START – Self-Tuning Adaptive Radix Tree

Philipp Fent, Michael Jungmair, Andreas Kipf, Thomas Neumann

Technische Universität München

April 20, 2020

Practical Database Indexes

Learned indexes

- Small and efficient
- Adapt to data distribution
- Fast for read-only workloads

Practical Database Indexes

Learned indexes

- Small and efficient
- Adapt to data distribution
- Fast for read-only workloads

ART – Adaptive Radix Tree

- Used in practice
- Well understood
- Fast for various workloads
- But:
  limited adaption to data distribution
- and slower than a well-trained machine learning model

Practical Database Indexes

Learned indexes

- Small and efficient
- Adapt to data distribution
- Fast for read-only workloads

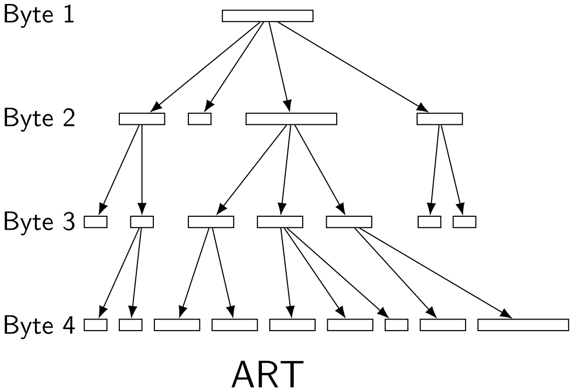
START
Self Tuning ART

- Bridges the gap
- Adapts to data: 85 % faster!
- Keep robustness

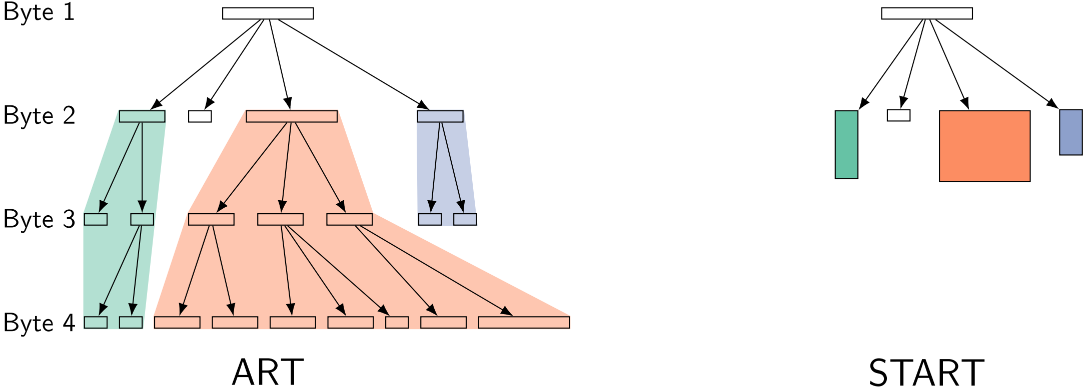
ART – Adaptive Radix Tree

- Used in practice
- Well understood
- Fast for various workloads
- But:
  limited adaption to data distribution
- and slower than a well-trained machine learning model

