DINT: Fast In-Kernel Distributed Transactions with eBPF

* **Yang Zhou**

Harvard University

* Xingyu Xiang

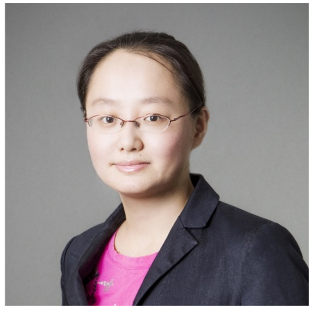
Peking University

Matthew Kiley

Harvard University

Sowmya Dharanipragada

Cornell University



Minlan Yu

Harvard University

* Co-primary author

Distributed Protocols for High Availability

Hadoop

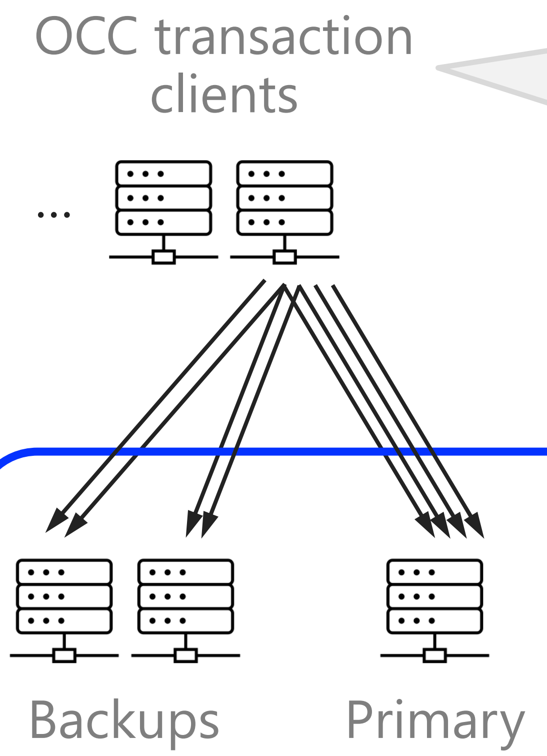
Kubernetes

YouTube

Electrode [NSDI'23] → Etcd running distributed consensus protocols

Spanner running distributed transaction protocols ← This work

Distributed Transactions inside a Datacenter




```
txn_read_lock(R, W)  // read read-set (R) versions + values,
                     // lock write-set (W)
// Update write-set locally
txn_validate(R)      // validate read-set versions
txn_log(W)           // log write-set updates
txn_commit           // update backups+ primary
```

Lock manager

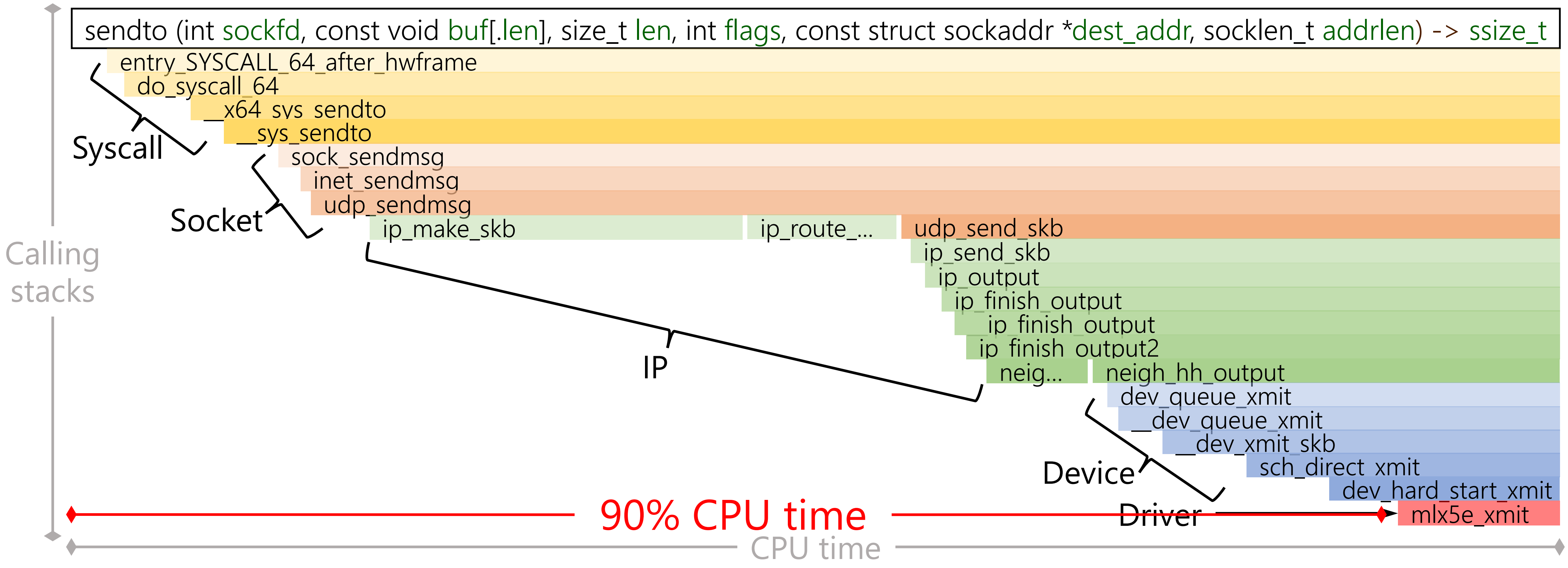
KV store

Log manager

Trend: storing states in memory
or persistent memory

Distributed transactions are network IO-intensive

Kernel Networking: High Kernel Overhead



Kernel Networking: High Kernel Overhead

Dist. protocols
Socket layer
Transport layer
…
Traffic control
NIC driver
NIC hardware

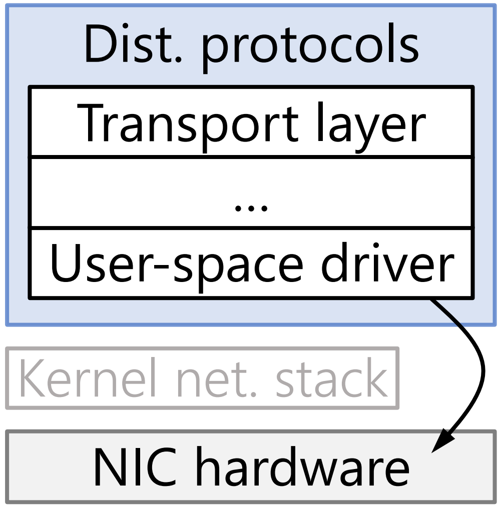
OCC distributed transactions

92% CPU time[1]

