Efficient Hardware Primitives for Immediate Memory Reclamation in Optimistic Data Structures

Ajay Singh

(ICS FORTH, Greece)

Trevor Brown

University of Waterloo

Michael Spear

Lehigh University

Greece 2.0 NATIONAL RECOVERY AND RESILIENCE PLAN

Funded by the European Union NextGenerationEU

Funded HAR.S.H. (project no. ΥΠ3ΤΑ-0560901), which is carried out within the framework of the National Recovery and Resilience Plan "Greece 2.0" with funding from the European Union – NextGenerationEU.

HAR.S.H. HARdware-aware Similarity searH

Modern Data Structures

- Follow optimistic synchronization or non blocking paradigms.
- Permit higher parallelism.
- Widely adopted in open-source software (e.g., Meta's Folly, Linux Kernel).

Key Property: Unsynchronized Reads

- Threads can read from a shared memory location while it is being concurrently modified.

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

“

Unsynchronized Reads/Traversals in concurrent data structures **enable high scalability** but **Complicate Memory Management!**

”

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Example: LockFree Stack [RK Treiber, IBM, 1986]

```
class Node{int data; Node* next;};
Node* Top = nullptr;
```

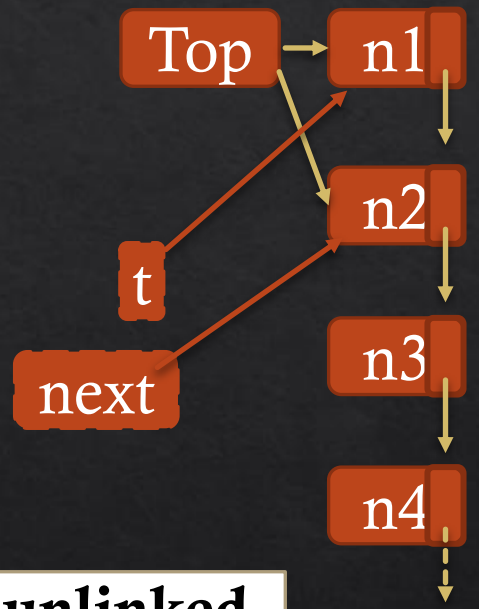
```
void push (int data) {
    Node* node = new Node(data);
    while (true) {
        Node* t = Top;
        node->next = Top;
        if (CAS(&Top, t, node))
          break;
    }
}
```

```
int pop () {
    while (true) {
        Node* t = Top;
        if (t == nullptr) return EMPTY;

        Node* next = t->next;
        if (CAS(&Top, t, next)) {
            int res = t->data
            delete t; // ???
            return res;
        }
    }
}
```

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Pop(): LockFree Stack [RK Treiber, IBM, 1986]

```
int pop () {
    while (true) {
        Node* t = Top;
        if (t == nullptr) return EMPTY;

        Node* next = t->next;
        if (CAS(&Top, t, next)) {
            int res = t->data
            delete t; // ???
            return res;
        }
    }
}
```



Works as long as the unlinked node is not reclaimed

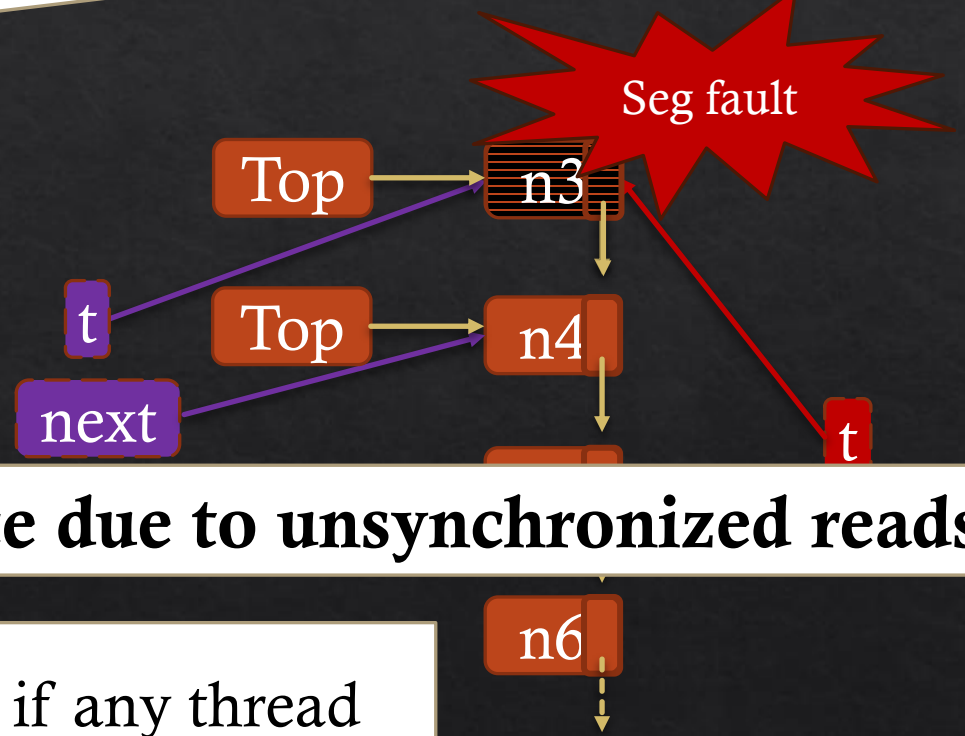
Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Problem: Read-Reclaim Race

```
int pop () {
    while (true) {
T1      Node* t = Top;
        if (t == nullptr) return EMPTY;

T1      Node* next = t->next;          T2
T1      if (CAS(&Top, t, next)) {
            int res = t->data
T1          delete t; // ???
T1          return res;
        }
    }
}
```

T2

Reader: does not know if any thread could concurrently free the node it is accessing.



read-reclaim race due to unsynchronized reads!

