Concurrent use of write-once memory

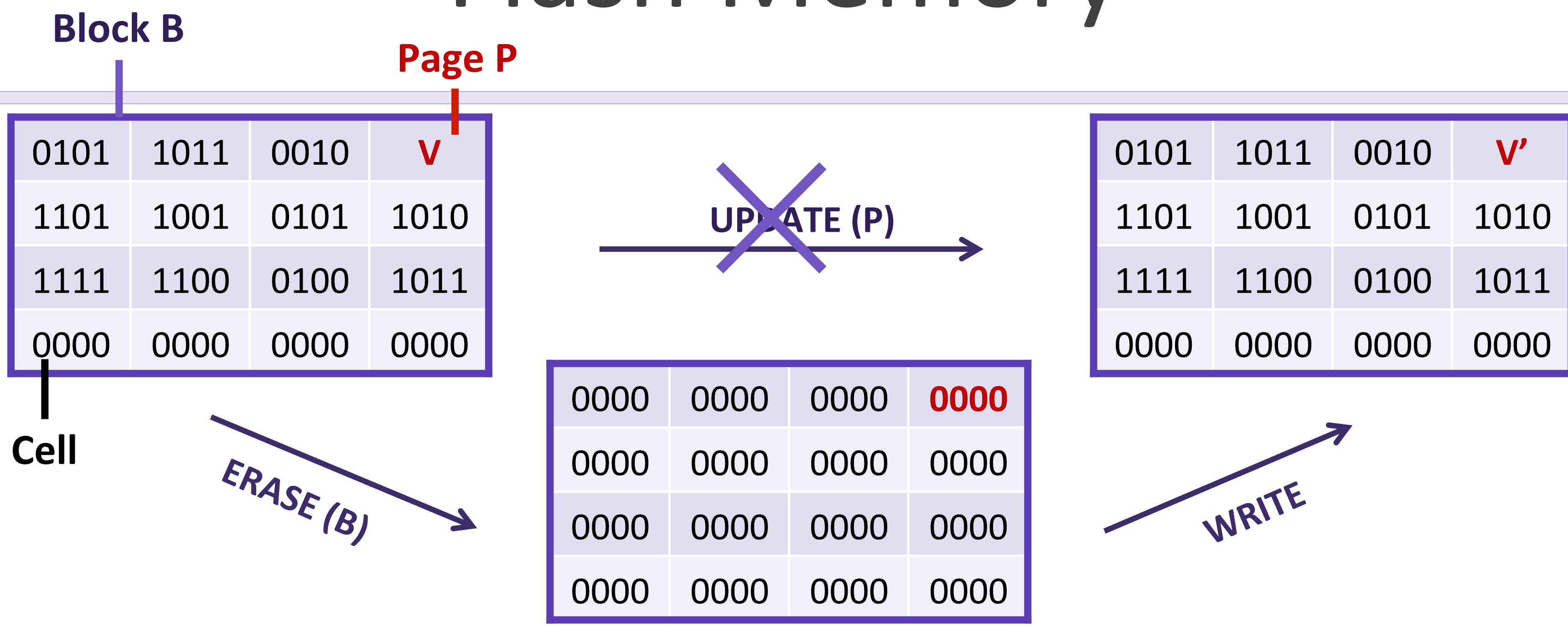
Keren Censor-Hillel, Technion

Joint work with:

