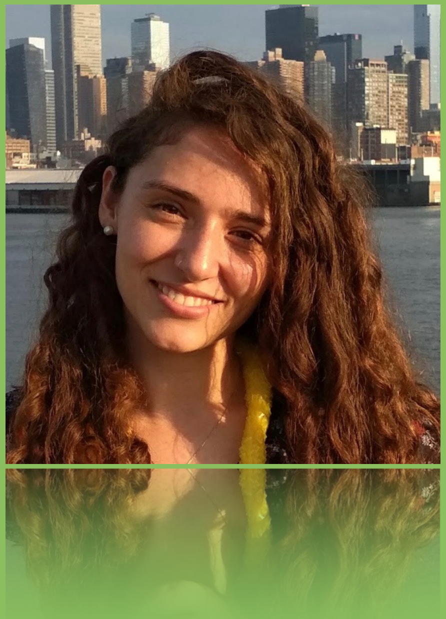


Michal Friedman

Maurice Herlihy

Virendra Marathe

Erez Petrank

Highlight Paper:

A Persistent Lock-Free Queue for Non-Volatile Memory (PPoPP'18)

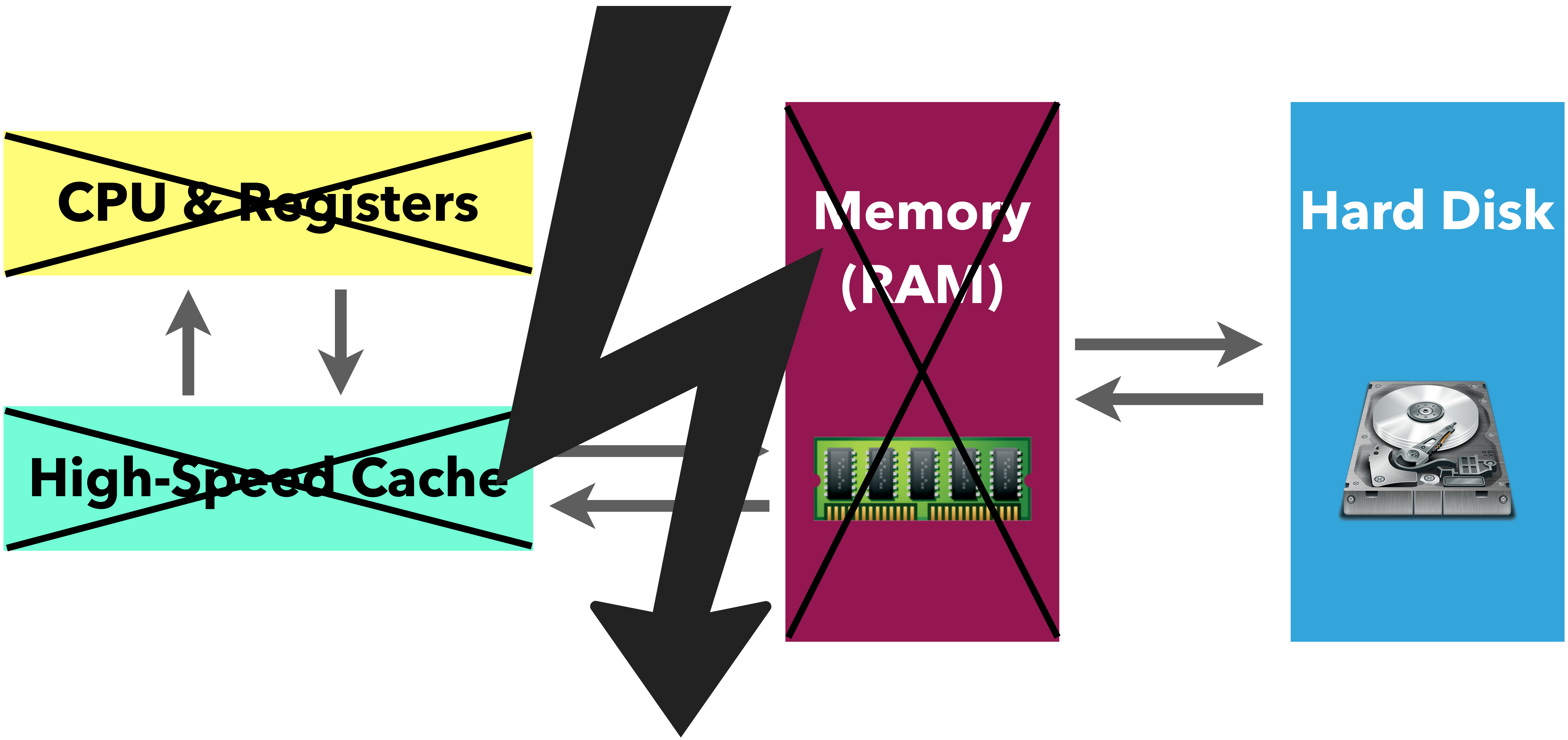
SYSTOR '19

THIS TALK

Concurrent Data Structure → Non-Volatile Byte-Addressable Memory

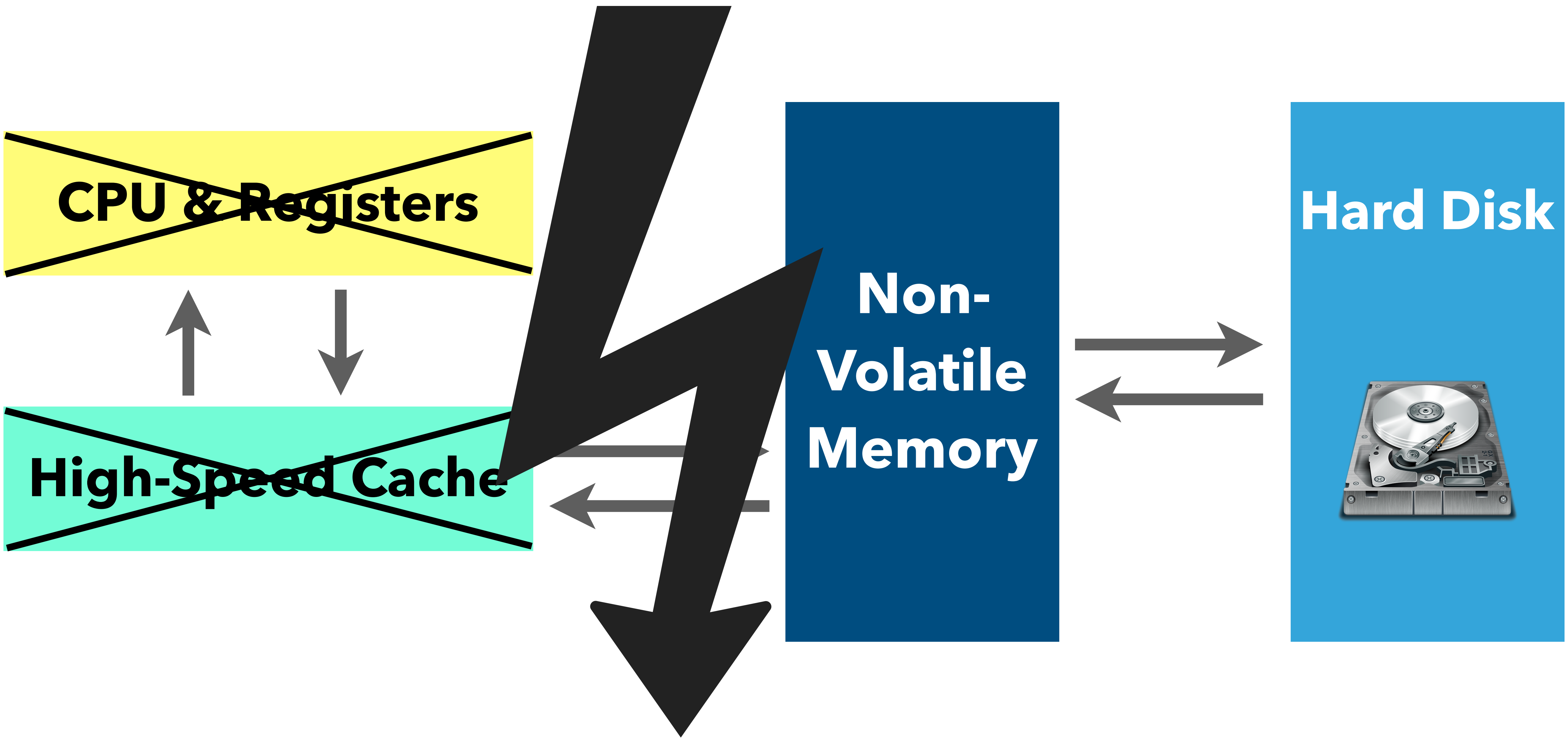
- Platform & Challenge
- Definitions
- Queue designs
- Evaluation

