Thread Summaries for Lock-Free Data Structures

[NWPT16]

Sebastian Wolff[1,2]

with

Lukáš Holík[3] Roland Meyer[2] Tomáš Vojnar[3]

[1] Fraunhofer ITWM [2] TU Braunschweig [3] Brno University

Setting

- verifying safety of
  - lock-free code ⇝ data structures
  - library code ⇝ ∞ clients
  - C-like memory ⇝ no GC
- fully automated
  - first success: Abdulla et al. TACAS'13

Lock-Free Programming

1. create local snapshot

2. apply changes locally

3. atomically:

    if snapshot inconsistent: go to step 1
    otherwise: write back modified snapshot

Lock-Free Programming

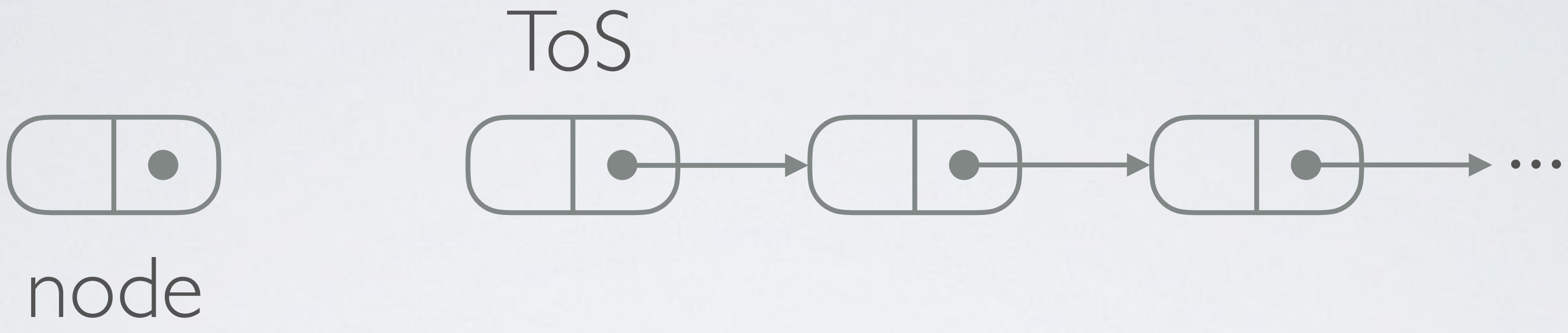
R. create local snapshot

M. apply changes locally

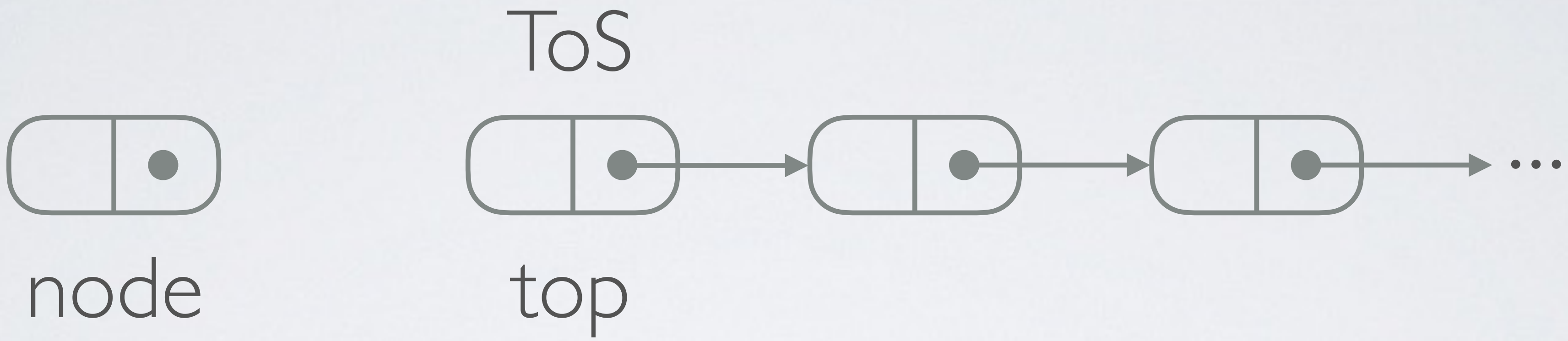
W. atomically:

- if snapshot inconsistent: go to step 1
- otherwise: write back modified snapshot

Treiber's Stack

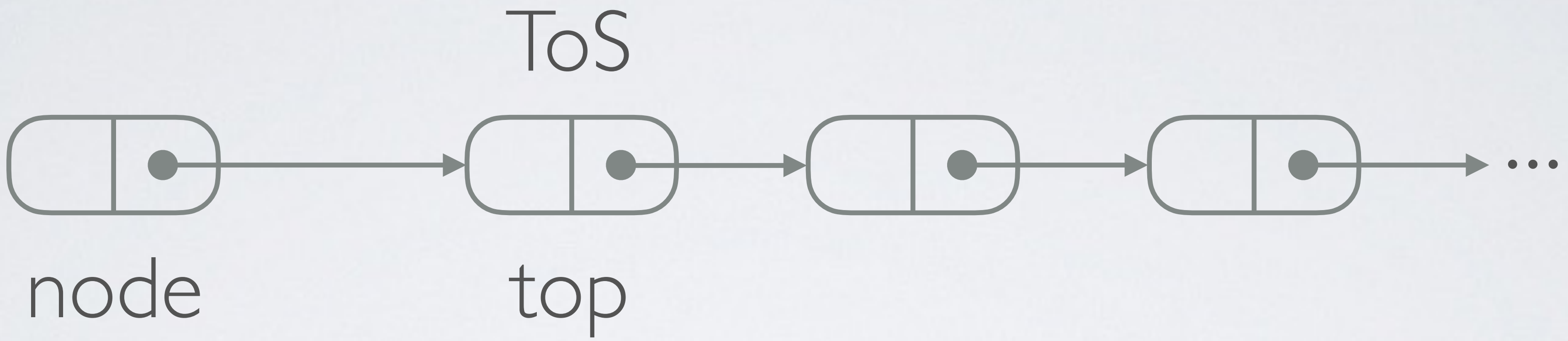


Treiber’s Stack



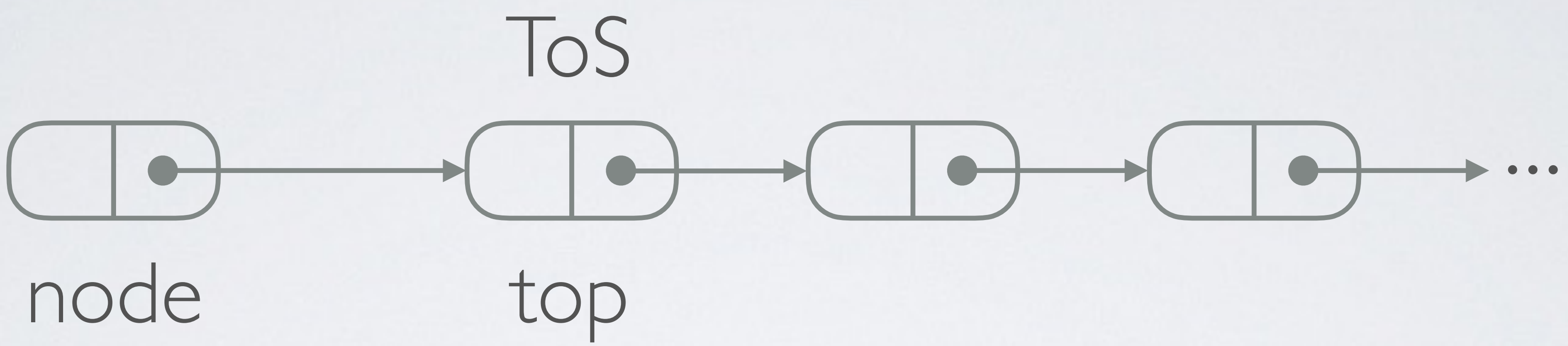
R. `top = ToS;`

Treiber's Stack



```
R.  top = ToS;

