DPDK

DATA PLANE DEVELOPMENT KIT

Reclaiming Memory – Efficient and Lock Free – rte_tqs

HONNAPPA NAGARAHALLI

ARM

Acknowledgements

Malvika

Gupta

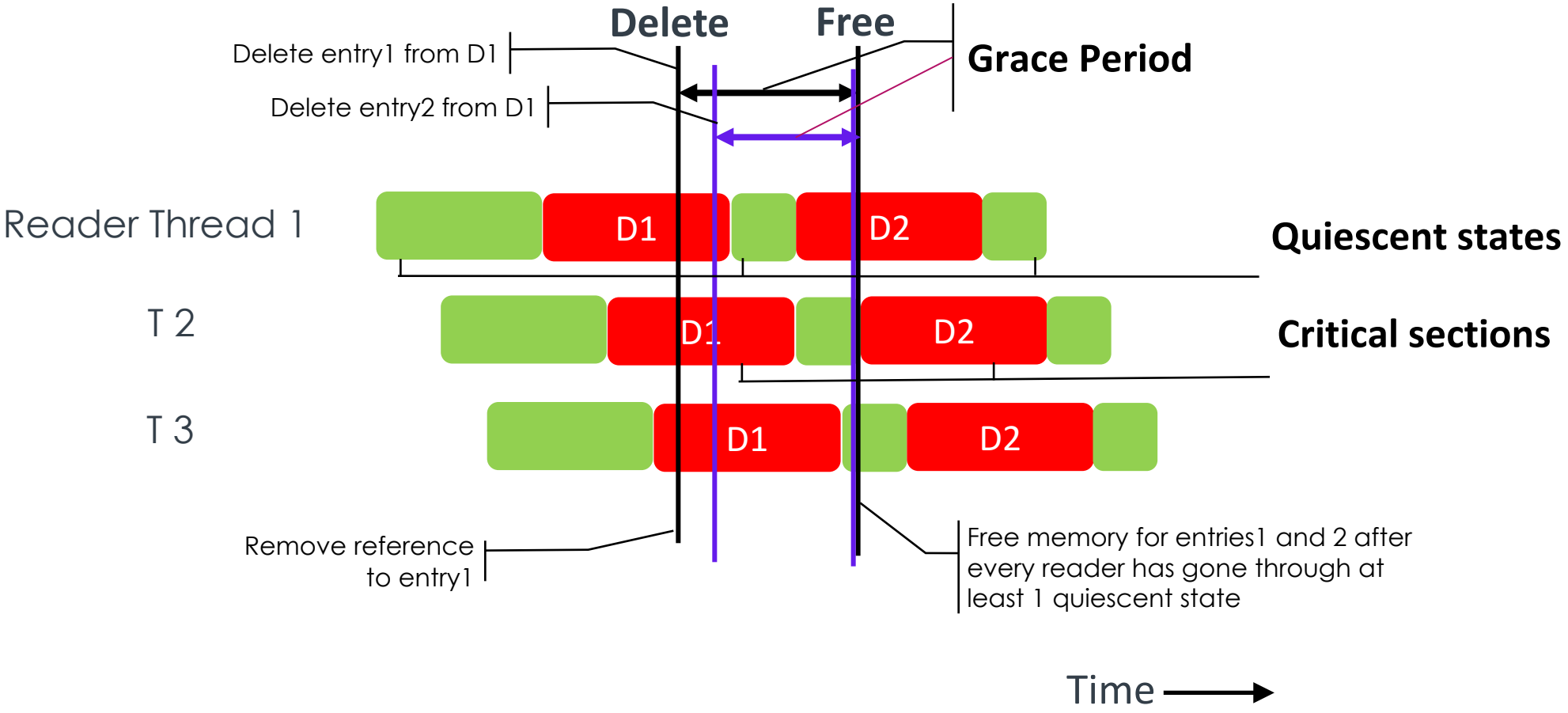
Dharmik

