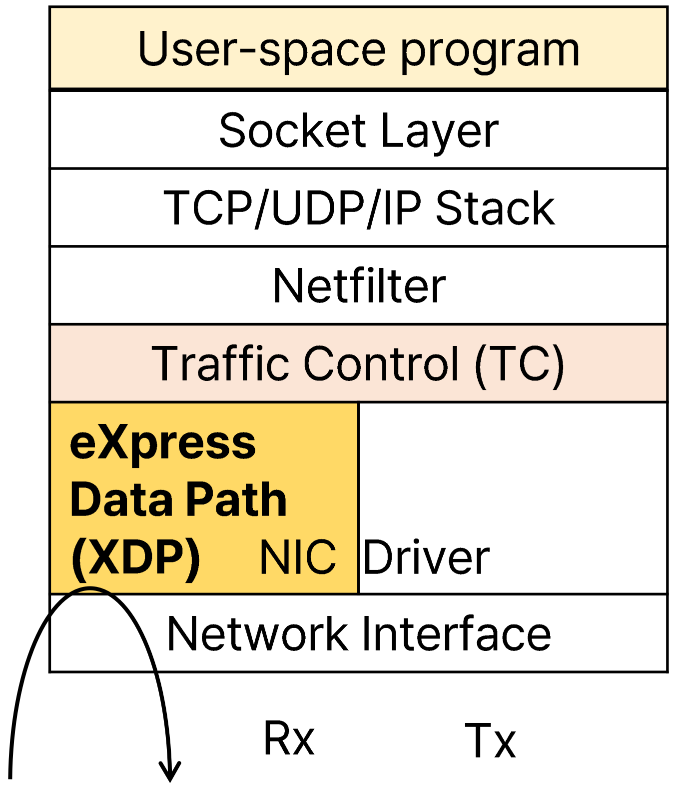
eBPF enables kernel-level execution of application logic

- High performance by avoding networking stacks
- Safety guarantee through static verification

Key-value stores [BMC@NSDI'21, DINT-KV@NSDI'24]

Consensus protocol [Electrode@NSDI'23]

Lock managers [DINT-Lock@NSDI'24]



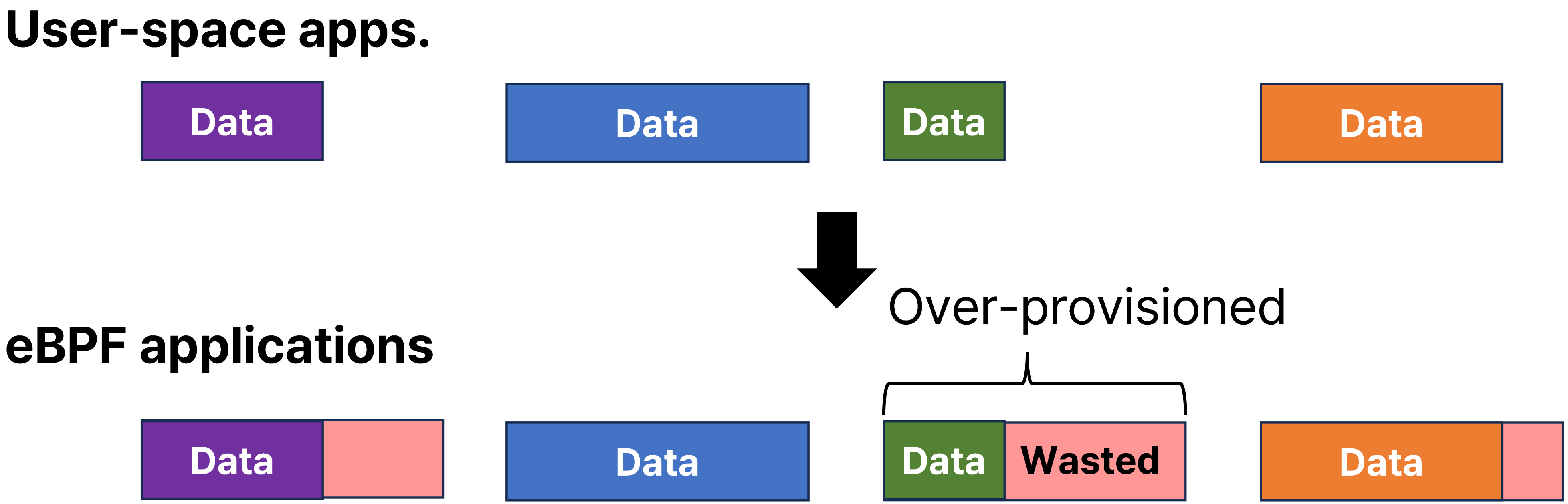
Memory Allocation Matters!

