Recoverable Computing

PANAGIOTA FATOUROU

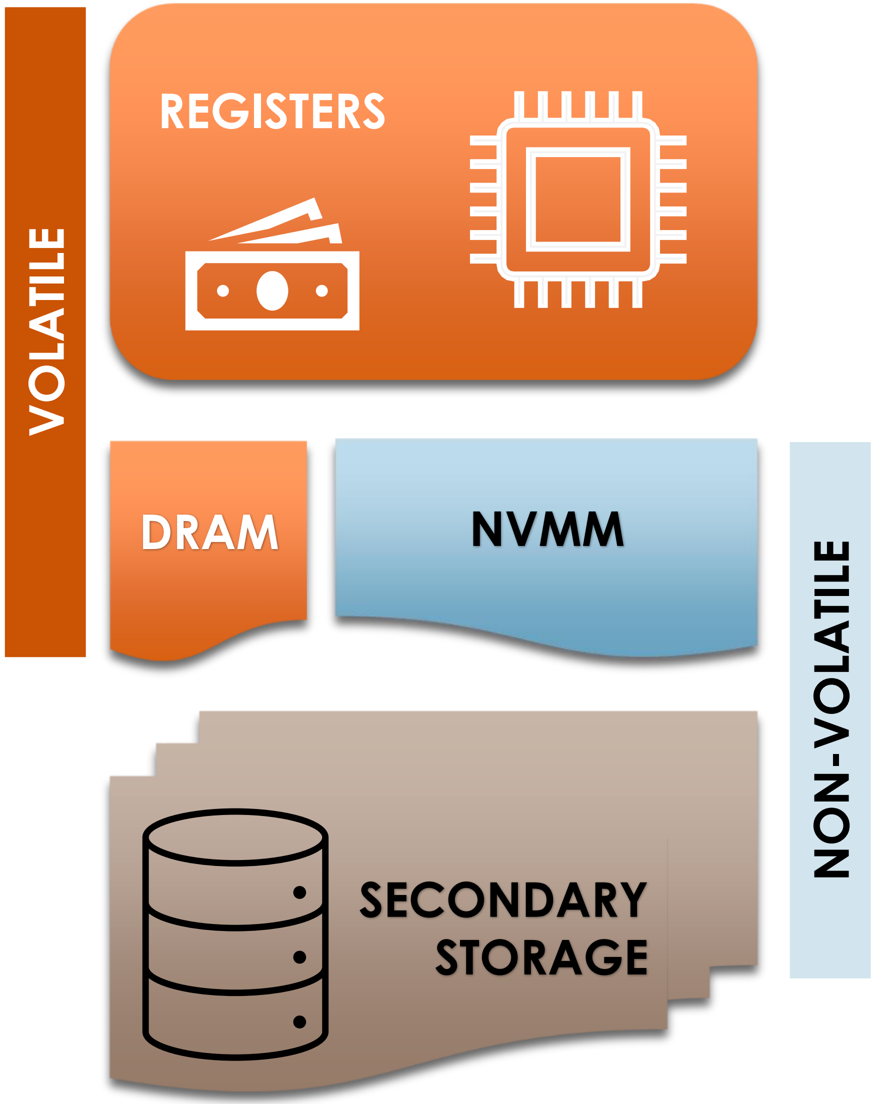
University of Crete, Department of Computer Science

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

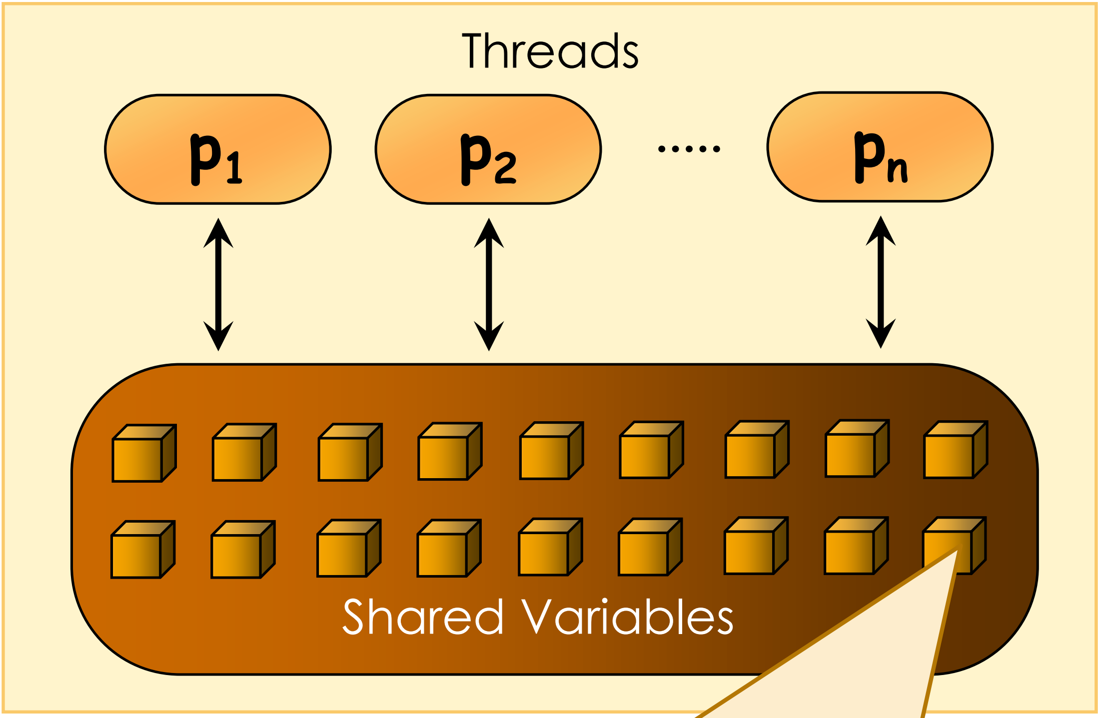
OPODIS 2022

Recoverable Computing

- **Non-Volatile Main Memory (NVMM)**
  - **byte-addressable**
  - **large** and **inexpensive**
  - Recovery in case of failures
    - resets all volatile variables to their initial values
    - the values of non-volatile variables are retained
- **expensive** persistence instructions
  - **`Flush (pwb)`**, **`pfence, psynch`**
- **Efficient** recoverable implementations of fundamental data structures
  - Stacks, queues, lists, trees, etc.



System



Read/Write Variable V
- supports **read(V)** and **write(V,val)**

CAS variable V
- supports **read(V)** and **CAS(V,old, new)**

- Some of the shared variables may be stored in volatile memory, whereas others may be stored in NVMM.

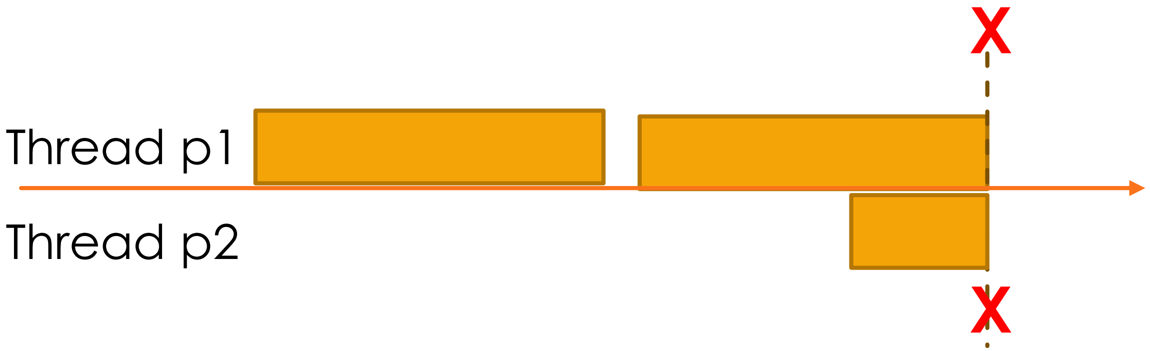
Persistent Instructions

- **Flush (pwb):** write back a cache line in NVM (async)
- **Pfence:** determine order among flushes (async)
- **Psynch:** block until preceding flushes have been realized.

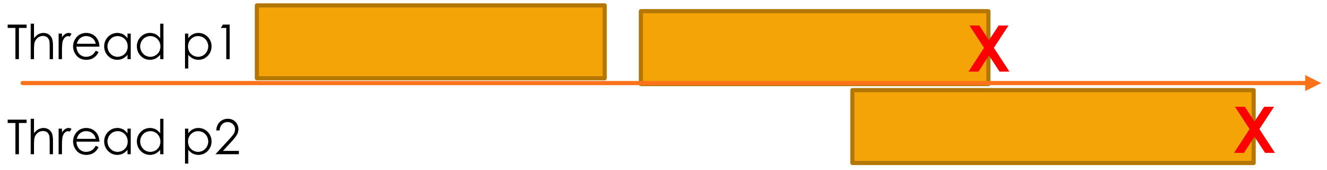
Challenge I

HOW TO APPROPRIATELY MODEL AND ABSTRACT FUNDAMENTAL ASPECTS OF NVM COMPUTING?

Failure Models



- **System-wide failures**
  - All threads fail at the same time
  - Values of variables written back in NVMM remain intact
  - Values of variables stored in volatile memory are lost.
- **Independent thread failures**
  - The execution of any thread p may be abruptly interrupted.
  - The values of local variables of p that are stored in volatile memory are lost.



Recovery Models

- **System-wide recovery**
  - When the system resumes, threads are resurrected.
  - Values of volatile variables are reset to their initial values.
  - A recovery function may exists for the system as a whole.

[NVTraverse]$_{PLDI'20}$, [MIRROR]$_{PLDI'21}$

- **Independent thread recovery**
  - Failed threads recover asynchronously, independently of one another.
    - Initiate new computation (e.g. a new operation, transaction, etc.)

      [CX-PUC, CX-PTM, Redo, RedoOpt]$_{EuroSys'20}$
    - Recovery functions may exist for threads.

      [Capsules]$_{SPPA'19}$, [PBcomb, PWFcomb]$_{PPOPP'22}$, [Tracking]$_{PPOPP'22}$
  - Local volatile variables of the recovered thread are reset to their initial values.
- **Failed threads never recover.** New threads are initiated instead.

[Montage]$_{ICPP'21}$, [nbMontage]$_{DISC'21}$

Progress

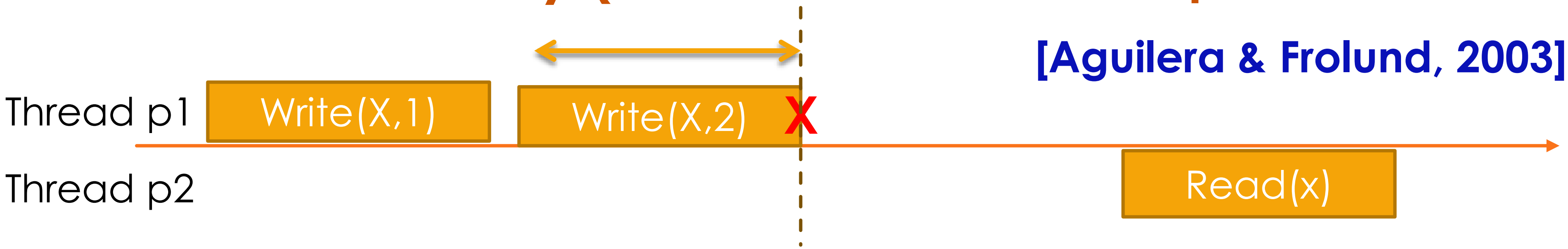
- **Wait-freedom:** Every operation completes within a finite number of steps, if the thread executing the operation does not experience any crash after some point of its execution.

- **Lock-freedom:** In every infinite execution that contains a finite number of crashes, an infinite number of operations complete.

