B-tree Design

Data Structure for Ordered Collection

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>approximate access time (ns)</th>
<th>cost per megabyte ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>5-20</td>
<td>10-75</td>
</tr>
<tr>
<td>main memory</td>
<td>60-120</td>
<td>0.50-5.00</td>
</tr>
<tr>
<td>secondary storage</td>
<td>20,000,000</td>
<td>.001-0.10</td>
</tr>
</tbody>
</table>

Table 29.1 Approximate access times in nanoseconds (ns) as compared with the cost for cache, main memory, and secondary storage (disk).

Designed for situation in which \( n \) (number of elements or tags in a tagged collection) is so large that B-tree and \( n \) references to elements cannot fit in main memory.

Data is moved from disk to RAM in chunks called page.
Page Fault

Virtual memory lets you store data in secondary storage (act as if it's in main memory).

If program accesses data not in main memory, it's a page fault.
For a binary search tree, information we must store.

- tag (comparable) date of a historical event
- location on disk (secondary storage) for the rest of information associated with event
- parent ref
- left child ref
- right child ref
Want to group nodes of binary search tree into a single B-tree node that fills a disk page.

Good

not a good choice because we may only look at one node. Usually look at very few

1/6th of data left to consider
IN ORDER

Internally
B-Tree Node

# tags
location on disk for data

children

max sorted
2t-1 tags

ADGJMPR

1 2 3 4 5 6 7 8

0 2t^2 2t-1

# children is at most 2t