eBPF typically relies on **static memory allocation** for safety

Cannot allocate **variable-size** memory space

Runtime allocation is only possible for restricted cases

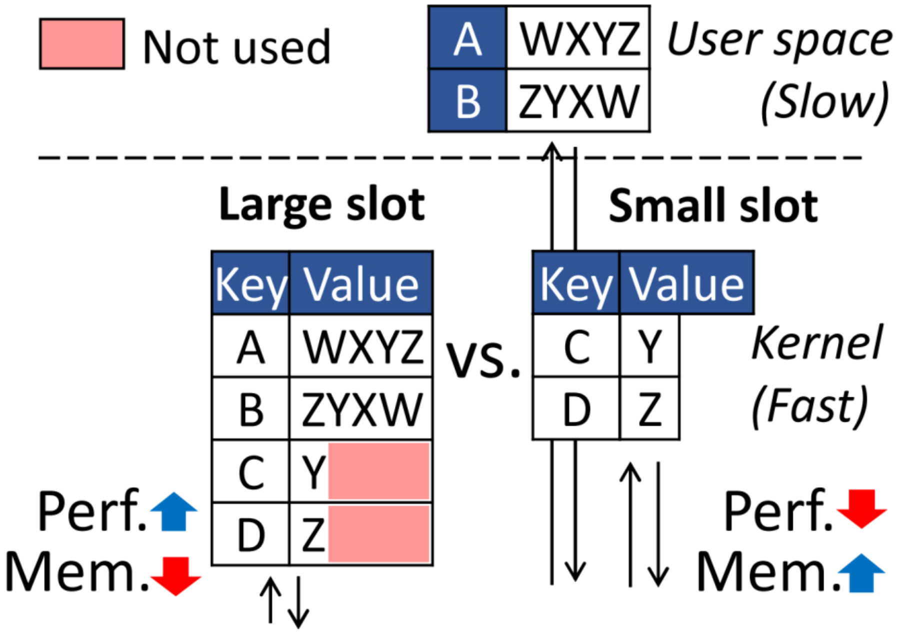
Critical for apps. with variable-size data or dynamic data structures



Trade-off Between Perf. And Mem. Efficiency

A large slot provides better performance but wastes memory

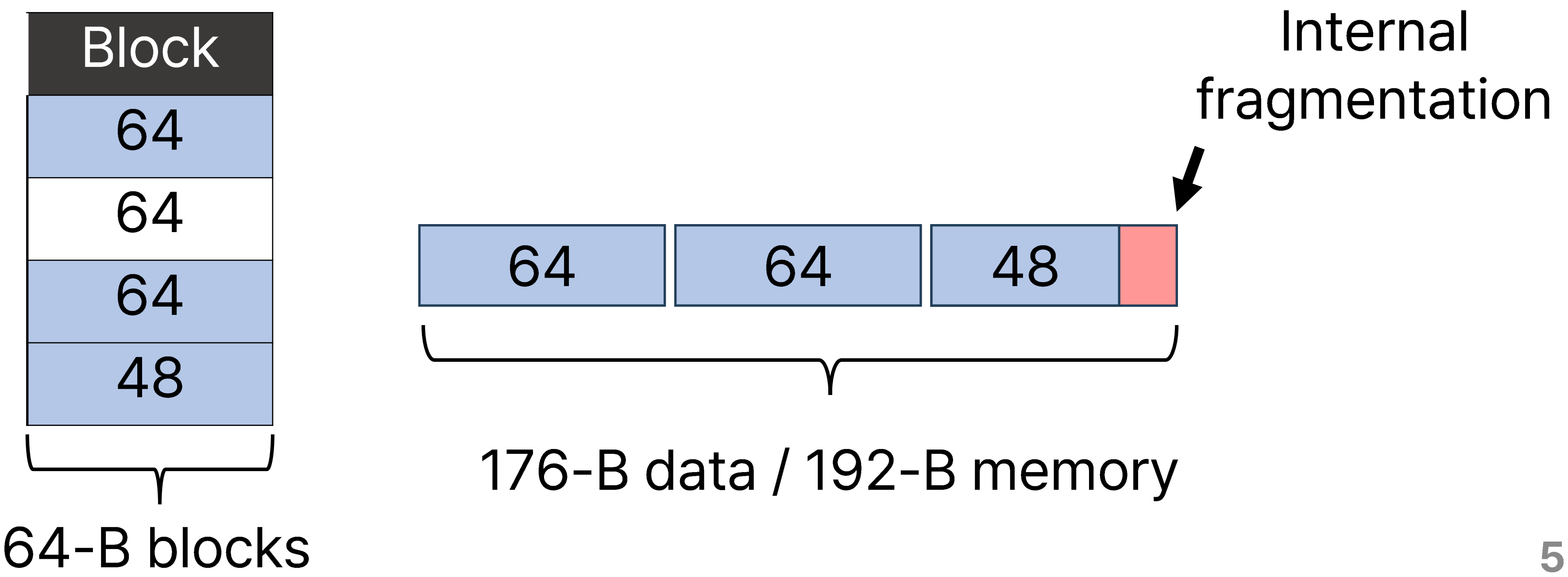
A small slot saves memory but degrades performance



BMC@NSDI'21
DINT-KV@NSDI'24

Kerby: A Memory Pool Allocator for eBPF

- **Idea:** dynamically manages a memory pool consisting of *fixed-size* memory blocks
  - Combines blocks to represent variable-size data
  - Fragmentation only happens in the tail block



Free List Management

The allocator should know which block is allocated or free

Bitmap-based free list?