Reclaimer: does not know if any thread can access the node it is trying to free.

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Solution: Safe Memory Reclamation (SMR)

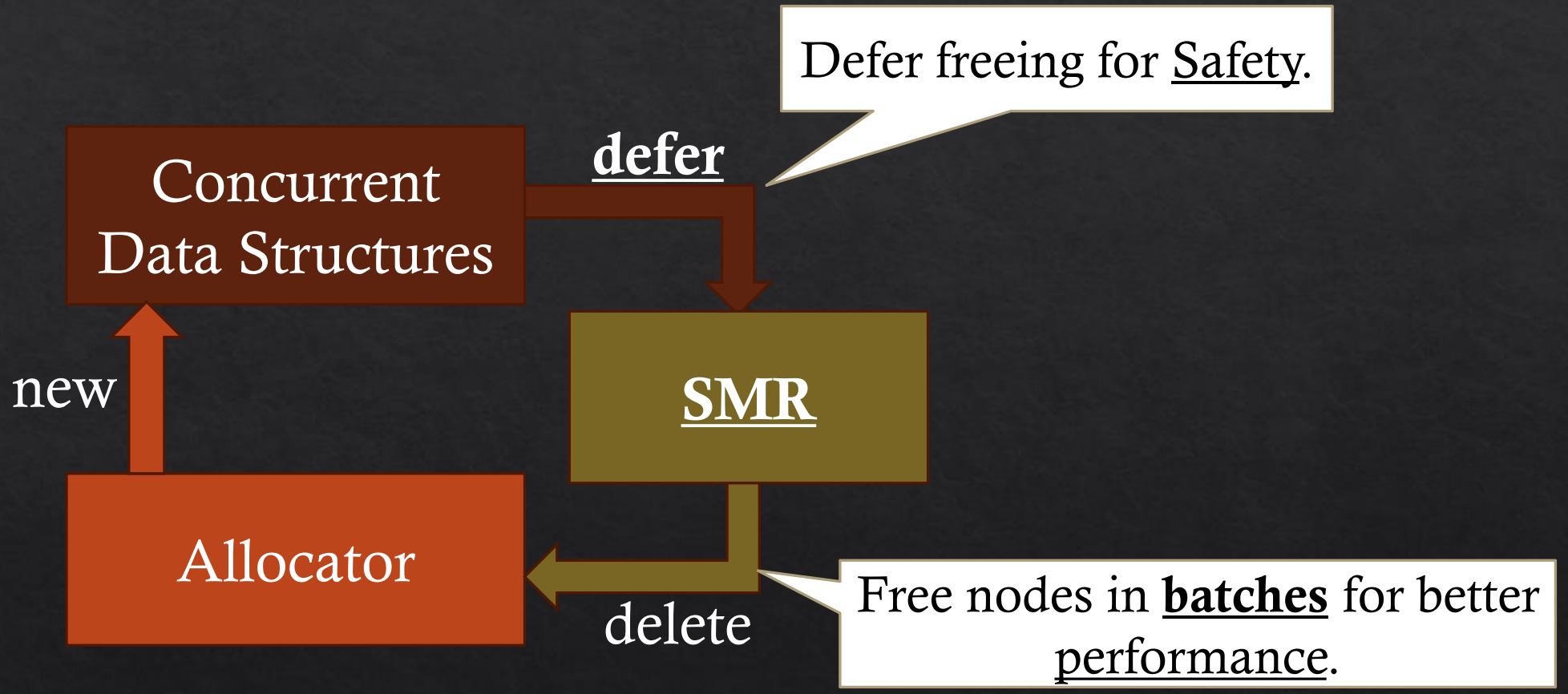
Synchronize readers and **reclaimers** to **decide a safe time to reclaim** unlinked nodes thereby resolving errors due to read-reclaim races.

Readers learn whether a node is safe to access i.e. will not be concurrently freed.

Reclaimers learn whether the node they have unlinked is safe to free i.e. no thread holds a pointer to the node.

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

SMR: Key Aspect



Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Problem with Deferred Freeing & Batching

- Memory footprint vs performance trade-off.
- Batching interferes with underlying allocator performance (e.g., jemalloc). [Amort. Freeing: Kim, Trevor, Ajay in PPOPP '24]
- Impedes memory overcommitment and ballooning in virtualized data centers.

Deferred reclamation has several downsides can we reclaim immediately and yet be fast?

