Last week

- We have discussed the different methods we can query spatial data based on their attributes, topology and geometry directly in SQL;
- We have discussed how the integration between database and client (pure SQL/GIS) handles the queries;
- A quick look at the computational complexity of these queries has been taken

Geo875 | FS25
University of Zürich

5. Lecture Spatial Databases

Spatial Indexing

Rolf Meile

Eidg. Forschungsanstalt für Wald, Schnee und Landschaft (WSL)

Swiss Federal Institute for Forest, Snow and Landscape Research

Esra Suel

Dept. of Geography, University of Zürich

Learning Objectives

- Understand the principles of indexing
- Understand the filter-refine paradigm for efficient execution of spatial queries
- Be able to apply the principles of data indexing on spatial data
- Understand the differences between space driven and data driven spatial indices; know their characteristics and pros/cons;

Overview

1. **Filter-Refine**
2. What Are Indices? What Are They Good for?
3. Trees – a Bit of Terminology
4. Spatial Indexing
5. Space-Driven Indices
6. Object-Driven Indices

Spatial queries – finding matching candidates

- A spatial query has to identify all matching records in a table based on a spatial (topological, geometrical) operator, applied to the geometries of these records:

```
SELECT s.id, s.geom
FROM   tmp_river_segment s  -- more than 170'000 recs
WHERE  ST_Intersects(s.geom,
ST_GeomFromText('LINESTRING(2683846 1250116, 2684111 1250222, 2685846
1251812)',2056)
);
```

- This means, that the query single geometry linestring must be compared against all line geometries of tmp_river_segement

Spatial queries – finding matching candidates

Graphically:
single line intersecting
many riversegments

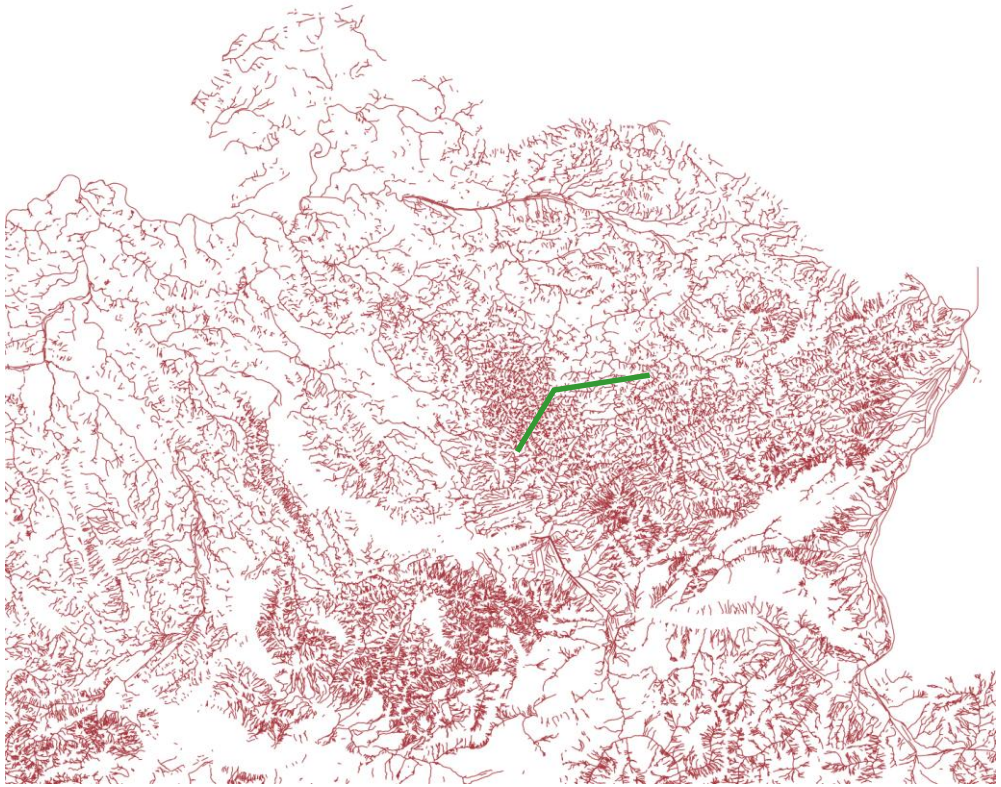


Table:
on the fly single line geometry
versus tmp_river_segment

- Math happens on these vertices
- Help needed to speed it up

SELECT r.id, r.geom FROM project25.tmp_river r | Enter a SQL expression to filter resu

Grid | Text | Spatial | ecord

	id	geom
1	1	MULTILINESTRING ((2693929.1752999984 1288494.6545000002,
2	2	MULTILINESTRING ((2614591.9565000013 1267608.2939999998, 2
3	3	MULTILINESTRING ((2666416.5747999996 1269007.482900001, 26
4	4	MULTILINESTRING ((2719039.058600001 1274689.2391999997, 27
5	5	MULTILINESTRING ((2712465.7632 1266640.3187999986, 2712462
6	6	MULTILINESTRING ((2632301.2848999985 1261832.1664999984, 2
7	7	MULTILINESTRING ((2651515.242899999 1265442.3951999992, 26
8	8	MULTILINESTRING ((2688742.8528000005 1258793.1535999998,
9	9	MULTILINESTRING ((2695030.3845000006 1265482.794399999, 2
10	10	MULTILINESTRING ((2694234.3308999985 1257787.7252000012,
11	11	MULTILINESTRING ((2582671.7892000005 1246649.064100001, 25
12	73	MULTILINESTRING ((2770042.7041999996 1203402.9952999987,
13	12	MULTILINESTRING ((2597280.3933000006 1248901.045400001, 25
14	13	MULTILINESTRING ((2614533.596500002 1248039.7747000009, 2
15	14	MULTILINESTRING ((2610864.860800002 1250821.616799999, 261
16	15	MULTILINESTRING ((2643798.014899999 1250332.2730999999, 2
17	16	MULTILINESTRING ((2646827.8704000004 1253481.5780000016, 2
18	17	MULTILINESTRING ((2656588.9494000003 1251236.6906999983,
19	18	MULTILINESTRING ((2677252.122299999 1242029.4565000013, 26
20	19	MULTILINESTRING ((2698099.028999999 1253816.7171999998, 26
21	20	MULTILINESTRING ((2702517.2465000004 1253278.3797999993,
22	21	MULTILINESTRING ((2713644.9684000015 1245819.7109000012, 27
23	22	MULTILINESTRING ((2726297.7391999997 1244965.2996999994

Spatial queries – spatial join finding matching candidates

- Spatial join by definition compares all against all; as a result, all spatial object interactions of the queried topology are given;
- Cartesian product, but restricted to topology

```
SELECT b.biogeoid, b.biogeoname, b.geom geom_bio,
       s.id, s.geom geom_rivseg
FROM   biogeo b
       -- 5 recs (multipolygon, many vertices!)
JOIN   tmp_river_segement s
       -- 170000 recs (multiline, some with many vertices as well)
ON     ST_Intersects(b.geom,s.geom);
```

- Query geometries biogeo must be compared against all geometries of tmp_river_segment; roughly 850'000 **comparisons**;
- Why do we have to wait for the results? What makes this query expensive? CPU usage?

Spatial queries – filter refine paradigm

- Very slow for larger tables, and for *MxN* comparisons
- Number of vertices of participating geometries decreases answer time proportionally
- Retrieval from harddisk (large full tables) is much slower than from memory; we cannot load everything to memory;

What we need

- Simpler and faster **comparing algorithms** and **simplified spatial objects** (to filter out some of our query objects)
- A **structure** that supports this with less need for memory space than original tables; disk space is ok;
- Having this in place we can reduce the number of objects that have to be compared with full complex algorithms (point in polygon; line intersection;...; refining)

Spatial queries – filter refine paradigm cont.

Filter

- Coarse, approximate search
- Simpler geometries
- Hence simpler comparing algorithms
- Clever structure used
- Mainly in memory - only partly from disk

Refine

- Detailed matching and checking for the true relationship – also in memory if possible
- This includes checking all vertices of two geometries against each other

- Similar strategies are applied also in non-spatial contexts, but with **simple dataypes** which can be ordered – see coming up slides

Overview

1. Filter-Refine
2. **What Are Indices? What Are They Good for?**
3. Trees – a Bit of Terminology
4. Spatial Indexing
5. Space-Driven Indices
6. Object-Driven Indices

Exercise 1 searching for rank

Find the city with the largest population in 2005 (unordered table, hence not indexed)

Cid	City	Country	Latitude	Longitude	pop1950	pop2005	pop2010	pop2015	pop2020	pop2025	pop2050
1	Sofia	Bulgaria	42.7	23.33	520	1170	1180	1210	1230	1240	1236
2	Mandalay	Myanmar	21.97	96.08	170	920	960	1030	1170	1310	1446
3	Nay Pyi Taw	Myanmar	19.75	96.1	0	60	930	1020	1180	1320	1461
4	Yangon	Myanmar	16.87	96.12	1300	3930	4090	4350	4840	5360	5869
5	Minsk	Belarus	53.89	27.57	280	1780	1800	1850	1880	1880	1883
6	Phnum PÈnh	Cambodia	11.56	104.91	360	1360	1470	1650	2030	2460	2911
7	El DjazaÔr	Algeria	36.78	3.05	520	3200	3350	3570	3920	4240	4499
8	Wahran	Algeria	35.74	-0.52	270	760	800	850	940	1030	1105
9	Douala	Cameroon	4.13	9.7	100	1770	1910	2110	2420	2720	2996
10	YaoundÈ	Cameroon	3.86	11.51	30	1490	1610	1790	2060	2310	2549
11	Calgary	Canada	51.03	-114.04	130	1060	1110	1180	1260	1300	1345
12	Edmonton	Canada	53.55	-113.5	160	1020	1060	1110	1170	1220	1256
13	MontrÈal	Canada	45.54	-73.65	1340	3600	3680	3780	3910	4010	4108
14	Ottawa-Gatineau	Canada	45.37	-75.65	280	1120	1140	1180	1230	1270	1315
15	Toronto	Canada	43.72	-79.41	1070	5040	5210	5450	5690	5830	5946
16	Vancouver	Canada	49.27	-122.96	560	2090	2150	2220	2310	2380	2444
17	N'DjamÈna	Chad	12.1	15.24	20	900	990	1130	1400	1750	2172
18	Santiago	Chile	-33.43	-70.65	1320	5600	5720	5880	6080	6220	6310
19	Valparaìso	Chile	-33.02	-71.55	330	840	850	880	920	960	982
20	Anshan, Liaoning	China	41.11	122.97	480	1610	1640	1700	1860	2030	2167
21	Anshun	China	26.25	105.93	190	820	850	900	990	1080	1164
22	Anyang	China	36.1	114.33	110	850	890	950	1060	1160	1240
23	Baoding	China	38.85	115.48	180	1040	1110	1210	1360	1480	1586
24	Baotou	China	40.65	109.8	530	1920	2040	2210	2470	2690	2869
25	Beijing	China	39.9	116.38	4330	10720	11110	11740	12840	13810	14545
26	Bengbu	China	32.93	117.35	230	870	890	940	1040	1140	1225

Exercise 2a searching for rank

Find the city with the seventh largest population in 2005 (unordered table, hence not indexed)

Cid	City	Country	Latitude	Longitude	pop1950	pop2005	pop2010	pop2015	pop2020	pop2025	pop2050
1	Sofia	Bulgaria	42.7	23.33	520	1170	1180	1210	1230	1240	1236
2	Mandalay	Myanmar	21.97	96.08	170	920	960	1030	1170	1310	1446
3	Nay Pyi Taw	Myanmar	19.75	96.1	0	60	930	1020	1180	1320	1461
4	Yangon	Myanmar	16.87	96.12	1300	3930	4090	4350	4840	5360	5869
5	Minsk	Belarus	53.89	27.57	280	1780	1800	1850	1880	1880	1883
6	Phnum PÈnh	Cambodia	11.56	104.91	360	1360	1470	1650	2030	2460	2911
7	El DjazaÔr	Algeria	36.78	3.05	520	3200	3350	3570	3920	4240	4499
8	Wahran	Algeria	35.74	-0.52	270	760	800	850	940	1030	1105
9	Douala	Cameroon	4.13	9.7	100	1770	1910	2110	2420	2720	2996
10	YaoundÈ	Cameroon	3.86	11.51	30	1490	1610	1790	2060	2310	2549
11	Calgary	Canada	51.03	-114.04	130	1060	1110	1180	1260	1300	1345
12	Edmonton	Canada	53.55	-113.5	160	1020	1060	1110	1170	1220	1256
13	MontrÈal	Canada	45.54	-73.65	1340	3600	3680	3780	3910	4010	4108
14	Ottawa-Gatineau	Canada	45.37	-75.65	280	1120	1140	1180	1230	1270	1315
15	Toronto	Canada	43.72	-79.41	1070	5040	5210	5450	5690	5830	5946
16	Vancouver	Canada	49.27	-122.96	560	2090	2150	2220	2310	2380	2444
17	N'DjamÉna	Chad	12.1	15.24	20	900	990	1130	1400	1750	2172
18	Santiago	Chile	-33.43	-70.65	1320	5600	5720	5880	6080	6220	6310
19	Valparaìso	Chile	-33.02	-71.55	330	840	850	880	920	960	982
20	Anshan, Liaoning	China	41.11	122.97	480	1610	1640	1700	1860	2030	2167
21	Anshun	China	26.25	105.93	190	820	850	900	990	1080	1164
22	Anyang	China	36.1	114.33	110	850	890	950	1060	1160	1240
23	Baoding	China	38.85	115.48	180	1040	1110	1210	1360	1480	1586
24	Baotou	China	40.65	109.8	530	1920	2040	2210	2470	2690	2869
25	Beijing	China	39.9	116.38	4330	10720	11110	11740	12840	13810	14545
26	Bengbu	China	32.93	117.35	230	870	890	940	1040	1140	1225

Exercise 2b searching for rank

Find the city with the seventh largest population in 2005 (ordered = in place index)

Cid	City	Country	Latitude	Longitude	pop1950	pop2005	pop2010	pop2015	pop2020	pop2025	pop2050
25	Beijing	China	39.9	116.38	4330	10720	11110	11740	12840	13810	14545
18	Santiago	Chile	-33.43	-70.65	1320	5600	5720	5880	6080	6220	6310
15	Toronto	Canada	43.72	-79.41	1070	5040	5210	5450	5690	5830	5946
4	Yangon	Myanmar	16.87	96.12	1300	3930	4090	4350	4840	5360	5869
13	MontrÈal	Canada	45.54	-73.65	1340	3600	3680	3780	3910	4010	4108
7	El DjazaÔr	Algeria	36.78	3.05	520	3200	3350	3570	3920	4240	4499
16	Vancouver	Canada	49.27	-122.96	560	2090	2150	2220	2310	2380	2444
24	Baotou	China	40.65	109.8	530	1920	2040	2210	2470	2690	2869
5	Minsk	Belarus	53.89	27.57	280	1780	1800	1850	1880	1880	1883
9	Douala	Cameroon	4.13	9.7	100	1770	1910	2110	2420	2720	2996
20	Anshan, Liaoning	China	41.11	122.97	480	1610	1640	1700	1860	2030	2167
10	YaoundÈ	Cameroon	3.86	11.51	30	1490	1610	1790	2060	2310	2549
6	Phnum PÈnh	Cambodia	11.56	104.91	360	1360	1470	1650	2030	2460	2911
1	Sofia	Bulgaria	42.7	23.33	520	1170	1180	1210	1230	1240	1236
14	Ottawa-Gatineau	Canada	45.37	-75.65	280	1120	1140	1180	1230	1270	1315
11	Calgary	Canada	51.03	-114.04	130	1060	1110	1180	1260	1300	1345
23	Baoding	China	38.85	115.48	180	1040	1110	1210	1360	1480	1586
12	Edmonton	Canada	53.55	-113.5	160	1020	1060	1110	1170	1220	1256
2	Mandalay	Myanmar	21.97	96.08	170	920	960	1030	1170	1310	1446
17	N'DjamÈna	Chad	12.1	15.24	20	900	990	1130	1400	1750	2172
26	Bengbu	China	32.93	117.35	230	870	890	940	1040	1140	1225
22	Anyang	China	36.1	114.33	110	850	890	950	1060	1160	1240
19	Valparaìso	Chile	-33.02	-71.55	330	840	850	880	920	960	982
21	Anshun	China	26.25	105.93	190	820	850	900	990	1080	1164
8	Wahran	Algeria	35.74	-0.52	270	760	800	850	940	1030	1105
3	Nay Pyi Taw	Myanmar	19.75	96.1	0	60	930	1020	1180	1320	1461

