Highly-Efficient Persistent Data Structures The performance principles that govern their design*

PANAGIOTA FATOUROU

Foundation for Research and Technology – Hellas, Institute of Computer Science

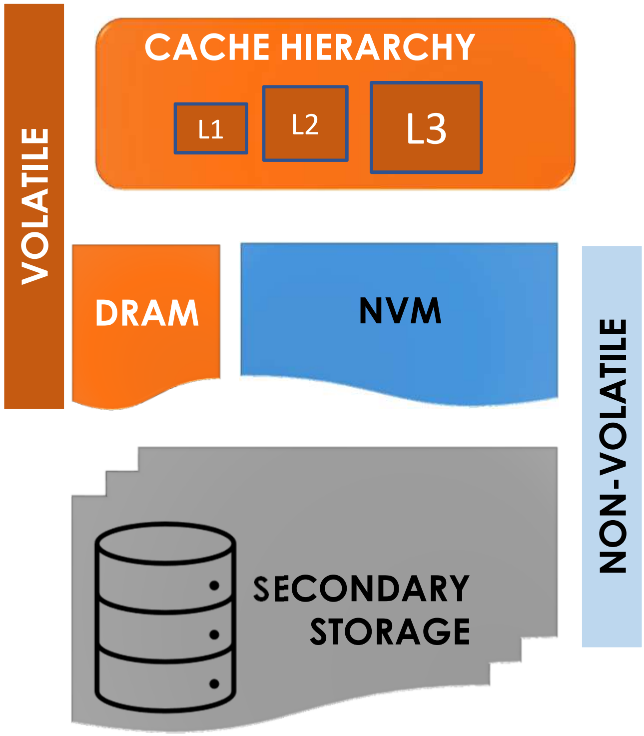
University of Crete, Department of Computer Science, Greece

EMERALD 2024

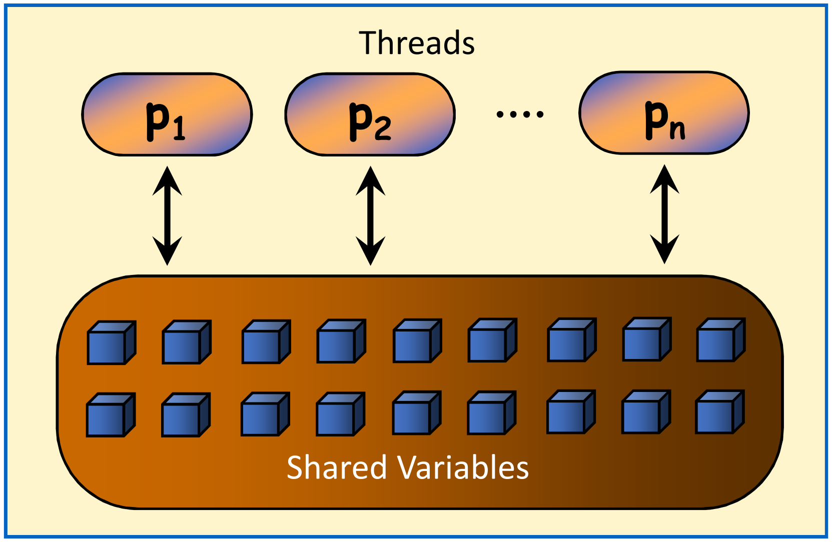
* Supported by the Hellenic Foundation for Research and Innovation (HFRI) under the "Second Call for HFRI Research Projects to support Faculty Members and Researchers" (project number: 3684)

Persistent Computing

- ❖ **Non-Volatile Memory (NVM)**
  - **byte-addressable, faster than secondary storage**
  - **large** and **less expensive than DRAM**
  - Recovery in case of failures



Persistent Computing



➢ Some of the shared variables may be stored in volatile memory, whereas others in NVM.

Persistent Instructions

➢ **Flush (pwb):** write back a cache line in NVM (async)

➢ **Psynch:** block until preceding flushes have been realized.

