CS186 Discussion Section Week 3

Tree-structured Indexing

Today

- Reminder: Why Use Indexes?
- Tree-based Indexes
  - ISAM
  - B+-Trees
- Classifying Indexes
- Worksheet

Why Index Anything? (and Why Not Index Everything?)

The good:

- Flexible lookups (equality, range, regex, composite)
- REALLY fast lookups (Log base F)
- Can build multiple indexes without duplicating data

The bad:

- Maintenance cost for updates
- Storage overhead

Today: Tree-based Indexes

Let’s Index GSIs! (pre-sorted by name)

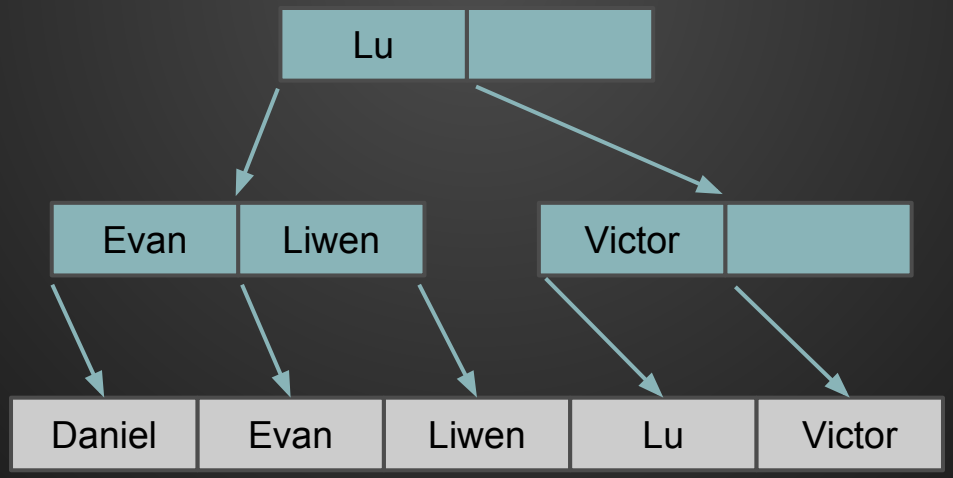
ISAM: Construction

- Static tree structure – “old”
- Make an index on a bunch of data by
  1. Sort records by index search key (e.g., “name”)
  2. Build a tree on top of them!

ISAM: Construction

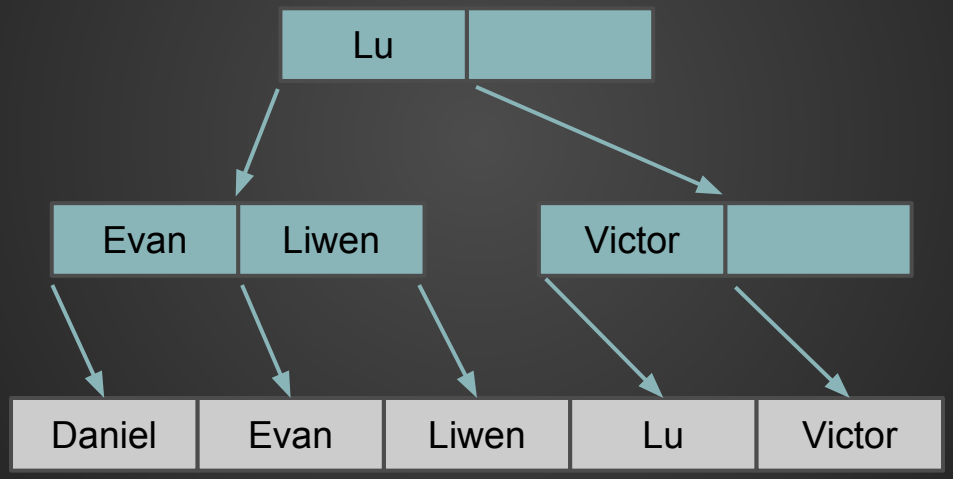
Each index block is 1 page.

We can fit 2 index entries per page, but only 1 data entry (bottom level)



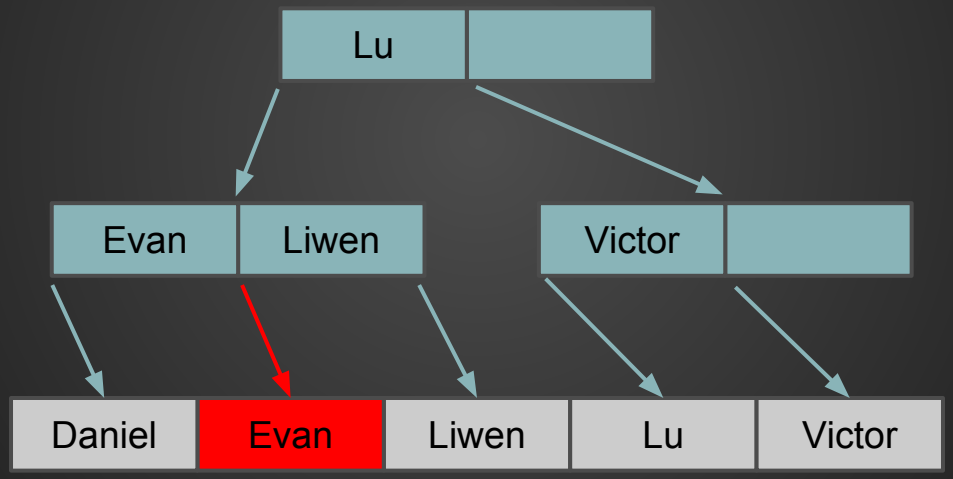
ISAM: Insertion

- Insert “Henry”



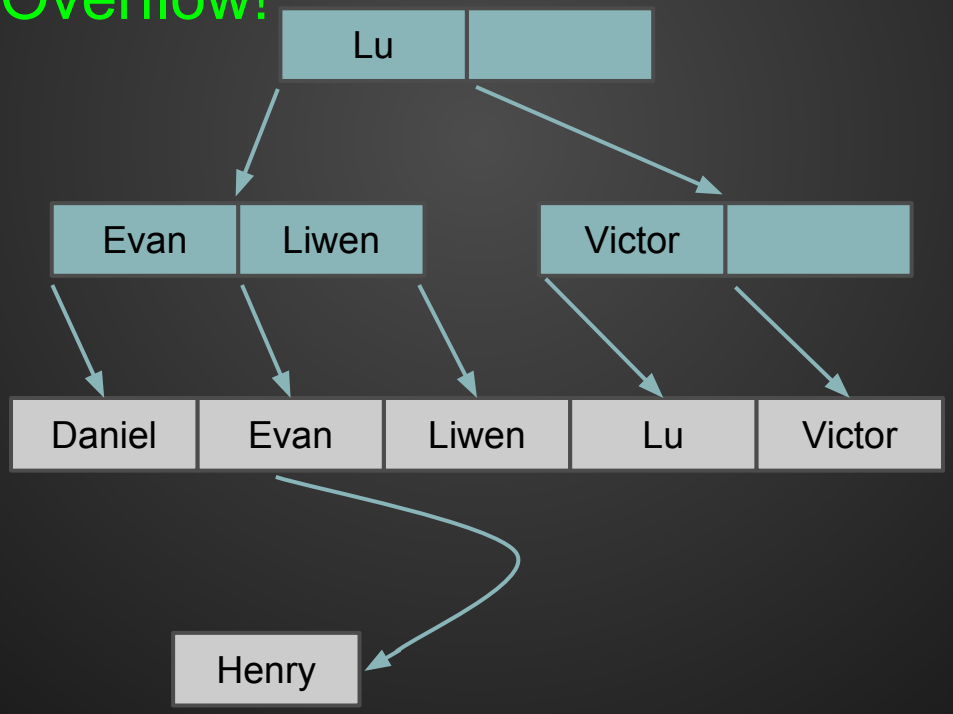
ISAM: Insertion

- Insert “Henry”
  - No Room!



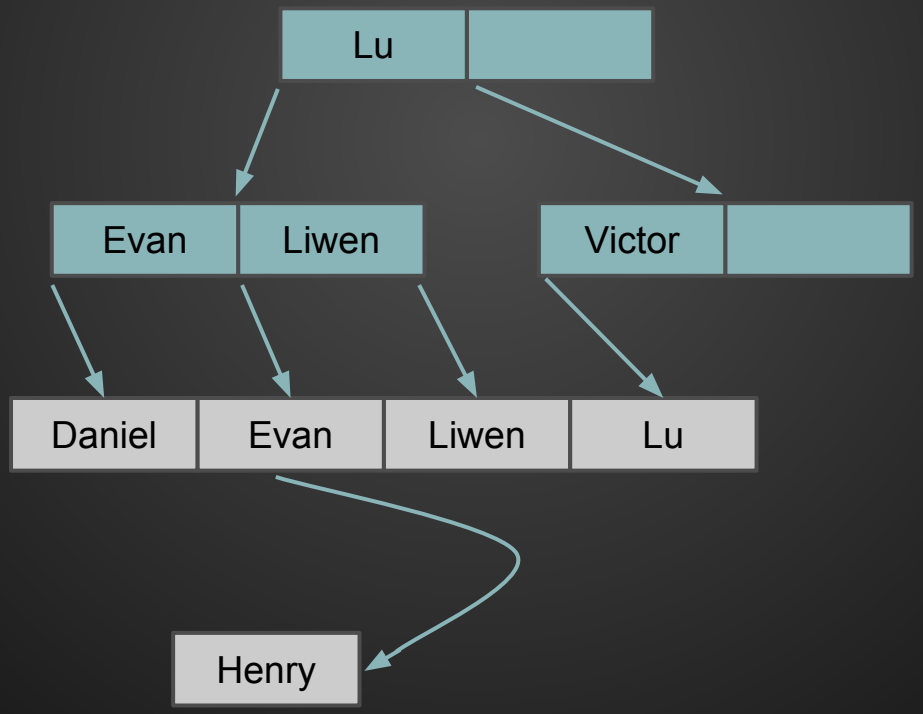
ISAM: Insertion

- Insert “Henry”
  - No Room!
  - So Overflow!

