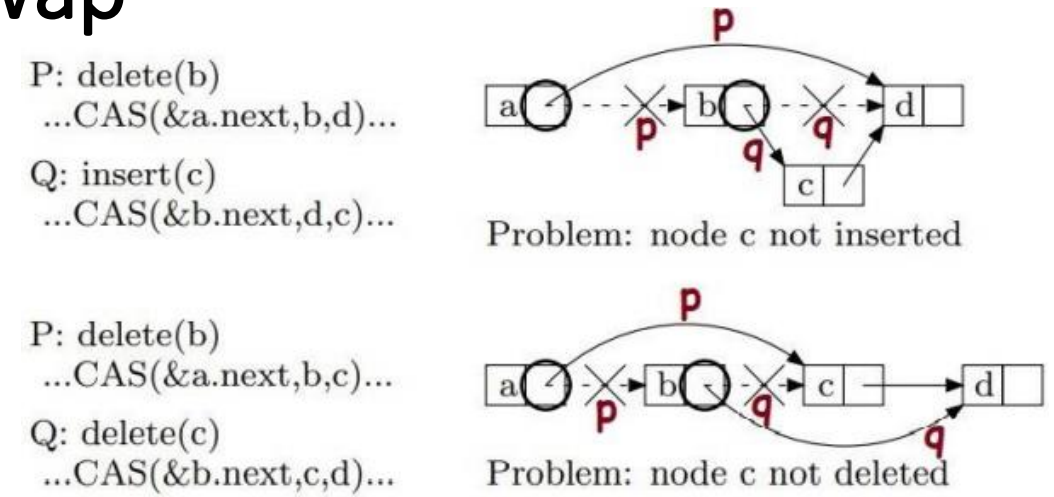
Nonblocking K-Compare Single-Swap

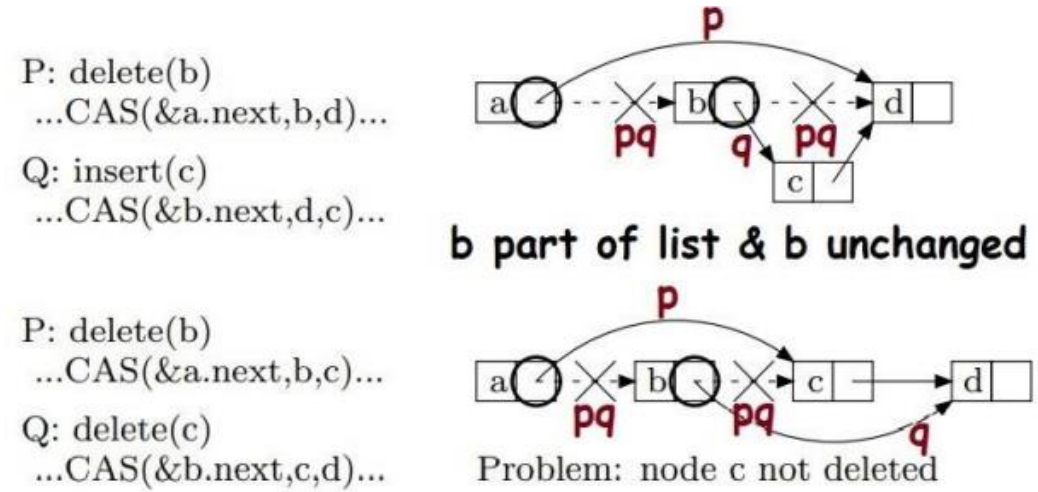
- **Victor Luchangco** (Sun), **Mark Moir** (Sun), **Nir Shavit** (Tel Aviv).
- **Theory of Computing Systems, Springer** (2008)
- **DOI**:10.1007/s00224-008-9155-5, **Corpus ID**: 40581685
- **US7865671B2** patent assigned to Oracle America Inc (2007-08).

- Why KCSS is needed?
- 2 extremes of software synchronization.
- Compare And Swap (CAS) vs Transactional Memory (TM)

- KCSS is somewhere in between.
- Requires 2 CAS, 2 stores, 2k loads.
- Lock-free (obstruction free), requires no barriers (TSO model).



Problems with lock-free linked-list
(Why KCSS is needed)



2CSS can be useful here

KCSS Operations

A k-location-compare single-swap (KCSS) operation takes k locations $a_1..a_k$, k expected values $e_1..e_k$, and a new value n_1. If the locations all contain the expected values, the KCSS operation atomically changes the first location a_1 from e_1 to n_1 and returns *true*; in this case, we say that the KCSS *succeeds*. Otherwise, the KCSS returns *false* and does not modify any memory location; in this case we say that it *fails*.