- We cannot call functions while holding a lock in eBPF for safety reasons

Block Index	0	1	2	3	4	5	6	7	8
Allocated?	1	0	0	1	1	0	1	0	1

```
lock();
Update_freelist();
unlock();
```

Prohibited!

No Free List in Kerby

1. Monotonically-increasing index up to 2^{64}-1
2. Access within the bound using a BPF hash map for the pool
    - Internally hashses index, resolves hash collisions using chaining
    - `BPF_F_NO_PREALLOC` flag enables runtime allocation

Index Allocator

(MAP_ARRAY)

Counter
64-bit

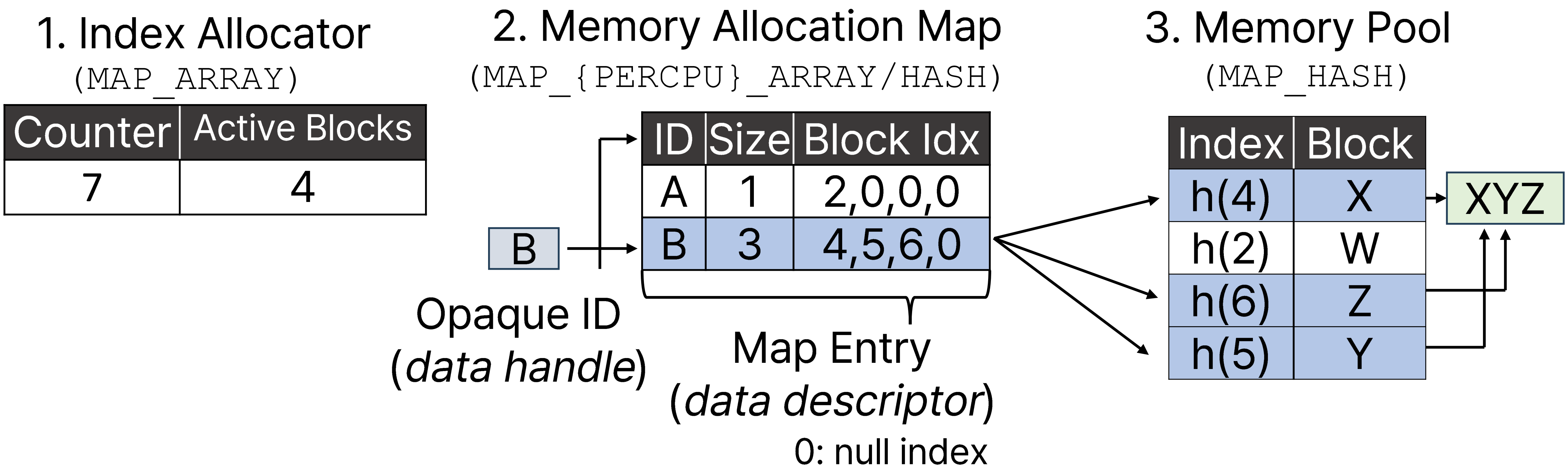
Memory Pool

(MAP_HASH)