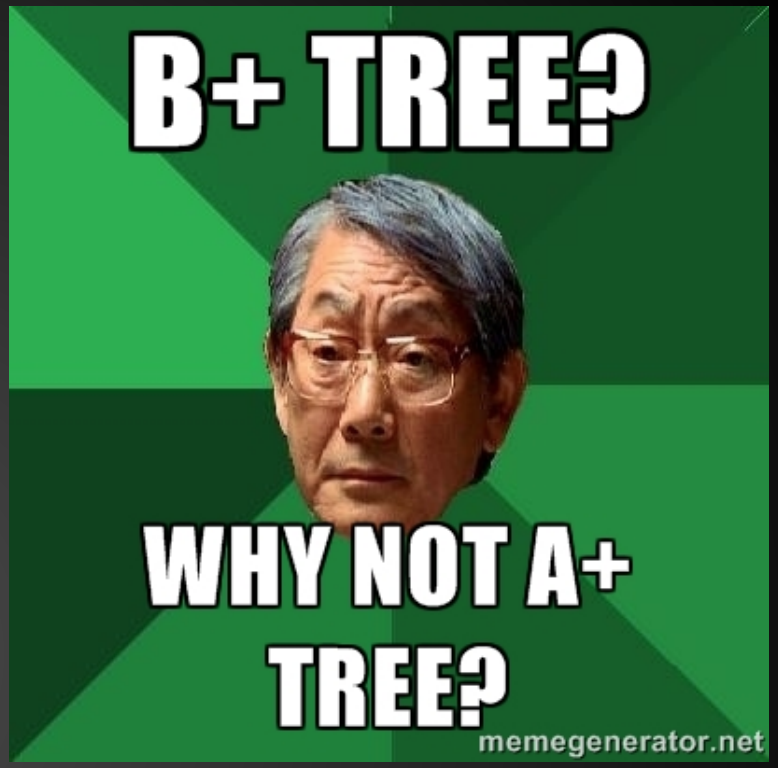


ISAM: Deletion

- Delete “Victor”
  - Index doesn’t change!



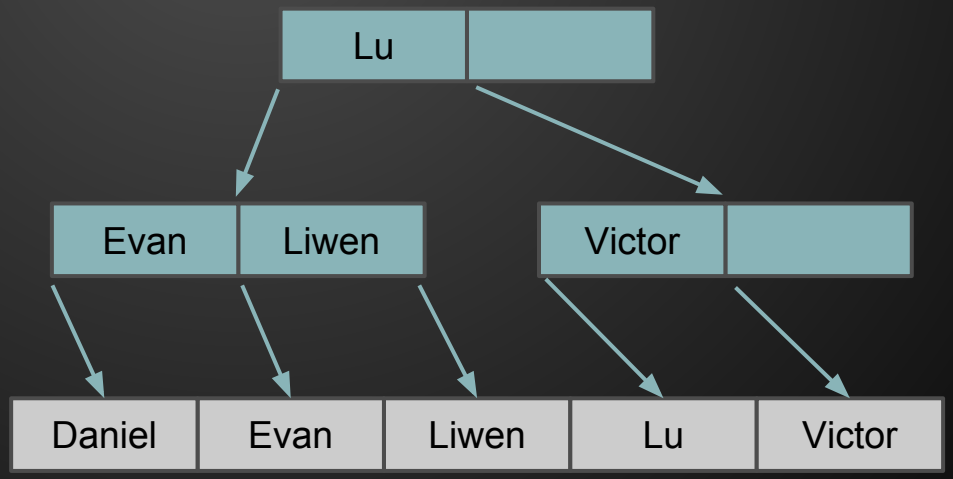
B+ Trees



- ISAM sucks! (why?)
- Idea: Don’t be stupid!
  - Allow the tree to change.
- Key ideas:
  - Maintain minimum 50% occupancy for page.
  - Only leaf pages contain the data/RIDs.

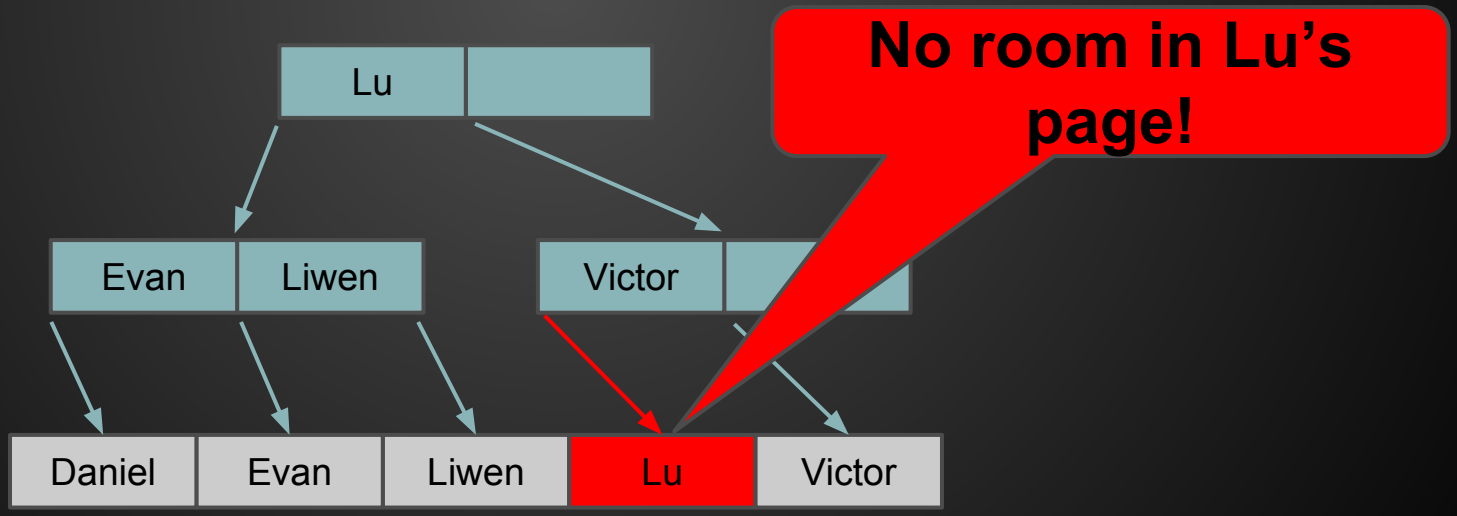
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Mike”



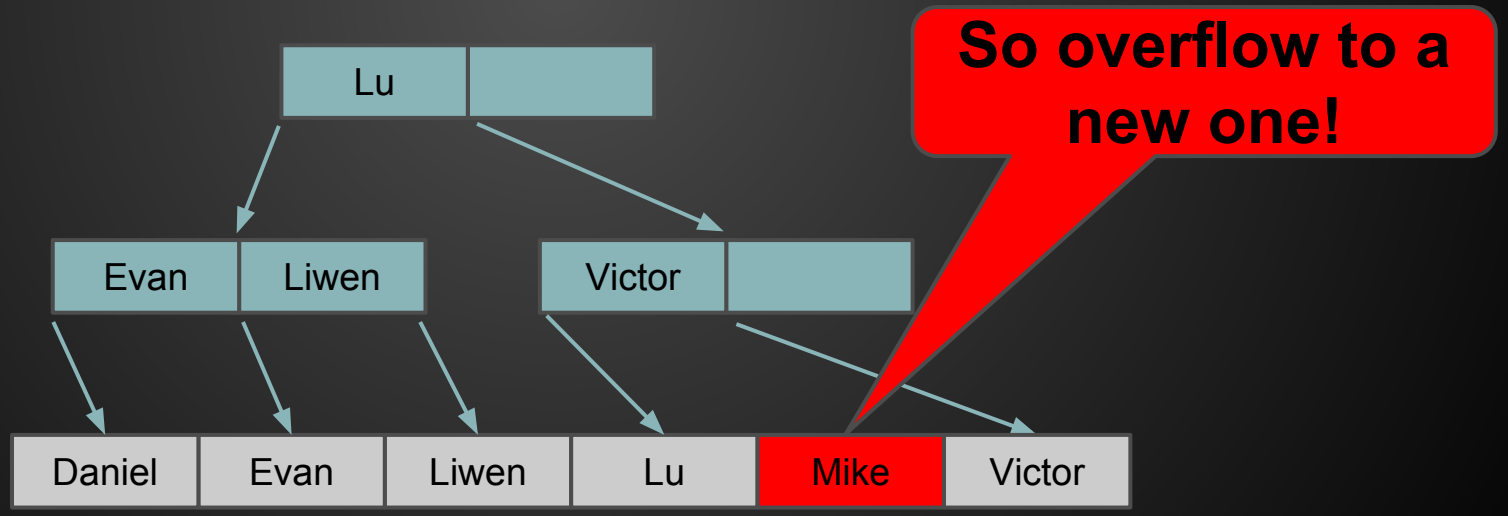
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Mike”



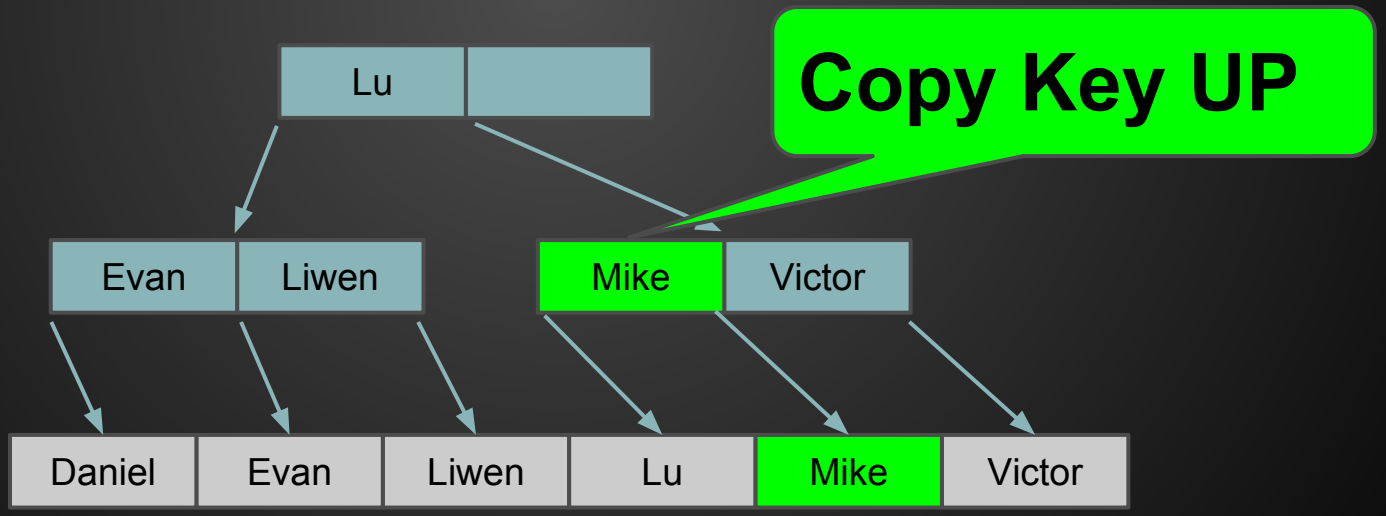
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Mike”