PLATFORM – BEFORE



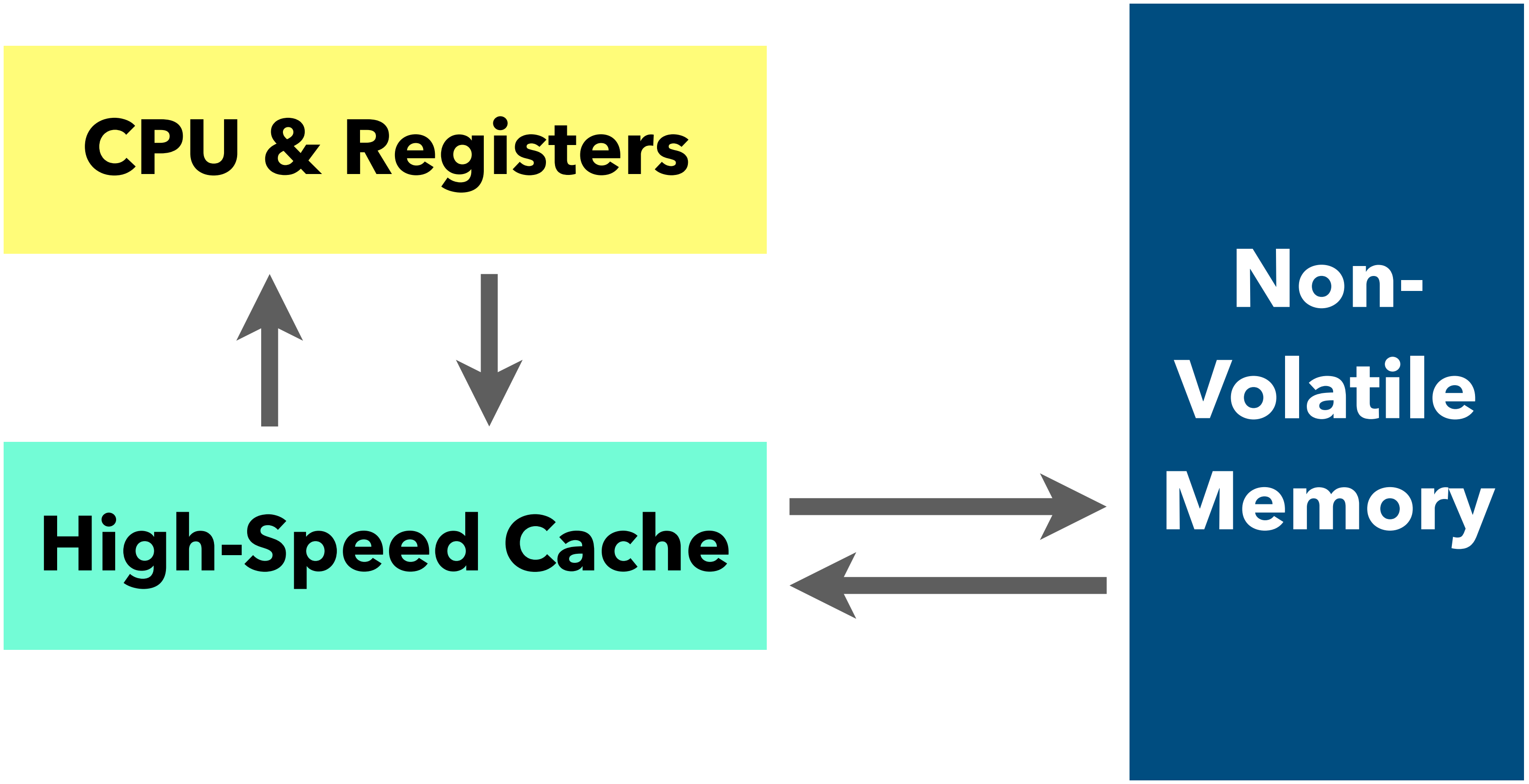
Upon a crash Cache and Memory content is lost

PLATFORM – AFTER



Upon a crash Cache content is lost

OPPORTUNITY



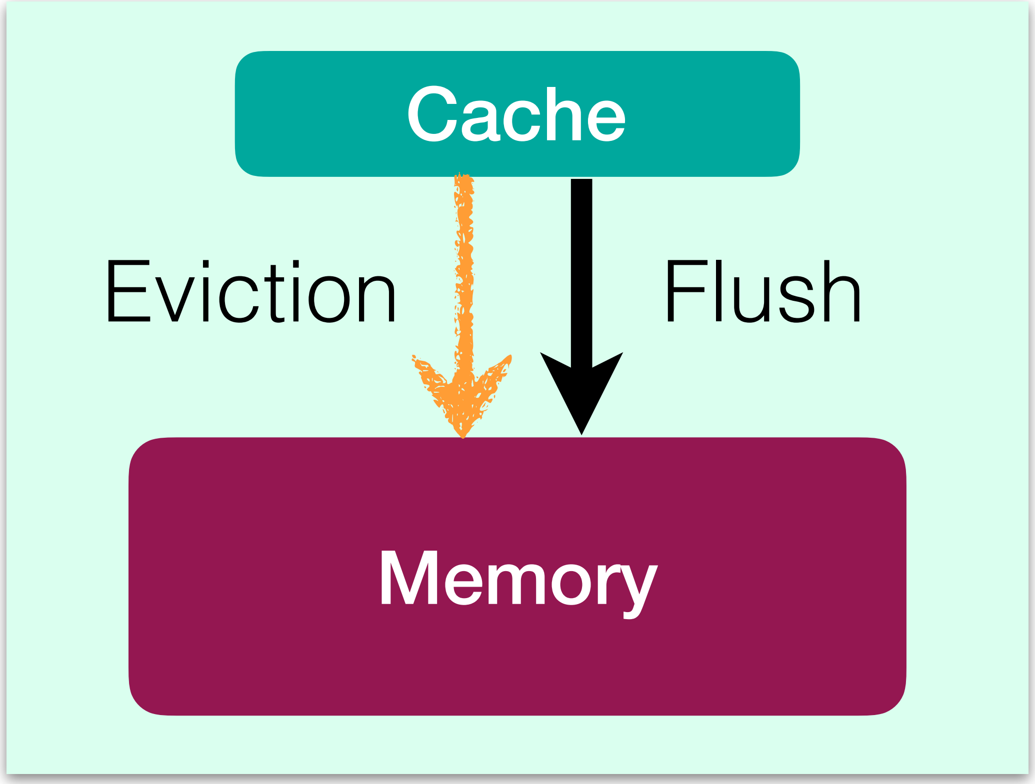
Instead of writing blocks to disk, make our normal data structures persistent!