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Cache Level Events Precede a Read-Reclaim Race

t1 `Node* t = Top;` t2

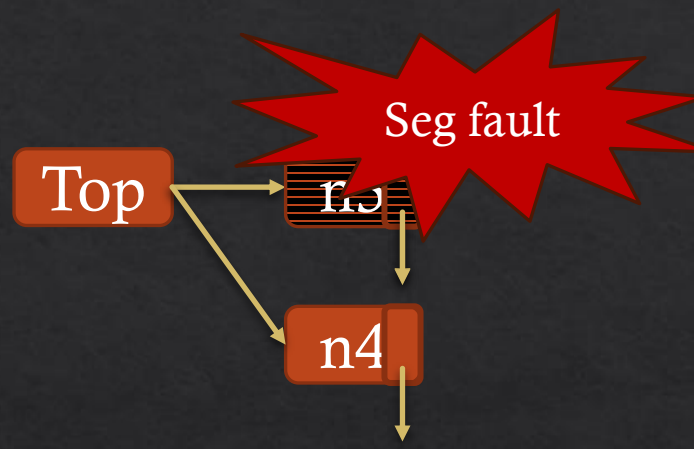
`...`

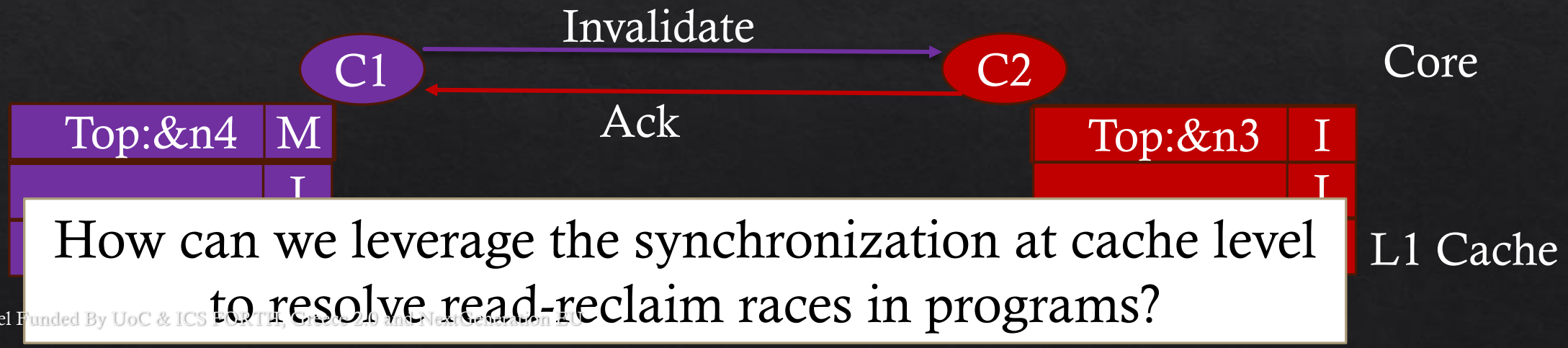
`Node* next = t->next;` t2

t1 `CAS(&Top, t, next)`

`...`

t1 `delete t;`





How can we leverage the synchronization at cache level to resolve read-reclaim races in programs?

Travel Funded By UoC & ICS FORTH, GREECE 2019 and Next Generation EU

Conditional Access: Hardware-Software Codesign

- **Capture** the cache-level synchronization events through a simple hardware extension.

- **Expose** these events to programmer level through memory access instructions.

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

(I) Capture Cache Events

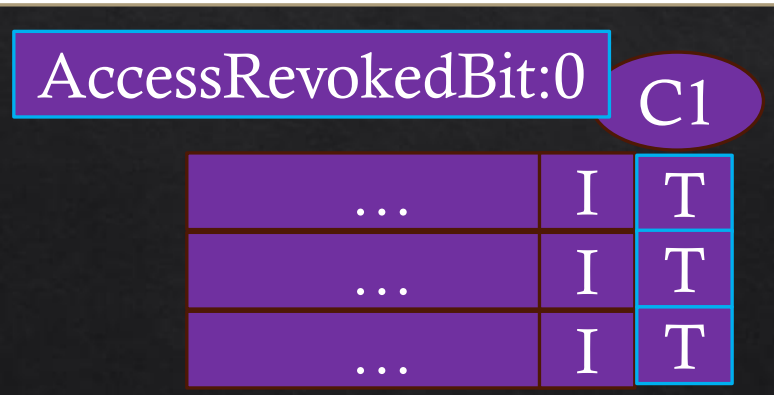
Tag Set: Tagged when accessed first time.

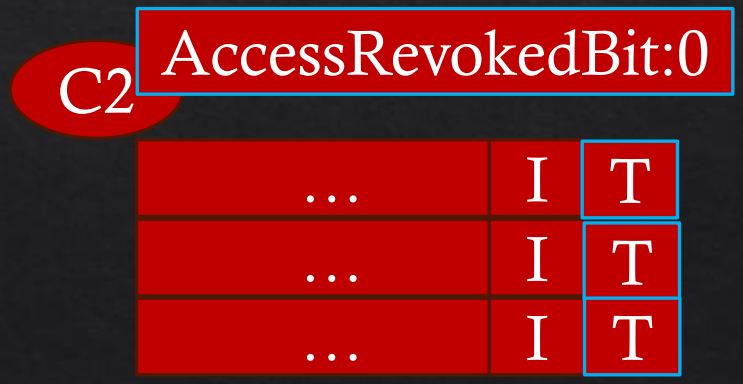
- subsequent invalidation events indicate a possible read-reclaim race.
- Enables **monitoring** of possible read-reclaim race to the lines in Tag Set.

TagBit: One bit per cache line.

AccessRevokedBit:

- Initially clear & Set when any of the tagged lines are invalidated or leave the cache.
- Enables **Recording** of possible read-reclaim race events for tagged cache lines.





Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

(II) Expose Cache Events to Programmers

cRead addr, dest
Atomically
- Add addr to TagSet if not in TagSet.
- **If** AccessRevokedBit is set, skip the load, and set a processor flag to indicate error to program.
- **Else** do a normal load.

