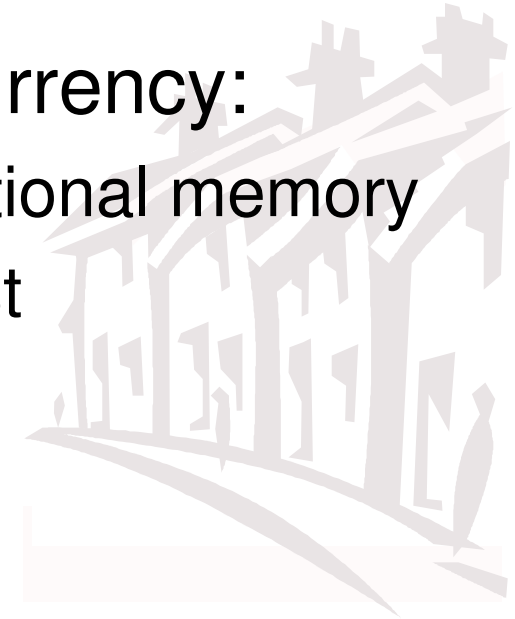
Highly-Concurrent Data Structures

Hagit Attiya and Eshcar Hillel
Computer Science Department
Technion

Talk Overview

- What are highly-concurrent data structures and why we care about them
- The concurrency of existing implementation techniques
- Two ideas for increasing concurrency:
  - Conflict management ⇨ Transactional memory
  - Locking order ⇨ Doubly-linked list



Data Items and Operations

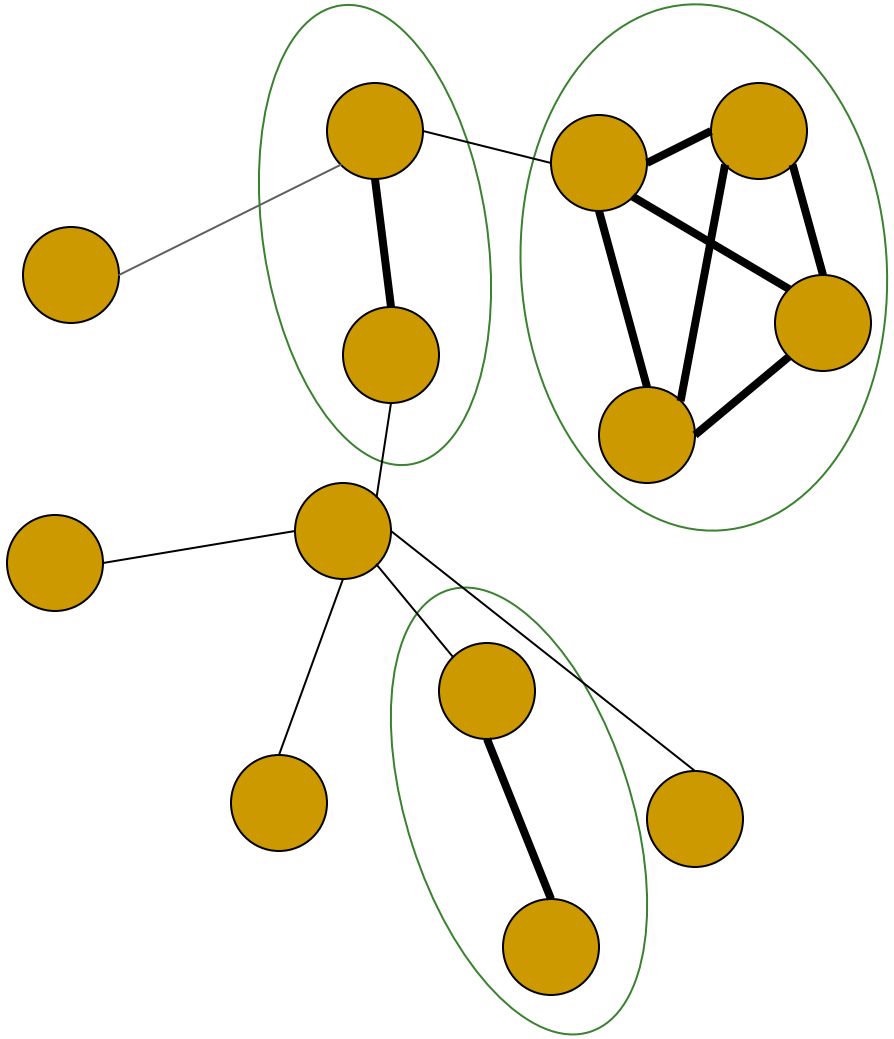
A data structure is a collection of items