MAJOR PROBLEM: ORDERING NOT MAINTAINED

- Write **x = 1**
- Write **y = 1**

Implicit eviction of y

- Flush &**x**
- Flush &**y**

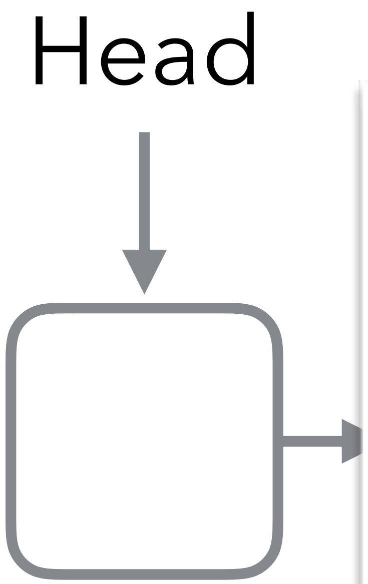


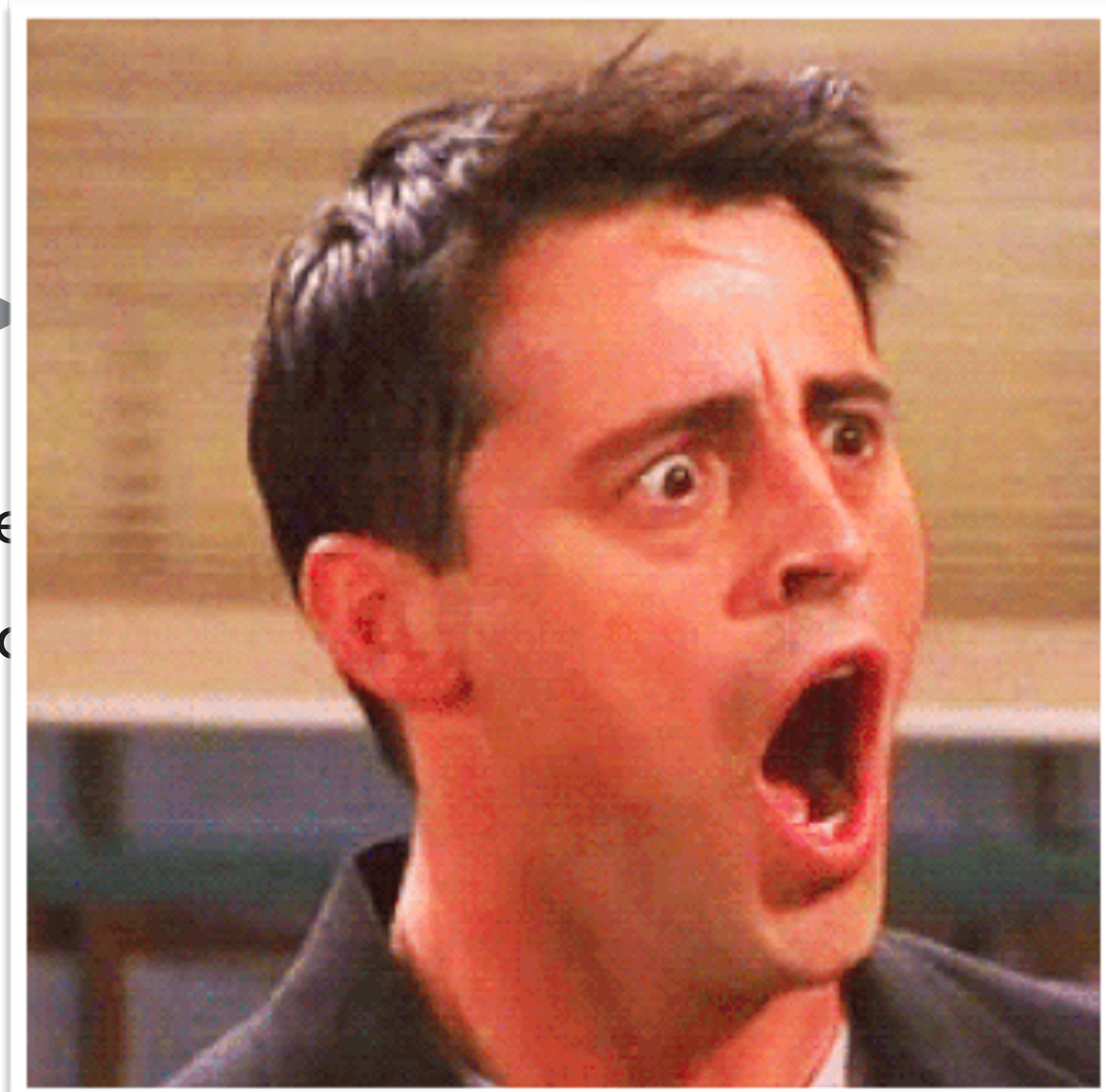
Due to implicit eviction:
Upon a crash, memory may contain **y = 1** and **x = 0**.

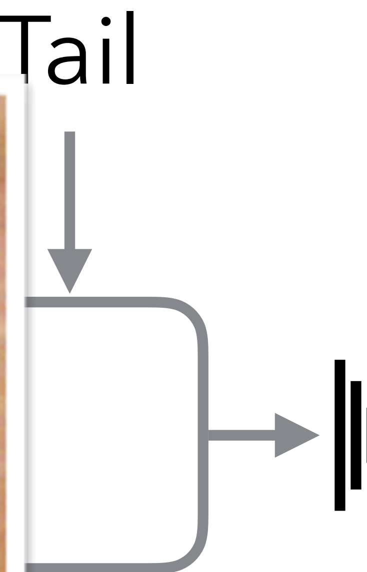
↓

O_2 can follow up on O_1, but only O_2 is reflected in the memory.

