Concurrent Order Maintenance

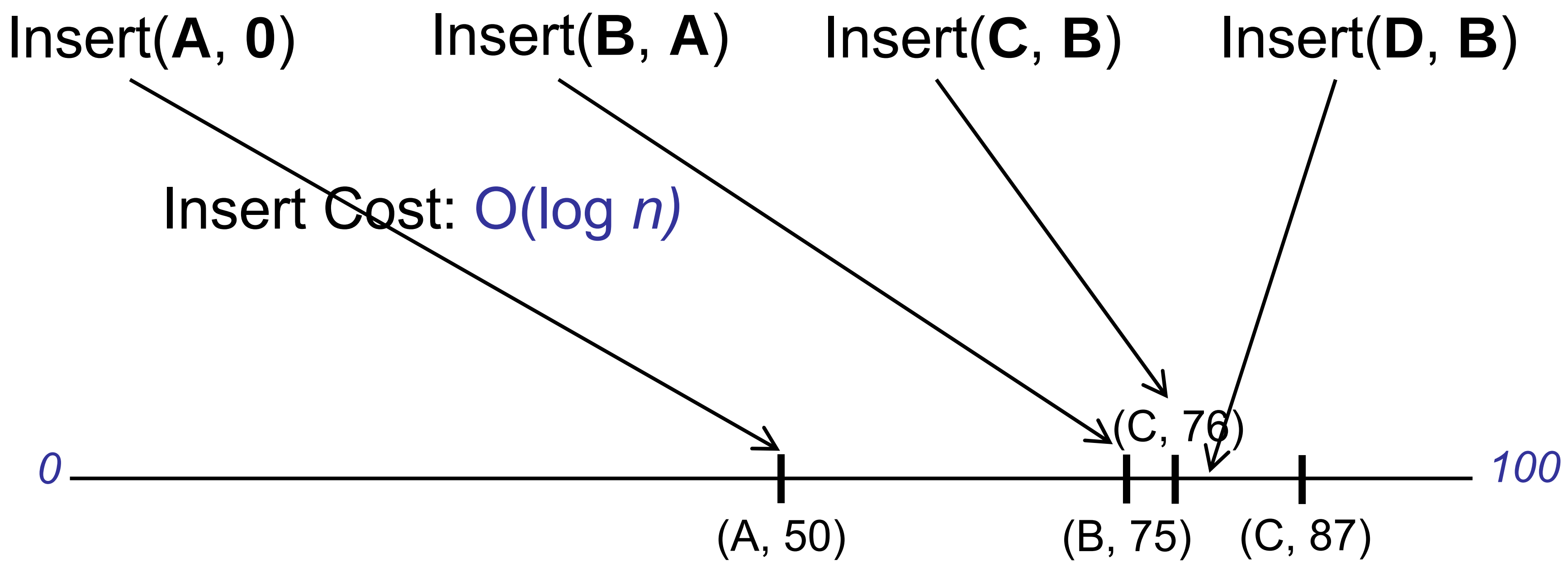
Seth Gilbert
(Collaboration with
Jeremy Fineman and Michael Bender)

