# CS280 Notes 4/4

*By Christian Sagel*

A balanced tree, rather than list nodes.
All the data structures we have used have been in-memory. But what if the data structure is too large to fit into memory We need to use external memory.
Very unlikely to have contiguous memory on a disk. We have pointers to sectors of the disk rather than addresses.
Programs designed for reading/writing data structures from/to external storage can bypass I/O overhead. Many functions used in RAM are too slow to use on drives.
All computations must be done in primary memory.

BTree
As the number of children per node increases, the height decreases.
The maximum number of children per node is called the *branching factor of the tree.*
BSTs have a branching factor of 2, 2-3-4 trees a factor of 4.

**Details**
Named by Bayer and McCreight who studied them in 1972. Used primarily for external searching. Based on multi-way trees. Each node in a B-Tree of order M has between M and M/2 children, except the root.
All nodes, except the root are at least half full.
All leaf nodes are at the same level. An internal node with N children has N - 1 keys.
The structure of a leaf node may be different than an internal node.
The nodes are page-based for optimal performance (determined by the hardware, hard drive sector/cluster size).

**Implementing B-Trees**
When storing the data in the node, a node may contain:

- The number of keys in the node.
- A sorted array of keys.
- A boolean flag indicating if the node is a leaf or not.

We want to store pointers to data rather than data in the nodes so we can have as much data as we can per node.
We take the size of PAGE, divided by the size of the data to get the number of maximum keys.

**Operations with B-Trees**
Operations have to account for disk access. Data may not be in memory, so must be retrieved from disk. Algorithms only run in memory/CPU. Typically some kind of manager handles the disk access. Disk access is transparent to the client (via an API). B-Trees facilitate key-range searches. (This is what databases are good at)

When we insert a key/data value pair, we insert into a sorted array (CPU complexity), similar to 2-3-4 trees; insert at the appropriate leaf. If the leaf is full, need to split: Middle key goes to parent, remaining keys get split into 2 nodes. Splitting the root causes the height to grow by 1.
When deleting a key/data pair, we often redistribute nodes, borrow children, merge into bigger nodes, etc.
Sometimes databases can just shut down for a few minutes and rebuild the tree rather than piecemeal.
Or we can mark nodes as deleted, and just not carry them over when we rebuild.

**B+Tree**
All data is stored outside of the nodes. Branching factor is maximized, used quite often in practice.
Allows for multiple sortings on the same data; Each B+Tree has the same pointers to the actual data. This is a huge win and keeps the data normalized. This could be done with binary search trees and other node-based containers.
Databases excel at these sorts by having different trees sorted on specific factors.
This is called normalizing the data. Having just one copy of the data and a bunch of pointers referencing it.

Database experience is very valuable for newer developers.
Relational calculus: A language tells you how to access data quickly. Structured query language.

NTFS, JFS, XFS, ReiserFS, Btrfs are implementing using B+Trees. Also Microsoft's SQL server and Oracle's database systems support B+Trees.