- **Blocking Algorithms**

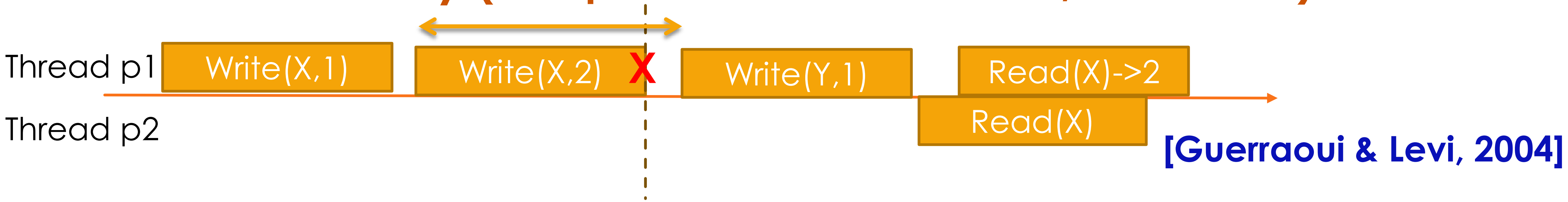
Correctness – Variations of Linearizability

Strict Linearizability (conventional crash-stop failures, no recovery)



Failed operations that are included in the linearization must be linearized by the time of the failure

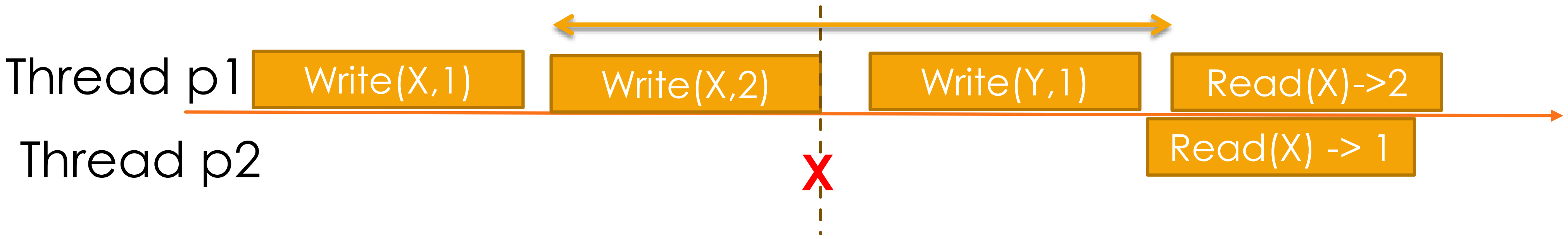
Persistent Atomicity (independent thread failures/recoveries)



Failed operations that are included in the linearization must be linearized before any subsequent invocation of an operation by the same process.

Correctness – Variations of Linearizability

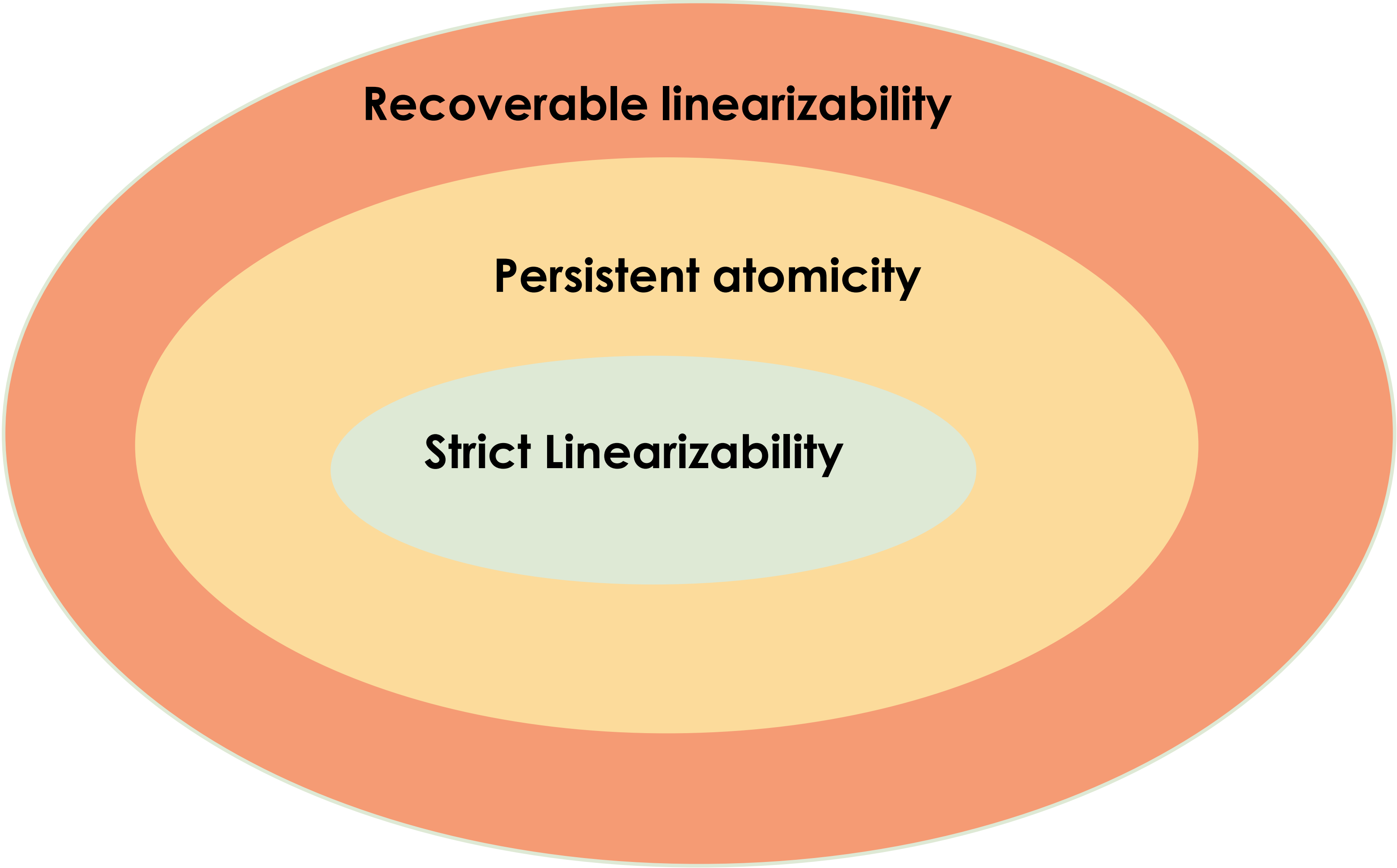
Recoverable Linearizability (system-wide failures)



Failed operations that are included in the linearization must be linearized before any subsequent invocation of an operation on the same object by the same process.

[Berryhill, Golab & Tripunitara, 2015]

Correctness – Variations of Linearizability

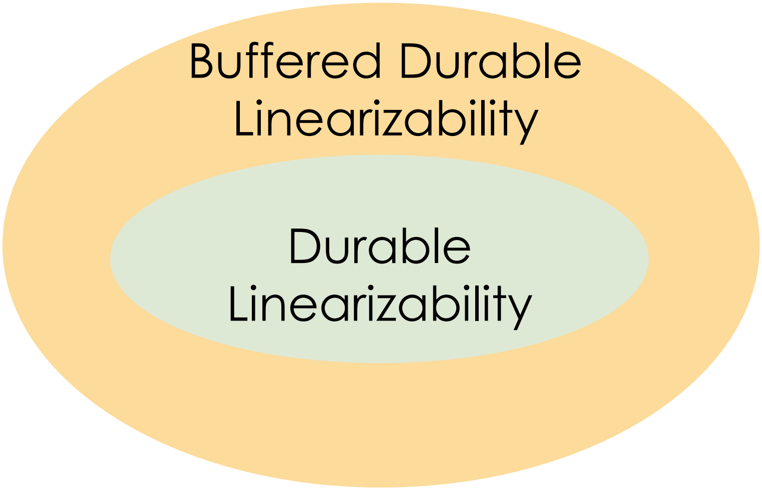


Correctness - System-wide failures, new threads are initiated after a crash

Durable Linearizability

- after a **crash** the state of the object must reflect a history containing all completed operations
- crashed operations may or may not be part of this history

[Izraelevitz, Mendes and Scott. 2016]



Buffered Durable Linearizability

- Relaxed version of durable linearizability which allows for removing some of the completed operations from the linearization

[Izraelevitz, Mendes and Scott. 2016]

Correctness

Detectability (independent thread failures/recoveries)

- A thread infers if its failed operation took effect or not before the crash
- if it took effect, the process obtains the response of its operation

[Friedman, Herlihy, Marathe and Petrank, 2018]

- **Detectability is orthogonal to the previous definitions and can be applied on top of any of them.**

Topics for Thought

- Different failure and recovery models
  - Most realistic? Fair comparison of results proposed for different models?
- Different persistence conditions have been presented under different models
  - Some can easily be transformed from one model to another, others not.
  - Enable a fair comparison of them conditions.
- There are many correctness conditions for the conventional crash-stop model, which have not been studied in a recoverable setting.
  - Causality-based conditions? Correctness conditions for specific settings (e.g., transactional systems, etc.)
- Study trade-offs between correctness and performance, progress and performance, and possibly also between correctness and progress.

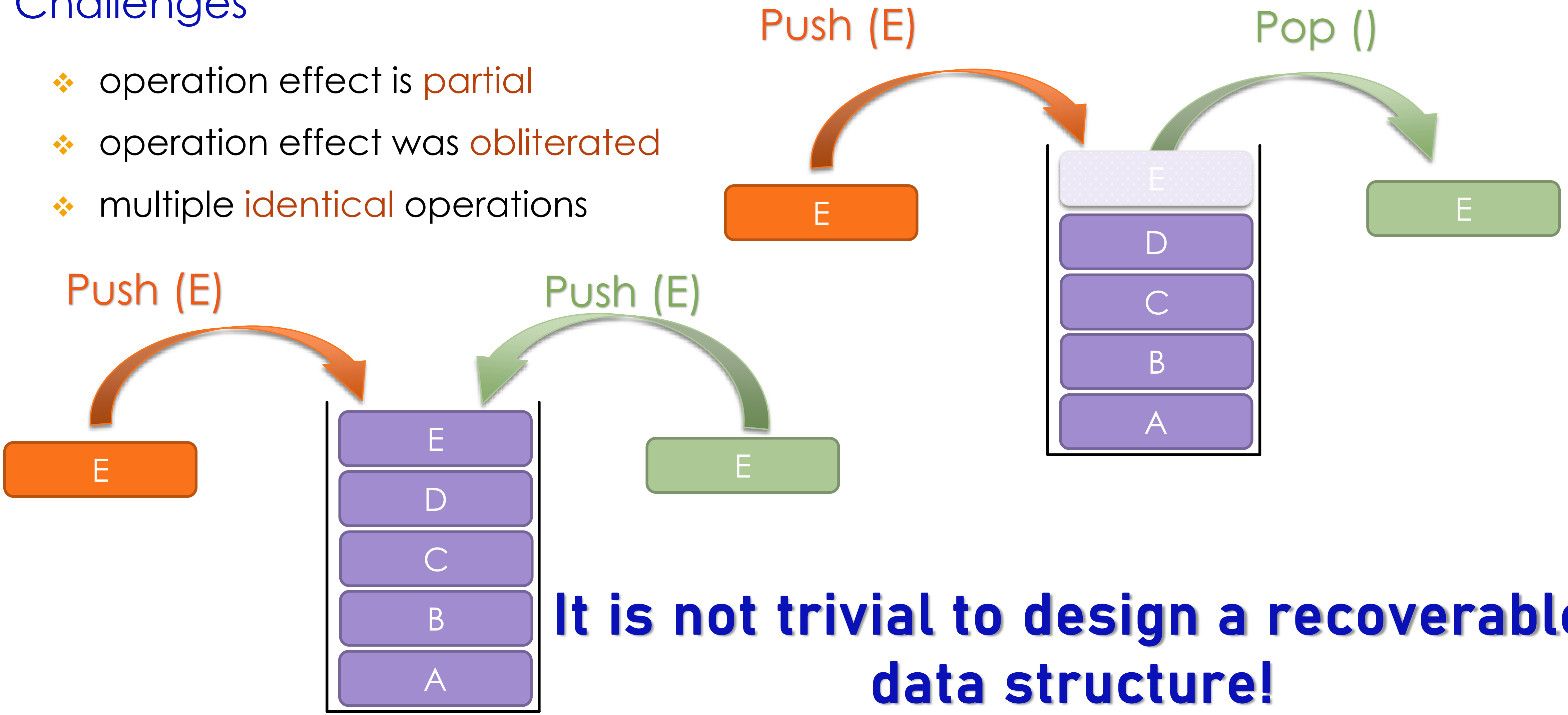
Challenge II

HOW TO COMPUTE IN A RECOVERABLE WAY AT NO SIGNIFICANT COST?