System-wide failures

Durable Linearizability [Izraelevitz, Mendes and Scott. 2016]

Detectability [Friedman et al., 2018]

Challenge

HOW TO DESIGN

PERSISTENT DATA STRUCTURES

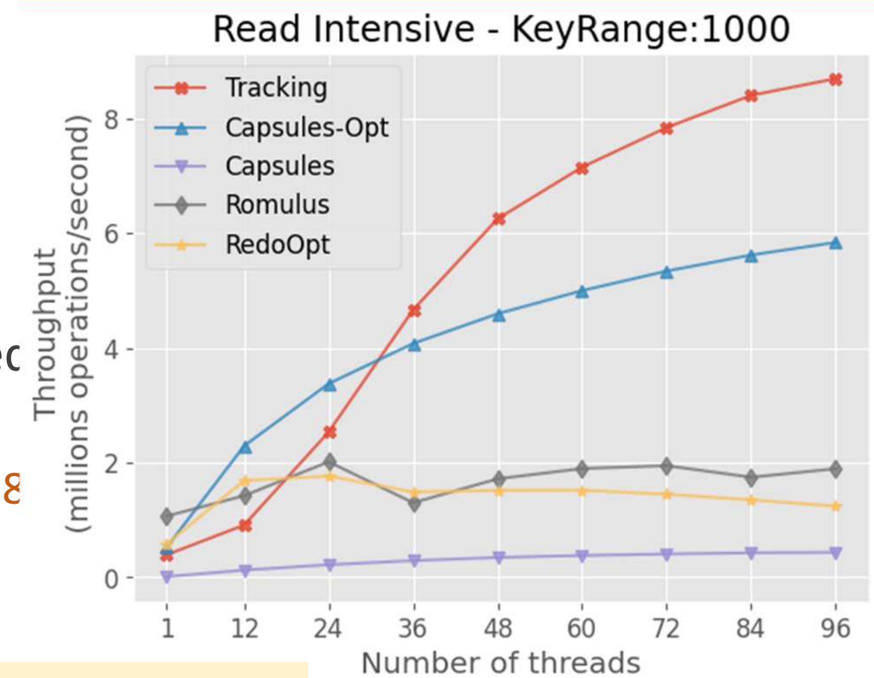
WITH LOW PERSISTENCE COST?

Are there persistence principles that we should take into consideration when designing such data structures?

Insights on Evaluating Persistent Algorithms

[Attiya, Ben-Baruch, Fatourou, Hendler, Kosmas, PPoPP 2022]

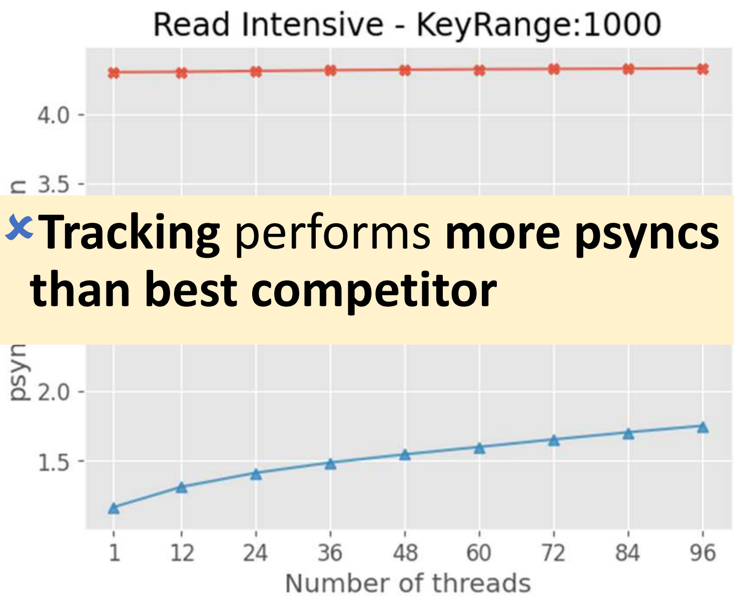
- **Tracking** Linked List (no hand-tunned)
- **Capsules-Opt:** strongly hand-tuned transformation of Harris' linked list using Capsules [Attiya et al., PPoPP 2022]
- Capsules: general scheme (not hand-tunned) [Ben-David, Blelloch, Wei. 2018]
- **Romulus** [Correia, Felber, Ramahlete, SPAA'18]
- **RedoOpt** [Correiaet al., Eurosys 2020]



