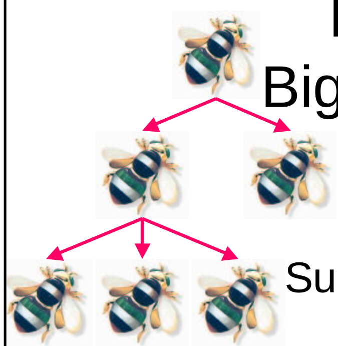
CSE 326: Data Structures
Lecture #11
Big, Bad B-Trees



Bart Niswonger
Summer Quarter 2001

Today's Outline

- Meeting times
- *m*-ary Trees
  - B-Trees
  - 2-3 Trees
  - 2-3-4 Trees

Announcements:

- Ashish's office hours
  Monday & Tuesday 4 - 4:50pm
- Bart's office hours
  Monday 12 – 1 pm & open door

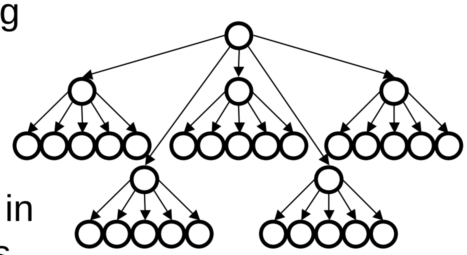
Meeting Times

- *Monday*
  - 1:00pm to 1:30pm   Ben & Rob
  - 1:30pm to 2:00pm   Grace & Margaux
  - 2:00pm to 2:30pm   Yukiyo
  - 2:30pm to 3:00pm   Rishi & Eric
  - 3:30pm to 4:00pm   Chris
  - 4:00pm to 4:30pm   Justice
- *Tuesday*
  - 12:00pm to 12:30pm   Ryan
  - 12:30pm to 1:00pm   Takako
- *Thursday*
  - 12:00pm to 12:30pm   Renata & Alex

m-ary Search Tree

- Maximum branching factor of ***M***
- Complete tree has depth = $\log_M N$
- Each internal node in a complete tree has ***M* - 1** keys

runtime:

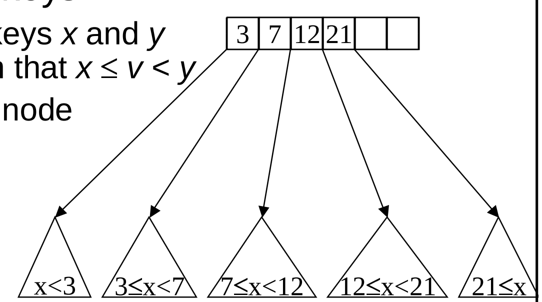


m-ary Search Trees: why?

- Related to *d*-heaps

B-Trees

- B-Trees are specialized *m*-ary search trees
- Each node has many keys
  - subtree between two keys x and y contains values v such that $x \le v < y$
  - binary search within a node to find correct subtree
- Each node takes one full {*page, block, line*} of memory



B-Tree Properties‡

- Properties
  - **maximum branching factor of M**
  - **the root has between 2 and M children *or* at most L keys**
  - other internal nodes have between $\lceil M/2 \rceil$ and M children
  - internal nodes contain only search keys (no data)
  - smallest datum between search keys x and y equals x
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and L keys
  - all leaves are at the same depth
- Result
  - tree is $\Theta(\log n)$ deep ($\sim \log_{M/2} n$)
  - all operations run in $\Theta(\log n)$ time
  - operations pull in about M or L items at a time

‡These are technically B^+-Trees

B-Tree Properties

- Properties
  - maximum branching factor of M
  - the root has between 2 and M children *or* at most L keys
  - **other internal nodes have between $\lceil M/2 \rceil$ and M children**
  - **internal nodes contain only search keys (no data)**
  - **smallest datum between search keys x and y equals x**
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and L keys
  - all leaves are at the same depth
- Result
  - tree is $\Theta(\log n)$ deep ($\sim \log_{M/2} n$)
  - all operations run in $\Theta(\log n)$ time
  - operations pull in about M or L items at a time

B-Tree Properties

- Properties
  - maximum branching factor of *M*
  - the root has between 2 and *M* children *or* at most *L* keys
  - other internal nodes have between $\lceil M/2 \rceil$ and *M* children
  - internal nodes contain only search keys (no data)
  - smallest datum between search keys *x* and *y* equals *x*
  - **each (non-root) leaf contains between $\lceil L/2 \rceil$ and *L* keys**
  - **all leaves are at the same depth**
- Result
  - tree is Θ(log n) deep (~ $\log_{M/2} n$)
  - all operations run in Θ(log n) time
  - operations pull in about *M* or *L* items at a time