Designing Recoverable Objects

Challenges

- operation effect is partial
- operation effect was obliterated
- multiple identical operations



It is not trivial to design a recoverable data structure!

Main Techniques

- Some form of logging

  - Undo log

    [Atlas] SIGPLAN Not.'14 , [REWIND] VLDB'15 , [Crafty] PLDI'20 , [Clobber-NVM] ASPLOS'21

  - Redo log

    [NV-Heaps] SIGARCH Comput. Archit. News'11 , [Pangolin] ATC'19 , [NVthreads] EuroSys'17 , [DudeTM] ASPLOS'17 , [Romulus] SPAA'18 , [Pisces] USENIX ATC'19 , [OneFile] DSN'19 , [DPTree] VLDB'19 , [PETRA] ACM TACO'21 , [SPHT] FAST'21

Main Techniques

❖ Dual copy techniques

One consistent copy and one working copy on which modifications are performed

- persist working copy, then apply changes to consistent copy

  [Persimmon]$_{\text{OSDI'20}}$, [Pisces]$_{\text{USENIX ATC'19}}$, [MIRROR]$_{\text{PLDI'21}}$

  - If a crash occurs while working copy is being changed, at recovery, copy data from consistent copy to working copy.
  - If a crash occurs after working copy has been persisted, at recovery, replay the write back of the working copy to the consistent copy.

- Alternate roles of working copy and consistent copy [PMThreads]$_{\text{PLDI'20}}$

Main Techniques

❖ Copy on Write

[NVthreads]$_{EuroSys'17}$, [Kamino-Tx]$_{EuroSys'17}$, [DudeTM]$_{ASPLOS'17}$, [WORT]$_{FAST'17}$, [Clfb-tree] $_{ACM\ Trans.\ Storage'18}$, [Trinity, Quadra]$_{PPoPP'21}$, [ArchTM]$_{FAST'21}$, [SPHT]$_{FAST'21}$

- Copy simulated state locally
- Update local copy
- Persist local copy
- Update shared pointer to point to local copy
- Persist the pointer

Main Techniques

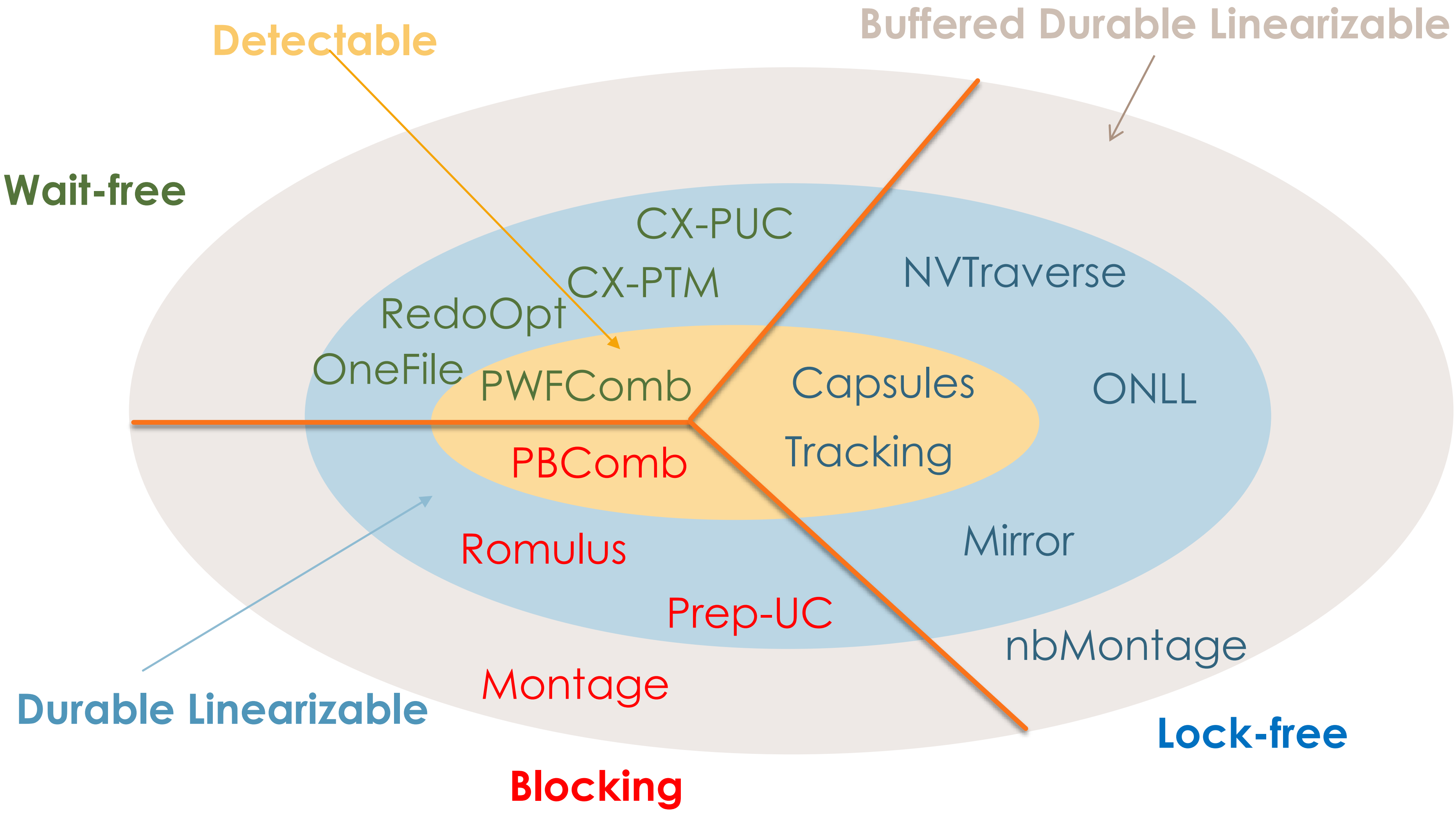
- **Use of Info records (or descriptors)** to record and persist state (info records are often found in lock-free algorithms for implementing helping)

  [Tracking]$_{\text{PPOPP'22}}$, [R. Guerraoui et al.]$_{\text{DISC'20}}$

- **Link-and-Persist** [David at al.]$_{\text{USENIX ATC'18}}$, [Tracking]$_{\text{PPOPP'22}}$, [FliT]$_{\text{PPOPP'22}}$
  - Avoid executing pwb instructions when the variable being flushed is clean.
  - Use a single bit in each memory word as a flag indicating whether or not it has been flushed since the last time it was updated.
  - A reader executes a pwb and psynch on any location it reads that had the flag up, and skips persisting every time the flag is down.

- **Combination of different techniques** for different components to exploit benefits and mask weaknesses.

Universal Constructions and General Transformations for Designing Persistent DS



- **PWFComb**, PBComb, Fatourou et al., PPoPP'22
- **Tracking**, Attiya et al., PPoPP'22
- **Capsules**, Ben-David et al., SPAA'19
- **CX-PTM, CX-PUC, RedoOpt**, Correia et al., EuroSys'20
- **OneFile**, Ramalhete et al., DSN'19
- **NVTraverse**, Friedman et al., PLDI'20
- **ONLL**, Cohen et al., SPAA'18
- **Mirror**, Friedman et al., PLDI'21
- **Romulus**, Correia et al., SPAA'18
- **Prep-UC**, Coccimiglio et al., SPAA'22
- **Montage**, Wen et al., ICPP'21
  nbMontage, Cai et al., DISC'21

Persistence Principles Crucial for Performance

Fatourou, Kallimanis & Kosmas, PPoPP'22

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 aims at achieving this]`
2. **The persistence instructions should be of low cost (e.g., by persisting less highly-contented shared variables)**
   - Avoid pwbs on variables on which CAS is performed before or after `[Tracking]`$_{\text{PPOPP'22}}$
   - Reduce accesses to recently flushed cache lines `[Sela & Petrank]`$_{\text{SPAA'21}}$, `[MIRROR]`$_{\text{PLDI'21}}$
3. **Data to be persisted should be placed in consecutive memory addresses, so that they are persisted all together**

   `[PBcomb, PWFcomb]`$_{\text{PPOPP'22}}$, `[ArchTM]`$_{\text{FAST'21}}$

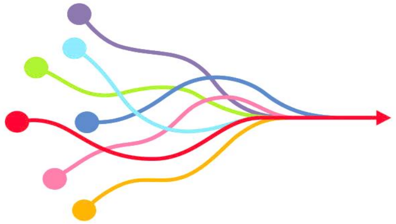
Persistent Software Combining

Design Decisions of Combining Protocols Crucial for Performance

A thread attempts to become a combiner and serve in addition to its own request, active requests by other threads

After announcing their requests , other threads may:

- either perform local spinning until the combiner performs their requests
- or perform the same actions as the combiner (although not always "successfully")



A. Mechanism to choose which thread will act as the combiner

B. Data structure to store the active requests

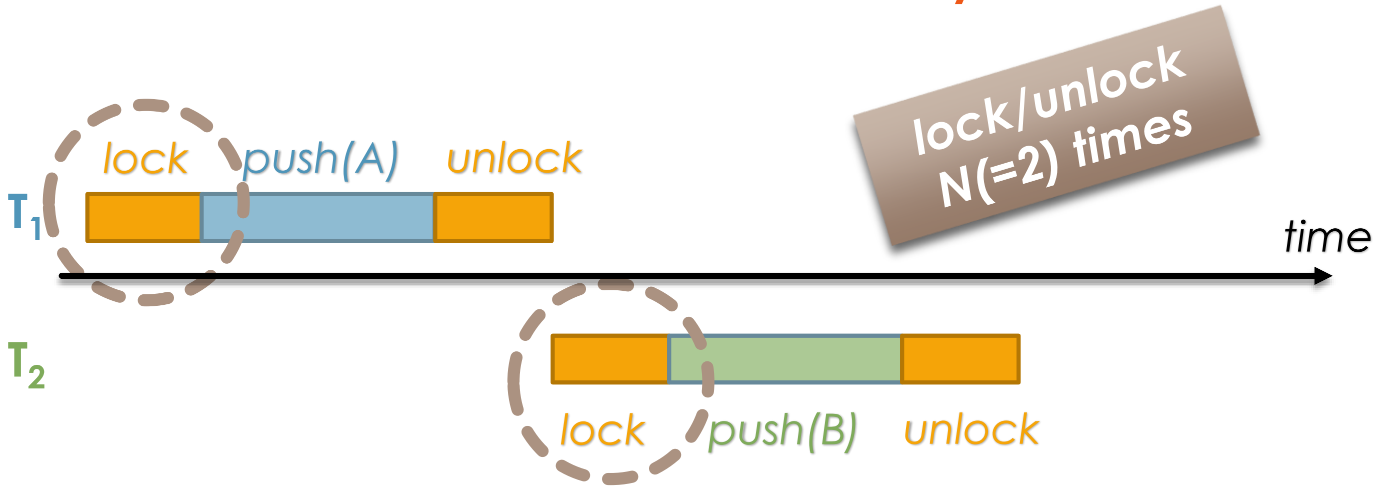
C. Mechanism to apply the updates

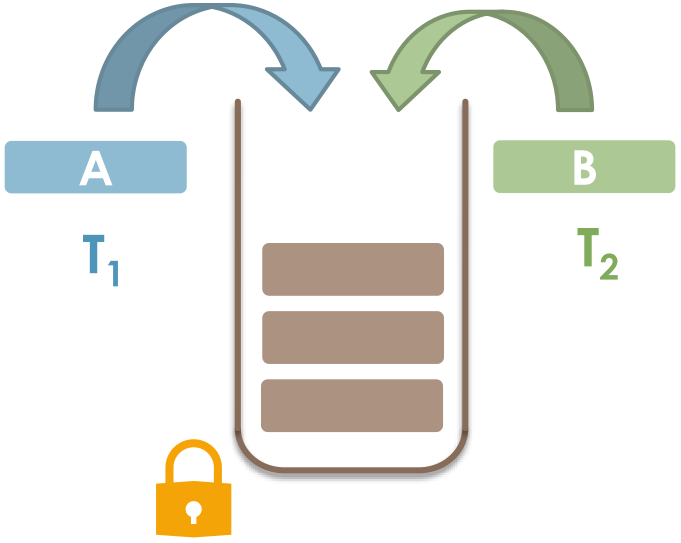
D. Mechanism for collecting the requests' responses