Exercise 3 searching for rank

Find the city with a population just larger than the seventh largest city in 1950

Cid	City	Country	Latitude	Longitude	pop1950	pop2005	pop2010	pop2015	pop2020	pop2025	pop2050	pop2005idx	pop1950idx
25	Beijing	China	39.9	116.38	4330	10720	11110	11740	12840	13810	14545	1	1
13	MontrÈal	Canada	45.54	-73.65	1340	3600	3680	3780	3910	4010	4108	5	2
18	Santiago	Chile	-33.43	-70.65	1320	5600	5720	5880	6080	6220	6310	2	3
4	Yangon	Myanmar	16.87	96.12	1300	3930	4090	4350	4840	5360	5869	4	4
15	Toronto	Canada	43.72	-79.41	1070	5040	5210	5450	5690	5830	5946	3	5
16	Vancouver	Canada	49.27	-122.96	560	2090	2150	2220	2310	2380	2444	7	6
24	Baotou	China	40.65	109.8	530	1920	2040	2210	2470	2690	2869	8	7
7	El DjazaÔr	Algeria	36.78	3.05	520	3200	3350	3570	3920	4240	4499	6	8
1	Sofia	Bulgaria	42.7	23.33	520	1170	1180	1210	1230	1240	1236	14	9
20	Anshan, Liaoning	China	41.11	122.97	480	1610	1640	1700	1860	2030	2167	11	10
6	Phnum PÈnh	Cambodia	11.56	104.91	360	1360	1470	1650	2030	2460	2911	13	11
19	Valparaìso	Chile	-33.02	-71.55	330	840	850	880	920	960	982	23	12
5	Minsk	Belarus	53.89	27.57	280	1780	1800	1850	1880	1880	1883	9	13
14	Ottawa-Gatineau	Canada	45.37	-75.65	280	1120	1140	1180	1230	1270	1315	15	14
8	Wahran	Algeria	35.74	-0.52	270	760	800	850	940	1030	1105	25	15
26	Bengbu	China	32.93	117.35	230	870	890	940	1040	1140	1225	21	16
21	Anshun	China	26.25	105.93	190	820	850	900	990	1080	1164	24	17
23	Baoding	China	38.85	115.48	180	1040	1110	1210	1360	1480	1586	17	18
2	Mandalay	Myanmar	21.97	96.08	170	920	960	1030	1170	1310	1446	19	19
12	Edmonton	Canada	53.55	-113.5	160	1020	1060	1110	1170	1220	1256	18	20
11	Calgary	Canada	51.03	-114.04	130	1060	1110	1180	1260	1300	1345	16	21
22	Anyang	China	36.1	114.33	110	850	890	950	1060	1160	1240	22	22
9	Douala	Cameroon	4.13	9.7	100	1770	1910	2110	2420	2720	2996	10	23
10	YaoundÈ	Cameroon	3.86	11.51	30	1490	1610	1790	2060	2310	2549	12	24
17	N'DjamÈna	Chad	12.1	15.24	20	900	990	1130	1400	1750	2172	20	25
3	Nay Pyi Taw	Myanmar	19.75	96.1	0	60	930	1020	1180	1320	1461	26	26

Indexing

Index

Indexing

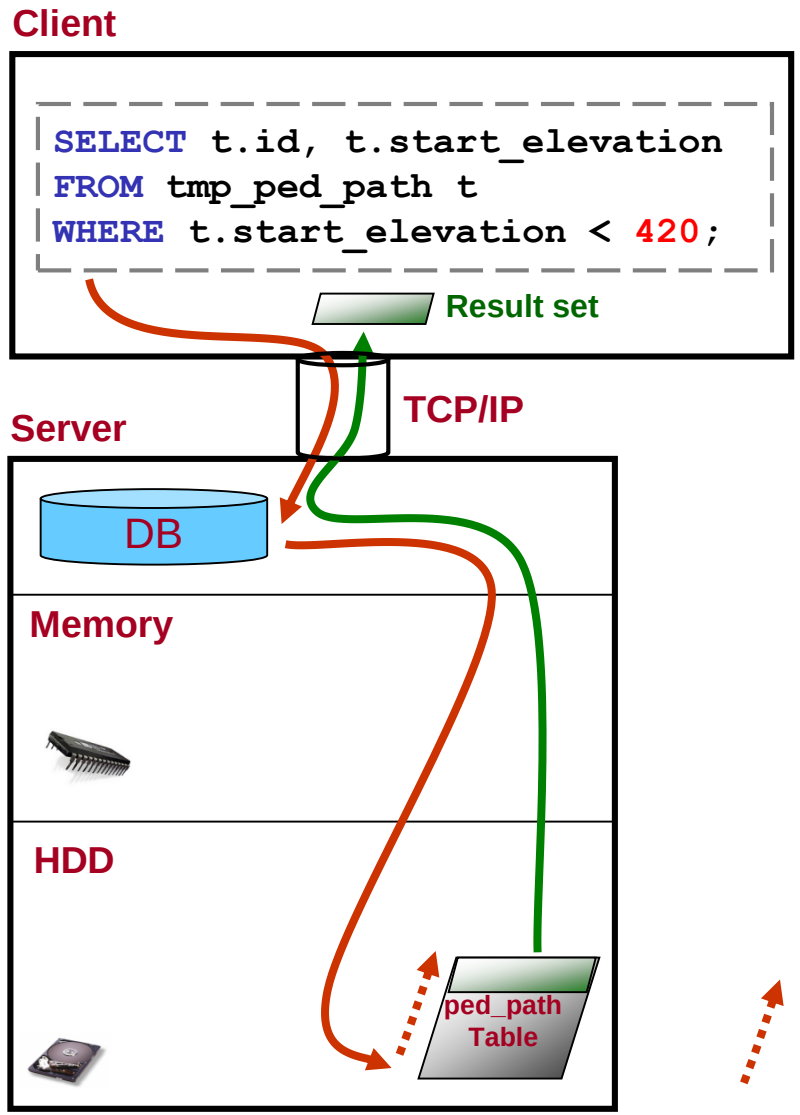
- Index: a data structure that improves the speed of retrieval of data matching a query;
- A way to speed up the identification of records in a database based on a given attribute – avoids having to inspect **all** records for matching values;
- Imposes an **ordering** on records based on the values of an attribute using an **additional structure** (index tables with ordered column values, …)
- Storage of data records can also be **physically** ordered (=on the hard disk; such as clustered index / as a index organized table (IOT); without additional structure); usually not the case in databases; but a book with its pages and page numbers is an index organized table by itself; presorted excel sheets as shown before;

Trade off

- Indices have to be maintained
- Indices take some more storage space; why?

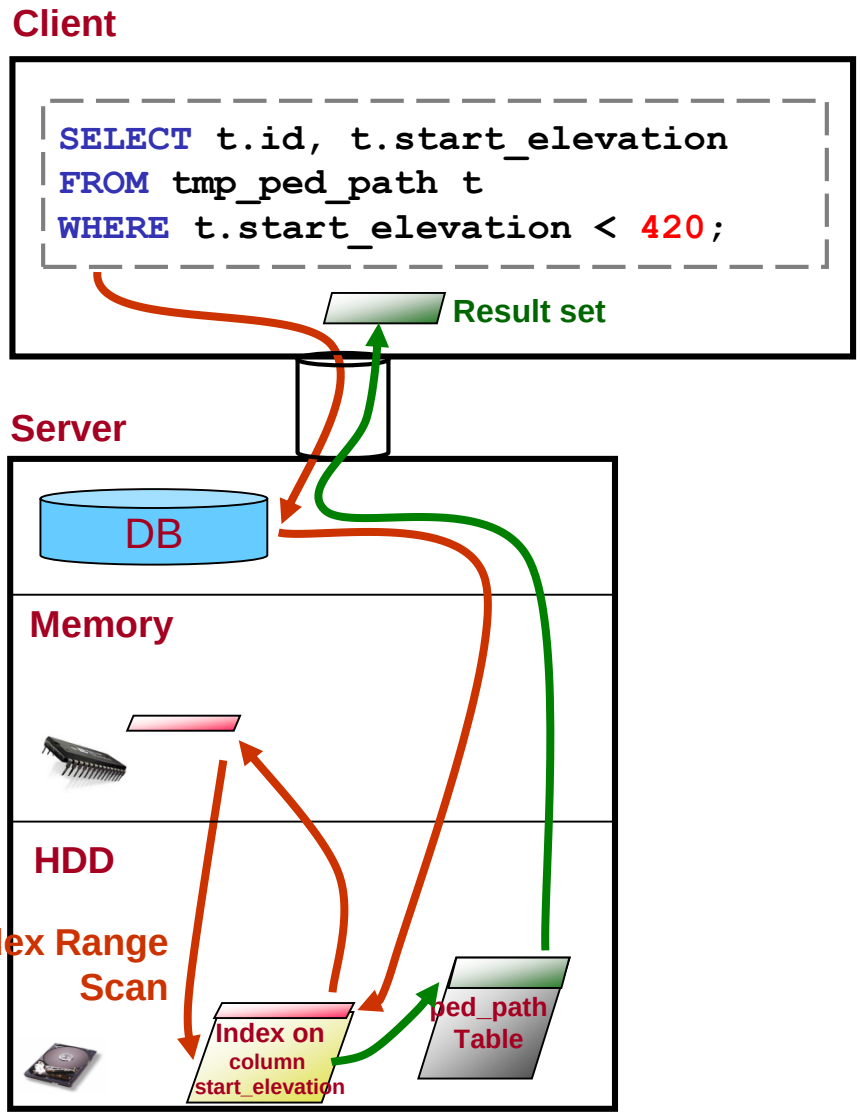
Indexes – used in querying (for numbers, strings and datetime)

Without index



= Full Table Scan!

Lexed



Indexes – DB query explain plan

No Index

```
EXPLAIN ANALYZE
SELECT t.id, t.start_elevation
FROM tmp_ped_path t
WHERE t.start_elevation < 420;
```

	QUERY PLAN
1	Seq Scan on tmp_ped_path t (cost=0.00..1332.83 rows=11448 wid
2	Filter: (hoehe_anfang < '420'::double precision)
3	Rows Removed by Filter: 19267
4	Planning time: 0.127 ms
5	Execution time: 17.311 ms

Indexed, sorted index on column `start_elevation`

```
CREATE INDEX pedpath_startheight_idx ON tmp_ped_path
  USING btree(start_elevation);
```

```
EXPLAIN ANALYZE
SELECT t.id, t.start_elevation
FROM tmp_ped_path t
WHERE t.start_elevation < 420;
```

	QUERY PLAN
1	Bitmap Heap Scan on tmp_ped_path t (cost=221.01..1313.11 rows
2	Recheck Cond: (hoehe_anfang < '420'::double precision)
3	Heap Blocks: exact=949
4	-> Bitmap Index Scan on pedpath_startheight_idx (cost=0.00..21
5	Index Cond: (hoehe_anfang < '420'::double precision)
6	Planning time: 0.498 ms
7	Execution time: 13.022 ms

Indexes – retrieval complexity

- Records with **close values** have **indices of similar values** (are close by). In clustered indices, they are even stored nearby
- On non-spatial tables, direct lookup is of linear complexity (directly proportional to the number of records, *O(n)*);
- Indexing allows to **reduce** this **to non-linear time** (e.g., *O(log(n))*)
- Indices are as well used to police database constraints:
  - NOT NULL;
  - UNIQUE;
  - PRIMARY KEY,… (**there is an index for all your PKs in your DB!**)
- Order in which you specify columns on your composite index (multiple columns) matters

Overview

1. Filter-Refine
2. What Are Indices? What Are They Good for?
3. **Trees – a Bit of Terminology**
4. Spatial Indexing
5. Space-Driven Indices
6. Object-Driven Indices

Exercise 4 recall exercise 2 but searching for value not rank

How would you find the cities with population of 520K in 1950 with least comparison steps (records known to be presorted/presaved on column pop1950)?

Cid	City	Country	Latitude	Longitude	pop1950	pop2005	pop2010	pop2015	pop2020	pop2025	pop2050
25	Beijing	China	39.9	116.38	4330	10720	11110	11740	12840	13810	14545
13	MontrÈal	Canada	45.54	-73.65	1340	3600	3680	3780	3910	4010	4108
18	Santiago	Chile	-33.43	-70.65	1320	5600	5720	5880	6080	6220	6310
4	Yangon	Myanmar	16.87	96.12	1300	3930	4090	4350	4840	5360	5869
15	Toronto	Canada	43.72	-79.41	1070	5040	5210	5450	5690	5830	5946
16	Vancouver	Canada	49.27	-122.96	560	2090	2150	2220	2310	2380	2444
24	Baotou	China	40.65	109.8	530	1920	2040	2210	2470	2690	2869
7	El DjazaÔr	Algeria	36.78	3.05	520	3200	3350	3570	3920	4240	4499
1	Sofia	Bulgaria	42.7	23.33	520	1170	1180	1210	1230	1240	1236
20	Anshan, Liaoning	China	41.11	122.97	480	1610	1640	1700	1860	2030	2167
6	Phnum PÈnh	Cambodia	11.56	104.91	360	1360	1470	1650	2030	2460	2911
19	Valparaìso	Chile	-33.02	-71.55	330	840	850	880	920	960	982
5	Minsk	Belarus	53.89	27.57	280	1780	1800	1850	1880	1880	1883
14	Ottawa-Gatineau	Canada	45.37	-75.65	280	1120	1140	1180	1230	1270	1315
8	Wahran	Algeria	35.74	-0.52	270	760	800	850	940	1030	1105
26	Bengbu	China	32.93	117.35	230	870	890	940	1040	1140	1225
21	Anshun	China	26.25	105.93	190	820	850	900	990	1080	1164
23	Baoding	China	38.85	115.48	180	1040	1110	1210	1360	1480	1586
2	Mandalay	Myanmar	21.97	96.08	170	920	960	1030	1170	1310	1446
12	Edmonton	Canada	53.55	-113.5	160	1020	1060	1110	1170	1220	1256
11	Calgary	Canada	51.03	-114.04	130	1060	1110	1180	1260	1300	1345
22	Anyang	China	36.1	114.33	110	850	890	950	1060	1160	1240
9	Douala	Cameroon	4.13	9.7	100	1770	1910	2110	2420	2720	2996
10	YaoundÈ	Cameroon	3.86	11.51	30	1490	1610	1790	2060	2310	2549
17	N'DjamÈna	Chad	12.1	15.24	20	900	990	1130	1400	1750	2172
3	Nay Pyi Taw	Myanmar	19.75	96.1	0	60	930	1020	1180	1320	1461