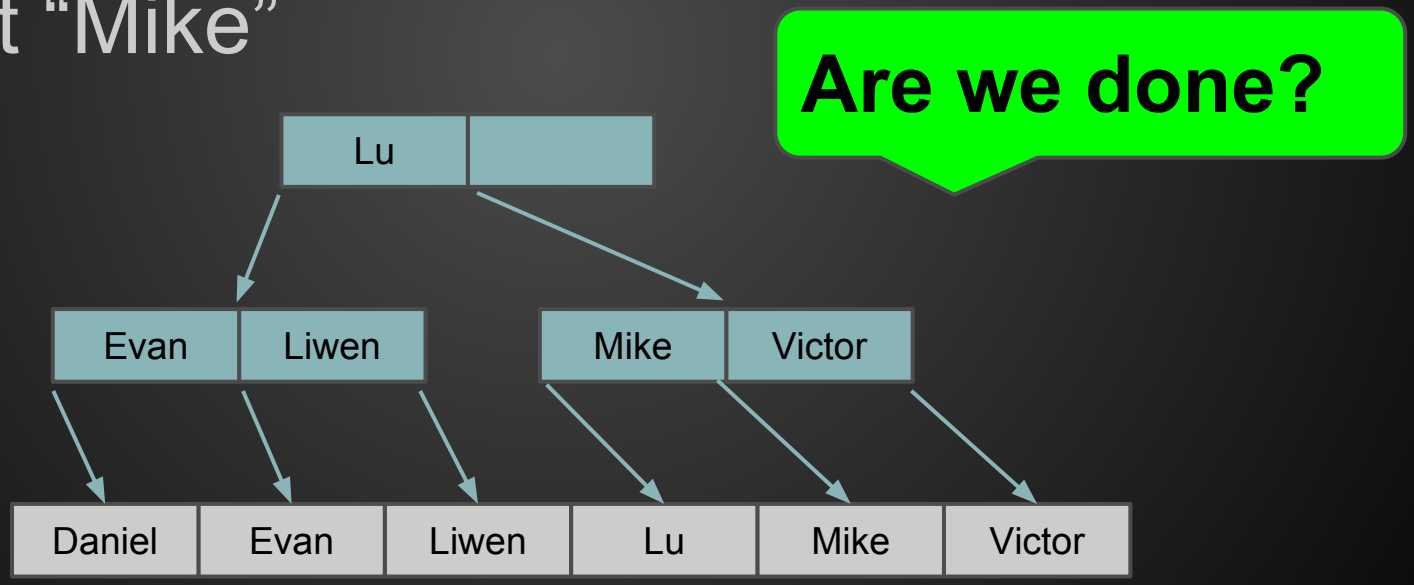
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Mike”



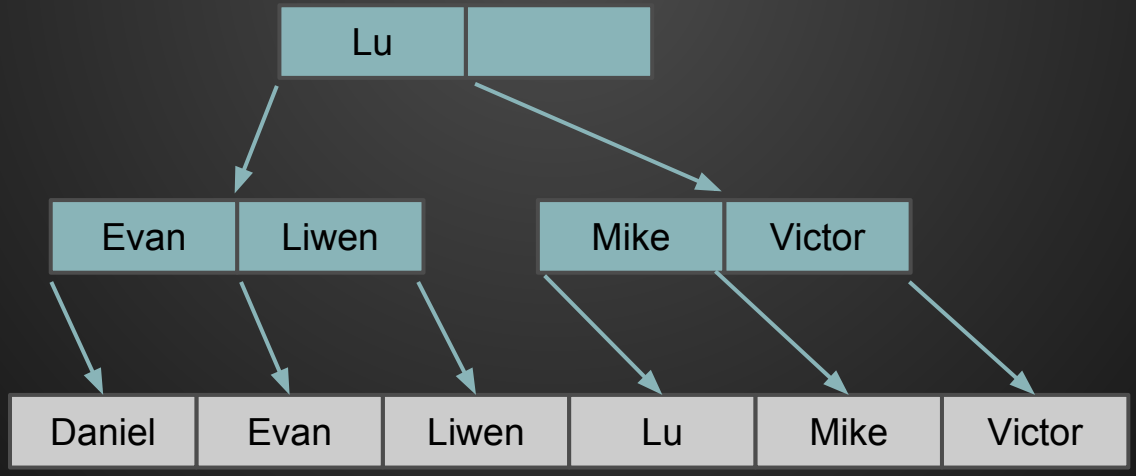
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Mike”



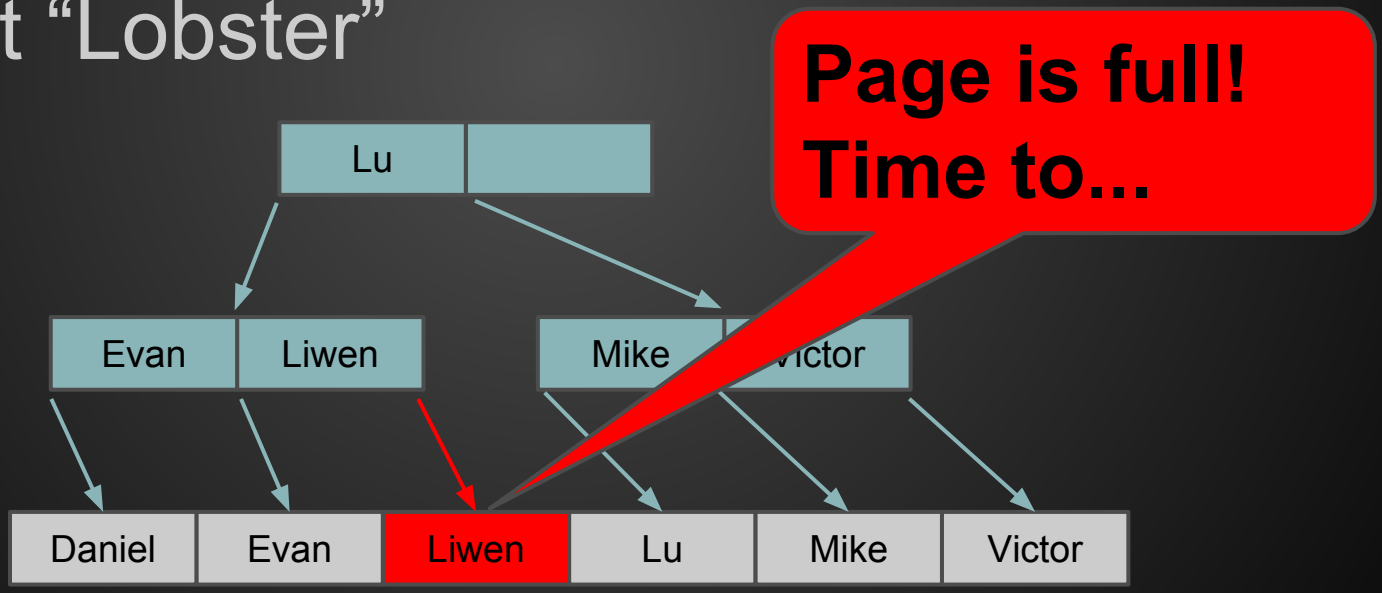
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Lobster”



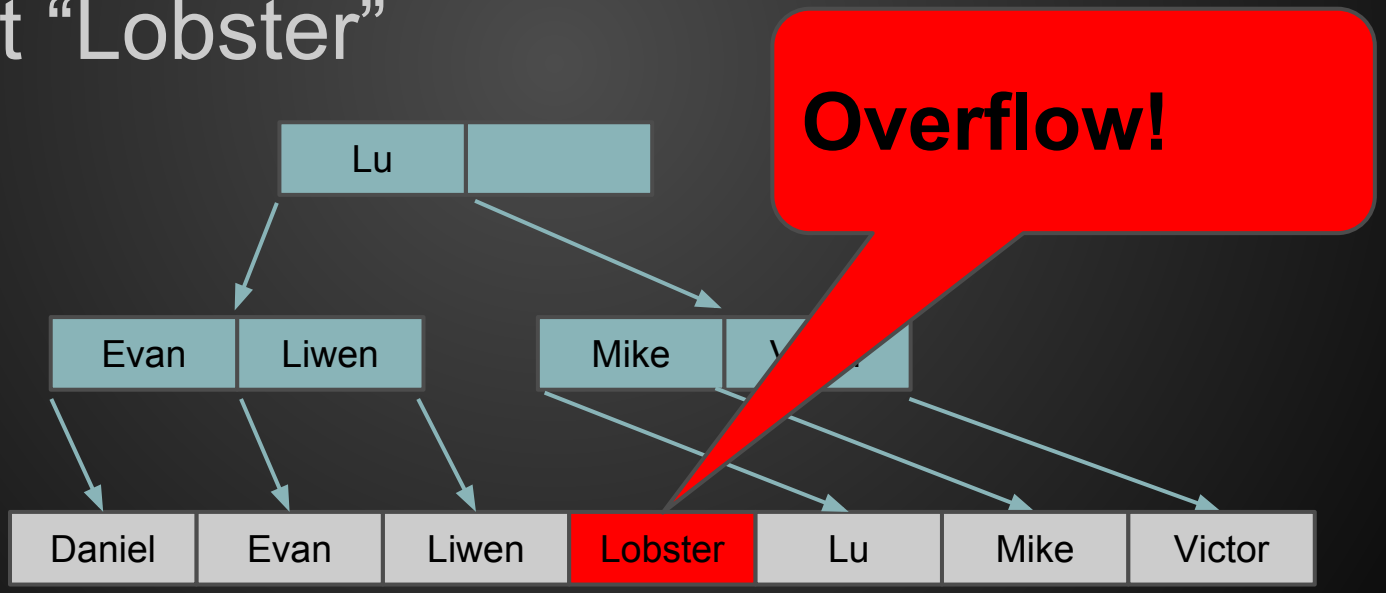
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Lobster”



