Problem

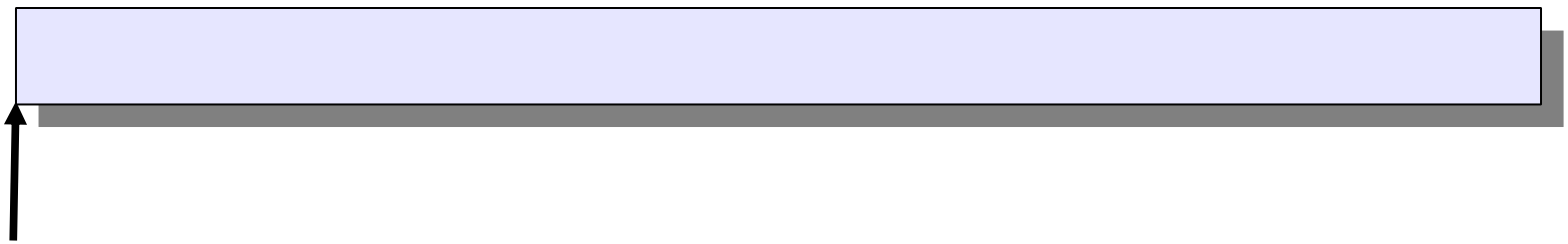
- Parallelize (serial) applications that use files.
  - Examples: compression tools, logging utilities, databases.
- In general
  - applications that use files depend on sequential output,
  - **serial append** is the usual file I/O operation.
- Goal:
  - perform file I/O operations in parallel,
  - keep the sequential, serial append of the file.

Results

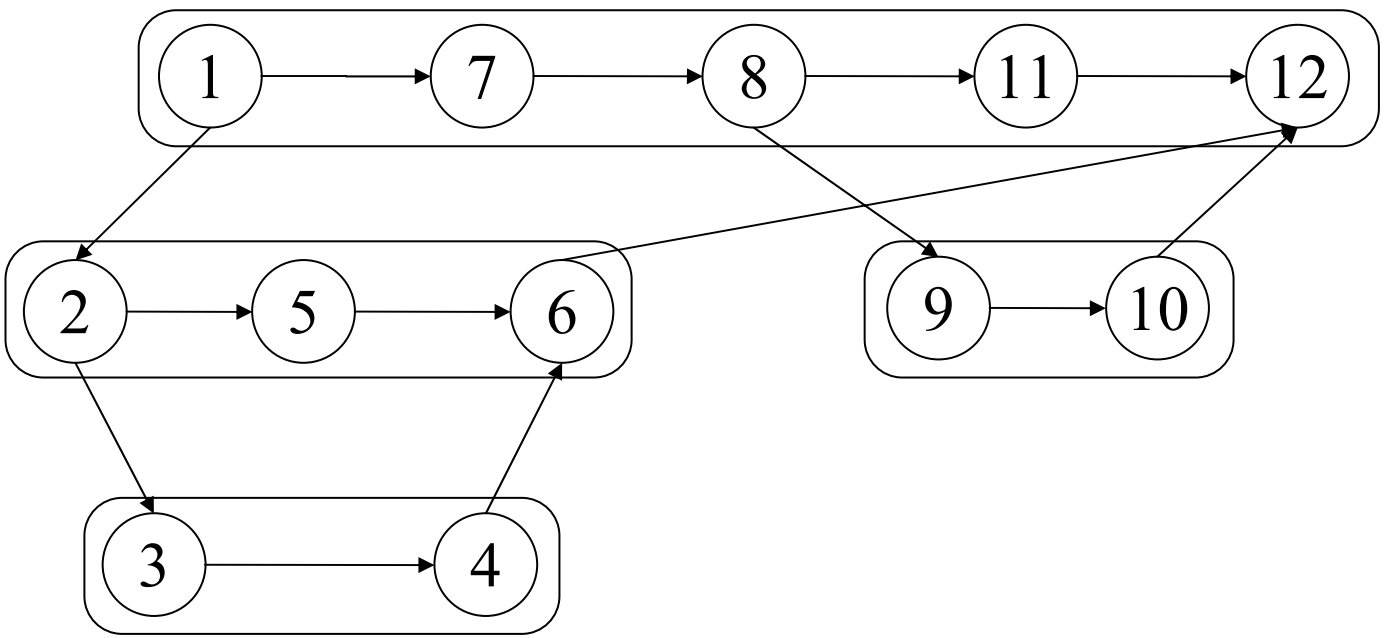
- *Cilk* **runtime-support** for serial append with good scalability.
- Three **serial append** schemes and implementations for *Cilk*:
  1. ported *Cheerio*, previous parallel file I/O API (*M. Debergalis*),
  2. simple prototype (with concurrent Linked Lists),
  3. extension, more efficient data structure (concurrent double-linked Skip Lists).
- Parallel bz2 using ***PLIO***.

Single Processor Serial Append

FILE (serial append)

