INTERRUPTING SNAPSHOTS AND THE JAVA SIZE() METHOD

Moran Tzafrir

Tel-Aviv University

Joint work with **Yehuda Afek** and **Nir Shavit**

Concurrent Data-Structures

- Multicore machines are the mainstream computing platform.
- Inter thread communication is the bottleneck to application performance
- Keeping concurrent data-structures scalable is key to applications' scalability.

Concurrent Data-Structure Libraries

- Operations **inherited** from the **sequential** API

- Typical **global-operations**:

ADT: LINKED-LIST	ADT: QUEUE	ADT: RB-TREE
❑Insert()	❑Enq()	❑Insert()
❑Remove()	❑Deq()	❑Remove()
❑Find()	❑Peek()	❑Find()
❑Size()	❑Size()	❑Size()
❑IsEmpty()	❑IsEmpty()	❑IsEmpty()
❑Clear()	❑Clear()	❑Clear()

- Global-operations, show **little or no** scalability.

The Size() Operation

- *Size()* linearizable operation added to any existing data-structure.

- Interface:
  - *Update()* – inc/dec with add/delete of elements.
  - *Size()* - get the number of elements in the data-structure.

- *Update()* more frequent than *Size()* by **order of magnitude**.

Known Size() Solutions

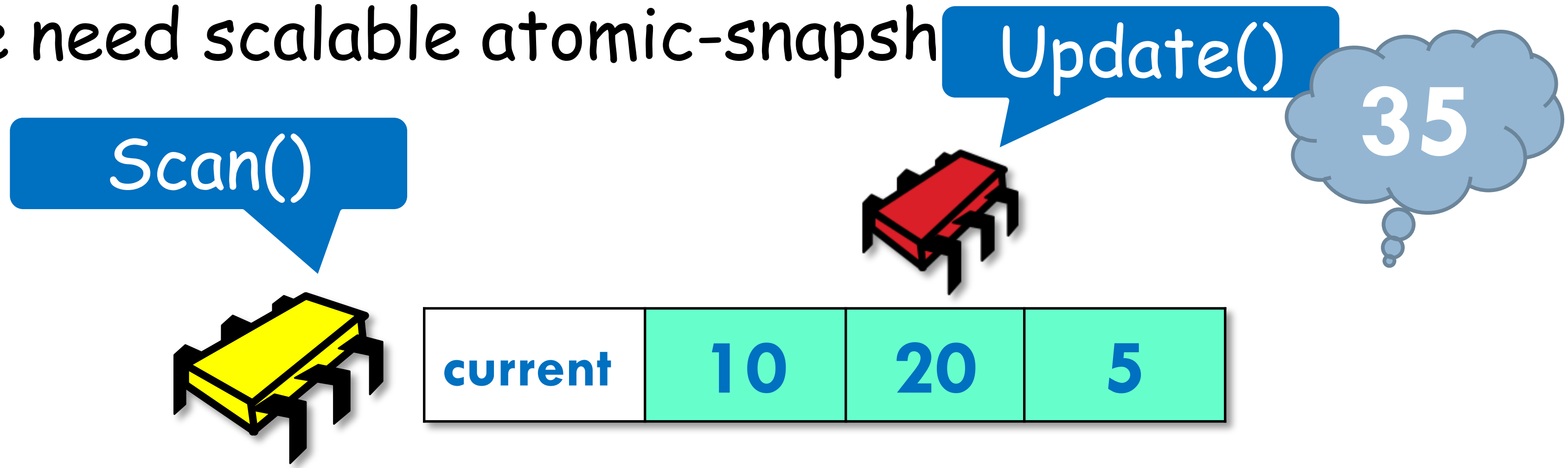
	Scalable	Update Performance	Size Performance
Global-Counter + CAS	X	X	√
Local-Counters + LOCKs	X	√̶	√̶
Weaken the spec (JDK)	√	√	√

Interrupting-Snapshots

- New *Size()* algorithm based on atomic-snapshots.
- Unlike existing solutions:
  - **Linearizable** (and not an approximation)
  - **Scalable**
  - **Wait-free**
- Potentially first example of using atomic-snapshots in industrial code.