M.  node.next = top;
```

Treiber's Stack

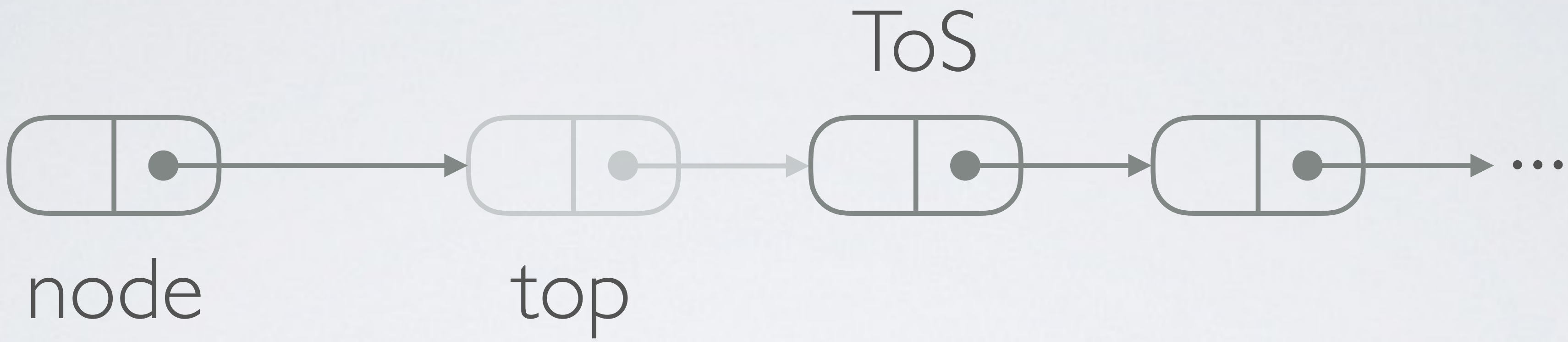


R. `top = ToS;`

M. `node.next = top;`

W. `CAS(ToS, top, node)`

Treiber’s Stack

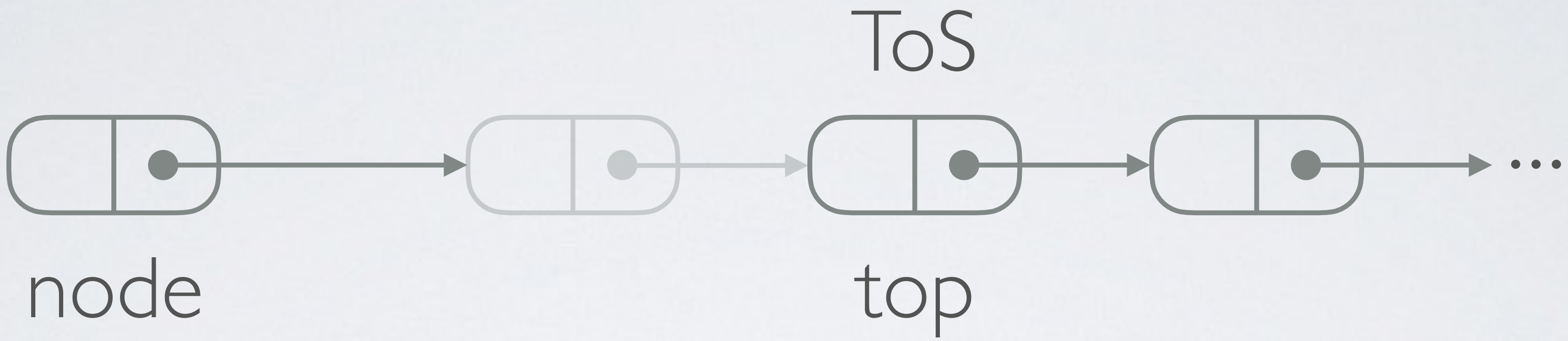


R. `top = ToS;`

M. `node.next = top;`

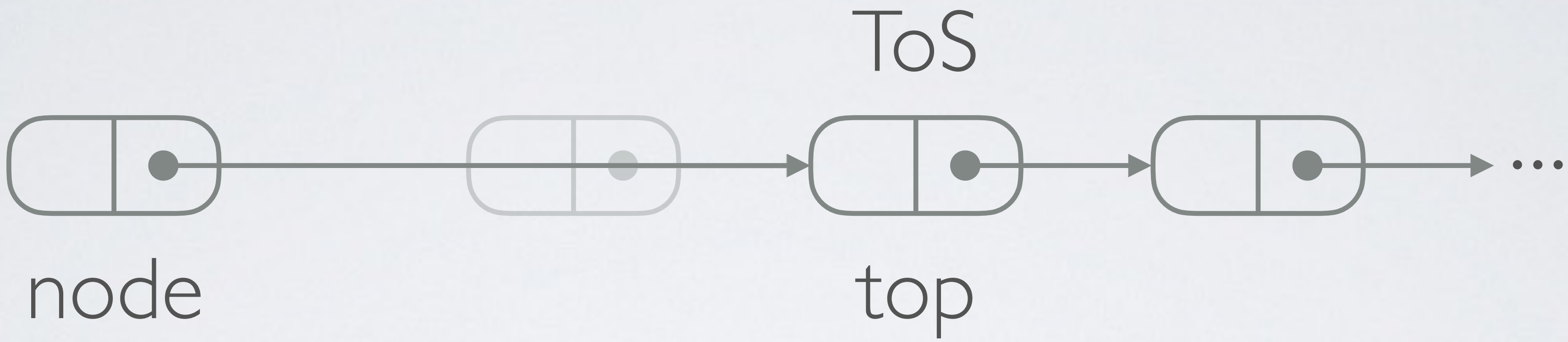