An operation accesses a data set

not necessarily a pair

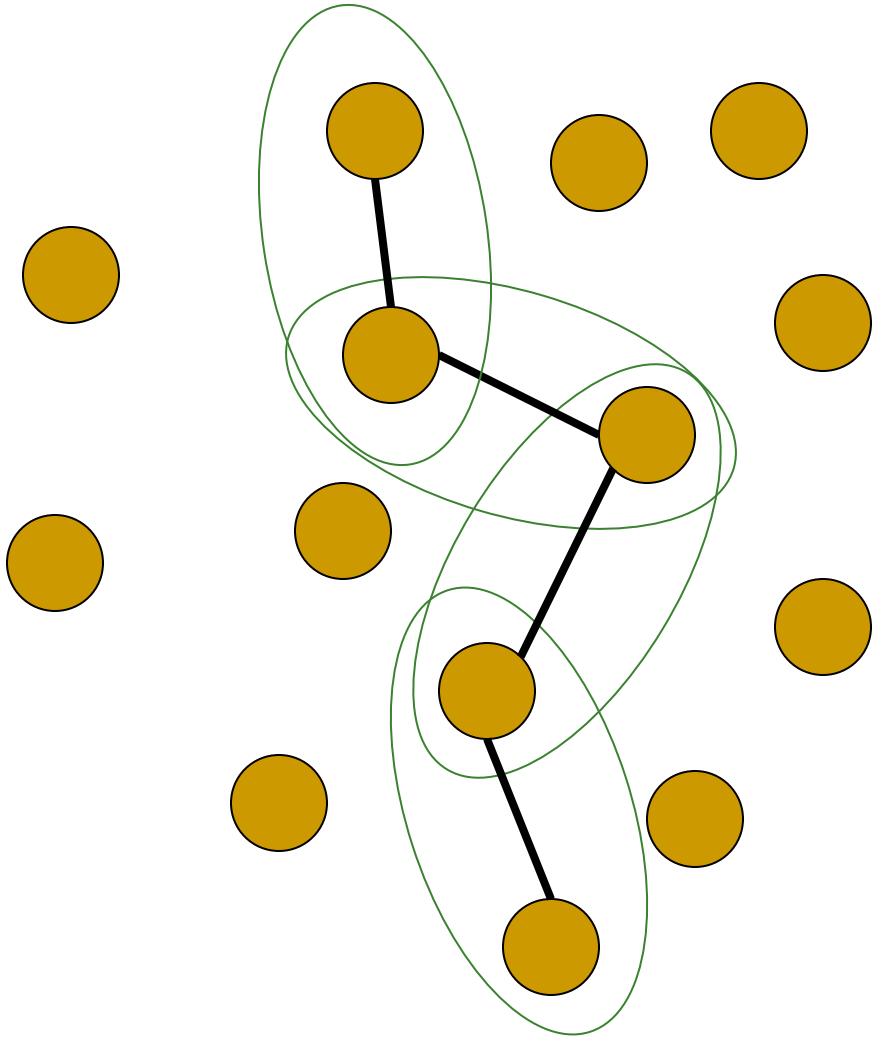
A collection of operations induces a conflict graph