Local Counters + Snapshots

- **Local Counters:** one per thread.
- *Size()* = $\Sigma_{i=1,n}$(snapshot of local-counters)
- We need scalable atomic-snapsh



Atomic Snapshot History

We need Practical Size()

√ *Update()* - O(1) read/write

√ *Scan()* - O(n) read/write + O(1) CAS

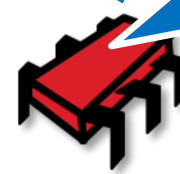
- **Introduc… …blev, Gafni, Merritt, Shavit, …**
- **Fundamental Problem: [PODC, DISC, SPAA, STOC, …OCS]**
- **Best in Theory: [Attiya, Rachman, 1988]**
  - ✗ *Update()* - O(n log n) read/write
  - √ *Scan()* - O(n log n) read/write
- **Suggested practical: [Riany, Shavit, Touitou, 1995]**
  - √ *Update()* - O(1) read/write
  - ✗ *Scan()* - O(n) costly CAS operations

Single Scanner Algorithm

[Kirousis, Spirakis, Ts … 95]

Read scan-seq (#6)

(curr.seq == thread seq) → update current value



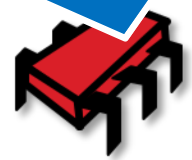
scan -seq	#6

current	10	20	5
curr.seq	#6	#6	#6
previous	8	4	0

Update() - If **(curr.seq == thread seq)**, update current value. **Else** copy current to previous with new seq before updating current.

Single Scanner Algorithm

Update(22)
(Read scan-seq #6)



scan -seq	#6

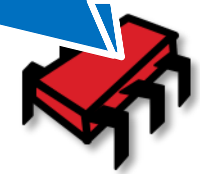
current	10	22	5
curr.seq	#6	#6	#6
previous	8	4	0

Update() - If **(curr.seq == my seq)**, update current value. **Else** copy current to previous with new seq before updating current.

Single Scanner Algorithm

(curr.seq < my seq) → copy

Due to *Scan()*, scan-seq == #7

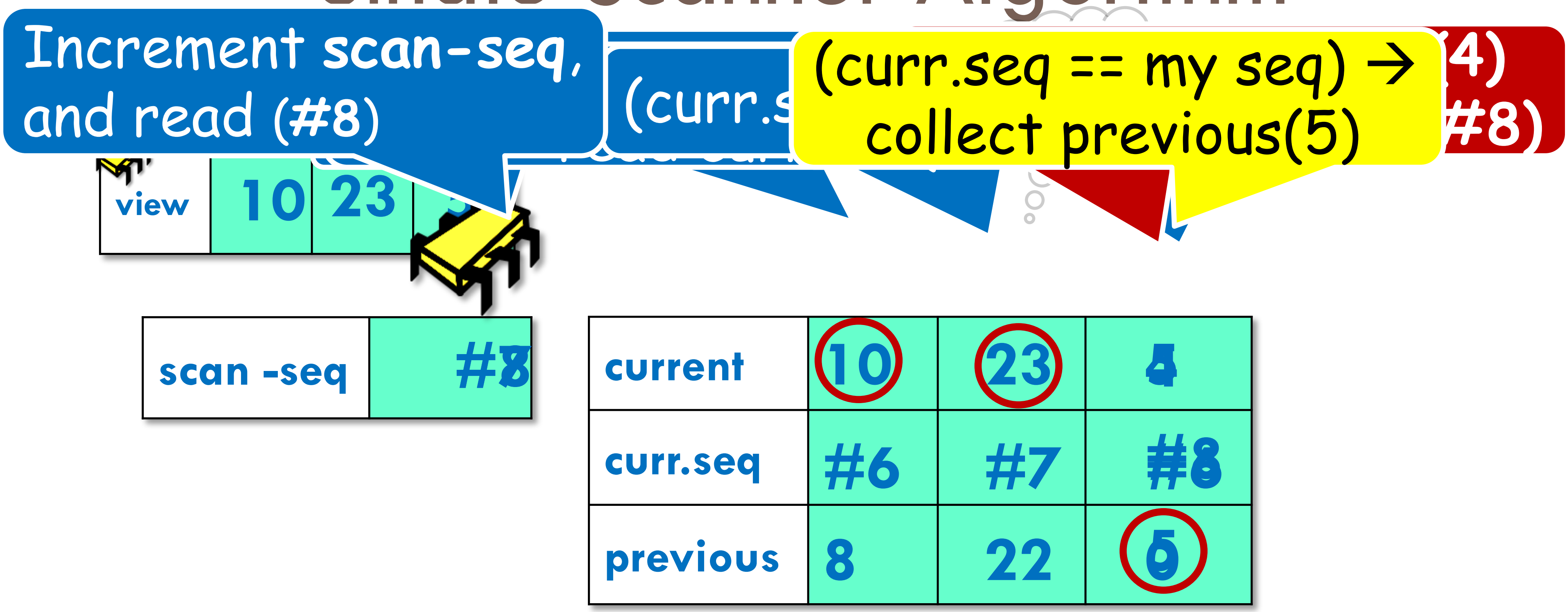


scan -seq	#6 #7

current	10	22 23	5
curr.seq	#6	#6 #7	#6
previous	8	42 22	0

Update() - If **(curr.seq == my seq)**, update current value. **Else** copy current to previous with new seq before updating current.