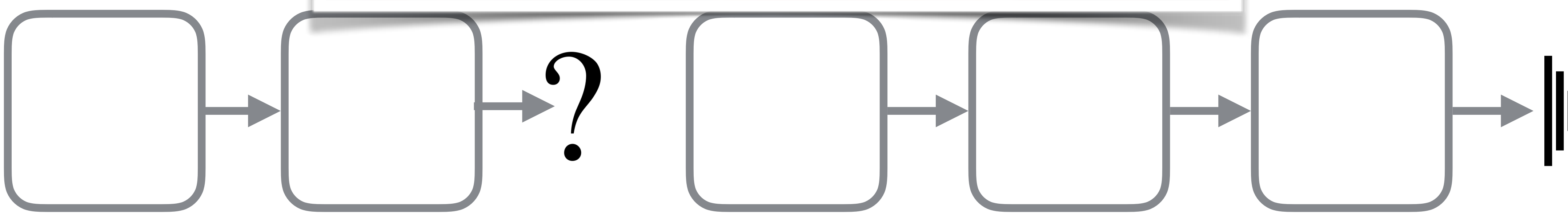
EXAMPLE



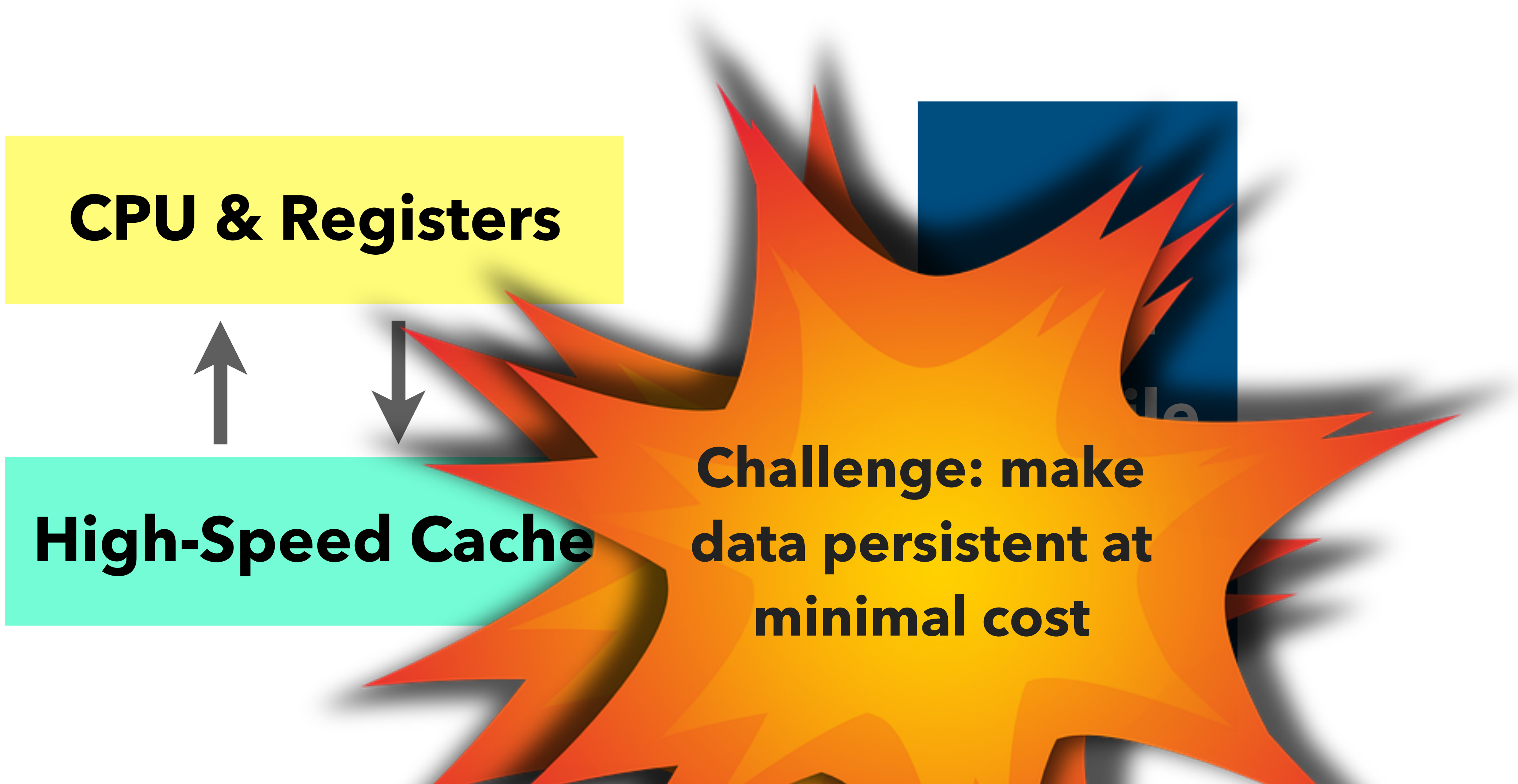




- Suppose eve his one pointer
- If a crash occ



CHALLENGE



Problem: Caches and registers are volatile.

- Usually don't care what's in the cache/memory
- Here we care!
- Flush some data to maintain consistency in memory - costly!

THE MODEL

- Main memory is non-volatile
- Caches and registers are volatile
- All threads crash together
  - New threads are created to continue the execution