Index	Block
h(2)	A
h(1)	D
h(3)	B
$h(2^{64} - 1)$	C

Kerby Overview

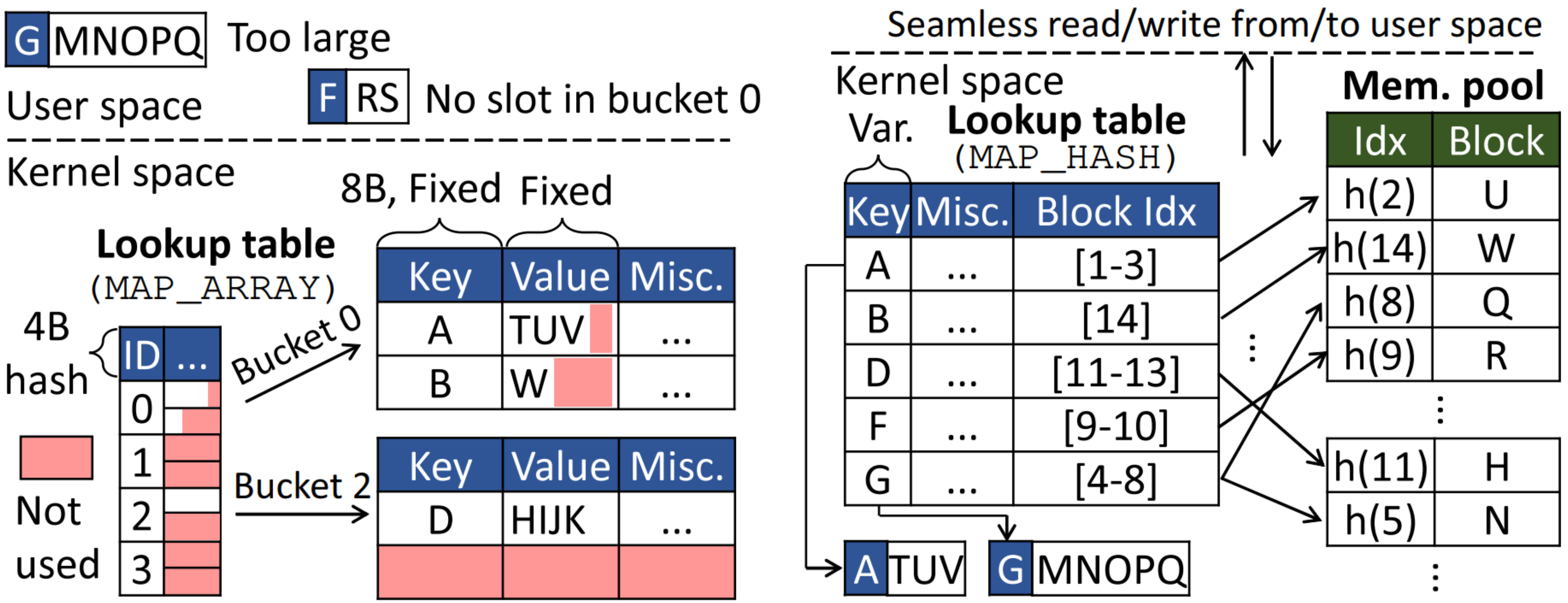
Kerby consists of three components



Key-Value Stores: Design

DINT-KV supports fixed-size items

Kerby-KV supports variable-size items

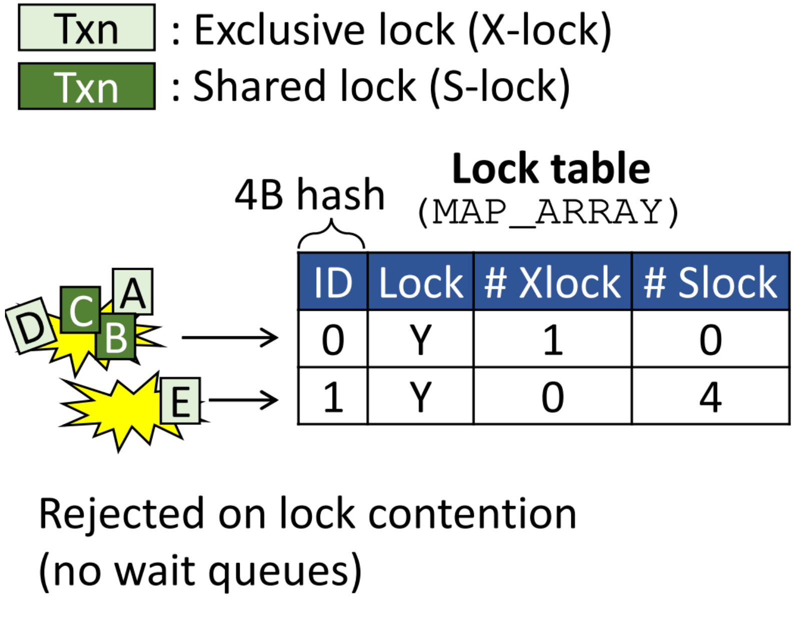


(a) DINT-KV (b) Kerby-KV

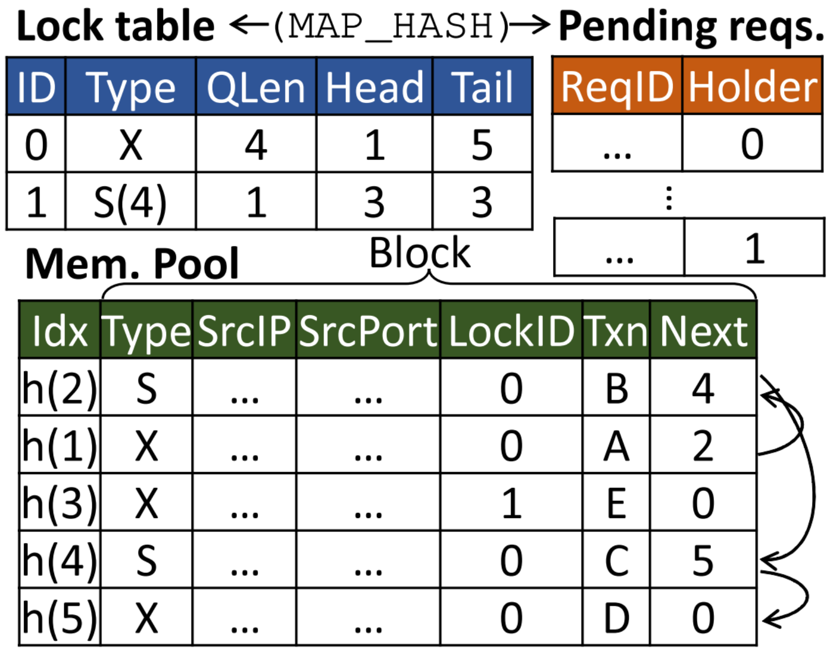
Lock Managers: Design

DINT-Lock lacks wait queues, cannot handle lock contention

Kerby-Lock supports dynamic variable-length wait queues by linking memory blocks <u>(dynamic linked-list)</u>