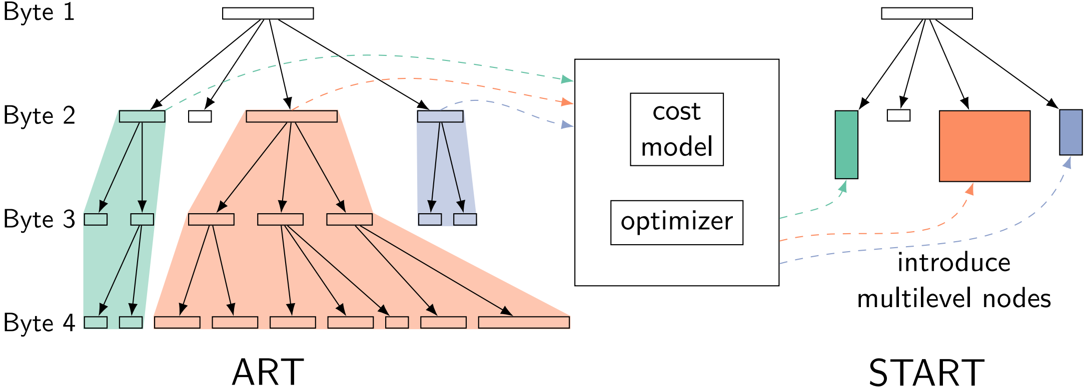
Self-Tuning ART



Self-Tuning ART



Self-Tuning ART



Multilevel Nodes

Rewired memory nodes

- Dense regions with many keys
- Subtrees of full nodes
- Keep data continuous

Multilevel Nodes

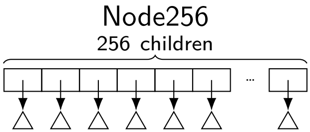
Rewired memory nodes

- Dense regions with many keys
- Subtrees of full nodes
- Keep data continuous

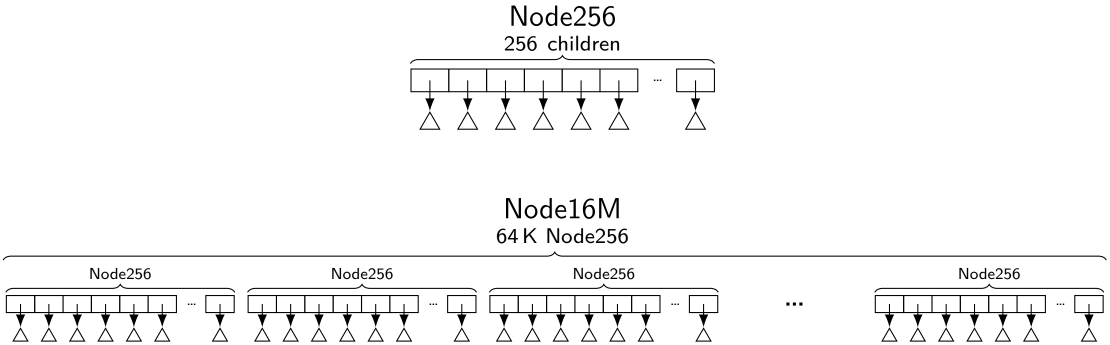
Sparse regions

- Already use path compression
- Still: improve common prefixes
- Reduce chains of small nodes

