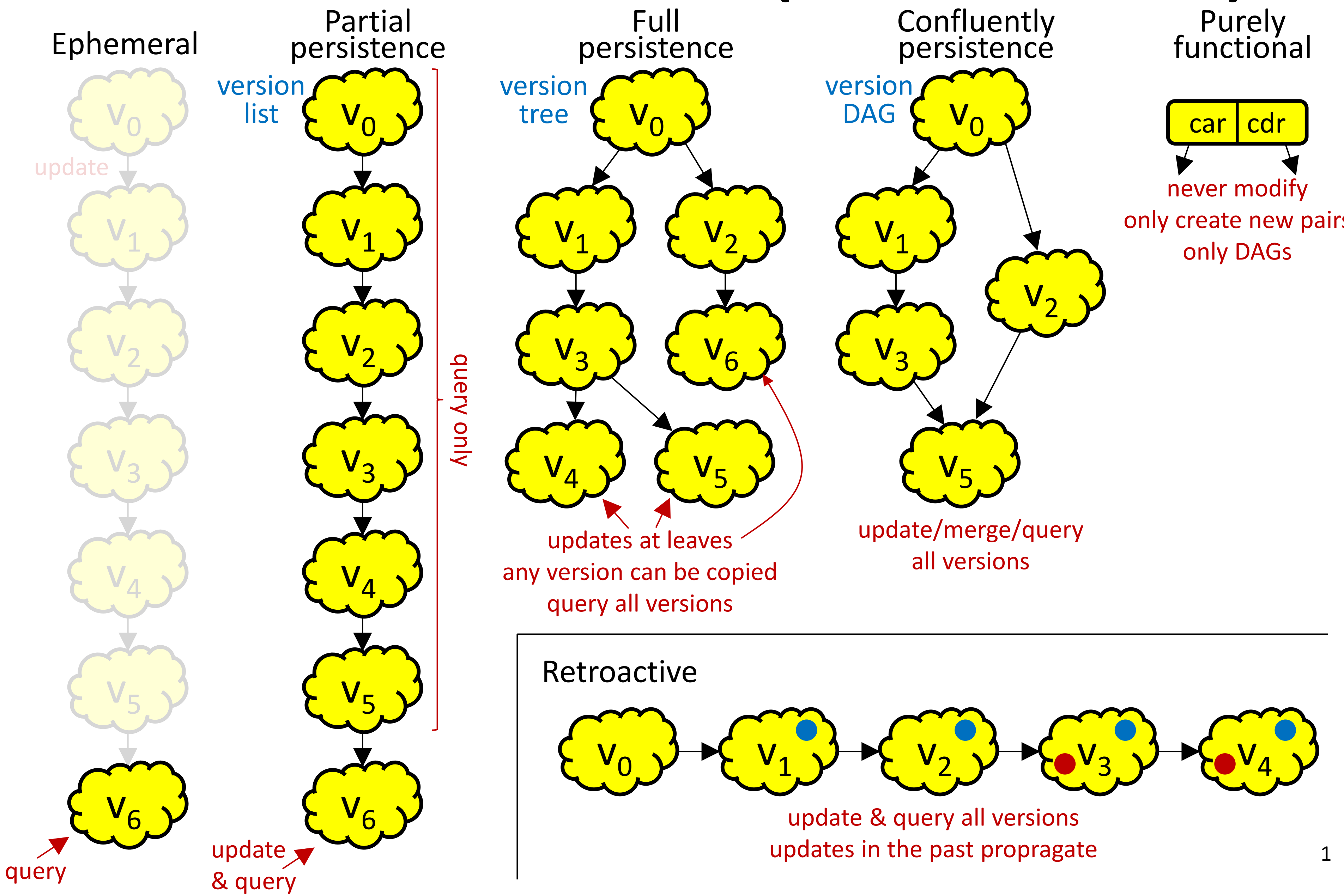
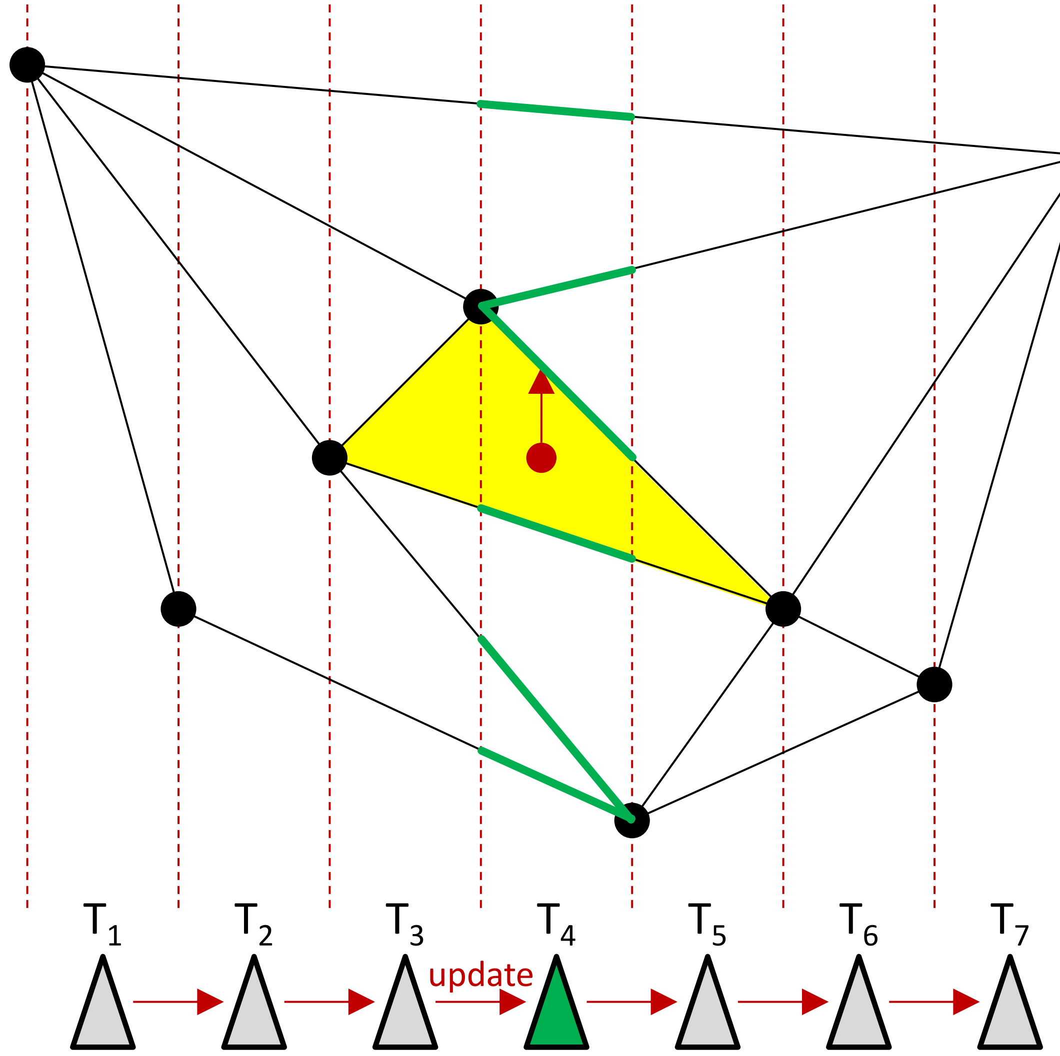
Persistent Data Structures (Version Control)