W. `CAS(ToS, top, node)`

Treiber's Stack



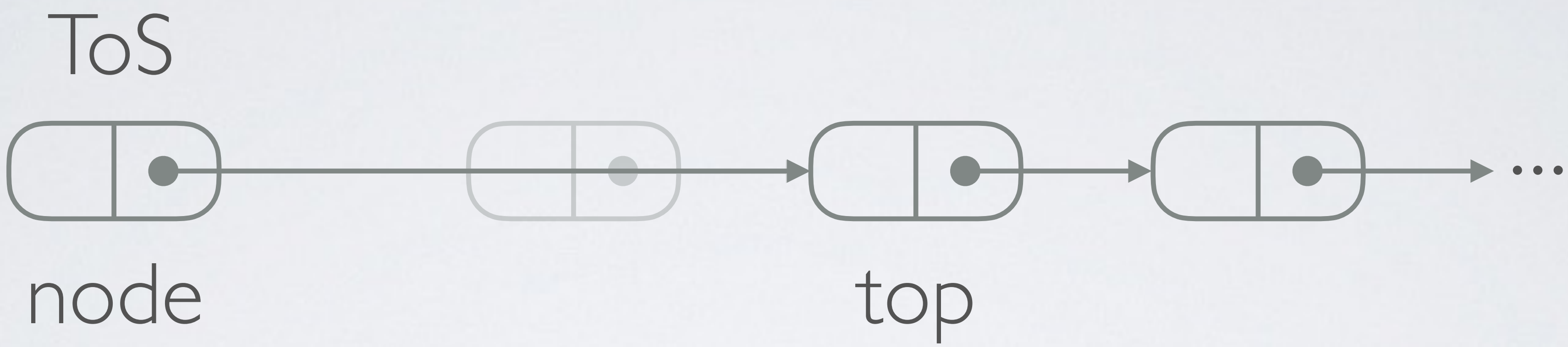
```
R.  top = ToS;

M.  node.next = top;

W.  CAS(ToS, top, node)
```

Treiber's Stack



```
R.  top = ToS;

M.  node.next = top;

W.  CAS(ToS, top, node)
```