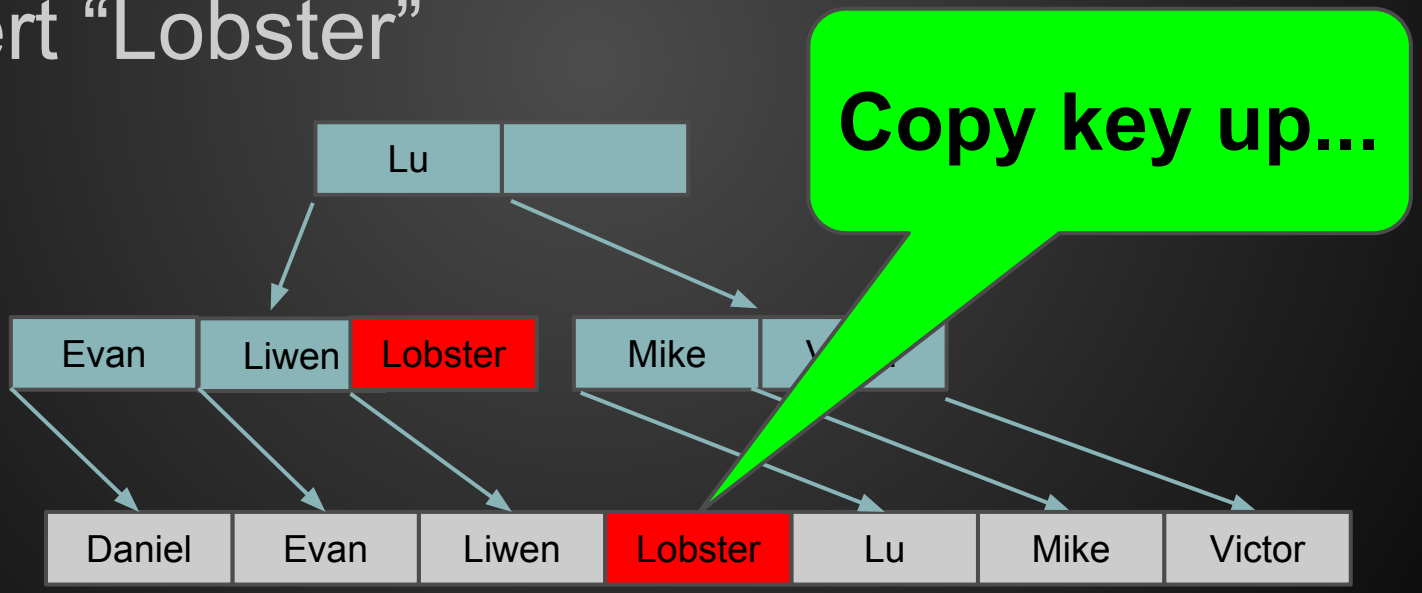
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Lobster”



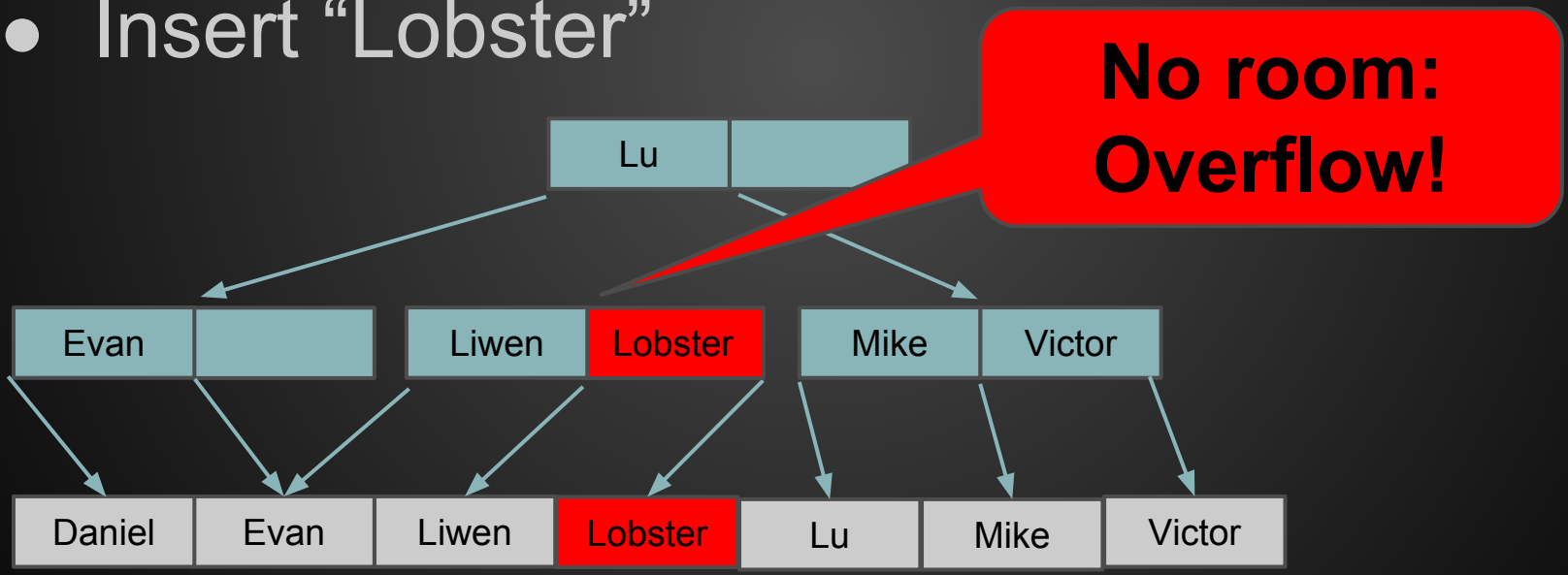
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Lobster”



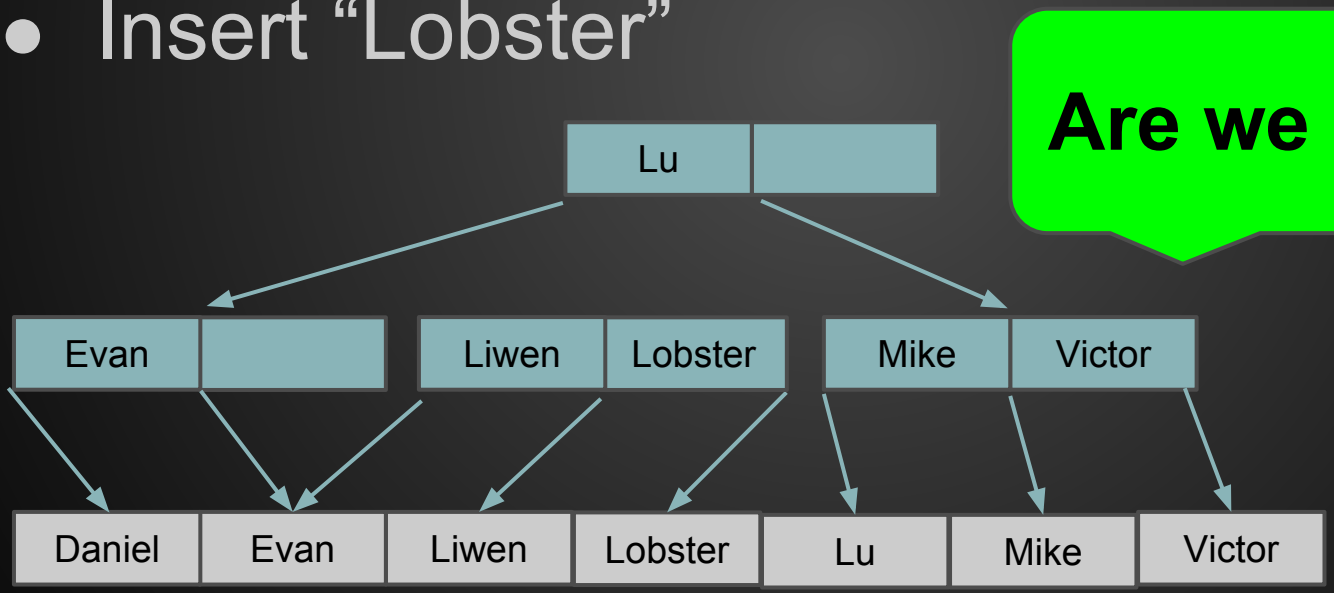
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Lobster”



B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Lobster”

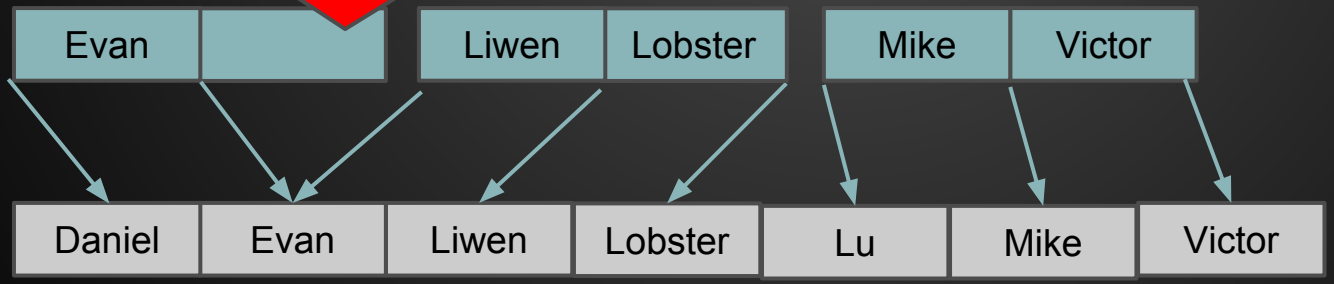


B+ Trees: Insertion

- On insertion overflow:
  - ...up
  - ...p
- ...