James Aspnes, Yale University

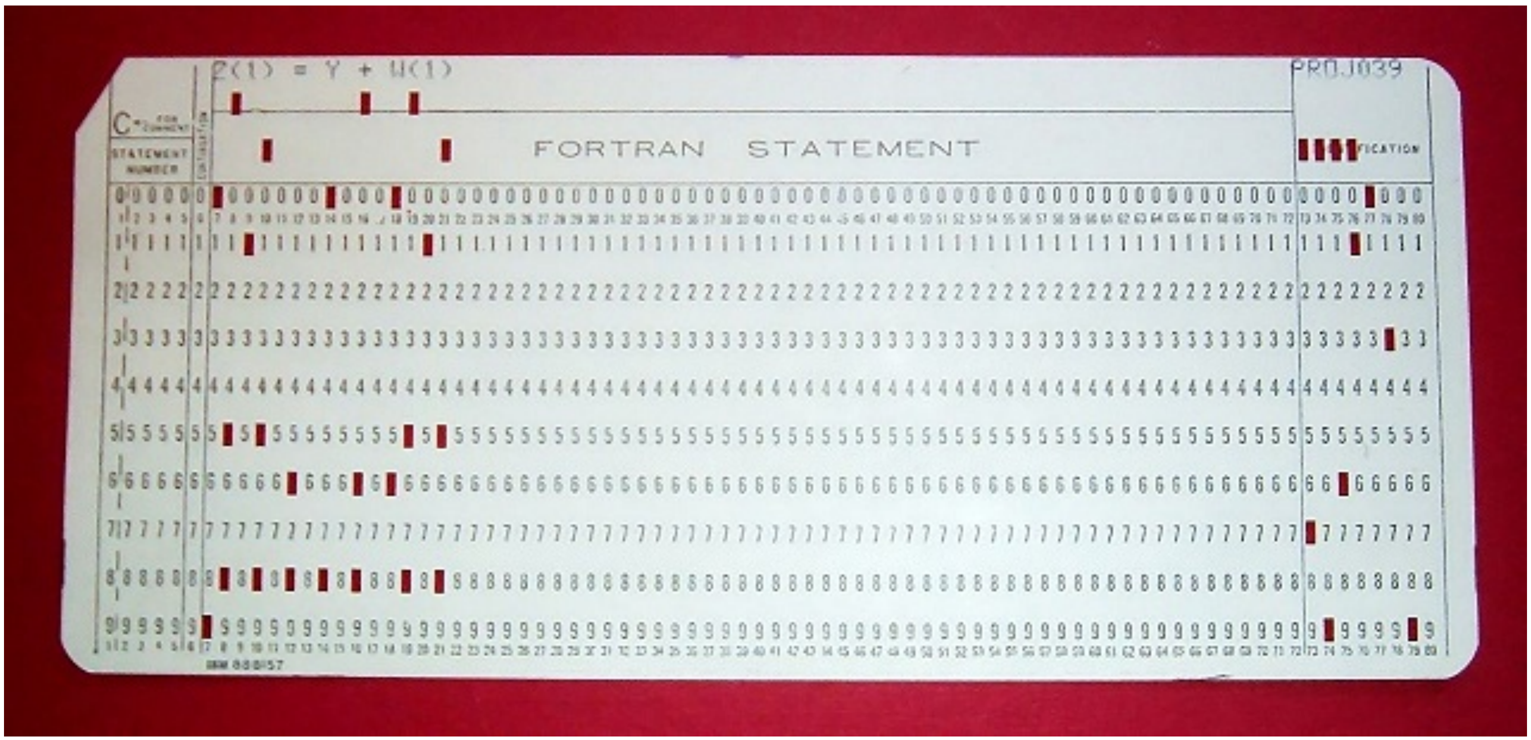
Eitan Yaakobi, Technion

Flash Memory



- Each cell represents one bit (initially 0)
- Read/Write pages quickly
- Erase before Write (no Updates)
- Erasing blocks is slow

Punch cards



Cannot “unpunch”

Write-once memory (WOM)

Memory is initialized to "0"s
Can only be updated to "1"s

Motivation from a theoretical viewpoint of concurrent algorithms:

- WOM has no **ABA** problems
  - ABA is hard to overcome
    need n-1 bounded registers to detect ABA
    [Aghazadeh and Woelfel 2015]

Write-once memory (WOM)

Memory is initialized to “0”s
Can only be updated to “1”s

Motivation from a theoretical viewpoint of concurrent algorithms:

- WOM is used by several implementations
  - **Sifters**
    [Alistarh and Aspnes 2011]
    [Giakkoupis and Woelfel 2012]
  - **Conflict detectors** [Aspnes and Ellen 2014]
  - **Max registers** [Aspnes, Attiya and C-H. 2012]