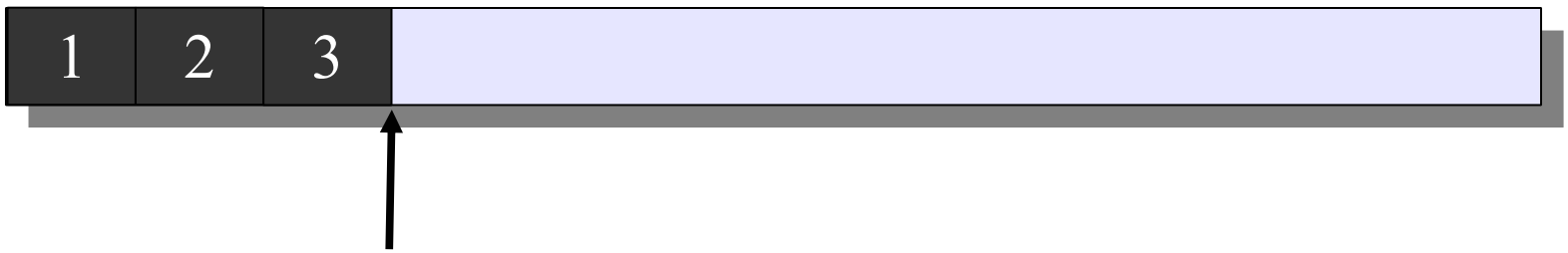


computation DAG

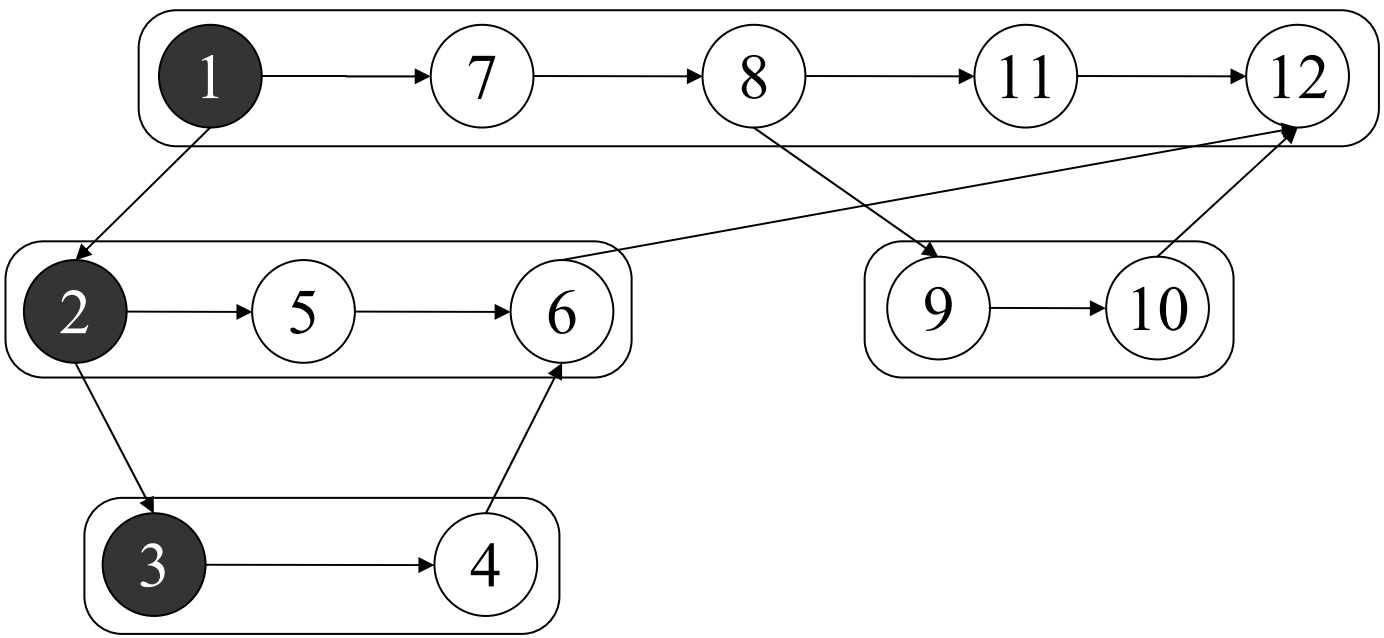


Single Processor Serial Append

FILE (serial append)

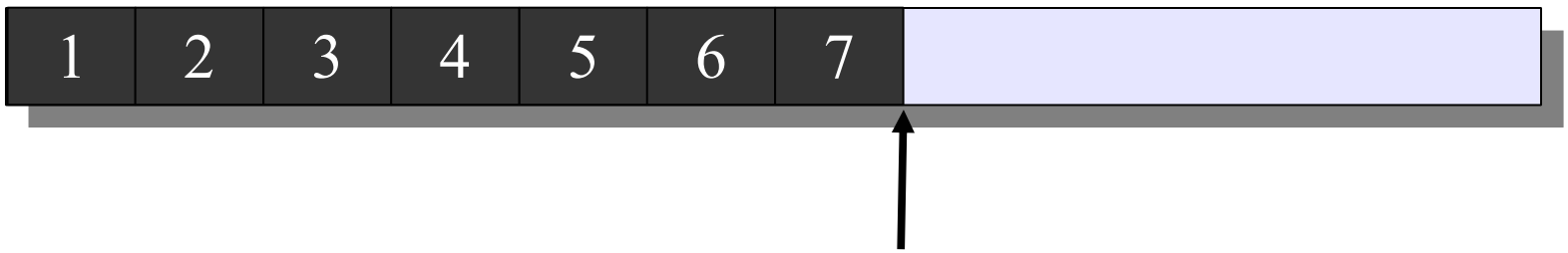


computation DAG

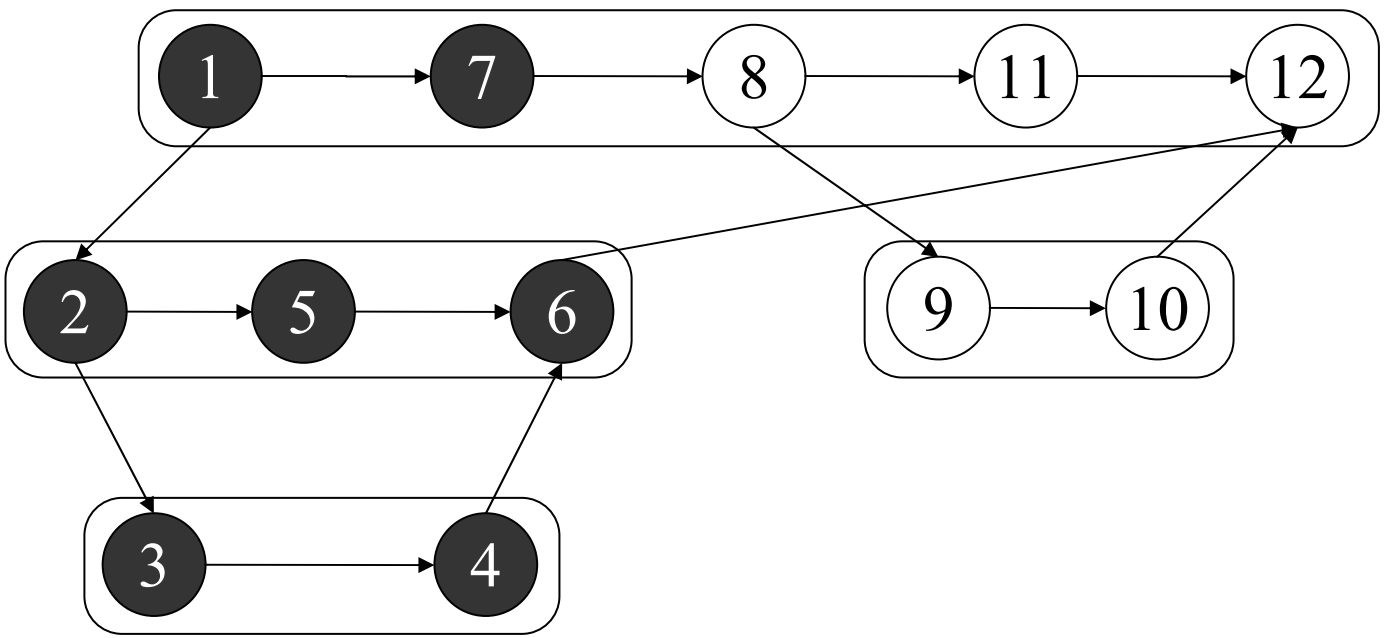


Single Processor Serial Append

FILE (serial append)