Only around $^{1}/_{10}$ is on NIC driver

[1] Experiment setting: TATP workload for the OCC transaction protocol with 3-way primary-backup replication and 3-way sharding, using UDP sockets from Linux kernel 6.1.0

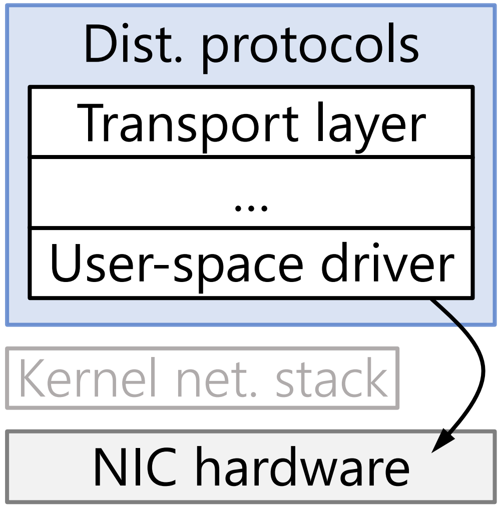
Kernel Bypass: Not a Panacea



DPDK (Data Plane Development Kit) or RDMA:

- Customized networking stacks in user space or NIC
- Busy polling instead of costly interrupt

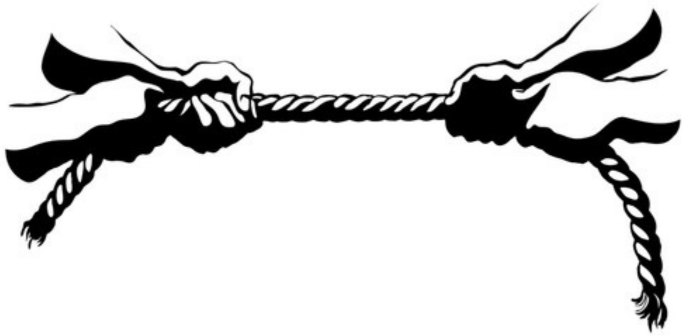
Kernel Bypass: Not a Panacea



DPDK (Data Plane Development Kit) or RDMA:

+ High performance
– Dedicated resources (eg, busy-polling cores)
– Security vulnerabilities (user manages NICs)[1,2]

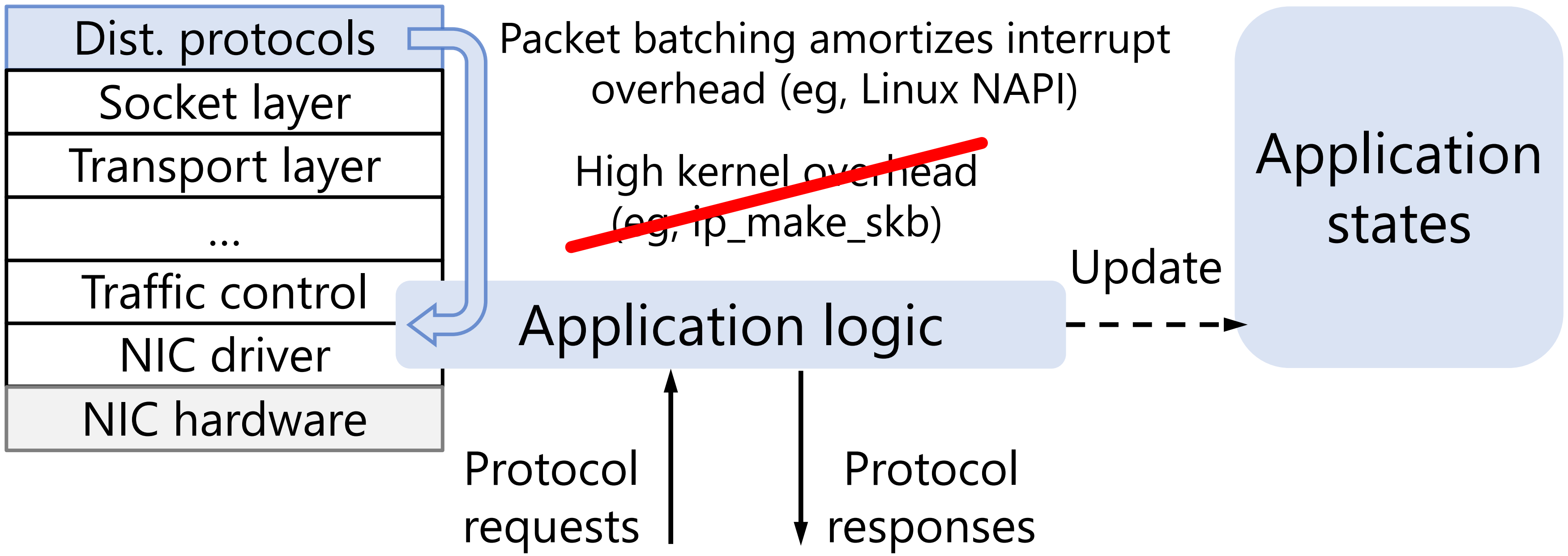
Kernel bypass:
high performance



Kernel:
resource sharing, security

[1] Bellovin, Steven M. "Security Problems in the TCP/IP Protocol Suite." SIGCOMM CCR 1989
[2] Smolyar et al. "Securing Self-Virtualizing Ethernet Devices." USENIX Security 2015