- Nodes represent items
- Edges connect items of the same operation



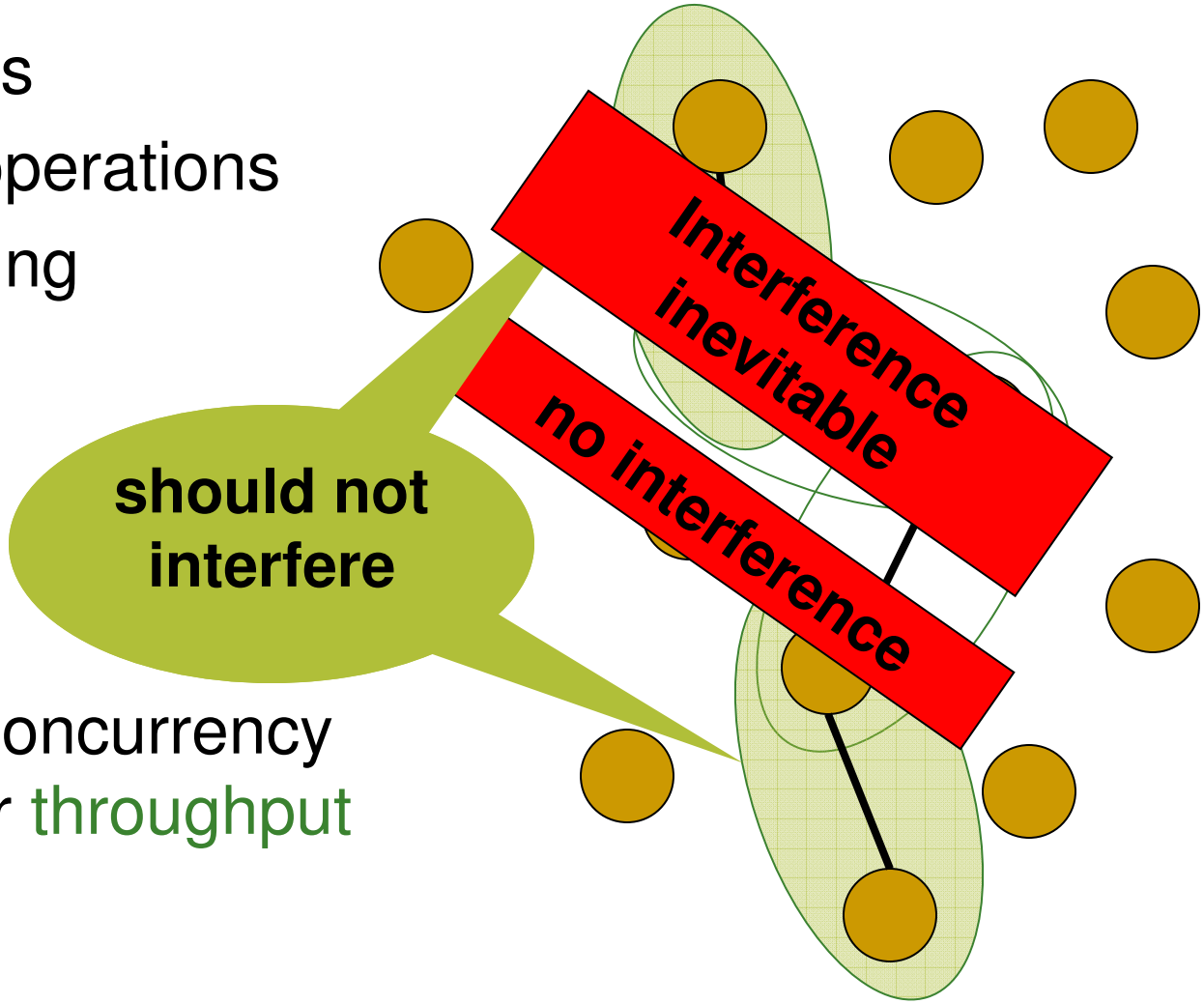
Spatial Relations between Operations

- Disjoint access
  - Non adjacent edges
  - Distance is infinite
- Overlapping operations
  - Adjacent edges
  - Distance is 0
- Chains of operations
  - Paths
  - Distance is length of path (in the example, 2)



Interference between Operations

- Disjoint access
- Overlapping operations
- Non-overlapping operations



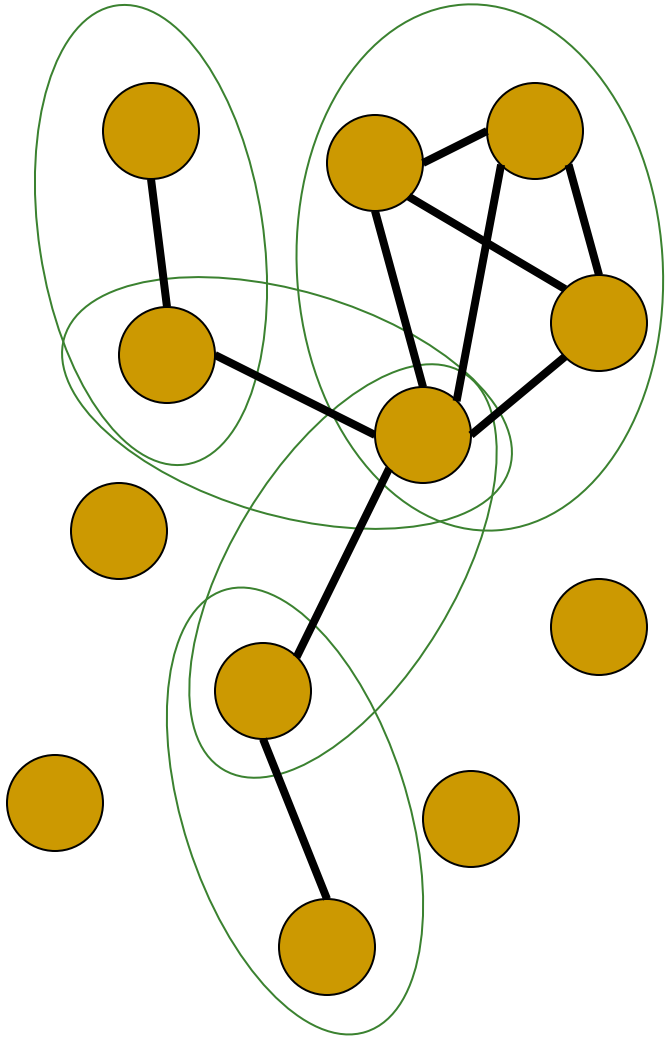
Provides more concurrency & yields better throughput

Measuring Non-Interference (Locality)

[Afek, Merritt, Taubenfeld, Touitou]

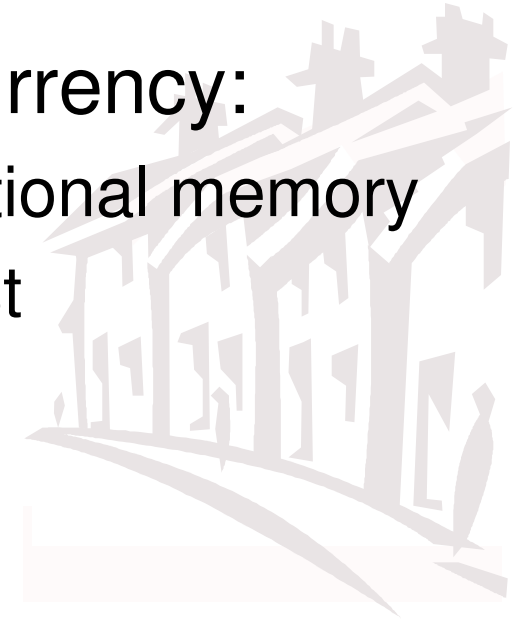
Distance in the conflict graph between overlapping operations that interfere

- *d*-local step complexity:
  Only operations at distance ≤ *d* delay each other
- *d*-local contention:
  Only operations at distance ≤ *d* access the same location