Thakkar

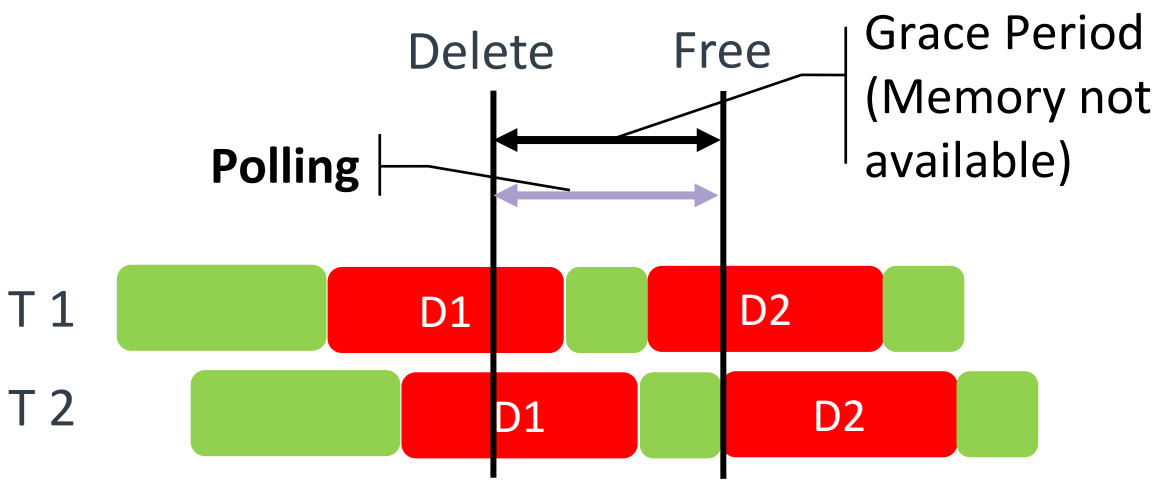
Agenda

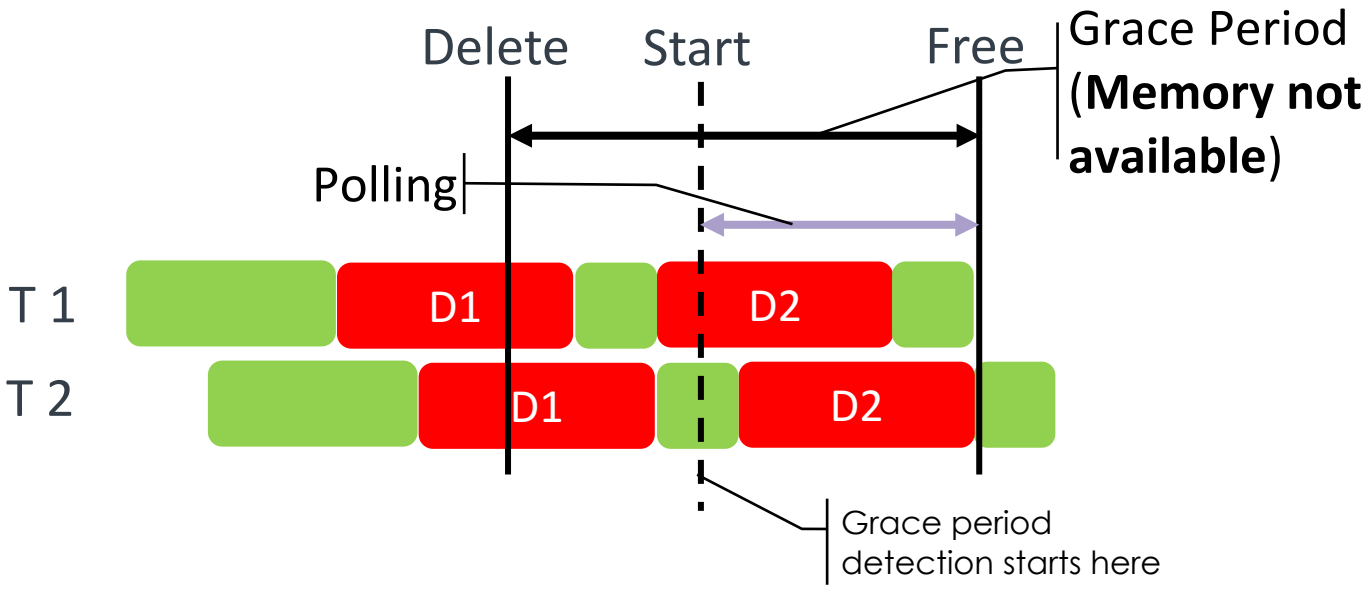
- Define Terms/Parameters
- Effect of these Parameters
- Requirements – from DPDK perspective
- Design