DINT[1]: Application-Customized Networking Stacks



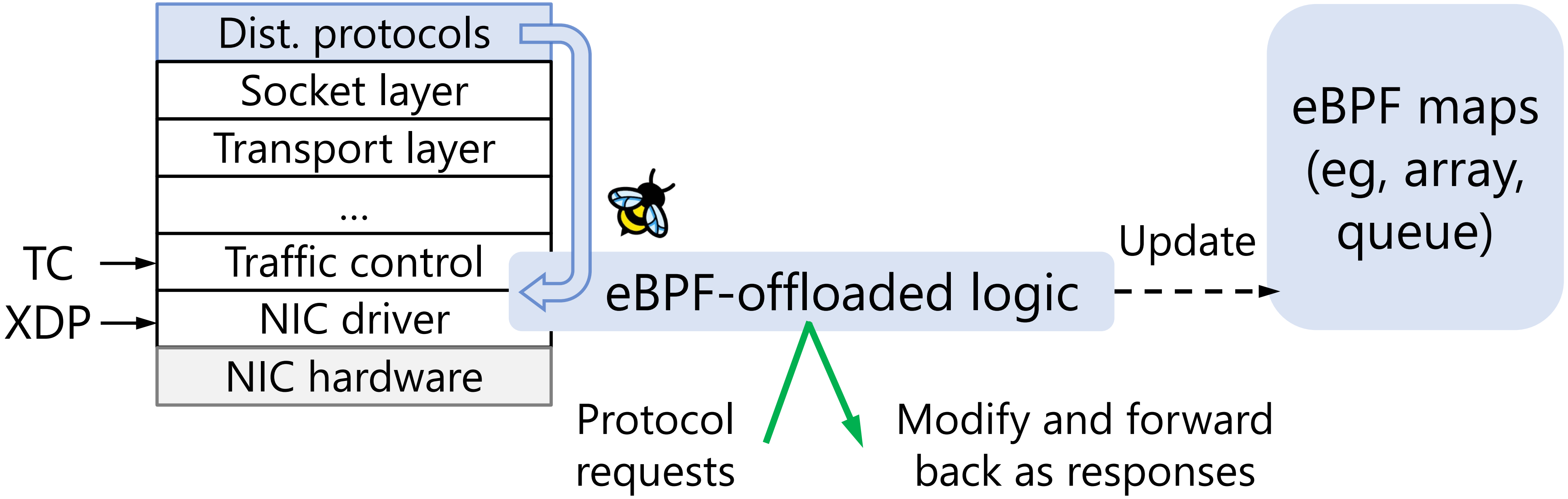
+ High performance
+ Resource sharing: interrupt-driven
+ Secure: kernel manages NICs

[1] DINT: an archaic word, meaning force and power

How to Guarantee Kernel Safety?

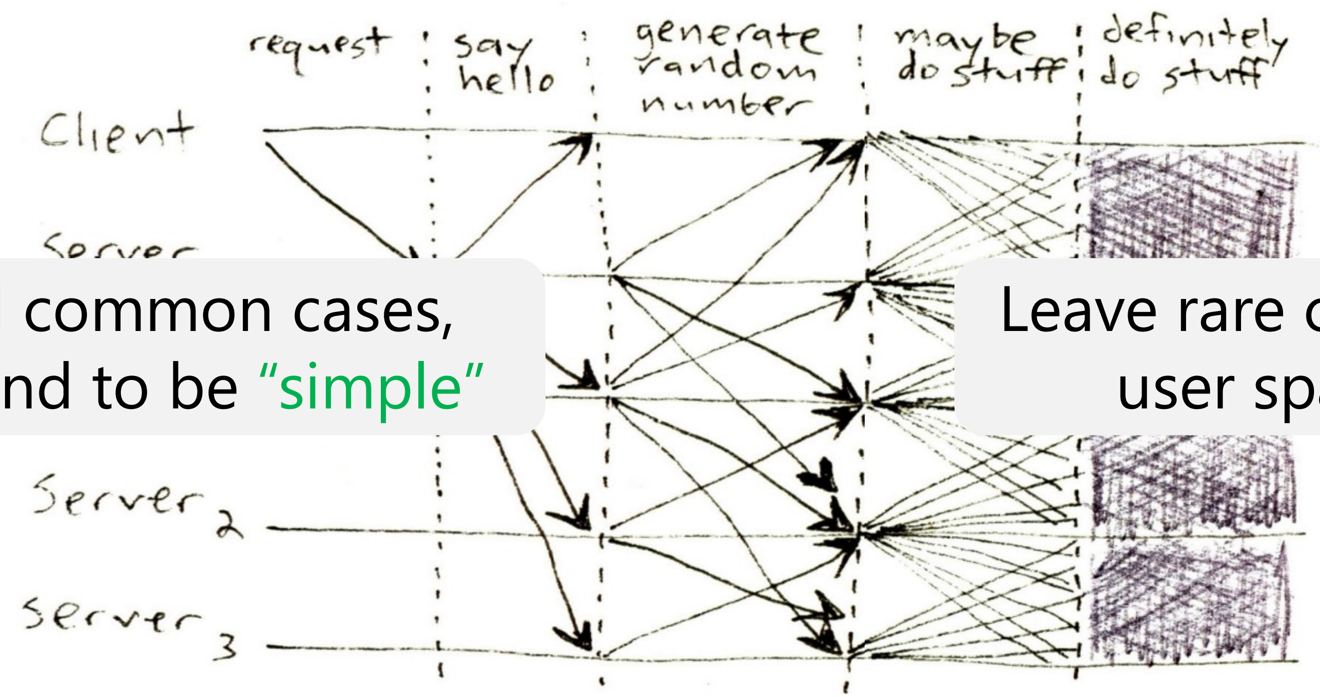
eBPF (extended Berkeley Packet Filter) to safely run programs in kernel at runtime

- Guaranteeing safety via static verification
- Originally for packet filtering and monitoring



Challenge of Kernel Offloads with eBPF

- eBPF programming model is **constrained** because of static verification
  - Limited # of instructions, bounded loops, static memory allocation
- Distributed protocols are **complex**
  - Some rare cases are too complex for eBPF: eg, failure, message loss, malloc