(a) DINT-Lock



(b) Kerby-Lock

Kerby APIs

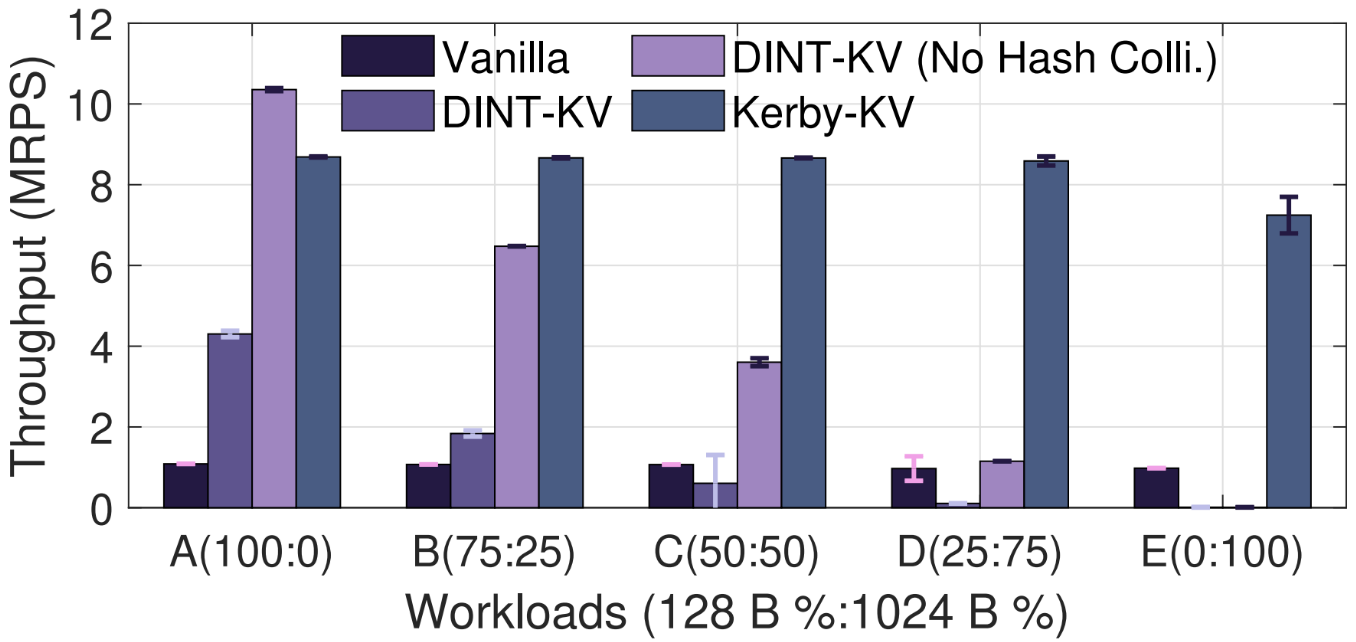
We expose developer-friendly APIs

Three primitives and extra helpers

Using only BPF maps and helpers, no kernel modifications

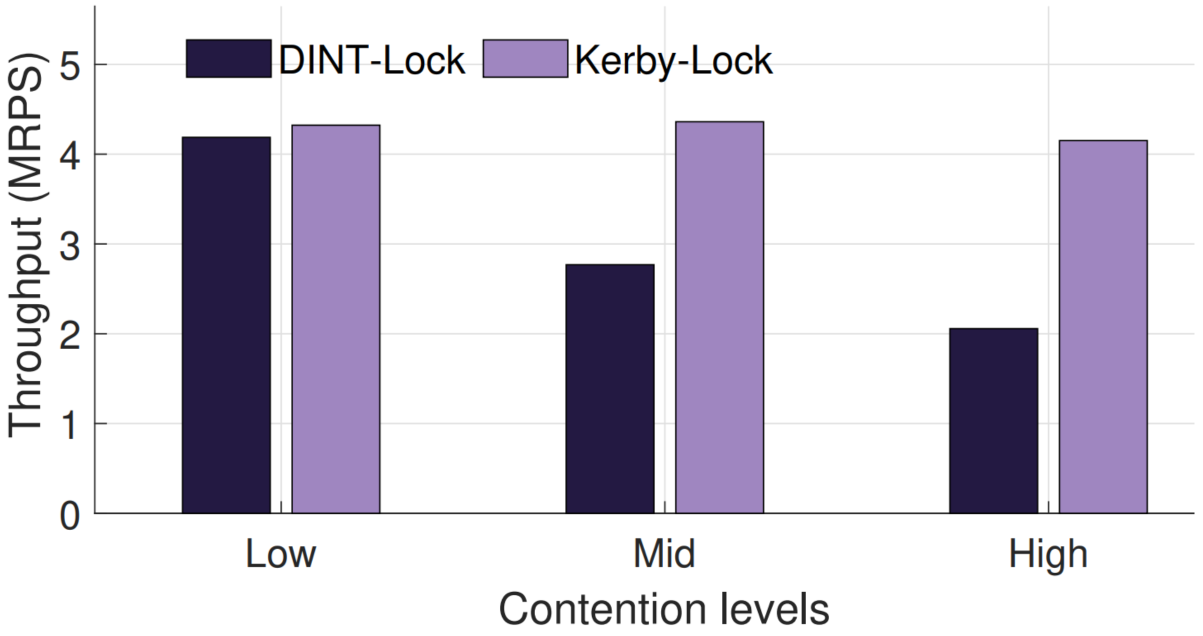
Abstract Signature	Category	Role
`malloc`(size, idx[])	Primitive	Allocate memory
`free`(idx[])	Primitive	Release memory
`resize`(size, idx[])	Primitive	Adjust allocation
`pool_update`(idx[], values)	Helper	Write blocks
`alloc_update`(key, idx[], size)	Helper	Update alloc. map
`alloc_del`(key)	Helper	Remove alloc. map