Terms/Parameters



Grace Period

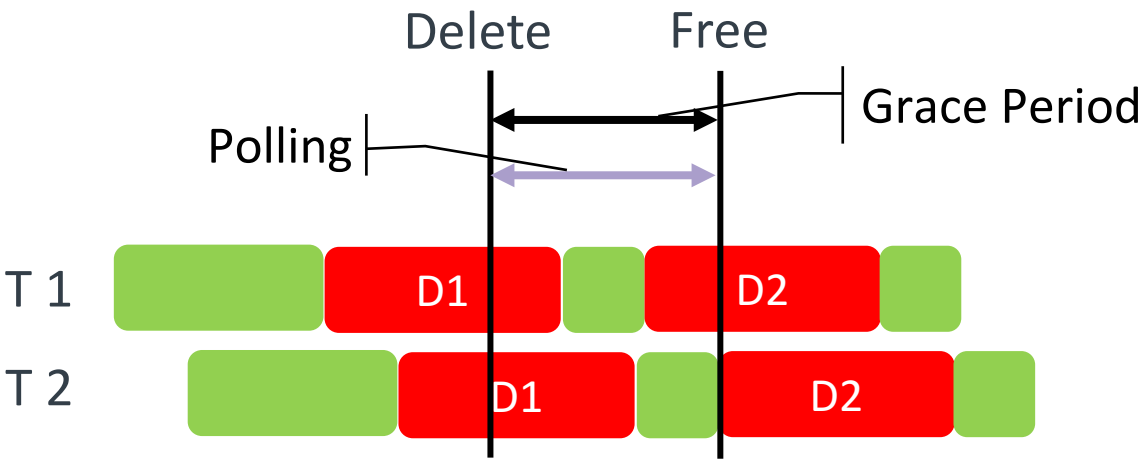


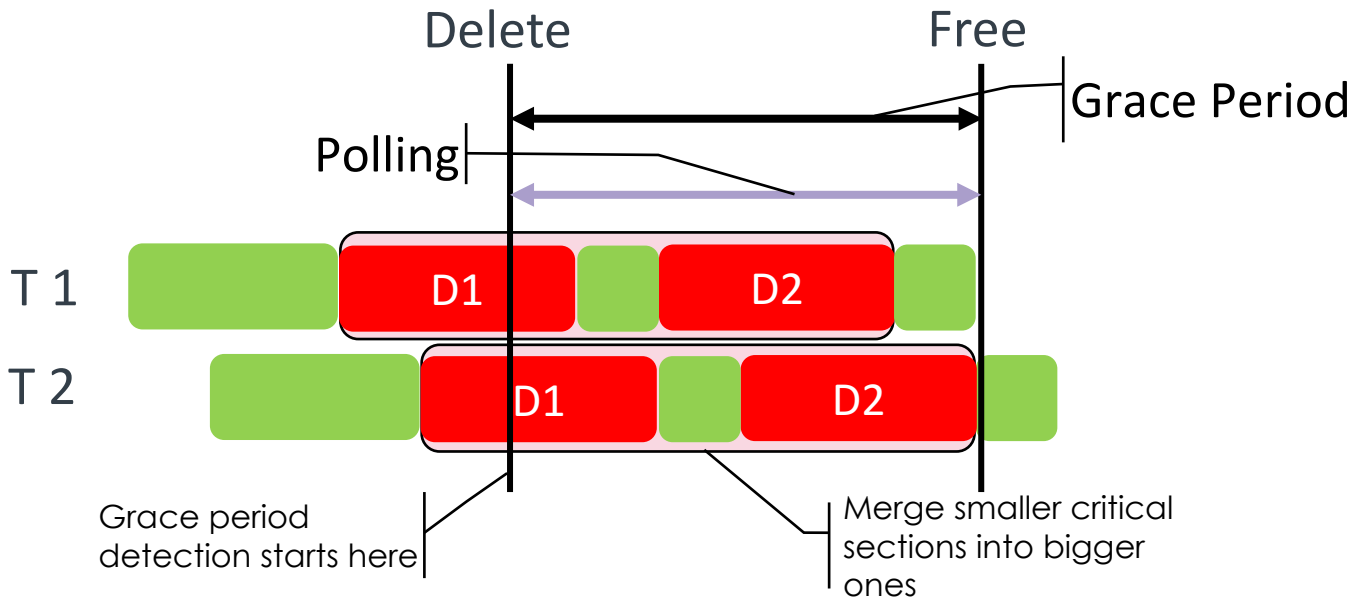


Continuous Polling => Increased memory access

Longer Grace Period => Extra additional memory

Critical Section – Smaller vs Bigger





Advantage – Smaller grace period

Disadvantage – More cycles required for reporting quiescent state.

Advantage – Less cycles for reporting quiescent state

Disadvantage – Larger grace period and more polling

Requirements

- Need characteristics of LARGE critical section and SMALL grace period
  - Reduce number of cycles required for grace period detection
  - Reduce polling during grace period
  - Distributed/Concurrent grace period detection - needs lock free algorithm
- Don’t enforce a programming model
- rte_event_dequeue_burst supports blocking mode – efficient ‘bother about me’, ‘don’t both about me’ APIs

Design

- The list of readers is represented as a bit-map
  - Efficient
  - Allows for lock-free/concurrent operations
- Quiescent state detection is split into 2 steps
  - Allows for multiple/concurrent quiescent state queries
  - Start (rte_tqs_start)
    - Lock-free – allows multiple writers to call this concurrently
    - Allows the writers to start the process without having to wait for end of grace period
    - Allows the writers to do other work while the grace period is underway – no cycles wasted
    - Continuous polling not required – reduces memory accesses

Design

- Quiescent state detection is split into 2 steps
  - Check (rte_tqs_check)
    - Lock-free – allows for multiple writers to call this concurrently
    - If the writers do enough other work, mostly a success is returned in the first polling attempt
    - Does not enforce threading model for batching
- rte_tqs_register/rte_tqs_unregister are lock-free – allows multiple worker threads to announce their participation concurrently