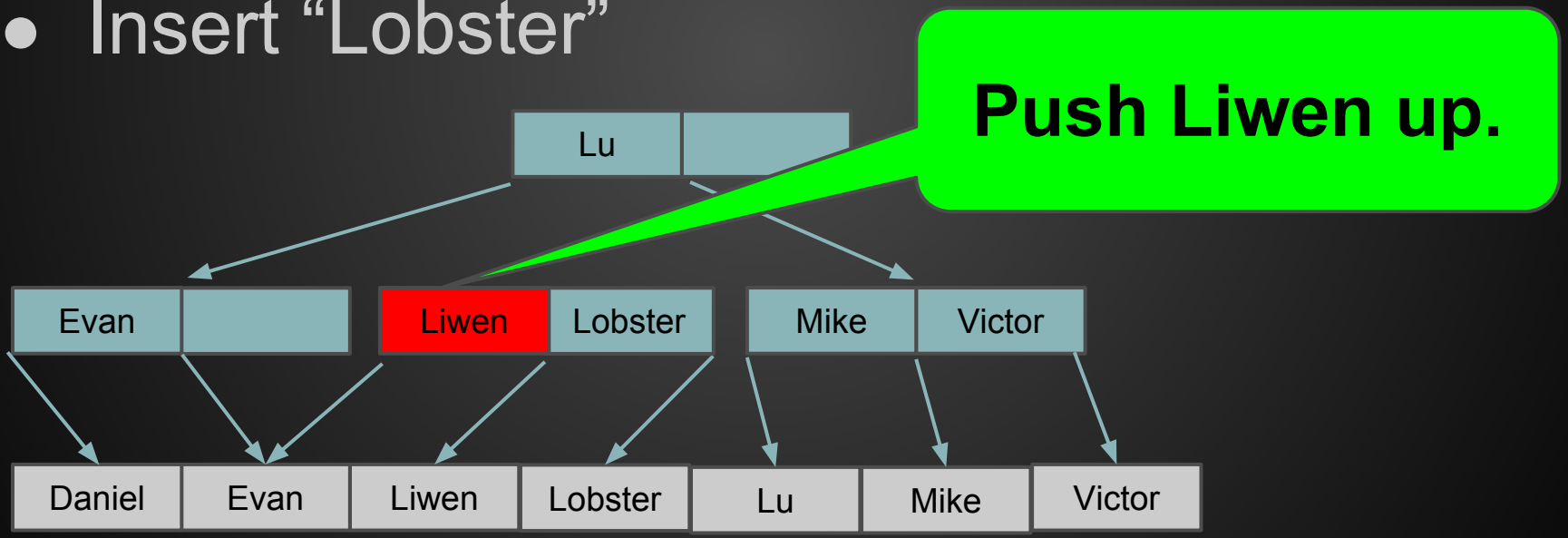
NO!

Are we done?



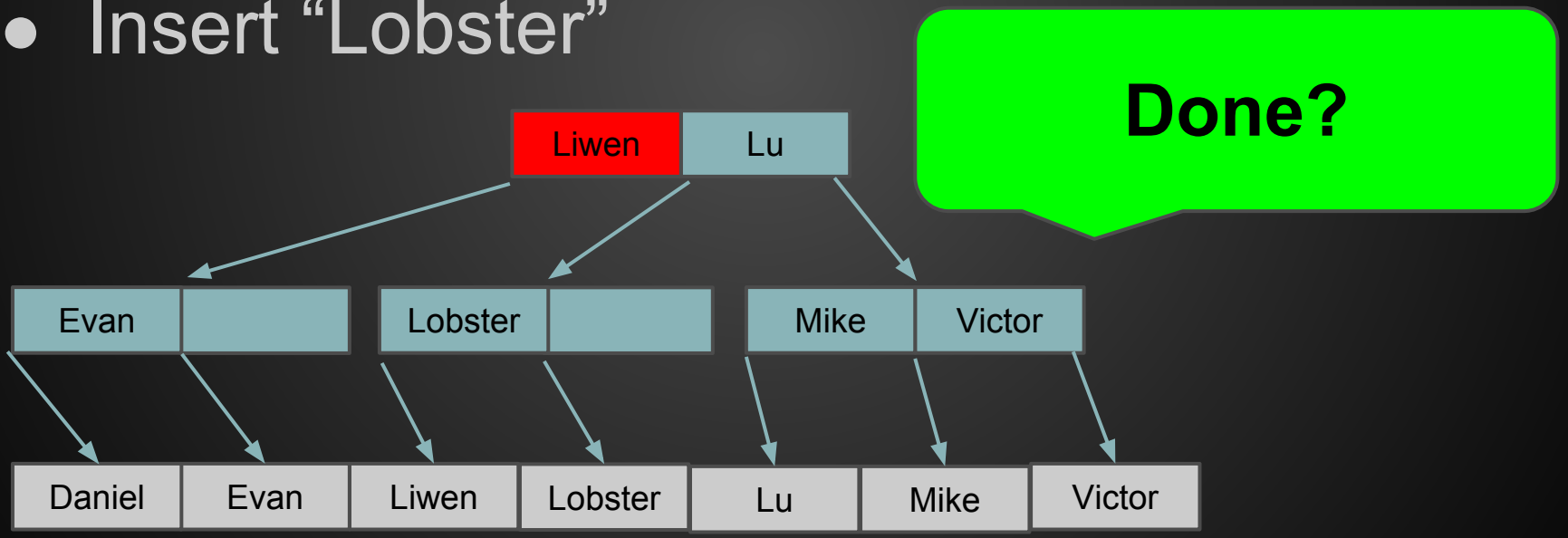
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Lobster”



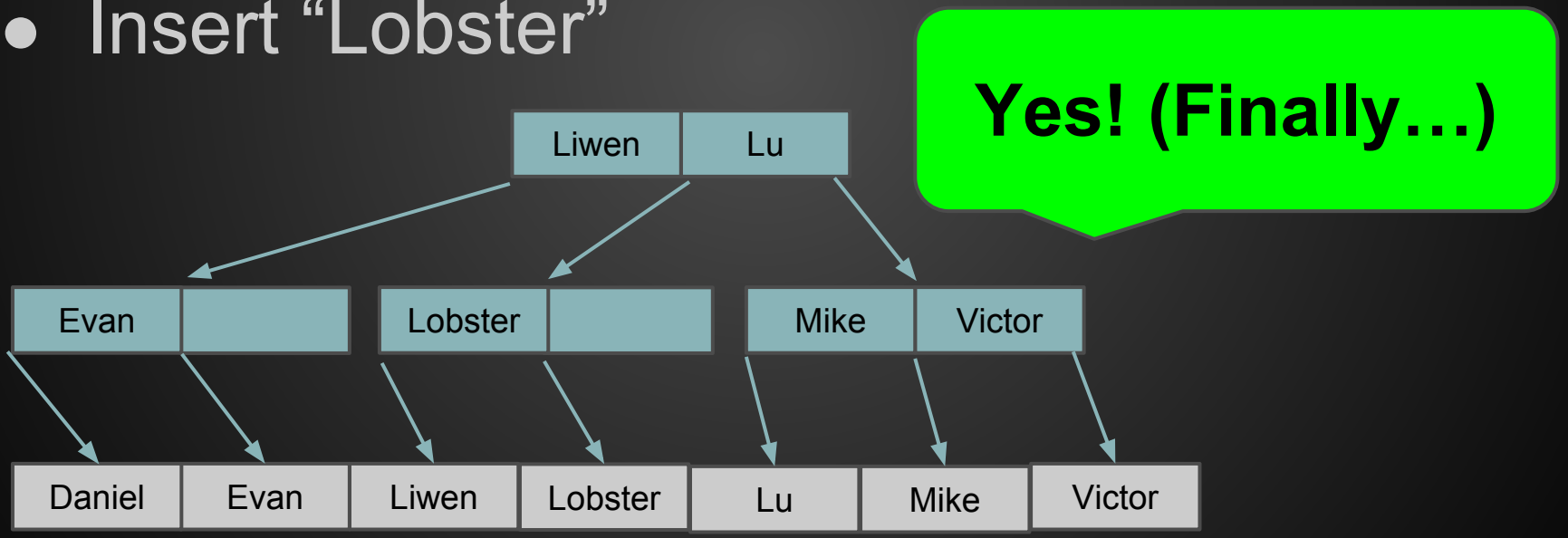
B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Lobster”



B+ Trees: Insertion

- On insertion overflow:
  - For leaf page, **copy** key up
  - For non-leaf, **push** key up
- Insert “Lobster”



Classifying Indexes: Selection Support

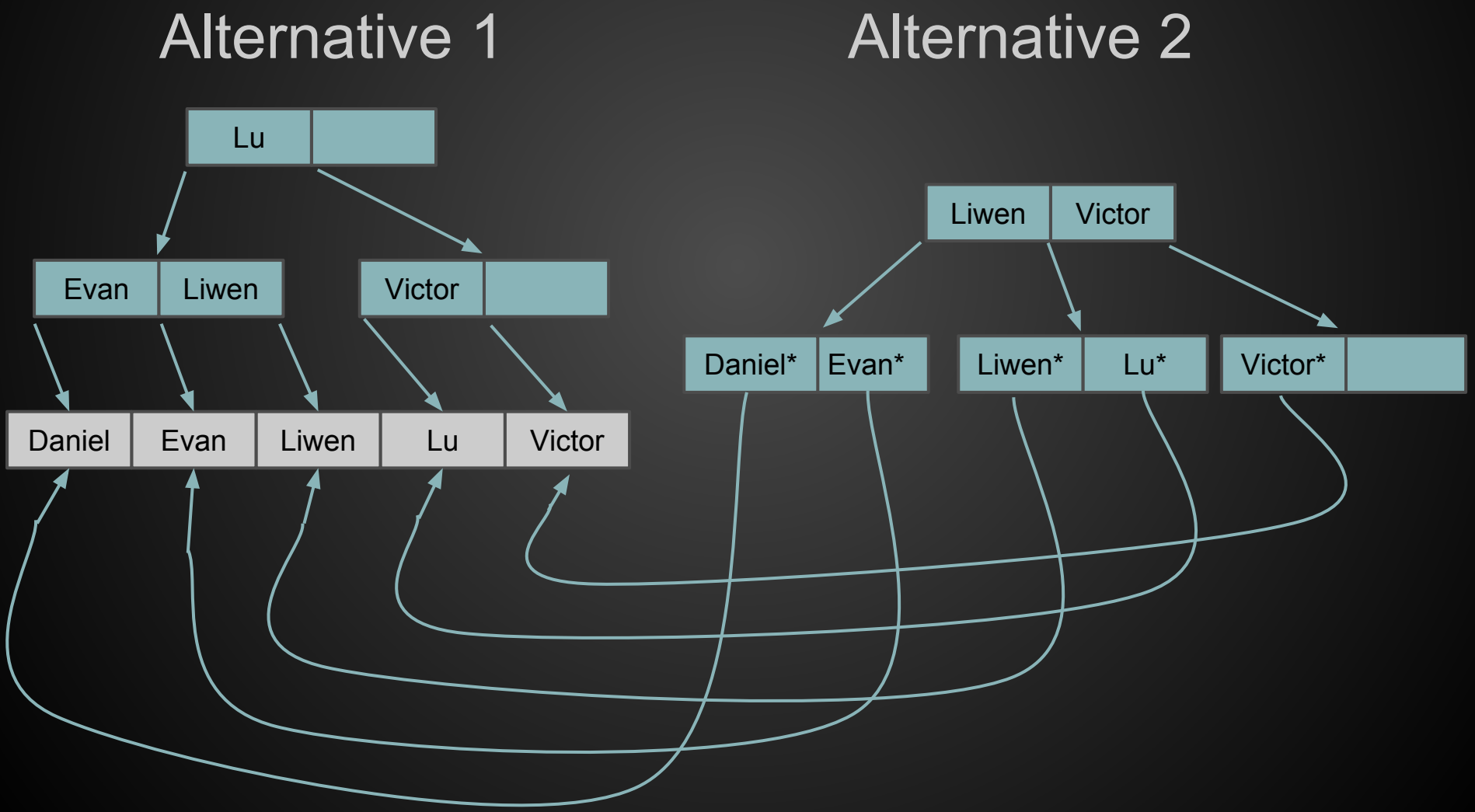
	Good for Equality Selection?	Equality Lookup Speed	Good for Range Selection?	Range Lookup Speed
Heap File	No	0.5 B	No	B
Sorted File (100% Full)	Meh	$\log_2 B$	Meh	$\log_2 B$ + # match pages
Tree Index (67% Full)	Yes	$\log_F 1.5B$	Yes	$\log_F 1.5B$ + # match pages
Hash Index	Yes	2*	No	B