WOM codes

- **Goal 1**: allow fast updates

WOM codes

- **Goal 1**: allow fast updates
  - Save extra space for additional updates

Write 11 then 10

1	**1**	0	0

1	1	**1**	**0**

WOM codes

- **Goal 2**: Update with a small space overhead

WOM codes

- **Goal 2**: Update with a small space overhead
  - Can do **t** updates of **m** bits in less than **mt** space [Rivest and Shamir 1982]

Write 11 then 10

0	0	1

1	**0**	**1**

Data	First Write	Second Write
00	000	111
01	100	011
10	010	101
11	001	110

Concurrent WOM

- **With concurrency?**
  - Suppose **p1** writes **11** and **p2** writes **10**
    (So **p1** writes **001** and **p2** writes **010**)
  - Then **p3** reads

0	1	1

  - But **011** corresponds to **01** as a second write (which was never written)

Data	First Write	Second Write
00	000	111
01	100	011
10	010	101
11	001	110

Our Results: **m** bits, **t** writes

- Single-Writer-Multi-Reader
  - one write: **m+1** space
  - erasable one write: **m+2** space
  - **t** writes: **2m+t** space

Our Results: **m** bits, **t** writes

- Single-Writer-Multi-Reader
  - **$(1+o(1))t$** space for **$t=\omega(m2^m)$**, amortized step complexity **$O(n2^m)$** for write, **$O(2^m)$** for read

Our Results: **m** bits, **t** writes

- Single-Writer-Multi-Reader
  - **$(1+o(1))t$** space for **$t=\omega(m2^m)$**,
    amortized step complexity **$O(n2^m)$** for write, **$O(2^m)$** for read
- Multi-Writer-Multi-Reader
  - **$(2+o(1))t$** space for **$t=\omega((m+\log n)n^6 2^m)$**,
    amortized step complexity **$O(n^2 2^m)$** for write and read
  - unbounded space,
    amortized step complexity **$O(\log t+m+\log n)$**

Tabular WOM

← ℓ=O(m) → ← m →

count	Increment
1110000000000000	
1000000000000000	

↑ k=O(2^m) ↓

$$value = \left(\sum_{i=1}^{k} A[i].increment \cdot A[i].count \right) \bmod 2^m$$

$$increment = (v - value) \bmod 2^m$$

Tabular WOM

$\leftarrow \ell = O(m) \rightarrow$ $\longleftarrow m \longrightarrow$

count	Increment
1110000000000000	
1000000000000000	

$\updownarrow$ $k = O(2^m)$

Single-Writer: update count after increment

➔ New value is committed when setting a single bit

Tabular WOM

← ℓ=O(m) → ← m →

count	Increment
1110000000000000	
1000000000000000	

↑ $k=O(2^m)$ ↓

Multi-Writer: use reduction of [Israeli and Shaham 2005]
Uses **n** SWSR objects → space per write could be **Θ(n)**

Tabular WOM

← ℓ=O(m) → ← m →

count	Increment
1110000000000000	
1000000000000000	

↑ $k=O(2^m)$ ↓

Multi-Writer: use reduction of [Israeli and Shaham 2005]

Uses **n** SWSR objects → space per write could be **Θ(n)**

Solution: use single table, allocate rows dynamically

Tabular WOM

← ℓ=O(m) → ← m →

safe-agreement	count	increment
	1110000000000000	
	1000000000000000	

↑ k=O(2^m) ↓

Multi-Writer: use reduction of [Israeli and Shaham 2005]
Uses **n** SWSR objects ➔ space per write could be **Θ(n)**
Solution: use single table, allocate rows dynamically [Borowski et al. 2001]

