Exercise 9

Jiska Classen
Tobias Volk

H #9.1 CAN

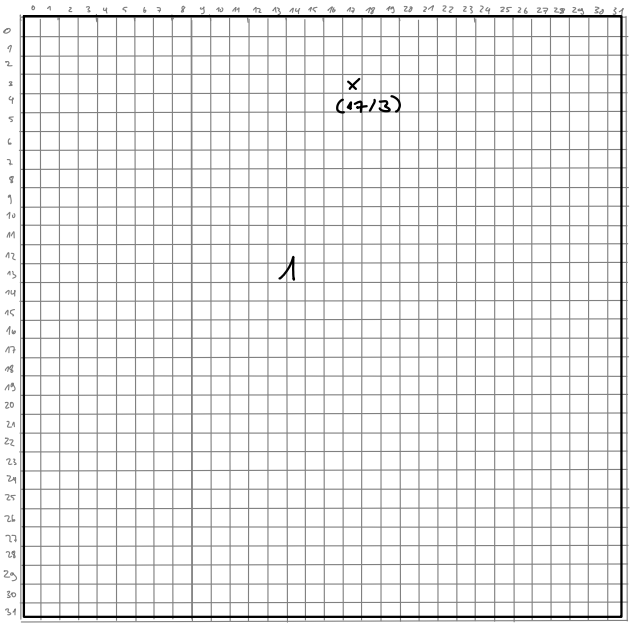
a)

[3, 49]