E. Mechanism to discover which requests have been applied or not.

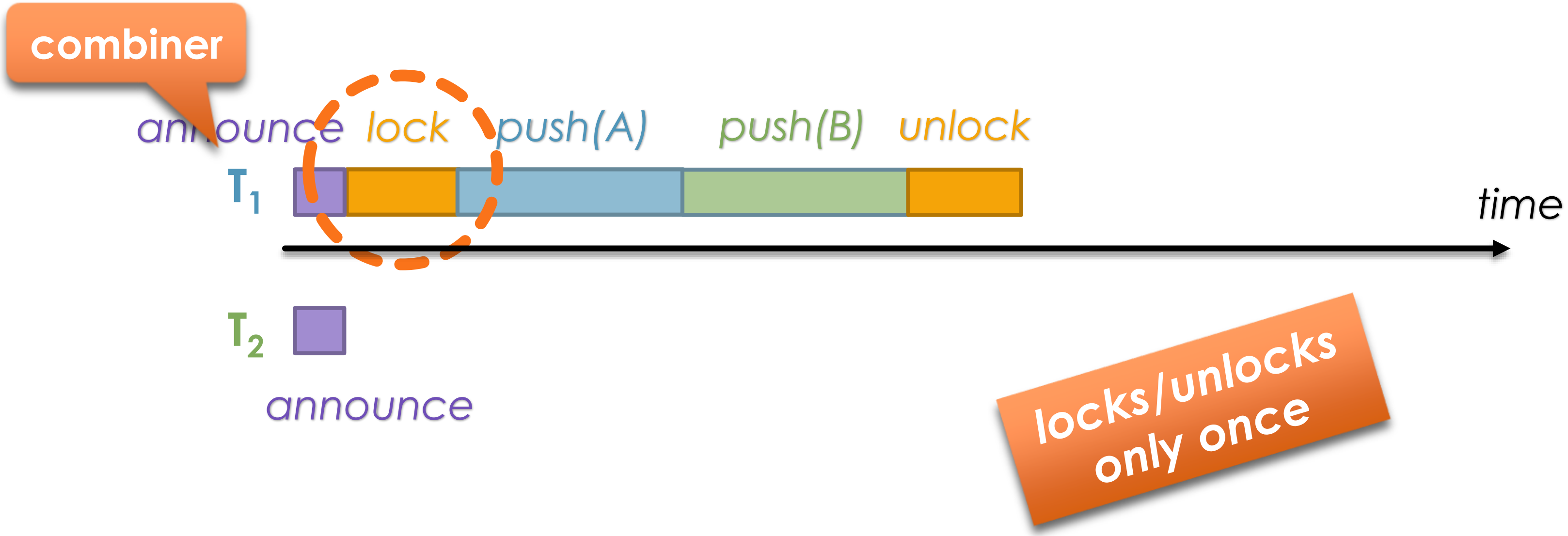
[Fatourou, Kallimanis & Kosmas, PPoPP 2022 - Best Paper Award]

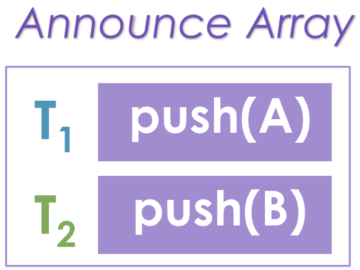
Why is combining promising in conventional DRAM systems?





conventional *lock-based implementation*

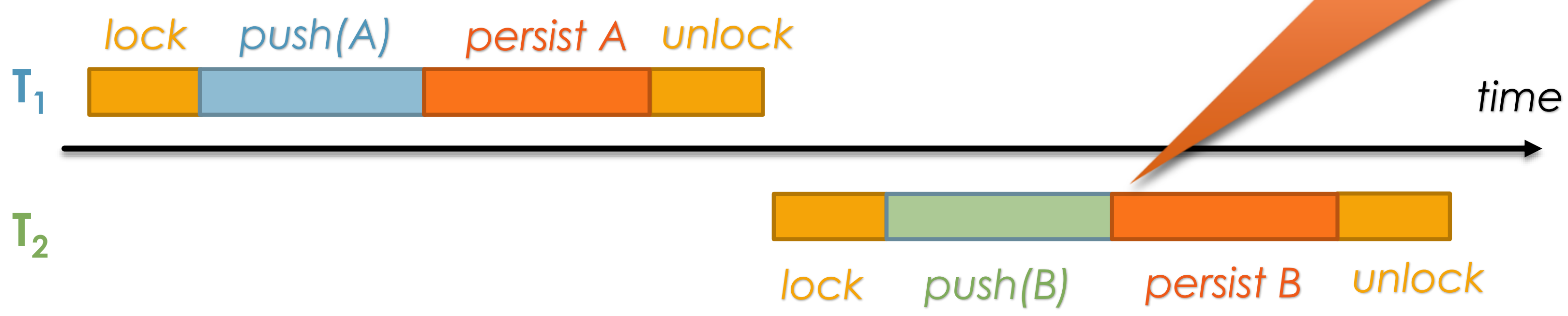




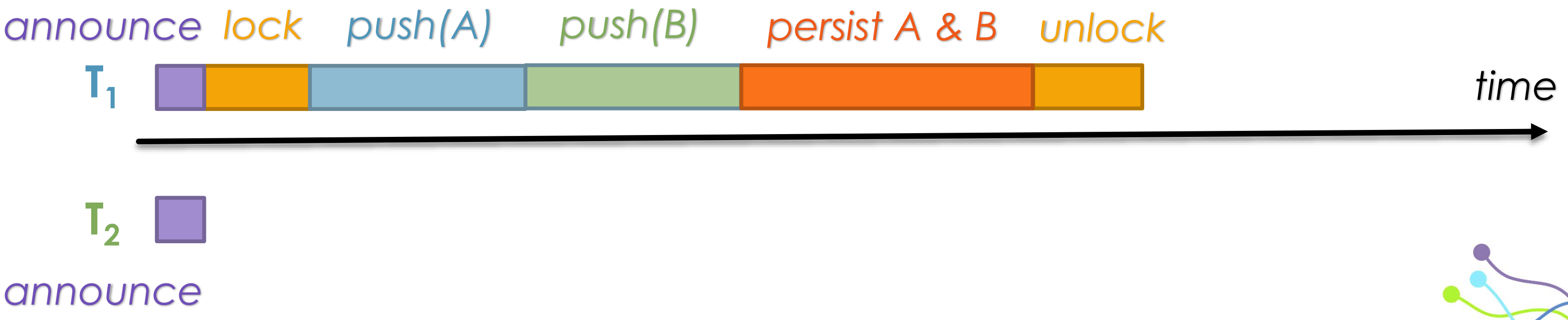
Software Combining *technique*

Why is combining promising in an NVM setting?

persist = usually two instructions (pwb & psynch)



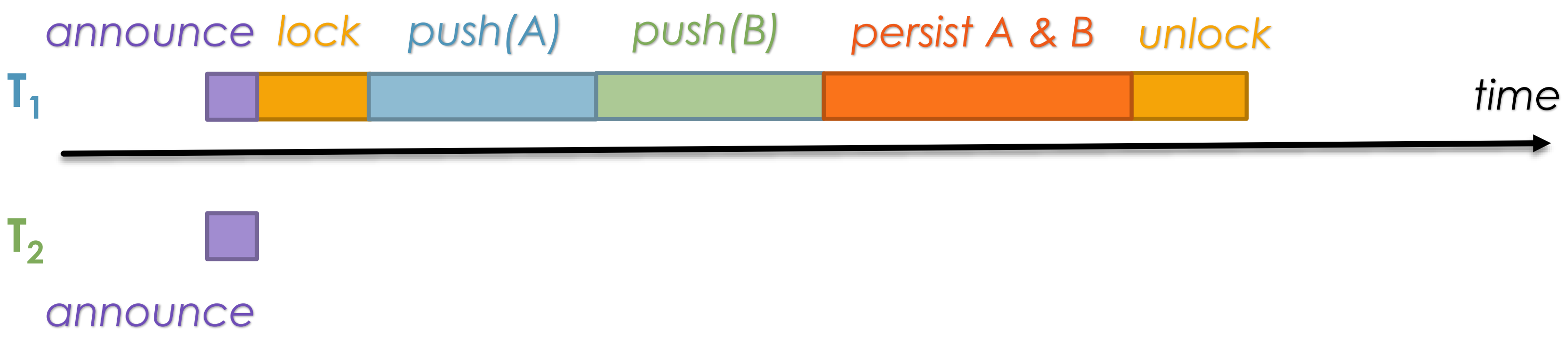
conventional
recoverable *lock-based implementation*

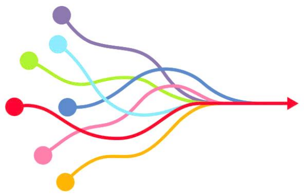


Software Combining
technique

Key Idea

Why is this a promising approach?





Software Combining →
Efficient Recoverable
Data Structures

Benefits:

- ✓ **reduced** number of **synch** instructions
- ✓ store **multiple** nodes into a single cache line → reduced number of flushes

Persistent Software Combining

Efficient recoverable blocking and wait-free

- **synchronization protocols**
  - **outperform** previously proposed recoverable UCs