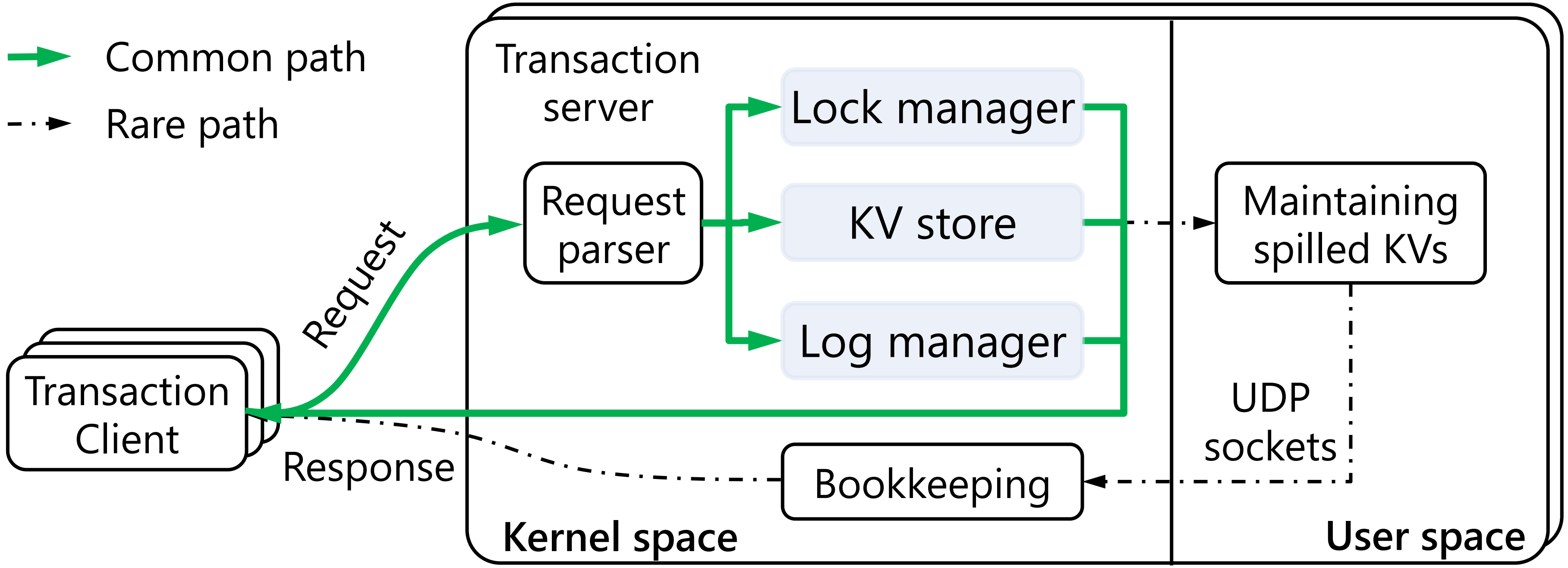


Offload common cases, which tend to be "simple"