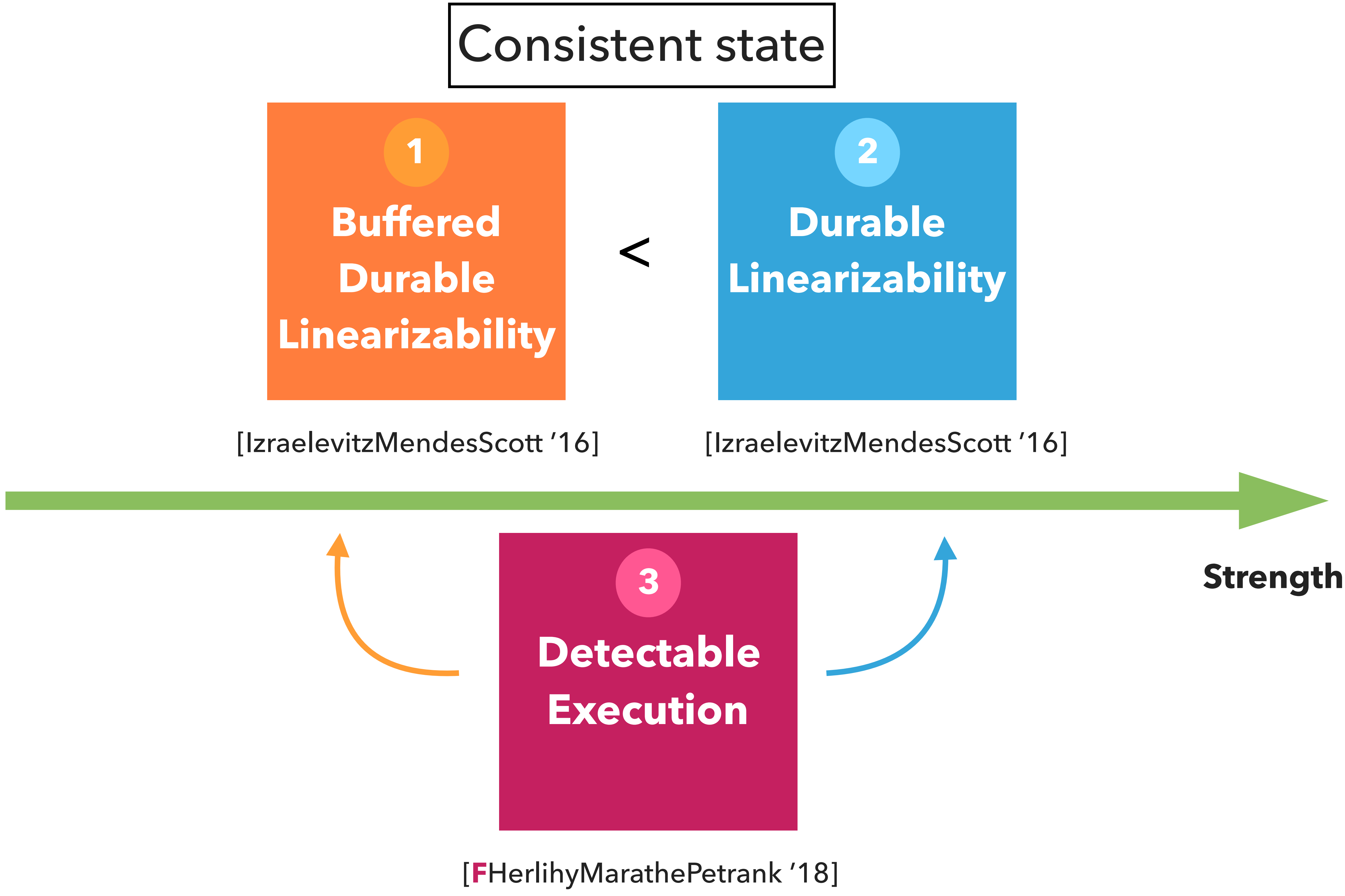
NEXT

- Definitions
- The queue designs
    - Surprisingly many details and challenges

LINEARIZABILITY

- [HerlihyWing '90]
  - Each method call should appear to take effect instantaneously at some moment between its invocation and response

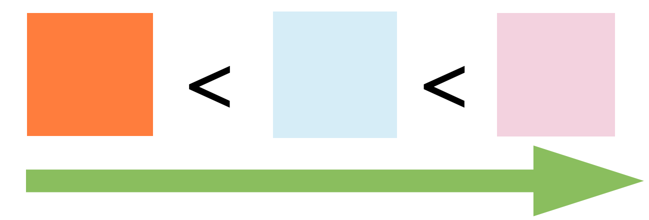
CORRECTNESS FOR NVM



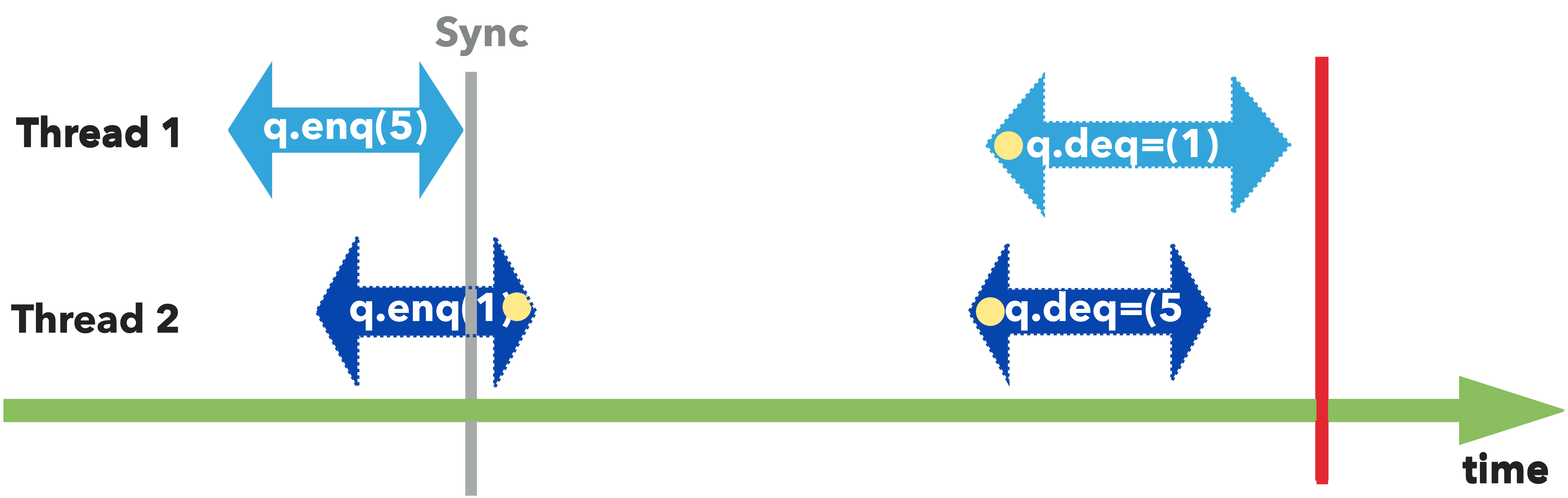
DURABLE LINEARIZABILITY

- [IzraelevitzMendesScott '16]
  - Operations completed before the crash are recoverable (plus some overlapping operations)
  - Prefix of linearization order

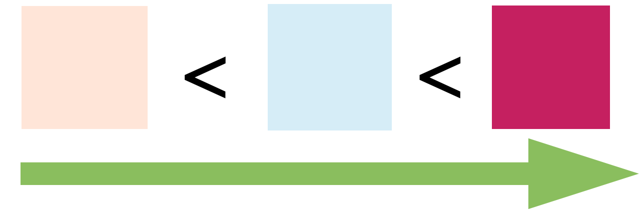
BUFFERED DURABLE LINEARIZABILITY



- [IzraelevitzMendesScott '16]
  - Some prefix of a linearization ordering
  - Support: a "sync" persists all previous operations



DETECTABLE EXECUTION



- [FHerlihyMarathePetrank ’18]
  - Even in durable-linearizability - no ability to determine completion
  - ***Detectable execution*** extends definitions:
    - Provide a mechanism to check if operation completed
    - Implementation example: a persistent log

THREE NEW QUEUE DESIGNS

- Three lock-free queues for non-volatile memory [**F**HerlihyMarathePetrank ’18]