DPDK
DATA PLANE DEVELOPMENT KIT

Questions?

Backup Slides

Why not liburcu?

- The list of readers is a linked list protected by a lock
  - Does not allow concurrent and lock-free insertion/deletion/traversal
  - Not very efficient to call on data plane threads
- Issues in Quiescent State detection API
  - synchronize_rcu (equivalent of rte_tqs_start + rte_tqs_check) is a blocking API, does not return till the grace period is over
  - Since it is blocking, it cannot be called on the data plane threads
  - Polling the reader threads, while waiting for grace period to end, wastes CPU cycles
  - Polling causes additional memory accesses
  - It uses a lock which does not allow multiple synchronize_rcu calls to run concurrently

Why not liburcu?

- The list of readers is a linked list protected by a lock
  - Does not allow concurrent and lock-free insertion/deletion/traversal
  - Not very efficient to call on data plane threads
- Issues in Quiescent State detection API
  - synchronize_rcu (equivalent of rte_tqs_start + rte_tqs_check) is a blocking API, does not return till the grace period is over
  - Since it is blocking, it cannot be called on the data plane threads
  - Polling the reader threads, while waiting for grace period to end, wastes CPU cycles
  - Polling causes additional memory accesses
  - It uses a lock which does not allow multiple synchronize_rcu calls to run concurrently

Parts of DPDK present in other projects

- Atomic operations APIs
  - https://github.com/ivmai/libatomic_ops - Supports much wider set of architectures
  - https://github.com/urcu/userspace-rcu/blob/master/doc/uatomic-api.md
- Any reasoning for
  - eal_common_string_fns.c and rte_string_fns.h
- Bunch of data structures that exist in DPDK provided here
  - https://github.com/urcu/userspace-rcu/blob/master/doc/cds-api.md