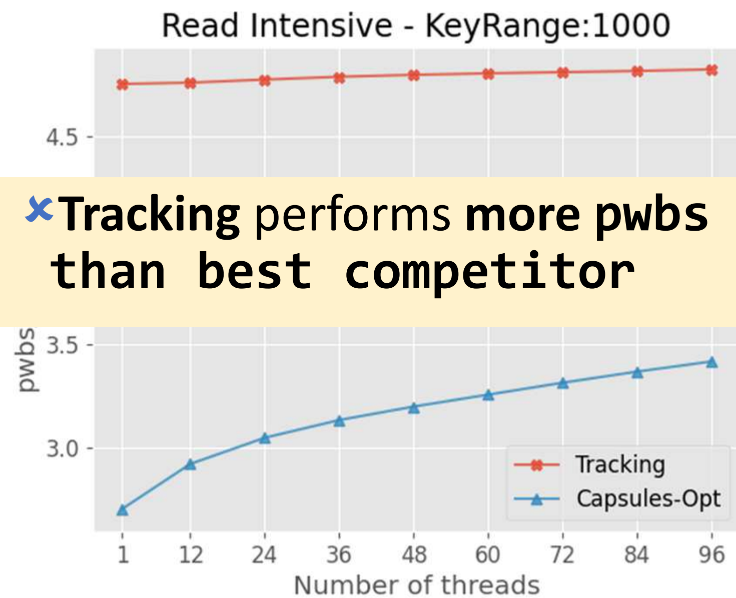
Tracking (red line) exhibits better performance than competitors as the number of threads increases.

Insights on Evaluating Persistent Algorithms

[Attiya, Ben-Baruch, Fatourou, Hendler, Kosmas, PPoPP 2022]



- **Tracking** performs **more psyncs than best competitor**



- **Tracking** performs **more pwbs than best competitor**

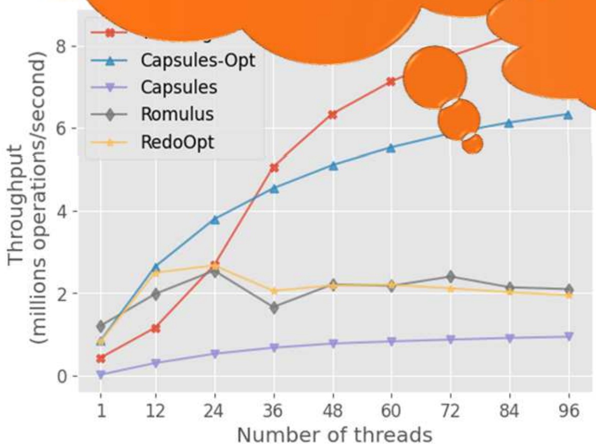
The synchronization cost of Tracking is also higher than that of best competitor.

Insights on Evaluating Persistent Algorithms

[Attiya, Ben-Baruch, Fatourou, Hendler, Kosmas, PPoPP 2022]

What causes the good performance of Tracking?

what about the impact of each single persistence instruction?



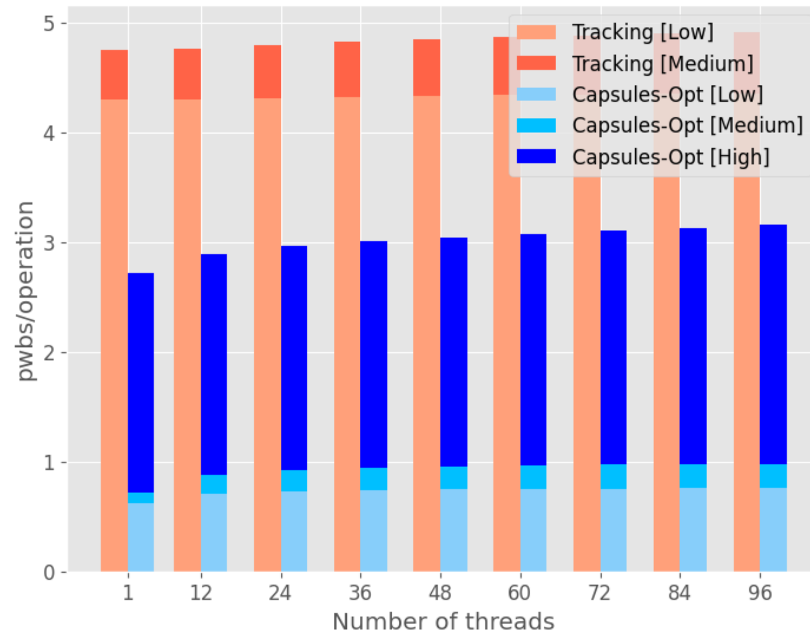
- The impact of psyncs is negligible!