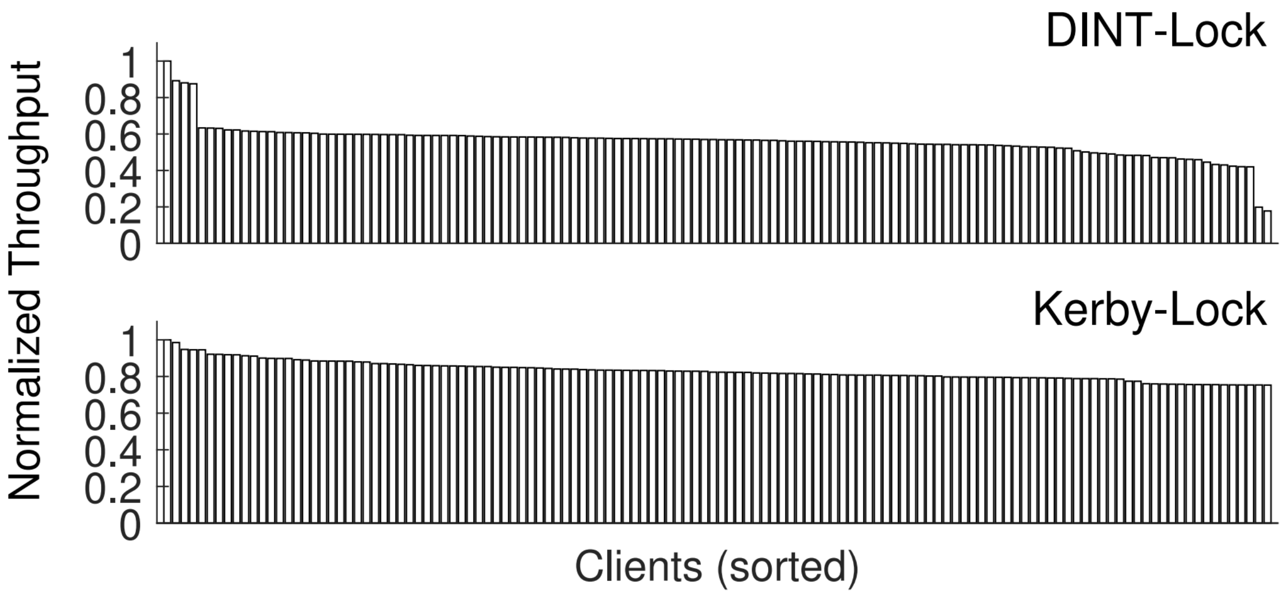
Key-Value Stores: Evaluation Results



Kerby-KV performs the best
DINT-KV spills 1024-B items to user space

Lock Managers: Evaluation Results





Kerby-Lock can handle high lock contention and achieves fairness

Conclusion

- Kerby is a dynamic eBPF memory pool allocator to overcome the limitation of static memory allocation in eBPF

- Kerby enables us to implement high-performance and memory-efficient eBPF applications through memory pooling

- Kerby can enrich existing and future eBPF applications

Thank you!