Order Maintenance

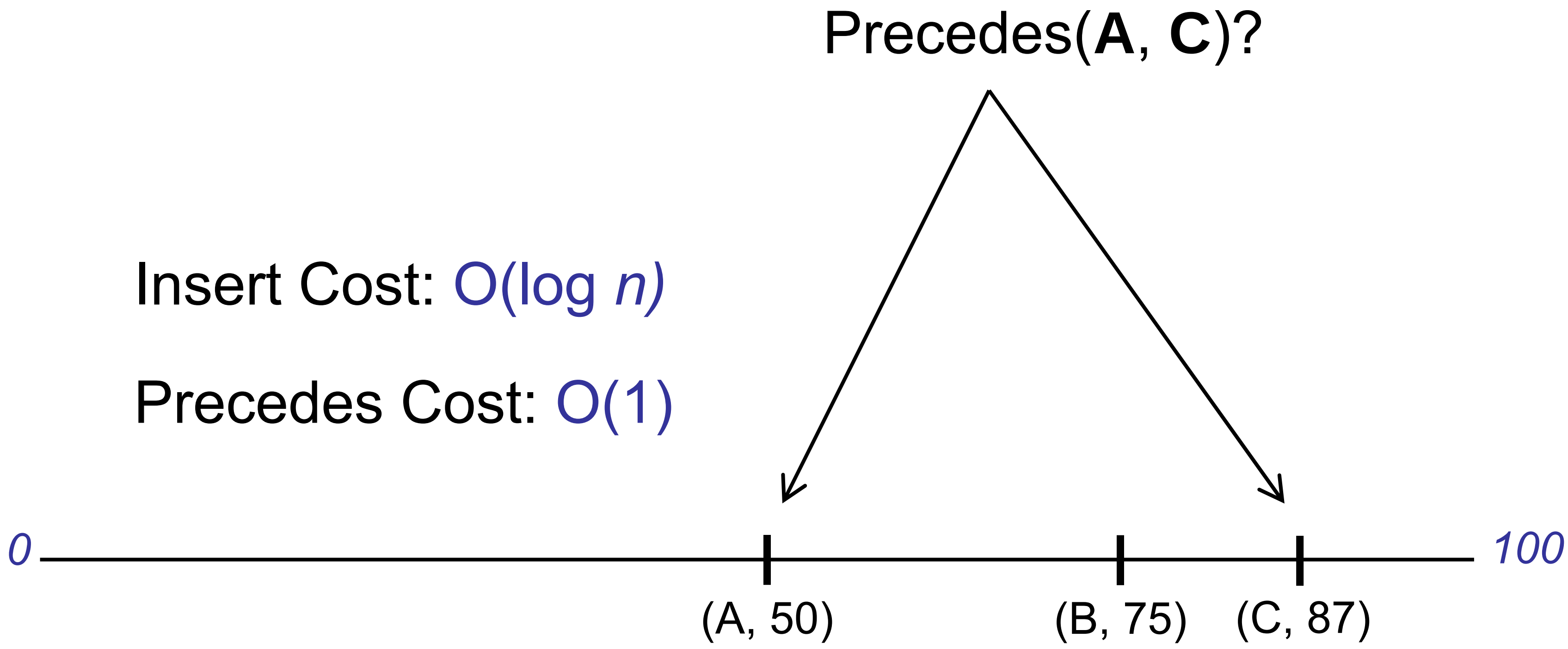
- Problem:
  - Insert(**Item**, **Predecessor**)
    - Inserts **Item** after **Predecessor**
    - Returns pointer to item
  - Precedes(**A**, **B**)
    - Does item **A** precede item **B**?
- Solutions:
    - Dietz, Sleator, *Order Maintenance Problem,* 1987
    - Bender, Cole, Demaine, Farach-Colton, Zito, 2002

Example

Insert Cost: O(log *n*)



Example



Insert Cost: O(log *n*)

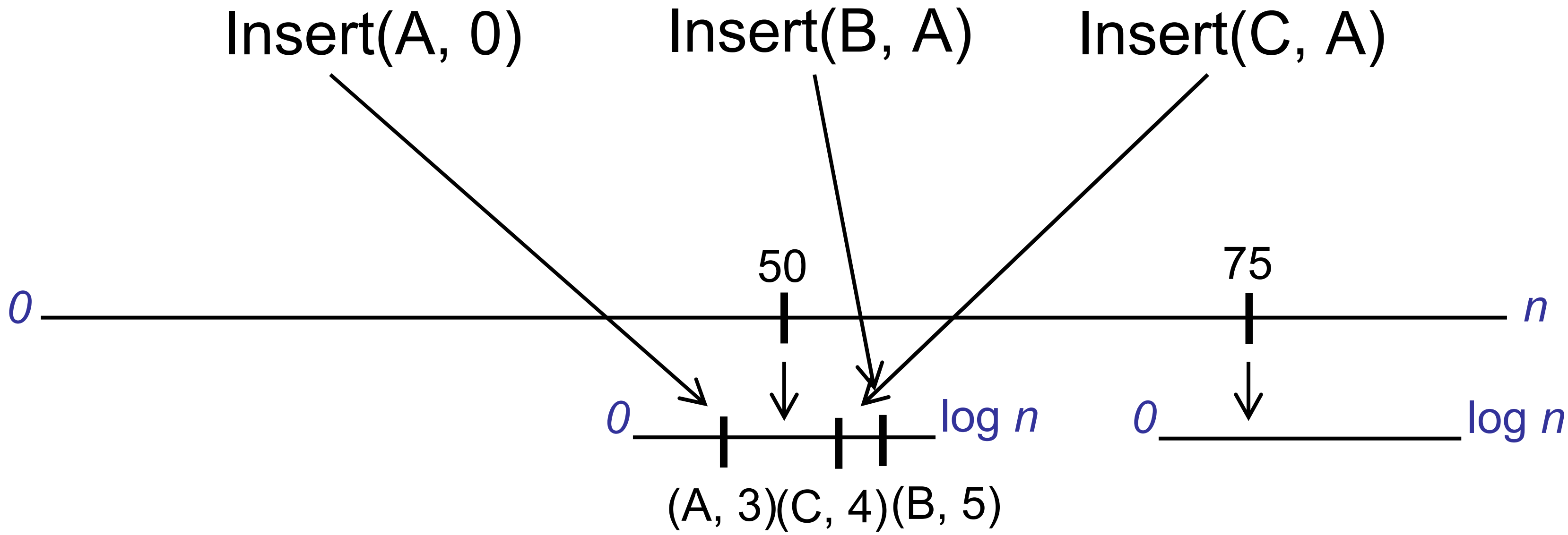
Precedes Cost: O(1)