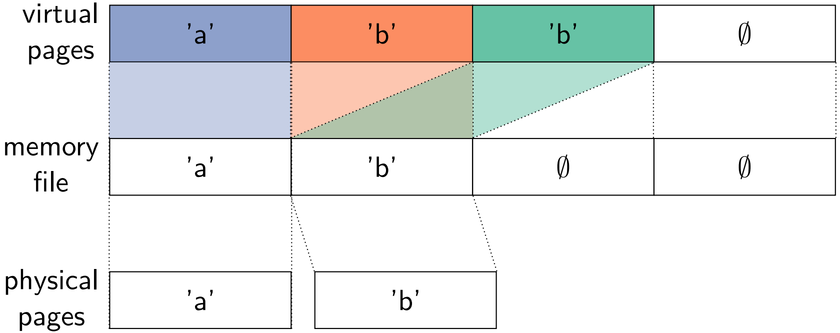
Rewired Memory Nodes



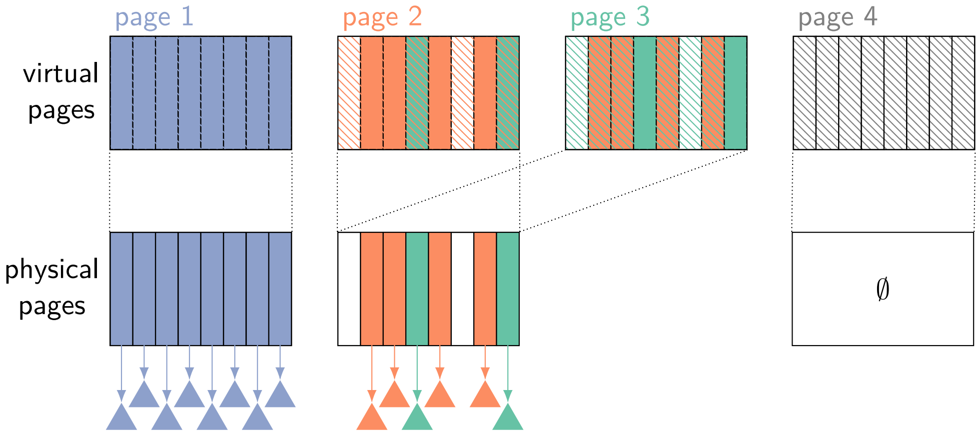
Rewired Memory Nodes



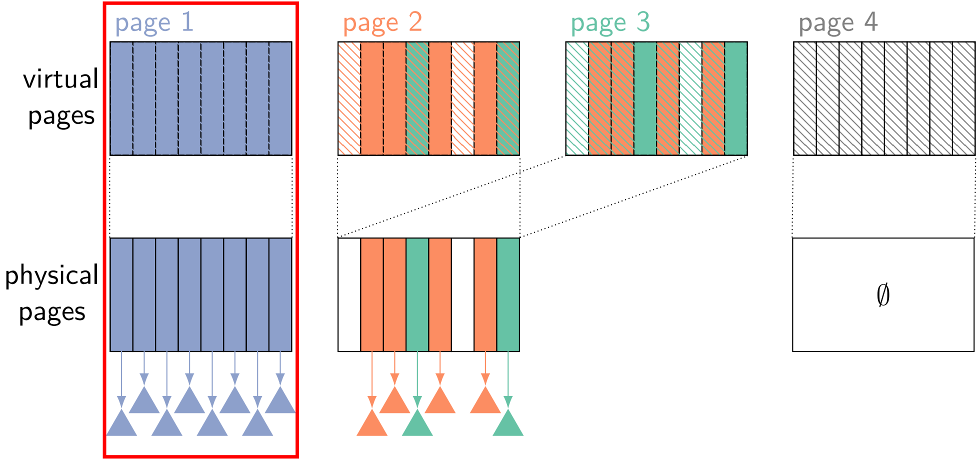
Rewired Memory Nodes



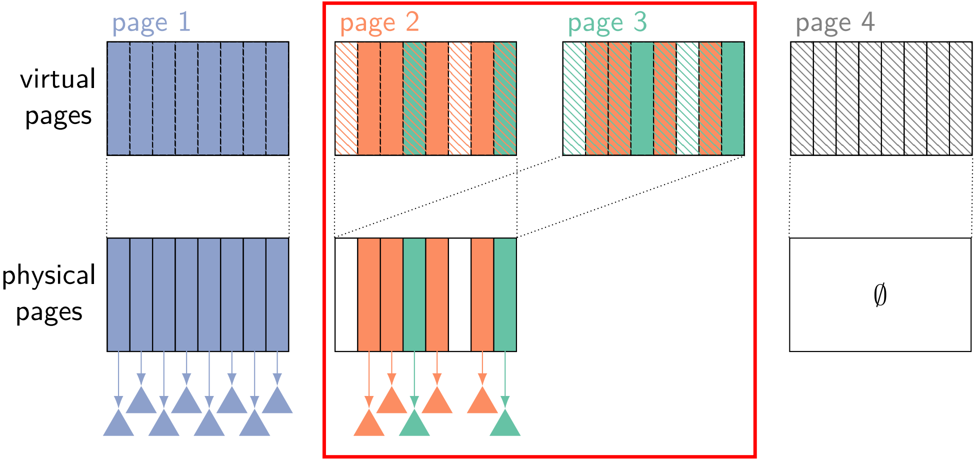
Rewired Memory Nodes



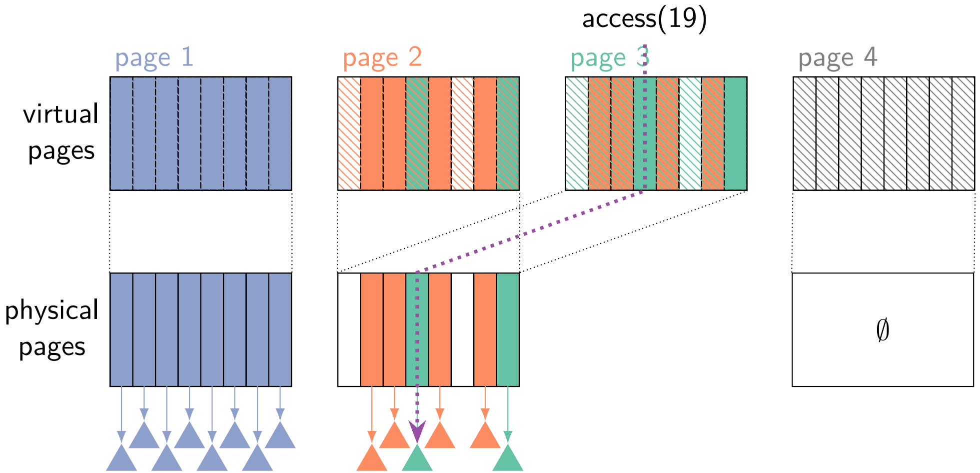
Rewired Memory Nodes



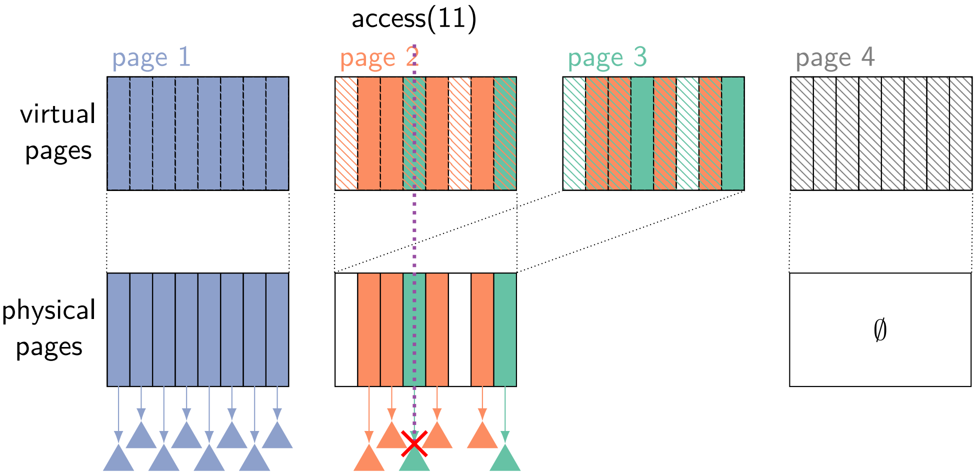
Rewired Memory Nodes