B-Tree Properties

- Properties
  - maximum branching factor of *M*
  - the root has between 2 and *M* children *or* at most *L* keys
  - other internal nodes have between $\lceil M/2 \rceil$ and *M* children
  - internal nodes contain only search keys (no data)
  - smallest datum between search keys *x* and *y* equals *x*
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and *L* keys
  - all leaves are at the same depth
- Result
  - **tree is Θ(log n) deep (~ $\log_{M/2} n$)**
  - **all operations run in Θ(log n) time**
  - **operations pull in about *M* or *L* items at a time**

When Big-O is Not Enough

B-Tree is about $\log_{M/2} n/(L/2)$ deep

$= \log_{M/2} n - \log_{M/2} L/2$

$= O(\log_{M/2} n)$

$= O(\log n)$ per operation (same as BST!)

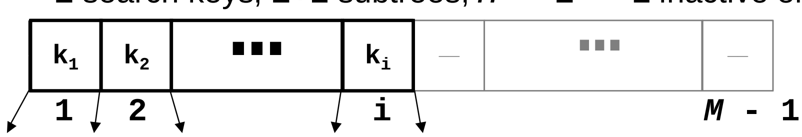
Where's the beef?!

$\log_2(\ 10{,}000{,}000\ ) = 24$ disk accesses

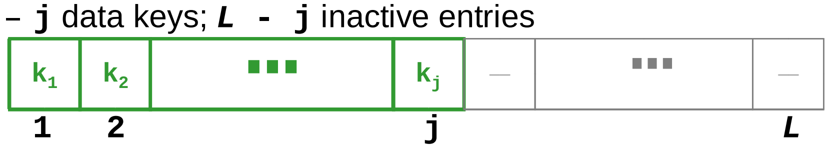
$\log_{200/2}(\ 10{,}000{,}000\ ) < 4$ disk accesses

B-Tree Nodes

- *Internal node*
  - **`i`** search keys; **`i+1`** subtrees; ***`M - i - 1`*** inactive entries