Outline

- Introduction
- **Indirection**
- Results – Total Work
  - $O\left(\sqrt{\log p}\left(T_1 + p \cdot T_\infty\right)\right)$
  - $O\left(\frac{1}{\varepsilon}\left(T_1 + p^{(1+\varepsilon)} \cdot T_\infty\right)\right),\ 0 < \varepsilon \leq 0.5$
- Non-blocking Implementations
- Conclusion

Getting Constant Time

- Maintain $n/\log n$ lists of size $\log n$



Getting Constant Time

- $O(n \cdot \log n)$ to insert n items
- Maintain $n/\log n$ lists of size $\log n$

$$\frac{n}{\log n} \times \log n = O(n)$$

- Maintain lists of size $\log n$
  - Easy!
  - No reorganization => constant time ops
  - Each insert divides tag space in half…

Outline

- Introduction
- Indirection
- **Results – Total Work**
  - $O\left(\sqrt{\log p}\left(T_1 + p \cdot T_\infty\right)\right)$
  - $O\left(\frac{1}{\varepsilon}\left(T_1 + p^{(1+\varepsilon)} \cdot T_\infty\right)\right)$, $0 < \varepsilon \leq 0.5$
- Non-blocking Implementations
- Conclusion

Parallel Problems

- Lock during inserts?
  - Queries are still fast
  - Inserts may be slow

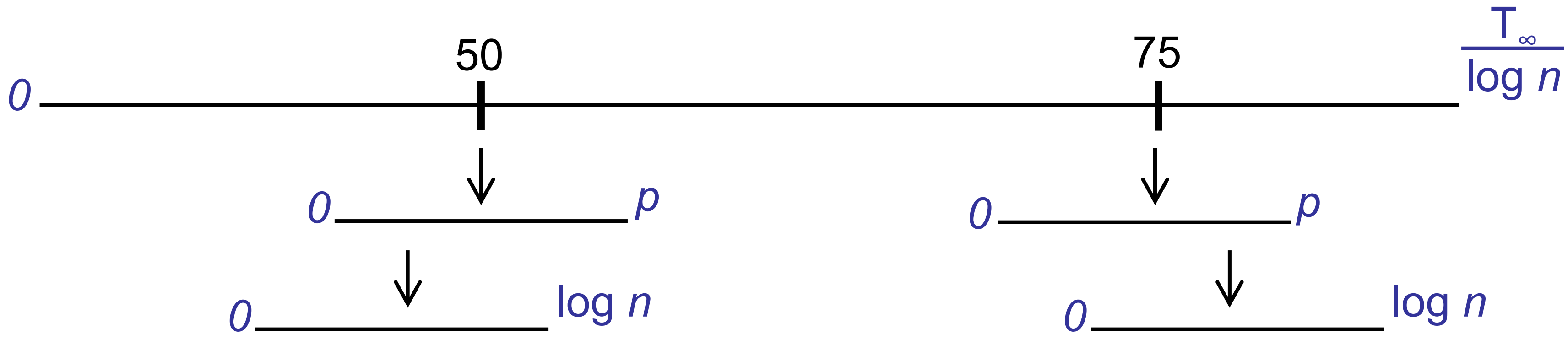
- Focus on Cilk graph (Non-Determinator)
  - ≤ T_1 Precedes queries
  - ≤ $p \cdot T_\infty$ Insert ops (steal attempts)
- Desired goal: $O(T_1 + p \cdot T_\infty)$ work
- Reality: slower…

Applications

- Non-Determinator
- Cache-oblivious B-Trees
- Distributed Search Data Structures