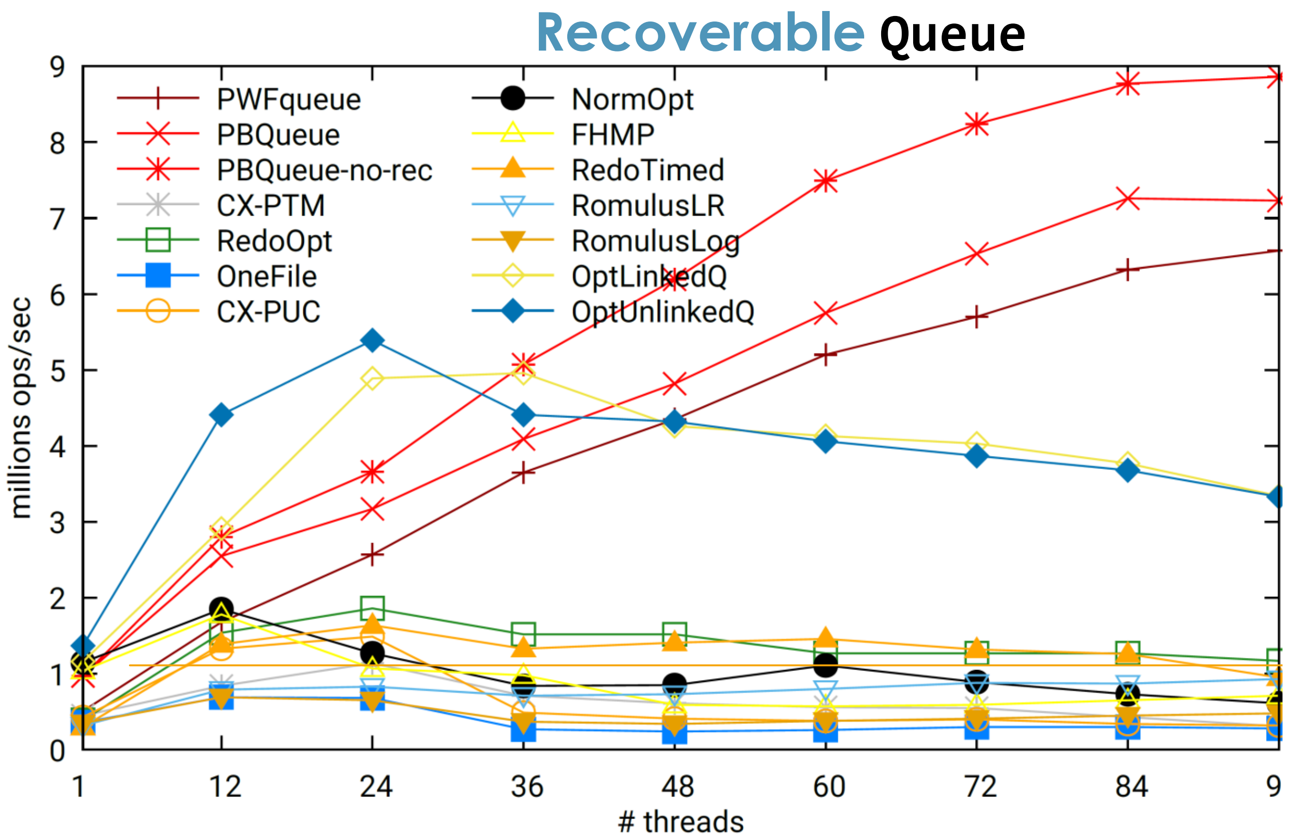
    [RedoOpt]$_{\text{EuroSys'20}}$ and STMs [CX-PTM]$_{\text{EuroSys'20}}$, [OneFile]$_{\text{DSN'19}}$

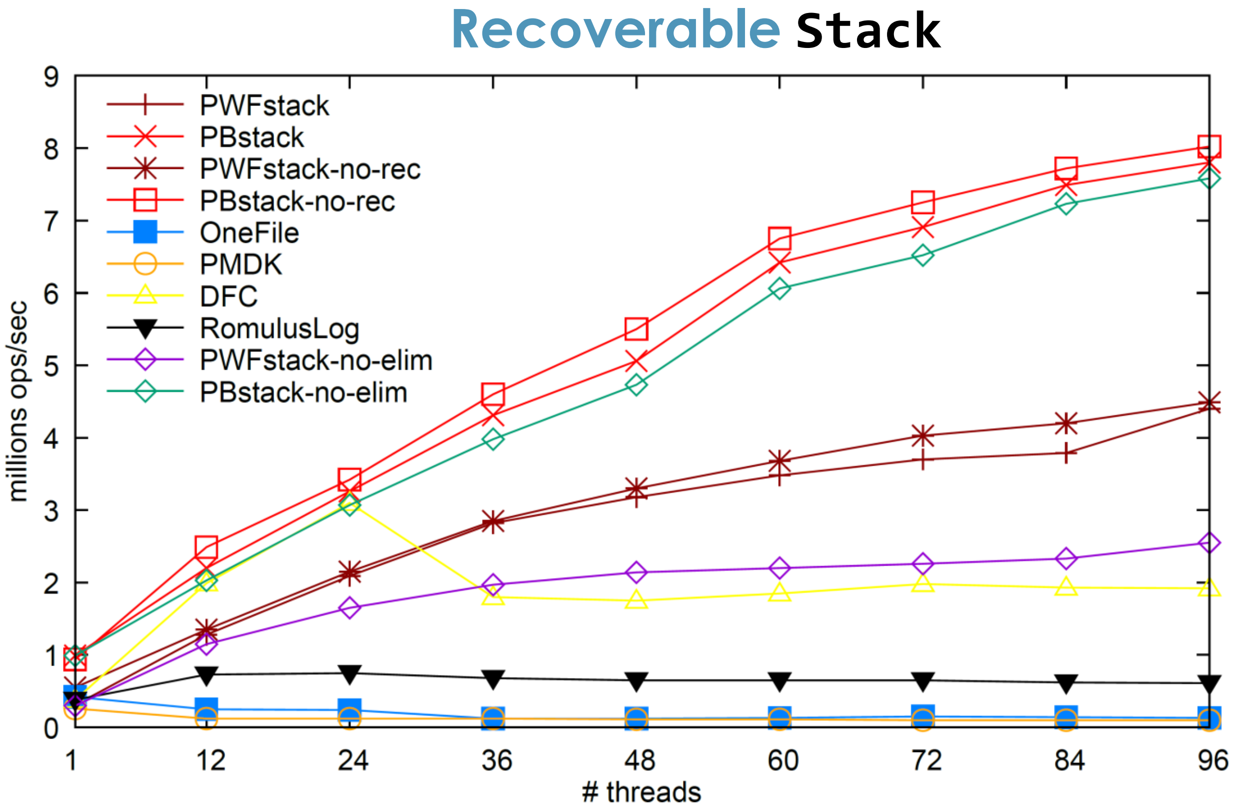
- **Stacks**, **queues** and **heaps**
  - **outperform** previous implementations (including specialized)
    - queues [OptLinkedQ, OptUnLinkedQ]$_{\text{SPAA'21}}$, [CX-PUC, CX-PTM, RedoOpt]$_{\text{EuroSys'20}}$, [OneFile]$_{\text{DSN'19}}$, [Capsules]$_{\text{SPPA'19}}$, [Friedman et al]$_{\text{PPoPP'18}}$,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

[Fatourou, Kallimanis & Kosmas, PPoPP 2022]

Performance Analysis

Fundamental Data Structures

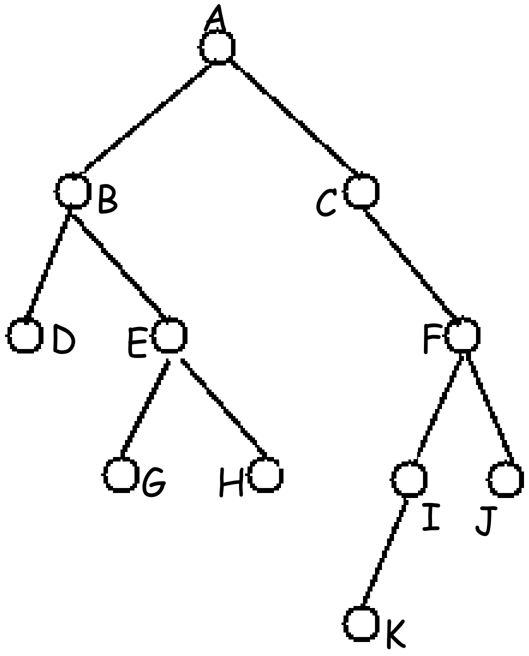


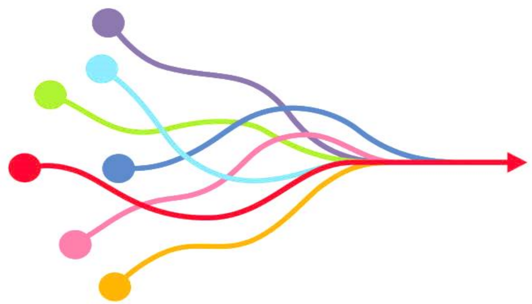


Combining Technique: Can it always be applied efficiently?

- Using a single thread to apply all active requests may restrict parallelism, if the size of the object is small or the number of synchronization points is constant.

- Multiple searches (or even updates) could proceed in parallel in a tree-like data structure.





Tracking – Detectable Lock-Free DS

Derive efficient **recoverable** implementations of concurrent, lock-free data structures

Technique:

- per-operation **Info Structure**
  - **tracks** operation's progress
  - it is **persisted** to NVM
- a **pragmatic** scheme to add persistence instructions
- **mechanical** transformation
  - linked list, binary search tree, exchanger

Benefits:

- ✓ **avoids** full-fledged logging
- ✓ **reduces** the persistence cost for ensuring detectable recovery → yields **efficient** implementations

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

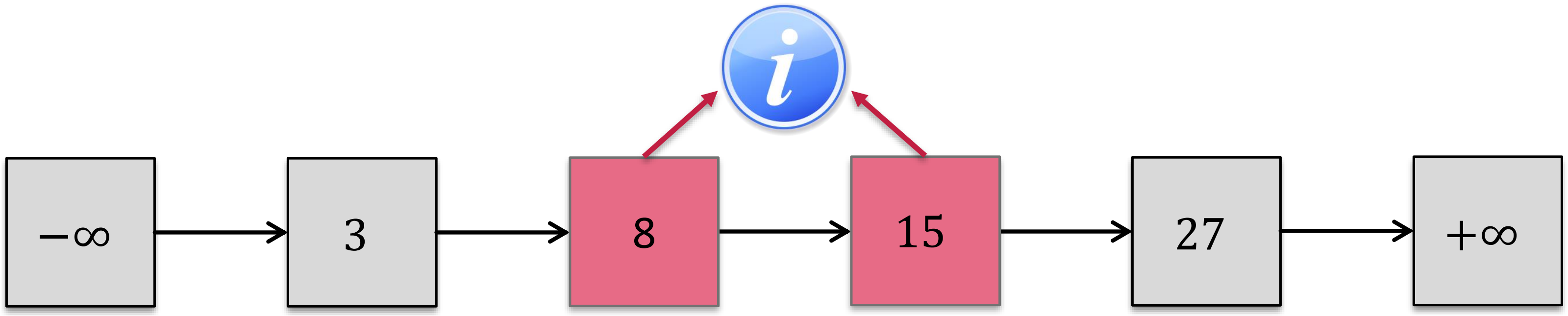
Info-Structure Based-Tracking

Example: Linked List

- each node is augmented with a special info field, containing a pointer to an **IS**

`Op: Delete(15)`

1. after **`Op`** initialize its **IS**, it attempts to install it in any node that **`Op`** may affect



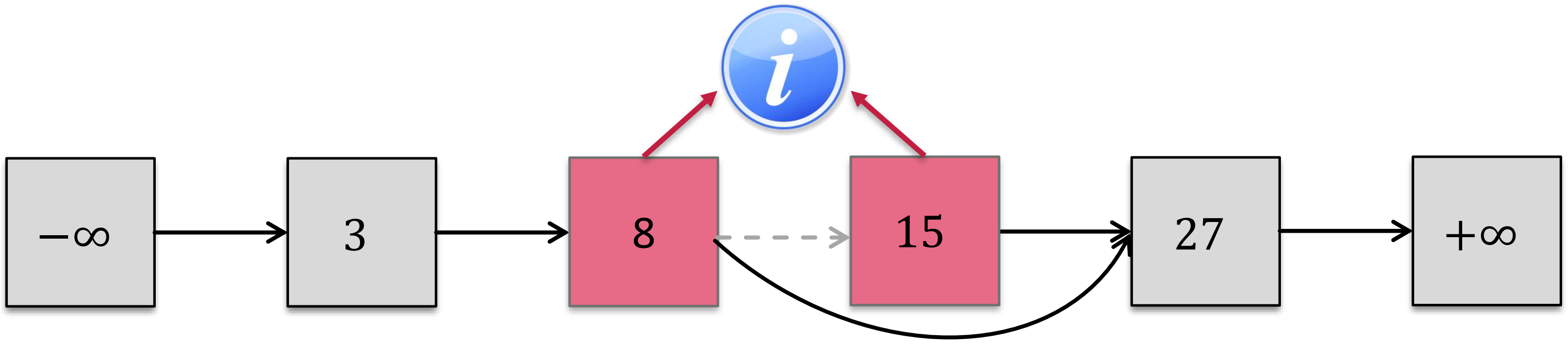
Info-Structure Based-Tracking

Example: Linked List

- each node is augmented with a special info field, containing a pointer to an **IS**