Tabular WOM

← $\ell=O(m)$ → ← m →

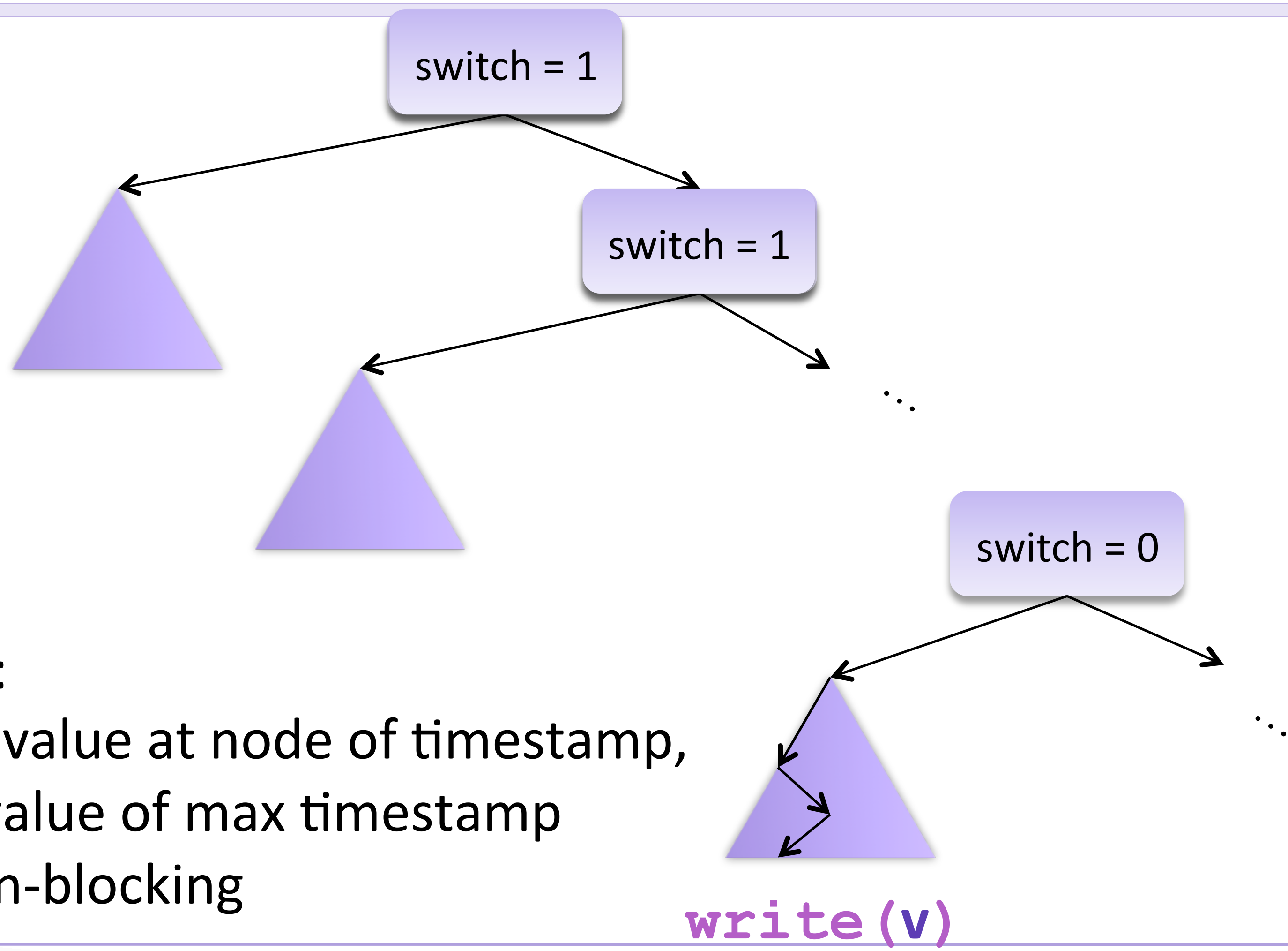
safe-agreement	count	increment
	1110000000000000	
	1000000000000000	

↑ $k=O(2^m)$ ↓

Solution: use single table, allocate rows dynamically [Borowski et al. 2001]