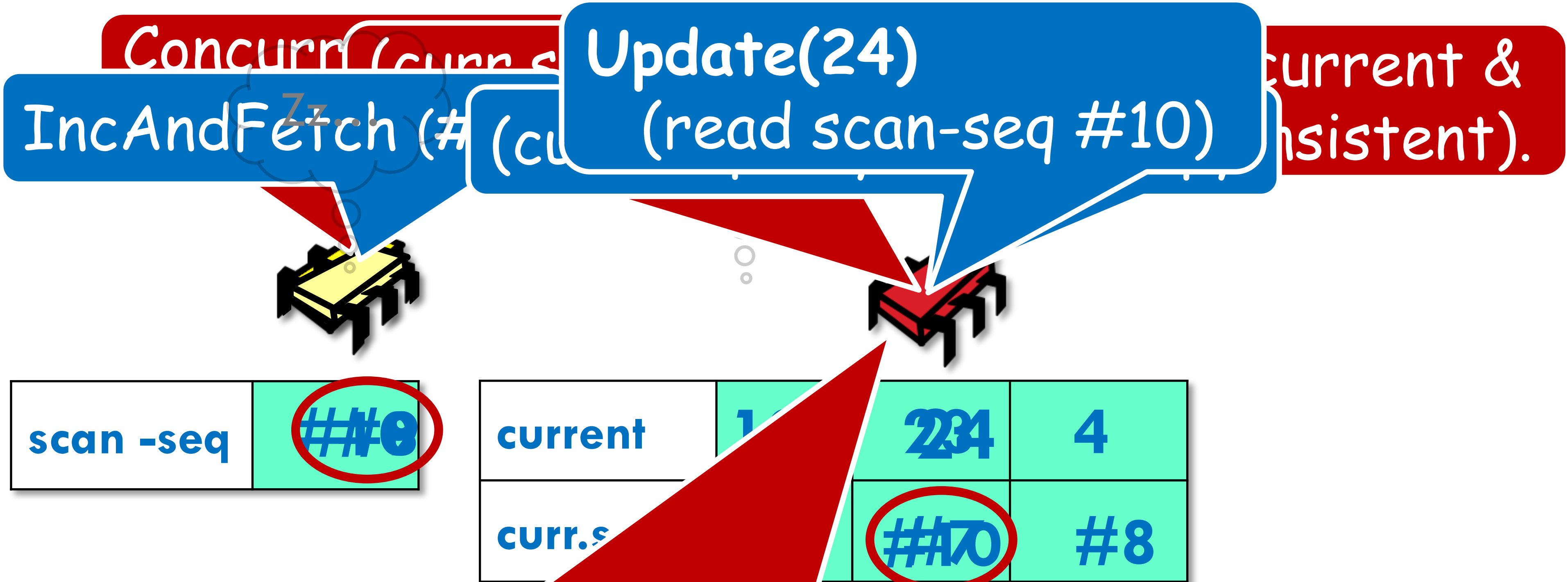
Single Scanner Algorithm



Scan() - Increment scan-seq (**a write**). Go over the local-counters, and collect values with