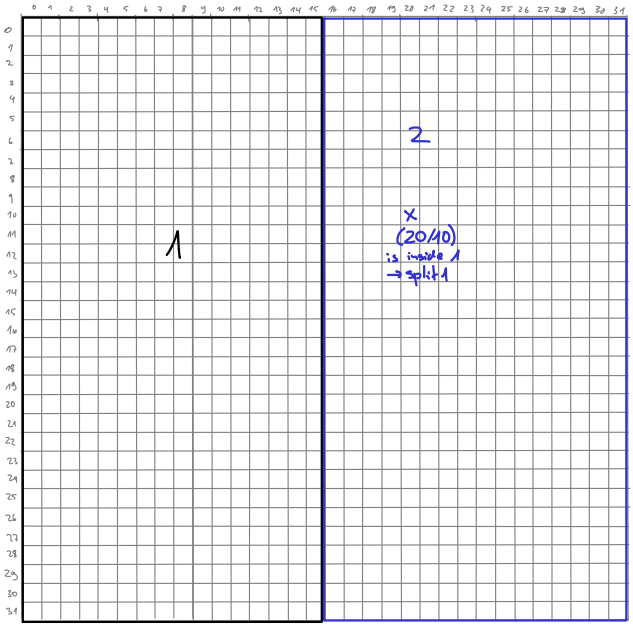
Scenario 1

32^2 identifier space

node 1 joins

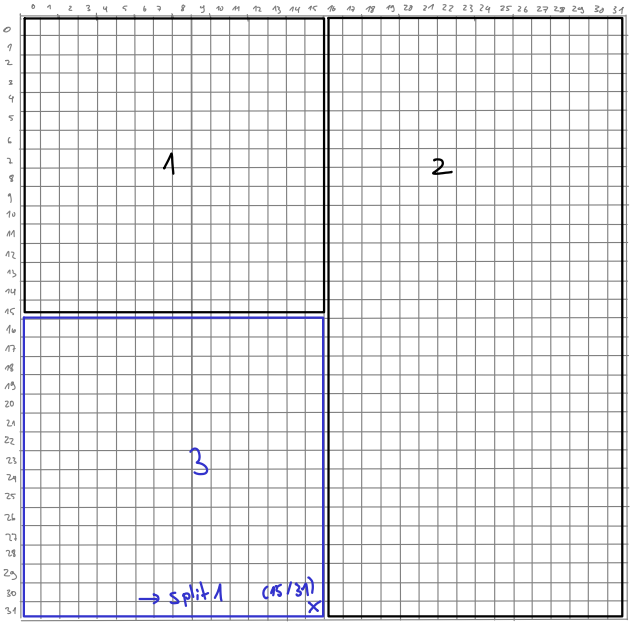


node 2 joins



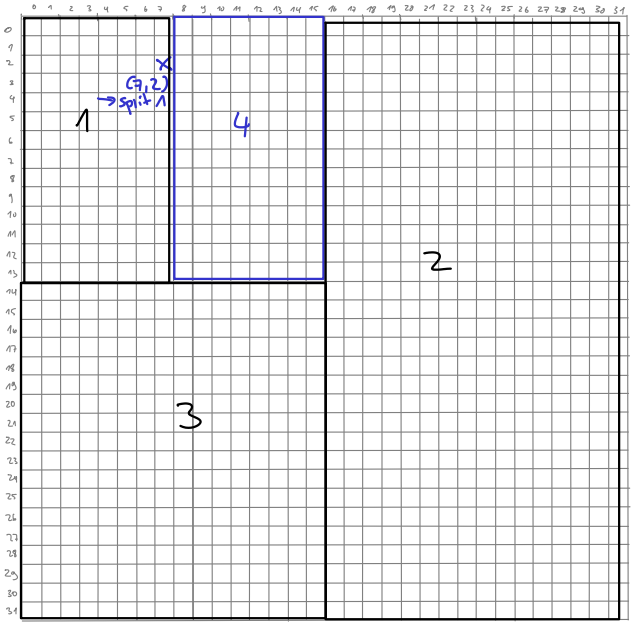
... receives right part of zone

node 3 joins

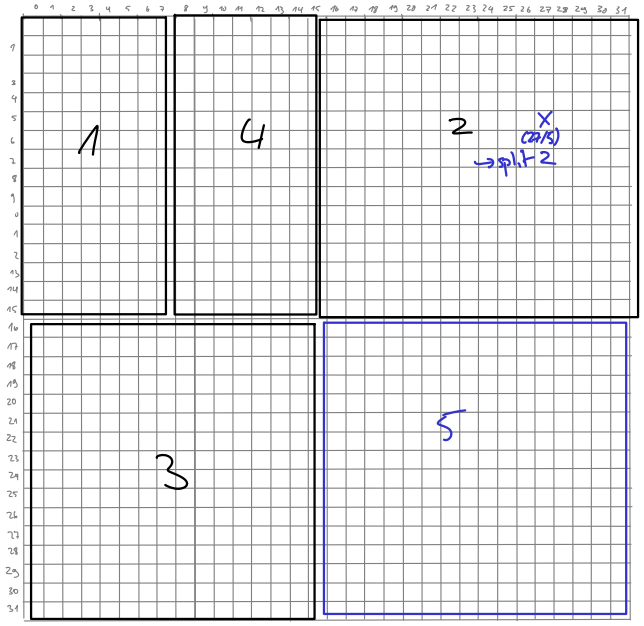