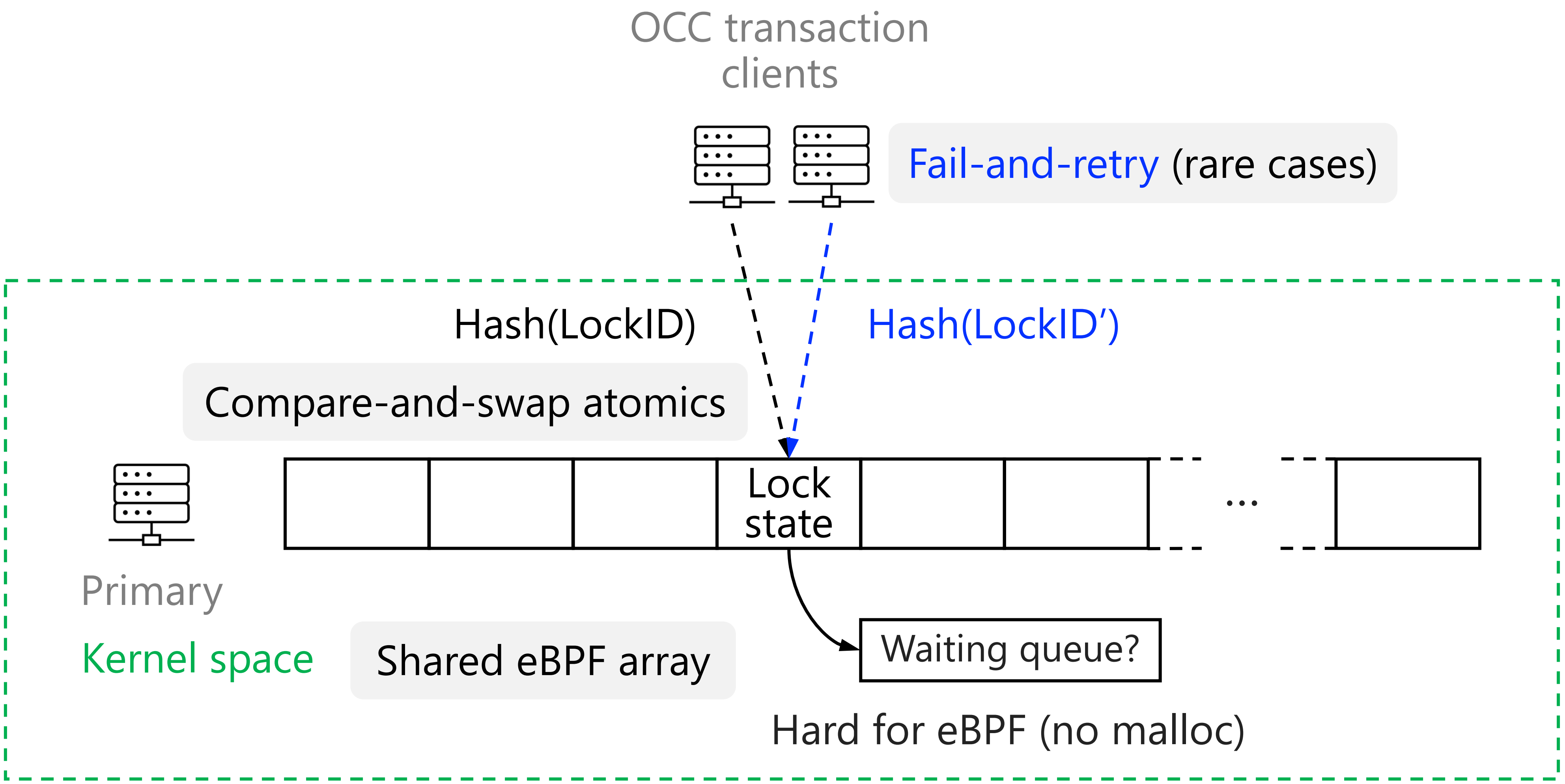
Leave rare cases to user space

Mickens, James. "The Saddest Moment." Login Usenix Mag 39.3 (2014): 52-54

DINT Overall Architecture



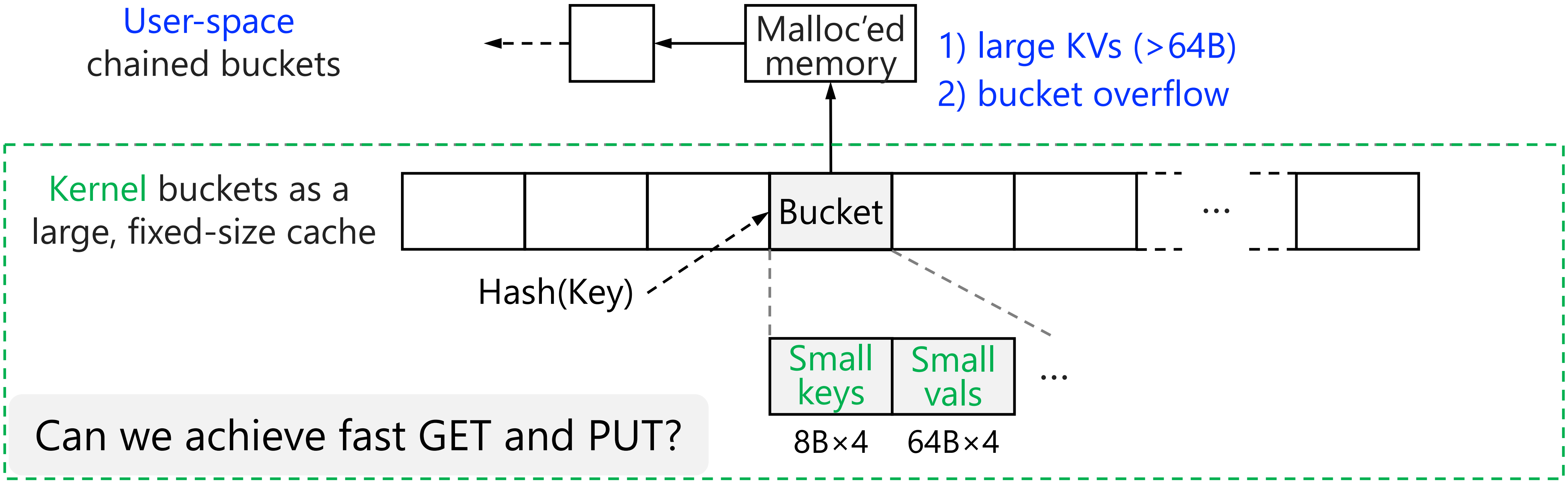
Offloading Lock Manager



Offloading KV Store

Common cases: most KVs are small in typical workloads

- Dozens of bytes in transactional workloads (eg, TATP, SmallBank)
- Statically-allocated eBPF map to store small KVs and avoid malloc

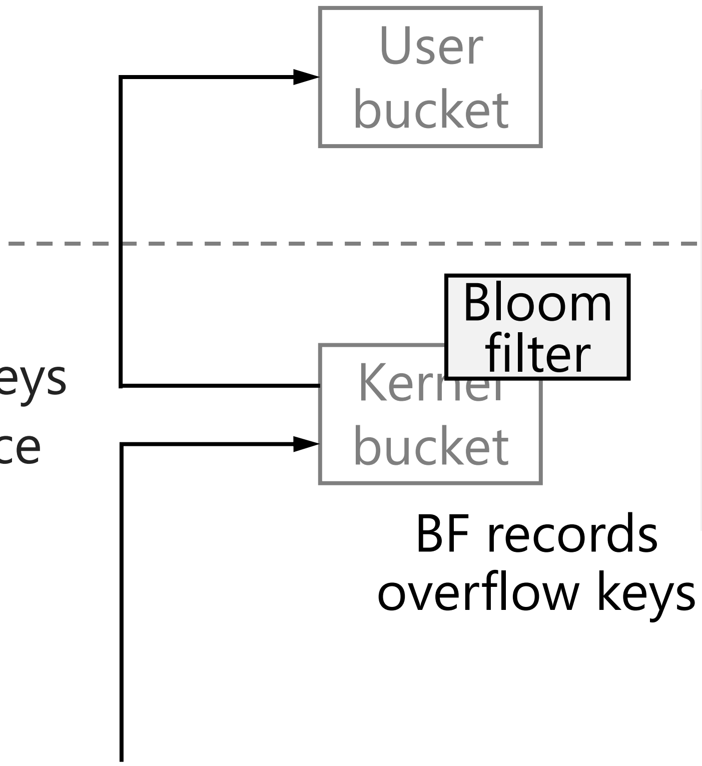


Can we achieve fast GET and PUT?

Offloading KV Store

How to achieve fast GET especially for non-existing keys?

GETs for non-existing keys always check user space



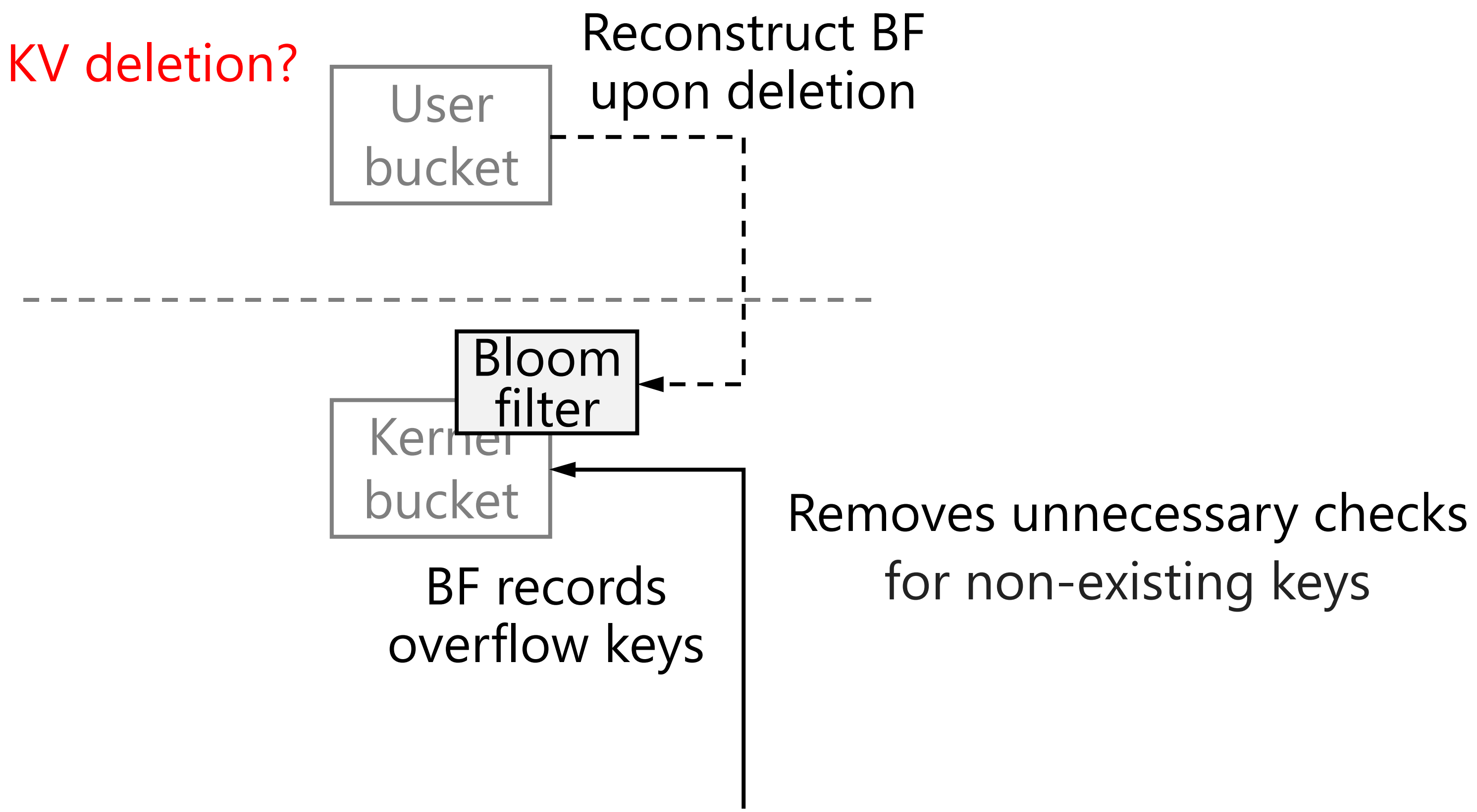
Bloom Filter[1]:

- An approximate data structure that quickly tells whether a key is in a set
- Using a fixed size of bitset array

[1] Bloom, Burton H. "Space/Time Trade-offs in Hash Coding with Allowable Errors." Communications of the ACM 13.7 (1970): 422-426

Offloading KV Store

How to achieve fast GET especially for non-existing keys?



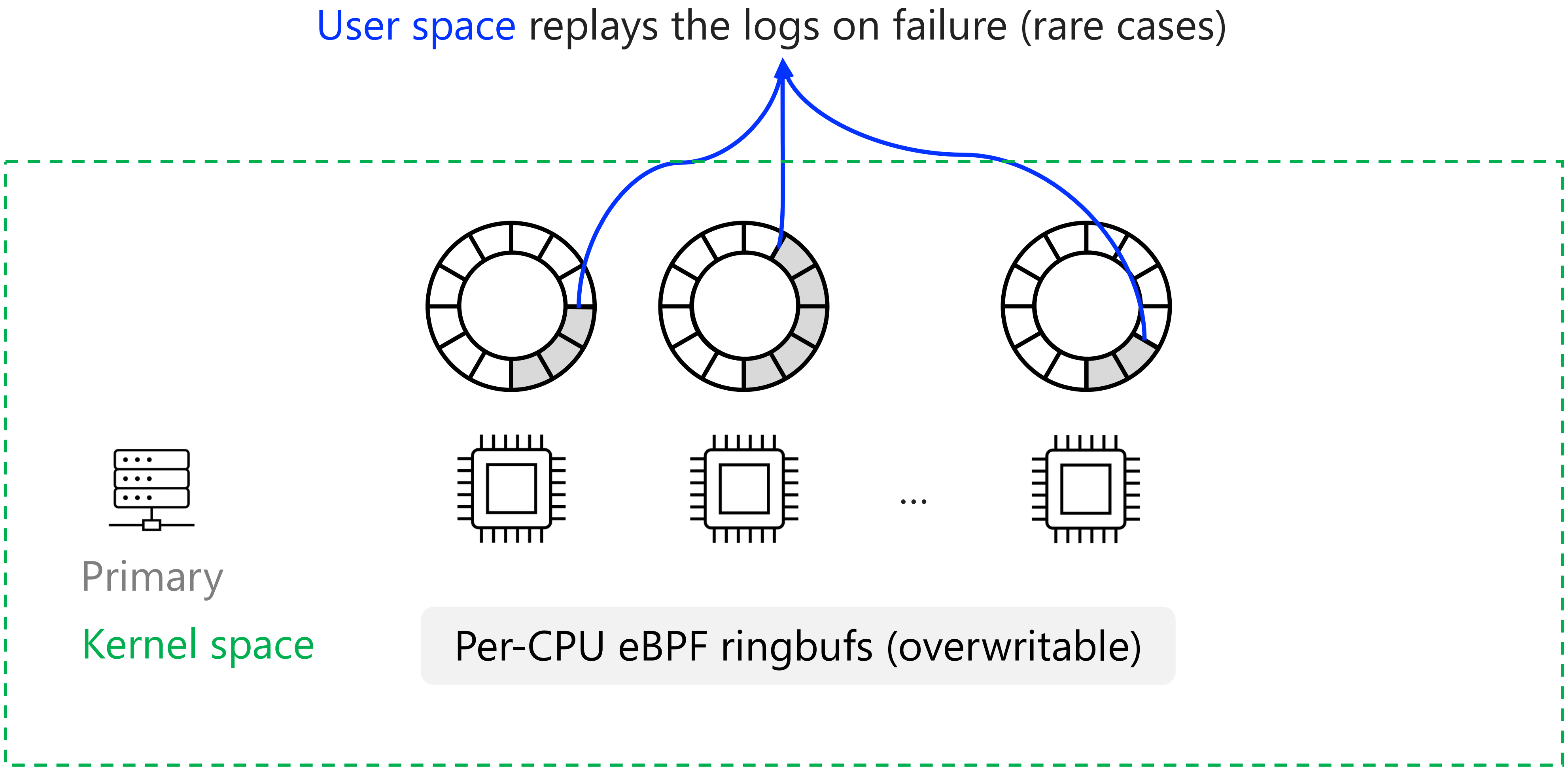
Offloading KV Store

How to achieve fast GET especially for non-existing keys?