Treiber's Stack

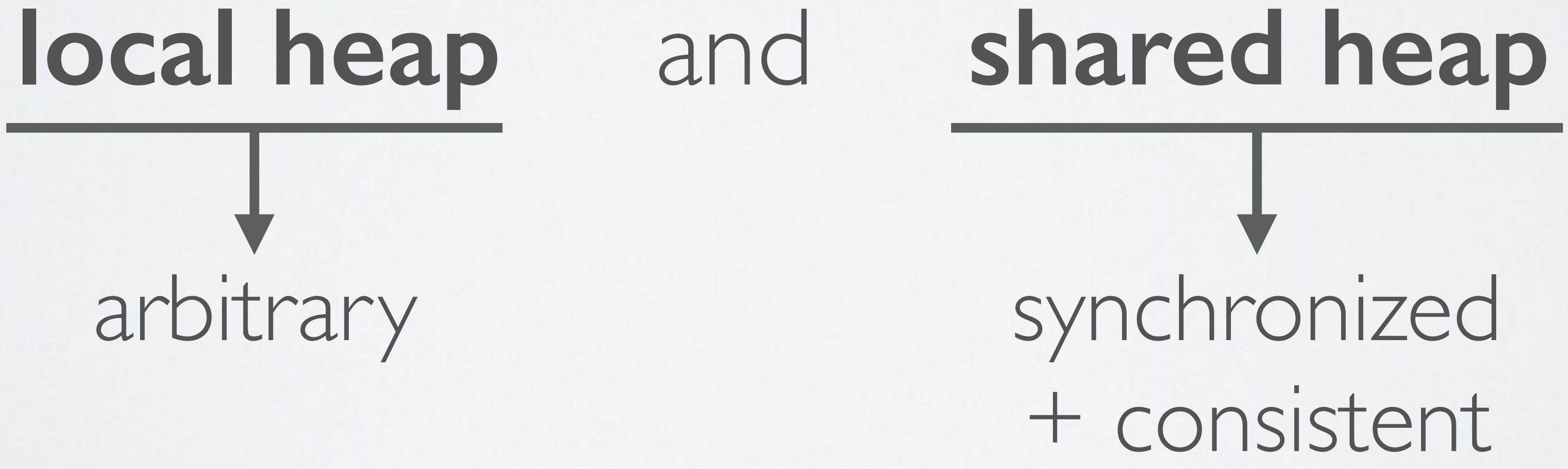


```
R.  top = ToS;

M.  node.next = top;

W.  CAS(ToS, top, node)
```

Observation

RMW inherent to lock-free programs.

Ensures modifications are restricted to:



local heap and **shared heap**

arbitrary

synchronized
+ consistent

Observation

Threads cannot observe the *atomicity* of other thread's RMW cycles.

RMW inherent to lock-free programs.

Ensures modifications are restricted to:

local heap and **shared heap**

arbitrary

synchronized + consistent

Implication on Analyses

- verify thread T in isolation
  - no-one tampers with T-local heap
  - prune interleavings
    - non-T RMW cycles are atomic
    - yet: same T configurations reachable

Observation II

Atomic RMW cycles are **stateless**:

- no local heap before *Read* phase
- relies solely on shared heap
- applies change definitely

Thread Summaries

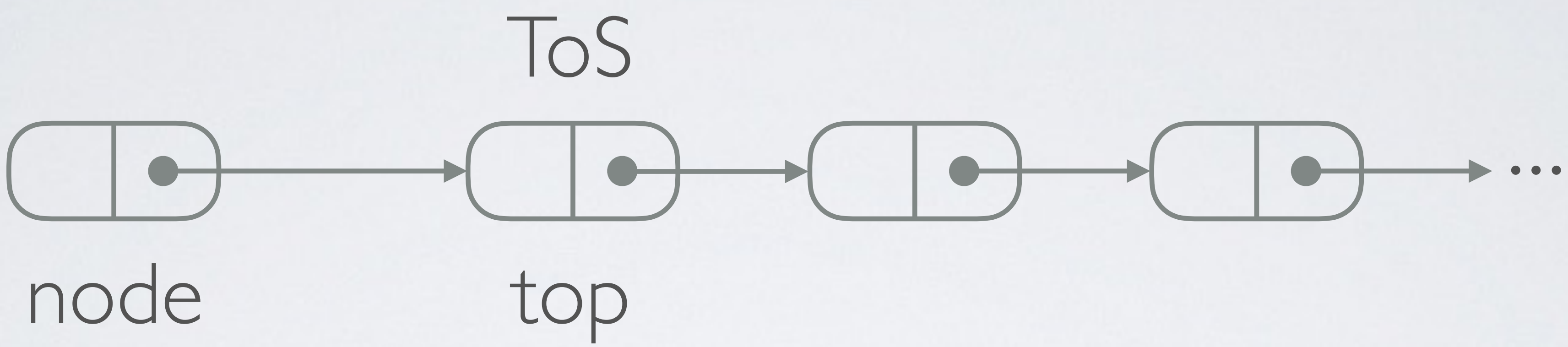
- apply the **effect** of atomic RMW cycles