curr.seq < my seq

Multi Scanner - Lock Free

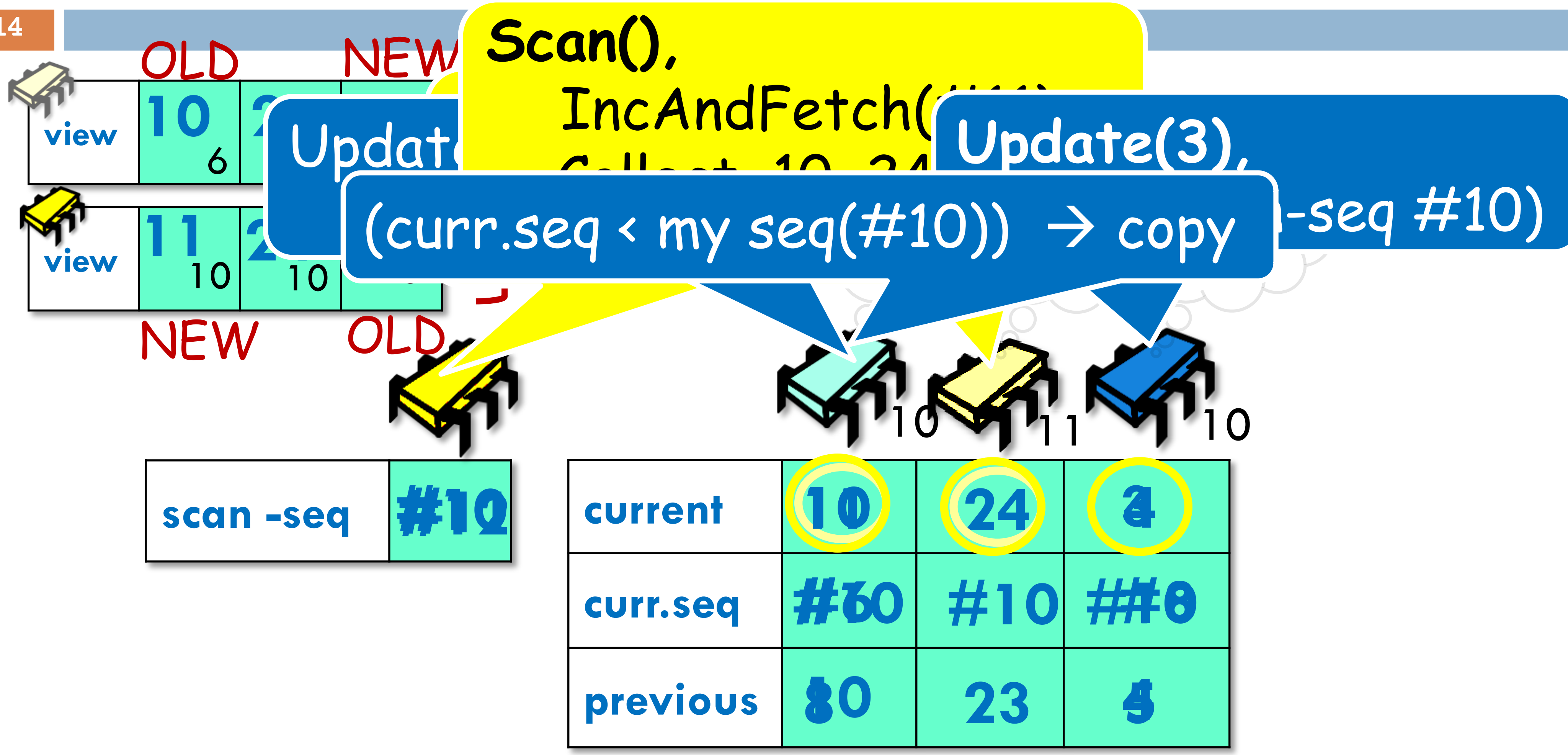


Solution: Do not increment scan-seq, **adopt interrupting sequence** (#10). → (Lock Free)

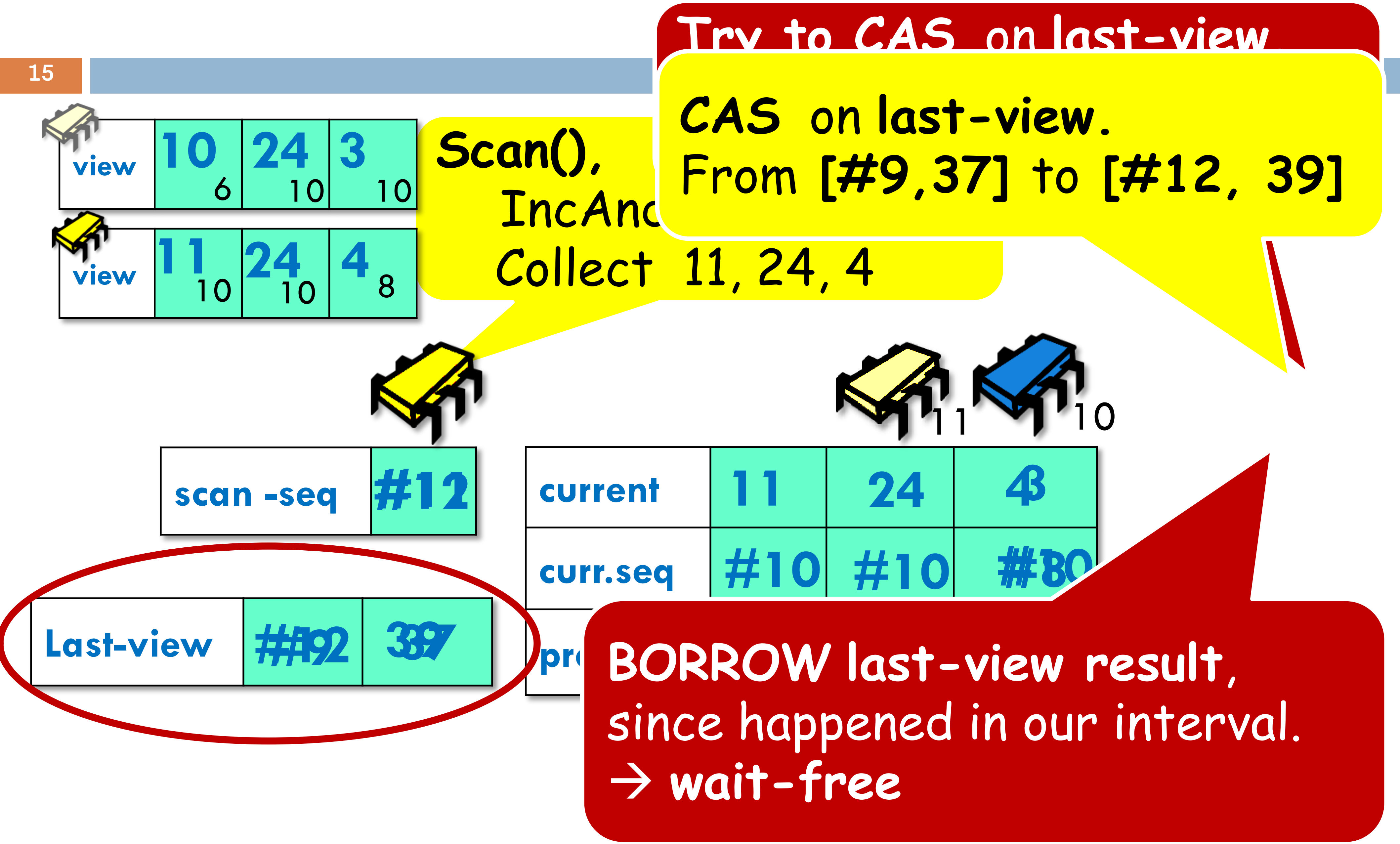
Scan() - Increment scan-seq using **CAS**. Collect values with **curr.seq < my seq**.