... receives lower part of zone

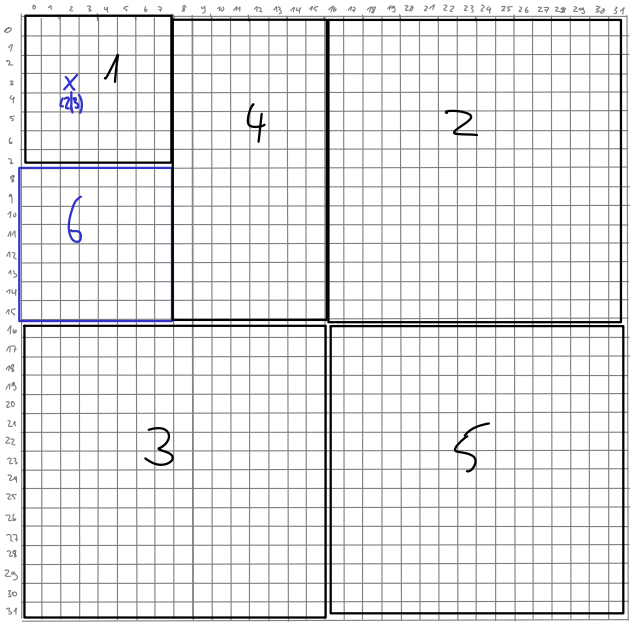
node 4 joins



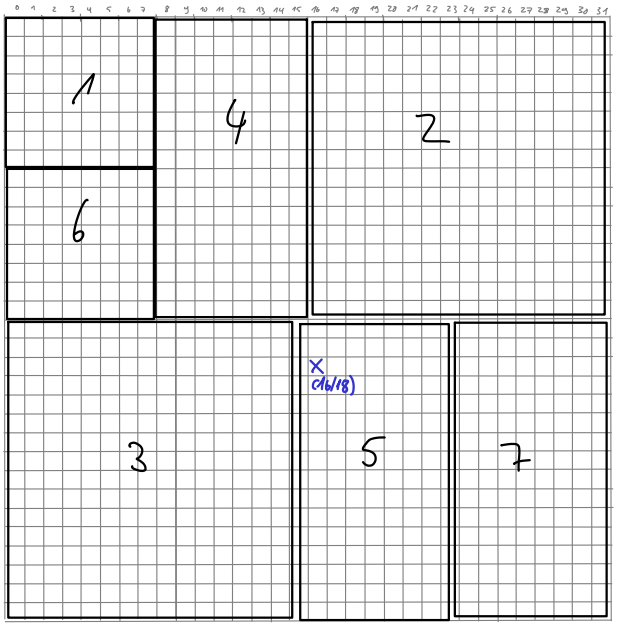
node 5 joins



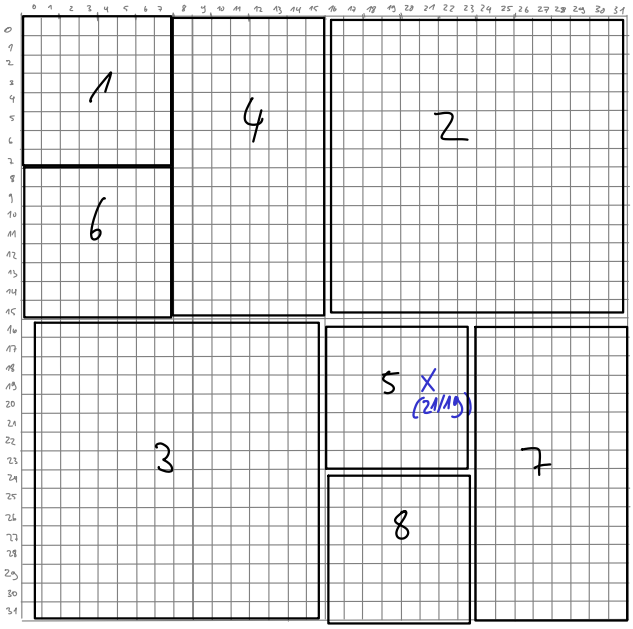
node 6 joins



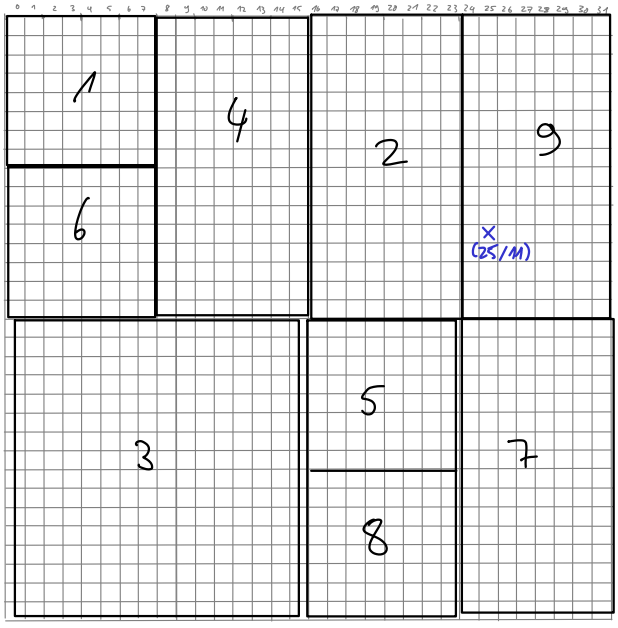
node 7 joins



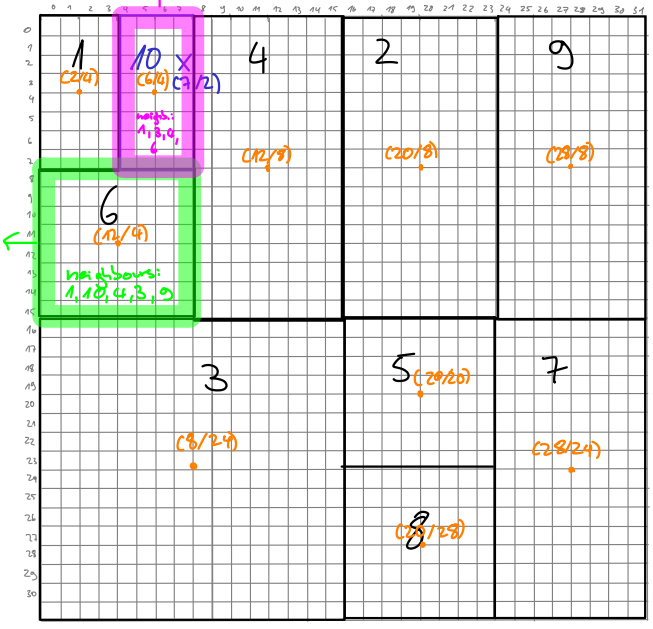