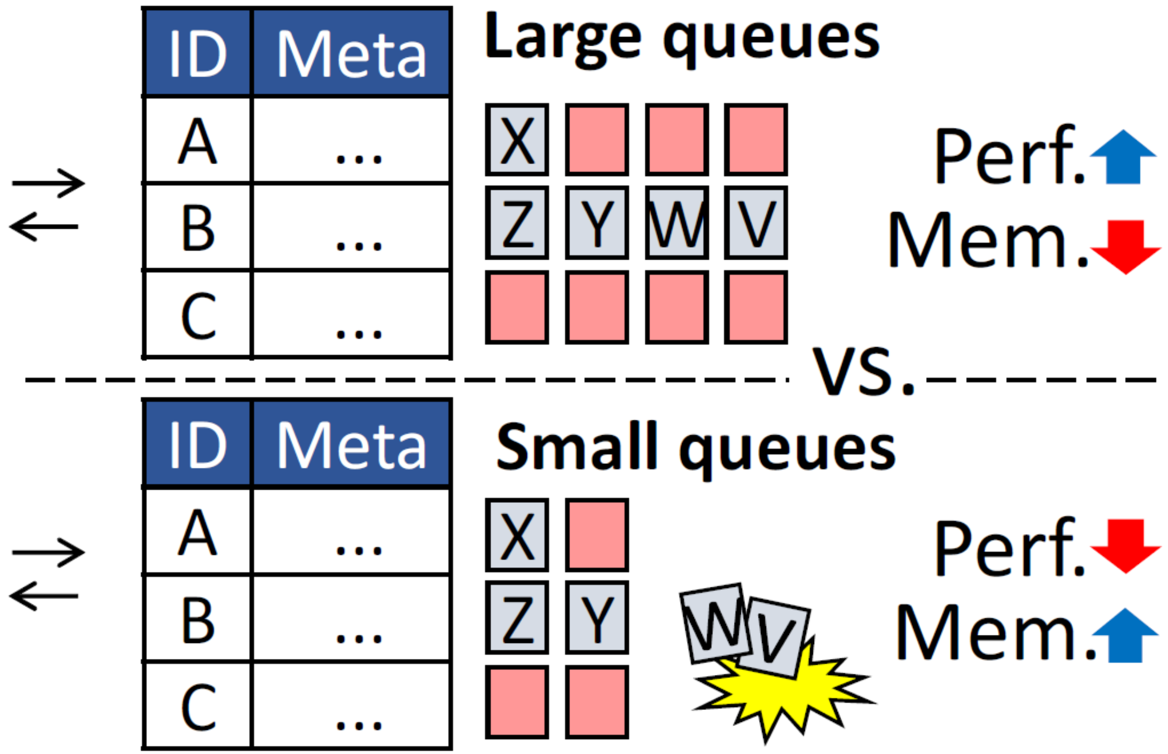
Questions?