- **Methodology** for measuring the overhead of **each pwb**

Insights on Evaluating Persistent Algorithms

[Attiya, Ben-Baruch, Fatourou, Hendler, Kosmas, PPoPP 2022]

- **Categorization**
  - **Low**, **Medium**, and **High** impact code lines with persistence instructions



- A flush that incurs high performance penalty
  - is executed on a highly-contended variable
  - precedes or follows CASes

Persistence Principles Crucial for Performance

[Fatourou, Kallimanis & Kosmas, PPoPP'22, BEST PAPER AWARD]

1. **The number of the persistence instructions should be kept as low as possible**
   - Store in NVM only those variables (and persist only those from their values) that are absolutely necessary for recoverability

   [Vast majority of work aimed at achieving this]

2. **The persistence instructions should be of low cost (e.g., by persisting less highly-contented shared variables)** [Tracking]PPoPP'22
   - Avoid pwbs on variables on which CAS is performed before or after [Tracking]PPoPP'22
   - Reduce accesses to recently flushed cache lines [Sela & Petrank]SPAA'21 , [MIRROR]PLDI'21

3. **Data to be persisted should be placed in consecutive memory addresses**
   - pwb and psynch operate on the granularity of a cache line [PBcomb, PWFcomb]PPOPP'22, [ArchTM]FAST'21

Principles Crucial for Performance

STUDY WHETHER PERSISTENCE CAN BE EFFICIENTLY SUPPORTED ON TOP OF STATE-OF-THE-ART ALGORITHMS DESIGNED FOR THE CONVENTIONAL SETTING.

Persistent Software Combining
[Fatourou, Kallimanis & Kosmas, PPoPP 2022, BEST PAPER AWARD]

Efficient persistent blocking and wait-free

- **synchronization protocols and universal constructions (UC)**
  - **outperform** previously proposed recoverable UCs

    [RedoOpt]$_{EuroSys'20}$ and STMs [CX-PTM]$_{EuroSys'20}$, [OneFile]$_{DSN'19}$

Designed based on state-of-the-art synch techniques & UCs presented in PPoPP'12 & SPAA'11.

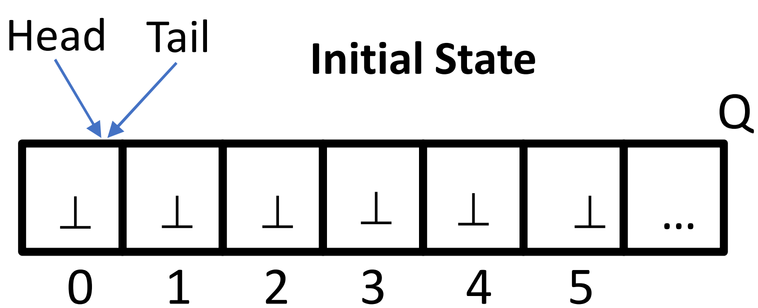
- **stacks**, **queues** and **heaps**
  - **outperform** previous implementations (including specialized)
    - queues [Sela & Petrank: OptLinkedQ, OptUnLinkedQ]$_{SPAA'21}$, [CX-PUC, CX-PTM, RedoOpt]$_{EuroSys'20}$, [OneFile]$_{DSN'19}$, [Capsules]$_{SPPA'19}$, [Friedman et al]$_{PPoPP'18}$, [Romulus]$_{SPAA'18}$
    - stacks [DFC]$_{arXiv'20}$, [OneFile]$_{DSN'19}$, [RomulusLog]$_{SPAA'18}$

Persistent FIFO Queues. Can we do better?
[Giachoudis, Fatourou, Mallis, SIROCCO 2024]

- PerLCRQ, a persistent implementation of FIFO queue that significantly outperforms all other persistent FIFO queue implementations.
  - It adds persistence on top of LCRQ (**Afek and Morrison, PPoPP 2013**), illustrating how to efficiently persist algorithms that use Fetch&Add.
- PerLCRQ introduces techniques for reducing the persistence cost that could be of general interest.
- Framework to simulate failures and measure the recovery cost of algs.