Rewired Memory Nodes

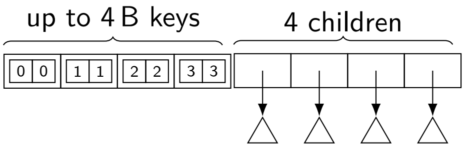


Rewired Memory Nodes



Multilevel Node4

- Helps reduce sparse affix nodes
- Use all 64 Bytes of a cacheline



Cost Model for Node Lookup

- Minimize the average child lookup cost
- Consider individual node cost

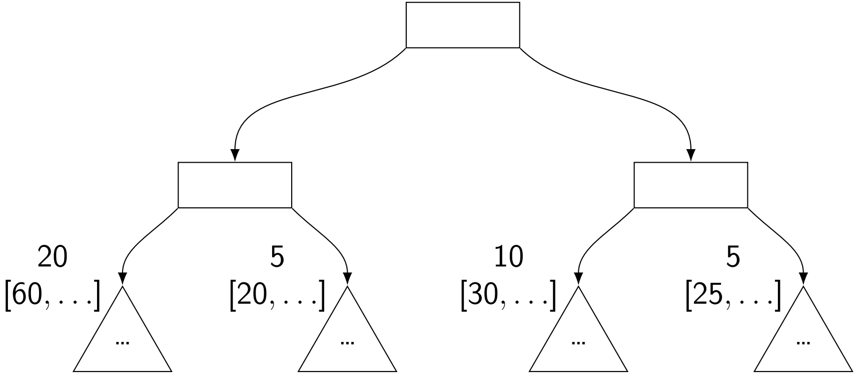
[ns/lookup]	Levels
Node4	1
Node16	1
Node48	1