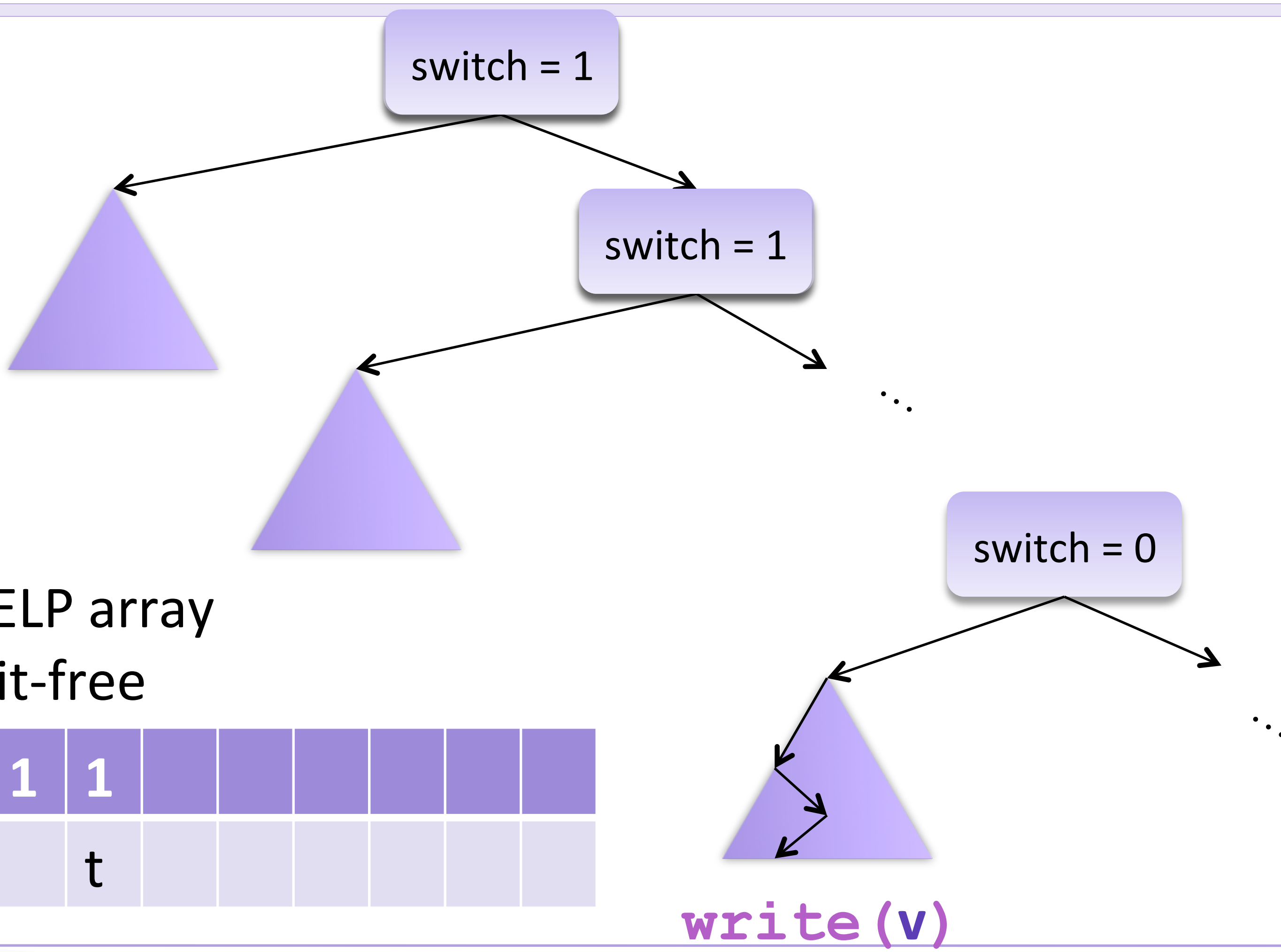
safe-agreement objects can get stuck, interleave **n** objects

Max-Register Based WOM



SWSR:
Write value at node of timestamp,
read value of max timestamp
➔ non-blocking