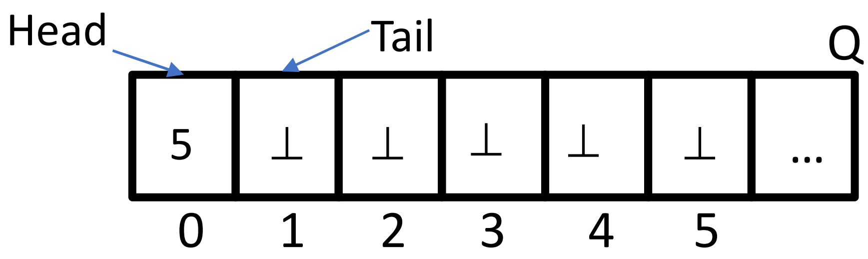
LCRQ Basic Idea: IQ algorithm [AM, PPoPP 2013]

- Implement Queue as an infinite table, Q.
- **Head** points to first-inserted element in Q
- **Tail** points to last-inserted element in Q



- Head and Tail: Fetch&Increment (FAI) objects that are incremented indefinitely.
- Enqueue (Dequeue) gets an index j by performing FAI on Tail (Head).
- Enqueue with index j inserts its item in Q[j mod R]
- A Dequeue with index j can exhaust only the item inserted by the Enqueue with index j.
- The use of Fetch&Increment ensures that every position of the array is accessed by just two processes.

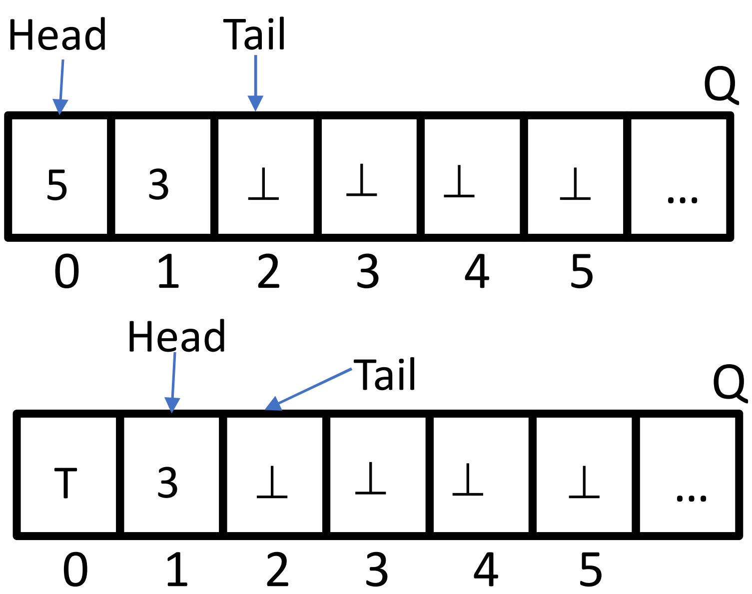
LCRQ Basic Idea: IQ algorithm [AM, PPoPP 2013]



Enqueue

1. Perform FAI on Tail to get next available position (pos) of Q
2. Perform Get&Set on Q[pos] to store the new value there
3. If result is ⊥, return
4. Otherwise, repeat steps above

LCRQ Basic Idea: IQ algorithm [AM, PPoPP 2013]



Dequeue

1. Perform FAI on Head to get next position of Q to dequeue from
2. Perform Get&Set on Q[pos] to store T there
3. If result is a value other than ⊥, return result
4. If Head > Tail, return EMPTY
5. Otherwise, repeat above steps

Persistent IQ

Enqueue

1. Perform FAI on Tail to get next available position (pos) of Q
2. Perform Get&Set on Q[pos] to store the new value there
3. If result is ⊥
   PERSIST Q[pos]
   return
4. Otherwise, repeat steps above

5	3	⊥	⊥	⊥	⊥	…
1	2	3	4	5	6	

Dequeue

1. Perform FAI on Head to get next position of Q to dequeue from

2. Perform Get&Set on Q[pos] to store T there

3. If result is not equal to ⊥
   PERSIST Q[pos]
   return result

1. If Head > Tail
   PERSIST Q[pos]
   return EMPTY

1. Otherwise, repeat above steps

T	3	⊥	⊥	⊥	⊥	…

Persistent IQ: Low Cost of Persistence

Persistence Principles crucial for Performance

- Low number of persistence instructions
- Avoid persisting highly-contended shared variables
- Persist **consecutive** data