node 8 joins



node 9 joins



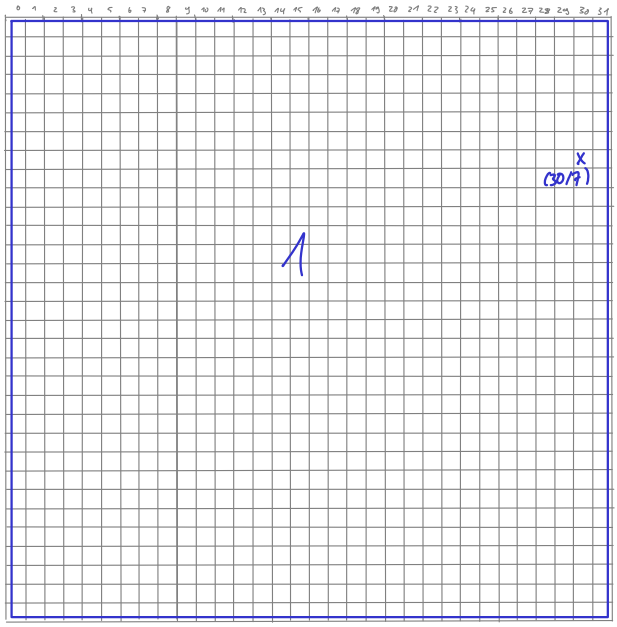
node 10 joins ⇒ result, new identifiers according to zone center



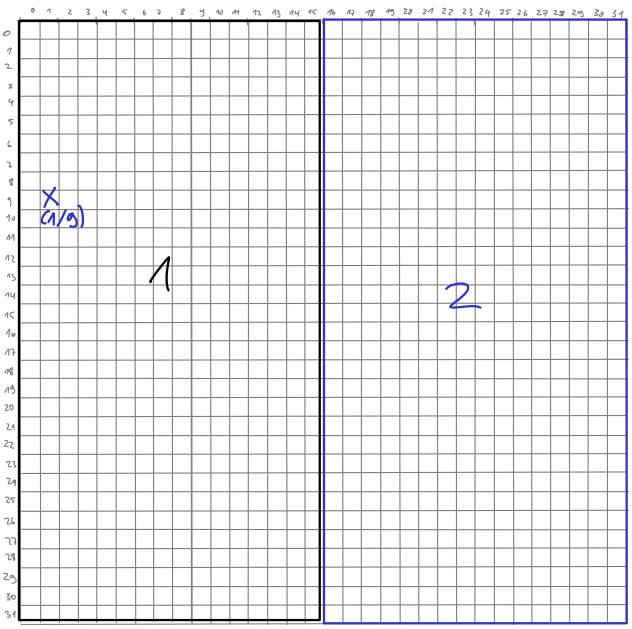
join of (7/2) again is possible since identifiers were reassigned to zone's center