Small Number Processors

- If $p \leq \log n$, use indirection



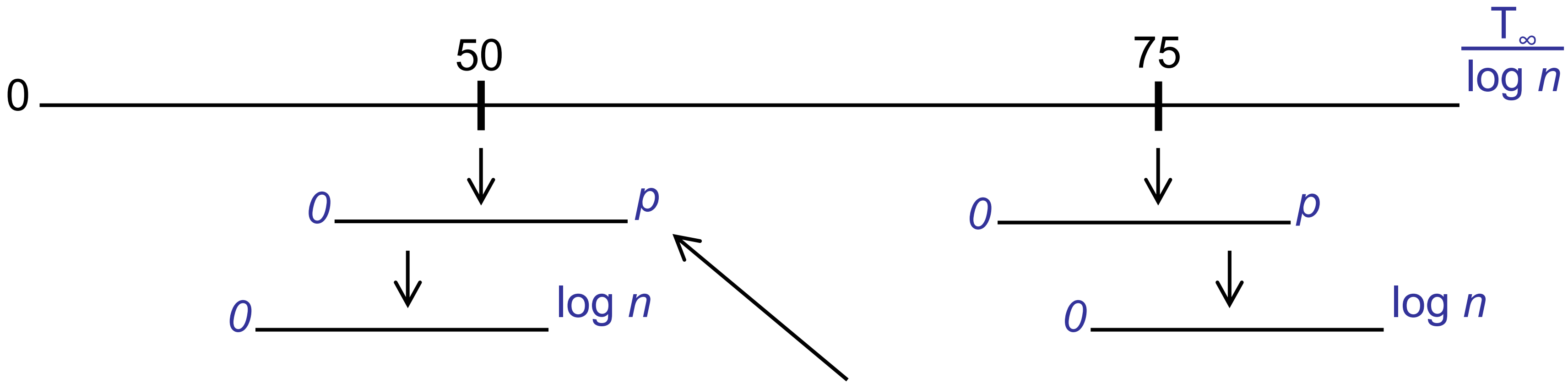
  - Waiting time per processor:

$$\frac{T_\infty}{\log n} \times \log n = O(T_\infty)$$

  - Total waiting time: $O(p \cdot T_\infty)$

One Level Indirection

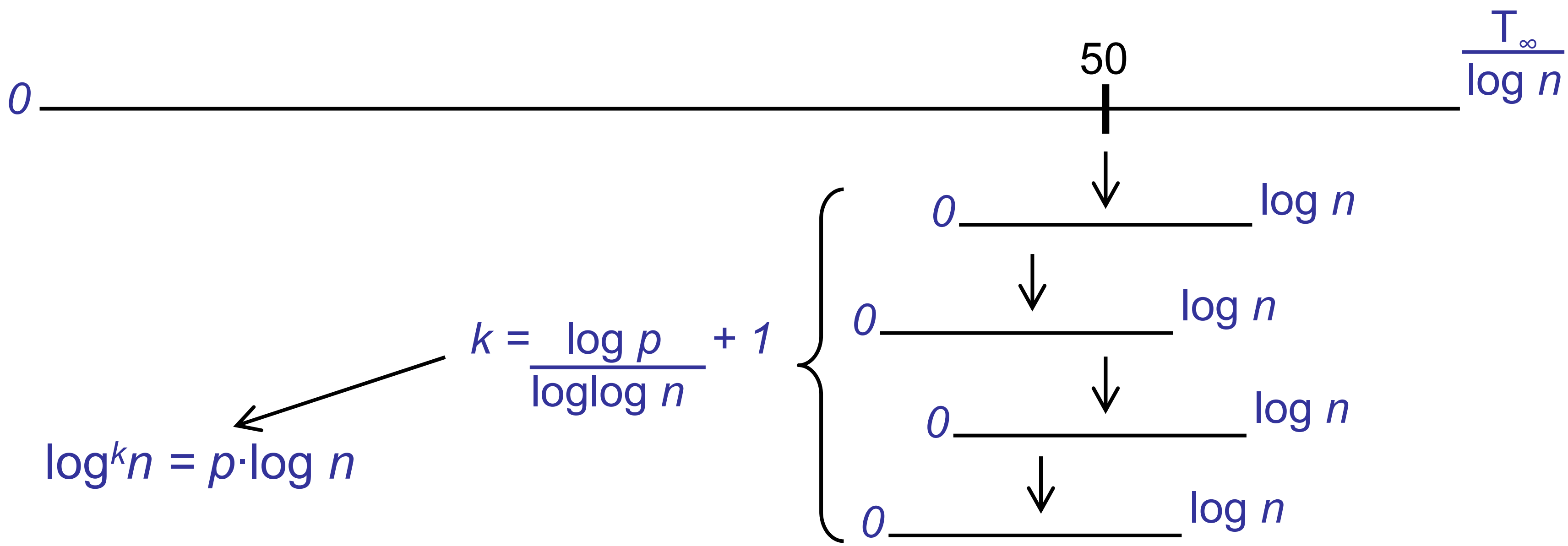
- Assume p is not so small



- How expensive is ordering p elements?
  - Waiting time per insert $O(p)$
  - Total waiting time: $p^2 \cdot T_\infty$
- Total work: $O(T_1 + p^2 \cdot T_\infty)$