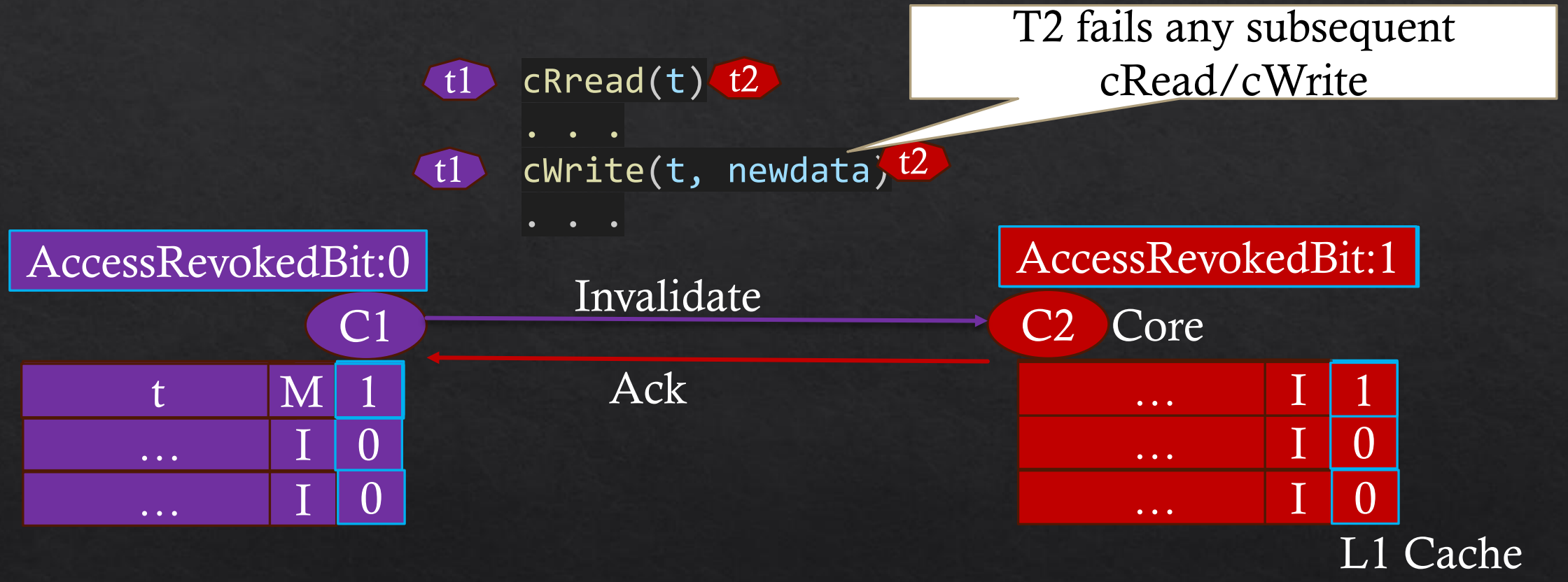
cWrite addr, v
Atomically
- **If** AccessRevokedBit is set or addr ∉ TagSet. Skip store and set a processor flag.
- **Else** do a normal store.

untagOne addr
Remove Address from TagSet.

untagAll
Clear TagSet and AccessRevokedBit.

Tpe Greece 2.0 and NextGeneration EU

Conditional Access in Action



Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Programmer’s view

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Using Conditional Access

```
int pop () {
    while (true) {
     Node* t = Top;
     if (t == nullptr) return EMPTY;

     Node* next = t->next;
      if (CAS(&Top, t, next)) {
       int res = t->data
       delete t;
       return res;
      }
   }
}
```

```
#define CACHECK if CAFAIL then untagAll(); goto
retry;

int pop () {
    retry:
      Node* t = CREAD(Top); CACHECK;
      if (t == nullptr) untagAll();return EMPTY;

      Node* next = CREAD(t->next); CACHECK;
      CWRITE(&Top, next); CACHECK;
      int res = t->data
      delete t;
      untagAll();
      return res;
}
```

Replace and Evaluate

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Conditional Access Prevents Read-Reclaim Race

```
#define CACHECK if CAFAIL then untagAll(); goto
retry;

int pop () {
    retry:
rec   Node* t = CREAD(Top); CACHECK;   rd1  rd2
      if (t == nullptr) untagAll();return EMPTY;

      Node* next = CREAD(t->next); CACHECK;   rd2
rec   CWRITE(&Top, next); CACHECK;   rd1
      int res = t->data
      delete t;
      untagAll();
      return res;
}
```