*Ideally. We'll learn more about this on Wednesday.

Classifying Indexes: Storage Alternatives

- 1: “Index-organized file”
  - Data lives at the leaves of the index.
  - Avoid pointer-following I/Os
  - Can only have one of these (why?)
- 2: Key-RID pairs.
  - leaves don’t store data!
  - Follow pointers, but can reorganize easily
  - Could have many of these
- 3: Key-RIDList pairs.
  - Like 2, but more compact (tradeoff: variable length entries)

Classifying Indexes: Storage Alternatives

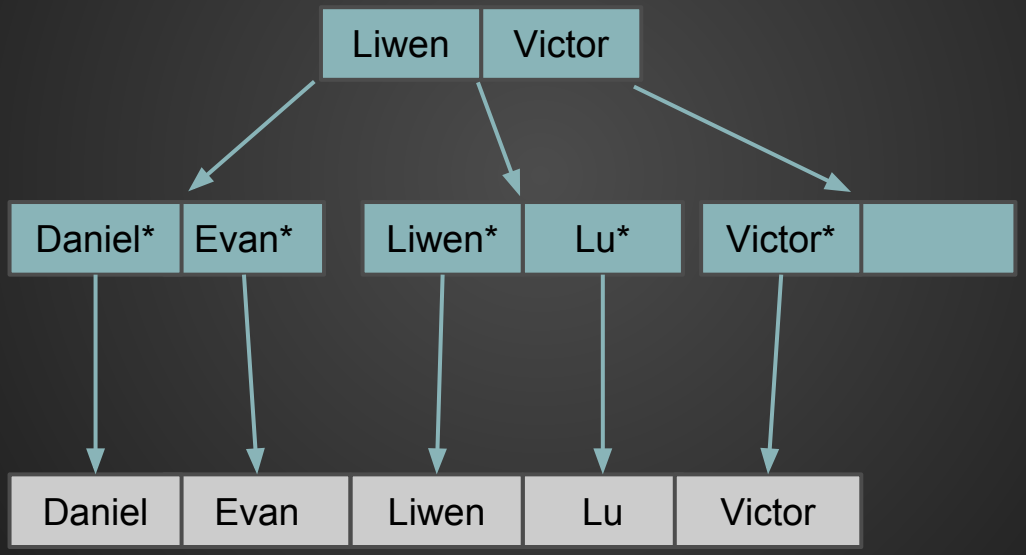


Classifying Indexes: Clustering

- “Clustered”
  - Data is (pretty much) sorted by index search key
  - Only one of these! But fast lookups/range search.
  - Alternative 1 is ALWAYS clustered.
- “Unclustered”
  - Data can be wherever.
  - Following pointers can get expensive!!

Classifying Indexes: Clustering

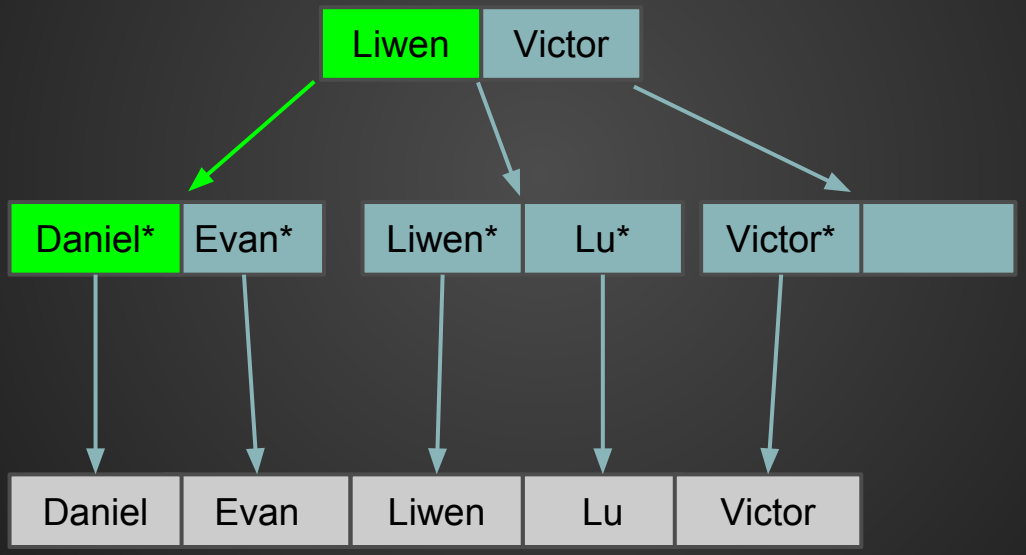
Clustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Clustering

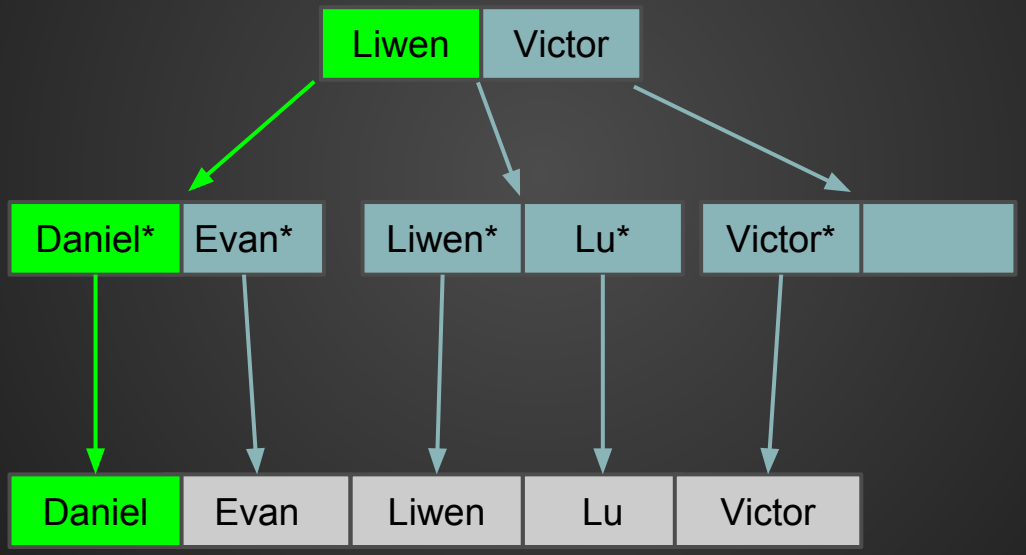
Clustered example



Let’s look up “All GSIs with names between D and L.”

Classifying Indexes: Clustering

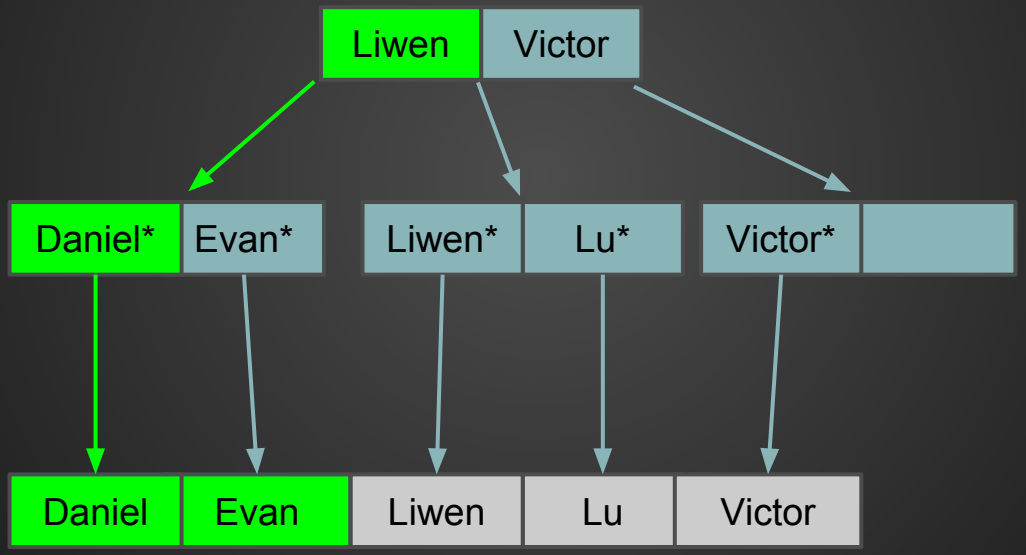
Clustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Clustering