More Indirection

- If $p > \log n$, but not too big:



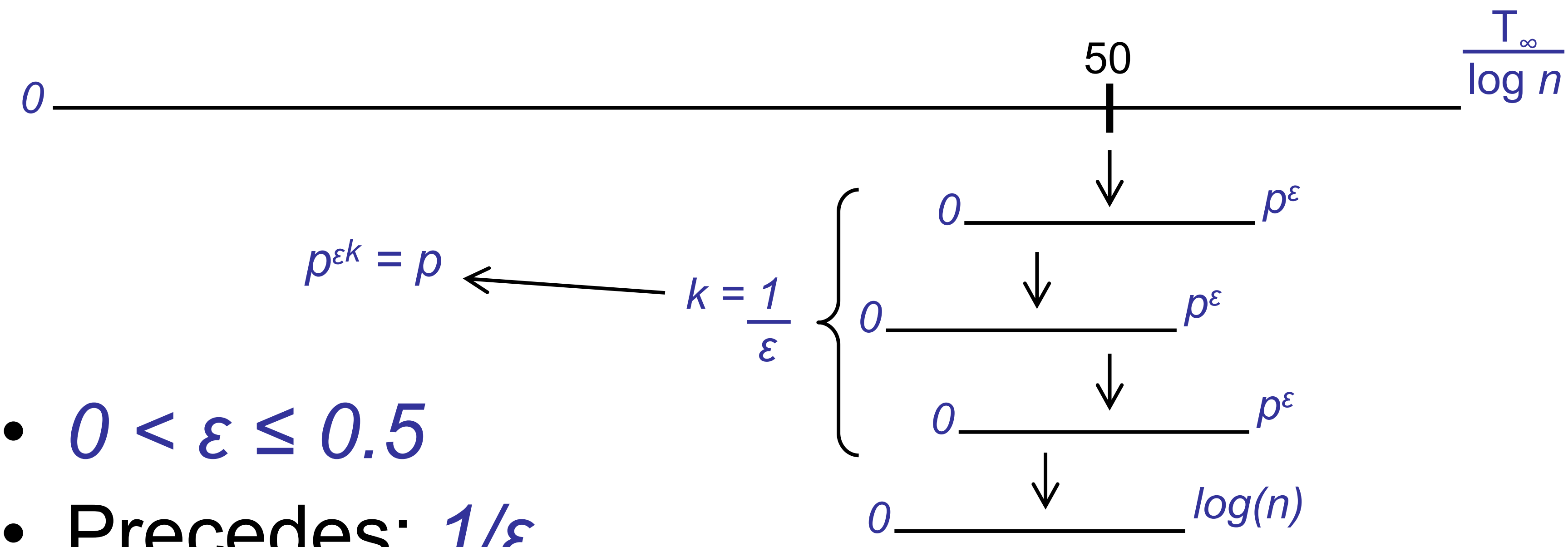
- Precedes: $O(\log p)$
- Total: $O\left[\log p \left[T_1 + p \cdot T_\infty\right]\right]$

Better when:

$$\frac{T_1}{T_\infty} < \frac{p^2}{\log p}$$

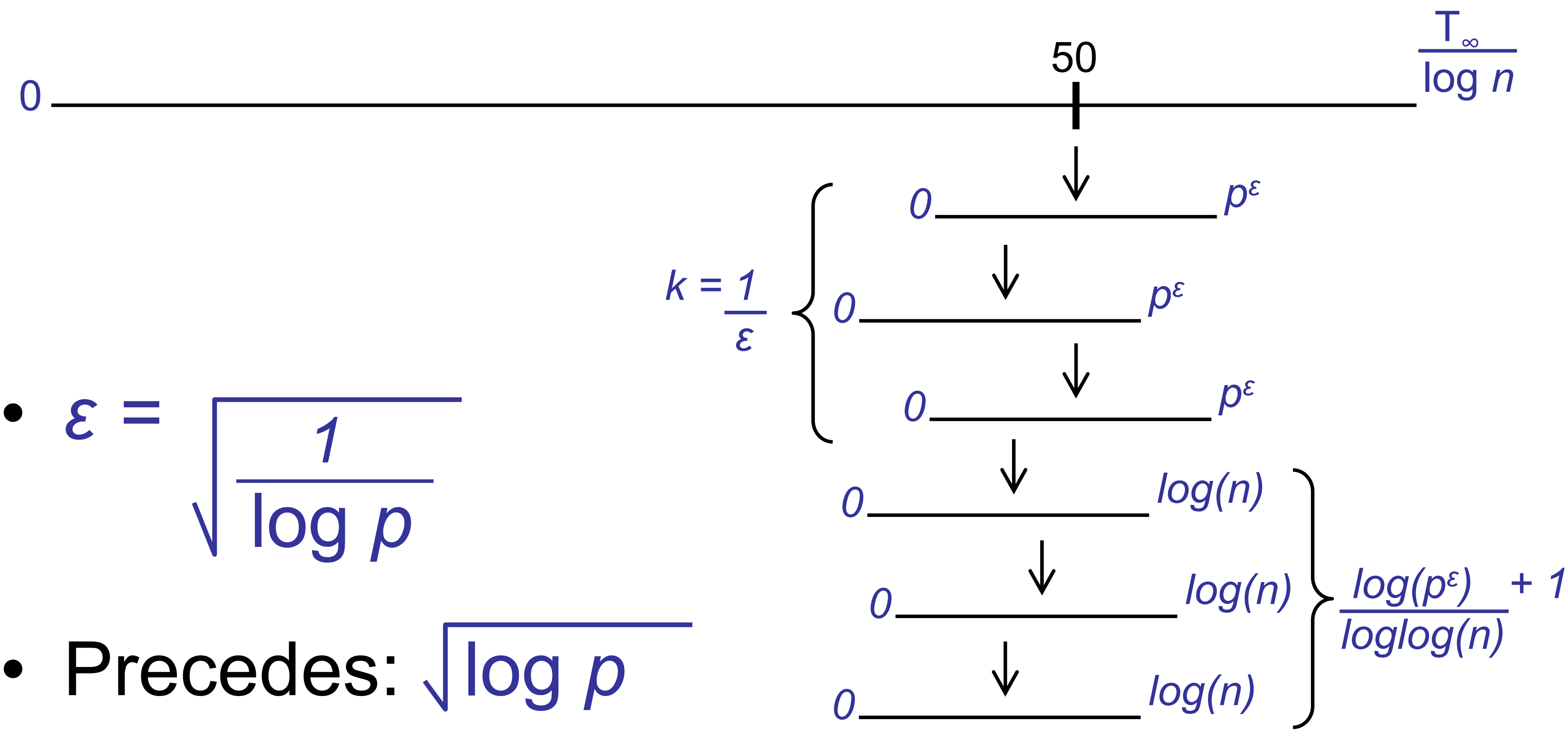
Variable Indirection

- Trade-off between queries and inserts:



- $0 < \varepsilon \leq 0.5$
- Precedes: $1/\varepsilon$
- Total: $O\left(\frac{1}{\varepsilon}\left(T_1 + p^{(1+\varepsilon)} \cdot T_\infty\right)\right)$

More Indirection (Again)



- $\varepsilon = \sqrt{\dfrac{1}{\log p}}$
- Precedes: $\sqrt{\log p}$
- Total: $O\left[\sqrt{\log p}\left[T_1 + p \cdot T_\infty\right]\right]$

Outline

- Introduction
- Indirection
- Results – Total Work
  - $O\left(\sqrt{\log p}\left(T_1 + p \cdot T_\infty\right)\right)$
  - $O\left(\frac{1}{\varepsilon}\left(T_1 + p^{(1+\varepsilon)} \cdot T_\infty\right)\right)$, $0 < \varepsilon \leq 0.5$
- **Non-blocking Implementations**
- Conclusion

Non-Blocking

- Assume DCAS
  - Compares two addresses with old values
  - DCAS(A, B, old-A, old-B, new-A, new-B)
    - if ((*A == old-A) && (*B == old-B))

      *A = new-A

      *B = new-B
- Lock-freedom / Obstruction-freedom
  - Some operation is always able to make progress
- Start with linked-list implementation

Concurrent Reorganization

- How to ensure that operations make progress?
  - Precedes queries can always proceed
  - Always renumber monotonically increasing

- How to ensure Insert does not interfere?
  - Increment “owner” field of predecessor
  - Only Insert or Renumber if you own the predecessor
  - Backoff