Application value

KCSS Tag

- **RESET(a)**: If address a is a tag, try replace with value.
- **READ(a)**: Read value at addres a, RESET is required.
- **LL(a)**: Load value at address a, insert new tag at a.
- **SC(a, n)**: Try replace tag at address a, with new value n.
- **COLLECT_VALUES(a[1..k])**: READ values at addresses a[1..k].
- **COLLECT_TAGGED_IDS(a[1..k])**: Read tags/values at addresses a[1..k].
- **SNAPSHOT(a[1..k])**: COLLECT_TAGGED_IDS and COLLECT_VALUES in mirrored order and ensure they match before returning read values.
- **KCSS(a[1..k], e[1..k], n)**: LL first address a1 and SNAPSHOT a[2..k] and if read values match expected values e[1..k], use SC to try write value n at a1, otherwise restore old vlaue at a1 with SC.

- LL: Load Linked
- SC: Store Conditional

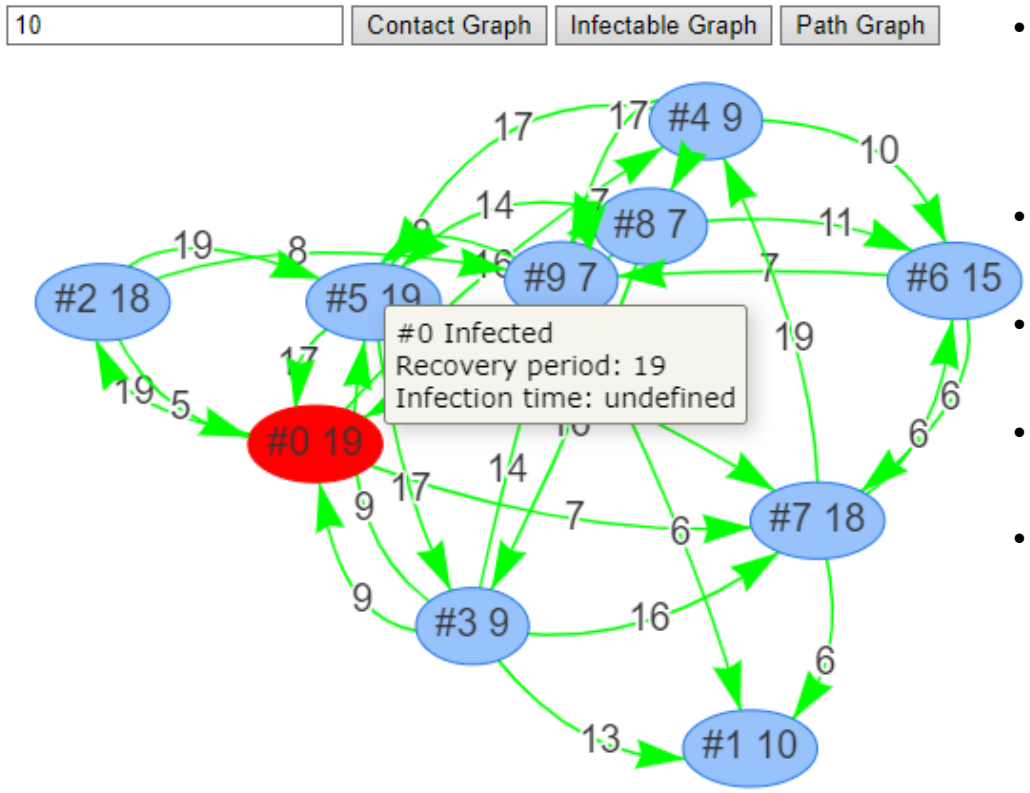
Transactions with KCSS?

- KCSS can support single-write transactions.
- Bank transactions are 2-reads 2-writes (2C2S).
- In general transactions require k-reads k-writes (KCKS).
- K-writes could be supported with k-tags (k - LL/SC).