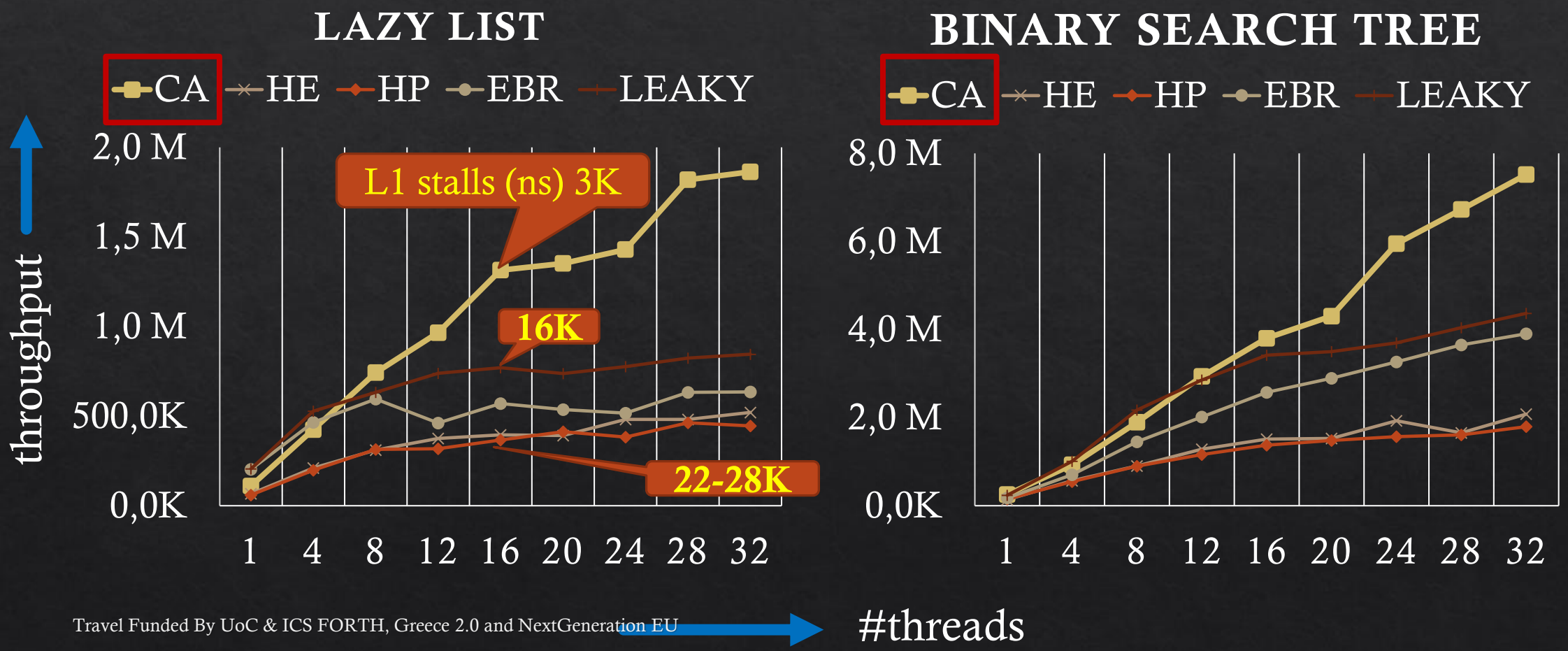
Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Experimental Evaluation

- Prototyped Conditional Access on Graphite multicore simulator [Anant Agarwal et. al. HPCA 10].
- **Safe memory reclamation algorithms:** hazard eras, hazard pointers, epoch-based reclamation, interval-based reclamation, leaky implementation as pseudo baseline.
- **Data Structures:** linked list, search tree, hash table and stack.
- **Simulator Configuration:** MESI/ MSI coherence protocol, 32K L1 cache, 256K L2 cache size, Line size 64Bytes, One threads pinned per core.

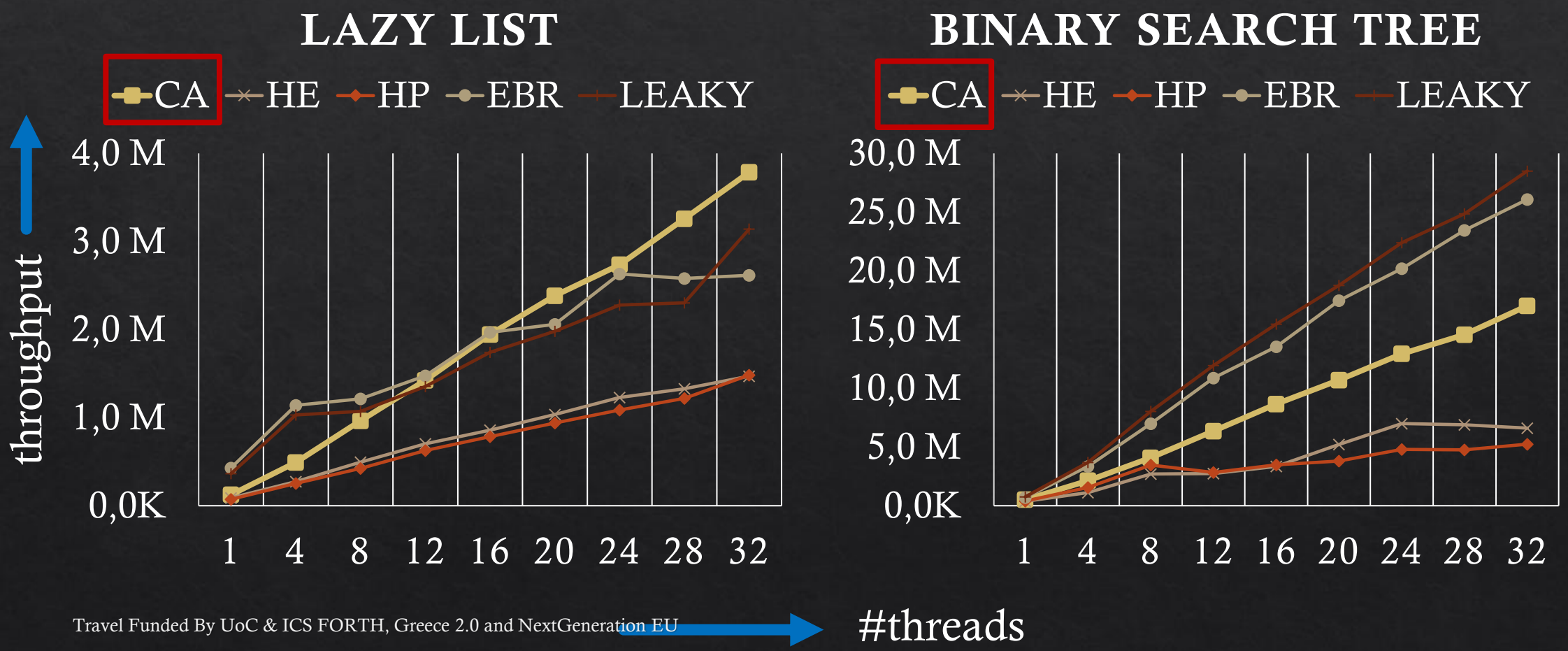
Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