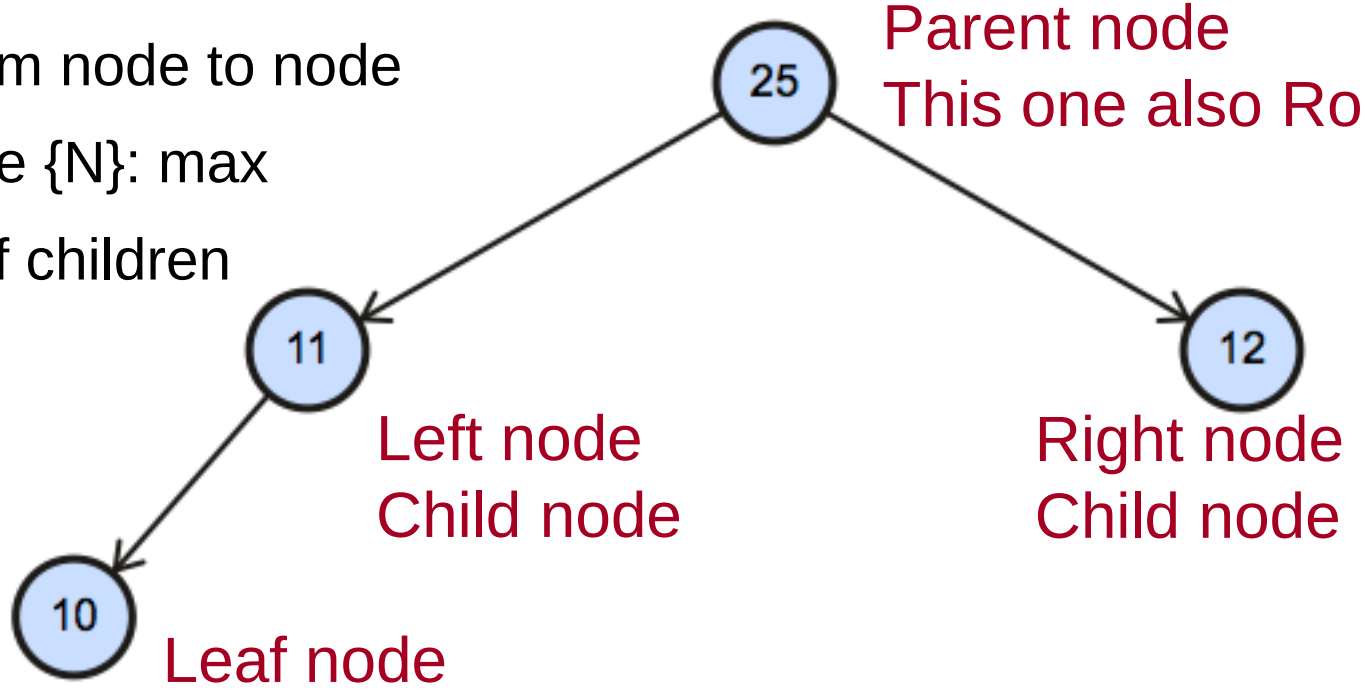
Indexes – tree structures for search and insertion

- All following examples with the **binary tree**
- A **tree** data structure enabling the organisation of data with logarithmic time complexity;
- Captures the idea introduced a moment ago;
- Enables efficient search, sequential access, insertion and deletion of data;
- Has a **natural equivalent array** form, that simplifies sequential search and reduces the storage of blank values in memory;
- The idea of the tree is reflected in a separate structure and must be filled and mainained (in files or in distinct structures/tables in databases)

Trees Principles

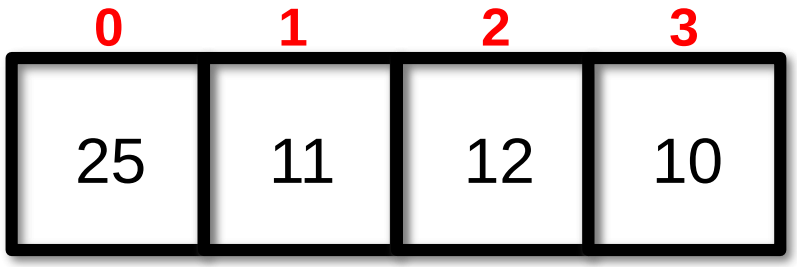
A tree is a directed graph data structure with

- Nodes
- Edges – from node to node
- Order of tree {N}: max num of children here 2



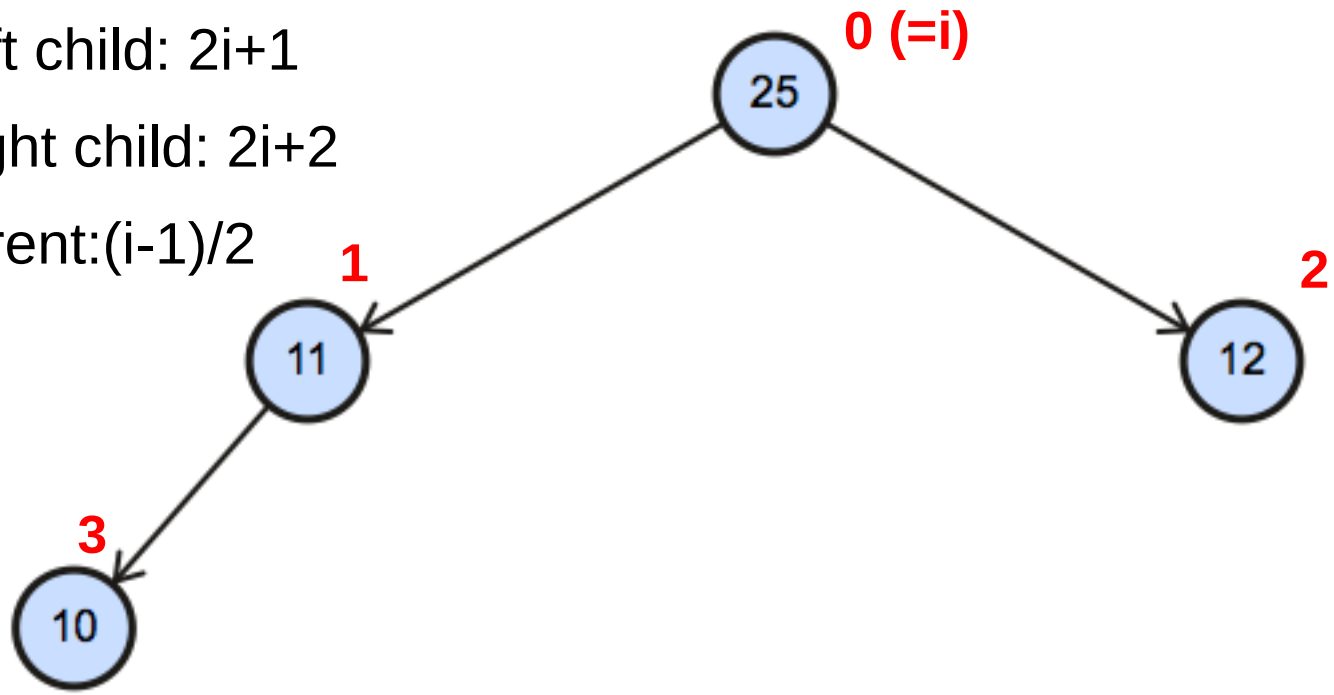
Corresponding array/storage structure:

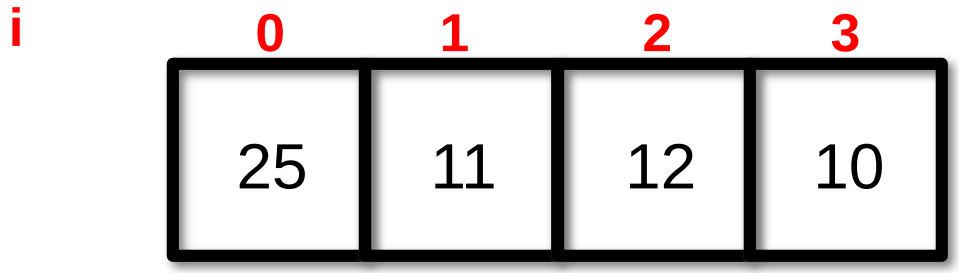
- Values and **cell index**



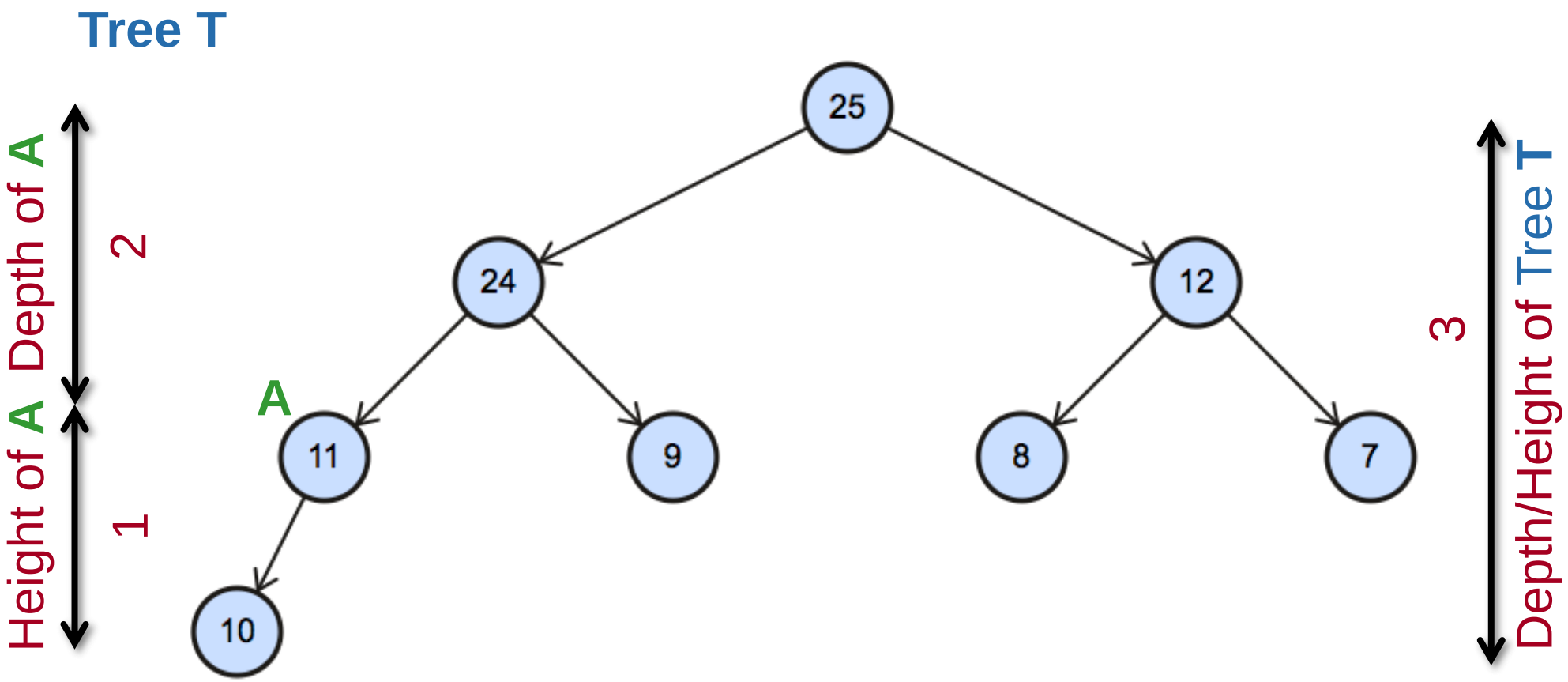
Trees (cont.): Array storage

- Natural mapping for most simple binary search tree between node i and cell value (starting i at 0 {root}):
- Index of Left child: 2i+1
- Index of Right child: 2i+2
- Index of parent:(i-1)/2



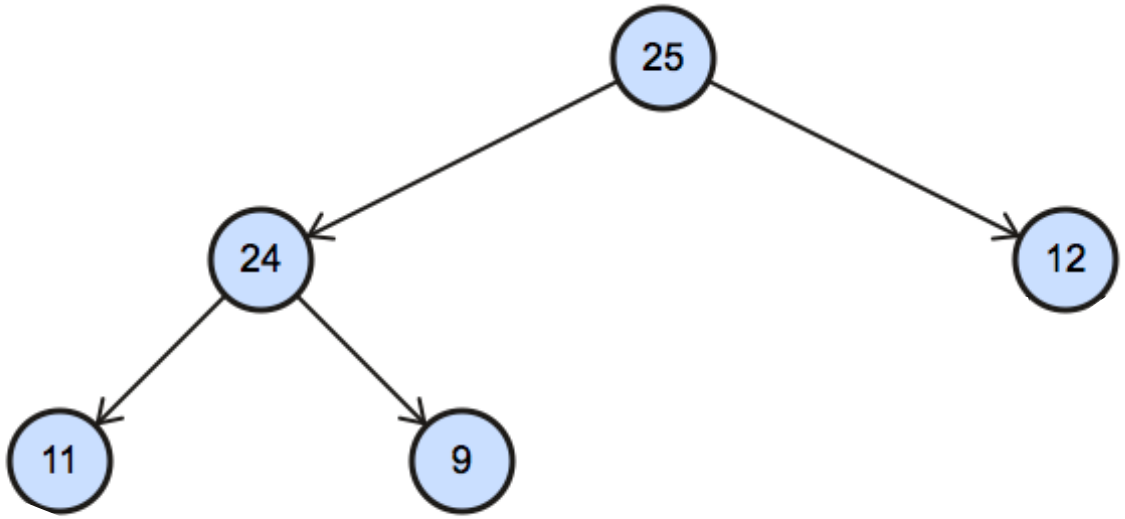


Trees (cont.): terminology



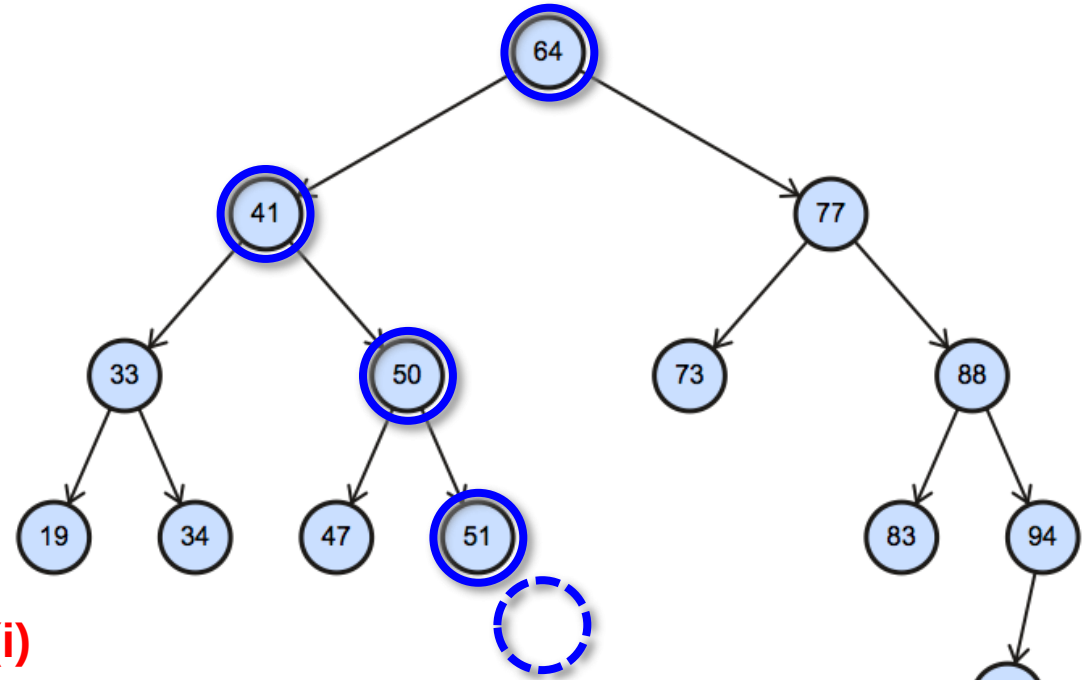
Trees (cont.): Properties

- Full tree: every node has 0 or 2 children (and contains 1 entry)
- Balanced tree: minimum possible height
- Complete tree: Completely filled tree with exc. of bottom row, filled from left to right



Trees: Lookup & Search

- We benefit from tree properties for search and insertion of data
- **Binary search tree** – sorted binary trees (BST trees):
- We can search in a tree (tree lookup)



Find node with value 55

Storage structure: cell index (i)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
64	41	77	33	50	73	88	19	34	47	51			83	94	…