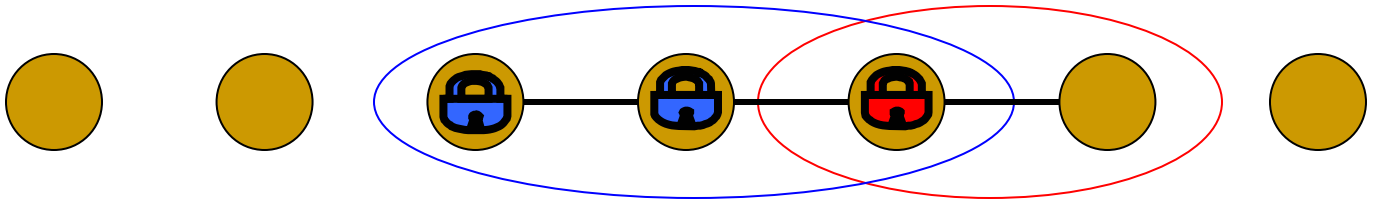
Talk Overview

- What are highly-concurrent data structures and why we care about them
- The concurrency of existing implementation techniques
- Two ideas for increasing concurrency:
  - Conflict management ⇨ Transactional memory
  - Locking order ⇨ Doubly-linked list



Virtual Locking

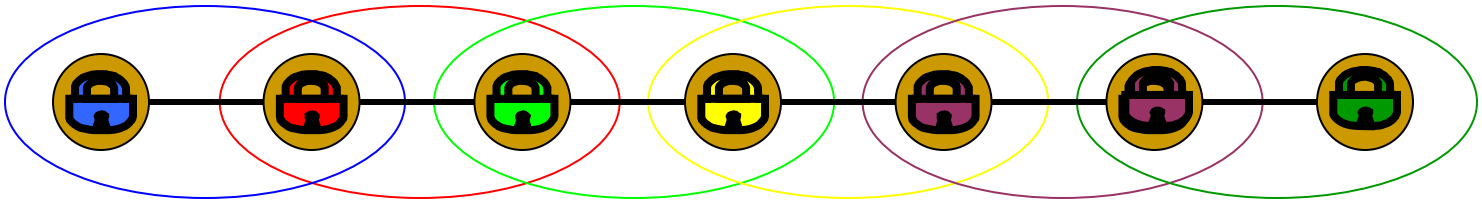
- It is easy to implement concurrent data structures if an arbitrary number of data items can be modified atomically
- Simulate with single-item Compare & Swap (CAS):
  - Acquire virtual locks on nodes in the data set
    - Perhaps in some order
  - Handle conflicts with blocking operations (holding a required data item)



A Nonblocking Implementation

[Turek, Shasha, Prakash] [Barnes]

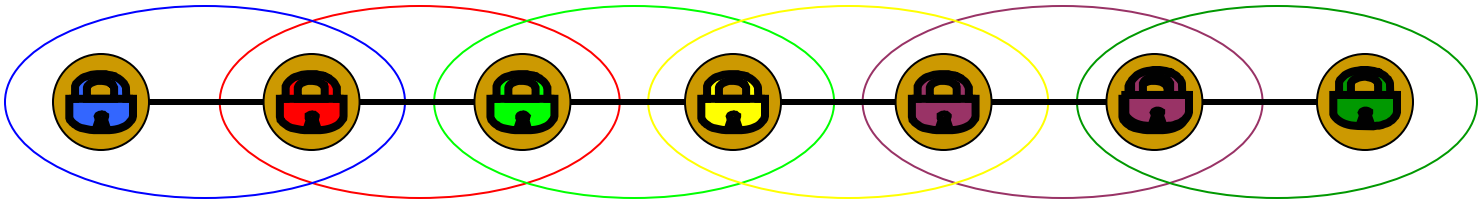
- Acquire locks by increasing addresses
  - Guarantees progress (the implementation is nonblocking)
- Help the blocking operation to complete (recursively)
- May result in long helping chains
  - *n*-local step complexity
  - *n*-local contention



Reducing Contention

[Shavit, Touitou]

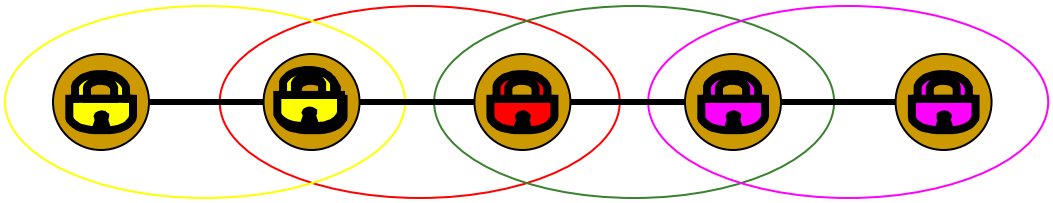
- Release locks when the operation is blocked
- Help an immediate neighbor (only) & retry…
- Short helping chains
  - *O(1)*-local contention
- Long delay chains
  - *n*-local step complexity



Color-Based Virtual Locking (Binary)

[Attiya, Dagan]

- Operations on two data items (e.g., DCAS)
- Colors define the locking order
  - Inspired by the left-right dinning philosophers algorithm [Lynch]
- Color the items when the operation starts
  - Non-trivial… [Cole, Vishkin]
- Bound the length of delay chains
  - (log*n)-local contention and step complexity
  - Due to the coloring stage



Color-Based Virtual Locking (Fixed k)