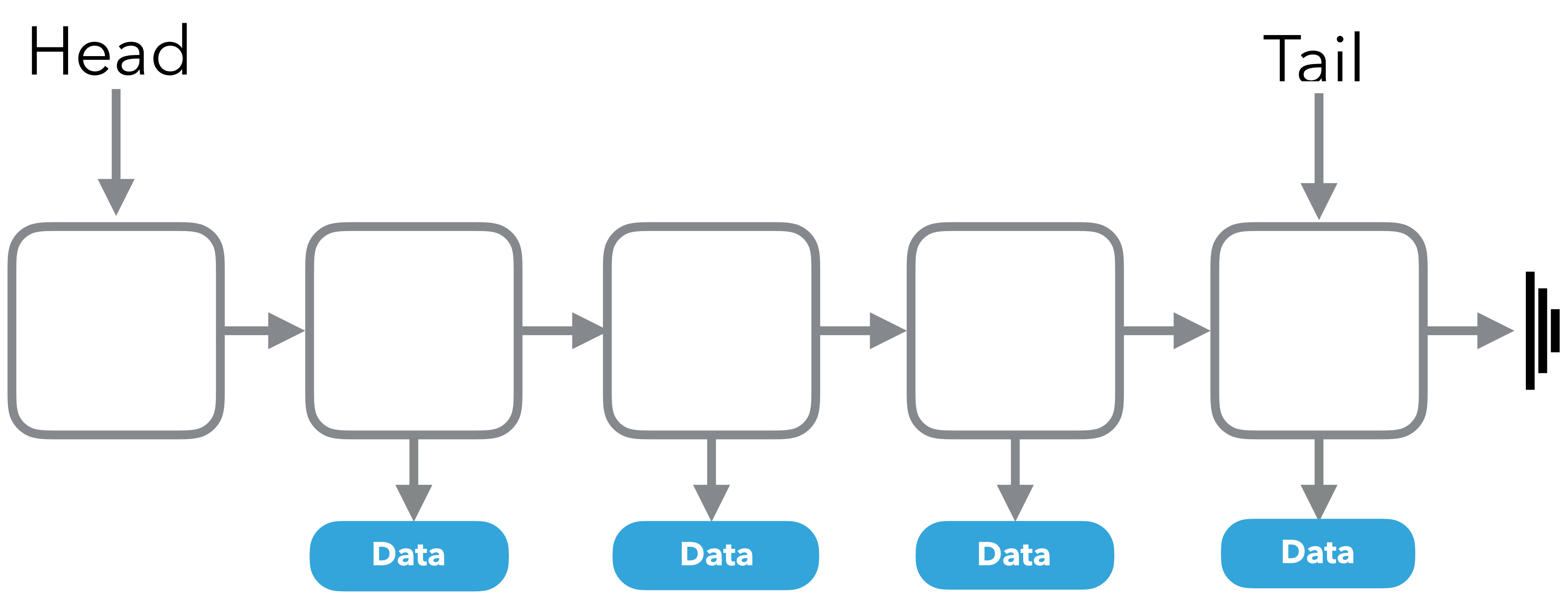
Relaxed	<	**Durable**	<	**Log**
A prefix of executed operations is recovered (**Buffered**)		All operations completed before the crash are recovered (**Durable**)		Durable + can tell if an operation recovered (**Detectable**)

- Based on lock-free queue [MichaelScott ’96]

- Design
- Evaluation

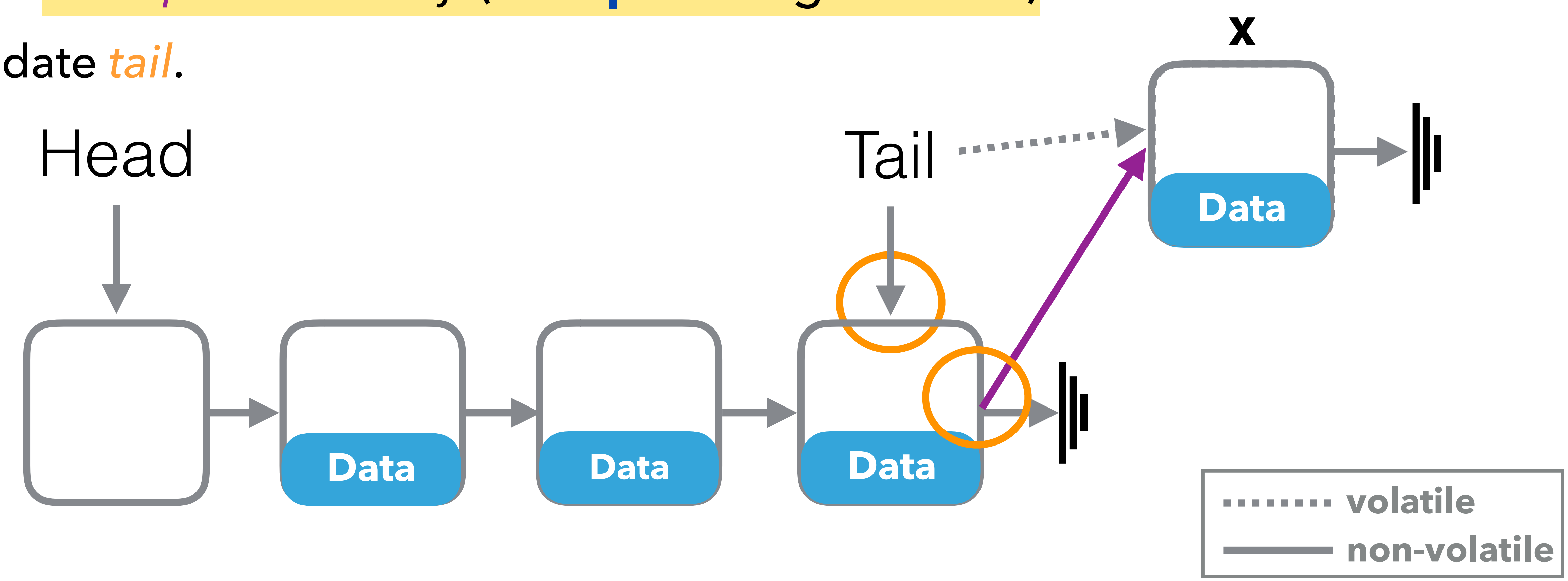
MICHAEL AND SCOTT'S QUEUE (BASELINE)

- A **Lock-Free** queue
- The base algorithm for the queue in java.util.concurrent
- A common simple data structure, but
- Complicated enough to demonstrate the challenges



DURABLE ENQUEUE

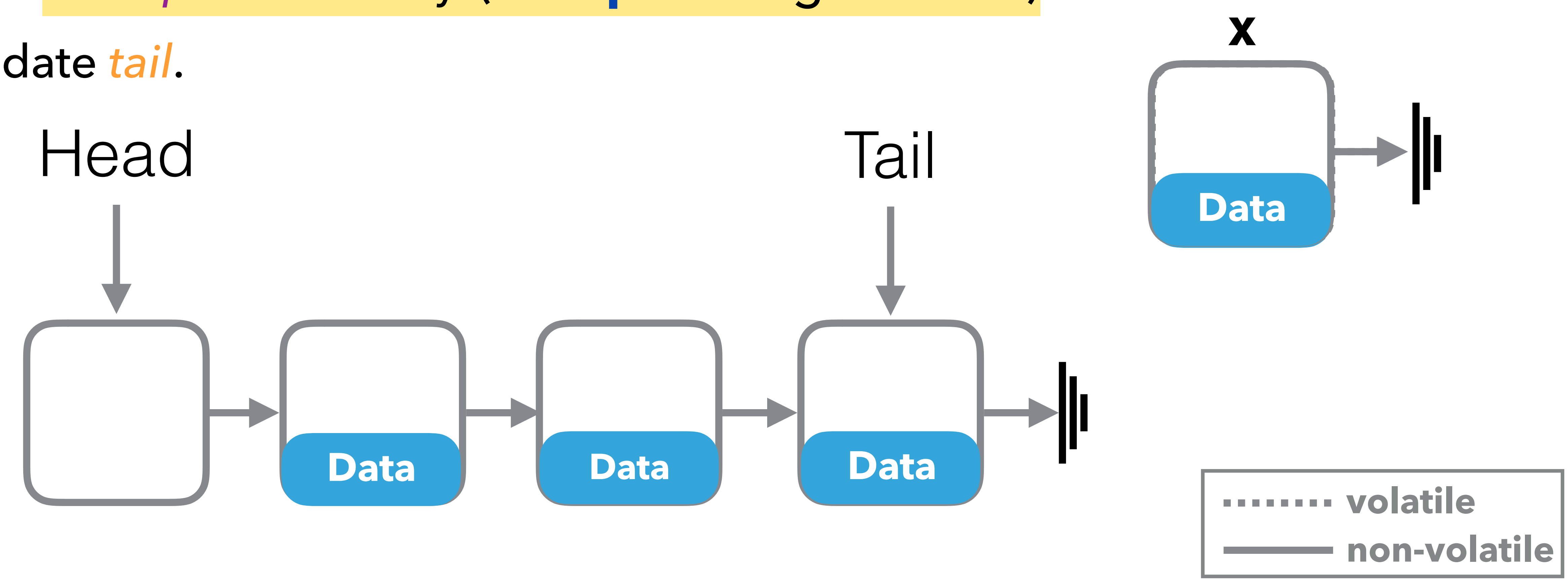
- Enqueue (data):
  1. Allocate a node with its values.
     1.a. Flush node content to memory. (**Initialization** guideline.)
  2. Read *tail* and *tail->next* values.
     2.a. Help: Update tail.
  3. Insert node to queue - CAS last pointer *ptr* point to it.
     3.a. Flush *ptr* to memory. (**Completion** guideline.)
  4. Update *tail*.