Scenario 2

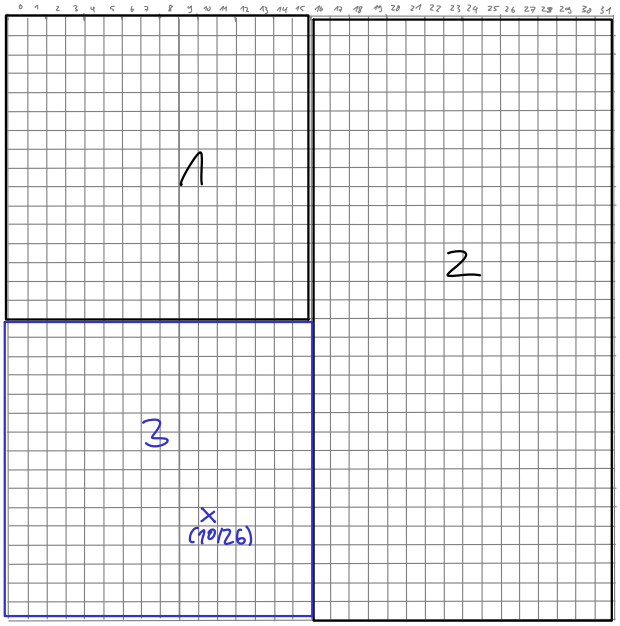
node 1 joins



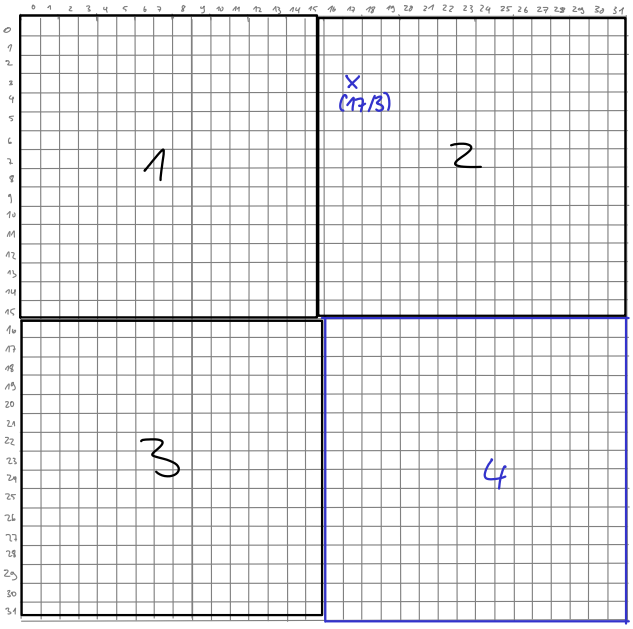
node 2 joins



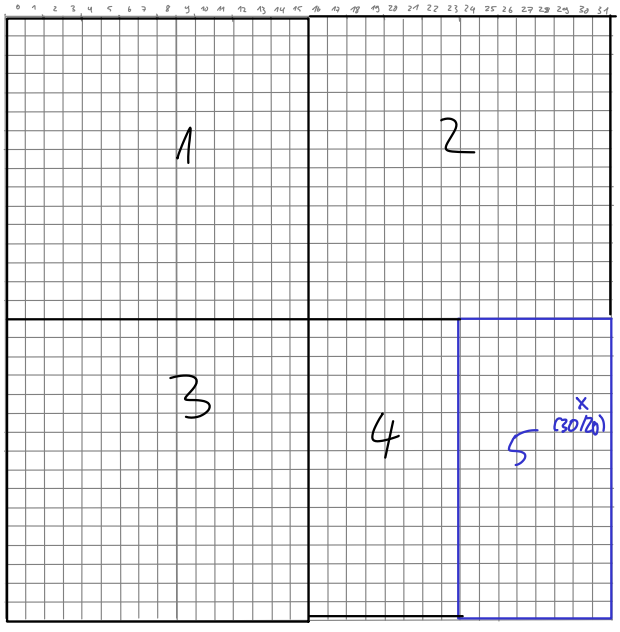
node 3 joins



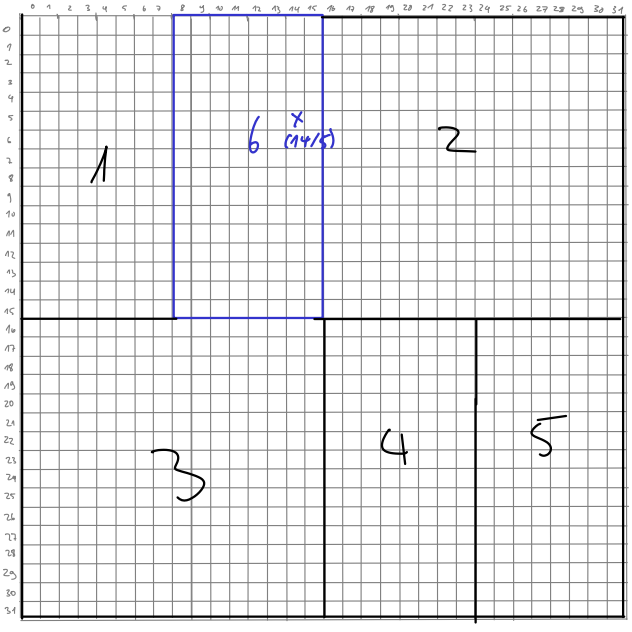
node 4 joins



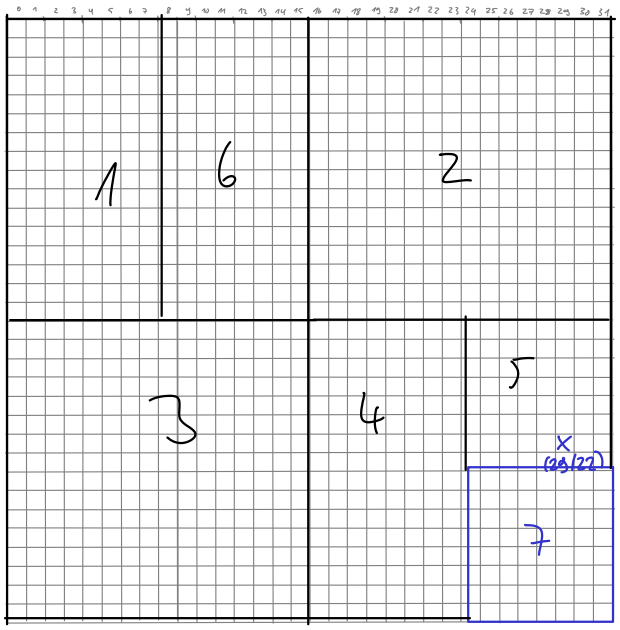
node 5 joins