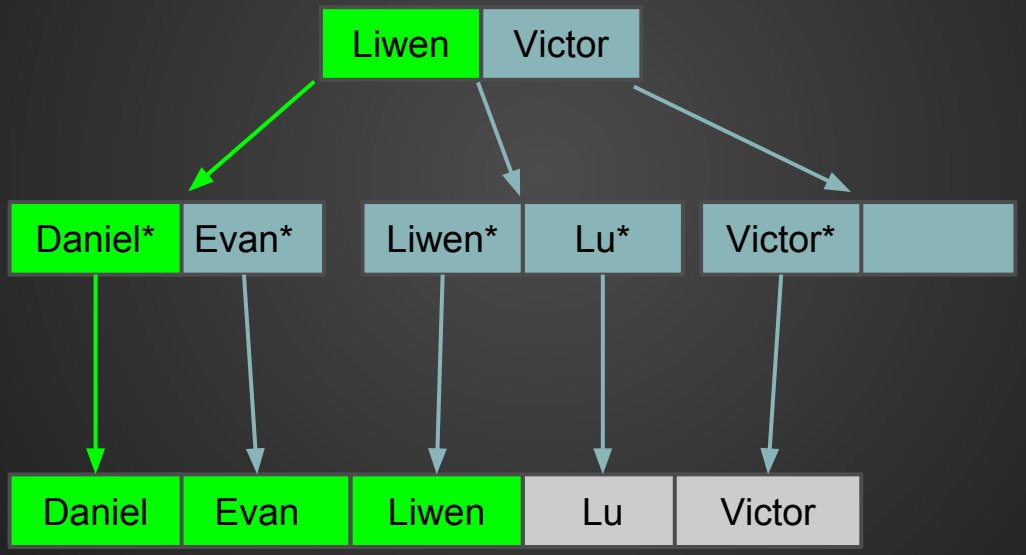
Clustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Clustering

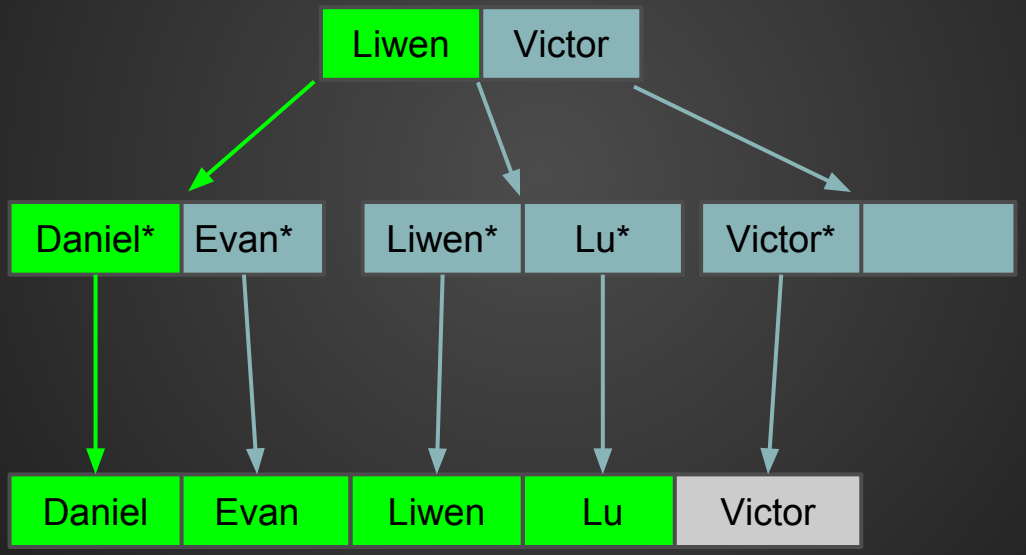
Clustered example



Let’s look up “All GSIs with names between D and L.”

Classifying Indexes: Clustering

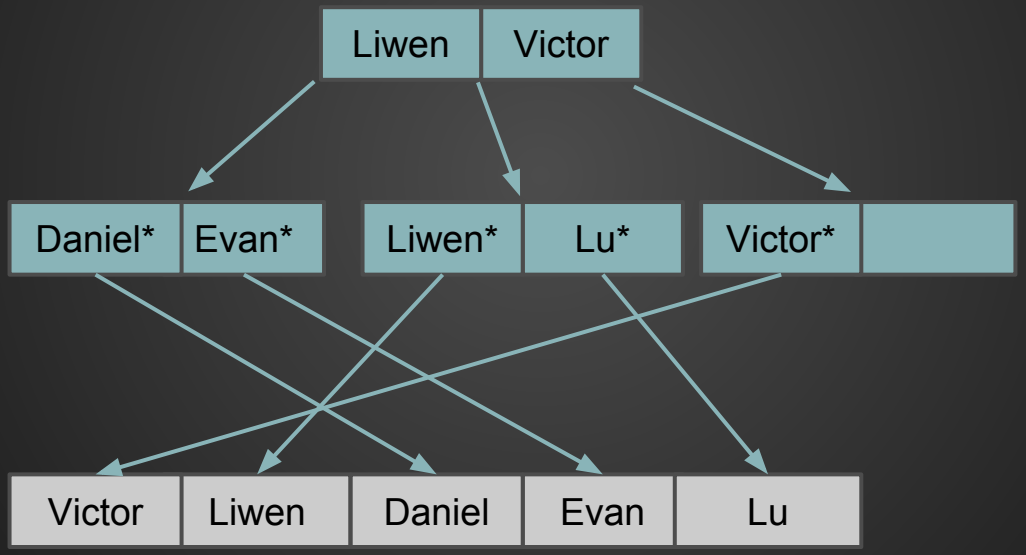
Clustered example



Let’s look up “All GSIs with names between D and L.”

Classifying Indexes: Clustering

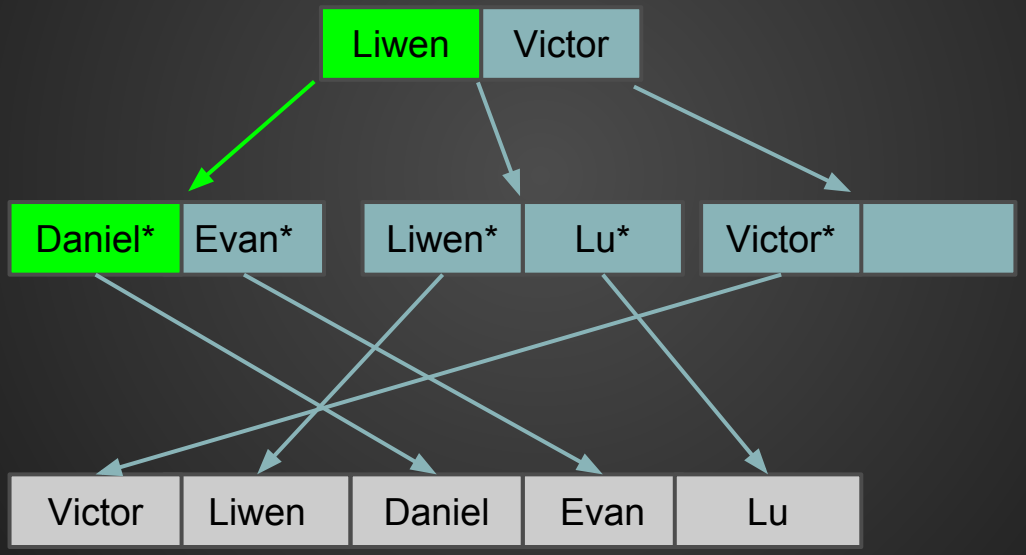
Unclustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Clustering

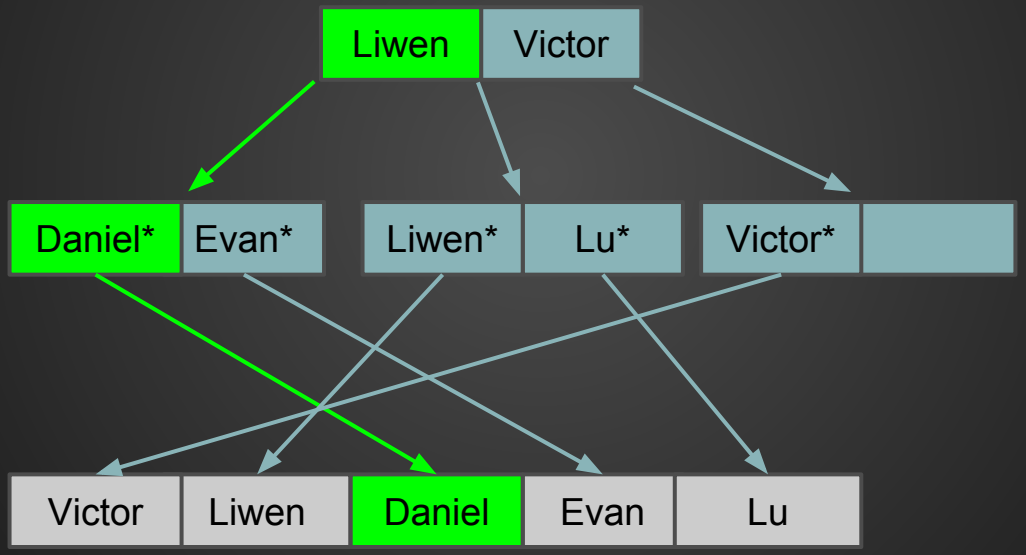
Unclustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Clustering

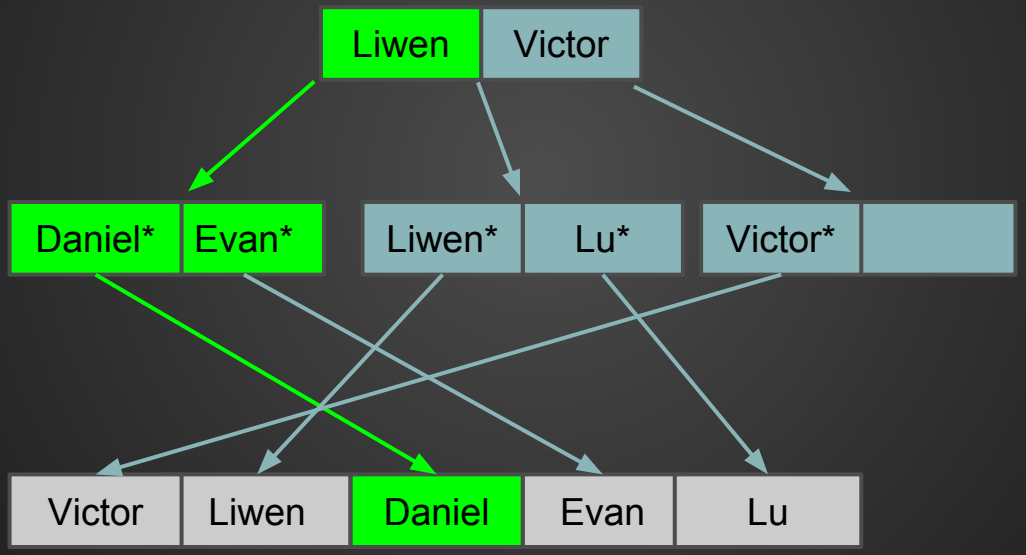
Unclustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Clustering