Conclusion

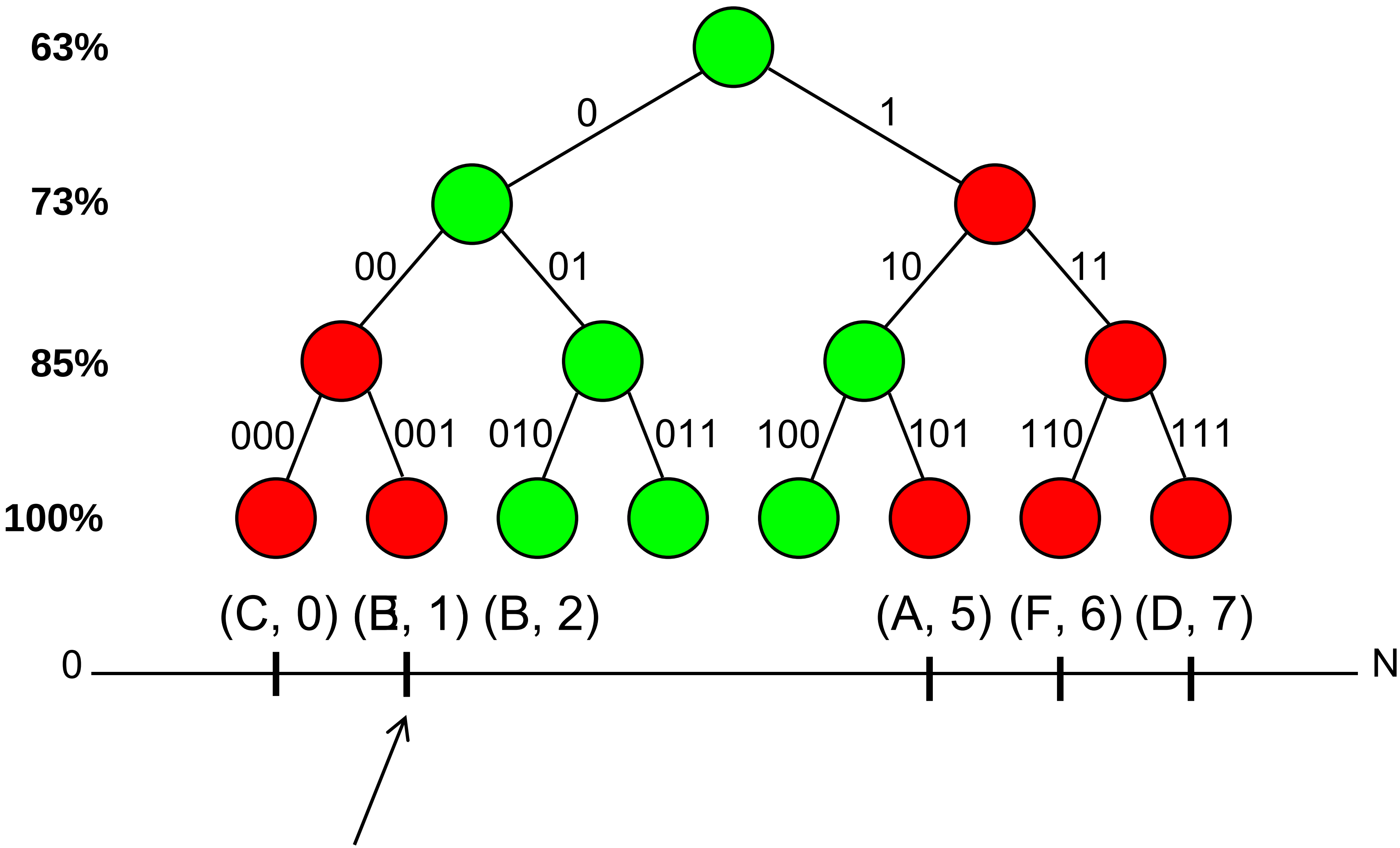
- Concurrent Order Maintenance
  - T_1 Precedes queries
  - pT_∞ Inserts (steals)
  - p Processors
- Results – Total Work
  - $O\left(\sqrt{\log p}\left[T_1 + p \cdot T_\infty\right]\right)$
  - $O\left(\frac{1}{\varepsilon}\left(T_1 + p^{(1+\varepsilon)} \cdot T_\infty\right)\right),\ 0 < \varepsilon \leq 0.5$

Conclusion

- Concurrent Order Maintenance
  - T_1 Precedes queries
  - pT_∞ Inserts (steals)
  - p Processors
- Results – Total Work
  - $O\left(\log\log p\left(T_1 + p \cdot T_\infty\right)\right)$
  - $O\left(\frac{1}{\log \varepsilon}\left(T_1 + p^{(1+\varepsilon)} \cdot T_\infty\right)\right)$, $0 < \varepsilon \leq 0.5$