[Afek, Merritt, Taubenfeld, Touitou]

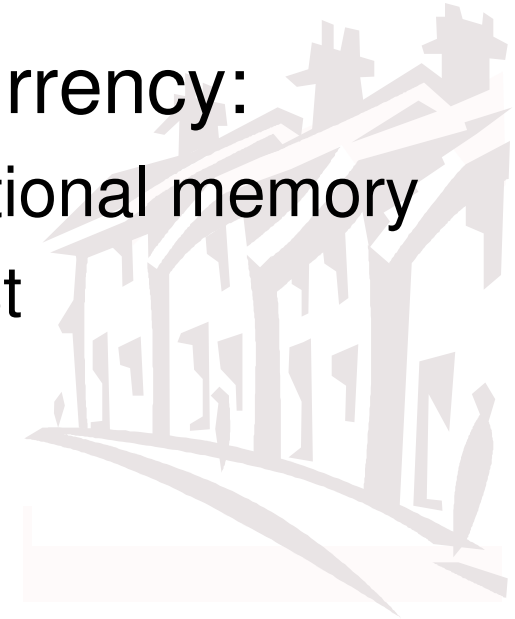
- Implements operations on *k* items, for a fixed *k*
- Based on the memory addresses, the conflict graph is decomposed into trees & items are legally colored

[Goldberg, Plotkin, Shannon]

  - Need to have the data set from the start
  - Recursive, with A&D at the basis and in each step
  - Even more complicated
  - (k+log*n)-local contention and step complexity

Talk Overview

- What are highly-concurrent data structures and why we care about them
- The concurrency of existing implementation techniques
- Two ideas for increasing concurrency:
  - Conflict management ⇨ Transactional memory
  - Locking order ⇨ Doubly-linked list



Transactional Memory

- Software transactional memory (STM) allows to specify a transaction, accessing an arbitrary data set, as if executed atomically
  - Static (data set is declared at invocation) vs. dynamic (data set is provided one-by-one)

- Alleged to be a silver bullet for concurrent programming


HPC wire

The Leading Source for Global News and Information Covering the Ec

Home Page | Free Subscription | A

Features:

Getting Serious About Transactional Memory

by Michael Feldman

Editor, HPCwire

The parallelization of computing, via multi-threading cores, multi-core processors

Locality of Known STMs

- Known STMs have O(n)-local step complexity
  - Some also O(n)-local contention
- AMTT can be considered as static STM for fixed-size data sets
  - (k+log*n)-local contention and step complexity
  - Recursive and restricted

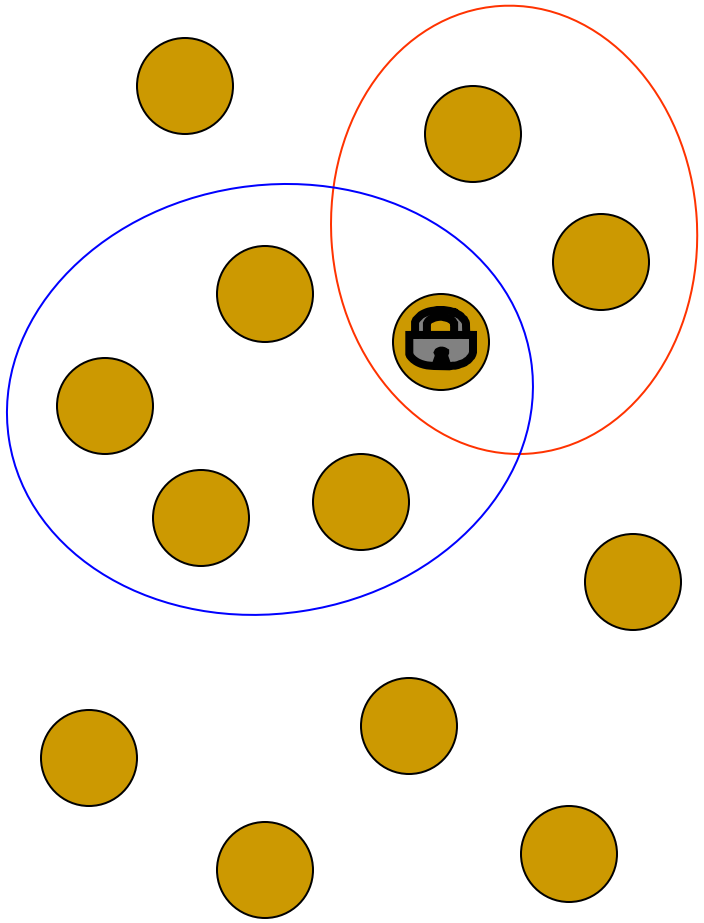
Highly-Concurrent STM

We concentrate on handling conflicts between operations contending for a data item

- Wait for the other operation
- Help the other operation
- Rollback the other operation