Persistence Properties of IQ

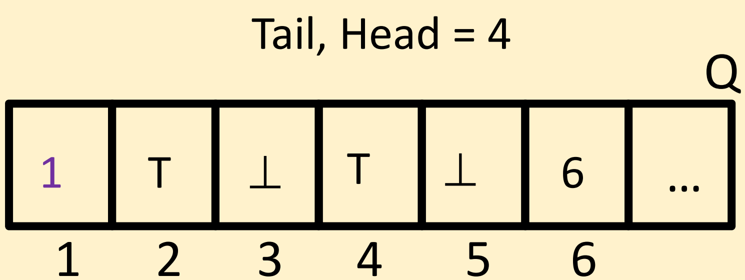
A single pair of a pwb and psync per operation

Persistence instructions are executed on shared vars on which at most two threads contend.

Data are stored in consecutive memory (array Q).

Persistent IQ: Low Cost of Persistence

- Avoiding persisting Head and Tail:
  1. adds complexity to the Recovery function,



  2. makes it hard to assign linearization points,
  3. results in long argumentation to prove that PerIQ is correct.

BUT

- Experiments show that it is much more efficient.

Persistent IQ: Recovery

Recovering Tail

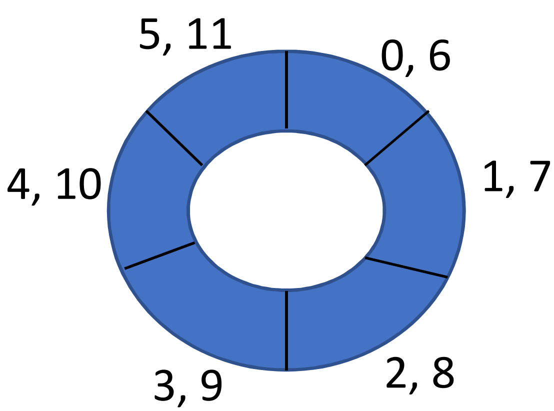
- Search Q for the first continuous streak of n unoccupied cells.
- Set Tail to be the index of the first cell of this streak.
  - When a crash occurs, some of the active enqueues have written back their value, others not.
  - All missing items are by active enqueues.
  - These are at most n.

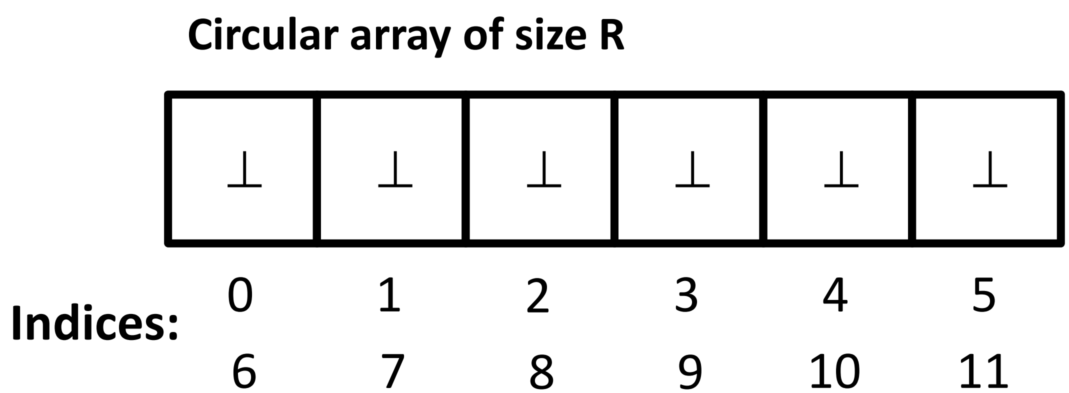
Recovering Head

- Starting from recovered Tail, traverse Q towards its beginning until meeting the first cell containing T.
- Let Head be the index of the next to this cell.
  - No element has the value T between Head and Tail.

Circular Queue (CRQ) [AM 2013] – Challenges

- There are many cases that require sync between enqueuers and dequeuers.
  - **Empty Transition:** dequeue with index j finds Q[j mod R] unoccupied.
  - **Unsafe Transition:** dequeue with index j arrives while Q[j mod R] is occupied by item of index lower than j.