`Op: Delete(15)`

1. after **`Op`** initialize its **IS**, it attempts to install it in any node **`Op`** may affect
2. once successful, **`Op`** can be completed using this information (also by other threads)



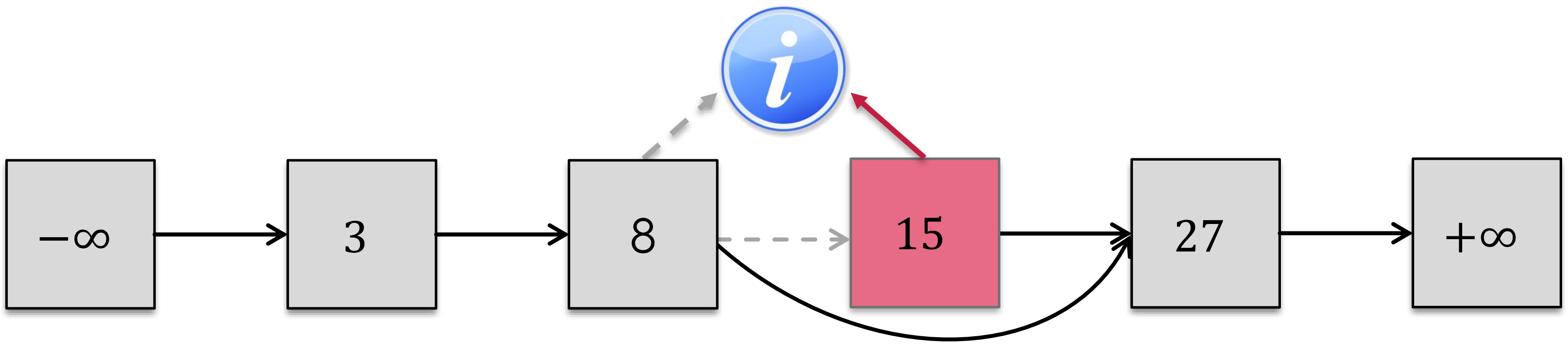
Info-Structure Based-Tracking

Example: Linked List

- each node is augmented with a special info field, containing a pointer to an **IS**

`Op: Delete(15)`

1. after **`Op`** initialize its **IS**, it attempts to install it in any node **`Op`** may affect
2. once successful, **`Op`** can be completed using this information (also by other processes)
3. after making its changes, **`Op`** uninstalls its **IS**

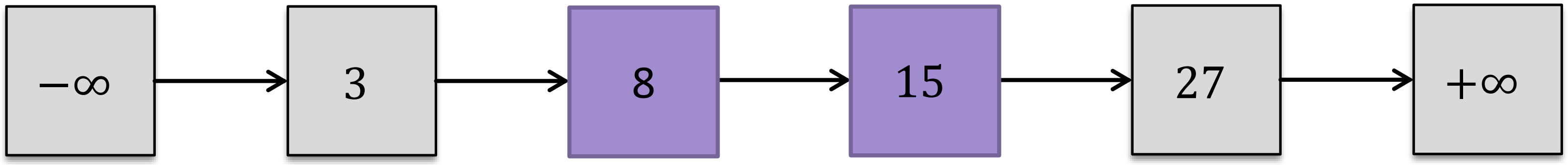


Info-Structure Based-Tracking

Mechanical Transformation

`Procedure Op (args)`

1. **`Gather Phase`**: collect nodes that may be affected by **`Op`** → **`AffectSet`**

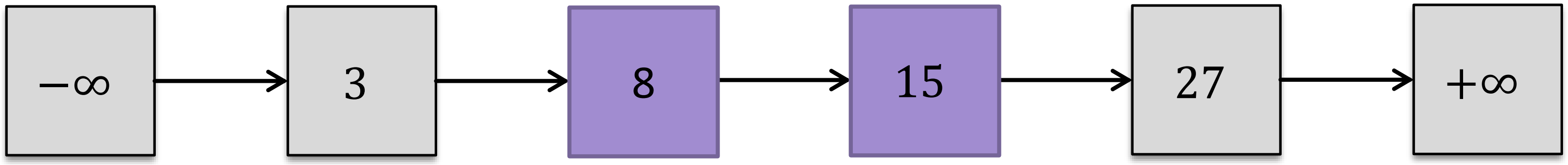


Info-Structure Based-Tracking

Mechanical Transformation

`Procedure Op (args)`

1. **`Gather Phase`**: collect nodes relevant to **`Op`** → **`AffectSet`**
2. **`Helping Phase`**: help operations pointed to by info of nodes in **`AffectSet`** if needed; restart
3. **`opInfo`** ← a new Info Structure containing the data of **`Op`**



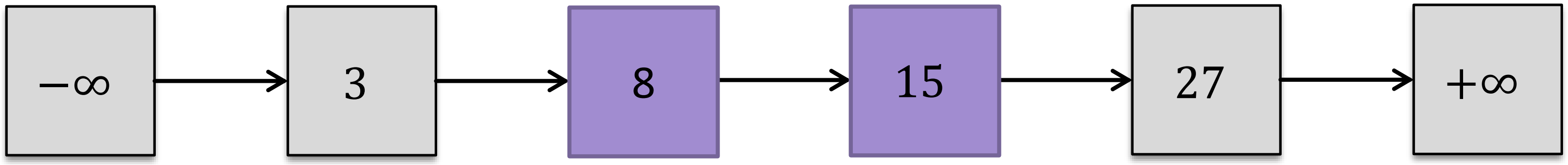
Info-Structure Based-Tracking

Mechanical Transformation

Procedure Op (args)

1. **Gather Phase**: collect nodes relevant to **Op** → **AffectSet**
2. **Helping Phase**: help nodes in **AffectSet** if needed; restart
3. **opInfo** ← a new Info Structure containing the data of **Op**

$\boldsymbol{AffectSet = \ node\ 8, node\ 15}$
$\boldsymbol{WriteSet = \ update\ node\ 8\ to\ point\ to\ node\ 27}$
$\boldsymbol{result = \perp}$

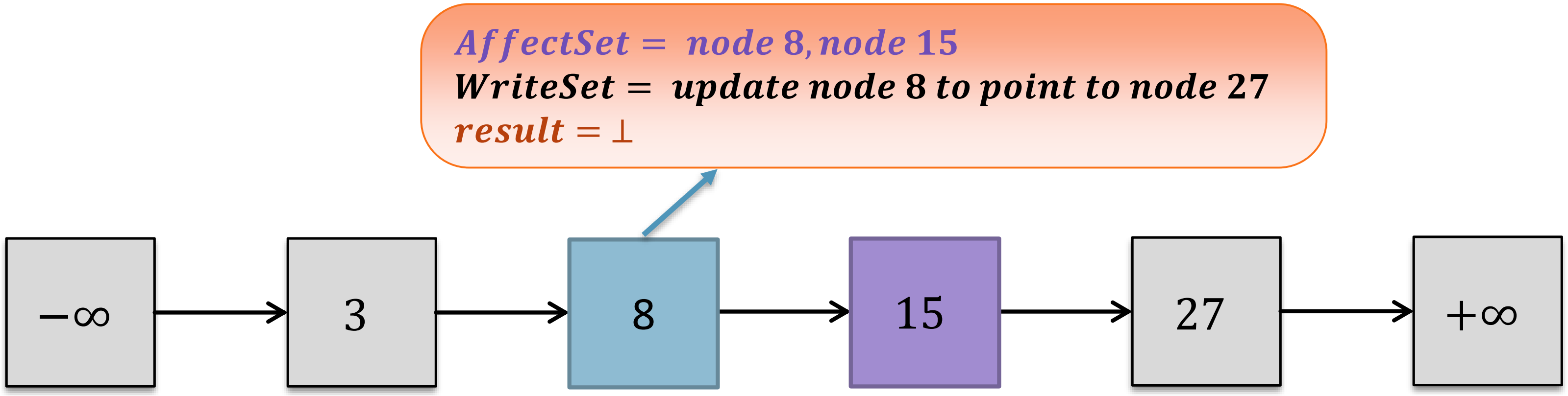


Info-Structure Based-Tracking

Mechanical Transformation

Procedure Op (args)

1. **Gather Phase**: collect nodes relevant to **Op** → **AffectSet**
2. **Helping Phase**: help nodes in **AffectSet** if needed; restart
3. **opInfo** ← a new Info Structure containing the data of **Op**
4. **Tagging Phase**: install pointer to **opInfo** in all nodes of **AffectSet**

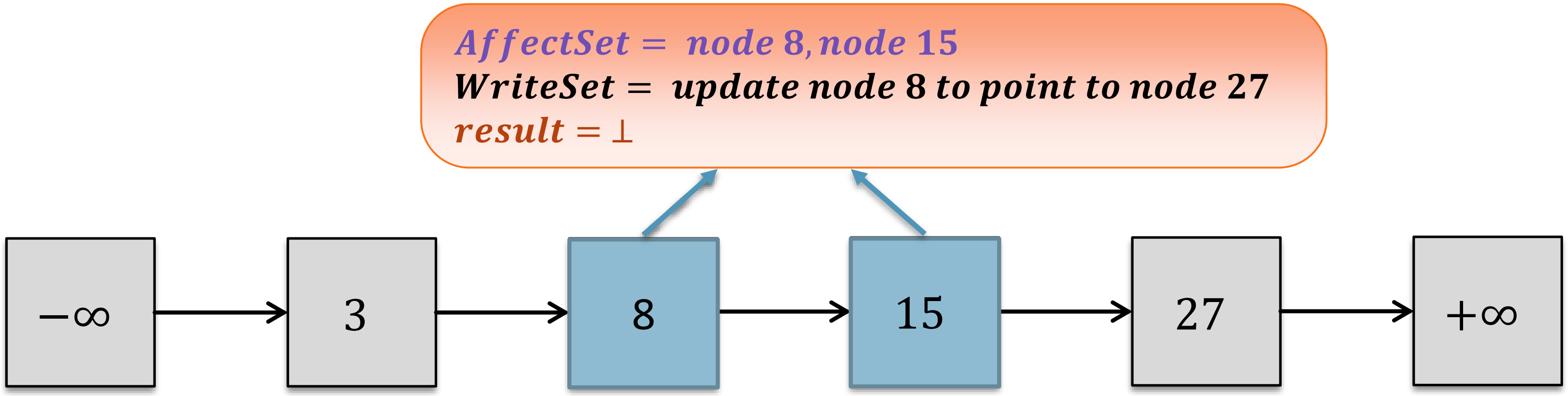


Info-Structure Based-Tracking

Mechanical Transformation

Procedure Op (args)

1. **Gather Phase**: collect nodes relevant to **Op** → **AffectSet**
2. **Helping Phase**: help nodes in **AffectSet** if needed; restart
3. **opInfo** ← a new Info Structure containing the data of **Op**
4. **Tagging Phase**: install pointer to **opInfo** in all nodes of **AffectSet**
   i. **Backtrack Phase**: if tagging fails, untag all nodes; restart