Acquire “locks” in arbitrary order

- No need to know the data set (or its size) in advance
- No pre-computation

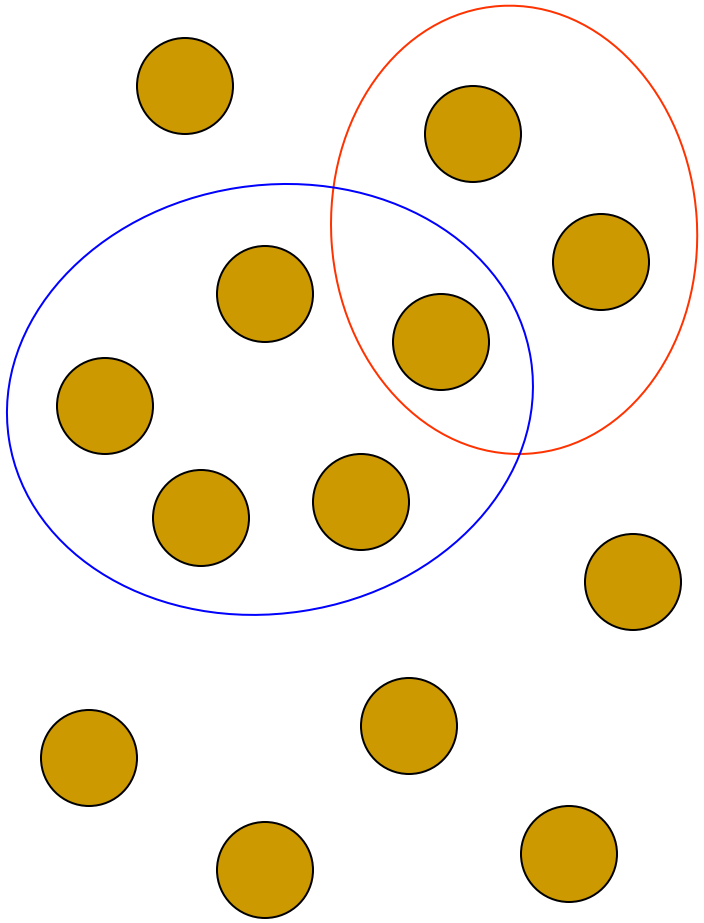


Conflict Resolution, How?

Conflict handling depends on the operations' progress so-far

More advanced operation gets the item

1. How to gauge progress?
2. What to do on a tie?



Who's More Advanced?

The operation that locked more data items is more advanced

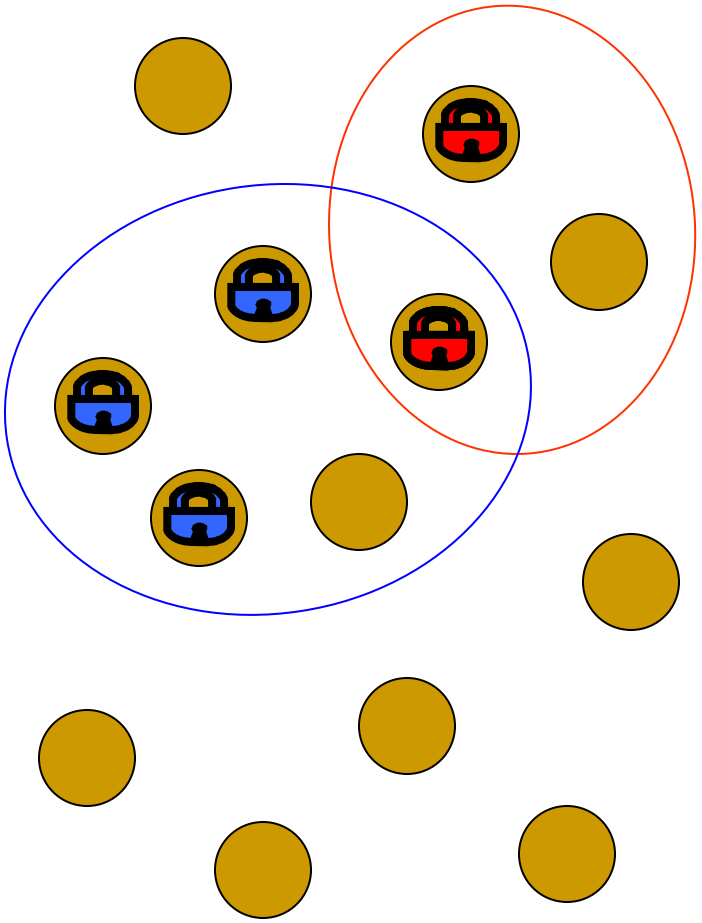
⇨ Locality depends on data set size

If a less advanced operation needs an item, then

- ☞ help the conflicting operation or
- ☞ wait (blocking in a limited radius)

If a more advanced operation needs an item, then

- ☞ rollback the conflicting operation until it releases the item



Local STM: What about Ties?