Bloom filter to record overflow keys

- Write-back cache for fast PUT
- Lock sharing for fast locking
- Per-core circular logs for fast logging
- Piggybacking states on packets for fast user-kernel synchronization
- …

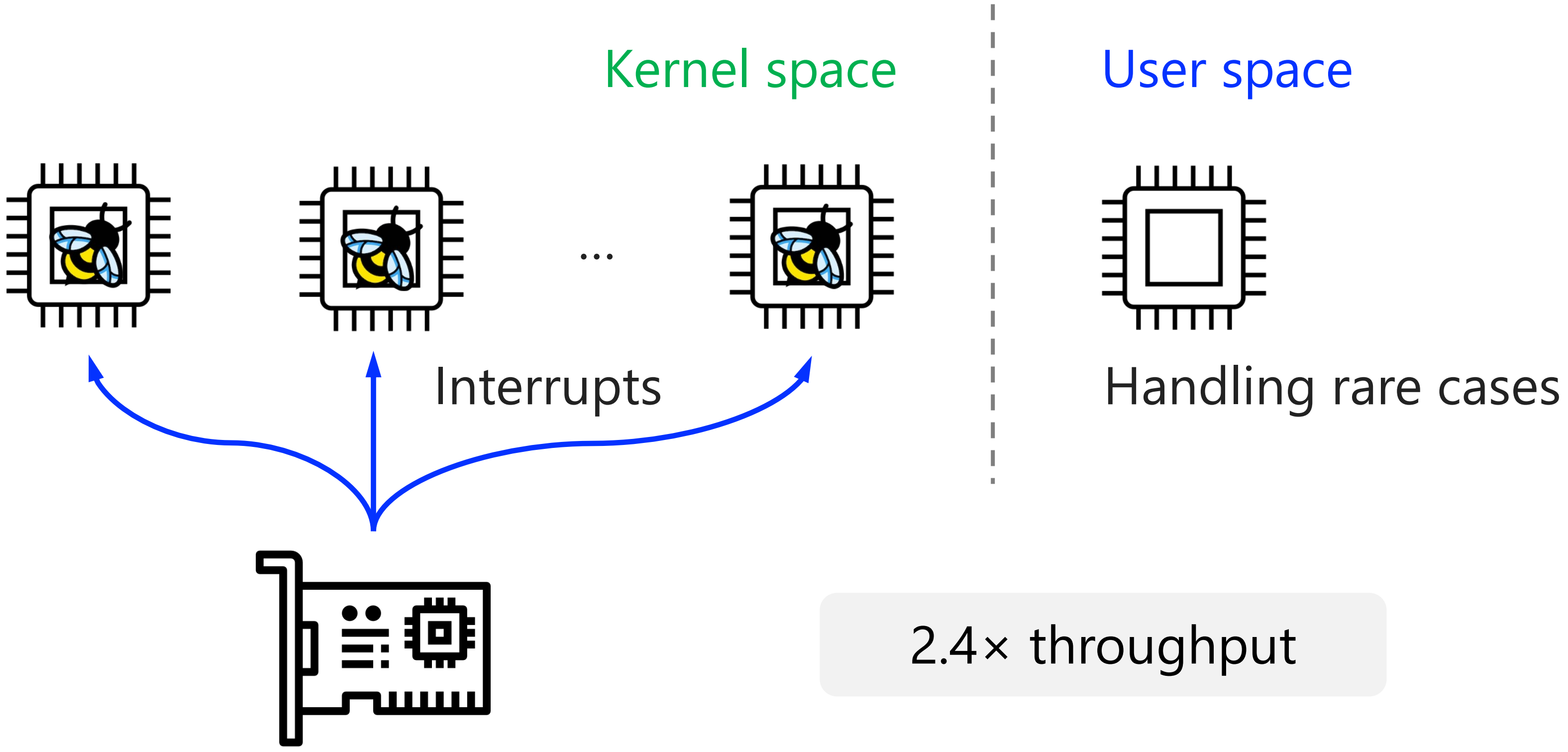
Offloading Log Manager



System Optimization: Interrupt Scheduling

Separating interrupt-handling cores and rare-case handling cores

➥ Avoiding user-kernel context switching overhead

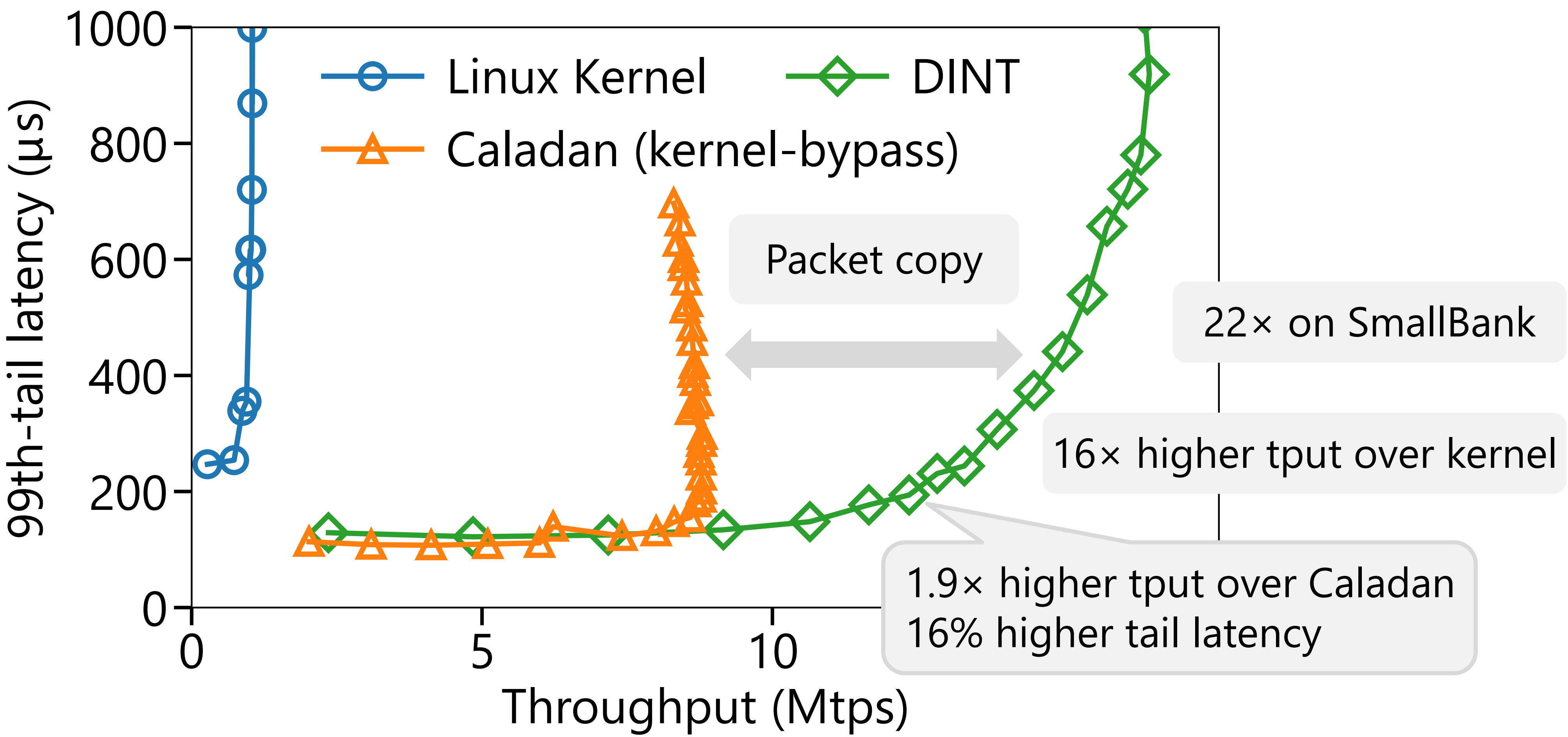


2.4× throughput

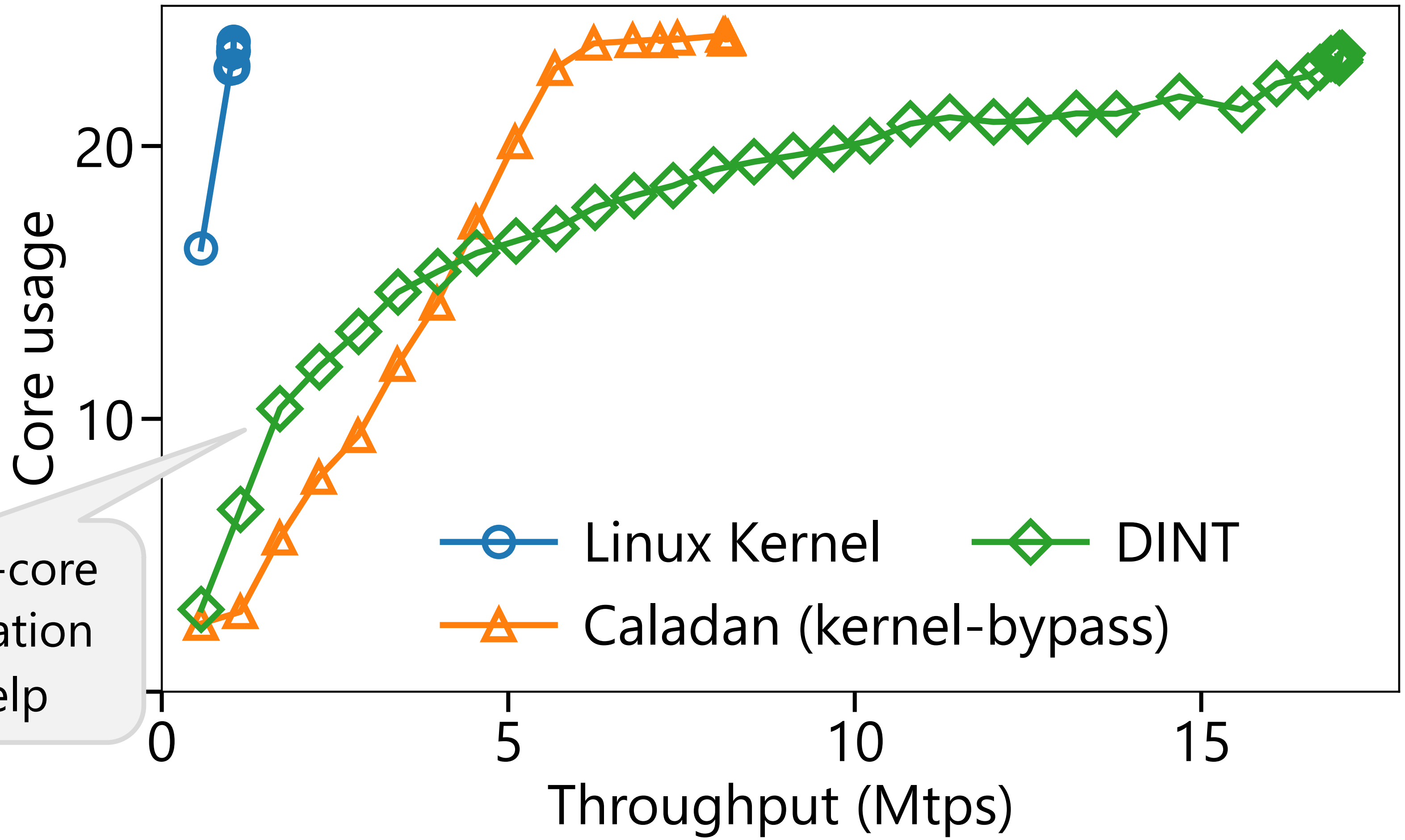
Implementation & Evaluation

- 2.1K lines eBPF and 4.3K C++ for OCC and 2PL distributed transactions
  - 3-way primary-backup replication, 3-way sharding