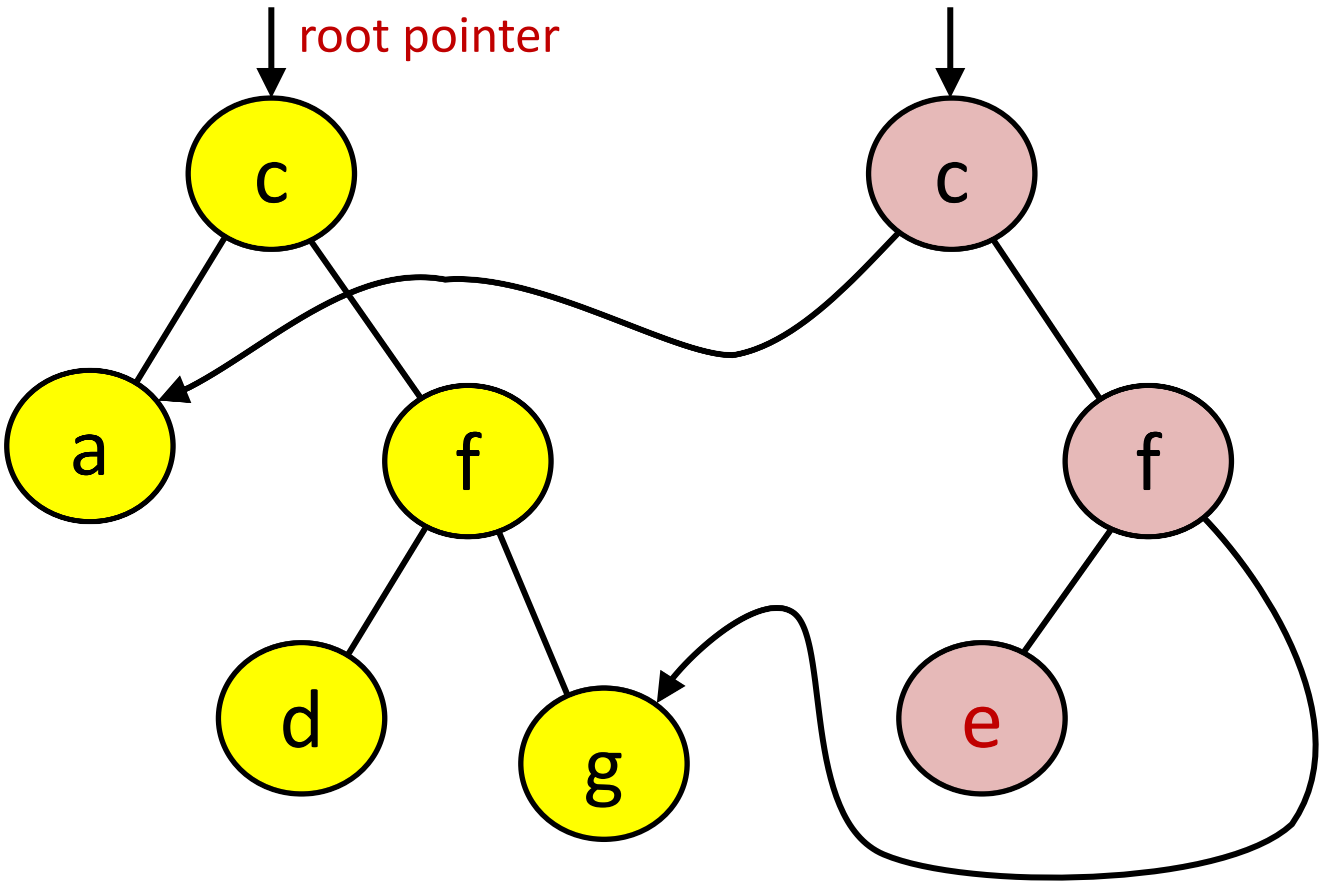
Planar Point Location



Partial persistent search trees

O(n·log n) preprocessing, O(log n) query

Path Copying (trees)



Partial Persistence

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

- Fat 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)

$field_1$:	**(0,x) (3,y) (7,z)**
$field_2$:	**(0,a) (14,c) (16,b)**

- 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 (cacading copying) NB: Needs to keep track of back-pointers