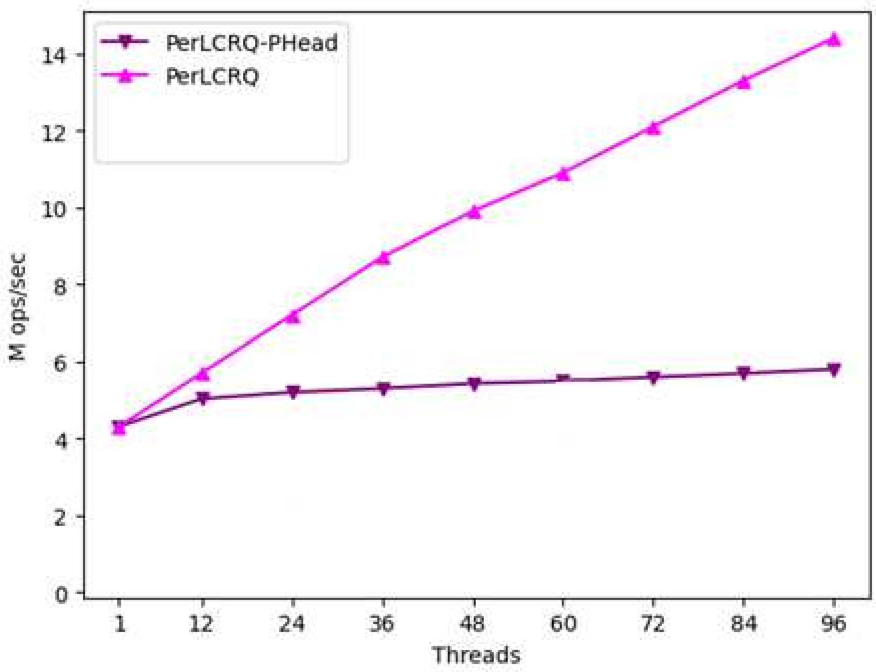
Persistent CRQ

☺ PerCRQ achieves to perform just one pair of pwb-psync instructions per operation

☹ In PerCRQ, the value of Head needs to be persisted once per successful dequeue operation.

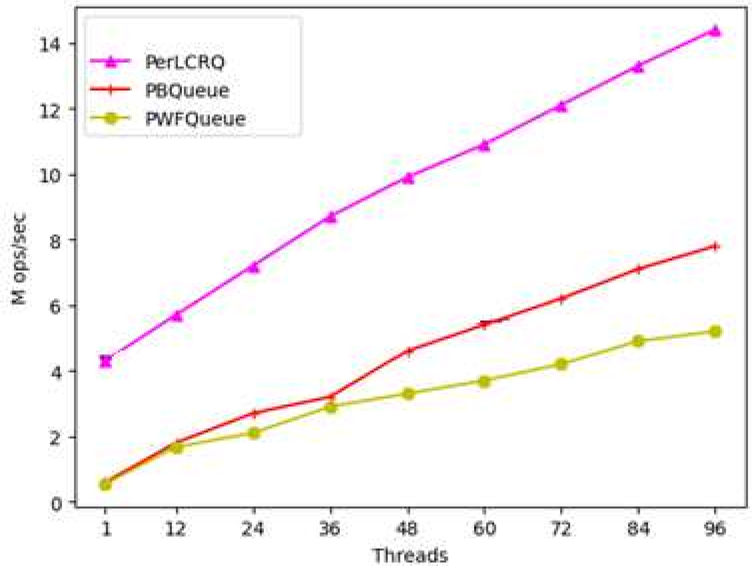
Technique for Reducing cost of persisting Head

- Each thread maintains a local copy of Head, which it updates every time it updates Head.
- When necessary, threads persist their local copy of Head instead of Head.
- At recovery, they read the persisted values of local copies of Head and decide what the recovered value of Head should be.



Evaluation

- Each experiment simulates 10^7 atomic operations (enqueues and dequeues) in total.
- Each of the n threads simulates $10^7/n$ operations.
- We measure millions of operations executed per second.
- PBQueue and PWFQueue are the previous state-of-the-art persistent FIFO queues, outperforming by far previous such implems.



Machine with 2 Intel(R) Xeon(R) 5318 processors with 24 cores each (96 logical cores total), equipped with 128 GB Intel Optane 200 Series Persistent Memory.