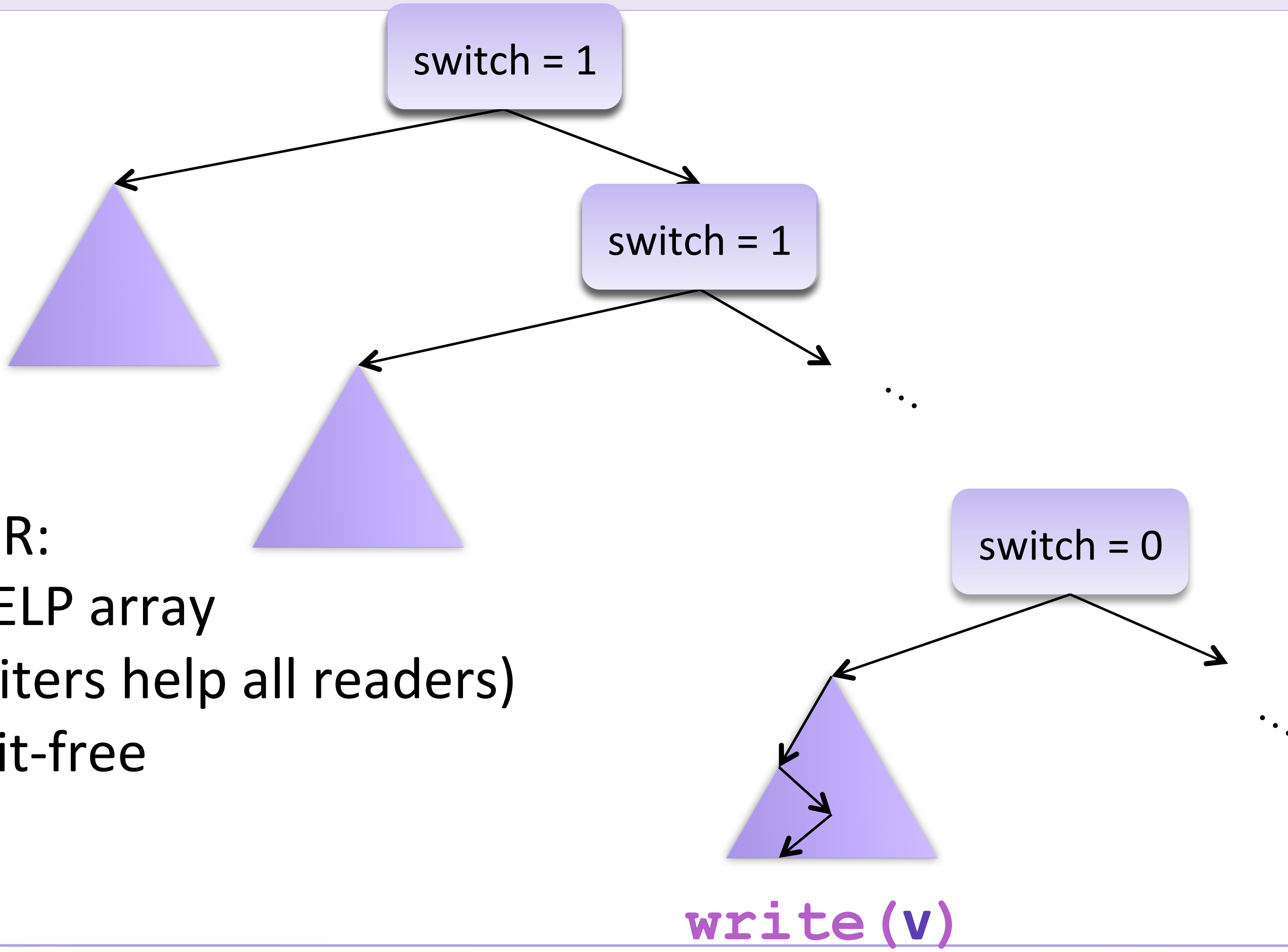
Max-Register Based WOM



SWSR:
Use HELP array
➔ wait-free

1	1	1	1						
			t						

Max-Register Based WOM



MWMR:
Use HELP array
(all writers help all readers)
➔ wait-free

Discussion

- Algorithms for concurrent WOM
- Open questions:
  - Combine low space and low time overheads?
  - Stronger primitives?
    - Test-and-set (non-resettable)