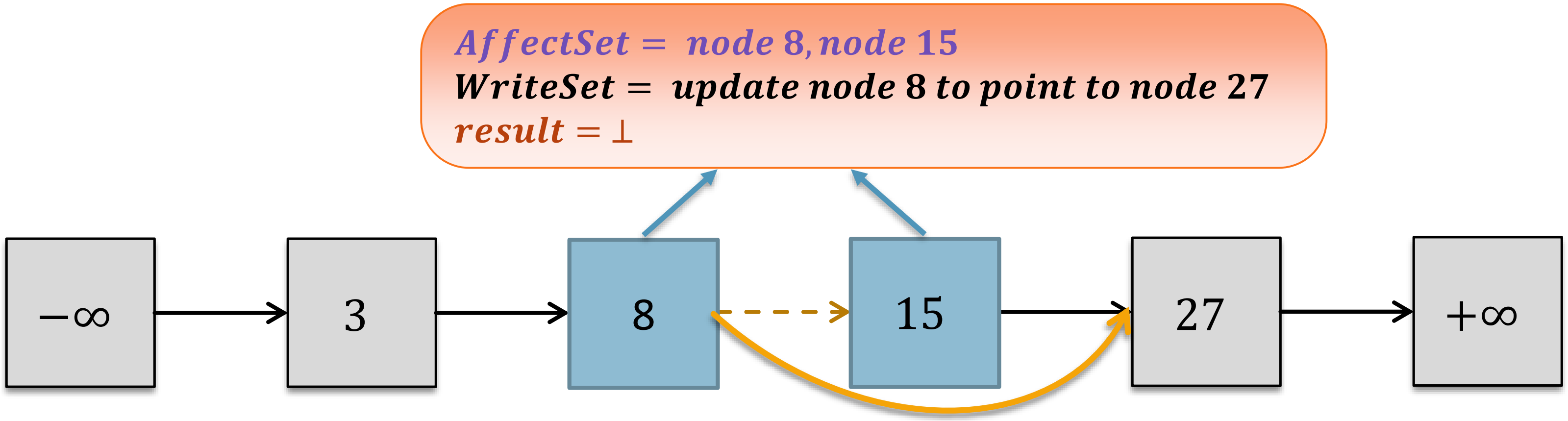


Info-Structure Based-Tracking

Mechanical Transformation

Procedure Op (args)

1. **Gather Phase**: collect nodes relevant to **Op** → **AffectSet**
2. **Helping Phase**: help nodes in **AffectSet** if needed; restart
3. **opInfo** ← a new Info Structure containing the data of **Op**
4. **Tagging Phase**: install pointer to **opInfo** in all nodes of **AffectSet**
   i. **Backtrack Phase**: if tagging fails, untag all nodes; restart
5. **Update Phase**: make all the changes of **Op**



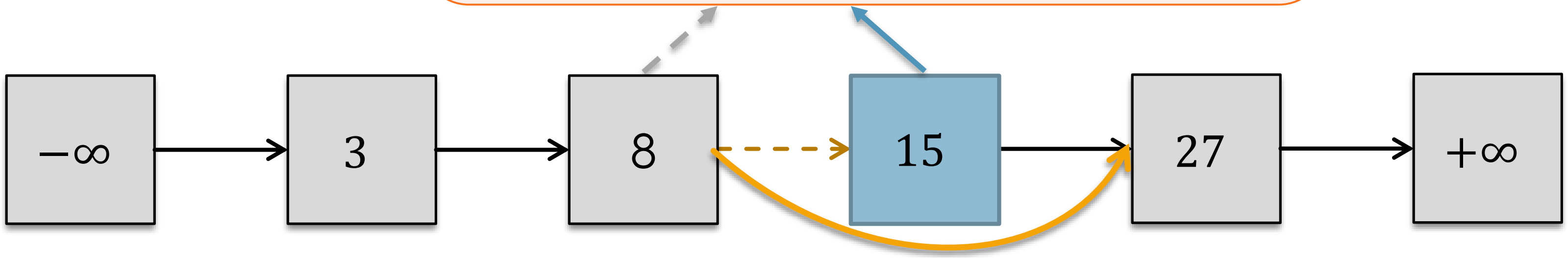
Info-Structure Based-Tracking

Mechanical Transformation

Procedure Op (args)

1. **Gather Phase**: collect nodes relevant to **Op** → **AffectSet**
2. **Helping Phase**: help nodes in **AffectSet** if needed; restart
3. **opInfo** ← a new Info Structure containing the data of **Op**
4. **Tagging Phase**: install pointer to **opInfo** in all nodes of **AffectSet**
   i. **Backtrack Phase**: if tagging fails, untag all nodes; restart
5. **Update Phase**: apply all the changes of **Op**
6. **opInfo.result** ← **Op**'s response
7. **Cleanup Phase**: untag nodes still in the DS

$AffectSet = node\ 8, node\ 15$
$WriteSet = update\ node\ 8\ to\ point\ to\ node\ 27$
$result = true$



Info-Structure Based-Tracking

Mechanical Transformation – Adding Persistence Instructions

Procedure Op (args)

1. **Gather Phase**: collect nodes relevant to `Op` → `AffectSet`
2. **Helping Phase**: help nodes in `AffectSet` if needed; restart
3. `opInfo` ← a new Info Structure containing the data of `Op`
   pwb(opInfo); psync();
4. **Tagging Phase**: install pointer to `opInfo` in all nodes of `AffectSet`
   pwb after any install
   i. Backtrack Phase: if tagging fails, untag all nodes
      pwb after any untag
      psync at the end

      restart

   psync();
5. **Update Phase**: make all the changes of `Op`
   pwb after any update
6. `opInfo.result` ← `Op`'s response
   pwb(opInfo.result); psync();
7. **Cleanup Phase**: untag nodes still in the DS

OPEN QUESTIONS

- Most proposed algorithms have been designed to ensure Performance Principle 1. Is it possible to design more efficient algorithms by taking into consideration all performance principles?
- Recoverable versions of concurrent data structures
  - Skip lists [*Chowdhury & Golab, SPAA'21, Xiao et al., IEEE Access'21*]
  - Priority Queues [**PBHeap**, Fatourou et al, PPoPP'22]
  - Specialized tree implementations
  - Specialized Queue implementations
  - Graphs
  - NUMA-aware data structures [**Prep-UC**, Coccimiglio et al., SPAA'22]
- Recoverable Garbage Collection

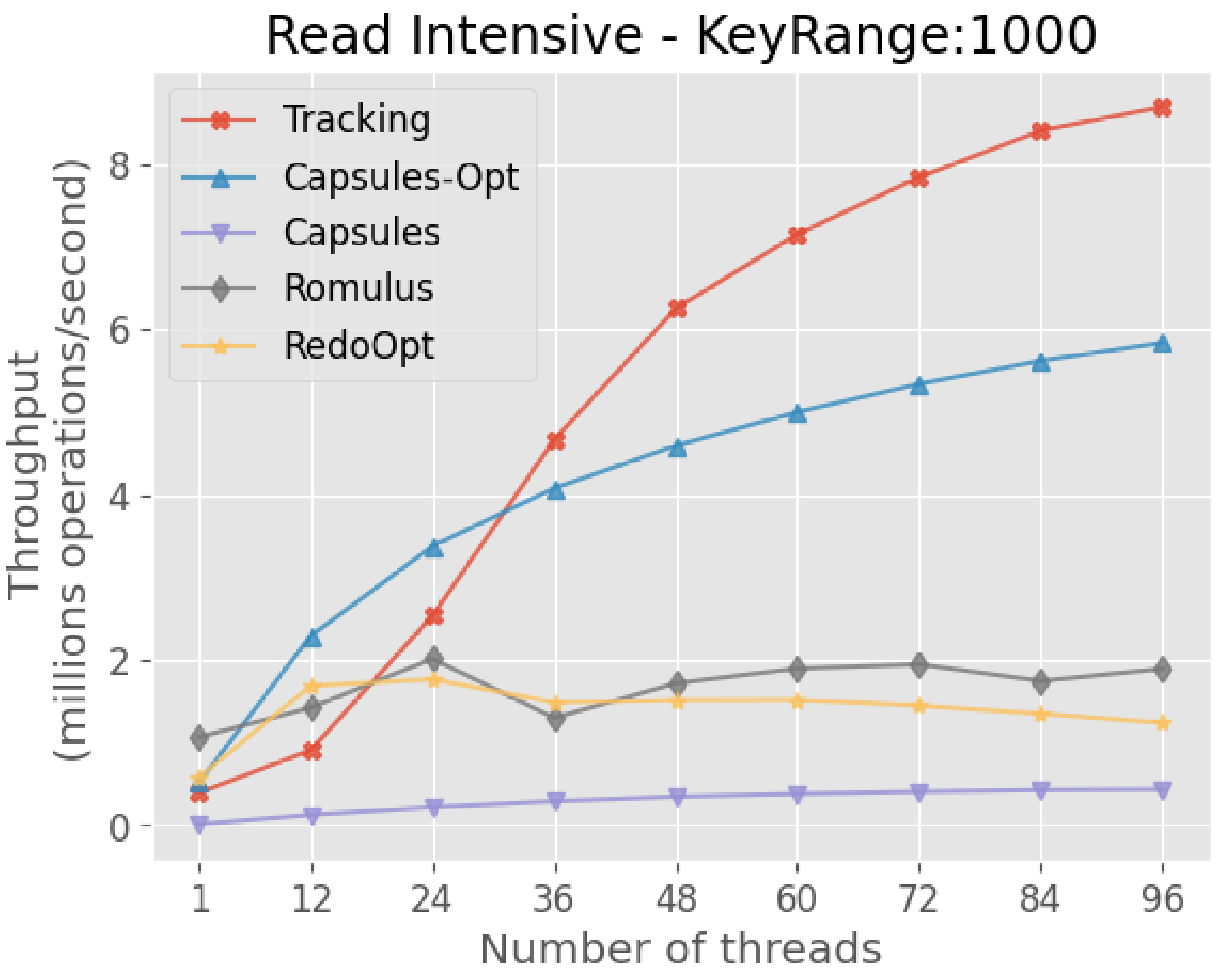
Challenge III

HOW TO ANALYZE THE COST OF RECOVERABLE ALGORITHMS?