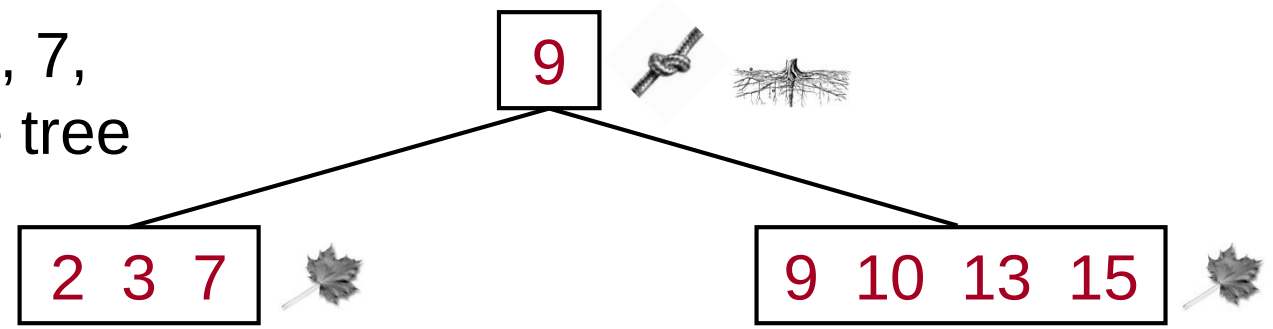
Trees: B$^+$-Tree Definition

- A generalisation of the binary search tree enabling for **more children** are the B-Trees (family of trees, e.g., BinarySortedTree, B$^+$Tree, B*Tree, others)
- Order of tree: maximum number of pointers to children (N)
- Children can be **leafs** or internal **nodes**
- **Root** has at least one key (value), non-root nodes at least N/2 subtrees
- All nodes/root contain **separator keys**
- All leaves are sorted, map to disk space (disk pages) & therefore **map the values** that are indexed
- All leaves are at the same level
- Often used in databases as a 'standard' index

Trees: B+-Tree

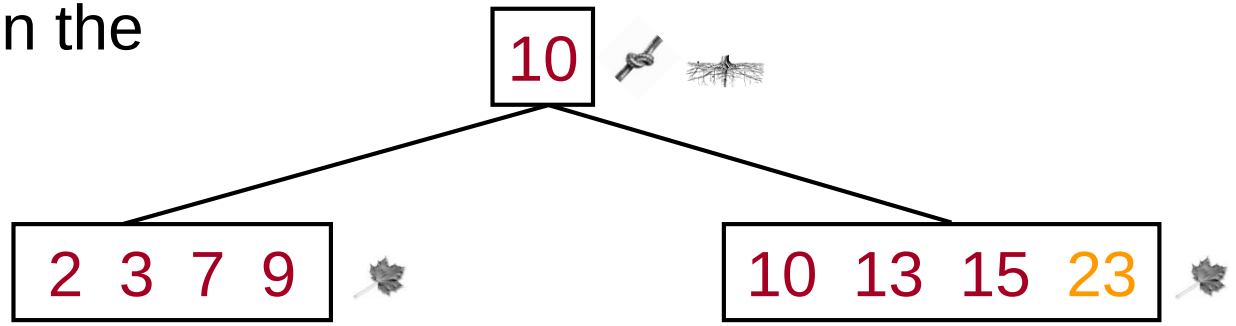
- Tree of order 5 {N}
- Maximum branching 5 {N}
- Root node should only have one entry
- Min 2 {(N-1)/2} entries per node in nodes or leaves (not root)
- Max 4 {N-1} entries per node or leave; 4 entries in a node can separate 5 {N} further nodes
- Will be a non-balanced tree
- All values are in leave nodes as well (unlike BinarySearchTree)

Insertion of 2, 3, 7,
9, 10, 13, 15 the tree

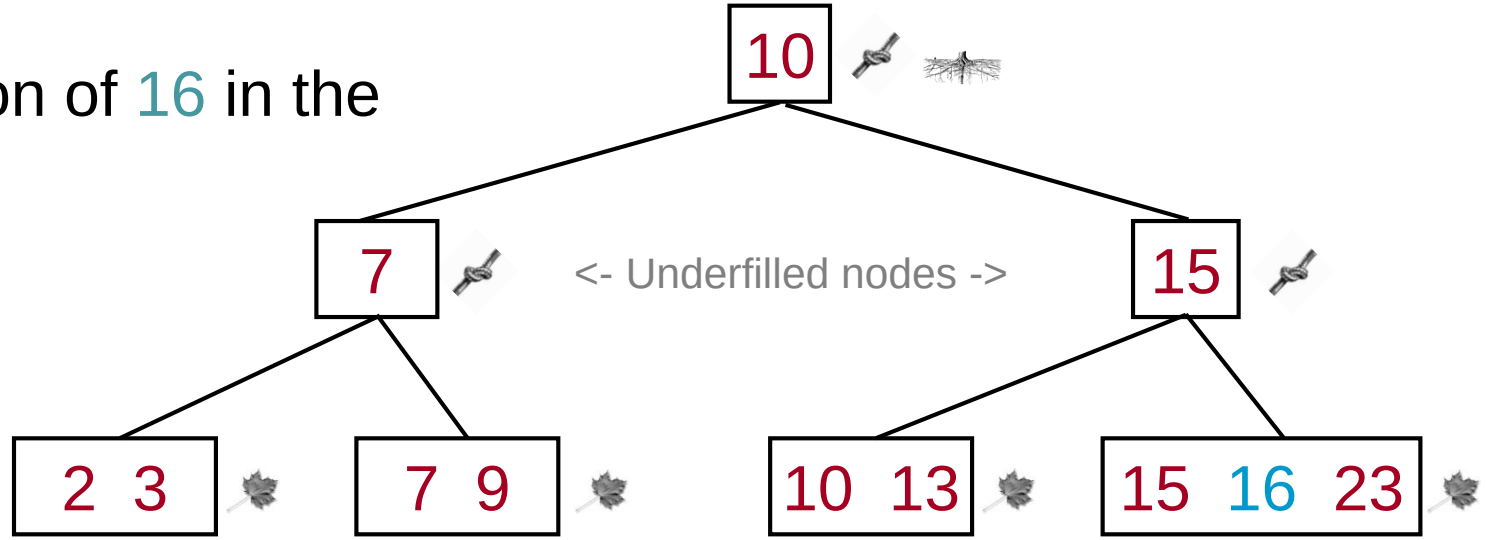


Trees: B+-Tree

Insertion of 23 in the tree

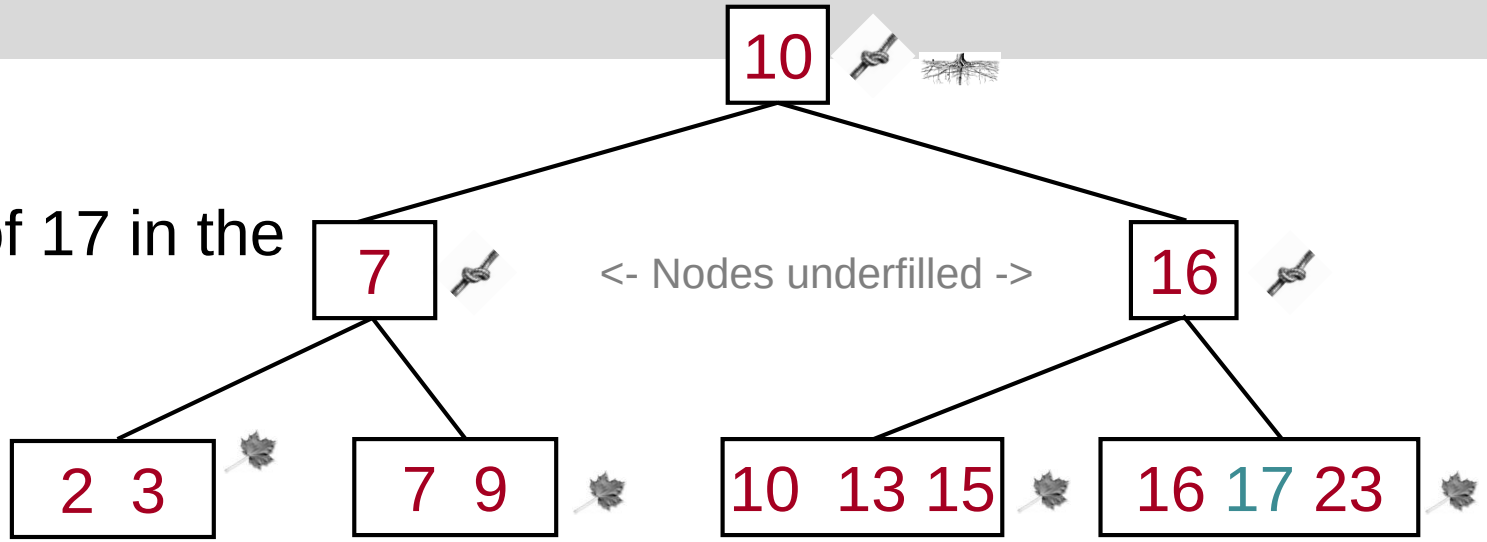


Insertion of 16 in the tree

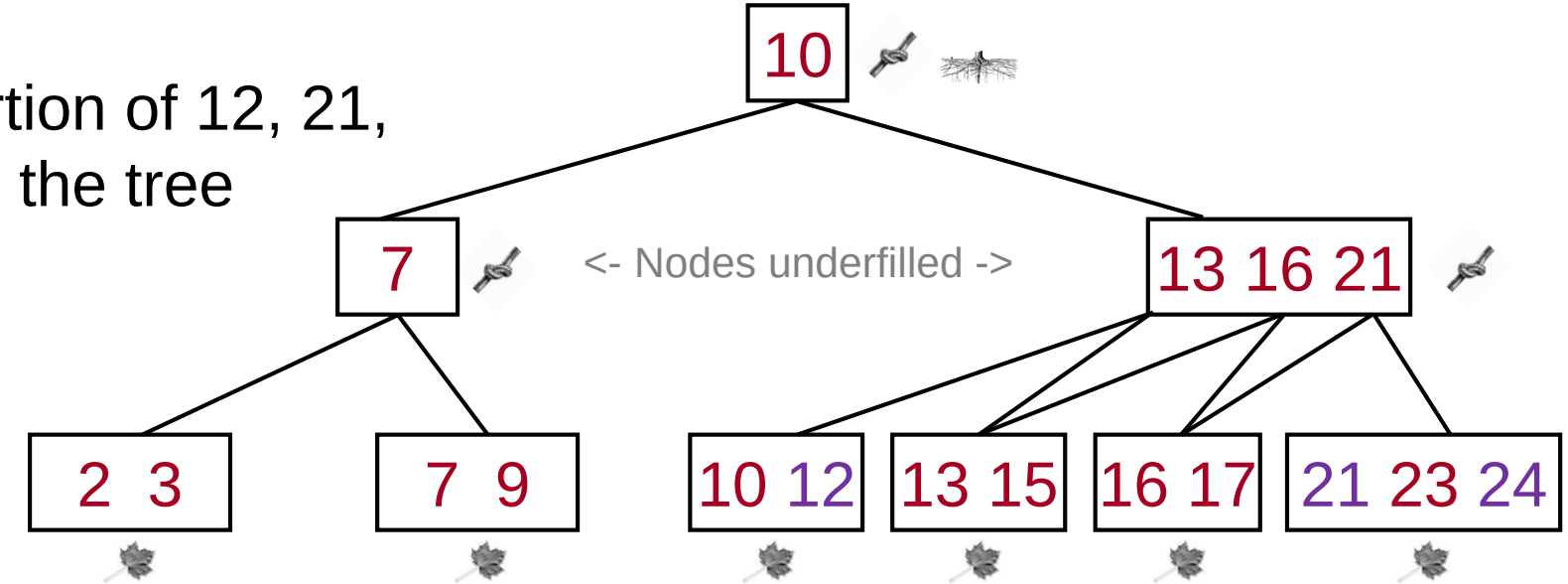


Trees: B+-Tree

Insertion of 17 in the tree



Insertion of 12, 21, 24 in the tree



B+-Tree summarized

- All values in nodes and leaves are **ordered**
- Nodes only contain references to the storage of dependant nodes or leaves (=B+-Tree)
- Contents of nodes are called **separators** and not record values
- Leaves in our example contain indexed data itself
- But leaves in real world indices contain data plus additional references to the physical storage location (pages on disk) of the data; example: further attributes for a record with an identifier;
- Leaves are **interlinked** (sorted) ➔ Range-Scan!

- Recall the excel-sheet value search in exercise 4; B+-Tree materializes this search method in the form of a stored index;

Overview

1. Filter-Refine
2. What Are Indices? What Are They Good for?
3. Trees – a Bit of Terminology
4. **Spatial Indexing**
5. Space-Driven Indices
6. Object-Driven Indices

Spatial indexes

- Generalisation of the previous ideas into 2+ dimensions
- We need to sort things to order and number them in space somehow.
- We want close things to be close together in an index;
- No natural way to sort x and y coordinate!

City	Latitude	Longitude	pop1950
Beijing	39.9	116.38	4330
Montréal	45.54	-73.65	1340
Santiago	-33.43	-70.65	1320
Yangon	16.87	96.12	1300
Toronto	43.72	-79.41	1070
Vancouver	49.27	-122.96	560
Baotou	40.65	109.8	530
El DjazaÔr	36.78	3.05	520
Sofia	42.7	23.33	520
Anshan, Liaoning	41.11	122.97	480
Phnum PÈnh	11.56	104.91	360
Valparaìso	-33.02	-71.55	330
Minsk	53.89	27.57	280
Ottawa-Gatineau	45.37	-75.65	280

Spatial indexes

Two approaches:

a) Space-driven indexing
   Space is regularly partitioned into regions (cells) with an assigned index value)

b) Data (content) driven indexing

Spatial indexes

- Problems
  - We have points, irregular lines, weird shaped polygons; all eventually "multi-";
  - Sizes of objects can vary largely
  - Application type: frequent writes or frequent reads?
- Compromises are necessary, and depend on the priorities
  - What kind of searches (range, exact,…)
  - How often we insert new objects?
  - How costly is the update of the index?

Spatial indexes - Summary: common approach

- We divide space or group objects based on **some** criteria;
- These spatial divisions or groups are somehow numbered (indexed);
- The numbering is meaningful, in order to support sequential search;
- Problem:
  - large objects, long objects, concave objects, many vertices
- Solution: hierarchical divisions
- Additional strategy: simplifying the geometry objects for index storage

Overview

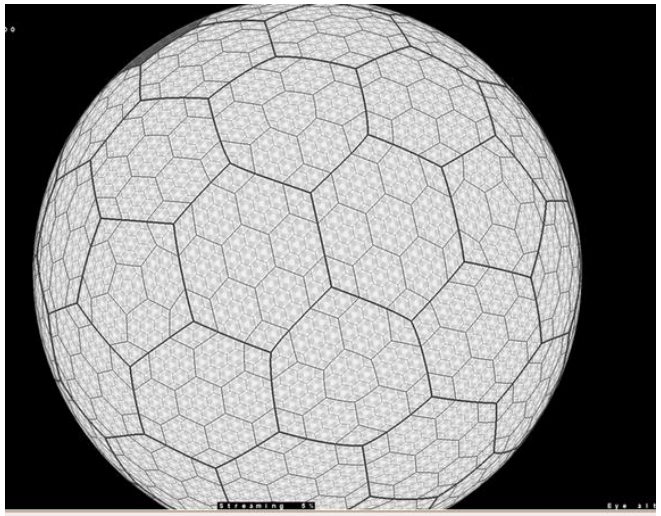
1. Filter-Refine
2. What Are Indices? What Are They Good for?
3. Trees – a Bit of Terminology
4. Spatial Indexing
5. **Space-Driven Indices**
6. Object-Driven Indices