Node256	1
Rewired64K	2
Rewired16M	3
MultiNode4	2-4

Cost Model for Node Lookup

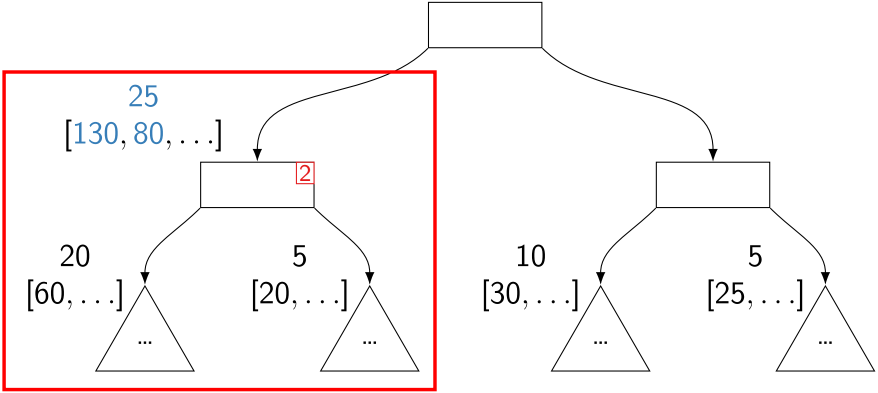
- Minimize the average child lookup cost
- Consider individual node cost

[ns/lookup]	Levels	Cached	Header Cached	Uncached
Node4	1	7	7	68
Node16	1	5	77	162
Node48	1	2	165	168
Node256	1	2	88	92
Rewired64K	2	6	87	162
Rewired16M	3	6	88	165
MultiNode4	2-4	6	6	68