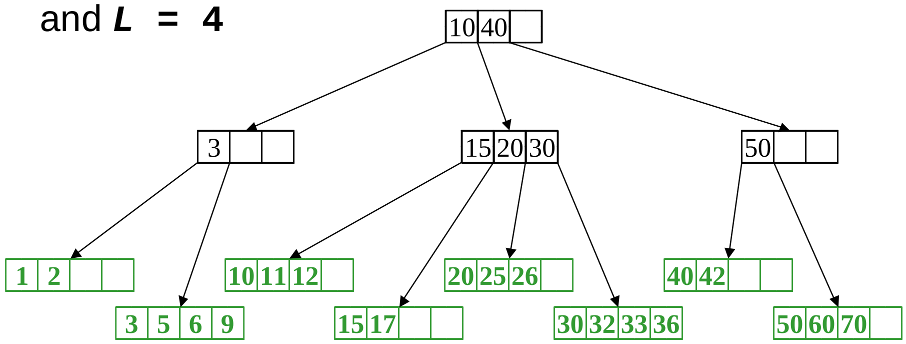


- *Leaf*
  - **`j`** data keys; ***`L - j`*** inactive entries

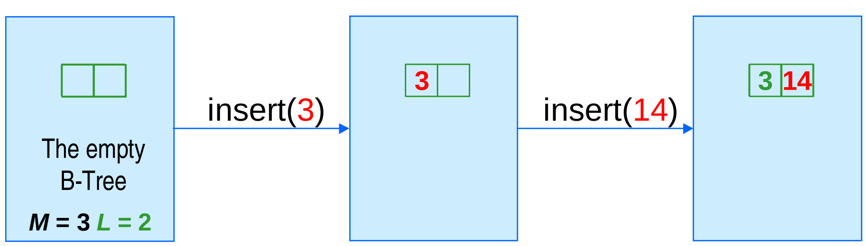


Example

B-Tree with $M = 4$
and $L = 4$

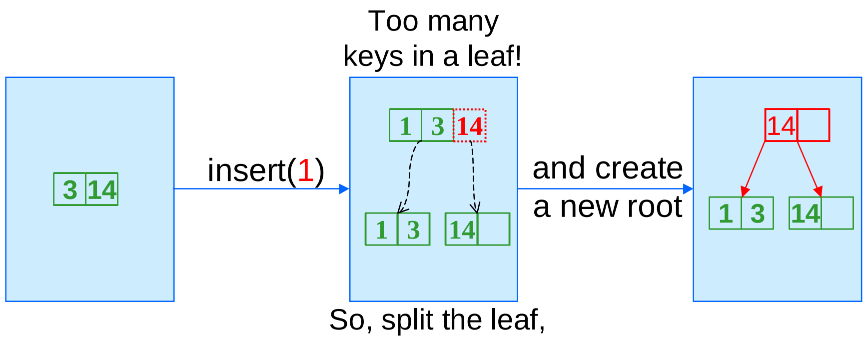


Making a B-Tree

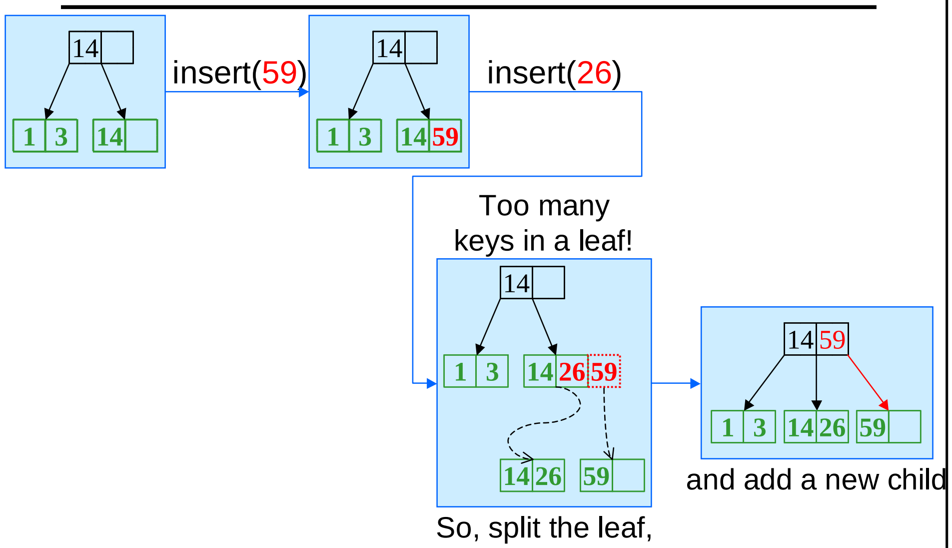


Now, insert(1)?