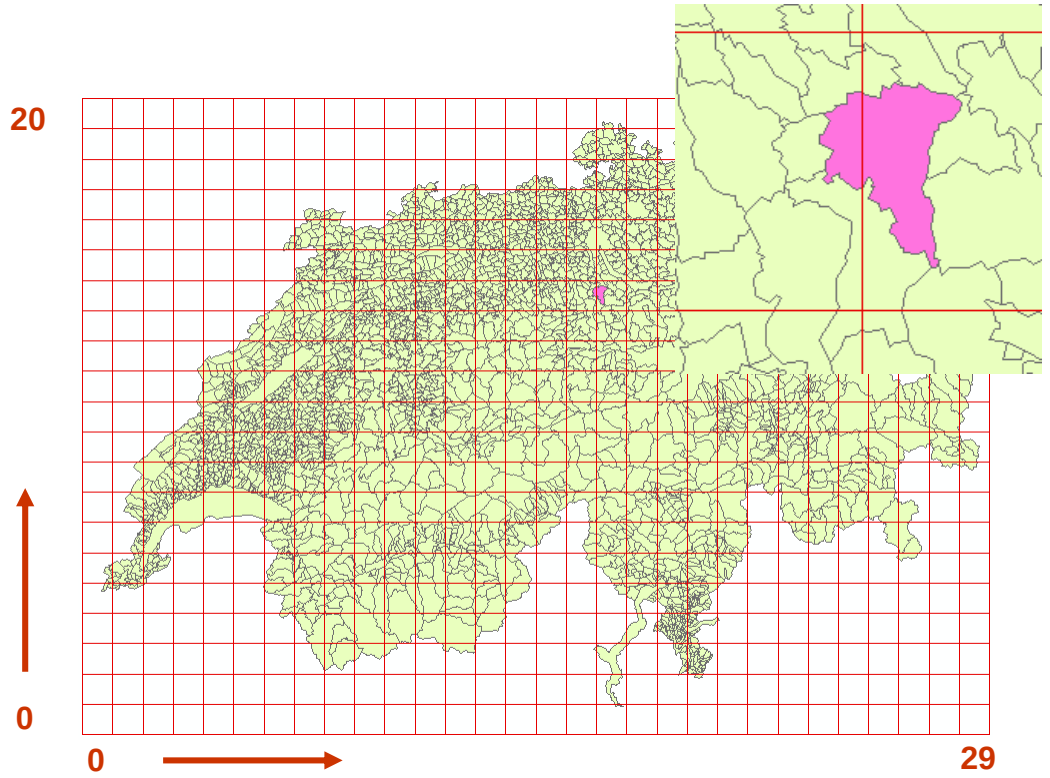
Space-driven spatial indexes

- We divide space into regular cells
- These cells may be hierarchically organised
- Cells may be always present (index slots always stored), or they may be generated based on the data inserted – but the scheme defines the division

Space-driven spatial indexes: Grid index

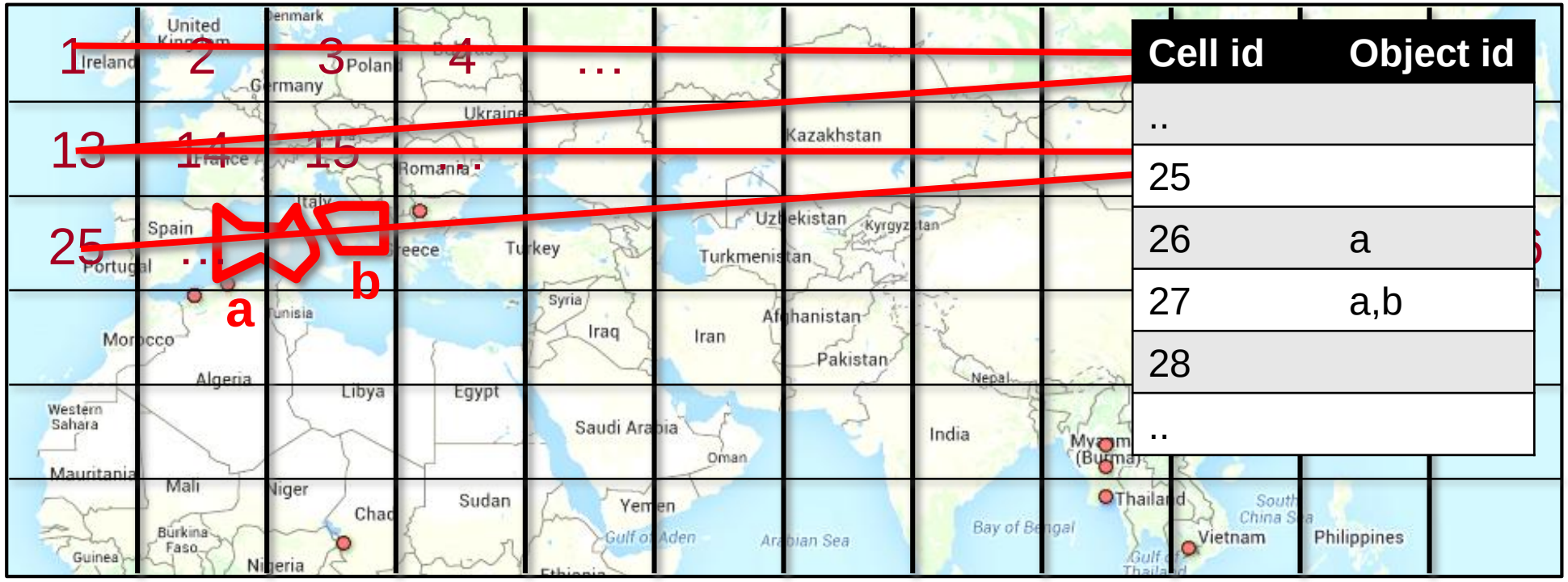
- We divide space into regular and fixed sized cells
- Objects get an address in this space – either numbered, or as x and y index
- One object may be assigned to multiple cells





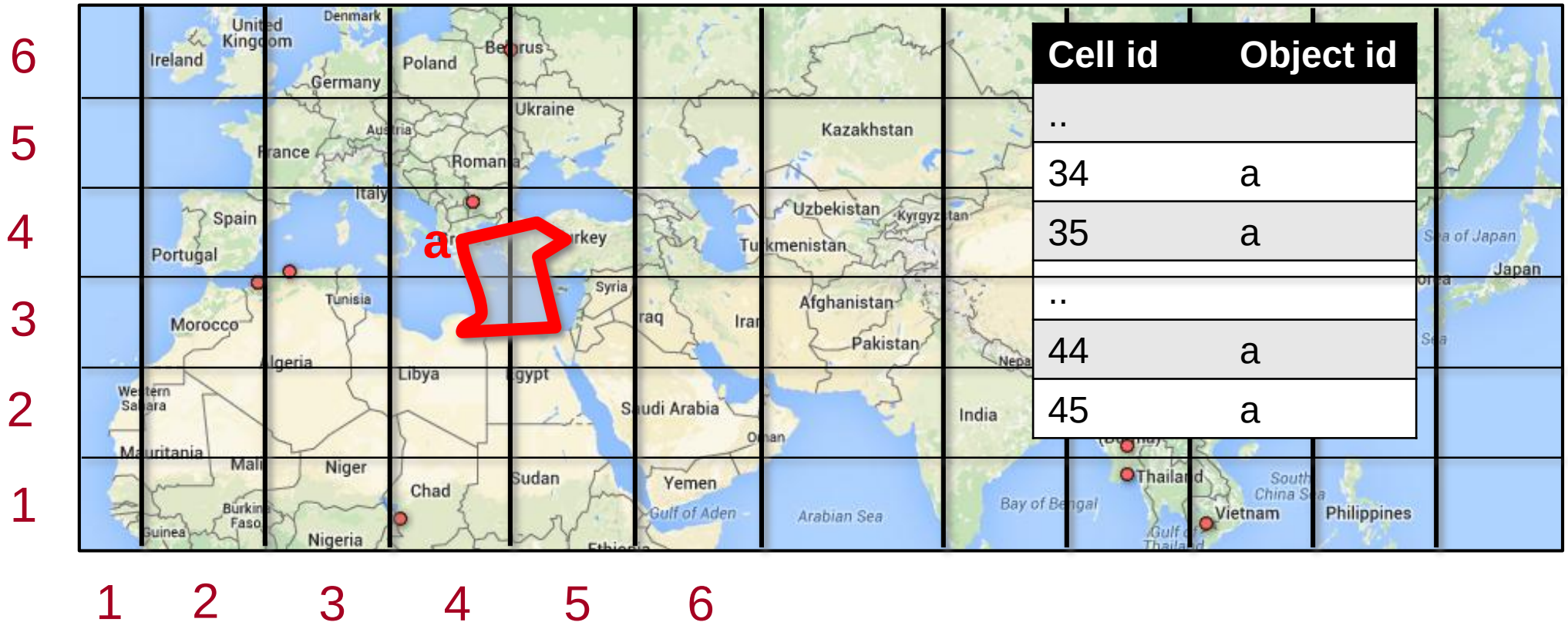
Space-driven spatial indexes: Grid index

- Sorting **of cells** is based on binary sorts (e.g., B-Tree)
- Numbering of cells following some logical order:
  - Row ordering (below) and others



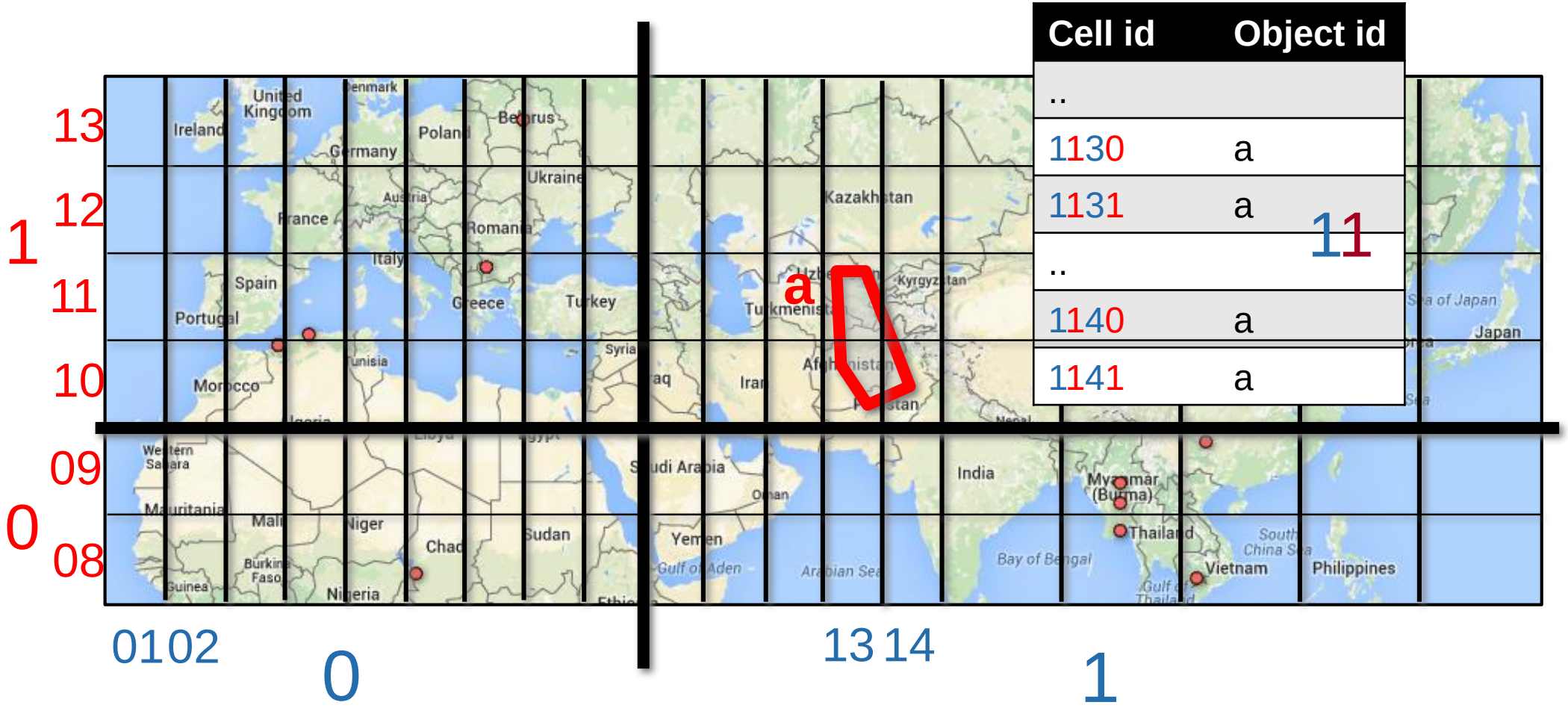
Space-driven spatial indexes: Grid index

- Sorting is based on binary sorts (e.g., B-Tree)
- Interleaved numbering of cells:



Space-driven spatial indexes: (Fixed) Grid index

- Numbering cells may be smart – and may have hierarchical implications
- Interweaved cell id values



Grid Indices: Pros & Cons

Pros

- Simple application of 1D sorting;
- Simplicity

Cons

- Inefficient in terms of storage size, may include large amounts of empty, pre-allocated space or redundancy
- Depending on object distribution, may be less efficient for range search (object close in space may be far in index);

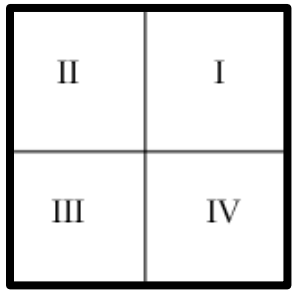
Space-driven spatial indexes: Quad tree index

- From the interleaving example: bringing in hierarchical partitioning is clever – helps deal with objects of different sizes;

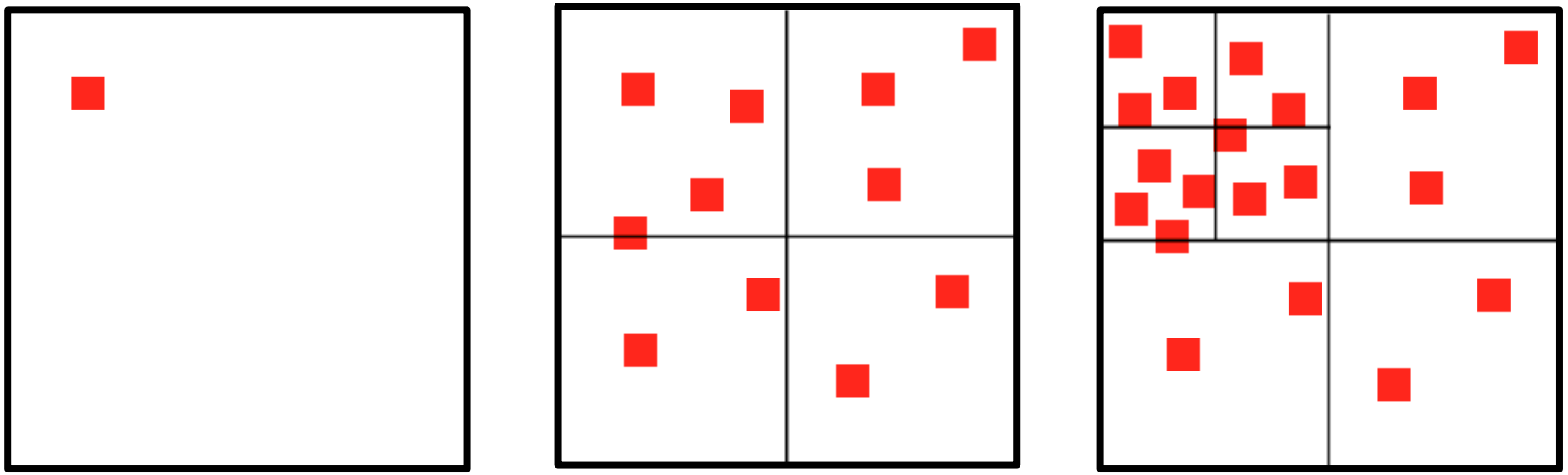
Quadtrees

- Divide space into hierarchically nested, adaptable cells;
- Each cell has a limit size (N of objects)
- The directory (the index array) follows a spatially motivated numbering scheme.
- Recursive (self-similar) numbering – space filling curves