DURABLE ENQUEUE – MORE COMPLEX

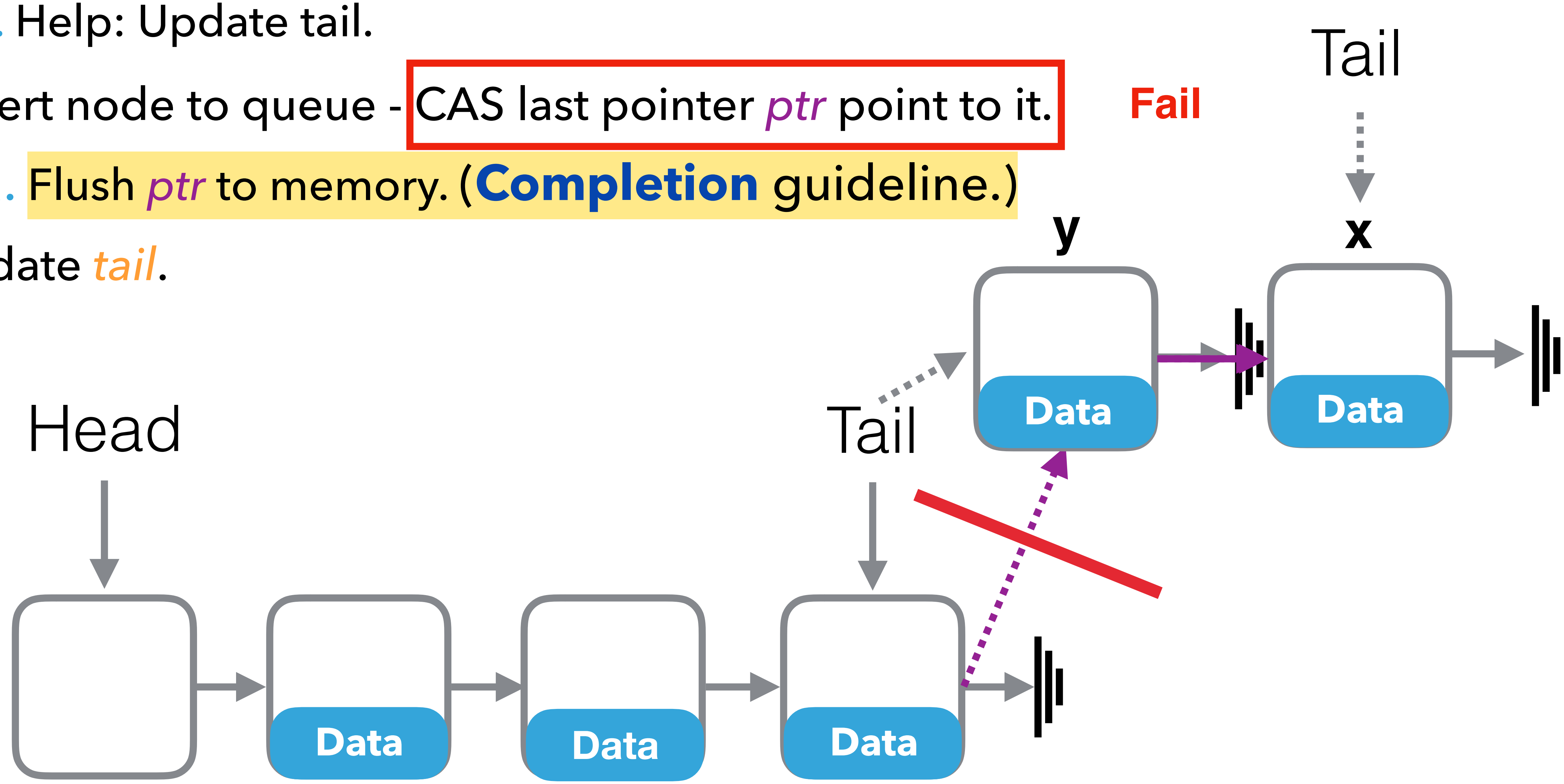
- Enqueue (data):
  1. Allocate a node with its values.
     1.a. Flush node content to mem
  2. Read *tail* and *tail->next* values.
     2.a. Help: Update tail.
  3. Insert node to queue - CAS last pointer *ptr* point to it.
     3.a. Flush *ptr* to memory. (**Completion** guideline.)
  4. Update *tail*.

For example, if this CAS fails due to concurrent activity, we need to be careful to maintain durable linearizability…



DURABLE ENQUEUE – MORE COMPLEX

- Enqueue (data):
  1. Allocate a node with its values.
     1.a. Flush node content to memory. (**Initialization** guideline.)
  2. Read *tail* and *tail->next* values.
     2.a. Help: Update tail.
  3. Insert node to queue - CAS last pointer *ptr* point to it.
     3.a. Flush *ptr* to memory. (**Completion** guideline.)
  4. Update *tail*.

