Parallel execution of sequential code
Data dependencies restrict parallel (out-of-order) execution of sequential code:
- Slow 🙆♂️
- Fast ☺️
- But wrong ☹️

Dataflow execution
Schedule operations by data dependencies:
- Common at the instruction level in out-of-order processors
- Registers form dependencies; register renaming removes false (output and anti) dependencies
- Possible at the task level
- If side effects (pointers, globals etc.) are not allowed, input arguments and return values form dependencies; renaming is possible
- If side effects are allowed, all memory accesses form dependencies; renaming much harder

Inherent limitation: when dependencies are unknown, deterministic dataflow execution is impossible
- Speculations help – outside the scope of this work

Irregular data structures
- Pointer based
- Memory layout is dynamically modified
- Locations of target nodes unknown in advance

The problem:
Dependencies can’t be reasoned about, specified and handled!

The Solution
O-structure
A memory element providing:
Versioning – matching producers and consumers ensures RaW
Renaming – multiple versions per location eliminate WaW and WaR
Per-version lock – facilitate handling unknown dependencies

Use:
- Encode dependencies in memory
- When a dependency is known, ask for a specific version
- When a dependency is unknown, ask for the latest version
- Using lock abilities, it will be the version you need!

Parallelizing irregular data structures
Prerequisite – Store shared data (e.g., pointers) in O-structures

Step 1 – Use versioning of head pointer to ensure initial ordering

Step 2 – Use locking to order possibly dependent operations

Step 3 – Use renaming to eliminate false dependencies

Actual order of execution doesn’t matter now!

Readers need not lock – thanks to renaming, they see the right state even if a writer bypasses them
Renaming not only increases parallelism, but also removes the need to track readers!

The technique can be used for any Unipath data structure operation:
1. Single entry point (head/root)
2. Only one path to each node (So no going back!)