Space-driven spatial indexes: Quad trees

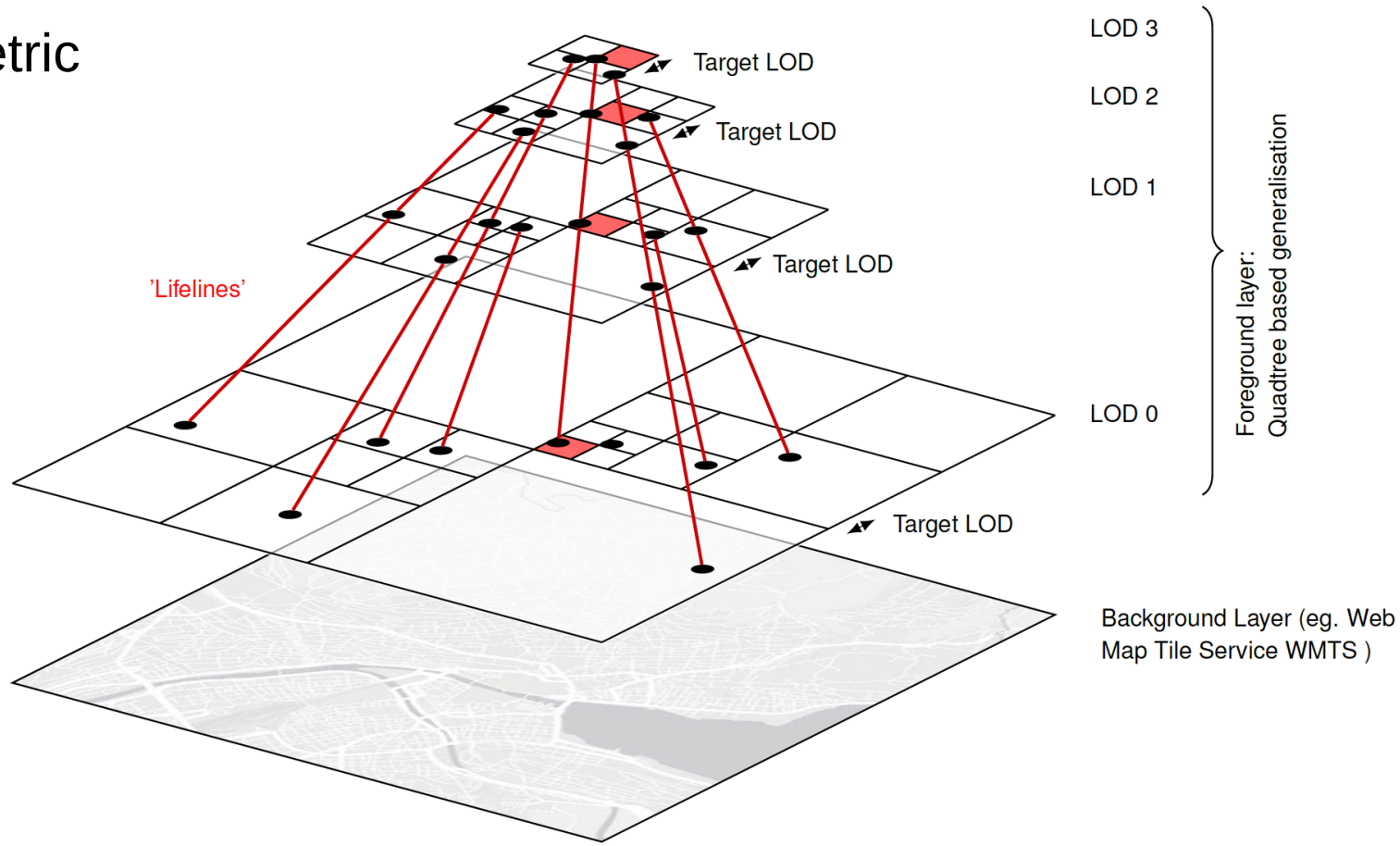


- Hierarchial regular subdivision – many variants
- Built up dynamically to fill subdivisions



Space-driven spatial indexes: Quad trees

- Tree is calculated based on grouping algorithms
- Attributes and geometric distance lead to subdivision



Bereuter 2015, Thesis: Quadtree-based Real-time Point Generalisation for Web and Mobile Mapping

Quadtrees: Pros&Cons

Pros

- Usually do not require recomputing after insertion/deletion of new geometry
- Simple to index using modifications of standard 1D sorting;
- Useful for raster indexing!

Cons

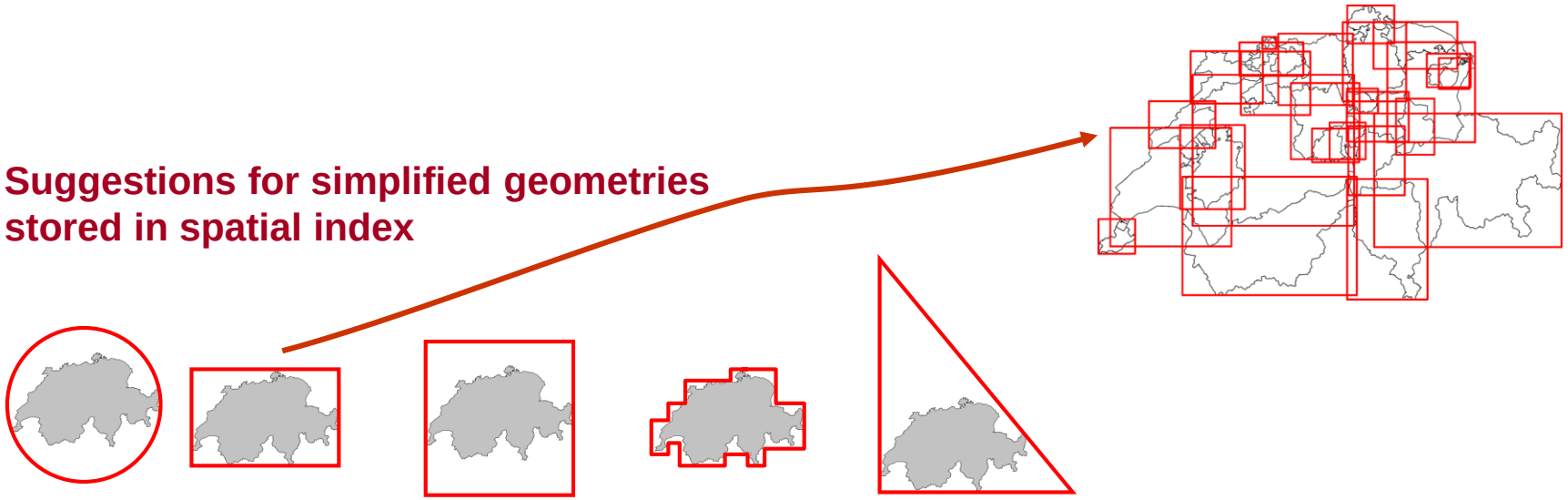
- Depending on object distribution, may be less efficient for range search (objects close but far in index);
- Trees can be unbalanced in depth – hard to estimate retrieval time;
- Depends on the order in which objects are inserted

Overview

1. Filter-Refine
2. What Are Indices? What Are They Good for?
3. Trees – a Bit of Terminology
4. Spatial Indexing
5. Space-Driven Indices
6. **Object-Driven Indices**

Object driven indexes

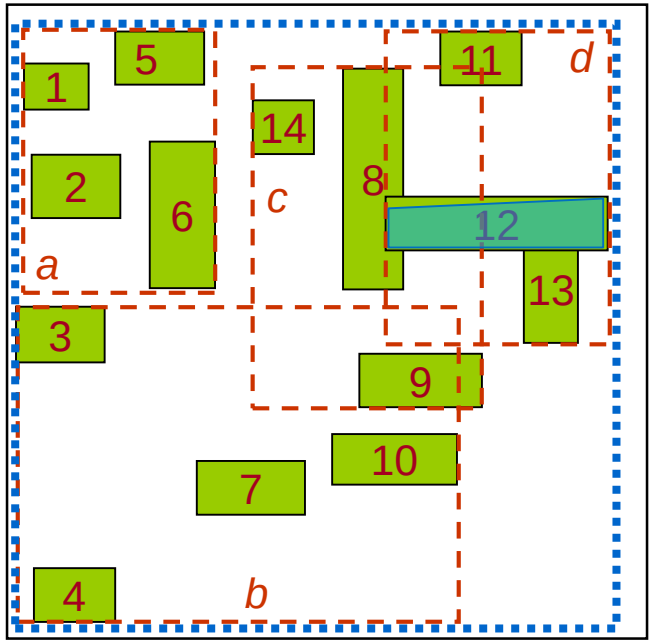
- Adapt to the distribution of objects (approximated by bounding shapes) in the plane/space;
- Do not fully cover space – only space occupied by spatial objects;
- Based on containment relationship, recursively
- Rely on a balanced hierarchical structure with direct mapping to disk space (=page)



R-Tree indexes

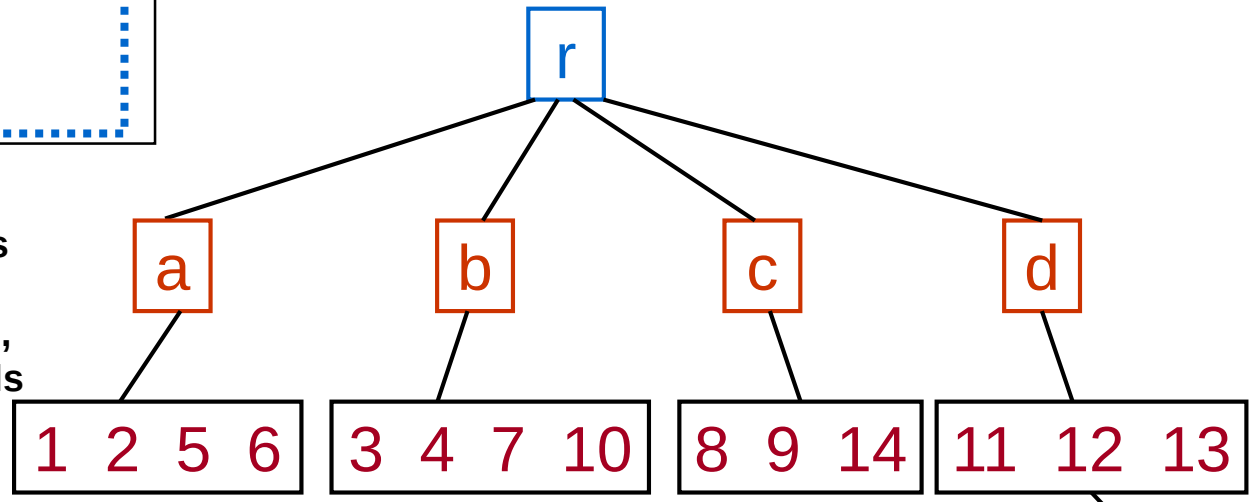
- Various refinements: R*Tree, R$^+$Tree, STR-Tree…
- Main principle:
  - **Depth-balanced tree** (maps well to B$^+$Tree) – all leaves on the same level;
  - Each node corresponds to a disk page
  - Each leaf node contains an array of leaf entries as tuple *[(**mbb**, **objectreference_on_disk**)]*;
  - Non-leaf nodes contain an array of node entries *[(**mbb, reference_node_pointer**)]*;
  - ***M*** is the maximum number of entries per node
  - Number of entries **x** in node should comply with ***m<x<M***, where ***m in [0,M/2]*** (avoid underfilling of nodes storage structure)

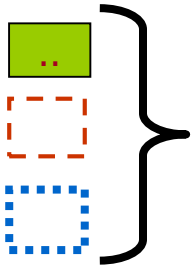
R-Tree indexes



-> Spatial index with its geographical extent

Tree with M=4 in this example



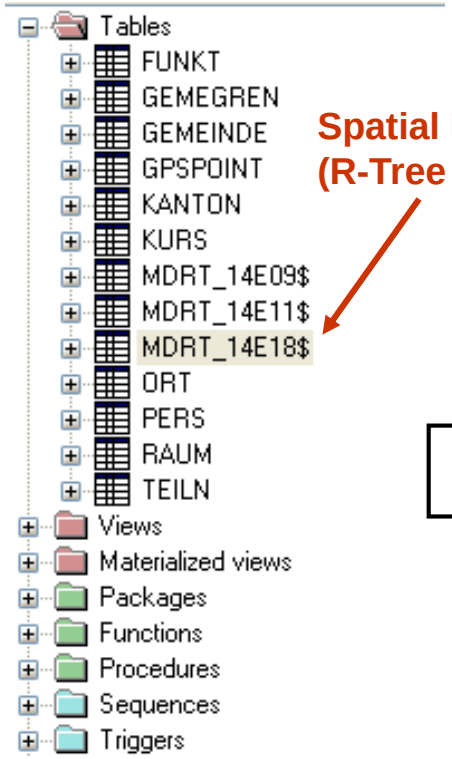


Minimum bounding boxes of objects, directory nodes, of two diff. levels

Spatial object (here a polygon) stored in the spatial table

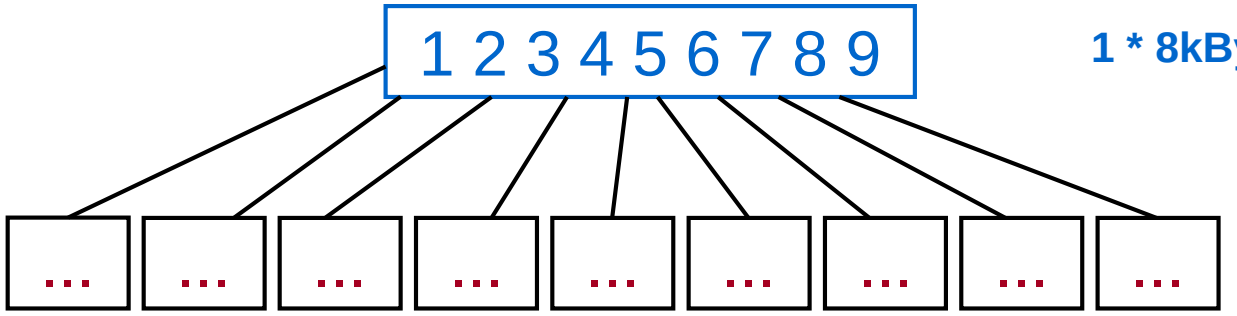
R-Tree index of a database table can be a table as well

- Example spatial table `gemegren` (geometries of communities) in Oracle Spatial. Table has a spatial index on its geometry attribute.
- Oracle can store 34 entries per node. A node has a capacity of 8kB , equivalent to max 34 records incl. MBB information



Spatial index (R-Tree as a Table)

Root and nodes



1 * 8kByte

Leave nodes

9 * 8kByte

The entries for the MBBs of the geometries of the 190 Gemeinden are stored in these leaves.