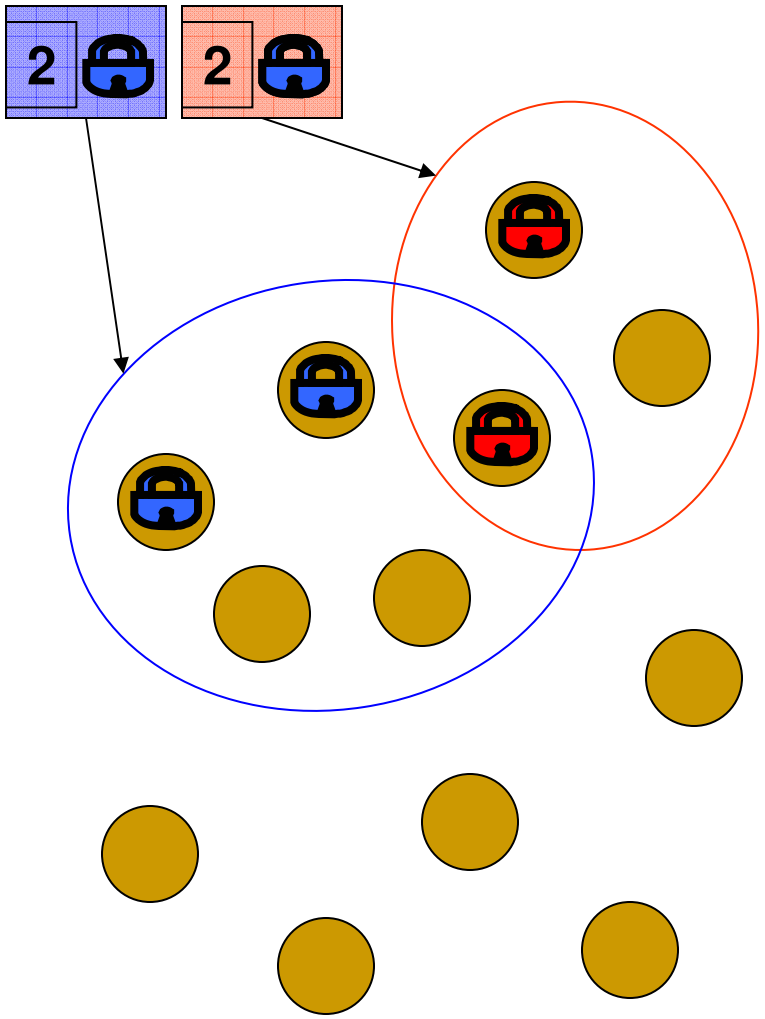
When both operations locked the same number of items

Each operation has a descriptor, holding information about it and a lock

The operations use DCAS to race for locking the two operation descriptors

Winner calls the shots…

There's (a lot) more…



Talk Overview

- What are highly-concurrent data structures and why we care about them
- The concurrency of existing implementation techniques
- Two ideas for increasing concurrency:
  - Conflict management ⇨ Transactional memory
  - Locking order ⇨ Doubly-linked list

Specific Data Structures

- Can be viewed as transactional memory
  - Our STM is highly-concurrent…
    - Relies on DCAS
  - But has high overhead

- Or handled by specialized algorithms
  - Ad-hoc and very delicate
  - Verification is an issue
  - Several wrong solutions...

Customized Virtual Locking

Instead of devising ad-hoc solutions,
design algorithms in a systematic manner…

- ☞ Lock the items that have to be changed &
  apply a sequential implementation on these items
- ☞ Lock items by colors to increase concurrency
- 👎 Need to re-color at the start of every operation

- ☝ But in a specific data structure
  - ☺ We manage a data structure since its infancy
  - ☺ The data sets of operations are predictable
    - 😐 unlike babies

Specifically: Doubly-Linked Lists

- An important special case underlying many distributed data structures
  - E.g., priority queue is used as job queue
- Insert and Remove operations
  - Sometimes only at the ends (priority queues / deques)
  - The data set is an item and its left / right neighbors (or left / right anchor)

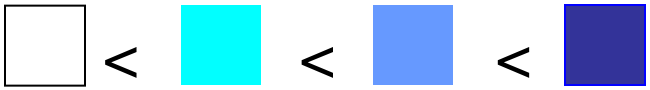
Built-In Coloring for Linked Lists

- Always maintain the list items legally colored
  - Adjacent items have different colors
  - Adjust colors when inserting or removing items
  - No need to color from scratch in each operation
    - No cost for coloring

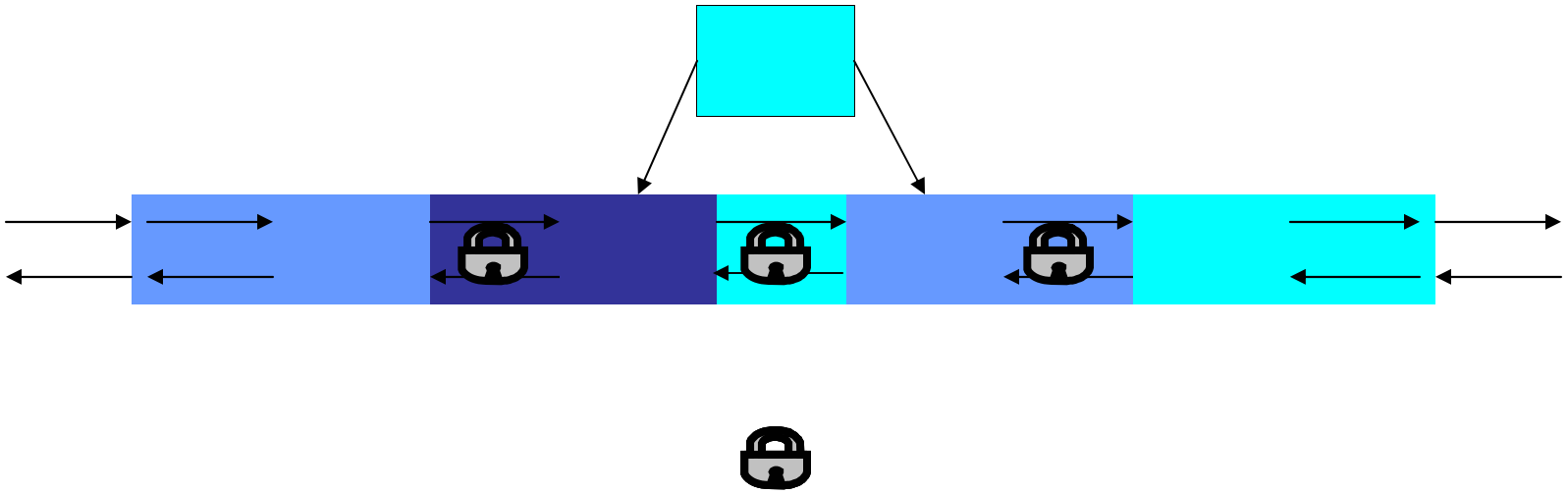
- Constant locality
  - Esp., operations that access disjoint data sets do not delay each other