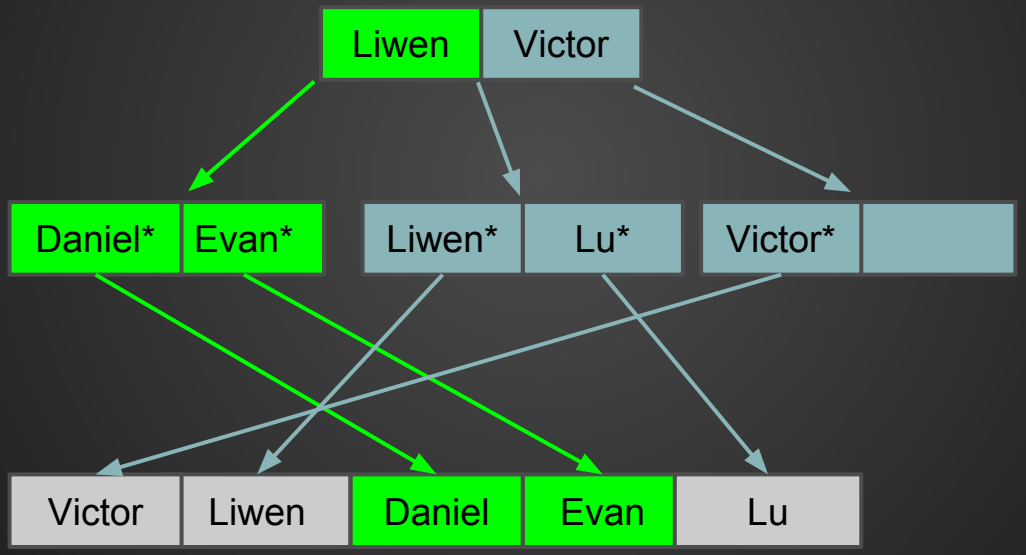
Unclustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Clustering

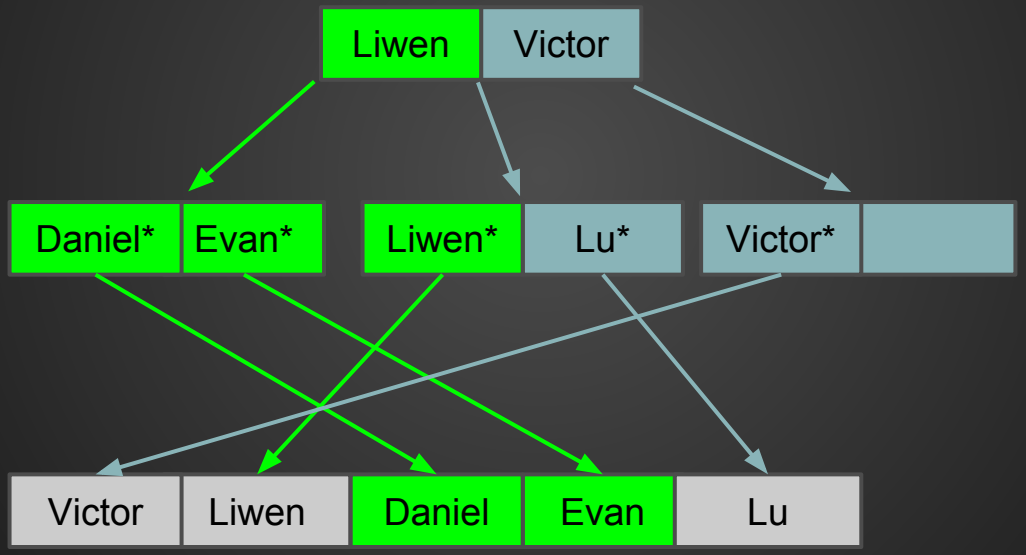
Unclustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Clustering

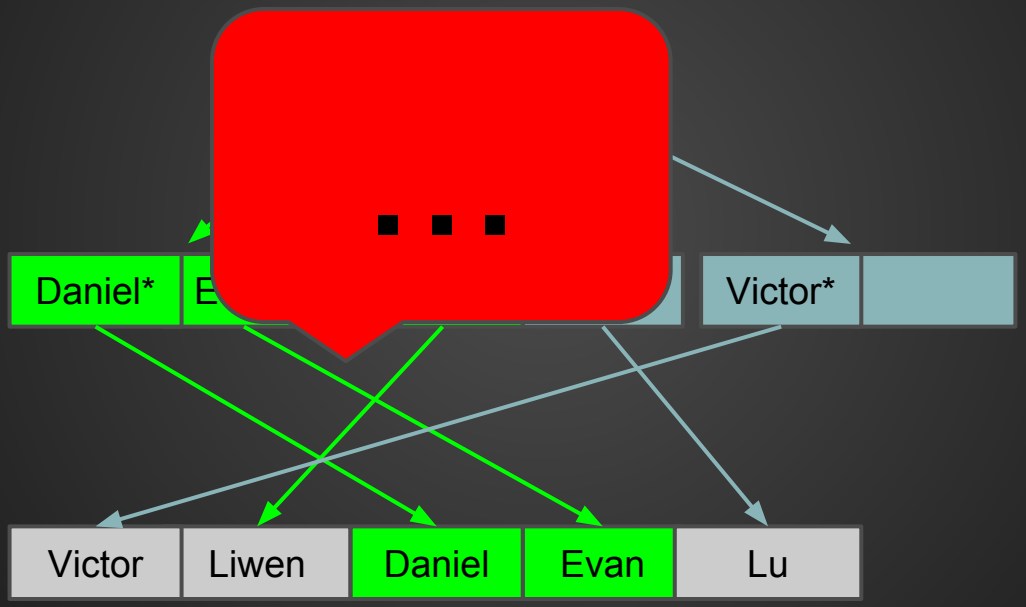
Unclustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Clustering

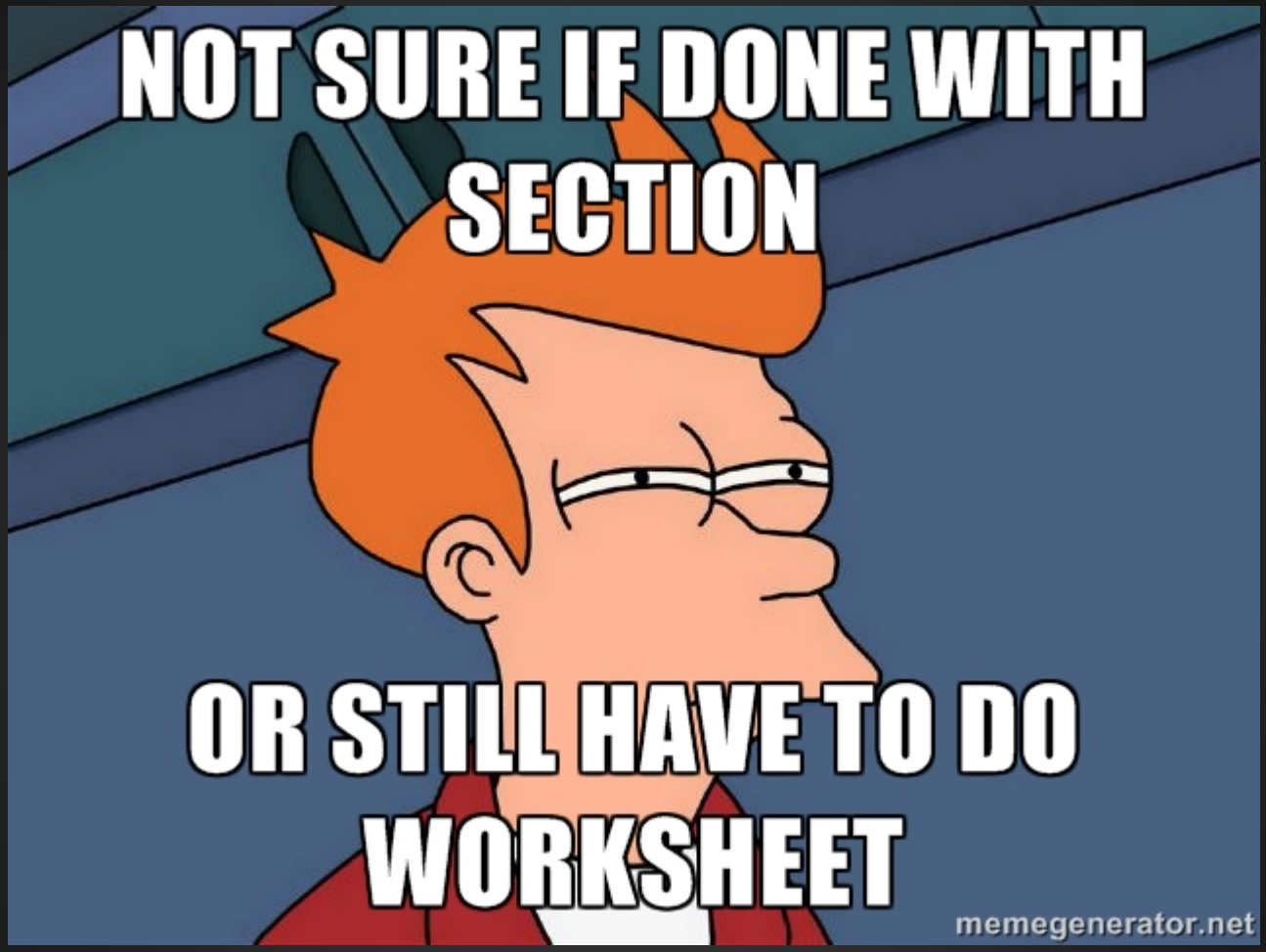
Unclustered example



Let's look up "All GSIs with names between D and L."

Classifying Indexes: Composite Keys

- Composite Key: search by >1 field.
  - Can only search keys in LEXICOGRAPHIC ORDER
  - Example: if we have an index on **<age, name>**, we can search for students **by age** or **by age AND name**, but NOT **by name** or **by age OR name**


NOT SURE IF DONE WITH SECTION
OR STILL HAVE TO DO WORKSHEET
memegenerator.net