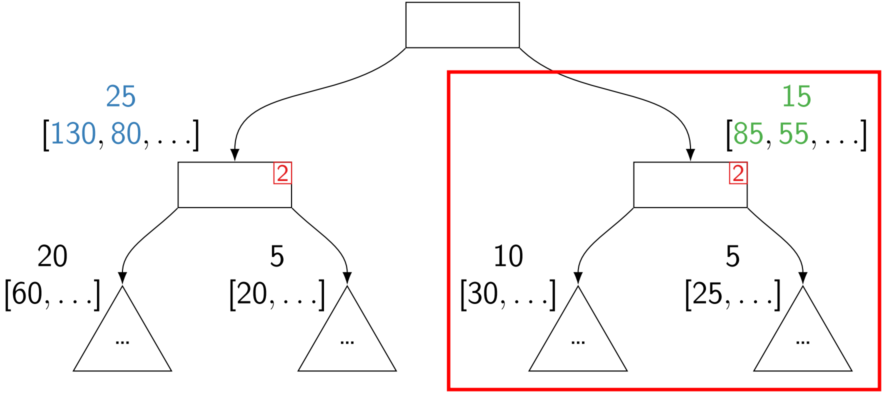
Bottom-up Optimization

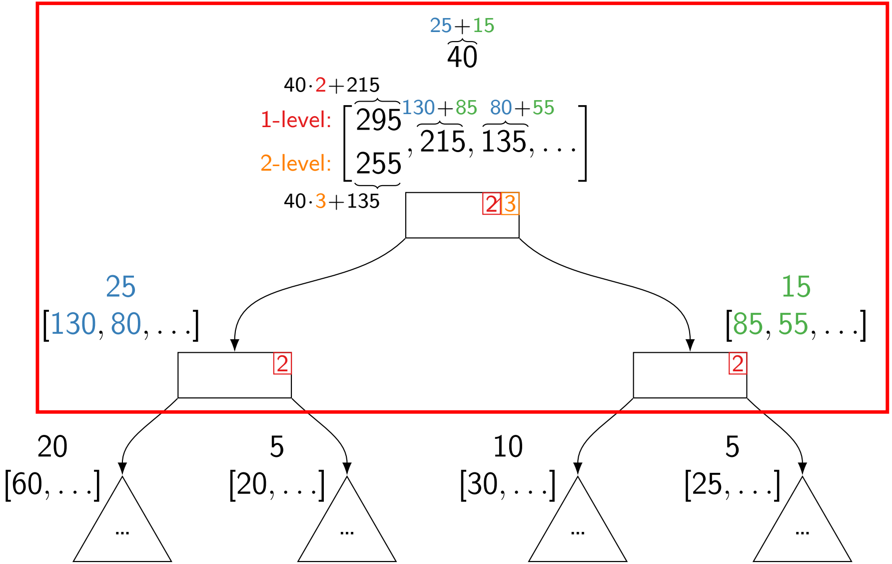


Bottom-up Optimization



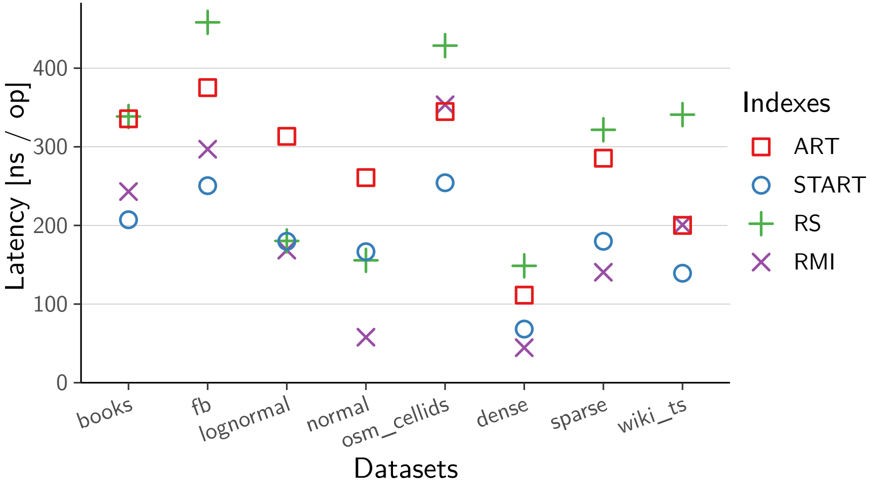
Bottom-up Optimization




25+15
40
40·2+215
1-level:
2-level:
295
255
130+85
80+55
215, 135, . . .
40·3+135
2
3
25
[130, 80, . . .]
15
[85, 55, . . .]
2
2
20
[60, . . .]
5
[20, . . .]
10
[30, . . .]
5
[25, . . .]
...
...
...
...

Performance

`https://learned.systems/sosd`



Conclusion

- START – Best of both worlds?
- Adapt to real-world data distribution
- Still with robust underpinning of ART
- Inserts still possible and efficient
- But: might degrade multilevel nodes

`github.com/jungmair/START`

`fent@in.tum.de`