If **(curr.seq > my seq)** abort and restart (without incrementing the scan-seq).

Counter Example - Multi Scanner



Multi Scanner - **Wait Free**



Complexity

- *Update()* - Wait-free **O(1)** read/write operations
- *Scan()* - Wait-free **Amortized O(1)** CAS operations

Performance Evaluation

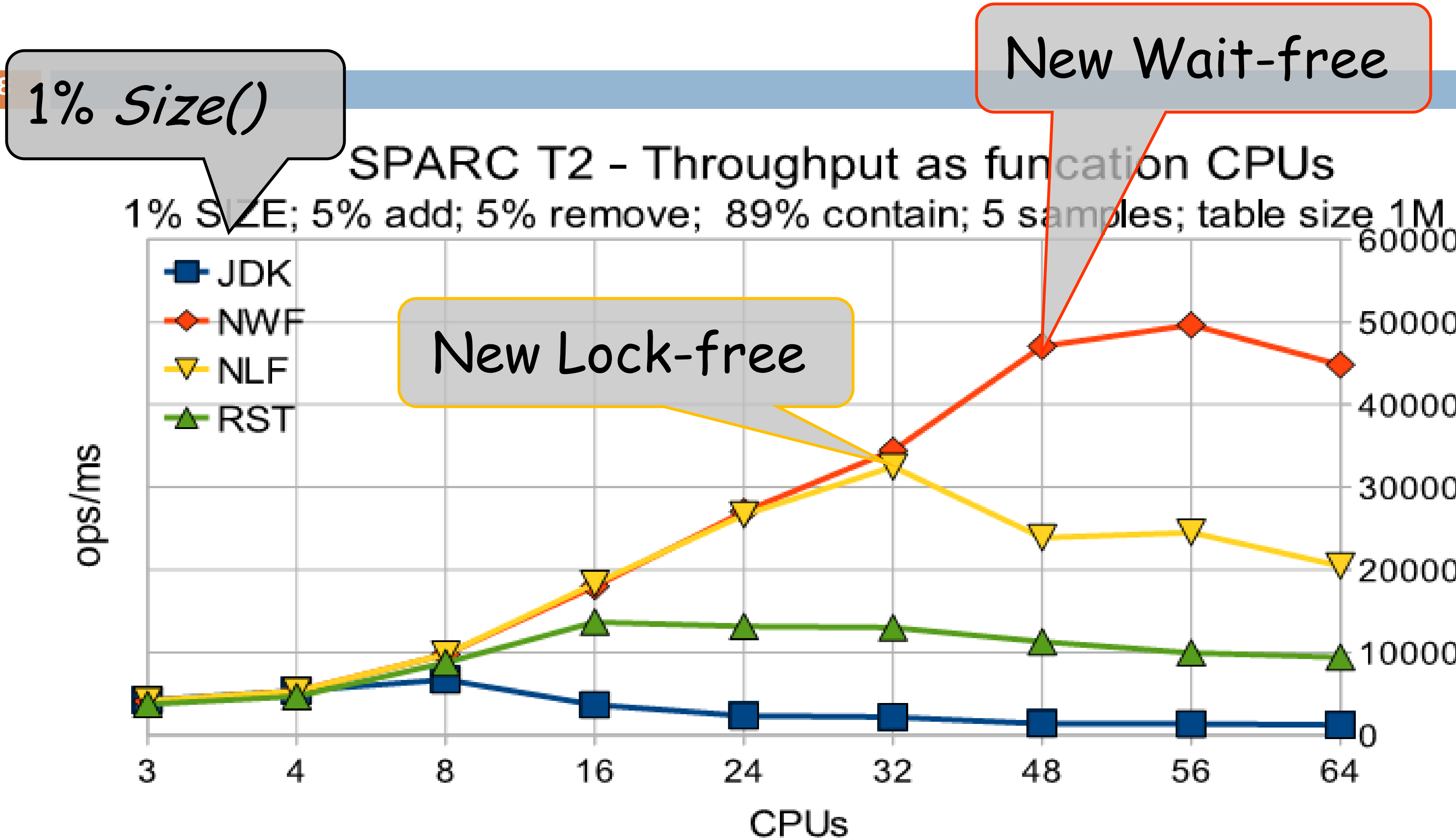
Compare

- New Lock-free Size implementation(**NLF**)
- New wait-free (**NWF**)
- Java Concurrency Package (**JDK**)
- Using [RianyShavitTouitou] Snapshot (**RST**)