Summary – Methodology for designing well-performed persistent data structures & algorithms

1. **Start by experimenting with the system/hardware to understand its performance properties and come up with principles crucial for performance.**

2. **Start with the STATE-OF-THE-ART concurrent implementation of the data structure you would like to implement using the new hardware.**

3. **Focusing on NVM, try to respect the persistence principles at all stages of the design of the persistent version**
   - Maintain the number of persistence instructions as low as possible.
   - Persistence instructions of low cost.
   - Data to be persisted should be placed in consecutive memory addresses.

4. **Come up with new techniques, if necessary, to fully respect all principles that are crucial for performance.**

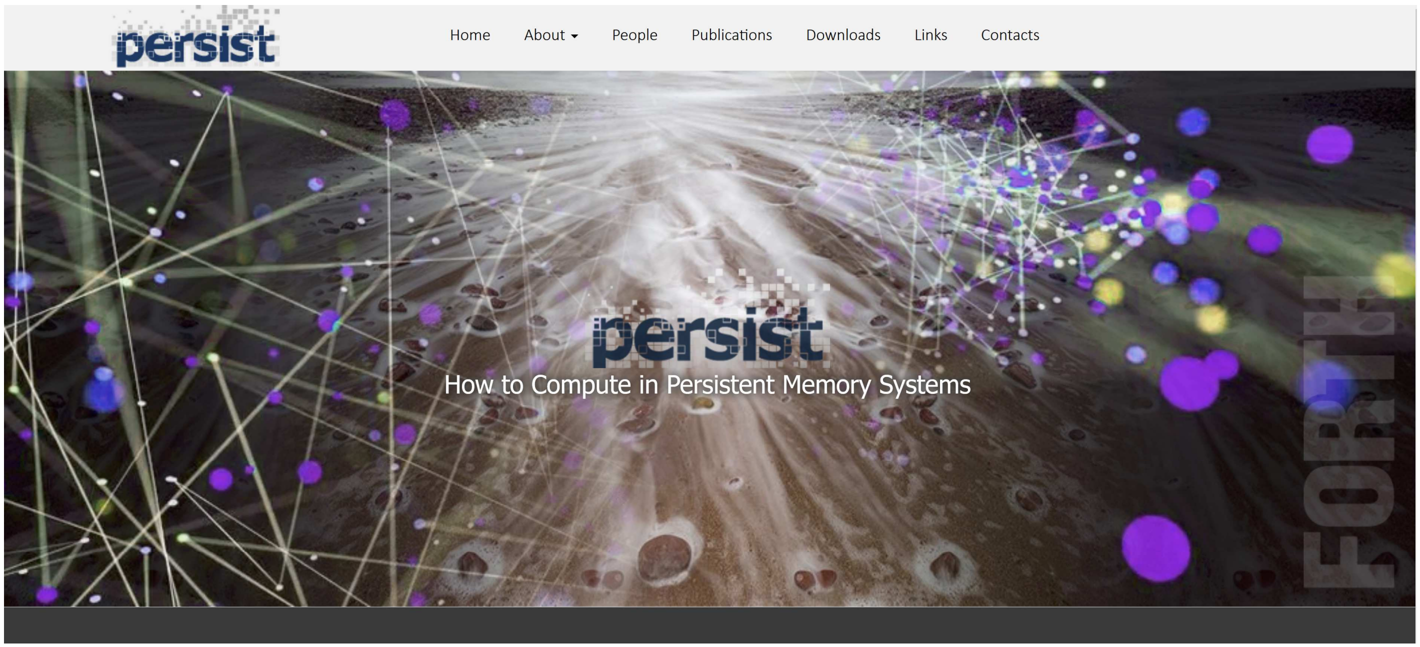
Concerns, Questions and Open Problems

- Future of NVM computing?
- Will these results be relevant if new NVM technology is provided in the future?

Principles & Techniques

- Some of them may no longer be needed.
- Most of them are quite fundamental.
  - Avoid performing operations on highly-contended variables.
  - Perform as less instructions as possible.
  - Study adaptations of state-of-the-art algorithms first.

Thank you!



https://persist-project.gr/

faturu@csd.uoc.gr

FORTH-ICS

H.F.R.I.
Hellenic Foundation for Research & Innovation