Priority Queue

- Insert operation is quite simple
- New items are assigned a temporary color

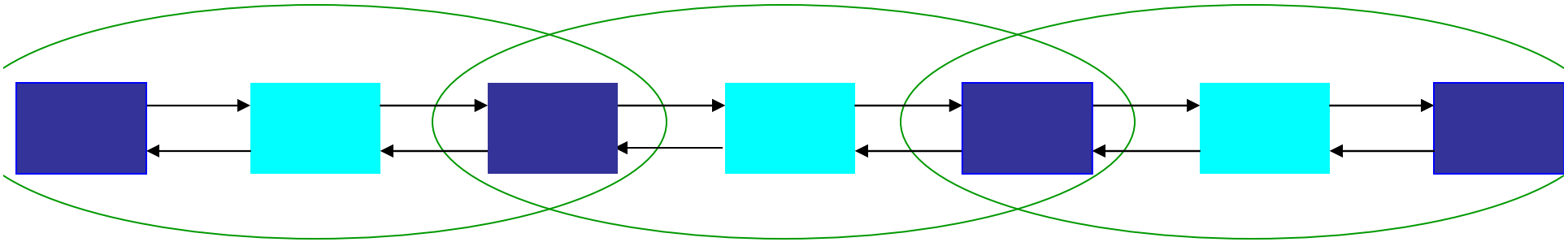


- Remove from the ends is similar to Insert
  - Locks three items, one of them an anchor



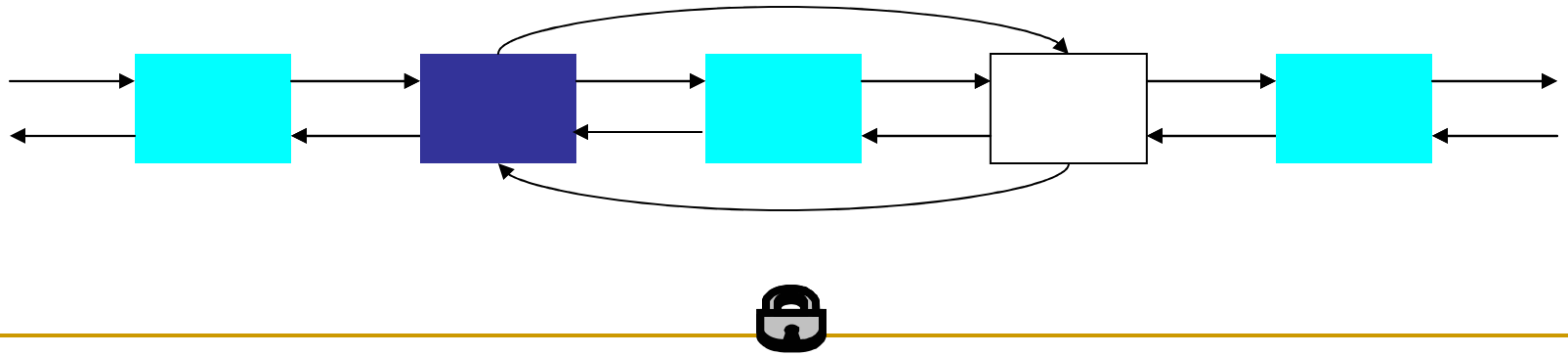
Full-Fledged Doubly-Linked List

- Remove from the middle is more complicated
  - Need to lock three list items
  - Possibly two with same color
- A chain of Remove operations may lead to a long delay chain in a symmetric situation



Doubly-Linked List: Remove

- Use DCAS to lock equally colored nodes



Recap

- Explicitly think about conflict resolution and locking order
  - Simplifies the design & correctness proofs while providing low interference
  - Ideas from / for fine-grained locking & resource allocation
    - Failure locality [Choi, Singh]
    - Concurrency control
- Our local STM incorporates contention management
  - More information about transactions' status & outlook

Next…

- Making it really work
  - Memory management
  - Special treatment for read-only items (read set)
- Other data structures
- Our results indicate the usefulness of DCAS
  - Provides a significant boost from CAS
  - Support in hardware? `Motorola 680x0, IBM S/370 ...`
  - Hand-crafted software implementations, further improving on [Attiya & Dagan]
  - Other implementations? [Fich, Luchangco, Moir, Shavit]
  - Proving inherent lower bounds
- Verification…

Thank you

Questions?