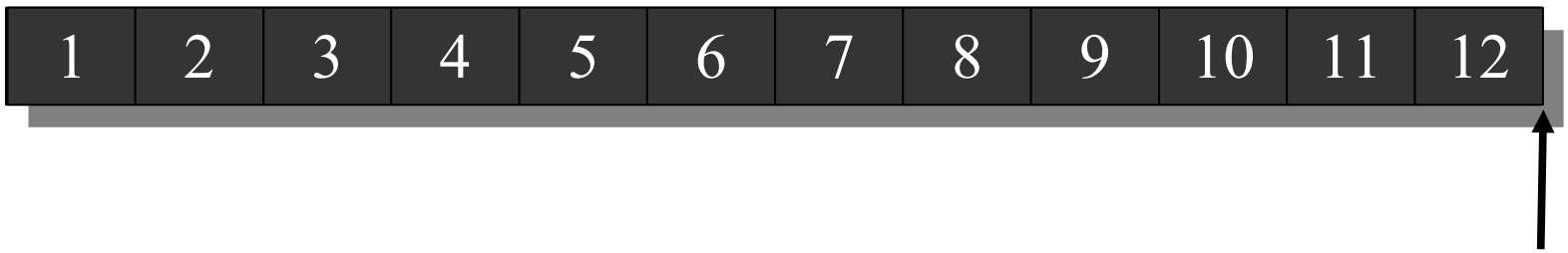


computation DAG

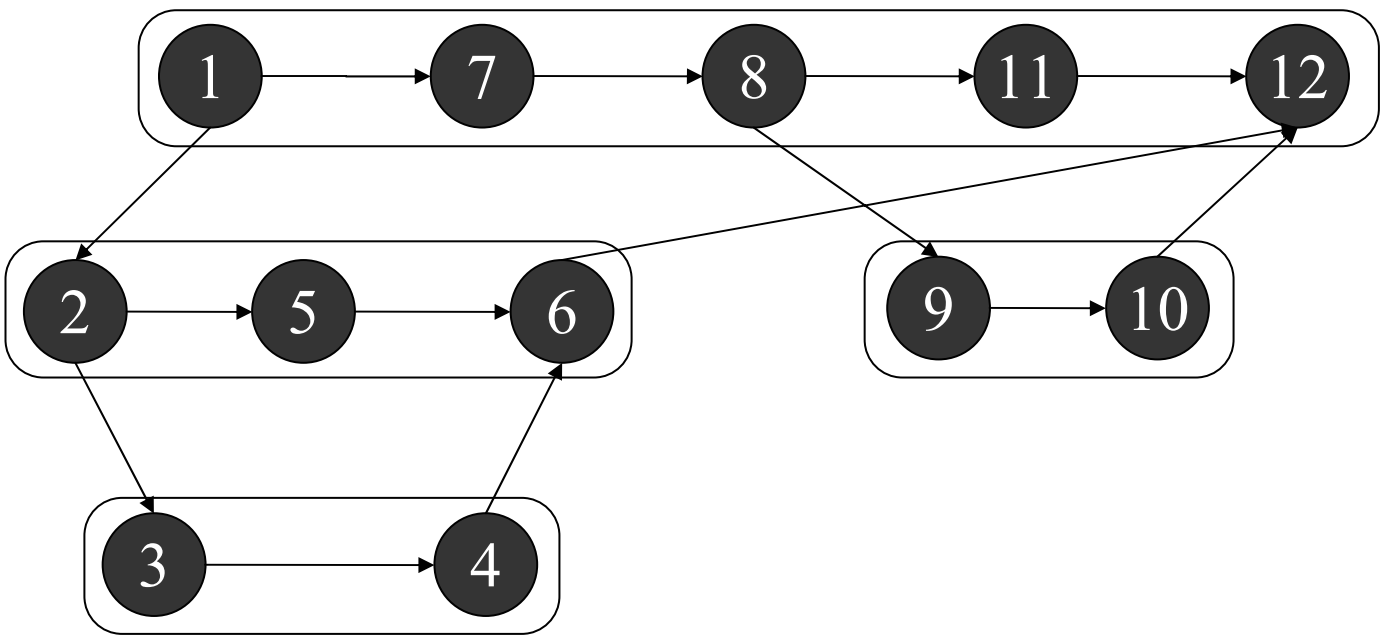


Single Processor Serial Append

FILE (serial append)

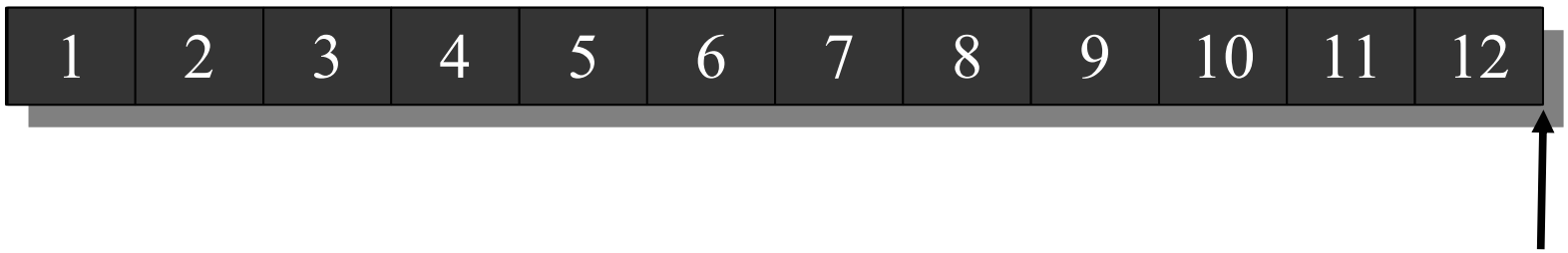


computation DAG



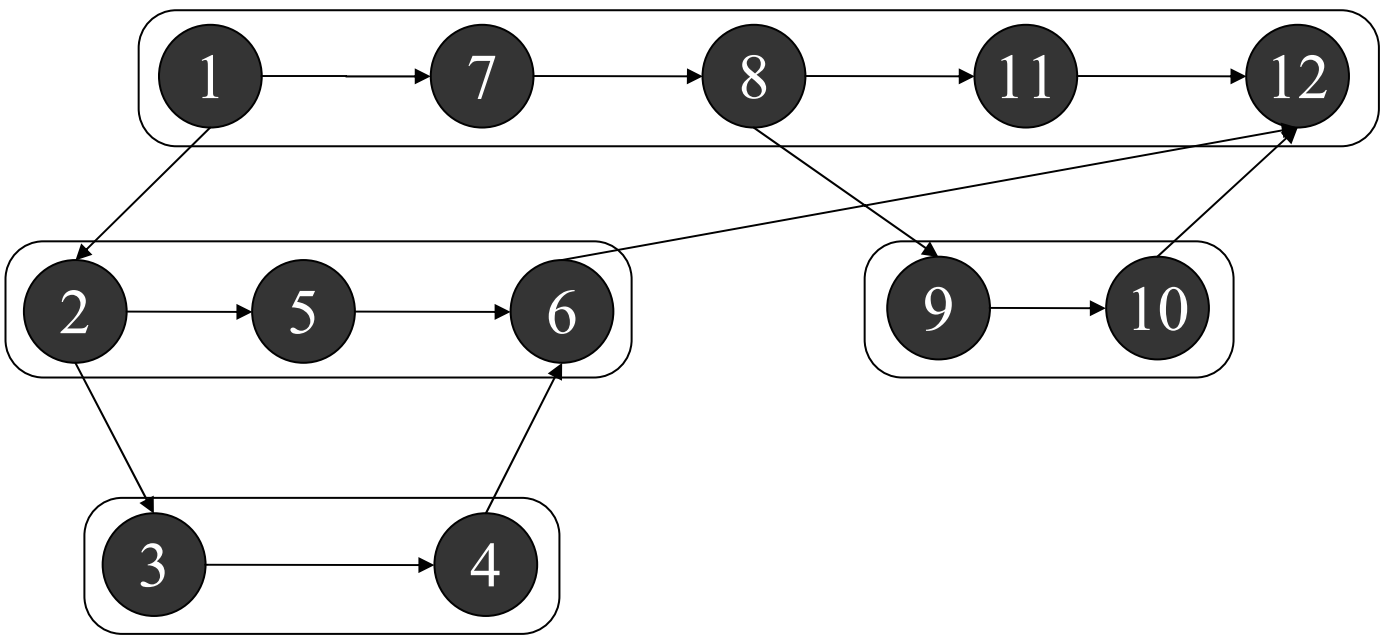
Single Processor Serial Append

FILE (serial append)



Why not in parallel?!