R-Tree indexes – under the cover – root node

SQL | Output | Statistics

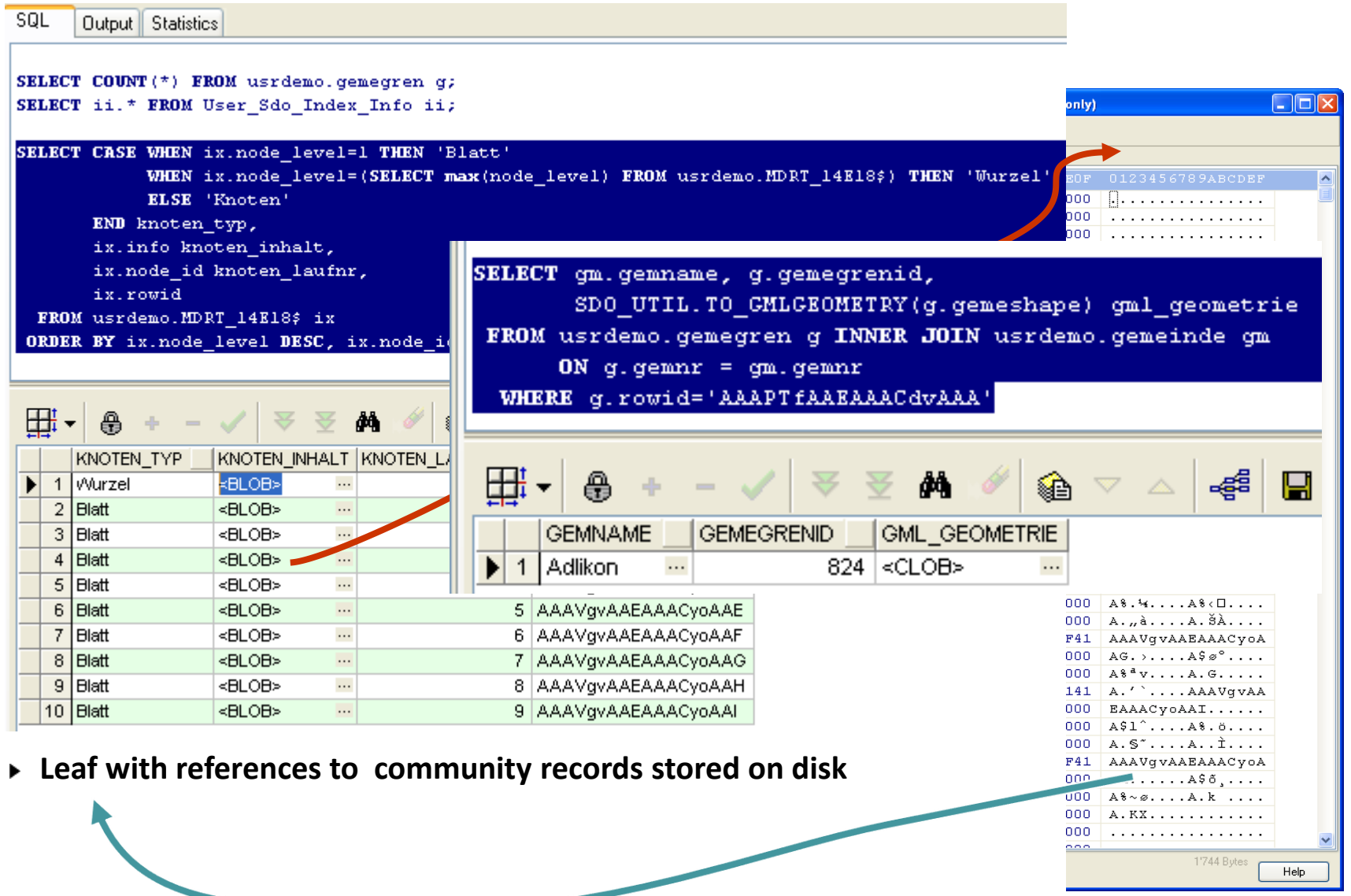
```
SELECT COUNT(*) FROM usrdemo.gemegren g;
SELECT ii.* FROM User_Sdo_Index_Info ii;

SELECT CASE WHEN ix.node_level=1 THEN 'Blatt'
            WHEN ix.node_level=(SELECT max(node_level) FROM usrdemo.MDRT_14E18$) THEN 'Wurzel'
            ELSE 'Knoten'
       END knoten_typ,
       ix.info knoten_inhalt,
       ix.node_id knoten_laufnr,
       ix.rowid
  FROM usrdemo.MDRT_14E18$ ix
 ORDER BY ix.node_level DESC, ix.node_id ASC;
```

	KNOTEN_TYP	KNOTEN_INHALT	KNOTEN_LAUFNR	ROWID
1	Wurzel	<BLOB>	10	AAAVgvAAEAAACykAAA
2	Blatt	<BLOB>	1	AAAVgvAAEAAACyoAAA
3	Blatt	<BLOB>	2	AAAVgvAAEAAACyoAAB
4	Blatt	<BLOB>	3	AAAVgvAAEAAACyoAAC
5	Blatt	<BLOB>	4	AAAVgvAAEAAACyoAAD
6	Blatt	<BLOB>	5	AAAVgvAAEAAACyoAAE
7	Blatt	<BLOB>	6	AAAVgvAAEAAACyoAAF
8	Blatt	<BLOB>	7	AAAVgvAAEAAACyoAAG
9	Blatt	<BLOB>	8	AAAVgvAAEAAACyoAAH
10	Blatt	<BLOB>	9	AAAVgvAAEAAACyoAAI

...etc

- Cryptic: the content of the root node points to the 9 rowids of the leaves
- The MBB of the respective rowid

R-Tree indexes – under the cover – leaf node



- **Leaf with references to community records stored on disk**

R-Tree indexes – Operation on indexes

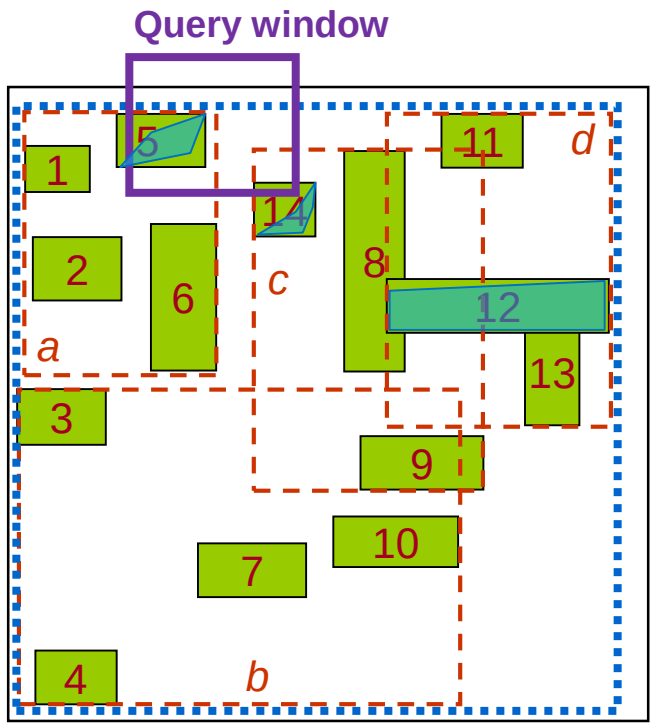
Search

- Lookup = Spatial search = Filter Step

Maintenance

- Insertion
- Splitting (of node)
- Deletion
- Update (deletion + insertion)

R-Tree indexes – Lookup/spatial search

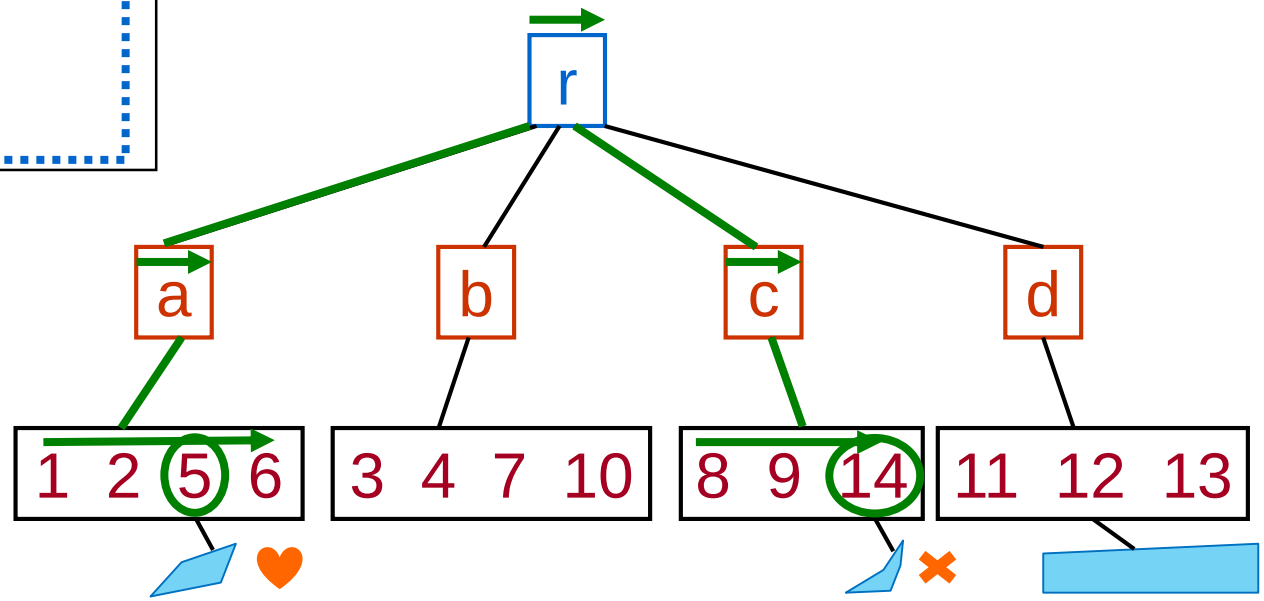


Compares the query window (or the bounding box of an irregular query geometry) with the incidence of all the nodes‘ MBBs (directory or leaf) in the tree, hierarchically.

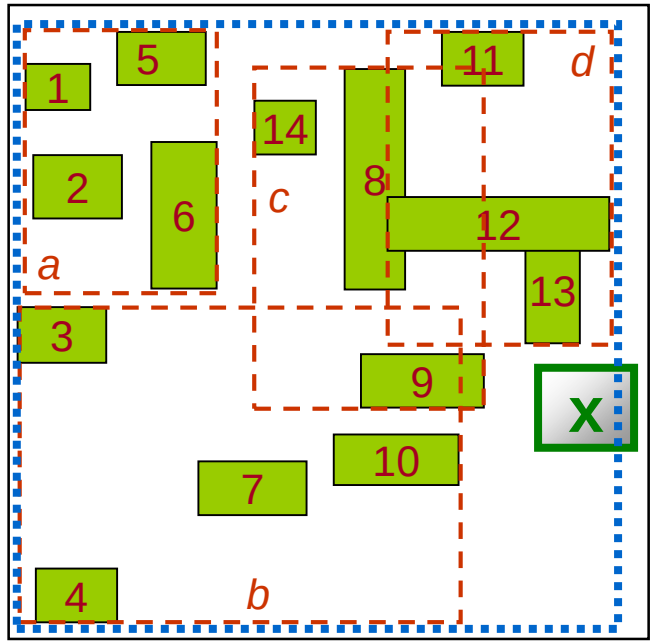
The result is a coarse filter - returns all the geometries with incident bounding boxes. Refinement happens in the second step, where all the geometries (identified by their bounding boxes) and the query geometry are matched with full algorithmic complexity.

Checks entries of the nodes

Geometries to be compared in refine step; not part of index algorithms;



R-Tree indexes: Insertion



For all entries of each node, check:

1. Object inside of a MBB?

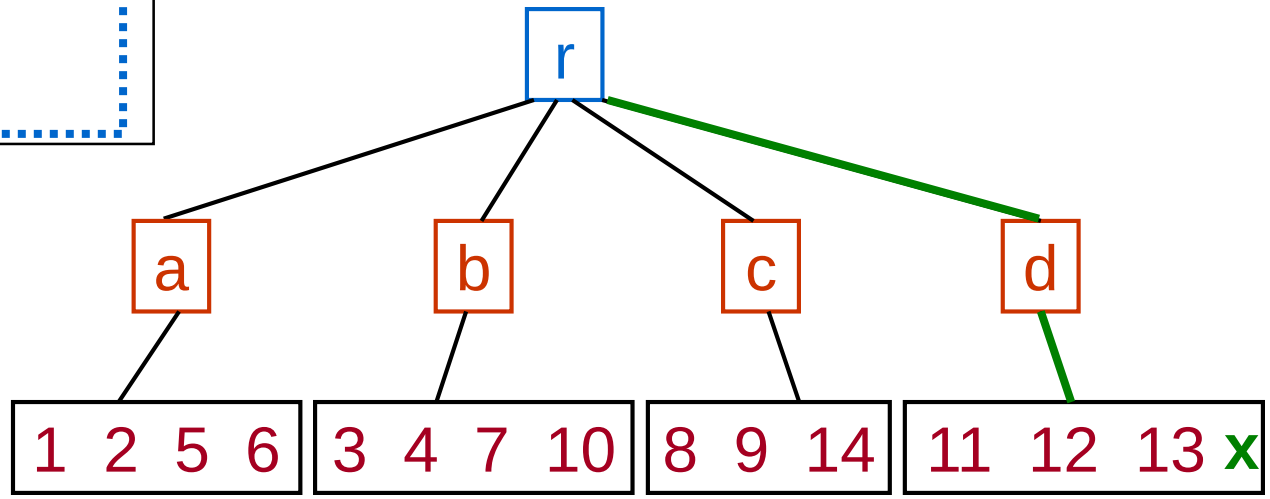
→ **select entry**

2. Object inside multiple MBBs?

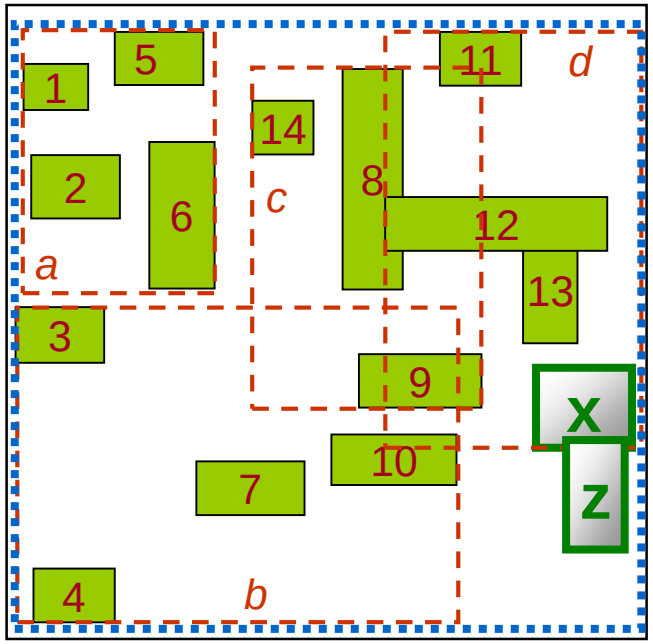
→ **Select entry with smallest area so far**