Tracking Evaluation

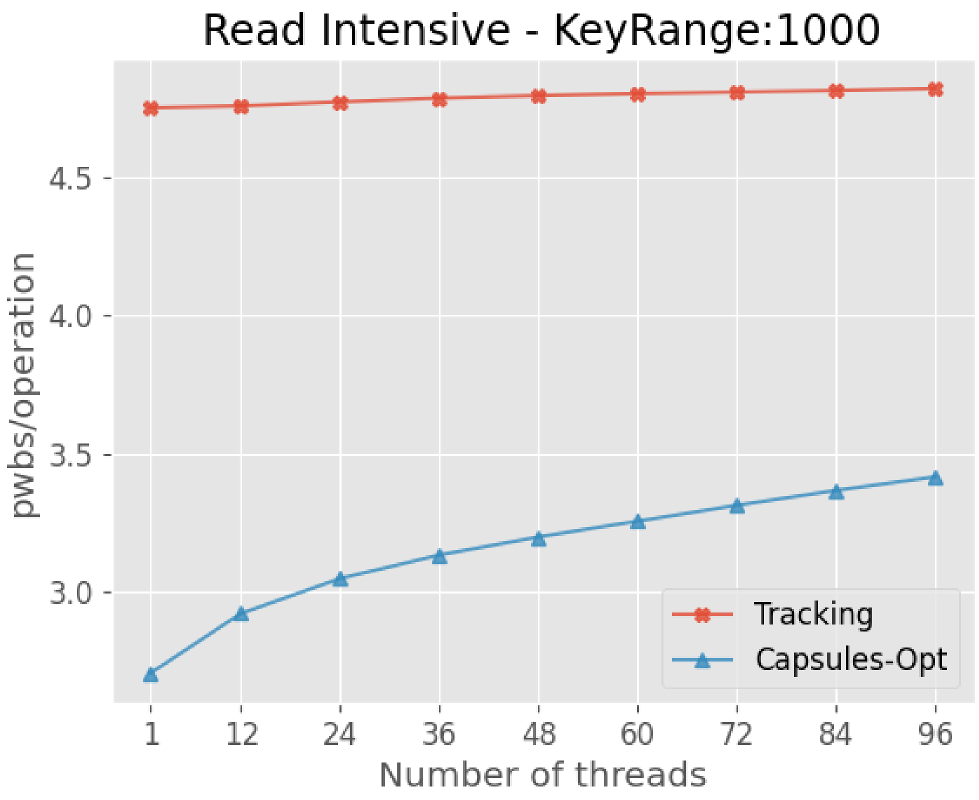
Linked-List Based Set

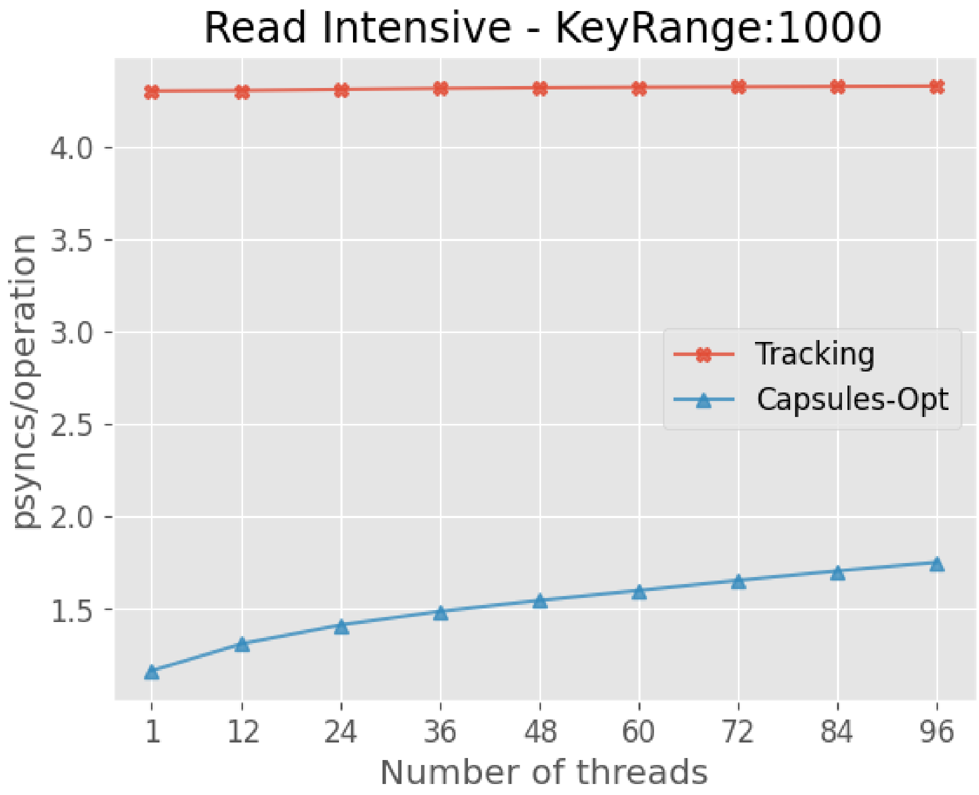
- **Tracking** Linked List (no hand-tuning has been applied)
- **Capsules-Opt:** strongly hand-tuned transformation of Harris' linked list using capsules
  [Attiya et al., PPoPP 2022]
- Capsules: general scheme described by Capsules authors (not hand-tunned)
  [Ben-David, Blelloch, Wei. 2018]
- **Romulus**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



Tracking exhibits better performance as the number of threads increases.

Evaluation
Linked-List Based Set



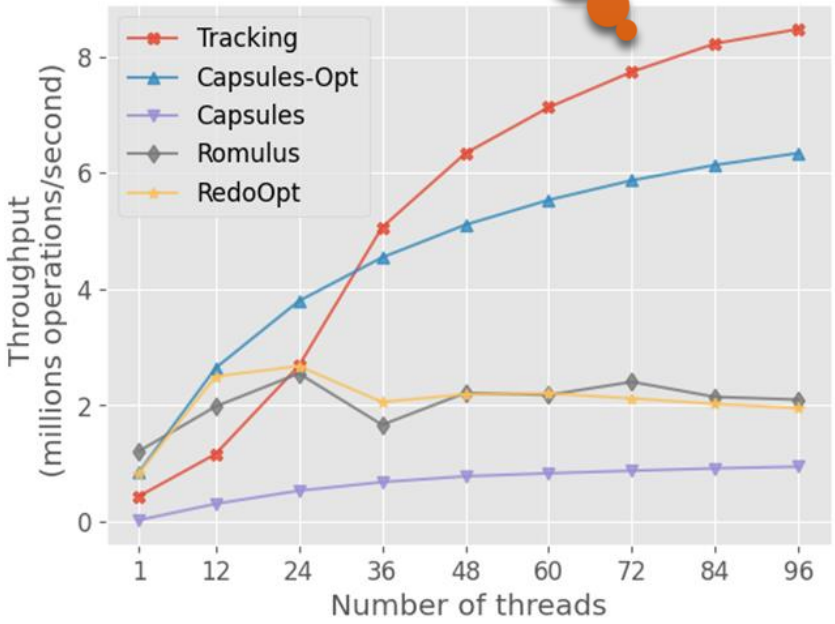


The synchronization cost of Tracking is also higher than that of Capsules-Opt.

Evaluation

Linked-List Based Set

What causes the good performance of Tracking?



- Tracking performs more psyncs → negligible cost
- **Tracking** performs **more pwbs**

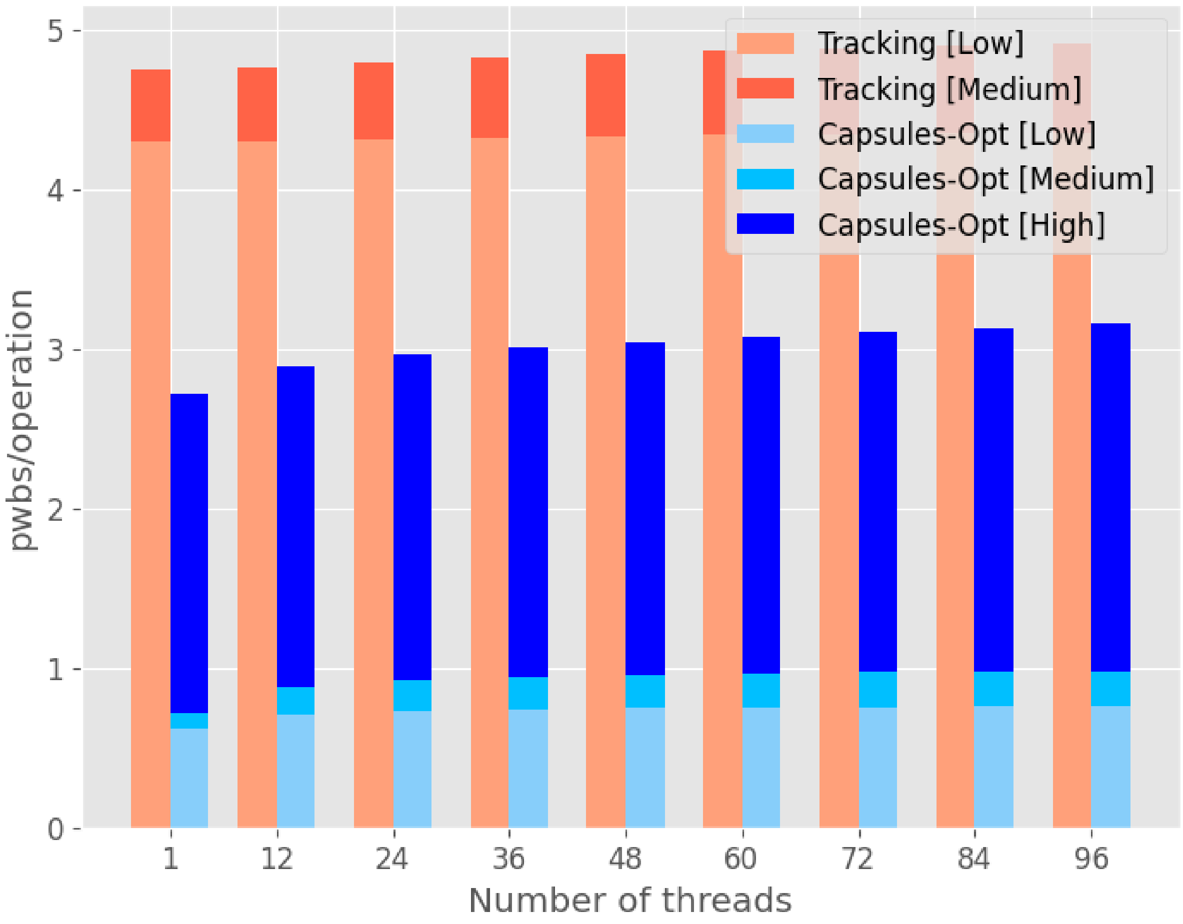
what about the impact of each single persistence instruction?

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

```
1. remove all code lines with persistence instructions
2. for each removed code line L that contains a pwb
3.       add L to code
4.       run experiment (to measure L's impact)
5.       remove L from code
```

- **Categorization**
  - **Low**, **Medium**, and **High** impact code lines

Evaluation
Linked-List Based Set



Evaluation

- ❖ The impact of psyncs in machines with existing NVM technology is negligible

- ❖ A low-cost flush is applied either on a private variable stored in NVM or in newly-allocated data that has not yet become shared.

- ❖ A flush that incurs high performance penalty is executed on a shared variable (cache line) which is accessed by many threads, as such flushes will result in a high number of cache misses.

- ❖ The paper provides reasons that different flushes incur different performance costs.

Challenge IV

WHEN IS RECOVERABLE CONSENSUS HARDER THAN CONSENSUS?

Consensus

Each process has an input value and must output a value.

Consensus Problem

Validity: Each output is the input of some process
Agreement: No 2 outputs differ
Termination: If a process takes enough steps without crashing,it outputs a value

Recoverable Consensus Problem (RC) [Golab, SPAA 2020]

Validity: Each output is the input of some process
Agreement: No 2 outputs differ (including 2 outputs of 1 process)
Progress: If a process takes enough steps between crashes, it outputs a value

Consensus Hierarchy

Consensus Number, cons(T)

Maximum number of processes that can solve wait-free consensus using objects of type T and registers tolerating permanent crashes

Recoverable Consensus (RC) Number, rcons(T)

Maximum number of processes that can solve recoverable consensus using objects of type T and registers tolerating independent crash-recovery failures

System-wide failures $\Rightarrow$ simultaneous RC number

Herlihy's Universality Result

Conventional crash-stop failure model

A type T can be used (with registers) to obtain a wait-free implementation of all object types in a system of n processes if and only if cons(T) is at least n.

Crash-Recovery Failure Model
(both system-wide and independent)

Universality result carries over to the model with crashes and recoveries, using RC in place of consensus.

[Berryhil, Golab, Tripunitara, OPODIS'15]

System-Wide Crash-Recovery Model

- Recoverable consensus is solvable among n processes using objects of type T and registers if and only if cons(T) is at least n.

  [Golab, SPAA’20, Delporte-Gallet et al., PODC’22]

Independent Crash-Recovery Model

rcons(T) $\leq$ cons(T)

- Any RC algorithm also solves consensus.
- So RC is at least as hard as consensus.

Independent Crash Recovery Model

Is RC (much) harder than consensus?

Can rcons(T) be (much) smaller than cons(T)?

Delporte-Gallet, Fatourou, Fauconier & Ruppert, PODC 2022

- Focused on **readable objects**
- Defined **n-recording property** of shared object types.

Theorem 1 (Sufficient Condition)

If a deterministic readable type T is n-recording, then objects of type T, together with registers, can be used to solve recoverable consensus for n processes.

Theorem 2 (Necessary Condition)

If a deterministic readable type T can be used, together with registers, to solve recoverable consensus for n processes, then T is (n-1)-recording.

Open Problems

- Is rcons(T) << cons(T) for some non-readable type T?
- Close gap between necessary and sufficient condition.
  - First step: Is 2-recording necessary for solving 2-process RC?

NVM: Re-shaping the traditional memory hierarchy

- Models, performance metrics, and analysis patterns may have to be re-developed
- Assumptions that were considered fundamental in the past may now vanish
- Standard algorithmic design choices may have to be re-thought
- Well-known trade-offs may now diminish.

Thank You!

QUESTIONS?

http://www.ics.forth.gr/~faturu/
faturu@csd.uoc.gr

UNIVERSITY OF CRETE

FORTH
Foundation for Research & Technology - Hellas