- are stateless

➡ can be executed by a single thread

➡ so we analyse: T || S

Treiber's Stack again

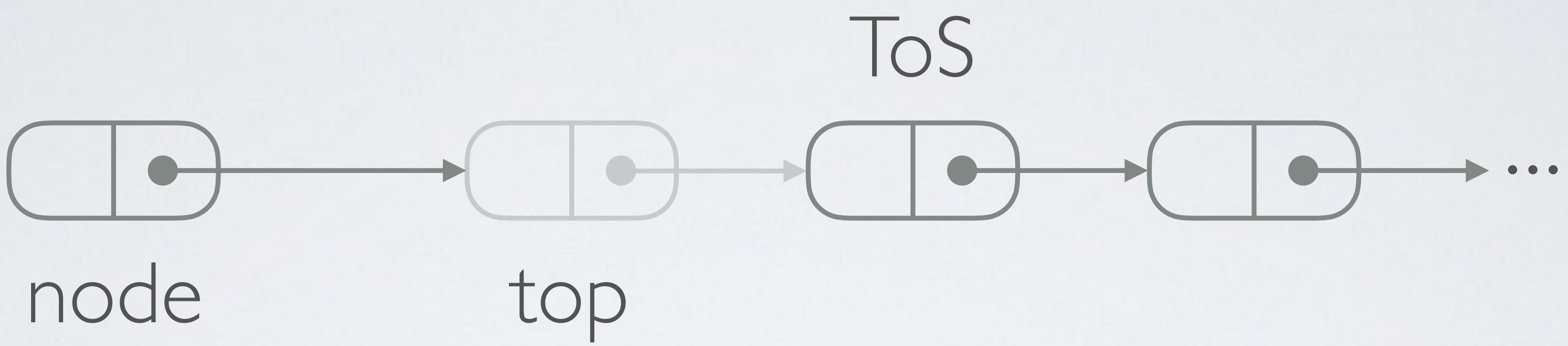


R. `top = ToS;`

M. `node.next = top;`

W. `CAS(ToS, top, node)`

Treiber's Stack again



```
R.  top = ToS;
M.  node.next = top;
    atomic {
      ToS = ToS.next;
    }
W.  CAS(ToS, top, node)
```

Soundness

- local heaps must be disjoint
  - ➡ ownership + no ownership violations [VMCAI16]

Soundness

- local heaps must be disjoint
  - ➡ ownership + no ownership violations [VMCAI16]
- summaries must be stateless

Soundness

- local heaps must be disjoint
  - ➡ ownership + no ownership violations [VMCAI16]
- summaries must be stateless
- summaries must cover all behaviours
  - ➡ check if S can mimic T actions on shared heap
  - ➡ can be done on-the-fly, interleaved with actual analysis

Evaluation

- adapted thread-modular analysis
  - apply summaries instead of interference
  - check correctness of summaries
- analyse linearizability of lock-free data structures
- C++, ~6000LOC, open source

Evaluation

	Treiber's Stack	Michael&Scott's Queue
Thread-Modular [VMCAI16]	25s	196m
Thread Summaries	1s	1m02s

:25

:190

Status

- done:
  - formalised summaries
  - proved reduction
- pending:
  - produce summaries
  - more experiments (mature tool)

Thanks.