  - 6.1K lines of C++ for baselines
- Experiment setup:
  - CloudLab r650 (10 clients, 3 servers) running **unmodified** Linux kernel 6.1.0
  - TATP for OCC, SmallBank for 2PL
- Open source: https://github.com/DINT-NSDI24/DINT

Tail Latency vs. Throughput



CPU Utilization vs. Throughput



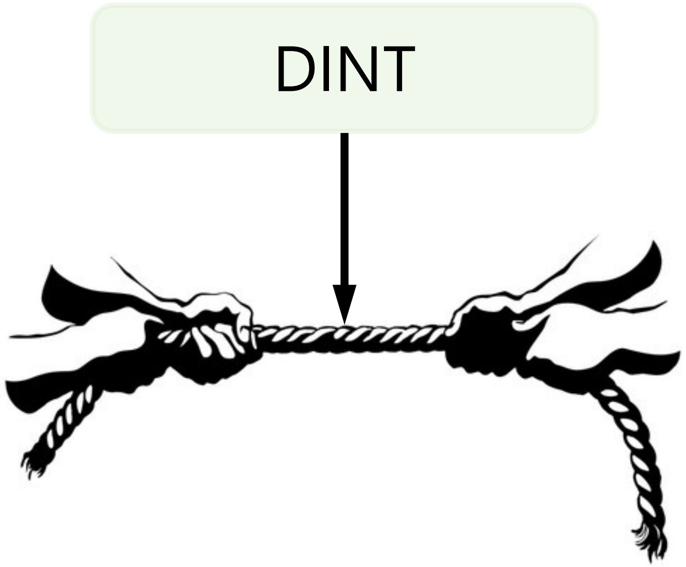
Interrupt-core consolidation may help

DINT Conclusion

We enable application-customized kernel networking stacks with:

- eBPF offloads for common cases, while user space for rare cases
- distributed transaction offloads, but generalizable to many distributed protocols

Kernel bypass:
high performance



Kernel:
resource sharing, security

Thank you!

yangzhou@g.harvard.edu