Persistent Data Structures (Version Control)

- **Ephemeral**
  - Partial persistence
    - Version list
      - Update: V0, V1, V2, V3, V4, V5, V6
      - Query: V0, V1, V2, V3, V4, V5, V6

- **Partial persistence**
  - Version tree
    - Version: V0, V1, V2, V3, V4, V5
    - Updates at leaves: V1, V2, V3, V4, V5
    - Query: V1, V2, V3, V4, V5

- **Full persistence**
  - Version DAG
    - Version: V0, V1, V2, V3, V4
    - Updates at leaves: V2, V3, V4
    - Query: V2, V3, V4

- **Confluently persistence**
  - Version DAG
    - Version: V0, V1, V2, V3
    - Updates at leaves: V3
    - Query: V3

- **Purely functional**
  - NEVER modify
  - ONLY create new pairs
  - ONLY DAGs

- **Retroactive**
  - Update & query all versions
  - Updates in the past propagate
Planar Point Location

Partial persistent search trees

$O(n \cdot \log n)$ preprocessing, $O(\log n)$ query
Path copying (trees)
Partial persistence

- Version ID = time = 0, 1, 2, ...

- Fast node (any data structure)
  - record all updates in node (version, value) pairs
  - field updates O(1)
  - field queries ≡ predecessor wrt version id (search tree/vEB)

- Node copying (O(1) degree data structures)
  - Persistent node = collection of nodes, each valid for an interval of versions, with Δ extra updates, Δ = max indegree
  - pointers must have subinterval of the node pointing to; otherwise copy and insert pointers (cascading copying)
  - NB: Needs to keep track of back-pointers

\[
\begin{array}{|c|c|c|}
\hline
[0,8[ & [8,13[ & [13,\infty[ \\
field_1: & (0,x) (3,y) & (8,z) (10,w) \\
field_2: & (0,a) (7,c) & (8,c) (9,d) \\
\hline
\end{array}
\]
Full persistence

- Fat node
  - Updates (1,x) (6,y) (7,z) to a field
  - Queries = binary search among versions
  - Update (7,z): Insert 7 as leftmost child of 4; insert pairs for 7 and 5=succ(7)

- Node splitting ($\geq 2\Delta$ ekstra fields)
Persistence techniques


- Partial persistence, trees, O(1) access, amortized O(1) update


- Partial & full persistence, O(1) degree data structures, O(1) access, amortized O(1) update


- Partial persistence, O(1) degree data structures, O(1) access & updates update


- Full persistence, RAM structures, O(loglog n) access, O(loglog n) amortized expected updates
Comparison of persistence techniques

- Copy data structure for each version
  - no query overhead, slow updates & wastes a lot of space

- Record updates & keep current version
  - fast updates & queries to current version, space efficient, slow queries in the past

- Path copying
  - applies to trees, no query overhead, space overhead = depth of update

- Fat node
  - partial persistence: $O(1)$ updates and space optimal, $\log \log n$ query overhead
  - full persistence: $O(\log \log n)$ expected amortized updates and space optimal, $\log \log n$ query overhead

- Node copying/splitting
  - fast updates & queries (amortized updates for full persistence)
  - only works for pointer-based structures with $O(1)$ degree