computation DAG



Fast Serial Append

Paralle**L** file **I/O** (**PLIO**) support
for Serial Append in
Cilk

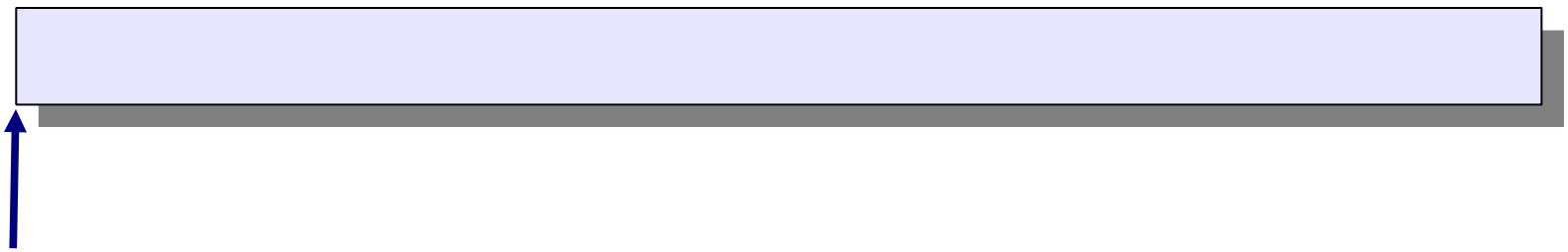
Alexandru Caracaş

Outline

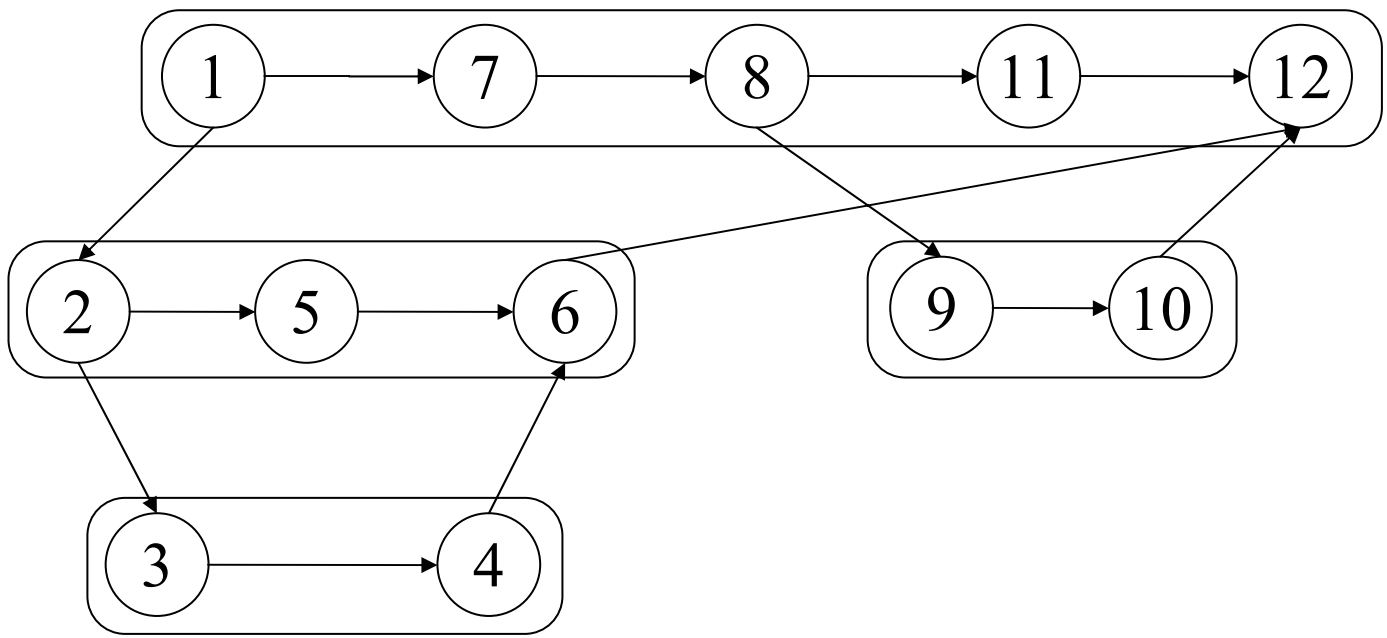
- Example
  - single processor & multiprocessor
- Semantics
  - view of *Cilk* Programmer
- Algorithm
  - modification of *Cilk* runtime system
- Implementation
  - Previous work
- Performance
  - Comparison

Multiprocessor Serial Append

FILE (serial append)

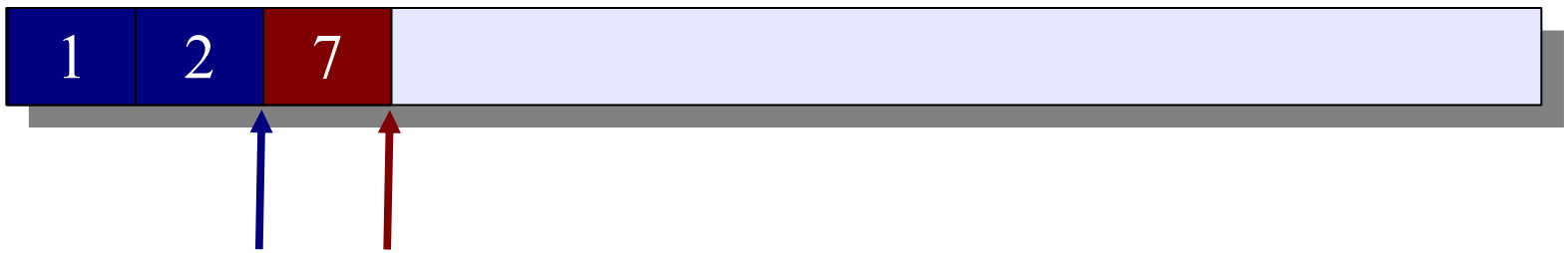


computation DAG

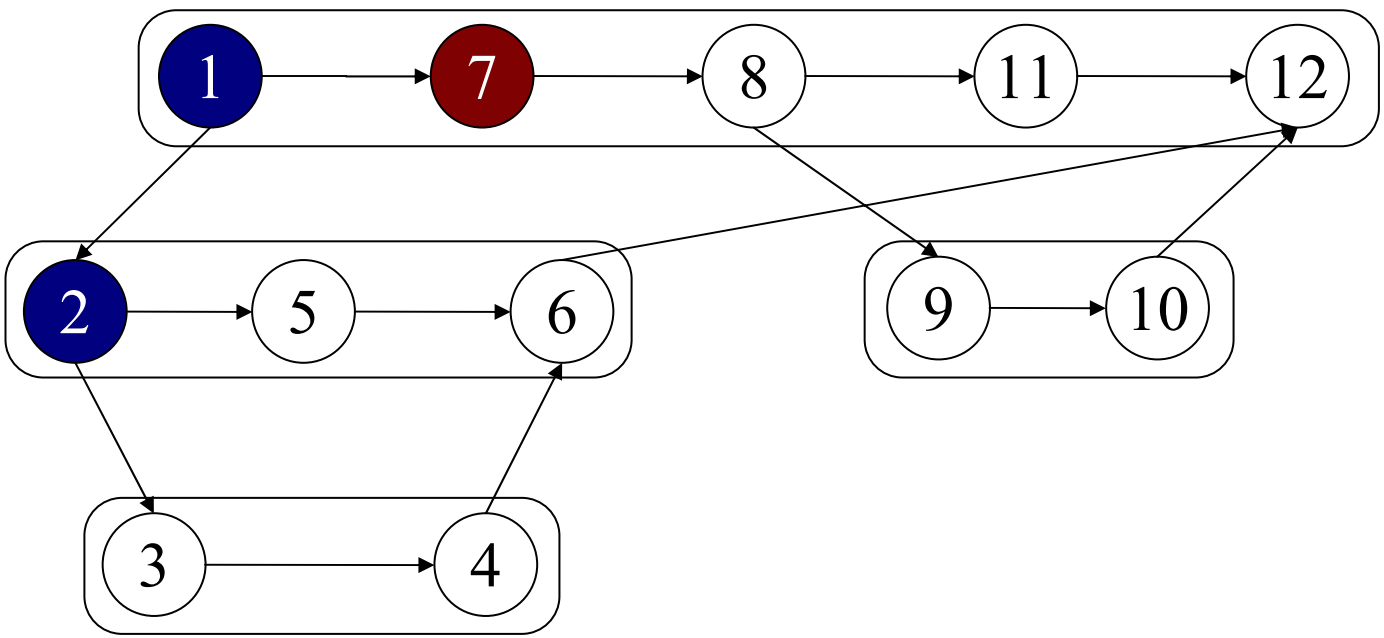


Multiprocessor Serial Append

FILE (serial append)