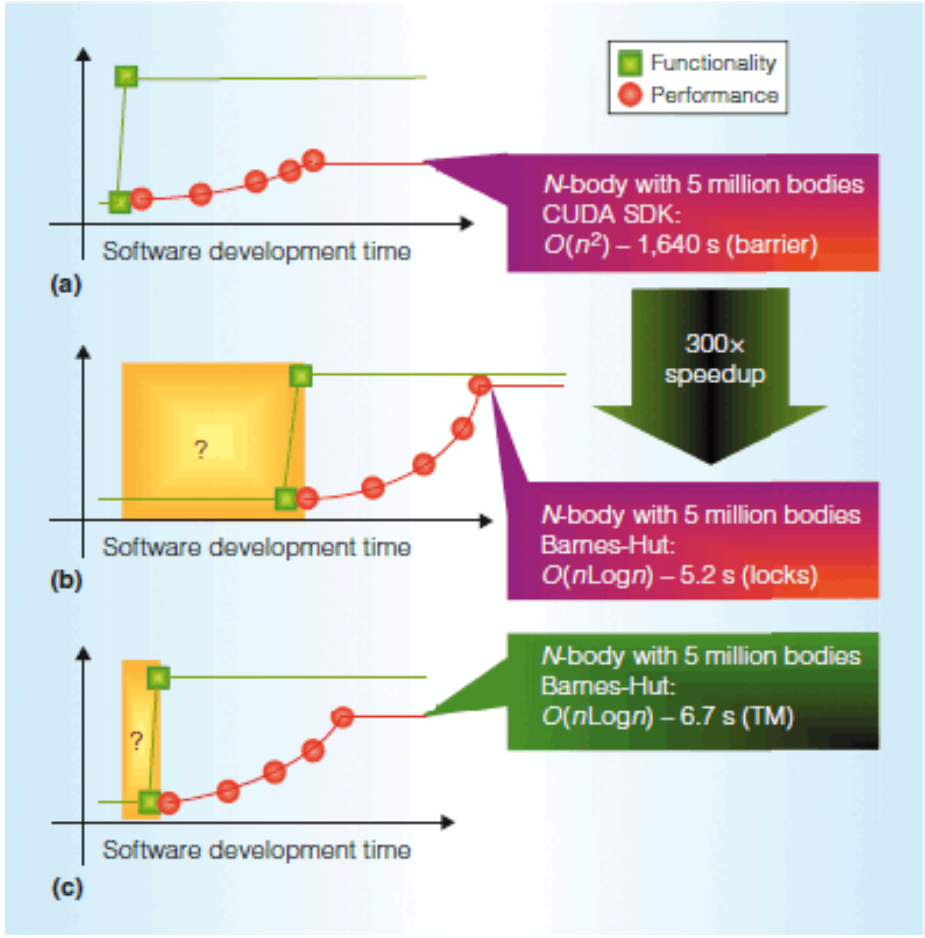
```
bool KCSS(int k, (loc_t *) a[1..k],
          value_t expvals[1..k], value_t newval){
    value_t  oldvals[1..k];
K1: while (true) {
K2:   oldvals[1] = LL(a[1]);
K3:   oldvals[2..k] = SNAPSHOT(k-1,a[2..k]);
K4:   if (for some i, oldvals[i] != expvals[i])
K5:     SC(a[1], oldvals[1]);
K6:     return false;
      // try to commit the transaction
K7:   if (SC(a[1], newval)) return true;
    } // end while
}
```

```
typedef struct loc_s {
  taggedid_t tid; // used for SNAPSHOT
  value_t val;    // atomically CASable
} loc_t;

void RESET(loc_t *a){
1:  value_t oldval = a->val;
2:  if(TAGGED_ID(oldval))
3:  CAS(&a->val, oldval, VAL_SAVE[ID(oldval)]); }

value_t LL(loc_t *a){
4:  while (true) {
5:    INC_MY_TAGGED_ID;          // increment local tag
6:    value_t val = READ(a);
7:    VAL_SAVE[MY_ID] = val;
8:    if (CAS(&a->val, val, MY_TAGGED_ID)) {
9:      a->tid = MY_TAGGED_ID;   // needed for SNAPSHOT
10:     return val;
      }
    }
}

bool SC(loc_t *a, value_t newval){
11: return CAS(&a->val, MY_TAGGED_ID, newval);
}

value_t READ (loc_t *a){
12: while (true) {
13:   value_t val = a->val;
14:   if (!TAGGED_ID(val)) return val;
15:   RESET(a);
    }
}
```

Transactional memory in GPU



- **N-body simulation** is a simulation of a dynamical system of **particles**, usually under the influence of **physical forces**, such as gravity.
- **Astronomy** (e.g. formation of galaxies)
- **Molecular Dynamics** (e.g. protein folding)
- **Other**: Fluid Dynamics, Plasma Physics, ..
- **Games**: Kerbal Space Program



barnesHut	*Object*	*Object*	BarnesHut is a quadtree based gravity model. This is the fastest, default and recommended solver for non-hierarchical layouts.