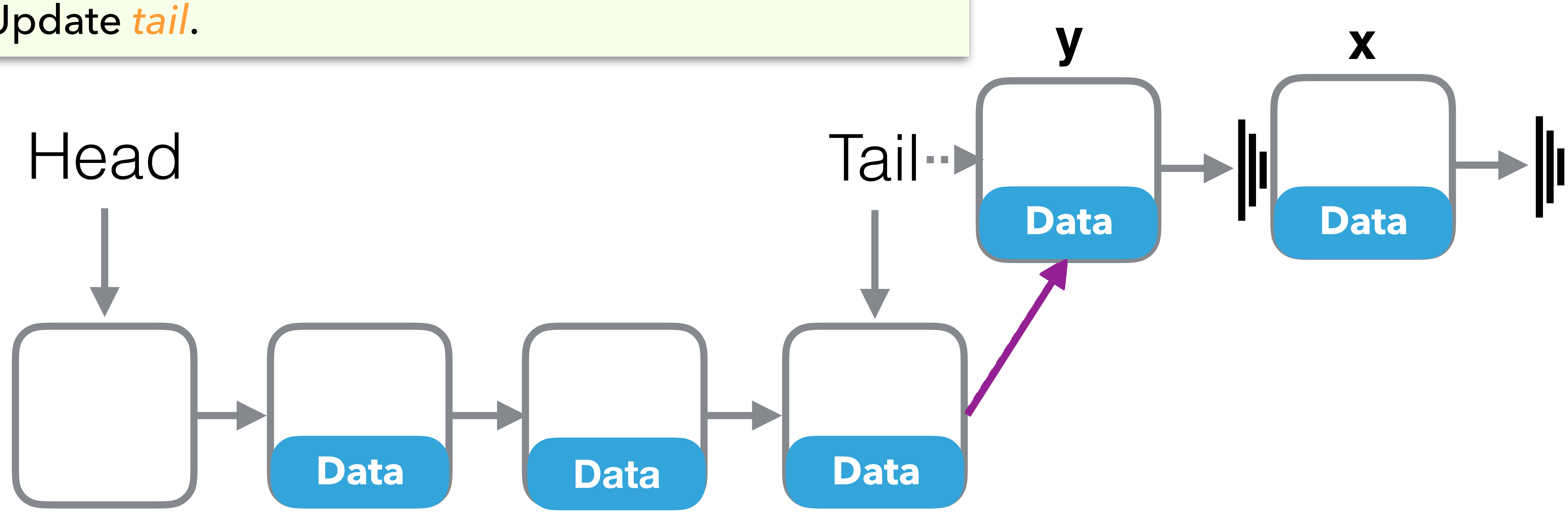


DURABLE ENQUEUE – MORE COMPLEX

- Enqueue (data):
  1. Allocate a node with its values.
     1.a. Flush node content to memory. (**Initialization** guideline.)
  2. Read *tail* and *tail->next* values.
     2.a. Help: Update tail.
  3. Insert node to queue - CAS last pointer *ptr* point to it.

Fail

- Complete (and persist) previous operation:
  5. Flush *ptr* to memory. (**Dependence** guideline.)
  6. Update *tail*.



RELAXED QUEUE

- Buffered Durable linearizable
- Challenge 1: Obtain snapshot at sync() time
- Challenge 2: Making sync() concurrent

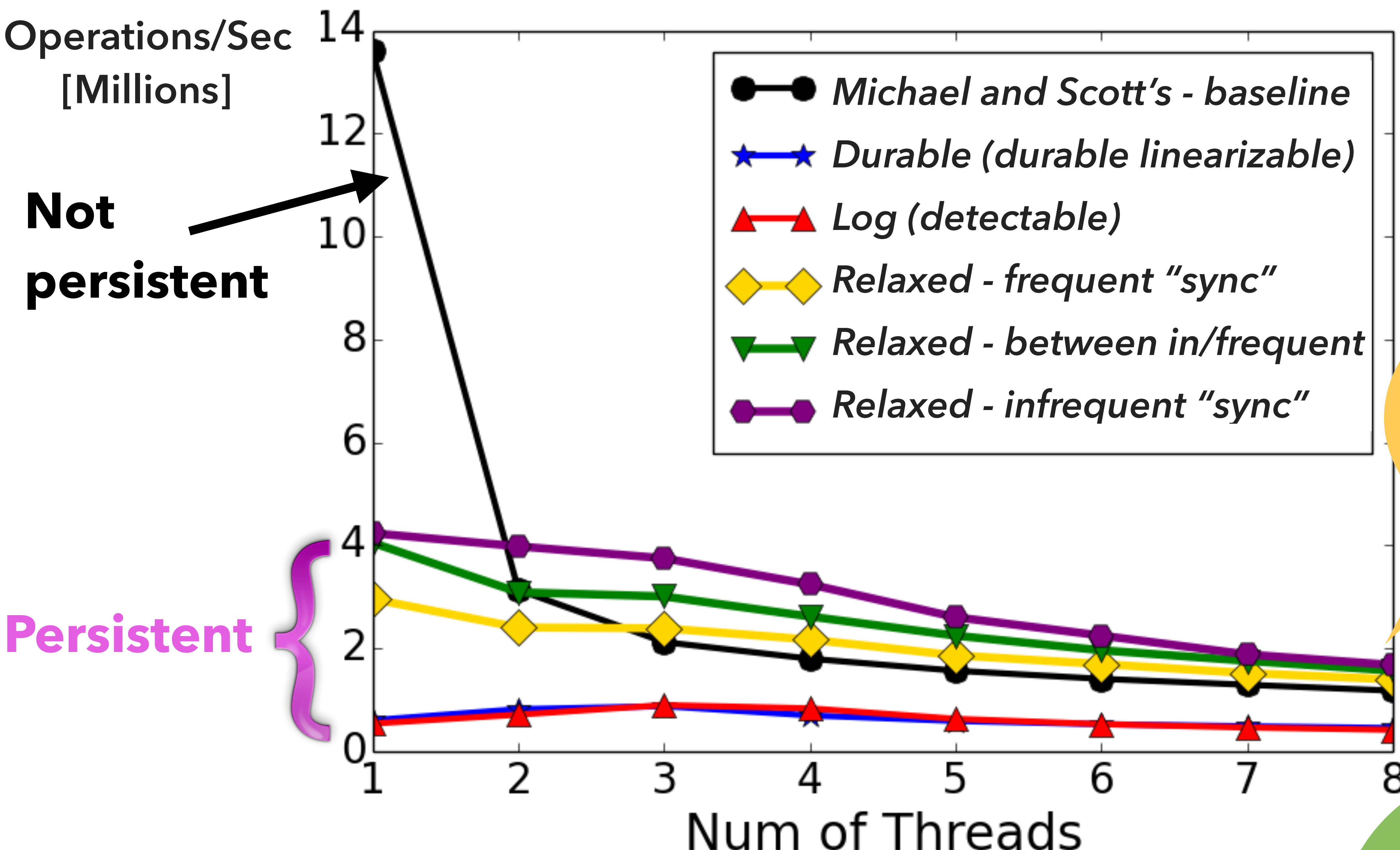
LOG QUEUE

- Durable linearizable
- Detectable execution
- Log operations
- More complicated dependencies and recovery

EVALUATION

- Compare the three queues: durable, relaxed, log and Michael and Scott’s queue
- Platform: 4 AMD Opteron(TM) 6376 2.3GHz processors, 64 cores in total , Ubuntu 14.04.
- Workload: threads run enqueue-dequeue pairs concurrently

EVALUATION – THROUGHPUT



Buffered durability **less** costly

Durability & detectable **costly.** **Similar** overhead

Implementation details:

- Frequent sync: every 10 ops/thread
- Infrequent sync: every 1000 ops/thread
- Queue initial size: 1 M

CONCLUSION

- A new definition: ***detectable*** execution
- ***Three lock-free queues*** for NVM: Relaxed, Durable, Log
- ***Guidelines***
- ***Evaluation***
    - Durability and detectability - similar overhead
    - Buffered durability is less costly