[0,8[

$field_1$:	**(0,x) (3,y)**
$field_2$:	**(0,a) (7,c)**

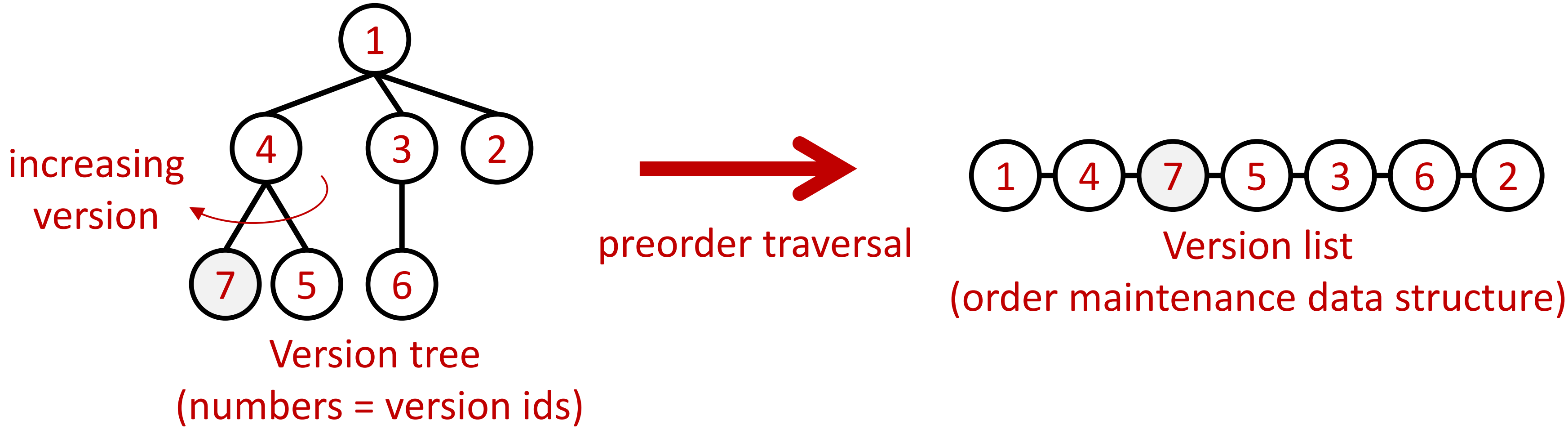
[8,13[

$field_1$:	**(8,z) (10,w)**
$field_2$:	**(8,c) (9,d)**

[13,∞[

$field_1$:	**(13,w) (q5,y)**
$field_2$:	**(13,e) (14,c)**

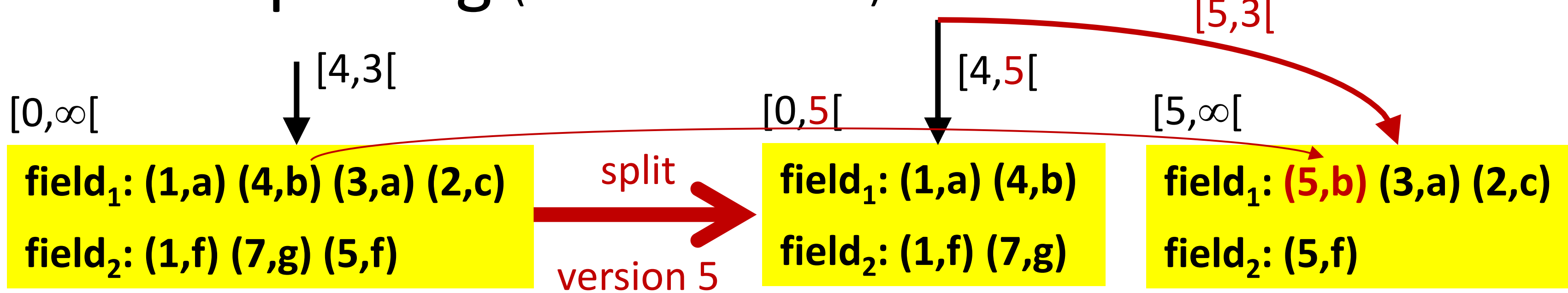
Full Persistence



- **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)

field: (1,x) (7,z) (5,x) (6,y) (2,x)

- **Node splitting** (≥ 2Δ ekstra fields)



Persistence Techniques

[N. Sarnak, R.E. Tarjan, *Planar point location using persistent search trees*, Communications of the ACM, 29(7), 669-679, 1986]

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

[J.R. Driscoll, N. Sarnak, D.D. Sleator, R.E. Tarjan, *Making Data Structures Persistent*, Journal of Computer and System Sciences, 38(1), 86-124, 1989]

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

[P.F. Dietz, R. Raman, *Persistence, Amortization and Randomization*. Proceedings 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, 78-88, 1991]
[G.S. Brodal, *Partially Persistent Data Structures of Bounded Degree with Constant Update Time*, Nordic Journal of Computing, volume 3(3), pages 238-255, 1996]

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

[P.F. Dietz, *Fully Persistent Arrays*. Proceedings 1st Workshop on Algorithms and Data Structures, LNCS 382, 67-74, 1989]