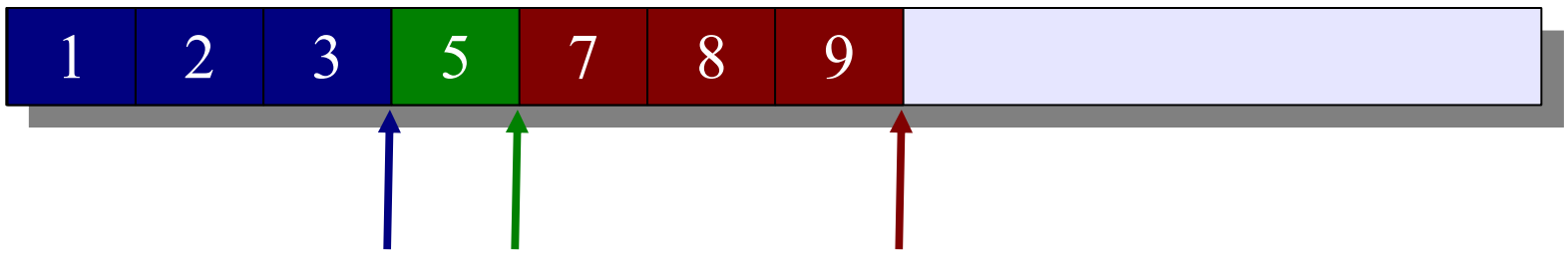


computation DAG

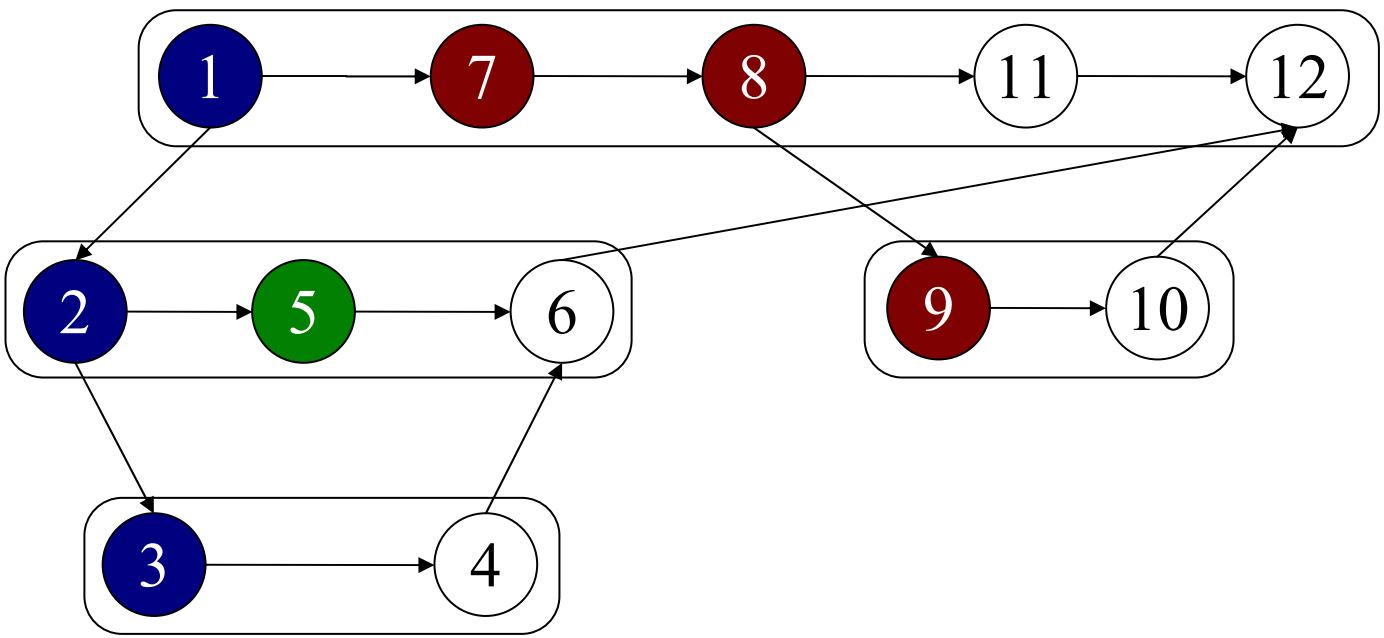


Multiprocessor Serial Append

FILE (serial append)

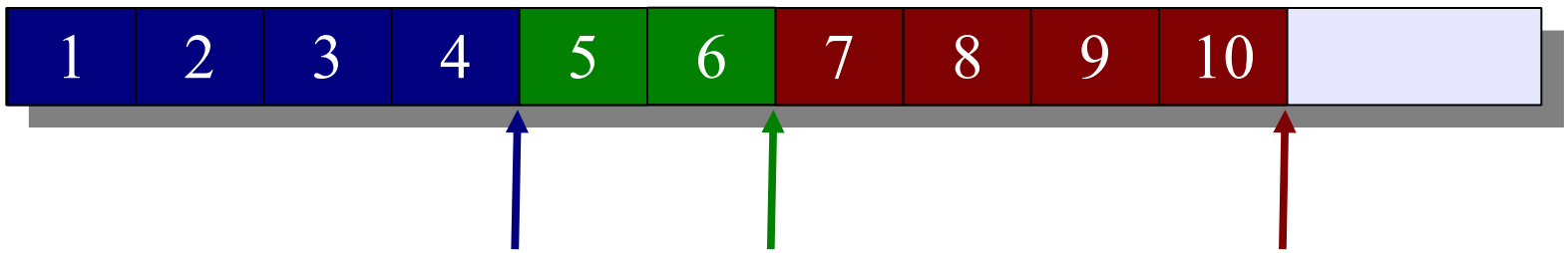


computation DAG

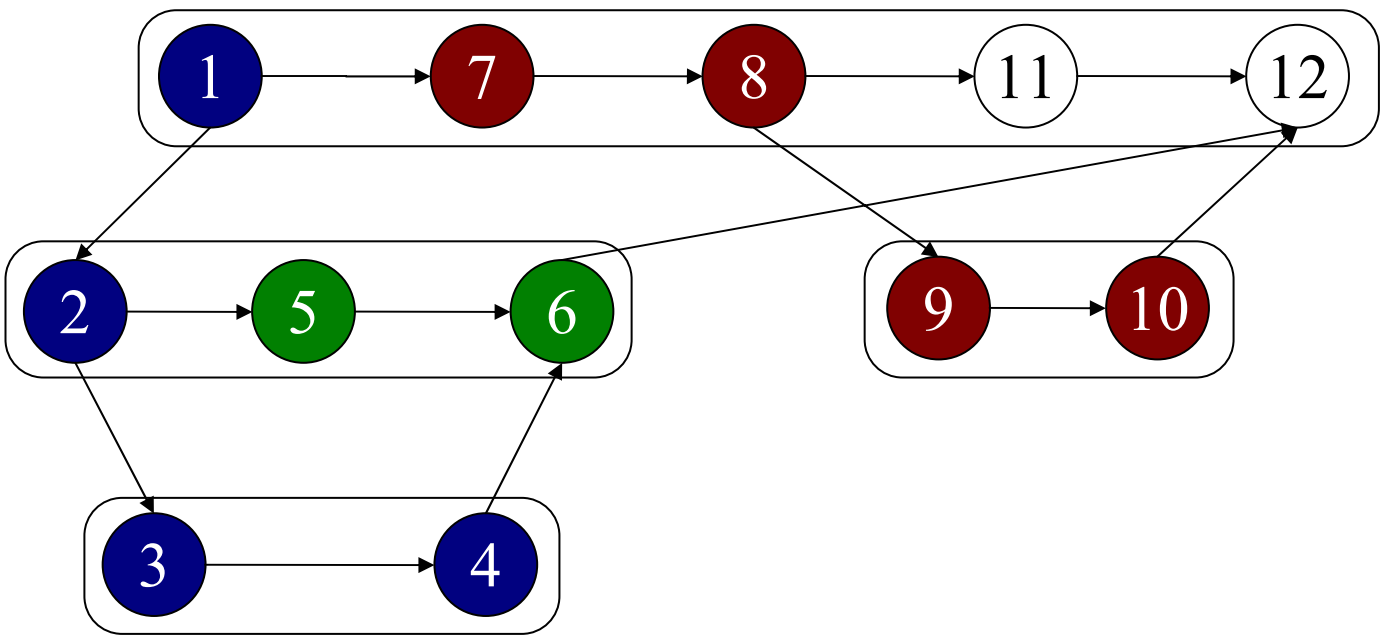


Multiprocessor Serial Append

FILE (serial append)