List and Search Tree with 100% updates



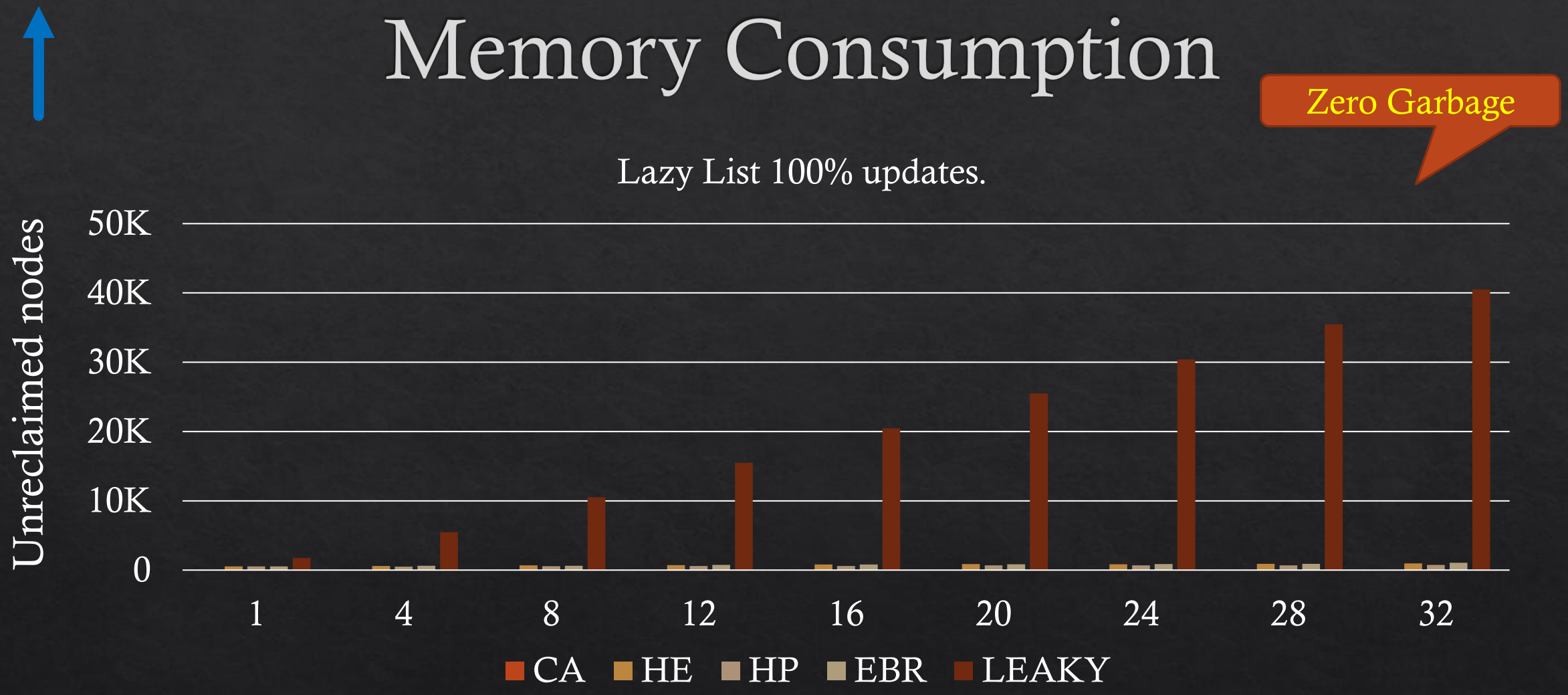
Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

List and Search Tree with read-only workload



Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Memory Consumption



Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Conclusion

- Conditional Access: an alternative paradigm to reclaim memory.
- Key Idea: Leverage hardware-software codesign to capture cache events and expose them to programmers to enable safe memory reclamation.
- Sequential data structure like ideal memory footprint along with concurrent data structure like throughput.
- In paper:
  - Usage examples with more complex data structures and proof of correctness.
  - Novel designs of concurrent data structures using the new instructions
  - More experiments, limitations and remedies.