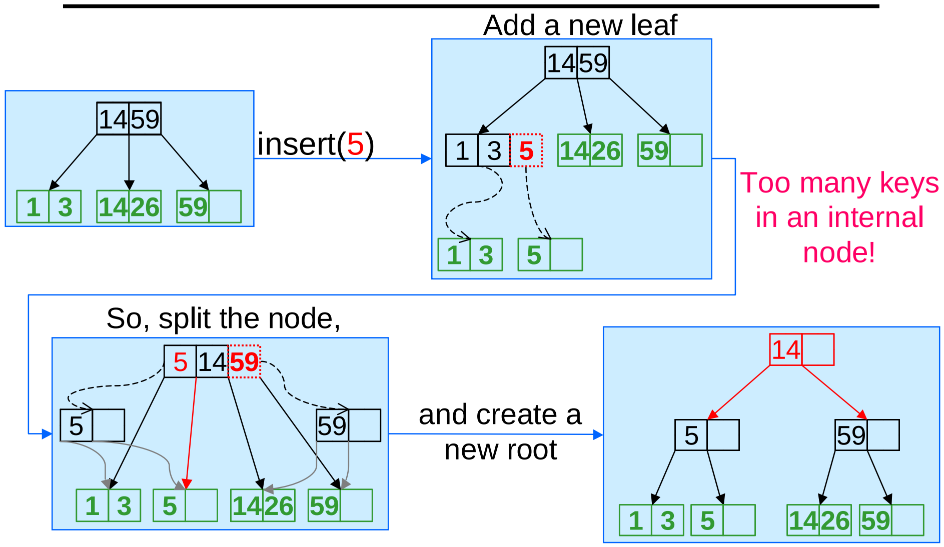
Splitting the Root



Insertions and Split Ends



Propagating Splits

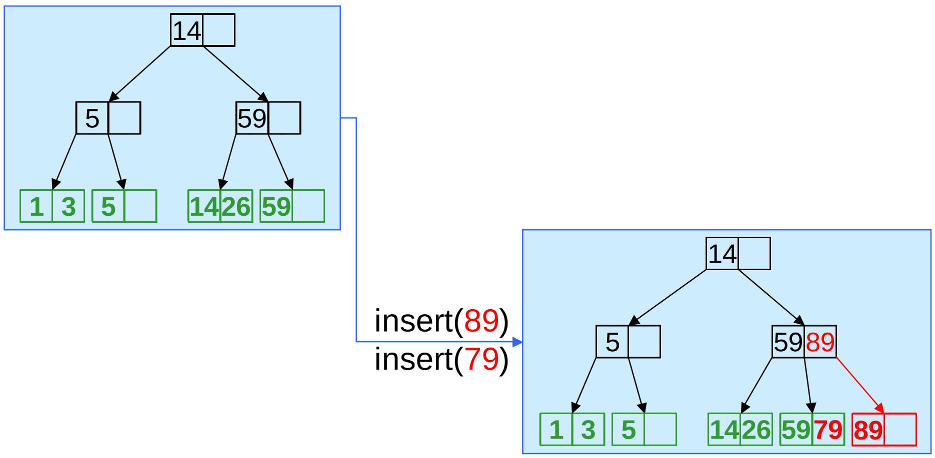


Insertion in Boring Text

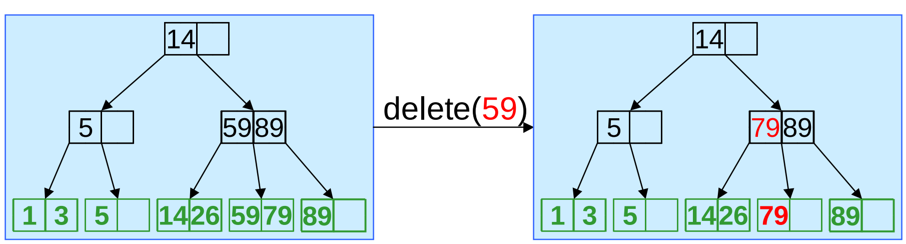
- Insert the key in its leaf
- If the leaf ends up with L+1 items, **overflow**!
  - Split the leaf into two nodes:
    - original with $\lceil (L+1)/2 \rceil$ items
    - new one with $\lfloor (L+1)/2 \rfloor$ items
  - Add the new child to the parent
  - If the parent ends up with $M+1$ items, **overflow**!
- If an internal node ends up with M+1 items, **overflow**!
  - Split the node into two nodes:
    - original with $\lceil (M+1)/2 \rceil$ items
    - new one with $\lfloor (M+1)/2 \rfloor$ items
  - Add the new child to the parent
  - If the parent ends up with $M+1$ items, **overflow**!
- Split an overflowed root in two and hang the new nodes under a new root

This makes the tree deeper!