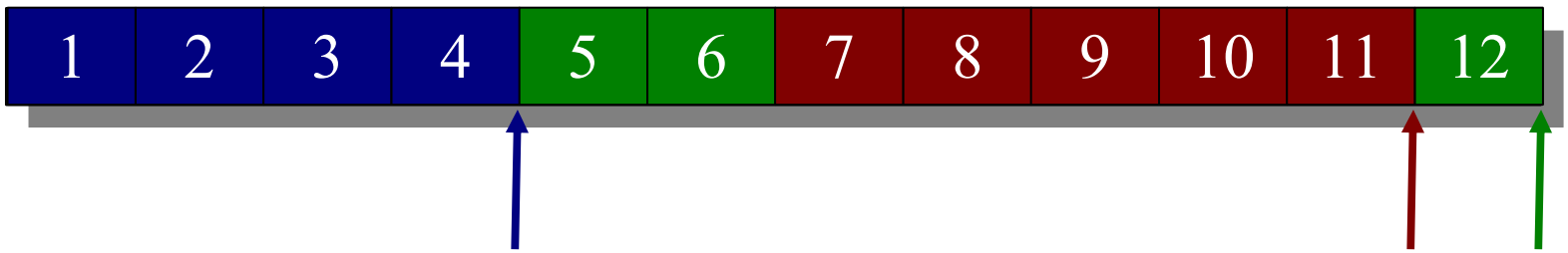


computation DAG

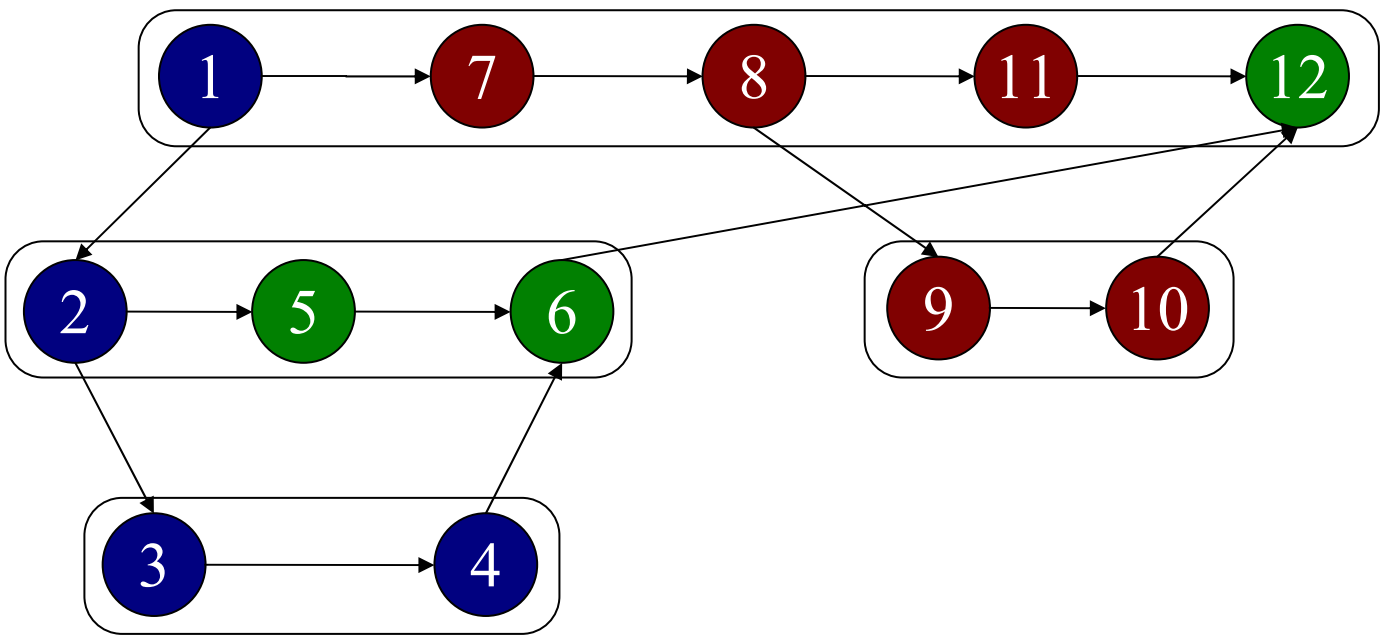


Multiprocessor Serial Append

FILE (serial append)



computation DAG



File Operations

- **open (FILE, mode) / close (FILE)**.
- **write (FILE, DATA, size)**
  - processor writes to its PION.
- **read (FILE, BUFFER, size)**
  - processor reads from PION.
    - Note: a seek operation may be required
- **seek (FILE, offset, whence)**
  - processor searches for the right PION in the ordered data structure

Semantics

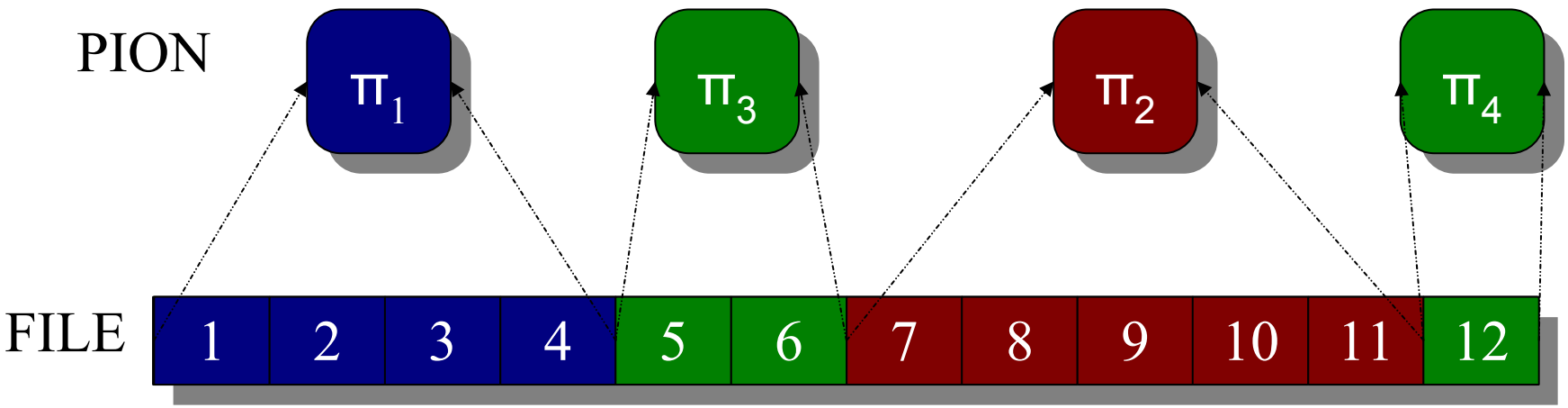
- View of *Cilk* programmer:
  - Write operations
    - preserve the sequential, serial append.
  - Read and Seek operations
    - can occur only after the file has been closed,
    - or on a newly opened file.