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

CA: Conditional Access

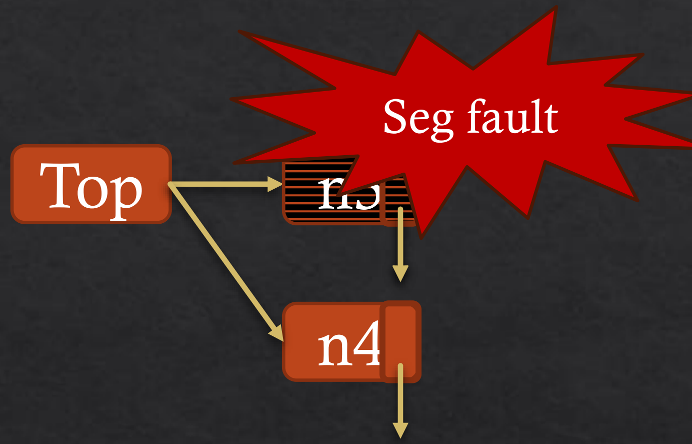
Deferred Reclamation → Immediate Reclamation

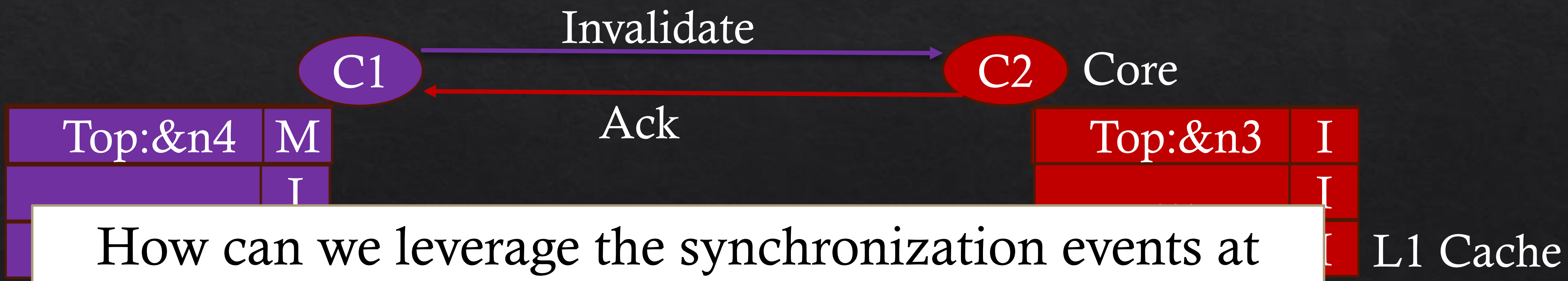
- Deferred Freeing Paradigm: Defer freeing for safety and free in batches for performance.
- Problems: Memory footprint vs performance trade-off, hinders memory overcommitment, triggers slow path in allocators [PPOPP ‘24].
- **Goal:** Can we reclaim immediately and yet be as fast as deferred reclamation based SMRs?
  - Turn to hardware-software codesign.

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Cache Events Precede Read-Reclaim Races

```
T1  Node* t = Top;               T2
    ...
    Node* next = t->next;        T2
T1  CAS(&Top, t, next)
    ...
T1  delete t;
```





How can we leverage the synchronization events at cache level to resolve read-reclaim races in programs?

Conditional Access: (I) Capture & (II) Expose

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

(I) Capture Cache Events

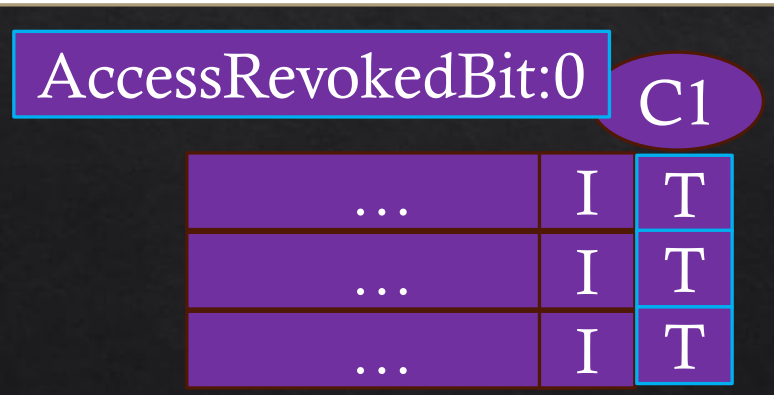
Tag Set: Tagged when accessed first time.

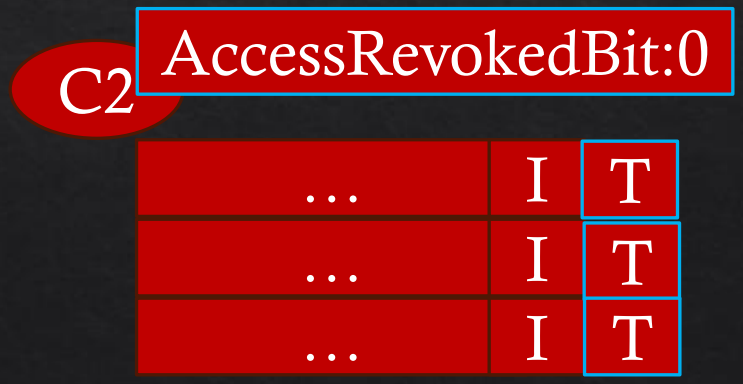
- subsequent invalidations indicate a possible read-reclaim race.
- Enables **monitoring** of possible read-reclaim race to the lines in Tag Set.

TagBit: One bit per cache line (or TCAM).

AccessRevokedBit:

- Initially clear & Set when any of the tagged lines are invalidated or leave the cache.
- Enables **Recording** of possible read-reclaim race events for tagged cache lines.





Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

(II) Expose Cache Events to Programmers

cRead addr, dest
Atomically
- Add addr to TagSet if not in TagSet.
- **If** AccessRevokedBit is set, skip the load, and set a processor flag to indicate error to program.
- **Else** do a normal load.

cWrite addr, v
Atomically
- **If** AccessRevokedBit is set or addr ∉ TagSet. Skip store and set a processor flag.
- **Else** do a normal store.

untagOne addr
Remove Address from TagSet.

untagAll
Clear TagSet and AccessRevokedBit.

Tpe Greece 2.0 and NextGeneration EU

Main Results [IPDPS ‘23]

- **Conditional Access:** Simple hardware extension to **capture cache events** and **expose them to programmers** through a set of new memory access instructions.
- Key Idea: Leverage hardware-software codesign to enable **immediate yet fast** safe memory reclamation.
- Prototyped Conditional Access on Graphite multicore simulator [Anant Agarwal et. al. HPCA 10] and Benchmark.

Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU

Conclusion

Neutralization Paradigm for SMR

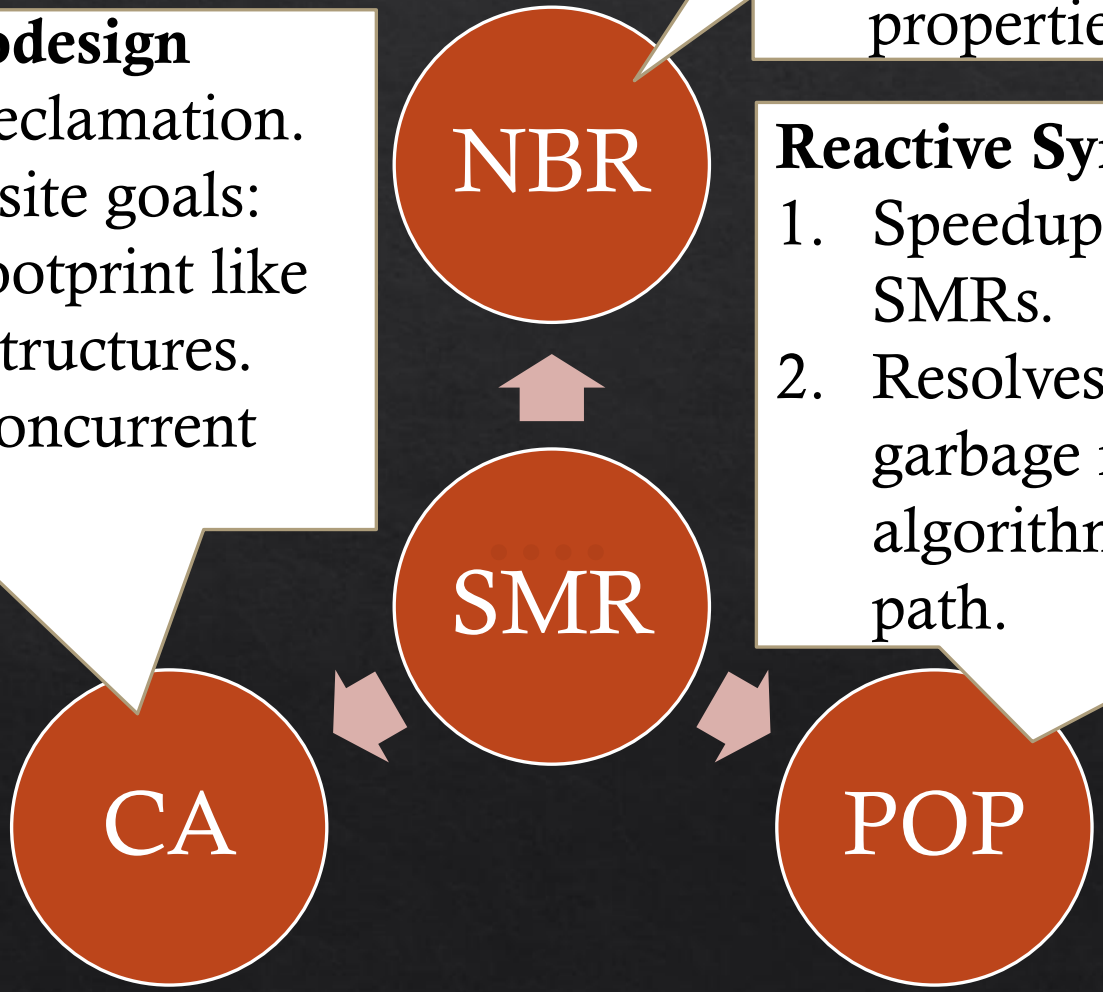
1. Efficiently reclaim memory while bounding the amount of garbage.
2. Yet retaining several desirable properties.

Hardware-Software Codesign

1. Allows immediate reclamation.
2. Balancing two opposite goals:
   1. Ideal Memory footprint like sequential data structures.
   2. Scalability like concurrent data structures

Reactive Synchronization Paradigm

1. Speedup Hazard Pointer like SMRs.
2. Resolves issue of unbounded garbage in EBR with a dual-mode algorithm, avoiding fast-past/slow-path.



Greece 2.0 NATIONAL RECOVERY AND RESILIENCE PLAN
Funded by the European Union NextGenerationEU
Travel Funded By UoC & ICS FORTH, Greece 2.0 and NextGeneration EU
HAR.S.H. HARdware-aware Similarity searH