3. Object outside, or cut?

→ **Select entry that needs to be extended by smallest amount**

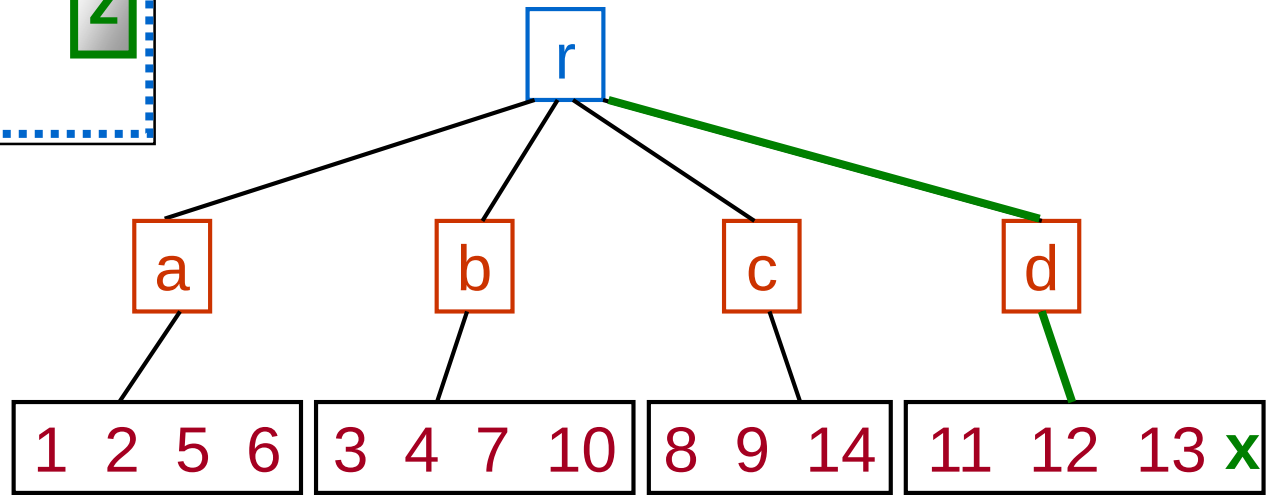


R-Tree indexes: Insertion

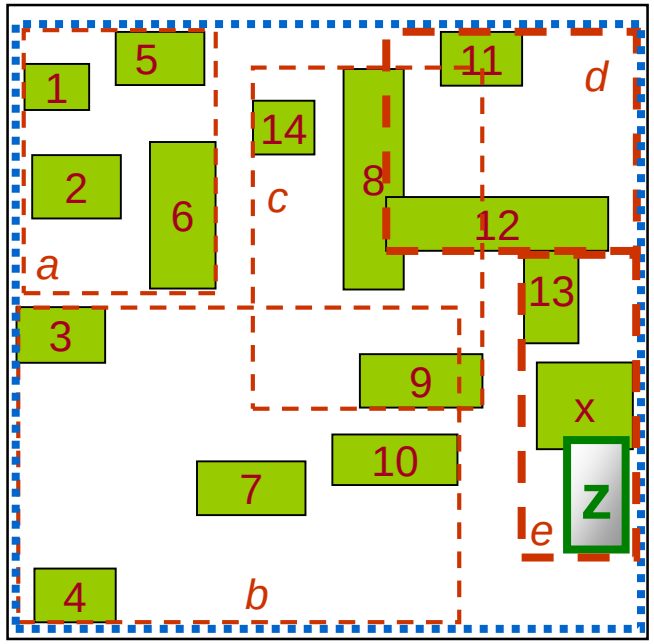


Adding z

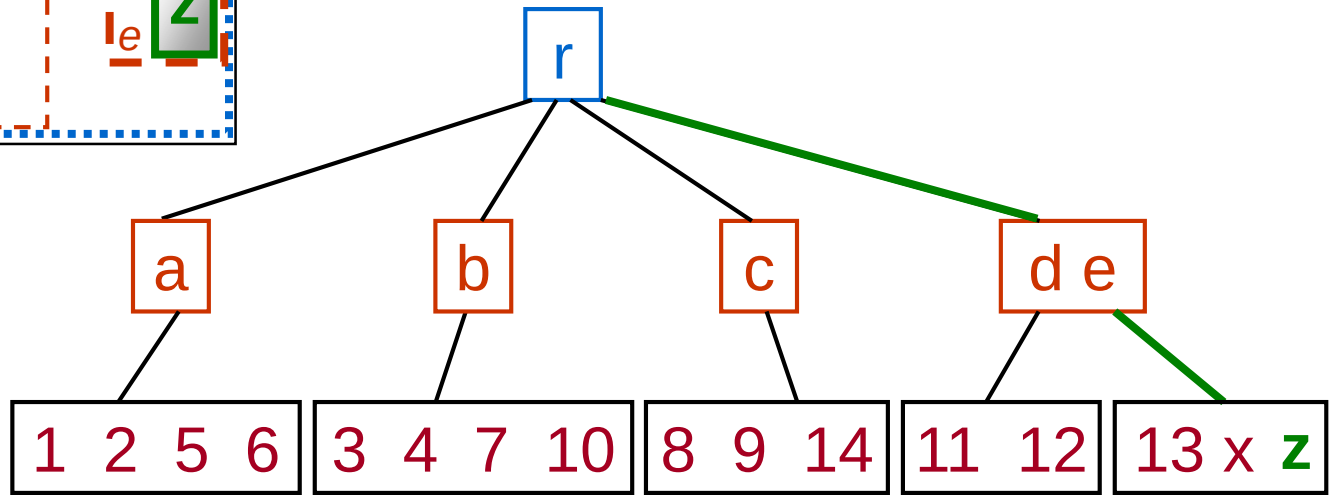
Consider the maximum capacity of a node to be 4.



R-Tree indexes: Insertion



We must apply split algorithms to find the right place for z.



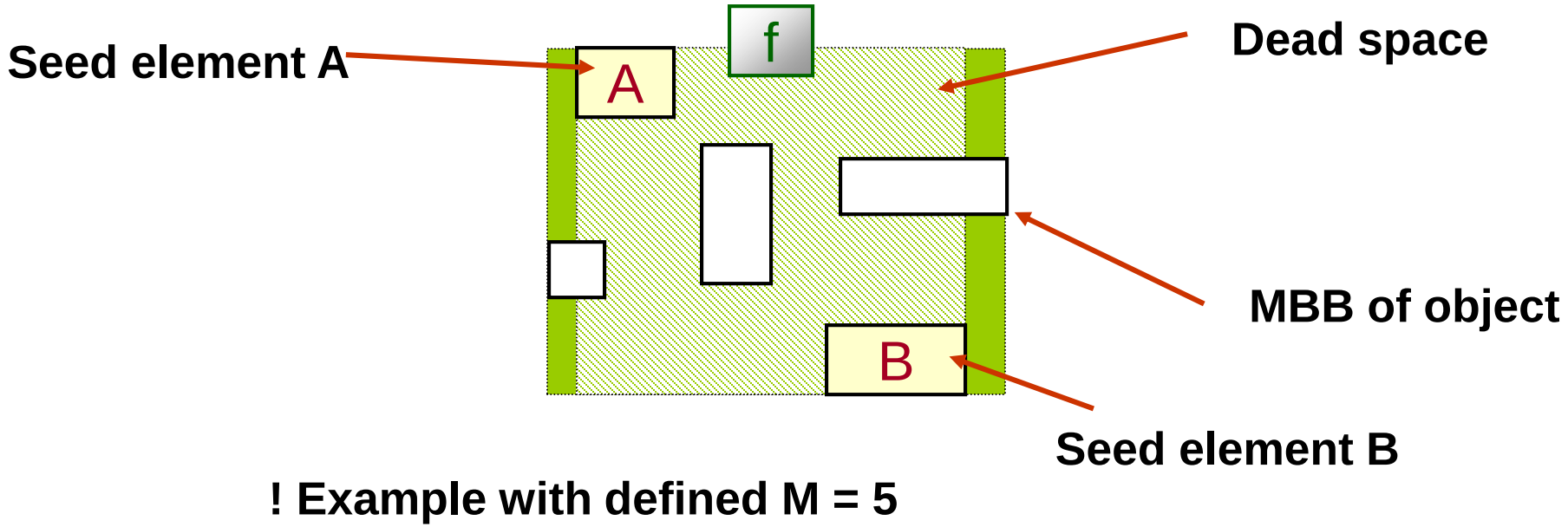
R-Tree indexes: Split by overflow of node

Search two seed elements for new nodes

- Compare pairwise all pairs of objects (N^2 complexity of alg.)
- Minimize total area of the two new nodes;
- Minimize overlap of the two new nodes

These conditions are in conflict - heuristic used:

- Maximise dead space between both seed elements, and iteratively expand to include additional elements, minimizing space/distance/number of elements.



! Example with defined M = 5

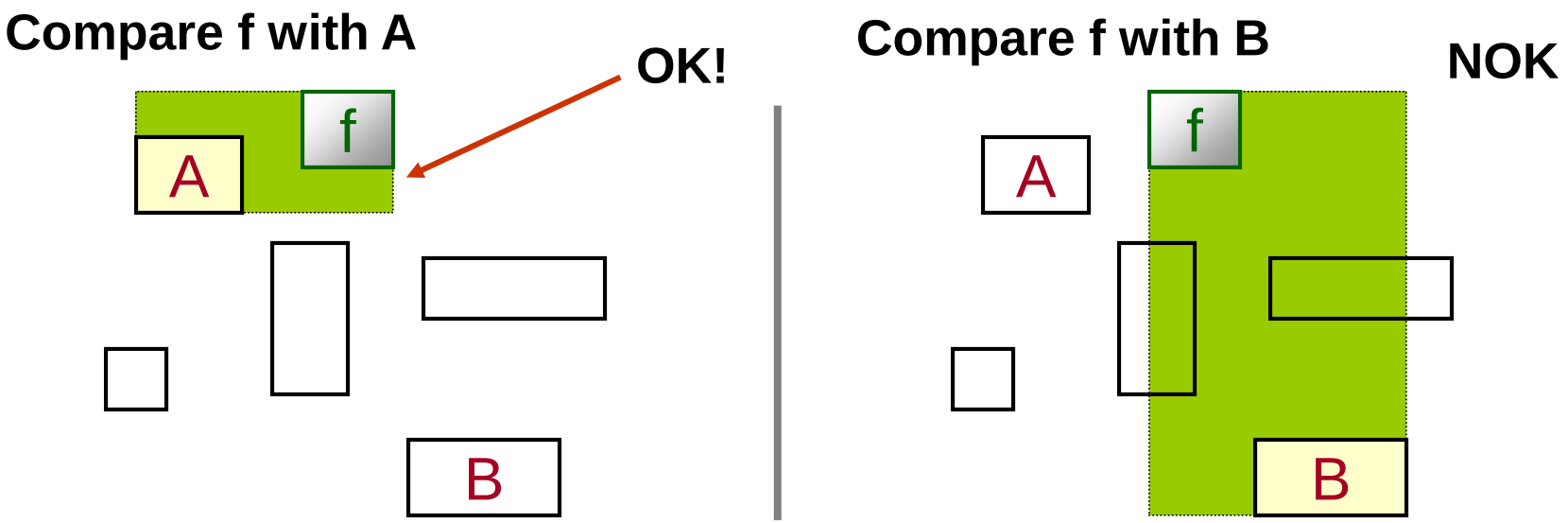
R-Tree indexes: Split by overflow of node

Search two seed elements for new nodes

- Compare pairwise all pairs of objects (N^2 complexity of alg.)
- Minimize total area of the two new nodes;
- Minimize overlap of the two new nodes

These conditions are in conflict - heuristic used:

- Maximise dead space between both seed elements, and iteratively expand to include additional elements, minimizing space/distance/number of elements.



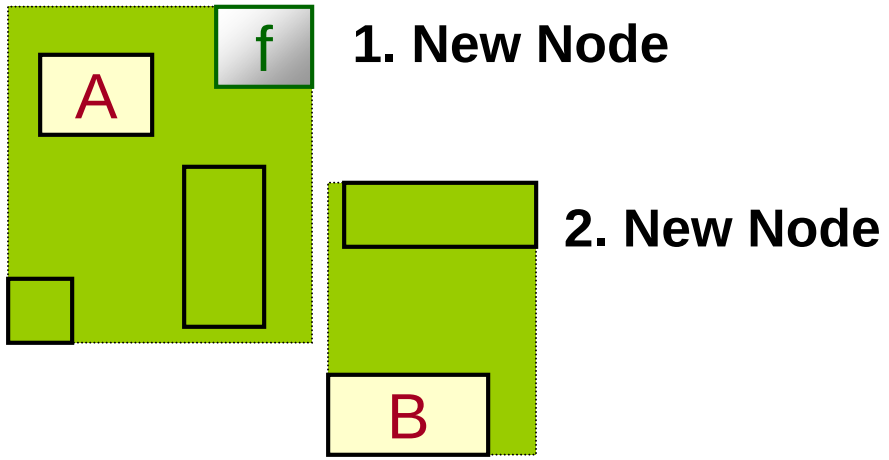
R-Tree indexes: Split by overflow of node

Search two seed elements for new nodes

- Compare pairwise all pairs of objects (N^2 complexity of alg.)
- Minimize total area of the two new nodes;
- Minimize overlap of the two new nodes

These conditions are in conflict - heuristic used:

- Maximise dead space between both seed elements, and iteratively expand to include additional elements, minimizing space/distance/number of elements.



R-Tree indexes: Deletion

Two situations

- Deletion of leave results in a node that is not underfilled
  - Leave is deleted
  - MBBs and references fixed in parent nodes, all the way to the root
- Deletion of leave results in underfilled parent node
  - Remaining leaves need to be re-inserted into the R-Tree;
  - Their references and MBBs are first deleted all the way to root, then re-computed (within the branch).

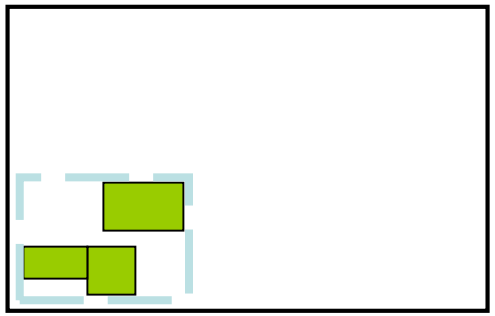
Underfilling → if the limit for m is not satisfied

m<x<M, where m>(M/2). Example: Oracle, max M = 34; this is stored in one page (memory and file based); storage amount 8K;

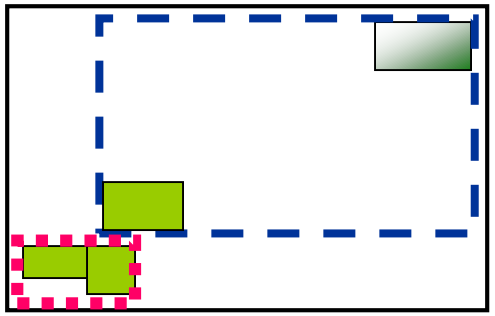
R-Tree indexes: Problematic cases

R-Trees can lead, after many I/Os to some problem cases, deteriorating the quality of the tree:

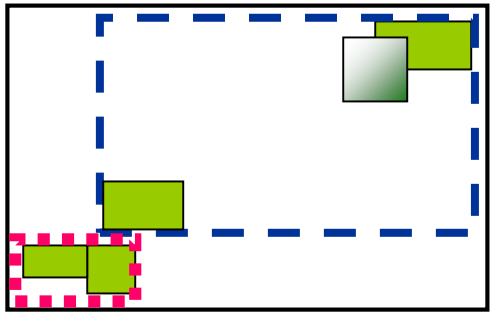
(1)



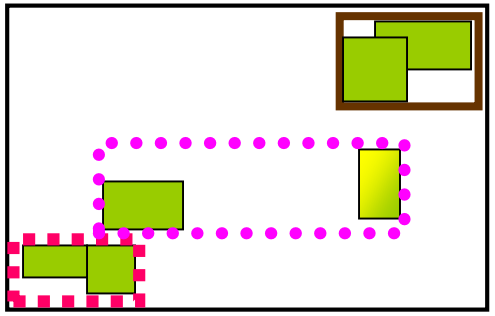
(2)



(3)



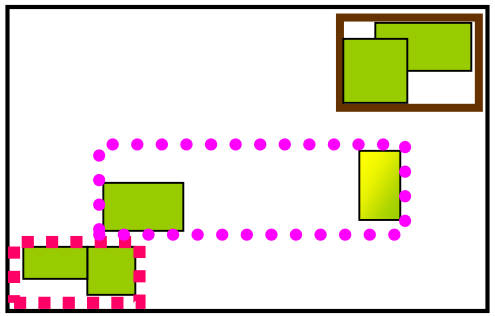
(4)



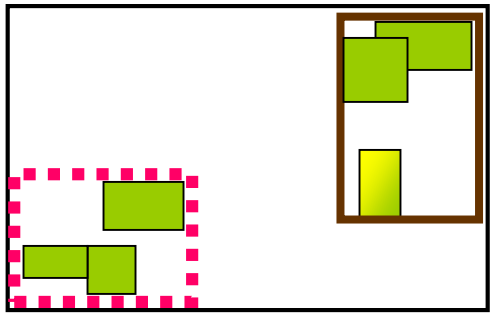
R-Tree indexes: Problem cases

Solution: recompute tree in its entirety:

(4)



(Opt)



```
-- Postgis: Analyze Table, then drop and recreate
VACUUM ANALYZE tmp_ped_path;
DROP INDEX tmp_ped_path_geom_sidx;
CREATE INDEX tmp_ped_path_geom_sidx ON tmp_ped_path
  USING GIST(geom);
```

```
-- REBUILD SPATIAL-INDEX in Oracle
ALTER INDEX USRDEMO.PARZPOLY_SHAPE_IDX REBUILD;
```

Bottom line – finding matching candidates

- Spatial index outperforms «no index» by at least of a factor of 10

```
EXPLAIN ANALYZE
SELECT s.id, s.geom
FROM   tmp_river_segment s  -- more than 170'000 recs
WHERE  ST_Intersects(s.geom,
ST_GeomFromText('LINESTRING(2683846 1250116, 2685846 1249812)',2056)
);
--> Scans on spatial index (=filter); does the detailed vertices math
(=refine) with 4 river segment records; results in 3 recs;

DROP INDEX tmp_river_segment_geom_sidx;
/* Rerun above query EXPLAIN ... */
--> Full scan on table; no index available; filter AND! refine with
170'000 records; 10-100 x slower;
```

Summary

- Indexing – sorting of data records, tree data structures
- Expansion of 1D ideas to 2D for spatial indexes:
  - Space driven:
  - Sort order
  - Grid, Quad Trees
- Object driven:
  - R-Tree as the generic case
- MBB-based:
  - hierarchical grouping
- Operations, storage & mapping to disk space
- Pros and Cons (for insert, deletion, for different distributions of spatial objects)