Approach (for *Cilk*)

- Bookkeeping (to reconstruct serial append)
  - Divide execution of the computation,
  - Meta-Data (PIONs) about the execution of the computation.
- Observation
  - In *Cilk,* **steals** need to be accounted for during execution.
- Theorem
  - expected # of steals = $O(PT_\infty)$.
- Corollary (see algorithm)
  - expected # of PIONs = $O(PT_\infty)$.

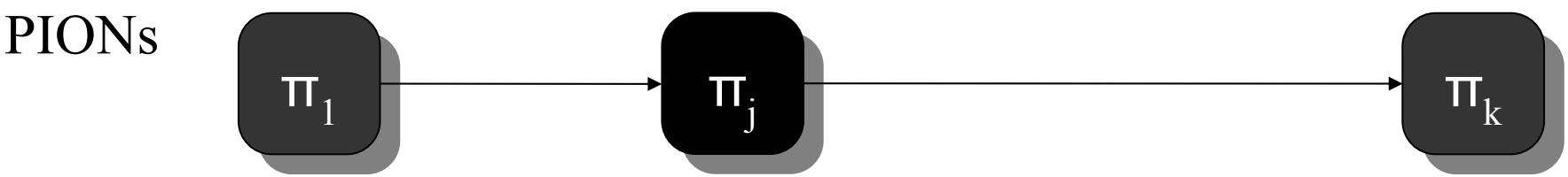
PION (**P**arallel **I/O** **N**ode)

- **Definition**: a PION represents all the write operations to a file performed by a processor in between 2 steals.
- A PION contains:
  - # data bytes written,
  - victim processor ID,
  - pointer to written data.



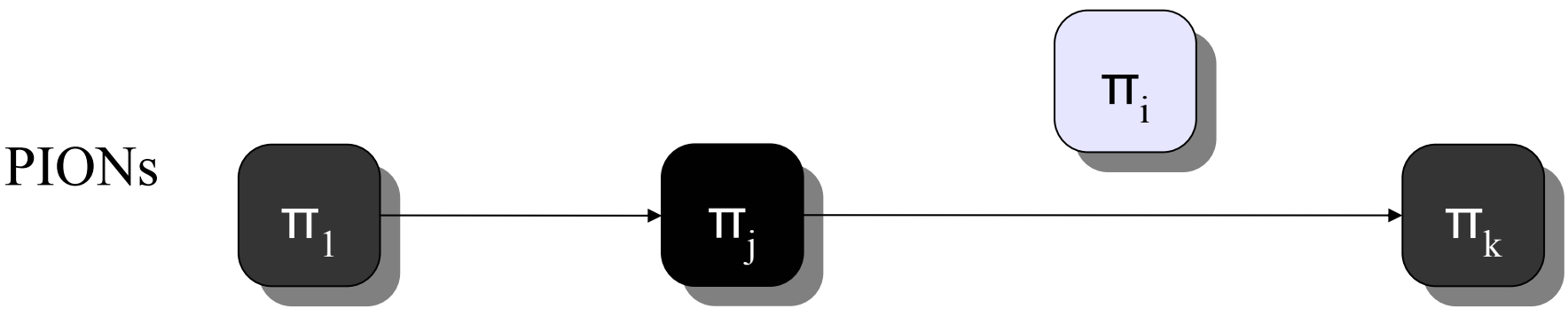
Algorithm

- All PIONSs are kept in an **ordered data structure**.
  - very simple Example: Linked List.
- On each steal operation performed by processor P_i from processor P_j:
  - create a new PION π_i,
  - attach π_i immediately after π_j, the PION of P_j in the order data structure.



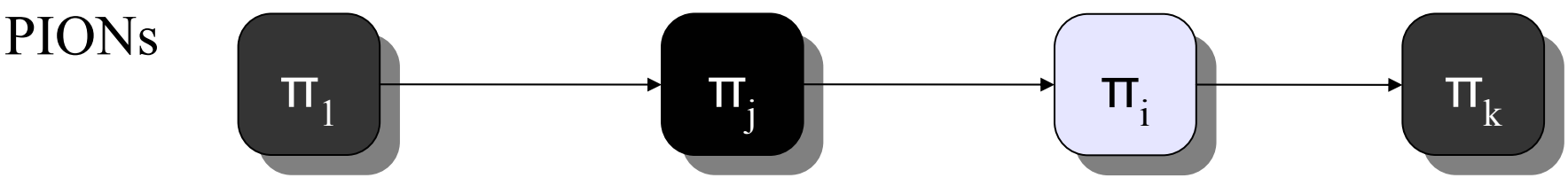
Algorithm

- All PIONSs are kept in an **ordered data structure**.
  - very simple Example: Linked List.
- On each steal operation performed by processor P_i from processor P_j:
  - create a new PION π_i,
  - attach π_i immediately after π_j, the PION of P_j in the order data structure.



Algorithm

- All PIONSs are kept in an **ordered data structure**.
  - very simple Example: Linked List.
- On each steal operation performed by processor P_i from processor P_j:
  - create a new PION π_i,
  - attach π_i immediately after π_j, the PION of P_j in the order data structure.



Implementation

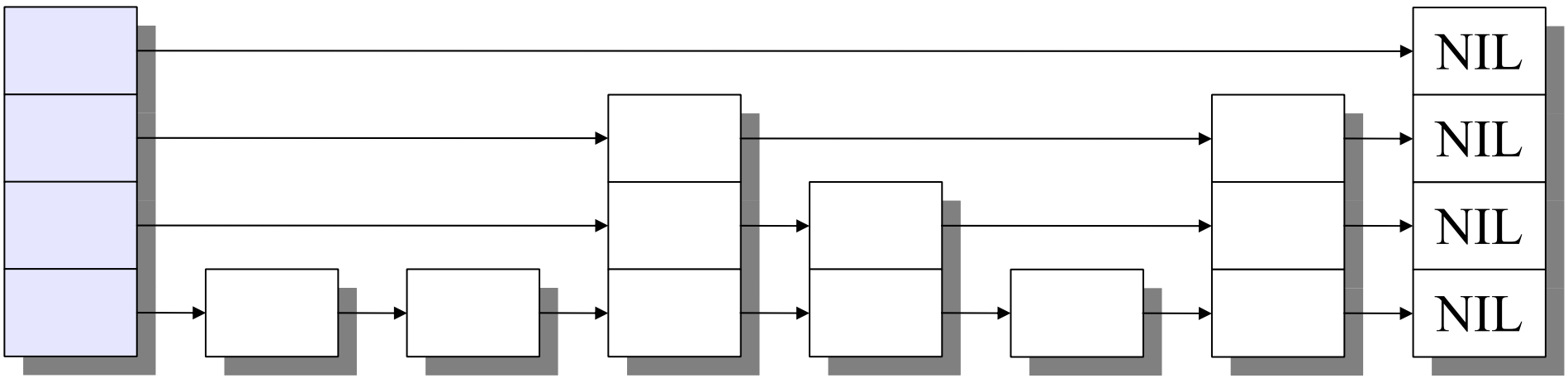
- Modified the *Cilk* **runtime system** to support desired operations.
  - implemented **hooks** on the steal operations.
- Initial implementation:
  - concurrent Linked List (easier algorithms).
- Final implementation:
  - concurrent double-linked Skip List.
- Ported *Cheerio* to *Cilk* 5.4.

Details of Implementation

- Each processor has a **buffer** for the data in its own PIONs
  - implemented as a file.
- Data structure to maintain the order of PIONs:
  - Linked List, **Skip List**.
- Meta-Data (order maintenance structure of PIONs)
  - kept in memory,
  - saved to a file when serial append file is closed.