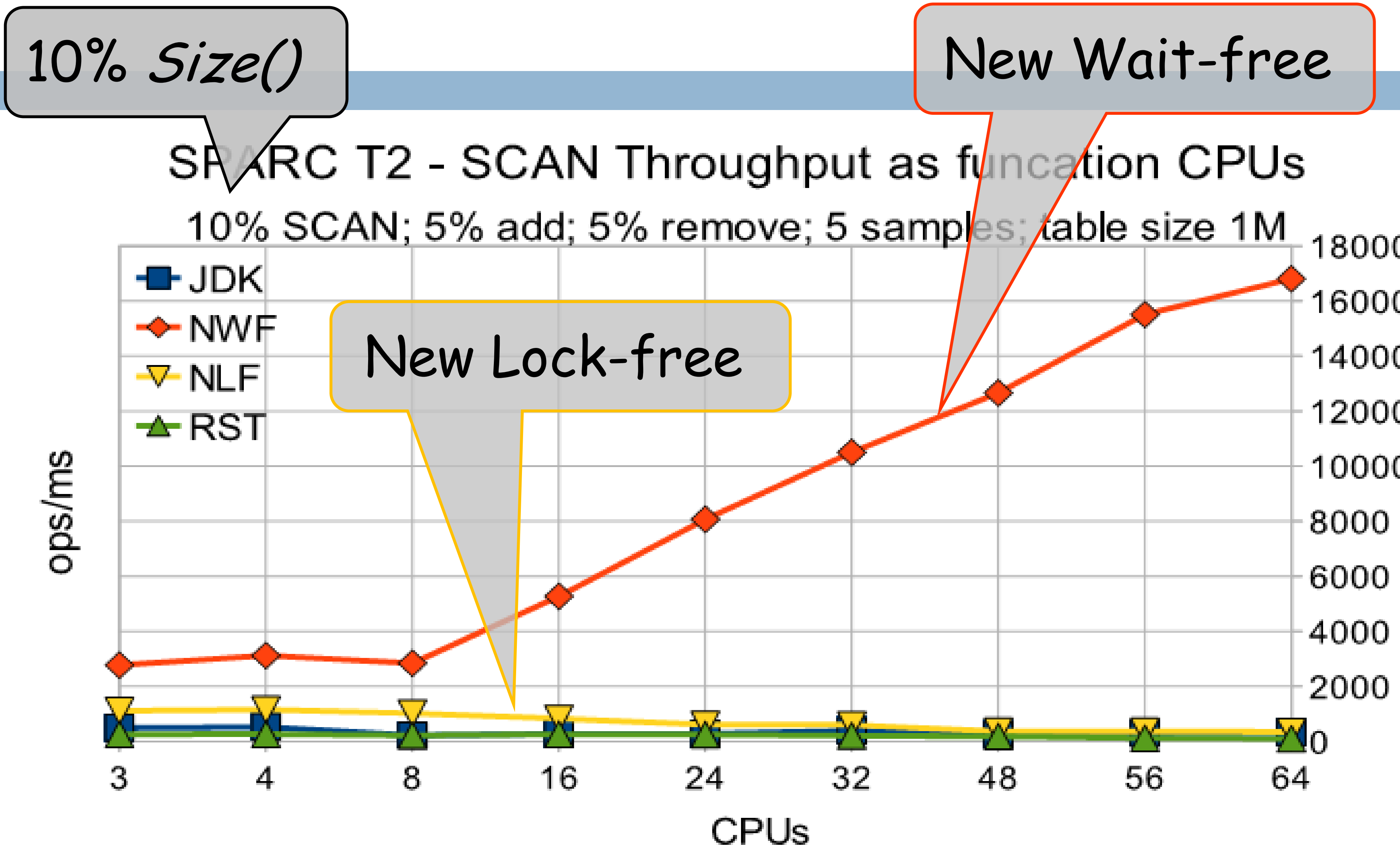
On

- Sun UltraSPARC T2 (64x)
- Azul Vega2 7200 series (~200x)