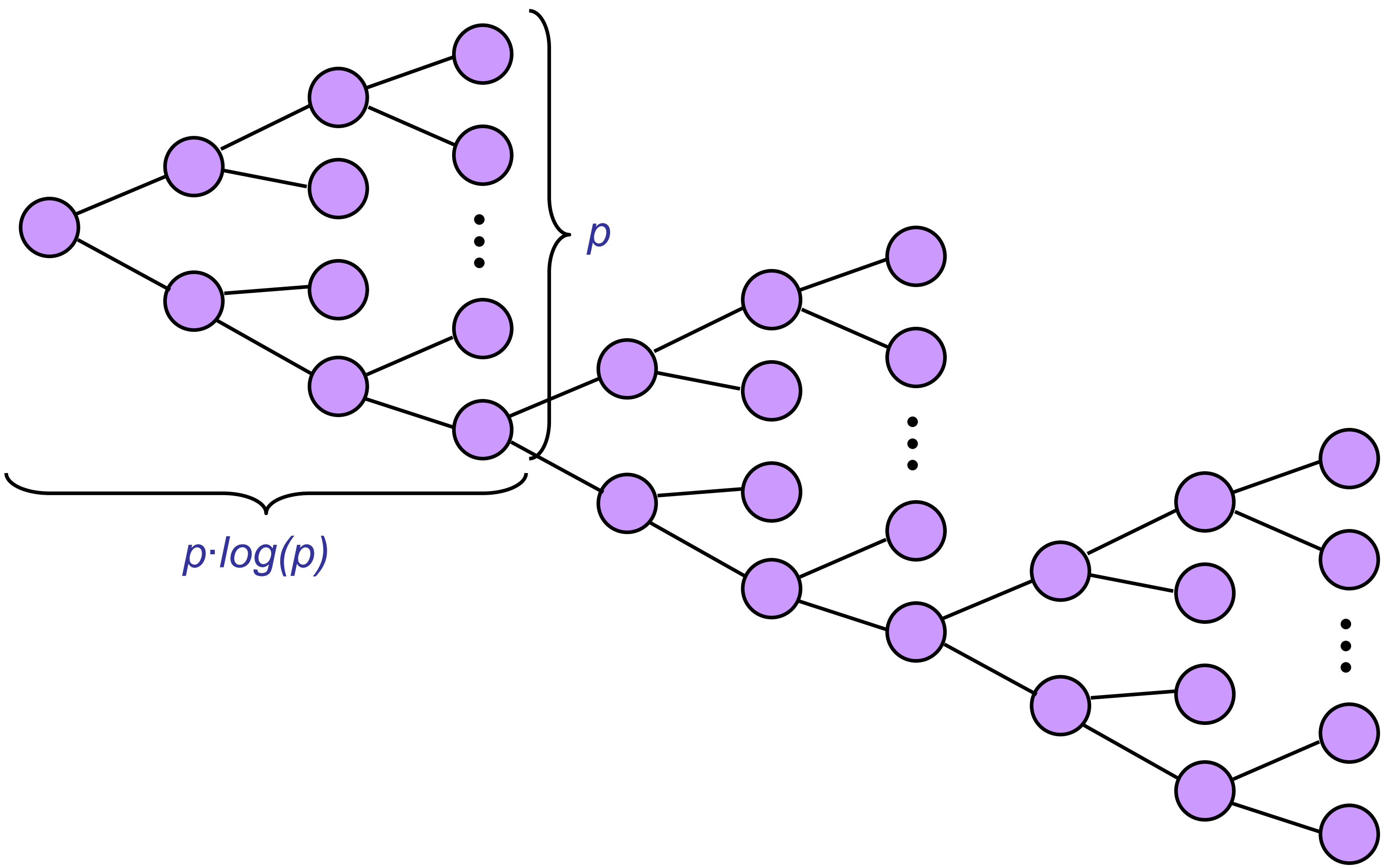
Backup Slides

Binary Tree



63%
73%
85%
100%
0
1
00
01
10
11
000
001
010
011
100
101
110
111
(C, 0) (B, 1) (B, 2)
(A, 5) (F, 6) (D, 7)
0
N

Bad Example



Why does it work?

- Concurrent reorganization can only help
- Successful insert implies some processor made progress
  - No worse than starting *after* insert completes
- At worst, same as locking:
  - Begin after operation completes