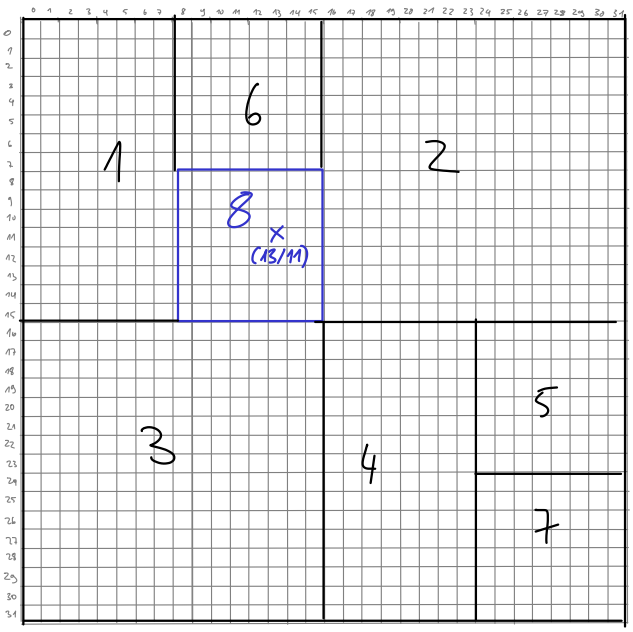
node 6 joins



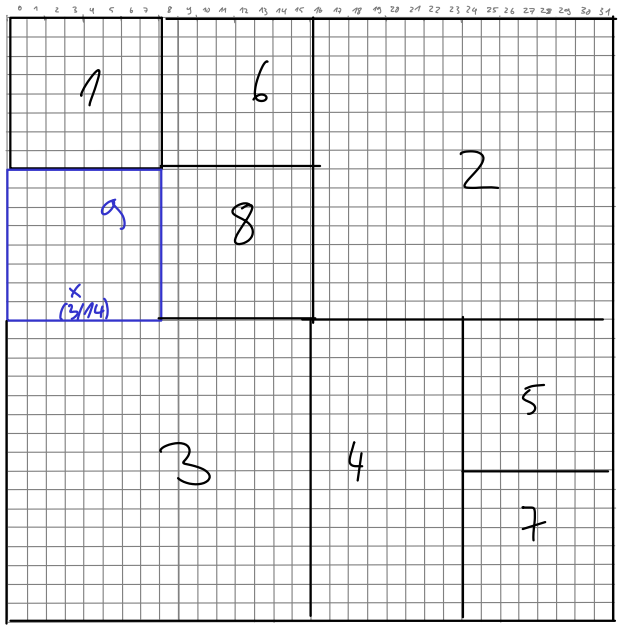
node 7 joins



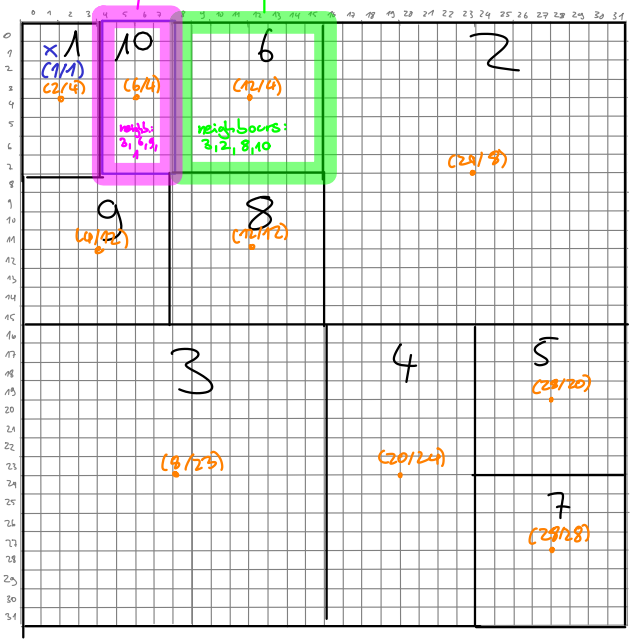
node 8 joins



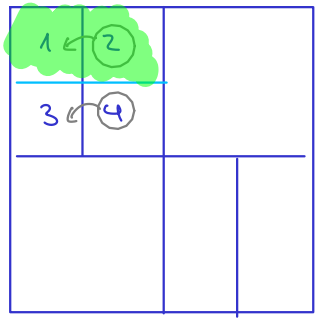
node 9 joins



node 10 joins → result

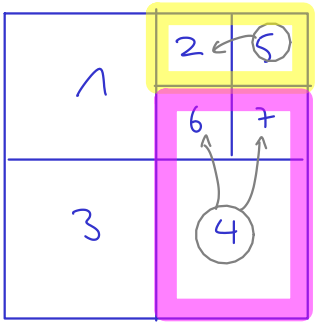


(a) normally it would be split like that:
... and in a next step:



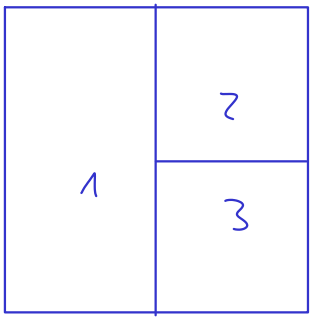
assuming node 1 is taken over by 2 and node 3 is taken over by 4, the shown partitions are possible.
(meaning: node 2 is now responsible for the green area which actually consists of 2 partitions)
also possible with additional joins/leaves.