Performance Evaluation

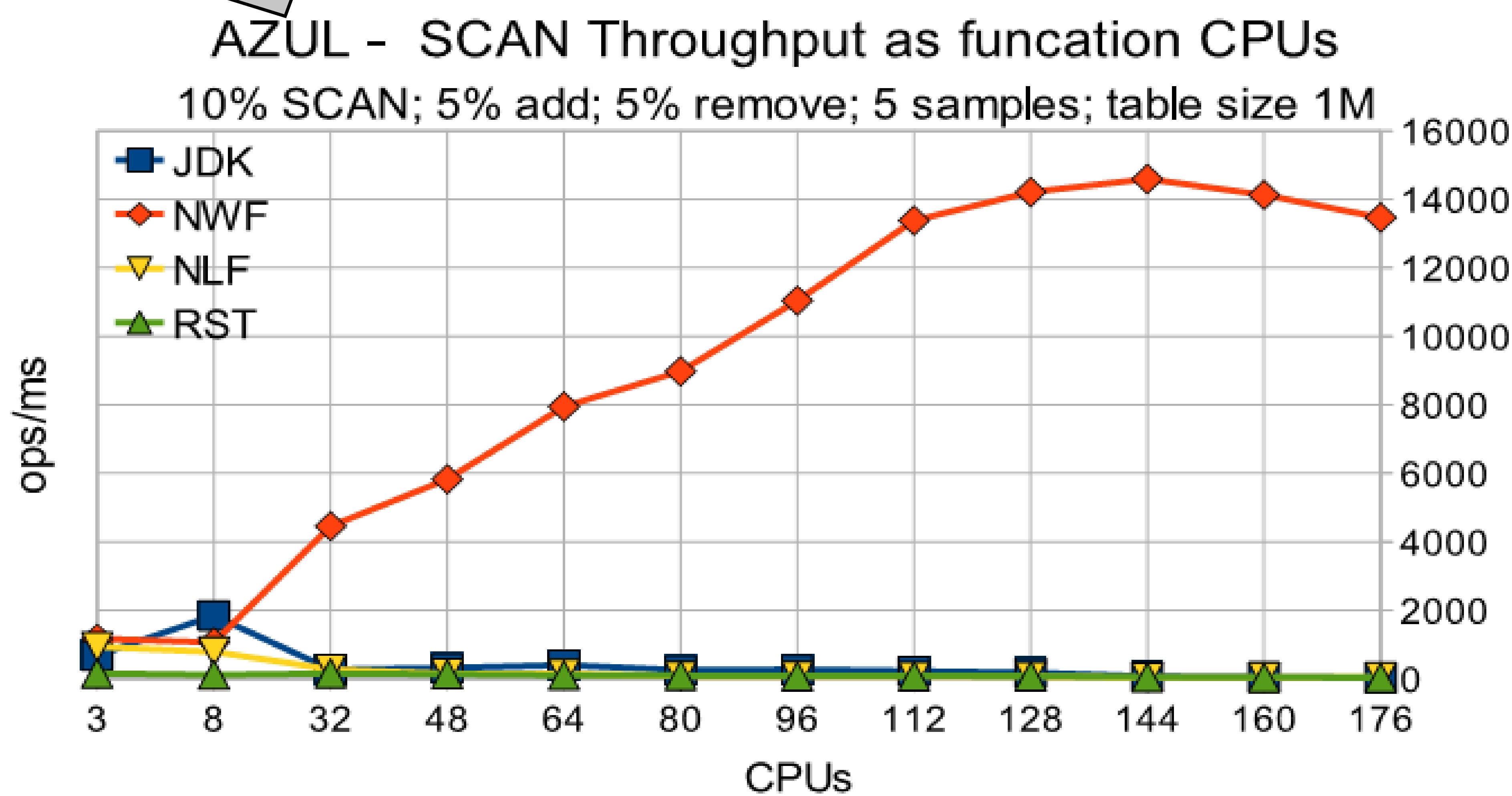


Performance Evaluation

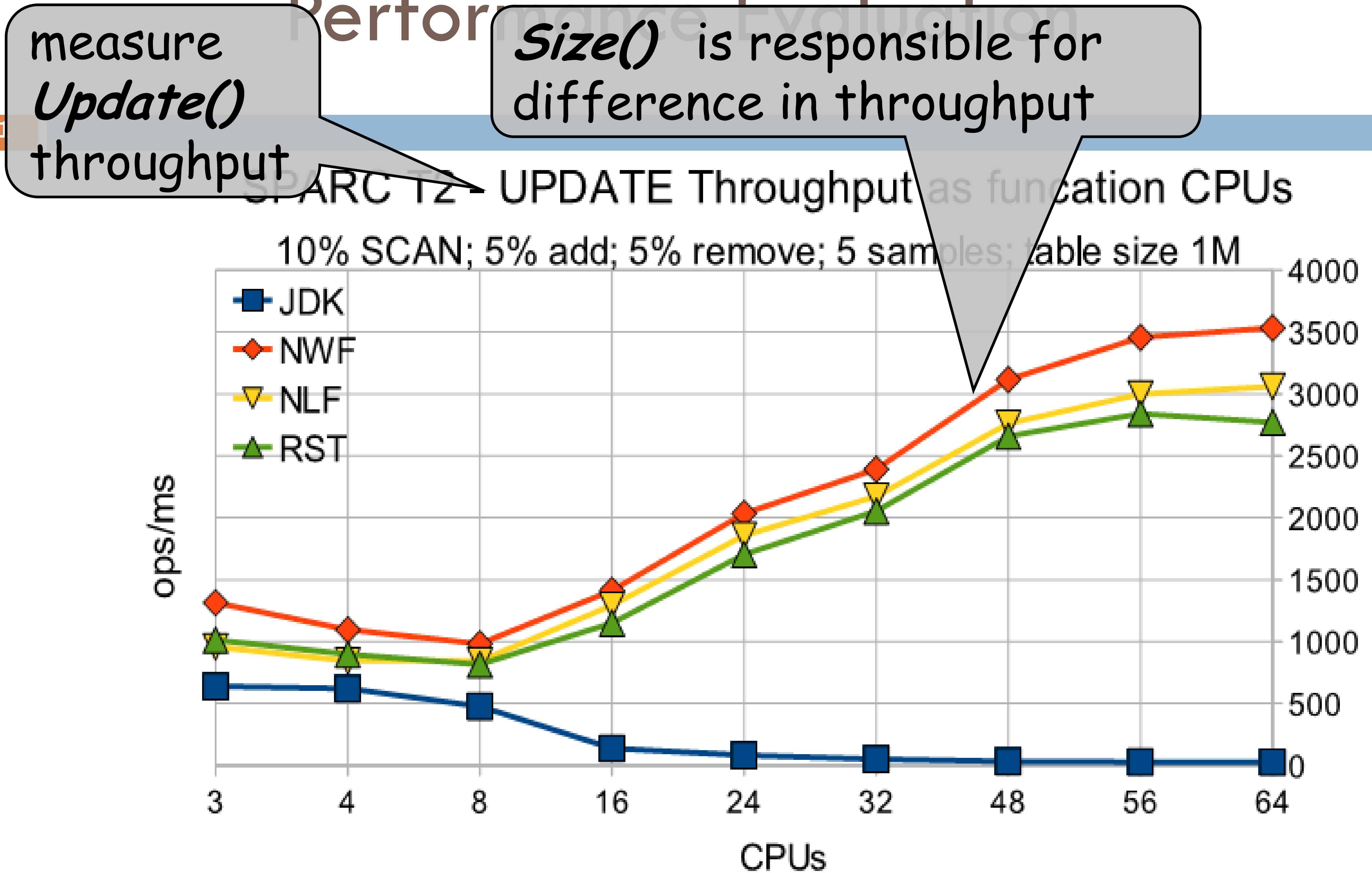


Performance Evaluation

Same behavior on **Azul**



Performance Evaluation



Conclusion

- Global operations *Size(), IsEmpty()* can be both accurate and scalable.

- Fundamental **atomic-snapshot** theory problem finds practical application

Future Work

Future Work:

- Uses for atomic-snapshots in TM, concurrent-applications, …
- Extending the Interruption Snapshots algorithm to a view that is more than a single word.

Thanks for your time ....

Questions ....