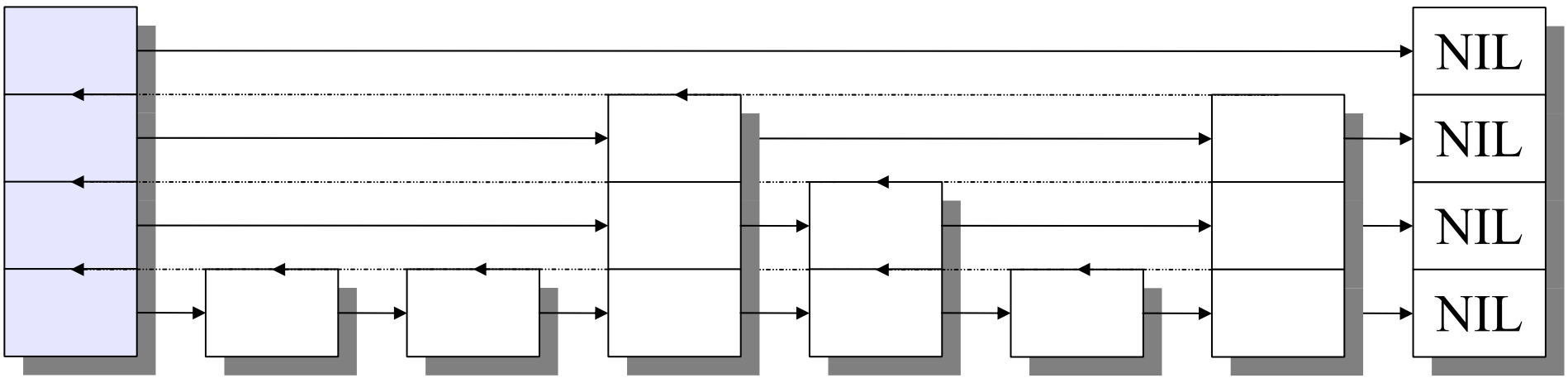
Skip List

- Similar performance with search trees:
  - O (log (SIZE)).



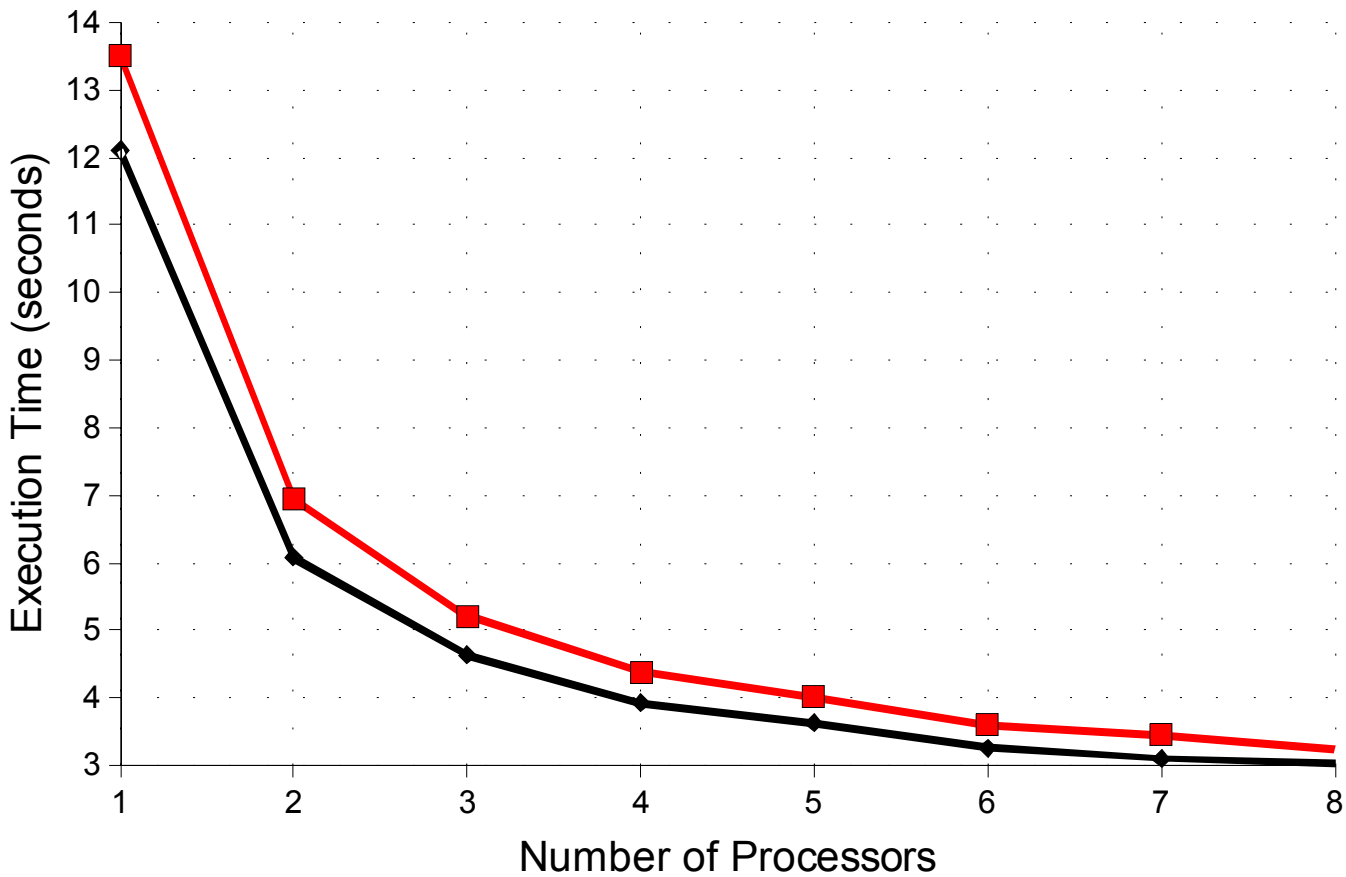
Double-Linked Skip List

- Based on Skip Lists (logarithmic performance).
- *Cilk* runtime-support in advanced implementation of PLIO as **rank order statistics**.



PLIO Performance

- no I/O *vs* writing 100MB with PLIO (w/ linked list),
- Tests were run on **yggdrasil** a 32 proc Origin machine.
- Parallelism=32,
- Legend:
  - black: no I/O,
  - red: PLIO.



Improvements & Conclusion

- Possible Improvements:
  - Optimization of algorithm:
    - delete PIONs with no data,
    - cache oblivious Skip List,
  - File system support,
  - Experiment with other order maintenance data structures:
    - B-Trees.
- Conclusion:
  - *Cilk* runtime-support for parallel I/O
    - allows serial applications dependent on sequential output to be parallelized.

References

- Robert D. Blumofe and Charles E. Leiserson. *Scheduling multithreaded computations by work stealing*. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pages 356-368, Santa Fe, New Mexico, November 1994.
- Matthew S. DeBergalis. *A parallel file I/O API for Cilk*. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2000.
- William Pugh. *Concurrent Maintenance of Skip Lists*. Departments of Computer Science, University of Maryland, CS-TR-2222.1, June, 1990.

References

- Thomas H. Cormen, Charles E. Leiserson, Donald L. Rivest and Clifford Stein. *Introduction to Algorithms (*2nd Edition). MIT Press. Cambridge, Massachusetts, 2001.
- Supercomputing Technology Group MIT Laboratory for Computer Science. *Cilk 5.3.2 Reference Manual*, November 2001. Available at http://supertech.lcs.mit.edu/cilk/manual-5.3.2.pdf.
- bz2 source code. Available at http://sources.redhat.com/bzip2.