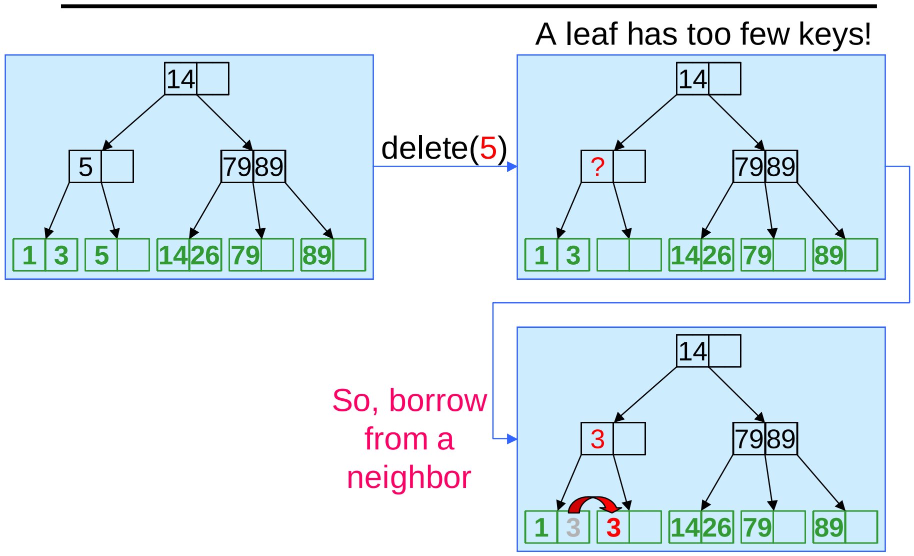
After More Routine Inserts



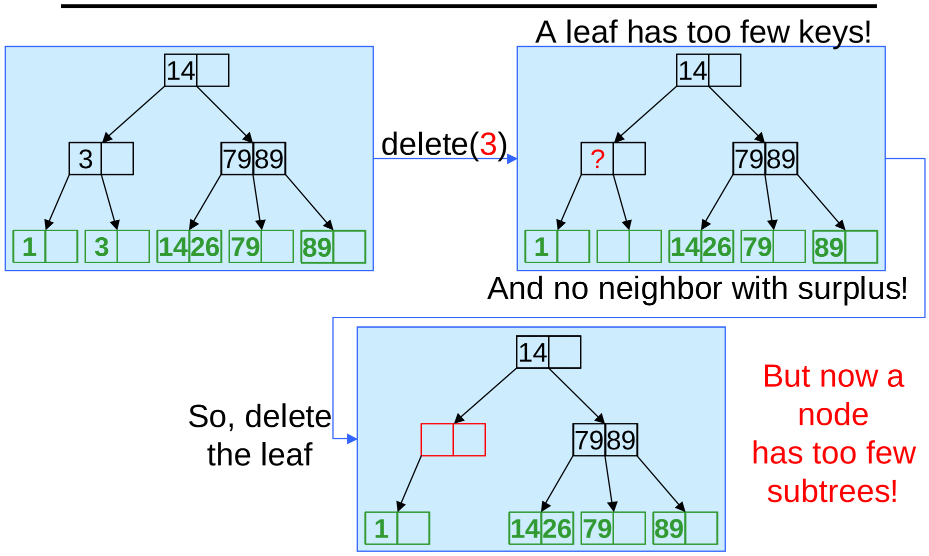
Deletion



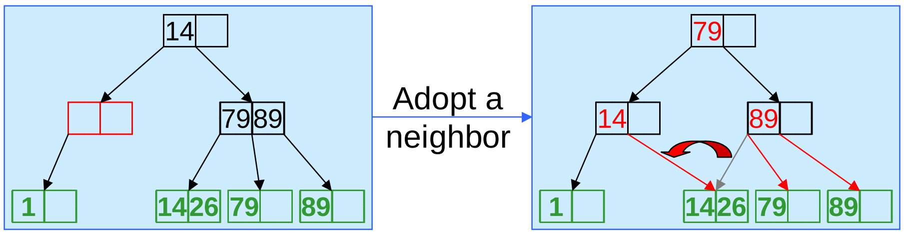
Deletion and Adoption



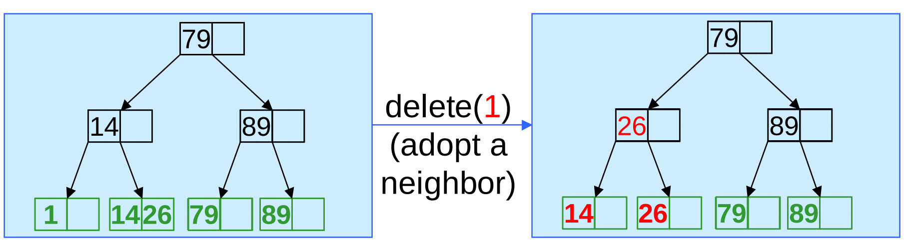
Deletion with Propagation



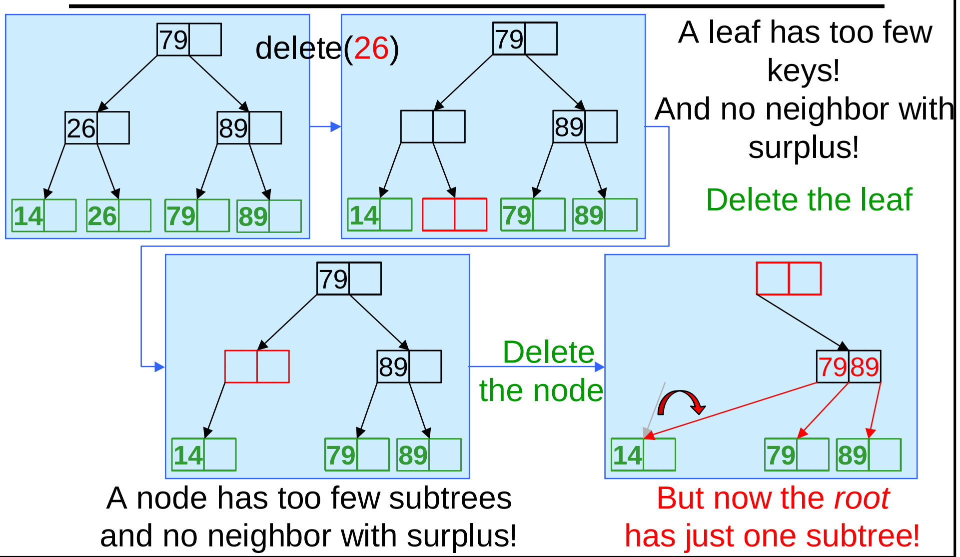
Finishing the Propagation



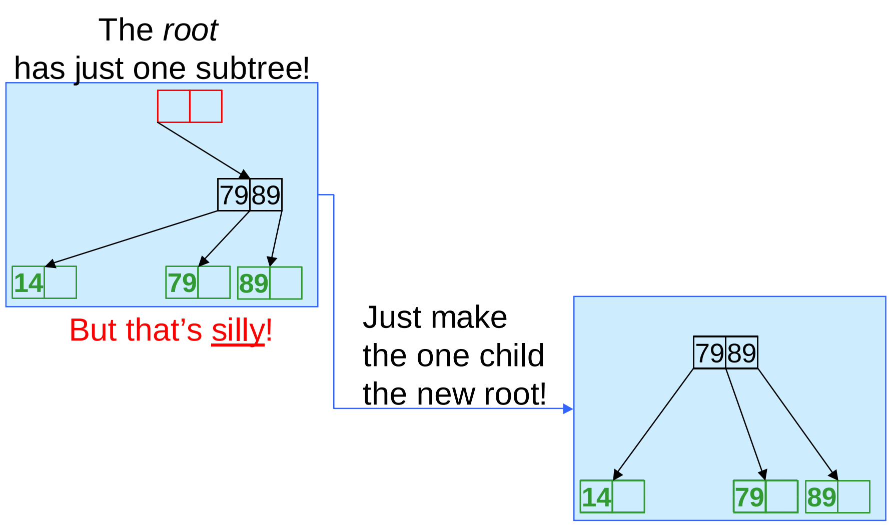
A Bit More Adoption



Pulling out the Root



Pulling out the Root (continued)



Deletion (in *Two* Boring Slides of Text)

- Remove the key from its leaf
- If the leaf ends up with fewer than $\lceil L/2 \rceil$ items, **underflow**!
  - Adopt data from a neighbor; update the parent
  - If borrowing won't work, delete node and divide keys between neighbors
  - If the parent ends up with fewer than $\lceil M/2 \rceil$ items, **underflow**!

Why will dumping keys always work if borrowing doesn't?

Deletion (Slide Two)

- If a node ends up with fewer than $\lceil M/2 \rceil$ items, **underflow**!
  - Adopt subtrees from a neighbor; update the parent
  - If borrowing won't work, delete node and divide subtrees between neighbors
  - If the parent ends up with fewer than $\lceil M/2 \rceil$ items, **underflow**!