(b) if buckets were split like that:



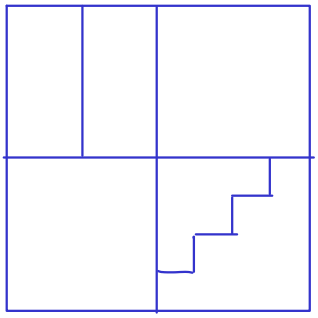
... and node 5 took over node 2,
and node 4 took over nodes 6 and 7, the resulting partitions are possible.
also possible with additional joins/leaves

(c) if buckets were split like that:



... it would be normal, this scenario can be created without any take over!

(d) is not possible! nodes are always split horizontal or vertical. only stk. like that would be possible (after many takeovers):



H# 9.2 Pastry / Tapestry

a) Routing Tables (Tapestry)

- neighbour tables (with largest node ID because we don't have ping times etc. to measure distance)

Staff.

- entries in level n:
  - have a common prefix of n-1 digits
  - one entry per digit is possible
  - the entry's digit is on position n
  - the rest of the digits is chosen so that the neighbour is as close as possible

node 1ef1

Level 1 empty here since nodes are on lower levels

Level \ Entry	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	0be4	—	2ed6	3cfa	4bcb	5f7d	6f65	7821	8d96	9dc0	Af38	Bc0f	Ce63	Dd09	E—	Fb51
2	10	11	12e4	13	1494	15	16	17	18	19	1A	1B	1C	1D	1E	1Fc0
3	1e0	1e1	1e2	1e3	1e4	1e5	1e6	1e7	1e8	1e9	1eA	1eB	1eC	1eD	1eE	1eF
4	1ef0	1ef1 node itself	1ef2	1ef3	1ef4	1ef5	1ef6	1ef7	1ef8	1ef9	1efA	1efB	1efC	1efD	1efE	1efF

empty, node doesn't exist

node 6f65

Level \ Entry	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	0bef	1ef1	2ed6	3cfa	4bcb	5f7d	6—	7821	8d96	9dc0	Af38	Bc0f	Ce63	Dd09	E—	Fb51
2	60	61	62	63	64	65	66	67	68	69	6A18	6B	6C	6D	6E	6F
3	6feb	6f1	6f2	6f3	6f4	6f5	6f6	6f7	6f8	6f9	6fA	6fB	6fC	6fD	6fE	6fF
4	6f60	6f61	6f62	6f63	6f64	6f65	6f66	6f67	6f68	6f69	6f6A	6f6B	6f6C	6f6D	6f6E	6f6F

6) Leaf nodes (Pastry)

Source of how Pastry routing and routing tables work: Paper "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems" by A. Rowstron and P. Druschel

Leaf set for $L = 4$ → 2 smaller and 2 larger nodes

- for node 1ef1
  - smaller: 12e4, 1494
  - larger: 1fc0, 21c4
- for node 6f65
  - smaller: 6a18, 6f06
  - larger: 7457, 74c0
- failing nodes are in a row

c) Routing (Pastry)

from 1ef1 to 6f06

- 1ef1 - compares 6f06 to leaf nodes → negative
  - compares 6f06 to its routing table (from level 4 to 1) and finds 6f65
  - routes message to 6f65
- 6f65 - compares 6f06 to leaf nodes → positive
  - routes message to 6f06
- 6f06 - processes the message

d) Back-up Nodes (Tapestry, Chord, KAD)

KAD doesn't implement backup nodes for routing table entries because it is querying multiple contacts in each step. It's possible to find a target in KAD even if a node on one possible route is down because there are still other contacts left.

After a failure of a node in Chord, the successor becomes responsible for that node's area. → The successor should be chosen as backup.
Finger entries pointing to the failure node become invalid. If a node is nearer (less hops) to the failing node, it's probability of having a finger to the failing node is higher.
⇒ the predecessor's predecessor should be chosen as 2nd backup.
(it is not the predecessor and successor as in exercise b) because the chord ring isn't bidirectional)