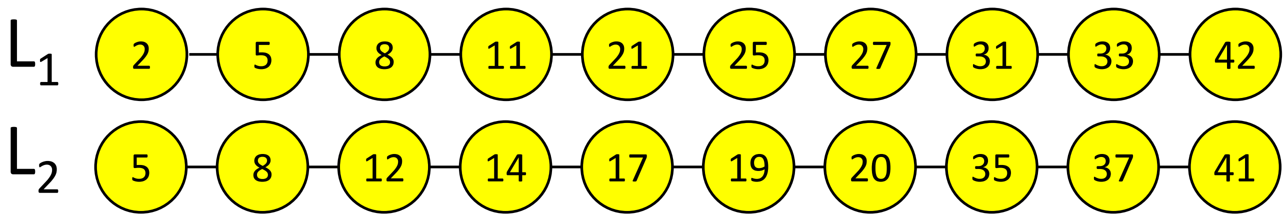
- Full persistence, RAM structures, O(loglog n) access, O(loglog n) amortized expected updates

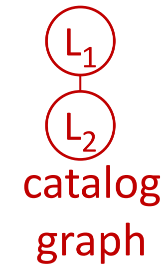
Comparison of Persistence Techniques

- 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, loglog n query overhead
  - full persistence: O(loglog n) expected amortized updates and space optimal, loglog n query overhead
- Node copying/splitting
  - fast updates & queries (amortized updates for full persistence)
  - only works for pointer-based structures with O(1) degree

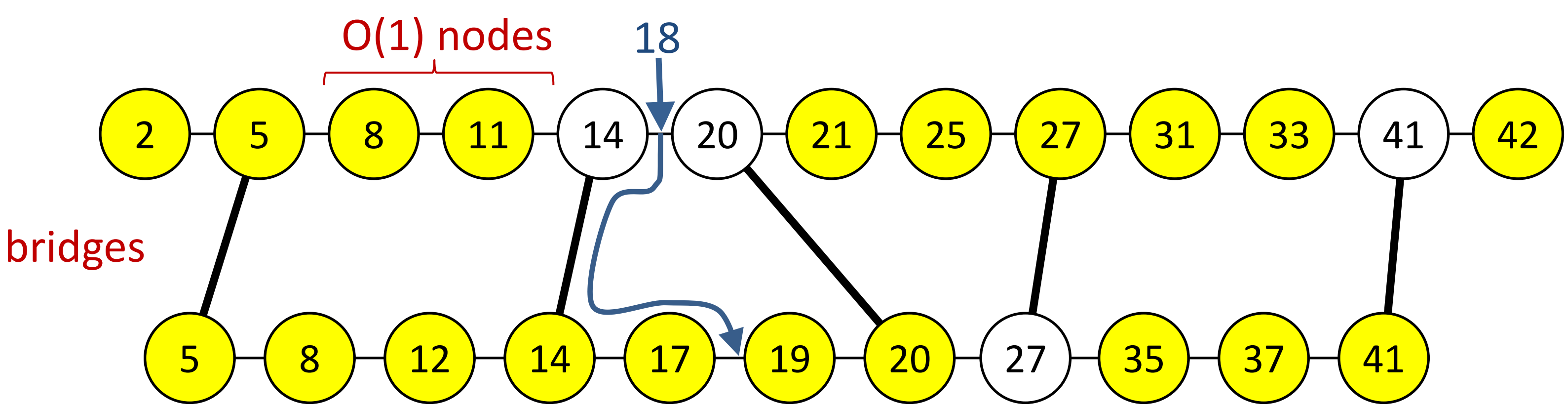
Fractional Cascading

- Basic Idea : 2 x BinSearch $\Rightarrow$ 1 BinSearch + O(1)





- Build bridges (and pointers to nearest original element)
- Searches to next list : Traverse nearest bridge
- Construction : Repeatedly create bridges until all gaps O(1)



- Generalizes to catalog graphs of degree O(1)

Fractional Cascading Techniques

[Bernard Chazelle, Leonidas J. Guibas, *Fractional Cascading: I. A Data Structuring Technique*, Algorithmica, 1(2): 133-162, 1986.]

[Bernard Chazelle, Leonidas J. Guibas: *Fractional Cascading: II. Applications*. Algorithmica 1(2): 163-191 (1986)]

- Static fractional cascading, O(1) worst-case access

[Kurt Mehlhorn, Stefan Näher: *Dynamic Fractional Cascading*. Algorithmica 5(2): 215-241 (1990)]

- Dynamic fractional cascading, O(loglog *N*) worst-case access, amortized insert and delete
- Insertion or deletion only, O(1) per worst-case access, amortized insert or delete