- If the root ends up with only one child, make the child the new root of the tree

This reduces the height of the tree!

Thinking about B-Trees

- B-Tree insertion can cause (expensive) splitting and propagation
- B-Tree deletion can cause (cheap) borrowing or (expensive) deletion and propagation
- Propagation is rare if ***M*** and ***L*** are large *(Why?)*
- Repeated insertions and deletion can cause thrashing
- If ***M* = *L* = 128**, then a B-Tree of height 4 will store at least 30,000,000 items
- Hard to implement!
- VERY common – the most common tree type?

A Tree with Any Other Name

FYI:

- B-Trees with ***M* = 3**, ***L* = x** are called
  2-3 trees
- B-Trees with ***M* = 4**, ***L* = x** are called
  2-3-4 trees

Why would we ever use these?

To Do

- Continue Project II
- Continue Homework 4
- Look forward to no quiz/homework for a week!
- (And prepare for the midterm)

Coming Up

- Midterm (Wednesday)
- Project II due (July 23rd)
- Homework 4 due (July 23rd)

To Do

- Study for midterm!
- Read through section 4.7 in the book
- Comments & Feedback
- Homework IV (studying)
- Project II – part B

Coming Up

- Midterm next Wednesday
- A **Huge** Search Tree Data Structure (not on the midterm)