Trade-off in Lock Managers

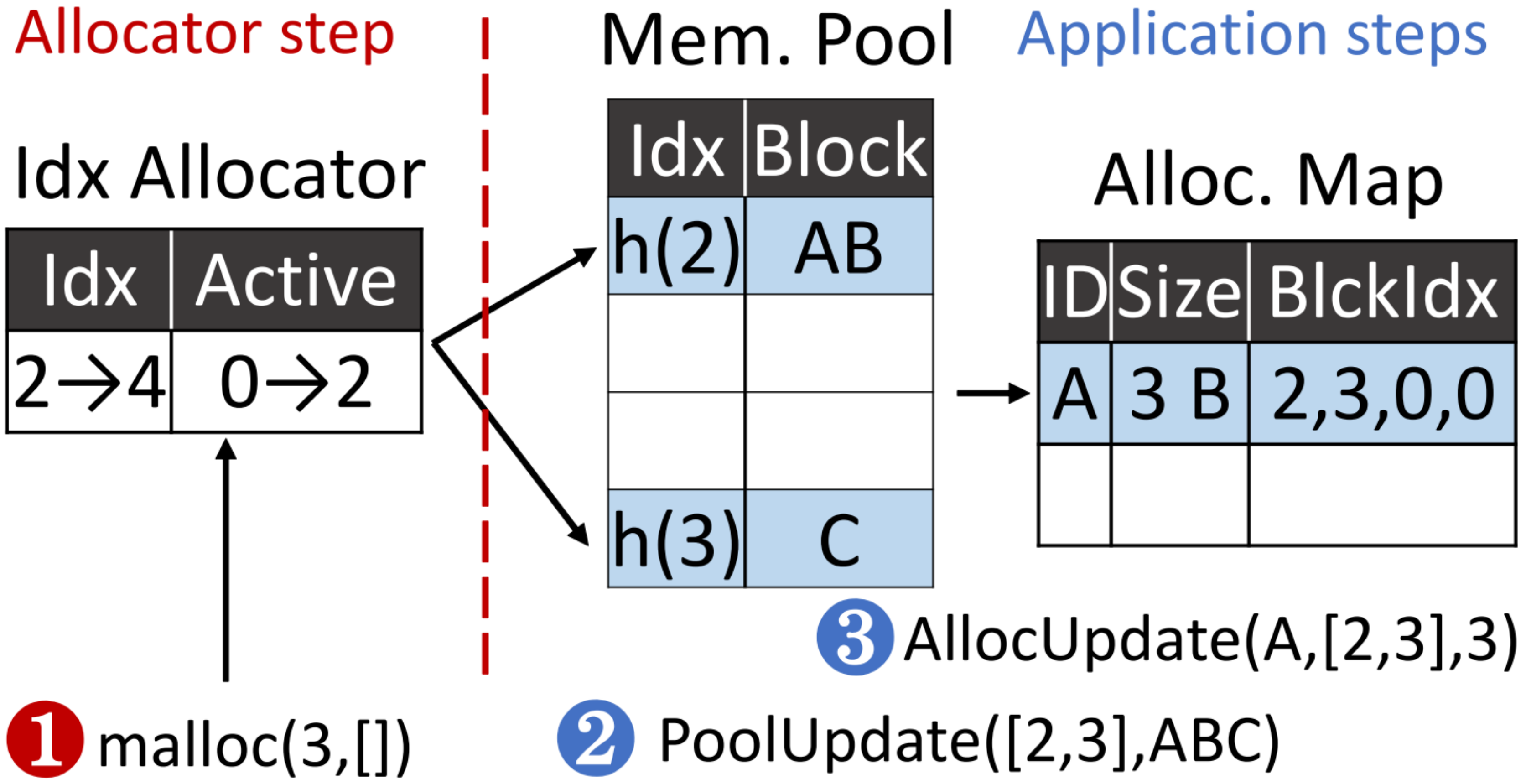
DINT-Lock@NSDI'24 lacks wait queues

We may implement wait queues using BPF queue maps

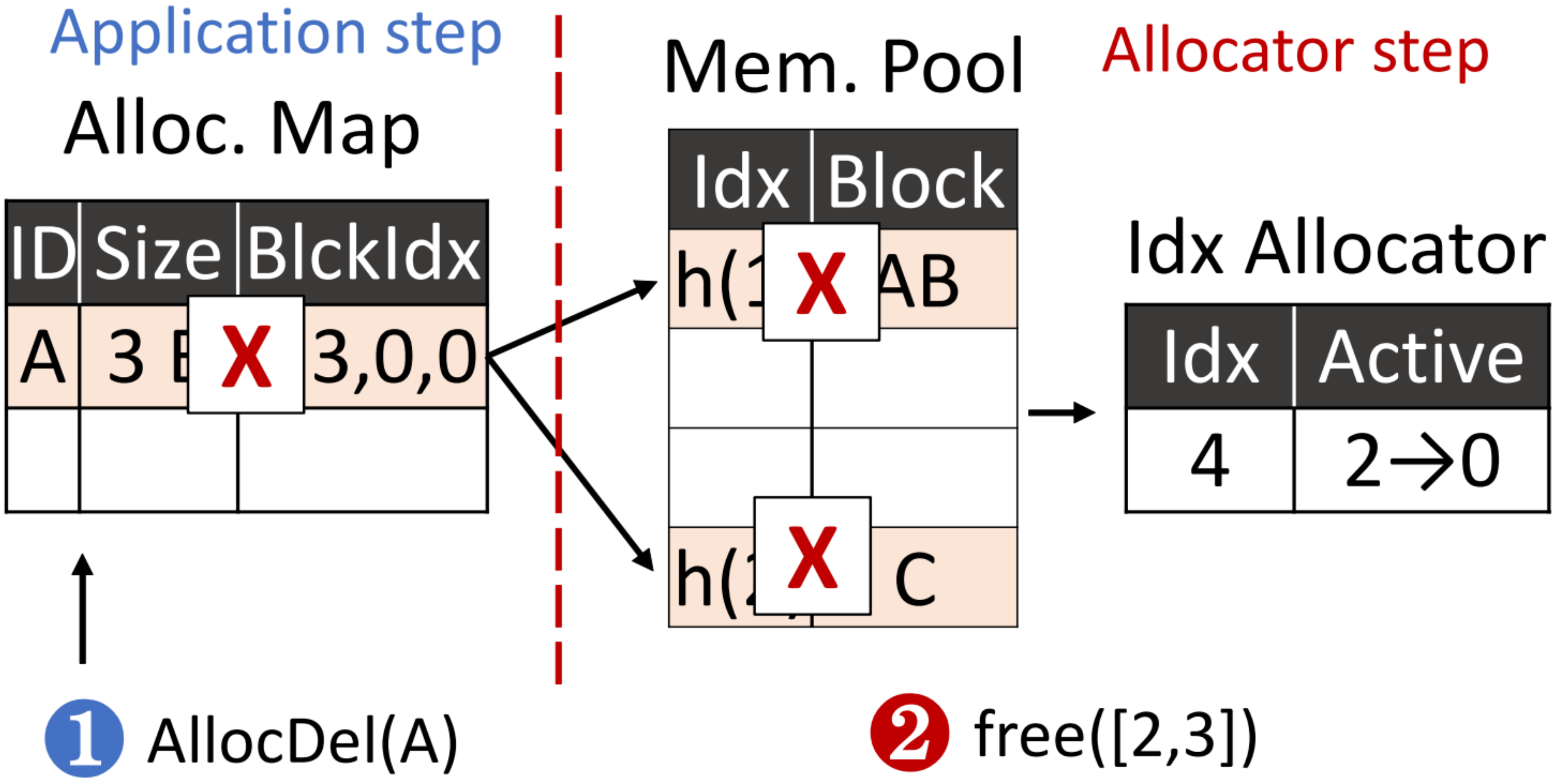
But, still has a trade-off between perf. and mem. efficiency



Allocation Workflow



Deallocation Workflow



Resizing Workflow